WISSEN
TECHNIK
LEIDENSCHAFT

Cloud Operating Systems: VirtIO

Clemens Walluschek, Dominik Gollner, Johannes Haring, Thomas Moder

May 10, 2021

Outline

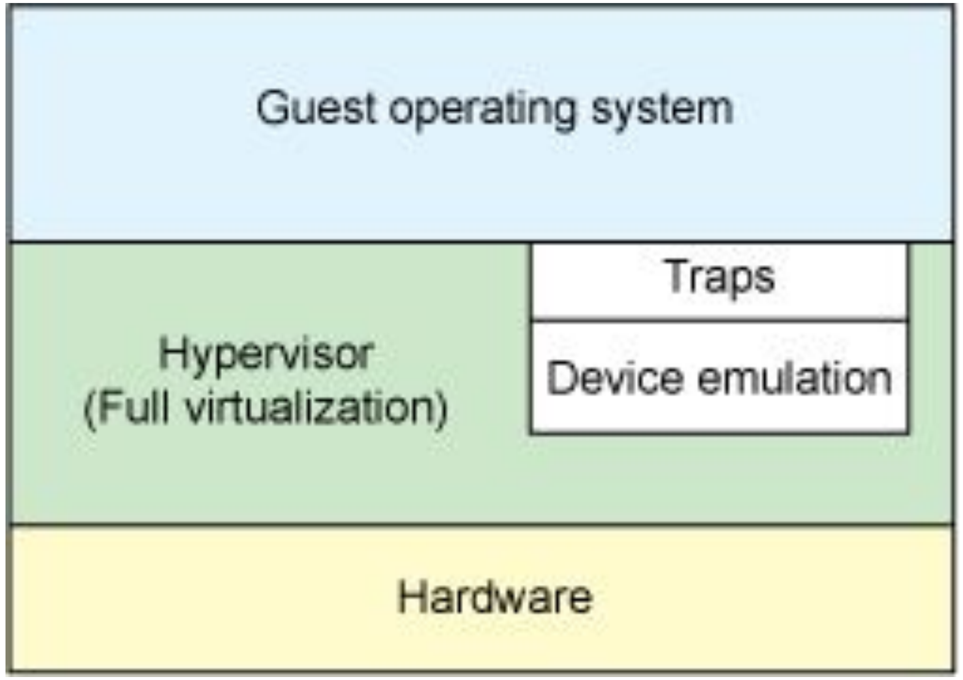
- **Introduction**
- Component Overview
- VirtIO over PCI
- Virtqueues
- Devices and Drivers

What is VirtIO?

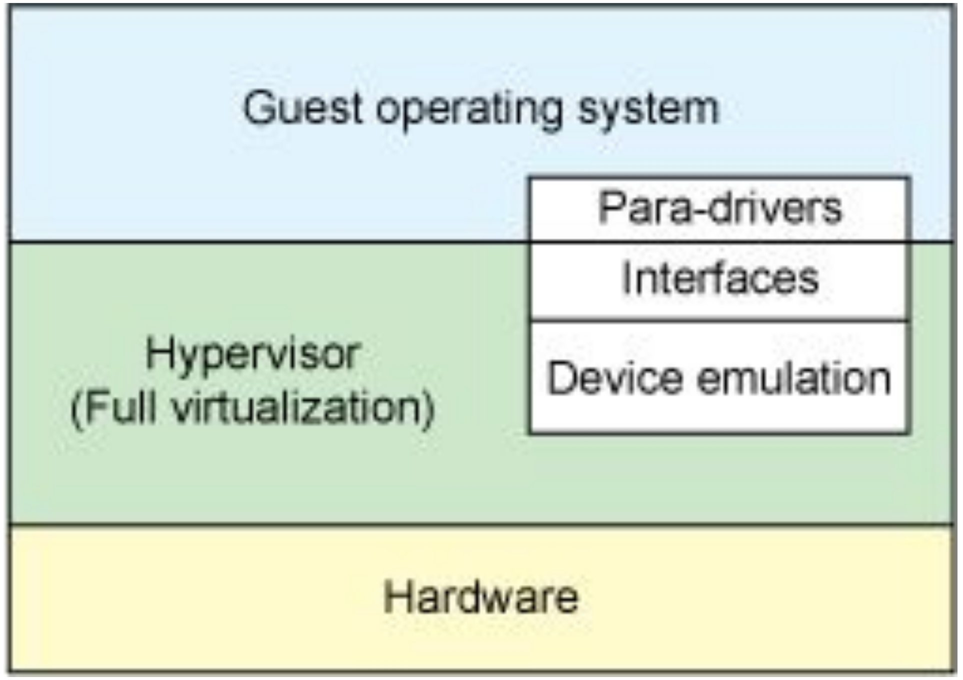
- Standardized interface which allows VMs access to "virtual" devices (network cards, block devices, …)
- Improved performance over "emulated" devices
- Guest:
  - Minimum setup and configuration to send and receive data
- Host:
  - Handles majority of setup of physical hardware

scribble.io
virt.io
imgflip.com

Full- vs Paravirtualization



Full



Para

https://developer.ibm.com/articles/l-virtio/

Difference to PCI Passthrough

- PCI Passthrough
  - Directly connect guest to hardware
  - Needs hardware specific driver
  - Better performance
  - Not managed by the hypervisor

Goals - Why use VirtIO?

VirtIO is designed to be

- **Straightforward**: Normal bus mechanisms of interrupts and DMA
- **Efficient**: Rings of descriptors for input and output (Virtqueues), optimized to avoid accessing cache lines simultaneously
- **Standard**: No environment assumptions, common interface for all VirtIO drivers
- **Extensible**: Feature bits ensure only supported functionality is used

VirtIO Structure

- Hypervisor exposes VirtIO devices to the guest
- Via different transport methods
- Guest discovers devices within VM as normal physical devices
- Driver-allocated memory regions, shared between hypervisor and devices for data communication

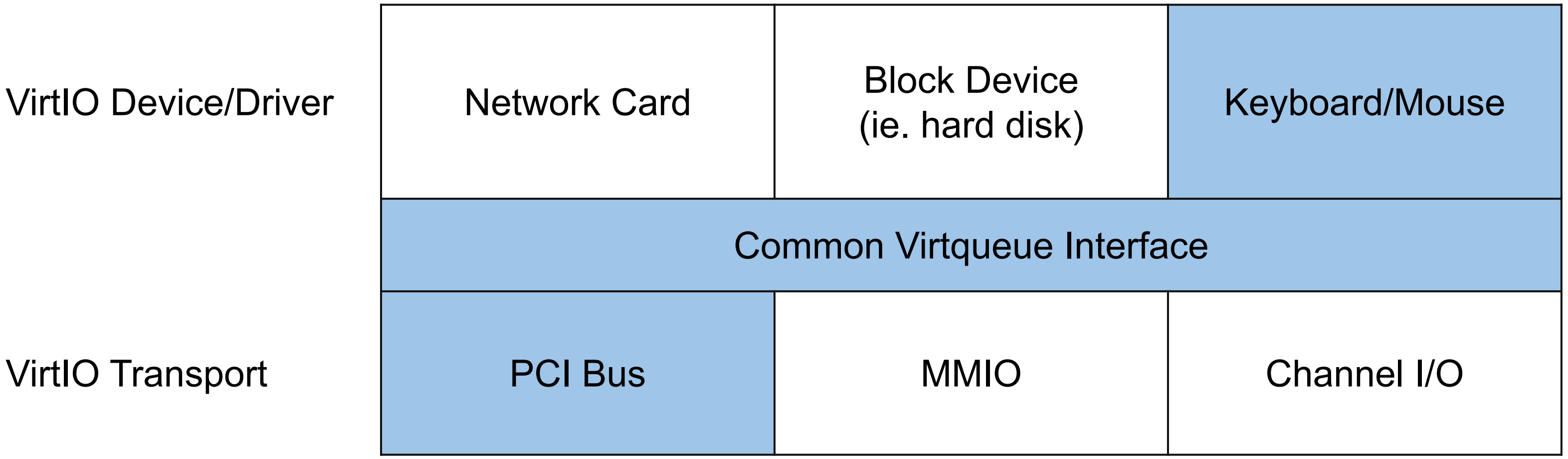
Outline

- Introduction
- **Overview**
- VirtIO over PCI
- Virtqueues
- Devices and Drivers

VirtIO Layers

VirtIO Device/Driver	Network Card	Block Device (ie. hard disk)	Keyboard/Mouse
	Common Virtqueue Interface		
VirtIO Transport	PCI Bus	MMIO	Channel I/O

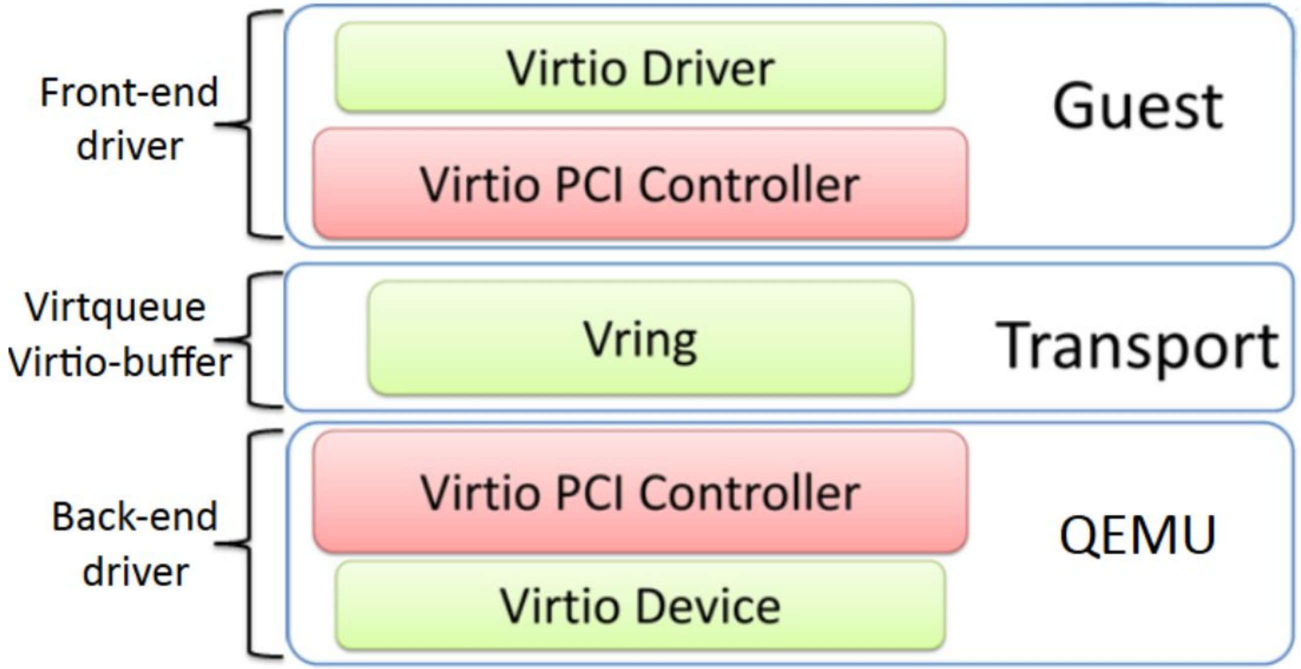
VirtIO Layers



VirtIO Device/Driver	Network Card	Block Device (ie. hard disk)	Keyboard/Mouse
	Common Virtqueue Interface		
VirtIO Transport	PCI Bus	MMIO	Channel I/O

```
# qemu -device virtio-keyboard-pci
```

VirtIO Communication Host/Guest



Front-end driver

Virtio Driver

Virtio PCI Controller

Guest

Virtqueue
Virtio-buffer

Vring

Transport

Back-end driver

Virtio PCI Controller

Virtio Device

QEMU

https://www.cs.cmu.edu/~412/lectures/Virtio_2015-10-14.pdf

VirtIO Sequence

1. Host: Create Transport (PCI, MMIO, Channel I/O) device and provide to Guest
2. Guest: Device Discovery according to transport option (at boot)
   a. Create Virtqueues
   b. Initialize Guest driver
   c. Perform device-specific setup
3. Guest and Host drivers use
   a. Virtqueues for communication
   b. Interrupts to notify about new buffers in the queue

Outline

- Introduction
- Component Overview
- **VirtIO over PCI**
- Virtqueues
- Devices and Drivers

VirtIO Device/Driver	Network Card	Block Device (ie. hard disk)	Keyboard/Mouse
	Common Virtqueue Interface		
VirtIO Transport	**PCI Bus**	MMIO	Channel I/O

Basic Virtio Device

- Discovered and identified by bus specific method
  - PCI Bus
  - MMIO
  - Channel I/O

Device Initialization

1. Reset the device
2. Set ACKNOWLEDGE bit
3. Set DRIVER bit
4. Read/write feature bits
5. Set FEATURE_OK bit
6. Re-read FEATURE_OK bit
7. Device specific setup
8. Set DRIVER_OK bit

Basic Virtio device/driver

- Device status field
- Device feature bits
- Notifications
- Device Configuration space
- One or more virtqueues

Device Status Field

- Provides indication for completed steps
- e.g. ACKNOWLEDGE, FEATURES_OK

Feature Bits

- Virtio offers features it accepts
- Driver reads bits and accepts subset
- Renegotiate only after reset
- Forward/backward compatibility

Notifications: ISR status capability

- Some devices need to notify the guest
  - ie. when a block from a hard disk has been read and is ready for the guest
- Implemented using interrupts

Other Fields

- Device Configuration Space
    - Generally for rarely-changing or initialization-time parameters
- Virtqueues
    - Mechanism for bulk data transport on virtio devices
    - Zero or more per device

VirtIO PCI Device Discovery

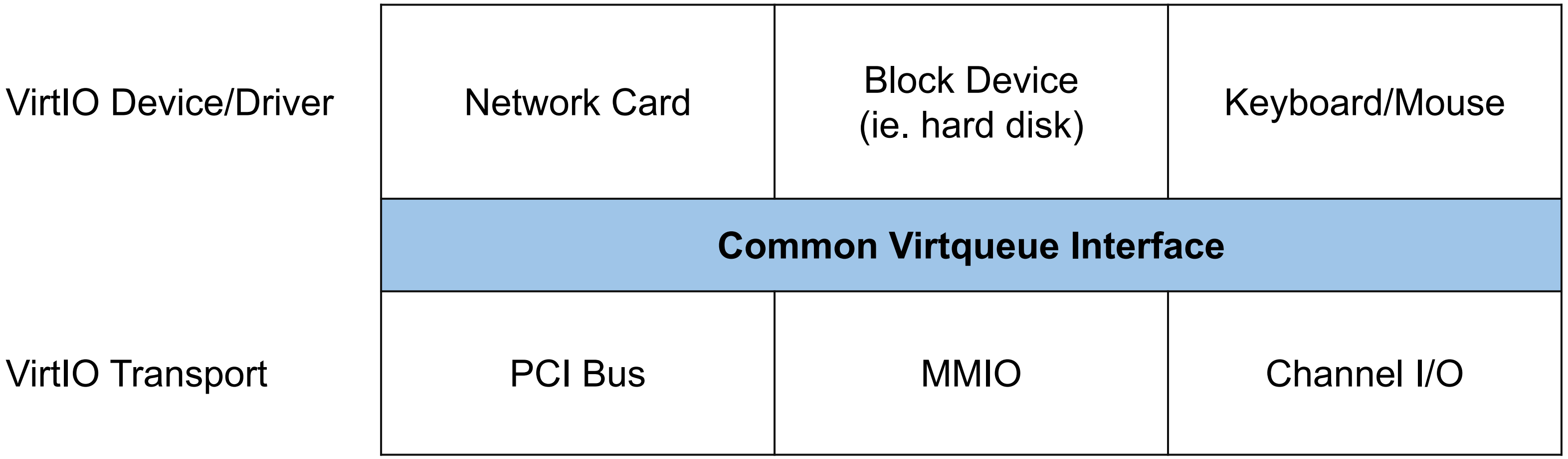
- Guest performs normal PCI device discovery
- Vendor ID: 0x1AF4
- PCI Device ID:
  - 0x1000 + device id (legacy) or
  - 0x1040 + device id

PCI Device Layout

- Configured via I/O and/or memory regions, specified by Virtio Structure PCI Capabilities
- Virtio Structure PCI Capabilities
  - Common configuration
  - Notifications
  - ISR Status
  - Device-specific configuration (optional)
  - PCI configuration access

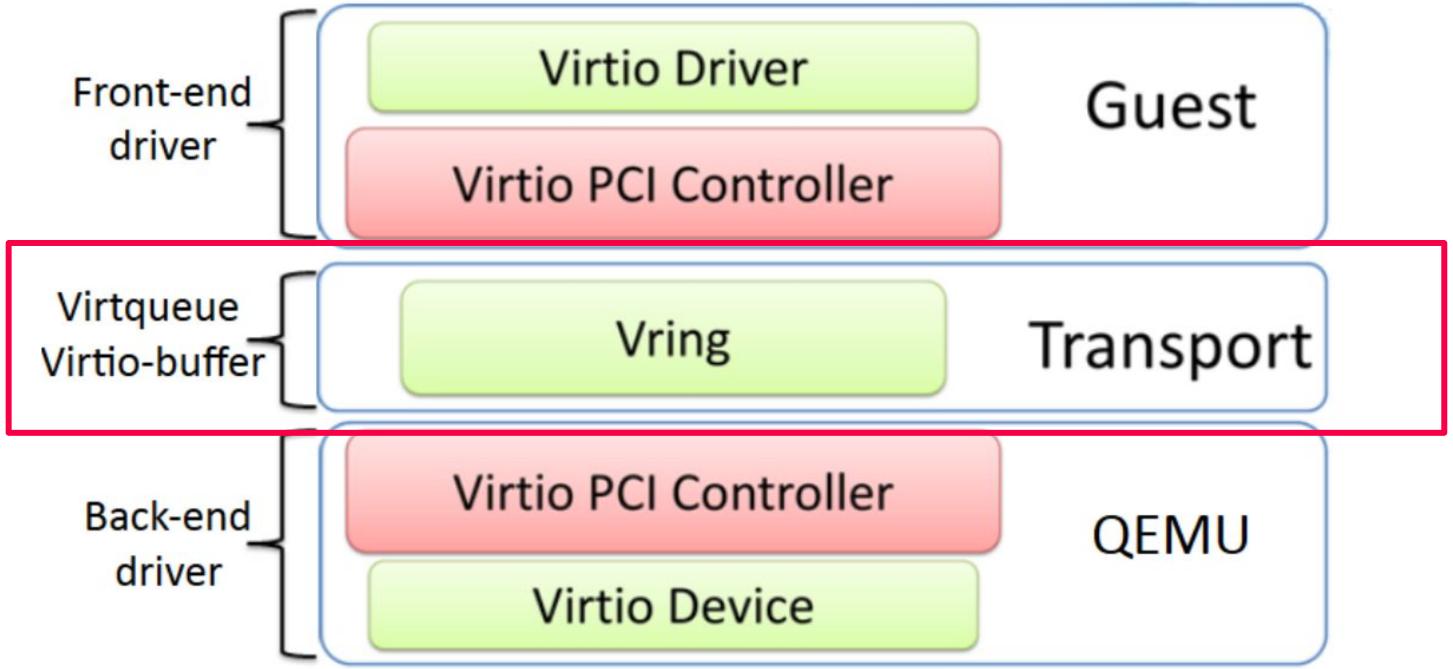
Outline

- Introduction
- Component Overview
- VirtIO over PCI
- **Virtqueues**
- Devices and Drivers



VirtIO Device/Driver	Network Card	Block Device (ie. hard disk)	Keyboard/Mouse
	Common Virtqueue Interface		
VirtIO Transport	PCI Bus	MMIO	Channel I/O

VirtIO Communication Host/Guest



https://www.cs.cmu.edu/~412/lectures/Virtio_2015-10-14.pdf

Basics

- Data channel between front-end and back-end
- Just a queue of guest’s buffers
    - host consumes
    - read / write
- Shared memory pages
    - inside guest physical memory
- Each device has own virtqueue
- Each virtqueue has own vring

Basics 2

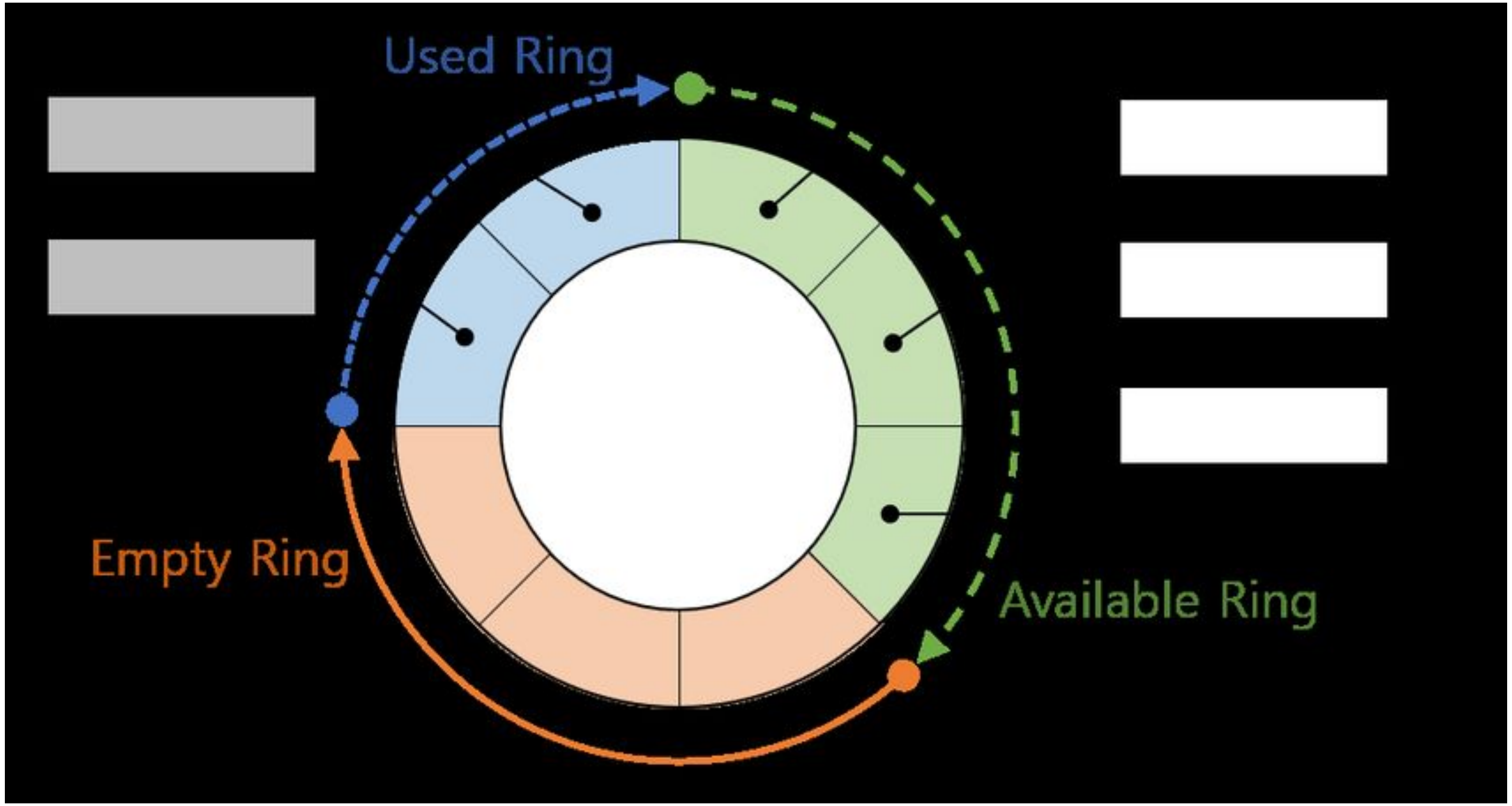
- Provides driver to device notifications
  - signal if buffers added to queue
- Up to transport to provide method to dispatch notification
  - PCI interruptions
  - Memory writing
  - virtqueue only standardizes semantics!

Ring Buffer

Ring Buffer

- Implemented as Vring
  - ring buffer based queue
  - push/pop operations
- Compontents:
  - Descriptor Ring
  - Available Ring
  - Used Ring

Ring Buffer



https://www.researchgate.net/figure/Management-of-RX-virtqueue_fig2_337760284

Descriptor Ring

- Array of guest addressed buffers and length
- Each Descriptor:
    - set of flags for information
        - if buffer continues in other buffer -> 0x1 set
        - if buffer is write-only for device -> 0x2 set
        - if read-only -> clear

Layout of a single descriptor

```
struct virtq_desc {
        le64 addr;
        le32 len;
        le16 flags;
        le16 next;
};
```

Available Ring

- Room where driver places descriptor
- Placed buffer not consumed immediately
- 2 important fields:
  - idx
    - where driver puts next descriptor entry
  - flags
    - least sign. bit indicates notification
    - VIRTQ_AVAIL_F_NO_INTERRUPT
- array of integers same length as descriptor ring

Layout of a avail virtqueue

```
struct virtq_avail {
        le16 flags;
        le16 idx;
        le16 ring[q_size];
};
```

Used Ring

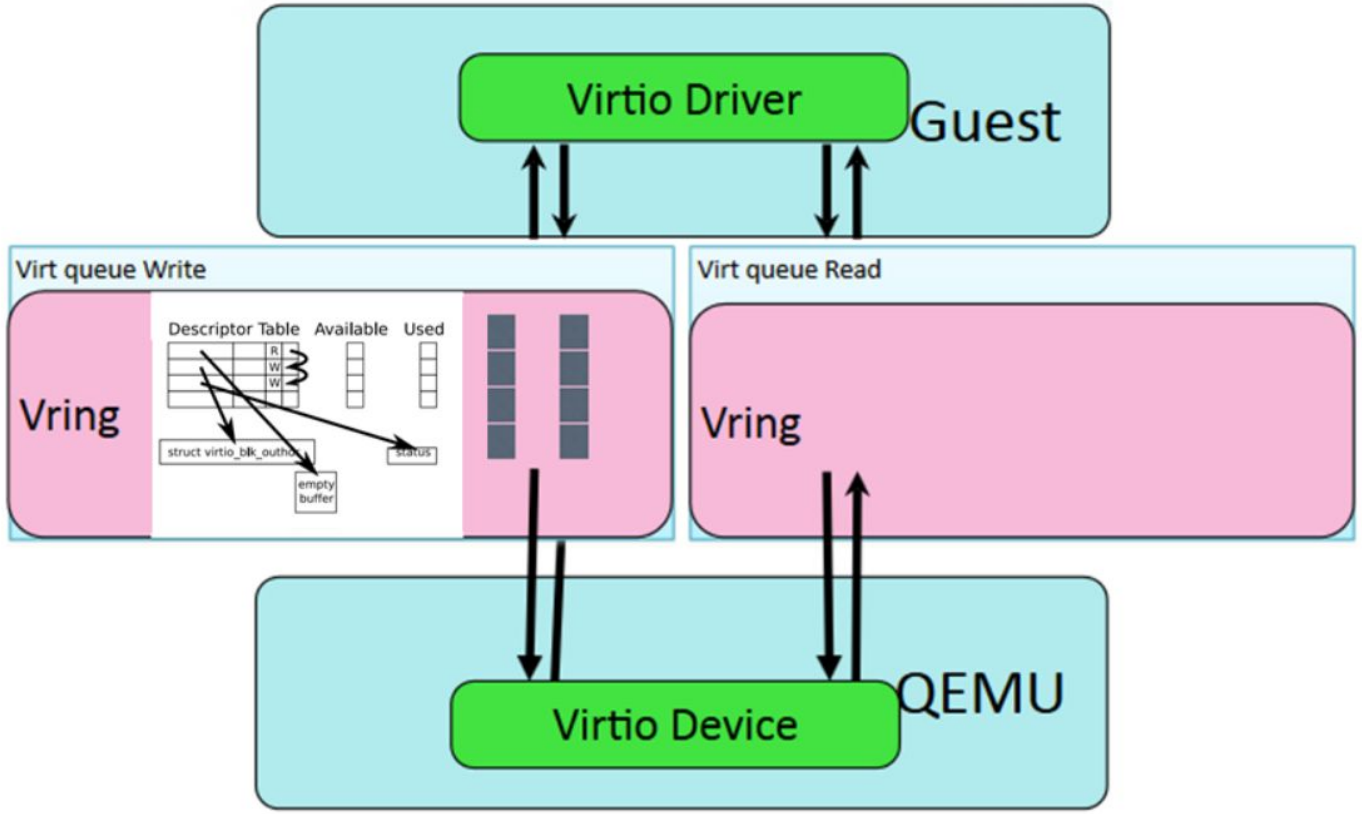
- Room where device returns used buffers
- 2 important fields:
    - idx
        - where driver puts next descriptor entry
    - flags
        - least sign. bit indicates notification
- array of used descriptors
    - device returns descriptor index and length (when written)

Layout of a used virtqueue

```
struct virtq_avail {
        le16 flags;
        le16 idx;
        le16 virtq_used_elem ring[q_size];
};
struct virtq_used_elem {
        le32 id;
        le32 len;
};
```

VRing

1. Guest:
   - adds buffer to vring
2. QEMU:
   - signaled to pop buffer
3. QEMU:
   - adds data to vring
4. Guest:
   - signaled to getbuffer
5. Guest:
   - gets buffer with data



https://www.cs.cmu.edu/~412/lectures/Virtio_2015-10-14.pdf

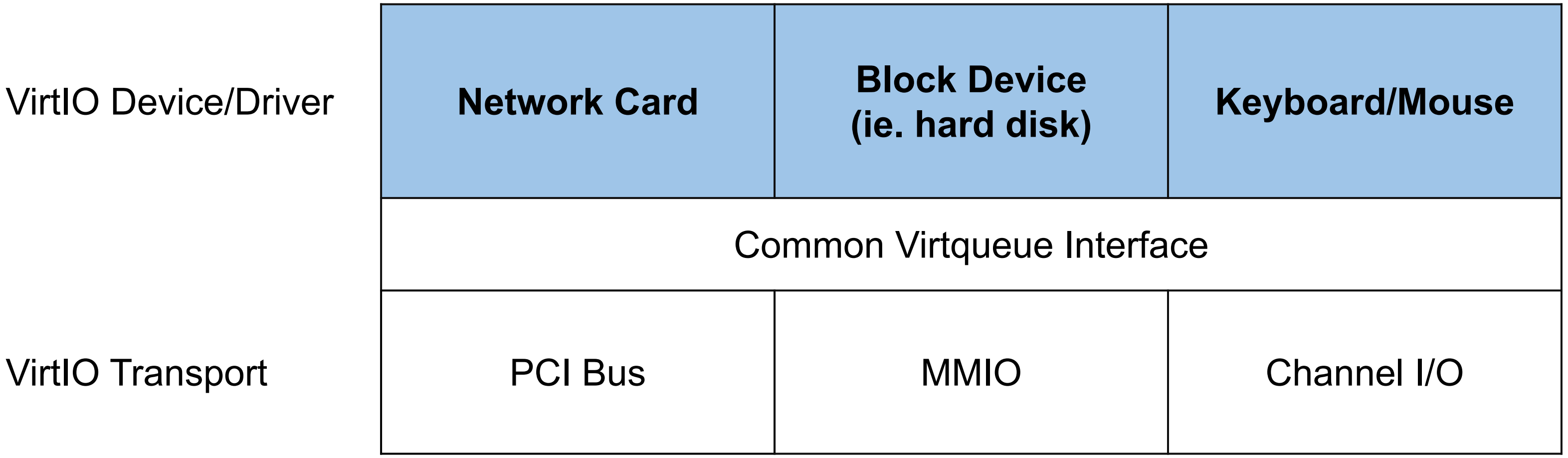
Virtqueue High Level Interface

```
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);
};
```

Source: Linux Kernel Source
https://elixir.bootlin.com/linux/v2.6.31/source/include/linux/virtio.h#L61

Outline

- Introduction
- Component Overview
- VirtIO over PCI
- Virtqueues
- **Devices and Drivers**



VirtIO Device/Driver

Network Card

Block Device (ie. hard disk)

Keyboard/Mouse

Common Virtqueue Interface

VirtIO Transport

PCI Bus

MMIO

Channel I/O

Device Types

0	reserved (invalid)	10	mac80211 wlan
1	network card	11	rproc serial
2	block device	12	virtio CAIF
3	console	13	memory balloon
4	entropy source	14	
5	memory ballooning (traditional)	15	
6	ioMemory	16	GPU device
7	rpmsg	17	Timer/Clock device
8	SCSI host	18	Input device
9	9P transport		

Block Device

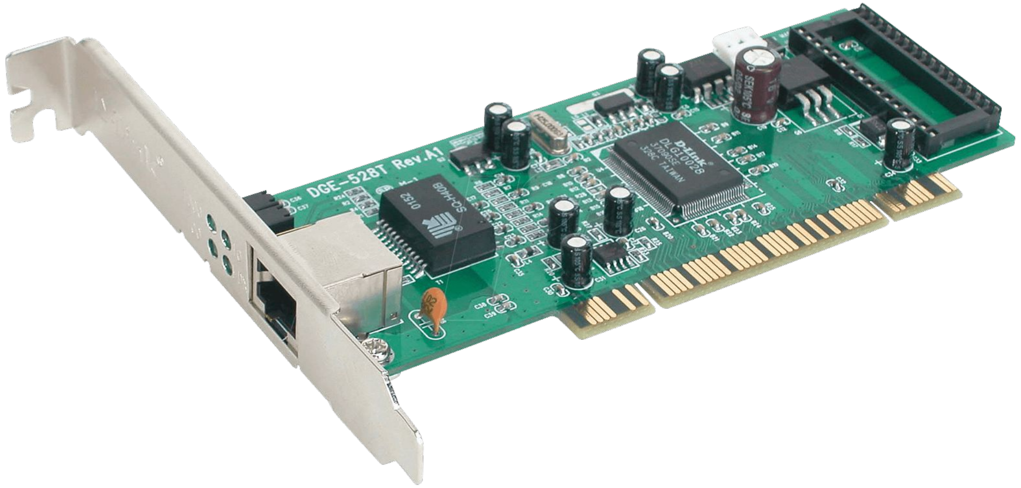
- Virtualized storage devices like hard disks, USB sticks, DVDs, …
- Single virtqueue for read and write requests
- Read requests must send along a host-writeable-buffer

Input Devices: Keyboard/Mouse

- evdev
  - generic input event interface used in Linux and FreeBSD
  - All input events (mouse movements, key presses) are translated to standardized format
- VirtIO input devices allow passing those events to the guest
- Easy to parse, even on kernels that do not use evdev

Network Card

- Exposes network interface (virtual or physical)
- Dedicated queues for data communication
  - Receive (RX)
  - Transmit (TX)
- Can use multiple pairs of queues



https://www.reichelt.com/de/en/10-100-1000-mbit-s-pci-network-interface-card-d-link-dge-528t-p69159.html

GPU device

- Can be used in
  - VESA mode
    - exposes VESA framebuffer
  - OpenGL
    - allows direct access to OpenGL interface

https://trendinline.com/new/nvidia-geforce-rtx-3090-founders-edition-graphics-card-2/

Resources for Implementing in SWEB

VirtIO Specification:
http://docs.oasis-open.org/virtio/virtio/v1.0/cs04/virtio-v1.0-cs04.html#x1-800004

OSDev Wiki:
https://wiki.osdev.org/Virtio

Driver Implementation Guide:
http://www.dumais.io/index.php?article=aca38a9a2b065b24dfa1dee728062a12

Linux Kernel Source:
https://elixir.bootlin.com/linux/latest/source/include/linux/virtio.h

Shameless Advertisement - DCTF 2021

From: Fri, 14 May 2021, 17:00
Until: Sun, 16 May 2021, 23:59

Beginner CTF, you should be more than qualified ;)

Everyone welcome to join

https://discord.gg/uPD44KA

Questions?

Feel free to ask.