Software Defined Networking and OpenFlow Protocol

Professor Chien-Chao Tseng

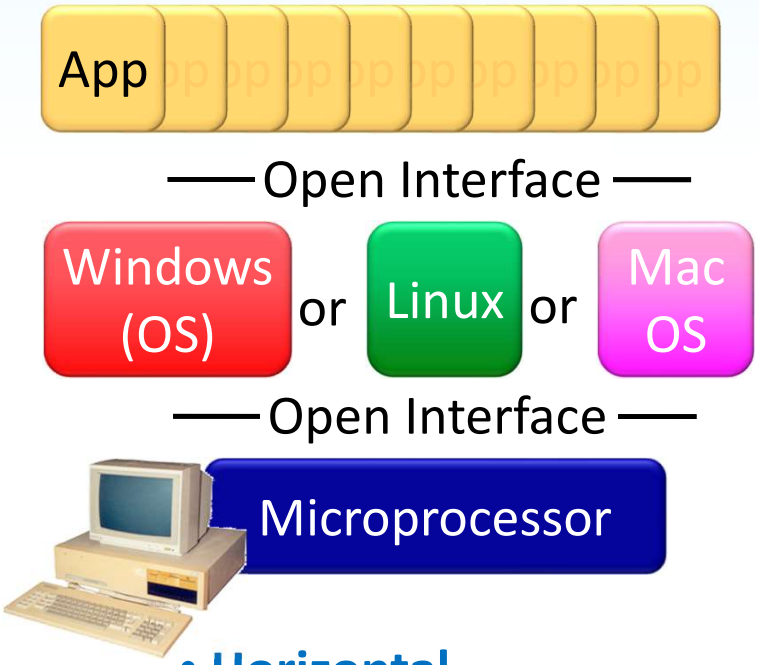
Department of Computer Science
National Yang Ming Chiao Tung University
cctseng@cs.nctu.edu.tw

Credited to:

1. Nick McKeown, Stanford
2. Scott Shenker, Berkeley
3. James Won-Ki Hong, POSTECH

Computer Evolution

- Vertically integrated
- Closed, proprietary
- Slow innovation
- Small industry



- Horizontal
- Open interfaces
- Rapid innovation
- Huge industry

The Current Network

Routing, management, mobility management, access control, VPNs, …

Feature · · · · Feature

Operating System

Million of lines of source code

5900 RFCs

Barrier to entry

Specialized Packet Forwarding Hardware

Billions of gates

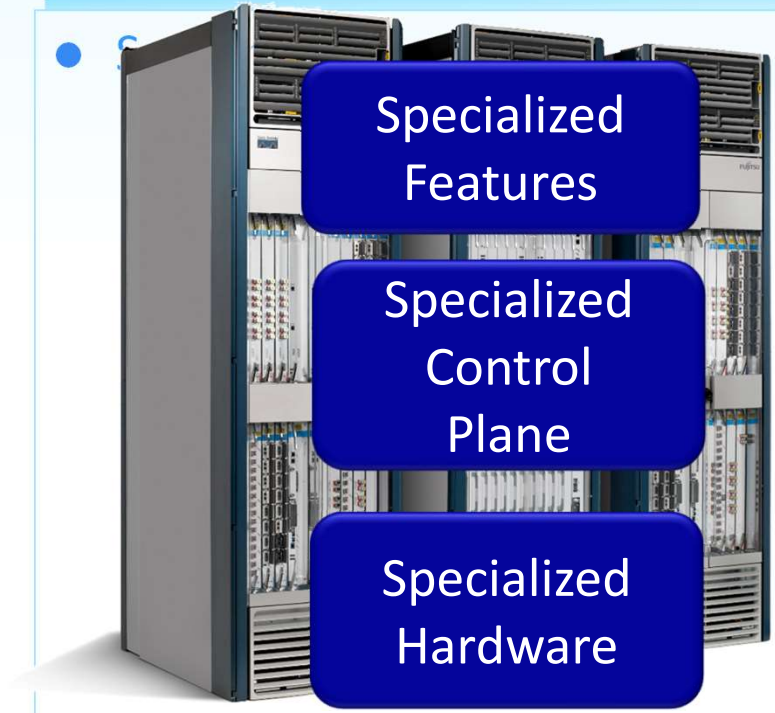
Bloated

Power Hungry

- Vertically integrated
- Many **complex functions** baked into **infrastructure**

  OSPF, BGP, multicast, differentiated services, Traffic Engineering, NAT, firewalls, MPLS, redundant layers, …
- **Looks** like **mainframe industry** in 1980s

Network Evolution



Vertically integrated
Closed, proprietary
Slow innovation

App

Open Interface

Open Network OS or Open Daylight or Control Plane

Open Interface

Merchant Switching Chips

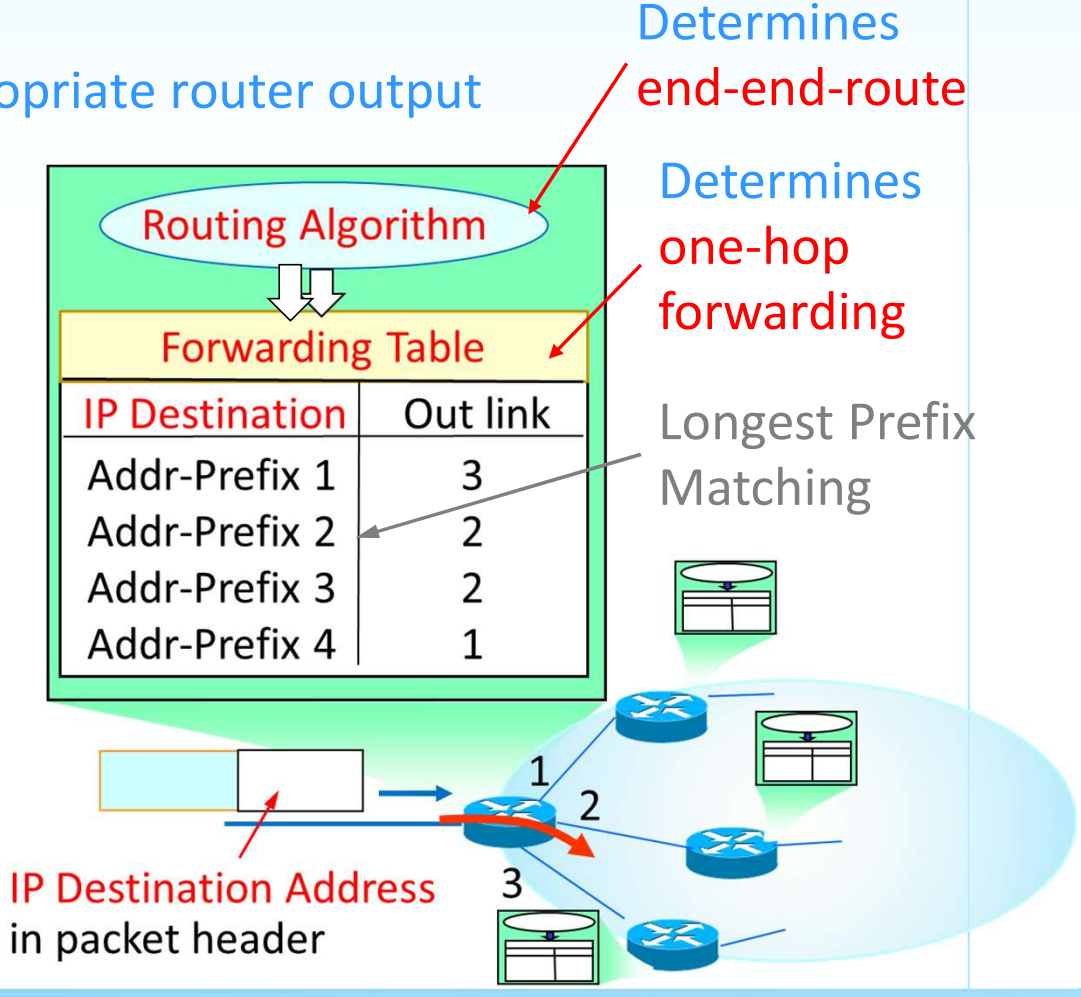
Horizontal
Open interfaces
Rapid innovation

Forwarding and Routing

1) Forwarding: data plane
   moves packets from router's input to appropriate router output
   - **Fast** time-scales (**per-packet**)

2) Routing: control plane
   determines route taken by packets
   from source to destination
   - **Slow** time-scales (**per control event**)

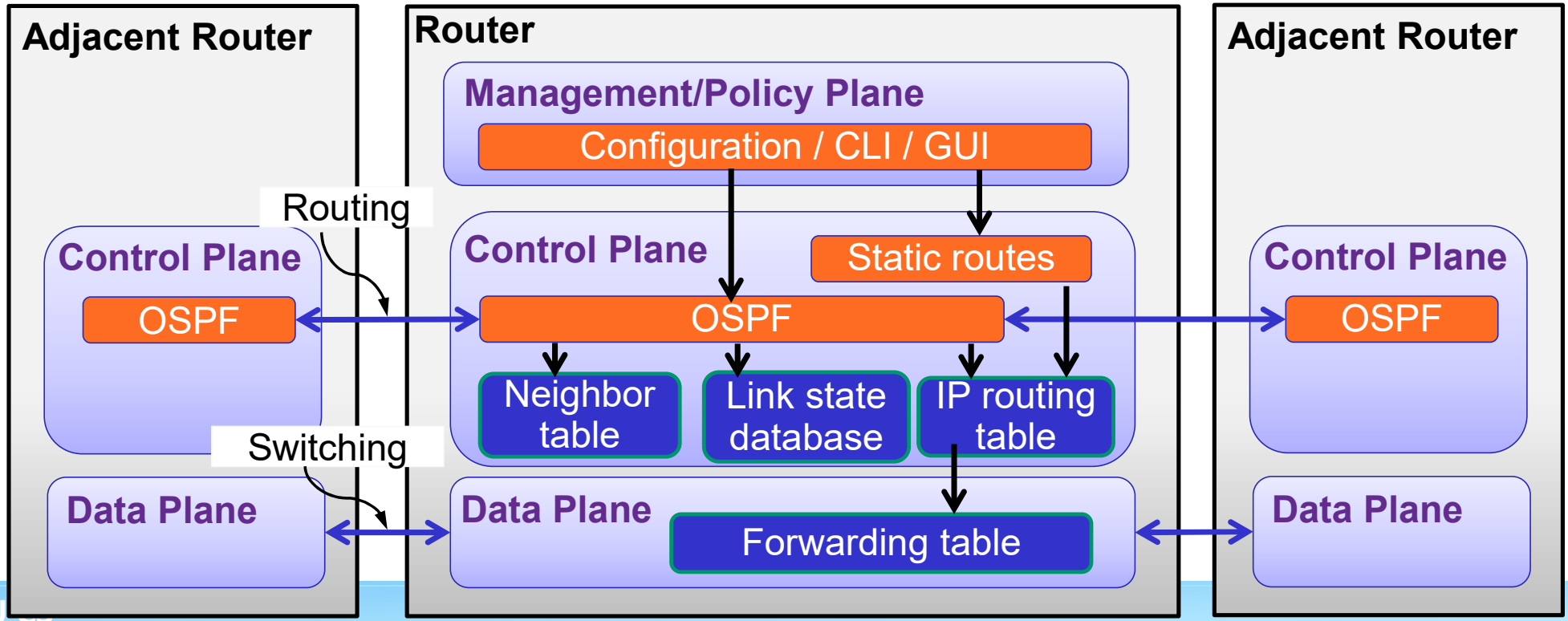
- Two approaches to structuring ***network control plane:***
  - Traditional
    - Per-router control
  - Software Defined Networking
    - Logically centralized control



Traditional Network Node

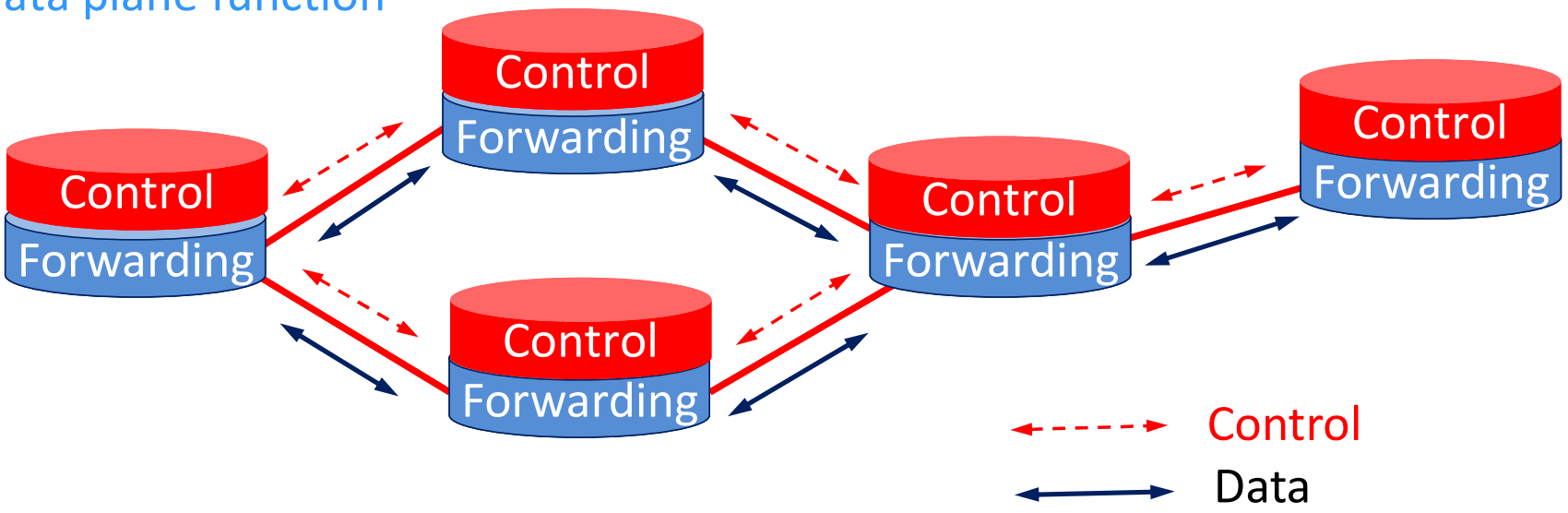
- Router: can be partitioned into three planes
  1. Management plane → Configuration
  2. Control plane → Make decision for the route
  3. Data plane → Data forwarding

Source: CSED702Y, James Won-Ki Hong, POSTECH



Traditional Networking

- Integrated Control and Data Planes
- Distributed Control
  - Distributed algorithm running between neighbors
  - Vender lock-in
- Fixed data plane function

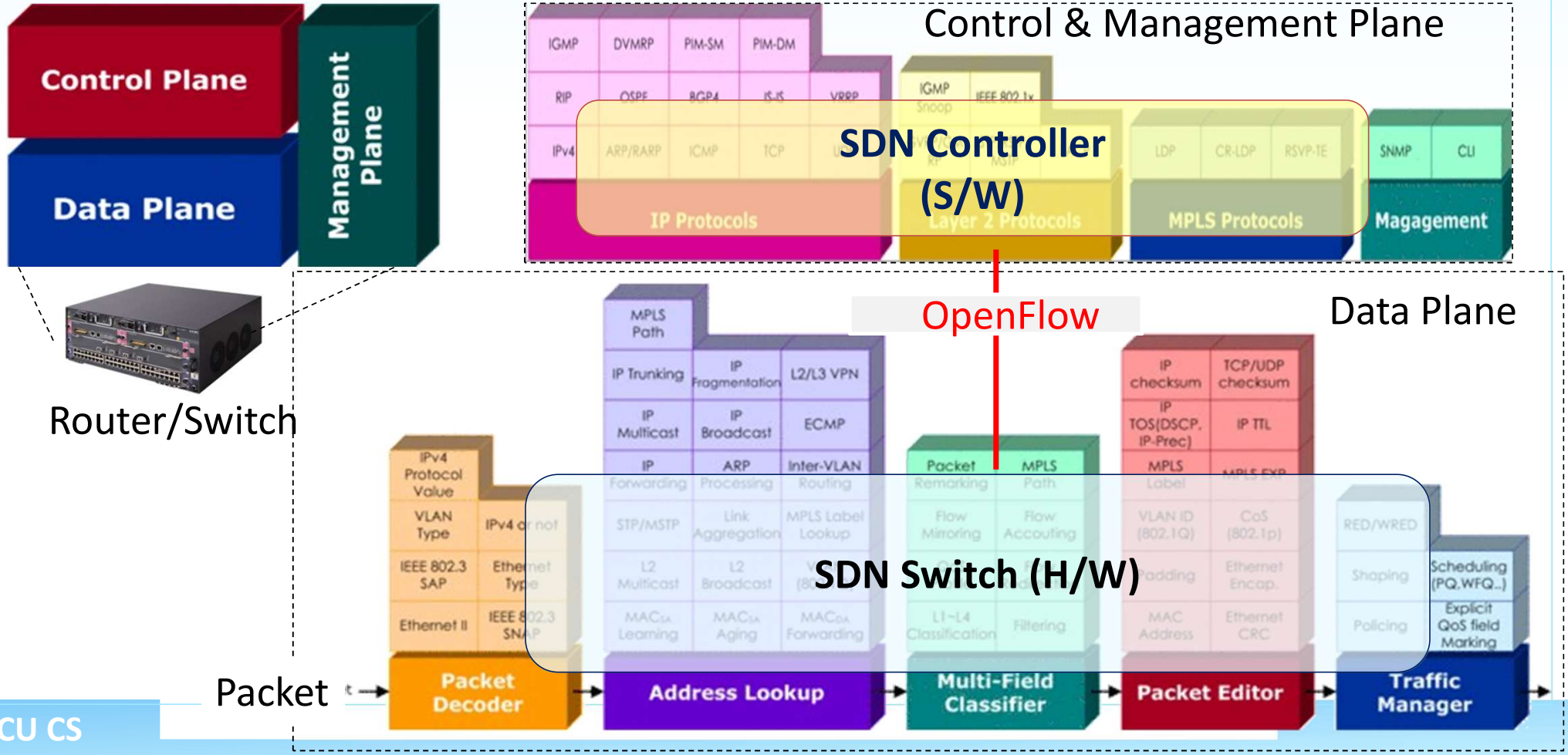


Credited to Prof. Scott Shenker (UC, Berkeley) and Prof.Nick Mckeown (Stanford University)

SDN Concept

- SDN separates control and data plane functions

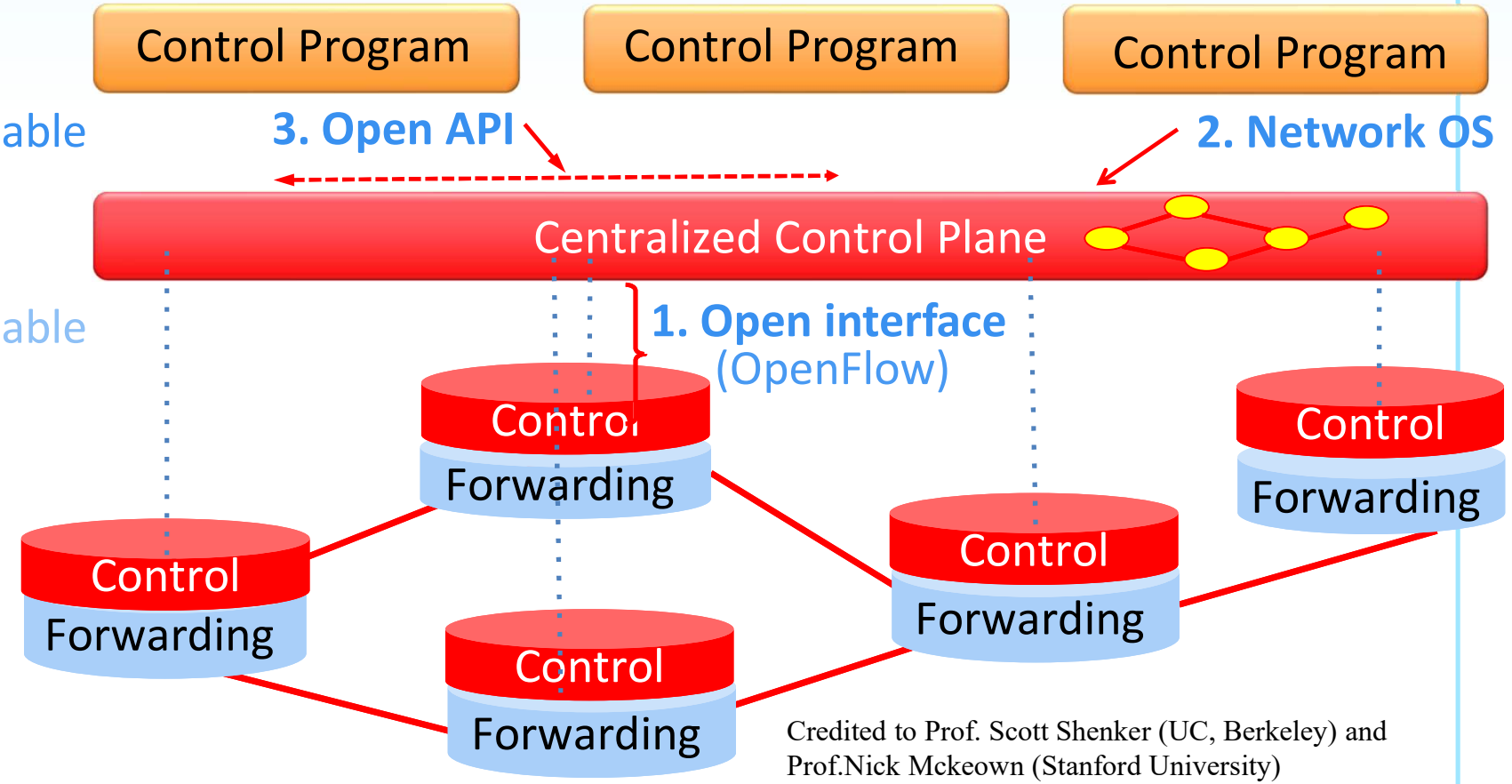
Source: CSED702Y, James Won-Ki Hong, POSTECH



Software Defined Network (SDN)

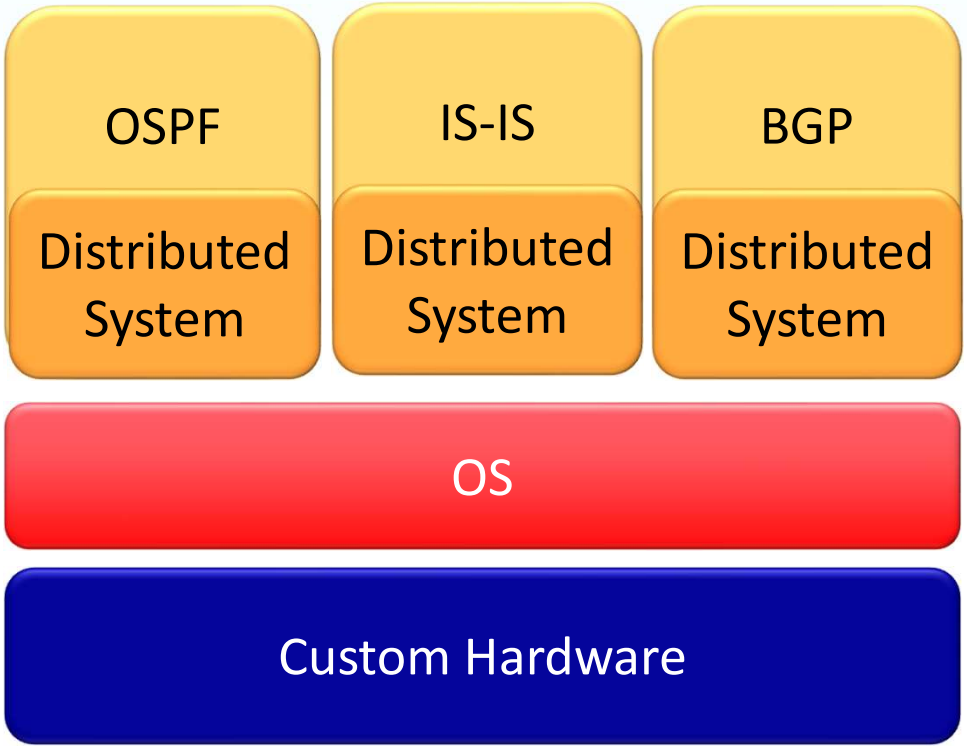
- Separation of Control Plane and Data Plane
- Centralized Control
  - Programmable
- Data plane function
  - Programmable

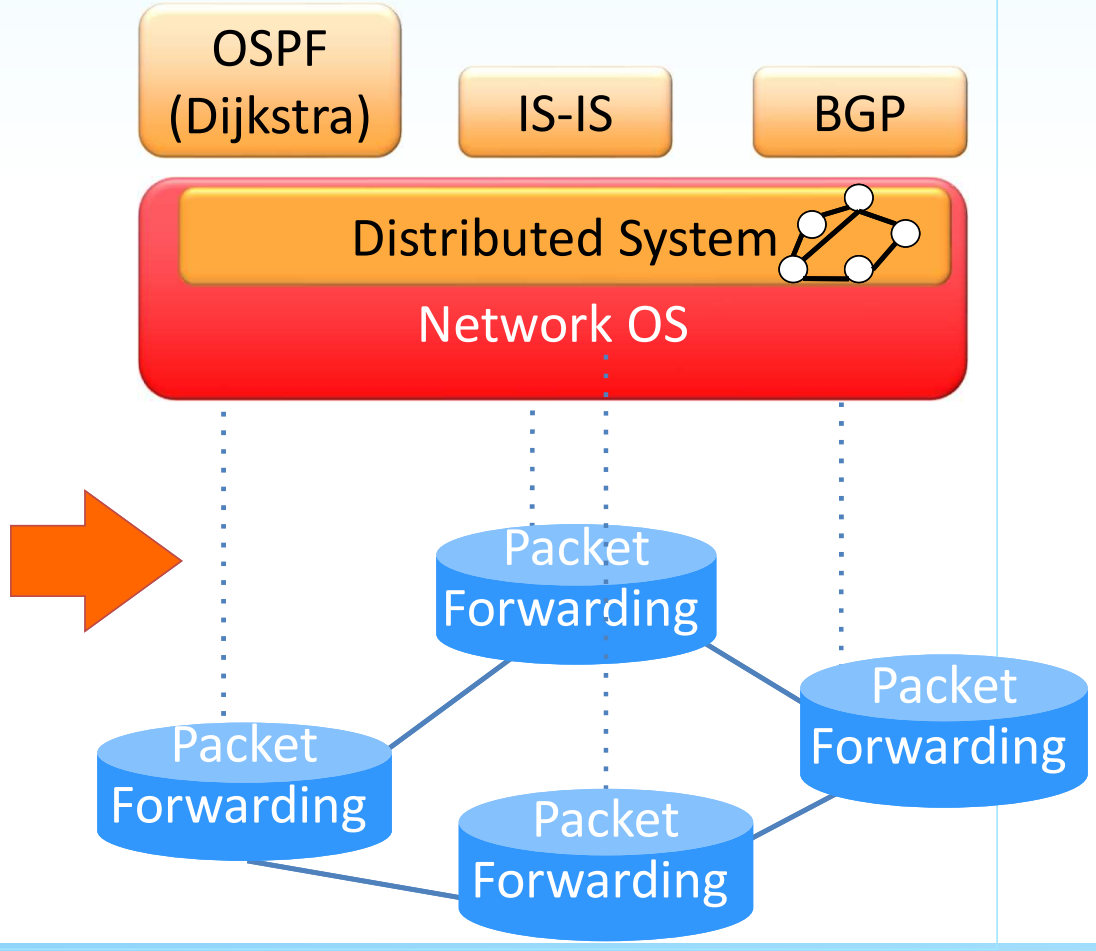
- Network owners take charge of their **control plane**



Credited to Prof. Scott Shenker (UC, Berkeley) and Prof.Nick Mckeown (Stanford University)

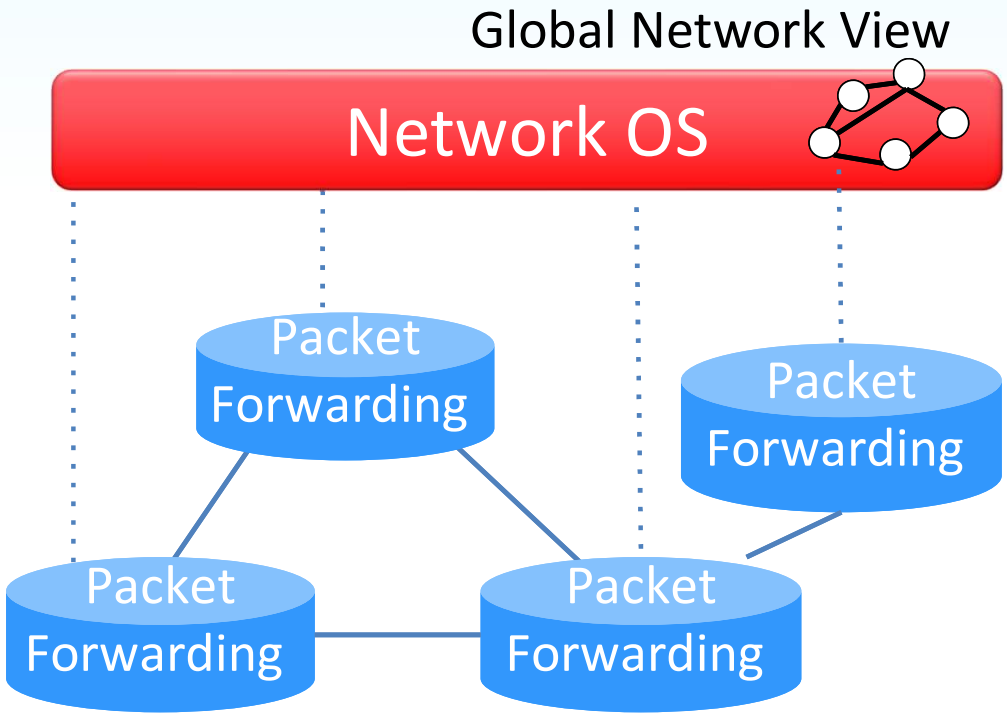
Centralized Control





Functions of Network OS

- **Network OS:**
  - Get state information from forwarding elements
  - Creates a consistent, up-to-date network view
  - Derive forwarding rules
  - Give control directives to forwarding elements

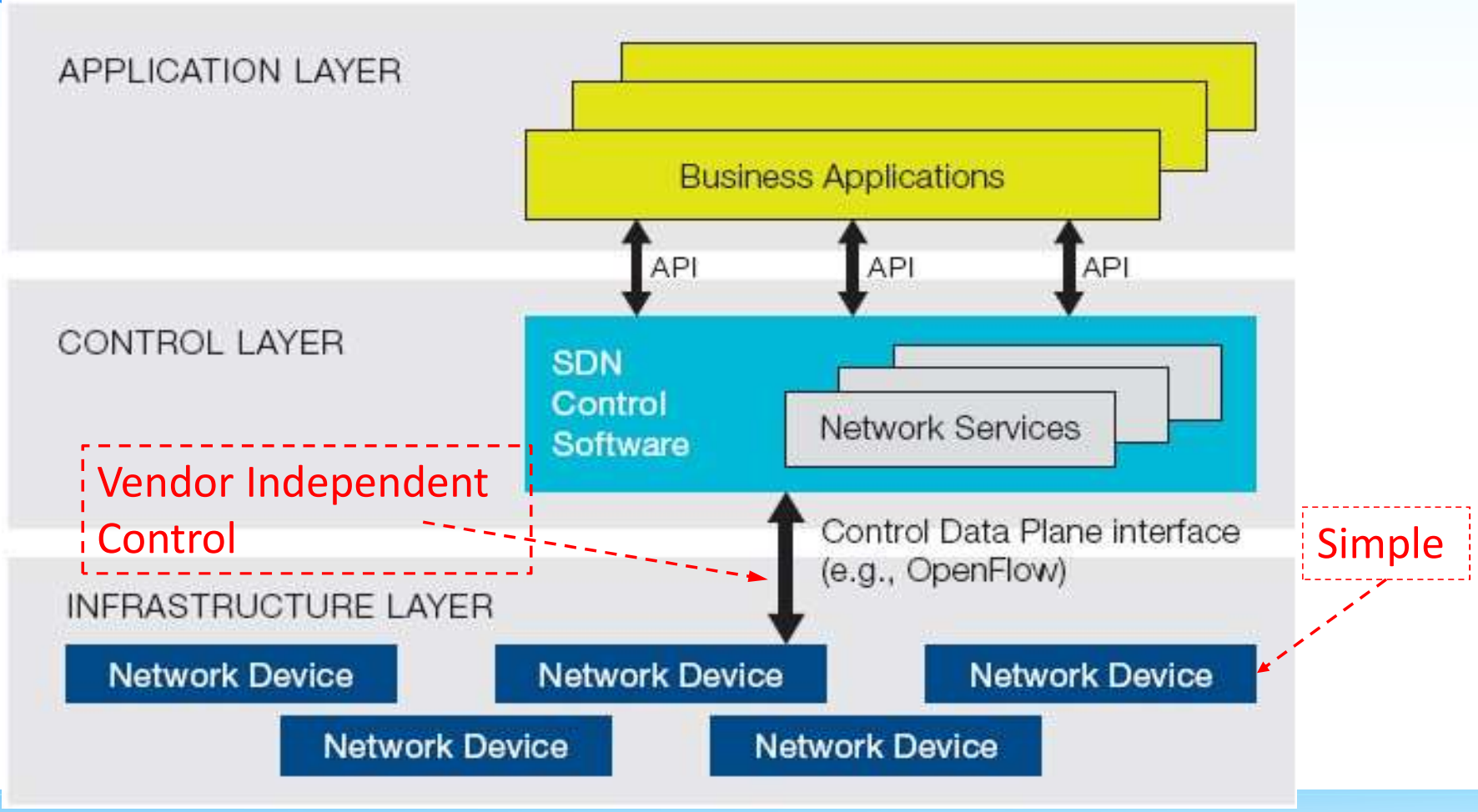


OpenFlow

Basics

Logical View of SDN architecture

- Separated Control and Data Planes

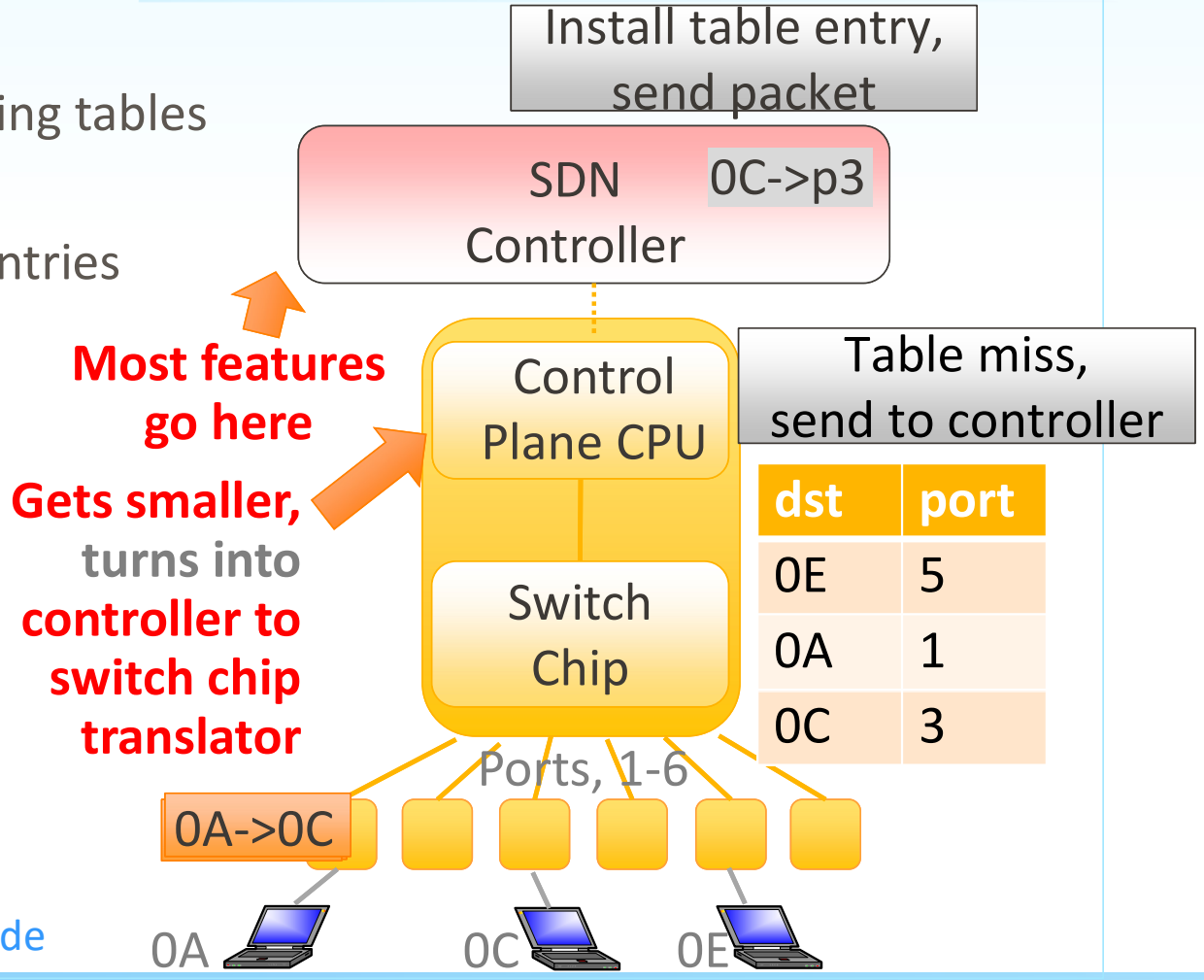


Software Defined Networks

Migrate the Control Plane to a Separate Controller

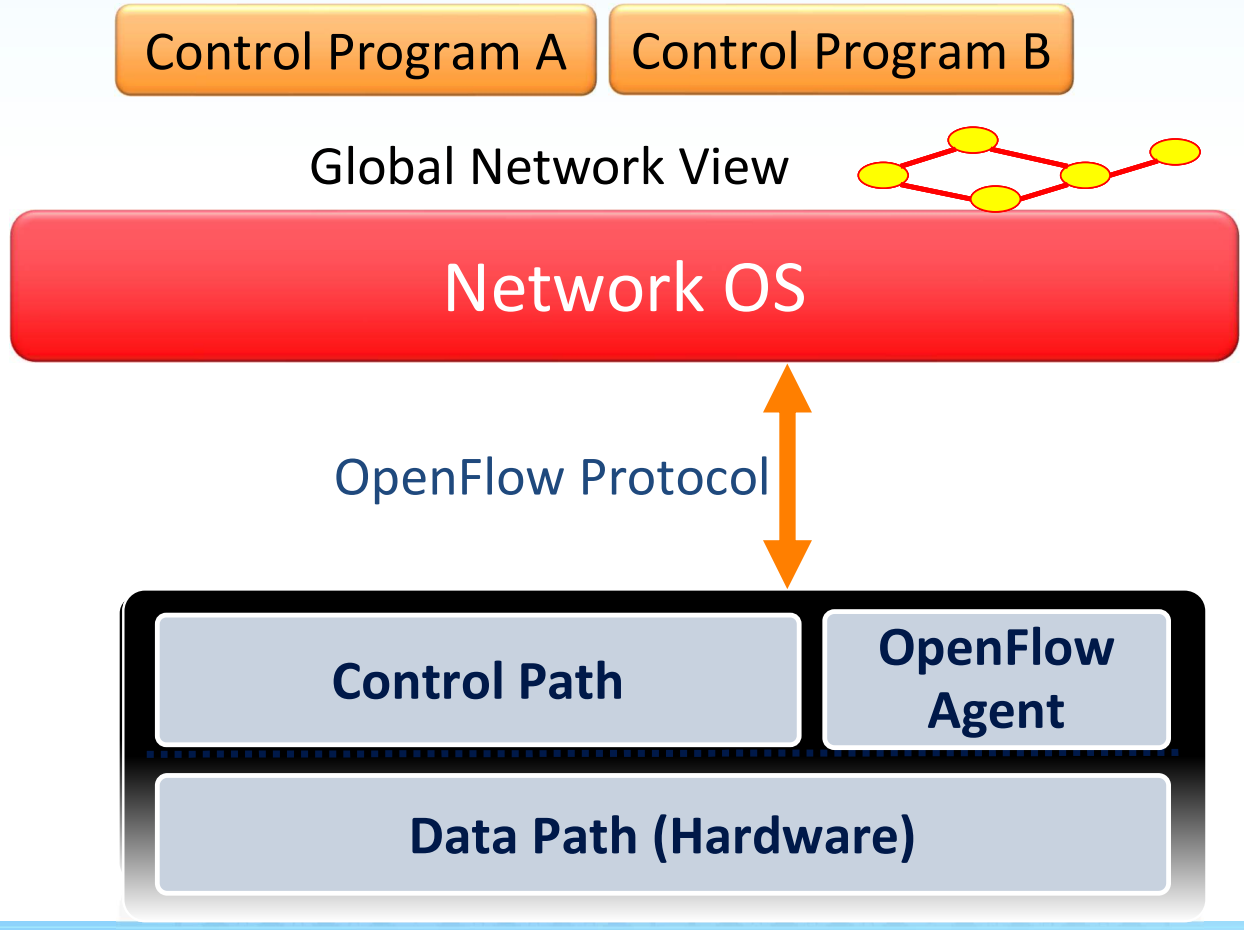
- Modern switches:
  - Control plane populates forwarding tables
    - Mac Learning
  - Data plane acts based on table entries
  - *Both run locally on switch*
- SDN
  - Decouple control plane from data plane
  - Data plane on switch
  - *Control plane elsewhere (Typically separated controller)*
  - *Example: OpenFlow Switch*

Source: Colin Dixon, Principal Engineer, Brocade



OpenFlow Basics – Architecture

- Separated Control Plane and Data Plane
- Controller
  - Network OS
  - Control Programs
- Switch
  - CPU
    - Control Plane
    - OpenFlow Agent
  - ASIC
    - Data Plane
- OpenFlow Protocol
  - Control Channel between Controller and Switch



SDN and OpenFlow

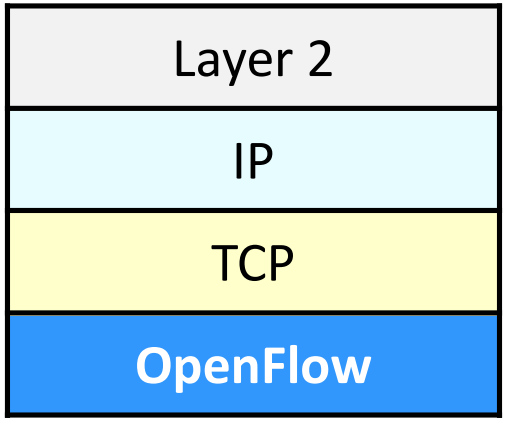
- OpenFlow is not equivalent to SDN
  - OpenFlow is one of Control-Data plane Protocols (or Interfaces)
    - ➢ No requirement for SDN
- Evolution of OpenFlow Specifications

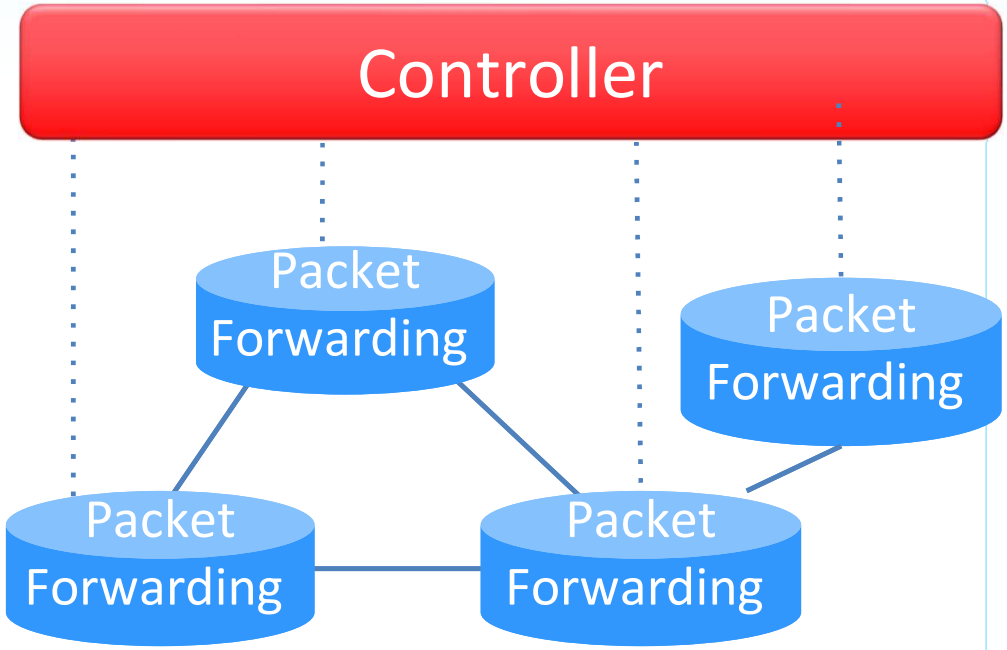
Cons: Consortium
Org.: Organization

Version	Date	No. of Headers	Characteristics	Org.
1.0	2009.12	12	MAC, IPv4, single flow table	OF Cons.
1.1	2011.2	15	MPLS/tunnel, multiple flow tables, group table	OF Cons.
1.2	2011.12	36	IPv6, Config., extensible match support	ONF
1.3	2012.9	40	QoS (meter table)…	ONF
1.4	2013.10	41	Optical port monitoring and config. (frequency, power)	ONF
1.5	2014.12	42	Egress table, pkt. type aware pipeline, flow entry statatistic trigger	ONF

OpenFlow Channel

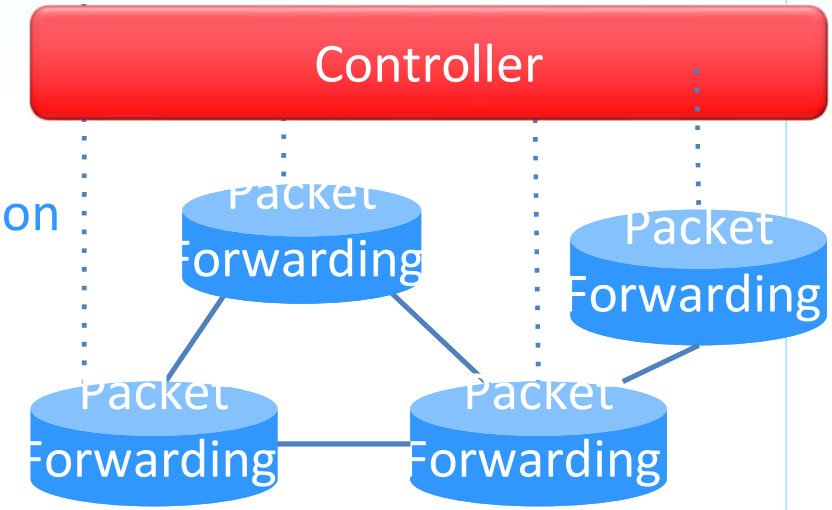
- OpenFlow channel uses TLS or plain TCP,
  - On default port 6653



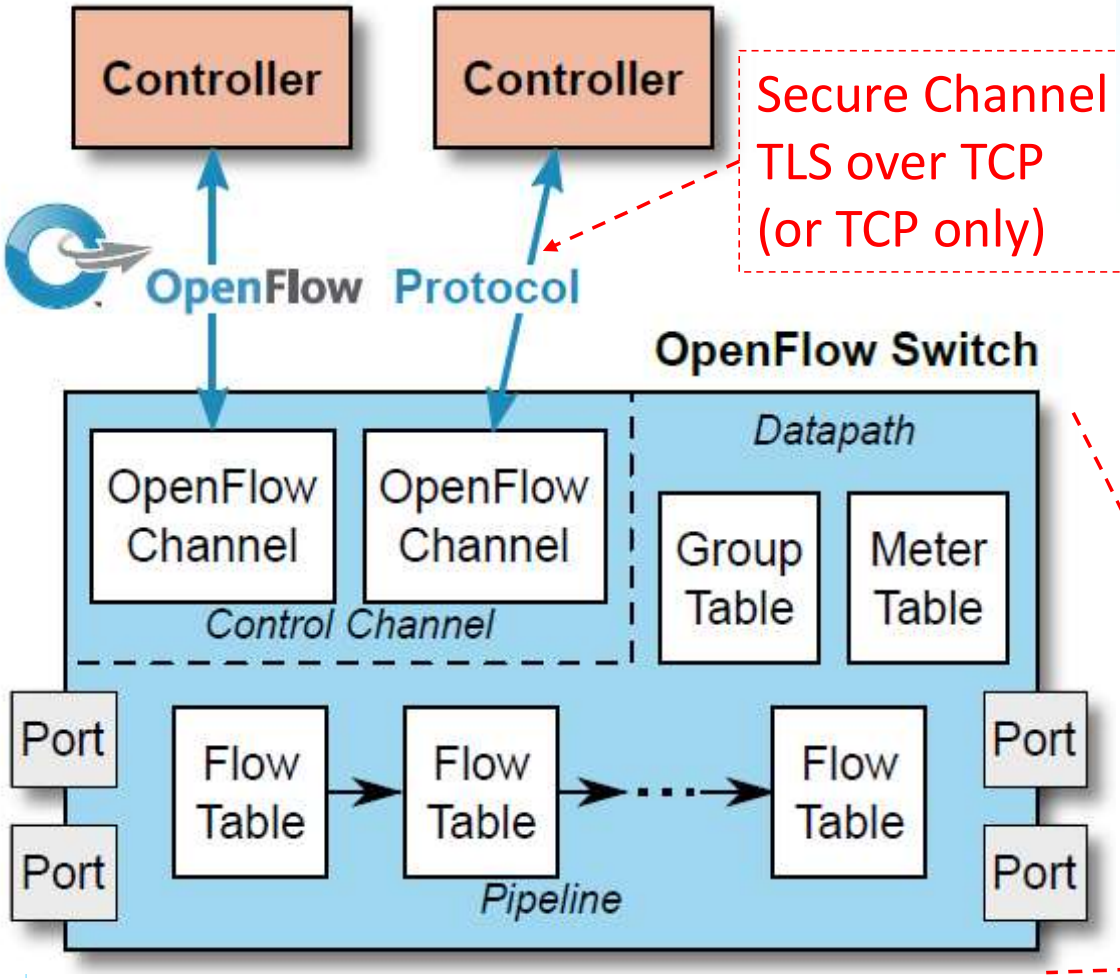


OpenFlow Control Channels

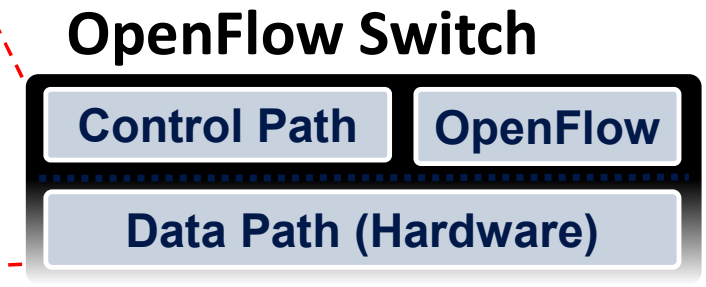
- Two Types:
  - **Out-of-band** controller connection,
    - Separated control and data connection
  - **In-band** controller connection
    - Uses data plane network for control connection
- **An OpenFlow Controller** may
  - Manage multiple OpenFlow channels,
    - each to a different OpenFlow switch.
- **An OpenFlow Switch** may have
  - One OpenFlow channel to a single controller, or
  - Multiple channels to multiple controllers
    - each to a different controller, SLAVE or MASTER, for reliability



Main Components of OpenFlow Switch

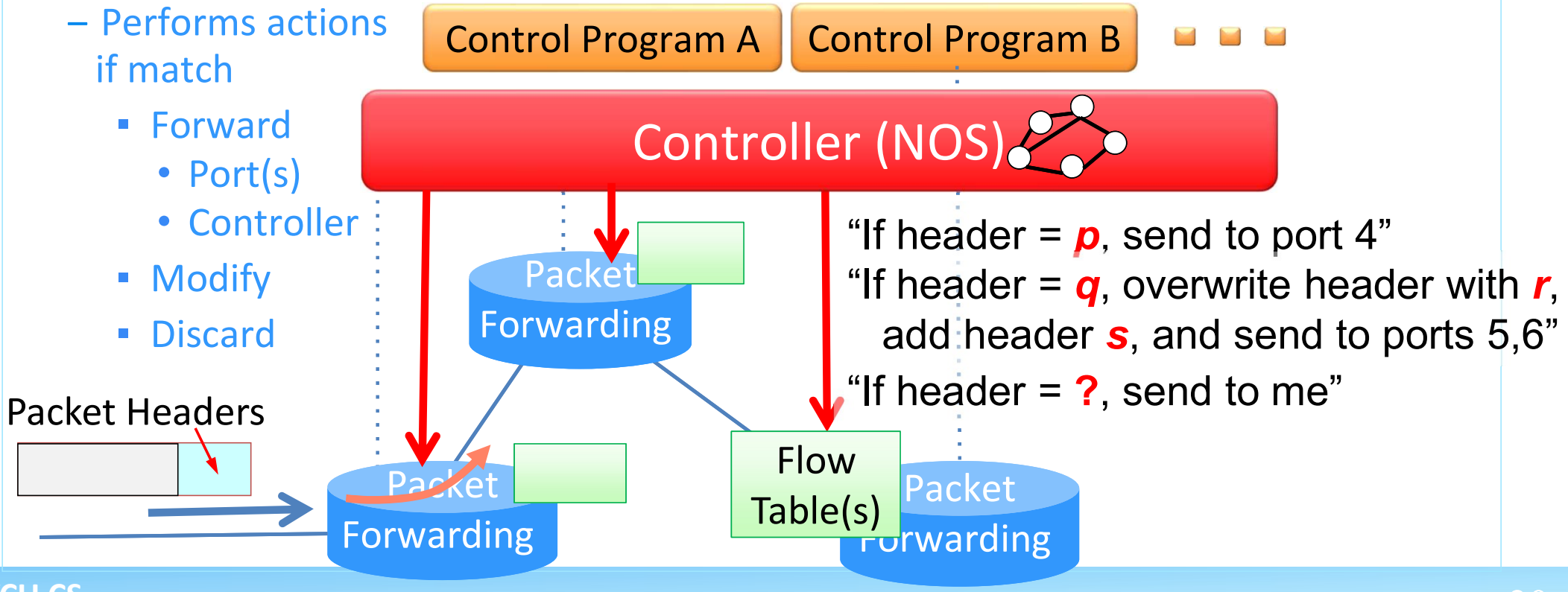


- **Pipeline**: set of linked Flow Tables
- **Flow Table**: stage of the pipeline
  - Consists of **Flow Entries**
- **Flow entry**:
  - (Match Fields, Actions)
- Group Table, Meter Table



OpenFlow Basics – Operation Concept

- Controller installs flow rules (flow entries) proactively or reactively
  - Flow entries: (Match, Actions)
- Switch compares Match fields with incoming packet headers
  - Performs actions if match
    - Forward
      - Port(s)
      - Controller
    - Modify
    - Discard



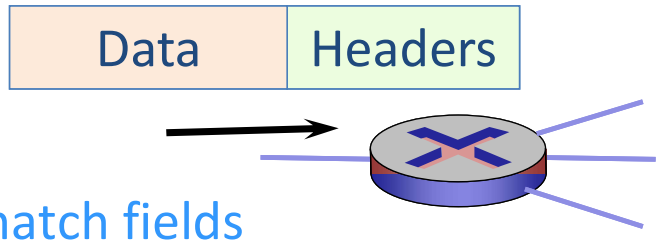
OpenFlow – Plumbing Primitives <*Match*, *Action*>

- **Flow Entries (Flow Rules):**

Match Fields	Instructions (Actions)

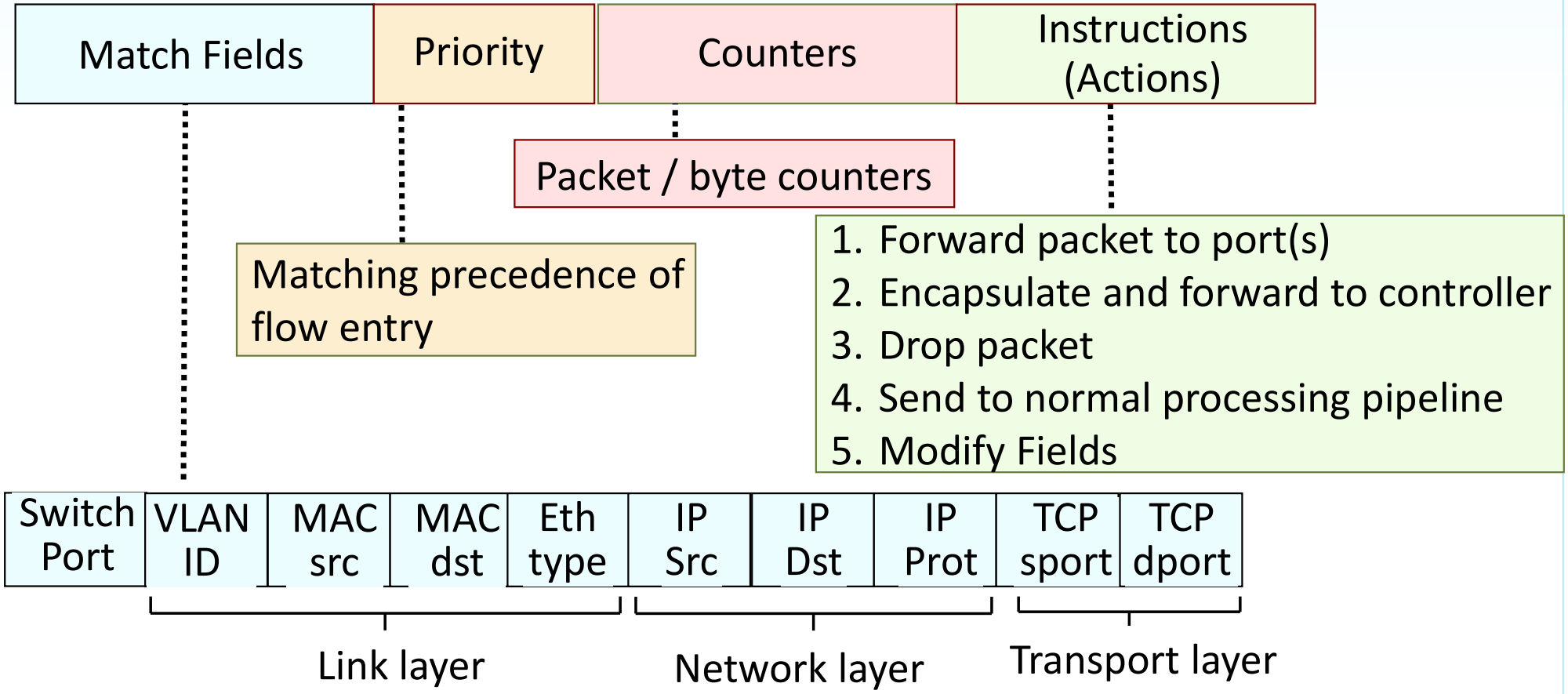
  - ***Match*** fields**:**
    a part of a flow entry which switch uses to match against a packet.
    - Can match various packet header fields
      - E.g., Ether Dst/Src MAC, IP Dst/Src, TCP Dst/Src Port, VLAN ID, …
  - ***Instructions*** (*Action*) fields**:**
    - Forward to port(s), drop, send to controller
    - Overwrite header with mask, push or pop
    - Forward at a specific bit-rate

Incoming Packet:



- Flows: defined by header fields, or more precisely by match fields
  - May be five-tuple flows, aggregated flows
    - Allows any flow granularity

OpenFlow: Flow Table Entries (1st Look)



Examples

- **Destination-based forwarding:**
  - *IP datagrams destined to IP address 5.6.7.8 forwarded to router output port 6*

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	*	*	*	*	5.6.7.8	*	*	*	port6

- **Firewall:**
  - *do not forward (**block**) all datagrams **destined to TCP port 22***

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	*	*	*	*	*	*	*	22	drop

  - *do not forward (**block**) all datagrams **sent by host 128.119.1.1***

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	*	*	*	128.119.1.1	*	*	*	*	drop

More Examples

- **Source-based layer 2 (switch) forwarding**:
  - *layer 2 frames from MAC address 22:A7:23:11:E1:02 forwarded to output port* 3

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	22:A7:23: 11: E1:02	*	*	*	*	*	*	*	*	port3

- **VLAN Switching**
  - L2 frames destined to Mac 00:1f.. In VLAN 1 flooded to ports 6, 7, 9

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	00:1f..	*	vlan1	*	*	*	*	*	port6, port7, port9

OpenFlow Abstraction

- *Match+Action:* unifies different kinds of devices

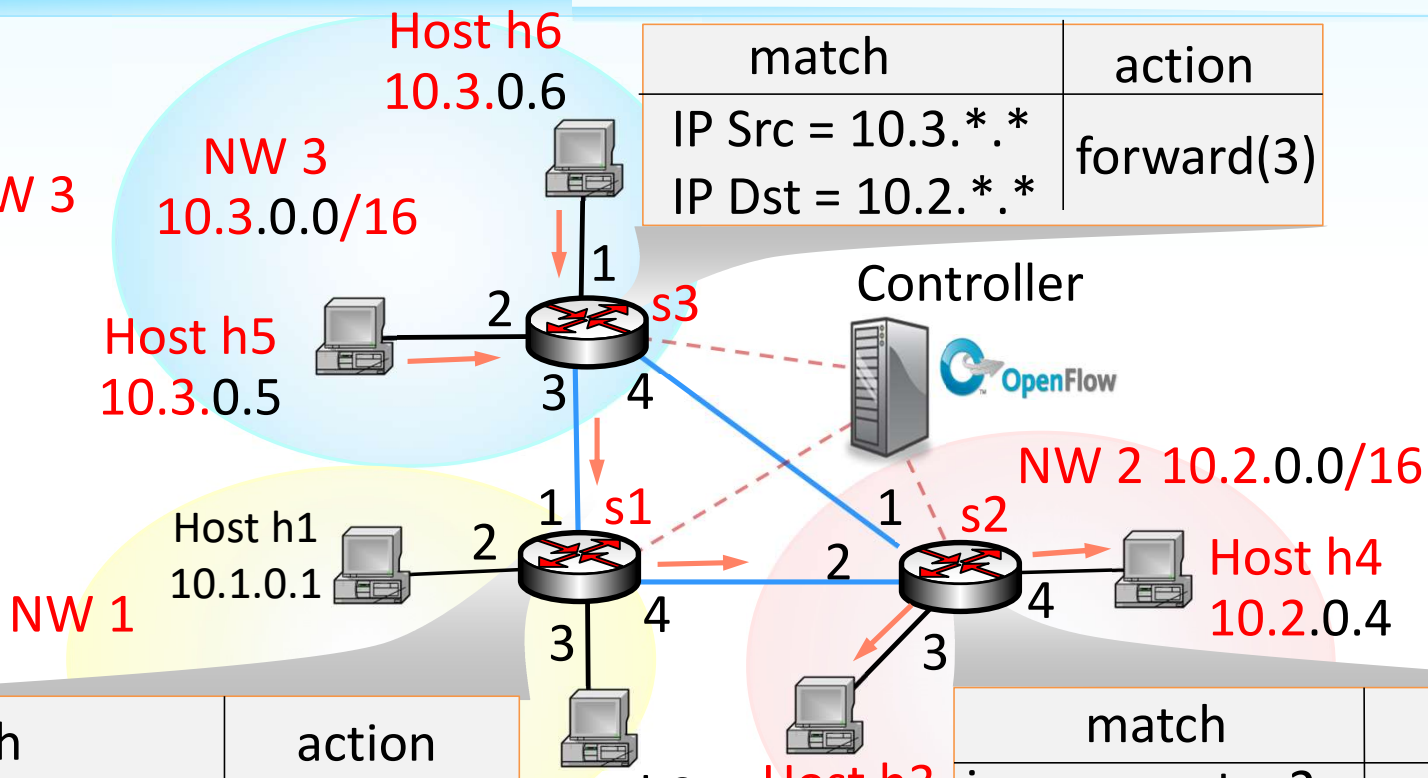
- **Router**
  - *match:* longest destination IP prefix
  - *action:* forward out a link
- **Switch**
  - *match:* destination MAC address
  - *action:* forward or flood

- **Firewall**
  - *match*: IP addresses and TCP/UDP port numbers
  - *action:* permit or deny

✓ **NAT**
  - *match:* IP address and port
  - *action:* rewrite address and port

OpenFlow Example

- *Example:*
  datagrams sent
  from hosts in NW 3
  to hosts in NW2
  - Must via s1



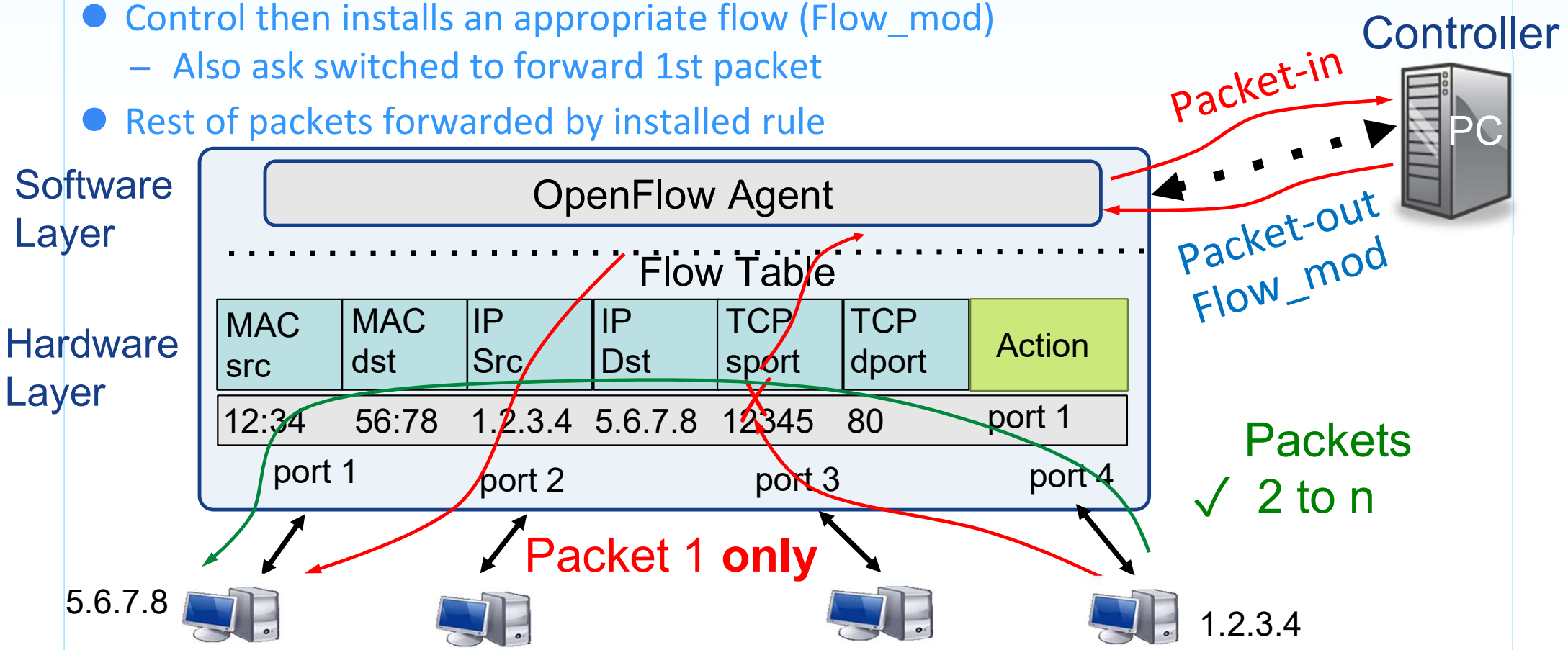
match	action
IP Src = 10.3.*.* IP Dst = 10.2.*.*	forward(3)

match	action
ingress port = 1 IP Src = 10.3.*.* IP Dst = 10.2.*.*	forward(4)

match	actio
ingress port = 2 IP Dst = 10.2.0.3	n forward(3)
ingress port = 2 IP Dst = 10.2.0.4	forward(4)

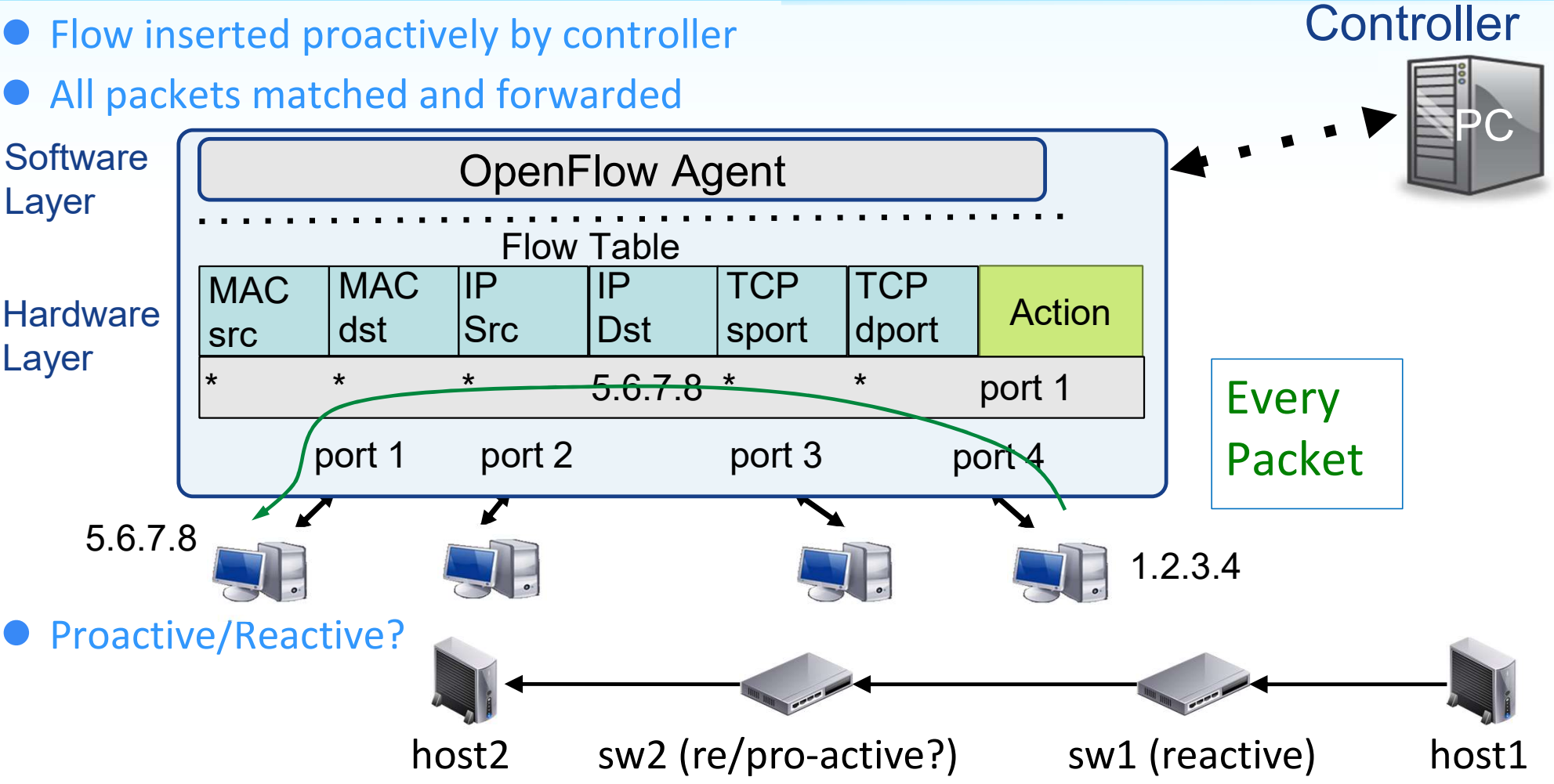
Reactive Packet Processing

- First non-matched packet sent to controller
- Control then installs an appropriate flow (Flow_mod)
  - Also ask switched to forward 1st packet
- Rest of packets forwarded by installed rule



Proactive Packet Processing

- Flow inserted proactively by controller
- All packets matched and forwarded



- Proactive/Reactive?

OpenFlow Protocol Format

- OpenFlow control message relies on TCP protocol, on default Port 6653
- OpenFlow Message Structure
  - Version
  - Type (version dependent)
  - Message length (starting from 1st byte of header)
  - Transaction ID (xid): unique value used to match requests to response

OFPT_HELLO = 0 (Symmetric)
OFPT_ERROR = 1 (Symmetric)
OFPT_PACKET_IN = 10, (Asynchronous)
OFPT_FLOW_REMOVED = 11 (Async.)

- **OpenFlow Message Structure**

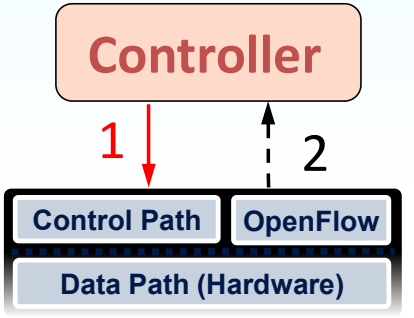
OFPT_PACKET_OUT = 13 (Controller-to-switch)
OFPT_FLOW_MOD = 14 (Controller-to-switch)

Bit Offset	0 ~ 7	8 ~ 15	16 ~ 23	24 ~ 31
0 ~ 31	Version	Type	Message Length	
32 ~ 63	Transaction ID			
64 ~ ?	Payload			

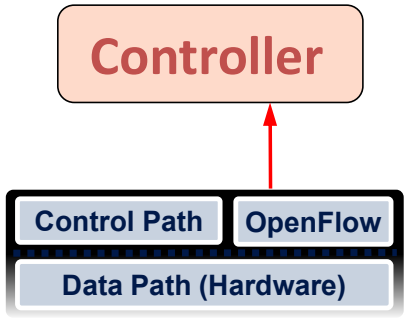
Types of OpenFlow Messages

- **Three types of OF messages**
  controller-to-switch, ***asynchronous***, and ***symmetric***

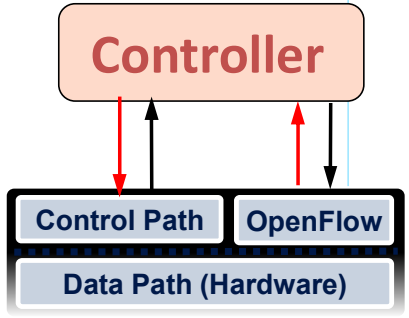
1. **Controller-to-switch messages**: initiated by **controllers**
   - used to manage or inspect state of switch.
   - may or may not require a response
2. **Asynchronous messages**: initiated by **switches**
   - without controller solicitations
   - Used to report to controller
     - Network events (Packet-INs) and
     - Switch state change.
3. **Symmetric messages**: in **either direction**, **without solicitation**



Controller-to-Switch



Asynchronous



Symmetric

Connection Setup and Topology Discovery

Source: CSED702Y, James Won-Ki Hong, POSTECH