上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

虚拟化

金耀辉 教授

网络信息中心/光纤通信国家重点实验室

http://front.sjtu.edu.cn/~jinyh/teaching

提纲

- 虚拟化之动机
- 虚拟化之定义
- 虚拟化之实现
- 虚拟化之应用
- 虚拟化之性能
- 虚拟化之研究

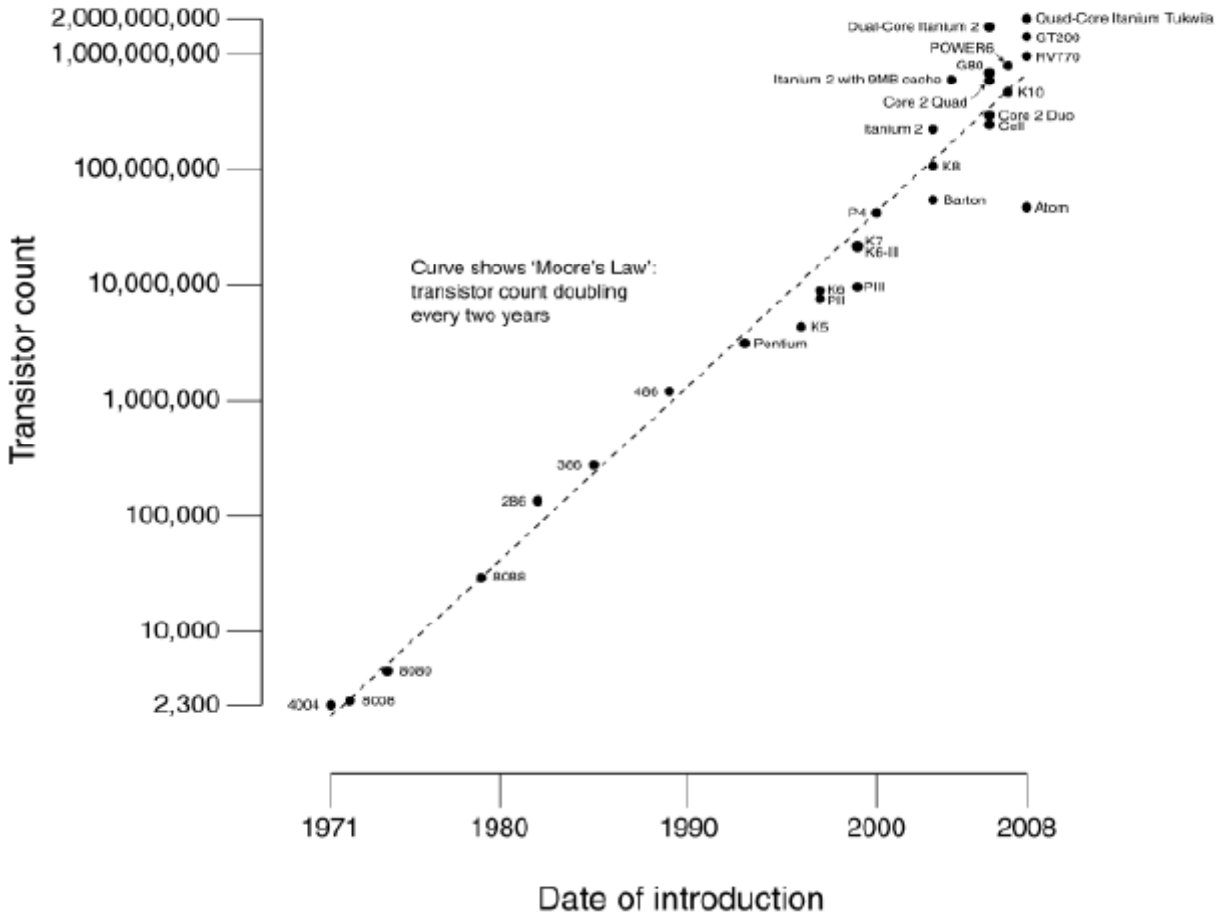
Not a New Idea

- IBM Cambridge Scientific Center
- Ran on IBM 360/67
  - Alternative to TSS/360, which never sold very well
- Replicated hardware in each “process”
  - Virtual 360/67 processor
  - Virtual disk(s), virtual console, printer, card reader, etc.
- CMS: Cambridge Monitor System
- A single user, interactive operating system
- Commercialized as VM370 in mid-1970s

虚拟化的动机？

- 硬件发展趋势
  - CPU，内存…
  - 一台普通PC的CPU负载通常是5～15%
- 成本是关键因素
  - 购买，电力，场地，维护…
- 其他更多考虑
  - 移动性，安全性，封装性…

硬件趋势：Moore's Law



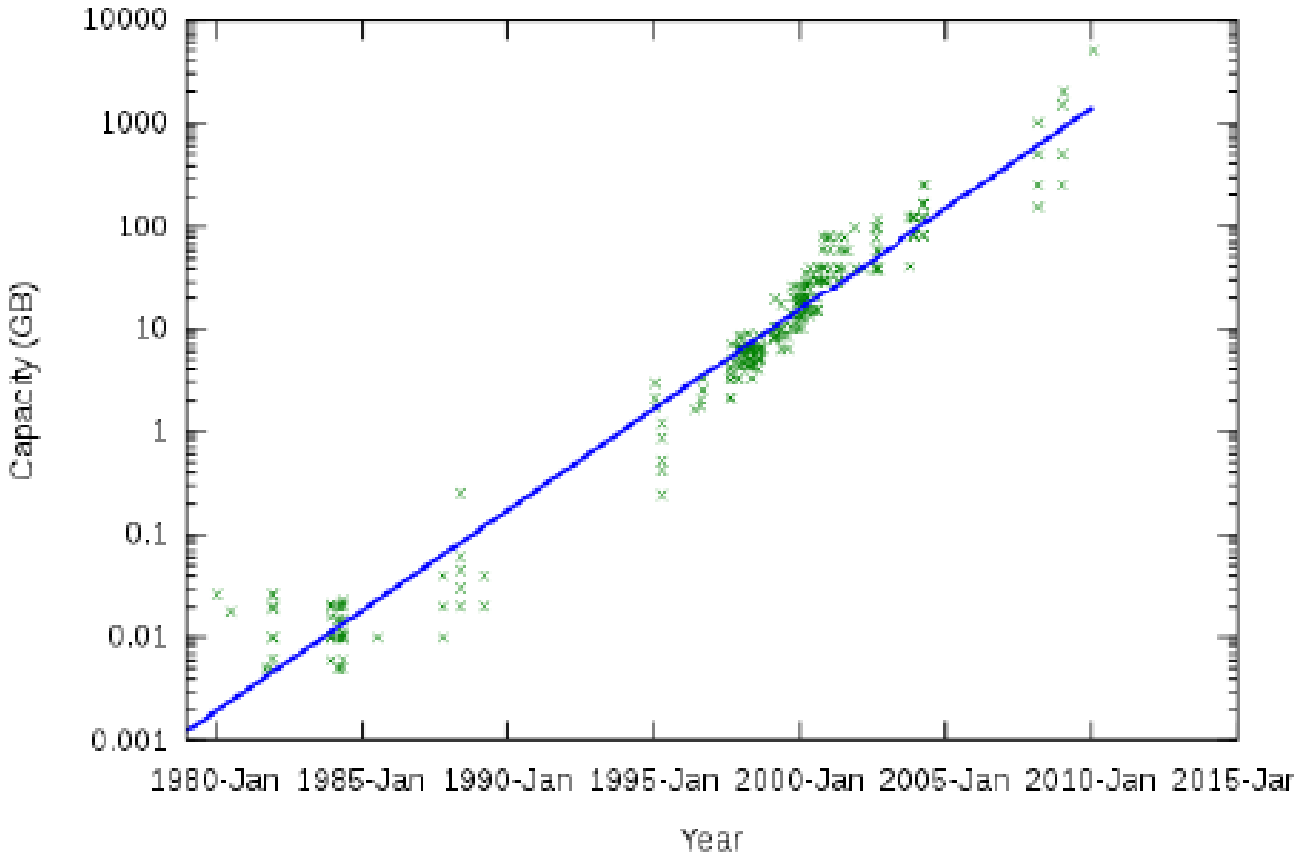
- 晶体管数量每2年翻番

硬件趋势：CPU频率

- 1975, Intel 8080, 2M Hz
- 1981, IBM PC, 4.77M Hz
- 1995, Pentium, 100M Hz
- 2002, Pentium4, 3G Hz

- CPU速率30年涨了1500倍！
- 牛X的微软，总是能把Intel的处理器变慢！

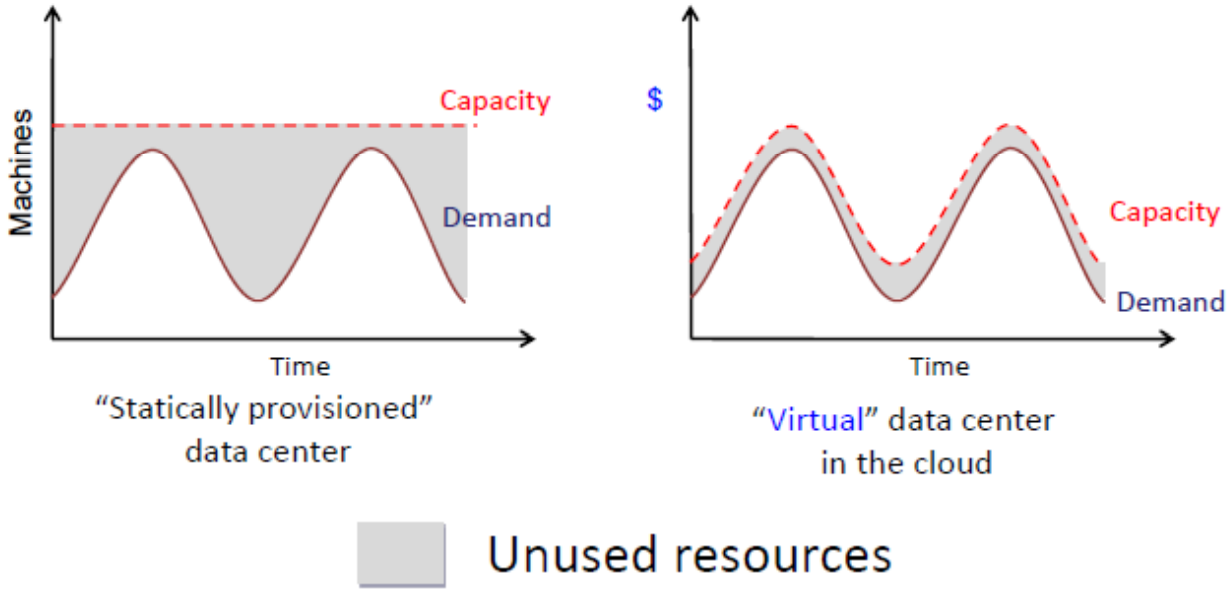
硬件趋势：内存/硬盘大小



- 内存/硬盘大小指数增加！

虚拟化的经济学考虑

- 低利用率
  - 如果峰值预测过于乐观，资源浪费
- 低供给率
  - 降低收入，客户流失

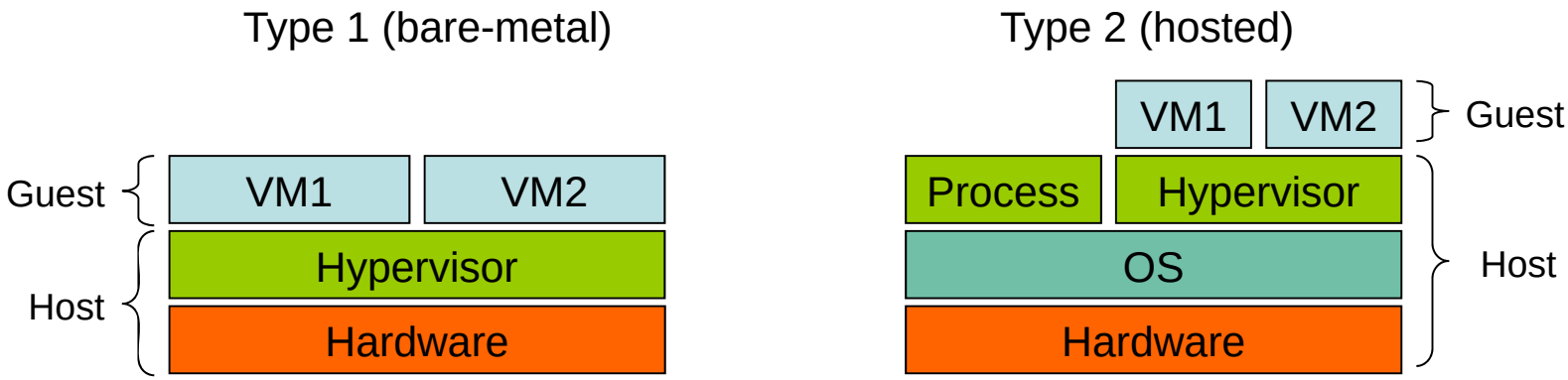


虚拟化是神马？

- 虚拟化就是Virtualization
- 虚拟化让每个进程看起来有一个单独的硬件资源
- 虚拟化可以让一个OS跑在另外一个OS上
- 虚拟化可以让多个OS跑在一个物理机上
- 虚拟化就是在软件中模拟一台物理机！
- 虚拟机是由虚拟化层提供的高效、独立的虚拟计算机系统，其皆拥有自己的虚拟硬件（CPU，内存，I/O 设备）。通过虚拟化层的模拟，虚拟机在上层软件看来，其就是一个真实的机器。

虚拟化术语

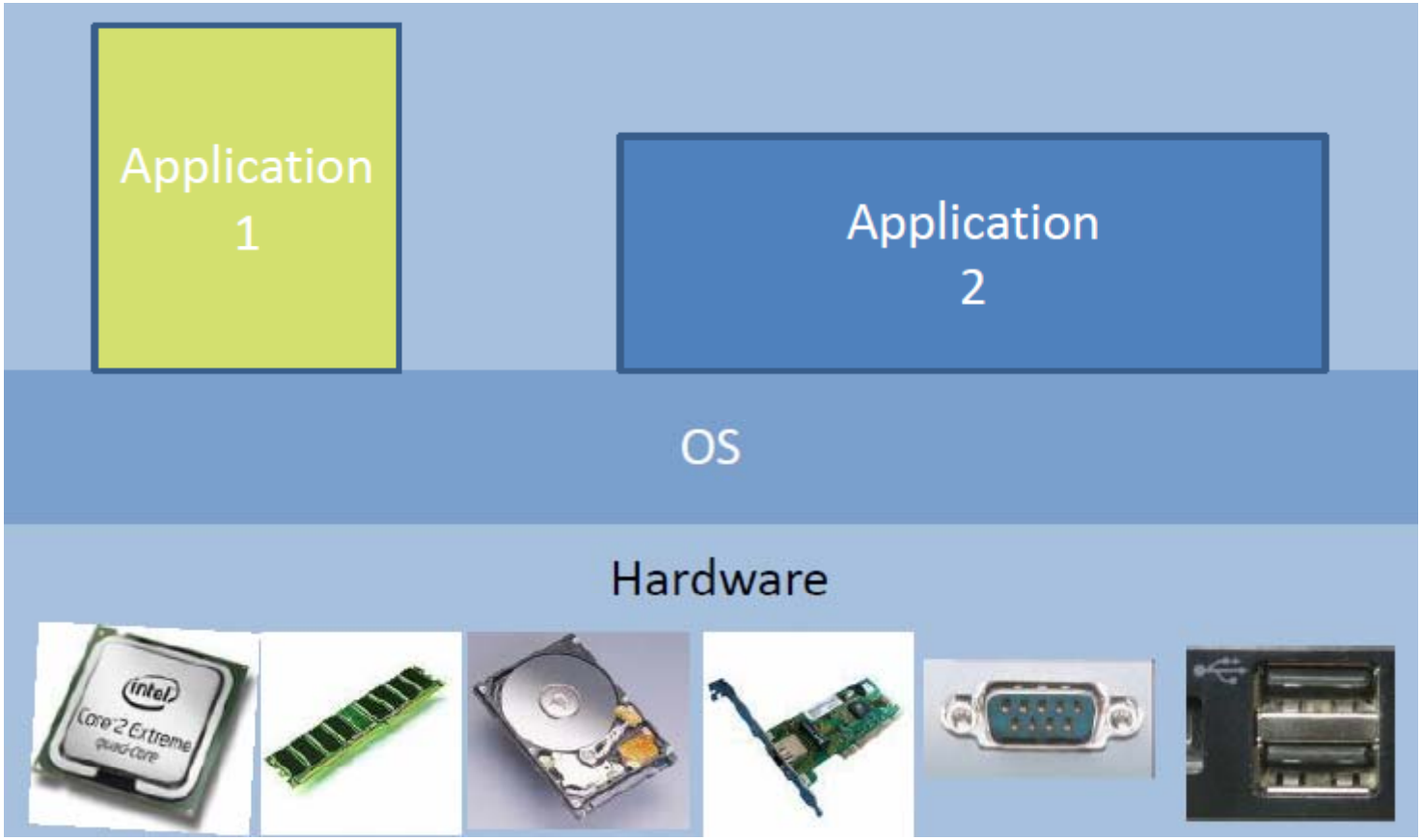
- Host：物理硬件和操作系统
- Guest：虚拟机和应用
- Virtual Machine Monitor (VMM)：允许多个虚拟机跑在一个物理机的软件层



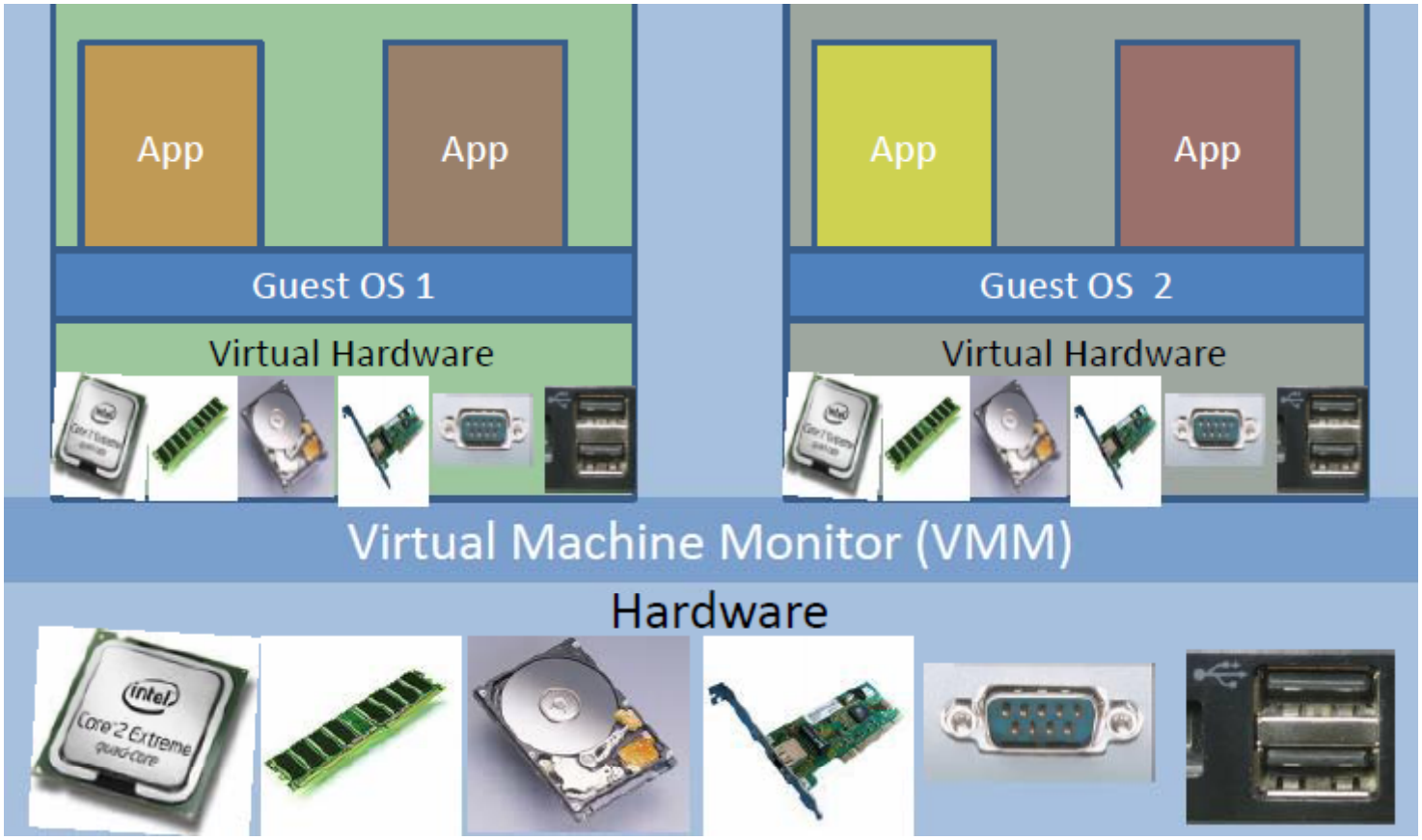
VMware ESX, Microsoft Hyper-V, Xen

VMware Workstation, Microsoft Virtual PC, Sun VirtualBox, QEMU, KVM

传统计算机


Application
1
Application
2
OS
Hardware
intel
Core 2 Extreme
quad-core

虚拟化计算机


App
App
App
App
Guest OS 1
Guest OS 2
Virtual Hardware
Virtual Hardware
Virtual Machine Monitor (VMM)
Hardware
intel
Core 2 Extreme
quad-core

虚拟化定义

- 虚拟化是资源的逻辑表示，不受物理限制的约束。
- 虚拟化技术的实现形式是在系统中加入一个虚拟化层，将下层的资源抽象成另一形式的资源，提供给上层使用。
- 本质上，虚拟化就是由位于下层的软件模块，通过向上一层软件模块提供一个与它原先所期待的运行环境完全一致的接口的方法，抽象出一个虚拟的软件或硬件接口，使得上层软件可以直接运行在虚拟环境上。
- 通过空间上的分割、时间上的分时以及模拟，虚拟化可将一份资源抽象成多份，亦可将多份资源抽象成一份。

虚拟机要求

G. Popek, R. Goldberg, “Formal Requirements for Virtualizable Third Generation Architectures“, Communication of the ACM, vol. 17, iss. 7

- 同质 (Equivalence)
  - VM 的运行环境和物理机的环境在本质上是相同的，表现上可以有一些差异。如 CPU 的 ISA 必须一致，CPU core 的个数可以不同。
- 高效 (Efficiency)
  - VM 的性能必须接近物理机。因此，常见的模拟器 (boches, simics ...) 就不能称为 VM为达此目的，软件在 VM 上运行时，大多数指令要直接在硬件上执行，只有少量指令需要 VMM 的模拟或处理。
- 资源受控 (Resource control)
  - VMM 对物理机的所有资源有绝对的控制力

虚拟化(VMM)的江湖

开源软件

Kernel-based Virtual Machine

VMWare Workstation
VMWare ESXi

XenDesktop
XenServer

Hyper-V

商业软件

擂台比武：透明、开销、性能、费用

虚拟化分类

- Full virtualization:
  - 所抽象的 VM 具有完全的物理机特性，OS 在其上运行不需要任何修改。
  - 典型的有 VMWare, Virtualbox, KVM ...
- Para-virtualization:
  - 需 OS 协助的虚拟化，在其上运行的 OS 需要修改。
  - 起初采用主要是为了解决 x86 体系结构上完全虚拟化的困难（没有 Intel VT & AMD-V 硬件虚拟化支持前；
  - 且不屑于动态扫描指令修补之方法的性能），后来则主要是为了提高虚拟化的效率。
  - 典型的有 Xen 。

物理资源的虚拟化

- 处理器虚拟化
- 内存虚拟化
- I/O 设备虚拟化

体系结构背景

- 处理器呈现给软件的接口就是一堆的指令（指令集）和一堆的寄存器（含用于通用运算的寄存器和用于控制处理器行为的状态和控制寄存器）。
- I/O 设备呈现给软件的接口也就是一堆的状态和控制寄存器（有些设备亦有内部存储）。
- 其中影响处理器和设备状态和行为的寄存器称为关键资源或特权资源。
  - x86 的CR0 ~ CR4，MIPS 的 CP0 寄存器，PowerPC 的 Privileged SPR。
- 可以读写系统关键资源的指令叫做敏感指令
  - x86 的 lgdt/sgdt/lidt/sidt/in/out，MIPS 的 mtc0/mfc0，PowerPC 的 mtmsr/mfmsr，SPARC 的 rdpr/wrpr 等，

特权指令

- 现代计算机体系结构一般至少有两个特权级，即用户态和核心态，
  - 未加虚拟化扩展的 SPARC和PowerPC
  - MIPS 有三个特权级（外加一个 Supervisor 态，没什么用）
  - x86 有四个特权级 (Ring0 ~ Ring3)）用来分隔系统软件和应用软件。
- 决大多数的敏感指令是特权指令，特权指令只能在处理器的最高特权级（内核态）执行，如果执行特权指令时处理器的状态不在内核态，通常会引发一个异常而交由系统软件来处理这个“非法访问”（Trap）。
- 对于一般 RISC 处理器，如 MIPS，PowerPC 以及 SPARC，敏感指令肯定是特权指令，唯 x86 例外。

倒外引起的江湖纷争

- VMware 为代表的 Full virtualization 派对无需修改直接运行理念的偏执。
- Xen 适当修改 Guest OS 后获得极佳的性能，以至让 Para virtualization 大热。
- 后来 Intel 和 AMD 卷入战火，从硬件上扩展，一来解决传统 x86 虚拟化的困难，二来为了性能的提升。
- 最后硬件扩展皆为 Full 派和 Para 派所采用，自此 Para 派的性能优势再不那么明显，Full 派的无需修改直接运行的友好性渐占上风。

经典的虚拟化方法

- 经典的虚拟化方法主要使用“特权解除” (Privilege deprivileging) 和“陷入－模拟” (Trap-and-Emulation) 的方式。即：将 Guest OS 运行在非特权级（特权解除），而将 VMM 运行于最高特权级（完全控制系统资源）。
- 解除了 Guest OS 的特权后，Guest OS 的大部分指令仍可以在硬件上直接运行，只有当执行到特权指令时，才会陷入到 VMM 模拟执行（陷入－模拟）。
- 其早期的代表系统是 IBM VM/370

虚拟化对体系结构 (ISA) 的要求

1. 须支持多个特权级
2. 非敏感指令的执行结果不依赖于 CPU 的特权级
   - “陷入－模拟” 的本质是保证可能影响 VMM 正确运行的指令由 VMM 模拟执行，大部分的非敏感指令还是照常运行。
3. CPU 需支持一种保护机制，如 MMU，可将物理系统和其它 VM 与当前活动的 VM 隔离
4. 敏感指令需皆为特权指令
   - 此为保证敏感指令在 VM 上执行时，能陷入到 VMM.
   - 因控制敏感指令的执行可能改变系统（处理器和设备）的状态，为保证 VMM 对资源的绝对控制力维护 VM 的正常运行，这类指令的执行需要陷入而将控制权转移到 VMM，并由其模拟处理之。
   - 行为敏感指令的执行结果依赖于 CPU 的最高特权级，而 Guest OS 运行于非最高特权级，为保证其结果正确，亦需要陷入 VMM，并由其模拟之。

x86 ISA分析

- x86 ISA 中有十多条敏感指令不是特权指令，因此 x86 无法使用经典的虚拟化技术完全虚拟化。
- 如：sgdt/sidt/sldt 可以在用户态读取特权寄存器 GDTR/IDTR/LDTR 的值；popf/pushf 在 Ring0 和 Ring3 的执行结果不同；其它的还有 smsw, lar, lsl, verr, verw, pop, push, call, jmp, int n, ret, str, move
- 关于这些指令的详细分析可以参见："Analysis of the Intel Pentium's Ability to Support a Secure Virtual Machine Monitor"

x86 虚拟化方法

- 鉴于 x86 指令集本身的局限，长期以来对 x86 的虚拟化实现大致分为两派，即以 VMWare 为代表的 Full virtualization 派和以 Xen 为代表的 Para virtualization 派。
- 两派的分歧主要在对非特权敏感指令的处理上，
- Full 派采用的是动态的方法，即：运行时监测，捕捉后在 VMM 中模拟；
- 而 Para 派则主动进攻，将所有用到的非特权敏感指令全部替换，这样就少掉了大量的陷入 —> 上下文切换 —> 模拟 —> 上下文切换过程，获得了大幅的性能提升，但缺点也很明显。
- 后来 x86 的主要厂商 Intel 和 AMD 也卷入战火，且都引入各自的硬件虚拟化技术来扩展。

全虚拟化派 (Full virtualization)

- 秉承无需修改直接运行的理念，该派一直在对“运行时监测，捕捉后模拟”的过程，进行偏执的优化。该派内部之实现又有些差别，其有 VMWare 为代表的 BT 和以 SUN 为代表的 Scan-and-Patch

基于二进制翻译 (BT) 的全虚拟化

- 在执行时将 VM 上执行的 Guest OS 之指令。
- 翻译成 x86 ISA 的一个子集，其中的敏感指令被替换成陷入指令。
- 翻译过程与指令执行交叉进行。
- 不含敏感指令的用户态程序可以不经翻译直接执行。
- 该技术为 VMWare Workstation，VMWare ESX Server 早期版本，Virtual PC 以及 QEMU 所采用。

基于扫描与修补 (Scan-and-Patch) 的全虚拟化

- VMM 会在 VM 运行每块指令之前对其扫描，查找敏感指令
- 补丁指令块会在 VMM 中动态生成，通常每一个需要修补的指令会对应一块补丁指令
- 敏感指令被替换成一个外跳转，从 VM 跳转到 VMM，在 VMM 中执行动态生成的补丁指令块
- 当补丁指令块执行完后，执行流再跳转回 VM 的下一条指令处继续执行
- SUN 之 Virtualbox 即采用该技术。

OS 协助的类虚拟化派 (Paravirtualization)

- 通过修改 Guest OS 的代码，将含有敏感指令的操作，替换为对 VMM 的超调用 (Hypercall，类似 OS 的系统调用，可将控制权转移到 VMM)。
- 该技术的优势在于 VM 的性能能接近于物理机，缺点在于需要修改 Guest OS.
- 该技术因 Xen 项目而广为人知。

硬件协助的虚拟化

- 鉴于 x86 在虚拟化上的缺陷，Intel 和 AMD 都引入自己的硬件虚拟化技术来协助完成虚拟化。
  - Intel VT-x (Virtualization Technology for x86)
  - Intel VT-i (Virtualization Technology for Itanium)
  - Intel VT-d (Virtualization Technology for Directed I/O)
  - AMD-V (AMD Virtualization)
- 引入新的处理器运行模式和新的指令，使得 VMM 和 Guest OS 运行于不同的模式下，Guest OS 运行于受控模式，原来的一些敏感指令在受控模式下全部会陷入 VMM，这样就解决了部分非特权的敏感指令的陷入 — 模拟难题，而且模式切换时上下文的保存恢复由硬件来完成，这样就大大提高了陷入 — 模拟时上下文切换的效率。
- 该技术的引入使 x86 可以很容易地实现完全虚拟化。其皆被双方所采用，如： KVM-x86，VMWare ESX Server 3，Xen 3.0

其它体系结构分析

- 其它 RISC 之体系结构，如 MIPS, PowerPC, SPARC 等，似乎不存在有敏感指令为非特权指令的情形。即它们应不存在虚拟化的困难。
- 但由于设计理念不一样，RISC 不太可能在硬件上实现像 x86 那样复杂的虚拟化扩张，因此为了性能的考虑，一样皆采用 Para-virtualization，在嵌入式领域尤其如此。
- 硬件上的扩展：
  - 2001 年 IBM 在 Power4 中加入虚拟化支持，并在 2004 年的 Power5 中推出增强的虚拟化支持，且在 2009 年发布的 Power ISA v2.06 中规范化。Freescale 亦在 e500mc 中实现 PowerISA v2.06 的虚拟化增强。
  - 2005 年 SUN 即在 SPARC 中引入虚拟化支持。
  - MIPS 至今未见动作（龙芯啊～～～）。

内存虚拟化

- 因为 VMM 掌控有所有系统资源，因此 VMM 握有整个内存资源，其负责页式内存管理，维护虚拟地址到机器地址的映射关系。
- 因 Guest OS 本身亦有页式内存管理机制，则有 VMM 的整个系统就比正常系统多了一层映射：
  - 虚拟地址 (VA)，指 Guest OS 提供给其应用程序使用的线性地址空间
  - 物理地址 (PA)，经 VMM 抽象的、虚拟机看到的伪物理地址
  - 机器地址 (MA)，真实的机器地址，即地址总线上出现的地址信号
- 映射关系
  - Guest OS: PA = f(VA)； VMM: MA = g(PA)
- VMM 维护一套页表，负责 PA 到 MA 的映射。
- Guest OS 维护一套页表，负责 VA 到 PA 的映射。
- 实际运行时，用户程序访问 VA，经 Guest OS 的页表转换得到 PA，再由 VMM 介入，使用 VMM 的页表将 PA1 转换为 MA。

页表虚拟化基本思想

- 普通 MMU 只能完成一次虚拟地址到物理地址的映射，在虚拟机环境下，经过 MMU 转换所得到的“物理地址”并不是真正的机器地址。若需得到真正的机器地址，必须由 VMM 介入，再经过一次映射才能得到总线上使用的机器地址。
- 显然，如果虚拟机的每个内存访问都需要 VMM 介入，并由软件模拟地址转换的效率是很低下的，几乎不具有实际可用性，为实现虚拟地址到机器地址的高效转换，现普遍采用的思想是：
- 由 VMM 根据映射 f 和 g 生成复合的映射 fg，并直接将这个映射关系写入 MMU，其可行性在于：
  - VMM 维护着映射 g
  - VMM 能够访问 Guest OS 的内存，因此可以直接查询 Guest OS 的页表，从而获得映射 f
  - 计算复合映射 fg 能够在恰当的时候高效地进行
- 目前采用的页表虚拟化方法主要有两个： MMU 类虚拟化 (MMU Paravirtualization) 和影子页表

MMU Para-virtualization

- 该技术为 Xen 所采用。
- 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 Xen 注册该页面，Xen 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 Xen 加以验证和转换。
- Xen 会检查页表中的每一项，确保他们只映射了属于该虚拟机的机器页面，而且不得包含对页表页面的可写映射。
- 然后，Xen 会根据自己所维护的映射关系 g，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。

影子页表

- 类虚拟化需要修改 Guest OS。对全虚拟化，则使用影子页表 (Shadow Page Table) 技术来实现。
- VMWare Workstation, VMWare ESX Server 和 KVM for x86 都是使用该技术。
- 与类虚拟化类似，影子页表技术也是采用将复合映射 fg 直接写入 MMU 的思路。不同在于类虚拟化技术直接将 fg 更新到 Guest OS 的页表项中，而影子页表则是 VMM 为 Guest OS 的每个页表维护一个“影子页表“，并将合成后的映射关系写入到”影子“中，Guest OS 的页表内容则保持不变。最后 VMM 将影子页表写入 MMU。

影子页表的实现挑战

- 影子页表的实现挑战在于其时间和空间的开销很大。
  - 时间：由于 Guest OS 构造页表时不会主动通知 VMM，VMM 必须等到 Guest OS 发生缺页时通过分析缺页原因，再为其补全影子页表。
  - 空间：VMM 需要支持多个虚拟机同时运行，而每个虚拟机的 Guest OS 通常会为其上运行的每个进程都创建一套页表系统，因此影子页表的空间开销会随着进程数量的增多而迅速增大。
- 在时间开销和空间开销中做出权衡的方法是使用影子页表缓存 (Shadow Page Table Cache)，即 VMM 在内存中维护部分最近使用过的影子页表，只有当影子页表在缓存中找不到时，才构建一个新的，当前主要的全虚拟化都采用了影子页表缓存技术。

I/O 虚拟化

- VMM 通过 I/O 虚拟化来复用有限的外设资源，其通过截获 Guest OS 对 I/O 设备的访问请求，然后通过软件模拟真实的硬件。
- 软件通过 I/O设备的一堆的状态寄存器、控制寄存器、中断（有些设备有内部存储）与其交互。
- 目前 I/O 设备的虚拟化方式主要有三种：
  - 设备接口完全模拟
  - 前端 / 后端模拟
  - 直接划分

设备接口完全模拟

- 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：没有额外的硬件开销，可重用现有驱动程序。
- 缺点：为完成一次操作要涉及到多个寄存器的操作，使得 VMM 要截获每个寄存器访问并进行相应的模拟，这就导致多次上下文切换；由于是软件模拟，性能较低。

前端／后端模拟

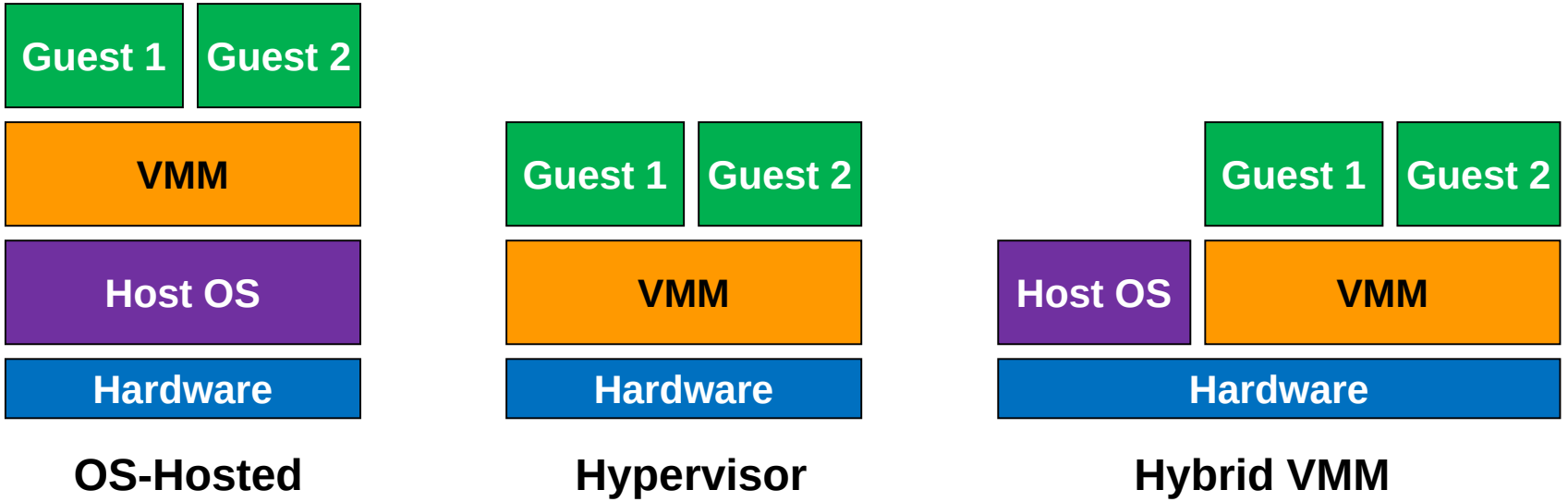
- VMM 提供一个简化的驱动程序（后端, Back-End）
- Guest OS 中的驱动程序为前端 (Front-End, FE)
- 前端驱动将来自其他模块的请求通过与 Guest OS 间的特殊通信机制直接发送给 Guest OS 的后端驱动，后端驱动在处理完请求后再发回通知给前端。
- Xen 即采用该方法。
- 优点：基于事务的通信机制，能在很大程度上减少上下文切换开销；没有额外的硬件开销
- 缺点：需要 VMM 实现前端驱动；后端驱动容易成为瓶颈

直接划分

- 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 VMM）
- 目前与此相关的技术有 IOMMU（Intel VT-d, PCI-SIG 之 SR-IOV 等）
- 优点：重用已有驱动；直接访问减少了虚拟化开销；
- 缺点：需要购买较多额外硬件。

VMM结构

- 当前主流的 VMM (Virtual Machine Monitor) 实现结构可以分为三类：
  - 宿主模型 (OS-hosted VMMs)
  - Hypervisor 模型 (Hypervisor VMMs)
  - 混合模型 (Hybrid VMMs)



宿主模型

- 该结构的 VMM，物理资源由 Host OS (Windows, Linux etc.) 管理
- 实际的虚拟化功能由 VMM 提供，其通常是 Host OS 的独立内核模块（有的实现还含用户进程，如负责 I/O 虚拟化的用户态设备模型）
- VMM 通过调用 Host OS 的服务来获得资源，实现 CPU，内存和 I/O 设备的虚拟化
- VMM 创建出 VM 后，通常将 VM 作为 Host OS 的一个进程参与调度
- 优点：可以充分利用现有 OS 的设备驱动，VMM 无需自己实现大量的设备驱动，轻松实现 I/O 设备的虚拟化。
- 缺点：因资源受 Host OS 控制，VMM 需调用 Host OS 的服务来获取资源进行虚拟化，其效率和功能会受到一定影响。
- 采用该结构的 VMM 有：VMware Workstation, VMWare Server (GSX), Virtual PC, KVM（早期）

Hypervisor 模型

- 该结构中，VMM 可以看作一个为虚拟化而生的完整 OS，掌控有所有资源（CPU，内存，I/O 设备）
- VMM 承担管理资源的重任，其还需向上提供 VM 用于运行 Guest OS，因此 VMM 还负责虚拟环境的创建和管理。
- 优点：因 VMM 同时具有物理资源的管理功能和虚拟化功能，故虚拟化的效率会较高；安全性方面，VM 的安全只依赖于 VMM 的安全
- 缺点：因 VMM 完全拥有物理资源，因此，VMM 需要进行物理资源的管理，包括设备的驱动，而设备驱动的开发工作量是很大的，这对 VMM 是个很大的挑战。
- 采用该结构的 VMM 有：VMWare ESX Server, KVM（后期）

混合模型

- 该结构是上述两种模式的混合体，VMM 依然位于最底层，拥有所有物理资源，但 VMM 会主动让出大部分 I/O 设备的控制权，将它们交由一个运行在特权 VM 上的特权 OS 来控制。VMM 只负责 CPU 和内存的虚拟化，I/O 设备的虚拟化由 VMM 和特权 OS 共同完成。
- 优点：可利用现有 OS 的 I/O 设备驱动；VMM 直接控制 CPU 和内存等物理资源，虚拟化效率较高；若对特权 OS 的权限控制得当，虚拟机的安全性只依赖于 VMM。
- 缺点：因特权 OS 运行于 VM 上，当需要特权 OS 提供服务时，VMM 需要切换到特权 OS，这里面就产生上下文切换的开销。
- 采用该结构的 VMM 有：Xen

Applications of Virtualization (1/2)

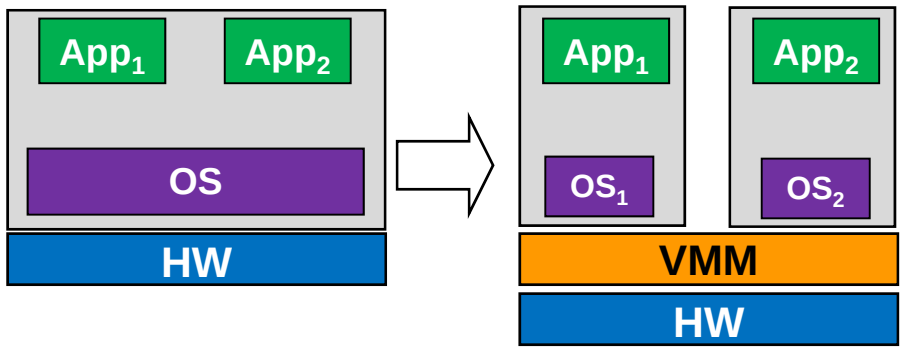
- Compatibility (ISA and/or operating system)
  - Ancient VAX/VMS software
- Hardware independence (e.g., Java's VM)
  - Related: Transmeta's code morphing
- Setup of new software test environments
  - Software migration
- Development and debugging
  - Complex software systems
  - Processor design (simulation vs emulation)
  - General performance analysis

Applications of Virtualization (2/2)

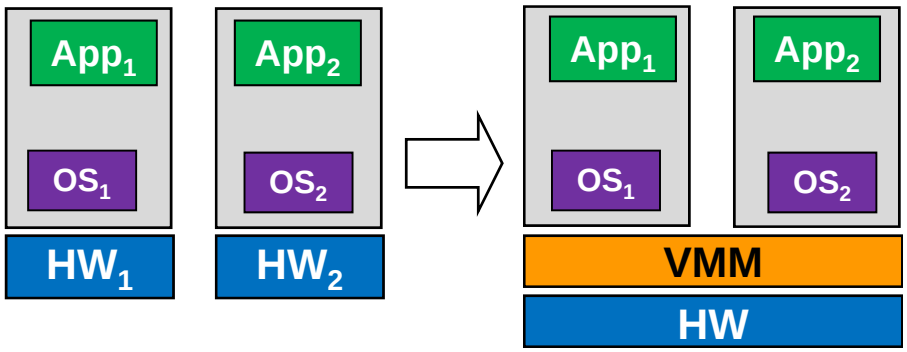
- Error and attack containment
  - Why aren't processes sufficient?
  - Faults and security considerations
- Dynamically share resources (memory, disk, processors)
  - Sever consolidation
  - 10 virtual machine web servers on a two-processor machine
  - Virtual machines "leased" from real server owner
- Server migration
  - Move a server across the room (or across the world)

虚拟化应用总结：隔离、整合与迁移

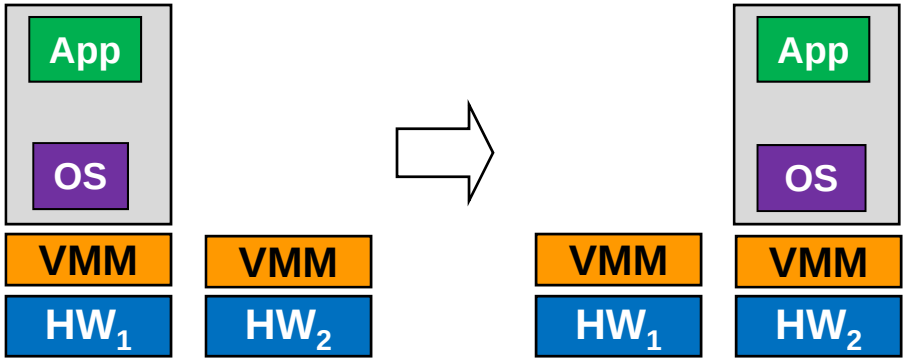
隔离



整合



迁移



N. Sahgal, D. Rodgers, Understanding Intel Virtualization Technology

虚拟化之性能比较

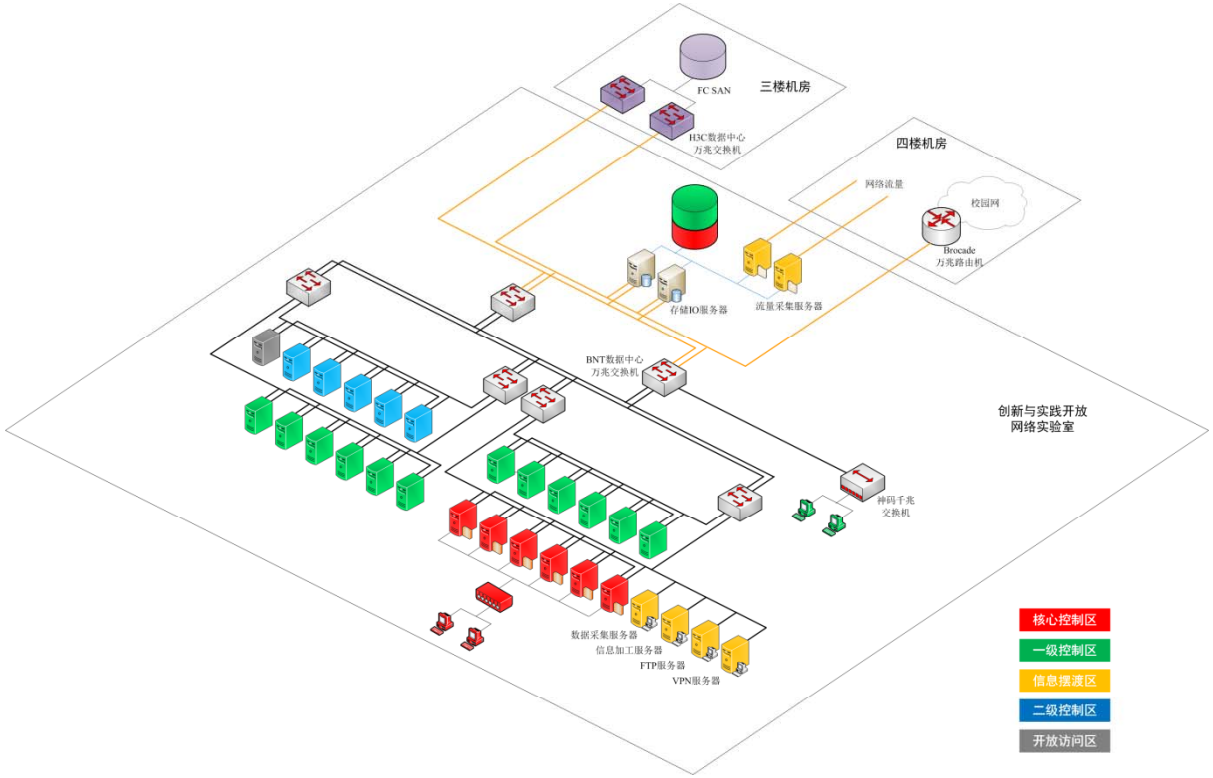
- 选手
  - Vmware、Virtualbox、KVM、XEN
- 项目
  - 单项赛
    - CPU、缓存和内存、I/O磁盘、网络
  - 综合赛
    - Web服务器、数据库、FTP、文件压缩
- 结果
  - Virtual machines performance comparison: vmware vs virtualbox vs kvm vs xen http://www.ilsistemista.net/index.php/virtualization/1-virtual-machines-performance-comparison.html

Advanced Topics

- Memory over-commit
- Nested virtualization
- Live migration
- Network virtualization
- Cloud computing
- ……

No lecture, wait for your exploration!

Open IT Lab with the IP way!



- 全万兆数据中心
- 与校园网骨干互通
- 开放实验室
- IP之道：**I**nnovation & **P**ractice; **I**nternet **P**rotocol

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Welcome to join us!

Send your CV and Transcript to
jinyh@sjtu.edu.cn