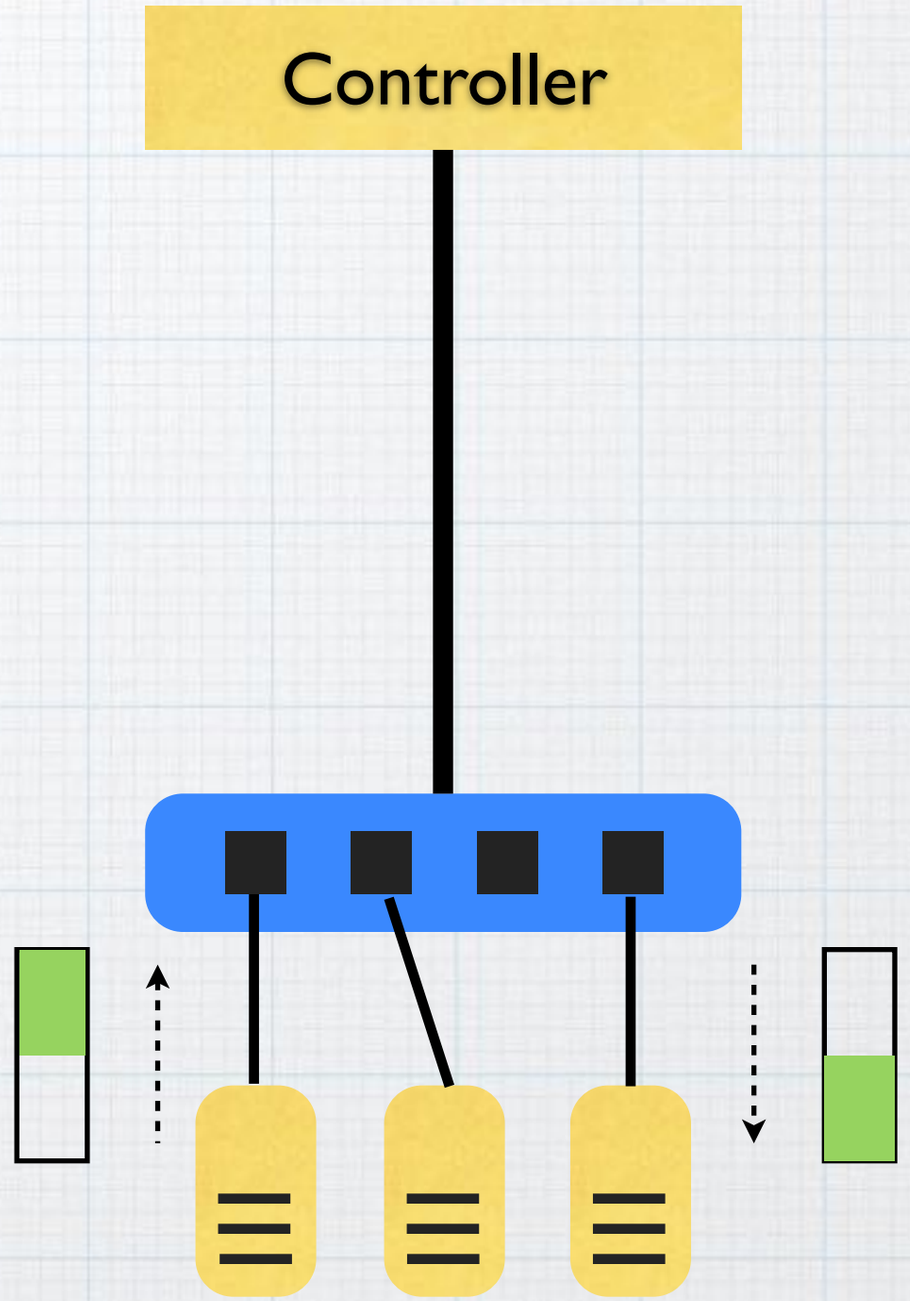
Cbench

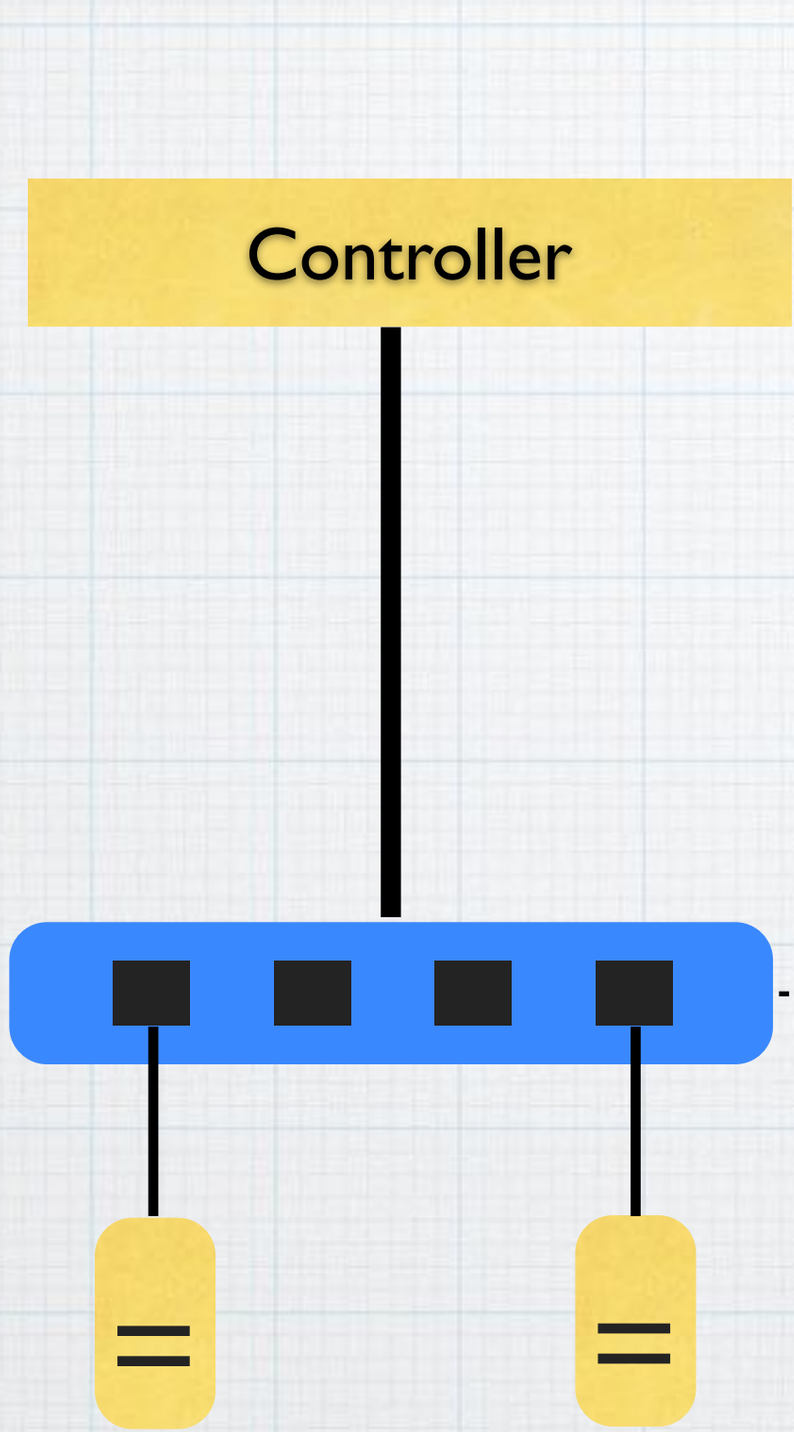
Fundamentals for designing a practical controller



The 3 important messages of OpenFlow

FlowMod · PacketIn · PacketOut

Controller



Flow Table

host1

00:00:..:01

192.168.0.1

host2

00:00:..:02

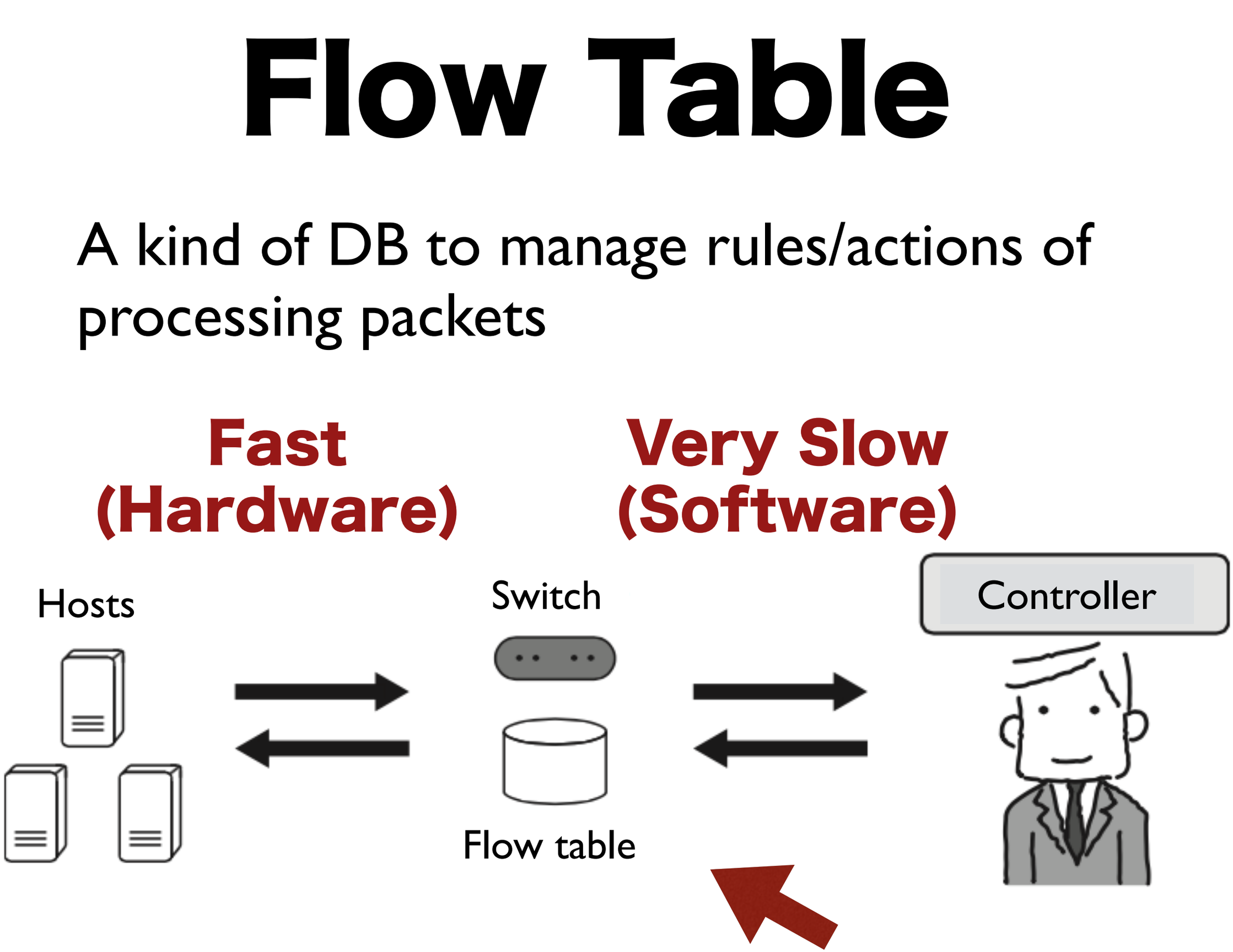
192.168.0.2

Flow Table

A kind of DB to manage rules/actions of processing packets

Fast (Hardware)

Very Slow (Software)



Flow Entries

Packet conditions	Transactions	Number of Packets
Match field	**Action**	**Counter**
Src IP address = 192.168.1.0	Forward packets to port 8	80
VLAN ID = 10	Forward packets to port 10	64
Src MAC address = 00:50:56:c0:00:08	Attach VLAN ID 2 to packets and forward them to port 8	24
Src IP address = 203.0.113.0/16	Discard packets	10

FlowMod

Modify (add, replace, delete) flow entries

```
# add a flow entry
send_flow_mod_add(
  dpid,  # Switch IĐ
  match: Match.new( … ) # Match field
  actions:  …  # Action
)
```

Match Field

Packet conditions | **Transactions** | **Number of Packets**

Match field	Action	Counter
Src IP address = 192.168.1.0	Forward packets to port 8	80
VLAN ID = 10	Forward packets to port 10	64
Src MAC address = 00:50:56:c0:00:08	Attach VLAN ID 2 to packets and forward them to port 8	24
Src IP address = 203.0.113.0/16	Discard packets	10

```
send_flow_mod_add(
  datapath_id,
  match: Match.new(in_port: 1)
  # …
```

- Match packets coming from port 1

```
send_flow_mod_add(
  datapath_id,
  match: Match.new(
    ip_destination_address: '192.168.0.30',
    transport_destination_port: 80
  )
  # …
```

- Match packets whose destination IP address and port number are 192.168.0.30 and 80, respectively.

Conditions for Match Field

- Ingress port
- Ether src
- Ether dst
- Ether type
- IP src
- IP dst
- IP proto
- IP ToS bits
- TCP/UCP src port
- TCP/UDP dst port
- VLAN id
- VLAN priority

```
send_flow_mod_add(
  datapath_id,
  match: ExactMatch.new(message),
  # …
)
```

- ExactMatch: match packets that is identical to “message” in terms of all the 12 conditions listed in the previous page

Action

Actions for packets that match the match field

```
# Add a flow entry
send_flow_mod_add(
  dpid,  # Switch IĐ
  match: Match.new( … ) # Match Field
  actions:  …  # Action
)
```

Action

Match field	Action	Counter
Src IP address = 192.168.1.0	Forward packets to port 8	80
VLAN ID = 10	Forward packets to port 10	64
Src MAC address = 00:50:56:c0:00:08	Attach VLAN ID 2 to packets and forward them to port 8	24
Src IP address = 203.0.113.0/16	Discard packets	10

Action

Forwarding packets

- SendOutPort.new(Port ID)

Rewrite content in packet header (the same fields as the 12 header fields for match fields)

- SetEtherDestinationAddress.new(New destination MAC address)
- SetIpDestinationAddress.new(New destination IP address)
- Etc.

```
send_flow_mod_add(
  datapath_id,
  match: Match.new(in_port: 1),
  actions: SendOutPort.new(4),
  # …
```

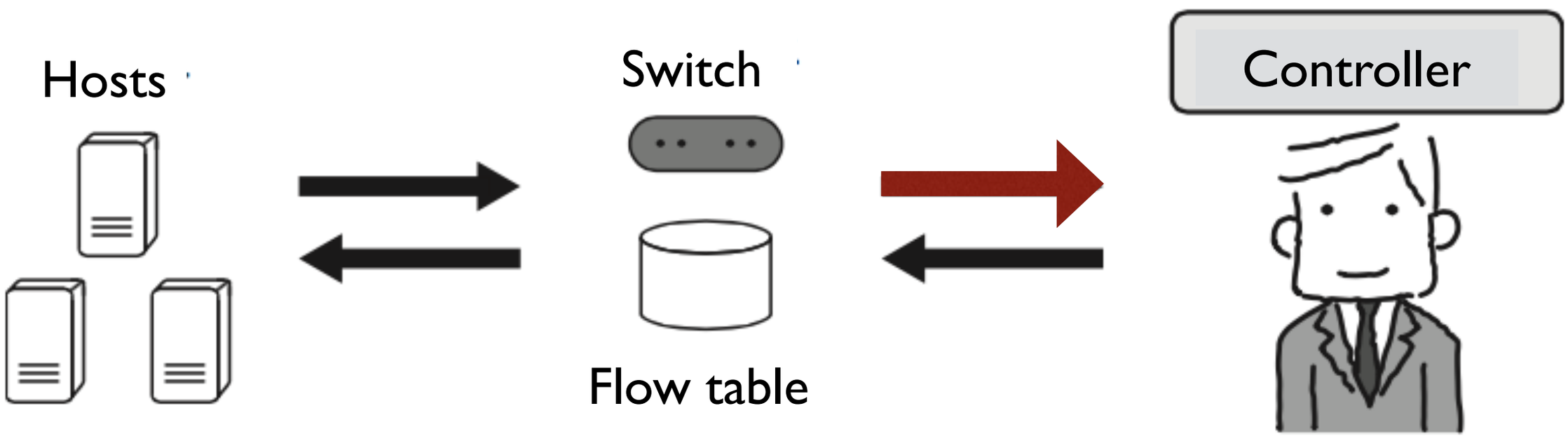
1. Match: If packets come from port 1
2. Actions: Send packets out port 4

```
send_flow_mod_add(
  datapath_id,
  match: Match.new(in_port: 1),
  actions: [SetEtherDestinationAddress.new(…),
            SetIpDestinationAddress.new(…),
            SendOutPort.new(4)]
  # …
```

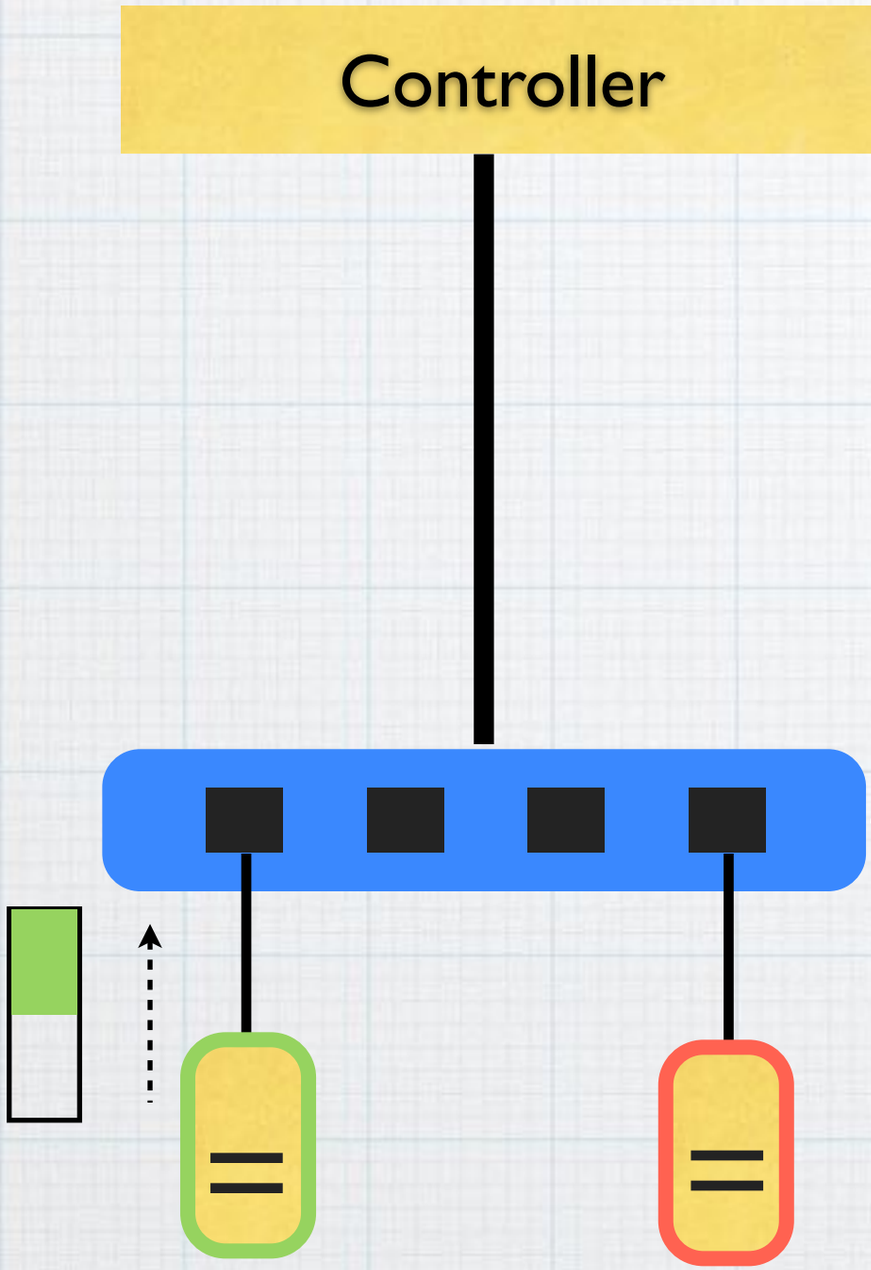
- Multiple actions can be specified
- e.g., rewrite two header fields, and then forward packets

PacketIn

A message to notify a controller of arrivals of packets that does not match any match fields. (i.e., a switch cannot determine how to process the packets)



Controller



Flow Table

host1

00:00:..:01

192.168.0.1

host2

00:00:..:02

192.168.0.2

Controller

Packet In

Flow Table

Arrival of an unknown packet

host1

00:00:..:01

192.168.0.1

host2

00:00:..:02

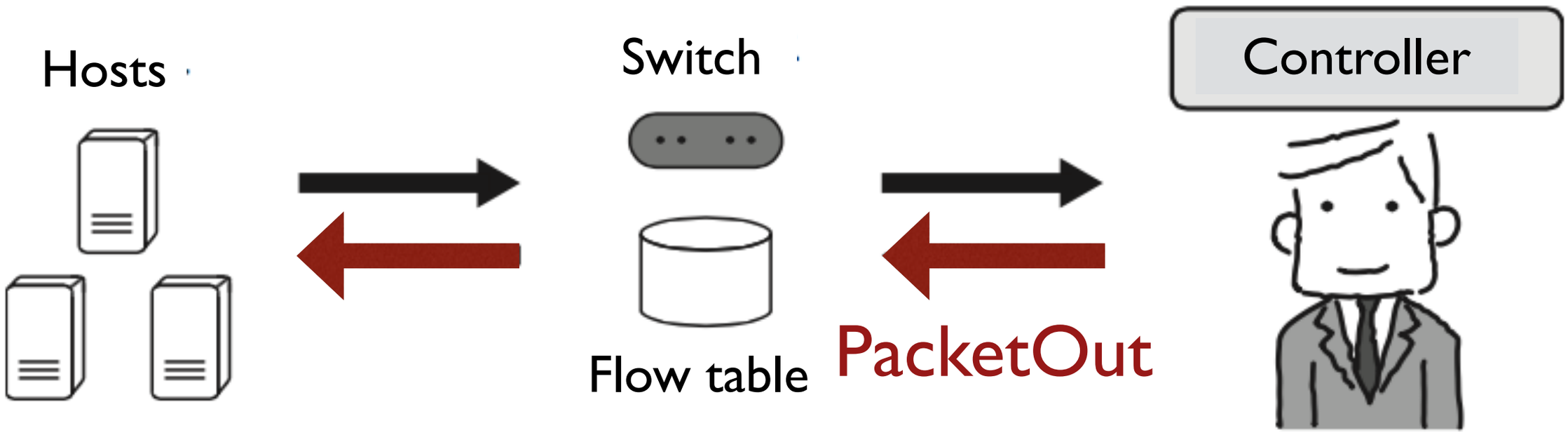
192.168.0.2

```
def packet_in(dpid, message)
  # Inspect the message
  # Do something
end
```

- A handler catch the PacketIn event
- message = PacketIn object

PacketOut

Send out packets
that trigger the PacketIn event



Controller

Packet Out

Packet In

Flow Table

Arrival of an unknown packet

host1

00:00:..:01

192.168.0.1

host2

00:00:..:02

192.168.0.2

```
def packet_in(dpid, message)
  send_packet_out(
    datapath_id,
    in_port: message.in_port,
    raw_data: message.raw_data,
    actions: SendOutPort.new(1)
  )
```

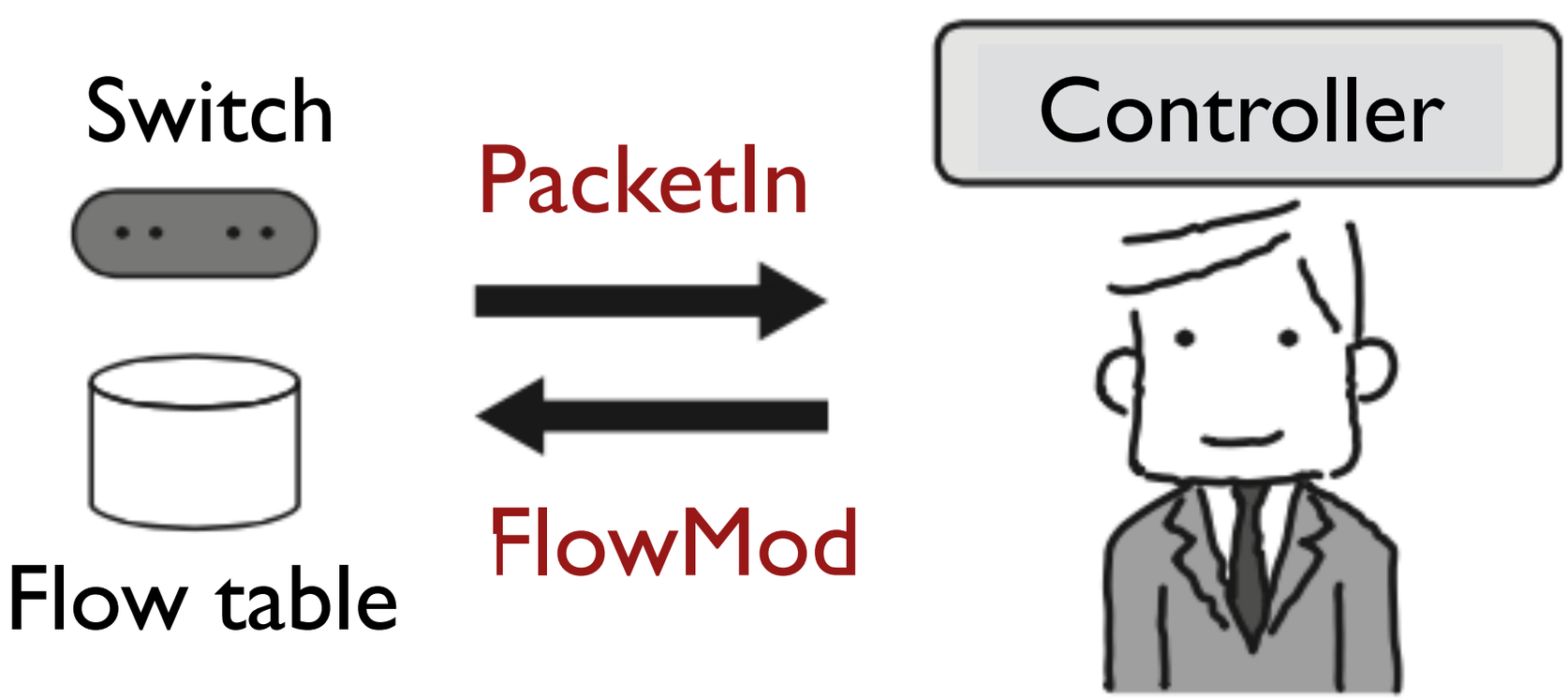
- Send the packet triggers the PacketIn event from port 1 (without any modifications)

```
def packet_in(dpid, message)
  send_packet_out(
    datapath_id,
    in_port: message.in_port,
    raw_data: message.raw_data,
    actions: SendOutPort.new(:flood)
  )
```

- Flood the packet to all ports (except for the incoming port)

cbench

A (micro) benchmark tool for OpenFlow (controllers)



Cbench

A (micro) benchmark tool for OpenFlow (controllers)

- Cbench connects to Trema (a controller), and then sends PacketIn messages to Trema
- Trema replies FlowMod messages and cbench measures the number of the FlowMod messages
- A better controller returns more messages than a poor one

```
send_flow_mod_add(
  datapath_id,
  match: ExactMatch.new(message),
  buffer_id: message.buffer_id,
  actions: SendOutPort.new(message.in_port + 1)
)
```

- The FlowMod message sent to cbench

Conclusion

- The three fundamental messages of OpenFlow
  - FlowMod
  - PacketIn
  - PacketOut
- Cbench is a benchmark tool for an OpenFlow controller
  - Cbench sends PacketIn messages and measures the number of FlowMod replies from the controller