Technology Consulting Company
Research, Development &
Global Standard

NVMe Driver for BitVisor

2017-12-05 @ BitVisor Summit 6

Ake Koomsin

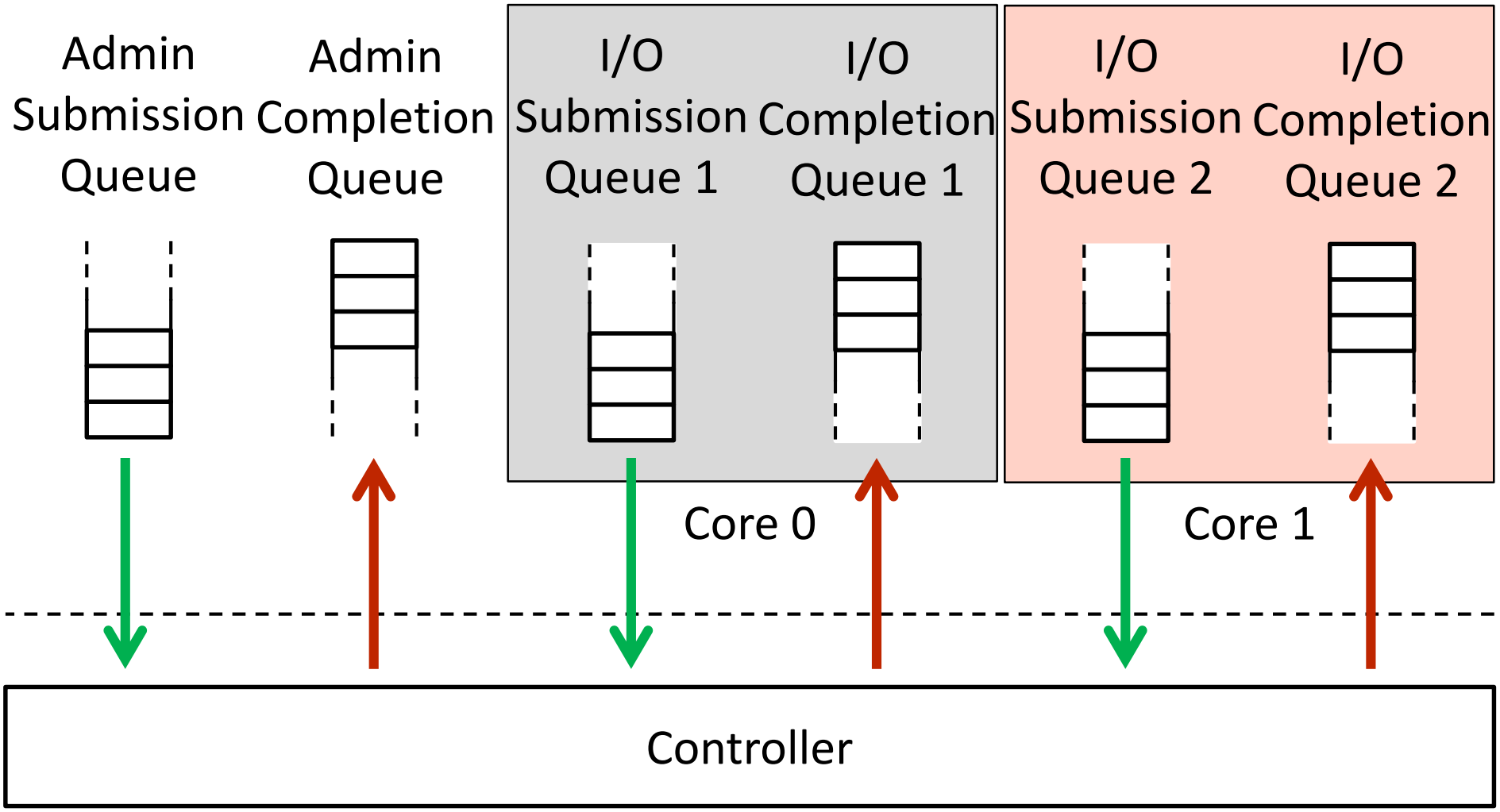
Agenda

- NVMe overview
- NVMe driver implementation
- Using NVMe driver

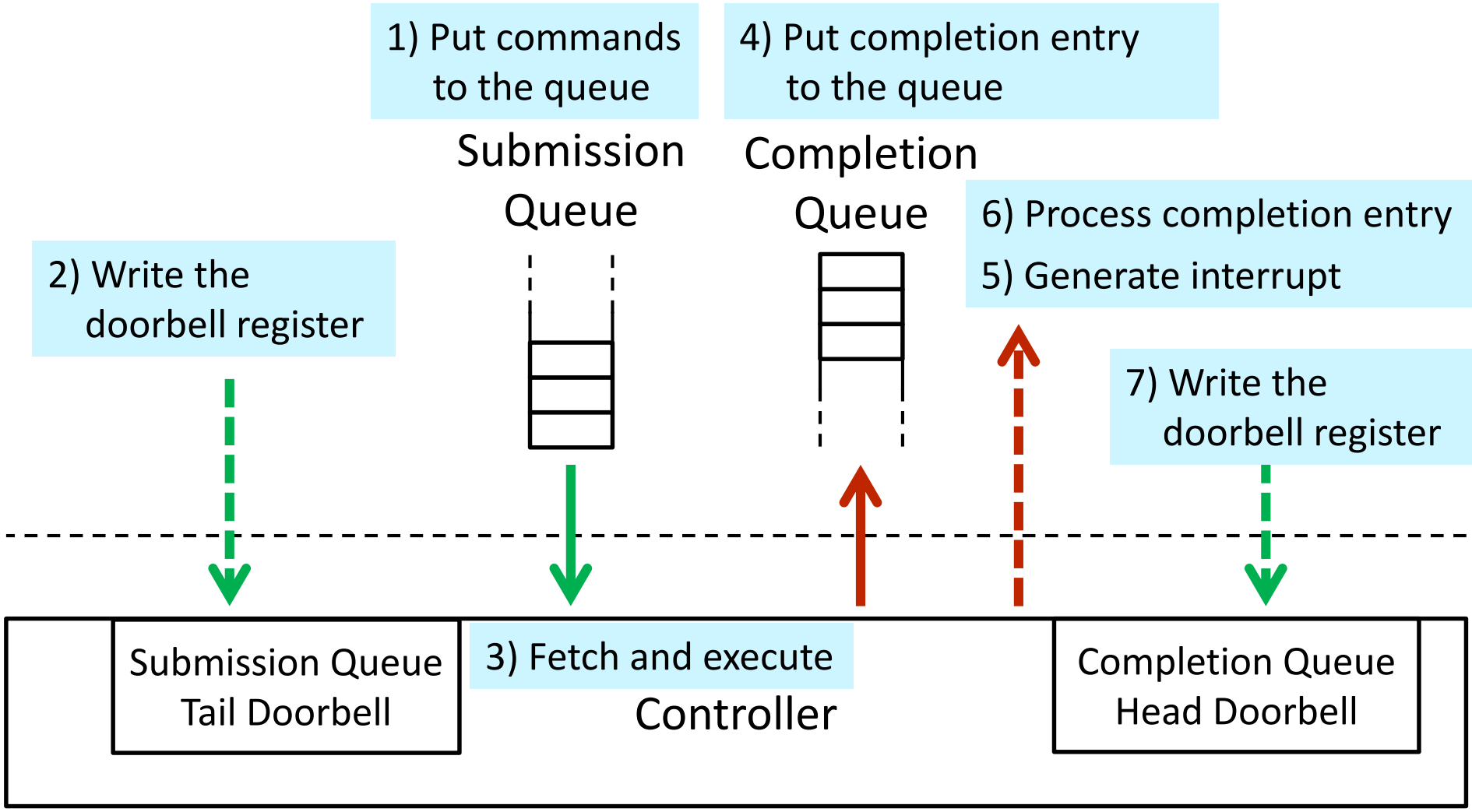
Agenda

- NVMe overview
- NVMe driver implementation
- Using NVMe driver

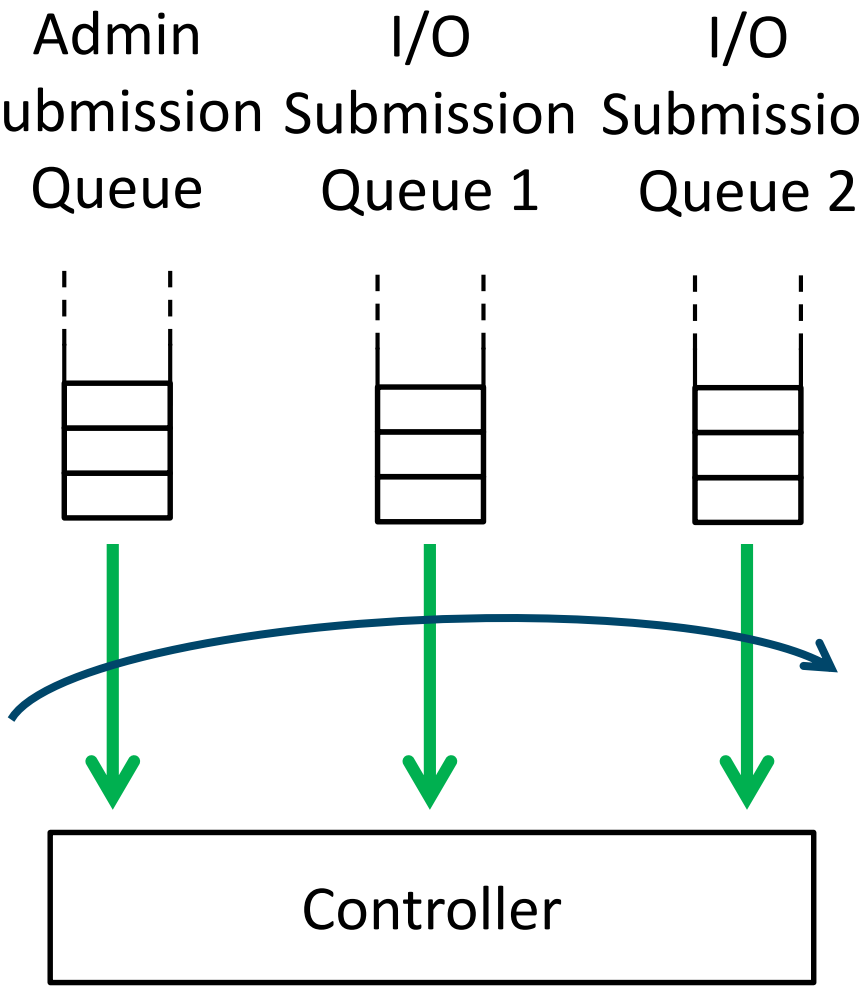
NVMe overview



NVMe command processing



NVMe queue arbitration



- Select one queue at a time
  - Round robin
  - Weighted round robin
- Fetch commands as many as the controller can
  - Execute commands in parallel

NVMe initialization (1)

- Configure Admin Queue
  - Admin Submission Queue Base Address (ASQ) register
  - Admin Completion Queue Base Address (ACQ) register
  - Admin Queue Attribute (AQA) register
    - Number of entries in ASQ and ACQ
- Configure Controller Configuration (CC) register
  - Arbitration mechanism
  - Memory page size
  - Submission/Completion queue entry size
- Start the controller by setting Enable bit in CC to 1
- Wait for readiness

NVMe initialization (2)

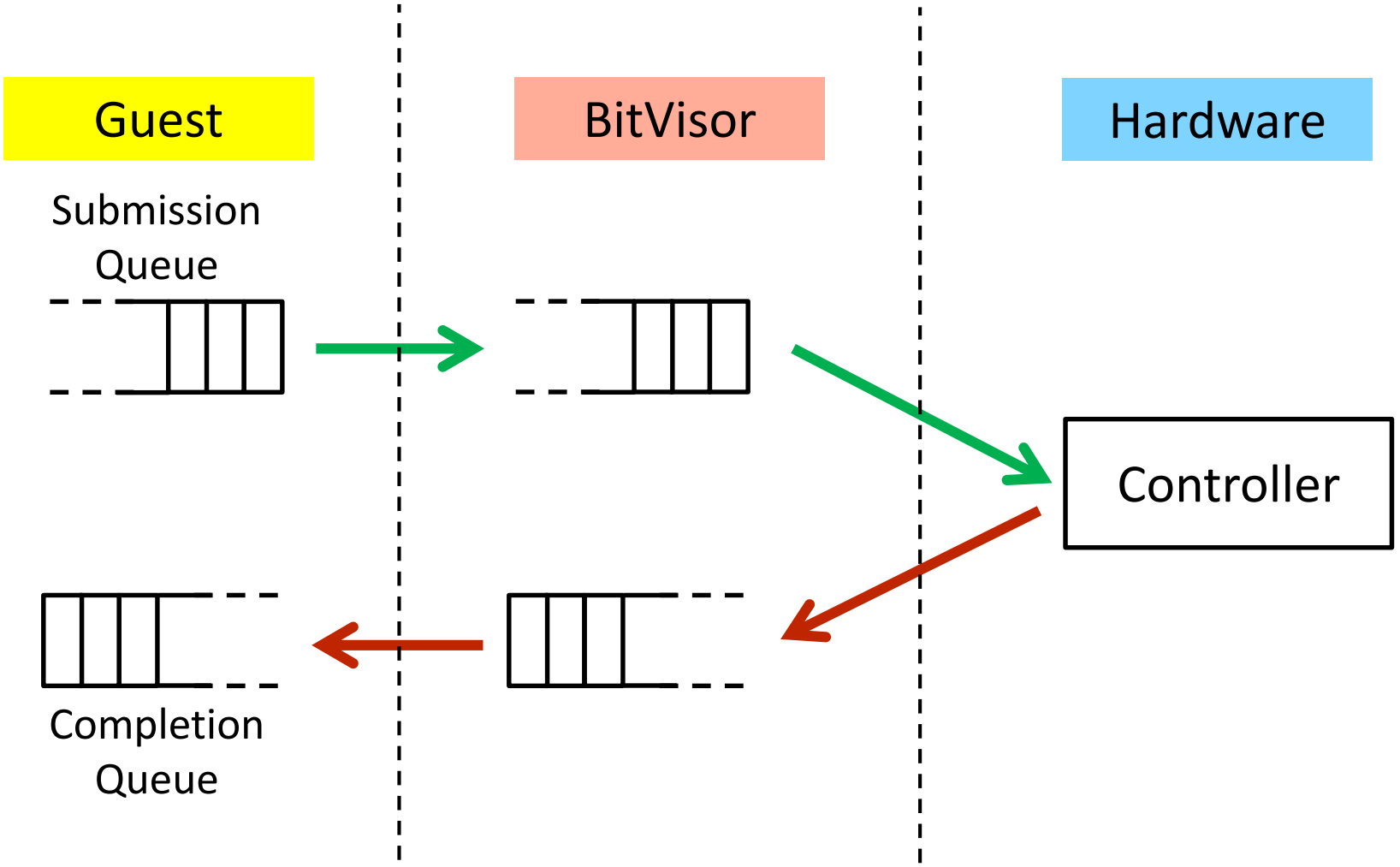
igel

- Submit Identify commands
  - Controller configuration
  - Each namespace information
- Determine number of queues the controller support to using Set Feature commands
- Configure interrupts (MSI/MSI-X)
- Create completion queues by Create I/O Completion Queue commands
- Create completion queues by Create I/O Submission Queue commands
- Ready to go!

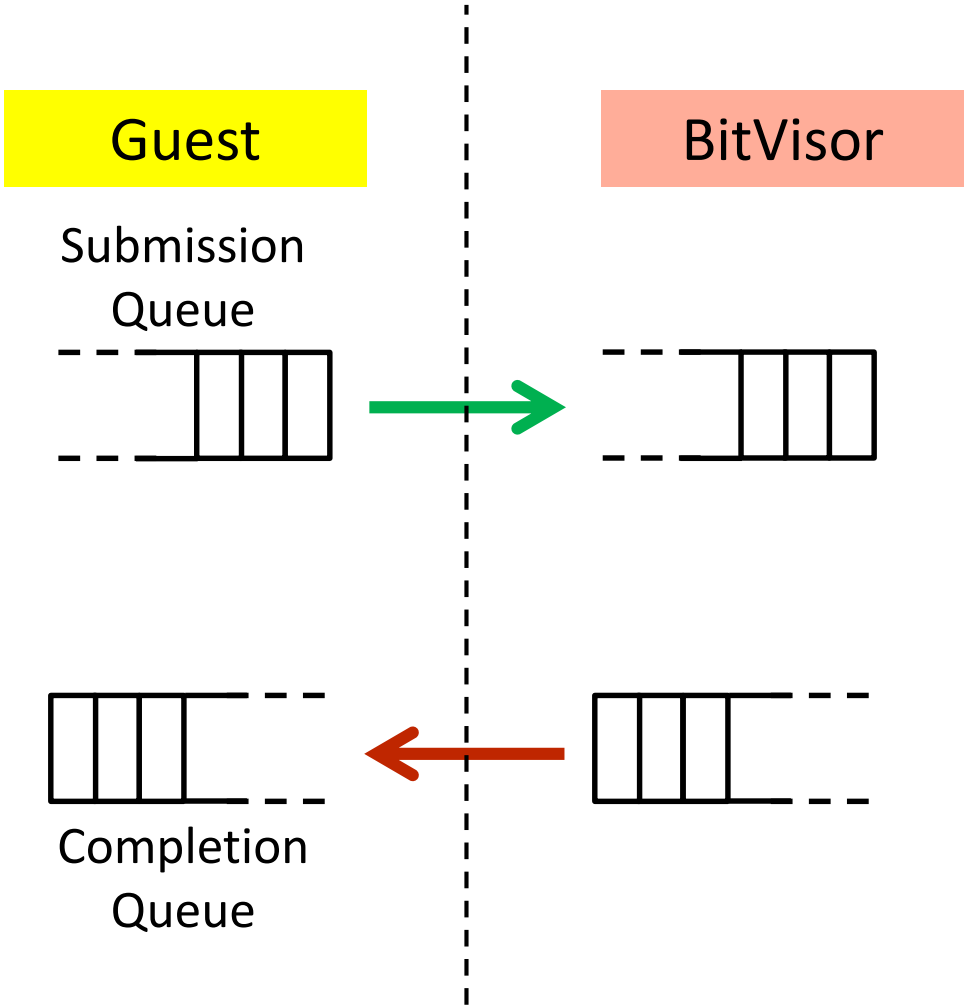
Agenda

- NVMe overview
- NVMe driver implementation
- Using NVMe driver

Implementation concept (1)



Implementation concept (2)



- Intercept doorbell writes for submission queues
- Use external interrupts as the event source for copying completion entries back

NVMe driver implementation (1)

igel

- Intercept Admin Queue related registers
  - Create shadow Admin Queues
  - Create Admin “Request Hub”
- Configure the driver based on value written to the CC register
- After the guest starts the controller, BitVisor submits Identify commands
  - Number of namespaces
  - Each namespace’s LBA size and number of LBAs
  - Additional initialization
  - Note that all guest commands are delayed until we retrieve all information we need

NVMe driver implementation (2)

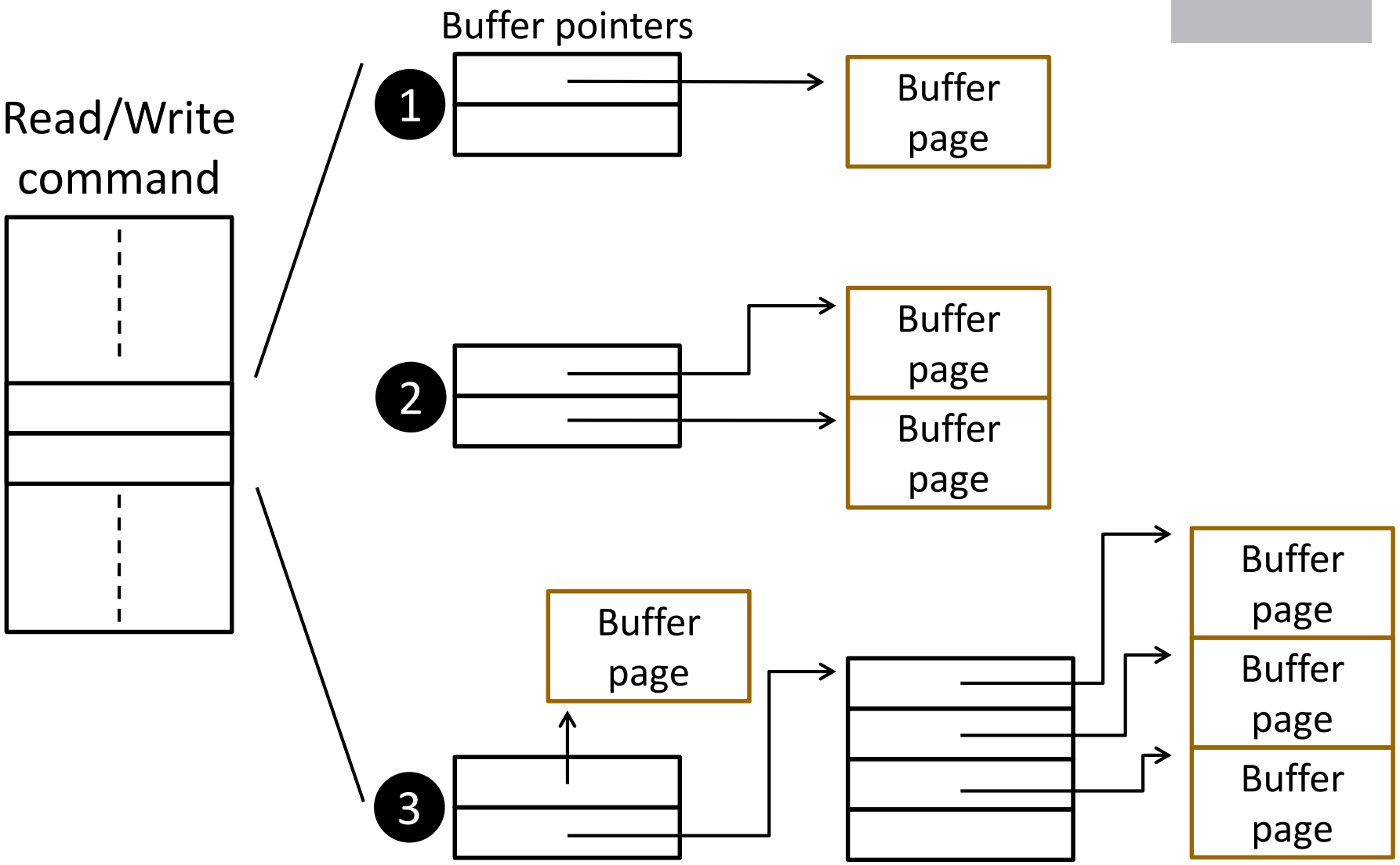
- Intercept Set Feature commands for number of I/O queues the guest is going to use
- Intercept Create I/O Completion Queue commands to create shadow Completion Queues
- Intercept Create I/O Submission Queue commands to create shadow Submission Queues
  - Create I/O “Request Hubs”

NVMe driver implementation (3)

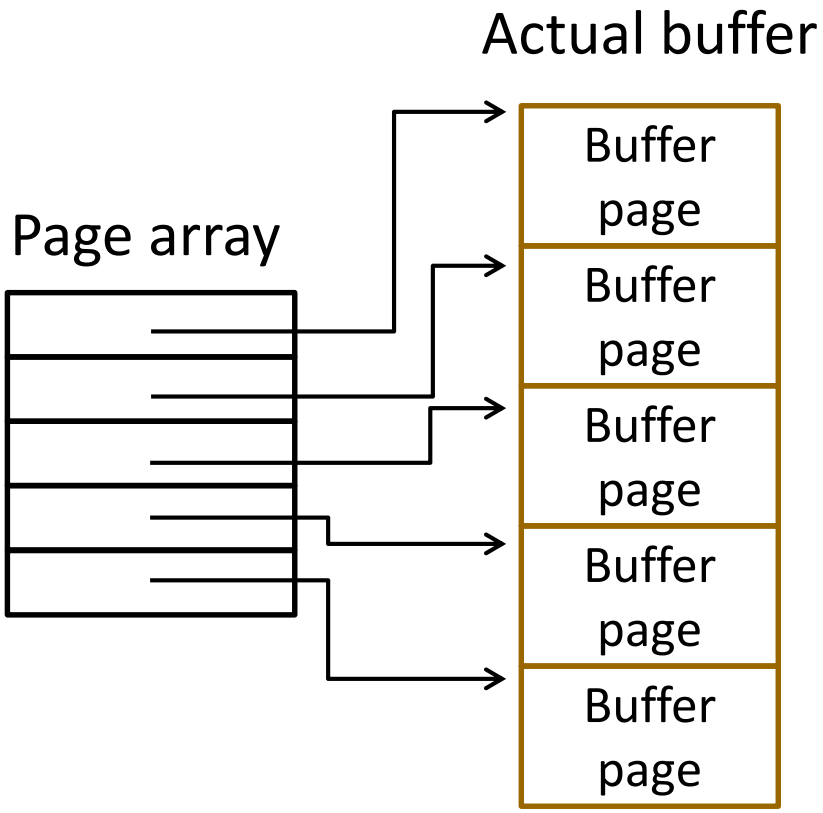
igel

- Request Hubs
  - Multiplex requests from both BitVisor and the guest
  - Currently in time sharing manner
    - Either host requests or guest requests at a time
    - Because of some controller problem

Buffer shadowing (PRP format)



NVMe driver implementation (4)



- Shadow buffer
  - Actual buffer + Page array
  - Copy page by page
    - Don't know whether memory in the guest is continuous or not
- Currently, maximum number of pages is 511
  - Specification allows > 511 pages, we are going to have a list of page array
  - No OS uses more than 511 pages, don't know how to test for correctness

Agenda

- NVMe overview
- NVMe driver implementation
- Using NVMe driver

Using NVMe driver

- We provide functions to interact with the NVMe driver
  - Read/Write NVMe drives
  - Extending the driver
- Can be found in **`nvme_io.h`**
- Still experimental

Using NVMe driver

- I/O descriptor

```
struct nvme_io_descriptor *
nvme_io_init_descriptor (struct nvme_host *host
                         u32 nsid,
                         u16 queue_id,
                         u64 lba_start,
                         u16 n_lbas);

u8
nvme_io_set_phys_buffers (struct nvme_host *host,
                          struct nvme_io_descriptor *io_desc,
                          phys_t *pagebuf_arr,
                          phys_t pagebuf_arr_phys,
                          u64 n_pages_accessed,
                          u64 first_page_offset);
```

Using NVMe driver

- Submitting I/O commands

```
u8
nvme_io_read_request (struct nvme_host *host,
                      struct nvme_io_descriptor *io_desc,
                      void (*callback) (struct nvme_host *host,
                                        void *arg1,
                                        void *arg2,
                                        void *arg3),
                      void *arg1, void *arg2, void *arg3);

u8
nvme_io_write_request (struct nvme_host *host,
                       struct nvme_io_descriptor *io_desc,
                       void (*callback) (struct nvme_host *host,
                                         void *arg1,
                                         void *arg2,
                                         void *arg3),
                       void *arg1, void *arg2, void *arg3);
```

Using NVMe driver

- Install an interceptor during starting up using **`nvme_io_install_interceptor()`** if you need to intercept commands submitted by the guest

```
struct nvme_io_interceptor {

        void *interceptor;

        u8   (*on_init) (void *interceptor);

        void (*on_read) (void *interceptor, ...);

        void (*on_write) (void *interceptor, ...);

        u32 (*on_data_management) (void *interceptor, ...);

        u8   (*can_stop) (void *interceptor);
};
```

Using NVMe driver

- **`interceptor`**
  - The reference to your interceptor object
- **`on_init()`**
  - For initialize your interceptor
  - Get called when the guest is ready to submit I/O commands
- **`on_read()`** and **`on_write()`**
  - Intercept read/write commands
- **`on_data_management()`**
  - Intercept trim deallocation commands
- **`can_stop()`**
  - Try to delay the controller stop event

Using NVMe driver

- Some of utility functions

```
/* At the end of interceptor initialization */
void
nvme_io_start_fetching_g_reqs (struct nvme_host *host);
/* When intercepting a command */
void
nvme_io_pause_guest_request (struct nvme_request *g_req);
void
nvme_io_resume_guest_request (struct nvme_host *host,
                              struct nvme_request *g_req,
                              u8 trigger_submit);
/* When the original command is not needed*/
void
nvme_io_change_g_req_to_flush (struct nvme_request *g_req);
/* When accessing to the request buffer is necessary */
u8 *
nvme_io_req_buf (struct nvme_host *host,
                 struct nvme_request *g_req,
                 u64 lba_offset);
```

Summary

igel

- How NVMe works in general
- How BitVisor NVMe driver intercepts guest's commands
- How can you make use the the driver and extend it

Thank you