Flexible Ethernet

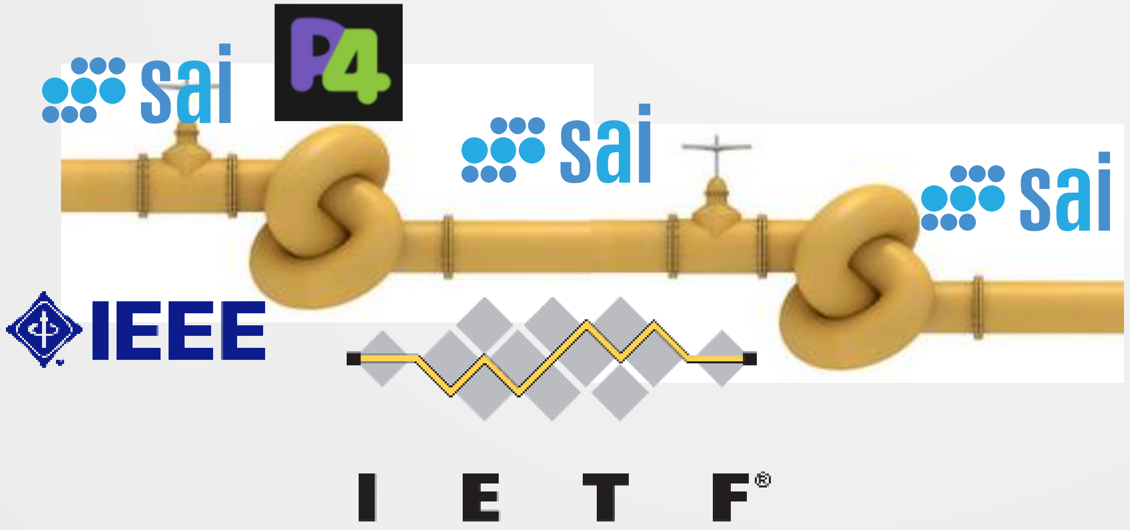
Old world

$$$ on legacy protocols
Best performance and stability
Low feature velocity

New world?

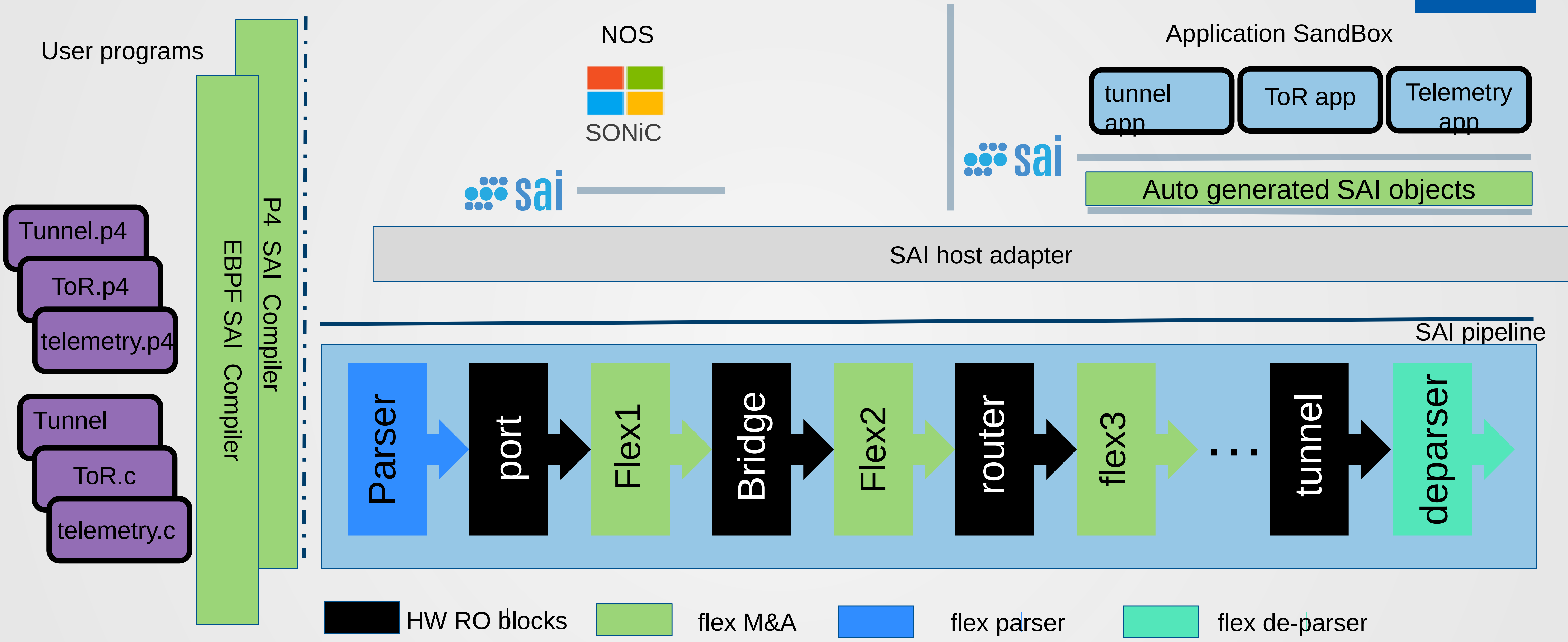
Write everything from scratch
Implement both standard and new applications
Variant feature velocity

Real world



Legacy protocols don't change
Application sand box for home grown needs
Extended HW longevity
High feature velocity

SAI programmability

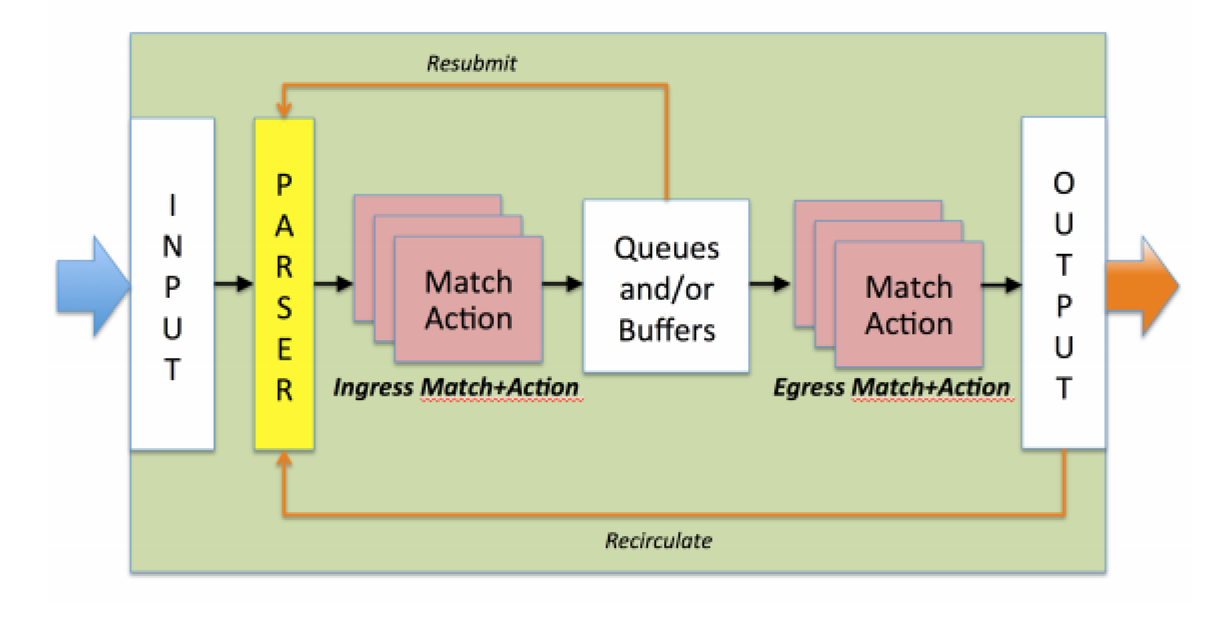


Multiple switching SW options, develop apps not NOS

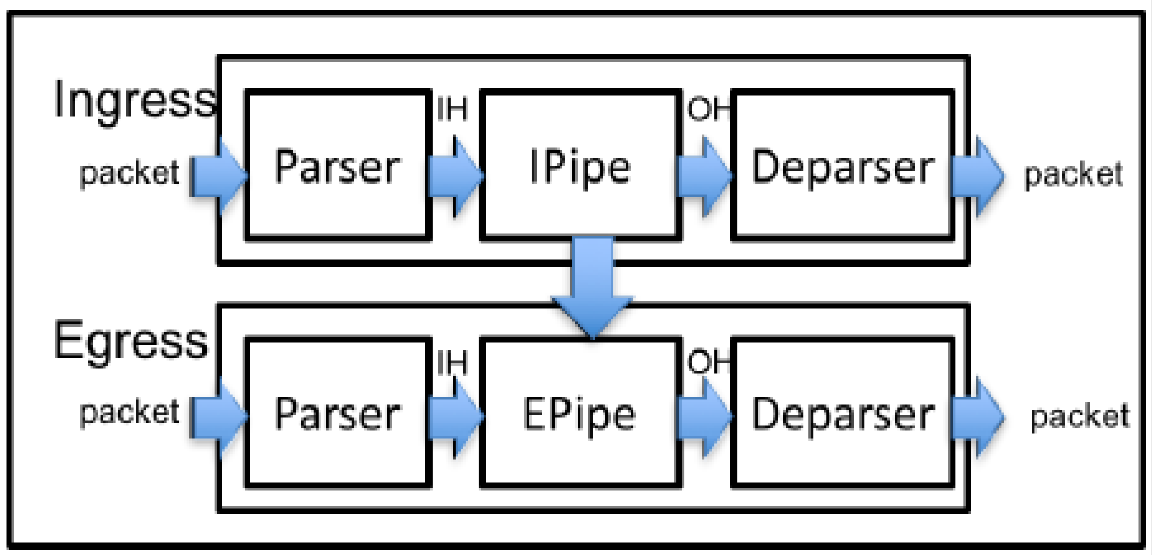
SAIFlexAPI – uniform API for all programing language

P4?

v14

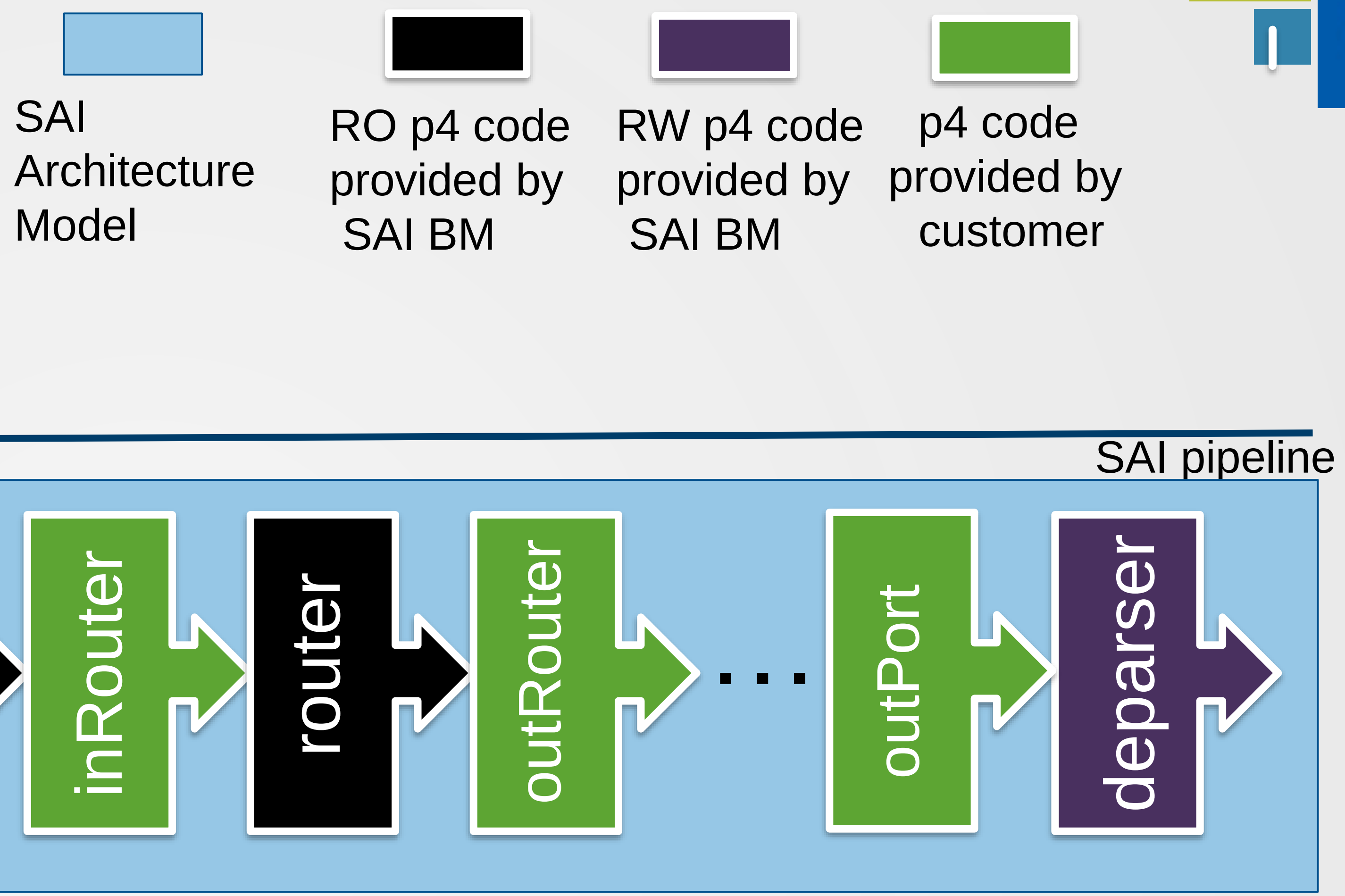


v16



From the spec:
- Introducing P4 architecture description language
- "The P4 architecture can be thought of as a contract between the program and the target"
- " Programmable blocks" i,.e. flexible blocks within a solid target
- "In general, P4 programs are not expected to be portable across different architectures"

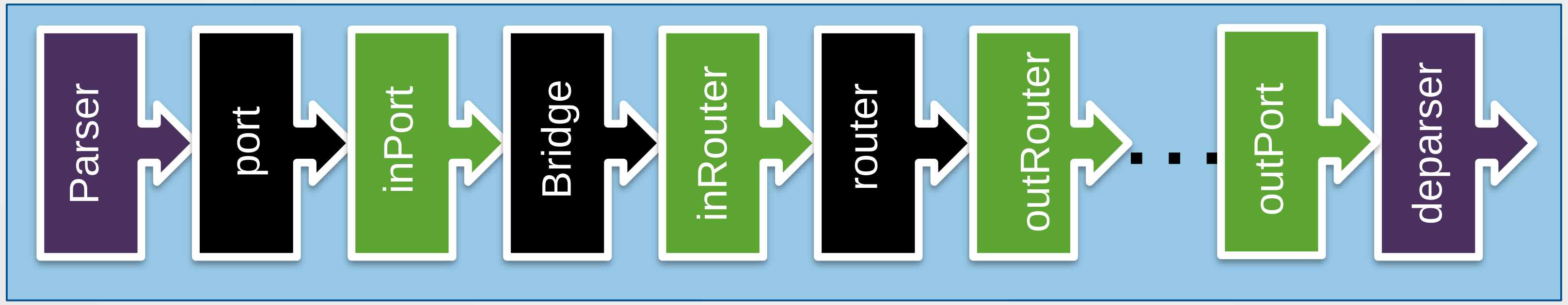
SAI target Architecture



SAI target Architecture

```
package SAI(
        Parser ,
        in_port,
        in_router,
        out_rouer,
        out_port,
        Deparser );
```

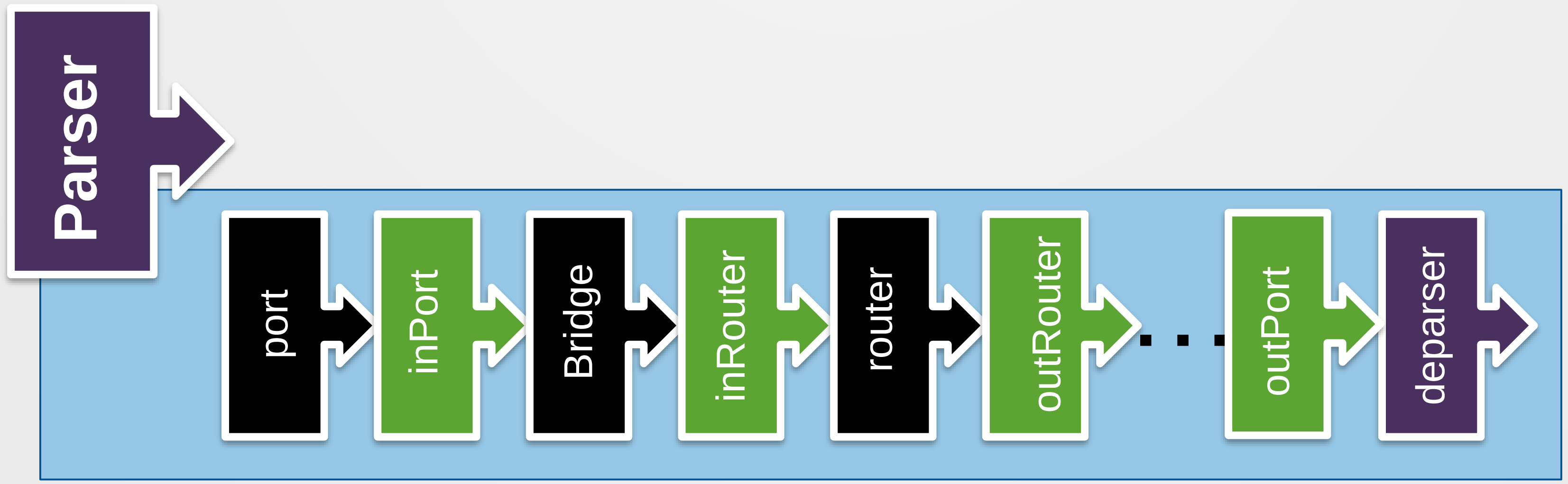
- SAI_header.p4 – SAI header definition
- SAI_parser.p4 - Provides basic parser , can be edit by the user
- SAI_deparser.p4 - Provides basic deparser , can be edit by the user
- SAI_action.p4- SAI supported action , can be extend by vendor
- SAI_metadata<stage>.p4- SAI standard metadata action , can be extend by vendor
  - <stage> - generic, inPort, inRouter …



SAI target Architecture

```
package SAI(
        Parser ,
        in_port,
        in_router,
        out_rouer,
        out_port,
        Deparser );
```

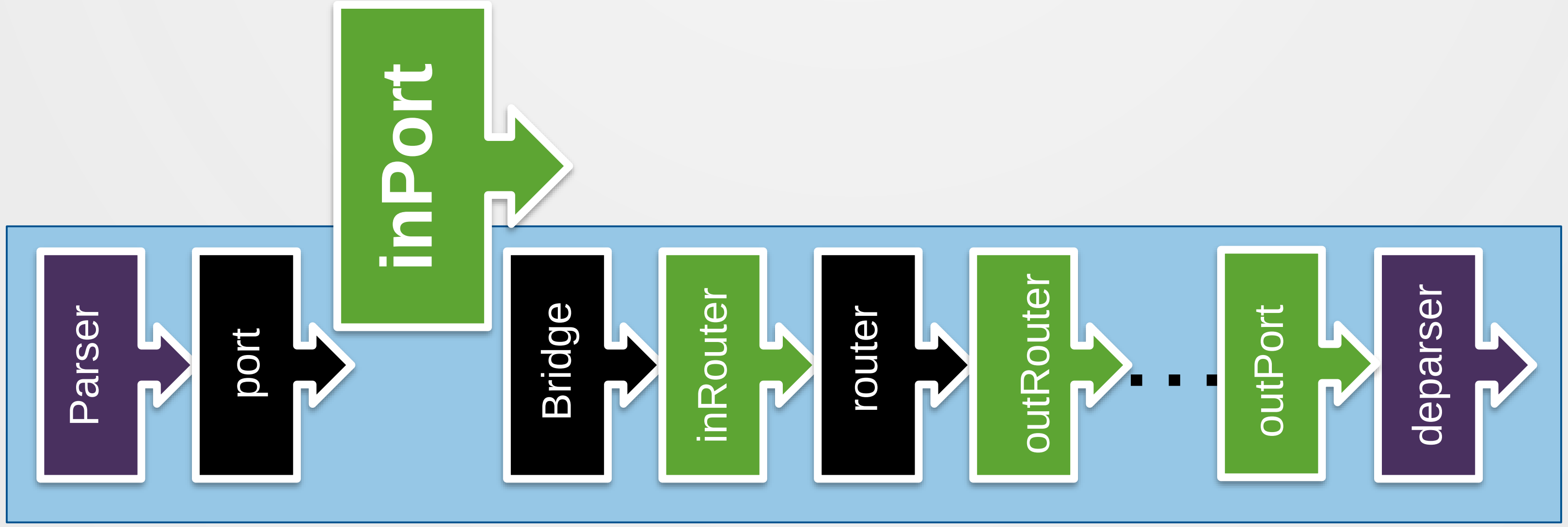
```
control parser(
        in packet ,
        inout generic_meta  g_mata,
        out SAI_header headers);
```



SAI target Architecture

```
package SAI(
        Parser ,
        in_port,
        in_router,
        out_rouer,
        out_port,
        Deparser );
```

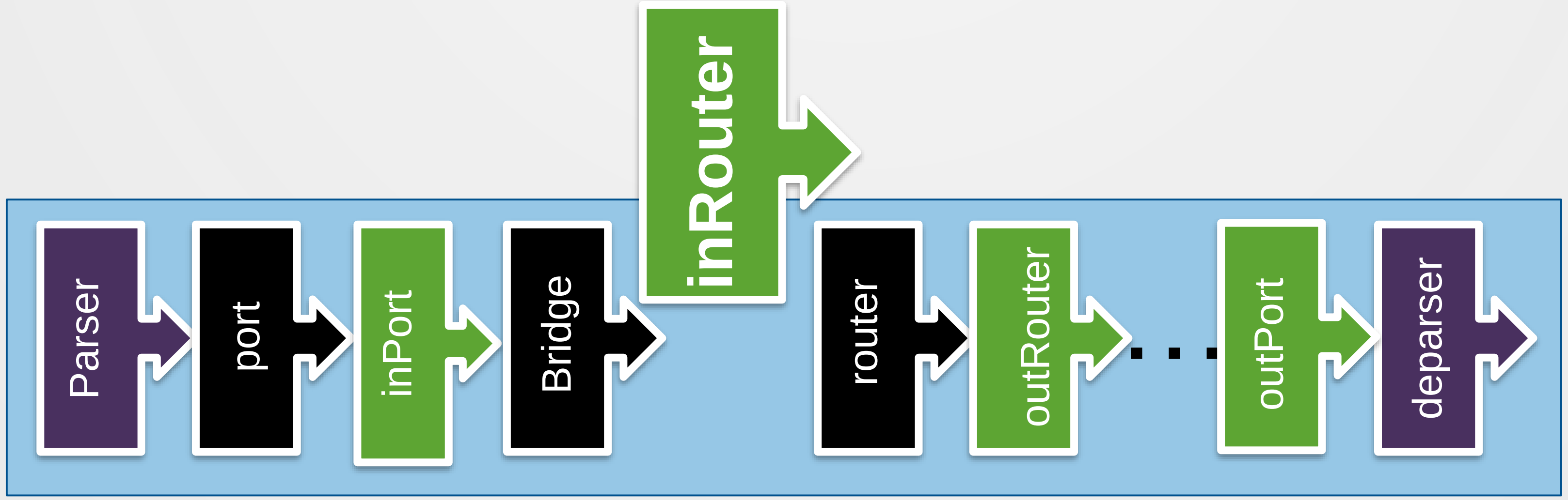
```
control in_port(
        inout SAI_header headers,
        inout generic_meta  g_meta,
        inout in_port_mata  in_port_meta,);
```



SAI target Architecture

package SAI(
        Parser ,
        in_port,
        in_router,
        out_rouer,
        out_port,
        Deparser);

control in_router(
        inout SAI_header headers,
        inout generic_meta g_meta,
        inout in_router_mata in_router_meta,);



Adding Bare Metal services to the Cloud

Goal

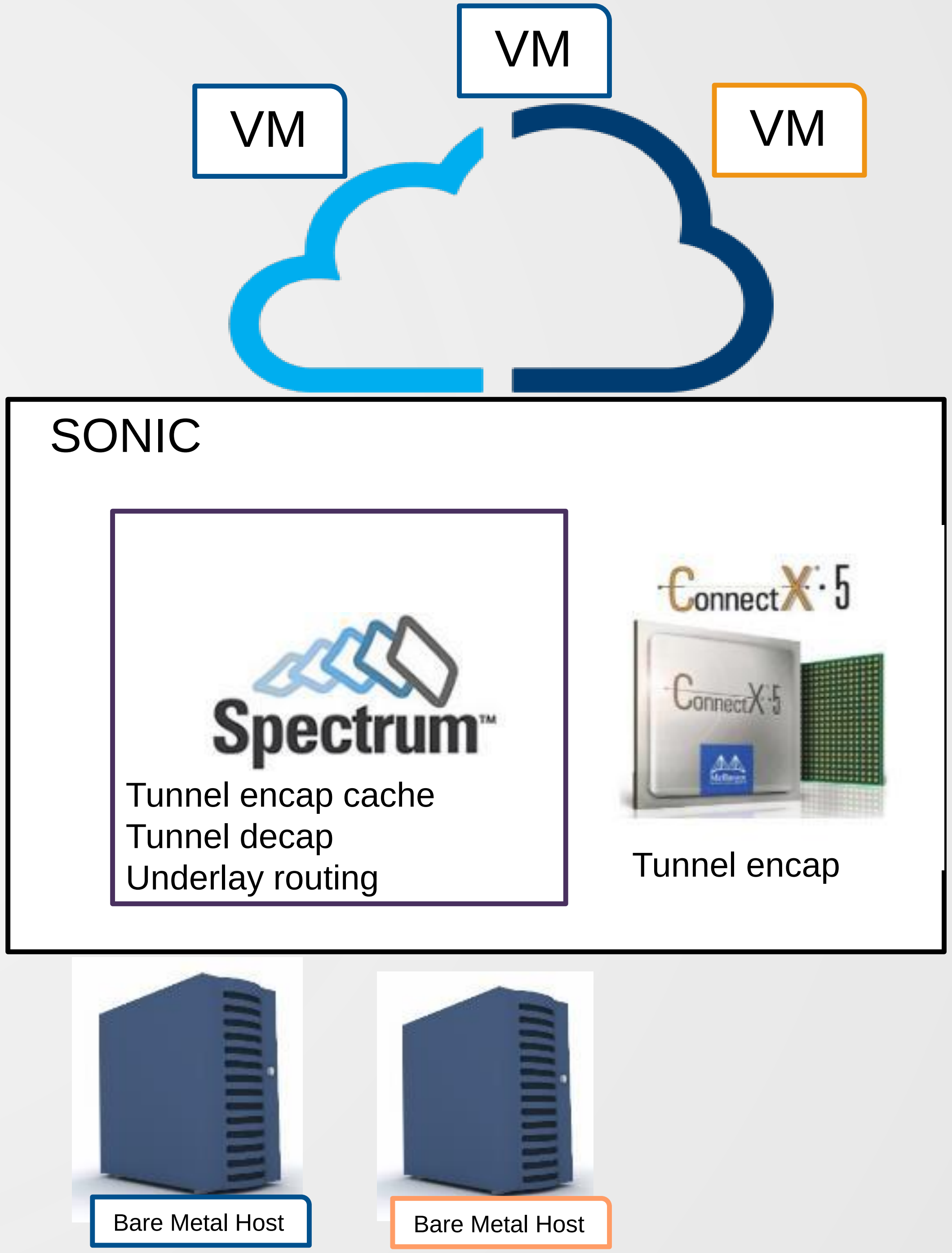
- Connect Bare Metal machine to cloud VMs

Challenges

- Non standard encapsulation logic
- 10M tunnels

Solution

- Programmable pipeline implementation for encapsulation logic across Mellanox ICs
  - Uniform APIs (SAI) for the ConnectX-5 eSWITCH and Spectrum switch
  - On top of legacy switching features
- Host/ Switch integrated pipe
  - Switch role: cache for active recent flows
  - Host role - scale

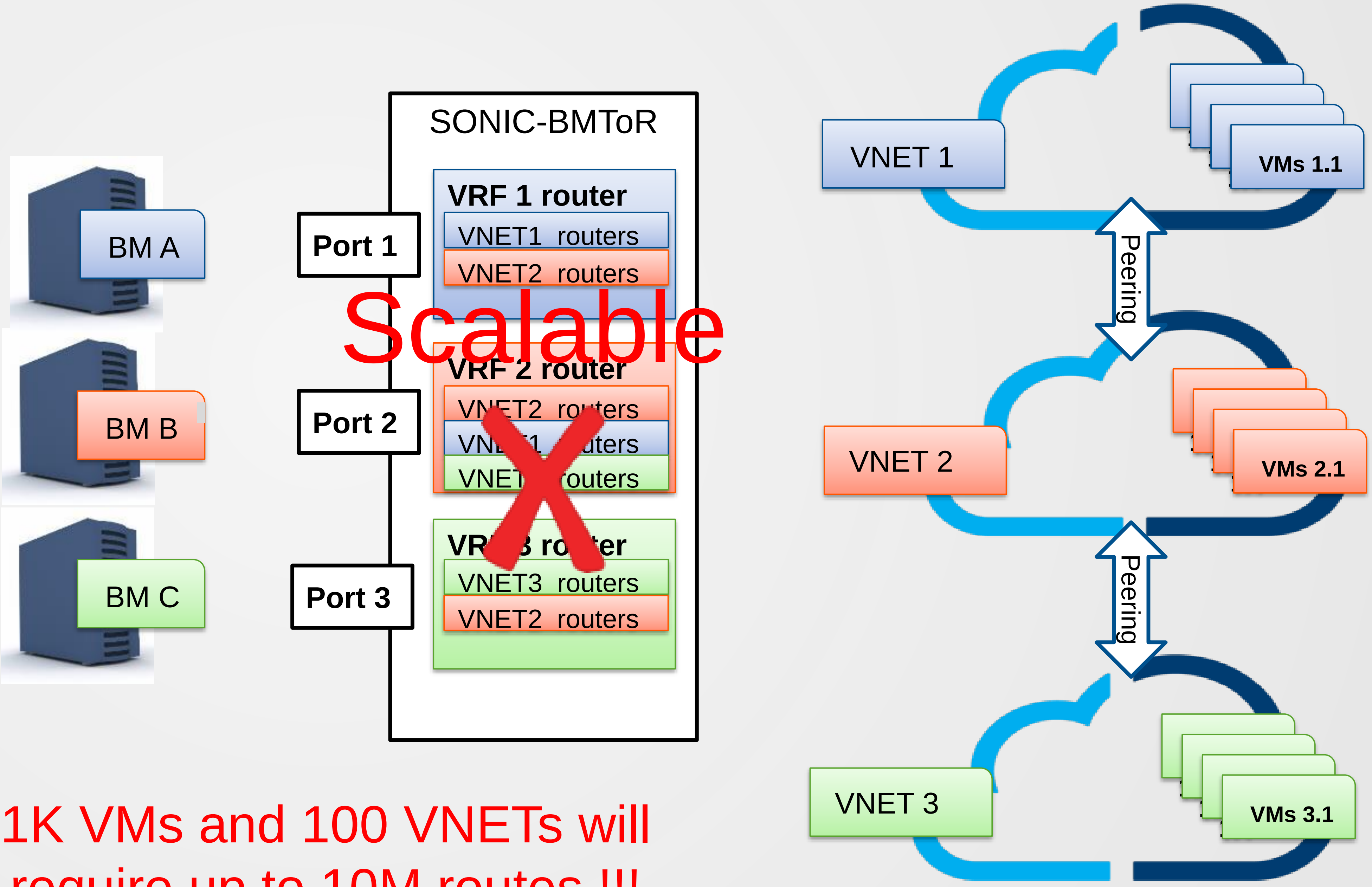


VNET peering in Legacy network

- VNET-virtual network
- VNET peering -Peering between virtual networks

Implementation:

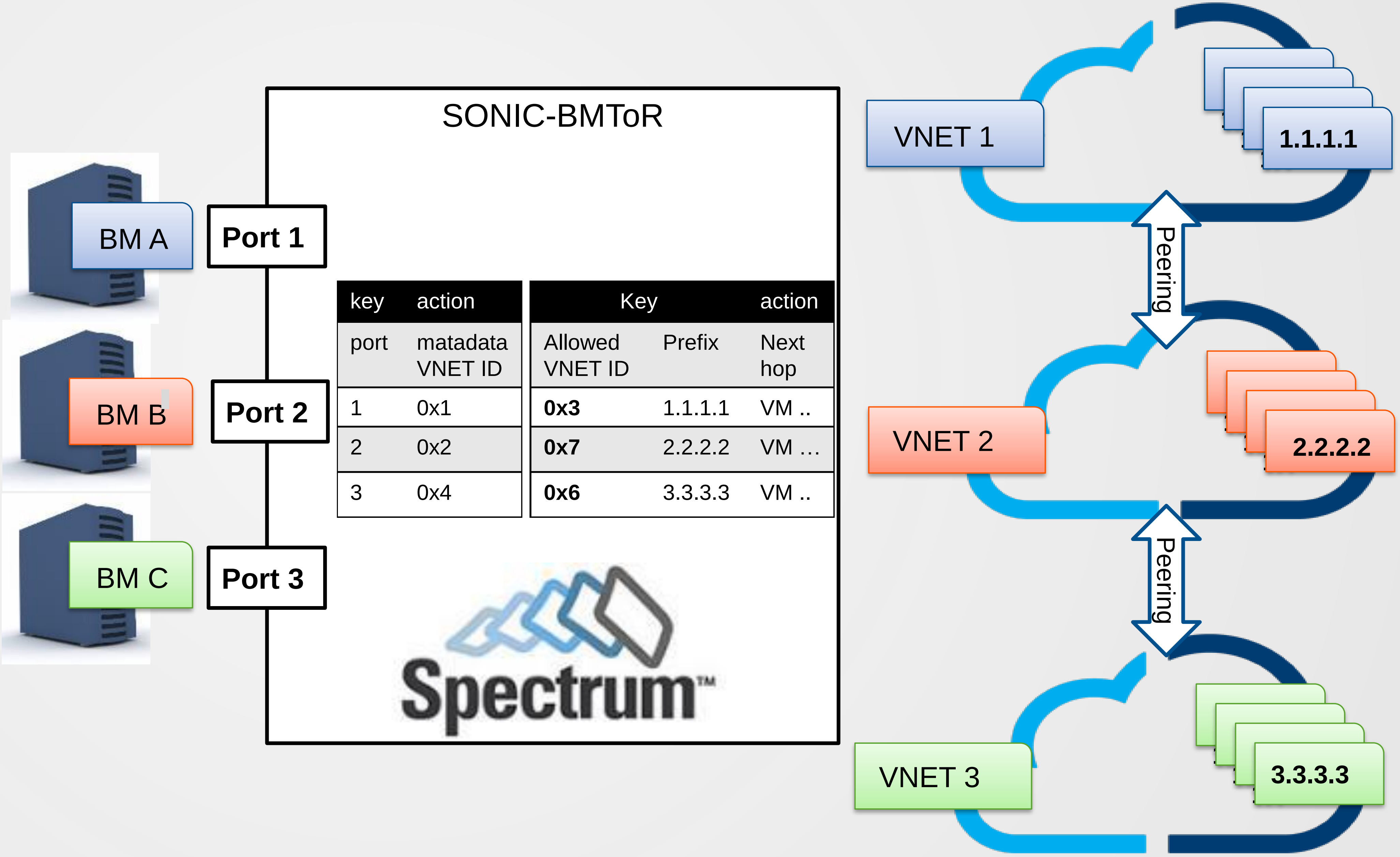
- VNET -> VRF
- VNET1 peering with VNET2 -> copy route from VNET1 to VNET2 and vice versa



1K VMs and 100 VNETs will require up to 10M routes !!!

VNET peering in programmable network

- Two match action tables
- Port to VNET
  - Key: Port
  - Action Set metadata
    - metadata = VNET ID
- VNET routing
  - Key: metadata , prefix
    - metadata vector of VNET peers
  - Action: next hop
- VNET1 peering with VNET2 -> turn on VNET1 VNET ID in VNET routing metadata of all routes originated by VNET2
  - A single route per VM
  - Single update per VM route



key	action
port	matadata VNET ID
1	0x1
2	0x2
3	0x4

Key		action
Allowed VNET ID	Prefix	Next hop
0x3	1.1.1.1	VM ..
0x7	2.2.2.2	VM …
0x6	3.3.3.3	VM ..

ToR SAI pipeline

So let’s code P4

and compile!

P4 Program

Auto generated SAI

P4 SAI compilation file

```
control control_in_port(inout Headers_t headers, inout metadata_t meta, inout standard_metadata_t
standard_metadata){
    #include "../inc/actions.p4"

    action set_vnet_bitmap(bit<12> vnet_bitmap){
        set_meta_reg(vnet_bitmap,0x0fff);
        hit_counter();
    }

    action to_tunnel(bit<32> tunnel_id, bit<32> underlay_dip, bit<16> bridge_id){
        set_bridge(bridge_id);
        vxlan_tunnel_encap(tunnel_id,underlay_dip);
        hit_counter();
    }

    table table_peering{
        key = {
            meta.metadatakeys.METADATA_SRC_PORT :exact;
            // TODO add vrf
        }
        actions = {set_vnet_bitmap;}
        size = PORTNUM;
    }

    table table_vhost{
        key = {
            meta.metadatakeys.METADATA_REG : ternary;
            headers.ip.v4.dstAddr :exact;
        }
        actions = {to_tunnel;
                   to_router;
                   to_port;}
        size=MSEE_TABLE_SIZE;
    }

    apply{
        table_peering.apply();
        table_vhost.apply();
                                                                     39,1          33%
```

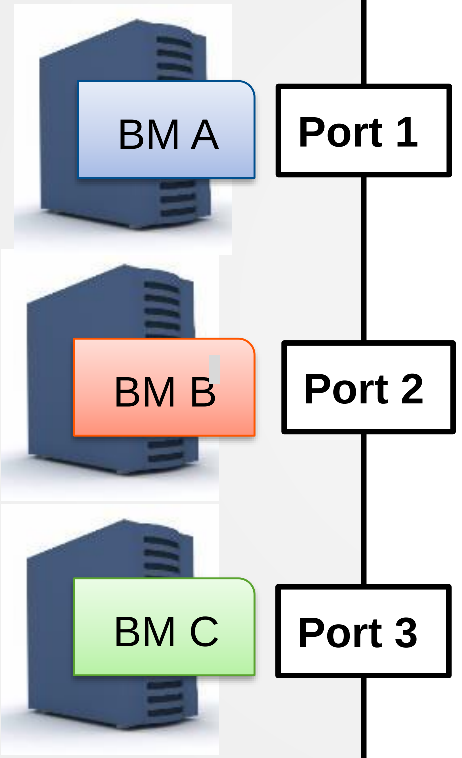
```
yonatanp@yonatanp-VirtualBox:~/p4_16/flextrum$ p4c-mlnx-spc p4src/bm_tor/bm_tor.p4 -o bmtor.
json
Spectrum backend
```

VNET peering in programmable network

1K VMs and 100 VNETs will require only 100k routes

- Two match action tables
- Port to VNET
  - Key: Port
  - Action Set metadata
    - metadata = VNET ID
- VNET routing
  - Key: metadata , prefix
    - metadata vector of VNET peers
  - Action: next hop
- VNET1 peering with VNET2 -> turn on VNET1 VNET ID in VNET routing metadata of all routes originated by VNET2
  - A single route per VM
  - Single update per VM route

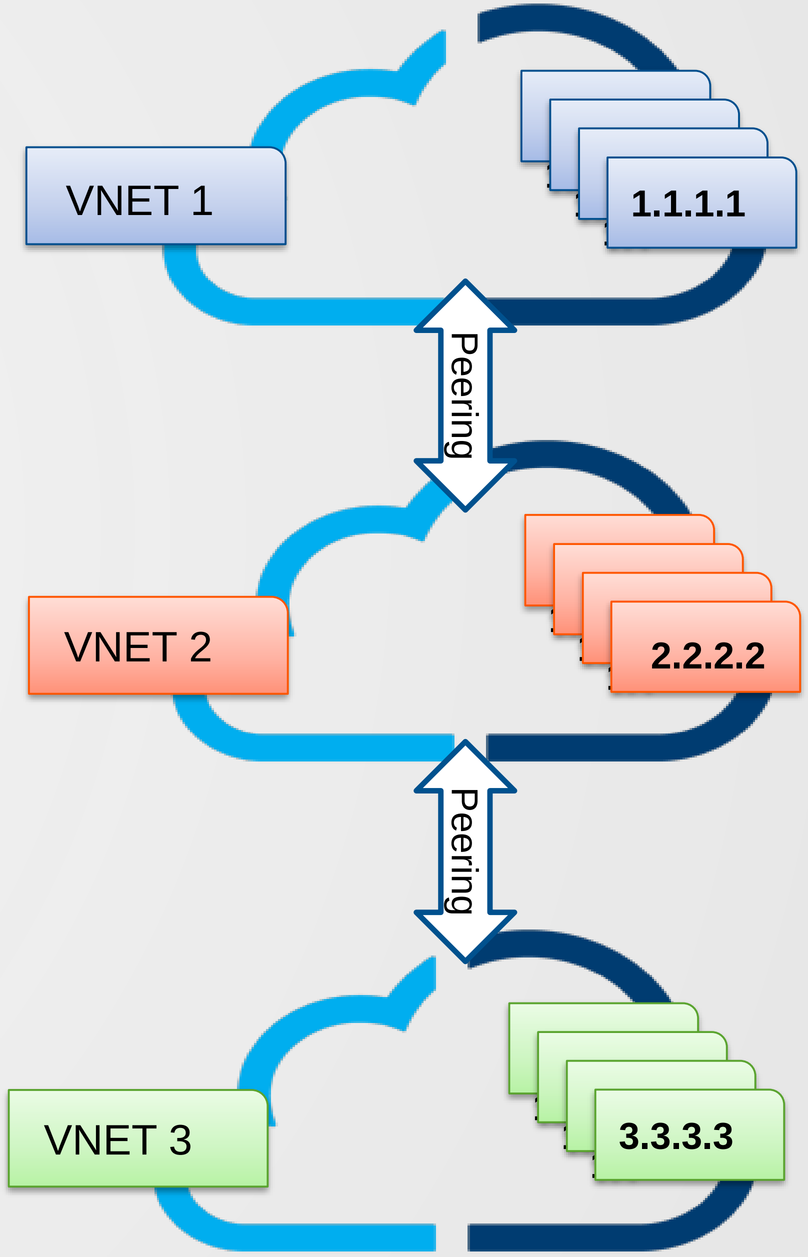
SONIC-BMToR



key	action
port	matadata VNET ID
1	0x1
2	0x2
3	0x4

Key		action
Allowed VNET ID	Prefix	Next hop
0x3	1.1.1.1	VM ..
0x7	2.2.2.2	VM …
0x6	3.3.3.3	VM ..

Spectrum™



But still 10K VMs and 1K VNETs will require 10M routes !!!