Next Generation Intel Ethernet Drivers

ixd & idpf

Sridhar Samudrala
Jayaprakash Shanmugam

Agenda

- Common Networking IP
- Device Operating Modes
- Product Configurations
- ixd/idpf design requirements
- Original monolithic idpf
- Refactored idpf architecture
- ixd architecture and design
- Advanced Features

Common Networking Subsystem IP

Unified IP and SW/FW Architecture – Mutiple Products

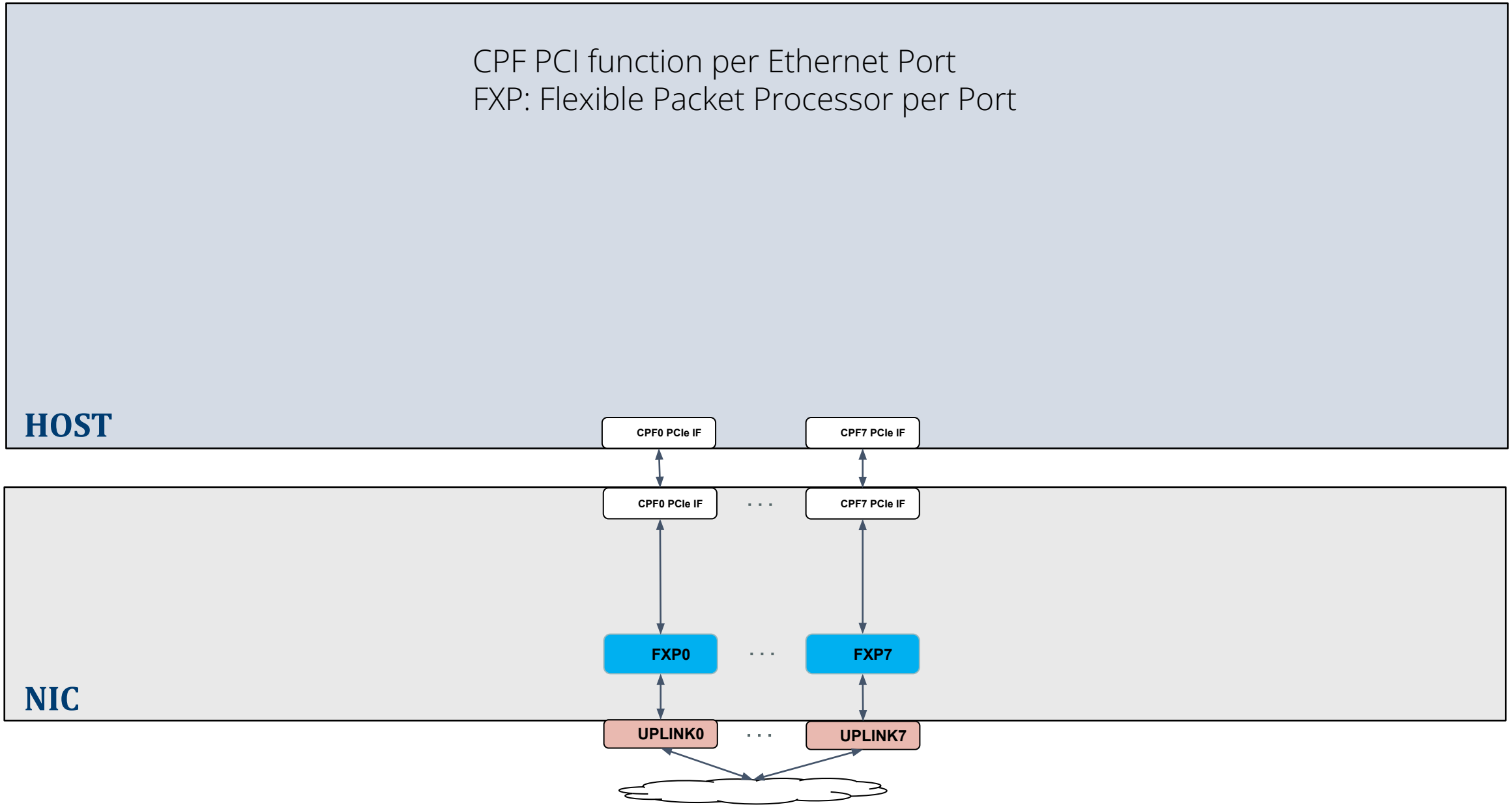
- Infrastructure Processing Unit (IPU)
    - Control Plane running in a compute complex
- Discrete PCI NIC aka Foundational NIC (FNIC)
- Edge Processing Unit (EPU)
    - SoC based platform
        - NIC Mode (SOC_NIC)
        - Switch Mode (SOC_SWITCH)

Device Operating Modes

- PF per Port (PFP)
    - Separate Control Plane Function(CPF) for each ethernet port
    - Traditional deployment model for foundational ethernet NICs.
    - Supported in
        - FNIC
        - SOC_NIC
- PF per Device (PFD)
    - Single Control Plane Function (CPF) for the entire device
    - Common deployment model for IPUs.
    - Supported across all product families
        - FNIC
        - IPU
        - SOC_NIC
        - SOC_SWITCH

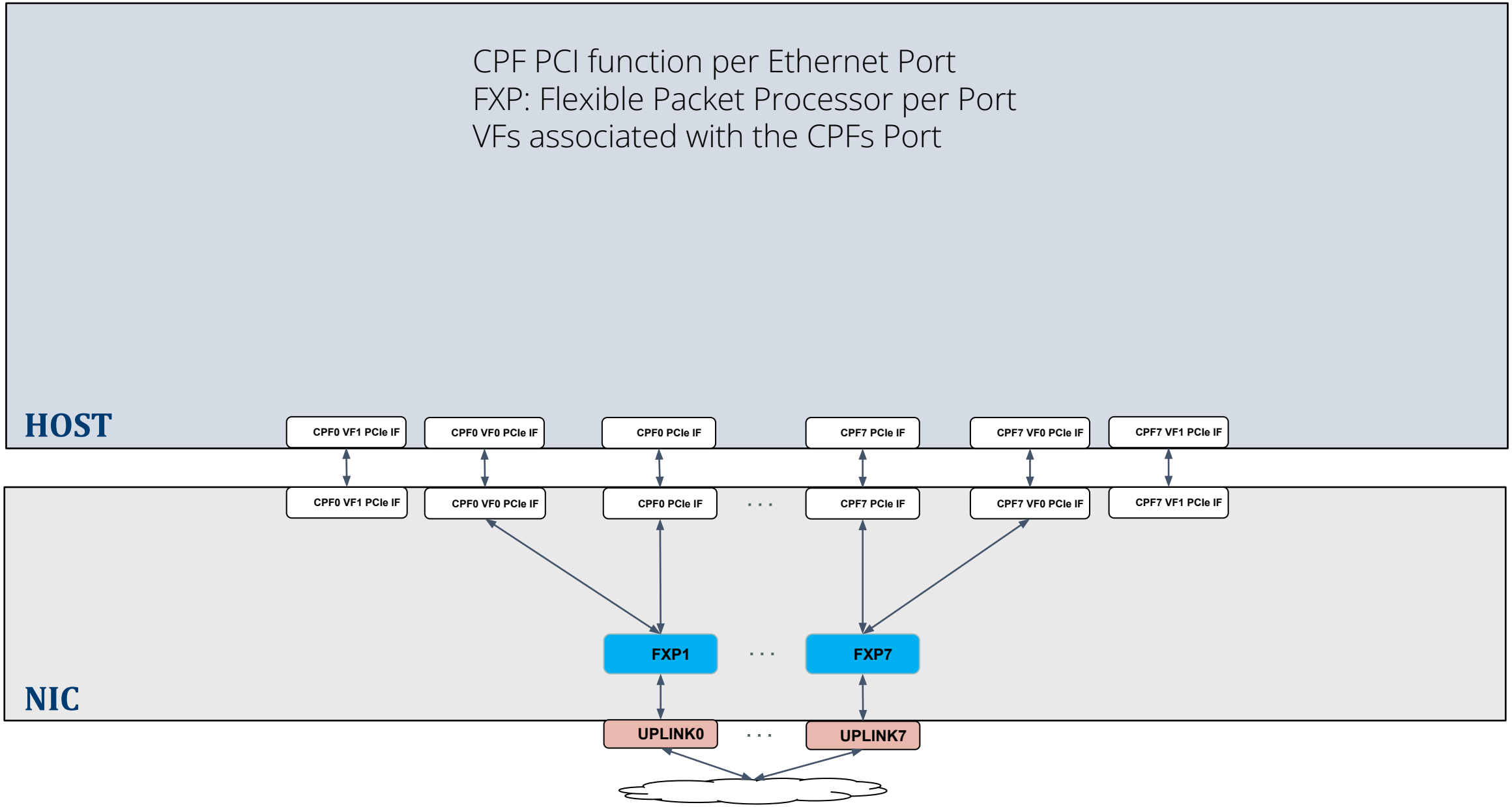
FNIC & SOC_NIC: PF per Port

CPF PCI function per Ethernet Port
FXP: Flexible Packet Processor per Port



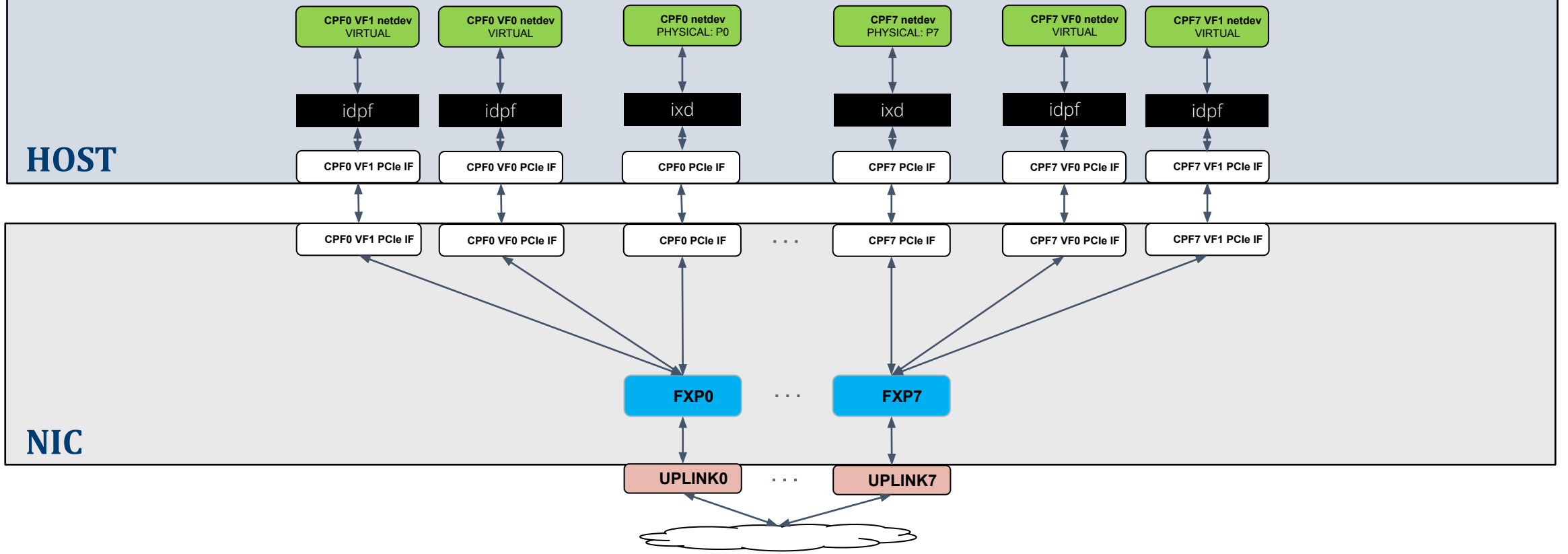
FNIC & SOC_NIC with VFs : PF per Port

CPF PCI function per Ethernet Port
FXP: Flexible Packet Processor per Port
VFs associated with the CPFs Port



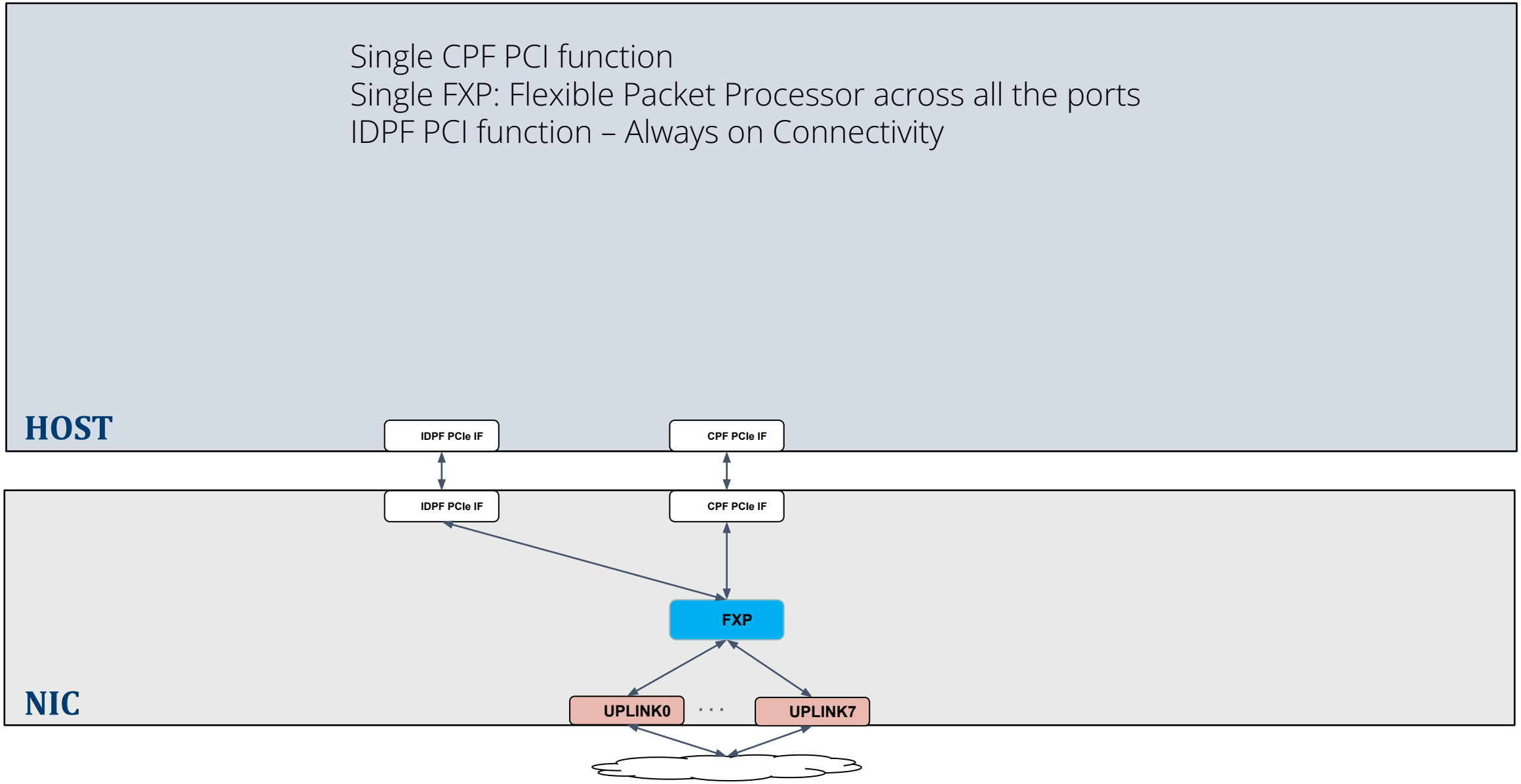
FNIC & SOC_NIC with VFs : PF per Port : Drivers

CPF PCI function per Ethernet Port
FXP: Flexible Packet Processor per Port
VFs associated with the CPFs Port
ixd: CPF driver idpf: VF driver



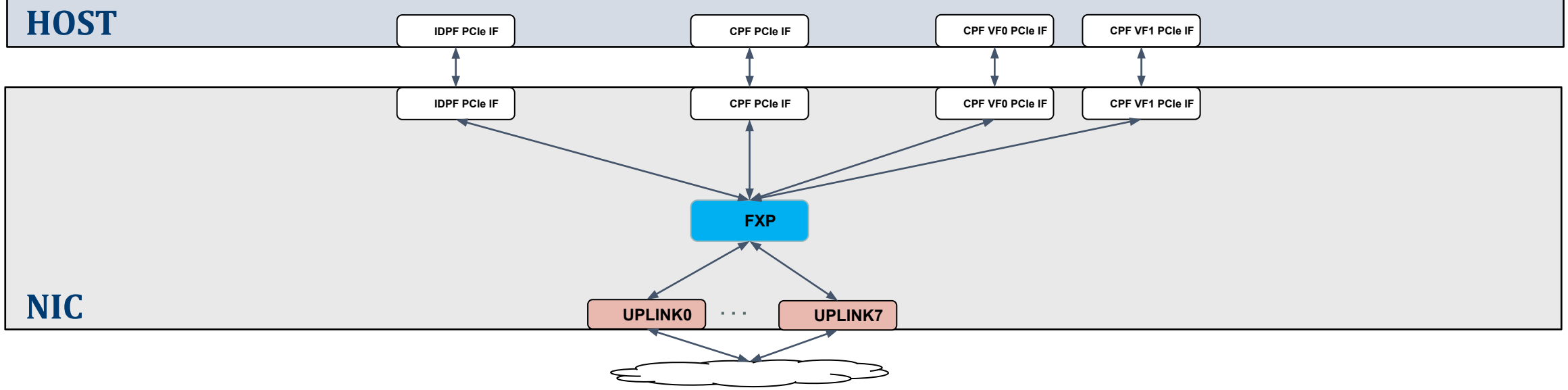
FNIC & SOC_NIC: PF per Device

Single CPF PCI function
Single FXP: Flexible Packet Processor across all the ports
IDPF PCI function – Always on Connectivity



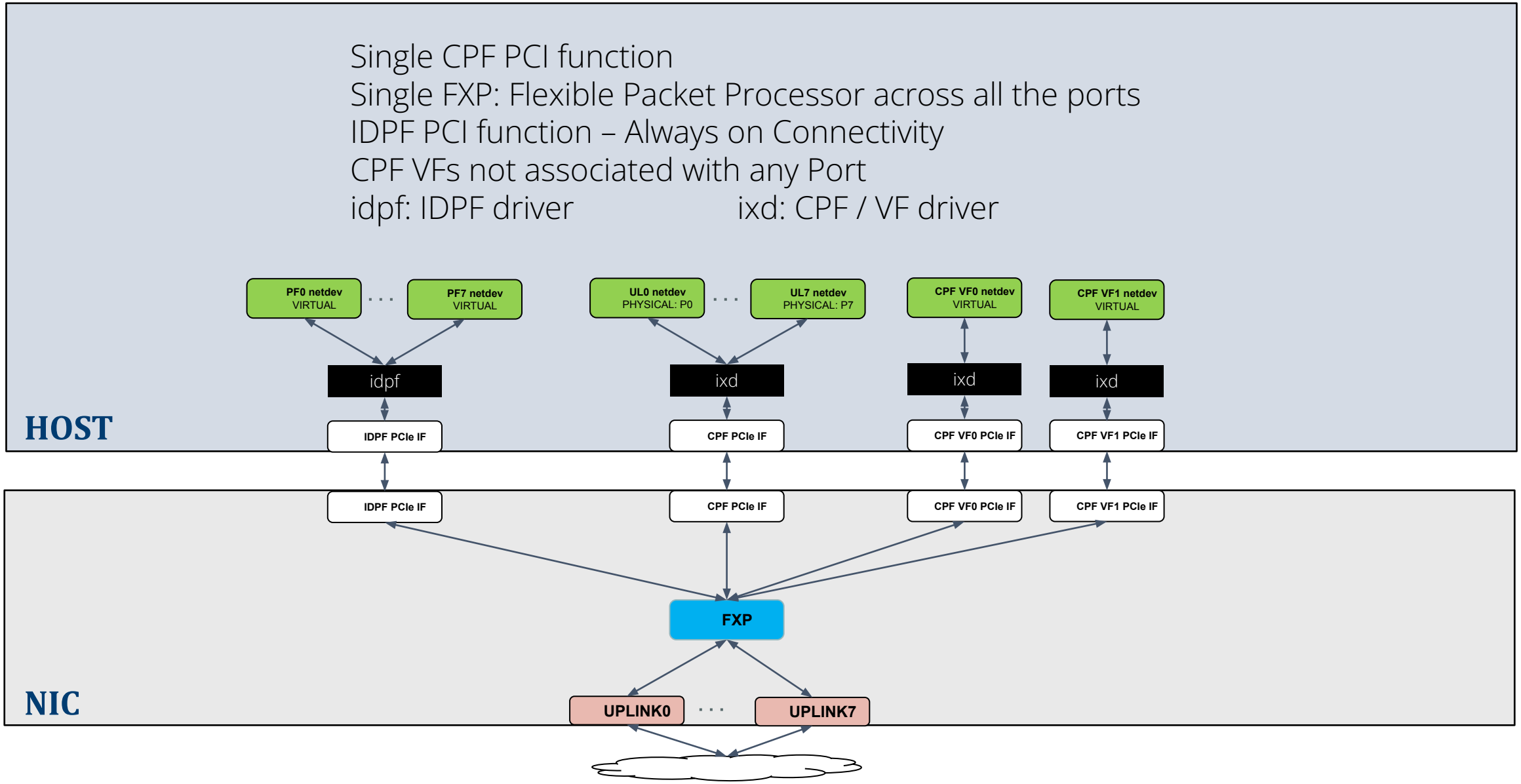
FNIC & SOC_NIC with VFs : PF per Device

Single CPF PCI function
Single FXP: Flexible Packet Processor across all the ports
IDPF PCI function – Always on Connectivity
CPF VFs not associated with any Port



FNIC & SOC_NIC with VFs : PF per Device : Drivers

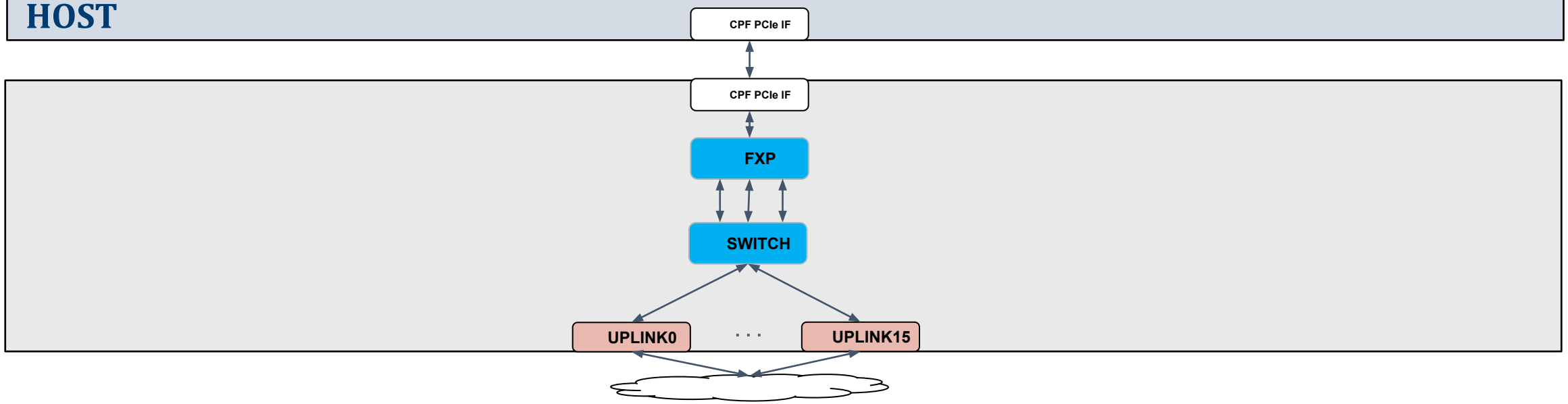
Single CPF PCI function
Single FXP: Flexible Packet Processor across all the ports
IDPF PCI function – Always on Connectivity
CPF VFs not associated with any Port
idpf: IDPF driver ixd: CPF / VF driver



SOC_SWITCH : PF per device

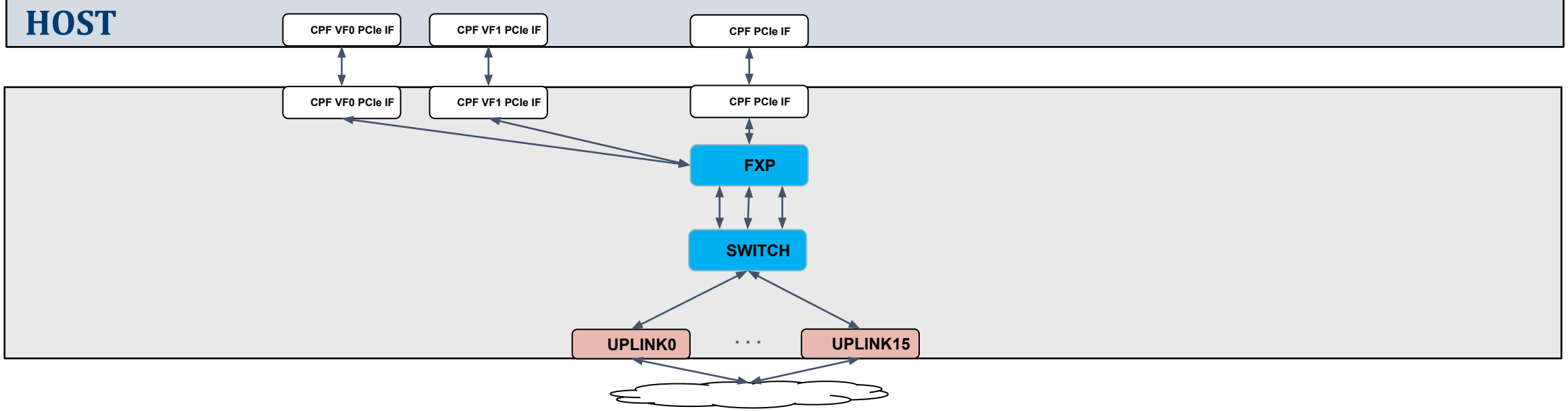
Single CPF PCI function
Single FXP: Flexible Packet Processor across all the ports
Low latency Ethernet Switch enables port to port switching

HOST



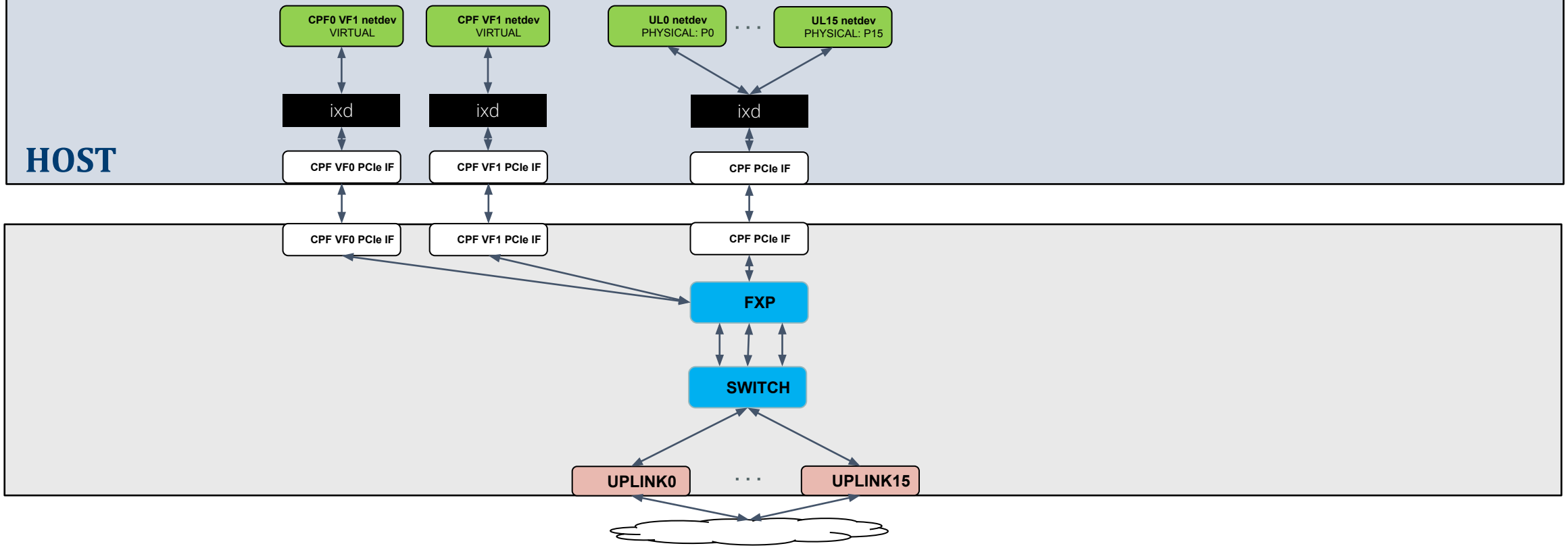
SOC_SWITCH with VFs : PF per Device

Single CPF PCI function
Single FXP: Flexible Packet Processor across all the ports
Low latency Ethernet Switch enables port to port switching
CPF VFs not associated with any port



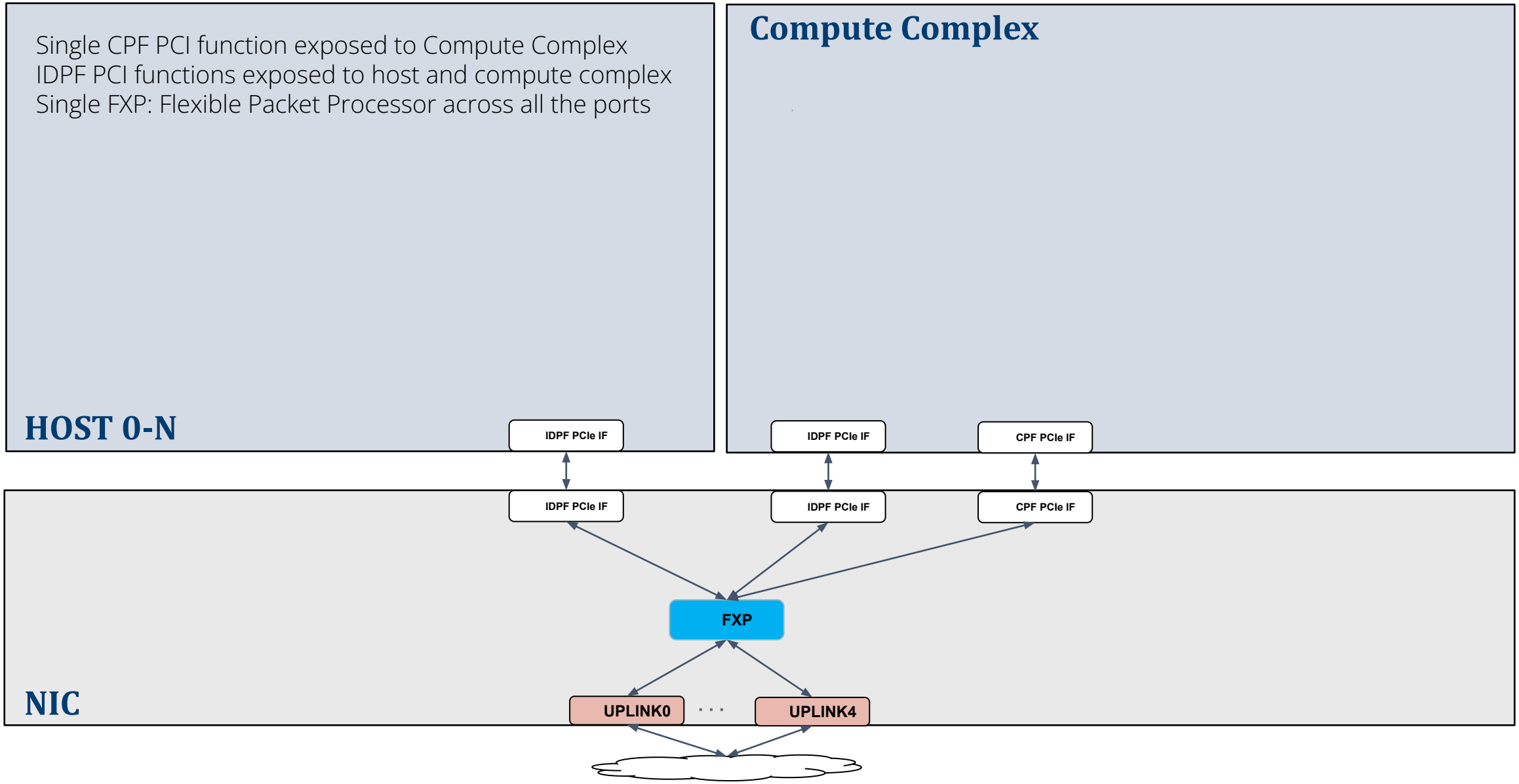
SOC_SWITCH with VFs : PF per Device : Drivers

Single CPF PCI function
Single FXP: Flexible Packet Processor across all the ports
Low latency Ethernet Switch enables port to port switching
CPF VFs not associated with any port
ixd: CPF / VF driver



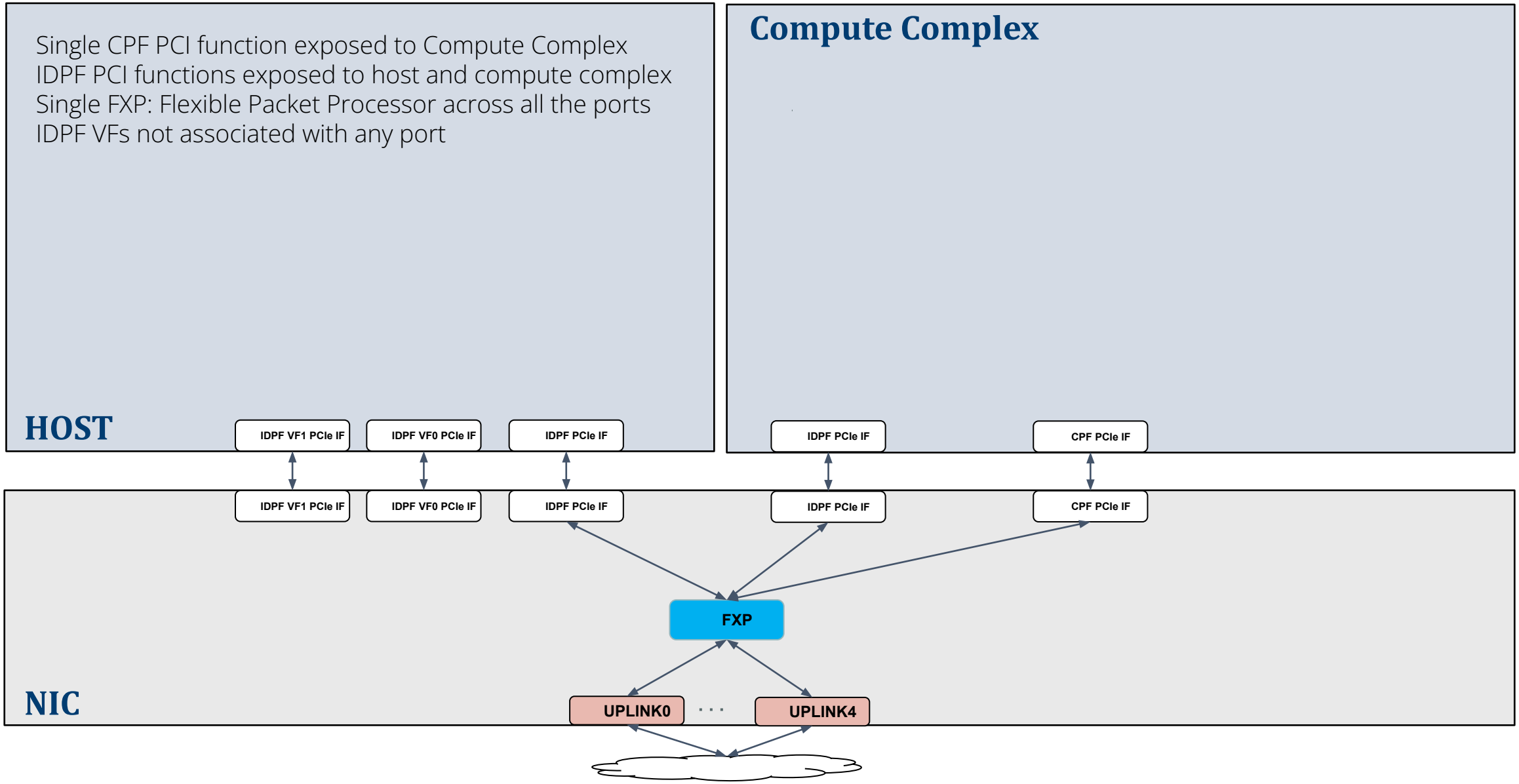
IPU : PF per Device

Single CPF PCI function exposed to Compute Complex
IDPF PCI functions exposed to host and compute complex
Single FXP: Flexible Packet Processor across all the ports



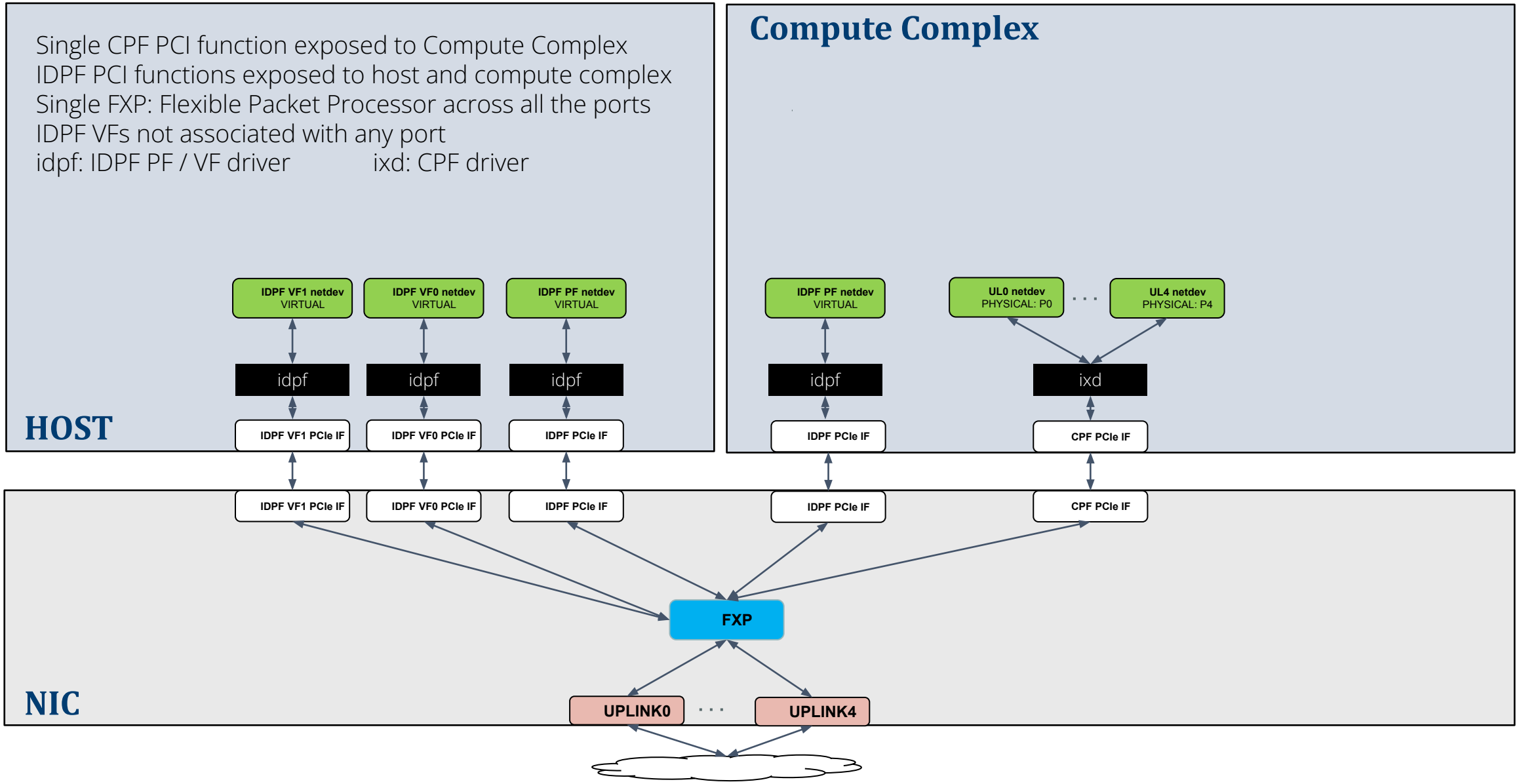
IPU with VFs : PF per Device

Single CPF PCI function exposed to Compute Complex
IDPF PCI functions exposed to host and compute complex
Single FXP: Flexible Packet Processor across all the ports
IDPF VFs not associated with any port



IPU with VFs : PF per Device : Drivers

Single CPF PCI function exposed to Compute Complex
IDPF PCI functions exposed to host and compute complex
Single FXP: Flexible Packet Processor across all the ports
IDPF VFs not associated with any port
idpf: IDPF PF / VF driver ixd: CPF driver



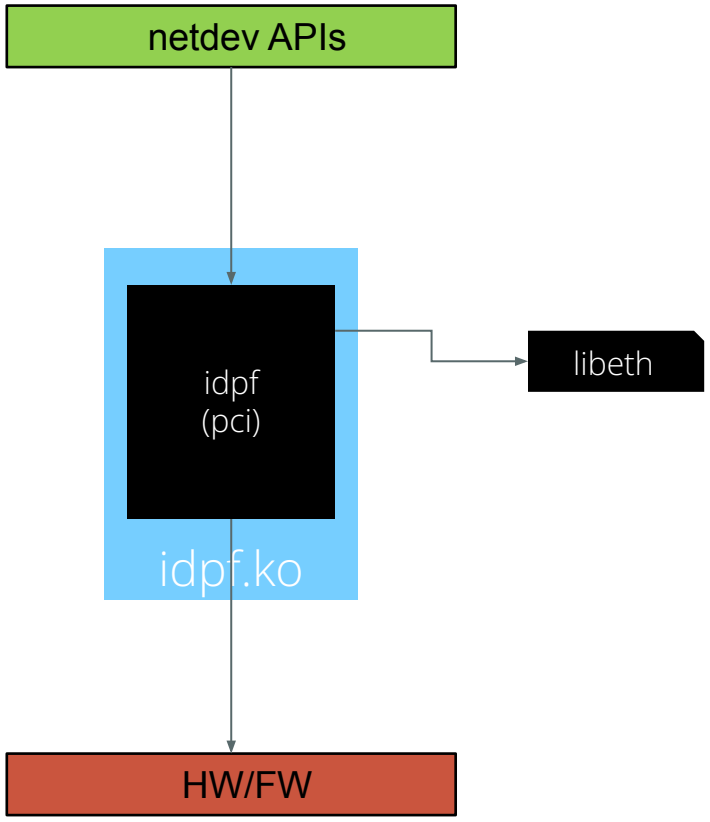
idpf/ixd design requirements

- idpf must be independently deployable
  - Loading idpf should not force loading of ixd
    - IPU Host systems using the standard IDPF interface
    - IDPF Virtual Functions(VFs) assigned to VMs
- ixd must be independently removable
  - Unloading ixd must not require unloading idpf
  - idpf continues to provide connectivity while ixd is upgraded, reloaded or removed
    - **Always-On Connectivity NIC** using the IDPF driver remains operational while the IXD control plane is unloaded.
- Avoid Code duplication across drivers sharing a common dataplane

Code sharing between IDPF and IXD

- Goal
  - Avoid **code duplication across drivers sharing a common dataplane**
    - data plane driver : idpf
    - data + control plane driver : ixd
- Mechanisms Used
  - Shared libraries (libie/libeth)
    - Common helper functions used by both drivers
  - Shared Ethernet auxiliary driver (idpf)
    - The monolithic idpf is split into a core pci driver and ethernet aux driver
      - Built as a single driver module (idpf.ko)
    - idpf acts as ethernet aux driver for
      - idpf ethernet auxiliary devices
      - ixd ethernet auxiliary devices

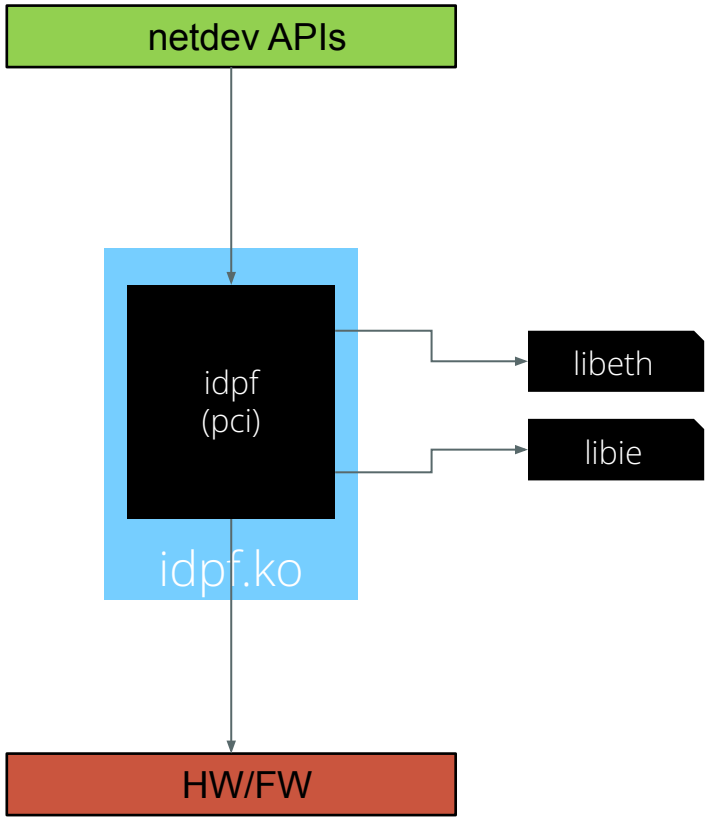
idpf pci driver



- IDPF (Infrastructure Datapath Function)
  - An open vedor-neutral PCIe programming interface specification for modern ethernet devices
  - Standardized by OASIS idpf technical committee
  - PCI-SIG assigned PCI programming interface of 0x1
    - network:ethernet:idpf (02:00:01)

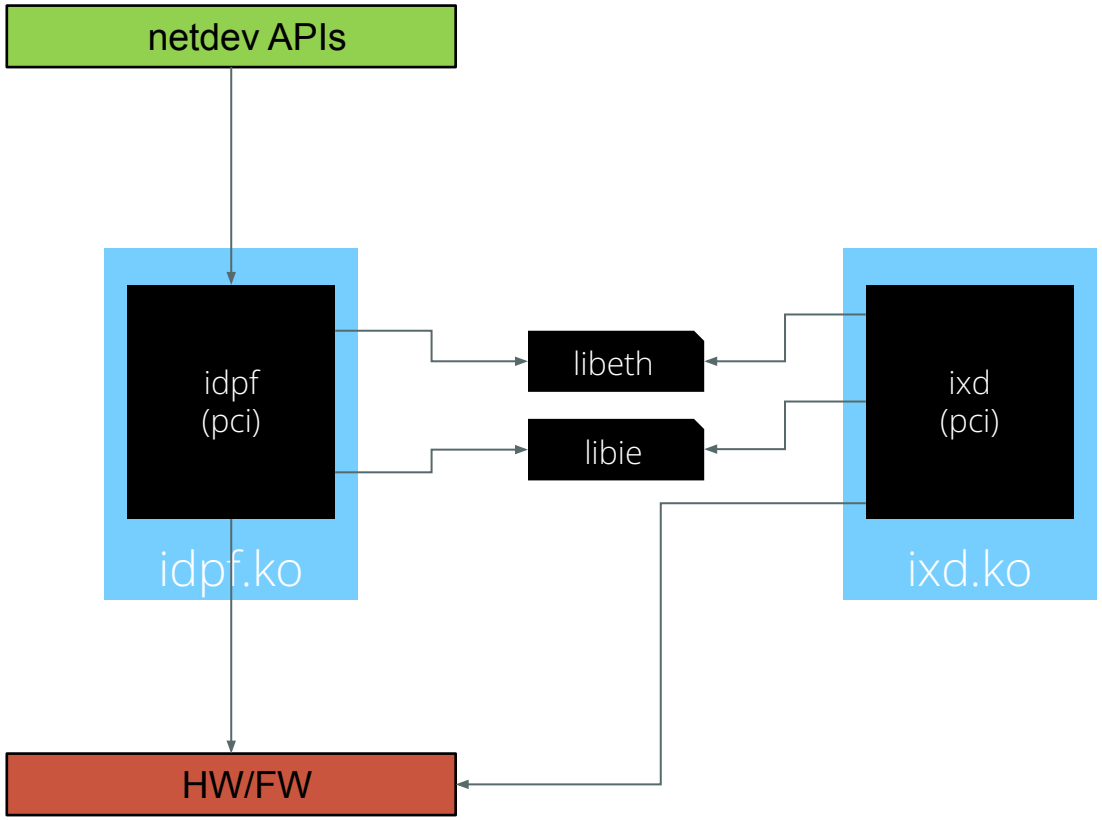
- idpf pci driver
  - mailbox communication to FW (virtchnl2)
  - pci resource management (function caps, msi-x vectors)
  - vport/queue/rss configuration, vector mapping(virtchnl2)
  - netdev creation and entry point for all the netdev triggered operations
    - net_device_ops
    - ethtool_ops
  - tx/rx, stateless offloads
  - xdp, rdma, ptp
  - ethtool flow steering(virtchnl2)
- libeth
  - pagepool helper functions
  - xdp helper functions

idpf pci driver + libie



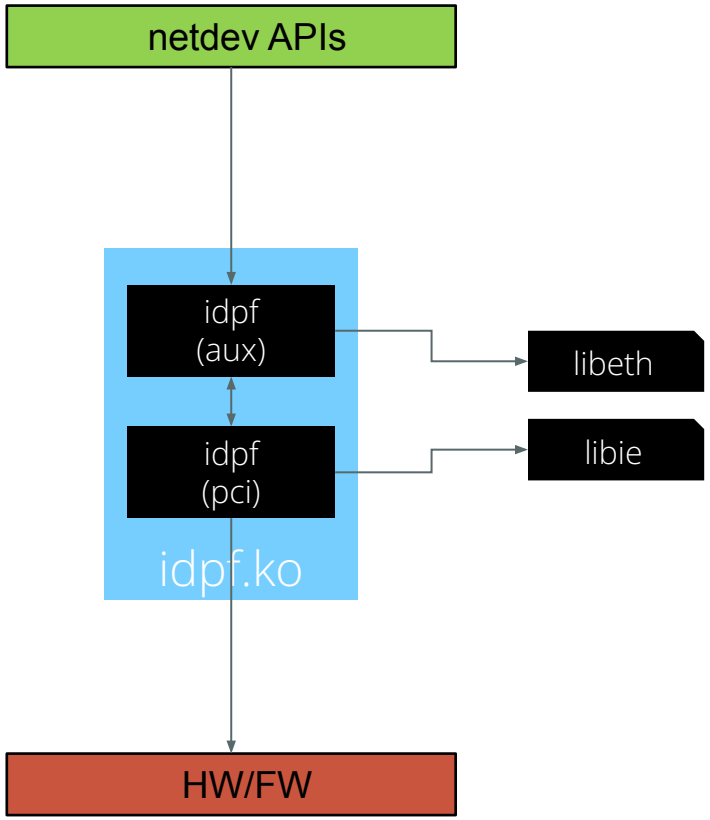
- idpf pci driver
  - mailbox communication to FW (virtchnl2)
  - pci resource management (function caps, msi-x vectors)
  - vport/queue/rss configuration, vector mapping(virtchnl2)
  - netdev creation and entry point for all the netdev triggered operations
    - net_device_ops
    - ethtool_ops
  - tx/rx, stateless offloads
  - xdp, rdma, ptp
  - ethtool flow steering(virtchnl2)
- libeth
  - pagepool helper functions
  - xdp helper functions
- libie
  - control queue helper functions
  - pci helper functions

idpf pci driver + libie + introduce ixd



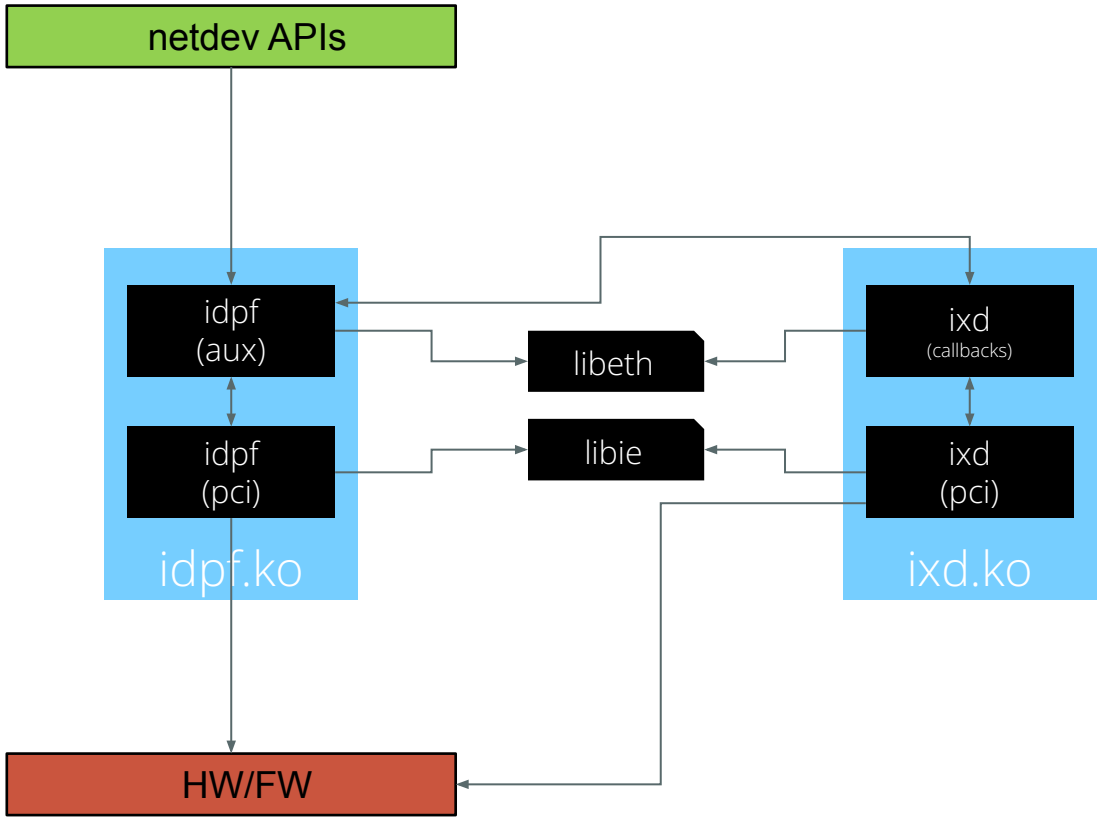
- idpf pci driver
  - mailbox communication to FW (virtchnl2)
  - pci resource management (function caps, msi-x vectors)
  - vport/queue/rss configuration, vector mapping(virtchnl2)
  - netdev creation and entry point for all the netdev triggered operations
    - net_device_ops
    - ethtool_ops
  - tx/rx, stateless offloads
  - xdp, rdma, ptp
  - ethtool flow steering(virtchnl2)
- libeth
  - pagepool helper functions
  - xdp helper functions
- libie
  - control queue helper functions
  - pci helper functions
- ixd pci driver
  - mailbox communication to FW (virtchnl2, cpchnl2)
  - pci resource management (function caps, msi-x vectors)

idpf split pci/aux driver



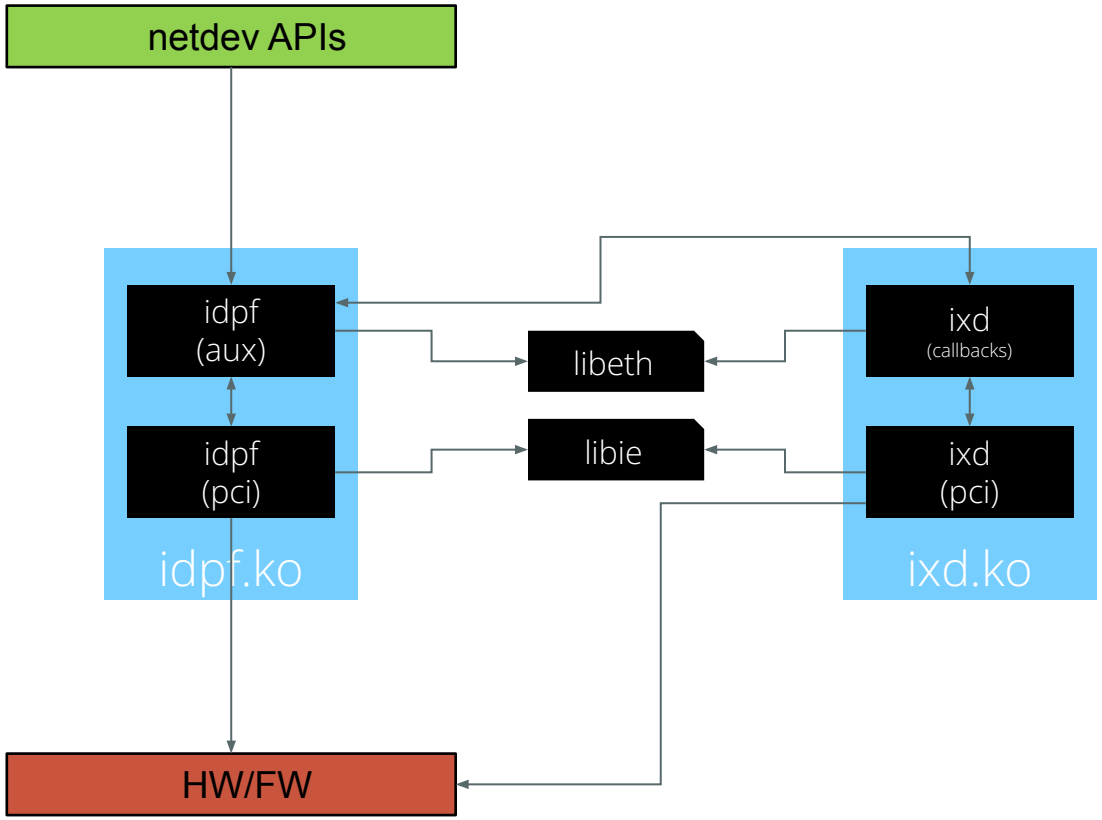
- idpf pci driver
  - mailbox communication to FW (virtchnl2)
  - pci resource management (function caps, msi-x vectors)
  - creates idpf eth auxiliary devices (idpf.eth)
  - rdma
- idpf aux driver
  - loads on idpf.eth aux devices
  - vport/queue/rss configuration, vector mapping(virtchnl2)
  - netdev creation and entry point for all the netdev triggered operations
    - net_device_ops
    - ethtool_ops
  - tx/rx, stateless offloads
  - xdp, rdma, ptp
  - ethtool flow steering (virtchnl2)
  - parent device event handler (link, reset etc)
- libeth
  - pagepool helper functions
  - xdp helper functions
- libie
  - control queue helper functions
  - pci helper functions

idpf split pci/aux driver + ixd



- idpf pci driver
  - mailbox communication to FW (virtchnl2)
  - pci resource management (function caps, msi-x vectors)
  - creates idpf eth auxiliary devices (idpf.eth)
  - rdma
- idpf aux driver
  - loads on idpf.eth & ixd.eth aux devices
  - vport/queue/rss configuration, vector mapping(virtchnl2)
  - netdev creation and entry point for all the netdev triggered operations
    - net_device_ops
    - ethtool_ops
  - tx/rx, stateless offloads
  - xdp, rdma, ptp
  - ethtool flow steering (virtchnl2)
  - parent device event handler (link, reset etc)
- libeth
  - pagepool helper functions
  - xdp helper functions
- libie
  - control queue helper functions
  - pci helper functions
- ixd pci driver
  - mailbox communication to FW (virtchnl2, cpchnl2)
  - pci resource management (function caps, msi-x vectors)
  - creates ixd eth auxiliary devices (ixd.eth)
  - ixd callbacks
    - virtchnl send

idpf split pci/aux driver + ixd advanced features



- idpf aux driver
  - additional net_dev_ops, ethtool_ops
  - xfrm_dev_ops, dcbnl_rtnl_ops
  - netdevice_notifiers, switchdev_notifiers
- ixd pci driver
  - mailbox communication to FW (virtchnl2, cpchnl2)
  - pci resource management (function caps, msi-x vectors)
  - creates ixd eth auxiliary devices (ixd.eth)
  - device level capabilities
  - resource provisioning to VFs/SFs
  - switchdev, representor netdevs, exception path
  - subfunctions
  - rdma, ptp
  - nvm update, package load
  - ixd callbacks
    - virtchnl send
    - fxp programming (sem, lem, wcm)
      - ethtool flow steering
      - tc flower
      - fdb offload
    - cxp programming (ipsec)
    - link management
    - qos, dcb, lag etc
- libeth
  - pagepool helper functions
  - xdp helper functions
- libie
  - control queue helper functions
  - pci helper functions
  - package helper functions
  - irq helper functions
  - ptp, rdma, subfunction, eswitch helper functions

Advanced Features : Require networking API extensions

- Resource Provisioning
  - devlink port params / devlink port function set
    - maximum tx/rx queues per VF/SF
    - maximum vports per VF
    - TX timestamp latches per VF
    - minipackage load/unload

- Device Operating Mode Change
  - devlink dev param <dev_mode>
    - CPF per Device
    - CPF per Port

- IXD VFs(vCPFS) in PFD mode
  - Trusted and can share the device's forwarding(FXP) pipeline
  - Minipackages enable each vCPF to own an independent dataplane
    - Can be provisioned dynamically after VF creation, but before the driver is loaded
    - Allow direct programming of flow rules
  - Direct TX timestamping

Thank you