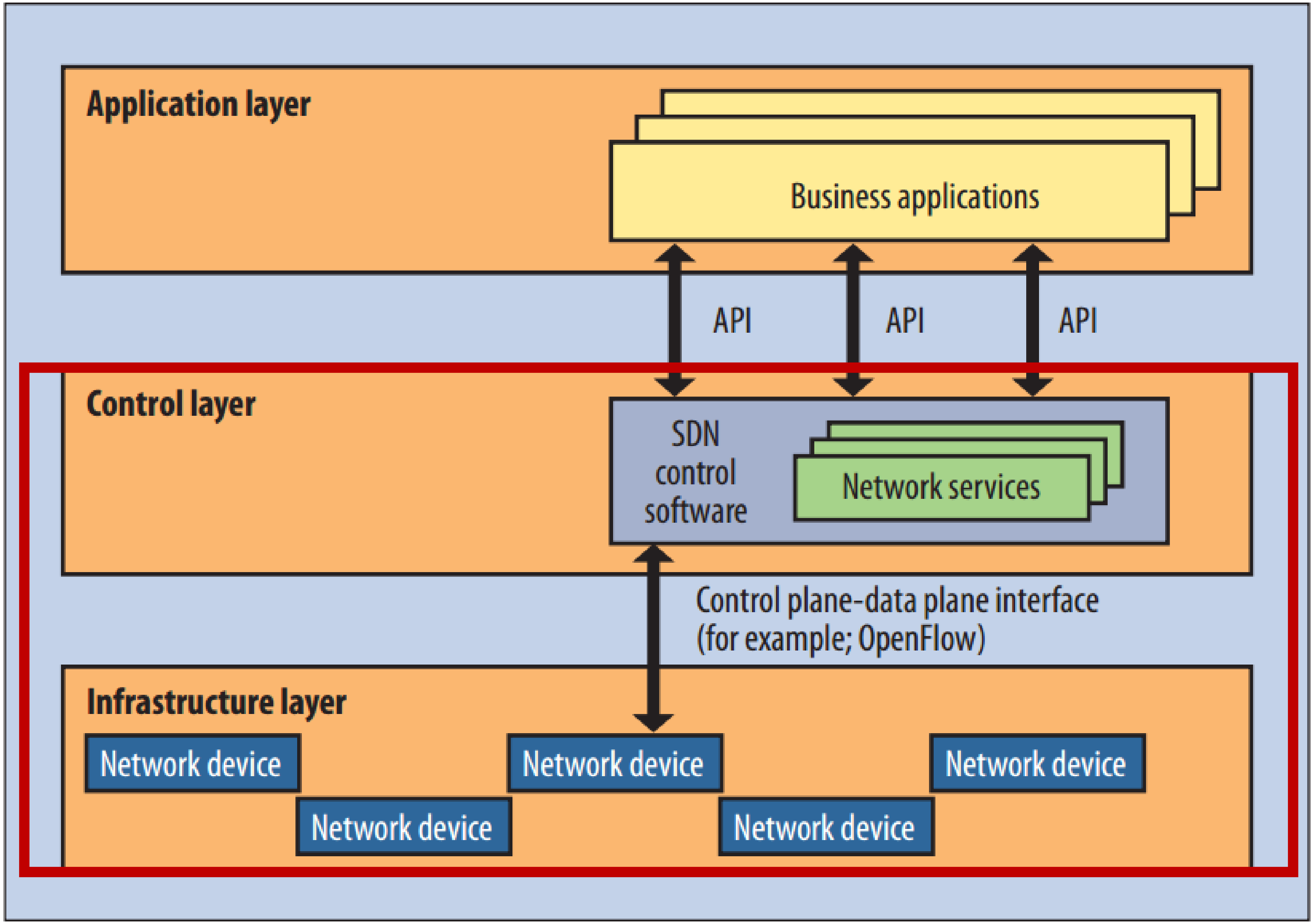
Software-defined Networking
Session III

Block Course – Winter 2016/17

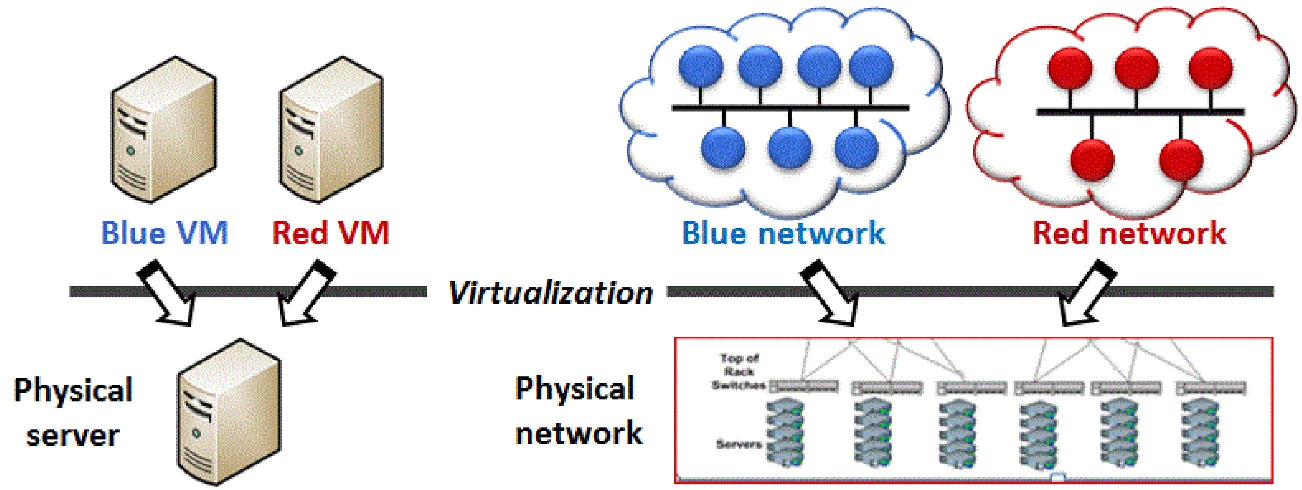
David Koll

Network Virtualization

This Lecture



Virtualizing OpenFlow



Server virtualization

- Run multiple virtual servers on a physical server
- Each VM has illusion it is running as a physical server

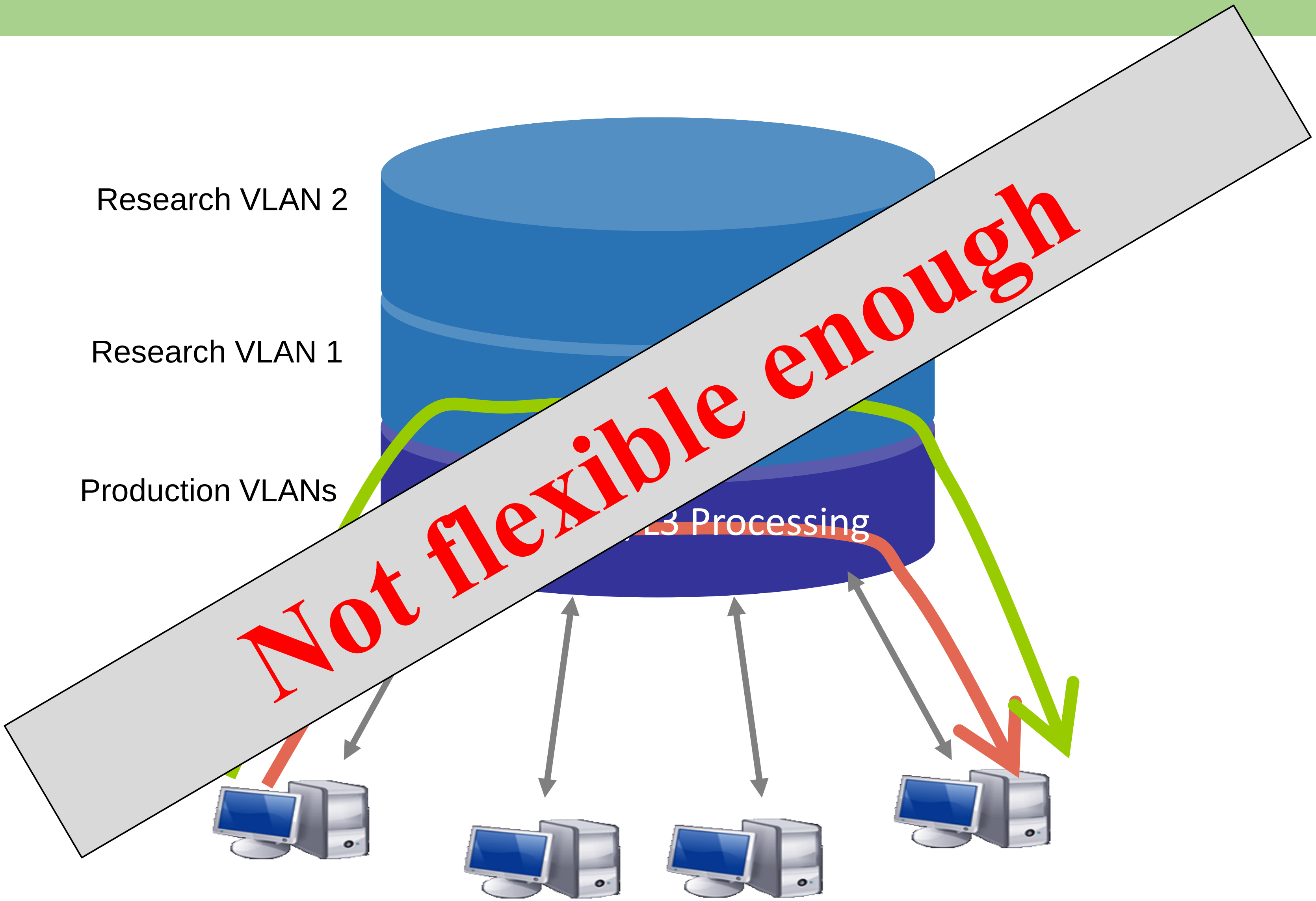
Network virtualization

- Run multiple virtual networks on a physical network
- Each virtual network has illusion it is running as a physical network

Virtualizing OpenFlow

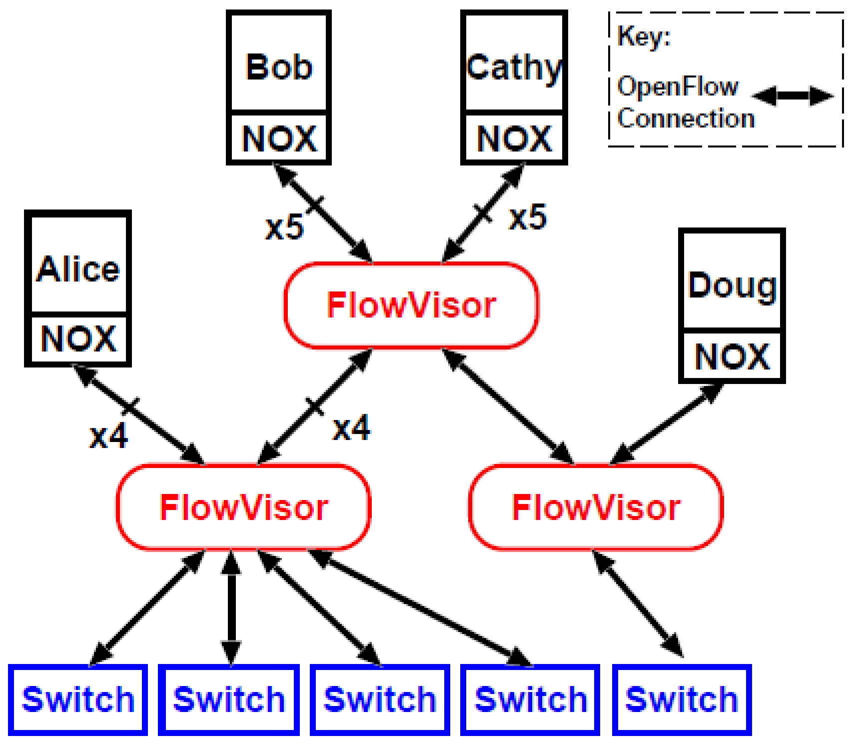
- Network operators "delegate" control of subsets of network hardware and/or traffic to other network operators or users
- Multiple controllers can talk to the same set of switches
- Imagine a hypervisor for network equipments
- Allow experiments to be run on the network in isolation of each other and production traffic

Virtualization: VLANs



FlowVisor

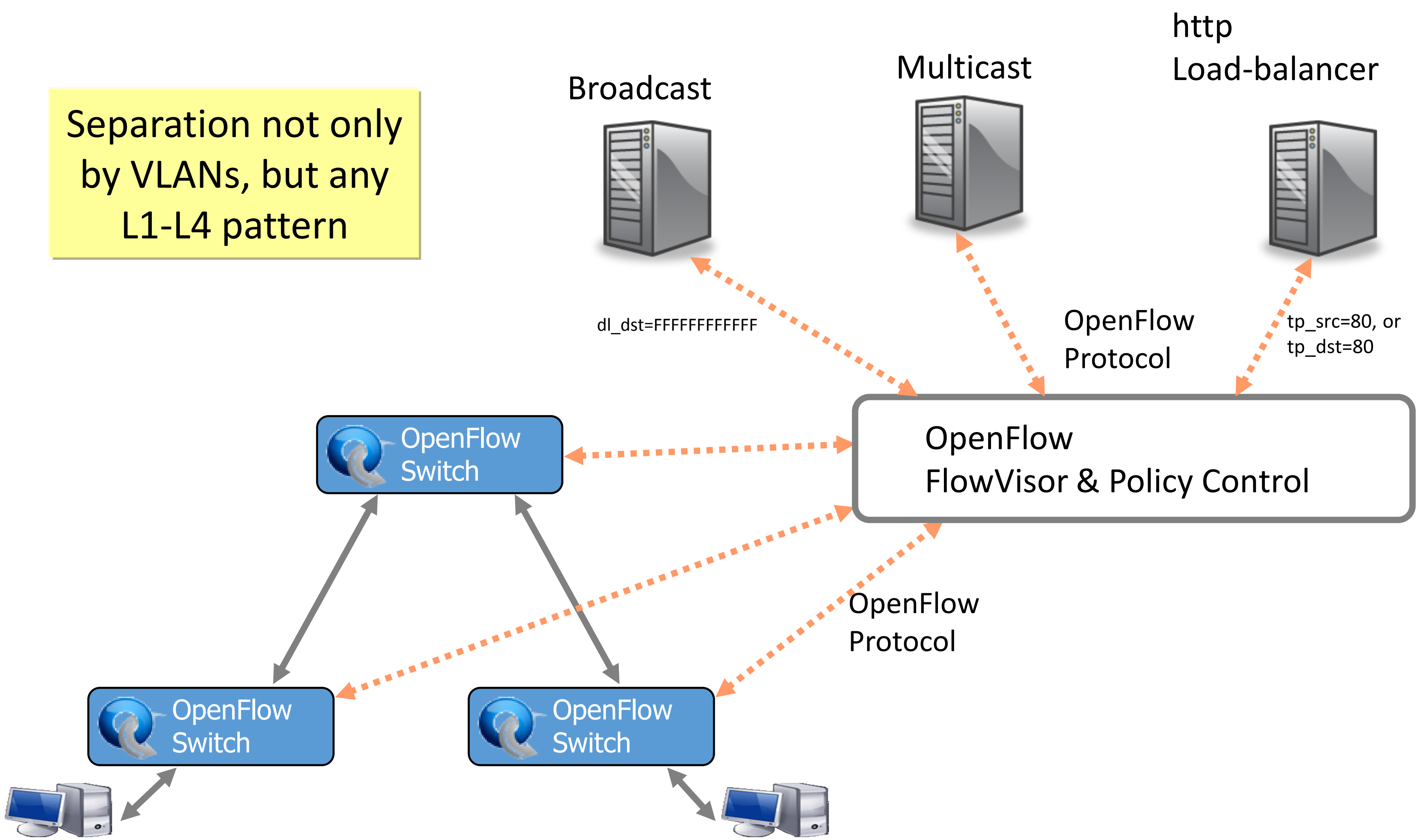
- A network hypervisor developed by Stanford
- A software proxy between the forwarding and control planes of network devices



Sherwood, Rob, et al. "Flowvisor: A network virtualization layer." *OpenFlow Switch Consortium, Tech. Rep* (2009): 1-13.

FlowVisor-based Virtualization

Separation not only by VLANs, but any L1-L4 pattern



Slicing Policies

- The policy specifies resource limits for each slice:
  - Topology
  - Link bandwidth
  - Fraction of switch/router CPU
  - Maximum number of forwarding rules

  - *FlowSpace: which packets does the slice control?*

FlowVisor Resource Limits

- FV assigns hardware resources to “Slices”
  - Topology
    - Network Device or Openflow Instance (DPID)
    - Physical Ports
  - Bandwidth
    - Each slice can be assigned a per port queue with a fraction of the total bandwidth

FlowVisor Resource Limits (cont.)

- FV assigns hardware resources to "Slices"
  - CPU
    - Employs Course Rate Limiting techniques to keep new flow events from one slice from overrunning the CPU
  - Forwarding Tables
    - Each slice has a finite quota of forwarding rules per device

FlowVisor FlowSpace

- FlowSpace is defined by a collection of packet headers and assigned to “Slices”
  - Source/Destination MAC address
  - VLAN ID
  - Ethertype
  - IP protocol
  - Source/Destination IP address
  - ToS/DSCP
  - Source/Destination port number

Use Case: VLAN Partitioning

- Basic Idea: Partition Flows based on Ports and VLAN Tags
  - Traffic entering system (e.g. from end hosts) is tagged
  - VLAN tags consistent throughout substrate

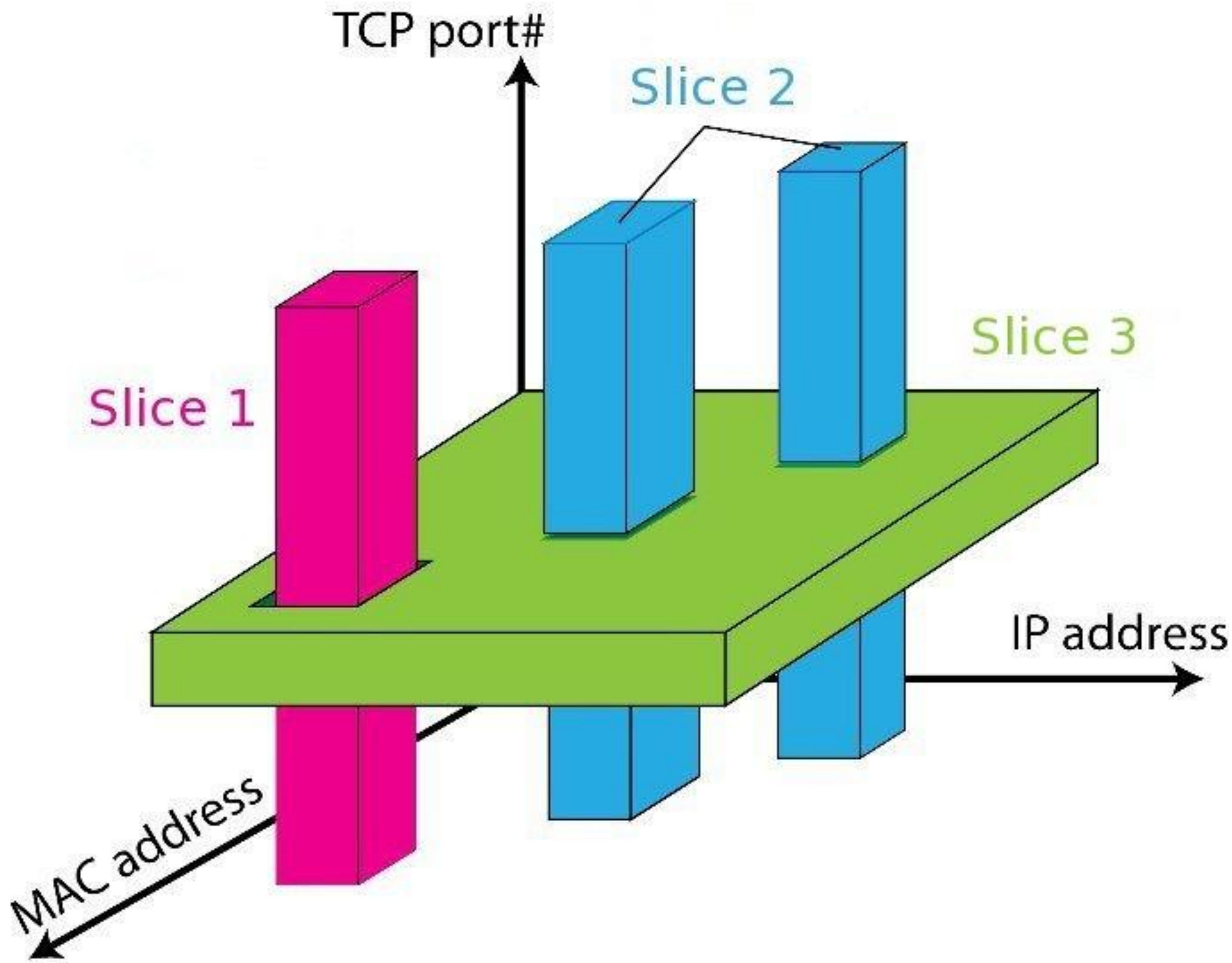
	Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport
Dave	*	*	*	*	1,2,3	*	*	*	*	*
Larry	*	*	*	*	4,5,6	*	*	*	*	*
Steve	*	*	*	*	7,8,9	*	*	*	*	*

Use Case: Content Distribution Network

- Basic Idea: Build a CDN where you control the entire network
  - All traffic to or from CDN IP space controlled by rules
  - All other traffic controlled by default routing
  - Topology is the entire network

	Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport
From CDN	*	*	*	*	*	84.65.*	*	*	*	*
To CDN	*	*	*	*	*	*	84.65.*	*	*	*
Default	*	*	*	*	*	*	*	*	*	*

FlowSpace: Maps Packets to Slices



Sherwood, Rob, et al. "Carving research slices out of your production networks with OpenFlow." *ACM SIGCOMM CCR* 40.1 (2010): 129-130.

FlowVisor Slicing Policy

- FlowVisor intercepts OpenFlow messages from devices
  - Send control plane messages to the slice controller only if source is in slice topology.
  - Rewrite OpenFlow feature negotiation messages so the slice controller only sees the ports in it's slice
  - Port up/down messages are pruned and only forwarded to affected slices

FlowVisor Slicing Policy

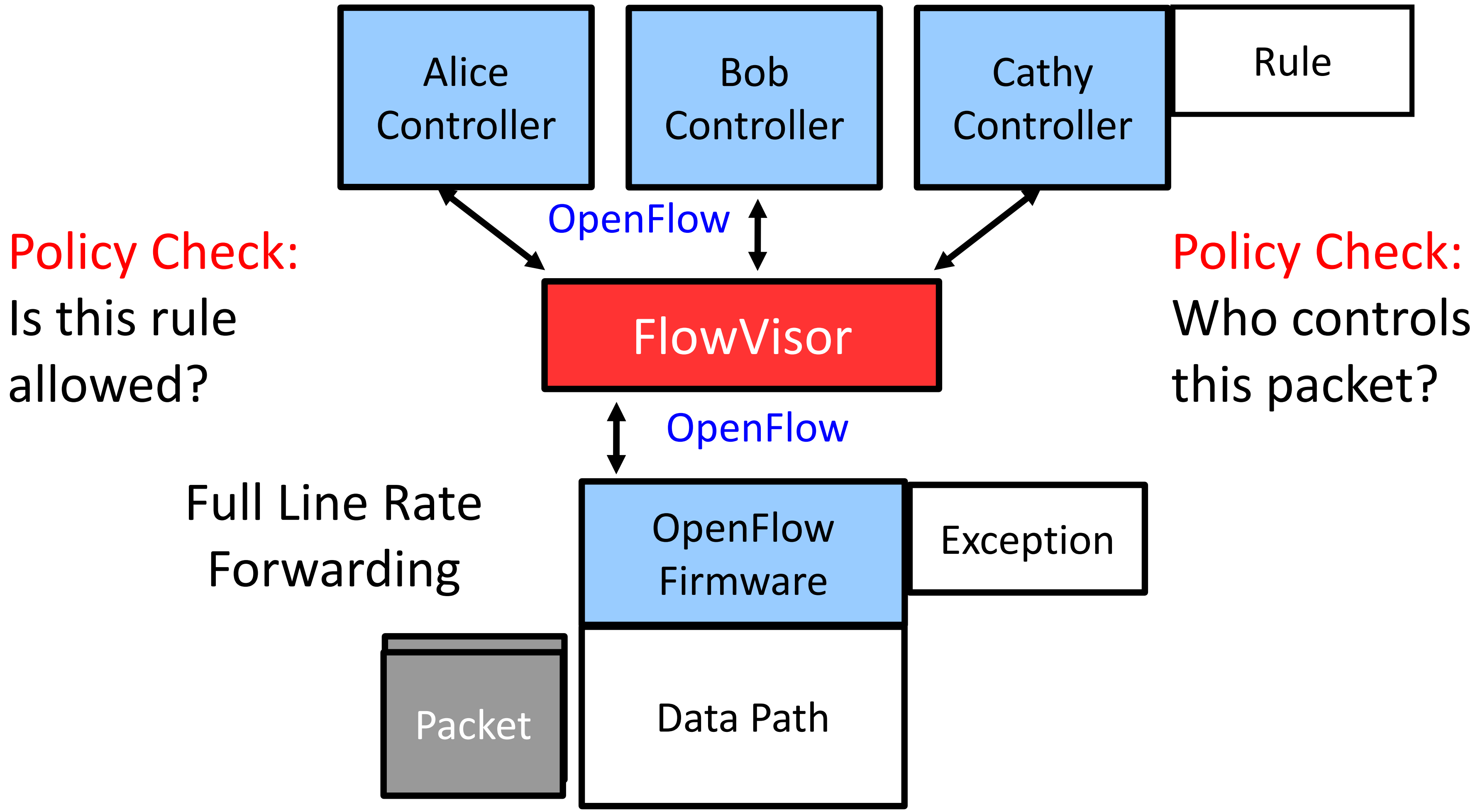
- FlowVisor intercepts OpenFlow messages from controllers
  - Rewrites flow insertion, deletion & modification rules so they don't violate the slice definition
    - Flow definition – e.g., limit control to HTTP traffic only
    - Actions – e.g., limit forwarding to only ports in the slice

FlowVisor Slicing Policy

- FlowVisor intercepts OpenFlow messages from controllers
  - Expand Flow rules into multiple rules to fit policy
    - Flow definition – e.g., if there is a policy for John's HTTP traffic and another for Alice's HTTP traffic, FV would expand a single rule intended to control all HTTP traffic into 2 rules.
    - Actions – e.g., rule action is send out all ports. FV will create one rule for each port in the slice.

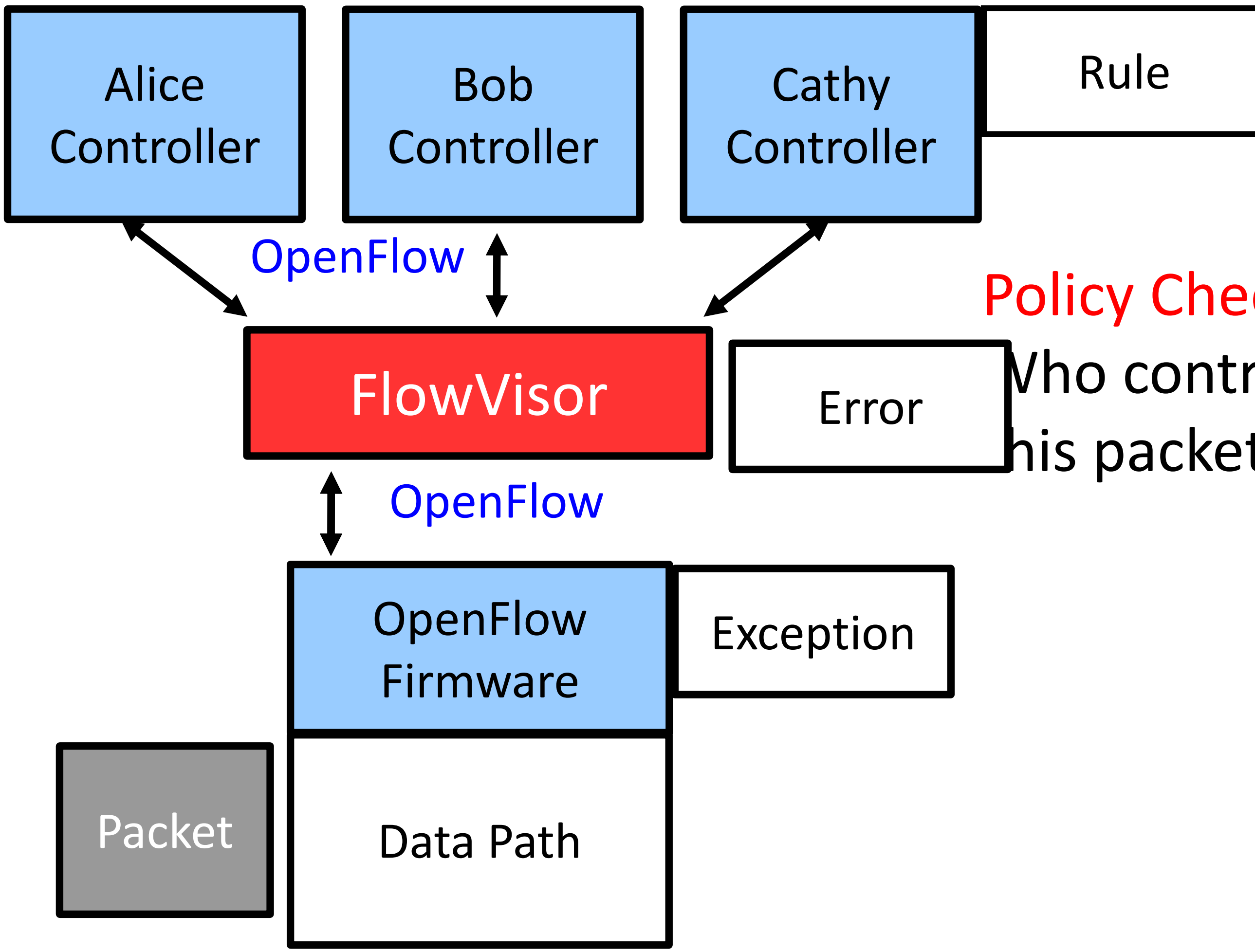
    - Returns "action is invalid" error if trying to control a port outside of the slice

FlowVisor Message Handling



FlowVisor Message Handling

Policy Check:
Is this rule allowed?



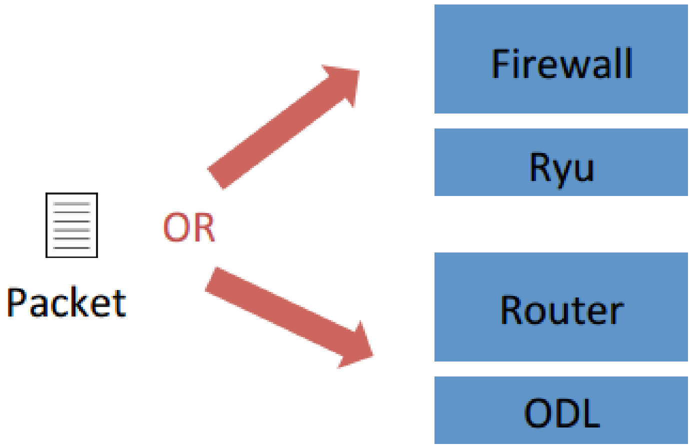
Policy Check:
Who controls this packet?

FlowVisor in Practice

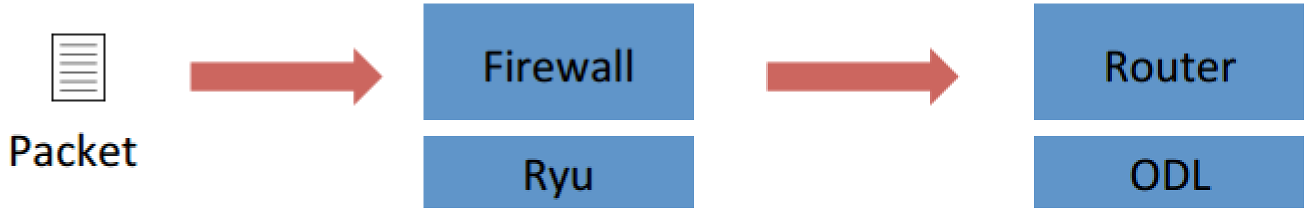
Video

CoVisor [1]

- FlowVisor allows controllers to work on disjoint slices of traffic only

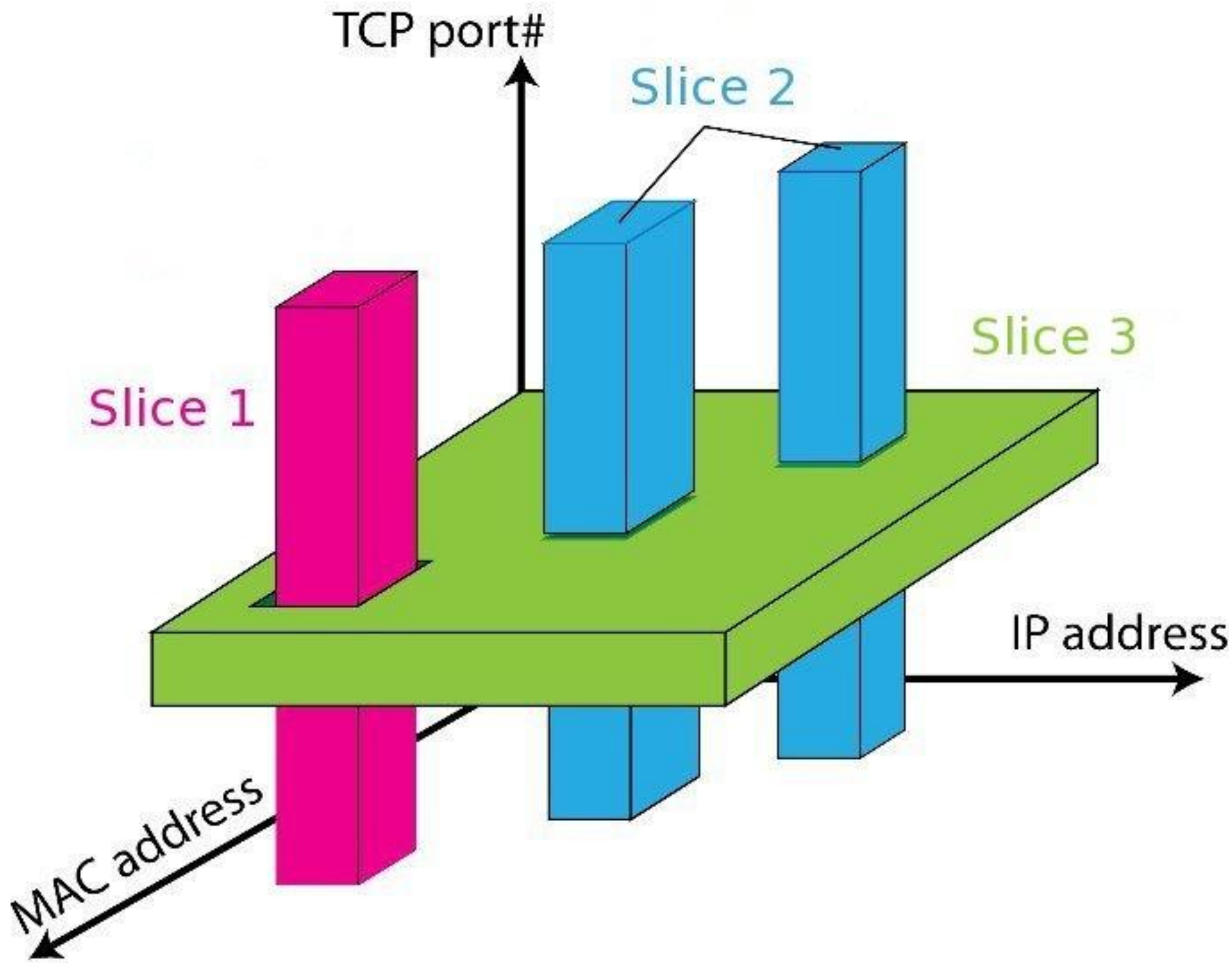


- How about multiple controllers collaborating on the same traffic?



[1] **Jin et al:** "CoVisor: A Compositional Hypervisor for Software-Defined Networks", *USENIX NSDI 2015*
Slides from the presentation at NSDI'15

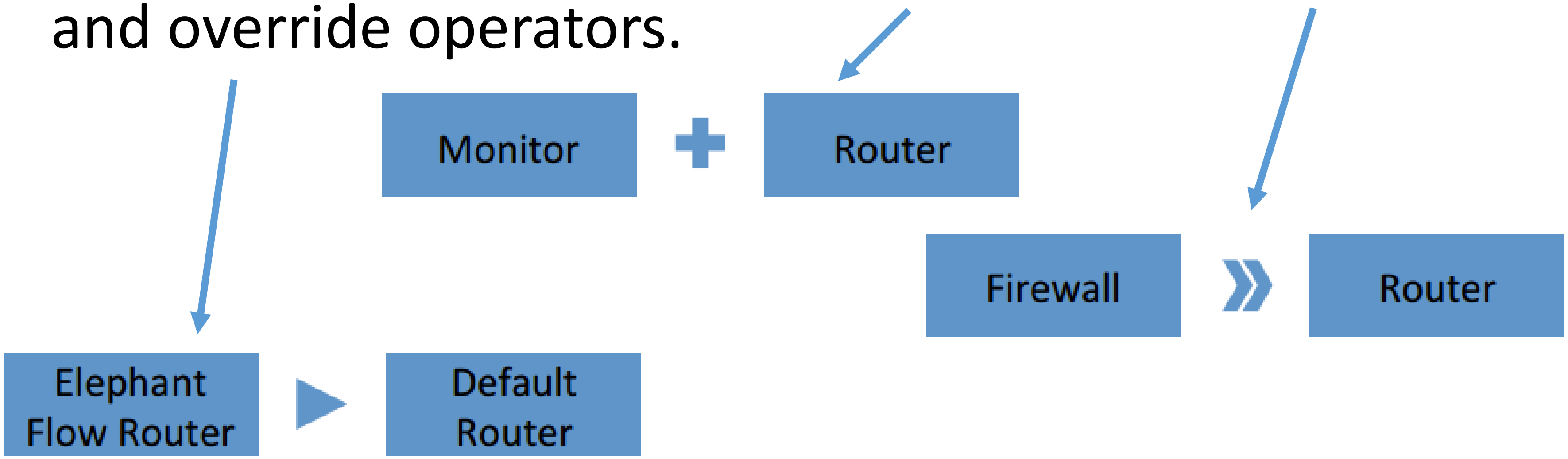
FlowVisor: Disjoint Slices



Sherwood, Rob, et al. "Carving research slices out of your production networks with OpenFlow." *ACM SIGCOMM CCR* 40.1 (2010): 129-130.

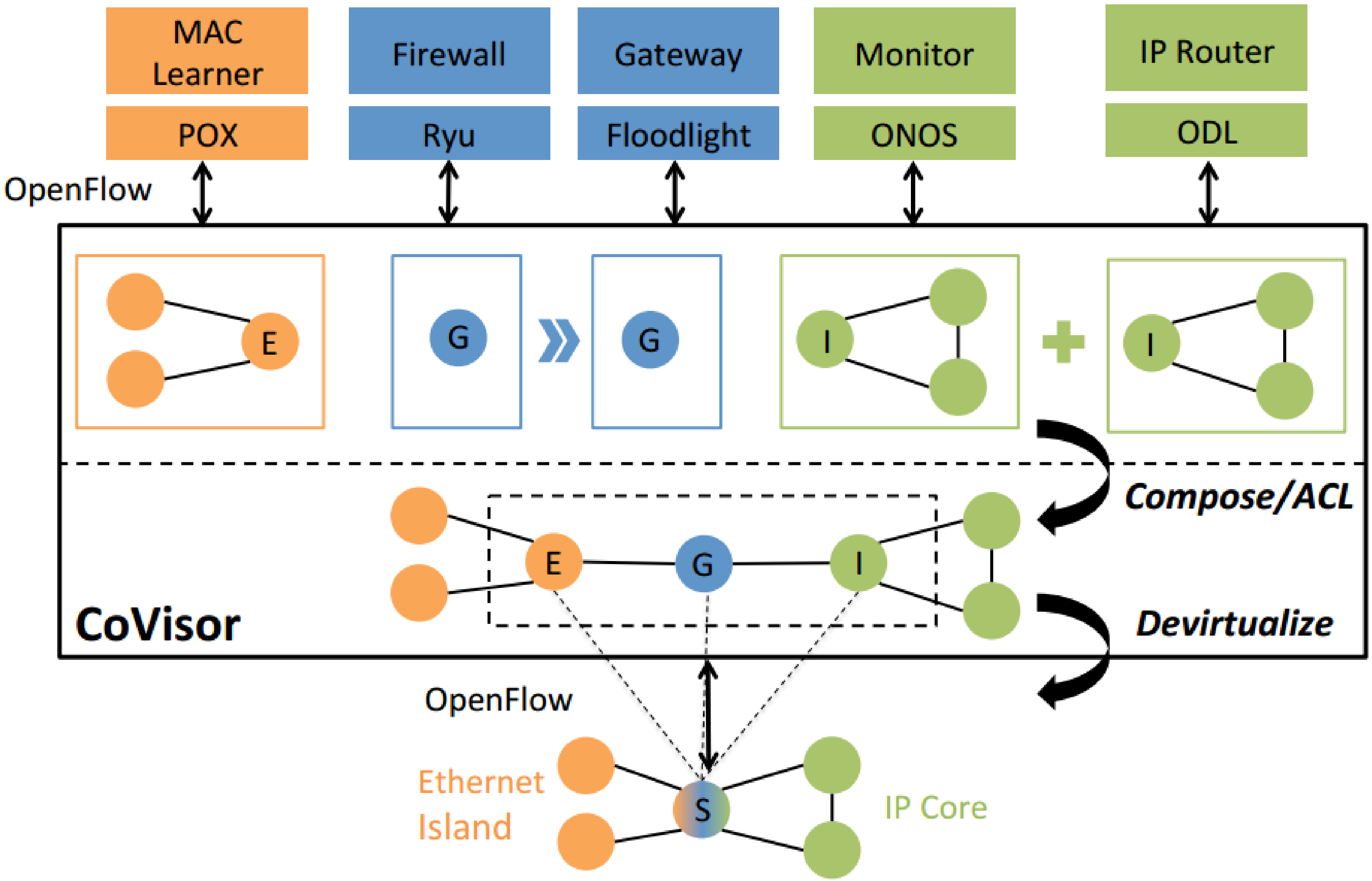
CoVisor – Controller Composition

- CoVisor allows combinations of parallel, sequential and override operators.



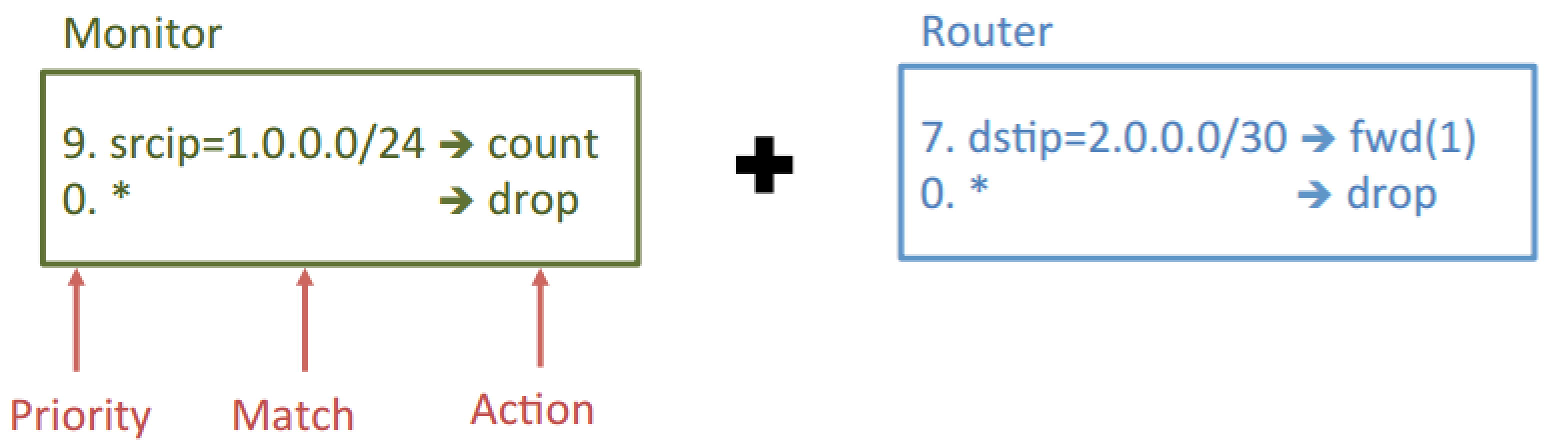
- Combination:

CoVisor – Overview



CoVisor – Policy Composition

- Policy: a list of rules
- Compile policies from controllers to a single policy



CoVisor – Policy Composition

- Policy: a list of rules
- Compile policies from controllers to a single policy

Monitor

9. srcip=1.0.0.0/24 → count
0. * → drop

+

Router

7. dstip=2.0.0.0/30 → fwd(1)
0. * → drop

=

?. srcip=1.0.0.0/24, dstip=2.0.0.0/30 → count, fwd(1)

CoVisor – Policy Composition

Monitor

9. srcip=1.0.0.0/24 → count
0. * → drop

+

Router

7. dstip=2.0.0.0/30 → fwd(1)
0. * → drop

=

?. srcip=1.0.0.0/24, dstip=2.0.0.0/30 → count, fwd(1)
?. srcip=1.0.0.0/24 → count
?. dstip=2.0.0.0/30 → fwd(1)
?. * → drop

CoVisor – Policy Composition

- Controllers continuously update their policies
- Hypervisor recompiles them and update switches

Monitor

9. srcip=1.0.0.0/24 → count
0. * → drop

+

Router

7. dstip=2.0.0.0/30 → fwd(1)
3. dstip=2.0.0.0/26 → fwd(2)
0. * → drop

=

?. srcip=1.0.0.0/24, dstip=2.0.0.0/30 → count, fwd(1)
?. srcip=1.0.0.0/24 → count
?. dstip=2.0.0.0/30 → fwd(1)
?. * → drop

?

CoVisor – Policy Composition

- Computation overhead
  - The computation to recompile the new policy
- Rule-update overhead
  - The rule-updates to update switches to the new policy

Monitor

```
9. srcip=1.0.0.0/24 → count
0. *                → drop
```

\+

Router

```
7. dstip=2.0.0.0/30 → fwd(1)
3. dstip=2.0.0.0/26 → fwd(2)
0. *                → drop
```

=

```
?. srcip=1.0.0.0/24, dstip=2.0.0.0/30 → count, fwd(1)
?. srcip=1.0.0.0/24                   → count
?. dstip=2.0.0.0/30                   → fwd(1)
?. *                                  → drop
```

?

CoVisor – Naïve Policy Composition

- Assign priorities from top to bottom by decrement of 1

Monitor

9. srcip=1.0.0.0/24 → count
0. * → drop

+

Router

7. dstip=2.0.0.0/30 → fwd(1)
0. * → drop

=

3. srcip=1.0.0.0/24, dstip=2.0.0.0/30 → count, fwd(1)
2. srcip=1.0.0.0/24 → count
1. dstip=2.0.0.0/30 → fwd(1)
0. * → drop

CoVisor – Naïve Policy Composition

- Assign priorities from top to bottom by decrement of 1

Monitor

9. srcip=1.0.0.0/24 ➔ count
0. * ➔ drop

+

Router

7. dstip=2.0.0.0/30 ➔ fwd(1)
3. dstip=2.0.0.0/26 ➔ fwd(2)
0. * ➔ drop

=

5. srcip=1.0.0.0/24, dstip=2.0.0.0/30 ➔ count, fwd(1)
4. srcip=1.0.0.0/24, dstip=2.0.0.0/26 ➔ count, fwd(2)
3. srcip=1.0.0.0/24 ➔ count
2. dstip=2.0.0.0/30 ➔ fwd(1)
1. dstip=2.0.0.0/26 ➔ fwd(2)
0. * ➔ drop

CoVisor – Naïve Policy Composition

- Assign priorities from top to bottom by decrement of 1

```
3. srcip=1.0.0.0/24, dstip=2.0.0.0/30 → count, fwd(1)
2. srcip=1.0.0.0/24                   → count
1. dstip=2.0.0.0/30                   → fwd(1)
0. *                                  → drop
```

Update ↓

```
5. srcip=1.0.0.0/24, dstip=2.0.0.0/30 → count, fwd(1)
4. srcip=1.0.0.0/24, dstip=2.0.0.0/26 → count, fwd(2)
3. srcip=1.0.0.0/24                   → count
2. dstip=2.0.0.0/30                   → fwd(1)
1. dstip=2.0.0.0/26                   → fwd(2)
0. *                                  → drop
```

Computation overhead

- Recompute the entire switch table and assign priorities

Rule-update overhead

- Only 2 new rules, but 3 more rules change priority

CoVisor – Incremental Solution

- Add priorities for parallel composition

Monitor

9. srcip=1.0.0.0/24 → count
0. * → drop

\+

Router

7. dstip=2.0.0.0/30 → fwd(1)
0. * → drop

=

9+7 = 16. srcip=1.0.0.0/24, dstip=2.0.0.0/30 → count, fwd(1)

CoVisor – Incremental Solution

- Add priorities for parallel composition

Monitor

9. srcip=1.0.0.0/24 → count
0. * → drop

+

Router

7. dstip=2.0.0.0/30 → fwd(1)
0. * → drop

=

9+7=16. srcip=1.0.0.0/24, dstip=2.0.0.0/30 → count, fwd(1)
9+0=9. srcip=1.0.0.0/24 → count
0+7=7. dstip=2.0.0.0/30 → fwd(1)
0+0=0. * → drop

CoVisor – Incremental Solution

- Add priorities for parallel composition

Monitor

9.	srcip=1.0.0.0/24	→ count
0.	*	→ drop

+

Router

7.	dstip=2.0.0.0/30	→ fwd(1)
3.	dstip=2.0.0.0/26	→ fwd(2)
0.	*	→ drop

=

9+7=16.	srcip=1.0.0.0/24, dstip=2.0.0.0/30	→ count, fwd(1)
9+3=12.	srcip=1.0.0.0/24, dstip=2.0.0.0/26	→ count, fwd(1)
9+0=9.	srcip=1.0.0.0/24	→ count
0+7=7.	dstip=2.0.0.0/30	→ fwd(1)
0+3=3.	dstip=2.0.0.0/26	→ fwd(1)
0+0=0.	*	→ drop

CoVisor – Incremental Solution

- Add priorities for parallel composition

16.	srcip=1.0.0.0/24, dstip=2.0.0.0/30	→ count, fwd(1)
9.	srcip=1.0.0.0/24	→ count
7.	dstip=2.0.0.0/30	→ fwd(1)
0.	*	→ drop

⬇ Update

16.	srcip=1.0.0.0/24, dstip=2.0.0.0/30	→ count, fwd(1)
12.	srcip=1.0.0.0/24, dstip=2.0.0.0/26	→ count, fwd(2)
9.	srcip=1.0.0.0/24	→ count
7.	dstip=2.0.0.0/30	→ fwd(1)
3.	dstip=2.0.0.0/26	→ fwd(2)
0.	*	→ drop

Computation overhead

- Only compose the new rule with rules in monitor

Rule-update overhead

- Add 2 new rules

CoVisor – Incremental Solution

- Add priorities for parallel composition
- Concatenate priorities for sequential composition

Load Balancer

3. srcip=0.0.0.0/2, dstip=3.0.0.0 → dstip=2.0.0.1
1. dstip=3.0.0.0 → dstip=2.0.0.2
0. * → drop

»

Router

1. dstip=2.0.0.1 → fwd(1)
1. dstip=2.0.0.2 → fwd(2)
0. * → drop

=

.

3 >> 1 = 25, srcip=0.0.0.0/2, dstip=3.0.0.0 → dstip=2.0.0.1, **fwd(1)**

011	001
High Bits	Low Bits

.

.

CoVisor – Incremental Solution

- Add priorities for parallel composition
- Concatenate priorities for sequential composition

Load Balancer

3. srcip=0.0.0.0/2, dstip=3.0.0.0 → dstip=2.0.0.1
1. dstip=3.0.0.0 → dstip=2.0.0.2
0. * → drop

»

Router

1. dstip=2.0.0.1 → fwd(1)
1. dstip=2.0.0.2 → fwd(2)
0. * → drop

=

25. srcip=0.0.0.0/2, dstip=3.0.0.0 → dstip=2.0.0.1, fwd(1)
9. dstip=3.0.0.0 → dstip=2.0.0.2, fwd(2)
0. * → drop

CoVisor – Incremental Solution

- Add priorities for parallel composition
- Concatenate priorities for sequential composition
- Stack priorities for override composition

Elephant Flow Router

1. srcip=1.0.0.0, dstip=3.0.0.0 ➔ fwd(3)

▷

Default Router (Max priority = 8)

1. dstip=2.0.0.1 ➔ fwd(1)
1. dstip=2.0.0.2 ➔ fwd(2)
0. * ➔ drop

=

1 + 8 = 9. srcip=1.0.0.0, dstip=3.0.0.0 ➔ fwd(3)
1. dstip=2.0.0.1 ➔ fwd(1)
1. dstip=2.0.0.2 ➔ fwd(2)
0. * ➔ drop

VMWARE NVP [1]

One more example for SDN-based virtualization:

- Scenario: datacenters, support of multiple tenants
  - i.e.: customers pay datacenter provider for computing/network resources

- Tenants need faithful abstractions
  - This includes networking resources
  - But: no access to networking resources (switches, etc.)
  - Thus: tenant not able to interconnect VMs in preferred way

- NVP to solve this dilemma
  - Used by VMWare for three years in production

[1] **Koponen et al:** "Network Virtualization in Multi-tenant Datacenters", *USENIX NSDI 2014*

NVP – Network Hypervisor

Network Hypervisor

- Host hypervisor: offers VM abstraction
  - Network hypervisor: should offer network abstractions
- Network hypervisor positioned between provider physical forwarding infrastructure and tenant control planes
- Offers two abstractions:
  - Control abstraction
  - Packet abstraction

NVP – Abstractions

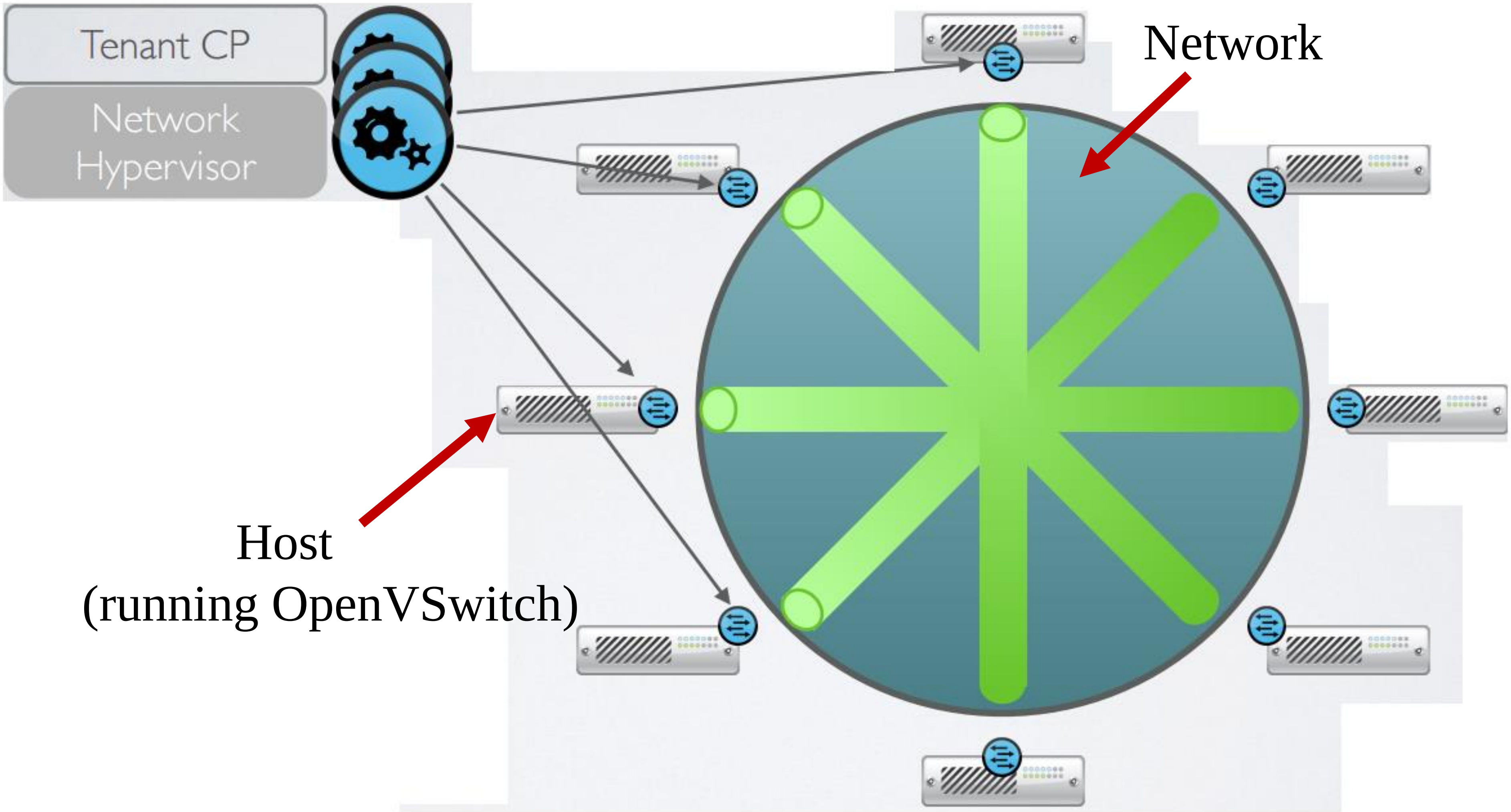
Control Abstraction

- Tenants can *define a set of logical network elements* and *configure them as they would physical network elements*

Packet Abstraction

- Packet sent by endpoints have to be given the same switching, routing, and filtering service as they would have in the tenant's home network

NVP Infrastructure



Taken from **Koponen's talk at** *USENIX NSDI 2014*

NVP

Proprietary…

- Details are obfuscated

…also at protocol level

- Initial version of NVP used OpenFlow for interaction between tenant controller and OpenVSwitch.
- OpenFlow not well suited for this
- Implementation of a proprietary VMWare protocol

Summary

We have discussed:

- FlowVisor *as an example* of using OpenFlow for Network Virtualization
- CoVisor as a more recent approach that can let multiple controllers work on the same traffic
- NVP as an example for a proprietary solution to virtualization in a specific application case