Introduction to Computer Networks

CS640

Software-Defined Networking

https://pages.cs.wisc.edu/~mgliu/CS640/S26/index.html

Ming Liu
mgliu@cs.wisc.edu

Outline

- Last
  - Link State Routing
- Today
  - Software-Defined Networking
- Announcements
  - Quiz2 today

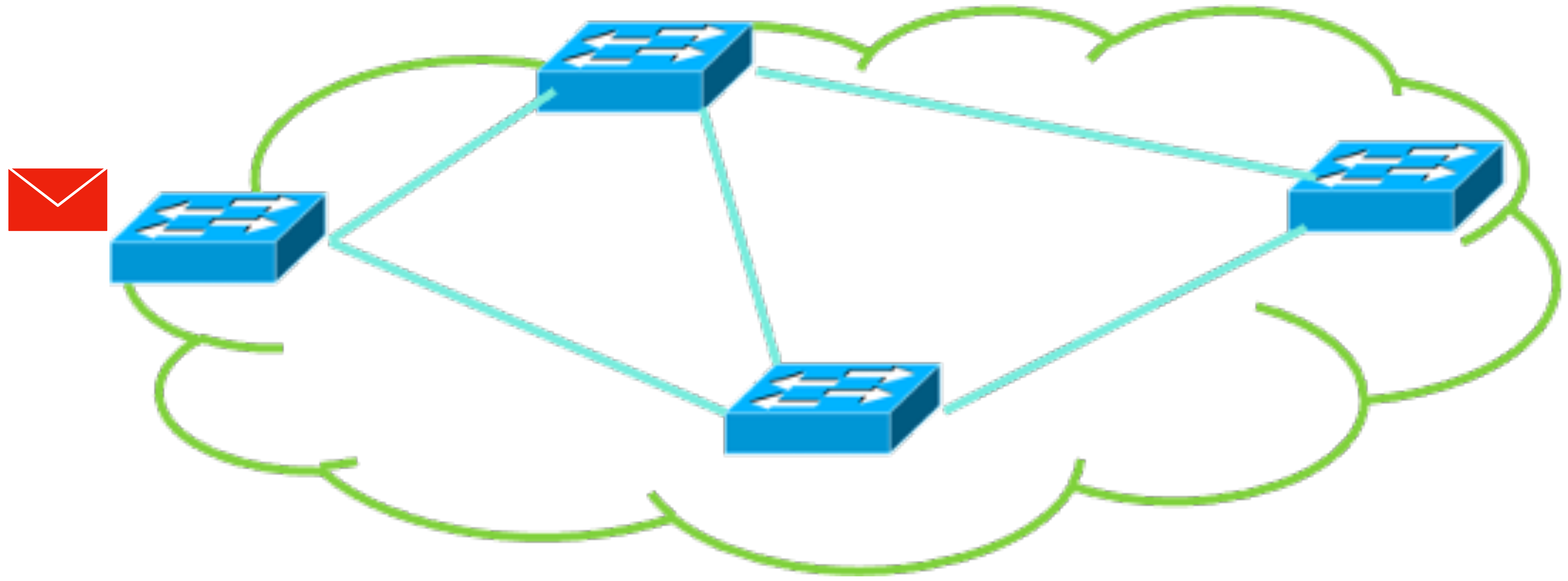
Recap: Link State Routing

- Key Questions:
  - How does link state routing tackle the issue of distance vector routing?
  - What are the limitations behind link state routing?

- Terminology
  - Reliable flooding
  - Link state packet
  - Dijkstra algorithm
  - OSPF

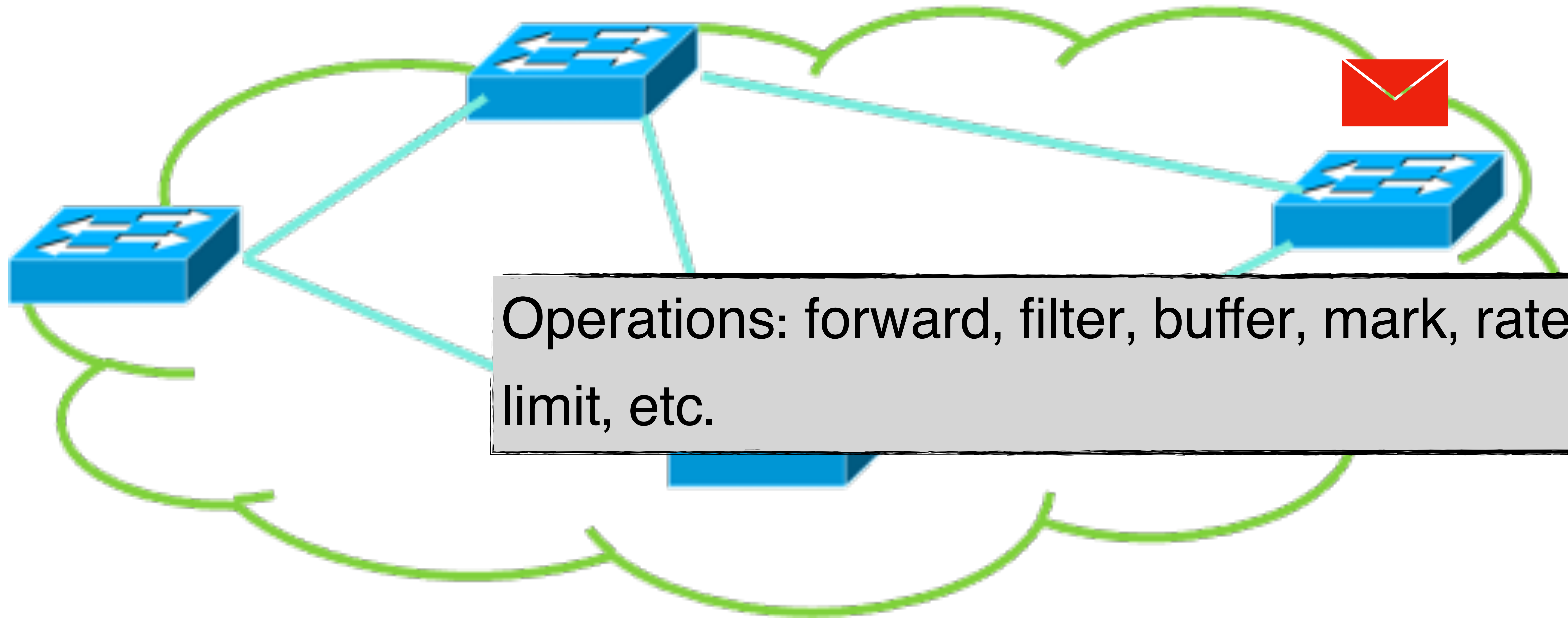
Traditional L3 Networks: Data Plane

- Data plane
  - Move packets across different devices along the communication path



Traditional L3 Networks: Data Plane

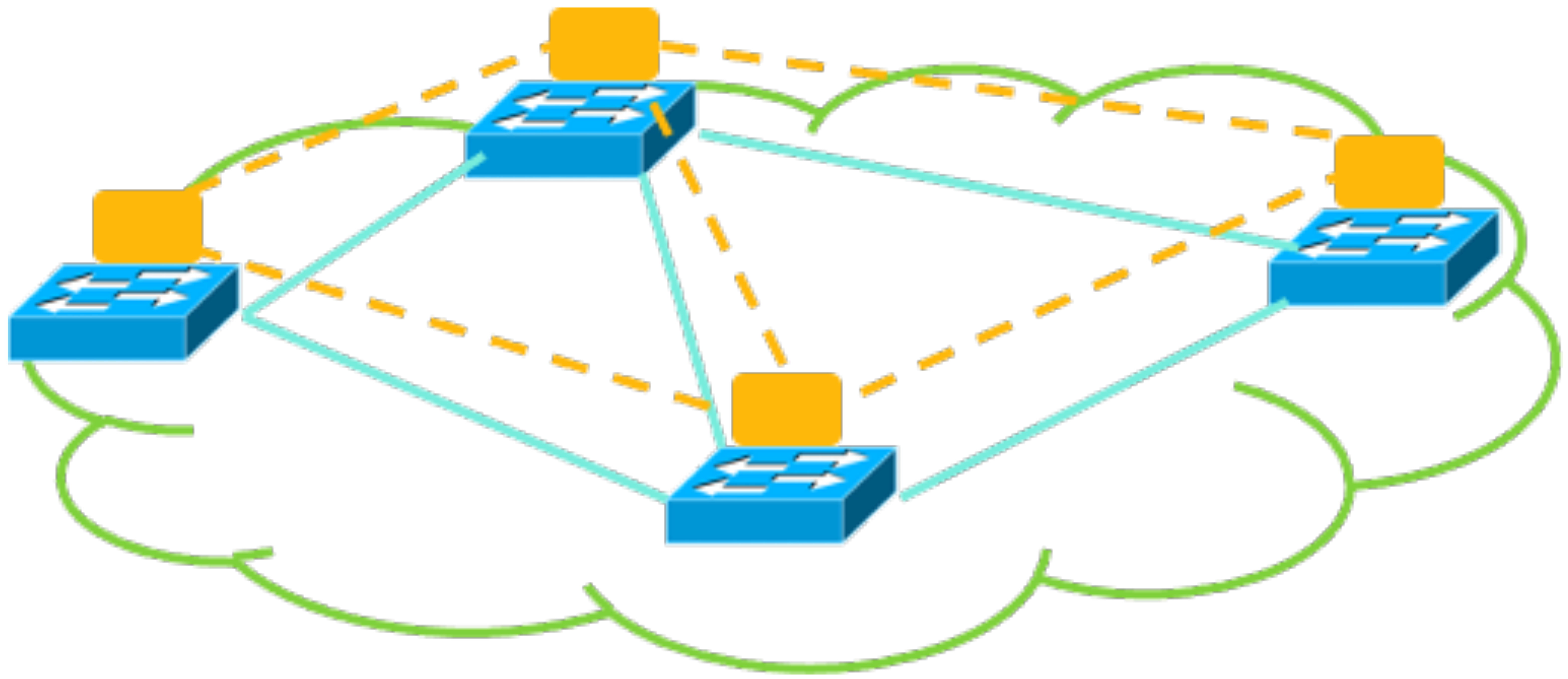
- Data plane
  - Move packets across different devices along the communication path



Operations: forward, filter, buffer, mark, rate-limit, etc.

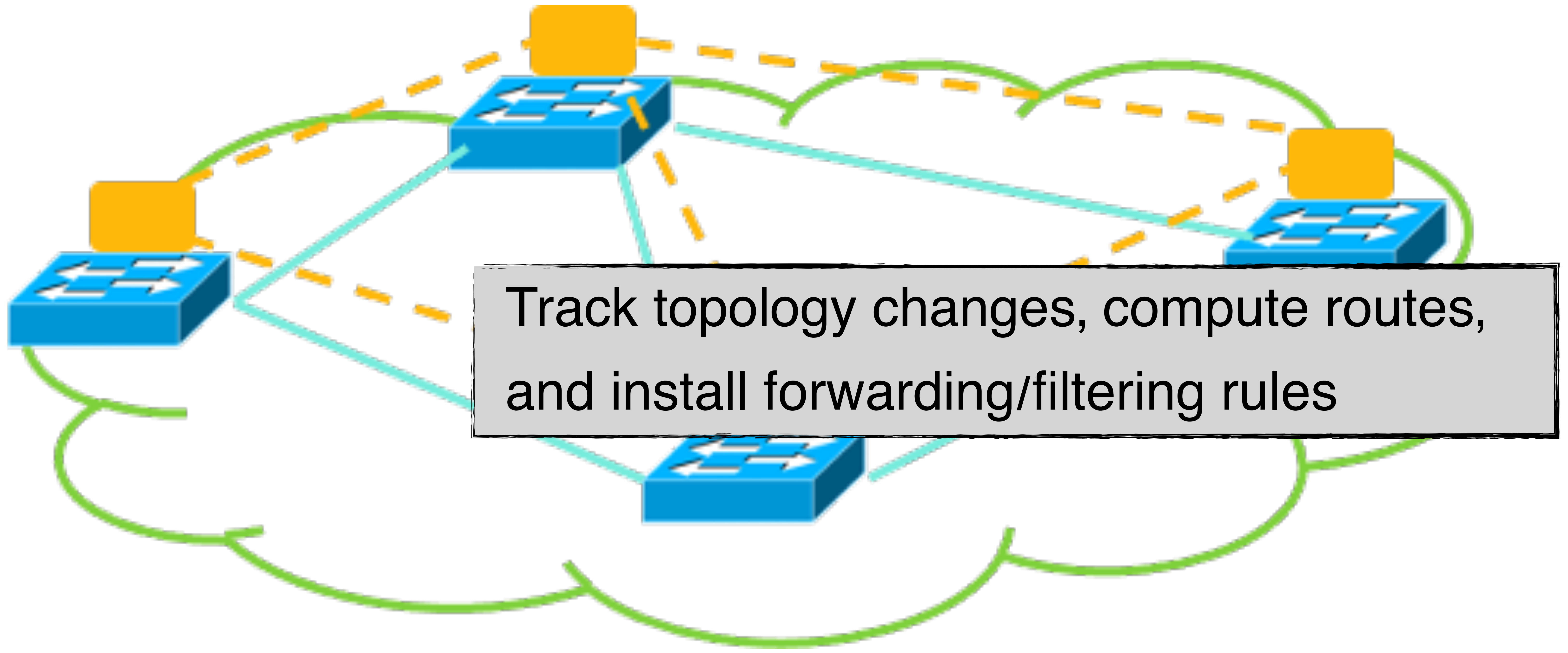
Traditional L3 Networks: Control Plane

- Control plane
  - Decide the packet communication path via the distributed algorithm



Traditional L3 Networks: Control Plane

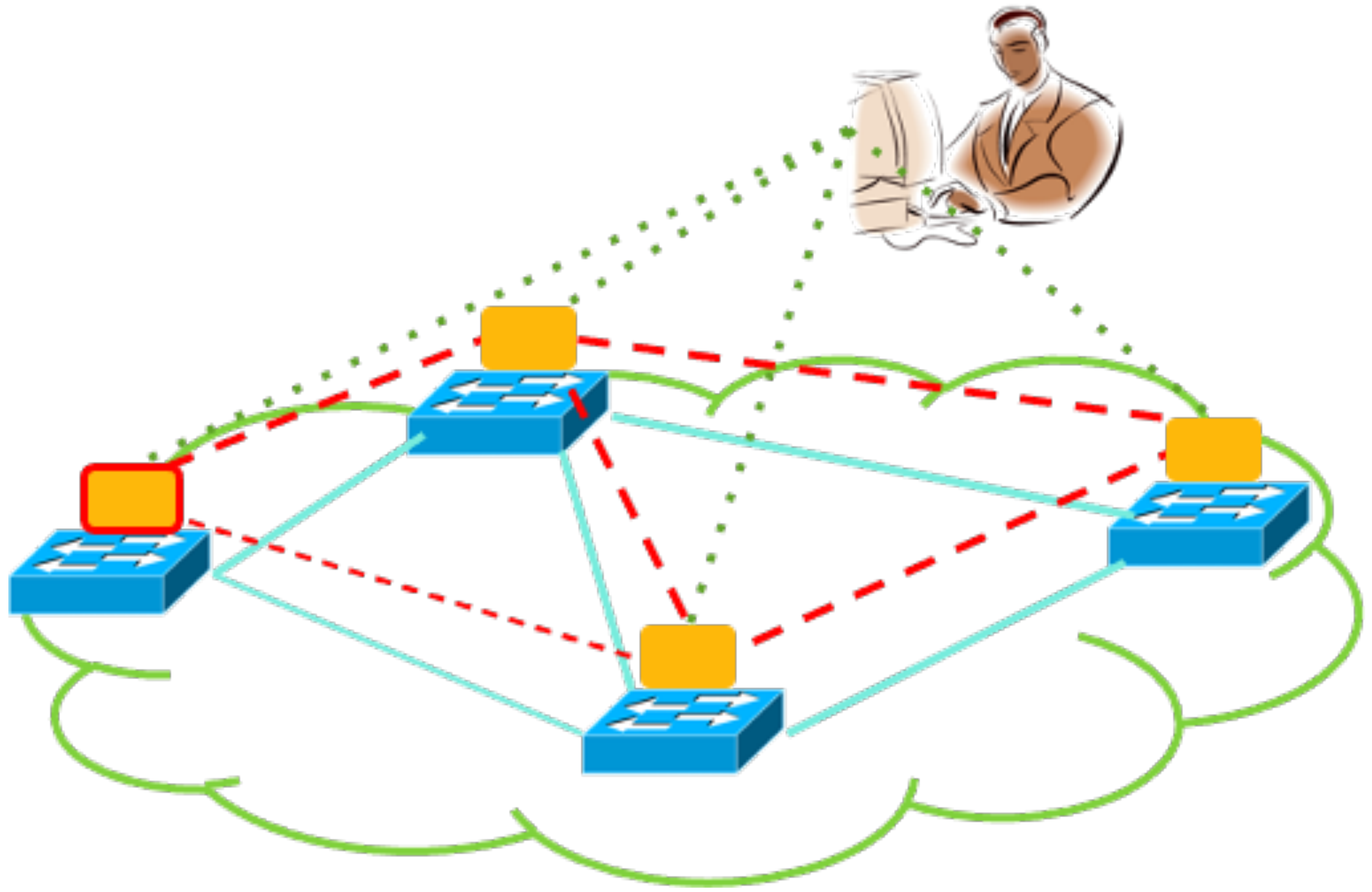
- Control plane
  - Decide the packet communication path via the distributed algorithm



Track topology changes, compute routes, and install forwarding/filtering rules

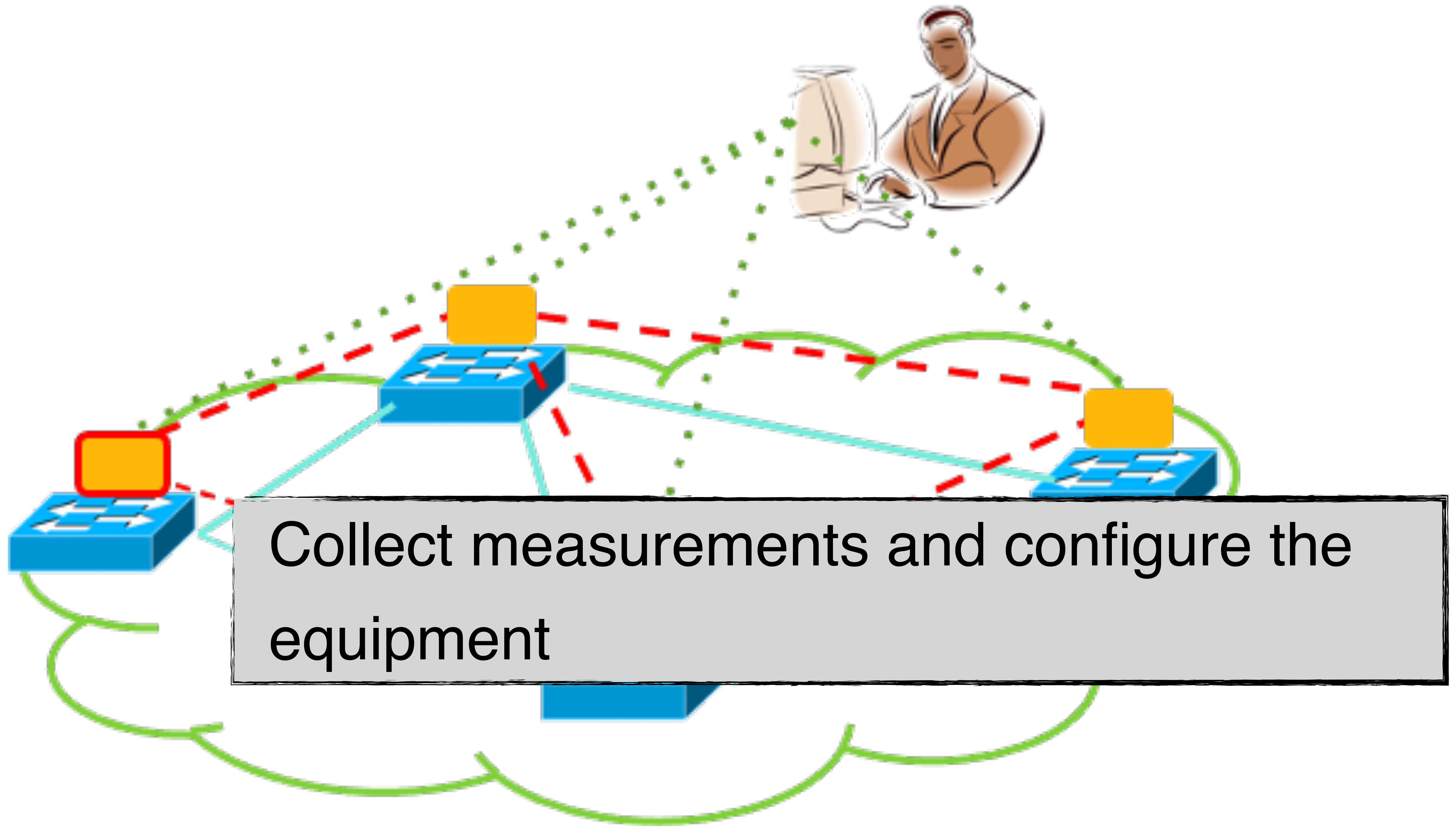
Traditional L3 Networks: Management Plane

- Management plane
  - Configure, monitor, and management the communication device



Traditional L3 Networks: Management Plane

- Management plane
  - Configure, monitor, and management the communication device

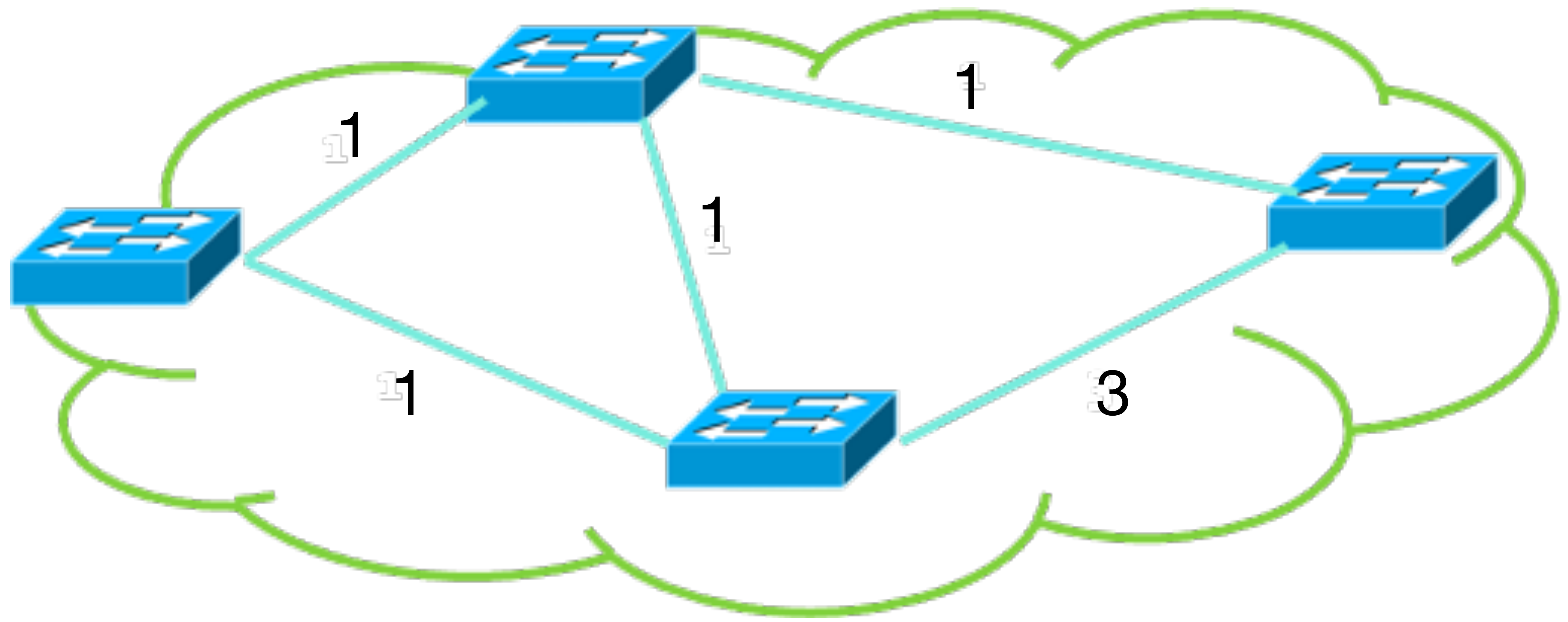

Collect measurements and configure the equipment

Shortest Path Routing

- Management plane: set the link weights
- Control plane: compute shortest paths via the routing algorithm
- Data plane: forward packets to the next hop

Shortest Path Routing

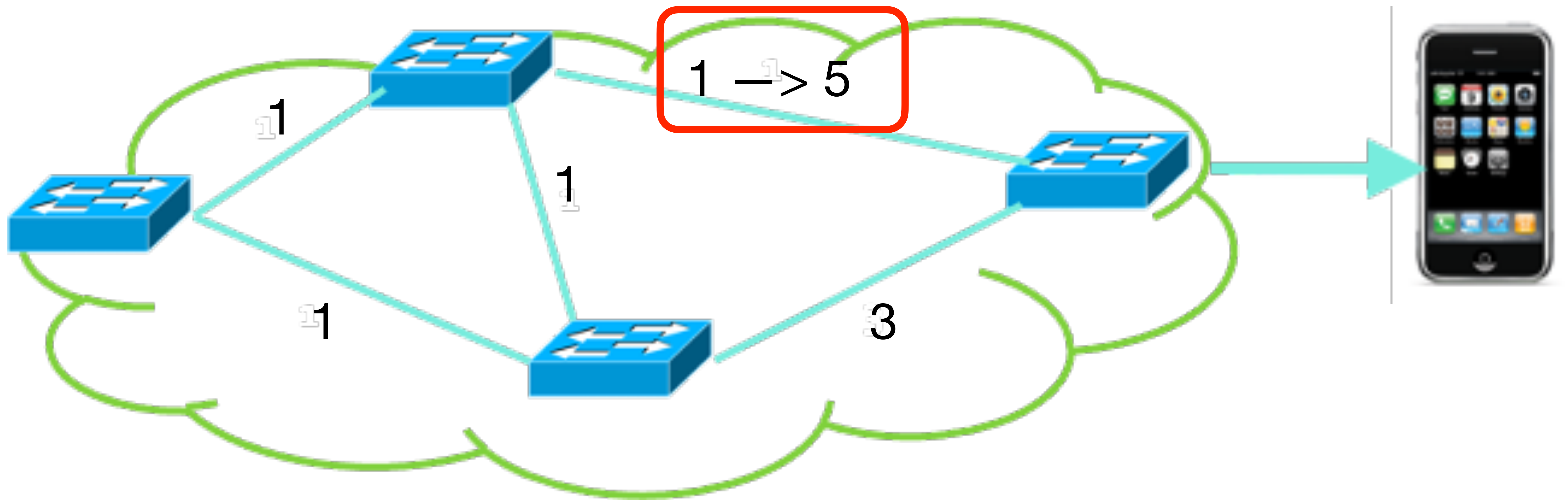
- Management plane: set the link weights
- Control plane: compute shortest paths via the routing algorithm
- Data plane: forward packets to the next hop



How can we introduce new designs in the networking layer?

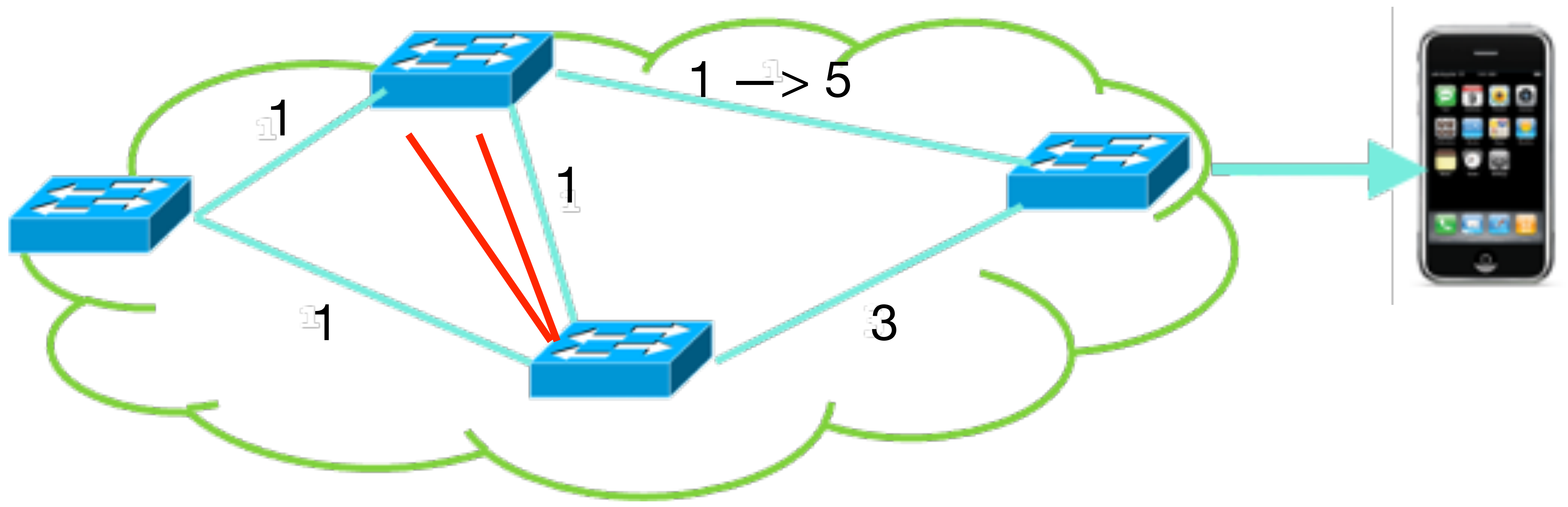
Example #1: Adaptive Routing

- Integrate traffic engineering into the routing algorithm
  - Change link weights
  - Add a new routing path
  - Avoid the congested path

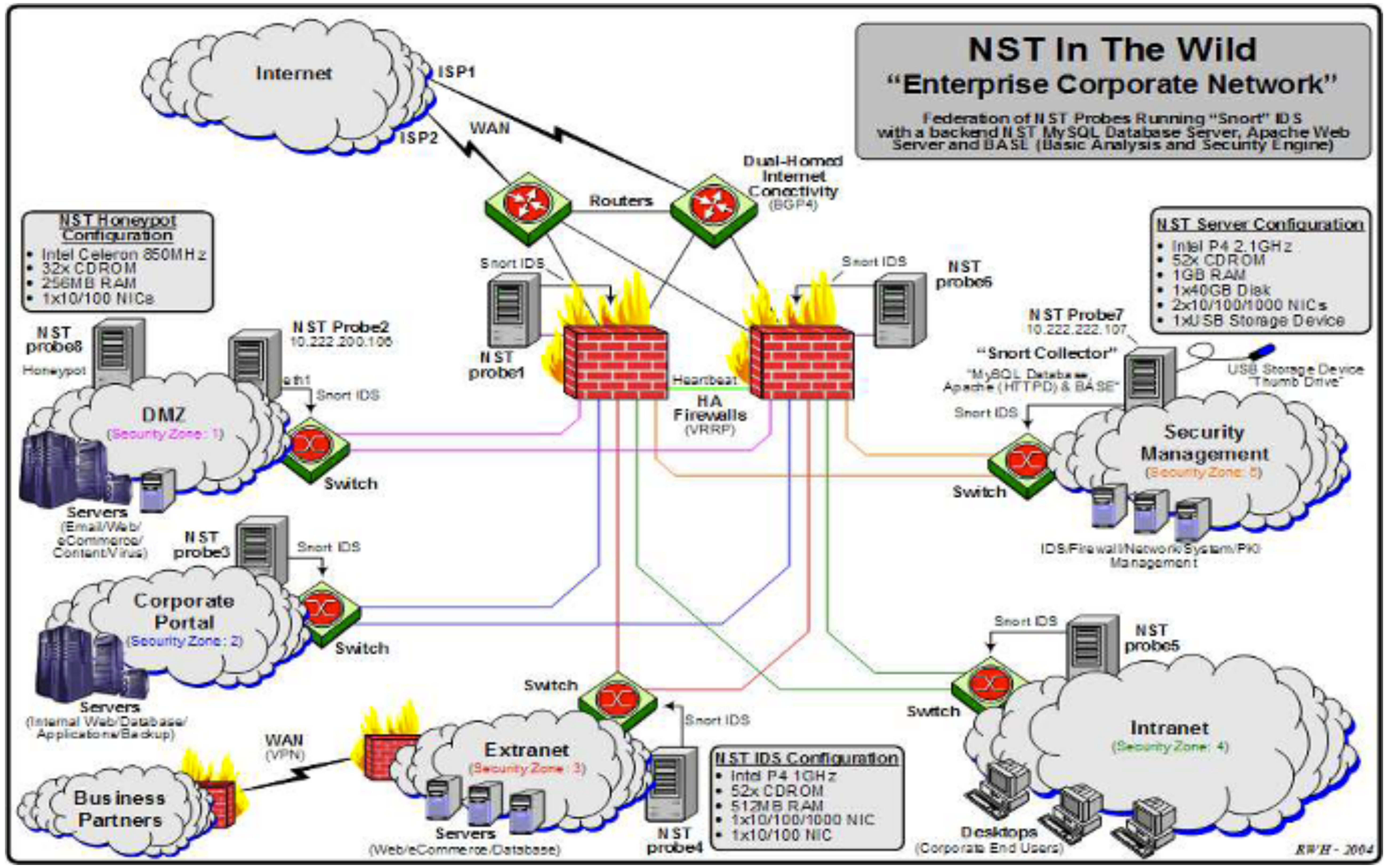


Example #2: Avoid Transient Anomalies

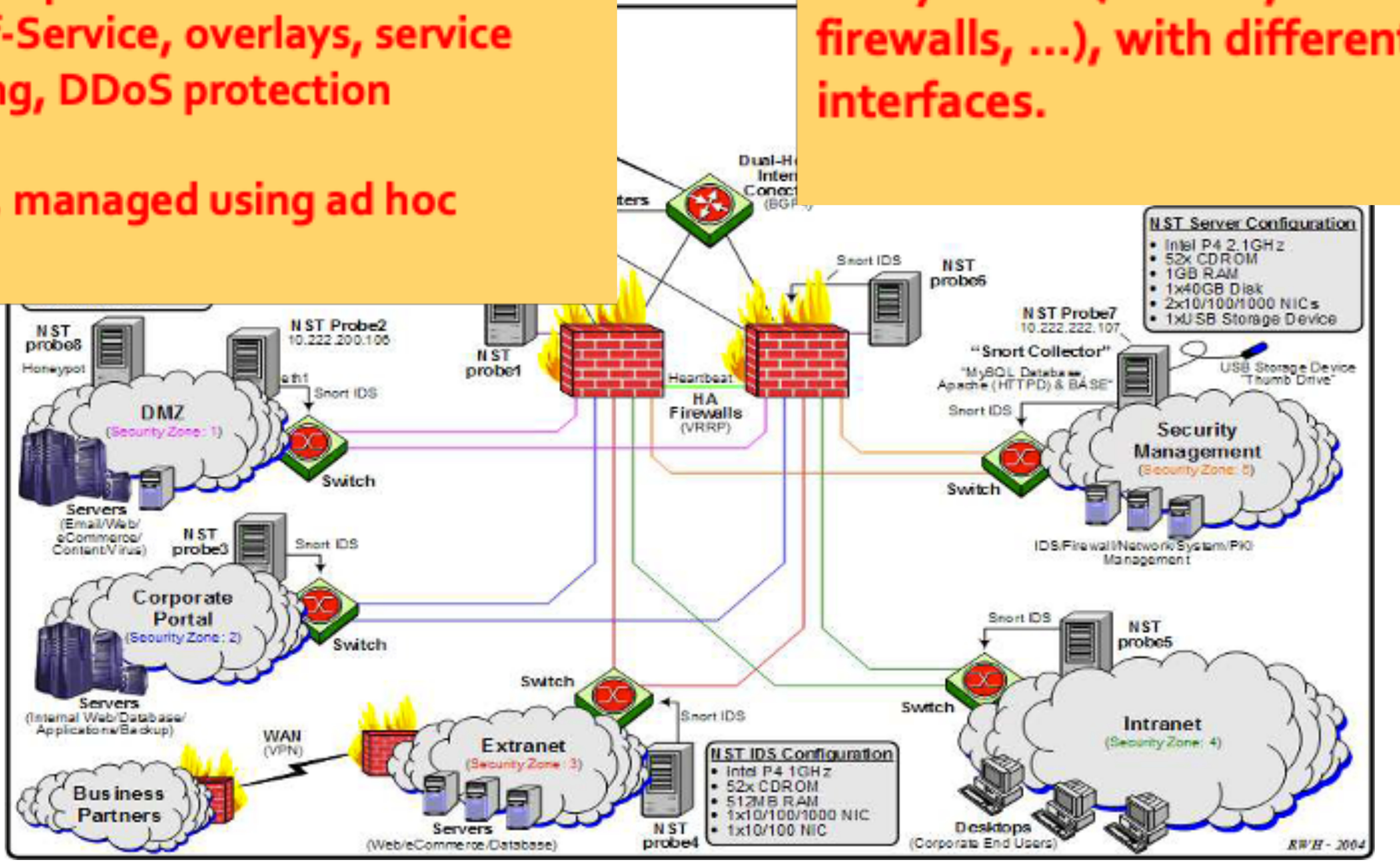
- Distributed protocol under race conditions
  - Temporary disagreement among the nodes
  - Cause packets stuck in the loop
  - Even though the changes were planned well



Messy Enterprise Networks


NST In The Wild
"Enterprise Corporate Network"
Federation of NST Probes Running "Snort" IDS with a backend NST MySQL Database Server, Apache Web Server and BASE (Basic Analysis and Security Engine)
Internet
ISP1
ISP2
WAN
Routers
Dual-Homed Internet Conectivity (BGP4)
NST Honeypot Configuration
• Intel Celeron 850MHz
• 32x CDROM
• 256MB RAM
• 1x10/100 NICs
NST Server Configuration
• Intel P4 2.1GHz
• 52x CDROM
• 1GB RAM
• 1x40GB Disk
• 2x10/100/1000 NICs
• 1xUSB Storage Device
Snort IDS
NST probe1
NST probe6
NST probe8
Honeypot
NST Probe2 10.222.200.106
eth1
NST Probe7 10.222.222.107
"Snort Collector"
"MySQL Database, Apache (HTTPD) & BASE"
USB Storage Device "Thumb Drive"
Heartbeat
HA Firewalls (VRRP)
DMZ (Security Zone: 1)
Switch
Servers (Email/Web/eCommerce/Content/Virus)
Security Management (Security Zone: 5)
IDS/Firewall/Network/System/PKI Management
NST probe3
Corporate Portal (Security Zone: 2)
Servers (Internal Web/Database/Applications/Backup)
NST probe5
Intranet (Security Zone: 4)
Extranet (Security Zone: 3)
WAN (VPN)
Business Partners
Servers (Web/eCommerce/Database)
NST probe4
NST IDS Configuration
• Intel P4 1GHz
• 52x CDROM
• 512MB RAM
• 1x10/100/1000 NIC
• 1x10/100 NIC
Desktops (Corporate End Users)
RWH - 2004

Messy Enterprise Networks



Dual-Homed Internet Connection (BGP)
NST probe8
Honeypot
NST Probe2 10.222.200.106
eth1
Snort IDS
NST probe1
Heartbeat
HA Firewalls (VRRP)
Snort IDS
NST probe6
NST Server Configuration
Intel P4 2.1GHz
52x CDROM
1GB RAM
1x40GB Disk
2x10/100/1000 NICs
1xUSB Storage Device
NST Probe7 10.222.222.107
"Snort Collector"
"MySQL Database Apache (HTTPD) & BASE"
USB Storage Device "Thumb Drive"
DMZ (Security Zone: 1)
Switch
Servers (Email/Web/eCommerce/Content/Virus)
Security Management (Security Zone: 5)
IDS/Firewall/Network/System/PKI Management
NST probe3
Corporate Portal (Security Zone: 2)
Servers (Internal Web/Database/Applications/Backup)
NST probe5
Intranet (Security Zone: 4)
Extranet (Security Zone: 3)
Servers (Web/eCommerce/Database)
WAN (VPN)
Business Partners
NST probe4
NST IDS Configuration
Intel P4 1GHz
52x CDROM
512MB RAM
1x10/100/1000 NIC
1x10/100 NIC
Desktops (Corporate End Users)
RWH - 2004

Other mgmt/control plane functions: access control, Quality-of-Service, overlays, service interposition, billing, DDoS protection

Non-routing state, managed using ad hoc mechanisms

Many boxes (routers, switches, firewalls, …), with different interfaces.

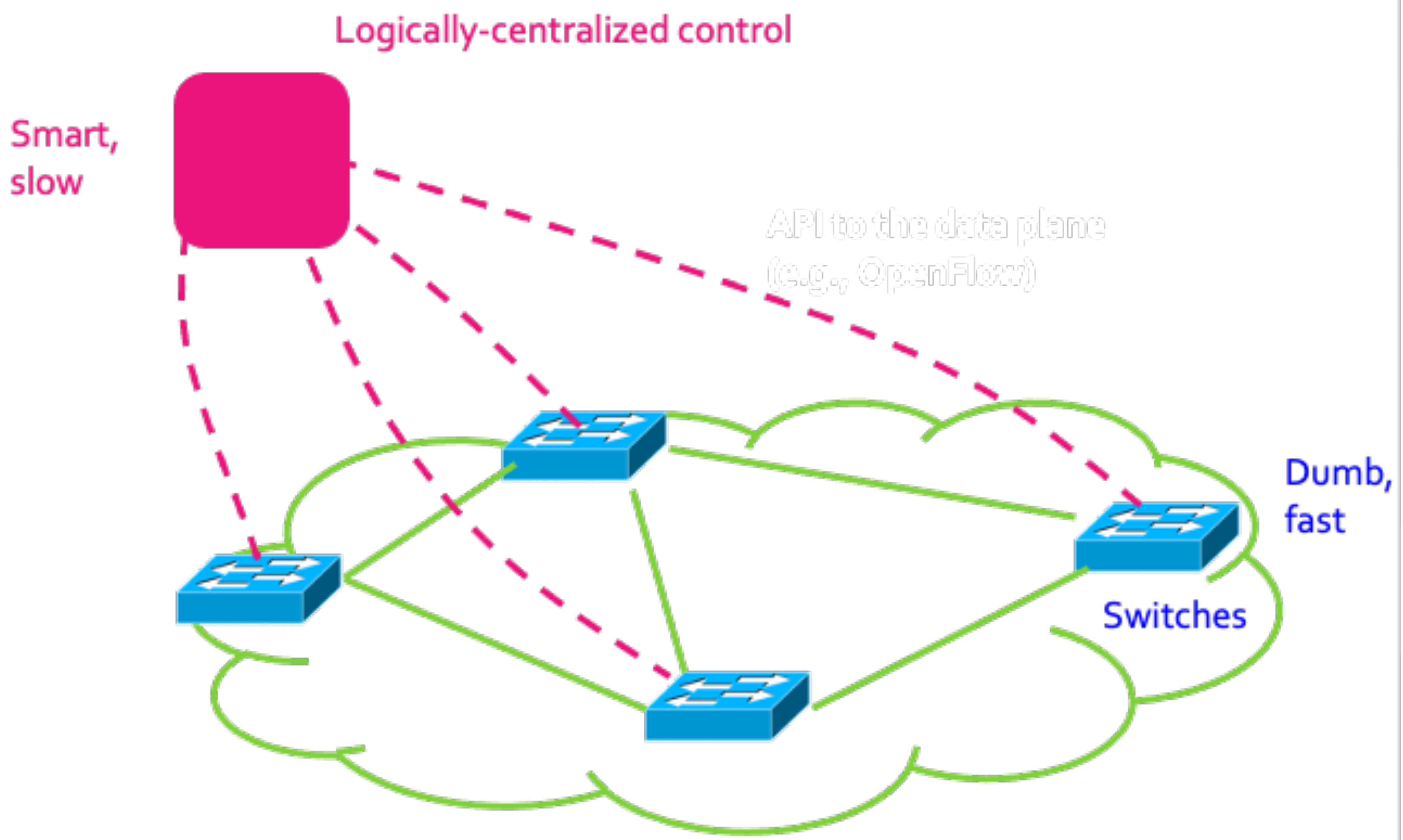
How can we simplify the network management and keep innovations?

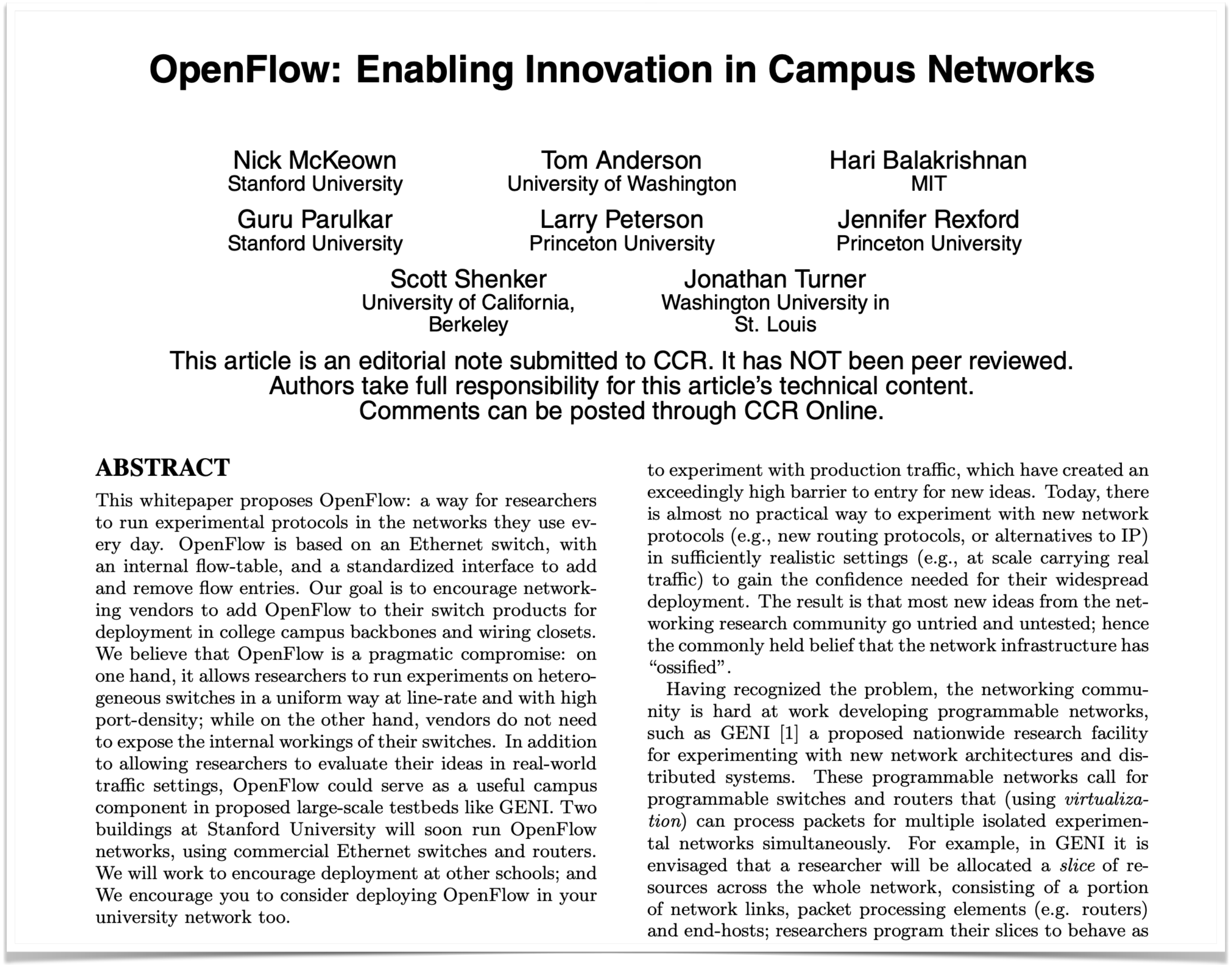
Why is this hard?

- #1: Closed equipment
  - Software tightly coupled with hardware
  - Vector-specific interfaces
- #2: Distributed control plane
  - Fast and reliable distributed algorithms are non-trivial to build
- #3: Ad-hoc device management
  - Great hardware heterogeneity
- #4: Slow protocol standardization
  - Many years of discussion between the communication and vendors

Solution: Software-Defined Networking

Software-Defined Networking (SDN)





OpenFlow: Enabling Innovation in Campus Networks

Nick McKeown
Stanford University

Tom Anderson
University of Washington

Hari Balakrishnan
MIT

Guru Parulkar
Stanford University

Larry Peterson
Princeton University

Jennifer Rexford
Princeton University

Scott Shenker
University of California, Berkeley

Jonathan Turner
Washington University in St. Louis

This article is an editorial note submitted to CCR. It has NOT been peer reviewed.
Authors take full responsibility for this article's technical content.
Comments can be posted through CCR Online.

ABSTRACT

This whitepaper proposes OpenFlow: a way for researchers to run experimental protocols in the networks they use every day. OpenFlow is based on an Ethernet switch, with an internal flow-table, and a standardized interface to add and remove flow entries. Our goal is to encourage networking vendors to add OpenFlow to their switch products for deployment in college campus backbones and wiring closets. We believe that OpenFlow is a pragmatic compromise: on one hand, it allows researchers to run experiments on heterogeneous switches in a uniform way at line-rate and with high port-density; while on the other hand, vendors do not need to expose the internal workings of their switches. In addition to allowing researchers to evaluate their ideas in real-world traffic settings, OpenFlow could serve as a useful campus component in proposed large-scale testbeds like GENI. Two buildings at Stanford University will soon run OpenFlow networks, using commercial Ethernet switches and routers. We will work to encourage deployment at other schools; and We encourage you to consider deploying OpenFlow in your university network too.

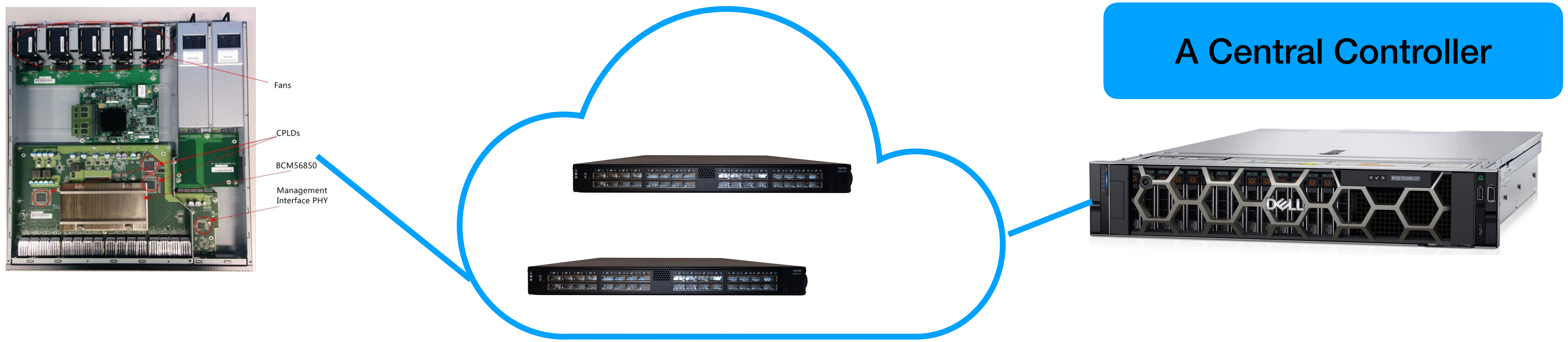
to experiment with production traffic, which have created an exceedingly high barrier to entry for new ideas. Today, there is almost no practical way to experiment with new network protocols (e.g., new routing protocols, or alternatives to IP) in sufficiently realistic settings (e.g., at scale carrying real traffic) to gain the confidence needed for their widespread deployment. The result is that most new ideas from the networking research community go untried and untested; hence the commonly held belief that the network infrastructure has "ossified".

Having recognized the problem, the networking community is hard at work developing programmable networks, such as GENI [1] a proposed nationwide research facility for experimenting with new network architectures and distributed systems. These programmable networks call for programmable switches and routers that (using *virtualization*) can process packets for multiple isolated experimental networks simultaneously. For example, in GENI it is envisaged that a researcher will be allocated a *slice* of resources across the whole network, consisting of a portion of network links, packet processing elements (e.g. routers) and end-hosts; researchers program their slices to behave as

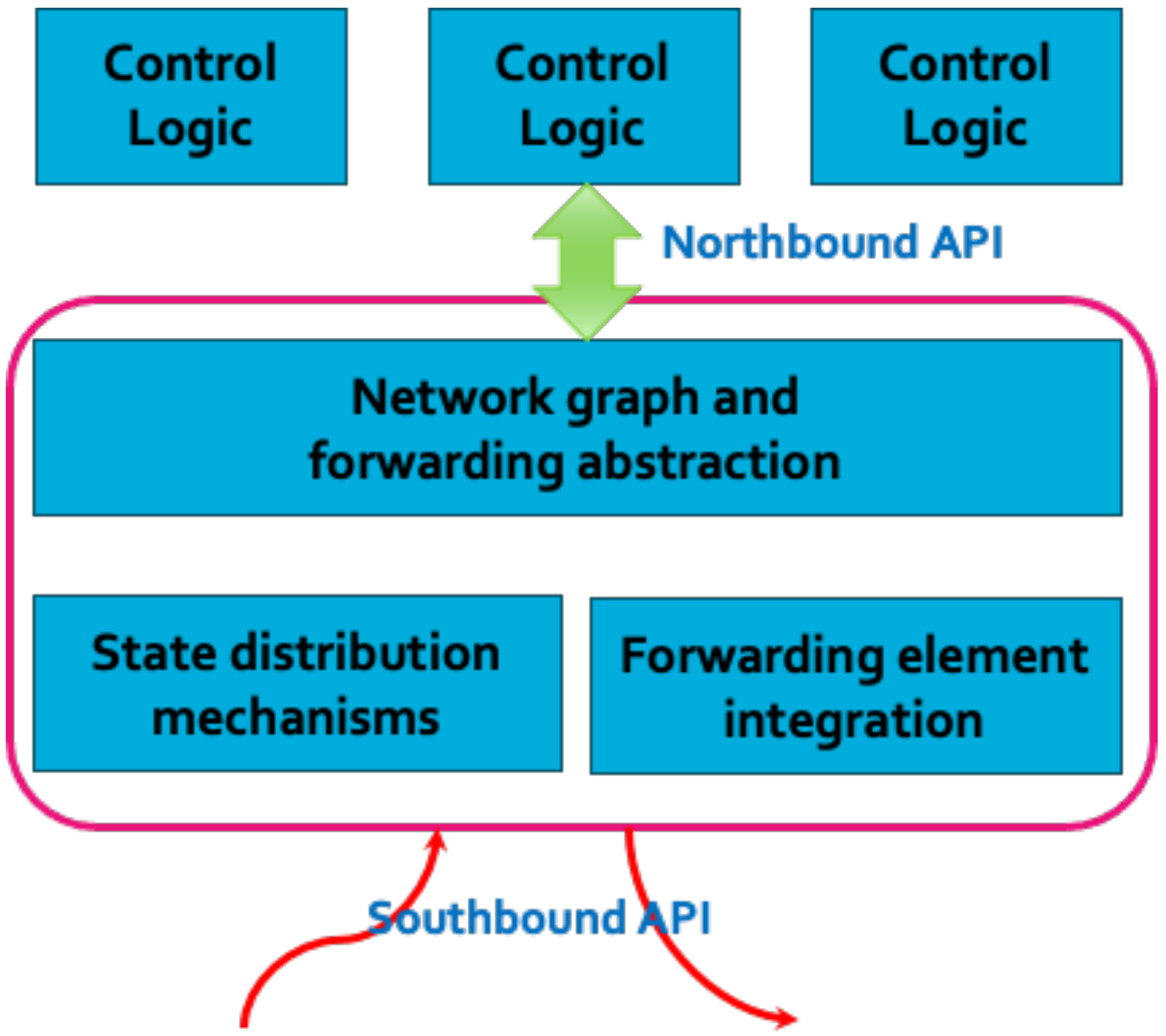
SDN Hardware Architecture

- #1: Central controller
  - General-purpose servers
  - Provide connectivity to all the routers in the managed domain

- #2: Router/Switch
  - Run a simple forwarding data-plane

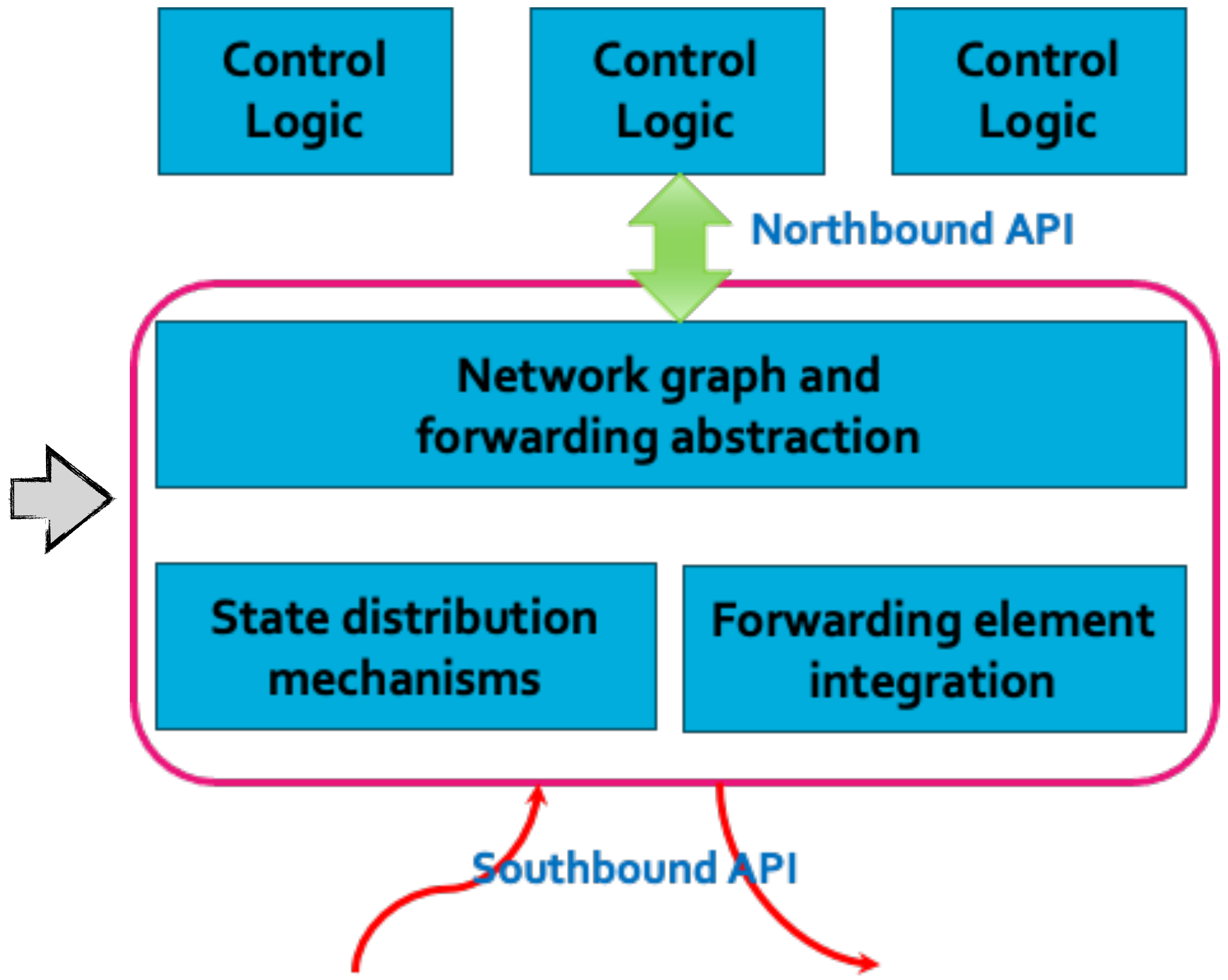


SDN Software Stack

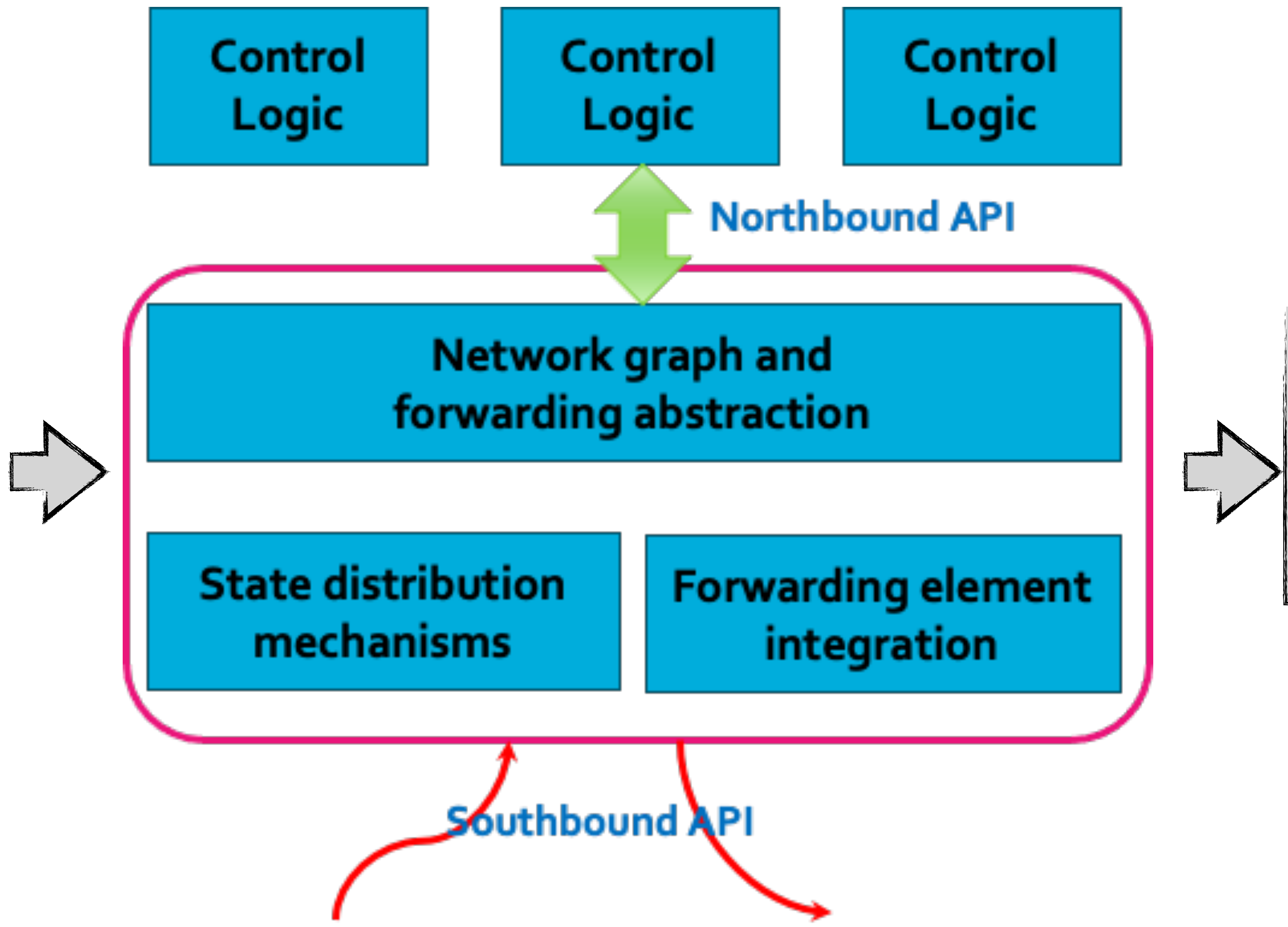

Control Logic
Control Logic
Control Logic
Northbound API
Network graph and forwarding abstraction
State distribution mechanisms
Forwarding element integration
Southbound API

SDN Software Stack

<u>Events from switches</u>
Topology changes,
Traffic statistics,
Arriving packets


Control Logic
Control Logic
Control Logic
Northbound API
Network graph and forwarding abstraction
State distribution mechanisms
Forwarding element integration
Southbound API

SDN Software Stack


Control Logic
Control Logic
Control Logic
Northbound API
Network graph and forwarding abstraction
State distribution mechanisms
Forwarding element integration
Southbound API

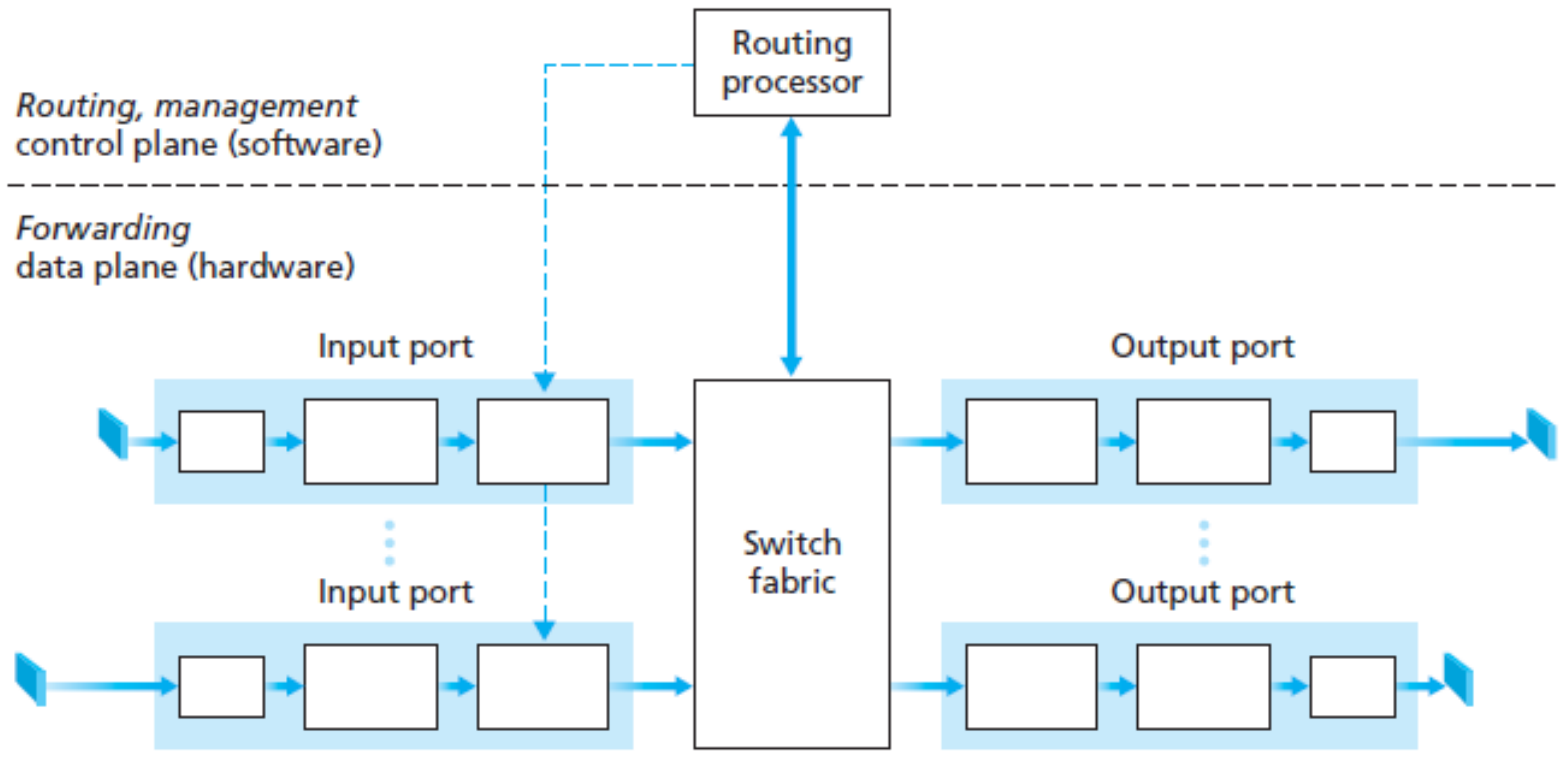
<u>Events from switches</u>
Topology changes, Traffic statistics, Arriving packets

<u>Commands to switches</u>
(un)install rules, Query statistics, Send packets

How does SDN work exactly?

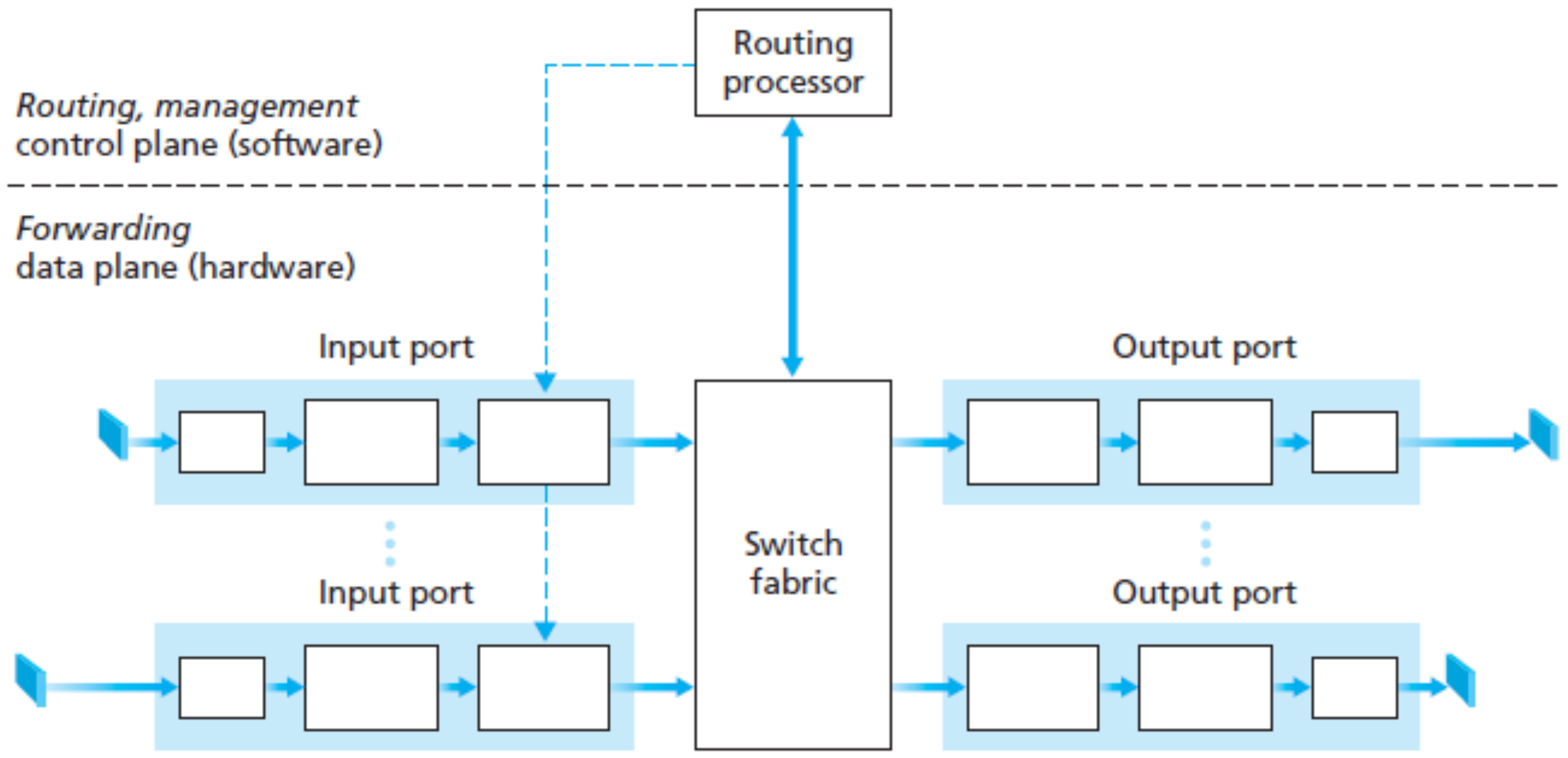
Packet Forwarding w/o SDN

- Packets are forwarded based on the routing table
  - The Routing processor runs the routing algorithm and constructs the table



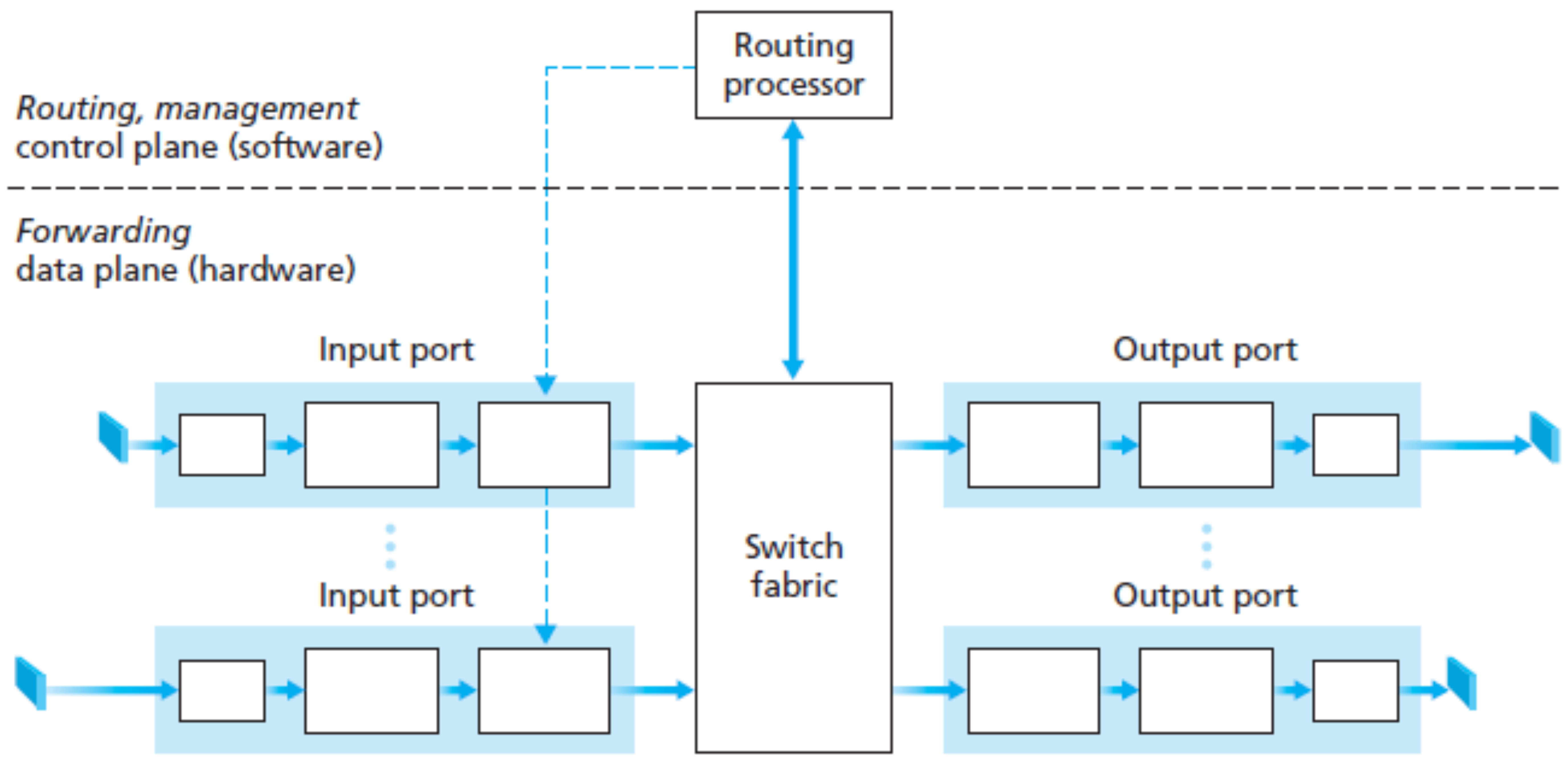
Packet Forwarding w/ SDN

- Has the data-plane changed?



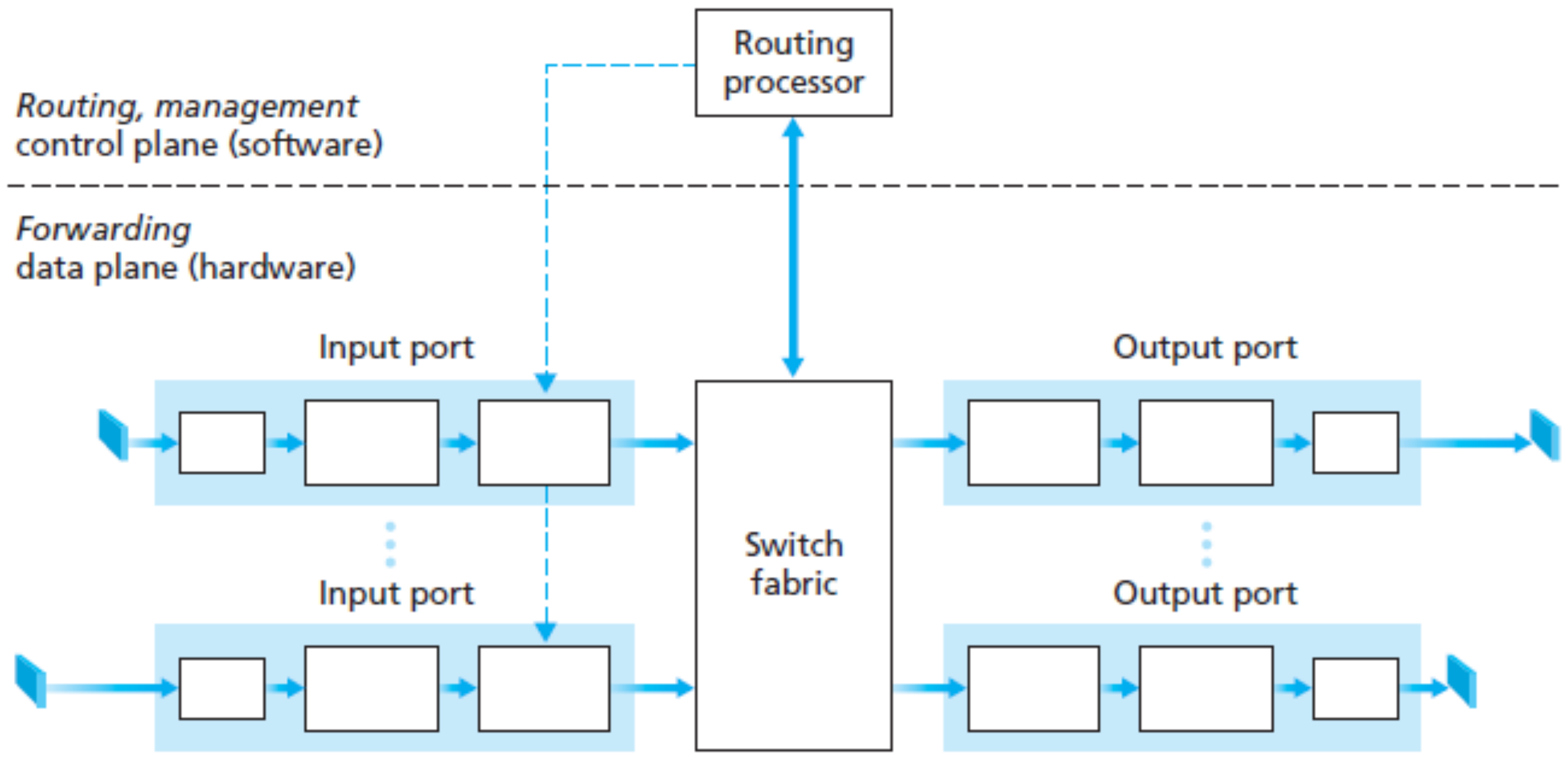
Packet Forwarding w/ SDN

- Has the data-plane changed?
  - No



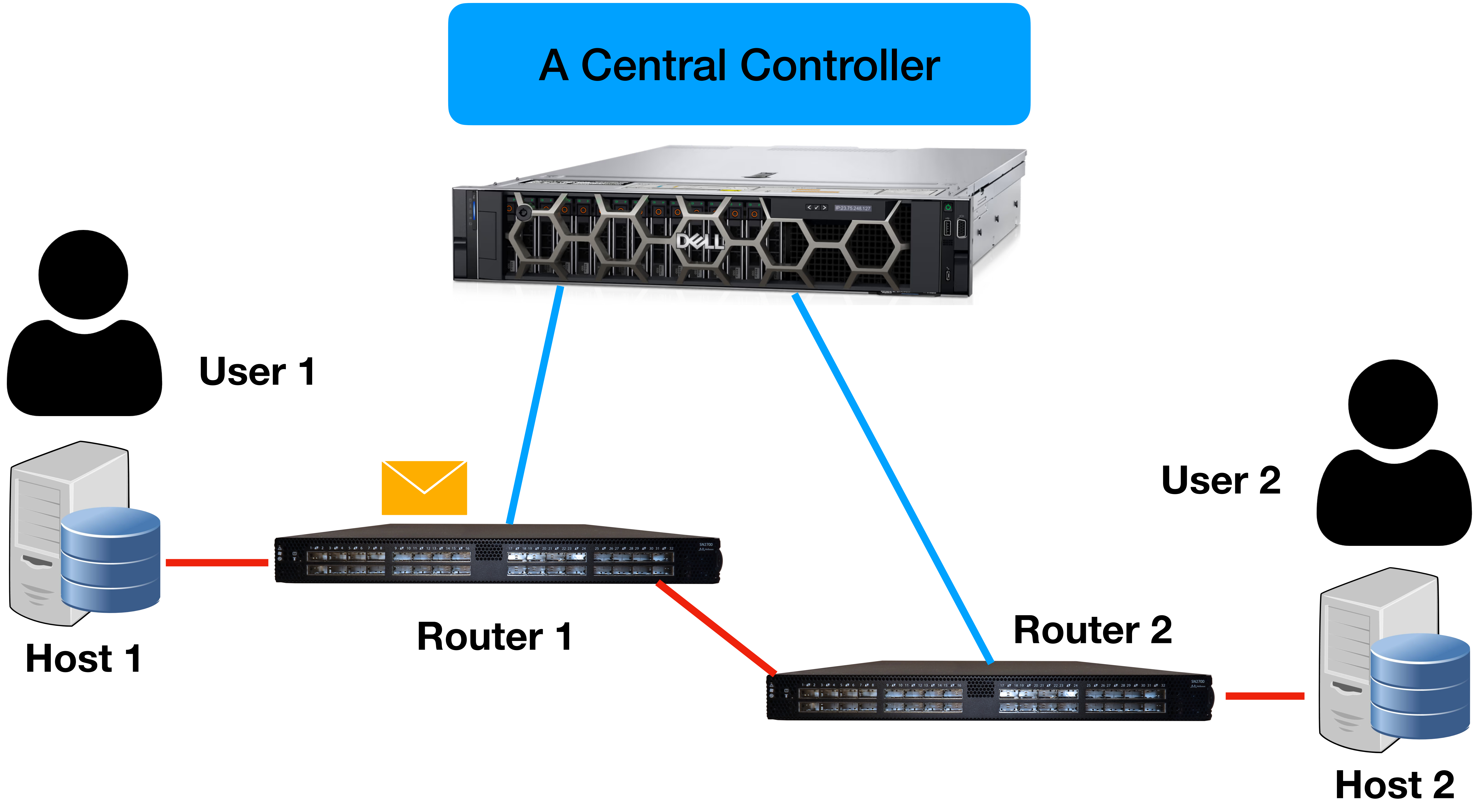
Packet Forwarding w/ SDN

- Has the data-plane changed?
  - No
- When missing in the routing table
  - Packets are forwarded to the central controller!



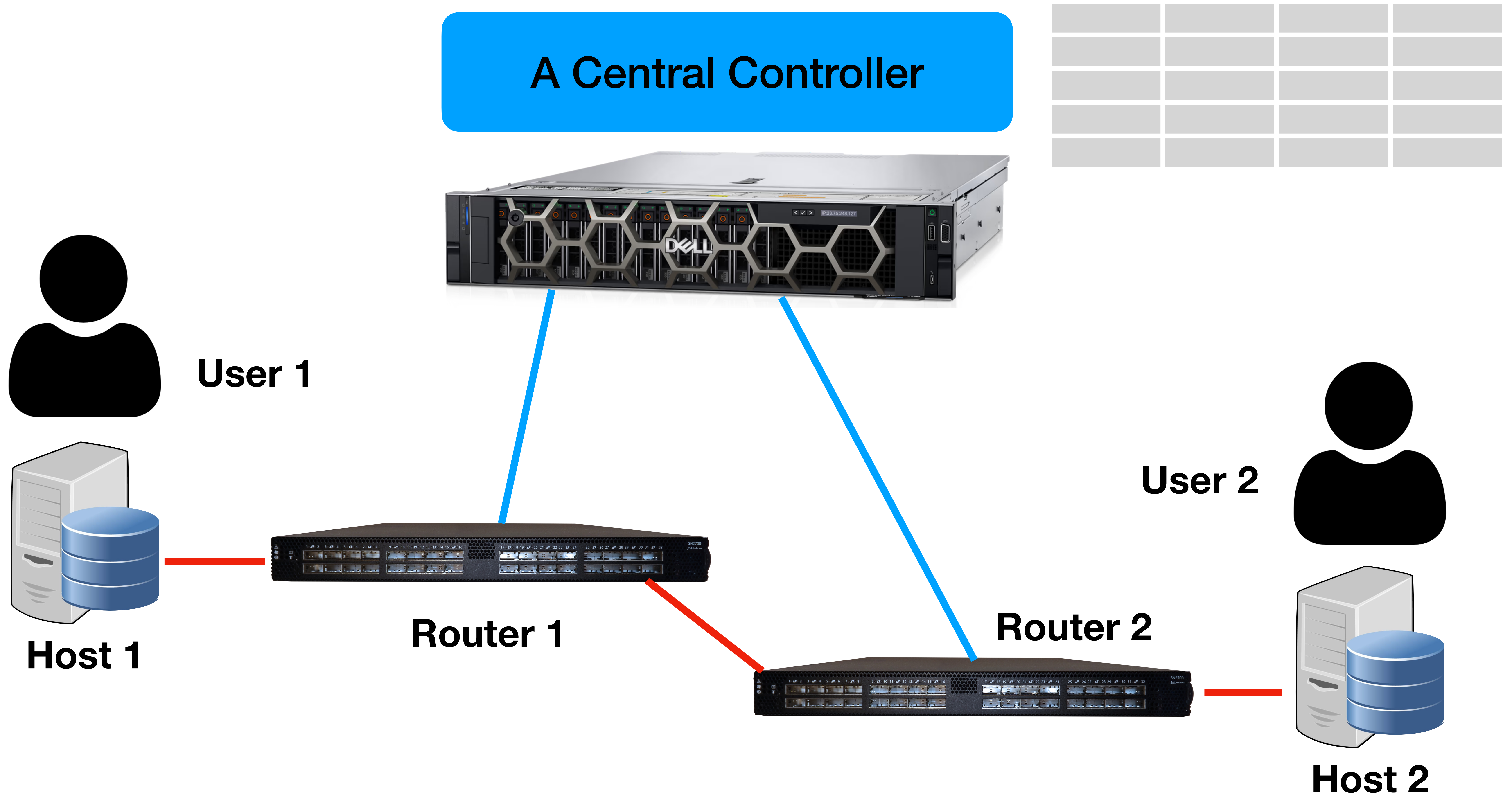
The Controller Capability

- #1: Global view
- #2: Programmability



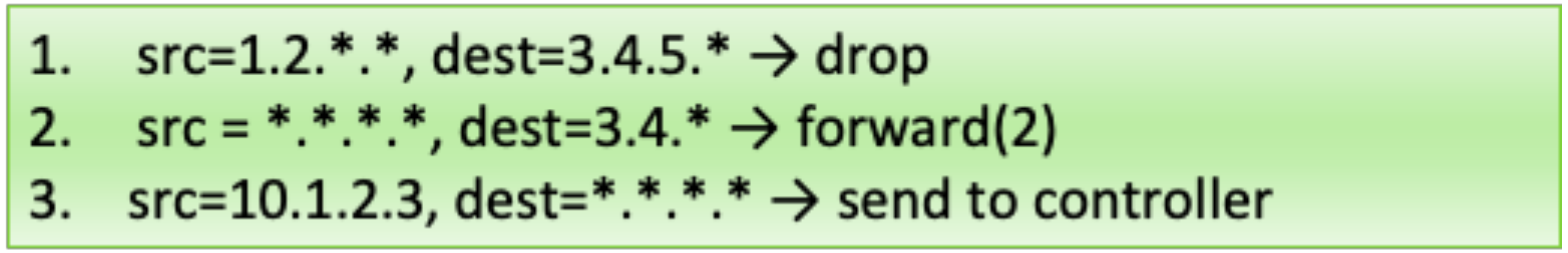
The Controller Capability

- #1: Global view
- #2: Programmability



Data-Plane Language System

- Packet-handling rules
  - Pattern: match packet header bits
  - Actions: drop, forward, modify, send to the controller
  - Priority: disambiguate overlapping patterns
  - Counters: #bytes and #packets



1. src=1.2.*.*, dest=3.4.5.* → drop
2. src = *.*.*.*, dest=3.4.* → forward(2)
3. src=10.1.2.3, dest=*.*.*.* → send to controller

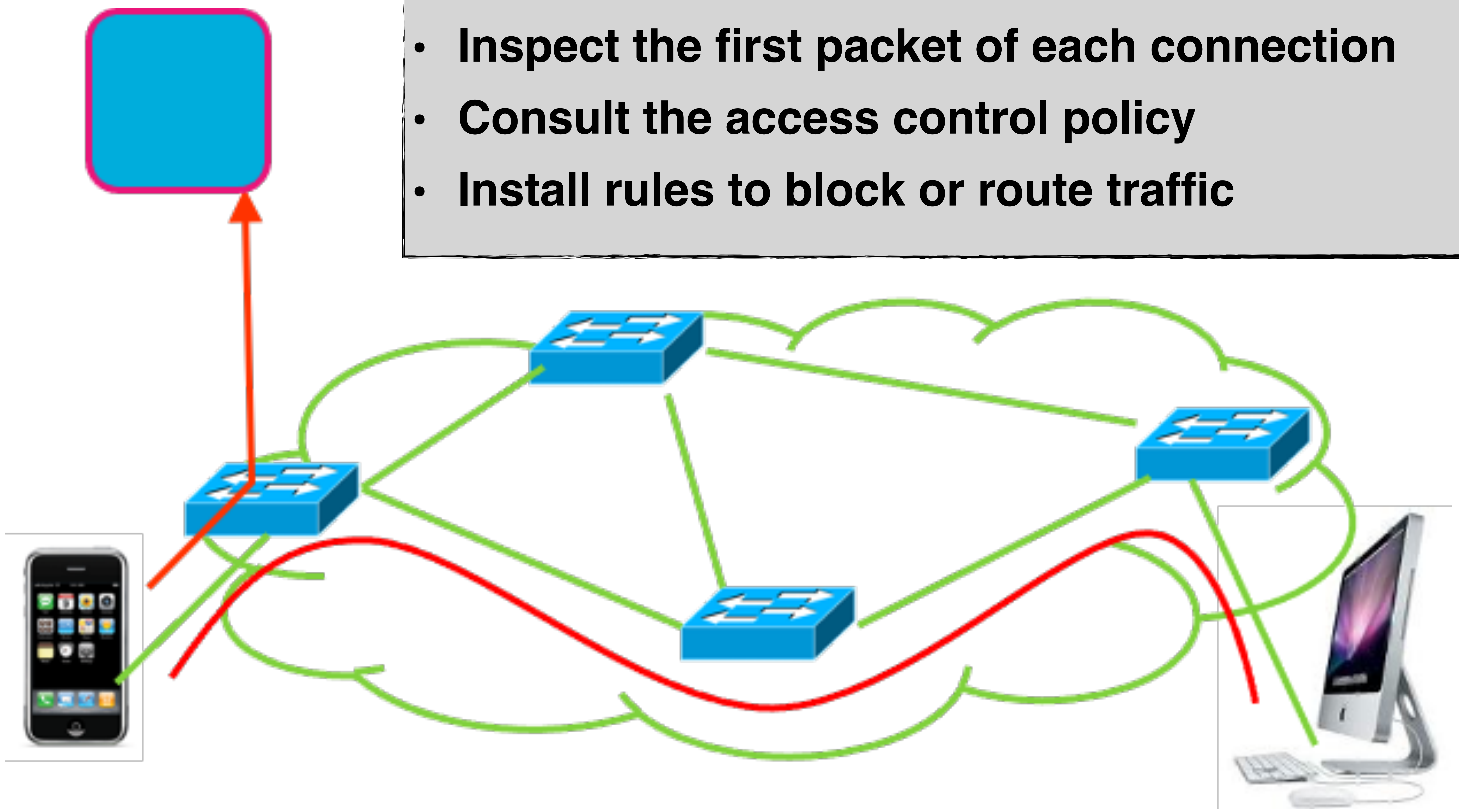
SDN Application Example

- Use cases
  - Dynamic access control
  - VM mobility/migration
  - Network virtualization
  - Load balancing
  - Traffic Engineering

- Commercial products
  - Network virtualization: Nicira/VMware/Broadcom, Azure, Google, etc.
  - Traffic engineering: Google’s B4, Microsoft’s SWAN, etc.

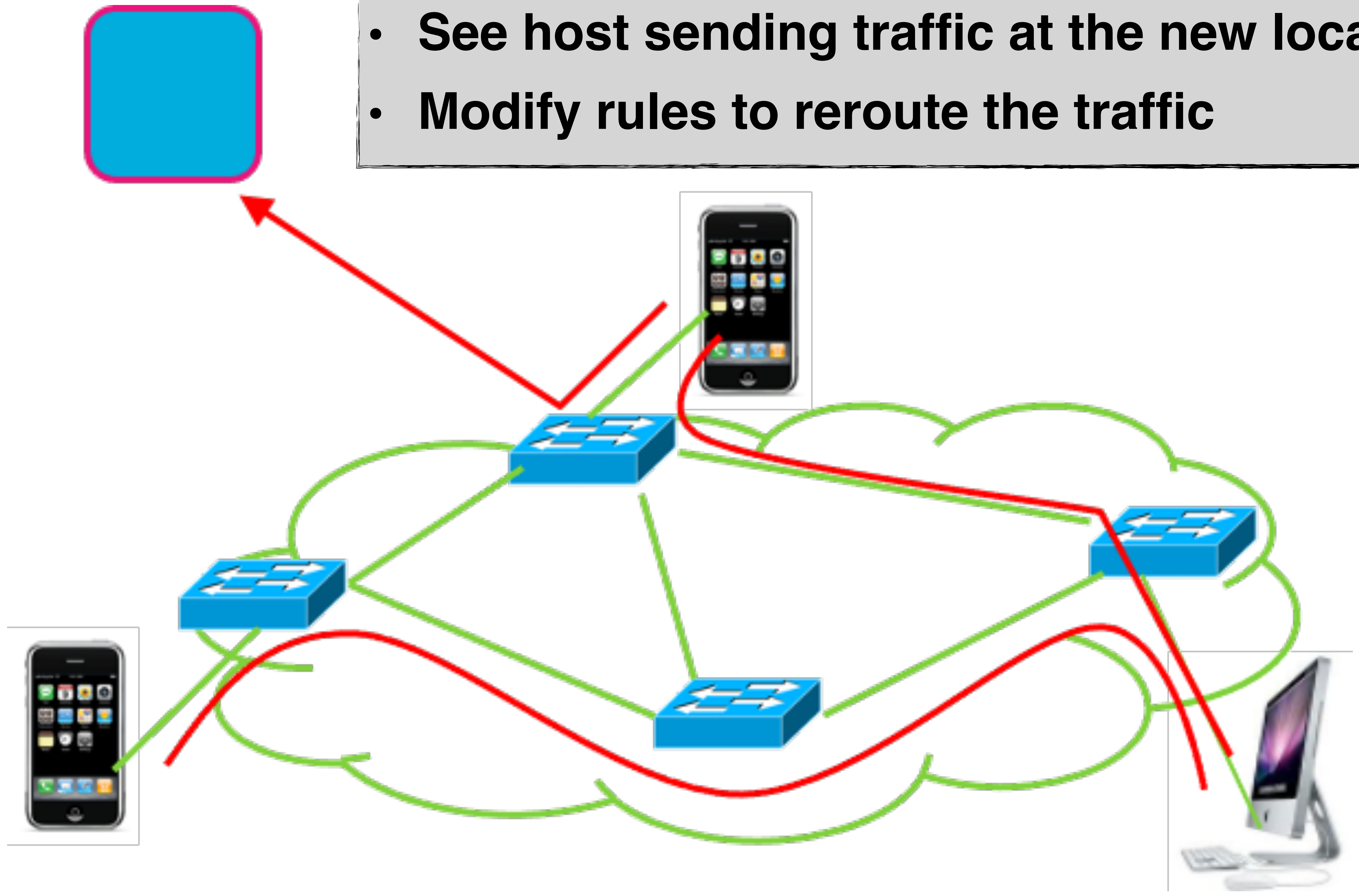
Example #1: Dynamic Access Control

- **Inspect the first packet of each connection**
- **Consult the access control policy**
- **Install rules to block or route traffic**



Example #2: Seamless Mobility/Migration

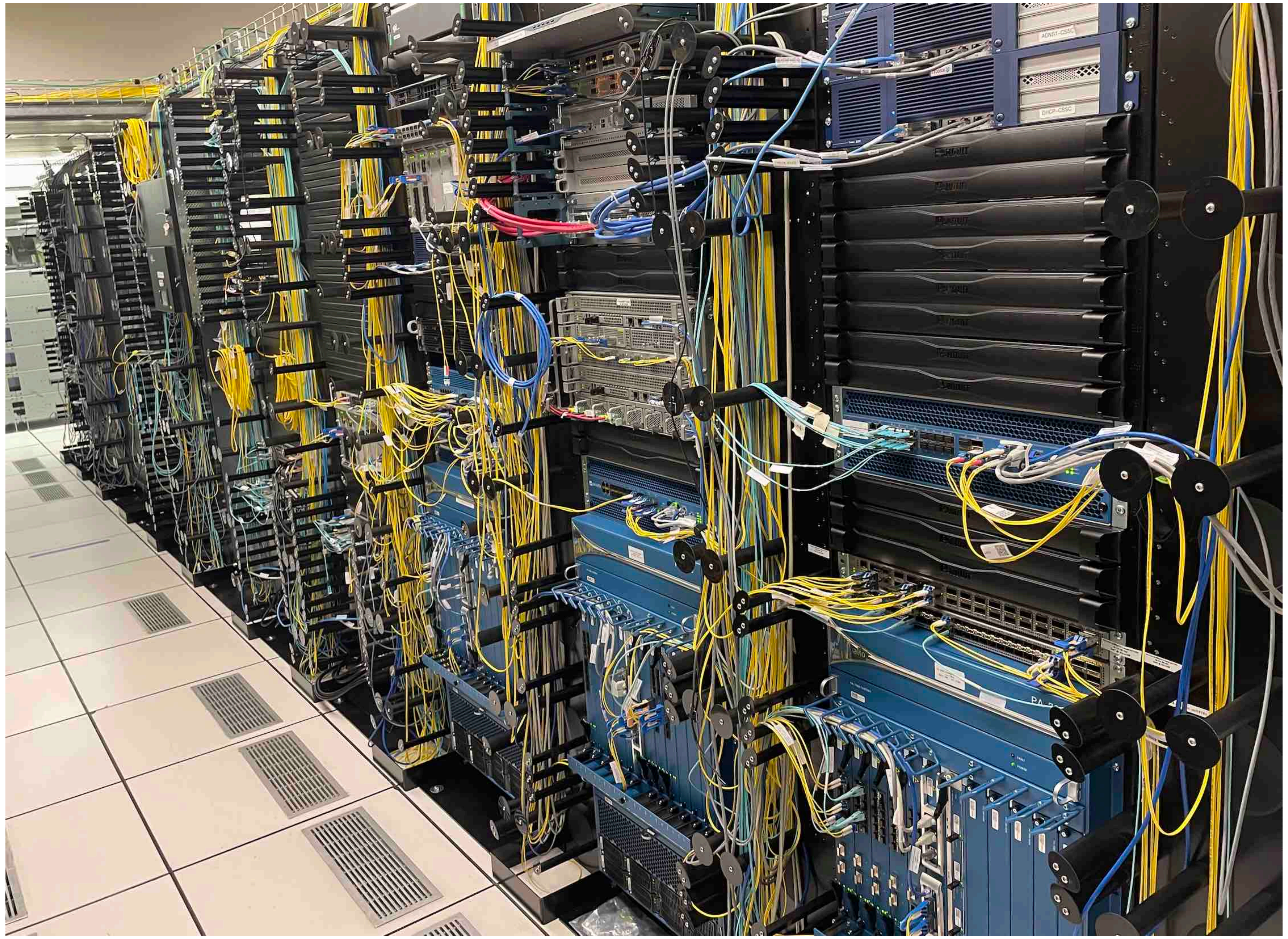
- **See host sending traffic at the new location**
- **Modify rules to reroute the traffic**



SDN/OpenFlow in the Wild

- Open Networking Foundation
  - Create software-defined networking standards
- Commercial OpenFlow Switches
  - Cisco, HP, NEC, Quanta, Dell, IBM, Juniper,…
- Controllers/Languages
  - NOX, Beacon, Floodlight, Nettle, NOIX, POX
  - Frenetic, MAPLE, Aspera, Pyretic
- Network deployments
  - Campus networks + commercial deployments

SDN@UW-Madison Campus Backbone



The efficacy of SDN highly depends on how effective the underlying distributed systems are!

Summary

- Today
  - Software-Defined Networking
- Next lecture
  - Inter-domain Routing