OPENFABRICS ALLIANCE

Foundations of Virtualization

Author: Paul Grun
Date: April 5, 2011

Server I/O Virtualization

1. Is there something about Channel I/O that makes it inherently a superior solution for I/O virtualization?

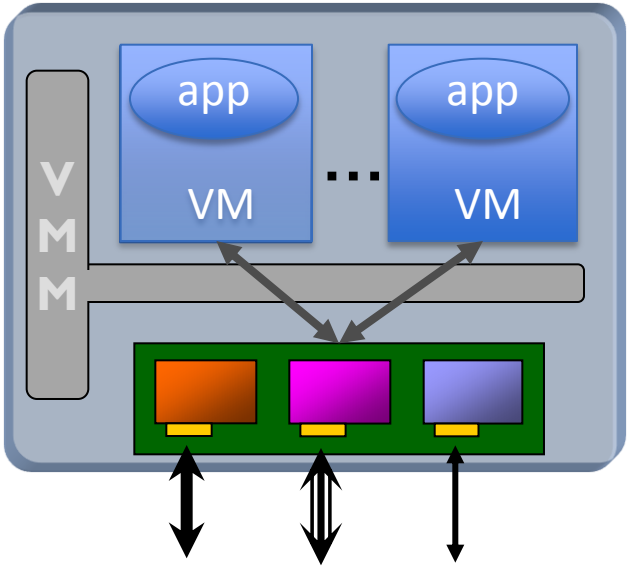
2. What, if anything, is being done today to ‘virtualize’ Channel I/O?

3. Is there something the OFA could/should be doing to capitalize on CI/O as virtualization solution?

At the High Level

- Server I/O virtualization
  - virtualize server I/O resources allowing support for multiple application containers
- Datacenter virtualization
  - Flexible allocation of datacenter resources to applications

Server I/O Virtualization



Lots of good reasons to do this.
Chiefly:

- improve server utilization
- server consolidation
- run unmodified guest OS
- create sandboxes…
- familiar compute and IT models

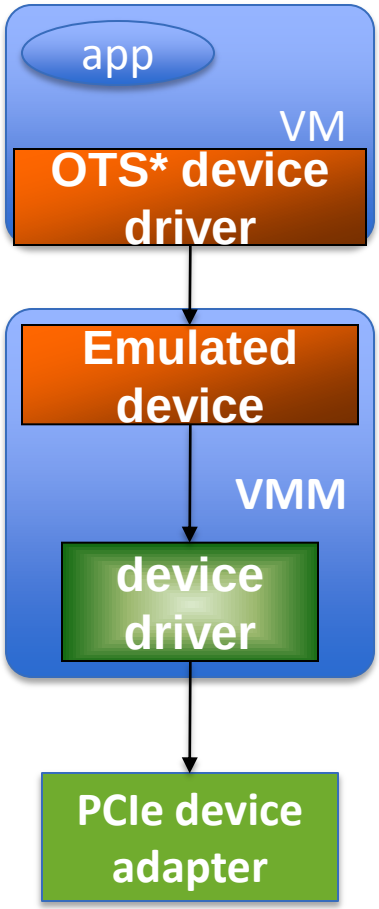
Naturally, there may be some downsides too

- variable workloads across VMs makes it hard to provision the server platform
- incorrect mix of resources (compute, memory, I/O) available on this platform

Some well-known I/O virtualization models

- Emulation
  - VMM emulates I/O devices in software.
- Direct Assignment
  - Guest VM has direct access to physical hardware (vs a device emulated in the VMM).
- Paravirtualization
  - Hypervisor presents a non-traditional, (higher level ?)interface to the guests

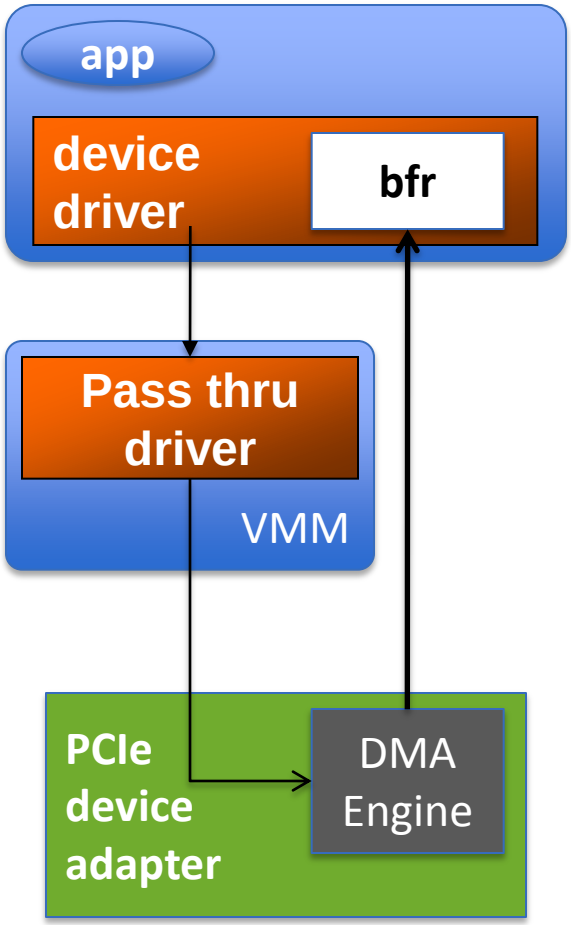
Emulated I/O



- VMM exactly emulates the physical device
- The guest VM runs a stock device driver
- Guest OS is (or can be) a stock, unmodified OS
- Can present some performance issues

*OTS = Off the Shelf

Direct Assignment



A Physical Adapter is assigned to just one Guest VM

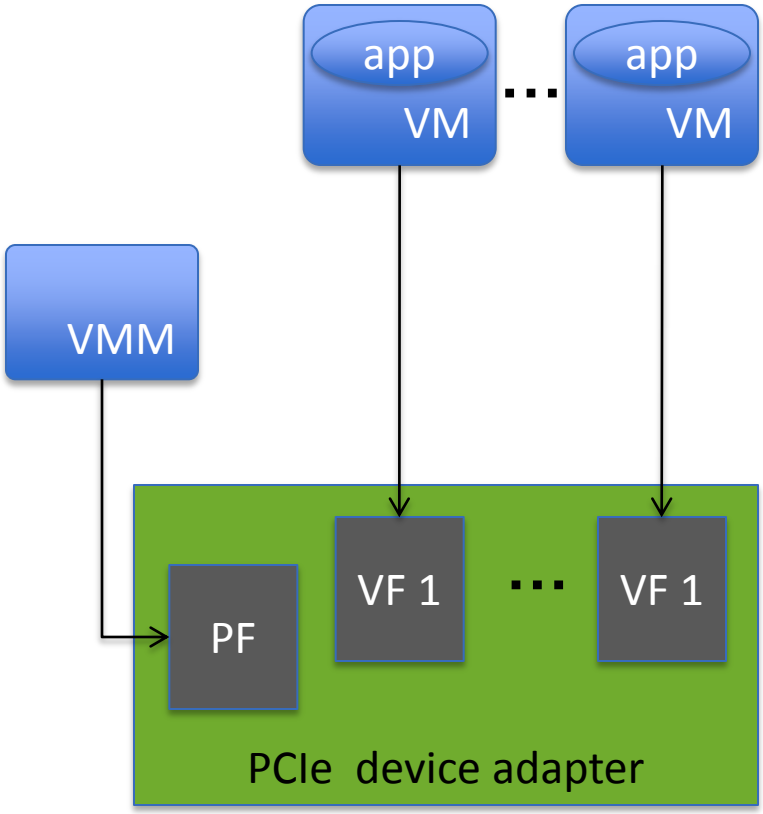
- Device driver resides in virtual space.
- But PCIe devices deal in physical addresses
- VMM passthru driver performs v-p address translation.
- Device adapter now knows the physical location of the guest's buffer
- Higher performance than Emulation

Limitations:

-Requires one physical device per guest VM
- Requires VMM support for v-to-p address translations and to handle interrupts.

SR-IOV

SR-IOV overcomes the ‘one adapter / one guest’ limitation of direct assignment.



- Each guest VM sees a virtual PCIe adapter - a Virtual Function (VF)

-VMM manages the physical adapter through a Physical Function (PF)

-The device adapter can support as many VMs as it has VFs (currently set to 256 functions per bus address.)

Limitations:
- Requires VMM support for v-to-p address translations and to handle interrupts.

An observation

All the above export some version of a PCIe device adapter to the guest.

- Direct assignment – exports the true device adapter

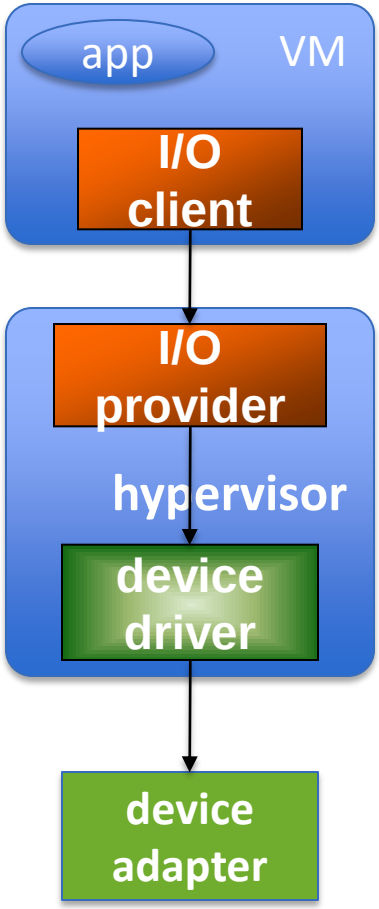
- Emulation – exports a virtualized device adapter

-SR-IOV – exports a variation on a physical device adapter

These all seek to preserve the PCIe *Device Adapter Model*, with as little change as possible

This seems to be about finding ways to share device adapters among guest VMs

Paravirtualized I/O



-Guest is not constrained to run native device drivers.

-This means that the hypervisor can present a “more idealized” device abstraction to the guests

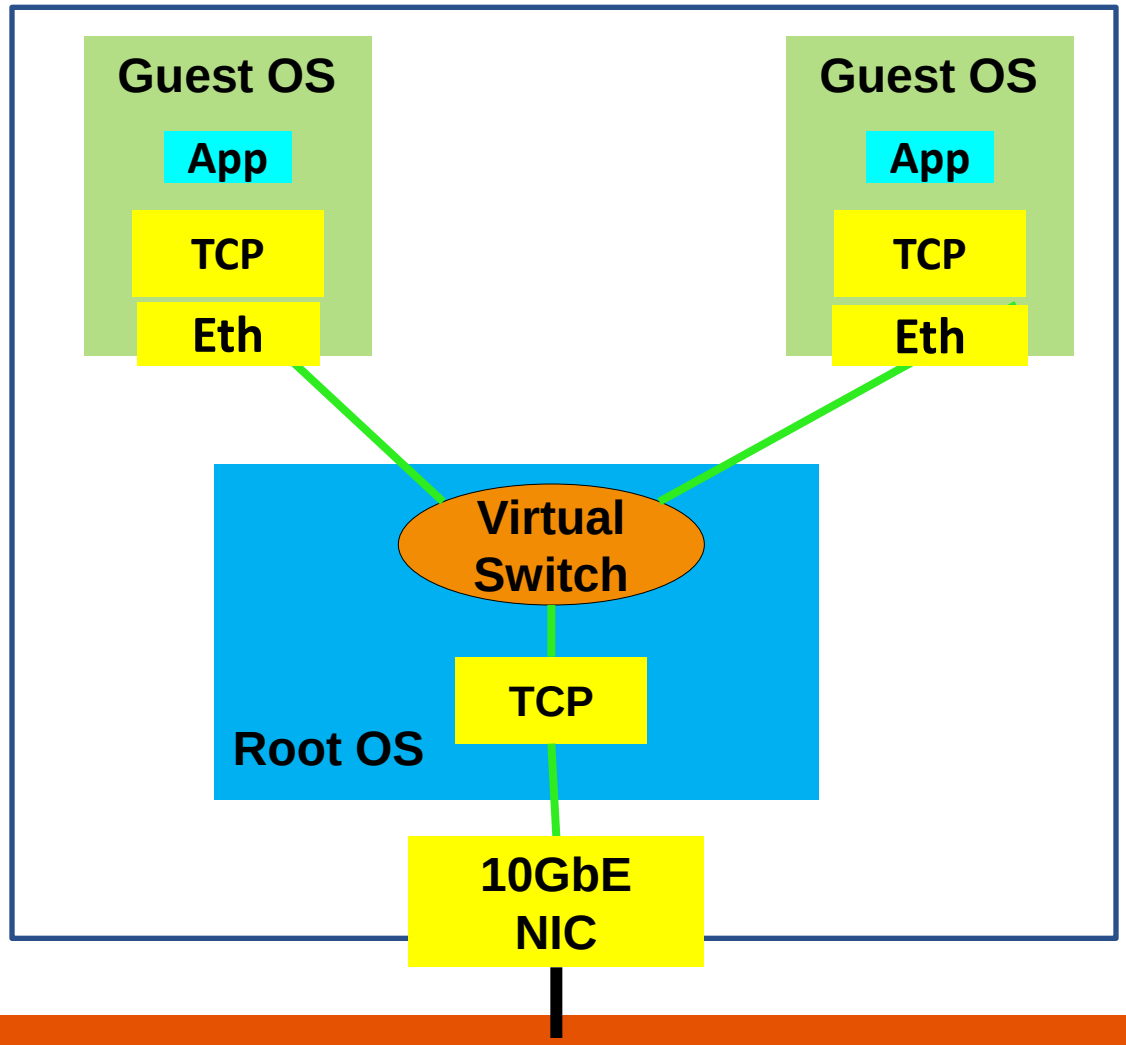
- Hypervisor provides a paravirtualized I/O service

- Hypervisor programs device adapter’s DMA engines with appropriate physical addresses

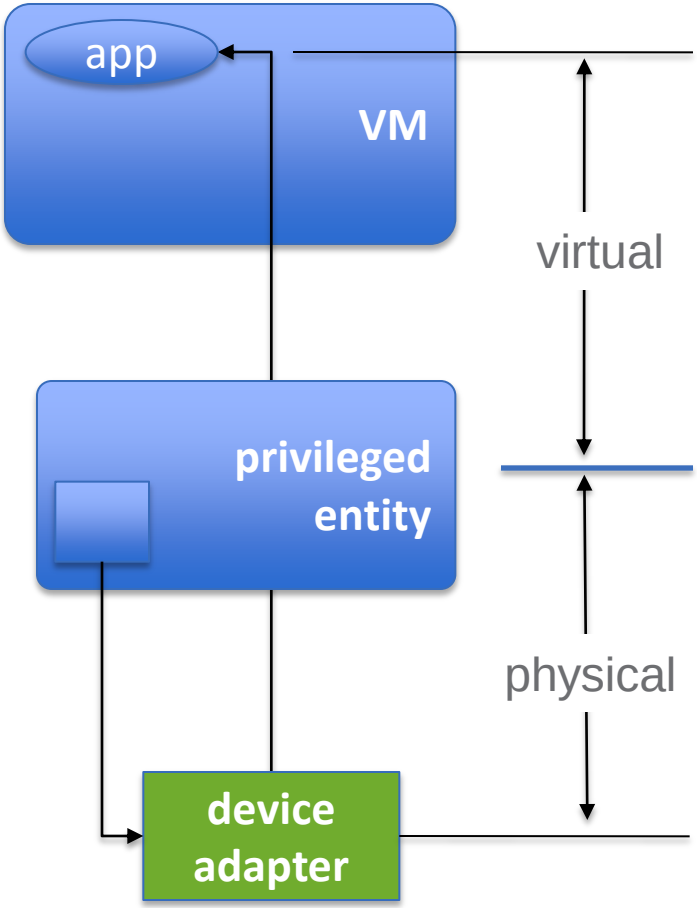
- Paravirtualization removes the ‘hard to virtualize’ x86 instructions from the guest OS.

KVM/virtio is an excellent example.

KVM example


Guest OS
App
TCP
Eth
Guest OS
App
TCP
Eth
Virtual Switch
TCP
Root OS
10GbE NIC

Just to state the obvious…

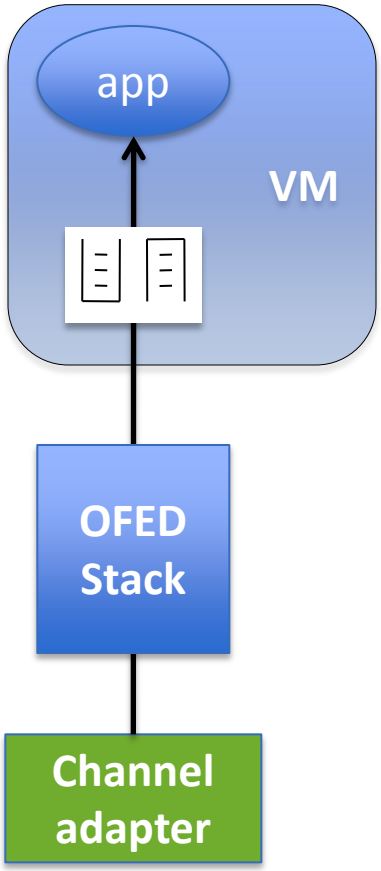


- PCIe device adapters operate in the physical realm. They know how to DMA data into physical memory

- Apps, on the other hand, live in a virtual realm

- This means that a privileged entity with knowledge of virtual to physical mappings must control the PCIe device adapter

Channel I/O



An RDMA Channel Adapter (IB HCA or iWARP rNIC) does the v-p-v translations automatically

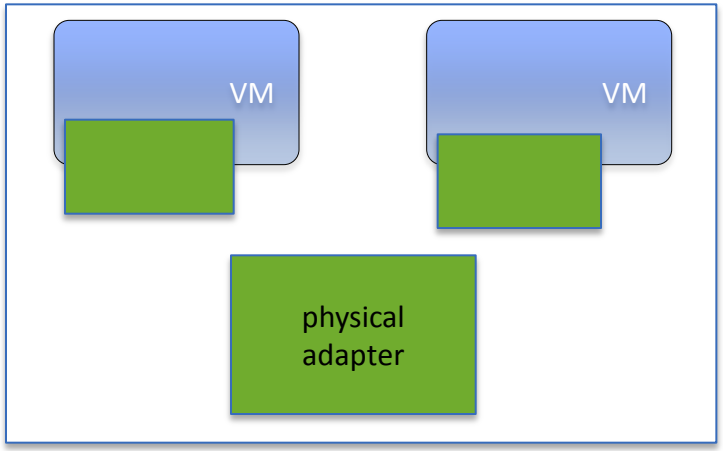
This is exactly what channel I/O does best

- exports a channel interface directly to an application's virtual space

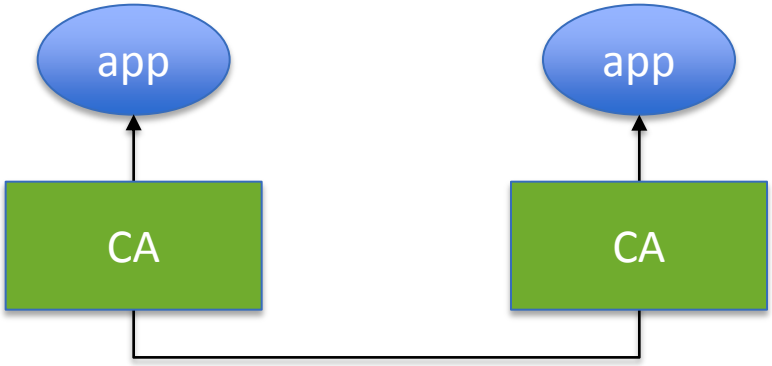
For virtualization, should we be exposing a higher level channel interface?

Has anybody explored creating a paravirtualized I/O message passing service based on OFED?

Virtualizing what?



Common I/O virtualization models focus on sharing an I/O adapter device

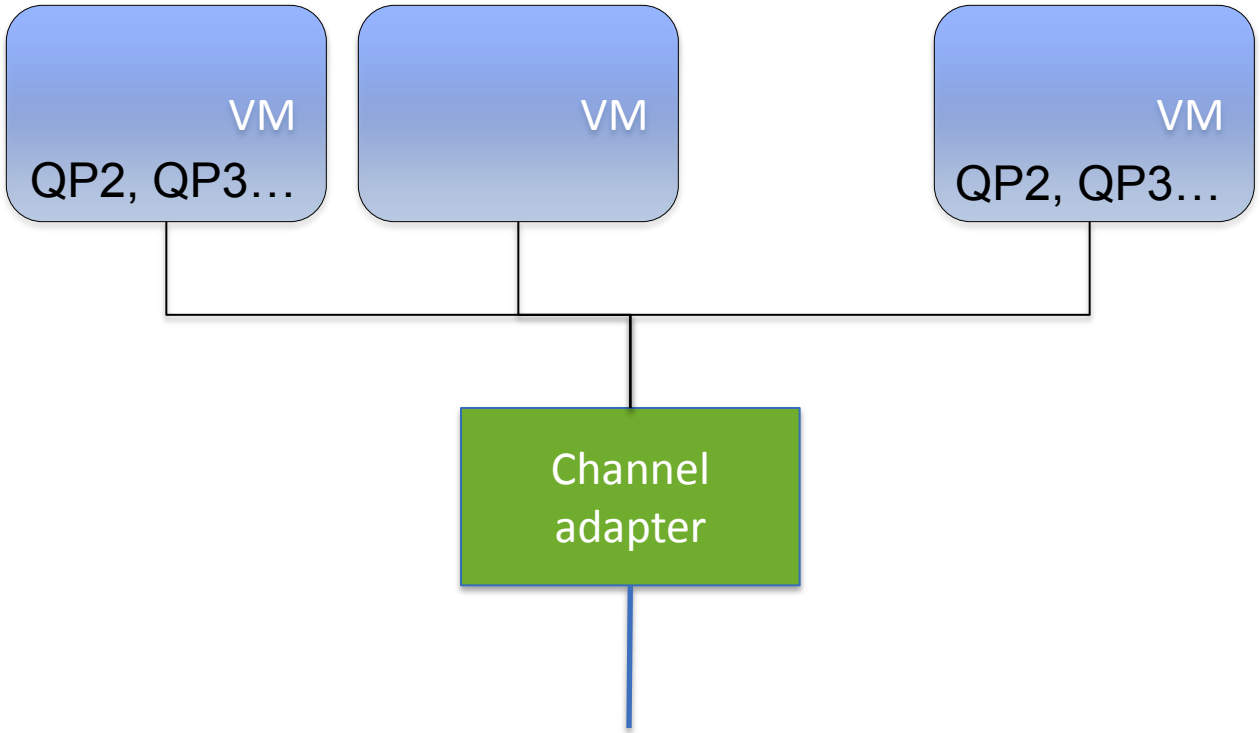


Channel I/O-based virtualiztion is a mechanism for delivering I/O messages to a virtual space

A couple of issues

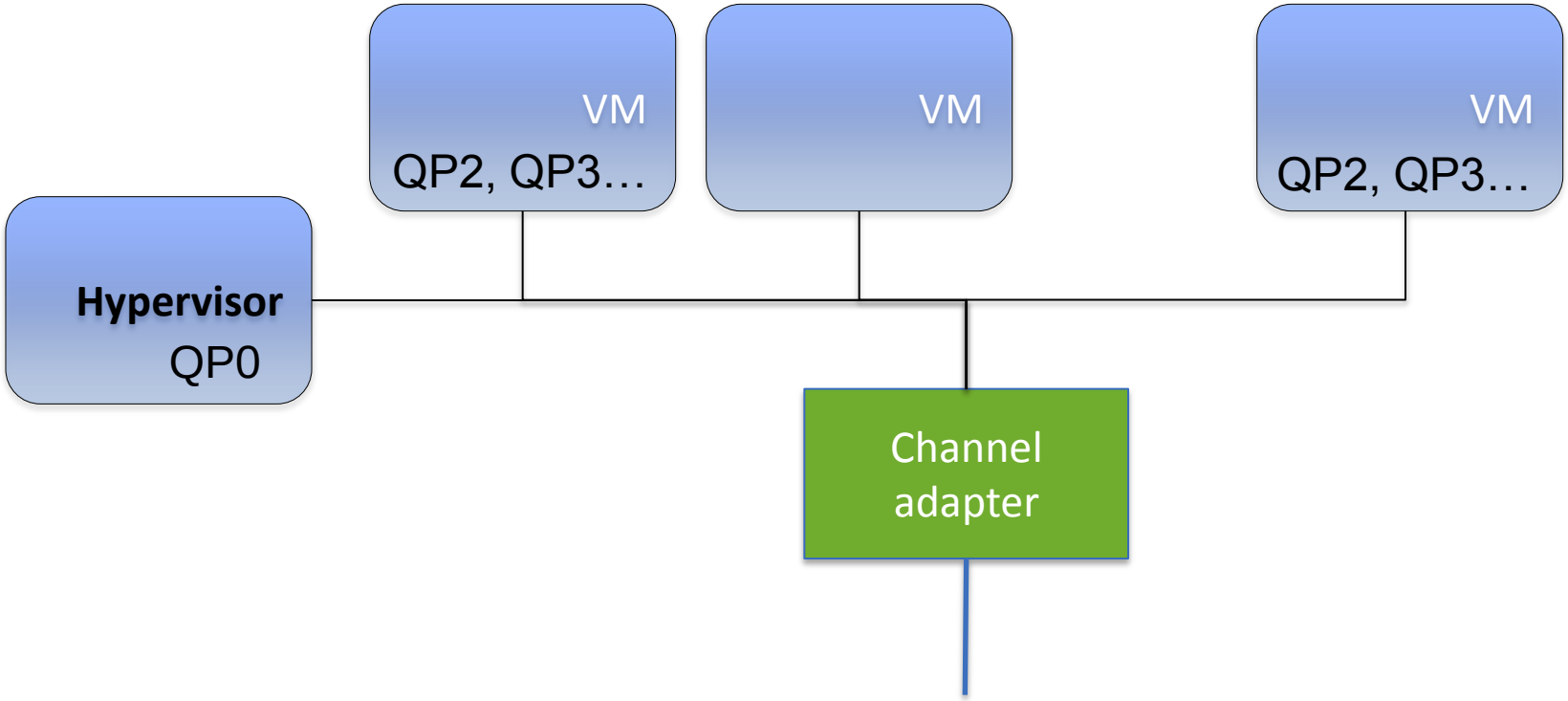
- CA support for non-unique QP spaces
  - Allow QP numbers to be freely assigned within each VM container
- Fabric management – QP0
- Connection management – QP1
- Addressing – identifying a VM

Non-unique QP space



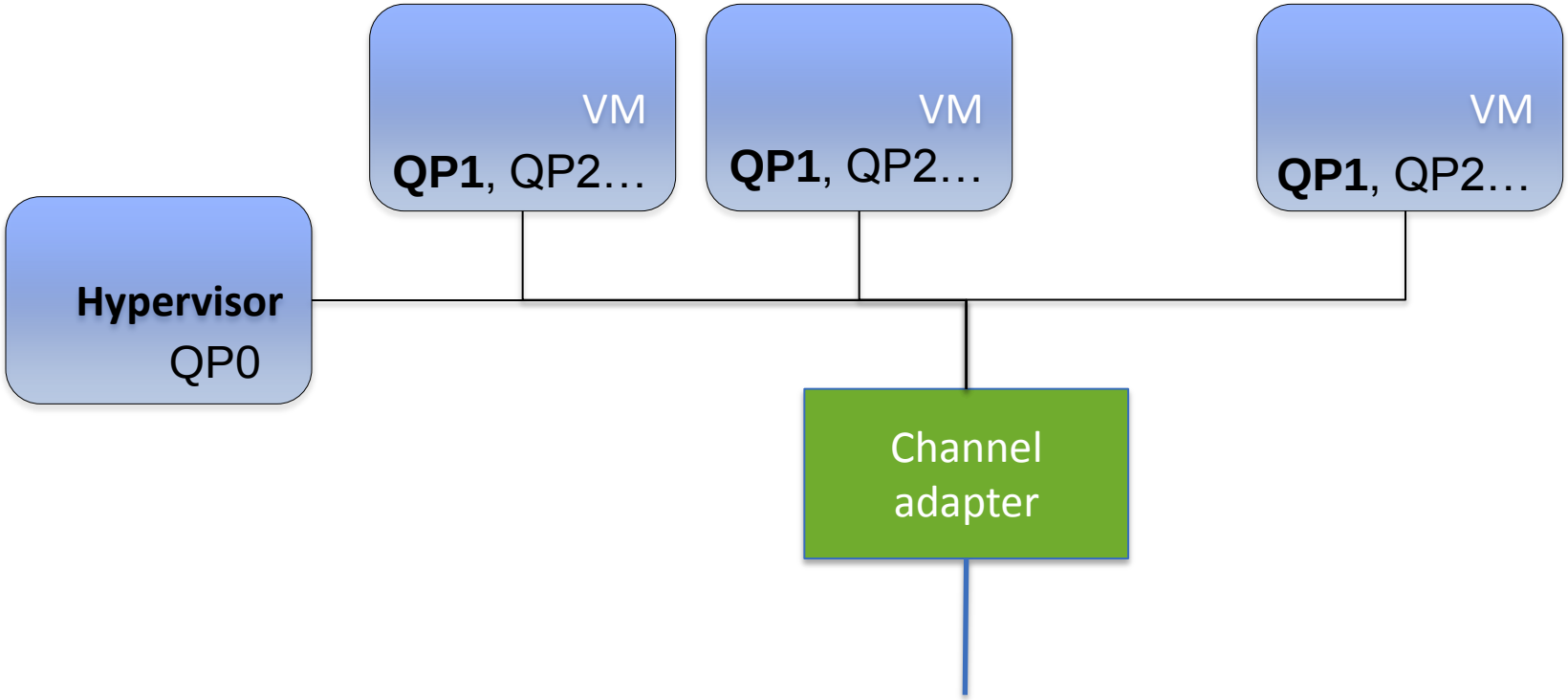
Prefer that the adapter's QP space be non-unique

Fabric management



Fabric Management – (this one's pretty easy)

General Services Interface



Each VM has a copy of QP1.
Inbound packets are steered to the VM via the GID
(note: GRH is currently optional for Unreliable Datagram service)

Summary

I/O device virtualization:

- share valuable adapter hardware
- enable unmodified OS to run in the guest
- sandboxes, private execution environments…

Channel I/O virtualization:

- flexible/agile data centers
- VM migration
- all the usual RDMA goodnesses (latency, CPU util…)

OPENFABRICS ALLIANCE

Thank you