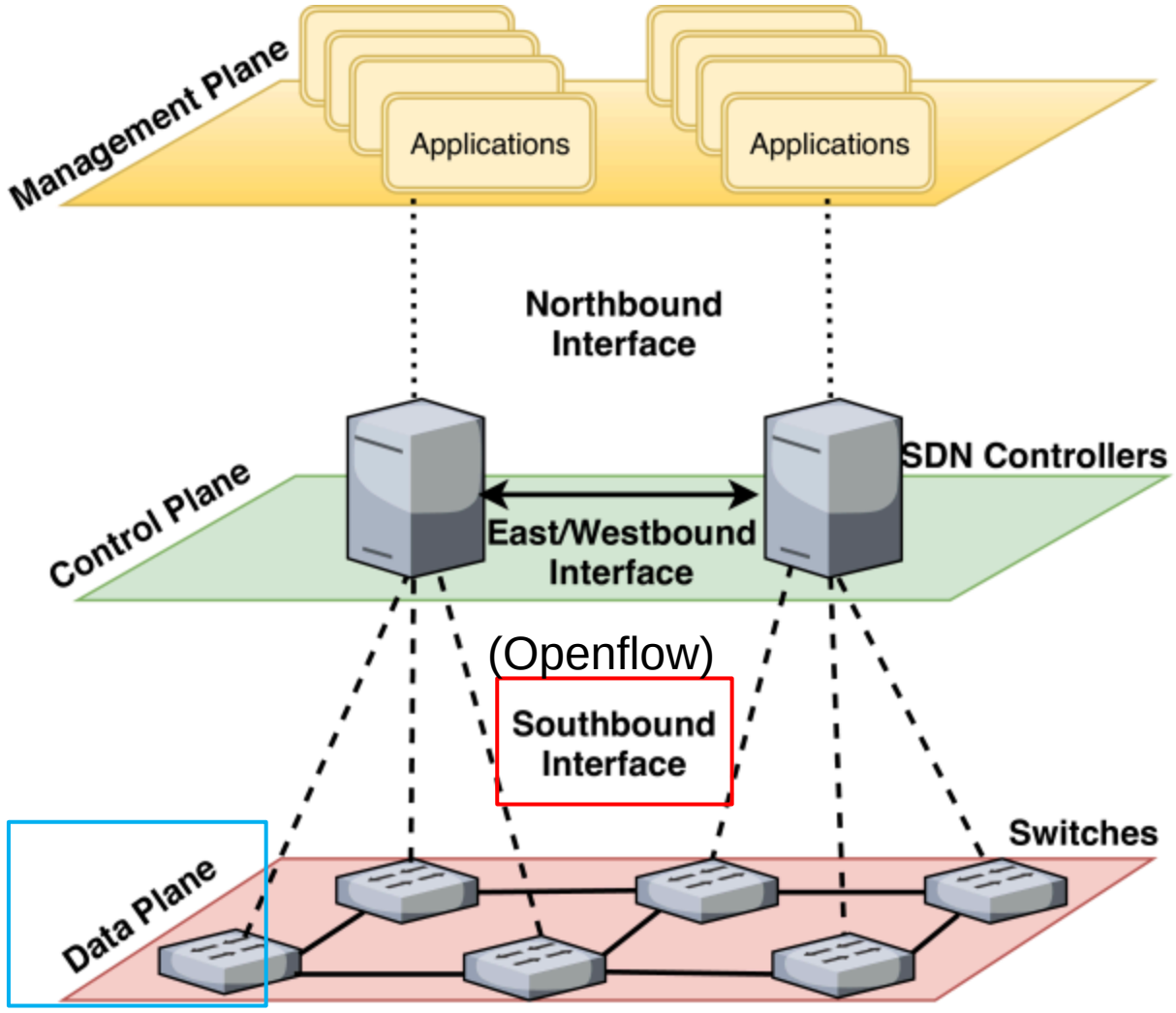
# EL6363 – LAB 3

OpenVSwitch (OVS) and Controllers

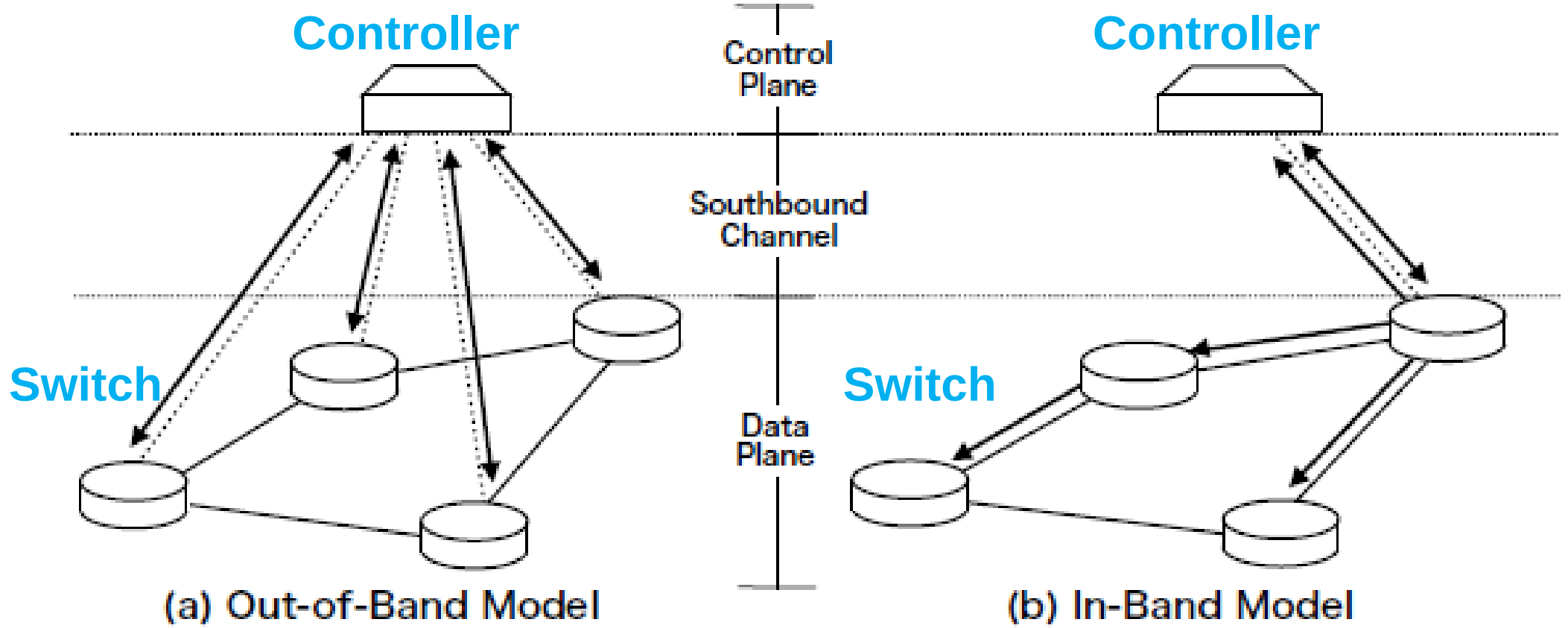
# Overview


Management Plane
Applications
Applications
Northbound
Interface
Control Plane
SDN Controllers
East/Westbound
Interface
(Openflow)
Southbound
Interface
Switches
Data Plane

# Overview

**Separate** network/link for controller switch connections

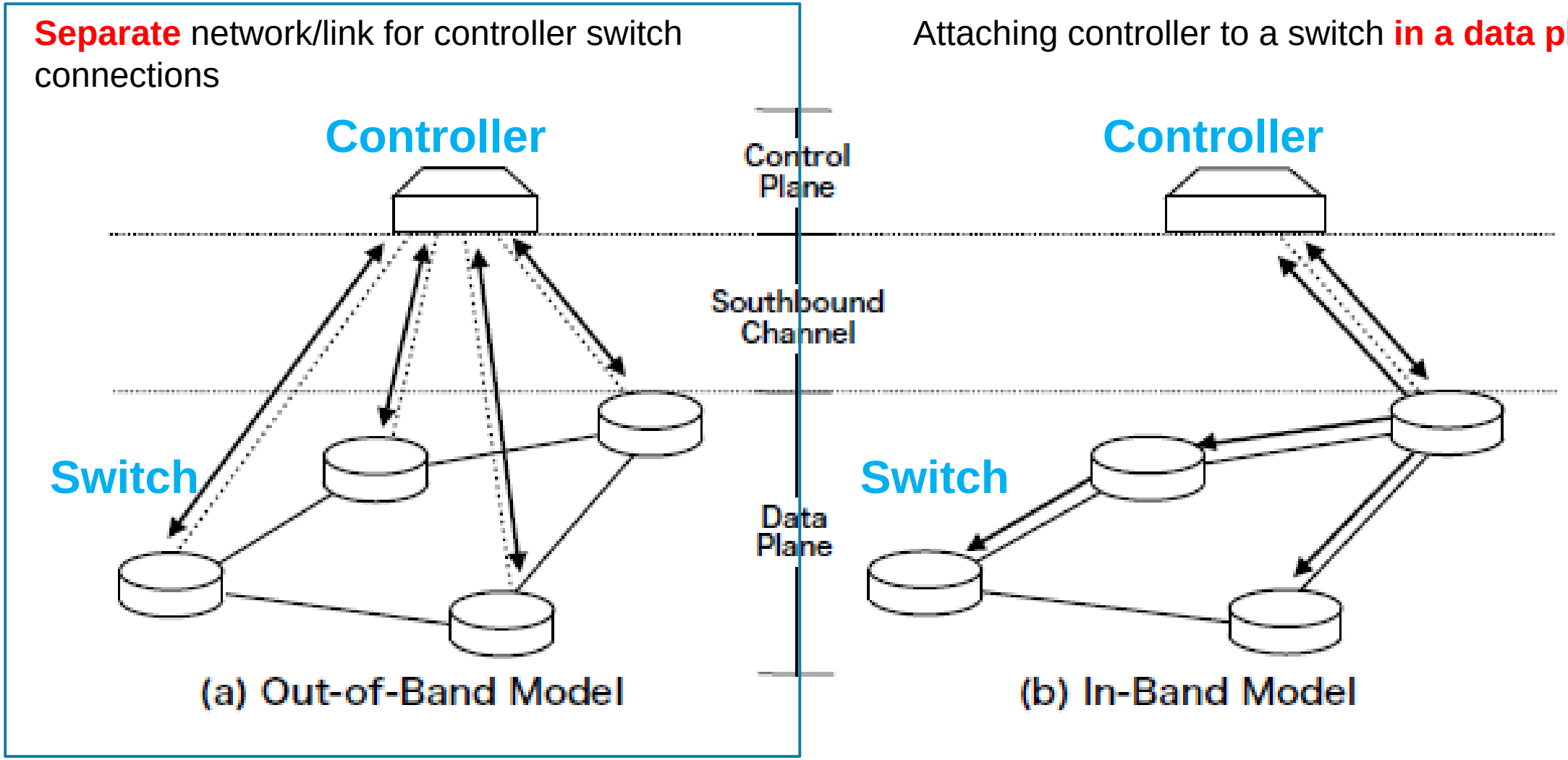
Attaching controller to a switch **in a data plane**



# Overview

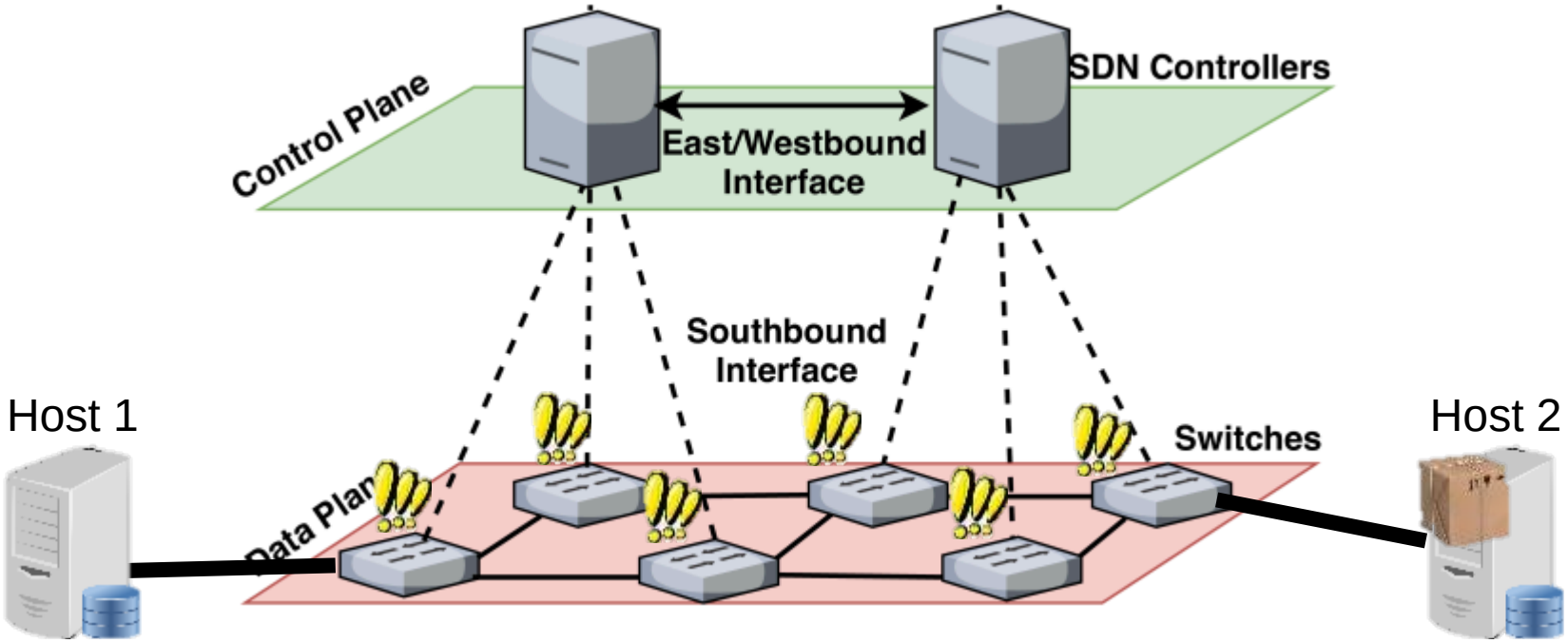
**Separate** network/link for controller switch connections

Attaching controller to a switch **in a data plane**



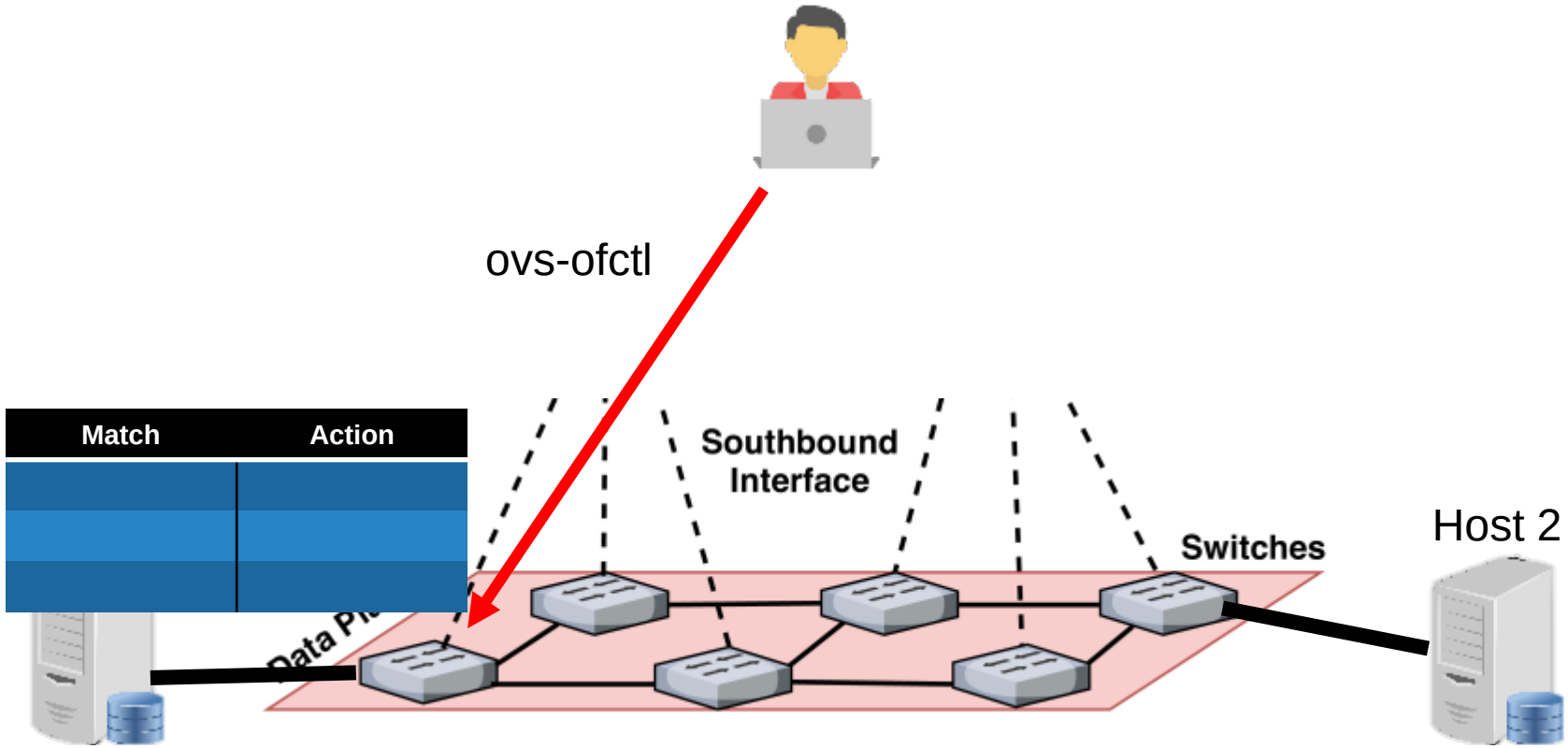
# Overview

**PktIn**: Packet-in
**PktOut**: Packet-out
**FlowMod**: flow entry modification



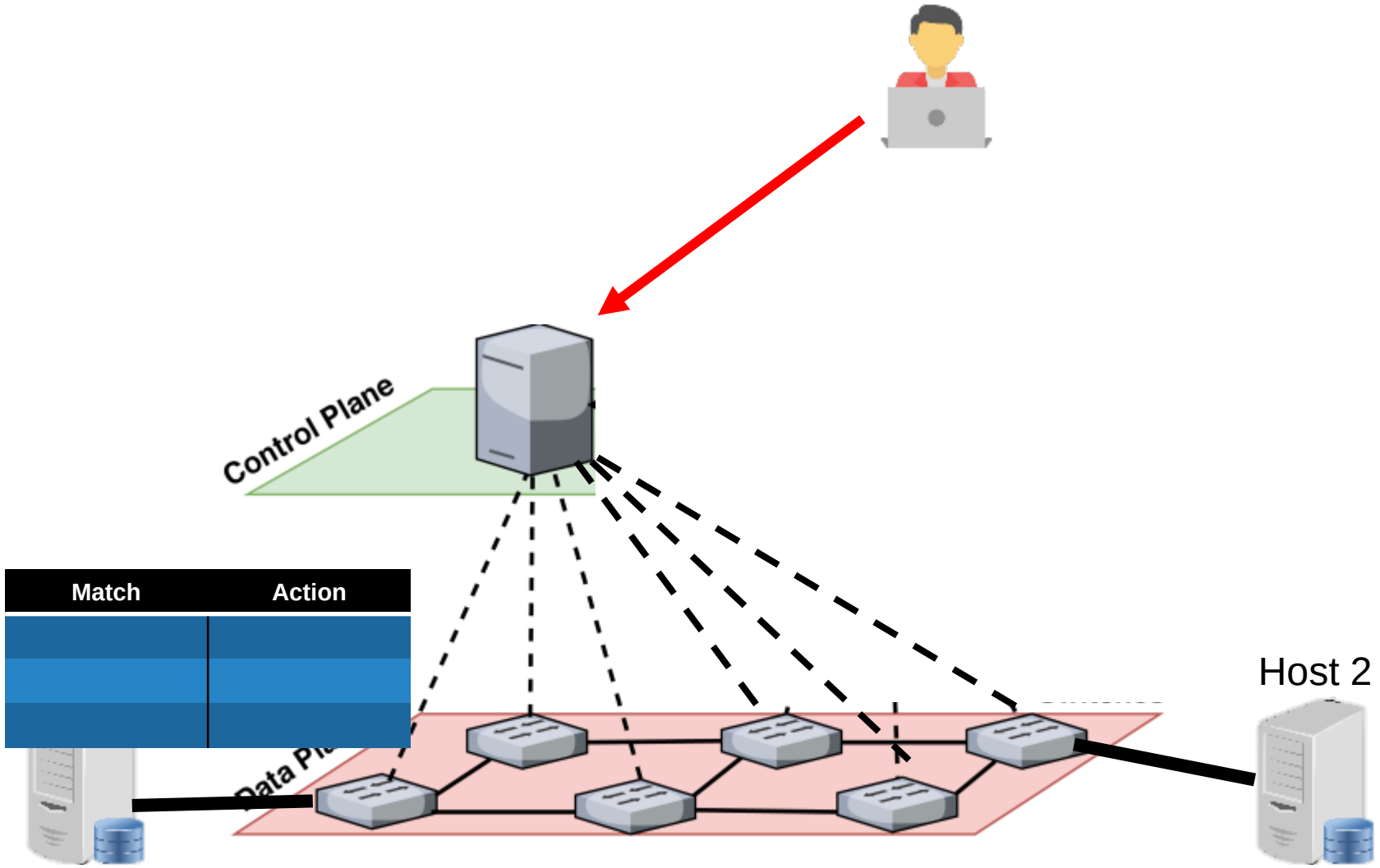
# In Lab 2

We use OpenFlow command to control OVS.
(set up flow entries)

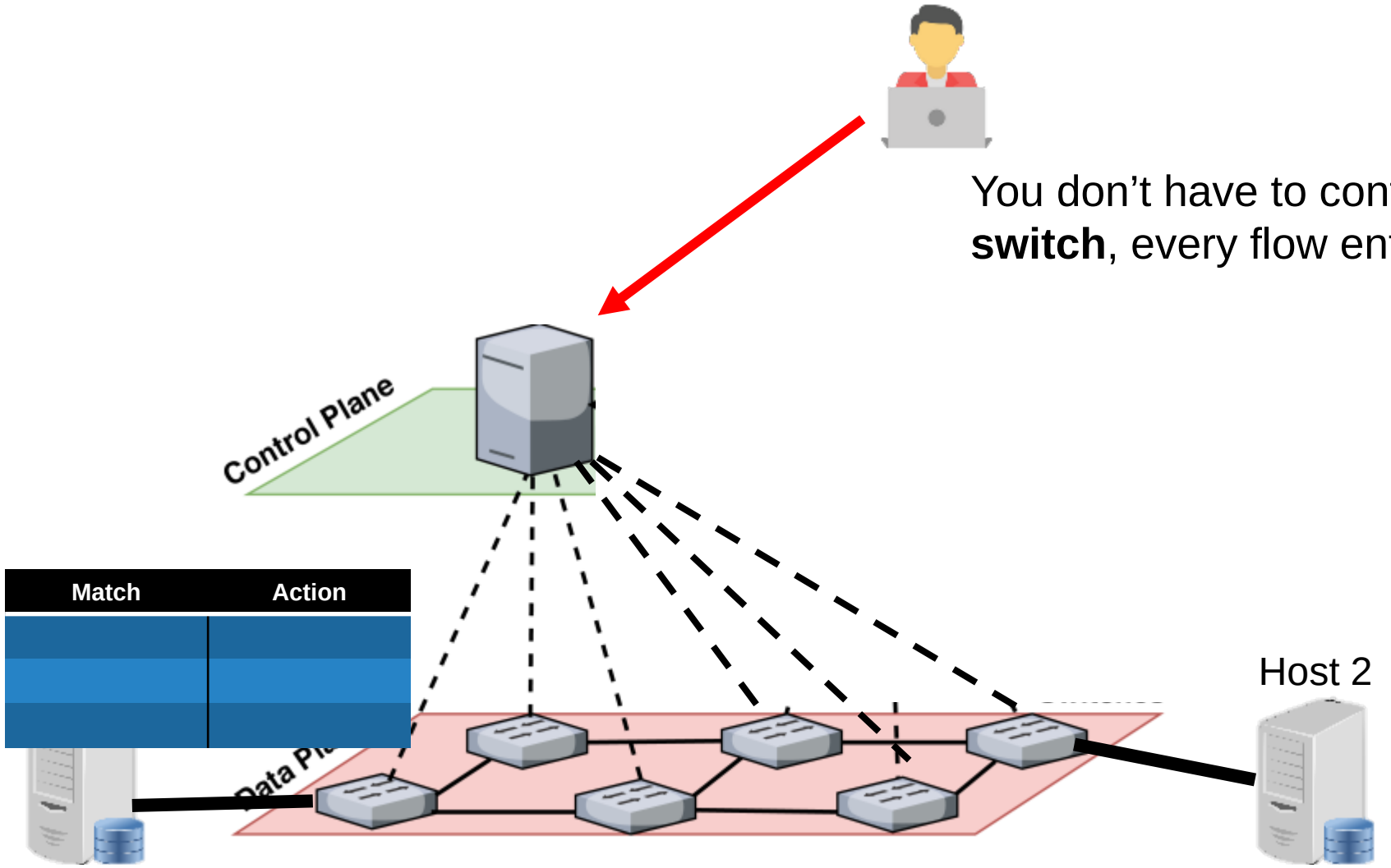


# In Lab 3

We use a controller control OVS.
(set up flow entries)

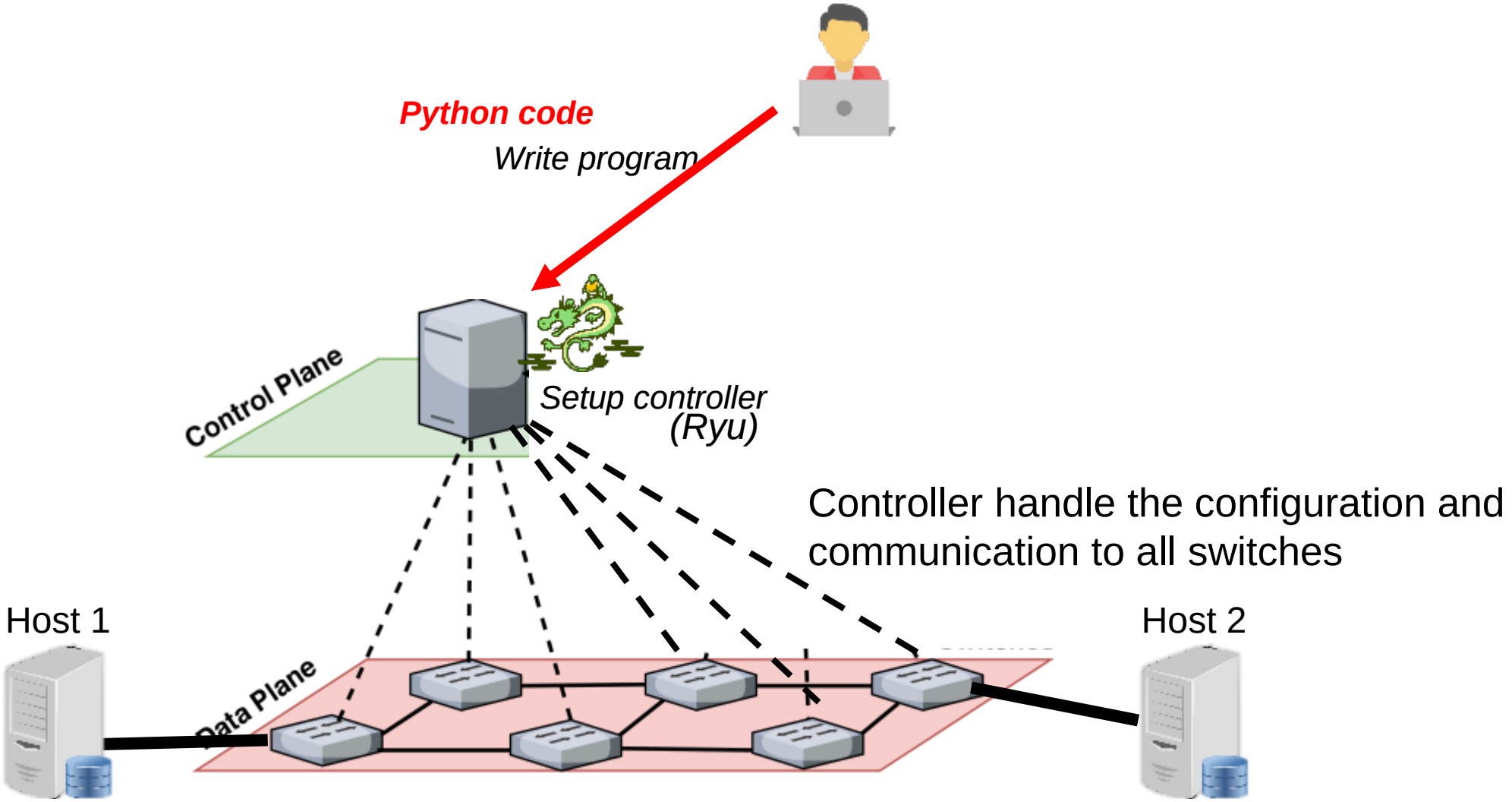


# In Lab 3




You don’t have to configure **every switch**, every flow entry yourself!

# In Lab 3


Python code
Write program
Control Plane
Setup controller
(Ryu)
Controller handle the configuration and
communication to all switches
Host 1
Host 2
Data Plane

# In Lab 3

## Controller choice

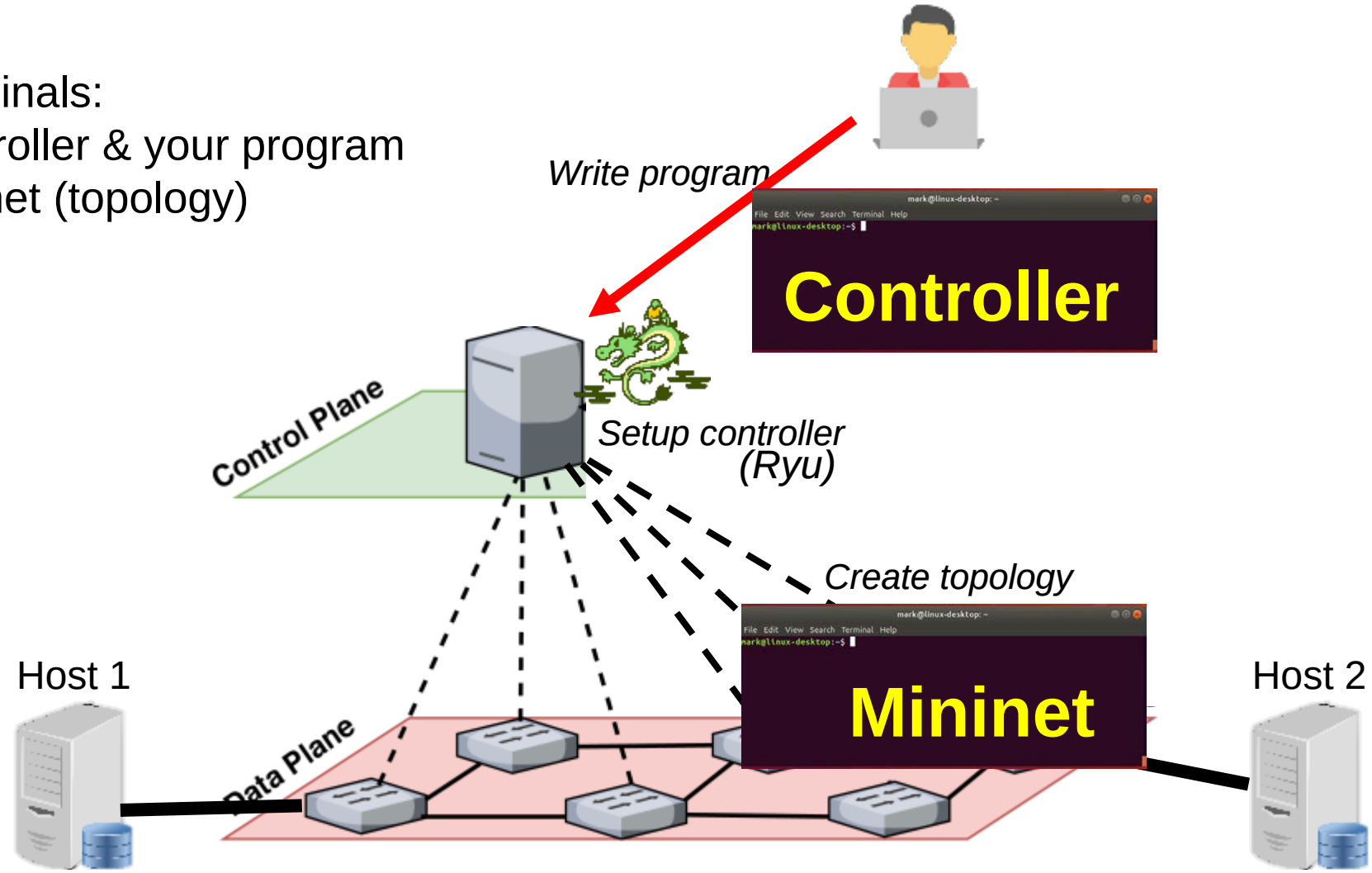
- There are many choice for writing the controller,
  - RYU (I will introduce this)
  - Beacon
  - POX
  - NOX
  - etc…

guration and
hes

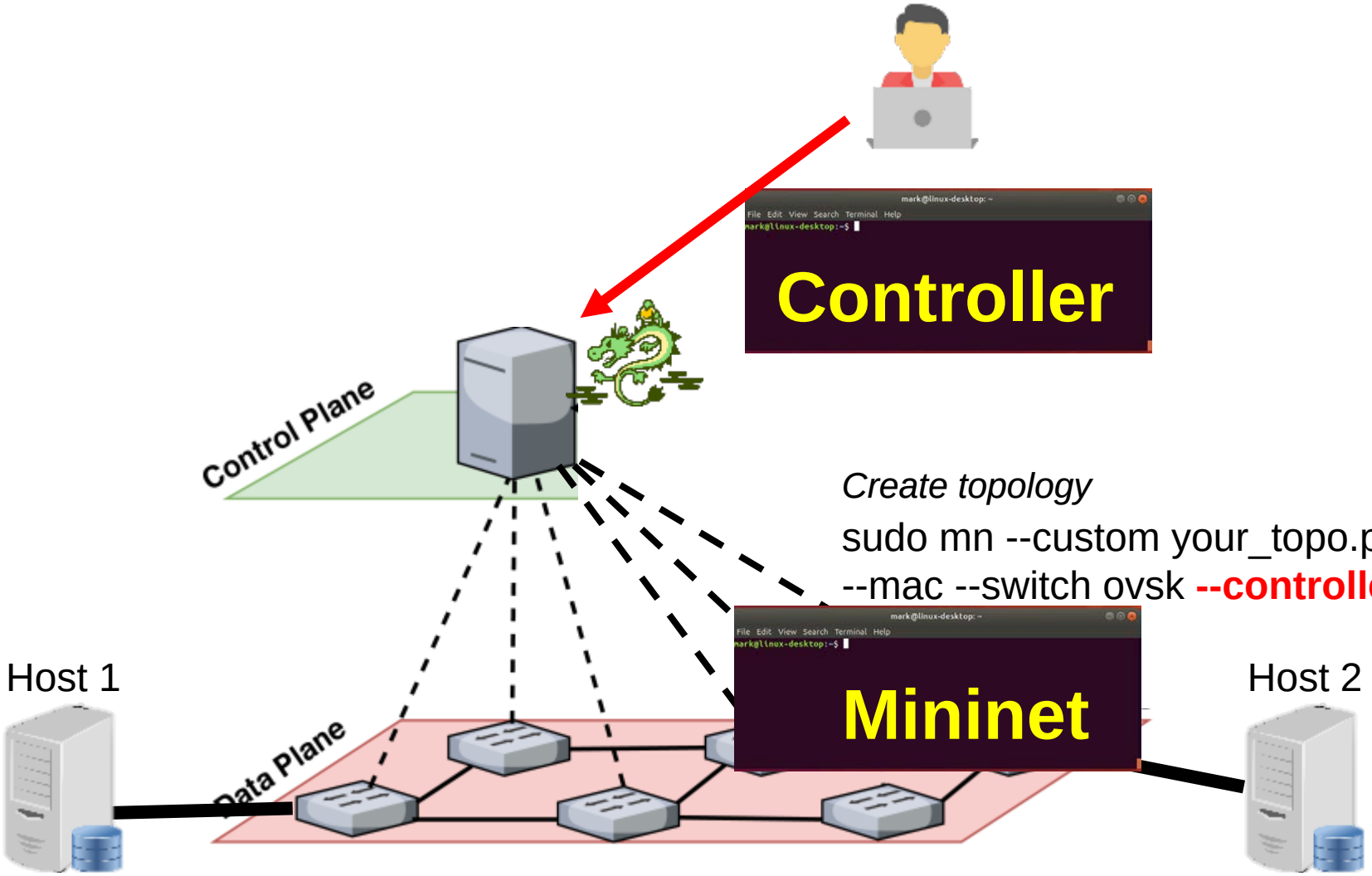
# In Lab 3

Two terminals:

1. Controller & your program
2. Mininet (topology)



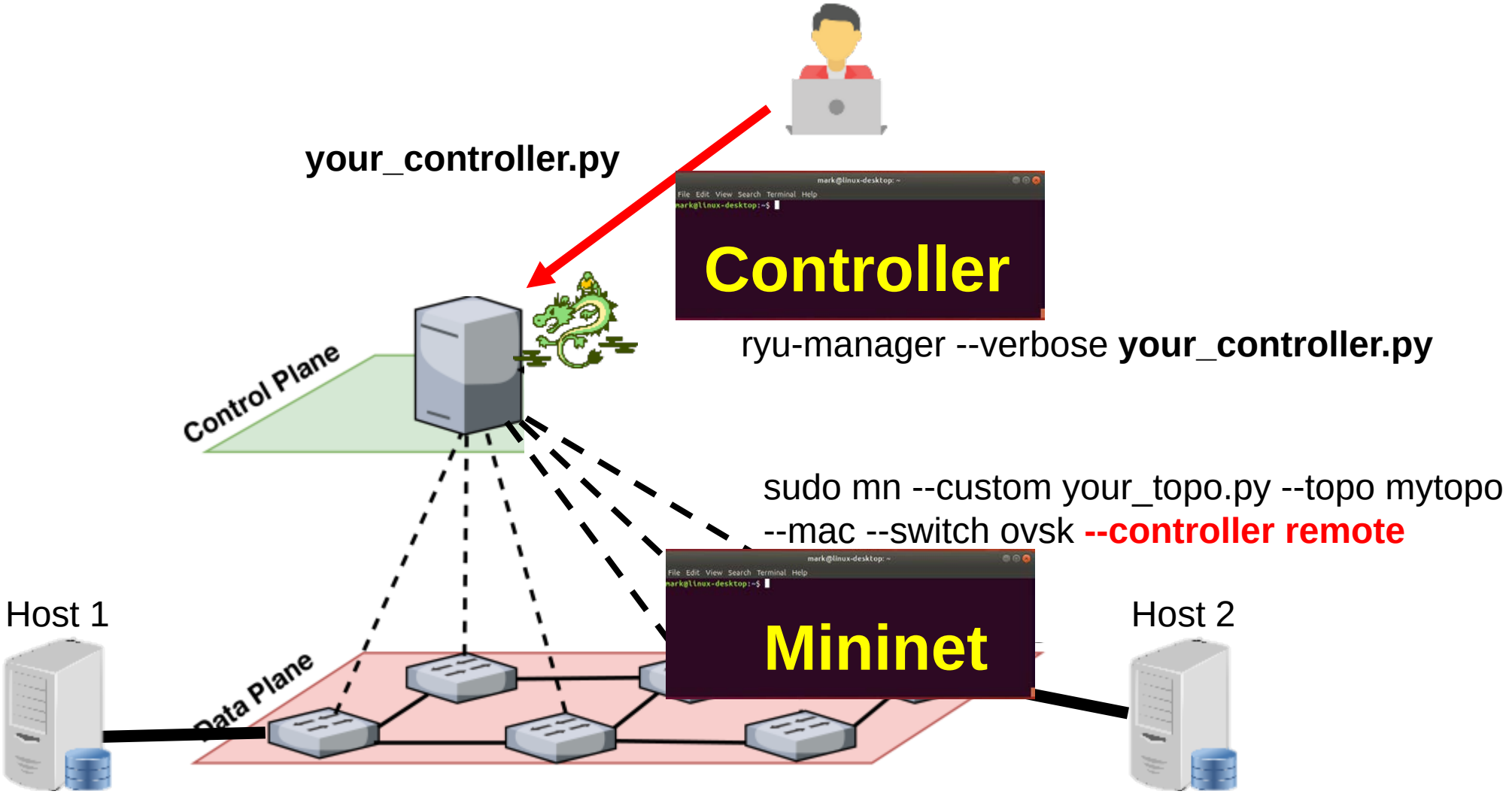
# In Lab 3



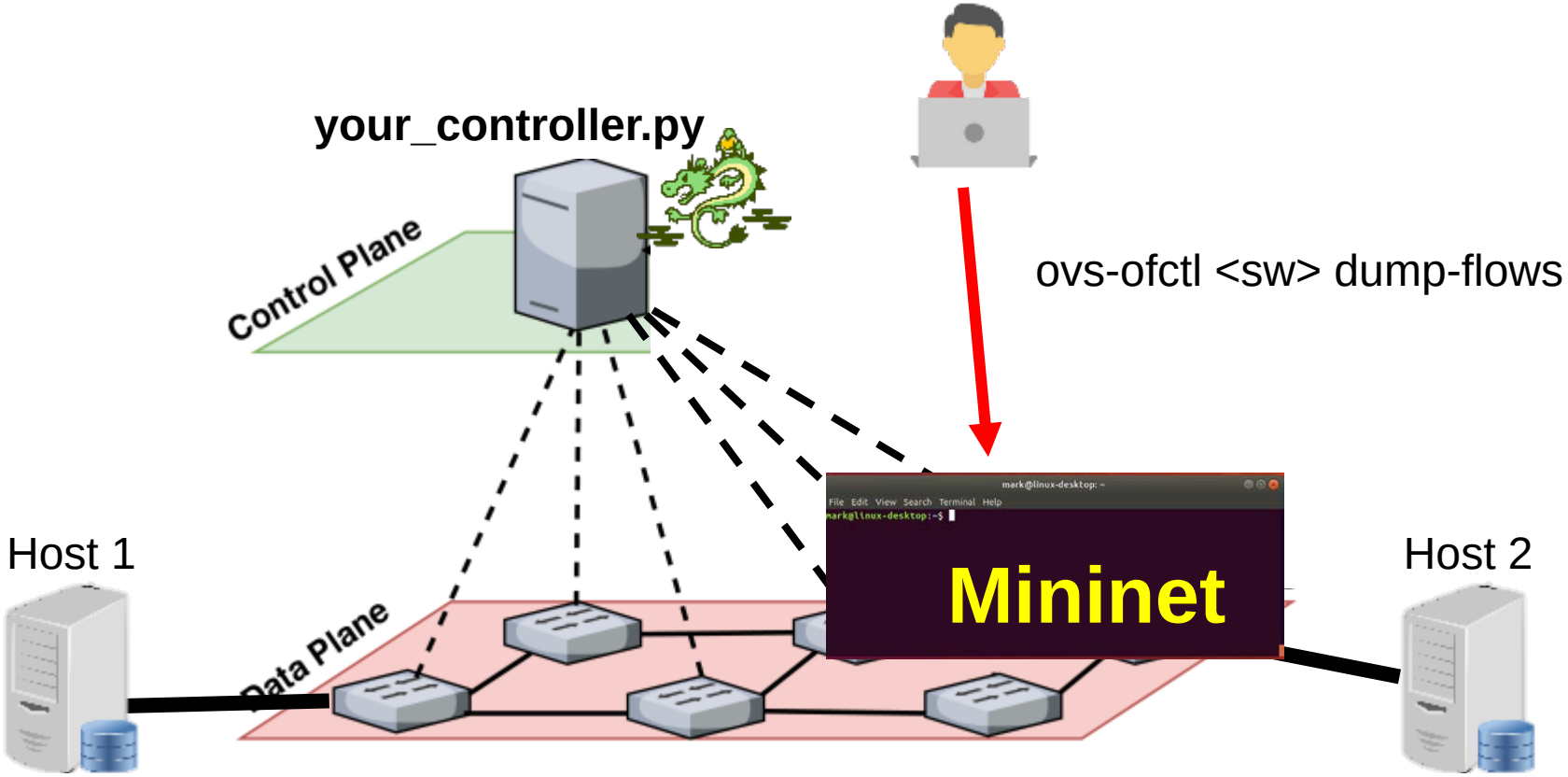

*Create topology*

sudo mn --custom your_topo.py --topo mytopo --mac --switch ovsk **--controller remote**

# In Lab 3


your_controller.py
Controller
ryu-manager --verbose your_controller.py
Control Plane
sudo mn --custom your_topo.py --topo mytopo
--mac --switch ovsk --controller remote
Mininet
Host 1
Host 2
Data Plane

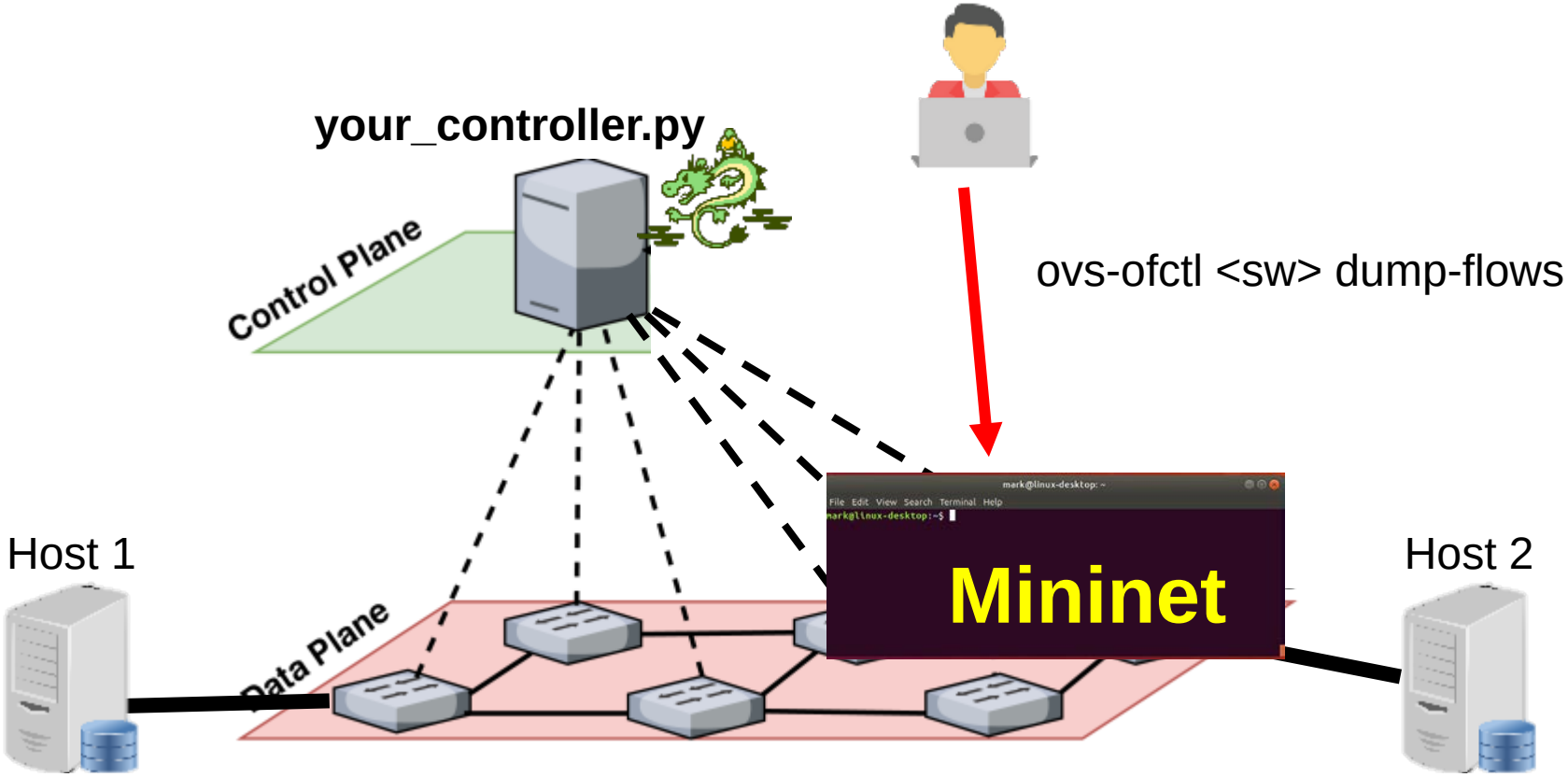
# Debugging And Verifying



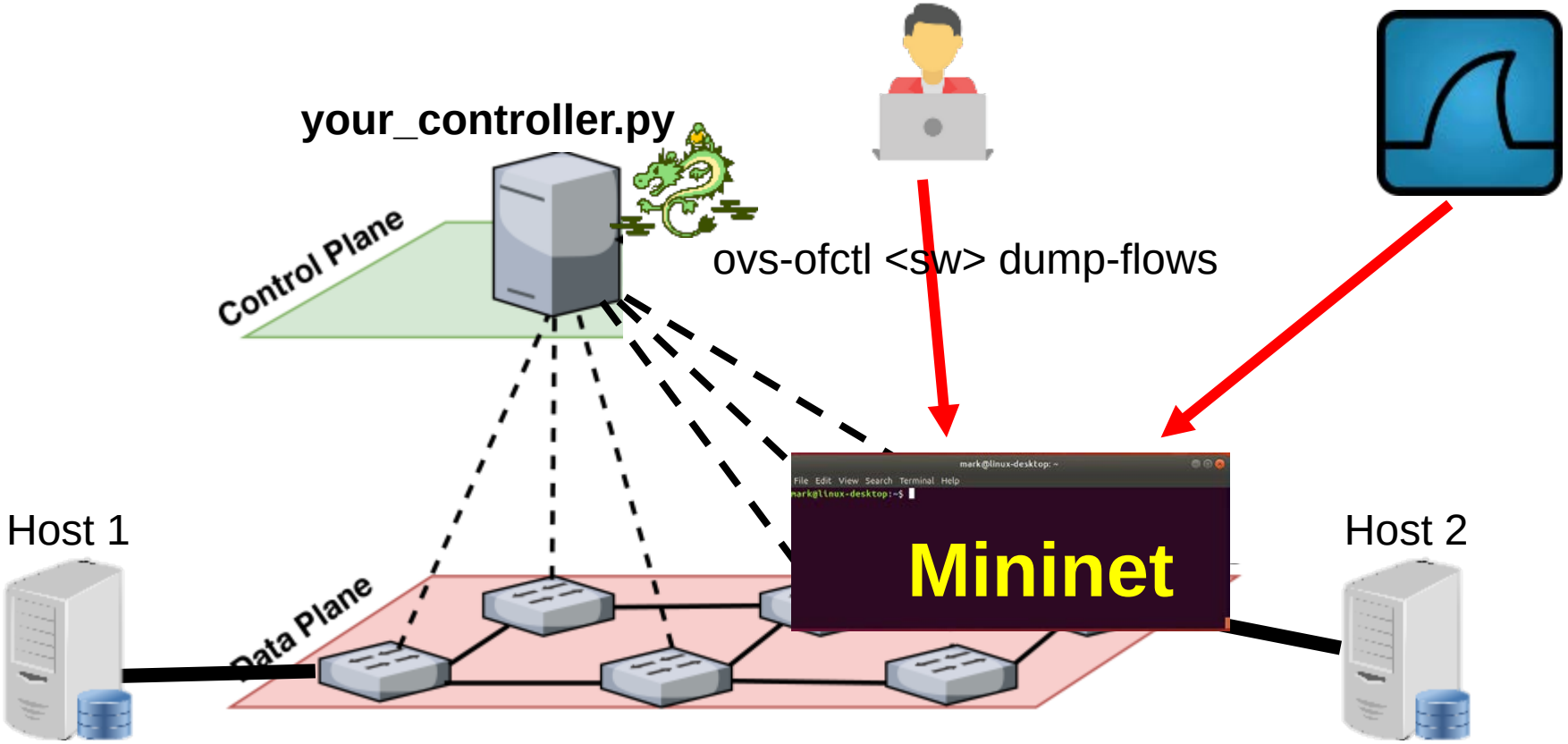
# Debugging And Verifying

```
 cookie=0x0, duration=839.122s, table=0, n_packets=5, n_bytes=490, priority=2,
icmp,dl_dst=10:00:00:00:00:02 actions=output:"s1-eth2"
 cookie=0x0, duration=2112.865s, table=0, n_packets=26, n_bytes=1708, priority
=0 actions=CONTROLLER:65535
cia@localhost:~$
```

S1

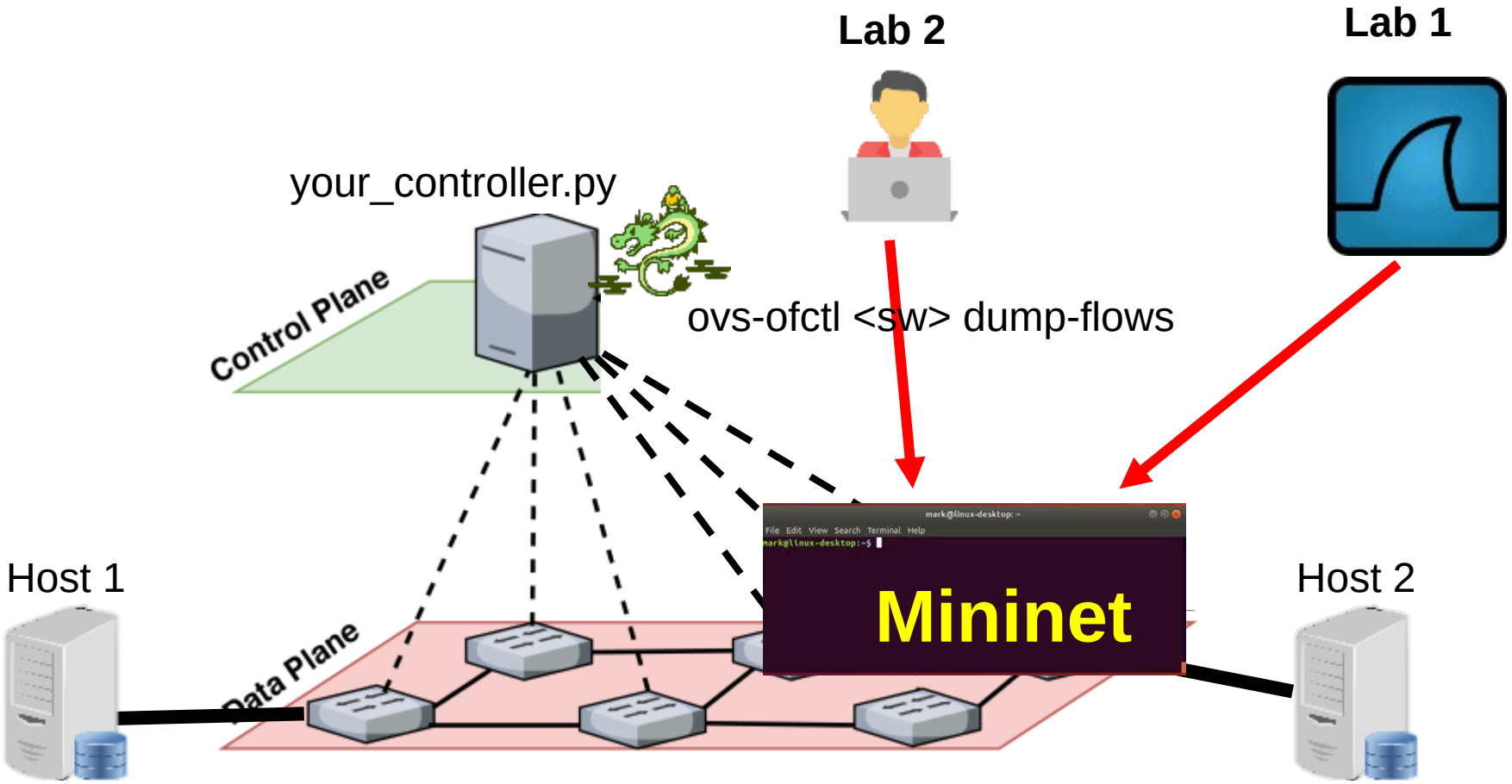


# Debugging And Verifying



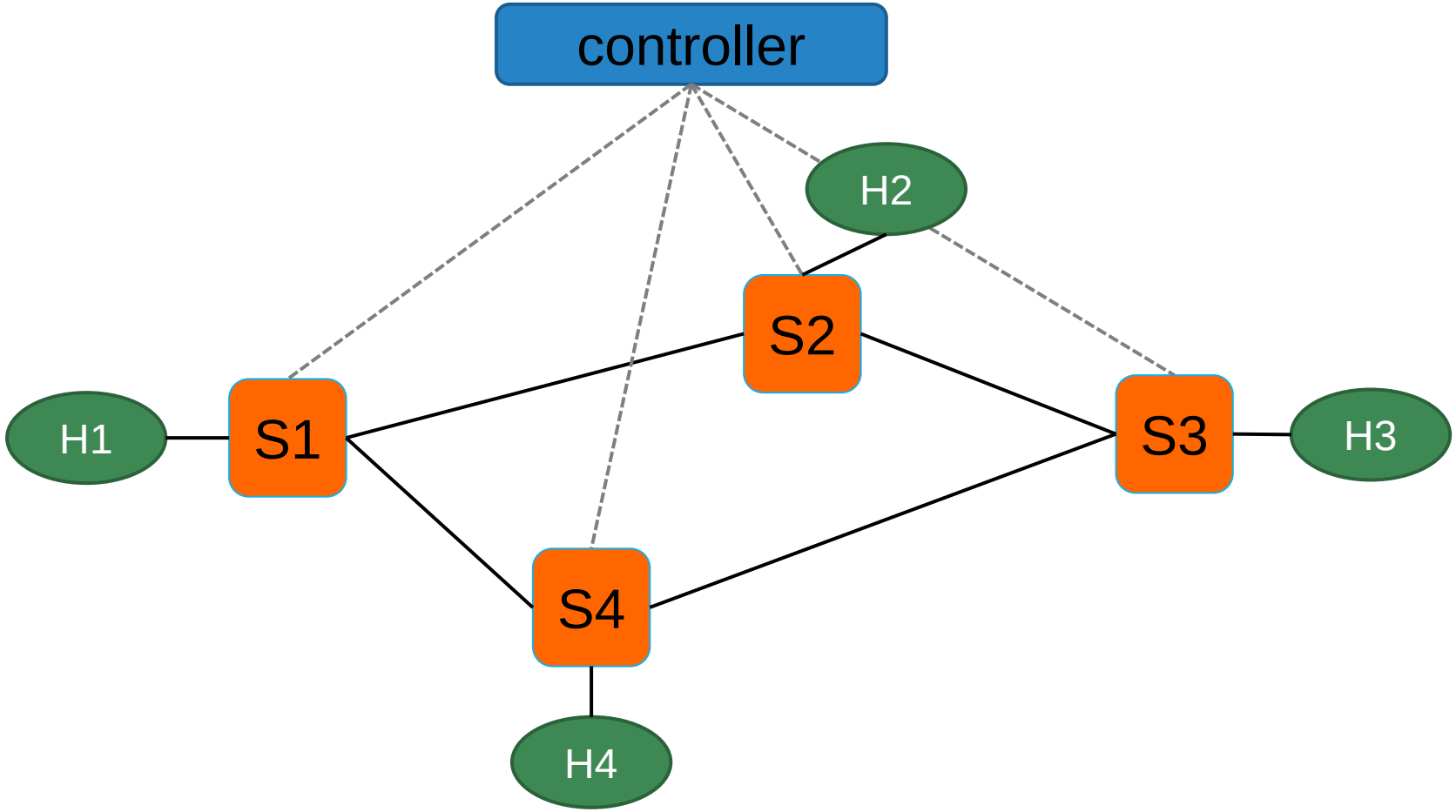
# Debugging And Verifying

Use the technique you learned in Lab 1 and Lab 2!



# Task in this Lab

- 1. Create the following topology using customized topology file

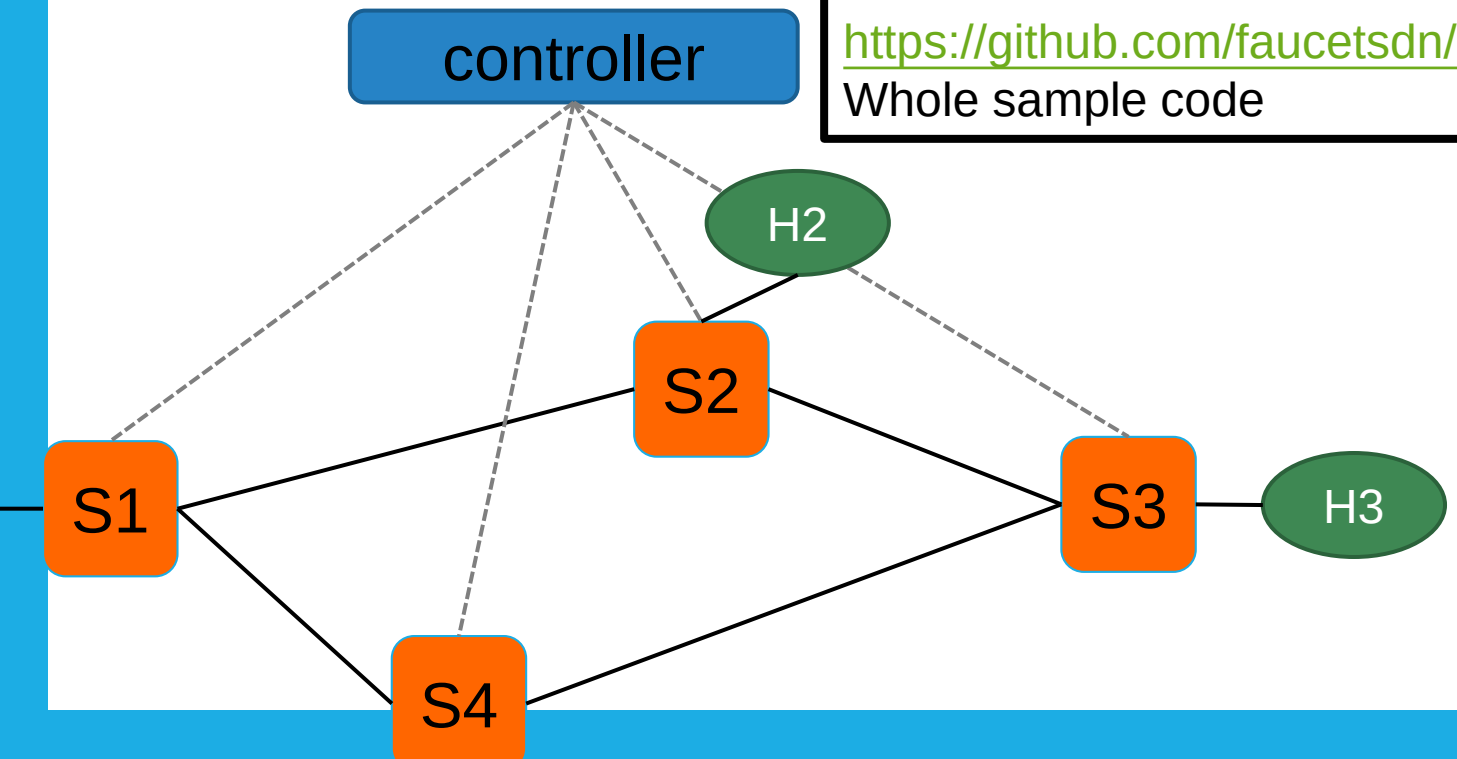


# Task in this Lab

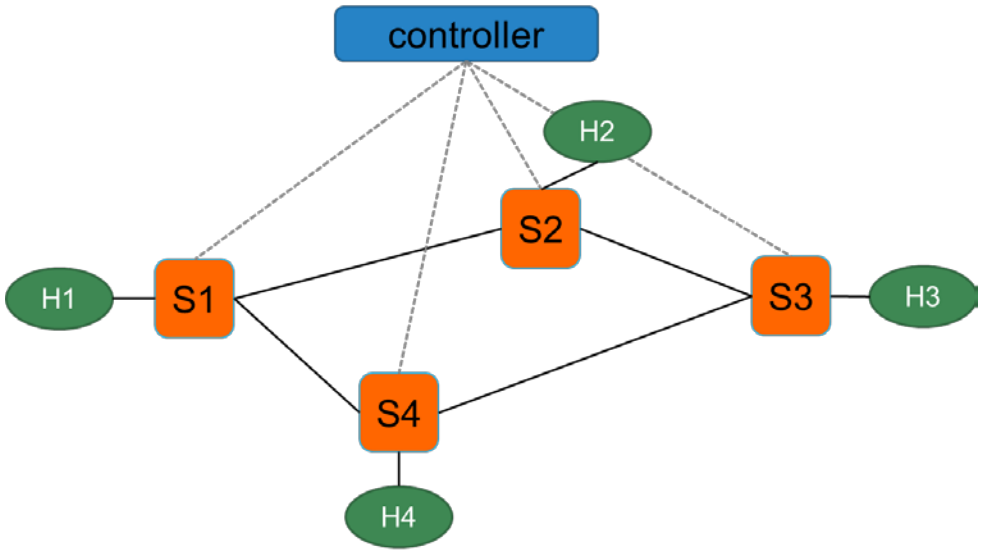
- 2. **Modify the sample code** to enforce the following rules

**Important !!!!**

https://github.com/faucetsdn/ryu/blob/master/ryu/app/simple_switch_13.py
Whole sample code

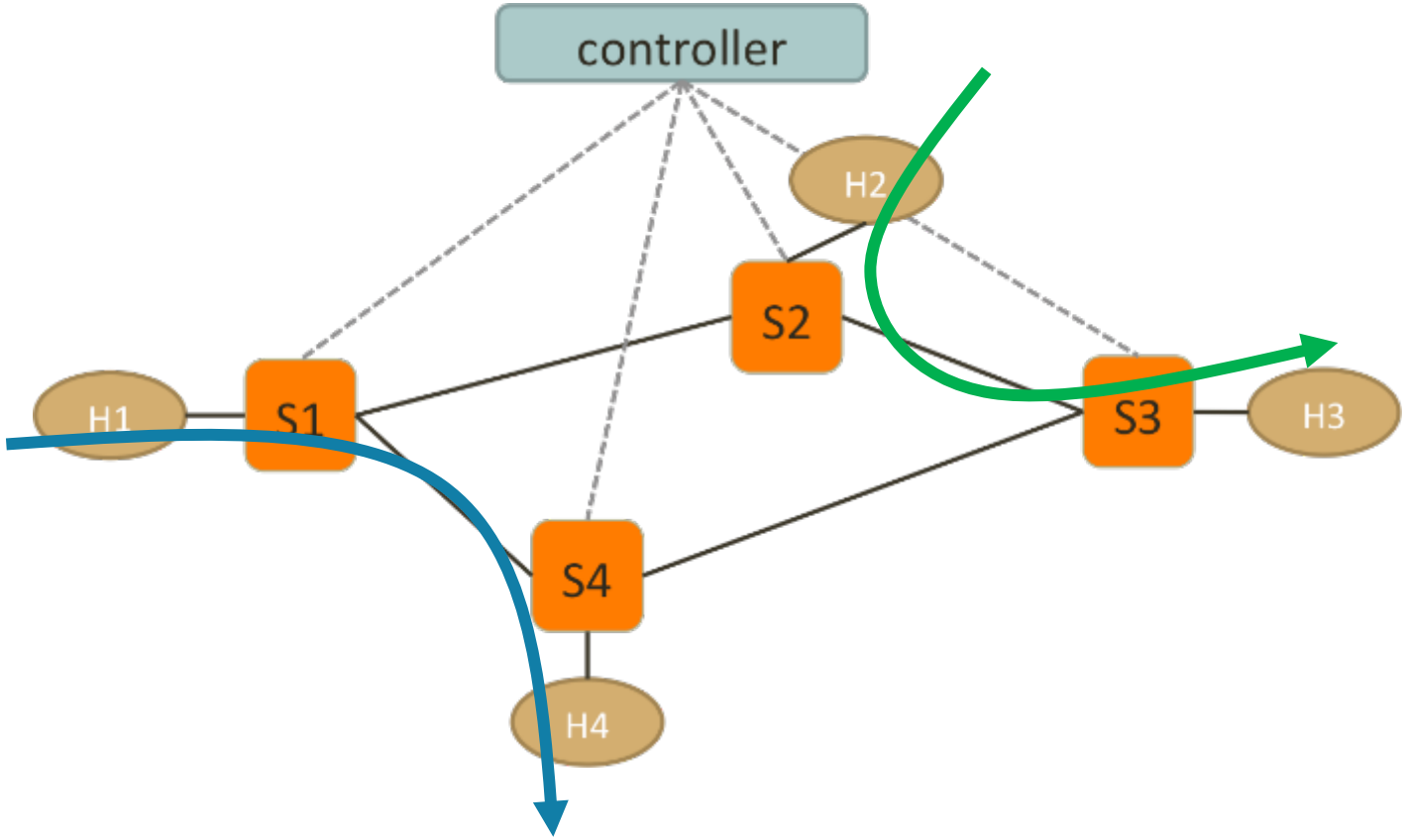


# Task in this Lab

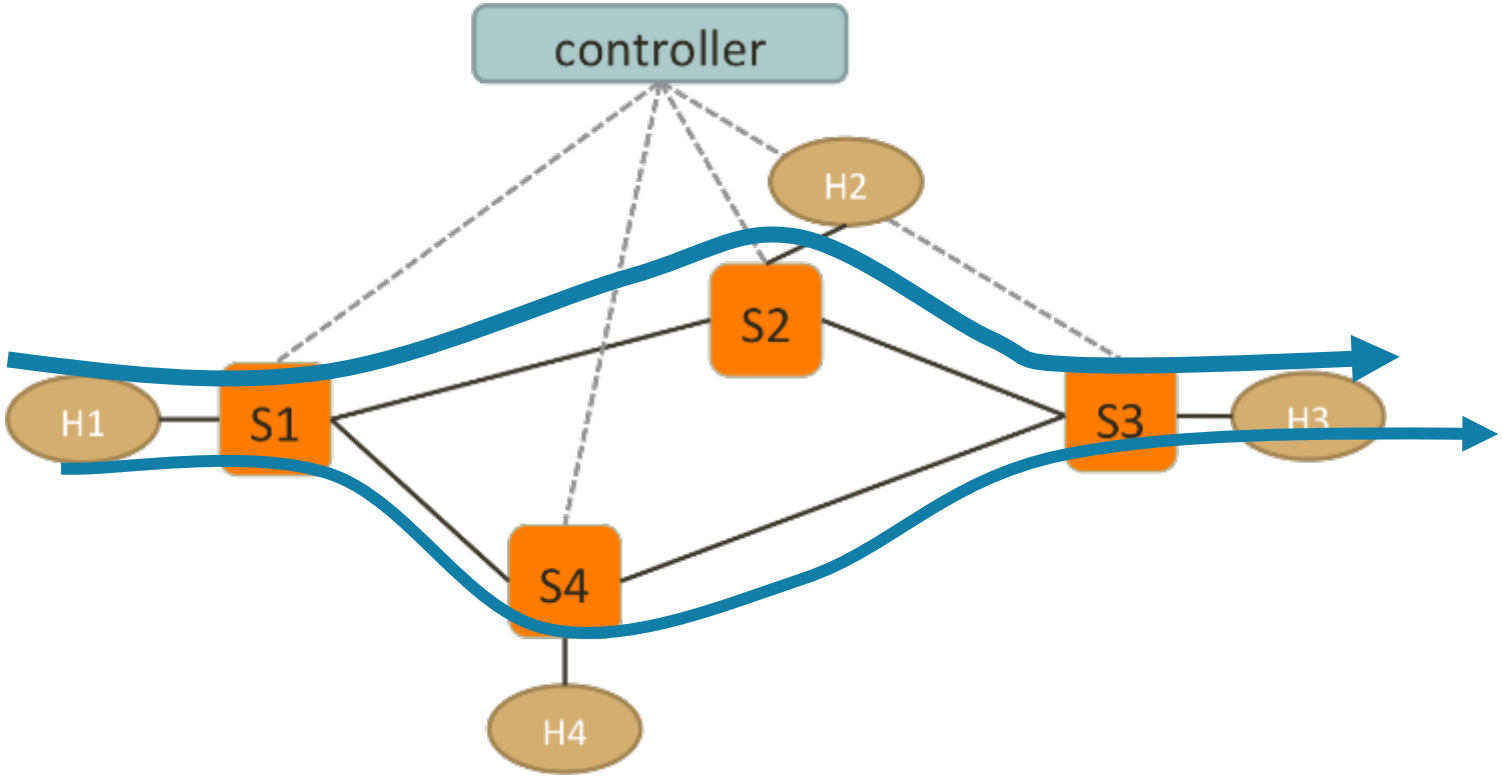



- **Enforce the following rules**
  - Everything follows shortest path
  - **If** there are two shortest paths available
    - ICMP and TCP packets take the clockwise path
      - S1-S2-S3 , S2-S3-S4
    - UDP packets take the counterclockwise path
      - S1-S4-S3 , S2-S1-S4
  - H2 and H4 cannot send HTTP traffic (TCP with dst_port:80)
    - New connections are dropped with a TCP RST sent back to H2 or H4
    - To be more specific, when the first TCP packet (SYN) arrives S2 or S4, forwarded it to controller, controller then create a RST packet and send it back to the host.
  - H1 and H4 cannot send UDP traffic
    - simply drop packets at switches

# 1. Shortest Path



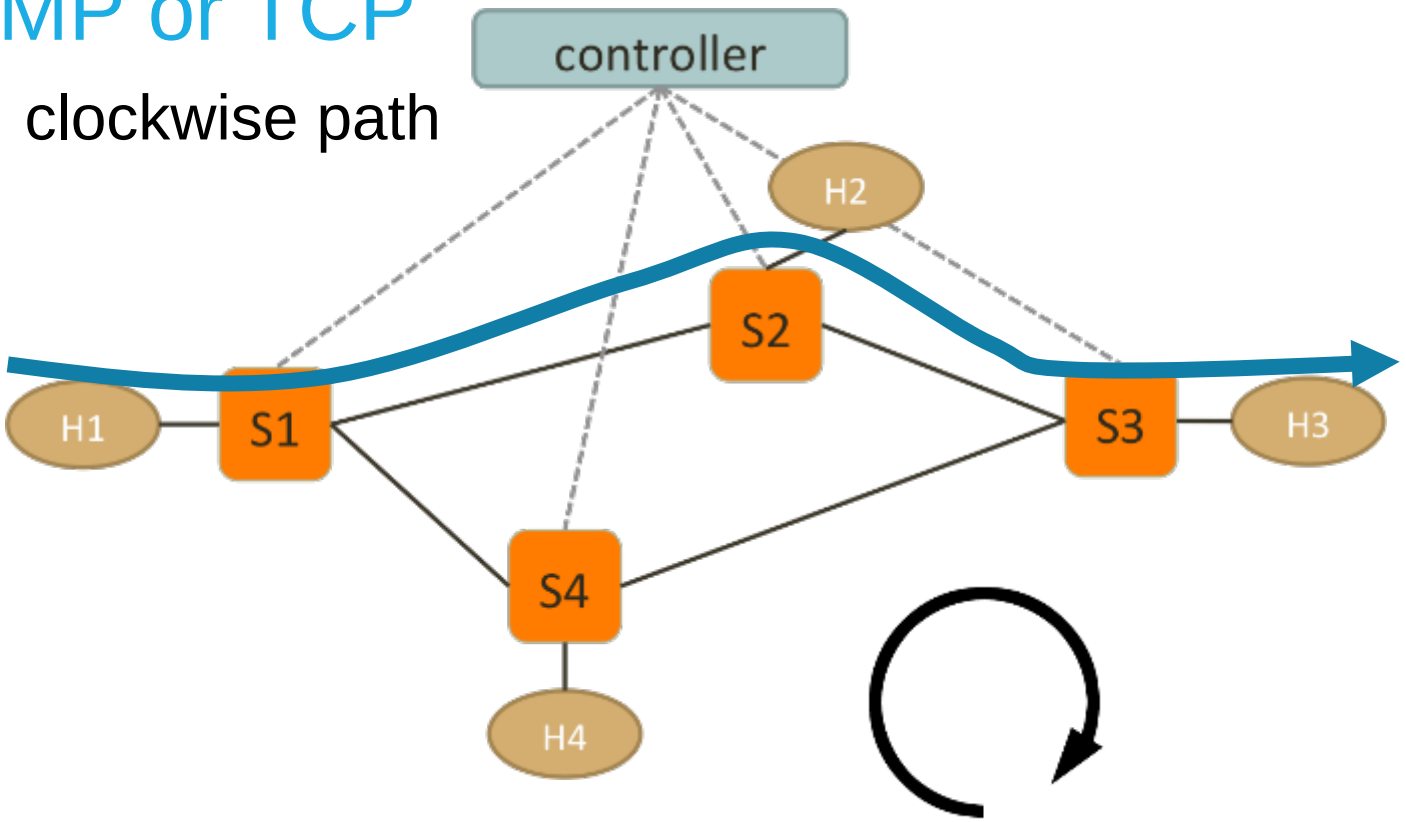
# 2. If two shortest paths



# 2. If two shortest paths

## If ICMP or TCP

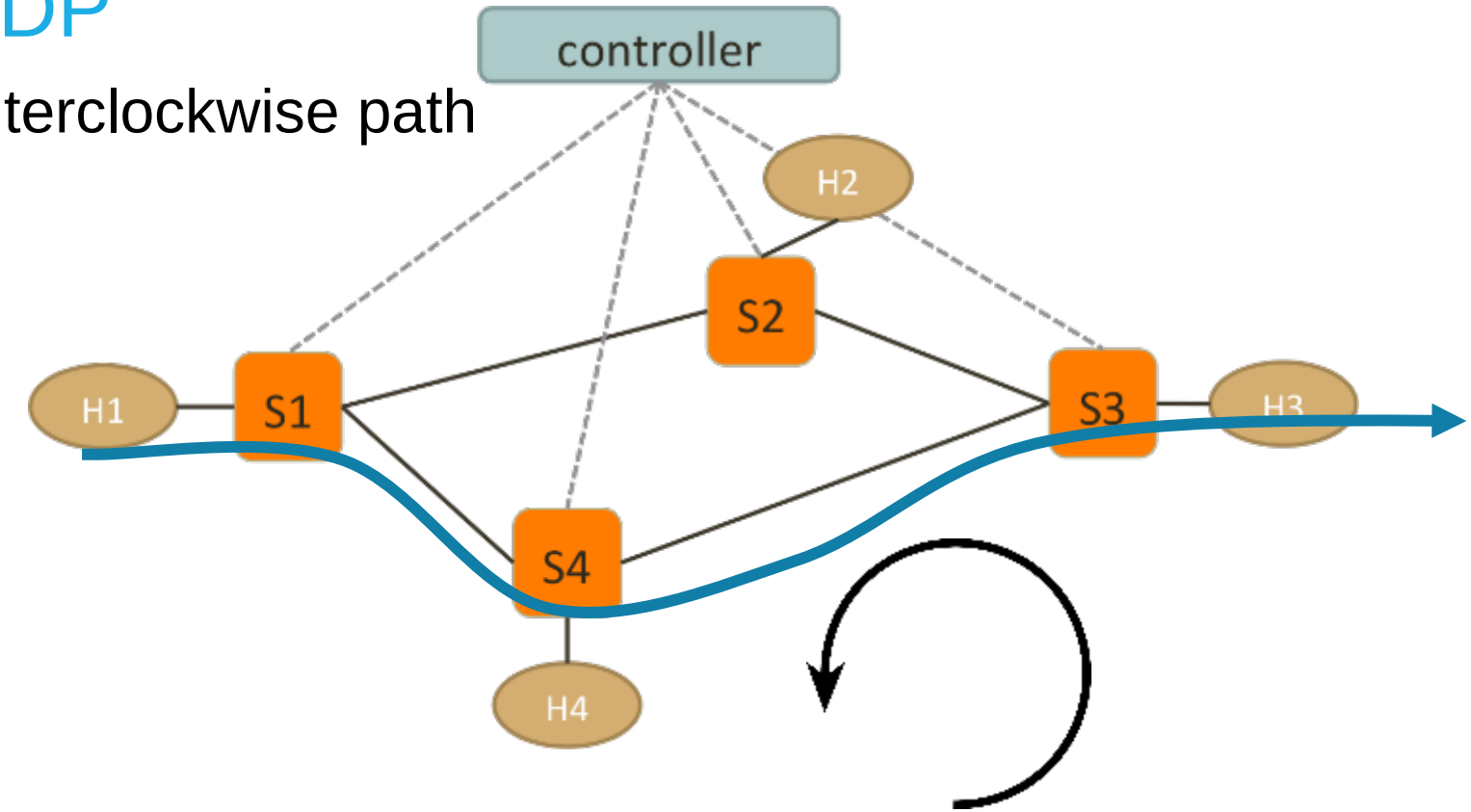
clockwise path



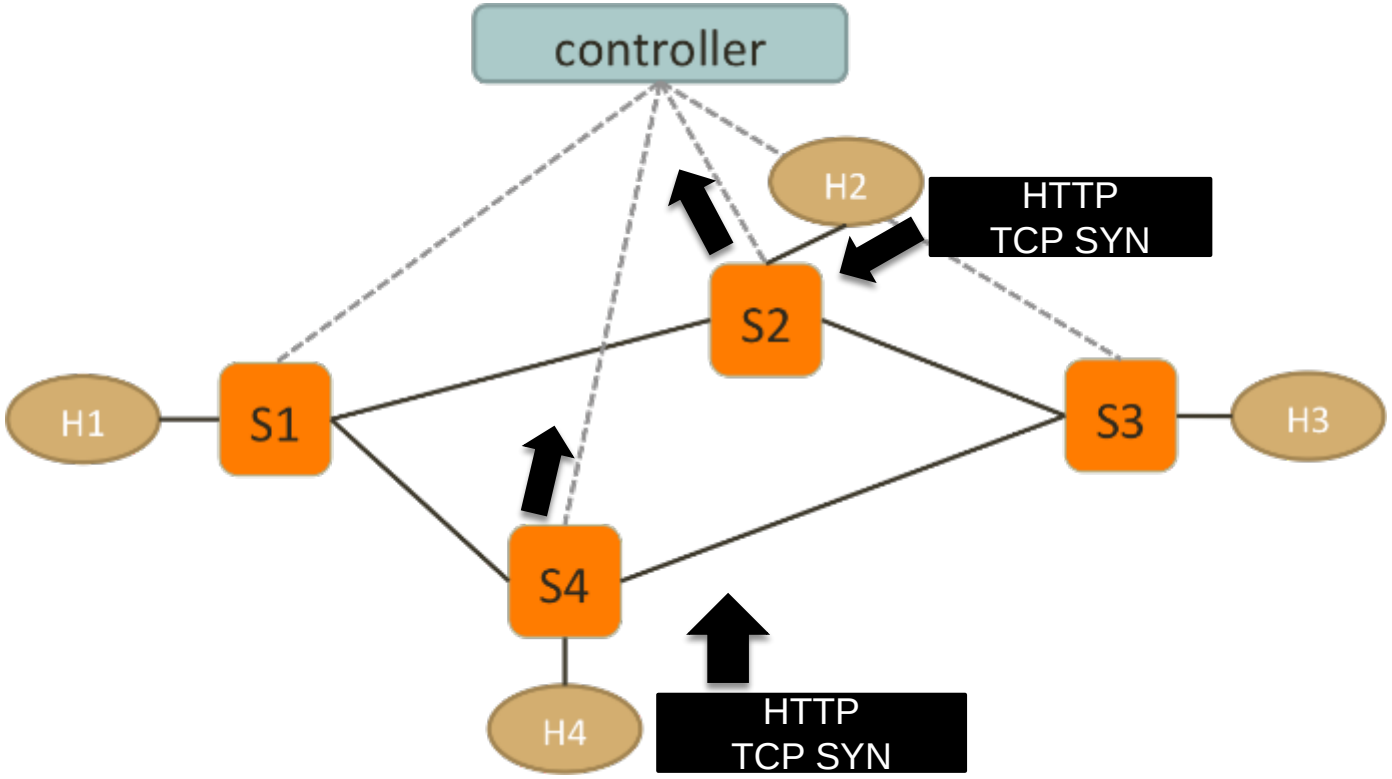
# 2. If two shortest paths

## If UDP

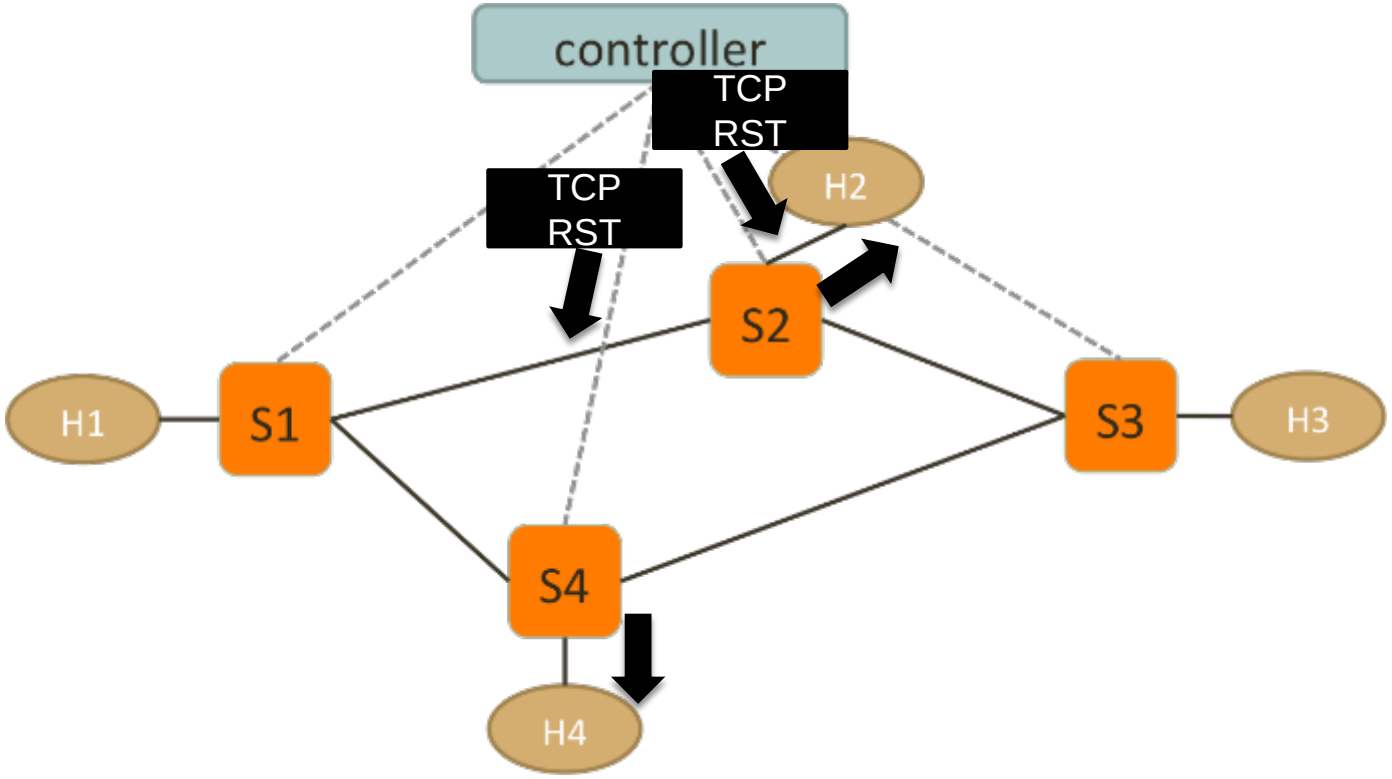
counterclockwise path



# 3. Drop HTTP send from H2 and H4

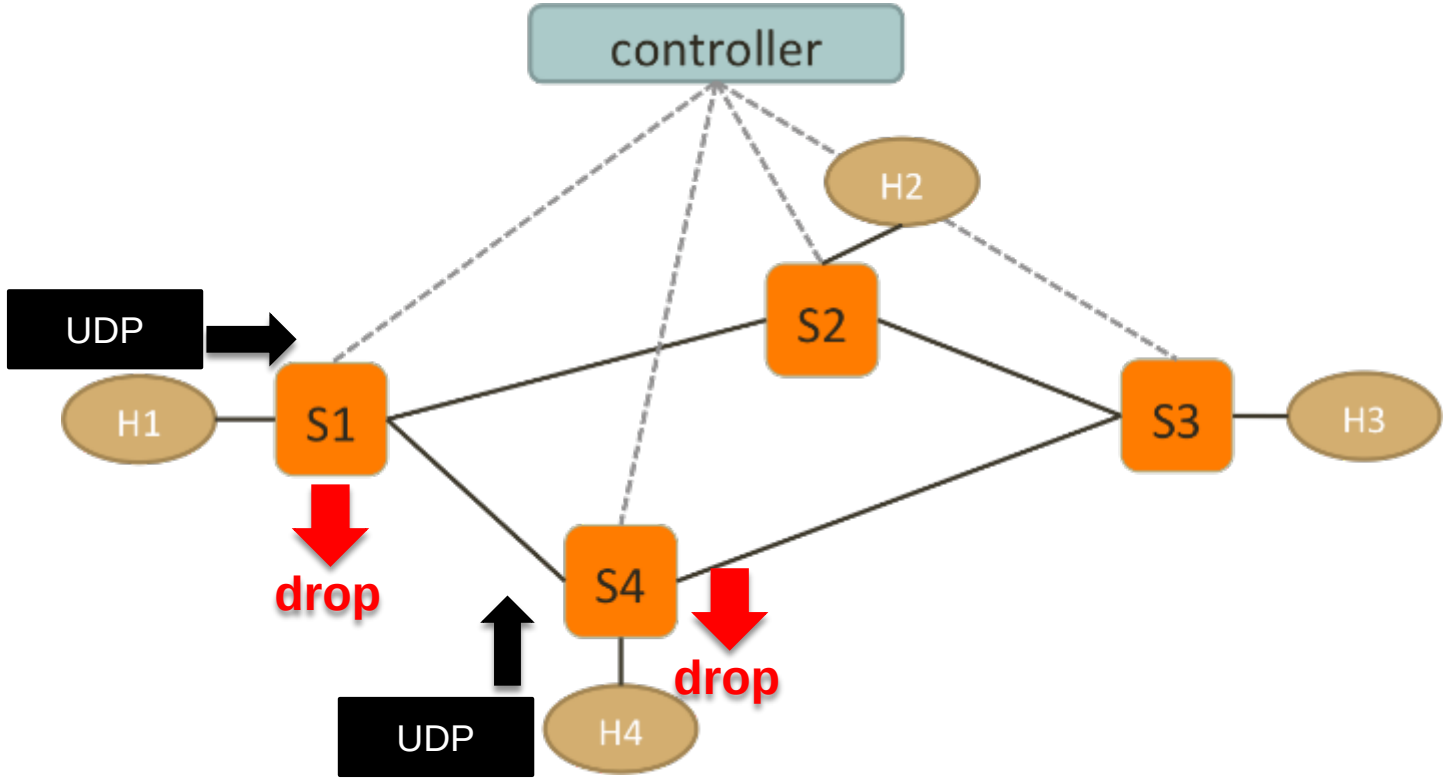


# 3. Drop HTTP send from H2 and H4



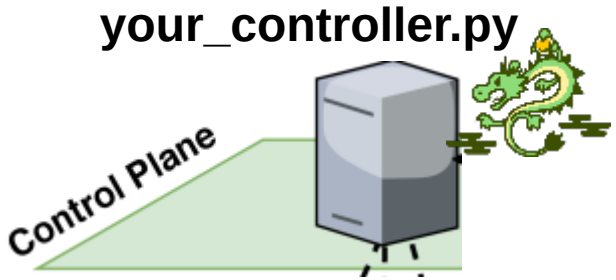

**Send reset packet to reject the connection!**

# 4. Drop HTTP send from H1 and H4

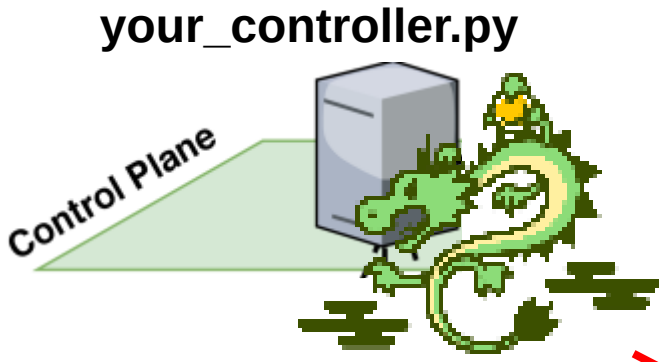


# Go through code

# Programming with Ryu



# Programming with Ryu



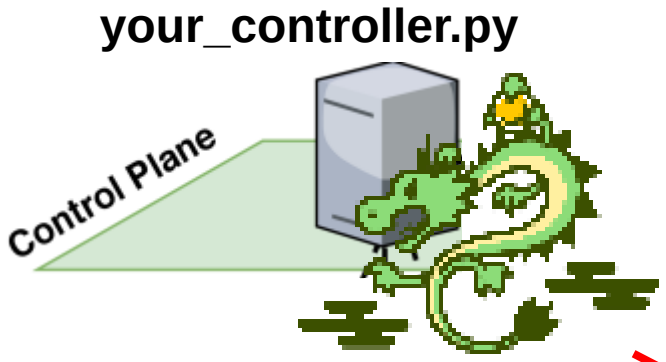

Event

Handle the event

Teach the controller how to handle the event (override the function)

*Your code …*

# Programming with Ryu




Event

- **Switch connected**
- **Packet-in**

**…**

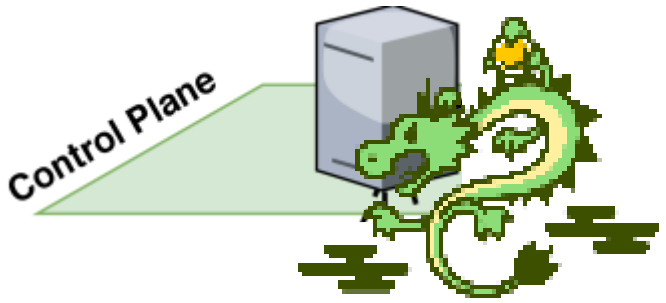
Handle the event

Teach the controller how to handle the event (override the function)

*Your code …*

# Programming with Ryu

**your_controller.py**




Event

**Switch connected**

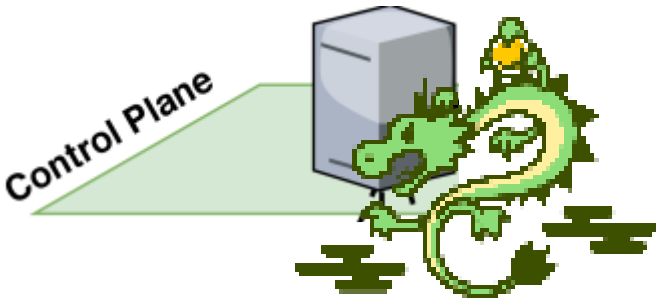
```
@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def switch_features_handler(self, ev):
```

*Your code …*

***Python code***

# Programming with Ryu

**your_controller.py**




Event

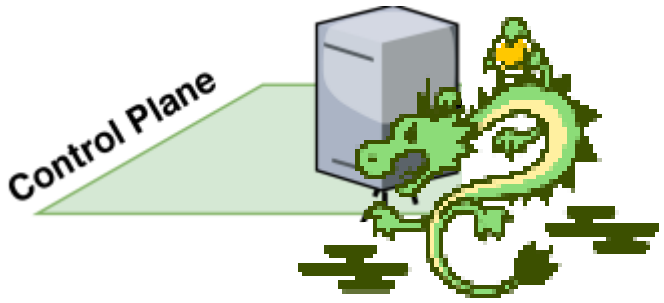
**Packet-in**

```
@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def switch_features_handler(self, ev):
    Your code …

@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
    Your code …
```

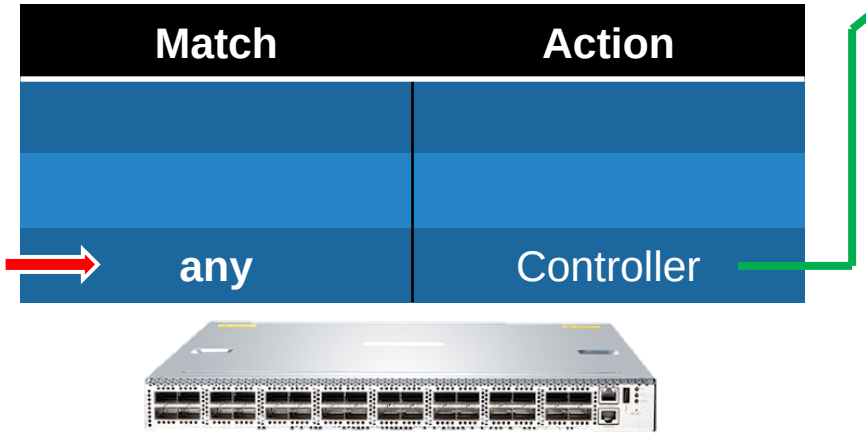
***Python code***

# Programming with Ryu

your_controller.py




Setup table-miss flow

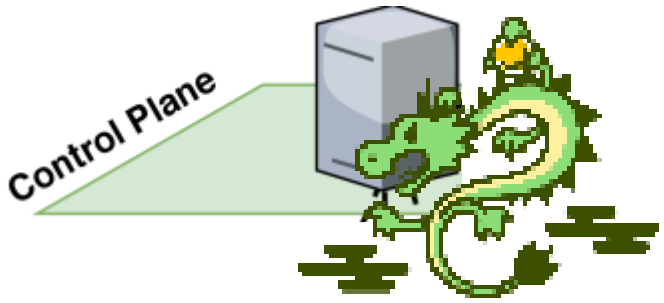



```
@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def switch_features_handler(self, ev):
    Your code …

@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
    Your code …
```

***Python code***

# Programming with Ryu

your_controller.py




```
@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def switch_features_handler(self, ev):

        Your code …


@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
        Your code …
```

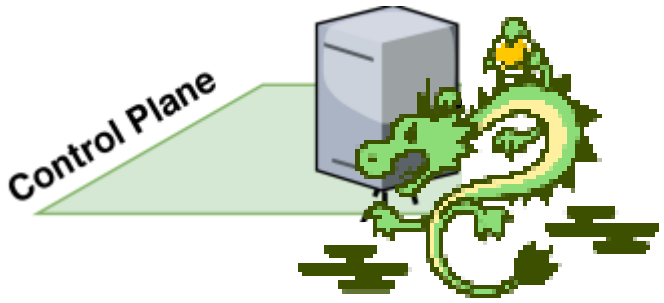
***Python code***

| Match | Action |
|---|---|
| any | Controller |

| Source | Destination |
|---|---|
| IP: 10.10.1.3 | IP: 10.10.1.2 |

# Programming with Ryu

**your_controller.py**




Install flow entry

```
@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def switch_features_handler(self, ev):
        Your code …

@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
        Your code …
```

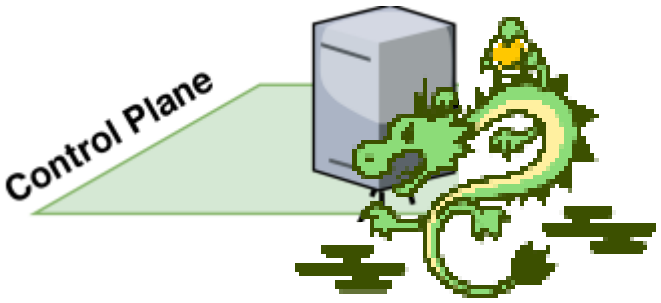
***Python code***

| Match | Action |
|---|---|
| | |
| IP: 10.10.1.3 | output:1 |
| **any** | Controller |

| Source | Destination |
|---|---|
| IP: 10.10.1.3 | IP: 10.10.1.2 |

# Programming with Ryu

**your_controller.py**




```
@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def switch_features_handler(self, ev):
    datapath = ev.msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
```

datapath.id : switch ID

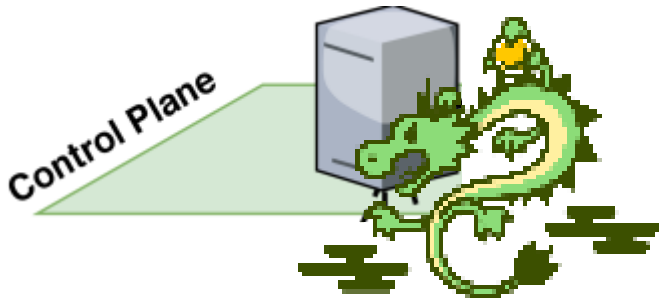
```
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
```

*Your code …*

***Python code***

# Programming with Ryu

your_controller.py




| Match | Action |
| --- | --- |
| any | Controller |

```python
@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def switch_features_handler(self, ev):
    datapath = ev.msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    match = parser.OFPMatch()
    actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
                                      ofproto.OFPCML_NO_BUFFER)]
    self.add_flow(datapath, 0, match, actions)

@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
```

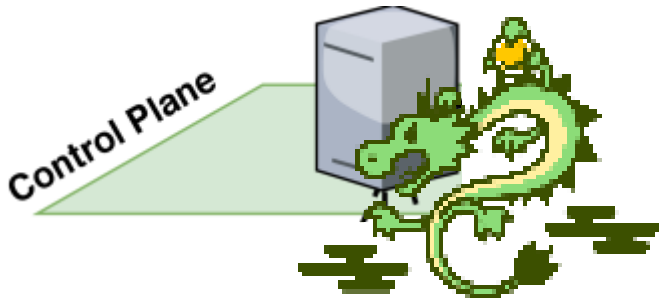
Match (anything)

*Your code …*

***Python code***

# Programming with Ryu

your_controller.py




| Match | Action |
| --- | --- |
| | |
| | |
| any | Controller |

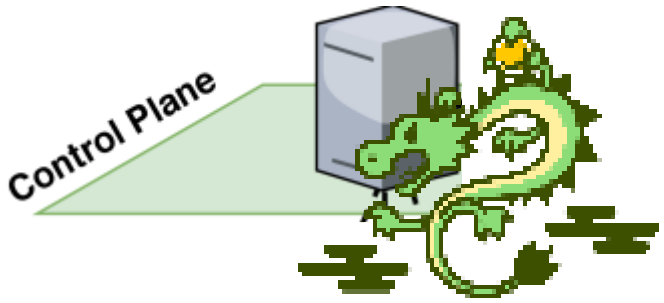
```python
@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def switch_features_handler(self, ev):
    datapath = ev.msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    match = parser.OFPMatch()
    actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
                                      ofproto.OFPCML_NO_BUFFER)]
    self.add_flow(datapath, 0, match, actions)



@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
```

*Your code …*

***Python code***

# Programming with Ryu

your_controller.py




| Match | Action |
| --- | --- |
| | |
| | |
| any | Controller |

```
@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def switch_features_handler(self, ev):
    datapath = ev.msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser



@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
    match = parser.OFPMatch()
    actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
                                      ofproto.OFPCML_NO_BUFFER)]
    self.add_flow(datapath, 0, match, actions)
```

Add matches

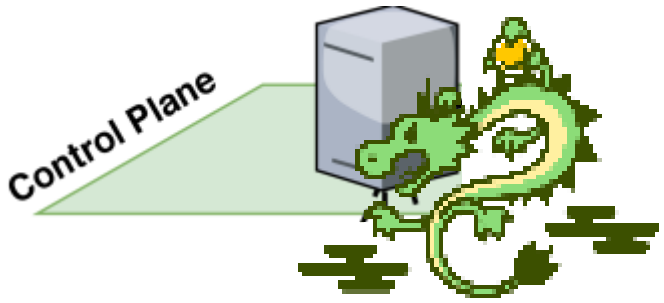
Add action

*Your code …*

***Python code***

# Programming with Ryu

**your_controller.py**




| Match | Action |
|---|---|
| | |
| In_port, eth_dst | Out_port |
| **any** | Controller |

```python
@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def switch_features_handler(self, ev):
    datapath = ev.msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
```

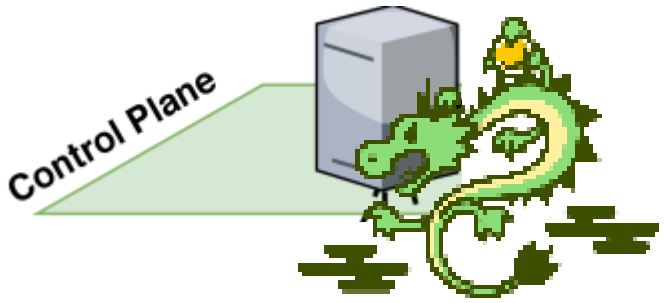
```python
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
    match = parser.OFPMatch(in_port=in_port, eth_dst=dst, eth_src=src)
    actions = [parser.OFPActionOutput(out_port)]
```

*Your code …*

***Python code***

# Programming with Ryu

**your_controller.py**




| Match | Action |
|---|---|
| | |
| In_port, eth_dst | Out_port |
| **any** | Controller |

```
@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def switch_features_handler(self, ev):
    datapath = ev.msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
```

**Important !!!!**
https://ryu.readthedocs.io/en/latest/ofproto_v1_3_ref.html
section: **Flow Match Structure**

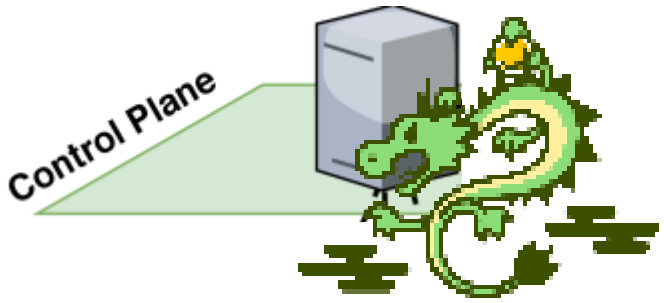
```
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
    match = parser.OFPMatch(in_port=in_port, eth_dst=dst, eth_src=src)
    actions = [parser.OFPActionOutput(out_port)]
```

*Your code …*

***Python code***

# Programming with Ryu

**your_controller.py**

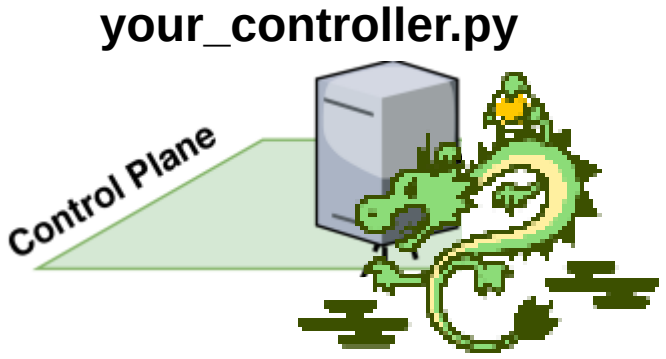



*Remember to packet-out!*

```python
@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def switch_features_handler(self, ev):
    datapath = ev.msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):

    out = parser.OFPPacketOut(datapath=datapath, buffer_id=msg.buffer_id,
                              in_port=in_port, actions=actions, data=data)
    datapath.send_msg(out)
```

***Python code***

# Programming with Ryu



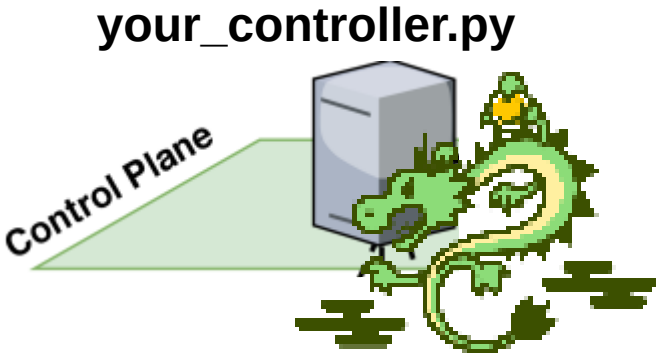

```python
@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def switch_features_handler(self, ev):

@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
```

***Python code***

**Important !!!!**

https://github.com/faucetsdn/ryu/blob/master/ryu/app/simple_switch_13.py
Whole sample code

# Programming with Ryu




```python
@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def switch_features_handler(self, ev):

@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
```

***Python code***

| Source | Destination |
|---|---|
| IP: ? | IP: ? |
| TCP: ? | TCP: ? |

To collect the header info, you'll need…

# Useful Functions in RYU

**Important !!!!**
https://ryu.readthedocs.io/en/latest/library_packet_ref.html

**from ryu.lib.packet**
**import packetfrom ryu.lib.packet**
**import ethernet**

```
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def packet_in_handler(self, ev):
    msg = ev.msg
    datapath = msg.datapath
    dpid = datapath.id
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
    in_port = msg.match['in_port']

    pkt = packet.Packet(msg.data)
    eth = pkt.get_protocols(ethernet.ethernet)

    src = eth.src
    dst = eth.dst
```

Get MAC address

# Useful Functions in RYU

```
# Get arp protocol, if it is not ARP, then it'll be
None(or false)
pk_arp = pkt.get_protocol(arp.arp)

if pk_arp:
    if (src not in self.net.nodes):
        self.AddHosts(src, datapath.id, in_port)

    print("[ARP] arrive at ", datapath.id, dst)
    if dst in self.net.nodes:
```

*Take care ARP packets first, then ICMP, TCP, UDP

*Recommend methods: (pick one)

1. “Hard-coded ARP table” with ovs-ofctl in Lab 2
2. Let the controller generate the ARP reply packets
3. Use “networkx” to construct the topology

https://ryu.readthedocs.io/en/latest/library_packet_ref.html

# Useful Functions in RYU

```
elif pkt_tcp:
    print("[TCP] arrive at ", datapath.id, dst)
    src_host_list = [n for n in self.net.neighbors(src)]
    src_sw_id = src_host_list[0]


    if (src_sw_id == 2 or src_sw_id == 4) and (pkt_tcp.dst_port == 80):

        mypkt = packet.Packet()

        mypkt.add_protocol(ethernet.ethernet(ethertype=eth.ethertype,src=dst,dst=src))
        mypkt.add_protocol(ipv4.ipv4(src=pkt_ipv4.dst,dst=pkt_ipv4.src,proto=6))
        mypkt.add_protocol(tcp.tcp(src_port=pkt_tcp.dst_port,
                                   dst_port=pkt_tcp.src_port,
                                   ack=pkt_tcp.seq+1,
                                   bits=0b010100))

        self._send_packet(datapath, in_port, mypkt)
        print("TCP: Reject connection")
```

*Use RST packet to reject the host

https://ryu.readthedocs.io/en/latest/library_packet_ref.html

# Useful Functions in RYU

```
elif pkt_tcp:
def _send_packet(self, datapath, port, pkt):
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
    pkt.serialize()
    self.logger.info("packet-out %s" % (pkt,))
    data = pkt.data
    actions = [parser.OFPActionOutput(port=port)]
    out = parser.OFPPacketOut(datapath=datapath,
                              buffer_id=ofproto.OFP_NO_BUFFER,
                              in_port=ofproto.OFPP_CONTROLLER,
                              actions=actions,
                              data=data)
    datapath.send_msg(out)
        Use RST packet to reject the host
```

https://ryu.readthedocs.io/en/latest/library_packet_ref.html

# Install RYU

- Python based framework
- Reference:
  - https://github.com/faucetsdn/ryu
- To install RYU
  - use pip: **pip3 install RYU** (recommended)
    - sudo apt install python3-ryu (for Ubuntu 20.04)
  - from source:
    - git clone git://github.com/osrg/ryu.git
    - cd ryu
    - python ./setup.py install
- Verify
  - $ ryu

# To Run Your Controller

- First, start your Mininet with your custom topology
  - sudo mn --custom your_topo.py --topo mytopo --mac --switch ovsk --controller remote
- Configure OpenFlow version on the switch
  - ex: ovs-vsctl set Bridge s1 protocols=OpenFlow13
  - set switch 1 to OpenFlow v1.3
- Run RYU controller
  - ryu-manager --verbose your_controller.py
  - verbose: allow the controller to print out more info

# Hint!

- Handling ARP table (IP-MAC mapping)
  - You can use --mac when creating the mininet topology, which may makes easier.
  - Let the controller reply to ARP request or manually setup the rules with “ovs-ofctl”
- When using addLink in your customized topology file
  - you can use addLink(H1, S1, 1, 1) to specify the port number