VIRTIO

Yi-Man Ma & Chang-Jung Lin

2011 / 01 / 05

Agenda

- Introduction
- Architecture
- VirtQueue & Vring
- Five APIs
- Adding a new Virtio Device / Driver

INTRODUCTION

Introduction

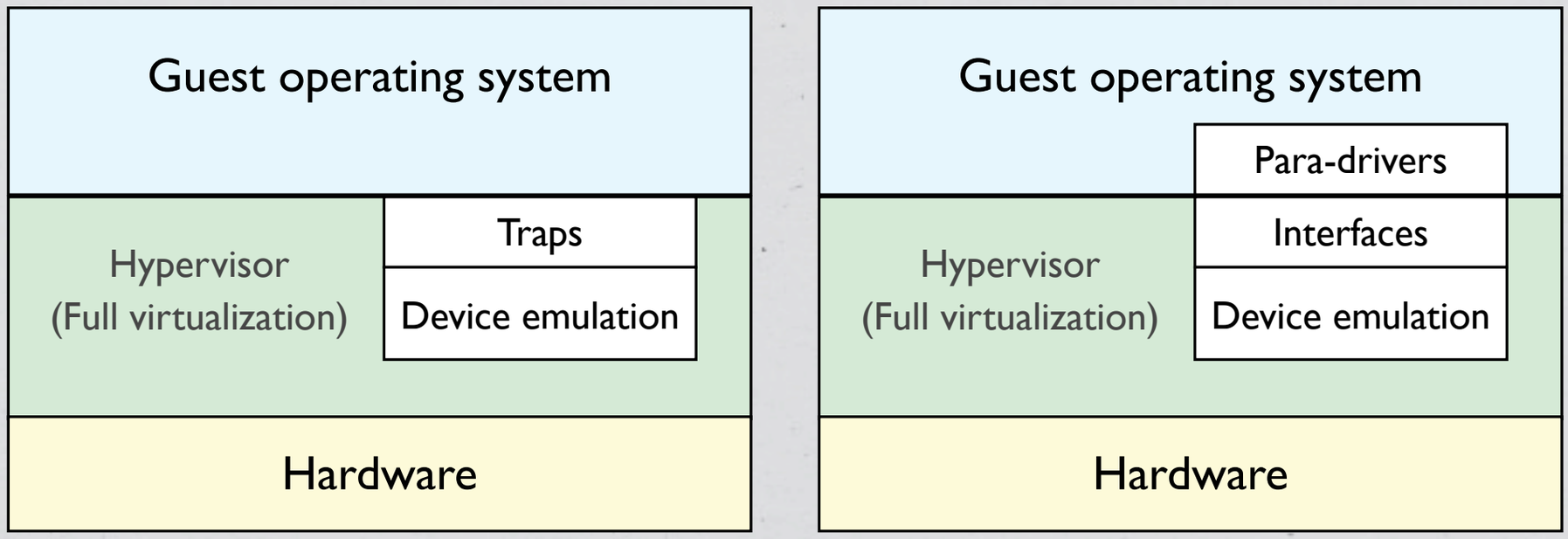
Create a shortcut to deal with IO Request

Coalesced IO Request by VirtQueue

Introduction

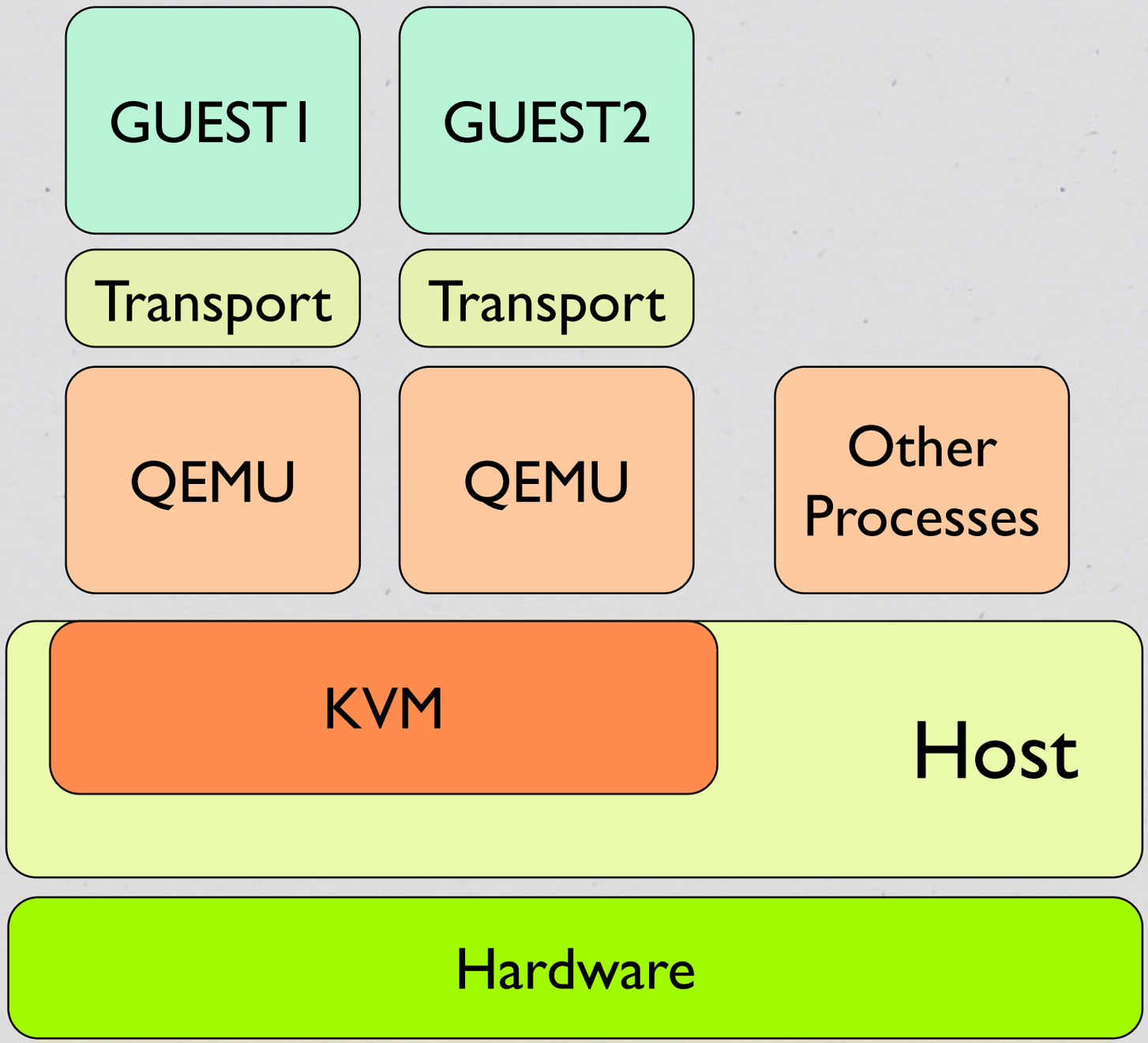
Device emulation in full virtualization

and paravirtualization environments

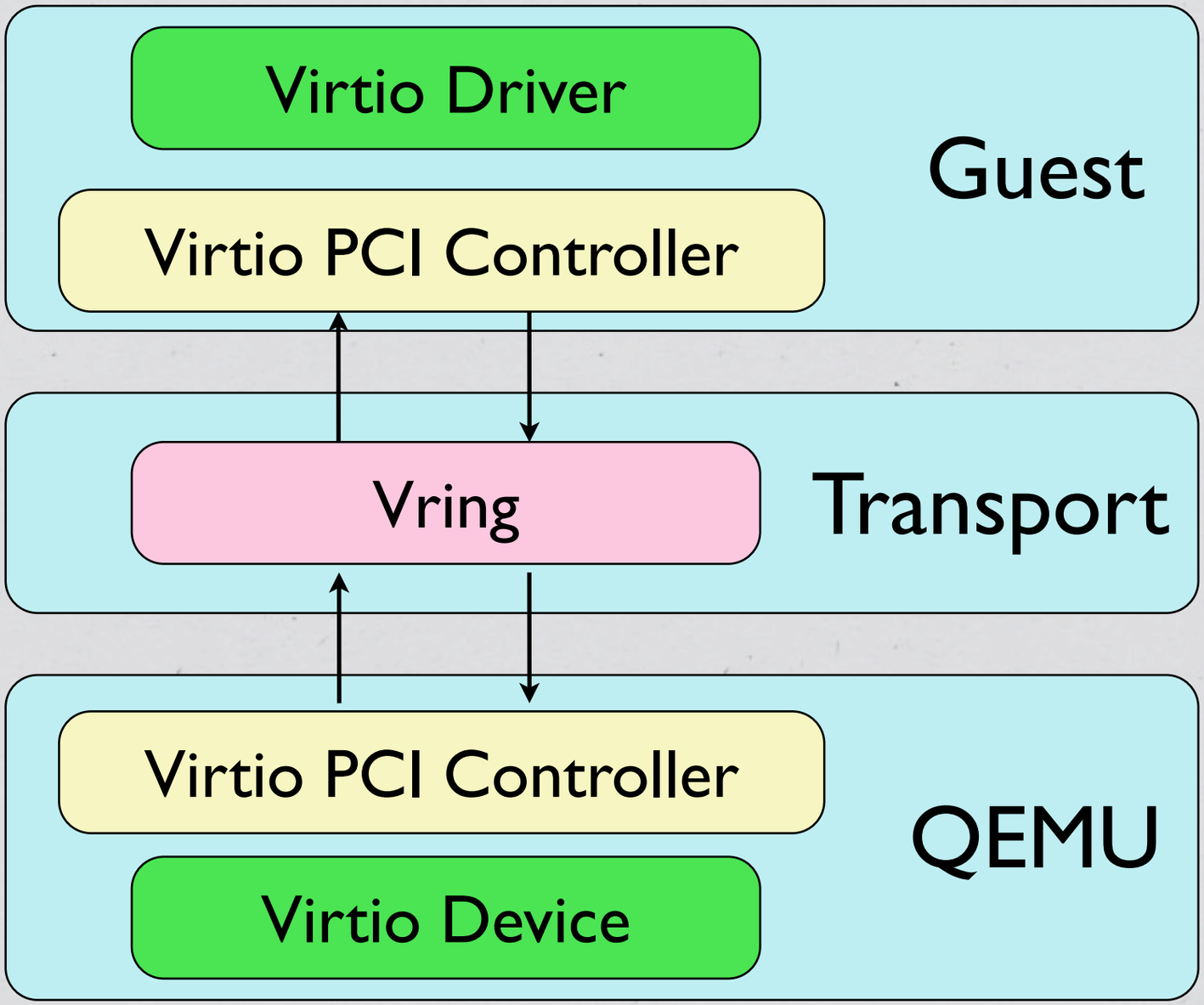


ARCHITECTURE

Architecture



Architecture



VIRTQUEUE & VRING

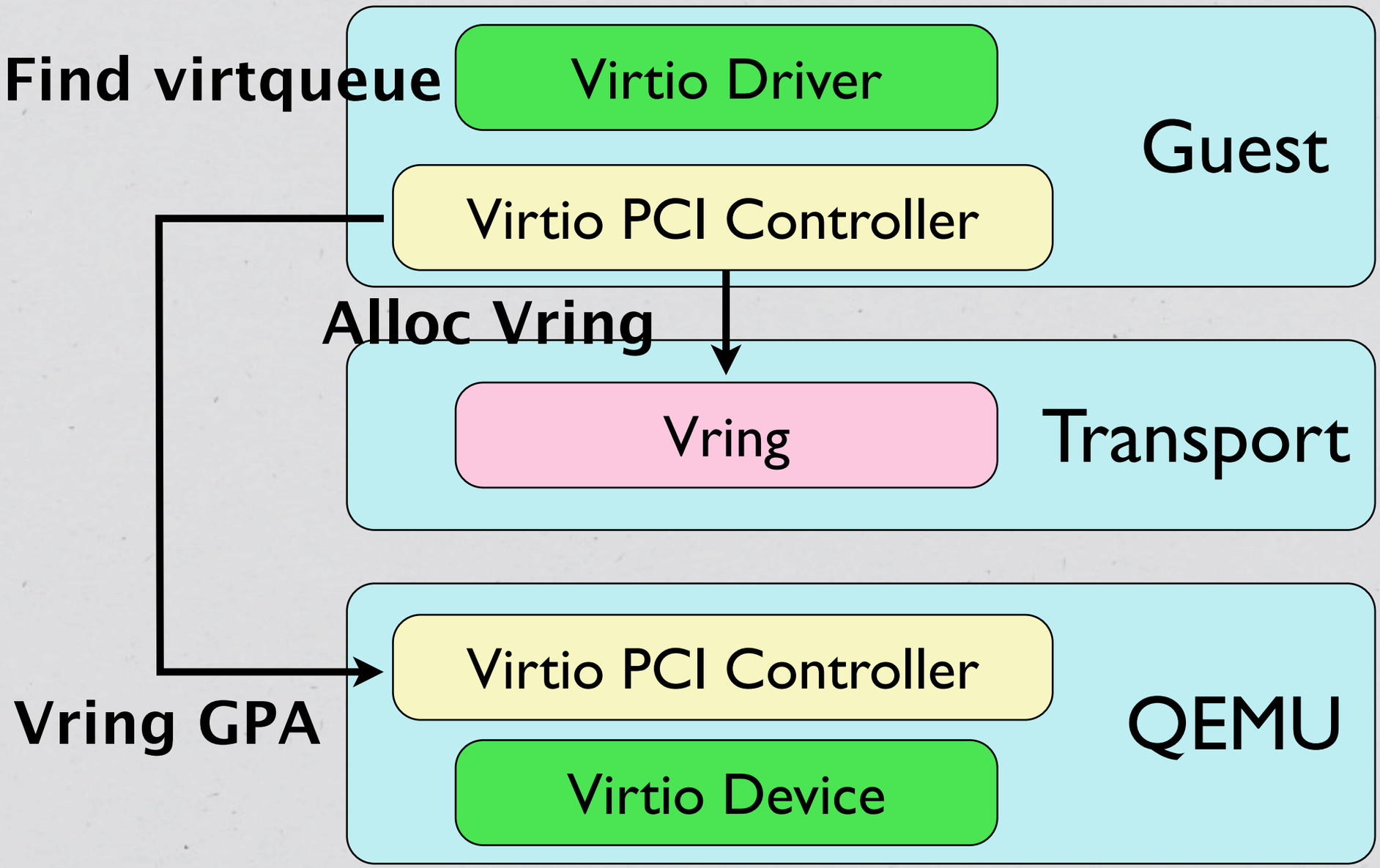
VirtQueue

- Each Device have it's own virtqueues.
- Queue Requests

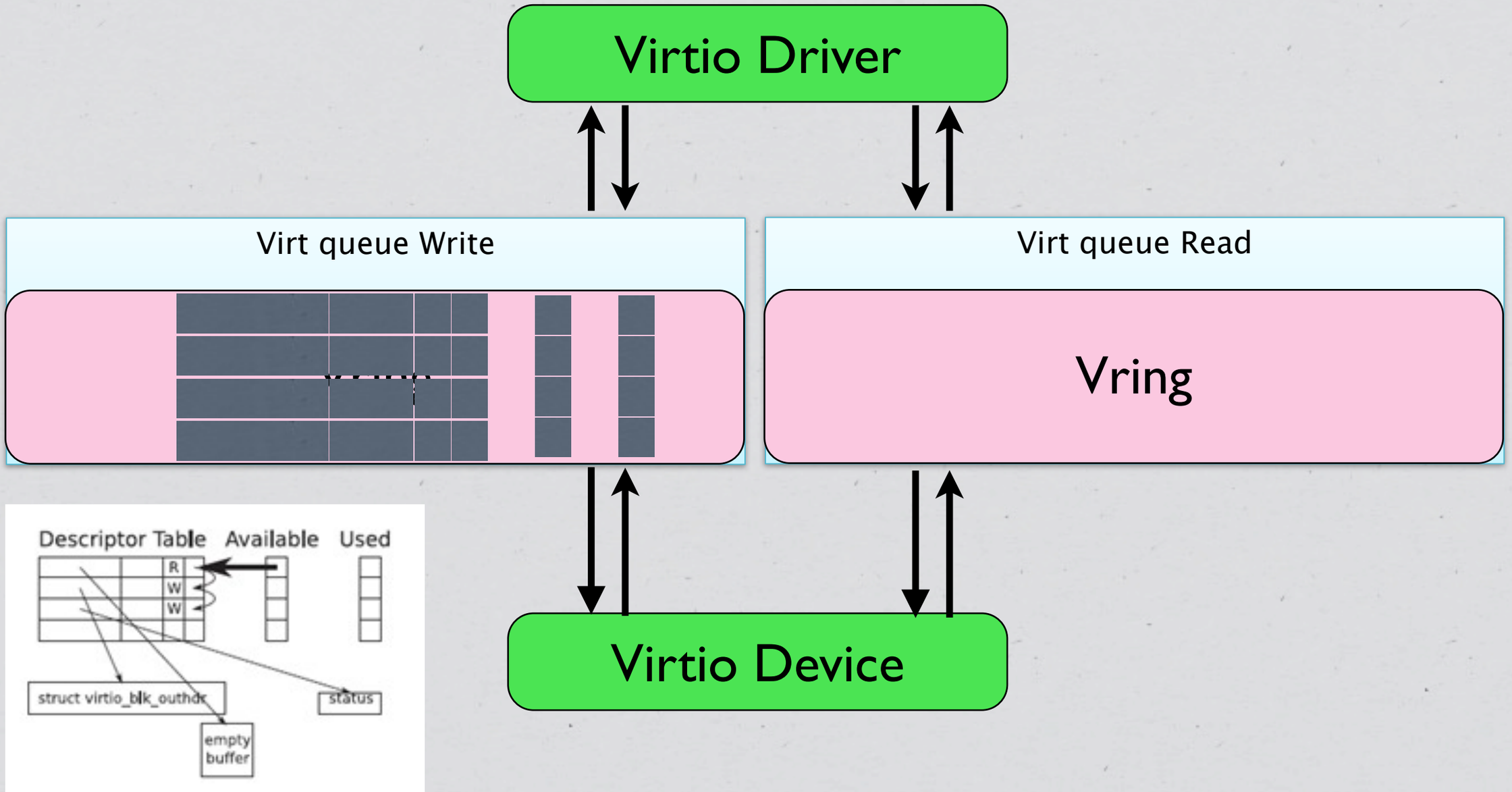
Vring

- Each virtqueue have it's own vring.
- Memory Mapped between QEMU & Guest
- Actual memory layout for queue

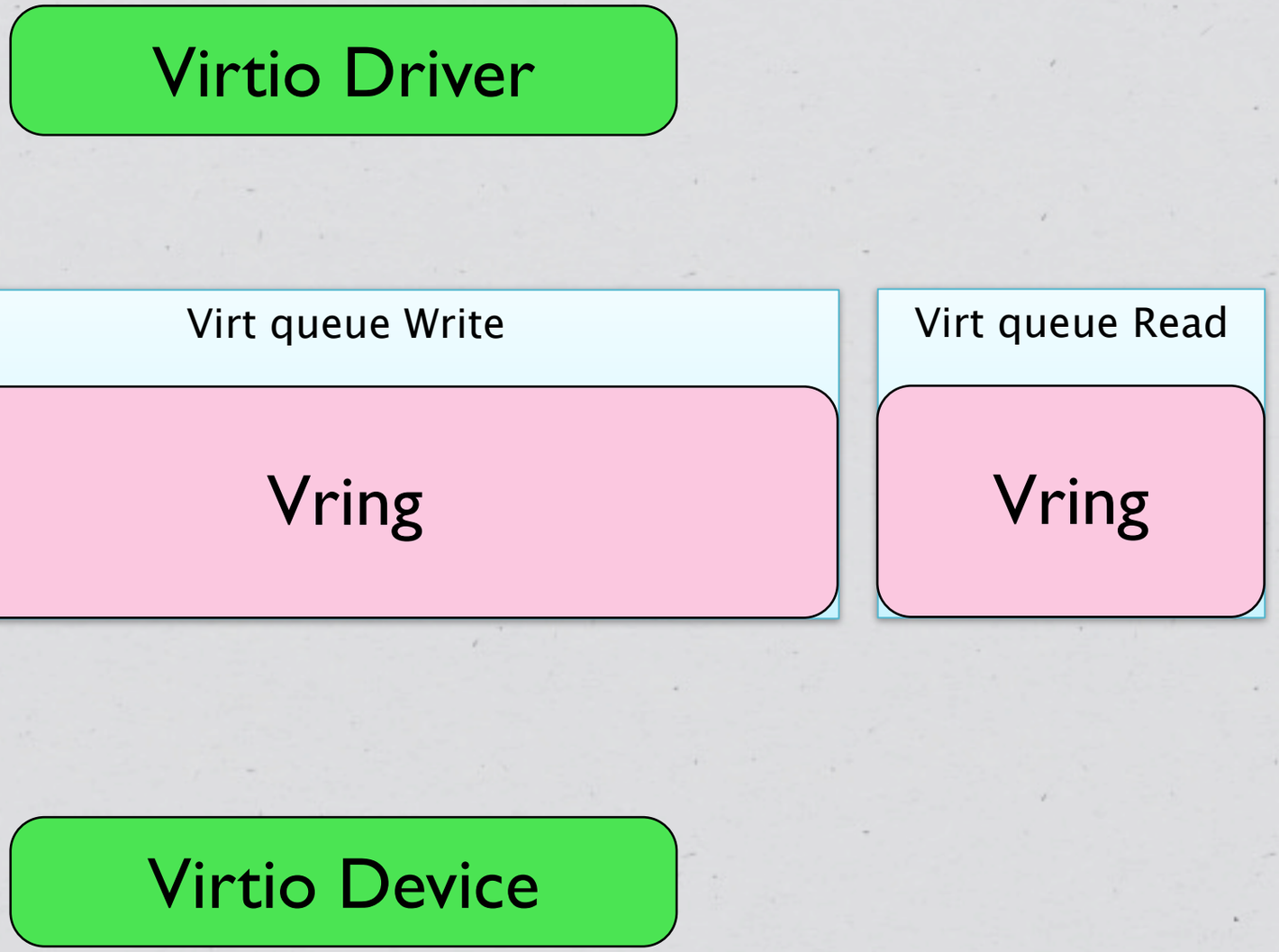
VirtQueue
Vring



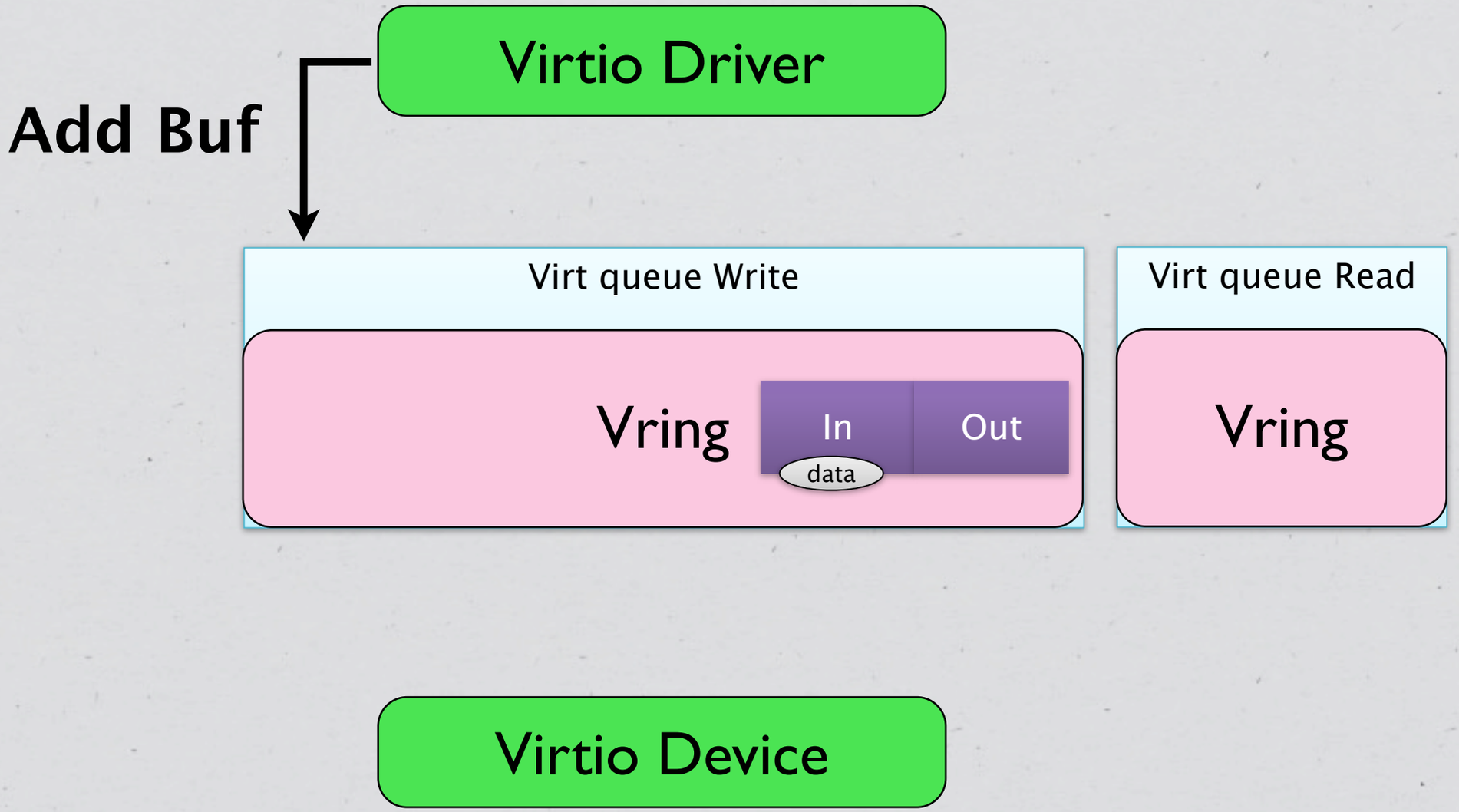
VirtQueue
Vring



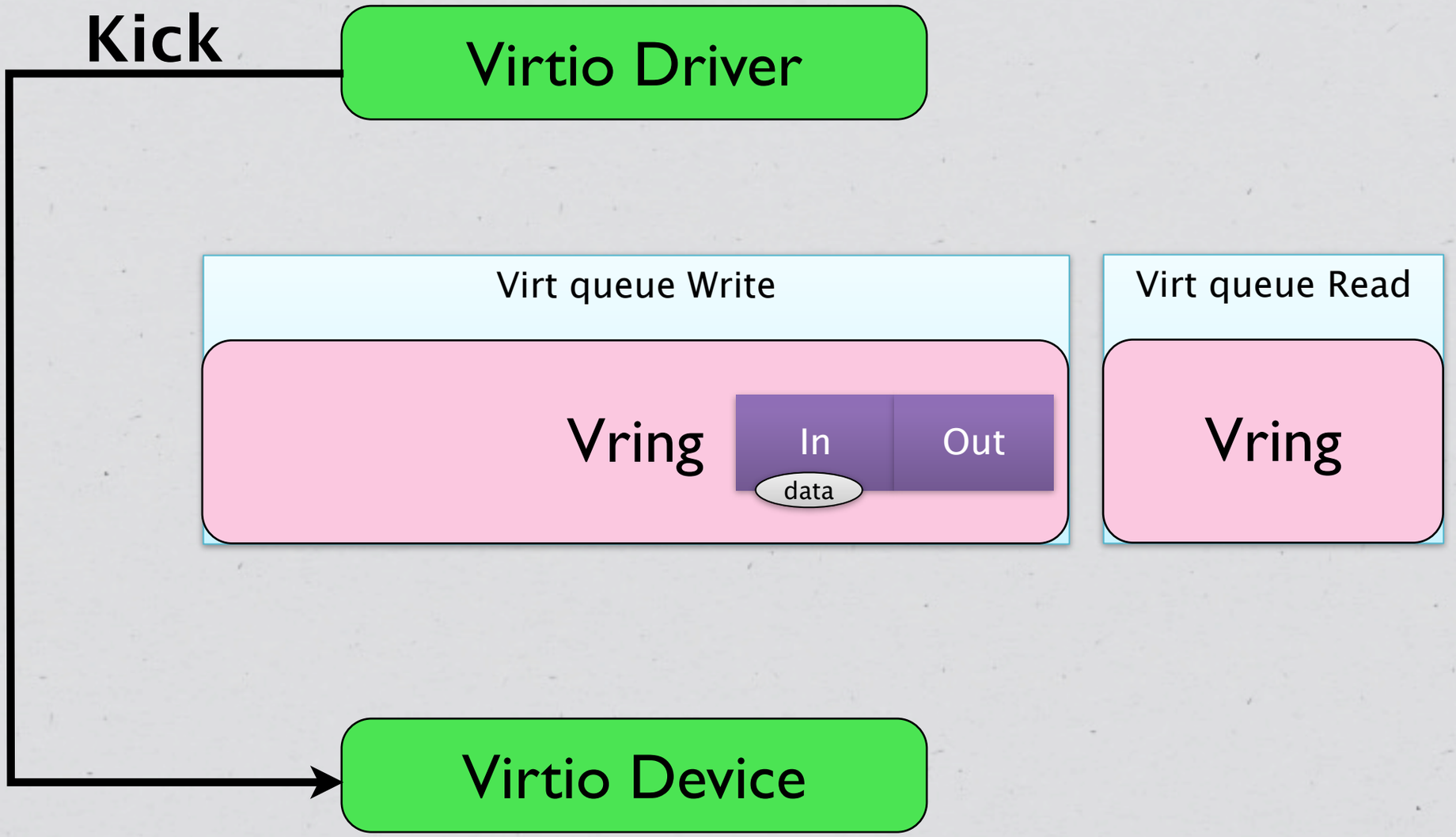
VirtQueue
Vring



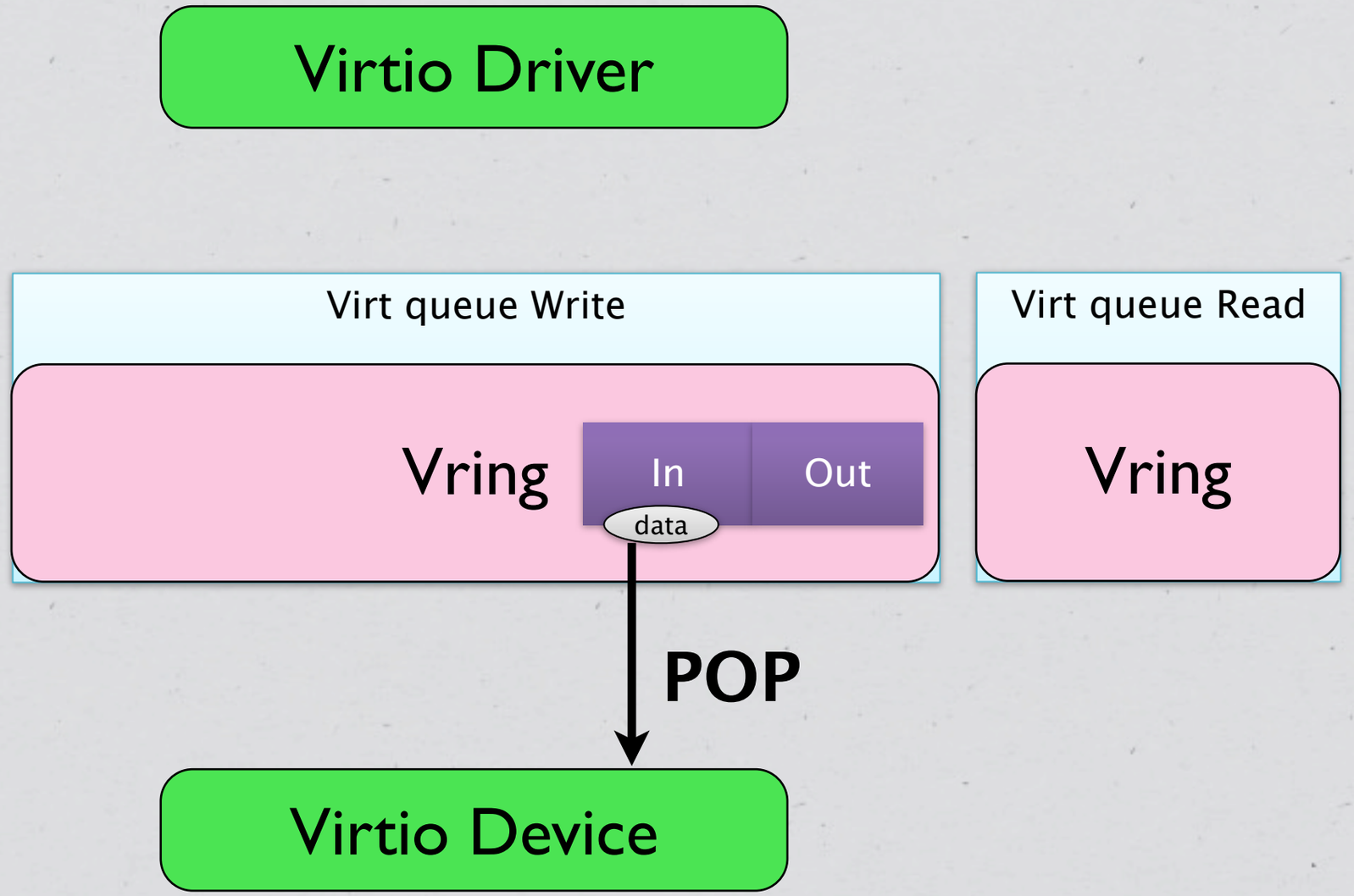
VirtQueue
Vring



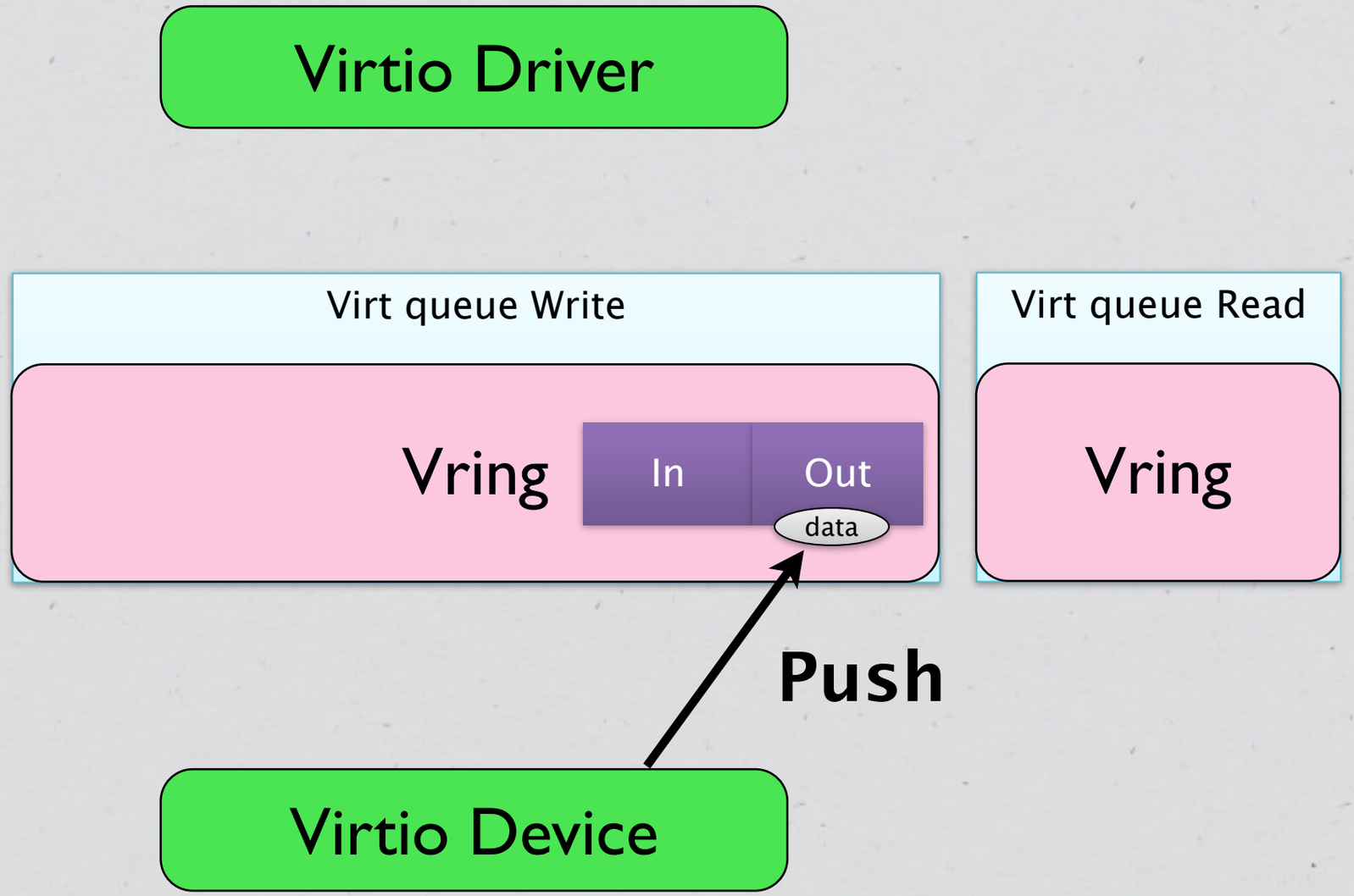
VirtQueue
Vring



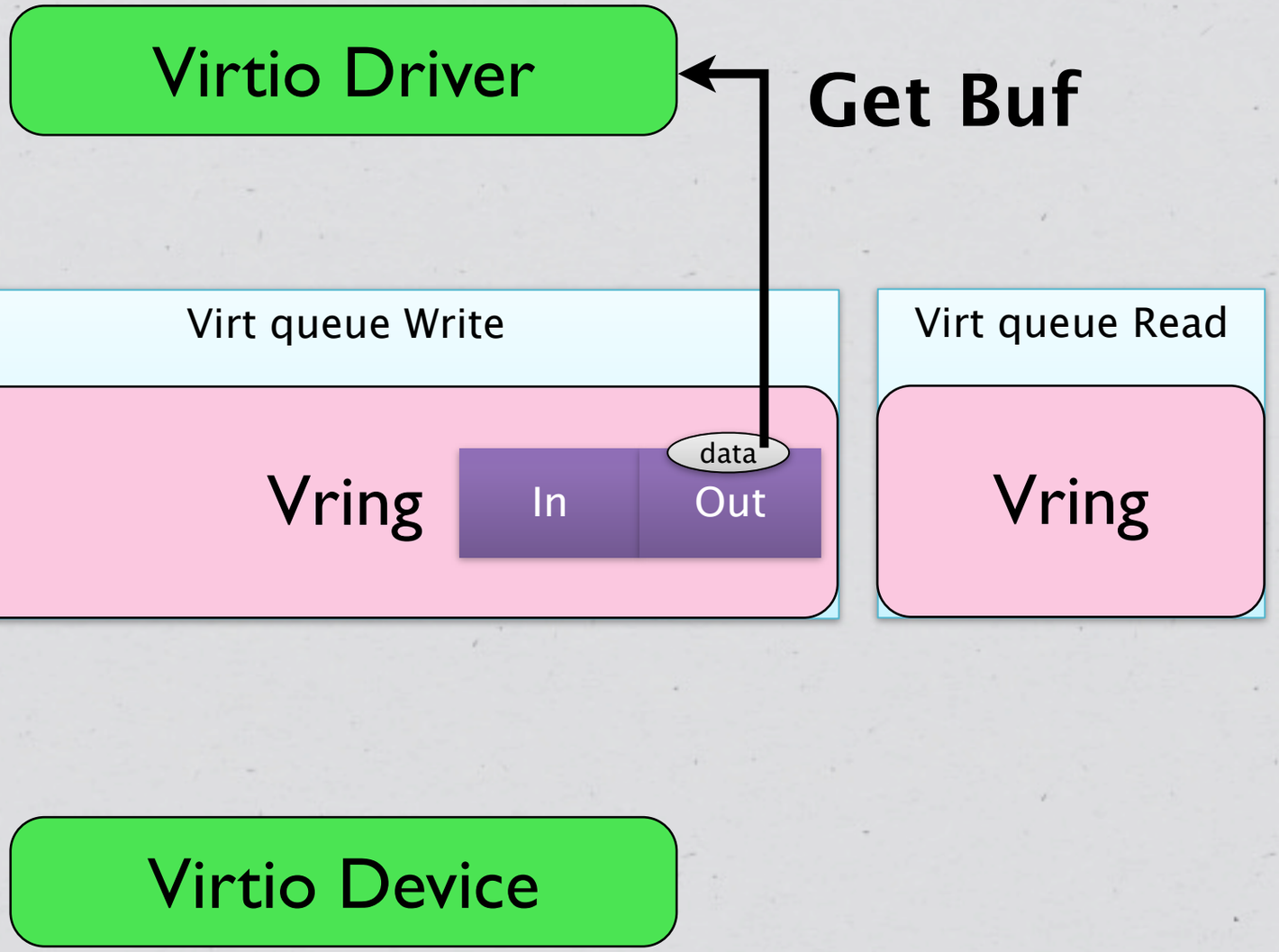
VirtQueue
Vring



VirtQueue
Vring



VirtQueue
Vring



FIVE APIS

Five APIs

* add_buf
* kick
* get_buf
* disable_cb
* enable_cb

Five APIs
add & get buf

* add_buf
  * expose buffer to other end
* get_buf
  * get the next used buffer

Five APIs kick

* kick
* update after add_buf
* notify QEMU to deal with the data

Five APIs
CB relative

* disable_cb
  * disable callbacks
    * not necessarily synchronous
    * unreliable and only useful as an optimization
* enable_cb
  * restart callbacks after disable_cb

NEW DEVICE / DRIVER

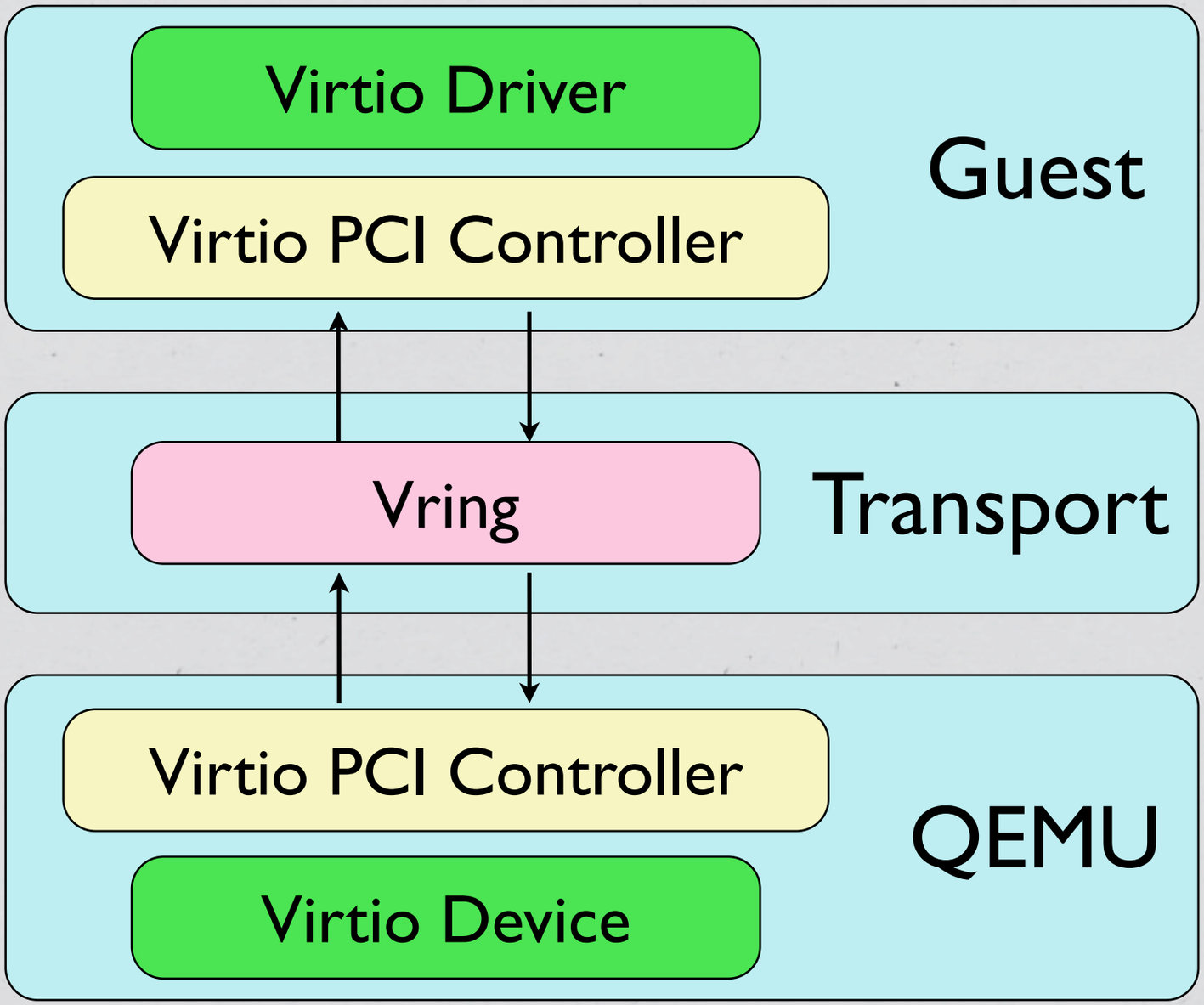
New Driver

* include/linux/virtio_ids.h
* include/linux/virtio.h
* include/linux/virtio_[device].h
* drivers/virtio/virtio_[device].c
* drivers/virtio/virtio_pci.c

New Device

* hw/virtio-[device].c
* hw/virtio-[device].h
* hw/pci.h
* hw/virtio_pci.c

Architecture



New Device
add device id

Virtio PCI Controller

* Modify qemu-src/hw/pci.h

```
#define PCI_DEVICE_ID_VIRTIO_[device]          0x1004
```

New Device
initial handler

Virtio PCI Controller

* Modify qemu-src/hw/virtio-pci.c

```
static int virtio_[device]_init_pci(PCIDevice *pci_dev)
{
    VirtIOPCIProxy *proxy =
                    DO_UPCAST(VirtIOPCIProxy, pci_dev, pci_dev);
    VirtIODevice *vdev;
    vdev = virtio_[device]_init(&pci_dev->qdev);
    if (!vdev) {
        return -1;
    }
    virtio_init_pci(proxy, vdev);
    return  0;
}
```

New Device
exit handler

Virtio PCI Controller

* Modify qemu-src/hw/virtio-pci.c

```
static int virtio_[device]_exit_pci (PCIDevice *pci_dev)
{
    VirtIOPCIProxy *proxy =
                DO_UPCAST(VirtIOPCIProxy, pci_dev, pci_dev);

    virtio_pci_stop_ioeventfd(proxy);
    virtio_[device]_exit(proxy->vdev);

    return  virtio_exit_pci(pci_dev);
}
```

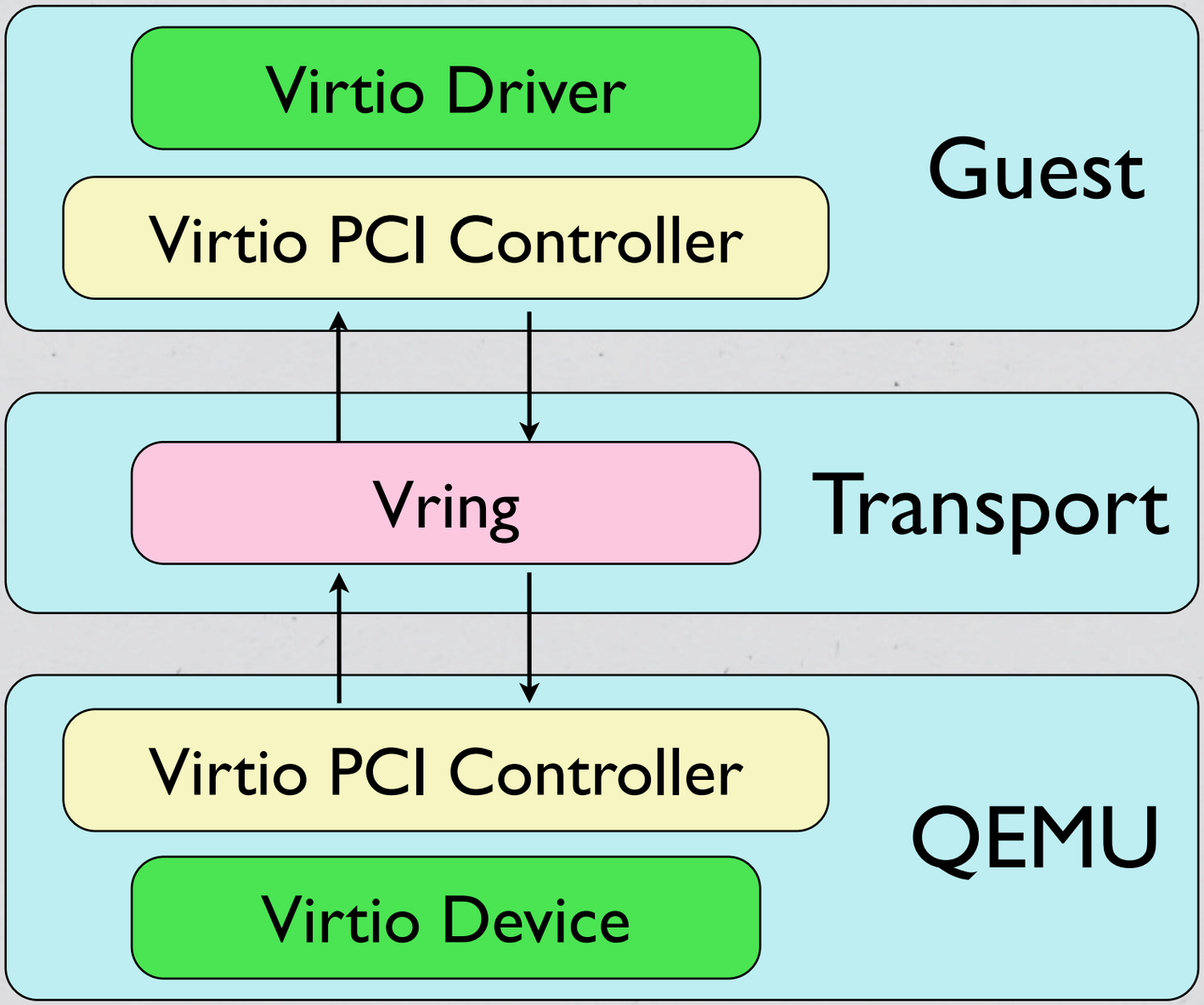
New Device
add device info

Virtio PCI Controller

* Modify qemu-src/hw/virtio-pci.c

```
static  PCIDeviceInfo virtio_info[] = {
    {
        .qdev.name =  "virtio-[device]-pci",
        .qdev.alias   =  "virtio-[device]",
        .init              =   virtio_[device]_init_pci,
        .exit               =  virtio_[device]_exit_pci,
        .device_id    = PCI_DEVICE_ID_VIRTIO_[device],
        .qdev.props = (Property[]) {
                DEFINE_VIRTIO_COMMON_FEATURES(
                                        VirtIOPCIProxy, host_features),
                DEFINE_PROP_END_OF_LIST(),
        },
        ..... (Skip)
```

Architecture



New Device

Virtio Device

* Add virtio-[device].c, virtio[device].h
* Put them under the directory qemu-src/hw/

New Device
virtio-[device].h

Virtio Device

```
#ifndef  _QEMU_VIRTIO_[device]_H

#include "virtio.h“

#define _QEMU_VIRTIO_[device]_H
#define VIRTIO_ID_[device]  [num]   //ID for your device

VirtIODevice *virtio_[device]_init(DeviceState *dev);
void virtio_[device]_exit(VirtIODevice *vdev);

#endif
```

New Device
virtio-[device].c

Virtio Device

```
#include  <qemu-common.h>
#include "qemu-error.h"
#include "virtio-[device].h"
#include "virtio.h"
```

```
typedef struct VirtIO[device]
{
    VirtIODevice vdev;
    VirtQueue *vq[num];
    DeviceState *qdev;
    int status;
} VirtIO[device];
```

```
struct virtio_[device]_config {
    int config;
} ;
```

New Device
virtio structure

Virtio Device

```
struct VirtIODevice
{
    ..........
    uint32_t   (*get_features)(VirtIODevice *vdev, uint32_t
requested_features);
    uint32_t   (*bad_features)(VirtIODevice *vdev);
    void   (*set_features)(VirtIODevice *vdev, uint32_t val);
    void   (*get_config)(VirtIODevice *vdev, uint8_t *config);
    void   (*set_config)(VirtIODevice *vdev, const uint8_t *config);
    void   (*reset)(VirtIODevice *vdev);
    void   (*set_status)(VirtIODevice *vdev, uint8_t val);
    .......
} ;
```

New Device
need implement

Virtio Device

```
static VirtIO[device]  *to_virtio_[device](VirtIODevice *vdev)
        //Get  device defined  by yourself
static uint32_t   virtio_[device]_ get_features (VirtIODevice *vdev,
uint32_t  features);
static uint32_t   virtio_[device]_ bad_features (VirtIODevice
*vdev);
static void  virtio_[device]_ set_features (VirtIODevice *vdev,
uint32_t val);
static void  virtio_[device]_ get_config (VirtIODevice *vdev,
uint8_t *config);
static void  virtio_[device]_ set_config (VirtIODevice *vdev, const
uint8_t *config);
```

New Device
need implement

Virtio Device

```
static void  virtio_[device]_ reset (VirtIODevice *vdev);
static void  virtio_[device]_ set_status (VirtIODevice *vdev,
uint8_t val);

VirtIODevice *virtio_[device]_init(DeviceState *dev);
        //Initialize your device
void virtio_[device]_exit (VirtIODevice *vdev);

static void virtio_[device]_save(QEMUFile *f, void *opaque);
static int virtio_[device]_load(QEMUFile *f, void *opaque, int
version_id);

static void virtio_[device]_handle_out(VirtIODevice *vdev,
VirtQueue *vq);
```

New Device
about features

Virtio Device

```
static uint32_t   virtio_[device]_ get_features (VirtIODevice *vdev,
uint32_t features)
{
    VirtIO[device] *n = to_virtio_[device](vdev);
    return  features;
}
static uint32_t   virtio_[device]_ bad_features (VirtIODevice
*vdev){
    uint32_t features = 0;
    return features;
}
static void  virtio_[device]_ set_features (VirtIODevice *vdev,
uint32_t val){
    VirtIOMy *n = to_virtio_[device](vdev);
}
```

New Device
about config

Virtio Device

```
static void  virtio_[device]_ get_config (VirtIODevice *vdev,
uint8_t *config)
{
    VirtIO[device] *n = to_virtio_[device](vdev);
    struct virtio_[device]_config cfg;
    stw_p(&cfg.config, n->status);
    memcpy(config, &cfg, sizeof(cfg));
}
static void  virtio_[device]_ set_config (VirtIODevice *vdev, const
uint8_t *config)
{
    VirtIO[de vice] *n = to_virtio_[device](vdev);
    struct virtio_[device]_config cfg;
}
```

New Device
set & reset

Virtio Device

```
static void  virtio_[device]_ reset (VirtIODevice *vdev)
{
}

static void  virtio_[device]_ set_status (VirtIODevice *vdev,
uint8_t val)
{
    VirtIO[device] *n = to_virtio_my(vdev);
}
```

New Device
initial device

Virtio Device

```
VirtIODevice *virtio_[device]_init(DeviceState *dev) {
    VirtIO[device] *s;

   s = (VirtIO[device]*)virtio_common_init("virtio-[device]",
VIRTIO_ID_[device], sizeof(struct virtio_[device]_config),
sizeof(VirtIO[device]));
   s->vdev.get_config = virtio_[device]get_config;
   s->vdev.set_config = virtio_[device]_set_config;
   s->vdev.get_features = virtio_[device]_get_features;
   s->vdev.set_features = virtio_[device]_set_features;
   s->vdev.bad_features = virtio_[device]_bad_features;
   s->vdev.reset = virtio_[device]_reset;
   s->vdev.set_status = virtio_[device]_set_status;

.....
```

New Device
initial device

Virtio Device

```
.....
  s->vdev.set_status = virtio_[device]_set_status;
    s->vq[0] = virtio_add_queue(&s->vdev, 1024,
virtio_[device]_handle_out1);
    s->vq[1] = virtio_add_queue(&s->vdev, 1024,
virtio_[device]_handle_out2);
    s->qdev = dev;
    s->status = 99;
    register_savevm(dev, "virtio-[device]”, 0, 2,
virtio_[device]_save, virtio_[device]_load, s);

    return &s->vdev;
}
```

New Device
exit device

Virtio Device

```
void virtio_[device]_exit (VirtIODevice *vdev)
{
    VirtIO[device] *s = to_virtio_[device](vdev);
    unregister_savevm(s->qdev, "virtio-[device]“ , s);
}
```

New Device
save & load

Virtio Device

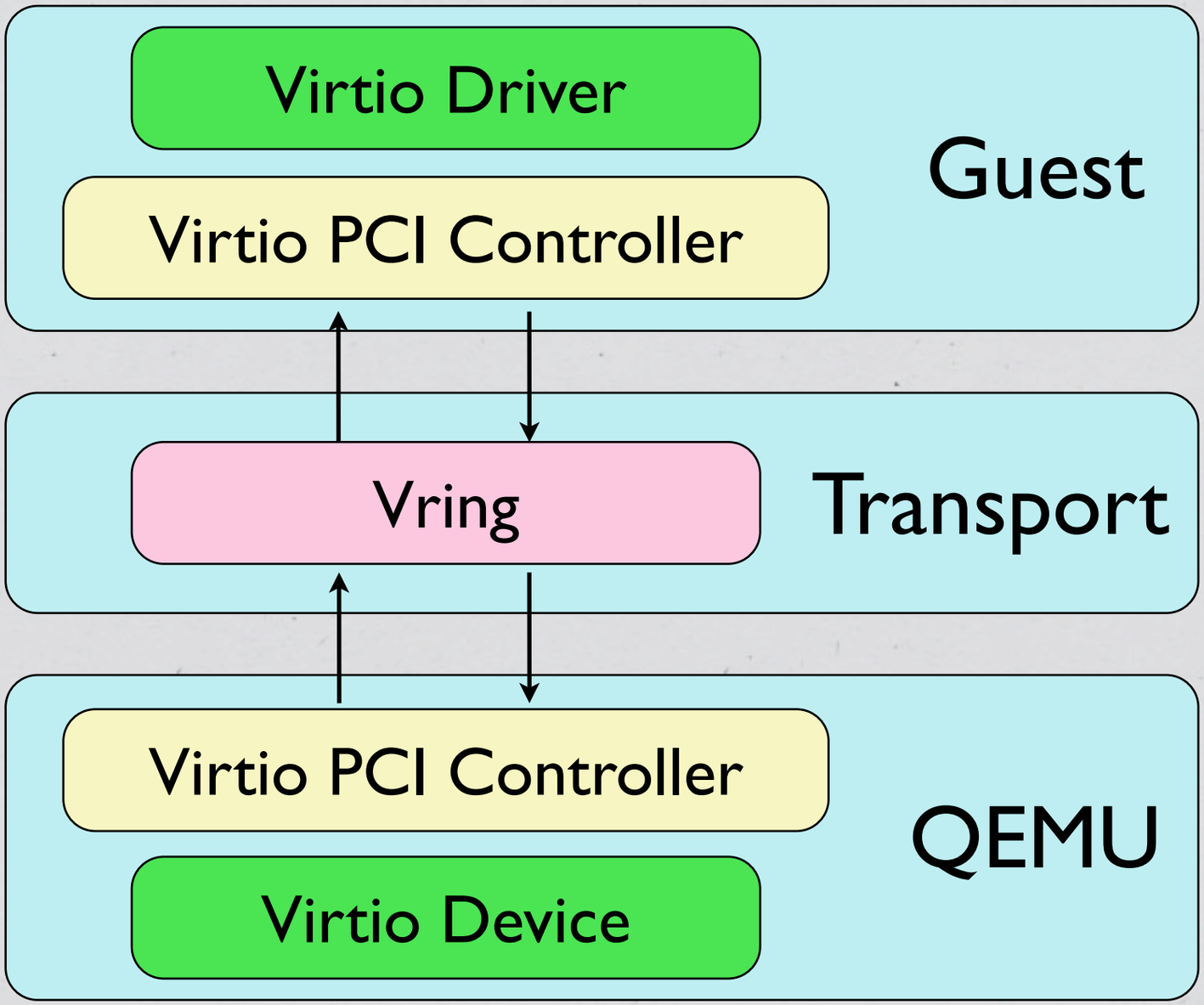
```
static void virtio_[device]_save(QEMUFile *f, void *opaque)
{
   VirtIO[device] *n = opaque;
   virtio_save(&n->vdev, f);
}
static int virtio_[device]_load(QEMUFile *f, void *opaque, int
version_id)
{
    VirtIO[device] *n = opaque;
    virtio_load(&n->vdev, f);
    return 0;
}
```

New Device
data handler

Virtio Device

```
static void virtio_[device]_handle_out(VirtIODevice *vdev,
VirtQueue *vq)
{
    VirtIO[device] *n = to_virtio_my(vdev);
    VirtQueueElement elem1, elem2;
    int K;
    memset(&elem1, 0, sizeof(elem1));
    memset(&elem2, 0, sizeof(elem2));
    virtqueue_pop(vq, &elem1);
    if(memcmp(&elem1, &elem2, sizeof(elem1)))
            printf("something happened!!!\n");
    K = *((int*)elem1.out_sg[0].iov_base) + 1;
    memcpy(elem1.in_sg[0].iov_base, &K, sizeof(int));
    virtqueue_push(vq, &elem1, sizeof(int));
}
```

Architecture



New Driver
device id

Virtio Driver

* Modify kernel-src/include/linux/virtio-ids.h

```
#define VIRTIO_ID_[device]        number
```

Should be the same number as defined in qemu

New Driver

Virtio Driver

```
#include   "linux/virtio.h“
#include  "linux/virtio_ids.h“
#include  "linux/virtio_pci.h“
#include  "virtio.h"
```

```
static struct   virtio_device_id id_table[] = {
        {VIRTIO_ID_[device], VIRTIO_DEV_ANY_ID}, {0}
};
struct virtio_[device]{
        int size;
        void* data;
        void* mapbuf;
        struct semaphore sem;
        struct cdev cdev;
        struct virtio_device *vdev;
        struct virtqueue *vq[2];
} my_cdev;
```

New Driver
define structure

Virtio Driver

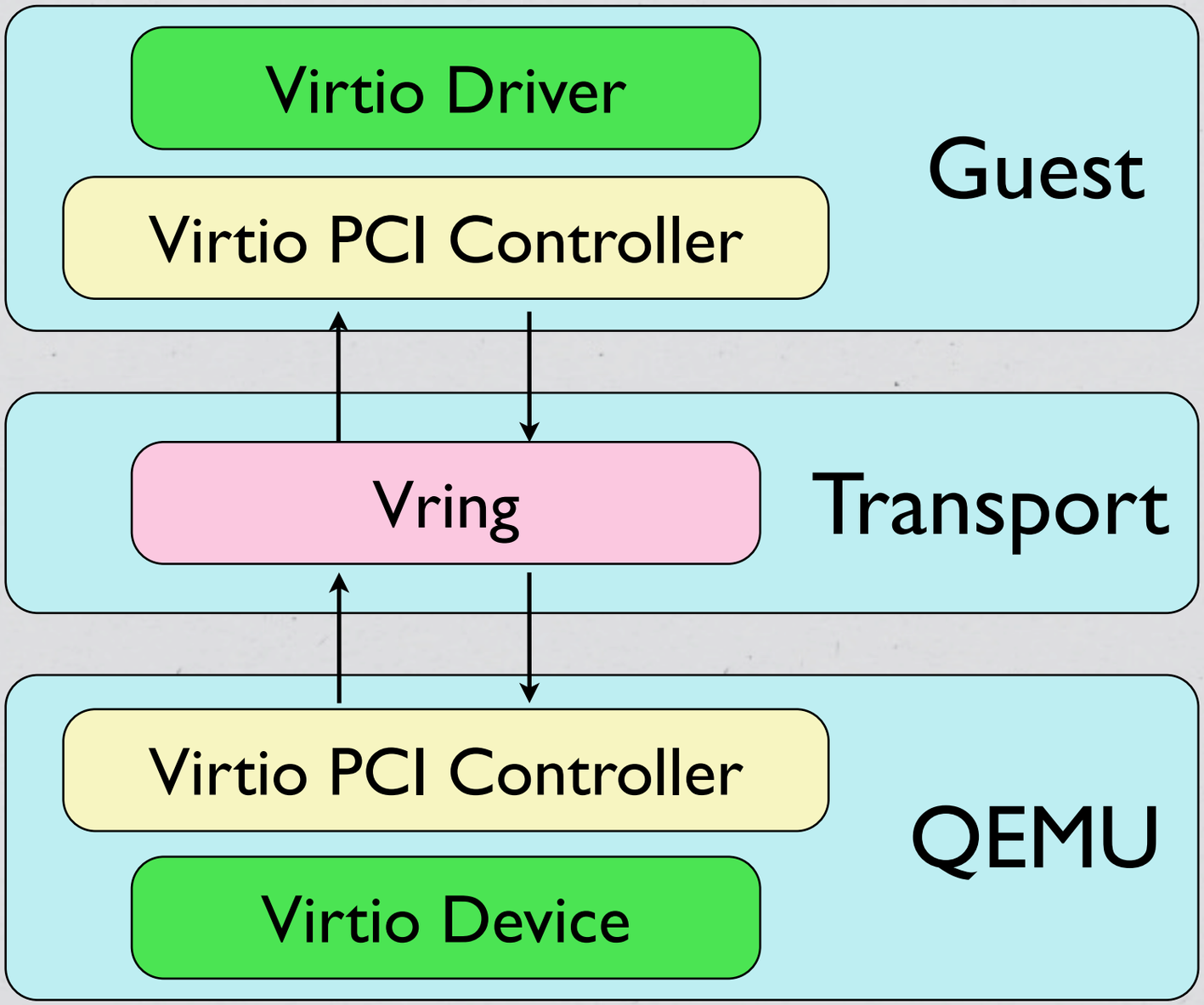
```
static struct virtio_driver virtio_[device]_driver = {
        .feature_table = features,
        .feature_table_size = ARRAY_SIZE(features),
        .driver.name = KBUILD_MODNAME,
        .driver.owner = THIS_MODULE,
        .id_table = id_table,
        .probe = my_probe,
        .remove = __devexit_p(my_remove),
};
```

New Driver

Virtio Driver

* Register & unregister driver - two APIs
* register_virtio_driver(&virtio_[device]_driver);
* unregister_virtio_driver(&virtio_[device]_driver);

Architecture



Communication
driver side

Vring

* Declare necessary parameter

```
char * buffer1, buffer2;
int* A[2] = {buffer1, buffer2};
struct scatterlist sg[2]; /*0 for output, 1 for input*/
struct virtqueue *vq[2];
const char *names[2] = { “[Q1]“ , “[Q2]“ };
vq_callback_t *cb[2] = {NULL, NULL};
```

* Find virtqueue

```
vdev->config->find_vqs(vdev, 2, vq, cb, names);
my_cdev.vq[0] = vq[0];
my_cdev.vq[1] = vq[1];
```

Communication
driver side

Vring

* Send data to qemu

```
sg_init_table(sg, ARRAY_SIZE(sg));
sg_set_buf(&sg[0], buffer1, [count]);
sg_set_buf(&sg[1], buffer2, [count]);
virtqueue_add_buf(myc->vq[0], sg, /*out*/1, /*in*/1, A);
virtqueue_kick(myc->vq[0]);
```

* Get data from qemu

```
virtqueue_get_buf(myc->vq[0], &len);
```

Communication
device side

Vring

* Set virtqueue handler @init function

```
s->vq[0] = virtio_add_queue(&s->vdev, 1024,
virtio_[device]_handle_out1);
s->vq[1] = virtio_add_queue(&s->vdev, 1024,
virtio_[device]_handle_out2);
```

* Declare variable for data @virtqueue handler

```
VirtQueueElement elem1, elem2;
memset(&elem1, 0, sizeof(elem1));
memset(&elem2, 0, sizeof(elem2));
```

Communication
device side

Vring

* VirtQueueElement

```
typedef struct VirtQueueElement
{
    unsigned int index;
    unsigned int out_num;
    unsigned int in_num;
    target_phys_addr_t in_addr[VIRTQUEUE_MAX_SIZE];
    target_phys_addr_t out_addr[VIRTQUEUE_MAX_SIZE];
    struct iovec in_sg[VIRTQUEUE_MAX_SIZE];
    struct iovec out_sg[VIRTQUEUE_MAX_SIZE];
} VirtQueueElement;
```

Communication
device side

Vring

* Pop

```
virtqueue_pop(vq, &elem1);
```

* Check if data exist

```
if(memcmp(&elem1, &elem2, sizeof(elem1)))
  do something;
```

* Put data to elem

```
memcpy(elem1.in_sg[0].iov_base, &data, sizeof(data));
```

* Push data back to guest

```
elem1.in_sg[0].iov_len = sizeof(data);
virtqueue_push(vq, &elem1, sizeof(data));
```

Reference

Virtio: An I/O virtualization framework for Linux

* http://www.ibm.com/developerworks/linux/library/l-virtio/index.html?ca=dgr-lnxw97Viriodth-LX&S_TACT=105AGX59&S_CMP=grlnxw97

virtio: Towards a De-Facto Standard For Virtual IO Devices

* http://www.tdeig.ch/kvm/pasche/32_virtio_Russel.pdf

QEMU & Kernel Source Code

* http://kernel.org/
* http://wiki.qemu.org/