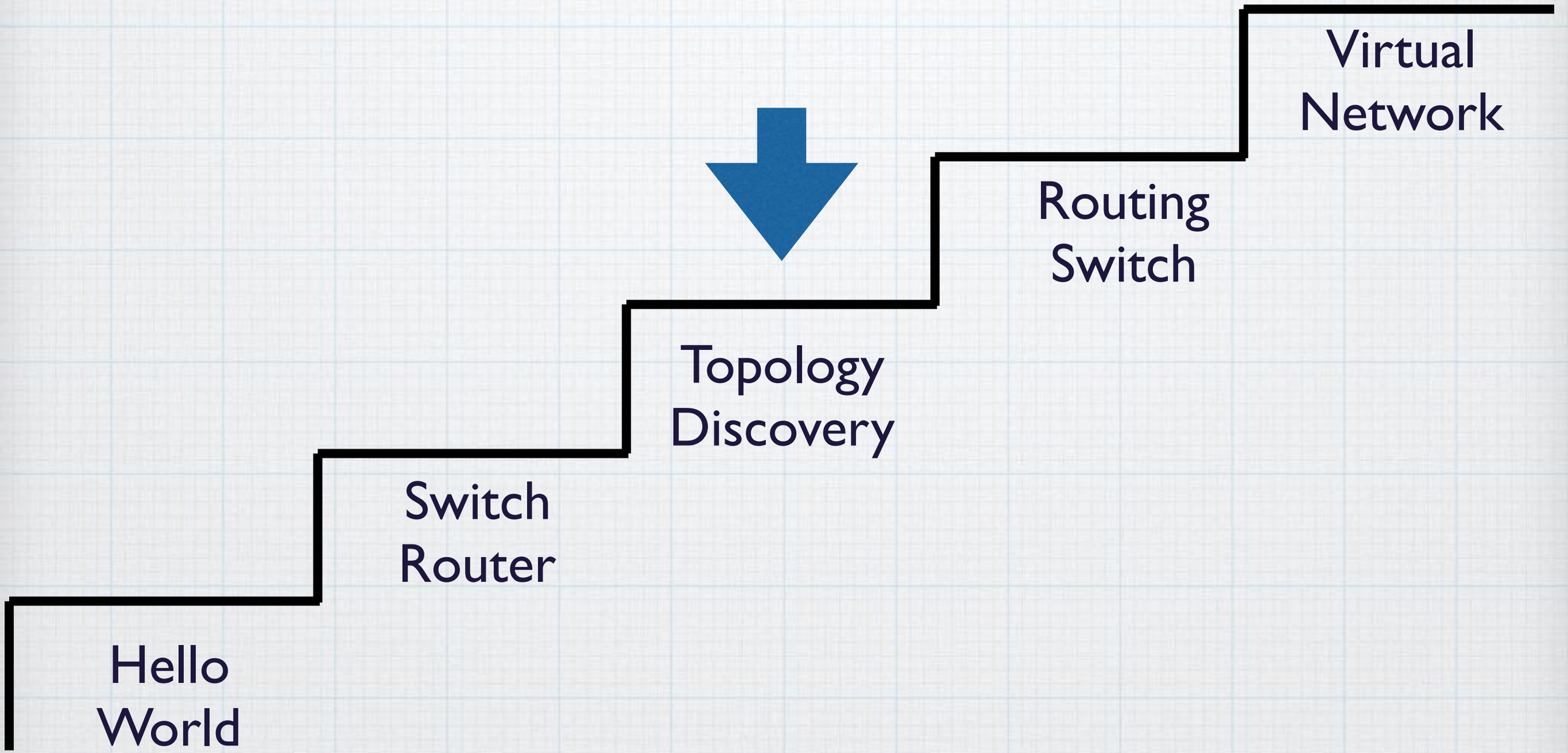
Network Topology Discovery

高宮 安仁 @yasuhito


Virtual
Network
Routing
Switch
Topology
Discovery
Switch
Router
Hello
World

«CONFIDENTIAL»

2012-6/15 10:00

古い版は
ュレッダーへ廃棄

#show.net

> Reborn to the Future

INTEROP

TOKYO | 12-15 JUNE 2012

Device ID X

LoopBack Address 46.0.0.X/32

P2P Address Block Link ID = XYZ
46.0.X.(Y2*X4)/30

P2P VLAN ID XYZ (Link ID)

MGMT Address 172.16.X.0/16 or 172.16.0.X
fd00:X::/64 or fd00::X

.monitor

.mgmt
Management

.noc

.ofsc
OpenFlow Showcase

.note

external

core

VPLS

.lastresort

.of-life

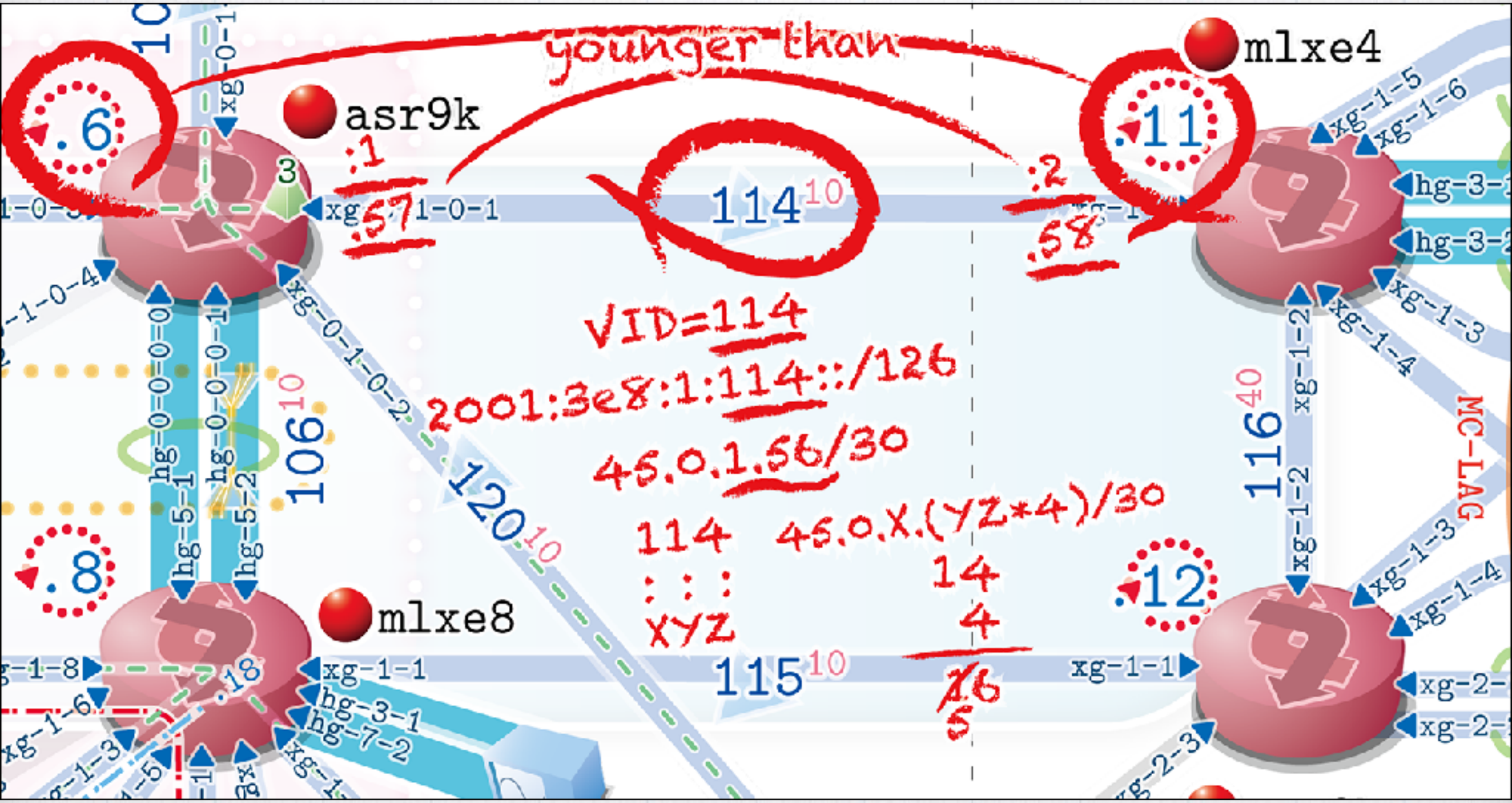
.of-security

OpenFlow

.orc

.life

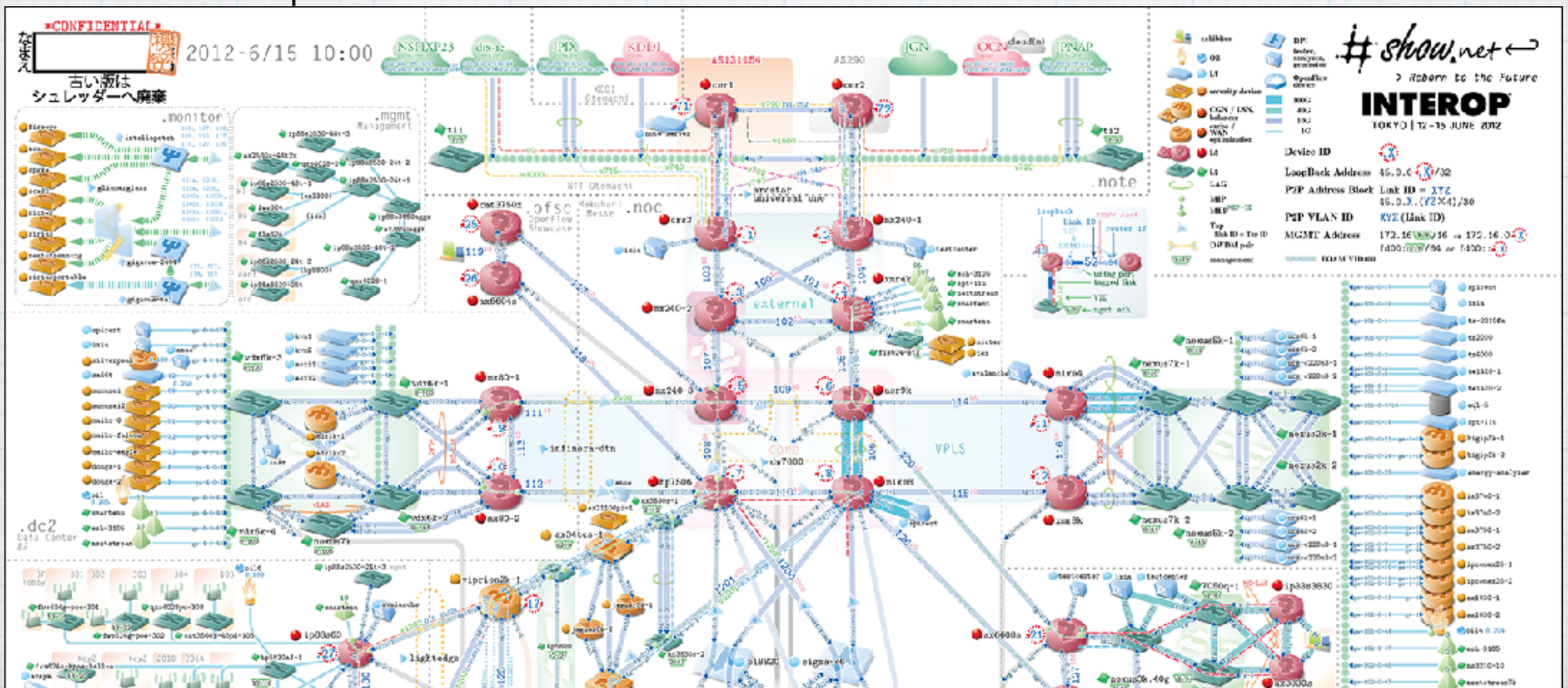
Visitors & Exhibitors



Topology

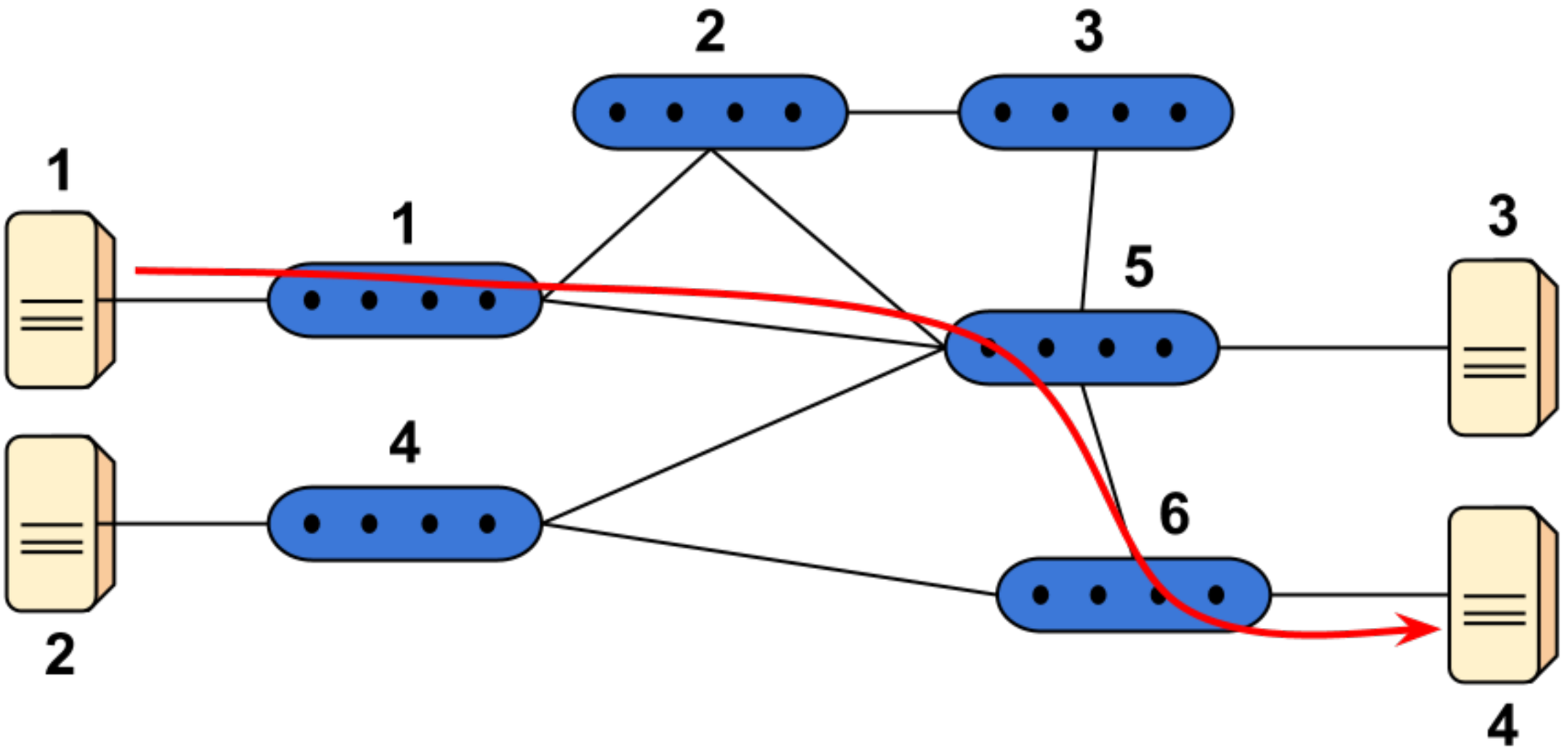
- Though an initial configuration can be statically determined, suitable configurations for a network in operation might be changing according to failures or network reconfigurations.

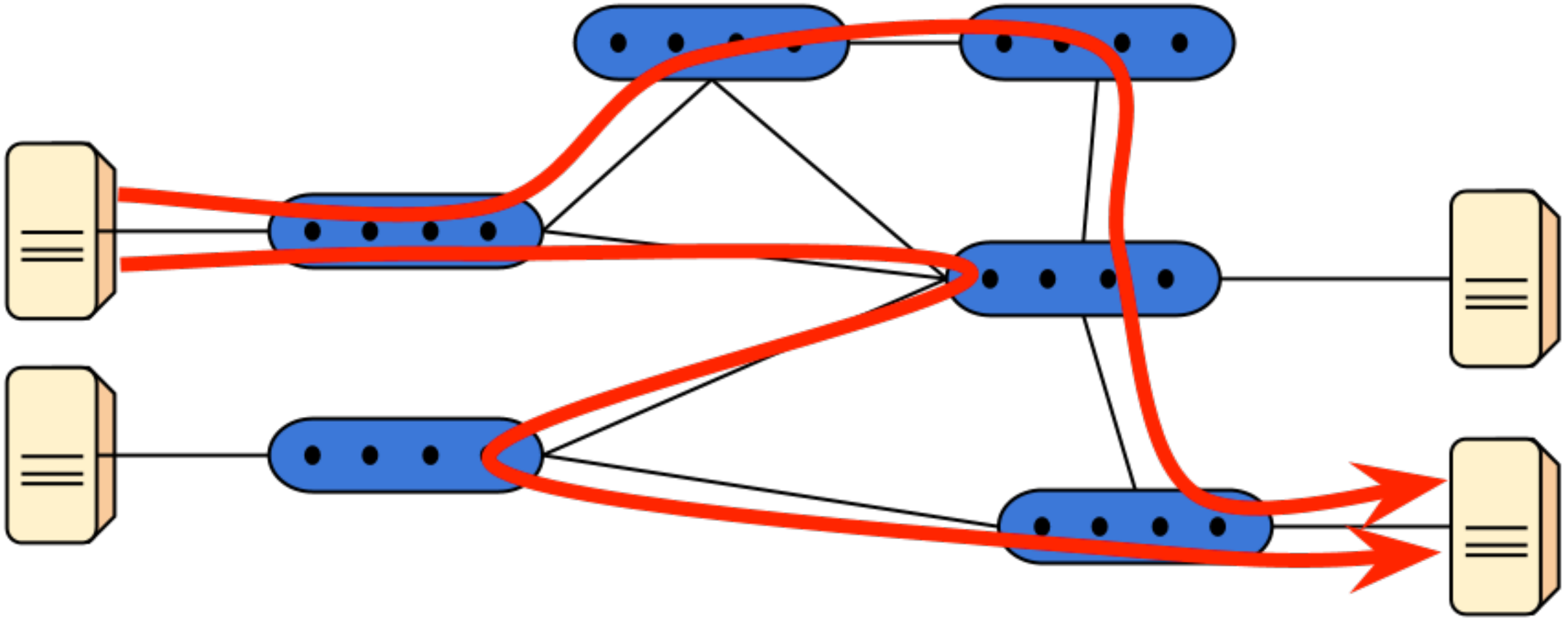
→ Detecting changes in network topologies is a useful technique for network operation.



Routing Switch

Modify Flow Entries





- A routing switch allows paths other than the shortest one to be used for transporting flows.

- Using multiple paths, it can utilize available bandwidth efficiently.

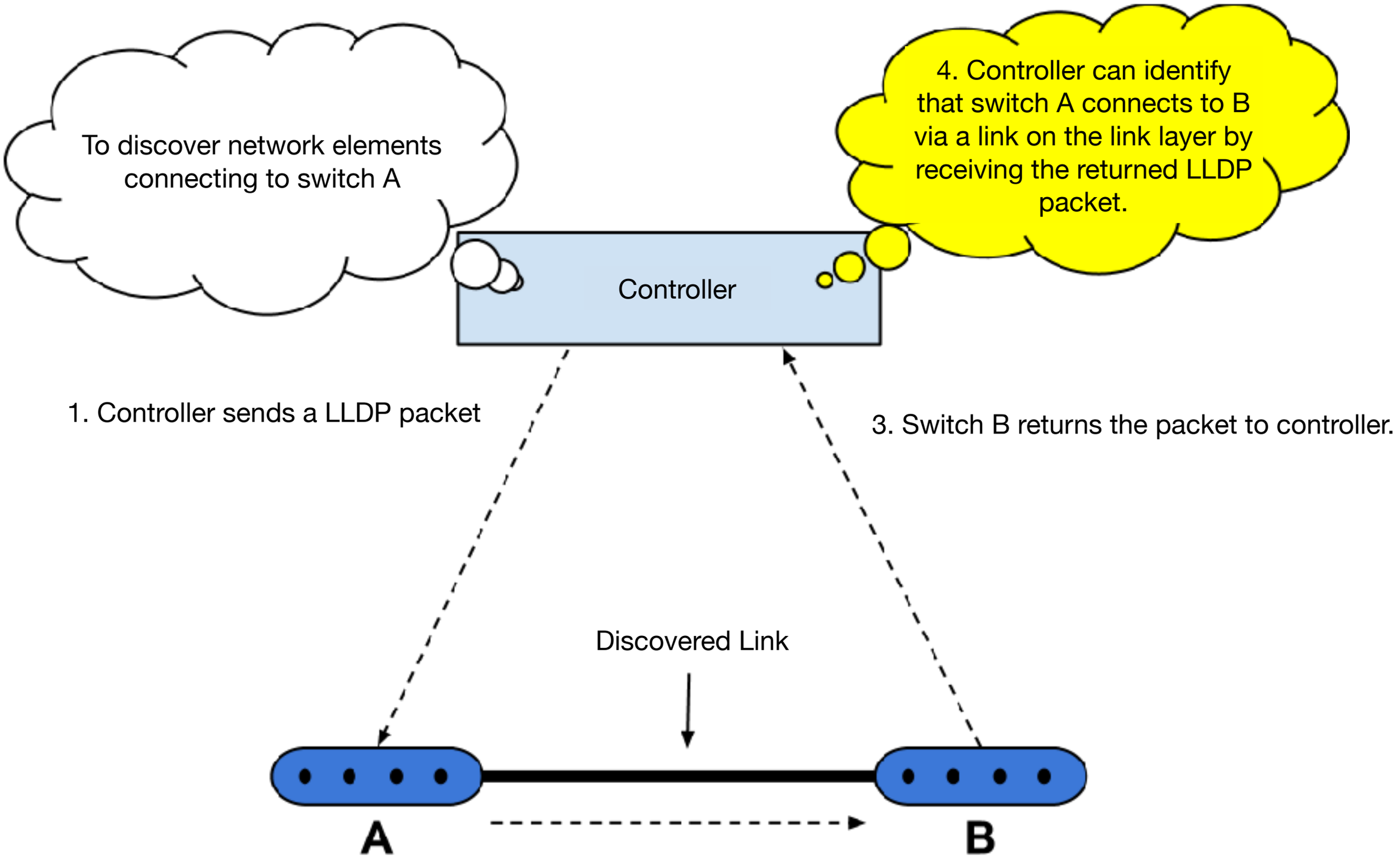
Topology Discovery

Link Layer Discovery Protocol (IEEE 802.1AB)

A protocol to detect connections on the link layer.

dpid

Destination MAC	Source MAC	Ethertype	Chassis ID TLV	Port ID TLV	Time to live TLV	Optional TLVs	End of LLDPDU TLV
01:80:c2:00:00:0e, or 01:80:c2:00:00:03, or 01:80:c2:00:00:00	Station's address	0x88CC	Type=1	Type=2	Type=3	Zero or more complete TLVs	Type=0, Length=0



2. Switch A has a link to switch B, and it forwards the LLDP packet to switch B.

To discover network elements connecting to switch A

Via a link on the link layer by receiving the returned LLDP packet.

Controller

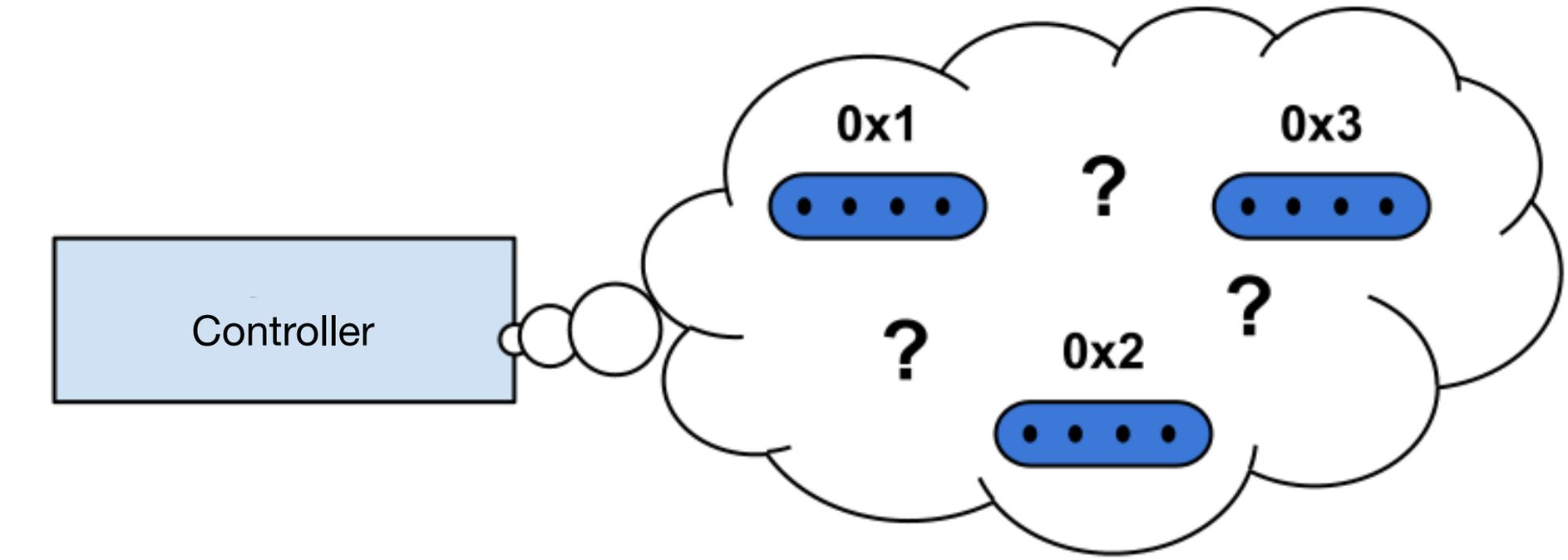
PacketOut

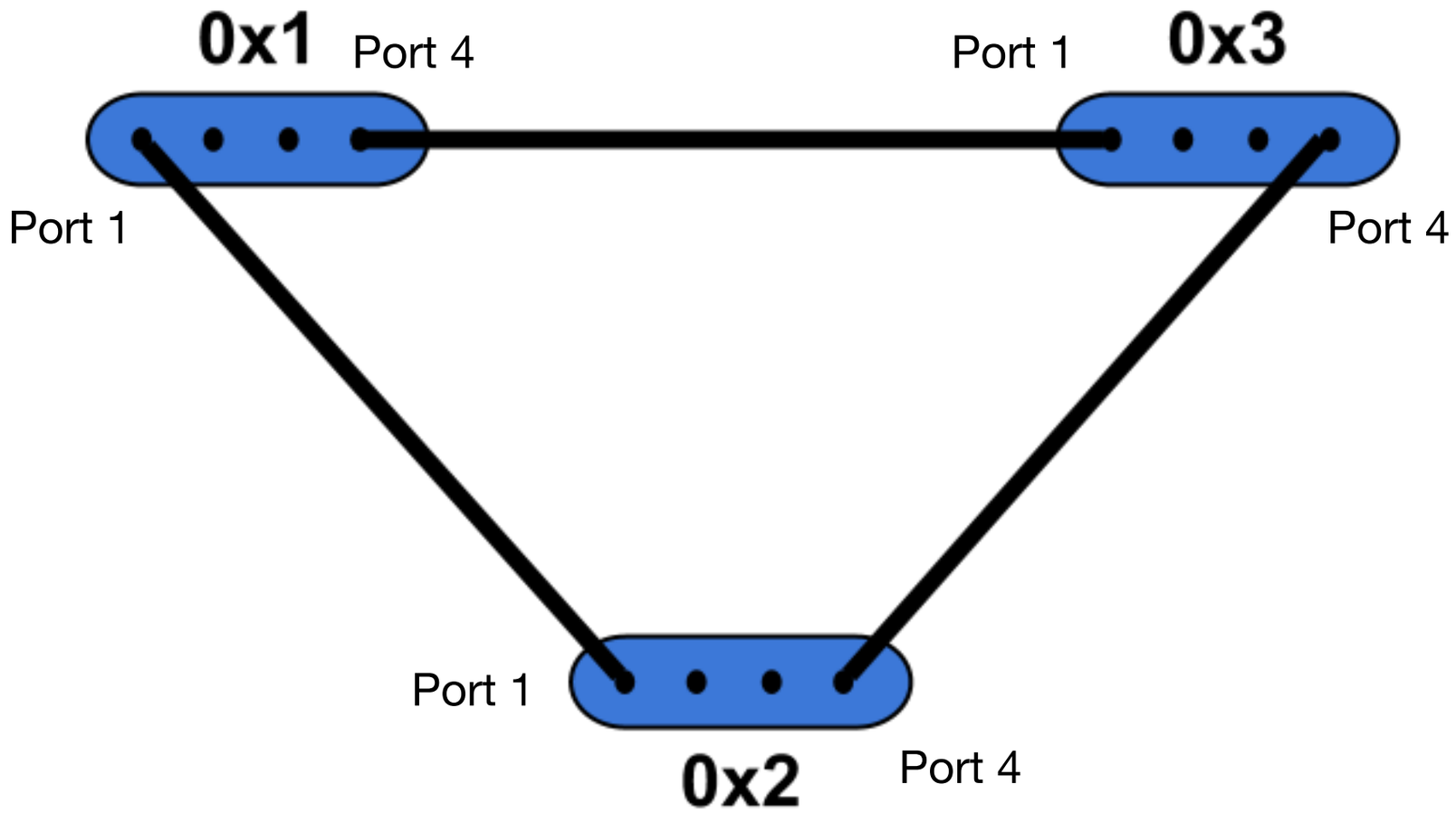
PacketIn

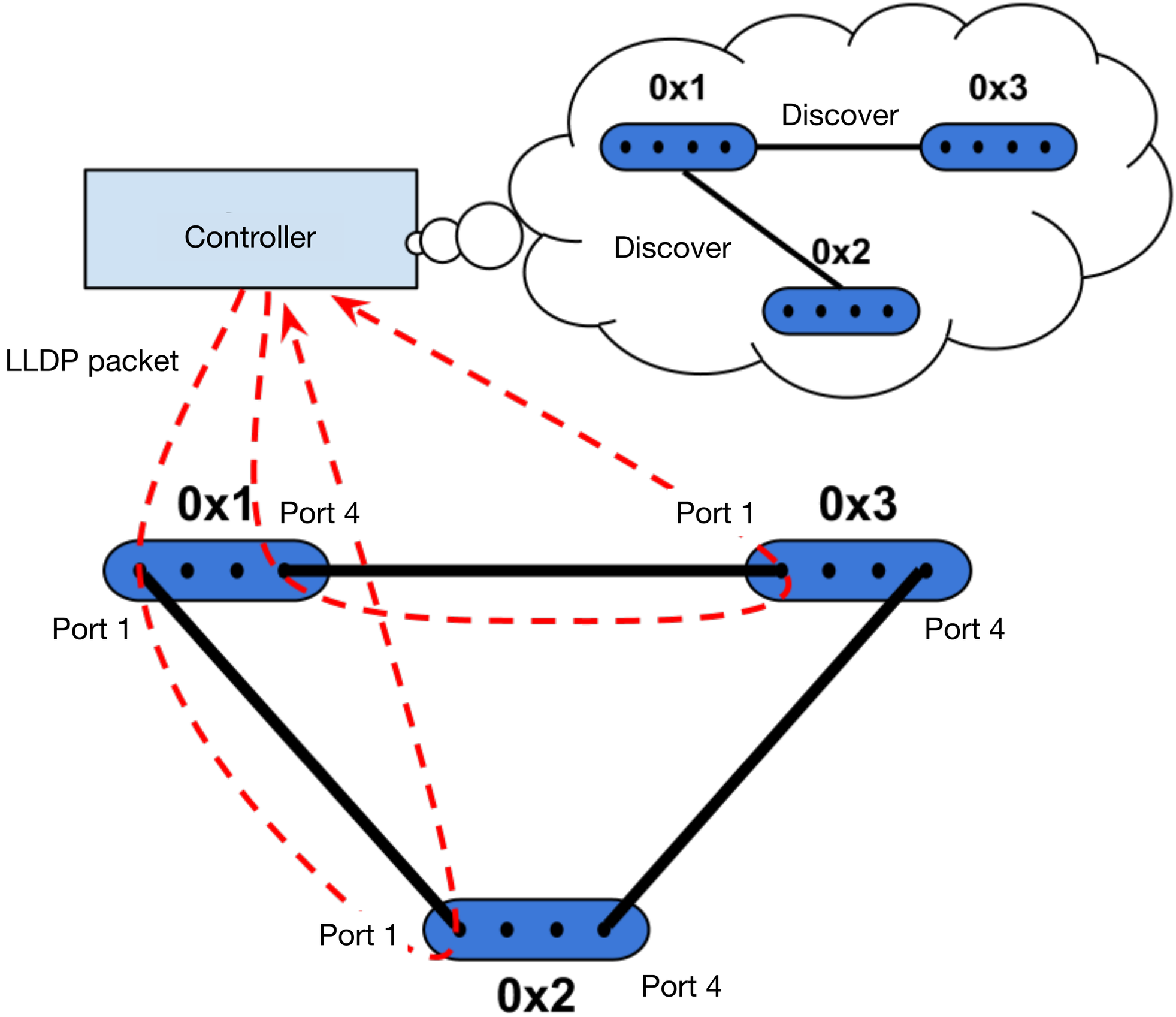
Packet In message
Input Port = 1

LLDP Packet
Switch = 0x1, Output Port = 4

Port 4 on switch A ⇔ Port 1 on switch B


0x1
0x3
?
?
?
0x2
Controller

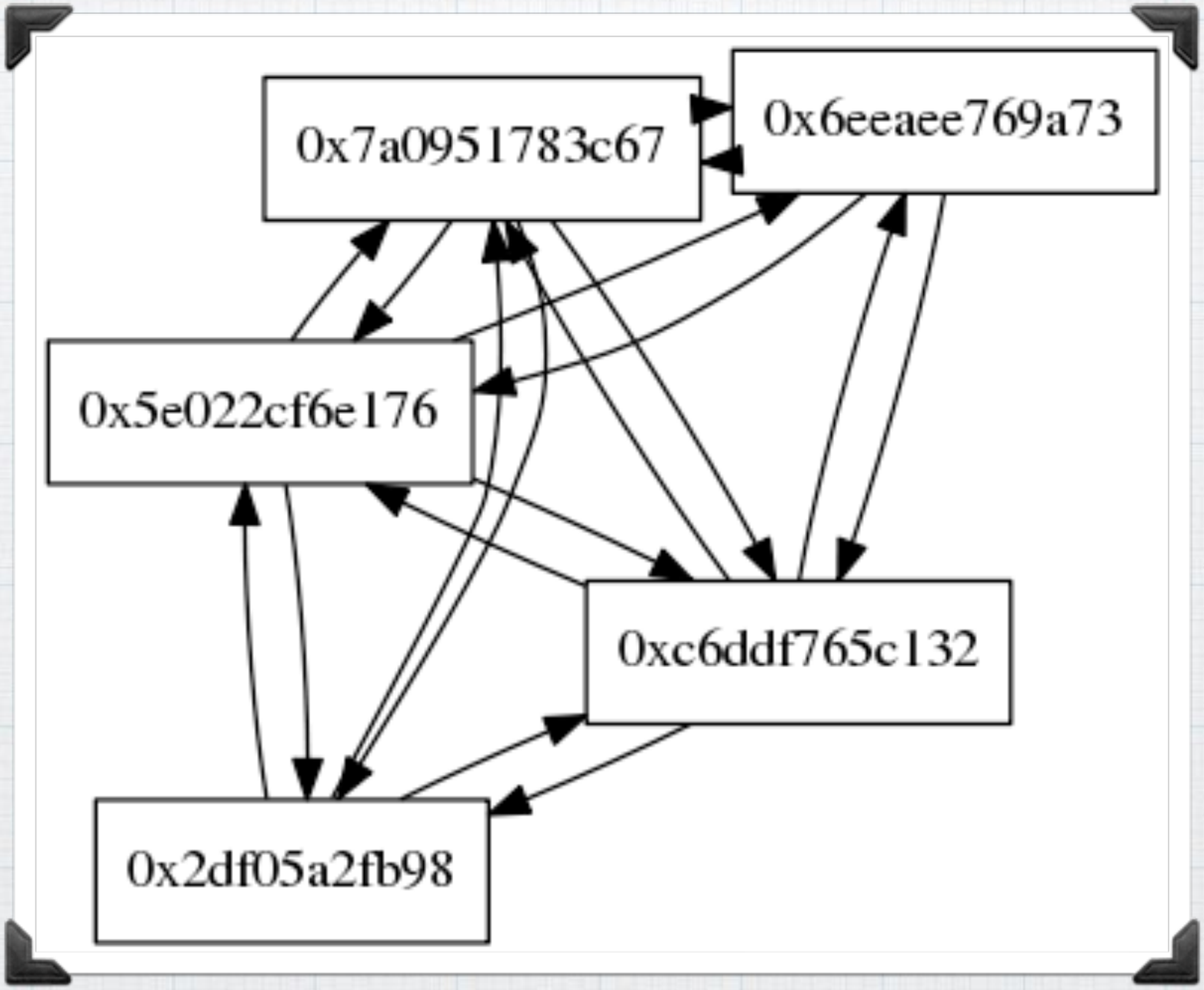

0x1
Port 4
Port 1
0x3
Port 1
Port 4
Port 1
Port 4
0x2


0x1
0x3
Discover
Controller
Discover
0x2
LLDP packet
0x1
Port 4
Port 1
0x3
Port 1
Port 4
Port 1
0x2
Port 4

An example implementation of topology discovery

- topology
  - topology: a name of a topology discovery tool
  - Discover topologies and display them (with text/images)
  - 300 lines in Ruby
  - Today's exercise is based on this tool

AN EXAMPLE OF OUTPUT



Today's exercise

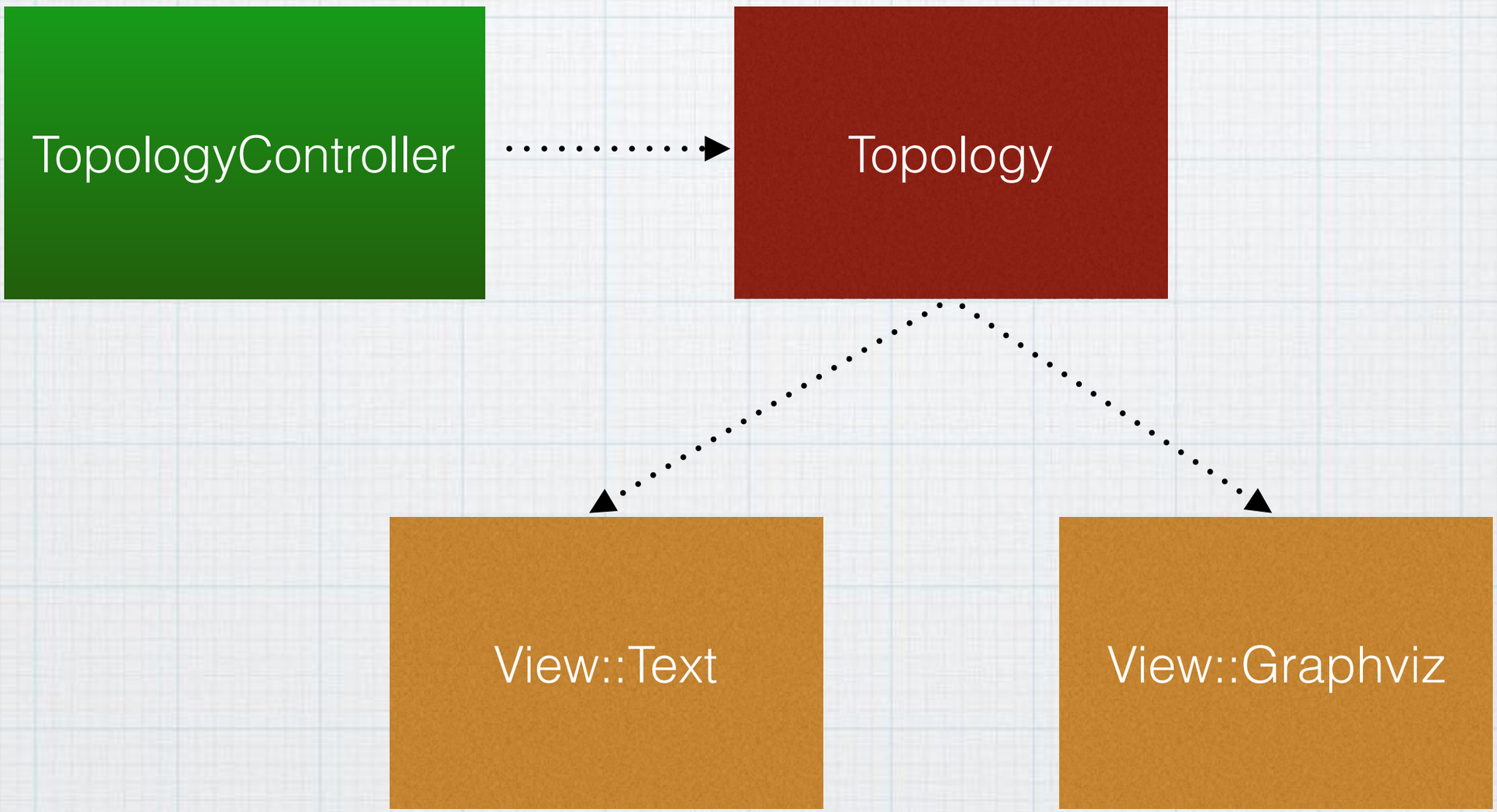
1. Install the "topology"

2. Construct a physical network consisting of about 10 switches

3. Confirm that "topology" discovers the topology of the constructed physical network and generates an images of the topology.

4. Understand the source code of "topology"

Structure of “topology”



OpenFlow Controller

Manage topology information

TopologyController

Topology

Display (text/image)

View::Text

View::Graphviz

Notify Packet In
of LLDP packets
and changes in states
of ports

TopologyController

Topology

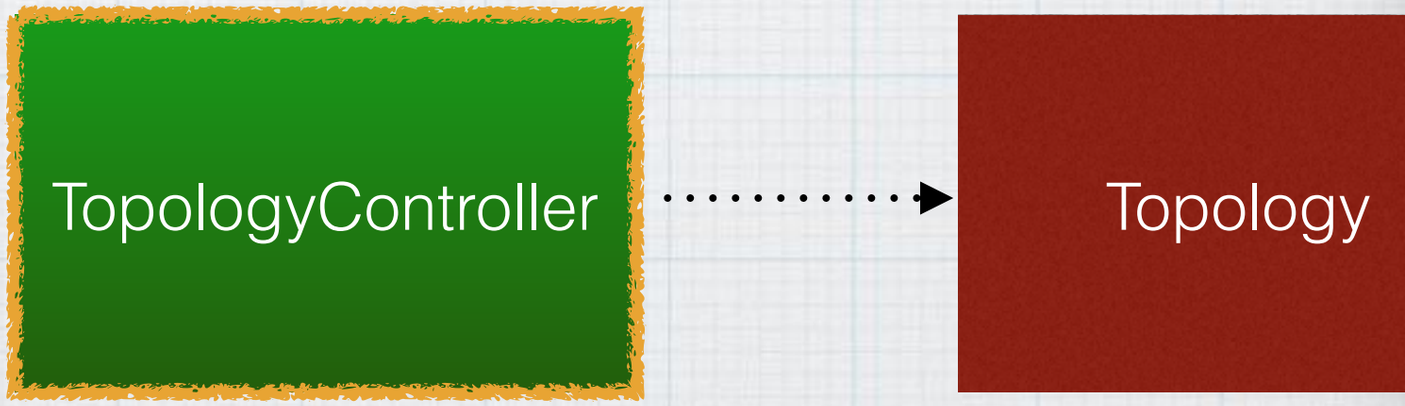
Notify changes in
topologies

View::Text

View::Graphviz

TopologyController

- Get a list of ports on switches
  FeaturesRequest, FeaturesReply
- Update information of ports
  #port_modify
- Flood LLDP packets periodically
  #flood_lldp_frames
- Update information of links
  #packet_in



List of Ports on Switches

```
def switch_ready(dpid)
  send_message dpid, Features::Request.new
end
```

```
def features_reply(dpid, features_reply)
  @topology.add_switch dpid, features_reply.physical_ports.select(&:up?)
end
```

- Add entries of discovered switches to topology information (@topology)
- physical_ports.select(&:up?)
  - Select ports whose states are "UP" amoung physical ports

Change Port Information (UP/DOWN)

```
def port_modify(_dpid, port_status)
  updated_port = port_status.desc
  return if updated_port.local?
  if updated_port.down?
    @topology.delete_port updated_port
  elsif updated_port.up?
    @topology.add_port updated_port
  else
    fail 'Unknown port status.'
  end
end
```

- If a port is DOWN, remove the entry of the port from the topology information
- If a port is UP, add an entry of the port to the topology information

Discovery of New Links/Hosts

```
def packet_in(dpid, packet_in)
  if packet_in.lldp?
    @topology.maybe_add_link Link.new(dpid, packet_in)
  else
    @topology.maybe_add_host(packet_in.source_mac,
                             packet_in.source_ip_address,
                             dpid,
                             packet_in.in_port)
  end
end
```

- If the packet is an LLDP packet, add a link
- Otherwise, add a host

Extract Link Information from an LLDP packet

```
class Link
  def initialize(dpid, packet_in)
    lldp = packet_in.data
    @dpid_a = lldp.dpid
    @dpid_b = dpid
    @port_a = lldp.port_number
    @port_b = packet_in.in_port
end
```

- There is a link between port @port_a on switch @dpid_a and port @port_b on switch @dpid_b

Periodical Flooding of LLDP Packets

```
class TopologyController < Trema::Controller
  timer_event :flood_lldp_frames, interval: 1.sec

  def flood_lldp_frames
    @topology.ports.each do |dpid, ports|
      send_lldp dpid, ports
    end
  end

  private

  def send_lldp(dpid, ports)
    ports.each do |each|
      port_number = each.number
      send_packet_out(
        dpid,
        actions: SendOutPort.new(port_number),
        raw_data: lldp_binary_string(dpid, port_number)
      )
    end
  end
```

Flood packets every 1 second

to all ports

Send LLDP packets

Parse and Generate LLDP Packets

```
# Load
require “pio”

# Parse an LLDP packet
Pio::Lldp.read(packet_in.data).dpid #=>
12345
```

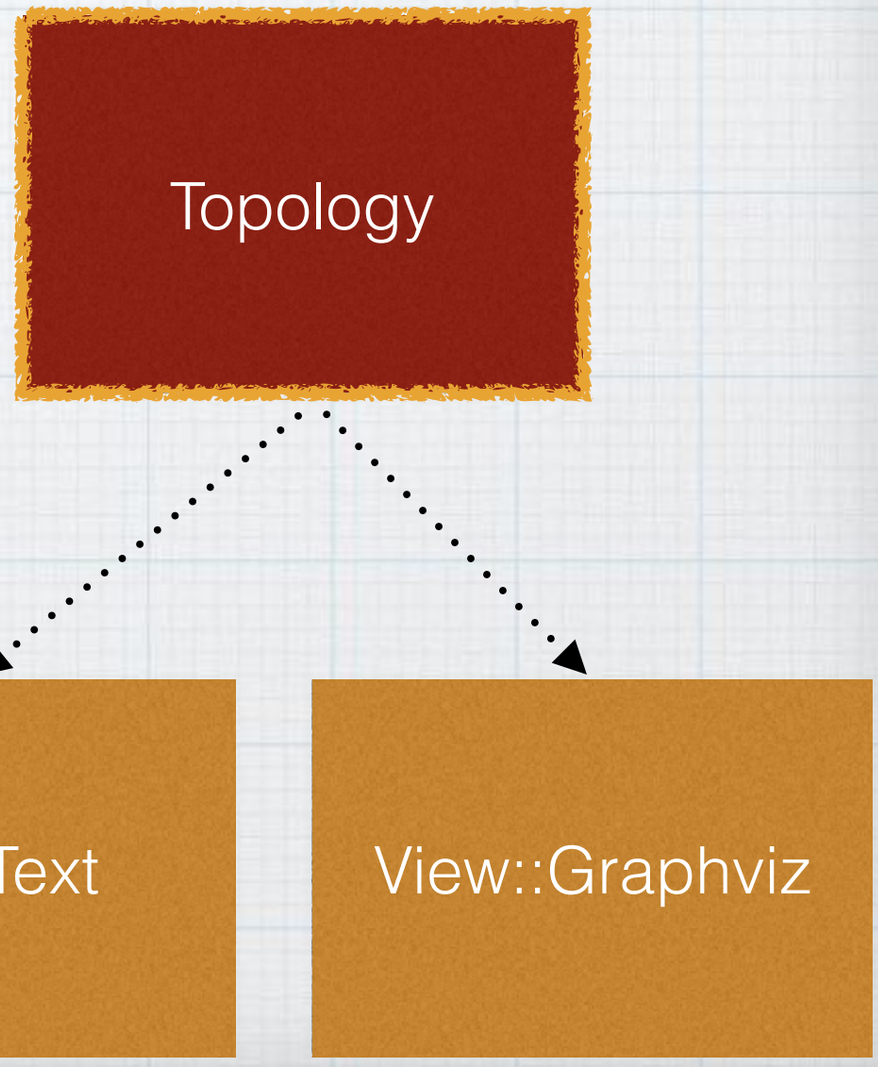
```
# Generate an LLDP packet
lldp = Pio::Lldp.new(:destination_mac
=> ...,
                     :source_mac => ...)

# Send it
send_packet_out(dpid,
                :data => lldp.to_binary,
                ...)
```

Topology

Observer Pattern: Topology notifies View of changes in topologies

- Add and delete ports
  #add_port, #delete_port
- Update port information (up/down)
  #update_port
- Add link information
  #add_link_by



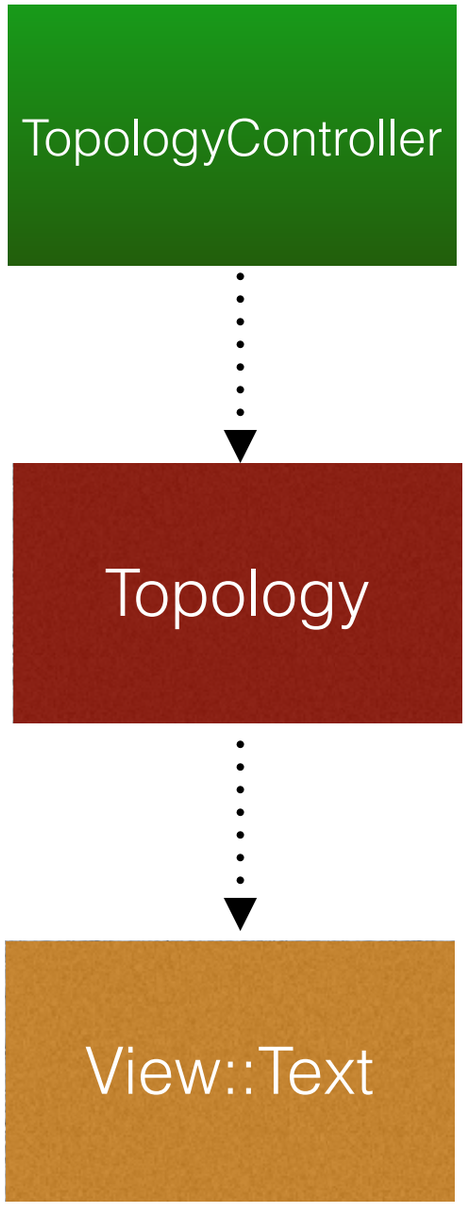
Observer Pattern

```
# TopologyController#start
def start(args)
  @command_line = CommandLine.new(logger)
  @command_line.parse(args)
  @topology = Topology.new
  @topology.add_observer @command_line.view
  logger.info "Topology started (#{@command_line.view})."
end
```

Add observer

- View (@command_line.view) observes topology information.
- When the topology changes, TopologyController is notified by View about the changes.

Notification of Switch Addition




```
def features_reply(dpid, features_reply)
  @topology.add_switch dpid,
features_reply.physical_ports.select(&:up?)
end
```

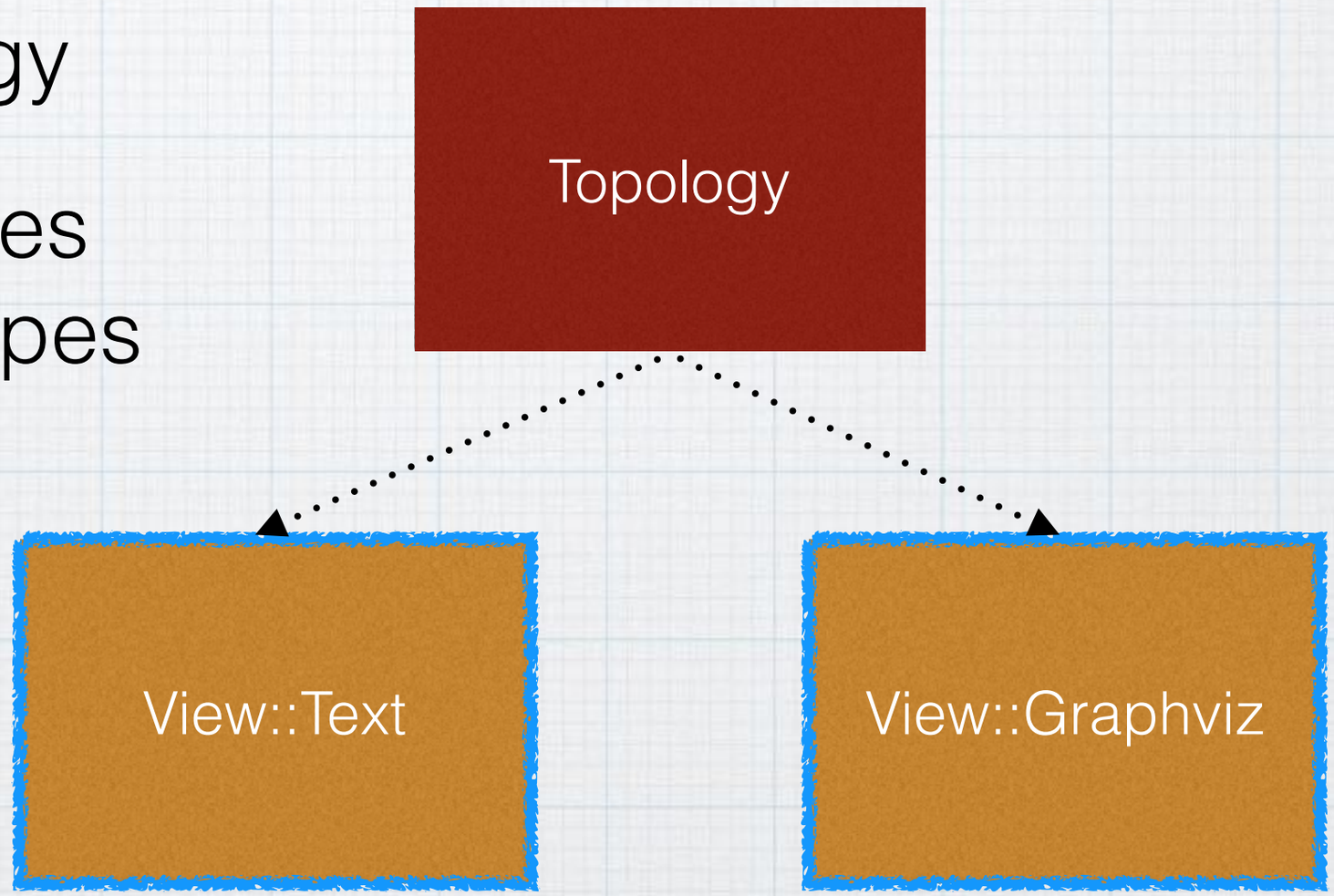
```
def add_switch(dpid, ports)
  ports.each { |each| add_port(each) }
  maybe_send_handler :add_switch, dpid, self
end
```

```
def add_switch(dpid, topology)
  show_status("Switch #{dpid.to_hex} added",
              topology.switches.map(&:to_hex))
end
```

View::*

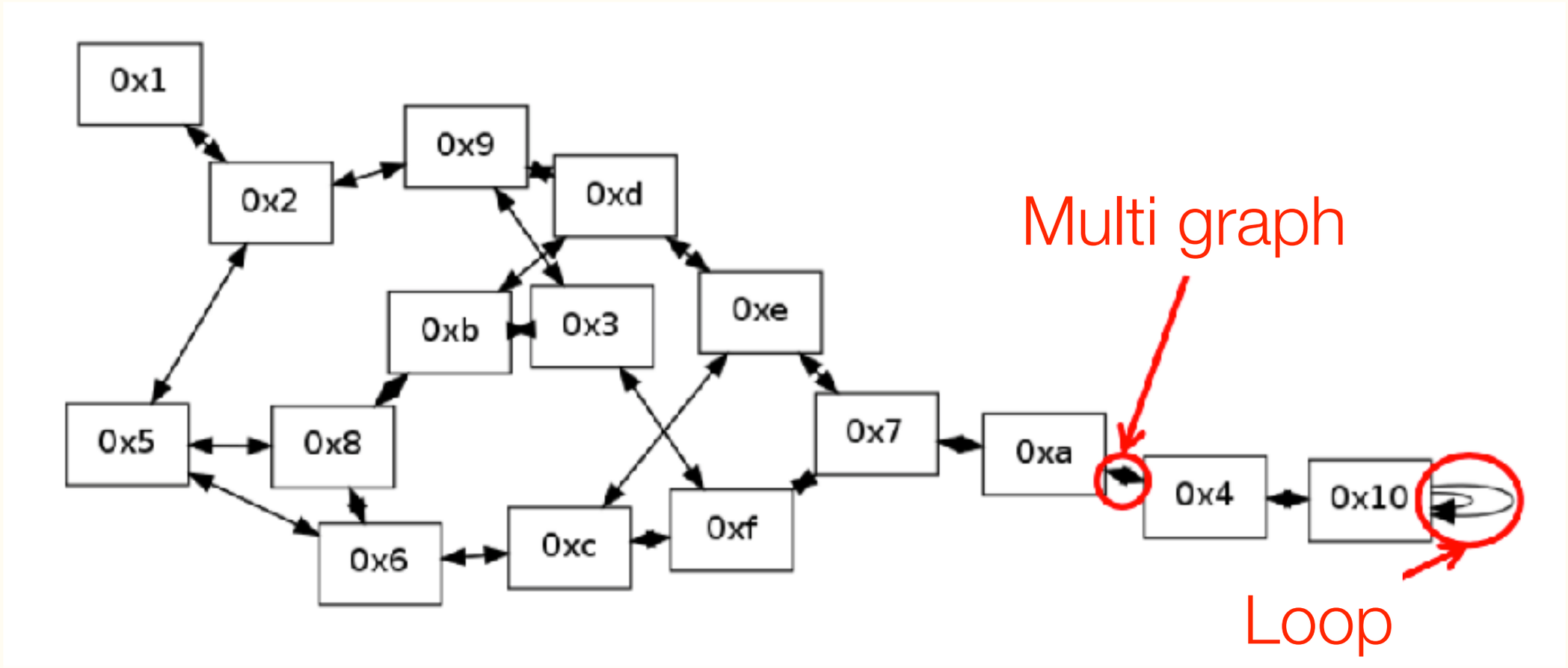
Observe changes in Topology (Observer Pattern)

- Draw the topology
- Implement classes depending on types of the display

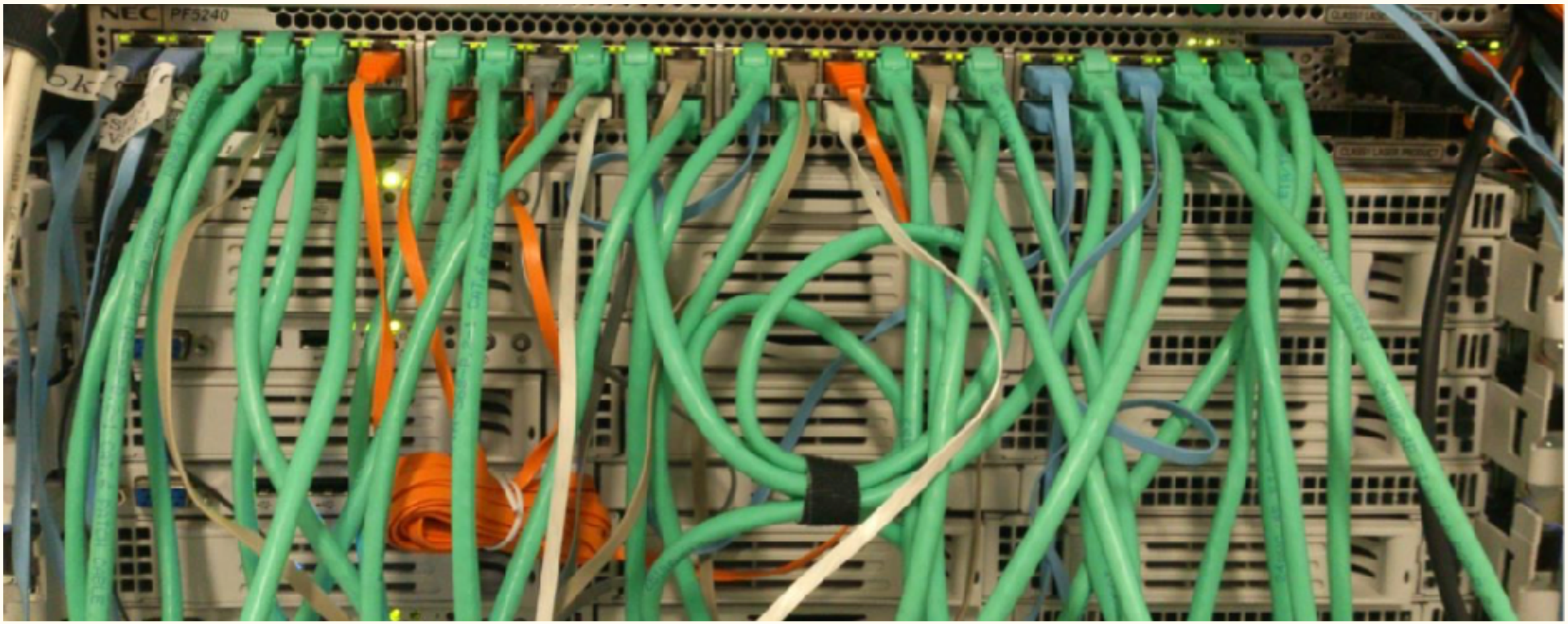


Report 1

1. Create 16 VSIs, each of which has at least 2 ports, on a physical switch
2. Wire all ports (as you like)
3. Display its topology with "topology"
4. Change the wiring and confirm that "topology" detects the changes in the topology and updates the displayed topology image

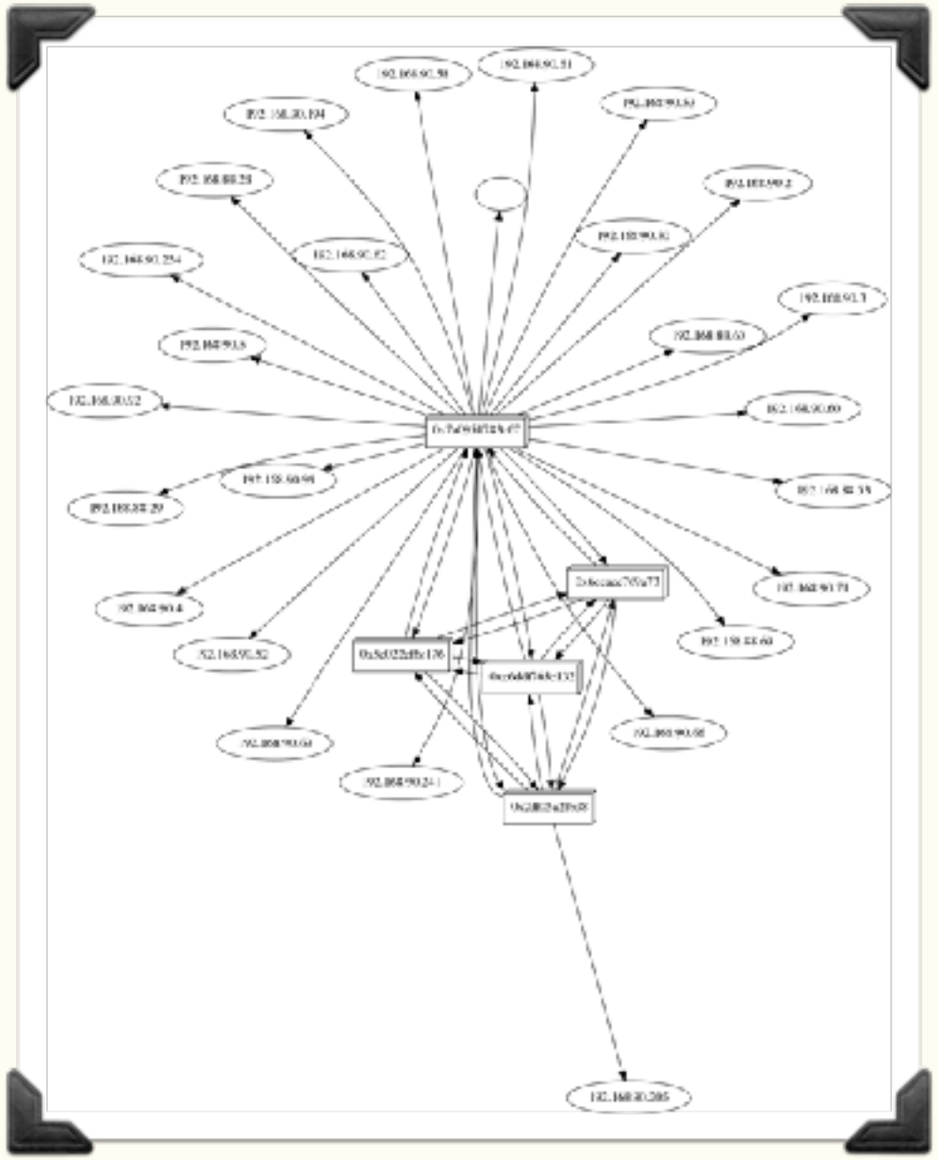


1. VSI x16
2. Assign 3 ports to each VSI
3. Wire them as you like



Report 2

- Modify "topology" so that it displays hosts in addition to switches
- Implement a function to display topologies on a web browser
  - Useful library: vis.js https://github.com/almende/vis



A dandelion drawn with Graphviz

vis.js

- vis.js can generates such images
- Hint: Implement View::VisJs class