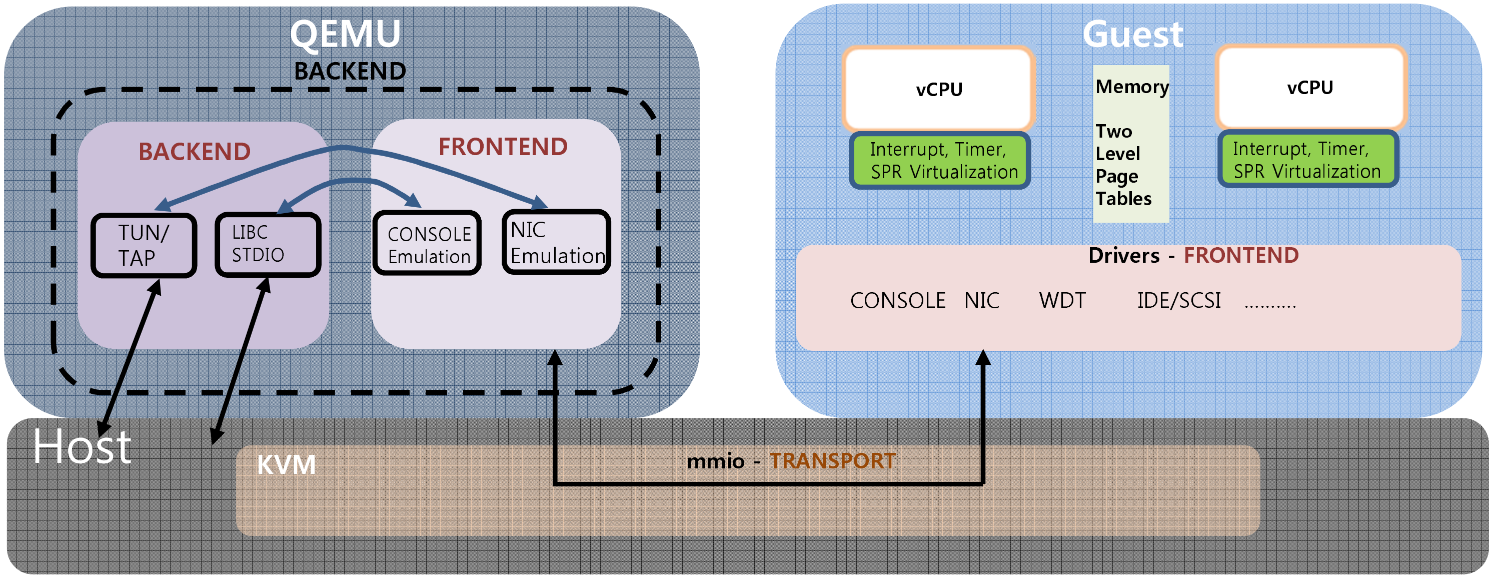
Reconnaissance of Virtio: What's new and how it's all connected?

Mario Smarduch
Senior Virtualization Architect
Open Source Group
Samsung Research America (Silicon Valley)
m.smarduch@samsung.com

Agenda

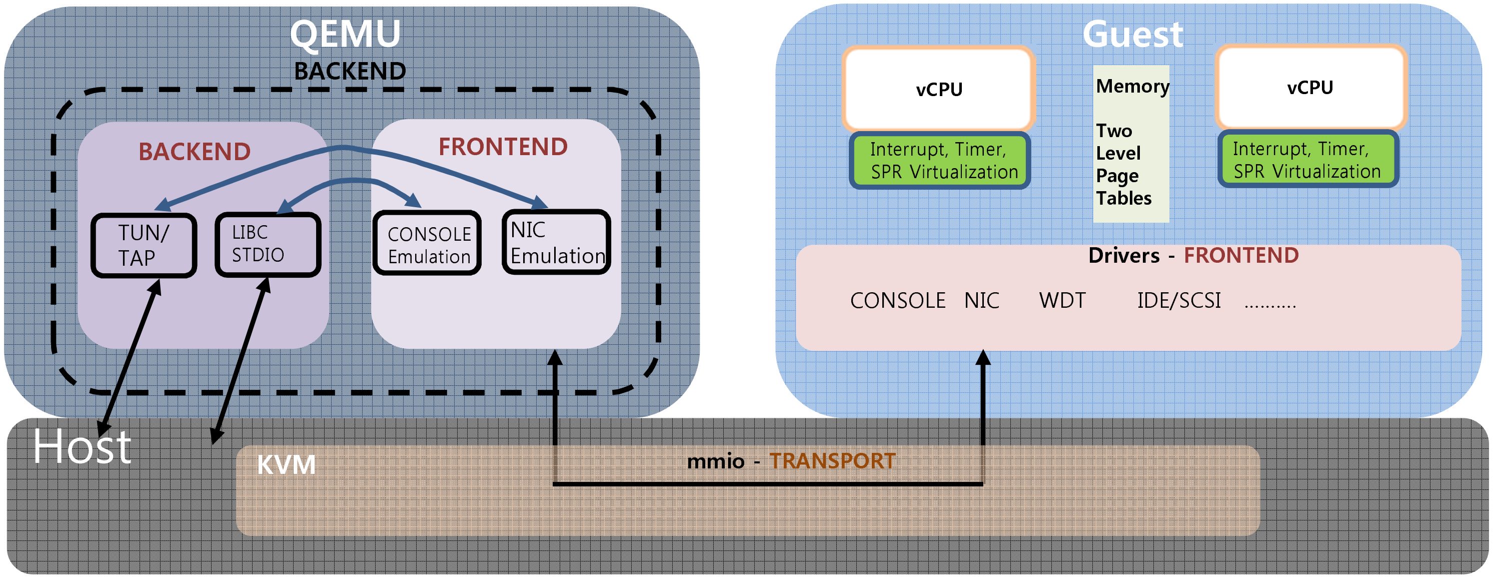
- QEMU/Guest Machine Model & IO Overview
- Concepts - transport/backend – recent re-factoring
- PCI transport and most recent virtio-mmio transport
- Virtio and Device Passthrough, virtio performance

Machine Model



- Like host, unmodified guest expects real hardware
- Machine model – combination of hw extensions, KVM, QEMU, GUest
  - **Interrupt Local and Distributor** – hw virt extensions + kvm
  - **Special Purpose Register** – i.e. enable/disable MMU, discover CPU features – hw virt ext + kvm
  - **Timer** – hw virt extensions + kvm
  - **Memory** – hw virt + kvm
  - **Drivers/Devices** – (i) mmio (ii) para-virtualized (iii) dev passthrough
  - **Machine Model** - defines hw – CPU, Peripherals, HW address map
- Some Terms
  - **Transport** – way Guest to (i) probe, discover backend – resources; (ii) configure backend
  - **Frontend** – guest driver
  - **Backend** – whole QEMU I/O emulation + host device

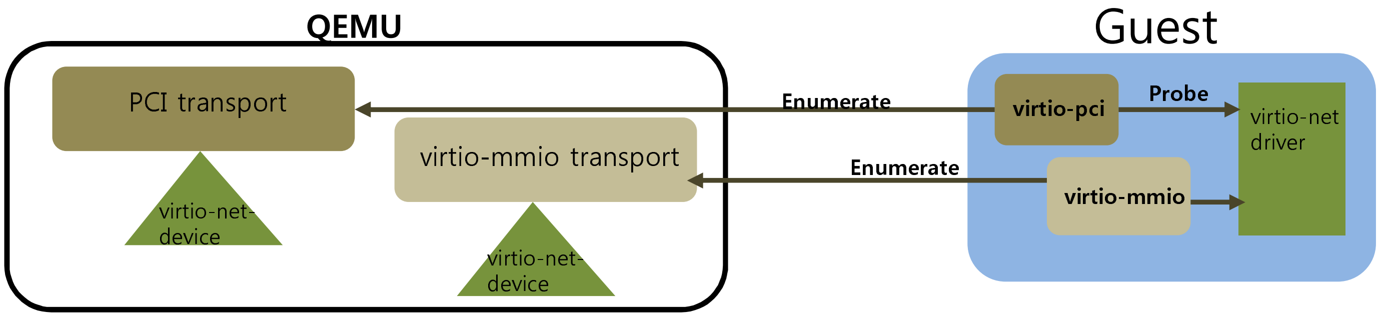
MMIO Example



- Typing a charter – '-nographic'
  - Keyboard stroke – QEMU backend (IO thread) reads from stdio
  - Finds Qemu Frontend – console emulation device passes character
  - Console device injects interrupt via KVM, guest exit/resume
  - Console interrupt handler – mmio read of device buffer
    - guest exits, decodes regs to packages addr/data size
    - Returns from vCPU KVM_RUN loop to QEMU
    - QEMU finds console device handler from addr (GPA)
    - Console handler returns data at address
    - Return to KVM, data placed in dest register
    - Resume guest
- MMIO a lot of overhead!

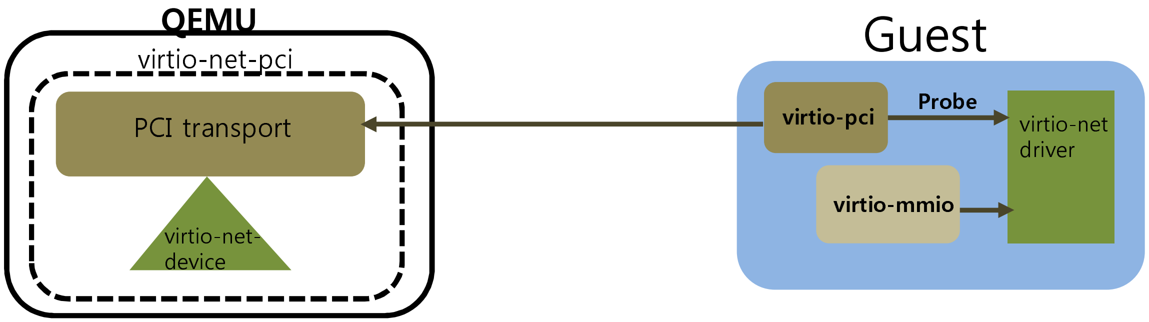
Vision and Practice

- QEMU/Guest - Vision
  - ❑ Portability any backend plugs into any transport – no clue about transport
    - ○ Typically one transport configured
    - ○ '- virtio_xxx_device' option – no hint of transport – plug into first available one
  - ❑ Guest virtio driver unaware of transport
    - ○ All transports can probe, discover backend
    - ○ Indirect transport interface – i.e. virtio-net does not know what transport
  - ❑ Example



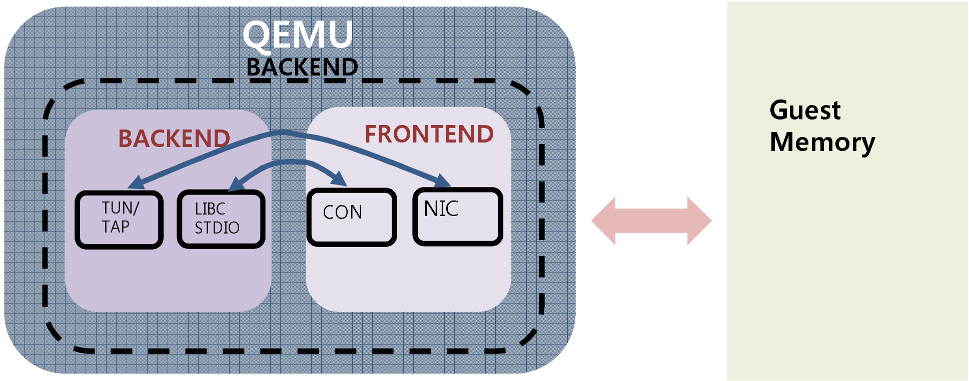
Vision and Practice

- In Practice – PCI preferred transport
  - ❑ Transport/backends 'fused'
  - ❑ Backend plugged into PCI
  - ❑ Prior knowledge of machine model required
    - ○ Command line – specify transport
    - ○ No Portability



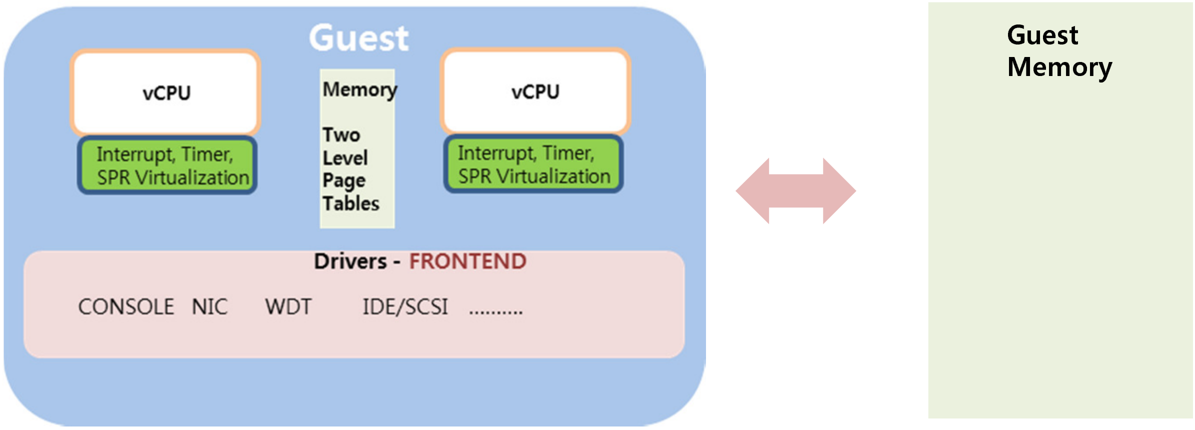
Virtio – moving data

- virtio – ring buffers accessed from several contexts
- Must deal with different addresses when moving data to/from virtio device



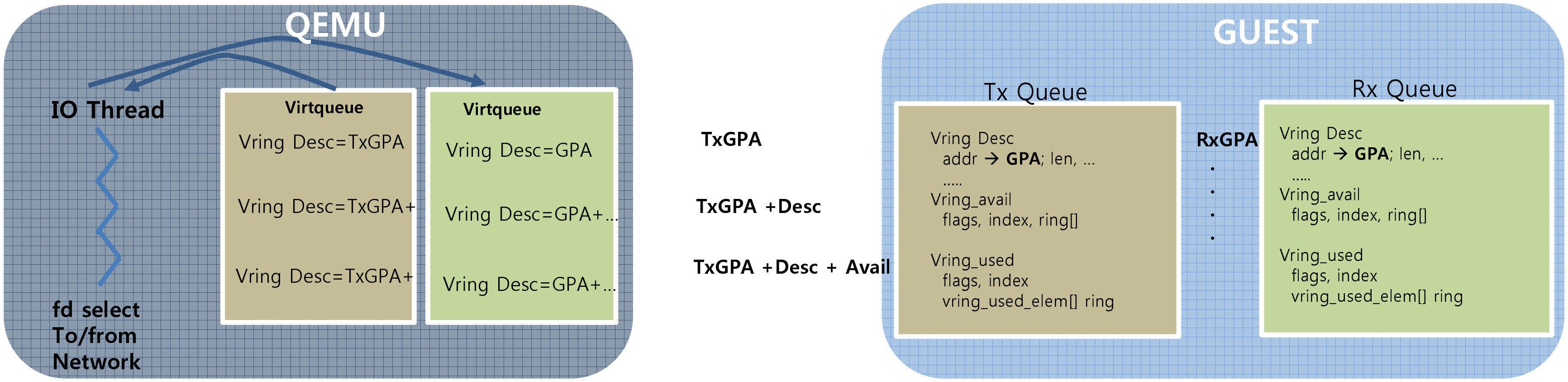
- Between Guest & QEMU – QEMU view
  - Host mmap() address – QEMU VA – HVA
  - To get HVA from GPA
    - Find memory region section
    - Offset = GPA – MemoryRegion base
    - Add HVA base in RAMBlock add offset
  - To get GPA from HVA
    - From RAMBlock find MemoryRegion
    - Offset = HVA address – HVA base
    - Add to MemoryRegion base address

Virtio – moving data



- Between Guest & QEMU or host – Guest view
  - Guest knows nothing abut HVA
  - Current hw supports two level page tables
  - 2nd level page table maps GPA → HPA

Virtio – moving data



QEMU

IO Thread

fd select
To/from
Network

Virtqueue
Vring Desc=TxGPA
Vring Desc=TxGPA+
Vring Desc=TxGPA+

Virtqueue
Vring Desc=GPA
Vring Desc=GPA+...
Vring Desc=GPA+...

TxGPA

TxGPA +Desc

TxGPA +Desc + Avail

GUEST

Tx Queue

Vring Desc
addr → **GPA**; len, ...
.....
Vring_avail
flags, index, ring[]
Vring_used
flags, index
vring_used_elem[] ring

RxGPA

Rx Queue

Vring Desc
addr → **GPA**; len, ...
.....
Vring_avail
flags, index, ring[]
Vring_used
flags, index
vring_used_elem[] ring

- Performance achieved through direct memory access (see Rusty Russels spec)

GUEST

Simple pkt – no fragments

1 - virtio_net_hdr (skb->cb[])
2 – skb->data[]

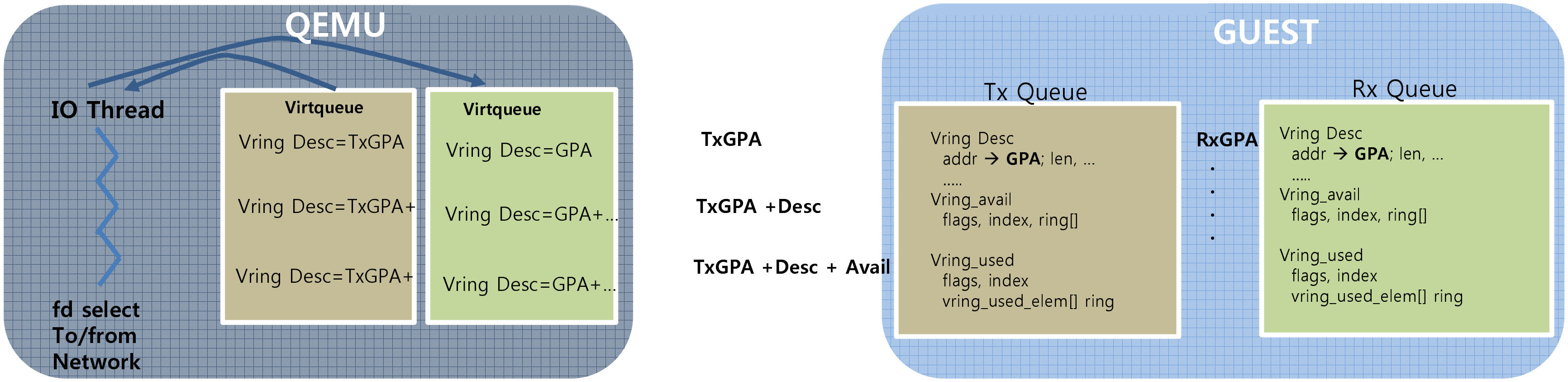
Scatterlist

1 – page_link = page – of virtio_net_hdr
offset = offset within page
length = sizeof virtio_net_hdr
2 – page_link = page – of skb->data
offset = ...
length = skb->len

Vring descriptor

1 – GPA addr of virtio_net_Hdr
length
2 – GPA addr of skb->data
length
NOTIFY

Virtio – moving data



QEMU

IO Thread

fd select
To/from
Network

Virtqueue
Vring Desc=TxGPA
Vring Desc=TxGPA+
Vring Desc=TxGPA+

Virtqueue
Vring Desc=GPA
Vring Desc=GPA+...
Vring Desc=GPA+...

TxGPA

TxGPA +Desc

TxGPA +Desc + Avail

GUEST

Tx Queue

Vring Desc
addr → **GPA**; len, ...
.....
Vring_avail
flags, index, ring[]
Vring_used
flags, index
vring_used_elem[] ring

RxGPA
.
.
.
.

Rx Queue

Vring Desc
addr → **GPA**; len, ...
.....
Vring_avail
flags, index, ring[]
Vring_used
flags, index
vring_used_elem[] ring

QEMU

VirtQueueElement

1. out_addr = GPA virtio_net_hdr
   out_sg.iov_len = virtio_net_hdr length
2. out_addr = GPA skb->data
   out_sg.iov_len = skb->len

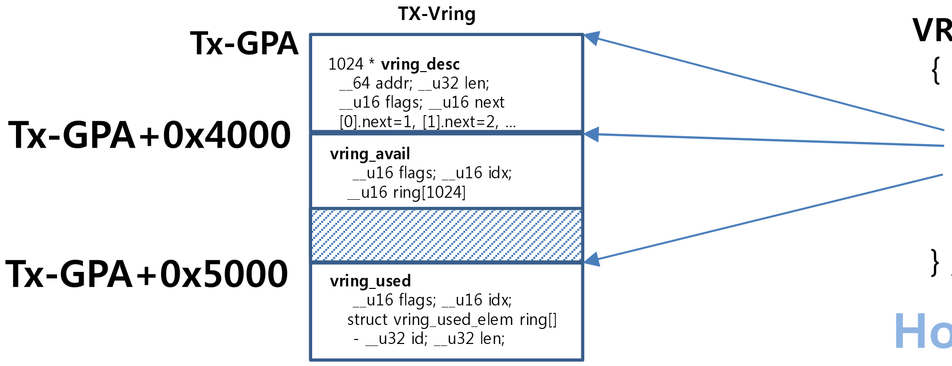
VirtQueueElement

1. out_sg.iov_base = HVA virtio_net_hdr
   out_sg.iov_len = virtio_net_hdr length
2. out_sg.iov_base= HVA skb->data
   out_sg.iov_len = skb->len

qemu_sendv_packet_async
(..., out_sg, out_len, ...
virtio_net_tx_coplete)

Virtio – moving data

Guest – convert GVA -> HPA



TX-Vring

Tx-GPA

Tx-GPA+0x4000

Tx-GPA+0x5000

```
1024 * vring_desc
  __64 addr; __u32 len;
  __u16 flags; __u16 next
  [0].next=1, [1].next=2, ...
```

```
vring_avail
    __u16 flags; __u16 idx;
   __u16 ring[1024]
```

```
vring_used
    __u16 flags; __u16 idx;
   struct vring_used_elem ring[]
    - __u32 id; __u32 len;
```

Guest Vring Operations

```
xmit_skb(...)
  sg_set_buf(scatterlist *sg, ..., virtio_net_hdr)
    - sg->page_link = page
    - sg->offset = page offset
    - sg->legnth ...
  sg_set_buf(scatterlist *sg, ..., skb->data)
    ......
  vq->vring.desc[i].flags = ...
  vq->vring.desc[i].addr = GPA of page
  vq->vring.desc[i].addr = sg->length

  ....
  vq->notify(vq)
    - mmio write – VIRTIO_xxx_QUEUE_NOTIFY
```

Host GPA -> HVA, HVA -> GPA

```
VRing vring;
 {
   unsinged int num=...
   hwaddr desc = Desc Tx-GPA
   hwaddr avail = Desc Tx-GPA + ofst
   hwaddr used = Tx-GPA + ofst
 } /* VRing */
```

Host Vring Operations

```
virtio_net_flush_tx(....)
   virtqueue_pop(q->tx_vq, &elem)
     hwaddr desc_pa = vq->vring.desc;
     i = virtqueue_get_head(vq, vq->last_avail_index++)
          - hwaddr pa = vq->vring.avail + offsetof(VRingAvail, ring[i])
```

GVA -> GVA Base + (pa – GPA)

```
          - return lduw_phys(pa)
     hwaddr desc_pa = vq->vring.desc
```

Convert to GVA

```
     flags = vring_desc_flags(desc_pa, i)
        pa = desc_pa + sizeof(VRingDesc) * i + offsetof(VringDesc, flags)
        return lduw_phys(pa)
          .
          .
```

Convert GPA – &vring_desc->addr to GVA

```
elem->out_addr[elem->out_num] = vring_desc_addr(desc_pa, i)
elem->out_sg[...].iov_len = vring_desc_len(desc_pa, i)
  ...
```

Convert GPA – vring_desc->addr to GVA

```
 elem->out_sg[...].iov_base = cpu_physical_memory_map(elem->out_addr[...], ...)

.....
```

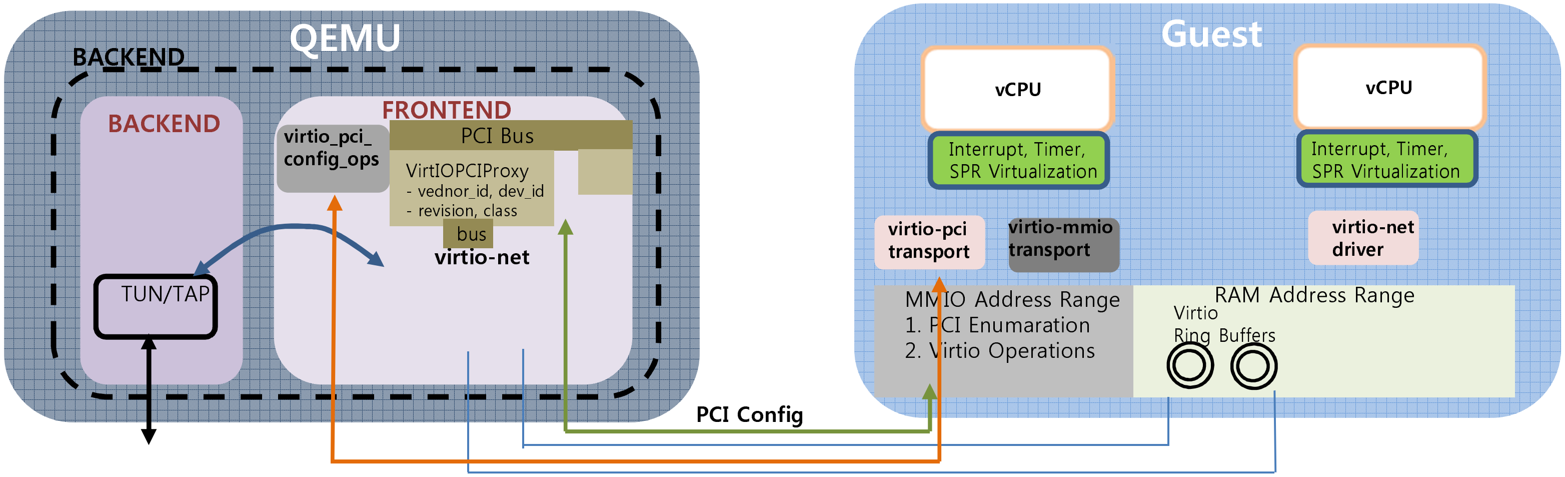
- Tx out Backend
- Notify guest – Tx interrupt completion

Virtio and Device Pass-through

Basic Operation	- Backend/Guest direct access to shared Vring buffers - PIO - Switching at software level - Management Flexibility – internal SDN support ovs-vsctl add-port br0 <phys-intfc> - vSwitch ovs-ofctl – control flows - IRQ bottleneck – QEMU – call into kvm inject Kernel – inject directly	- Direct access to hw memory regions - DMA Support - Switching at hw level – SR-IOV depends on #of Queues - Management Flexibility – external SDN capable - IRQ bottleneck – hw enhancements, posted interrupts, exitless EOI improve things – closer to native
Migration	- Virtio lockless - Saves device state, tracks dirty pages	- QEMU sets ‘unmigratable’, or installs migration blocker - Guest can be holding a lock – deadlock, hw state, ….
Scalability	- Practical limitations – primarily per formance	- Number of Devices limited, limits #VMs - SR-IOV - #of VF - # of queues
Network Performance	- Soft switching – bridge, vSwitch - Several IO HOPS - Can approach near native – 10Ge for few bridged Guest	- Switching done at HW level – hw queues - Performance scales with # of Guests - DMA support - IRQ Passthrough still a problem
Host Performance	- PIO – takes cpu cycles - Exits – few but still - Guest pages swapable	- Guest pinned – can’t swap - Fewer exits - Less PIO
Cloud Environment	- Cloud friendly – migration, SDN, paging	- Not Cloud friendly, great for NFV/RT DPDK, run to completion

Virtio PCI Architecture

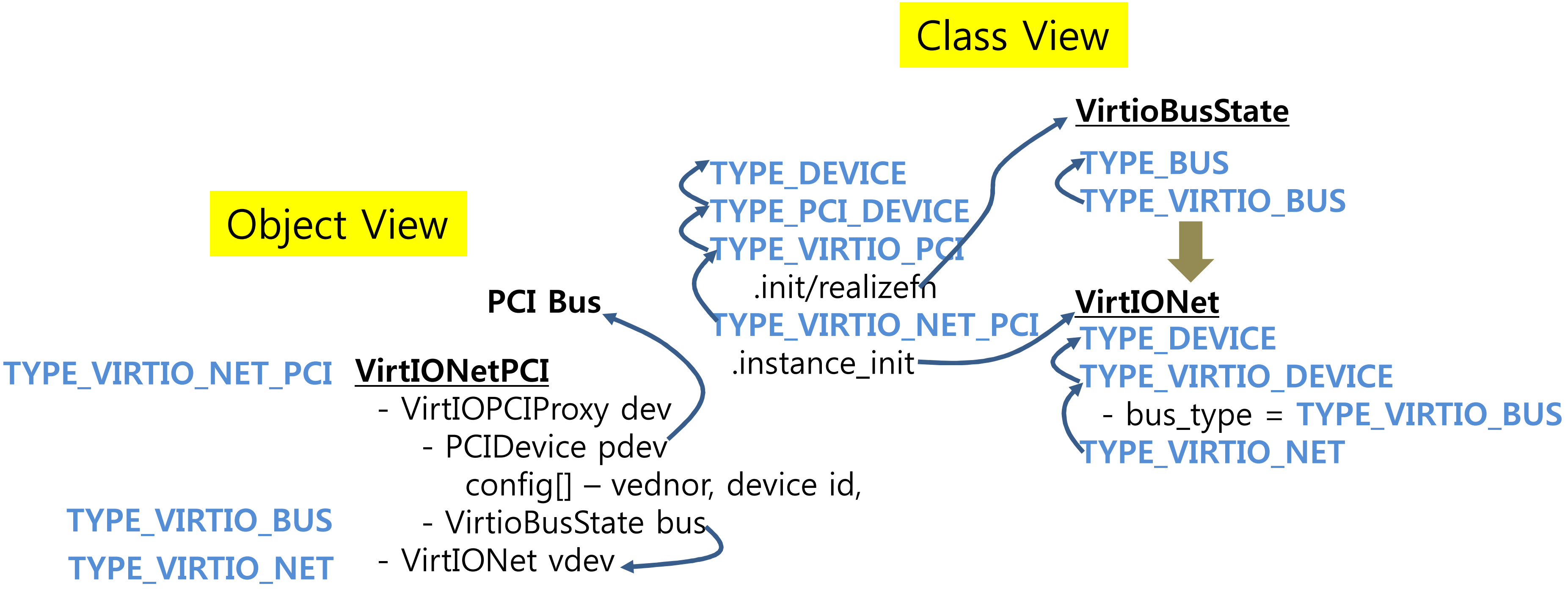
- virtio-net example with QEMU backend – virtio-pci



- Virtio device – combination of mmio & paravirt device
- Before Guest Runs …. QEMU does
  - creates proxy that plugs into PCI Bus
  - During instantiation of VirtIOPCIProxy its
    - PCIDevice vendor id, device id, class, … are set
  - Instantiates virtio-net – bus_type = TYPE_VIRTIO_BUS
    - Plugs into VirtIOPCIProxy bus – TYPE_VIRTIO_BUS
    - Fills in PCI BAR0 type PIO
    - Associates virtio_pci_config_ops with B/D/F BAR0
- Guest
  - Enumarates PCI Bus – discovers virtio-net – via mmio
  - Loads virtio-pci, creates virtio-net device
  - virtio-net driver loads probes virtio-net backend – via mmio

QEMU Object Model

- QEMU Class, Object view of '–device virtio-net-pci'
  - First instantiate Class – C++ Class definition
  - Next the Object – C++ Declare Class variable
  - Realize it– C++ constructor default or defined



Class View

Object View

VirtioBusState

TYPE_BUS

TYPE_VIRTIO_BUS

TYPE_DEVICE

TYPE_PCI_DEVICE

TYPE_VIRTIO_PCI

.init/realizefn

TYPE_VIRTIO_NET_PCI

.instance_init

VirtIONet

TYPE_DEVICE

TYPE_VIRTIO_DEVICE

- bus_type = TYPE_VIRTIO_BUS

TYPE_VIRTIO_NET

PCI Bus

TYPE_VIRTIO_NET_PCI **VirtIONetPCI**

- VirtIOPCIProxy dev
  - PCIDevice pdev
    config[] – vednor, device id,
  - VirtioBusState bus

TYPE_VIRTIO_BUS

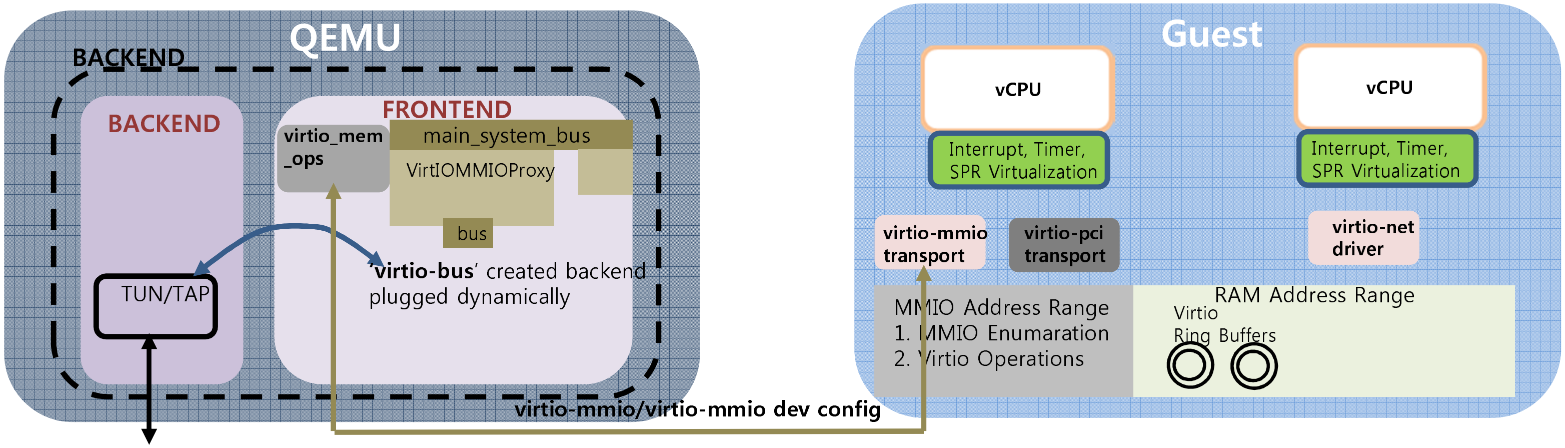
TYPE_VIRTIO_NET - VirtIONet vdev

virtio-mmio transport

- virtio-net example with QEMU backend – virtio-mmio
- Discovery/Probing … like PCI
- Primarily ARM – with Guest QEMU/Guest PCI support – virtio-mmio less use
- Some Use cases
  - ❑ Want your own Machine Model – don’t want PCI, have Device Tree support
  - ❑ Lots of Embedded Devices – simplified machine model
    - o Automotive, Edge Network, Set top Box
  - ❑ virtio-mmio another option

Virtio MMIO Architecture

- Virtio-mmio – example similar to PCI

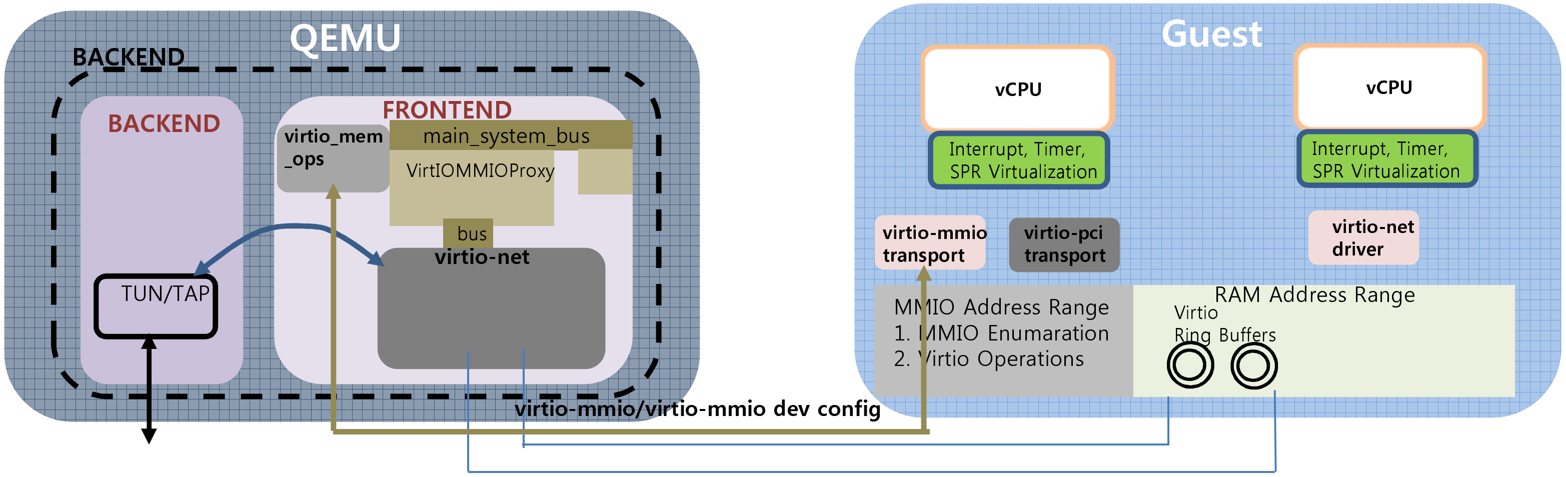


1. Instantiate multiple virtio-mmio devices – no qemu args implicitly done

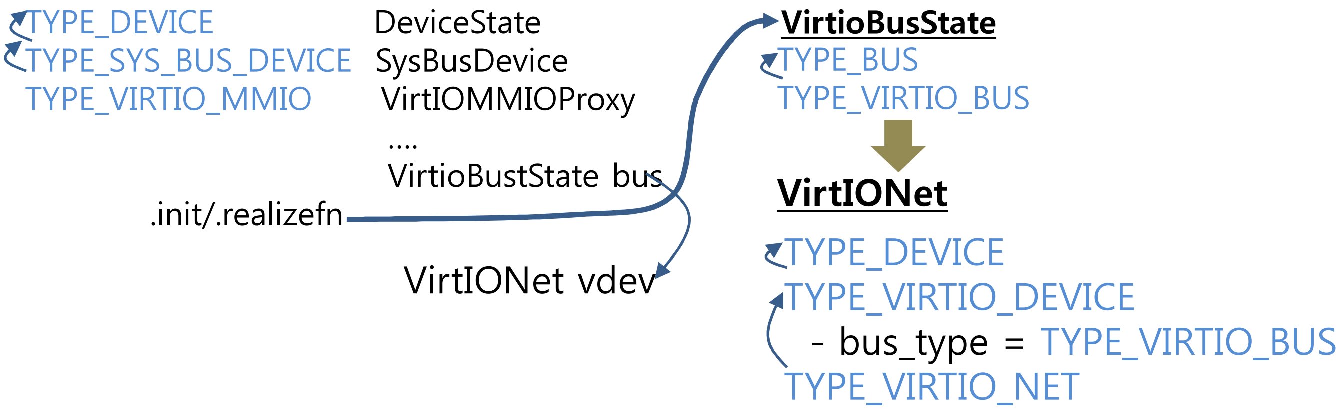
TYPE_DEVICE	DeviceState	**VirtioBusState**
TYPE_SYS_BUS_DEVICE	SysBusDevice	TYPE_BUS
TYPE_VIRTIO_MMIO	VirtIOMMIOProxy	TYPE_VIRTIO_BUS
	....	
	VirtioBustState bus	
.init/.realizefn		

2. Associate resources with each virtio-mmio range
   - MMIO address range a page, interrupt # - only machine models what resources

Virtio MMIO Architecture



3. '-virtio-net-device' – instantiates/plugs TYPE_VIRTIO_NET
   - No transport specified any backend (virtio-net, virtio-blk,…) plug into transport
   - Virtio-net inherits VirtIODevice which sets 'bus_type = TYPE_VIRTIO_BUS'
   - Finds matching bus VirtIOMMIOproxy->bus, plugs TYPE_VIRTIO_NET
   - Finds and binds to QEMU backend – f.e. –netdev type=tap ….



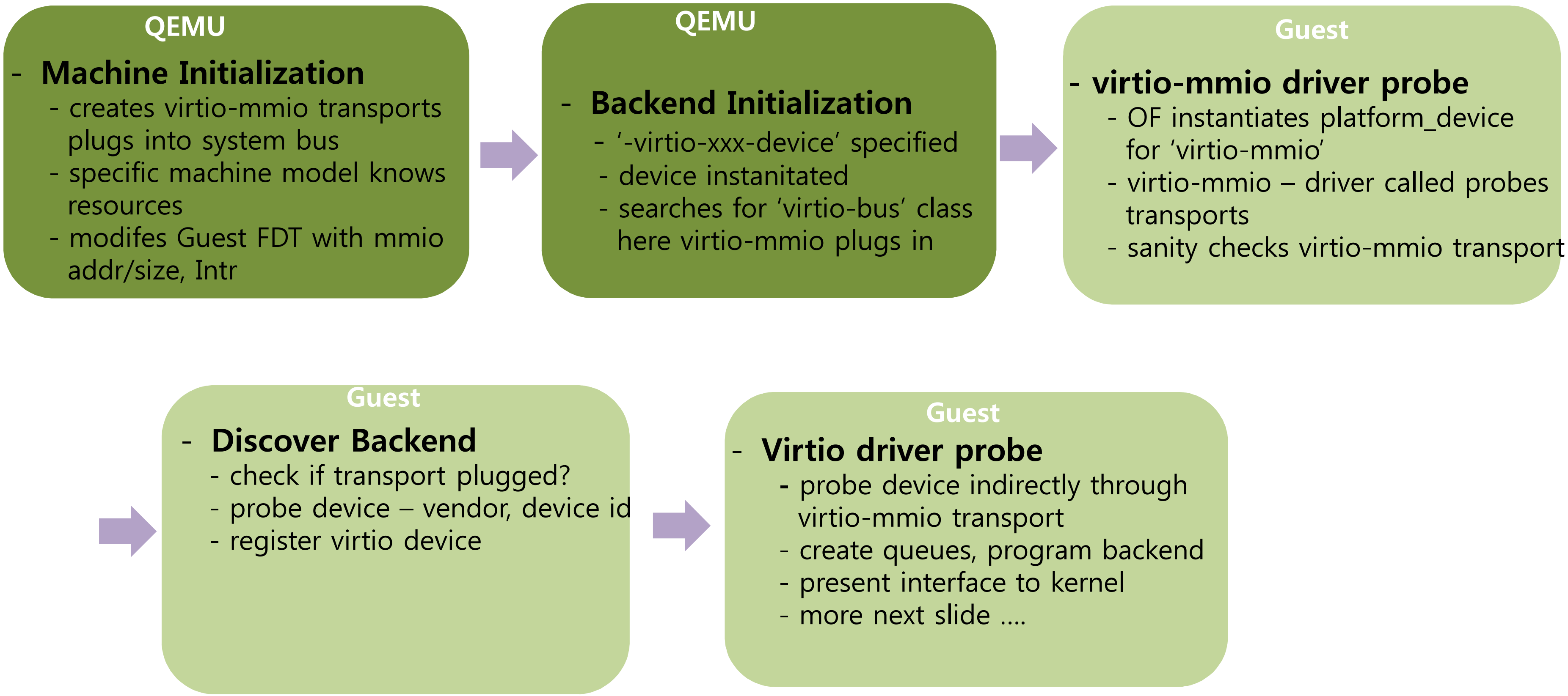
TYPE_DEVICE DeviceState
TYPE_SYS_BUS_DEVICE SysBusDevice
TYPE_VIRTIO_MMIO VirtIOMMIOProxy
….
VirtioBustState bus
.init/.realizefn
VirtIONet vdev

VirtioBusState
TYPE_BUS
TYPE_VIRTIO_BUS

VirtIONet
TYPE_DEVICE
TYPE_VIRTIO_DEVICE
- bus_type = TYPE_VIRTIO_BUS
TYPE_VIRTIO_NET

Guest virtio discovery framework – virtio-mmio view

- Transparent to Guest – enable virtio and mmio
- Device Tree used



QEMU

- **Machine Initialization**
  - creates virtio-mmio transports plugs into system bus
  - specific machine model knows resources
  - modifes Guest FDT with mmio addr/size, Intr

QEMU

- **Backend Initialization**
  - '-virtio-xxx-device' specified
  - device instanitated
  - searches for 'virtio-bus' class here virtio-mmio plugs in

Guest

- **virtio-mmio driver probe**
  - OF instantiates platform_device for 'virtio-mmio'
  - virtio-mmio – driver called probes transports
  - sanity checks virtio-mmio transport

Guest

- **Discover Backend**
  - check if transport plugged?
  - probe device – vendor, device id
  - register virtio device

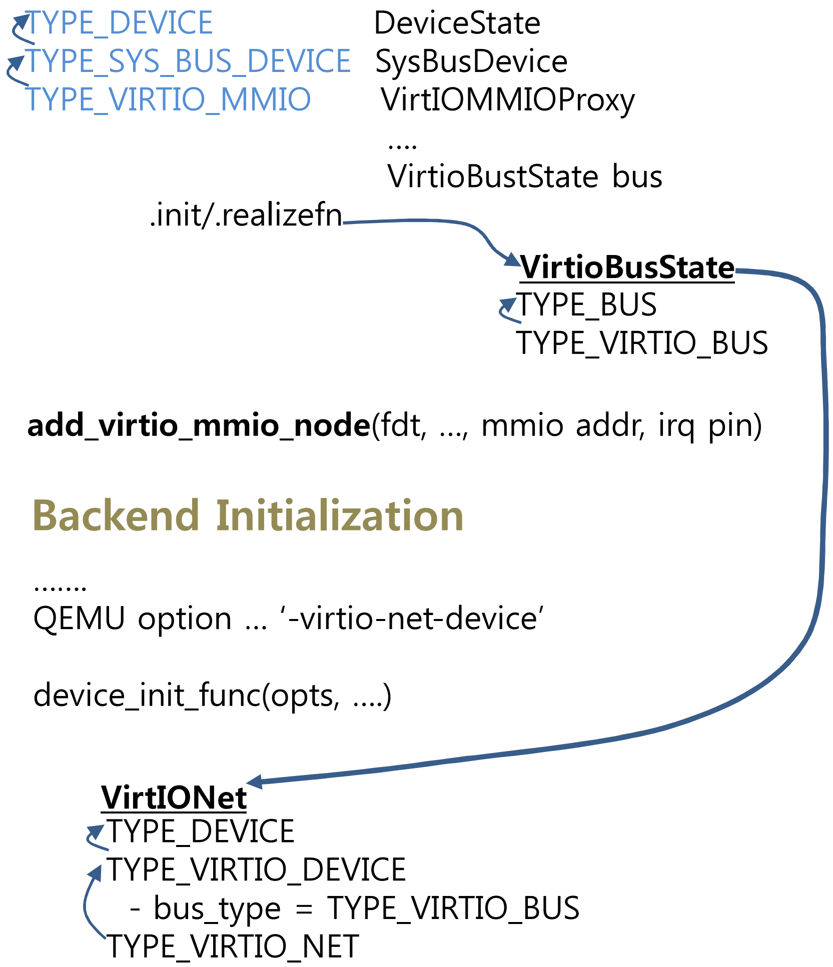
Guest

- **Virtio driver probe**
  - probe device indirectly through virtio-mmio transport
  - create queues, program backend
  - present interface to kernel
  - more next slide ....

Guest virtio discovery framework – virtio-mmio view

Machine Initialization

....
sysbus_create_simple(“virtio-mmio’, base, pic[irq])
....



TYPE_DEVICE DeviceState
TYPE_SYS_BUS_DEVICE SysBusDevice
TYPE_VIRTIO_MMIO VirtIOMMIOProxy
....
VirtioBustState bus

.init/.realizefn

VirtioBusState
TYPE_BUS
TYPE_VIRTIO_BUS

add_virtio_mmio_node(fdt, ..., mmio addr, irq pin)

Backend Initialization

....
QEMU option ... ‘-virtio-net-device’

device_init_func(opts, ...)

VirtIONet
TYPE_DEVICE
TYPE_VIRTIO_DEVICE
- bus_type = TYPE_VIRTIO_BUS
TYPE_VIRTIO_NET

virtio-mmio driver probe

of_platform_populate(..., of_device_id match[], ...)

```
DT Node
virtio-mmio {
   addr, size, irq;
}
```

➡

```
platform_device {
   ....
}
```

```
... virtio_mmio_driver = {
    .probe = virtio_mmio_probe,
    ...
}
```

virtio_mmio_probe(*pdef)
- virtio_mmio_vdev *vm_dev
vm_dev->base = ioremap(virtio-mmio – GPA, size)
virtio_device transport interface, PCI has one too
vm_dev->vdev.config = &virtio_mmio_config_ops
sanity check – mmio to ‘virtio_mem_ops’ handlers
magic = readl(vm_dev->base + VIRTIO_MMIO_MAGIC_VALUE)
version = readl (.....)

Guest virtio discovery framework – virtio-mmio view

Discover Backend

Identify if device plugged, if yes identify device

vm_dev->vdev.id.device = readl(vm_dev->base + VIRTIO_MMIO_DEVICE_ID)
vm_dev->vdev.id.vendor = readl(….)

register the device

register_virtio_device(struct virtio_device dev=vm_dev->vdev)

Ack device found by transport, use transport interface

- dev->config->set_status(… get_stattus() | VIRTIO_CONFIG_S_ACKNOWLEDGE)

find matching driver on virtio bus

- bus_for_each_drv(….)
  - virtio_dev_match(dev, drv)

Ack driver found for device

- dev->config->set_status(…get_status() | VIRTIO_CONFIG_S_DRIVER)

Feature Negotiation – these are key performance features

- Get backed features – be_features = dev->config->get_features(vdev)
- walk driver feature table - check if backend supports – be_features bit set
  - if supported set vdev->features
- select features – vdev->config->finalize_features(vdev->features[])
  - a) backend features not supported by driver don't get selected
  - b) driver features not supported by backend don't get selected
- call driver probe – virtnet_probe()

virtio driver probe

instantiate network device interface

dev = alloc_etherdev_mq(…, # of queues)
….

Various performance features – primarily offload, big packets

- Check supported features – from vdev->features – set dev->hw_features
- Vdev->config->find_vqs(…)
  - Initialize queues – allocated by guest
  - Tell backend GFN of Vring and buffer count for each queue
  - Backend – sets GPA and GPA indexes into Descriptors, Available, Used ring.

Virtio Performance

- When transport/backend are not 'fused' performance features not exported
  - ❑ Due to way QEMU instantiates objects – properties set at TYPE_DEVICE class
  - ❑ After device plugged – properties not set
  - ❑ If transport/backend not fused – properties/performance features not used
  - ❑ Created patch for virtio-mmio – applies when backend plugged
    - ○ https://github.com/mjsmar/virtio_net_fix.git

Virtio Performance

- Performance features
  - ❑ Red Hat multi-queue tap
    - o tap arg – 'queues=n' for scalability
    - o Creates multiple queue virtio/tap tx/rx queue pairs
    - o vCPU scaling for tx/rx, serializes flows - TCP sessions, UDP connections
  - ❑ tx=timer,x-txtimer=<n>uS
    - o Host kicks the backend periodically – limit exits
    - o you can adjust how often backend polls tx virtqueue – tune latency vs. CPU
  - ❑ Offload – bigger pkts few exits, offload to host
    - o Ring descriptors have – 1 for virtio_net_hdr other for data
    - o probe tun/tap for vnet hdr support – for offloads - IFF_VNET_HDR
    - o Probe tun/tap for GSO – TCP,UDP, - TSO, UFO
    - o Options eventually make it to virtio-net net_device 'features'
      - ▪ virtio_net_hdr – flags for CSUM & define range
      - ▪ check skb fragments for for GSO – set vnet_hdr_net gso_type, size

Reconnaissance of Virtio: What’s new and how it’s all connected?

Q & A

Thank you.

Mario Smarduch
Senior Virtualization Architect
Open Source Group
Samsung Research America (Silicon Valley)
m.smarduch@samsung.com