VIRTIO 1.0

Paravirtualized I/O for KVM and beyond

Stefan Hajnoczi <stefanha@redhat.com>

8th February 2014

What we're going to cover

How VIRTIO 1.0 works

You want to:

- Understand paravirtualized I/O
- Design custom devices
- Get familiar before tackling the spec/code

Not covering every VIRTIO 1.0 spec change

See Rusty's linuxconf.au talk: http://goo.gl/wd9Xfp

What is virtio?

"Straightforward, efficient, standard and extensible mechanism for virtual devices"

- Network card, SCSI controller, etc

Designed for situations where accessing device is expensive, device accessing memory is cheap

- Real hardware is the opposite!

Like USB class-compliant devices, a standard driver means compatibility across OSes and hypervisors

What's happening in virtio land?

Community (led by Rusty Russell)

Independent, informal document

QEMU, lguest, Linux, FreeBSD, VirtualBox

OASIS Committee (chaired by Rusty Russell)

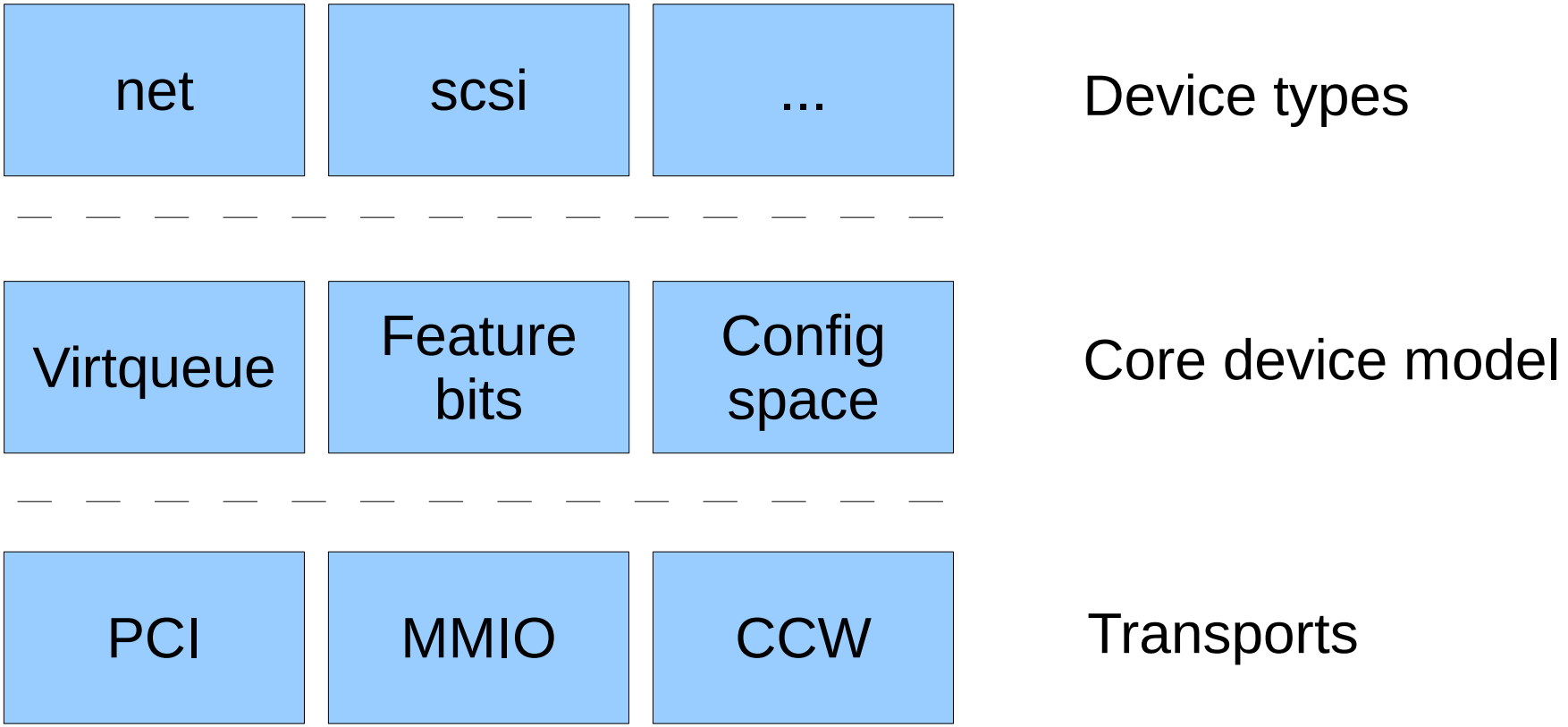
Formal process, formal document

QEMU, lguest, Linux, FreeBSD, VirtualBox,

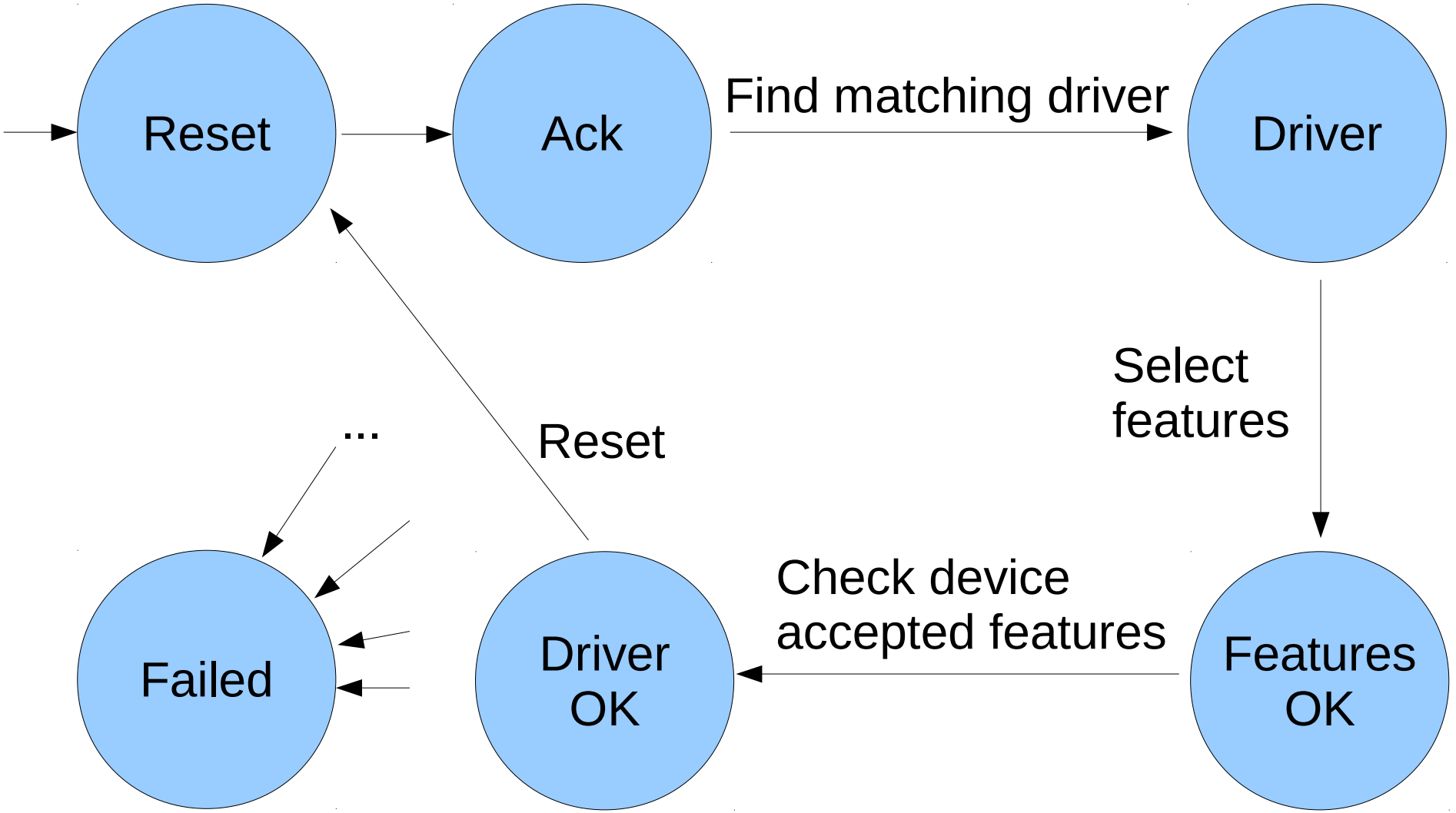
Xen, etc

Virtio architecture

Three layers defined by virtio:



Device lifecycle and device status field



Feature bit negotiation

The feature bit field enables extensibility

- New features can be added to spec in future

Steps for negotiation:

1. Device shows all supported feature bits
2. Driver selects subset of features it supports
3. Driver sets FEATURES_OK in status field
4. Device leaves FEATURES_OK set if ok

Configuration space

Contains device parameters

- Read/write
- 32-bit atomic access (careful with bigger accesses)
- Version counter for consistent >32-bit reads
- No consistent >32-bit writes!
- Device notifies driver via interrupt on update

Consider using a config virtqueue for complex device configuration or error handling.

Virtqueues and the device model

Devices have virtqueues to transfer data buffers

Driver adds buffer, device processes and returns it

Buffers may be:

- Scatter-gather lists (multiple memory regions)
- Handled out-of-order by device, if appropriate

Interrupt notifies driver of buffer completion

Virtqueue programming interface example

```
void
virtqueue_add_sgs(struct virtqueue *vq,
                  struct scatterlist sg[],
                  unsigned int out_sgs,
                  unsigned int in_sgs,
                  void *data, gfp_t gfp);
void *virtqueue_get_buf(
                  struct virtqueue *vq,
                  unsigned int *len);
```


Virtqueue memory layout (aka vring)

Driver allocates vring and configures device with its address:

Descriptor table

Index	Addr	Len	Flags	Next
0				
1				
...				

Available ring

Device Driver

Used ring

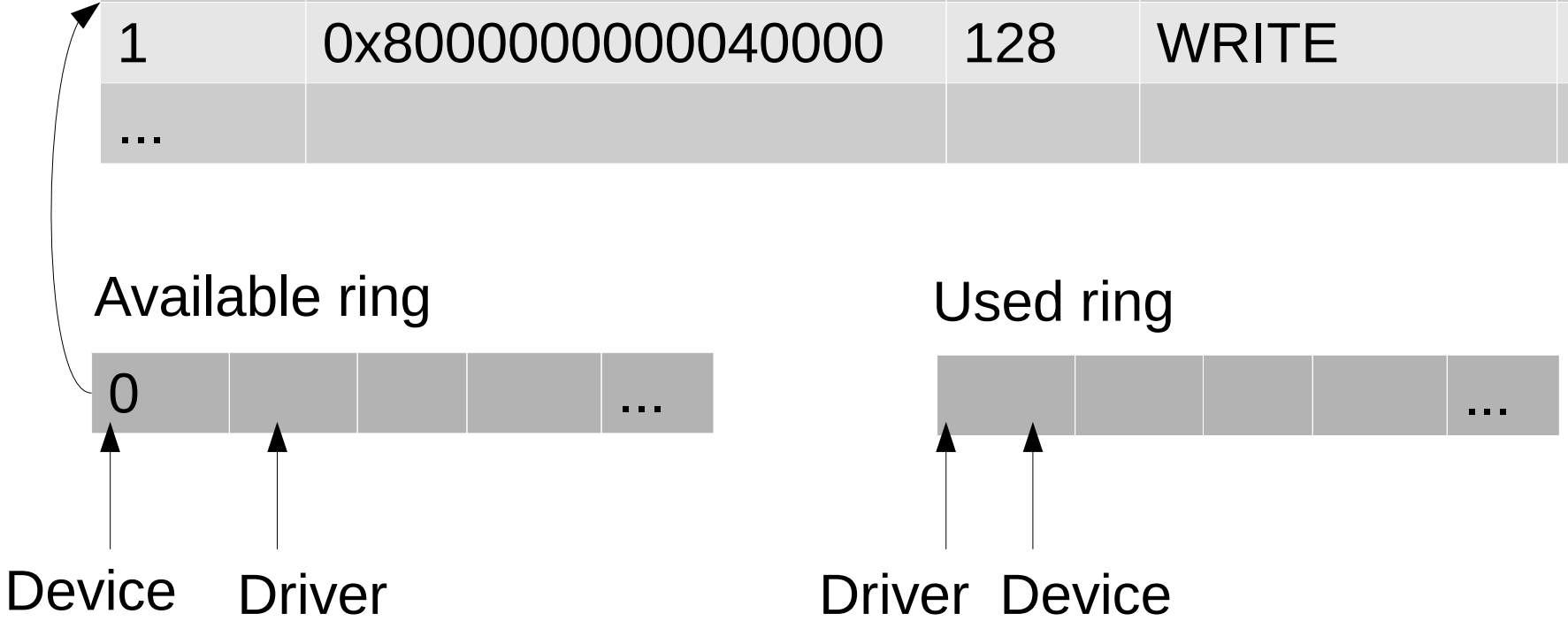
Driver Device

Adding buffers to the vring

Driver puts scatter-gather list into descriptor table, adds head index to available ring, and then kicks device.

Descriptor table

Index	Addr	Len	Flags	Next
0	0x8000000000000000	4096	NEXT	1
1	0x8000000000040000	128	WRITE	0
...				

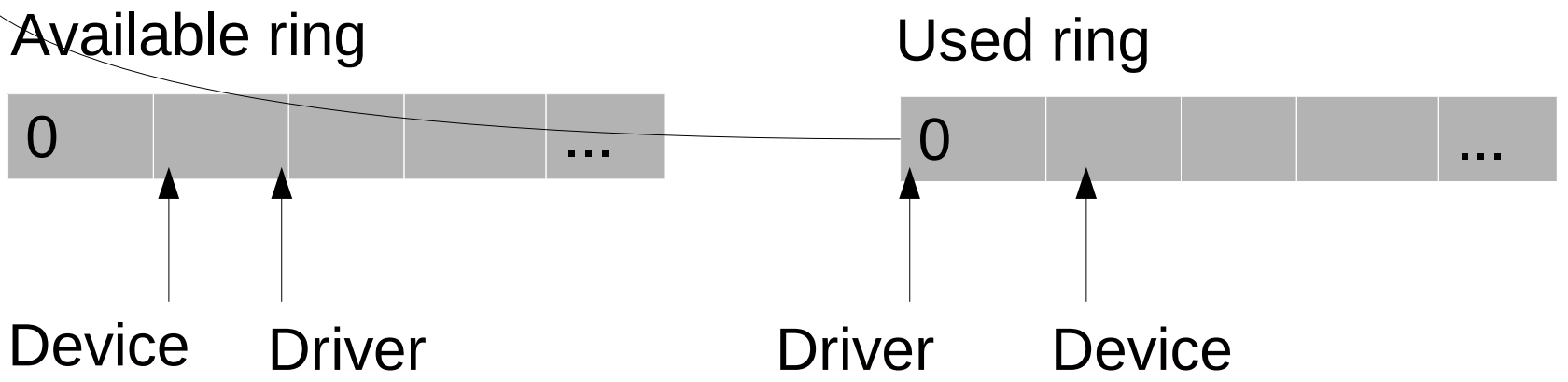


Returning completed buffers to the vring

Device adds head index to used ring and then notifies driver.

Descriptor table

Index	Addr	Len	Flags	Next
0	0x8000000000000000	4096	NEXT	1
1	0x8000000000040000	128	WRITE	0
...				



Example device: virtio-scsi

Virtqueues:

0.Control

1.Events

2.Requests

3.Requests

4.... (multiqueue)

```
struct virtio_scsi_req_cmd {
    u8 lun[8]; le64 id;
    ...
    char cdb[cdb_size];
    char dataout[];
    ...
};
```

Configuration space:

```
struct virtio_scsi_config {
    le32 num_queues;
    le32 seg_max;
    le32 max_sectors;
    le32 cmd_per_lun;
    le32 event_info_size;
    le32 sense_size;
    le32 cdb_size;
    le16 max_channel;
    le16 max_target;
    le32 max_lun;
};
```


More information

VIRTIO 1.0 draft: http://goo.gl/BQ1Kbu

Mailing list: virtio-dev@lists.oasis-open.org

QEMU virtio code: hw/virtio/

Linux virtio driver code: drivers/virtio/

Linux vhost device code: drivers/vhost/

My blog: http://blog.vmsplice.net/

My email: stefanha@redhat.com