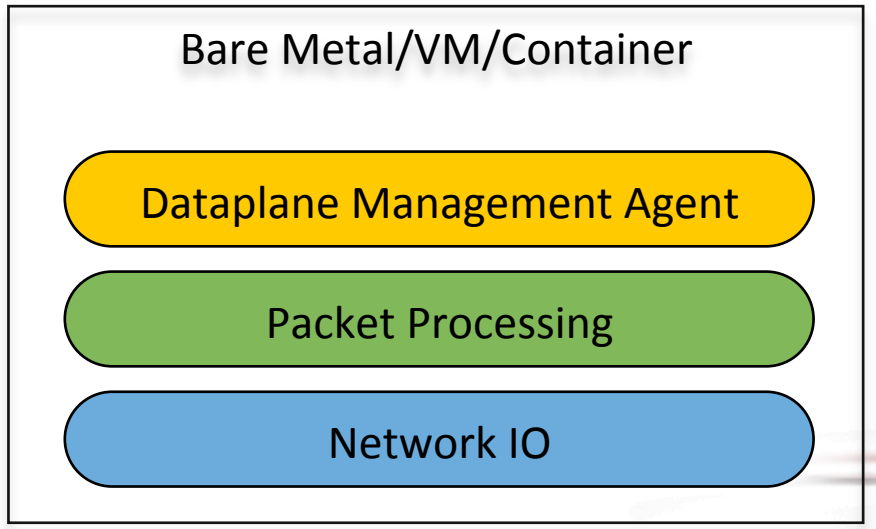
VPP: The Fast Dataplane

FD.io: The Universal Dataplane

- Project at Linux Foundation
  - Multi-party
  - Multi-project
- Software Dataplane
  - High throughput
  - Low Latency
  - Feature Rich
  - Resource Efficient
  - Bare Metal/VM/Container
  - Multiplatform

- Fd.io Scope:
  - **Network IO -** NIC/vNIC <-> cores/threads
  - **Packet Processing** – Classify/Transform/Prioritize/ Forward/Terminate
  - **Dataplane Management Agents -** ControlPlane



Multiparty: Broad Membership

Service Providers

AT&T

COMCAST

Network Vendors

CISCO

ERICSSON

HUAWEI

ZTE

Metaswitch Networks

BROCADE

Chip Vendors

intel

CAVIUM NETWORKS

Integrators

redhat.

INOCYBE OPENDAYLIGHTDSTRBUTION

Multiparty: Broad Contribution

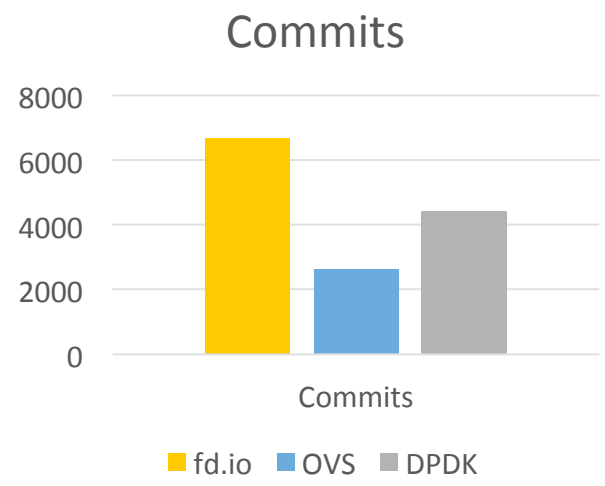
Yandex

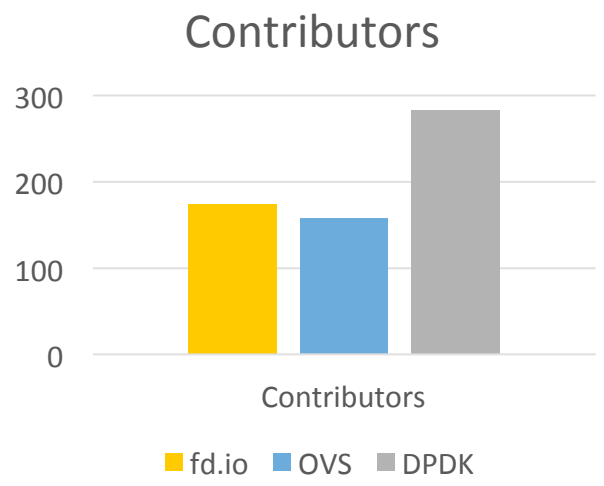
Qiniu

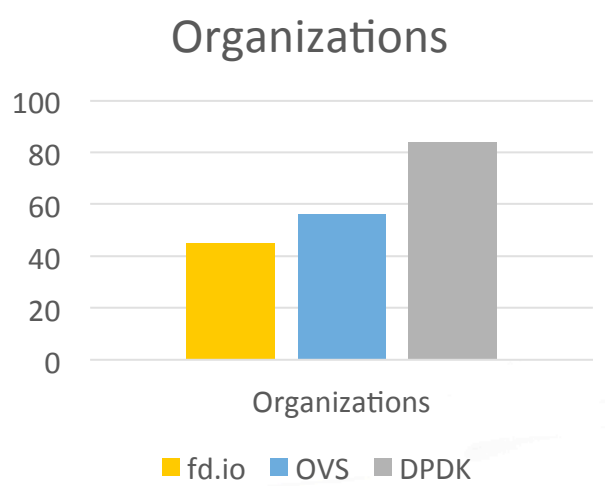
Universitat Politècnica de Catalunya (UPC)

Code Activity

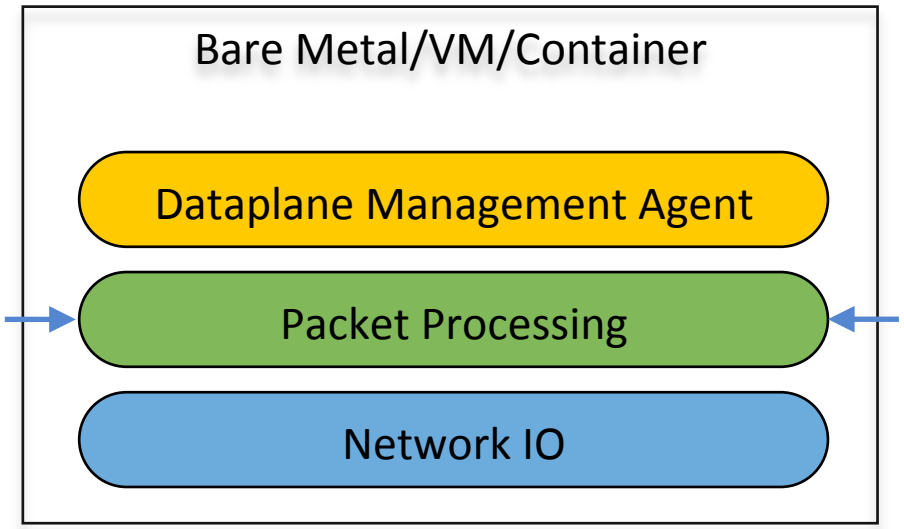
2016-02-11 to 2017-05-14	Fd.io	OVS	DPDK
Commits	6662	2625	4430
Contributors	173	157	282
Organizations	45	56	84





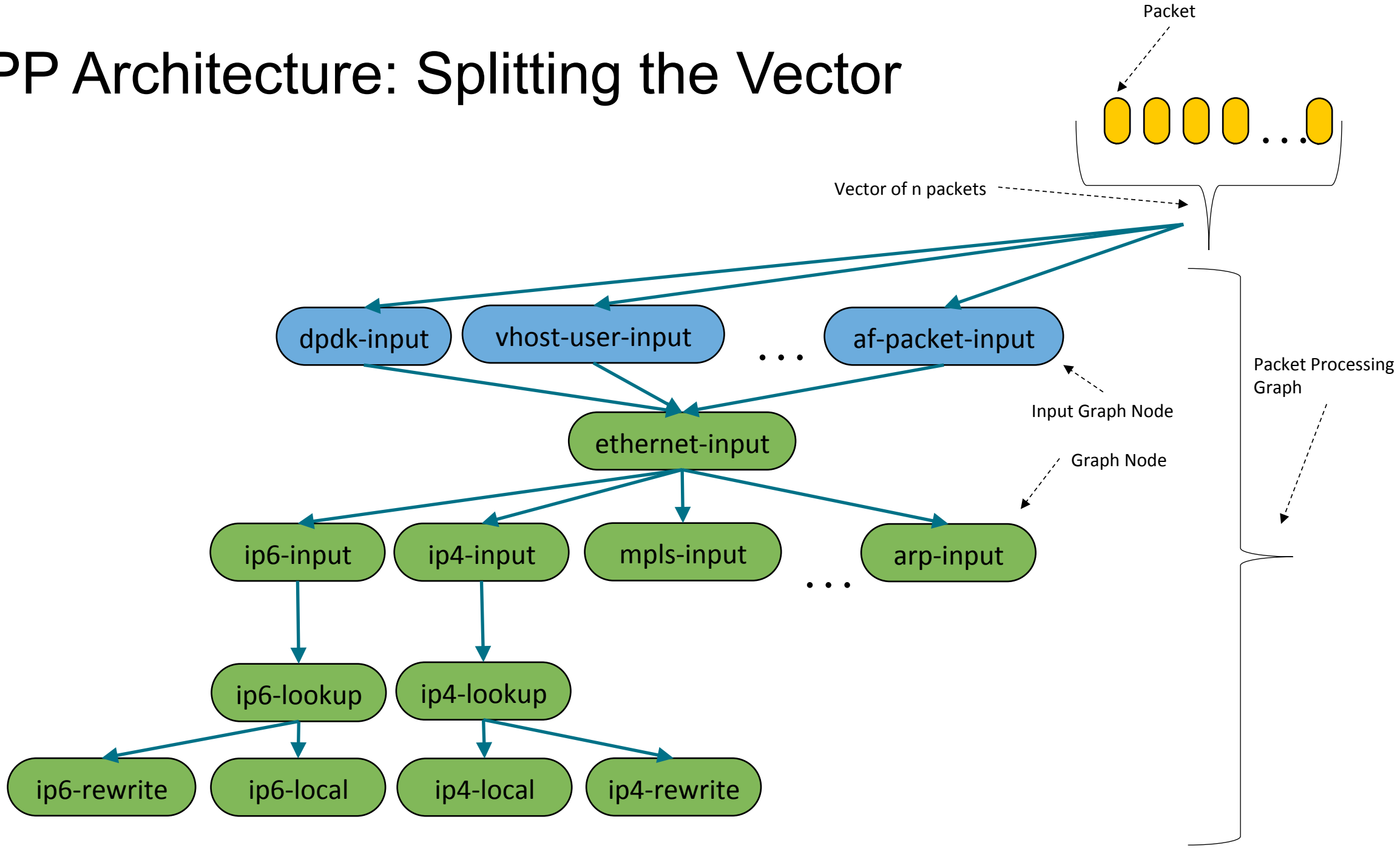


Vector Packet Processor - *VPP*

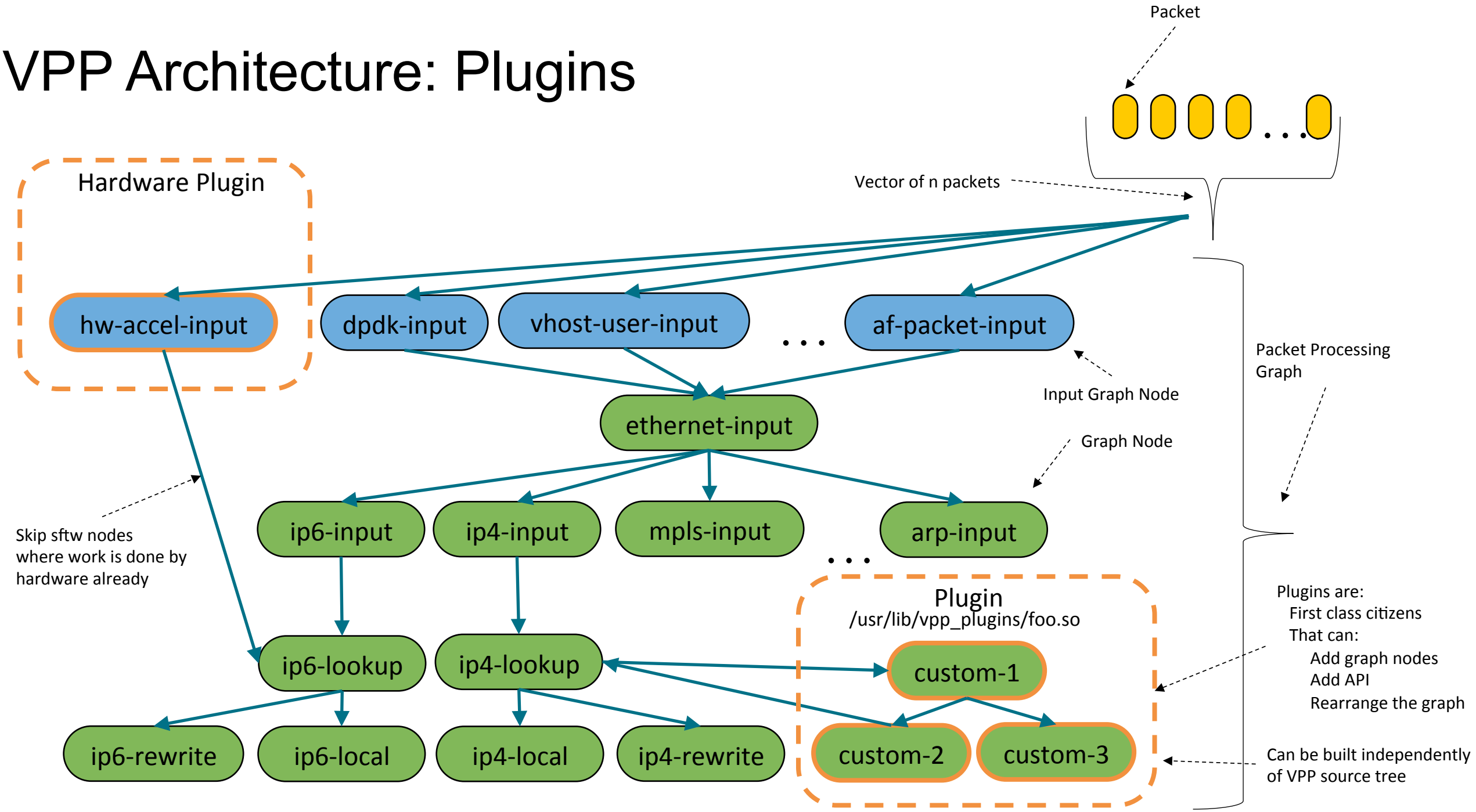


- Packet Processing Platform:
  - High performance
  - Linux User space
  - Run's on commodity CPUs: intel /ARM / Power
  - Shipping at volume in server & embedded products since 2004.

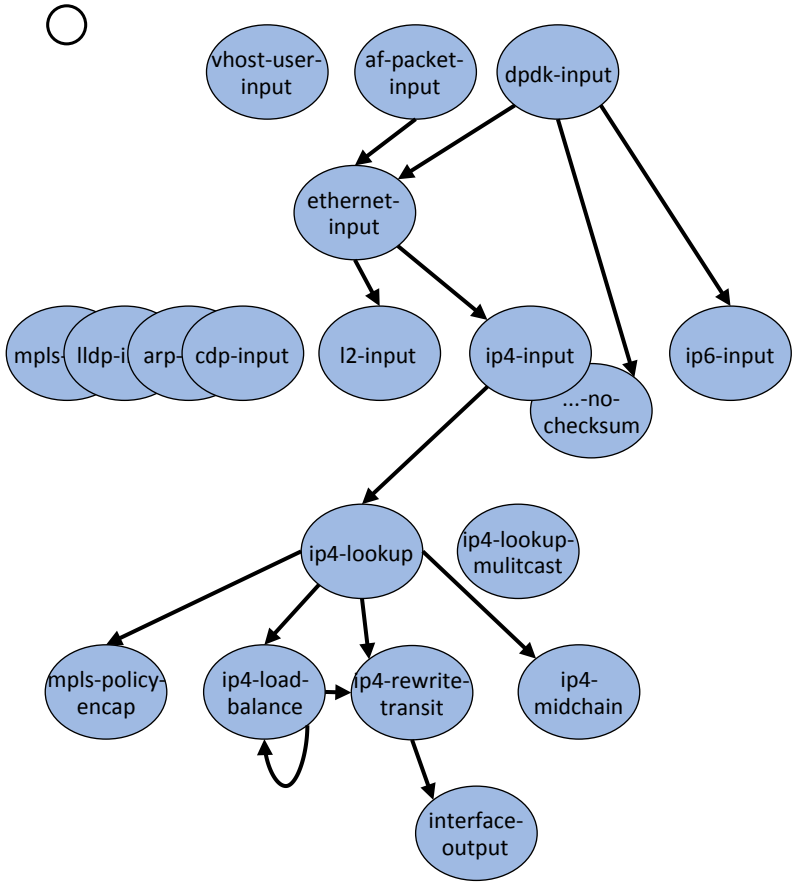
VPP Architecture: Splitting the Vector



VPP Architecture: Plugins

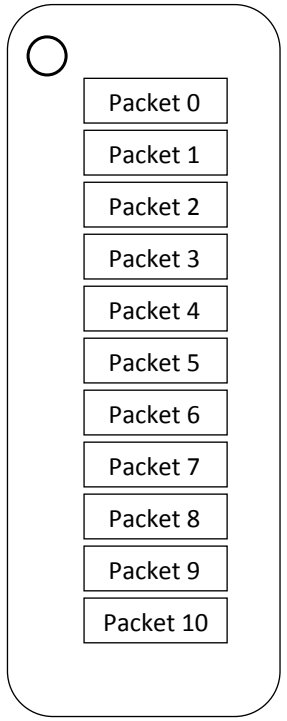


VPP: How does it work?



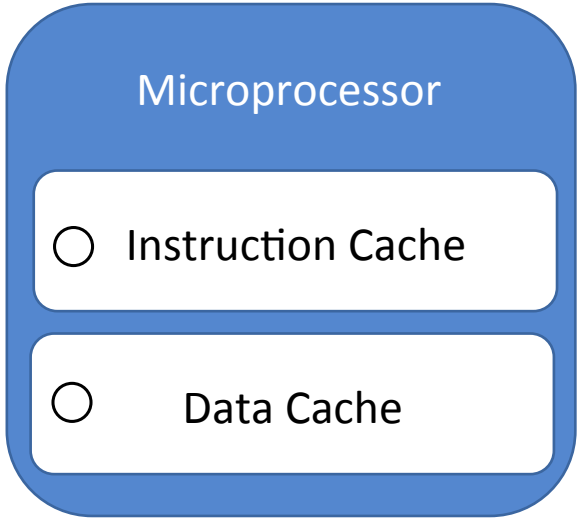
Packet processing is decomposed into a directed graph node …

* approx. 173 nodes in default deployment



… packets moved through graph nodes in vector …

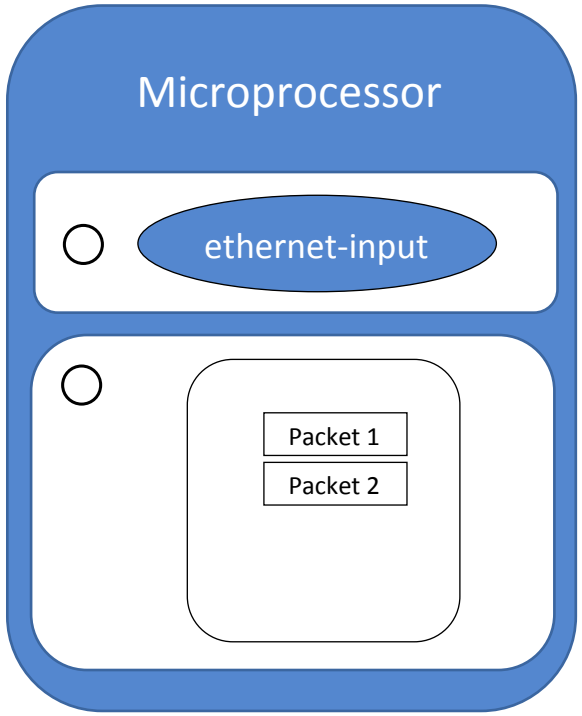
… graph nodes are optimized to fit inside the instruction cache …



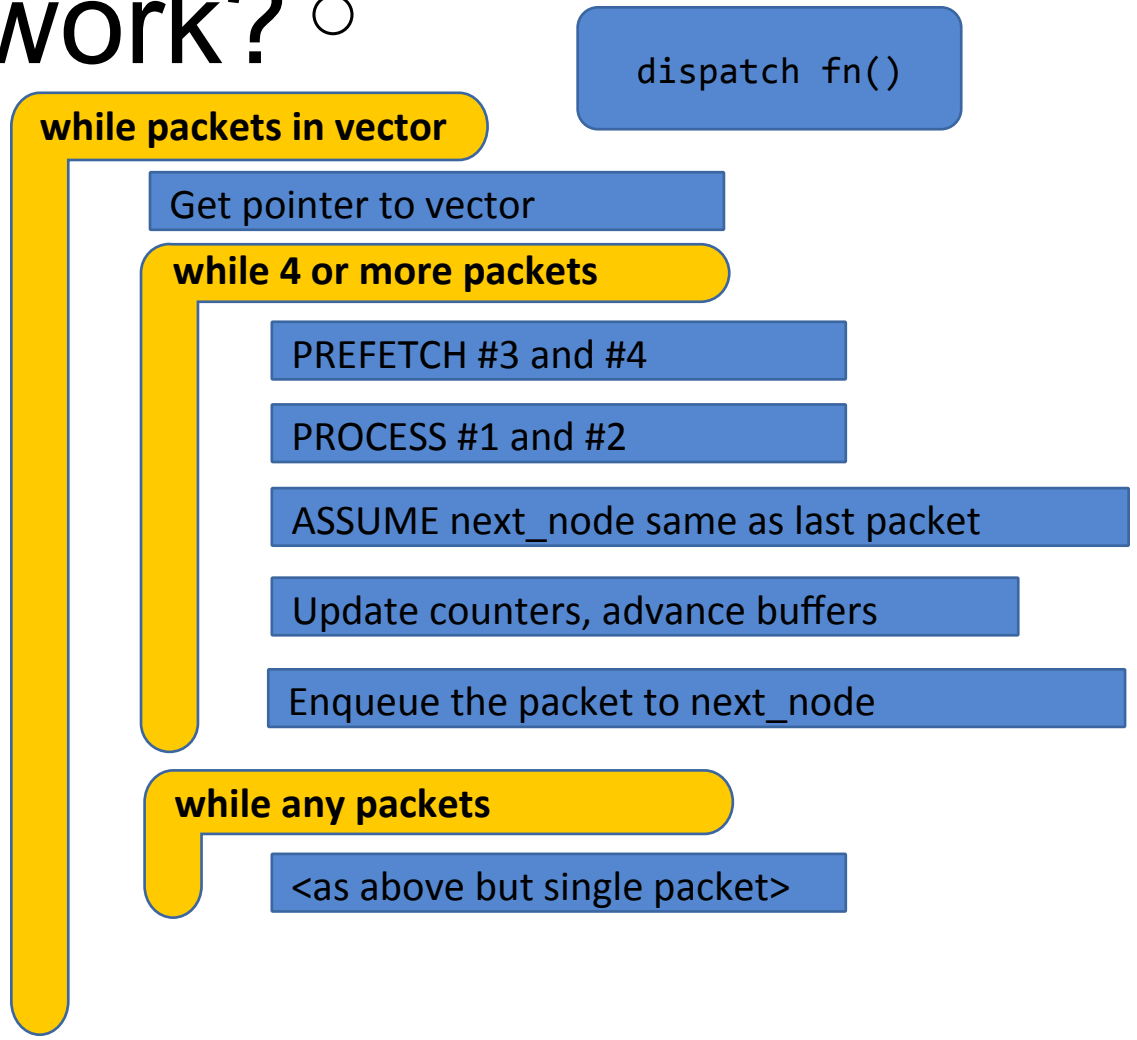
… packets are pre-fetched, into the data cache …

VPP: How does it work? ○

… instruction cache is warm with the instructions from a single graph node …

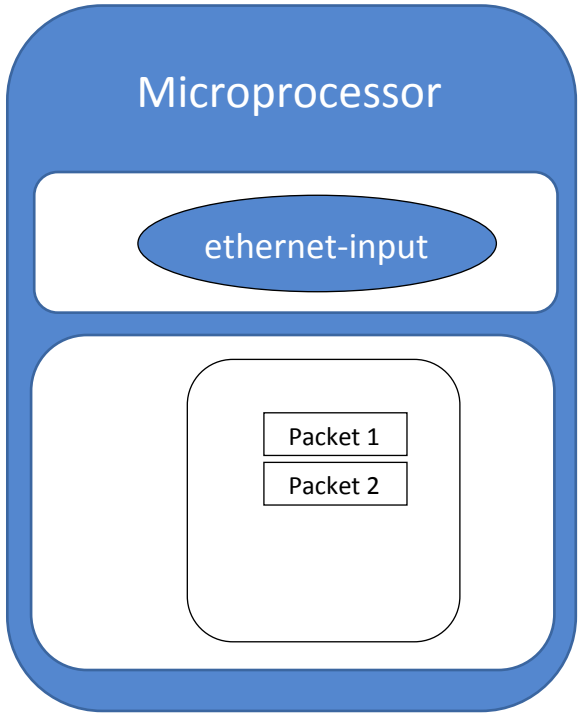


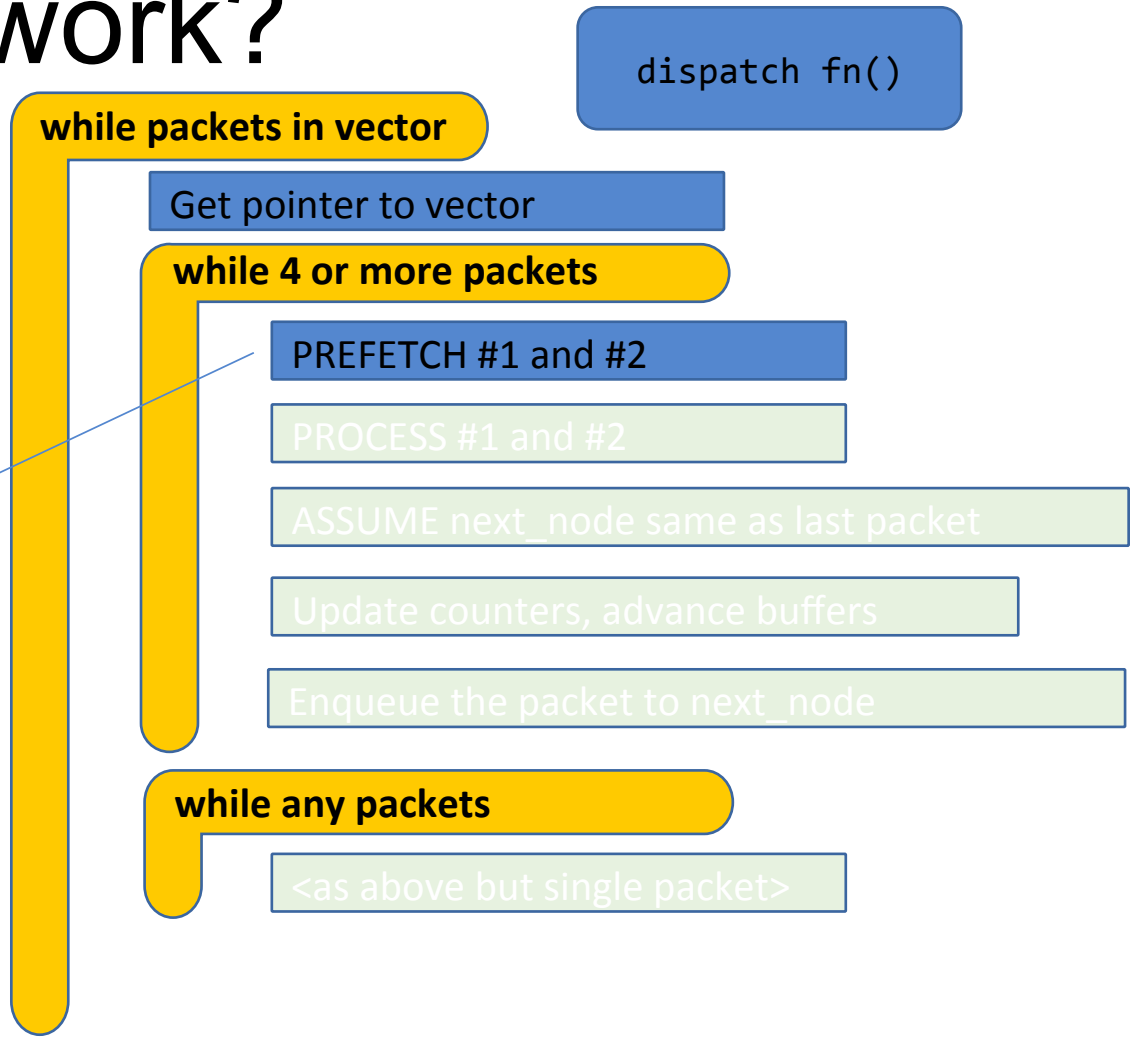
… data cache is warm with a small number of packets ..



… packets are processed in groups of four, any remaining packets are processed on by one …

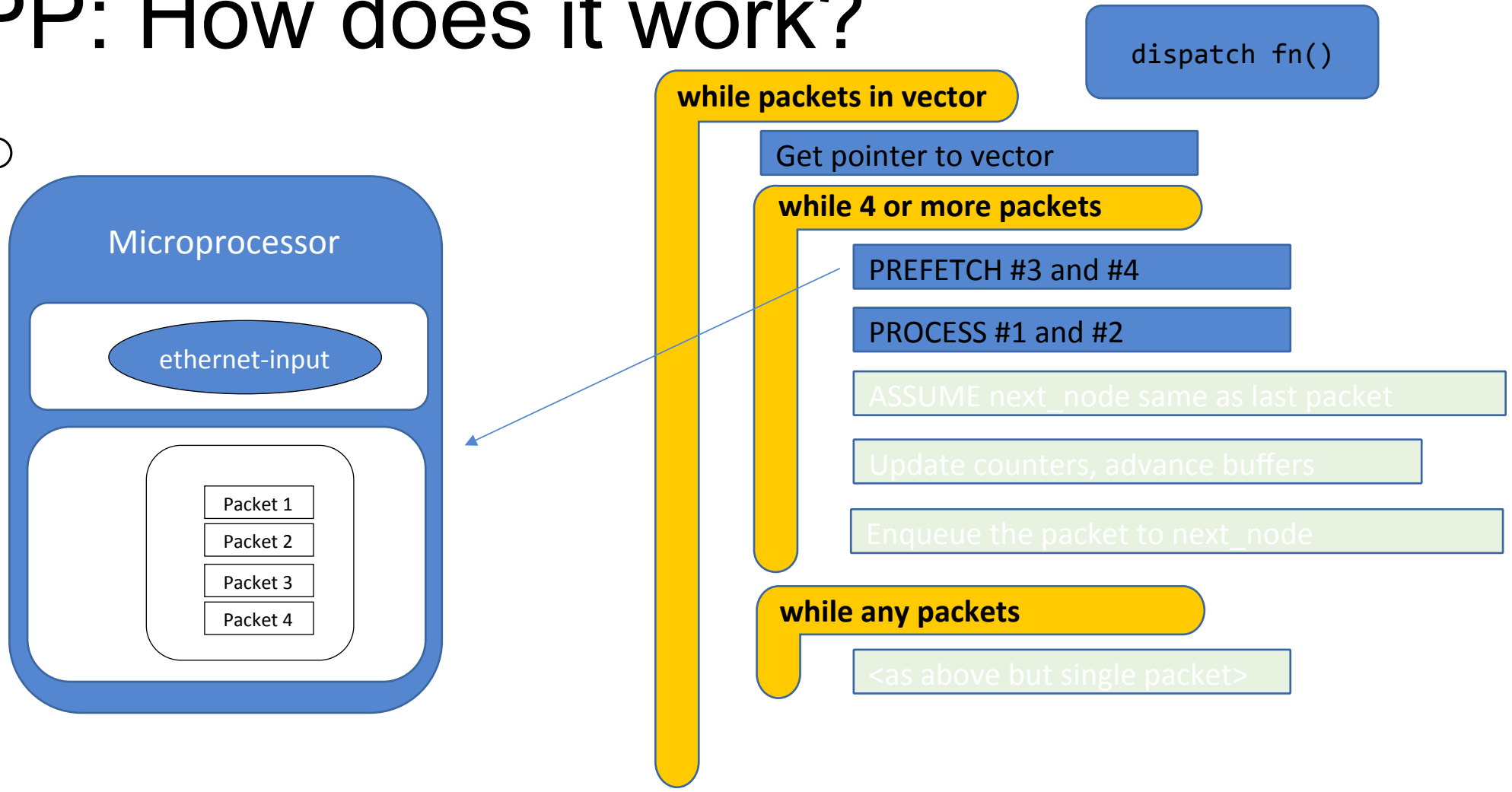
VPP: How does it work?





… prefetch packets #1 and #2 …

VPP: How does it work?

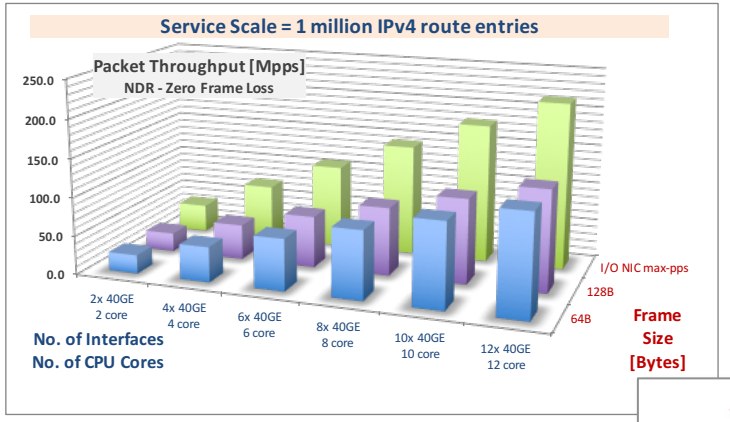


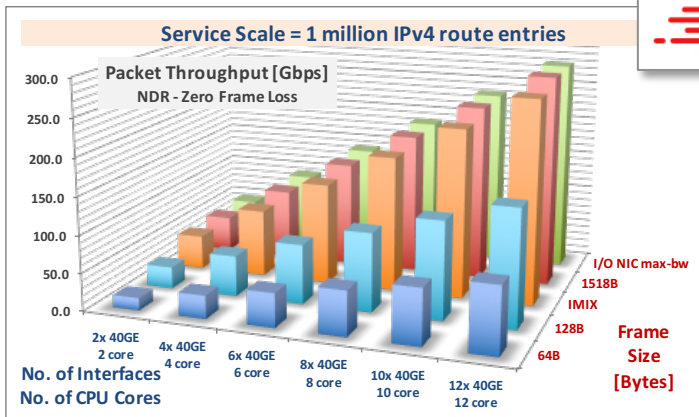
… process packet #3 and #4 …
… update counters, enqueue packets to the next node …

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

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

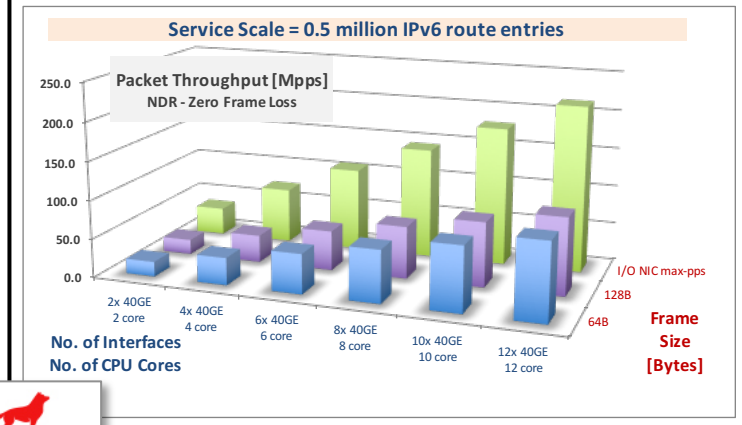
IPv4 Routing

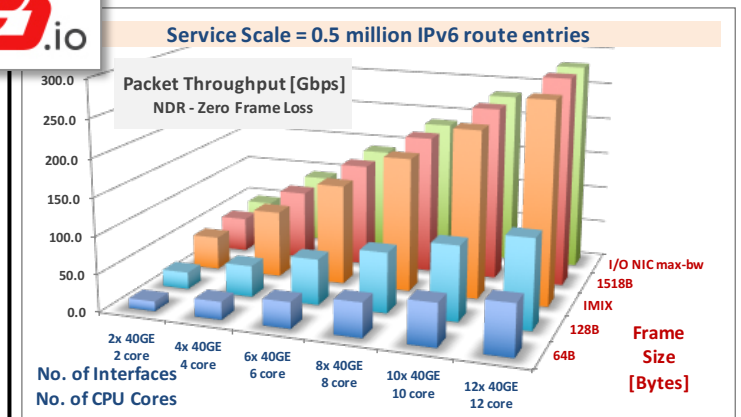




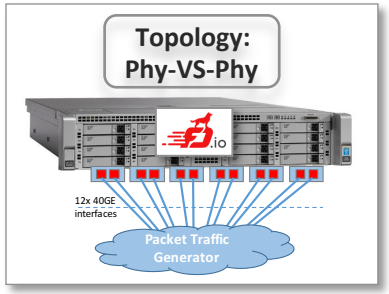
IPv4 Thput [Mpps]	2x 40GE 2 core	4x 40GE 4 core	6x 40GE 6 core	8x 40GE 8 core	10x 40GE 10 core	12x 40GE 12 core
64B	24.0	45.4	66.7	88.1	109.4	130.8
128B	24.0	45.4	66.7	88.1	109.4	130.8
IMIX	15.0	30.0	45.0	60.0	75.0	90.0
1518B	3.8	7.6	11.4	15.2	19.0	22.8
I/O NIC max-pps	35.8	71.6	107.4	143.2	179	214.8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



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

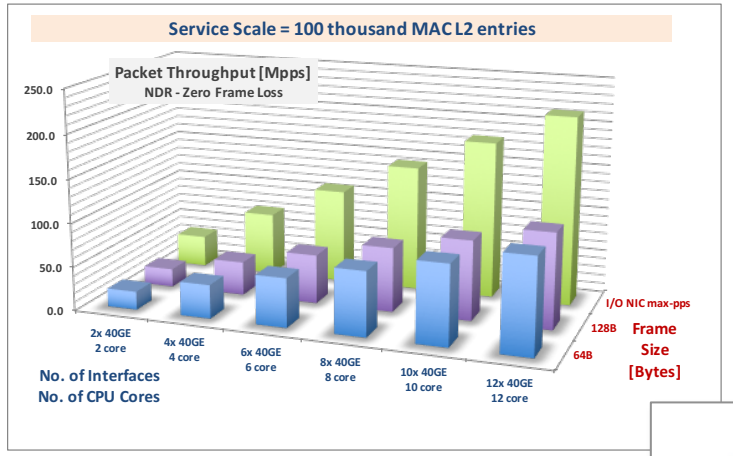
Software
Linux: Ubuntu 16.04.1 LTS
Kernel: ver. 4.4.0-45-generic
FD.io VPP: VPP v17.01-5~ge234726 (DPDK 16.11)

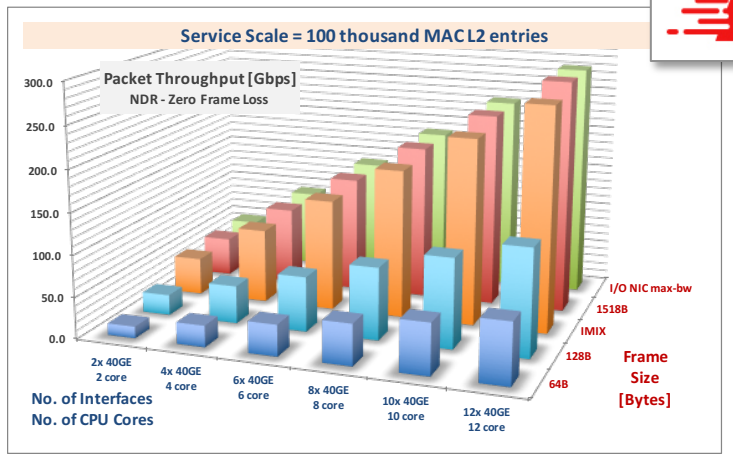
Resources
1 physical CPU core per 40GE port
Other CPU cores available for other services and other work
20 physical CPU cores available in 12x40GE seupt
Lots of Headroom for much more throughput and features

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

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

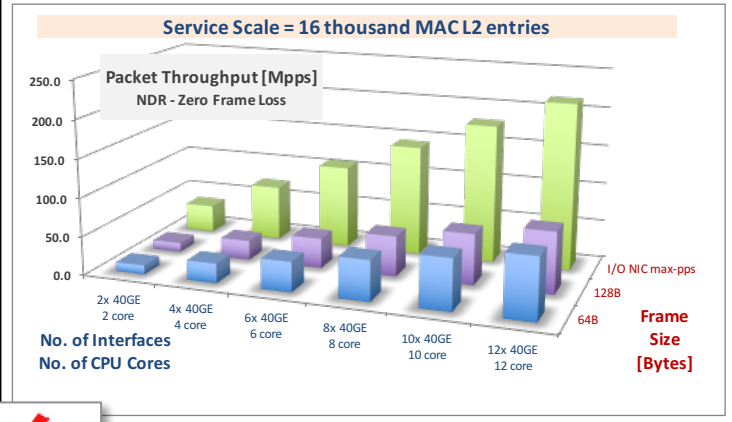
L2 Switching

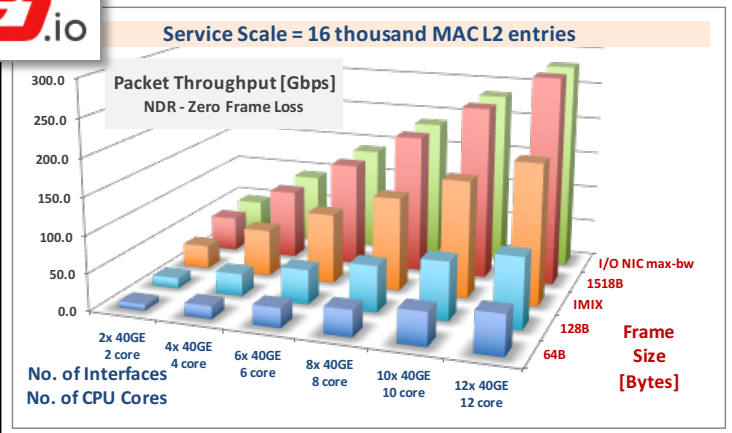




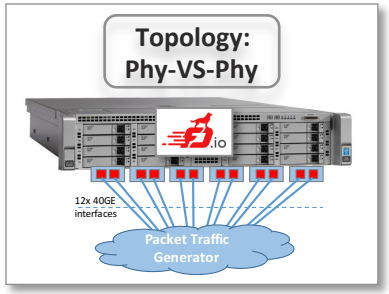
MAC Thput [Mpps]	2x 40GE 2 core	4x 40GE 4 core	6x 40GE 6 core	8x 40GE 8 core	10x 40GE 10 core	12x 40GE 12 core
64B	20.8	38.4	55.9	73.5	91.0	108.6
128B	20.8	38.4	55.9	73.5	91.0	108.6
IMIX	15.0	30.0	45.0	60.0	75.0	90.0
1518B	3.8	7.6	11.4	15.2	19.0	22.8
I/O NIC max-pps	35.8	71.6	107.4	143.2	179	214.8
NIC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

L2 Switching with VXLAN Tunneling





MAC 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	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 NIC max-pps	35.8	71.6	107.4	143.2	179	214.8
NIC max-bw	46.8	93.5	140.3	187.0	233.8	280.5



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

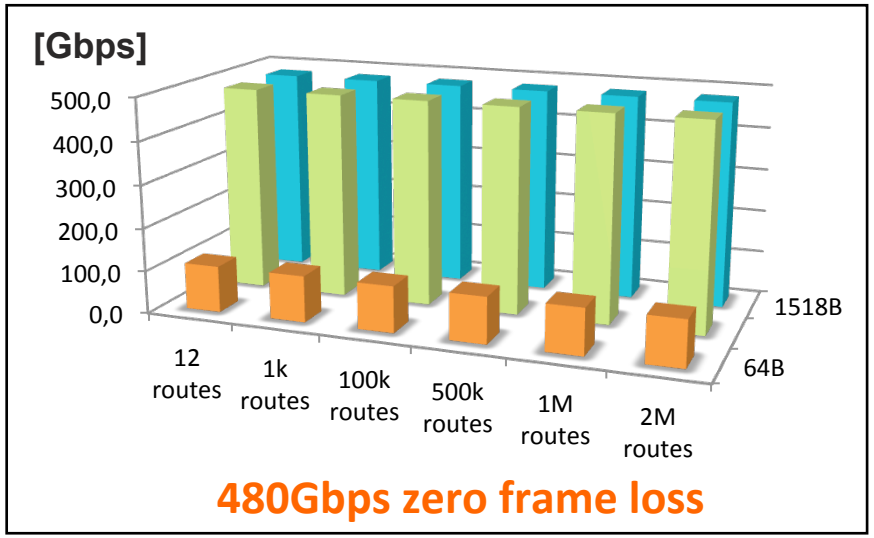
Software
Linux: Ubuntu 16.04.1 LTS
Kernel: ver. 4.4.0-45-generic
FD.io VPP: VPP v17.01-5~ge234726 (DPDK 16.11)

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

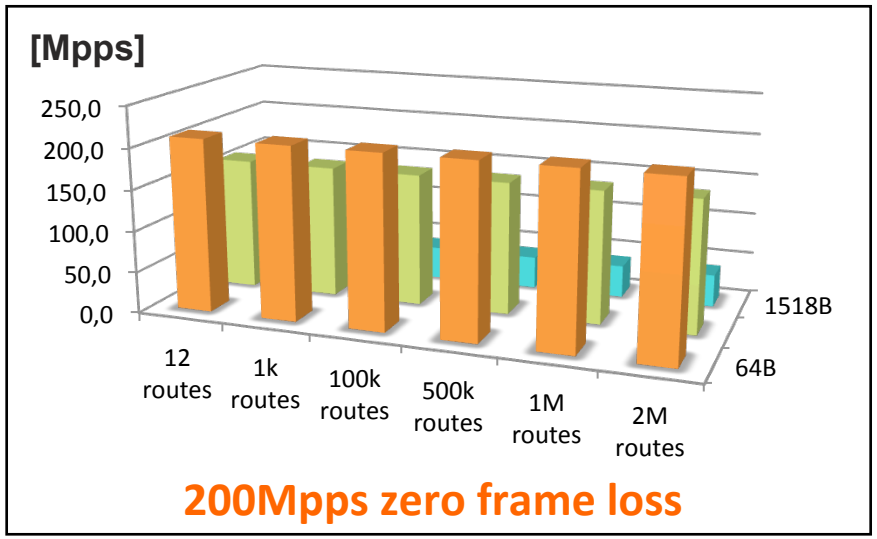
VPP Performance at Scale

Phy-VS-Phy

IPv6, 24 of 72 cores

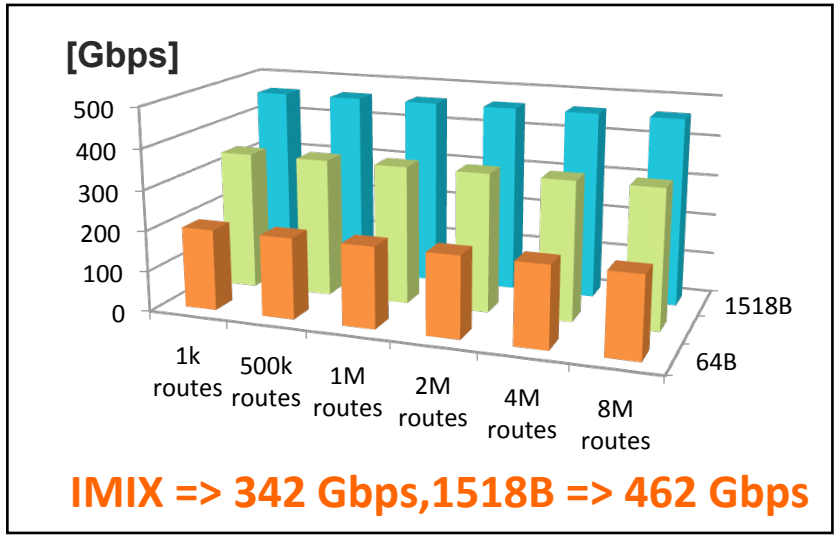


480Gbps zero frame loss

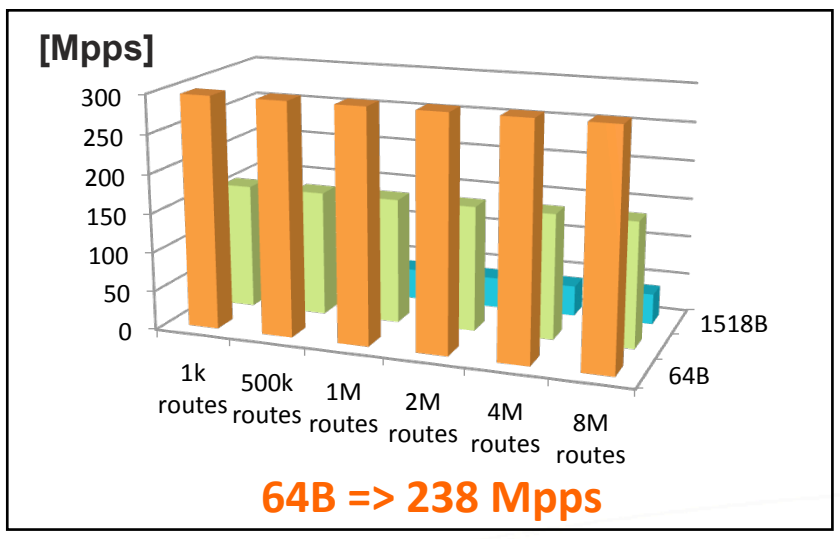


200Mpps zero frame loss

IPv4+ 2k Whitelist, 36 of 72 cores



IMIX => 342 Gbps,1518B => 462 Gbps



64B => 238 Mpps

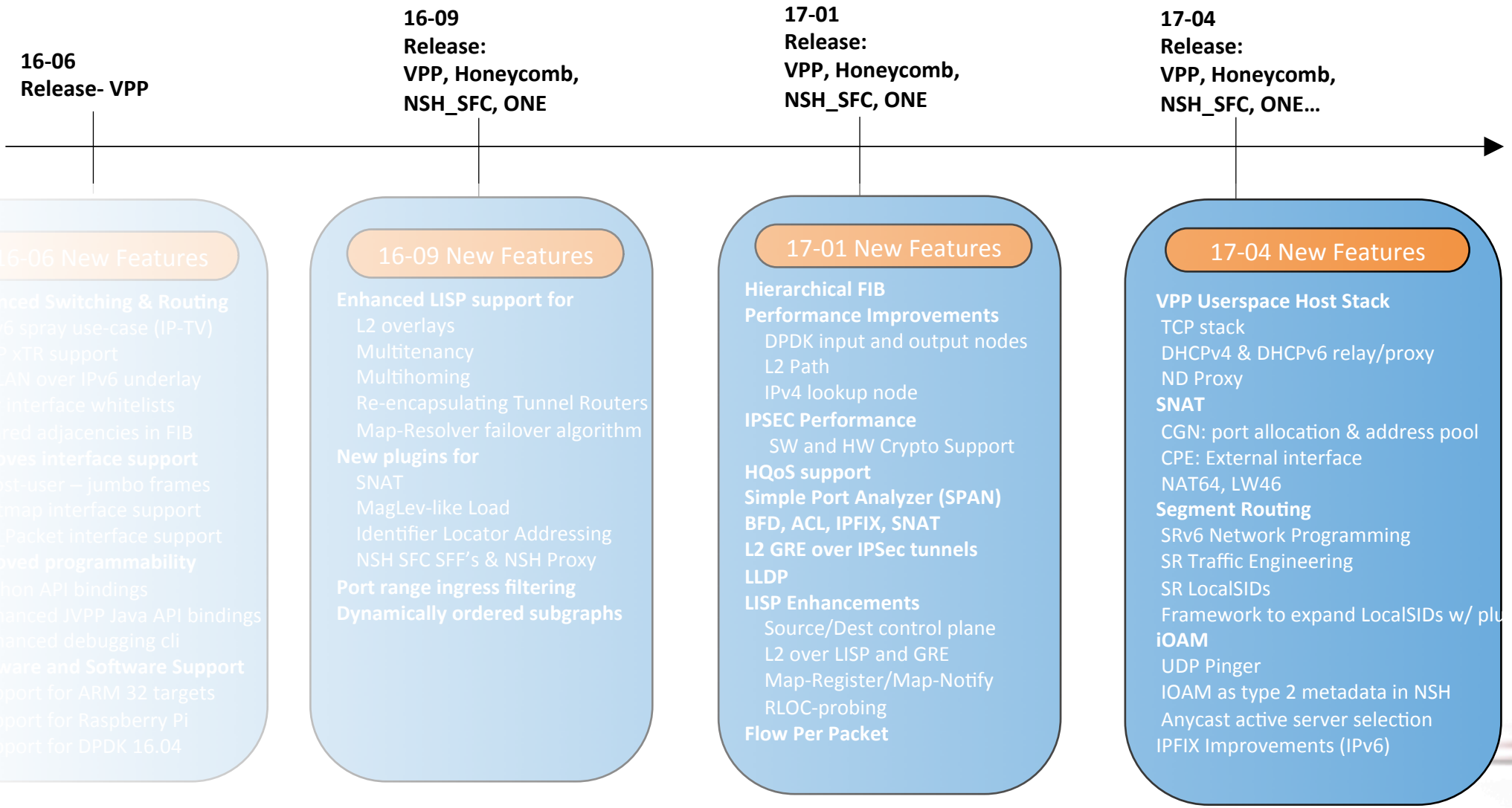
Zero-packet-loss Throughput for 12 port 40GE

Hardware: Cisco UCS C460 M4
Intel® C610 series chipset
4 x Intel® Xeon® Processor E7-8890 v3 (18 cores, 2.5GHz, 45MB Cache)
2133 MHz, 512 GB Total
9 x 2p40GE Intel XL710 18 x 40GE = 720GE !!

Latency
18 x 7.7trillion packets soak test
Average latency: <23 usec
Min Latency: 7…10 usec
Max Latency: 3.5 ms

Headroom
Average vector size ~24-27
Max vector size 255
Headroom for much more throughput/ features
NIC/PCI bus is the limit not vpp

FD.io is evolving rapidly!



Universal Dataplane: Features

Hardware Platforms

Pure Userspace - X86,ARM 32/64,Power
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
IPSEC
Including HW offload when available

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

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

Switching

VLAN Support
Single/ Double tag
L2 forwd 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 assigmt
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
SNAT
Ingress Port Range Filtering
Per interface whitelists
Policy/Security Groups/GBP (Classifier)

Network Services

DHCPv4 client/proxy
DHCPv6 Proxy
MAP/LW46 – IPv4aas
MagLev-like Load
Identifier Locator Addressing
NSH SFC SFF's & NSH Proxy
LLDP
BFD
Policer
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

New in 17.04

VPP Userspace Host Stack

TCP stack
DHCPv4 relay multi-destination
DHCPv4 option 82
DHCPv6 relay multi-destination
DHPCv6 relay remote-id
ND Proxy

SNAT

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

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

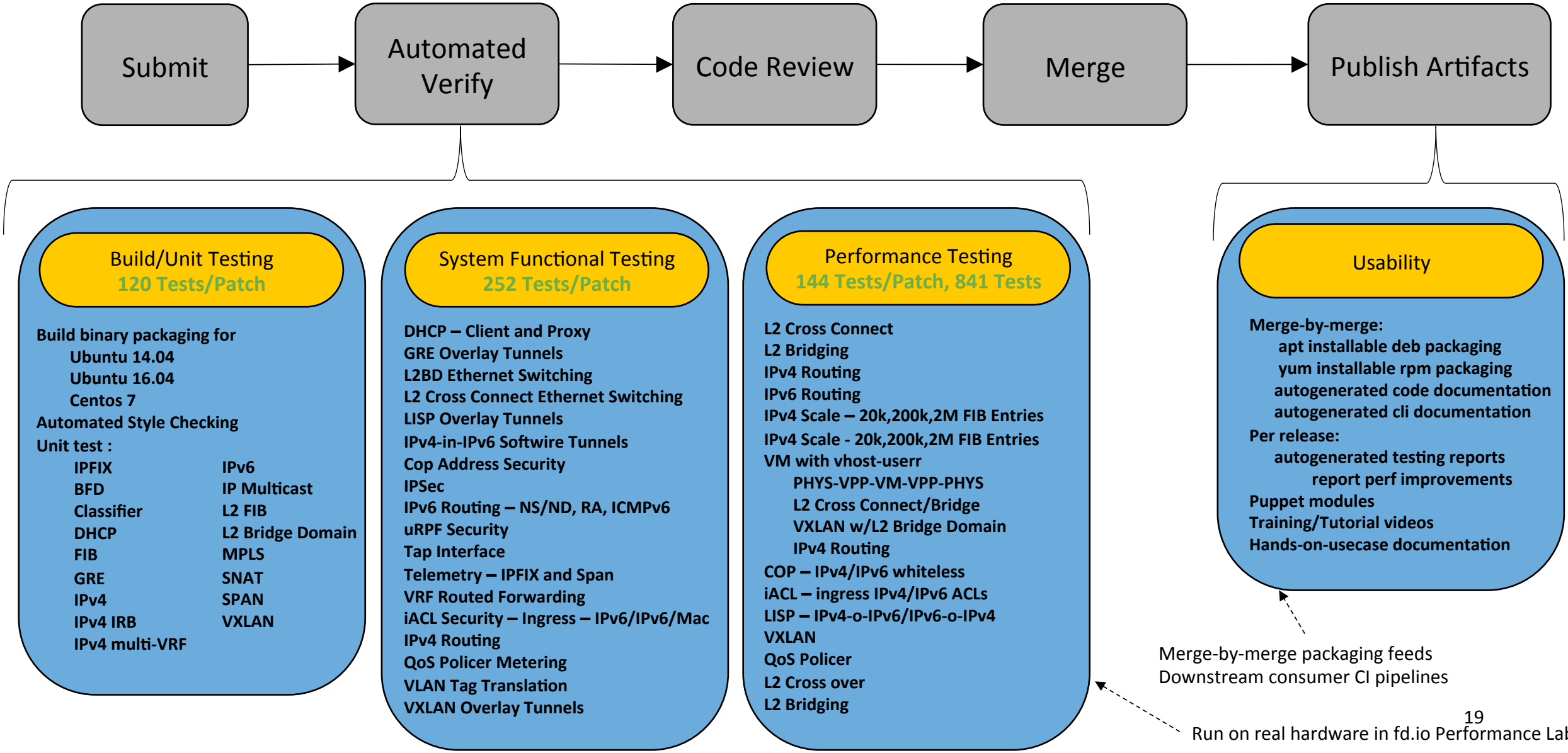
UDP Pinger w/path fault isolation
IOAM as type 2 metadata in NSH
IOAM raw IPFIX collector and analyzer
Anycast active server selection

IPFIX

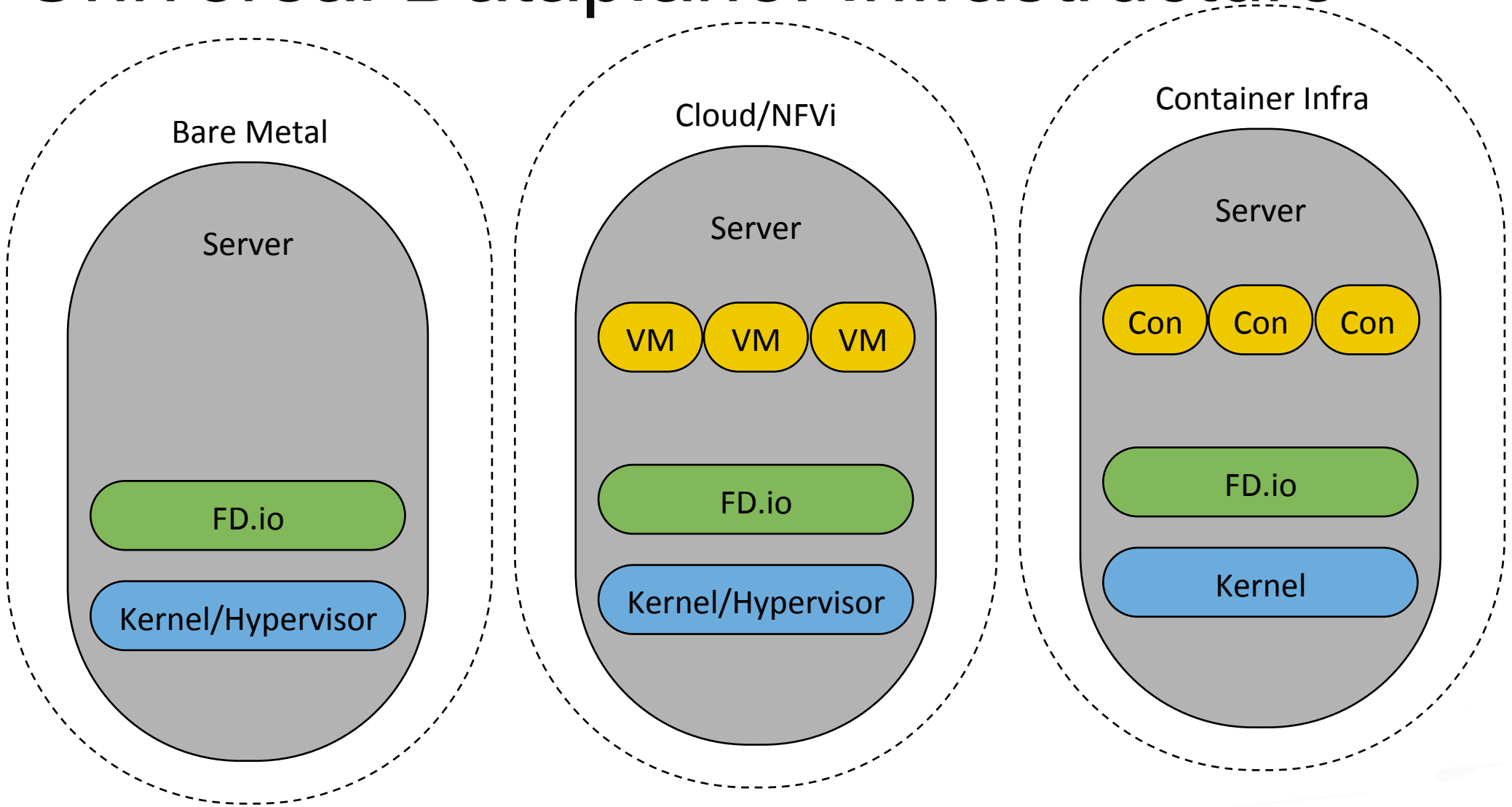
Collect IPv6 information
Per flow state

Continuous Quality, Performance, Usability

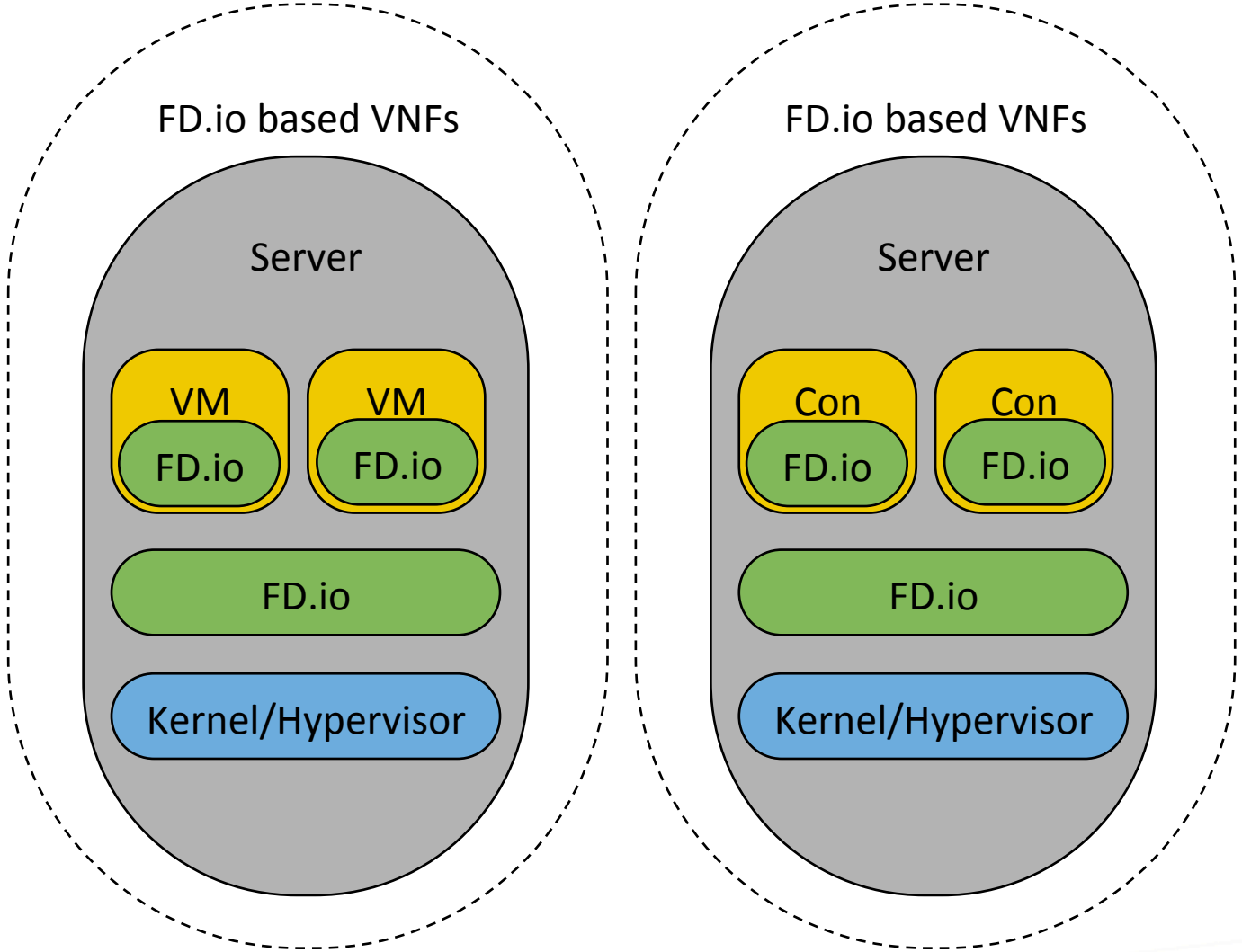
Built into the development process – patch by patch



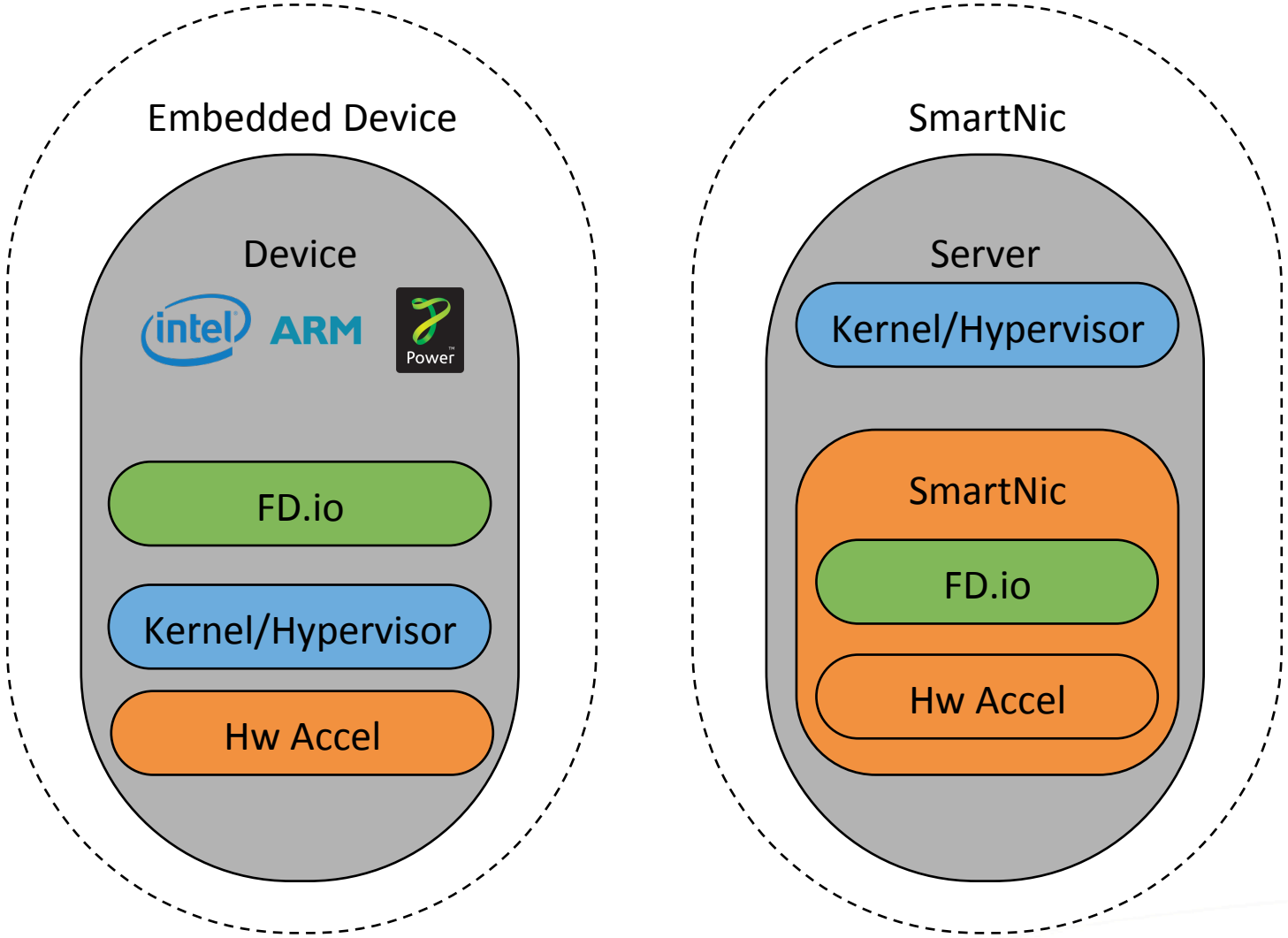
Universal Dataplane: Infrastructure



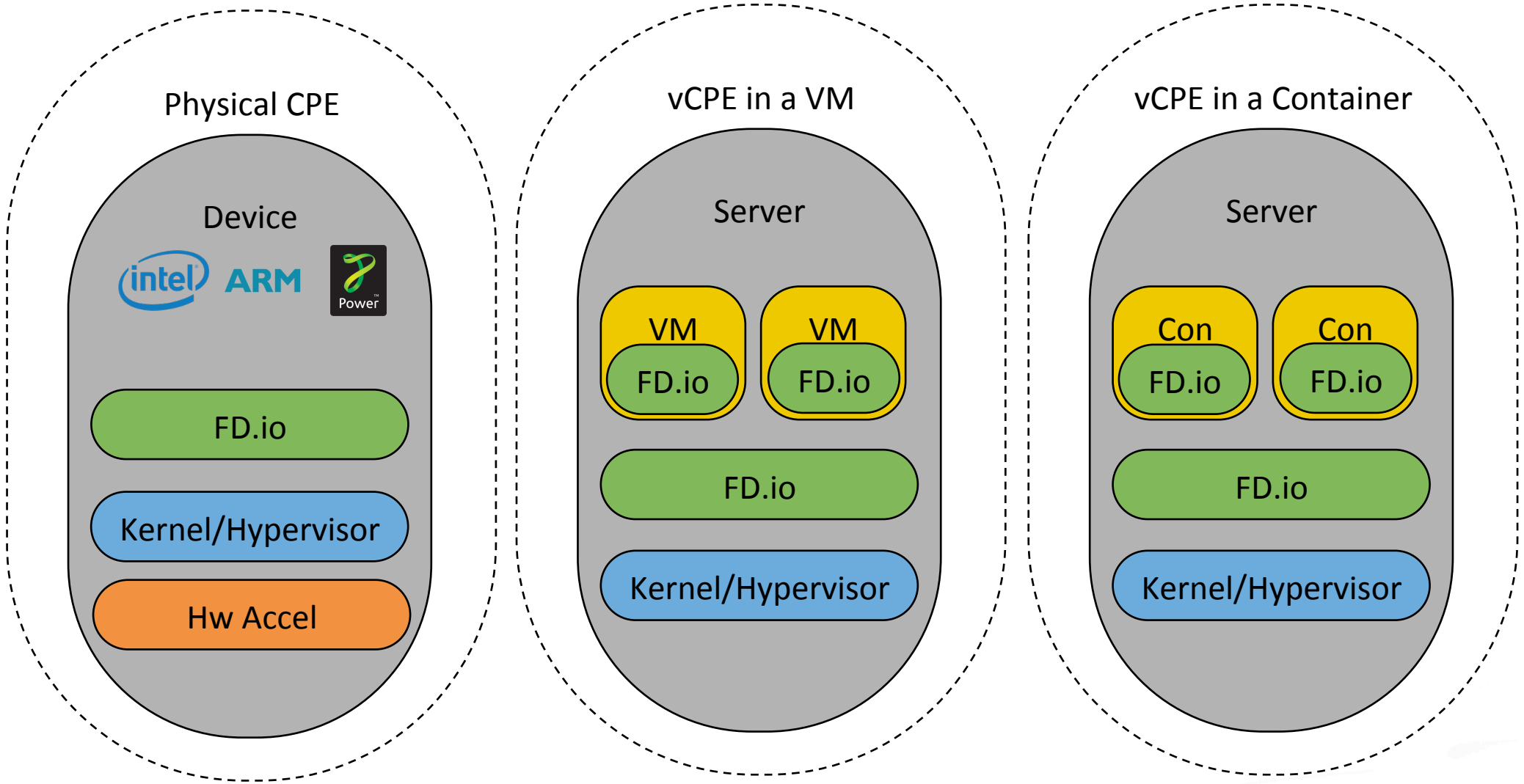
Universal Dataplane: VNFs



Universal Dataplane: Embedded



Universal Dataplane: CPE Example



Opportunities to Contribute

- Firewall
- IDS
- Hardware Accelerators
- Integration with OpenCache
- Control plane – support your favorite SDN Protocol Agent
- Spanning Tree
- DPI
- Test tools
- Cloud Foundry Integration
- Container 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