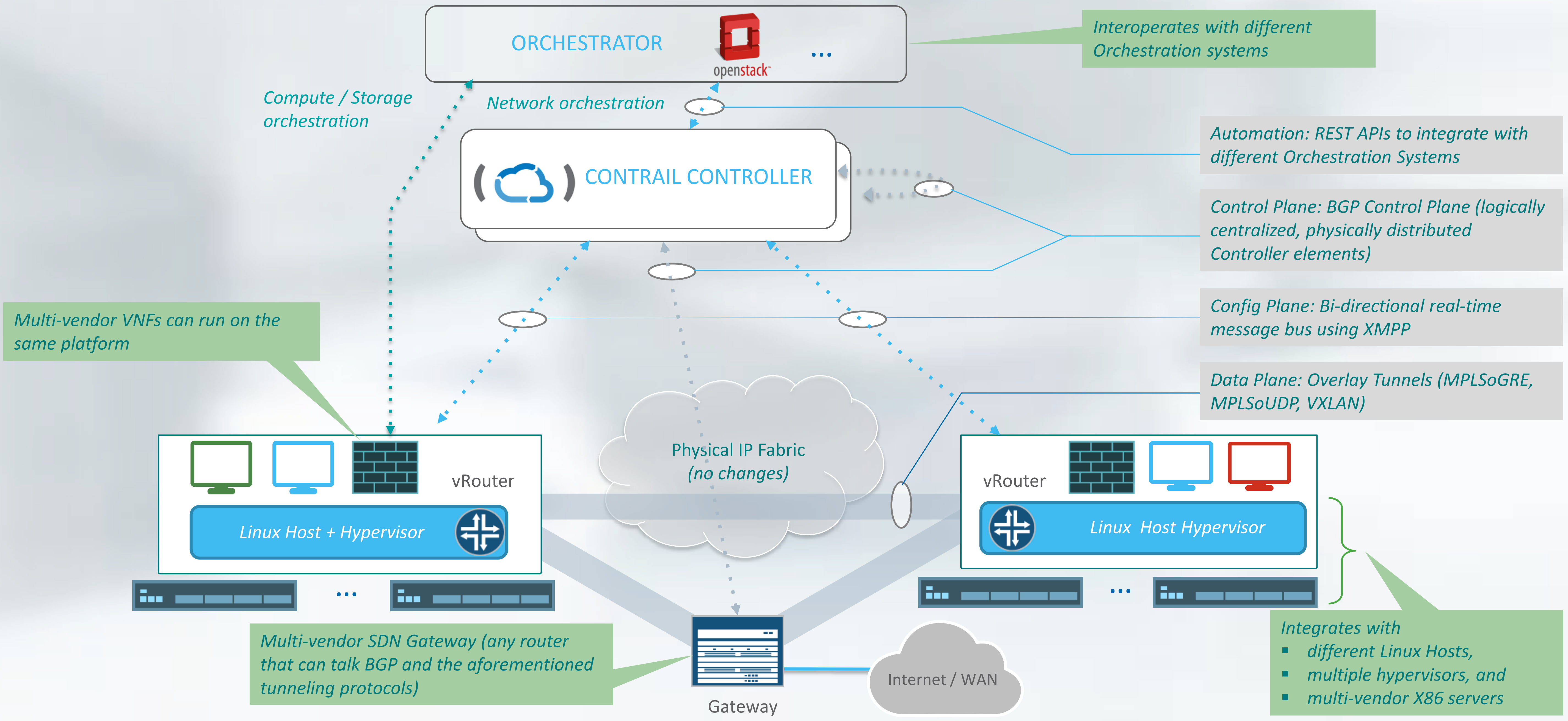
DPDK Summit 2016

OpenContrail vRouter / DPDK Architecture

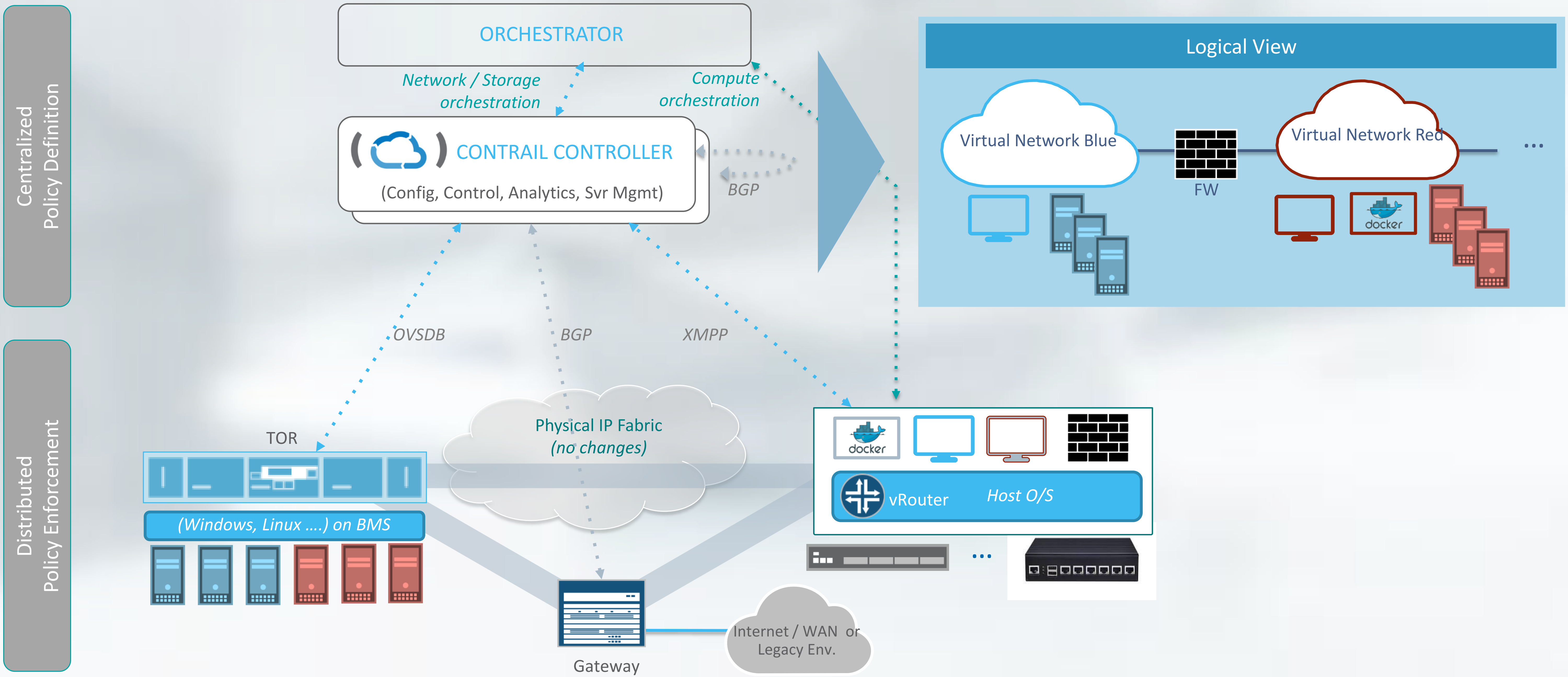
Raja Sivaramakrishnan, Distinguished Engineer

Aniket Daptari, Sr. Product Manager

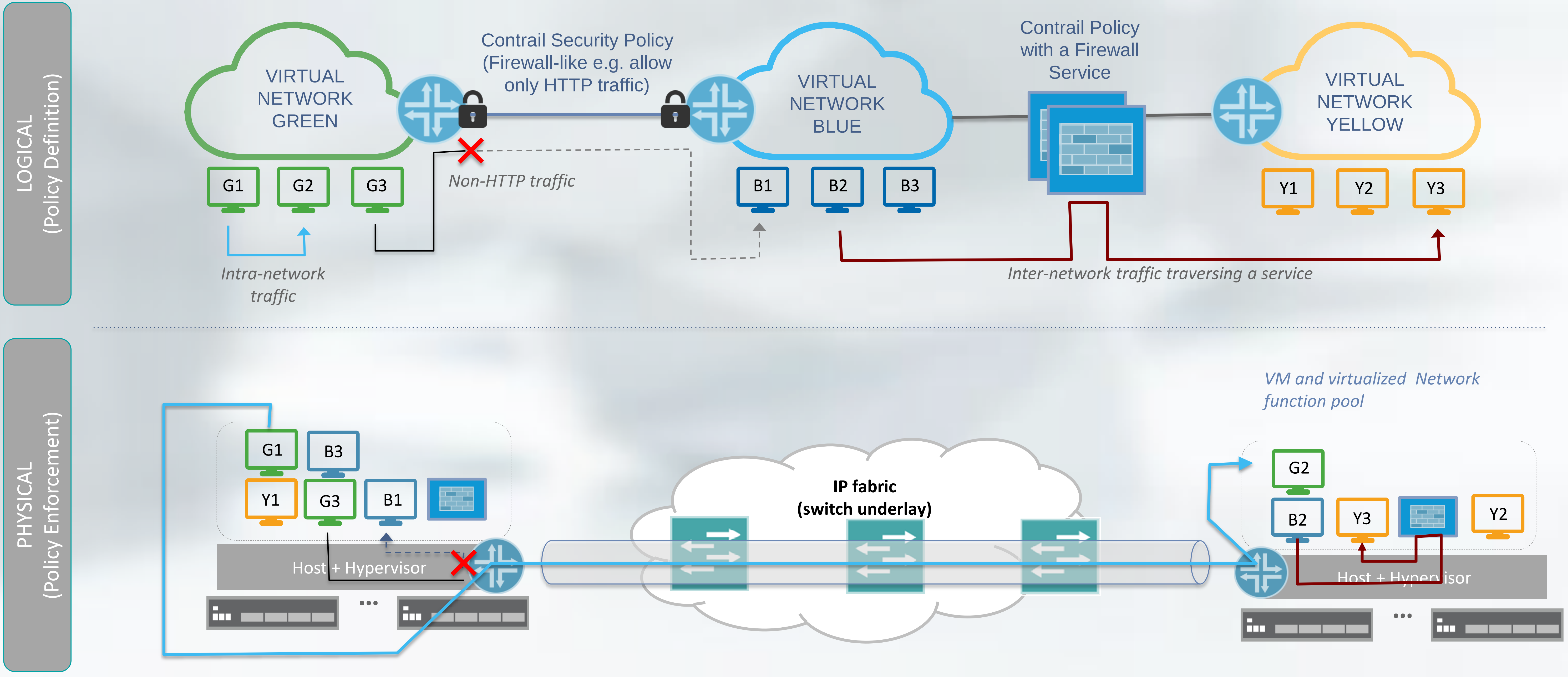
CONTRAIL (MULTI-VENDOR) ARCHITECTURE



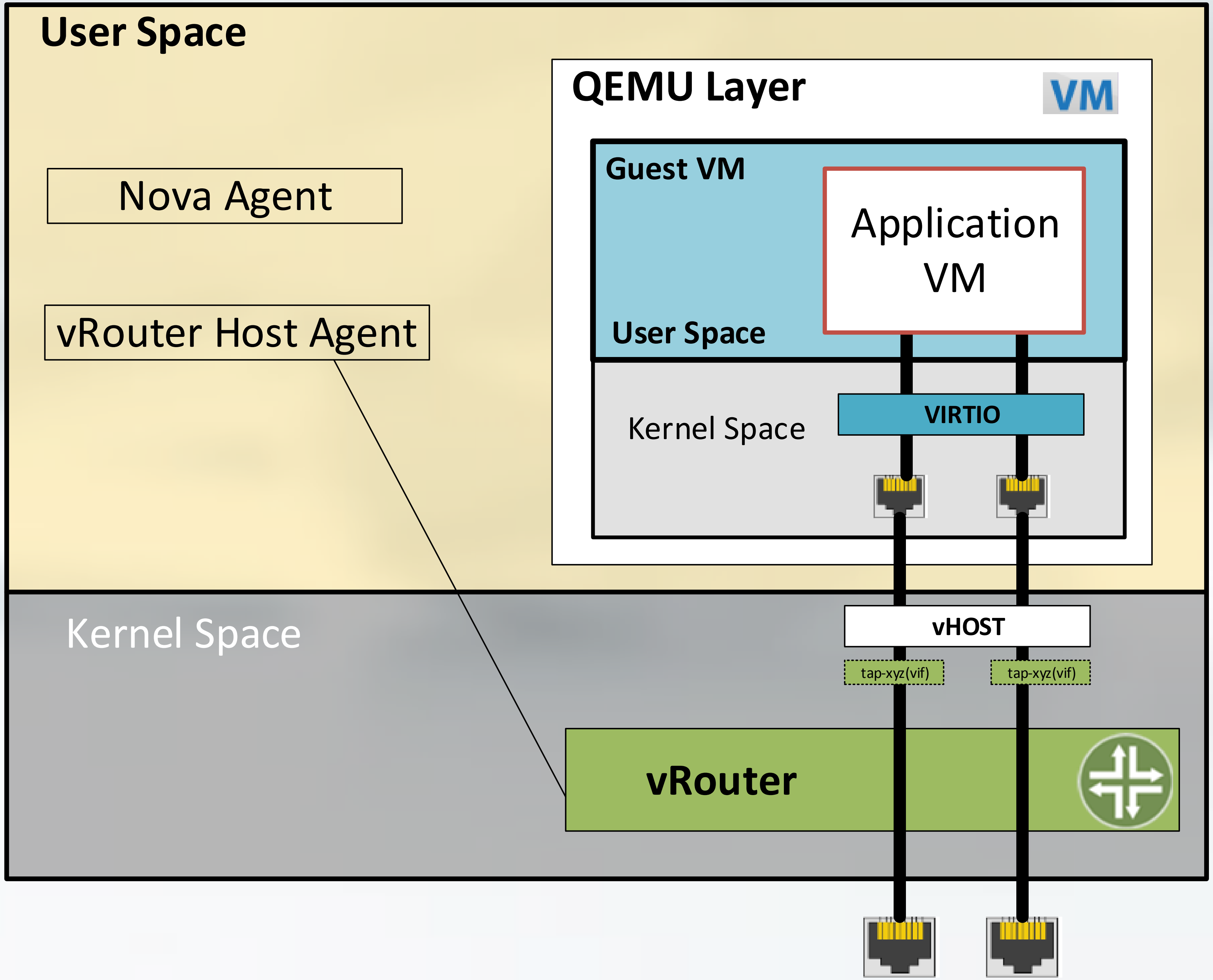
CONTRAIL ARCHITECTURE



VIRTUAL NETWORKS: LOGICAL VERSUS PHYSICAL



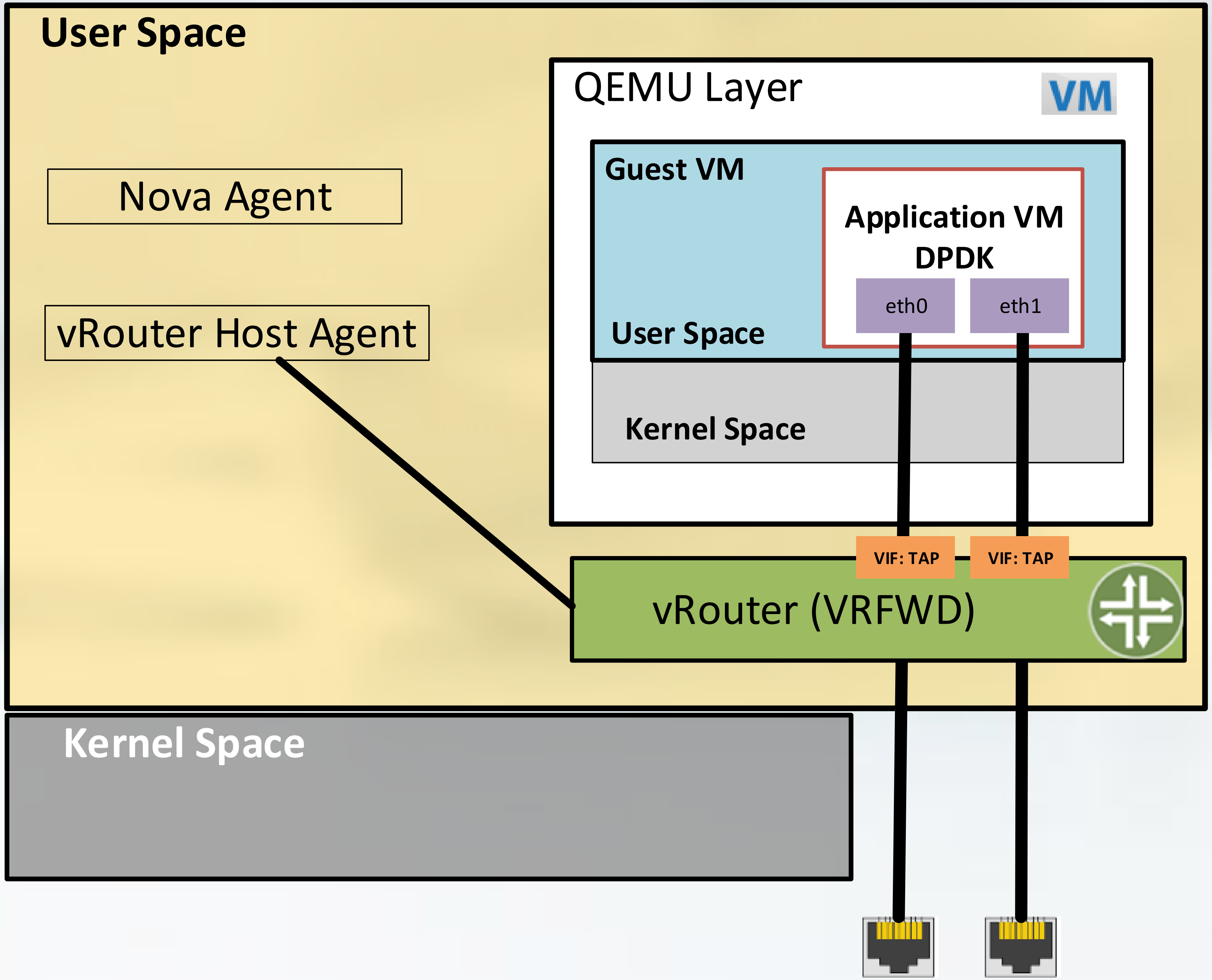
vRouter Overview (Today)



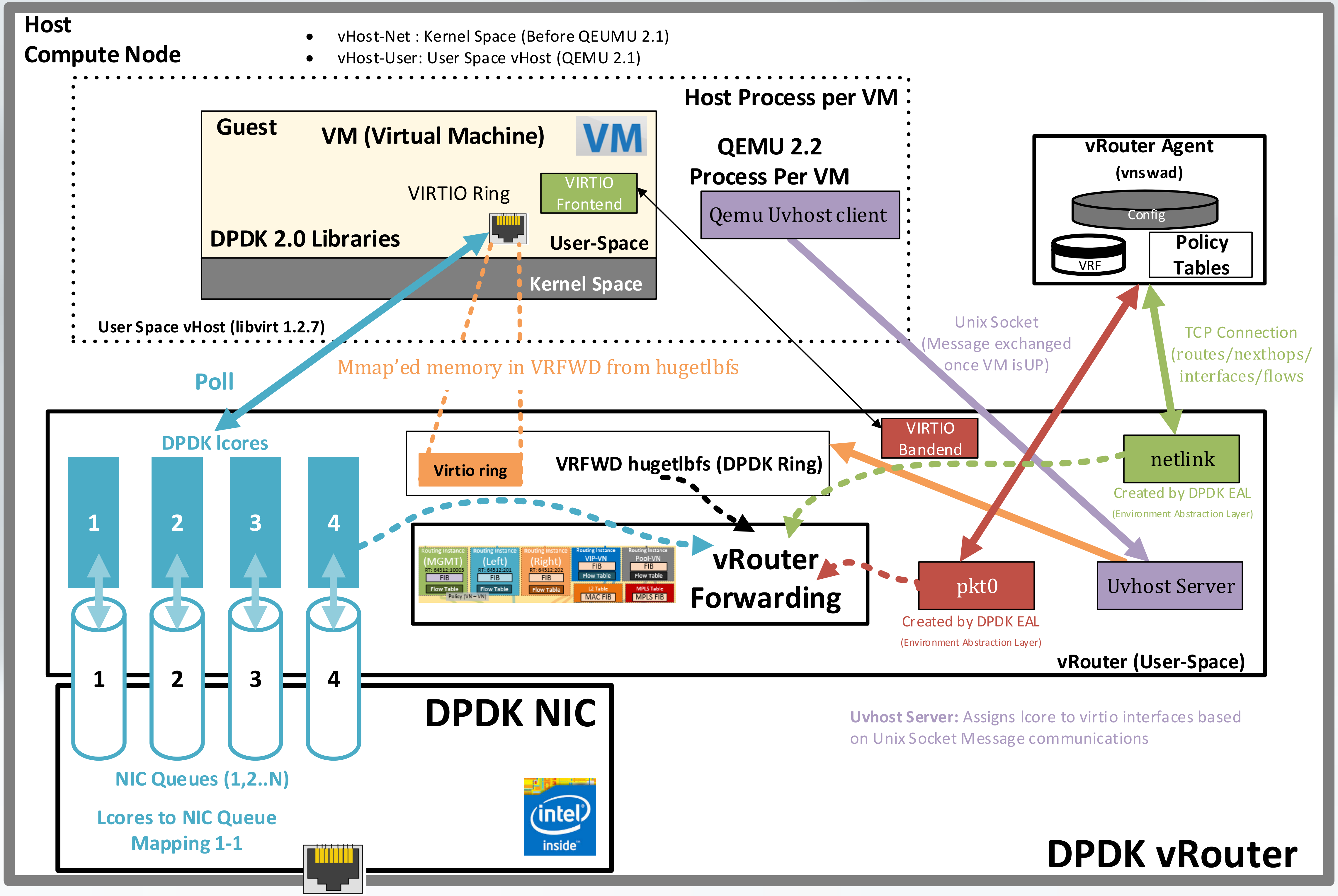
DPDK Overview

- DPDK based forwarding is implemented completely in user space
- The application runs as multiple logical cores
  - Lcores are pthreads with core affinity
  - Lcores run in poll mode and handle bursts of packets for maximum performance
- DPDK provides lockless rings for communicating between Lcores
- Highly optimized for Intel architecture
  - Efficient use of CPU caches
  - Huge pages to reduce TLB misses
  - NUMA aware

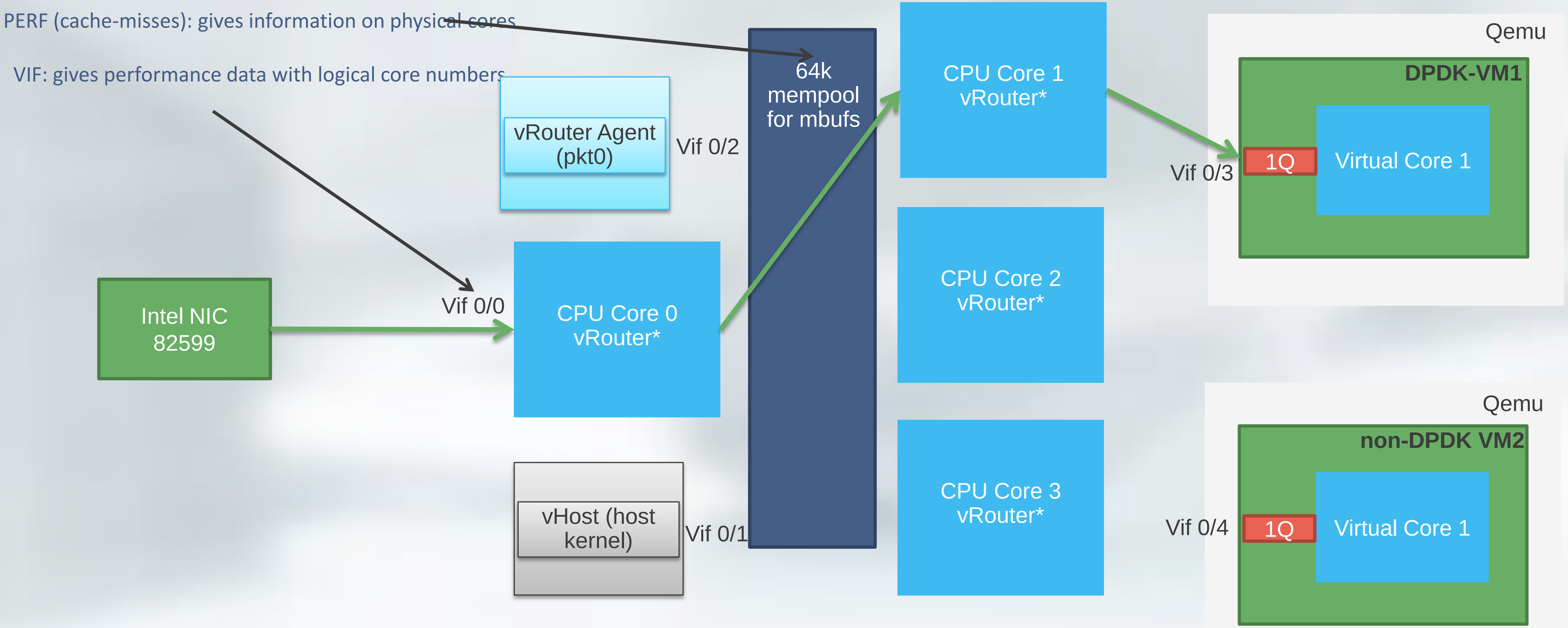
DPDK vRouter Overview



DPDK vRouter Architecture



DPDK 2.1 Contrail vRouter packet walk-through (from NIC to Guest) for a DPDK VM



STEPS:

1. For all traffic from SDN Gateway, packets will go from Intel NIC to Core0 for inner IP lookup and hashing. This is because the NIC can't hash on inner header it only sees GRE header
2. Core 0 hashes based on 5-tuple to Core1 or 2 or 3
3. Say Core 0 sends to Core 1. Core 1 provides lookup, flow table creation, re-write and policy/SG enforcement then sends to DPDK-VM1

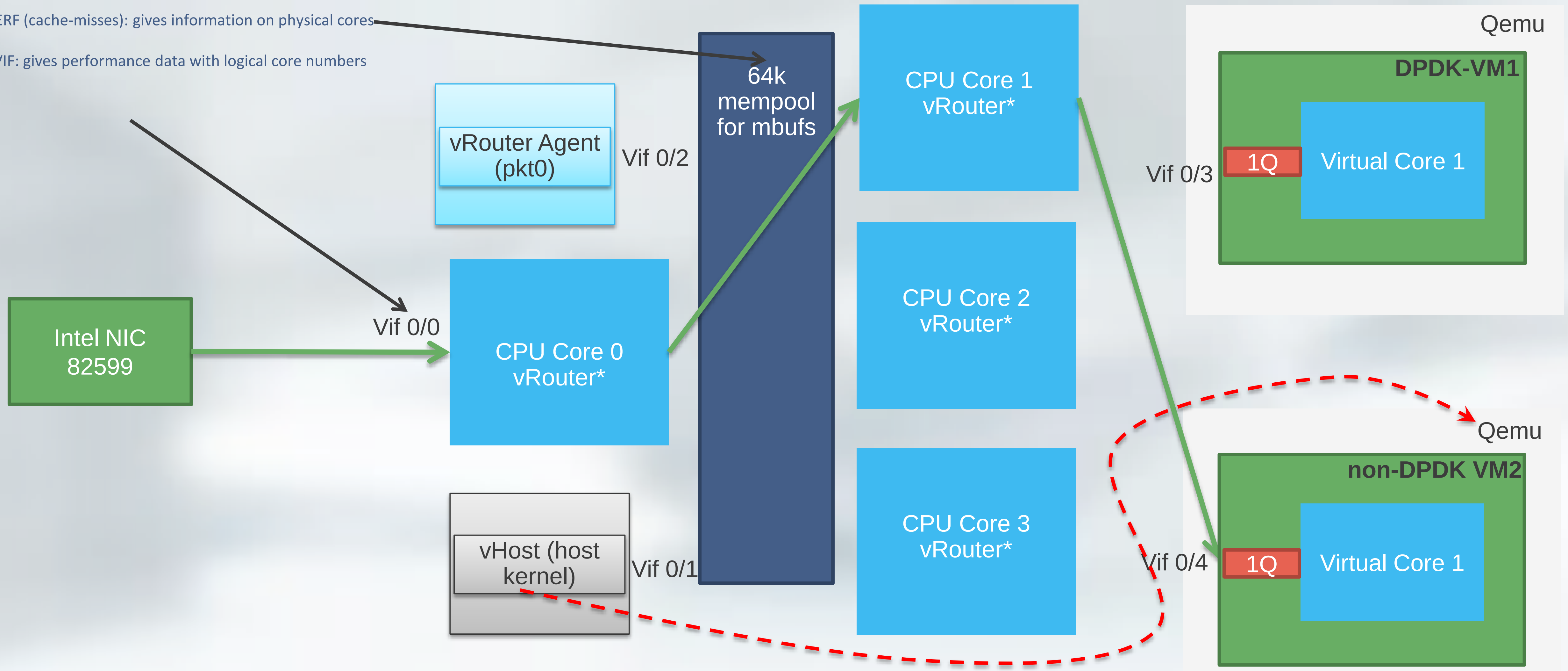
*** vRouter runs as a multi-core process that exists on all 4 Cores. Also we have a scale-out approach to packet processing using multiple cores so the performance of 1VM is NOT limited by the performance of 1 core**

Note: the VM has a single queue that 3 Lcores could be sending to it.

DPDK 2.1 Contrail vRouter packet walk-through (from NIC to Guest) for a non-DPDK VM

PERF (cache-misses): gives information on physical cores

VIF: gives performance data with logical core numbers

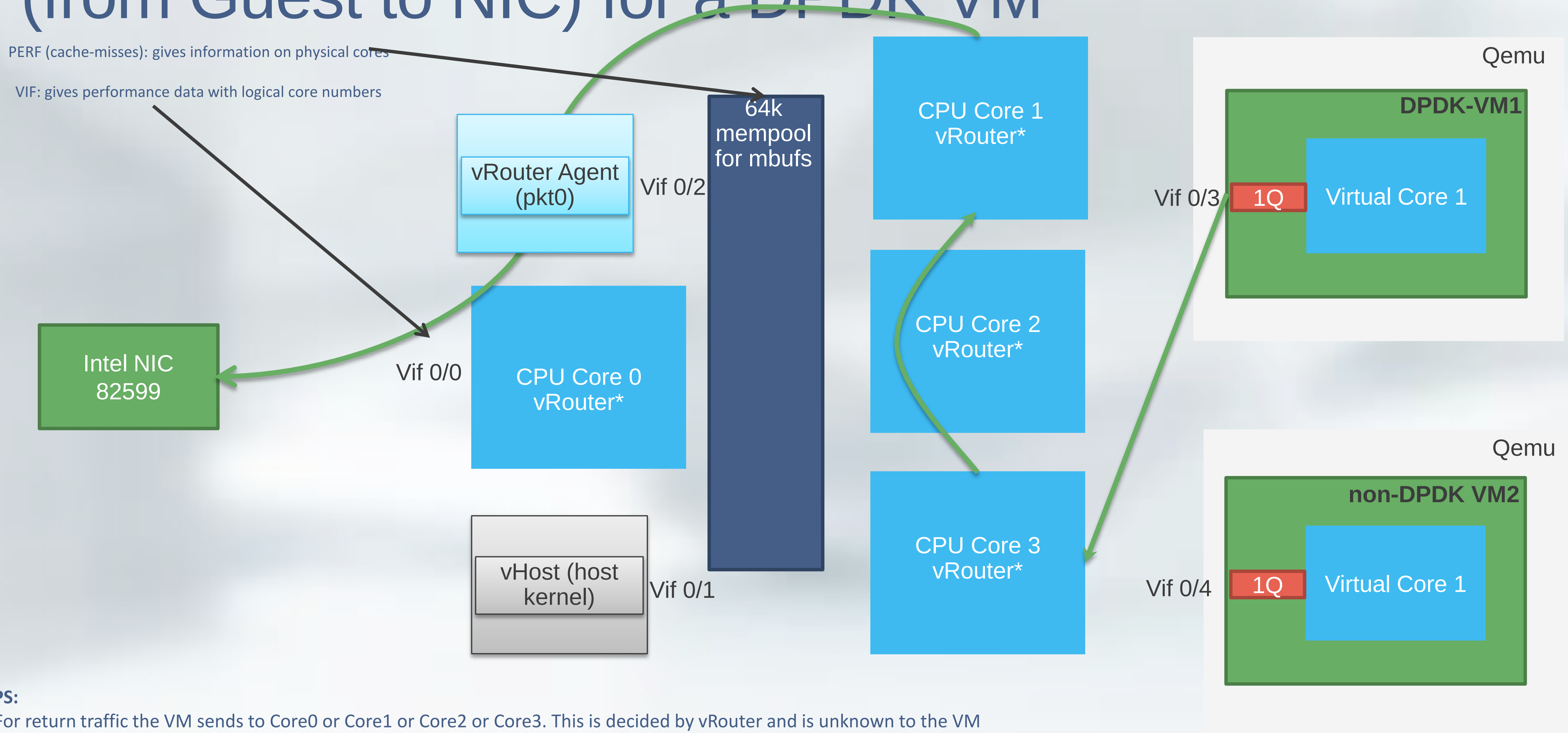


STEPS:

1. For all traffic from SDN Gateway, packets will go from Intel NIC to Core0 for inner IP lookup and hashing. This is because the NIC can't hash on inner header it only sees GRE header
2. Core 0 hashes based on 5-tuple to Core1 or 2 or 3
3. Say Core 0 sends to Core 1. Core 1 provides lookup, flow table creation, re-write and policy/SG enforcement then sends to non-DPDK VM2
4. **NOTE: When sending to the VM, vHost raises an interrupt in the guest. This is an additional step after copying the packet to the VM so the interrupt is only needed because the VM is not polling for packets. The vHost writes to a file descriptor, which tells the Kernel to raise an interrupt to non-DPDK VM2. The initial file descriptor is sent by Qemu to vHost when the VM is spawned. Also note that an Interrupt is raised for a burst of packets, not for every packet.**

*** vRouter runs as a multi-core process that exists on all 4 Cores. Also we have a scale-out approach to packet processing using multiple cores so the performance of 1VM is NOT limited by the performance of 1 core**

DPDK 2.1 Contrail vRouter packet walk-through (from Guest to NIC) for a DPDK VM

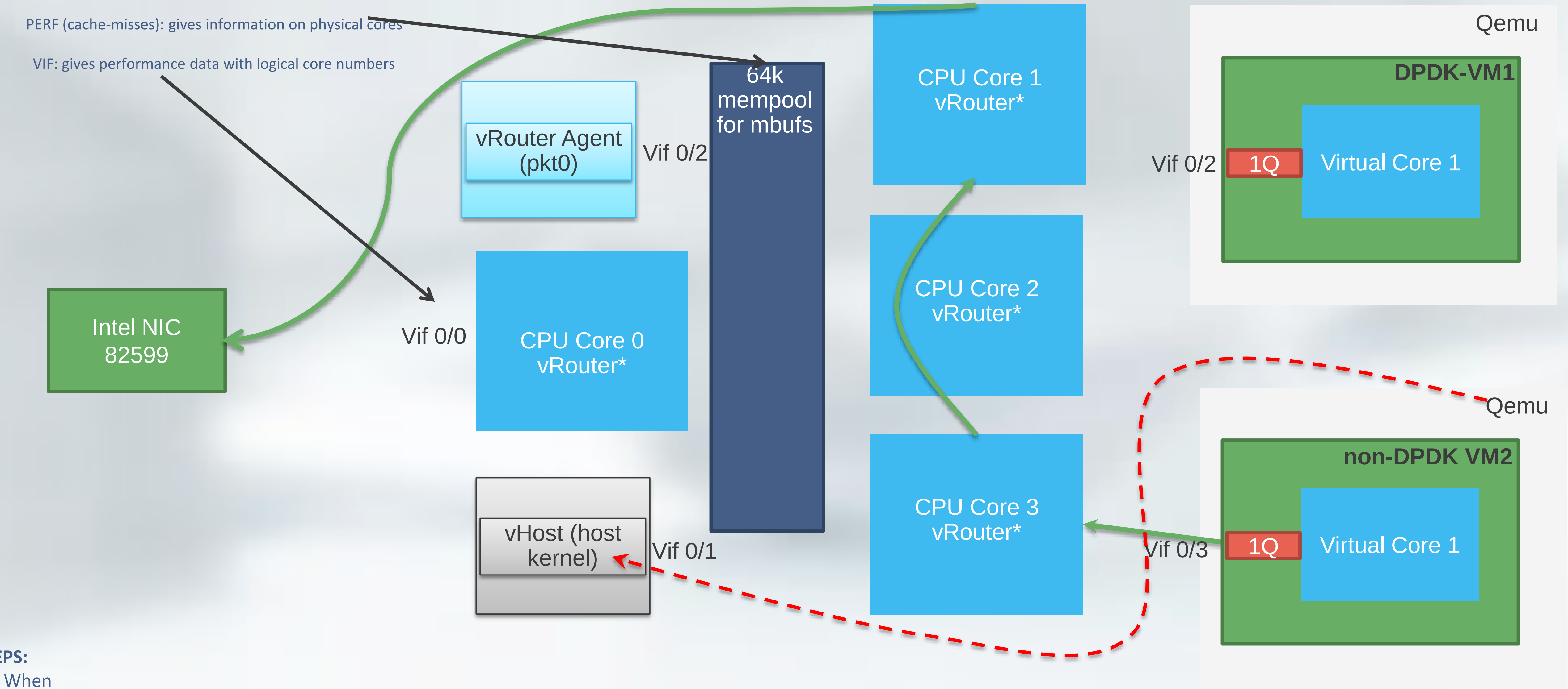


STEPS:

1. For return traffic the VM sends to Core0 or Core1 or Core2 or Core3. This is decided by vRouter and is unknown to the VM
2. Say VM1 sends to Core 3. Core3 hashes based on the 5-tuple and sends to Core0 or Core1 or Core 2.
3. Say Core3 sends to Core1. Core1 does all the packet handling provides lookup, flow table creation, re-write and policy/SG enforcement then sends to Intel NIC

*** vRouter runs as a multi-core process that exists on all 4 Cores. Also we have a scale-out approach to packet processing using multiple cores so the performance of 1VM is NOT limited by the performance of 1 core**
Note: the VM has a single queue that 3 Cores could be sending to it.

DPDK 2.1 Contrail vRouter packet walk-through (from Guest to NIC) for a non-DPDK VM

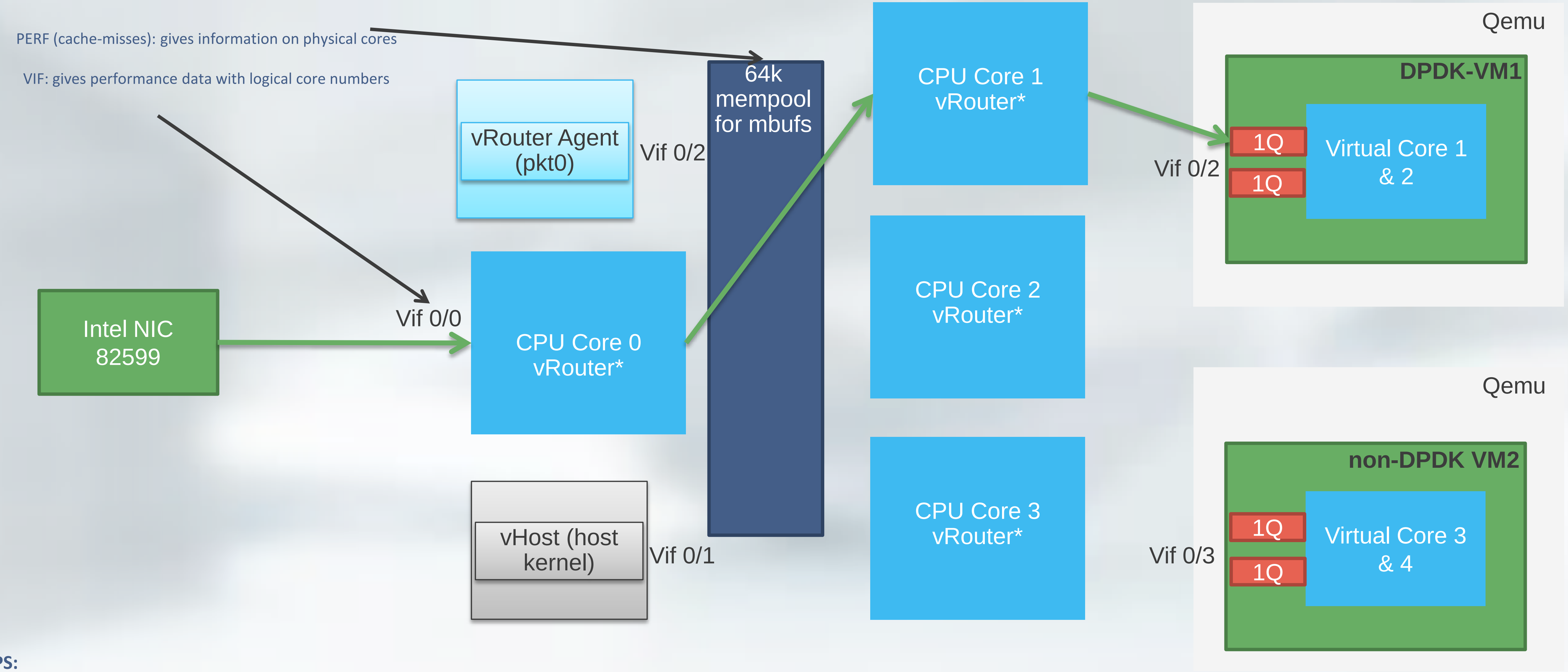


STEPS:

1. When
2. For return traffic the VM sends to Core0 or Core1 or Core2 or Core3. This is decided by vRouter and is unknown to the VM
3. Say VM1 sends to Core 3. Core3 hashes based on the 5-tuple and sends to Core0 or Core1 or Core 2.
4. Say Core3 sends to Core1. Core1 does all the packet handling provides lookup, flow table creation, re-write and policy/SG enforcement then sends to Intel NIC
5. **NOTE: When sending to the vRouter, Qemu raises an interrupt vHost. This is an additional step after copying the packet to user-space so the interrupt is only needed because the vHost is not polling for packets. The Qemu writes to a file descriptor, which tells the Kernel to raise an interrupt to vHost. vHost tells CPU Core 3 using a descriptor where to fetch the packets for processing. Also note that an Interrupt is raised for a burst of packets so not for every packet.**

***vRouter runs as a multi-core process that exists on all 4 Cores. Also we have a scale-out approach to packet processing using multiple cores so the performance of 1VM is NOT limited by the performance of 1 core**

DPDK 2.2 Contrail vRouter packet walk-through with MQ-Virtio (from NIC to Guest) for a MQ-Virtio DPDK VM

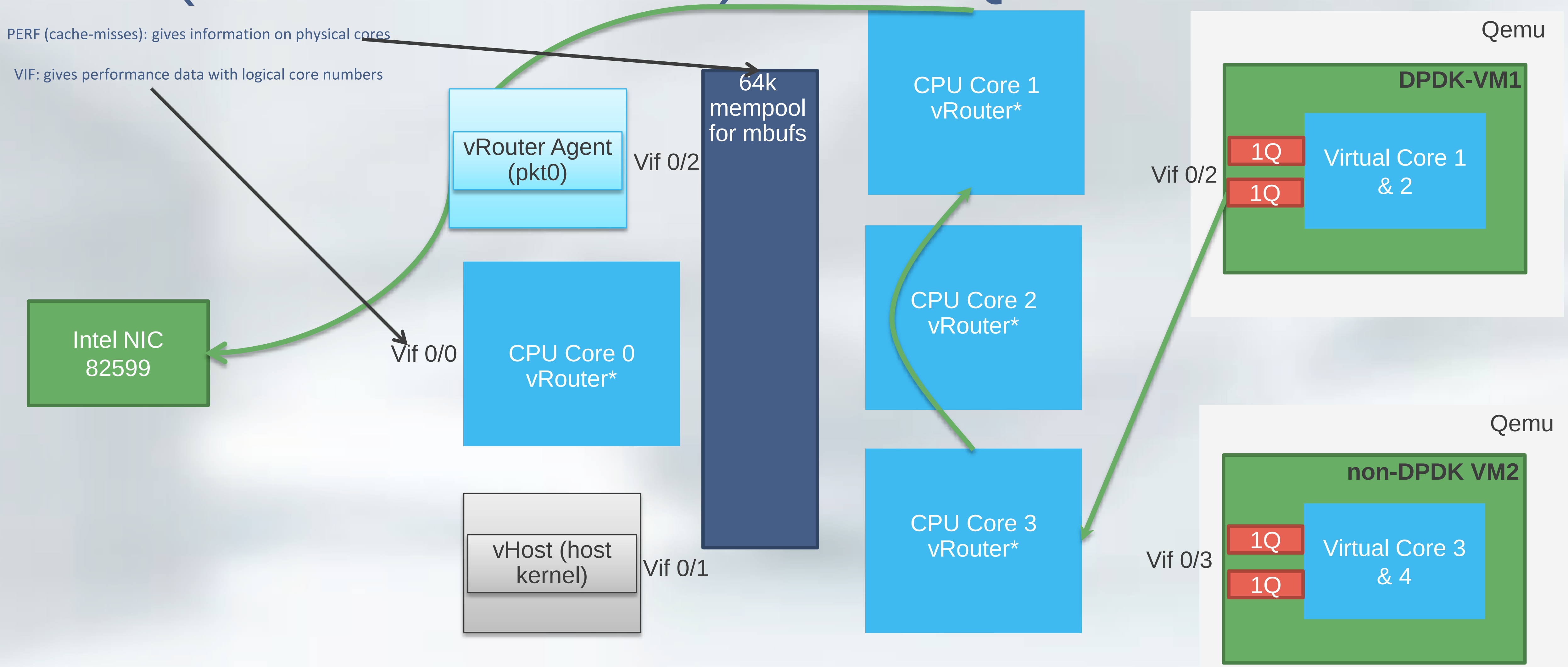


STEPS:

1. For all traffic from SDN Gateway, packets will go from Intel NIC to Core0 for inner IP lookup and hashing. This is because the NIC can't hash on inner header it only sees GRE header
2. Core 0 hashes based on 5-tuple to Core1 or Core2 or Core3
3. Say Core 0 sends to Core 1. Core 1 provides lookup, flow table creation, re-write and policy/SG enforcement then sends to VM1

*** vRouter runs as a multi-core process that exists on all 4 Cores. Also we have a scale-out approach to packet processing using multiple cores so the performance of 1VM is NOT limited by the performance of 1 core**
Note: vRouter might have more cores than there are queues in the VM. Queues can only be shared when vRouter cores send to the VM queue. When receiving from the queue, exactly one vRouter core will read from a queue (I.e. Queues are not shared to prevent packet reordering).

DPDK 2.2 Contrail vRouter packet walk-through with MQ-Virtio (from Guest to NIC) for a MQ-Virtio DPDK VM



STEPS:

1. For return traffic the VM sends to Core0, Core1, Core2 or Core3. This is decided by vRouter and is unknown to the VM
2. Say VM1 sends to Core 3. Core3 hashes based on the 5-tuple and sends to Core0 or Core1 or Core 2.
3. Say Core3 sends to Core1. Core1 does all the packet handling provides lookup, flow table creation, re-write and policy/SG enforcement then sends to Intel NIC

*** vRouter runs as a multi-core process that exists on all 4 Cores. Also we have a scale-out approach to packet processing using multiple cores so the performance of 1VM is NOT limited by the performance of 1 core**
Note: vRouter might have more cores than there are queues in the VM. Queues can only be shared when vRouter cores send to the VM queue. When receiving from the queue, exactly one vRouter core will read from a queue (I.e. Queues are not shared to prevent packet reordering).

Thank You!!