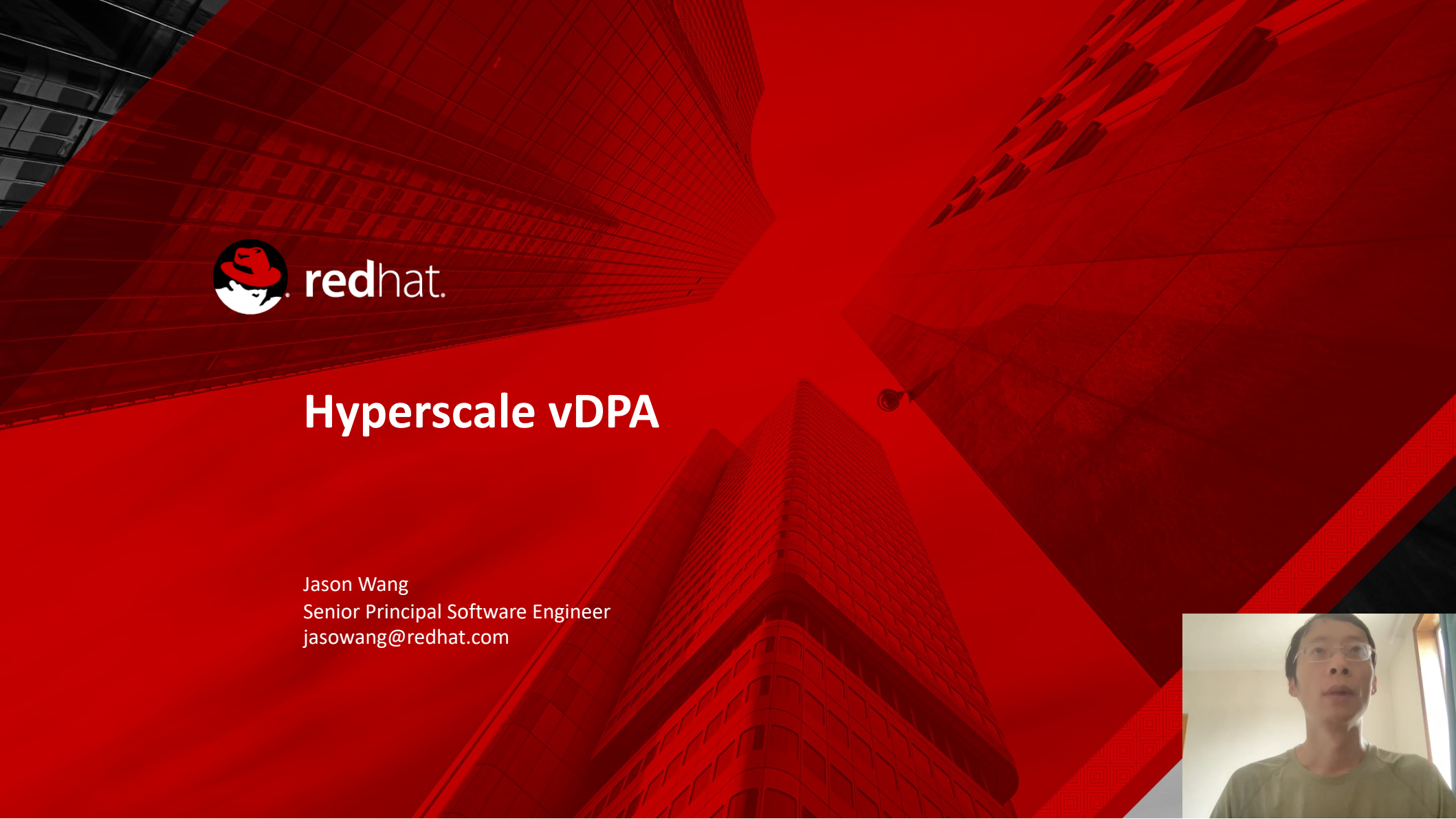


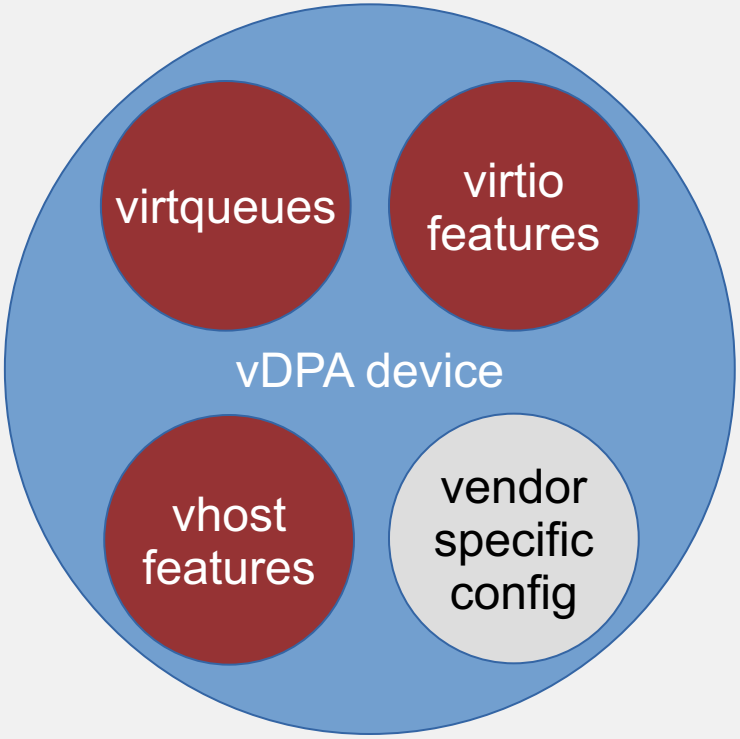
Hyperscale vDPA

Jason Wang
Senior Principal Software Engineer
jasowang@redhat.com

Outline

- vDPA architecture overview
- Demand for hyper-scalability
- Scale vDPA instances
- Secure DMA environments
- Interrupt scalability
- Provisioning capability
- Status & Summary

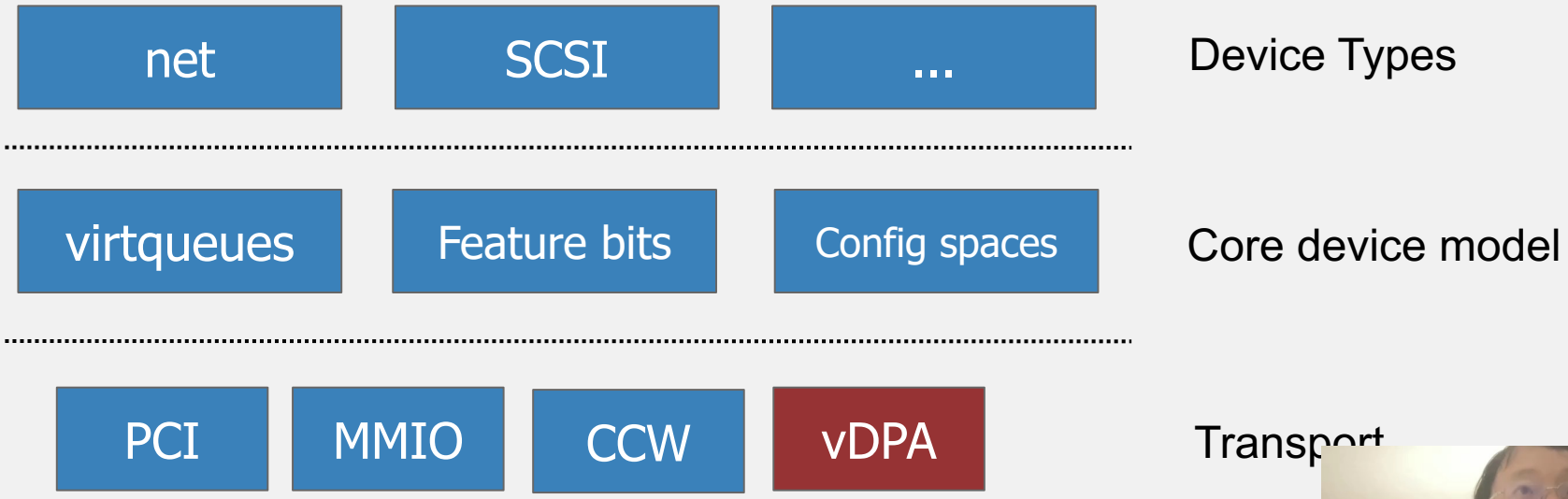
vDPA device – hardware perspective



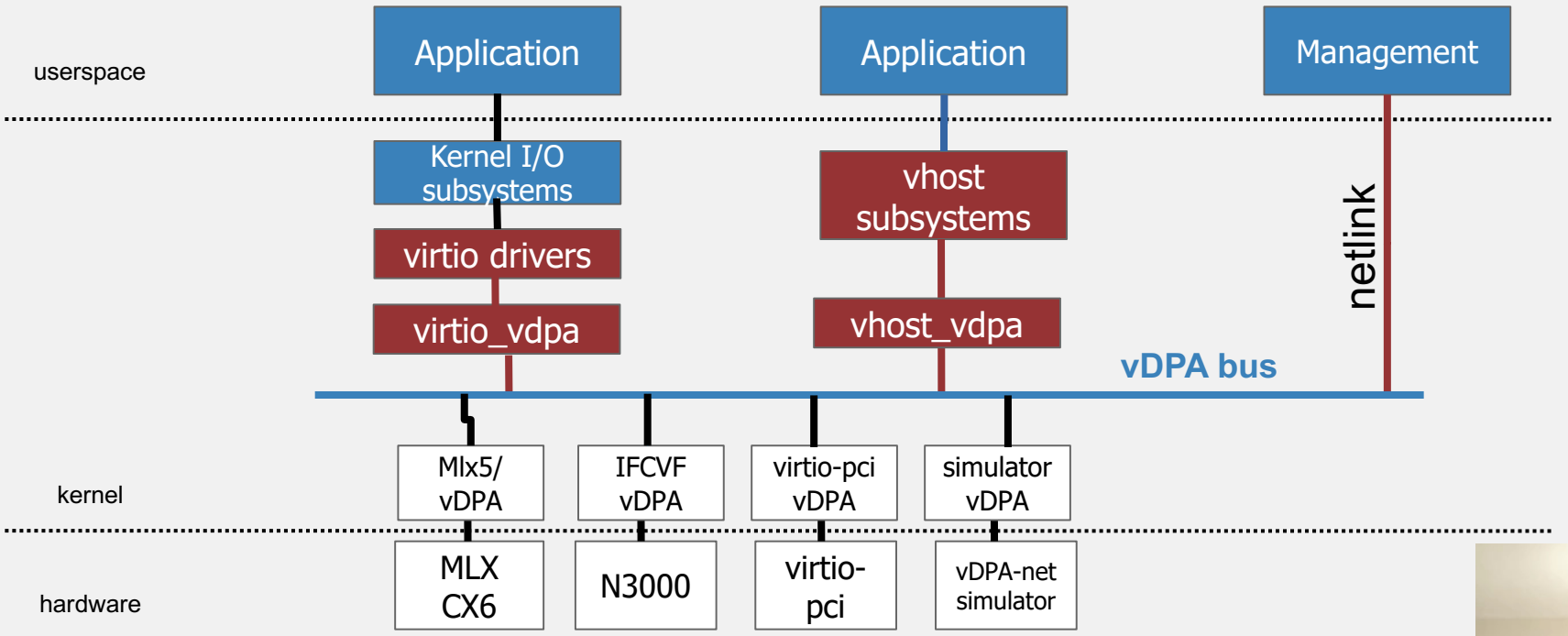
vDPA devices (parents)

- Intel N3000 - PCI device – blk and networking
- Mellanox CX-6 - vDPA on top of vendor specific hardware architecture – networking so far
- **Virito-pci** – Red Hat as a “vendor” – all virtio devices
- VDUSE - userspace vDPA device (RFC) – start from blk
- vDPA simulator - kernel vDPA simulator (on the road for production environment) – blk and networking
- More vDPA parents are on the road

vDPA - virtio architecture perspective



vDPA software architecture



Demand for hyper-scalability

- The need for density
    - Containerized workload became popular
    - Want 10K+ or 100K+ vDPA instances
    - Finer grain (e.g split/slice VF)
- Provisioned
    - Flexibility
    - Saving resources

Challenges

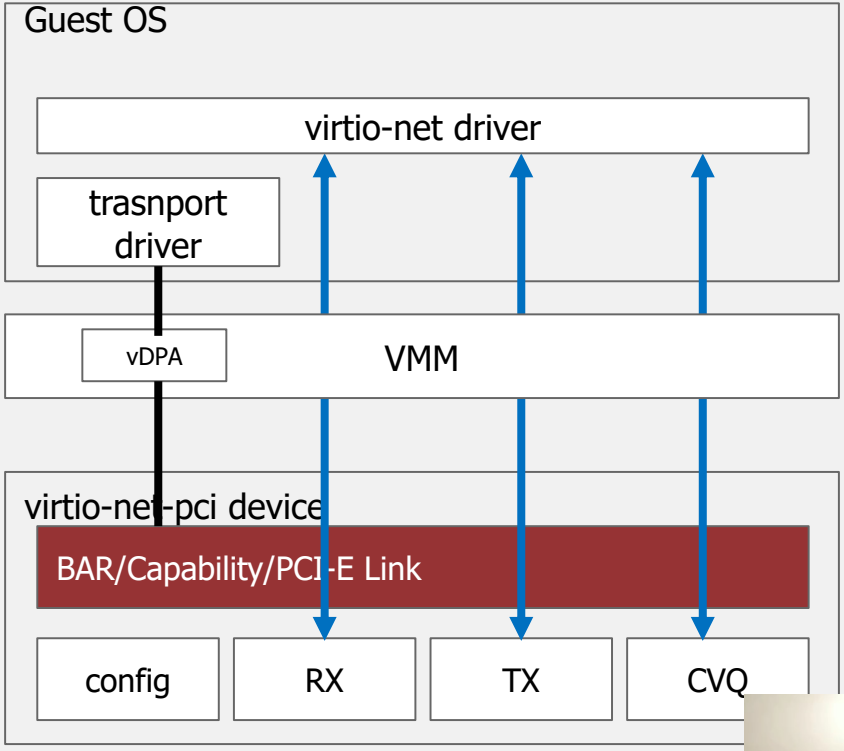
- Fine grain and light weight vDPA instance
- Secure DMA context for each vDPA instance
- Interrupt scalability, E.g PCI-E allows only 2048 MSI-X entries
- Provisioned, on demand creating of vDPA instance

Lightweight vDPA instance

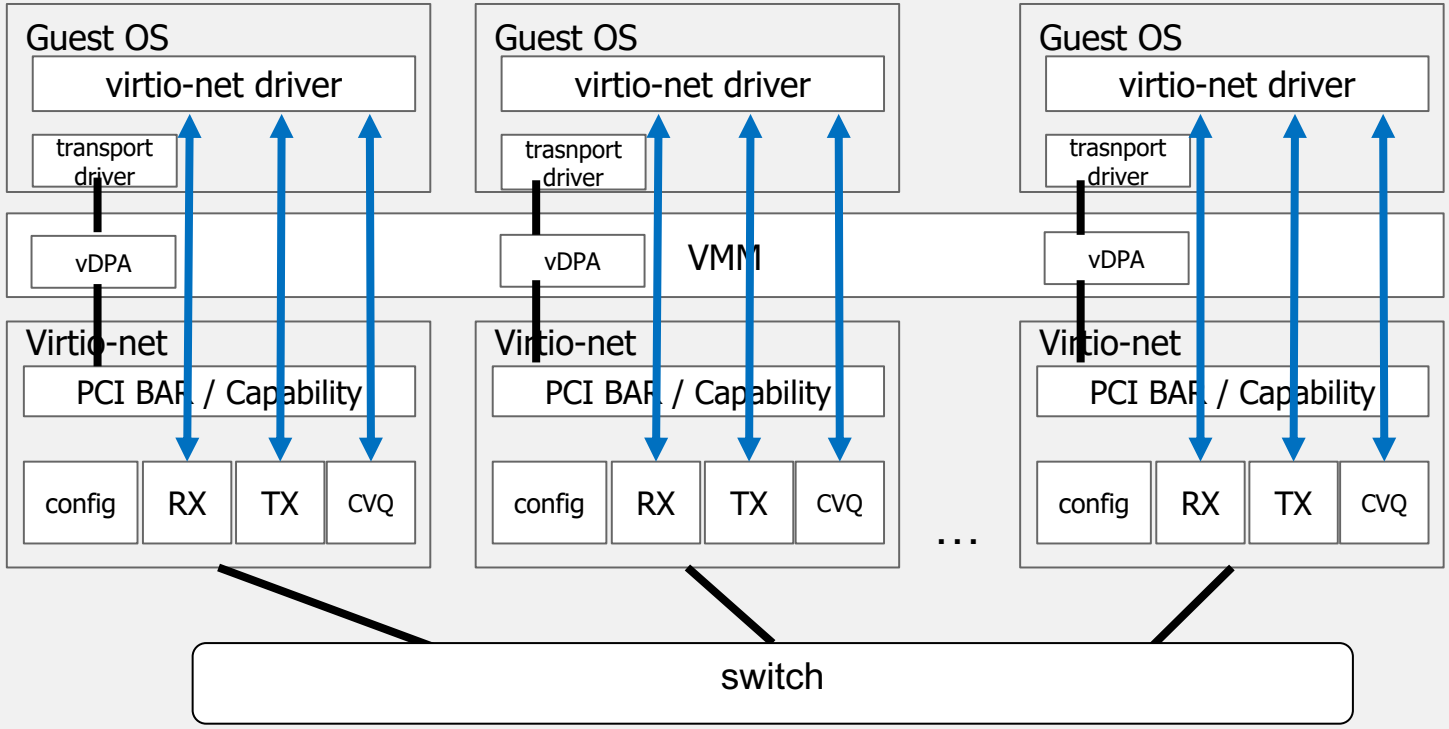
- As minimal resources (hardware/transport) as possible
- Software mediation
    - Software scales better/easier than hardware

Virtio-net via PCI

- RX/TX queue pair
- Optional CVQ
  - Filters/MAC/MQ
- Device configuration space
- Transport
  - Configuration
  - Discovery

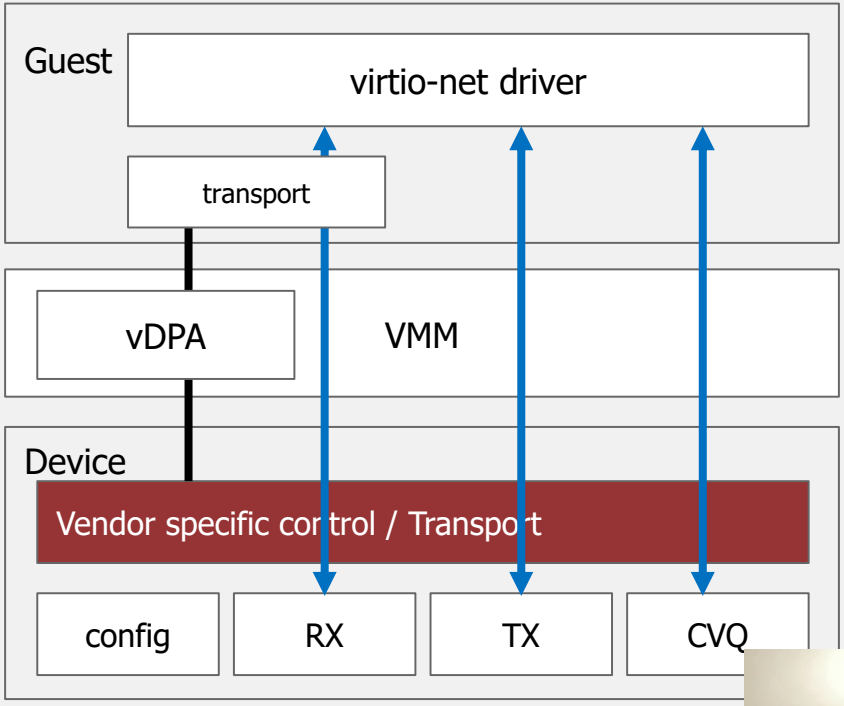


Scaling virtio-net-pci instances



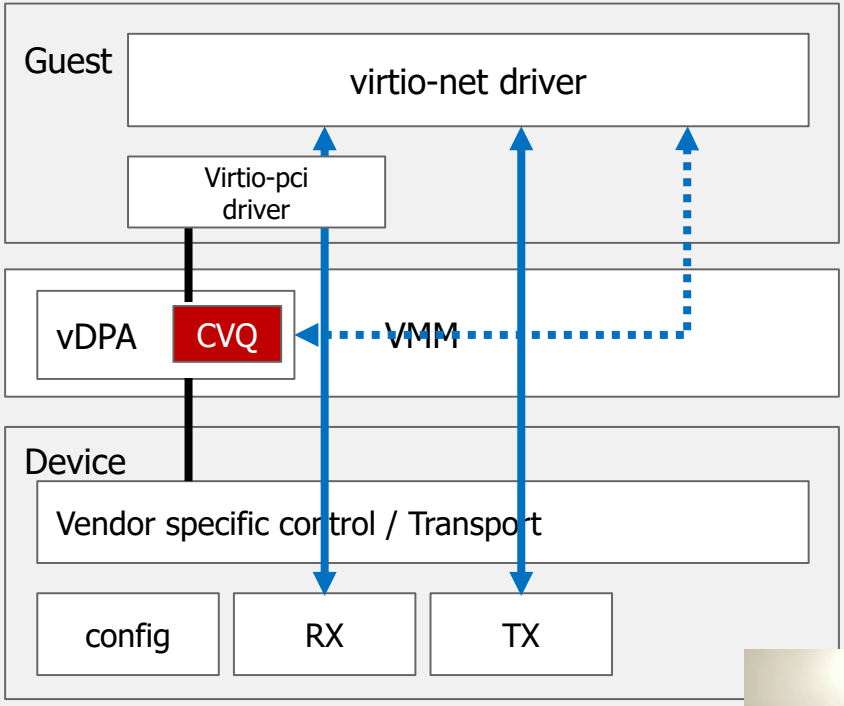
Virtio-net via vDPA

- RX/TX queue pair
- Optional CVQ
    - Filters/MAC/MQ
- Device configuration space
- Transport
    - Configuration
    - Discovery

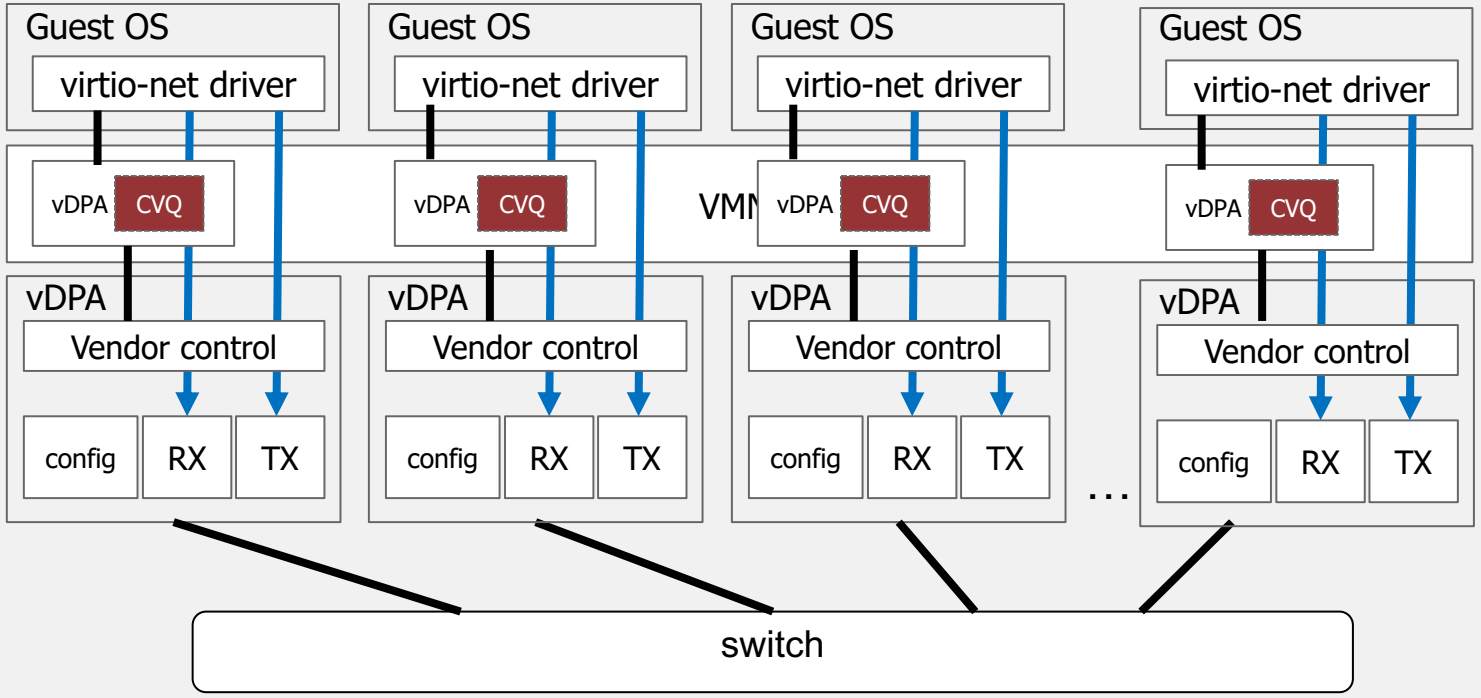


vDPA for the hyper-scalability (software CVQ)

- No hardware CVQ
- CVQ features is implemented in a vendor specific way
- vDPA presents software CVQ
  - Decode CVQ commands
  - Translate them to vendor commands
- Save one virtqueue

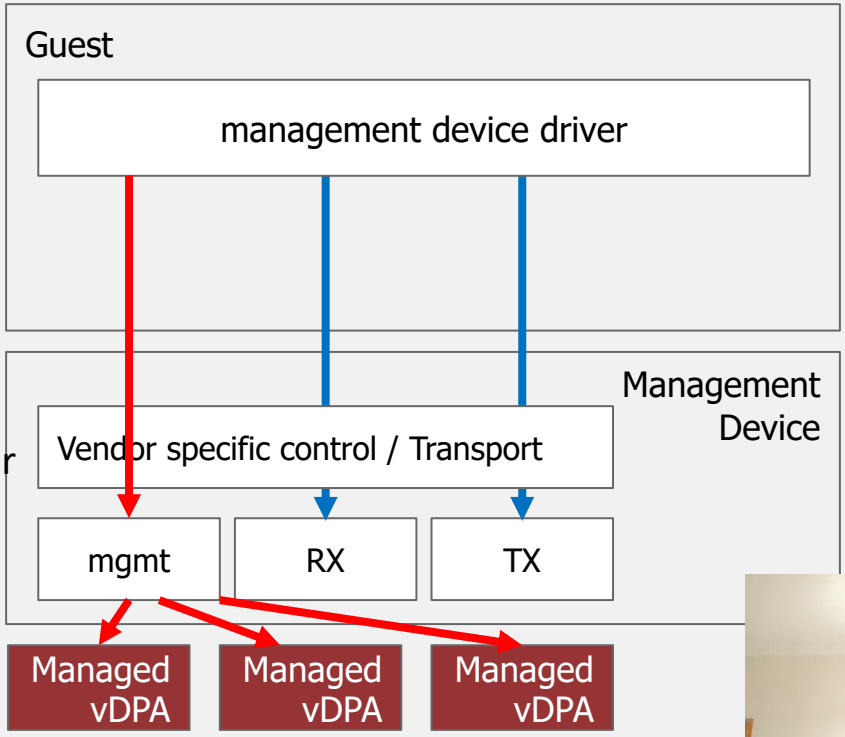


vDPA for the hyper-scalability (software CVQ)

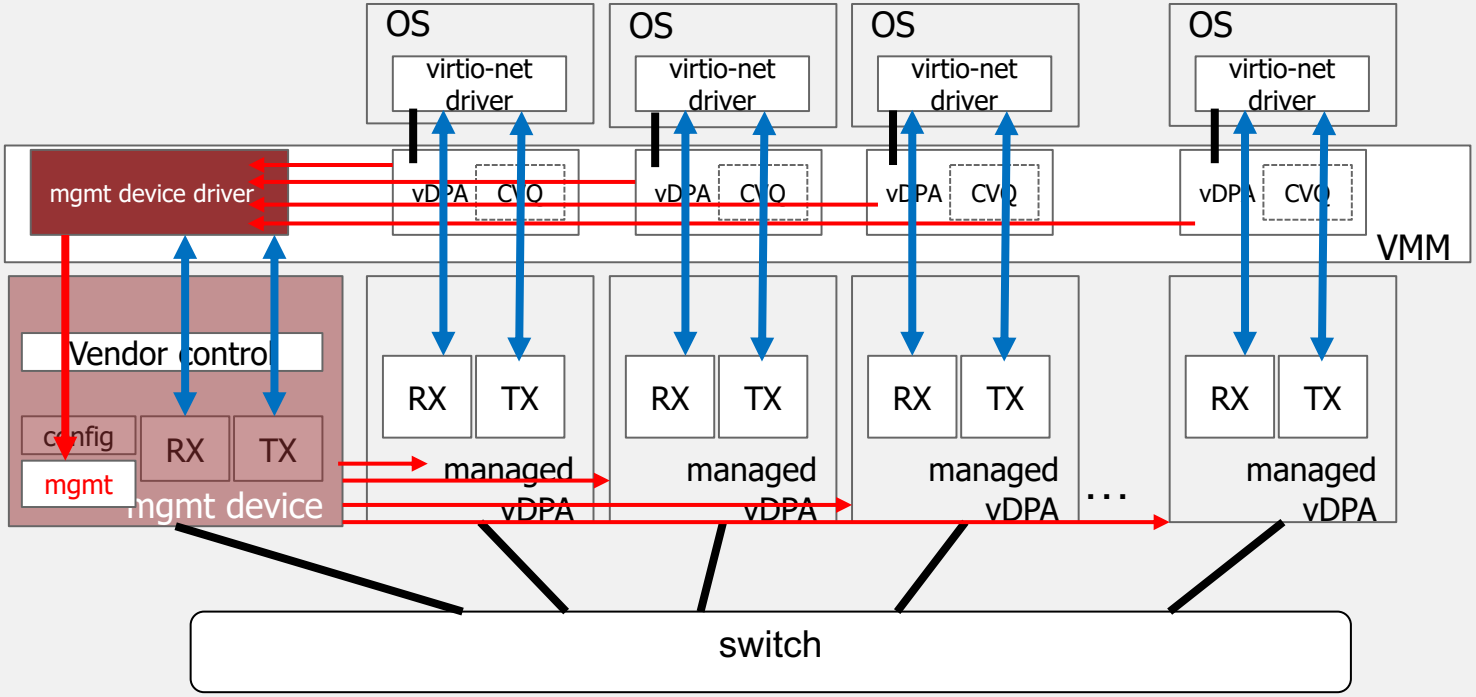


vDPA for the hyper-scalability (managed device)

- Management device provides vendor specific control for transporting managed device
- Vendor specific transport via Management device
- Help to keep vDPA minimal
- Complicate the software part
  - Synchronization is required for concurrent requests for different managed vDPA
  - Or we may need QOS if a queue is used



vDPA for the hyper-scalability (managed device)



Virtio spec for scaling

- Transport specific support
    - Device command capabilities
    - Managed devices
- Using virtqueue as transport
    - Management virtqueue

Device specific command capability

```
struct virtio_net_ctrl {
        u8 class;
        u8 command;
        u8 command-specific-data[];
        u8 ack;
};
```

- partial transport of virtio_net_ctl
- one less virtqueue
- one more capability
- less flexible than cvq

```
/* Device specific command */
#define VIRTIO_PCI_CAP_DEVICE_CMD_CFG   11

struct virtio_pci_device_cmd_cap {
        struct virtio_pci_cap cap;
        u8 class;
        u8 command;
        u8 command-specific-data[256];
        u8 execute;
        u8 ack;
};
```

Managed Device capability

- minimal extension for the spec
- but transport specific

```
/* Managed device configuration */
#define VIRTIO_PCI_CAP_MANAGED_DEV_CFG   12

struct virtio_pci_managed_dev_cap {
        struct virtio_pci_cap cap;
        le32 num_devices;       /* read-only for driver */
        le32 device_select;      /* read-write */
…
```

Managed Device capability - example

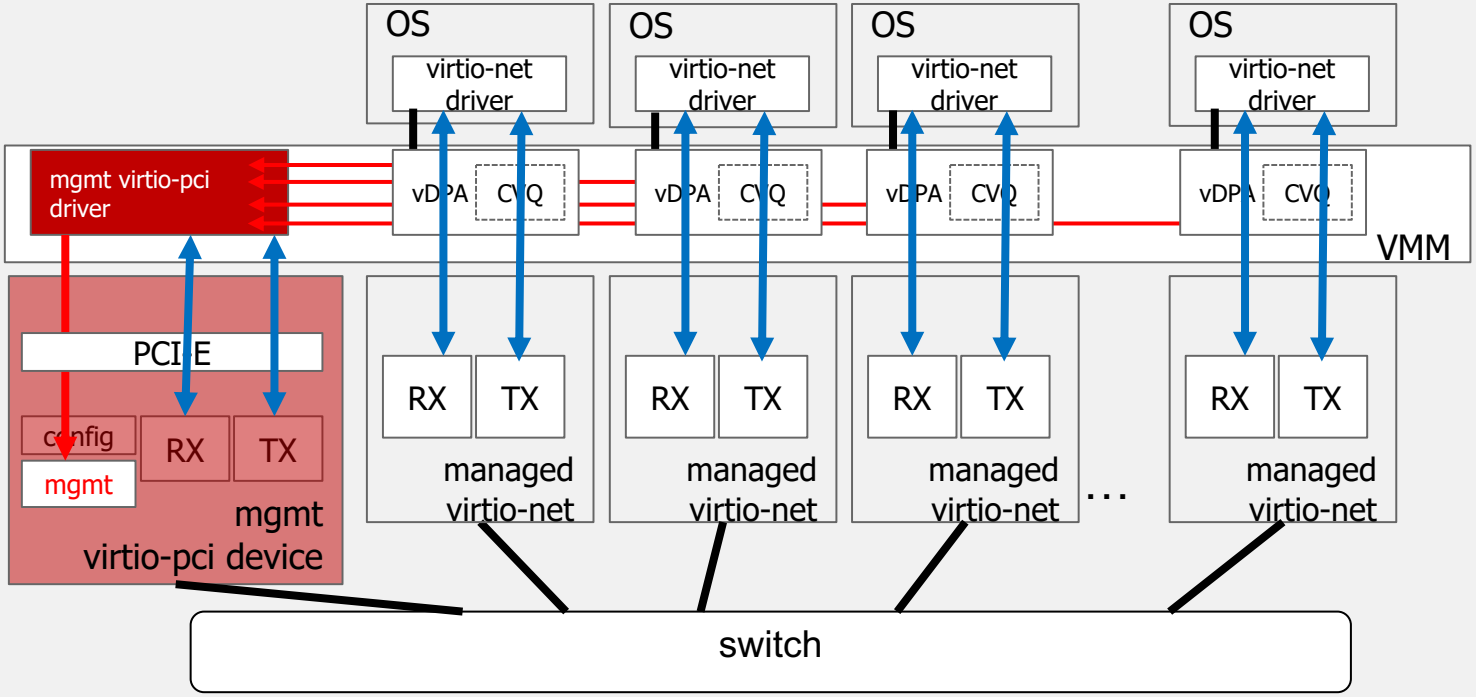
- Configurating virtqueue 0 address for managed device 1

```
iowrite(1, &mgmt->device_select);
iowrite(0, &cfg->queue_select);
Iowrite(0x8000, &cfg->queue_desc_lo);
```

- Configurating management device itself (device 0 is reserved)

```
iowrite(0, &mgmt->device_select);
```

Virtio-pci for the hyper-scalability



Virtqueue as transport

- dedicated virtqueue for the management device
- transport for the managed device
- commands for basic facilities & control vq commands
- management device is probed in other transport (e.g PCI)

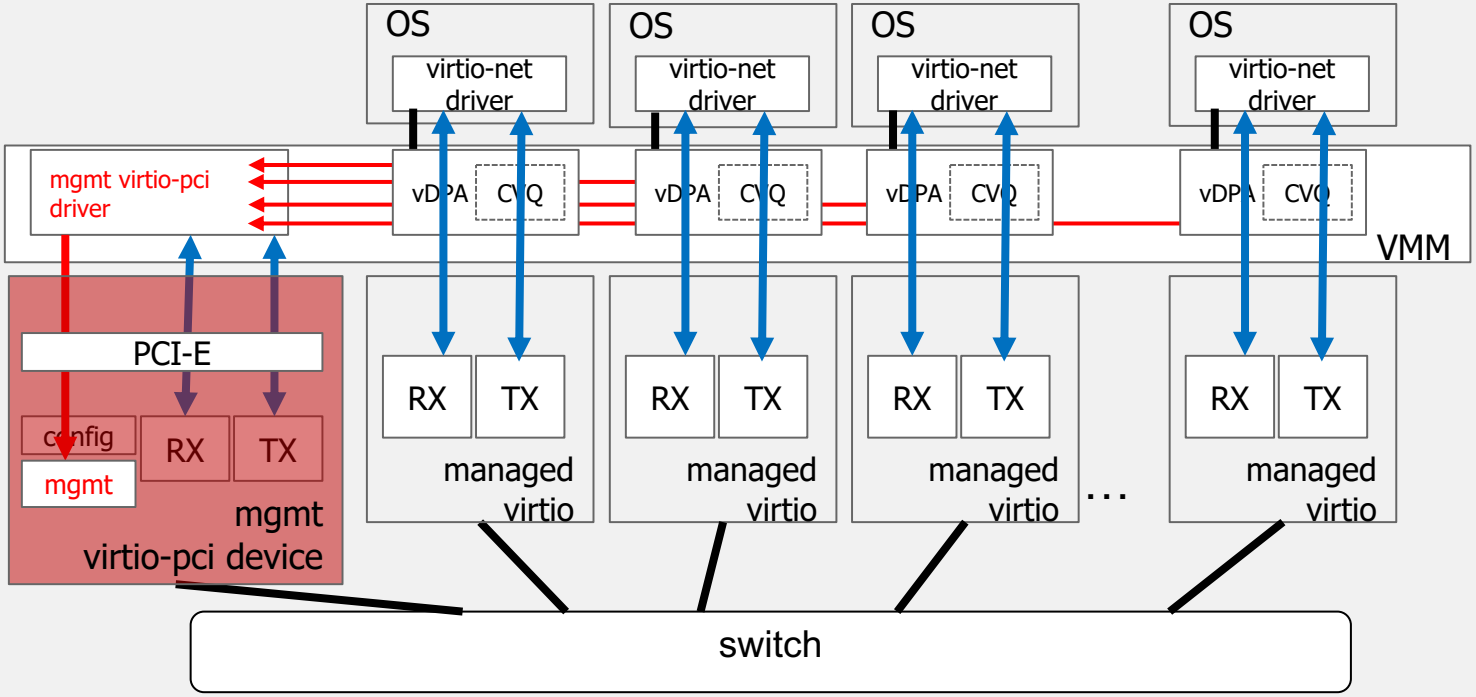
- transport independent
- more flexible than the capability
- async interface
- more complicated
- may require QOS

```
struct virtio_admin_ctrl {
        u64 device_id;
        u16 class;
        u16 command;
        u8 command-out-data[];
        u8 ack;
        u8 command-in-data[]
};
```

Virtqueue as transport – command classes

```
#define VIRTIO_ADMIN_CTRL_FEAT    3
#define VIRTIO_ADMIN_CTRL_STATUS    4
#define VIRTIO_ADMIN_CTRL_GENERATION    5
#define VIRTIO_ADMIN_CTRL_CONFIG    6
#define VIRTIO_ADMIN_CTRL_MSI    7 /* MSI-X storing */
…
#define VIRTIO_ADMIN_CTRL_VQ_ADDR    9
#define VIRTIO_ADMIN_CTRL_VQ_ENABLE    10
#define VIRTIO_ADMIN_CTRL_VQ_SIZE    11
#define VIRTIO_ADMIN_CTRL_VQ_NOTIFY    12 /* Notification/Door
```

Virtio for the hyper-scalability



Secure DMA context for vDPA

- isolate DMAs among vDPA instance
- transport/platform specific method
- vendor specific method
- isolating at virtio level

Transport specific method – PCI-E

- PASID (Process Address Space ID) – PCI-E
- Assign PASID per vDPA instance
    - Even per virtqueue
- Platform IOMMU support (PASID capable)
- vDPA provides vendor specific way for configurating PASID

- Leverage platform features
- Platform dependent

Vendor specific method – Device MMU

- Device has its own MMU
    - IOVA -> transport specific DMA address
- DMA is isolated at vDPA instance level
- work with/without transport specific DMA isolation method (e.g PASID)

- platform independent
- more flexible
- complicated in the implementation

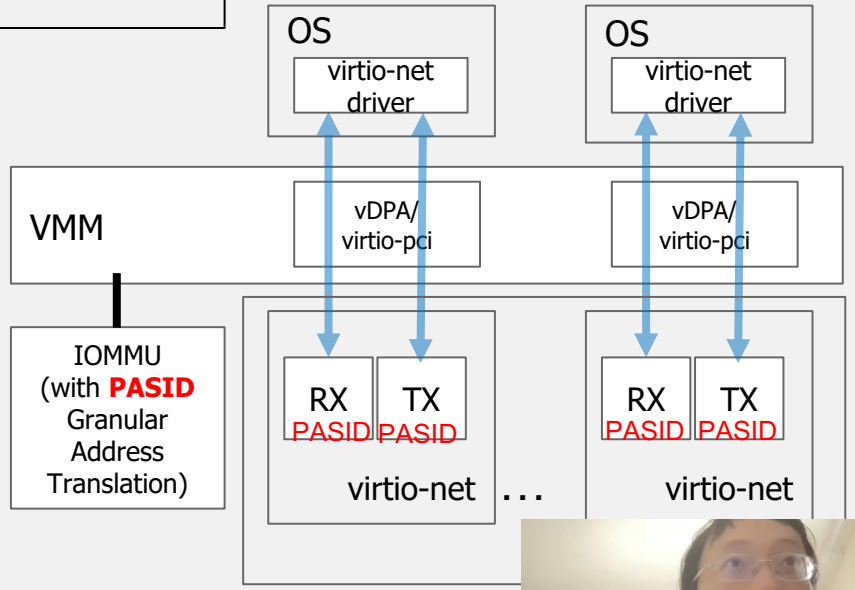
Fine grain DMA isolating in the spec

- Spec support for the transport specific DMA isolation method (PASID)
- DMA isolation at virtio

Spec support for PASID (virtio-pci)

- leverage platform features
- platform dependent
- simple & standard

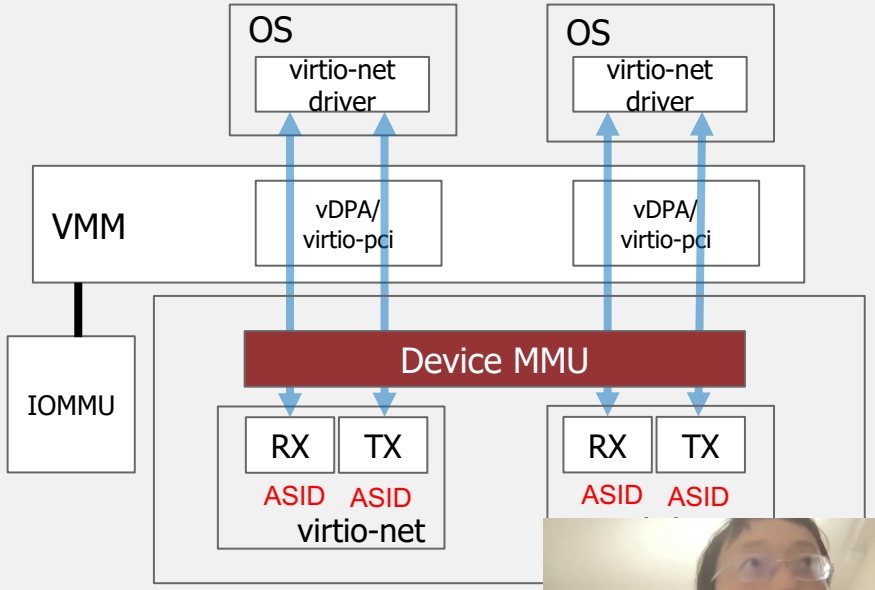
```
/* PASID configuration */
#define VIRTIO_PCI_CAP_PASID        11

struct virtio_pci_cfg_cap {
        struct virtio_pci_cap cap;
        le16 queue_select;      /* read-write */
        le32 {
          queue_pasid : 20;     /* read-write */
          reserved : 12;        /* read-write */
        };
        u8 pasid_enable;        /* read-write */
};
```



Spec support for device MMU

- Define device MMU in the spec
- DMA translation became two stages
  - Stage one: device IOMMU (IOVA -> intermediate address)
  - Stage two: platform IOMMU (intermediate address -> physical)
- Two possible interfaces:
  - Queue based (similar to virtio-iommu)
  - Page table based

- platform independent
- more flexible
- complicated in the implementation
- standard



Interrupt scalability

- Transport limitation
    - E.g 2048 MSI-X entries per PCI-E device
    - No MSI-X support in MMIO transport
- Scale the #MSI-X entries

Interrupt scalability - vDPA

- Vendor specific way to
  - store MSI-X entries
  - mask and unmask MSI-X entries

Interrupt scalability – spec support

- Virito specific MSI-X message storing (transport specific or virtqueue)

```
/* MSI-X configuration */
#define VIRTIO_CPI_CAP_MSIX_CFG        10

struct virtio_pci_msix_cap {
        struct virtio_pci_cap cap;
        u8 enable;                      /* read-write */
        le16 num_vectors;               /* read-only for driver */
        le16 vector_select;             /* read-write */
        le32 msix_address_high;         /* read-write */
        le32 msix_address_low;          /* read-write */
        le32 msix_data;                 /* read-write */
        u8 mask;                        /* read-write */
};
```

Could be done via management virtqueue as well:

```
#define VIRTIO_ADMIN_CTRL_MSI    7
```

vDPA provisioning

- Config specified via the netlink
    - E.g the mac address, # queue pairs etc
- Provisioning is done at vDPA parent level

Spec support for provisioning

```
/* Managed device configuration */
#define VIRTIO_PCI_CAP_MANAGED_DEV_CFG   12

struct virtio_pci_managed_dev_cap {
        struct virtio_pci_cap cap;
        u32 num_devices;          /* read-only for driver */
        le32 device_select;       /* read-write */
        le32 config_select;       /* read-write */
        u8 config[256];           /* read-write */
        u8 create;                /* read-write */
        u8 create_ok;             /* read-only for driver */
};
```

Summary

- discuss the approaches to scale
  - vDPA/virtio instances
  - secure IOMMU contexts
  - interrupt
  - provisioning
- other technologies:
  - devices sharing or scheculing
  - shared virtqueue.
- none of these approaches comes for free
- vendor need to balance the pron/cons

Reference

- Virtio spec
  - https://docs.oasis-open.org/virtio/virtio/v1.1/csprd01/virtio-v1.1-csprd01.html
- RFCs
  - Transport managed device via admin virtqueue https://markmail.org/message/7cwvyxcbjxn5zynk
  - PASID support for virtio-pci https://markmail.org/message/resayicr2ftbqk77
  - MSI-X storing for virtio-pci https://markmail.org/message/fjijwezbvyrsqmzk
  - Transport managed virtio via virtio-pci https://markmail.org/message/g4uuzfvpudfj2sbb

Please visit

https://vdpa-dev.gitlab.io/