# HW High-Availability and Link Aggregation for Ethernet switch and NIC RDMA using Linux bonding/team

Tzahi Oved tzahio@mellanox.com ; Or Gerlitz ogerlitz@mellanox.com

Netdev 1.1 | 2016

Mellanox TECHNOLOGIES
Connect. Accelerate. Outperform.™

## Bonding / Team drivers

- both expose software netdevice  that provides LAG / HA toward the networking stack
- team/bond is considered “upper” device to “lower” NIC net-devices through which packets are flowing to the wire
- different modes of operation: Active/Passive, 802.3ad  (LAG) and policies: link monitoring, xmit hash, etc
- Bonding – legacy
- Team - introduced in 3.3, more modular/flexible design, extendable, state-machine in user-space library/daemon

## HW LAG using SW Team/Bond

- Idea: use SW LAG on netdevices to apply LAG into HW offloaded traffic
- offloaded traffic – doesn't pass through the network stack

**100Gbs Switch**

- each port is represented by netdevice
- SW LAG on few ports netdevs → set HW LAG on physical ports (mlxsw, upstream 4.5)

**40/100Gbs NIC**

- each port of the device is Eth netdevice
- RDMA traffic is offloaded from the network stack
- port netdevice serves for plain Eth networking and control pass for the RDMA stack
- SW LAG on two NIC ports netdevs → HW LAG for RDMA traffic (mlx4, upstream 4.0)
- under SRIOV, SW LAG on PF NIC ports → HW LAG for vport used by VF (mlx4, upstream 4.5)
- for 100Gbs NIC (mlx5) – coming soon…

## Network notifiers && their usage for HW LAG

- notification sent to subscribed consumers in the networking stack on a change which is about to take place, or that just happened
- the notification contains events type and affected parties
- Notifications used for LAG: pre change-upper, change-upper

HW driver usage for LAG notifications:

- pre-change upper: refuse certain configurations, NAK the change
- change upper: create / configure HW LAG

## Switch HW driver

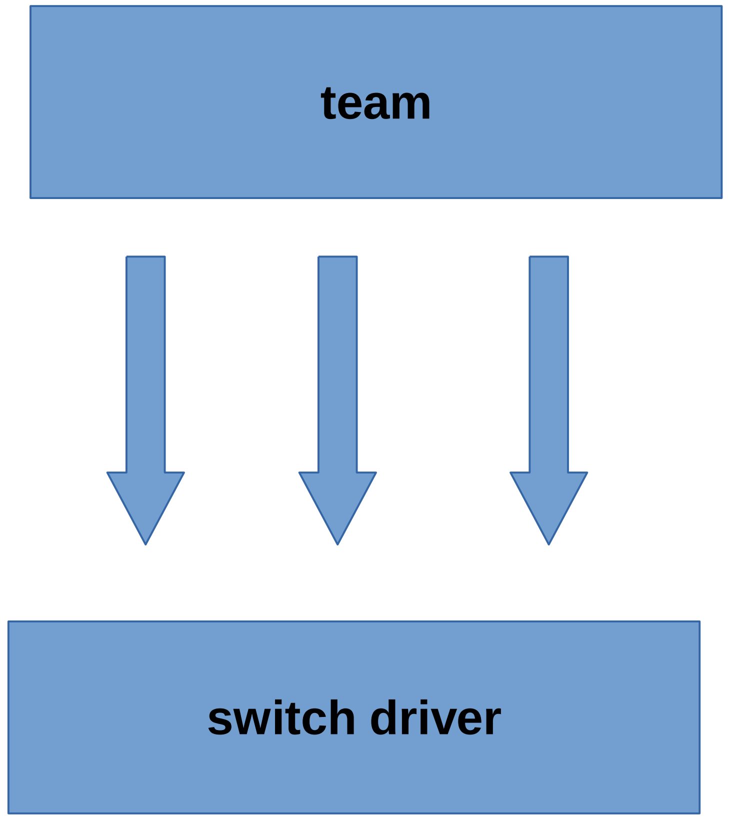
- *ip link set dev sw1p1 master team0*
  - NETDEV_PRECHANGEUPPER
    - if lag type is not LACP, etc - NAK

      → operation fails
  - NETDEV_CHANGEUPPER
    - observe that new lag is created for the switch

      → create HW LAG and add this port there

- *ip link set dev sw1p2 master team0*
  - NETDEV_PRECHANGEUPPER
    - […]
  - NETDEV_CHANGEUPPER
    - observe that this lag already exists

      → add this port there

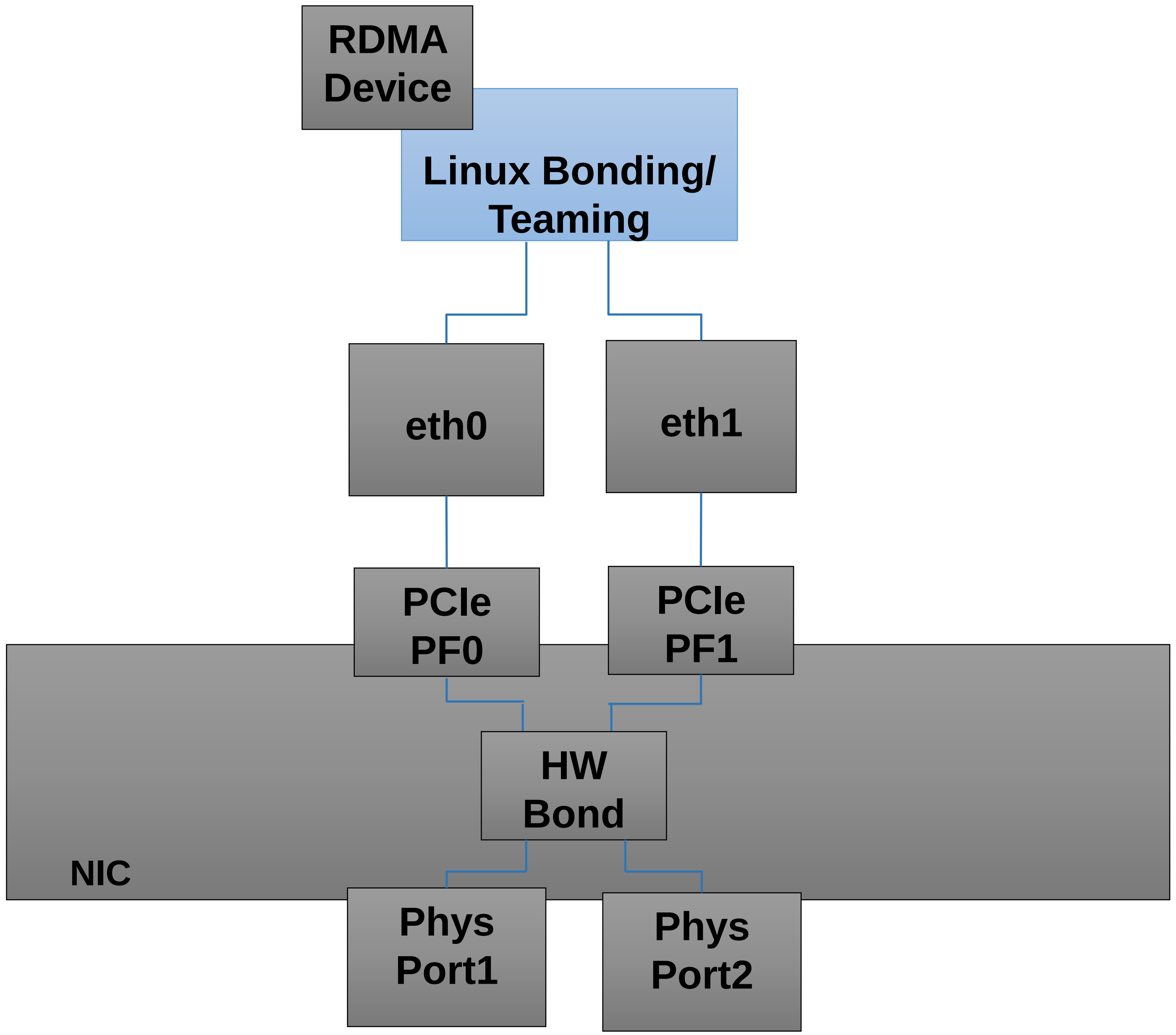


## RDMA over Ethernet (RoCE) / RDMA-CM

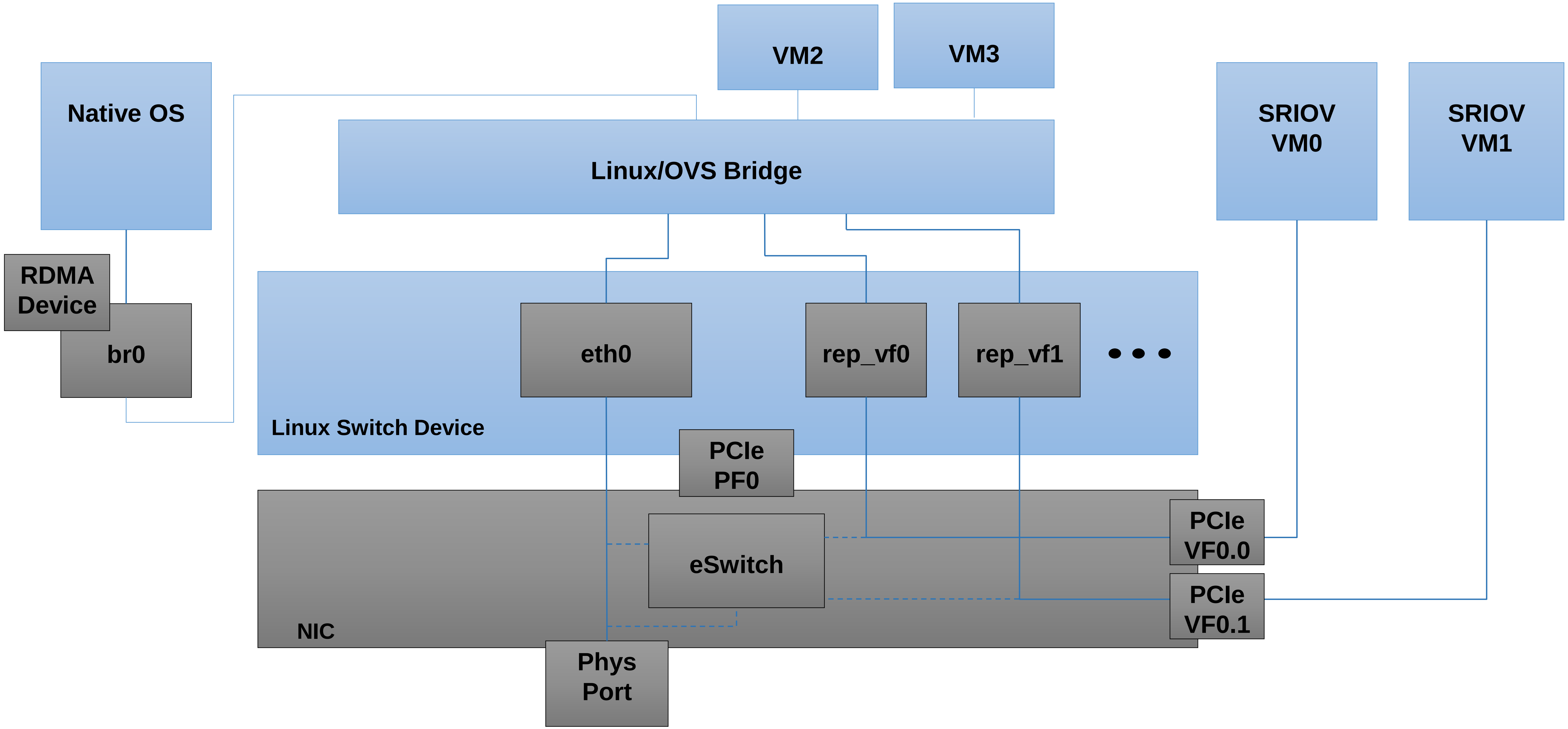
- The upstream RDMA stack supports multiple transports: **RoCE**, IB, iWARP
- RoCE – **R**DMA **o**ver **C**onverged **E**thernet, RoCE V2 (upstream 4.5), IBTA RDMA headers over UDP. Uses IPv4/6 addresses set over the regular Eth NIC port netdev
- RoCE apps use **RDMA-CM** API for control path and **verbs** API for data path
- RDMA-CM API *(include/rdma/rdma_cm.h*)
  - Address resolution – Local Route lookup + ARP/ND services (rdma_resolve_addr())
  - Route resolution – Path lookup in IB networks (rdma_resolve_route())
  - Connection establishment – per transport CM to wire the offloaded connection (rdma_connect())
- Verbs API
  - Send/RDMA – Send message or perform RDMA operation (post_send())
  - Poll– Poll for completion of Send/RDMA or Receive operation (poll_cq())
    - Async completion handling and fd semantics are supported
  - Post Receive Buffer – Hand receive buffers to the NIC (post_recv())
- RDMA Device
  - The DEVICE structure, exposes all above operations
  - Associated with net_device
- Available for both RoCE and user mode Ethernet programming (DPDK)

## Native Model – HW Teaming

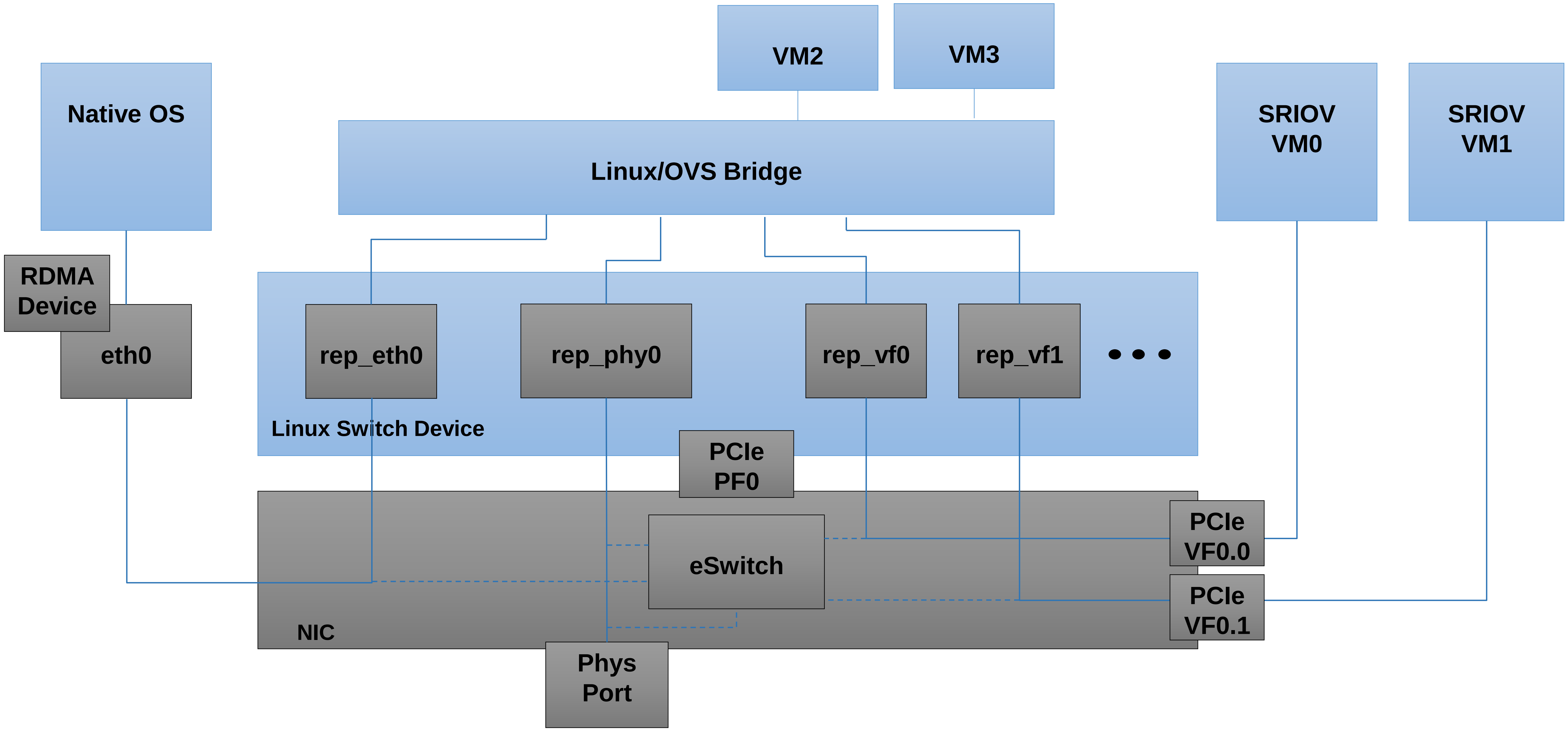
- Configuration
  - Native Linux administration
  - RoCE Bonding is mainly auto configured
- RoCE
  - Use transport object (QP, TIS) attribute: port affinity
  - RDMA devices associated with eth0, eth1 will be used for port management only (through Immutable caps)
    - And will unregister and register to drop existing consumers
  - Register new ib_dev attached to the bond
    - eth0, eth1 will listen on Linux bond enslavement netlink events
    - New RDMA device will always use vendor pick of PCIe Function (PF0/1 or both)
- LACP ((802.3ad)
  - Either handled by Linux bonding/teaming driver
  - Or in HW/FW for supporting NICs (required for many PFs to single phys port configurations)
- HW Bond
  - NIC logic for HW forwarding of ingress traffic to bond/team RDMA device
  - net_dev traffic is passed directly to owner net_dev according to ingress port



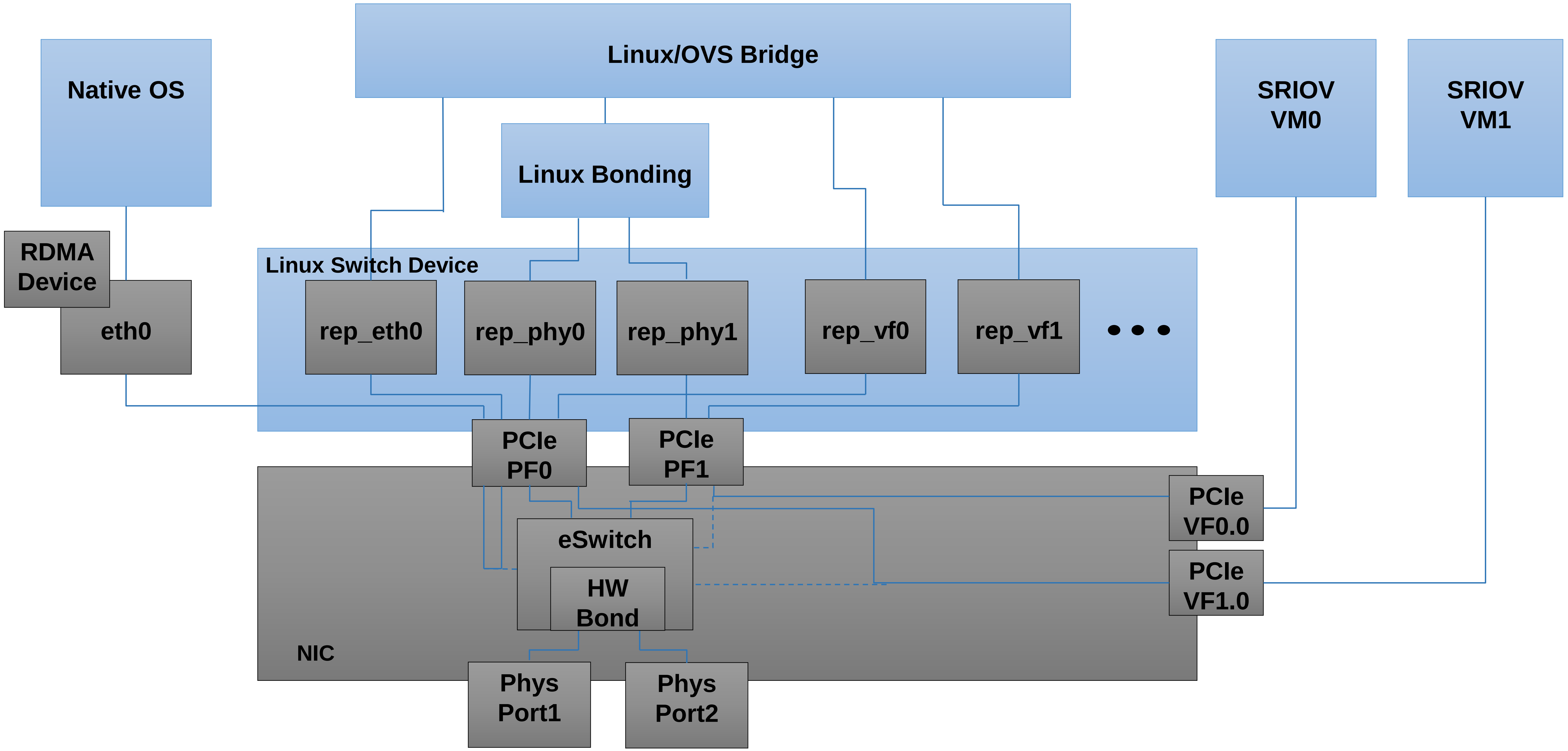
## eSwitch Software Model – Option I



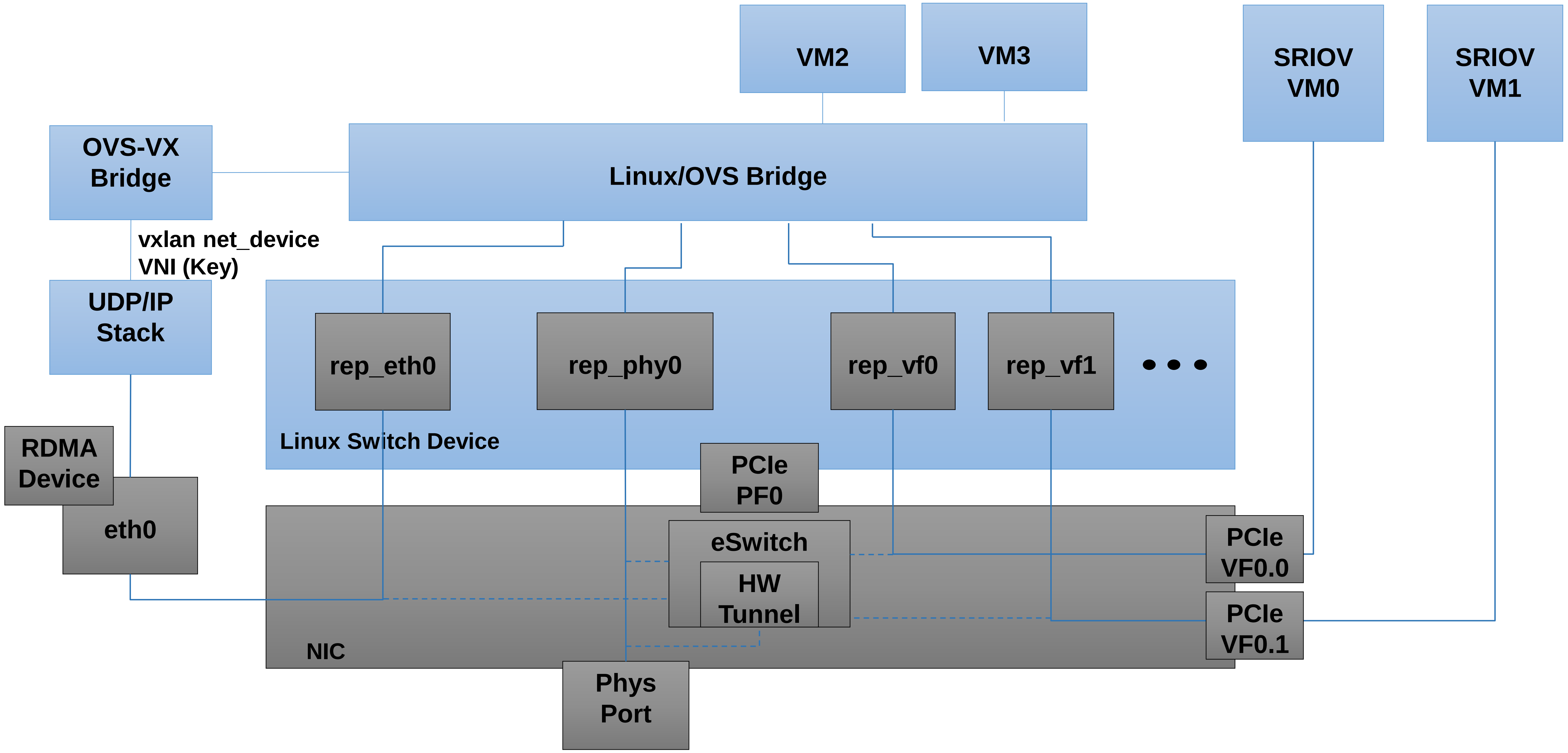
# eSwitch Software Model – Option II



## eSwitch Software Model with HA

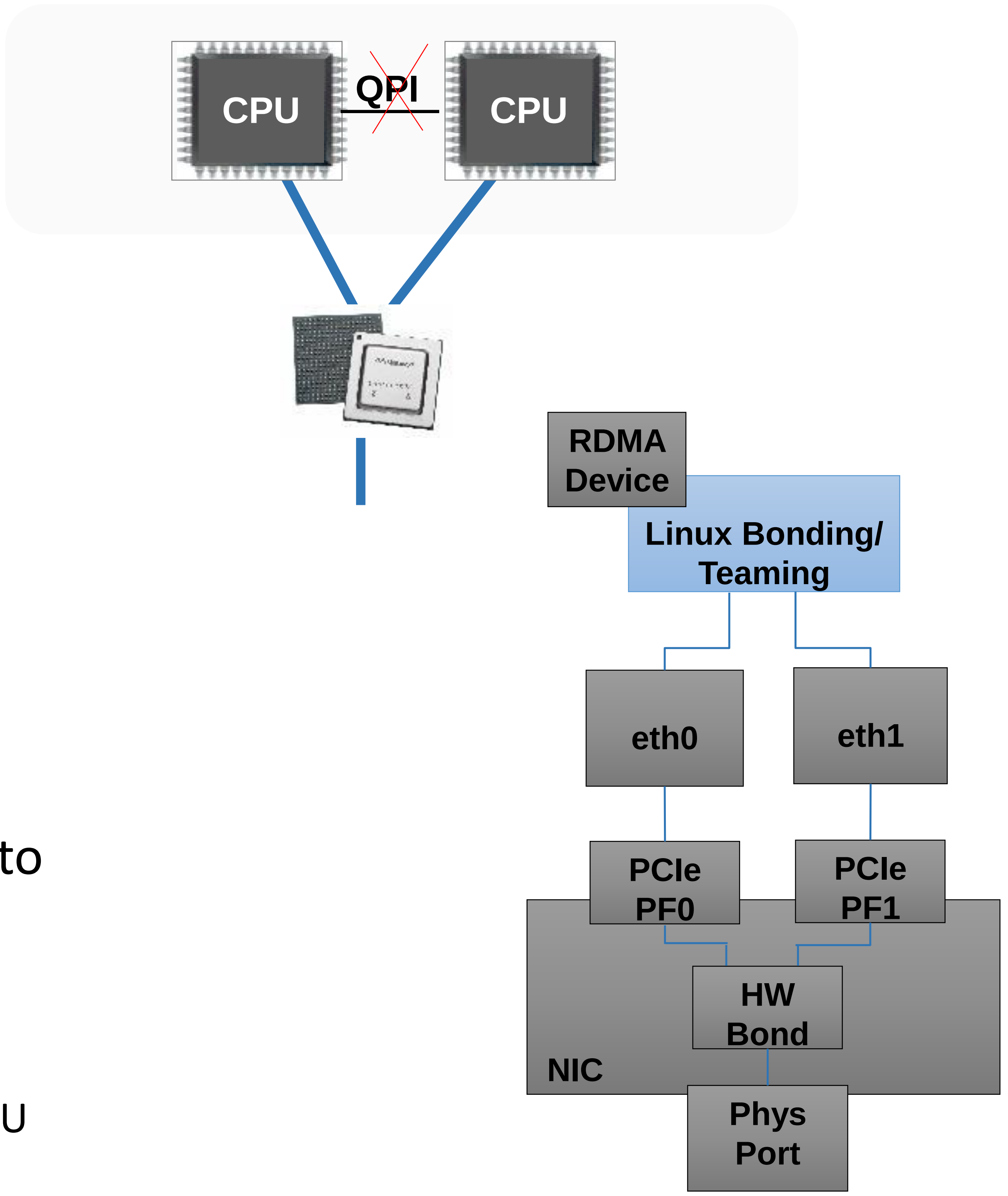


## eSwitch Software Model with Tunneling



## Multi-PCI Socket NIC

- Multiple PCIe end point NIC - NIC can be connected through one or more PCIe buses
- Each PCIe bus is connected different NUMA node
- Exposed as 2 or more net_device each with it's own associated RDMA device
- Enjoy direct device to local NUMA access
- Application use & feel – would like to work with single net interface
- Use Linux bonding with RDMA device bonding
  - For TCP/IP traffic on TX, select slave according to calling context affinity
  - For RDMA traffic select slave according to:
    - Transport object (QP) logical port affinity
    - Or transport object creation thread CPU affinity
    - Don't share HW resources (CQ, SRQ) on different CPU sockets – each device has it's own HW resources


CPU
QPI
CPU
RDMA Device
Linux Bonding/ Teaming
eth0
eth1
PCIe PF0
PCIe PF1
HW Bond
NIC
Phys Port