参考文献

- 参考文献1：软硬件协同
- 参考文献2：永恒的内存战争
- 参考文献3：三星Knox
- 参考文献4：HyperKRP内核运行时安全架构
- 参考文献5：虚拟化软件栈安全研究
- 参考文献6：虚拟化能力测评

Case1： Knox Overview

1 ANDROID FRAMEWORK

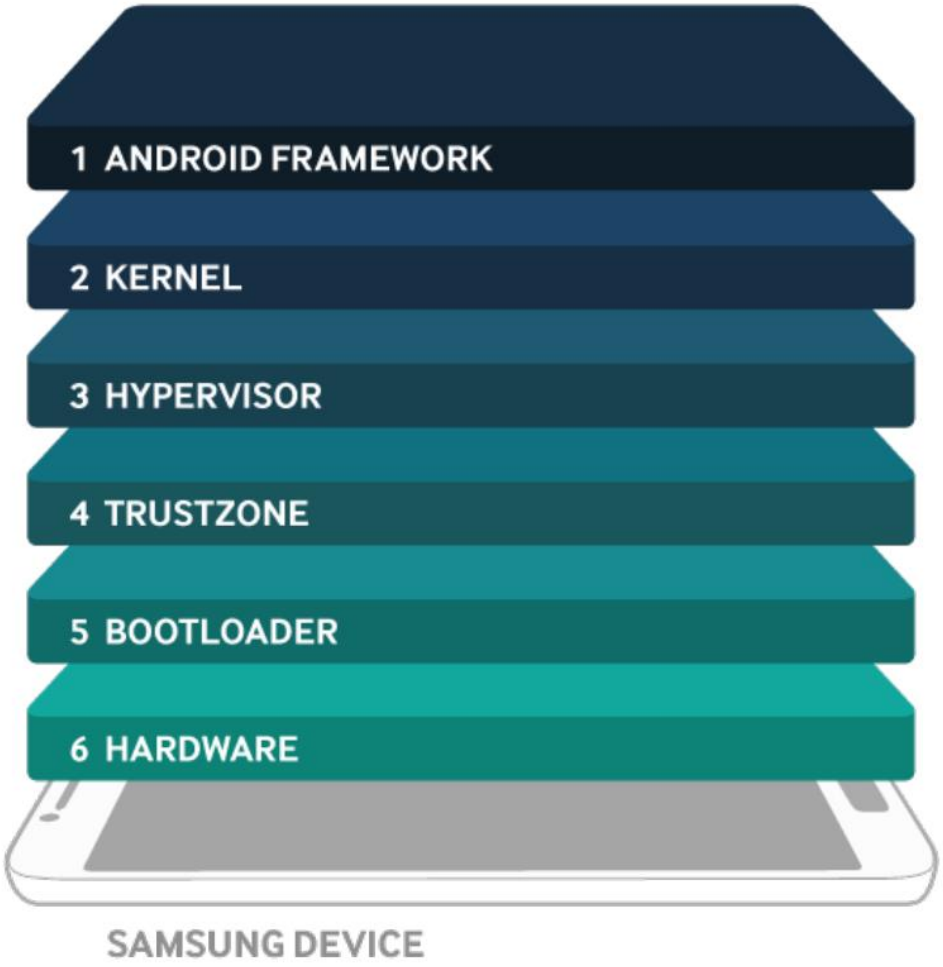
- Knox Workspace
- SE for Android
- Sensitive Data Protection

2 KERNEL

- DM-Verity
- On Device Encryption
- SE Linux

3 HYPERVISOR

- Real-time Kernel Protection



4 TRUSTZONE

- Attestation
- Trusted UI
- TIMA KeyStore
- Client Certificate Management
- Real-time Kernel Protection
- Periodic Kernel Measurement

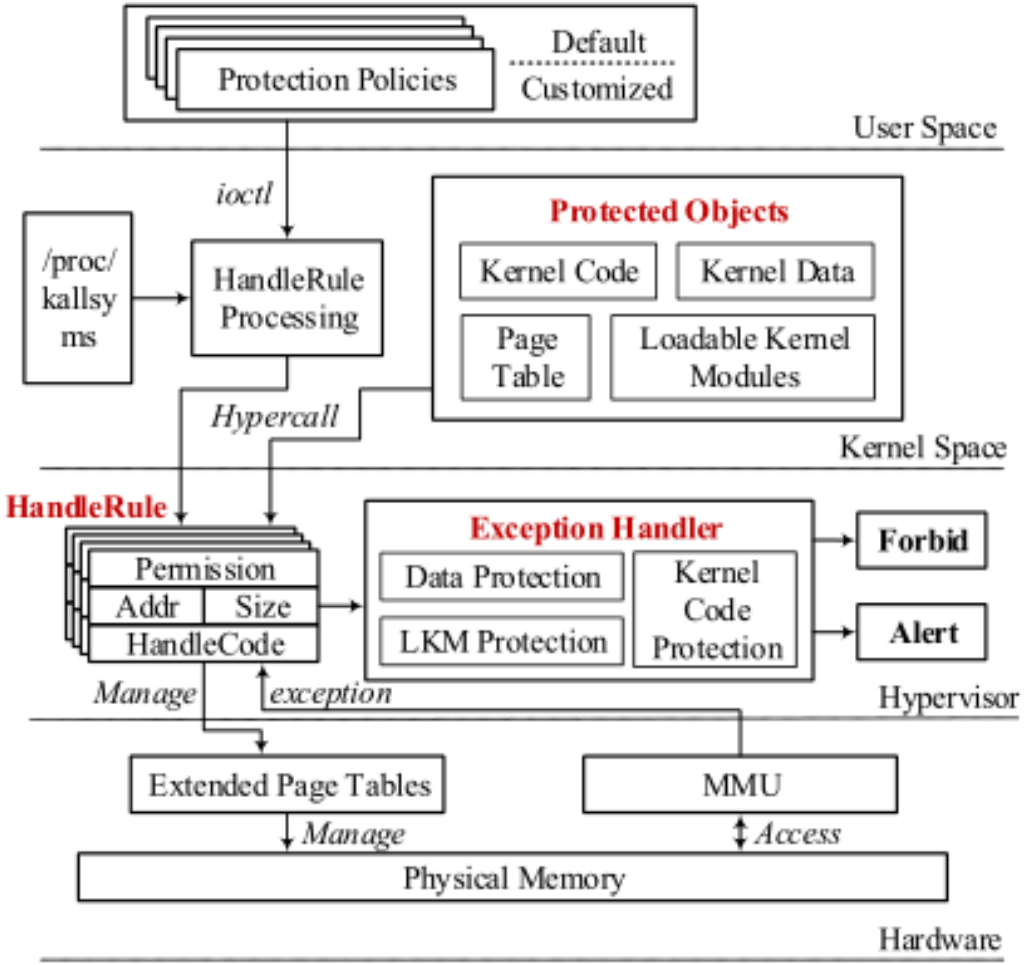
5 BOOTLOADER

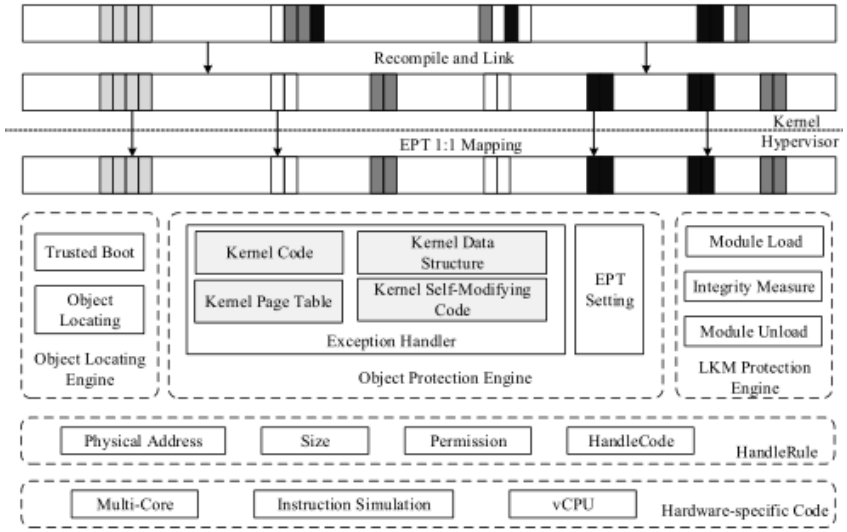
- Secure/Trusted Boot
- Rollback Prevention

6 HARDWARE

- Samsung Secure Boot Key
- Rollback Prevention Fuses
- Device Root Key
- Device Unique Hardware Key
- Knox Warranty Bit

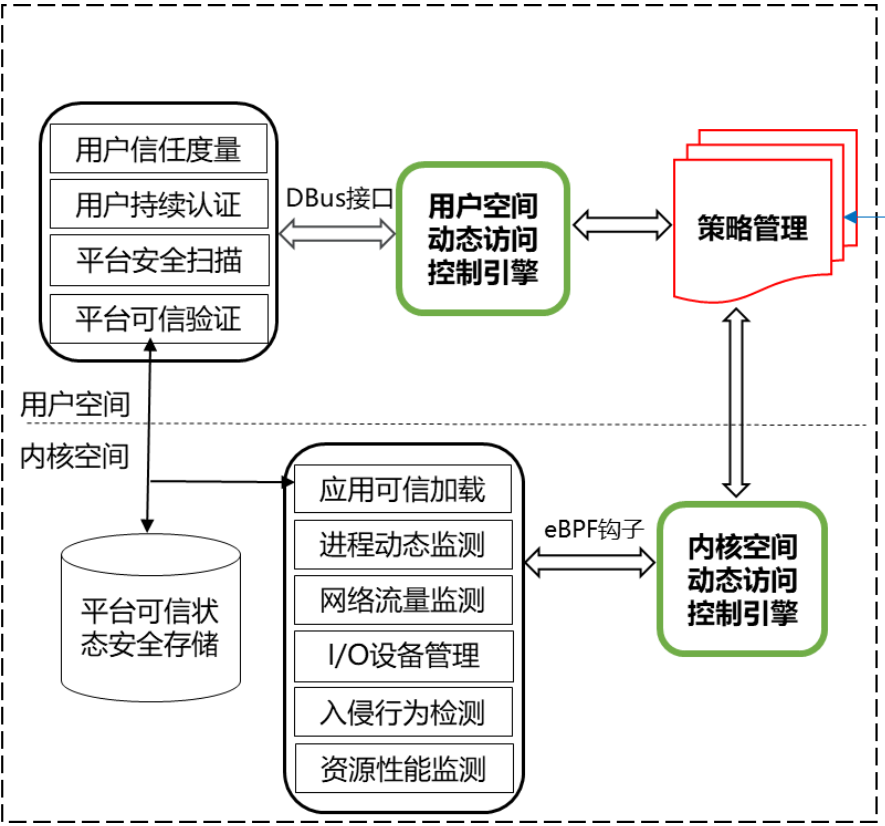
Case2：HyperKRP

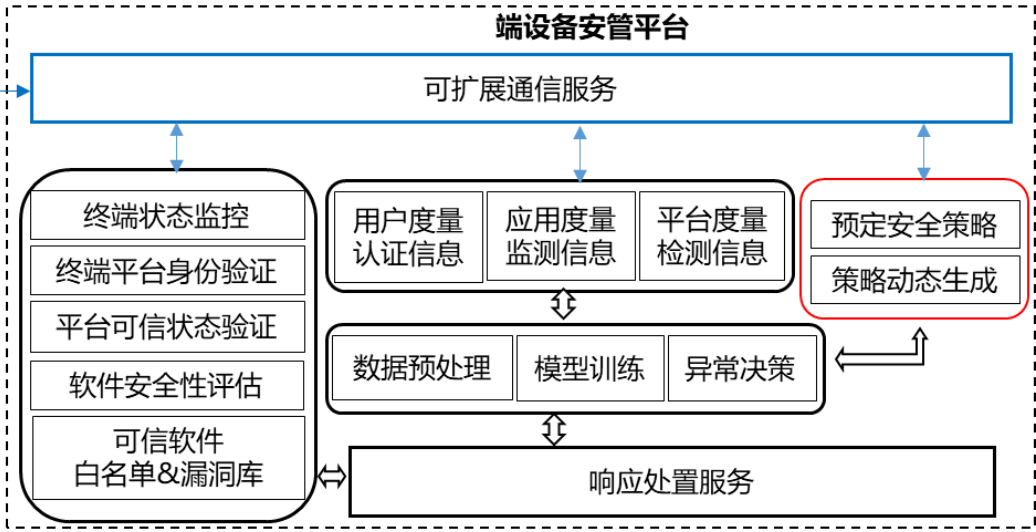




Globcom'22-HyperKRP: A Kernel Runtime Security Architecture with A Tiny Hypervisor on Commodity Hardware

Case3: 持续度量架构





利用操作系统原生技术：eBPF、DBus、Hook、PAM、IMA等融合现有安全技术：操作系统认证、可信度量、平台证明等

操作系统安全

Operating system security

[第13次课] 虚拟机技术原理

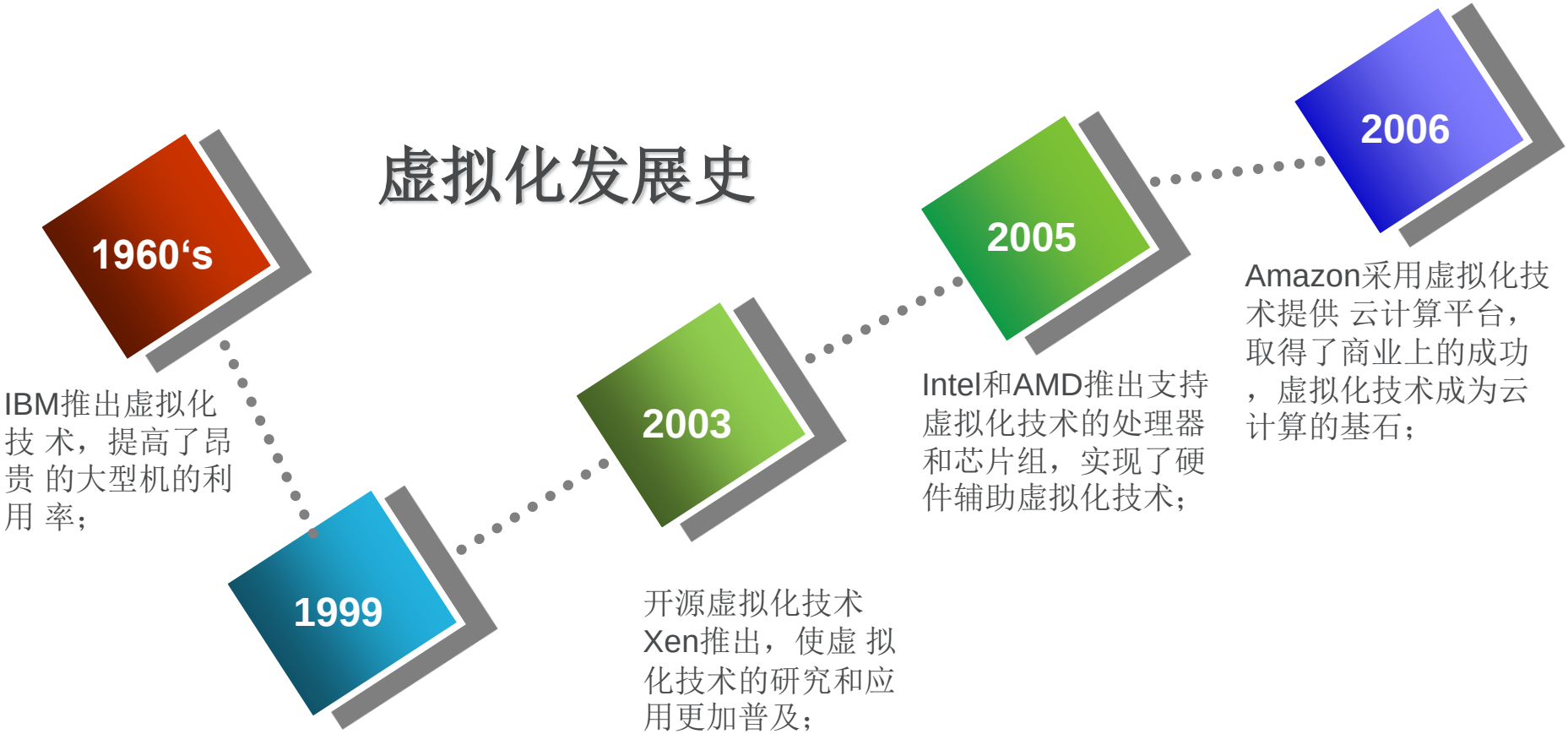
授课教师：涂碧波

授课时间：2024.05.24

云计算模型

- 云计算的一个核心理念是借助一定的商业模式（IaaS、PaaS和SaaS etc.）把强大的计算能力按需提供给用户使用。
- 虚拟化是云计算的使能技术。

虚拟化的发展


虚拟化发展史
1960‘s
IBM推出虚拟化技 术，提高了昂贵 的大型机的利用 率；
1999
VMware公司解决了X86虚拟化问题，推出了X86平台的虚拟机软件，使虚拟化技术开始走向普通用户
2003
开源虚拟化技术Xen推出，使虚 拟化技术的研究和应用更加普及；
2005
Intel和AMD推出支持虚拟化技术的处理器和芯片组，实现了硬件辅助虚拟化技术；
2006
Amazon采用虚拟化技术提供 云计算平台，取得了商业上的成功，虚拟化技术成为云计算的基石；

虚拟化应用模式

- Division – Time-shared System
- Cluster-Single System Image（SSI）
- Consolidation- Server Consolidation
- Remote- Virtual Desktop Infrastructure

虚拟化概念

- 虚拟化是一个广义的术语，在计算机科学领域中，虚拟化代表着对计算/存储资源的抽象。虚拟化是一个抽象层，它将物理硬件与虚拟环境分开，从而提供更高的 IT 资源利用率和灵活性。
  - 虚拟化意味着对计算机资源的抽象，把物理资源变为逻辑上可管理的资源，打破物理结构之间的壁垒。
  - 经过虚拟化后的逻辑资源对用户隐藏了不必要的细节。

- 理论上，虚拟化技术采用的抽象层次可以是图中的任意层次。选择的多样性则决定了虚拟化的多样性，然而实质是一样的，将底层资源进行虚拟，为上层提供多样化的执行环境。
- **封装**

虚拟化分类

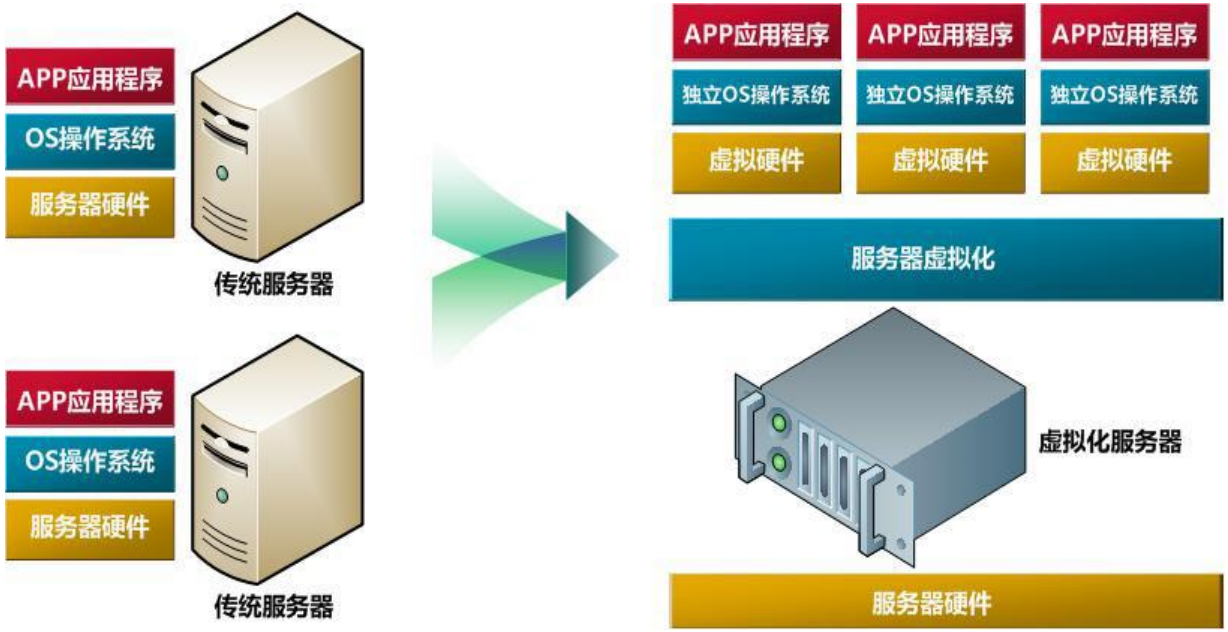
- 指令级虚拟化：它通过纯软件方法，即代码翻译，模拟出与实际运行的应用程序不同的指令集去执行，称为模拟器。
  - 如Bochs和Qemu，将虚拟机发出的执行翻译成本地指令，在真实硬件上执行
- 资源虚拟化，针对特定的系统资源的虚拟化，比如内存、存储、网络资源等。
- 程序库级虚拟化，它通过在应用程序和运行库之间引入仿真的系统API，从而隐藏与操作系统相关的细节。为应用程序提供一个虚拟的运行环境，包括仿真、模拟、解释技术等。
  - 其典型代表为Cygwin、Wine等。Wine可将win32程序运行在类Unix系统上，因它为win32程序提供了Windows的系统API。
- 编程语言虚拟化：它是在应用层提供一套自定义的、与处理器无关的编程语言指令集。
  - 将编程语言的执行转为成具体平台所使用的指令，屏蔽了硬件的异构性，解决可执行程序在不同体系结构计算机之间迁移问题。
  - 最常见的包括Java虚拟机JVM、Mono、.NET等。

虚拟化分类

- 系统虚拟化：又称为平台虚拟化，针对计算机和操作系统的虚拟化，为每个虚拟机提供虚拟硬件的视图（如CPU、内存和I/O设备），从而让虚拟机中的操作系统或者应用程序认为其运行在真实硬件之上。
  - 实现操作系统与物理计算机的分离
  - 使得在一台物理计算机上可以同时安装和运行一个或多个虚拟的操作系统，并表现出高度的客户系统的隔离性，这种隔离性不仅存在于每个客户机之间，而且还存在于客户机和宿主机之间。
  - 常见的云平台即采用这种虚拟化，如 VMware ESX、Xen、KVM等。

服务器虚拟化

- 服务器虚拟化是将系统虚拟化技术应用于服务器上。
  - 将一台服务器变成几台甚至上百台相互隔离的虚拟服务器，不再受限于物理上的界限，而是让CPU、内存、磁盘、I/O等硬件变成可以动态管理的“资源池”。
  - 提高资源的利用率，简化系统管理，实现服务器整合，让IT对业务的变化更具适应力（与云计算需求完全吻合）。



虚拟之前：需要两个服务器分别部署不同的操作系统部署不同的应用

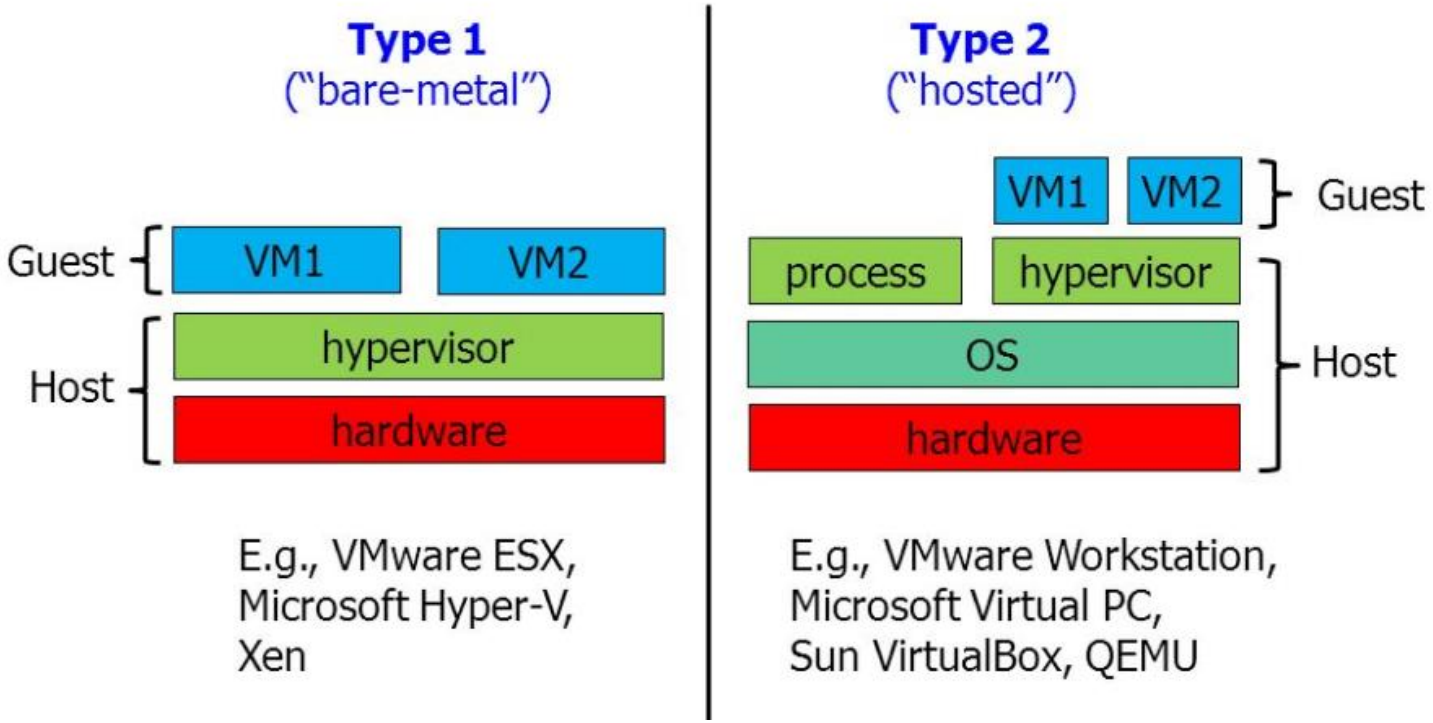
虚拟之后：只需要一个服务器同时运行两个不同的系统完成不同的业务

Hypervisor的功能和目标

- 虚拟化技术由Hypervisor实现，Hypervisor位于硬件和虚拟机之间，其主要功能是：
  - 将虚拟硬件资源映射到真正的硬件资源上，实现资源共享
    - Hypervisor利用EPT/NPT页表对虚拟机进行逻辑隔离。
    - Hypervisor利用I/O模拟实现外围设备的共享和I/O操作模拟。
  - 对虚拟机之间进行强制保护和安全隔离，保证执行环境的独立性
  - 完成虚拟机管理（创建、启动、关闭、销毁等）、调度和迁移
  - 实现虚拟机之间的通信
- Hypervisor的目标：
  - 保真度（fidelity）：运行在Hypervisor之上的VM与运行在硬件上无差别
  - 性能（performance）：VM中的大部分指令可直接在硬件上运行，无需Hypervisor的干涉
  - 资源控制（resource control）：必须能够管理所有的系统资源
  - 安全（safety）：VM之间互不影响

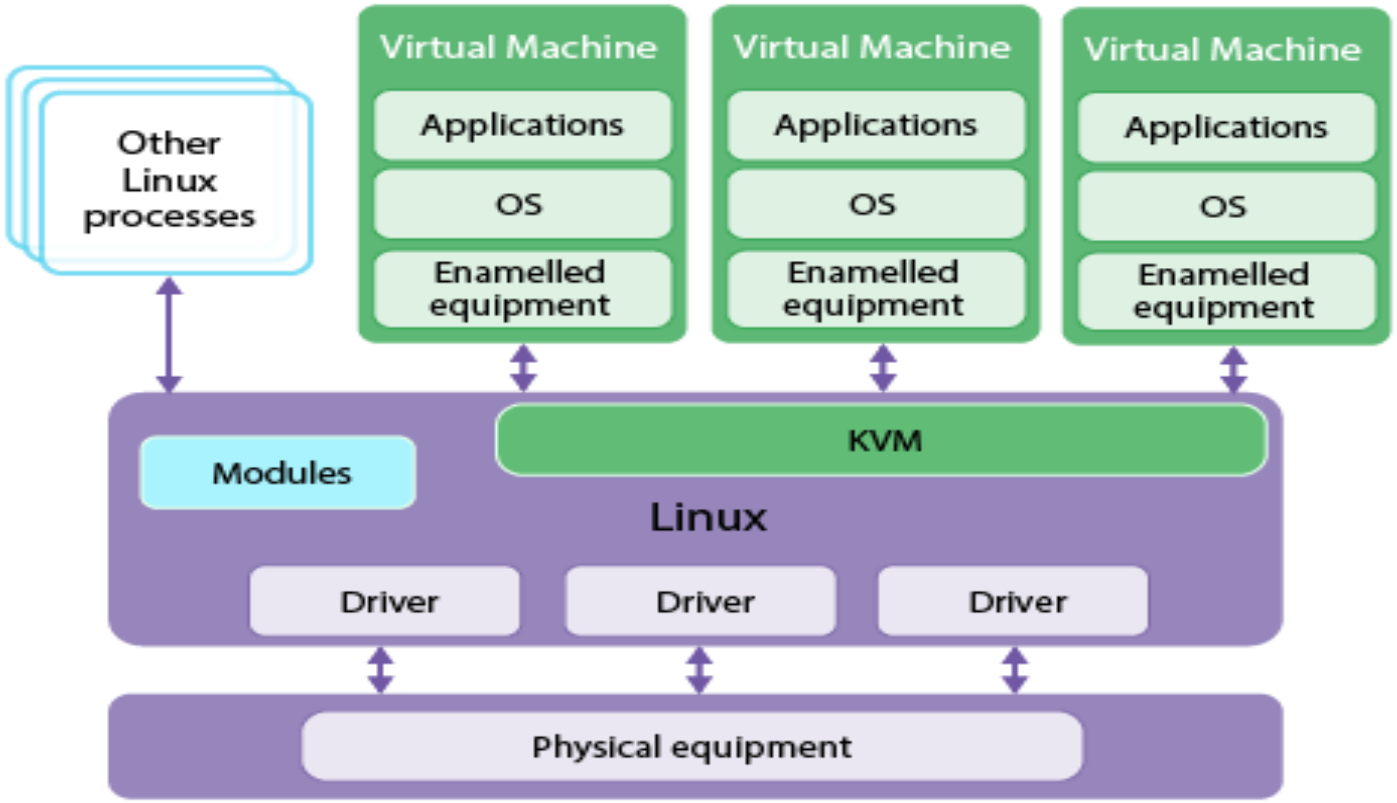
服务器虚拟化类型

- Hypervisor可被看作是一个位于计算机硬件和Guest OS之间的特殊操作系统，负责管理和隔离上层运行的VM。
- 根据Hypervisor与硬件之间的位置关系，Hypervisor可被分为2种模式：宿主模式（Type 2）、裸机模式（Type 1）。



典型的宿主模式-KVM

- KVM自Linux-2.6.20入住内核主线，依赖硬件虚拟化(Intel-VT、AMD-V)
  - KVM通过添加内核模块的形式将内核自身变成Hypervisor
  - KVM引入了客户模式（有自己的内核和用户模式），用于执行Guest OS的非I/O代码
  - KVM的I/O虚拟化工作借助Qemu完成，显著地降低了实现的工作量



虚拟化技术的实现方式

- 根据是否需要修改Guest OS，虚拟化分为两类：全虚拟化和半虚拟化。根据其实现方式全虚拟化又可分为软件模拟和硬件辅助两类。
  - 全虚拟化可以直接创建一个已有的客户机，不需修改，但往往需要获得处理器级别的硬件支持（也称硬件虚拟化），或者对客户机执行代码进行动态二进制代码转换（Binary translation）
  - 半虚拟化通过修改客户操作系统的内核来配合此时的环境，从而达到较好的性能，但也付出了兼容性和维护性差的代价
  - 硬件虚拟化提供了全新的架构，简化了Hypervisor的设计和实现，并提升了其对虚拟机的掌控灵活度和力度，相比那些纯软件虚拟实现方法会在很大程度上提高性能

全虚拟化

- 全虚拟化技术的核心理念是Hypervisor可以向VM虚拟出和真实硬件完全相同的运行环境，为每个VM提供完整的硬件支持服务，包括虚拟BIOS、虚拟设备和虚拟内存管理等。
  - 这个过程并不需要硬件或操作系统的协助，因而不需要修改Guest OS的内核，Guest OS完全感知不到其运行在一个虚拟化的环境中
  - 全虚拟化技术使用户不需要另外附加安装特殊程序，就可以在虚拟环境下安装操作系统，就好像在真正硬件平台上安装系统一样。
  - 多数运行在Guest OS中的特权指令被Hypervisor捕获并模拟这些指令。一些用户模式下无法被捕获的指令将通过二进制翻译技术处理。
  - 著名的全虚拟化Hypervisor有 Microsoft Virtual PC、VMware Workstation、Sun Virtual Box和QEMU等。

半虚拟化

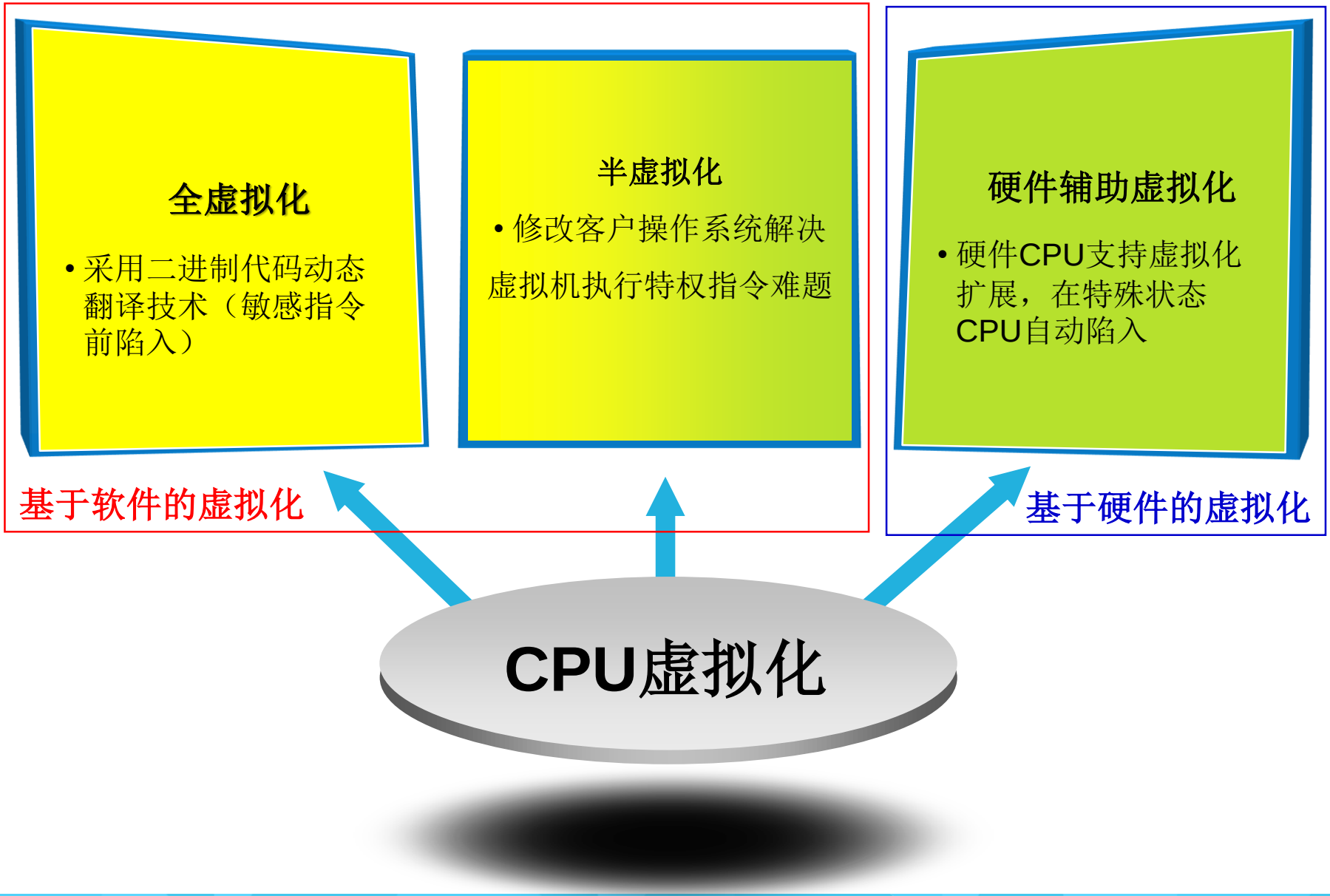
- 半虚拟化是一种修改 Guest OS 部分访问特权状态的代码以便直接与Hypervisor交互的技术。
  - 在超虚拟化虚拟机中，部分硬件接口以软件的形式提供给Guest OS，当前以Hypercall（Hypervisor提供给 Guest OS 的直接超级调用，与系统调用类似）的方式来提供。
  - 例如，Guest OS 把切换页表的代码修改为调用 Hypercall 来直接完成修改影子 CR3 寄存器和翻译地址的工作。由于不需要产生额外的异常和模拟硬件执行流程，超虚拟化可以大幅度提高性能。
  - 著名的 半虚拟化Hypervisor 有 Denali、Xen。

硬件辅助虚拟化

- 硬件辅助虚拟化是指借助硬件的支持来实现高效的全虚拟化。
  - 为实现体系结构支持的硬件辅助虚拟化技术，Intel提出了Intel-VT，AMD提出了AMD-V以及ARM提出了ARM-VE。
  - 有了硬件技术的支持，Guest OS 和 Hypervisor的执行环境自动地实现了隔离，Guest OS 有自己的“全套寄存器”，可以直接运行在最高级别。
  - 因此，Guest OS 能够自己执行修改页表的汇编指令。

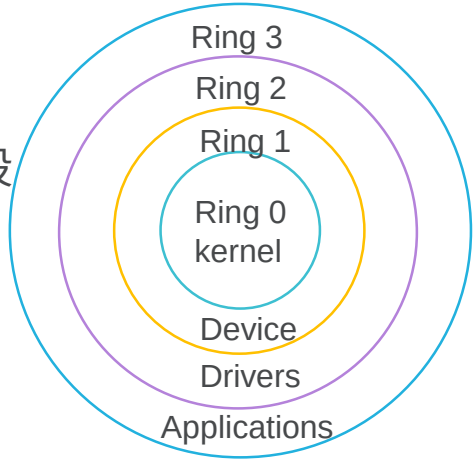
CPU虚拟化

CPU虚拟化


全虚拟化
• 采用二进制代码动态翻译技术（敏感指令前陷入）
半虚拟化
• 修改客户操作系统解决虚拟机执行特权指令难题
硬件辅助虚拟化
• 硬件CPU支持虚拟化扩展，在特殊状态CPU自动陷入
基于软件的虚拟化
基于硬件的虚拟化
CPU虚拟化

CPU虚拟化问题-1

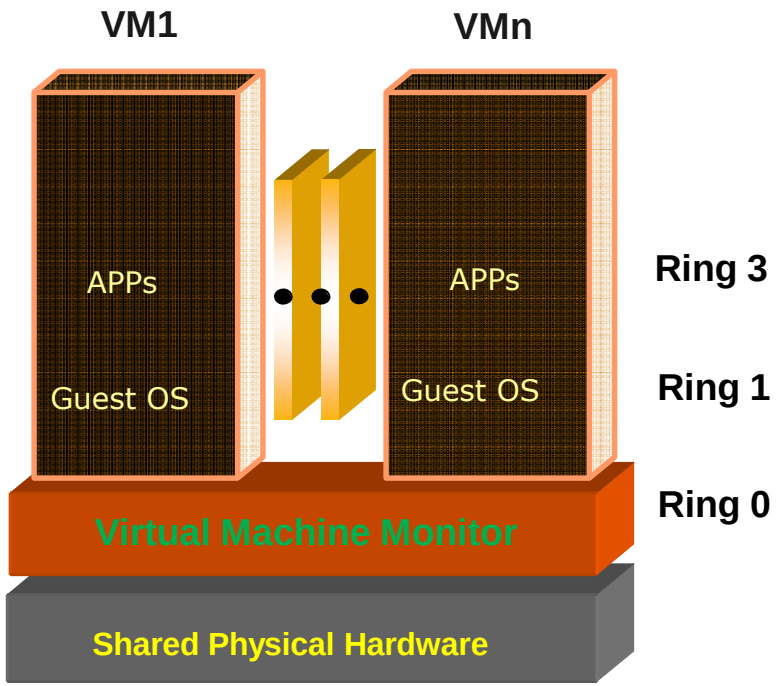
- 在传统x86处理器上，共有4种模式的操作，也就是常说的4个特权级，这些特权级经常被描述为保护环（protection ring）。
  - 4种模式的运行应具有的运行：
    - **Ring0: operating system kernel； Ring1: operating system services**
    - **Ring2: custom extensions；Ring3: ordinary user applications**
  - 操作系统和CPU合谋限制用户模式的应用程序的执行
  - 有3种主要的资源受到保护：内存，I/O端口以及执行特殊机器指令的能力。
    - 在任一时刻，CPU只在一个特定的特权级下运行，这决定了可以执行的代码。
    - 从外到内特权级别逐渐增高，当前的操作系统内核只用到其中的2个特权级：0和3。
- 对特权级的管理，CPU利用了段选择符和段选择表。
  - 数据段选择符可由程序直接加载到各个段寄存器当中
    - 比如ss（堆栈段寄存器）和ds（数据段寄存器）。
    - 包含了请求特权级（Requested Privilege Level，RPL）字段



CPU虚拟化问题-2

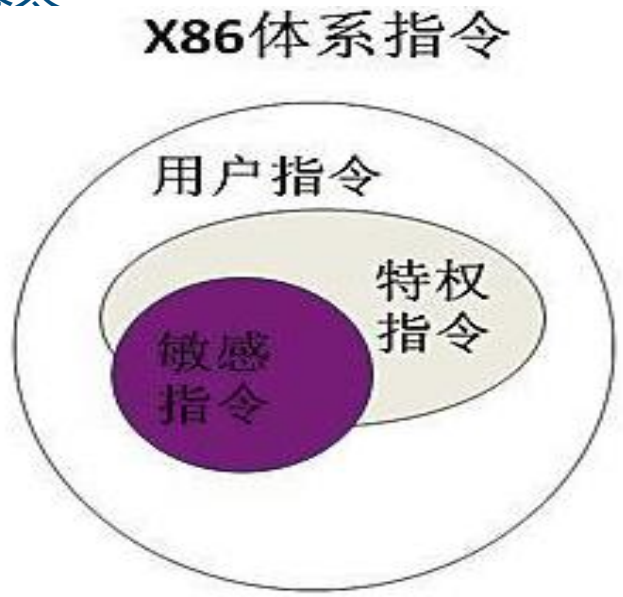
- x86在设计之初并没有考虑支持虚拟化，为了能够在x86平台上应用虚拟化，VMware、Xen等提出了基于软件的虚拟方法
  - 特权级压缩： Hypervisor必须运行在Ring 0，同时为了避免Guest OS控制系统资源，Guest OS不得不降低自身的运行级别而运行于Ring 1/3。
    - CPU只有运行在 Ring 0 ~ 2 级时，才可以访问特权资源或执行特权指令，而运行在Ring 0 级CPU可以访问所有的特权状态
    - 特权级压缩带来了许多问题

通常将这些问题统称为x86平台的虚拟化缺陷



CPU全虚拟化

- 全虚拟化是指Hypervisor模拟完整的底层硬件，使得为原始硬件设计的Guest OS或其它系统软件完全不做任何修改就可以在虚拟机中运行
- 对于特权指令：
  - 特权指令执行时，CPU产生一个保护异常（Protection Exception）→→异常处理传递给Hypervisor→→ Hypervisor模拟特权指令
- 对于某些敏感指令（非特权指令）则采用二进制代码动态翻译技术解决。
  - 二进制可以对17条敏感指令的代码进行替换，从而避免不确定行为的发生。
  - 为提高效率，二进制翻译后的代码会进行缓存，避免每次都进行翻译。
  - 在需要Hypervisor监控和模拟的位置插入陷入命令

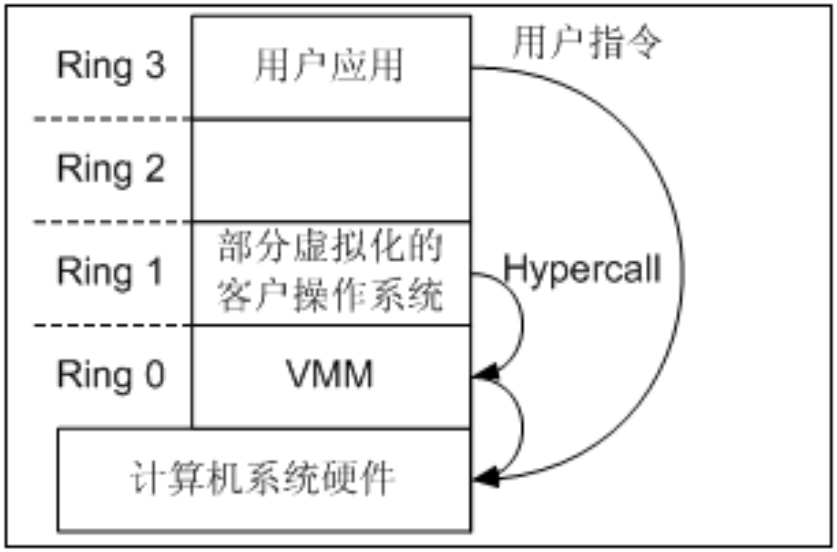


CPU全虚拟化优缺点

- 优点：无需对Guest OS进行修改，兼容性强。
- 缺点：需要执行大量的二进制翻译、系统调用截获等陷入操作，致使性能成为瓶颈。
  - 为加快系统调用的速度，Intel引入了SYSENTER和SYSEXIT指令，但SYSENTER指令总是换到Ring 0，且从0以外的特权级执行SYSEXIT指令将导致故障。SYSENTER和SYSEXIT指令总是先陷入到Hypervisor，经仿真后再交给Guest OS
  - x86使用EFLAGS.IF位来控制中断的屏蔽，修改IF位需要在CPL<=IOPL的情况下进行，否则将产生故障。Guest OS可能需要频繁地修改IF位，这会导致频繁地陷入Hypervisor

CPU半虚拟化

- 半虚拟化所采用的超级替换的方法是一种全新的设计理念：它将问题的中心，由Hypervisor移向Guest OS自身，通过主动的方式由Guest OS去处理这些指令，而不是被移交给Hypervisor做处理
  - 半虚拟化需对Guest OS进行一些修改，用超级系统调用(hypercall)模拟这些指令，避免了捕获异常、翻译、模拟的过程，性能损耗非常低
  - 通过修改Guest OS的内核。使Guest OS明确知道自己是运行在1环上，而不是通常OS的0环，有效的避免了虚拟化的执行冲突问题。
  - 超级调用Hypercall的机制使用，不仅使x86架构的指令虚拟化得以实现，也为后面的内存虚拟化和I/O虚拟化提供了新的思路和方法。
- 半虚拟化从另外一个角度解决了敏感指令的问题：与其千方百计去捕获这些指令，不如直接不用这些指令

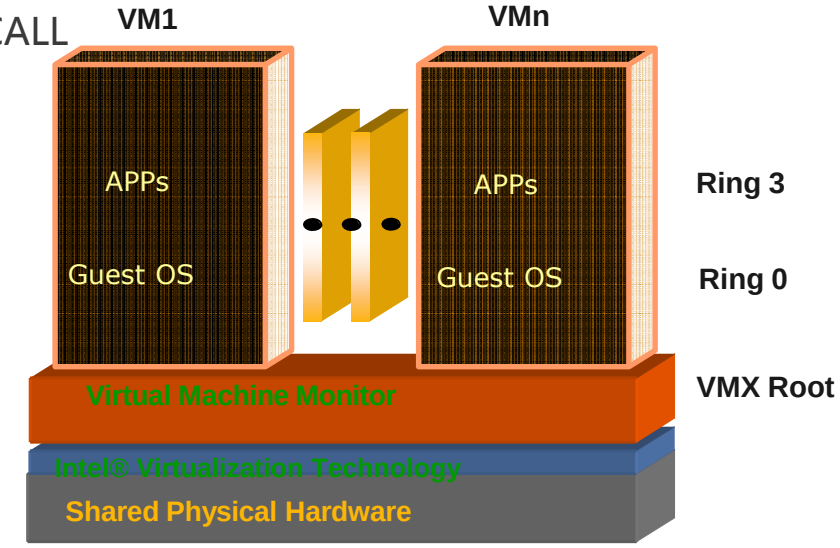


CPU半虚拟化优缺点

- 优点：
  - 相比全虚拟化，由于避免了trap捕获过程和二进制翻译 ，性能大幅提升
- 缺点：
  - 需要对Guest OS的系统内核的深度修改，面对各种版本系统，工作量非常大
  - 由于Windows不开源，没办法进行修改，因此不支持Windows
  - Guest OS将与Hypervisor强耦合，增加了系统维护成本，降低了可移植性

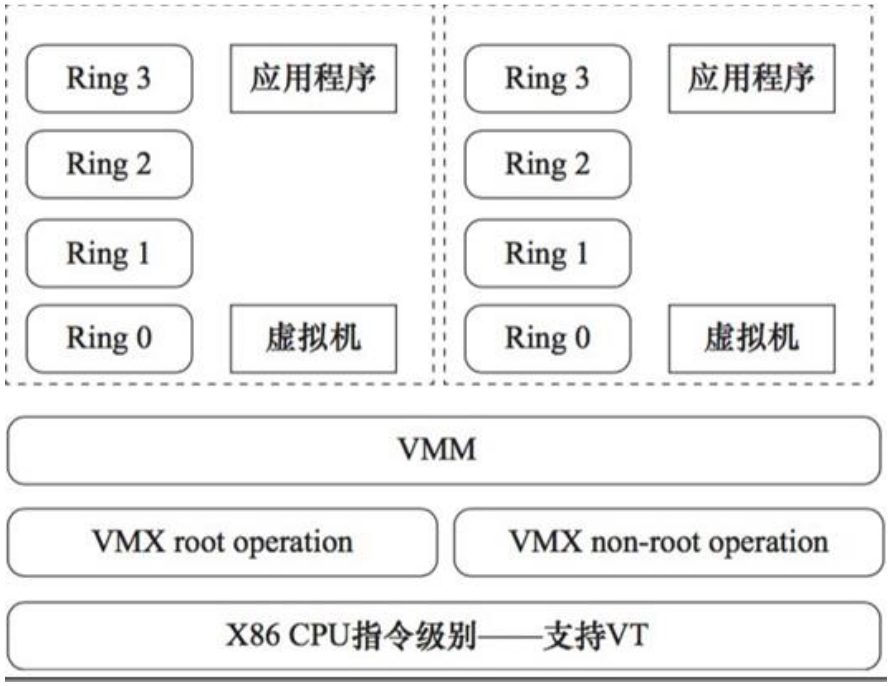
硬件辅助CPU虚拟化

- **硬件辅助虚拟化**：针对敏感指令为CPU添加了一个工作模式VMX (Virtual Machine eXtension) ，使从硬件上支持“陷入-模拟”方式的虚拟化
- 以Intel VT-x为例，VMX 提供了两类操作模式：根操作模式(root)和非根操作模式(non-root)。Hypervisor和客户软件分别运行于root和non-root操作模式。
  - 两种操作模式都支持 Ring 0 ~ Ring 3 特权级，因此 Hypervisor 和 Guest OS 都可以自由选择它们所期望的运行级别，同时Hypervisor拥有最高权限（所谓的特权级-1 ）
  - Root操作模式下软件的行为与在没有 VT-x 技术的CPU上的行为基本一致。而non-root操作模式则有所不同，最大区别是VMX提供了一套敏感指令，当Guest OS运行这些指令或遇到某些事件时，发生 VM Exit。
  - VM Exit分两种情况
    - 条件退出：即遇到某事件的时候，根据 VMCS 的控制执行域的设置来决定是否退出
    - 无条件退出：遇到某事件或直接调用VMCALL

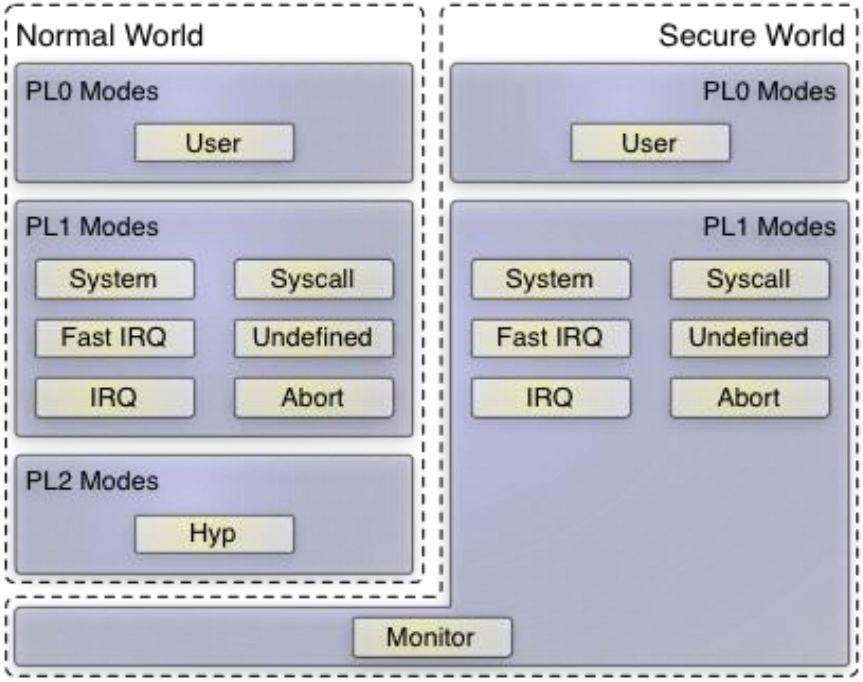


硬件辅助虚拟化

- 在x86架构中，Intel提出了VMX
  - Root和non-root是平行的
  - Non-root在root受控模式下运行



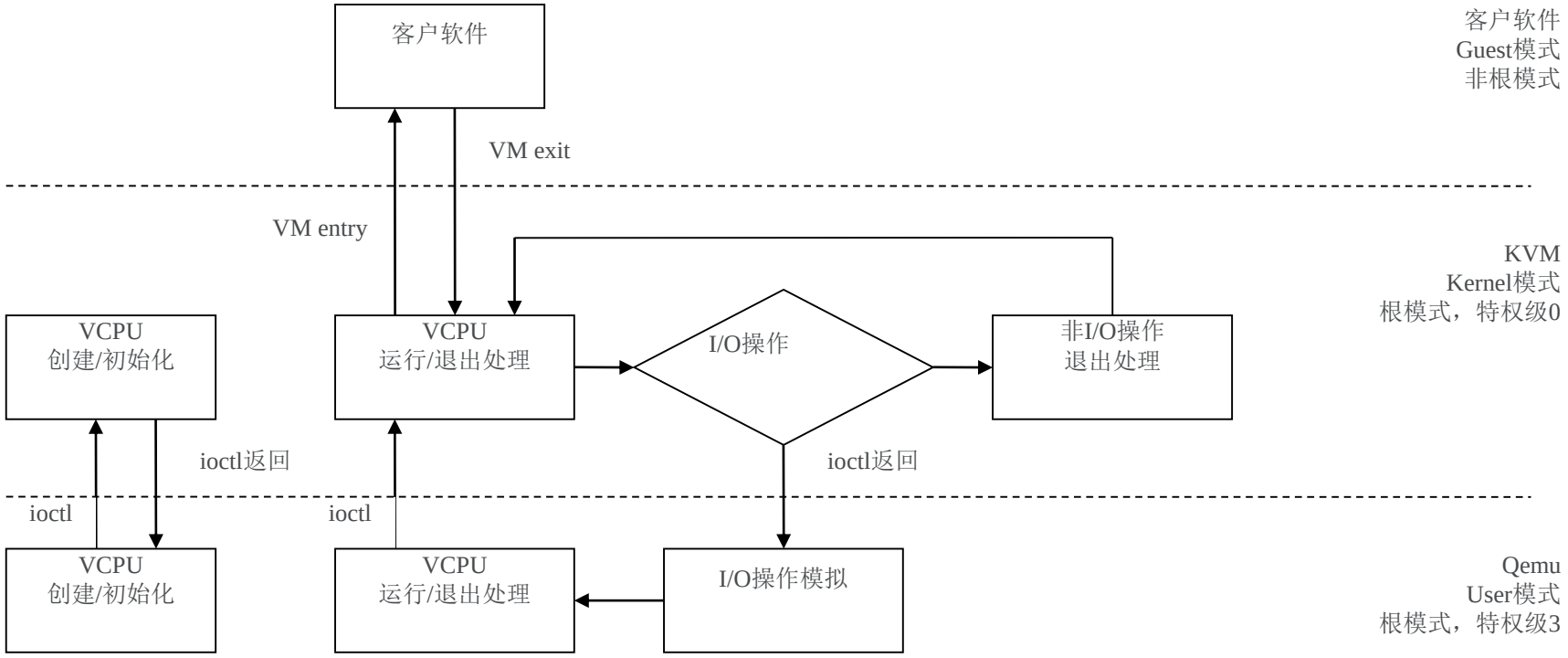
- 在arm架构中，提出了虚拟化扩展
  - 在原有运行级之下引入了新特权层
  - 不同模式之间是并列的



ARM的TrustZone与x86的VMX更为相似。在TrsutZone的安全世界中也可以通过设置寄存器的值捕获普通世界的事件；安全世界拥有比普通世界更高的权限，对硬件资源具有控制权

硬件辅助CPU虚拟化优缺点

- 优点：实现了高性能的全虚拟化，能够支持多种未修改的Guest OS直接运行，使得Hypervisor的设计得到极大简化。Hypervisor能够能够按通用标准进行编写，减少了Hypervisor运行时的开销
- 缺点：由于还是通过trap解决，导致性能上可能还比不上半虚拟化



CPU虚拟化总结

全虚拟化	半虚拟化	硬件辅助虚拟化
主要采用优先级压缩和二进制代码翻译技术这两个技术。优先级压缩能让Hypervisor和Guest运行在不同的特权级下，对X86架构而言，就是Hypervisor运行在特权级最高Ring 0下，Guest的内核代码运行在Ring 1下，Guest的应用代码运行在Ring 3下。通过这种方式能让Hypervisor截获一部分在Guest上执行的特权指令，并对其进行虚拟化。但是有一些对虚拟化不友好的指令则需要二进制代码翻译来处理， 它通过扫描并修改Guest的二进制代码来将那些难以虚拟化的指令转化为支持虚拟化的指令。	其通过修改Guest OS的代码，使其将那些和特权指令相关的操作都转换会发给Hypervisor的Hypercall，而且Hypercall支持批处理和异步这两种优化方式，使得通过超级调用能得到近似于物理机的速度	主要有Intel的VT-x和AMD的AMD-V这两种技术，都是通过引入新的指令和运行模式，来让Hypervisor和 Guest OS能分别运行在其合适的模式下。在实现方面，VT-x支持两种处理器工作方式：第一种称为Root模式（Operation），Hypervisor运行于此模式，用于处理特殊指令，另一种称为Non-Root模式（Operation），Guest OS运行于此模式，当在Non-Root模式Guest执行到特殊指令的时候，系统会切换到运行于Root模式Hypervisor，让Hypervisor来处理这个特殊指令。

内存虚拟化

内存虚拟化-1

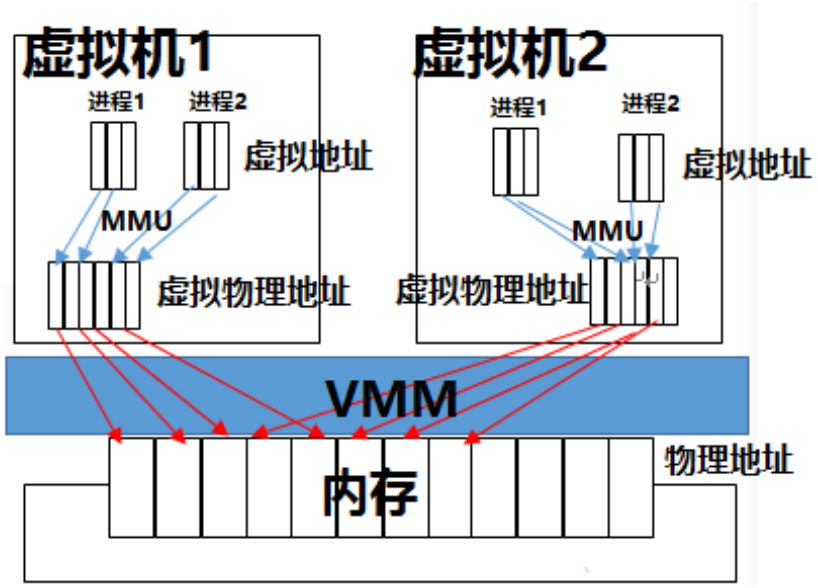
- 内存虚拟化技术把物理机的真实物理内存统一管理，包装成多个VM所见到的“物理内存”，使得每个VM拥有自己独立的物理内存空间
- 为了实现内存虚拟化，让客户机使用一个隔离的、从0开始且具有连续的内存空间，虚拟机监视器引入一层新的地址空间，即客户机物理地址空间(guest physical address space)。它并不是真正的物理地址空间，不能直接发送到系统总线上，它只是宿主机虚拟地址空间在客户机地址空间的一个映射

GVA: guest virtual address

GPA: guest physical address

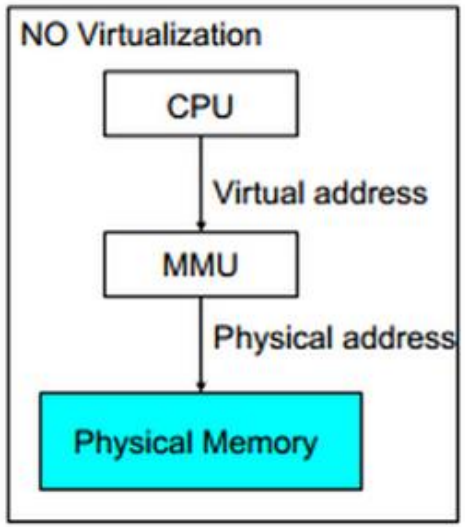
HVA: host virtual address

HPA: host physical address

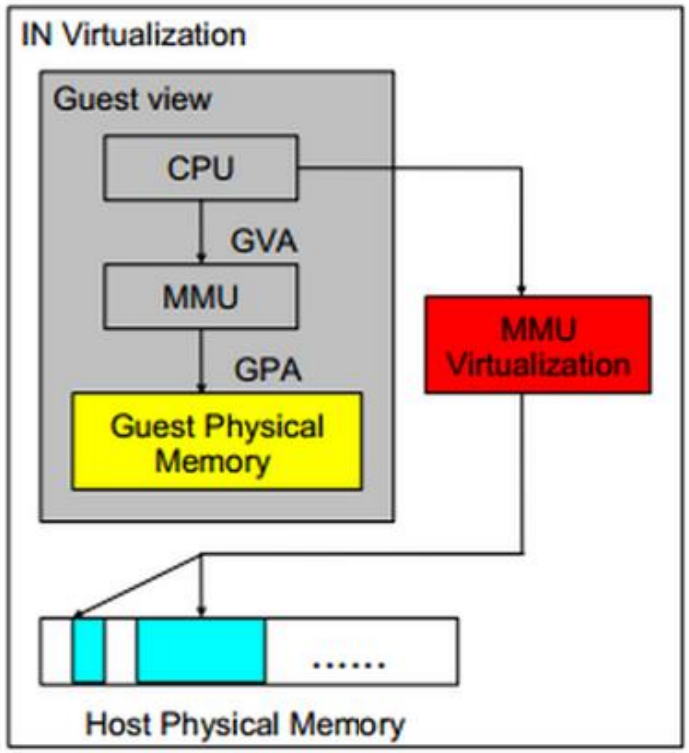


内存虚拟化-2

- 简单来说内存虚拟化是一个结构映射问题，需要进行两次地址翻译
  - Guest OS维护VM中进程所使用的GVA (guest virtual address) → GPA (guest physical address)的映射：GPA=addr_trans1(GVA)
  - Hypervisor 维护GPA → HPA (host physical address)之间的映射：HPA=addr_trans2(GPA)
  - VM所使用的GVA到CPU可以执行的HPA需要经常两次转换
    - HP

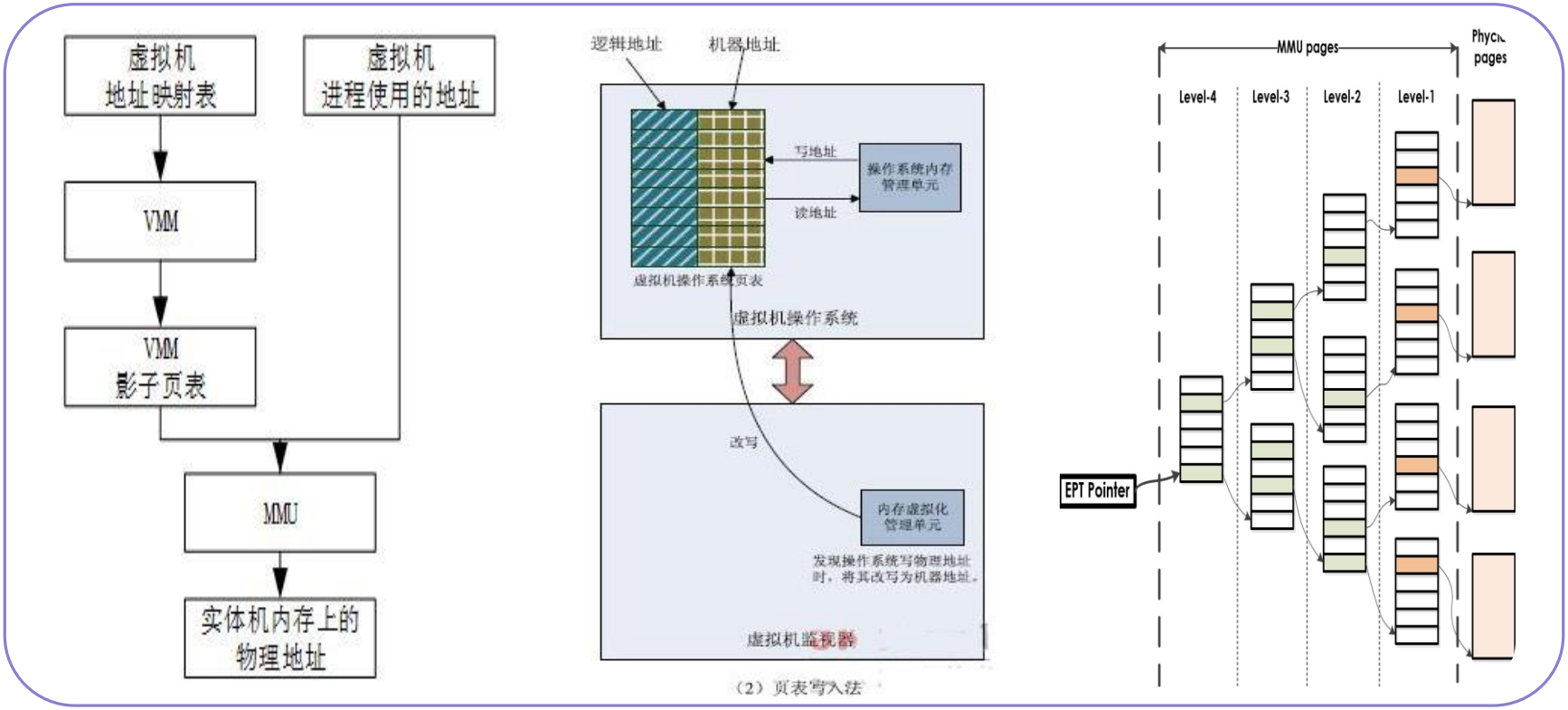


GVA: guest virtual address
GPA: guest physical address



内存虚拟化-3

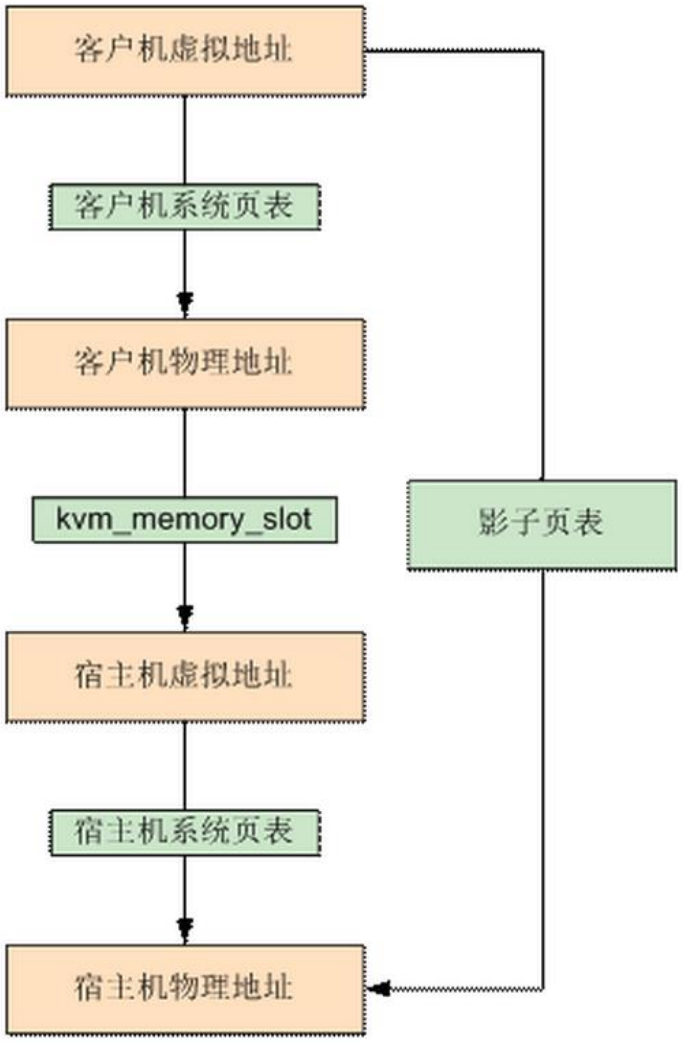
- 内存虚拟化在某种程度上依赖于cpu 虚拟化。实现内存虚拟化，最主要的是实现客户机虚拟地址到宿主机物理地址之间的转换。映射关系由Hypervisor负责，其实现方法有如下3种：



影子页表法　　页表写入法　　EPT/NTP/S2PT

影子页表

- 第一代的 Intel-VT 技术，并没有提供硬件支持的内存虚拟化技术。内存虚拟化一般采用影子页表(shadow page tables, SPT)。SPT是纯软件的内存虚拟化方案，Guest OS维护进程页表。Hypervisor为Guest OS中的每一个进程都维护一组影子页表与其相对应，负责GVA到HPA之间的直接转换。
- 为使SPT工作，Hypervisor需对MMU进行虚拟
  - Guest OS所看到的是虚拟MMU
  - Guest OS所维护的页表只能被客户载入到虚拟MMU中，不能被物理MMU使用
  - 真正被加载到物理MMU中的页表是影子页表



影子页表优缺点

- 影子页表简化了翻译，不需要GVA → GPA → HPA的双层转换，SPT降低了访存开销
- 需要为Guest VM中的每个进程维护一套相应的SPT，增加了内存开销
- 因为Guest OS在读写CR3、执行INVLPG指令或客户页表不完整等情况下均会导致VM exit，这导致了内存虚拟化效率很低
- Guest OS页表和SPT的同步也实现复杂，主要方式有两种。1）Hypervisor 将客户机使用的页表的物理地址区域设置为只读，当其更新自己的页表时，发生 VM Exit，这样 Hypervisor 则可保证两个页表的一致性。2）如上页INVLPG那样，将SPT中相应的页表项设为不存在

直接写入法

- 直接写入模式，Guest OS使用自己的页表直接访问真实物理内存，这就需要Xen改写Guest OS内核对页表的访问及TLB操作
- 其基本原理是：当Guest OS创建一个新的页表时，会从它所维护的空闲内存中分配一个页面，并向Hypervisor注册该页面，Hypervisor会剥夺Guest OS对该页表的写权限，之后Guest OS对该页表的写操作都会陷入到Hypervisor加以验证和转换。 Hypervisor会检查页表中的每一项，确保他们只映射了属于该VM的物理页，而且不得包含对页表页面的可写映射。
- Hypervisor会根据自己所维护的映射关系，将页表项中的物理地址替换为相应的机器地址，最后再把修改过的页表载入MMU。如此，MMU就可以根据修改过页表直接完成虚拟地址到机器地址的转换。

直接写入法优劣

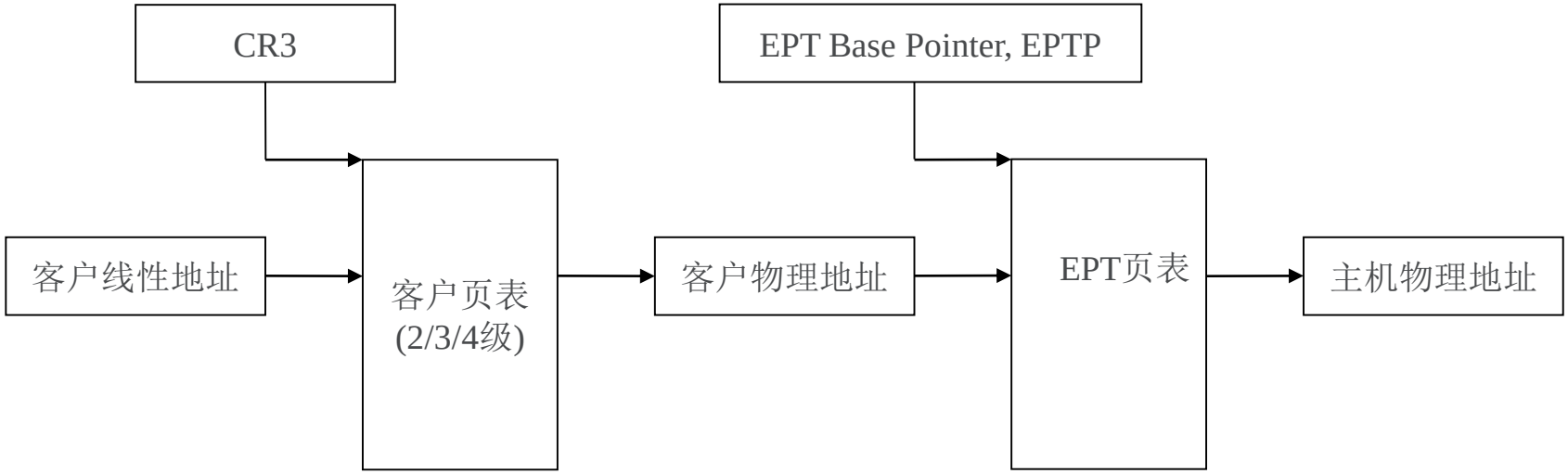
- 优：
  - 直接模式用在半虚拟化中， Guest OS维护的页表对MMU 可见，地址转换的效率很高；
- 劣：
  - 需要修改Guest OS，以便方便的提供内存管理必须经过修改；
  - 对于闭源的操作系统不可用

硬件扩展页表-VPID

- 为了解决影子页表的低效，第二代的 Intel-VT 技术，提供了硬件支持的内存虚拟化技术： VPID 和 Extended Page Table (EPT)
  - VPID 是一种硬件层对 TLB 管理的优化。VPID通过对每个TLB项上增加一个标志，用于唯一标识并区分来自不同的VCPU，因此避免了在每次VM-Entry/VM-Exit时使全部的TLB失效，提升了性能。
    - VPID是一个16位的域，每个TLB表项与一个VPID相关联。当进行虚拟地址到物理地址转换的时候，只有一个TLB表项对应的VPID与当前正在运行的虚拟机的VCPU的VPID相同的时候，才可以用该TLB表项进行地址转换
    - 为每个 VMCS 分配一个 独立的、区别于其它VCPU的VPID ，且保证这个 VPID 非 0 (VPID=0被用于Hypervisor)
    - 在 VMCS 中将 Enable VPID 位置 1

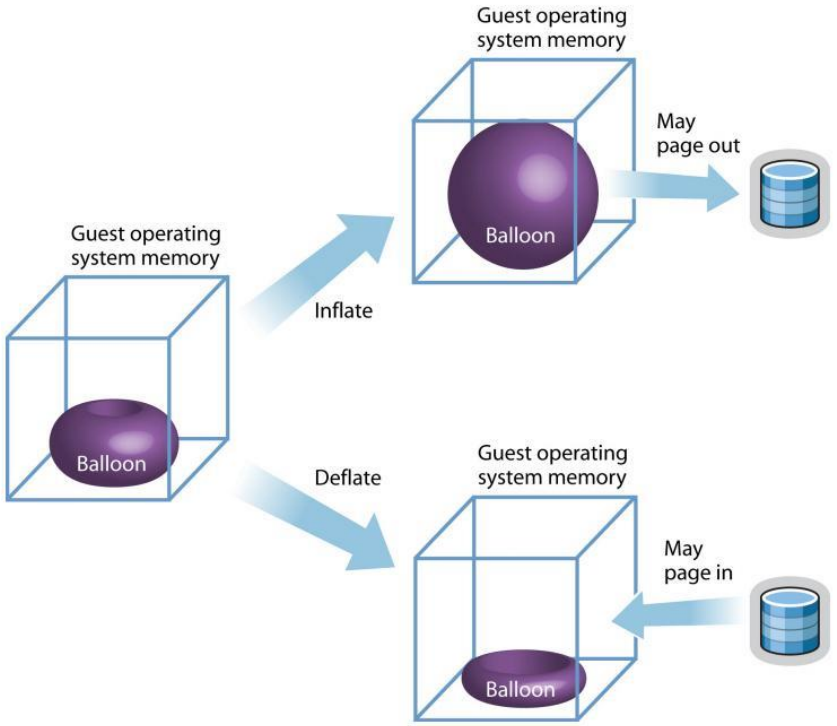
硬件扩展页表-EPT

- EPT引入了一套页表结构，定义了GPA到HPA之间的映射关系，直接在MMU上实现GVA → GPA → HPA的两次转换，降低了内存虚拟化的难度，也提高了性能。所有的客户物理地址须经由EPT页表转换后访存
  - 每个物理 Core都有两个寄存器，分别指向Guest OS进程自身的页表和per-VM的EPT的根目录地址。
  - 只有在 VMX non-root 操作模式下，EPT 页表才会被启用，参与地址转换
  - 有了 EPT ，客户机操作系统设置 CR3 寄存器来切换页表的时候，无需产生 VM exit；Hypervisor 也不用写保护它管理的页表



内存虚拟化—气球驱动

- 在物理内存的管理中，Hypervisor引入了气球驱动模型来调节分配给各虚拟机的物理内存。
  - 气球驱动作为驱动运行在Guest OS中，Guest OS通过该驱动与Hypervisor通信。当虚拟机需要更多内存时，将通过气球驱动向Xen提交内存申请，Hypervisor可向气球驱动减压以便将气球驱动所占用的部分空闲内存或通过气球驱动从其它虚拟机回收的内存分配给提交申请的虚拟机。
  - 如果Hypervisor的可用空余内存过低，Hypervisor可向气球驱动加压使气球膨胀，Guest OS将回收页面、释放内存以便给本地气球分配足够的内存空间，然后气球驱动将分配到的页面传给Hyp



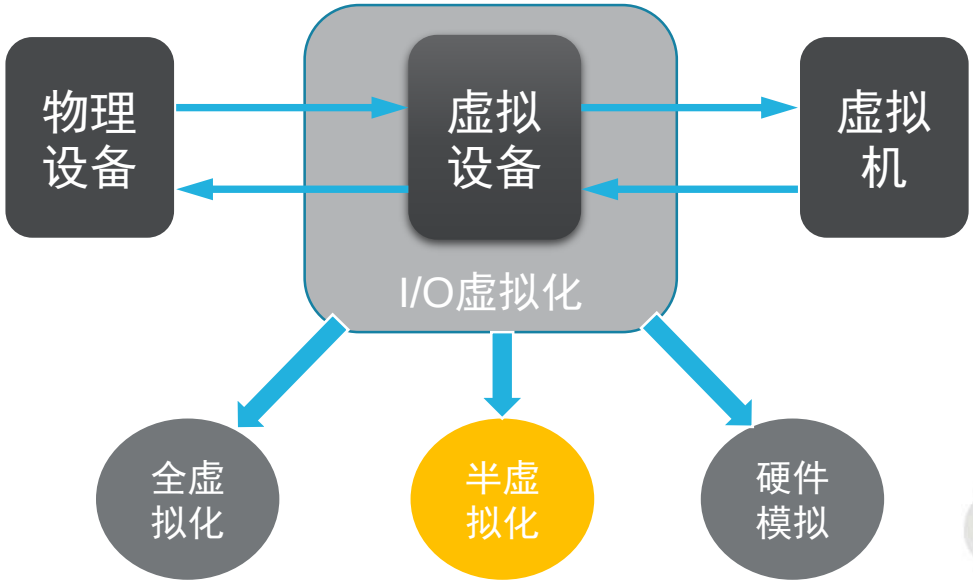
内存虚拟化方式

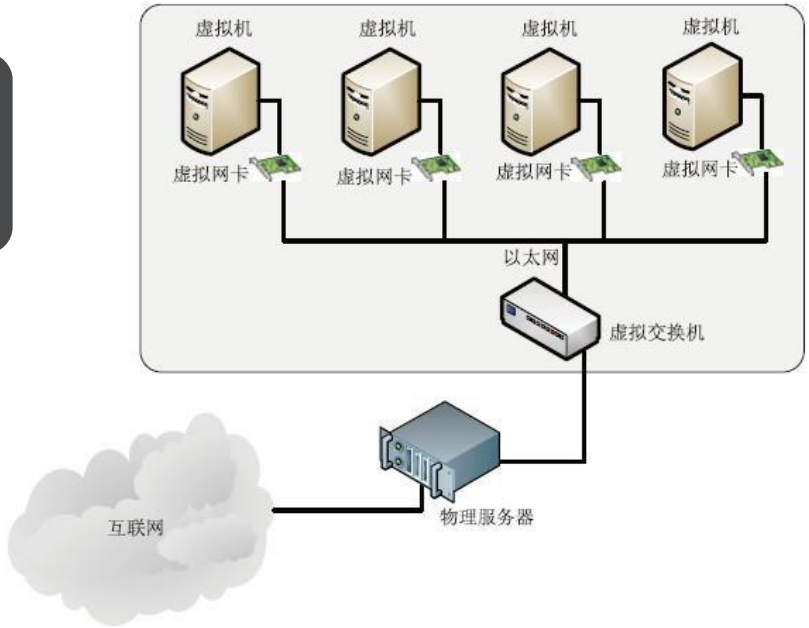
全虚拟化	半虚拟化	硬件辅助虚拟化
影子页表就是为每个Guest VM都维护一个“影子页表”，在这个表中写入虚拟化之后的内存地址映射关系，而Guest OS的页表则无需变动，最后，Hypervisor将影子页表交给MMU进行地址转换。	页表写入法，当Guest OS创建一个新的页表时，其会向Hypervisor注册该页表，之后在Guest运行的时候，Hypervisor将不断地管理和维护这个表，使Guest上面的程序能直接访问到合适的地址。	EPT扩展页表，EPT通过使用硬件技术，使其能在原有的页表的基础上，增加了一个EPT页表通过这个页表能够将Guest的物理地址直接翻译 为主机的物理地址，从而减低整个内存虚拟化所需的Cost。

I/O虚拟化

I/O虚拟化

- I/O虚拟化是服务器虚拟化技术的重要组成部分，在服务器虚拟化技术领域，计算虚拟化（如CPU和内存虚拟化）已经日趋成熟，但是，I/O虚拟化技术的发展相对比较滞后。
- I/O设备虚拟化技术把真实的设备统一管理起来，包装成多个虚拟设备给若干个虚拟机使用，响应每个虚拟机的设备访问请求和I/O请求。I/O虚拟化由Hypervisor进行管理。





I/O虚拟化

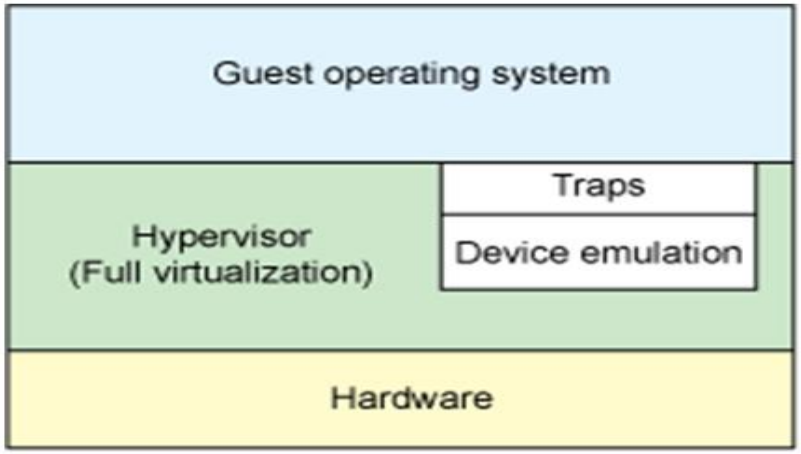
- 要对I/O设备进行虚拟化，Hypervisor需要监视客户虚拟机的行为，截获客户虚拟机的I/O操作，对其加以分析解释，然后发送给设备驱动程序，由设备驱动程序来完成对设备的操作。
- 当前，主流的I/O虚拟化技术有三种：软件模拟、半虚拟化和硬件模拟虚拟化，而硬件虚拟化包括设备直通和SR-IOV。
  - 仿真：Hypervisor为Guest OS模拟一个设备，具有很好的兼容性，但有性能限制
  - Virtio：类似仿真，但Hypervisor提供的是针对客户软件设计的设备接口
  - 直接分配：Hypervisor直接将设备分配给虚拟机，但需要主板支持虚拟化功能
  - I/O 设备分享：一个 I/O 设备支持多个接口，每个接口分配给一个虚拟机

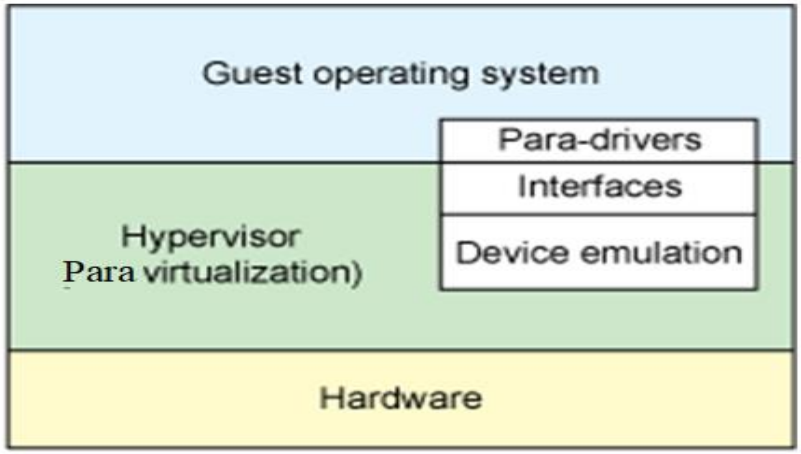
I/O全虚拟化

- 即软件精确模拟与物理设备完全一样的接口，Guest OS驱动无须修改就能驱动这个虚拟设备。优点是没有额外的硬件开销，可重用现有驱动程序，缺点是为完成一次操作要涉及到多个寄存器的操作，使得 Hypervisor 要截获每个寄存器访问并进行相应的模拟，导致多次上下文切换，性能较低。
- 当虚拟机之中的应用程序需要对设备进行操作时：
  - 它首先通过系统调用把设备操作请求发送给Guest OS的内核层；
  - Guest OS的内核层对此设备操作请求进行预处理后（分析、解释、格式的转换等），然后发送给设备驱动程序来完成此操作；
  - 设备驱动程序会按照发送过来的设备操作请求，逐条的向虚拟设备发出I/O 指令，来完成整个的设备操作。
  - 当Guest OS发出I/O 指令时，Hypervisor会捕捉到这些指令，并对这些指令进行分析和解释，然后对相应的真实物理设备进行相应的操作。
  - 全虚拟化I/O的好处是无需修改Guest OS，但因Guest OS每执行一次I/O 指令都会被截获并陷入Hypervisor，因此需要大量的上下文切换，并需要执行更多的代码，严重影响了系统的I/O 性能。

半虚拟化-VirtIO

- 为了改善全虚拟化性能瓶颈，提出了Virtio半虚拟化技术
  - Hypervisor提供一个简化的驱动程序（后端, Back-End），Guest OS中的驱动程序为前端 (Front-End, FE)，前端驱动将来自其它模块的请求通过与Guest OS间的特殊通信机制直接发送给Guest OS的后端驱动，后端驱动在处理完请求后再发回通知给前端。
  - 优点是由于基于事务的通信机制，能在很大程度上减少上下文切换开销，没有额外的硬件开销，缺点是需要Hypervisor实现前端驱动，后端驱动可能成为瓶颈。
- Guest OS知道自己运行在 hypervisor 之上，并包含了充当前端的驱动程序。Hypervisor 为特定的设备模拟实现后端驱动程序。通过在这些前端和后端驱动程序中的 virtio，为开发模拟设备提供标准化接口，实现跨平台和高效率。





I/O硬件虚拟化

- 即直接将物理设备分配给某个Guest OS，由Guest OS直接访问I/O设备（不经 Hypervisor），目前与此相关的技术有IOMMU（Intel VT-d，SR-IOV ），旨在建立高效的I/O虚拟化通道。优点是减少了性能开销，缺点是硬件需要支持虚拟化。
  - 基于PCI总线实现的IO虚拟化技术VT-d
    - Pass-through透传技术
  - 基于网络的虚拟化技术VT-c
    - SR-IOV技术

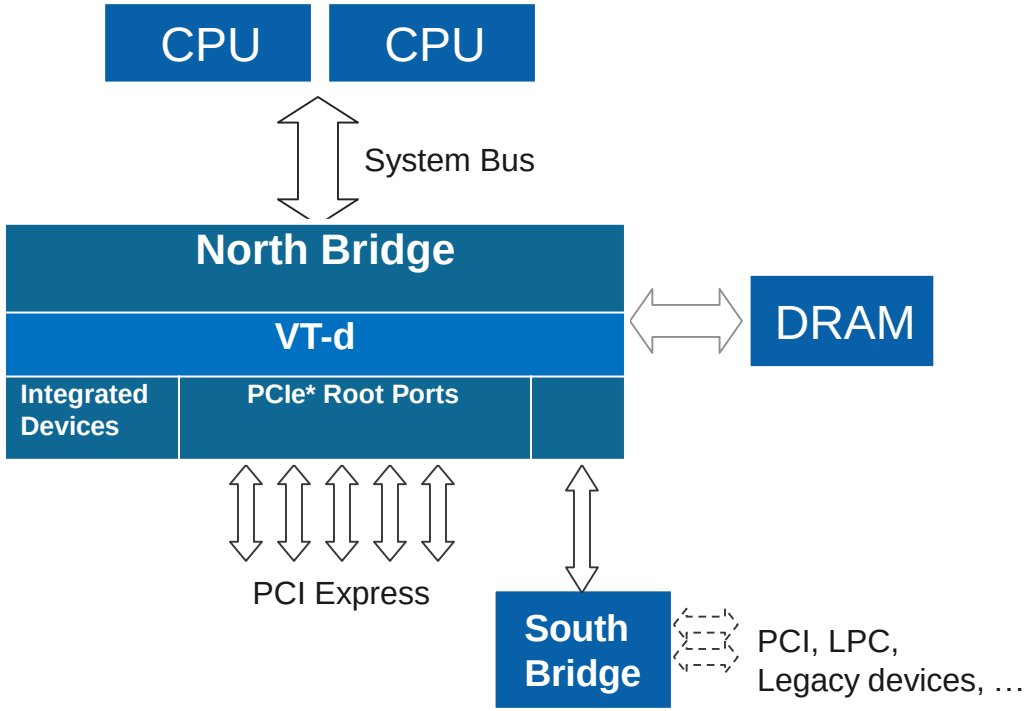
VT-d

- 基于PCI总线实现的IO虚拟化技术VT-d
  - 允许将宿主机中的物理 PCI 设备直接分配给客户机使用。支持客户机以独占方式访问这个宿主机的 PCI/PCI-E 设备。通过硬件支持的 VT-d 技术将设备分给客户机后，在客户机看来，设备是物理上连接在PCI或者PCI-E总线上的。
  - 几乎所有的 PCI 和 PCI-E 设备都支持直接分配，除了显卡以外。
    - SATA 或者 SAS 等类型的硬盘的控制器都是直接接入到 PCI 或者 PCI-E 总线的，可以将硬盘作为普通的PCI设备直接分配个客户机。
    - 当分配硬盘时，实际上将其控制器作为一个整体分配到客户机中。

VT-d

- 网卡直通支持虚拟机绕过Hypervisor层，直接访问物理I/O设备，具有最高的性能，在同一时刻，物理I/O设备只能被一个虚拟机独享。
  - 通过硬件层的映射使得虚拟机内的IO请求（IO用到的资源有中断、DMA）直接映射到实际硬件上。如果没有VT-d，则需要软件来维护这个映射表。

- VT-d作为芯片组的一部分
  - 提供了Guest OS直接访问真实硬件的机制，极大的减少了服务器CPU的负载，改善I/O 设备在虚拟化环境中的性能，并帮助用户提高系统的安全性和可靠性。



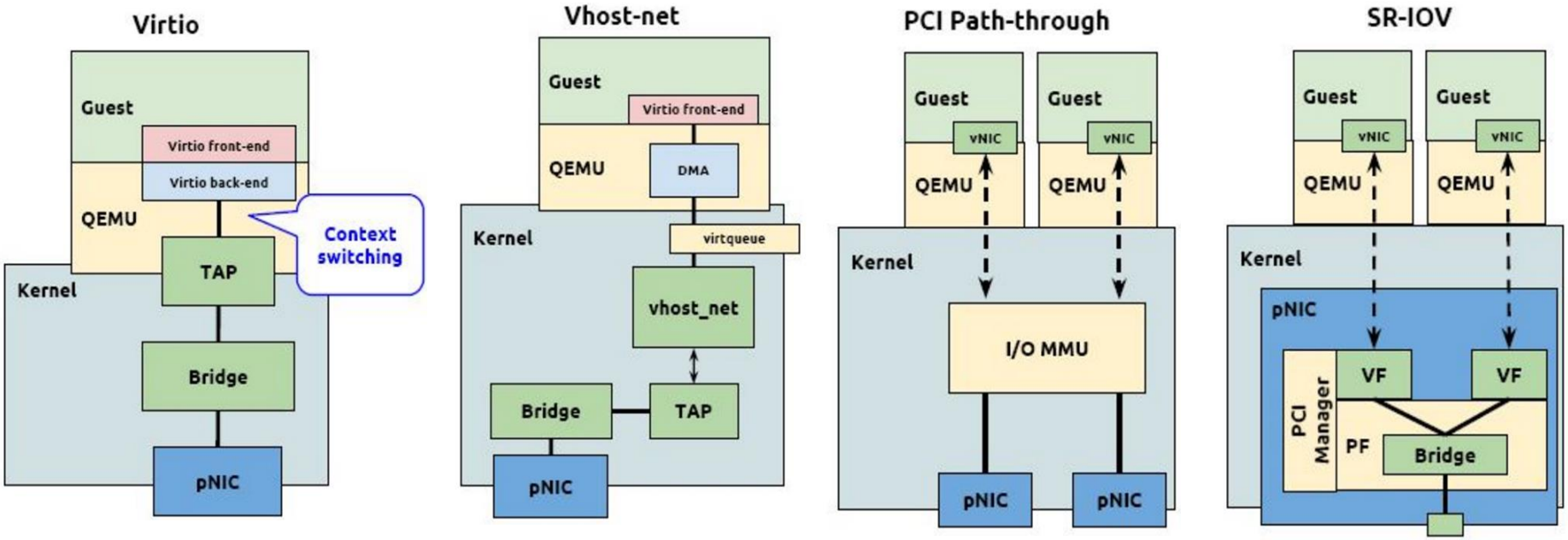
SR-IOV

- VT-d 的性能非常好，但是它的物理设备只能分配给一个客户机使用。
- 为实现多个虚机共享一个物理设备，并且达到直接分配的目的。PCI-SIG 组织发布了 SR-IOV （ Single Root I/O Virtualization ）规范，它定义了一个标准化的机制用以生地实现多个虚拟机共享一个设备。
- Intel在2007年提出了虚拟化网络I/O（VT-c）的SR-IOV硬件技术方案
  - SRIOV属于VT-d技术的一个分支，要实现SRIOV功能，前提条件就是网卡支持SRIOV，且主板支持VT-d技术
  - SR-IOV 使得一个单一的功能单元（一个以太网端口）能看起来像多个独立的物理设备。目前 SR-IOV 只广泛地应用网卡上。
  - 该技术不仅能够继承网卡直通的高性能优势，而且同时支持物理I/O设备的跨虚拟机共享，具有较好的应用前景。

SRIOV优劣

优势	不足
1. 真正实现设备共享（多个虚拟机共享一个 SR-IOV 设备的物理端口）	1. 对设备有依赖，目前只有部分设备支持 SR-IOV。RedHat Linux 只是测试了 Intel 的几款高端网卡。
2. 接近硬件设备的原生性能	2. 支持的VF具有数量限制；使用时配置较繁琐
3. 相比 VT-d，SR-IOV可以使用更少的设备来支持更多的客户机，可以提高数据中心的空间利用率。	3. 使用SR-IOV时不方便动态迁移客户机。这是因为这时候虚机直接使用主机上的物理设备，因此虚机的迁移（migiration）和保存（save）目前都不支持。这个在将来有可能被改变。

虚拟化方式的比较


Virtio
Guest
Virtio front-end
Virtio back-end
QEMU
TAP
Kernel
Bridge
pNIC
Context switching
Vhost-net
Guest
Virtio front-end
QEMU
DMA
Kernel
virtqueue
vhost_net
Bridge
TAP
pNIC
PCI Path-through
Guest
Guest
vNIC
vNIC
QEMU
QEMU
Kernel
I/O MMU
pNIC
pNIC
SR-IOV
Guest
Guest
vNIC
vNIC
QEMU
QEMU
Kernel
pNIC
PCI Manager
VF
VF
PF
Bridge

I/O虚拟化总结

- 在全虚拟化模式中，hypervisor 必须模拟设备硬件，它是在会话的最低级别进行模拟的（例如，网络驱动程序）。尽管在该抽象中模拟很干净，但它同时也是最低效、最复杂的。
- 在半虚拟化模式 中，客户操作系统和 hypervisor 能够共同合作，让模拟更加高效。半虚拟化方法的缺点是操作系统知道它被虚拟化，并且需要修改才能工作。
- 在硬件辅助虚拟化中，硬件资源直接分配给某个虚拟机使用，虚拟机可以直接对其进行访问。但是对于设备的使用效率则明显降低。此外，当前的硬件辅助虚拟化并不是所有的硬件设备都支持。

X86虚拟化技术总结

- CPU、内存和I/O虚拟化技术概括

	全虚拟化	半虚拟化	硬件辅助虚拟化
CPU虚拟化	二进制代码翻译	Hypercall	VT-x
内存虚拟化	影子页表	页表写入法	EPT
I/O虚拟化	模拟I/O设备	前端/后端架构	VT-d

Q & A