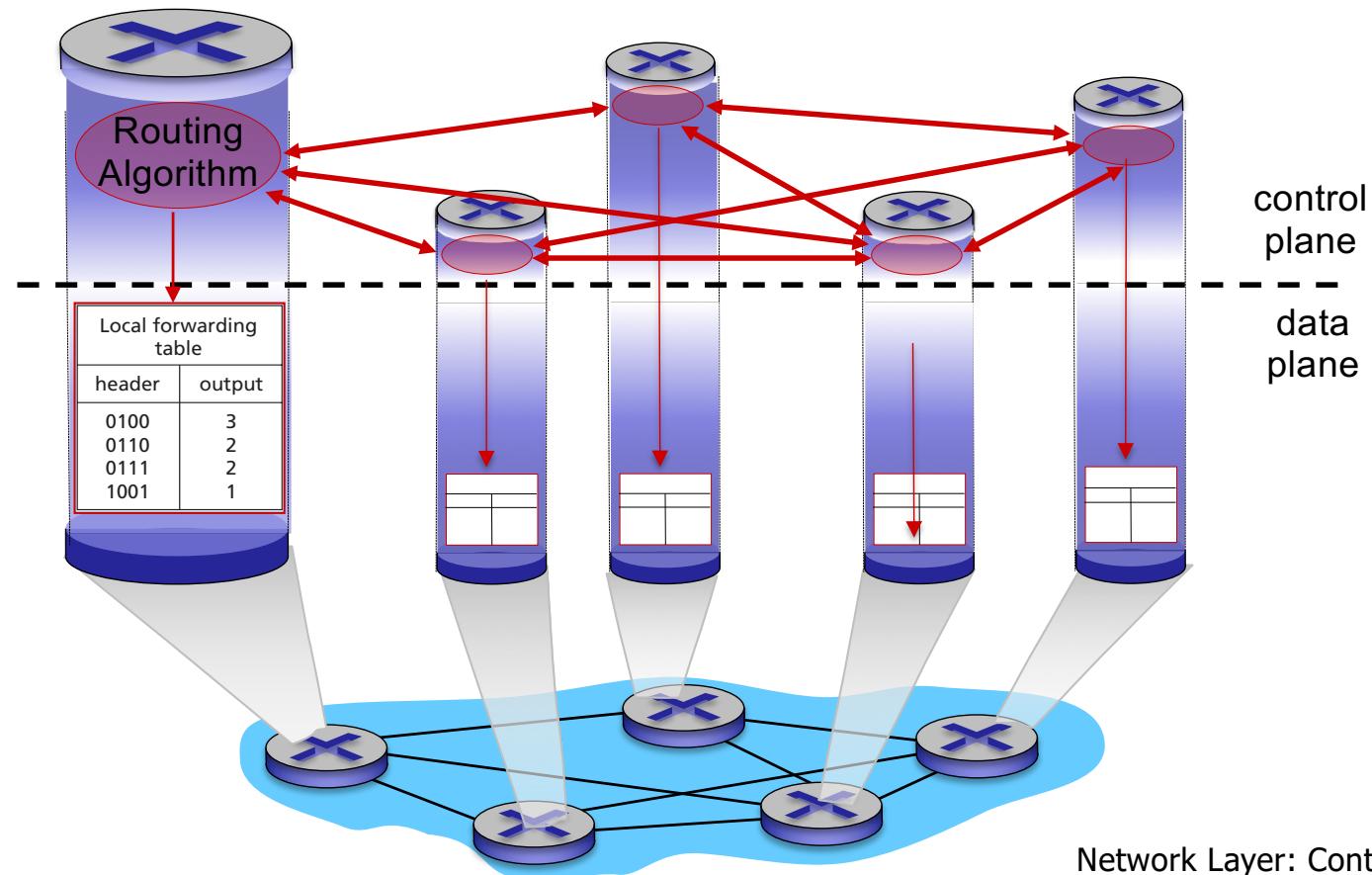


# Software defined networking (SDN)

- Internet network layer: historically has been implemented via distributed, per-router approach
  - *monolithic* router contains switching hardware, runs proprietary implementation of Internet standard protocols (IP, RIP, IS-IS, OSPF, BGP) in proprietary router OS (e.g., Cisco IOS)
  - different “middleboxes” for different network layer functions: firewalls, load balancers, NAT boxes, ..
- ~2005: renewed interest in rethinking network control plane

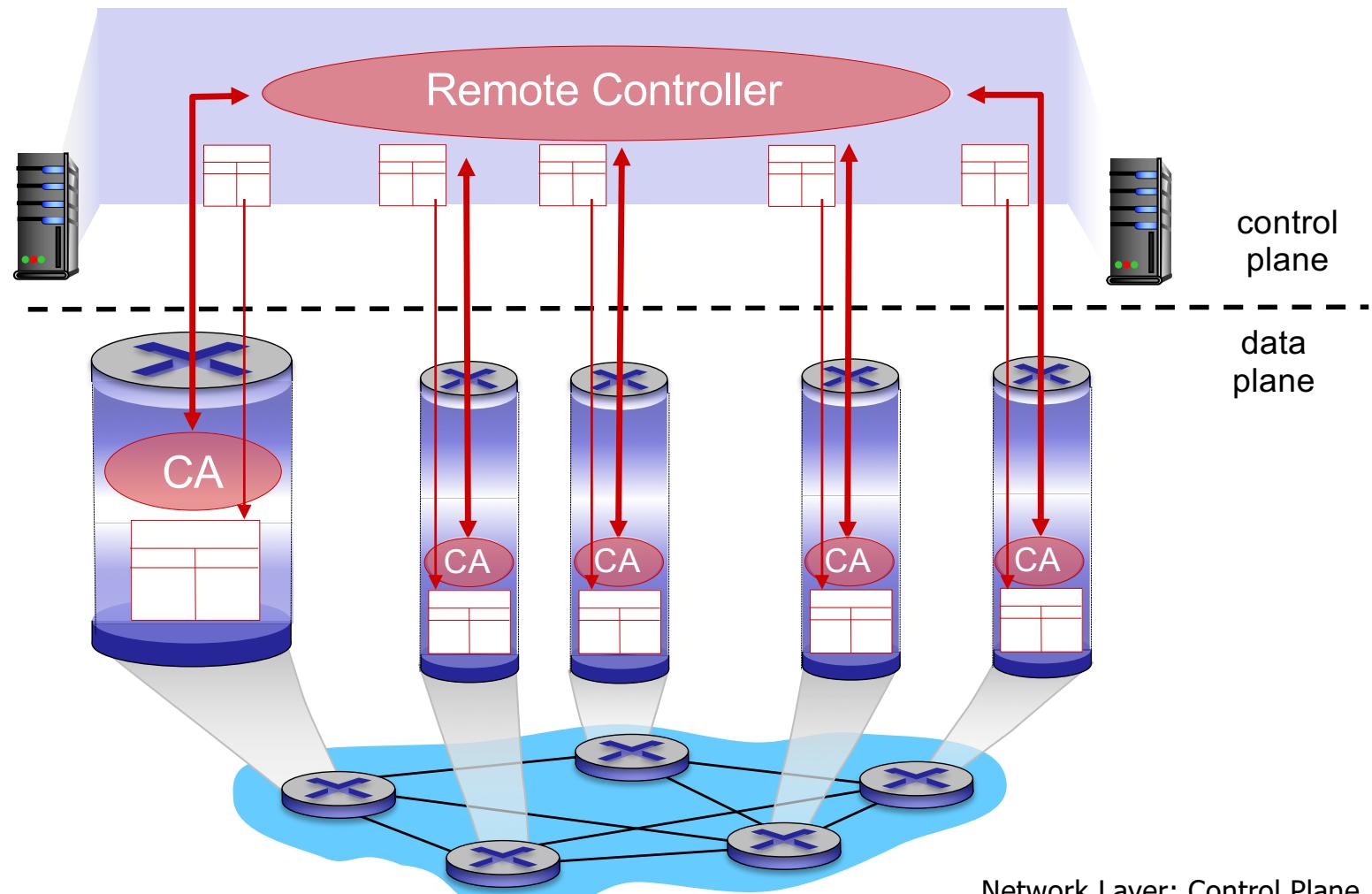
# Recall: per-router control plane

Individual routing algorithm components *in each and every router* interact with each other in control plane to compute forwarding tables



# Recall: logically centralized control plane

A distinct (typically remote) controller interacts with local control agents (CAs) in routers to compute forwarding tables



# Software defined networking (SDN)

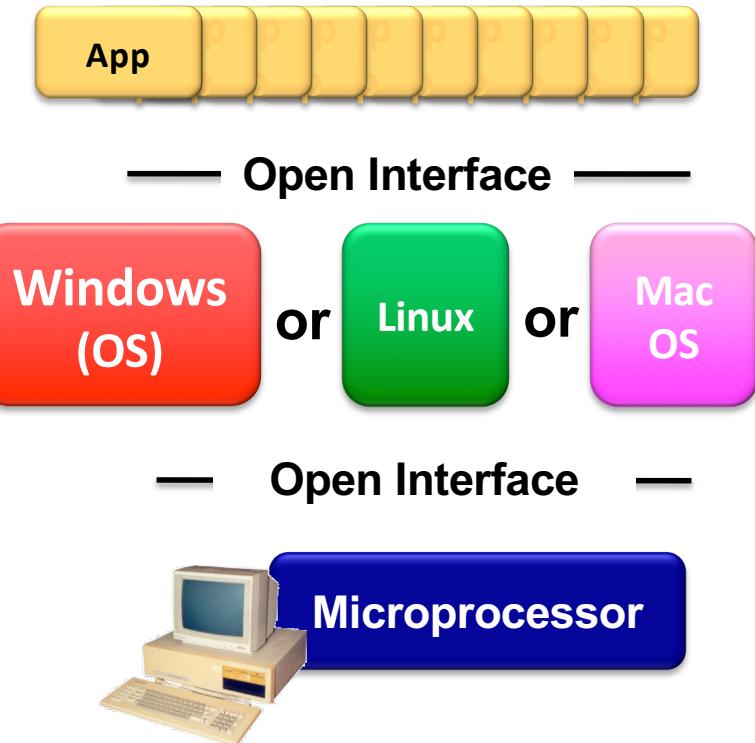
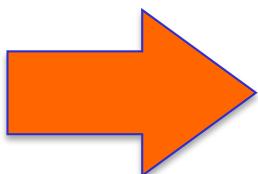
*Why* a *logically centralized* control plane?

- easier network management: avoid router misconfigurations, greater flexibility of traffic flows
- table-based forwarding (OpenFlow API discussed shortly) allows “programming” routers
  - centralized “programming” easier: compute tables centrally and distribute
  - distributed “programming” more difficult: compute tables as result of distributed algorithm (protocol) implemented in each and every router
- open (non-proprietary) implementation of control plane

# Analogy: mainframe to PC evolution\*

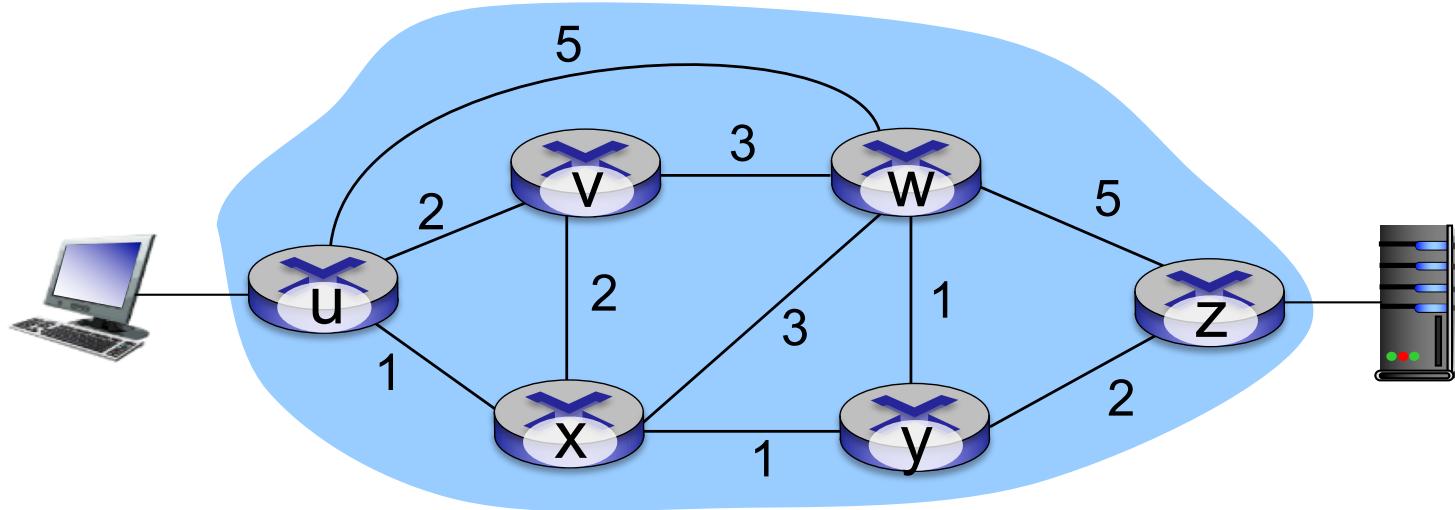


Vertically integrated  
Closed, proprietary  
Slow innovation  
Small industry



Horizontal  
Open interfaces  
Rapid innovation  
Huge industry

# Traffic engineering: difficult traditional routing

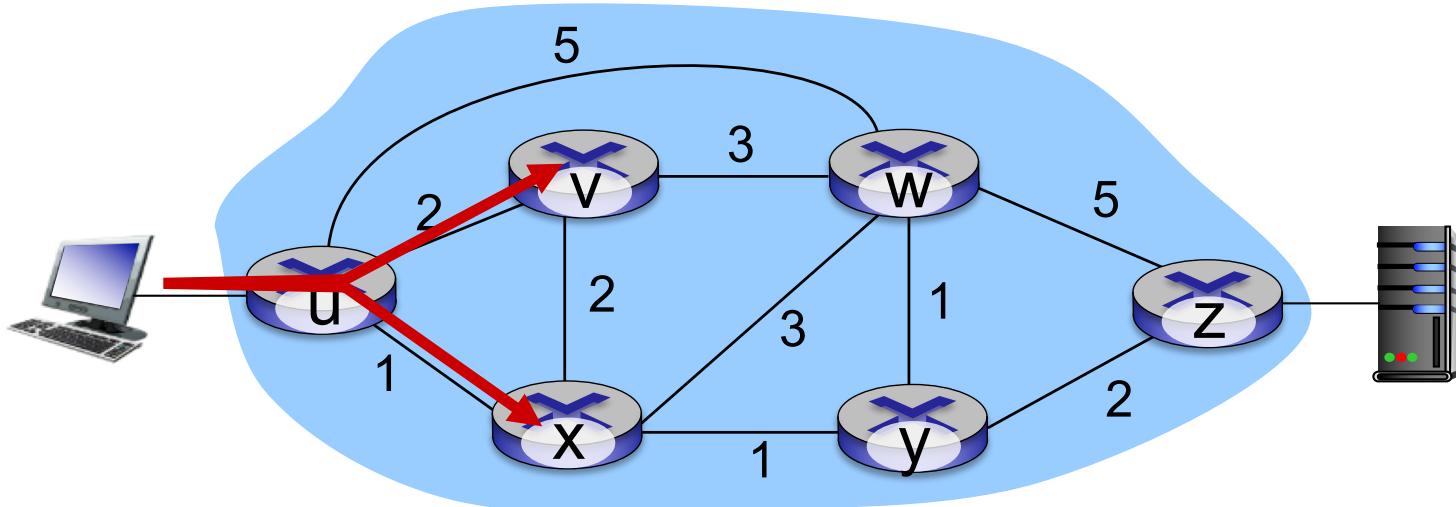


Q: what if network operator wants u-to-z traffic to flow along uvwz, x-to-z traffic to flow xwyz?

A: need to define link weights so traffic routing algorithm computes routes accordingly (or need a new routing algorithm)!

*Link weights are only control “knobs”: wrong!*

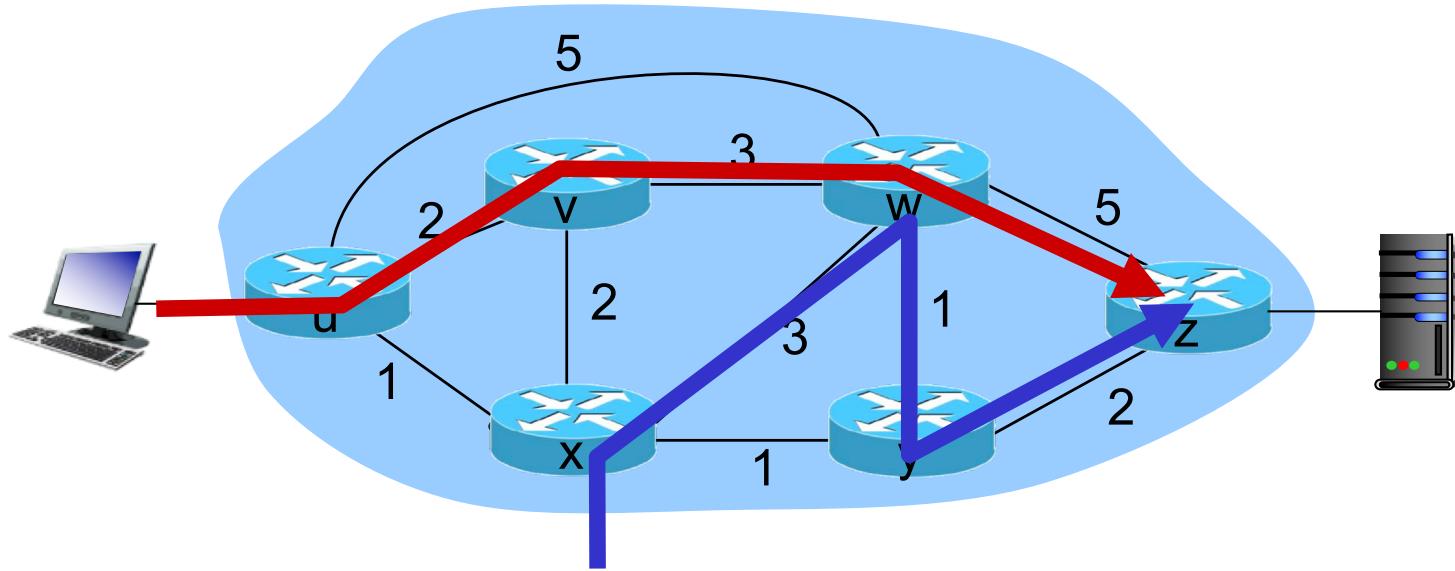
# Traffic engineering: difficult



Q: what if network operator wants to split u-to-z traffic along uvwz *and* uxyz (load balancing)?

A: can't do it (or need a new routing algorithm)

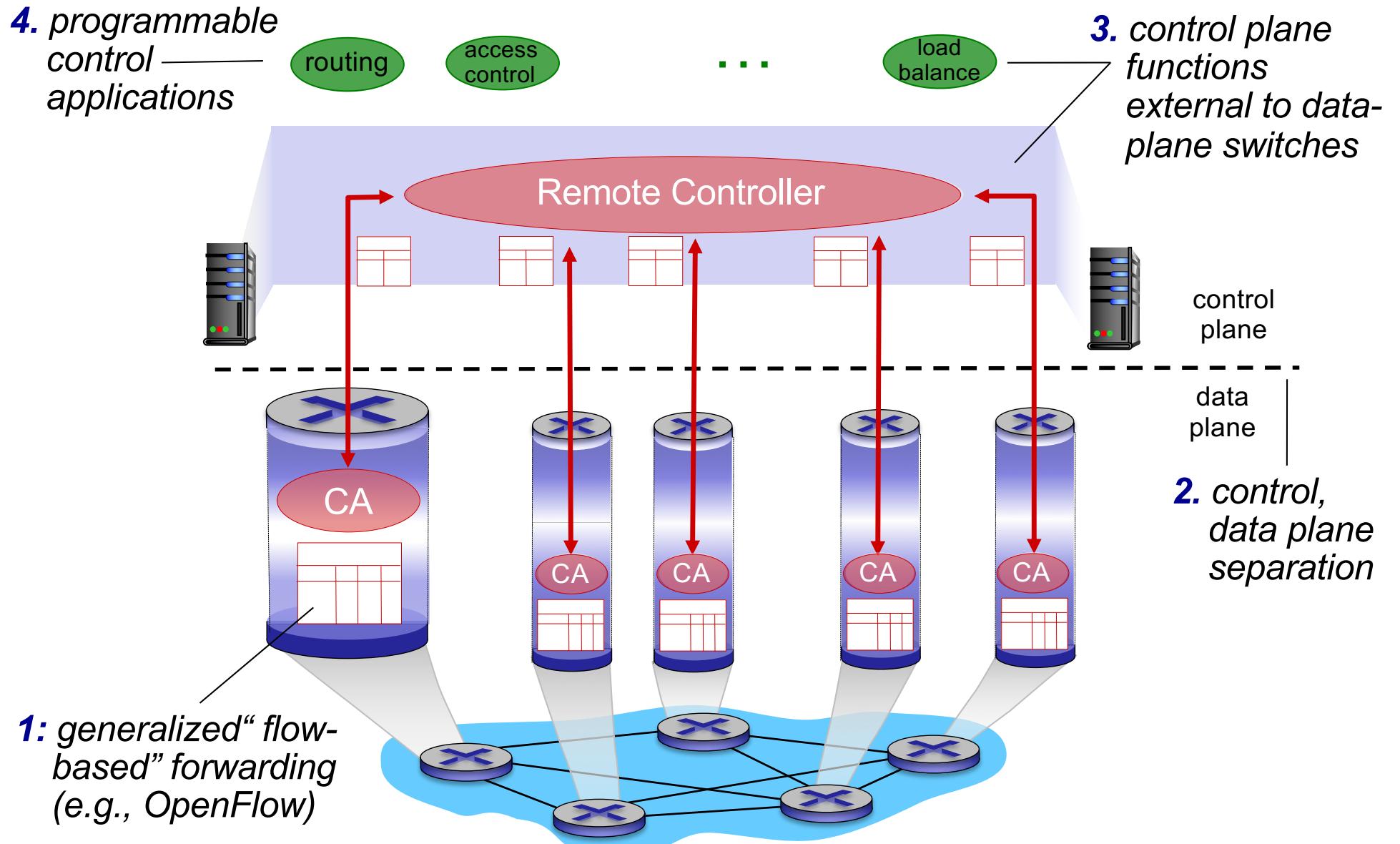
# Traffic engineering: difficult



Q: what if w wants to route blue and red traffic differently?

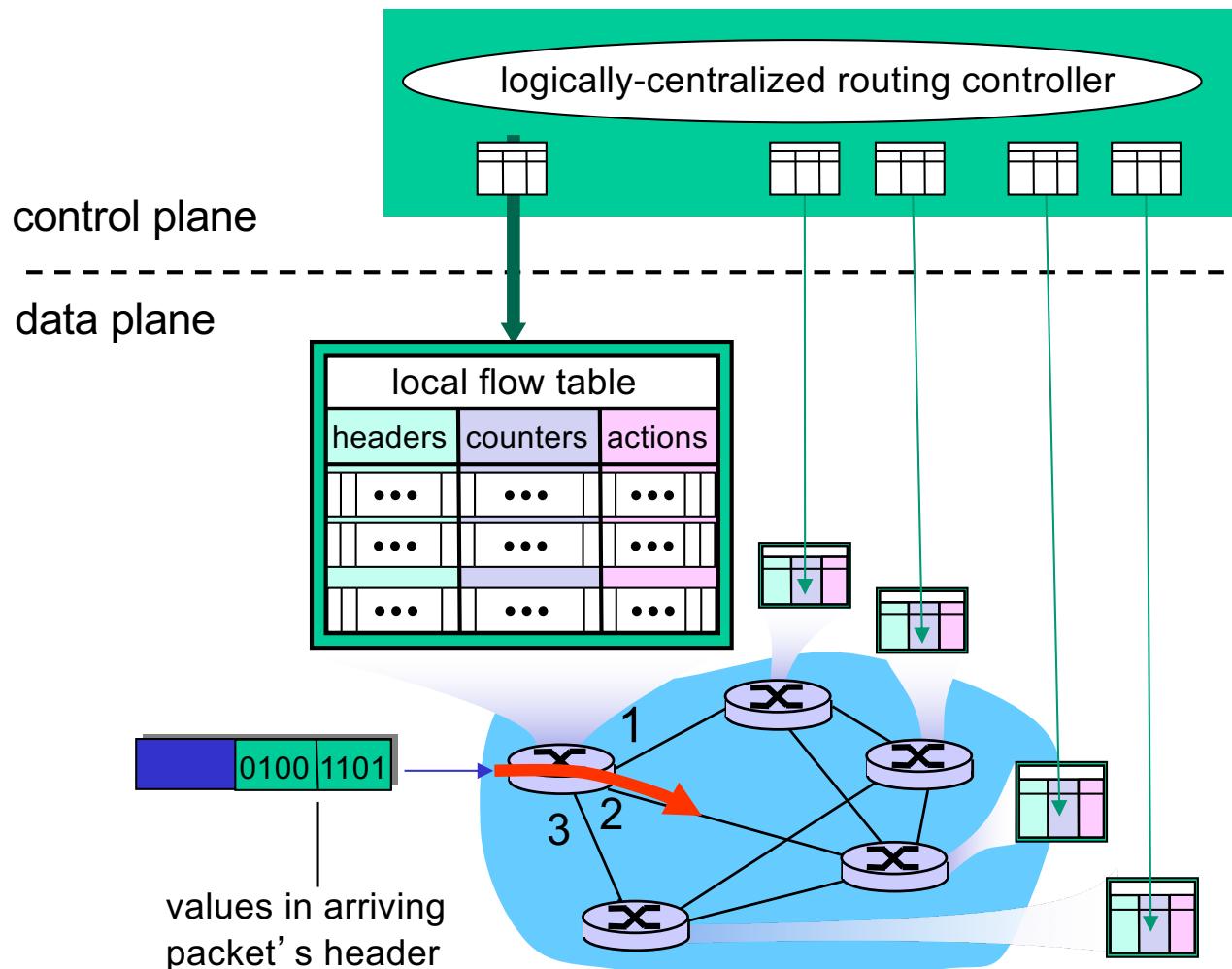
A: can't do it (with destination based forwarding, and LS, DV routing)

# Software defined networking (SDN)



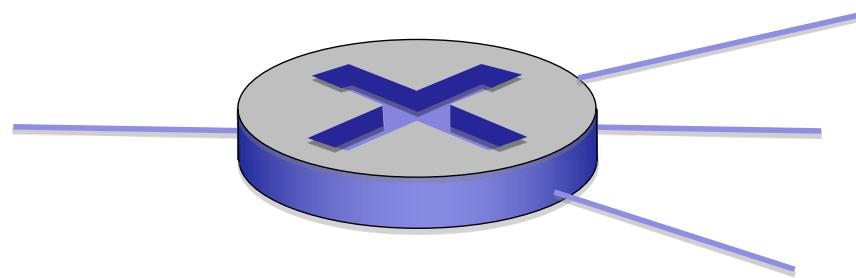
# Generalized Forwarding and SDN

Each router contains a *flow table* that is computed and distributed by a *logically centralized* routing controller



# OpenFlow data plane abstraction

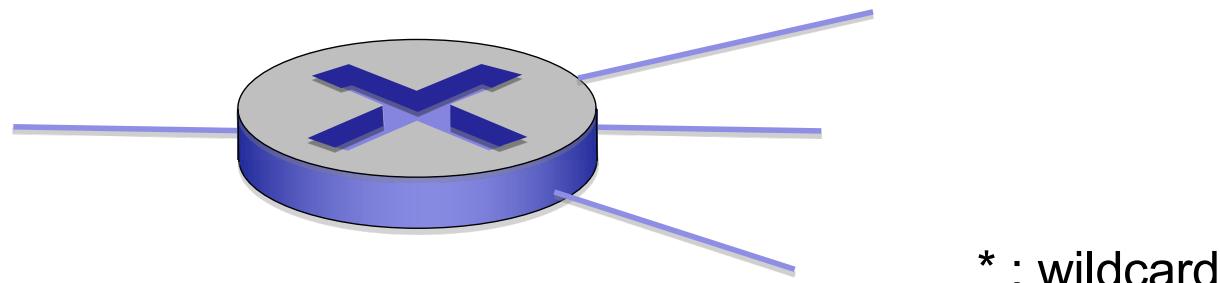
- *flow*: defined by header fields
- generalized forwarding: simple packet-handling rules
  - *Pattern*: match values in packet header fields
  - *Actions: for matched packet*: drop, forward, modify, matched packet or send matched packet to controller
  - *Priority*: disambiguate overlapping patterns
  - *Counters*: #bytes and #packets



*Flow table in a router (computed and distributed by controller) define router's match+action rules*

# OpenFlow data plane abstraction

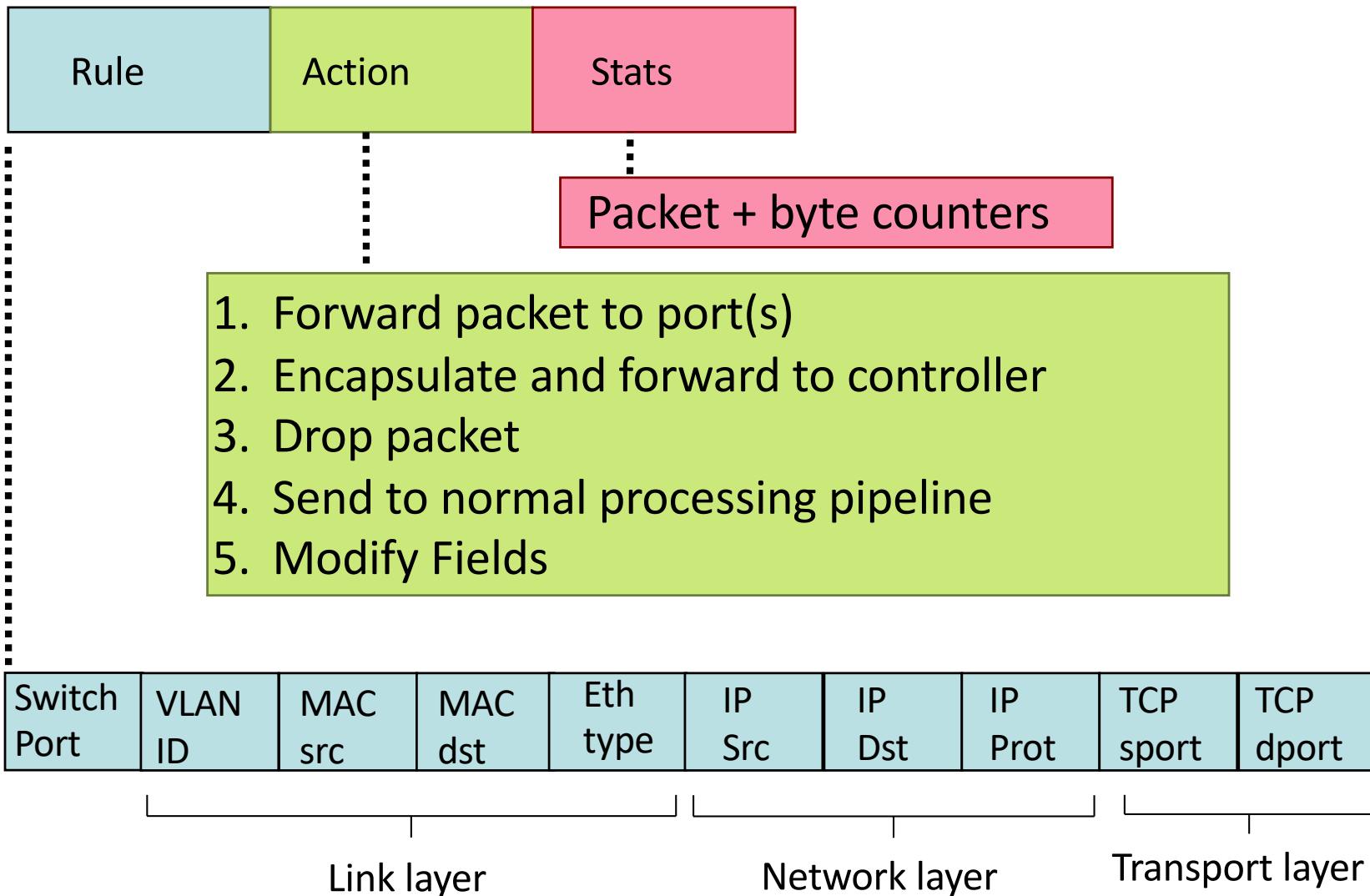
- *flow*: defined by header fields
- generalized forwarding: simple packet-handling rules
  - *Pattern*: match values in packet header fields
  - *Actions: for matched packet*: drop, forward, modify, matched packet or send matched packet to controller
  - *Priority*: disambiguate overlapping patterns
  - *Counters*: #bytes and #packets



\* : wildcard

1.  $\text{src}=1.2.*.*$ ,  $\text{dest}=3.4.5.* \rightarrow \text{drop}$
2.  $\text{src} = *.*.*.*$ ,  $\text{dest}=3.4.*.* \rightarrow \text{forward}(2)$
3.  $\text{src}=10.1.2.3$ ,  $\text{dest} = *.*.*.* \rightarrow \text{send to controller}$

# OpenFlow: Flow Table Entries



# Examples

## Destination-based forwarding:

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

*IP datagrams destined to IP address 51.6.0.8 should be forwarded to router output port 6*

## Firewall:

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 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
*	*	*	*	*	128.119.1.1	*	*	*	*	drop

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

# Examples

Destination-based layer 2 (switch) forwarding:

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

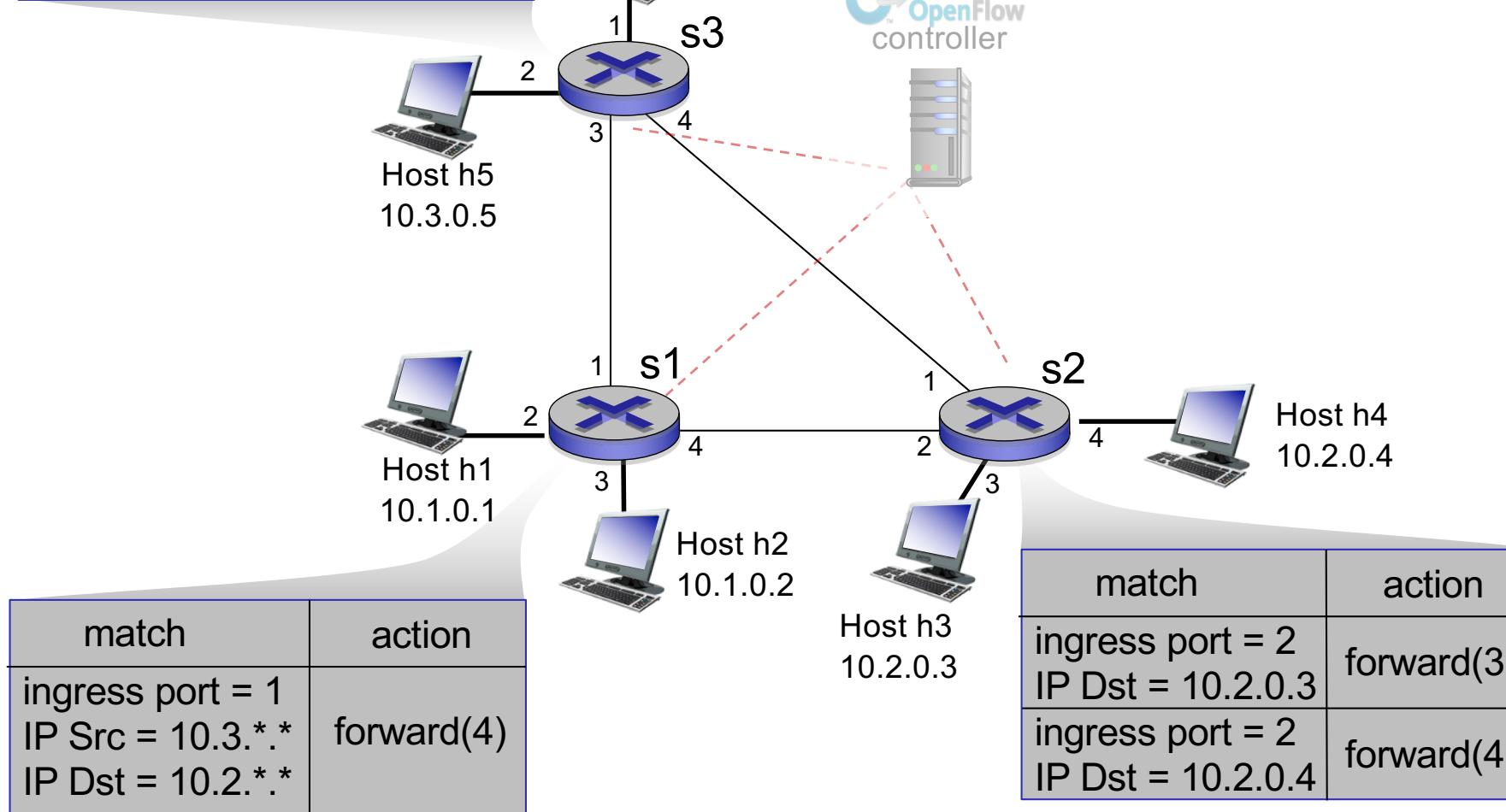
*layer 2 frames from MAC address 22:A7:23:11:E1:02  
should be forwarded to output port 3*

# 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

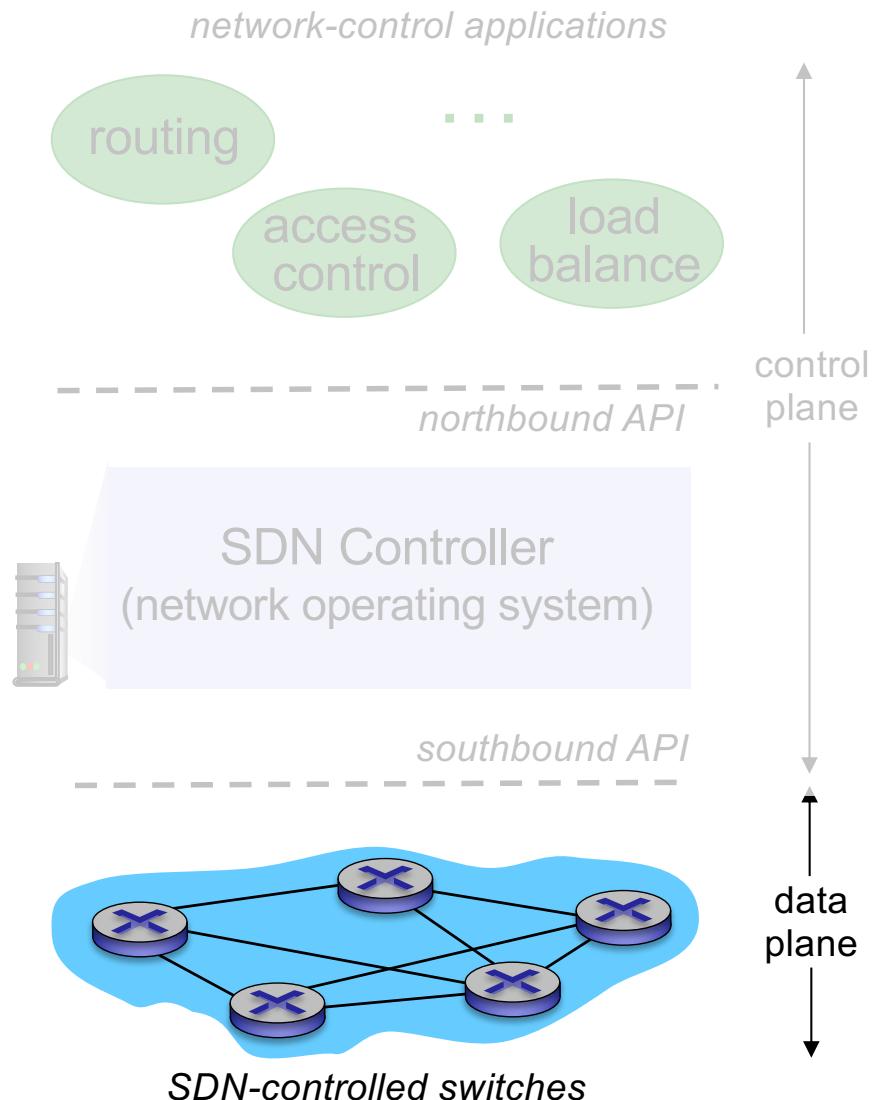
match	action
IP Src = 10.3.*.*	
IP Dst = 10.2.*.*	forward(3)



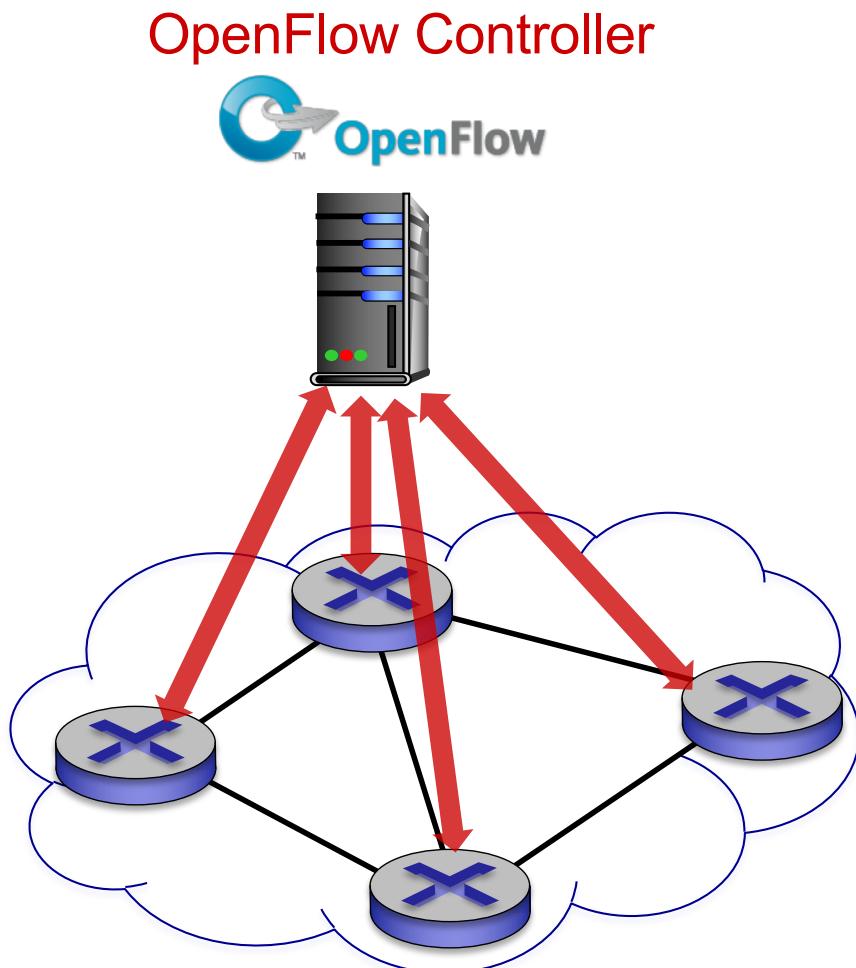
# SDN perspective: data plane switches

## *Data plane switches*

- fast, simple, commodity switches implementing generalized data-plane forwarding (Section 4.4) in hardware
- switch flow table computed, installed by controller
- API for table-based switch control (e.g., OpenFlow)
  - defines what is controllable and what is not
- protocol for communicating with controller (e.g., OpenFlow)



# OpenFlow protocol

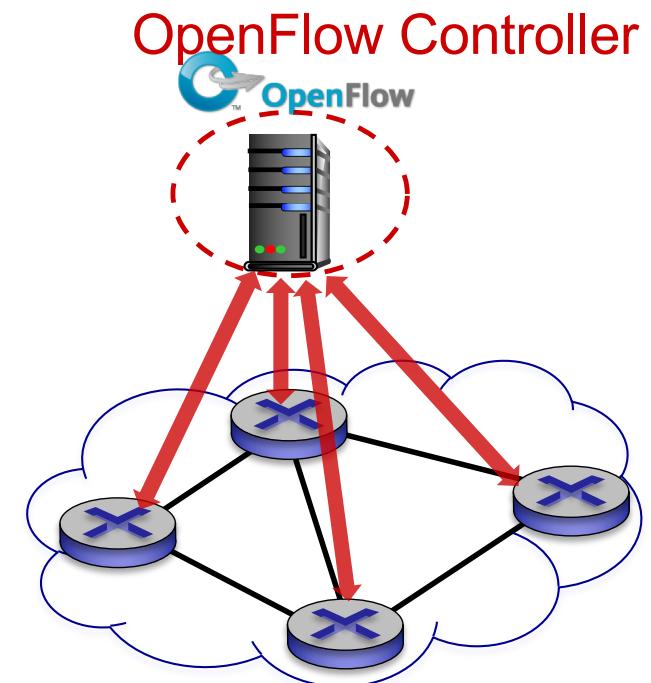


- operates between controller, switch
- TCP used to exchange messages
  - optional encryption
- three classes of OpenFlow messages:
  - controller-to-switch
  - asynchronous (switch to controller)
  - symmetric (misc)

# OpenFlow: controller-to-switch messages

## *Key controller-to-switch messages*

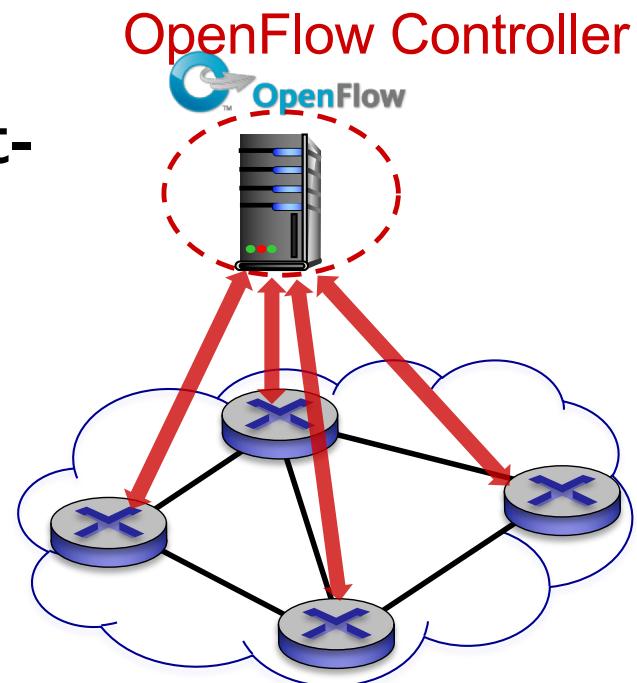
- ***features***: controller queries switch features, switch replies
- ***configure***: controller queries/sets switch configuration parameters
- ***modify-state***: add, delete, modify flow entries in the OpenFlow tables
- ***packet-out***: controller can send this packet out of specific switch port



# OpenFlow: switch-to-controller messages

## Key switch-to-controller messages

- **packet-in:** transfer packet (and its control) to controller. See packet-out message from controller
- **flow-removed:** flow table entry deleted at switch
- **port status:** inform controller of a change on a port.

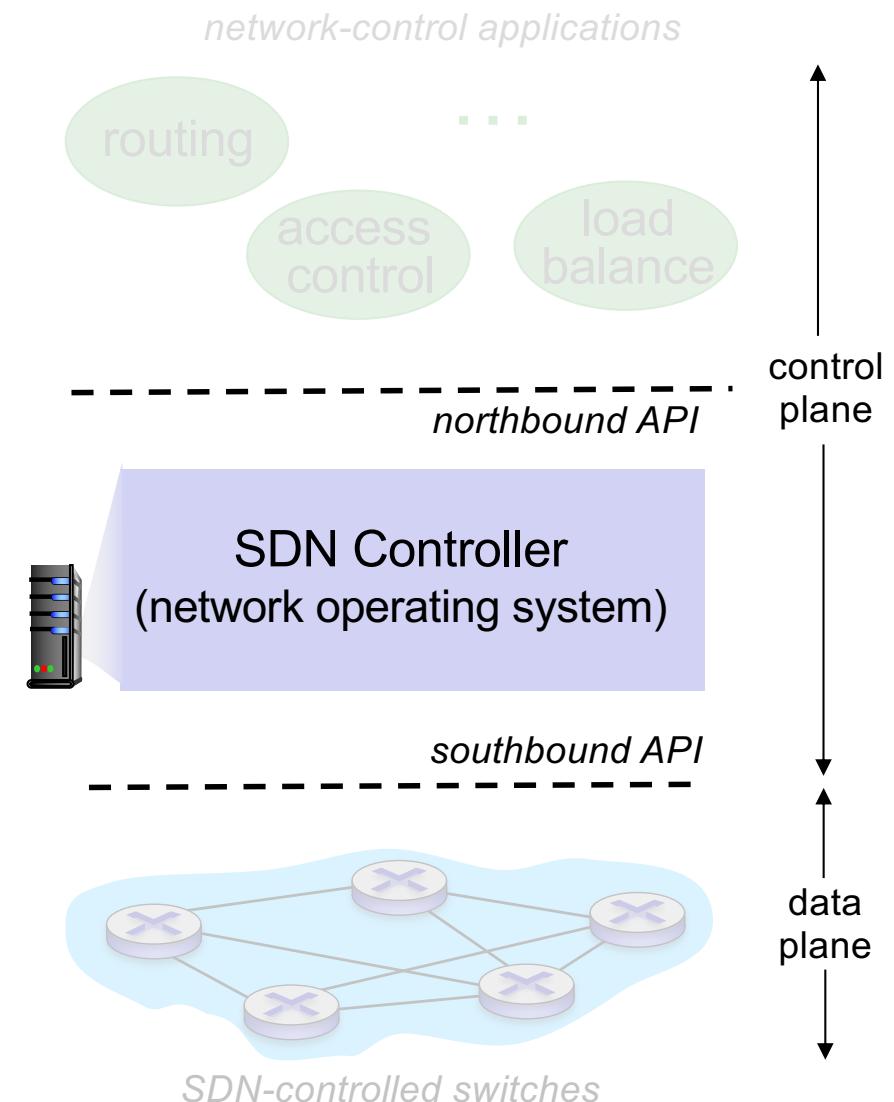


Fortunately, network operators don't “program” switches by creating/sending OpenFlow messages directly. Instead use higher-level abstraction at controller

# SDN perspective: SDN controller

## *SDN controller (network OS):*

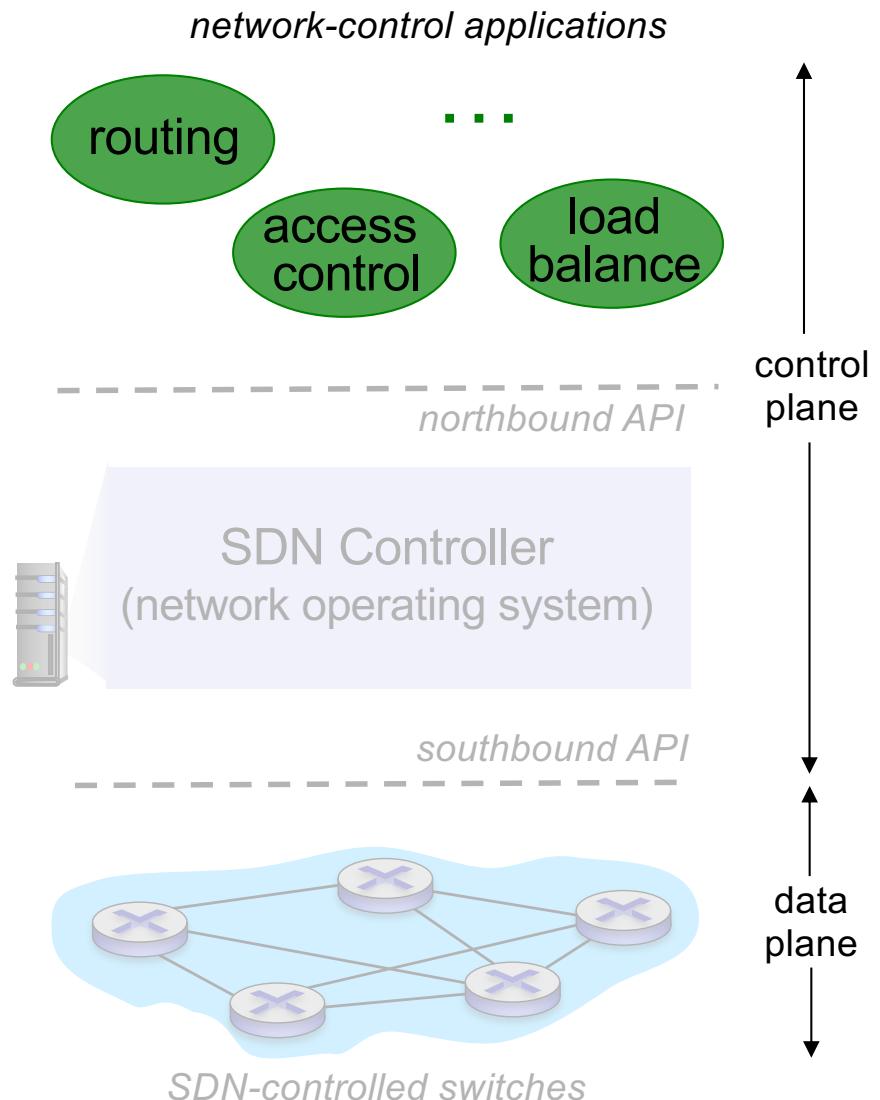
- maintain network state information
- interacts with network control applications “above” via northbound API
- interacts with network switches “below” via southbound API
- implemented as distributed system for performance, scalability, fault-tolerance, robustness



# SDN perspective: control applications

## *network-control apps:*

- “brains” of control: implement control functions using lower-level services, API provided by SDN controller
- *unbundled*: can be provided by 3<sup>rd</sup> party: distinct from routing vendor, or SDN controller

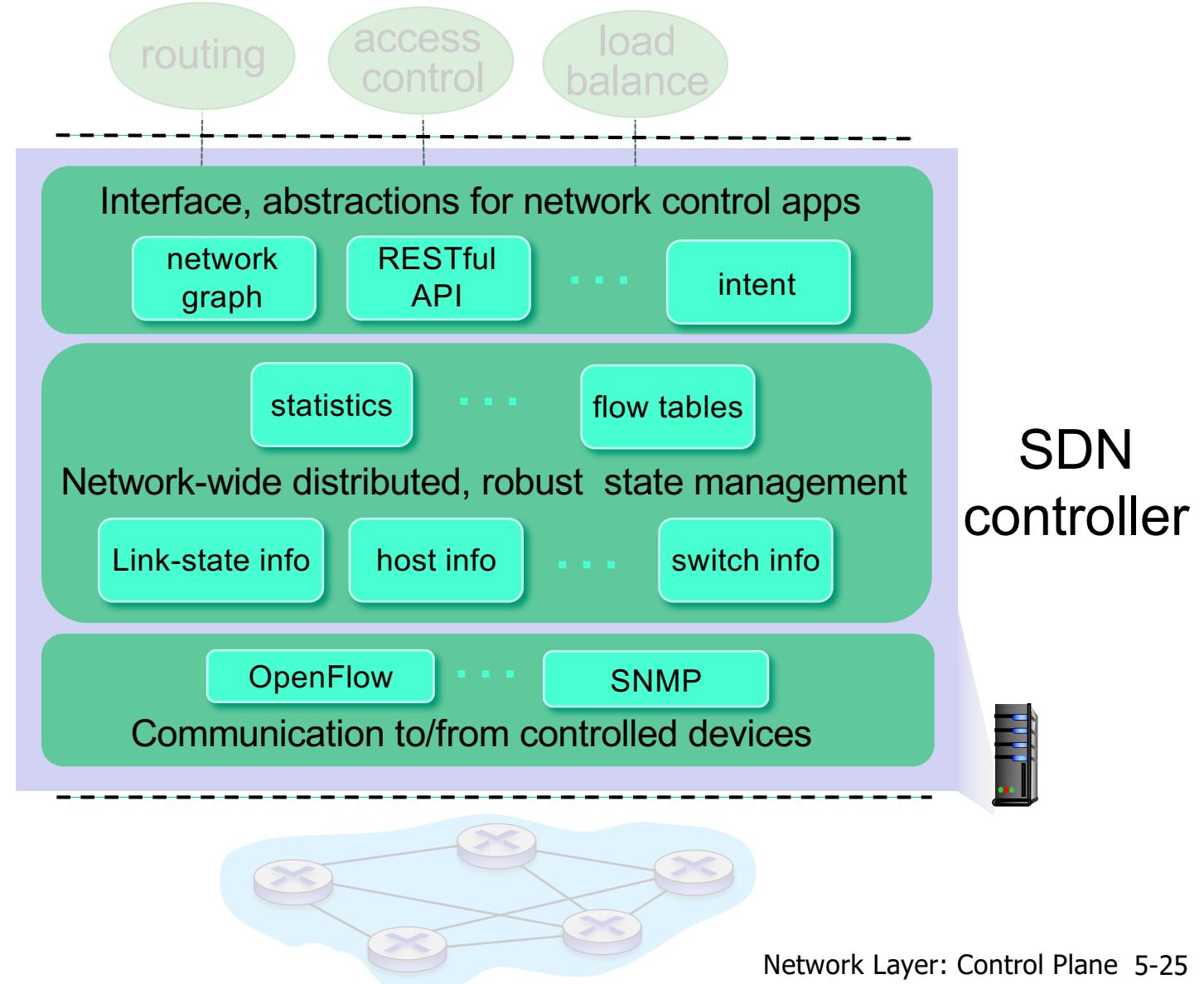


# Components of SDN controller

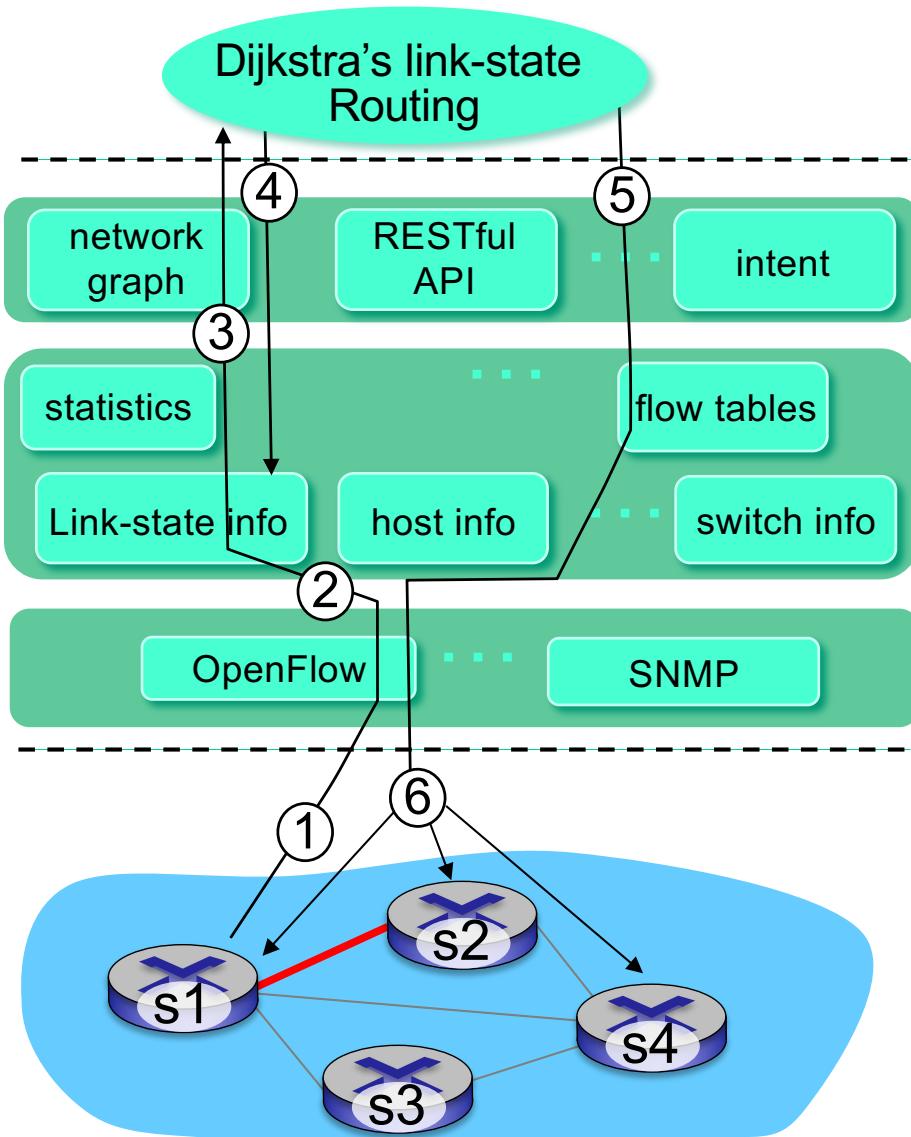
Interface layer to network control apps: abstractions API

Network-wide state management layer: state of networks links, switches, services: a *distributed database*

*communication layer*: communicate between SDN controller and controlled switches

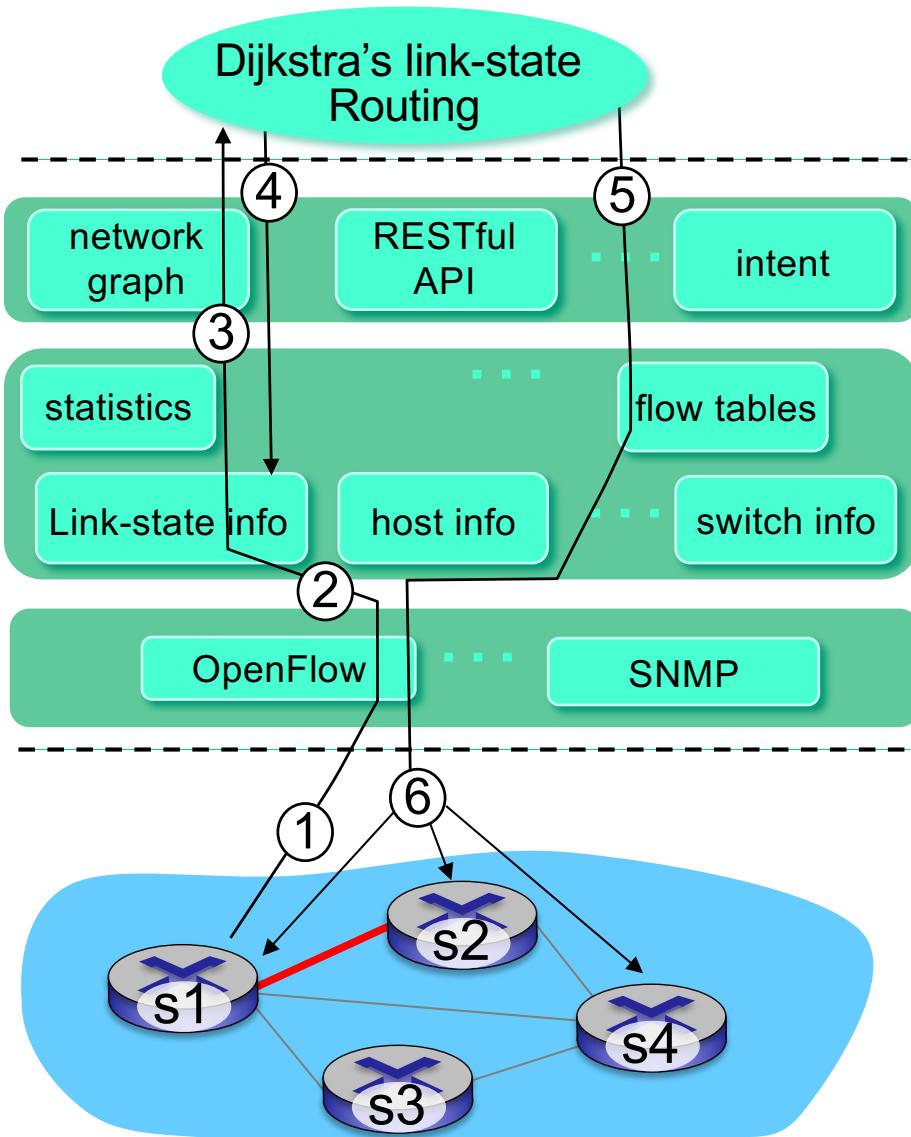


# SDN: control/data plane interaction example



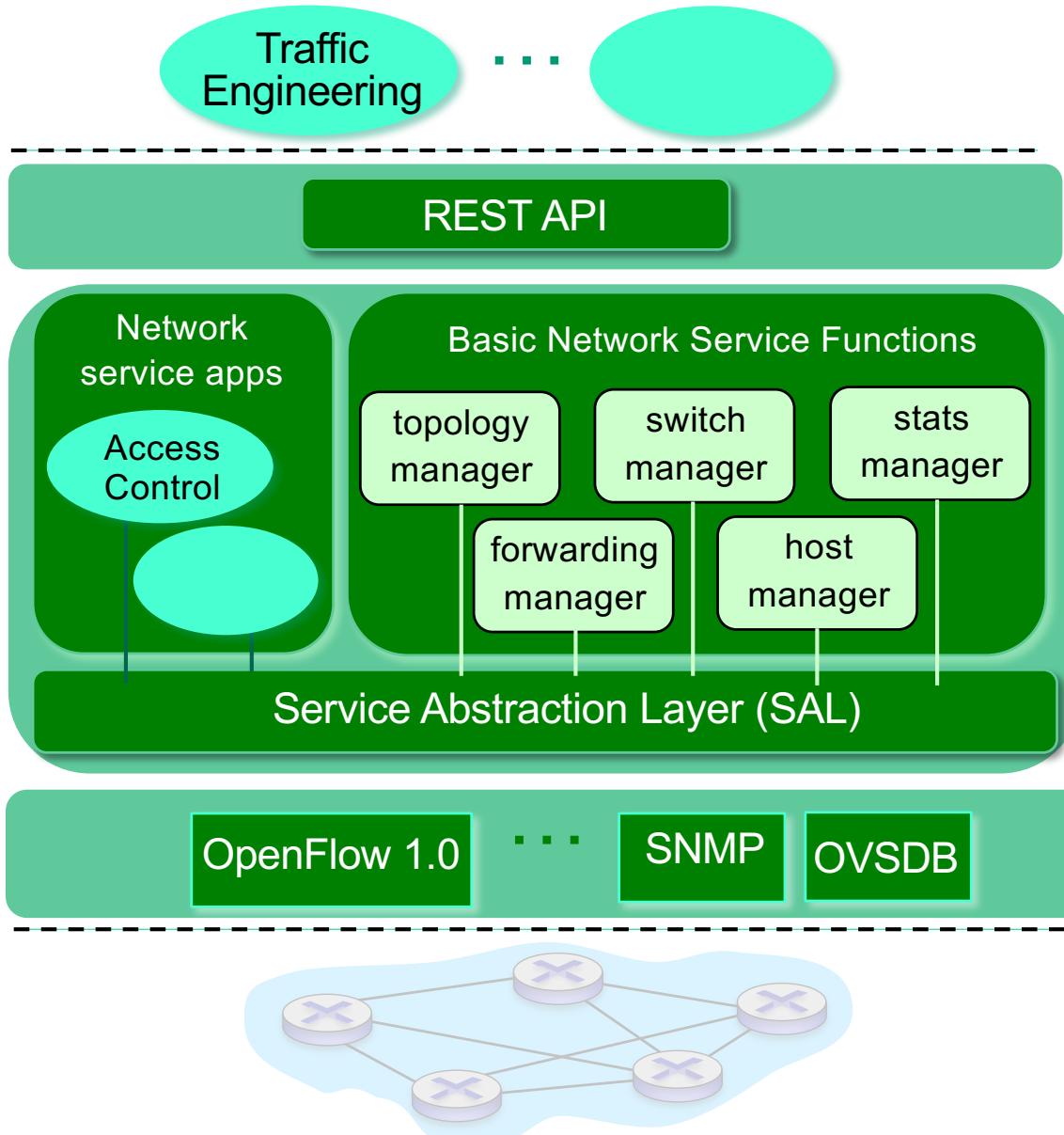
- ① SI, experiencing link failure using OpenFlow port status message to notify controller
- ② SDN controller receives OpenFlow message, updates link status info
- ③ Dijkstra's routing algorithm application has previously registered to be called when ever link status changes. It is called.
- ④ Dijkstra's routing algorithm access network graph info, link state info in controller, computes new routes

# SDN: control/data plane interaction example



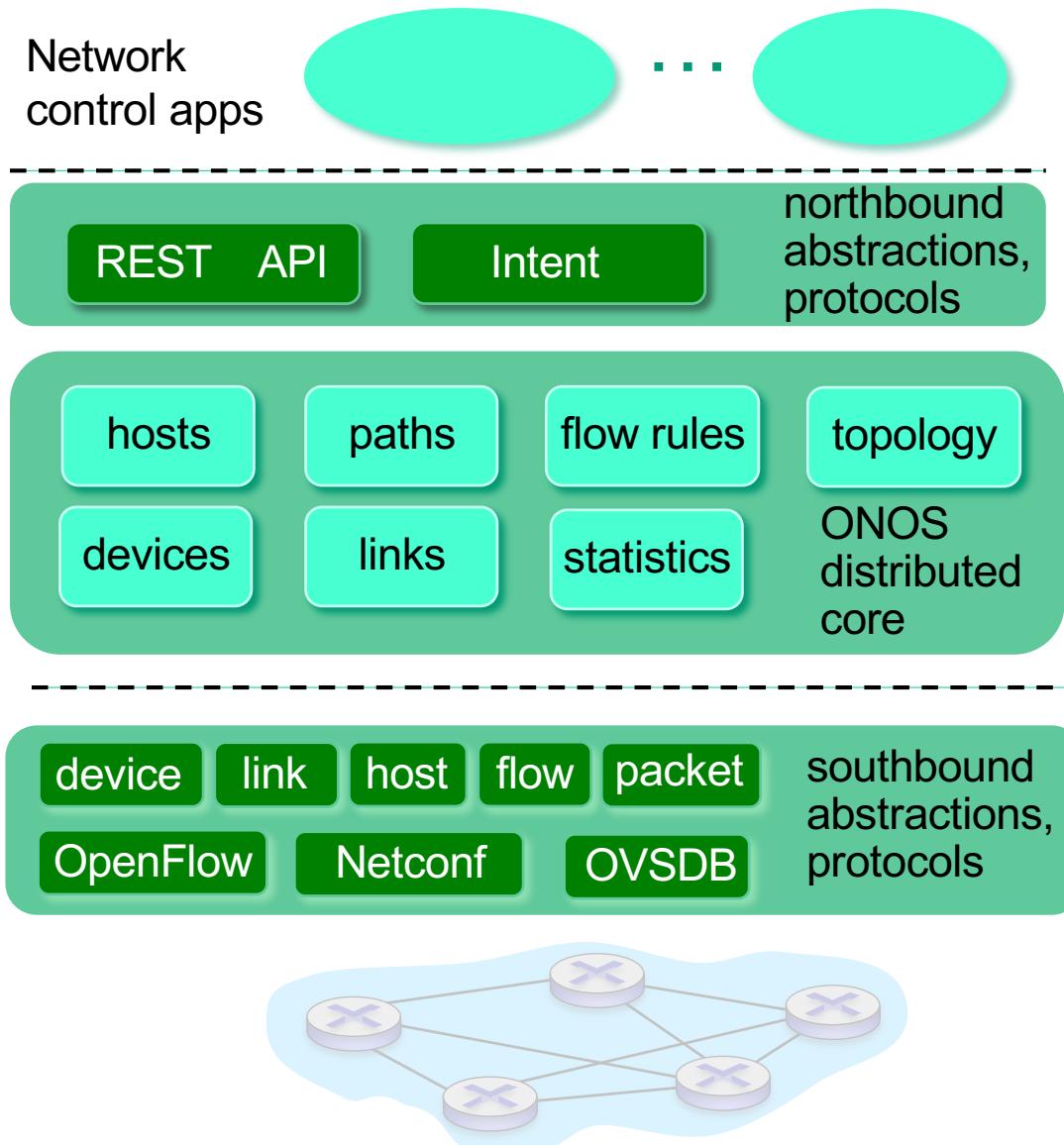
- ⑤ link state routing app interacts with flow-table-computation component in SDN controller, which computes new flow tables needed
- ⑥ Controller uses OpenFlow to install new tables in switches that need updating

# OpenDaylight (ODL) controller



- ODL Lithium controller
- network apps may be contained within, or be external to SDN controller
- Service Abstraction Layer: interconnects internal, external applications and services

# ONOS controller



- **control apps separate from controller**
- **intent framework:** high-level specification of service: what rather than how
- **considerable emphasis on distributed core:** service reliability, replication performance scaling

# SDN: selected challenges

- hardening the control plane: dependable, reliable, performance-scalable, secure distributed system
  - robustness to failures: leverage strong theory of reliable distributed system for control plane
  - dependability, security: “baked in” from day one?
- networks, protocols meeting mission-specific requirements
  - e.g., real-time, ultra-reliable, ultra-secure
- Internet-scaling

# What is network management?

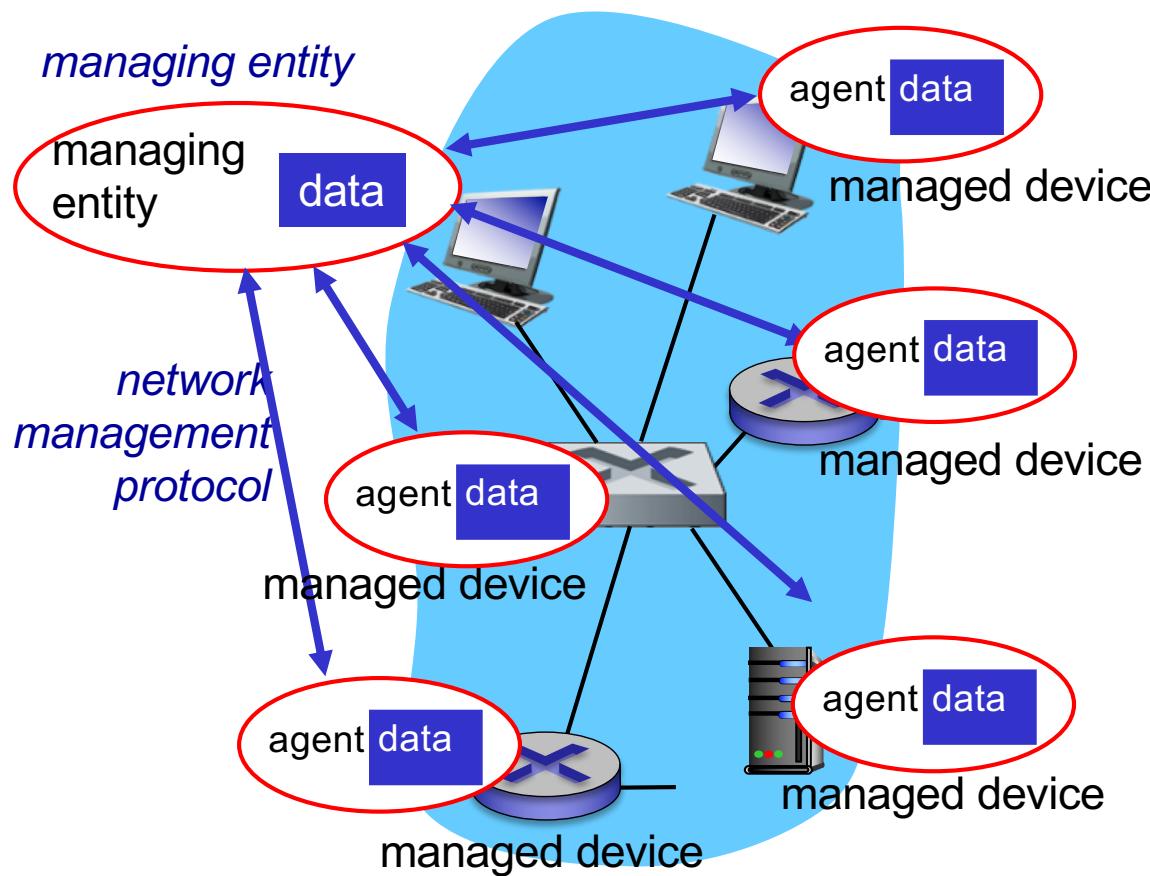
- **autonomous systems (aka “network”):** 1000s of interacting hardware/software components
- other complex systems requiring monitoring, control:
  - jet airplane
  - nuclear power plant
  - others?



"**Network management** includes the deployment, integration and coordination of the hardware, software, and human elements to monitor, test, poll, configure, analyze, evaluate, and control the network and element resources to meet the real-time, operational performance, and Quality of Service requirements at a reasonable cost."

# Infrastructure for network management

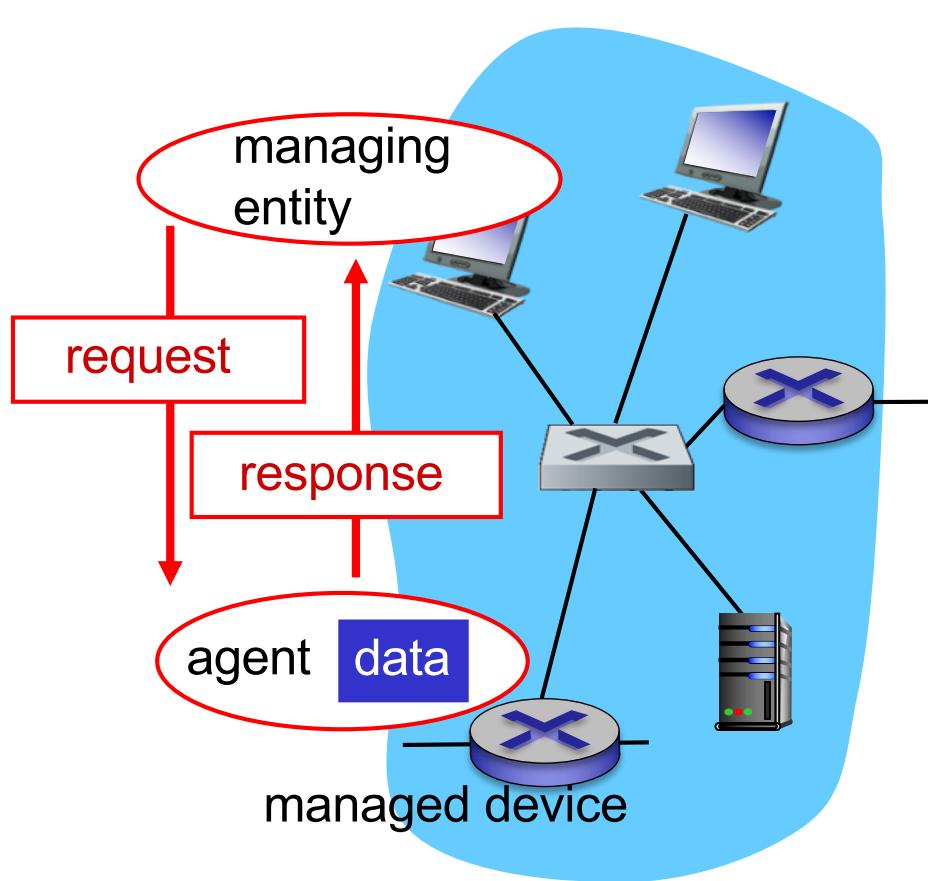
definitions:



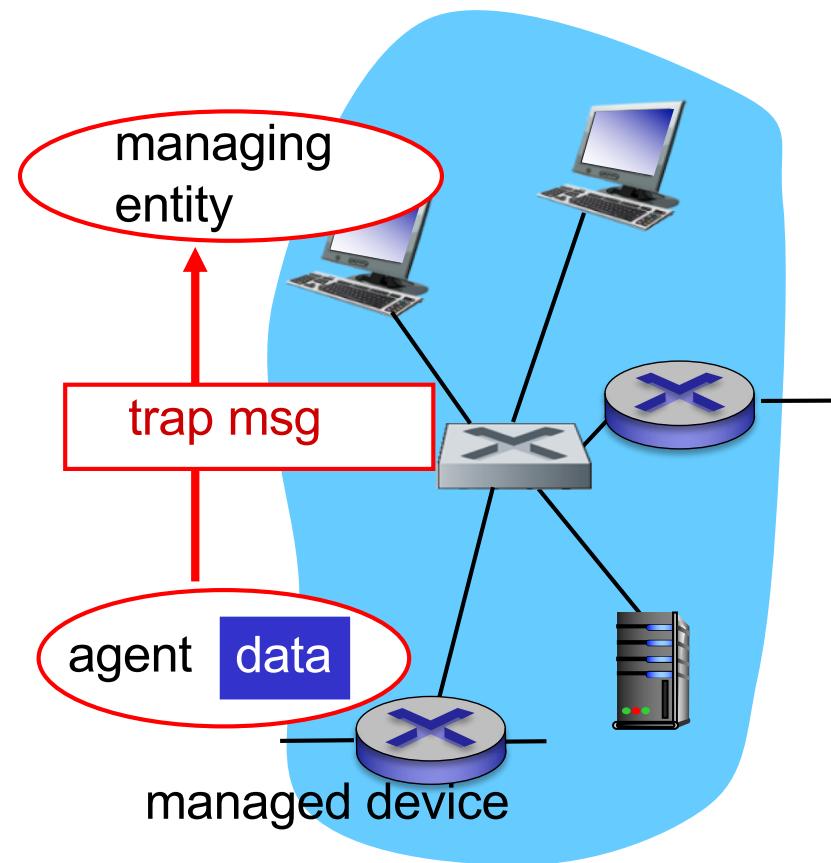
*managed devices contain managed objects whose data is gathered into a Management Information Base (MIB)*

# SNMP protocol

Two ways to convey MIB info, commands:



request/response mode

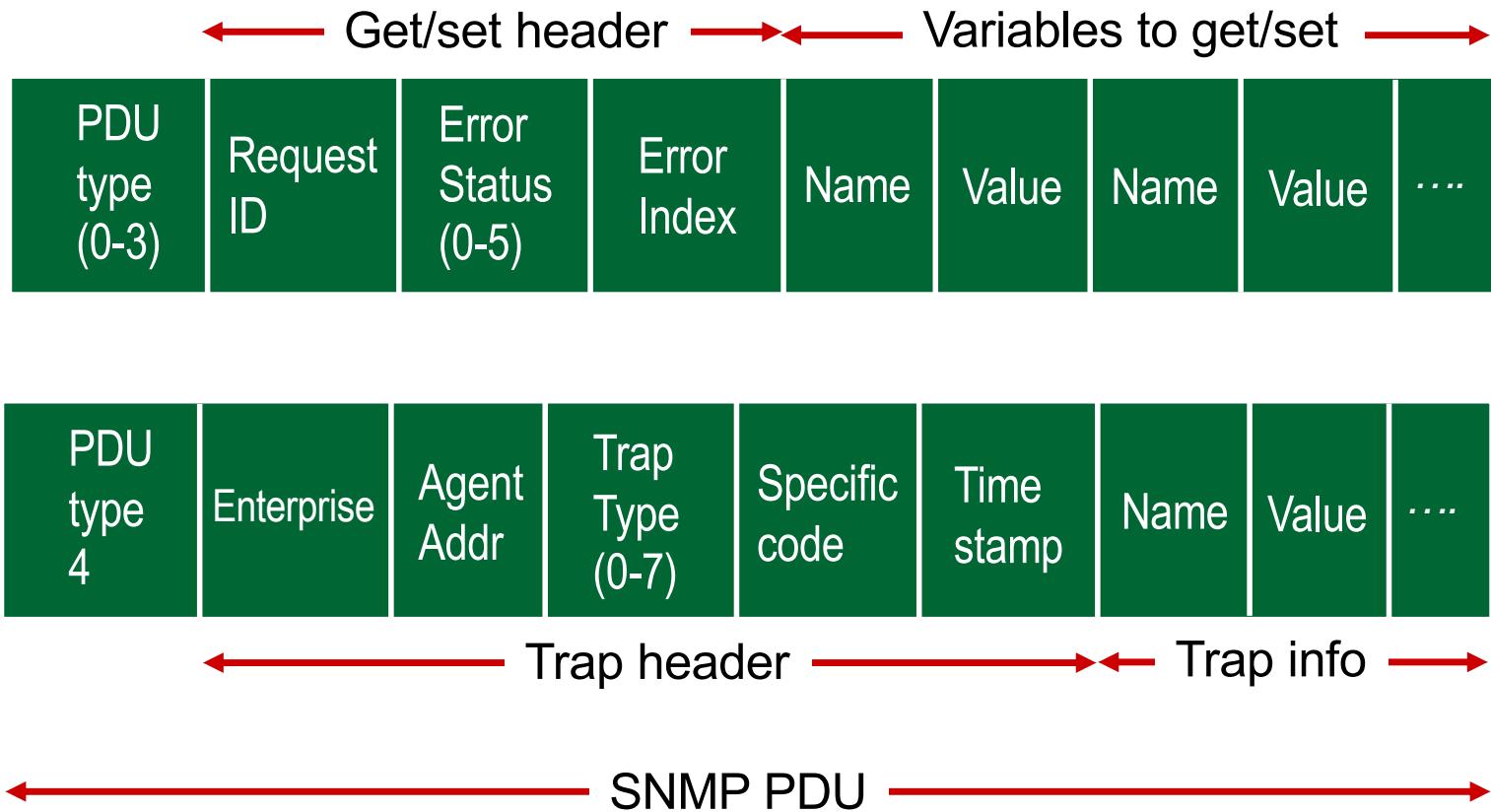


trap mode

# SNMP protocol: message types

<u>Message type</u>	<u>Function</u>
GetRequest GetNextRequest GetBulkRequest	manager-to-agent: “get me data” (data instance, next data in list, block of data)
InformRequest	manager-to-manager: here’s MIB value
SetRequest	manager-to-agent: set MIB value
Response	Agent-to-manager: value, response to Request
Trap	Agent-to-manager: inform manager of exceptional event

# SNMP protocol: message formats



*More on network management: see earlier editions of text!*