Agenda

- About us
- Why para-virtualize RDMA
- Project overview
- Open issues
- Future plans

ORACLE redhat.

About us

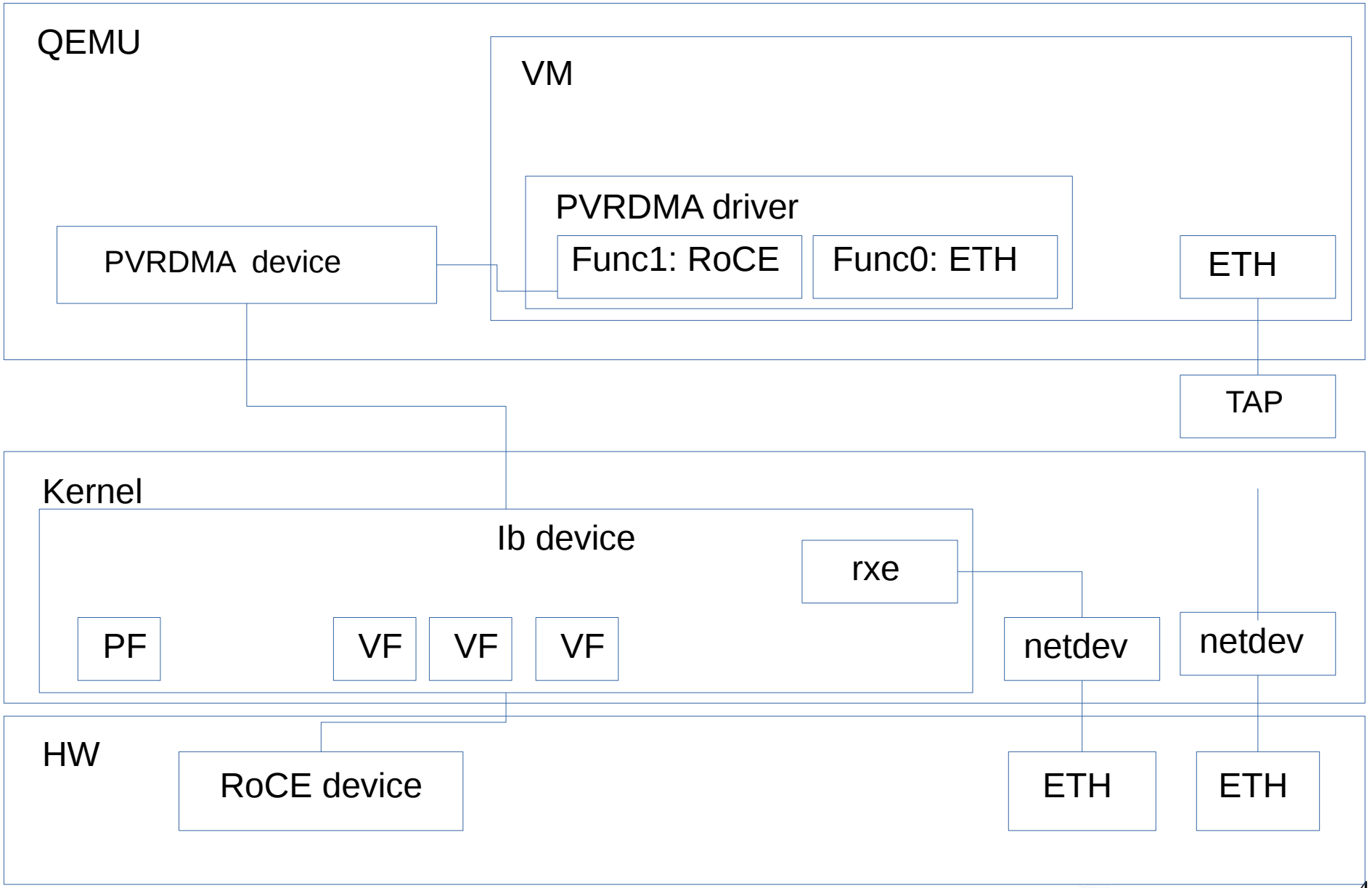
- Marcel from KVM team in Redhat
- Yuval from Networking/RDMA team in Oracle
- This is a shared-effort open source project to implement a paravirt RDMA device.
  - We implement VMware's pvrdma device on QEMU since it save us time (guest drivers and lib)

ORACLE redhat.

Why para-virtualize RDMA

- Make Guests device-agnostic while leveraging hw capabilities
- SRIOV becomes optional (by using multiple GIDs)
- Migration support
- Memory overcommit (without hw support)
- Fast build of testing cluster

Overview



ORACLE redhat.

Expected performance

- Comparing:
  - Ethernet virtio NICs (state of the art para-virt)
  - PVRDMA RoCE with soft Roce backend
  - PVRDMA RoCE with Phys RDMA VF.
- Looking for throughput of small/large packets

PCI registers

BAR 0: MSIX

Vec0: RING EVT | Vec1: Async EVT | Vec3: Cmd EVT

BAR 1: Regs

VERSION | DSR | CTL | REQ | ERR | ICR | IMR | MAC

BAR 2: UAR

QP_NUM | SEND|RECV || CQ_NUM | ARM|POLL

ORACLE redhat.

PCI registers – BAR 1

BAR 1: Regs

VERSION	DSR	CTL	REQ	ERR	ICR	IMR	MAC

- Version
- DSR: Device Shared Region (command channel)
  - CMD slot address
  - RESPONSE slot address
  - Device CQ ring
- CTL
  - Activate device
  - Quiesce
  - Reset
- REQ
  - Execute command at CMD slot
  - The result will be stored in RESPONSE slot
- ERR
  - Error details
- IMR
  - Interrupt mask

PCI registers – BAR 2

BAR 2: UAR

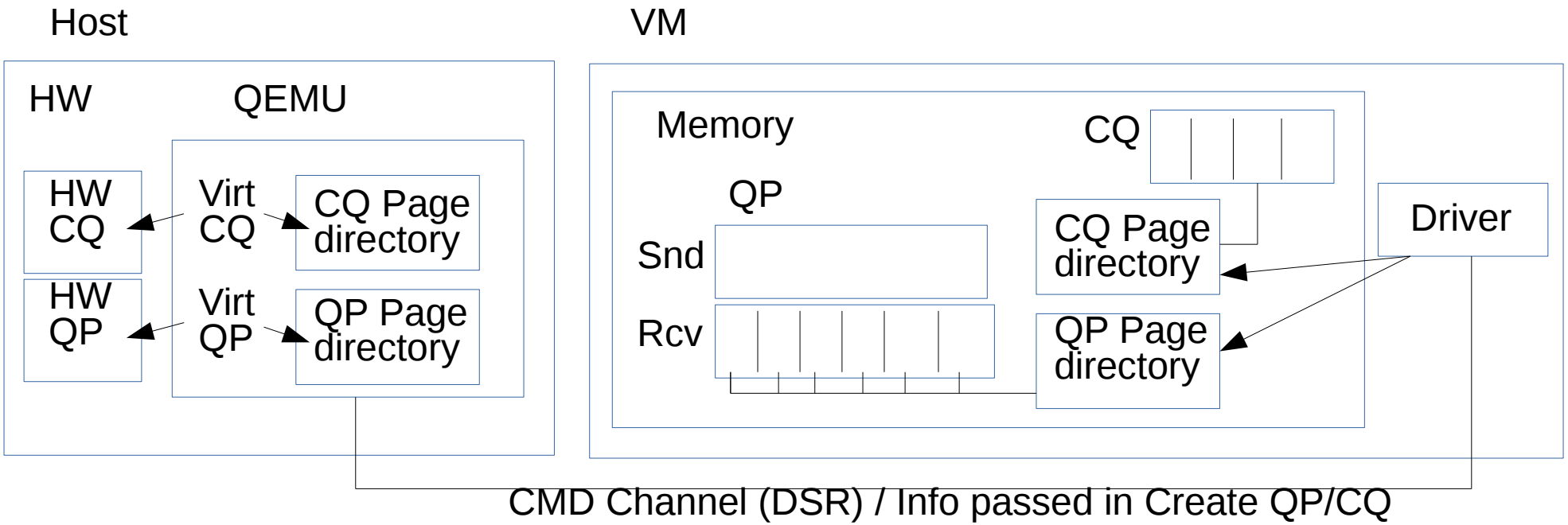
QP_NUM	SEND|RECV	CQ_NUM	ARM|POLL

- QP operations
  - Guest driver writes to QP offset (0) QP num and op mask
  - The write operation ends after the operation is sent to the backend device
- CQ operations
  - Guest driver writes to CQ offset (4) CQ num and op mask
  - The write operation ends after the command is sent to the backend device
- Only one command at a time

ORACLE redhat.

Resource manager

- All resources are “virtualized” (1-1 mapping with backend dev)
  - PDs, QPs, CQs,…
- The pvrdma device shares the resources memory allocated by the guest driver by means of a “shared page directory”



Flows – Create CQ

- Guest driver
  - Allocates pages for CQ ring
  - Creates page directory to hold CQ ring's pages
  - Initializes CQ ring
  - Initializes 'Create CQ' command object (cqe, pdir etc)
  - Copies the command to 'command' address (DSR)
  - Writes 0 into REQ register
- Device
  - Reads request object from 'command' address
  - Allocates CQ object and initialize
    - CQ ring based on pdir
  - Creates backend(HW) CQ
  - Writes operation status to ERR register
  - Posts command-interrupt to guest
- Guest driver
  - Reads HW response code from ERR register

ORACLE redhat.

Flows – Create QP

- Guest driver
  - Allocates pages for send and receive rings
  - Creates page directory to hold the ring's pages
  - Initializes 'Create QP' command object (max_send_wr, send_cq_handle, recv_cq_handle, pdir etc)
  - Copies the object to 'command' address
  - Writes 0 into REQ register
- Device
  - Reads request object from 'command' address
  - Allocates QP object and initialize
    - Send and recv rings based on pdir
    - Send and recv ring state
  - Creates backend QP
  - Writes operation status to ERR register
  - Posts command-interrupt to guest
- Guest driver
  - Reads HW response code from ERR register

ORACLE redhat.

Flows – Post receive

- Guest driver
  - Initializes a wqe and place it on recv ring
  - Writes to qpn|qp_recv_bit (31) to QP offset in UAR
- Device
  - Extracts qpn from UAR
  - Walks through the ring and do the following for each wqe
    - Prepares backend CQE context to be used when receiving completion from backend (wr_id, op_code, emu_cq_num)
    - For each sge prepares backend sge; maps the dma address to qemu virtual address
    - Calls backend's post_send

Flows – Process completion

- A dedicated thread is used to process backend events
- At initialization it attach to device and create communication channel
- Thread main loop:
  - Polls completion
  - Unmaps sge's virtual addresses
  - Extracts emu_cq_num, wr_id and op_code from context
  - Writes CQE to CQ ring
  - Writes CQ number to device CQ
  - Sends completion-interrupt to guest
  - Deallocates context
  - Acks the event (ibv_ack_cq_events)

Open issues

- RDMA CM support
- Guest memory registration
- Migration support
- Memory overcommit support
- Providing GIDs to guests

Open issues (cont)

- RDMA CM support
  - Emulate QP1 at QEMU level
  - Modify RDMA CM to communicate with QEMU instances:
    - Pass incoming MADs to QEMU
    - Allow QEMU to forward MADs
  - What would be the preferred mechanism?

Open issues (cont)

- Guest memory registration
  - Guest userspace:
    - Register memory region verb accept only a single VA address specifying both the on-wire Key and a memory range.
  - Guest kernels:
    - In-kernel code may need to register all memory (ranges not used are not accessed → kernel is trusted)
    - Memory hotplug changes present ranges

Open issues (cont)

- Migration support
  - Moving QP numbers – ensure same QP numbers on destination
  - Memory tracking
    - Migration requires detection of hw accesses to pages
    - Maybe we can avoid the need for it
  - Moving QP state
    - How QP should behave during migration , some “busy state”
  - Network update/packet loss recovery
    - Can the system recover from some packet loss?

Open issues (cont)

- Memory overcommit support
  - Do we really need “on-demand” pages?
    - Everything goes through QEMU which is a User Level app in host that fw the request to host kernel.
    - Did we miss anything?

Open issues (cont)

- Providing GIDs to guests (multi-gid)
  - Instead of coming up with extra ipv6 addresses, would be better to set the host GID based on guest ipv6 address.
  - No API for doing that from user level.

Future plans

- Submit VMware's PVRDMA device implementation to QEMU (and later add RoCE v2 support)
- Improve performance
- Move to a virtio based RDMA device
  - We would like to agree on a clean design:
    - Virtio queue for the command channel
    - Virtio queue for each Rcv/Send/Cq buffer
    - Anything else?