FOSDEM'20

Rethinking kubernetes networking with SRv6 and Contiv-VPP

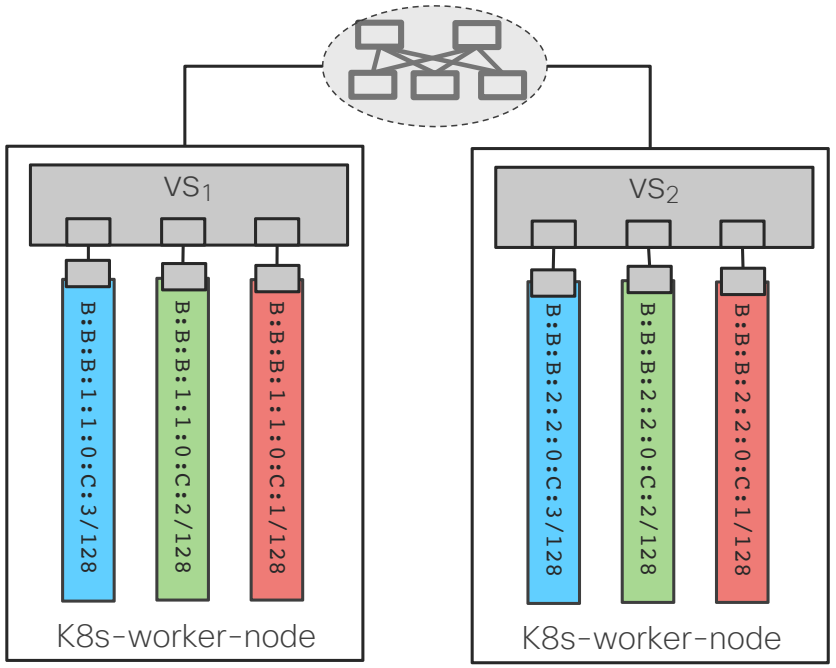
Ahmed Abdelsalam, Cisco Systems; **Daniel Bernier,** Bell Canada; **Rastislav Szabo, Filip Gschwandtner,** Pantheon.tech; **Miroslaw Walukiewicz,** Intel

Agenda

- Kubernetes networking
- SRv6
  - Introduction to SRv6
  - Kubernetes networking with SRv6
- Contiv-VPP
  - Introduction to Contiv-VPP
  - SRv6 support in Contiv-VPP
- Accelerating SRv6 with Intel N3000 smartNIC

Kubernetes networking (1)

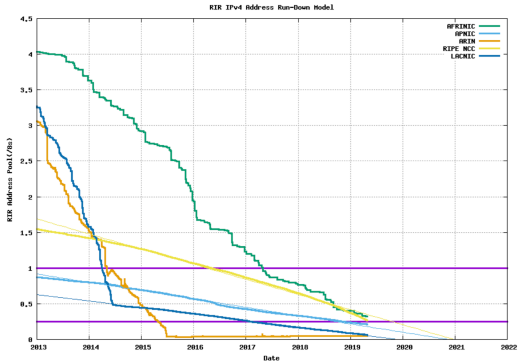
- Kubernetes does not provide any solution for handling containers networking
  - It offloads networking to third-party certified plugins called CNI plugins
- Connectivity
  - Create an interface inside the pod
  - Connect the pod interface to the fabric
  - Allocate the Pod IP
- Reachability
  - Make Pod IP reachable by the whole cluster.



Kubernetes networking (2)

- Problem statement
  - All your Containers need IP addresses
  - We do not have more enough IPv4 addresses
- Solution
  - IPv6

What about IPv4 Address Exhaustion?



RIR Address Pool runout projections (as of April 2019):

ARIN	– no free pool left
AFRINIC	– May 2020
LACNIC	– November 2019
APNIC	– November 2020
RIPE NCC	– January 2020

https://ripe78.ripe.net/presentations/39-2019-05-23-bgp2018.pdf

Pinned Tweet

RIPE NCC @ripencc · Nov 25, 2019

Today, at 15:35, we made our final /22 IPv4 allocation from the last remaining addresses in our available pool. We have now run out of IPv4 addresses.

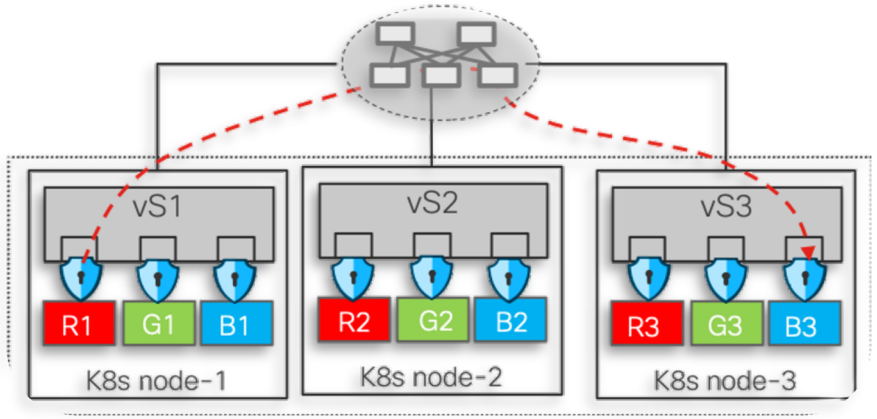
Read our full announcement here:
ripe.net/publications/n...

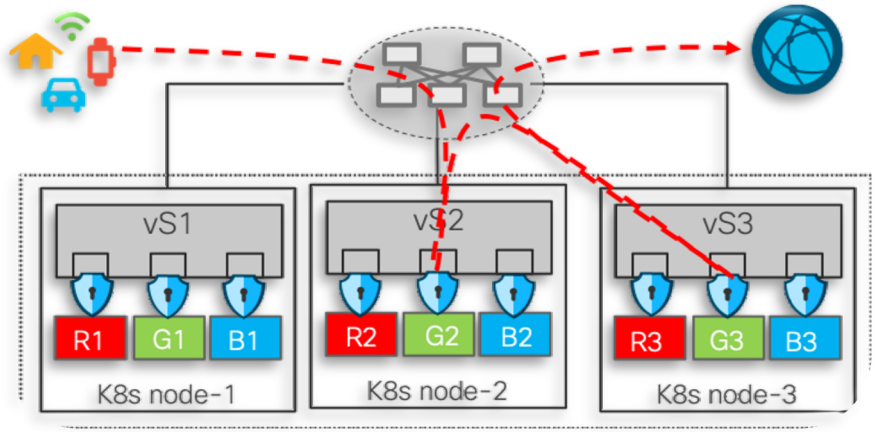
In the picture, the Registration Services team at the RIPE NCC

https://twitter.com/ripencc/status/1198977232452145152

Kubernetes networking (3)

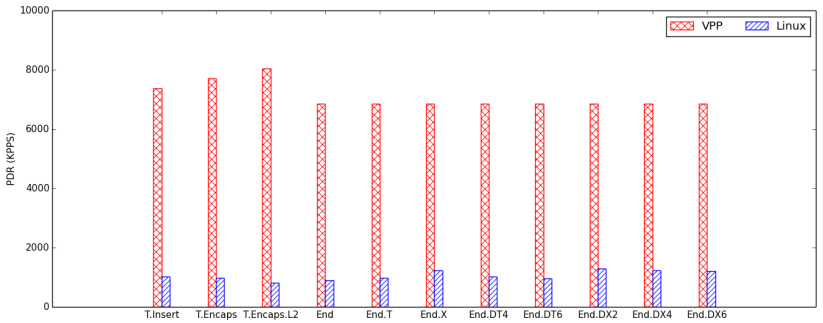
- Problem statement
  - Pod-to-Pod
  - Network policy
  - Kubernetes services
  - Ingress
  - Service chaining
  - Inter-cluster, hybrid cloud, multi-cloud, …
- Solution
  - SRv6



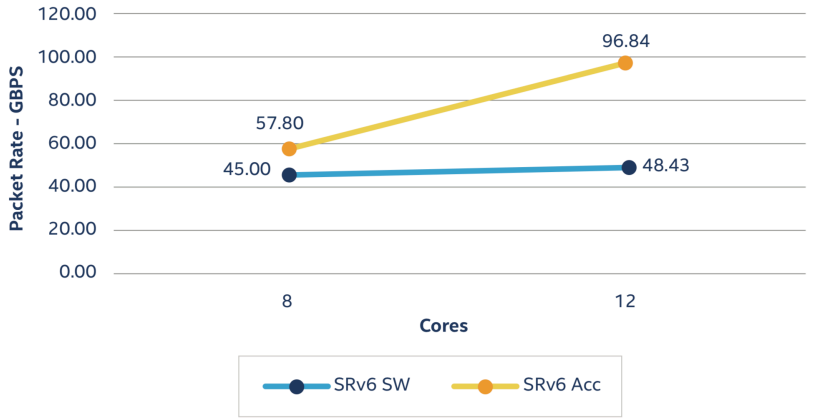


Kubernetes networking (4)

- Problem statement
  - Dataplane for fast packet I/O
    - > Kernel forwarding
    - > XDP
    - > VPP
- Solution
  - VPP
  - smartNIC (accelerated VPP)



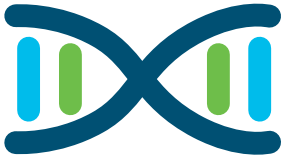
https://arxiv.org/pdf/2001.06182v1.pdf



https://www.intel.la/content/dam/www/programmable/us/en/pdfs/literature/wp/wp-01295-hcl-segment-routing-over-ipv6-acceleration-using-intel-fpga-programmable-acceleration-card-n3000.pdf

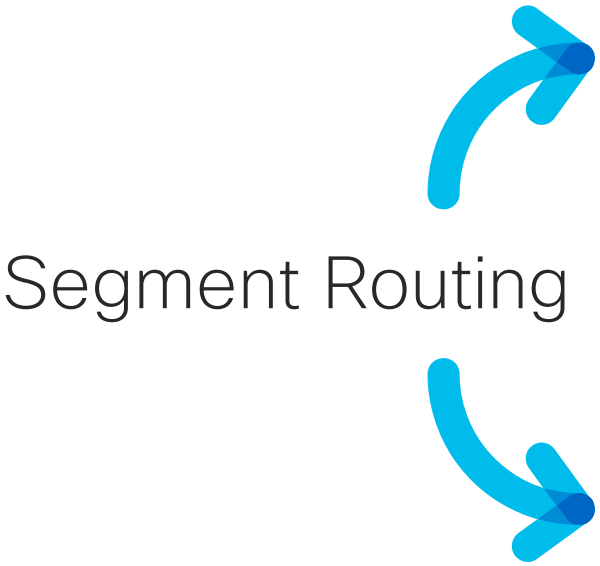
SRv6

Segment Routing



- Source Routing
  - A node steers a packet through an ordered list of instructions, called "segments".
  - Each segment has a segment identifier (SID) based on the dataplane instantiation
  - the topological and service (NFV) path is encoded in packet header
- Scalability
  - the network fabric does not hold any per-flow state for TE or NFV
- Simplicity
  - automation: TILFA sub-50msec FRR
  - protocol elimination: LDP, RSVP-TE, NSH, VXLAN...
- End-to-End
  - DC, Metro, WAN

Two dataplane instantiations



MPLS – SRMPLS

- leverage the mature MPLS HW with only SW upgrade
- 1 SID = 1 MPLS label
- SID list = MPLS label stack

IPv6 – SRv6

- leverages RFC8200 provision for source routing extension header
- 1 SID = 1 IPv6 address
- defines a new IPv6 extension header, called SRH.
- SID list = an address list in the SRH

SRv6 Ecosystem

Network Equipment Manufacturers

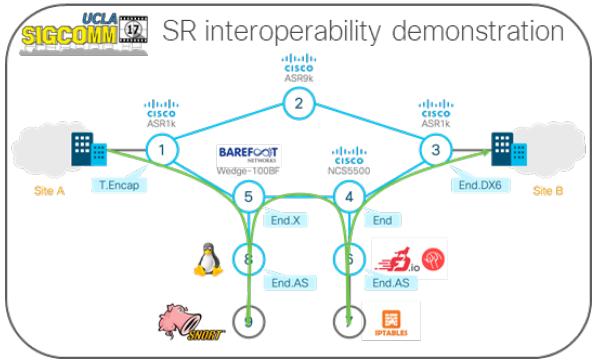
Merchant Silicon

Open-Source Applications

Open-Source Networking Stacks

Smart NIC

NFV Partners



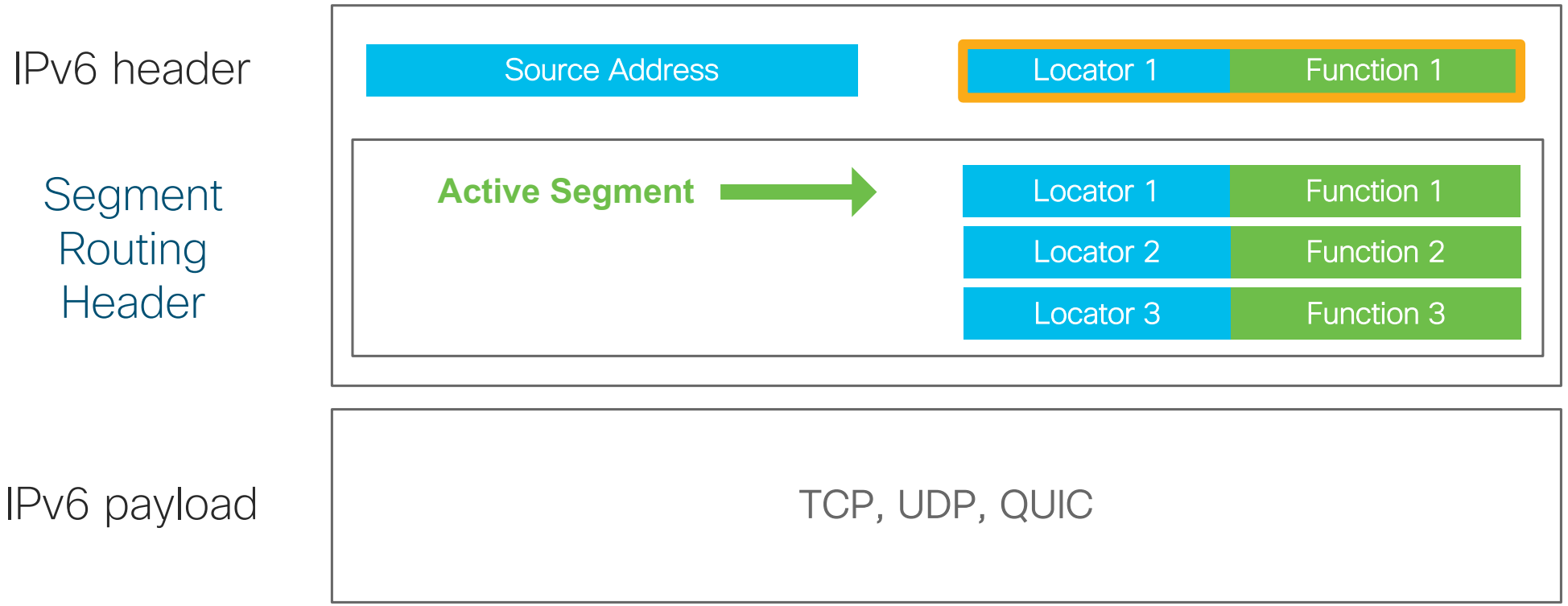
SRv6 Network programming

- The SRv6 Network Programming framework enables a network operator or an application to specify a packet packet processing program by encoding a sequence of instructions in the IPv6 packet header.
- Each instruction is implemented on one or several nodes in the network and identified by an SRv6 Segment Identifier in the packet.
- IETF standardization in progress
  - https://tools.ietf.org/html/draft-ietf-spring-srv6-network-programming-08

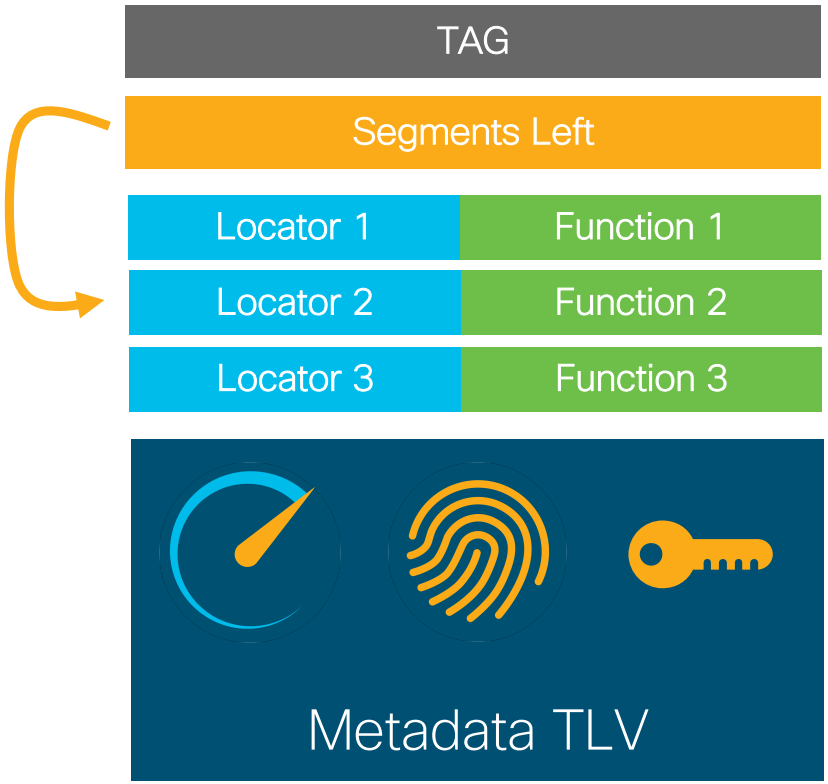
Network instruction

- 128-bit SRv6 SID
  - Locator: routed to the node performing the function
  - Function: any possible function
    either local to NPU or app in VM/Container
  - Flexible bit-length selection

Network Program in the Packet Header



SRv6 Header




```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Next Header   |  Hdr Ext Len  | Routing Type  | Segments Left |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Last Entry   |     Flags     |              Tag              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|            Segment List[0] (128 bits IPv6 address)            |
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                                                               |
                               ...
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|            Segment List[n] (128 bits IPv6 address)            |
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
//                                                             //
//         Optional Type Length Value objects (variable)       //
//                                                             //
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

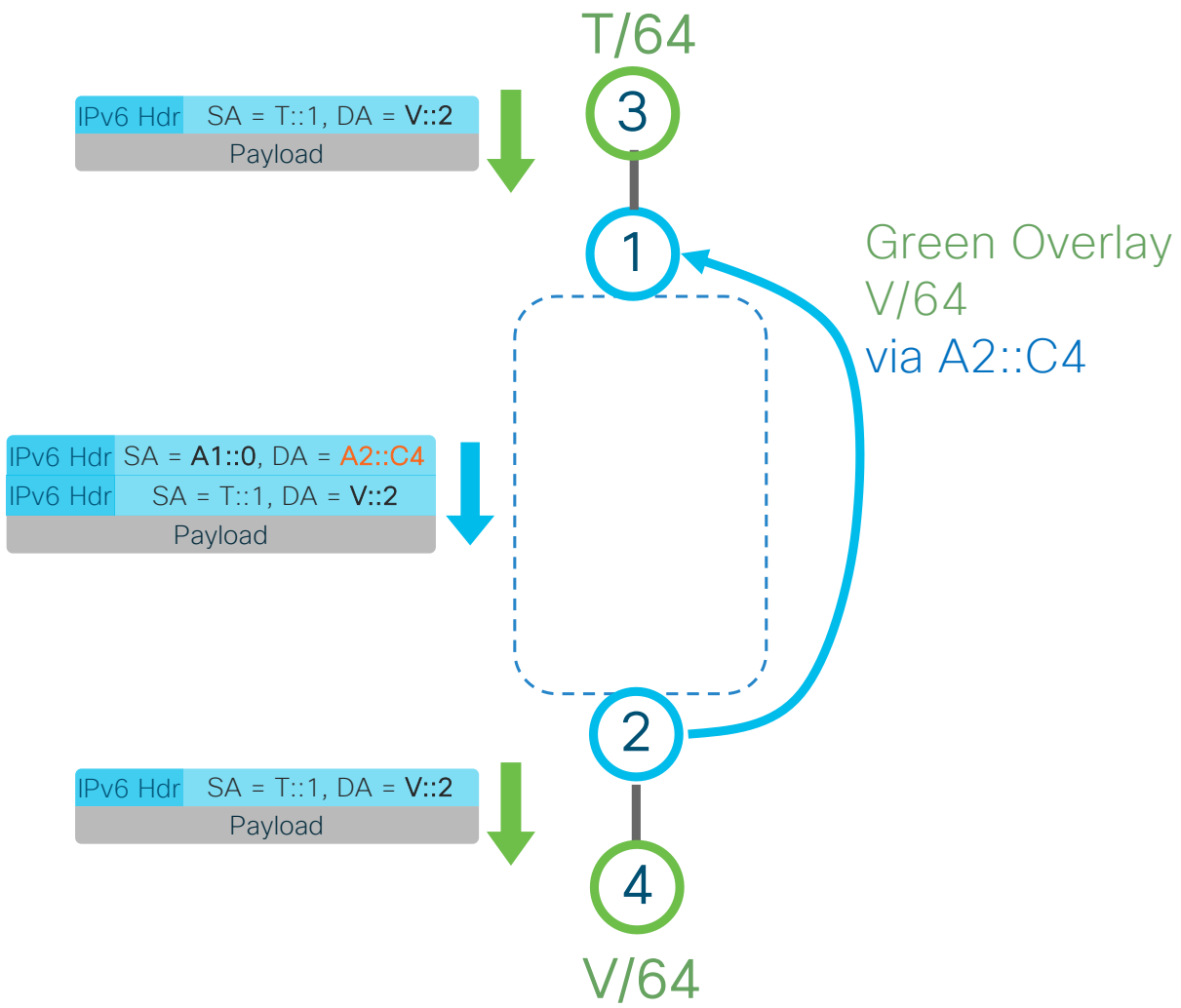
SRv6 behaviors specs summary

Headend	Behavior	Use-case
H.Encaps	SR Headend with Encapsulation in an SRv6 Policy	L3 Traffic
H.Encaps.L2	H.Encaps Applied to Received L2 Frames	L2 traffic

Endpoint	Behavior	Use-case
End	Endpoint	TE (underlay)
End.X	Endpoint with Layer-3 cross-connect	
End.DX6	Endpoint with decapsulation and IPv6 cross-connect	IPv6 L3VPN (overlay)
End.DT6	Endpoint with decapsulation and specific IPv6 table lookup	
End.DX4	Endpoint with decapsulation and IPv4 cross-connect	IPv4 L3VPN (overlay)
End.DT4	Endpoint with decapsulation and specific IPv4 table lookup	
End.DX2	Endpoint with decapsulation and Layer-2 cross-connect	L2VPN (overlay)
End.AS	Endpoint to SR-unaware APP via static proxy	Service chaining
End.AD	Endpoint to SR-unaware APP via dynamic proxy	
End.AM	Endpoint to SR-unaware APP via masquerading proxy	

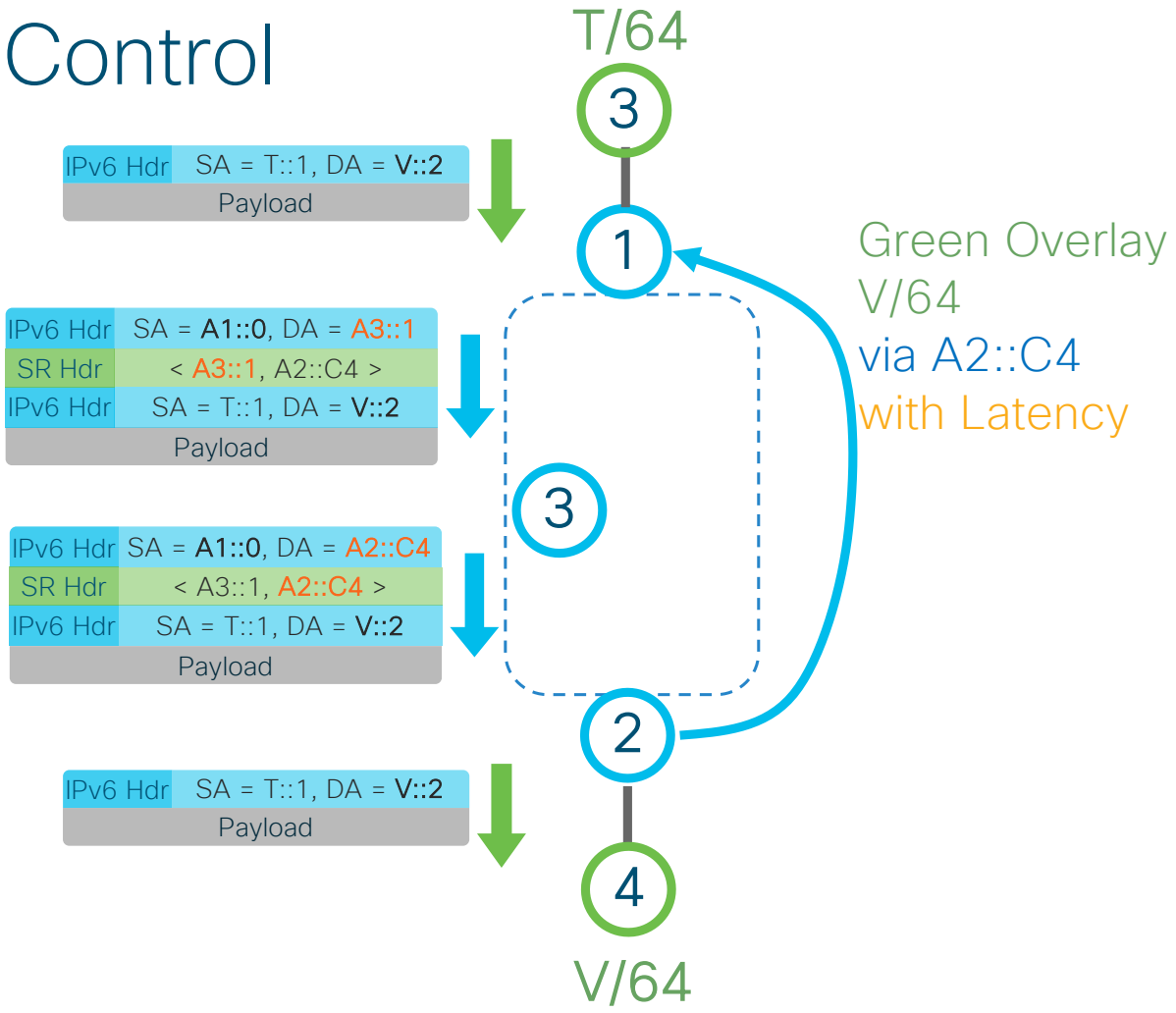
Overlay

- Automated
  - No tunnel to configure
- Simple
  - Protocol elimination
- Efficient
  - SRv6 for everything



Overlay with Underlay Control

- SRv6 does not only eliminate unneeded overlay protocols
- SRv6 solves problems that these protocols cannot solve



Kubernetes networking with SRv6

kubernetes networking (currently)

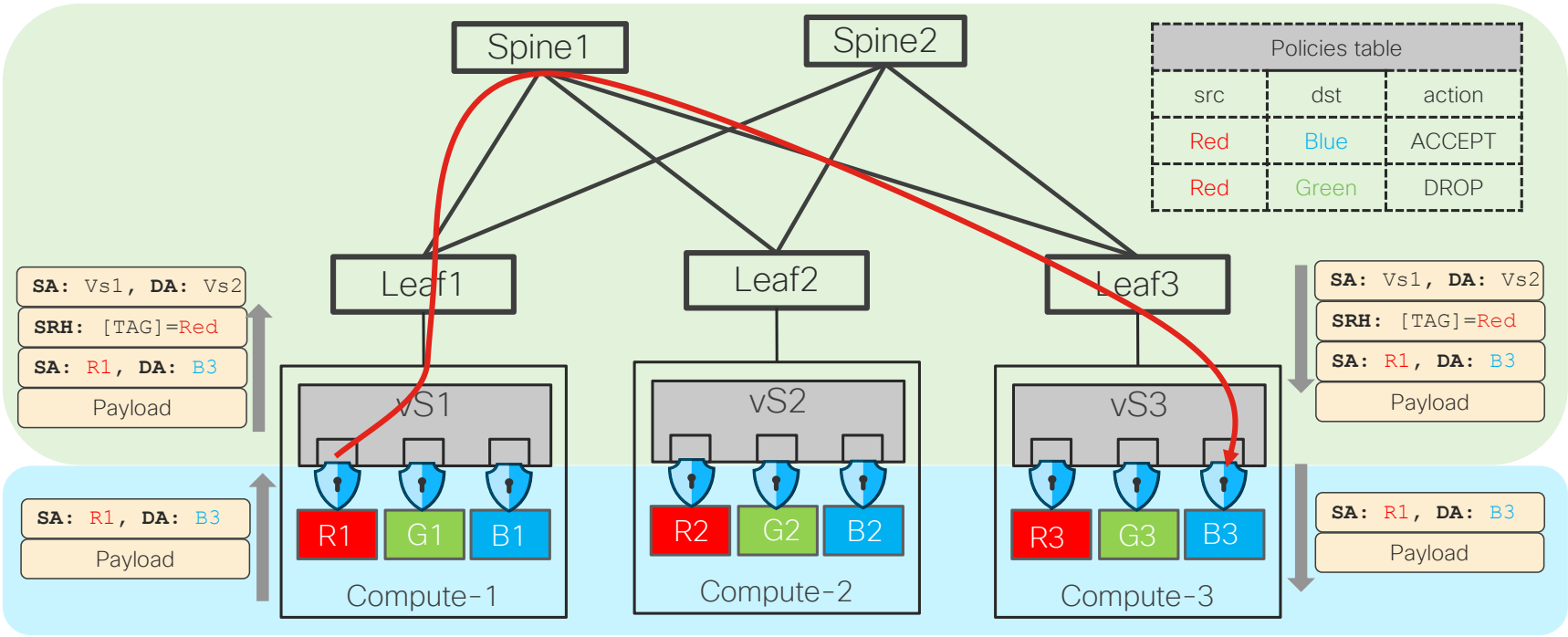
- CNI plugins are responsible for networking in kubernetes
  - Load Balancing → `Linux iptables NAT / VPP NAT`
  - Port Forwarding → `Linux iptables NAT / VPP NAT`
  - Network Policy → `Linux iptables firewall/VPP ACLs`
  - Overlay networking → `VXLAN/IP-in-IP/GENEVE/GRE/...`
  - Service chaining → `stitching of interfaces/VXLAN tunnels`
- The result
  - NAT everywhere
  - Complex network policy model that relies on container IPs
  - iptables everywhere which uses non scalable linear search matching
  - Service chaining is very complex “nearly impossible”
  - Inter-cluster communication, hybrid cloud, multi-cloud, network wide policy ???

kubernetes networking (IPv6 + SRv6)

- IPv6 for reachability
- SRv6 for everything
  - Overlay with no extra protocols → `SRv6 Encap + Decap`
  - Scalable network policy model → `Leveraging SRH TAG`
  - Port forwarding → `An IPv6 address per application`
  - Load Balancing → `One SR policy + multiple SID lists`
  - Service chaining → `Out-of-box using the SRH SID list`
  - Inter-cluster, hybrid cloud, multi-cloud, … → `SRv6 + NSM`

Network policy using SRv6

- Scalable policy table
- Fully integrated with the overlay
- Independent of container IP's

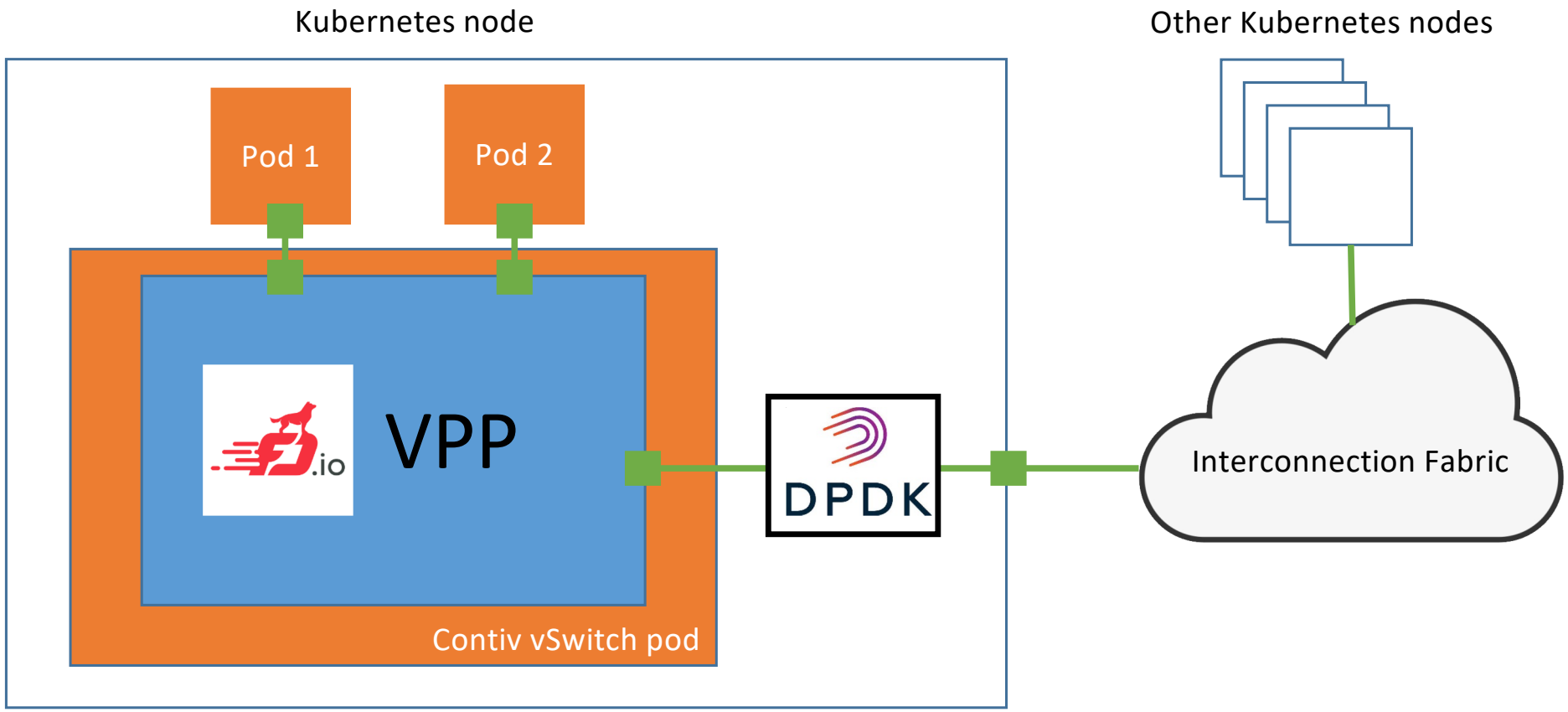


Contiv-VPP CNI Intro

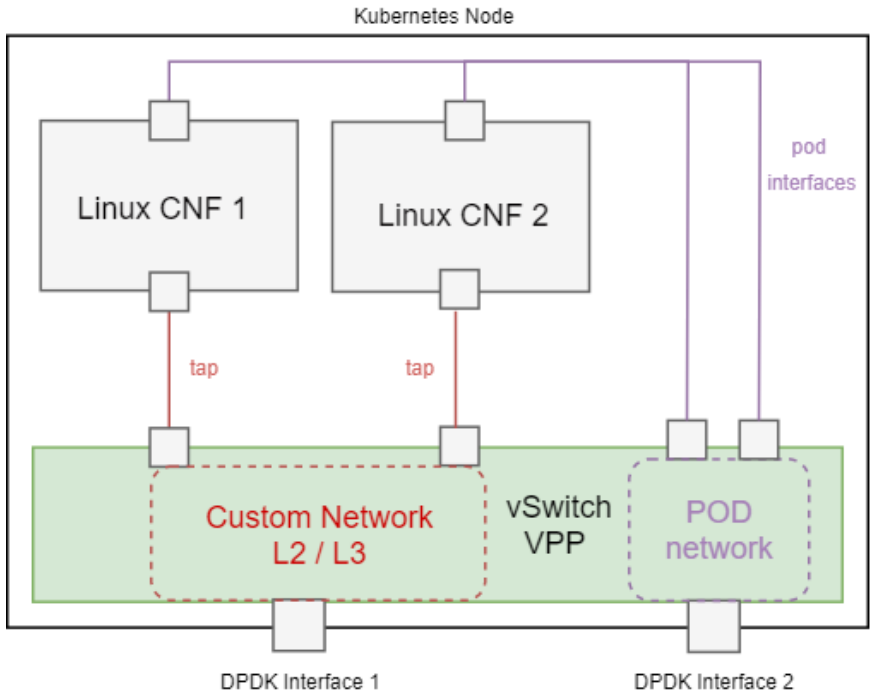
Contiv-VPP CNI

- Uses **FD.io VPP** with **DPDK** as the data-plane for packet forwarding
- Kube-proxy implemented in the user space (on VPP)
- Production-ready CNI (passes all k8s conformance tests)
- Swiss army knife CNI for **cloud-native networking deployments**:
  - Multiple network interfaces per pod
  - Multiple isolated L2/L3 networks
  - Service chaining between pods for CNF (Cloud-Native Network Functions) deployments
  - IPv6 support
  - SRv6 support

Contiv-VPP Data Plane



Multiple Pod Interfaces & Custom Networks

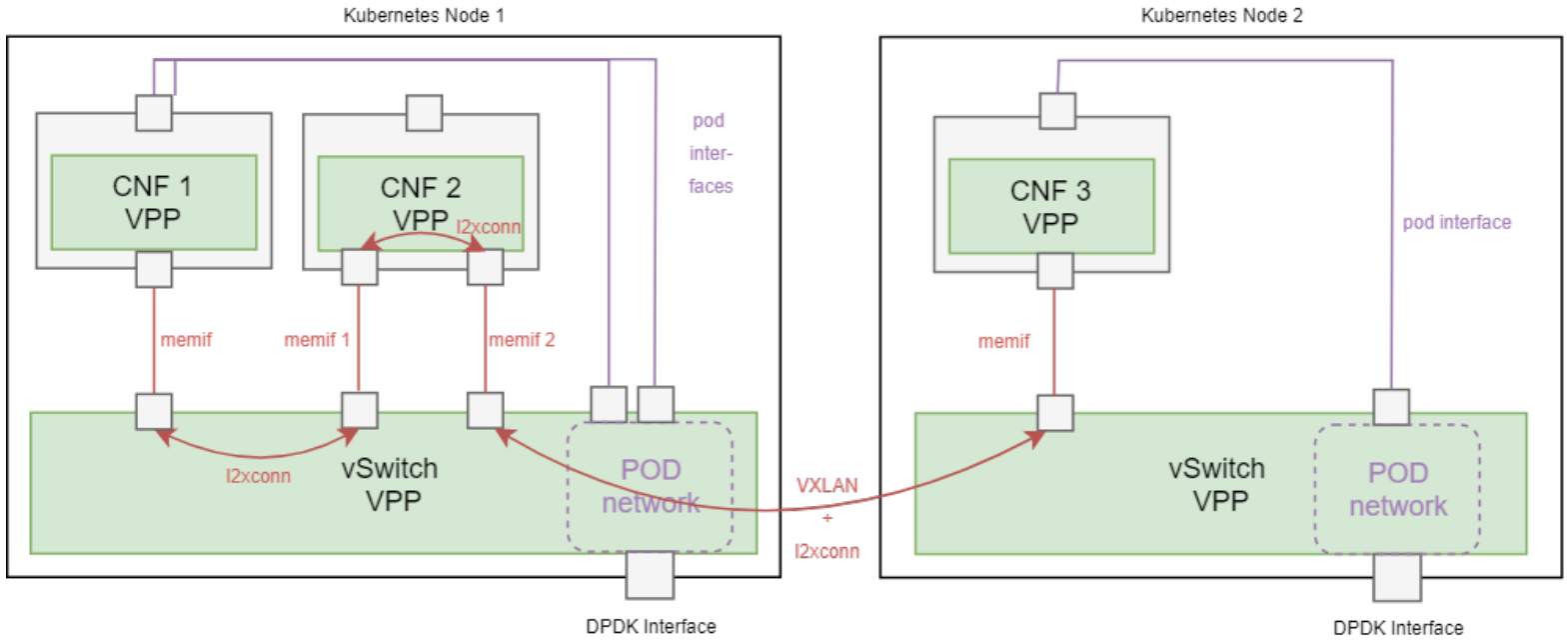



```
---
apiVersion: contivpp.io/v1
kind: CustomNetwork
metadata:
  name: l2net
spec:
  type: L2

---
kind: Pod
metadata:
  name: linux-cnf1
  annotations:
    contivpp.io/custom-if: tap1/tap/l2net
spec:
  …
```

https://github.com/contiv/vpp/tree/master/k8s/examples/custom-network

Service Chaining Between CNF Pods (L2-XConnect -Based)



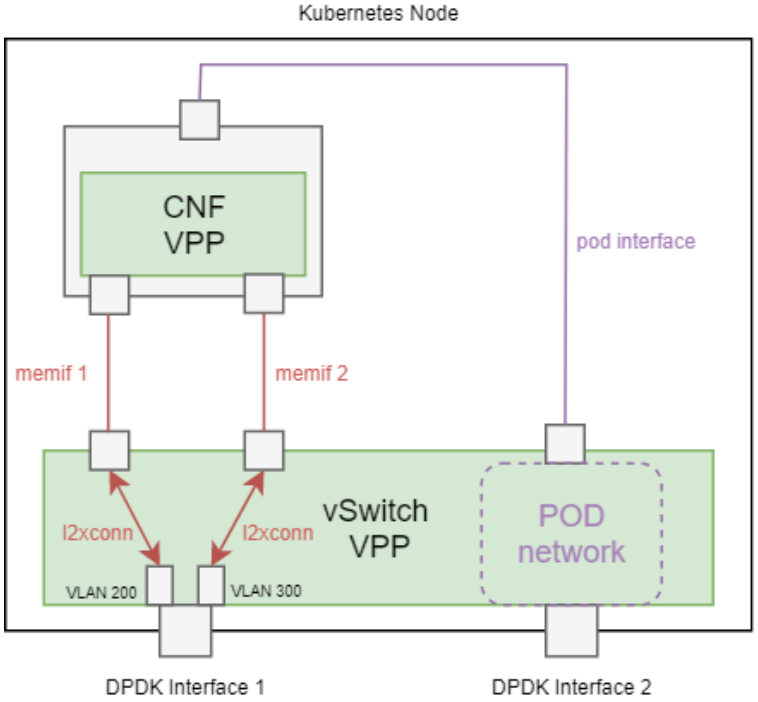
https://github.com/contiv/vpp/tree/master/k8s/examples/sfc

```
---
apiVersion: contivpp.io/v1
kind: ServiceFunctionChain
metadata:
  name: vpp-chain
spec:
  chain:
    - name: CNF 1
      type: Pod
      podSelector:
        cnf: vpp-cnf1
      interface: memif1

    - name: CNF 2
      type: Pod
      podSelector:
        cnf: vpp-cnf2
      inputInterface: memif1
      outputInterface: memif2

    - name: CNF 3
      type: Pod
      podSelector:
        cnf: vpp-cnf3
      interface: memif1
```

Service Chaining With External Interfaces (L2-XConnect -Based)



https://github.com/contiv/vpp/tree/master/k8s/examples/sfc

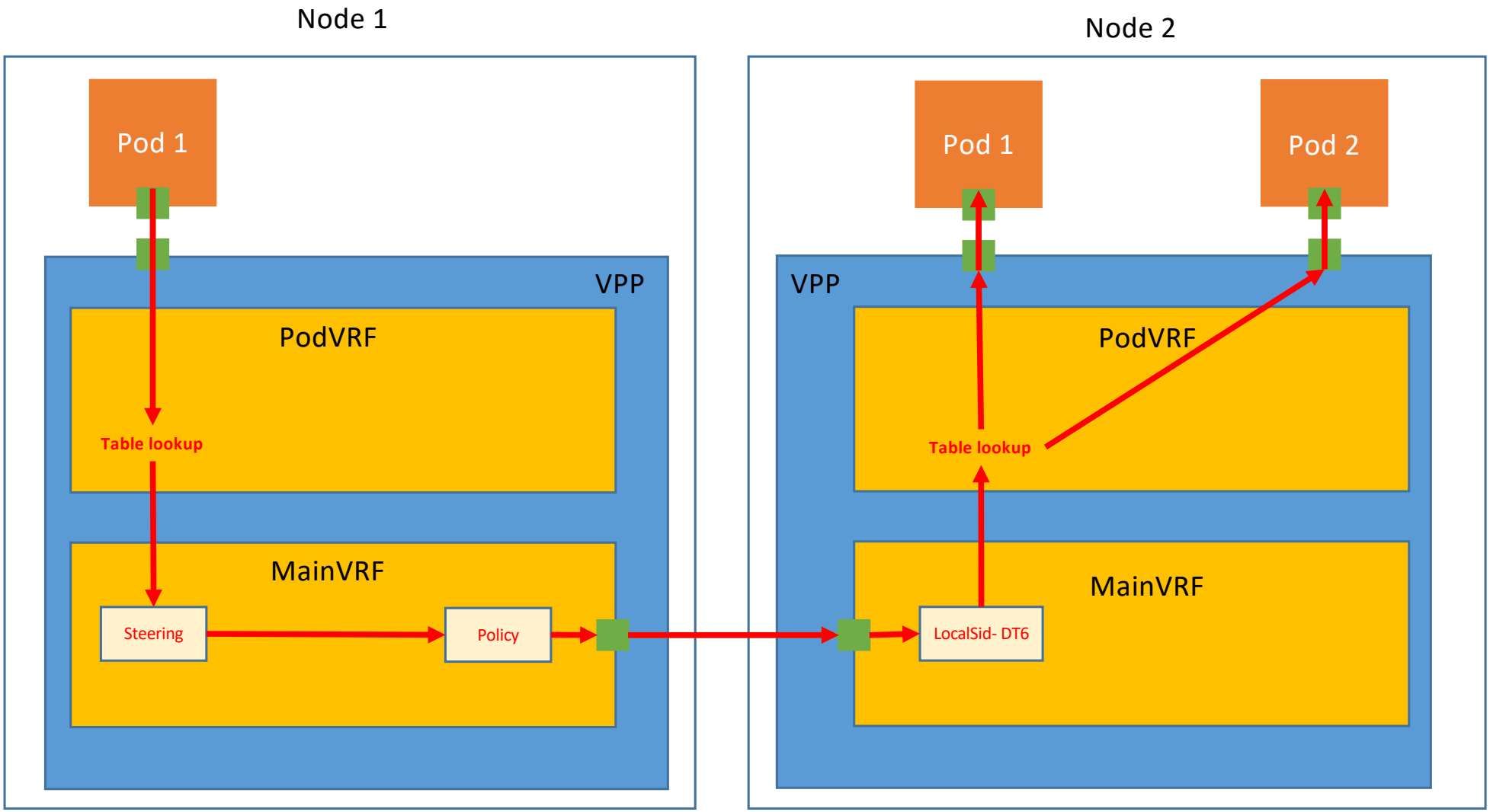
```
---
apiVersion: contivpp.io/v1
kind: ExternalInterface
metadata:
  name: vlan-200
spec:
  type: L2
  nodes:
    - node: k8s-master
      vppInterfaceName: GigabitEthernet0/a/0
      vlan: 200

---
apiVersion: contivpp.io/v1
kind: ServiceFunctionChain
metadata:
  name: vpp-chain
spec:
  chain:
    - name: VLAN 200 interface
      type: ExternalInterface
      interface: vlan-200

    - name: CNF
      type: Pod
      podSelector:
        cnf: vpp-cnf
```

SRv6 in Contiv-VPP CNI

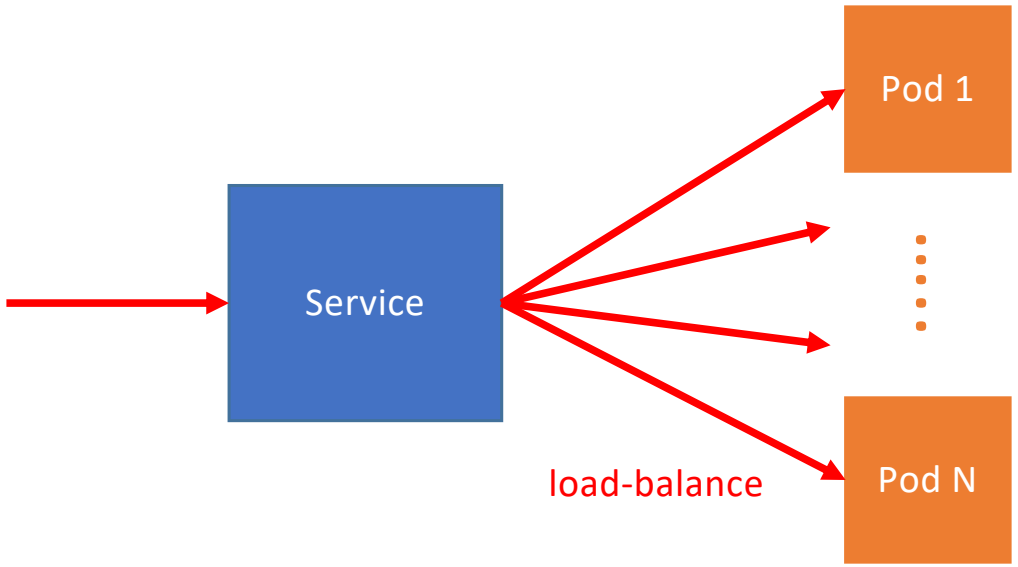
Pod-to-pod Communication in Contiv/VPP with SRv6



K8s Services with SRv6

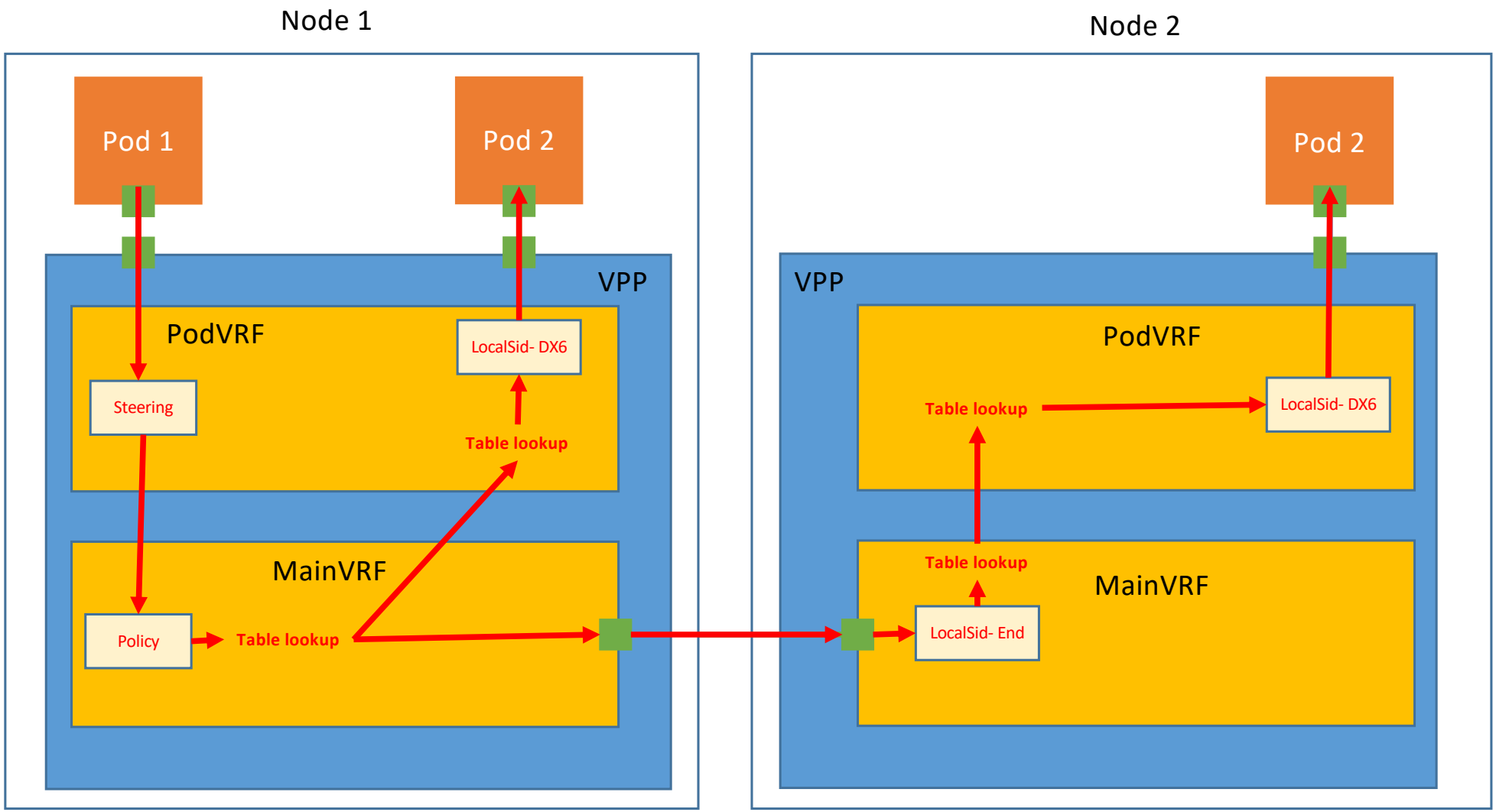
- For, implemented with NAT on VPP
- WithIPv4 SRv6, NAT is eliminated
- Traffic is steered via SRv6 policies, load-balanced between multiple SRv6 segment lists

  - Steering based on destination IP - Service Virtual IP address (Cluster IP)
  - Steering based on destination port on K8s node (NodePort)



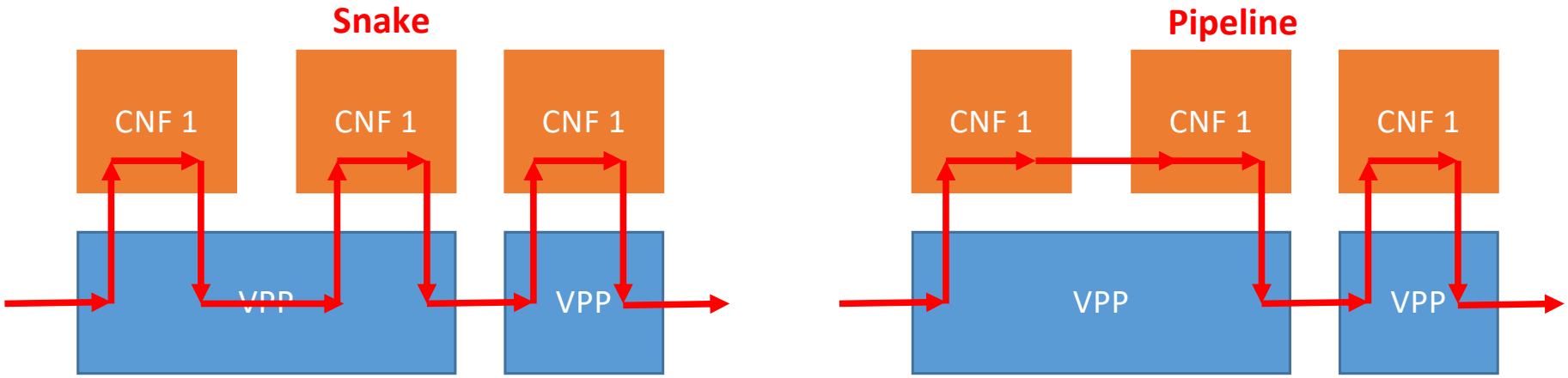
https://github.com/contiv/vpp/blob/master/docs/setup/SRV6.md

Pod-to-service Communication with SRv6



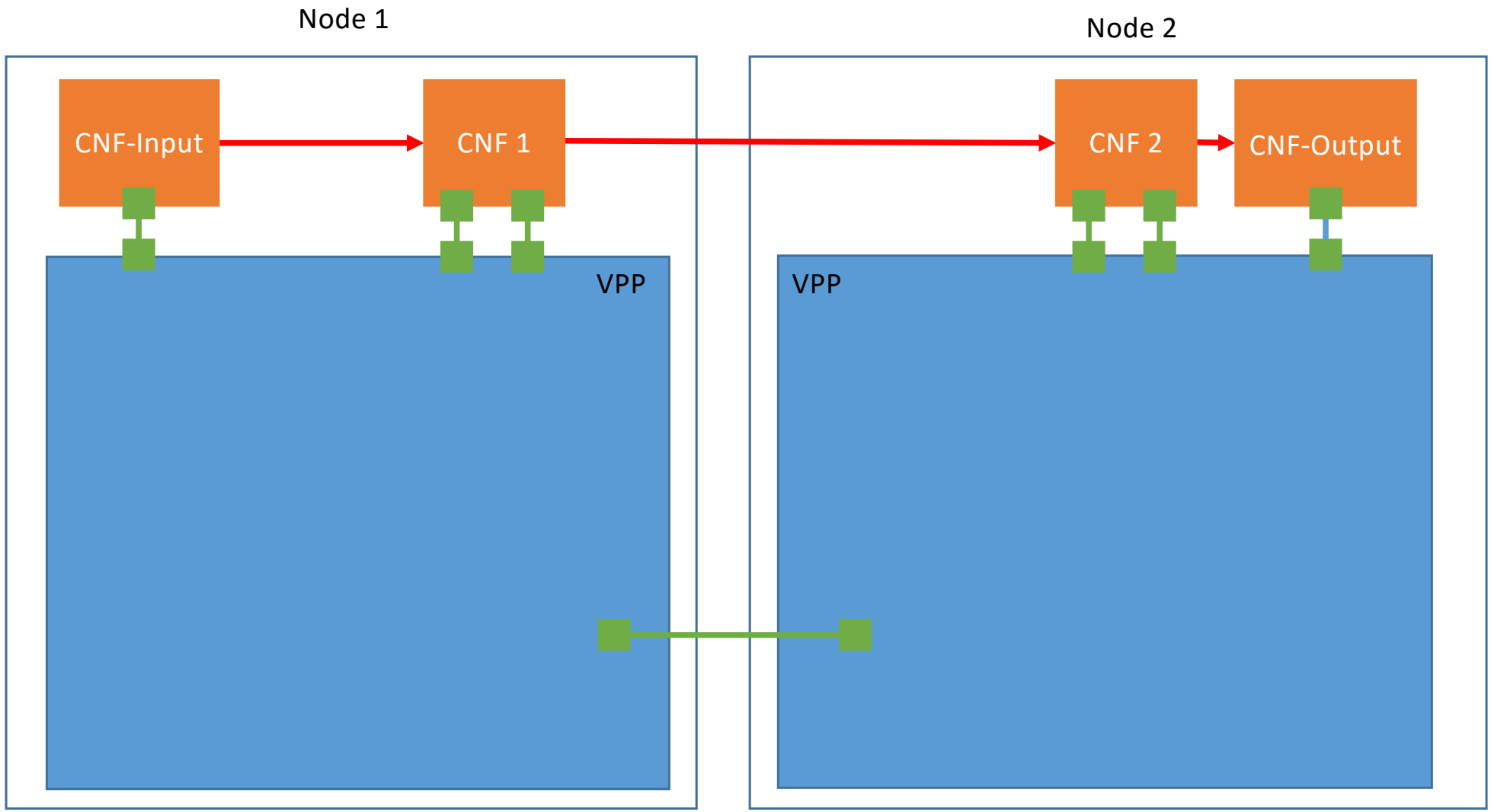
Service Function Chaining with SRv6

- SRv6 implementation for handling chains between SR-unaware CNFs
- “Snake" CNF chaining only (“Pipeline" is more complex to be done at the CNI level)
- Uses the same CRD APIs as the L2-Xconnect –based SFC

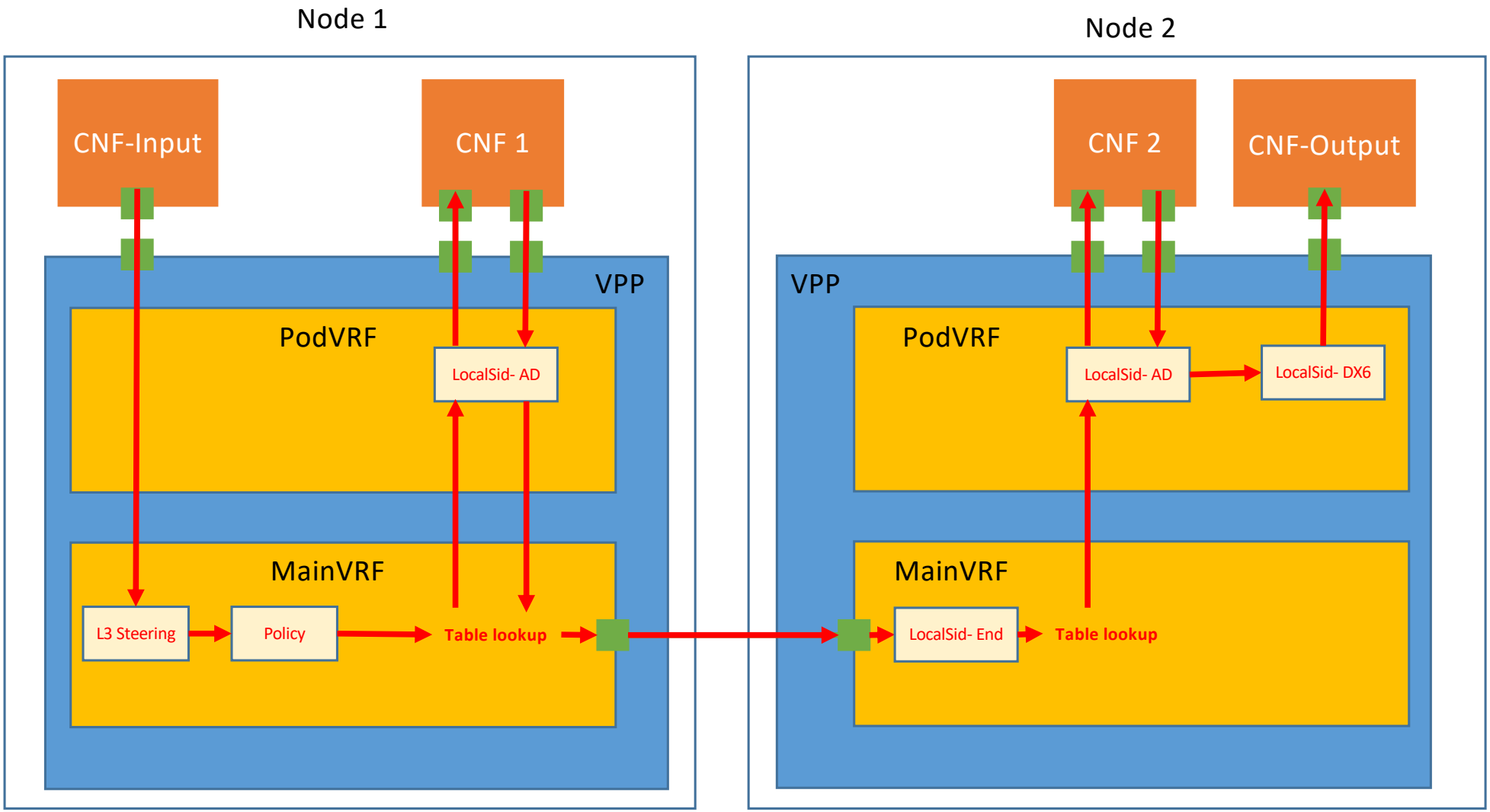


https://github.com/contiv/vpp/blob/master/docs/dev-guide/SFC.md#srv6-renderer

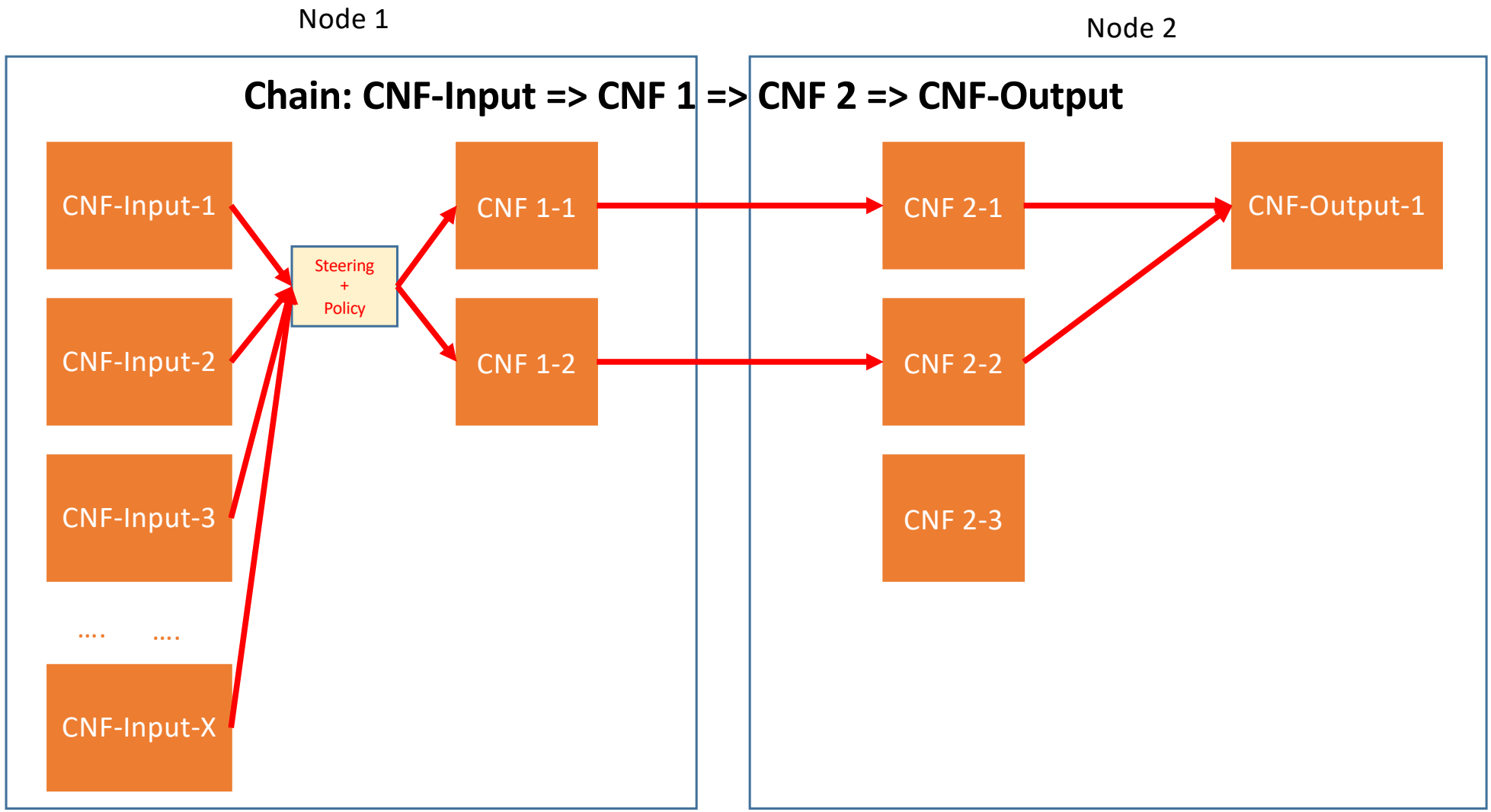
Service Function Chain Between CNFs



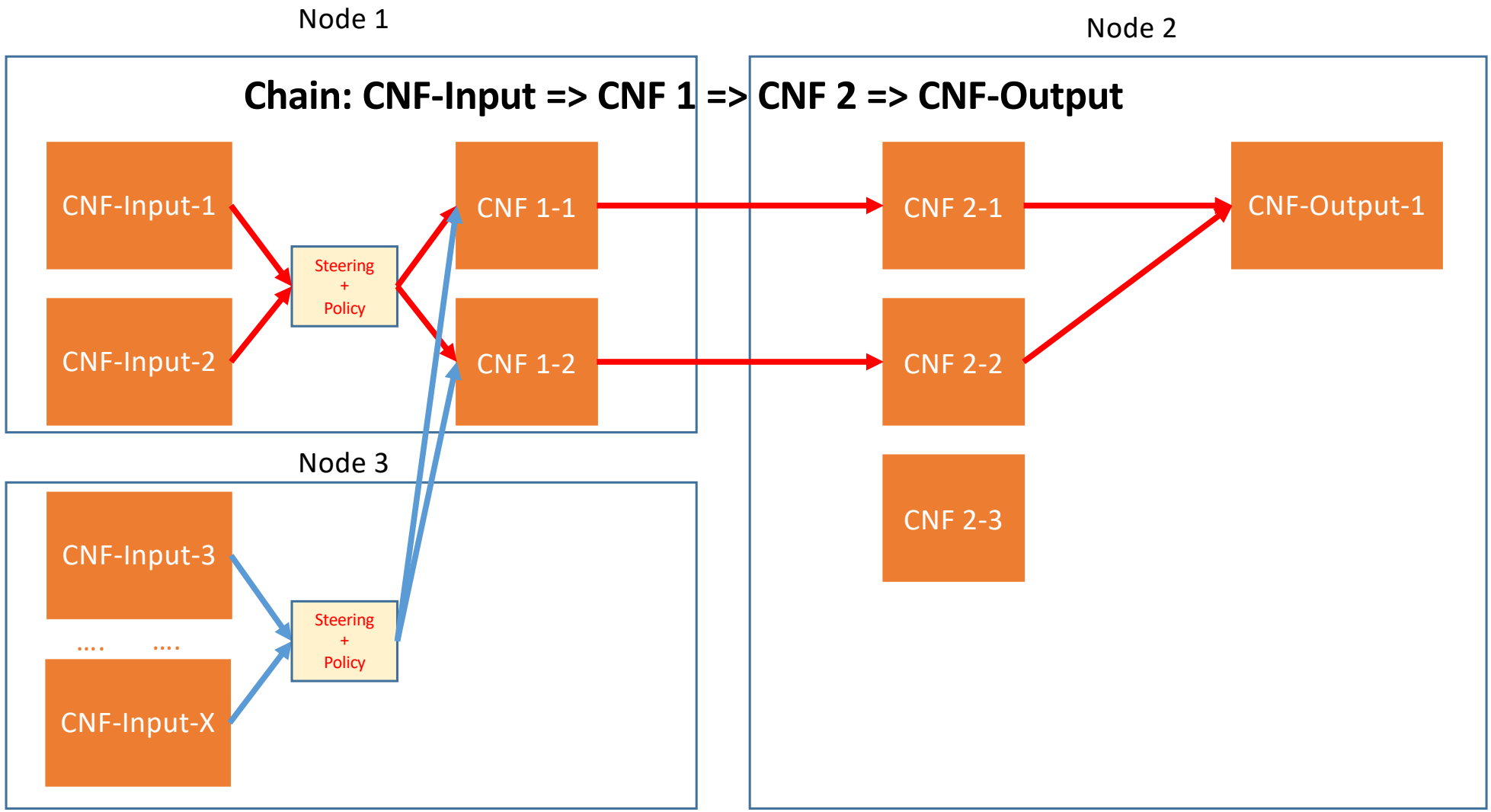
SFC Rendering with SRv6



Multi-path SFC Rendering with SRv6



Multi-path SFC Rendering with SRv6 – Multi-node



Accelerating SRv6 with Intel N3000 smartNIC


HCL
ENGINEERING
THE INTELLIGENT
CONNECTED
ECOSYSTEM
SRv6 Acceleration Solution

HW accelerator on Intel –PAC N3000

Thanks to Computer History Museum in Mtn View, CA

Thanks to Intel

SRv6 acceleration use case scenario

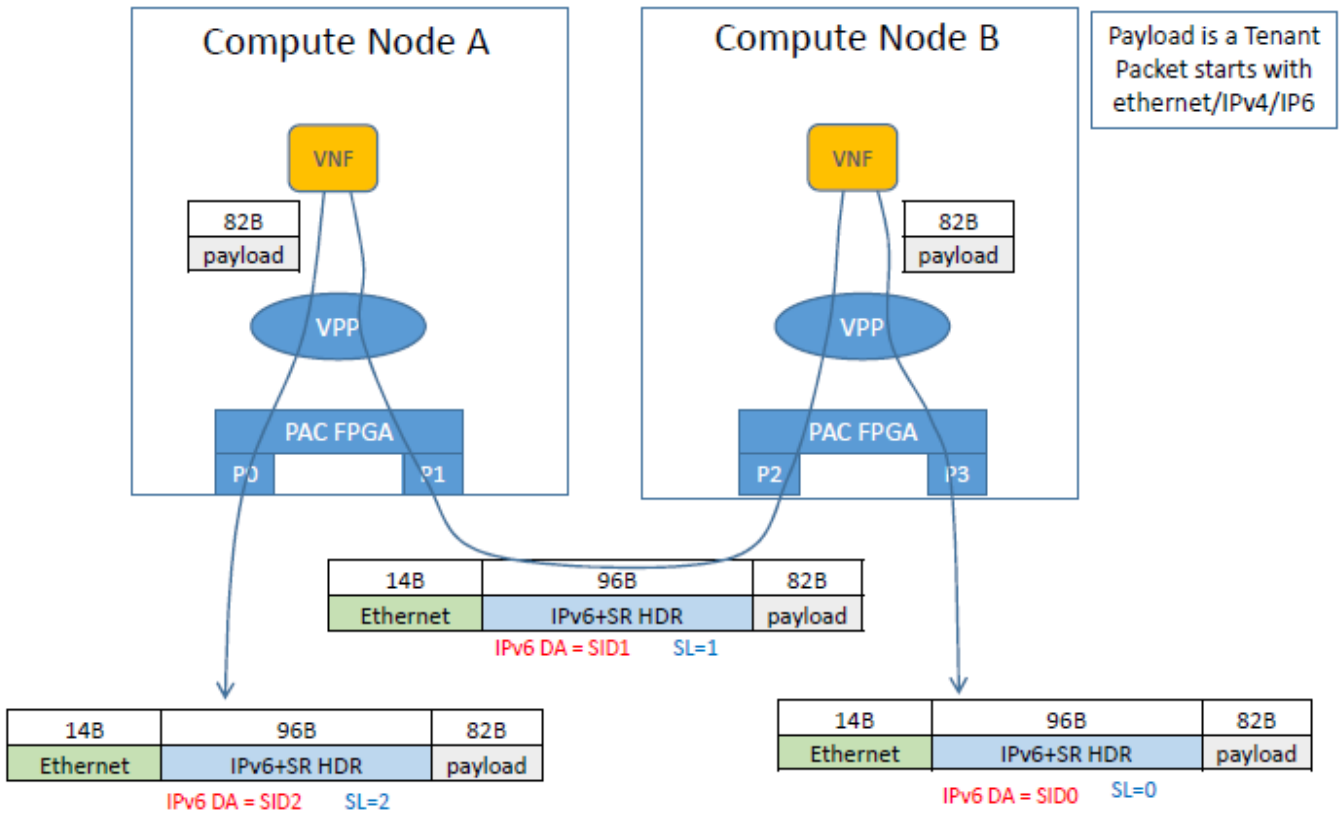
SRv6 Packet 192 Byte size

Offloaded Headers

14B	96B	82B
Ethernet	IPv6+SR HDR	payload

IPv6		Version	0.5
		Trafic Class	1
		Flow Label	2.5
		Payload Length	2
		Next Header	1
		Hop Limit	1
		Src Addr	16
		Dest Addr	16
	SRv6	Next Header	1
		Hdr Ext Len	1
		Routing Type	1
		Segment Left	1
		Last Entry	1
		Flags	1
		Tag	2
		SID List - SID0	16
		SID1	16
		SID2	16

SRv6 Service Function Chaining



SRv6 – Accelerated vs Non accelerated

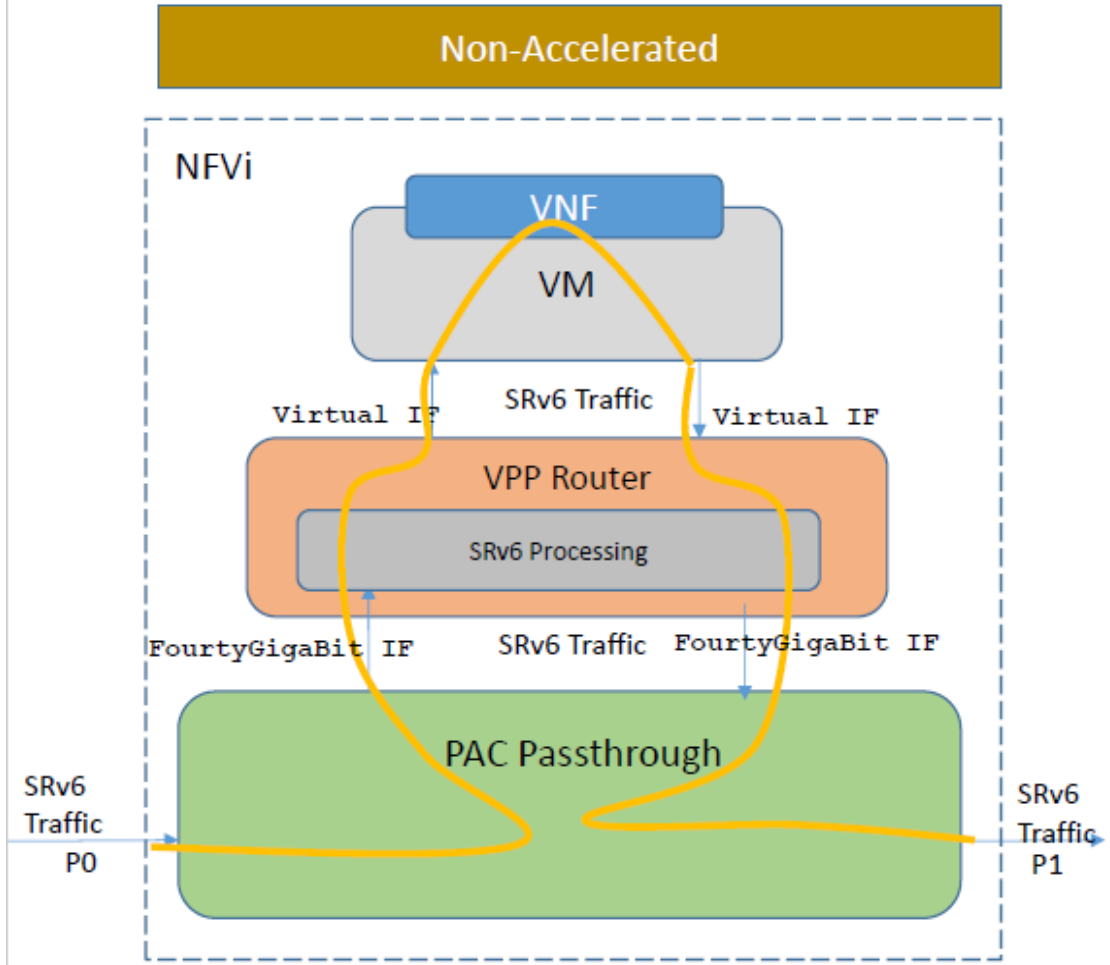


Figure 6 - SRv6 Non Acceleration

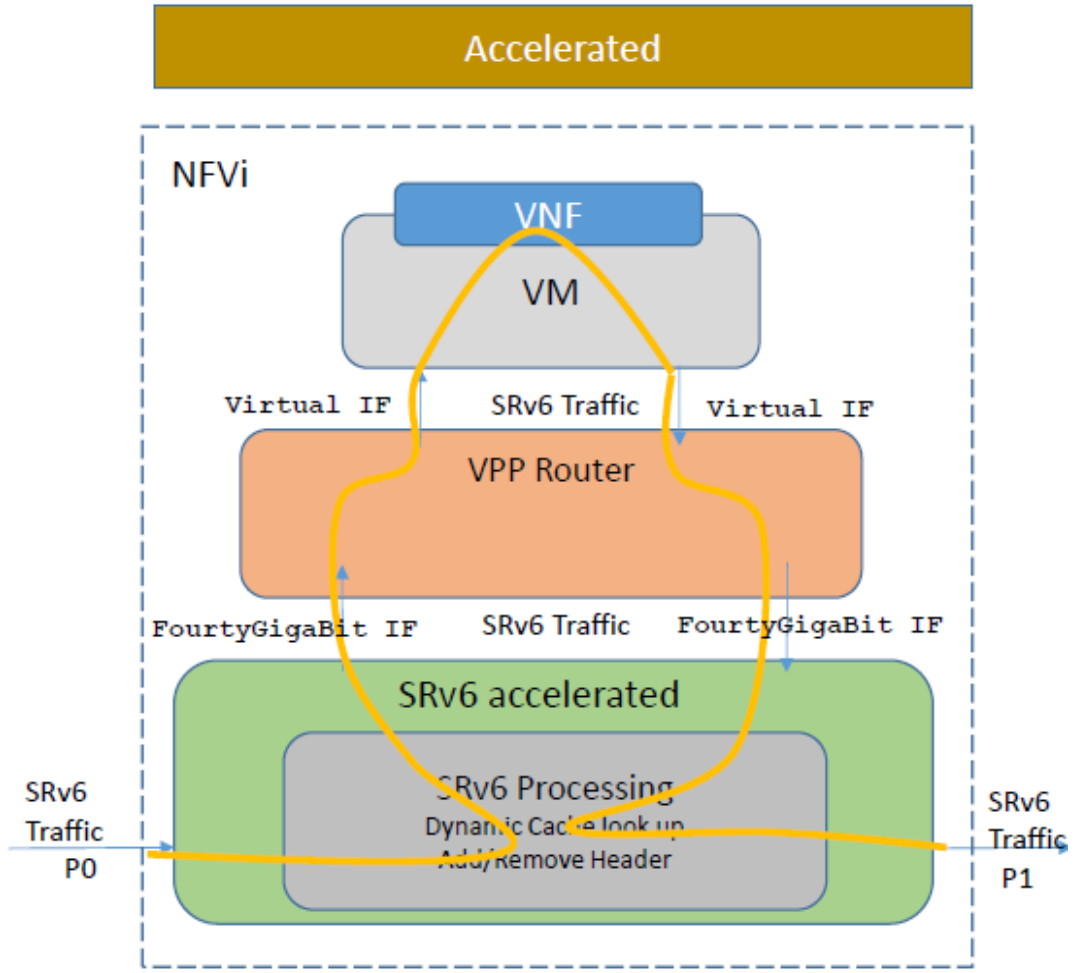
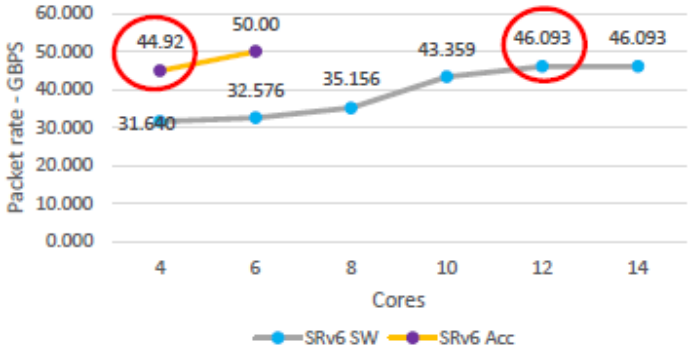


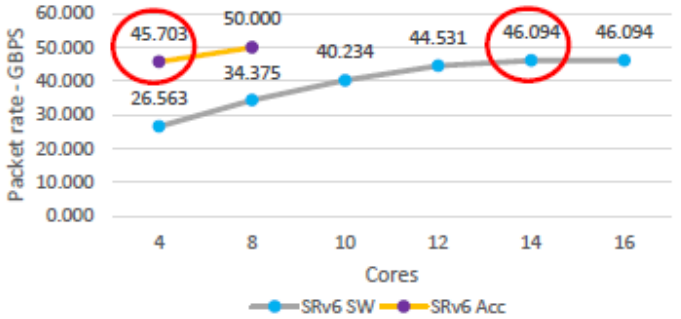
Figure 7 - SRv6 Acceleration Solution

SRV6 Acceleration Solution – Performance Data

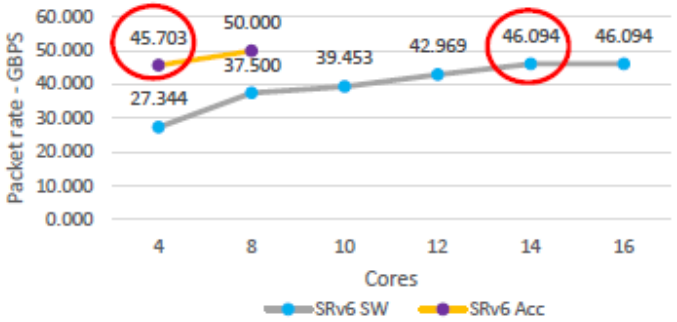


SRv6 AD2 Acceleration - 192B - GBPS

- 3x Performance improvement
- 8 to 10 cores savings



SRv6 AD6 Acceleration - 192B - GBPS



SRv6 AD4 Acceleration - 192B - GBPS

Conclusion

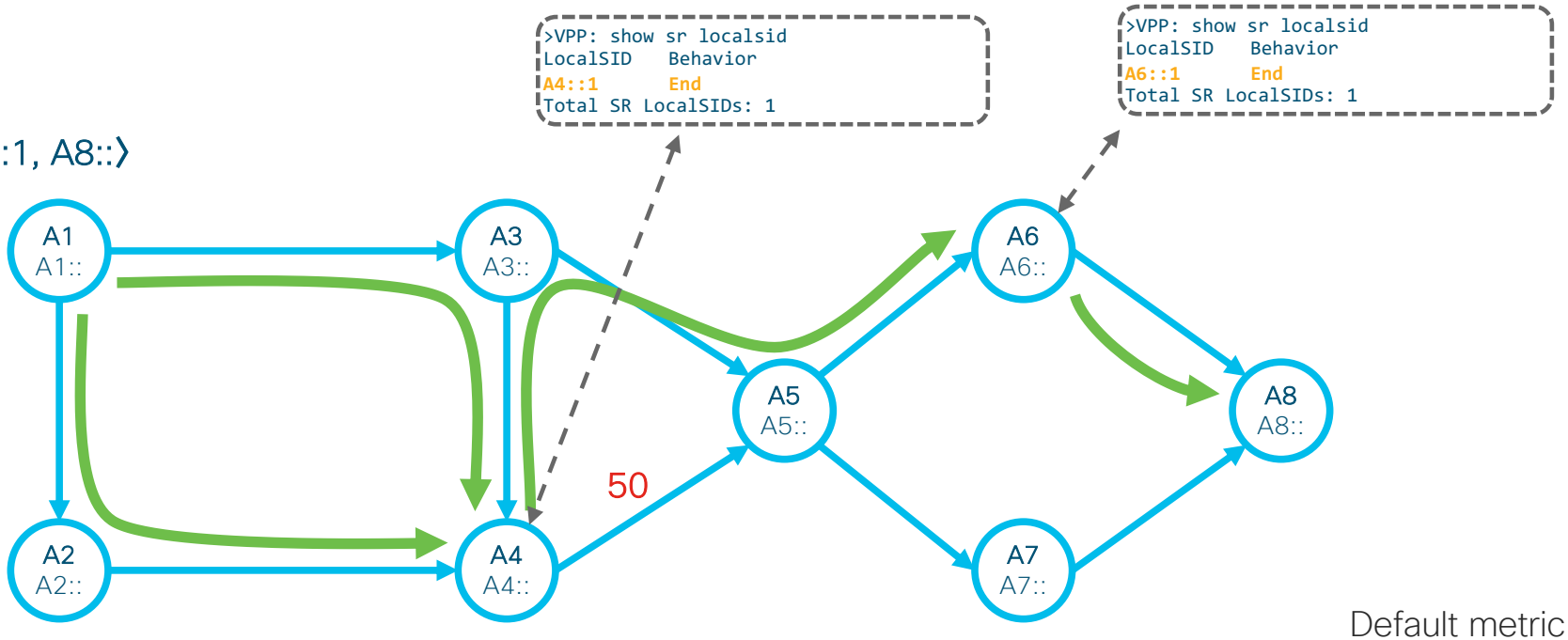
Conclusion

- Kubernetes does not provide any solution for handling containers networking. Instead, it offloads networking to the CNI plugins.
- CNI Plugins provides both connectivity and reachability to k8s pods
- IPv6 is needed to provide addressing and reachability for continuously increasing number of endpoints (i.e., containers/pods)
- SRv6 leverages the IPv6 dataplane and can handles the various k8s networking use-cases in simple and scalable way.
- Contiv-VPP + DPDK and Intel PAC N3000 smartNIC can provide faster I/O for k8s pods
  - They both support SRv6

Backup

Endpoint function

SR: ⟨A4::1, A6::1, A8::⟩

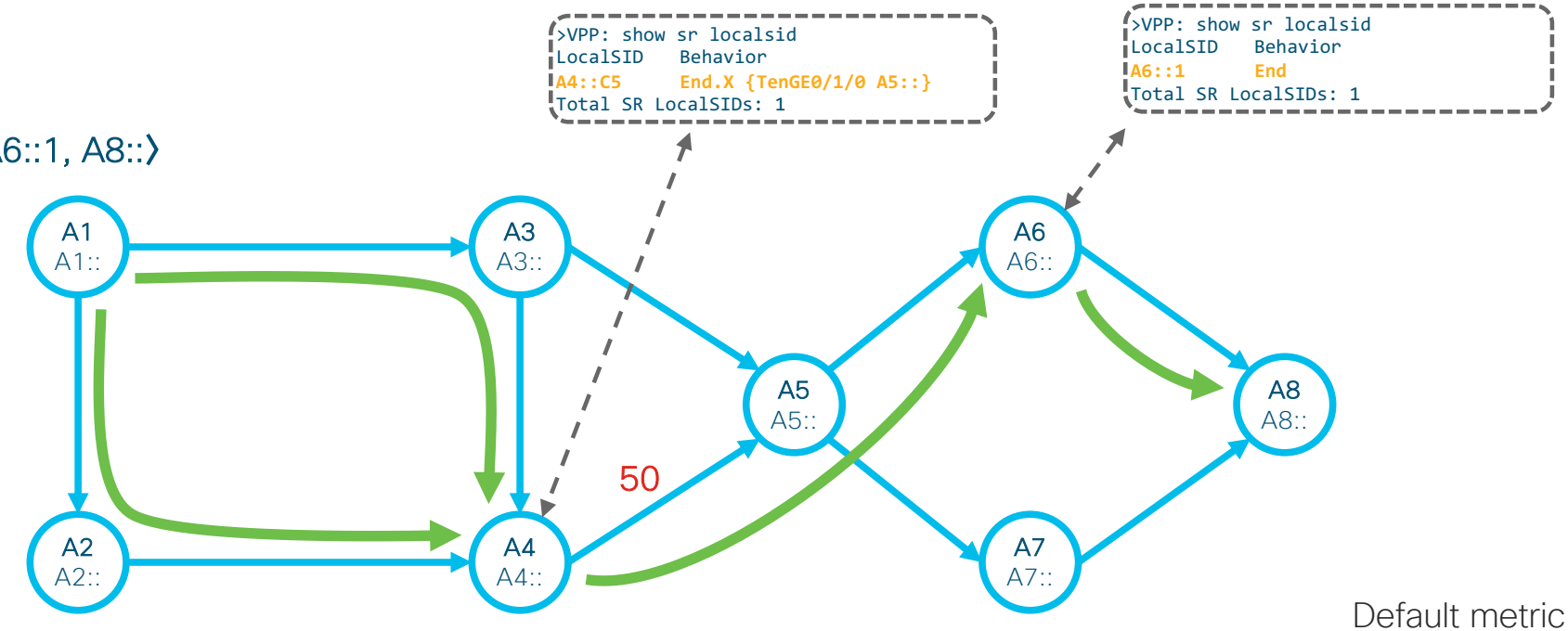


Default metric 10

- For simplicity function 1 denotes the most basic function
- Shortest-path to the Node

Endpoint then xconnect to neighbor function

SR: ⟨A4::C5, A6::1, A8::⟩



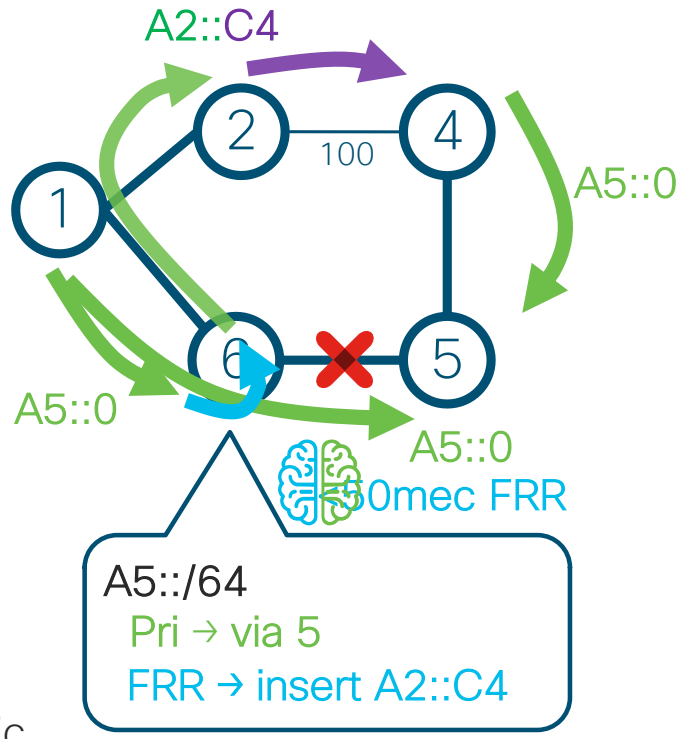
Default metric 10

- For simplicity Ak::Cj denotes:
  - Shortest-path to the Node K and then x-connect (function C) to the neighbor J

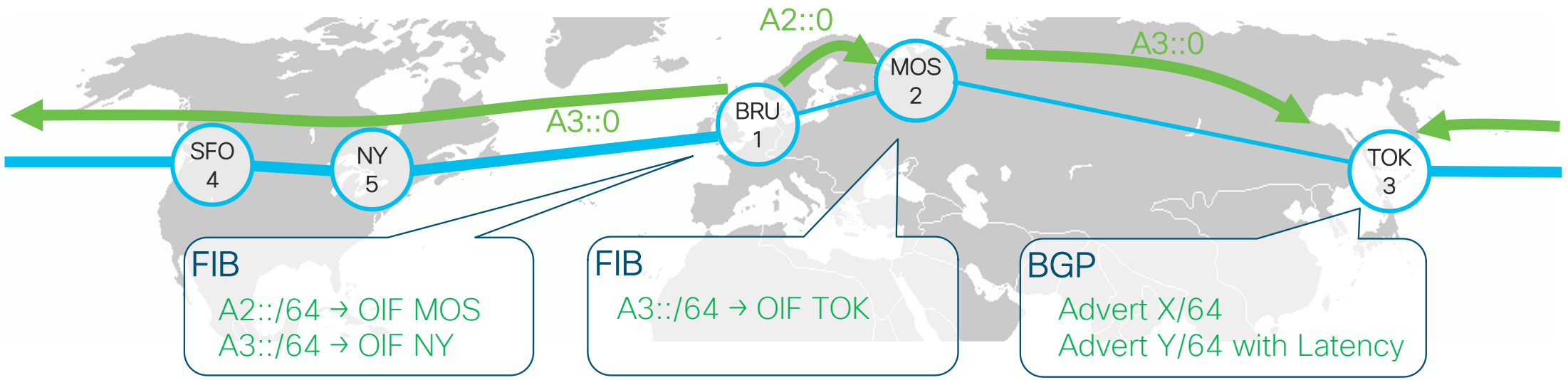
Deployment use-cases

TILFA

- 50msec Protection upon
  local link, node or SRLG failure
- Simple to operate and understand
  - automatically computed by the router's IGP process
  - 100% coverage across any topology
  - predictable (backup = postconvergence)
- Optimum backup path
  - leverages the post-convergence path, planned to carry the traffic
  - avoid any intermediate flap via alternate path
- Incremental deployment
- Distributed and Automated Intelligence

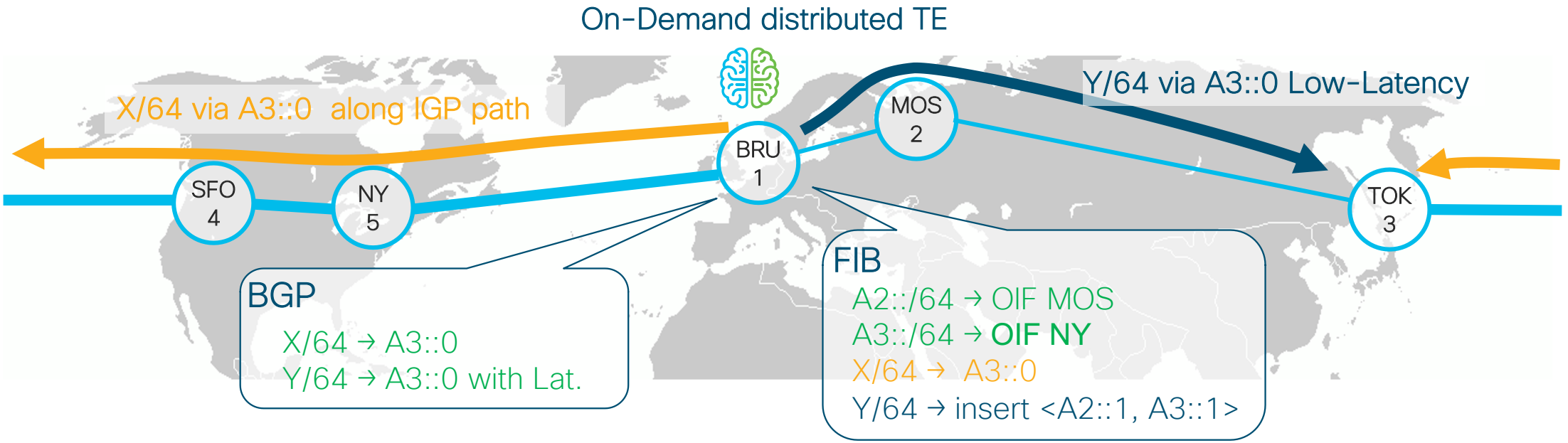


Distributed & Automated TE



- IGP minimizes cost instead of latency

Distributed & Automated TE



- Distributed and Automated Intelligence
- Dynamic SRTE Policy triggered by learning a BGP route with SLA contract
- No PBR steering complexity, No PBR performance tax, No RSVP, No tunnel to configure

Centralized TE

Input Acquisition

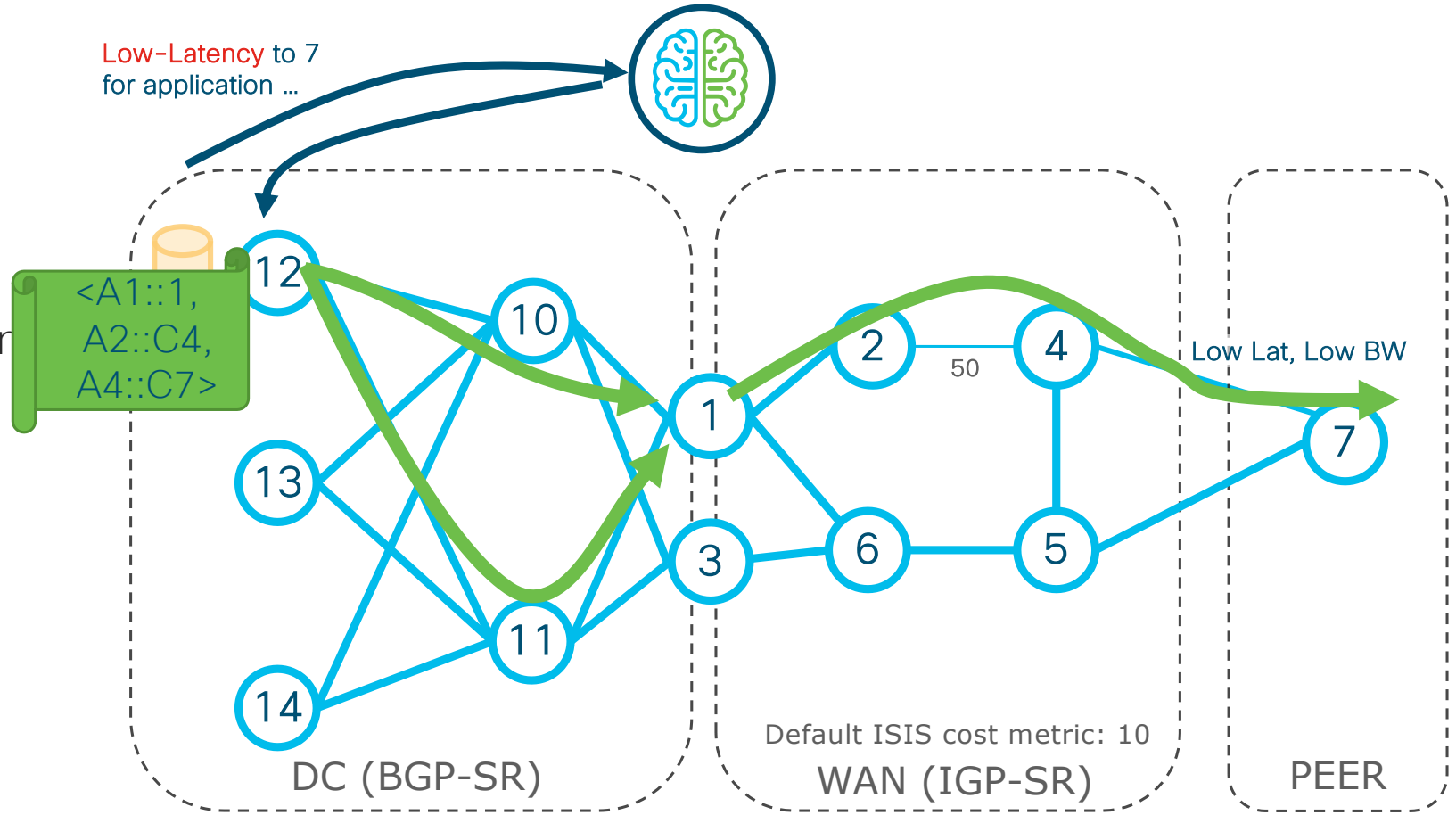
- BGP-LS
- Telemetry

Policy Instantiation

- PCEP
- BGP-TE
- Netconf / Yang

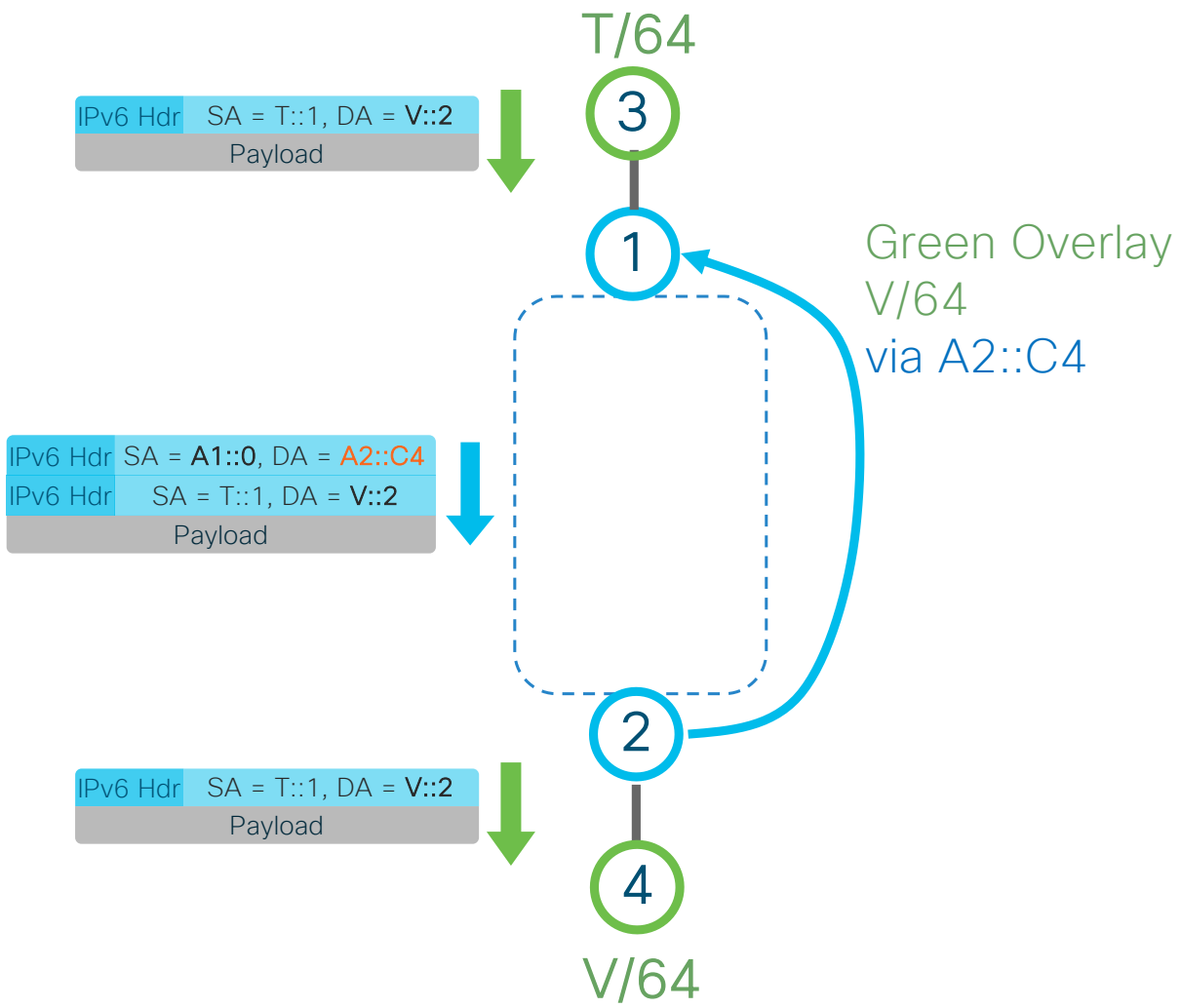
Algorithm

- SR native



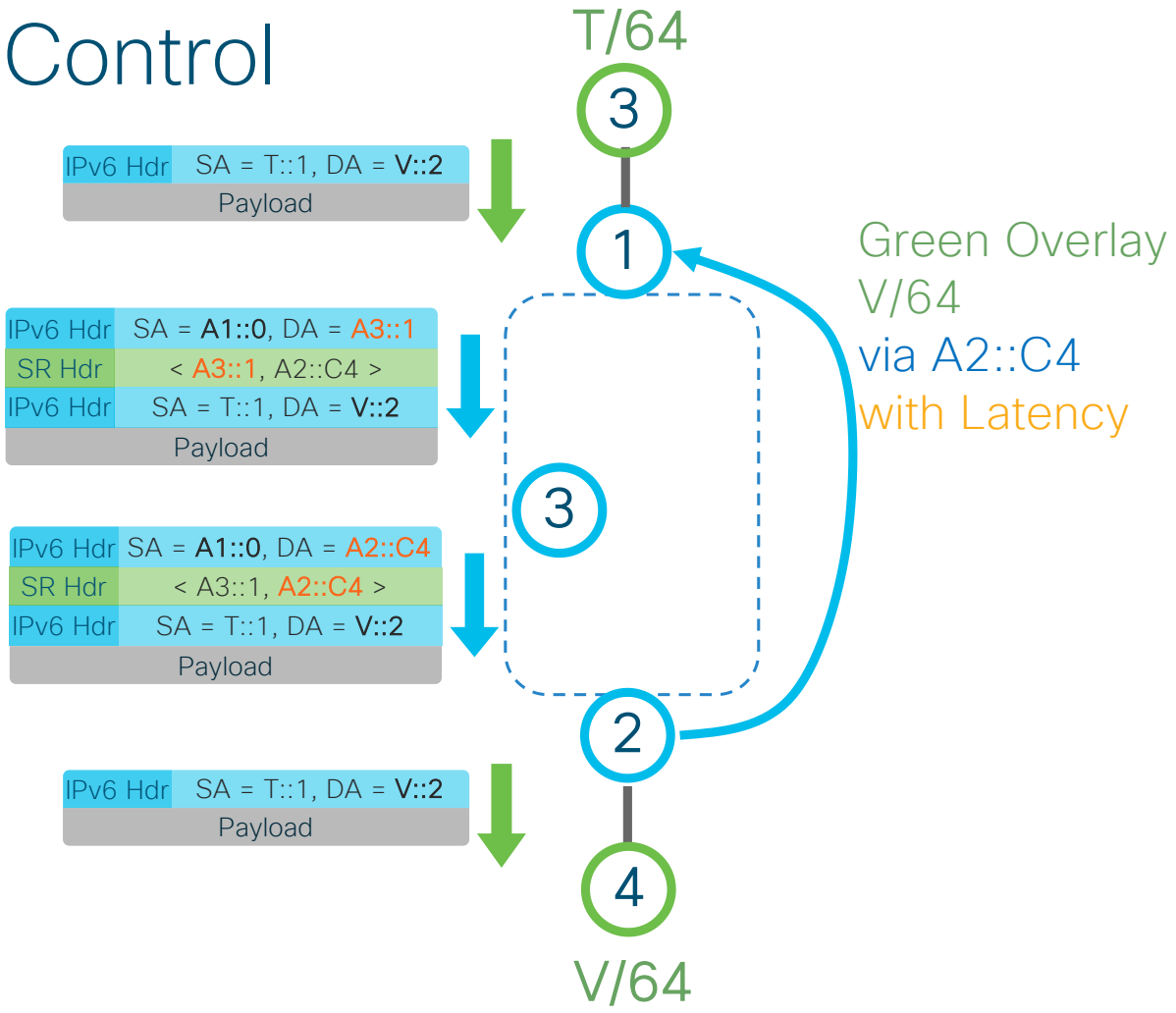
Overlay

- Automated
  - No tunnel to configure
- Simple
  - Protocol elimination
- Efficient
  - SRv6 for everything



Overlay with Underlay Control

- SRv6 does not only eliminate unneeded overlay protocols
- SRv6 solves problems that these protocols cannot solve



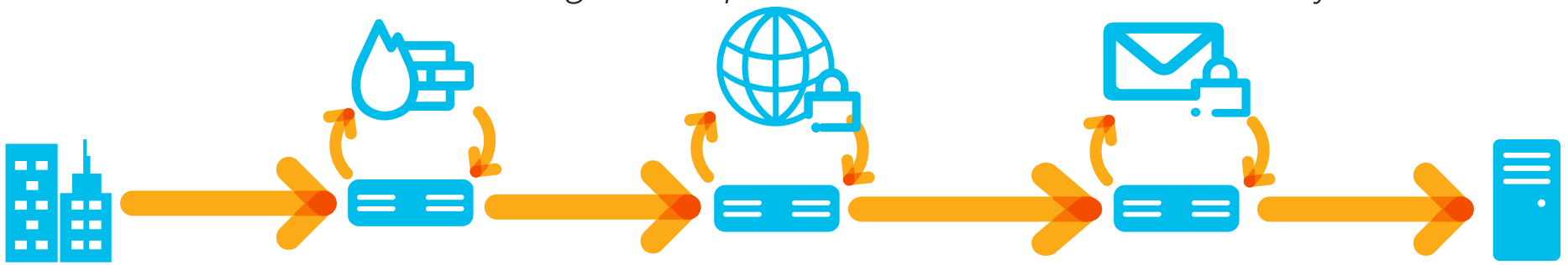
Service chaining

Service Chaining

Packets from are steered through a sequence of services on their way to the server

Service Chaining with **NSH**

Packets from are steered through a sequence of services on their way to the server



- Dedicated encapsulation header
  - State to be maintained for each service chain

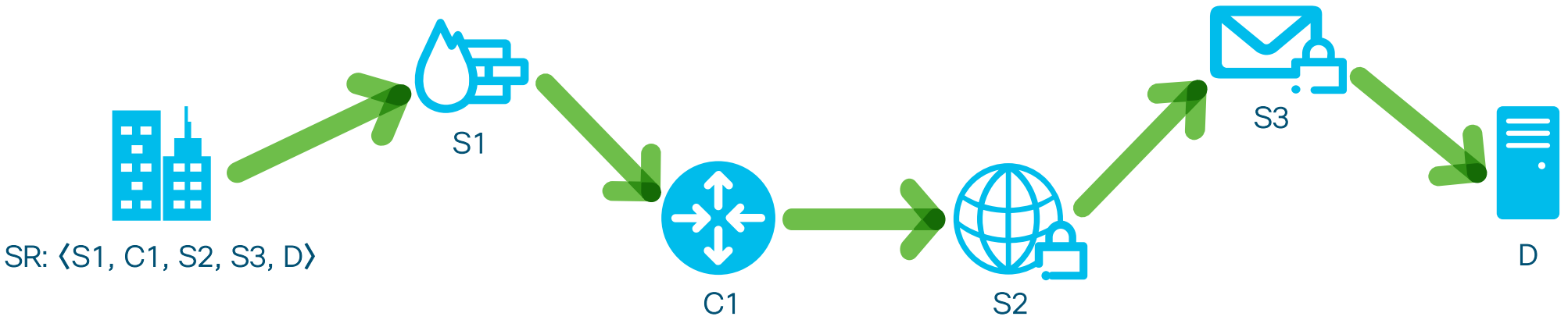
Service Chaining with **SRv6**

Packets from are steered through a sequence of services on their way to the server

- **Services** are expressed with **segments**
  - Flexible
  - Scalable
  - Stateless

Service Chaining with **SRv6**

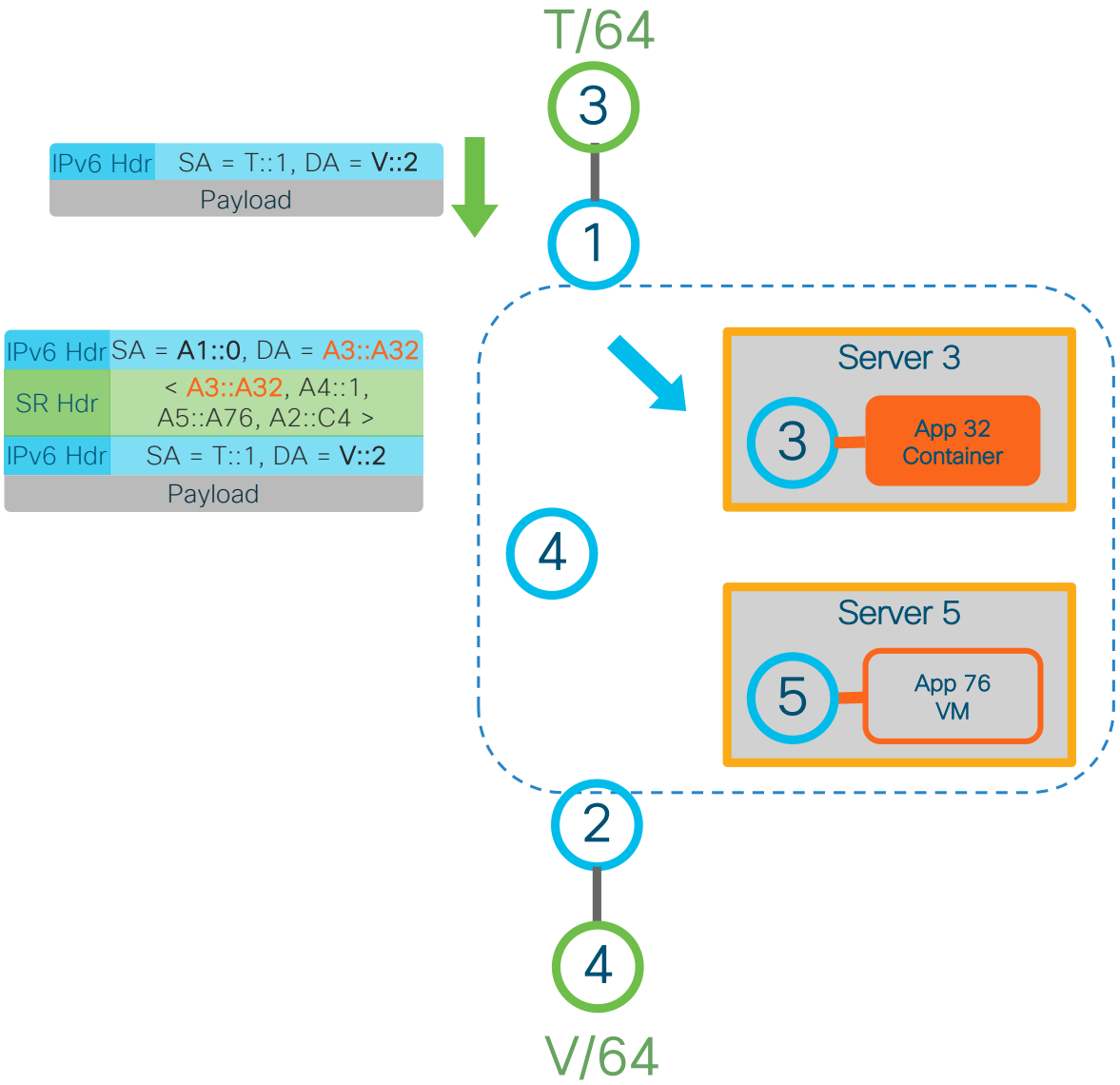
Packets from are steered through a sequence of services on their way to the server



- **Services** are expressed with **segments**
  - Flexible
  - Scalable
  - Stateless

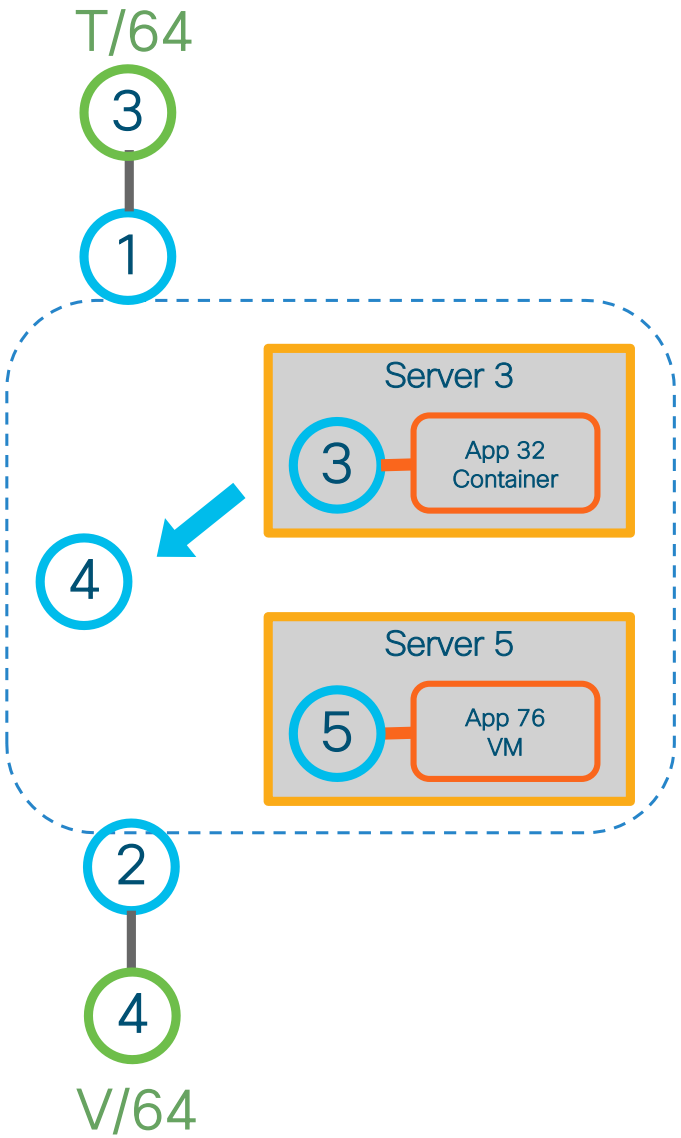
Integrated NFV

- Stateless
  - NSH creates per-chain state in the fabric
  - SR does not
- App is SR aware or not
- App can work on IPv4, IPv6 or L2



Integrated NFV

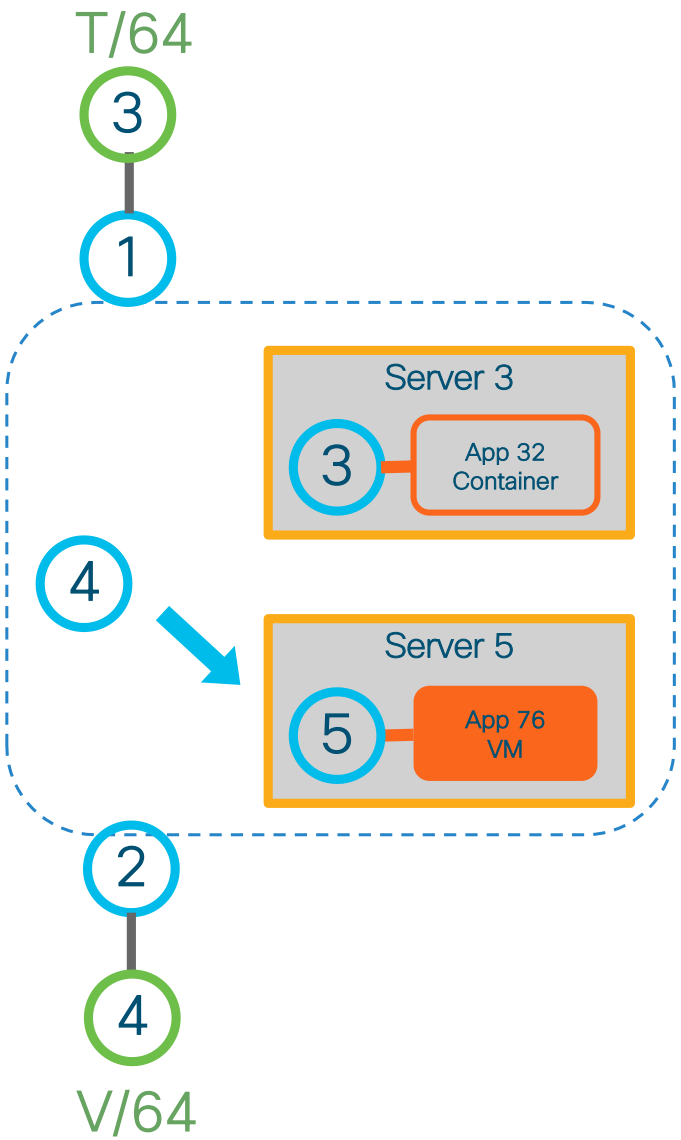
- Integrated with underlay SLA



Integrated NFV

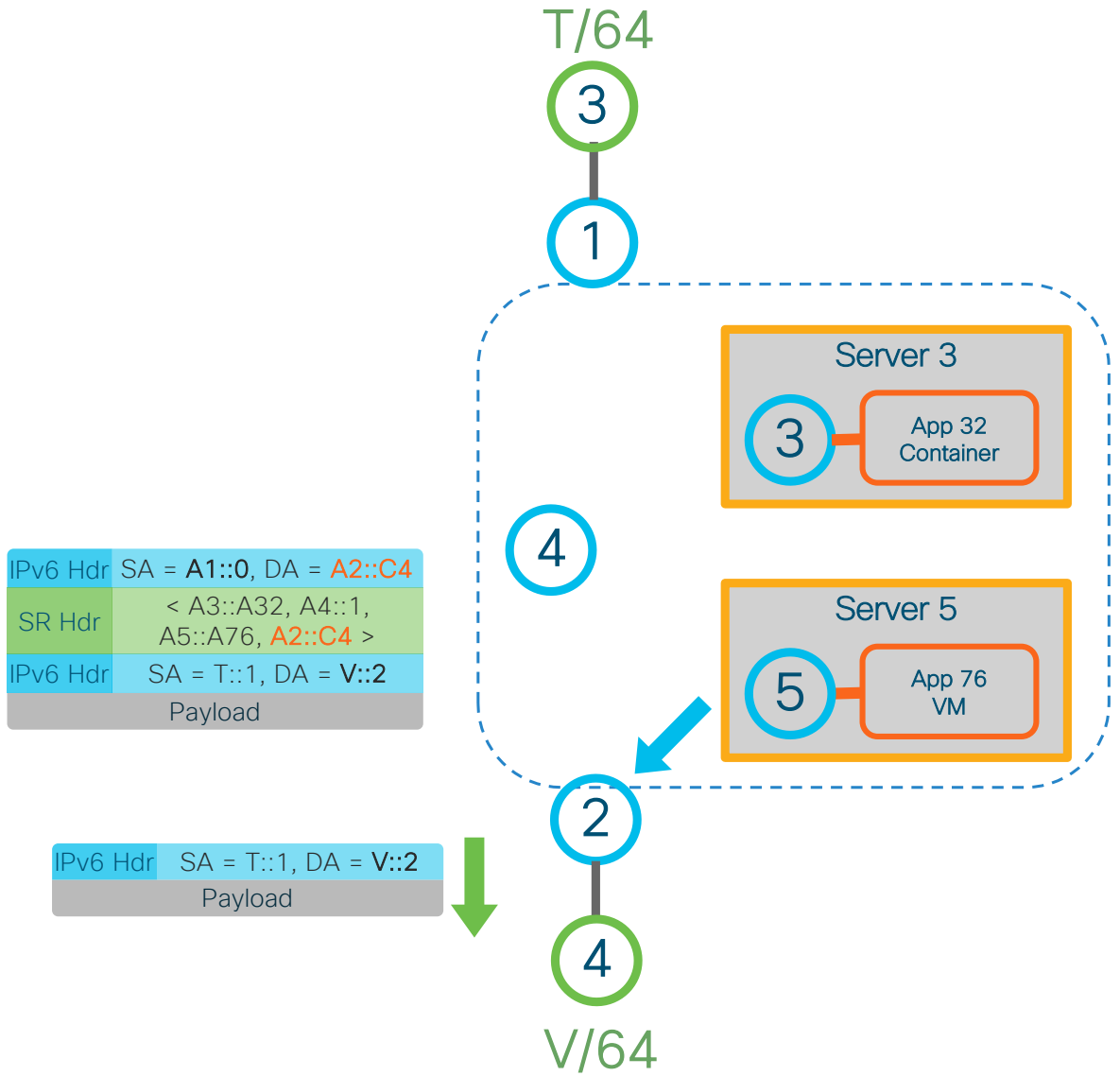
- Stateless
  - NSH creates per-chain state in the fabric
  - SR does not
- App is SR aware or not
- App can work on IPv4, IPv6 or L2

IPv6 Hdr	SA = **A1::0**, DA = A5::A76
SR Hdr	< A3::A32, A4::1, A5::A76, A2::C4 >
IPv6 Hdr	SA = T::1, DA = **V::2**
Payload	



Integrated NFV

- Integrated with Overlay



Service Chaining with **SRv6**

Types of VNFs

SR-Aware VNFs:

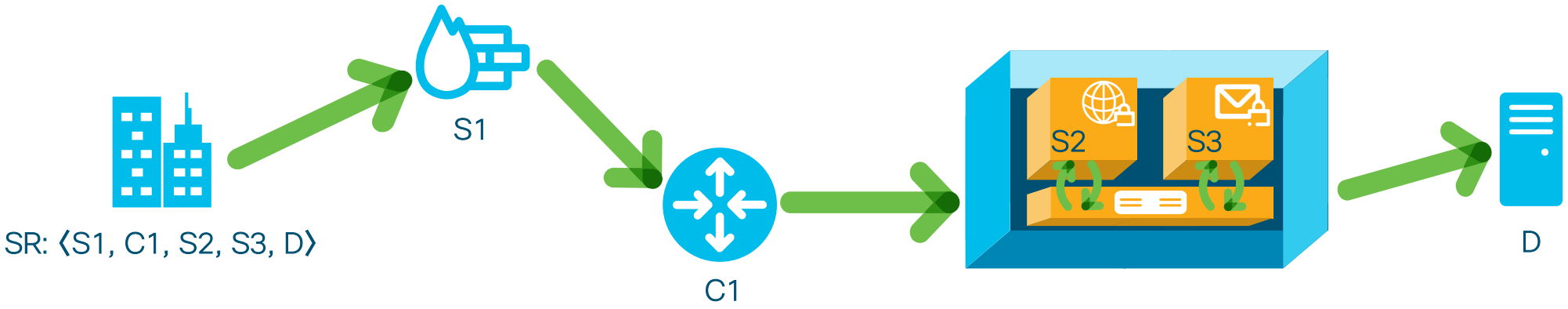
- Leverage SRv6 Kernel support to create smarter applications
- SERA: SR-Aware Firewall (extension to iptables)

SR-UnAware VNFs:

- Application is not aware of SR at all
- Leverage VPP as a vm/container vSwitch to do SRv6 processing

SR-UnAware VNFs

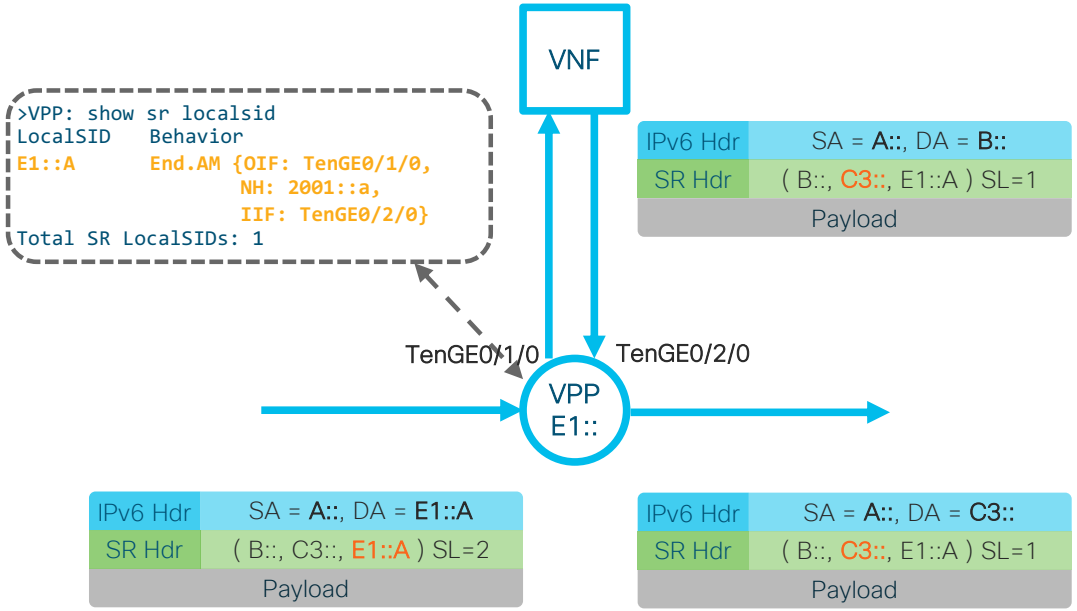
- End.AM – Endpoint to SR-unaware app via masquerading
- End.AD – Endpoint to SR-unaware app via dynamic proxy
- End.ASM – Endpoint to SR-unaware app via shared memory



End.AM – Endpoint to SR-unaware app via masquerading

RFC2460:
"A Routing header is not examined or processed until it reaches the node identified in the Destination Address field of the IPv6 header."

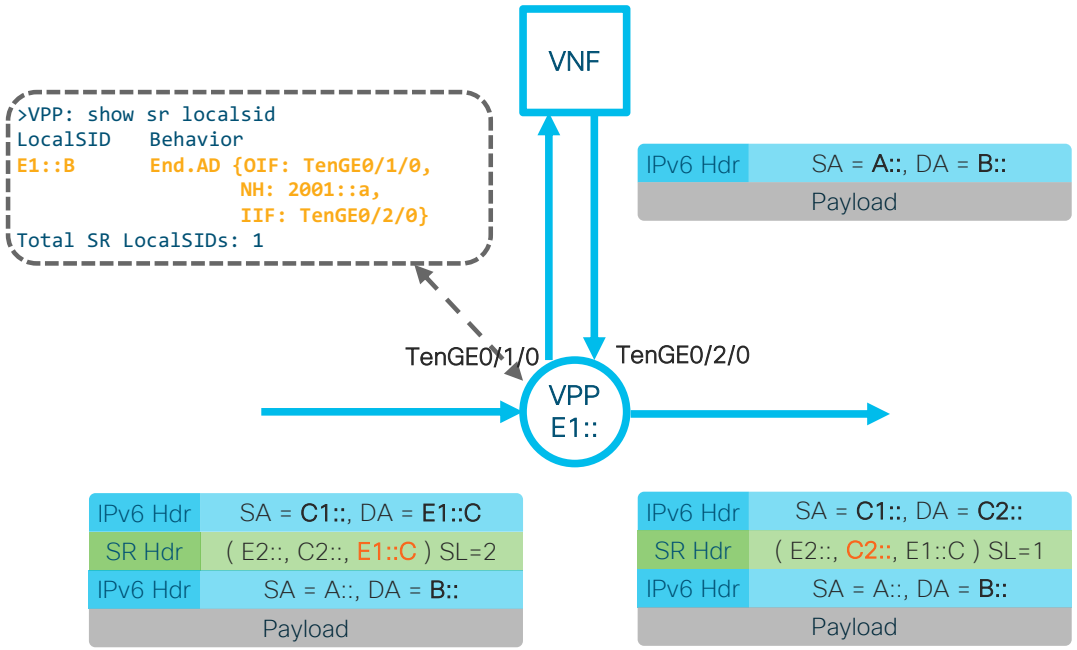
- Ingress:
  - Active SID is E1::A where function 0xA is associated with End.AM
  - Replace DA with the last segment B::
  - Forward to VNF (OIF, NH)
- Egress:
  - Inspect SRH and update DA with active segment C3::



End.AD – Endpoint to SR-unaware app via dynamic proxy

- Ingress:
  - Active SID is E1::B where function 0xB is associated with End.AD
  - Pop and store outer IP and SR headers
  - Forward to VNF (OIF, NH)
- Egress:
  - Push the IP and SR headers
  - Forward based on next segment

- Valid for IPv4 and IPv6 traffic
- Per-chain dynamic configuration



End.ASM – Endpoint to SR-unaware app via shared mem.

1. Put the received packet in a shared memory region
2. Perform SR processing on the host
   Pass a **pointer** of the inner packet to S2
3. Perform SR processing on the host
   Pass a **pointer** of the inner packet to S3
4. Move the packet from the shared memory into the output iface buffer ring

- Valid for IPv4 and IPv6 traffic
- Max. theoretical achievable performance