Building high performance network functions in VPP

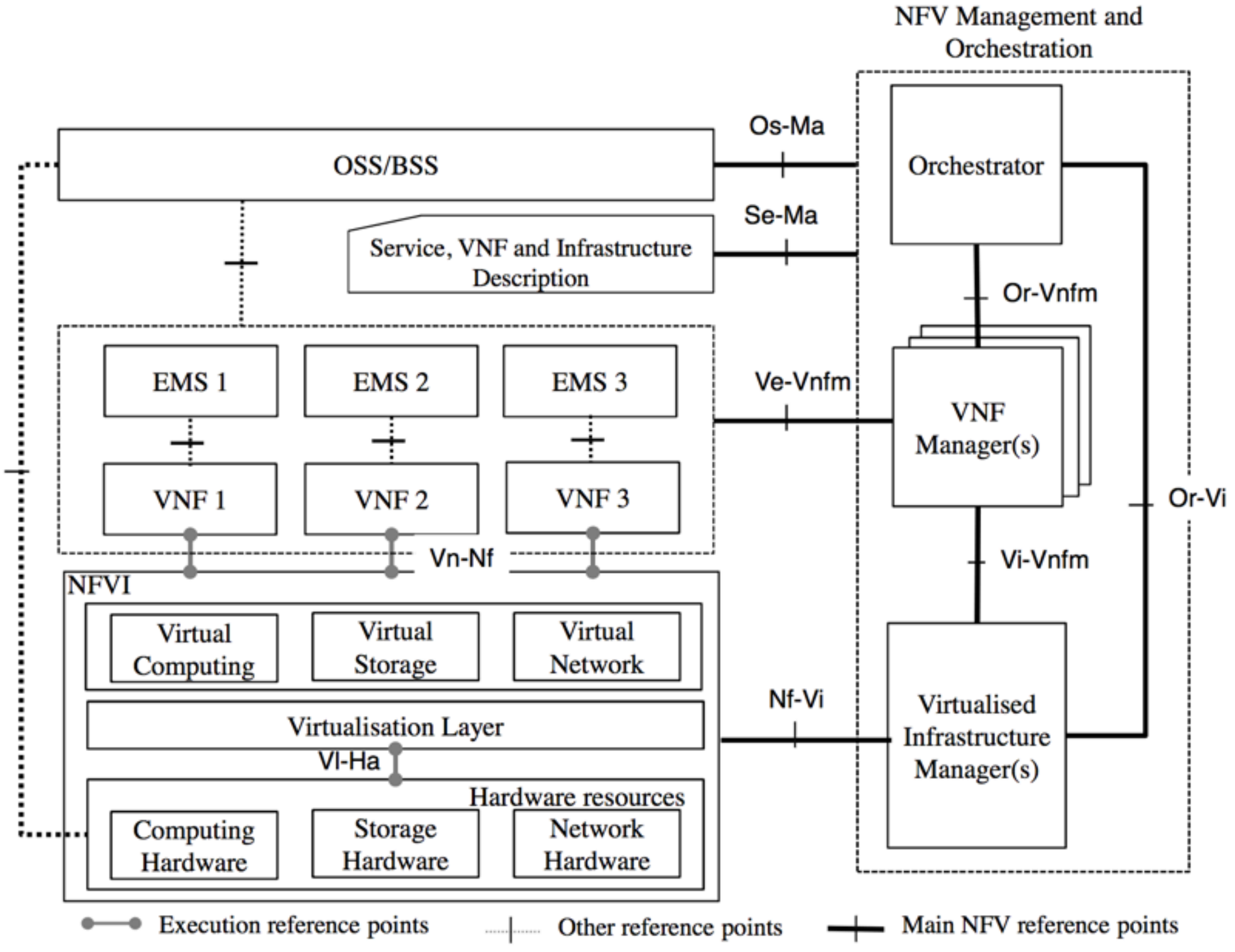
Ole Trøan, ot@cisco.com, VPP contributor

FOSDEM 2018

This talk?

- Goal: Make you into VPP developers
- Agenda:
  VPP architecture
  An example decomposed VPP VNF implementation

What's this NFV malarkey anyway?

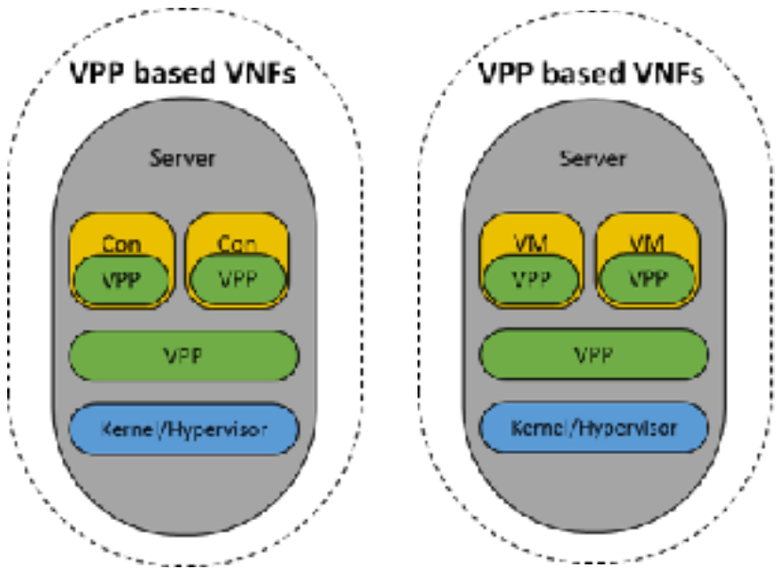

NFV Management and Orchestration
OSS/BSS
Os-Ma
Orchestrator
Se-Ma
Service, VNF and Infrastructure Description
Or-Vnfm
EMS 1
EMS 2
EMS 3
Ve-Vnfm
VNF Manager(s)
VNF 1
VNF 2
VNF 3
Or-Vi
Vn-Nf
Vi-Vnfm
NFVI
Virtual Computing
Virtual Storage
Virtual Network
Nf-Vi
Virtualised Infrastructure Manager(s)
Virtualisation Layer
Vl-Ha
Hardware resources
Computing Hardware
Storage Hardware
Network Hardware
Execution reference points
Other reference points
Main NFV reference points

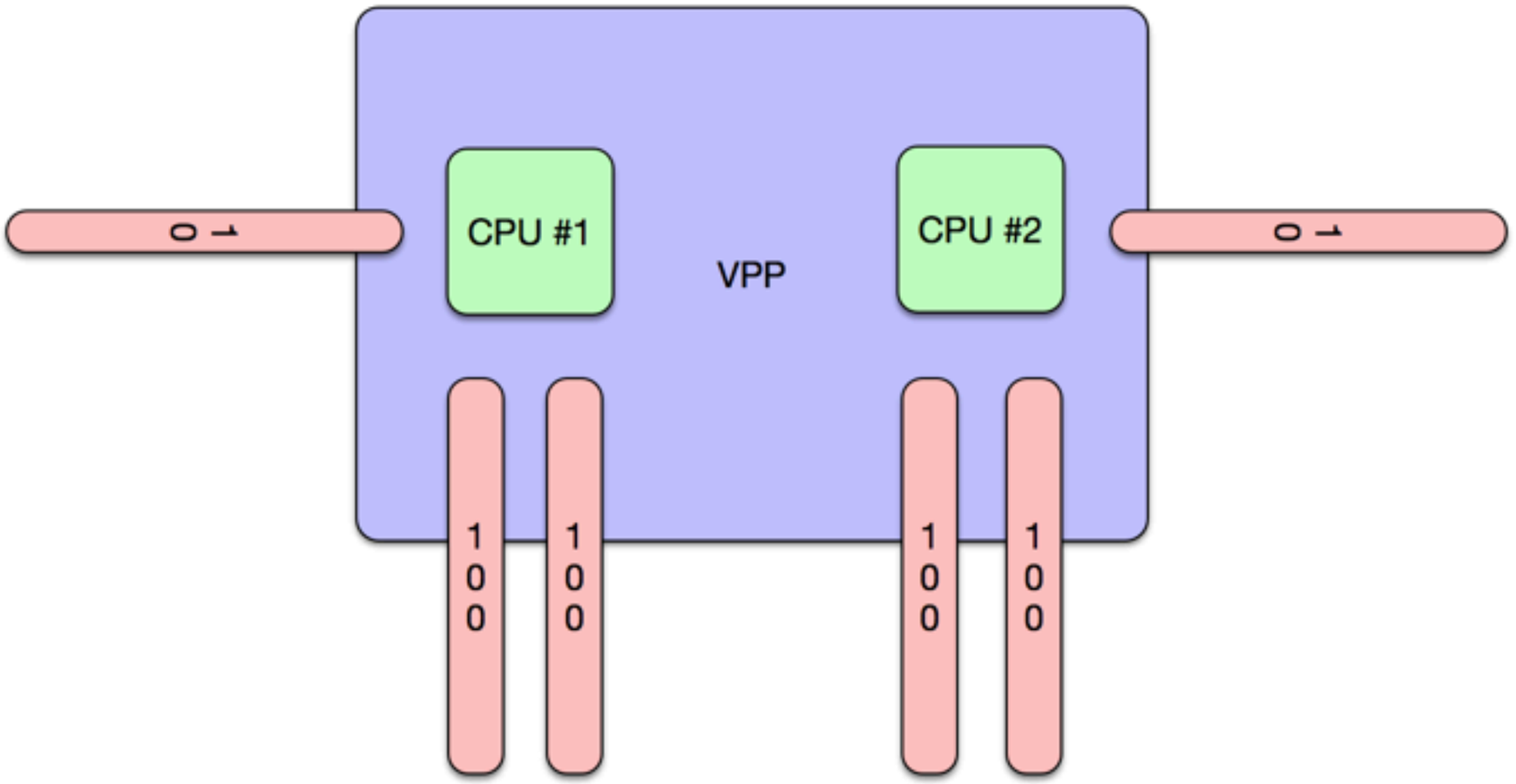
Approach #1

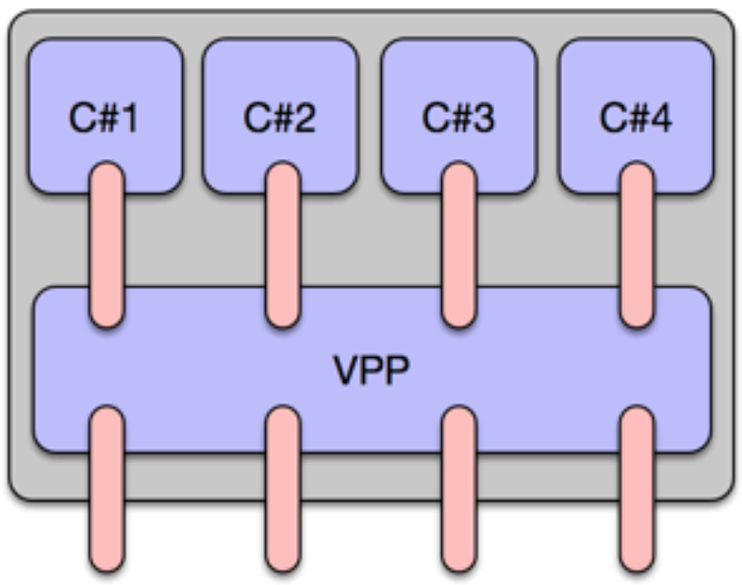
- Take your hardware appliance, port it to Linux, stuff it into a VM. Done.
- Disaggregation: Take each one of all the features on the router's datapath, stuff it into a separate VM and invent some marvellous way of chaining them all back together.

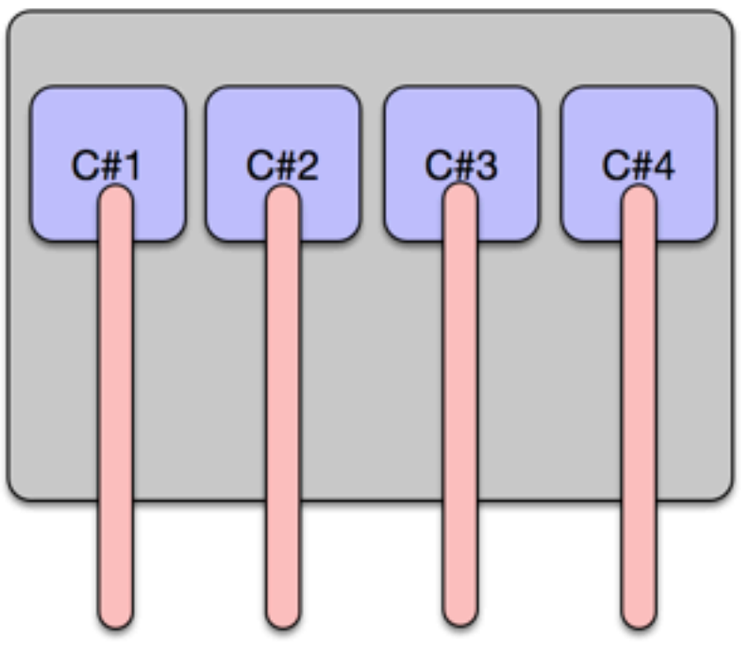
Approach #2

- Decomposition / Disaggregation
- Deployment model:
  Bare-metal, VM, container, (unikernel)
- Control plane?




0 1
CPU #1
VPP
CPU #2
0 1
1 0 0
1 0 0
1 0 0
1 0 0


C#1
C#2
C#3
C#4
VPP


C#1
C#2
C#3
C#4

What is VPP?

- A framework for building forwarding functions.
- Written in C
- Multi-core, Portable. x86, ARM, PowerPC
- Data-plane / Control-plane separation
  High performance shared memory API
  Debug CLI
- Punt path to Linux kernel / Control plane apps.
- Tracing, logging, counters
- Scheduler, lightweight processes
- Host stack with own UDP/TCP implementation
- Drivers for: AF_PACKET, TAP, MEMIF, DPDK…

VPP highlights

- Vector packet processing. As opposed to scalar packet processing. A vector of packets are processed through a graph of forwarding functions. This ensures optimal usage of the instruction cache.

- Modern, lock-less scalable data structures. A use of indices instead of pointers for direct lookup and robustness (as opposed to data structures chasing pointers). Effective use of a modern CPUs memory hierarchy and modern CPU instructions (e.g. AVX2).

- Forwarding as a graph. Where each graph node is independent and does a limited amount of work on a vector of packets. Which ensures a very high hit rate on the instructions cache.

- Plugins as first-class citizens. Allows for a completely bespoke forwarding plane, by dynamically adjusting the forwarding graph. The graph can be custom built for the application, the software image and running application need only contain the set of graph nodes used. And the graph node can be developed using VPP libraries outside or inside of the main VPP software repository.

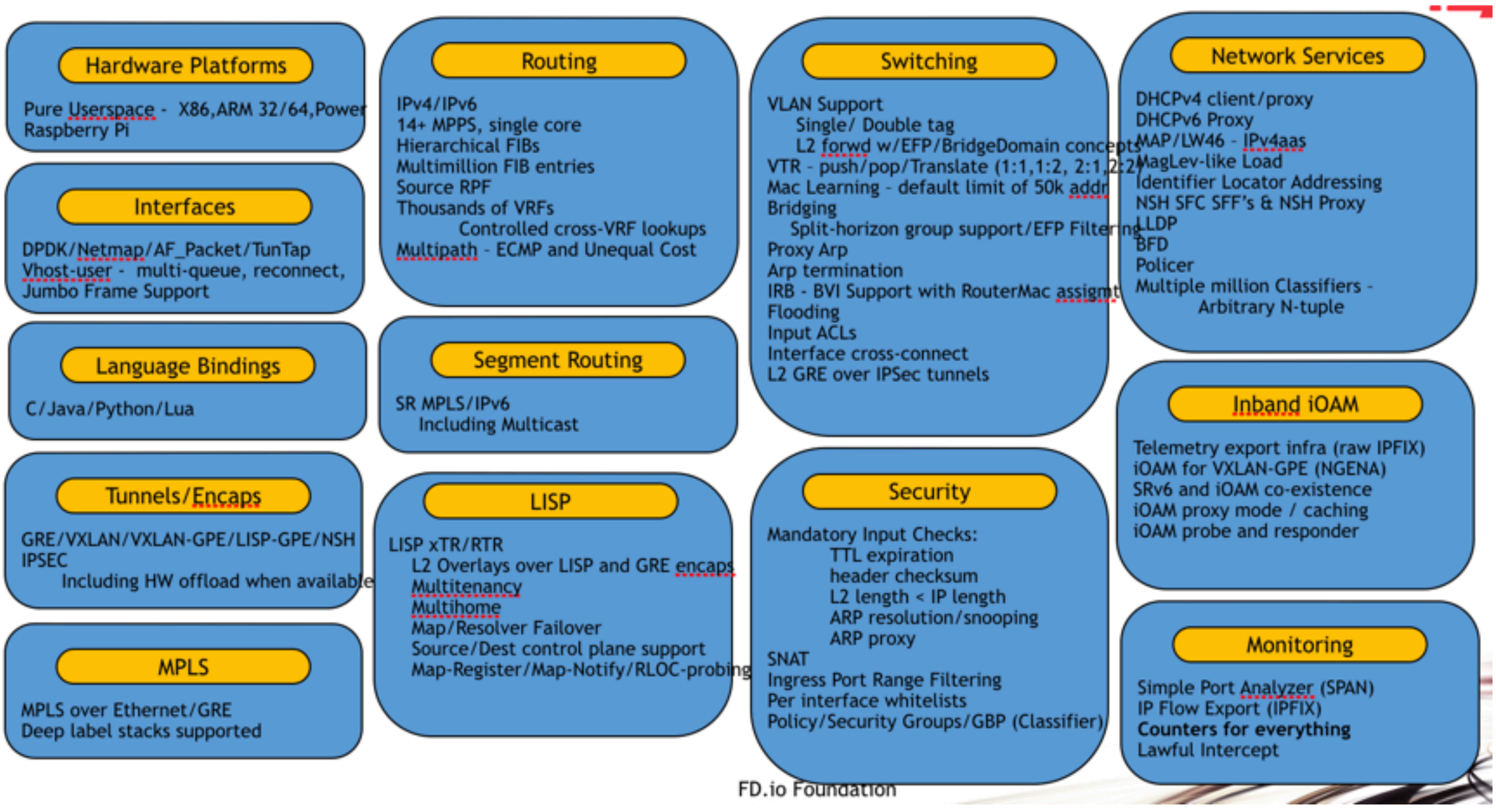
- Performance. Linear scale by number of cores. 2.8 instructions per cycle (on Intel Xeon Broadwell), very efficient use of a CPUs instruction cache and memory hierarchy.

- Robust and secure. VPP processes packets with a consistent latency and does not have a hierarchy of gradually slower and more feature complete forwarding paths (slow-path, fast-path as in IOS).

- Super programmable data plane. VPP is already integrated with ML2, ODL, ACI. It offers through its middleware agent interfaces over NETCONF/YANG, REST. It has direct language bindings in Python, Lua, Java and C. It is programmable over GRPC. Everything is done over APIs. The APIs are the first class citizens.

- New shared memory interfaces for container to container communication and from container to VPP

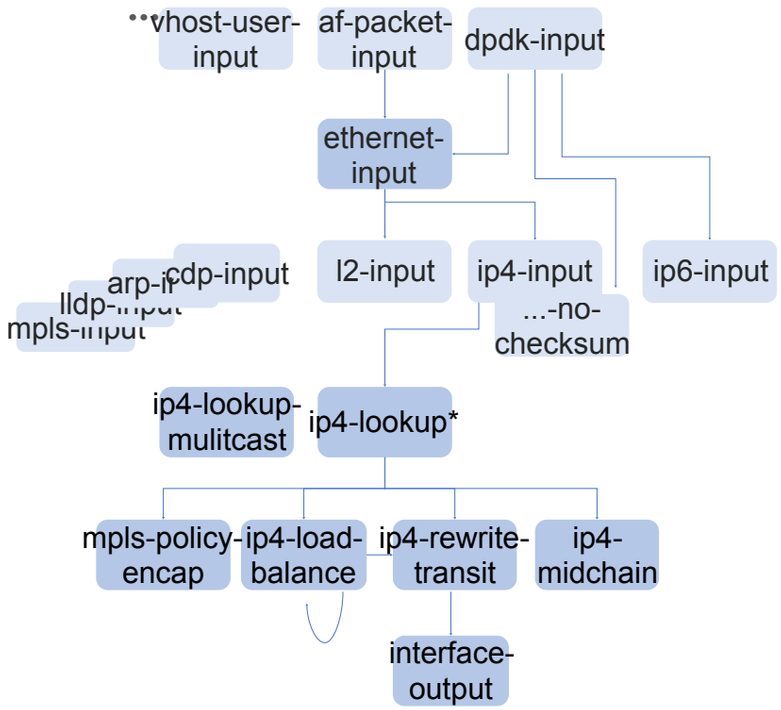
Universal Dataplane: Features



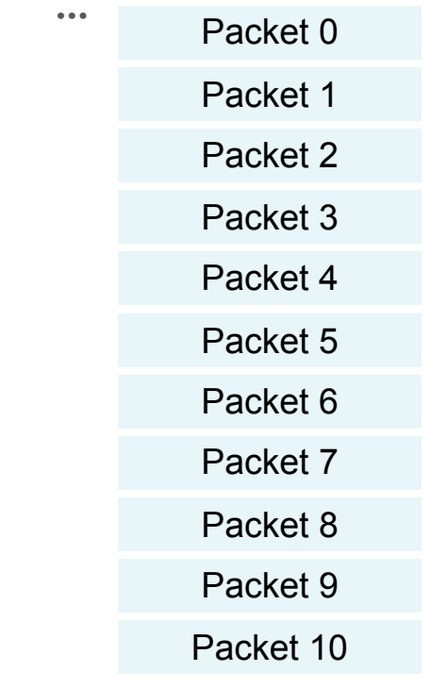
VPP – How does it work?

Compute Optimized SW Network Platform

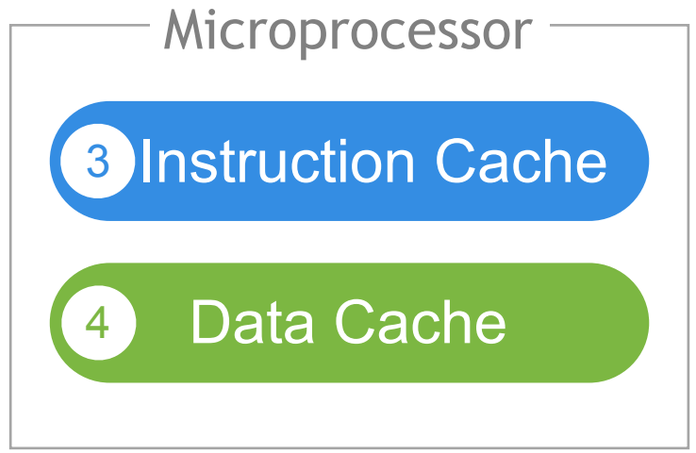
1 Packet processing is decomposed into a directed graph of nodes



2 … packets move through graph nodes in vector



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

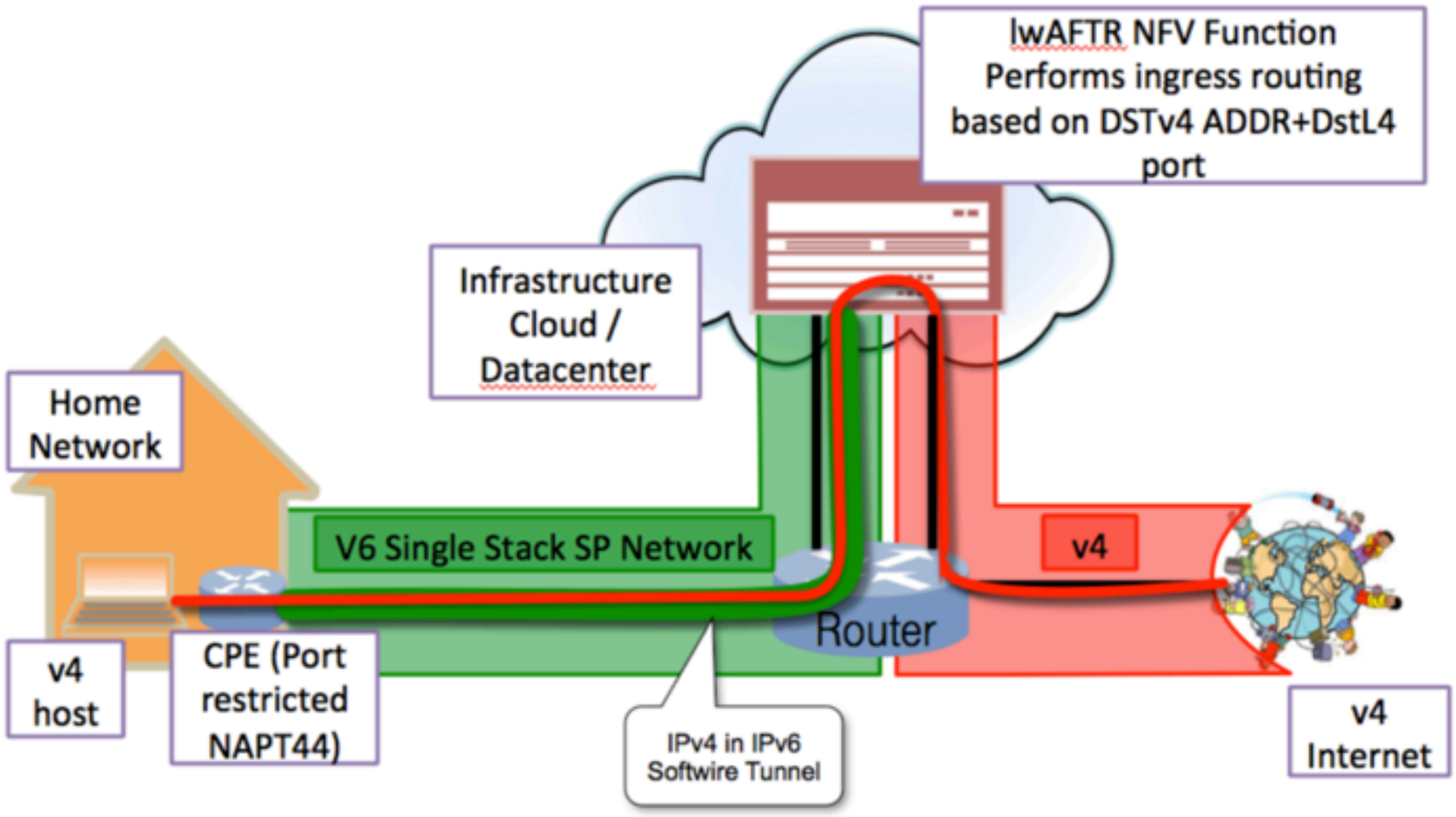


4 … packets are pre-fetched into the data cache.

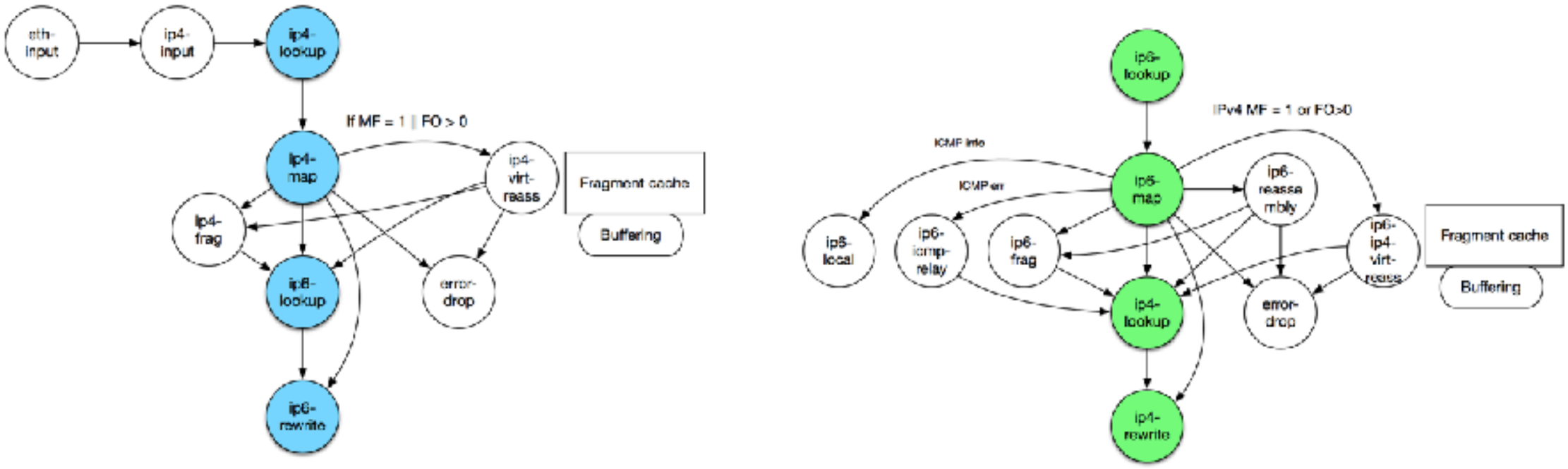
* Each graph node implements a “micro-NF”, a “micro-NetworkFunction” processing packets.

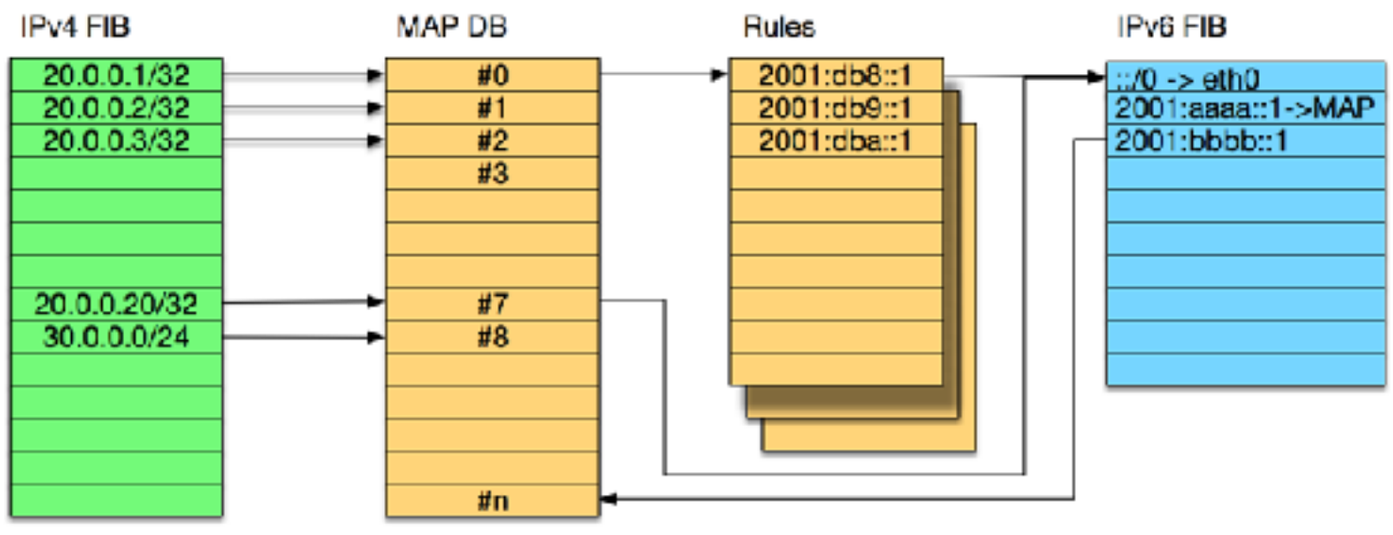
Makes use of modern Intel® Xeon® Processor micro-architectures.
Instruction cache & data cache always hot → Minimized memory latency and usage.

VNF: MAP/LW46


lwAFTR NFV Function
Performs ingress routing
based on DSTv4 ADDR+DstL4
port
Infrastructure
Cloud /
Datacenter
Home
Network
V6 Single Stack SP Network
v4
Router
v4
host
CPE (Port
restricted
NAPT44)
IPv4 in IPv6
Softwire Tunnel
v4
Internet

MAP VPP VNF


eth-input
ip4-input
ip4-lookup
If MF = 1 | FO > 0
ip4-map
ip4-virt-reass
Fragment cache
Buffering
ip4-frag
ip6-lookup
error-drop
ip6-rewrite
ip6-lookup
IPv4 MF = 1 or FO>0
ICMP info
ICMP err
ip6-map
ip6-reassembly
ip6-local
ip6-icmp-relay
ip6-frag
ip6-ip4-virt-reass
Fragment cache
Buffering
ip4-lookup
error-drop
ip4-rewrite


IPv4 FIB
MAP DB
Rules
IPv6 FIB
20.0.0.1/32
20.0.0.2/32
20.0.0.3/32
20.0.0.20/32
30.0.0.0/24
#0
#1
#2
#3
#7
#8
#n
2001:db8::1
2001:db9::1
2001:dba::1
::/0 -> eth0
2001:aaaa::1->MAP
2001:bbbb::1

Done. Questions?

- Code: https://github.com/FDio/vpp

- MAP VNF: 100s of line of forwarding code. 1000000s line of control plane.

- Does it need to be so darn complicated?