

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

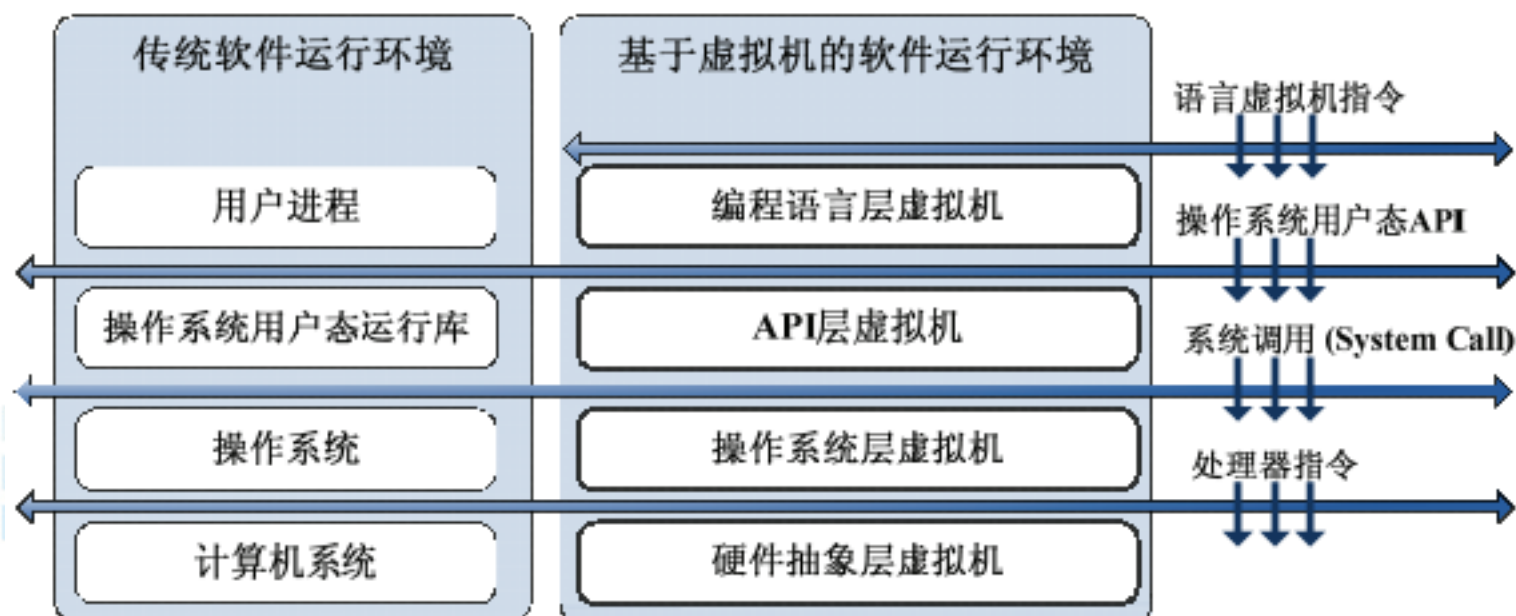
2021年3月16日下午14:00-15:30

内容回顾



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)



什么是系统虚拟机

- Popek and Goldberg(1974)

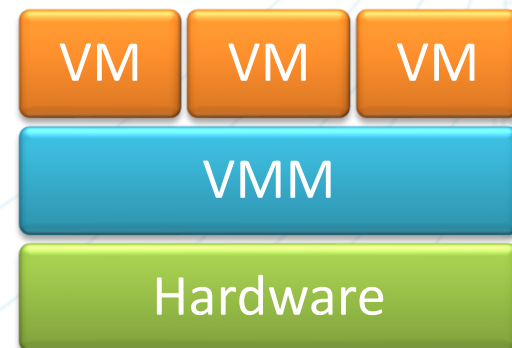
- 虚拟化典型特征

- 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

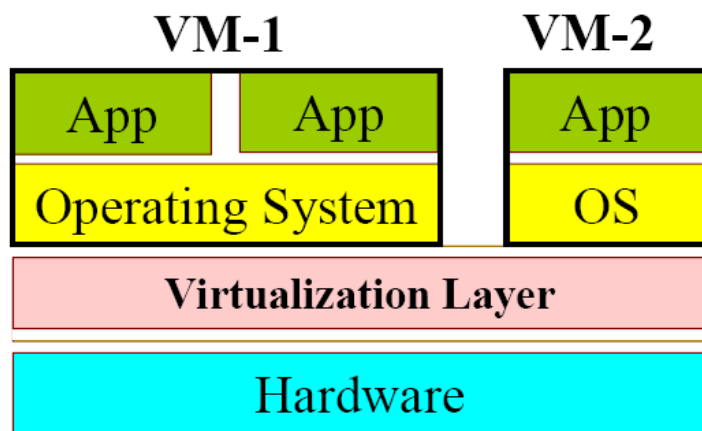
- 管理硬件设备
- 对上层VM呈现出虚拟的硬件平台



经典虚拟机模型

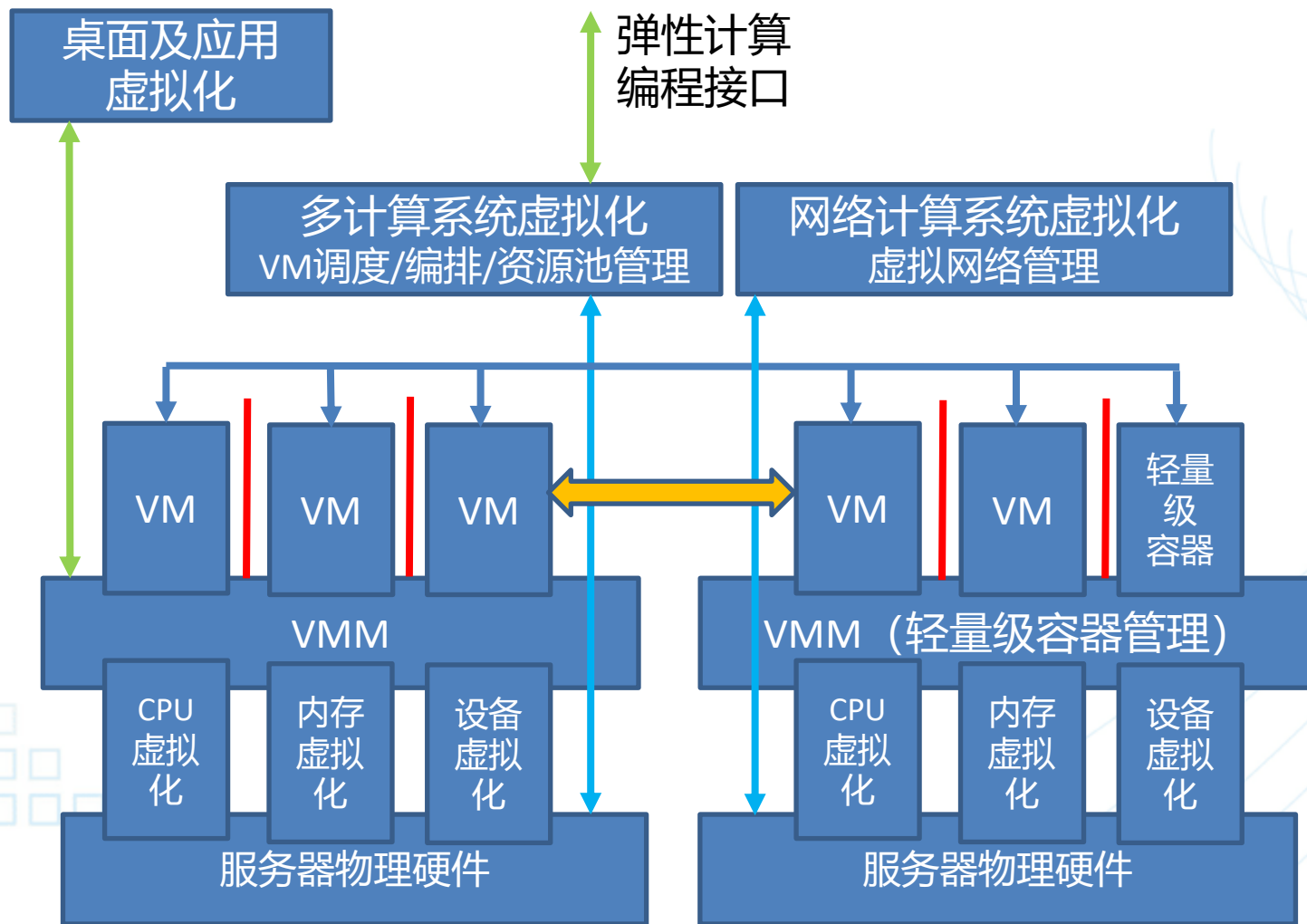
系统虚拟化技术是

- 虚拟化是在“硬件”和“软件”之间的一种抽象



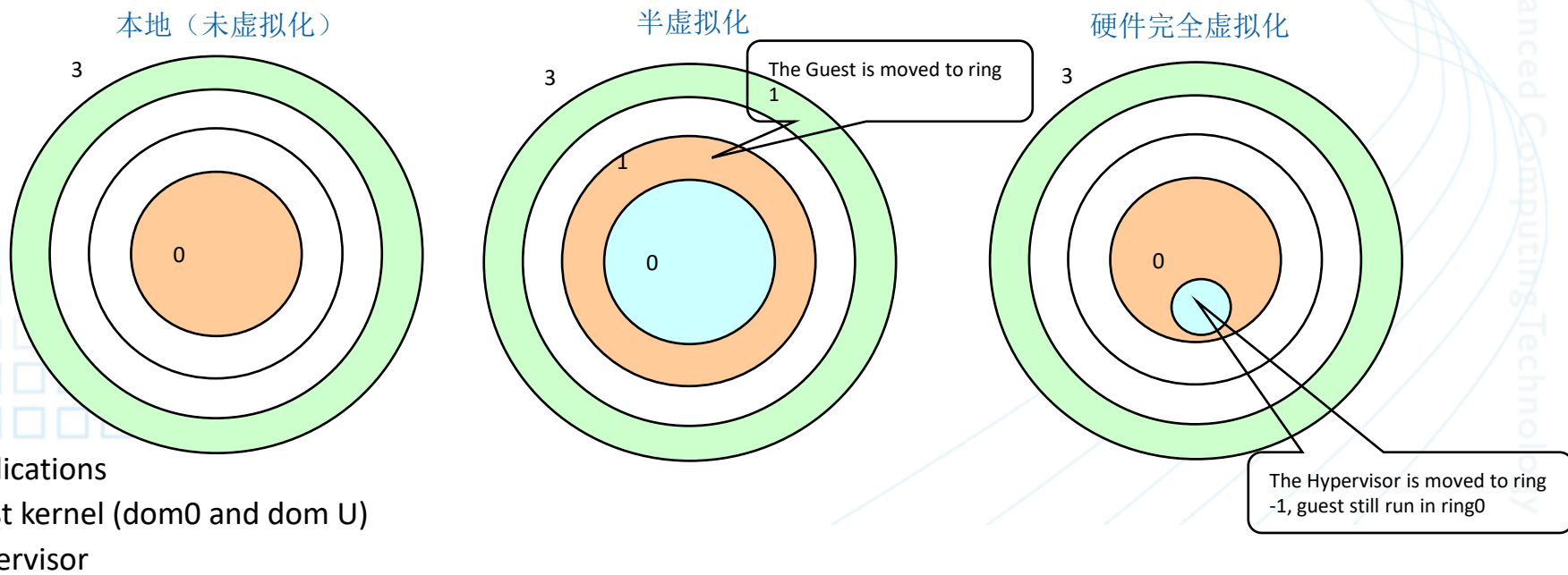
- 主要的优势：**封装、隔离、灵活、便于迁移**
 - 有效提高系统可控可管能力
 - 增强系统可信保障能力
- 核心：虚拟机监控器(VMM)
 - 由于硬件提供的虚拟化支持，虚拟化后的性能开销可控
 - 追求的目标：轻载、高效

计算系统虚拟化研究



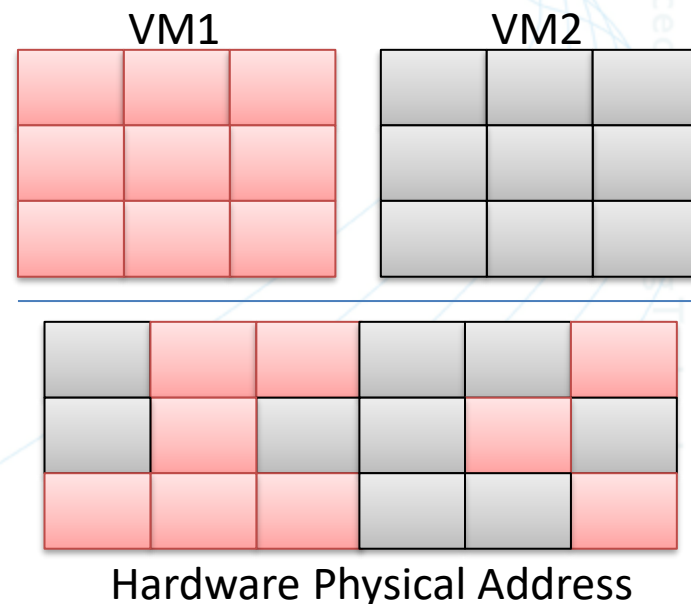
CPU 虚拟化

- Native: Ring 0运行特权指令； Ring 3运行非特权指令
- Complete software interpreter machine (CSIM): performance penalty, violate Gold
- Binary Translation: 扫描指令流，识别敏感指令，跳转到等价的模拟指令
- Para-Virtualization: 将Guest kernel移到ring 1 (Xen)
- Hardware-Assist: Kvm Hypervisor运行在ring -1



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）
 - It sees hardware address, but there is only one “address 0”
 - Must virtualizes hardware memory to “fool” VMs
 - Must protect itself from VM
 - Responsible for allocating and managing memory
- VM addressing
 - GVA -> GPA -> HPA
- Key Issues
 - Maintain GPA<->HPA mapping
 - Intercept guest access to GPA, and translate it



I/O 虚拟化

- 虚拟设备 (Virtual Device)

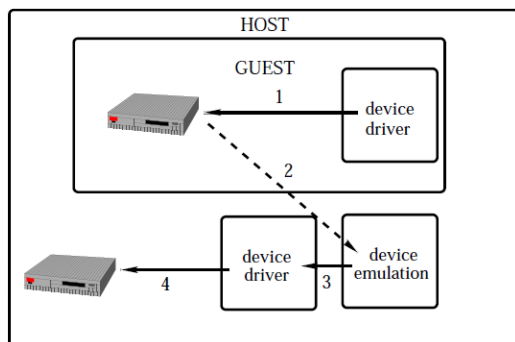
- Device Model

- 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

- Software Interface

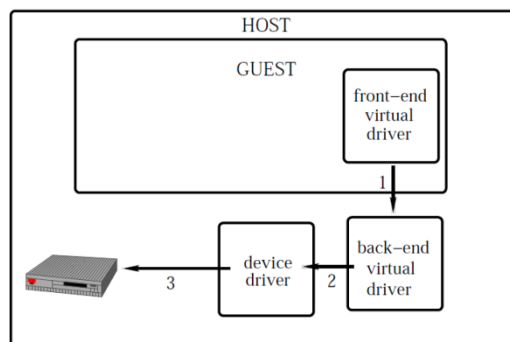
- A PCI Device: (1)config;(2)PIO; (3)MMIO; (4) DMA; (5) Interrupt

- Interface Interception and emulation



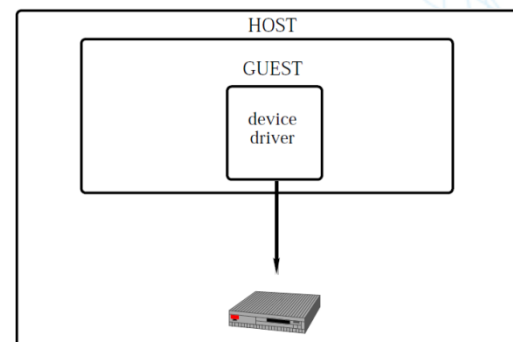
Emulation

- (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

网络的虚拟化



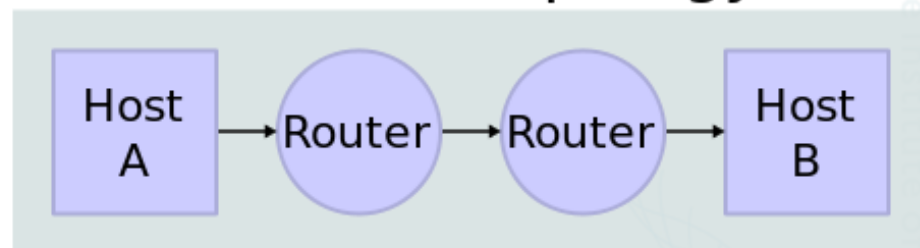
网络研究什么

- 硬件：交换（分组、链路）、路由
- 软件：协议
- 核心：共享与协同

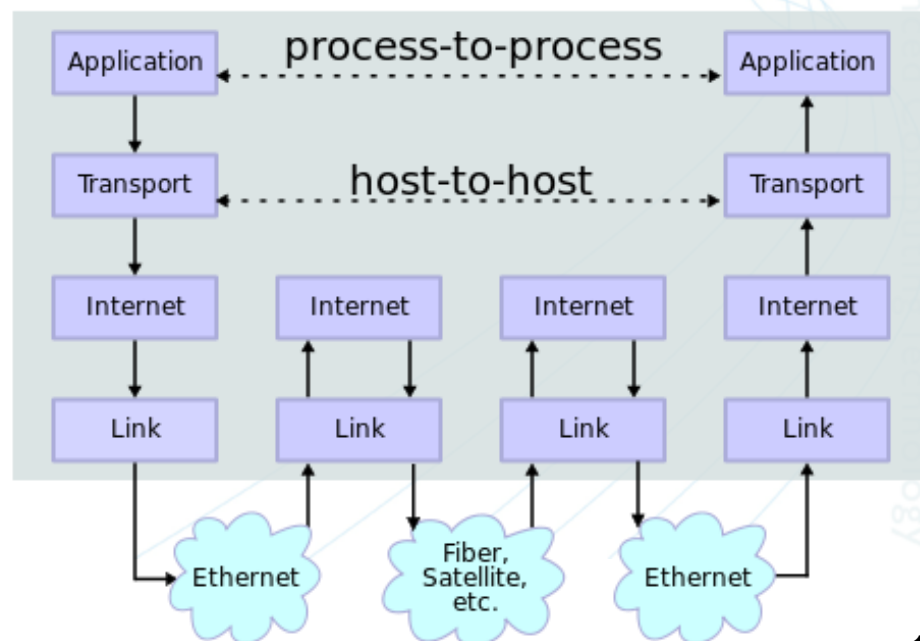
现有网络体系结构

- 逐个网络交换
- 端到端虚通讯

Network Topology

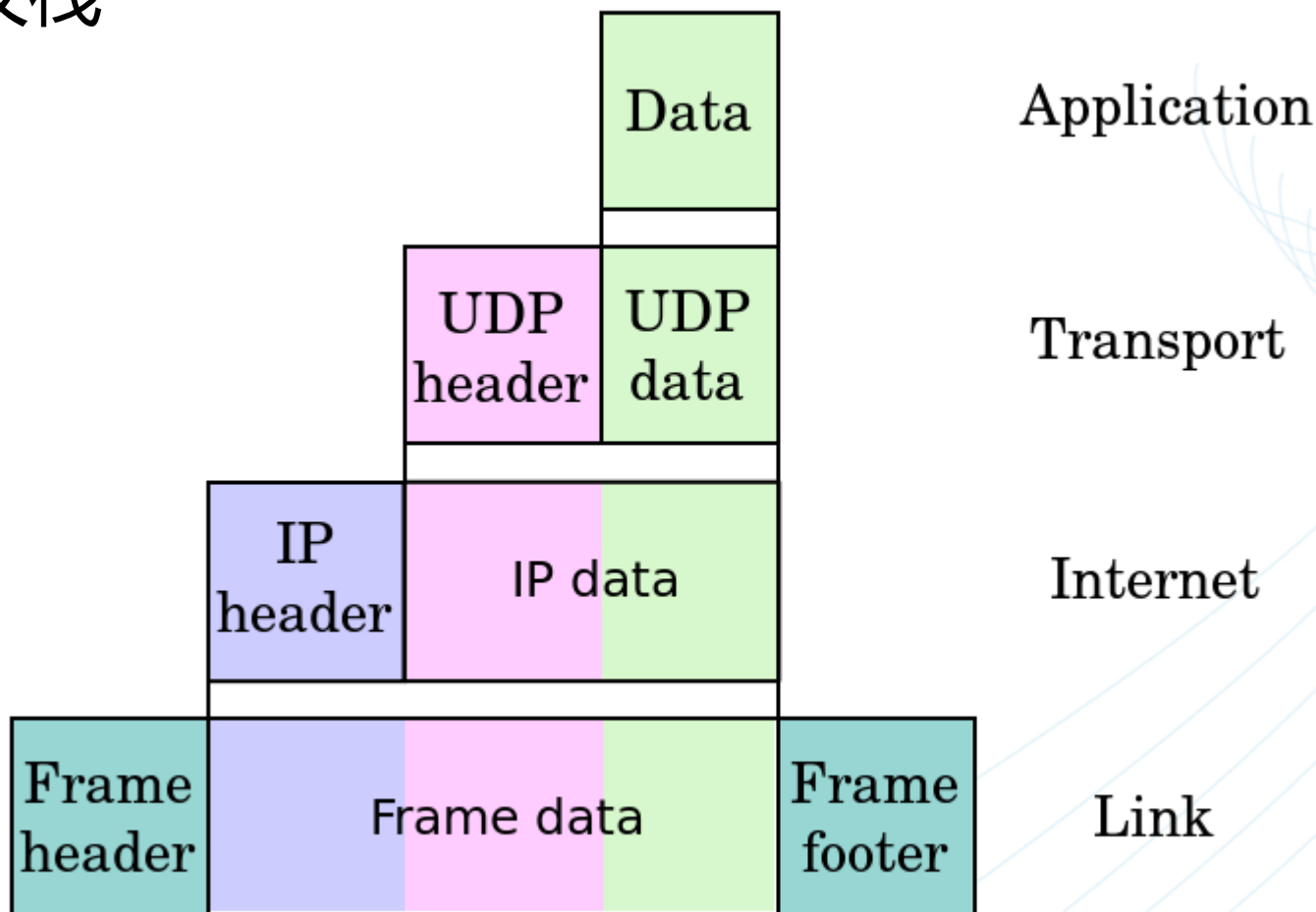


Data Flow



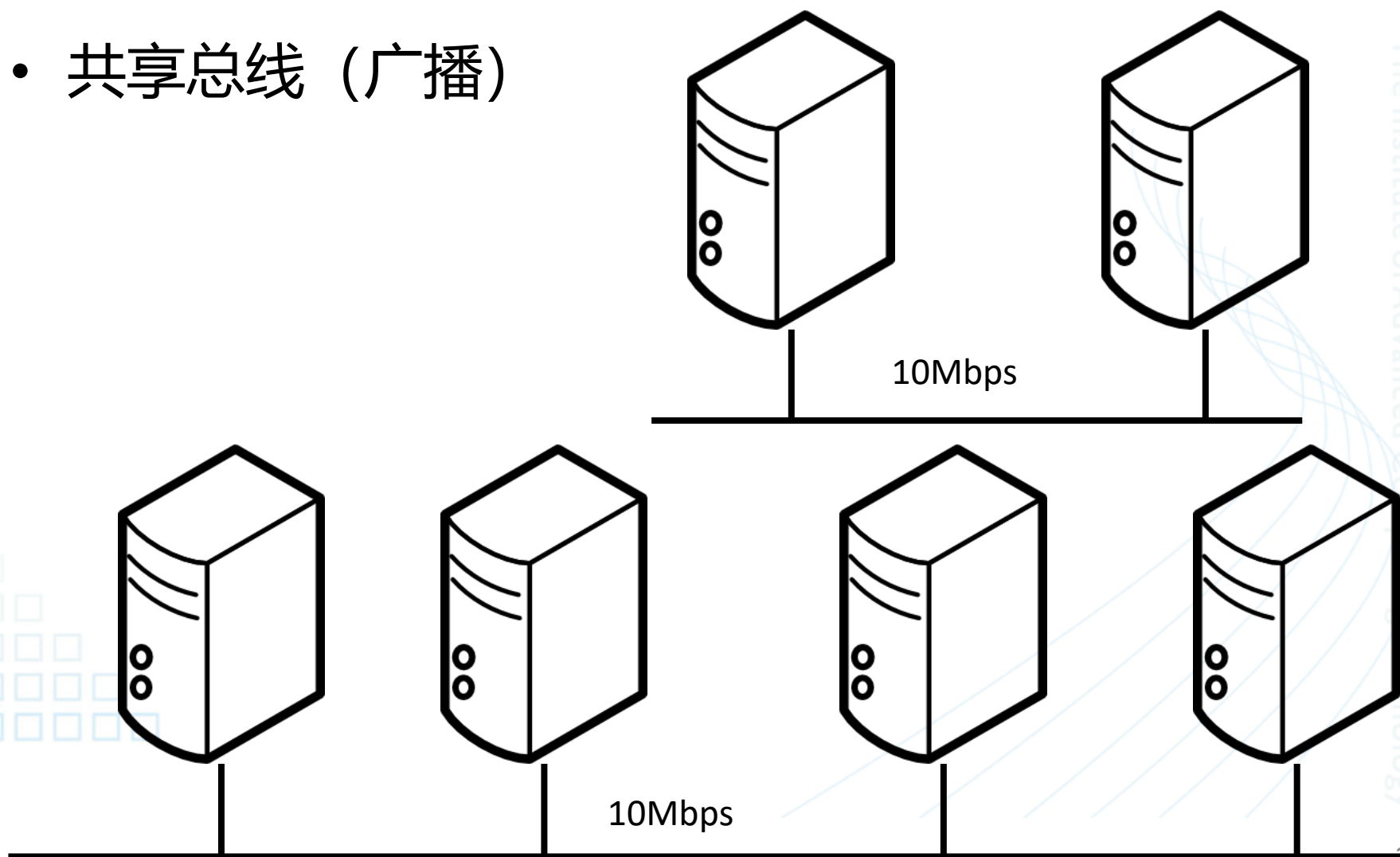
现有网络体系结构

- 协议栈



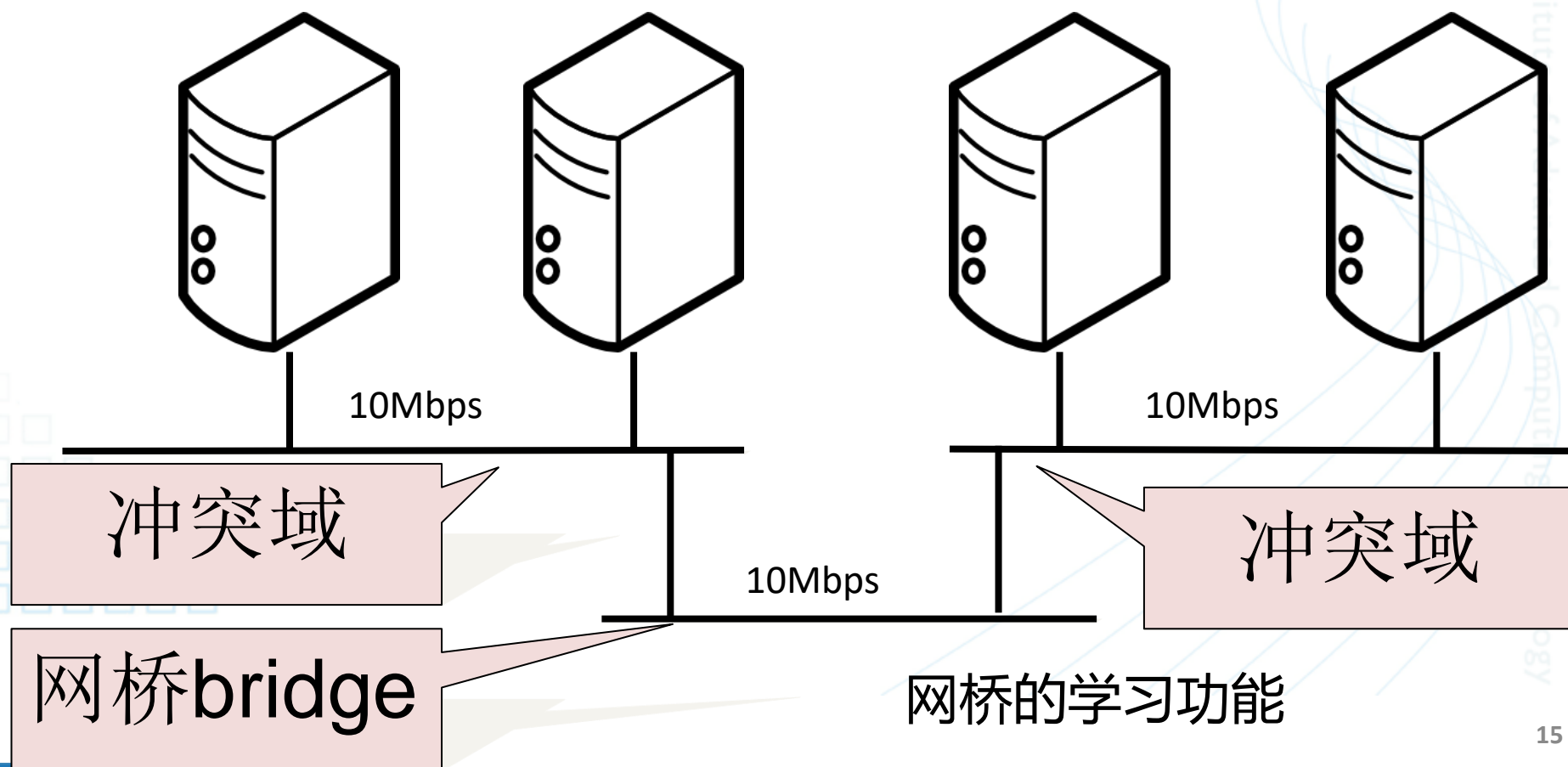
网络交换

- 共享总线（广播）



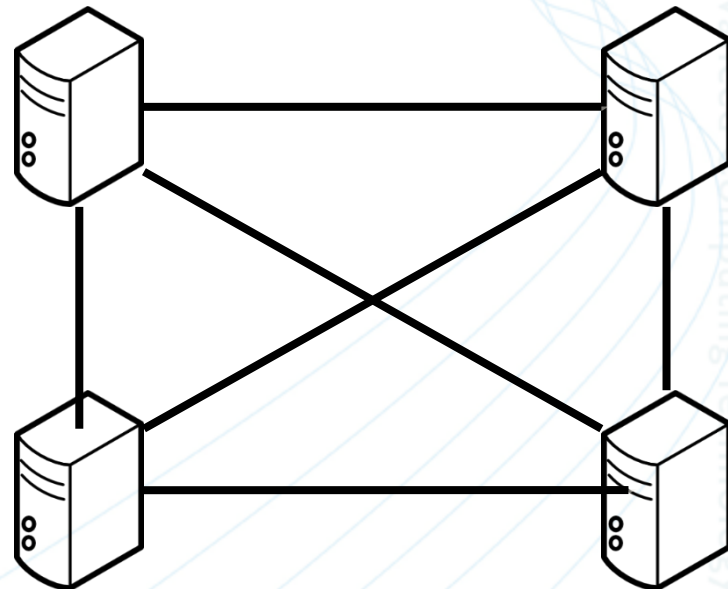
网路交换

- 网桥——隔离与共享的矛盾



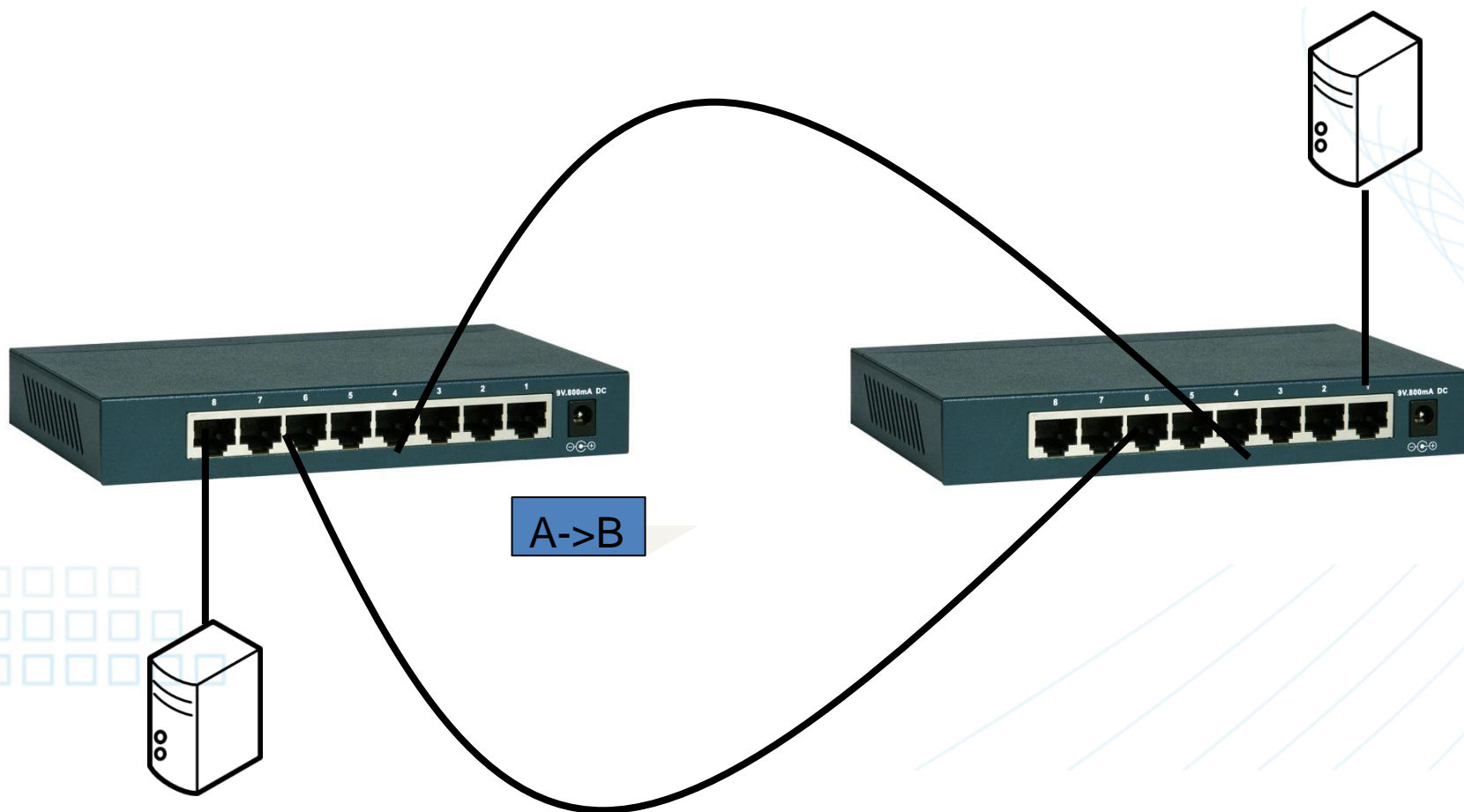
如何提高交换效率

- 尽量减小冲突
 - 极端情况将每台机器放在一个冲突域中
- 交换机：多端口网桥
 - 硬件化（固件化）
 - 高效、线速



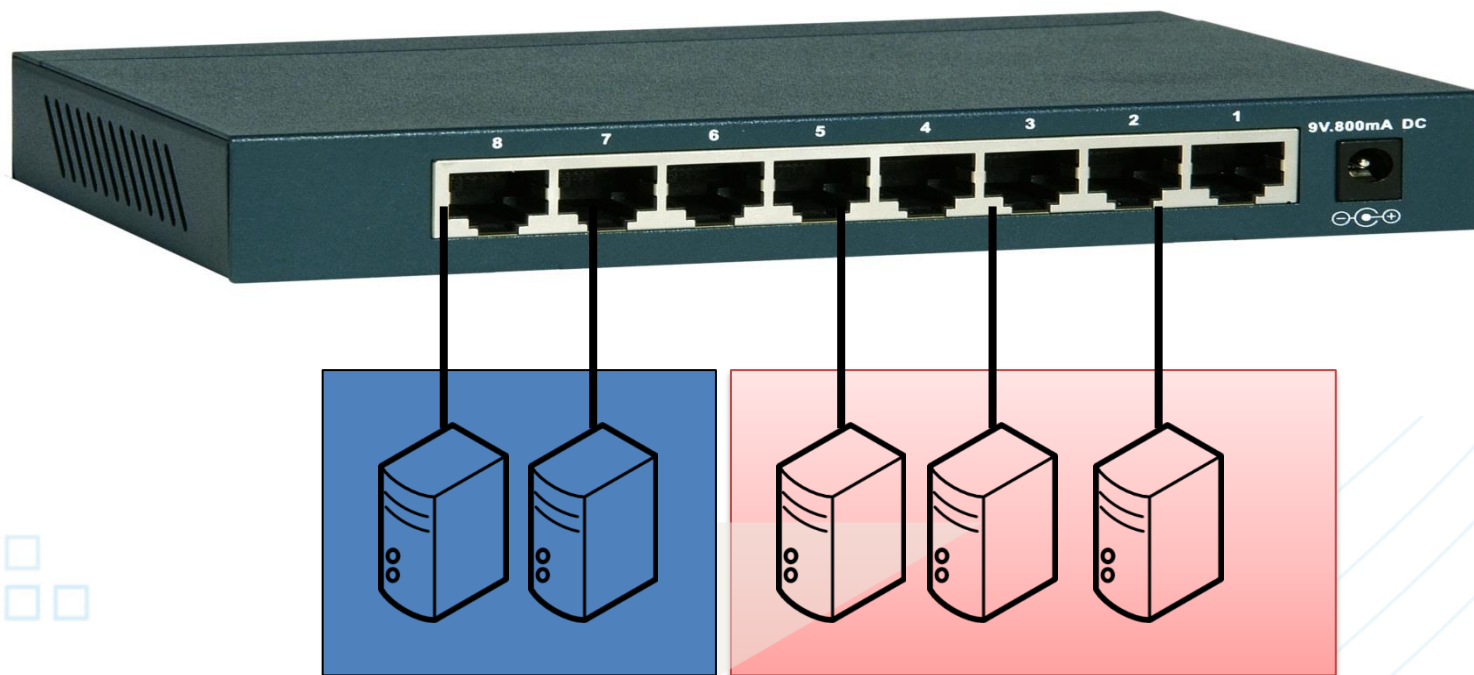
交换机的主要功能

- 避免循环交换——STP



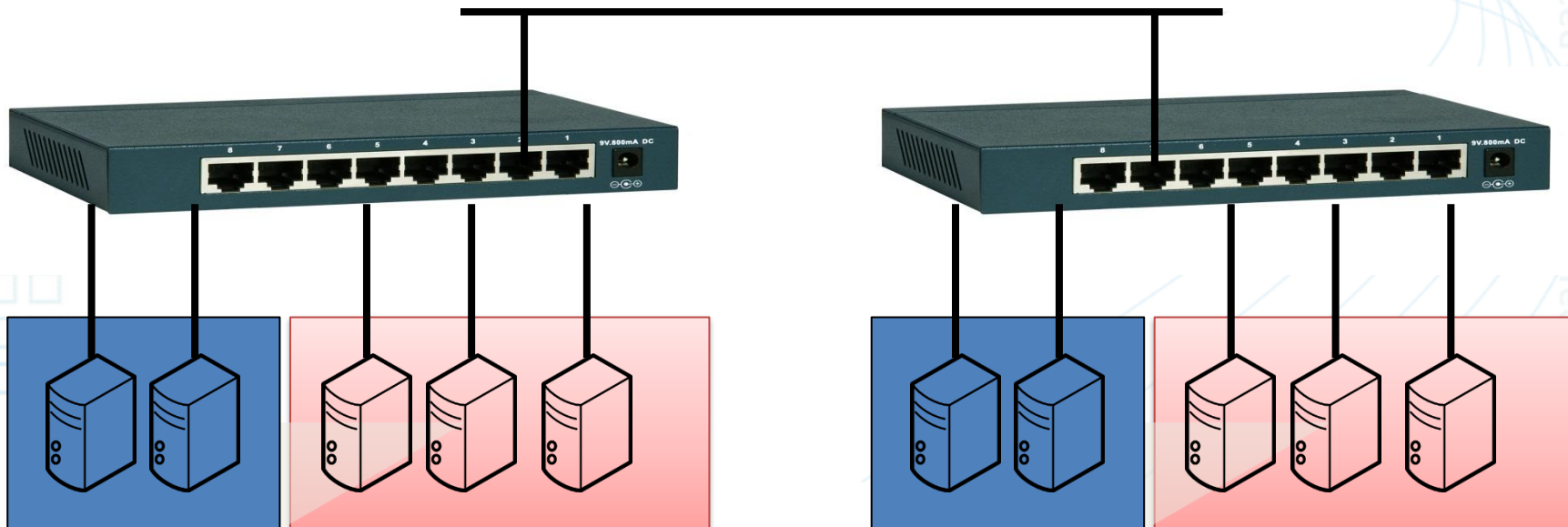
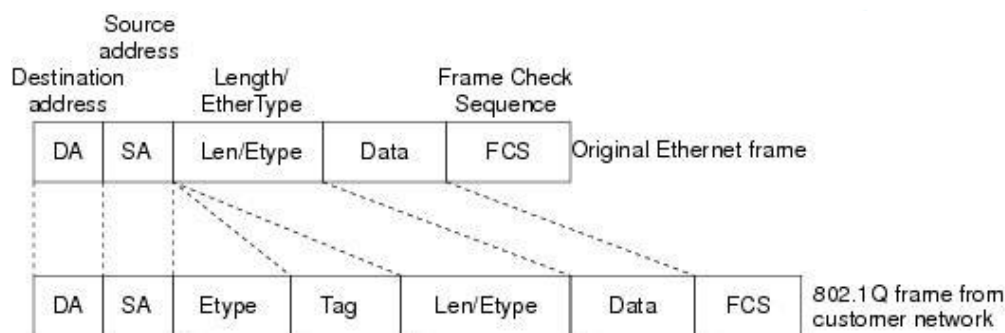
交换机的主要功能

- 划分VLAN



交换机的主要功能

- 跨交换机VLAN——802.1Q

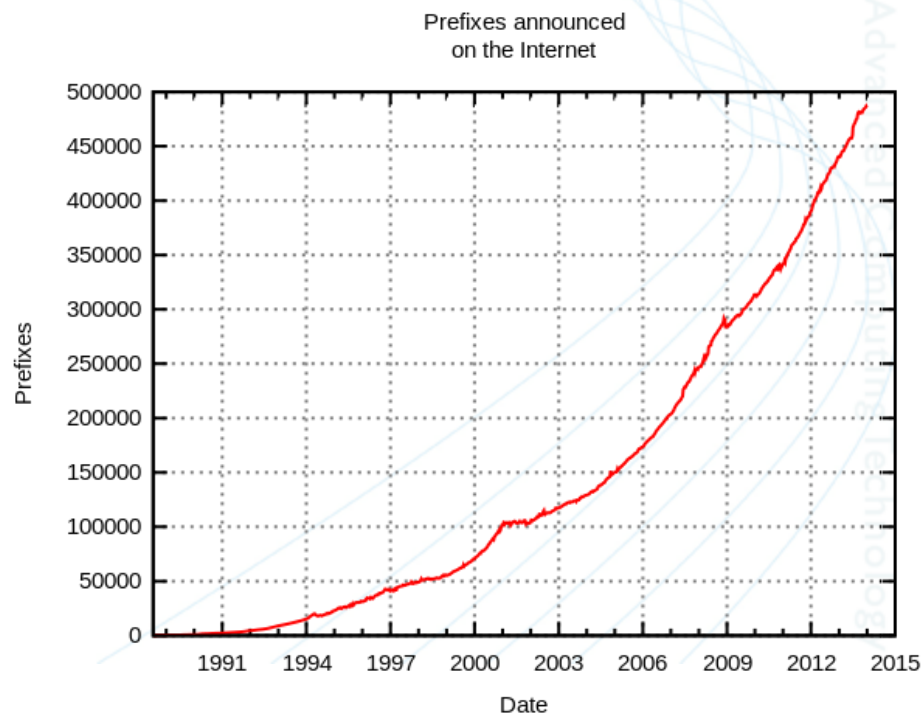


路由 vs 交换

- 交换机（网桥）的局限
 - 网络规模不断扩大
 - 网络管理、通信性能
 - 无法阻止广播
- 路由器：连接两个以上网络的设备
 - 树 vs 网
 - 多路径冗余
 - 基于“路由表”，提供更多转发依据

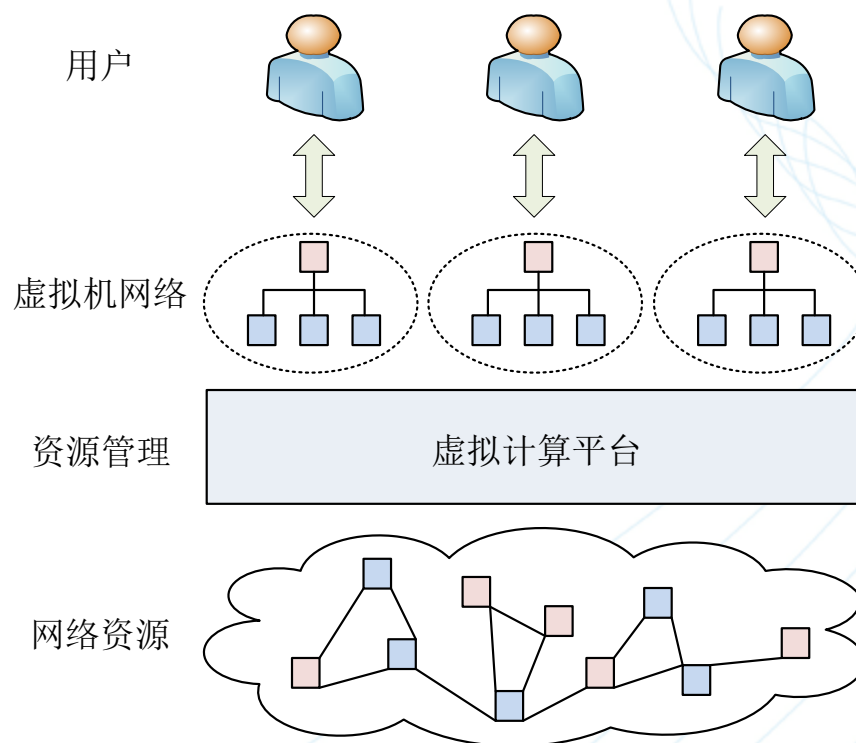
路由

- 路由协议
 - 静态路由：手工设定路由表
 - E.g. 默认路由（网关）
 - 动态路由
 - 距离矢量：RIP
 - 链路状态：OSPF
 - AS间路由协议：BGP



物理网络 vs 虚拟网络

- What Changes?
 - 动态性
 - 资源共享
 - 可管理性



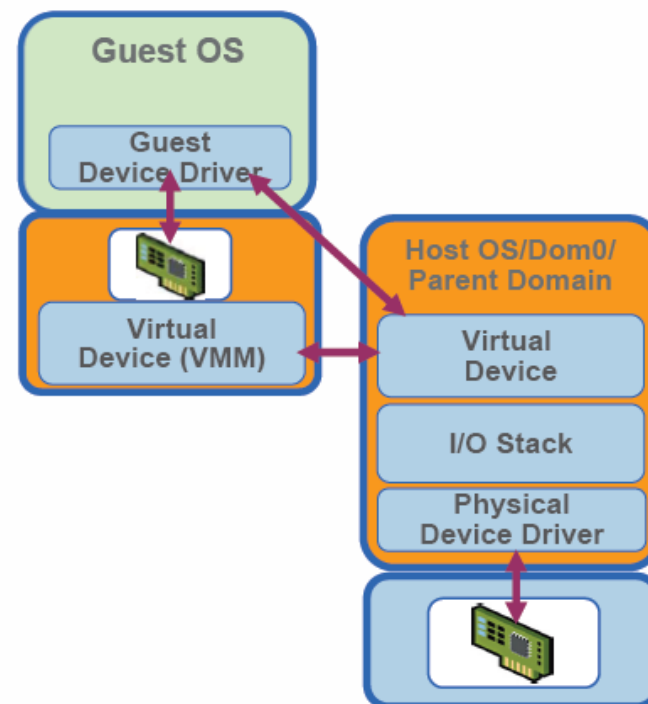
虚拟机组网

- 建立节点网络（虚拟网卡）
 - 不同网络虚拟化方案
- 建立二层网络
 - “物理” 连接
 - 地址分配
- 建立三层网络
 - “物理” 连接
 - 地址分配

虚拟网卡

- 软件模拟硬件
 - QEMU User Networking (SLIRP) (NAT)
 - 无需root权限
 - 共享性好
 - 性能差
 - 不支持ICMP
 - Guest不能被外网访问

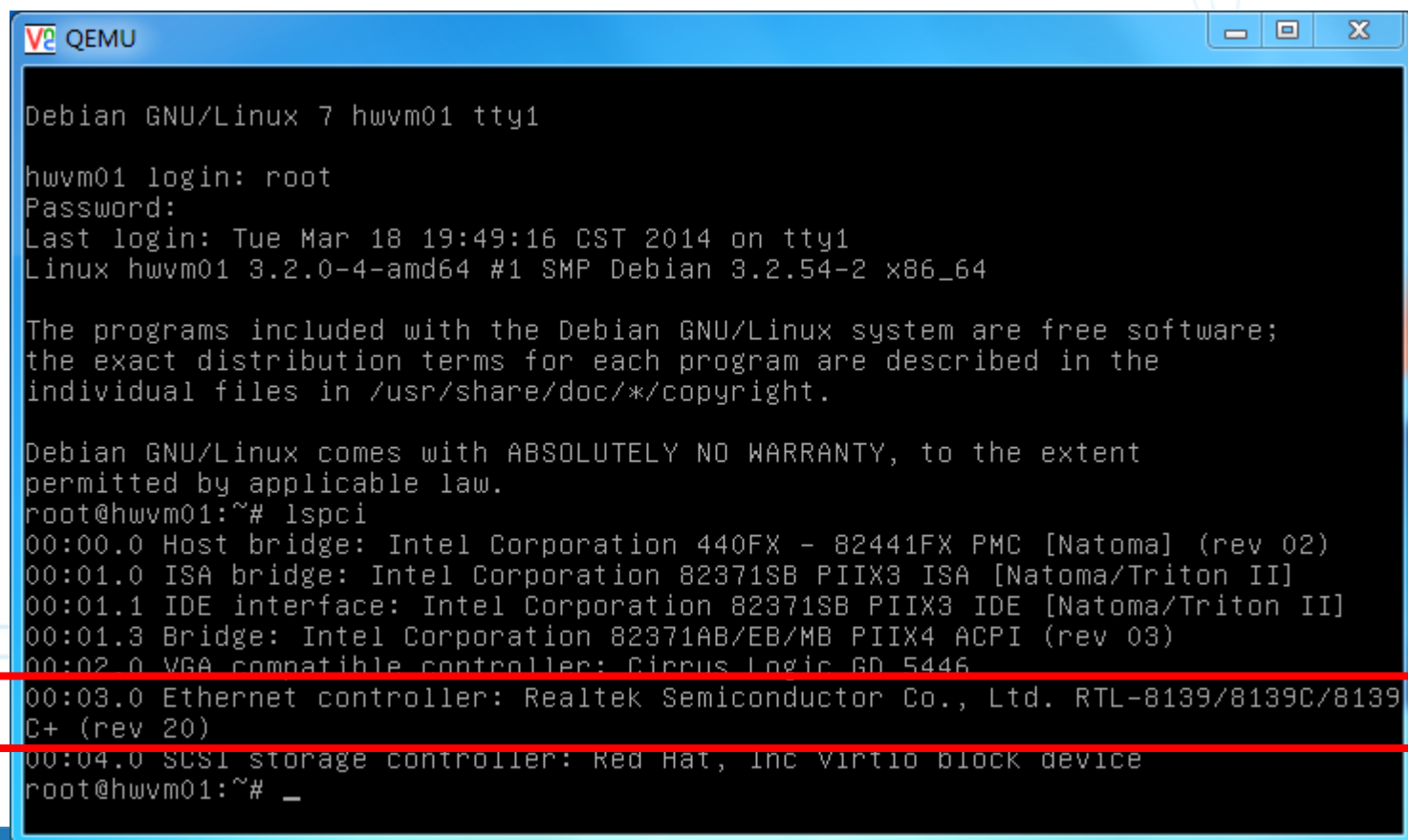
– `kvm -net nic,model=?`



虚拟网卡

- 软件模拟硬件(Emulation)

`kvm -vnc :1 -drive file=vm1.cow,if=virtio -m 4096m`



```
QEMU
Debian GNU/Linux 7 hwvm01 tty1

hwvm01 login: root
Password:
Last login: Tue Mar 18 19:49:16 CST 2014 on tty1
Linux hwvm01 3.2.0-4-amd64 #1 SMP Debian 3.2.54-2 x86_64

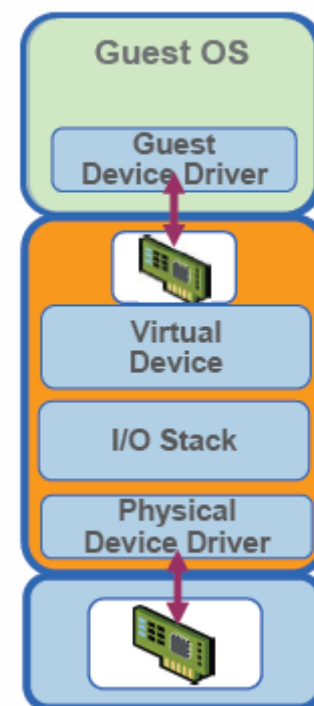
The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
root@hwvm01:~# lspci
00:00.0 Host bridge: Intel Corporation 440FX - 82441FX PMC [Natoma] (rev 02)
00:01.0 ISA bridge: Intel Corporation 82371SB PIIX3 ISA [Natoma/Triton II]
00:01.1 IDE interface: Intel Corporation 82371SB PIIX3 IDE [Natoma/Triton II]
00:01.3 Bridge: Intel Corporation 82371AB/EB/MB PIIX4 ACPI (rev 03)
00:02.0 VGA compatible controller: Cirrus Logic GD 5446
00:03.0 Ethernet controller: Realtek Semiconductor Co., Ltd. RTL-8139/8139C/8139C+ (rev 20)
00:04.0 SCSI storage controller: Red Hat, Inc Virtio block device
root@hwvm01:~# _
```

虚拟网卡

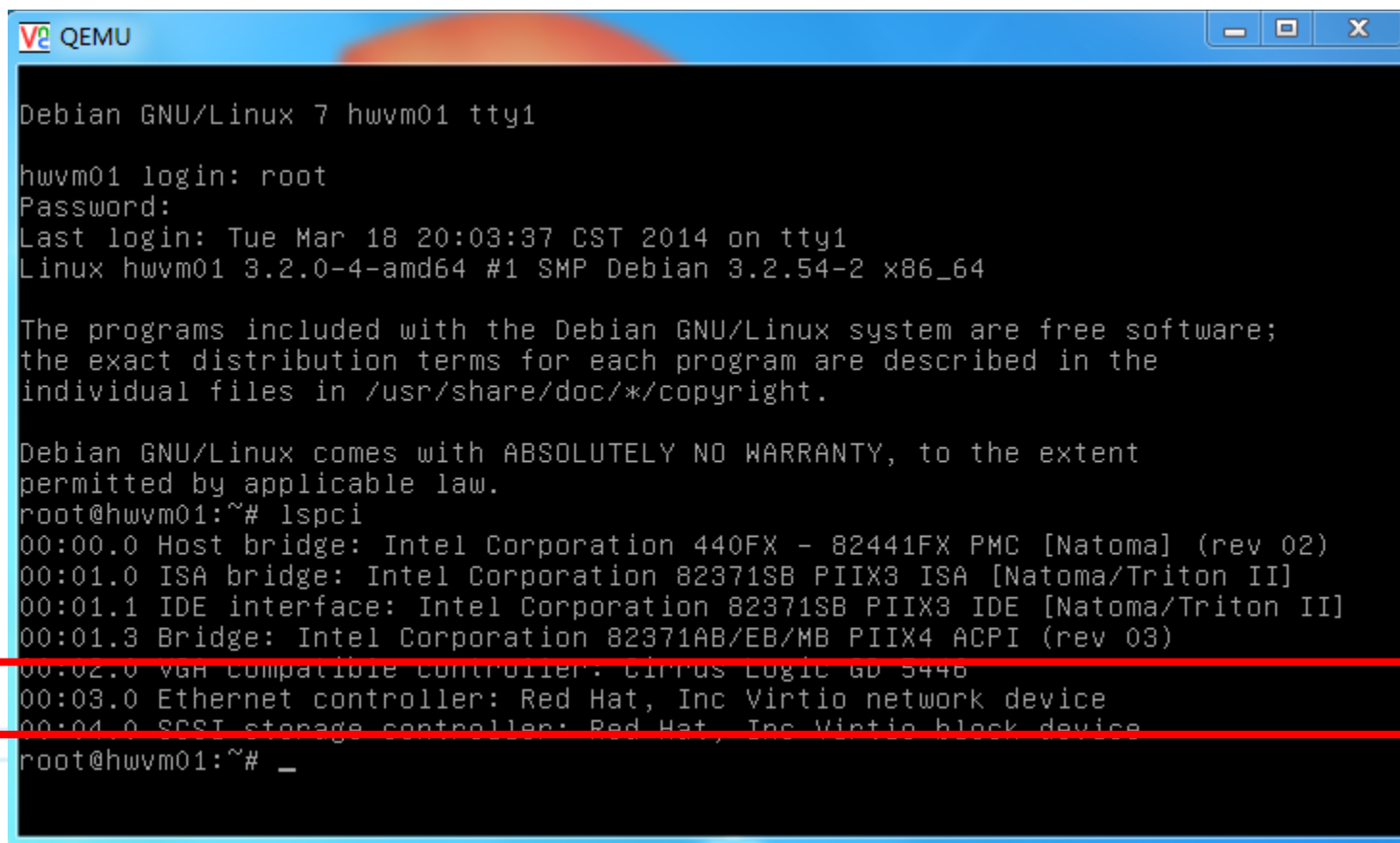
- 半虚拟化网卡(Virtual Split Driver)
 - 性能好
 - 可共享物理硬件
 - Guest内部需要专用驱动
 - Redhat: virtio
 - XEN: xen-pv
 - VMWare: vmxnet2、vmxnet3

Hypervisor Direct



虚拟网卡

- `kvm -vnc :5 -drive file=vm1.cow,if=virtio -m 4096m -device virtio-net-pci`



The screenshot shows a QEMU window titled 'QEMU' with a terminal output. The terminal shows a Debian GNU/Linux 7 login prompt. The user 'root' logs in successfully. The terminal then displays the output of the 'lspci' command, which lists the hardware configuration of the virtual machine. A red rectangle highlights the line '00:03.0 Ethernet controller: Red Hat, Inc Virtio network device', indicating the presence of a virtual network card.

```
Debian GNU/Linux 7 hvm01 tty1
hvm01 login: root
Password:
Last login: Tue Mar 18 20:03:37 CST 2014 on tty1
Linux hvm01 3.2.0-4-amd64 #1 SMP Debian 3.2.54-2 x86_64

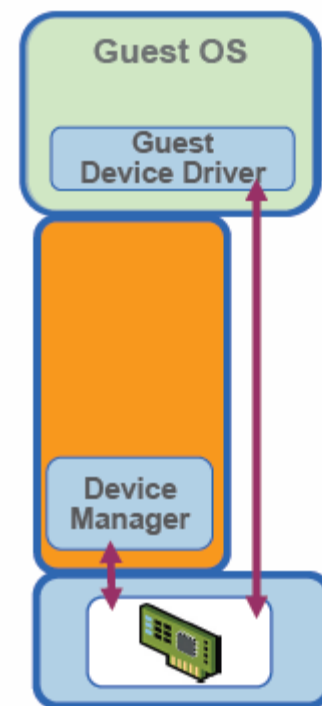
The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
root@hvm01:~# lspci
00:00.0 Host bridge: Intel Corporation 440FX - 82441FX PMC [Natomia] (rev 02)
00:01.0 ISA bridge: Intel Corporation 82371SB PIIX3 ISA [Natomia/Triton II]
00:01.1 IDE interface: Intel Corporation 82371SB PIIX3 IDE [Natomia/Triton II]
00:01.3 Bridge: Intel Corporation 82371AB/EB/MB PIIX4 ACPI (rev 03)
00:02.0 VGA compatible controller: Cirrus Logic GD 5446
00:03.0 Ethernet controller: Red Hat, Inc Virtio network device
00:04.0 SCSI storage controller: Red Hat, Inc Virtio block device
root@hvm01:~# _
```

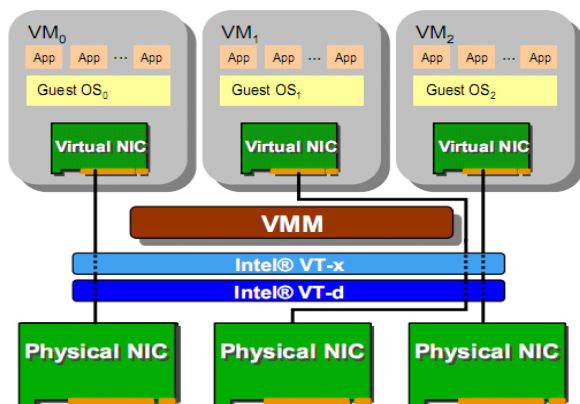
虚拟网卡

- Passthrough I/O
 - 直接将物理网卡分配给Guest
 - 开销小，性能好
 - 共享性差
 - Guest中直接安装网卡驱动
 - 需要IOMMU支持
- SR-IOV
 - Passthrough I/O的特例
 - 需要硬件支持

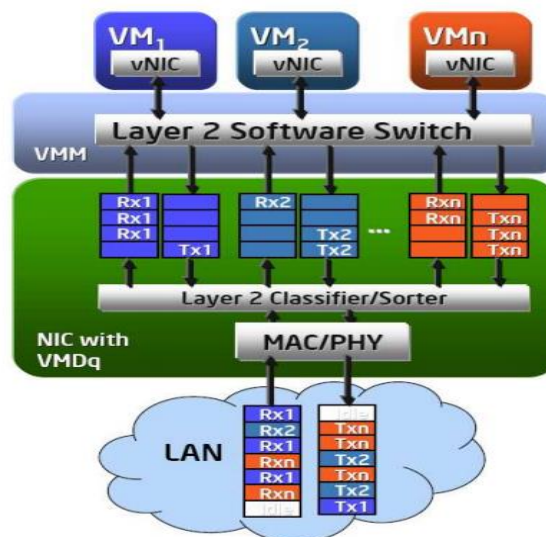
Passthrough I/O



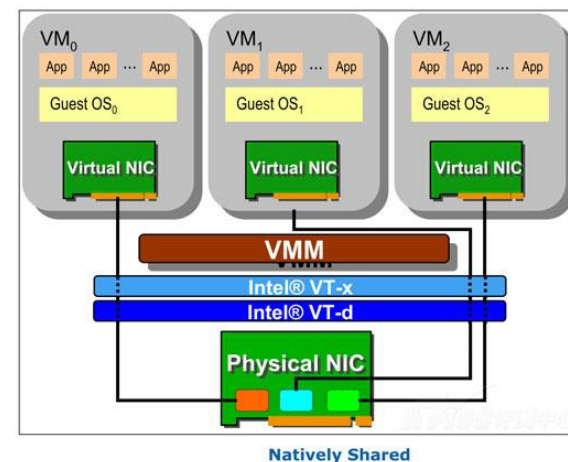
几种Intel的IO虚拟化技术



Intel 直接 I/O 虚拟化技术 (VT-d)



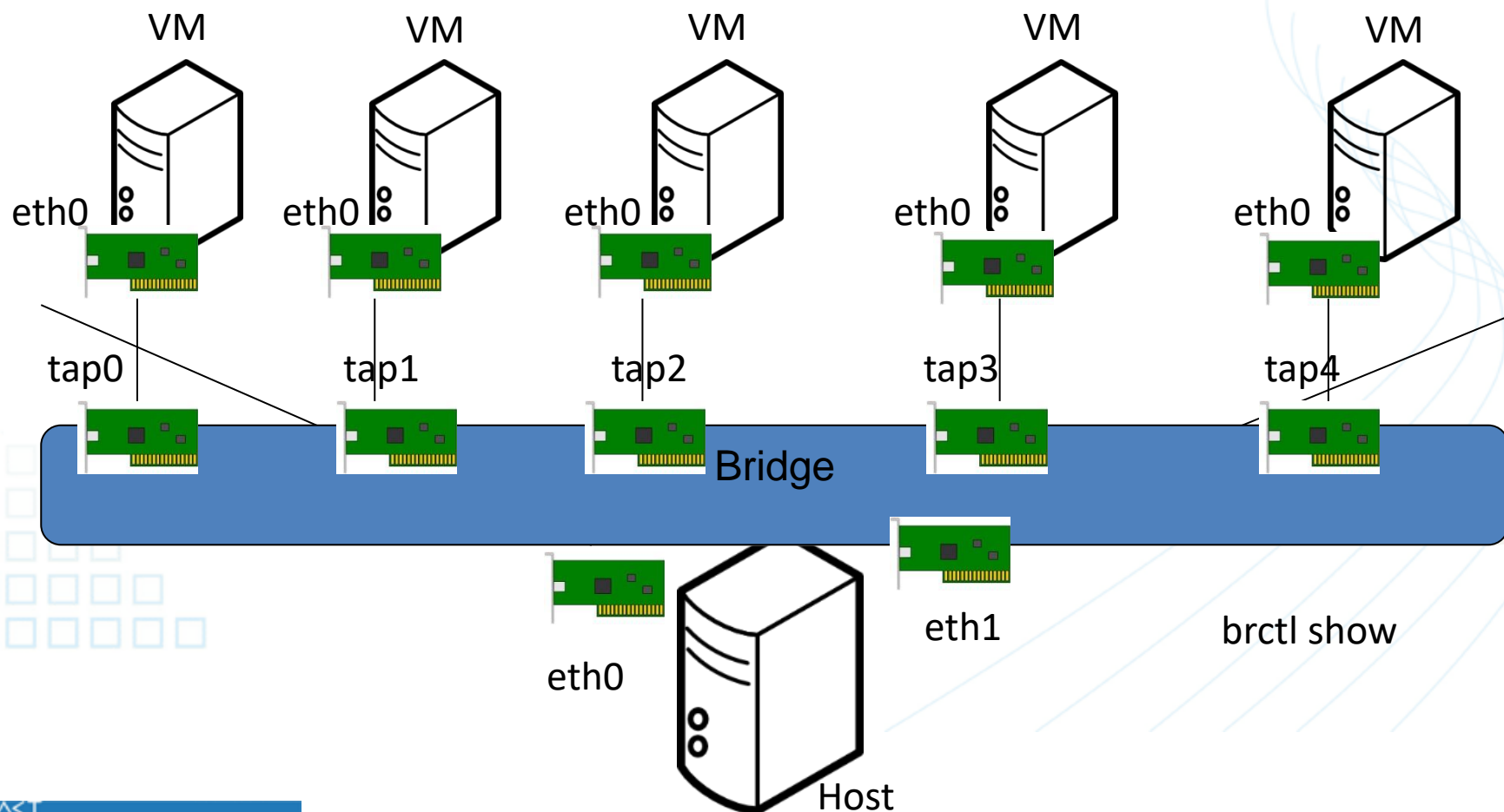
Intel 虚拟设备队列 (VMDq)



SR-IOV

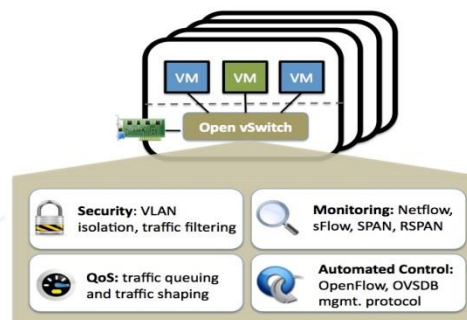
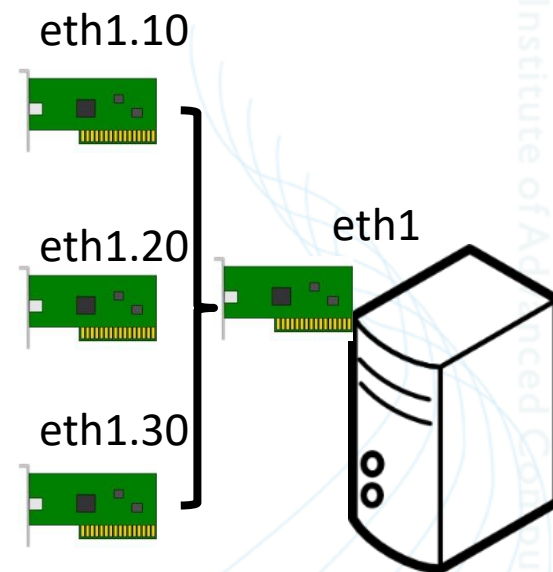
二层虚拟化

- 一个物理机内：软件网桥



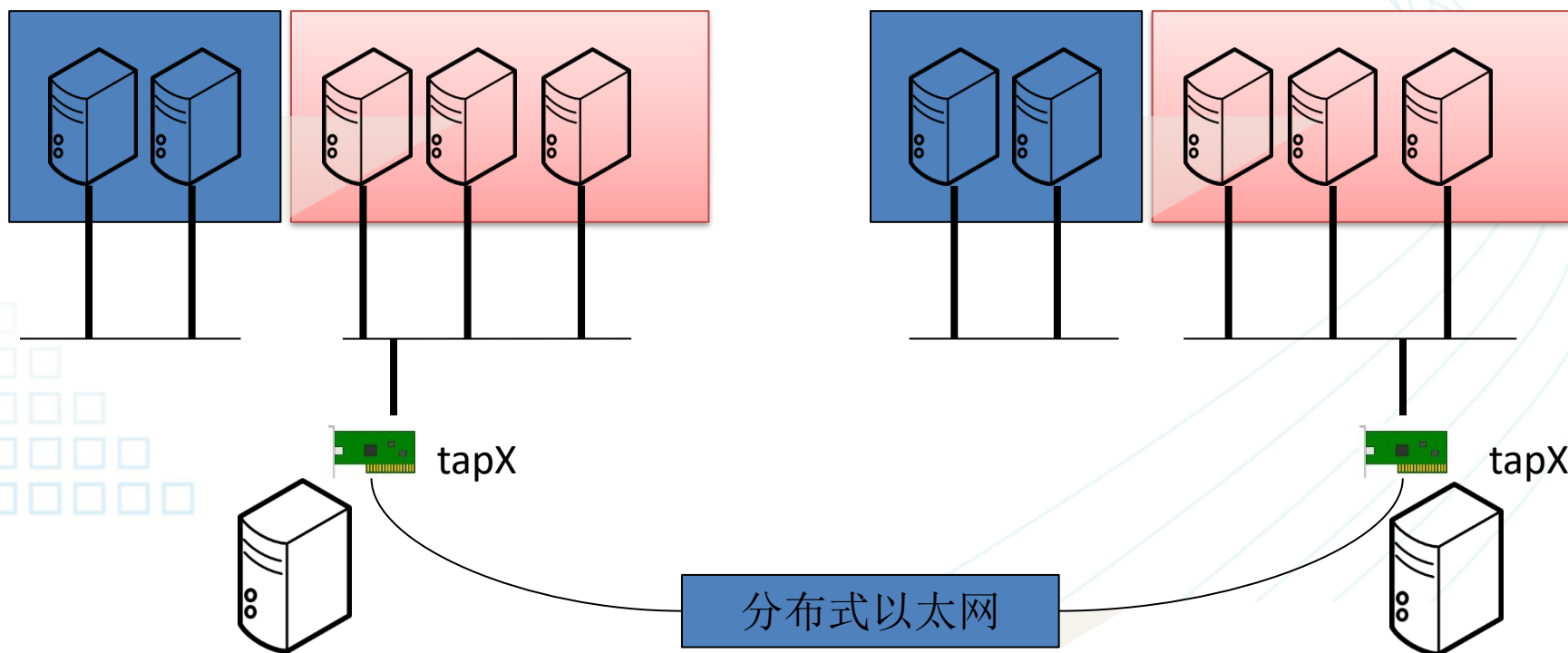
二层虚拟化——高级交换机功能

- VLAN支持
 - 基于原生Linux Kernel Bridge
 - Linux Kernel支持802.1Q
 - 生成虚拟子接口，加入不同bridge
 - `vconfig eth1 vlan 10`
 - 基于Open vSwitch
 - <http://openvswitch.org/>



跨宿主机二层组网

- 多个物理机上部署的VM之间纳入同一二层网（冲突域）
 - 共享 vs 隔离



跨宿主机二层组网

- 分布式以太网

- 基于物理交换机 (Trunk端口)

- 可管理性差
 - 不能跨越网络

- OpenVPN

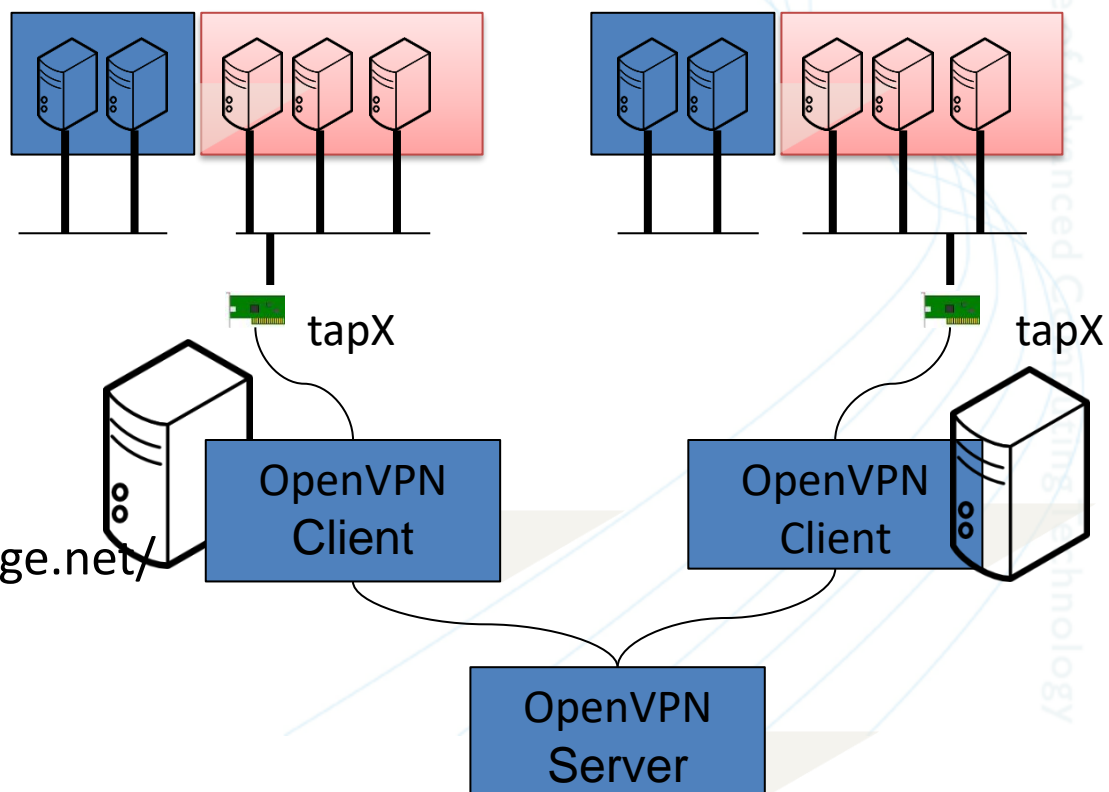
可管理

- 性能差

- Server瓶颈

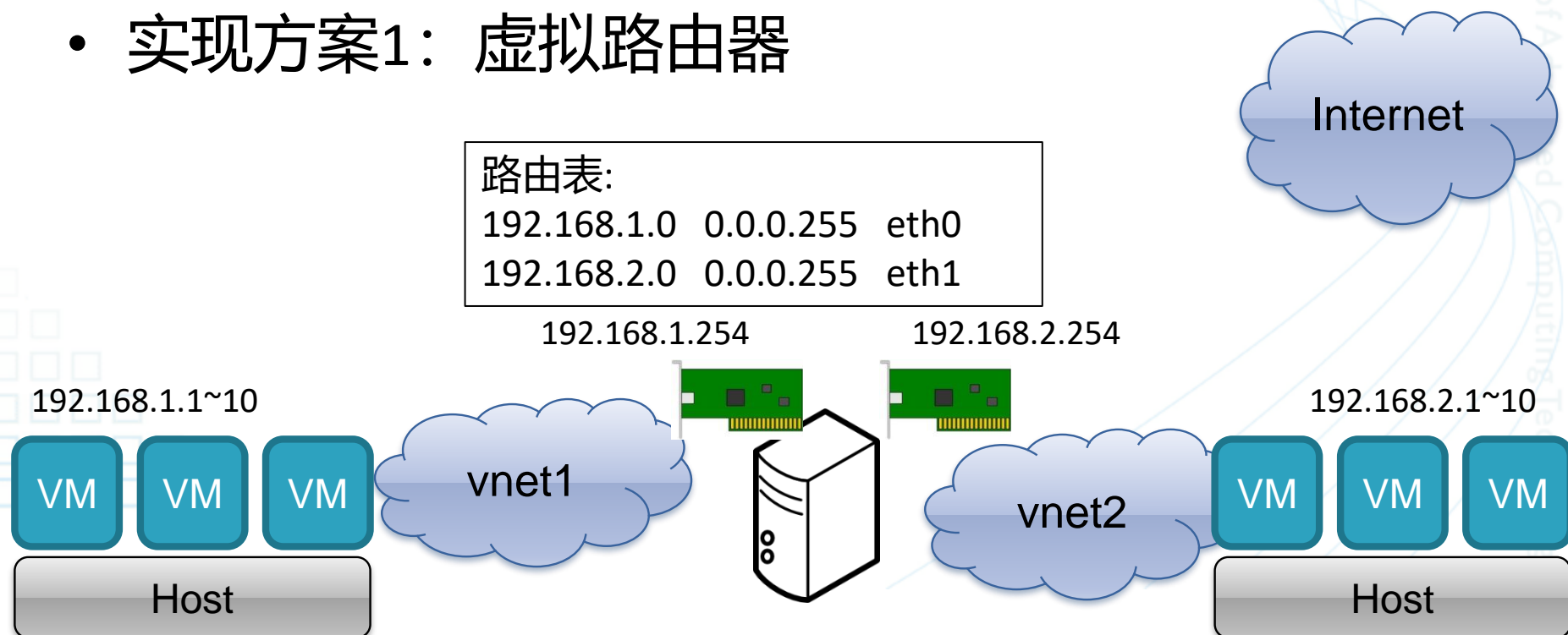
- VDE

- <http://vde.sourceforge.net/>



建立三层虚拟网络

- 任务：
 - 将2个以上虚拟网络连接
 - 或者将虚拟网络连入Internet
- 实现方案1：虚拟路由器



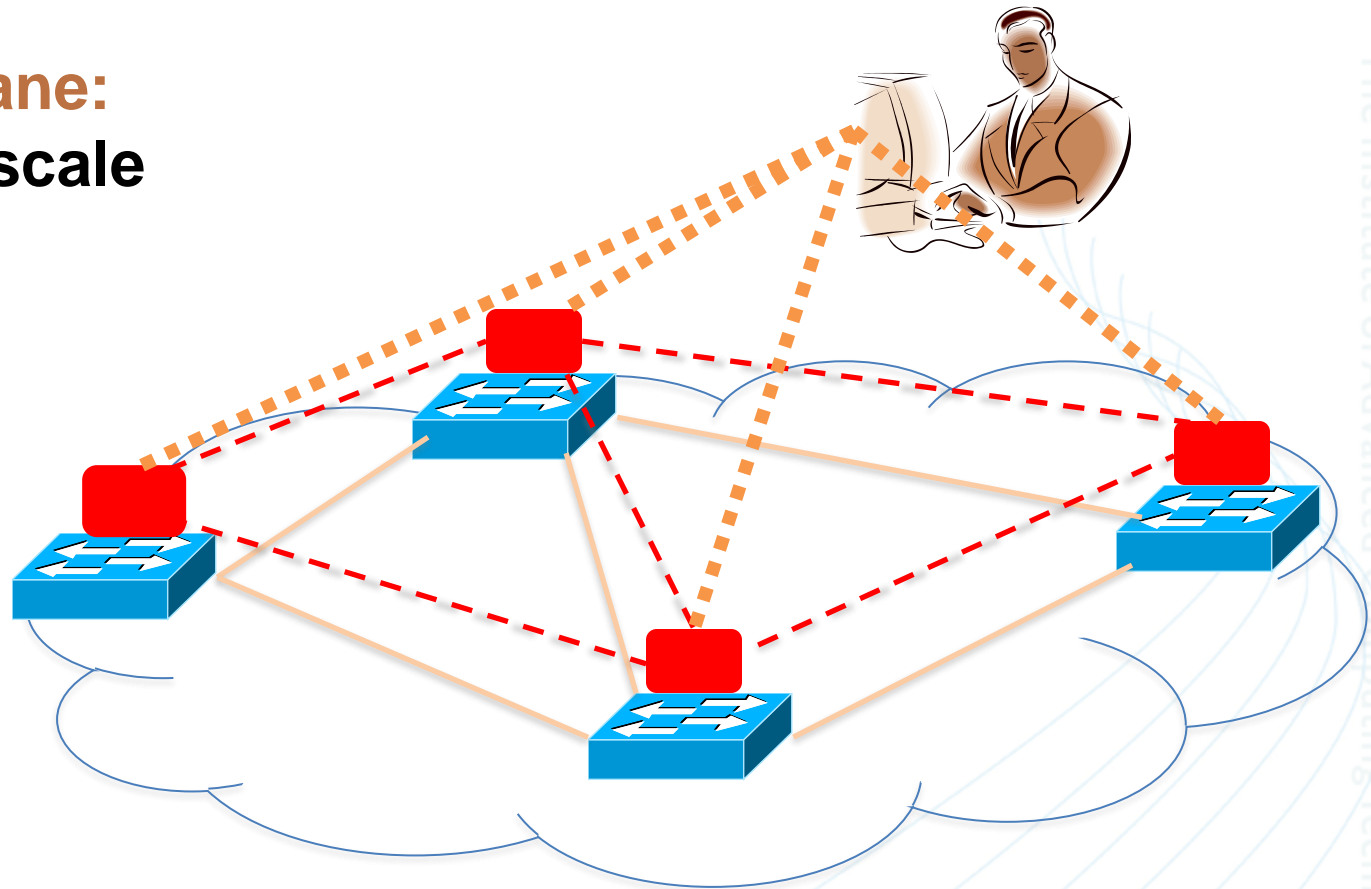
SDN与OpenFlow

- SDN——让网络可编程
- 开放的网络API? ——OpenFlow
- openflow.org
- 基于流的交换
- 标准化
 - Open Networking Foundation (ONF)



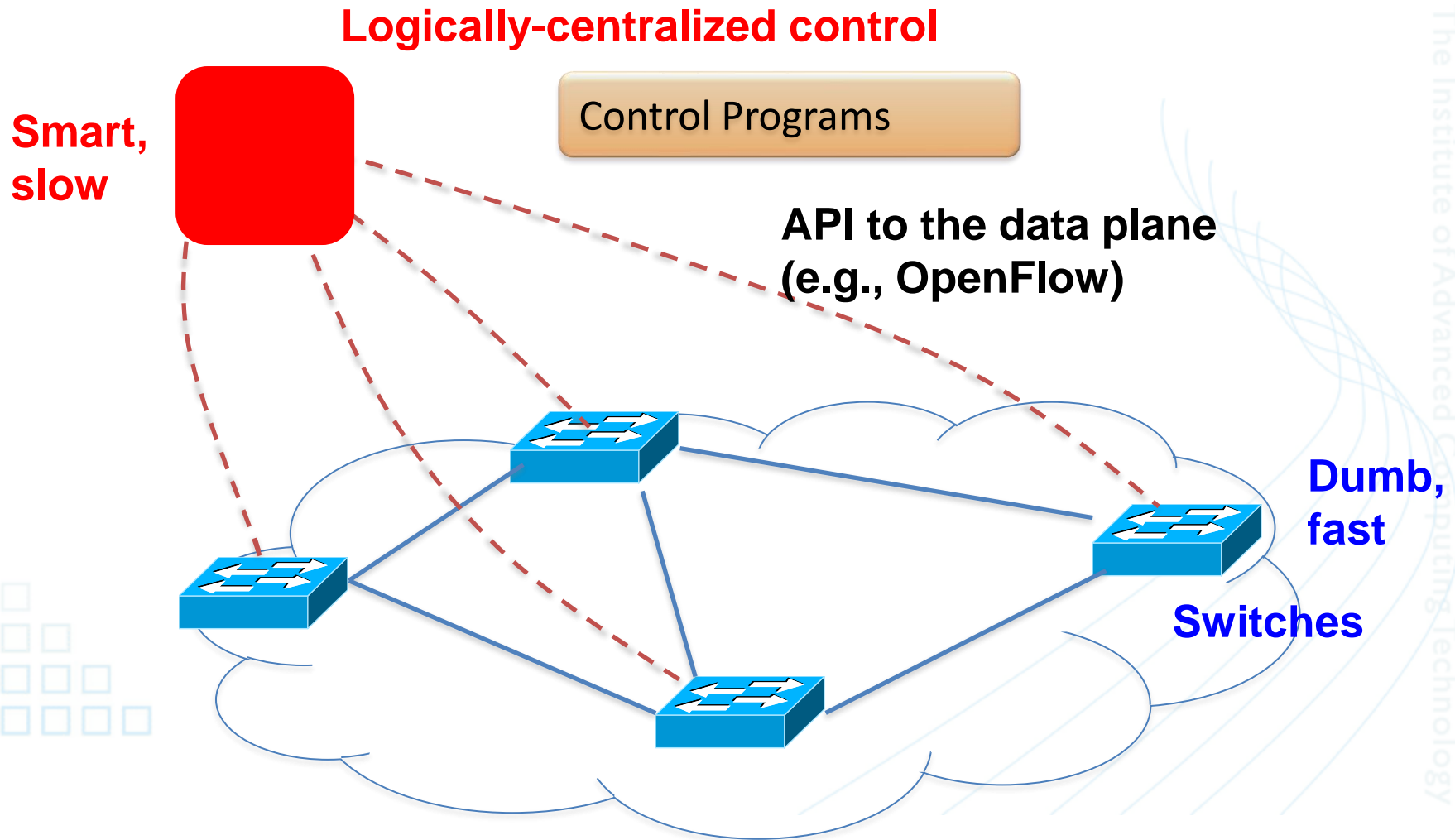
Traditional Computer Networks

Management plane:
Human time scale



Collect measurements and configure the equipment

Software Defined Networking (SDN)



容器与轻量级虚拟化

轻量级虚拟化

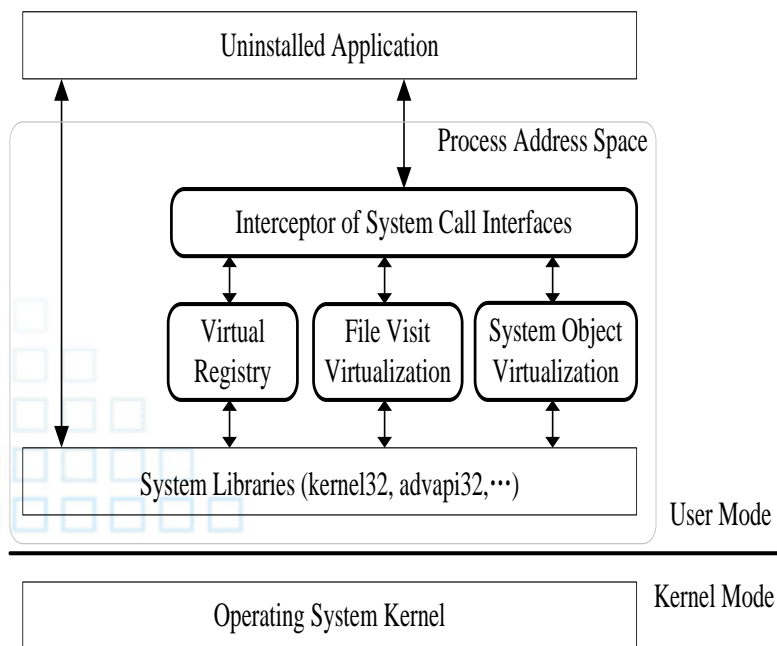
- VM: Host OS没有限制, 但Guest OS有限制
- 如果VM负载不可预测
 - Amazon EC2, 阿里云...
 - 全虚拟化, 硬件模拟或直通
- 如果VM负载可预测
 - 零距软件、PaaS、Hadoop/Spark over VM
 - 轻量级虚拟化
 - Windows: FVM
 - Linux: OpenVZ, LXC (cgroup)... Linux容器技术

软件运行隔离

- 虚拟机技术
 - 在软、硬件之间引入虚拟层
 - 在同一台计算机上可以同时运行多个操作系统
 - VMWare Workstation、Xen、KVM、Virtual PC
- 轻量级的隔离技术（容器）
 - 将程序对OS的操作重定向到独立空间，并提供安全性
 - Jails、SEE、Consh、Alcatraz、PDS
 - [Containers—Not Virtual Machines—Are the Future Cloud](#),
Linux Journal 2013
 - Linux-Vserver, OpenVZ、FreeVPS、LXC: Linux container tools,
此外, Solaris Zones和BSD jails都基于containers原理实现

软件运行隔离

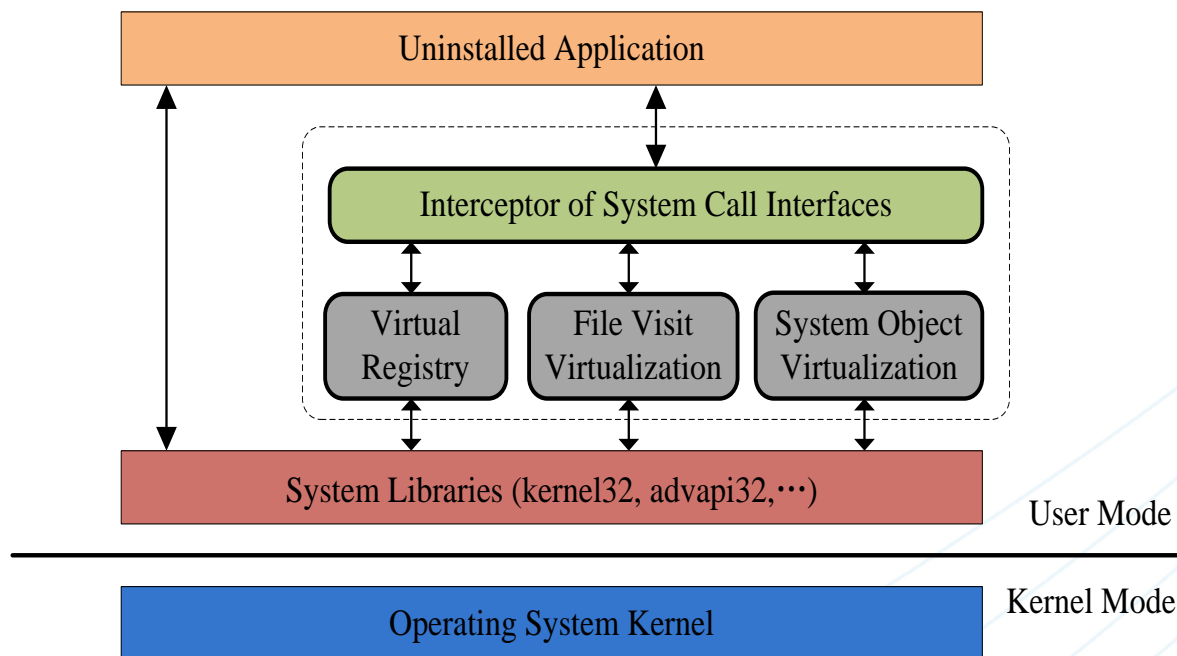
- Windows上轻量级虚拟机
 - 实现了应用软件环境的按需部署和安全隔离
 - 提高了虚拟资源的聚合使用能力



- **系统调用拦截模块：** 挂钩时机、挂钩API选择、共享数据空间
- **虚拟注册表模块：** 物理存储格式、运行时存储格式、一致性维护
- **文件访问虚拟化模块：** 重定向访问策略、枚举方式访问、文件删除
- **系统对象虚拟化模块：** 创建系统对象、打开系统对象

轻量级虚拟机隔离技术

- 提供在Windows平台软件执行轻量级隔离
 - 实现了Windows OS上应用软件环境的按需部署和安全隔离
 - 基于windows server session机制实现用户桌面隔离



Linux的Namespace

- 系统资源（进程、用户账户、文件系统、网络）都属于某个namespace
- 每个namespace下的资源对于其他的namespace资源是透明的，不可见的

```
17 | struct nsproxy {  
18 |     atomic_t count;  
19 |     struct uts_namespace *uts_ns;  
20 |     struct ipc_namespace *ipc_ns;  
21 |     struct mnt_namespace *mnt_ns;  
22 |     struct pid_namespace *pid_ns_for_children;  
23 |     struct net          *net_ns;  
24 | };
```

Linux的CGroup

- Control Groups: 提供限制、记录、隔离进程组所使用的物理资源
- (子系统: CPU /内存/IO)
- 主要功能:
 - 限制进程组可以使用的资源数量
 - 控制进程组优先级 (如cpu share)
 - 记录进程组使用资源的数量
 - 进程组隔离 (与namespace结合)
 - 进程组控制

Linux的CGroup

下面就跑一个死循环程序，导致CPU使用率到达100%。

```
1 | PID USER      PR  NI   VIRT   RES    SHR S  %CPU  %MEM    TIME+  COMMAND
2 | 5046 yan        20   0   25928   4848   2324 R  100.0   0.1    0:22.47 python
```

现在执行如下的命令：

```
1 | echo "50000" >/sys/fs/cgroup/cpu/geekcome/cpu.cfs_quota_us
2 | echo "5046" >/sys/fs/group/cpu/geekcome/tasks
```

再top查看一下：

```
1 | PID USER      PR  NI   VIRT   RES    SHR S  %CPU  %MEM    TIME+  COMMAND
2 | 5046 yan        20   0   25928   4844   2324 R   49.8   0.1    0:49.27 python
```

LXC: Linux Container

- 对进程分组 (cgroup subsystems)
 - 软隔离、配额管理 (quota mngt)
- 进程组对本地设备进行直接访问
 - I/O瓶颈

LXC使用:

创建一个容器:

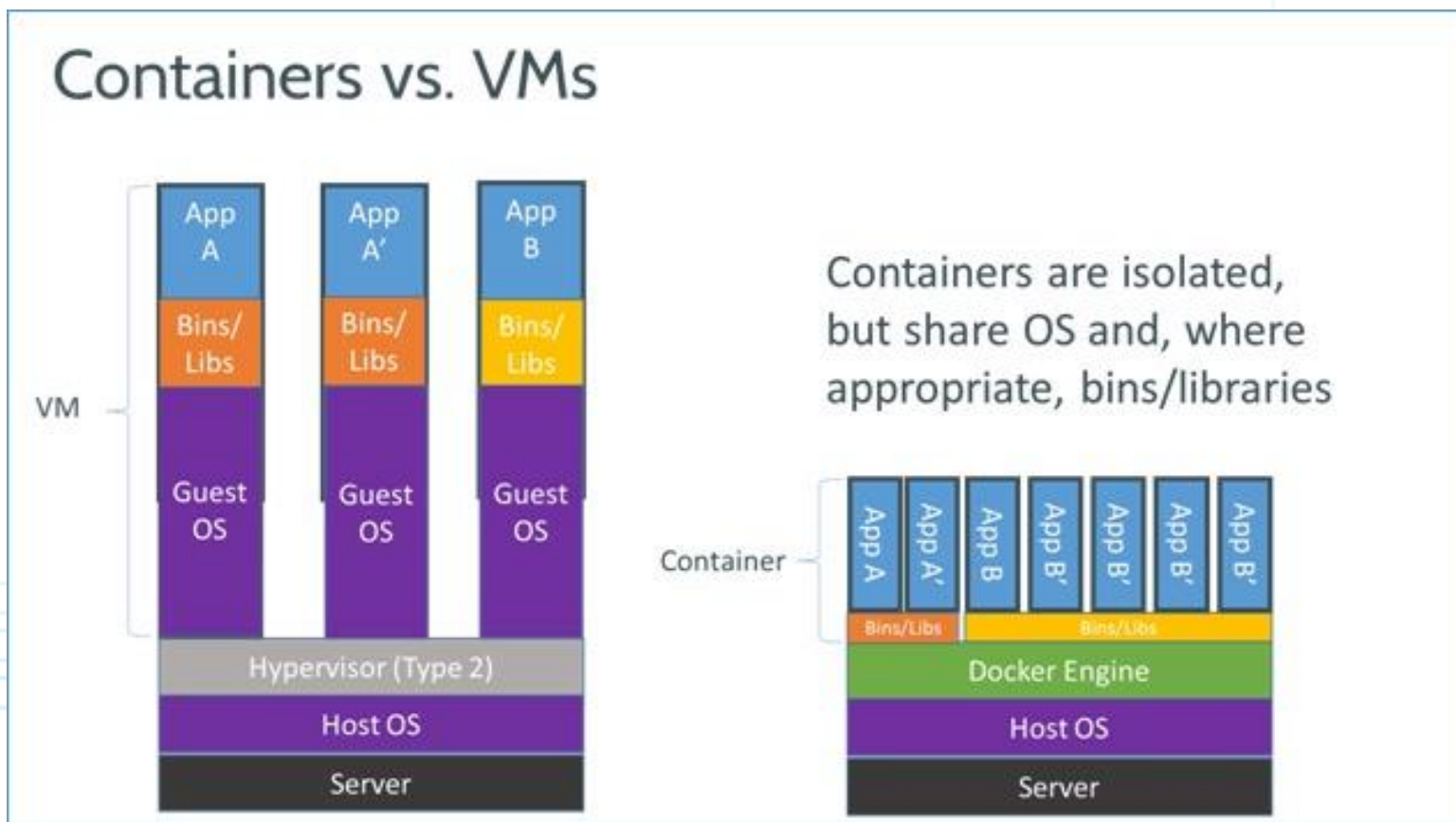
```
1 | lxc-create -n name [-f config_file] [-t template]
```

```
2 | sudo lxc-create -n ubuntu01 -t ubuntu
```

-n就是虚拟机的名字，-t是创建的模板，保存路径在/usr/lib/lxc/templates。模板就是一个脚本文件，执行一系列安装命令和配置（穿件容器的挂载文件系统，配置网络，安装必要软件，创建用户并设置密码等）。

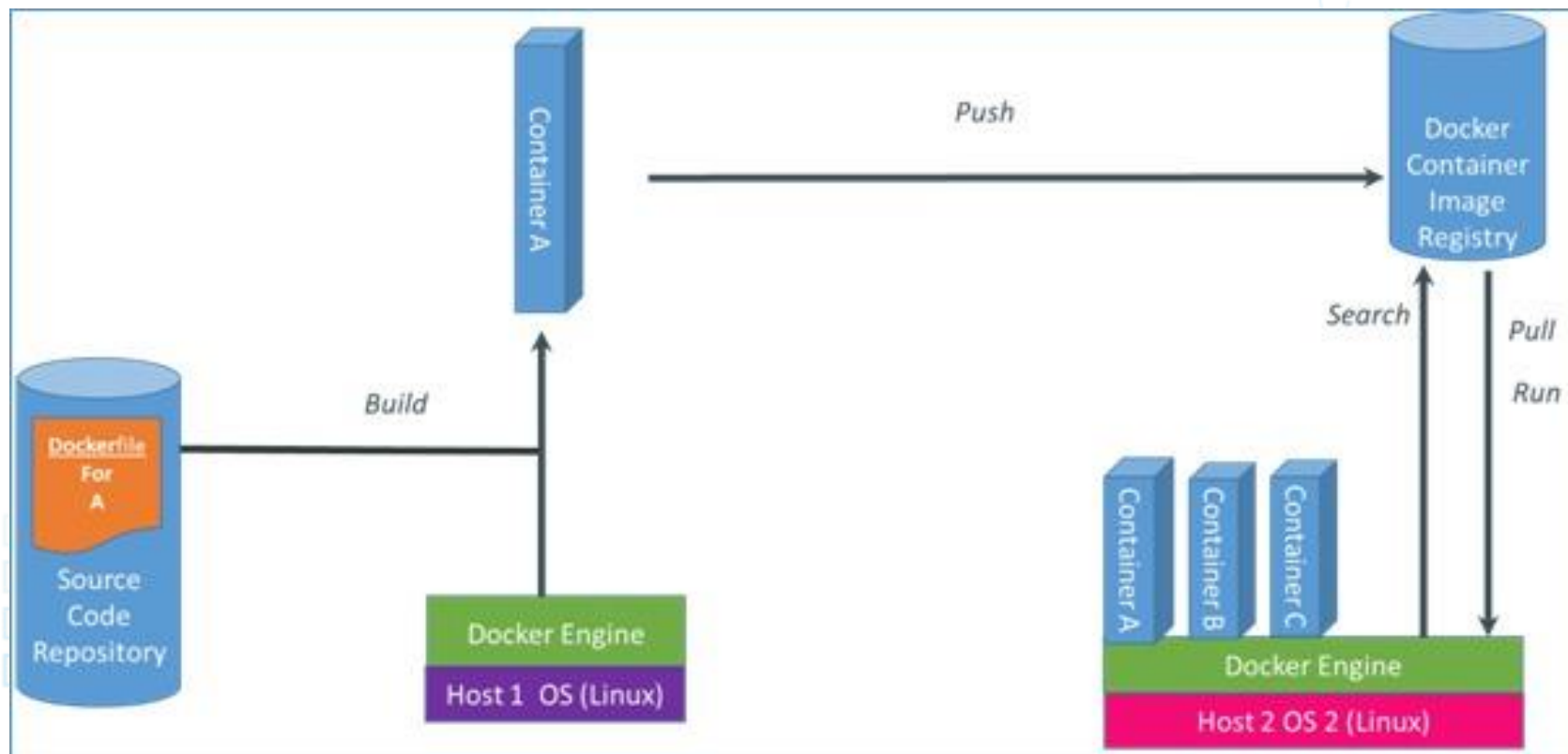
基于LXC的虚拟资源调度与管理

- Docker (2013-)



基于LXC的虚拟资源调度与管理

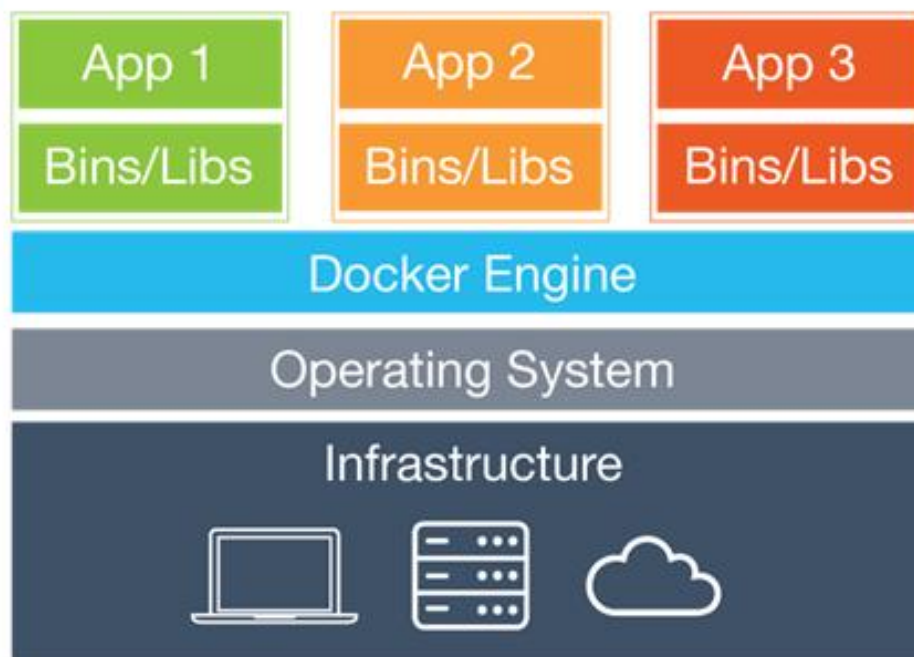
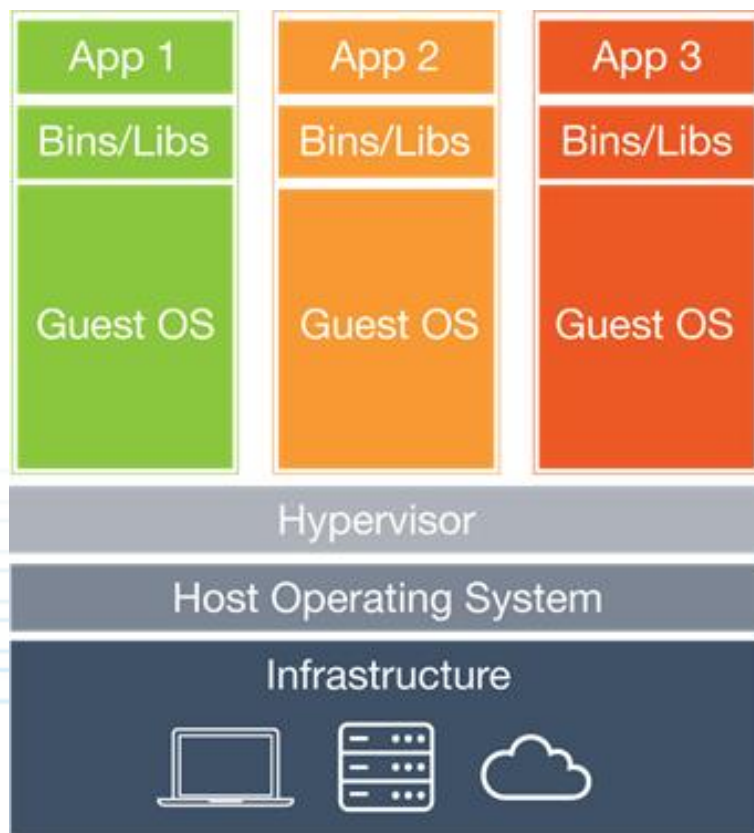
- Docker (2013-)



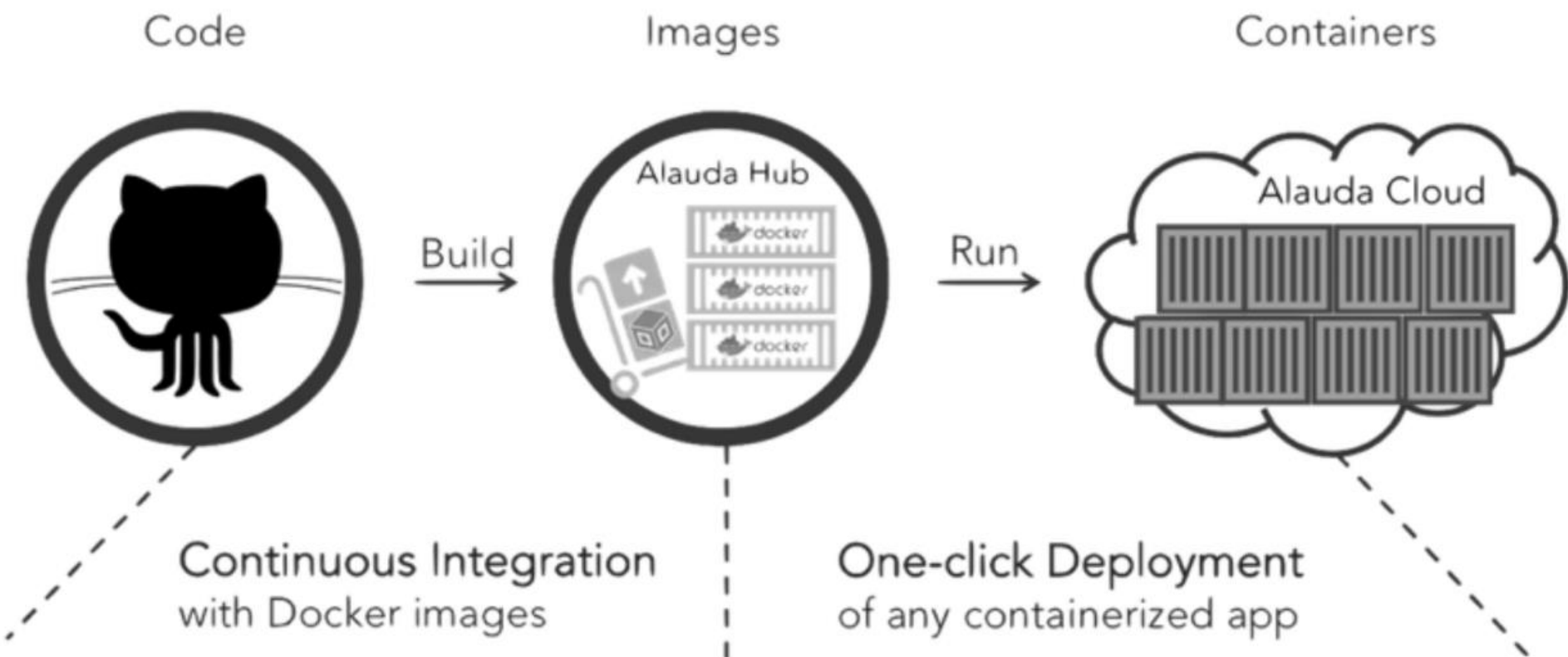
容器 vs 虚拟机

容器是操作系统级的虚拟化技术，共享内核特性使之相较于VM更加的轻量

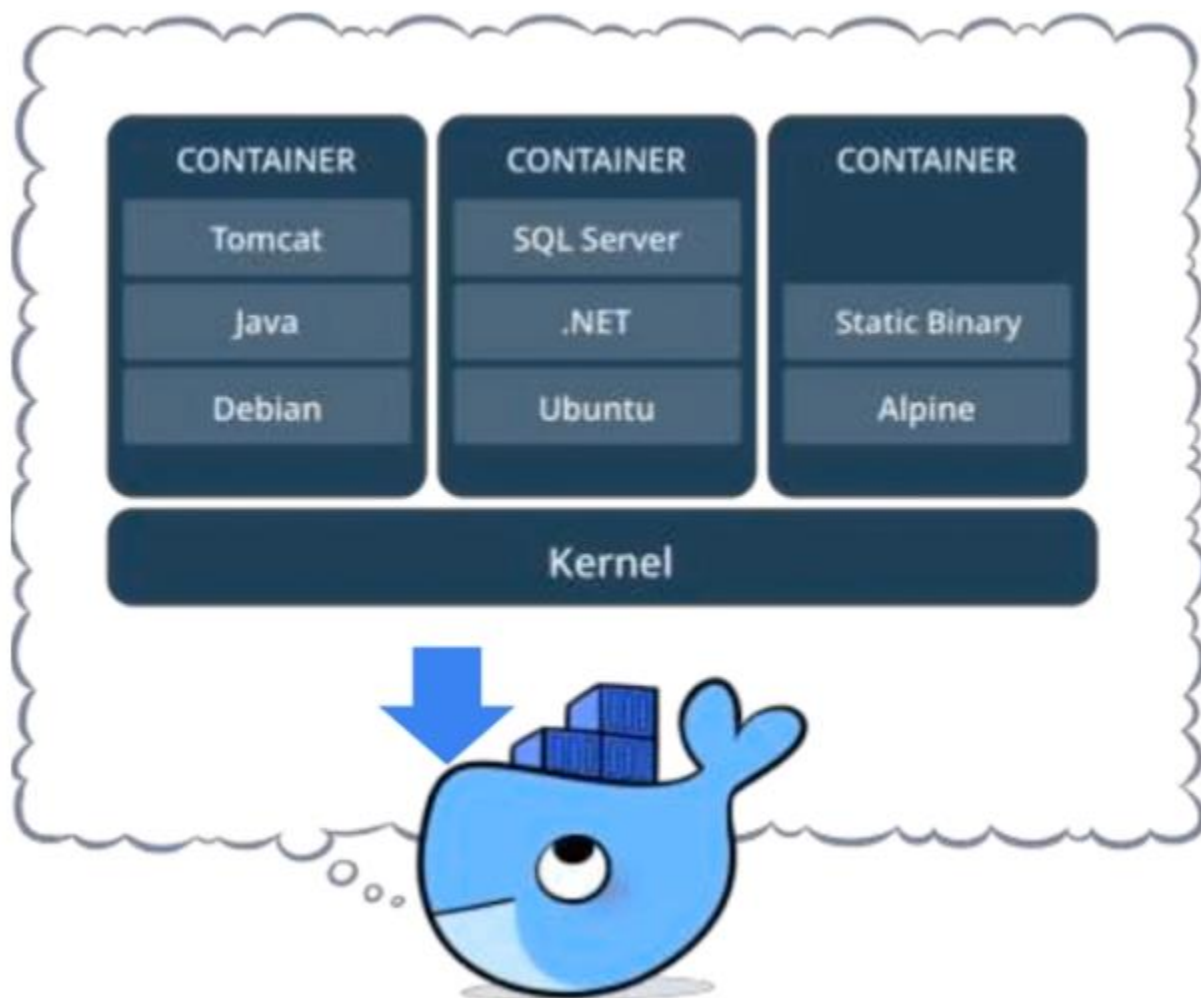
- 更快的启动速度
- 更低的内存占用
- 更好的弹性



容器镜像封装了应用及其依赖的运行环境，已经成为软件交付的标准



Docker的用处



Docker的用处



容器是APP层面的隔离



虚拟化是物理资源层面的隔离

基于LXC的虚拟资源调度与管理

- Docker (2013-)
 - 构建PaaS
 - 托管Web应用
 - 应用的封装与迁移
 - (同OS的) 安全沙盒
 - 应用部署自动化
 - 轻量级桌面虚拟化

Docker的基本概念

- 镜像：Image
 - Linux启动后挂载root文件系统
 - Docker镜像相当于root文件系统
 - 镜像内容不会改变，不包含动态数据
 - 联合文件系统：Union FS
- 容器：Container（运行时）
 - 实质是运行在独立命名空间的进程
 - 容器状态会随容器删除而丢失（无状态）

Docker的基本概念

- 仓库：Repository
 - 镜像的集中存放服务器，分发镜像
 - 私有 vs 公有
- Docker File：定制镜像
 - 从一个容器镜像出发
 - 在上面进行一系列的安装、配置操作

思考1

- 能否在Linux服务器上做一个Windows的Docker镜像?

思考2

- 如何在Windows服务器上运行Linux Docker镜像?

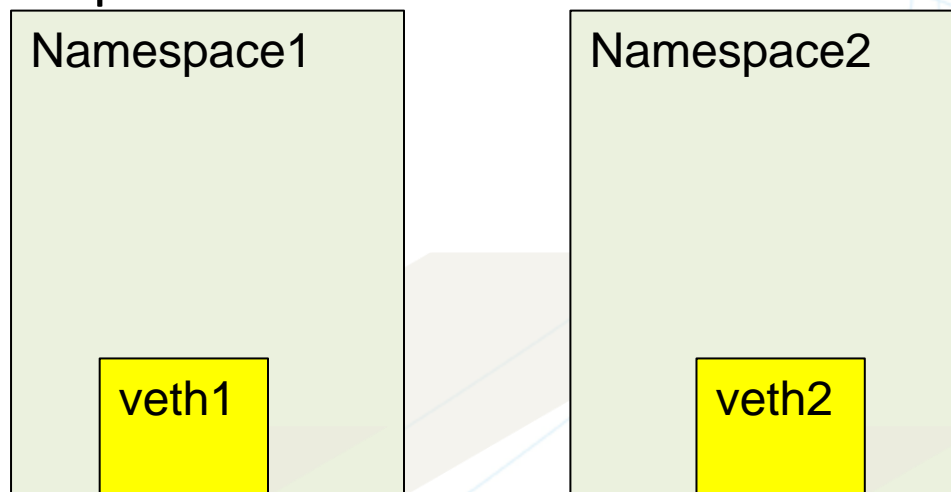


思考3

- Docker如何提供网络支持?

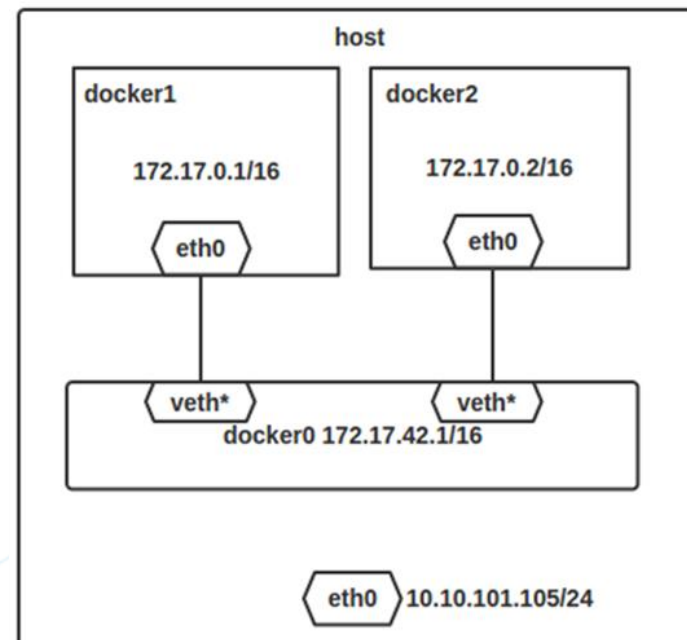
Docker的网络模型

- Linux中的网络namespace
 - `ip netns add <name>`
- 两个namespace互相不可见
 - `ip link add veth0 type veth peer name veth1`
 - `ip set veth1 netns namespace2`

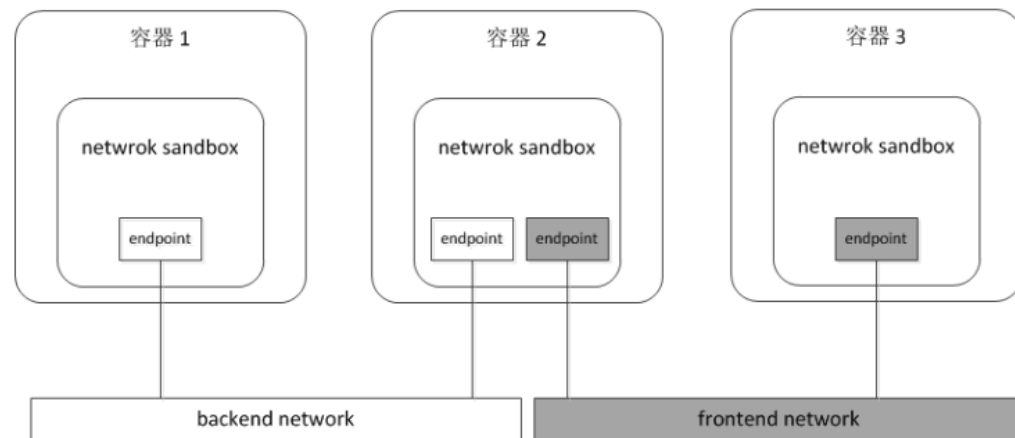
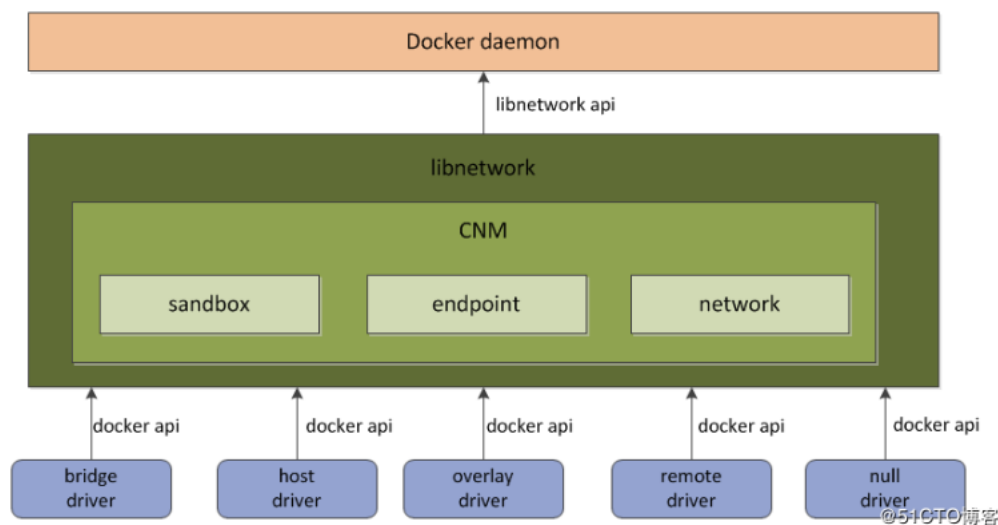


Docker的网络模型

- Docker的网络模型
 - Host模式, `--net=host`
 - Container模式, `--net=container:<id>`
 - None模式, `--net=none`
 - Bridge模式, `--net=bridge`



Docker的网络模型：libnetwork



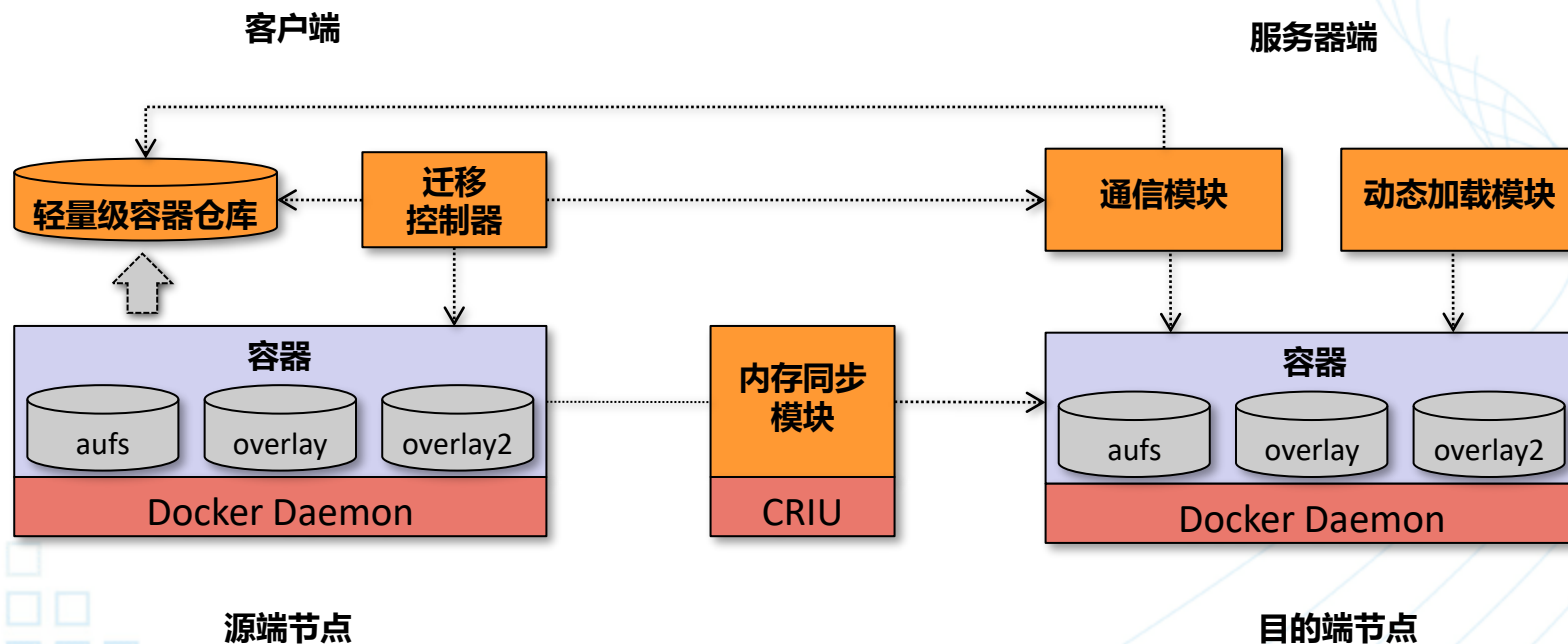
图：CNM组件示例图

思考4

- Docker是否可以迁移?
- 是否可以热迁移?

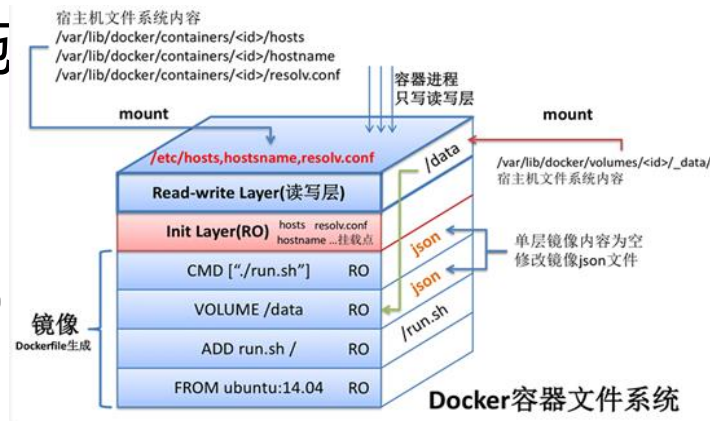
思考4

- 容器的热迁移

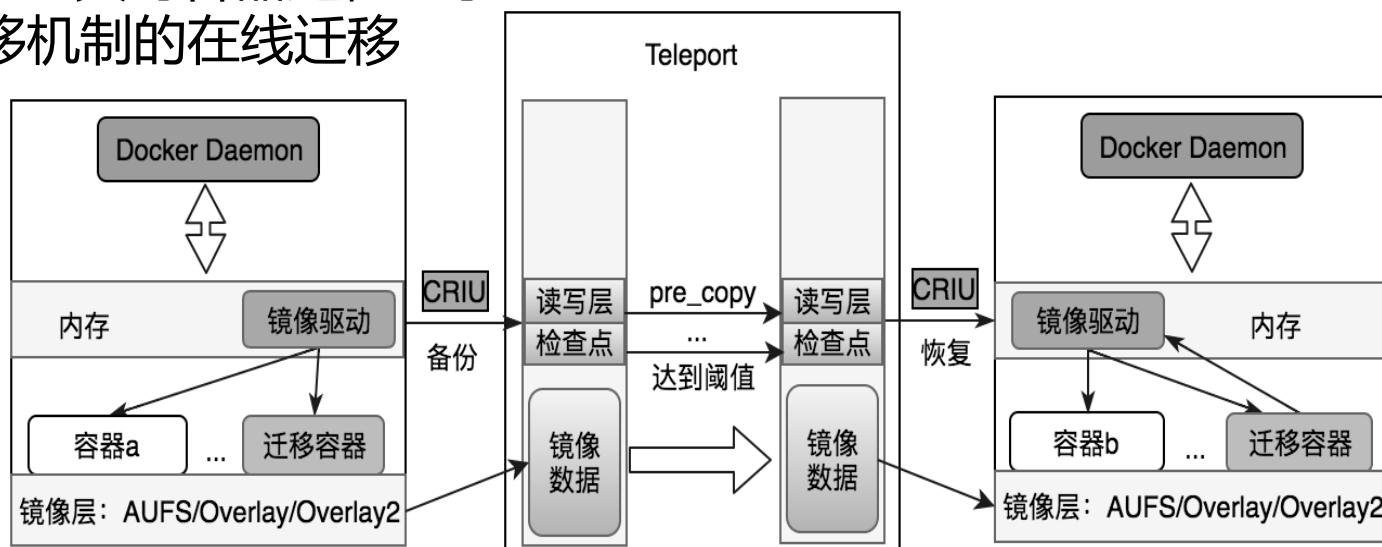


例：容器在线迁移技术

- 问题：容器迁移涉及多个层面，实施困难
- 解决方案
 - 基于文件系统驱动(aufs/overlay/overlay2)实现镜像数据整合
 - 利用CRIU工具对容器进程基于Pre-Copy迁移机制的在线迁移



- 迁移后，动态加载镜像信息
- 实施效果
- 已完成两种场景的迁移测试

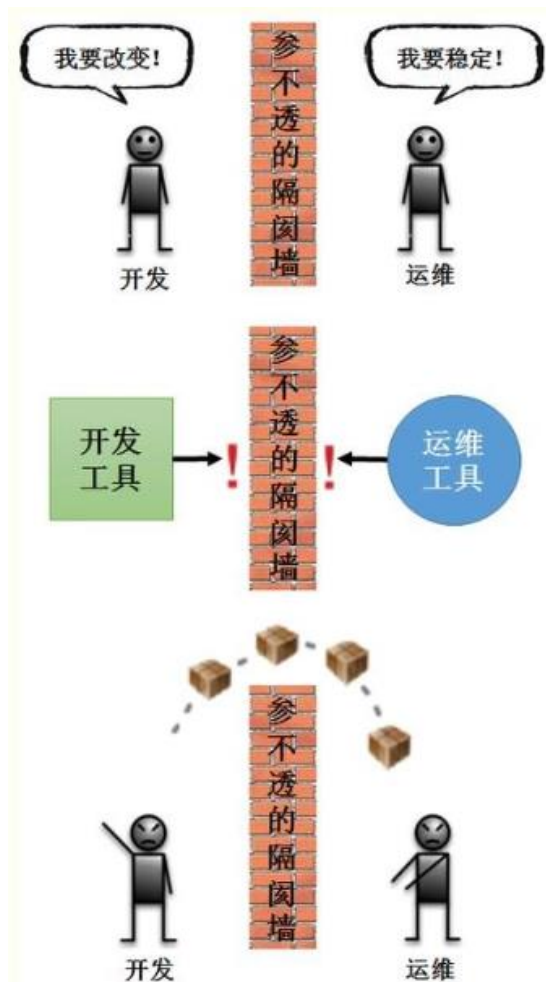


镜像信息整合 → Pre-copy在线迁移 → 驱动动态加载

容器编排与DevOps

DevOps：开发运维一体化

- 开发和运维之间的难题
- DevOps是为了填补开发端和运维端之间的信息鸿沟
 - 开发
 - 测试 (QA)
 - 上线运维



<https://www.cnblogs.com/Ming8006/p/9145135.html>

DevOps：开发运维一体化

- 需求：快速迭代，上线优化，持续发布

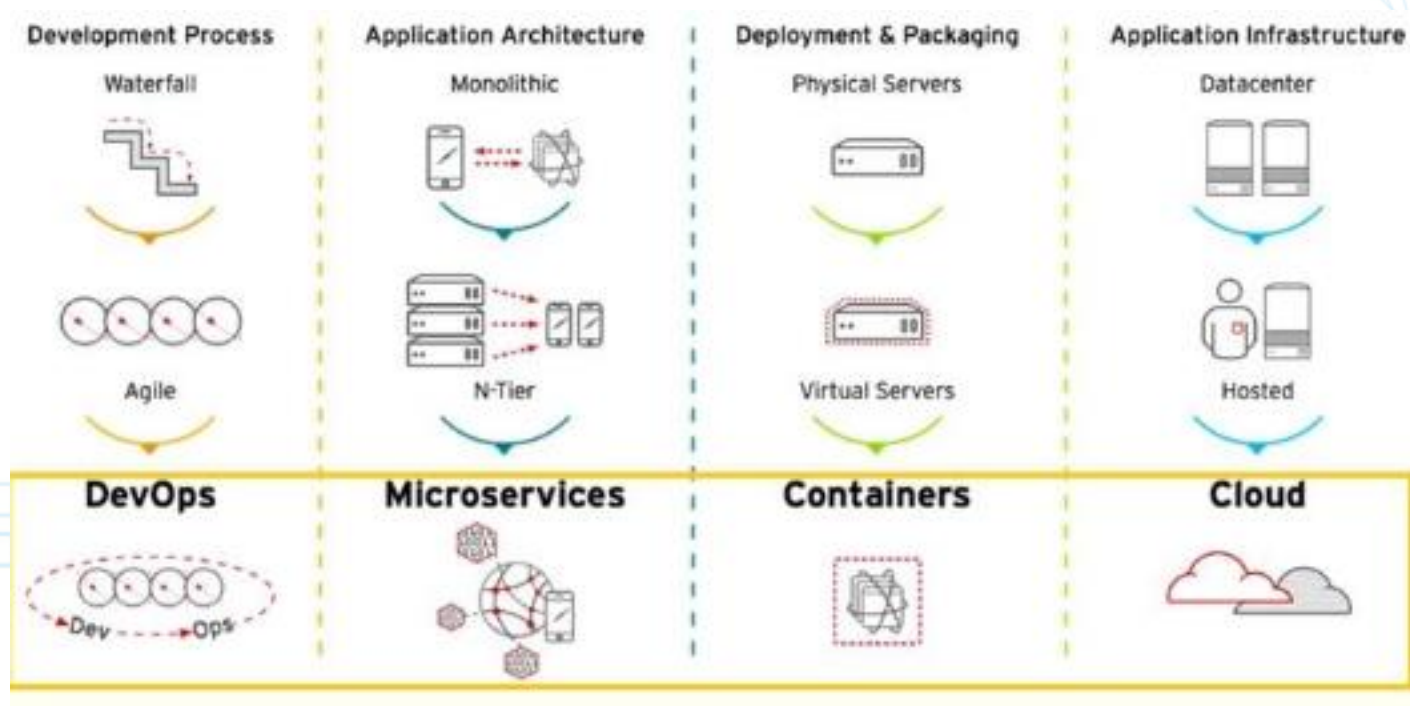
DevOps能力环



无尽头的可能性：DevOps涵盖了代码、部署目标的发布和反馈等环节，闭合成一个无限大符号形状的DevOps能力闭环。

DevOps: 开发运维一体化

- 需求：快速迭代，上线优化，持续发布
- DevOps：软件产品交付过程中 IT 工具链的打通，使各个团队减少时间损耗，更加高效地协同工作

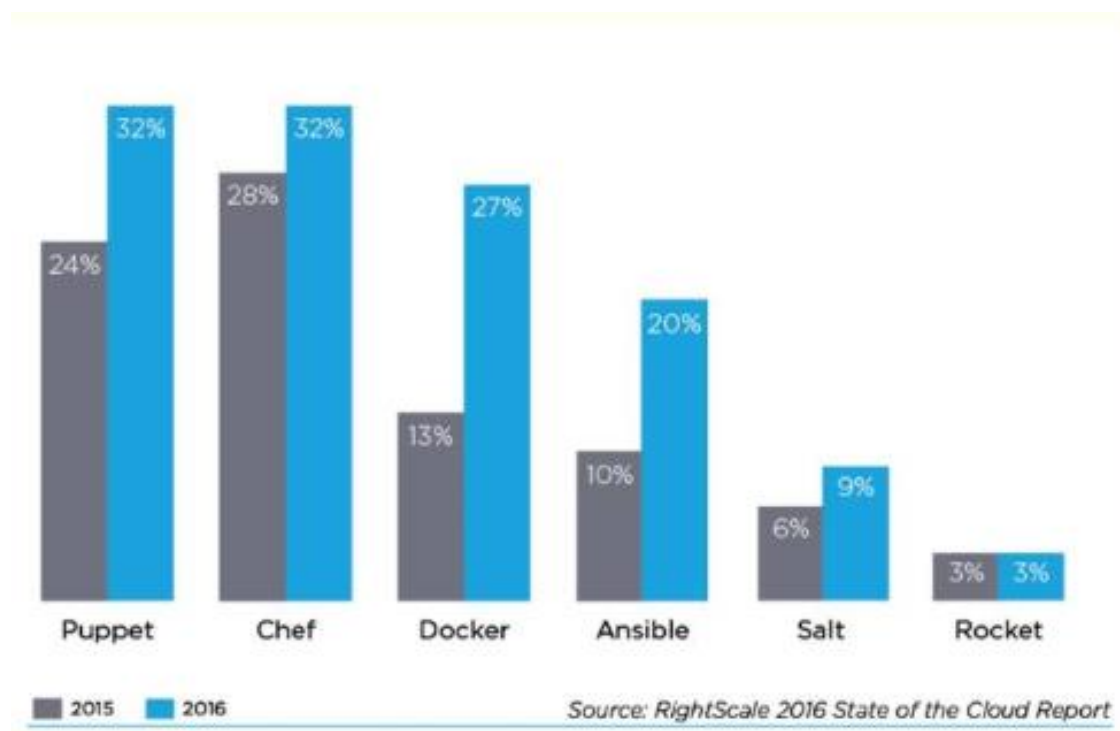


DevOps：开发运维一体化

- DevOps的实施：工具链 + 管理规范
- 代码管理（版本/协作）：Git, Github, Gitlab, Subversion
- 构建工具：Ant, Gradle, maven
- 持续集成：
- 配置管理：
- 容器支持：Docker, LXC, 容器云服务
- 编排：K8S（kubernetes），Mesos, Docker Swarn
- 其他工具：监控、日志、性能分析、压测...

DevOps：开发运维一体化

- 容器支持： Docker, LXC, 容器云服务
- 编排： K8S (kubernetes) , Mesos, Docker Swarn



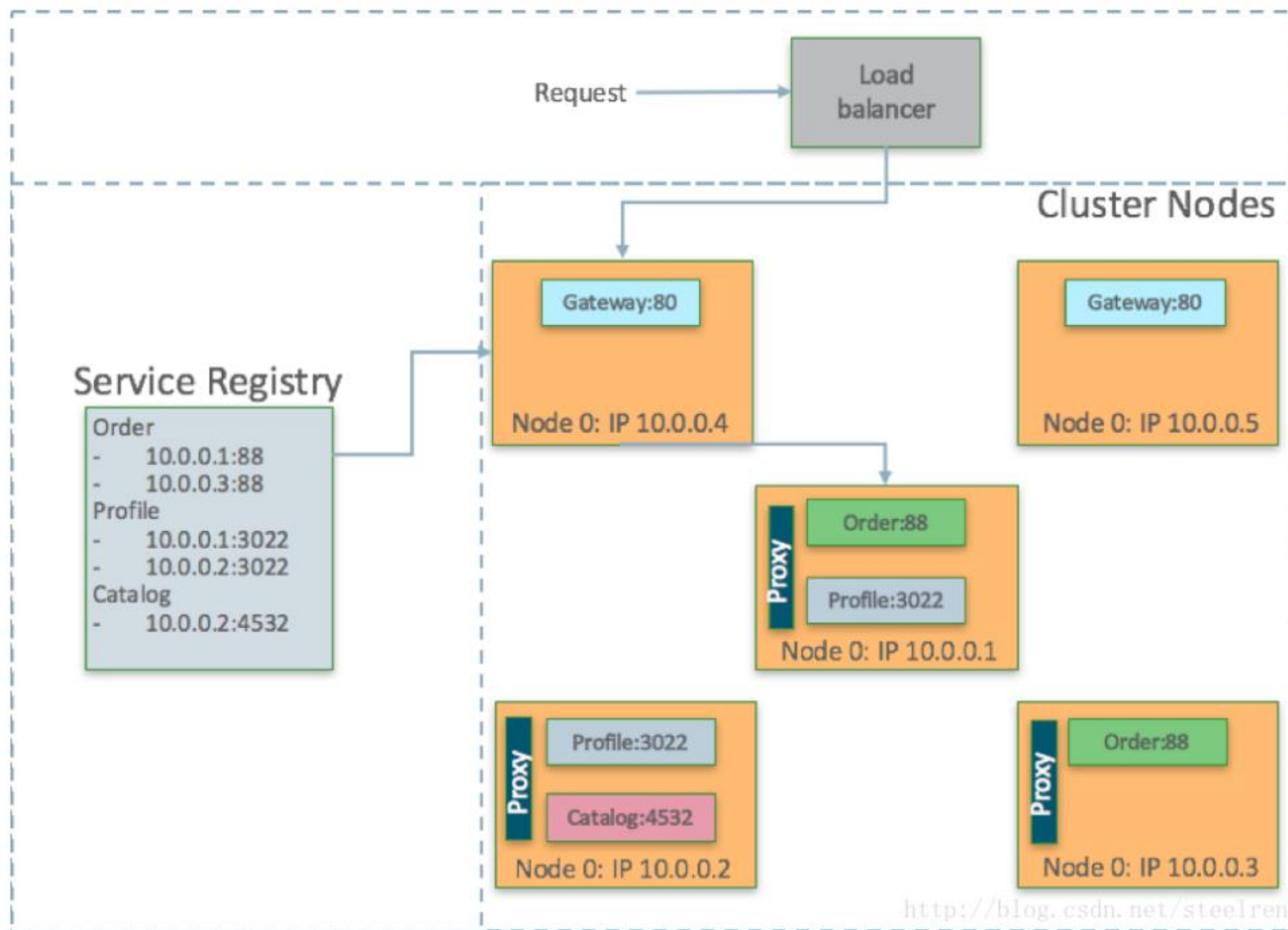
Docker的编排

- Docker-compose
 - 定义一组容器

```
# cat docker-compose.yml
version: '2'
services:
  db:
    image: mysql:5.7
    volumes:
      - "../data/db:/var/lib/mysql"
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: wordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress

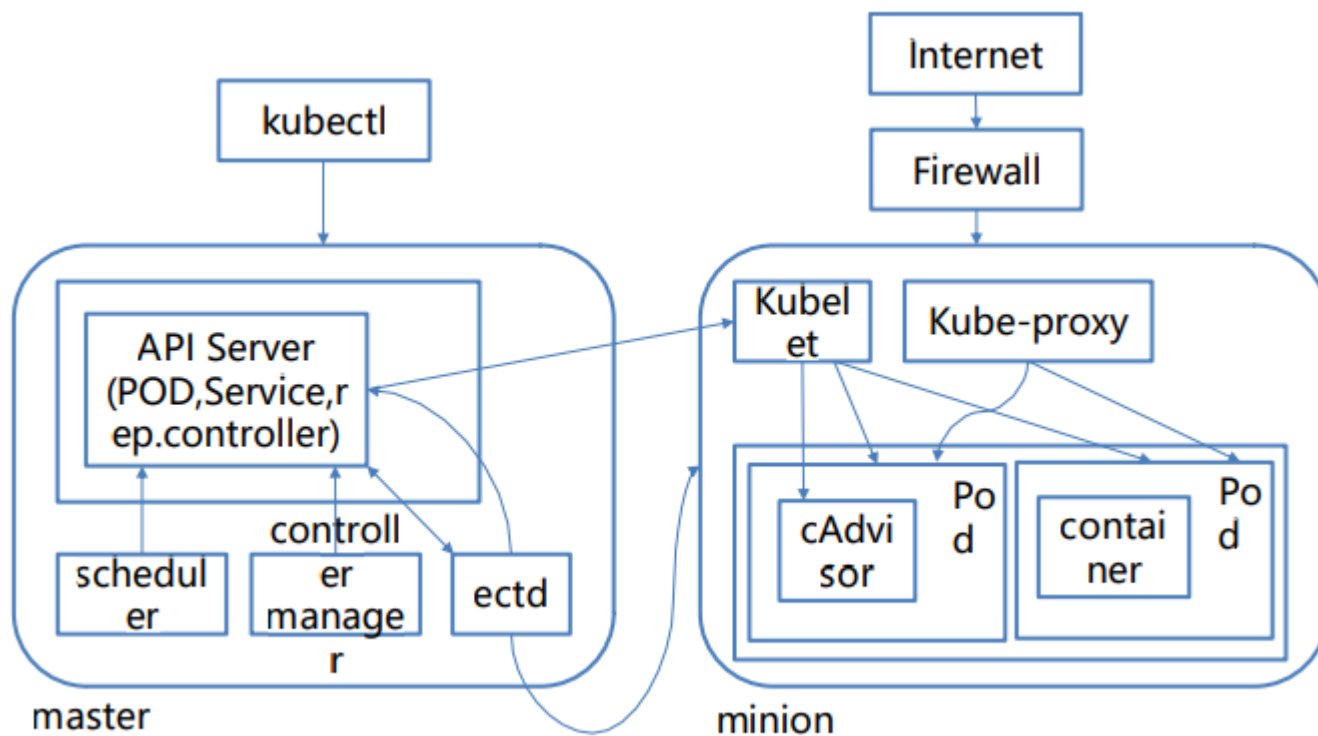
  wordpress:
    depends_on:
      - db
    image: wordpress:latest
    links:
      - db
    ports:
      - "8000:80"
    restart: always
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_PASSWORD: wordpress
```

微服务 microservice



容器编排平台

- Docker Swarm
- Google Kubernetes (K8S)



更加轻量化？ FaaS

- lamda支持弹性扩展的函数服务



AWS Lambda

aws.amazon.com/lambda

无服务器 / Serverless Computing

- **服务提供商**: 关注server概念, 按需扩容和缩容, 保证服务调用的QoS
- **用户/应用**: 只关注自己的功能 (函数) - FaaS
 - Short running stateless function
- **更细粒度的计费模式**: 不调用不计费

Serverless computing is a platform that hides server usage from developers and runs code on-demand automatically scaled and billed only for the time the code is running.

- Paul Castro, etc. The rise of serverless computing, CACM 2019.12
<https://cacm.acm.org/magazines/2019/12/241054-the-rise-of-serverless-computing/fulltext>

无服务器 / Serverless Computing

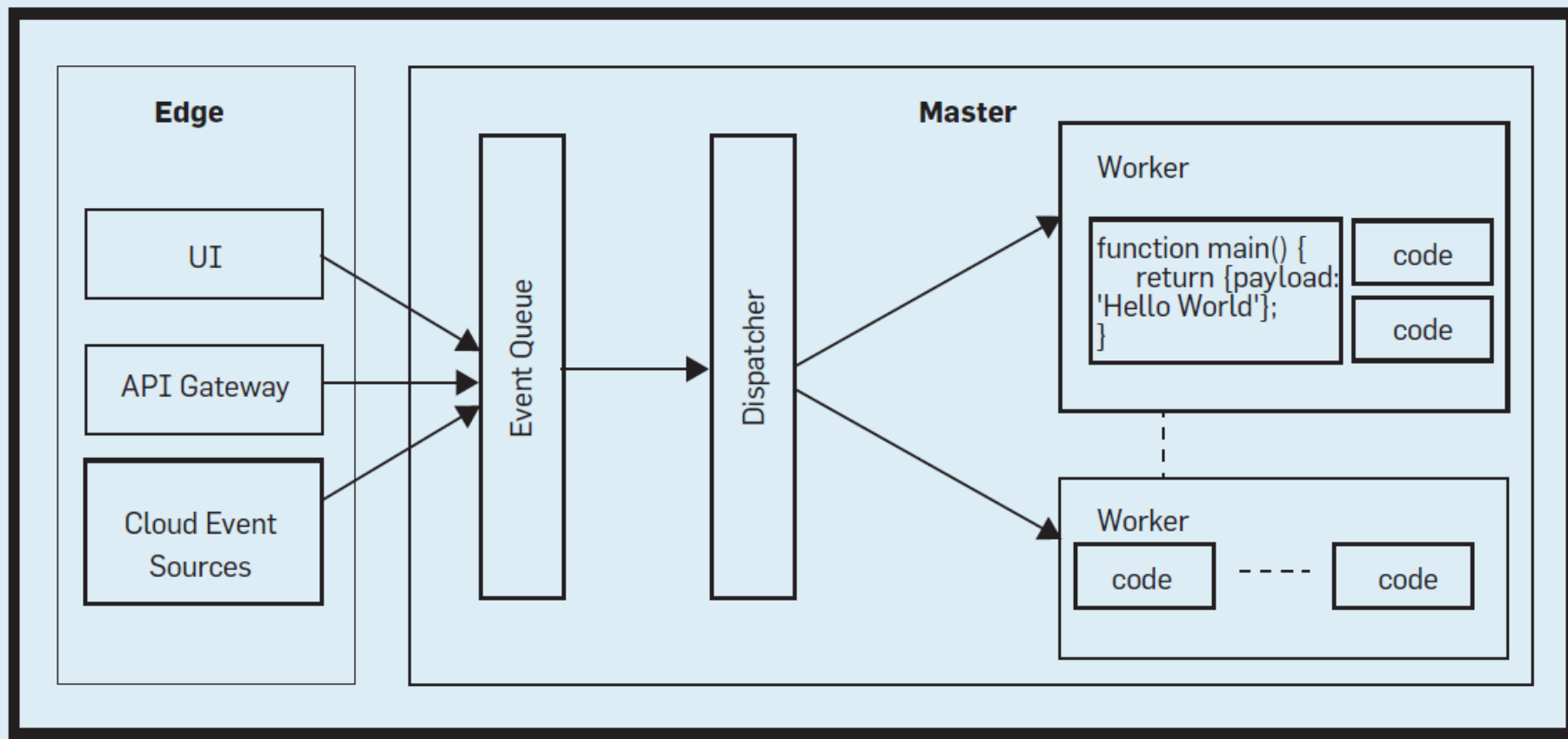
Table 1. Comparison of different choices for cloud as a service.

	IaaS	1st Gen PaaS	FaaS	BaaS/SaaS
Expertise required	High	Medium	Low	Low
Developer Control/Customization allowed	High	Medium	Low	Very low
Scaling/Cost	Requires high-level of expertise to build auto-scaling rules and tune them	Requires high-level of expertise to build auto-scaling rules and tune them	Auto-scaling to work load requested (function calls), and only paying for when running (scale to zero)	Hidden from users, limits set based on pricing and QoS
Unit of work deployed	Low-level infrastructure building blocks (VMs, network, storage)	Packaged code that is deployed and running as a service	One function execution	App-specific extensions
Granularity of billing	Medium to large granularity: minutes to hours per resource to years for discount pricing	Medium to large granularity: minutes to hours per resource to years for discount pricing	Very low granularity: hundreds of milliseconds of function execution time	Large: typically, subscription available based on maximum number of users and billed in months

- Paul Castro, etc. The rise of serverless computing, CACM 2019.12
<https://cacm.acm.org/magazines/2019/12/241054-the-rise-of-serverless-computing/fulltext>

无服务器 / Serverless Computing

Figure 1. High-level serverless FaaS platform architecture.



- Paul Castro, etc. The rise of serverless computing, CACM 2019.12
<https://cacm.acm.org/magazines/2019/12/241054-the-rise-of-serverless-computing/fulltext>

小结及研究点

• 虚拟机如何联网

- 虚拟网卡、宿主机内的网络模拟
- 跨宿主机组网：二层组网与三层组网
- 按需隔离、VPN与SDN

• 轻量级容器

- 轻量级化的原理，和VM的比较
- DevOps的概念，虚拟机的编排
- FaaS：将轻量化推进到极致

• 容器研究

- 高效、隔离、迁移、快速启动
- 基于容器的微服务架构、FaaS架构
- 保证QoS质量

参考文献

- David Bernstein, Containers and Cloud: From LXC to Docker to Kubernetes, IEEE Cloud, 2014.09.
<https://ieeexplore.ieee.org/document/7036275>
- Paul Castro, etc. The Rise of Serverless Computing, Communication of ACM, 2019.12.
<https://ieeexplore.ieee.org/document/7036275>
- Docker for Windows.
 - <https://www.cnblogs.com/stilldream/p/10627831.html>
 - <https://www.cnblogs.com/stilldream/p/10634620.html>
 - <https://www.cnblogs.com/stilldream/p/10636049.html>
 - <https://www.cnblogs.com/stilldream/p/10637008.html>

谢谢！