Wireless Network: **JTechs**
Password: **Summer2012**

URL for Slides:

**http://tinyurl.com/JT-Stanford**

# Openflow in a day

InCNTRE

Indiana Center for Network Translational Research and Education

the research arm of

GlobalNOC

Global Research Network Operations Center

# Instructors

Steven Wallace
ssw@iu.edu

Chris Small
chsmall@iu.edu

Brent Sweeny
sweeny@indiana.edu

John Hicks
jhicks@iu.edu

Brent Salisbury
brent@uky.edu

# Tools that we'll be using today...

Open VSwitch - the OpenVSwitch distribution includes an OF controller (i.e., ovs-controller) and a useful command-line utility ovs-ofclt.

FlowVisor - FV supports the virtual "slicing" of OF networks.

WireShark - an open source network "sniffer"

# Forward looking statement...

OpenFlow is still new.

The exercises will be using beta software and firmware (we'll be rebooting things frequently).

This is our fifth OpenFlow Workshop. Based on student input, and changes in the OpenFlow environment, each one has been substantially different. Feedback will inform future workshops!

# Teaching HTML to explain the WWW

**OpenFlow's promise is its application,
not its internal workings**

Yet much of today is about OpenFlow's internal workings, and very little will be polished examples of its application.

# What is required to fully participate

Laptop with Internet connection

ssh client & x-windows server
(window users can use PuTTY)

competency with Ethernet switching

# Logistics

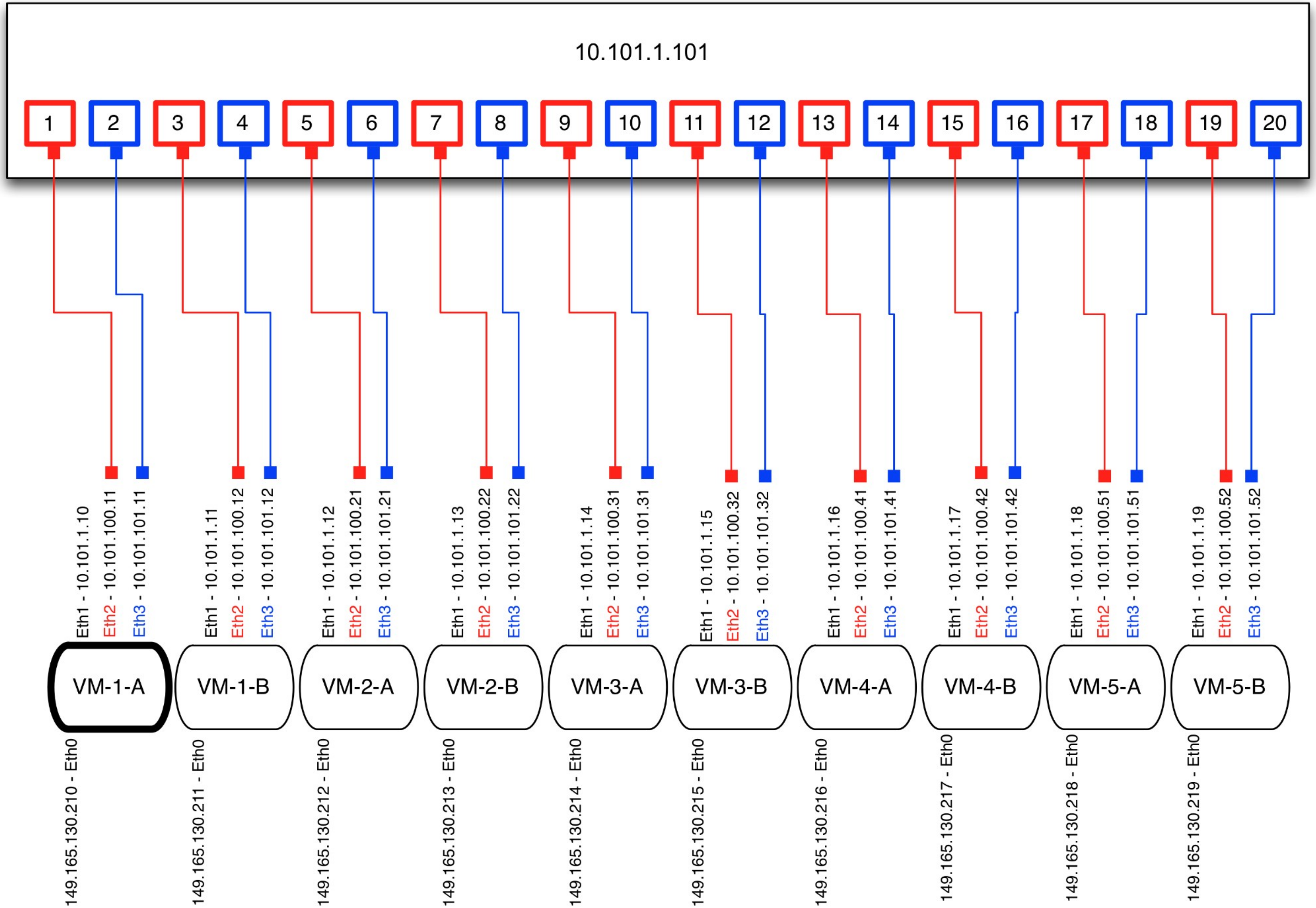
Up to 17 teams of two students each

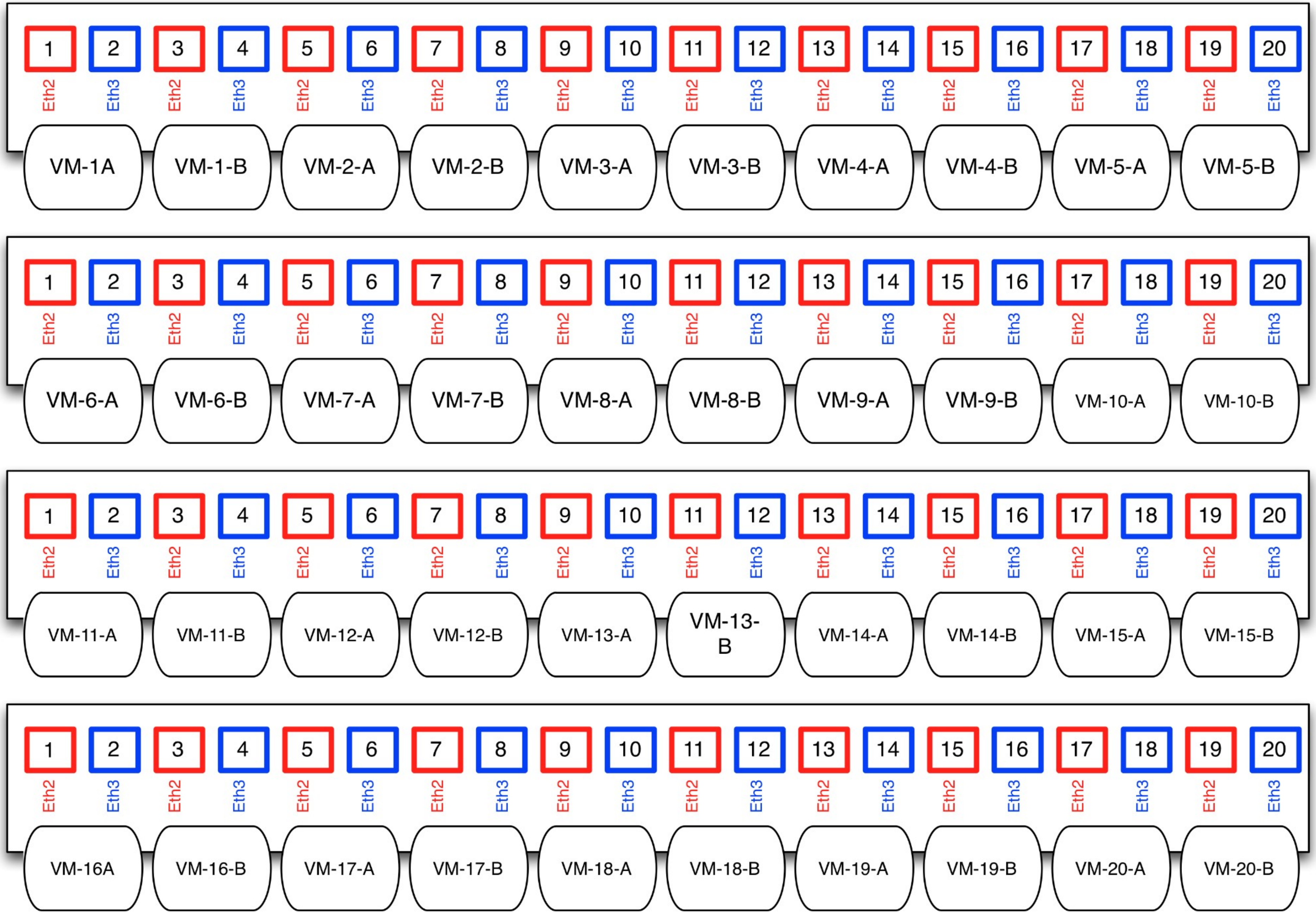
e.g., VM-1-A and VM-1-B are a team.

Each student will login to his/her own VM. Each VM has four interfaces. Eth0 is the interface for remote access.

- ssh -X openflow@[IP Address of your Eth0] (do this twice to open two windows)
- Password "indianajt"

1A


10.101.1.101
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
Eth1 - 10.101.1.10
Eth2 - 10.101.100.11
Eth3 - 10.101.101.11
VM-1-A
149.165.130.210 - Eth0
Eth1 - 10.101.1.11
Eth2 - 10.101.100.12
Eth3 - 10.101.101.12
VM-1-B
149.165.130.211 - Eth0
Eth1 - 10.101.1.12
Eth2 - 10.101.100.21
Eth3 - 10.101.101.21
VM-2-A
149.165.130.212 - Eth0
Eth1 - 10.101.1.13
Eth2 - 10.101.100.22
Eth3 - 10.101.101.22
VM-2-B
149.165.130.213 - Eth0
Eth1 - 10.101.1.14
Eth2 - 10.101.100.31
Eth3 - 10.101.101.31
VM-3-A
149.165.130.214 - Eth0
Eth1 - 10.101.1.15
Eth2 - 10.101.100.32
Eth3 - 10.101.101.32
VM-3-B
149.165.130.215 - Eth0
Eth1 - 10.101.1.16
Eth2 - 10.101.100.41
Eth3 - 10.101.101.41
VM-4-A
149.165.130.216 - Eth0
Eth1 - 10.101.1.17
Eth2 - 10.101.100.42
Eth3 - 10.101.101.42
VM-4-B
149.165.130.217 - Eth0
Eth1 - 10.101.1.18
Eth2 - 10.101.100.51
Eth3 - 10.101.101.51
VM-5-A
149.165.130.218 - Eth0
Eth1 - 10.101.1.19
Eth2 - 10.101.100.52
Eth3 - 10.101.101.52
VM-5-B
149.165.130.219 - Eth0



| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| Eth2 | Eth3 | Eth2 | Eth3 | Eth2 | Eth3 | Eth2 | Eth3 | Eth2 | Eth3 | Eth2 | Eth3 | Eth2 | Eth3 | Eth2 | Eth3 | Eth2 | Eth3 | Eth2 | Eth3 |
| VM-1A | | VM-1-B | | VM-2-A | | VM-2-B | | VM-3-A | | VM-3-B | | VM-4-A | | VM-4-B | | VM-5-A | | VM-5-B | |

| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| Eth2 | Eth3 | Eth2 | Eth3 | Eth2 | Eth3 | Eth2 | Eth3 | Eth2 | Eth3 | Eth2 | Eth3 | Eth2 | Eth3 | Eth2 | Eth3 | Eth2 | Eth3 | Eth2 | Eth3 |
| VM-6-A | | VM-6-B | | VM-7-A | | VM-7-B | | VM-8-A | | VM-8-B | | VM-9-A | | VM-9-B | | VM-10-A | | VM-10-B | |

| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| Eth2 | Eth3 | Eth2 | Eth3 | Eth2 | Eth3 | Eth2 | Eth3 | Eth2 | Eth3 | Eth2 | Eth3 | Eth2 | Eth3 | Eth2 | Eth3 | Eth2 | Eth3 | Eth2 | Eth3 |
| VM-11-A | | VM-11-B | | VM-12-A | | VM-12-B | | VM-13-A | | VM-13-B | | VM-14-A | | VM-14-B | | VM-15-A | | VM-15-B | |

| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| Eth2 | Eth3 | Eth2 | Eth3 | Eth2 | Eth3 | Eth2 | Eth3 | Eth2 | Eth3 | Eth2 | Eth3 | Eth2 | Eth3 | Eth2 | Eth3 | Eth2 | Eth3 | Eth2 | Eth3 |
| VM-16A | | VM-16-B | | VM-17-A | | VM-17-B | | VM-18-A | | VM-18-B | | VM-19-A | | VM-19-B | | VM-20-A | | VM-20-B | |

# Logistics

Go ahead and ssh to your VM.

Once you're connected, make sure your x server is running first, then run wireshark:

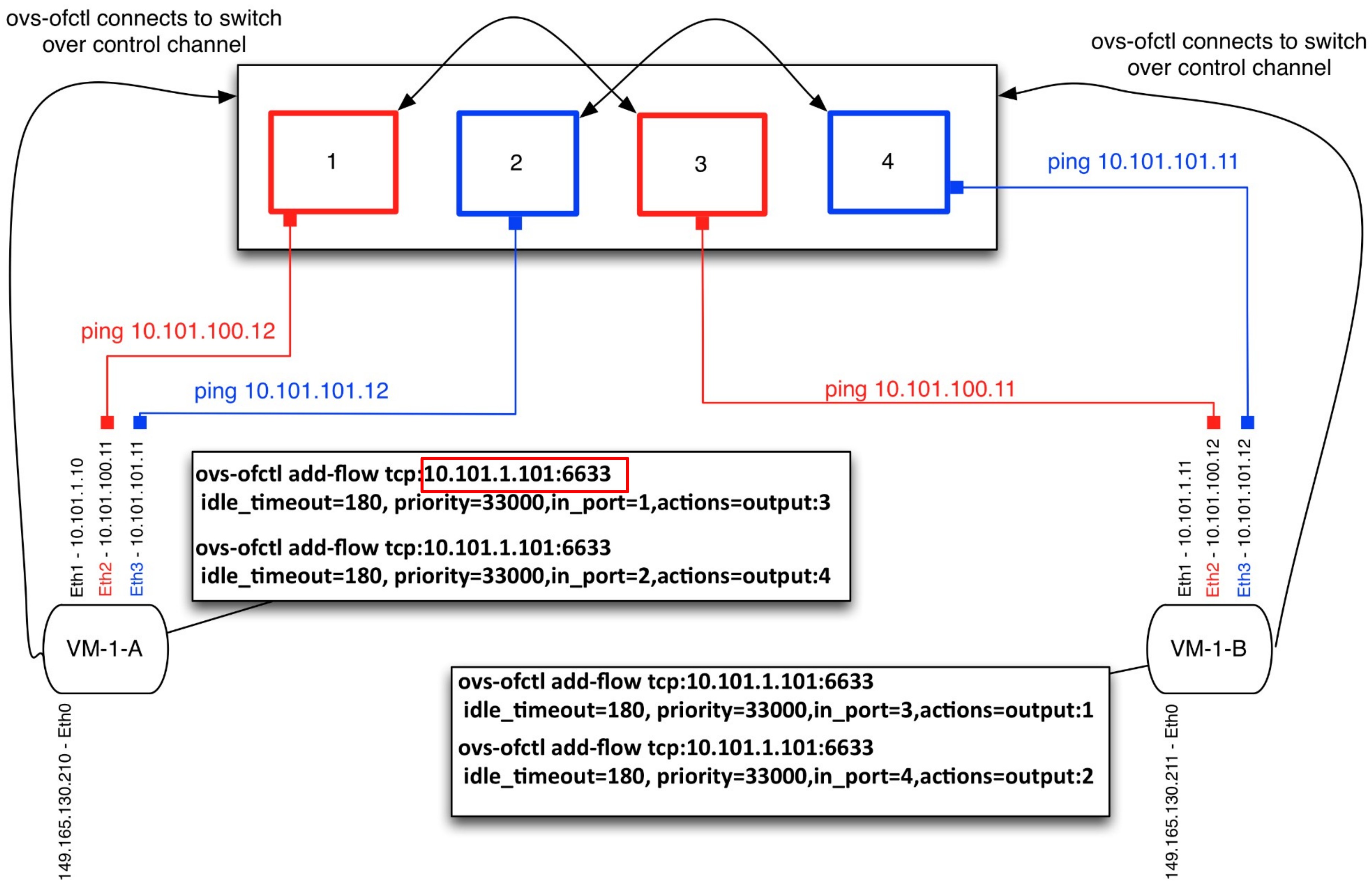
sudo wireshark &

Select ok in popup box to run Wireshark as root

Configure wireshark:

1. sniff on Eth1
2. filter type "of"

# Using OpenFlow Port-based Rules To Interconnect Ports via ovs-ofctl



```
ovs-ofctl add-flow tcp:10.101.1.101:6633
 idle_timeout=180, priority=33000,in_port=1,actions=output:3

ovs-ofctl add-flow tcp:10.101.1.101:6633
 idle_timeout=180, priority=33000,in_port=2,actions=output:4
```

```
ovs-ofctl add-flow tcp:10.101.1.101:6633
 idle_timeout=180, priority=33000,in_port=3,actions=output:1

ovs-ofctl add-flow tcp:10.101.1.101:6633
 idle_timeout=180, priority=33000,in_port=4,actions=output:2
```

# Logistics

A bit more about the VM configuration:

Eth0 - remote access (place you ssh to)

Eth1 - OpenFlow channel

Eth2 & Eth3 are connected to ports on the OpenFlow switch. These will be used to test the effect of putting OpenFlow rules in the switch.

# Rough Agenda

Logistics (the part we just did)

OpenFlow's value

OpenFlow's origin

OpenFlow's owner

OpenFlow's oxygen

OpenFlow (the details)

Hands-on with OpenFlow

Whiteboard exercises, if OF was your only tool, how would you solve these problems: Bonjour printing, moving magic packets, and preventing dsniff man-in-the-middle.

# OpenFlow's Value

Enterprise

Data Center

WAN

# What can OpenFlow bring to the enterprise

- Automated configuration of new equipment in your enterprise network (think controller-based wireless)
- Choose from a marketplace of solutions for common network requirements (e.g., PCI-DSS compliance, NAC network access control, etc.)
- Delegate control of network slices to their proper steward (e.g., CCTV, door locks, etc.)
- Address new requirements (e.g., bonjour printing, guest access, BYOD) through new software, not new equipment

# What can OpenFlow bring to the data center

- Standard API for network provisioning (i.e. orchestration)
- Integration with VM-based switches (e.g. Open vSwitch)
- New network behaviors that permit scaling to million-VM data centers
- Potential for ODMs to provide more cost effective solutions

# What can OpenFlow bring to the wide area network

- Standard API for network provisioning of bandwidth-on-demand services (e.g. Internet2 OS3E)
- Standard API upon which to address new requirements (e.g. lawful intercept)
- Delegate control of network slices upon which arbitrary virtual networks can coexist on a common network platform

# OpenFlow Origin

Clean Slate Program at Stanford

- Early work on SANE circa 2006 (security architecture)
- inspired Ethane circa 2007, which lead to OpenFlow

2009 Stanford publishes OF 1.0.0 spec

2009 Nicira Series A funding

2010 Big Switch seed funding

2011 Open Network Foundation is created

2012 Google announces migration to OF (migration started in 2009)

# OpenFlow's Owner:

## Open Networking Foundation

## ONF members:

A10 Networks, Alcatel-Lucent, Argela, Big Switch Networks, Broadcom, Brocade, Ciena, Cisco, Citrix, Colt, Comcast, CompTIA, Cyan, Dell, Deutsche Telekom, Elbrys, Ericsson, ETRI, Extreme Networks, EZchip, F5, Facebook, Force10 Networks, France Telecom Orange, Fujitsu, Gigamon, Goldman Sachs, Google, Hitachi, HP, Huawei, IBM, Infinera, Infoblox, Intel, IP Infusion, Ixia, Juniper Networks, Korea Telecom, LineRate Systems, LSI, Luxoft, Marvell, Mellanox, Metaswitch Networks, Microsoft, Midokura, NCL Communications K.K., NEC, Netgear, Netronome, Nicira Networks, Nokia Siemens Networks, NTT Communications, Oracle, PICA8, Plexxi Inc., Radware, Riverbed Technology, Samsung, SK Telecom, Spirent, Telecom Italia, Tencent, Texas Instruments, Vello Systems, Verizon, VMware, Yahoo, ZTE Corporation --- Board Members

# Open Networking Foundation

Membership-based 30K a year.

Members agree to share IP on reasonable terms.

Working group evolve the standard.

Not like IETF, ITU, IEEE, etc.

# OpenFlow's Oxygen

## (hype is adrenaline, not oxygen)

Large data center operators can roll their own. They make their own servers, their own data center designs, and their own software. Offer them a standard protocol that provides fine-grain control of COTS network hardware, they will supply lots of oxygen. Examples include:

if “Floor Plan Entropy” has got your bisection bandwidth down, build fat tree networks based on low-cost switches by programming the network for the data center via Openflow (e.g., PortLand)

if network provisioning is slow and manual, leverage an open network API to create better orchestration

# Reducing the oxygen requirement

Merchant Silicon: “off the shelf” chips that perform packet processing at high speed vs. vertically integrated custom designed chips designed & built by switch vendors.

Q: What do the following have in common:

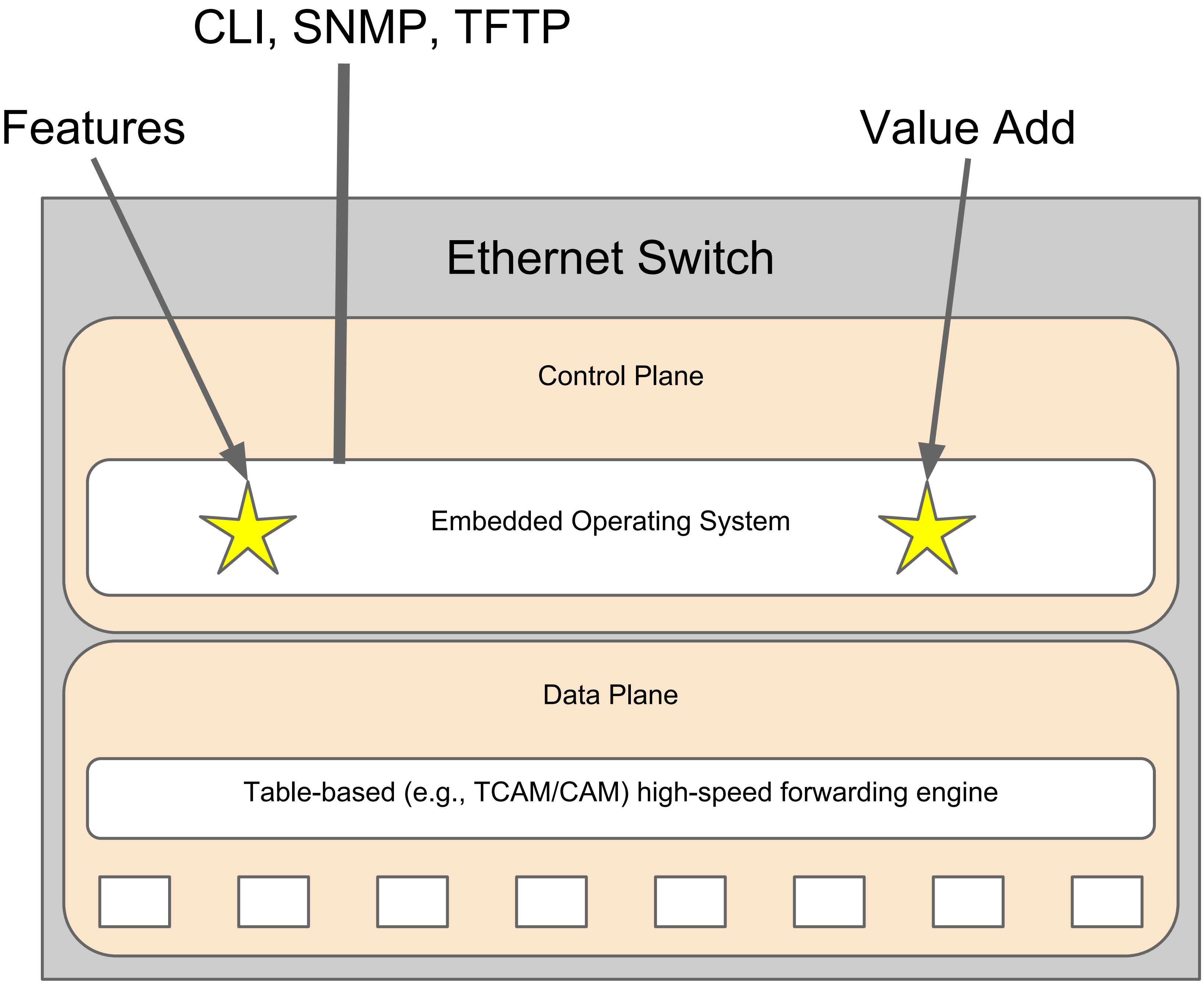
Juniper QFX3500, IBM BNT RackSwitch G8264, Alcatel-Lucent OminiSwitch 6900, Cisco Nexus 3064, HP 5900AF 48XG, Dell Force 10 S4810, and Arista 7050S-64?

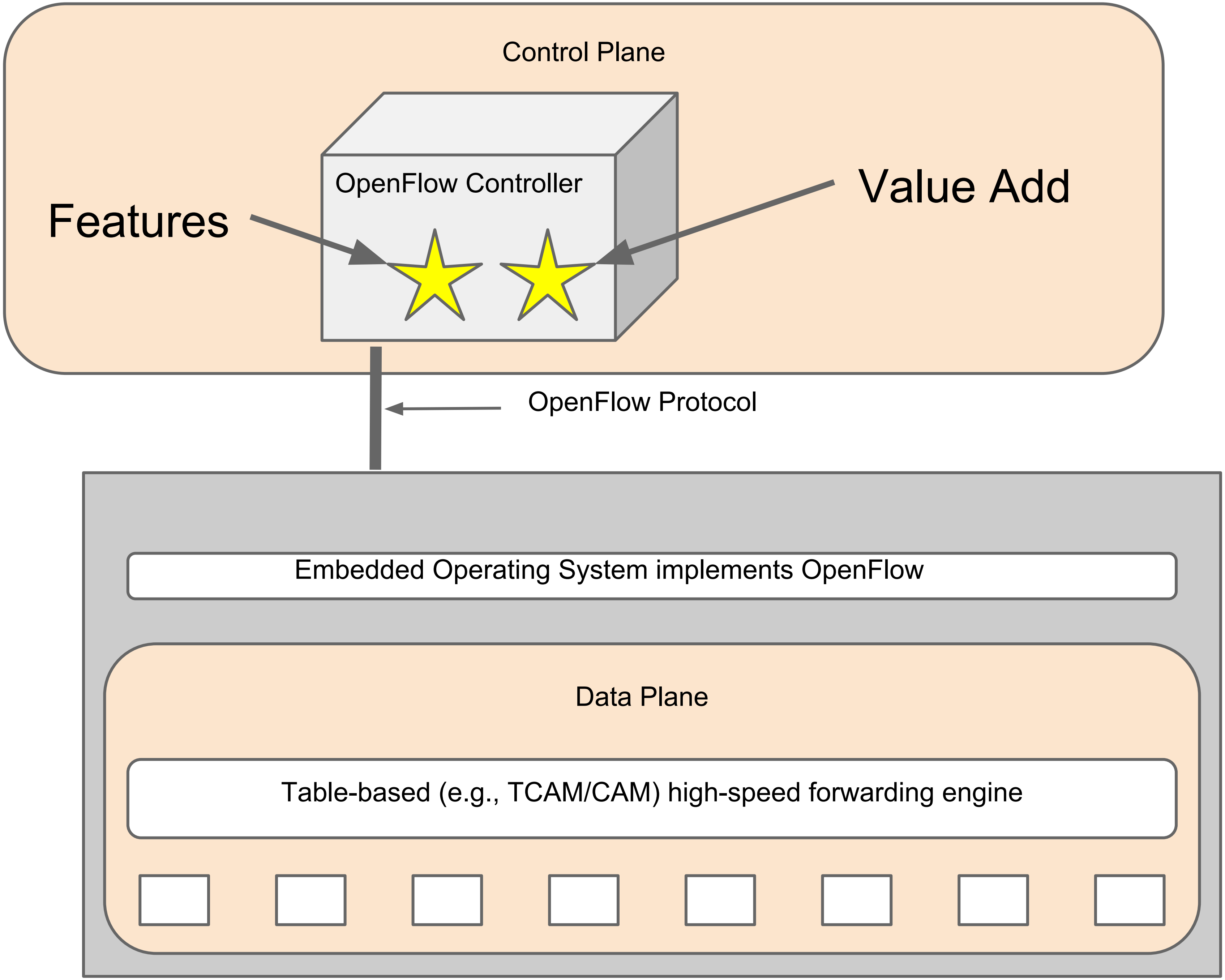
A: Broadcom silicon.

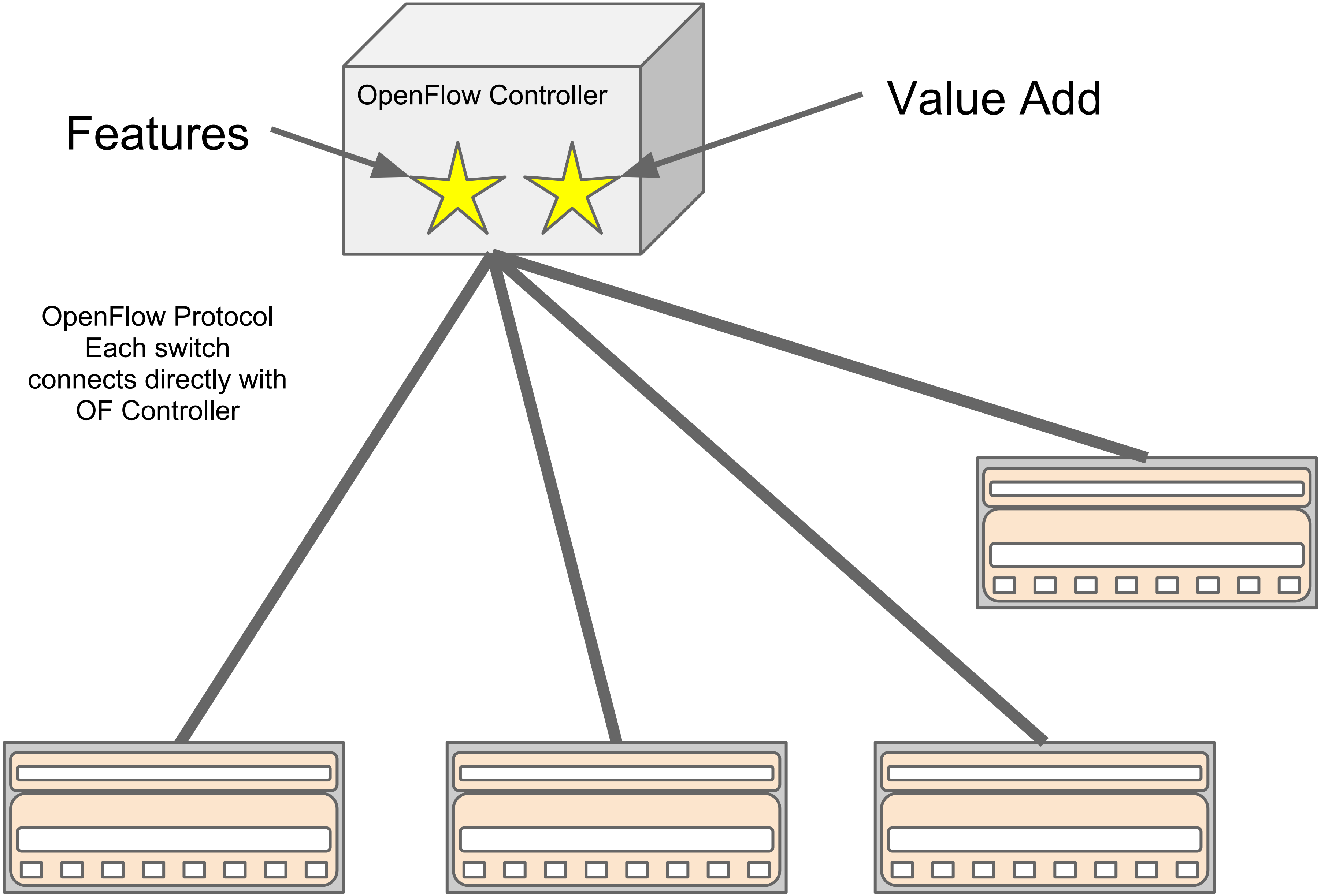
**ODMs** (Original Design manufacturer) have their own design, typically based on merchant silicon.

# What is OpenFlow?

- It's a protocol for control the forwarding behavior of Ethernet switches in a Software Defined Network
- Initially released by the Clean Slate Program at Stanford, its specification is now maintained by the Open Networking Forum
- Most of today's material is based on the OpenFlow 1.0 specification
- As of now, the specification has advanced to OpenFlow 1.3


CLI, SNMP, TFTP
Features
Value Add
Ethernet Switch
Control Plane
Embedded Operating System
Data Plane
Table-based (e.g., TCAM/CAM) high-speed forwarding engine


Control Plane
OpenFlow Controller
Features
Value Add
OpenFlow Protocol
Embedded Operating System implements OpenFlow
Data Plane
Table-based (e.g., TCAM/CAM) high-speed forwarding engine



Features

OpenFlow Controller

Value Add

OpenFlow Protocol
Each switch
connects directly with
OF Controller

# Flow Table

| Header Fields | Counters | Actions | Priority |
|---|---|---|---|
| Ingress Port | **Per Flow Counters** | Forward | |
| Ethernet Source Addr | Received Packets | (All, Controller, Local, | |
| Ethernet Dest Addr | Received Bytes | Table, IN_port, Port#, | |
| Ethernet Type | Duration seconds | Normal, Flood) | |
| VLAN id | Duration nanosecconds | | |
| VLAN Priority | | Enqueue | |
| IP Source Addr | | Drop | |
| IP Dest Addr | | Modify-Field | |
| IP Protocol | | | |
| IP ToS | | | |
| ICMP type | | | |
| ICMP code | | | |

Required Per Spec

# Flow Table

| Header Fields | Counters | Actions | Priority |
|---|---|---|---|
| If ingress port == 2 | | Drop packet | 32768 |
| if IP_addr == 129.79.1.1 | | re-write to 10.0.1.1, forward port 3 | 32768 |
| if Eth Addr == 00:45:23 | | add VLAN id 110, forward port 2 | 32768 |
| if ingress port == 4 | | forward port 5, 6 | 32768 |
| if Eth Type == ARP | | forward CONTROLLER | 32768 |
| If ingress port == 2 && Eth Type == ARP | | forward NORMAL | 40000 |

# Special Ports

Controller (sends packet to the controller)

Normal (sends packet to non-openflow function of switch)

Local (can be used for in-band controller connection)

Flood (flood the packet using normal pipeline)

# Flow Table

| Header Fields | Counters | Actions | Priority |
|---|---|---|---|
| If ingress port == 2 |  | Drop packet | 32768 |
| if IP_addr == 129.79.1.1 |  | re-write to 10.0.1.1, forward port 3 | 32768 |

Each Flow Table entry has two timers: **idle_timeout**

seconds of no matching packets after which the flow is removed

**hard_timeout**

seconds after which the flow is removed

If both **idle_timeout** and **hard_timeout** are set, then the flow is removed when the first of the two expires.

# Populating the Flow Table

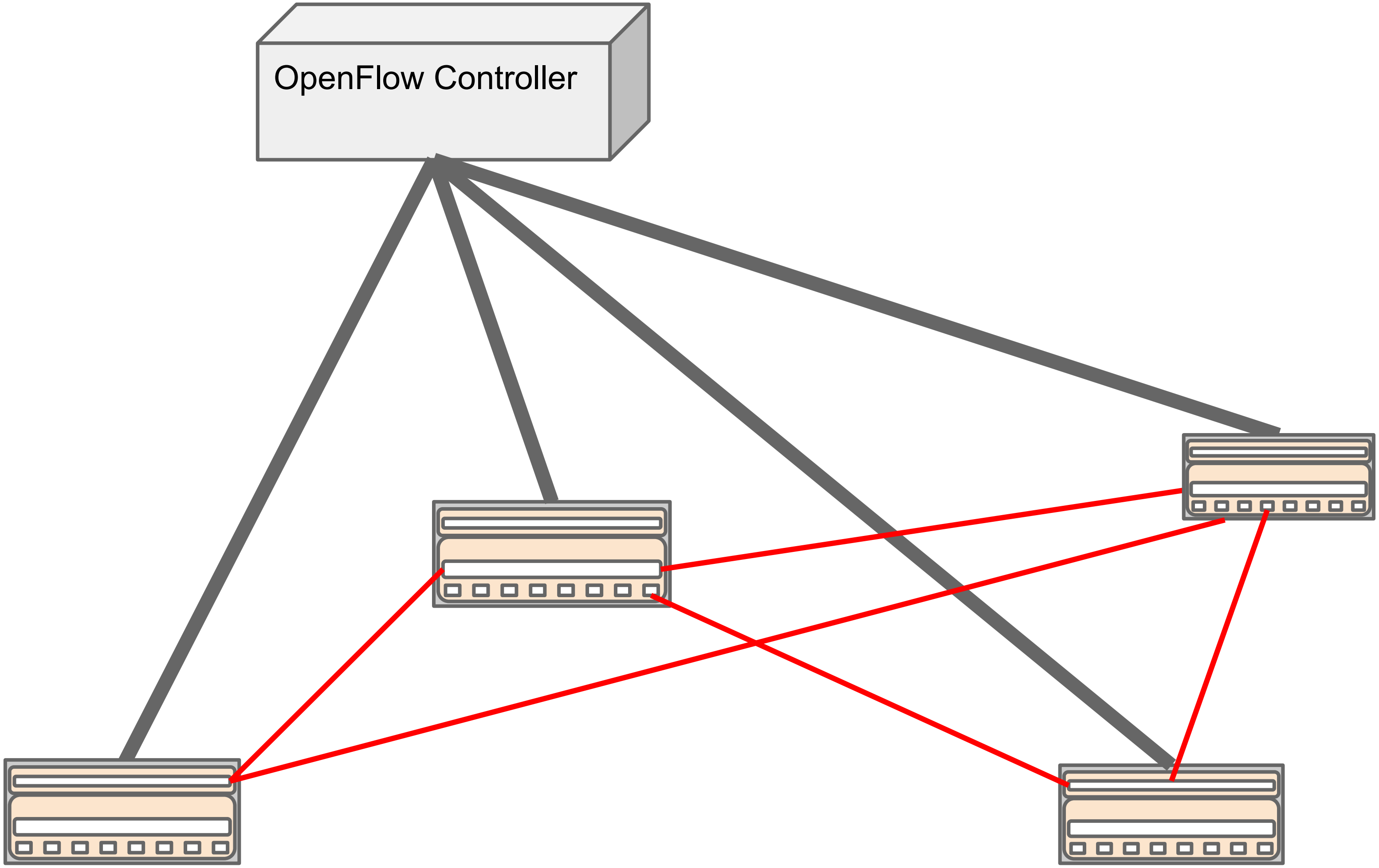
Proactive

Rules are relatively static, controller places rules in switch before they are required.

Reactive

Rules are dynamic. Packets which have no match are sent to the controller (packet in). Controller creates appropriate rule and sends packet back to switch (packet out) for processing.

# Example application: topology discovery



# Bootstrapping a new switch

Switch requires minimal initial configuration (e.g., IP address, default GW, and OpenFlow controller)

Switch connects to controller. Controller requests things like a list of ports, etc.

Controller proceeds to determine the switch's location.

# Bootstrapping a new switch

Controller *proactively* places a rule in the switch.

If ether_type = LLDP, actions=output:controller

Then the controller creates an LLDP packet, sends it to the switch, and instructs the switch to send it out a port (repeat for all ports).

Since all switches in the controller's network have a rule to send LLDP packets to the controller, the controller is able to determine the topology.

# OpenFlow 1.0 to 1.1

# Flow Table

1.0

| | | | |
|---|---|---|---|
| Header Fields | Counters | Actions | Priority |

1.1

| | | | | |
|---|---|---|---|---|
| Match Fields | Priority | Counters | Instructions | Cookie |

·····

New Data Structure in Pipeline

| | | |
|---|---|---|
| media data | packet | Action Set |

| | | | |
|---|---|---|---|
| Group ID | Type | Counters | Action Buckets |

·····

# Packet Processing

1.0

Does packet match flow table entry, if so, perform action.

1.1

Does packet match flow table entry, if so, look at instructions...

# Actions vs. Instructions

1.1

- Flow entries contain instructions.
- Instructions <u>may</u> be immediate action(s), or
- instructions <u>may</u> set actions in the action set
- Instructions can also change pipeline processing:
    - Goto table X
    - Goto group table entry x

# More Tables

1.1

- Allows for multiple flowtables
- Includes a group table with multiple group table types
- Instructions can jump to other tables, but only in a positive direction

# OpenFlow QoS

OF 1.0

- Optional action "Enqueue"
  Forwards packet through a queue attached to a port. The behavior of the queue is determined outside the scope of OF.
- Header fields can include VLAN priority and IP ToS, so they can be matched against and re-written.

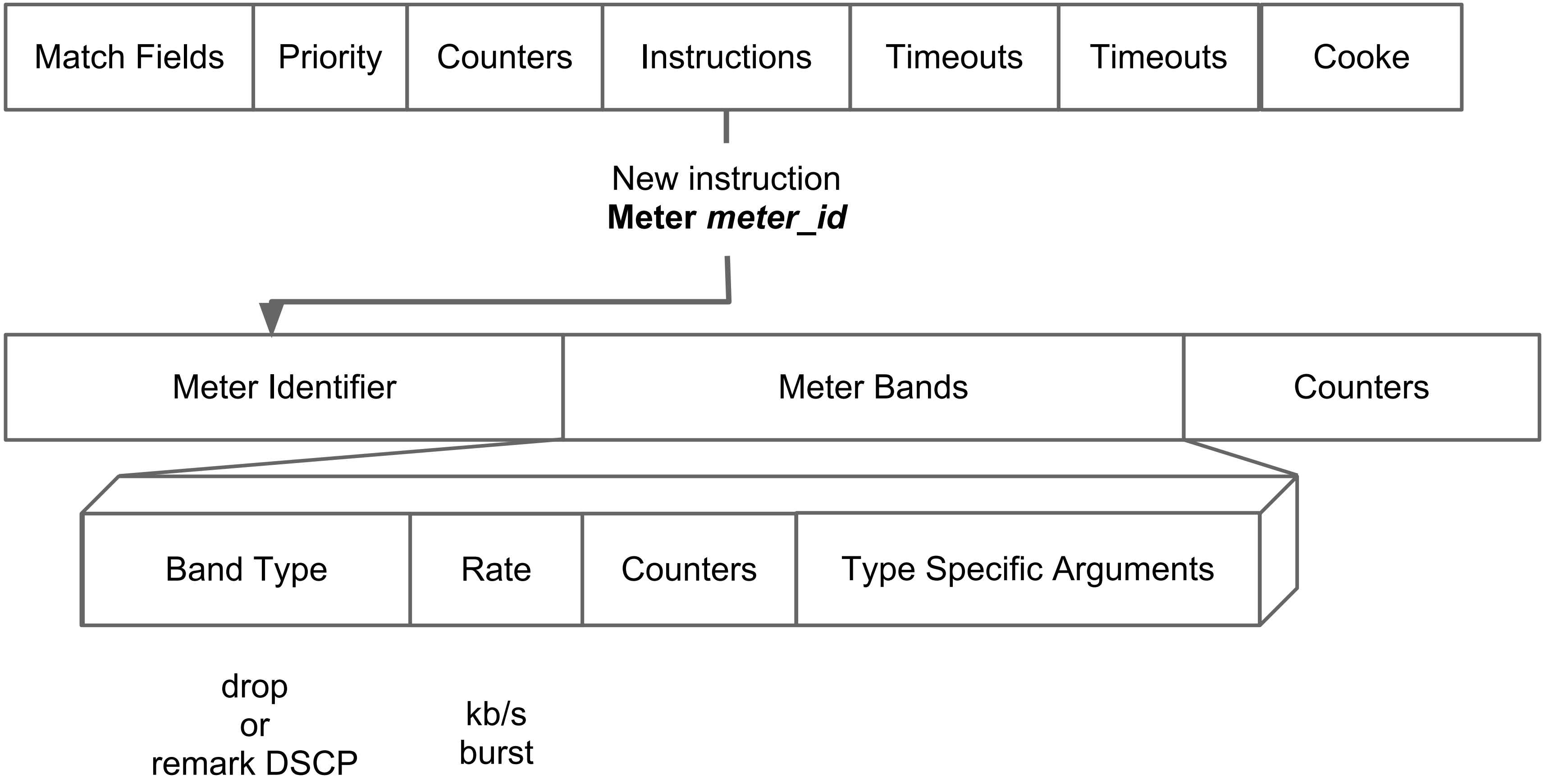
# OpenFlow QoS

OF 1.3

- Stuff from 1.0
- New table "Meter Table"

| Meter Identifier | Meter Bands | Counters |
|---|---|---|
| 32 bit integer<br>used to identify the meter | list of meter bands<br>each band specifies rate and behavior | |

# OpenFlow QoS (1.3 cont.)



Match Fields | Priority | Counters | Instructions | Timeouts | Timeouts | Cooke

New instruction
**Meter *meter_id***

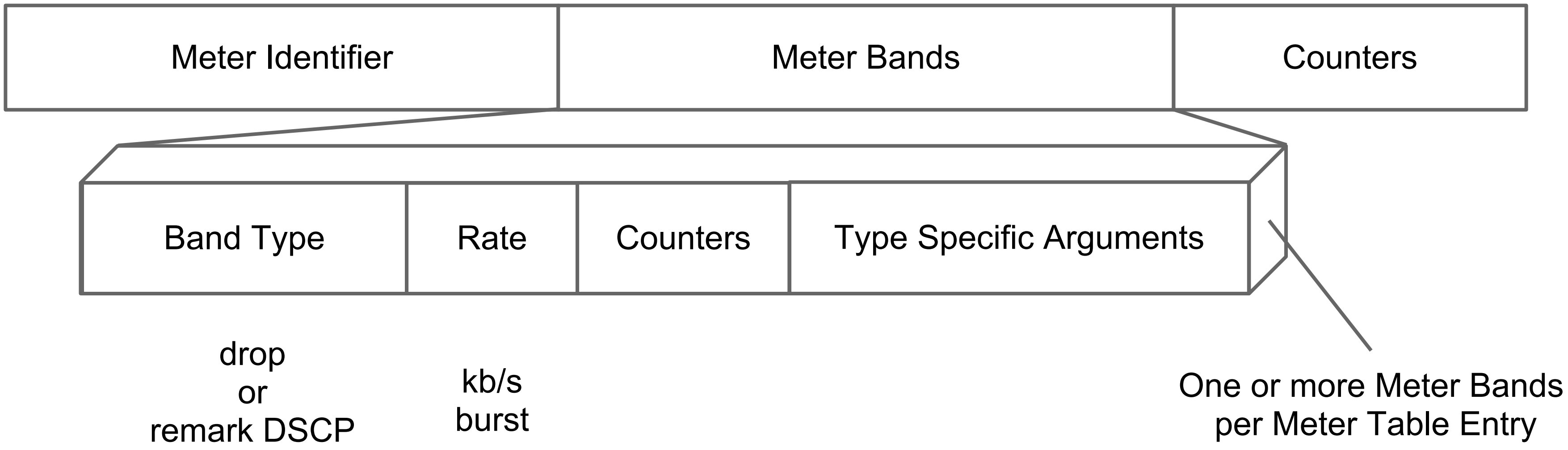
Meter Identifier | Meter Bands | Counters

Band Type | Rate | Counters | Type Specific Arguments

drop
or
remark DSCP

kb/s
burst

# OpenFlow QoS (1.3 cont.)



Meter Identifier

Meter Bands

Counters

Band Type

Rate

Counters

Type Specific Arguments

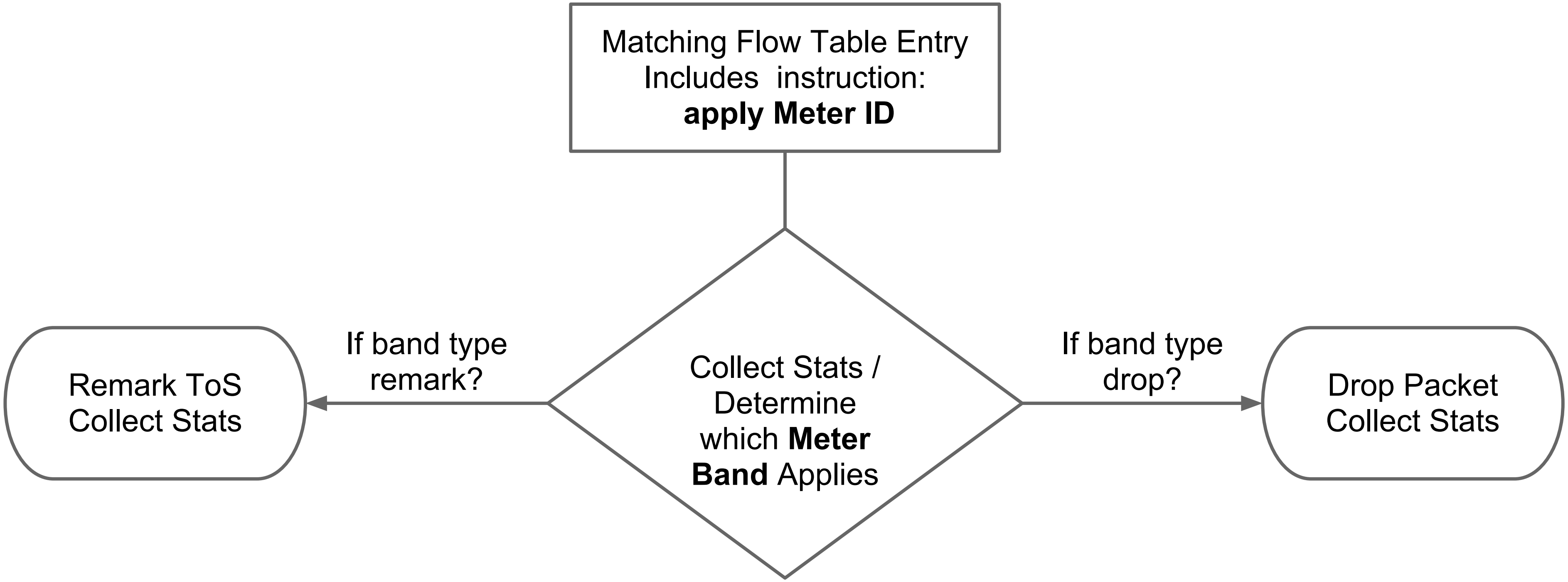
drop
or
remark DSCP

kb/s
burst

One or more Meter Bands
per Meter Table Entry

*"the meter applies the meter band with the highest configured rate that is lower than the current measured rate"*

# OpenFlow QoS (1.3 cont.)



Matching Flow Table Entry Includes instruction: **apply Meter ID**

Collect Stats / Determine which **Meter Band** Applies

If band type remark?

Remark ToS Collect Stats

If band type drop?

Drop Packet Collect Stats

# Hands-on with OpenFlow

## (quick review of the table)

| Header Fields | Counters | Actions | Priority |
|---|---|---|---|
| Ingress Port<br>Ethernet Source Addr<br>Ethernet Dest Addr<br>Ethernet Type<br>VLAN id<br>VLAN Priority<br>IP Source Addr<br>IP Dest Addr<br>IP Protocol<br>IP ToS<br>ICMP type<br>ICMP code | **Per Flow Counters**<br>Received Packets<br>Received Bytes<br>Duration seconds<br>Duration nanosecconds | Forward<br>(All, Controller, Local, Table, IN_port, Port# Normal, Flood)<br>Enqueue<br>Drop<br>Modify-Field | |

# Hands-on with OpenFlow
## (switch config)

- HP switches run in hybrid Openflow mode
    - can act as a regular switch or as an openflow switch
    - implemented on a per VLAN basis or aggregation mode
    - capable of running multiple openflow instances
    - openflow capabilities:

# HP Switch Configuration

- Enter configuration mode
  - # configure
- Create a VLAN for your Openflow instance
  - # vlan 10
- Add ports to the VLAN
  - In our case we have untagged traffic coming in on ports 1-20
    - untagged 1-20
  - Port 21 is used for management, 23-24 interconnects
    - # tagged 21
    - # tagged 23-24

# HP Switch Configuration

- Currently we have a VLAN with the ports we need. The configuration looks like this:

```
vlan 10
name "VLAN10"
untagged 1-20
tagged 21,23-24
no ip address
exit
```

# HP Switch Configuration

- Now to enable Openflow on the VLAN
  - # openflow vlan 10 enable
- Tell the Openflow instance to actively connect to an Openflow controller
  - # openflow controller tcp:10.101.1.39:6633
  - 6633 is the port that is listening on the controller
- If the switch can't connect to the controller, we want the switch to forward using current rules
- # openflow fail-secure on

# HP Switch Configuration

- Lastly, we want the ability to manually connect to the switch to check and set state
    - the openflow instance on the vlan will be listening on port 6633 for dpctl ovs-ofctl commands
        - # openflow listener ptcp:6633
    - Limit the listener to a specific IP address
        - # openflow listener ptcp:10.101.1.210:6633

(to see status of listener port and state for vlan 10: "show openflow 10")

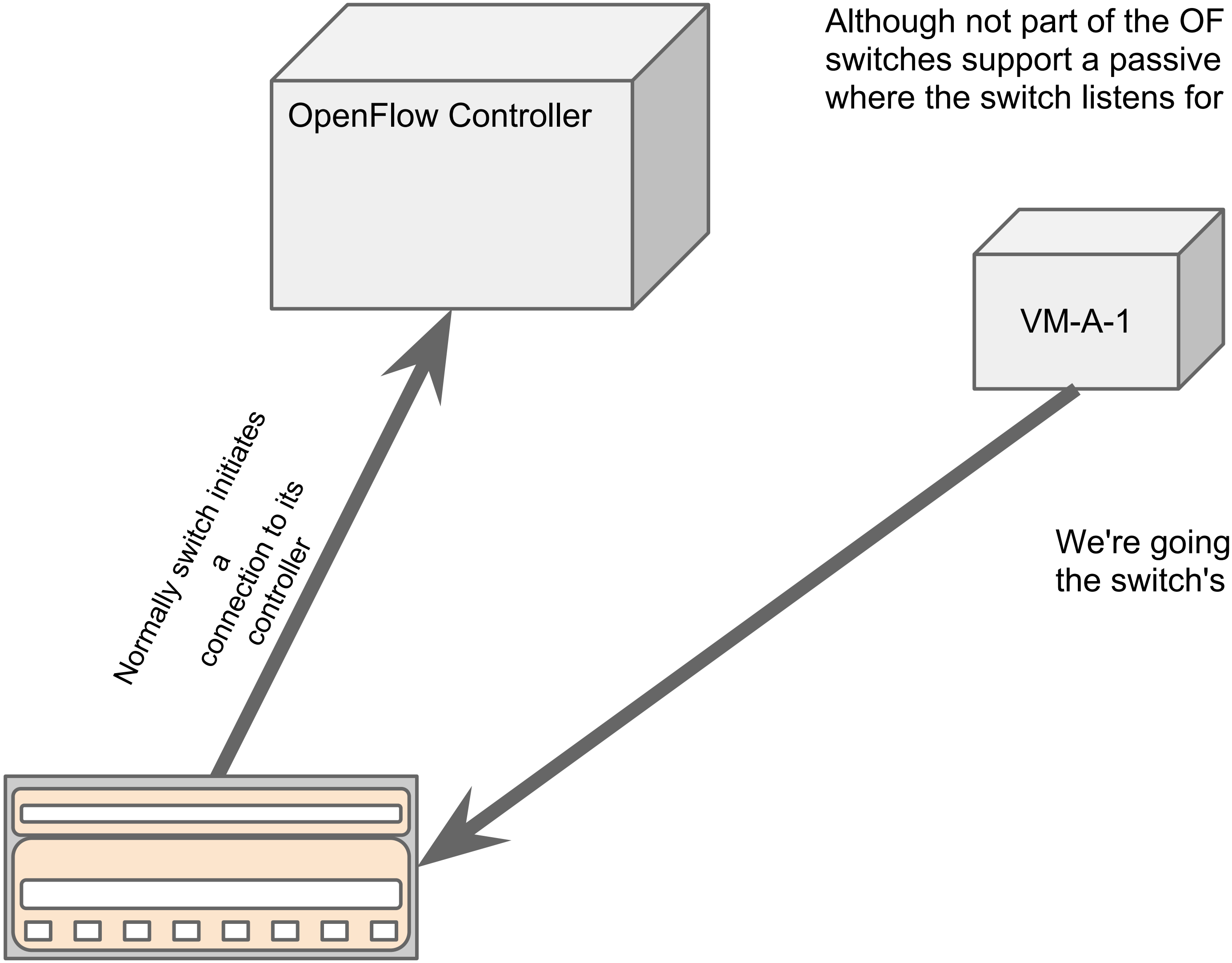
# Actual Switch Configuration

```
Running configuration:

; J9470A Configuration Editor; Created on release #K.15.06.5008
; Ver #02:10.0d:1f

hostname "sw-1"
time timezone -300
time daylight-time-rule Continental-US-and-Canada
module 1 type J94ddA
vlan 1
   name "DEFAULT_VLAN"
   untagged 22
   no untagged 1-21,23-24
   no ip address
   exit
vlan 2
   name "VLAN2"
   untagged 21
   ip address 10.101.1.101 255.255.255.0
   exit
vlan 10
   name "VLAN10"
   untagged 1-20
   tagged 21,23-24
   no ip address
   exit
openflow
   vlan 10
      enable
      controller "tcp:10.101.1.50:6633" listener "ptcp:6633" fail-secure on
      exit
   exit
snmp-server community "public" unrestricted
```

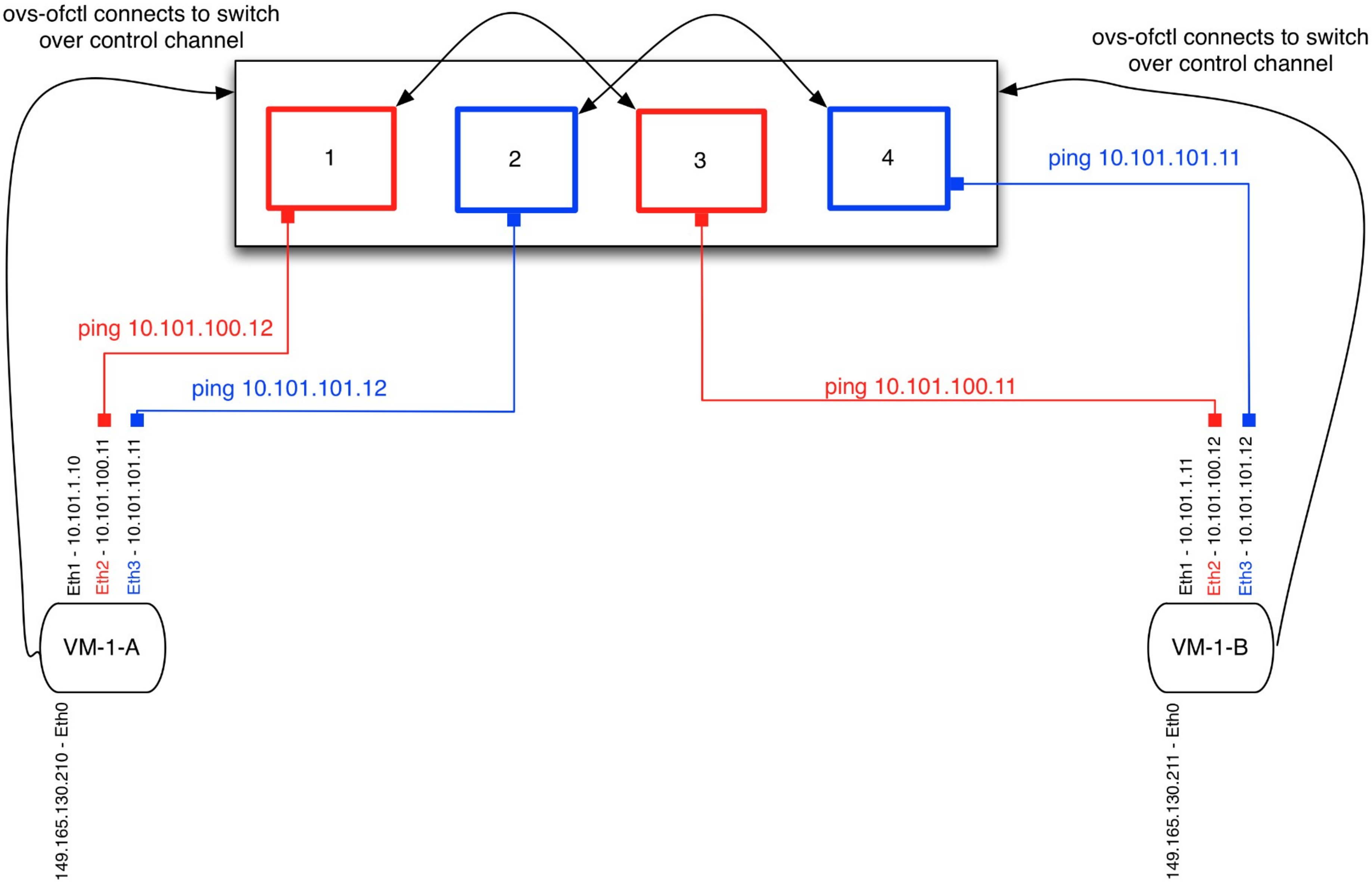
# Hands-on with OpenFlow



Although not part of the OF spec, many switches support a passive OF connection, where the switch listens for a connection.

We're going to use ovs-ofctl to query the switch's status.

# Using OpenFlow Port-based Rules To Interconnect Ports via ovs-ofctl



ovs-ofctl connects to switch over control channel

ovs-ofctl connects to switch over control channel

1

2

3

4

ping 10.101.101.11

ping 10.101.100.12

ping 10.101.101.12

ping 10.101.100.11

Eth1 - 10.101.1.10

Eth2 - 10.101.100.11

Eth3 - 10.101.101.11

VM-1-A

149.165.130.210 - Eth0

Eth1 - 10.101.1.11

Eth2 - 10.101.100.12

Eth3 - 10.101.101.12

VM-1-B

149.165.130.211 - Eth0

# A bit about ovs-ofctl

- packaged with openvswitch-common
- alternative to dpctl (openflow reference controller)
- command-line utility that sends basic Openflow messages
  - useful for viewing switch port and flow stats, plus manually inserting flow entries
  - tool for early debugging
- Talks directly to the switch
  - This does not require a controller
- Switch must support a listener port

# First Step!

- Wireshark is running?
- Run:

  -
    ```
    $ ovs-ofctl show tcp:10.101.1.10X:6633
    ```
    - The 'show' command connects to the switch and prints out port state and OF capabilities
- What were the results?
- What do you see via wireshark?
- View Openflow rules on the switches at
  - http://workshop.incntre.iu.edu/flows/

# ovs-ofctl - show

```
$ ovs-ofctl show tcp:10.101.1.10X
OFPT_FEATURES_REPLY (xid=0x1): ver:0x1, dpid:000a2c27d7772d80
n_tables:2, n_buffers:256
features: capabilities:0x87, actions:0x7ff
 1(1): addr:2c:27:d7:77:2d:bf
     config:     0
     state:      0
     current:    100MB-FD AUTO_NEG
     supported:  10MB-HD 10MB-FD 100MB-HD 100MB-FD AUTO_NEG
 2(2): addr:2c:27:d7:77:2d:be
     config:     0
     state:      0
     current:    100MB-FD AUTO_NEG
     supported:  10MB-HD 10MB-FD 100MB-HD 100MB-FD AUTO_NEG
```

# ovs-ofctl dump-flows

- ovs-ofctl dump-flows tcp:10.101.1.10X
  - Gives us information about the flows installed
  - Rule itself
  - Timeouts
  - Actions
  - Packets and bytes processed by flow

# ovs-ofctl dump-flows

```
$ ovs-ofctl dump-flows tcp:10.101.1.10X
```

```
1. NXST_FLOW reply (xid=0x4):
2. cookie=0x0, duration=30.625s, table=4, n_packets=0,
n_bytes=2612, idle_timeout=180,priority=33000,in_port=1
actions=output:2
3. cookie=0x0, duration=22.5s, table=4, n_packets=0,
n_bytes=2612, idle_timeout=180,priority=33000,in_port=2
actions=output:1
```

# ovs-ofctl dump-ports

```
$ ovs-ofctl dump-ports tcp:10.101.1.10X
```

- Gives physical port information
- Rx, tx counters
- Error counters

1. OFPST_PORT reply (xid=0x1): 14 ports
2. port 2: rx pkts=25211, bytes=3856488, drop=0, errs=0, frame=0, over=0, crc=0tx pkts=7144, bytes=767594, drop=0, errs=0,coll=0
3. port 5: rx pkts=18235, bytes=3142702, drop=0, errs=0, frame=0, over=0, crc=0tx pkts=0, bytes=0, drop=0, errs=0, coll=0

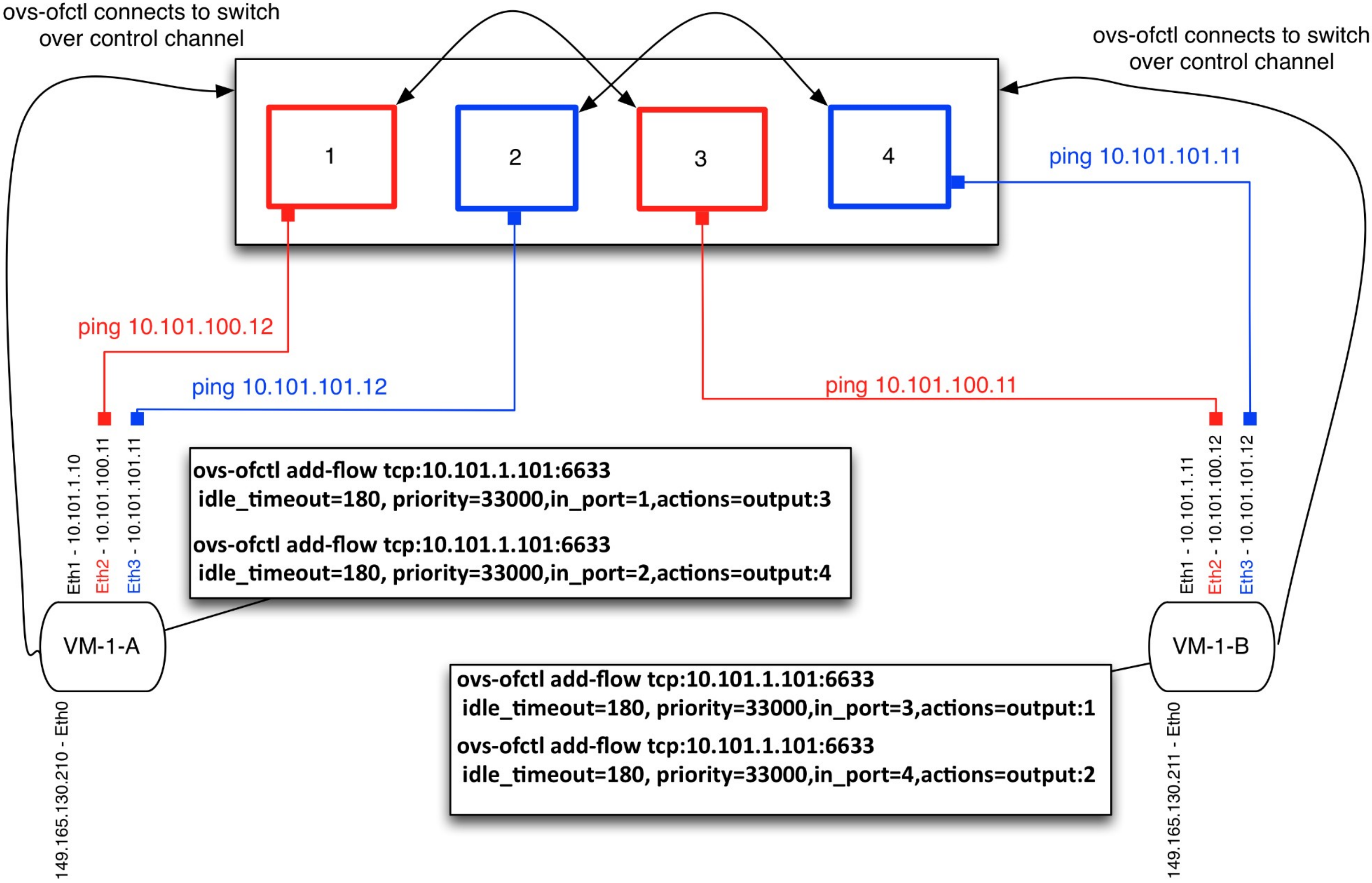
# ovs-ofctl del-flows

- we can remove all or individual flows from the switch
- $ ovs-ofctl del-flows match
  - ex. $ ovs-ofctl del-flows tcp:10.101.1.101 dl_type=0x800
  - ex. $ ovs-ofctl del-flows tcp:10.101.1.101 in_port=1

# Exercise #1

Using ovs-ofctl to insert simple, port-based rules

# Using OpenFlow Port-based Rules To Interconnect Ports via ovs-ofctl



```
ovs-ofctl add-flow tcp:10.101.1.101:6633
 idle_timeout=180, priority=33000,in_port=1,actions=output:3

ovs-ofctl add-flow tcp:10.101.1.101:6633
 idle_timeout=180, priority=33000,in_port=2,actions=output:4
```

```
ovs-ofctl add-flow tcp:10.101.1.101:6633
 idle_timeout=180, priority=33000,in_port=3,actions=output:1

ovs-ofctl add-flow tcp:10.101.1.101:6633
 idle_timeout=180, priority=33000,in_port=4,actions=output:2
```

# Do the pings work?

What do you see with
ovs-ofctl dump-flows tcp:10.101.1.10X?

Wireshark?

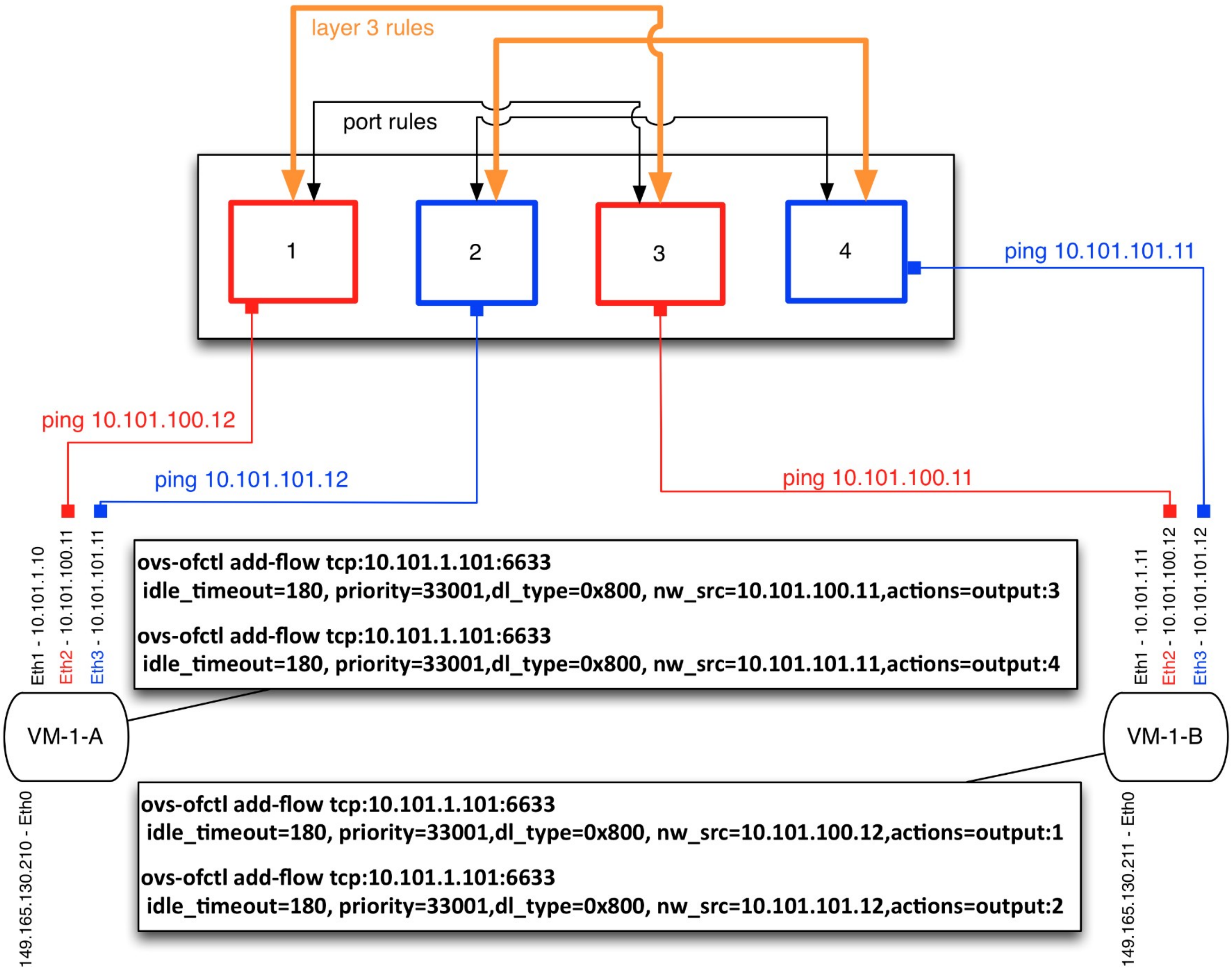
Do the counters increase as expected?

What's going on with the timeouts?

# Exercise #2 - Moving up the stack...

First rule was port-based.

Next rule is IP source address-based.



```
ovs-ofctl add-flow tcp:10.101.1.101:6633
 idle_timeout=180, priority=33001,dl_type=0x800, nw_src=10.101.100.11,actions=output:3

ovs-ofctl add-flow tcp:10.101.1.101:6633
 idle_timeout=180, priority=33001,dl_type=0x800, nw_src=10.101.101.11,actions=output:4
```

```
ovs-ofctl add-flow tcp:10.101.1.101:6633
 idle_timeout=180, priority=33001,dl_type=0x800, nw_src=10.101.100.12,actions=output:1

ovs-ofctl add-flow tcp:10.101.1.101:6633
 idle_timeout=180, priority=33001,dl_type=0x800, nw_src=10.101.101.12,actions=output:2
```

# Do the pings work?

Did the port-based rules time out?

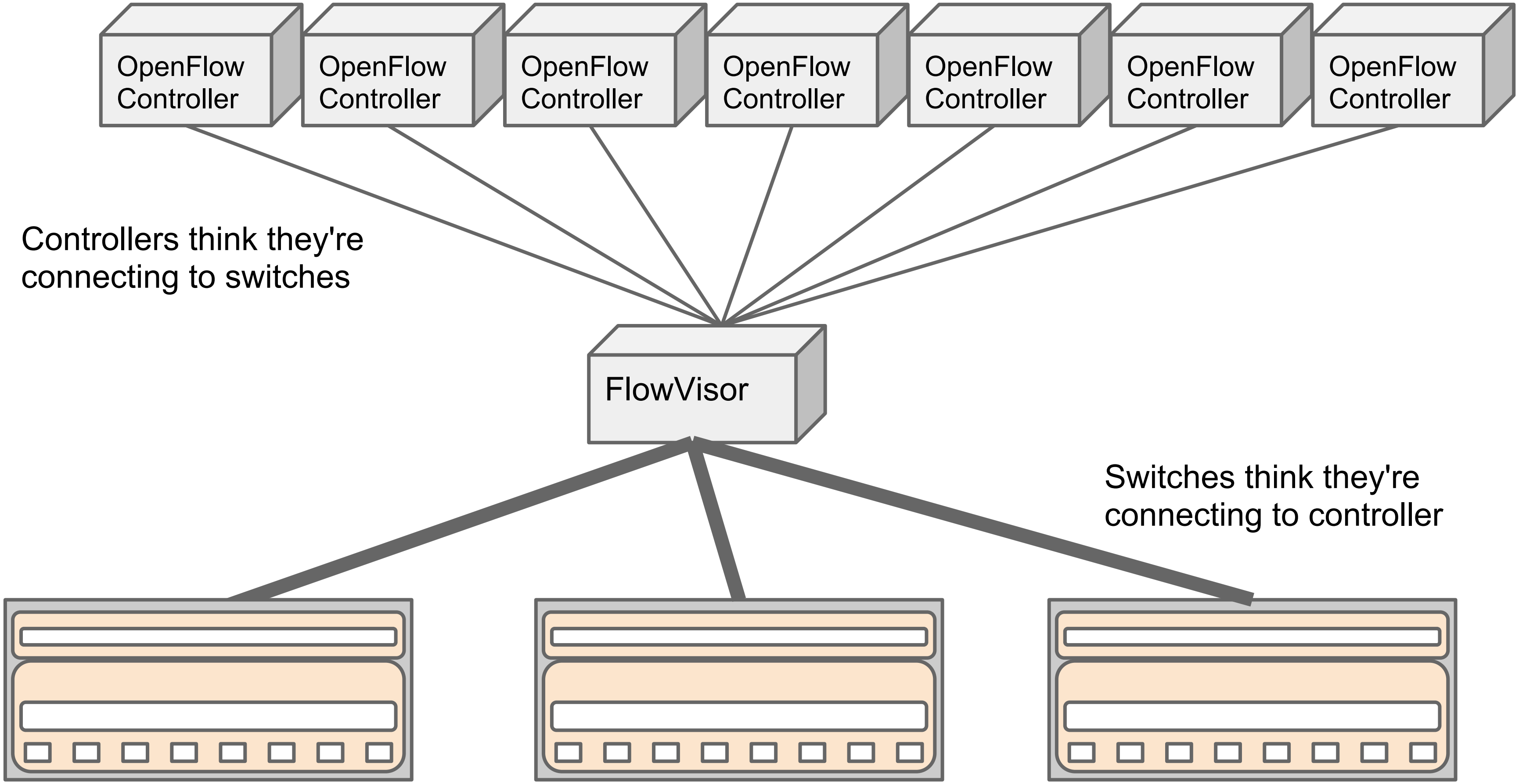
If there are no port-based rules, why would the pings fail?

Can you verify this hypothesis by looking at the counters?

# Exercise #3 - uses an OF controller... but we have to introduce FlowVisor first...

The practical reason we're using FlowVisor at this point is to provide enough virtual OpenFlow switches so that each student can operate his own OpenFlow controller.

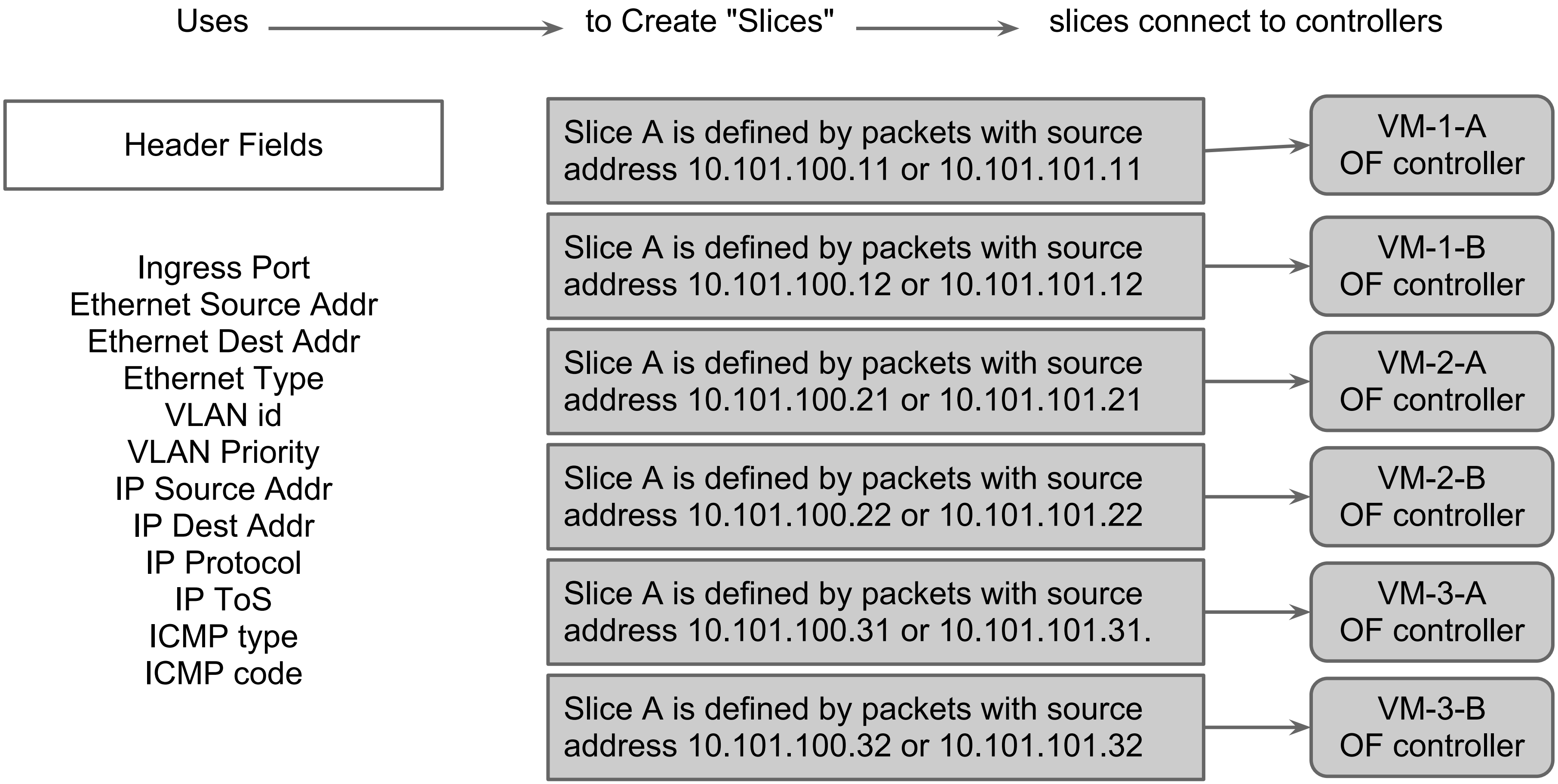
# FlowVisor



Controllers think they're connecting to switches

Switches think they're connecting to controller

# FlowVisor



Uses → to Create "Slices" → slices connect to controllers

Header Fields

Ingress Port
Ethernet Source Addr
Ethernet Dest Addr
Ethernet Type
VLAN id
VLAN Priority
IP Source Addr
IP Dest Addr
IP Protocol
IP ToS
ICMP type
ICMP code

Slice A is defined by packets with source address 10.101.100.11 or 10.101.101.11 → VM-1-A OF controller

Slice A is defined by packets with source address 10.101.100.12 or 10.101.101.12 → VM-1-B OF controller

Slice A is defined by packets with source address 10.101.100.21 or 10.101.101.21 → VM-2-A OF controller

Slice A is defined by packets with source address 10.101.100.22 or 10.101.101.22 → VM-2-B OF controller

Slice A is defined by packets with source address 10.101.100.31 or 10.101.101.31. → VM-3-A OF controller

Slice A is defined by packets with source address 10.101.100.32 or 10.101.101.32 → VM-3-B OF controller

# OpenFlow Controller

ovs-controller

simple controller

will use to push a few rules into the switch (really a slice)

to get started let's start the controller with debug options and see what happens:

```
sudo ovs-controller --verbose --noflow --mute ptcp:
```

# What happened?

What do you see in debug output?

Did a switch connect to the controller?

What shows up on wireshark?

# Create some flows to push into a switch

use pico, vi, whatever...

to create a text file that contains flow entries in the same format as the ovs-ofctl command, name the file flows.txt

For VM-1-A flows.txt would look like this:

```
priority=33000,in_port=1,actions=output:3
priority=33000,in_port=2,actions=output:4
```

For VM-1-B flows.txt would look like this:

```
priority=33000,in_port=3,actions=output:1
priority=33000,in_port=4,actions=output:2
```

```
sudo ovs-controller --verbose --noflow --mute --with-flows flows.txt ptcp:
```

# Does the ping work???

```
ovs-ofctl dump-flows tcp:10.101.1.10X
```

How did these rules:

```
priority=33000,in_port=1,actions=output:3
priority=33000,in_port=2,actions=output:4
priority=33000,in_port=3,actions=output:1
priority=33000,in_port=4,actions=output:2
```

Become these:

```
in_port=2,nw_src=10.101.100.11 actions=output:4
in_port=2,nw_src=10.101.101.11 actions=output:4
in_port=4,nw_src=10.101.100.12 actions=output:2
in_port=4,nw_src=10.101.101.12 actions=output:2
in_port=3,nw_src=10.101.100.12 actions=output:1
in_port=3,nw_src=10.101.101.12 actions=output:1
in_port=1,nw_src=10.101.100.11 actions=output:3
in_port=1,nw_src=10.101.101.11 actions=output:3
```

# Exercise 4 - Slicing the network

| Switch | Flowvisor VM |
|---|---|
| #1 - 10.101.1.101 | 149.165.130.250 |
| #2 - 10.101.1.102 | 149.165.130.251 |
| #3 - 10.101.1.103 | 149.165.130.252 |
| #4 - 10.101.1.104 | 149.165.130.253 |

- Each person needs to login to the VM that is running your switch's Flowvisor
  - ssh openflow@149.165.130.25X

# Creating Slices

- Each person needs to create a slice for their VM
  - fvctl --passwd-file=/usr/etc/flowvisor/fvpasswd createSlice slicename controller_url email

  - fvctl --passwd-file=/usr/etc/flowvisor/fvpasswd createSlice [your VM's name e.g. "VM_1_A"] tcp:[IP address of your Eth1]:6633 fakemail@you.com

- Verify that your slice exists:
  - fvctl --passwd-file=/usr/etc/flowvisor/fvpasswd listSlices

# Verifying Slices

- Verify that the settings are correct:

  -
    ```
    fvctl --passwd-file=/usr/etc/flowvisor/fvpasswd getSliceInfo [your slice's name]
    ```

```
connection_1=00:0a:2c:27:d7:76:ea:80-->NONE (retry in 7 seconds: max + 15)
contact_email=fakename@incntre.iu.edu
controller_hostname=10.101.1.10
controller_port=6633
creator=root
```

Unless you are running a controller on port 6633, you should not see a connection the controller

# Adding Flowspace

- You need to find the DPID of your switch
  - fvctl --passwd-file=/usr/etc/flowvisor/fvpasswd listDevices
    - Device 0: 00:0a:2c:27:d7:77:2d:80

# Find your Eth2 and Eth3 Ethernet addresses...

run ifconfig on your VM to find your VM's Eth2 and Eth3 Ethernet addresses, you'll need them shortly.

# addFlowSpace

- We need to add Flowspace to allow you to write rules from your controller to the switch with the FV as a proxy
  - fvctl --passwd-file=/usr/etc/flowvisor/fvpasswd addFlowSpace [switch_dpid] [priority] [flow_match] [slice_name]
  - Ex:
    - fvctl --passwd-file=/usr/etc/flowvisor/fvpasswd addFlowSpace 00:0a:2c:27:d7:76:ea:80 100 dl_type=0x800,nw_src=[IP Address of your Eth2] Slice:[name of your slice]=4
    - fvctl --passwd-file=/usr/etc/flowvisor/fvpasswd addFlowSpace 00:0a:2c:27:d7:76:ea:80 100 dl_type=0x806,dl_src=[Ethernet address of your Eth2] Slice:[name of your slice]=4
    - (more commands on next page...)

# Need to add a bit more flowspace

- fvctl --passwd-file=/usr/etc/flowvisor/fvpasswd addFlowSpace 00:0a:2c:27:d7:76:ea:80 100 dl_type=0x800,nw_src=[IP Address of your Eth3] Slice:[name of your slice]=4
- fvctl --passwd-file=/usr/etc/flowvisor/fvpasswd addFlowSpace 00:0a:2c:27:d7:76:ea:80 100 dl_type=0x806,dl_src=[Ethernet address of your Eth3] Slice:[name of your slice]=4

# listFlowSpace

- verify the flowspace that you added
  - fvctl --passwd-file=/usr/etc/flowvisor/fvpasswd listFlowSpace

# Running the Controller

- Similar to before, we are going to run the ovs-controller on your VM
  - sudo ovs-controller -v ptcp:6633 --with-flows your_flows.txt
- What do the flows in your_flows.txt need to look like?
  - you need to handle
    - dl_type=0x806
    - dl_type=0x800
    - Do you want to do layer 2 only rules or also match on IP?

# An example of reactive OF Control

Implement an observe the behavior of a controller-based learning switch

Using Floodlight as the controller and OpenVSwitch as the OpenFlow Switch

# OpenVswitch

What is it?

Where did come from?

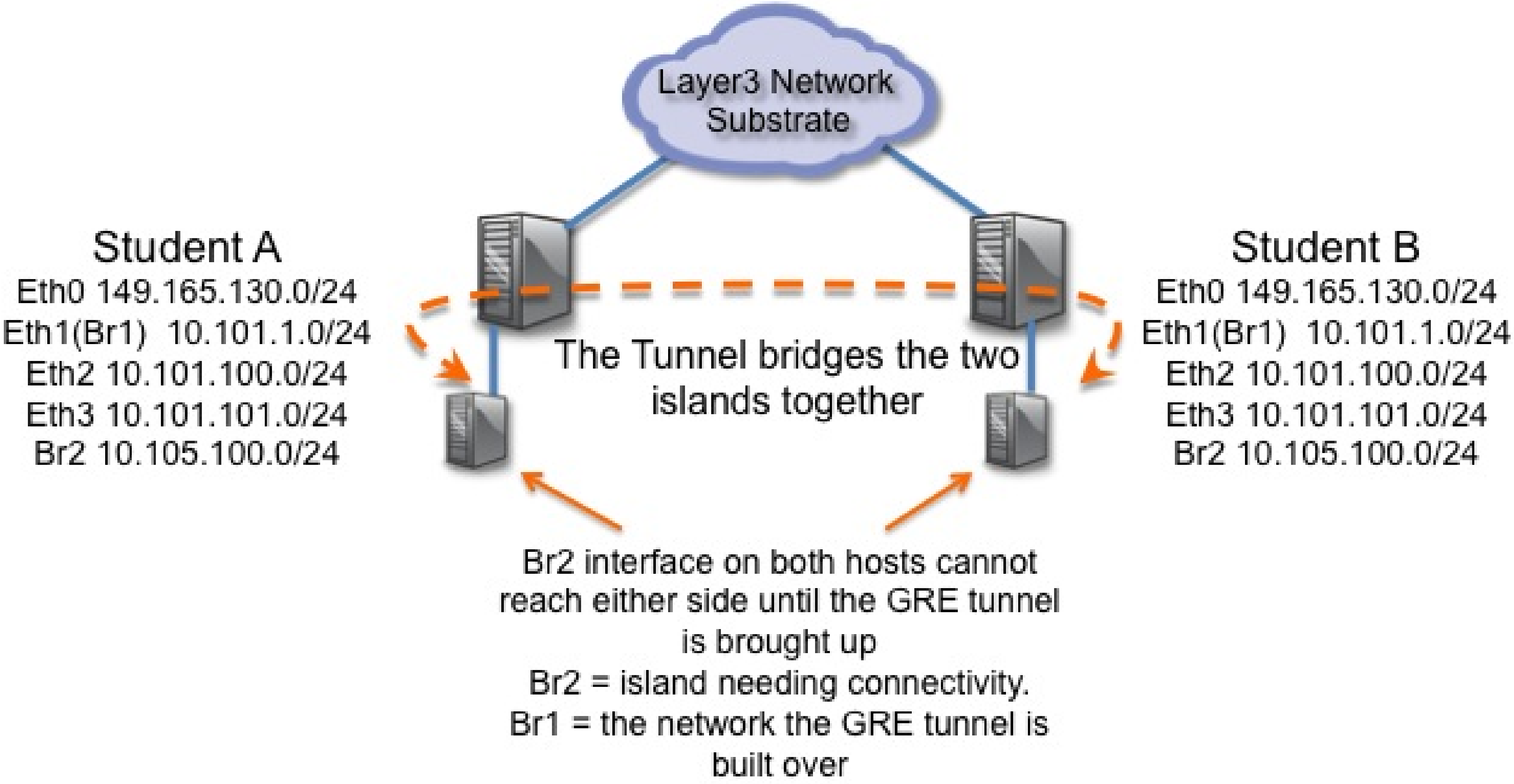
Why is it useful?

# Floodlight

Real controller

Can be used as a development platform

# Topology of OVS switch exercise

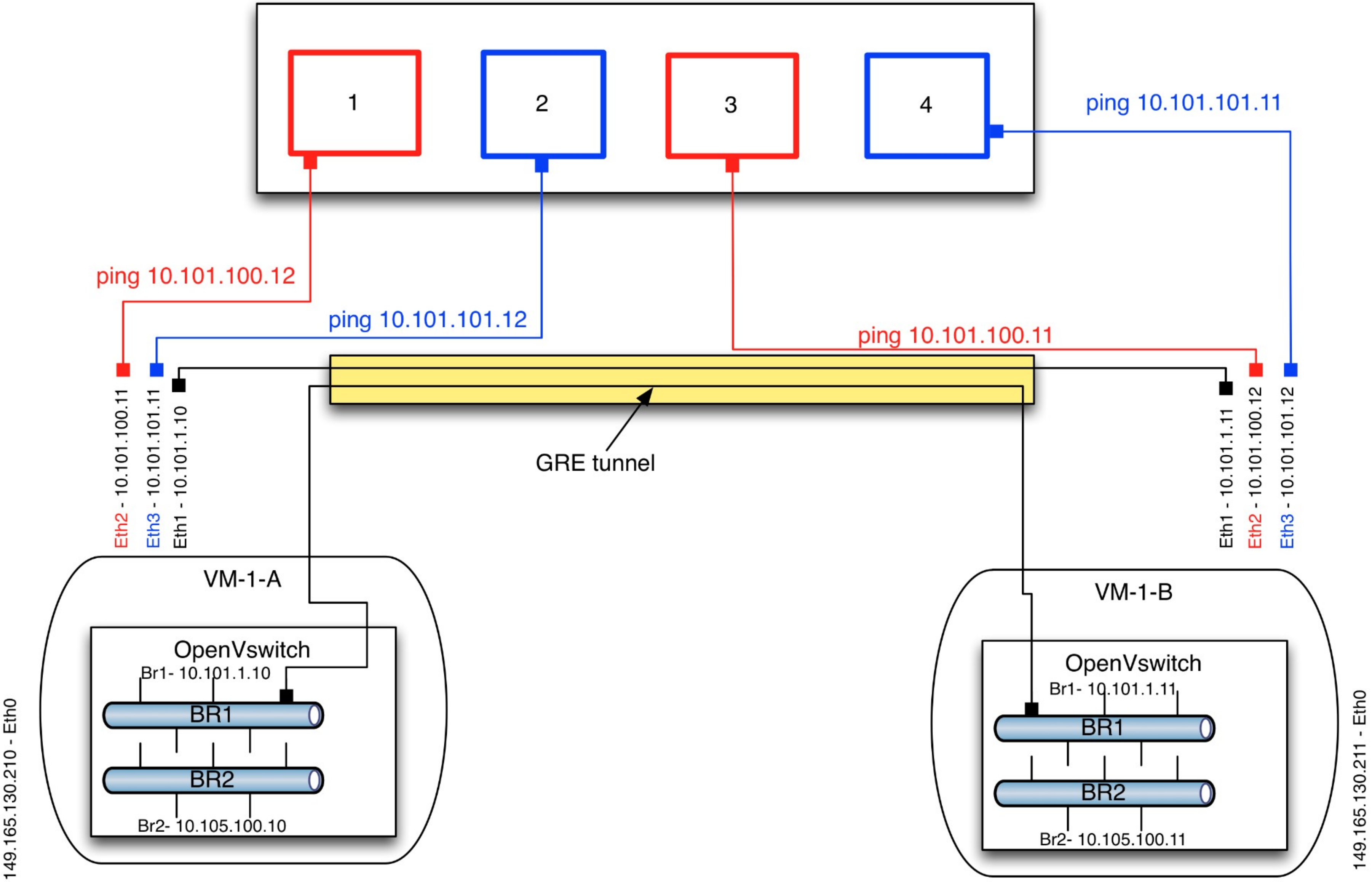


Student A
Eth0 149.165.130.0/24
Eth1(Br1) 10.101.1.0/24
Eth2 10.101.100.0/24
Eth3 10.101.101.0/24
Br2 10.105.100.0/24

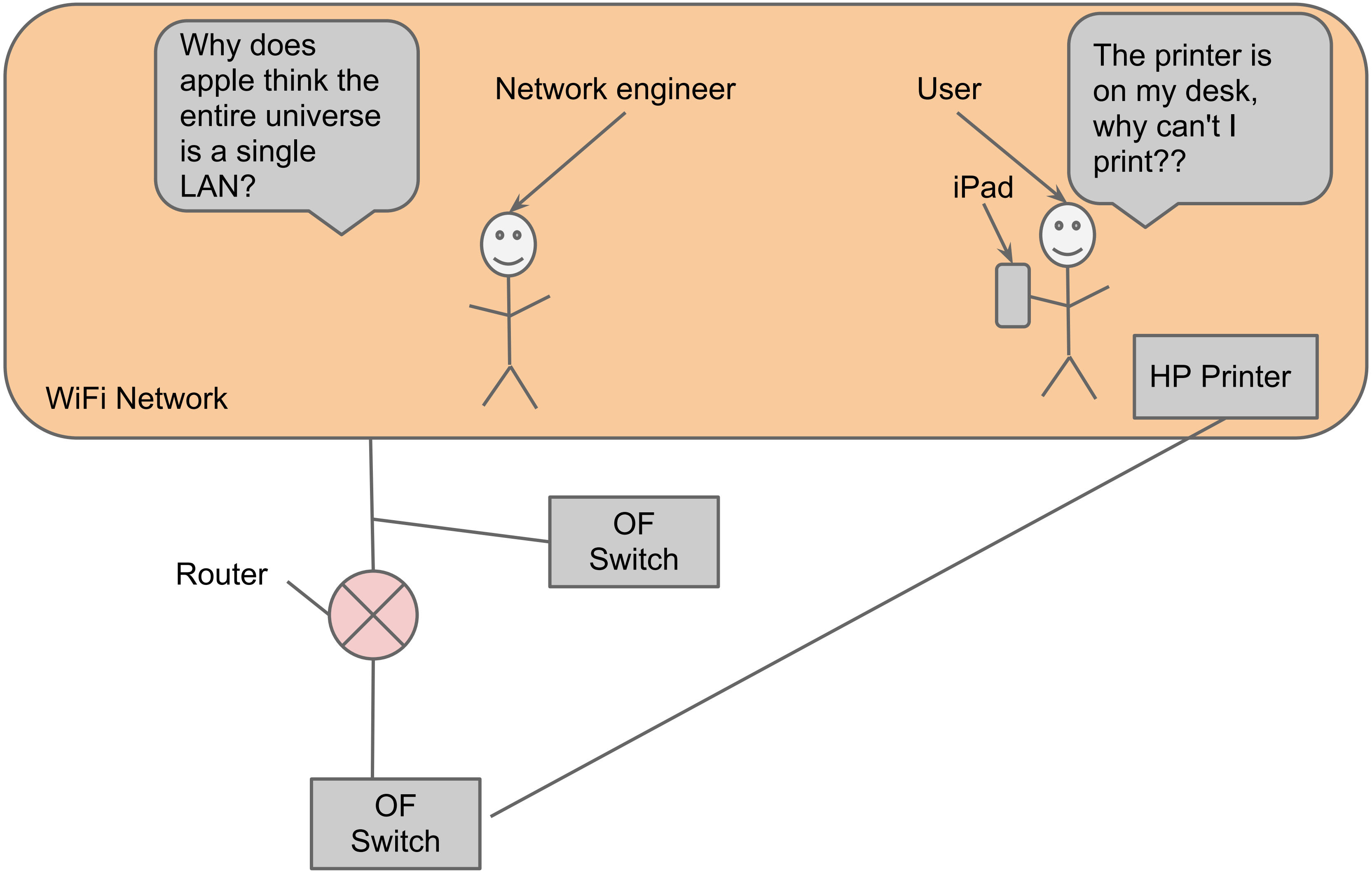
Student B
Eth0 149.165.130.0/24
Eth1(Br1) 10.101.1.0/24
Eth2 10.101.100.0/24
Eth3 10.101.101.0/24
Br2 10.105.100.0/24

The Tunnel bridges the two islands together

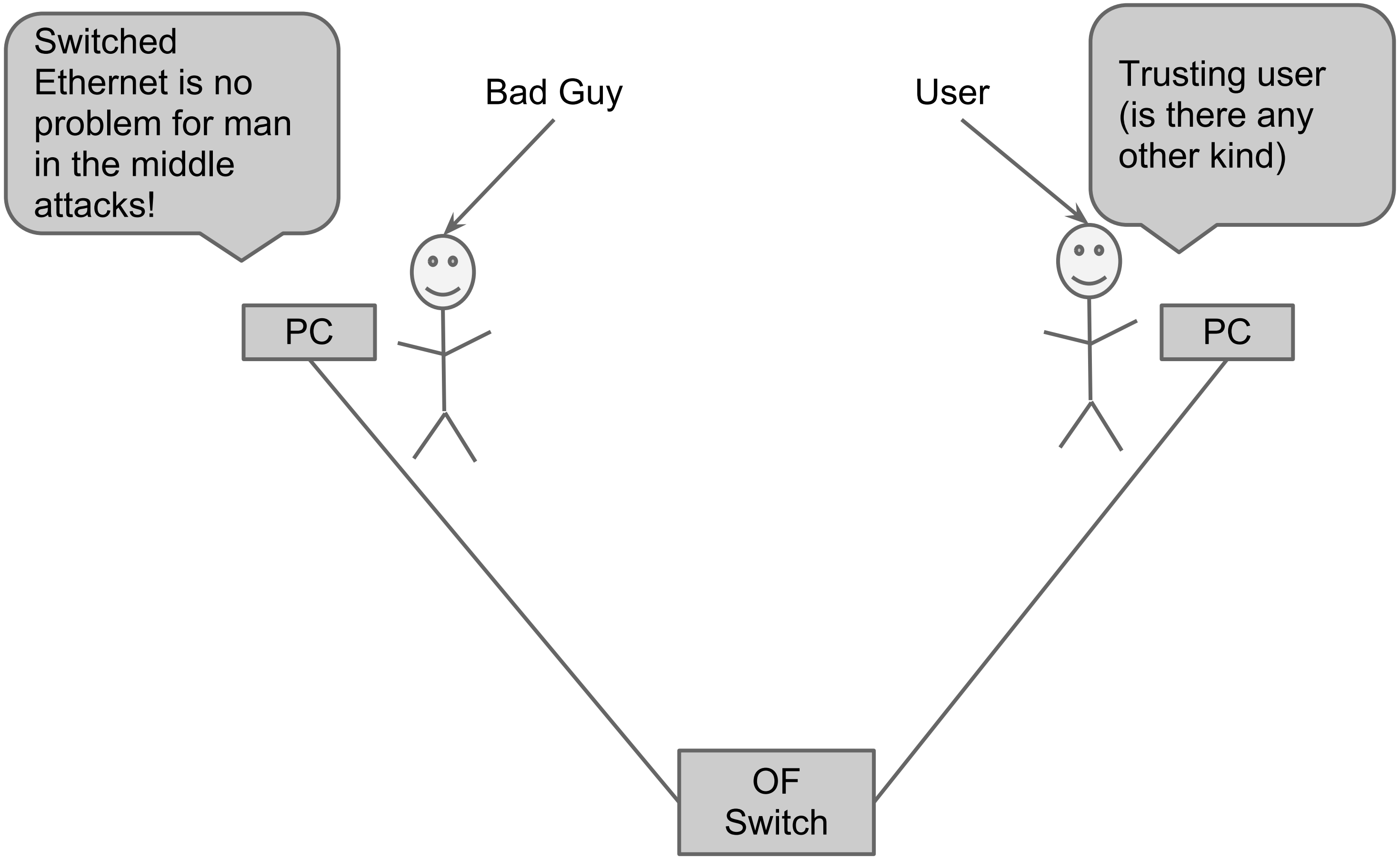
Br2 interface on both hosts cannot reach either side until the GRE tunnel is brought up
Br2 = island needing connectivity.
Br1 = the network the GRE tunnel is built over


1
2
3
4
ping 10.101.101.11
ping 10.101.100.12
ping 10.101.101.12
ping 10.101.100.11
Eth2 - 10.101.100.11
Eth3 - 10.101.101.11
Eth1 - 10.101.1.10
GRE tunnel
Eth1 - 10.101.1.11
Eth2 - 10.101.100.12
Eth3 - 10.101.101.12
VM-1-A
OpenVswitch
Br1- 10.101.1.10
BR1
BR2
Br2- 10.105.100.10
149.165.130.210 - Eth0
VM-1-B
OpenVswitch
Br1- 10.101.1.11
BR1
BR2
Br2- 10.105.100.11
149.165.130.211 - Eth0

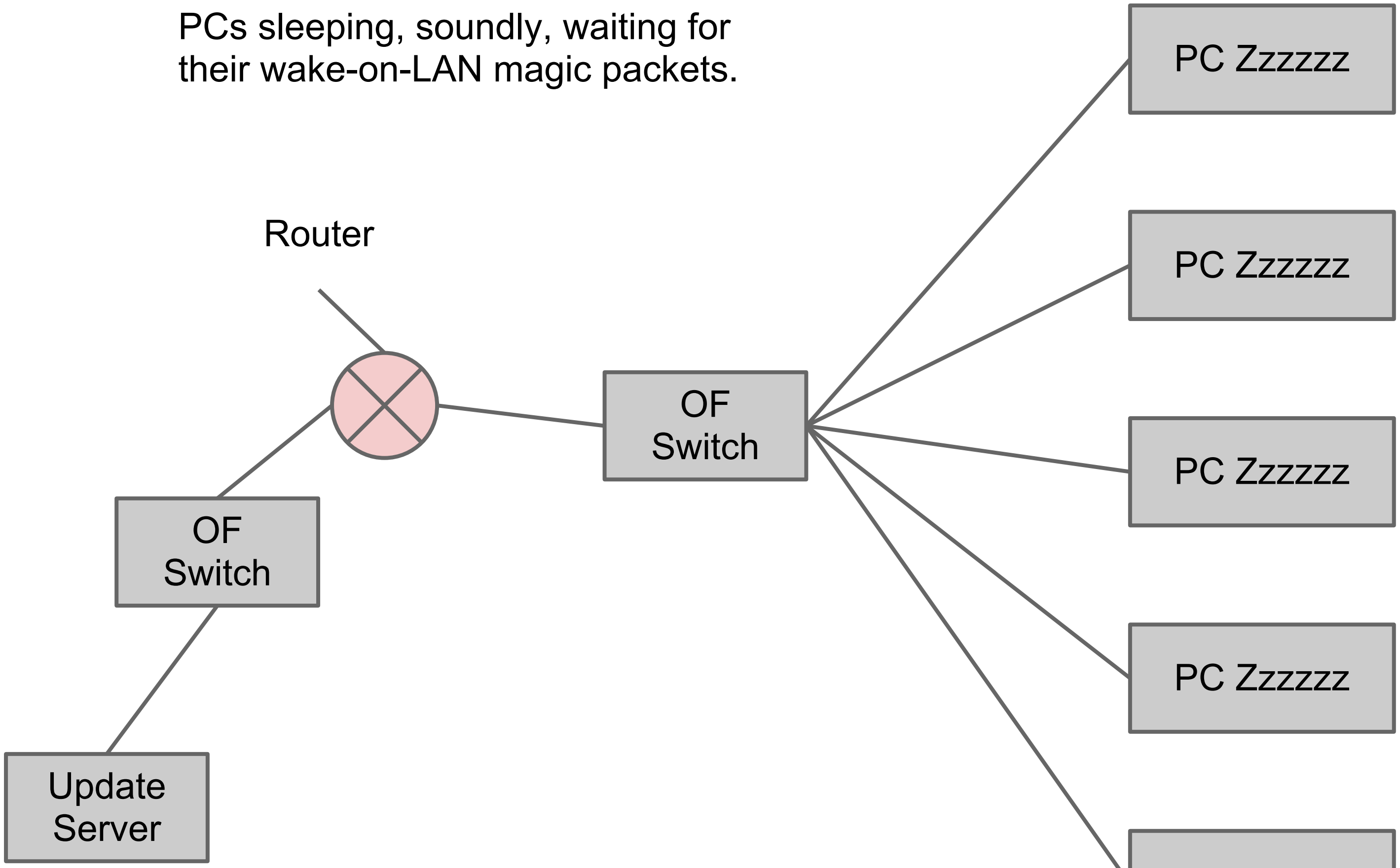
# If OpenFlow was your only tool...


Why does apple think the entire universe is a single LAN?
Network engineer
User
The printer is on my desk, why can't I print??
iPad
WiFi Network
HP Printer
OF Switch
Router
OF Switch

# If OpenFlow was your only tool...


Switched Ethernet is no problem for man in the middle attacks!
Bad Guy
User
Trusting user (is there any other kind)
PC
PC
OF Switch

# If OpenFlow was your only tool...



PCs sleeping, soundly, waiting for
their wake-on-LAN magic packets.