第三章：网络设备与异构设备的虚拟化技术概述

2021.9

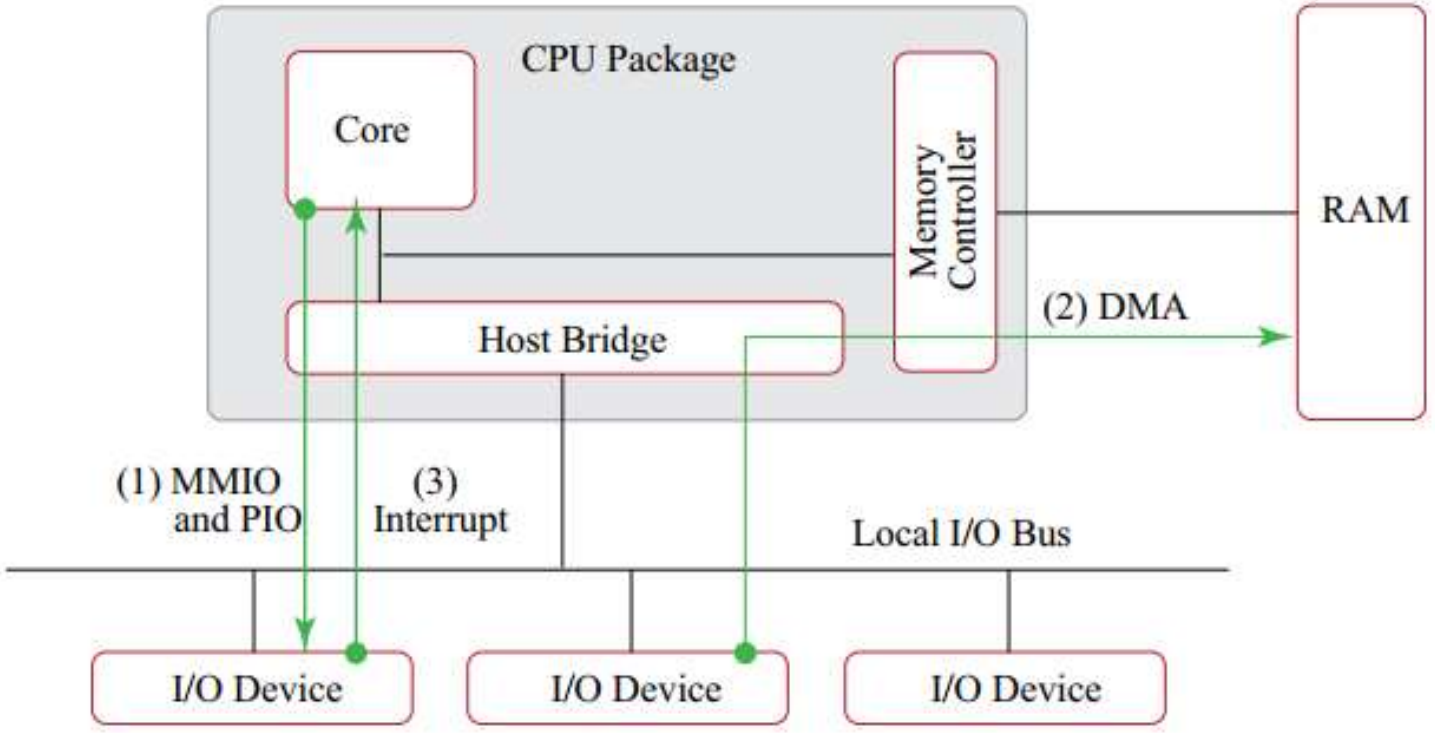
上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

目录 Contents

1. 半虚拟化与设备直通

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

网卡如何工作



I/O设备与CPU核内存访问的三种方式

相关基础概念

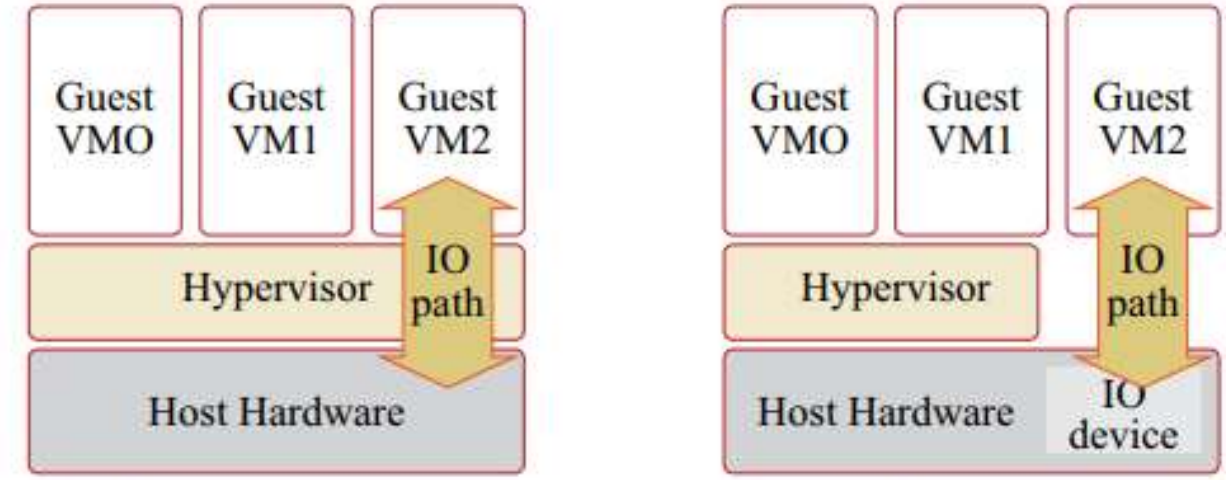
- **MMIO:** Guest's MMIOs are regular loads/stores from/to guest memory pages, so the hypervisor can arrange for these memory accesses to trap by mapping the pages as reserved/non-present (both loads and stores trigger exits) or as read-only (only stores trigger exits).
- **PIO:** Guest's PIOs are privileged instructions, and the hypervisor can configure the guest's VMCS to trap upon them.
- **DMA:** Emulating DMAs to/from guest memory is trivial for the hypervisor, because it can read from and write to this memory as it pleases.
- **Interrupt:** the hypervisor can use the VMCS to inject interrupts to the guest. Each interrupt causes at least two VM-Exit(direct interrupt delivery and VT-d posted interrupt).

I/O 虚拟化

- 基于软件的 I/O 虚拟化:
  - 功能丰富
    - I/O Sharing, Security, Isolation, Mobility.
  - 简化管理
    - Encapsulate VM state->VM Suspend/Resume, Live Upgrade and Live migration.
- 硬件辅助的 I/O 虚拟化:
  - High performance I/O: throughput, latency.
  - Forgo certain virtualization benefits.

I/O 虚拟化的主要分类

- I/O interposition (I/O emulation and Paravirtualization)
- Direct Device Assignment (Passthrough)

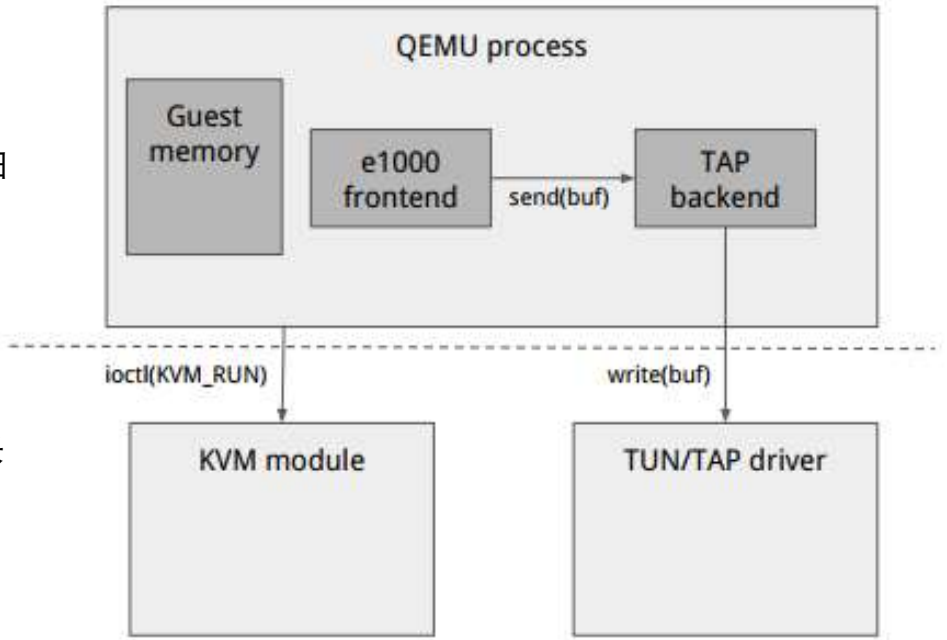


(a) Emulation/paravirtualization

(b) Direct device assignment

I/O Emulation

- Hypervisor既要阻止虚机直接访问host上的物理设备，又要使虚机具有独占设备的假象
- I/O emulation是一种常见的I/O虚拟化方案，核心是对虚机的I/O相关操作进行trap，接着由hypervisor进行指令的模拟
- **前端：**QEMU 设备模拟层，理解host上设备的特性，并与虚机内的驱动进行交互
- **后端：**前端所使用的，将虚拟设备的功能转换为对host上真实设备资源的控制指令的具体软件
- **TAP：**虚拟网络设备的一种，现代linux中自带。可以对TAP设备所连接的进程中的网络数据包进行转发

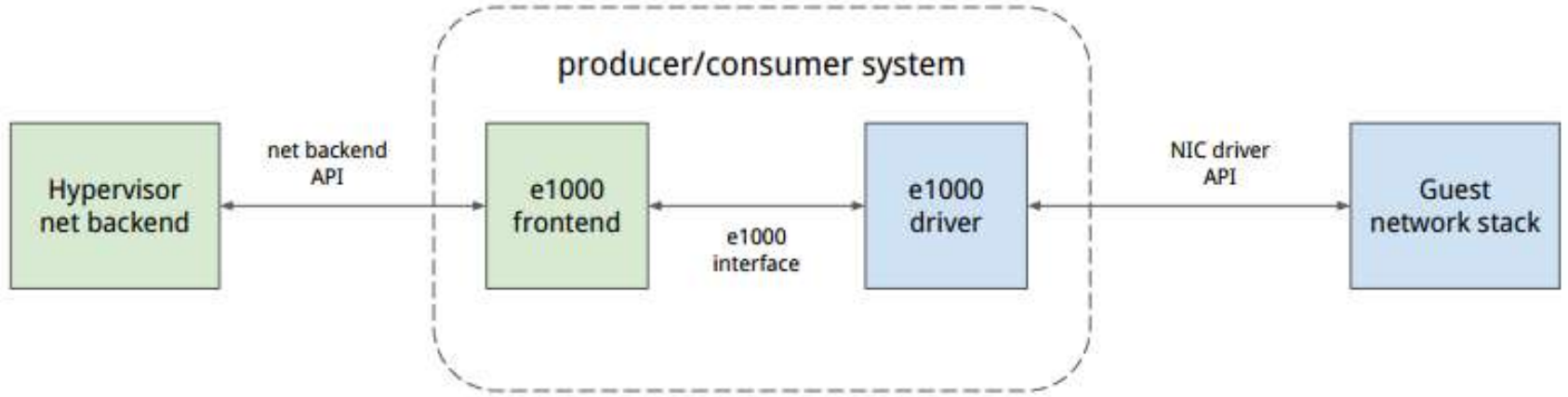


I/O Emulation

- 客户机确信自己对I/O设备有独占控制能力
- Hypervisor 对I/O相关请求进行 trap & emulate
- 优点:
  - 全虚拟化，对客户机操作系统无需任何修改
  - 容易获得软件设备状态，更好的灵活性:
    - 动态分配与调整物理设备与虚拟设备的对应关系
    - 热升级、热配置、热迁移
  - I/O consolidation 提升系统效率，降低开销
  - I/O 聚合：更高性能、更好稳定性.
- 缺点:
  - 无法完全利用硬件特性加速I/O性能
  - Host上存在一定的性能开销

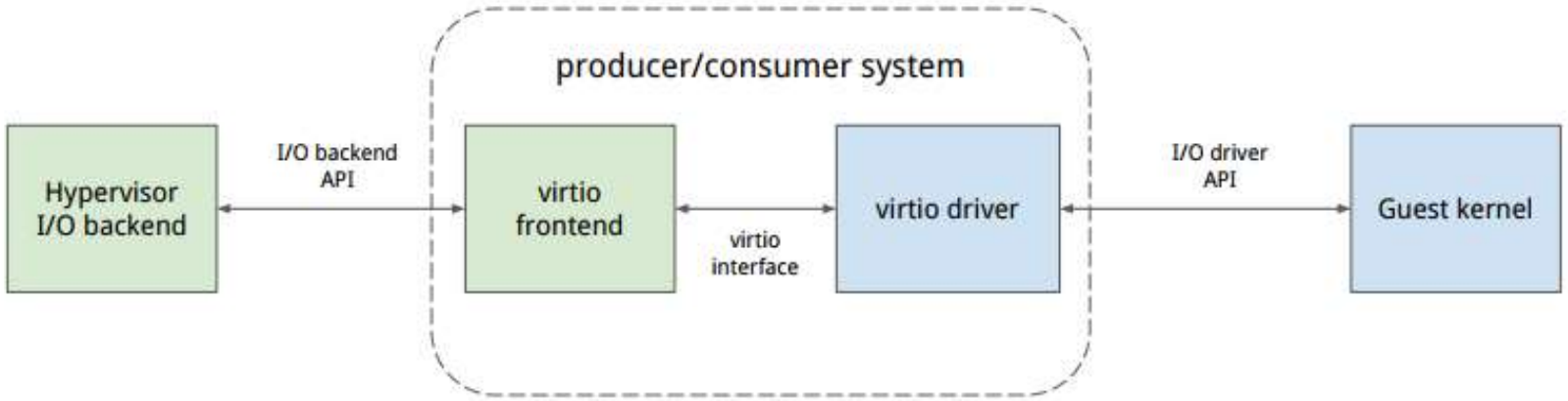
I/O 半虚拟化概念

- 能否构建更简单、更高效的生产者/消费者模型?
  - 匹配现有 e1000 驱动+虚机前端?



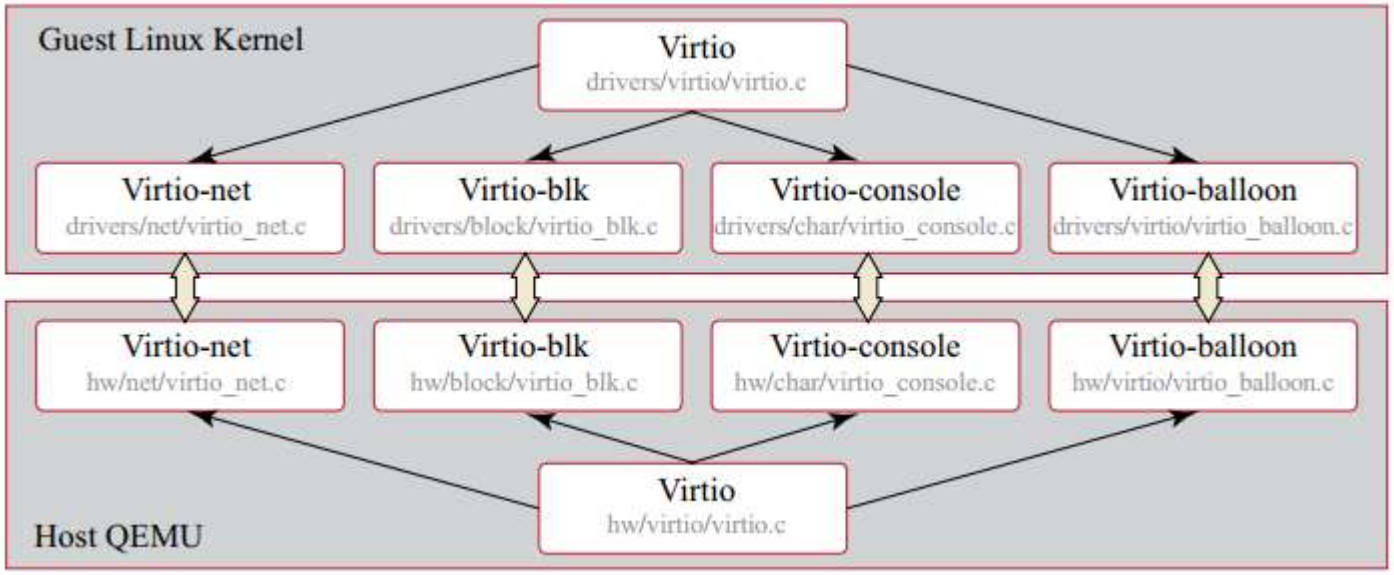
VirtIO 标准

- KVM/QEMU中提供的半虚拟化框架叫VirtIO，其提供了通用的guest-host通信接口和传输机制。



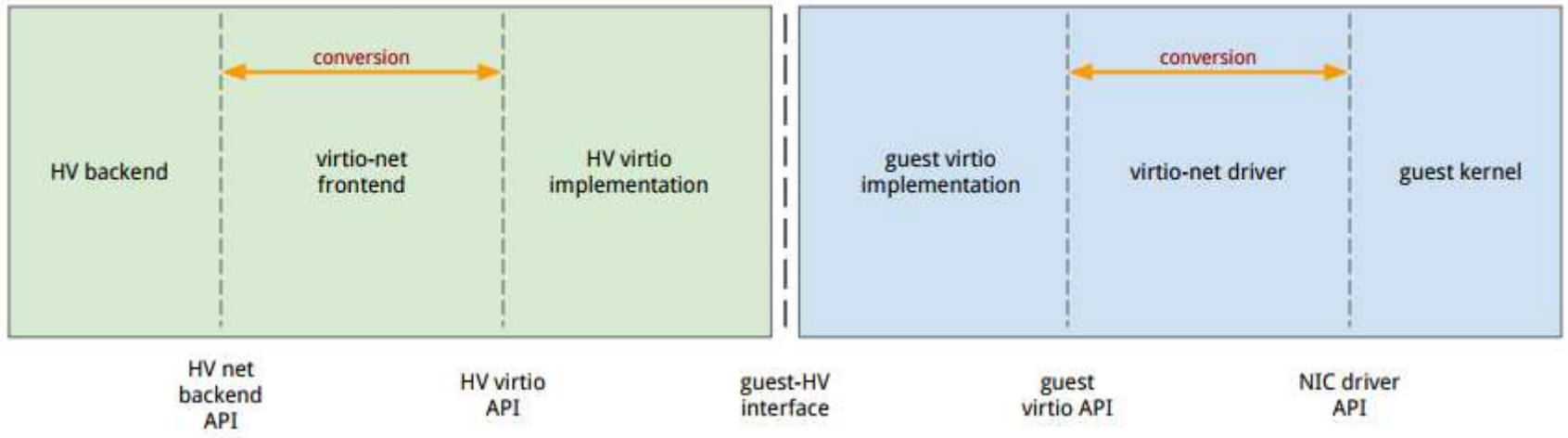
VirtIO 标准

- 同样的机制可以应用到网卡以外的其他设备上，例如块设备、串口设备等。本质上，所有类型的I/O事件可以看作一个生产者/消费者的系统在互相交换信息。



VirtIO 标准

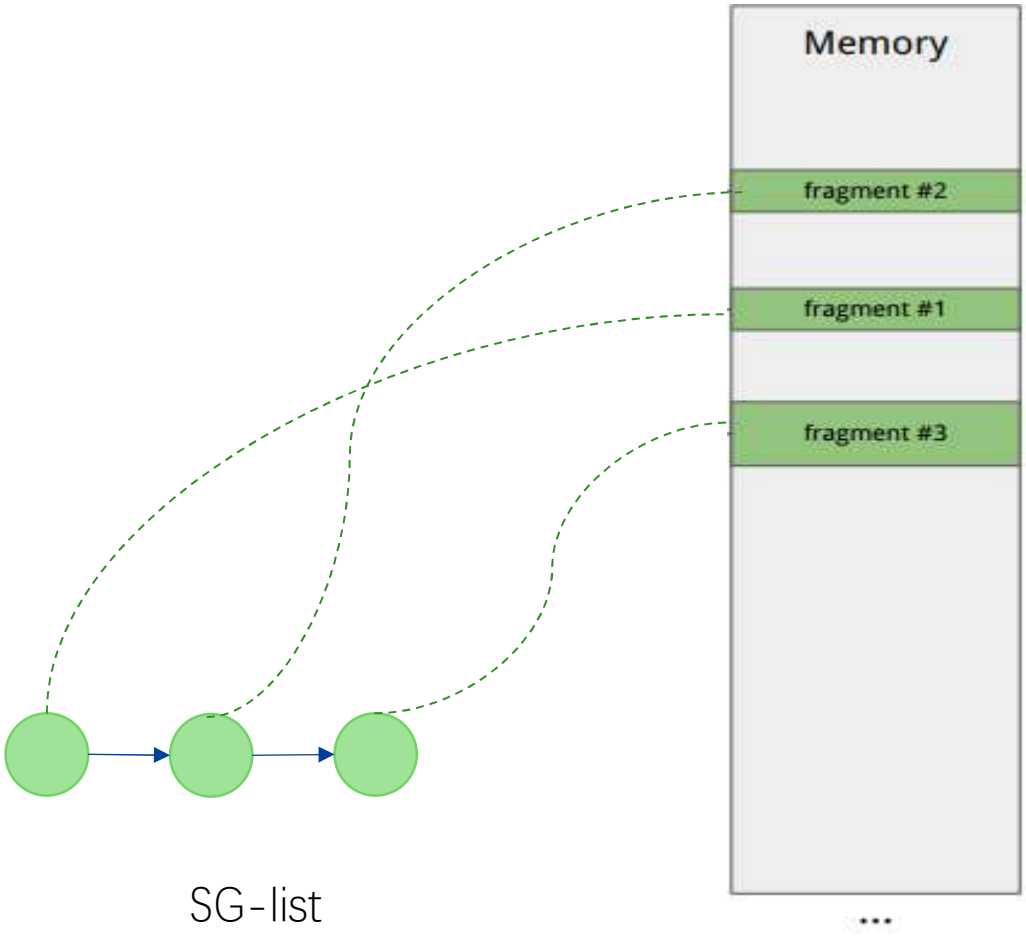
- VirtIO驱动的任务是将面向不同操作系统的各类信息组织方式（例如Linux网络驱动的skb对象）转换为VirtIO信息格式，反之亦然。
- virtio-net frontend 在hypervisor侧执行类似的功能，将VirtIO信息格式转换为后端能理解的信息格式。



Scatter-Gather List (SGL)

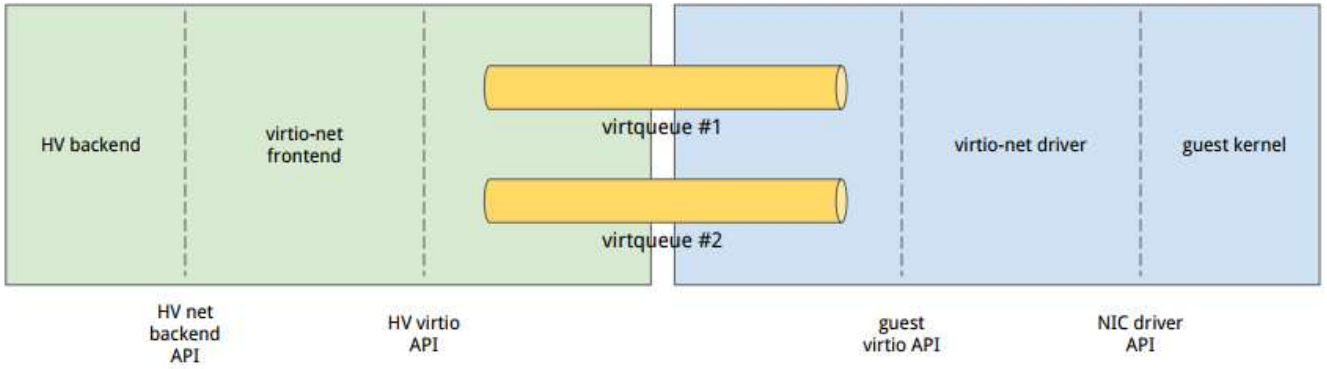
- VirtIO 使用 Scatter-Gather (SG) lists 进行数据交互

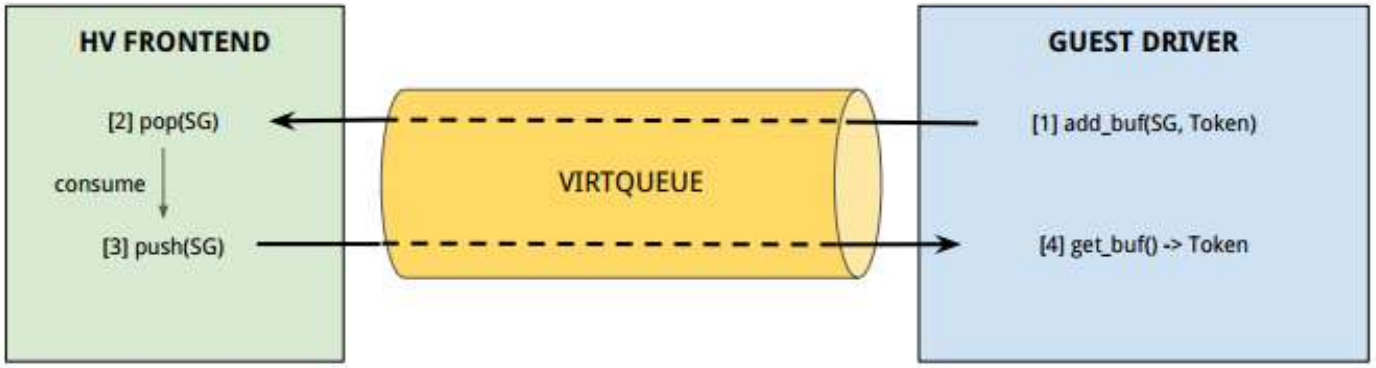
- 一个 SG list 从概念上是一个物理地址的列表



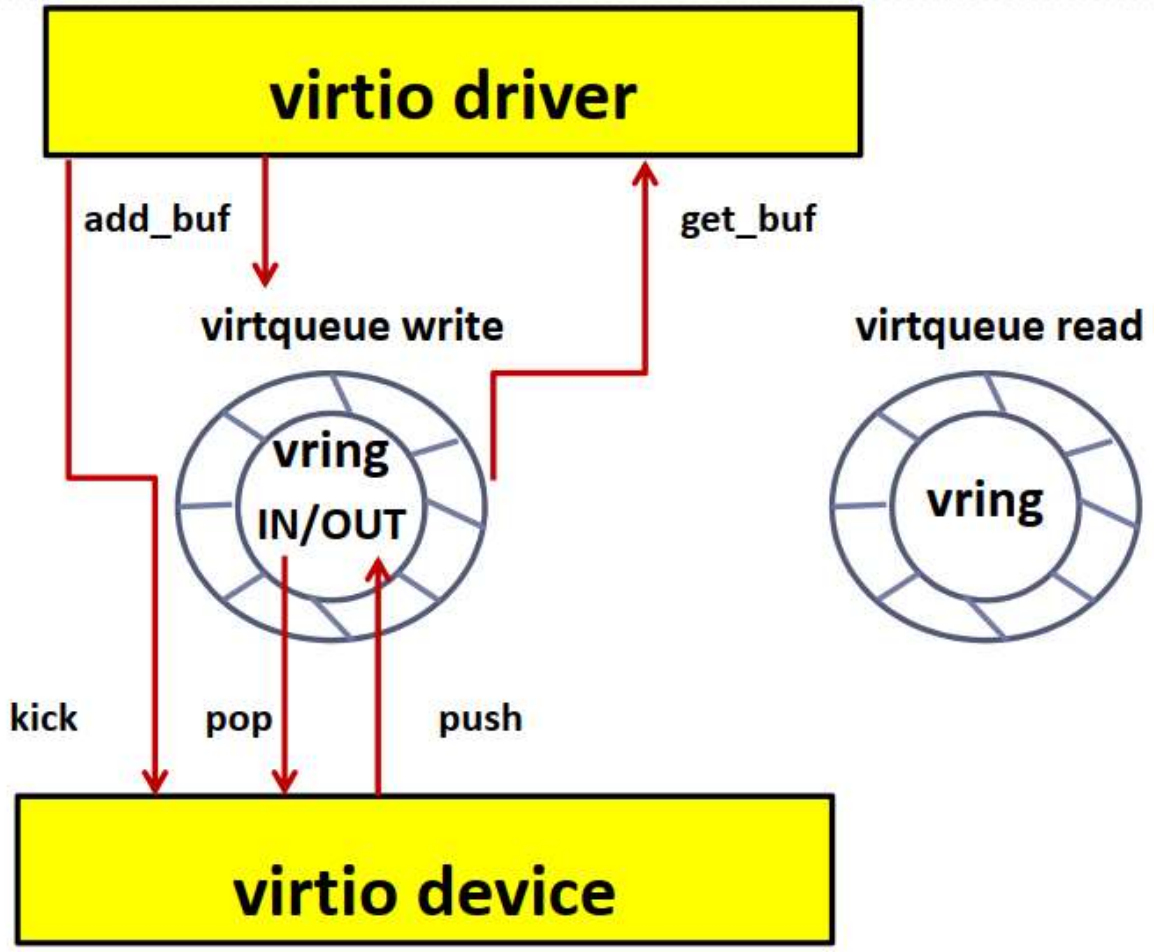
Virtqueues

- VirtIO中的核心是Virtqueue (VQ)，即一个guest驱动填充信息并由hypervisor读取的SG队列。输出的 SGs 由guest产生, 输入的 SGs 由guest读取来自hypervisor的信息。一个设备可以由多个VQ，具体有几个VQ由驱动协商得到。



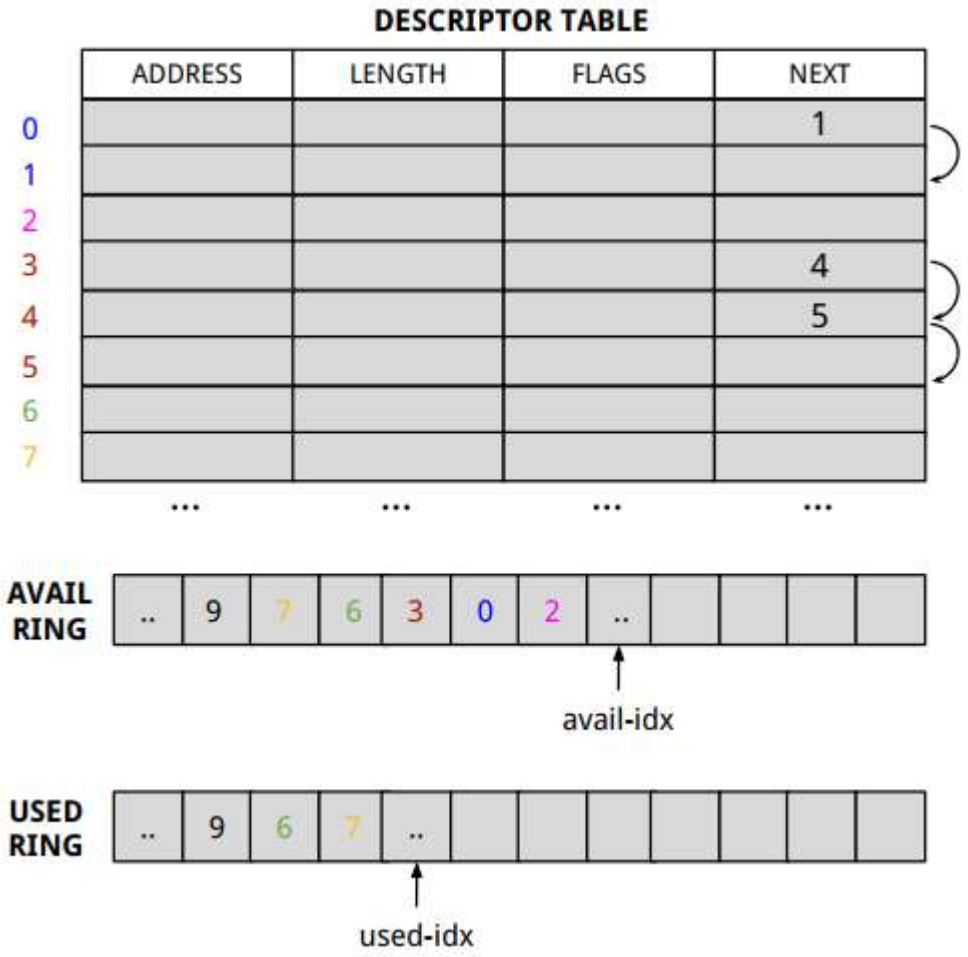


Virtqueues


virtio driver
add_buf
get_buf
virtqueue write
virtqueue read
vring
IN/OUT
vring
kick
pop
push
virtio device

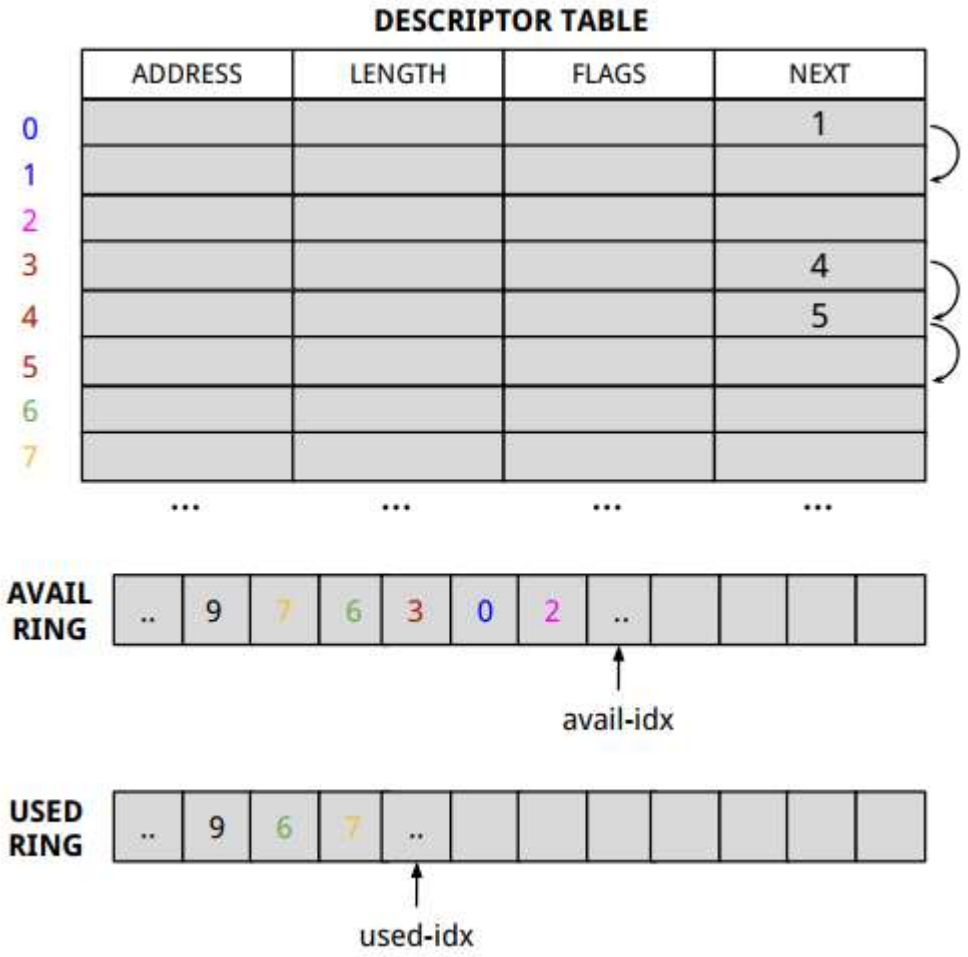
Vring的实现

- 通常需要三个数据结构来组成一个VirtQueue:
  - Descriptor Table
  - Avail Ring
  - Used Ring
- Multi-fragments SG
- Out-of-order SG consumption
- Optimized cache usage.



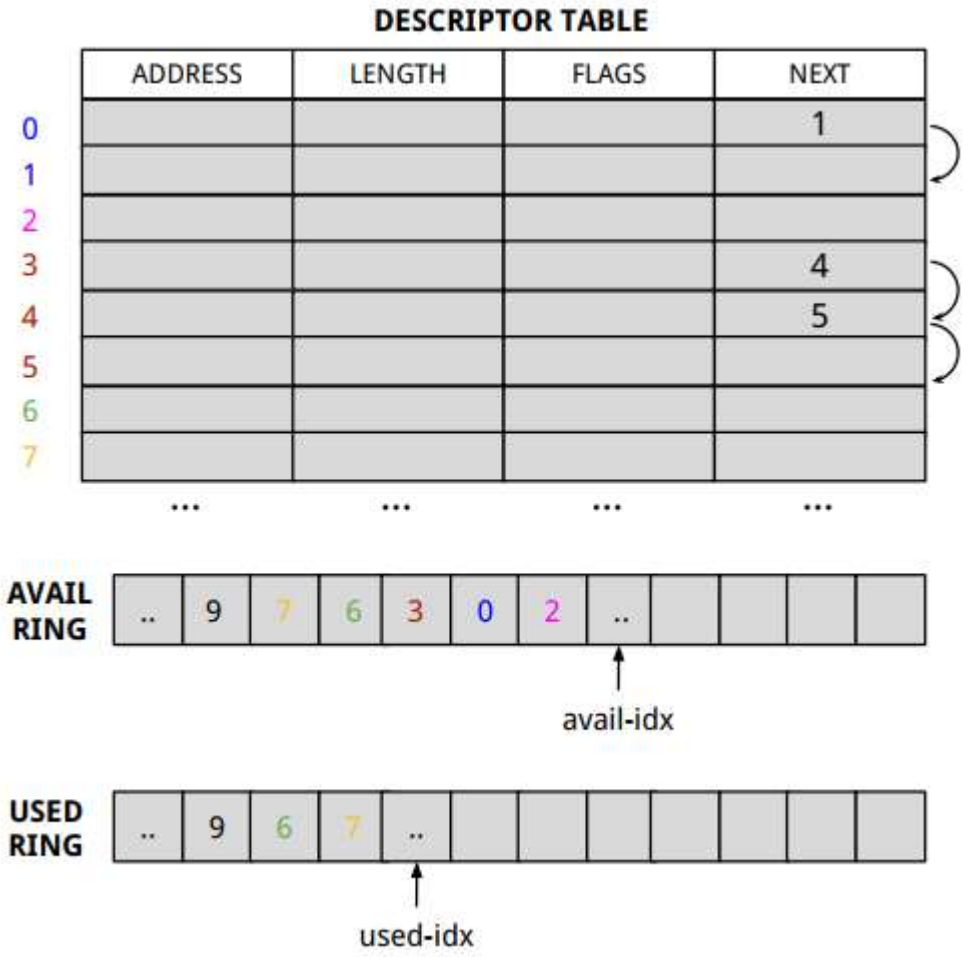
Vring的实现

- Descriptor Table
  - 数据缓存区的物理地址和长度
  - 如果是链接的descriptor，外加一个指向下一块descriptor的指针
  - DMA flag (输入或输出).
- 仅由guest写入数据
- 例如：一个有N分段的SG映射到N个descriptor



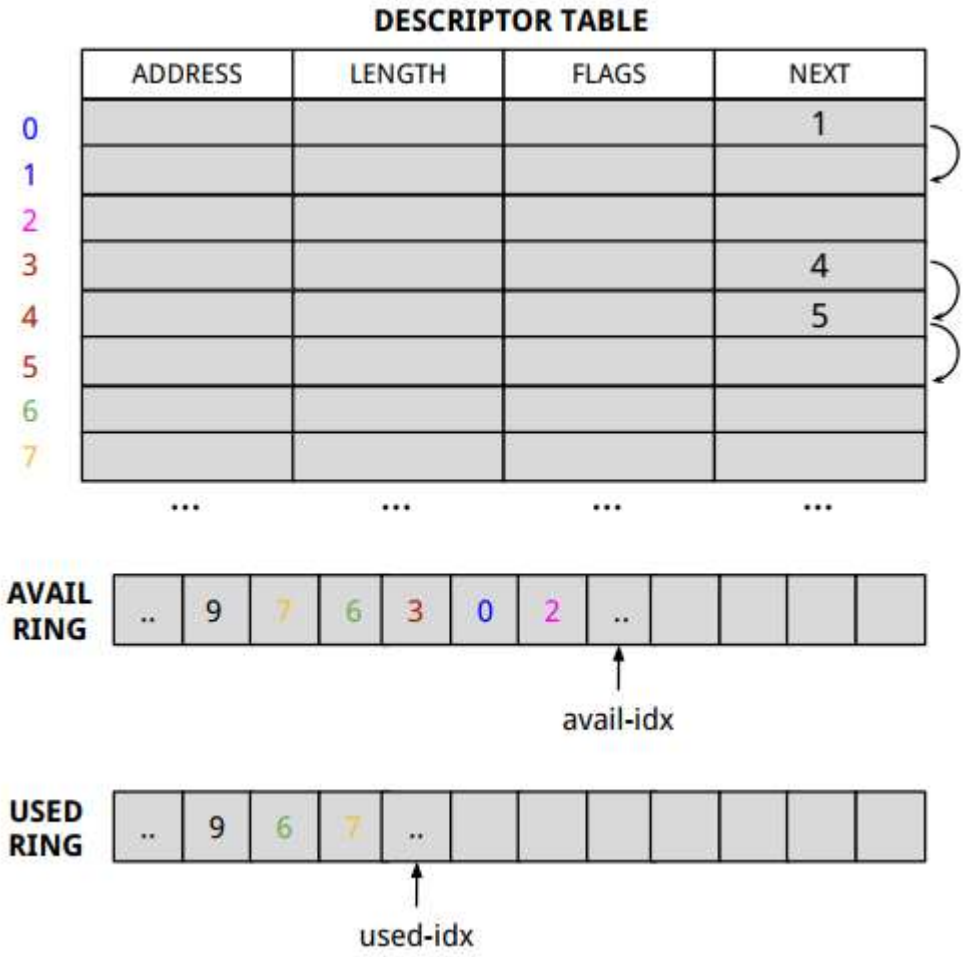
Vring的实现

- Avail Ring
  - Guest用这个ring告诉hypervisor有多少可用的descriptor.
  - Each slot contains a head which is an index in the desc table-to the first desc of a SG.
  - Avail-idx like the tail register of e1000 ring but different. It is stored in memory, not a MMIO register.
- Decouple counting of available SGs from notification.



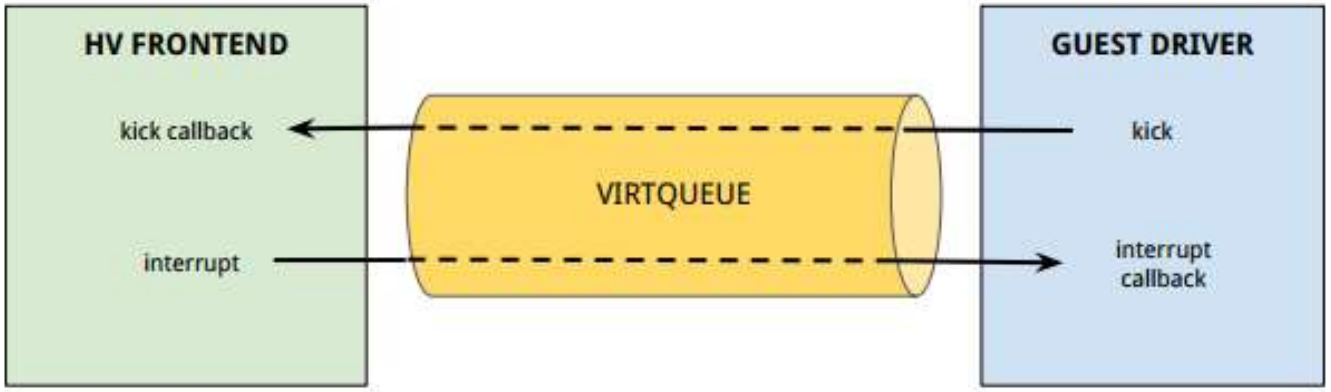
Vring的实现

- Used Ring
  - Hypervisor 用这个ring告诉虚机哪些 SGs 已经被使用.
  - Each slot contains the head of the consumed desc and a length field(Output 0, Input total length).
  - Used-idx like the head registe of e1000 ring but different. It is stored in memory, not a MMIO register.



最小化通信开销

- 当前端希望通知后端，告诉后端可以使用SG时，前端使用 **VQ kick** 机制。其实现方式为对一个指定的寄存器（虚拟地址）进行写操作。类似的，当后端希望通知前端时，后端生成 **VQ interrupt**.
- VQ kick 和 interrupt 是 VirtIO 接口的重要部分. 前端应该生成尽可能多的SG，然后再KICK后端。类似的，后端应该使用掉尽可能多的SG，再生成中断通知前端。这样的设计能够将通信开销平均到尽可能多的SG处理时间中，从而最小化通信开销。



性能对比摘要

	Metric	e1000	Virtio-net	Ratio
Guest	throughput (Mbps)	239	5,230	22x
	exits per second	33,783	1,126	1/30x
	interrupts per second	3,667	257	1/14x
TCP segments	per exit	1/9	25	225x
	per interrupt	1	118	118x
	per second	3,669	30,252	8x
	avg. size (bytes)	8,168	21,611	3x
	avg. processing time (cycles)	652,443	79,132	1/8x
Ethernet frames	per second	23,804	–	–
	avg. size (bytes)	1,259	–	–

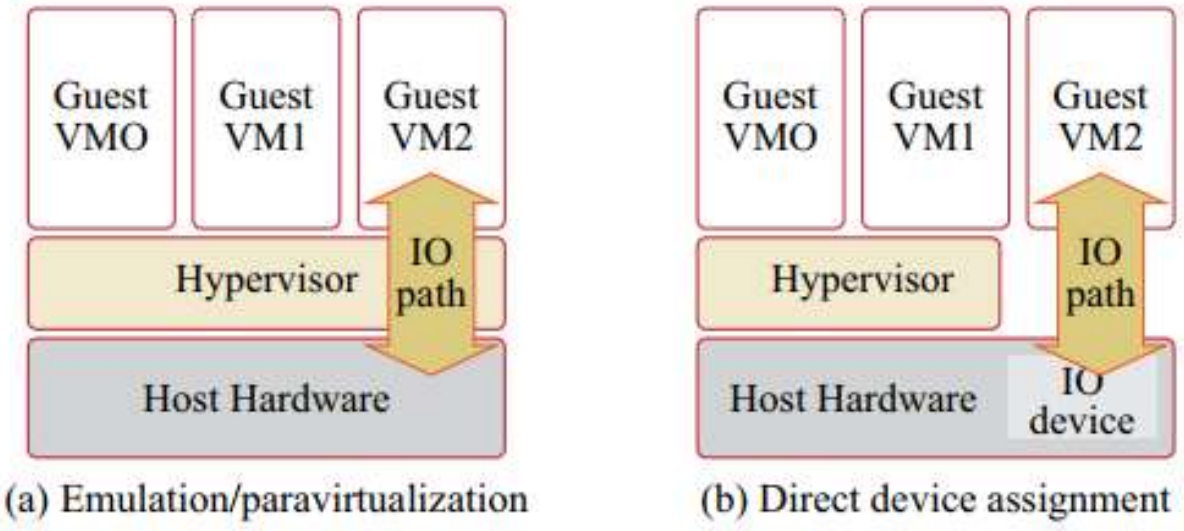
Netperf TCP stream running in a Linux 3.13 VM on top of Linux/KVM (same version) and QEMU 2.2, equipped with e1000 or virtio-net NICs, on a Dell PowerEdge R610 host with a 2.40GHz Xeon E5620 CPU.

I/O 半虚拟化总结

- 重新设计了虚拟设备及其接口
- 客户机需要特殊的驱动用以最小化设备模拟的开销
- 优点:
  - 具有绝大多数全虚拟化的优点
  - 性能相比全虚拟化更好
- 缺点:
  - 距离裸机性能仍有差距（尤其是大数据包时）
  - 需要客户机的配合，损失一部分灵活性
    - 需要客户机安装半虚拟化驱动（例如在windows中使用virtio）
    - 需要在各种客户机OS中实现半虚拟化驱动

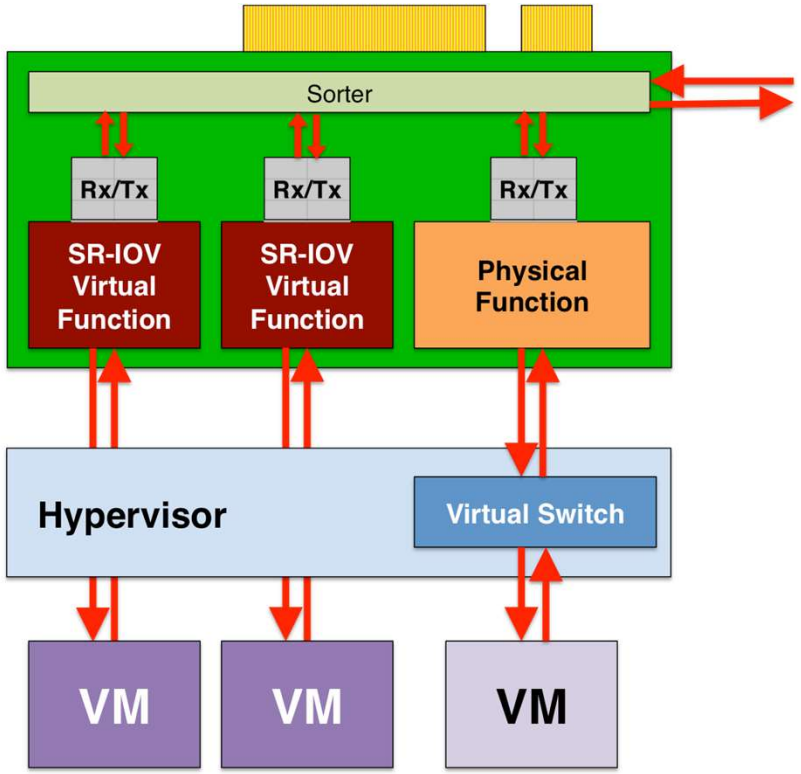
设备直通

- 虽然I/O emulation与半虚拟化均能够提供安全的I/O虚拟化，但是他们性能存在较大损失
- 设想：对于物理设备 d, 如果 hypervisor 能够将对 d 的访问权限唯一授予某个虚机 v, 那么其他虚机，甚至是 hypervisor 都无法驱动设备 d，因此虚机 v 将获得该设备的直接访问能力，或者称为：设备直通

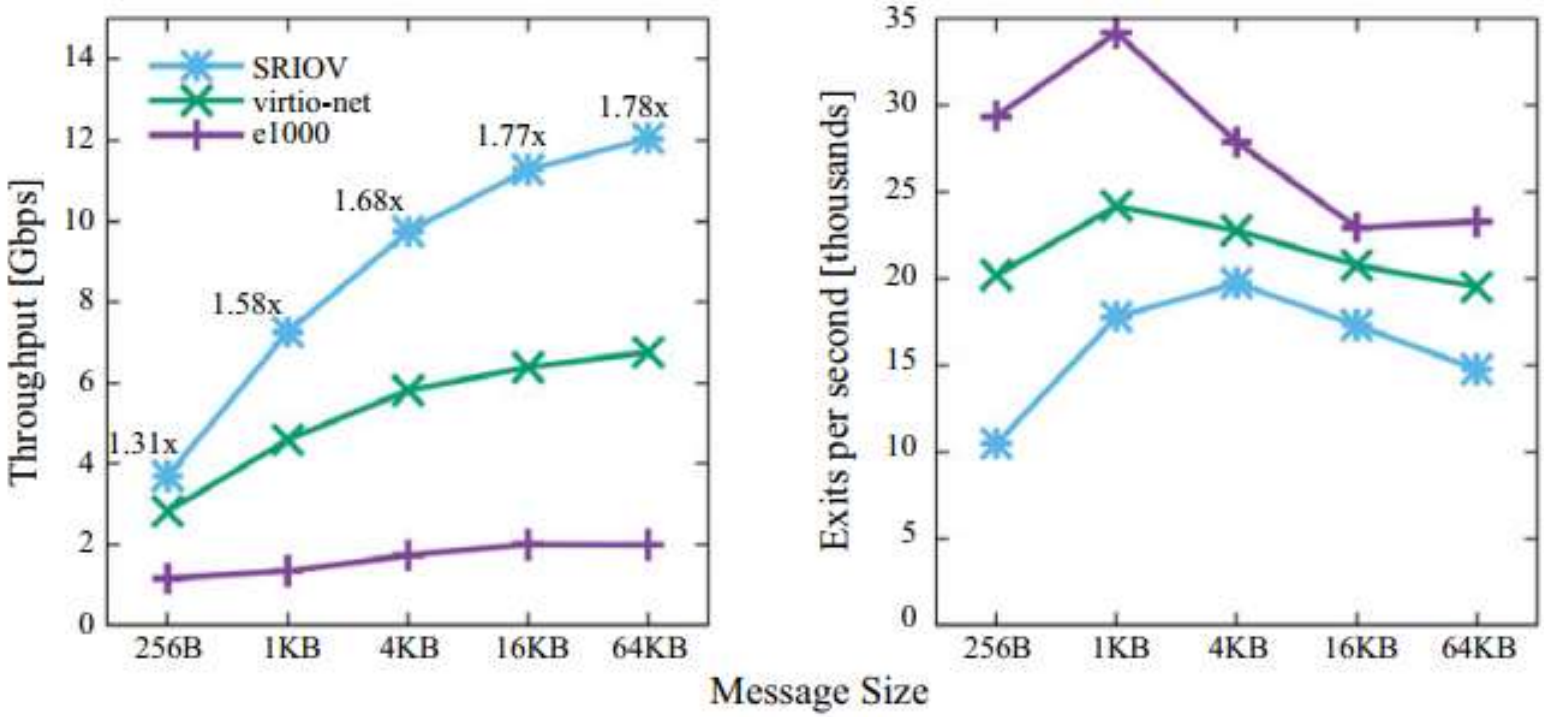


单根I/O虚拟化（SR-IOV）

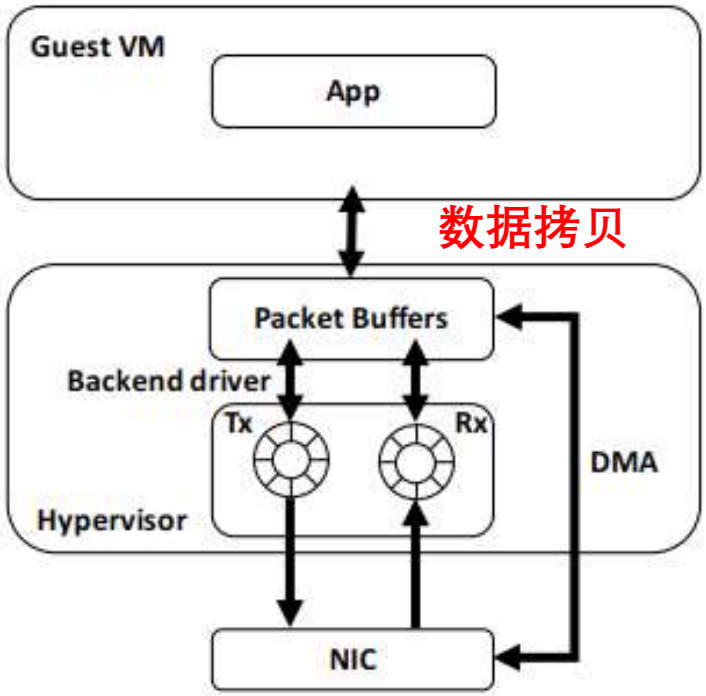
- SR-IOV 标准源自PCI-SIG，其为the PCI Express (PCIe) 标准套件进行了扩展，使其支持多个虚机共享同一个PCIe的物理资源
- 一个物理设备称为一个 Physical function (PF)，其可以生成多个虚拟设备，称为 Virtual function (VF)。每个虚拟设备有自己的DMA流和中断向量。



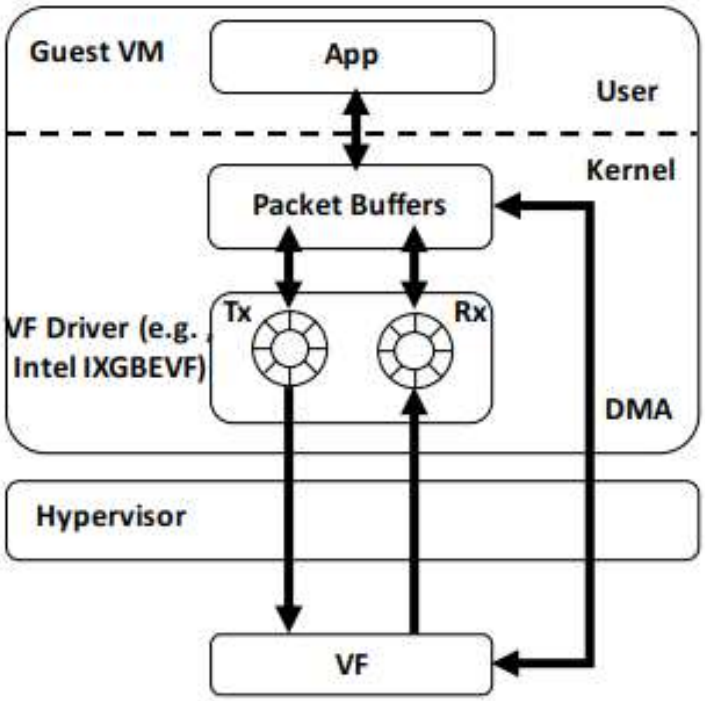
SR-IOV简要性能对比



SR-IOV系统



(a) Para-Virtualized I/O



(b) SR-IOV with Intel IXGBEVF Driver

设备不允许无效DMA地址，需要pin住虚机的page buffer地址

SR-IOV总结

- 优势:
  - 硬件自虚拟化，无设备模拟开销
  - 高性能：高吞吐，低延迟
  - 有效解决可扩展性问题 (Intel 710 系列网卡的每个 PF 支持创建 64 个 VF)

- 劣势:
  - 牺牲了灵活性
  - 无法利用内存优化机制
  - 存在热迁移的困难
    - Pre-copy阶段的脏页难以记录
    - 物理设备状态的获取与恢复非常困难

2. 智能网卡与虚拟化

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

智能网卡成就裸金属服务器

裸金属服务器是什么

A 'bare-metal server' is a computer server that is a 'single-tenant physical server.

—— Wiki百科

裸金属服务器（Bare Metal Server，BMS）是一款兼具虚拟机弹性和物理机性能的计算类服务，为您和您的企业提供专属的云上物理服务器

—— 华为云

既然有了传统虚拟化技术，为什么还需要裸金属服务器

或许，我们可以试着从云运营商的角度进行理解

智能网卡成就裸金属服务器

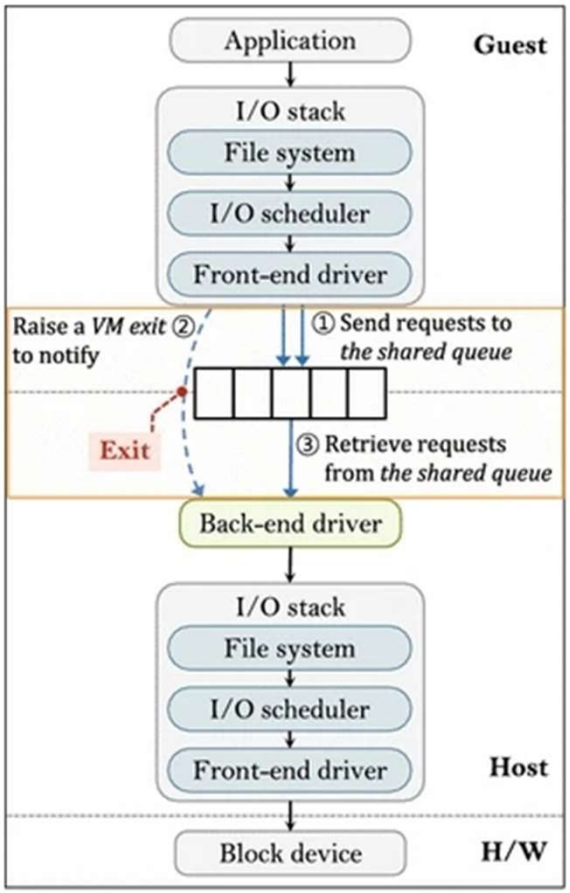
传统虚拟化效率相对低下

1. I/O路径长

2. 控制面通知策略低效

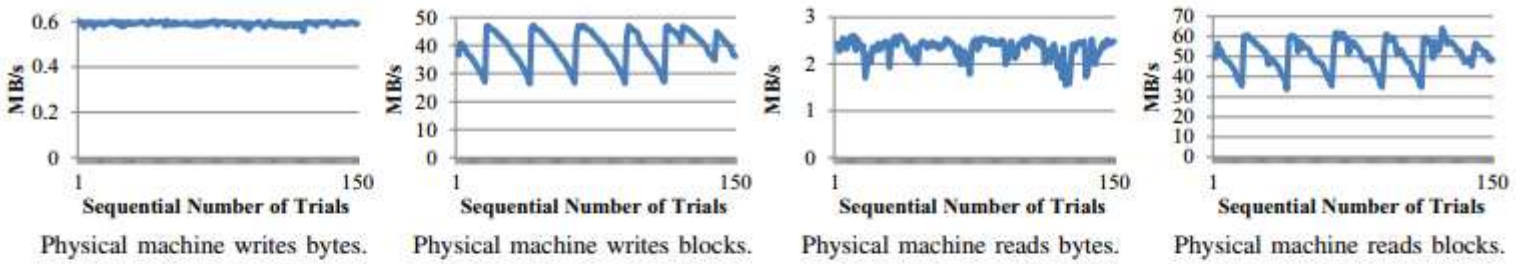
3. 数据面拷贝操作低效

4. 优化方案的局限性

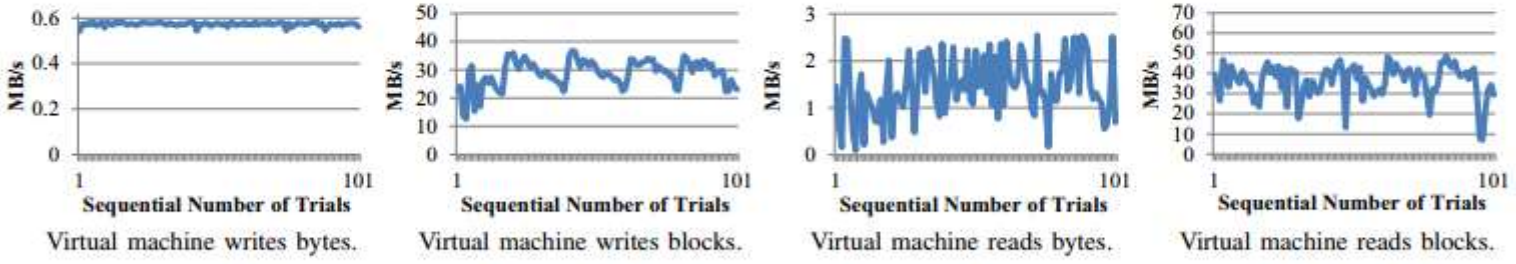


智能网卡成就裸金属服务器

传统虚拟化性能抖动严重



Physical machine disk performance



Virtual machine disk performance

相同配置的物理机/虚拟机磁盘性能对比（2017 IEEE paper)

智能网卡成就裸金属服务器

思考题：

如何将硬件资源“池化”，使物理硬件设备“流动”于不同物理机之间，从而具备传统虚拟化的灵活性。

提示：

1. **网卡的使用至关重要（为什么）**
2. **软件管理层十分必要（为什么）**

智能网卡成就裸金属服务器

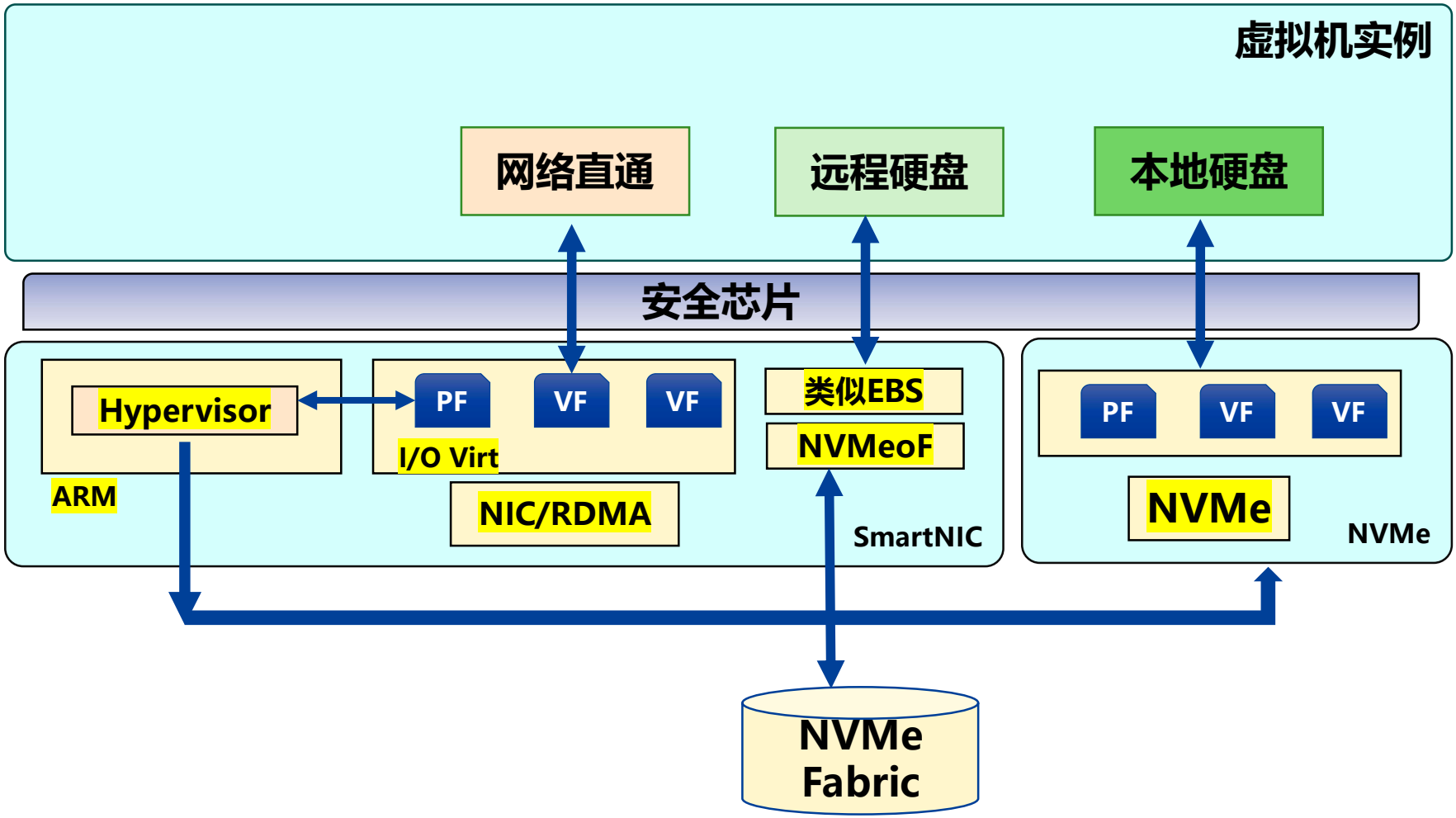
思考题：

1. 使用了软件是不是意味着性能又要回到从前（传统虚拟化）了
2. 运行软件的CPU是否还能原价卖出
3. 如果尽可能压缩软件的开销

提示：

1. 传统虚拟化场景的软件有哪些，和目标场景有何不同
2. 硬件软化是以性能为代价增加灵活性，是否可以反向操作以提高性能

SmartNIC+hypervisor软件层



智能网卡成就裸金属服务器

思考题：

hypervisor虽然没有消耗服务器CPU核，但消耗了SmartNIC的核，这样做的意义何在？改为消耗单个服务器CPU核，同时将SmartNIC核卖给用户，是否可以达到相同的效果呢？

提示：

1. **服务器CPU核和SmartNIC核的区别是什么**
2. **哪个核更适合运行hypervisor**

智能网卡的功能与挑战

- 硬件组成：CPU + Mem + FPGA + NIC + 加速器
- 软件：整合服务器硬件、操作系统、网络、虚拟化技术、云管理平台等。
- 在网卡上完成部分计算任务，有成功的实践，
  - LSO/LRO: 发送和接收方向 Large Segment Offload 和 Large Receive Offload
  - GSO/GRO: Generic Segmentation Offload 和 Generic Receive Offload
  - TSO/UFO: TCP Segmentation Offload 和 UDP fragmentation offload
  - 很多ASIC网卡上，就有多种固化、简单的卸载实例
- 难点是复杂任务的卸载：
  - I/O虚拟化的过程的卸载：避免类似SR-IOV直通带来的可管理性缺失。
  - hypervisor的卸载：关键指令在外设无权执行，但决策计算可以卸载。
  - 计算密集型应用的任务卸载：加解密、压缩为代表的计算密集型应用。
  - 云安全性增强的方法：秘钥管理和保护。

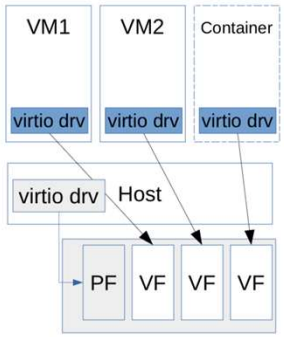
裸金属服务器现状

- 云计算大厂完全自研：亚马逊、阿里、华为高度定制。
- 中型以下云商，独立研发能力不足，整体定制难度高。
  - “智能网卡+通用服务器”的方案比较合适。
- 目标模式：基于Intel、AMD、Mellanox等芯片厂商的与底层技术，为云计算、大数据、人工智能等场景定制适配平台（软件定义基础设施SDI）。
  - 组成：SmartNIC + 通用服务器。
  - 特点：基础架构通用，功能可叠加、可定制组合。

Inventec c5020x：virtio后端卸载

Hardware virtio SRIOV

- **Hardware SRIOV support**
  - Virtio PF to generate multiple virtio VFs
  - VFs can serve difference VM/Container
- **Accelerate NFV**
  - Virtio-net driver only

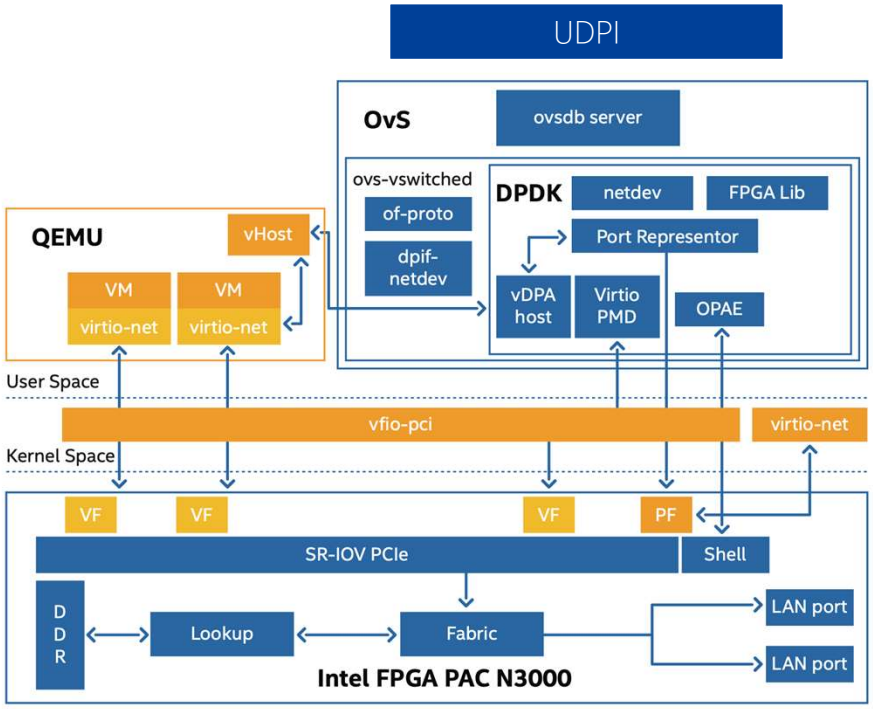


- **传统SR-IOV网卡直通效率高，但是迁移困难。**
- **利用Bonding-driver会导致用户看到多张网卡。**
- **如果Hypervisor将其抽象后统一提供给虚拟机，会屏蔽掉网卡上部分硬件加速特性。**

可以定制通过PCI-e呈现virtio接口编制的设备：

关键性问题集中在kvm/Qemu软件框架的配套设计上

Intel PAC N3000：hypervisor卸载



. The OvS acceleration solution Data Plane Development Kit (DPDK) architecture overview

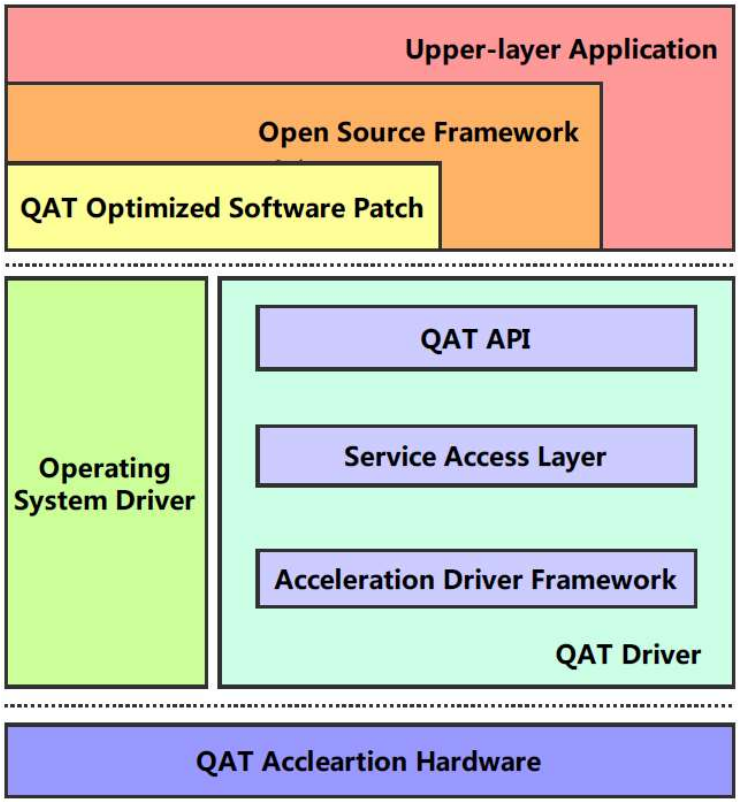
1. 系统资源切分
2. 无缝弹性伸缩
3. 快速实时迁移
4. 无盘启动、
5. Hypervisor 卸载至网卡
6. OVS交换任务卸载至网卡
7. I/O卸载与直通
8. 安全强化
9. Instance定制与维护
10. BMaaS 成云管理 (VPC)

3. 智能加速器-传算融合优化

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

QAT加速器

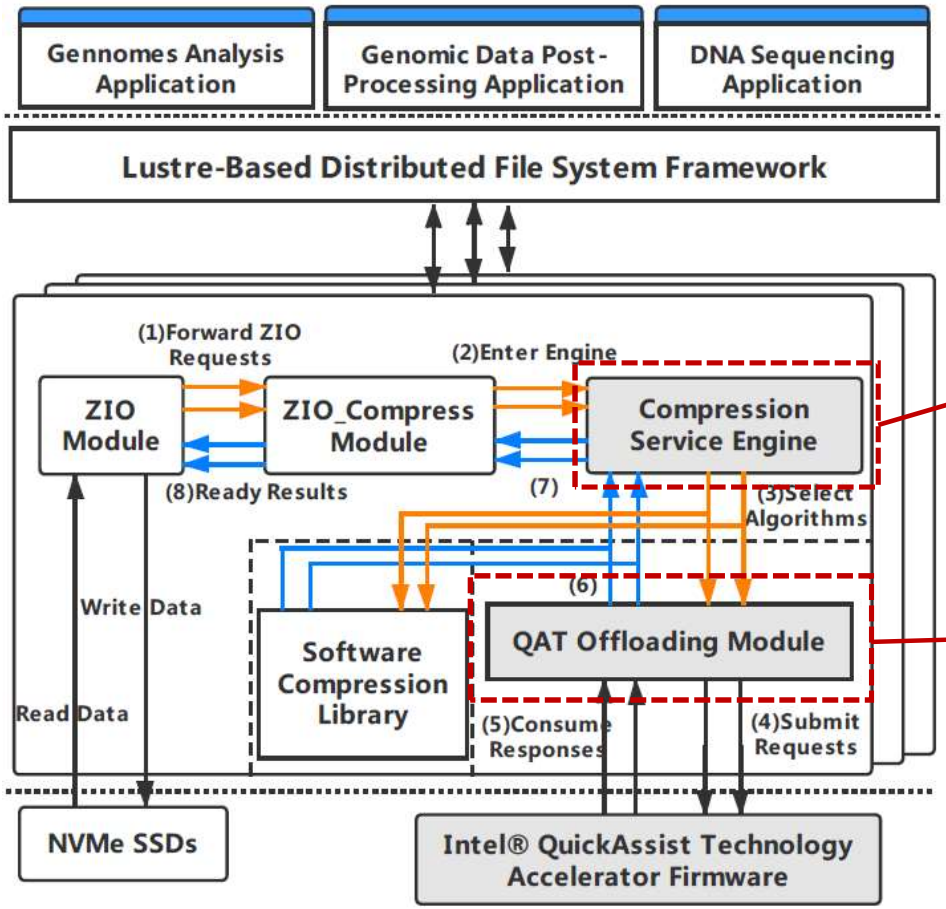
- Intel QuickAssist Technology 加速器
  - 支持数据压缩与数据加解密
  - PCIe设备：DMA直通访问
  - 服务访问层 (Service Access Layer)：提供服务抽象接口
  - 加速驱动框架层(Acceleration Driver Framework)：提供硬件的通信服务



融合异构算力的通用框架

- 通用型加速设备：Microsoft Xpress、TerseCades和Intel Compression
- 专用型加速设备：QAT、Cavium 和 AHA

案例1. 文件压缩卸载加速（QZFS）

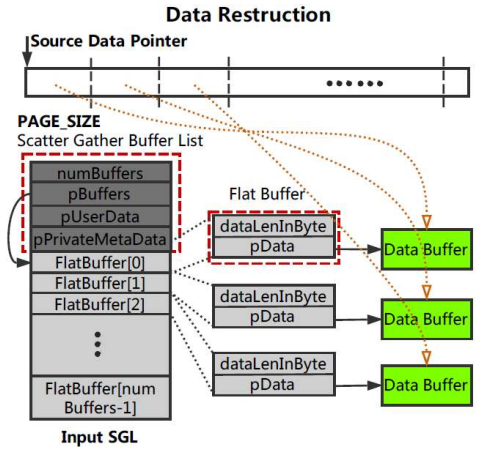


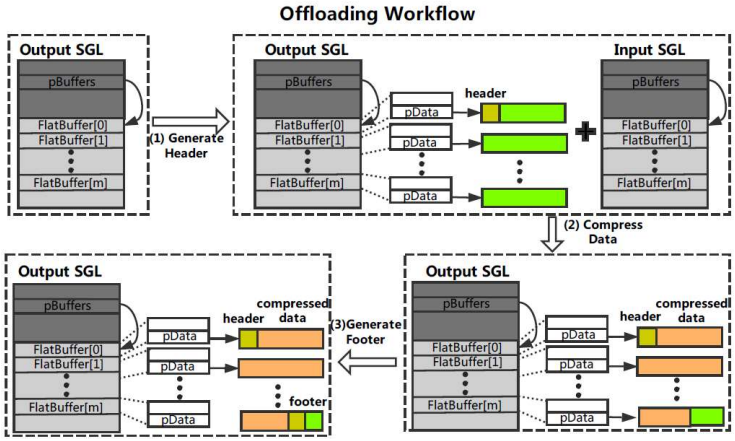
- 基于ZFS文件系统的基本框架
- 可集成于Lustre分布式文件系统
- 使用QAT的硬件压缩技术
- 软硬件协同设计

QAT压缩模块流程与优化点

软硬件间高效协同交互框架设计及针对压缩的优化

- QAT卸载GZIP数据压缩及解压缩任务
- 数据采用向量I/O模式： Scatter Gather List
- 数据重构技术：内存零拷贝
- 内存页对齐与缓冲区溢出处理
- 负载均衡与错误恢复





效果综述

- FIO性能测试工具
  - 模拟对QZFS发起不同模式的I/O请求：读写模式，不同块大小和压缩算法
  - 吞吐性能：QAT加速较软件压缩和不压缩分别平均提升了1.78x和1.61x·
  - 耗费效率参数：最高提升4.x倍
- 生物大数据测试工具
  - 大约3TB生物数据来自欧洲基因组计划和国际基因组样本库
  - SAMTOOL和BIOBAMBAM2：五个具有代表性的数据操作
  - 耗费效率参数：最高提升7.x倍
  - 压缩减少的硬盘I/O次数，降低了数据获取的延迟。
  - QAT节省CPU更多用于数据处理，更多CPU用于上层应用。
  - 大数据处理是计算和I/O的混合操作，QAT加速生物大数据处理的标准操作：
    - 转化、合并操作减少了73.37%和63.23%执行时间；
    - 视察、索引操作减少了43.54%和59.67%。

处理相同数据所用时间减少2-3倍

案例2. Web负载卸载加速（QWEB）

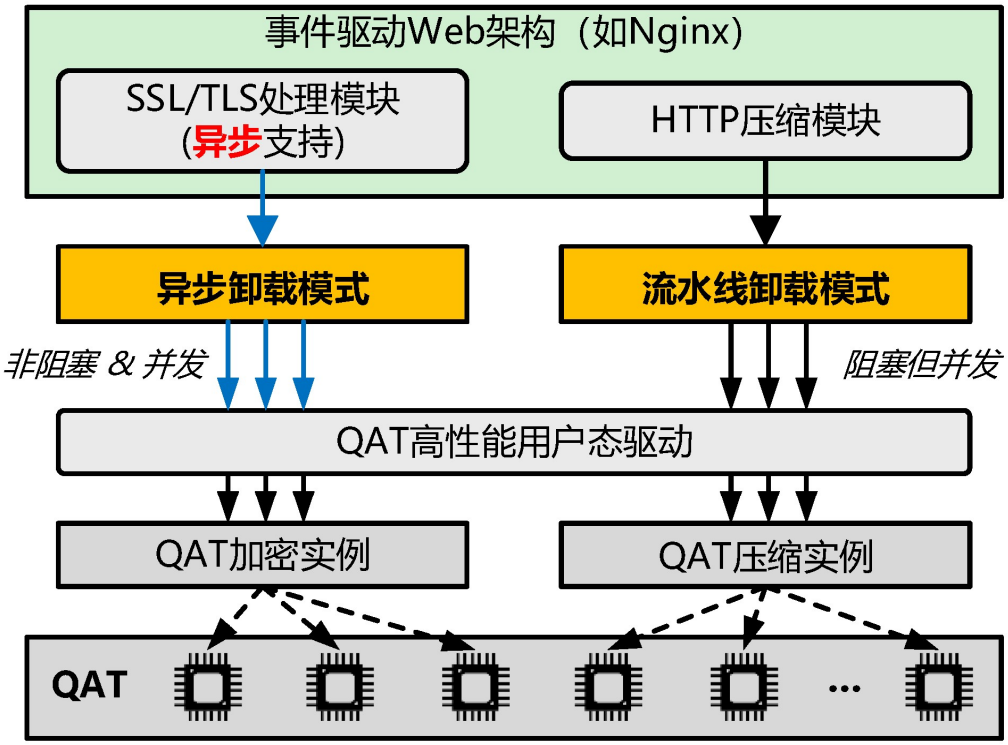
- 云数据中心具备代表性的计算密集型任务——Web负载
  - 安全/加密：HTTPS中的SSL/TLS协议，广泛使用
  - 数据压缩：对HTTP内容的压缩，网站使用率高达80.8%
- 高性能事件驱动Web架构
  - 以Nginx为代表，在单进程事件循环中处理数千个并发连接
  - 相比于线程模型（为每连接创建线程），在性能和可扩展性上均具备优势
  - 市场规模：采用事件驱动Web架构的应用，包括 Nginx、Cloudflare Server和LiteSpeed等，在前100万网站中占比接近57%
  - 核心实现：非阻塞socket + 高效I/O复用机制（如epoll）

加速器的并发卸载模式

- 核心思想：允许在同一进程/线程中**并发卸载**计算任务，减轻甚至消除阻塞性能损失。

异步并发

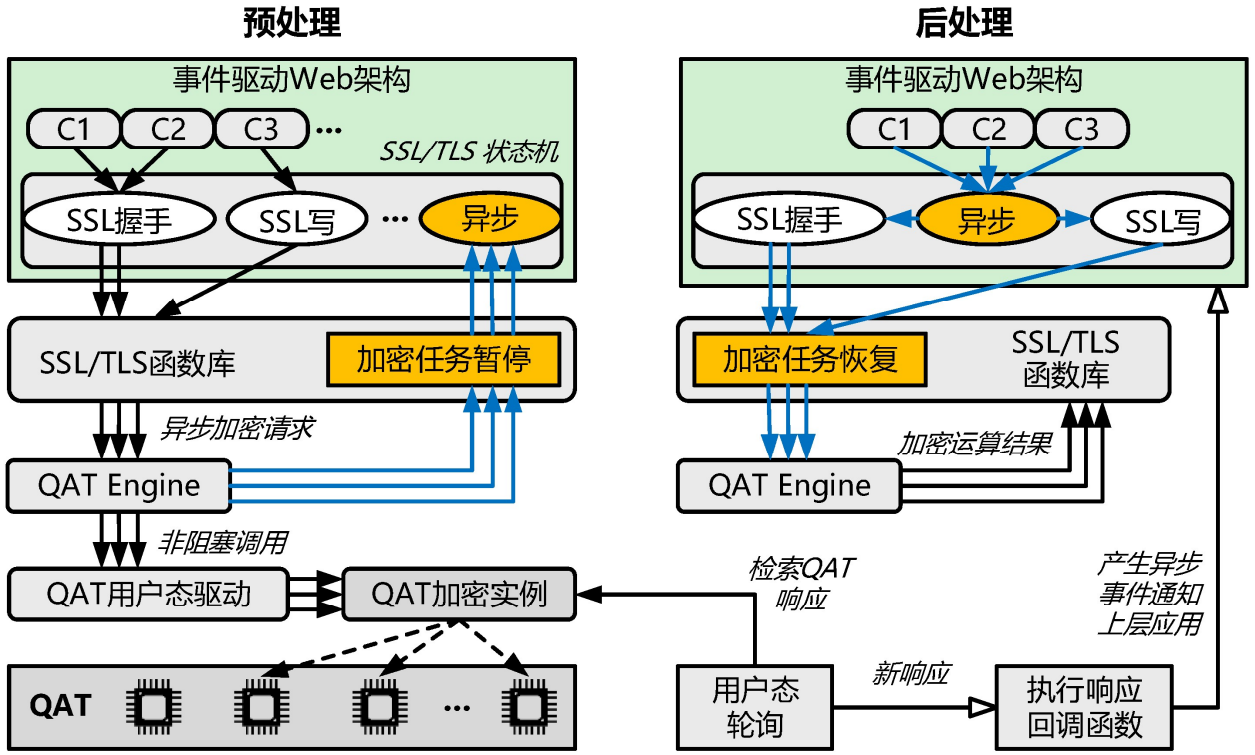
- 优：高性能，完全消除阻塞
- 劣：Web处理流程的较大改动



同步并发

- 性能和改动的平衡

引入异步卸载到SSL/TLS



- SSL软件栈**每一层**引入异步加密支持，将卸载I/O分为预处理和后处理
- SSL函数库层引入加密任务**暂停**和**恢复**（基于用户态纤程实现）
- 应用层SSL状态机中引入一个新的状态称为“异步”

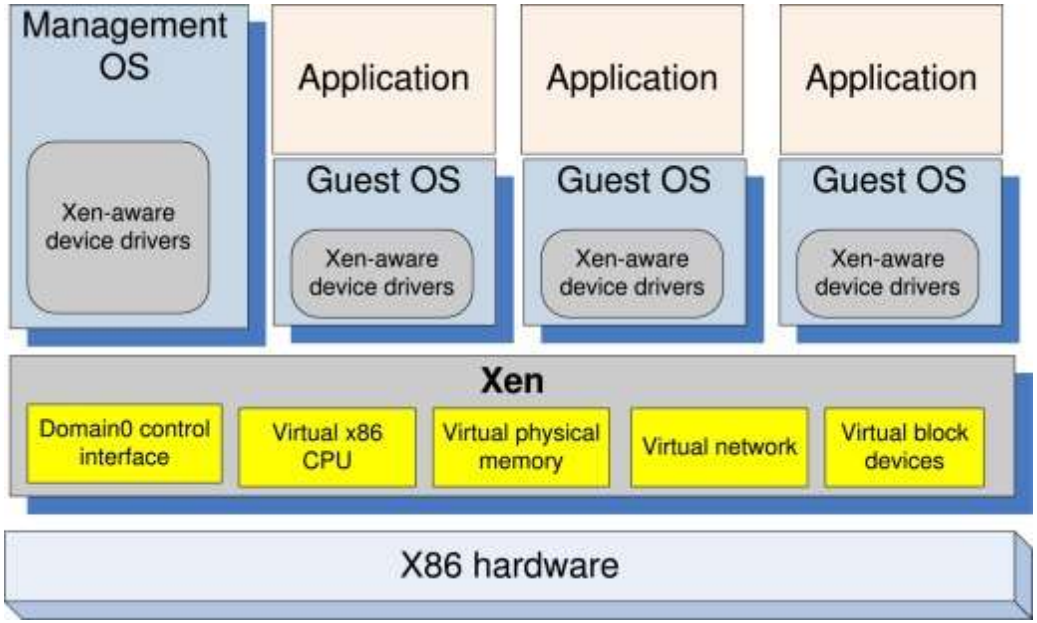
SSL/TLS异步卸载优化

- 性能优化：启发式轮询机制
  - 随着异步加密请求的累积，需要一种高效轮询机制去适时检索QAT响应
  - 轮询线程策略：为每个应用进程配备一个线程去定时轮询
    - 缺点一：可能引发应用进程和轮询线程之间的频繁上下文切换
    - 缺点二：难以确定一个合适的轮询时间间隔
  - 启发式轮询设计
    - 观察： 上层应用是加密操作的产生者，具备足够的知识来确定何时检索QAT
    - 启发式轮询集成至应用内部：避免切换；利用应用层知识指导轮询行为
    - **效率**考量： 高卸载流量时，将足够多的QAT响应聚合到一次轮询，并定义“处理中加密请求数量”作为指标，达到一定阈值再触发轮询
    - **及时性**考量：并发连接较少时，可能无法触发上述阈值，则需引入及时轮询以减少延迟，判断条件是“所有活跃连接均已提交加密卸载请求”

4. GPU虚拟化

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

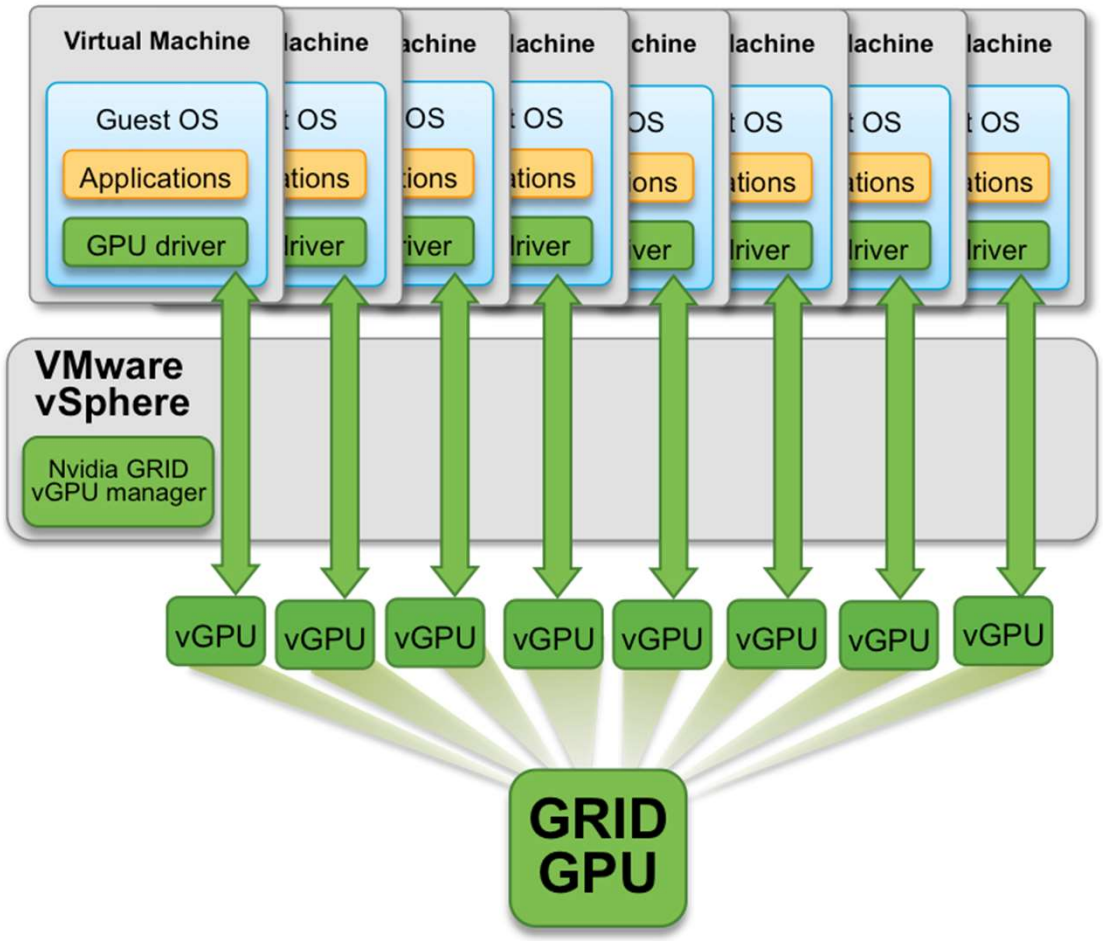
软件模拟



1. 做GPU的任务无需真正的硬件
2. 通过xen-aware device 驱动来完成模拟
3. 降低成本，提高CPU使用范围

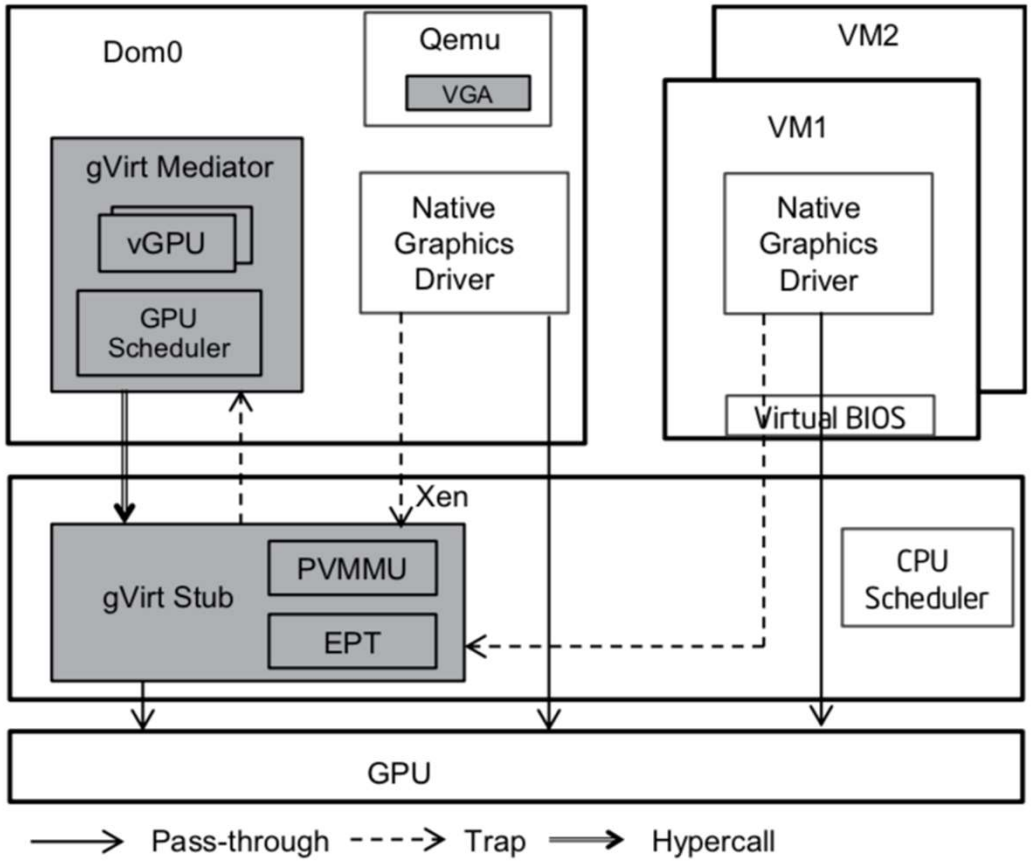
- 软件模拟，无法真正继承GPU的优点
- 指令集更新， xen 驱动无法自适应
- 能耗高，内存高，CPU无法适应
- 性能低，3D模拟无法达到应用的QoS

设备直通



- 允许一个虚拟机直接独占一个GPU硬件
- 直接可以享受硬件加速
- 无法提高硬件利用率
- 安全隐患

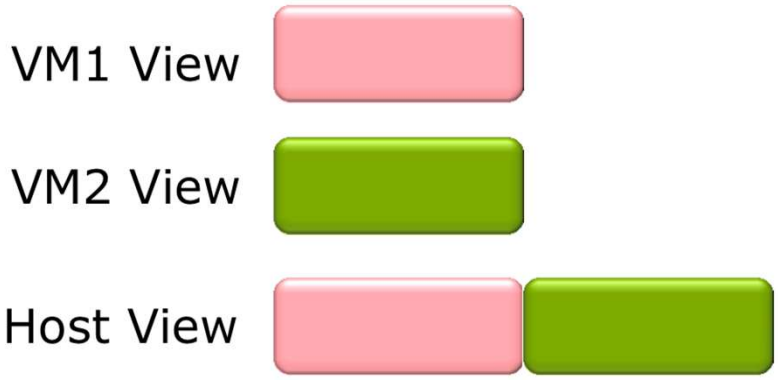
直通+共享



gVirt (ATC14)

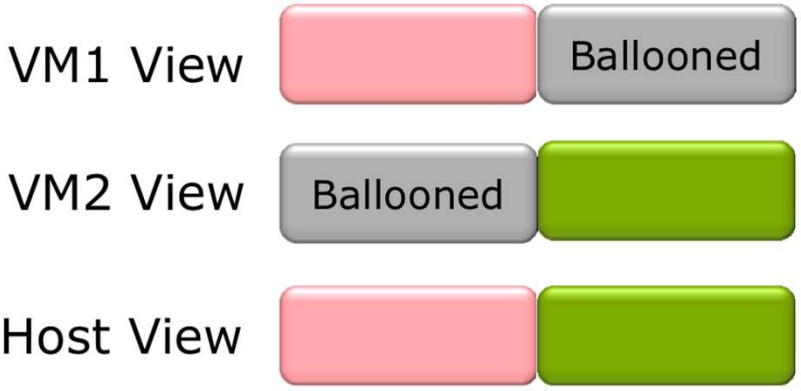
- 设置了一个gvirt stub，将所有的GPU请求送入 gvirt stub
- 通过pvmmu和ept，对一些指令和数据进行筛选，保证安全。
- gvirt mediator管理vGPU之间的切换
- GPU scheduler负责调度。

直通+共享（设备内存）



(虚拟机的角度)

- 少地址空间
- 每个地址都从0开始



(虚拟机的角度)

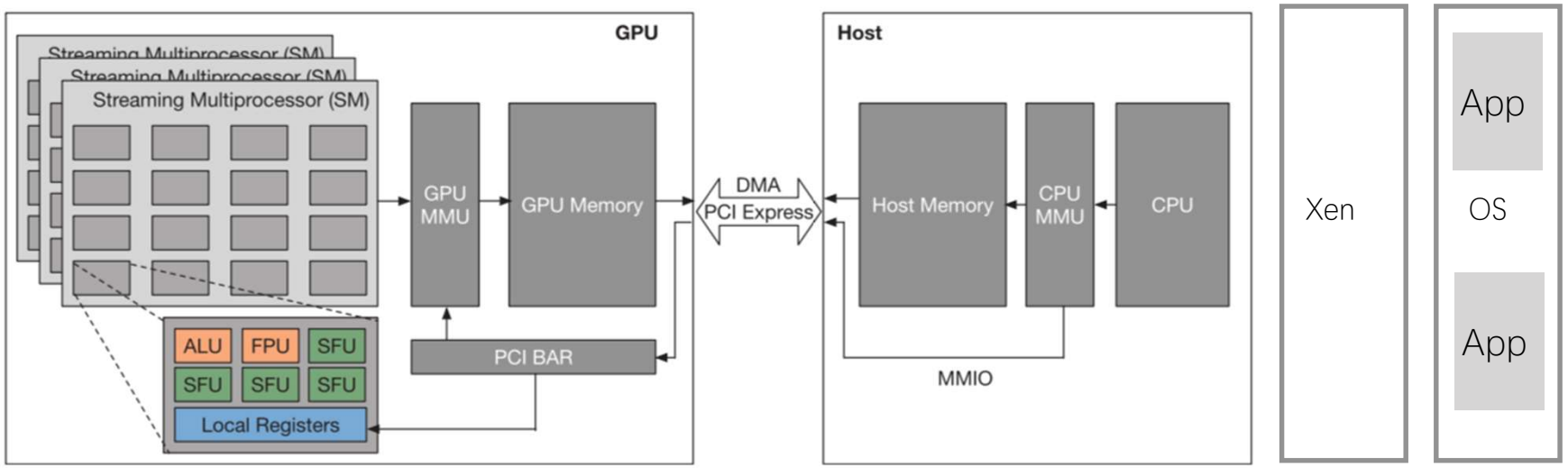
- 正常地址空间
- 每个地址都从原始位开始

存在的问题

- 上述的方法除了直通共享，其余的已经废弃。
- 硬件需要跟系统有接触。
- 现存直通共享技术需要对GPU内存有访问和修改权限
- 上述GPU虚拟化秉持一个核心原则：扩大GPU能力。
- 上述GPU虚拟化前提条件：GPU相较于CPU短缺。

GPU虚拟化发展（1）

- CPU & GPU （独显）



1. 用户只和CPU进行交流
2. 通过CPU编译将指令通过pcie和dma传输给GPU

GPU虚拟化发展（2）

- API **转发** (Thin-Client sosp15)

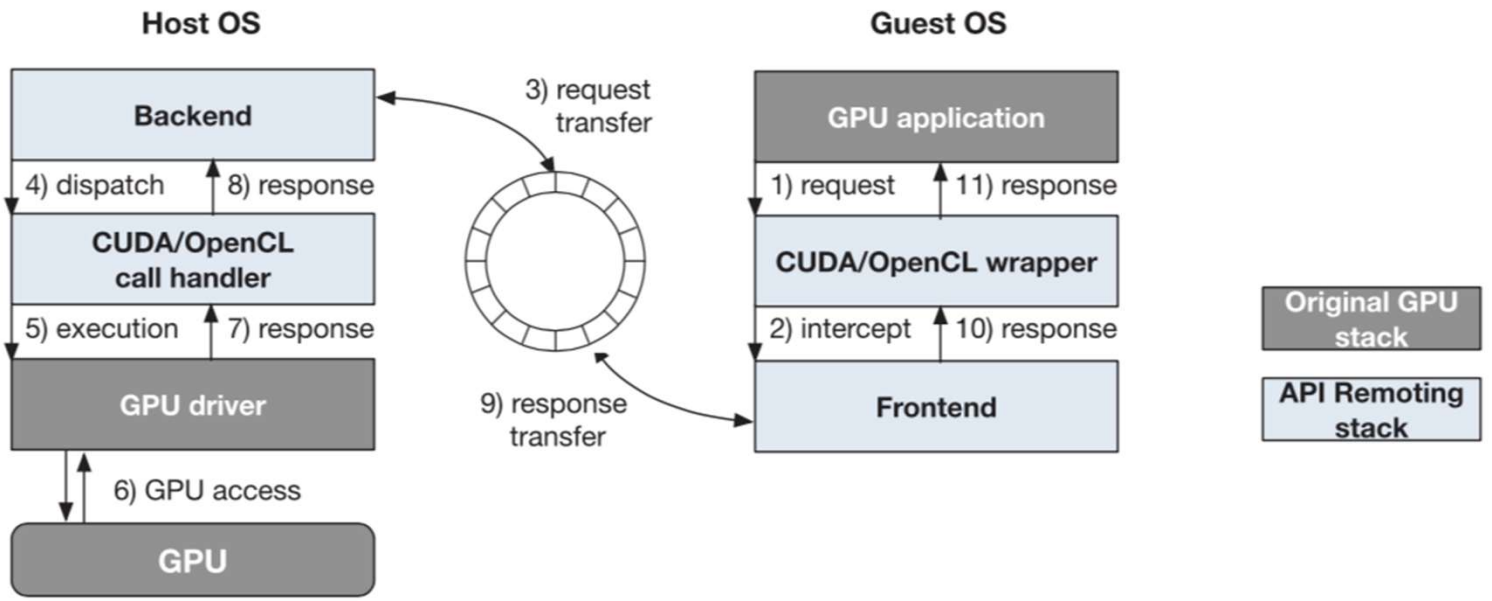
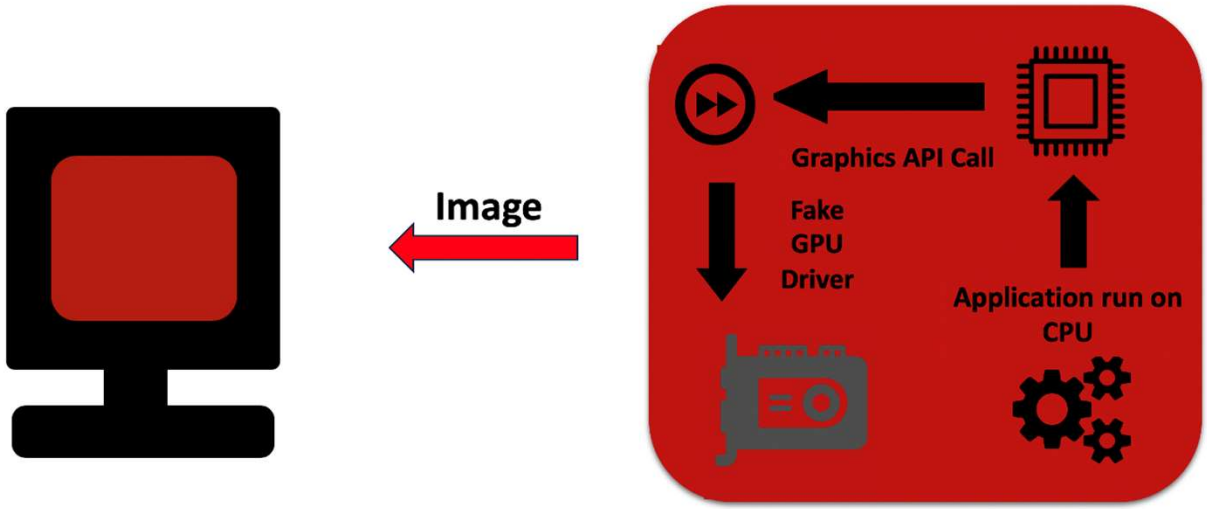


Fig. 2. Architecture of the API remoting approach.

GPU虚拟化发展（3）

- 云游戏 （瘦客户端）



Xbox Play Anywhere. https://www.xbox.com/xbox-play-anywhere

NVIDIA Geforce Now. https://www.nvidia.com/en-us/geforce-now/

GPU虚拟化发展（4）

- 远程API 转发 （远程虚拟化）

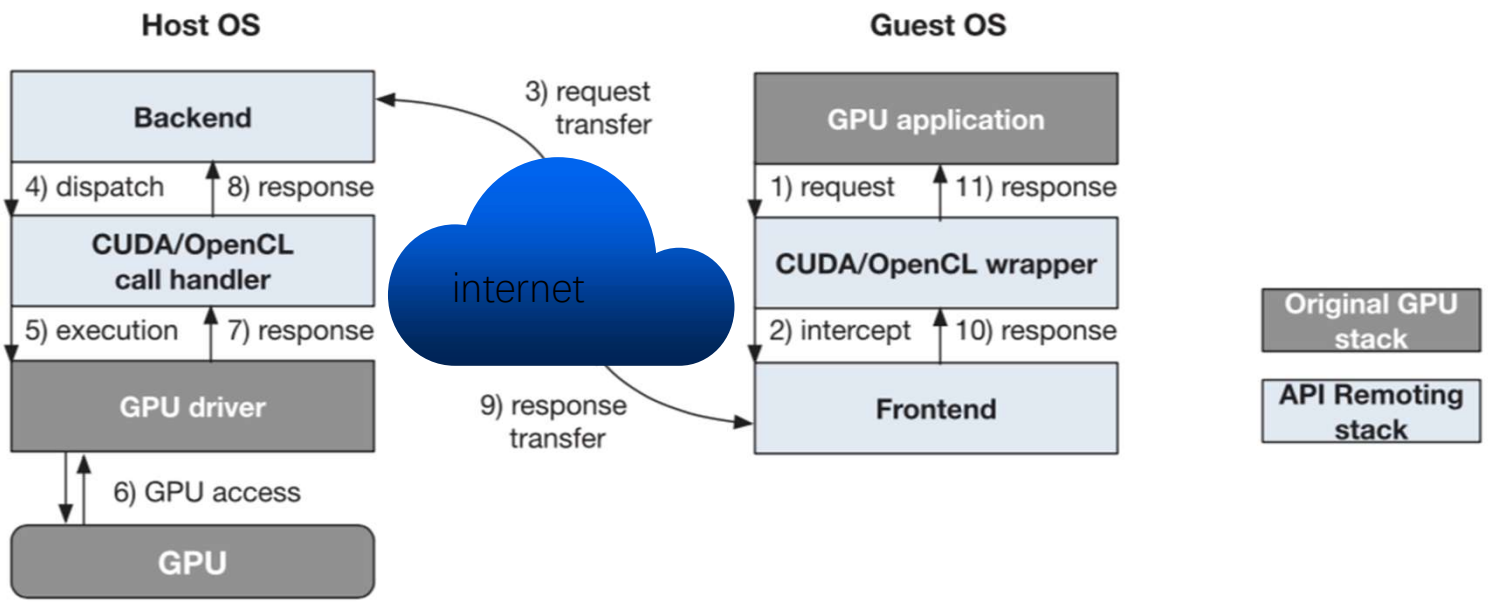
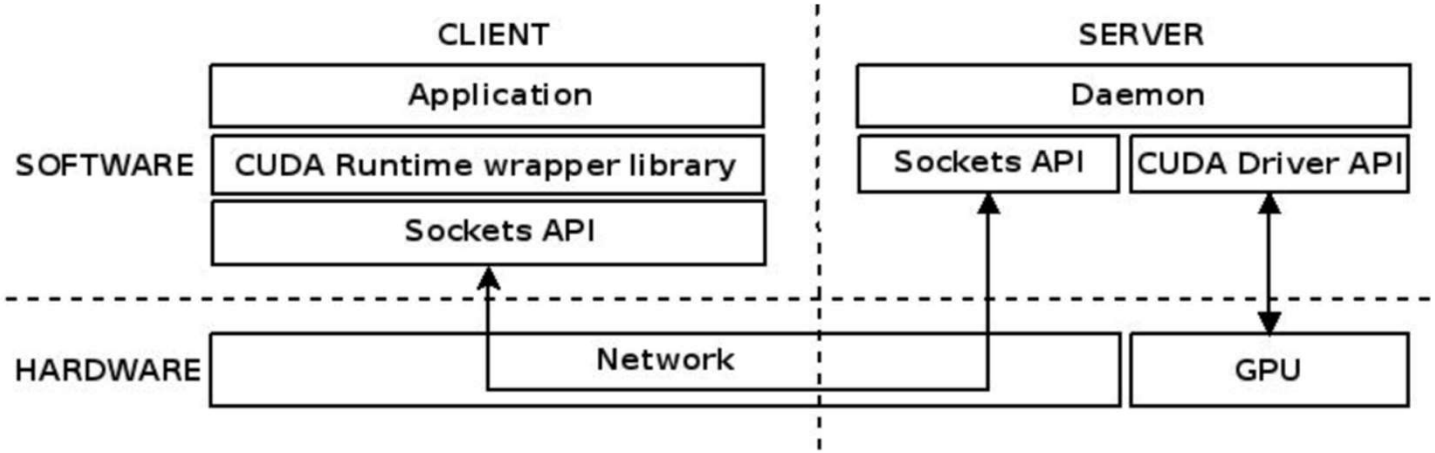


Fig. 2. Architecture of the API remoting approach.

GPU虚拟化发展（5）

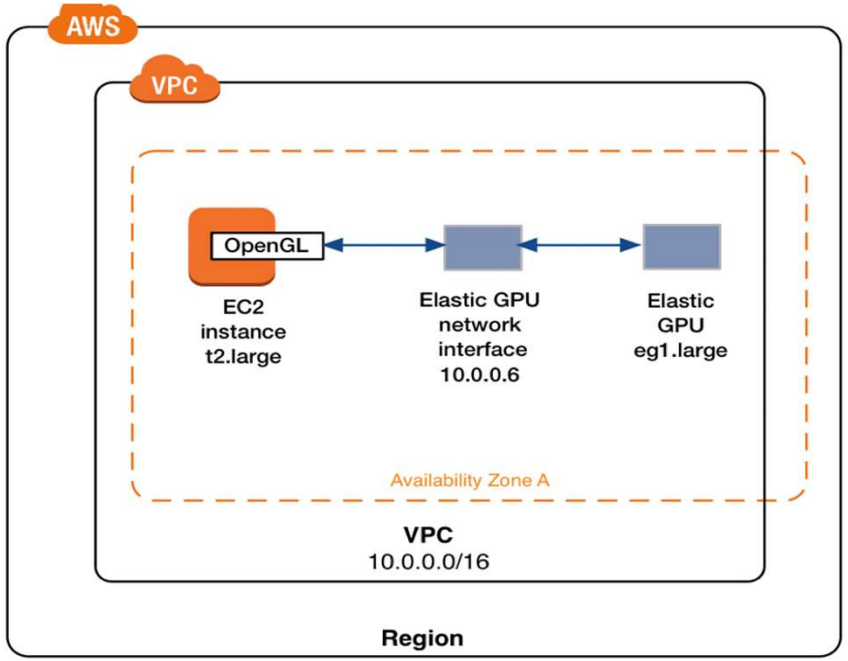
- **远程虚拟化** （rCUDA hpcs10）



1. 通过wrapper library 截获相关GPU指令
2. 通过sockets 将其送往远端GPU
3. 享受到远端硬件加速
4. 将结果通过socket传回

GPU虚拟化发展（5）

- **远程虚拟化（**Amazon Elastic GPU**）**



1. 扩大应用范围，不再局限于本地机器，远端机器也可以享用硬件加速（rCUDA）
2. 相较于瘦客户端， 远程虚拟化可以零成本提高硬件配置 （gremote）
3. 提高云内的服务器利用率。 （microsoft xcloud）

GPU虚拟化发展（6）

- **解决CPU 短缺的方案**

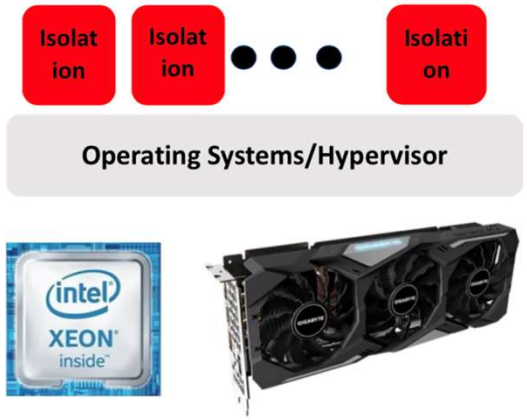
- *gRemote (HPDC'20) expands CPU resources from only server-side to both sides*
- *Sharerender (MM'16) replace VM with container*
- *Pliant (HPCA'19) lowers data precision to save CPU resources*

GPU虚拟化挑战

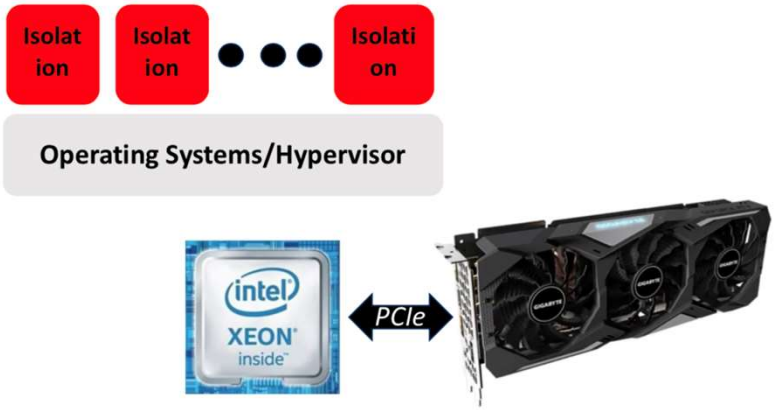
- GPU资源共享与隔离
- 远程虚拟化网络带宽问题
- GPU的迁移问题

GPU虚拟化挑战（1）

- 资源共享瓶颈



What the authors imagine



What we see

1. 所有方案间接作用于GPU，有overhead.
2. 功能有限。
3. 增加CPU的负担

GPU虚拟化挑战（2）

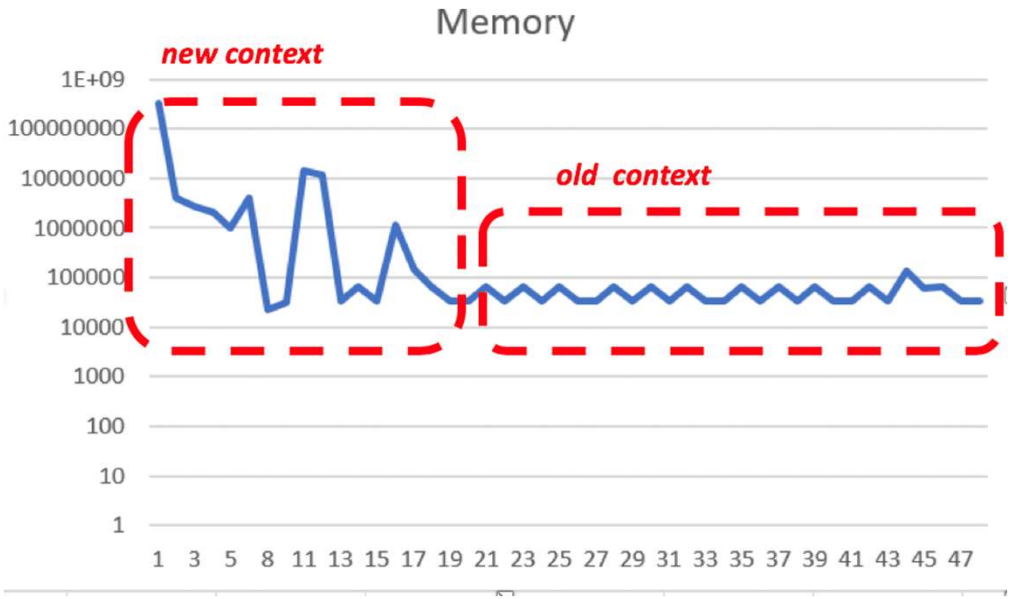
- 网络带宽消耗（motivation）

	FPS	Bandwidth (MiB/s)
blender	56	39.6
dinoshade	55	28.6
scube	60	34.2
spot	61	37.8
glxgears	58	36.5

PS. One command in google earth will bring more than 10 MB data.
glVertexAttribPointer in Uginine Heaven will bring nearly 100 MB data.

GPU虚拟化挑战（2）

- 网络带宽损耗（瓶颈）



对于大应用而言，大数据量往往发生在新的状态的第一帧或前几帧；往后，数据量会接近于平稳，而对于很多方式方法，都是在状态平稳后做文章。

GPU虚拟化挑战（3）

- GPU迁移（motivation）

1. *每个GPU应用对于硬件资源的需求都是动态的。 (DCUDA (SOCC'20), gRemote (HPDC'20))*
2. *系统不在真正作用于一个服务器上，而是一个集群 (rCUDA (tpds'19))*
3. *在这个集群中，我们需要把所有的服务器全部连起来. (Microsoft XCloud)*

- GPU迁移瓶颈

1. 迁移时间过长，所有的技术基本上都在用retrace 和 replay

2. 整个迁移时间是秒级别的，比起cpu迁移的ms级别相差很多

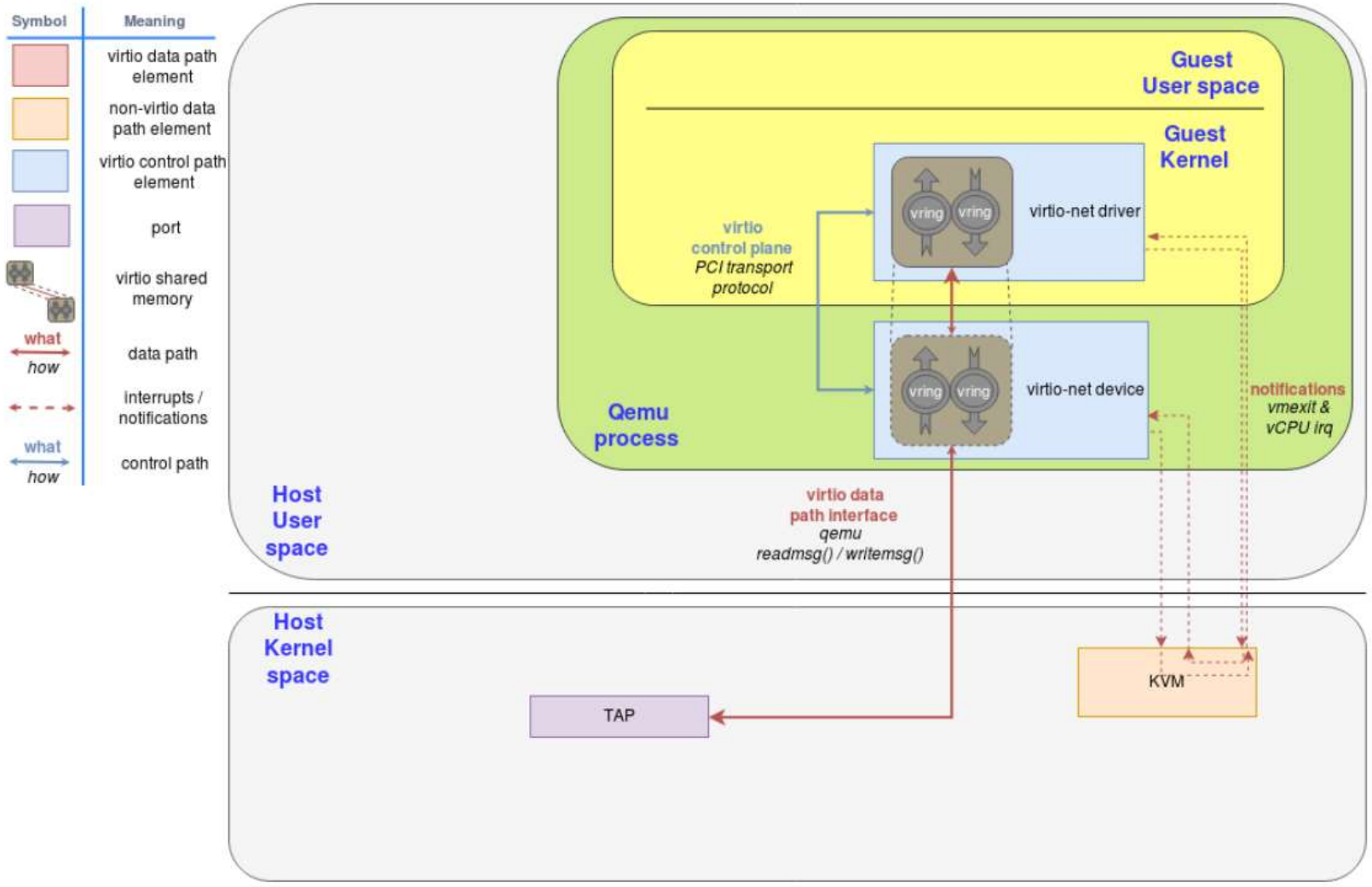
3. 因此，对于GPU的迁移，目前没人说是实时迁移。

Reference

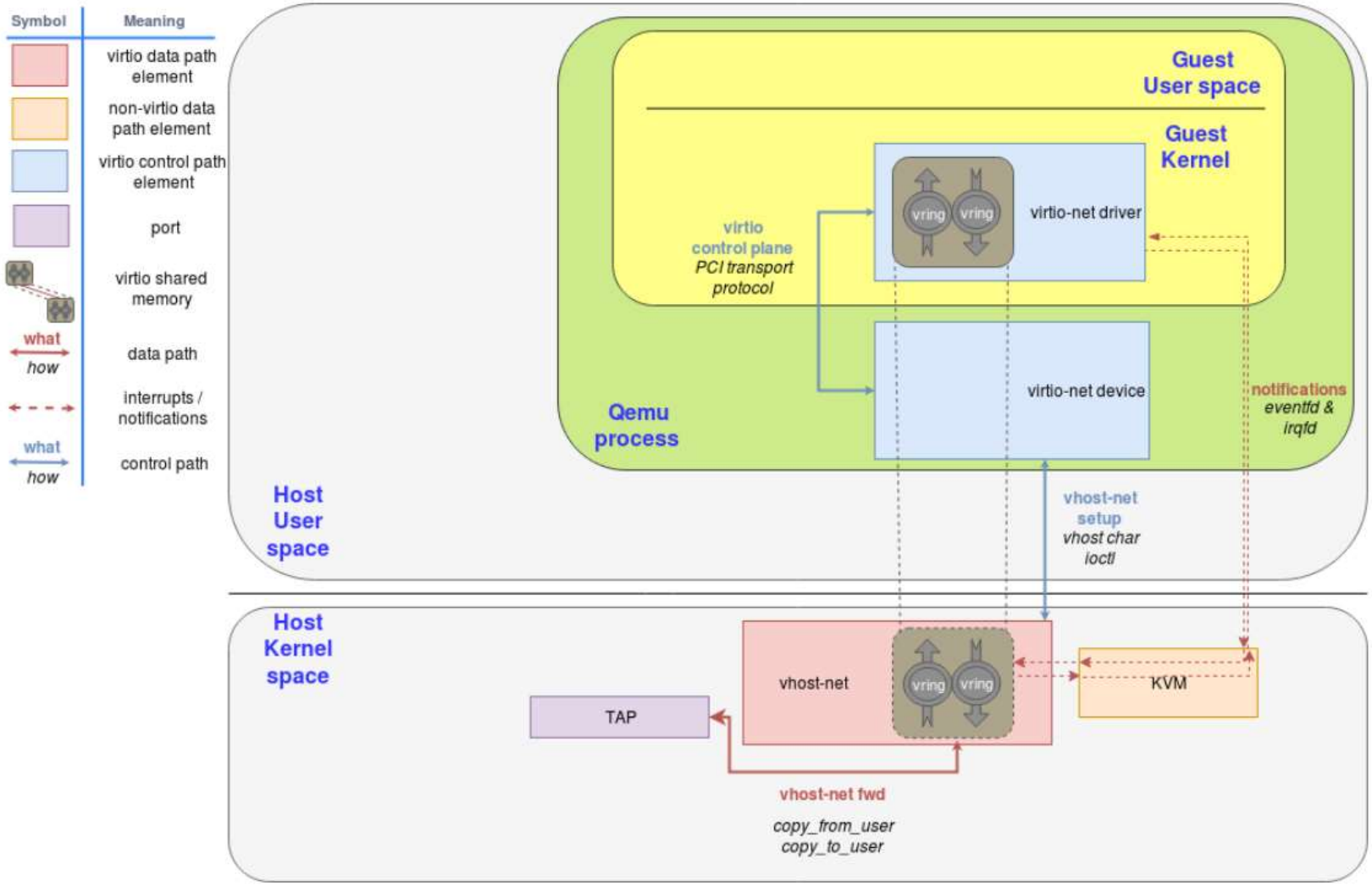
- Bugnion, Edouard, Jason Nieh, and Dan Tsafrir. "Hardware and software support for virtualization." *Synthesis Lectures on Computer Architecture* 12.1 (2017): 1-206.
- Motika, Gal, and Shlomo Weiss. "Virtio network paravirtualization driver: Implementation and performance of a de-facto standard." *Computer Standards & Interfaces* 34.1 (2012): 36-47.
- Xu, Xin, and Bhavesh Davda. "Srvm: Hypervisor support for live migration with passthrough sr-iov network devices." *ACM SIGPLAN Notices* 51.7 (2016): 65-77.
- Dong, Yaozu, et al. "High performance network virtualization with SR-IOV." *Journal of Parallel and Distributed Computing* 72.11 (2012): 1471-1480.

Q&A Thanks!

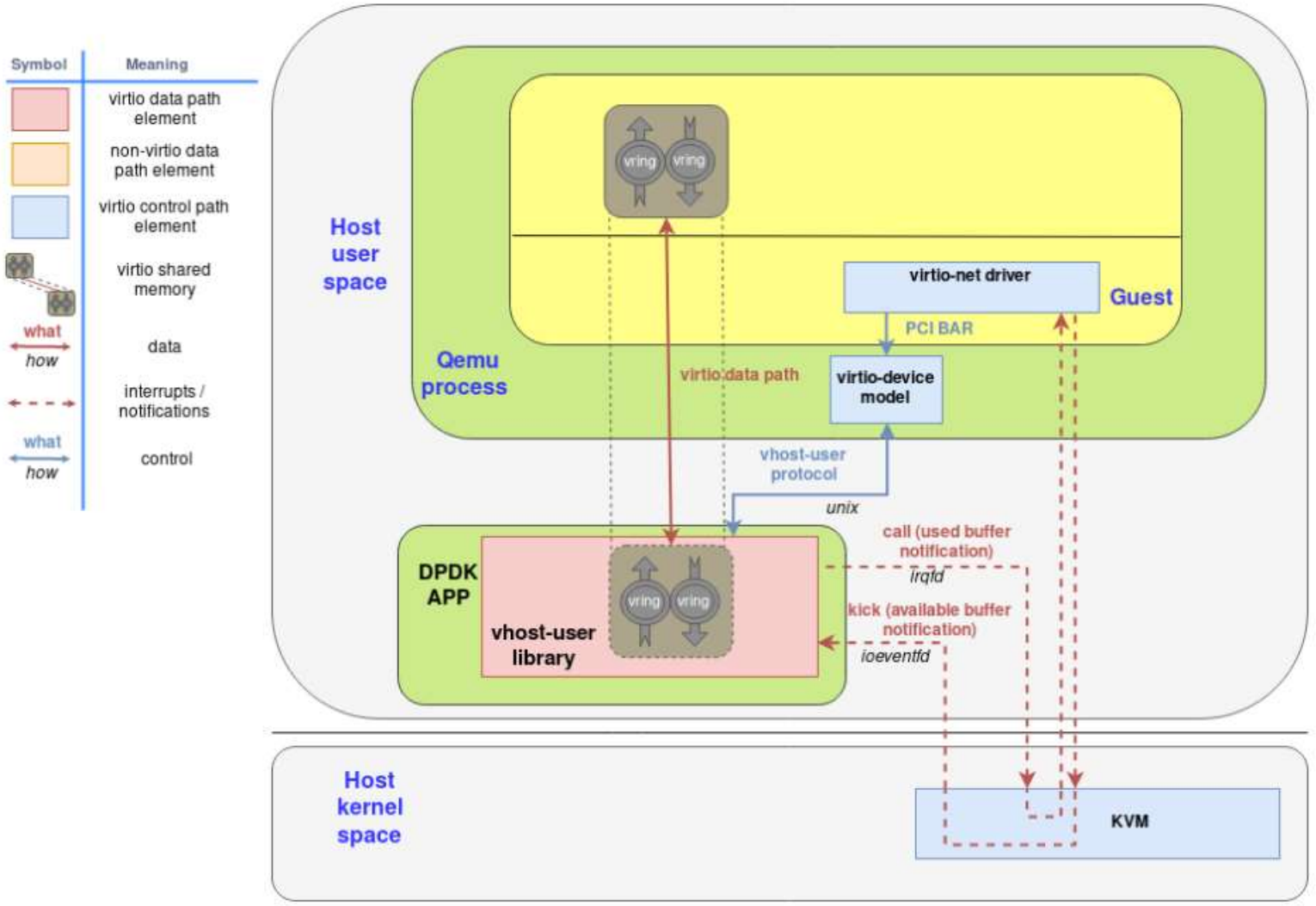
Virtio-net


Symbol
Meaning
virtio data path element
non-virtio data path element
virtio control path element
port
virtio shared memory
what
how
data path
interrupts / notifications
what
how
control path
Guest User space
Guest Kernel
virtio control plane
PCI transport protocol
vring
vring
virtio-net driver
vring
vring
virtio-net device
Qemu process
notifications
vmexit &
vCPU irq
Host User space
virtio data path interface
qemu
readmsg() / writemsg()
Host Kernel space
TAP
KVM

vhost-net


Symbol
Meaning
virtio data path element
non-virtio data path element
virtio control path element
port
virtio shared memory
what
how
data path
interrupts / notifications
what
how
control path
Guest User space
Guest Kernel
vring
vring
virtio-net driver
virtio control plane
PCI transport protocol
virtio-net device
Qemu process
notifications
eventfd & irqfd
Host User space
vhost-net setup
vhost char ioctl
Host Kernel space
vhost-net
vring
vring
KVM
TAP
vhost-net fwd
copy_from_user
copy_to_user

vhost-user


Symbol
Meaning
virtio data path element
non-virtio data path element
virtio control path element
virtio shared memory
what
how
data
interrupts / notifications
what
how
control
Host user space
vring
vring
virtio-net driver
Guest
PCI BAR
Qemu process
virtio data path
virtio-device model
vhost-user protocol
unix
DPDK APP
vhost-user library
vring
vring
call (used buffer notification)
irqfd
kick (available buffer notification)
ioeventfd
Host kernel space
KVM