ons
EUROPE
OPEN NETWORKING //
Integrate, Automate, Accelerate

September 25 - 27, 2018
Amsterdam, The Netherlands

Accelerating the Development of Cloud-native C~~V~~NFs

Giles Heron
Principal Engineer, Cisco
giheron@cisco.com

Maciek Konstantynowicz
FD.io CSIT Project Lead
Distinguished Engineer, Cisco
mkonstan@cisco.com

Damjan Marion
FD.io VPP Committer
Principal Engineer, Cisco
damarion@cisco.com

Agenda

- Context
- FD.io / VPP
- Ligato
- Memif
- The Numbers

DISCLAIMERs

- **'Mileage May Vary'**
  - Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your opinion and investment of any resources. For more complete information about open source performance and benchmark results referred in this material, visit https://wiki.fd.io/view/CSIT and/or https://docs.fd.io/csit/rls1807/report/.
- **Trademarks and Branding**
  - This is an open-source material. Commercial names and brands may be claimed as the property of others.

SDN NFV Evolution to Cloud-native

Moving on from VMs to Pods/Containers

- Network function workloads moving from VMs to Containers

  Native code execution on compute nodes, much less execution overhead

  Lighter workloads, many more of them, much more dynamic environment

- Orchestration moving from OpenStack VMs to K8s Pods/Containers

  Pod/Container networking being addressed: Ligato, Network Services Mesh, Multus

- Pressing need for optimised user-mode packet virtual interface

  Equivalent of “virtio-vhostuser” for Containers, but much faster

  Must be compatible with Container orchestration stack

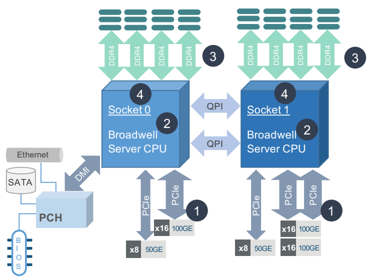
  Opportunity to do it right!

Should allow us to get closer to the native bare-metal limits ...

Bare-Metal Data Plane Performance Limit

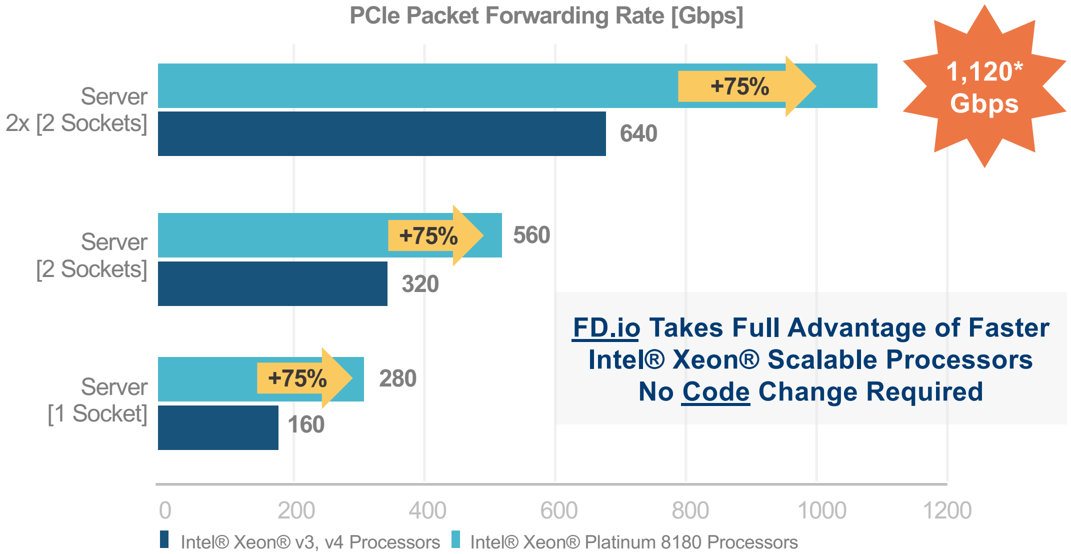
FD.io benefits from increased Processor I/O

YESTERDAY



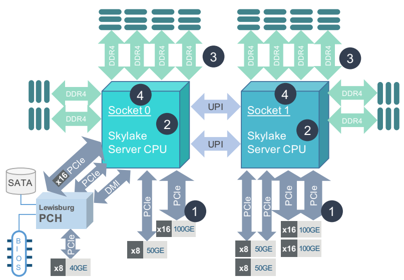
1. Network I/O: **160 Gbps**
2. Core ALU: **4-wide** parallel µops
3. Memory: **4-channels** 2400 MHz
4. Max power: **145W** (TDP)

Intel® Xeon® E5-2699v4
22 Cores, 2.2 GHz, 55MB Cache



* On compute platforms with all PCIe lanes from the Processors routed to PCIe slots.

TODAY



1. Network I/O: **280 Gbps**
2. Core ALU: **5-wide** parallel µops
3. Memory: **6-channels** 2666 MHz
4. Max power: **205W** (TDP)

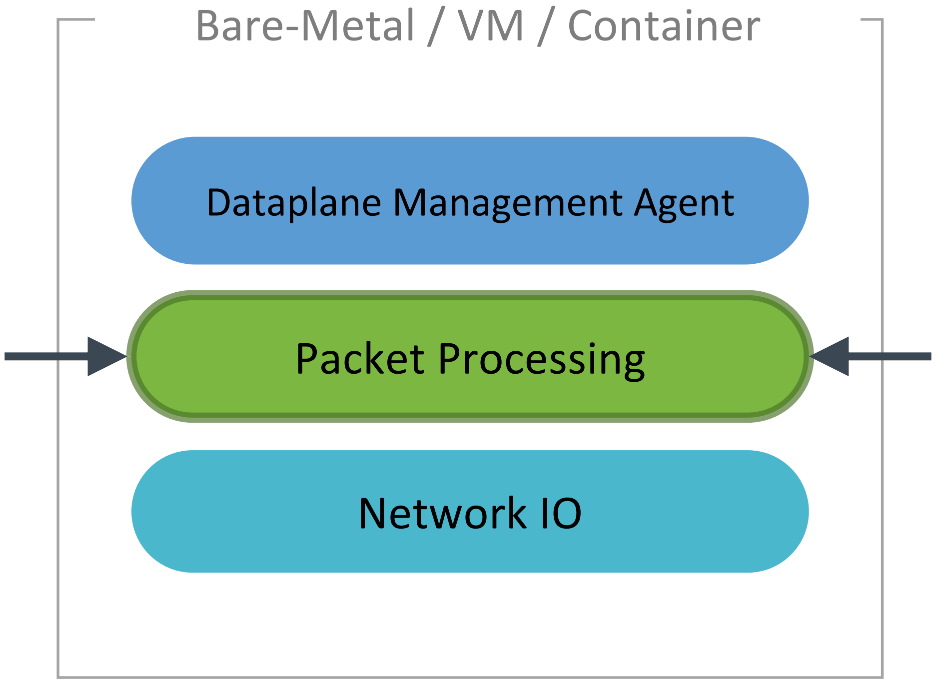
Intel® Xeon® Platinum 8168
24 Cores, 2.7 GHz, 33MB Cache

https://goo.gl/UtbaHy

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

FD.io VPP – Vector Packet Processing

Compute-Optimised SW Networking Platform



Packet Processing Software Platform

- High performance
- Linux user space
- Runs on compute CPUs:
  - And “knows” how to run them well !

Shipping at volume in server & embedded products

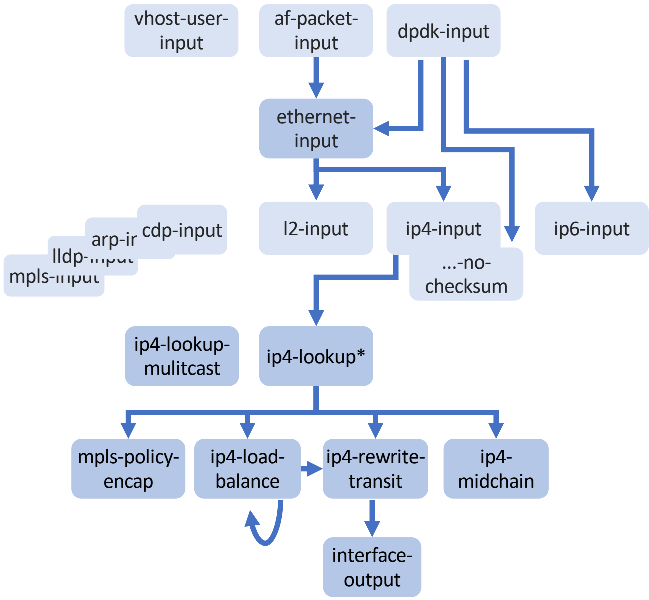
FD.io VPP – How does it work?

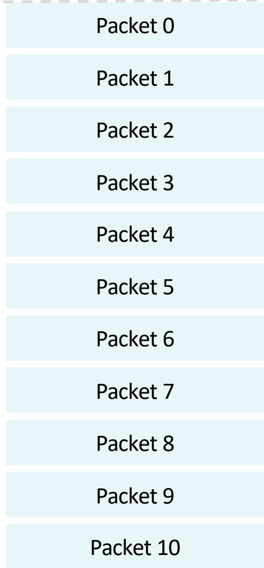
Compute Optimised SW Networking Platform

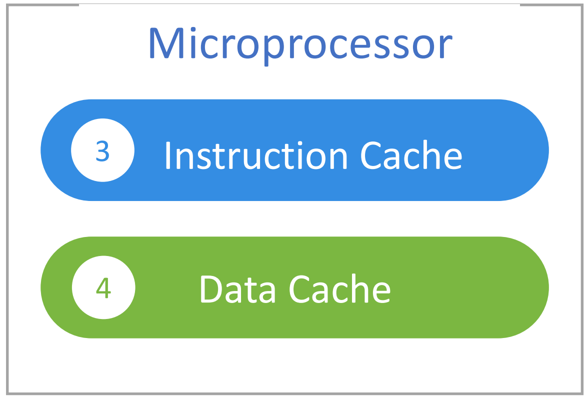
1 Packet processing is decomposed into a directed graph of nodes …

2 … packets move through graph nodes in vector …

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







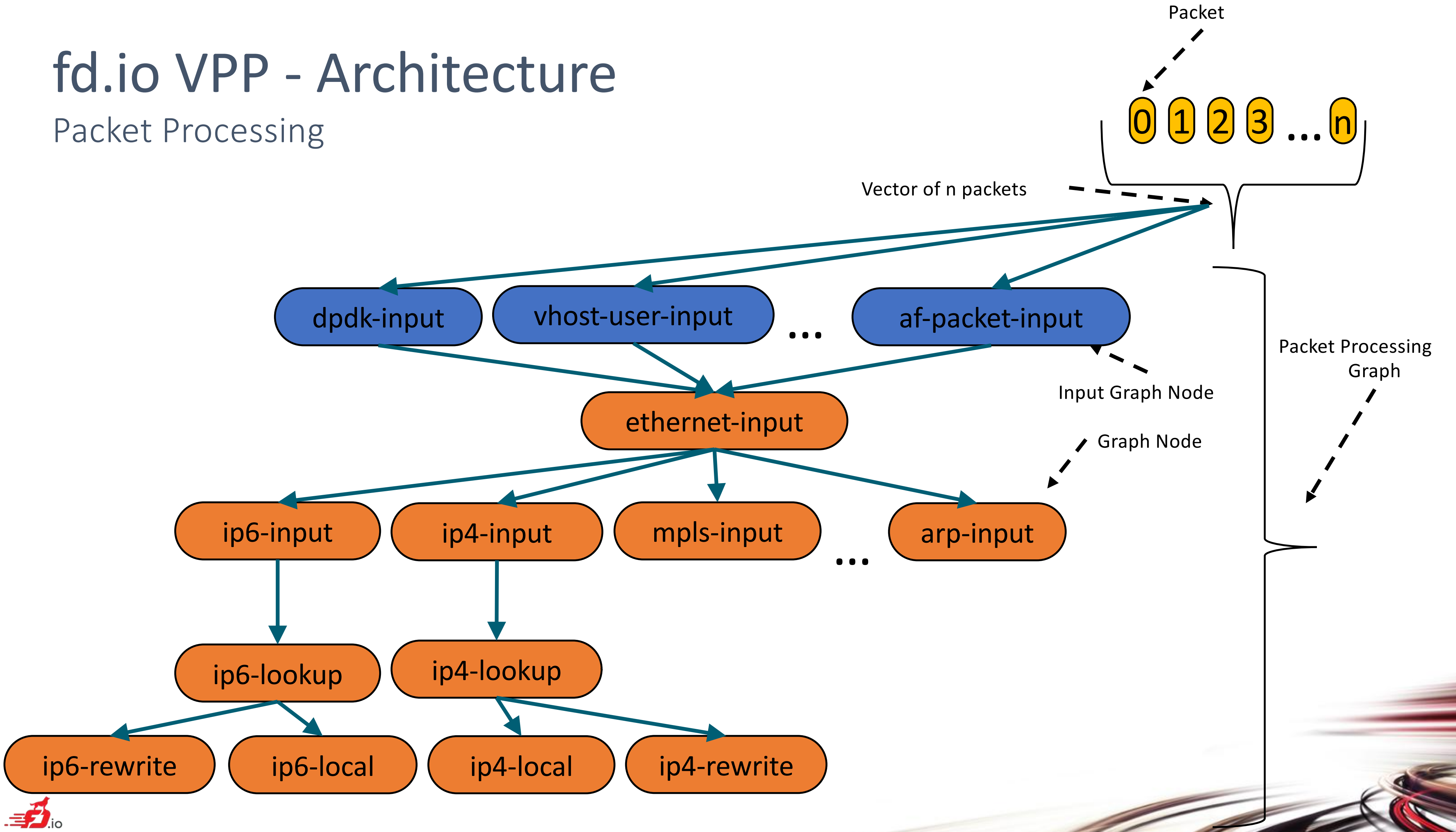
4 … packets are pre-fetched into the data cache.

* Each graph node implements a “micro-NF”, a “micro-NetworkFunction” processing packets.

Makes use of modern Intel® Xeon® Processor micro-architectures.
Instruction cache & data cache always hot ➔ Minimized memory latency and usage.

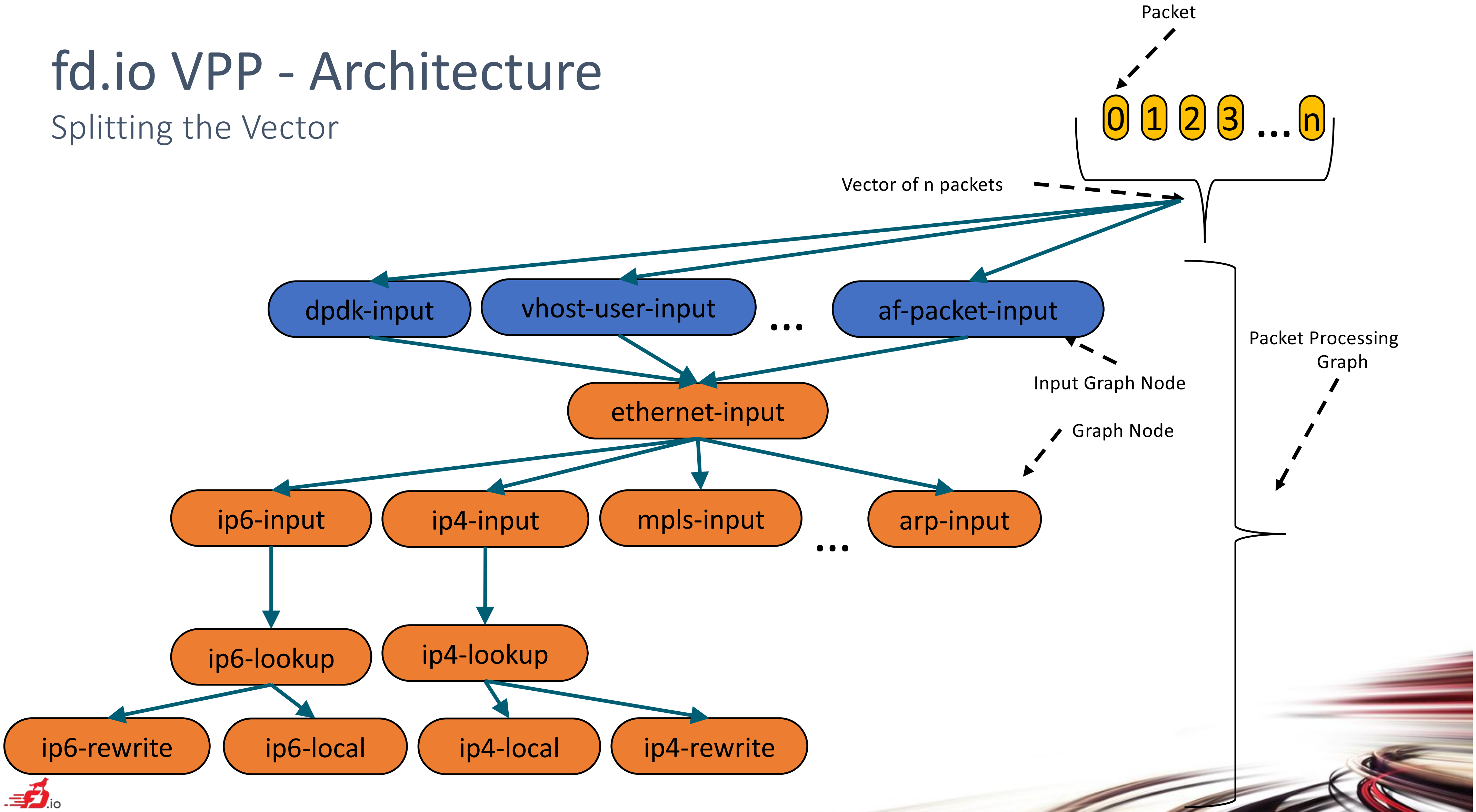
fd.io VPP - Architecture

Packet Processing



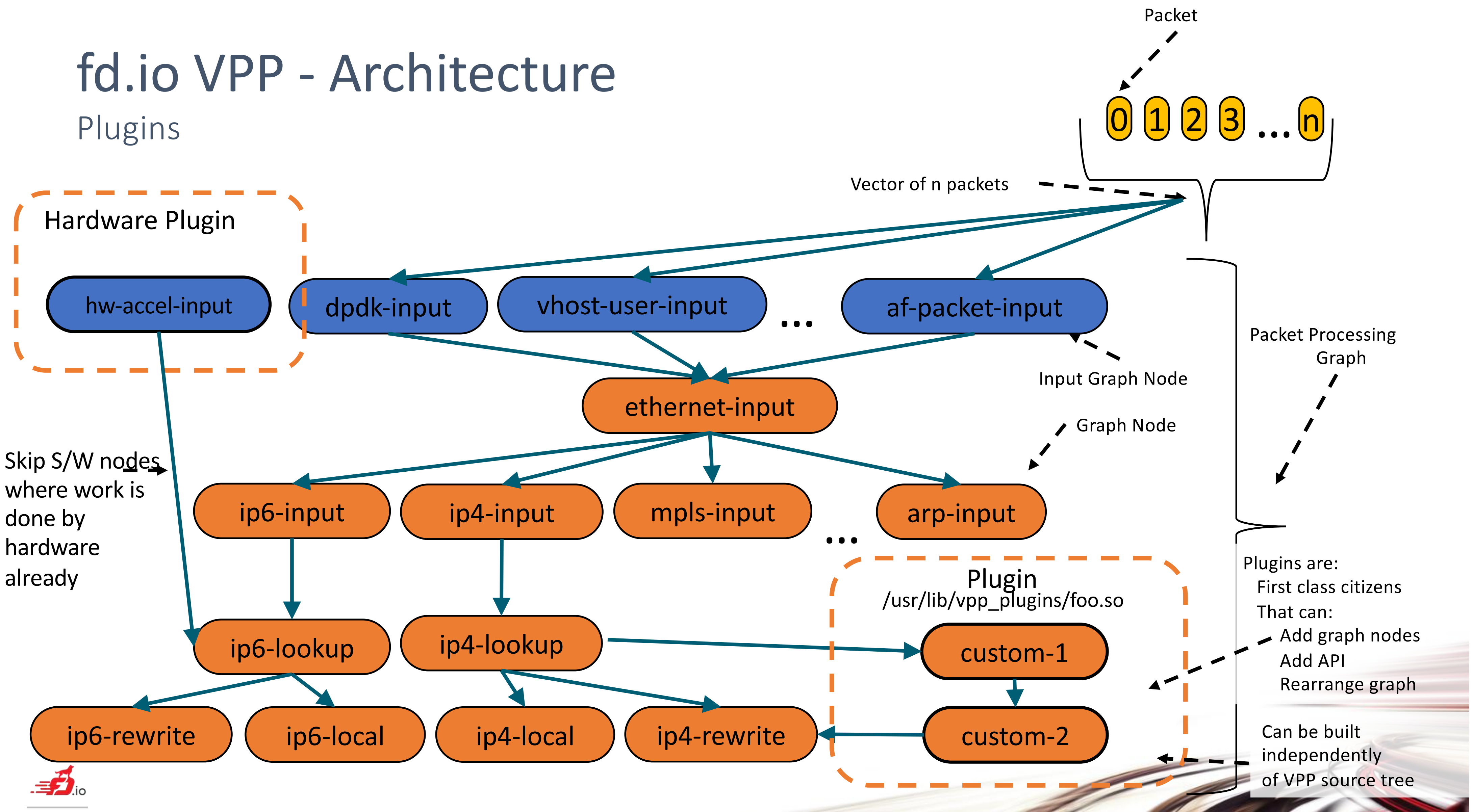
fd.io VPP - Architecture

Splitting the Vector



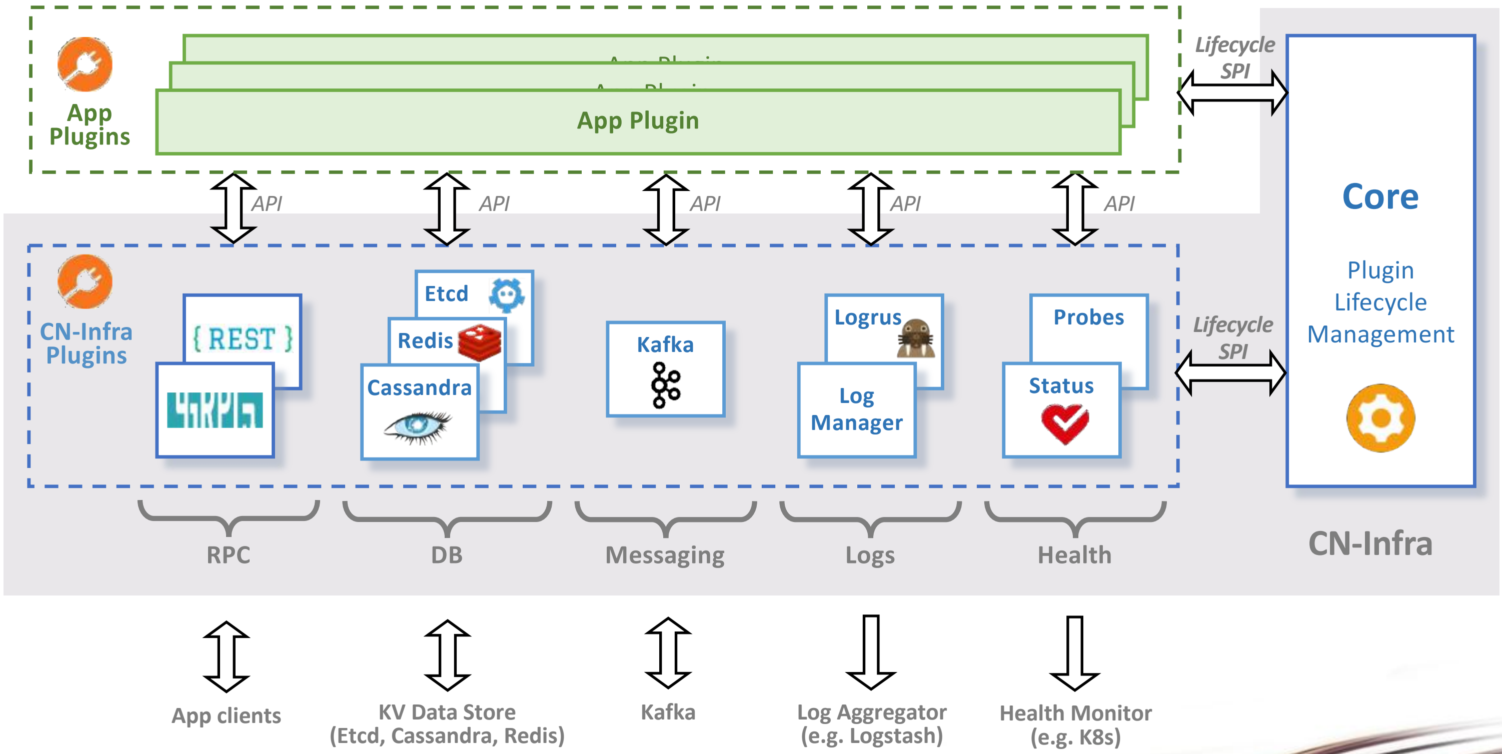
fd.io VPP - Architecture

Plugins



Ligato CN-Infra: a CNF* Development Platform

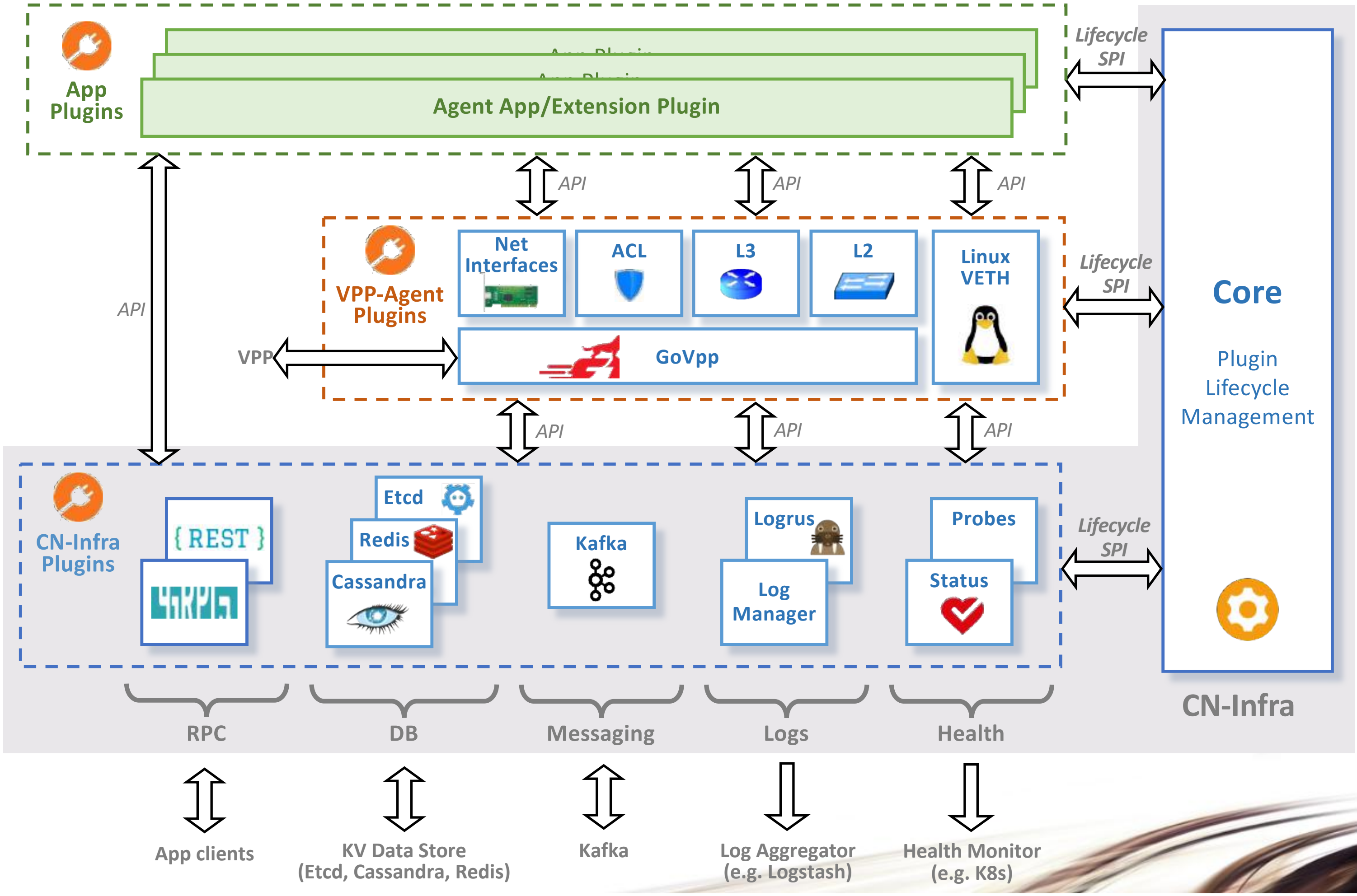
www.github.com/ligato/cn-infra



* CNF – Cloud-native Network Function

Ligato VPP Agent: a CNF Management Agent

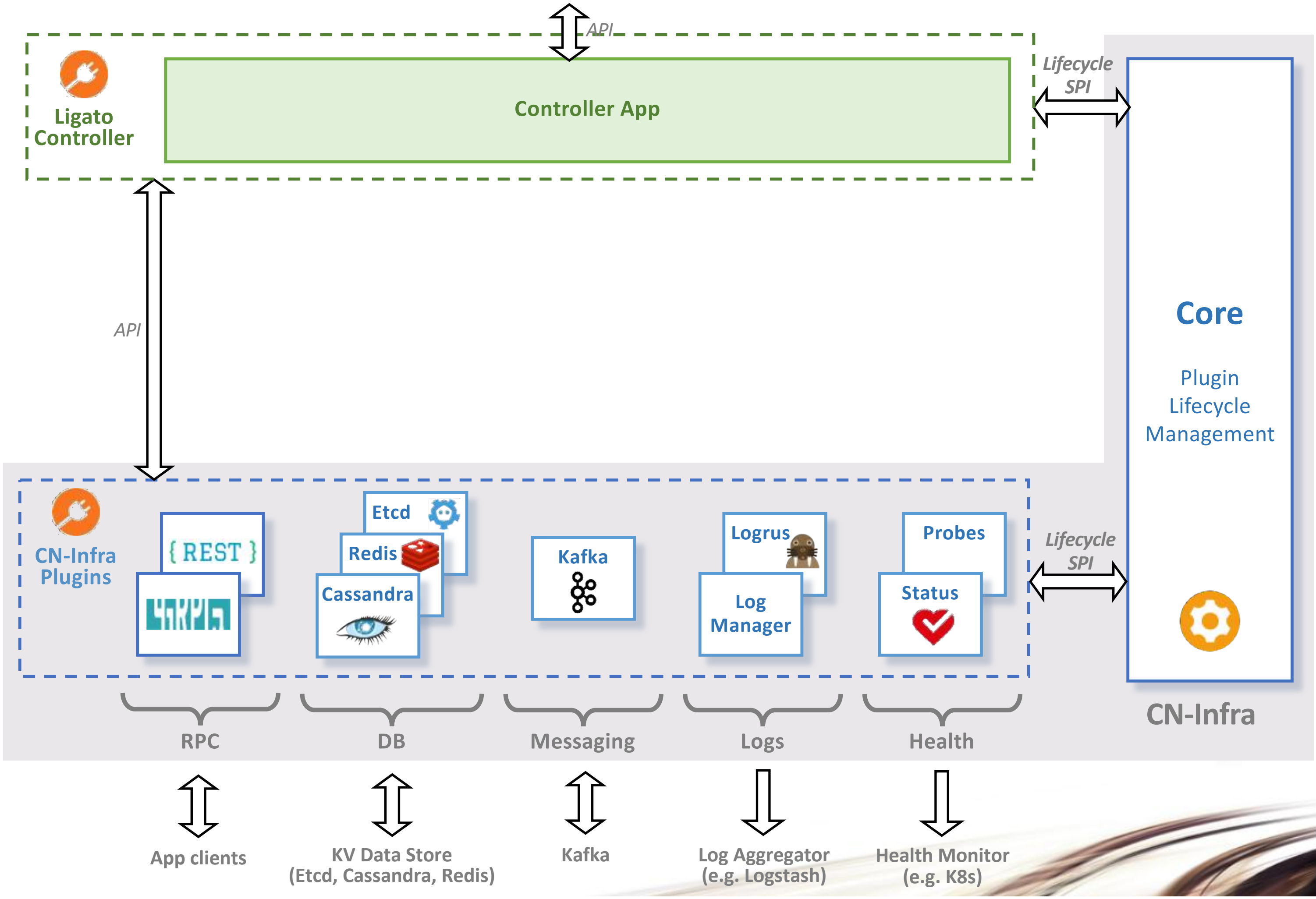
www.github.com/ligato/vpp-agent



.io

Ligato Controller: a CNF Deployment Platform

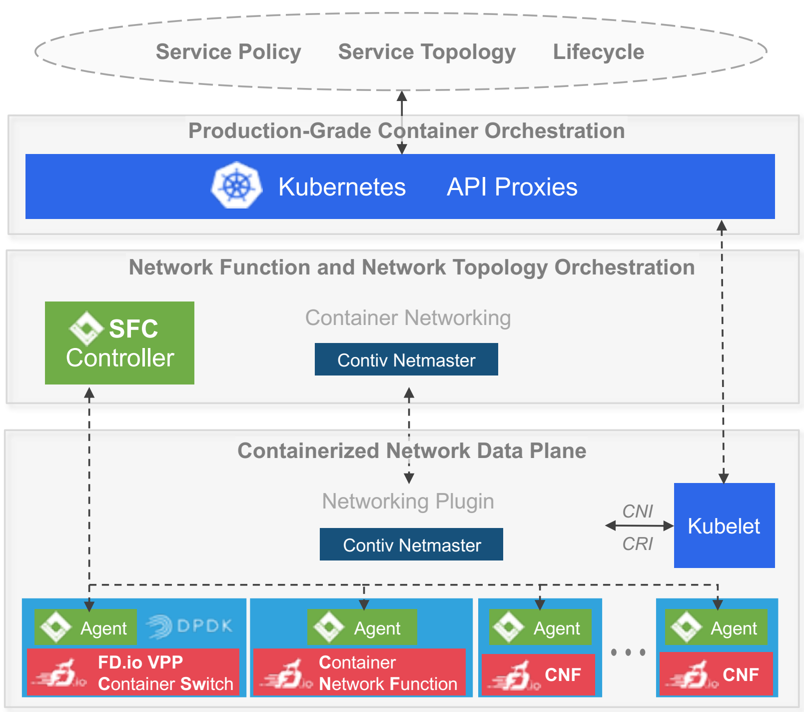
www.github.com/ligato/sfc-controller



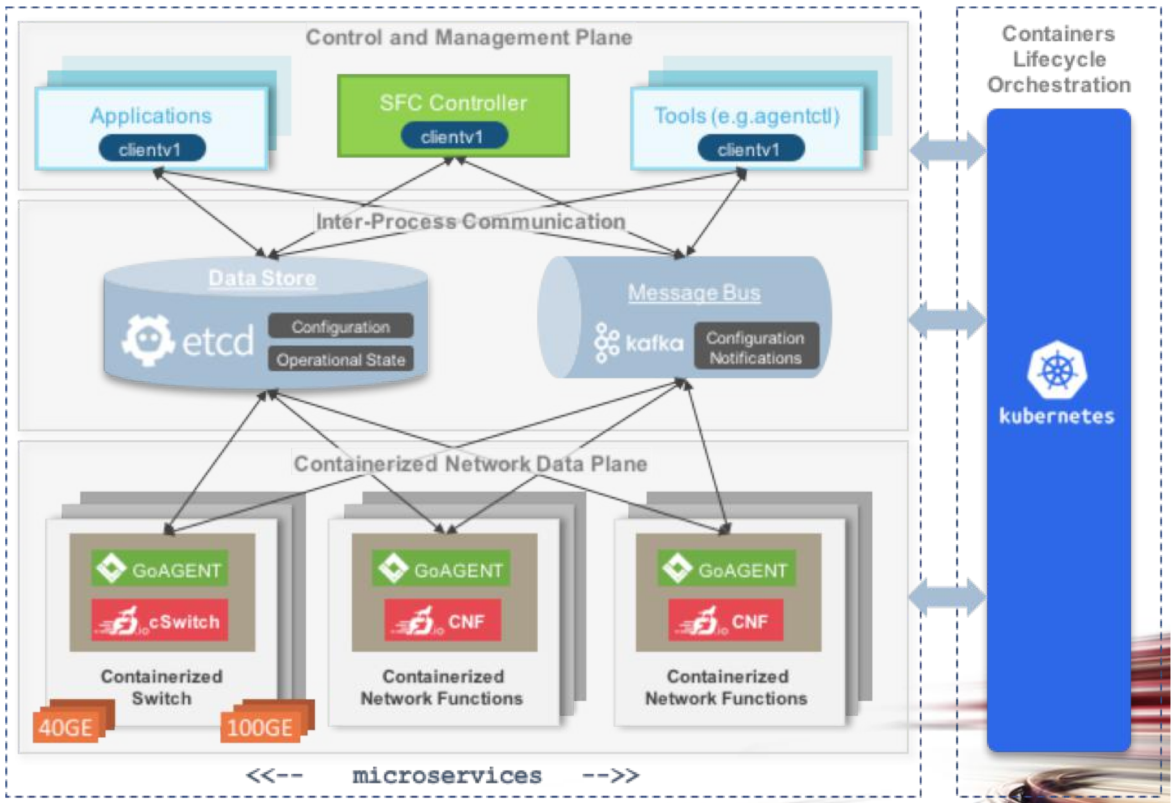
Ligato – Cloud-native Network Functions (CNF)

Putting It All Together Now – The Software Architecture

Functional Layered Diagram



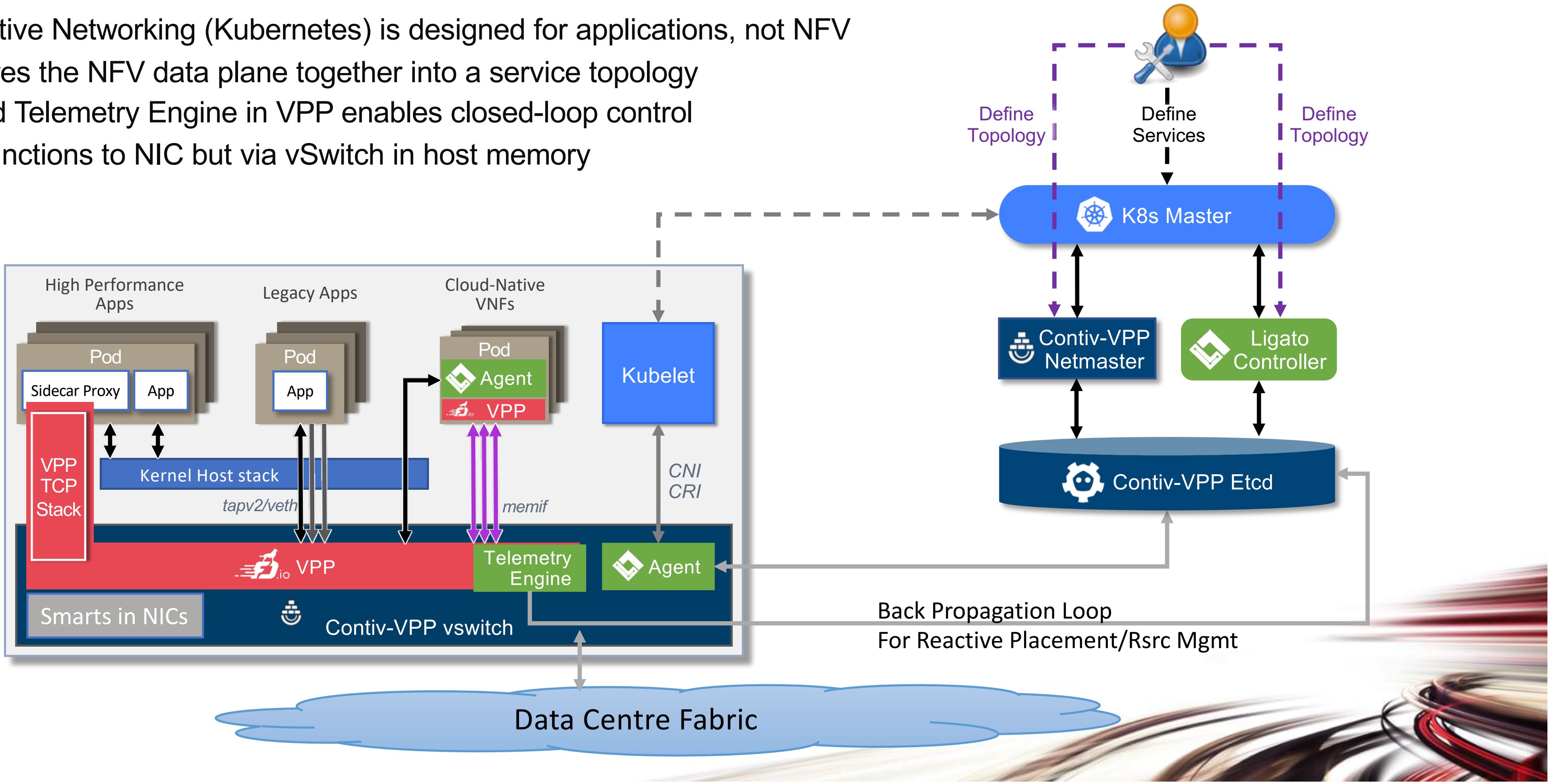
Software Architecture Diagram



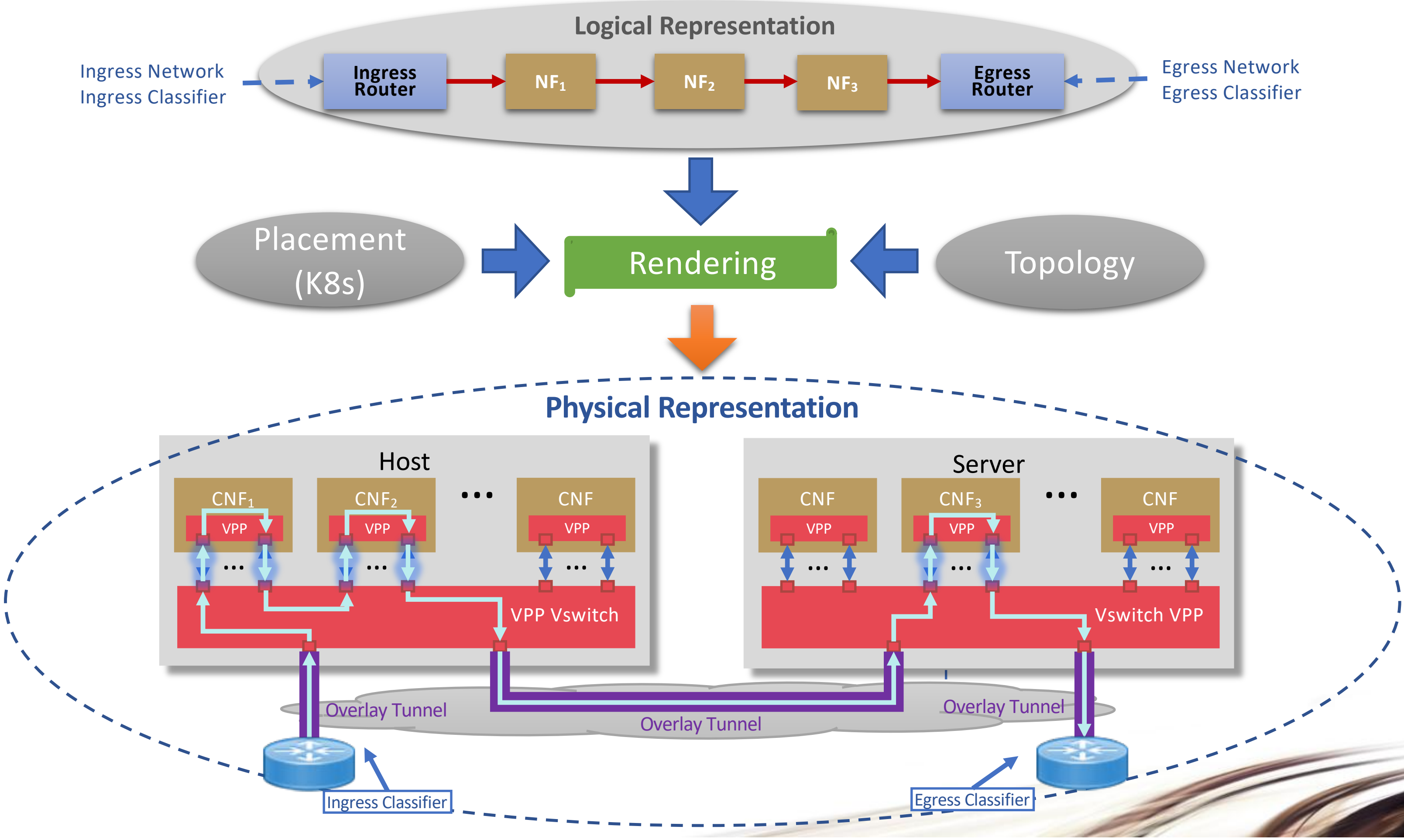
Ligato – Cloud-native Network Functions (CNF)

Putting It All Together Now – The System Architecture

- Cloud-Native Networking (Kubernetes) is designed for applications, not NFV
- Ligato wires the NFV data plane together into a service topology
- Dedicated Telemetry Engine in VPP enables closed-loop control
- Offload functions to NIC but via vSwitch in host memory



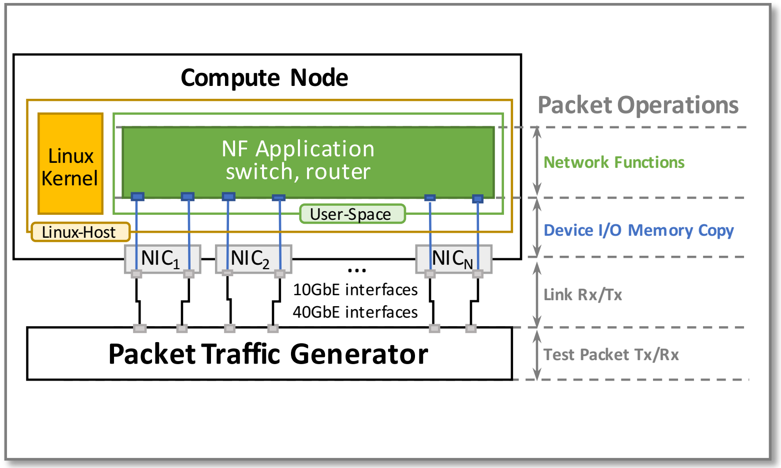
Service Function Chaining with Ligato



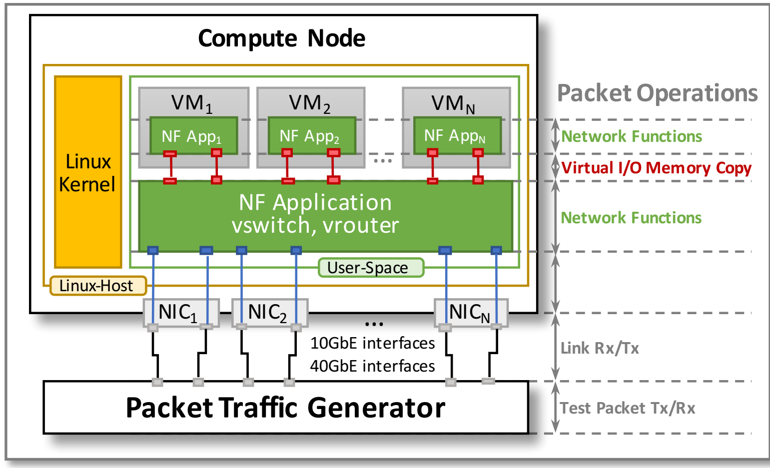
Cloud-native Network Functions

Optimising Performance within the Compute Node

Bare-metal



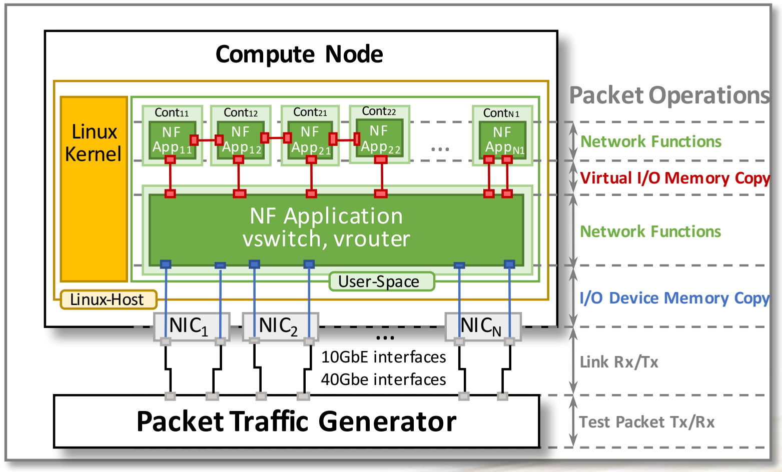
Virtual Machines



virtio
vhost

Moving “**Virtualisation**” to the **Native** Operation

Containers



memif !

Getting closer to bare-metal speeds ...

With a New **Cloud-native** Network Packet **Virtual Interface, memif**

memif – Motivation

- Create packet based shared memory interface for user-mode application
- Be container friendly (no privileged containers needed)
- Support both polling and interrupt mode operation
  - Interrupts simulated with linux eventfd infrastructure
  - Support for interrupt masking in polling mode
- Support vpp-to-vpp, vpp-to-3rd-party and 3rd-party-to-3rd-party operation
- Support for multiple queues (incl. asymmetric configurations)
- Jumbo frames support (chained buffers)
- Take security seriously
- Multiple operation mode: ethernet, ip, punt/inject
- Lightweight library for apps - allows easy creation of applications which communicate over memif

It needs to be fast, but performance is not a number 1 priority.

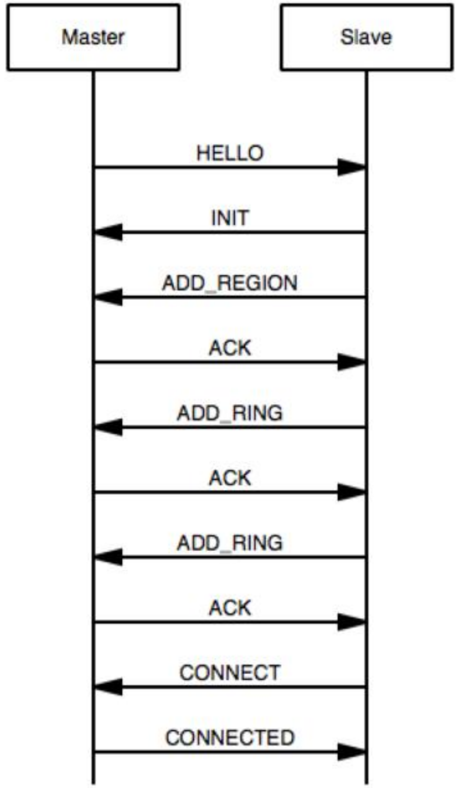
memif – Security

- Point-to-point Master/Slave concept:
  - **Master** - Never exposes memory to slave
  - **Slave** - Responsible for allocation and sharing memory region(s) to Master
  - Slave can decide if it will expose internal buffers to master or copy data into separate shared memory buffer
- Shared memory data structures (rings, descriptors) are pointer-free
- Interfaces are always point-to-point, between master-slave pair
- Shared memory is initialized on connect and freed on disconnect
- Interface is uniquely identified by unix socket filename and interface id pair
- There is optional shared secret support per interface
- Optionally master can get PID, UID, GID for each connection to socket listener

Memory copy is a MUST for security.

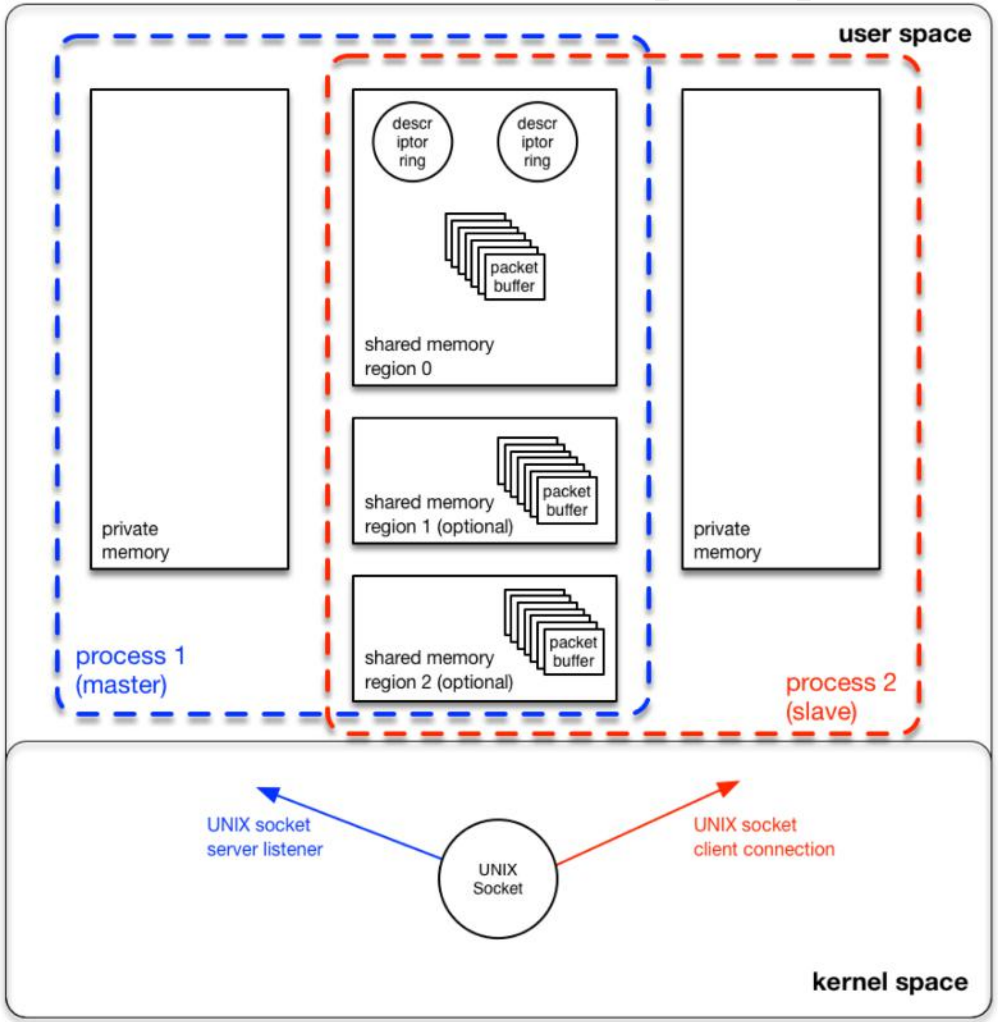
memif – Control Channel

- Implemented as Unix Socket connection (AF_UNIX)
- Master is socket listener (allows multiple connections on single listener)
- Slave connects to socket
- Communication is done with fixed size messages (128 bytes):
    - **HELLO** (m2s): announce info about Master
    - **INIT** (s2m): starts interface initialization
    - **ADD_REGION** (s2m): shares memory region with master (FD passed in ancillary data)
    - **ADD_RING** (s2m): shares ring information with master (size, offset in mem region, interrupt eventfd)
    - **CONNECT** (s2m): request interface state to be changed to connected
    - **CONNECTED** (m2s): notify slave that interface is connected
    - **DISCONNECT** (m2s, s2m): disconnect interface
    - **ACK** (m2s, s2m): Acknowledge



.io

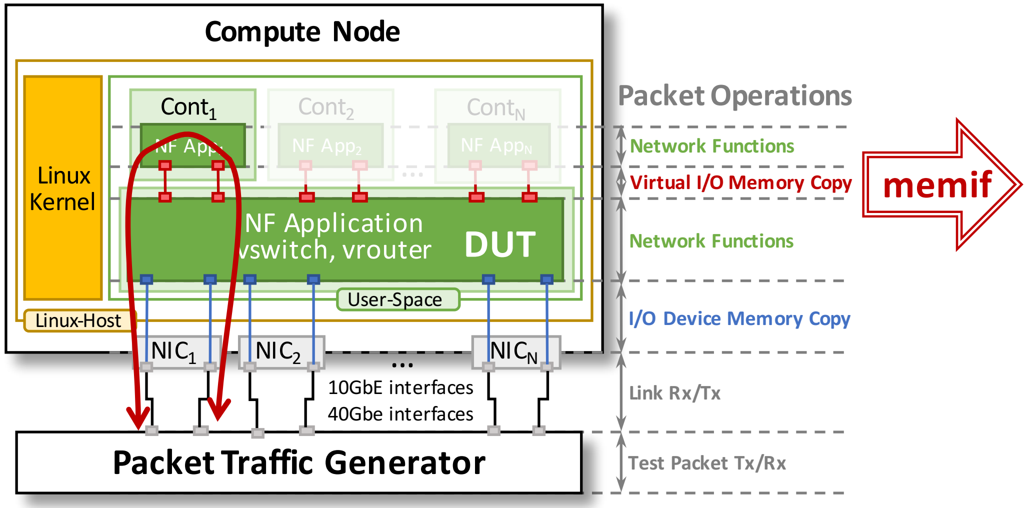
memif – Shared Memory layout

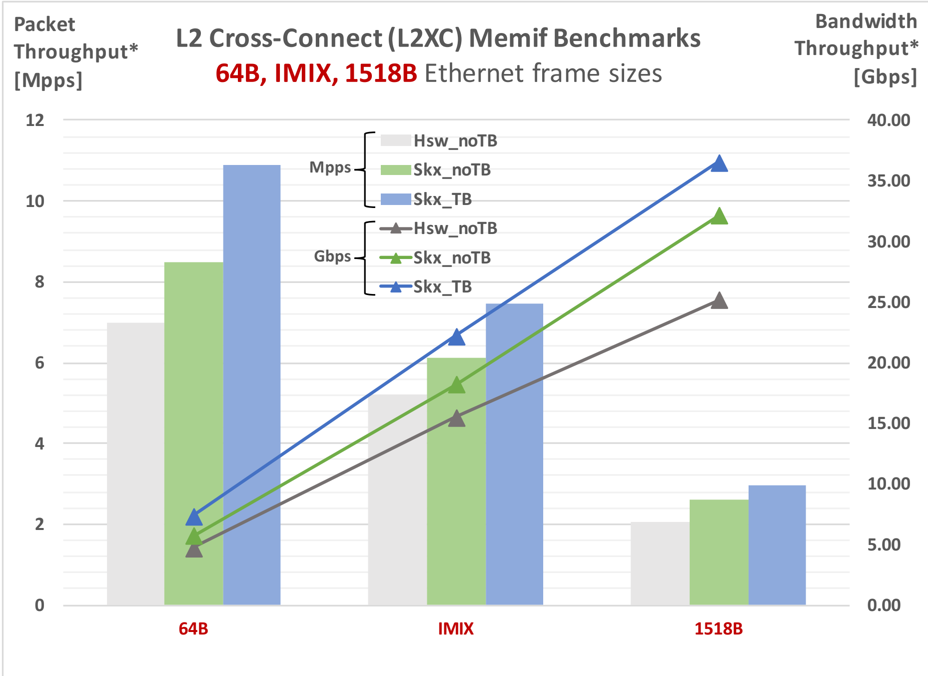


memif – Shared Memory layout

- Rings and buffers in shared memory are referenced with (region_index, offset) pair
  - Much easier to deal with SEGFAULTS caused by eventual memory corruption
- Slave shares one or more memory regions with master by passing mmap() file descriptor and region size information (ADD_REGION message)
- Slave initializes rings and descriptors and shares their location (region_index, offset), size, direction and efd with master (ADD_RING) message
- Each ring contains header and array of buffer descriptors
  - number of descriptors is always power-of-2 for performance reasons (1024 as default)
- Buffer descriptor is 16 byte data structure which contains:
  - **flags (2byte) –** space for various flags, currently only used for buffer chaining
  - **region_index (2 byte)** – memory region where buffer is located
  - **offset (4 bytes)** – buffer start offset in particular memory region
  - **length (4 byte)** – length of actual data in the buffer
  - **metadata (4 byte)** – custom use space

Memif Performance – L2





Packet Size	Packet Throughput* [Mpps]			Bandwidth Throughput* [Gbps]		
	Hsw_noTB	Skx_noTB	Skx_TB	Hsw_noTB	Skx_noTB	Skx_TB
64B	7.0	8.5	10.9	4.7	5.7	7.3
IMIX	5.2	6.1	7.5	15.5	18.2	22.2
1518B	2.0	2.6	3.0	25.2	32.1	36.5

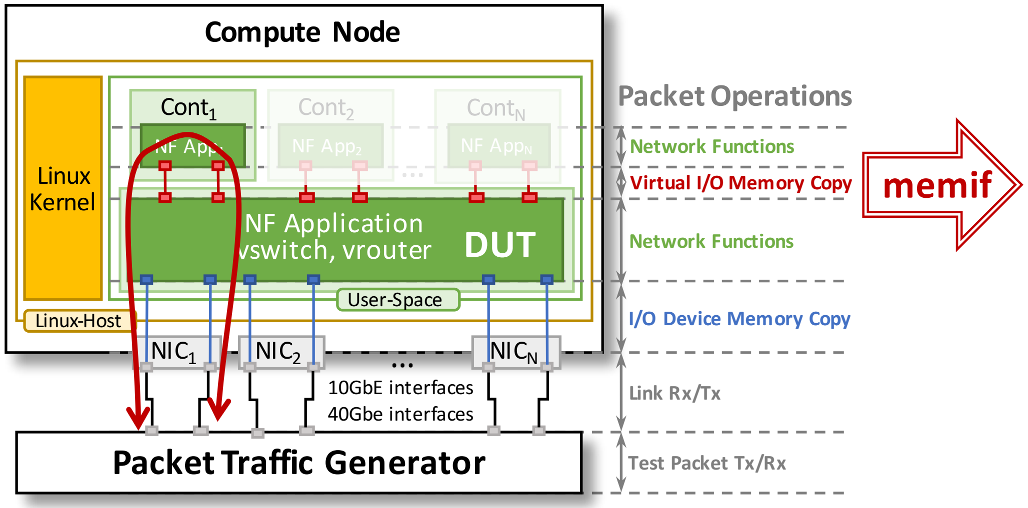
Note: packets are passing “vswitch, vrouter” DUT twice per direction, so the external throughput numbers reported in the table should be doubled to get per CPU core throughput.

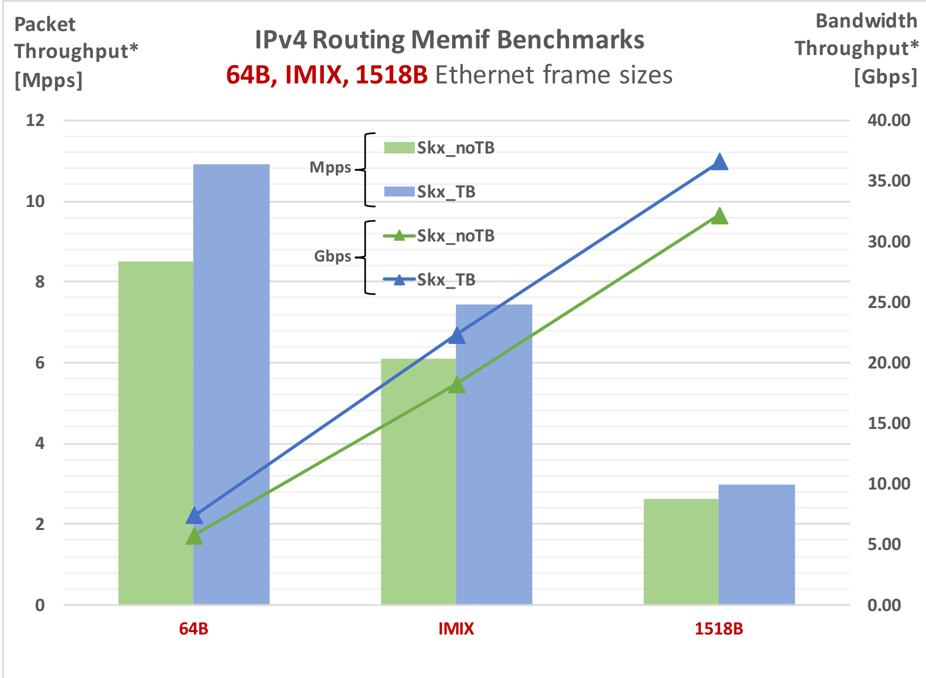
* Maximum Receive Rate (MRR) Throughput - measured packet forwarding rate under the maximum load offered by traffic generator over a set trial duration, regardless of packet loss.

Hsw – Intel Xeon® Haswell, E5-2699v3, 2.3GHz, noHT. Results scaled up to 2.5GHz and HT eanbled.
Skx – Intel Xeon® Skylake, Platinum 8180, 2.5GHz, HT enabled.
TB – TurboBoost enabled.
noTB – TurboBoost disabled

.io

Memif Performance – IPv4





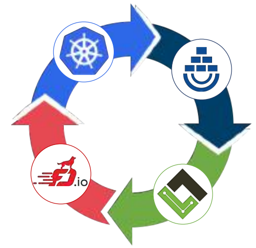
Packet Size	Packet Throughput*		Bandwidth Throughput*	
	Skx_noTB	Skx_TB	Skx_noTB	Skx_TB
64B	6.15	7.32	4.13	4.92
IMIX	4.49	5.5	13.40	16.41
1518B	2.44	2.62	30.02	32.24

* Maximum Receive Rate (MRR) Throughput - measured packet forwarding rate under the maximum load offered by traffic generator over a set trial duration, regardless of packet loss.

Note: packets are passing “vswitch, vrouter” DUT twice per direction, so the external throughput numbers reported in the table should be doubled to get per CPU core throughput.

Hsw – Intel Xeon® Haswell, E5-2699v3, 2.3GHz, noHT. Results scaled up to 2.5GHz and HT eanbled.
Skx – Intel Xeon® Skylake, Platinum 8§80, 2.5GHz, HT enabled.
TB – TurboBoost enabled.
noTB – TurboBoost disabled

Summary



- FD.io VPP enables flexible software Network Functions
  On Bare-Metal, VMs and Containers
  High-performance

- Ligato manages lifecycle and topology of CNF services
  Enables network Service Function Chaining (SFC)
  Integrated with K8s

- FD.io memif is a virtual packet interface for Apps and Containers
  Optimised for performance (Mpps, Gbps, CPP* and IPC**)
  Safe and Secure, Zero memory copy on Slave side

- Memif library for cloud-native Apps available
  Allows easy integration for communicating over memif
  Potential to become a de facto standard..

* CPP, Cycles Per Packet
** IPC, Instructions per Cycle

ONS EUROPE

OPEN NETWORKING //
Integrate, Automate, Accelerate

September 25 - 27, 2018
Amsterdam, The Netherlands

Accelerating the Development of Cloud-native C~~V~~NFs

THANK YOU !

Opportunities to Contribute

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)
- Read/Watch the Tutorials
- Join the Mailing Lists
- Join the IRC Channels
- Explore the wiki
- Join FD.io as a member

Thank you!