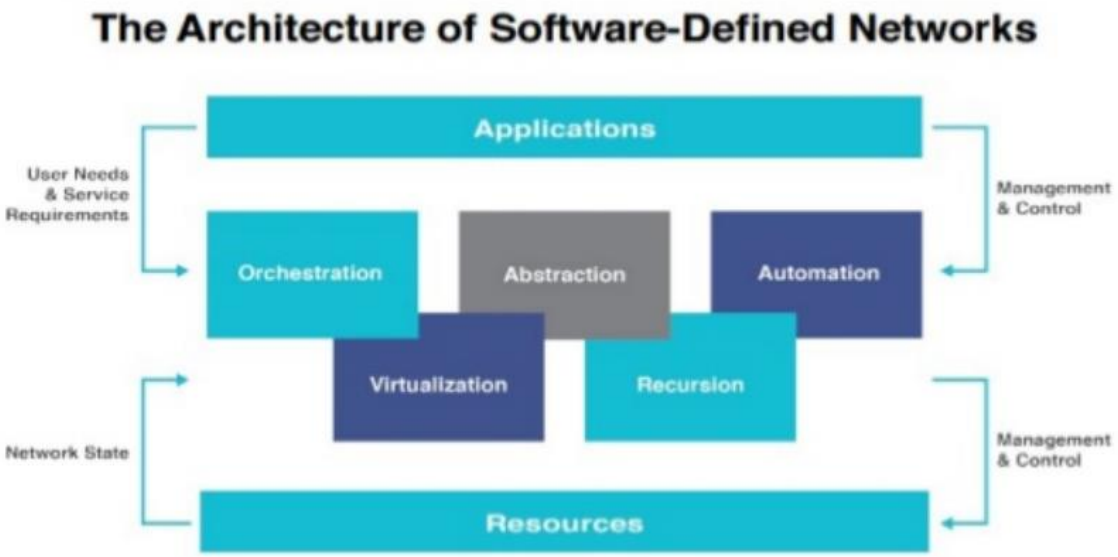
Software Defined Networks and OpenFlow

Mestrado Integrado em Engenharia de Computadores e Telemática

2019/2020

Need for Programmable Networks

- Changing traffic patterns
- Rise of cloud services
- ‘Big data’ requires more bandwidth
- Inability to scale
- Need for open programmable networks
- Open standards-based and vendors-neutral



Characteristics of SDN

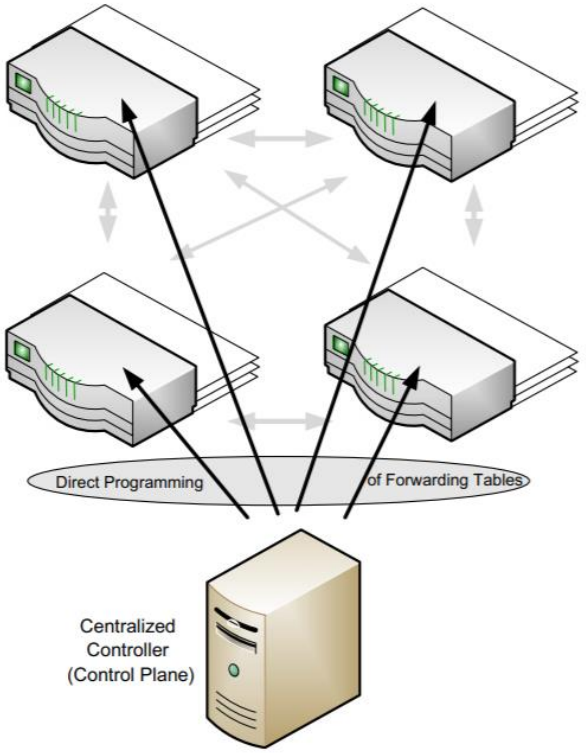
- Plane separation
- Simplified device
- Centralized control
- Network automation and virtualization
- Openness

Plane Separation

- Separation of the forwarding and control planes
- Forwarding plane:
  - logic and tables for choosing how to deal with incoming packets based on characteristics such as MAC address, IP address, and VLAN ID
  - Forward, drop, consume, or replicate an incoming packet
  - Device determines the correct output port by performing a lookup in the address table in the hardware
  - Special-case packets that require processing by the control or management planes are consumed and passed to the appropriate plane

Plane Separation

- Control plane
  - Determines how the forwarding tables and logic in the data plane should be programmed
  - Run routing or switching protocols
  - Performed in a centralized controller
  - Can be co-located in the switches



Simplified Device and Centralized Control

- Control software is removed from the device and placed in a centralized controller
- Software-based controller manages the network using higher-level policies
- Controller provides primitive instructions to the simplified devices when appropriate
  - Allow devices to make fast decisions about how to deal with incoming packets.

Network Automation and Virtualization

- SDN is said to be a natural evolution for the problem of network control
- Distributed state abstraction provides the network programmer with a global network view
  - Specify the necessary forwarding behaviors without any knowledge of vendor-specific hardware
  - Express the desired goals of the overall network without getting lost in the details of how the physical network will implement those goals
- Open interface on the controller to allow for automated control of the network
  - Northbound and southbound: interface is to the applications or to the devices

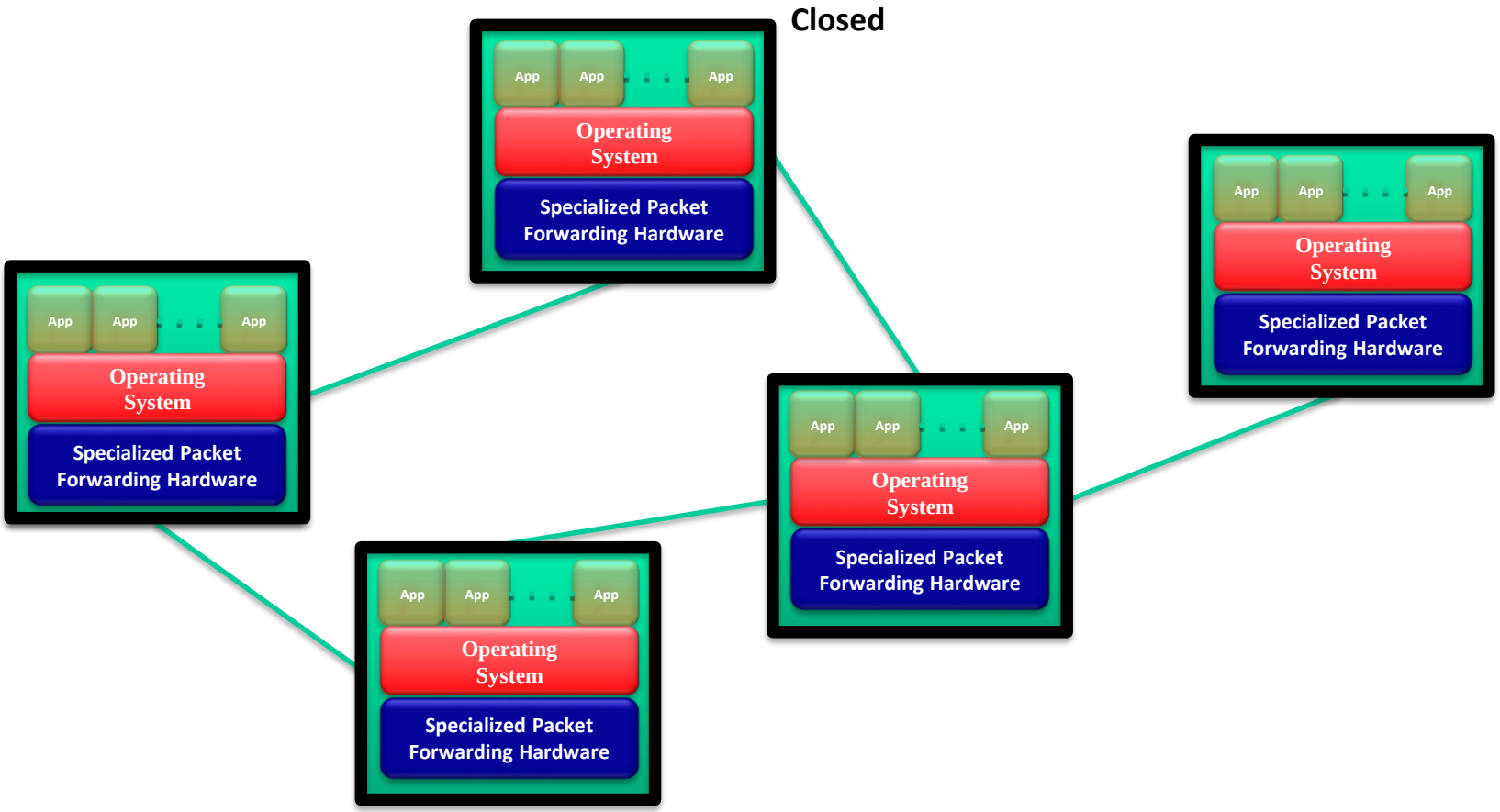
Network Automation and Virtualization

- Southbound API is the OpenFlow interface to program the network devices
- Northbound API for software applications to be plugged into the controller
  - Algorithms and protocols
  - Can quickly and dynamically make network changes as the need arises
  - Interface that allows the software above it to operate without knowledge of the individual characteristics and of the network devices themselves
  - Applications can be developed that work over a wide array of manufacturers' equipment that may differ substantially in their implementation details.
  - Ability to virtualize the network, decoupling the network service from the underlying physical network
    - Unaware if network resources are virtual or physical

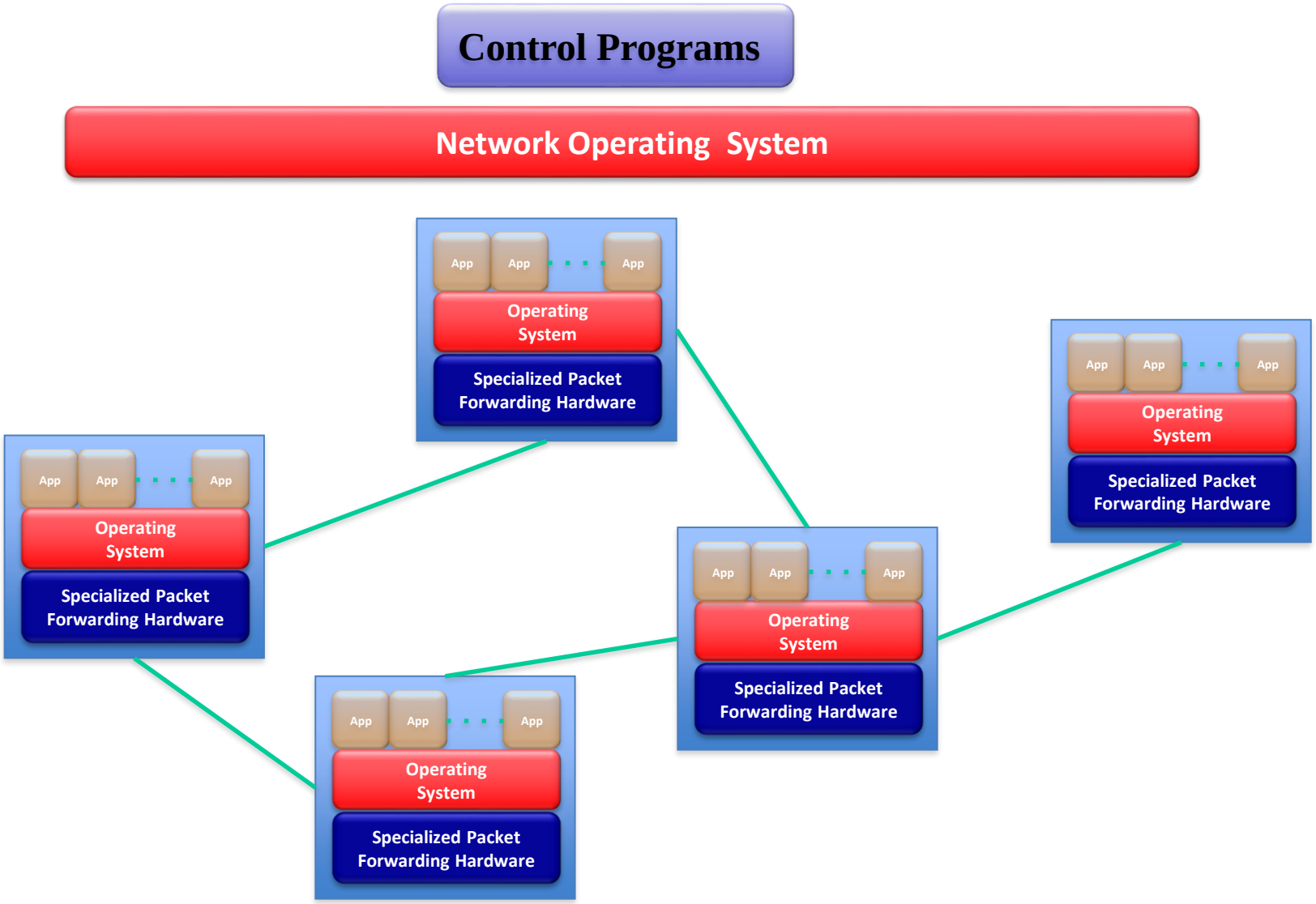
Openness

- Standard interfaces should remain standard, well documented, and not proprietary
- APIs to give software sufficient control to experiment with and control various control plane options
- Northbound and southbound interfaces
  - Easily experiment with and test new ideas
  - Resulting in better and faster technological advancement in the structure and functioning of networks
  - Open interfaces also encourages SDN-related open source projects, and permit equipment from different vendors to interoperate

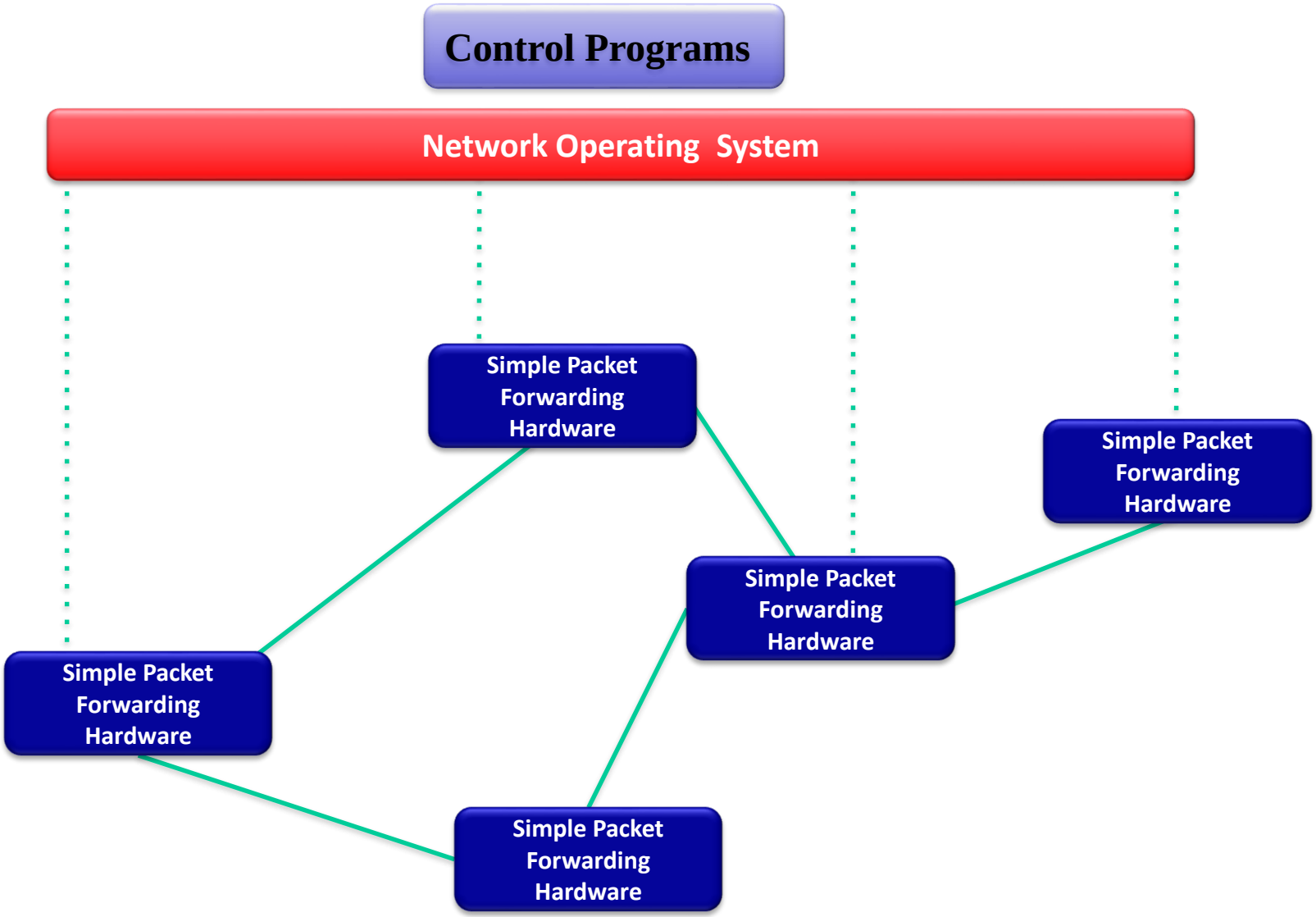
Idea: An OS for Networks



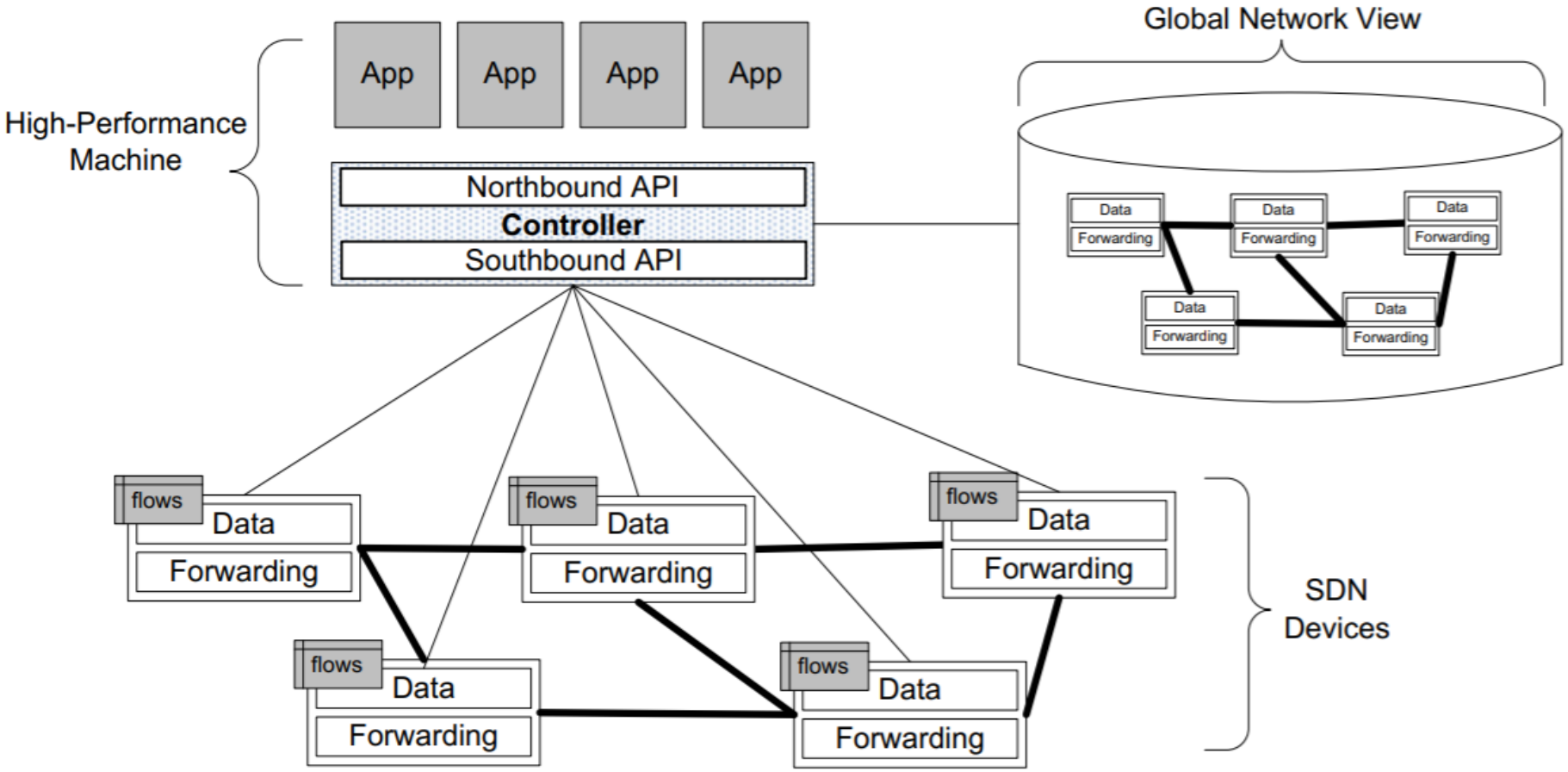
Idea: An OS for Networks



Idea: An OS for Networks



SDN: Architecture and Operation



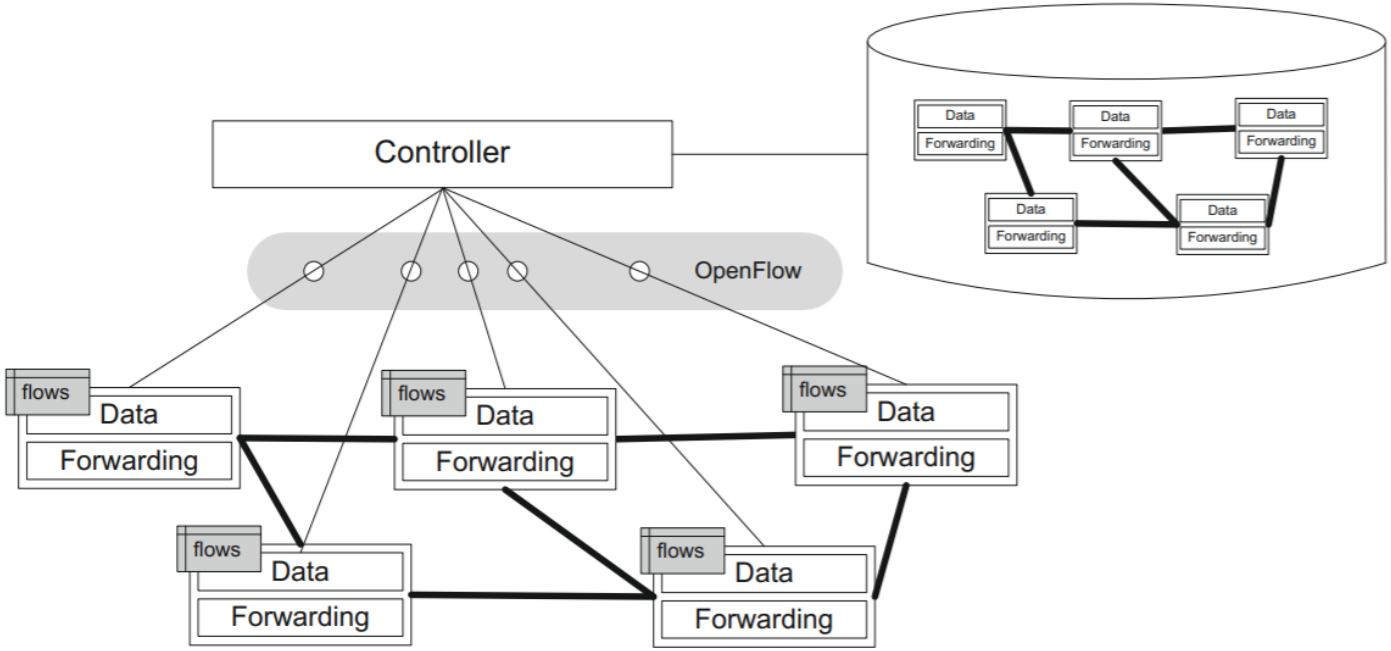
Flow describes a set of packets transferred from one network endpoint (or set of endpoints) to another endpoint (or set of endpoints).

SDN Operation: Flows

- IP address-TCP/UDP pairs, VLAN endpoints, layer three tunnel endpoints, and input ports, among other things
- Rules describes the forwarding actions that the device should take for all packets belonging to that flow
- Flows are unidirectional, opposite direction is a separate flow
- Flow table: series of flow entries and the actions to perform when a packet matching that flow arrives at the device; if it does match any flow, it is discarded or decision does to the controller

SDN Operation: Controller

- Allows the SDN application to define flows on devices
- Helps the application respond to packets that are forwarded to the controller by the SDN devices
- Calculates optimal forwarding solutions for the network in a deterministic, predictable manner



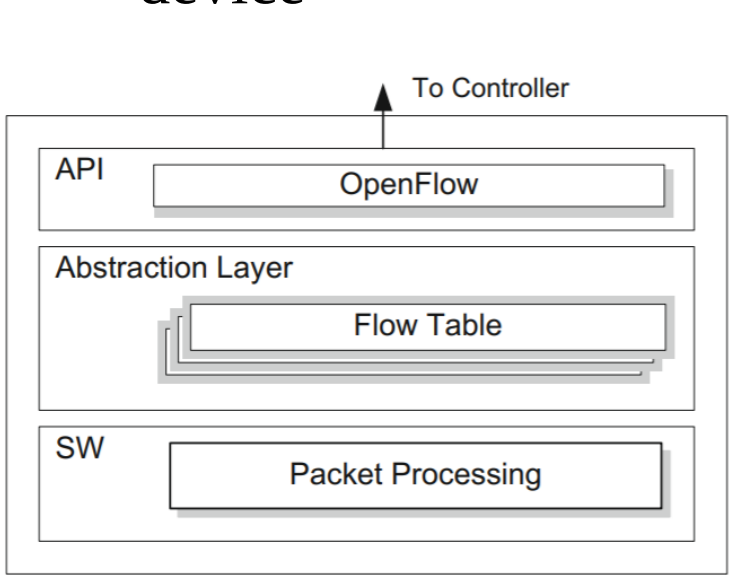
SDN Operation: Applications

- Part of network layers two and three
- Use SDN controller to set proactive flows on the devices and to receive packets that have been forwarded to the controller
- Flows defined in response to a packet forwarded to the controller
  - SDN application will instruct the controller as to how to respond to the packet
  - If appropriate, will establish new flows on the device in order to allow

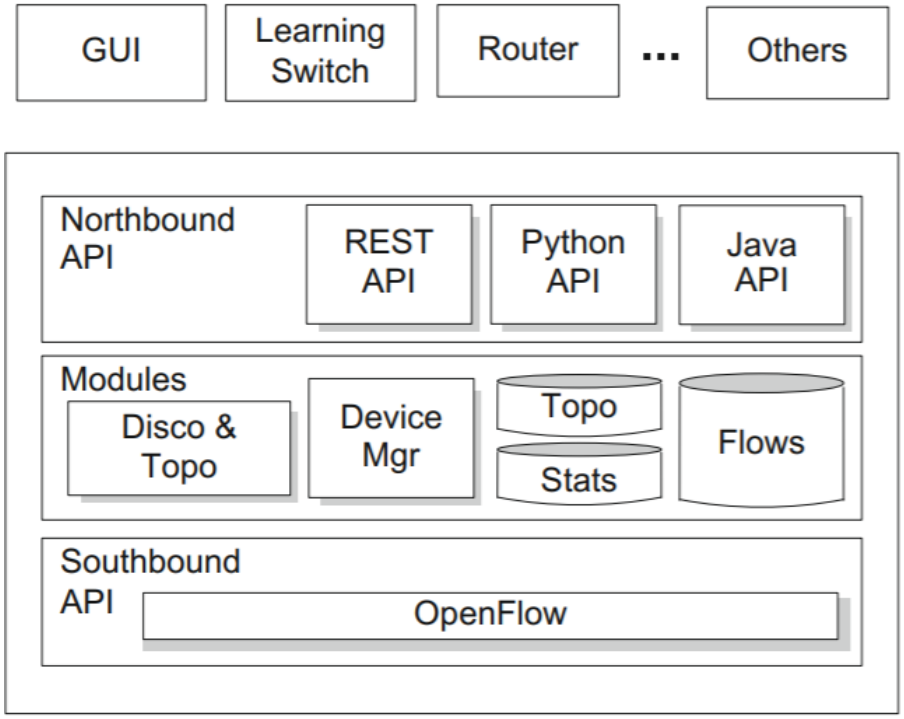
  that device to respond locally the next time it sees a packet belonging to that flow: reactive flows
- Software applications: forwarding, routing, overlay, multipath, and access control functions, among others
- Controller can insert flows reactively in response to other data sources such as intrusion detection systems (IDS) or traffic analyzer

SDN Operation: OpenFlow

- Means of communication between the controller and the device



SDN switch



SDN controller

SDN Operation: SDN controller

- Device and topology discovery and tracking
  - Learning of the existence of switches (SDN devices) and end-user devices and tracking the connectivity between them
- Flow management, device management, and statistics tracking
  - Maintains a flow cache that mirrors the flow tables on the various switches it controls
  - Locally maintains per-flow statistics that it has gathered from its switches

SDN Operation: Scaling the number of flows

- At the edge, flows will permit different policies to be applied to individual users and even different traffic types of the same user
  - Multiple flow entries for a single user

- Core devices, the flow definitions will be generally more coarse, with a single aggregated flow entry matching the traffic from a large number of users whose traffic is aggregated in some way, such as a tunnel, a VLAN, or a MPLS LSP
  - Policies applied deep into the network to aggregated flows

SDN switches

- Software:
  - Open vSwitch (OVS)
  - Indigo
- Network equipment manufacturers
  - Cisco
  - HP
  - NEC
  - IBM
  - Juniper
  - Extreme
- Have added OpenFlow support to some of their legacy switches

SDN controllers

- Software:
  - Ryu
  - Floodlight controller (Java and a RESTful API)
  - OpenDaylight controller (RESTful API)
  - NOX
  - Beacon
  - Trema
- Network equipment manufacturers
  - Cisco
  - NEC
  - IBM
  - HP

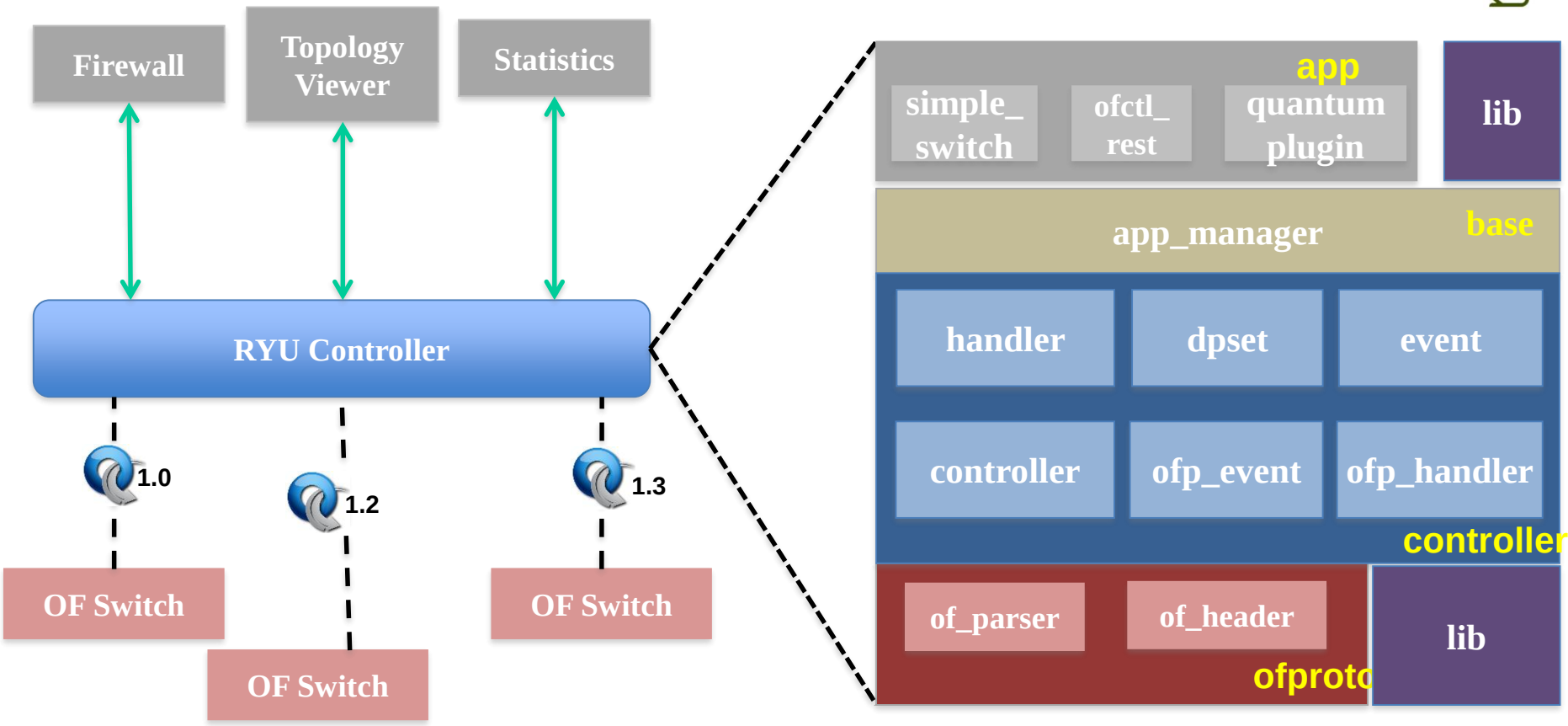
Open-source controllers

Controller	Notes
Ryu	•Apache license •Python
NOX/POX	•GPL •C++ and Python
Stanford's Beacon	•BSD-like license •Java-based
Maestro (from Rice Univ)	•GPL •Based on Java
NEC's Trema	•Open-source •Written in C and Ruby •Included test harness
Big Switch's Floodlight	•Apache license •Java-based

Sample Commercial Switches

Model	Virtualize	Notes	
HP Procurve 5400zl or 6600	1 OF instance per VLAN	-LACP, VLAN and STP processing before OpenFlow -Wildcard rules or non-IP pkts processed in s/w -Header rewriting in s/w -CPU protects mgmt during loop	
NEC IP8800	1 OF instance per VLAN	-OpenFlow takes precedence -Most actions processed in hardware -MAC header rewriting in h/w	
Brocade MLX routers	Multiple OF instance per switch	-Hybrid OpenFlow switch with legacy protocols and OpenFlow coexisting -OpenFlow commands can override state created by legacy protocos	
Pronto 3290 or 3780 with Pica8 or Indigo firmware	1 OF instance per switch	-No legacy protocols (like VLAN, STP) -Most actions processed in hardware -MAC header rewriting in h/w	

Intro to RYU: OpenFlow Control



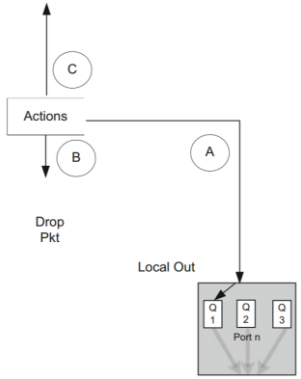
Components:

- Provides interface for control and state and generates events
- Communicates using message passing

Libraries:

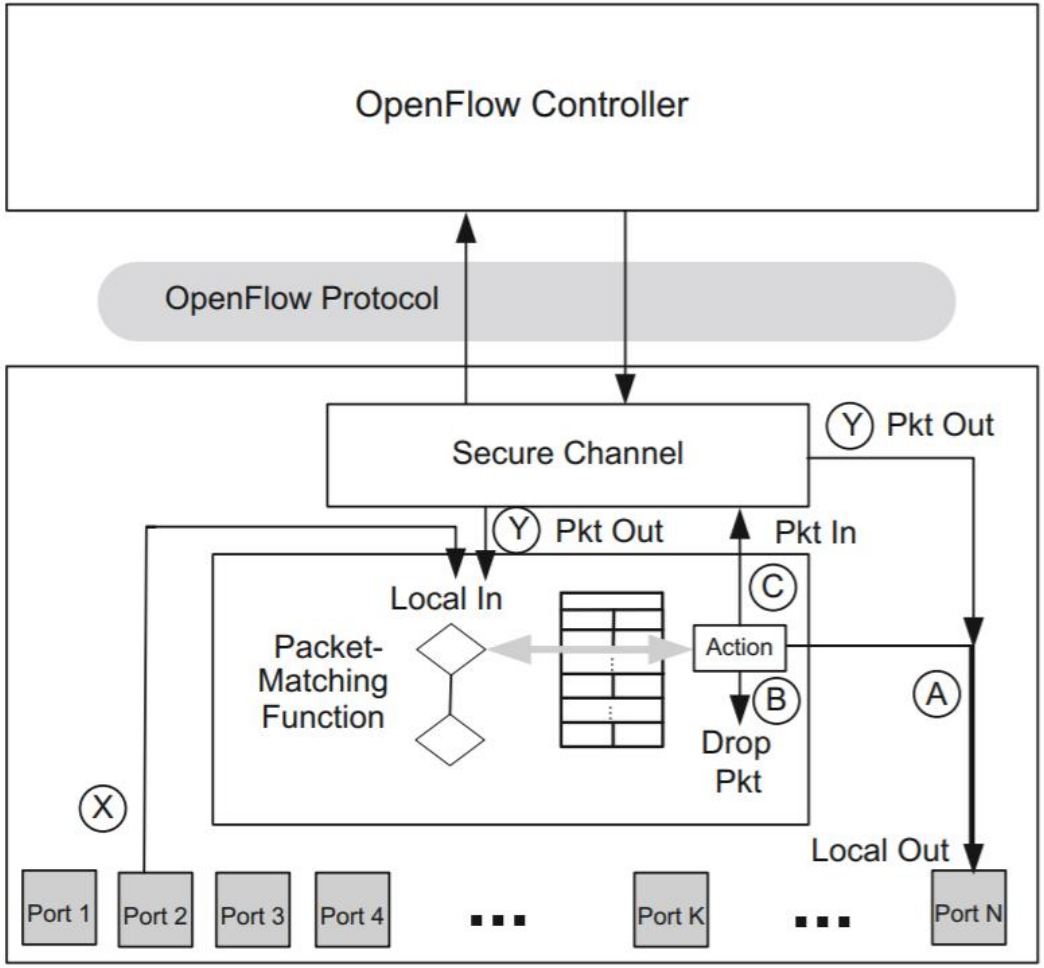
- Functions called by components
- Ex: OF-Config, Netflow, sFlow, Netconf, OVSDB

OpenFlow: base



- Interface between switches and network controller

- A. Forward the packet out a local port, possibly modifying certain header fields first.
- B. Drop the packet
- C. Pass the packet to the controller



OpenFlow: Flow Tables

- Header fields: match criteria to determine whether an incoming packet matches this entry
- Counters: track statistics relative to this flow, such as how many packets have been forwarded or dropped for this flow
- Actions fields prescribe what the switch should do with a packet matching this entry

Flow Entry 0		**Flow Entry 1**		...	**Flow Entry F**		...	**Flow Entry M**	
Header Fields	Inport 12 192.32.10.1, Port 1012	Header Fields	Inport * 209.*.*.*, Port *		Header Fields	Inport 2 192.32.20.1, Port 995		Header Fields	Inport 2 192.32.30.1, Port 995
Counters	val	Counters	val		Counters	val		Counters	val
Actions	val	Actions	val		Actions	val		Actions	val

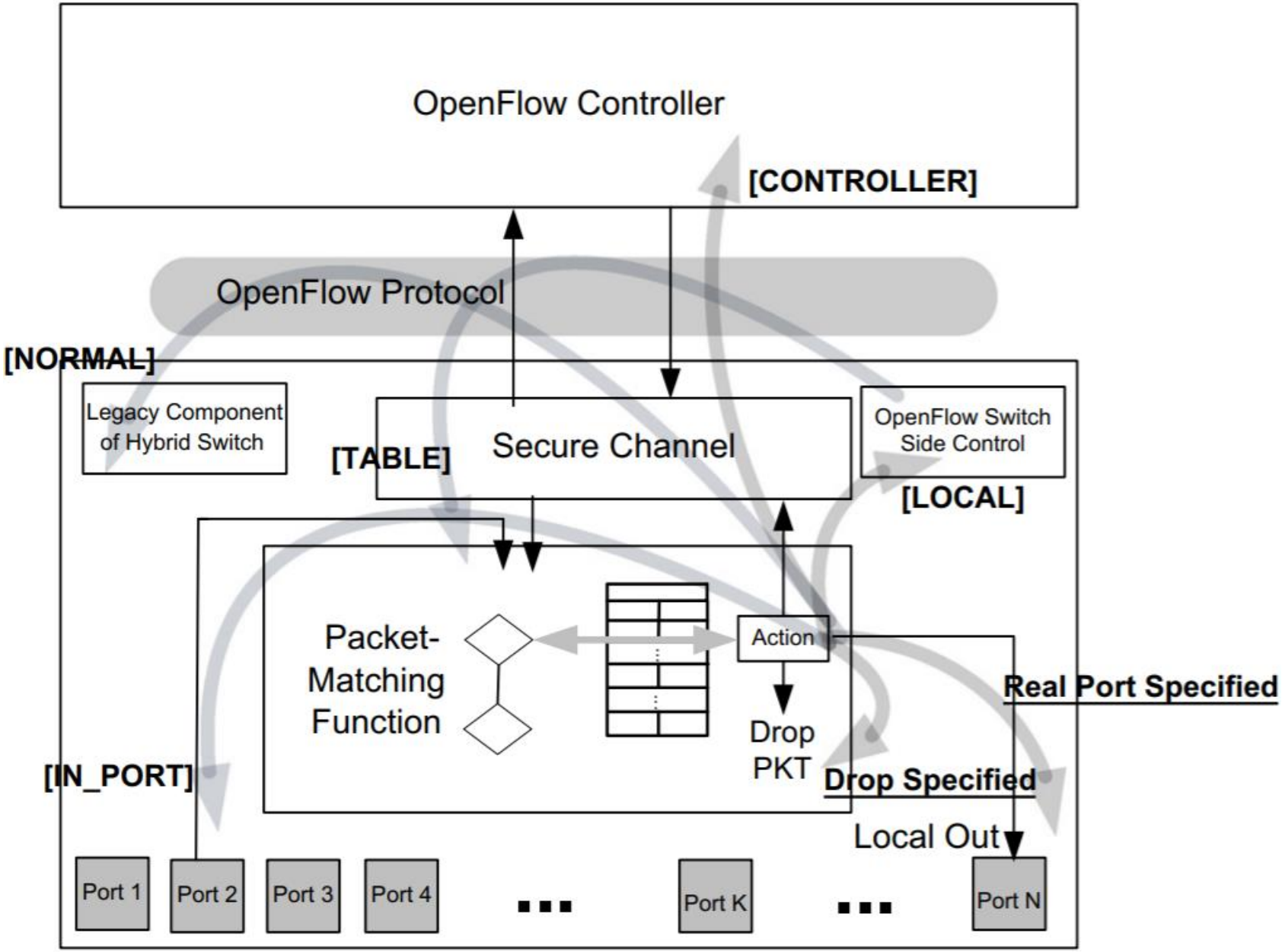
OpenFlow: Packet Matching criteria

- Flow entries are processed in order: once a match is found, no further match attempts are made against that flow table
  - Switch input port
  - VLAN ID
  - VLAN priority
  - Ethernet source/destination address
  - Ethernet frame type
  - IP source/destination address
  - IP protocol
  - IP Type of Service (ToS) bits
  - TCP/UDP source/destination port
  - …

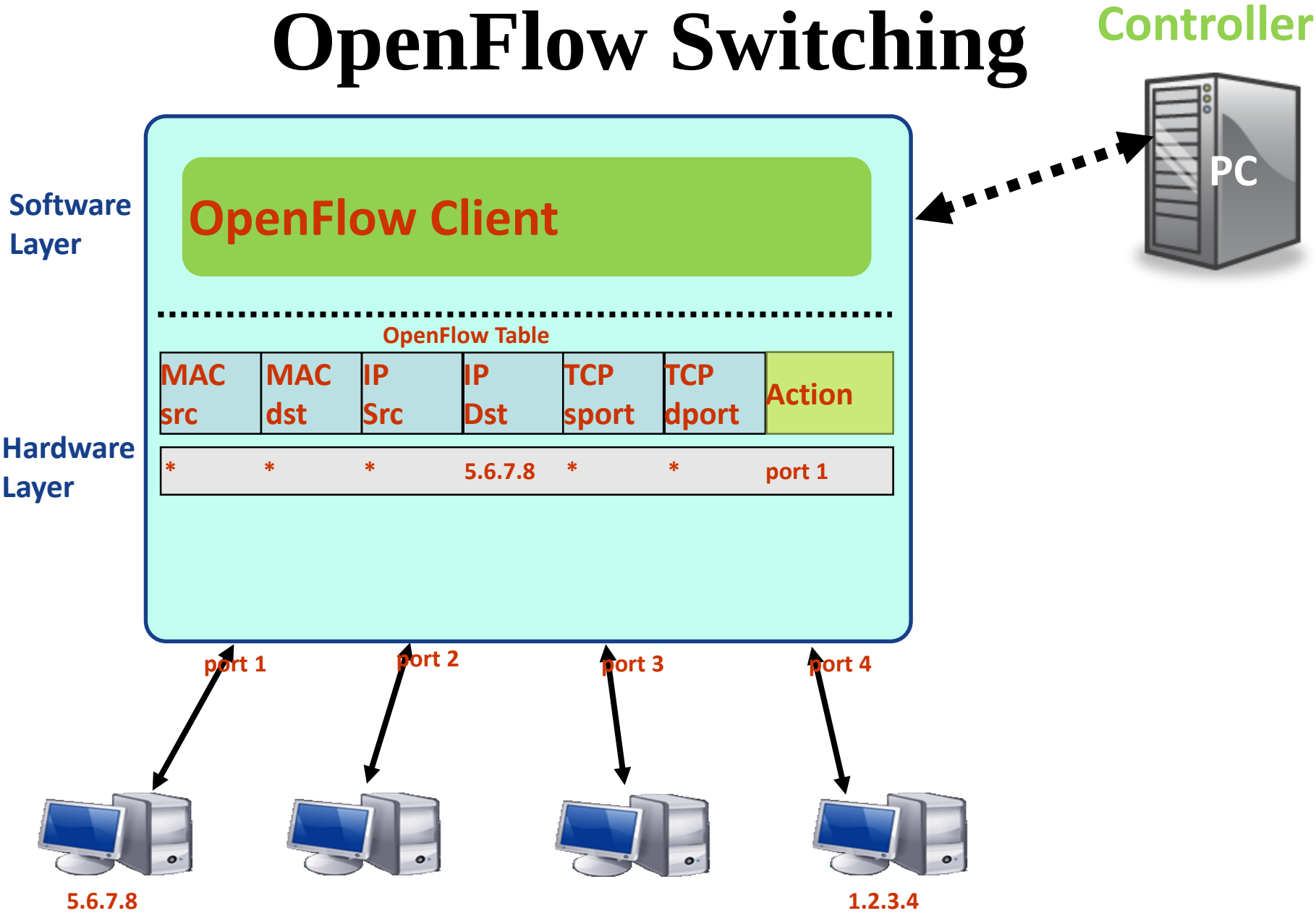
OpenFlow: Packet Forwarding Actions

- Local: packet needs to be processed by the local OpenFlow control software
- All/Flood: flood a packet out all ports on the switch except the input port
- Controller: forward to the OpenFlow controller
- In_Port: forward the packet back out of the port on which it arrived (2 virtual machines on the same server to communicate)
- Table: packets that the controller sends to the switch, which arrive as part of the PACKET_OUT message from the controller, which includes an action list
- Normal: forward to the legacy forwarding logic of the switch

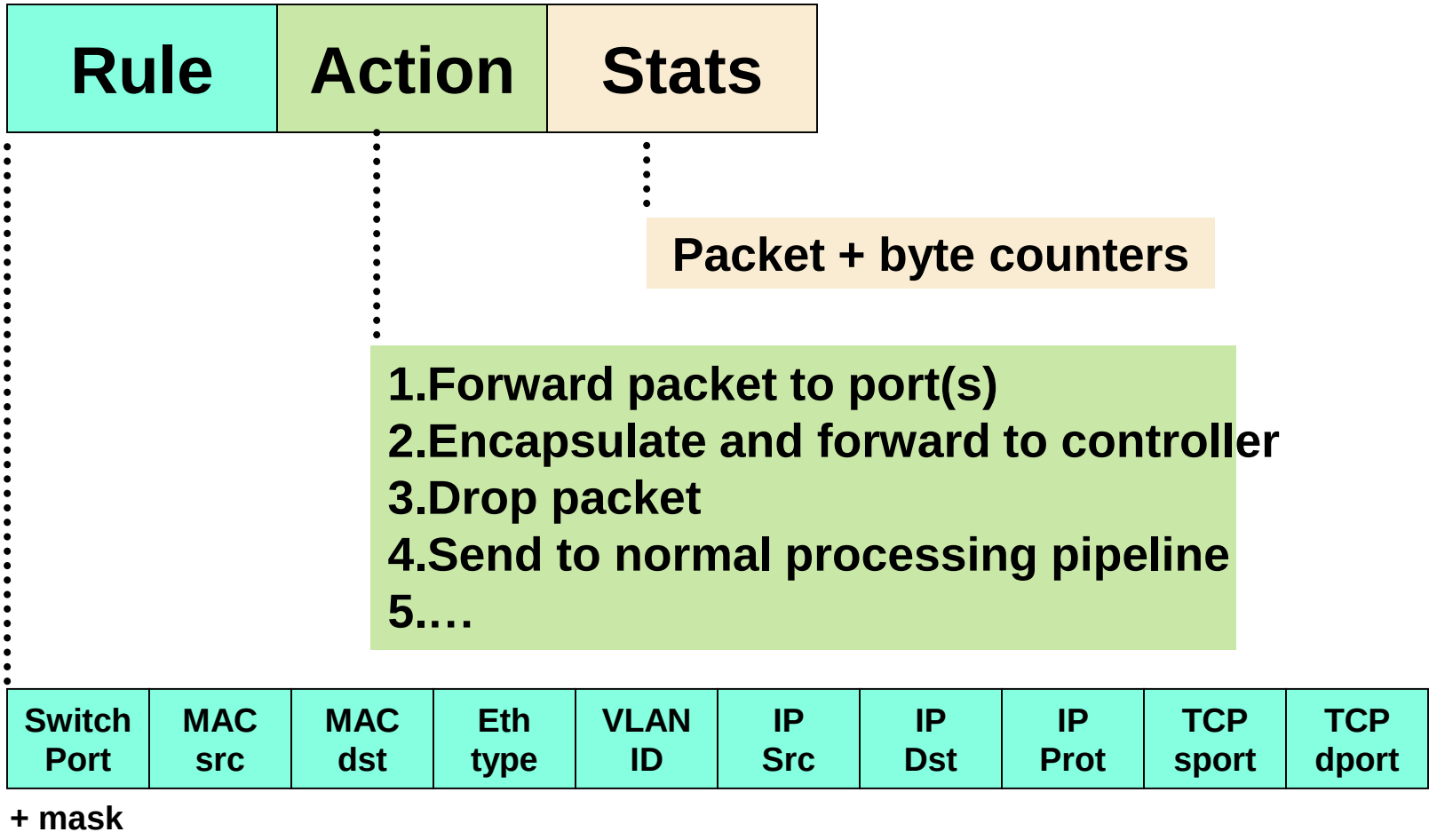
OpenFlow: Packet Forwarding



OpenFlow Switching



OpenFlow Table Entry



OpenFlow Examples

Switching

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

Routing

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

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

OpenFlow Examples

Flow Switching

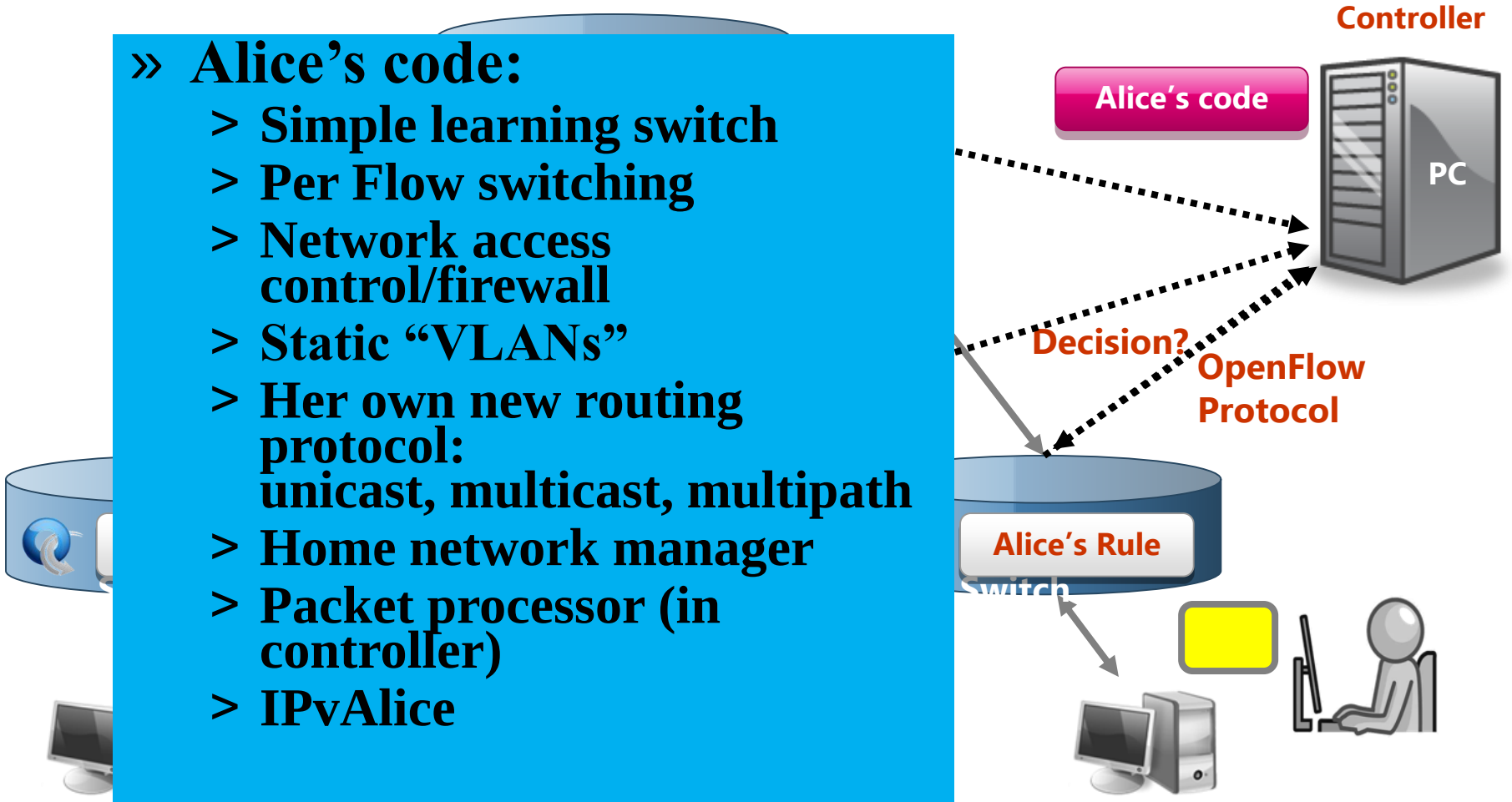
Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
port3	00:20..	00:1f..	0800	vlan1	1.2.3.4	5.6.7.8	4	17264	80	port6

VLAN Switching

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 Usage

- Alice's code:
  - Simple learning switch
  - Per Flow switching
  - Network access control/firewall
  - Static "VLANs"
  - Her own new routing protocol: unicast, multicast, multipath
  - Home network manager
  - Packet processor (in controller)
  - IPvAlice

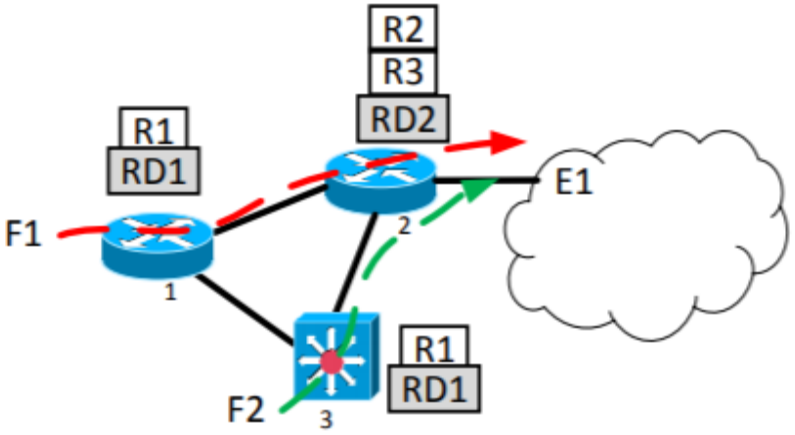


OpenFlow Rules in Different Switches

- Access control rules placement
- Firewall policy is transformed into:
  - list of rules R1, R2, R3 used to block matching packets
  - two default rules RD1, RD2 for forwarding non-matching packets towards endpoint E
  - Rules distributed on several switches to ensure that flows F1 and F2 pass through all rules R1, R2, R3 to satisfy the policy.

Block traffic to **10.0.0.1** or to port **22** or from **10.0.0.2**

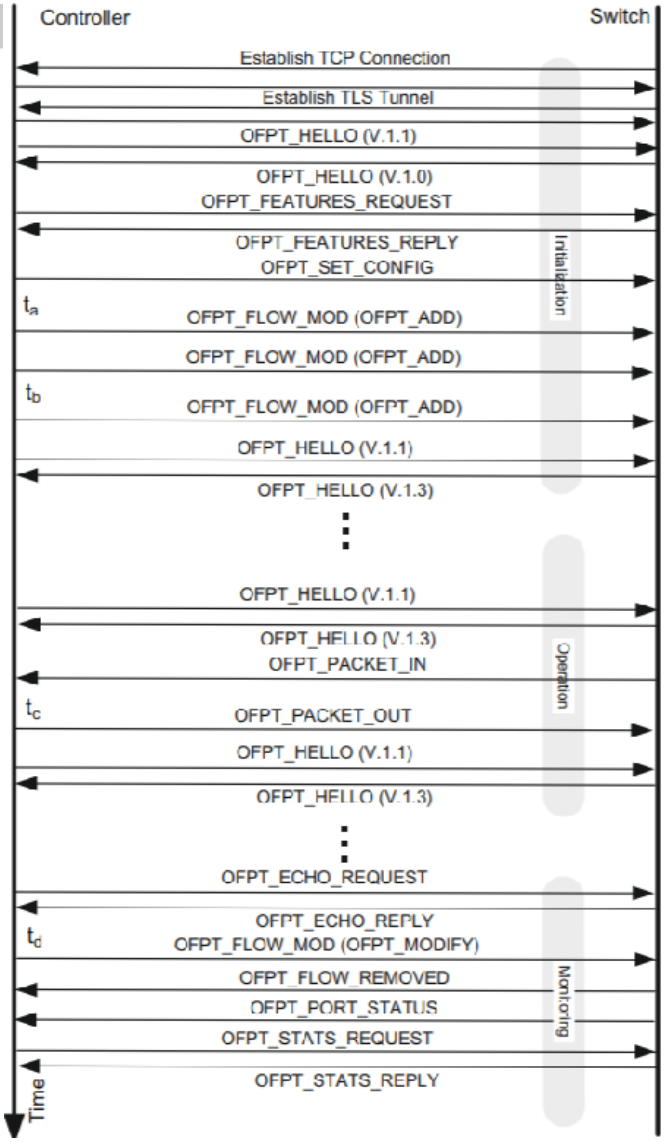
Rule	Match	Action
R1	dst_ip=10.0.0.1	Drop
R2	dst_port=22	Drop
R3	src_ip=10.0.0.2	Drop
RD1	*	To Switch 2
RD2	*	E1



Switch Memory Capacity = 3

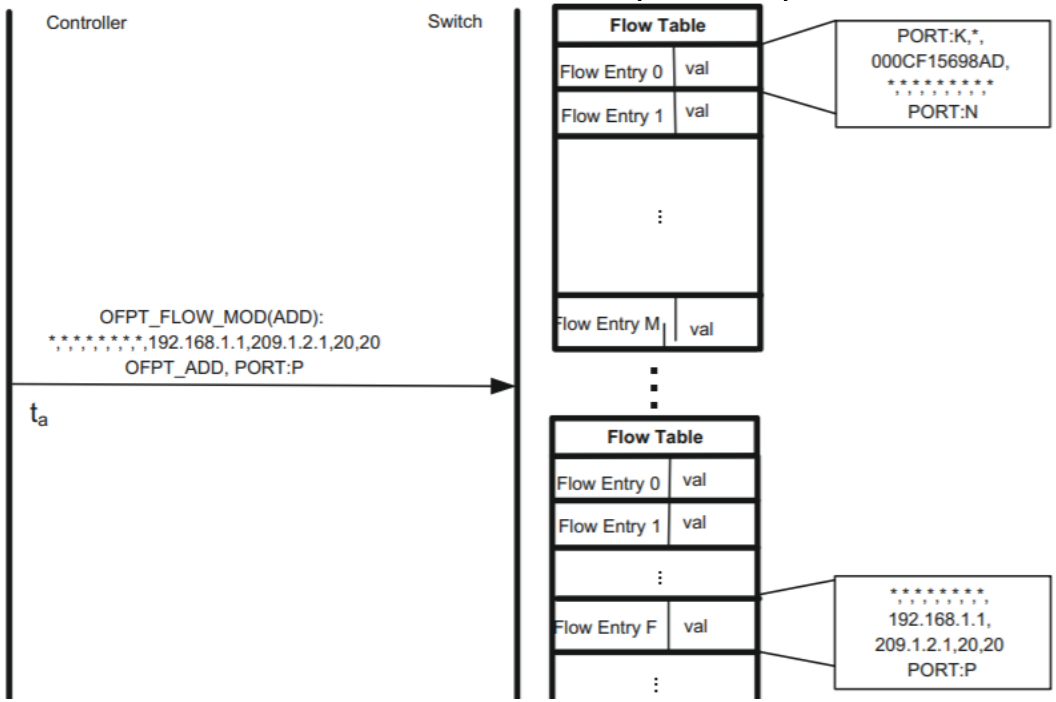
OpenFlow: Messages (examples)

Message Type	Category	Subcategory
HELLO	Symmetric	Immutable
ECHO_REQUEST	Symmetric	Immutable
ECHO_REPLY	Symmetric	Immutable
VENDOR	Symmetric	Immutable
FEATURES_REQUEST	Controller-switch	Switch configuration
FEATURES_REPLY	Controller-switch	Switch configuration
GET_CONFIG_REQUEST	Controller-switch	Switch configuration
GET_CONFIG_REPLY	Controller-switch	Switch configuration
SET_CONFIG	Controller-switch	Switch configuration
PACKET_IN	Async	NA
FLOW_REMOVED	Async	NA
PORT_STATUS	Async	NA
ERROR	Async	NA
PACKET_OUT	Controller-switch	Cmd from controller
FLOW_MOD	Controller-switch	Cmd from controller
PORT_MOD	Controller-switch	Cmd from controller
STATS_REQUEST	Controller-switch	Statistics
STATS_REPLY	Controller-switch	Statistics
BARRIER_REQUEST	Controller-switch	Barrier
BARRIER_REPLY	Controller-switch	Barrier
QUEUE_GET_CONFIG_REQUEST	Controller-switch	Queue configuration
QUEUE_GET_CONFIG_REPLY	Controller-switch	Queue configuration



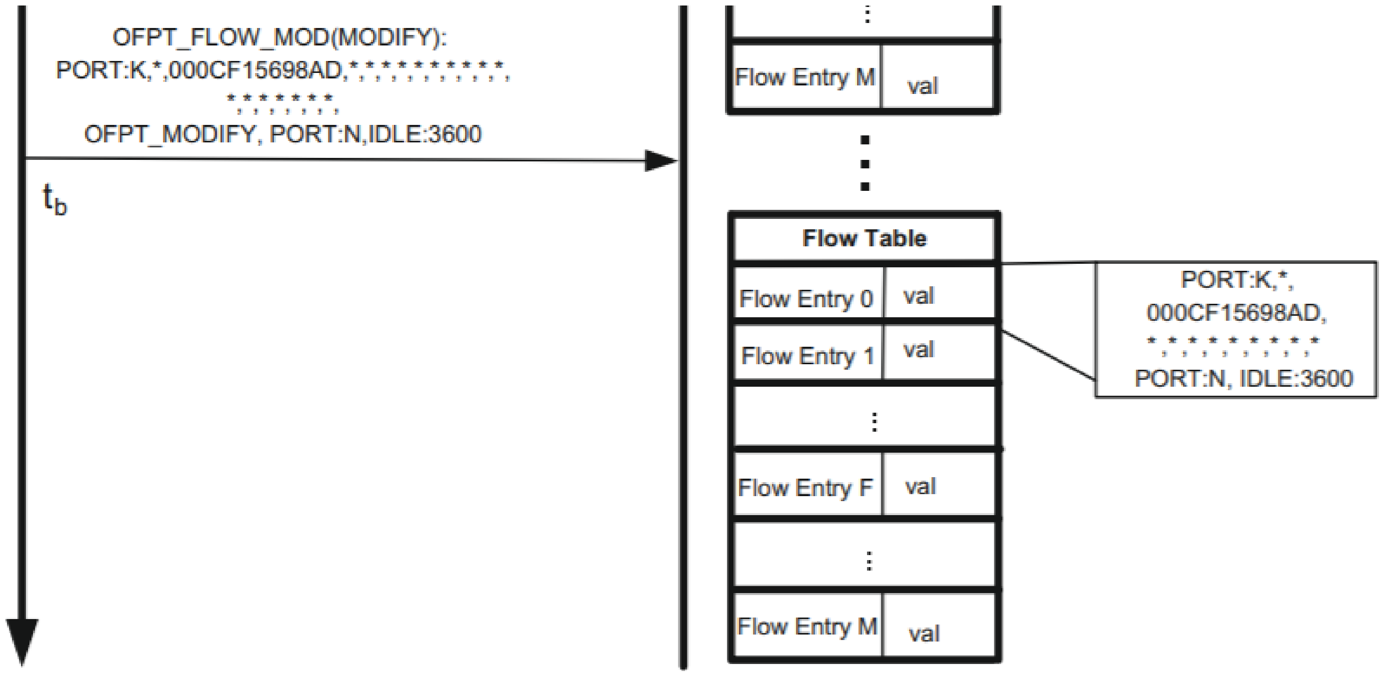
OpenFlow: Controller Programming Flow Table

- Entry 0: all Ethernet frames entering the switch on input port K with a destination Ethernet address of 0x000CF15698AD should be output on output port N
- At time ta, the controller sends a FLOW_MOD (ADD) command to the switch, adding a flow for packets with source IP addresses 192.168.1.1 and destination 209.1.2.1, source TCP port 20, and destination port 20. All other match fields have been wildcarded
- The outport port is specified as P



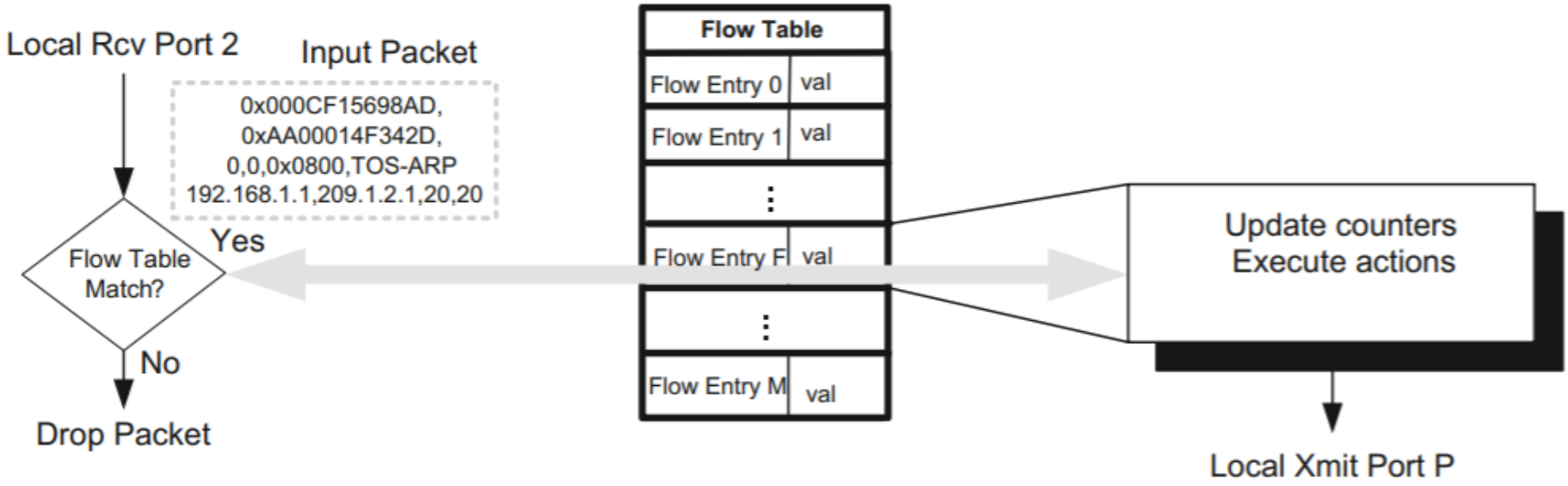
OpenFlow: Controller Programming Flow Table

- FLOW_MOD (MODIFY) command for flow entry zero
- One-hour (3600-second) idle time on that flow
  - After that number of seconds of inactivity on that flow, the flow should be deleted

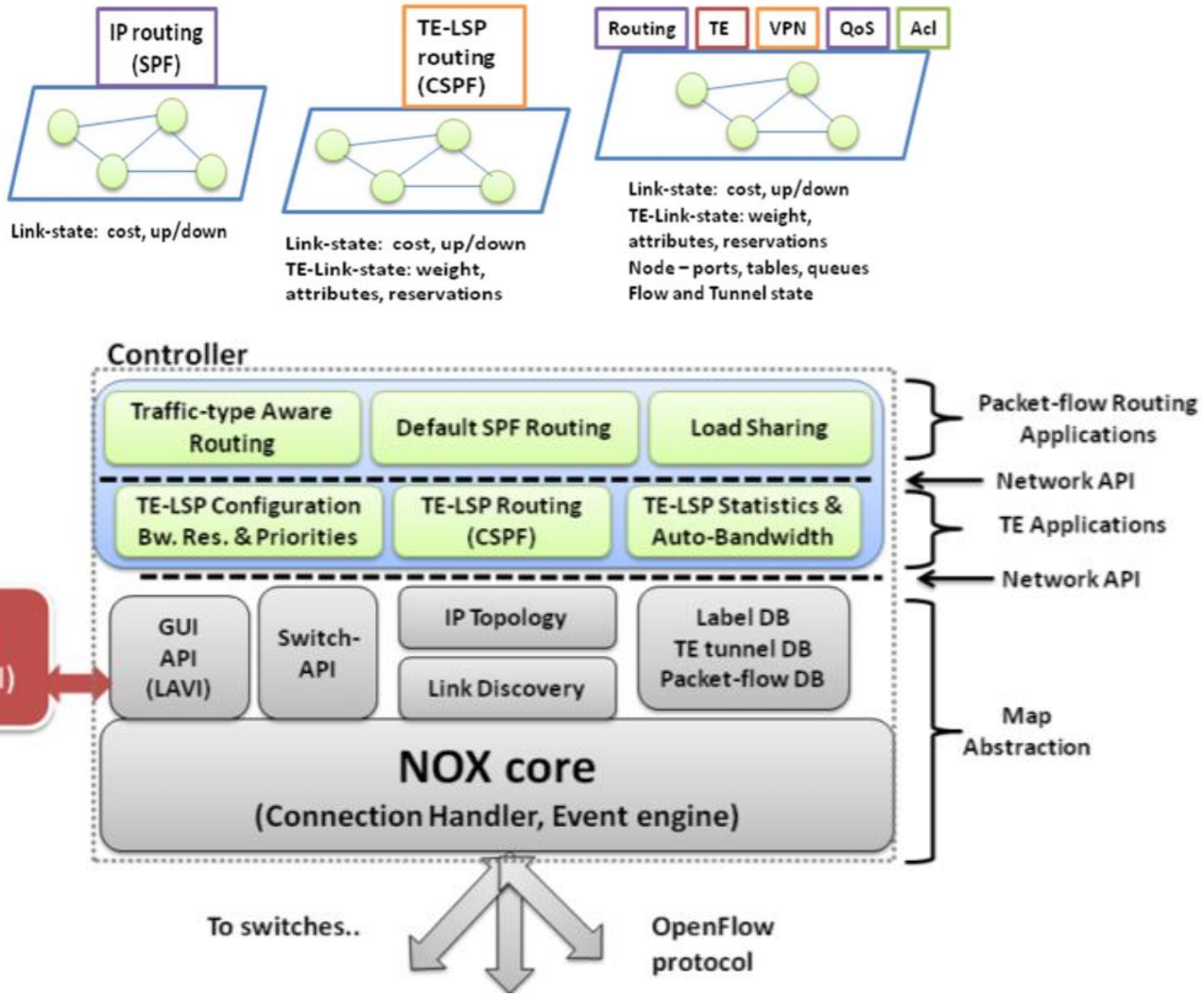


OpenFlow: Basic Packet Forwarding

- Packet arriving through port 2 with source IPv4 address of 192.168.1.1 and destination IPv4 address of 209.1.2.1
- Packet-matching function scans the flow table
- Finds a match in flow entry F → matching packet should be forwarded out port P



OpenFlow: MPLS

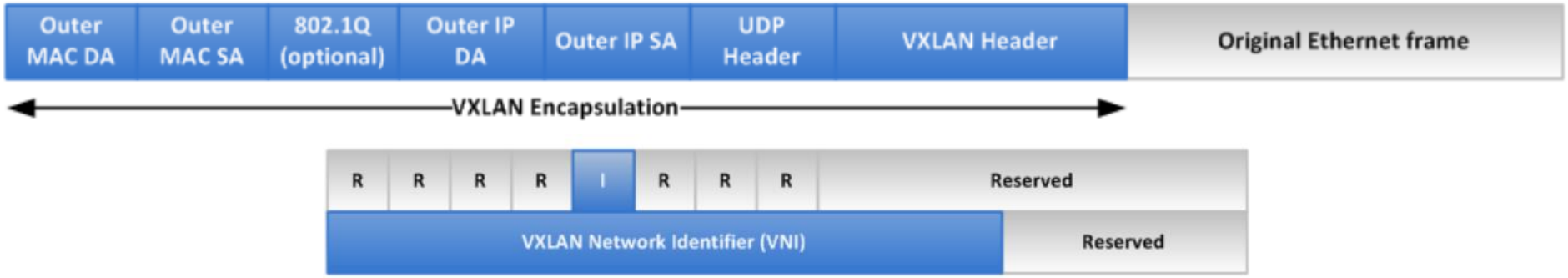


Virtual Traffic and Tunnels

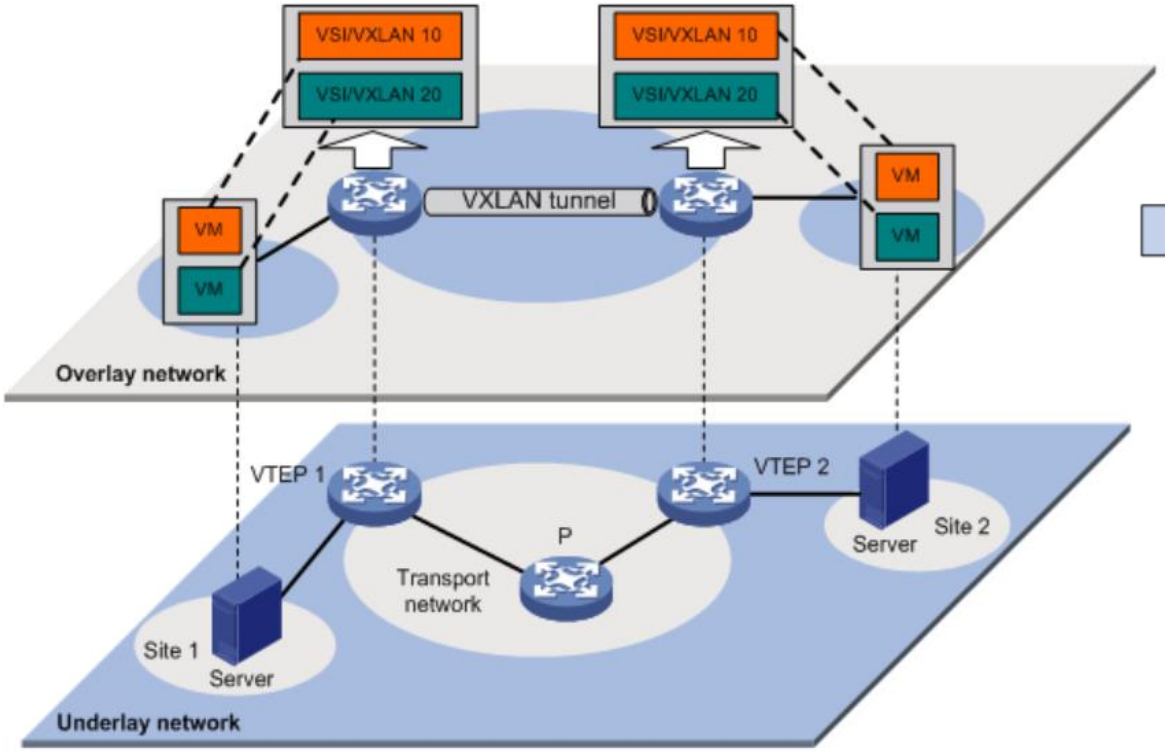
- Virtual network traffic runs above the physical network infrastructure
  - Endpoints are unaware of the details of the physical topology, the way routing occurs, or other basic network functions
  - Virtual networks can be controlled entirely by the devices at the edge of the network → Tunneling
  - When a packet enters the edge of the virtual network at the source, the networking device will encapsulate it within another frame
    - Edge of the virtual network: virtual tunnel endpoint (VTEP)
  - It is then sent, through information programmed by the controller, to the destination's VTEP
  - This VTEP decapsulates the packet and forwards it to the destination host
  - MAC-in-IP tunneling: VXLAN (Virtual eXtensible LAN), NVGRE, STT
  - Multiple overlay networks can exist independently and simultaneously over the same physical network

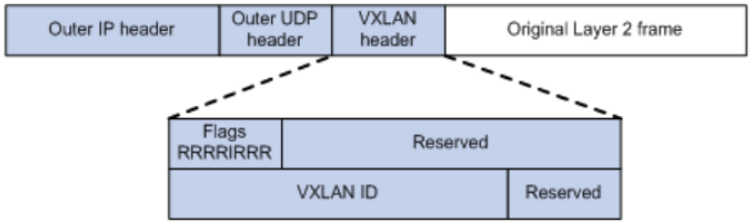
VXLAN: Virtual eXtensible LAN

- Same services as VLANs, in a virtualized environment
- Also provide a means to stretch L2 network over a L3 network
- VXLAN ID (called VXLAN Network Identifier or VNI) is 24-bits long compared to 12-bits of VLAN ID (over 16 million unique IDs)
- VTEPs connect an access switch (currently virtual switch) to the IP network
- The function of VTEP is to encapsulate the virtual traffic within an IP header to send across an IP network

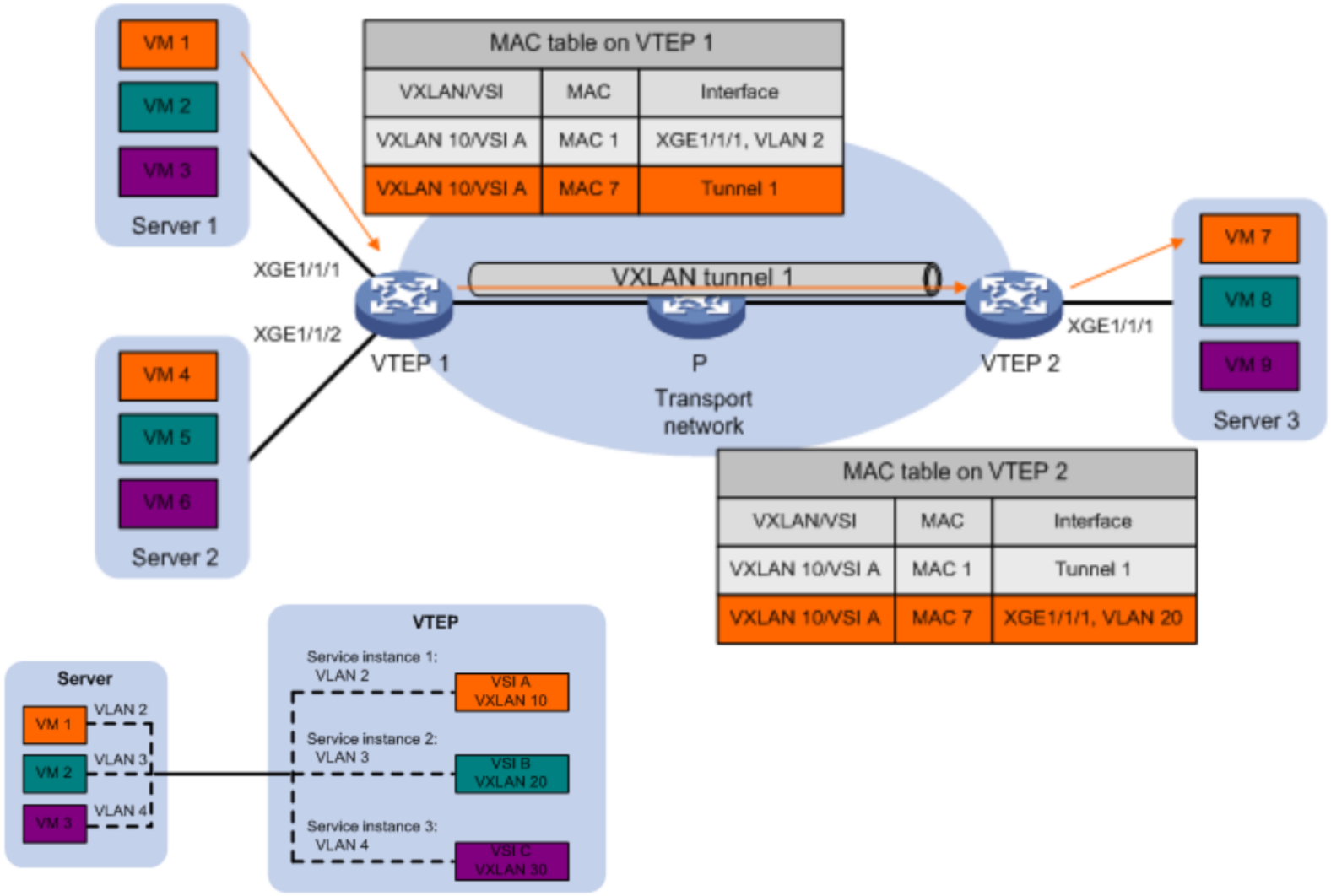


VXLAN Example

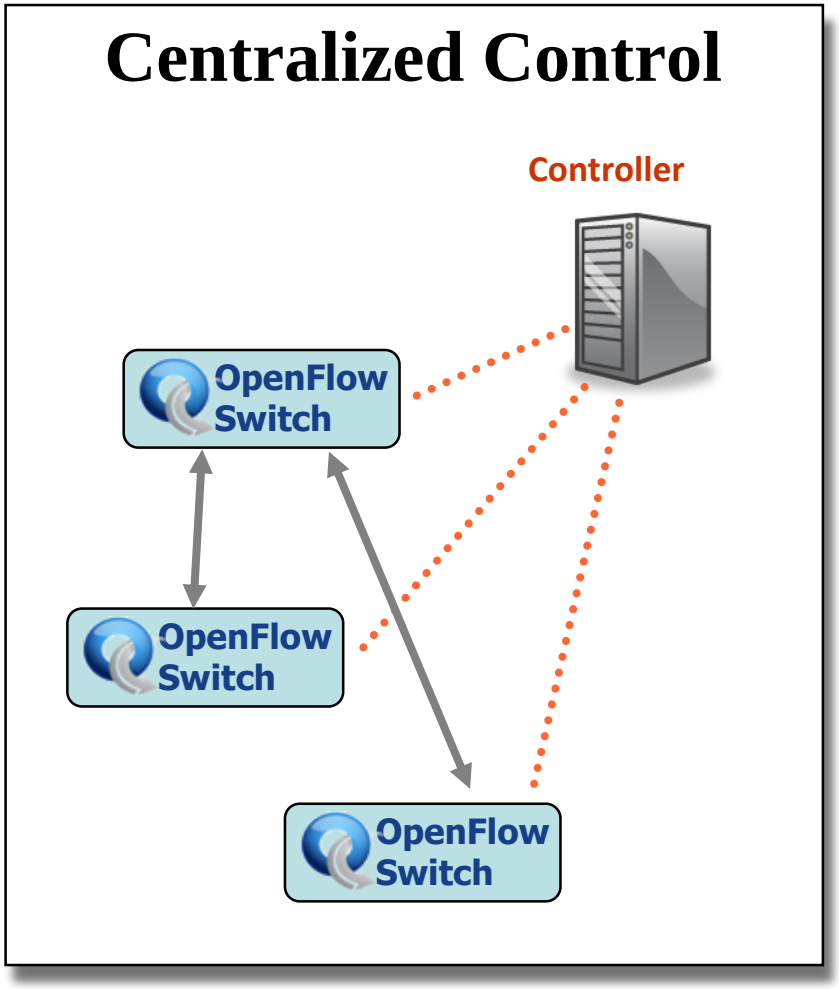


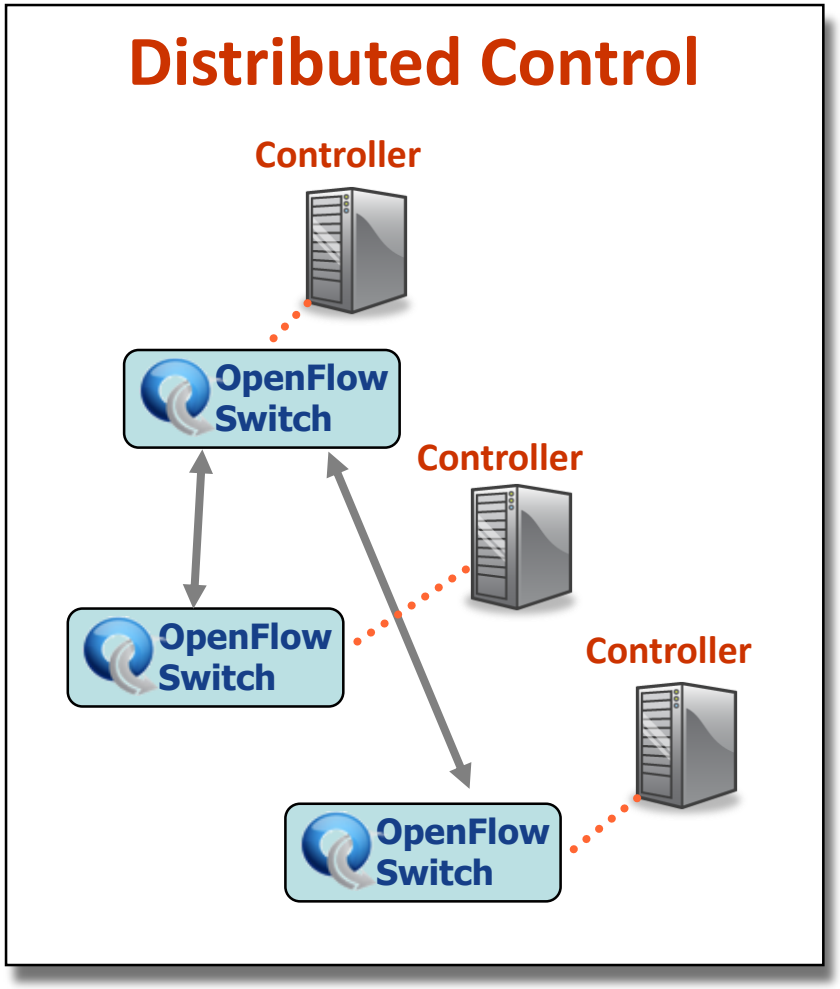


VXLAN Example



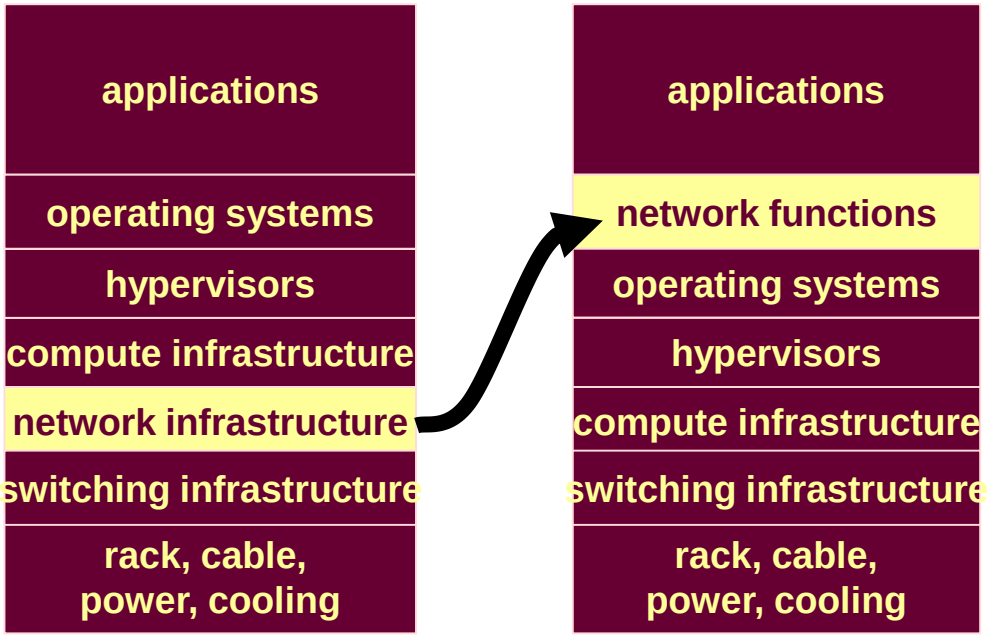
Centralized vs Distributed Control



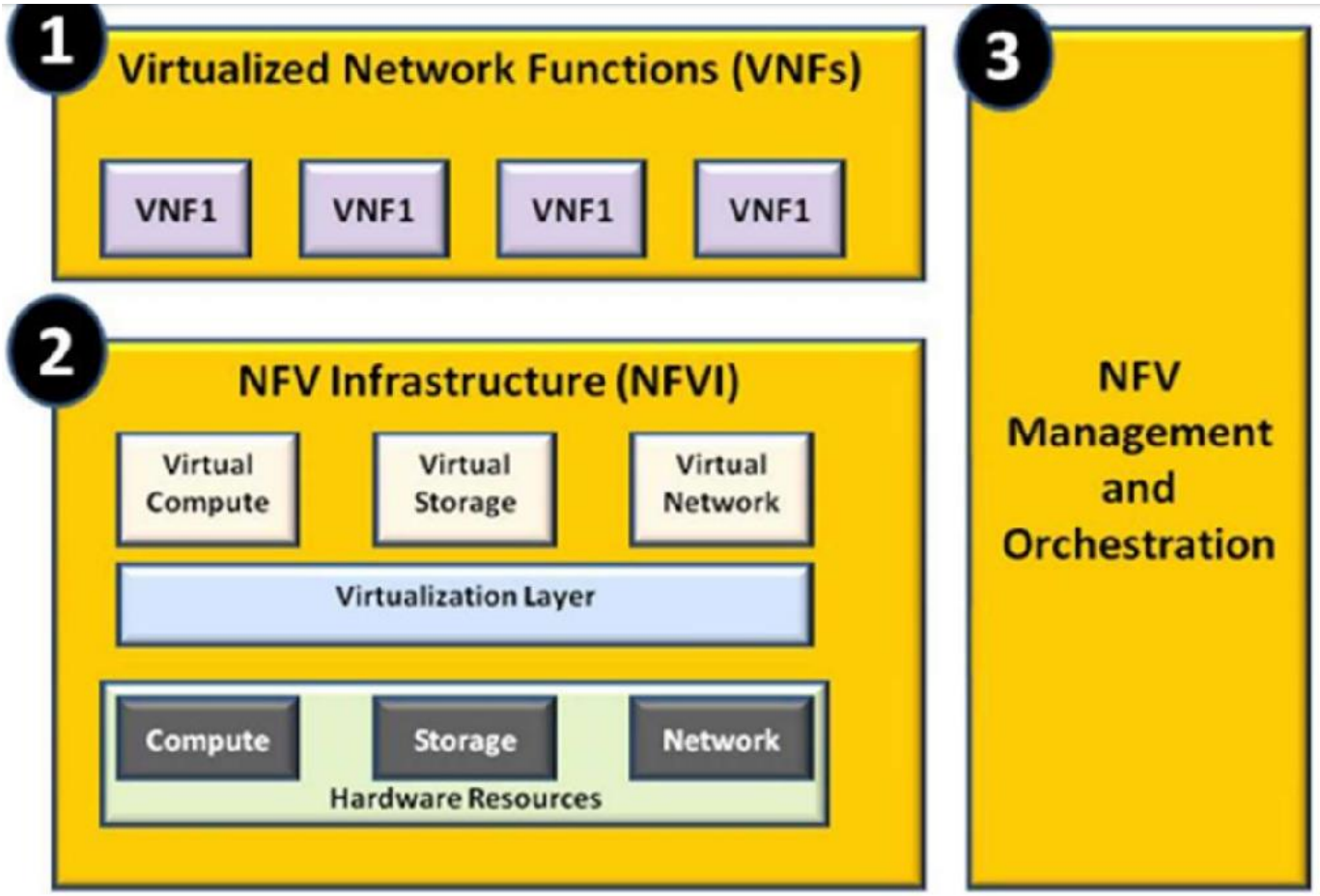


Network Functions Virtualization

- **Management & orchestration**
  - **infrastructure management standards**
  - **multi-level identity standard**
  - **resource description language**



NFV Architecture



NFV Examples