第五章云使能技术

--虚拟化与交互技术

§ 5.1 宽带网络和Internet架构
§ 5.2 数据中心技术
§ 5.3 虚拟化技术
§ 5.4 Web技术
§ 5.5 多租户技术
§ 5.6 服务技术
§ 5.7 云计算数据处理架构
§ 5.8 分布式文件系统
§ 5.9 分布式数据库
§ 5.10 MapReduce技术

§5.3虚拟化技术

虚拟化是指将物理IT资源转换为虚拟IT资源的过程。
大多数IT资源都能被虚拟化，包括：

- 服务器（server）——一个物理服务器可以抽象为一个虚拟服务器。
- 存储设备（storage）——一个物理存储设备可以抽象为一个虚拟存储设备或一个虚拟磁盘。
- 网络（network）——物理路由器和交换机可以抽象为逻辑网络，如VLAN。
- 电源（power）——一个物理UPS和电源分配单元可以抽象为通常意义上的虚拟UPS。

虚拟化技术的优势

- 硬件无关性
  - ✓ 直接在硬件上安装OS时，若硬件变化系统需重新配置
  - ✓ 虚拟化将硬件转换为基于软件的标准化版本，运行在虚拟化环境的操作系统基本不受底层硬件的影响
- 服务器整合
  - ✓ 虚拟化技术允许在一台物理服务器上创建多台虚拟服务器，可提高资源利用率，它是按需使用、资源池、可扩展性、可恢复性等云特性的基础
- 资源复制
  - ✓ 虚拟机的状态最终保存为文件的形式，通过简单操作文件可实现虚拟机的复制、迁移、快照

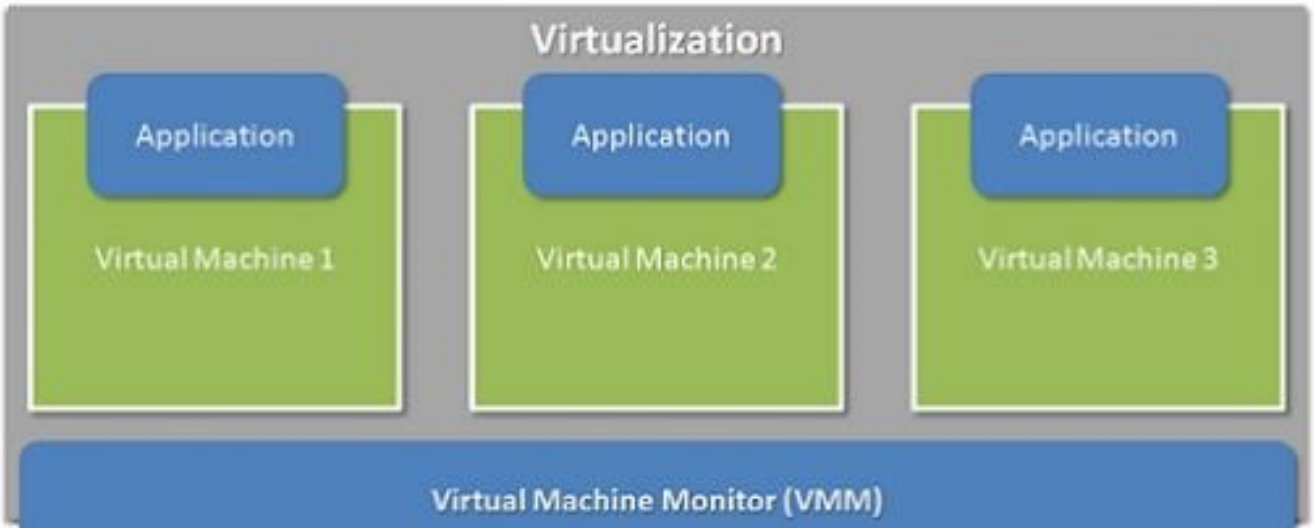
服务器虚拟化技术分类

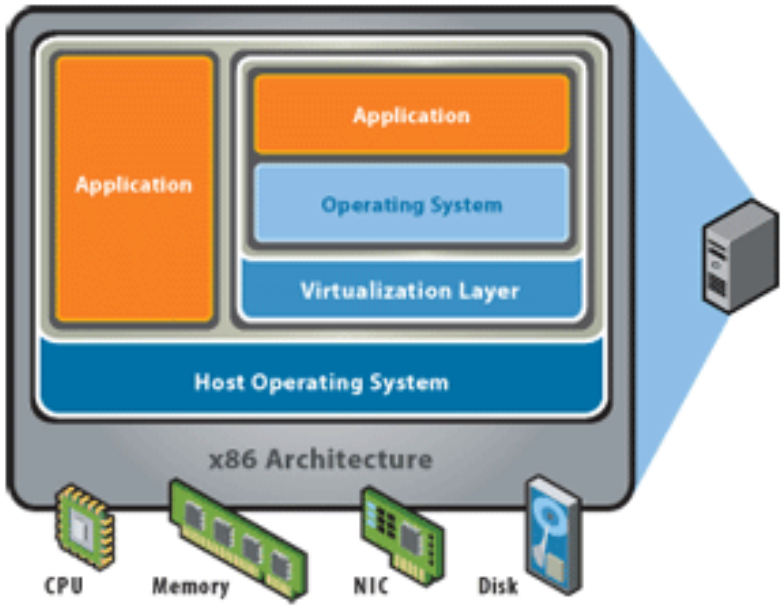
- 虚拟机（VM）
- 容器（container）
- 函数（Function）

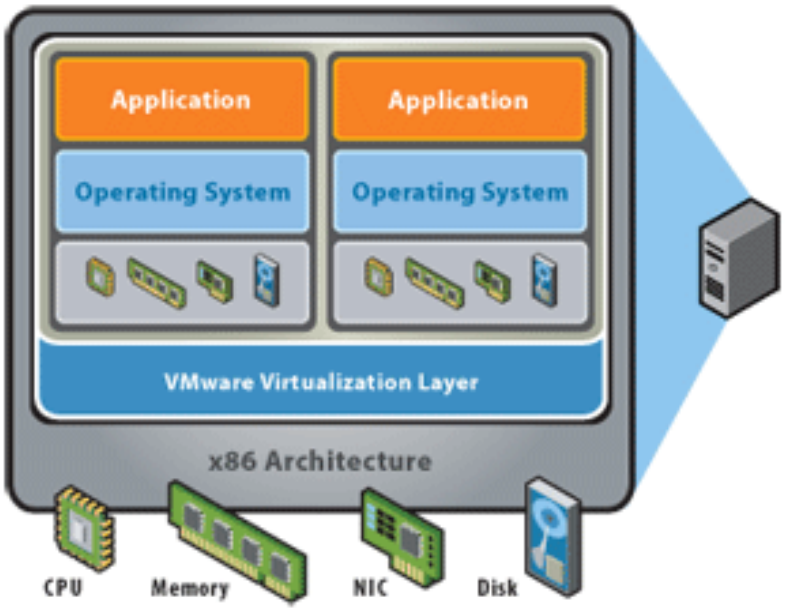
虚拟机

- 一台物理机→多台虚拟机（Virtual Machine, VM）
  - 虚拟CPU
  - 虚拟内存
  - 虚拟I/O
  - 独立软件
  - 数据隔离

虚拟机架构


Virtualization
Application
Application
Application
Virtual Machine 1
Virtual Machine 2
Virtual Machine 3
Virtual Machine Monitor (VMM)


Application
Application
Operating System
Virtualization Layer
Host Operating System
x86 Architecture
CPU
Memory
NIC
Disk


Application
Application
Operating System
Operating System
VMware Virtualization Layer
x86 Architecture
CPU
Memory
NIC
Disk

基于操作系统的虚拟化--寄生架构（Hosted）

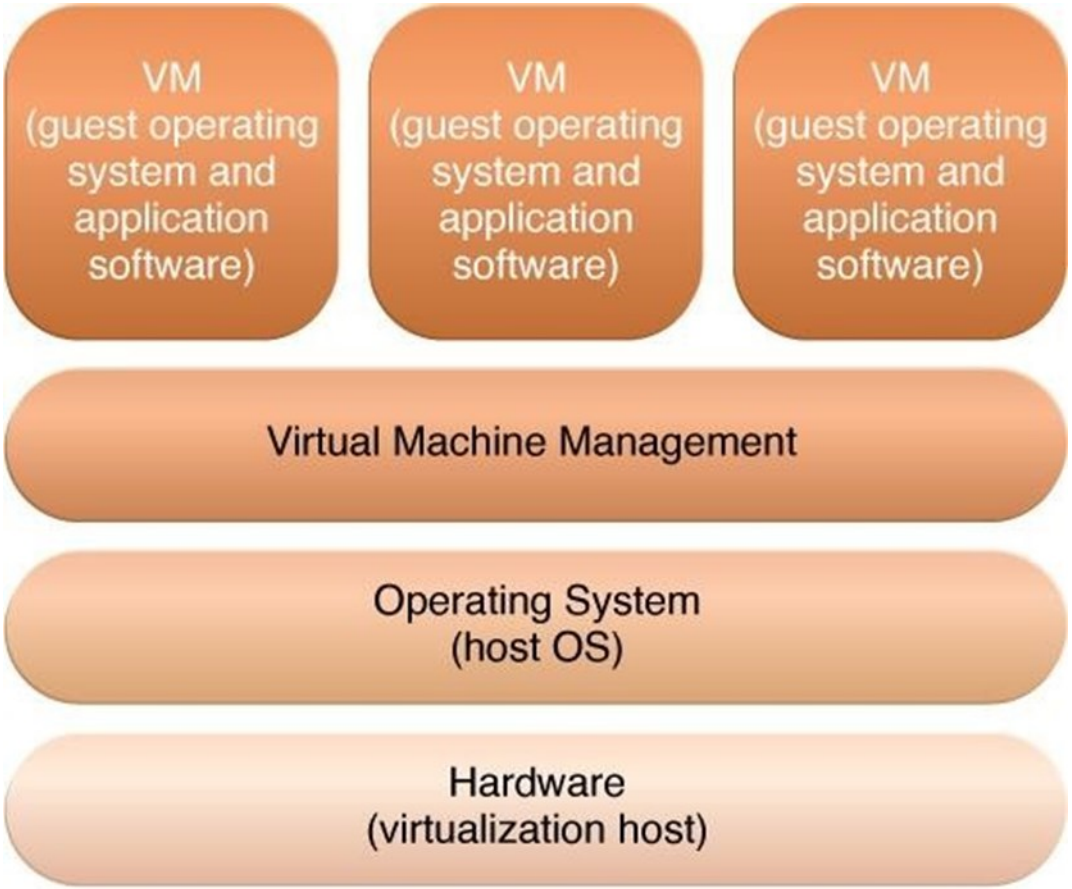


Figure 5.8 – 基于操作系统虚拟化的逻辑分层。其中，VM首先被安装在完整的宿主操作系统上，然后被用于产生虚拟机

- ✓ 宿主操作系统
  - ◆ 管理所有硬件资源
  - ◆ 虚拟机监控器为该操作系统上的一个应用程序
- ✓ 虚拟机监控器（VMM）
  - ◆ 模拟硬件的一些行为
- ✓ 客户操作系统
  - ◆ 运行用户的应用

基于硬件的虚拟化--裸金属架构（Bare-metal）

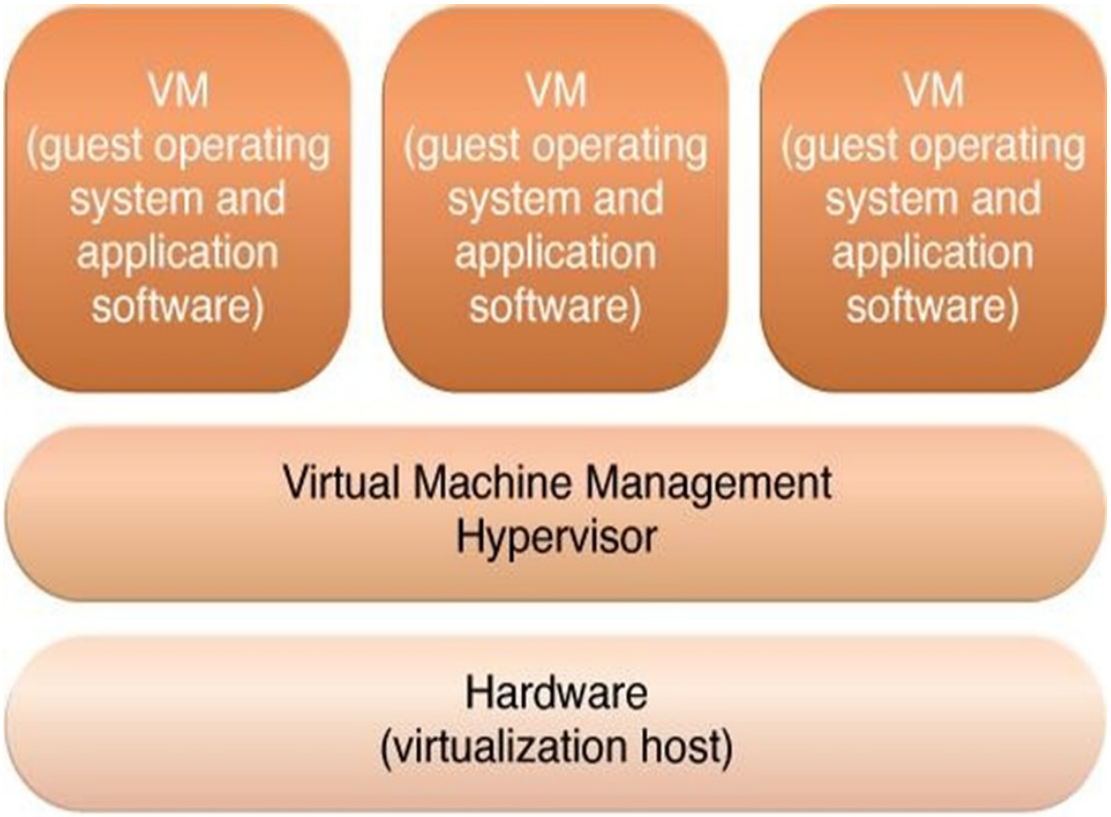
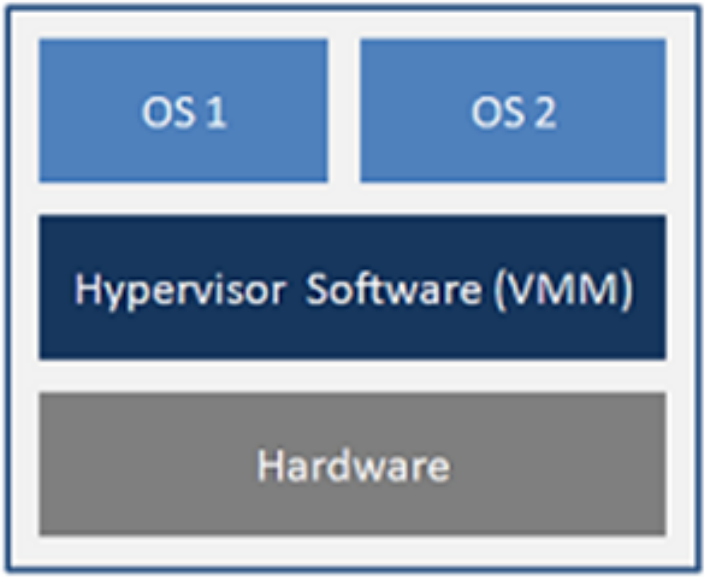


Figure 5.9 – 基于硬件虚拟化的逻辑分层，不再需要另一个宿主操作系统

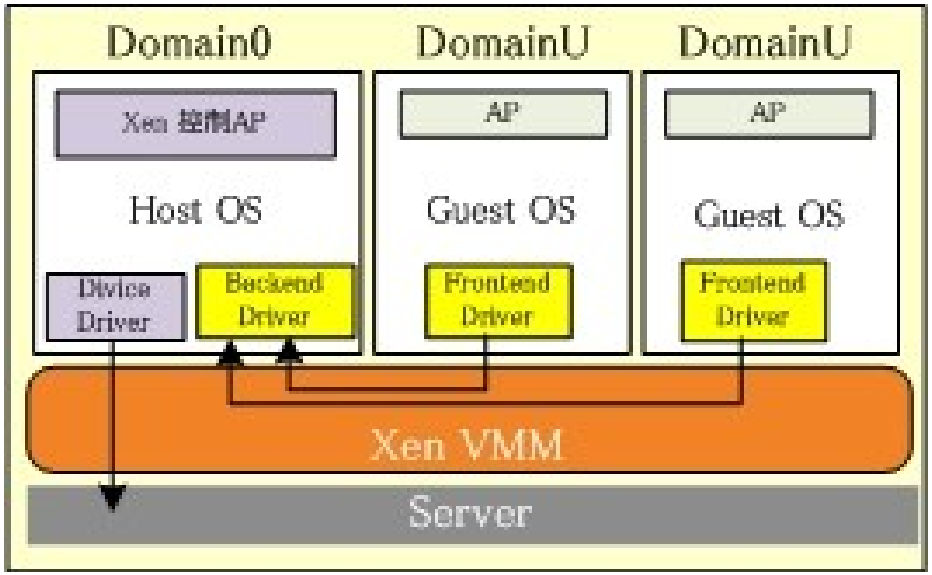
- ✓ 虚拟机监控器（VMM）
  - ◆ 一个实现核心功能的轻量级操作系统，可称Hypervisor
  - ◆ 接管客户操作系统中的一些特殊指令，如操作硬件的指令
- ✓ 客户操作系统
  - ◆ 运行用户的应用

虚拟机架构 – 裸金属架构（Bare-metal）

- VMM + (Host OS +) OS



Bare-Metal Architecture



Hosted vs. Bare-metal

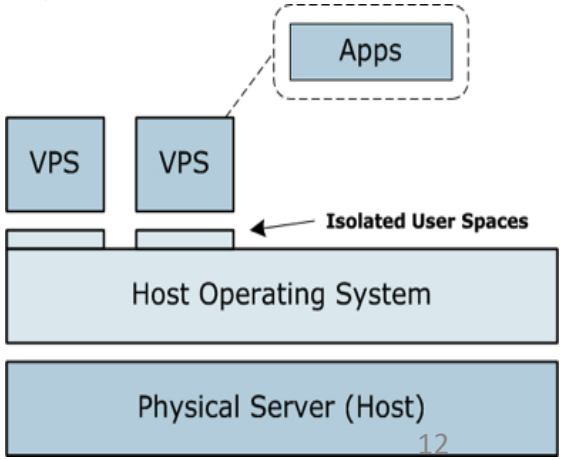
Bare-metal (Type I)	Hosted (Type II)
效率高 （不受Host OS影响、I/O优化）	
安全性高	
	使用方便
	功能丰富（e.g. 3D加速）
适于服务器系统	适于桌面系统

容器技术（Container）

- 传统虚拟化技术的缺陷
  - ✓ 每一个虚拟机都是一个完整的操作系统，当虚拟机数量增多时，操作系统本身的资源消耗较大
  - ✓ 开发环境和线上环境的通常存在区别，所以开发环境与线上环境之间无法达到很好的桥接，在部署上线应用时，需要花时间去处理环境不兼容的问题
- 容器技术
  - ✓ 把应用程序运行需要的环境整体打包，这个包成为容器
  - ✓ 容器作为进程运行在操作系统中，无需完整操作系统映像
  - ✓ 容器可以把开发环境及应用整个打包带走，打包好的容器可以在任何的环境下运行，这样就可以解决开发与上线环境不一致的问题了

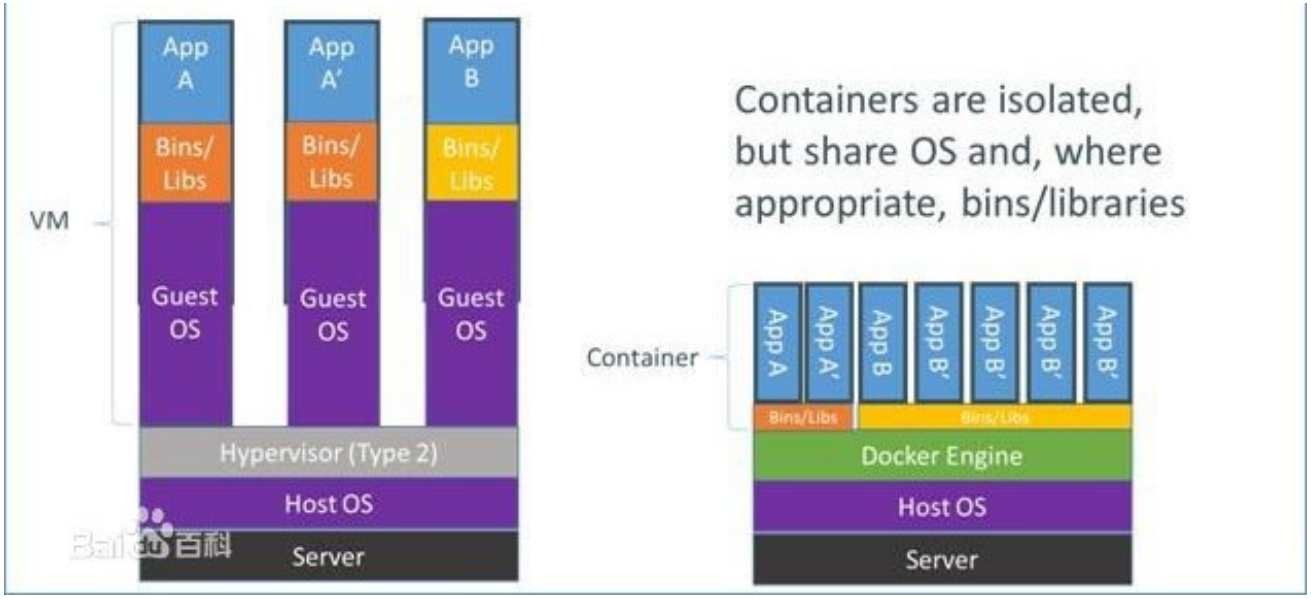
容器技术（Container）

- 容器在操作系统中的运行
  - ✓ 容器自带环境在操作系统中启动进程
  - ✓ 所有容器共用一个操作系统
  - ✓ 通过命名空间和进程组来提供隔离性
- Docker容器
  - ✓ 开源的应用容器引擎
  - ✓ 开发者可以打包他们的应用以及应用的依赖包，放到一个可移植的容器中
  - ✓ 然后发布到任意的机器上以实现虚拟化



容器虚拟化

- 优势
  - ✓ 轻量级虚拟化，没有独立的操作系统，额外开销很小
  - ✓ 每个主机可以支撑上千容器，且容器启动时间在毫秒级
  - ✓ 自带运行环境，开发过程中打包好的应用，在生产环境中能迅速部署
- 缺点
  - ✓ 隔离性、安全性较差
  - ✓ 多容器共享操作系统，不能随意修改操作系统相关的配置



容器虚拟化

- 微服务
  - ✓ 微服务是一种将应用分解成小的自治服务的软件架构
  - ✓ 每个服务被独立的开发、测试、部署
  - ✓ 服务提供方以集群的方式提供服务
- 微服务运行在容器中
  - ✓ 每个微服务依赖的环境打包到一个容器中
  - ✓ 通过容器管理工具如Kubernetes管理大量的容器（微服务）
- 优缺点分析
  - ✓ 优点：应用在迭代开发过程中可单独更新一个微服务
  - ✓ 挑战：将一个应用拆分成很多微服务并不容易，分离之后微服务的大小、微服务之间的通信、数据一致性等问题面临挑战

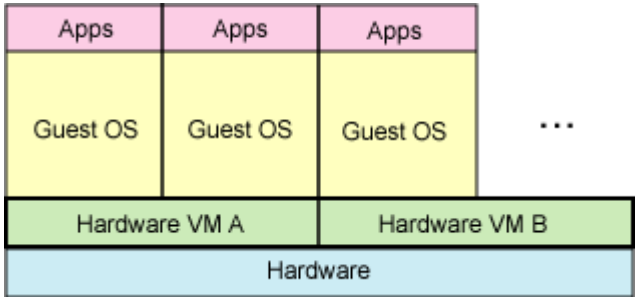
函数虚拟化

- 函数库虚拟化
  - ✓ 用一组库函数实现另一组库函数，称为函数库虚拟化
  - ✓ 应用场景：Linux的程序不能在Windows下运行，采用函数库虚拟化则可达到这一目的
- 典型案例
  - ✓ Wine：在Linux操作系统上运行Windows应用
  - ✓ Cygwin：在Windows操作系统上运行Linux应用

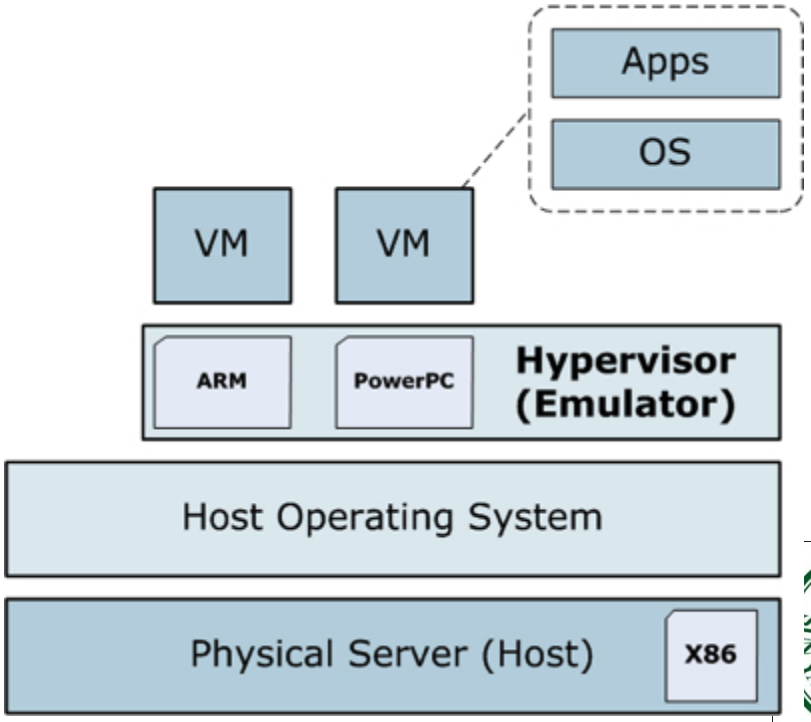
虚拟机技术

- 硬件仿真（**Emulation**）
- 全虚拟化（**Full Virtualization**）
- 半虚拟化（**Parairtulization**）
- 硬件辅助虚拟化（**Hardware Assisted Virtualization**）

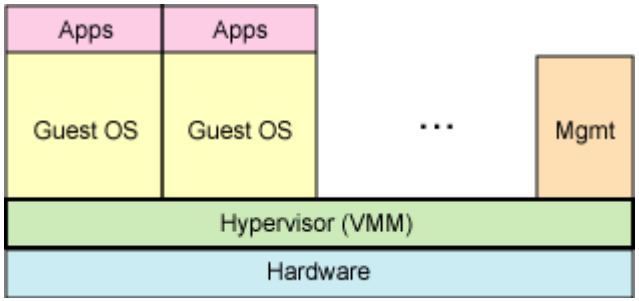
硬件仿真（Emulation）



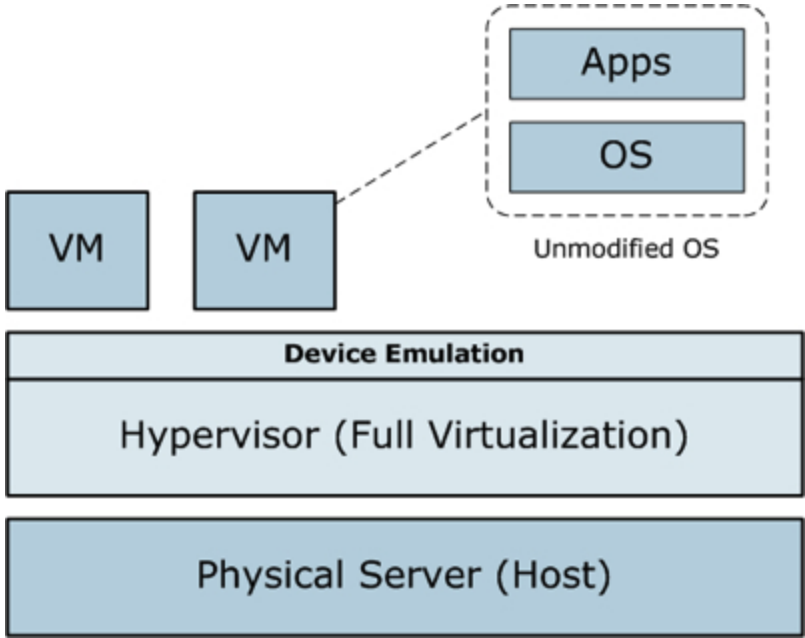
- 属于Hosted架构
- Host OS → VMM (Emulator) → Guest OS。
- 技术要点
  - 特权指令(Guest)>>陷入>>VMM>>模拟>>特权指令(Host)
- 知名产品
  - Bochs (open source)
  - QEMU (free)
  - Virtual PC (Microsoft)



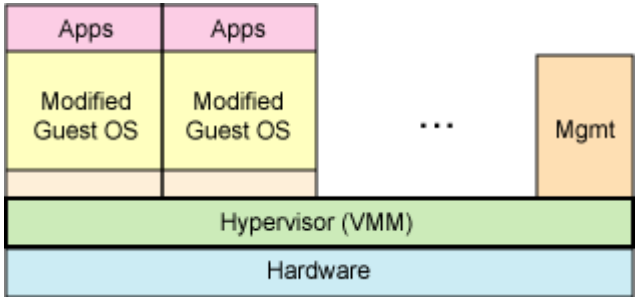
全虚拟化（Full Virtualization）



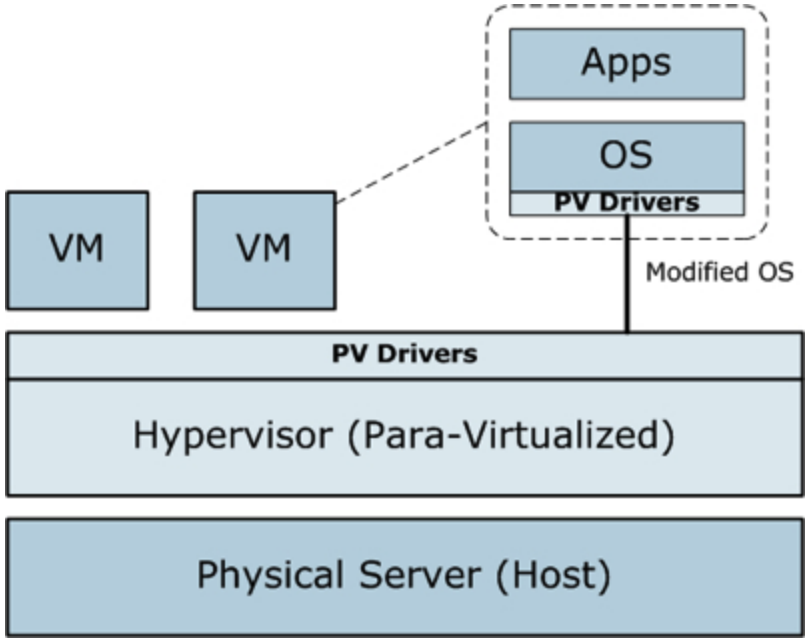
- 最常见、最成熟的虚拟化技术
- Hosted 和 Bare-metal都有
- 技术要点：
  - 在客户操作系统和硬件之间捕捉和处理那些对虚拟化敏感的特权指令，使客户操作系统无需修改就能运行
- 知名产品
  - IBM CP/CMS
  - Oracle VirtualBox
  - KVM
  - VMware ESX



半虚拟化（Parairtulization）



- Bare-metal模式
- 技术要点：
  - 与全虚拟化类似,利用VMM实现对底层硬件的访问
  - Guest OS集成与半虚拟化有关的代码，以配合VMM
  - 无需重新编译或捕获特权指令，性能非常接近物理机
- 知名产品
  - Xen
  - Microsoft Hyper-V



Hardware Assisted Virtualization

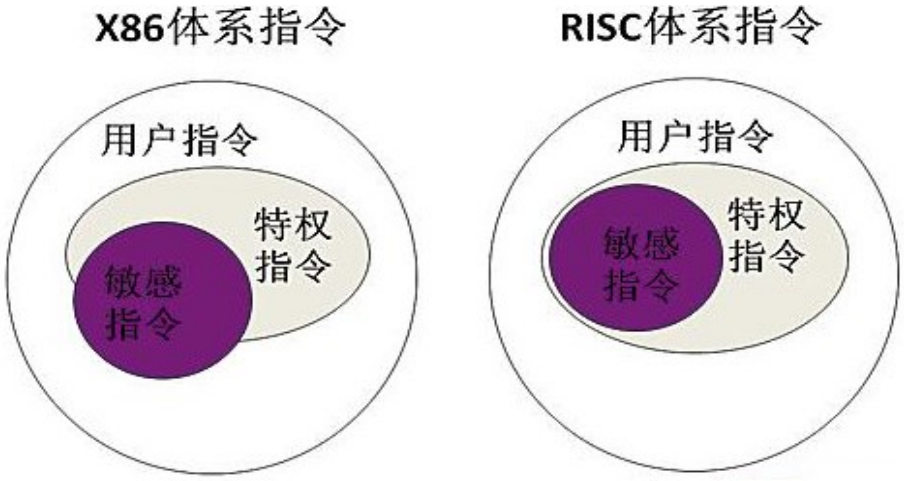
- 不是独立的虚拟化技术
- 结合到全/半虚拟化技术中
- 技术要点：
  - 通过对部分全虚拟化和半虚拟化使用到的软件技术进行硬件化来提高性能
- 知名产品
  - Intel VT-x
  - AMD-V (AMD SVM)

Comparisons

	硬件仿真虚拟化	全虚拟化	半虚拟化	硬件辅助虚拟化
速度	<30%	30%~80%+	80%+	80%+
模式	Hosted	Hosted/Bare-metal	Bare-metal	Hosted/Bare-metal
优点	Guest OS无需修改 非常适合硬件、固件及OS的开发	Guest OS无需修改，速度和功能都不错，使用非常简单	比全虚拟化架构更精简，速度上有优势	速度快
缺点	速度非常慢（有时速度比物理情况慢100倍以上）	基于Hosted模式时性能较差，特别是I/O方面	需要对Guest OS进行修改，用户体验较差	硬件实现不够优化
趋势	颓势但仍存	主流	一定份额	普遍采用

CPU虚拟化

- CPU基于不同技术进行虚拟化
  - 硬件仿真、全虚拟化、半虚拟化、硬件辅助
- 不同的CPU架构可能适于不同的技术
- CPU虚拟化的核心问题是特权/敏感指令执行



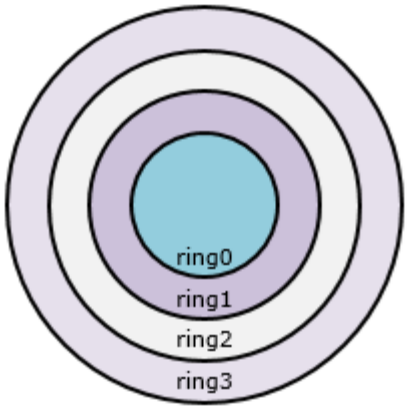
- **特权指令**：系统中有一些操作和管理关键系统资源的指令，只有在最高特权级上能够正确运行。如果在非最高特权级上运行，特权指令会引发一个异常，处理器会陷入到最高特权级，交由系统软件处理了。
- **敏感指令**：操作特权资源的指令，包括修改虚拟机的运行模式或者下面物理机的状态；读写时钟、中断等寄存器；访问存储保护系统、地址重定位系统及所有的I/O指令。

CPU虚拟化的挑战

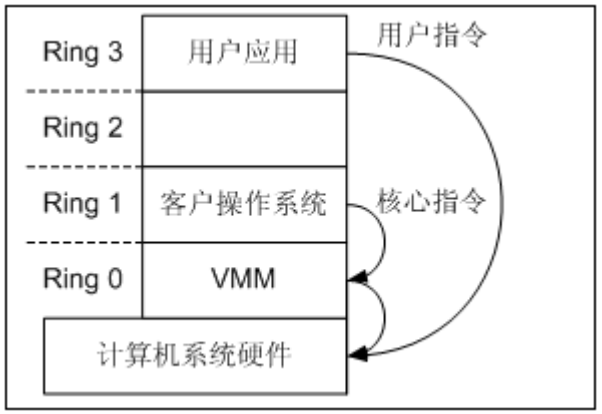
- X86架构的4个特权等级：Ring0-3
  - ✓ 操作系统内核需要直接访问硬件，运行在最高等级Ring0
    - ◆ 这样，内核可以使用特权指令控制中断、修改页表、访问设备
  - ✓ Ring1与Ring2也用于操作系统服务，被保留
  - ✓ 应用程序运行在最低特权级别的Ring3
- 应用程序执行方式
  - ✓ 一般指令运行在Ring3级别，不受管控
  - ✓ 访问磁盘、发送消息等操作需要通过系统调用进入内核执行
    - ◆ 执行权限由Ring3转到Ring0，任务完成后，系统调用返回，执行权限回归Ring3。以上过程成为用户态和内核态的切换
- 问题
  - ✓ 虚拟机作为一个应用程序运行在宿主机操作系统中，应该运行在Ring3
  - ✓ 但是，虚拟机操作系统的内核指令又需要运行在Ring0

CPU全虚拟化（模拟执行）

- 优先级压缩（Ring Compression）技术
  - 优先级压缩能让VMM和Guest运行在不同的特权级下
  - 让VMM截获一部分在Guest上执行的特权指令，并对其进行虚拟化。
  - Trap-and-emulation：
    - 当OS有特权指令产生时，产生“中断”，陷入VMM
    - VMM将OS所请求的特权指令进行截获，然后使用模拟仿真将特权指令模拟仿真的方式执行一遍。



X86 protection ring
- Ring 0 – VMM
- Ring 1 – Guest OS
- Ring 3 – Application



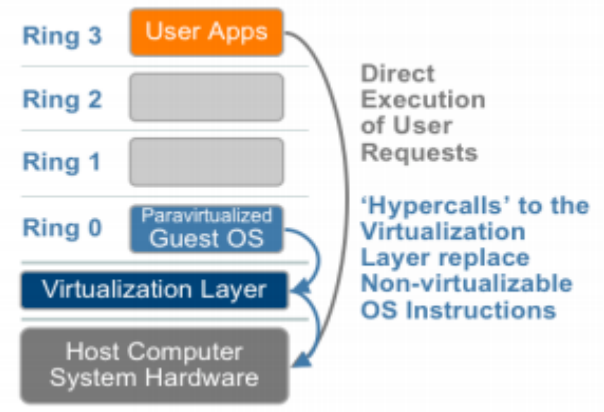
CPU全虚拟化（模拟执行）

- 二进制代码翻译技术（Binary Translation）技术
  - 用于一些对虚拟化不友好的指令
  - 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。
- 具体技术
  - ✓ 解释执行：逐条指令解释，用一个微程序模拟原始指令的功能
  - ✓ 静态翻译：重新编译
  - ✓ 动态翻译：在敏感指令之前插入跳转指令或者VMM陷入指令
    - ◆ VMM开辟一块缓存专门存放敏感指令的翻译后指令，并记录对应关系

类别	优点	缺点
解释执行	开发容易，不需用户干预，高度兼容	效率很差
静态翻译	离线翻译，可以进行更好的优化，效率较高	依赖解释器、运行环境的支持，需要终端用户参与
动态翻译	无需解释器和运行环境的支持，无需用户参与，利用动态信息来发现优化的机会	翻译代码效率不高，对目标机器有额外的开销

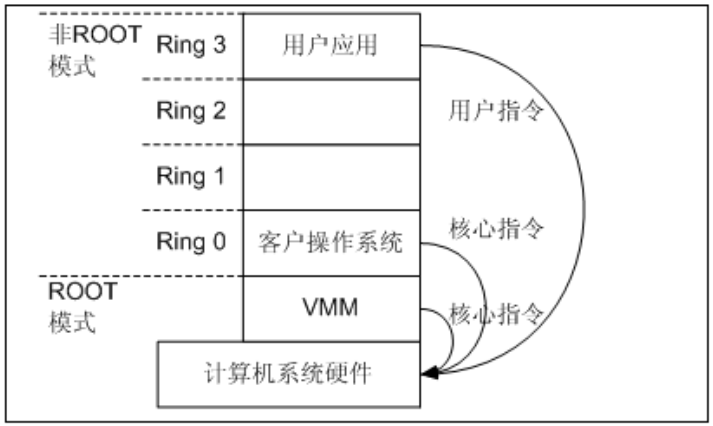
CPU半虚拟化（操作系统辅助）

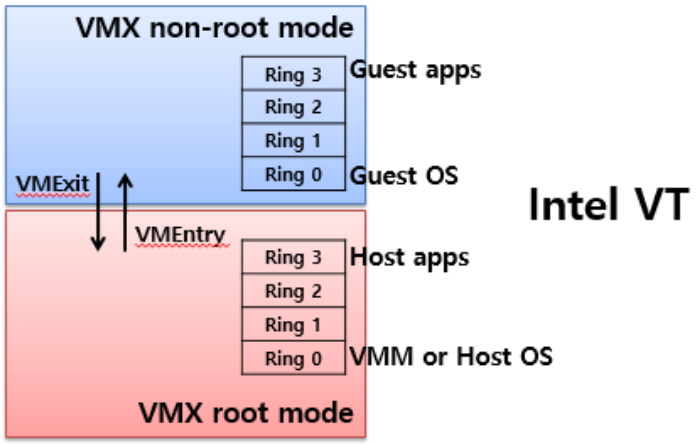
- 基于OS级的支持
- 通过修改Guest OS的代码，使其将那些和特权指令相关的操作都转换会发给VMM的Hypercall（超级调用）
- Hypercall支持Batch（批处理）和异步这两种优化方式，使Hypercall能得到接近物理机的速度。



CPU硬件辅助虚拟化

- 通过引入新的指令和运行模式，来让VMM和Guest OS能分别运行在其合适的模式下
- VT-x支持两种处理器工作方式
  - Root模式，VMM运行于此模式，用于处理特殊指令
  - Non-Root模式，Guest OS运行于此模
  - 当在Non-Root 模式Guest执行到特殊指令的时候，系统会切换到Root模式VMM，让VMM来处理特殊指令。

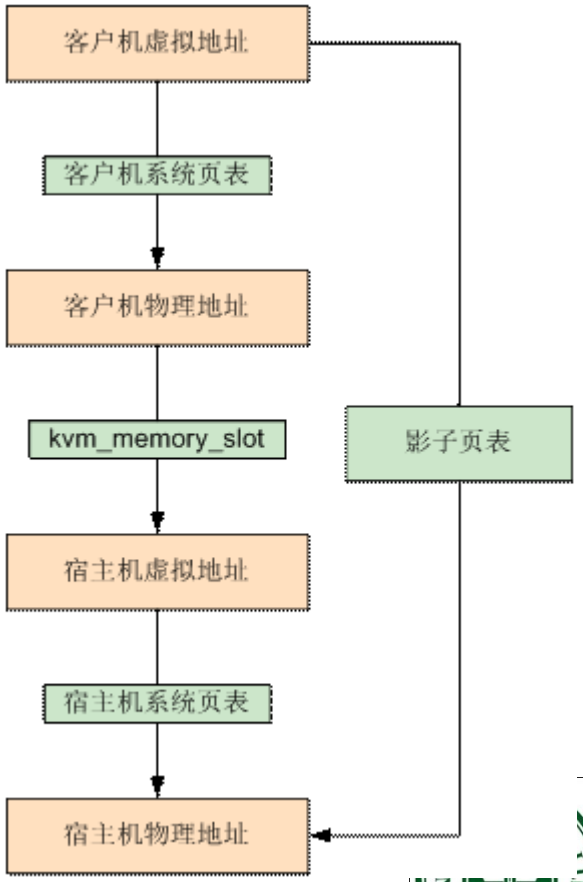




Intel VT

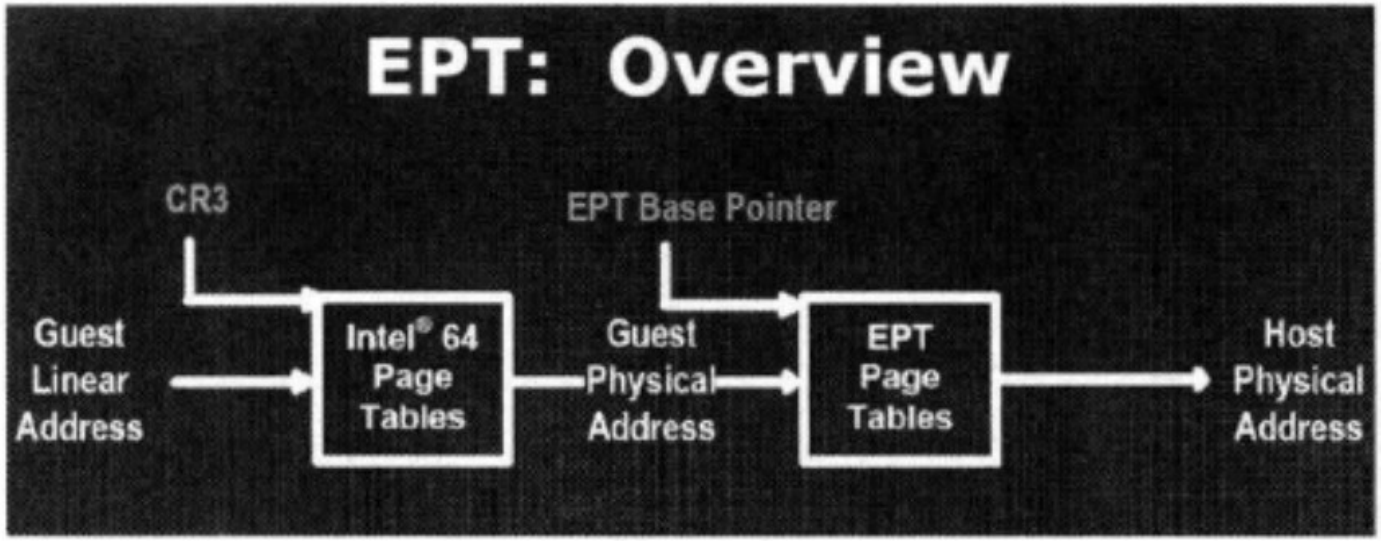
内存虚拟化

- **目标：**
  - 做好虚拟机内存空间之间的隔离，使每个虚拟机都认为自己拥有了整个内存地址，并且效率也能接近物理机。
- **全虚拟化：影子页表 Shadow Page Table**
  - 为每个Guest都维护一个影子页表
  - 写入虚拟化之后的内存地址映射关系
  - Guest OS的页表则无需变动
  - GVA->GPA->HVA->HPA→SPT
  - VMM将影子页表交给MMU进行地址转换
    - GVA到HPA
    - 避免了两次地址转换



内存虚拟化

- **硬件辅助虚拟化**
- 扩展页表EPT（Extended Page Table, by Intel）
  （NPT by AMD）
- 虚拟机内部维护自己的GVA->GPA页表结构
- 硬件辅助维护EPT/NPT来进行GPA->HPA的映射

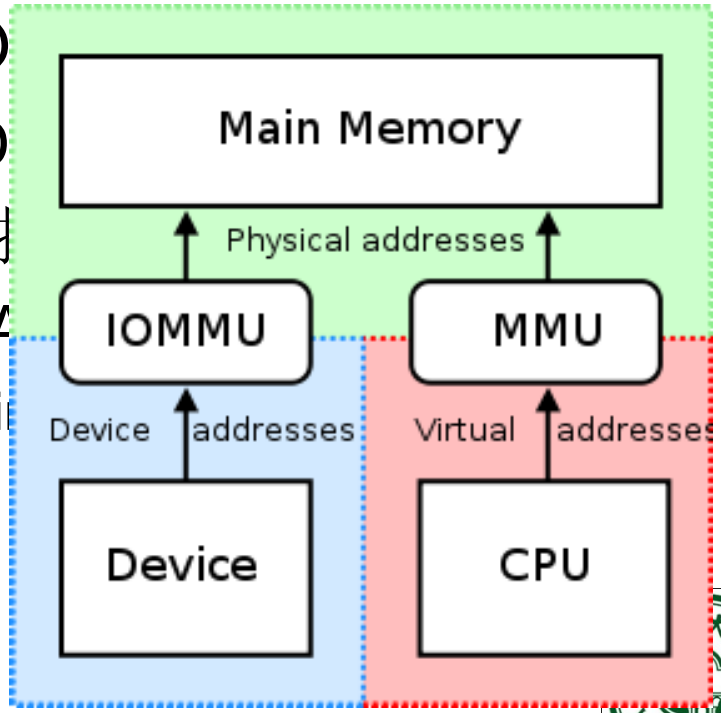


I/O虚拟化

- 目标：
  - 不仅让虚拟机访问到它们所需要的I/O资源，而且要做好它们之间的隔离工作，更重要的是，减轻由于虚拟化所带来的开销。
- 全虚拟化：
  - 通过模拟I/O设备（磁盘和网卡等）来实现虚拟化。
  - Guest OS每次I/O操作都会陷入到VMM，让VMM来执行。
  - 对Guest OS而言，它看到就是一组统一的I/O设备，完全透明。
- 半虚拟化：
  - 通过前端（Front-End）/后端（Back-End）架构，将Guest的I/O请求传递到特权域（Privileged Domain，也被称为Domain-0）。

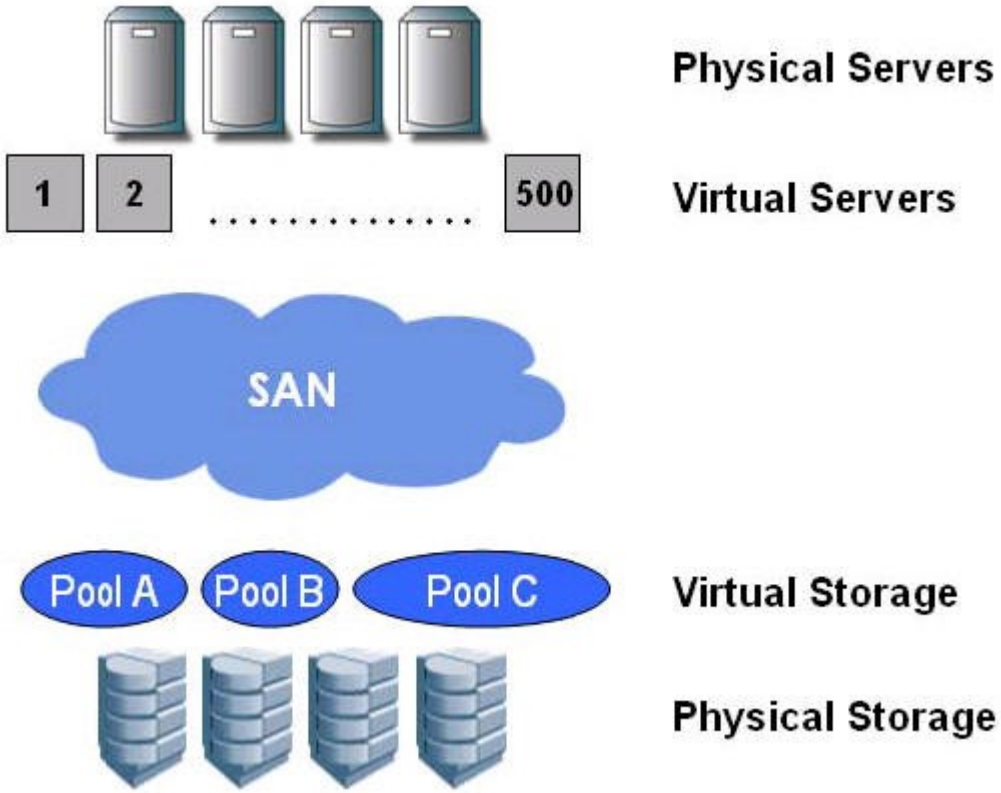
I/O虚拟化

- **软件模拟虚拟化：**
  - 用软件模拟I/O设备
  - Guest OS的操作被VMM捕获并交给Host OS的用户态进程，由其进行系统调用
- **硬件辅助虚拟化（直接划分）**
  - 代表产品：Intel的VT-d，AMD
  - 其核心思想就是让虚拟机能直
  - 技术要点：I/O地址访问和DMA
    - 通过采用DMA重映射（Remappi

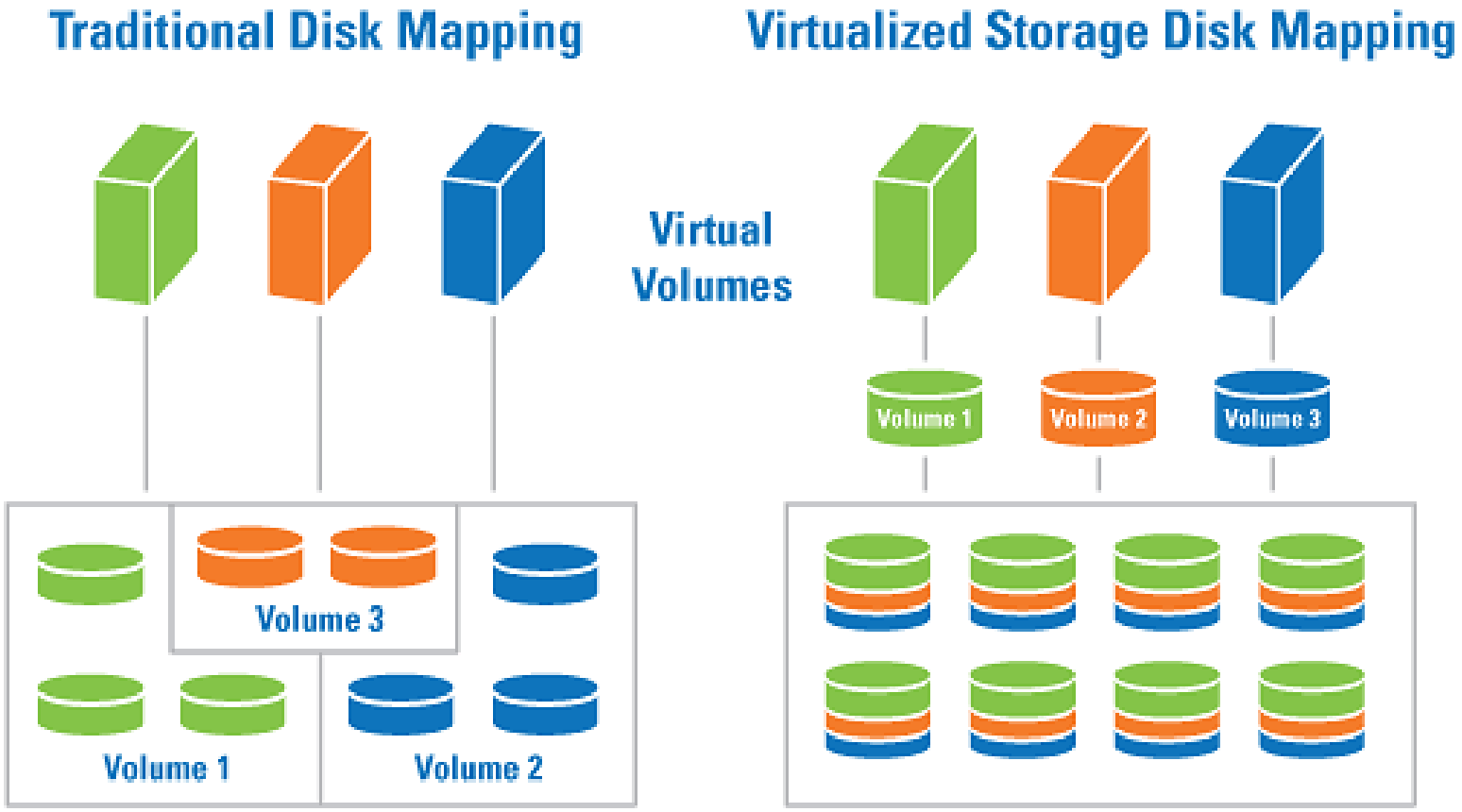


存储虚拟化

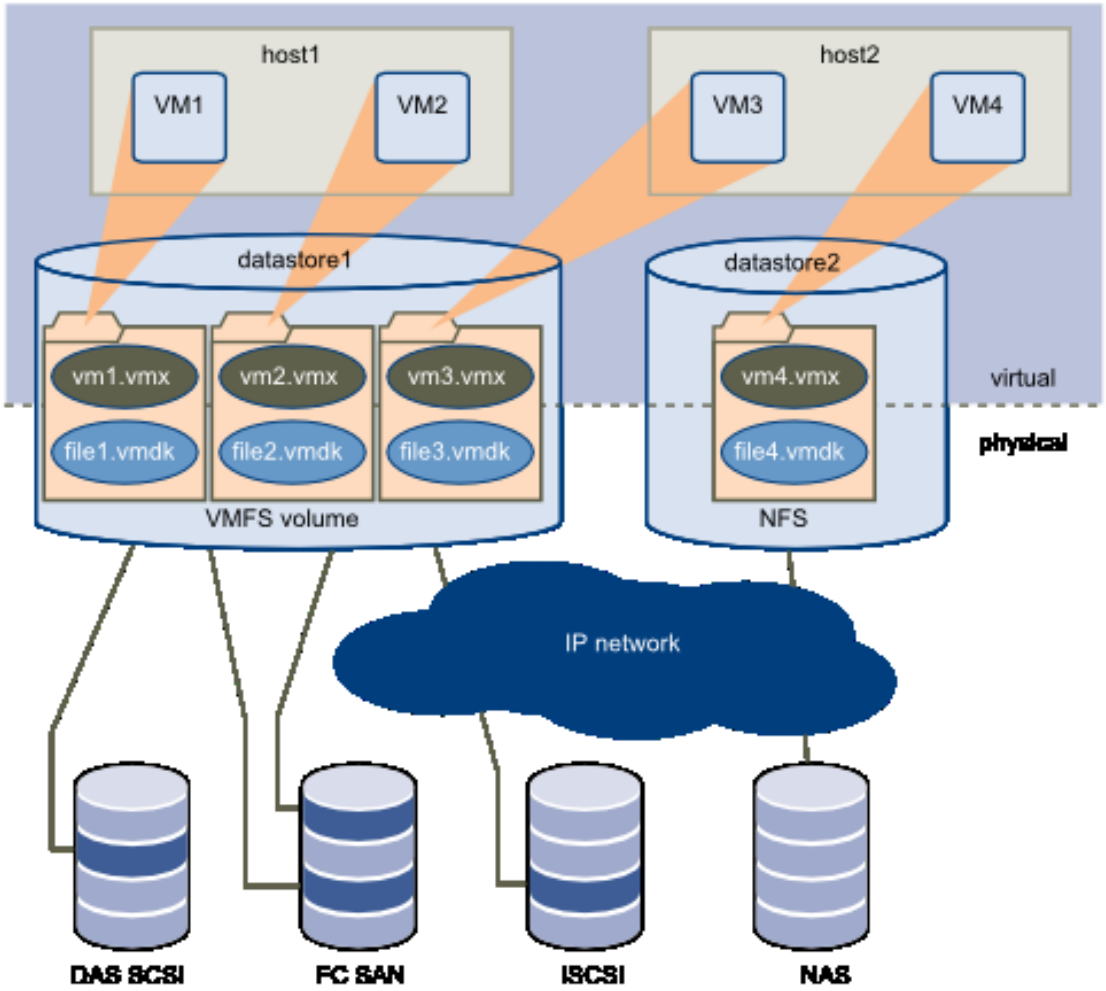
- 目标：实现与应用/网络无关的数据存储和管理
  - 存储功能 →|||抽象隔离|||←应用、主机、网络



传统存储 vs. 虚拟存储

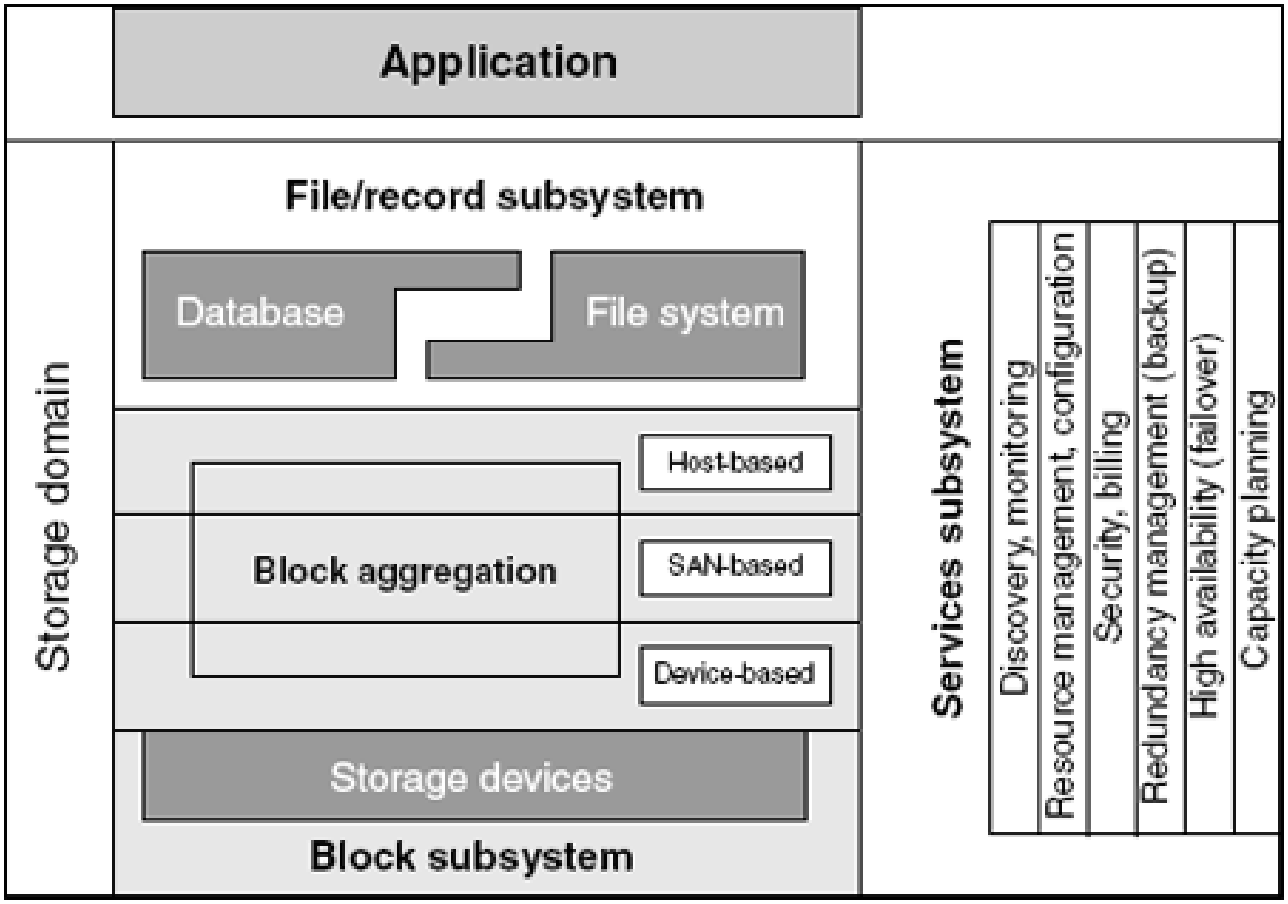

Traditional Disk Mapping
Virtualized Storage Disk Mapping
Virtual Volumes
Volume 1
Volume 2
Volume 3
Volume 3
Volume 1
Volume 2

整体架构（示例）


host1
VM1
VM2
host2
VM3
VM4
datastore1
datastore2
vm1.vmx
vm2.vmx
vm3.vmx
vm4.vmx
file1.vmdk
file2.vmdk
file3.vmdk
file4.vmdk
virtual
physical
VMFS volume
NFS
IP network
DAS SCSI
FC SAN
iSCSI
NAS

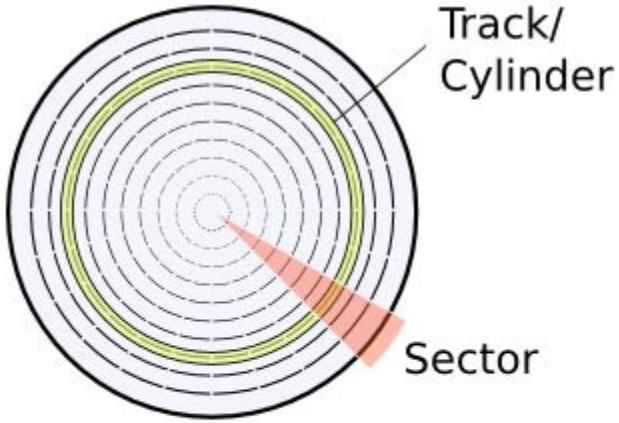
存储系统的基本结构 (from SNIA)

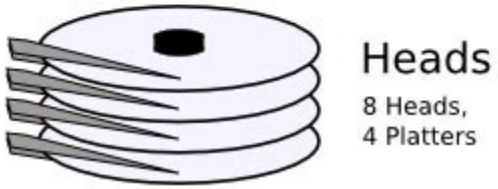
- 文件/记录层：为上层应用提供存储访问接口
- 块聚合层：将底层存储设备聚合为统一存储资源
- 存储设备层：识别数据块存储的物理位置，执行物理设备的数据读写。



设备层虚拟化

- 目标：
  - 虚拟化磁盘的物理属性，实现统一的寻址
    （不同类型的物理磁盘可以有相同的访问方式）
- 实现技术：
  - 磁盘固件，CHS信息→逻辑块编号

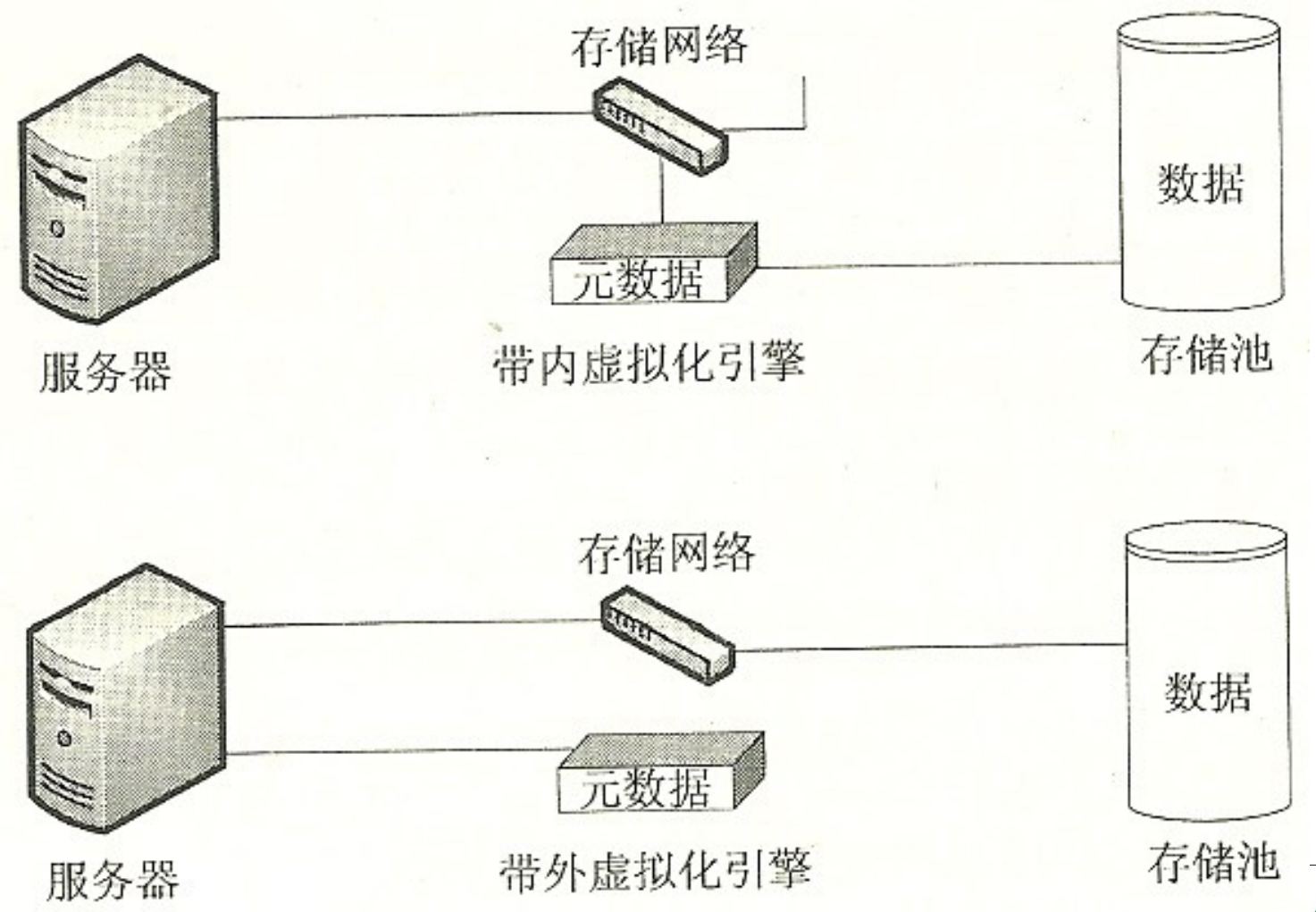




块聚合层虚拟化

- 目标：
  - 整合不同的物理存储设备
- 基本策略：
  - 先将多个物理存储器虚聚合成一个单一的虚拟存储器，提高容量、可靠性等；
  - 再将虚拟存储器划分成多个小存储器分配给用户。
- 虚拟化技术
  - 带内（In-band）虚拟化vs. 带外（Out-of-band）虚拟化

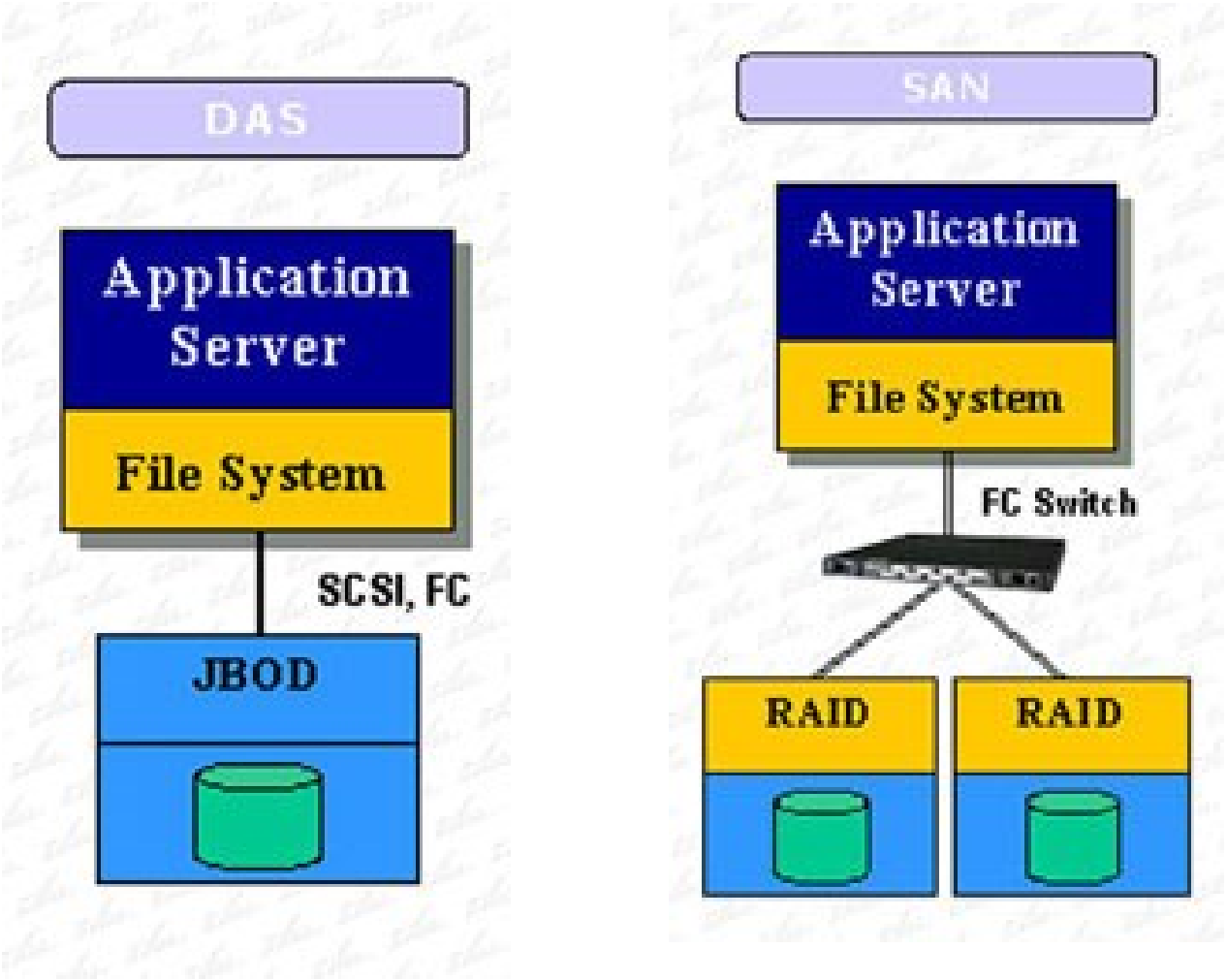
In-band vs. Out-of-band



块聚合层虚拟化技术

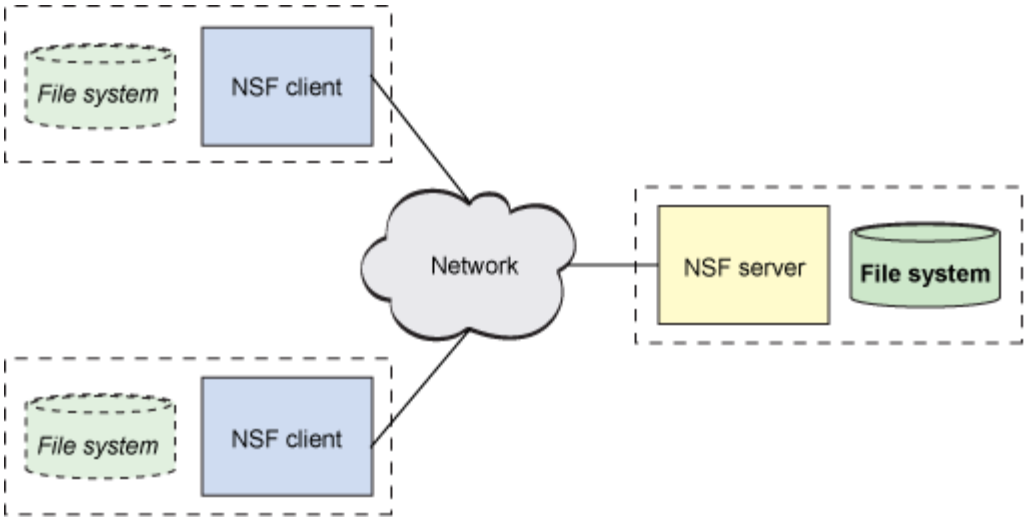
- 基于主机的虚拟化（带内）
  - 通过主机上的LVM（逻辑卷管理器）实现
  - 稳定、兼容异构；手工管理，配置开销大
- 基于存储网络的虚拟化（带内）
  - 由存储网络设备截获和处理存储请求
  - 两端透明；网络设备处理成瓶颈
- 基于存储设备的虚拟化（带内或者带外）
  - 功能全，性能高；不同方案不兼容

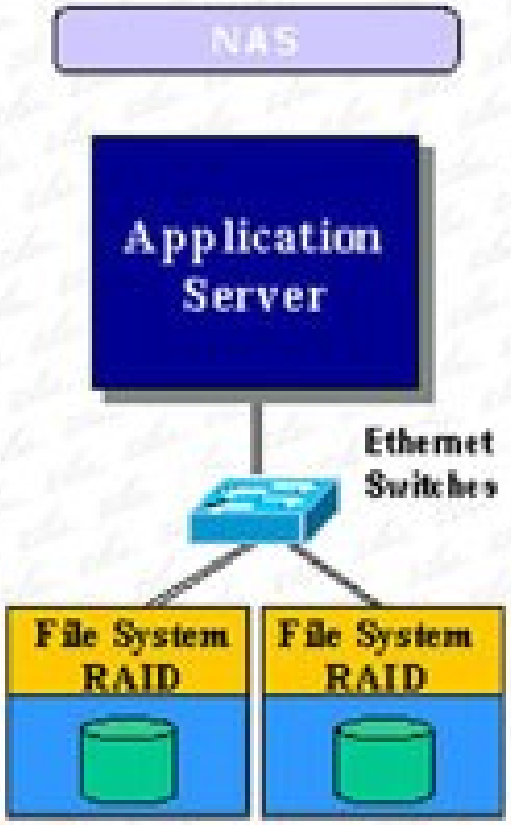
块聚合层虚拟化实例: DAS、SAN



文件/记录层虚拟化

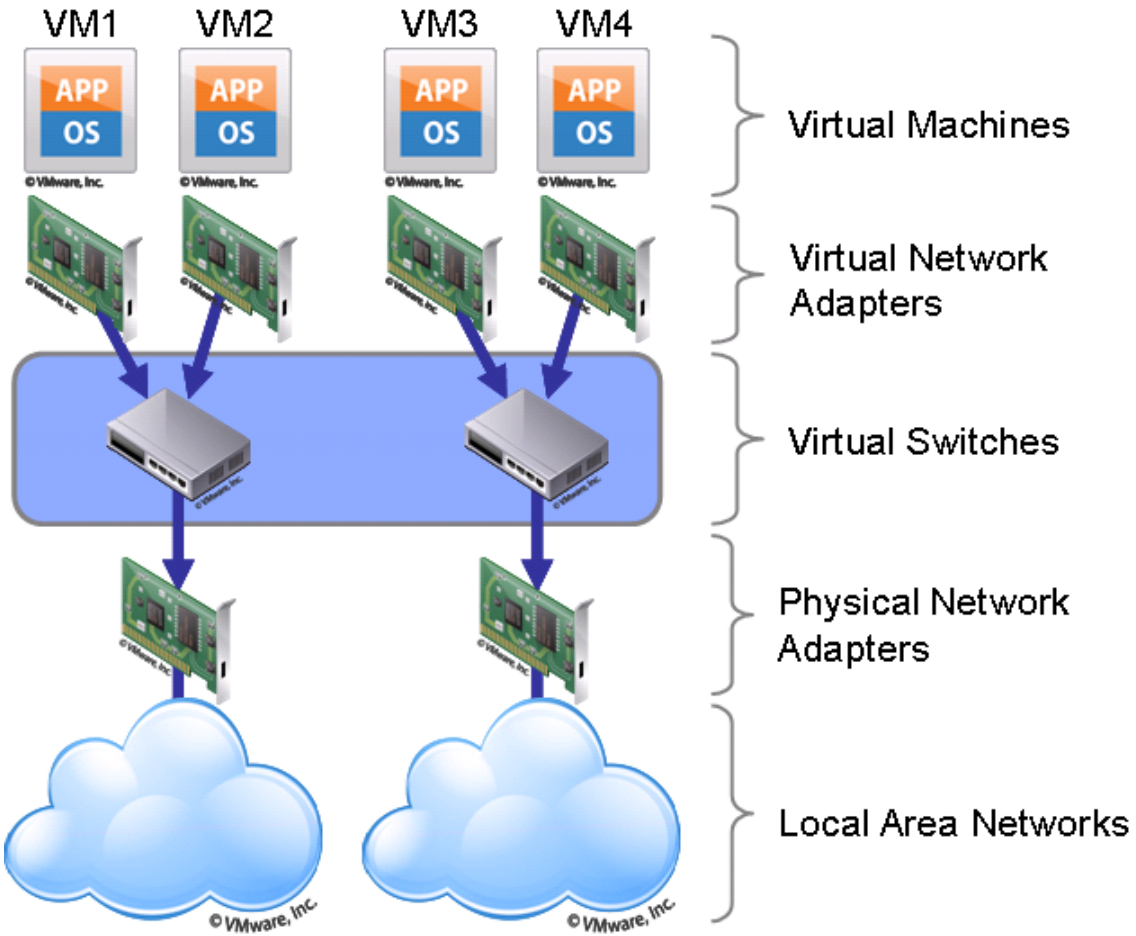
- 目标：
  - 整合不同的文件系统
  - 使上层用户能够透明、高效地访问存储在远程的文件。
- 实现技术
  - NFS、CIFS、NAS
  - 分布式文件系统





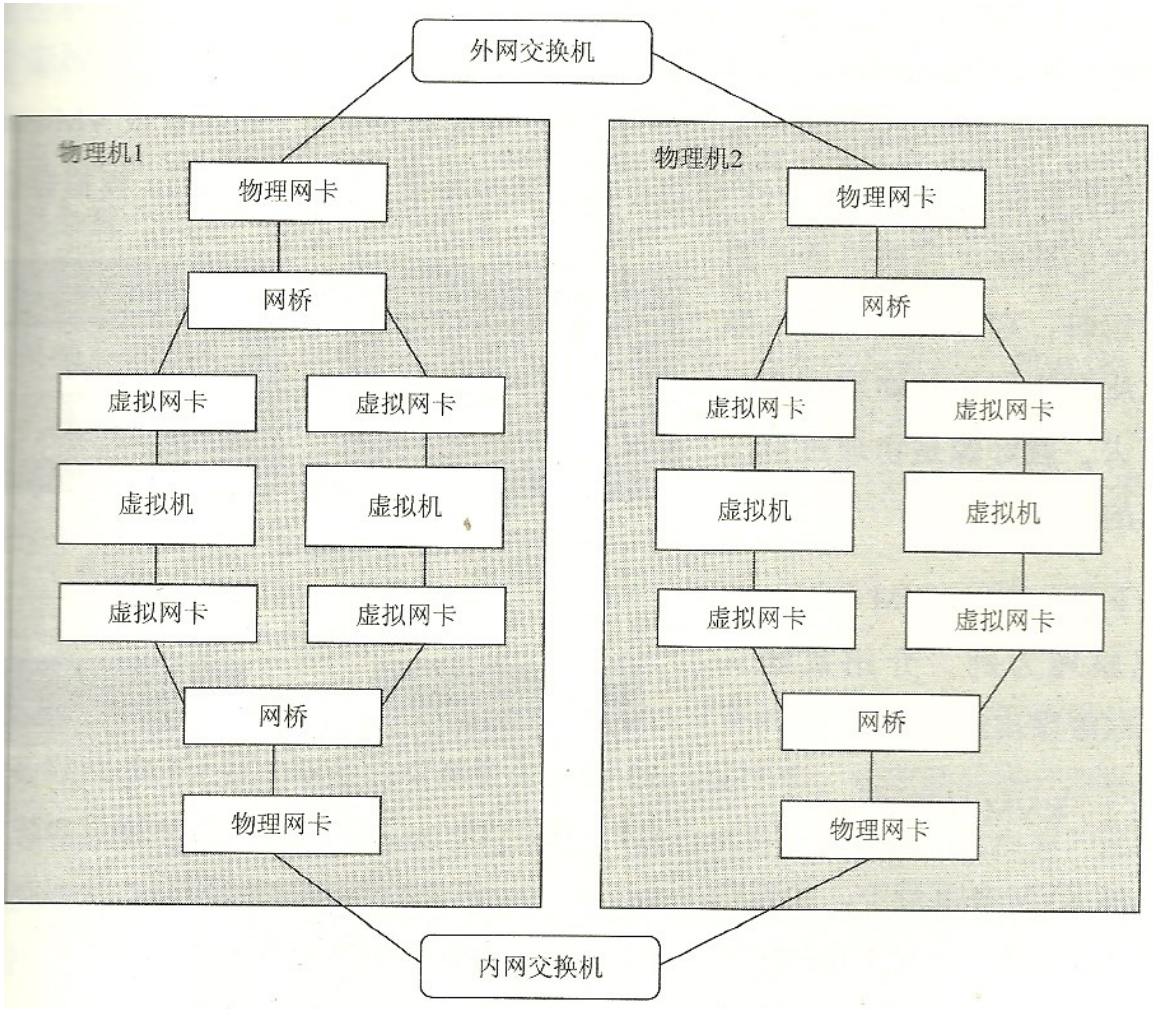
网络虚拟化

- 将网络资源和功能集成到一个软件中统一管控



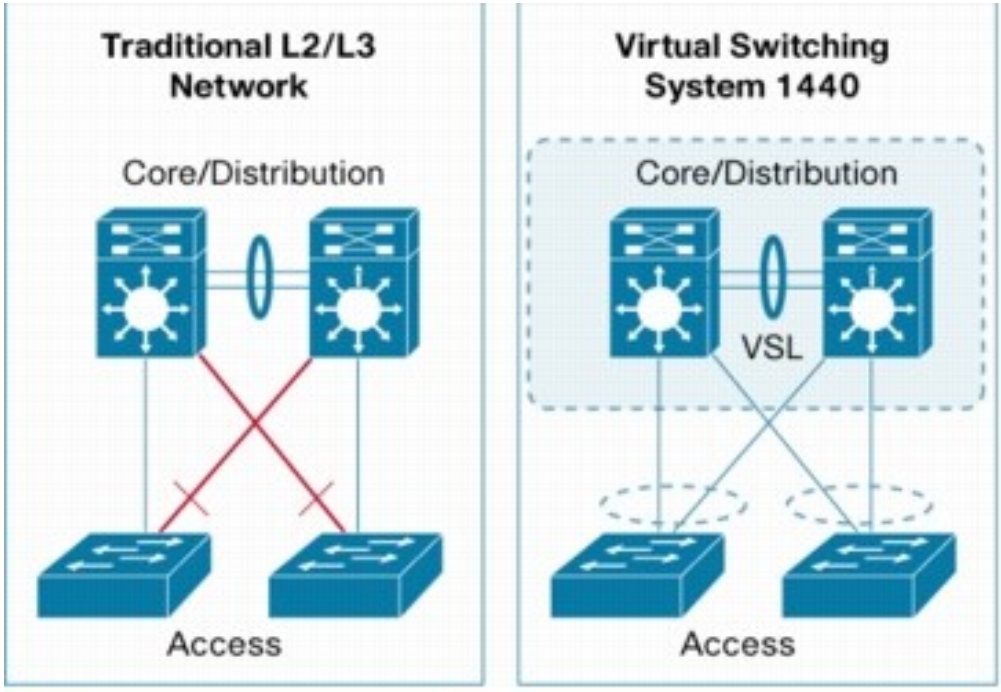
主机网络虚拟化

- 单台物理主机的网络设备的虚拟化

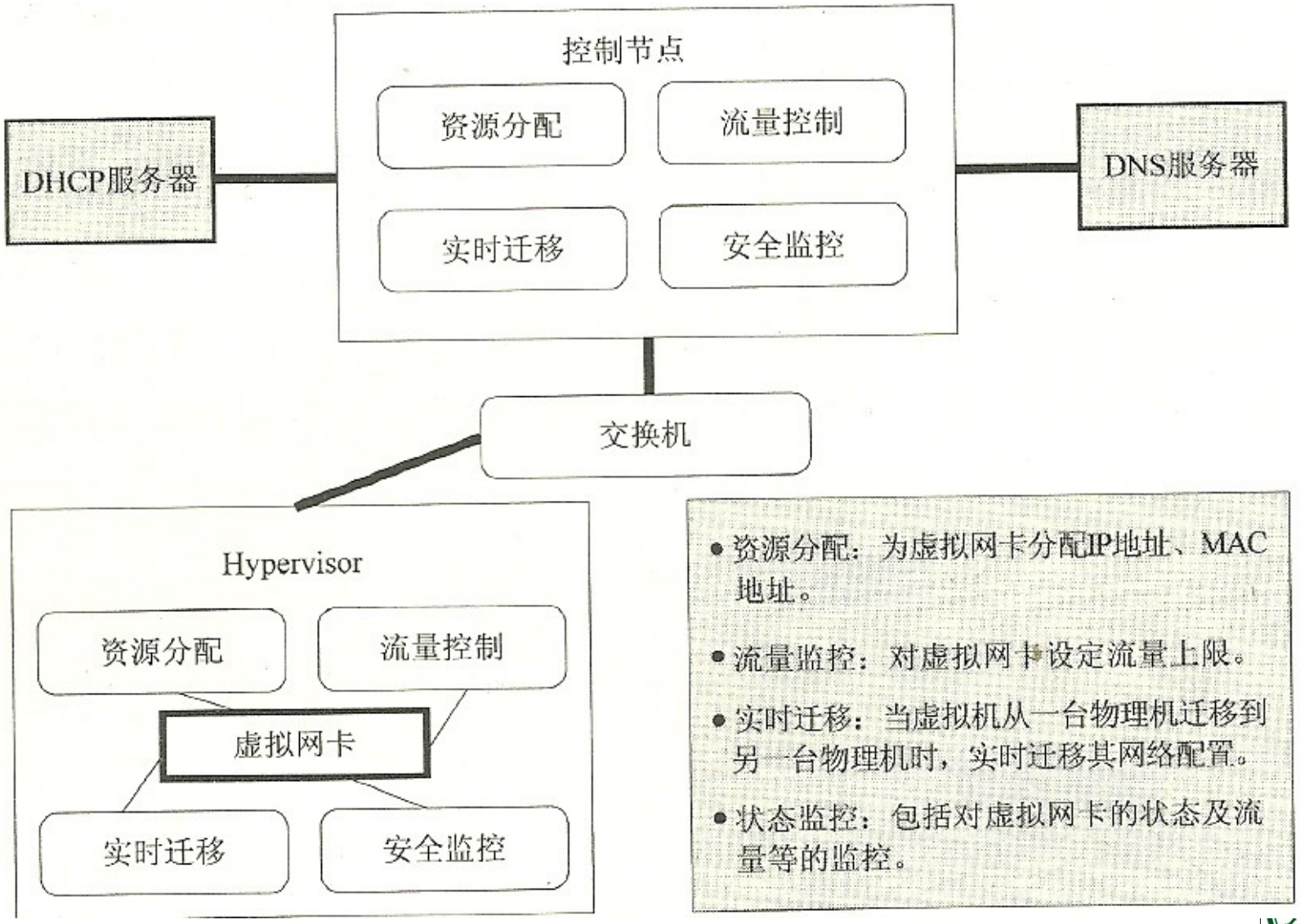


网络设备虚拟化

- 适应云计算环境的需求
  - 增加容量、应对横向流量、大缓存低延迟等。
- 技术要点
  - 网络交换设备层面实施虚拟化
- 两种形式
  - 横向整合：多个变一个
  - 纵向分割：VLAN等



网络虚拟化管理


控制节点
资源分配
流量控制
实时迁移
安全监控
DHCP服务器
DNS服务器
交换机
Hypervisor
资源分配
流量控制
虚拟网卡
实时迁移
安全监控
• 资源分配：为虚拟网卡分配IP地址、MAC地址。
• 流量监控：对虚拟网卡设定流量上限。
• 实时迁移：当虚拟机从一台物理机迁移到另一台物理机时，实时迁移其网络配置。
• 状态监控：包括对虚拟网卡的状态及流量等的监控。

§5.4 Web技术

- 用于访问云服务
  - I、P、S
- 需求要点
  - 迅速的动态交互
  - 丰富的交互手段
  - 高效的呈现性能
  - 界面可定制
  - 离线使用
  - 直观教程

Web应用

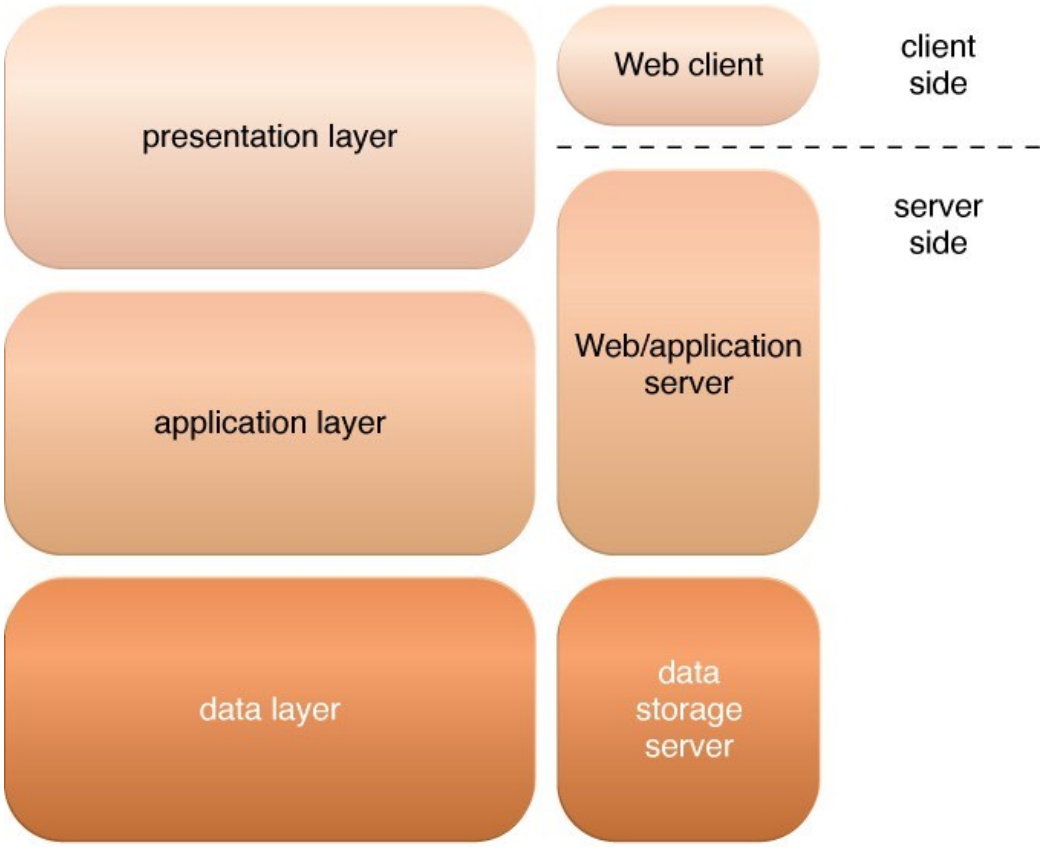


Figure 5.10 – Web应用的三层基本架构

基于插件的Web呈现技术

- Flash
- Silverlight
- JavaFX

基于浏览器的Web呈现技术

- HTML5
  - 实现网页结构与内容描述的扩展
  - 满足功能需求
  - 满足离线使用需求
- CSS3
  - 页面显示特效
  - 更接近客户端效果
- Ajax
  - 部分、异步交互数据
  - 避免页面重载导致的不连贯

HTML5

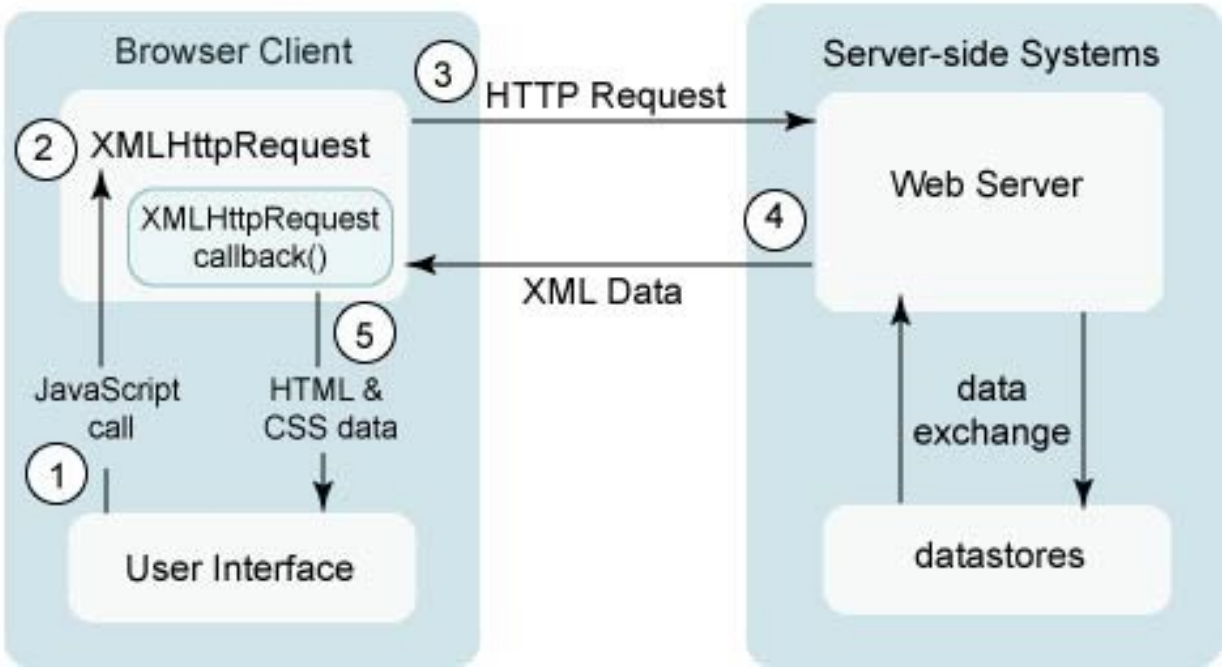
- W3C，2007
- 主要改进（相比HTML4及以前版本）
  - 增加Audio、Video等多媒体元素
    - 嵌入编解码器
    - 支持定时播放、播放控制
    - 支持H.264和Ogg
  - 增加Canvas元素
    - 用JavaScript绘图
  - 提供Geolocation地理位置API
  - 本地存储功能
  - 增加了结构化标签
  - 。 。 。

CSS3

- Cascading Style Sheets，层叠样式表
- 用来为结构化文档（如HTML文档或XML应用）添加样式（字体、间距和颜色等）
- CSS3的新特性
  - 文字特效
  - 动画支持
  - 。 。 。

Ajax

- 能在不更新整个页面的前提下维护数据
- 不是一种单一的技术，而是利用了一系列技术



小结

- 虚拟化技术分类
  - 虚拟机、容器…
- 虚拟机架构及技术原理
  - 硬件仿真、全/半虚拟化、硬件辅助…
- 存储虚拟化
- 网络虚拟化
- Web交互技术