Virtio: An I/O virtualization framework for Linux

Anish Jain
Subodh Asthana
Suraj Kasi

Fall 2015: October 14th

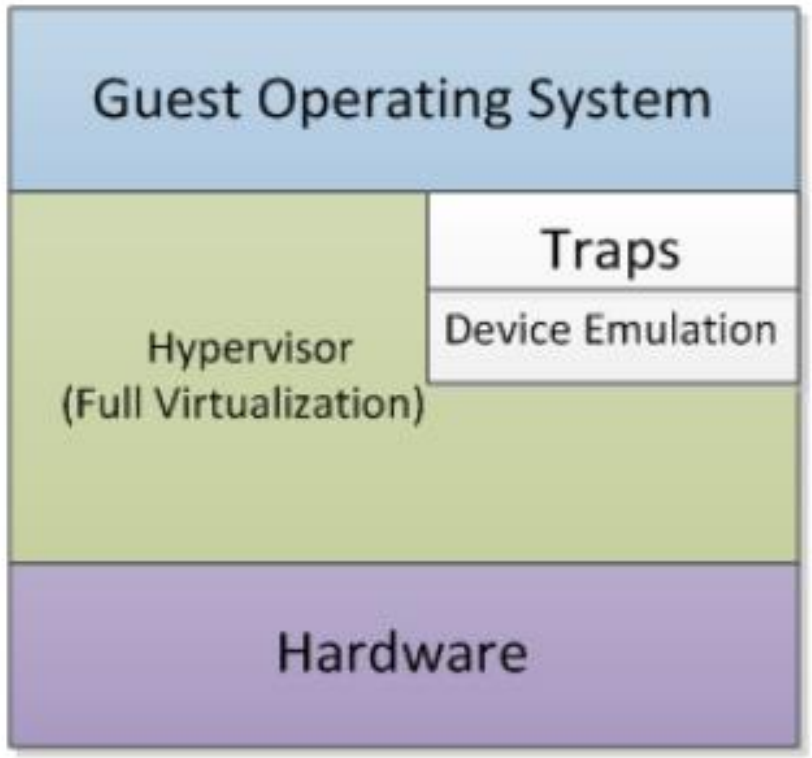
Agenda

- Motivation
- Full Virtualization vs Paravirtualization
- Virtio Architecture
- Virtio Transport Layer
- Vring
- Data Exchange Flow
- Interrupt Handling (speculative)
- Example : VirtIO Block Device Driver

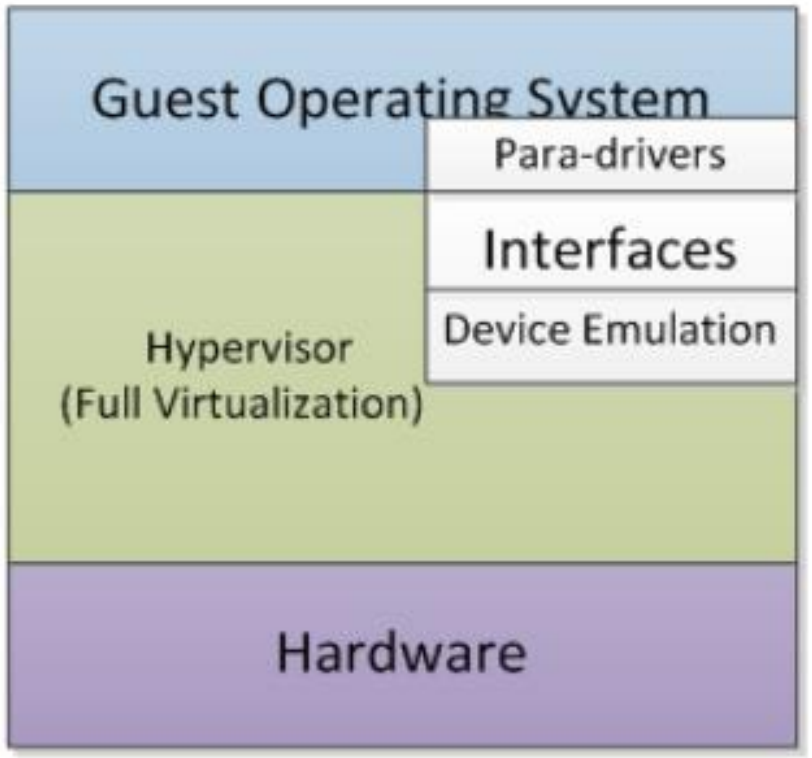
Motivation

- Linux supports 8 distinct virtualization systems:
  - Xen, KVM, VMWare, ...
  - Each of these has its own block, console, network, … drivers
- VirtIO – The three goals
  - Driver unification
  - Uniformity to provide a common ABI for general publication and use of buffers
  - Device probing and configuration

Full Virtualization vs Paravirtualization



Full virtualization



Para-virtualization

Full Virtualization vs Paravirtualization

- **Kernel's device communication with VMware (emulated):**

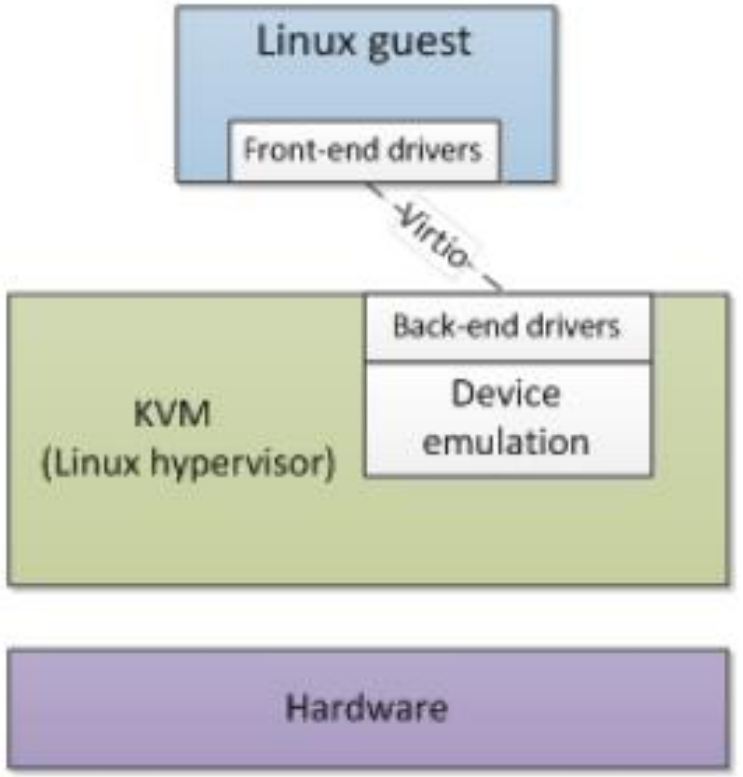
```
void nic_write_buffer(char *buf, int size)
{
    for (; size > 0; size--) {
        nic_poll_ready();           // many traps
        outb(NIC_TX_BUF, *buf++);   // many traps
    }
}
```

- **Kernel's device communication with hypervisor (hypercall):**

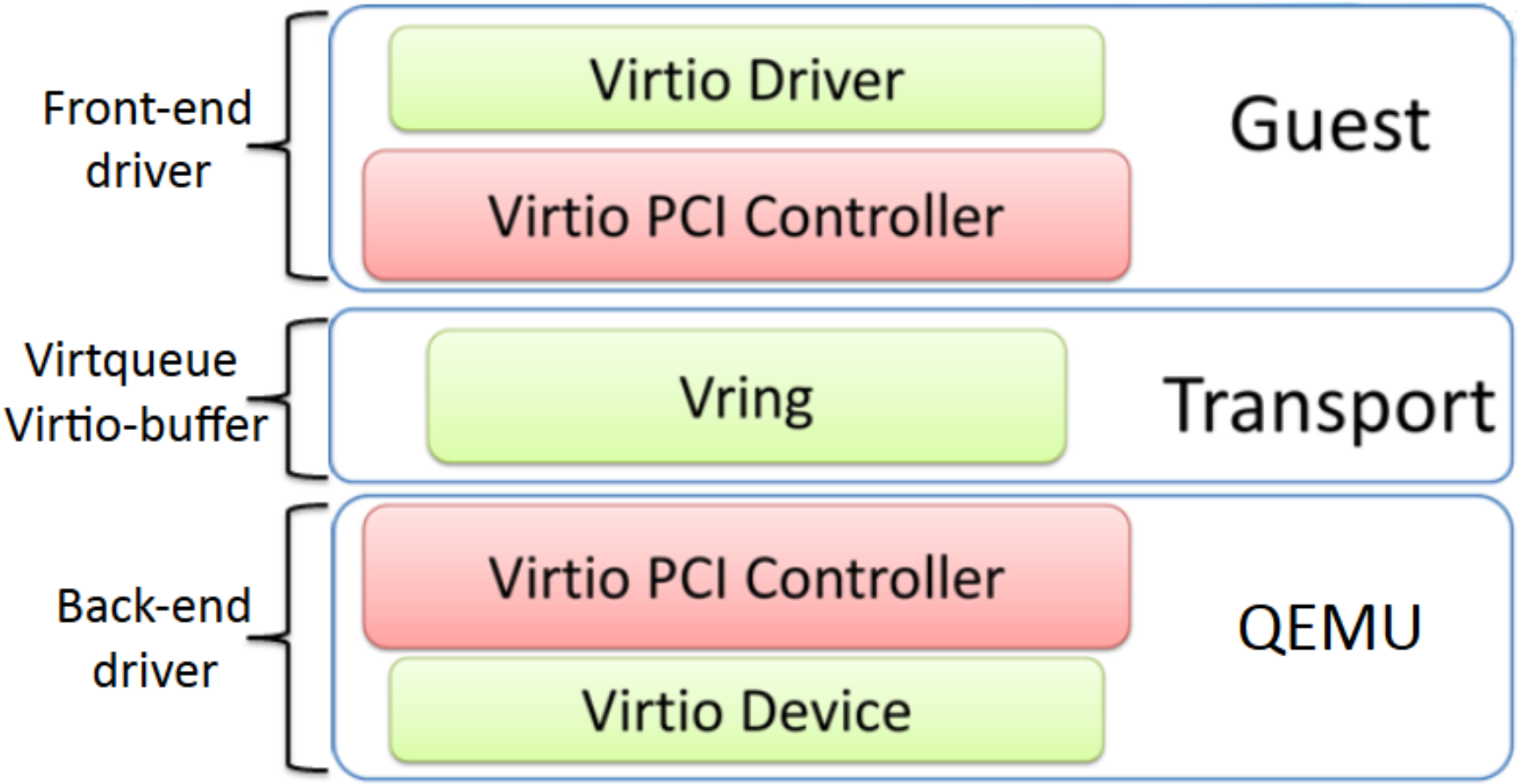
```
void nic_write_buffer(char *buf, int size)
{
    vmm_write(NIC_TX_BUF, buf, size);  // one trap
}
```

VirtIO Architecture

- Front End Driver
  - A kernel module in the guest OS
  - Accepts I/O requests from the user process
  - Transfer I/O requests to back-end driver
- Back-end Driver
  - Accepts I/O requests from front-end driver
  - Perform I/O operation via physical device



VirtIO Architecture



VirtIO Transport layer

- Virtqueue
  - It is a part of the memory of the guest OS
  - A channel between front-end and back-end
  - It is an interface Implemented as Vring
    - Vring is a memory mapped region between QEMU and guest OS
    - Vring is the memory layout of the virtqueue abstraction

```
struct virtqueue_ops {
    int (*add_buf)(struct virtqueue *vq,
                   struct scatterlist sg[],
                   unsigned int out_num,
                   unsigned int in_num,
                   void *data);
    void (*kick)(struct virtqueue *vq);
    void *(*get_buf)(struct virtqueue *vq,
                     unsigned int *len);
    void (*disable_cb)(struct virtqueue *vq);
    bool (*enable_cb)(struct virtqueue *vq);
};
```

Vring

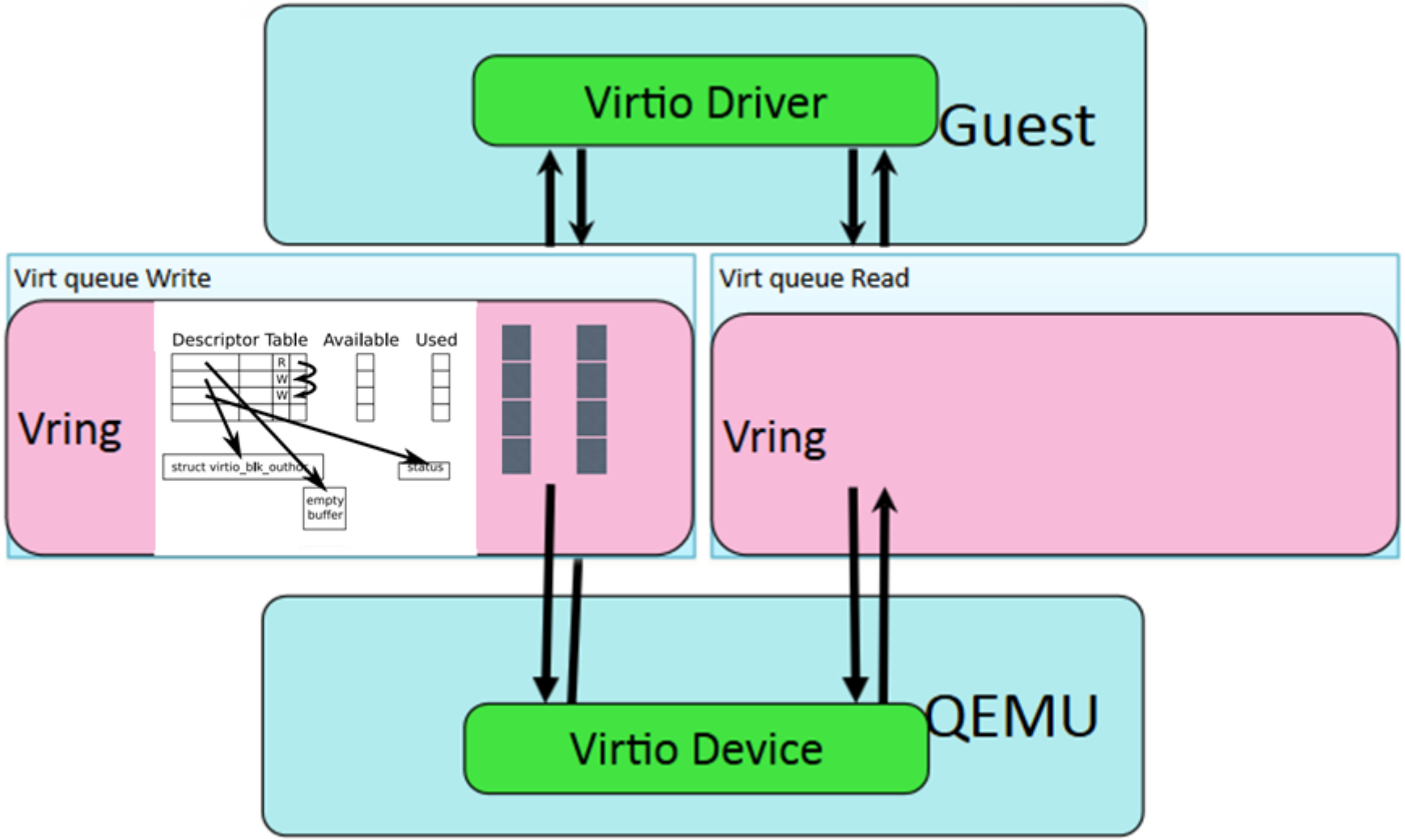
```
struct vring_desc
{
        __u64 addr;
        __u32 len;
        __u16 flags;
        __u16 next;
};
```

Vring Descriptor structure

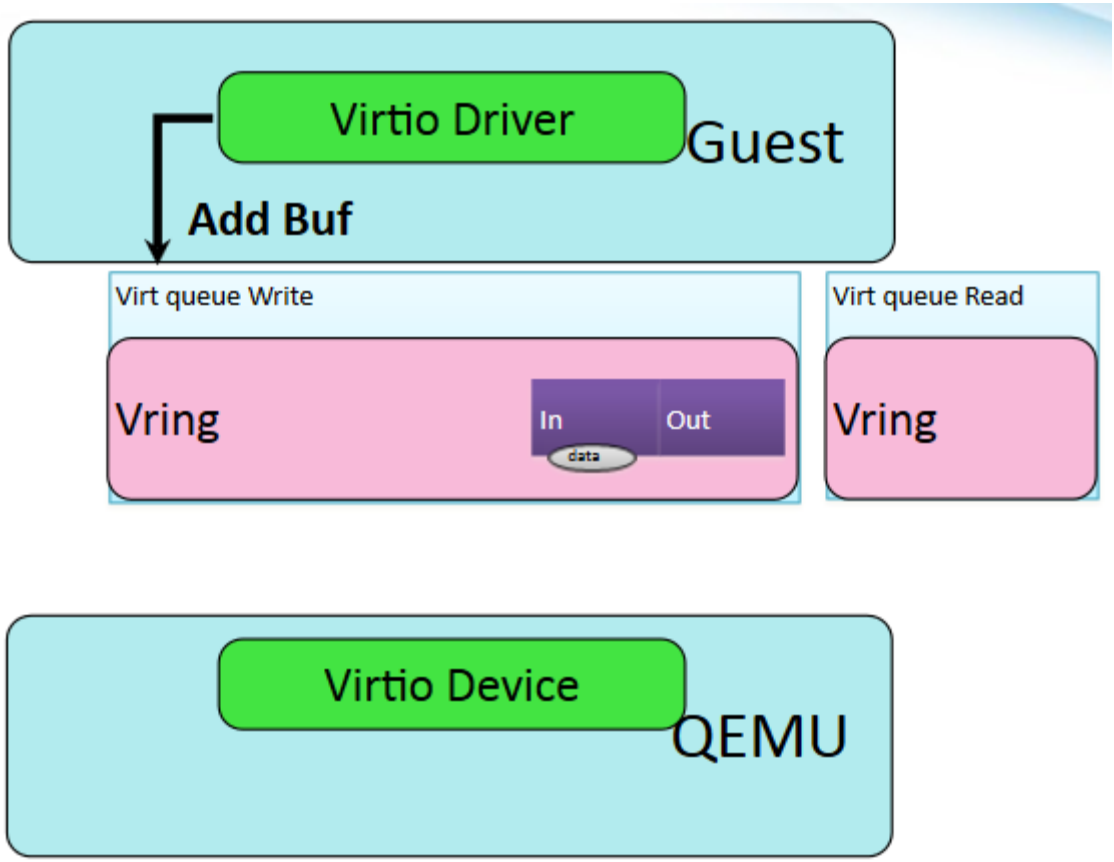
```
struct vring_avail
{
        __u16 flags;
        __u16 idx;
        __u16 ring[NUM];
};
```

Vring available structure

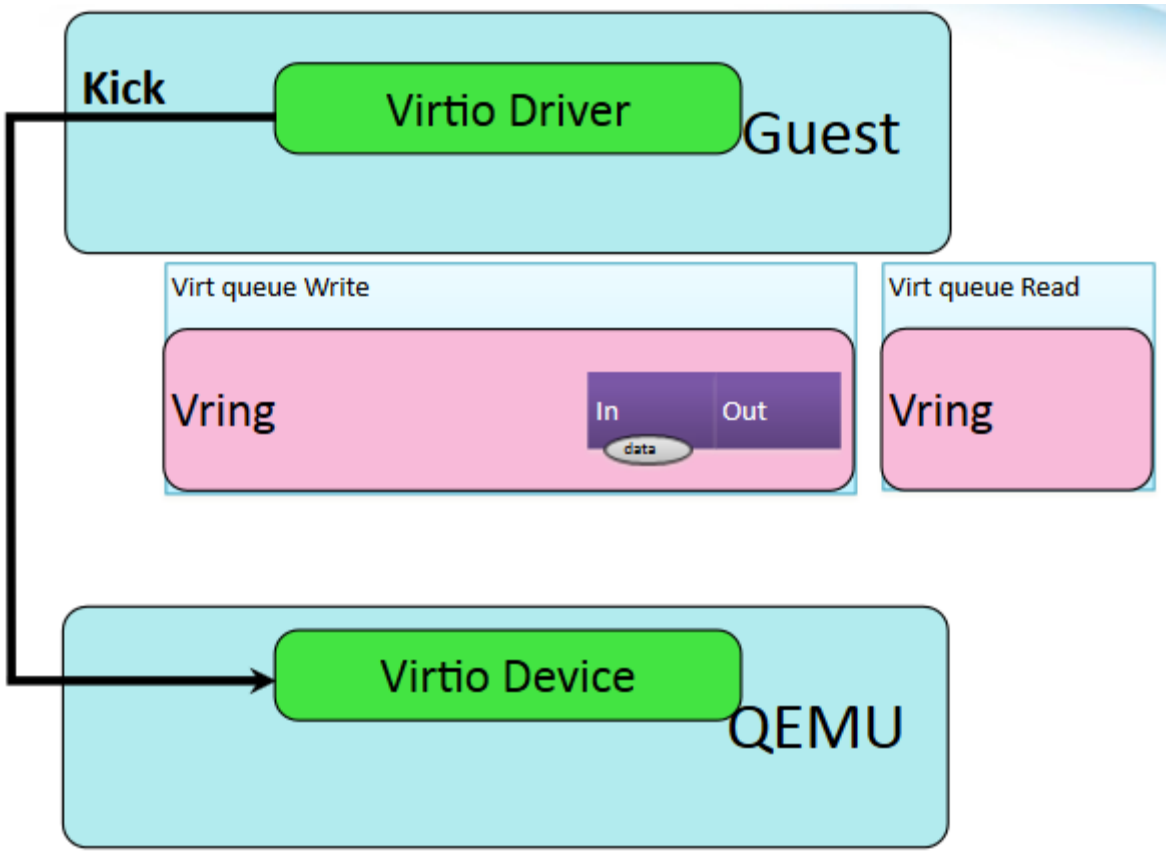
Vring


Virtio Driver
Guest
Virt queue Write
Virt queue Read
Descriptor Table
Available
Used
R
W
W
Vring
Vring
struct virtio_blk_outhdr
status
empty buffer
Virtio Device
QEMU

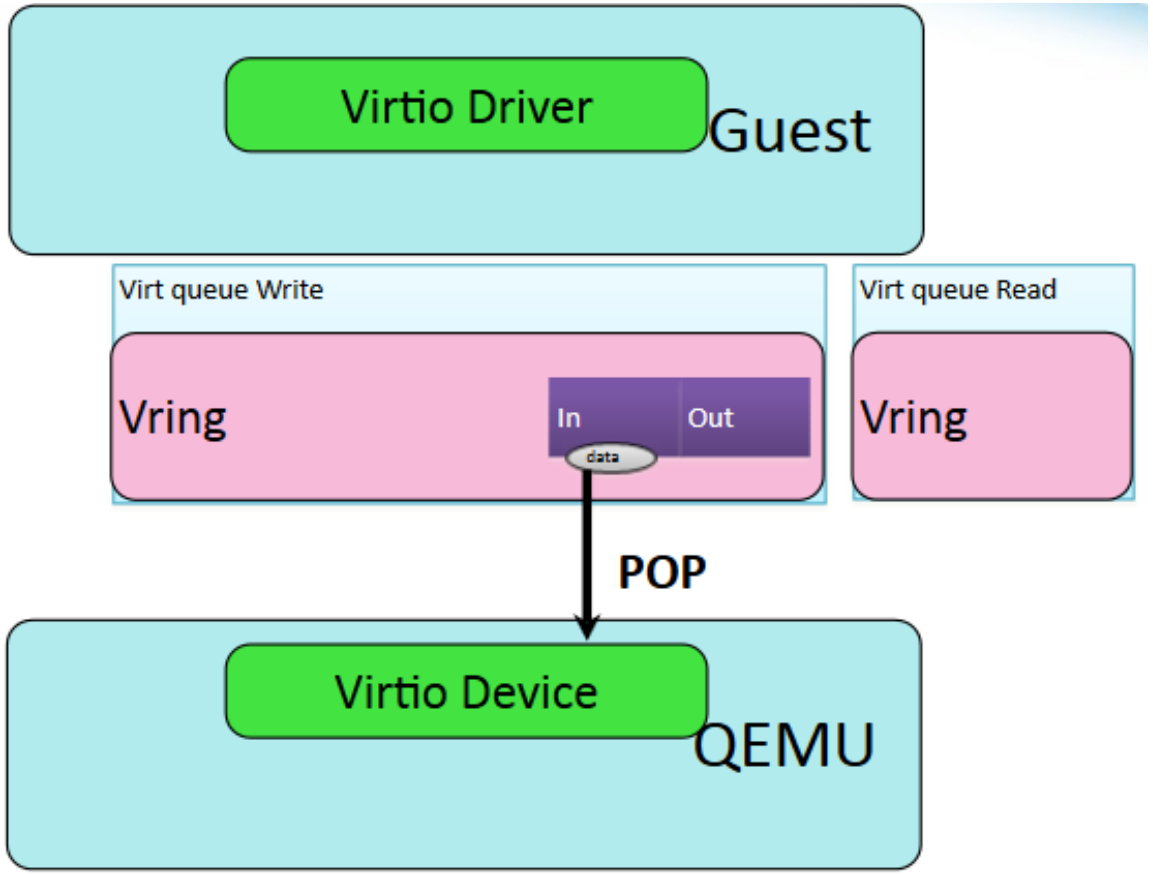
Data Exchange Flow



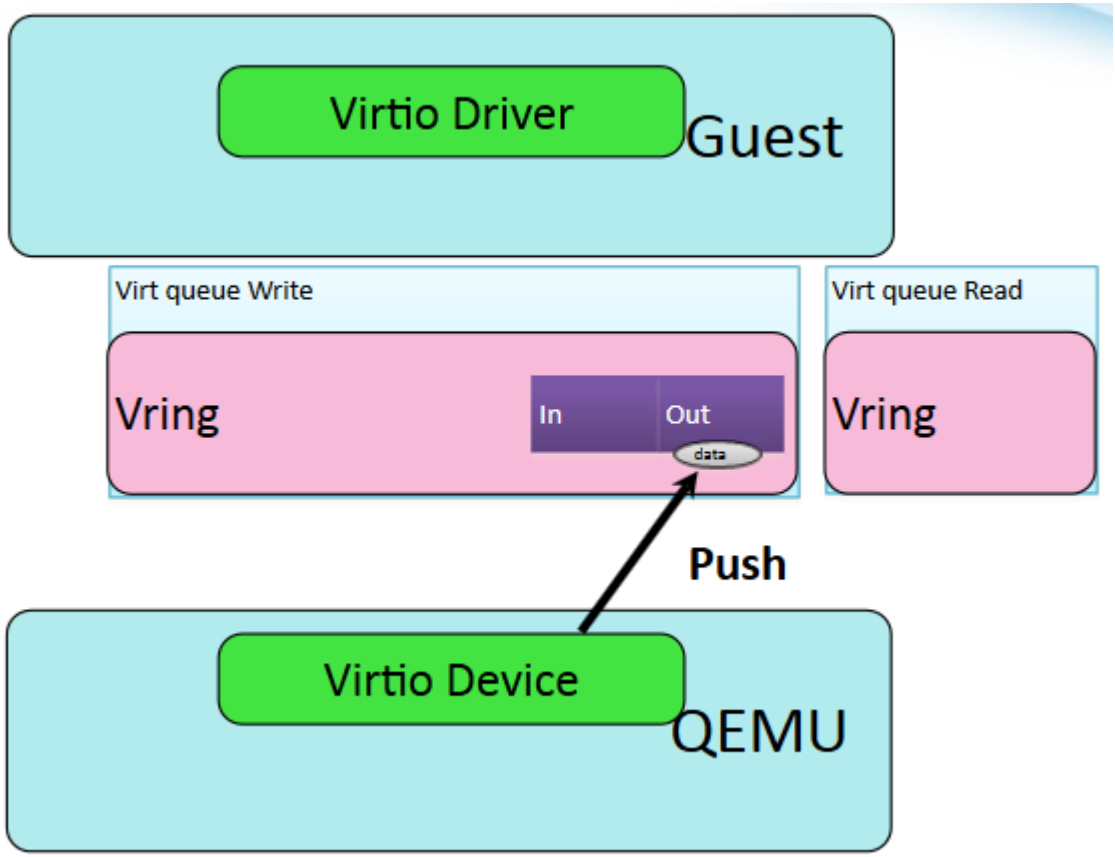
Data Exchange Flow



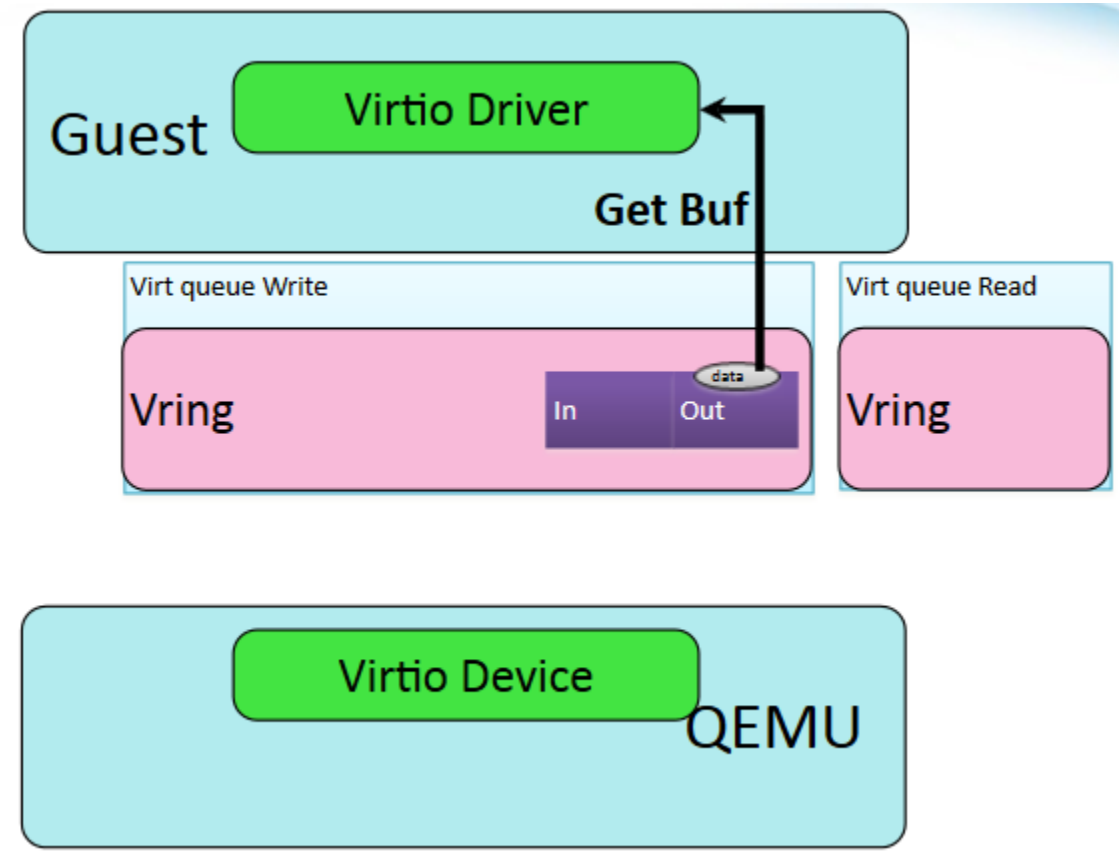
Data Exchange Flow



Data Exchange Flow



Data Exchange Flow

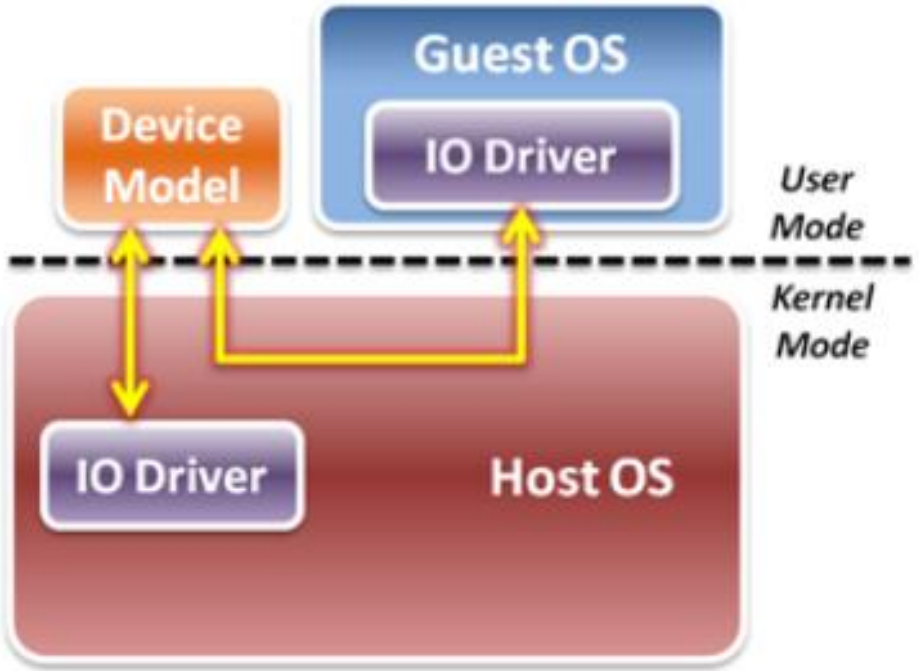


Interrupt Handling

2) There is one QEMU process for each guest system. When multiple guest systems are running, the same number of QEMU processes are running.

Fujitsu's : Kernel Based Virtual Machine Technology white paper

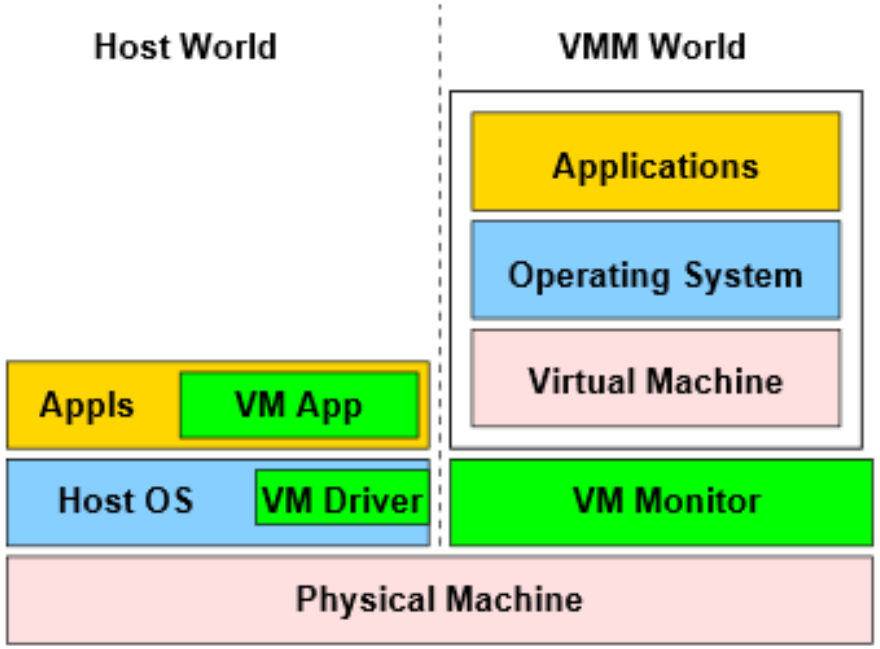
`disable_cb` is a hint that the guest doesn't want to know when a buffer is used: this is the equivalent of disabling a device's interrupt. The driver registers a callback for



Taken from: Rusty Russel's Virtio paper

Interrupt handling (VMWare Workstation)

- Hosted virtual machine model splits the virtualization software between a virtual machine monitor that virtualizes the CPU, an application the uses a host operating system for device support, and an operating system driver for transitioning between them



VirtIO Driver Example – VirtIO Block Driver

```
struct virtio_blk_outhdr
{
    __u32 type;
    __u32 ioprio;
    __u64 sector;
};
```

Figure 1: Header structure of block device

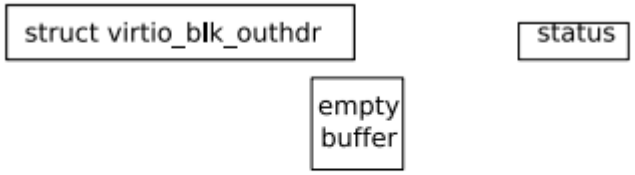


Figure 2: Ingredients for a virtio block read

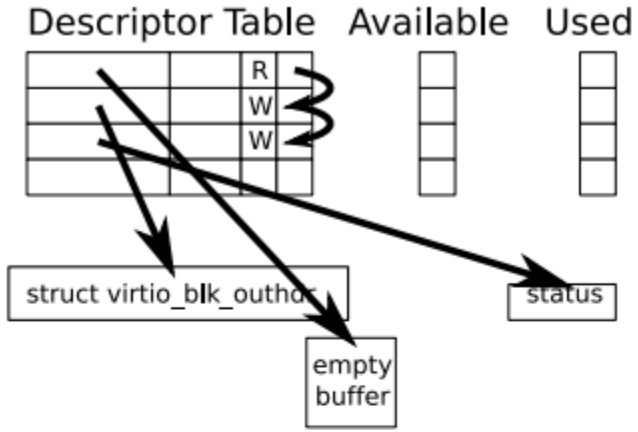


Figure 3: Virtio request placed into descriptor table

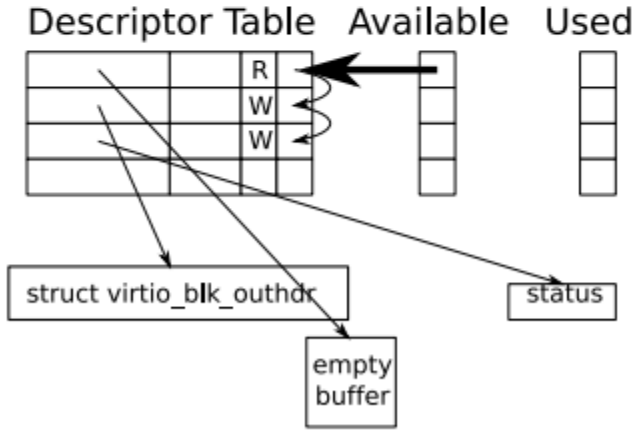


Figure 4: Virtio block read ready to be serviced

References

- ftp://ftp.os3.nl/people/nsijm/INR/Week%201/papers/32_virtio_Russel.pdf
- http://www.ibm.com/developerworks/library/l-virtio/
- http://bit.ly/1ZtUPah
- http://www.linuxinsight.com/files/kvm_whitepaper.pdf
- http://lxr.free-electrons.com/source/include/linux/virtio.h?v=2.6.34
- 15-410 lecture slides
- Fujitsu's Kernel-based Virtual Machine Technology white paper
- Quamranet's Kernel based Virtualization Driver white paper