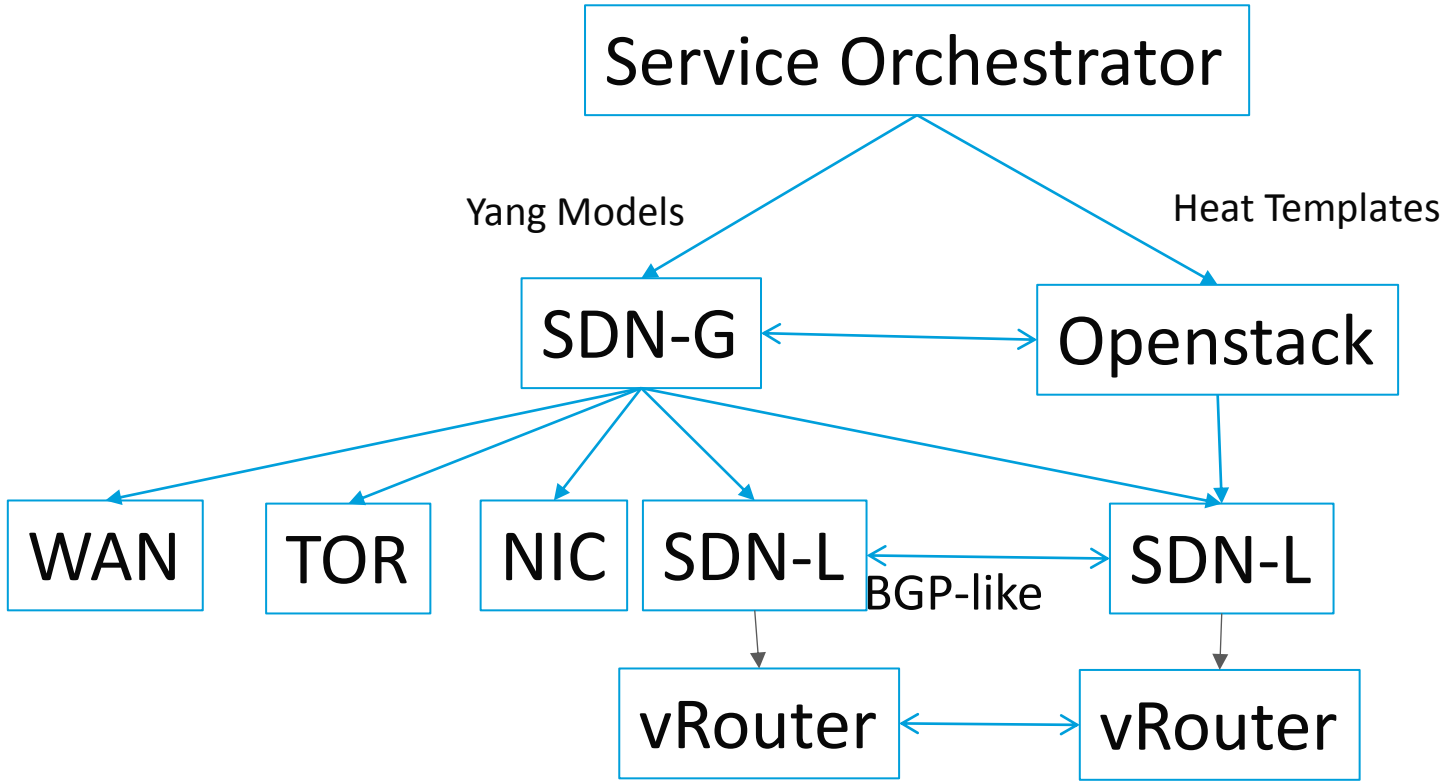
Gluon: An Enabler for NFV

Bin Hu
PMTS, AT&T

Ian Wells
Principal Engineer, Cisco

Ildikó Váncsa
OpenStack Coordinator, Ericsson

Controller Relationships



SDN-G – SDN Global Controller
SDN-L – SDN Local

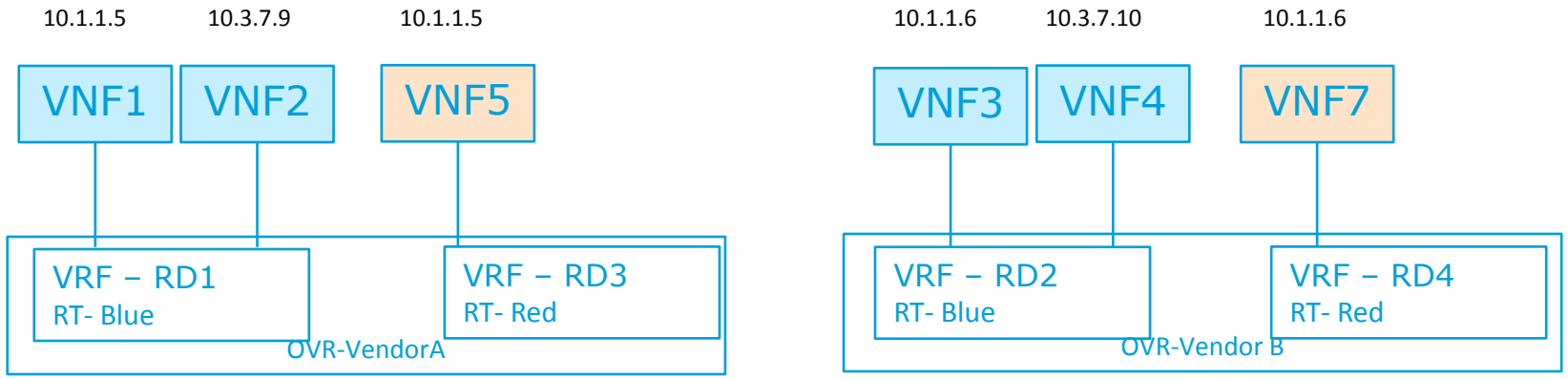
Overall Scenario

- Common Data Plane (MPLS/GRE, MPLS/UDP, VxLAN)
- Common Control Plane (EVPN – IRB, L2, L3; L3VPN)
- Common Configuration Parameters
- Common Openstack-Controller Interface

Any to Any Base Case

Tenant 1
10.1.1.0/24
10.3.7.0/24

Tenant 2
10.1.1.0/24



Example Neutron/Network API Calls (multi-vendor OVR and one OVR per host)

1. Create Network
2. Create Network VRF Policy Resource
    1. This sets up that when this tenant is put on a HOST that:
        1. There will be a RD assigned per VRF
        2. There will be a RT used for the common any-to-any communication
3. Create Subnet
4. Create Port (subnet, network vrf policy resource)
    1. This causes controller to:
        1. Create vrf in vRouter's FIB, or Update vrf if already exists
        2. Install an entry for Guest's HOST-Route in FIBs of Vrouters serving this tenant Virtual Network
        3. Announce Guest HOST-Route to WAN-GW via MP-BGP

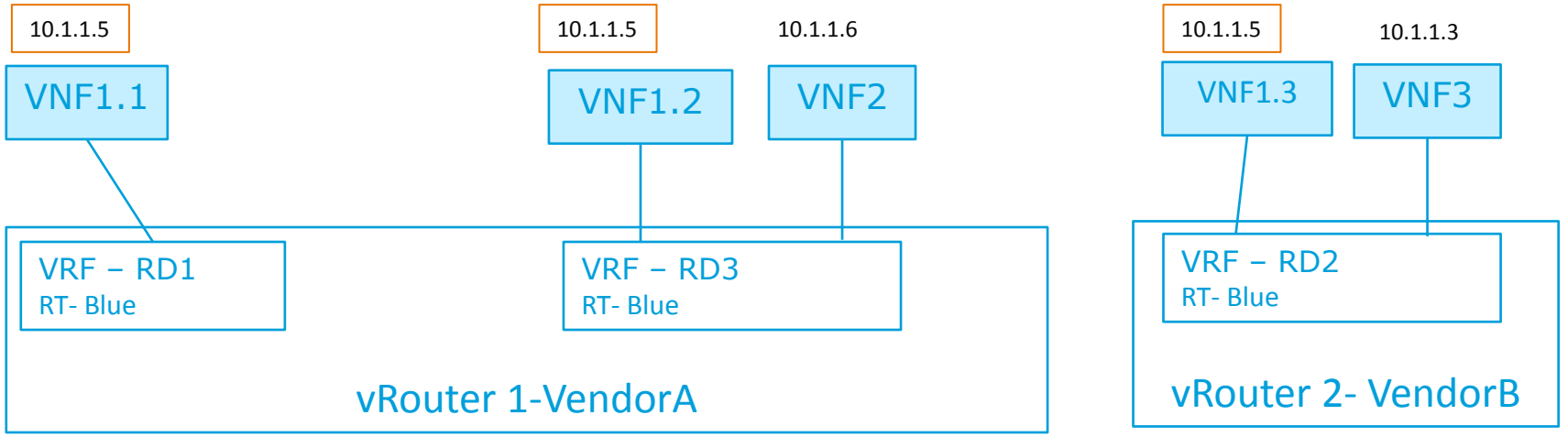
VRF Lets us do:

1. Overlapping Addresses
2. Segregation of Traffic

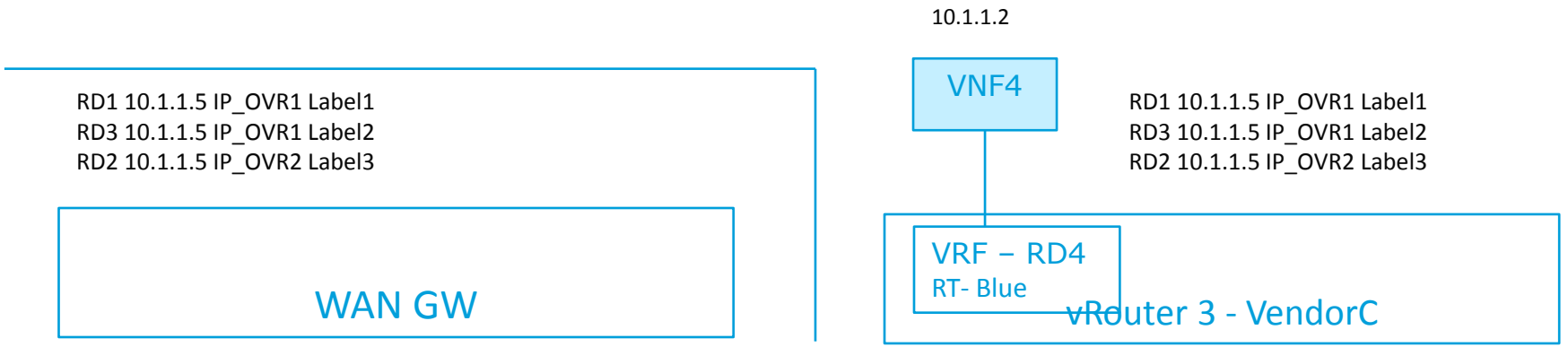
ECMP Load Splitting Case – AnyCast

Tenant 1
10.1.1.0/24

INGRESS Does load split regardless of Host and
Regardless of external (from WAN GW) or internal (from VNF4)
Need separate RD for any cast end point to segregate traffic

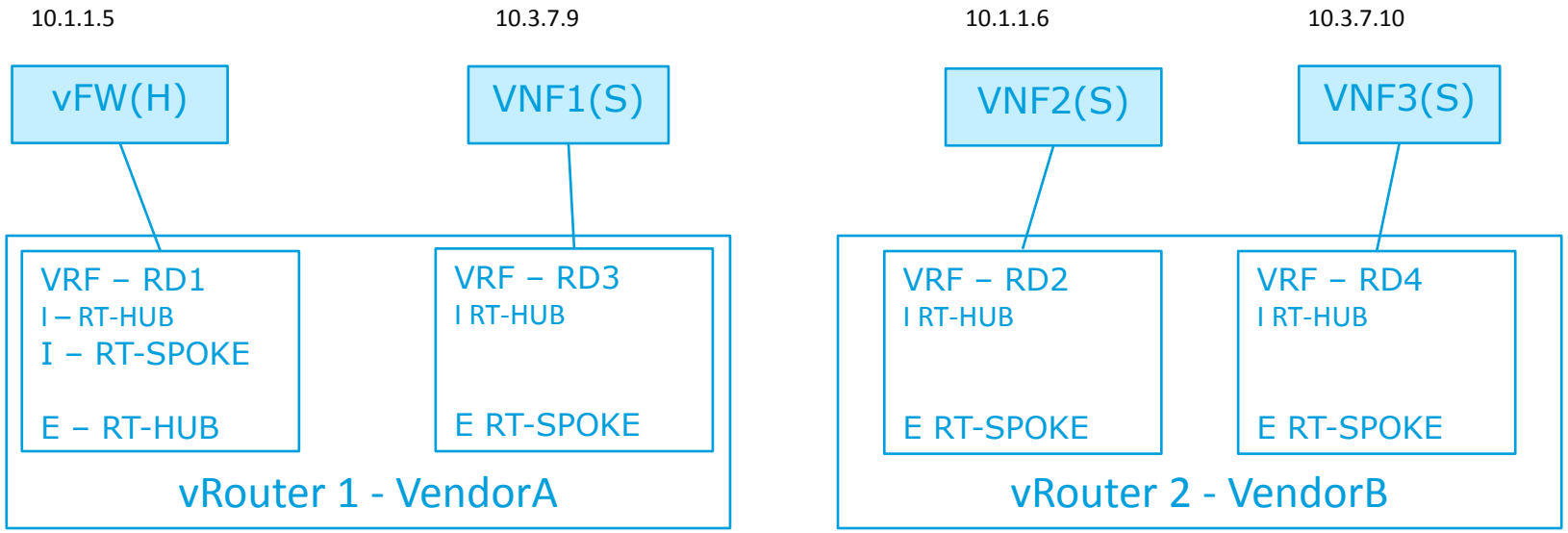


Traffic to Anycast 10.1.1.5 can be load split from either WAN GW or another VM like G5



Hub and Spoke Case

Tenant 1
10.1.1.0/24
10.3.7.0/24



G1 Hub VRF
RD1 10.1.1.5 IP_OVR1 Label1
RD1 0/0 IP_OVR1 Label1
Label 1 Local IF (10.1.1.5)
RD3 10.3.7.9 IP_OVR1 Label2
RD2 10.1.1.6 IP_OVR2 Label3
RD4 10.3.7.10 IP_OVR2 Label3

G2 Spoke VRF
RD1 0/0 IP_OVR1 Label1
RD3 10.3.7.9 IP_OVR1 Label2

Neutron/Network API Calls
1. Create Network
2. Create VRF Policy Resource
   - Any to Any
   - Any to Any w/ ECMP
   - Hub and Spoke (Hub, Spoke)
3. Create Subnet
4. Create Port
   - w/ Subnet
   - w/ VRF Policy Resource, [H | S]

Use Cases

- Multiple Servers running different vendor controller & overlay routing software for L3/L2 VPNs – MPLS VRFs
- Multi-vendor routers in one subnet
- Sharing of common control plane parameters – BGP communities (e.g. Route Targets (RT)) across the multi-vendor controllers
- Controllers and routers communicate directly without going through any gateway for east-west traffic

- Use case above just examples of how telco landscape is rapidly changing and the need of:
  - Agile method to enable new use cases in telco market
  - Accelerated time-to-market of launching new services to telco customers
  - Enhanced business agility to benefit telco ecosysrem
  - Flexibility of technology to implement new use cases
  - Common APIs with diversified backend implementations

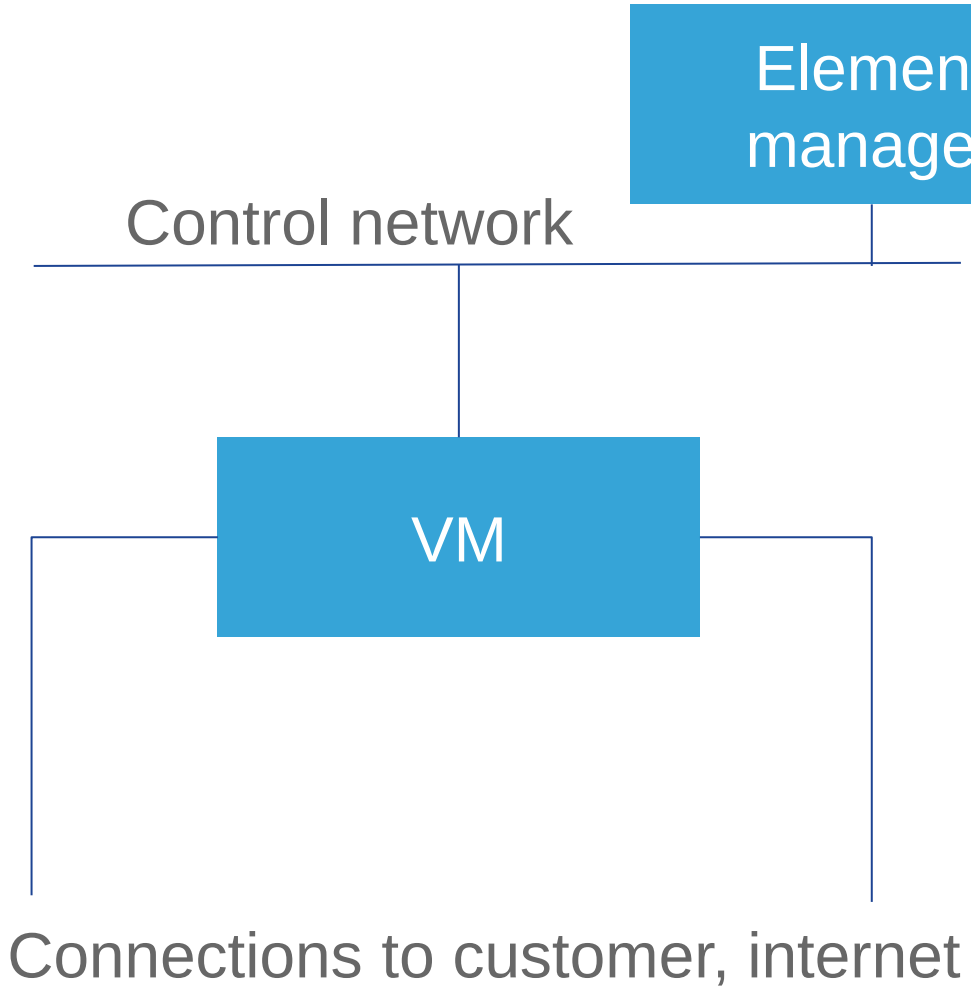
Our software requirements

- We want to use Neutron
  - Neutron is still key to what we do
- We want to use APIs that are increasingly unlike Neutron
  - MPLS is a L3 domain between ports, in reality
- We want clear water between the API and the backend
  - These use cases all use a dedicated SDN controller, so the cloud API need only feed that SDN controller
- We want to try new things fast … and fail fast
- We want to use multiple APIs and backends simultaneously

Gluon: the core concept

- Gluon's aim is to connect network service providers with VMs
  - Neutron is one of those providers - but it doesn't have to be the only one
- New API endpoints can be written for new networking concepts, as long as they share the idea of a 'port' to which a VM can be attached
  - The rest of the structure can be completely different – so, VRFs instead of networks, no subnets, different addressing systems
- Ports are registered with Gluon on creation and Gluon helps Nova to talk to the right API at bind time

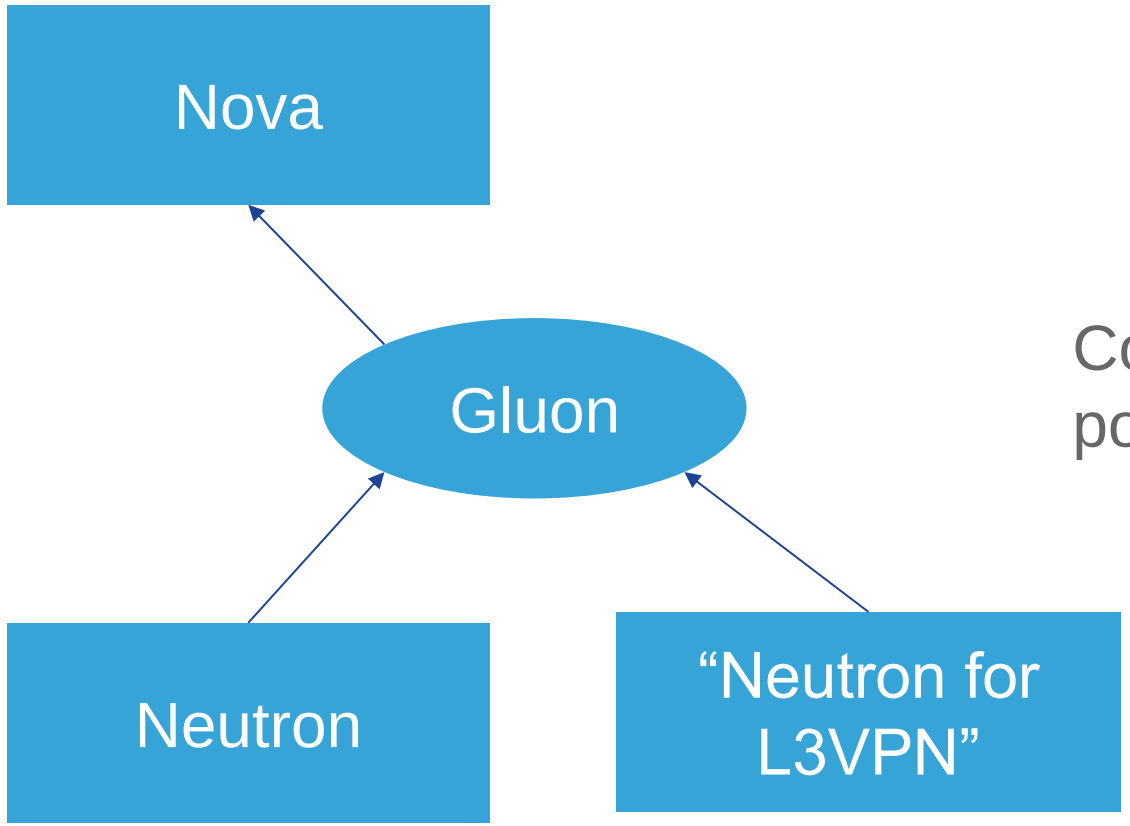
Typical VM setup in NFV



Low bandwidth control traffic – where we use Neutron

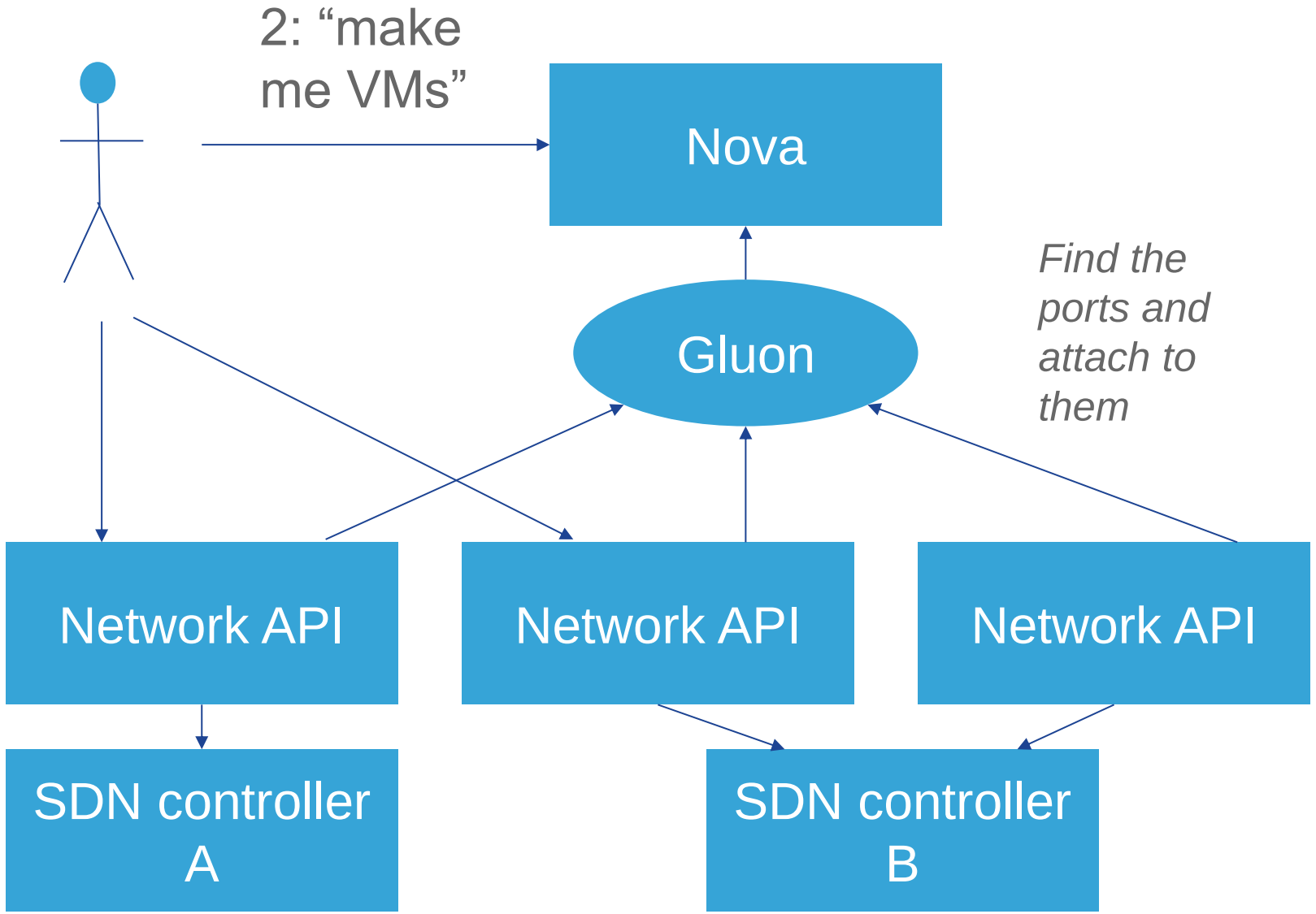
High bandwidth 'work' traffic - where we use NFV centric APIs

Gluon



Communication via ports alone

Gluon



What did we change?

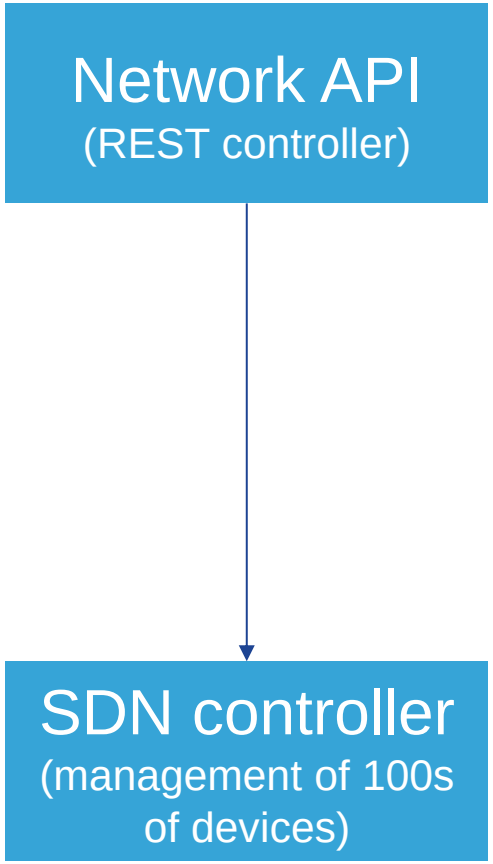
- Nova: replaced the Neutron network plugin with a Gluon plugin
  - … and fixed a minor bug
- Neutron: added two lines of code to register a port
- Gluon: a little helper that sits between the two services and proxies their communications
- An API – more on that below

The implications

The APIs: simple REST models
Code is a web service – fast request-response
Here, we share API constructs (e.g. the basics of a port) and base code (lots of boilerplate)

The protocol: synchronise desired state from the API objects to the network controller (big problems: fault tolerance, asynchronicity; solve them once and well)

The controller: a choice of implementation
Code is event driven - doing hundreds of things at once
Support common implementations, frameworks



Network APIs – ‘protons’

- Now we can add new APIs, but surely it takes a long time to write them?
- Well, actually…

Using a particle generator, 101

- Write a model of your API
- Use ParticleGenerator.py to read it
- Get:
  - A REST API with validation
  - A DB schema
  - A backend communication system

```
VPNPort:
    attributes:
        id:
            type: 'ProtonBasePort'
            required: True
            primary: True
        vpn_instance:
            type: 'VpnInstance'
            required: True

VpnInstance:
    attributes:
        vpn_instance_name:
            required: True
            type: string
            length: 32
        description:
            type: string
            length: 255
        ipv4_family:
            type: VpnAfConfig
        ipv6_family:
            type: VpnAfConfig
        route_distinguishers:
            type: string
            length: 32

VpnAfConfig:
    attributes:
        vrf_rt_value:
            type: string
            length: 32
        vrf_rt_type:
            type: enum
            values:
                - export_extcommunity
```

On multiple SDN controllers

- One aim here is to have multiple controllers do different tasks in a network
  - If I'm implementing a network today, I either have to find a controller that does all of what I want or miss some features I want to use
  - But these tasks are increasingly specialised
  - With Gluon, I can use more than one SDN controller and use them to deliver different features

Our hopes

- More innovation
- More simplicity – the individual API endpoints are much simpler because they only have to do one thing (Neutron included, we hope)
- More choice – I don’t have to find one tool that does everything I want

What are the risks we run?

- More proliferation – everyone writing their own different API for the same type of networking
- Less quality – there's no one implementation everyone's working on

Consider the IETF approach: 'rough consensus and working code'

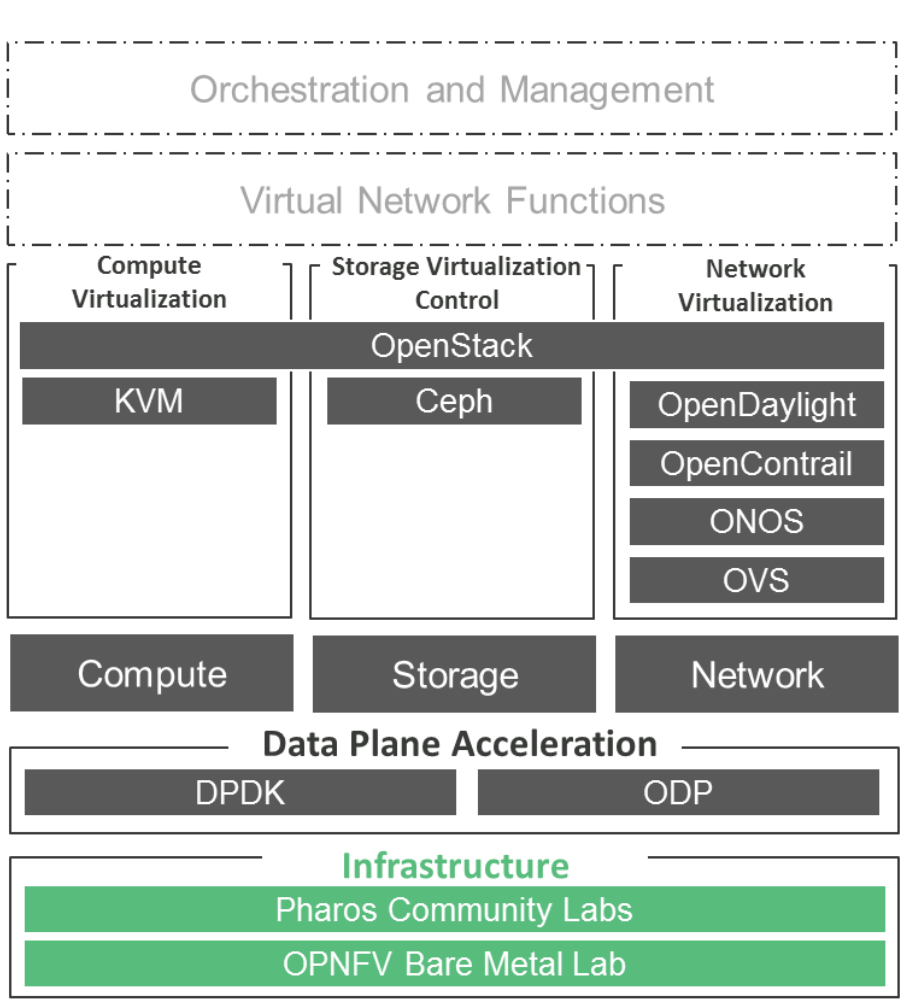
TAKE THE FIRST STEPS

- What we need
  - NFV-ready cloud platform
  - Standardized APIs that fulfills NFV use cases
  - Flexible solution to integrate different SDN controllers into the platform

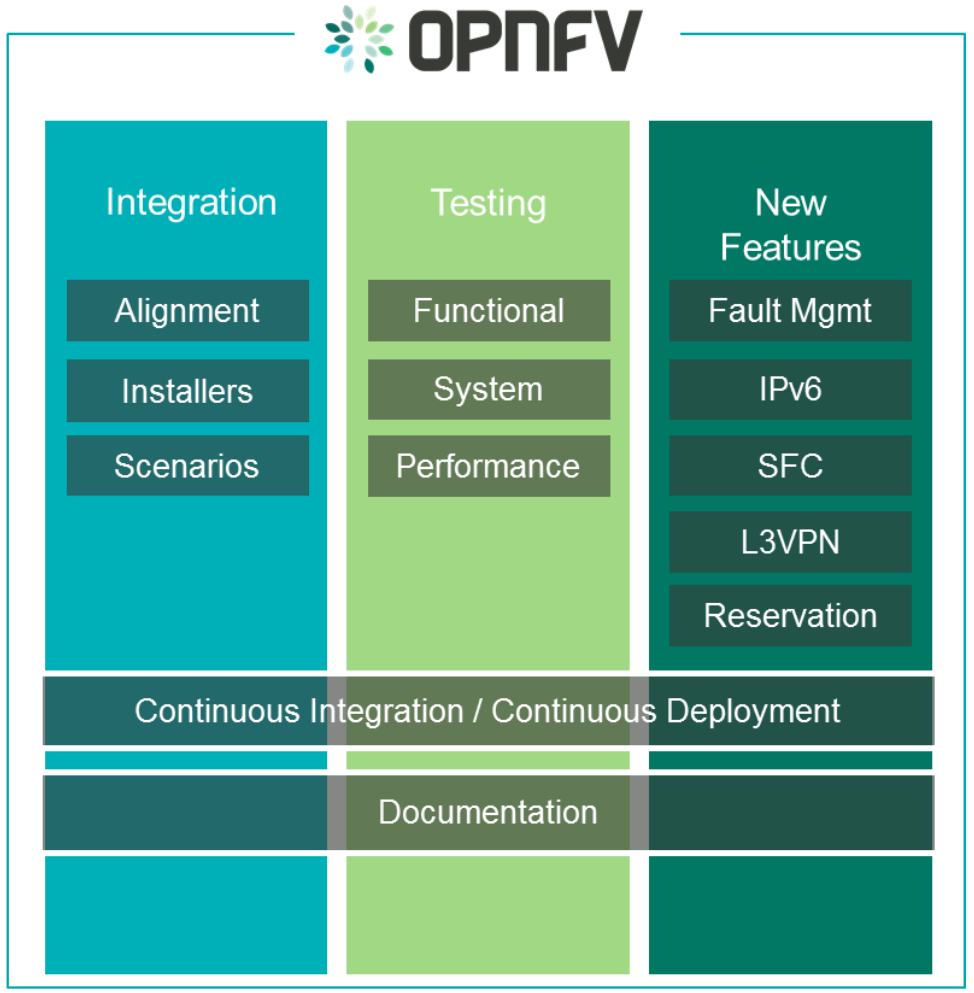
- How we get it
  - Find the gaps
  - Identify alternatives to fulfill the requirements
  - Create prototypes, evaluate the candidates
  - Implement the chosen solution

- Do it as a community effort

OPNFV OVERVIEW



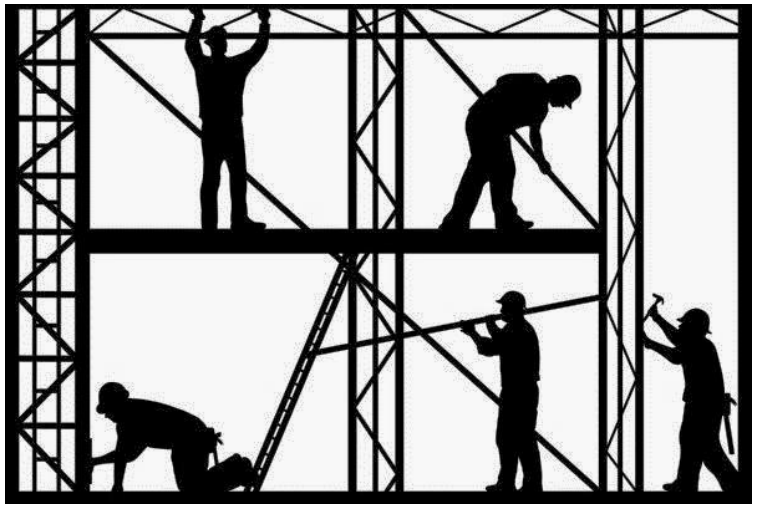
Upstream Project Collaboration:



WHY OPNFV?

- A community that gathers telecom vendors and service providers together
- Gives a clear NFV focus

- Provides a framework
  - Analysis
  - Implementation
  - Integration
  - Testing

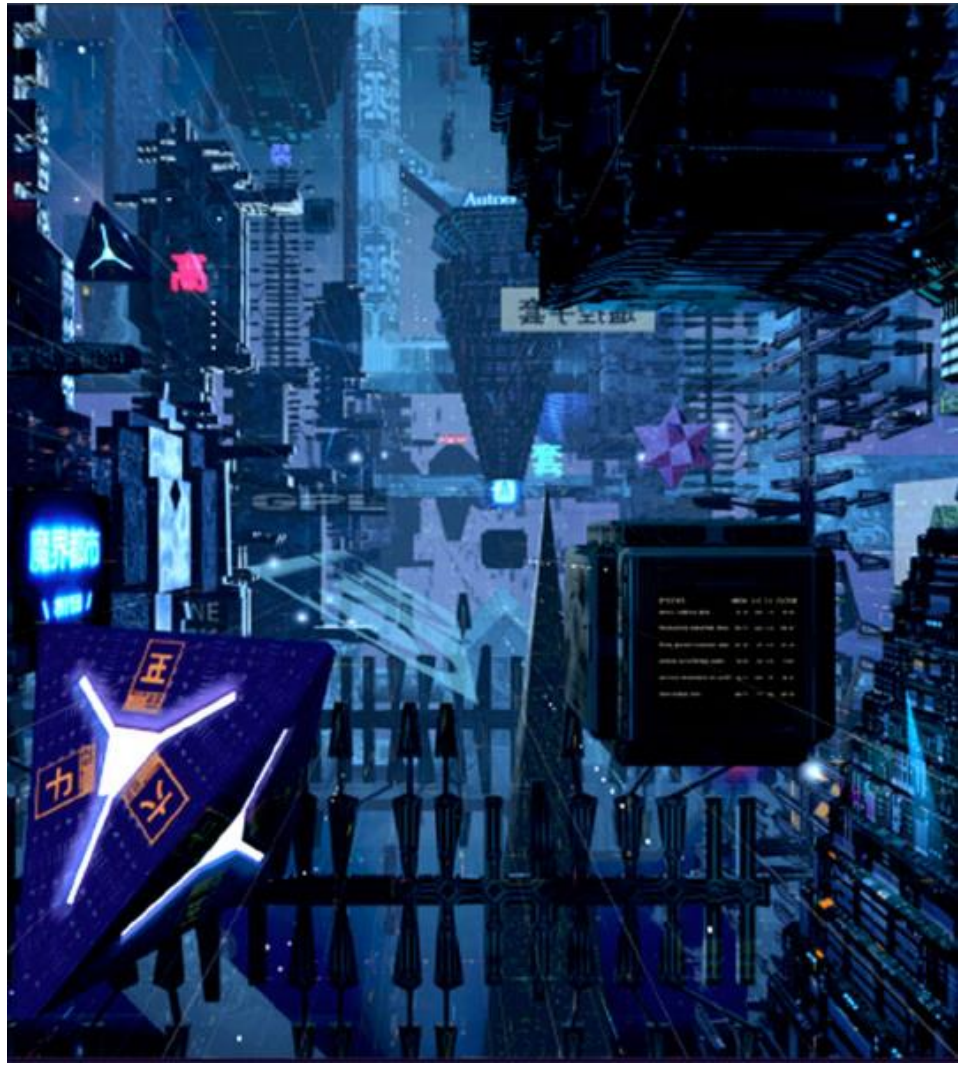


- Works together with upstream communities

- Connected to standardization bodies

NETREADY

- New OPNFV initiative
- Focusing on OpenStack as VIM
  - Find the limitations
  - Create an enhanced solution
- Collaborates with existing networking projects
- Leverages OPNFV test frameworks
  - Evaluate the prototypes
  - Use different SDN controllers as back ends



COME AND JOIN US!

https://wiki.opnfv.org/display/netready/NetReady