CSCI 6907.11

Adv. Net. Sys. Prog.

Week 7
Software Defined Networks

Tim Wood
CS@GWU
2015

Some content / ideas from Nick Feamster's Coursera SDN class and Kurose & Ross's book

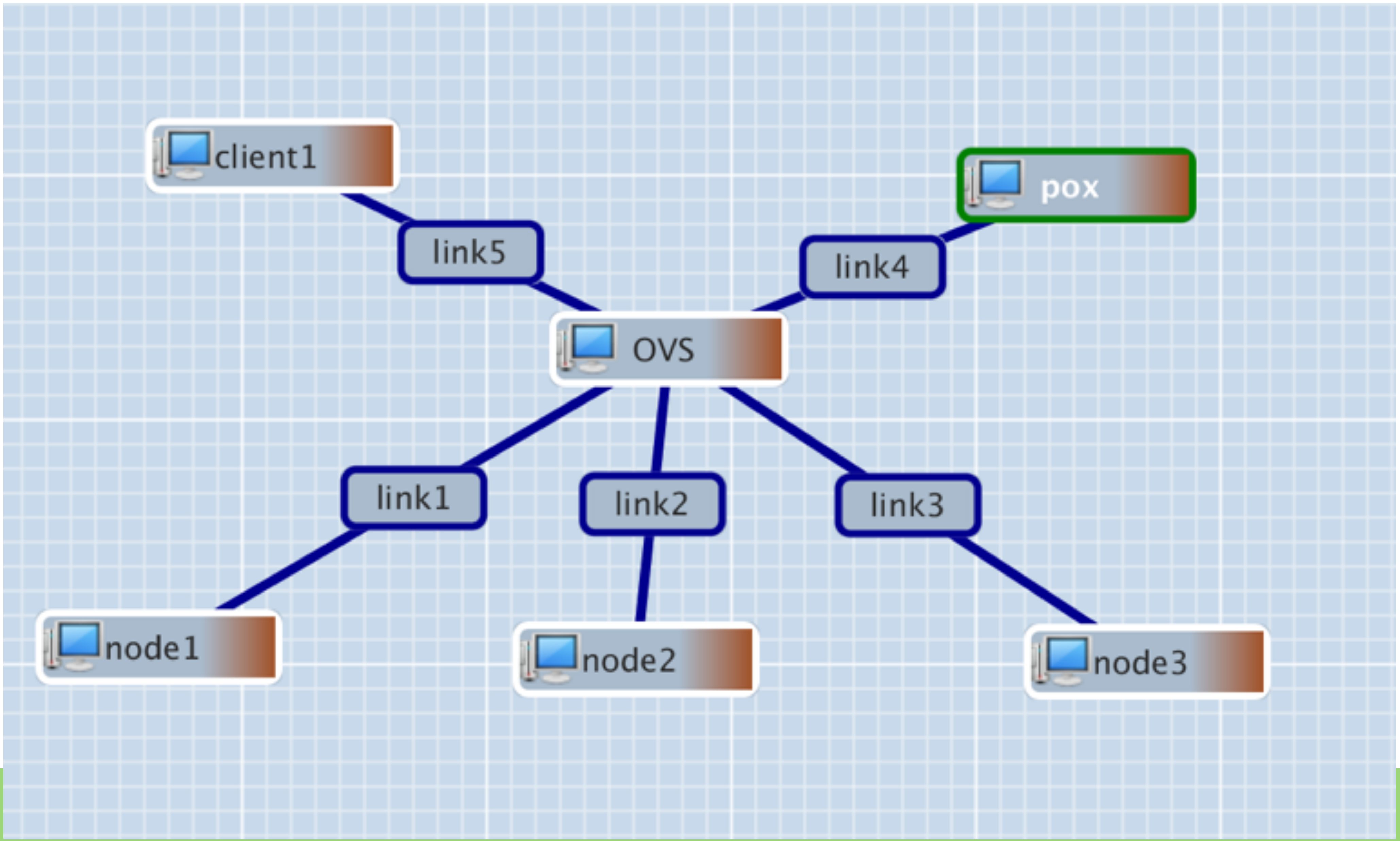
Today

- GENI Competition Plans
- Issues Round 2
- Control and Data Planes
- SDNs on GENI
- Open vSwitch: software data plane
- Pox: control plane

SDN Setup

Use jFed to request these resources from **InstaGENI**

- 6 Xen VMs
- Client, POX, node1-3 all use “Ubuntu 14.04” disk image
- OVS uses “Ubuntu 12.04 with OVS (Niky)” disk image



Issue Sharing

Show the code for the issue(s) you solved to your table neighbors

Code review!

- Check for style, readability, comments
- Check for correctness! Are there test cases?

Control & Data

Networks send data *and* commands

Telephone

- Data? Control?

Internet?

- Data? Control?

Central vs Distributed

AT&T Network Control Plane

- **Centralized** management point for voice network
- Can directly observe network status
- Easily deploy new services (800 numbers)
- Pros and cons?

Internet routing

- Routers communicate with adjacent routers to advertise routes
- Distributed shortest path algorithm used to decide best route
- Pros and cons?

Software Defined Networking

Separate control and data

- Control: SDN Controller
- Data plane: switches and routers

(Logically) Centralized controller

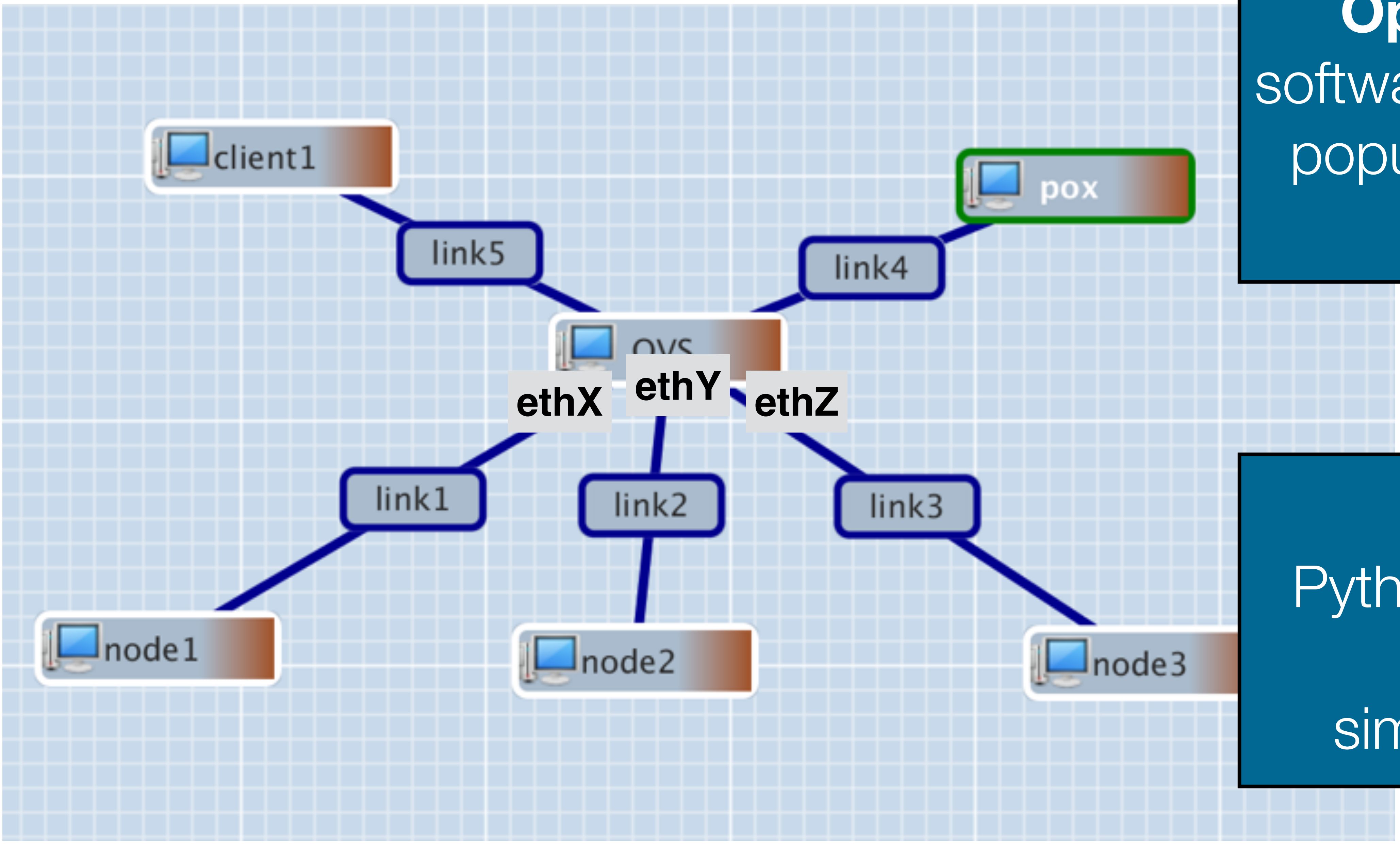
- Makes decisions about how packets are routed
- Sends rules to data plane elements

Data plane

- High performance but dumb
- Just follows the rules told to it by the controller

SDN Setup

We will use **OVS** and **POX** to control how data is forwarded between node1-3



Open VSwitch
software based switch popular for bridging VMs

POX
Python-based SDN Controller
simple and easy

Setup nodes

Set them all to have IPs in the same subnet

```
// ssh into each node and set the IP
// don't edit client, POX, or OVS hosts yet
timwoo01@node1:~$ sudo ifconfig eth1 192.168.10.1
timwoo01@node2:~$ sudo ifconfig eth1 192.168.10.2
timwoo01@node3:~$ sudo ifconfig eth1 192.168.10.3
```

What happens if you try to ping from one node to the other? Why?

Don't accidentally mess up eth0!!!!!

Setup on OVS VM

Disable existing interfaces

- **Don't shutdown eth0!!!!!!!**

```
sudo ifconfig ethX 0
sudo ifconfig ethY 0
sudo ifconfig ethZ 0
```

Check that OVS is working

```
sudo ovs-vsctl show
725406ec-3e20-46fe-85cf-8d5030bcfa4a
    ovs_version: "1.9.3"
```

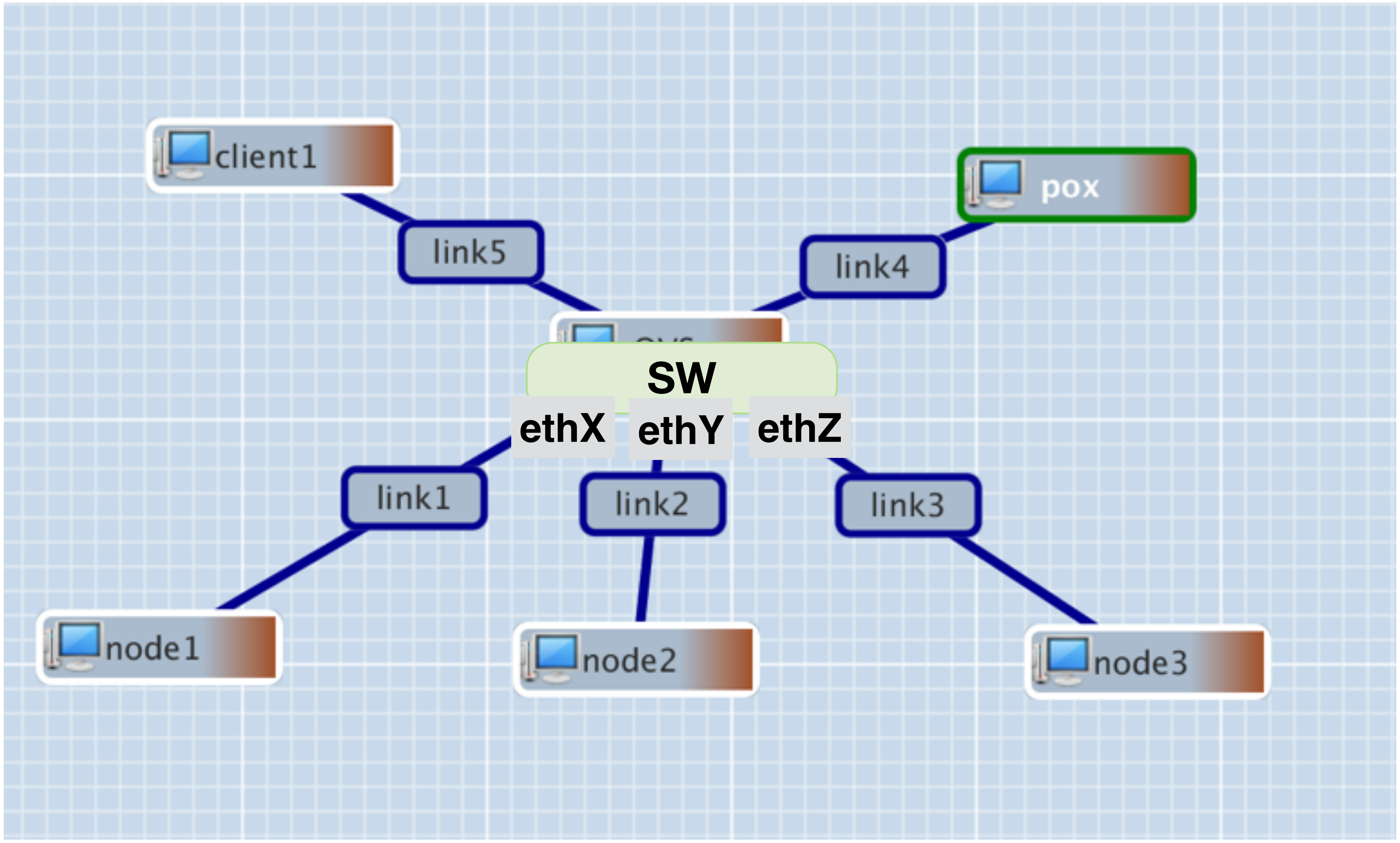
Create a new bridge and assign interfaces to it

```
sudo ovs-vsctl add-br br0
sudo ovs-vsctl add-port br0 ethX
sudo ovs-vsctl add-port br0 ethY
sudo ovs-vsctl add-port br0 ethZ
```

Now can you ping?

OVS

Open vSwitch is acting as an ethernet switch, forwarding packets between the hosts on its bridge



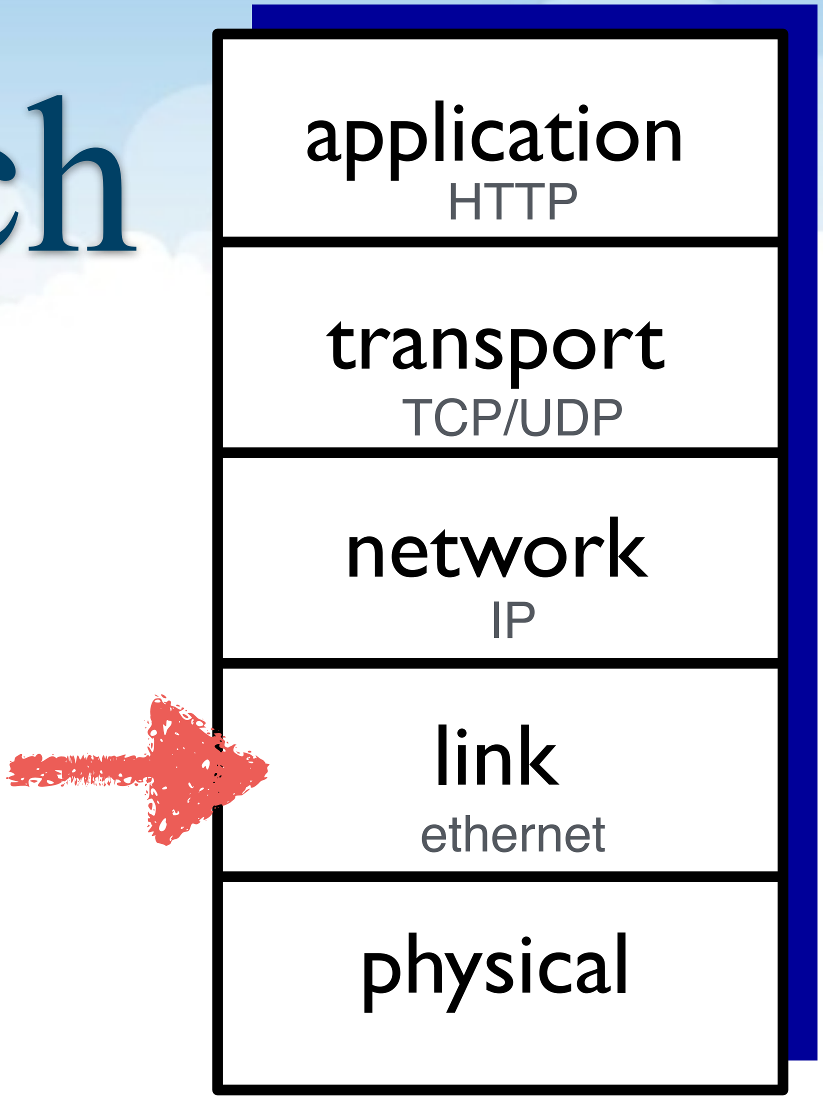
Layer 2 Switch

Switches work at Layer 2

- Ethernet

Forwards packets based on MAC

- Address hard coded into NIC hardware
- Or assigned to a VM at bootup



802.3 Ethernet packet and frame structure

Preamble	Start of frame delimiter	MAC destination	MAC source	802.1Q tag (optional)	Ethertype (Ethernet II) or length (IEEE 802.3)	Payload	Frame check sequence (32-bit CRC)	Interpacket gap
7 octets	1 octet	6 octets	6 octets	(4 octets)	2 octets	46(42)[b]–1500 octets	4 octets	12 octets

wikipedia

Software Defined Networking

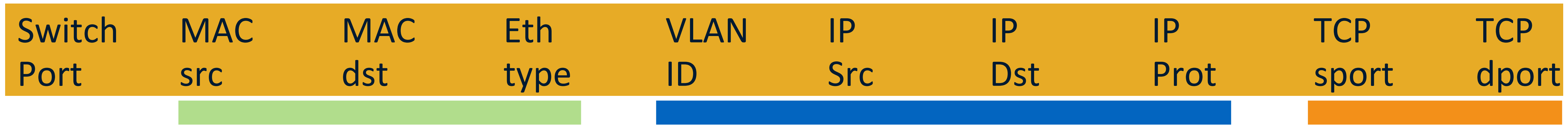
Separate the network's control and data plane

Data plane: forwards packets between switches

Control plane: determines routes for packets

Controller matches **flows**:

- a stream of packets from a source to a destination



Controller provides **actions** for each flow:

- drop
- forward out a port
- modify headers

Open Flow

Open Flow is a protocol used in SDN

- Defines the messages between a switch and the controller
- Originally designed for simple hardware switches
- Is slowly evolving to support more advanced software switches

Defines events

- Switch boots up
- New packet arrives

Defines match/action rules

- How to define a flow
- What actions can be performed on a flow

Not all hardware switches support OpenFlow!

SDN Controllers

Lots of options…

	NOX	POX	Ryu	Floodlight	ODL
Language	C++	Python	Python	Java	Java
Performance	Fast	Slow	Slow	Fast	Fast
Distributed	No	No	Yes	Yes	Yes
OpenFlow	1.0 (CPqD: 1.1, 1.2, 1.3)	1.0	1.0, 1.1, 1.3, 1.4	1.0	1.0, 1.3
Multi-tenant Clouds	No	No	Yes	Yes	Yes
Learning Curve	Moderate	Easy	Moderate	Steep	Steep

POX

Simple, low performance, python SDN controller

- Popular for prototyping and research projects

POX core engine handles communication with switch

- Uses Openflow 1.0

You write handlers to respond to certain events

- A new packet arriving that doesn't match a rule
- A new switch coming online
- etc

Your python code can do anything

- Has the POX libraries available to send open flow messages

Setting up POX

Check out our repo to get POX

- Be sure to update your fork first!

```
// on POX host
git clone https://github.com/YOURFORK/adv-net-samples.git
```

Run a pox program

```
cd adv-net-samples/sdn/pox
./pox.py --verbose SuperSimple
```

This just prints out info about any packets it gets

Setting up POX

Need to tell OVS where the controller is

- Also tell it to **only** use rules explicitly sent to it by the controller

```
// on OVS host
sudo ovs-vsctl set-controller br0 tcp:<controller_ip>:6633
sudo ovs-vsctl set-fail-mode br0 secure
```

How well do your pings work now? Why?

Arrrrrp???

What is this ARP thing?

Sockets work with IP addresses (or host names)

Switches and ethernet frames work with MACs…

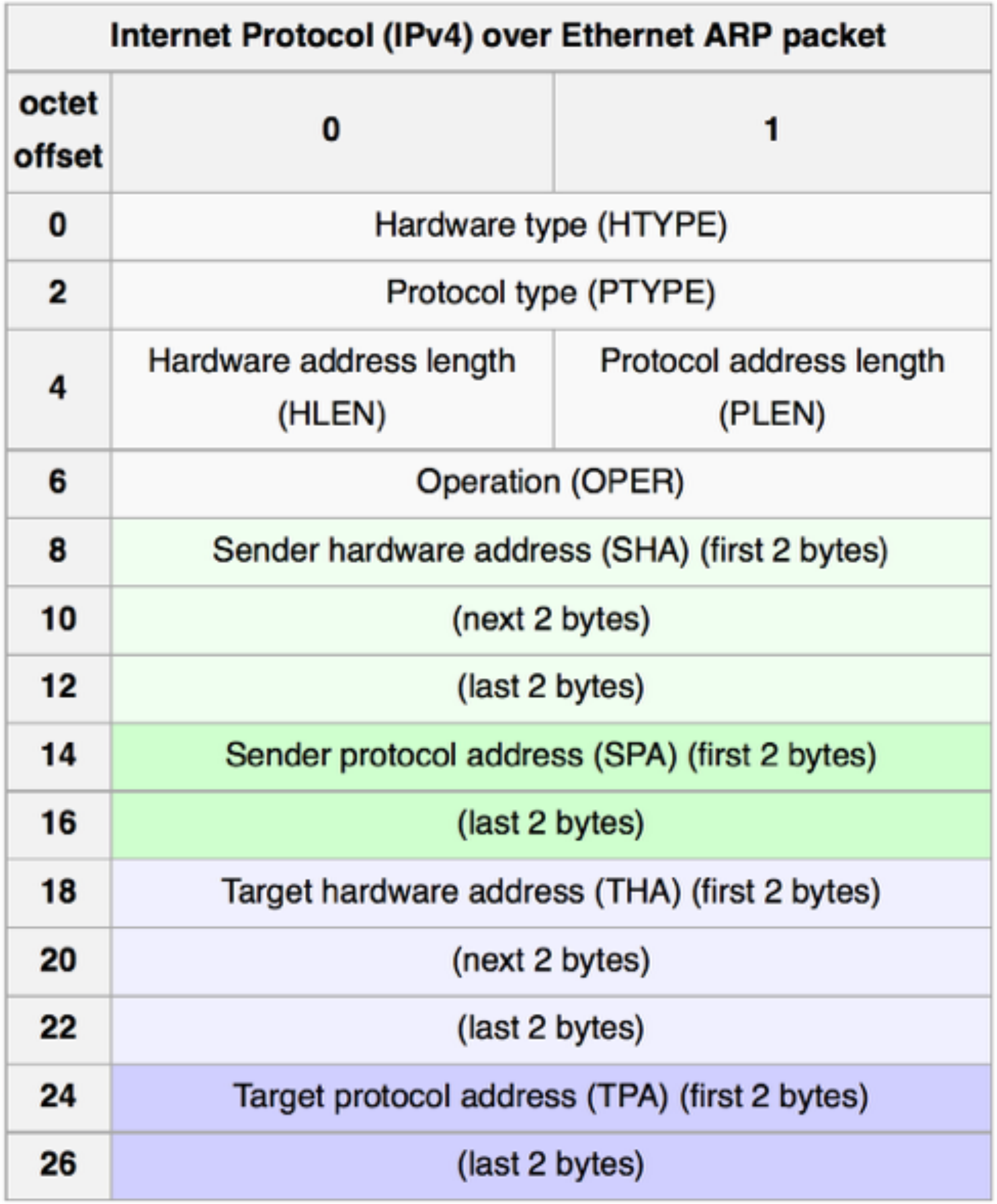
Address **R**esolution **P**rotocol

- Translates from Link Layer (L2 ethernet) to Network Layer (L3 IP)

Sending a message to IP **14.164.13.123**:

- Check ARP cache to see if it has 14.164.13.123
- If yes, send to the MAC stored for it
- If not, broadcast an ARP request to MAC ff:ff:ff:ff:ff:ff
- Wait for a reply which will indicate the MAC for that IP

ARP Message Format



Internet Protocol (IPv4) over Ethernet ARP packet		
octet offset	0	1
0	Hardware type (HTYPE)	
2	Protocol type (PTYPE)	
4	Hardware address length (HLEN)	Protocol address length (PLEN)
6	Operation (OPER)	
8	Sender hardware address (SHA) (first 2 bytes)	
10	(next 2 bytes)	
12	(last 2 bytes)	
14	Sender protocol address (SPA) (first 2 bytes)	
16	(last 2 bytes)	
18	Target hardware address (THA) (first 2 bytes)	
20	(next 2 bytes)	
22	(last 2 bytes)	
24	Target protocol address (TPA) (first 2 bytes)	
26	(last 2 bytes)	

Src MAC

Src IP

Dest MAC

Dest IP

Arp

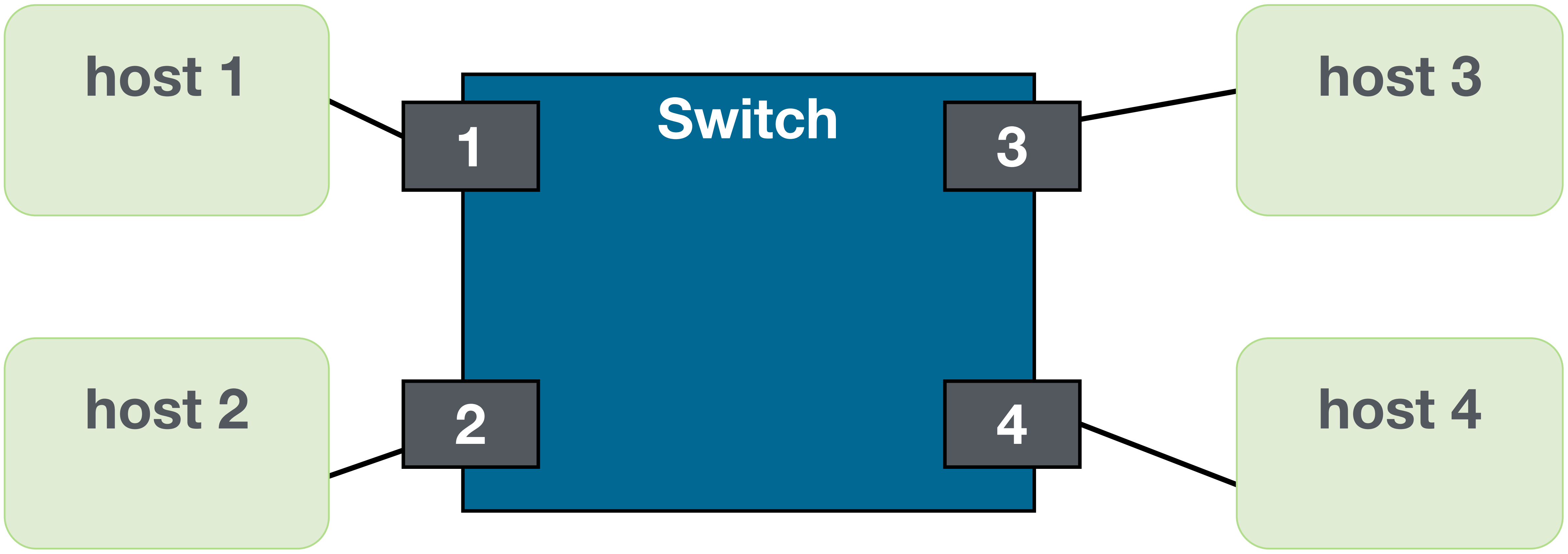
Use the **arp** command to view a host's arp table:

```
timwoo01@node2:~$ arp
Address                  HWtype  HWaddress           Flags           Iface
172.17.253.254           ether   fe:ff:ff:ff:ff:ff   C               eth0
172.16.0.1               ether   fe:ff:ff:ff:ff:ff   C               eth0
192.168.10.3             ether   02:80:f8:10:25:68   C               eth1
172.16.0.3               ether   fe:ff:ff:ff:ff:ff   C               eth0
```

Can manually add/remove entries in cache

How to Switch?

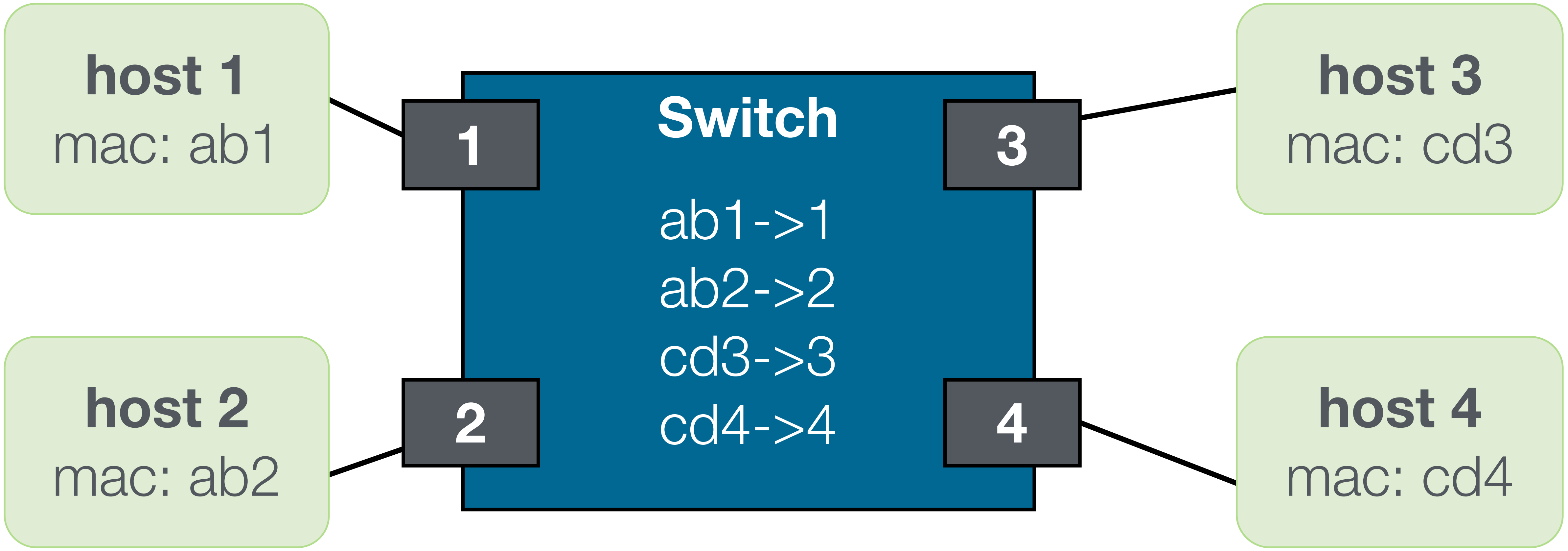
What does a simple switch actually do?



How to Switch?

Learns which MACs are attached to which ports

Broadcasts to unknown MACs and learns replies



POX Switch

Let's make a hard coded switch table

Find the MAC address for each of your nodes

- it's listed in **ifconfig**

Find what port each node is connected to

- an OVS command? output from SuperSimple Pox?

Write rules in **_handle_PacketIn** based on dest mac

- **ff:ff:ff:ff:ff:ff** -> broadcast to all
- a node's MAC -> send out the appropriate port

```
msg = of.ofp_packet_out()
msg.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD))
msg.data = event.ofp
msg.in_port = event.port
self.connection.send(msg)
```

Pox Issues

Let’s learn python and POX!

Lots of new Issues have been posted… try one out!

Put your code in a separate branch in your fork!