

VPP

The Universal Fast Dataplane



FD.io: The Universal Fast Dataplane

• Project at Linux Foundation • FD.io Scope

- Multi-party
- Multi-project

• Software Dataplane

- High throughput
- Low Latency
- Feature Rich
- Resource Efficient
- Bare Metal/VM/Container
- Multiplatform



- **Network IO** – NIC/vNIC <-> cores/threads
- **Packet Processing** – Classify / Transform / Prioritize / Forward / Terminate
- **Dataplane Management Agents** – Control Plane

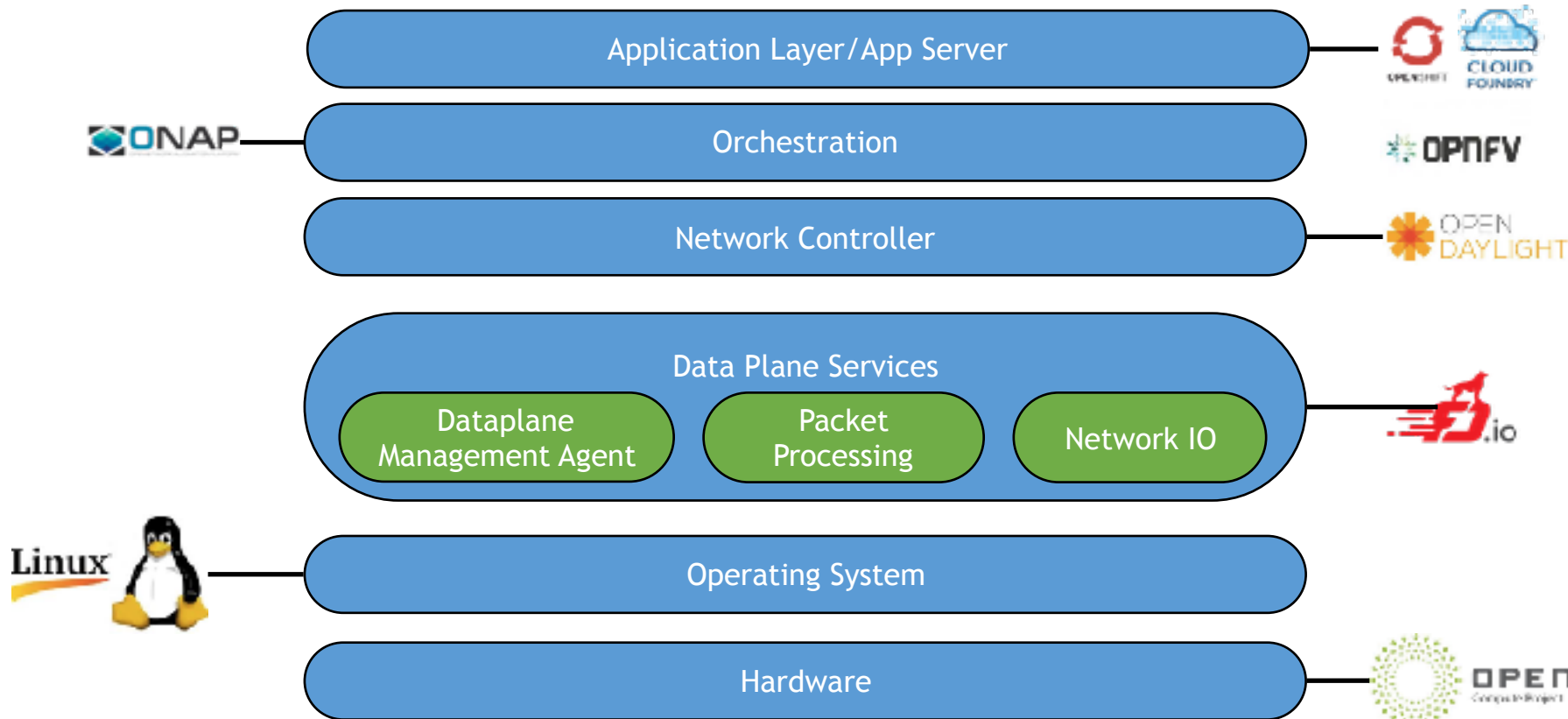
Bare Metal/VM/Container

Dataplane Management Agent

Packet Processing

Network IO

Fd.io in the overall stack



Multiparty: Broad Membership

Service Providers



Network Vendors



Chip Vendors



Integrators



Multiparty: Broad Contribution

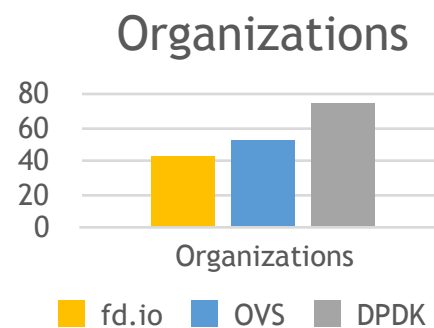
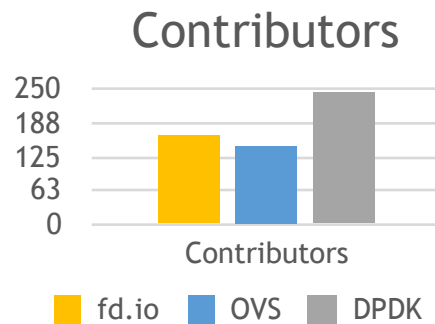
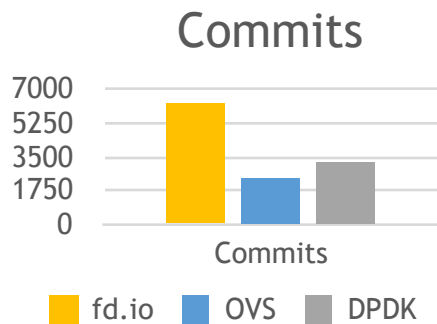


Universitat Politècnica de Catalunya (UPC)

Code Activity

- In the period since its inception, fd.io has more commits than OVS and DPDK combined, and more contributors than OVS

| 2016-02-11 to 2017-04-03 | Fd.io | OVS | DPDK |
|-----------------------------|-------|------|------|
| Commits | 6283 | 2395 | 3289 |
| Contributors | 163 | 146 | 245 |
| Organizations | 42 | 52 | 78 |



Multiproject: Fd.io Projects

Dataplane Management Agent

Honeycomb

hc2vpp

Packet Processing

NSH_SFC

ONE

TLDK

CICN

odp4vpp

VPP Sandbox

VPP

Testing/Support

CSIT

puppet-fdio

trex

Network IO

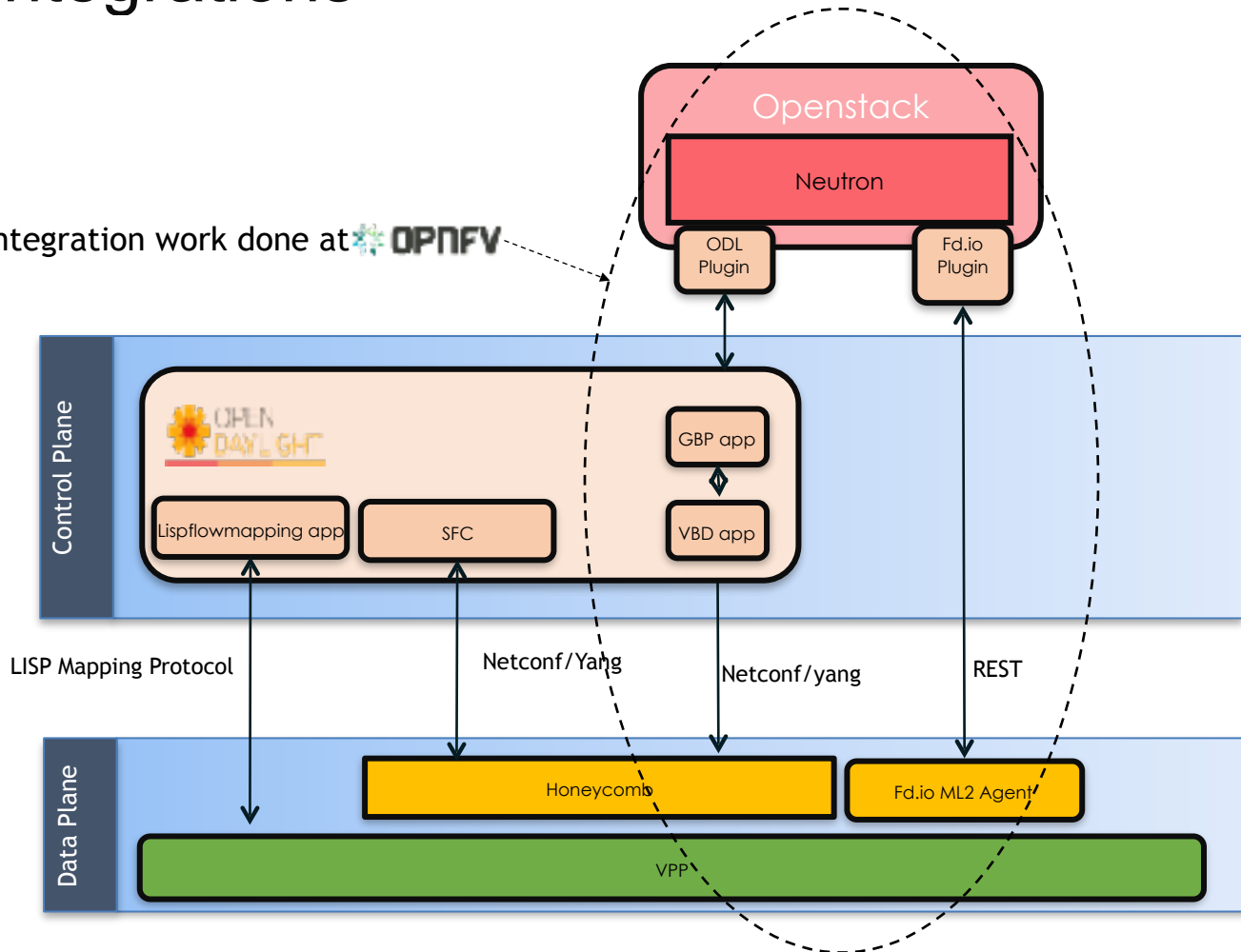
deb_dpdk

rpm_dpdk

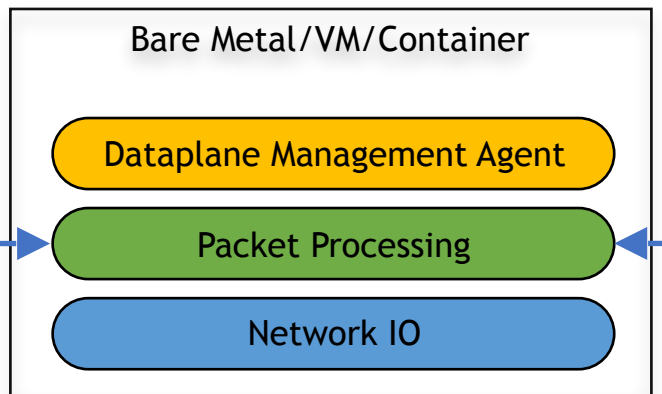
Fd.io Integrations






Integration work done at  **OPENNFV**



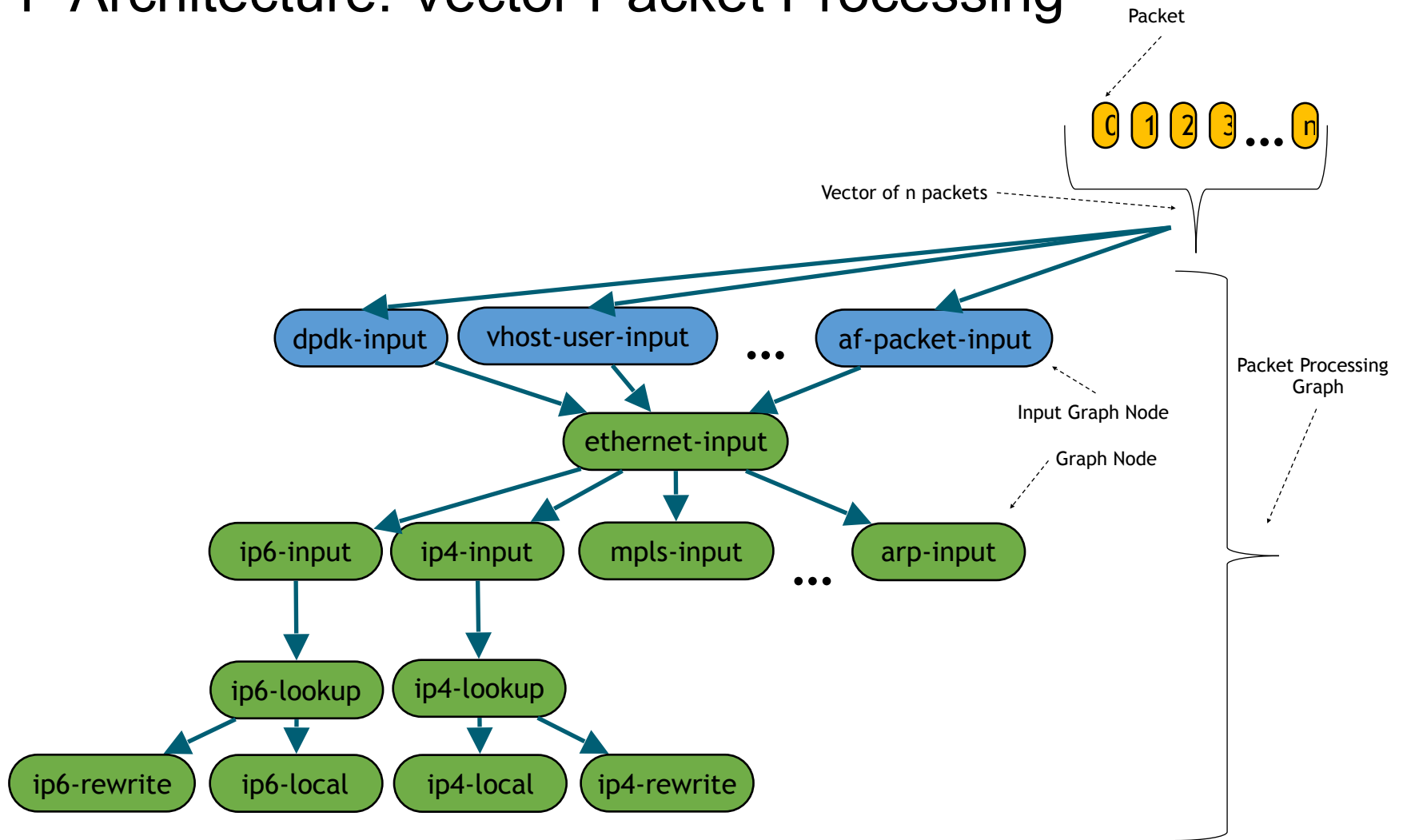
Vector Packet Processor - VPP



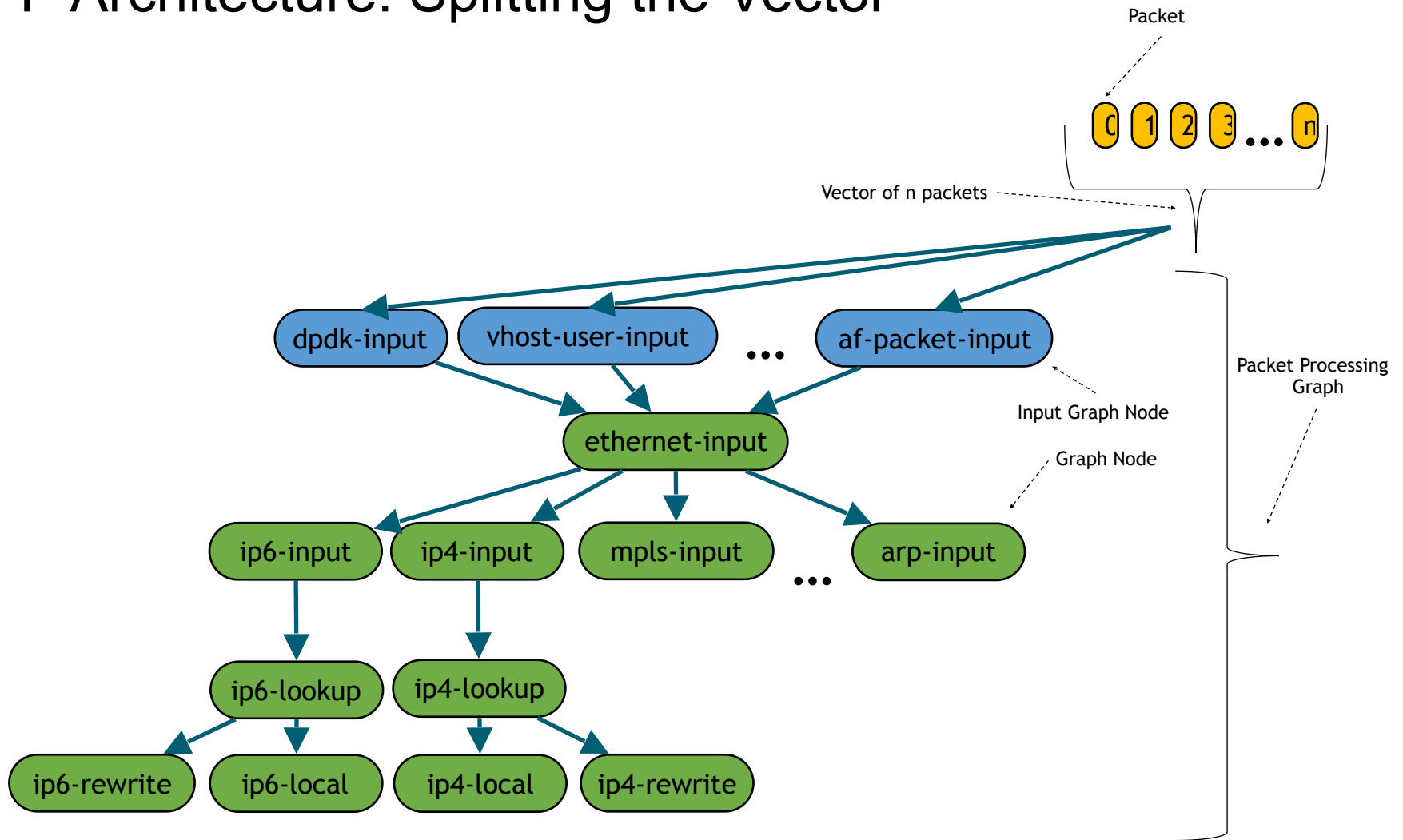
• Packet Processing Platform

- High performance
- Linux User space
- Run's on commodity CPUs:  /  / 
- Shipping at volume in server & embedded products since 2004.

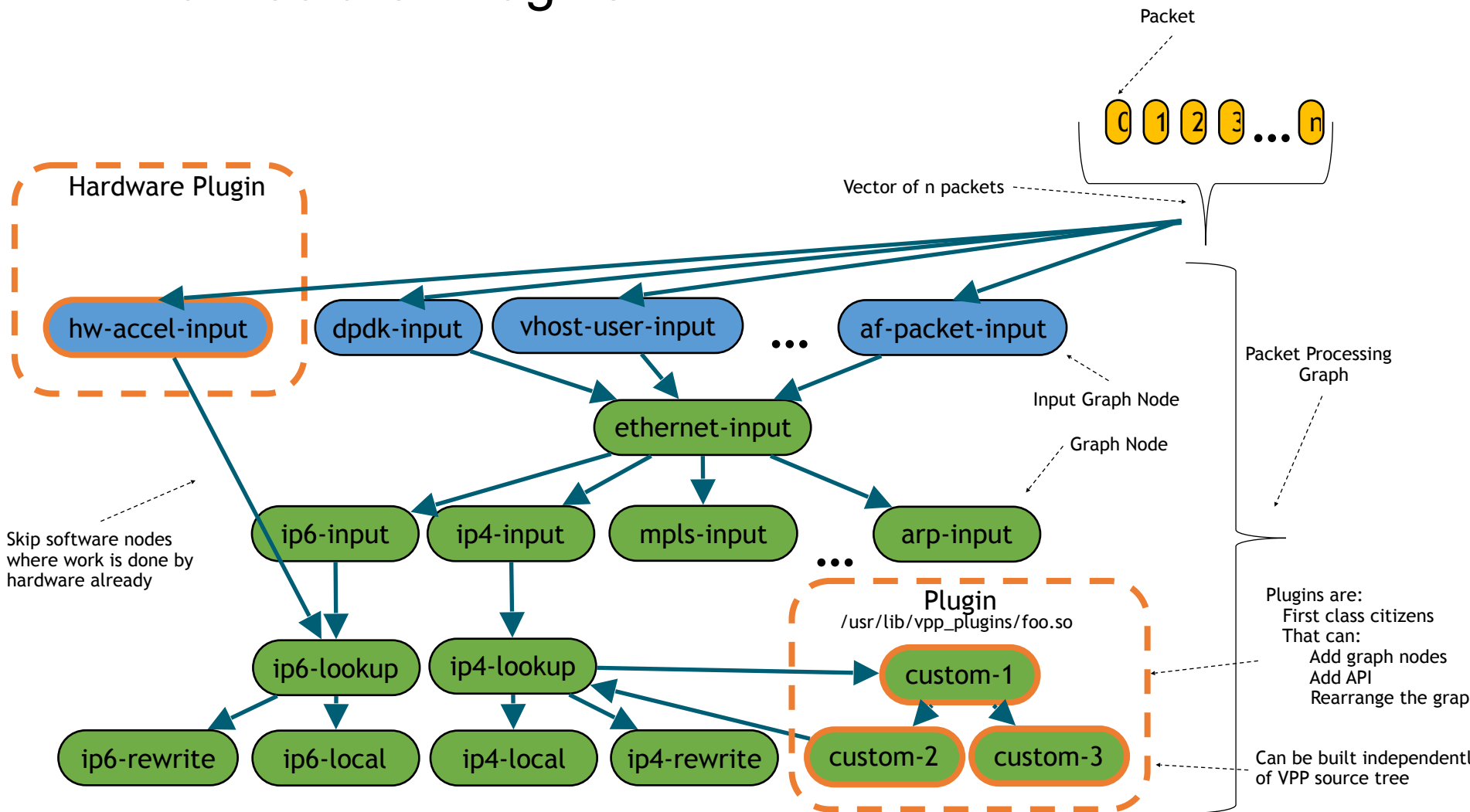
VPP Architecture: Vector Packet Processing



VPP Architecture: Splitting the Vector



VPP Architecture: Plugins

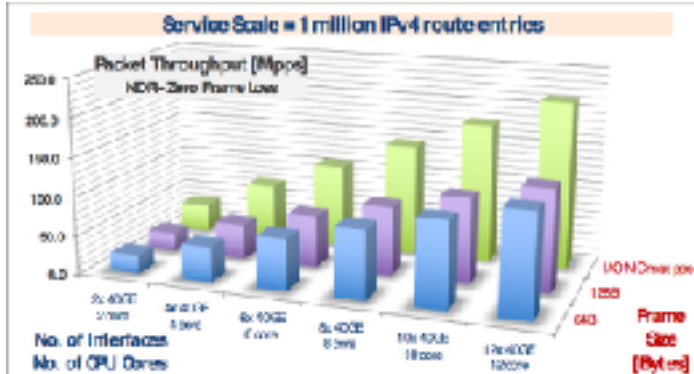


- Let's look at performance data at scale
- Packet throughput for
 - IPv4 routing,
 - IPv6 routing,
 - L2 switching,
 - L2 switching with VXLAN tunnelling.

VPP Universal Fast Dataplane: Performance at Scale [1/2]

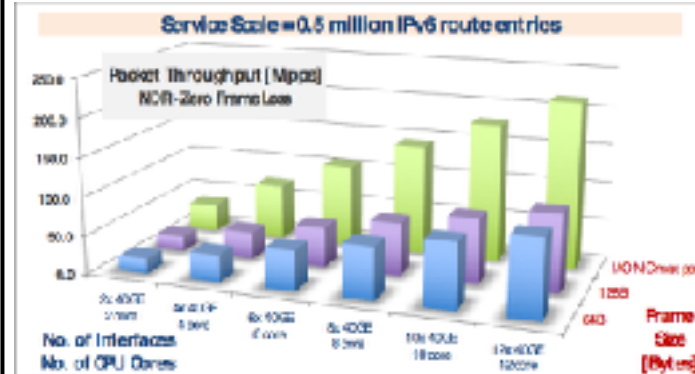
Per CPU core throughput with linear multi-thread(-core) scaling

IPv4 Routing



| IPv4 Thput [Mpps] | 2x 40GE 2 core | 4x 40GE 4 core | 6x 40GE 6 core | 8x 40GE 8 core | 10x 40GE 10 core | 12x 40GE 12 core |
|-------------------|-------------------|-------------------|-------------------|-------------------|---------------------|---------------------|
| 64B | 24.0 | 45.4 | 66.7 | 88.1 | 109.4 | 130.8 |
| 128B | 24.0 | 45.4 | 66.7 | 88.1 | 109.4 | 130.8 |
| IMIX | 15.0 | 30.0 | 45.0 | 60.0 | 75.0 | 90.0 |
| 1518B | 3.8 | 7.6 | 11.4 | 15.2 | 19.0 | 22.8 |
| I/O NICmax-pps | 35.8 | 71.6 | 107.4 | 143.2 | 179 | 214.8 |
| NICmax-bw | 46.8 | 93.5 | 140.3 | 187.0 | 233.8 | 280.5 |

IPv6 Routing



| IPv6 Thput [Mpps] | 2x 40GE 2 core | 4x 40GE 4 core | 6x 40GE 6 core | 8x 40GE 8 core | 10x 40GE 10 core | 12x 40GE 12 core |
|-------------------|-------------------|-------------------|-------------------|-------------------|---------------------|---------------------|
| 64B | 19.2 | 35.4 | 51.5 | 67.7 | 83.8 | 100.0 |
| 128B | 19.2 | 35.4 | 51.5 | 67.7 | 83.8 | 100.0 |
| IMIX | 15.0 | 30.0 | 45.0 | 60.0 | 75.0 | 90.0 |
| 1518B | 3.8 | 7.6 | 11.4 | 15.2 | 19.0 | 22.8 |
| I/O NICmax-pps | 35.8 | 71.6 | 107.4 | 143.2 | 179 | 214.8 |
| NICmax-bw | 46.8 | 93.5 | 140.3 | 187.0 | 233.8 | 280.5 |

Topology:
Phy-VS-Phy



Hardware:

Cisco UCS C240 M4

Intel® C610 series chipset

2 x Intel® Xeon® Processor E5-2698 v3 (16 cores, 2.3GHz, 40MB Cache)

2133 MHz, 256 GB Total

6 x 2p40GE Intel XL710=12x40GE

Software

Linux: Ubuntu 16.04.1 LTS

Kernel: ver. 4.4.0-45-generic

FD.io VPP: VPP v17.01-5~ge234726 (DPDK 16.11)

Resources

1 physical CPU core per 40GE port

Other CPU cores available for other services and other work

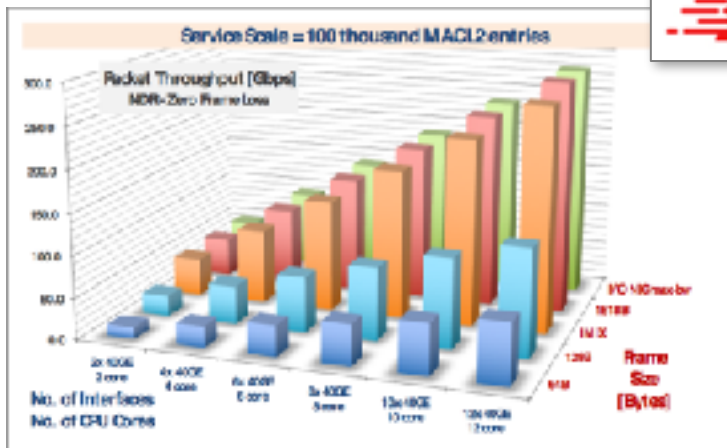
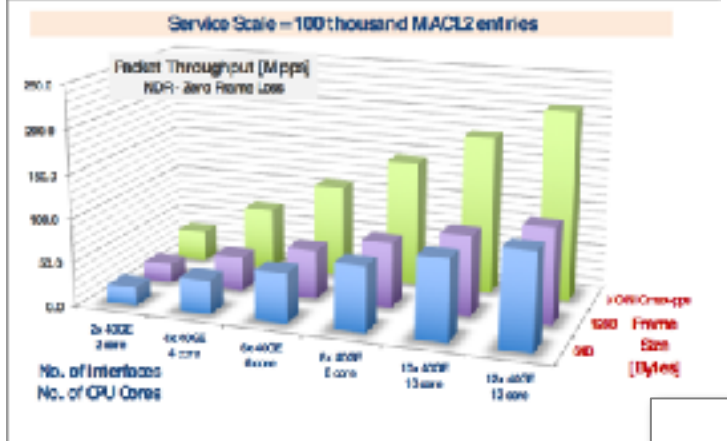
20 physical CPU cores available in 12x40GE seupt

Lots of Headroom for much more throughput and features

VPP Universal Fast Dataplane: Performance at Scale [2/2]

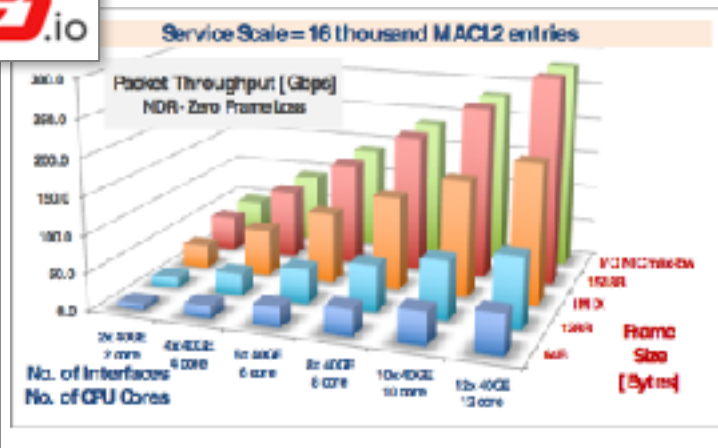
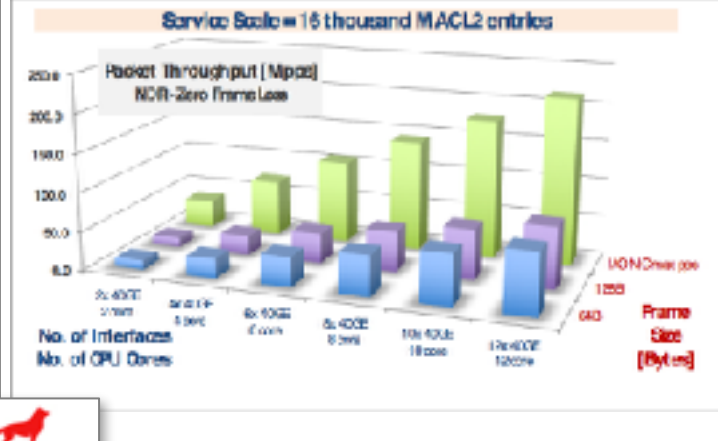
Per CPU core throughput with linear multi-thread(-core) scaling

L2 Switching



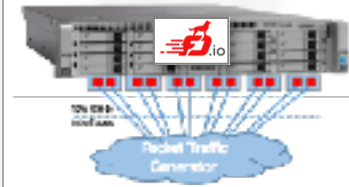
| MACThput [Mpps] | 2x 40GE 2 core | 4x 40GE 4 core | 6x 40GE 6 core | 8x 40GE 8 core | 10x 40GE 10 core | 12x 40GE 12 core |
|-----------------|-------------------|-------------------|-------------------|-------------------|---------------------|---------------------|
| 64B | 20.8 | 38.4 | 55.9 | 73.5 | 91.0 | 108.6 |
| 128B | 20.8 | 38.4 | 55.9 | 73.5 | 91.0 | 108.6 |
| IMIX | 15.0 | 30.0 | 45.0 | 60.0 | 75.0 | 90.0 |
| 1518B | 3.8 | 7.6 | 11.4 | 15.2 | 19.0 | 22.8 |
| I/O NICmax-pps | 35.8 | 71.6 | 107.4 | 143.2 | 179 | 214.8 |
| NICmax-bw | 46.8 | 93.5 | 140.3 | 187.0 | 233.8 | 280.5 |

L2 Switching with VXLAN Tunneling



| MACThput [Mpps] | 2x 40GE 2 core | 4x 40GE 4 core | 6x 40GE 6 core | 8x 40GE 8 core | 10x 40GE 10 core | 12x 40GE 12 core |
|-----------------|-------------------|-------------------|-------------------|-------------------|---------------------|---------------------|
| 64B | 11.6 | 25.1 | 38.6 | 52.2 | 65.7 | 79.2 |
| 128B | 11.6 | 25.1 | 38.6 | 52.2 | 65.7 | 79.2 |
| IMIX | 10.5 | 21.0 | 31.5 | 42.0 | 52.5 | 63.0 |
| 1518B | 3.8 | 7.6 | 11.4 | 15.2 | 19.0 | 22.8 |
| I/O NICmax-pps | 35.8 | 71.6 | 107.4 | 143.2 | 179 | 214.8 |
| NICmax-bw | 46.8 | 93.5 | 140.3 | 187.0 | 233.8 | 280.5 |

Topology:
Phy-VS-Phy



Hardware:

Cisco UCS C240 M4

Intel® C610 series chipset

2 x Intel® Xeon® Processor E5-2698 v3 (16 cores, 2.3GHz, 40MB Cache)

2133 MHz, 256 GB Total

6 x 2p40GE Intel XL710=12x40GE

Software

Linux: Ubuntu 16.04.1 LTS

Kernel: ver. 4.4.0-45-generic

FD.io VPP: VPP v17.01-5~ge234726 (DPDK 16.11)

Resources

1 physical CPU core per 40GE port

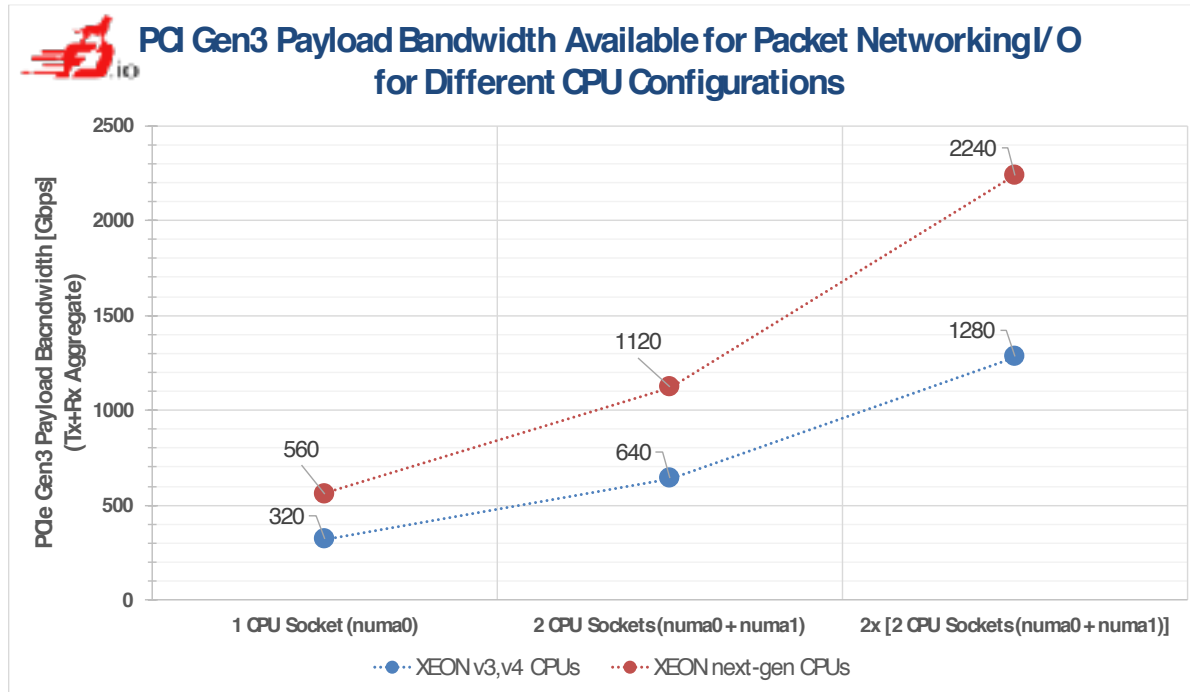
Other CPU cores available for other services and other work

20 physical CPU cores available in 12x40GE seupt

Lots of Headroom for much more throughput and features

Scaling Up The Packet Throughput with FD.io VPP

Can we squeeze more from a single 2RU server ?



1. Today's Intel® XEON® CPUs (E5 v3/ v4):

- a. Per socket have 40 lanes of PCIe Gen3
- b. 2x 160Gbps of packet I/O per socket



2. Tomorrow's Intel® XEON® CPUs:

- a. Per socket support More lanes of PCIe Gen3
- b. 2x 280Gbps of packet I/O per socket

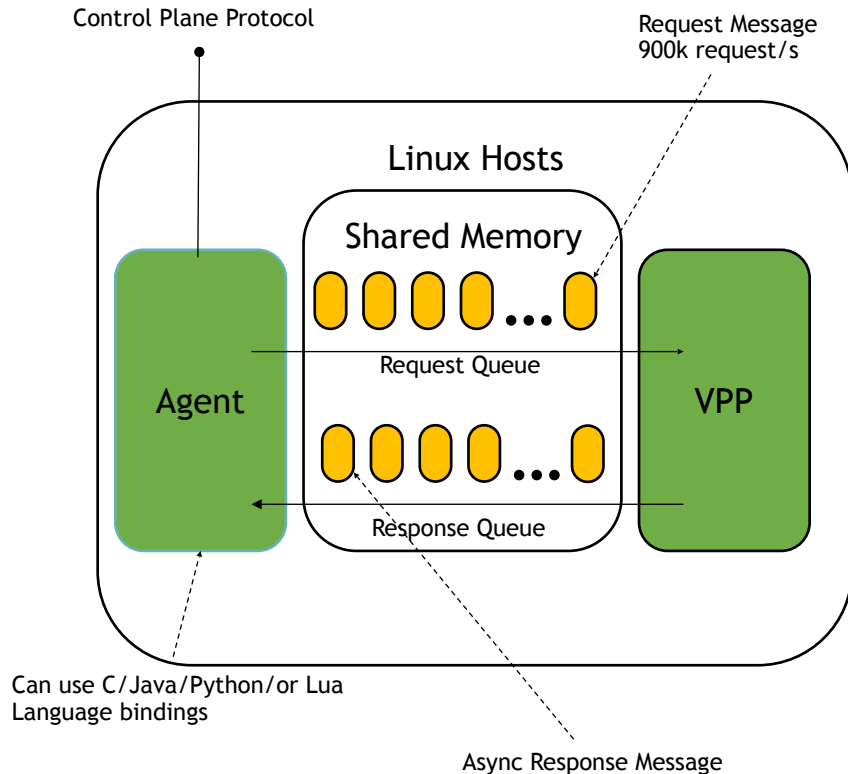
VPP enables linear multi-thread(-core) scaling
up to the packet I/O limit per CPU => on a path to **one terabit software router (1TFR)**.



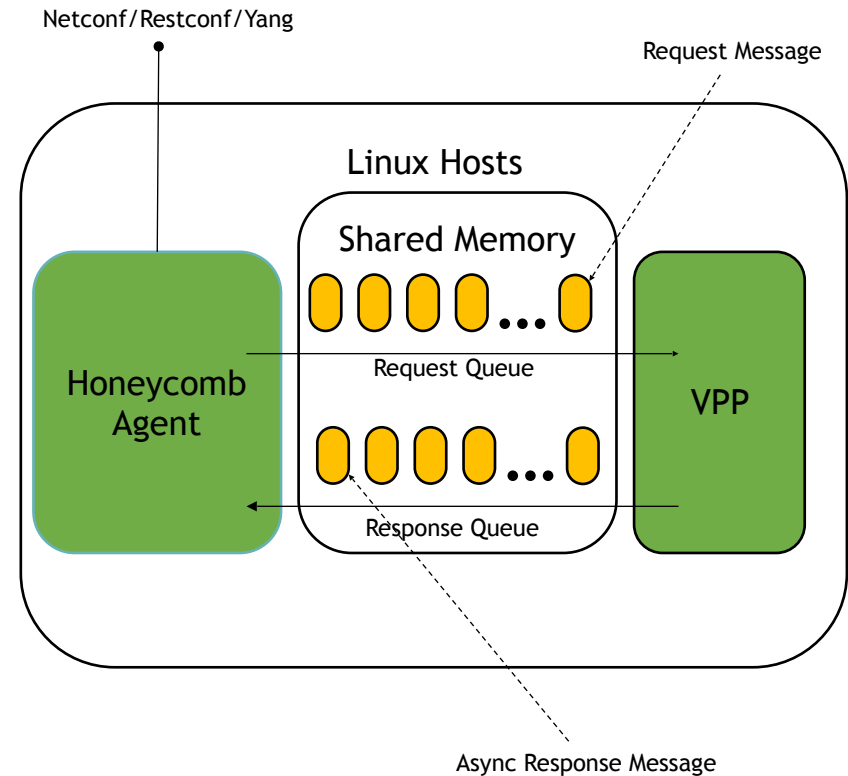
Breaking the Barrier of Software Defined Network Services
1 Terabit Services on a Single Intel® Xeon® Server !!!

VPP Architecture: Programmability

Architecture



Example: Honeycomb



Universal Fast Dataplane: Features



Hardware Platforms

Pure Userspace - X86, ARM 32/64, PowerPC
Raspberry Pi

Interfaces

DPDK/Netmap/AF_Packet/TunTap
Vhost-user - multi-queue, reconnect,
Jumbo Frame Support

Language Bindings

C/Java/Python/Lua

Tunnels/Encaps

GRE/VXLAN/VXLAN-GPE/LISP-GPE
NSH, L2TPv3, SRv6
IPSec Tunnel, Transport
Including HW offload if present

MPLS

MPLS over Ethernet/GRE
Deep label stacks supported

Routing

IPv4/IPv6
14+ MPPS, single core
Hierarchical FIBs
Multimillion FIB entries
Source RPF
Thousands of VRFs
Controlled cross-VRF lookups
Multipath - ECMP and Unequal Cost
IP Multicast

Segment Routing

SR MPLS/IPv6
Including Multicast

LISP

LISP xTR/RTR
L2 Overlays over LISP and GRE encaps
Multitenancy
Multihome
Map/Resolver Failover
Source/Dest control plane support
Map-Register/Map-Notify/RLOC-probing
IPSec transport mode

Switching

VLAN Support
Single/ Double tag
L2 forwrd w/EFP/BridgeDomain concepts
VTR - push/pop/Translate (1:1,1:2, 2:1,2:2)
Mac Learning - default limit of 50k addr
Bridging
Split-horizon group support/EFP Filtering
Proxy Arp
Arp termination
IRB - BVI Support with RouterMac assignmt
Flooding
Input ACLs
Interface cross-connect
L2 GRE over IPSec tunnels

Security

Mandatory Input Checks:
TTL expiration
header checksum
L2 length < IP length
ARP resolution/snooping
ARP proxy
NAT
Ingress Port Range Filtering
Per interface whitelists
Policy/Security Groups/GBP (Classifier)

Network Services

DHCPv4 client/proxy
DHCPv6 Proxy
MAP/LW46 - IPv4aaS
CGNAT
MagLev-like Load Balancer
Identifier Locator Addressing
NSH SFC SFF's & NSH Proxy
LLDP
BFD
QoS Policer 1R2C, 2R3C
Multiple million Classifiers -
Arbitrary N-tuple

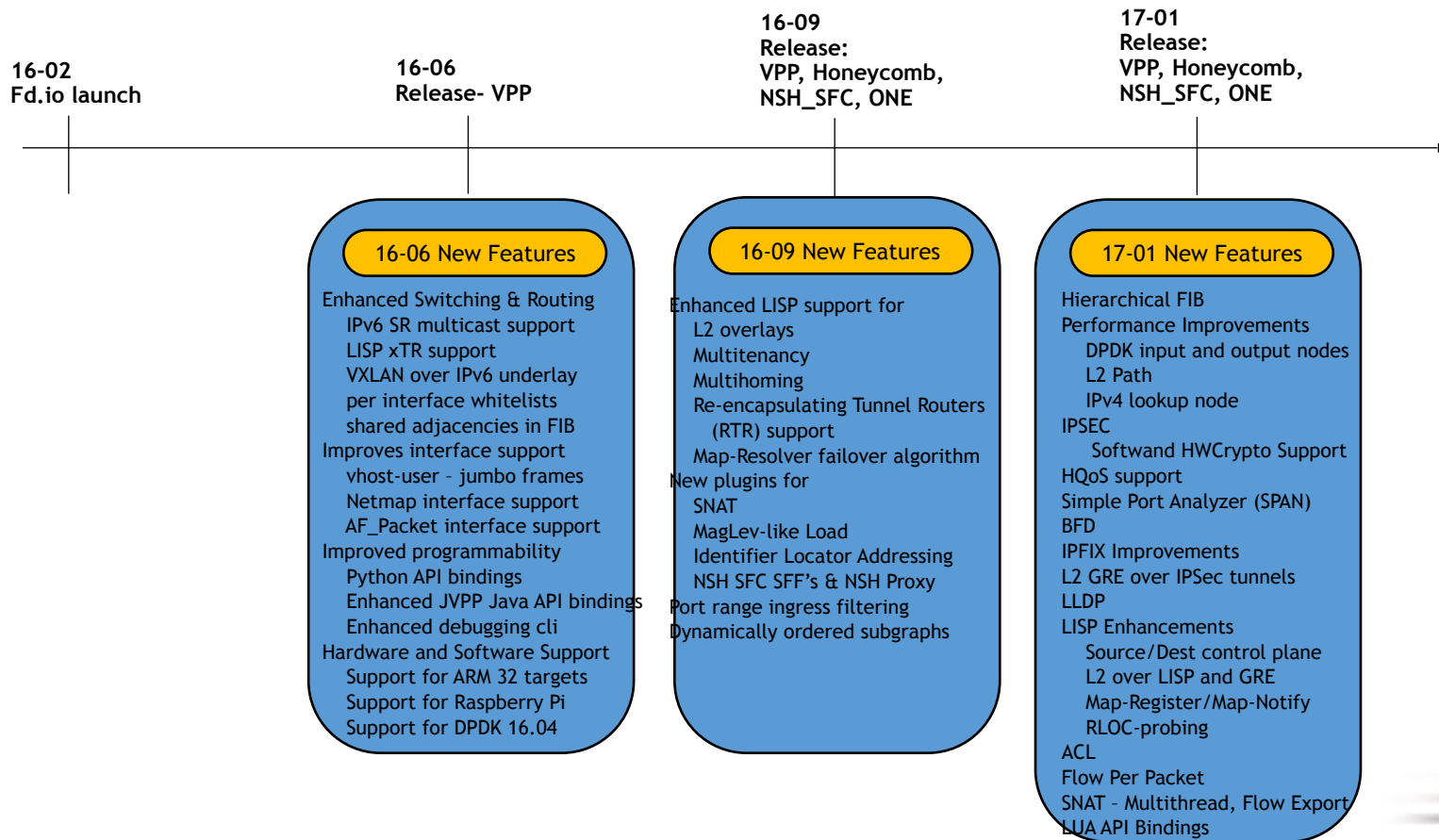
Inband iOAM

Telemetry export infra (raw IPFIX)
iOAM for VXLAN-GPE (NGENA)
SRv6 and iOAM co-existence
iOAM proxy mode / caching
iOAM probe and responder

Monitoring

Simple Port Analyzer (SPAN)
IP Flow Export (IPFIX)
Counters for everything
Lawful Intercept

Rapid Release Cadence – ~3 months



New in 17.04 – Released Apr 19

VPP Userspace Host Stack

- TCP stack
- DHCPv4 relay multi-destination
- DHCPv4 option 82
- DHCPv6 relay multi-destination
- DHCPv6 relay remote-id
- ND Proxy

NAT

- CGN: Configurable port allocation
- CGN: Configurable Address pooling
- CPE: External interface
- DHCP support
- NAT44, NAT64, LW46

Stateful Security Groups

- Routed interface support
- L4 filters with IPv6 Extension Headers

API

- Move to CFFI for Python binding
- Python Packaging improvements
- CLI over API
- Improved C/C++ language binding

Segment Routing v6

- SR policies with weighted SID lists
- Binding SID
- SR steering policies
- SR LocalSIDs
- Framework to expand local SIDs w/plugins

iOAM

- UDP Pinger w/path fault isolation
- IOAM as type 2 metadata in NSH
- IOAM raw IPFIX collector and analyzer
- Anycast active server selection

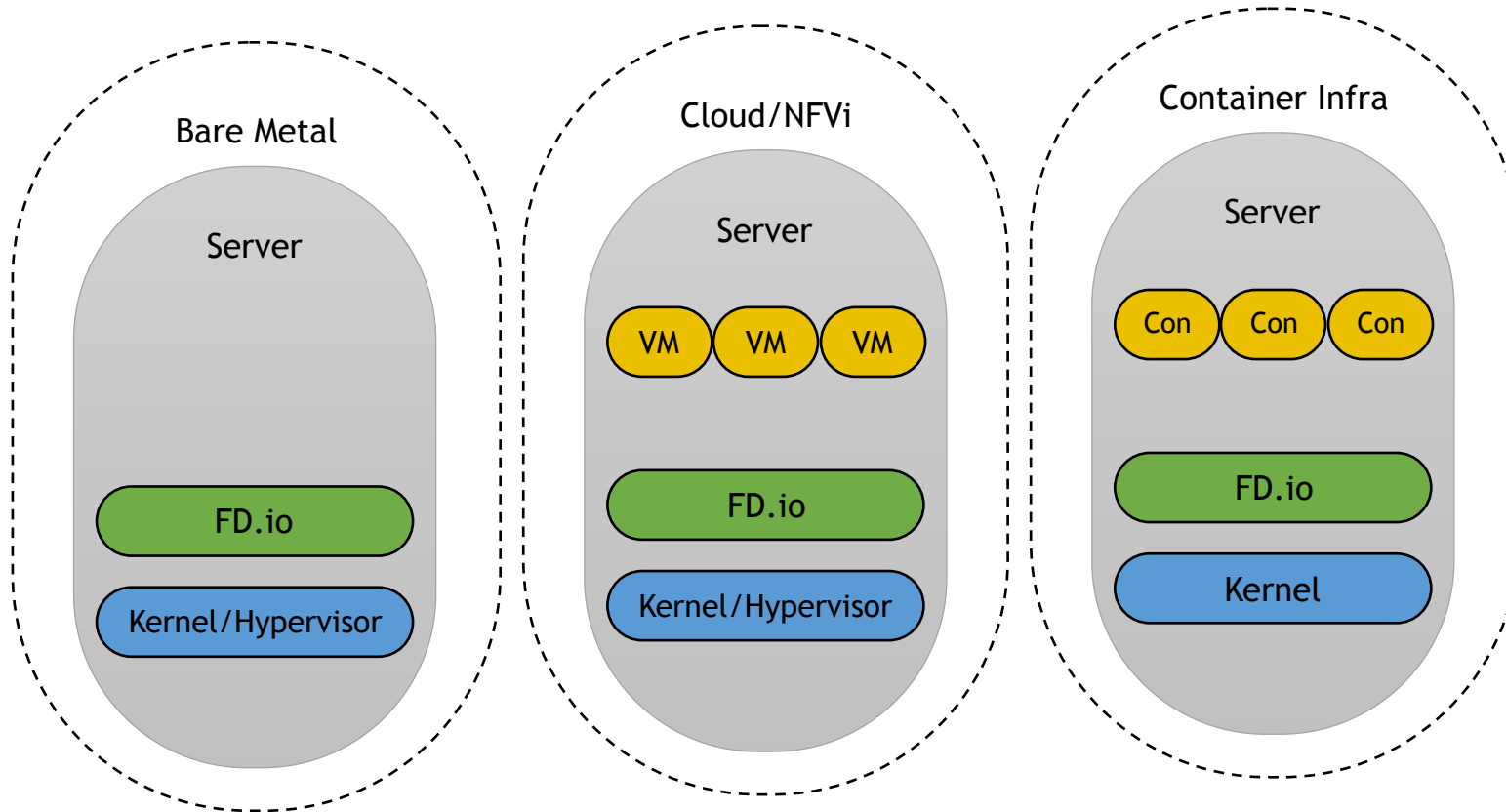
IPFIX

- Collect IPv6 information
- Per flow state

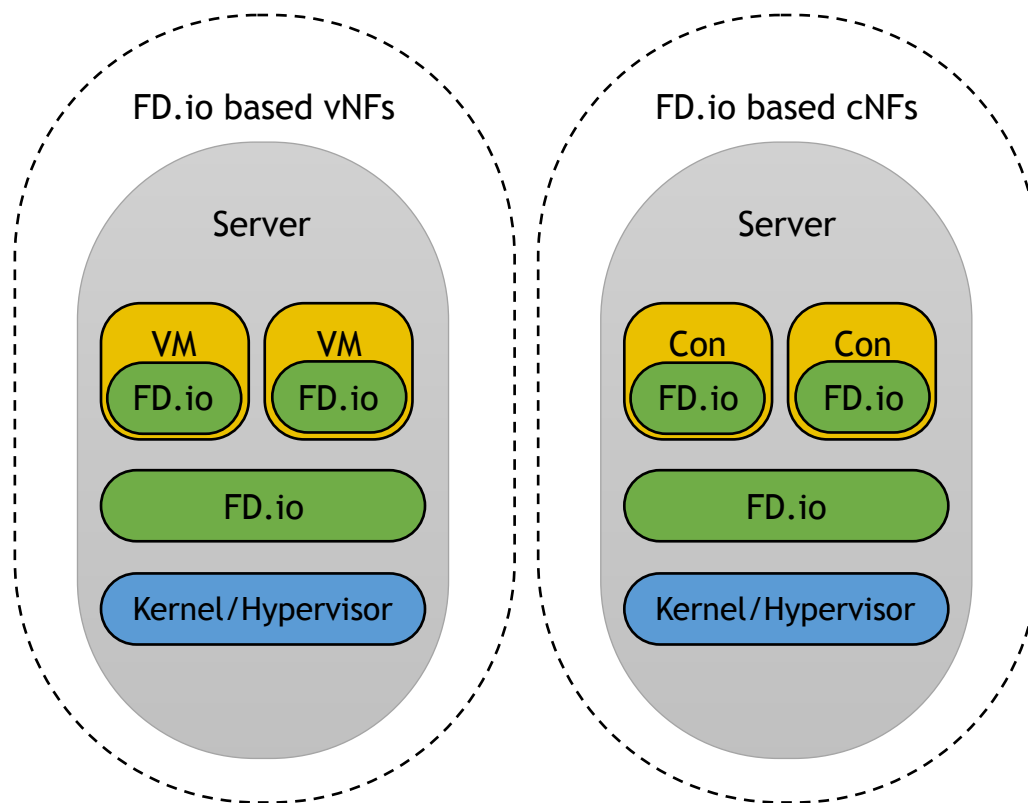
Release notes: https://docs.fd.io/vpp/17.04/release_notes_1704.html

Images at: <https://nexus.fd.io/>

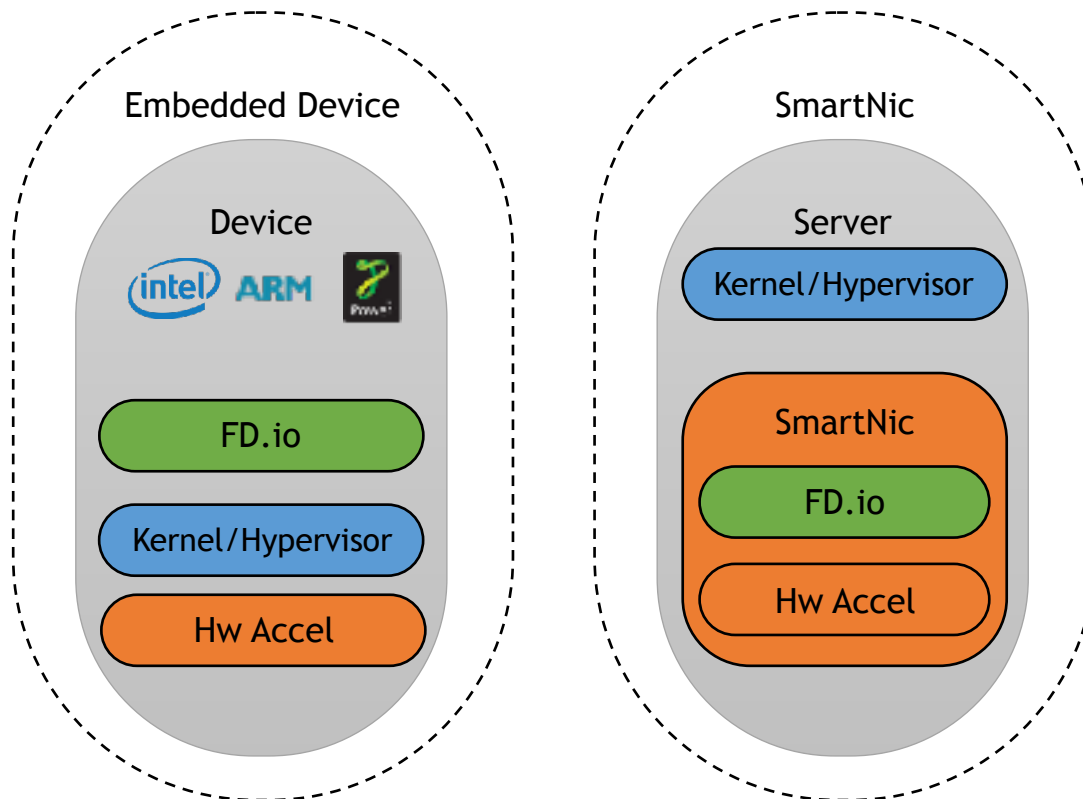
Universal Fast Dataplane: Infrastructure



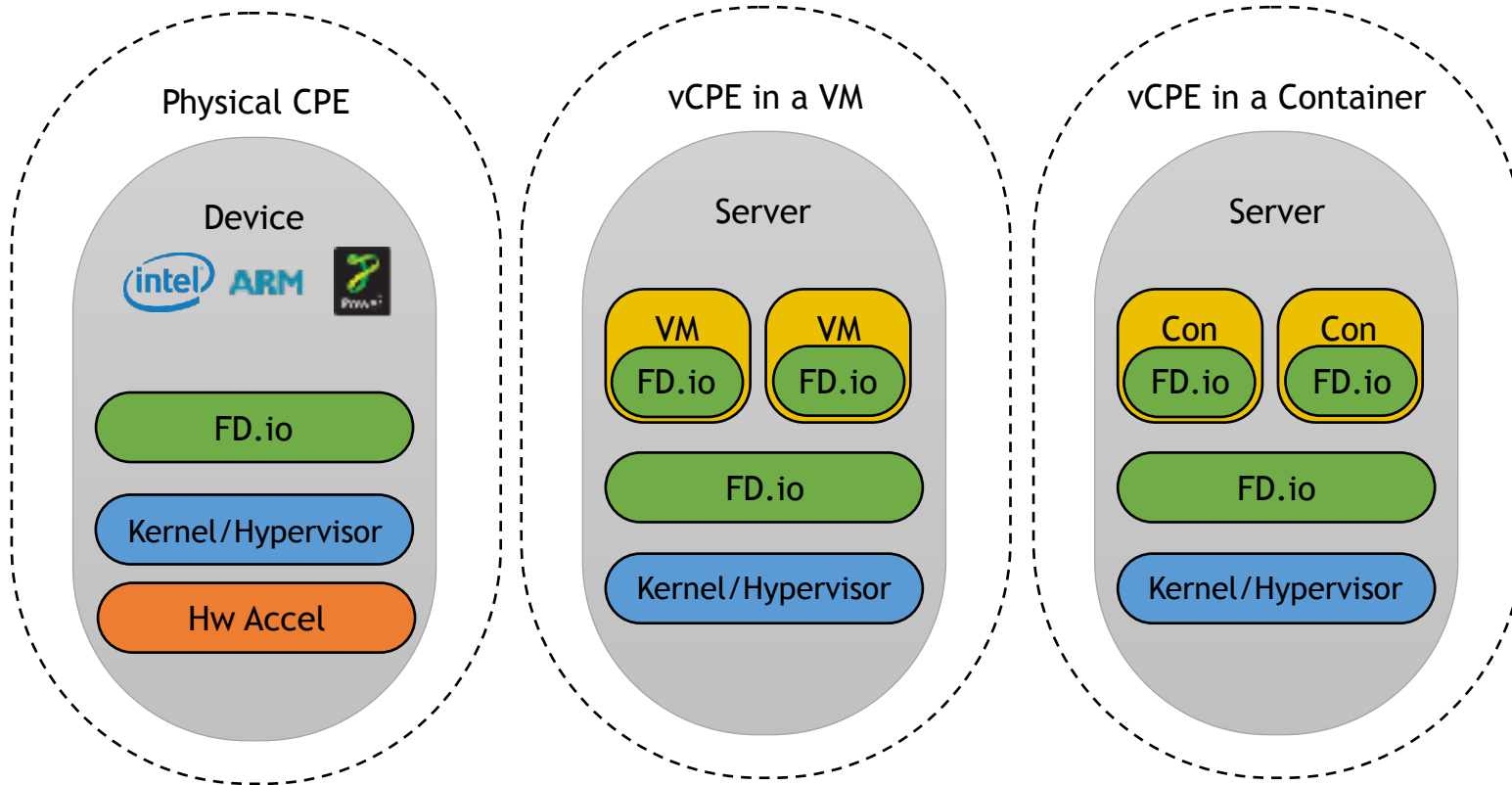
Universal Fast Dataplane: xNFs



Universal Fast Dataplane: Embedded



Universal Fast Dataplane: CPE Example



Opportunities to Contribute

- Firewall
- IDS
- DPI
- Flow and user telemetry
- Hardware Accelerators
- Container Integration
- Integration with OpenCache
- Control plane - support your favorite SDN Protocol Agent
- Test tools
- Cloud Foundry Integration
- Packaging
- Testing

We invite you to Participate in fd.io

- Get the Code, Build the Code, Run the Code
- [Try the vpp user demo](#)
- [Install vpp from binary packages \(yum/apt\)](#)
- [Install Honeycomb from binary packages](#)
- [Read/Watch the Tutorials](#)
- [Join the Mailing Lists](#)
- [Join the IRC Channels](#)
- [Explore the wiki](#)
- [Join fd.io as a member](#)

FD.io git repos:

<https://git.fd.io/>
<https://git.fd.io/vpp/>
<https://git.fd.io/csit/>

FD.io project wiki pages:

https://wiki.fd.io/view/Main_Page
<https://wiki.fd.io/view/VPP>
<https://wiki.fd.io/view/CSIT>

Q&A



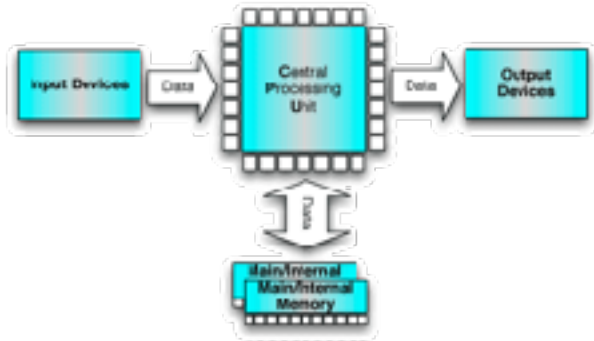
Aside [1/4]: Computer Evolution For Packet Processing

It started with ALU ...

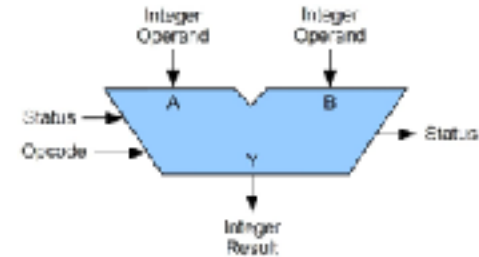
(2) Became universal and faster ...

(1) It started simple ...

A simple generic purpose computer



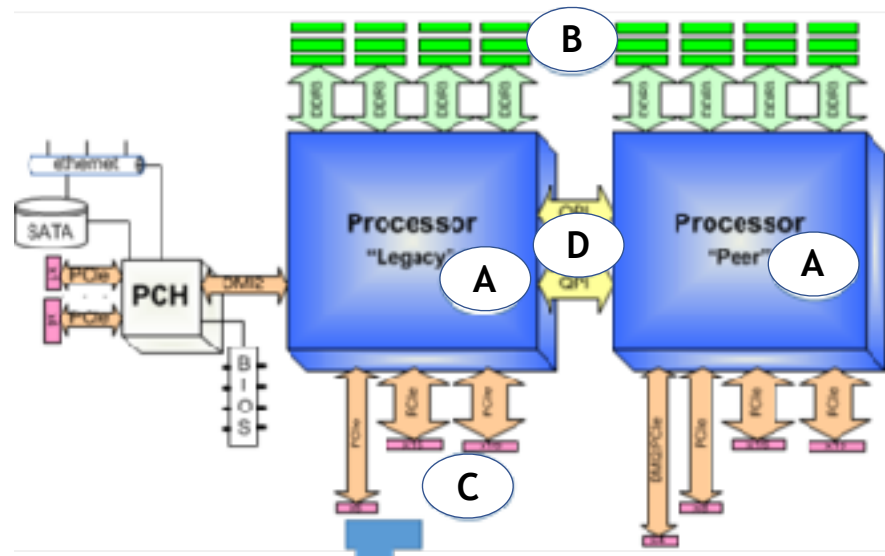
An Arithmetic Logic Unit (ALU)



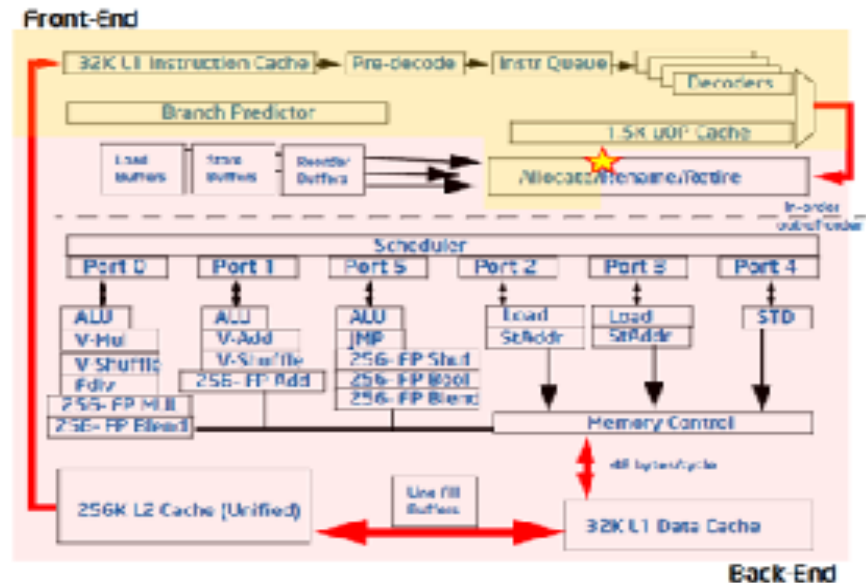
(3) Then it got much faster ...

(4) ... but far from simple !

A modern two CPU socket server



The pipeline of a modern high-performance XEON CPU*

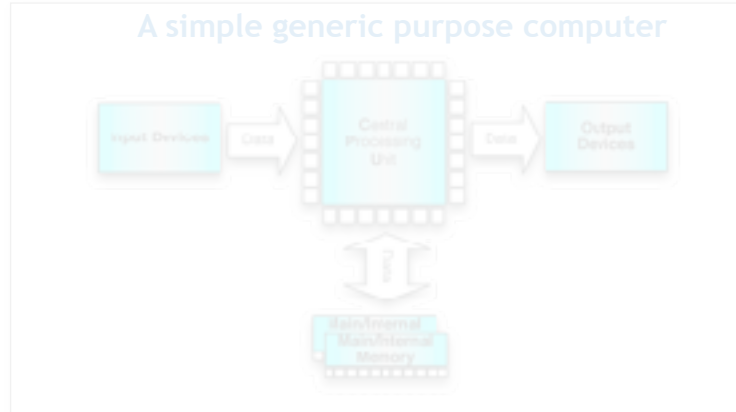


* Intel Top-Down Microarchitecture Analysis Method for tuning applications, <https://software.intel.com/en-us/top-down-microarchitecture-analysis-methods>

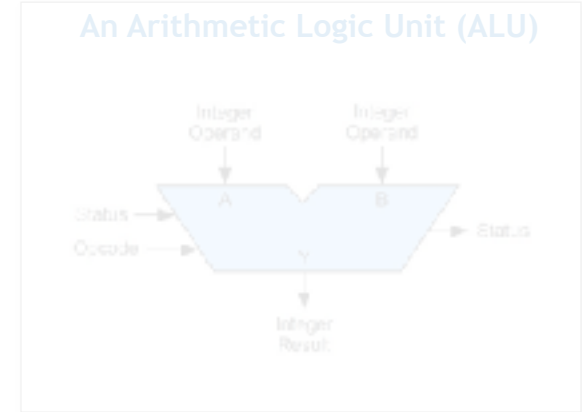
Aside [2/4]: Computer Evolution For Packet Processing

... and we arrived to modern multi-socket COTS server ...

(2) Became universal and faster ...

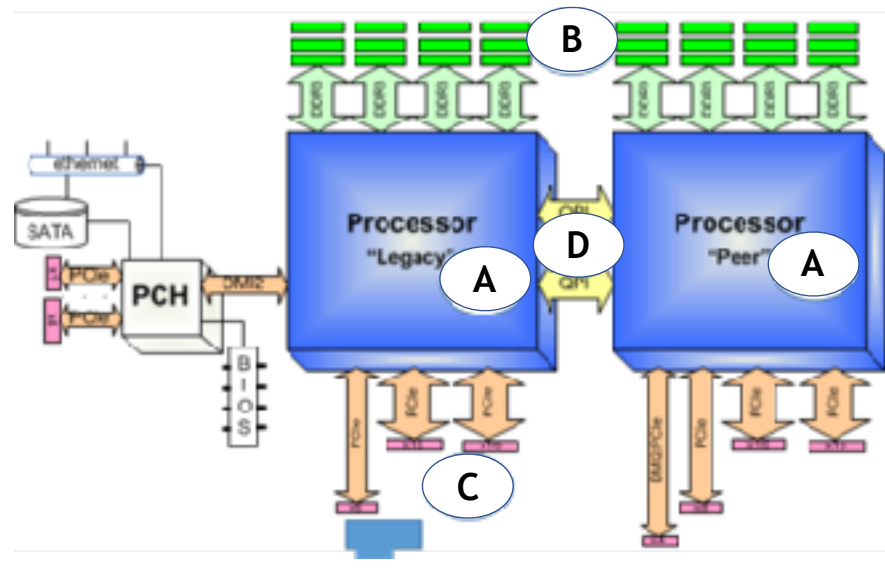


(1) It started simple ...



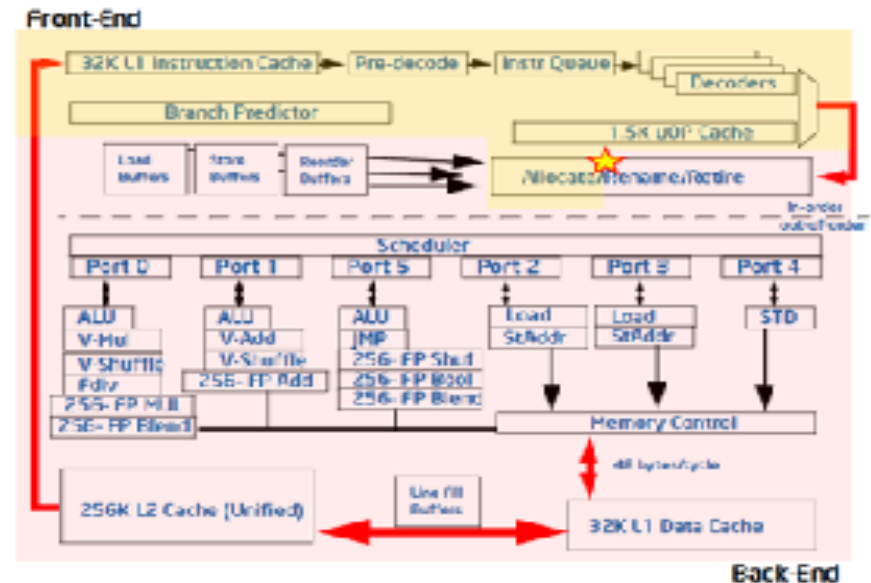
(3) Then it got much faster ...

A modern two CPU socket server



(4) ... but far from simple !

The pipeline of a modern high-performance XEON CPU*



* Intel Top-Down Microarchitecture Analysis Method for tuning applications, <https://software.intel.com/en-us/top-down-microarchitecture-analysis-method-overview>

Aside [2/4]: Computer Evolution For Packet Processing

... and we arrived to modern multi-socket COTS server ...

Four main functional dimensions important for processing packets:

A. CPUs executing the program(s):

- a) **Minimize Instructions per Packet** - Efficient software logic to perform needed packet operations.
- b) **Maximize Instructions per CPU core clock cycle** - Execution efficiency of an underlying CPU micro-architecture.

B. Memory bandwidth: **Minimize memory bandwidth utilization** - Memory access is slow.

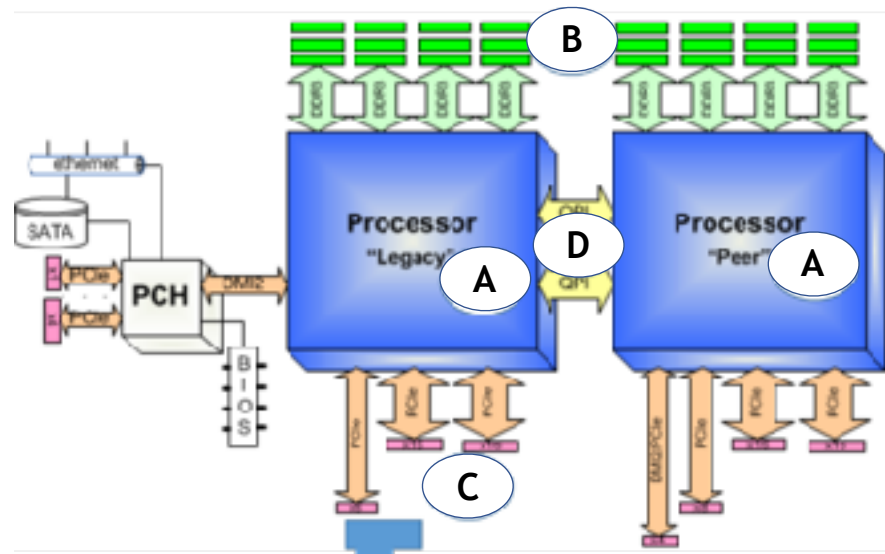
C. Network I/O bandwidth: **Make efficient use of PCIe I/O bandwidth** - It is a limited resource.

D. Inter-socket transactions: **Minimize cross-NUMA connection utilization** - It slows things down

Hint: Start with optimizing the use of CPU micro-architecture => **Use vectors !**

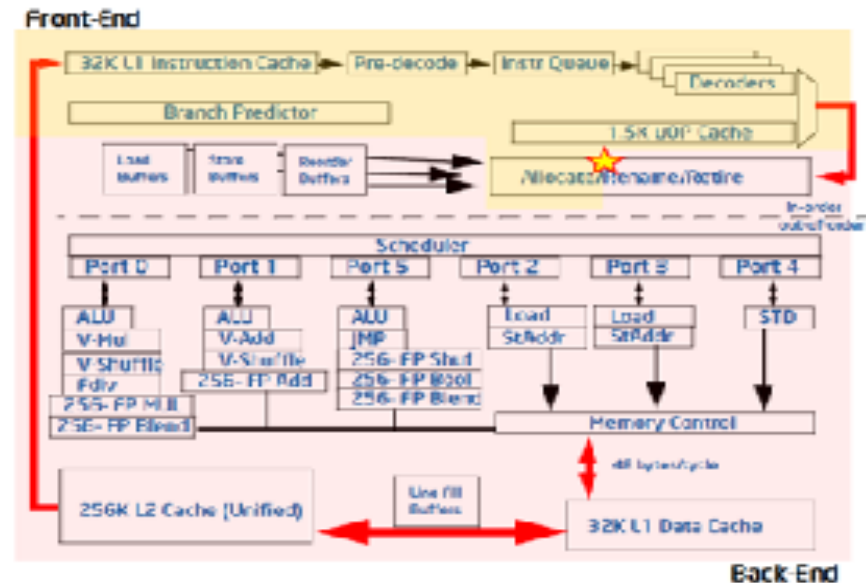
(3) Then it got much faster ...

A modern two CPU socket server



(4) ... but far from simple !

The pipeline of a modern high-performance XEON CPU*

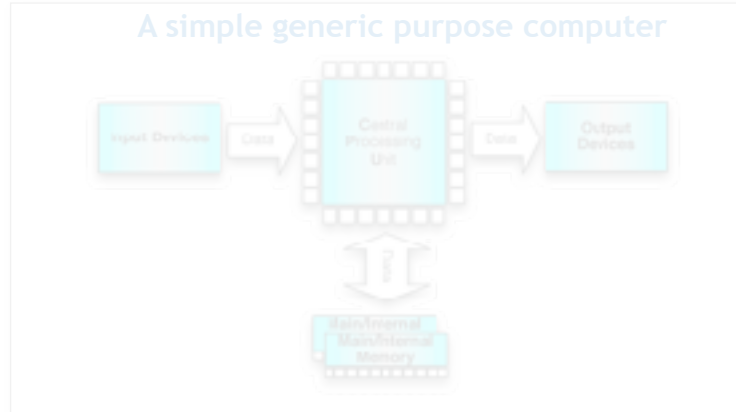


* Intel Top-Down Microarchitecture Analysis Method for tuning applications, <https://software.intel.com/en-us/top-down-microarchitecture-analysis-methods>

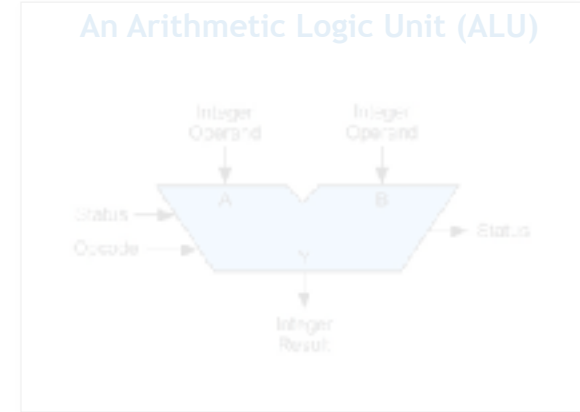
Aside [3/4]: Computer Evolution For Packet Processing

... we then optimize software for network workloads ...

(2) Became universal and faster ...

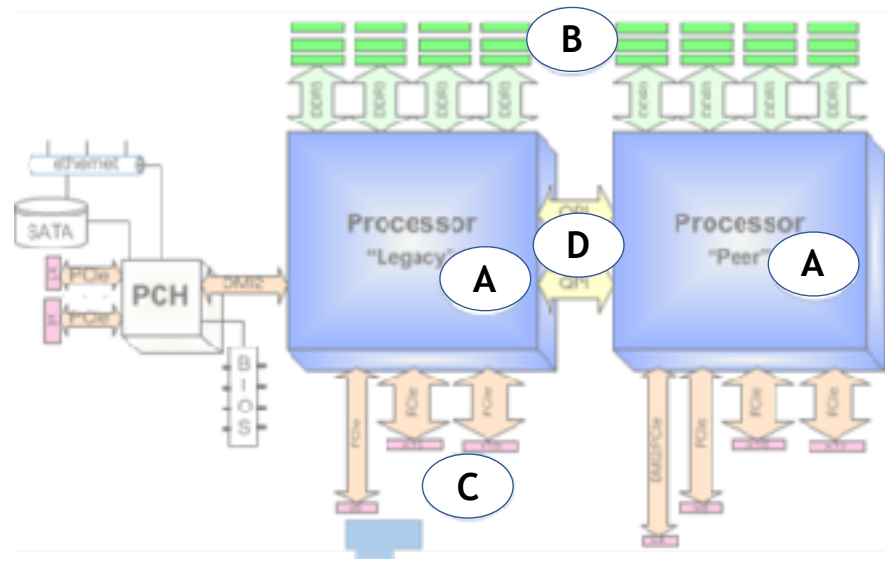


(1) It started simple ...



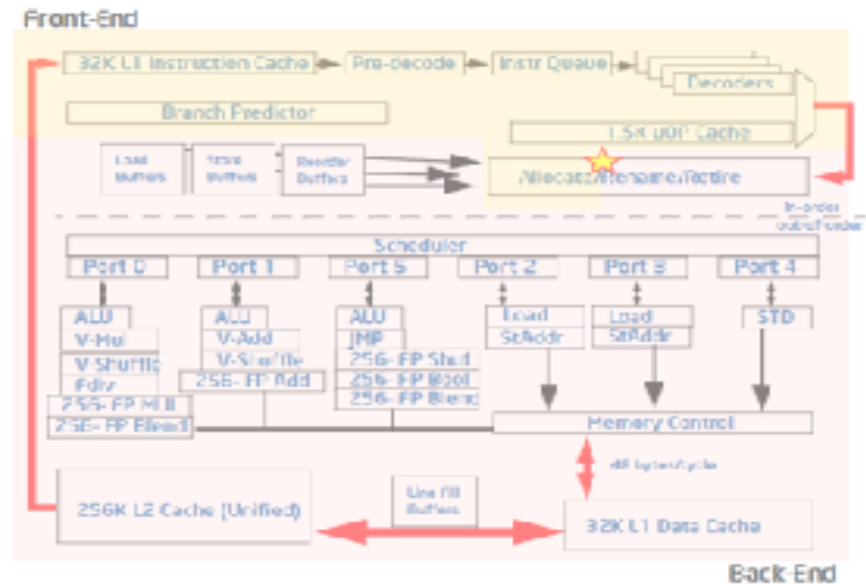
(3) Then it got much faster ...

A modern two CPU socket server



(4) ... but far from simple !

The pipeline of a modern high-performance XEON CPU*



* Intel Top-Down Microarchitecture Analysis Method for tuning applications, <https://software.intel.com/en-us/top-down-microarchitecture-analysis-methods>

... we then optimize software for network workloads ...

- ## (1) It started simple ...

- **Packet processing efficiency is essential**

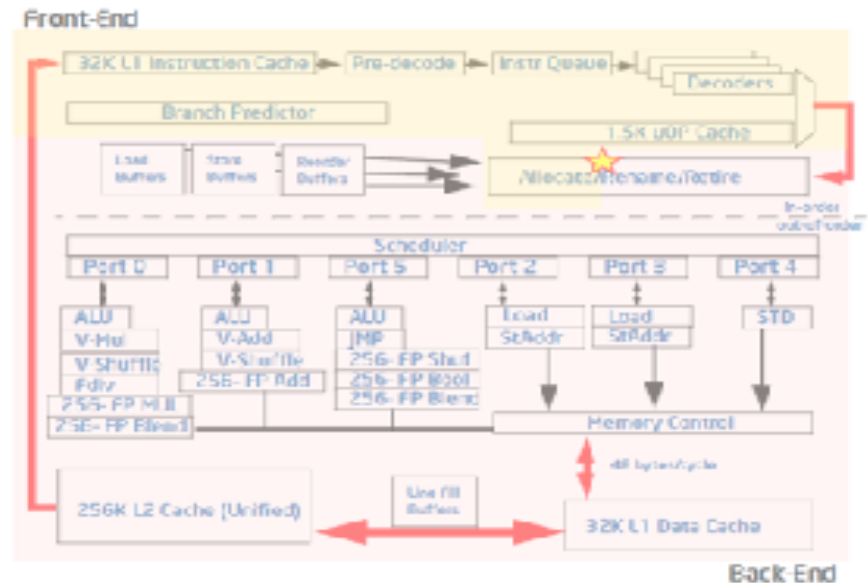
- **Moving packets**

- **Processing packets**

- Need packet processing optimized for CPU platforms.
- Header manipulation, encaps/decaps, lookups, classifiers, counters.

(4) ... but far from simple !

The pipeline of a modern high-performance XEON CPU



Aside [3/4]: Computer Evolution For Packet Processing

... we then optimize software for network workloads ...

(2) Became universal and faster ...

- Network workloads are **very different** from compute ones

- They are all about processing packets, at rate.
- At 10GE, 64B packets can arrive at 14.88Mpps
=> **67 nsec per packet.**
- With 2GHz CPU core clock cycle is 0.5nsec
=> **134 clock cycles per packet.**
- To access memory it takes ~70nsec
=> **too slow to do it per packet !**

- **Packet processing efficiency** is essential

- **Moving packets**

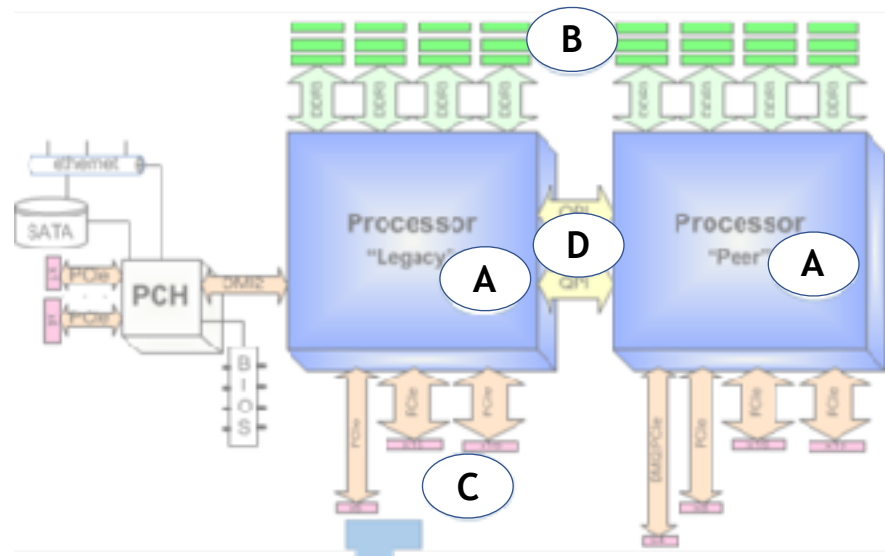
- Packets arrive on physical interfaces (NICs) and virtual interfaces (VNFs) - need **CPU optimized drivers** for both.
- Drivers and buffer management software must not rely on memory access – see time budget above, **MUST use CPU core caching hierarchy** well.

- **Processing packets**

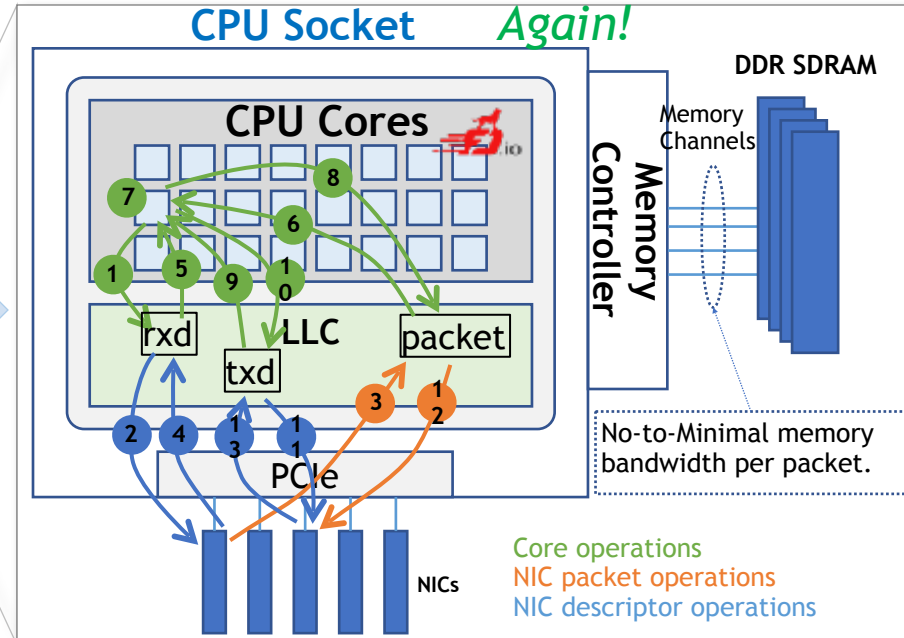
- Need **packet processing optimized for CPU platforms.**
- Header manipulation, encaps/decaps, lookups, classifiers, counters.

(3) Then it got much faster ...

A modern two CPU socket server



AND => (4a) ... let's make it Simple Again!



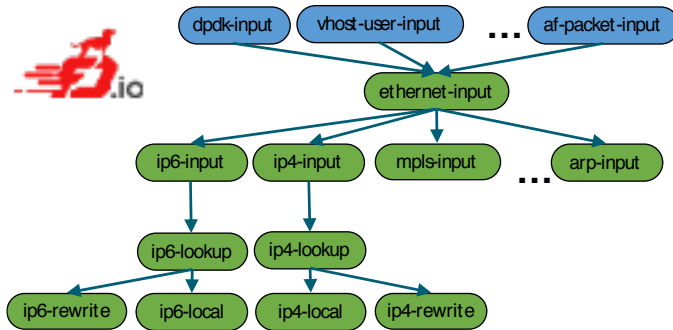
Aside [4/4]: Computer Evolution For Packet Processing

... and use *FD.io VPP* to make them fast for packets.

- (1) Core writes Rx descriptor in preparation for receiving a packet.
- (2) NIC reads Rx descriptor to get ctrl flags and buffer address.
- (3) NIC writes the packet.
- (4) NIC writes Rx descriptor.
- (5) Core reads Rx descriptor (polling or irq or coalesced irq).
- (6) Core reads packet header to determine action.
- (7) Core performs action on packet header.
- (8) Core writes packet header (MAC swap, TTL, tunnel, foobar..)
- (9) Core reads Tx descriptor.
- (10) Core writes Tx descriptor and writes Tx tail pointer.
- (11) NIC reads Tx descriptor.
- (12) NIC reads the packet.
- (13) NIC writes Tx descriptor.

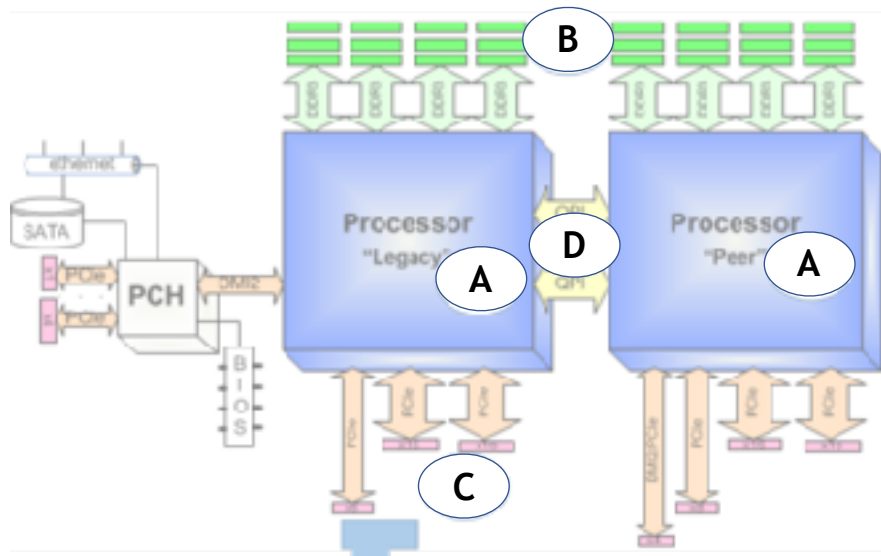
AND => (4a) ... let's make it Simple Again!

Vector Packet Processing software worker thread



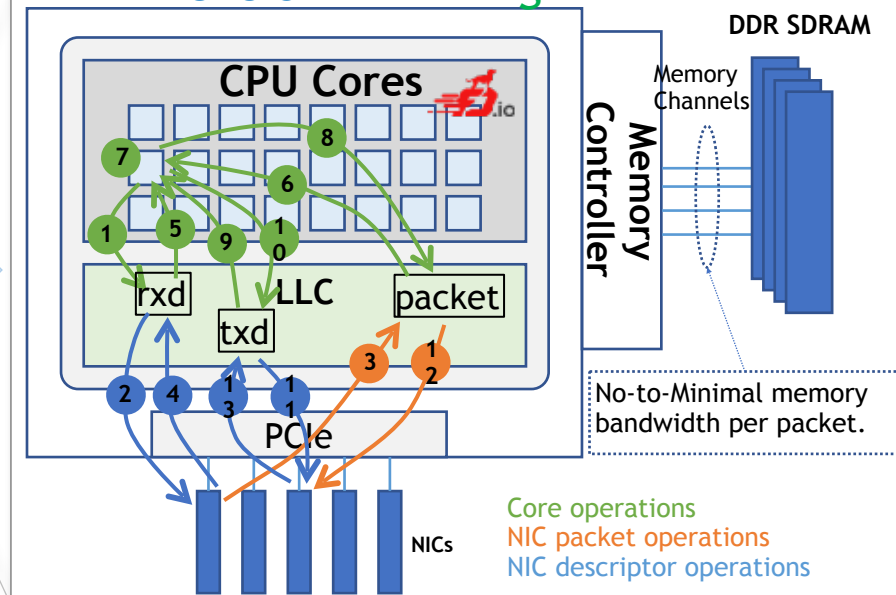
(3) Then it got much faster ...

A modern two CPU socket server



CPU Socket

Again!



Aside [4/4]: Computer Evolution For Packet Processing

... and use *FD.io VPP* to make them fast for packets.

Steps 1-to-13 in a Nutshell:

- ✓ VPP software worker threads run on CPU cores.
- ✓ Use local caching with No-to-Minimal memory bandwidth per packet.
- ✓ Get speed with predictive prefetching and smart algos.
- ✓ And make CPU cache hierarchy always “Hot” => Packet processing at rate.

Making VPP simply tick the A-B-C-D server optimization points!

