BGPVPN for Edge Cloud

Hareesh Puthalath, Technical Lead
Sebastian Jeuk, Senior Consulting Engineer
Ian Wells, Distinguished Engineer
May 23, 2018

Agenda

- Edge Clouds
- Service Provider Networks
- MPLS in OpenStack Today
- OpenStack BGP-VPN API
- Our Implementation
- Conclusion

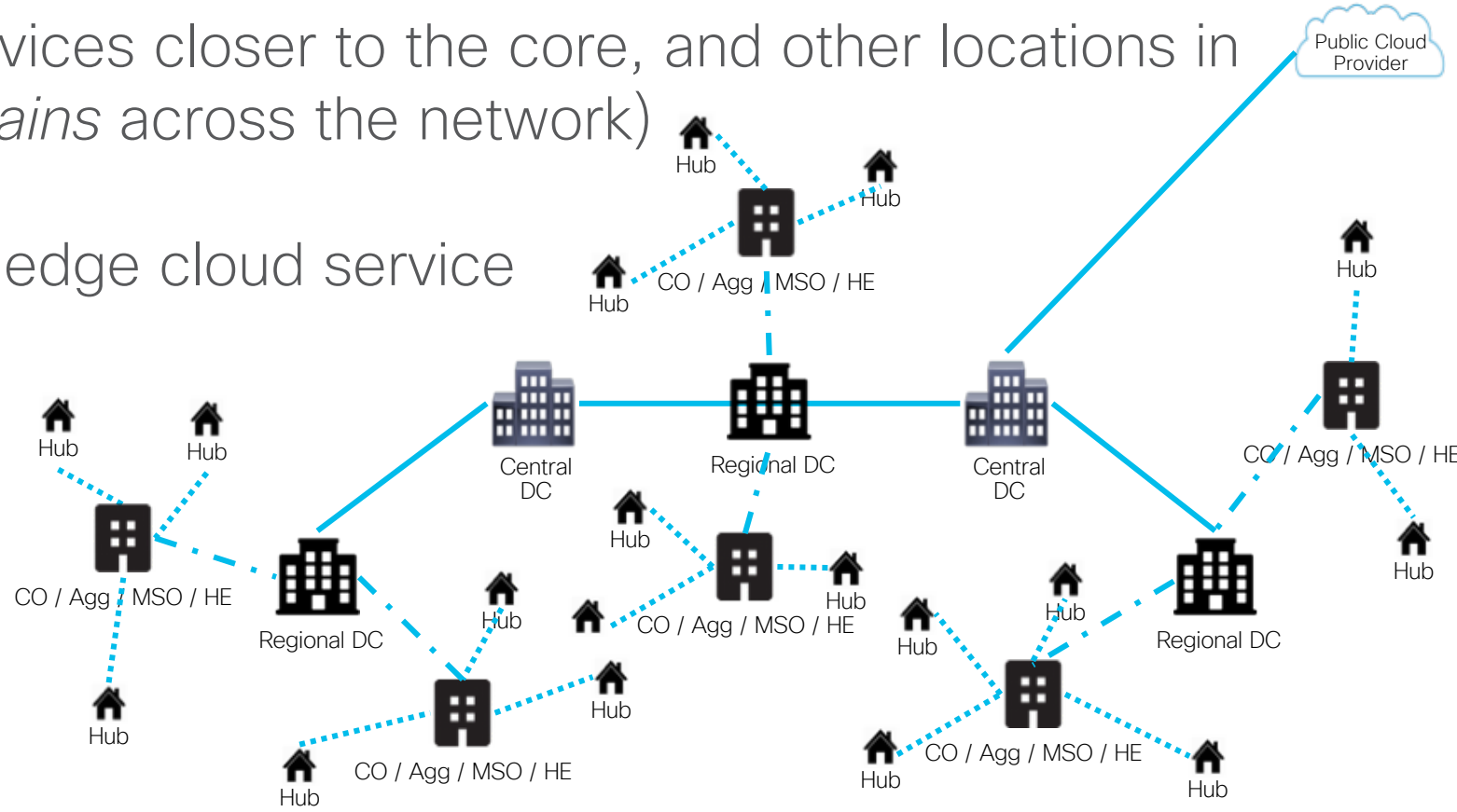
Edge Clouds

Introduction to Edge Cloud

- Edge Cloud provides access to distributed VNFs close to end user
- Why?
  - Reduces Endpoint <-> VNF connection overhead (low latencies)
  - Reduces backhaul (traffic from edge to core)
- Typical Requirements:
  - Small Footprint
  - Reduced Power Consumption
  - Seamless Integration into SP Network

Introduction to Edge Cloud – Cont.

- Why does it need to integrate with the SP network?
- Needs to integrate with services closer to the core, and other locations in the SP network (*service chains* across the network)
- Logical location defined by edge cloud service
  - Cell Site/Access
  - Hub
  - Aggregate
  - Regional DC



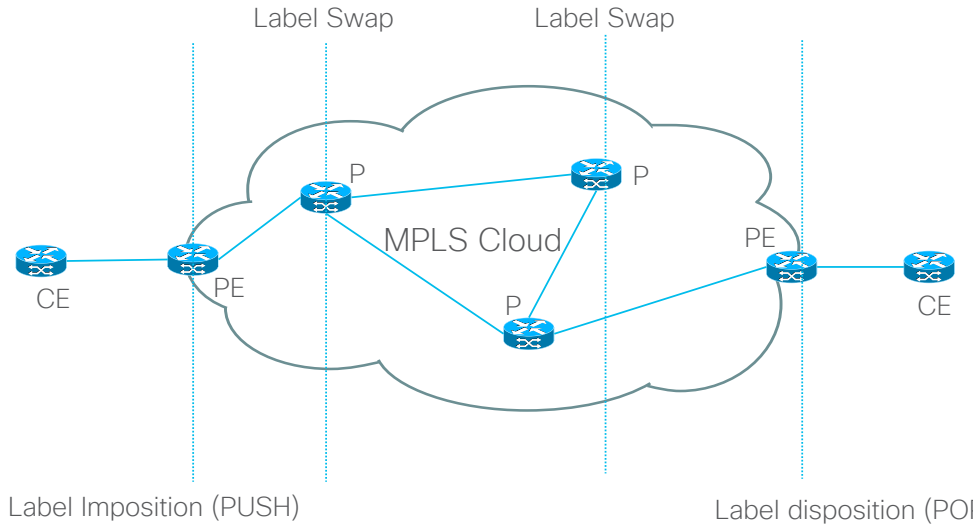
Service Provider NWs

Brief SP WAN Technology Overview

- Common WAN Technology Features:
  - Defining the “how” in reaching a destination – *the control plane protocols and interpreting them*
  - Defining an overlay independent of SP network transport – *division of responsibility*
  - Secure Separation to define multi-tenancy – *working with cloud networking*
  - Traffic Engineering – *within the responsibility of the network team*

Brief Introduction to MPLS

- Packets are labeled (MPLS Label) and switched across the MPLS cloud
  - *It's not IP*
- MPLS uses existing IP control protocols to exchange label information
- MPLS Forwarding Operations include:
  - PUSH
  - SWAP
  - POP



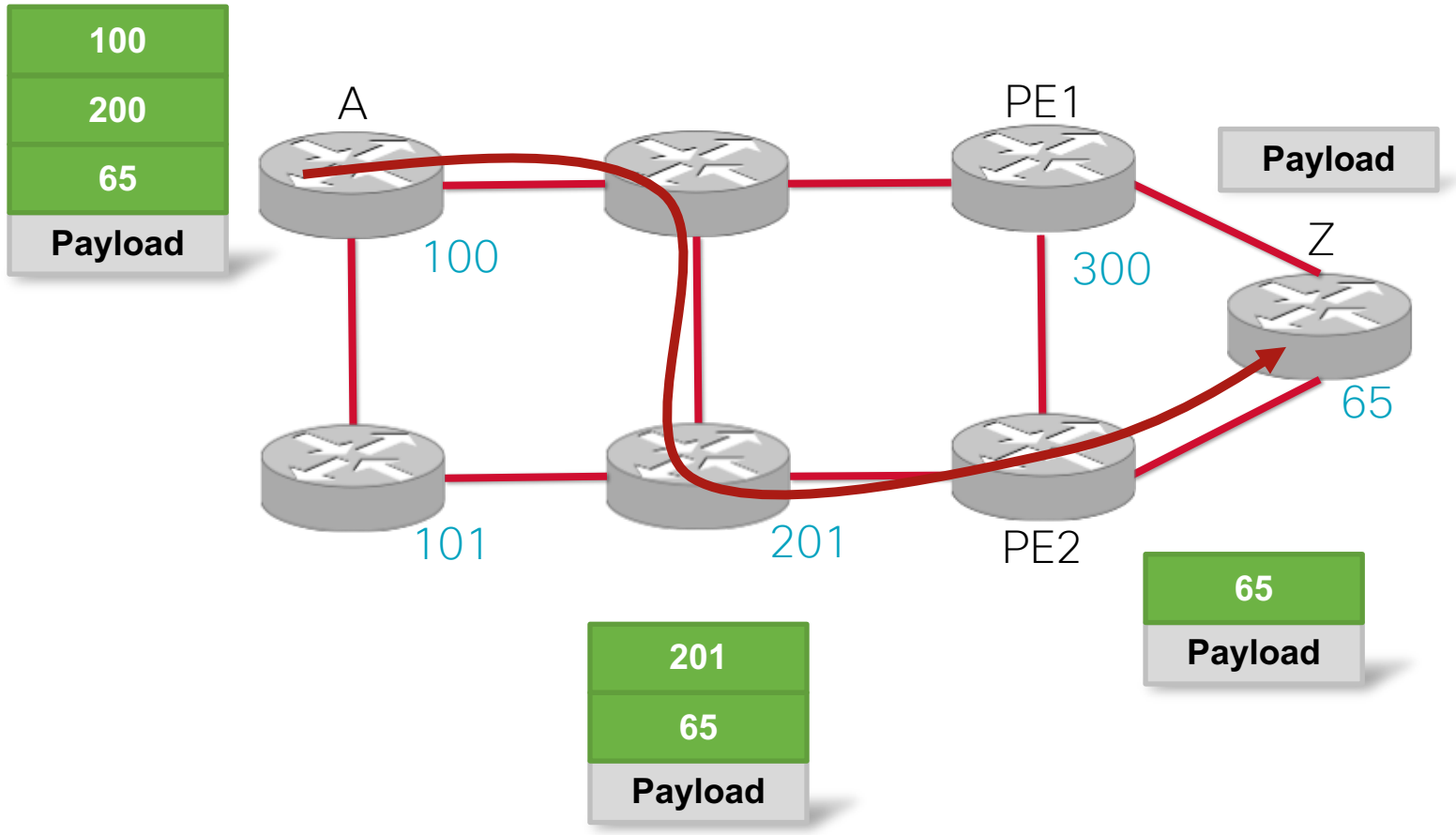
Why MPLS?

- It's hard being an SP
  - Fibre is expensive
  - Router interfaces are expensive
  - They have to get the most value out of the infrastructure they have
- MPLS helps SPs use their network as fully as possible
  - They can steer traffic down specific links to use the network more fully
  - They can reserve traffic for specific purposes like enterprise-grade connectivity

Brief Introduction to Segment Routing

- Based on Source Routing Concept:
  - Source chooses path
  - encodes it in packet header
  - network executes encoded instructions
- Two data plane instantiations (MPLS and IPv6)
  - Segment (identifier for instruction) == Labels (MPLS) vs IPv6 Address (SRv6)
- Network path is encoded in packet header; Network no longer holds state

Brief Introduction to Segment Routing



E-VPN: Next Gen. Overlay Solution

Complementary to the transport layer

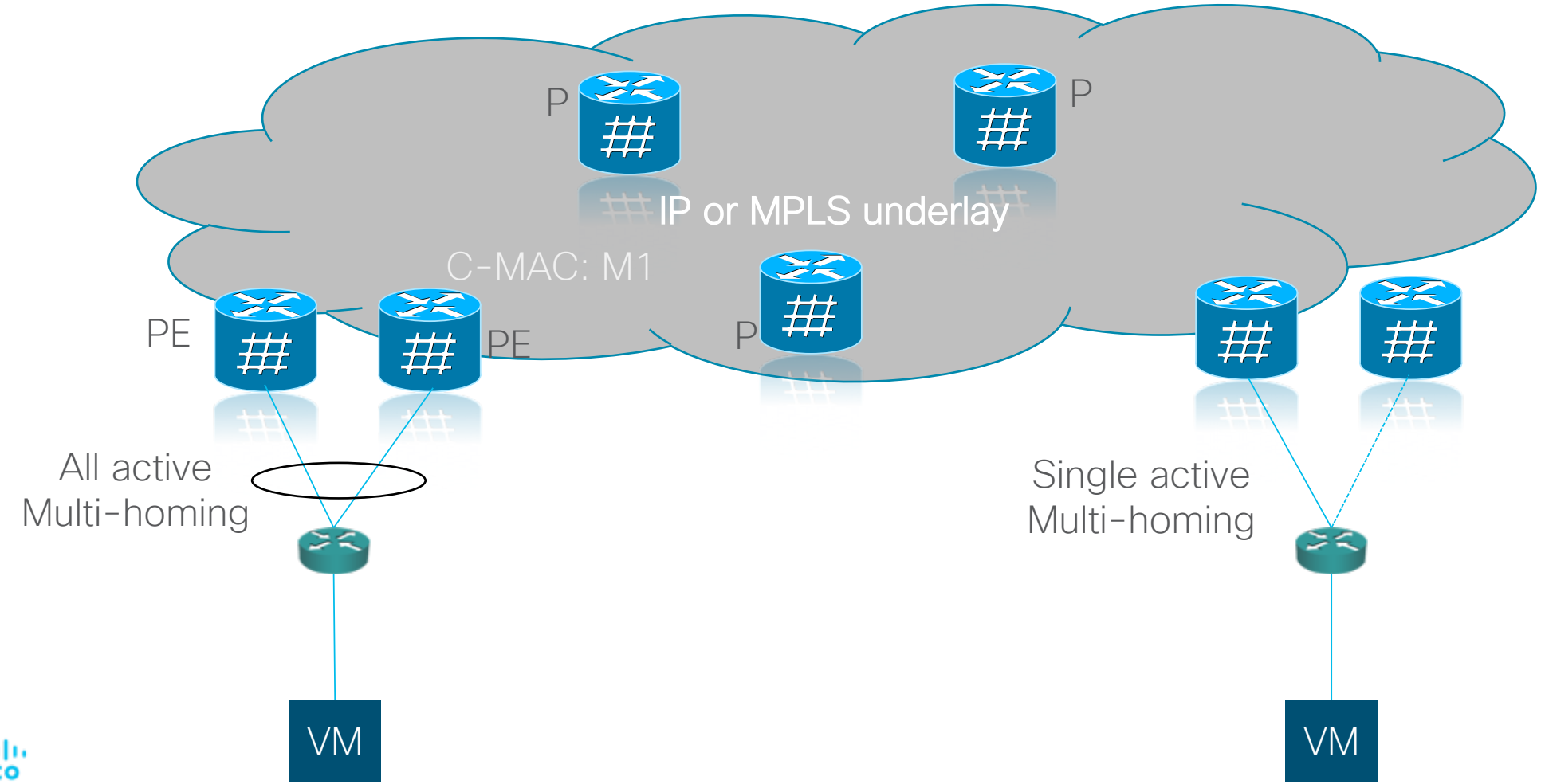
We use BGP to find out where a MAC is in the WAN

- Control Plane advertises learned MACs from CE

Addresses shortcomings:

- Network Inefficiency
- Different Operational Models
- Lack of programmability and policy control

E-VPN: Next Gen. Overlay Solution



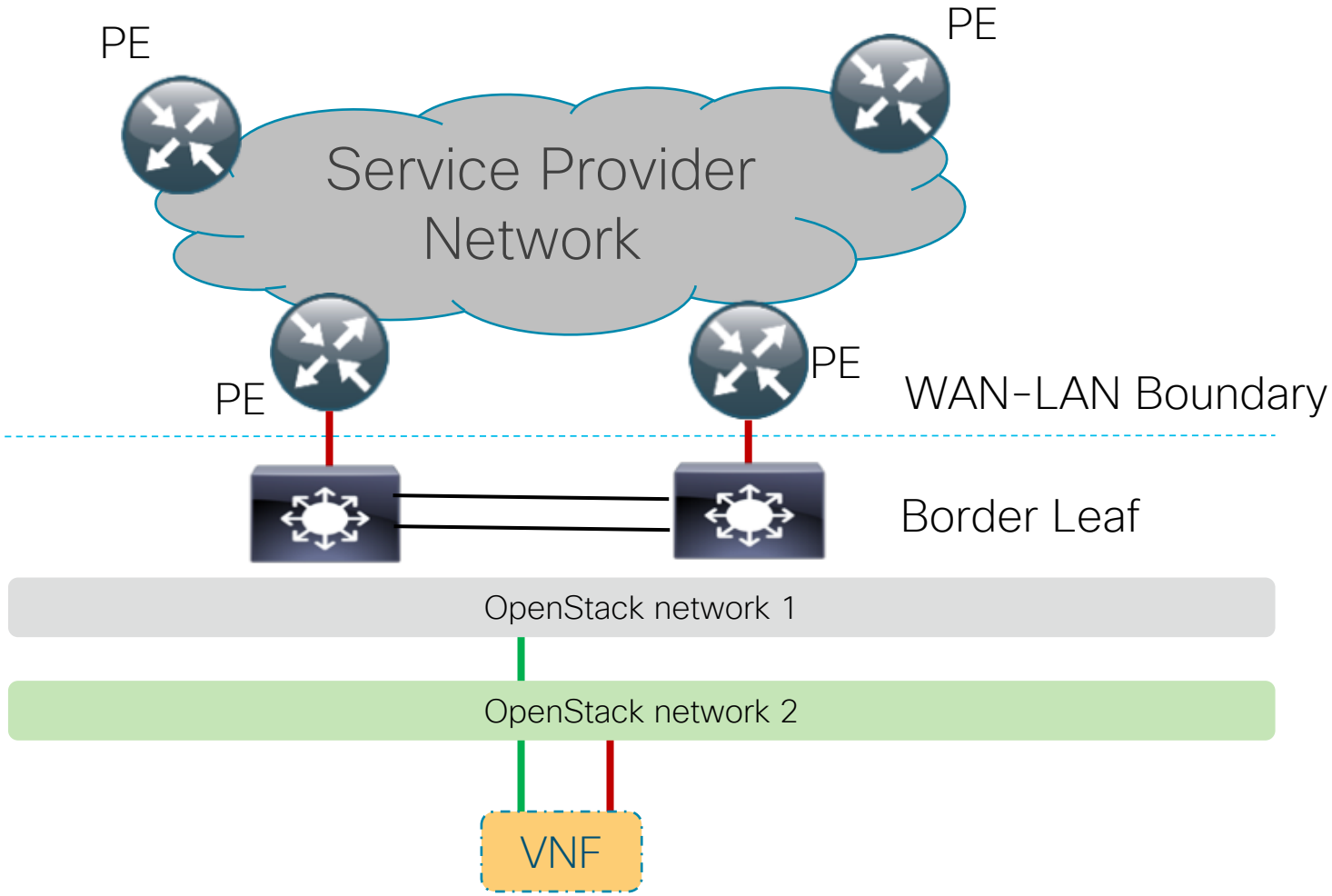
MPLS in OpenStack Today

Conventional WAN Edge Model

PEs – 'provider edge' routers – deal with moving the traffic onto the SP backbone fabric

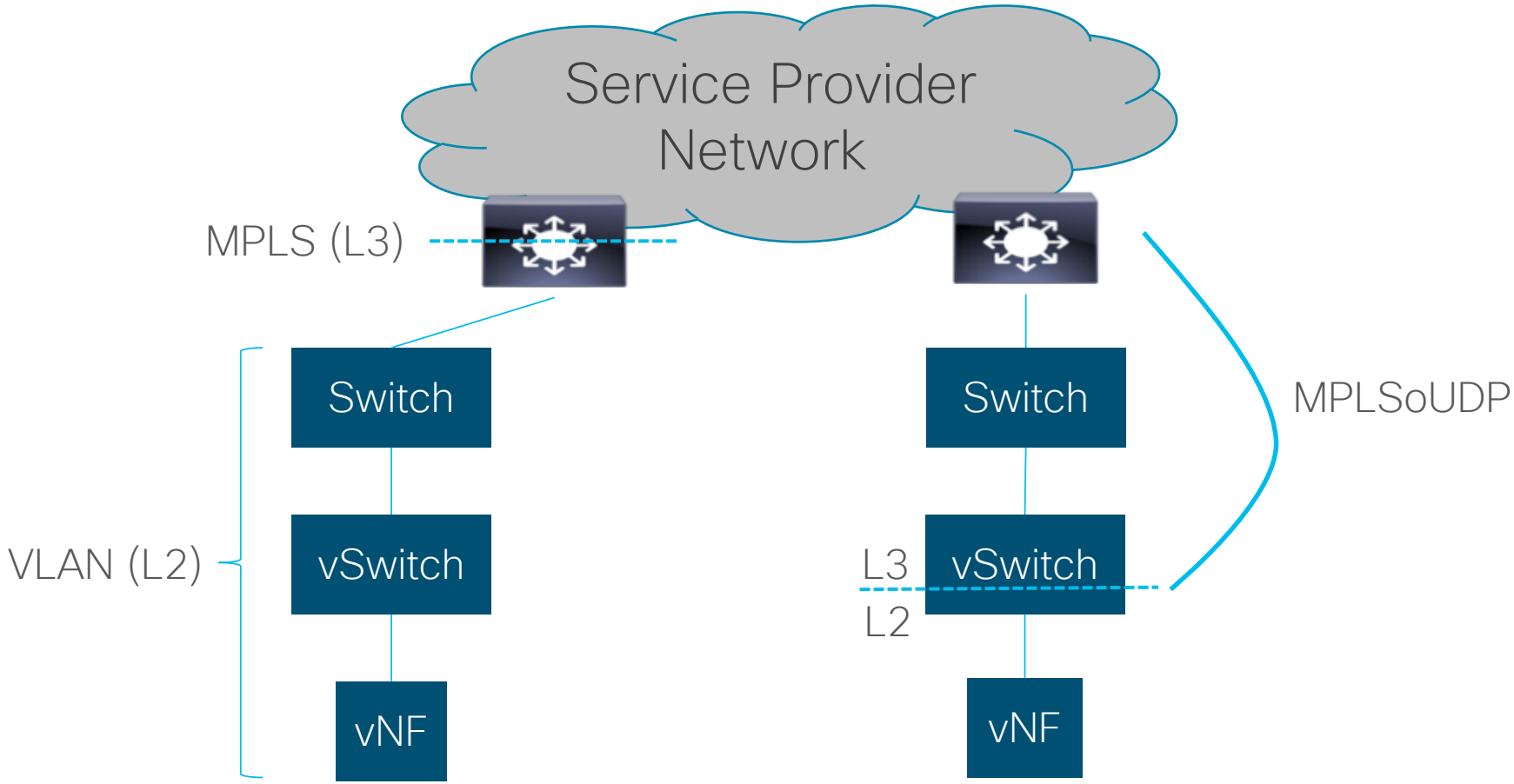
Hand-off to PEs using DC technologies and provider networks

PE converts this to MPLS or SR for use in the SP network



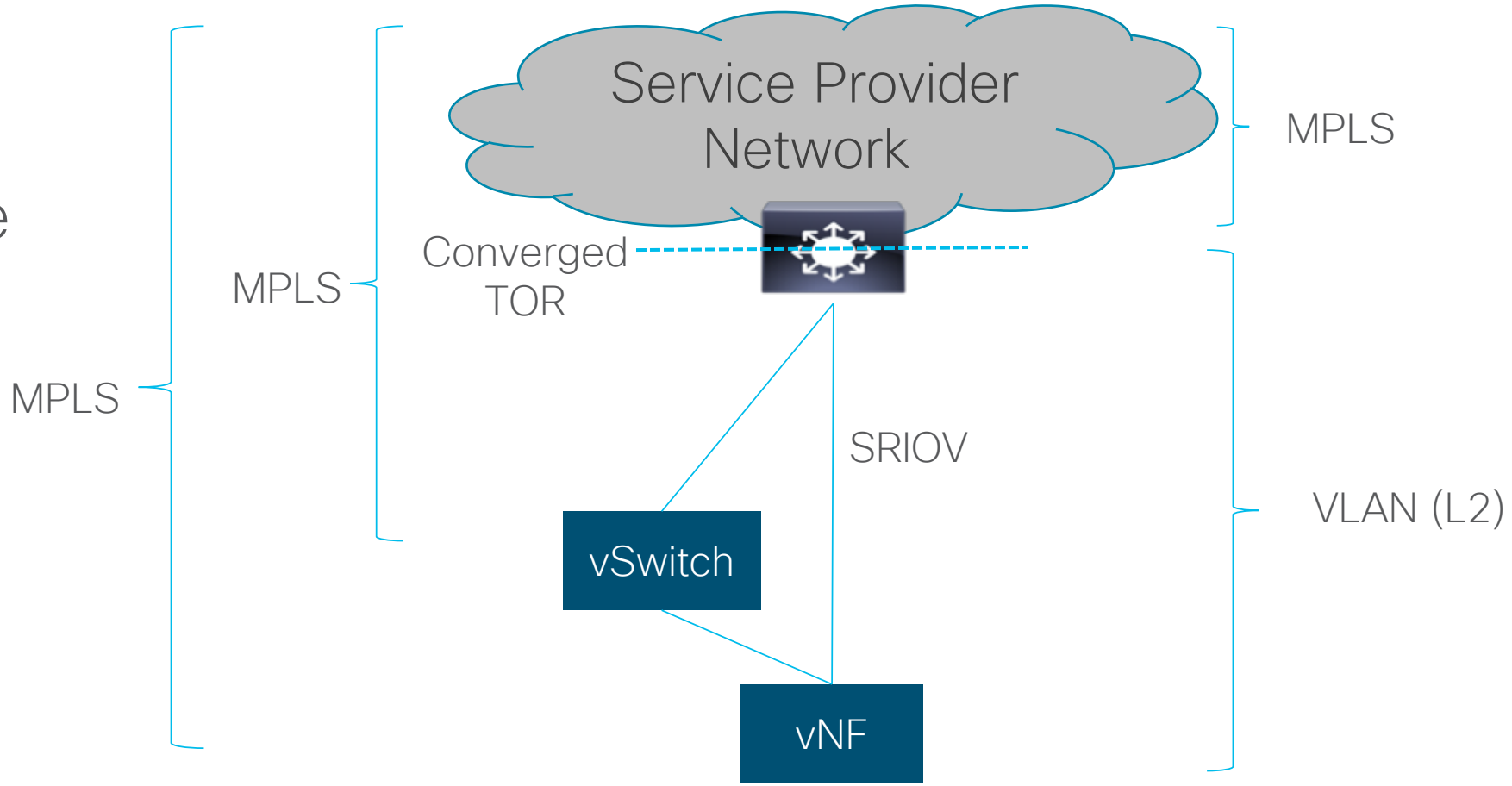
MPLS Implementations Today

What we see today



Towards an Integrated WAN-Edge Model

What we believe is needed

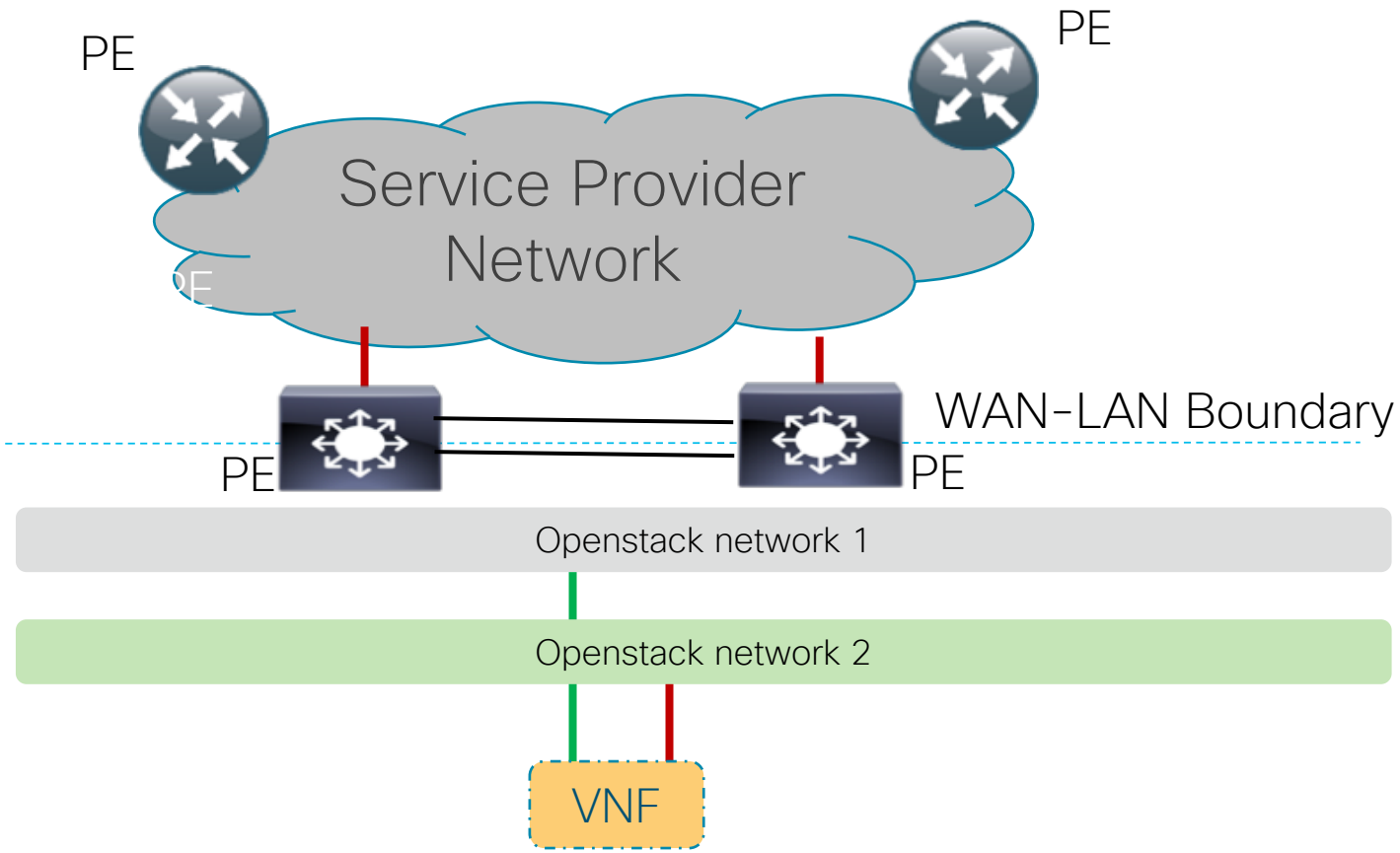


Integrated WAN-Edge Model

Service Provider PE operations are moved into the control of the Cloud SDN

Neutron now manages both L2 networking as well as PE VPN functionality

Hybrid TOR/Router: SP + L2 capabilities within a single entity

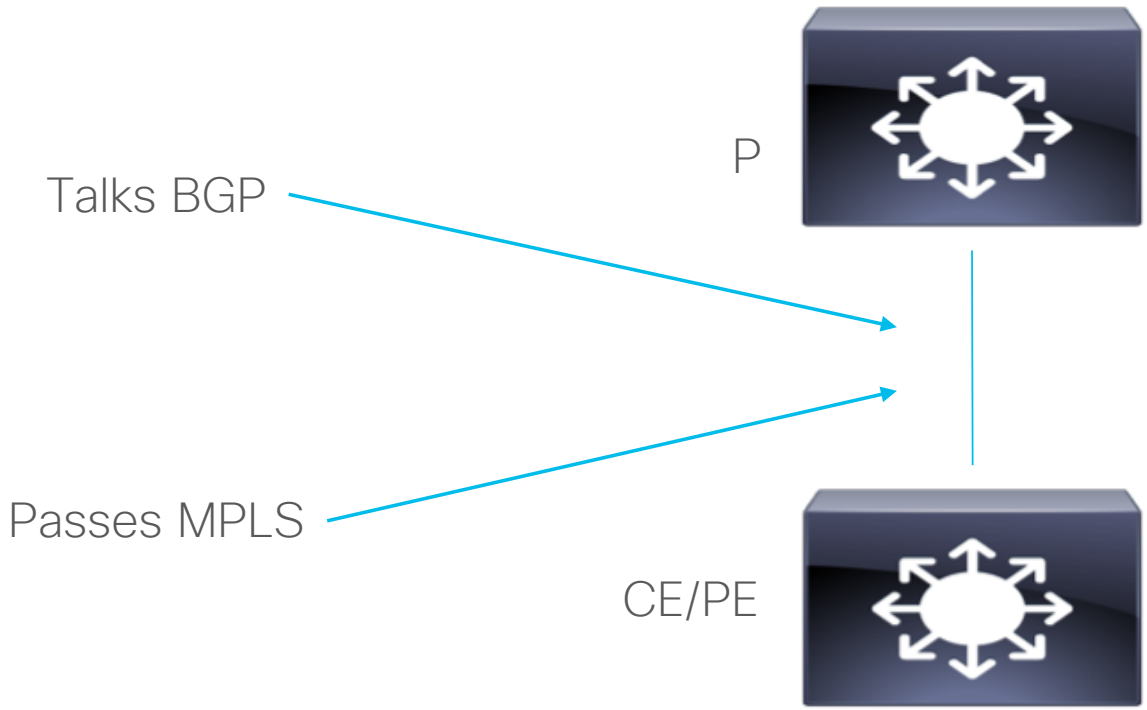


Benefits of Integrated WAN-Edge Model

- Physical Elements go away --> less space, power and cost!
- More possibilities for networking
- Consistent MPLS end-to-end (OAM, TE, etc.)

SP WAN Operations in a cloud world

- Clearly divided responsibility separation between WAN and Cloud teams
  - but not the way they used to be divided



WAN operators provide BGP peering details

BGP policies to *protect* WAN integrity

Cloud controls CE/PE elements

OpenStack BGP-VPN API

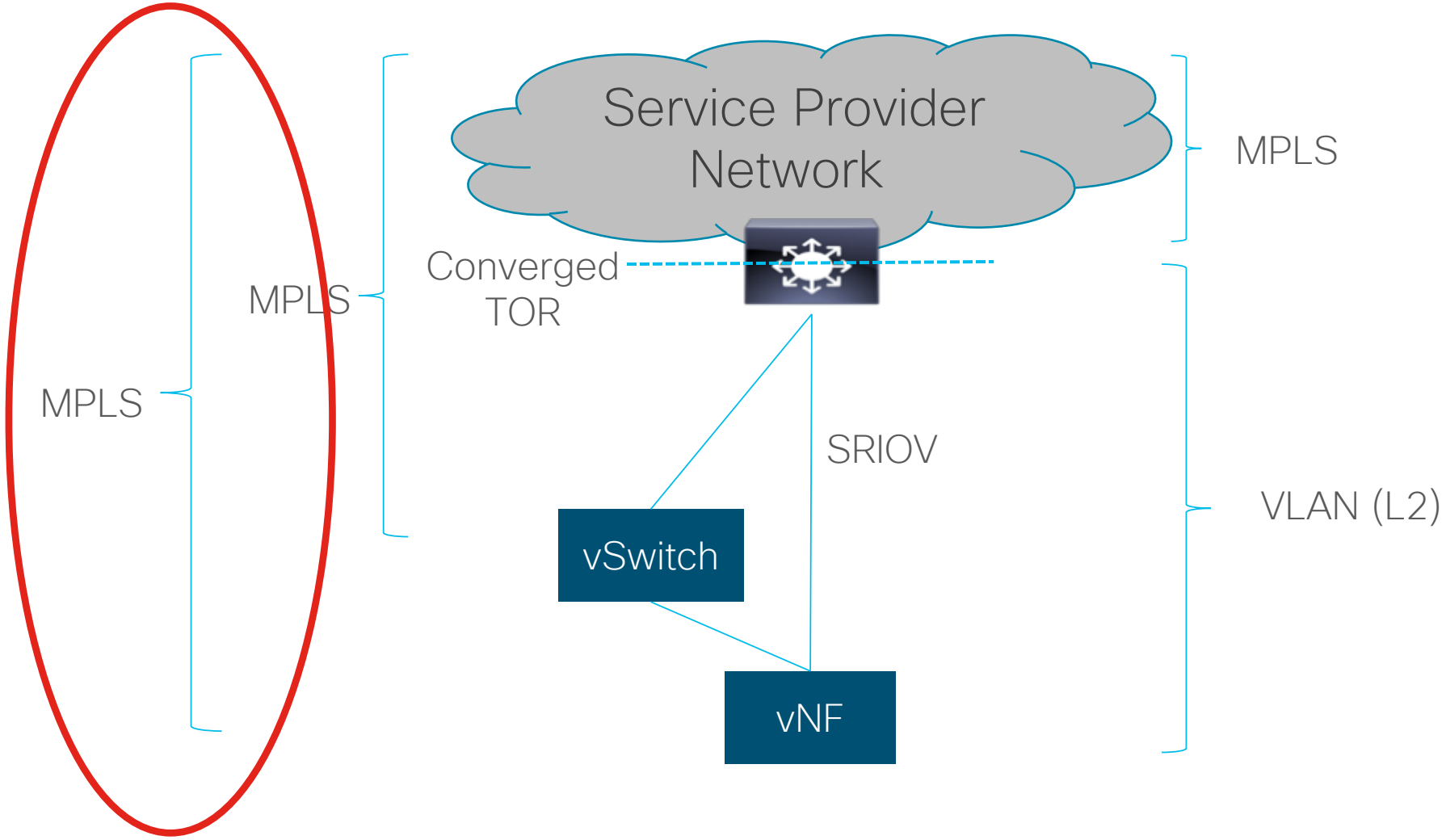
API versus implementation

- The BGP VPN API today describes network overlays and how they attach to OpenStack networking
- It is *completely independent* of the implementation
- Changing the implementation as described above doesn't need us to change the APIs
  … but we can extend the APIs to add new functionality that we can now offer

WAN-Edge models: a recap

This is a new model

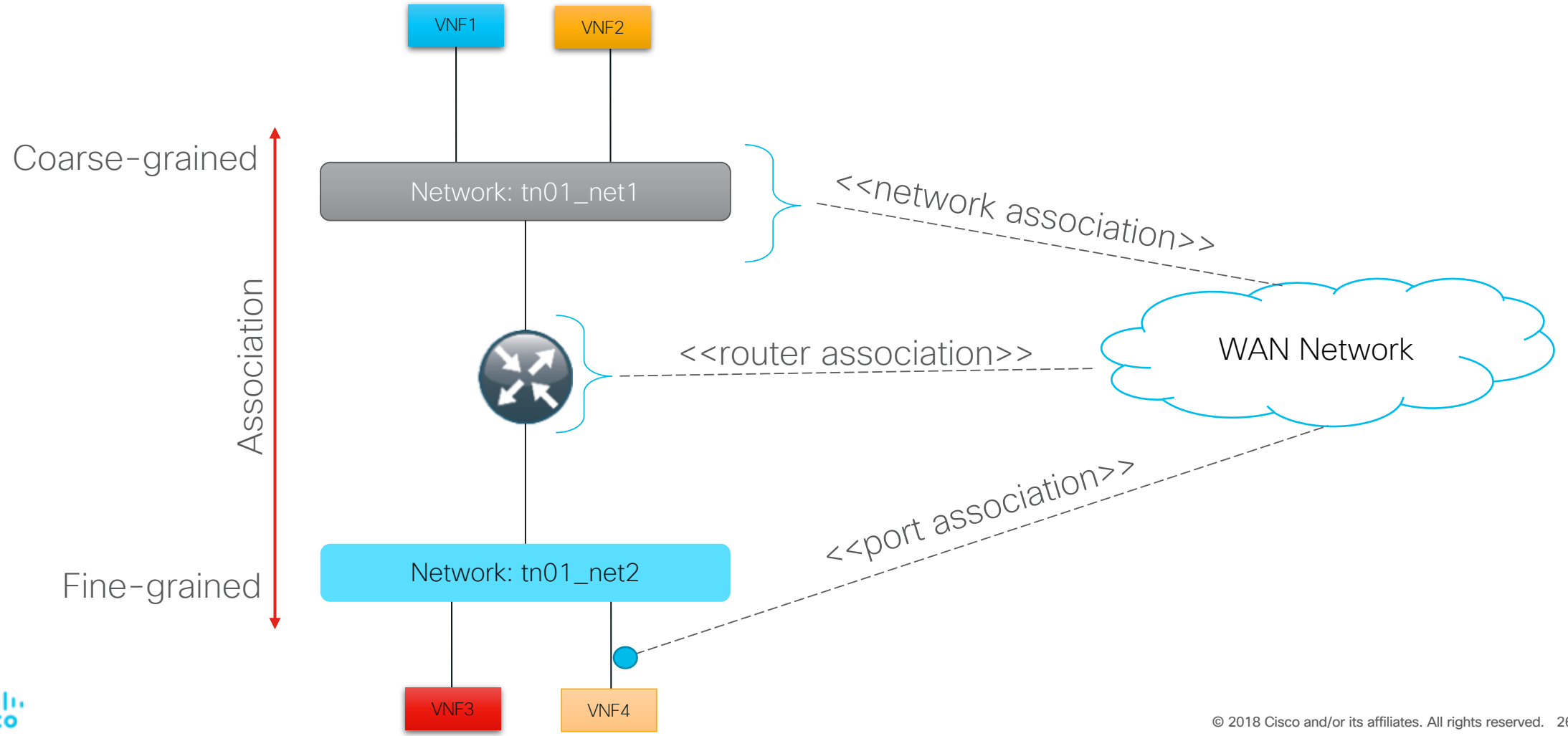
To pass MPLS labels to VMs, we need to find new ways to describe what we're doing



Modelling API for WAN <-> Edge Cloud

- VPN modelled as abstract resource
- Admin/Operator defines key VPN properties
  - Route Target
  - Route Distinguisher
  - VNI etc.
- Tenant associates VPN with Neutron object
  - Network
  - Router
  - Port

Modelling API for WAN <-> Edge Cloud – Cont.



Uniform Representation of Connection Types

Common API to configure different types of connections across a WAN

L2 Overlays

- Represents a switched connection between two endpoints
- Mostly applicable for DCI

L3 Overlays

- Represents a routed connection

Point-to-Point Overlays – *a new connection type*

- Used to define a service chain of VNFs located at different locations

Our thoughts

- Changing APIs is hard – *agreeing* APIs is hard
- We want to do further experiments with implementation to ensure we have an API model that works *before* we propose the details
- Watch this space

Our Implementation

Our Implementation

Edge Cloud defines a set of new requirements:

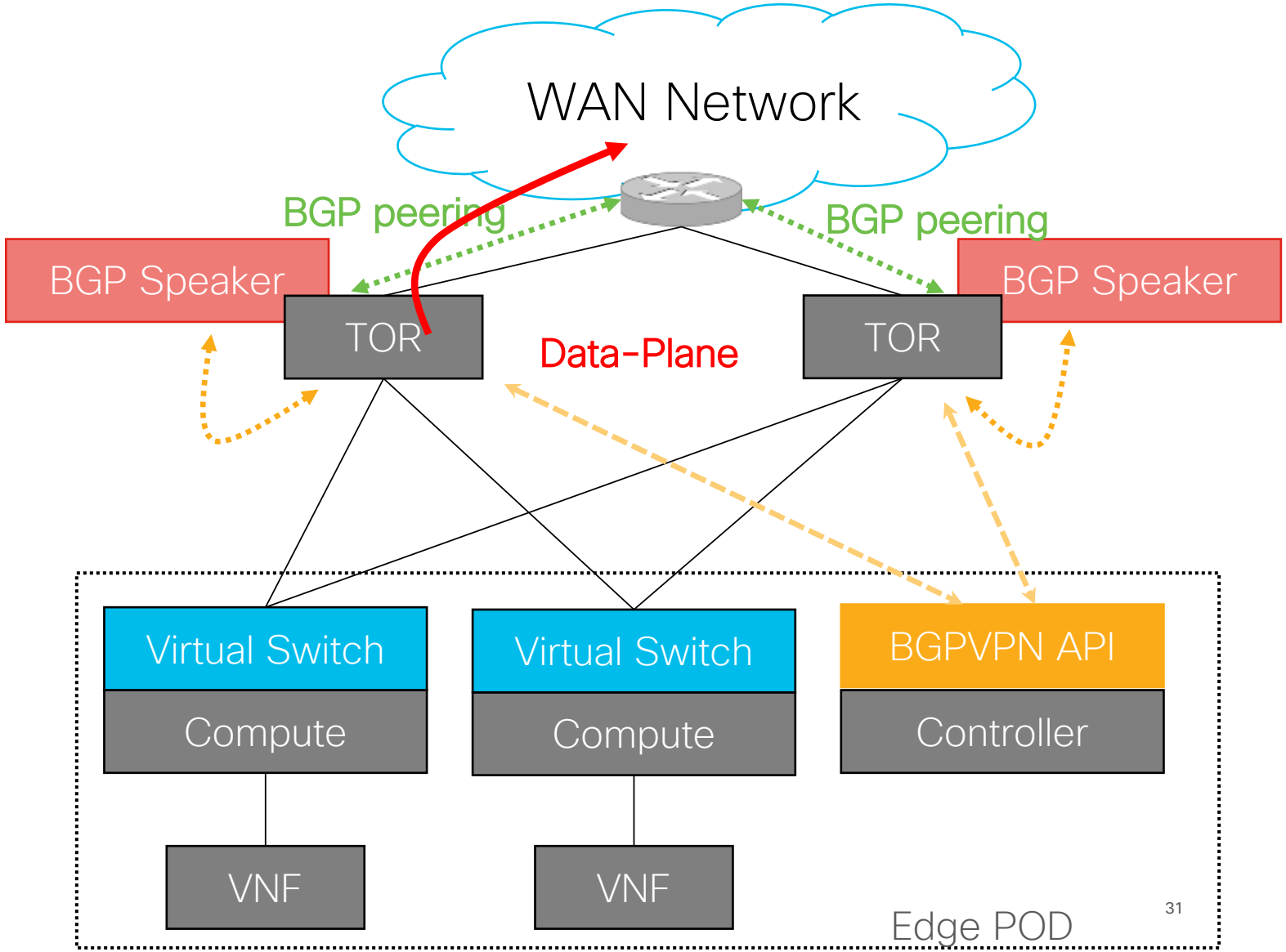
- Maximize Compute Resource footprint while keeping physical network resources to a minimum
- Minimize power/space consumption

How can we realize the Edge Cloud requirements?

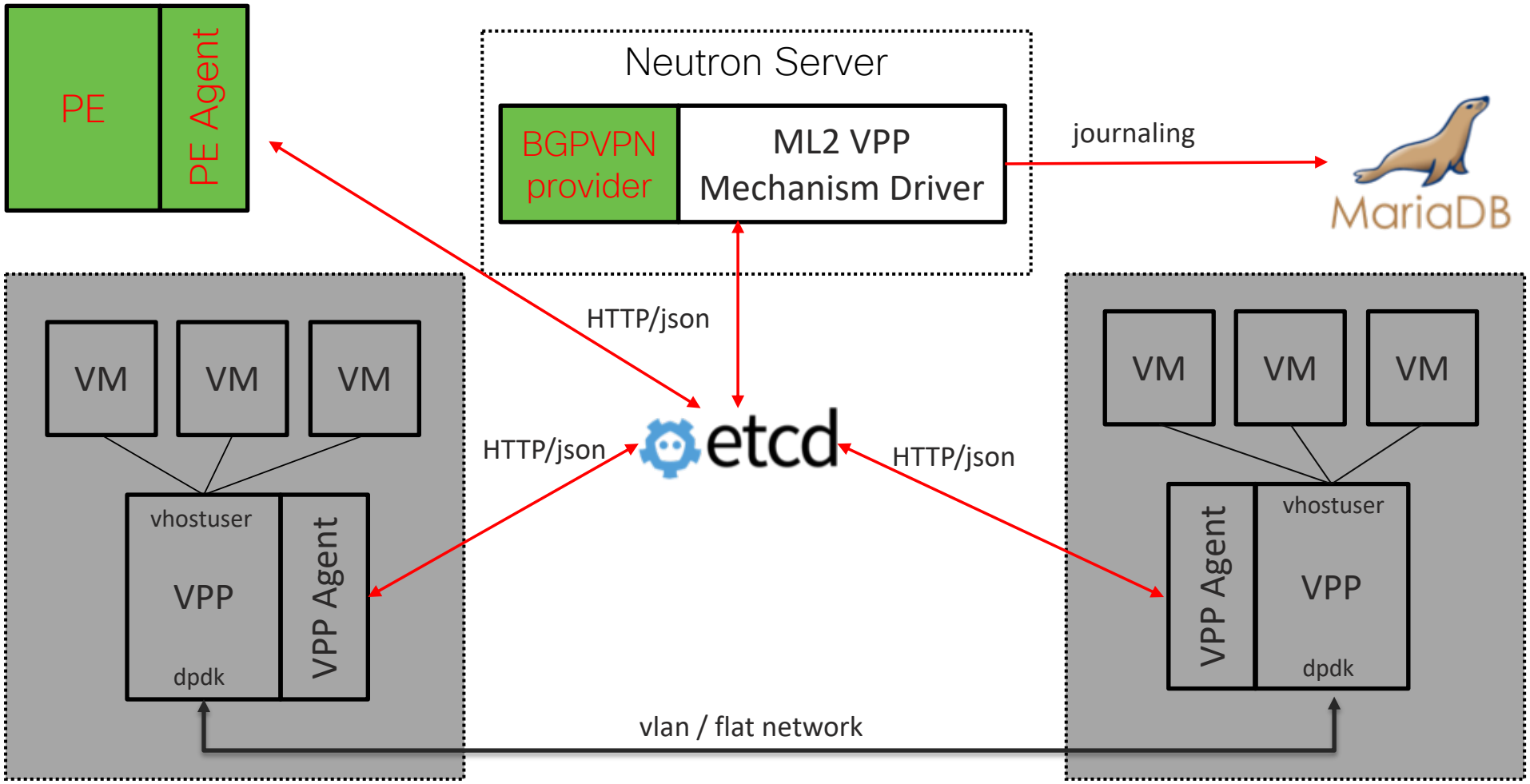
Converged TOR Implementation

BGP Implementation details are handled on converged physical TOR

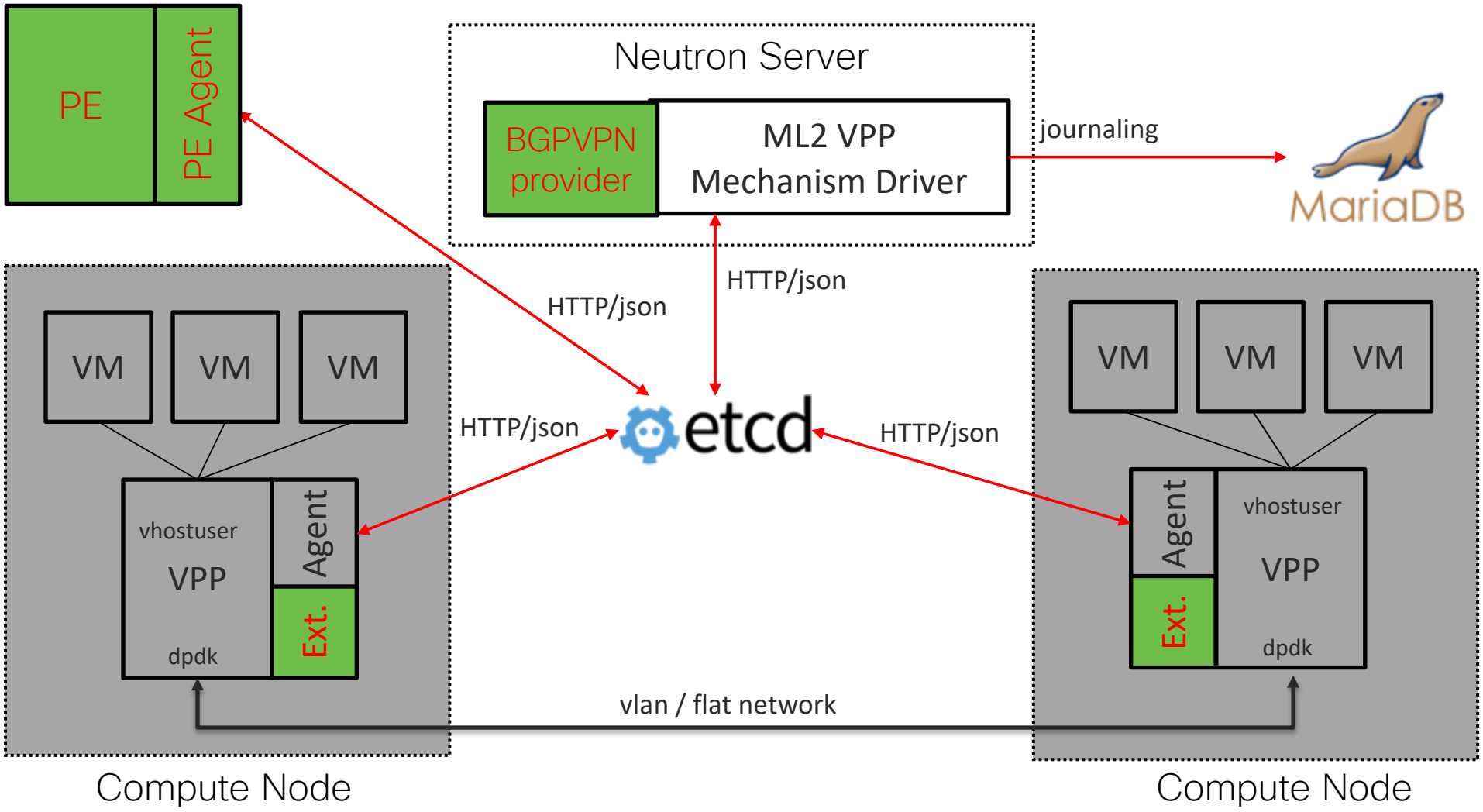
OpenStack control plane stays with controller node



Our networking-vpp implementation



Our networking-vpp implementation



Mapping API Constructs

Hardware Router acting as collapsed TOR; controlled by BGP VPN API

Configuration Snippet for router reflect API elements:

- Tenant Network <-> EVPN EVI and BD
- VPN (RD & RT) <-> NONE
- Tenant Router <-> VRF
- Tenant Router attached to Network <-> BVI within Router VRF
- Router associated to VPN <-> RD and RTs configured & VRF defined in BGP

An Example Implementation – Cont.

OpenStack BGPVPN API Operation → Results in → Hardware Router Configuration

1 Create VPN construct and define RD and RT

```
neutron bgpvpn-create --name VPN1 --route-targets
    16000:100 --route-distinguisher 16000:500
```

2 Create OpenStack Router and attach to Network

```
neutron router-create VPN-ROUTER; neutron
    router-interface-add VPN-ROUTER VPN-SUBNET
```

3 Associate Router to VPN

```
neutron bgpvpn-router-assoc-create VPN1
    --router VPN-ROUTER
```

1 No configuration applied on Router

2 Define VRF and BVI

```
vrf <tenant-router-id>
   description <tenant-VRF>
interface BVI 100
   vrf <tenant-router-id>
   ipv4 address 100.0.0.1/24
```

3 Fill-in VRF and Define BGP Configuration

```
vrf <tenant-router-id>
  description <tenant-VRF>
  address-family ipv4 unicast
    import route-target
      16000:100
    export route-target
      16000:500
router bgp <BGP-AS>
    vrf <tenant-router-id>
      rd 2704:5000
      address-family ipv4 unicast
        label mode per-vrf
        redistribute connected
```


And over to you...


CISCO