中山大學 软件工程学院

SUN YAT-SEN UNIVERSITY SCHOOL OF SOFTWARE ENGINEERING

SSE316：云计算技术
Cloud Computing Technology

陈壮彬

软件工程学院

https://zbchern.github.io/sse316.html

虚拟化技术

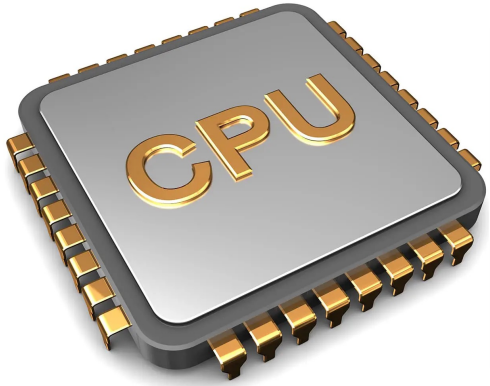
- 虚拟化技术发展的动机
- 虚拟化技术的实现
- 虚拟机迁移
- 虚拟化技术的隐患
- 高效虚拟机监视器Xen

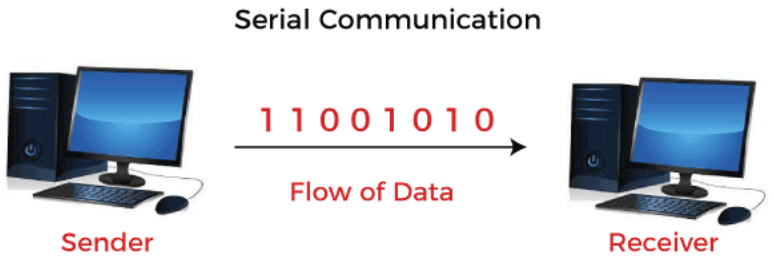
虚拟化技术

- 虚拟化技术发展的动机
- 虚拟化技术的实现
- 虚拟机迁移
- 虚拟化技术的隐患
- 高效虚拟机监视器Xen

虚拟化技术发展的动机

- 描述计算系统的操作需要抽象以下三种基本资源





processors　　memory　　communication links

随着系统规模和用户规模的增长，计算系统的资源管理变得极具挑战性

虚拟化技术发展的动机 (cont'd)

- 资源管理的难点
  - ✓满足高峰需求 —» 超额资源供应
  - ✓硬件和软件的异构性
  - ✓无法避免的机器故障

虚拟化是云计算的基本使能技术，它简化了三种抽象物理资源的管理

虚拟化技术的定义

In computing, virtualization is the act of creating a virtual (rather than actual) version of something at the same abstraction level, including virtual computer hardware platforms, storage devices, and computer network resources.

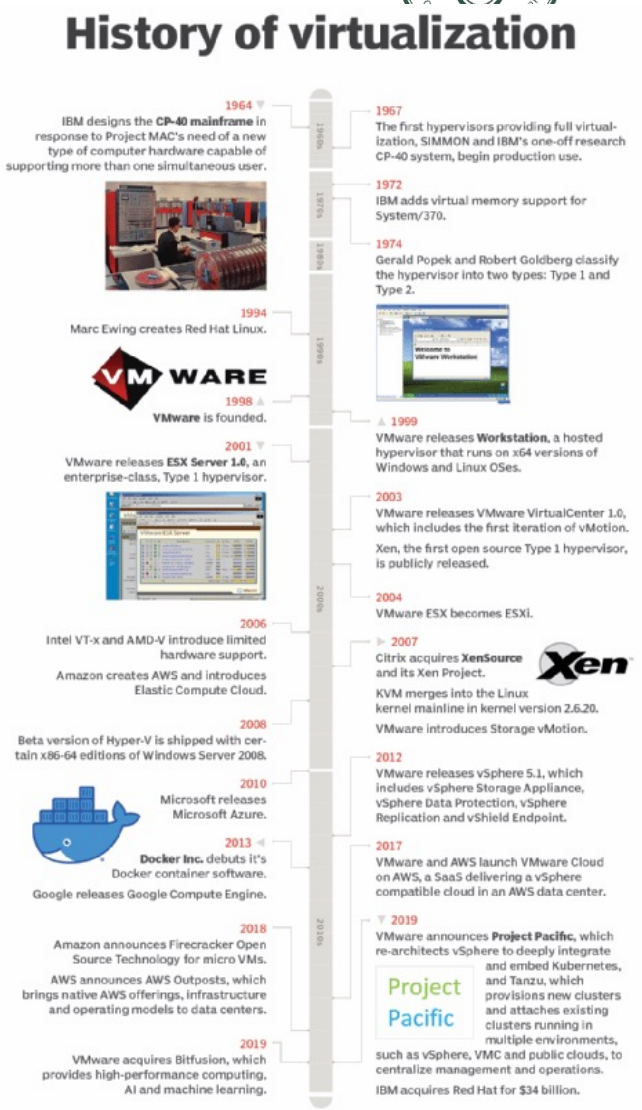
from Wikipedia

虚拟化技术能做什么？

- 在一个操作系统上运行另一个操作系统
- 在一个机器上运行多个（虚拟）机器
- 虚拟机的状态可被保存并迁移到另一个机器上
- 虚拟化允许用户在熟悉的环境中操作
- …

虚拟化技术的发展历史

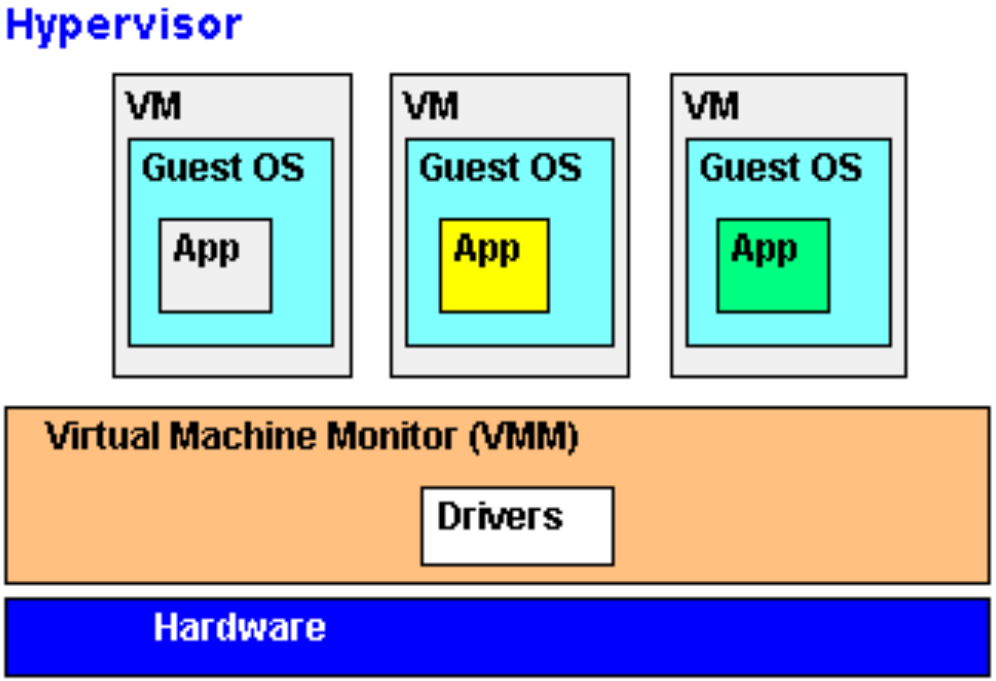
- **1960s:** The first virtualization technologies were developed for mainframe computers, including the IBM CP-40/CMS.
- **1970s:** The concept of virtual machines (VMs) was introduced, allowing multiple operating systems to run on the same physical machine. IBM's VM/370 operating system was one of the first commercial implementations of virtual machines.
- **1990s:** Virtualization technologies were developed for x86-based systems. The first commercial x86 virtualization products were released, including VMware Workstation and Virtual PC.
- **2000s:** Hardware-assisted virtualization technology was introduced. Open-source virtualization platforms, such as Xen and KVM, were also developed during this period.
- **2010s:** Cloud computing and containerization emerged as popular virtualization technologies. Cloud platforms such as Amazon Web Services and Microsoft Azure became dominant players in the cloud computing market, while containerization platforms such as Docker and Kubernetes gained popularity as lightweight alternatives to traditional virtual machines.



https://www.techtarget.com/searchitoperations/feature/The-history-of-virtualization-and-its-mark-on-data-center-management

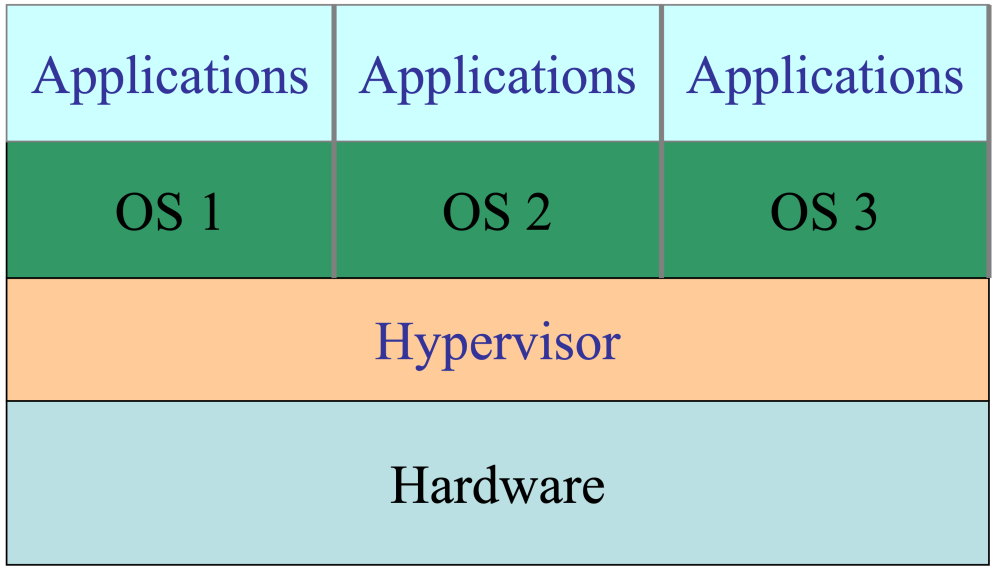
虚拟机监控器

虚拟机监控器（Virtual Machine Monitor, VMM/Hypervisor）将计算机系统的资源划分为一个或多个虚拟机（VM），允许多个操作系统在单个硬件平台上同时运行。



虚拟机监控器 (cont'd)

- 虚拟机监控器的类型



Type-1型：虚拟机直接运行在硬件上，直接控制硬件资源，可理解为一种特殊的操作系统。

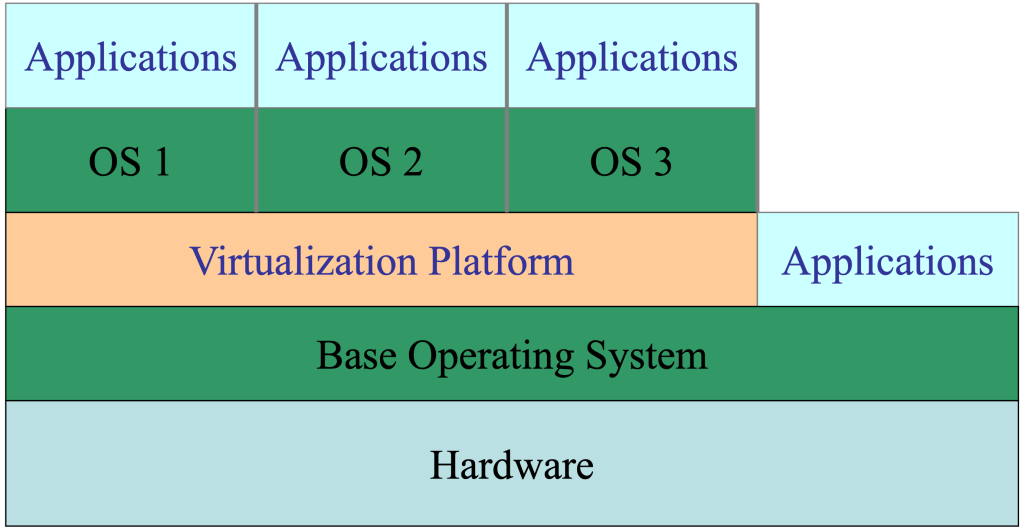
Microsoft Hyper-V

vmware vSphere 7

Xen Server™

虚拟机监控器 (cont'd)

- 虚拟机监控器的类型



Type-2型：依托一个宿主操作系统，复用宿主操作系统中的调度和资源管理等功能。

虚拟机监控器 (cont'd)

- 高效虚拟化的三个条件 (from Popek and Goldberg, 1974)

  ✓在虚拟机管理程序下运行的程序应该与直接在等价计算机上运行时所体现的行为基本相同

  ✓虚拟机管理程序应完全控制虚拟化资源

  ✓机器指令中具有统计意义的部分（statistically significant fraction）必须在没有虚拟机管理程序的干预下执行

虚拟化技术

- 虚拟化技术发展的动机
- 虚拟化技术的实现
- 虚拟机迁移
- 虚拟化技术的隐患
- 高效虚拟机监视器Xen

虚拟化技术的底层实现

- VirtualBox是否仅仅是一个用户程序？

No

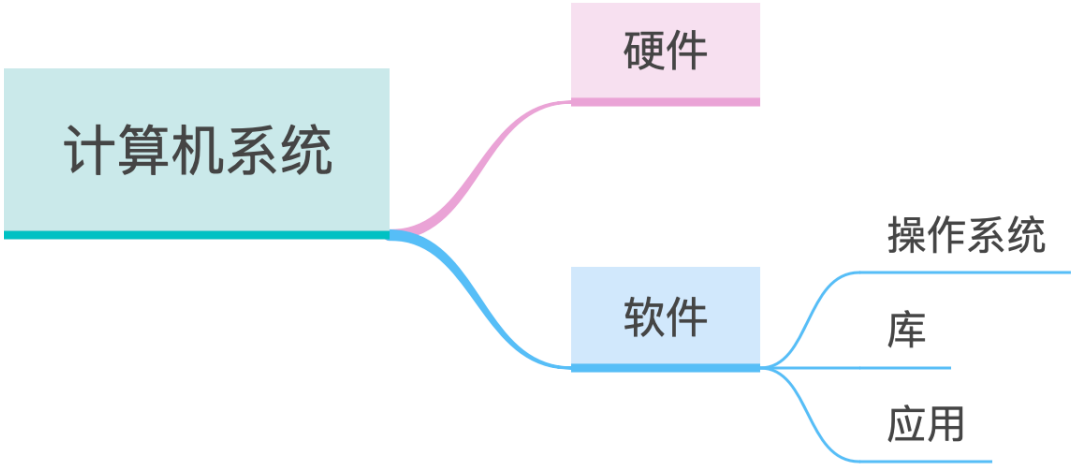
用户程序仅能以非特权的形式运行

```
instructions
    $ cat foo.c
    main()
    {
        __asm__("movl    %eax, %cr3"::);
    }
    $ gcc foo.c
    $ ./a.out
    Segmentation fault (core dumped)
```

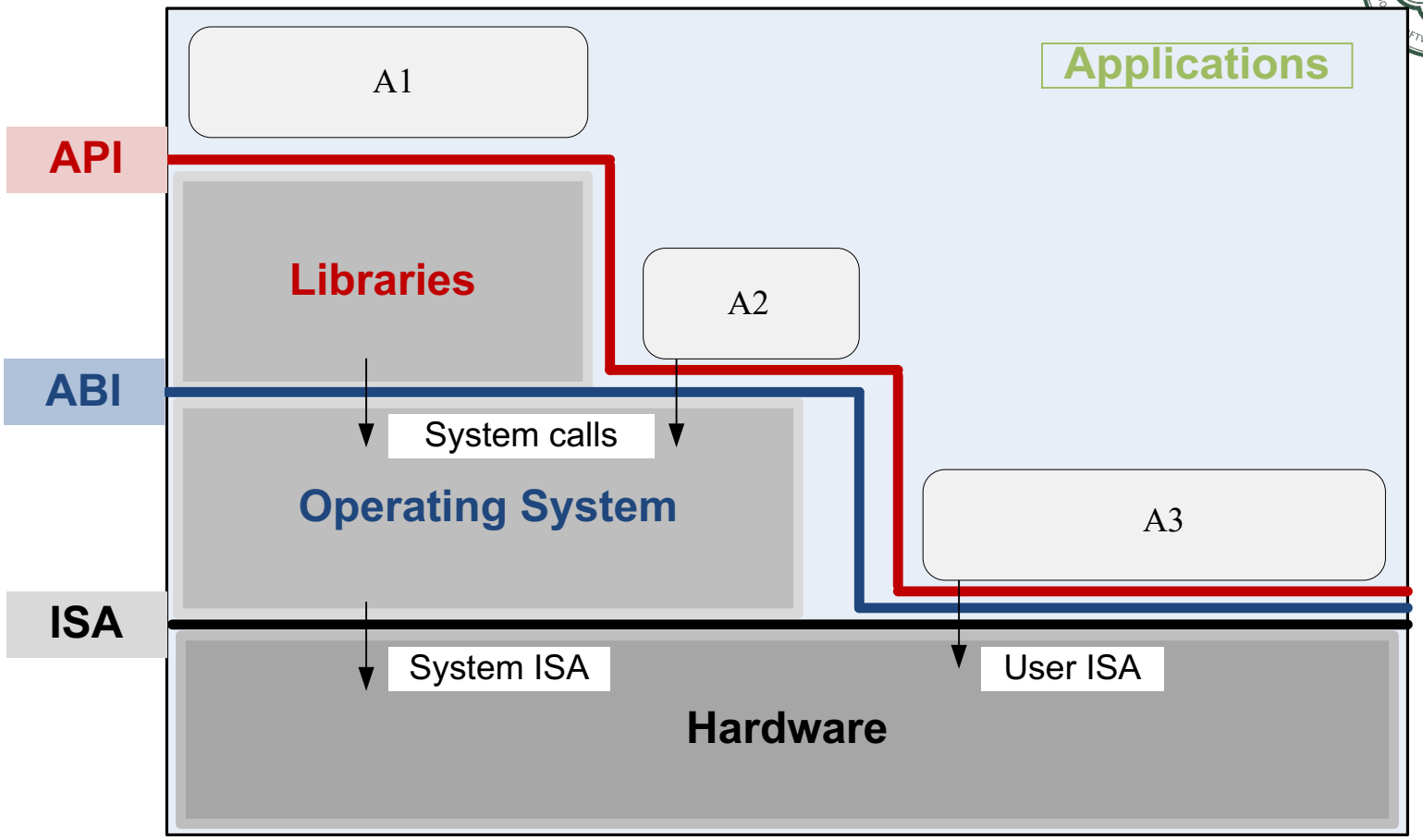
背景回顾 – 分层

- 分层 - 管理系统复杂性的常用方法
  - ✓简化子系统的描述；对子系统间的接口进行抽象
  - ✓最小化复杂系统各子系统之间的相互作用
  - ✓通过分层，我们能够独立地设计、实施和修改各个子系统

- 计算机系统中的分层



背景回顾 – 分层 (cont'd)



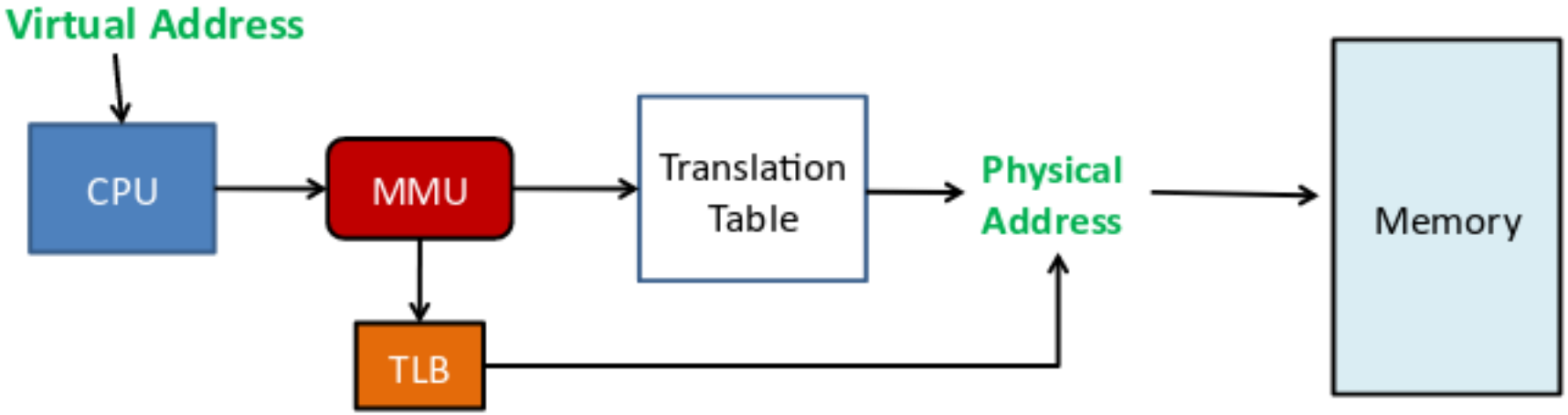
Application Programming Interface (API), Application Binary Interface (ABI), and Instruction Set Architecture (ISA). An application uses library functions (A1), makes system calls (A2), and executes machine instructions (A3).

背景回顾 – 分层 (cont'd)

- Instruction Set Architecture (ISA)
  - ✓软硬件之间的接口，定义处理器可执行的指令集以及其格式
  - ✓用户ISA（用户态和内核态调用），系统ISA（内核态调用）
- Application Binary Interface (ABI)
  - ✓ 定义应用程序与操作系统或运行时环境之间的低级接口
  - ✓包含用户ISA和系统调用
- Application Program Interface (API)
  - 一组编程接口或协议，用于构建软件应用程序
  - 包含库的接口和用户ISA

背景回顾 – 虚拟和物理地址

- 虚拟地址和物理地址转换
  - ✓每个进程都有其独立的虚拟地址空间，经过地址转换访问物理地址
  - ✓不能访问其他进程/操作系统的地址空间
  - ✓TLB (Translation Lookaside Buffer，旁路转换缓冲)：缓存最近使用的虚拟地址映射至物理地址的页表项

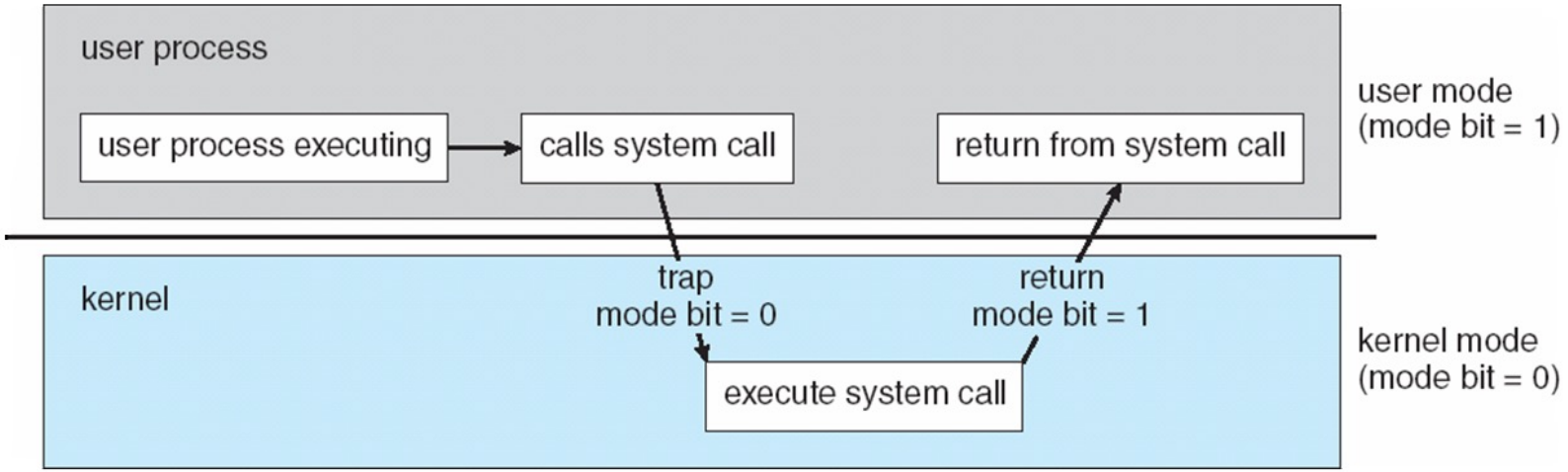


背景回顾 – 用户和内核模式

- 用户模式/内核模式
  - ✓内核模式 (kernel mode)： 操作系统对计算机的硬件和软件资源拥有完全的控制权，并可以执行任何系统级任务
  - ✓用户模式 (user mode) ： 用户级应用程序和进程运行，并且它们对计算机的硬件和软件资源的访问受到限制

enter on interrupt/page fault/system call

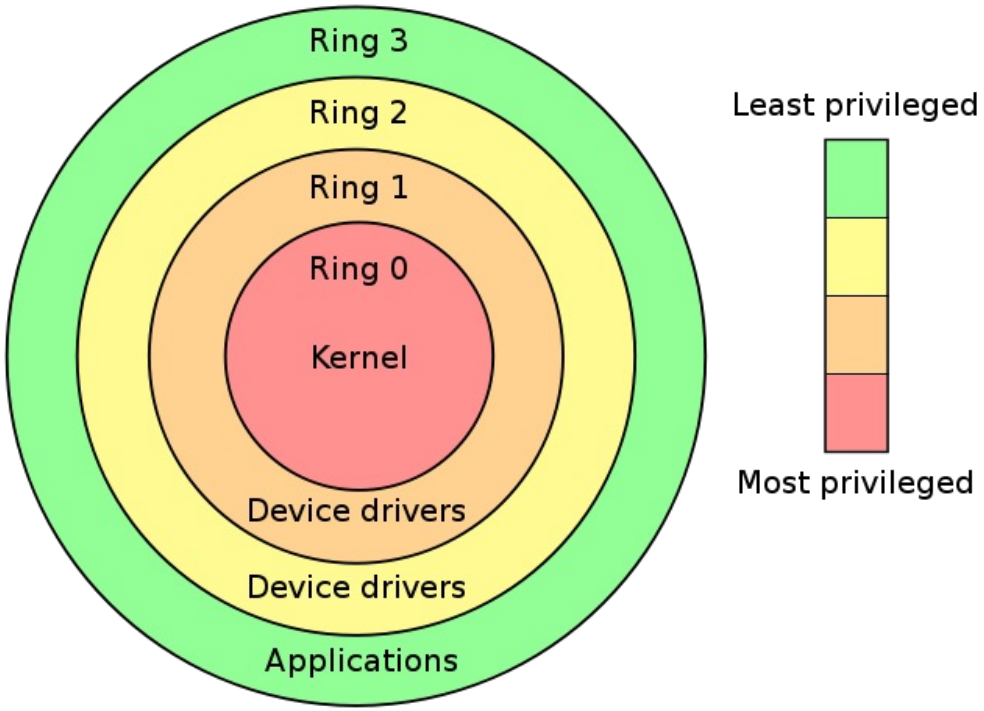
exit back to user program



do protected stuff

背景回顾 – 用户和内核模式 (cont'd)

- 用户模式/内核模式
  - x86架构提供4个保护环，即ring0, 1, 2, 3
  - 通常只用ring 0 (内核模式)和ring 3 (用户模式)



如何在一个OS上运行另一个OS？

如何在一个OS上运行另一个OS？

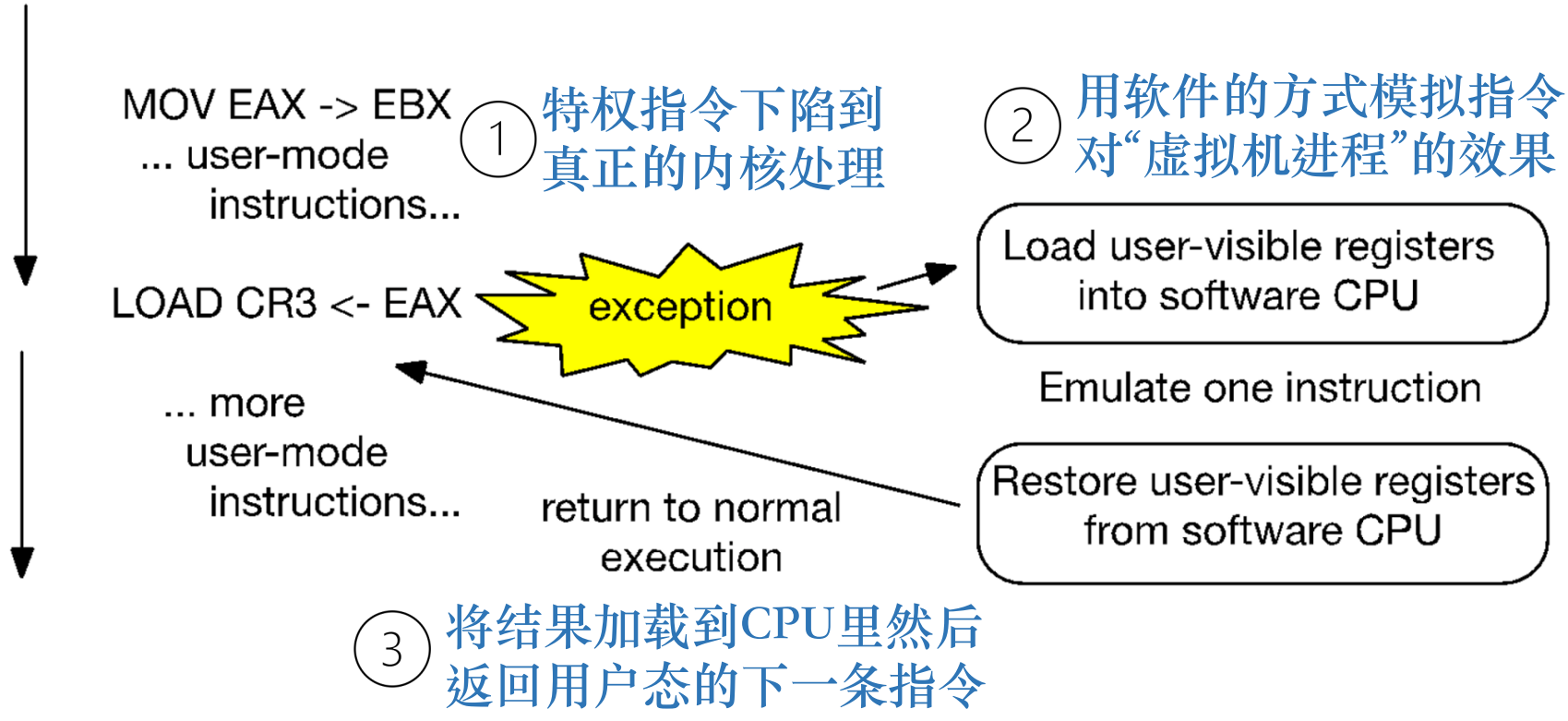
- 模拟每个指令

```
char memory[EMULATED_MEM_SIZE];
int R1, R2, R3, ...;
int PC, SP, CR1, CR2, CR3, ...;
bool S; /* supervisor mode */

while (true):
    instr = memory[PC];
    switch (instr):
      case "MOV R1 -> R2":
        R2 = R1; break;
      case "JMP":
        PC = … ; break;
      case "STORE Rx, <addr>":
            <paddr> = TLB[<addr>]
        if <paddr> is real memory:
          memory[paddr] = Rx
        else
          simulate_IO_access(paddr, Rx)
      .... Etc. (for ~1000 more instructions)
```

如何在一个OS上运行另一个OS？

- 模拟每个指令（cont'd）
  - ✓以用户程序的方式模式所有指令
  - ✓特权指令通过下陷-模拟（trap and emulate）的方法处理



如何在一个OS上运行另一个OS？

- 模拟每个指令（cont'd）
  - ✓此方法是否可行？
  - ✓No
  - ✓在1969年IBM machines上可工作，但是x86和ARM是不可虚拟化的架构
  - ✓x86架构下，用户程序运行在Ring 3，OS在Ring 0

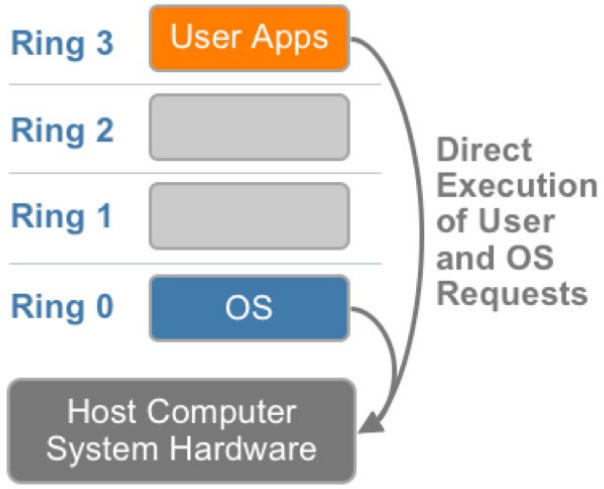


Figure 4 – x86 privilege level architecture without virtualization

如何在一个OS上运行另一个OS？

- 模拟每个指令（cont'd）

特权指令
privileged instructions

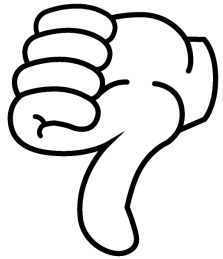
在用户态执行时会触发下陷的指令，包括主动触发下陷的指令和不允许在用户态执行的指令。

敏感指令
sensitive instructions

管理系统物理资源或者更改CPU状态的指令。x86有17条敏感指令，在用户态执行时不会出发下陷。

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。

可虚拟化架构



不满足可虚拟化架构特征的即为不可虚拟化架构。

不可虚拟化架构

在x86架构下实现虚拟化的技术

1. 二进制翻译实现全虚拟化

2. 半虚拟化技术

3. 硬件虚拟化技术

方案一：二进制翻译

- 全虚拟化（full virtualization）
  - ✓无须修改客户OS
  - ✓客户OS可以在VMM下运行，就像它直接在硬件平台上运行一样，每个VM都运行实际硬件的精确副本

- 二进制翻译（binary translation）
  - ✓二进制翻译扫描客户OS及其运行的应用程序的二进制码，用安全代码替换敏感但非特权的指令，以模拟原始指令

方案一：二进制翻译 (cont'd)

- 二进制翻译
  - ✓用户级指令直接运行
  - ✓当客户OS尝试下陷到内核，运行内核代码并缓存结果
- 代表软件
  - ✓VMware
  - ✓Microsoft Virtual Server
- 优点
  - ✓无需硬件协助
  - ✓不需要更改客户OS
  - ✓隔离、安全
- 缺点
  - 性能损耗

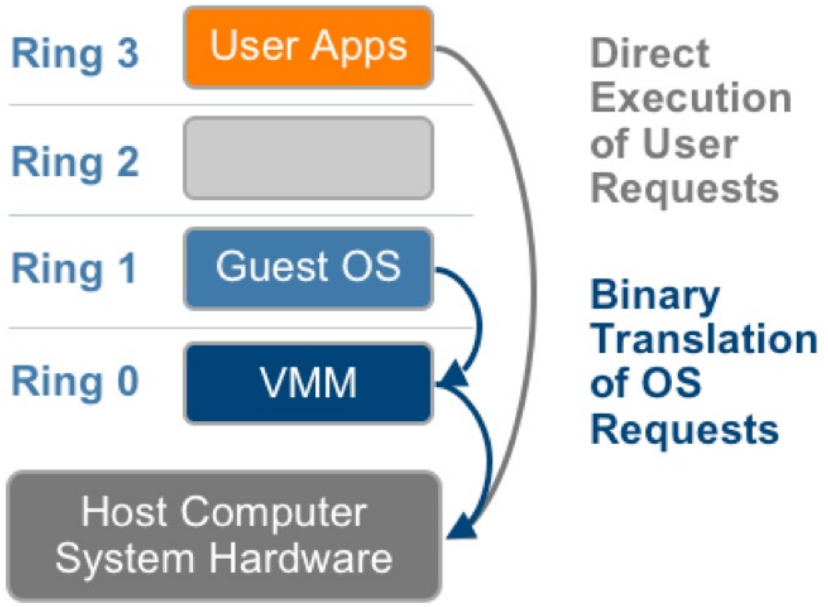


Figure 5 – The binary translation approach to x86 virtualization

方案一：二进制翻译 (cont'd)

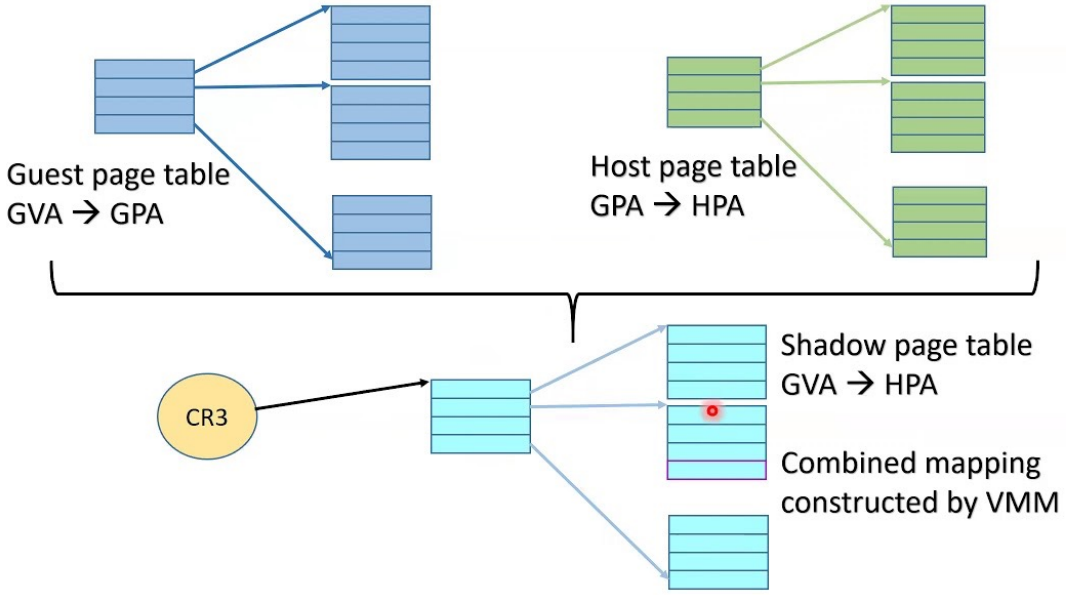
- 例子

```
ADD   R1+R2 -> R2
ADD   R2+R3 -> R3
MUL   2,R3  -> R3
```

```
LOAD Rx <- &emulated_R1
LOAD Ry <- &emulated_R2
LOAD Rz <- &emulated_R3
ADD Rx,Ry -> Ry
ADD Ry,Rz -> Rz
MUL 2,Rz  -> Rz
STORE Ry -> &emulated_R2
STORE Rz -> &emulated_R3
RET
```

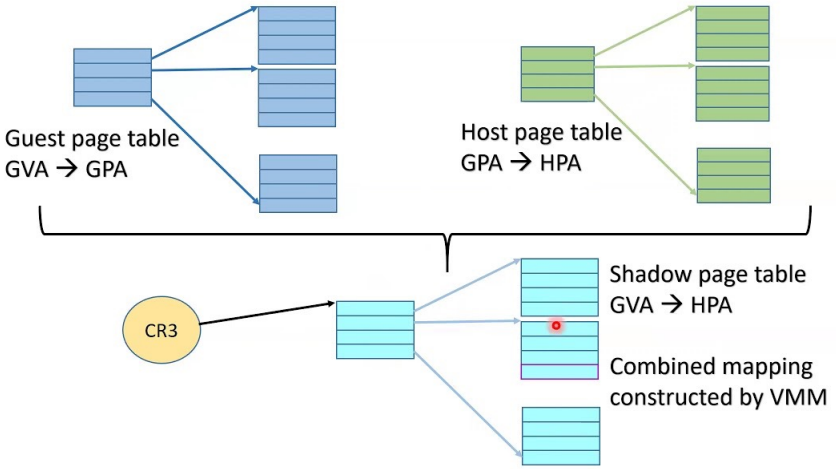
方案一：二进制翻译 (cont'd)

- 问题
  - ✓此方案需要两级地址翻译
    - 客户虚拟地址（GVA）—» 客户“假物理地址”（GPA）
    - 客户“假物理地址”（GPA）—» 主机真实物理地址（HPA）
  - ✓然而CPU只有一级地址翻译
- 解决方案
  - ✓影子页表（shadow page tables）



方案一：二进制翻译 (cont'd)

- 影子页表
  - ✓模拟CPU（客户看到的）包含一个CR3寄存器*指向假页表
  - ✓真实CPU包含一个CR3寄存器指向真正的页表
  - ✓当发生缺页中断，hypervisor检查假页表中是否存在相应的客户虚拟地址到假物理地址的入口
    - 若存在，计算假物理地址到真实物理地址的转换，并将客户虚拟地址到真实物理地址到转换安装到真页表
    - 若不存在，将缺页中断交给客户OS



*CR3为页表基地址寄存器

方案二：半虚拟化技术

- 半虚拟化（para-virtualization）
  - ✓Hypervisor为虚拟机提供超级调用（hypercalls），与系统调用类似
  - ✓半虚拟化涉及修改客户OS内核，将不可虚拟化的指令替换为超级调用
- 代表软件
  - ✓Xen, Denali
- 优点
  - ✓执行快
  - ✓虚拟化开销低
- 缺点
  - 可移植性差

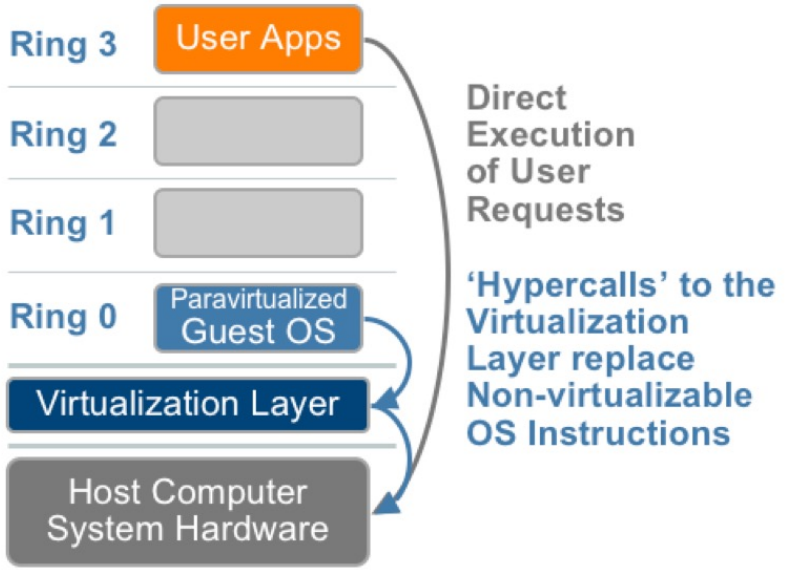


Figure 6 – The Paravirtualization approach to x86 Virtualization

方案三：硬件支持的虚拟化技术

- 硬件支持的虚拟化（hardware-assisted virtualization）
  - ✓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)
  - ✓过去不会引起下陷的敏感指令都能被hypervisor捕获
- 代表软件
  - ✓Intel VT-x, Xen 3.x
- 优点
  - ✓更快的执行

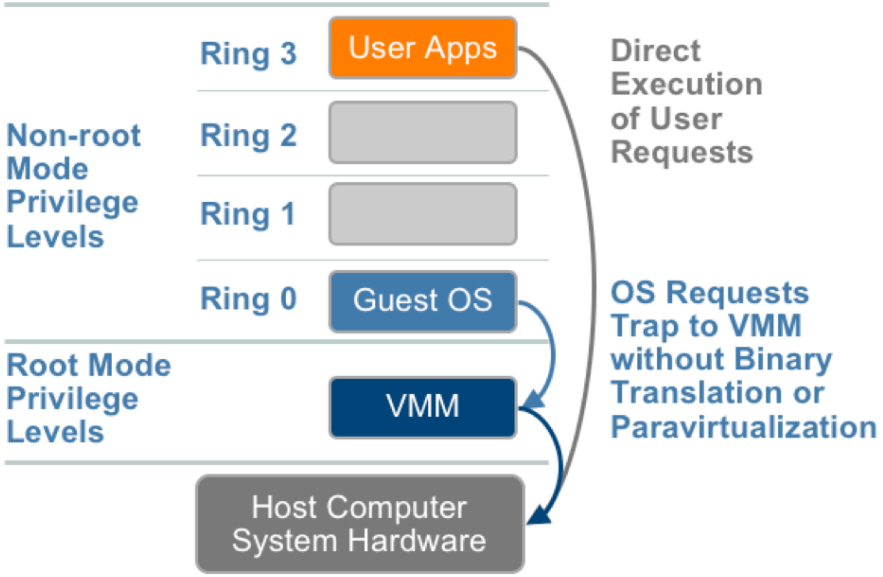
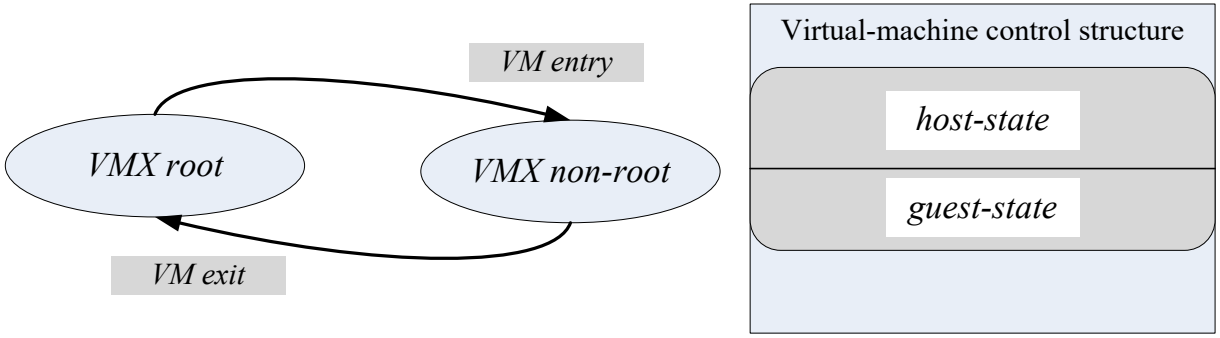


Figure 7 – The hardware assist approach to x86 virtualization

方案三：硬件支持的虚拟化技术

- Intel VT-x支持两种模式的操作
  - ✓VMX root – hypervisor
  - ✓VMX non-root – VM

- 模式切换
  - ✓VM entry - 从客户VM的状态区中加载处理器状态；控制权从VMM转移到VM
  - ✓VM exit - 将处理器状态保存在访客VM的状态区中；然后它从主机状态区域加载处理器状态，最后将控制转移到VMM



(a) (b)

虚拟化技术

- 虚拟化技术发展的动机
- 虚拟化技术的实现
- 虚拟机迁移
- 虚拟化技术的隐患
- 高效虚拟机监视器Xen

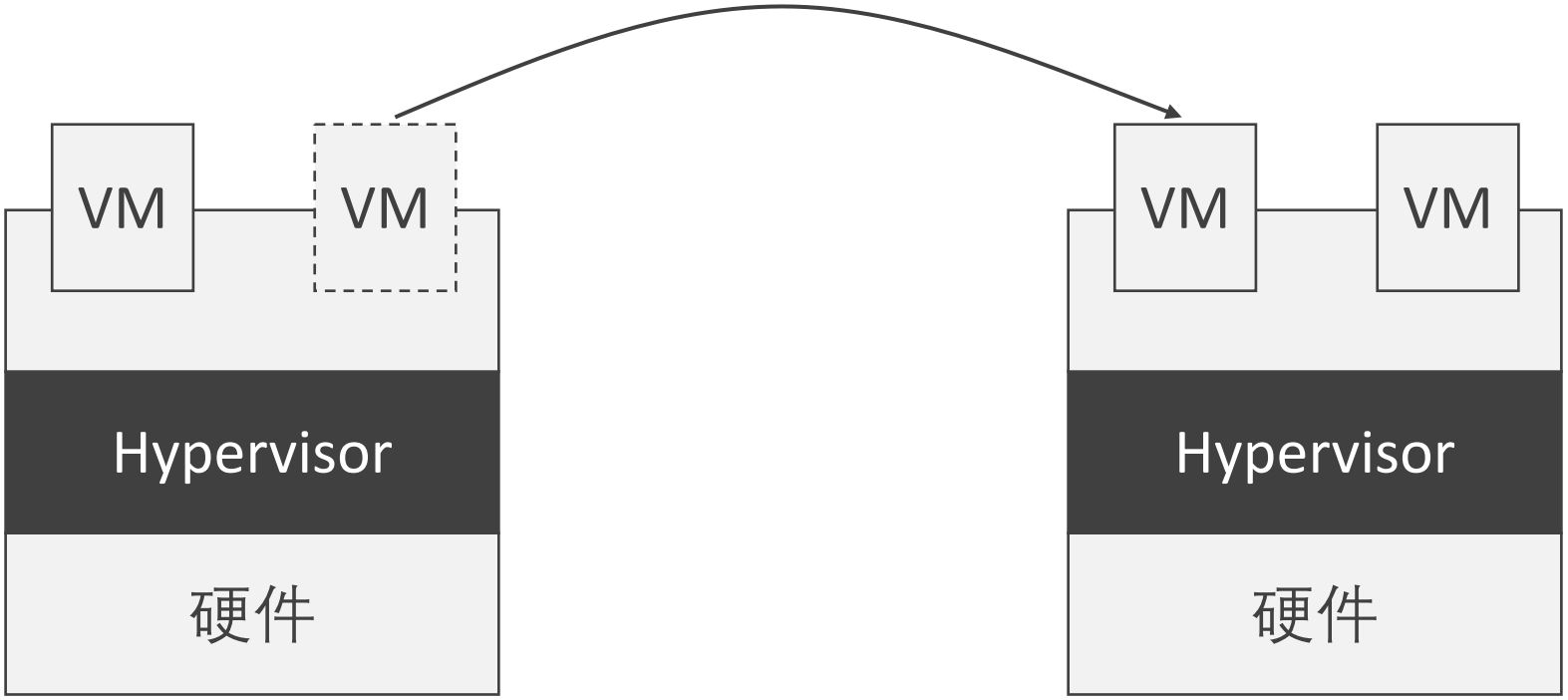
虚拟机迁移

虚拟机迁移是将虚拟机实例从源宿主机迁移到目标宿主机，并且在目标宿主机上能够将虚拟机运行状态恢复到其在迁移之前相同的状态，以便能够继续完成应用程序的任务。

从虚拟机迁移的源与目的地角度可分为

- 物理机到虚拟机的迁移（P2V）
- 虚拟机到虚拟机的迁移（V2V）
- 虚拟机到物理机的迁移（V2P）

虚拟机迁移 (cont'd)



虚拟机迁移示意图

虚拟机迁移 (cont'd)

实时迁移（Live Migration），就是保持虚拟机运行的同时，把它从一个计算机迁移到另一个计算机，并在目的计算机恢复运行的技术。

第一	云计算中心的物理服务器负载经常处于动态变化中，当一台物理服务器负载过大时，管理员可以将其上面的虚拟机迁移到其他服务器，达到负载平衡
第二	当升级维护服务器时，管理员可以将其上面的虚拟机迁移到其他服务器，等升级维护完成之后，再把虚拟机迁移回来

虚拟机迁移 (cont'd)

- 虚拟机迁移主要包含对以下三部分的迁移

disk

network

memory

虚拟机迁移 (cont'd)

- Disk迁移
  - ✓迁移disk的最大障碍在于需要占用大量时间和网络带宽，通常的解决办法是以共享的方式共享数据和文件系统，而非真正迁移
  - ✓目前大多数集群使用NAS(Network Attached Storage，网络连接存储)作为存储设备共享数据
  - ✓在局域网环境下，NAS已经完全可以实现异构平台之间，如Windows NT、UNIX等的数据级共享

虚拟机迁移 (cont'd)

- **Network迁移**
  - ✓虚拟机这种系统级别的封装方式意味着迁移时VM的所有网络设备，包括协议状态(如TCP连接状态)以及IP地址
  - ✓在局域网内，可以通过发送ARP重定向包，将VM的IP地址与目的机器的MAC地址相绑定，之后的所有包就可以发送到目的机器上

虚拟机迁移 (cont'd)

- **Memory的迁移**
  - ✓memory的迁移是虚拟机迁移最困难的部分

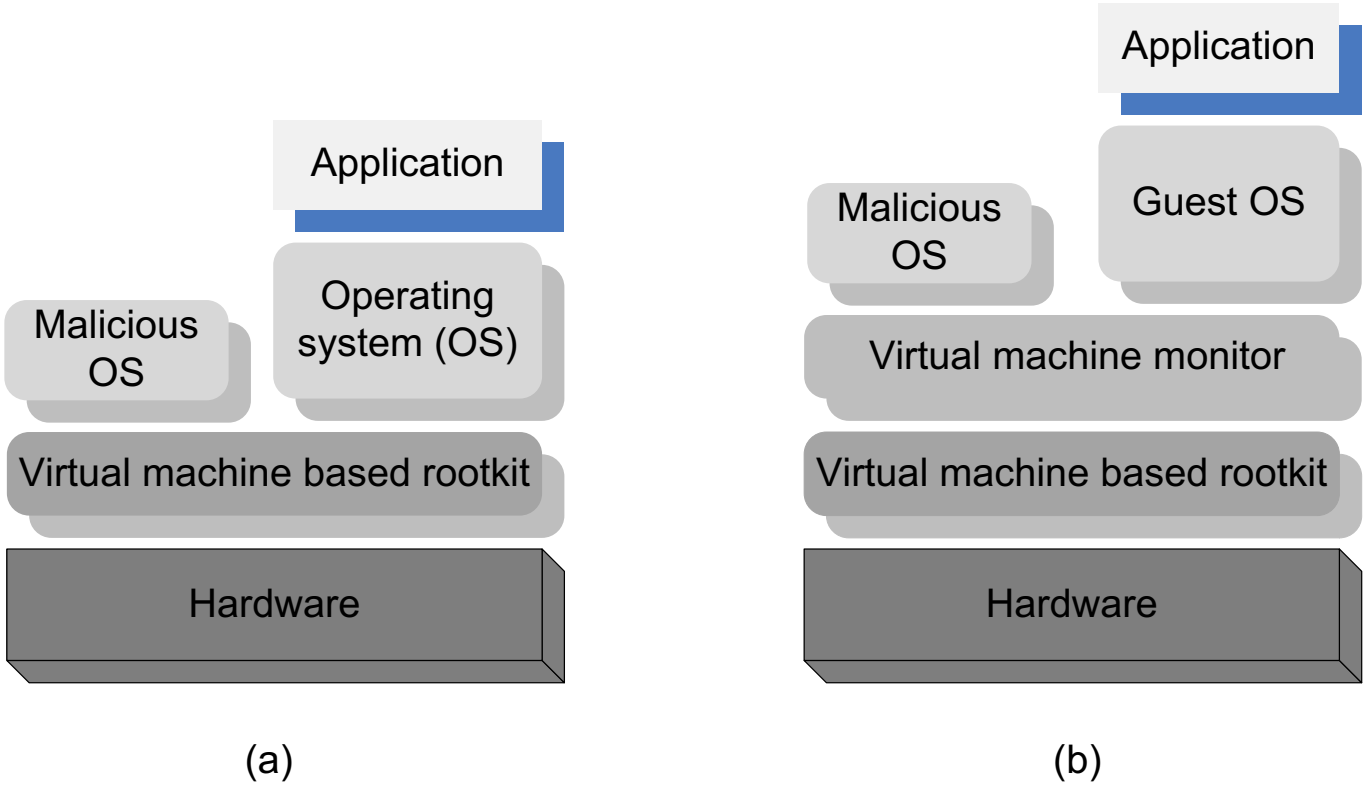
  - ✓方法一：Pre-copy memory迁移
    1. Hypervisor拷贝源虚机的所有内存页到目的节点
    2. 若在此期间出现dirty pages，则重新拷贝
    3. 当 dirty pages的数量较少时，短暂暂停虚机并拷贝剩下的所有页面

  - ✓方法二：Post-copy memory迁移
    1. 先暂停虚机，并将最小的执行状态集合（CPU状态、寄存器等）传输到目的节点
    2. 虚机在目的节点启动
    3. 源节点同步将剩下的内存页传输到目的节点
    4. 若期间虚机在目的节点的缺页中断将被重定向回源节点

虚拟化技术

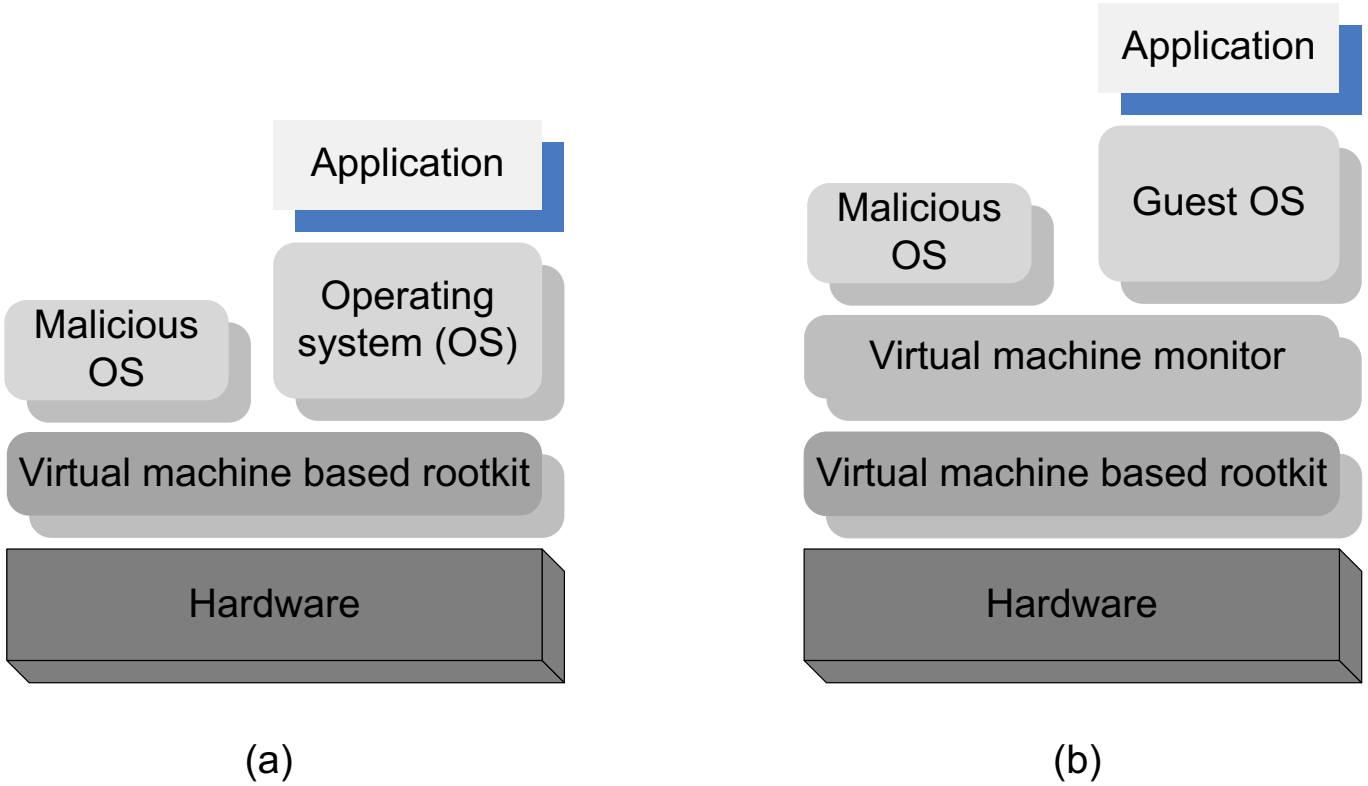
- 虚拟化技术发展的动机
- 虚拟化技术的实现
- 虚拟机迁移
- 虚拟化技术的隐患
- 高效虚拟机监视器Xen

虚拟化技术的隐患



The insertion of a Virtual-Machine Based Rootkit (VMBR) as the lowest layer of the software stack running on the physical hardware; (a) below an operating system; (b) below a legitimate virtual machine monitor. The VMBR enables a malicious OS to run surreptitiously and makes it invisible to the genuine or the guest OS and to the application.

虚拟化技术的隐患



Under the protection of the VMBR, the malicious OS could:

- observe the data, the events, or the state of the target system
- run services, such as spam relays or distributed denial-of-service attacks
- interfere with the application

虚拟化技术

- 虚拟化技术发展的动机
- 虚拟化技术的实现
- 虚拟机迁移
- 虚拟化技术的隐患
- 高效虚拟机监视器Xen

Xen的设计理念

- 不需要修改用户应用程序
- 对OS内核进行细微更改，以降低复杂性并提高性能（半虚拟化技术）
- 在一个服务器上能同时支持一百台虚拟机（与非虚拟化的操作系统基本无异）
- 强大的性能隔离

https://xenproject.org/

移植到Xen的成本

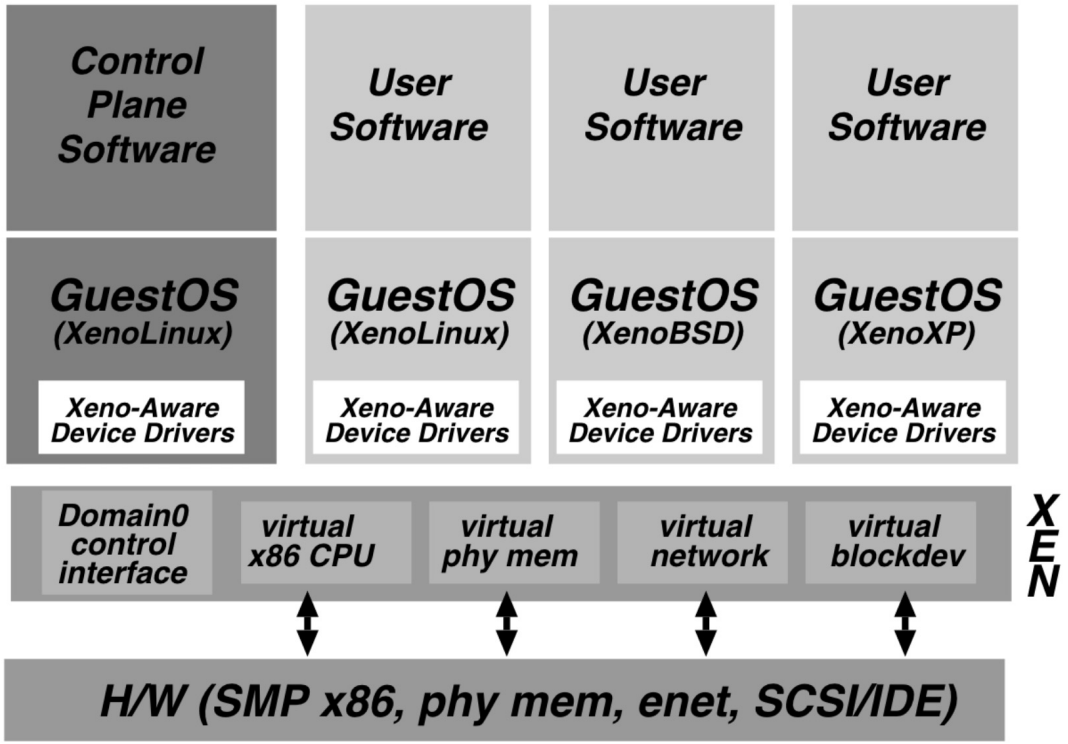
OS subsection	# lines	
	Linux	XP
Architecture-independent	78	1299
Virtual network driver	484	–
Virtual block-device driver	1070	–
Xen-specific (non-driver)	1363	3321
Total	**2995**	**4620**
(Portion of total x86 code base	**1.36%**	**0.04%)**

Table 2: The simplicity of porting commodity OSes to Xen. The cost metric is the number of lines of reasonably commented and formatted code which are modified or added compared with the original x86 code base (excluding device drivers).

- 两个系统均可被高效地移植到Xen中
- Windows XP的修改量较大，且移植的代价比Linux高

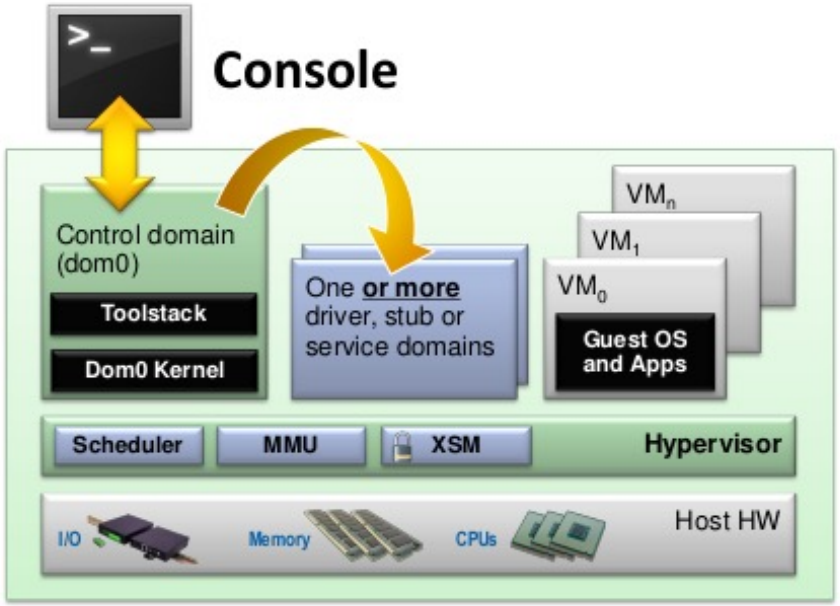
Xen架构

- Type-1 hypervisor
- 客户虚拟机在Xen中称为“domain”
  - ✓dom0： 专用于执行Xen控制功能和特权指令
  - ✓domU：用户域



Xen架构

- dom0具有如下两个功能：
  - ✓dom0能够与hypervisor对话，以改变其配置或指示其启动和停止guest VMs
  - ✓dom0（不是hypervisor）包含寻址硬件所需的设备驱动程序，是与硬件交互的主要接口，与客户操作系统共享硬件资源



Console
- Interface to the outside world

Control Domain aka Dom0
- Dom0 kernel with drivers
- Xen Project Management Toolstack

Guest Domains
- Your apps

Driver/Stub/Service Domain(s)
- A "driver, device model or control service in a box"
- De-privileged and isolated
- Lifetime: start, stop, kill

Xen内存管理设计

- 旁路转换缓冲（TLB）刷新
  - ✓TLB（Translation Lookaside Buffer）
    - 缓存最近虚拟地址映射至物理地址的页表项
  - ✓问题
    - X86体系架构不支持TLB标记、也不支持TLB软件管理
    - 当hypervisor激活不同的OS时，地址空间切换需要完整的TLB刷新
  - ✓解决方案
    - 每个访客OS的地址空间保留顶部64MB加载Xen，不能被访客OS访问或重新映射
    - 将硬件页表的管理委托给访客OS，Xen干预最少
    - 避免完整的TLB刷新影响性能

Xen内存管理设计 (cont'd)

- 客户OS创建页面
  - ✓每当客户OS需要新的页面（比如创建新进程），它从自己的内存中分配和初始化一个页面，用Xen注册
  - ✓写操作权交给hypervisor，访客OS只能映射自己的页面
  - ✓所有写操作需要经过Xen验证，可以批量化以提升性能

Xen CPU管理设计

- CPU特权级别
  - ✓通常OS内核运行在最高特权级别，而hypervisor应该比客户OS具有更高的级别
  - ✓许多处理器架构仅有2个级别，导致客户OS与应用需共享同一个较低的特权级别
  - ✓x86保护环模型具有4个特权级别（0-3）。在Xen中，hypervisor运行在0级，客户OS运行在1级，应用程序运行在3级
  - ✓防止客户OS直接执行特权指令，同时使其与应用程序安全隔离

Xen CPU管理设计

- 超级调用（hypercall）
  - ✓应用程序使用由Xen处理的超级调用进行系统调用
  - ✓访客OS发出的特权指令是半虚拟化的，需要由Xen验证才能使用
  - ✓访客OS向Xen注册一个描述表，包含用于验证的异常处理程序的地址，与原生x86处理程序相同
  - ✓访客OS可注册快速异常处理程序，绕过Xen直接执行

Xen 数据传输设计

- I/O环
  - ✓客户OS和Xen之间的数据操作由一个非常高效的数据结构完成：I/O环
  - ✓循环队列，由描述符组成，不包含数据，实现零拷贝
  - ✓每个请求有唯一的ID
  - ✓Xen的响应有相同的ID，以避免乱序 (out of order)

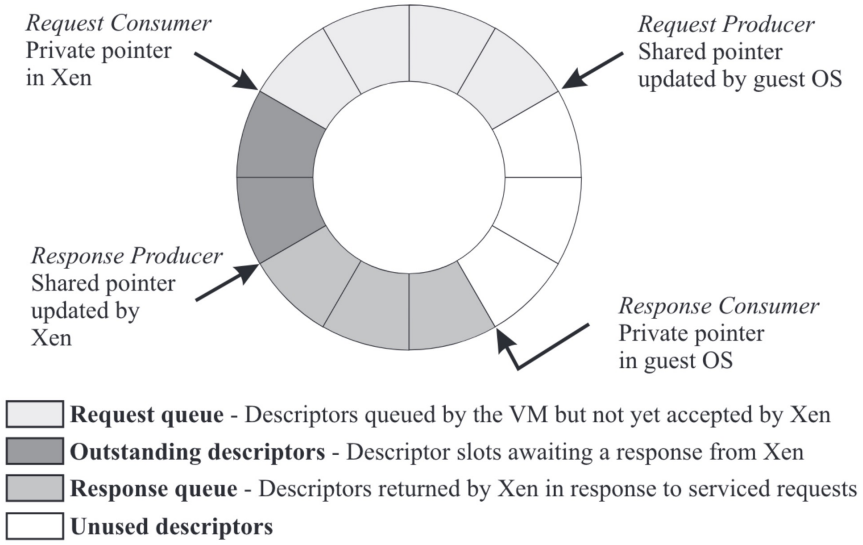


Figure 2: The structure of asynchronous I/O rings, which are used for data transfer between Xen and guest OSes.

性能比较 – 相对性能

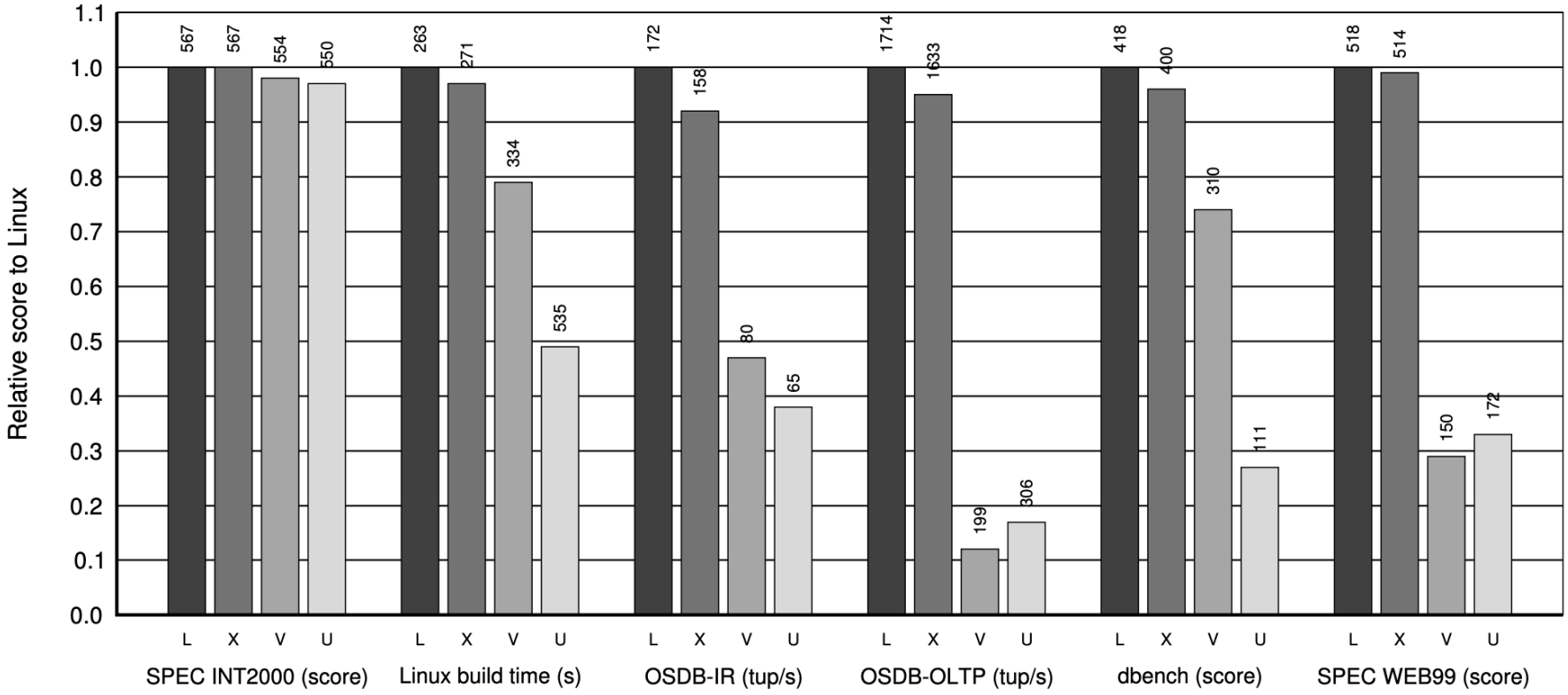


Figure 3: Relative performance of native Linux (L), XenoLinux (X), VMware workstation 3.2 (V) and User-Mode Linux (U).

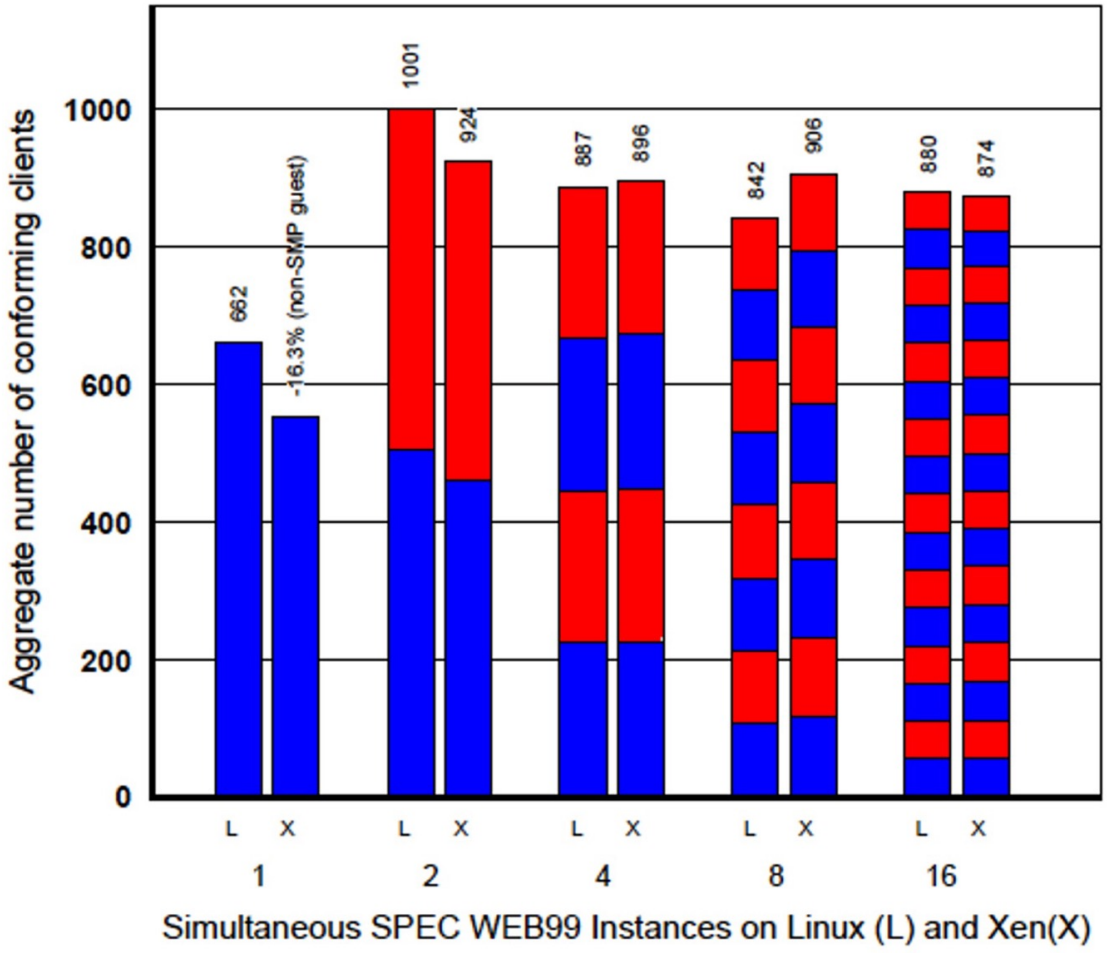
性能比较 – 并行运行

Multiple Apache processes in Linux

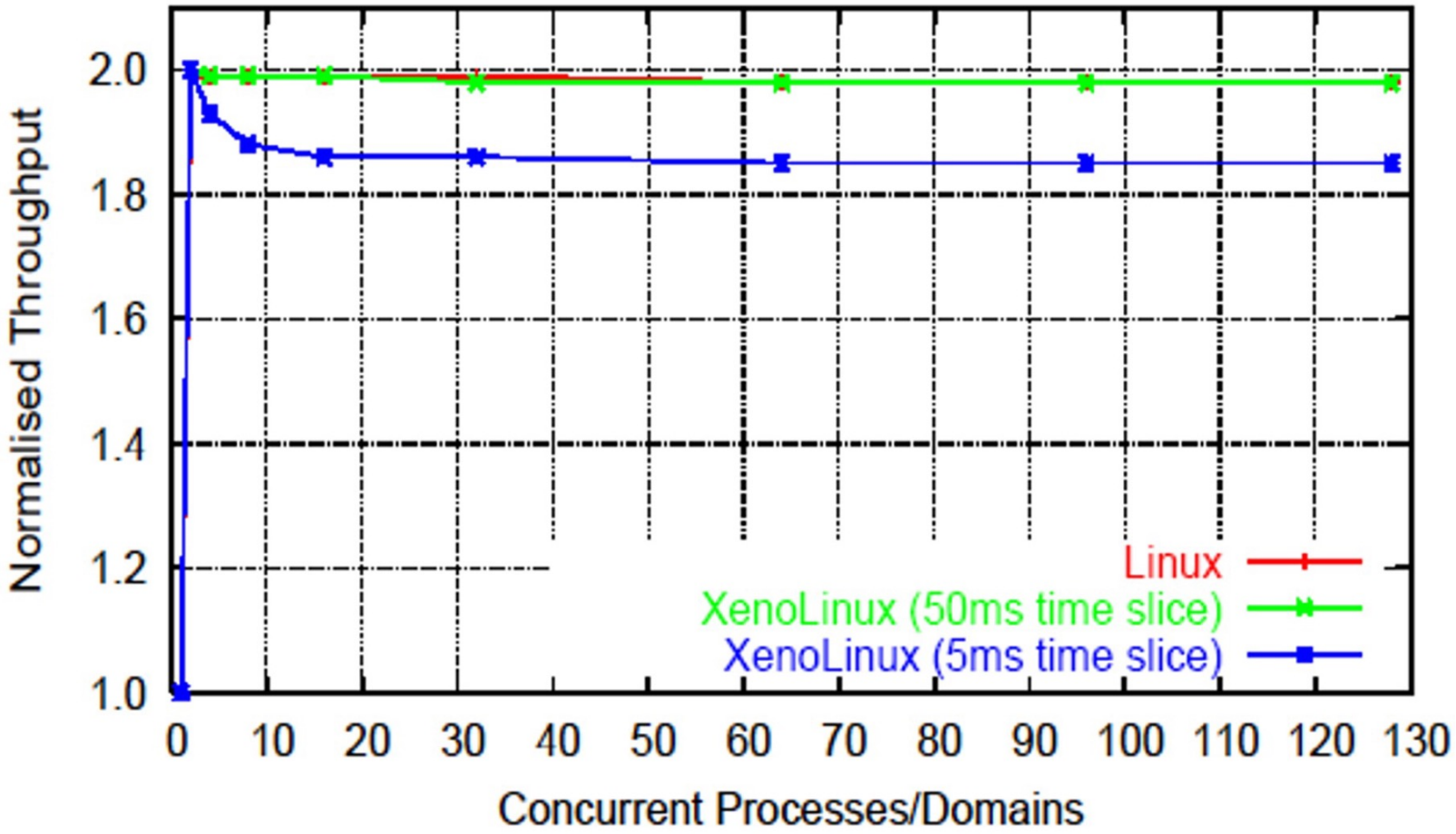
vs.

One Apache process in each guest OS

Similar performance to running seperate processes



性能比较 – 可扩展性



Normalized aggregate performance of a subset of SPEC CINT2000 (CPU intensive) running concurrently on 1-128 domains

性能评估总结

- Demonstrated good performance for I/O intensive, direct access through kernel
- Can support lots of concurrent virtual machines
- Good performance isolation
- Can run lots of CPU intensive applications

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

https://zbchern.github.io/sse316.html