SR-IOV* HANDS-ON LAB

Rahul Shah

Clayne Robison

*Single Root IO Virtualization

THE NEW CENTER OF POSSIBILITY

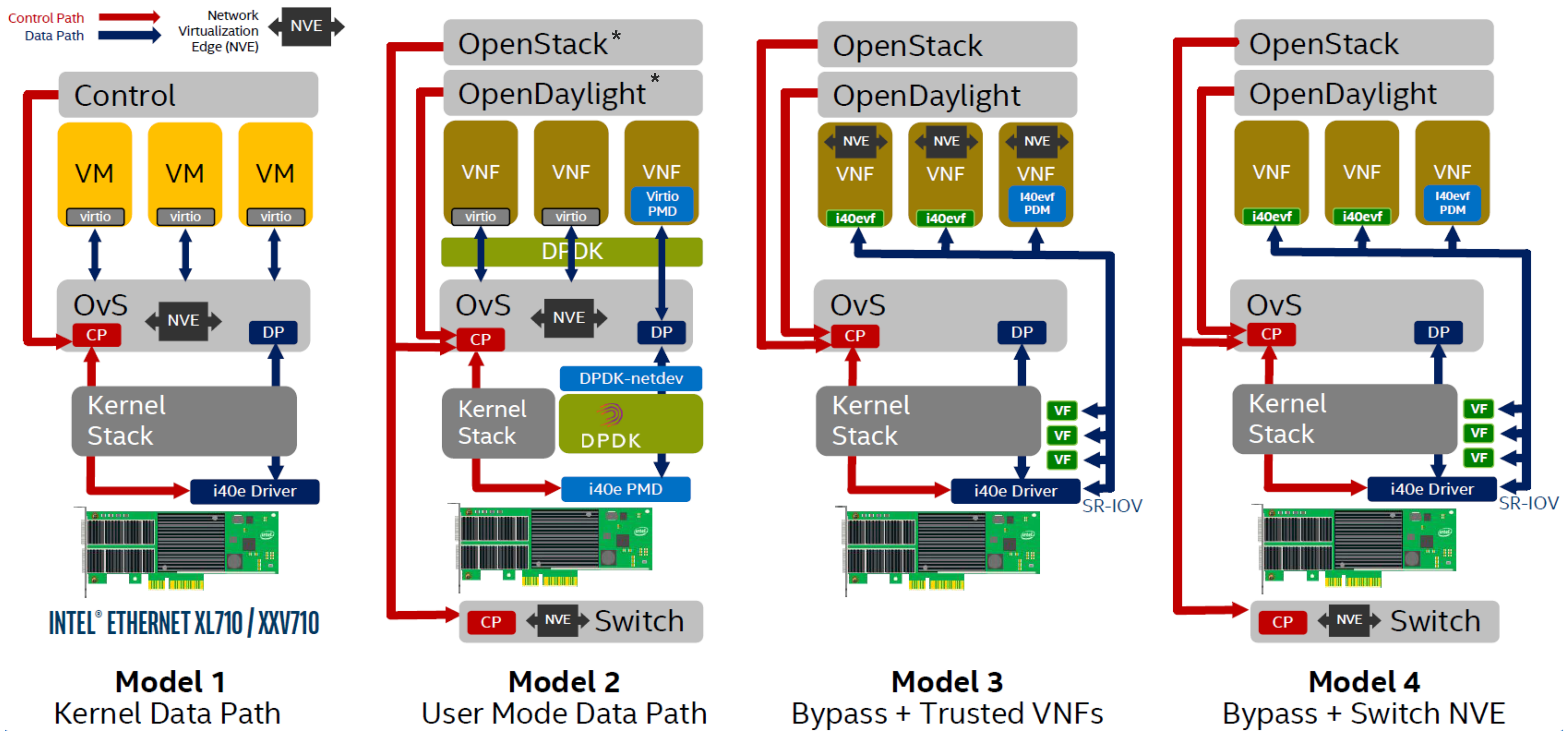
experience what's inside™

NOTE

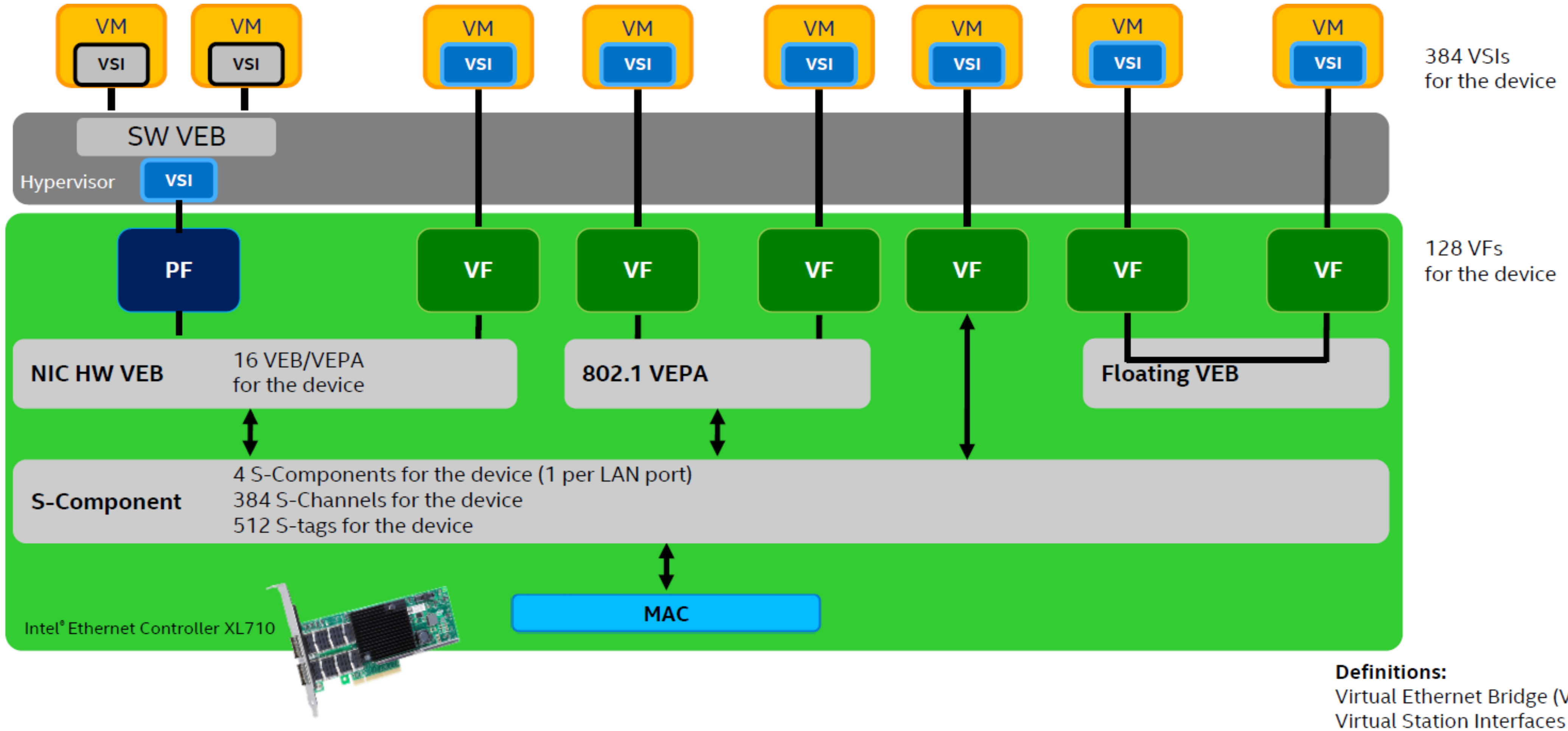
These slides were originally presented as part of a hands-on lab at the IEEE NFV/SDN conference in November 2016. They have been modified to make them more relevant to an audience that does not have access to the resources that were available at the time.

THE NEW CENTER OF POSSIBILITY

NFV MODELS – DPDK* SR-IOV USAGE



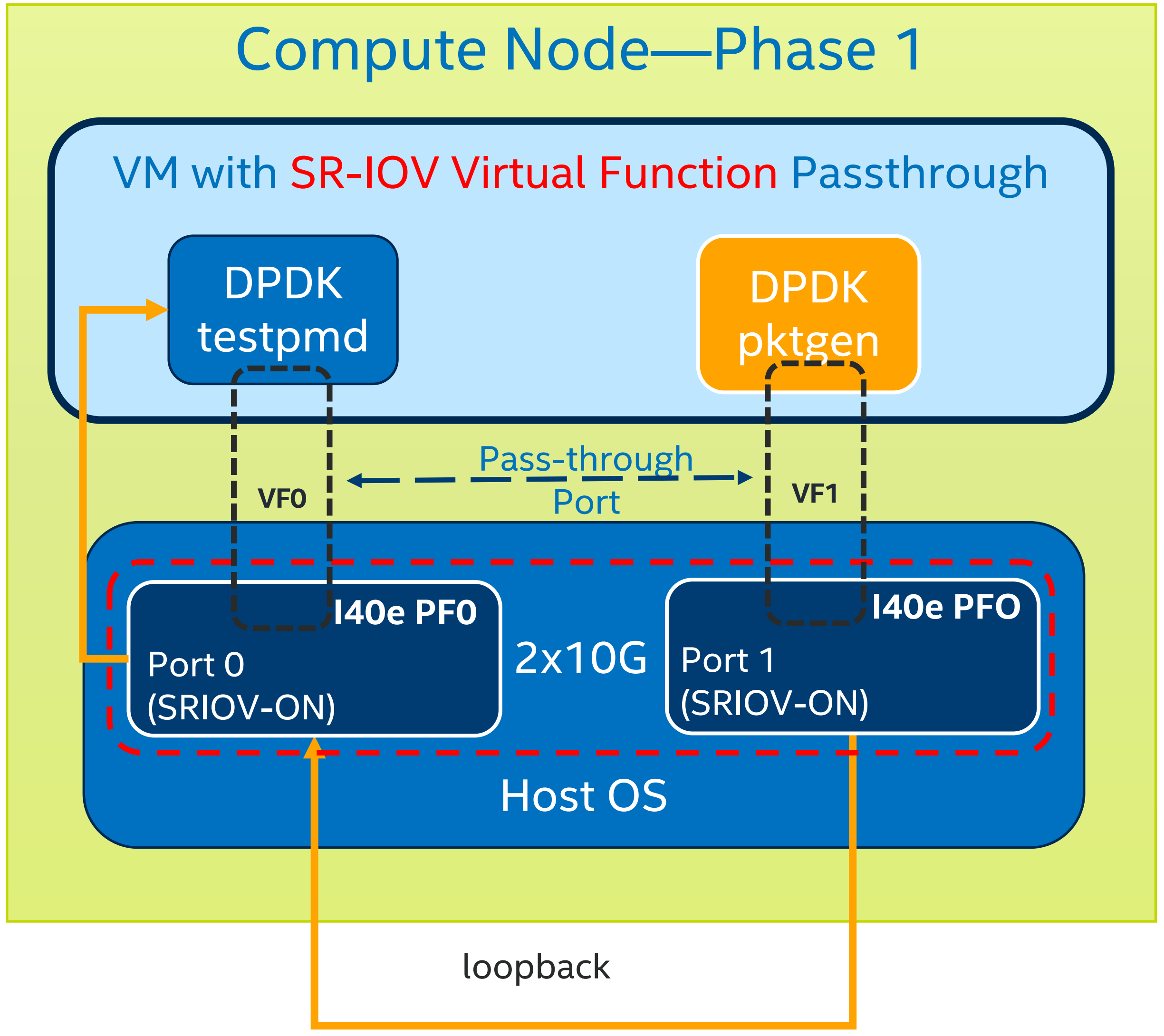
INTEL ETHERNET XL710 FAMILY (I/O VIRTUALIZATION)

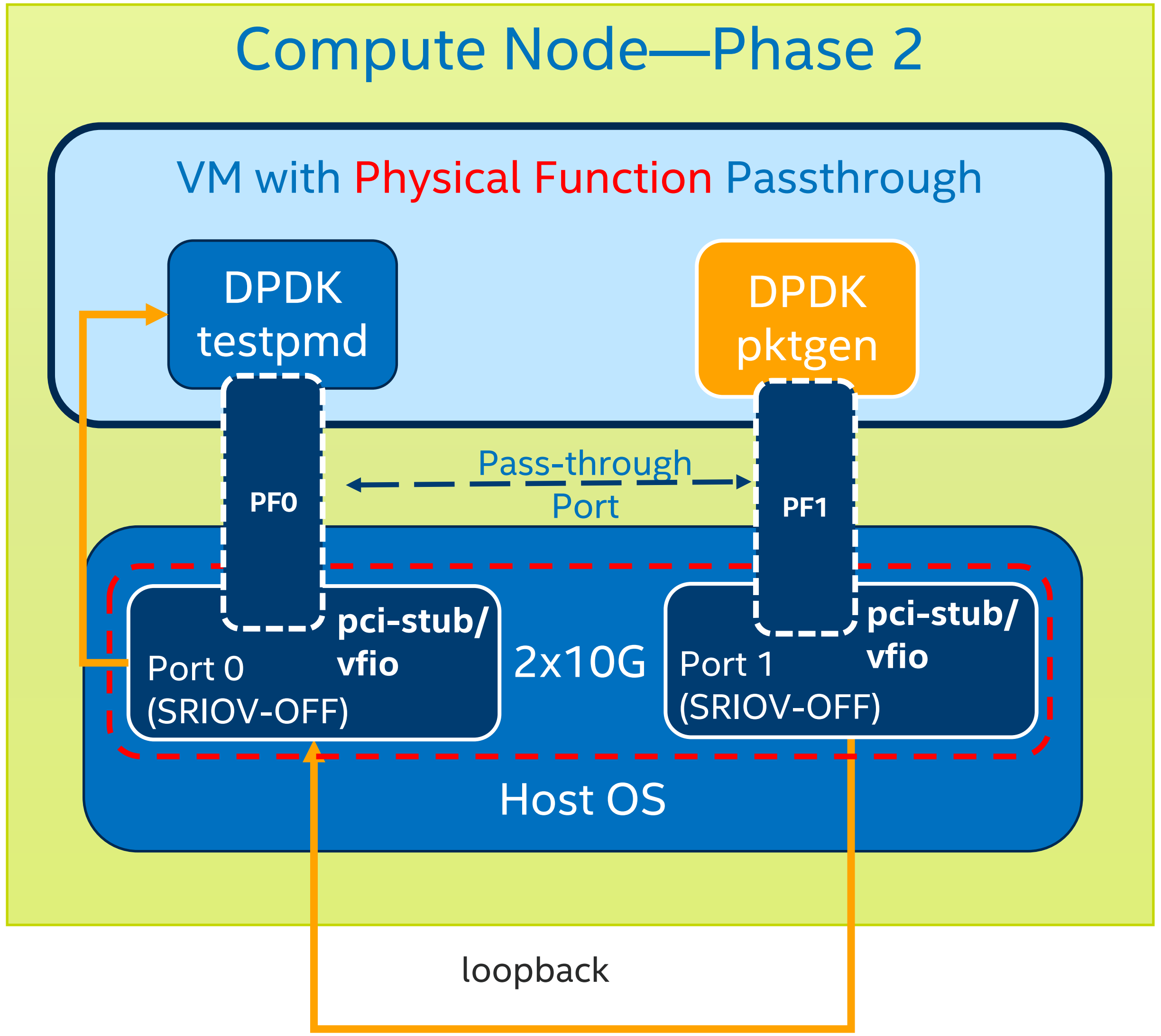


Definitions:
Virtual Ethernet Bridge (VEB)
Virtual Station Interfaces (VSIs)
Virtual Ethernet Port Aggregator (VEPA)

HANDS-ON SESSION (SR-IOV)

SR-IOV HANDS-ON: SYSTEM SETUP

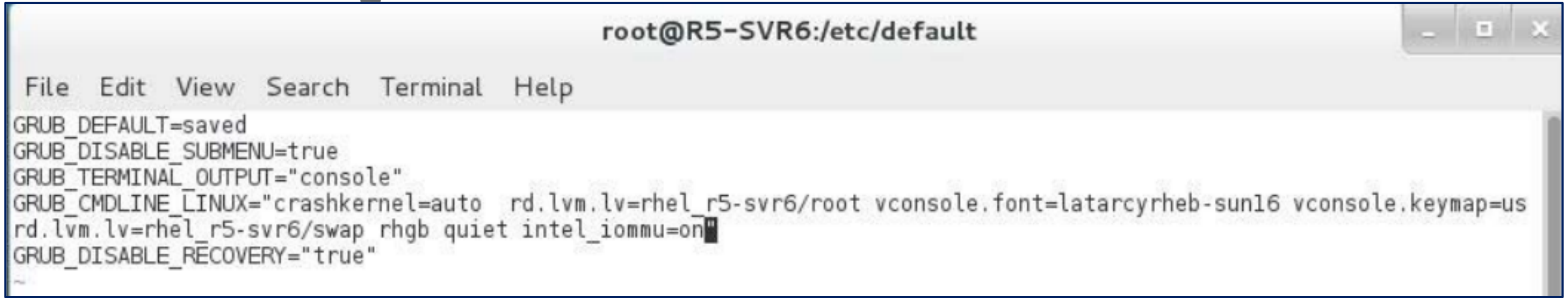




PREPARE COMPUTE NODE FOR I/O PASS-THROUGH

IOMMU support is required for VF to function properly when assigned to VM. The following boot parameter is required to enable IOMMU support for Linux* kernels

1. (Done) Before booting **compute node** OS, enable Intel VT features in BIOS
2. (Done) Append "`intel_iommu=on`" to the GRUB_CMDLINE_LINUX entry in /etc/default/grub



root@R5-SVR6:/etc/default

File Edit View Search Terminal Help

```
GRUB_DEFAULT=saved
GRUB_DISABLE_SUBMENU=true
GRUB_TERMINAL_OUTPUT="console"
GRUB_CMDLINE_LINUX="crashkernel=auto  rd.lvm.lv=rhel_r5-svr6/root vconsole.font=latarcyrheb-sun16 vconsole.keymap=us
rd.lvm.lv=rhel_r5-svr6/swap rhgb quiet intel_iommu=on"
GRUB_DISABLE_RECOVERY="true"
~
```

3. (Done) Update the **compute node** grub configuration using grub2-mkconfig command

```
$sudo grub2-mkconfig -o /boot/grub2/grub.cfg
```

3. (Done) Reboot the **compute node** for the iommu change to take effect
4. ➔Check the **compute node** kernel command line in Linux and look for "iommu=on"

```
$cat /proc/cmdline.
```

CREATE VIRTUAL FUNCTIONS

Linux* does not create VFs by default. The Intel Ethernet Controller X710 supports up to 32 VFs per port. The Intel® Ethernet Controller XL710 server adapters supports up to 64 VFs per port.

1. On the **compute node,** create the Virtual Functions:

```
# echo 4 > /sys/class/net/[INTERFACE NAME]/device/sriov_numvfs
```

(for kernel versions 3.8.x and above)

(For Kernel versions 3.7.x and below, to get 4 VFs per port) `#modprobe i40e max_vfs=4, 4`

2. On the **compute node,** verify that the Virtual Functions were created:

```
# lspci | grep 'X710 Virtual Function'
```

3. On the **compute node**, bring up the link on the virtual functions

```
# ip l set dev [INTERFACE NAME] up
```

4. You can assign a MAC address to each VF on the **compute node.**

```
# ip l set dev enp6s0f0 vf 0 mac aa:bb:cc:dd:ee:00
```

Upon successful VF creation, the Linux operating system automatically loads the i40 vf driver.

PREPARE HYPERVISOR – KVM/LIBVIRT METHOD

To simplify integration with VMs, SR-IOV Virtual Functions can be deployed as a pool of NICs in a libvirt network.

1. **Compute node**: Create an XML fragment/file that describes an SR-IOV network:

```
<network>
        <name>sr-iov-port0</name>
        <forward mode='hostdev' managed='yes'>
                <pf dev='[INTERFACE NAME]'/>
        </forward>
</network>
```

2. **Compute node**: Use virsh to create a network based on this XML fragment

```
# virsh net-define <sr-iov-network-description.xml>
```

3. **Compute node**: Activate the network.

```
# virsh net-start sr-iov-port0
```

PREPARE AND LAUNCH THE VM IMAGE – KVM/LIBVIRT METHOD

On the **`compute node`**, once you have a libvirt network based on the SR-IOV virtual functions, Add a NIC from that network

1. Create an XML fragment/file that describes the NIC, and optionally add a MAC address

```
<interface type='network'>
        <mac address='aa:bb:cc:dd:ee:ff'/>
        <source network='sr-iov-port0'/>
</interface>
```

2. Use `#virsh edit [VM NAME]` to insert the XML fragment into the VM Domain definition

```
<controller type='virtio-serial' index='0'>
        <address …/>
</controller>
<interface type='network'>
        <mac address='aa:bb:cc:dd:ee:ff'/>
        <source network='sr-iov-port0'/>
</interface>
```

3. Launch the VM

```
#virsh start [VM NAME]
```

PREPARE THE HYPERVISOR—QEMU* METHOD

Using qemu directly is not as elegant, but it works as well

1. Get the PCI Domain:Bus:Slot.Function information for the VF

```
# lshw –c network -businfo
```

2. Load the pci-stub driver if necessary

```
# modprobe pci-stub
```

3. Unbind the NIC PCI device from the i40e driverFollow the below steps to passthrough each VF port in VM

```
# echo "8086 154c" > /sys/bus/pci/drivers/pci-stub/new_id

# echo [PCI Domain:Bus:Slot.Function] > /sys/bus/pci/devices/[PCI
Domain:Bus:Slot.Function]/driver/unbind

# echo [PCI Domain:Bus:Slot.Function] > /sys/bus/pci/drivers/pci-
stub/bind
```

LAUNCH THE IMAGE – QEMU* METHOD

1. Start the virtual machine by running the following command

```
# qemu-system-x86_64 –enable-kvm                           \
          –smp 4 –cpu host –m 4096 –boot c                 \
          –hda [your image]                        \
         -nographic –no-reboot                             \
         -device pci-assign, host=[VF PCI Bus:Slot.Function]              \
```

INSTALL DPDK ON THE SR-IOV VF ON THE VIRTUAL MACHINE

Now that the compute node has been set up, get the virtual machine ready

1. From the Jump Server, ssh into your assigned Virtual Machine (`$ssh HostVM-____`)
2. Scripts for the lab are located in `/home/user/training/sr-iov-lab`.
3. View the Virtual Functions that have already been loaded into the Virtual Machine.

```
$ ./00_show_net_info.sh
```

1. Steps 01-03 are only necessary if the DPDK lab was done before the SR-IOV lab
2. Compile DPDK and load it onto the Virtual Functions

```
$ ./04_build_load_dpdk_on_vf.sh
```

1. Write down the MAC addresses that were displayed in the previous step

TESTPMD_MAC=___:___:___:___:___:___
PKTGEN_MAC   =___:___:___:___:___:___

BUILD AND RUN TESTPMD AND PKTGEN

1. On the **virtual machine**, build and run testpmd. Look at the parameters in the build script. Running testpmd requires that you know the MAC address of the port on which pktgen is going to run. This was output to the console in step 04.

```
# 05_build_start_testpmd_on_vf.sh [PKTGEN MAC]
```

1. Look at the command line to see what parameters we are using.
2. Open another SSH session into your assigned virtual machine (HostVM-____)
3. Build and launch pktgen.

```
# 06_build_start_pktgen_on_vf.sh
```

4. You need to know the MAC address of the port where pktgen is going to send packets, which is the port on which testpmd is waiting. You can find the testpmd port when you launch testpmd. You'll see lines that look like this:

```
Configuring Port 0 (socket 0)
Port 0: 52:54:WW:XX:YY:ZZ (This is the testpmd MAC address)
Checking link statuses...Port 0 Link Up - speed 10000 Mbps - full-duplex
```

   Note: You can also get the testpmd MAC address from step 04

5. Allow CRC stripping. In a VM, using a VF, we can't disable CRC stripping. Edit pktgen-port-cfg.c and change line 94 to "`.hw_strip_crc   = 1,`"

```
#vi /usr/src/pktgen-3.0.14/pktgen-port-cfg.c
```

   Note: testpmd also has this problem, but we take care of it on the command line: `--crc-strip`

GENERATE AND MEASURE TRAFFIC FLOW WITH SR-IOV

Now that we have pktgen and testpmd launched, start the traffic

1. In pktgen, set the mac 0 address to point to the testpmd SR-IOV port

```
> set mac 0 [TESTPMD MAC]
```

2. Start generating traffic

```
> start 0
```

3. View stats in testpmd

```
> show port stats 0
```

4. Record the RX and TX info

Mbit/s RX:___ TX:___
PPS RX:___ TX:___