SONiC and Linux

Guohan Lu (Microsoft)
Kalimuthu Velappan (Broadcom)
Kiran Kella (Broadcom)
Marian Pritzak (Nvidia)

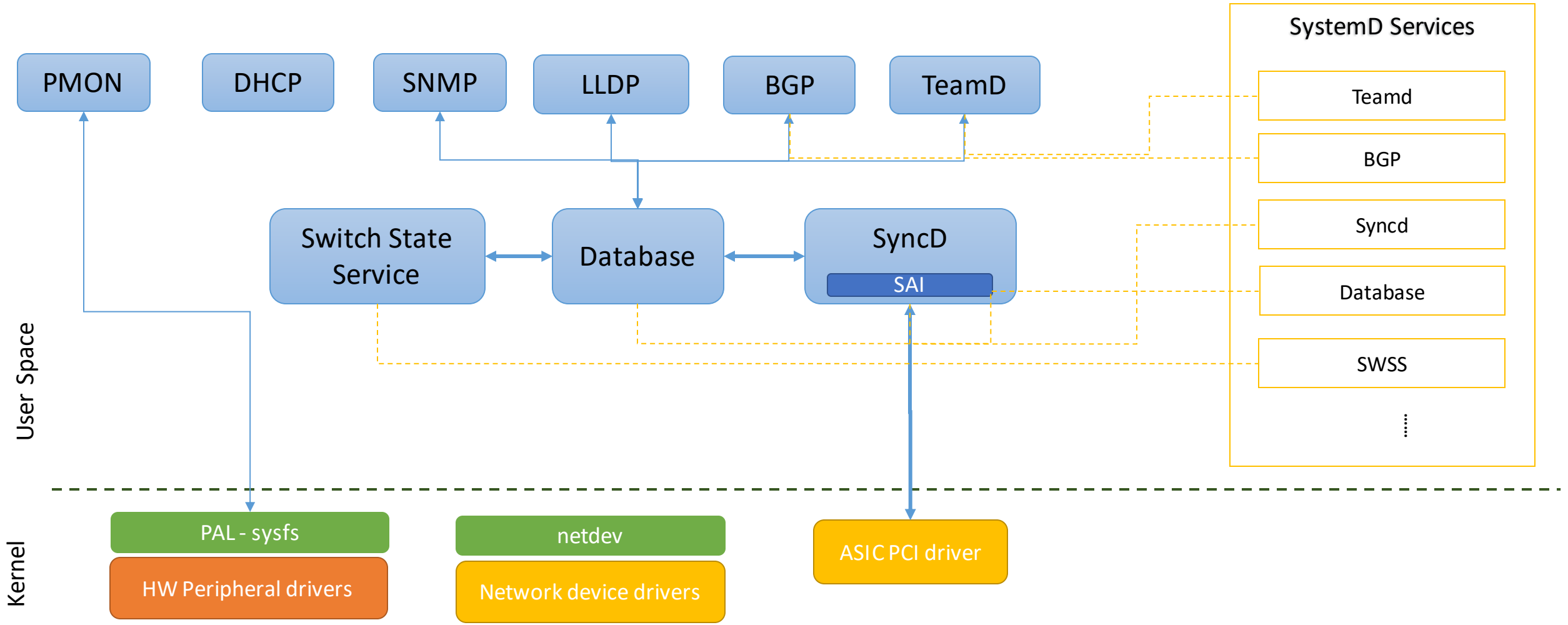
Agenda

- Motivation
- SONiC Architecture
- Beyond Single ASIC
- Sonic features and linux
- System Scaling
- Full cone NAT

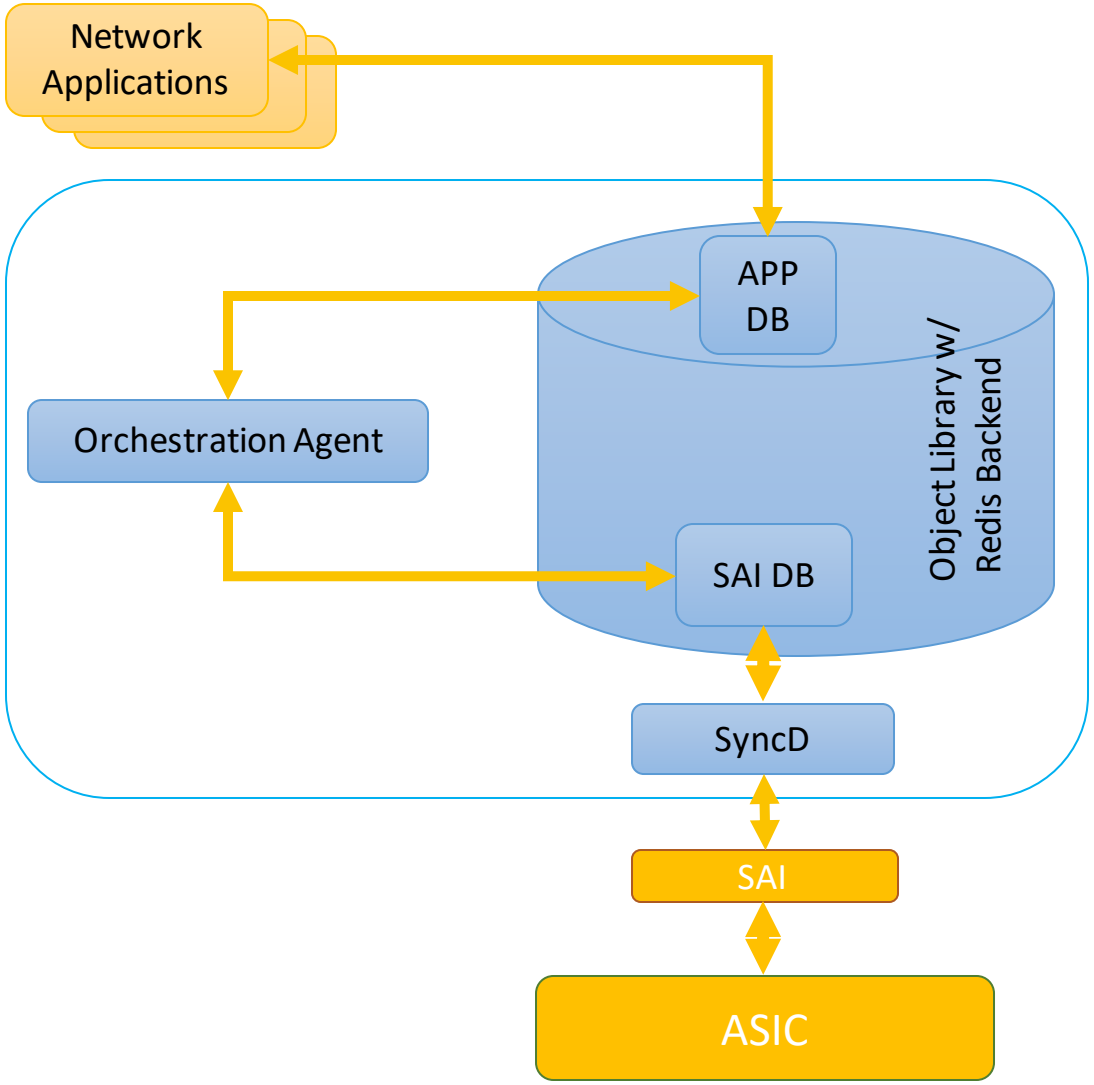
What is SONiC

- Software for Open Networking in the Cloud
- A collection of software components/tools
  - Builds on the foundations of SAI
  - Provides L2/L3 functionalities targeted for the cloud
  - Linux-based switch operating system, looks and feels like Linux
- Community driven, open source effort
  - Shared on GitHub, Apache License
  - Believe in working code + quick iteration

SONiC Container Architecture – Single ASIC



Switch State Service (SSS)



SAI DB: persist SAI objects

APP DB: persist App objects

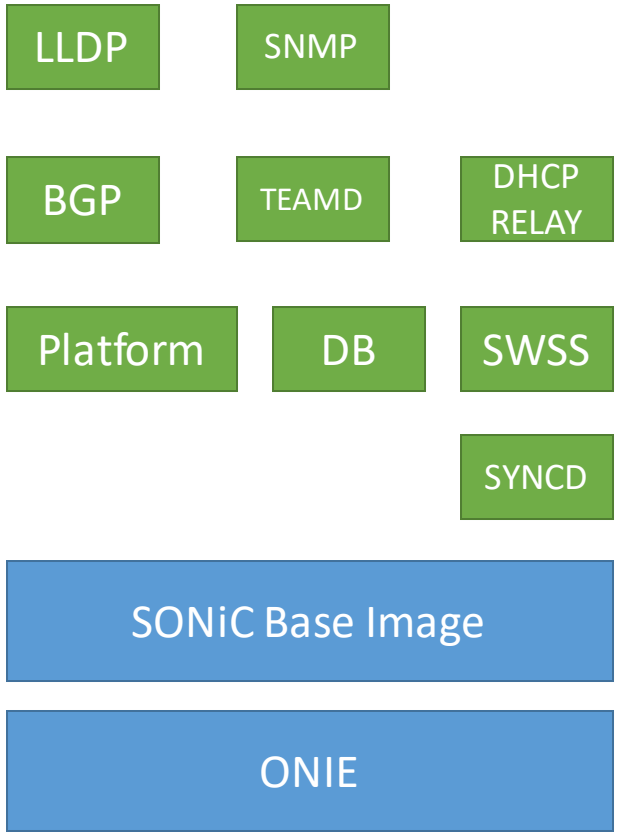
DB backend: redis with object library

SyncD: sync SAI objects between software and hardware

Orchestration Agent: translation between apps and SAI objects, resolution of dependency and conflict

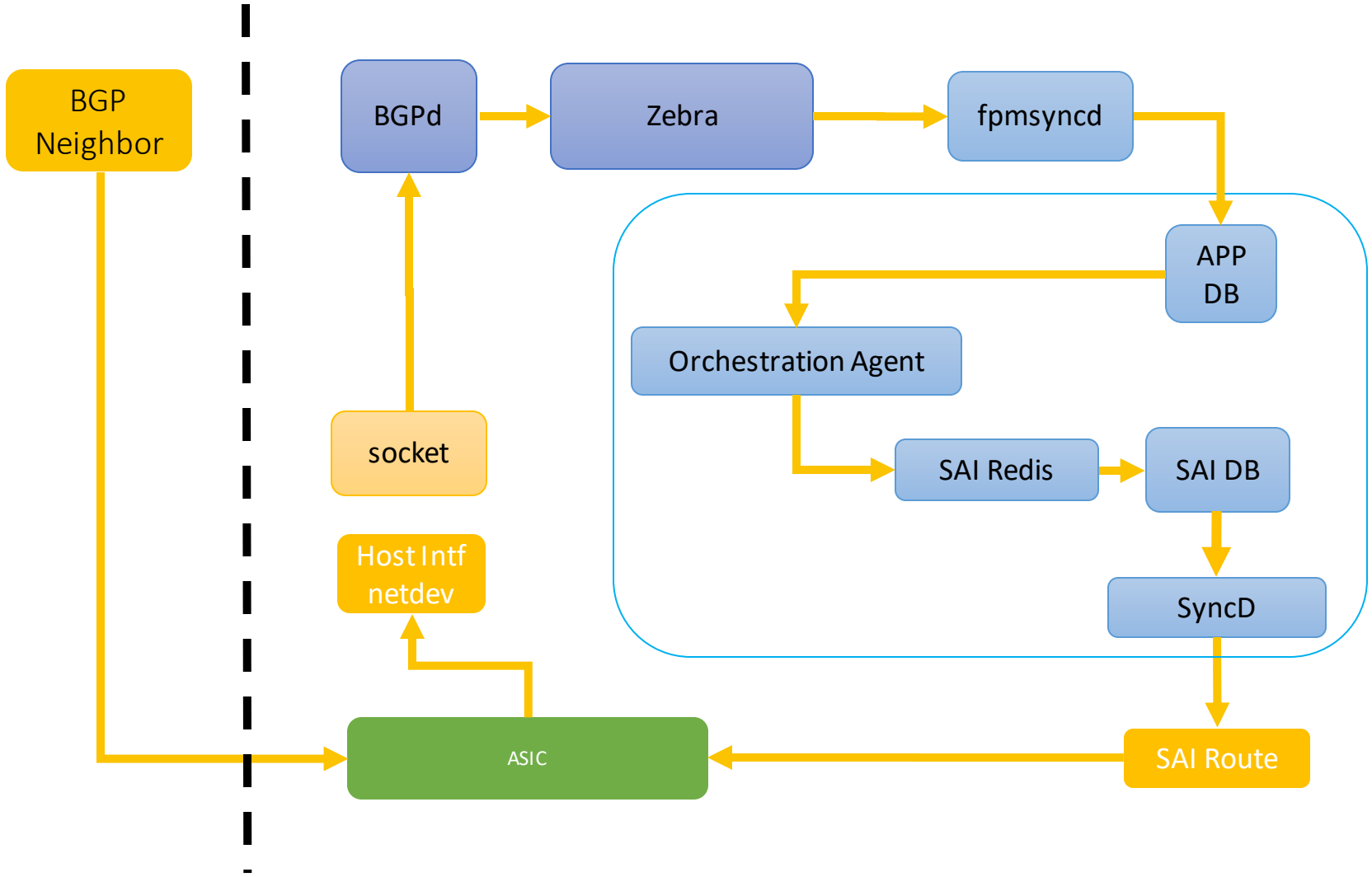
Key Goal: Evolve components independently

SONiC Software Module

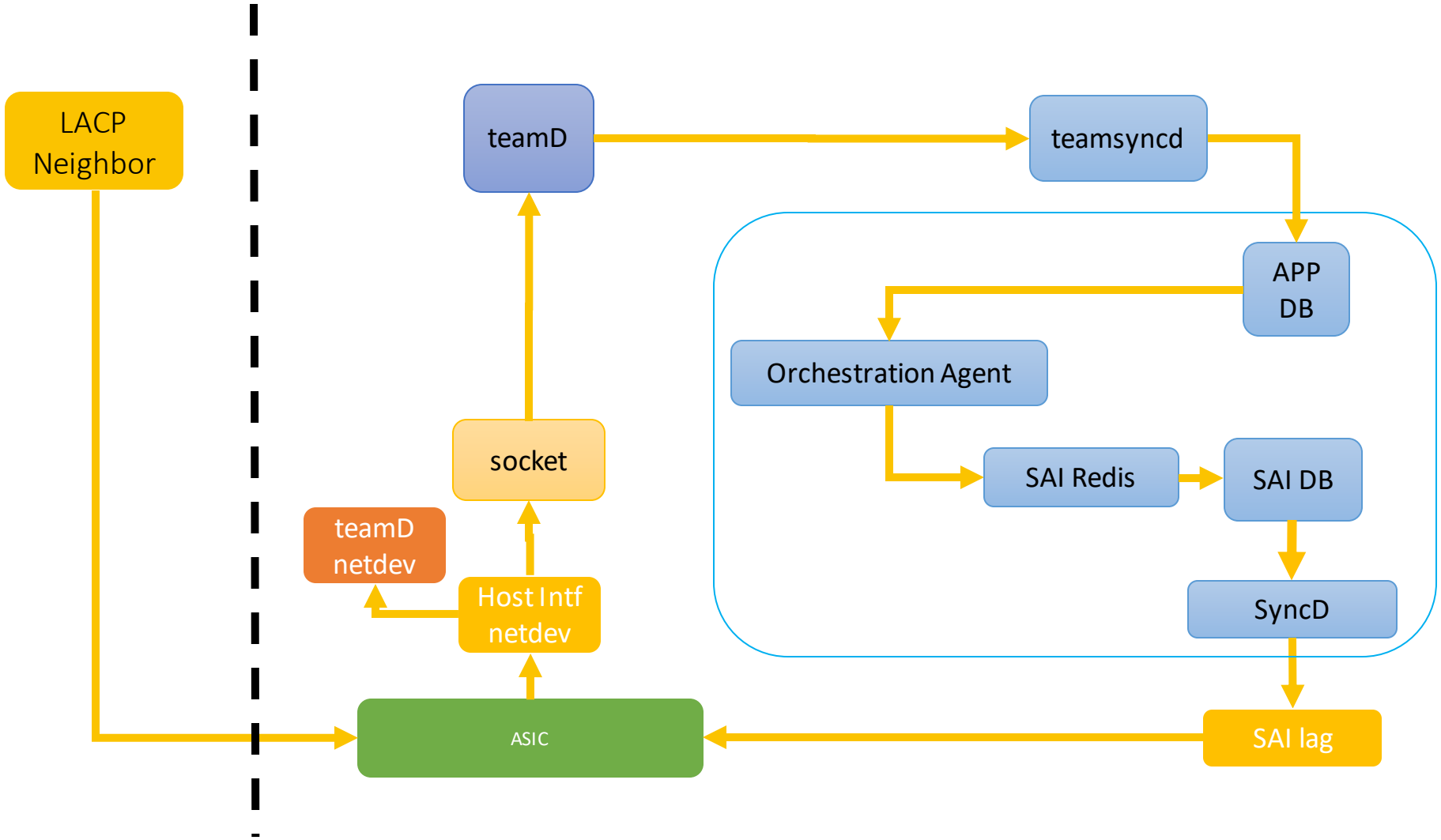


- TEAMD: https://github.com/jpirko/libteam/tree/master/teamd
- LLDP: https://github.com/vincentbernat/lldpd
- BGP: Quagga
- SNMP: Net-SNMP + SNMP subagent
- DHCP Relay: isc dhcp
- Platform: sensors
- DB: redis
- SWSS: switch state service
- Syncd: sairedis + syncd agent

How Routing Works in SONiC

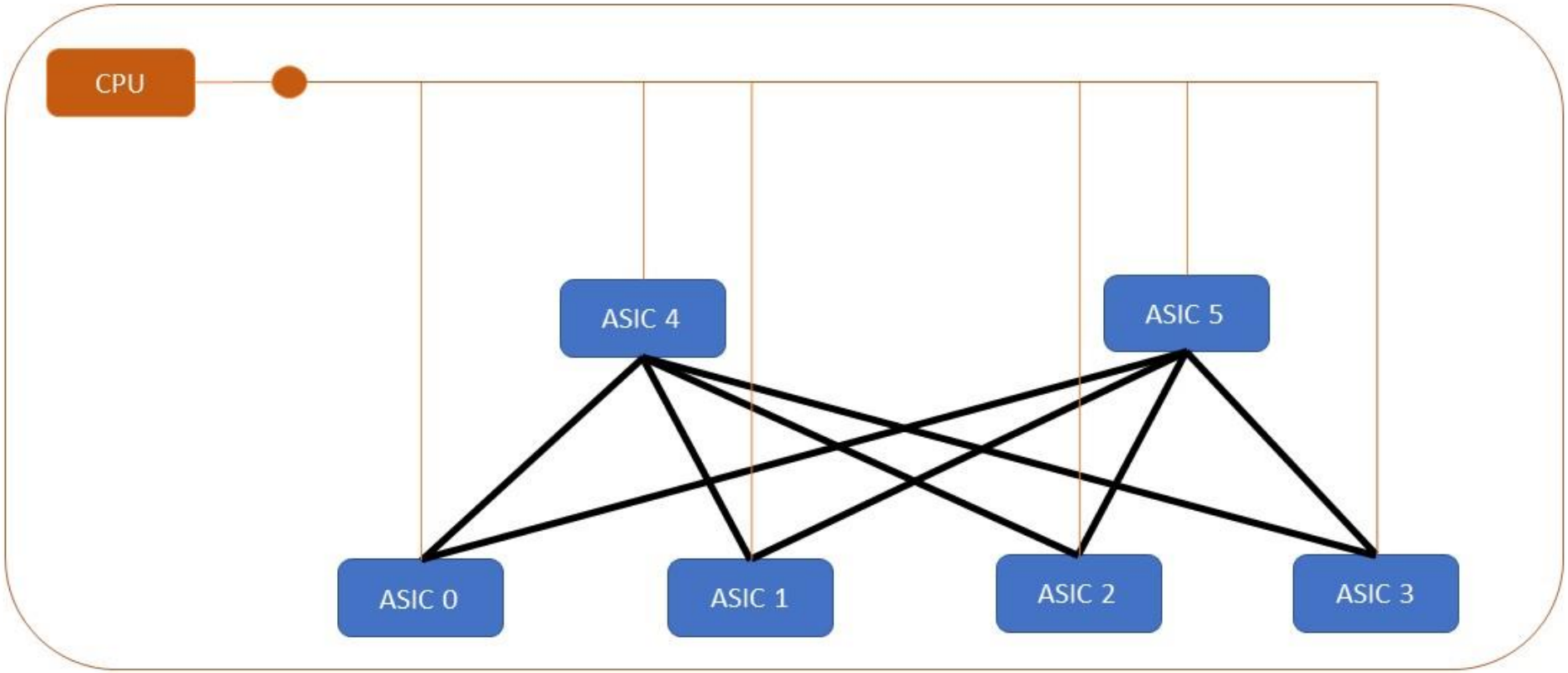


How LAG Works in SONiC



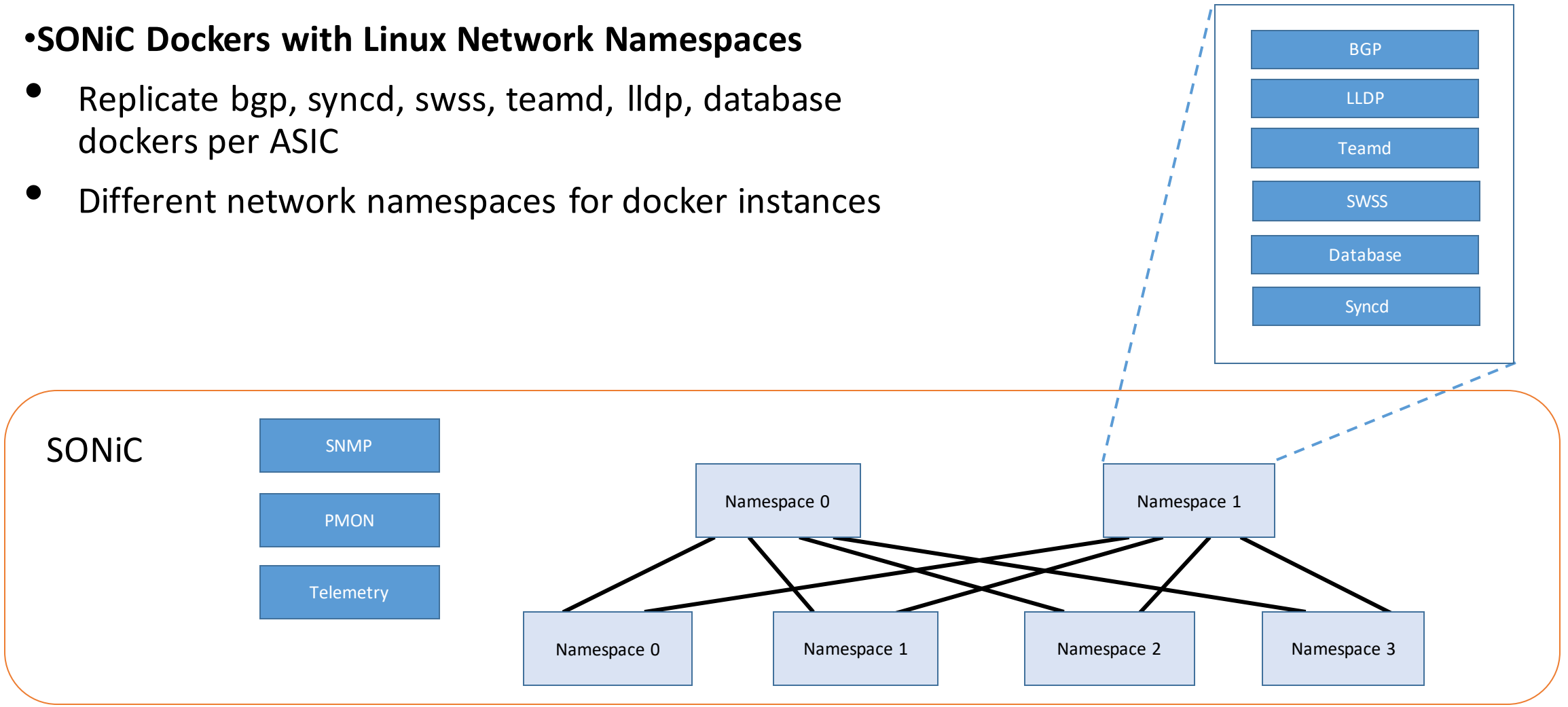
Beyond Single ASIC

A Multi ASIC Device

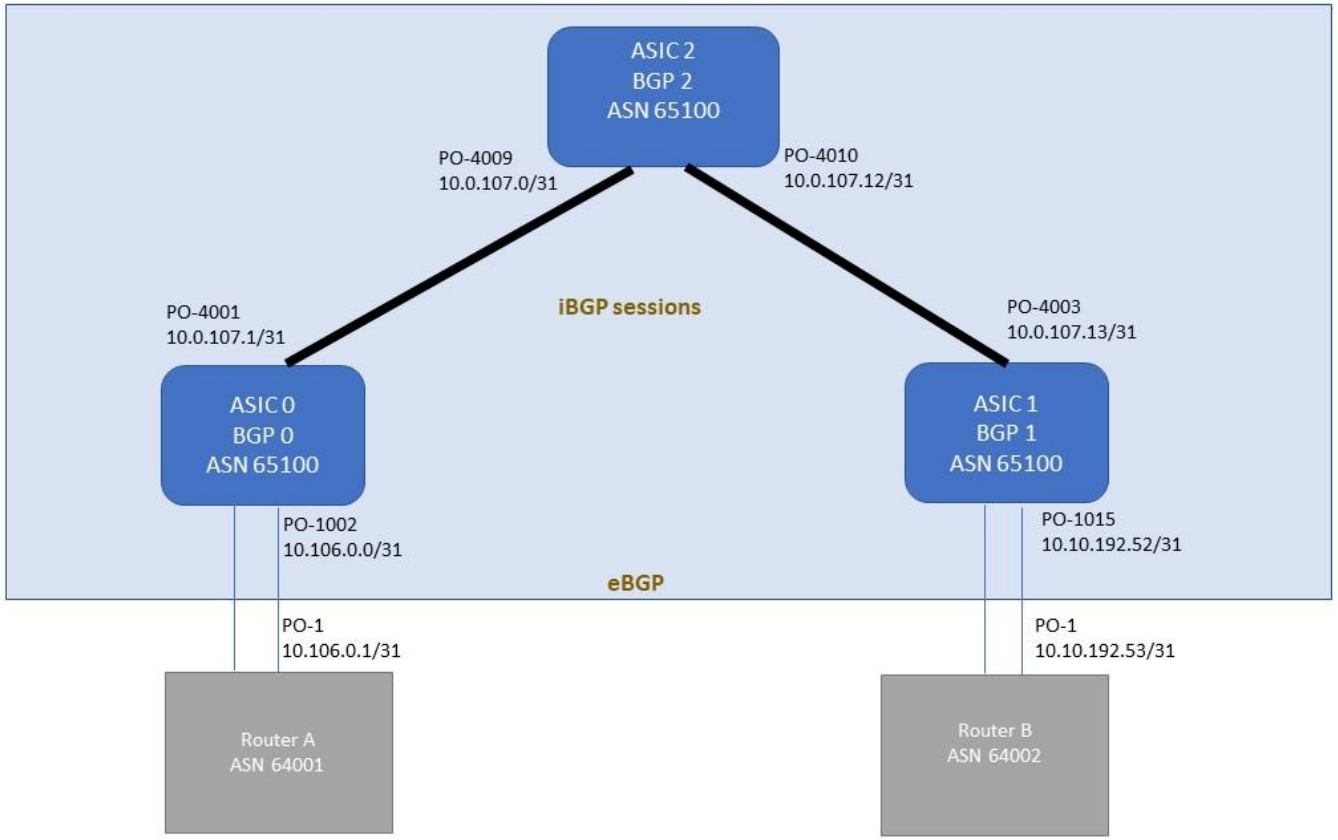


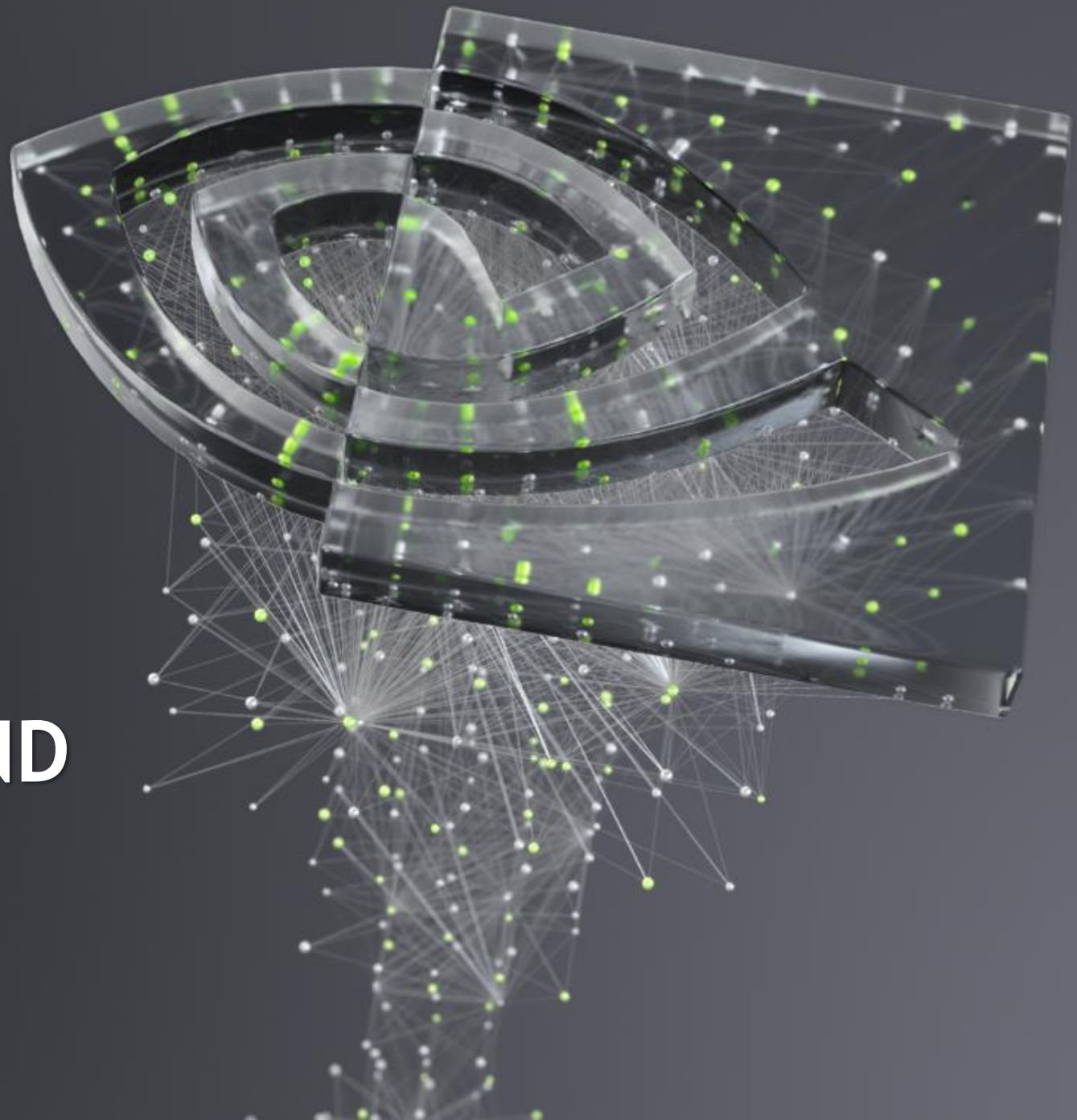
Linux Network Namespaces For Multiple ASICs

- **SONiC Dockers with Linux Network Namespaces**
- Replicate bgp, syncd, swss, teamd, lldp, database dockers per ASIC
- Different network namespaces for docker instances



How routing works?





SONIC FEATURES AND LINUX

Marian Pritsak, August 2020

Features

VNET

Sflow

ACL

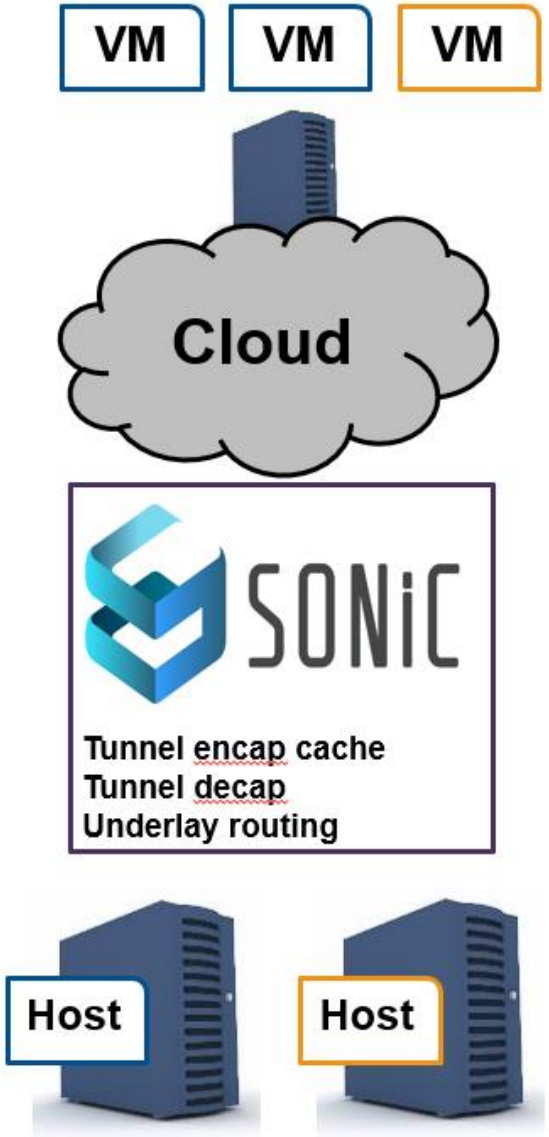
NAT

Switch Memory Management

SONiC Virtual Switch

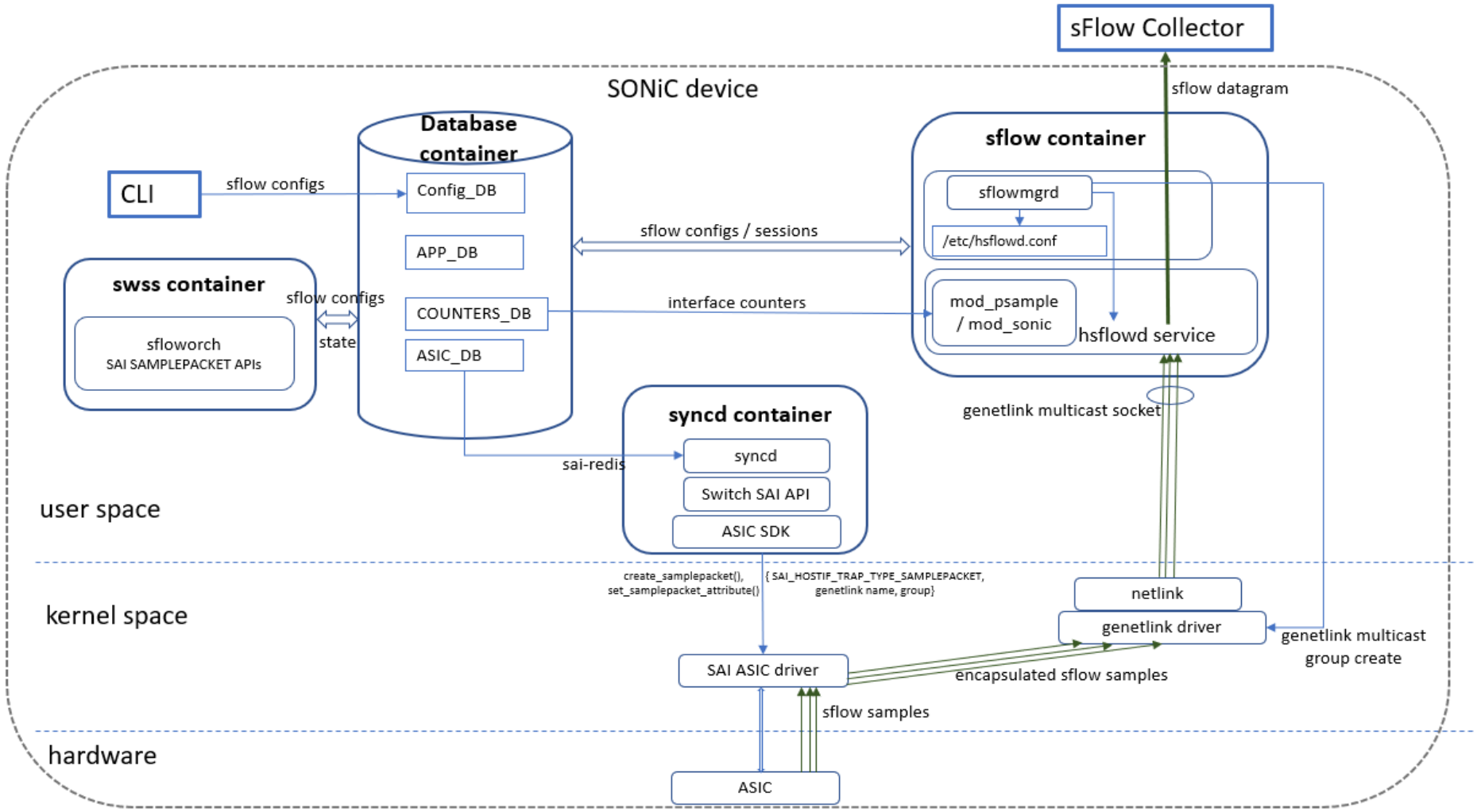
VNET

- VxLAN routing in SONiC
  - Connect Bare Metal machines to cloud VMs
- Provisioning done by controller
  - Routes
  - Neighbors
- No VxLAN device created in Linux
- Underlay routing is programmed by BGP
- As opposed to EVPN design (WIP) that is fully reflected in Linux



SFLOW

- Psample driver ported to Linux 4.9
- NPU drivers are required to support psample
- Netlink as host interface channel for packets
- tc_sample is used for virtual SONiC switch testing environment

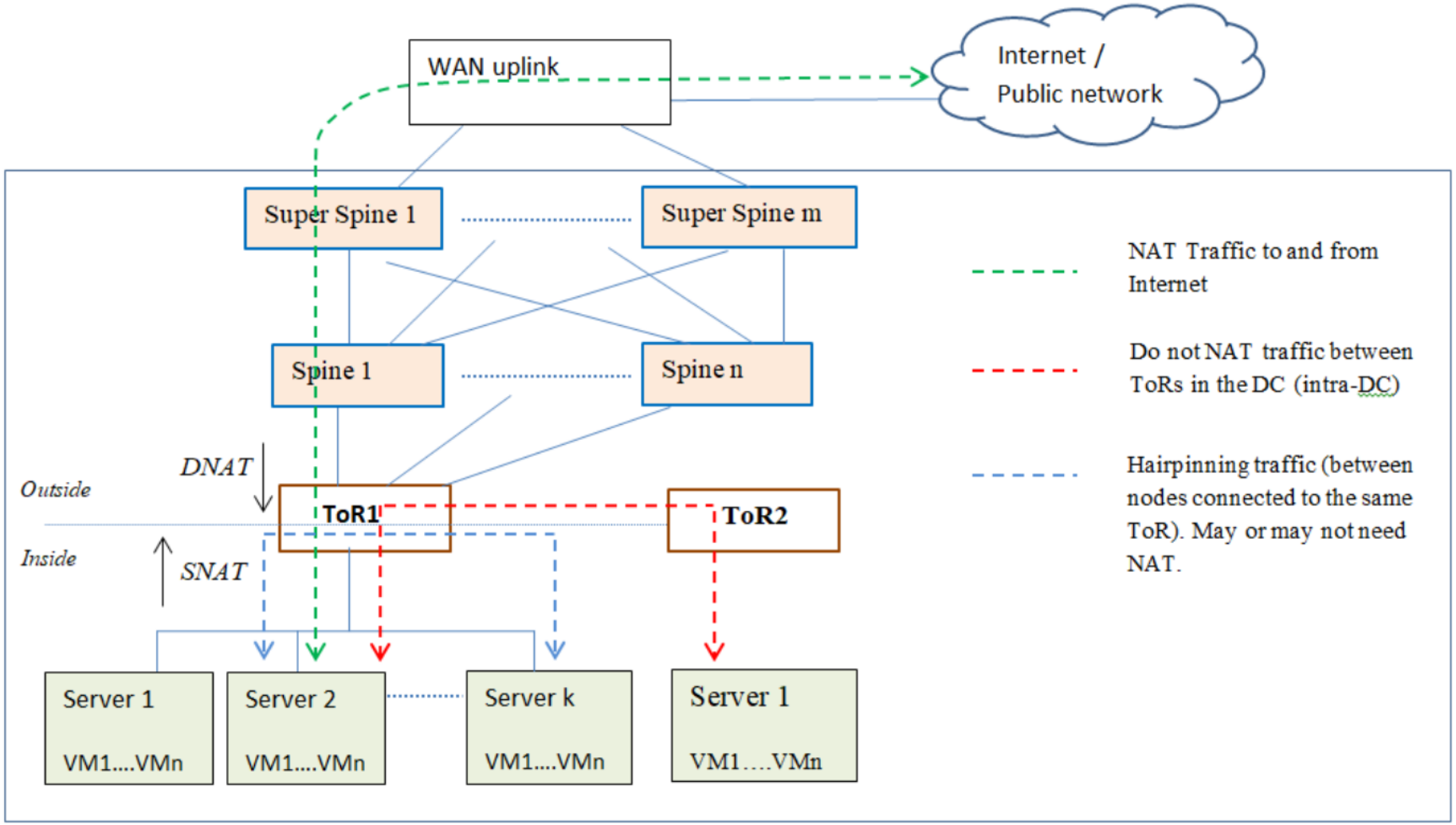


ACL

- ACL in SONiC is used for:
  - Firewalling
  - Data plane telemetry
  - Packet mirroring
  - Enable/Disable NAT
  - PBR
- ACL tables are not programmed in Linux,
  - Except NAT
- SONiC keeps a separate table for control plane ACL
  - Implemented with iptables

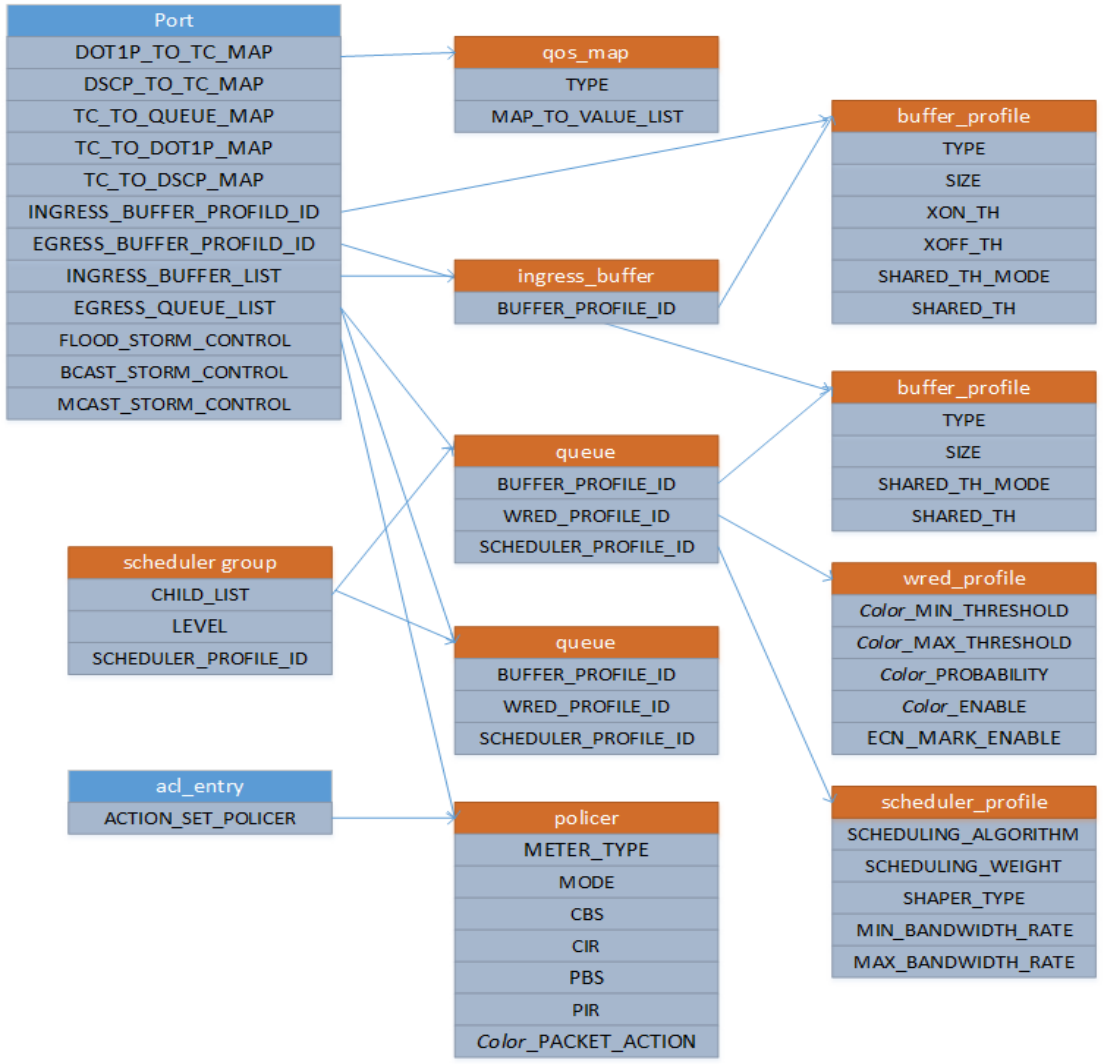
NAT

- Use cases
  - DNAT
  - SNAT
  - Hairpinning
  - NAPT
  - Full cone
- All reflected in Linux with iptables



SWITCH MEMORY MANAGEMENT

- Comprehensive MMU object model
  - QoS maps
  - Shared buffers
  - Policers
  - Schedulers
- Queueing algorithms
  - WRED
  - ECN
- Fine tuning for specific use cases
  - TCP networks
  - RDMA networks
- Programmed directly through SAI



SONIC VIRTUAL SWITCH

- Virtual SONiC platform for generic feature validation
- Two flavors
  - Docker image
  - VM
- Core forwarding behavior modeled by Linux
  - Virtual ports for the data plane
  - L2 forwarding
  - L3 routing
- Sflow
- Control plane ACL

Netlink Message Filter

- Kalimuthu Velappan

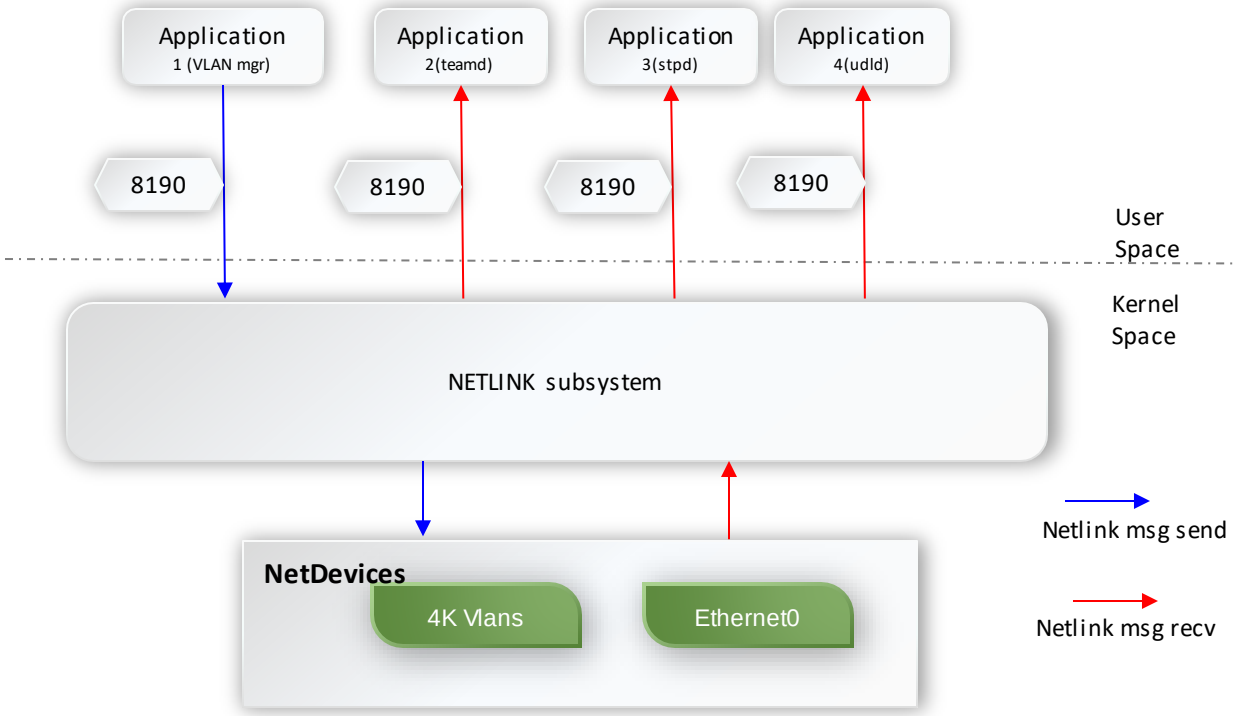
Introduction

- Netlink messaging Architecture
- System Scaling Issue
- Proposed solution
  - BPF – Berkeley Packet Filter
- Q & A

Netlink Messaging Framework

- Network Applications mainly uses the NETLINK_ROUTE family for receiving the netdevice notifications (Port, Up/Down, MTU, Speed, etc.)
- Each netdevice notifies the netlink subsystem about the change in its port-properties
- It is a broadcast domain
- Netlink subsystem posts a netlink message to socket recv-Q of all the registered applications
- Application then reads the message from the recv-Q

System Scaling Issue

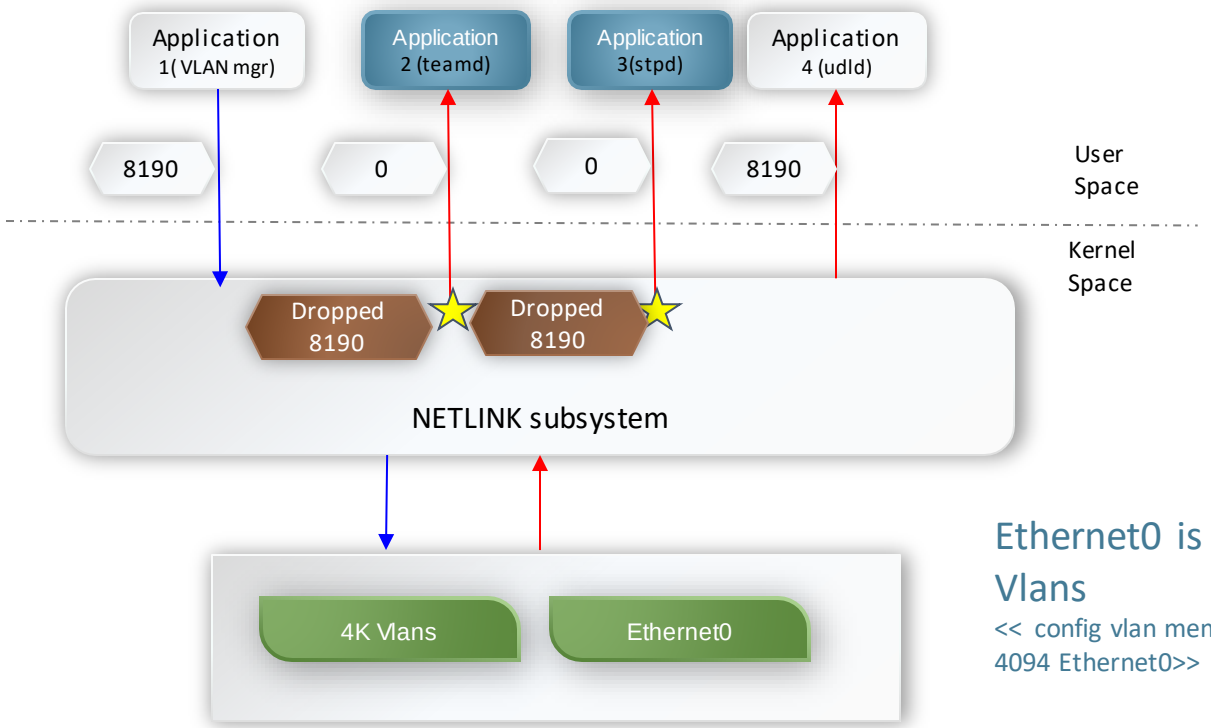


Ex: Ethernet0 is added to 4K Vlans

<<config vlan member range add 2 4094 Ethernet0>>

- Every net device has multiple attributes
- Any attribute change will generate a netlink message notification
- Each application registers for kernel netlink notification
- Application has to receive/process all the messages, whether it's interested in them or not
- When 4K VLAN is configured per port, It generates ~8K netlink messages
- On a scaled system
  - More than 1M unnecessary messages can be broadcast across system
  - Application is not able to process all the messages during config reload and system reboot
  - Due to this burstiness, important Netlink messages might get delayed/dropped in kernel(ENOBUFF)
  - Dropped Netlink messages can't be retrieved!!!

Proposed Solution – Berkeley Packet Filter



- Filter to drop all unwanted netlink messages in Kernel using Berkeley Packet Filter(BPF)
- Filter is applied for each application socket in kernel
- Filter is based on the one or more message attributes
- Application will get a notification only when a requested attribute changes

Ethernet0 is added to 4K Vlans

<< config vlan member range add 2 4094 Ethernet0>>

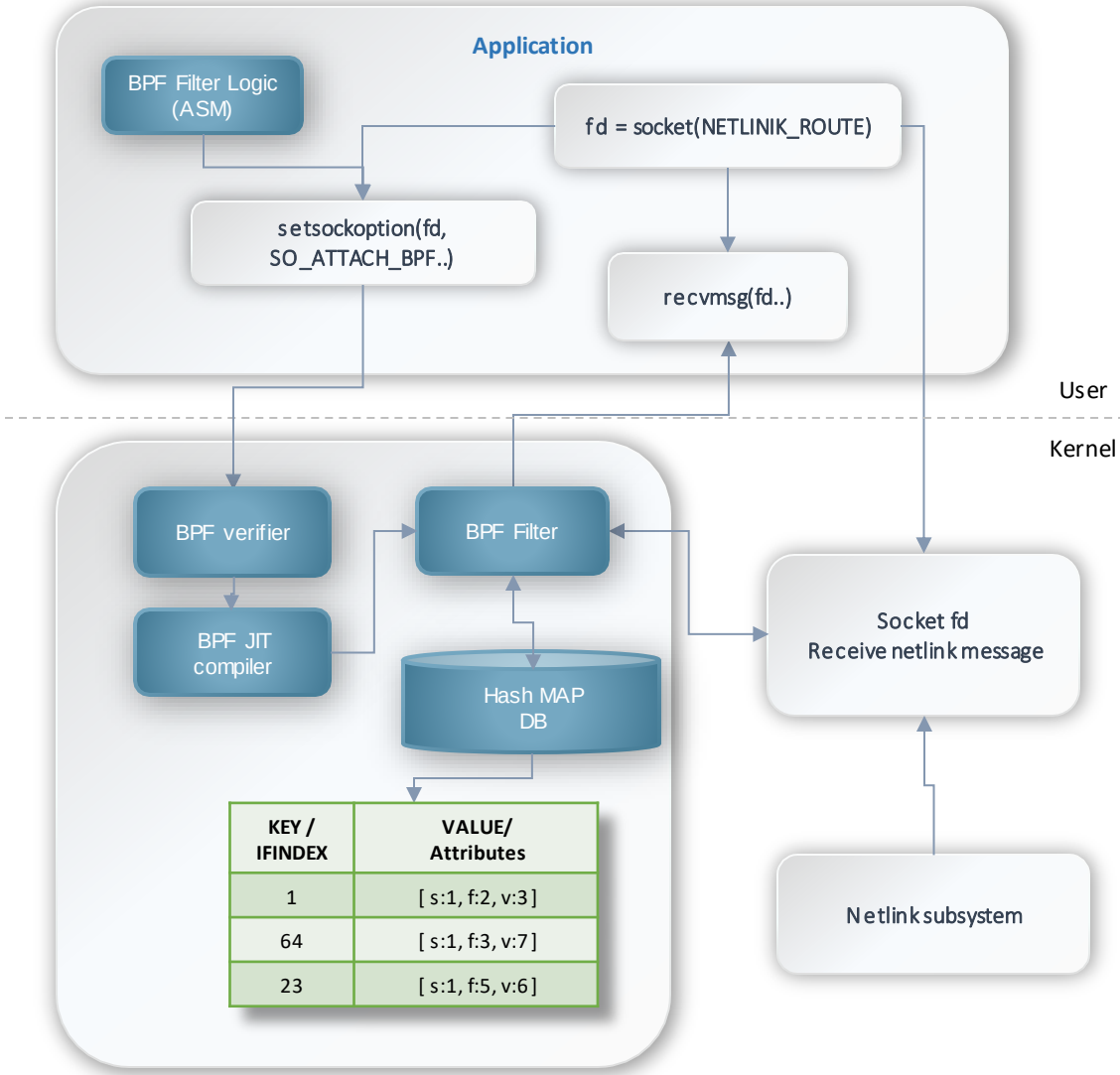
Netlink Message Filtering Mechanism

- Berkeley Socket Filter (BPF) - Interface to execute Micro ASM in the kernel as a Minimal VM
- ASM Filter code gets executed for every packet reception
- Return value decides whether to accept/drop the packet
- Gets executed as part of Netlink message sender context

Required Changes

- Kernel patch for *nlattr* and *nestednlattr* helper functions.
- Customized eBPF filter logic to drop unnecessary messages for the application

- Filter Logic:
  - Entry { KEY: Interface, VALUE { Attr : Value, … } }
  - Hash map to store only the required attribute information
  - It filters all the NetLink messages except the interested attribute changes
  - Application will get notification only when interested attribute changes



Support for full cone NAT

- Kiran Kumar Kella

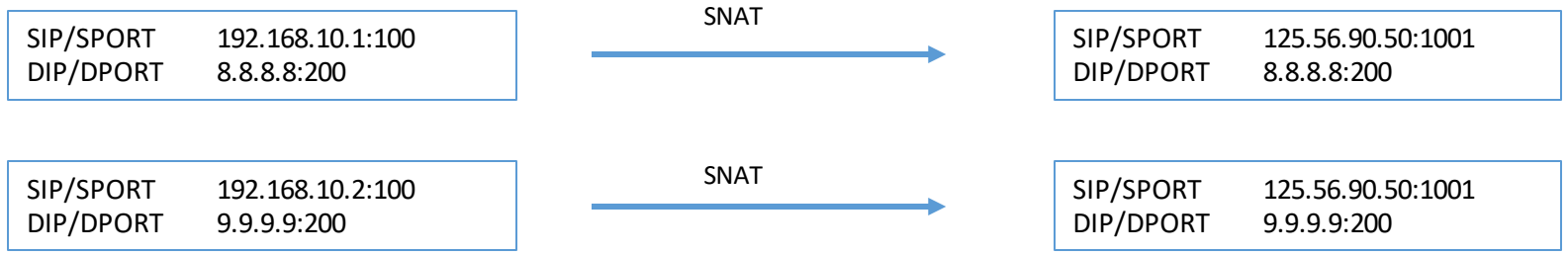
NAT in Linux

- Linux today does NAT based on 5-tuple uniqueness of the translated conntrack entries
- For example, with an iptables rule, the following 2 traffic flows are subjected to SNAT as below

```
#iptables -t nat -nvL

Chain POSTROUTING (policy ACCEPT 33097 packets, 2755K bytes)

 pkts bytes target     prot opt in     out     source               destination
41987 2519K SNAT       udp  --  *      *       192.168.10.0/24      0.0.0.0/0       to:125.56.90.50:1001-2000
```



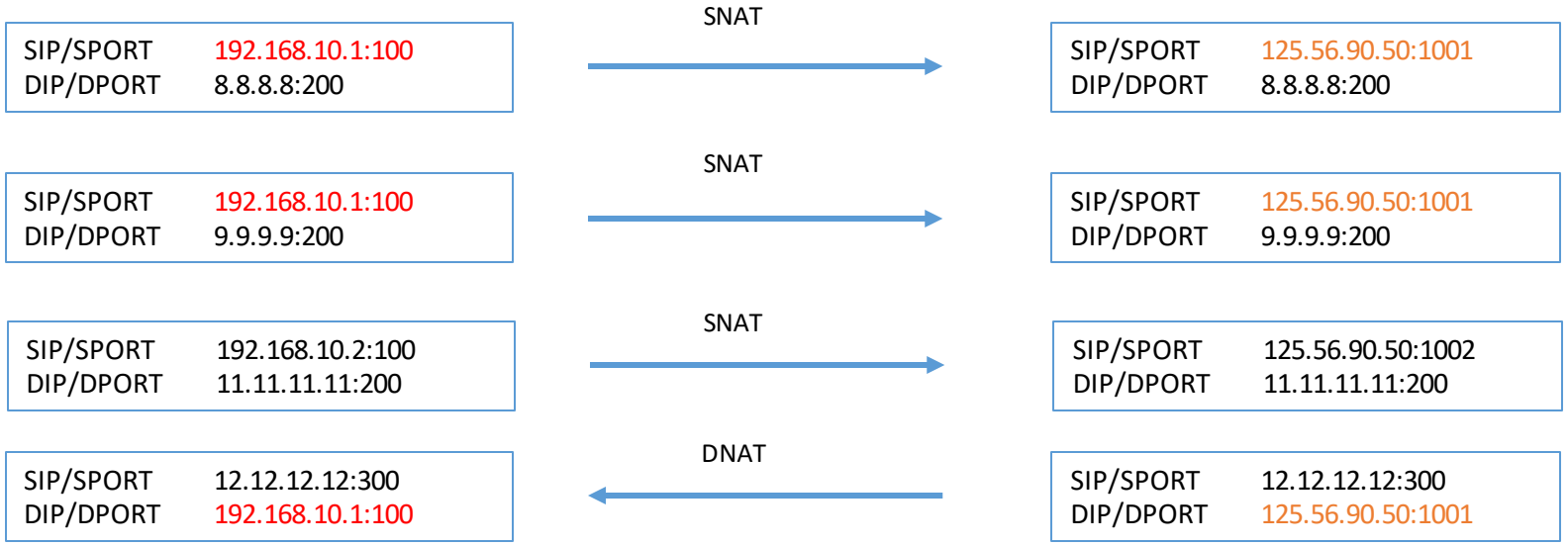
- Both flows SNAT to the same external IP + Port (125.56.90.50:1001) as they are 5-tuple unique [Protocol + SIP + SPORT + DIP + DPORT]

Support for full cone NAT in Linux

- RFC 3489 says:

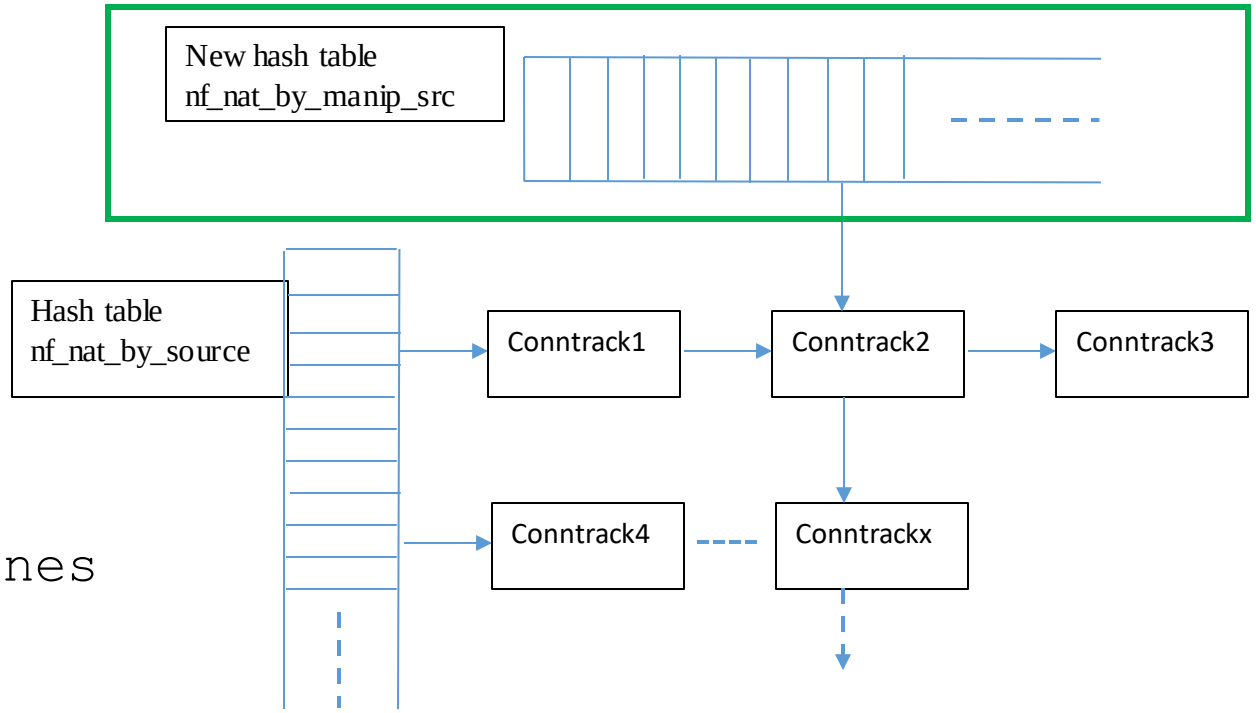
> Full Cone: A full cone NAT is one where all requests from the same internal IP address and port are mapped to the same external IP address and port. Furthermore, any external host can send a packet to the internal host, by sending a packet to the mapped external address.

- Some switching ASICs that can leverage from Linux NAT feature would need full cone NAT support in Linux
- In other words, to support full cone NAT would need 3-tuple uniqueness of the conntrack entries



Changes done in NAT/conntrack modules

- New hash table (nf_nat_by_manip_src) is added as an infra to support the 3-tuple uniqueness. This table hashes on the 3-tuple translated source (Protocol + SIP + SPORT)



- Core changes needed in

`nf_nat_core.c in the routines get_unique_tuple() and nf_nat_l4proto_unique_tuple()`

Changes done in NAT/conntrack modules

- The new hash table is updated during the SNAT to ensure 3-tuple unique translation (full cone) for a given internal IP + Port.
- The same table is looked up by hashing on the destination IP + Port in the reverse direction during DNAT to achieve the full cone behavior.

- Enhancement needed in iptables tool to pass the fullcone option to the kernel

```
#define NF_NAT_RANGE_FULLCONE                    (1 << 6)
```

```
#iptables -t nat -nvL

Chain POSTROUTING (policy ACCEPT 33097 packets, 2755K bytes)

 pkts bytes target     prot opt in     out     source               destination
41987 2519K SNAT       udp  --  *      *       192.168.10.0/24      0.0.0.0/0       to:125.56.90.50:1001-2000 fullcone
```

Questions? Thank You