Advanced Computer Networks

CS740

Network Virtualization in Data Center Networks (I)

https://pages.cs.wisc.edu/~mgliu/CS740/F25/index.html

Ming Liu
mgliu@cs.wisc.edu

Outline

- Last lecture
  - Load balancing in data center networks (II)

- Today
  - Network virtualization in data center networks

- Announcements
  - Lab2 released today due 11/05/2025 11:59 PM
  - Midterm report due 11/04/2025 11:59 PM

Where we are?

Data Center Network

Multiple communication paths exist when accessing and traversing data center networks!

Physical Connectivity + Networking Architecture (L1, L2, L3)

Where we are?

Data Center Network

The forwarding (destination) address and routing table determine how packets are forwarded!

Addressing and Routing (L4, L5)

Physical Connectivity + Networking Architecture (L1, L2, L3)

Where we are?

Data Center Network

Flow scheduling requires knowing the loading status (congestion degree) of path candidates!

Flow Scheduling (L6, L7)

Addressing and Routing (L4, L5)

Physical Connectivity + Networking Architecture (L1, L2, L3)

Where we are?

Data Center Network

A performant load-balancer design requires per-packet and per-flow processing at line rate with traffic monitoring.

Load balancing (L8, L9)

Flow Scheduling (L6, L7)

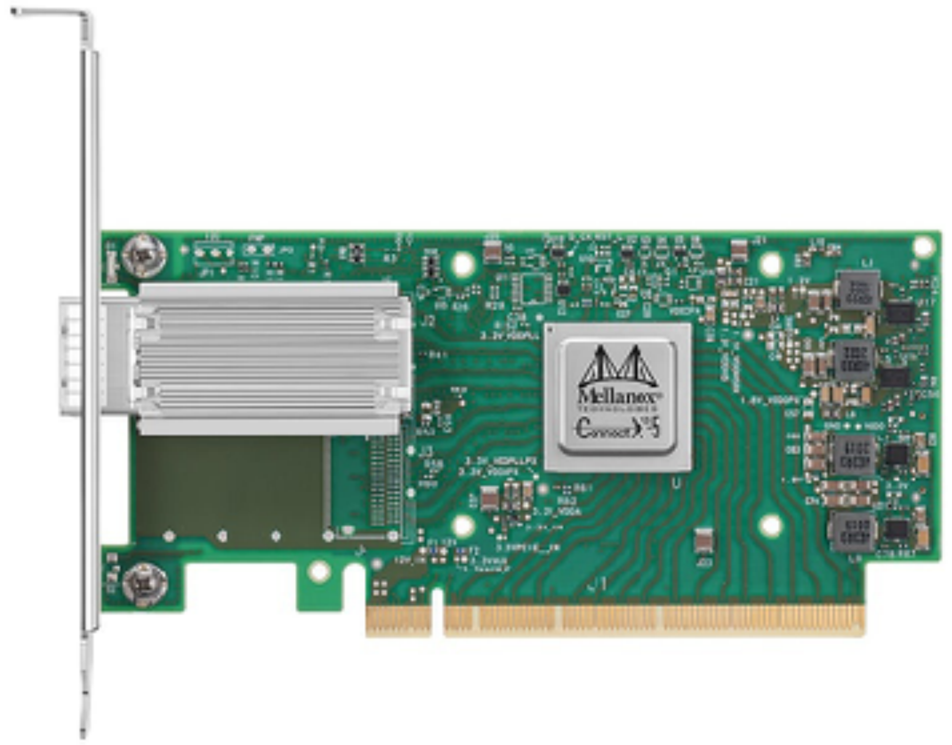
Addressing and Routing (L4, L5)

Physical Connectivity + Networking Architecture (L1, L2, L3)

Where we are?

Data Center Network

A performant load-balancer design requires per-packet and per-flow proc ith traffic monitoring.



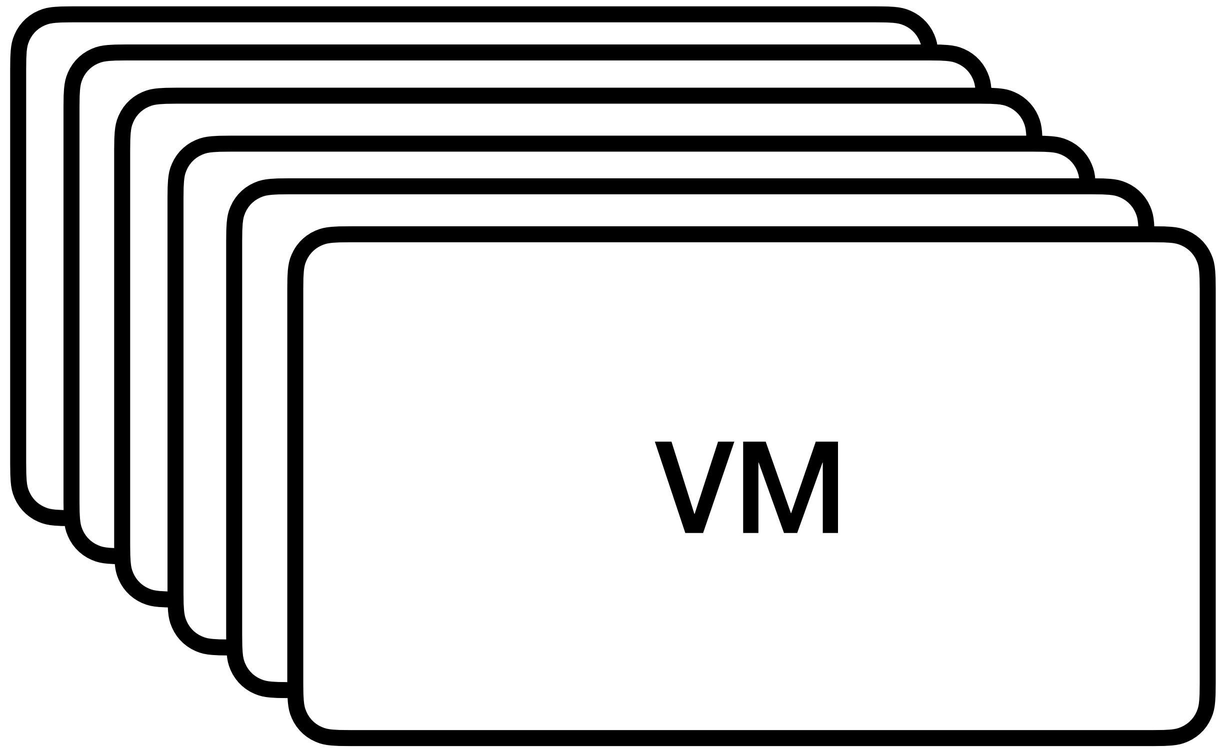
Addressing and Routing (L4, L5)

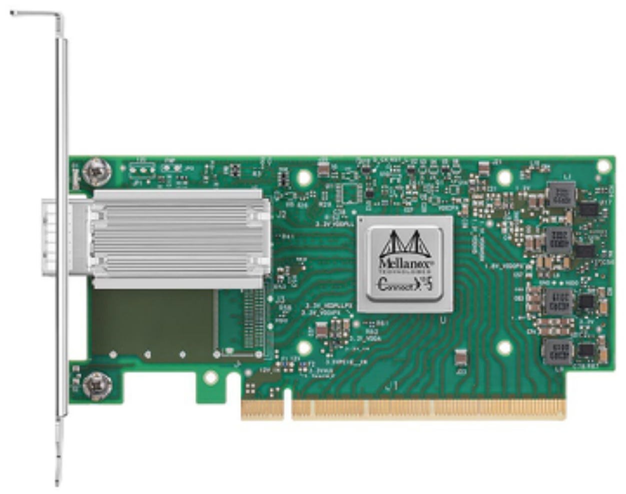
Physical Connectivity + Networking Architecture (L1, L2, L3)

Data centers are managed environments. The predominant system abstraction (user interface) is a virtual machine (VM)!

How network is virtualized?

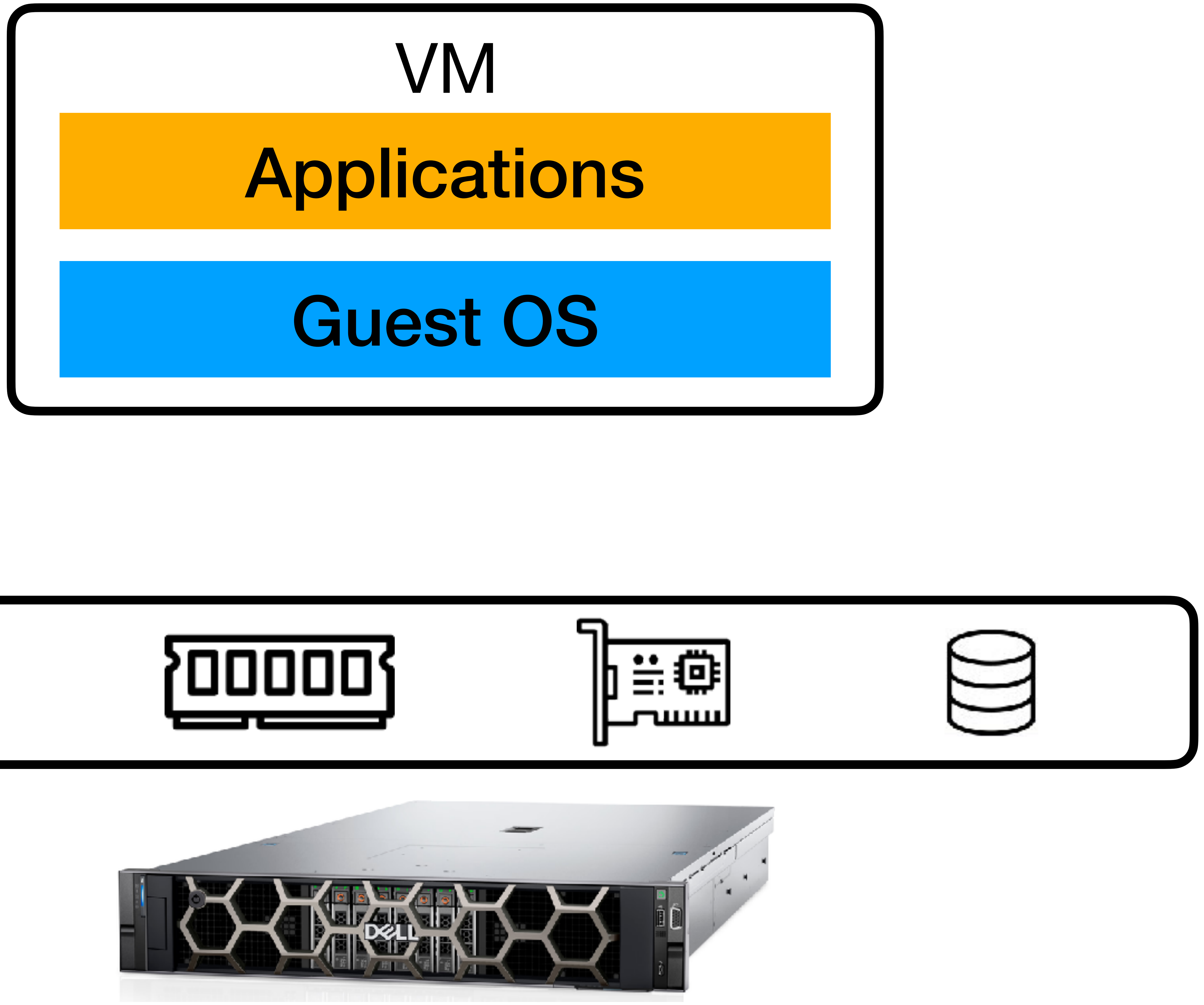
How network is virtualized?





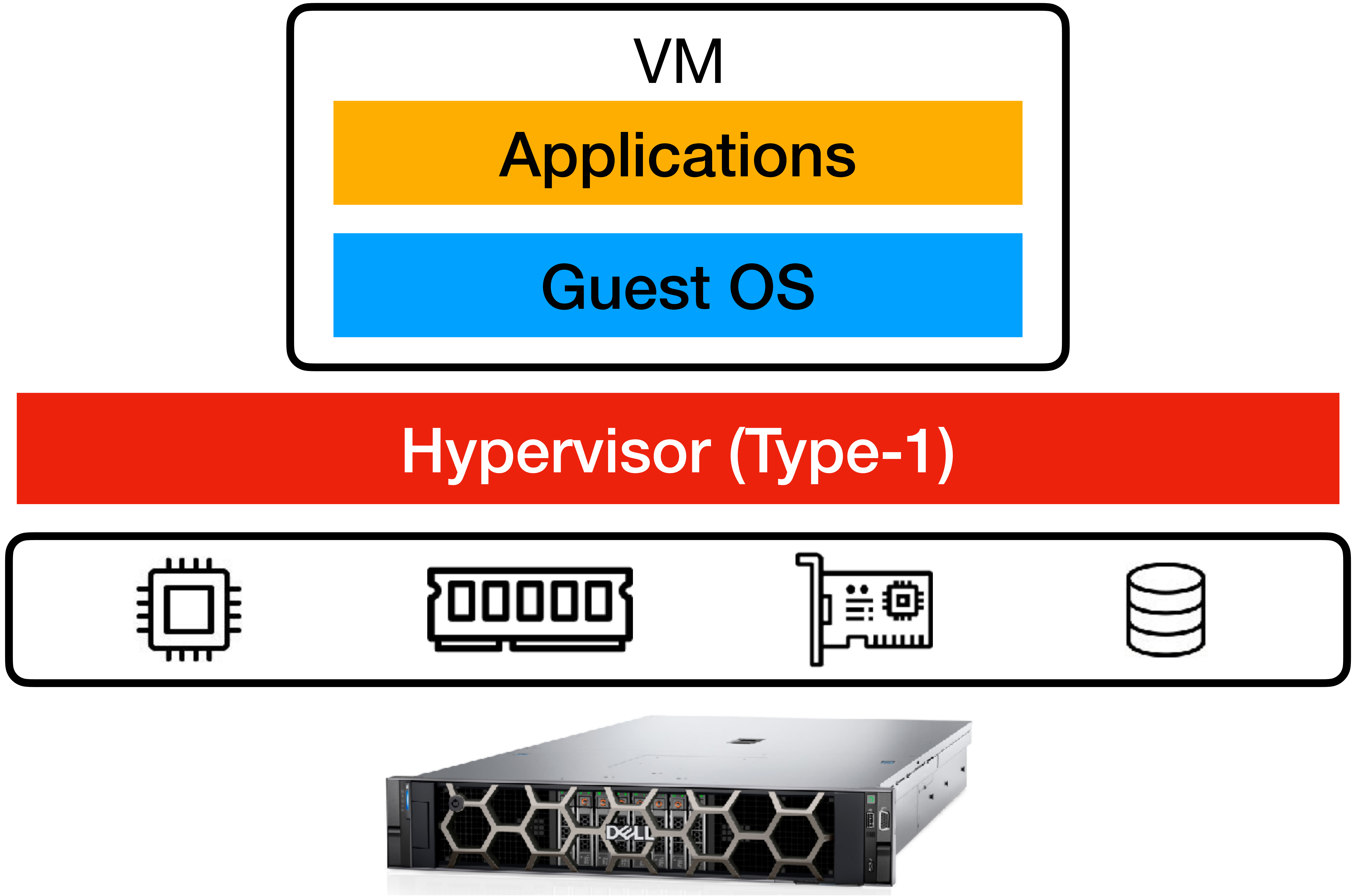
Virtual Machine Overview

- A VM is an isolated computing environment
  - CPU, memory, network, and storage



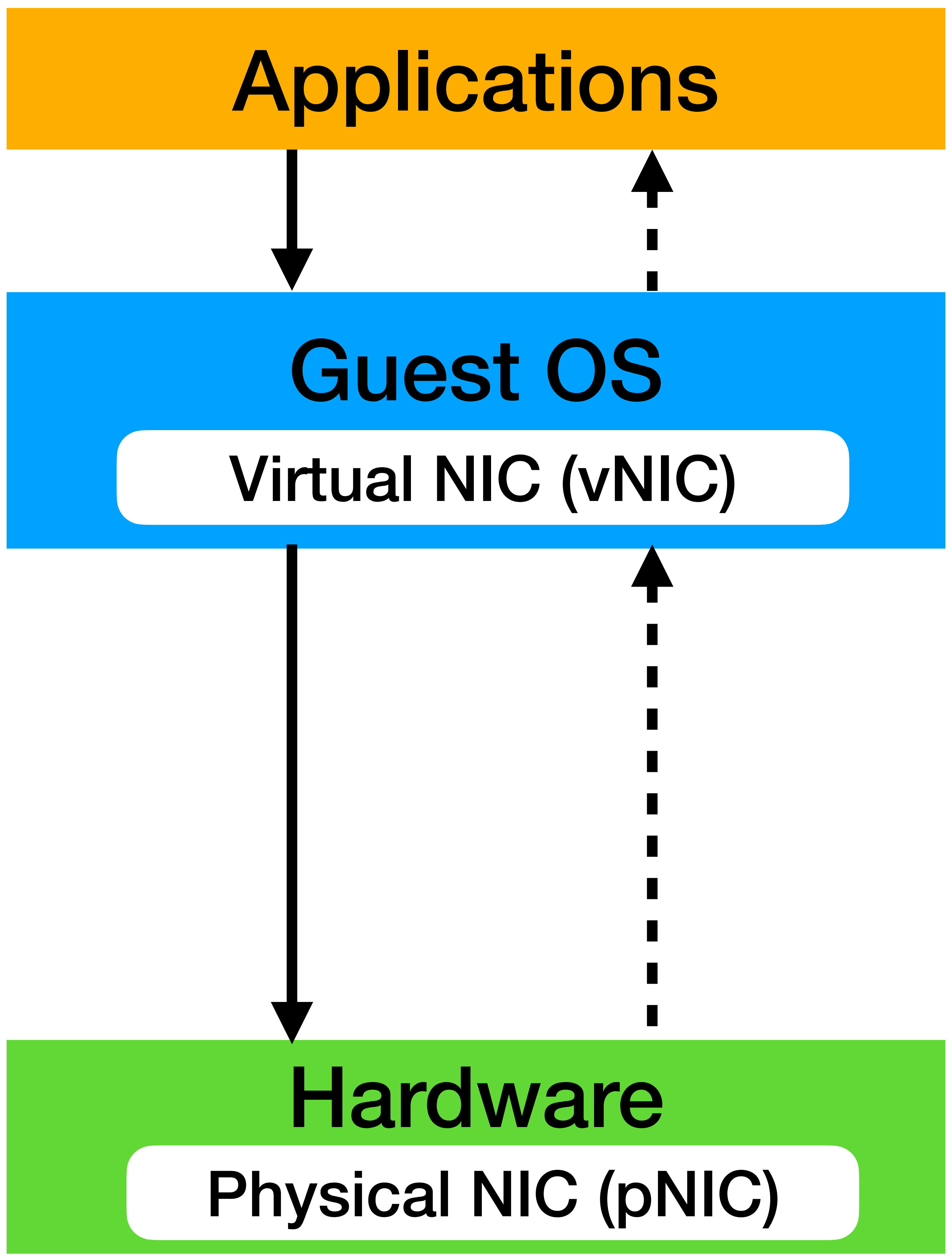
Virtual Machine Overview

- A hypervisor is the privileged software for resource management.
  - E.g., KVM, Xen, Hyper-V, VMware ESXi, …

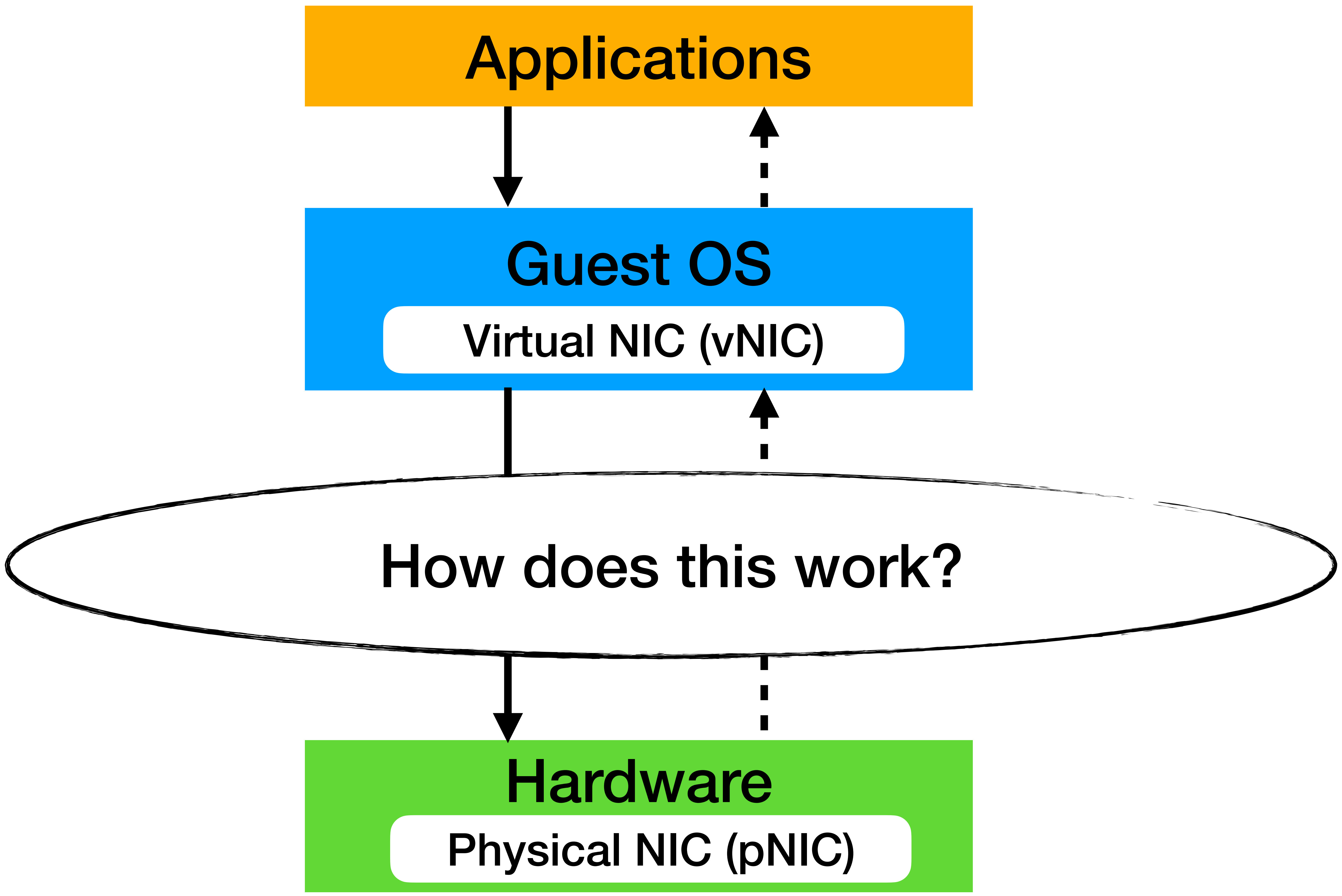


We use KVM and virtio as the case study.

Set Up the Context



Set Up the Context



What is a vNIC?

- The network interface representation of a virtual machine
  - E.g., ifconfig

What is a vNIC?

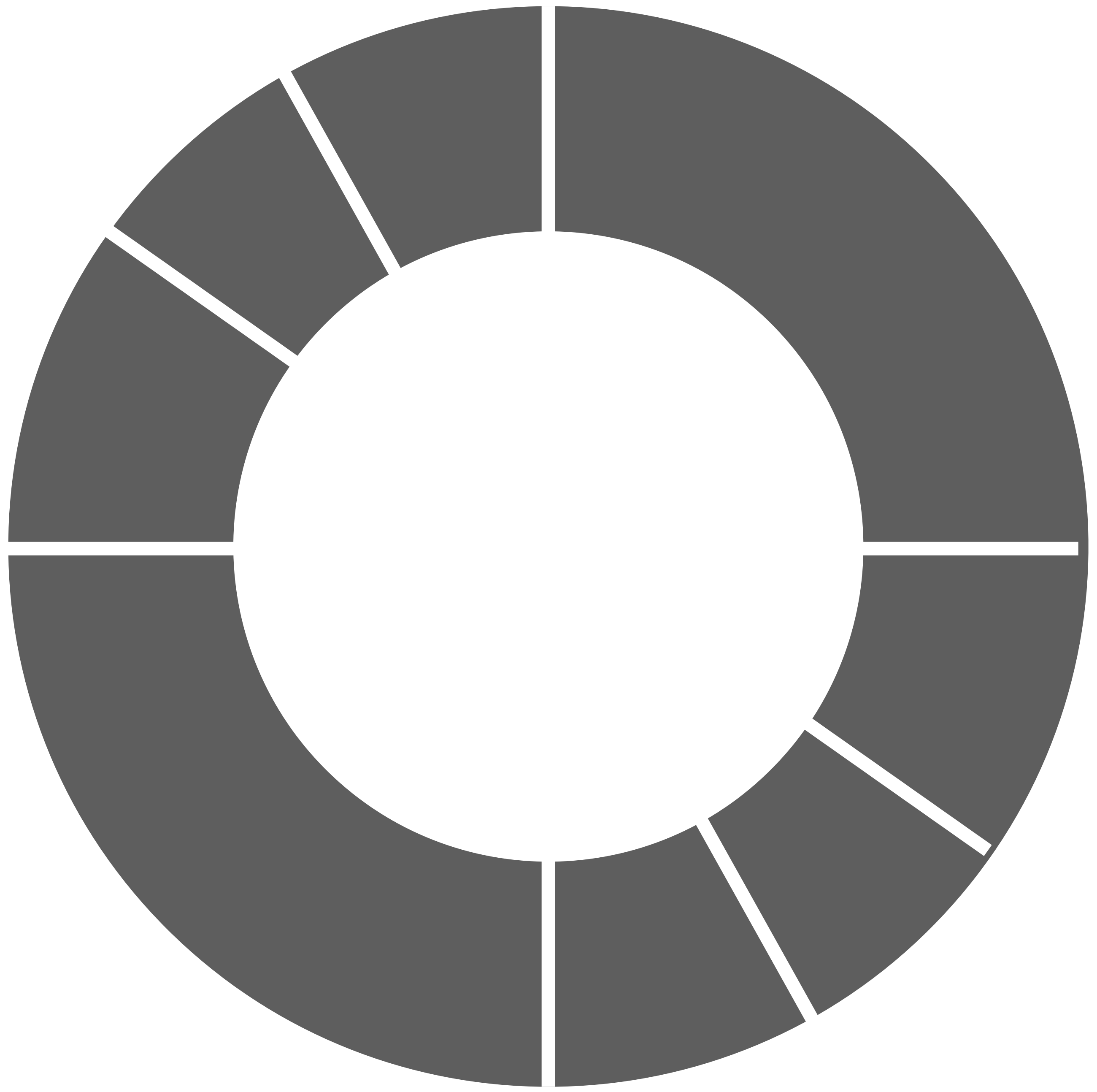
- The network interface representation of a virtual machine
  - E.g., ifconfig

But what is “representation”?

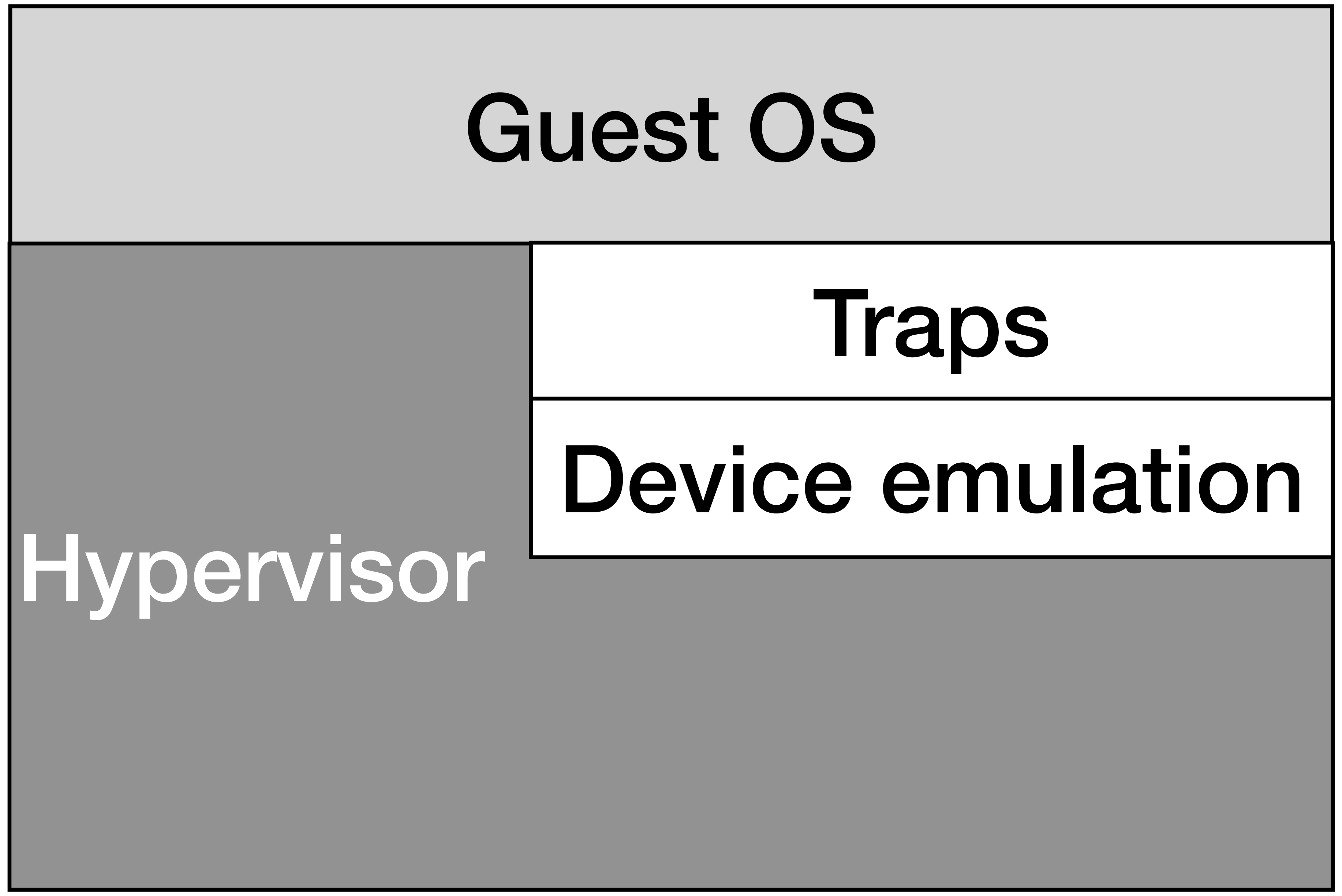
What is a vNIC?

- The network interface representation of a virtual machine
  - E.g., ifconfig

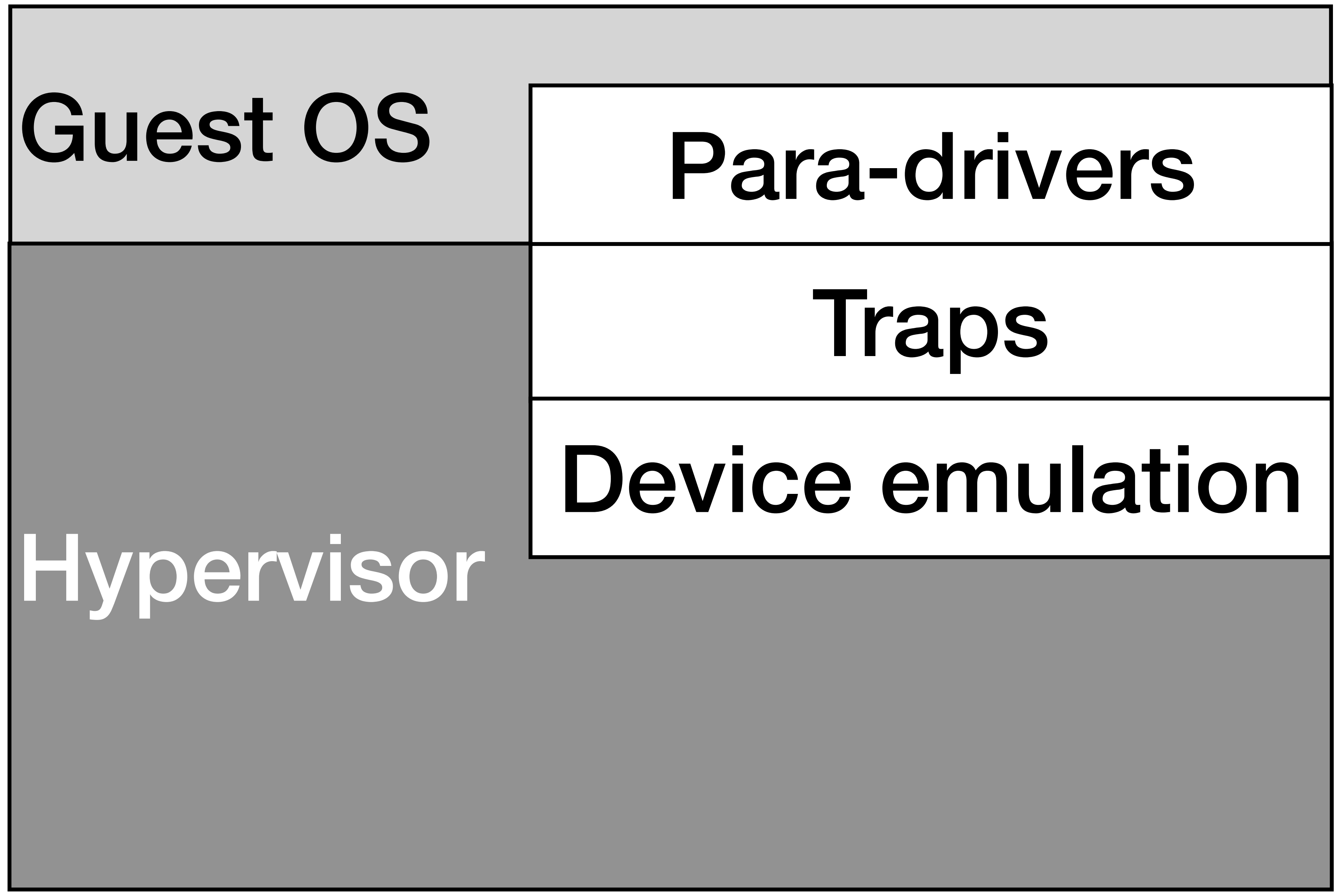
- The communication buffer exposed by the virtual device



Full Virtualization v.s. Para-Virtualization



Full Virtualization



Para-Virtualization

Full Virtualization v.s. Para-Virtualization

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

Full Virtualization

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

Para-Virtualization

Full Virtualization v.s. Para-Virtualization

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

Full Virtualization

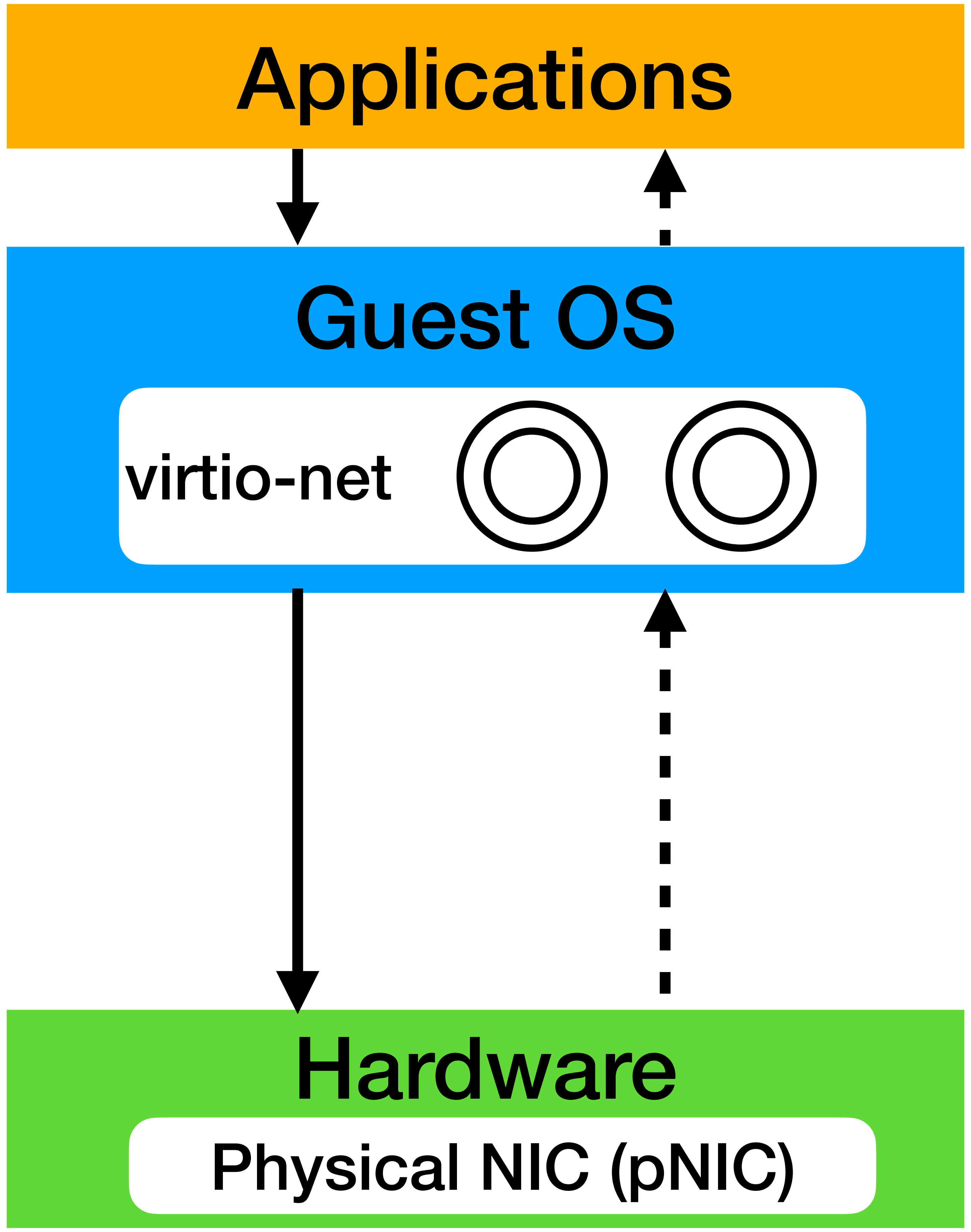
```
void nic_write_buffer(char *buf, int size) {
  vmm_write(NIC_TX_BUF, buf, size);
}
```

Our focus: KVM

Para-Virtualization

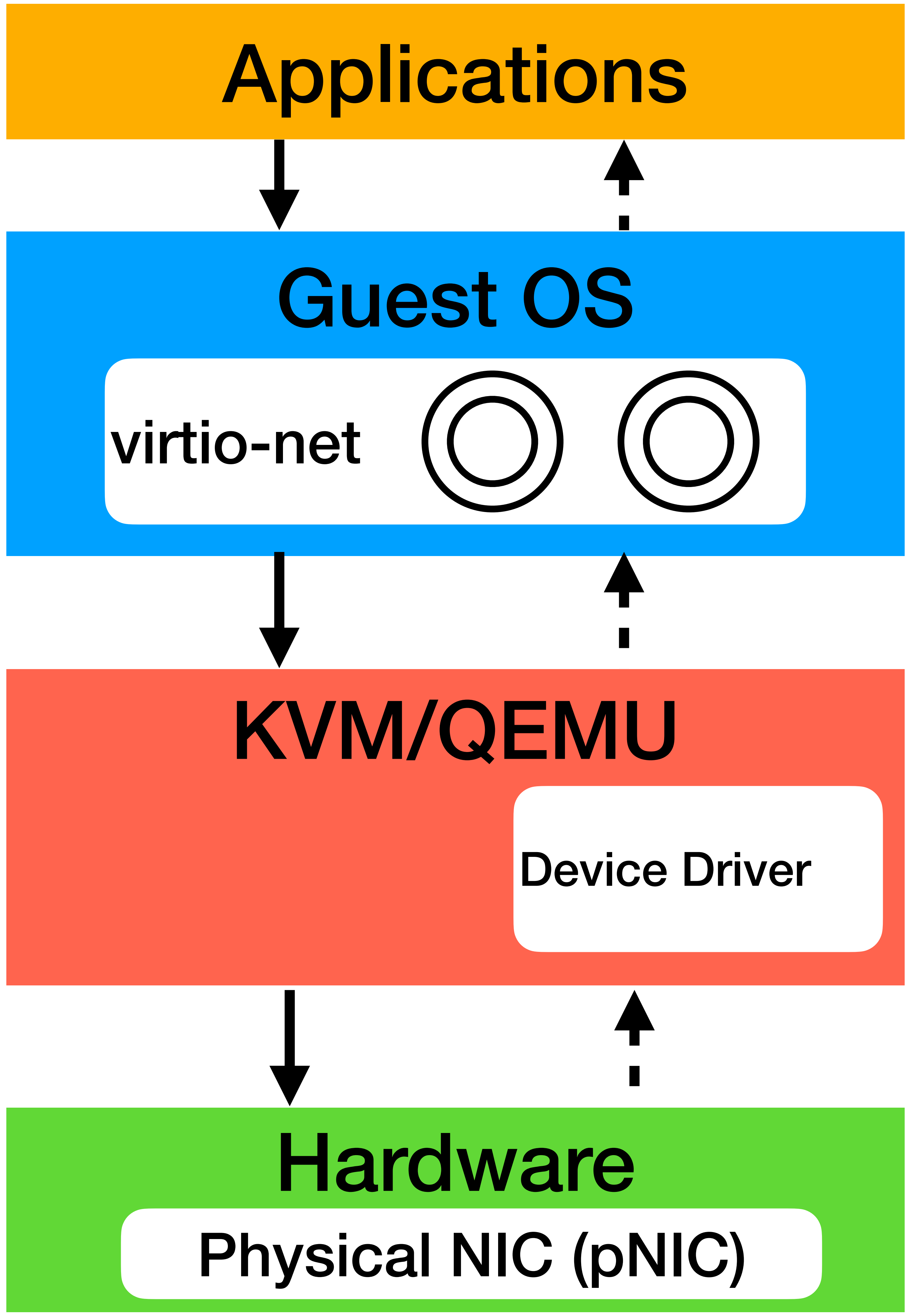
Front-end Virtio

- virtio_net: run in the guest OS and provide “virtuqueue”
  - See linux/include/linux/virtio.h



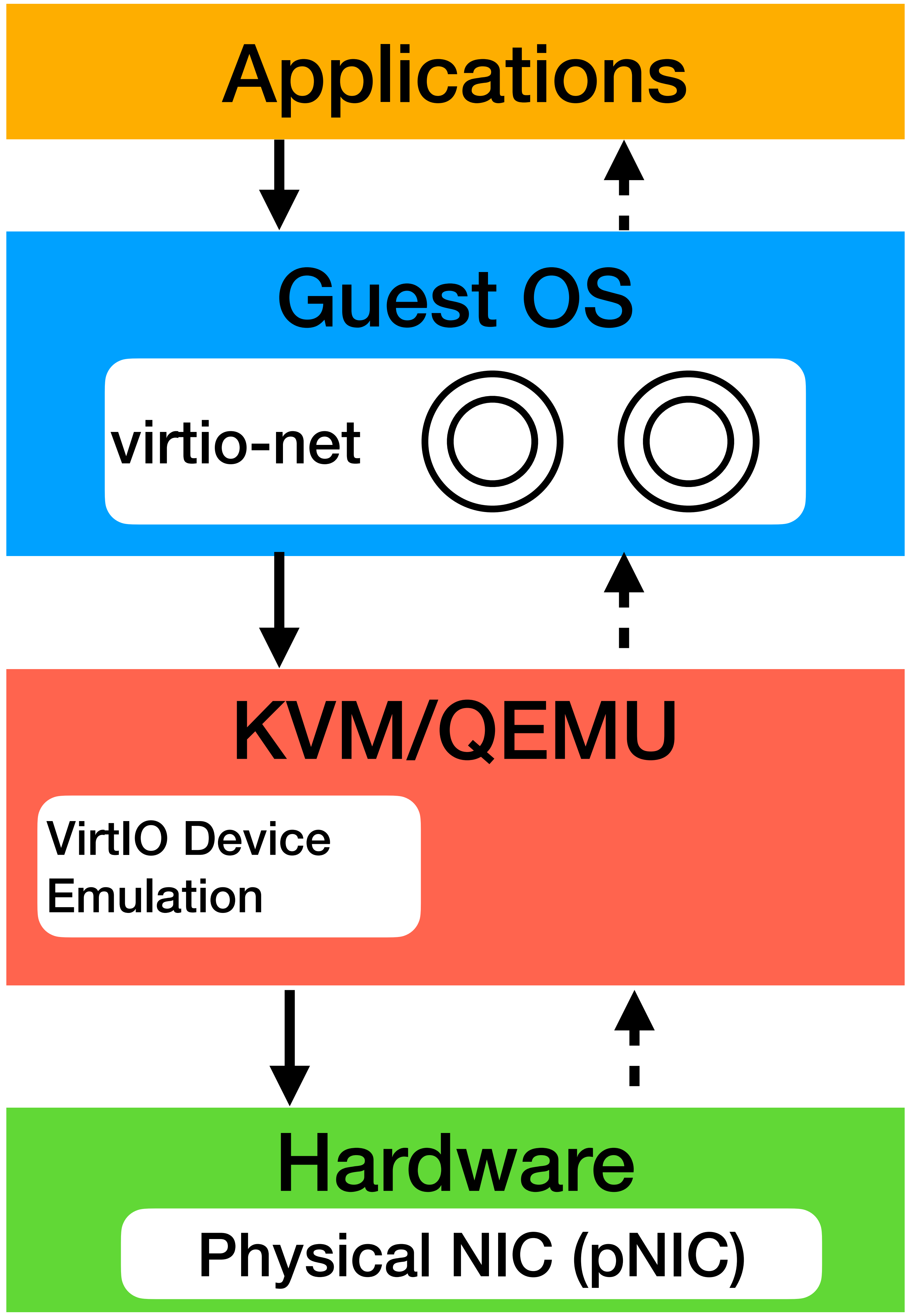

```
/* Register buffers for sending or receiving. */
struct virtqueue {
        struct list_head list;
        void (*callback)(struct virtqueue *vq);
        const char *name;
        struct virtio_device *vdev;
        unsigned int index;
        unsigned int num_free;
        void *priv;
};
```

Front-end Packet Flow



1. “virtio_net” driver in guest OS manipulate data entires in the “virtuqueue”;

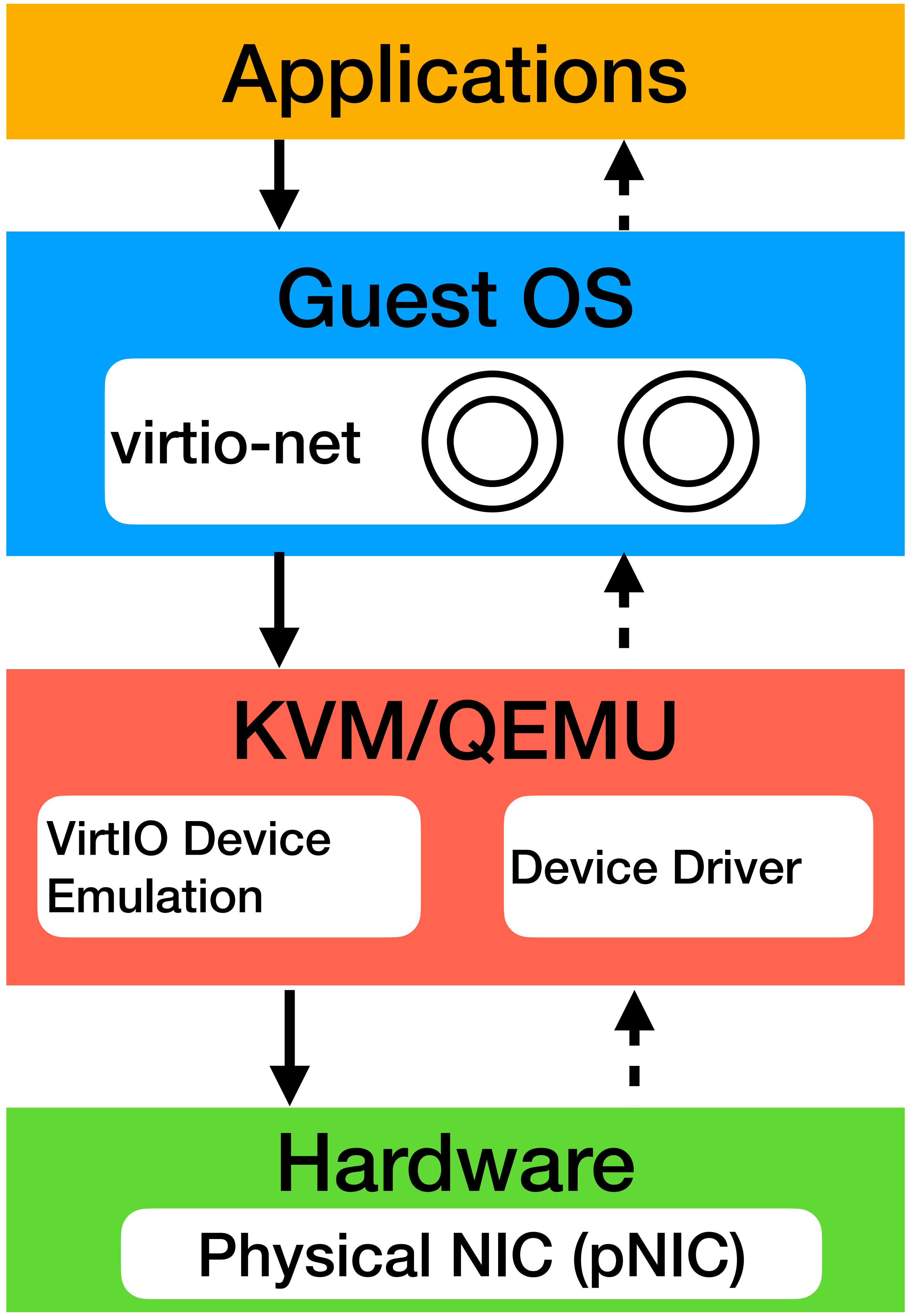
Front-end Packet Flow



1. “virtio_net” driver in guest OS manipulate data entires in the “virtuqueue”;

2. “virtio_net” driver in guest OS calls “virtqueue_kick_prepare()” and “virtqueue_notify()” to interact with the backend device;

Front-end Packet Flow

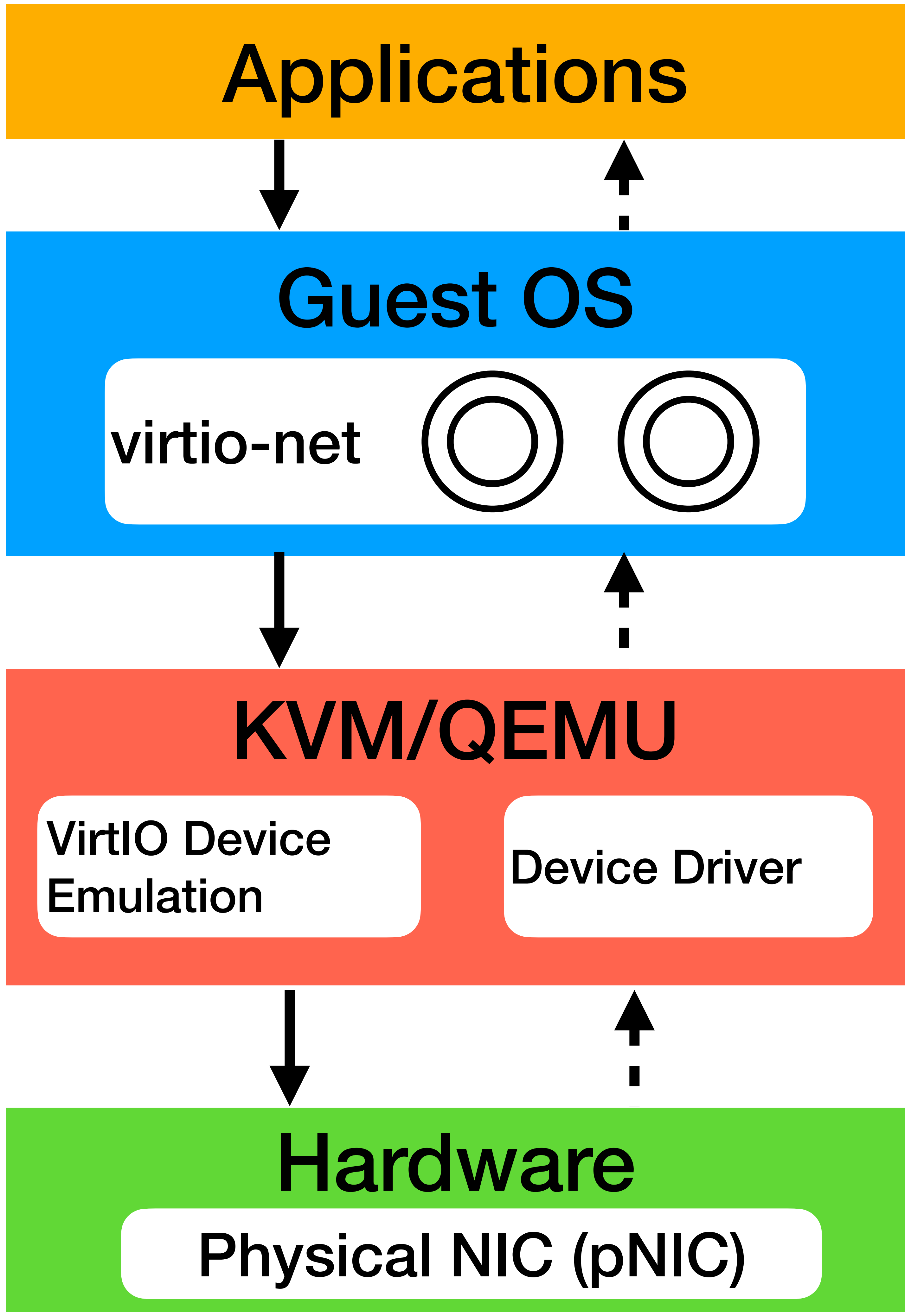


1. “virtio_net” driver in guest OS manipulate data entires in the “virtuqueue”;

2. “virtio_net” driver in guest OS calls “virtqueue_kick_prepare()” and “virtqueue_notify()” to interact with the backend device;

3. The back-end device driver pops requests, processes it, and writes to the device buffer;

Front-end Packet Flow



1. “virtio_net” driver in guest OS manipulate data entires in the “virtuqueue”;

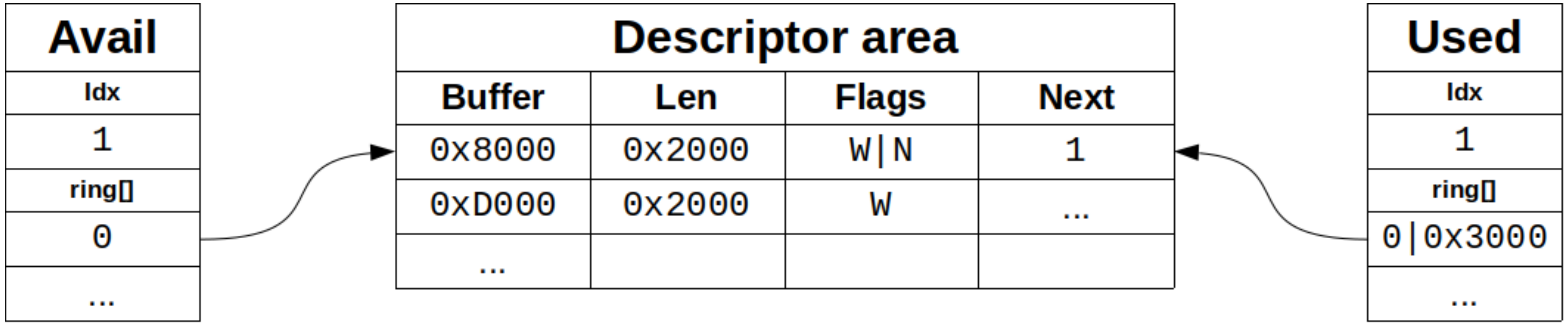
2. “virtio_net” driver in guest OS calls “virtqueue_kick_prepare()” and “virtqueue_notify()” to interact with the backend device;

3. The back-end device driver pops requests, processes it, and writes to the device buffer;

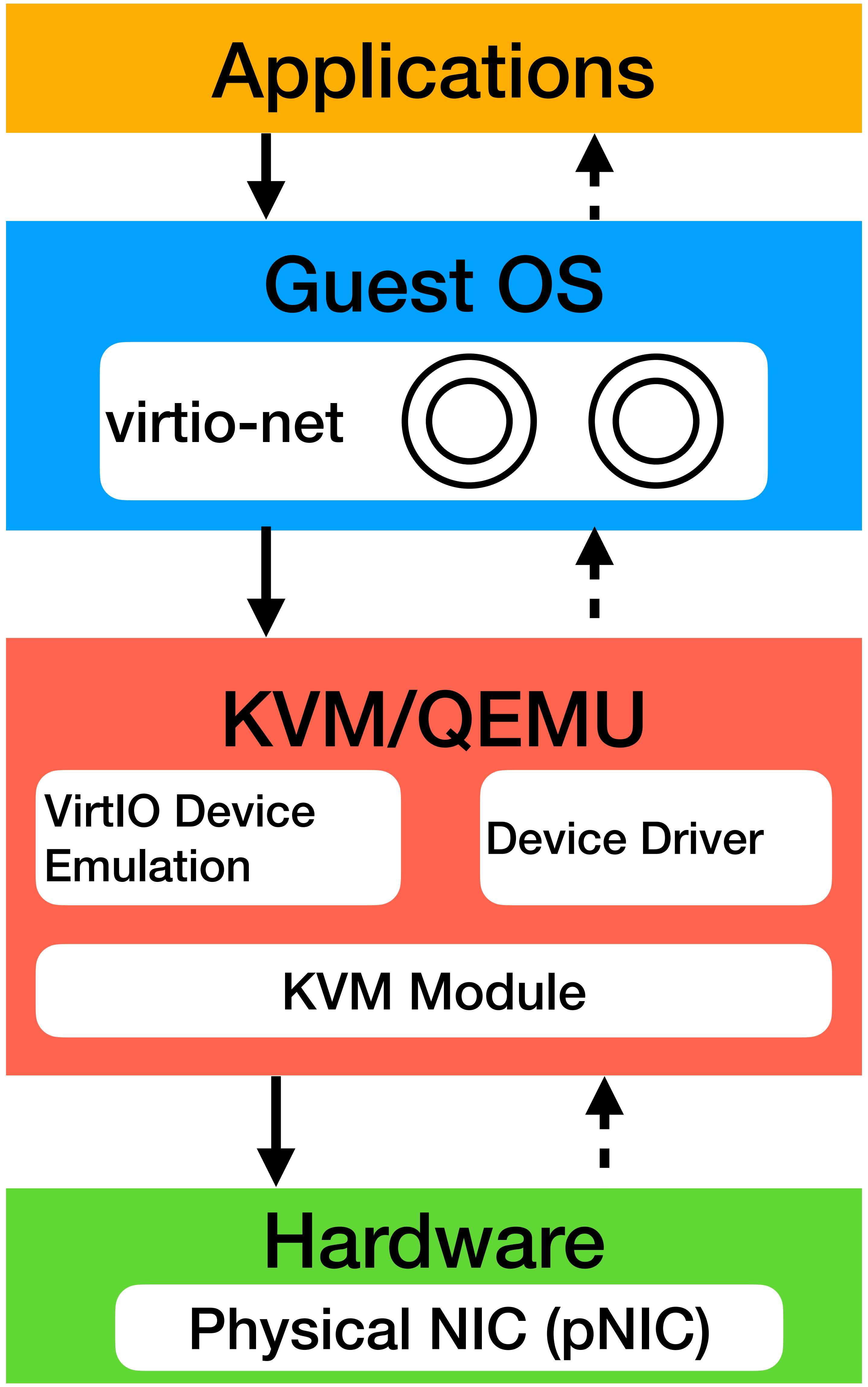
4. The context return to the guest OS to indicate the completion;

Virtqueue Details

- A channel between front-end and back-end
  - Part of the memory of the guest OS
- Three parts
  - Descriptor area: describing buffer status
  - Driver area: data supplied by the front-end driver to the back-end device
  - Device area: data supplied by the backend-end device

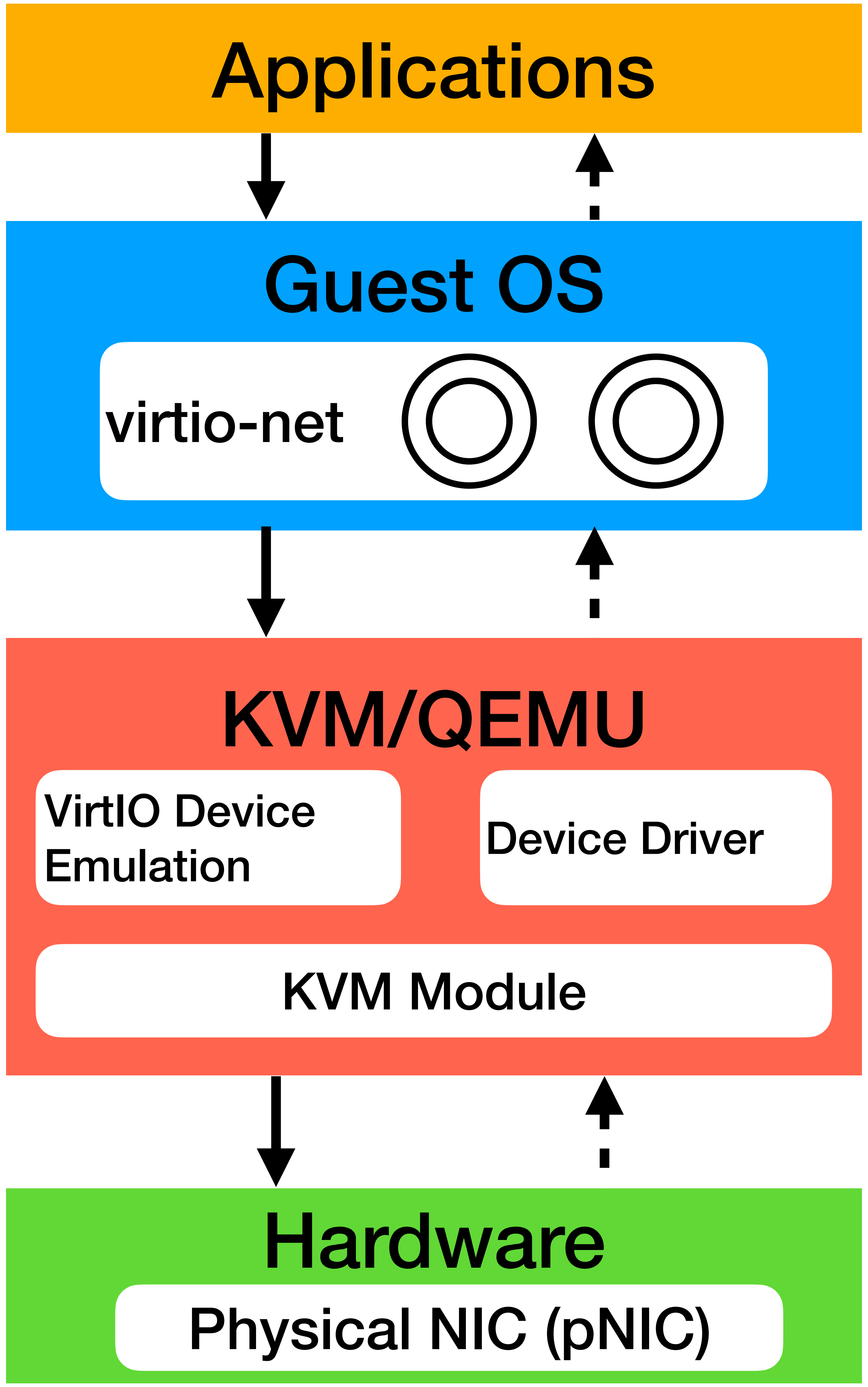


Back-end Virtio Packet Flow



1. When the guest OS writes data to the address, it triggers a VM-exit (reason code = EPT_MISCONFIGURATION);

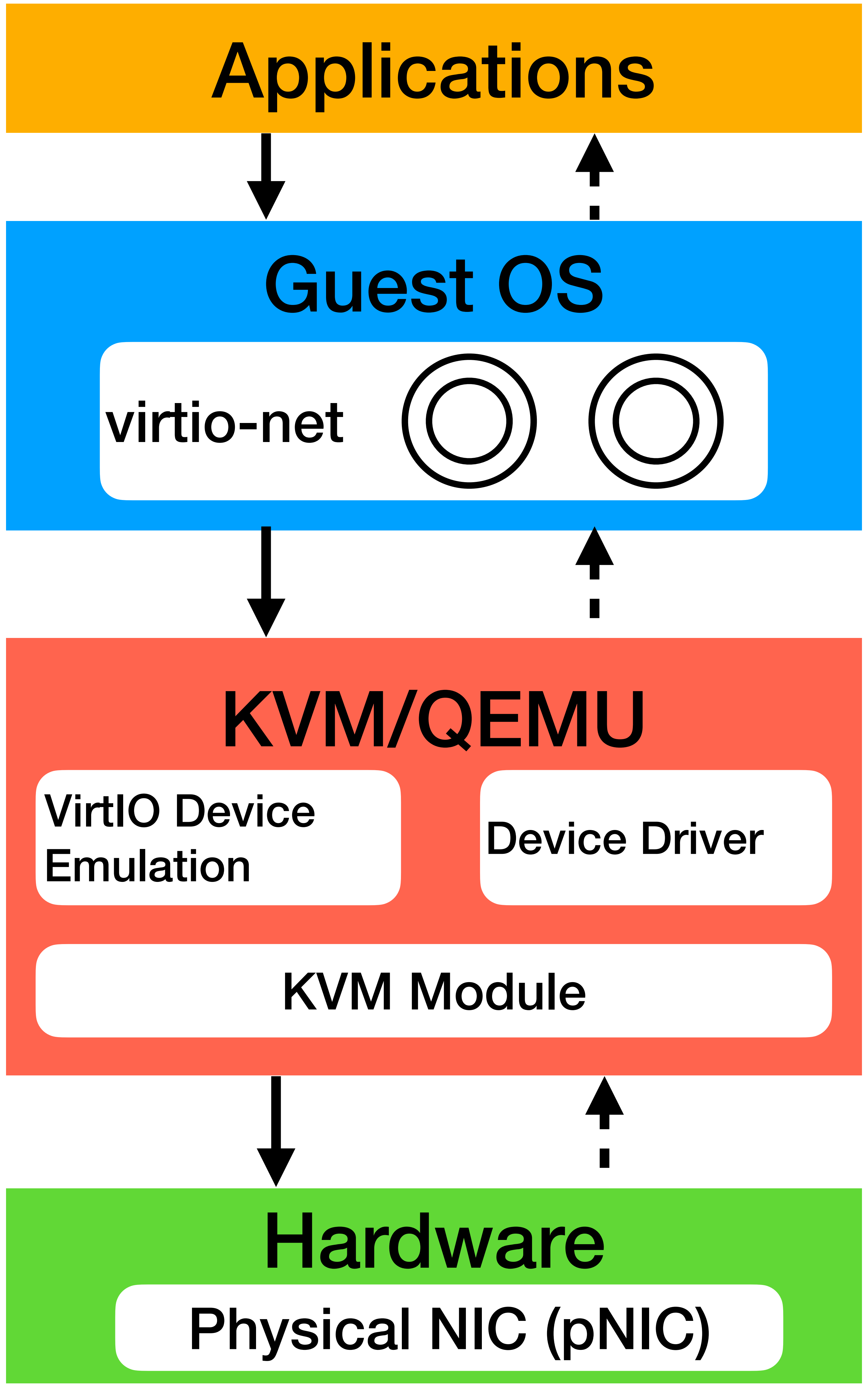
Back-end Virtio Packet Flow



1. When the guest OS writes data to the address, it triggers a VM-exit (reason code = EPT_MISCONFIGURATION);

2. A corresponding KVM VM-exit handler is called, which notifies QEMU based on the registered ioventfd address range;

Back-end Virtio Packet Flow

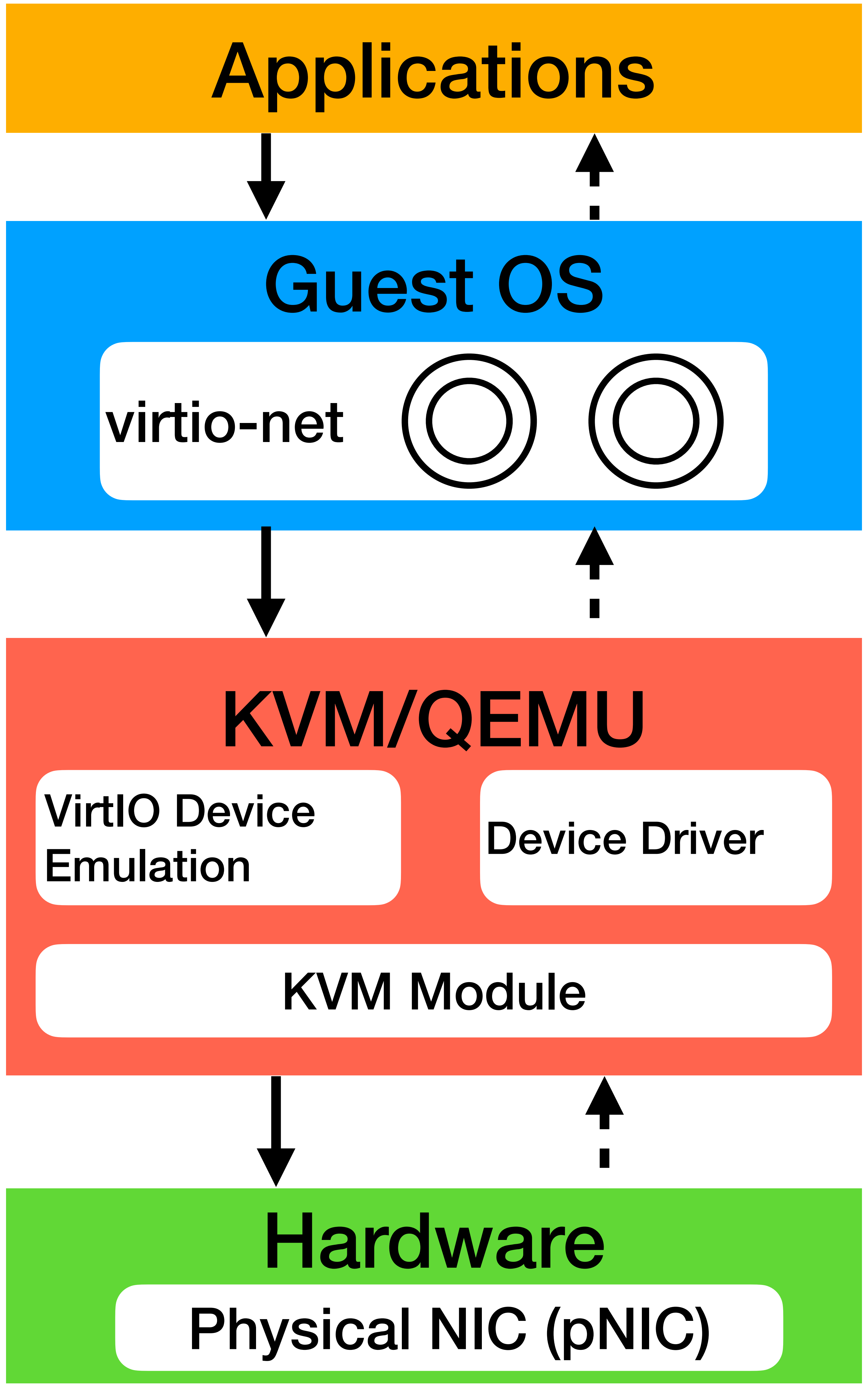


1. When the guest OS writes data to the address, it triggers a VM-exit (reason code = EPT_MISCONFIGURATION);

2. A corresponding KVM VM-exit handler is called, which notifies QEMU based on the registered ioventfd address range;

3. Once finished reading/writing operations, QEMU injects an IRQ to notify that the operation is complete;

Back-end Virtio Packet Flow



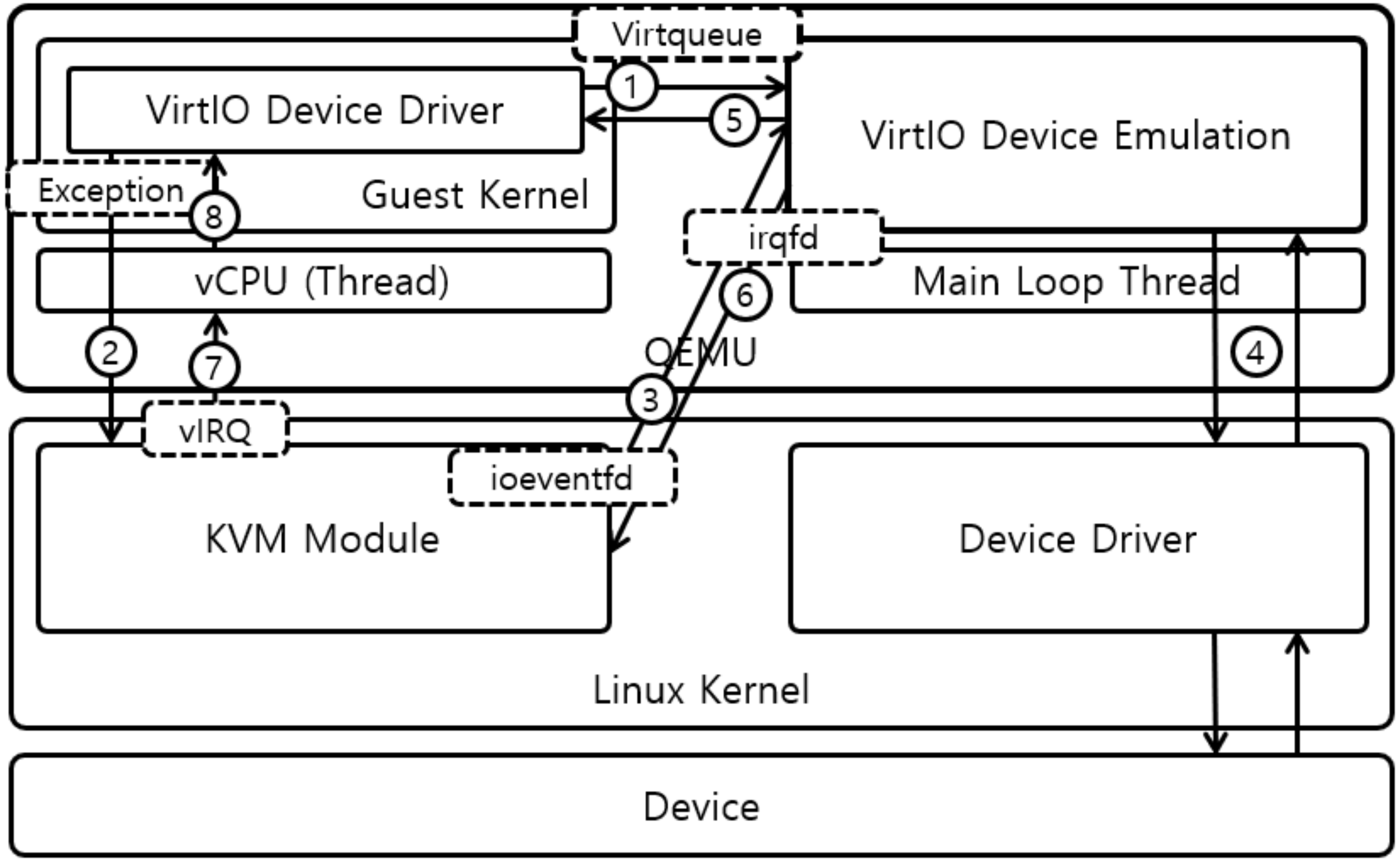
1. When the guest OS writes data to the address, it triggers a VM-exit (reason code = EPT_MISCONFIGURATION);

2. A corresponding KVM VM-exit handler is called, which notifies QEMU based on the registered ioventfd address range;

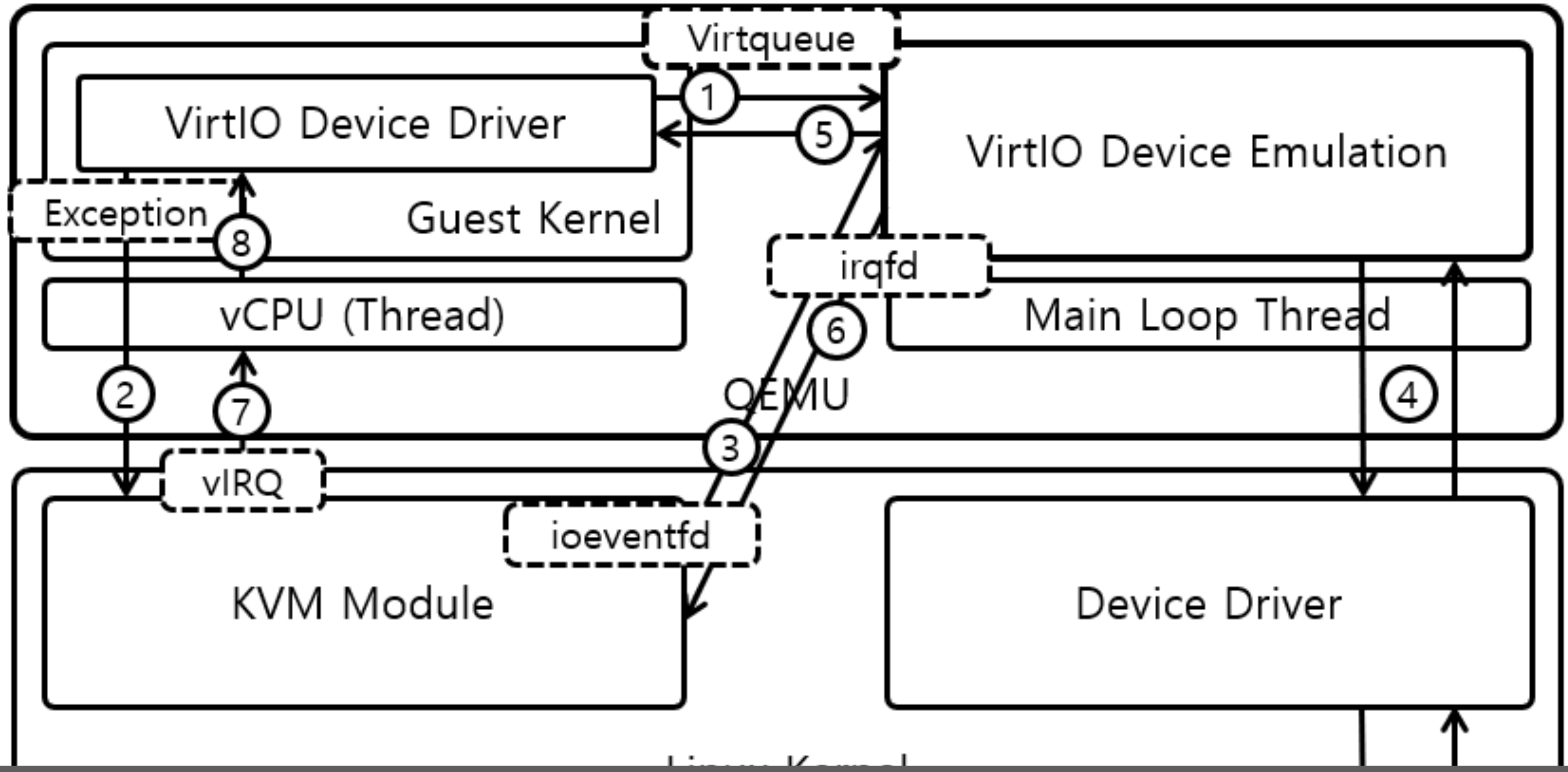
3. Once finished reading/writing operations, QEMU injects an IRQ to notify that the operation is complete;

4. A vIRQ is further delivered to the guest kernel;

VirtIO Overall

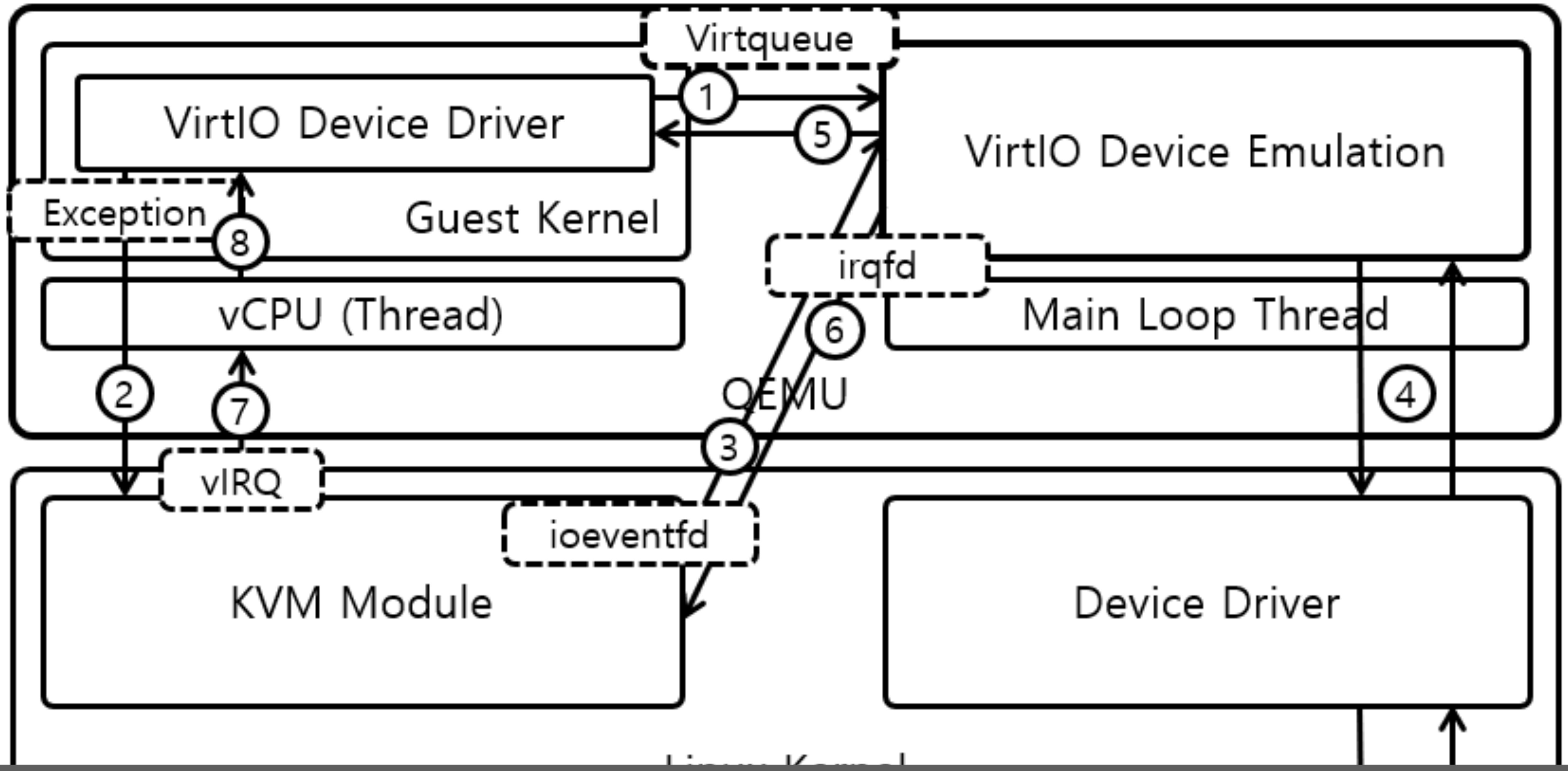


VirtIO Overall



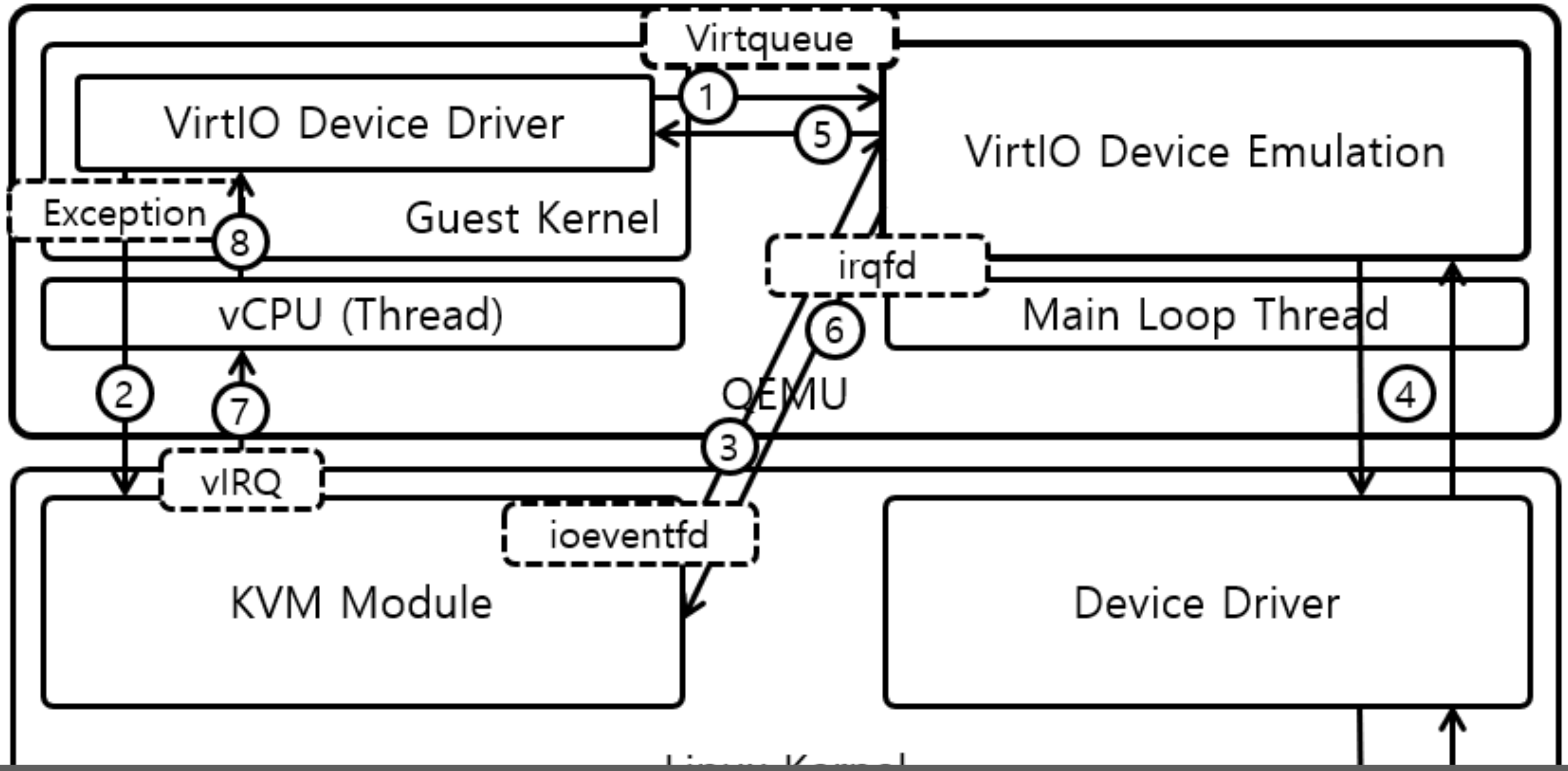
- Step 1: The guest OS rings a doorbell after inserting into the virtqueue;
- Step 2: The context is forwarded to the host KVM handler;
- Step 3: The context is forwarded to the QEMU process via ioeventfd;

VirtIO Overall



- Step 4: QEMU process reads the request from the virtqueue and handles it
- Step 5: After completion, QEMU puts the results into the virtqueue;
- Step 6: QEMU injects an IRQ through irqfd to the guest;

VirtIO Overall



- Step 7: A vIRQ is injected into the guest OS;
- Step 8: The guest OS execution is resumed, the request I/O operation is done and virtio device driver gets results data from the virtqueue;

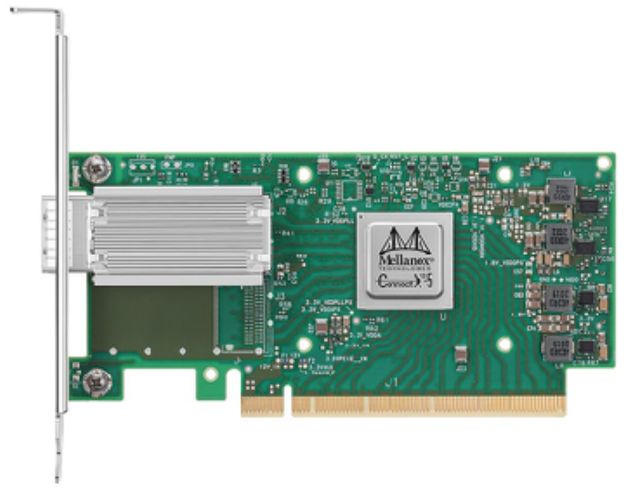
How is the VM network policy applied?

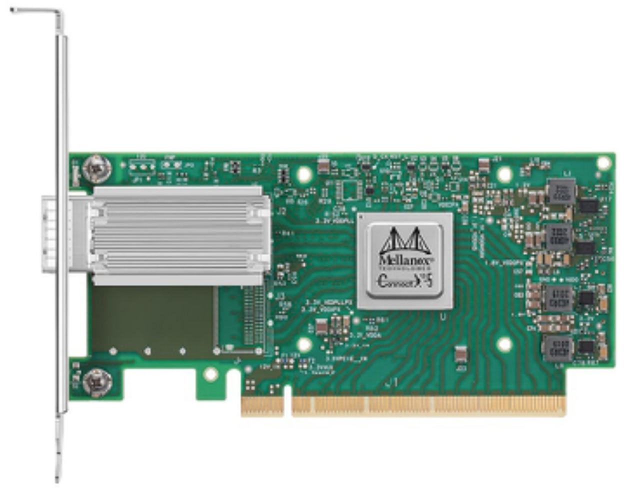
Host Virtual Networking

VM

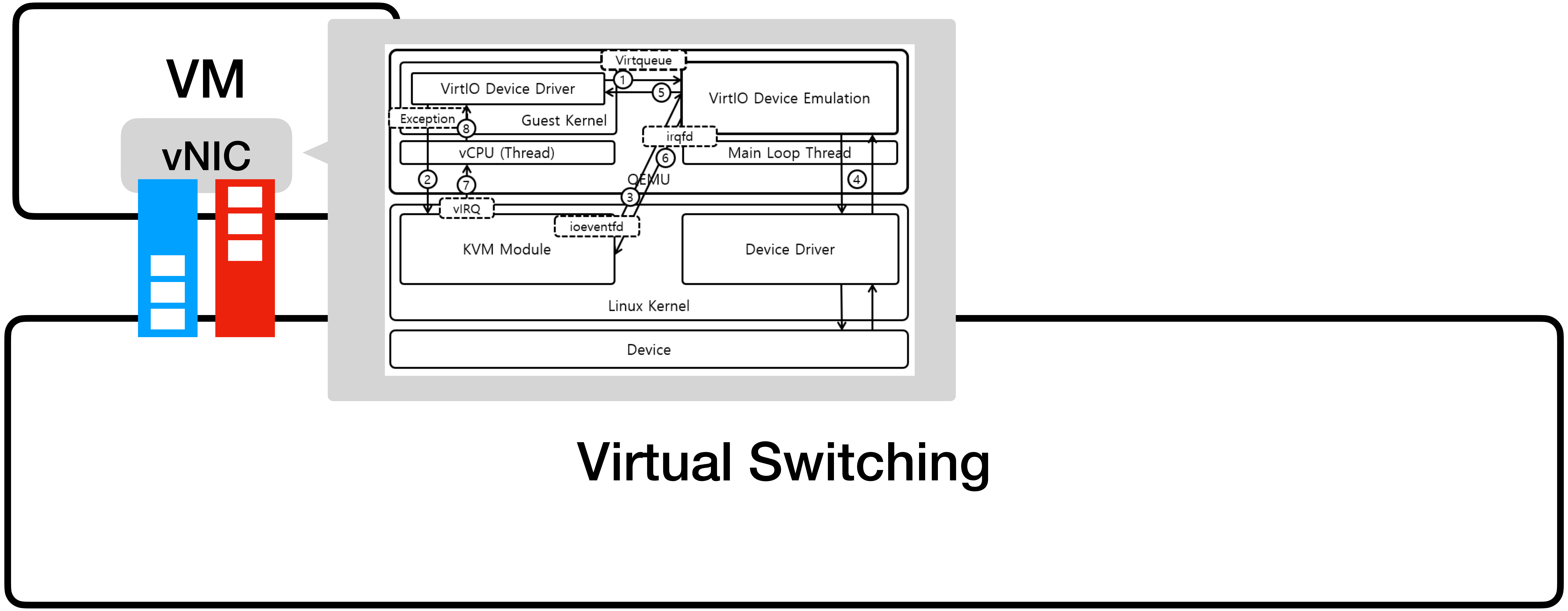
vNIC

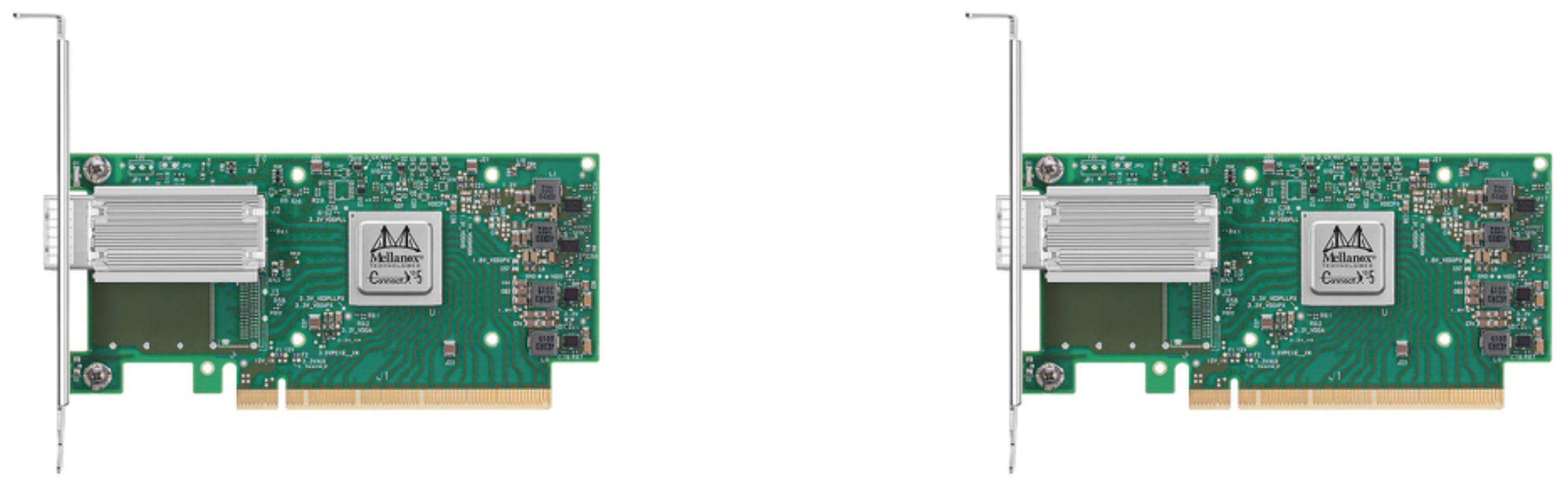
Virtual Switching



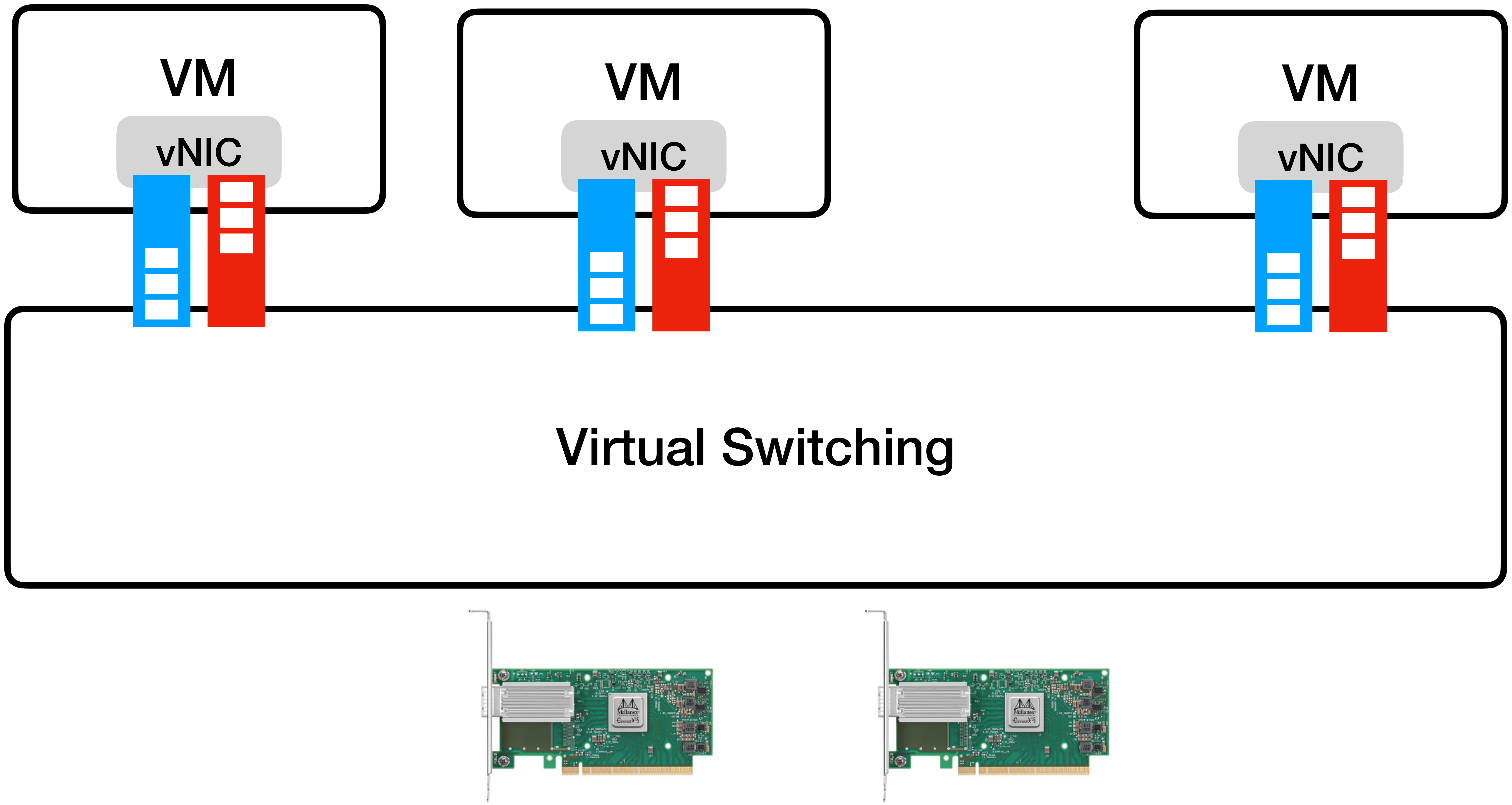


Host Virtual Networking

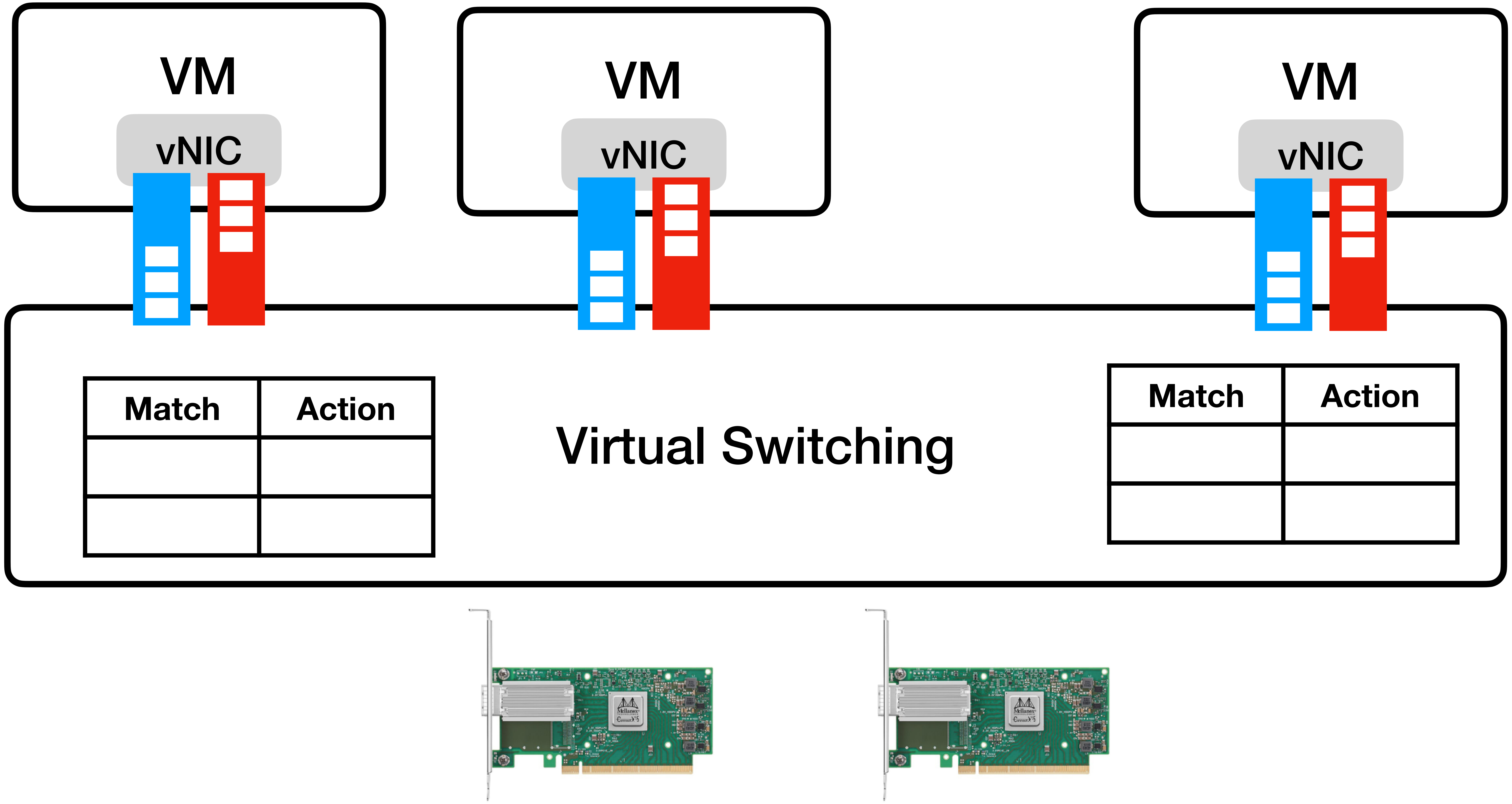




Host Virtual Networking



Host Virtual Networking



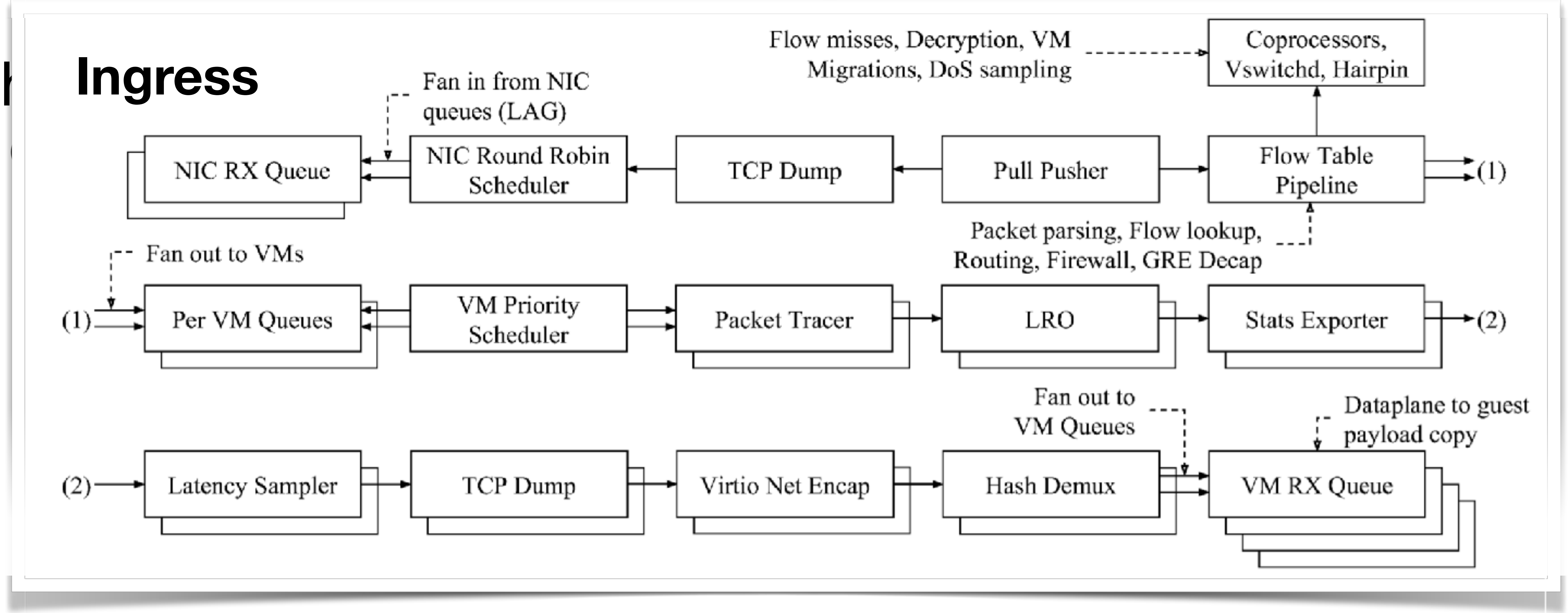
Andromeda

- Kernel bypass data plane
  - Fast packet flow between guest OS and hypervisor

- Chained table execution
  - On both ingress and egress

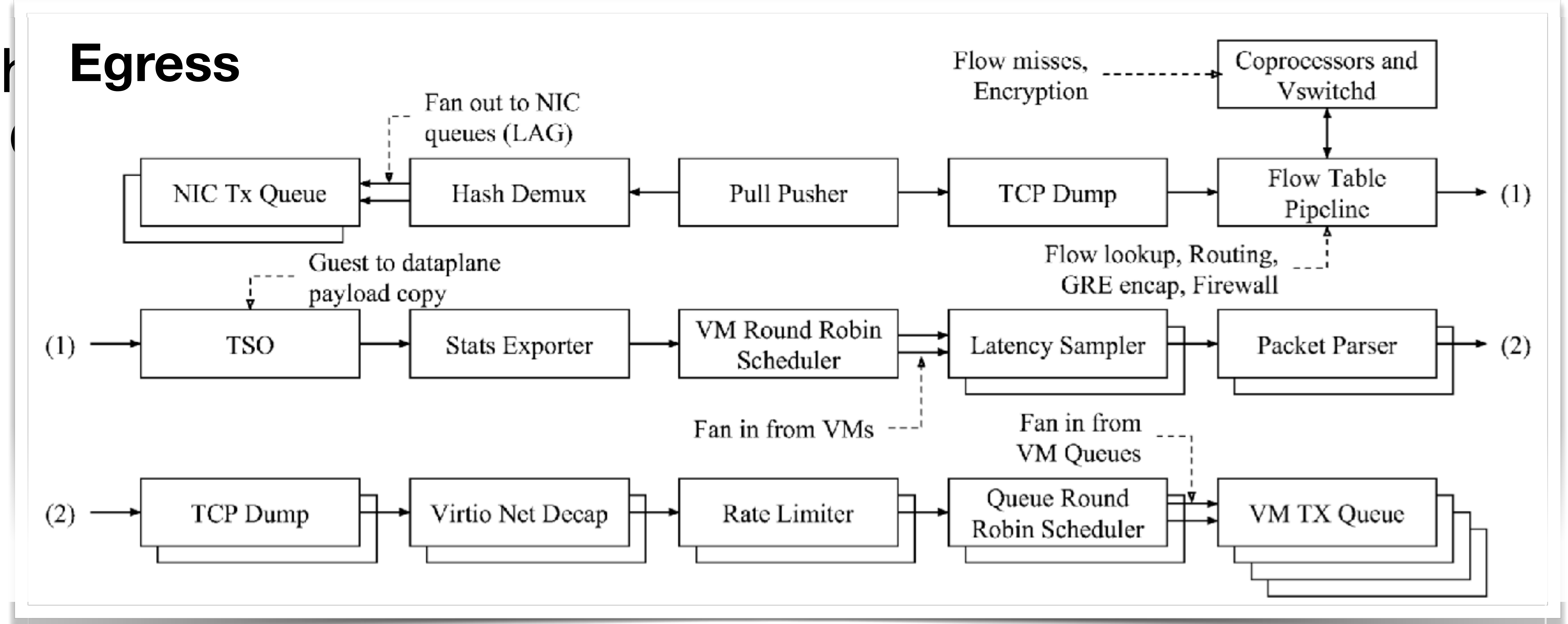
Andromeda

- Kernel bypass data plane
  - Fast packet flow between guest OS and hypervisor
- Ch
  - (



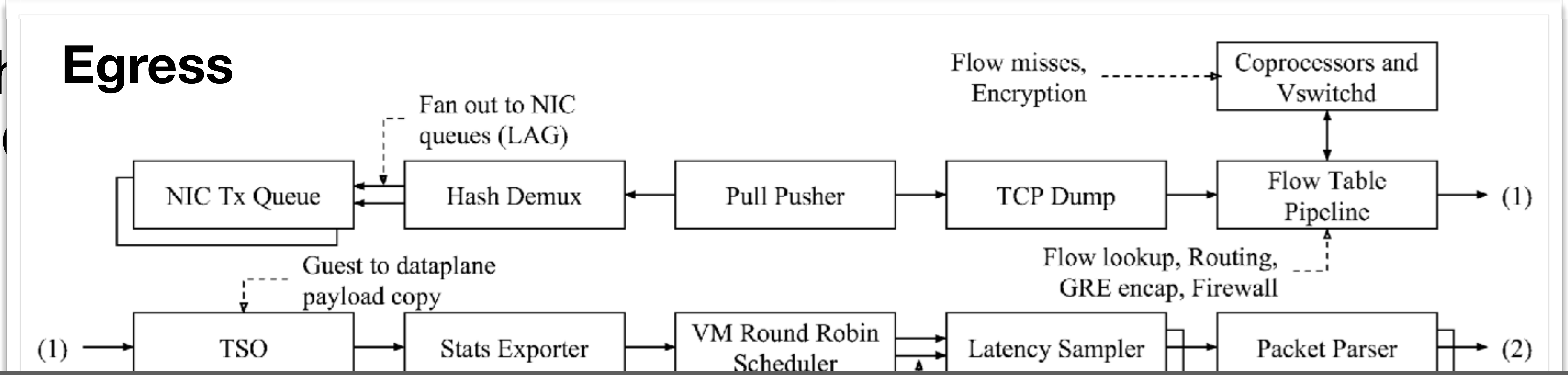
Andromeda

- Kernel bypass data plane
  - Fast packet flow between guest OS and hypervisor
- Ch
  - (



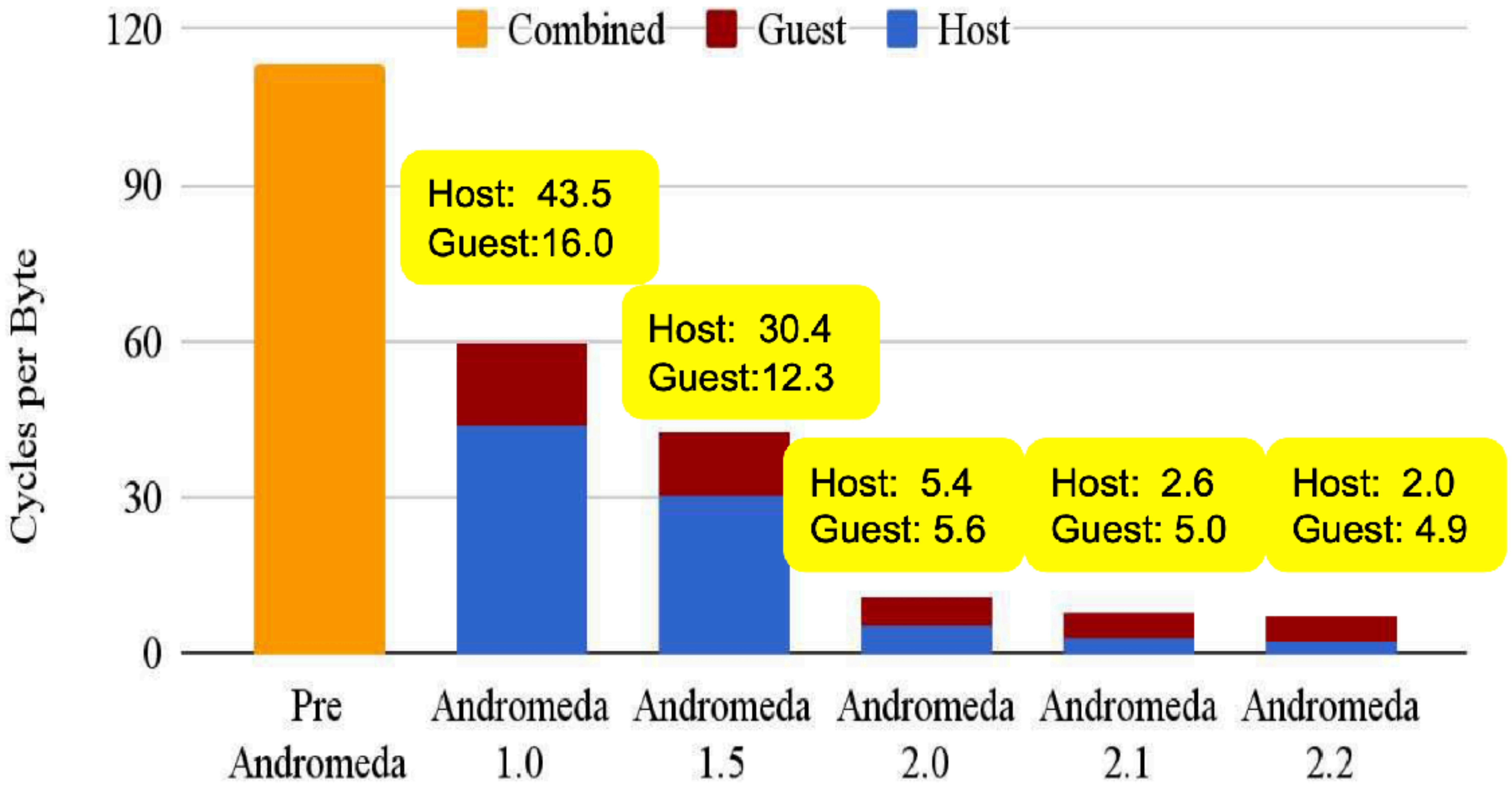
Andromeda

- Kernel bypass data plane
  - Fast packet flow between guest OS and hypervisor
- Ch
  - (



Assign a co-processor thread with a guest VM to run CPU-intensive ops

Andromeda Evolution



Andromeda 1.0
Kernel datapath

Andromeda 1.5
Optimize pipeline

Andromeda 2.0
OS bypass, 1 thread hop

Andromeda 2.1
Remove thread hop

Andromeda 2.2
Memory copy offload

Read the code, check virtio/virtio-net/Vhost implementations, and play with it!

Summary

- Today
  - Network virtualization in data center networks (I)

- Next
  - PicNIC (SIGCOMM’19)