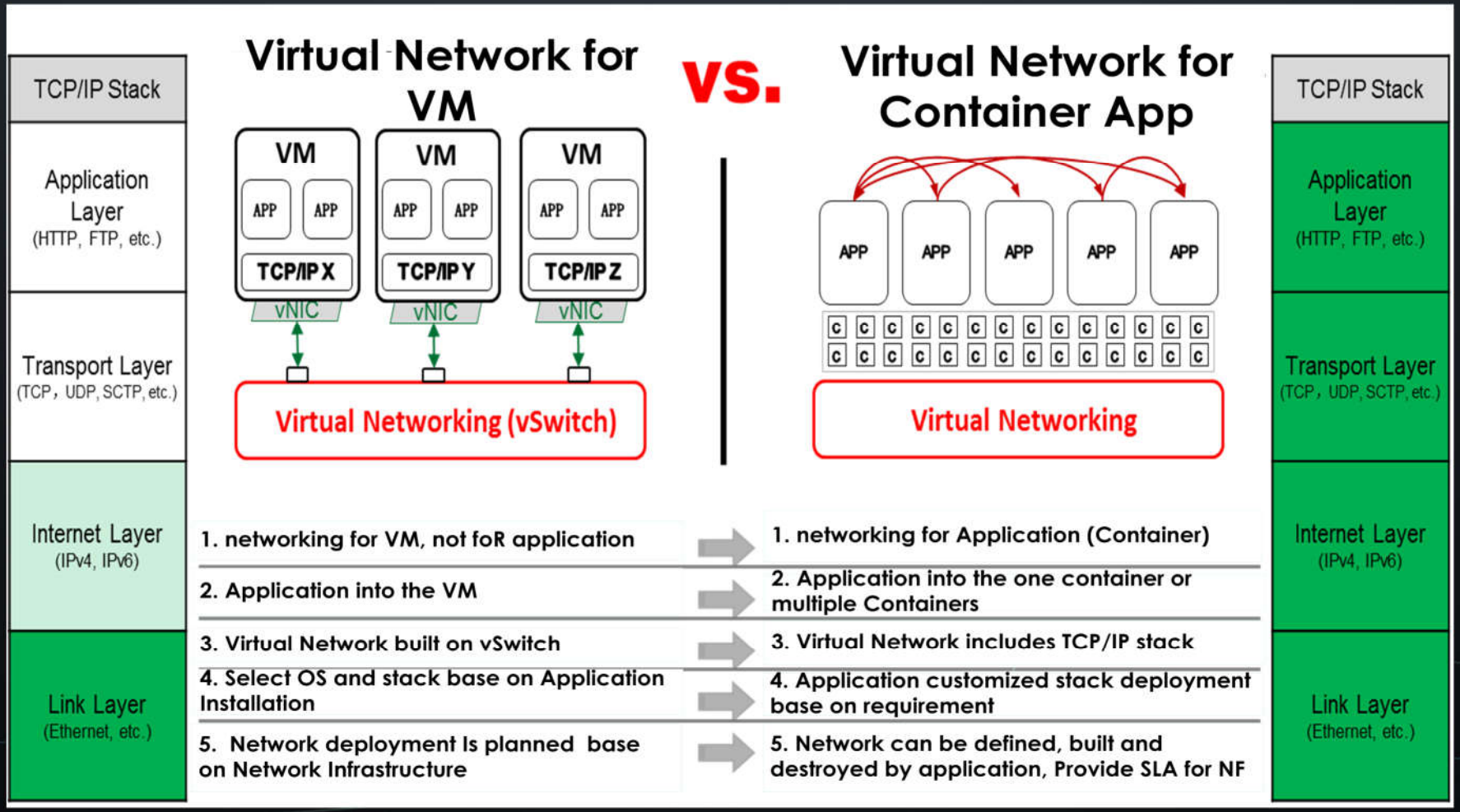
Simplify Networking for Containers

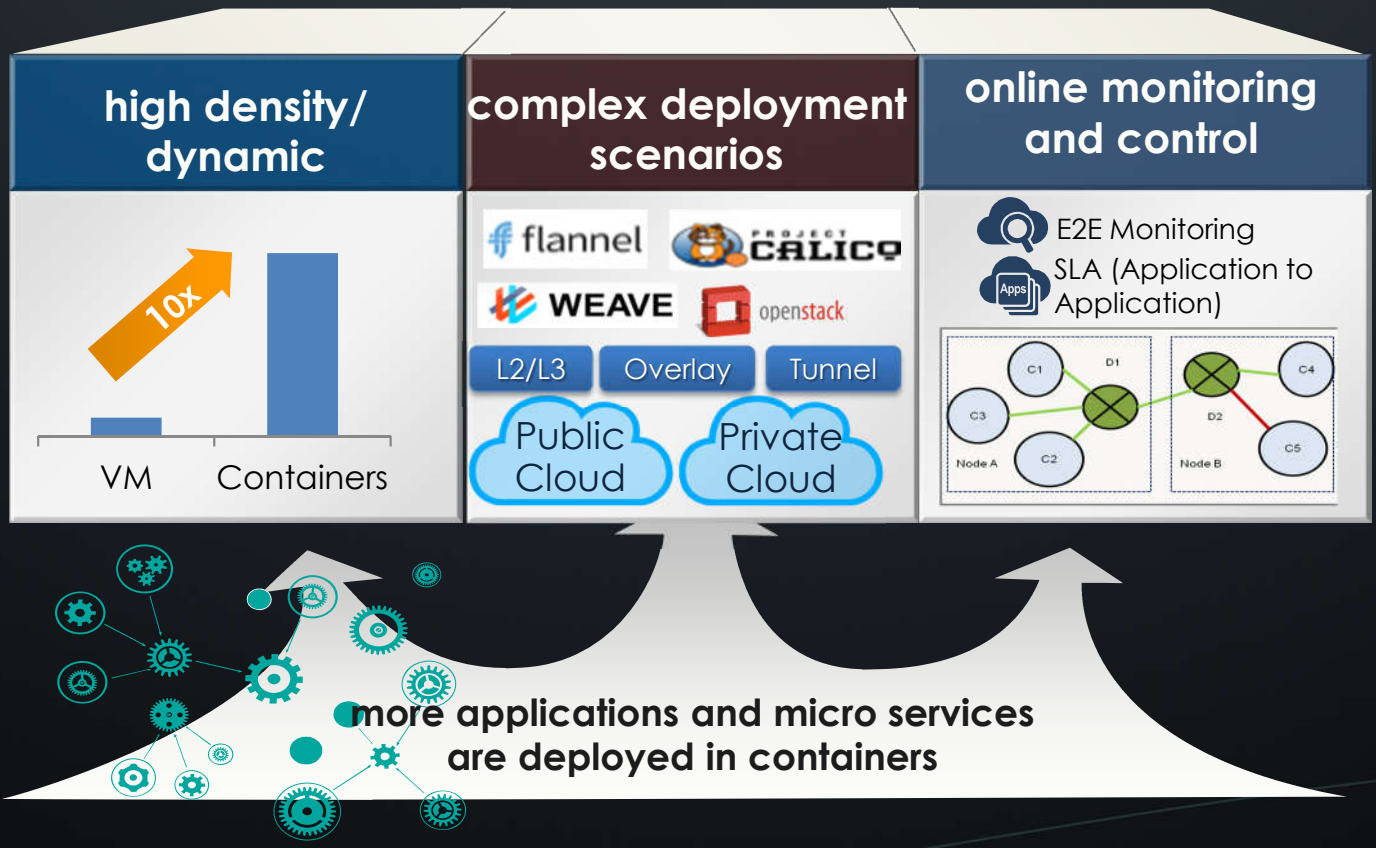
叶磊 曹水

华为 中央软件院 云网络实验室

The Nature of Container Network



cloud native and containerised micro-services



deployment complexity

simple flat container network model: CNI

complex deployment scenarios

public clouds: AWS/Azure/HEC

private clouds: openstack/vmware/ baremetal

NFV: SR-IOV/L2/L3

deployment complexity

imple flat container
etwork model: CNI

existing solutions are suitable for limited cases with hard-coded “plugins”

flannel
PROJECT CALICO
WEAVE
Contiv
openstack.

mplex deployment scenarios

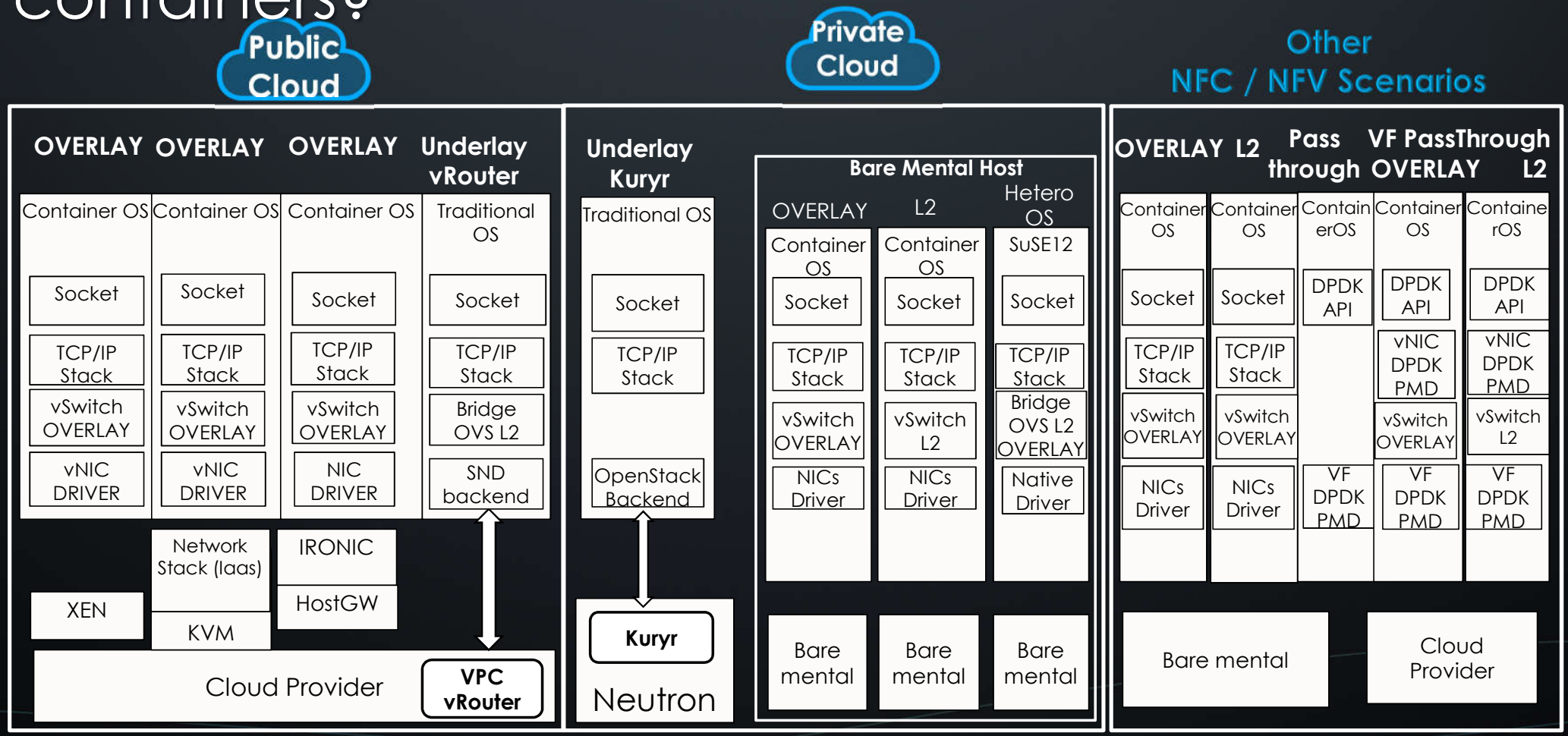
public clouds: AWS/Azure/HEC

private clouds: openstack/vmware/ baremetal

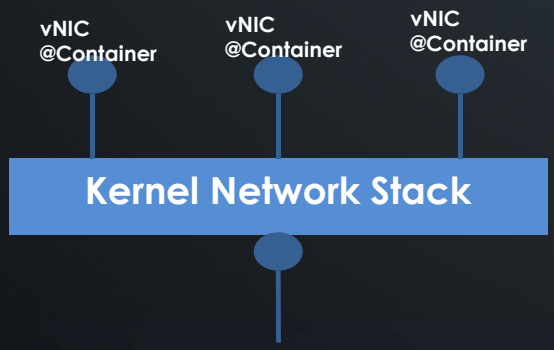
NFV: SR-IOV/L2/L3

require a flexible solution that always adapts the best technology based on specific situation

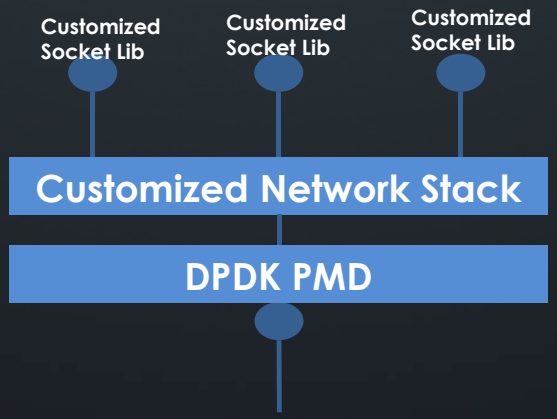
How we deal with so many scenarios for containers?



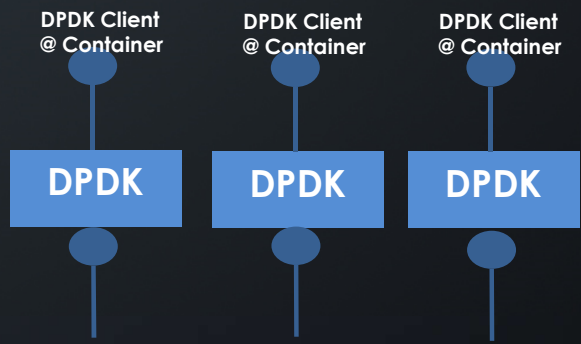
Why we need so many models



Function feature	Rich, identical to Kernel
Performance	Normal
Compatibility	Very good



Function feature	Normal, according to Customized Stack
Performance	Good, about 3 times than Kenel
Compatibility	Normal, maybe miss some socket function

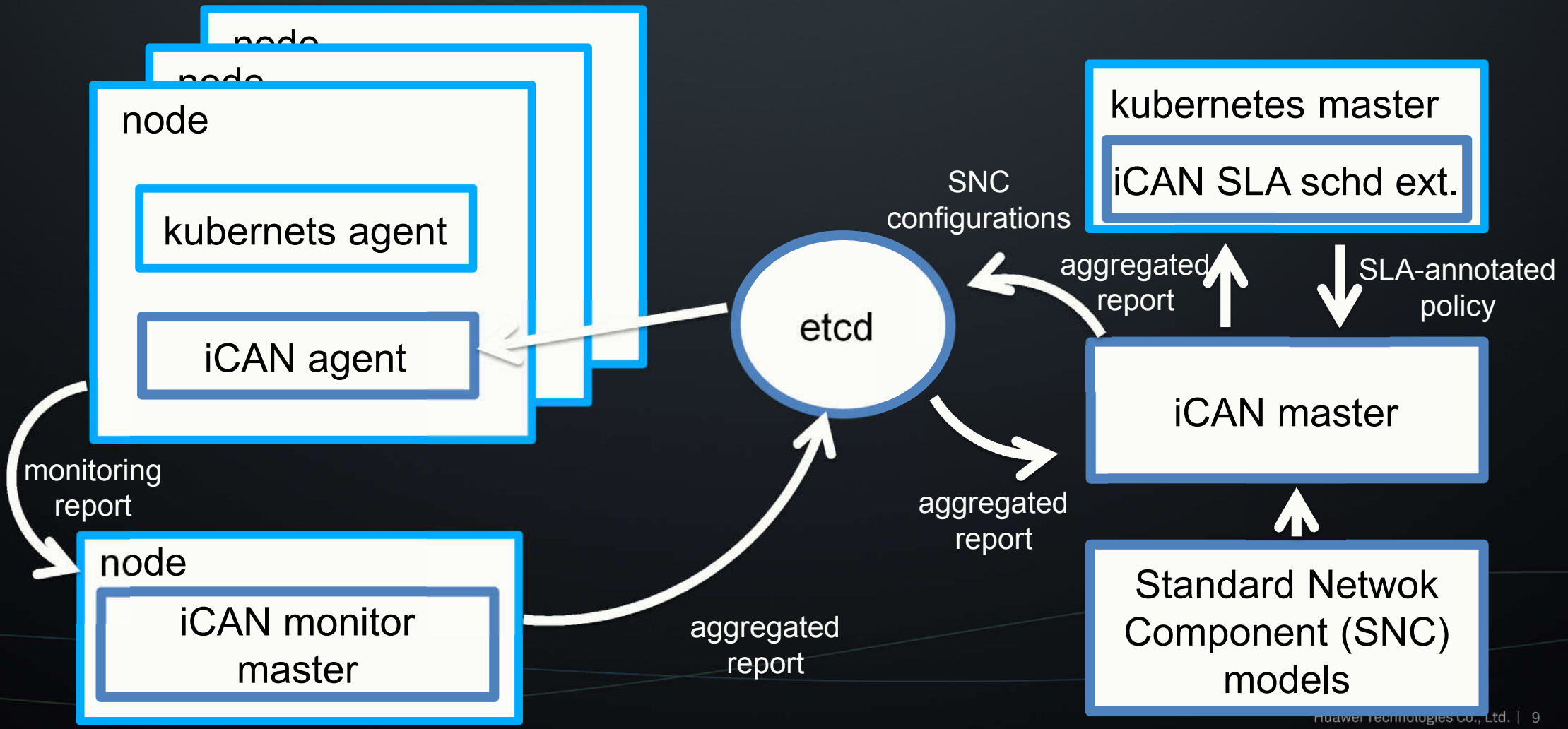


Function feature	Poor, according to DPDK application
Performance	Very good, identical to wire speed
Compatibility	Poor, only DPDK ENV

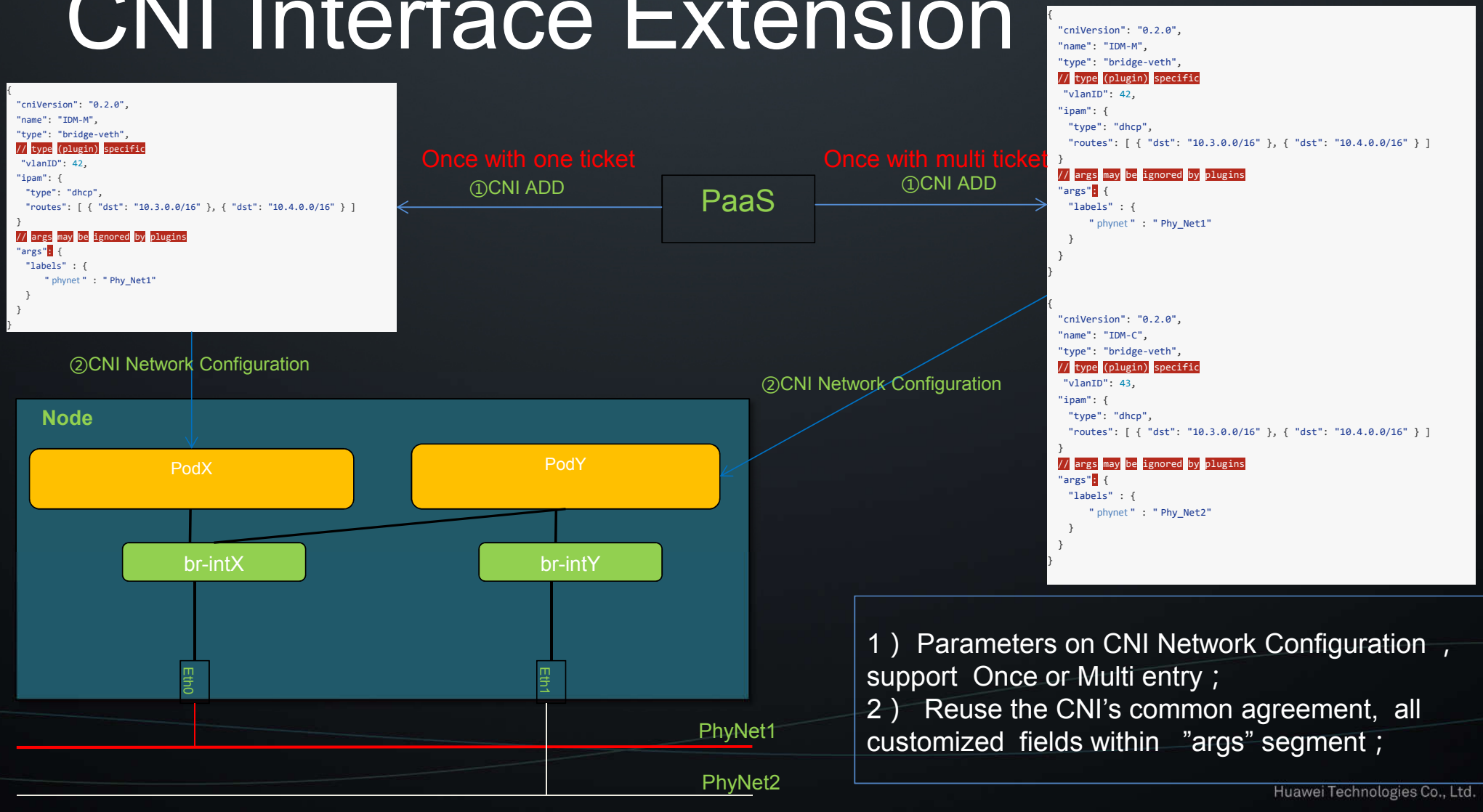
Our solution: iCAN (intelligent Container Network)
an extensible framework to

- program various container network data path and policies
- adapt to different orchestrators
- support end-to-end SLA between containerised applications

iCAN architecture



CNI Interface Extension

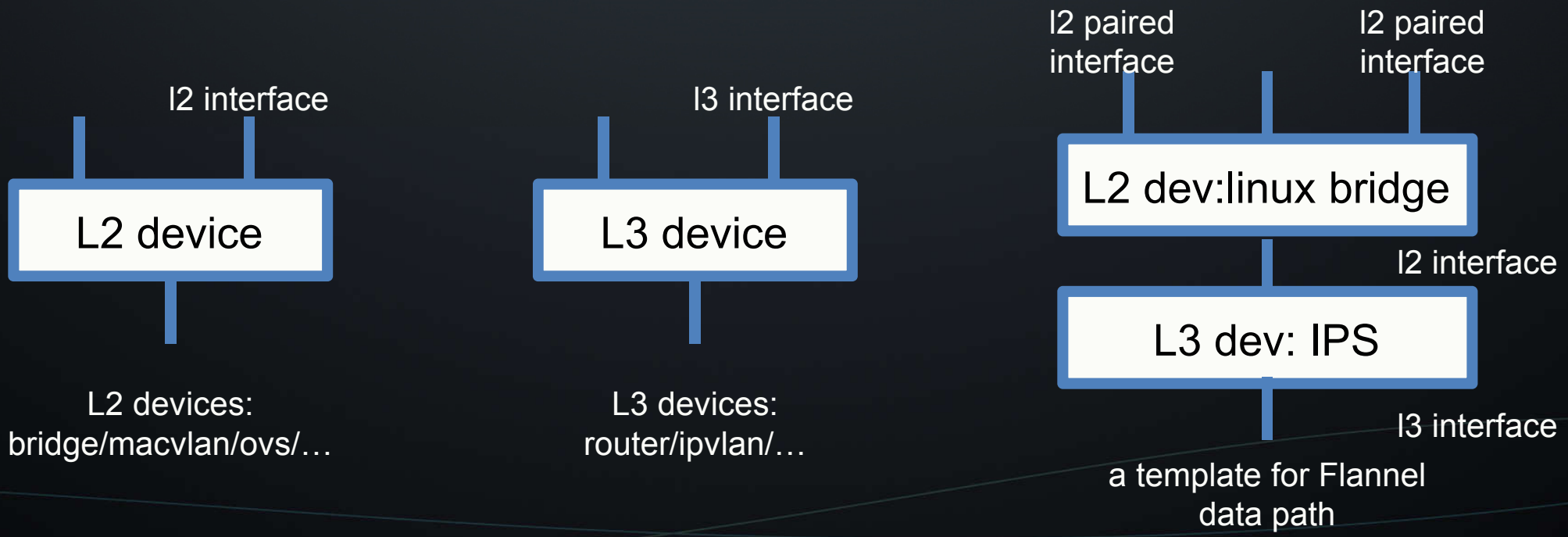


1) Parameters on CNI Network Configuration , support Once or Multi entry ;
2) Reuse the CNI's common agreement, all customized fields within "args" segment ;

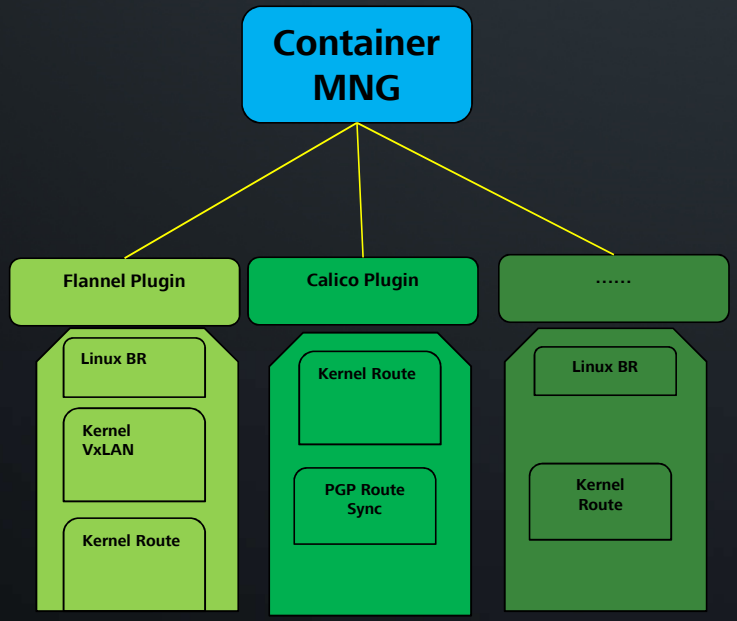
Standard Network Component (SNC) model

abstract for network components in data-path

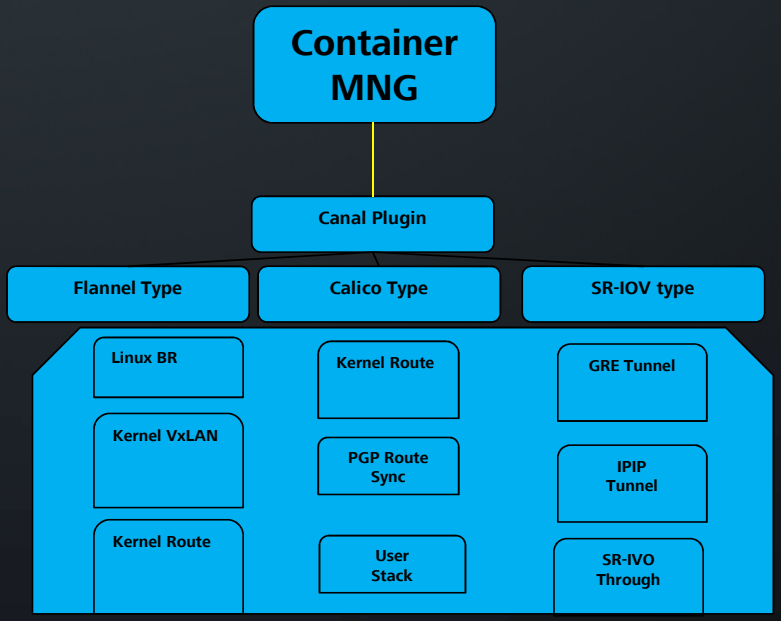
- interfaces, devices and templates



Unified Framework For Multi Models

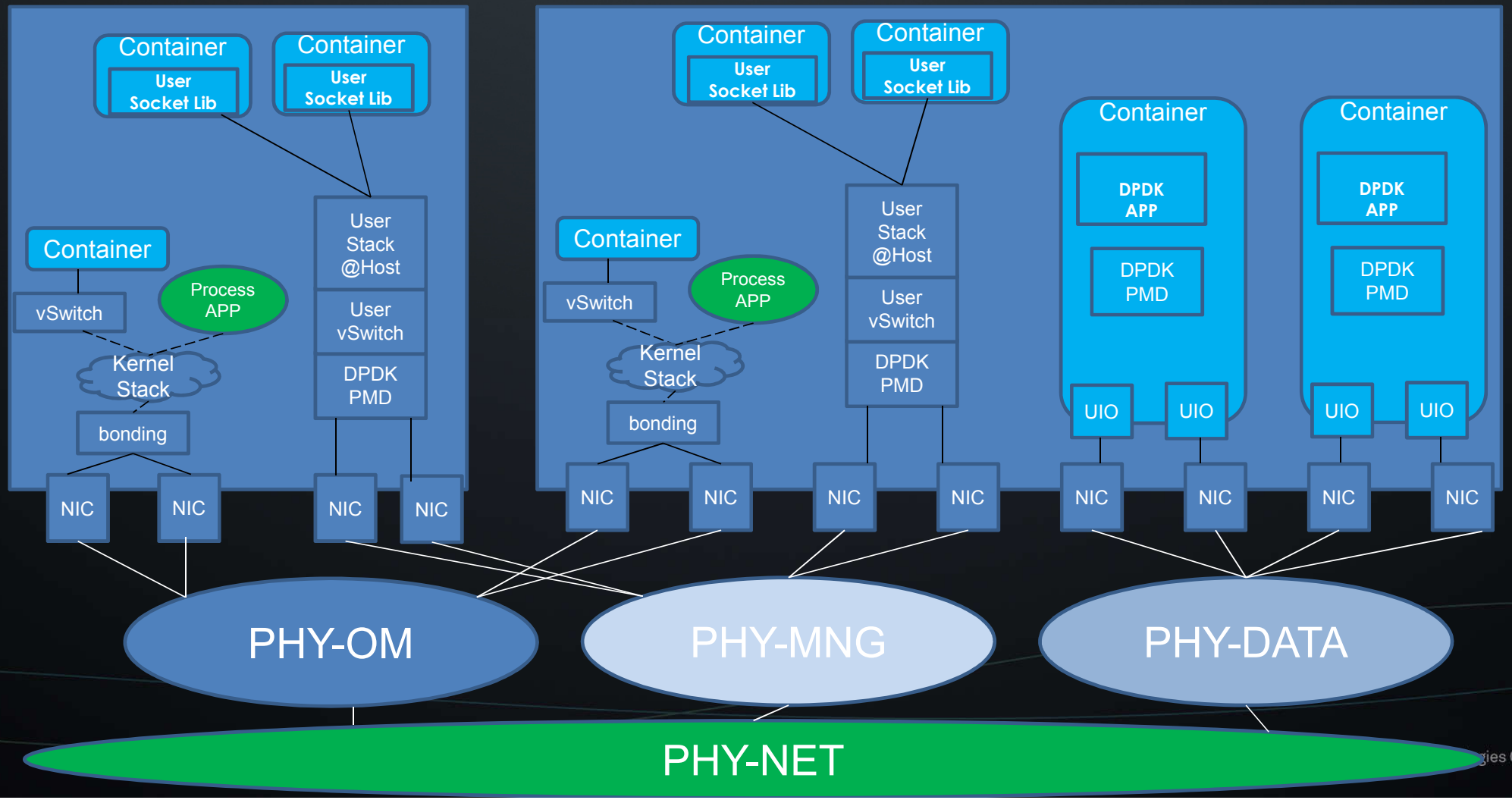


- Existing every Plugins only support its own model
- Though they employ common data module, the function is isolated

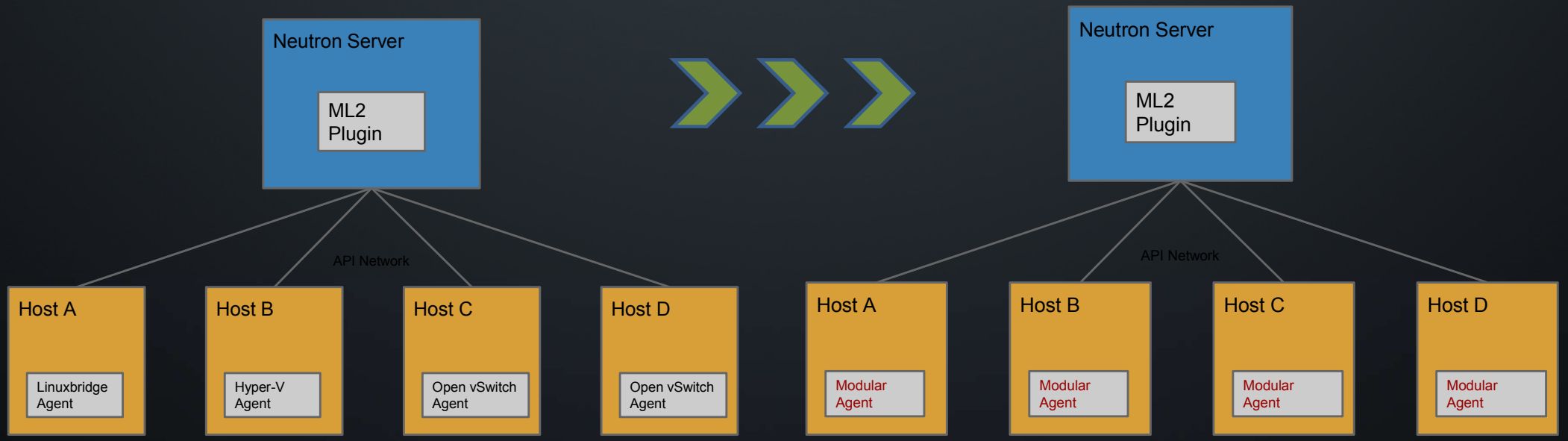


- After deconstruct different data path , we setup a DSL language to describe them ,using abstracted standard component
- Unified Framework with Pluggable drivers for additional vSwitches, Linux BR, SR-IOV, ...

Big Pic of Multi-modes && Multi-planes



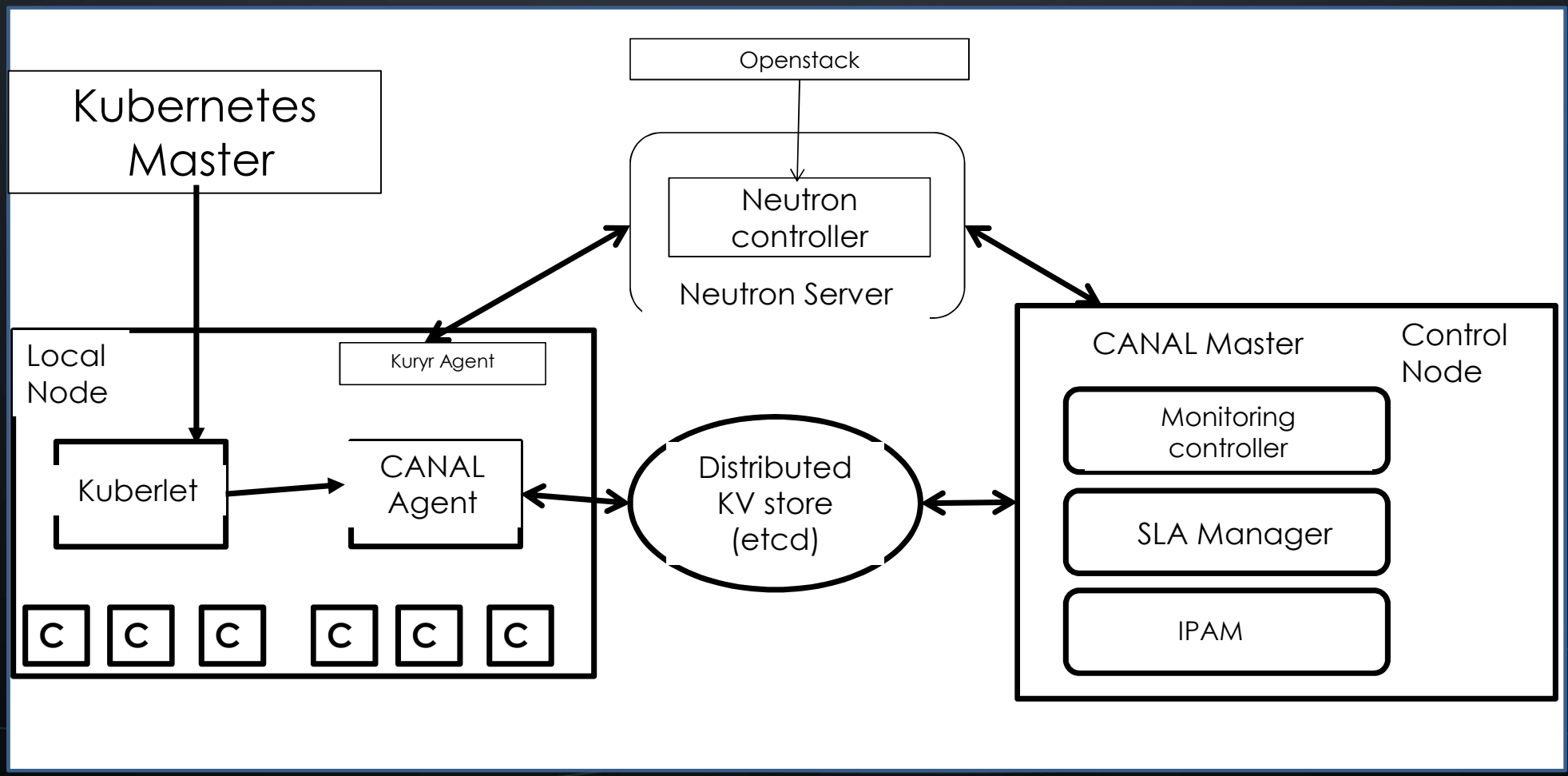
Open stack Neutron Ml2 Solution



- Existing ML2 Plugin works with existing agents
- Separate agents for Linuxbridge, Open vSwitch, and Hyper-V

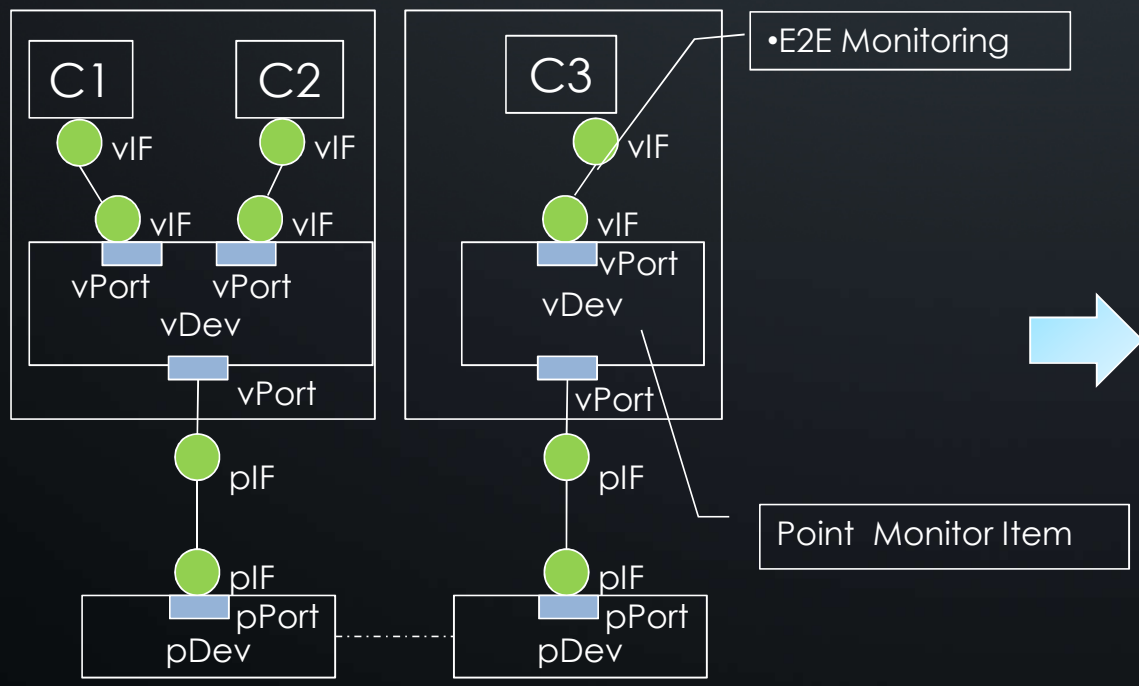
- Combine Open Source Agents, a single agent which can support Linuxbridge and Open vSwitch
- Pluggable drivers for additional vSwitches, Infiniband, SR-IOV, ...

iCAN Control Plane Integrated with Openstack

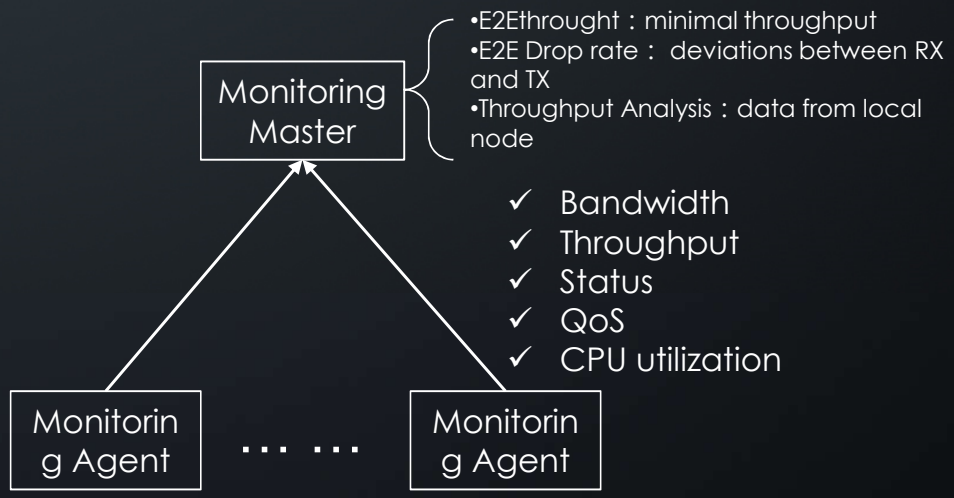


Monitoring based SNC Modeling

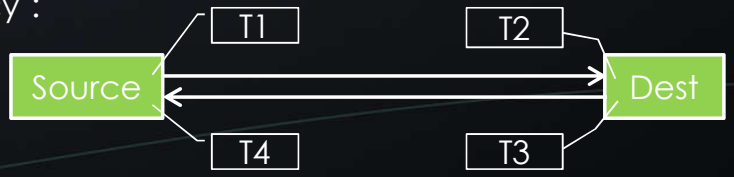
Monitoring on local SNC components ：



Generate E2E monitoring data in master node：



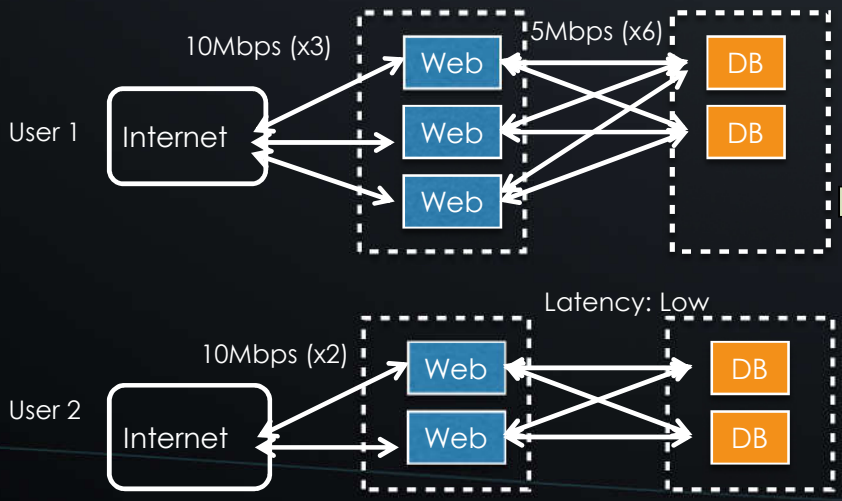
Latency：

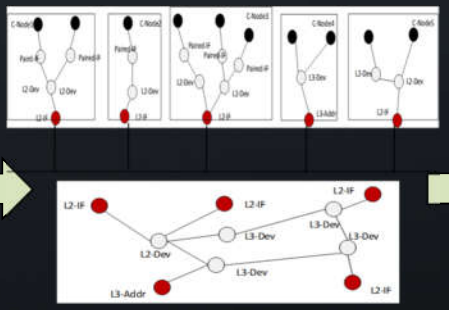


Latency = ((T4 - T1) - (T3 - T2)) / 2

Simplify Network SLA modeling

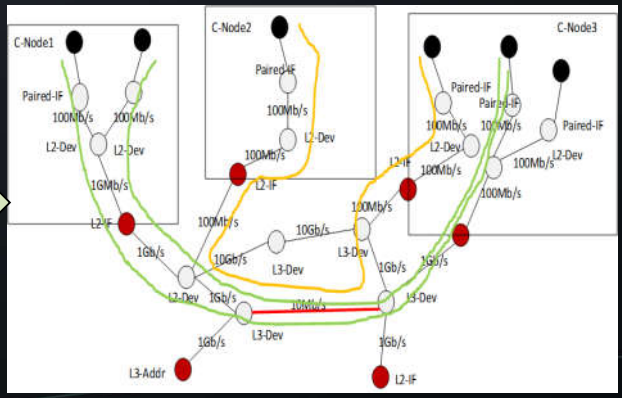
- iCAN provides north bound interfaces for orchestration and applications to define their requirements through PG(Pod Group: a group of pods with the same functions), Linking (network requirement between PG) , SLA Service types and Service LB Type.
- Given topology and link bandwidth, evaluate the offers when deploying pods. Essentially a evaluation for pod placement, and validate the deployment.
- 2-Tiers Network topology management Underlay Network (Stable and Predictable) and Overlay Network (Customizable and Dynamic)
- Support: bandwidth, latency and drop rate
  - Bandwidth <5%
  - Latency <10%, more non-deterministic, affected by many factors such as queuing in software switch and hardware, application response, server IO, etc





Convert link requirement to node requirement

Polices Deployment Scheduler validation



iCAN Container networking

Powerful Monitoring

✓Implement “monitoring on-demand ”and “E-to-E monitoring” based on the topology
✓Facilitate on-demand DSL based troubleshooting
✓Cooperate with the SLA subsystem to assess the SLA quality

Rich Network Support

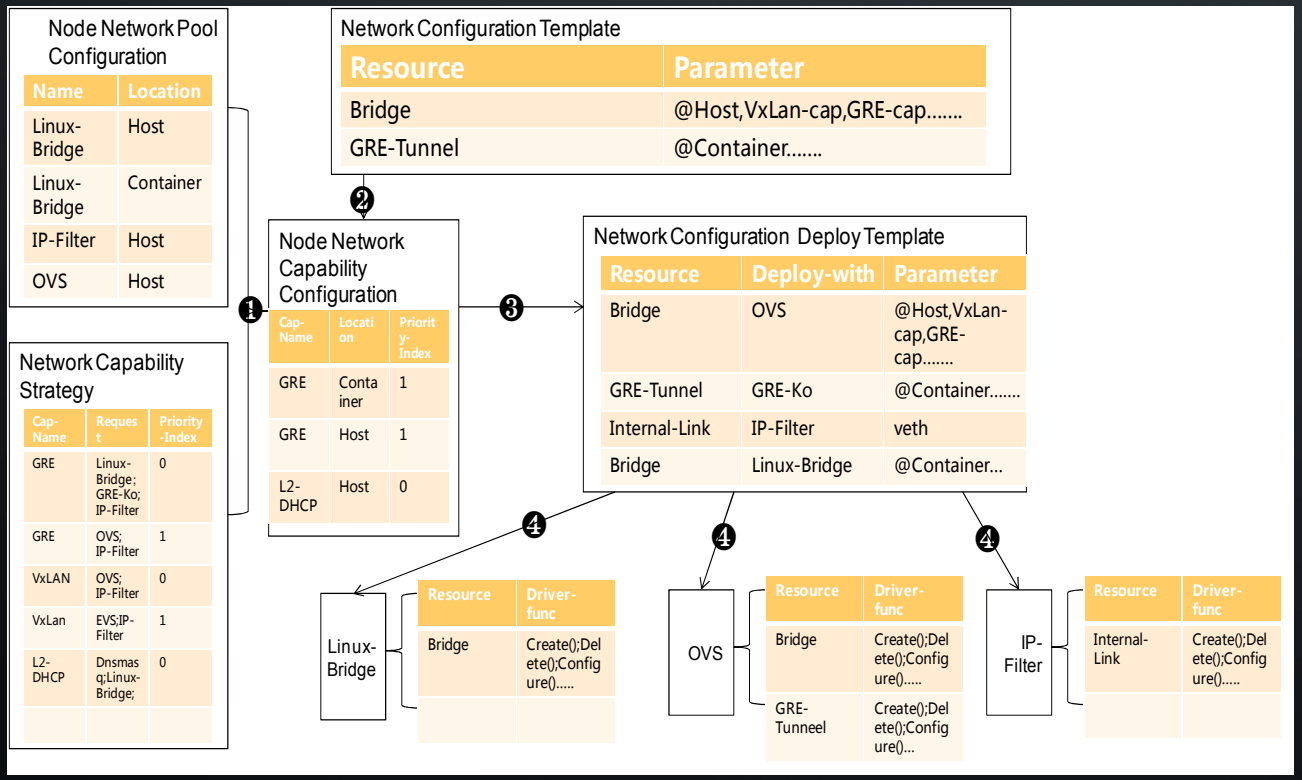
✓Powerful network component modeling : SNC and Modeling via Yang
✓Rich network schemes, support L2, Overlay, NAT, VLAN, L3, BGP, VPC
✓Accelerated Network Stack

Multi-dimension SLA& Security

✓Performance Isolation with bandwidth, latency, drop rate(Proactive Network SLA and Reactive Network SLA)
✓Security Isolation: VLAN/VXLAN, ACL

Thank You.

SNC Template Execution Workflow



❶ Network-Agent Local initialized base on Node Network Pool Configuration and Network Capability Strategy， generate Node Network Capability Configuration(NNCC) .

❷ Node received template deployment request, check NNCC. If node can't meet requirement, return failure, otherwise will return Network Configuration Deployment Template (NCDT) with information ❸ ;

❹After, send network deployment request to Network-Element as NCDT defined, Finally executed by related network driver;

Modeling for Standard Network Component

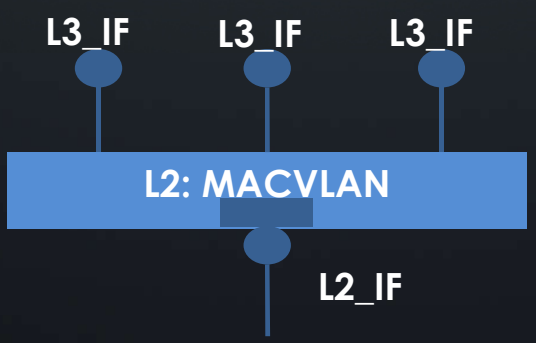
Standard Network Component can help to :

✓Decouple network control with implementation

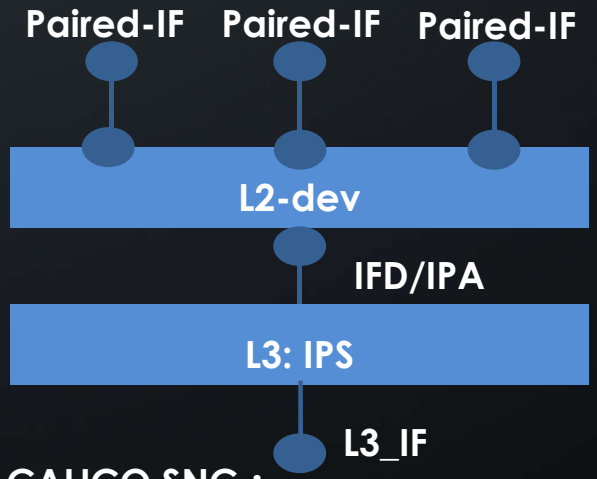
✓Replace and upgrade network components seperately

✓ Provide on-demanding network solution and SLA for application

IPVLAN SNC : L3_IF
L3_DEV
L2_IF



MACVLAN SNC : L3_IF
L2_DEV
L2_IF



CALICO SNC :

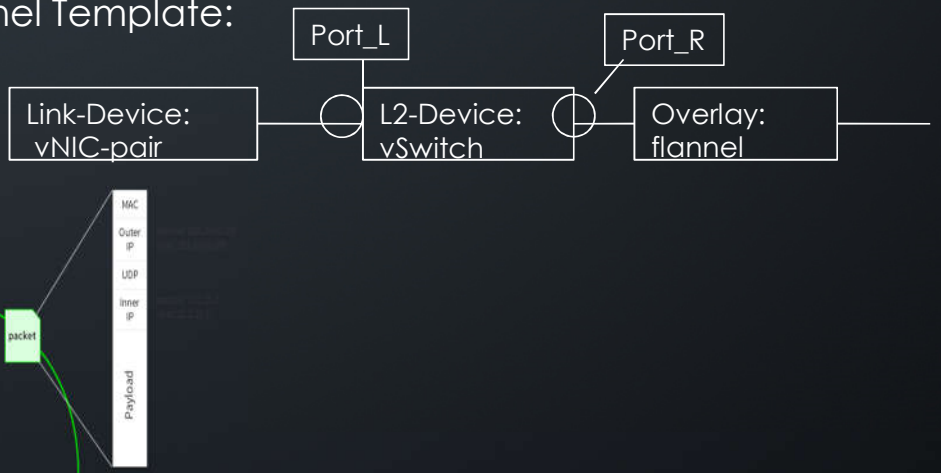
Paired-IF
L2_DEV
L3_DEV
L3_IF

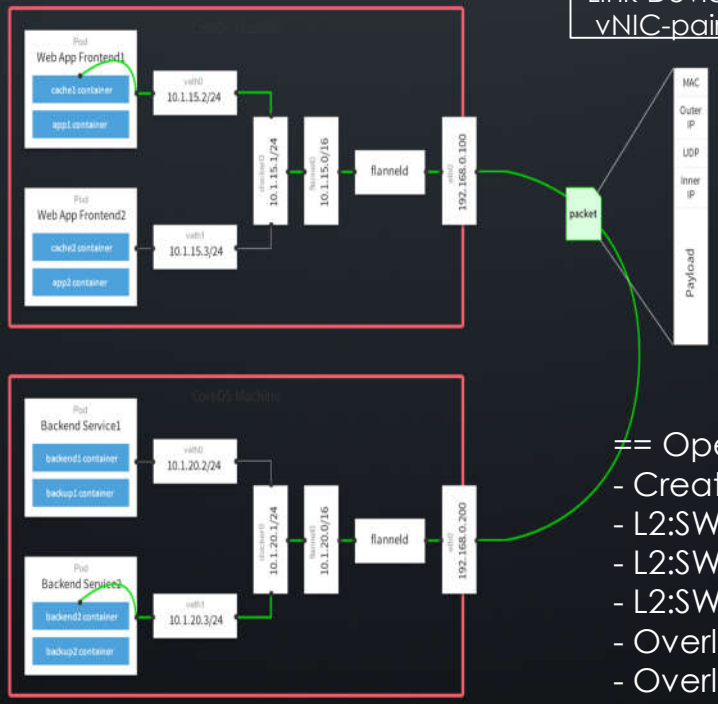
Example: Support with Flannel(VxLAN backend mode)via SNC Modeling (kernel based Overlay)


```
SNC interfaces: /* L2:SW device definition */
{
    /* members */
     string port[];

    /* methods */
     CreateDevice(); // creat L2:SW device
     CreatePort(string port_name);
}
 /* Overlay:Flannel device definition */
{
     /* members */
     string inf_L;
     string inf_R;
      /* methods */
     CreateDevice();
     Connect(string inf, string port);
}
 /* Link:VNIC-pair device definition */
{
     /* members */
     string inf_L;
     string inf_R;
      /* methods */
     CreateDevice();
     Connect(string inf, string port)
}
```

Flannel Template:





== Operating abstraction:
- CreateSubnet() -- get subnet information via etcd API
- L2:SW.CreateDevice() => "l2_sw_dev"
- L2:SW.CreatePort(port_L)
- L2:SW.CreatePort(port_R)
- Overlay:Flannel.CreateDevice() => "flannel_dev"
- Overlay:Flannel.Connect(flannel_dev.inf_L, l2_sw_dev.port_R)
- Overlay:Flannel.Connect(flannel_dev.inf_R, eth0)
- Link:vNIC-pair.CreateDevice() => "link_dev"
- Link:vNIC-pair.Connect(link_dev.inf_R, l2_sw_dev.port_L)