Introduction to Software Defined Networking

Sofiane Zemouri
John Murphy

Outline

- Introduction & Traditional Networking
- What is SDN?
- OpenFlow basics
- SDN Challenges

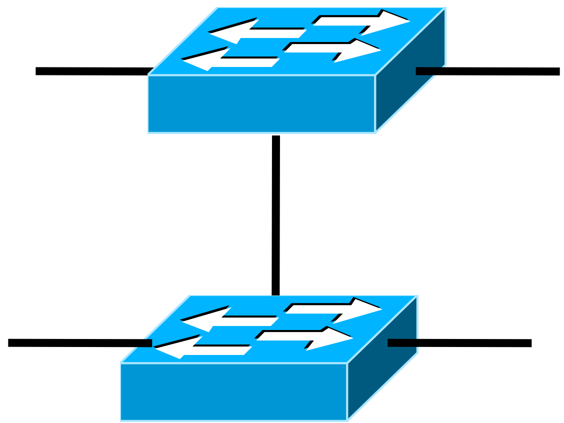
INTRODUCTION

The Internet: A Remarkable Story

- Tremendous success
  - From research experiment to global infrastructure
- Brilliance of under-specifying
  - Network: best-effort packet delivery
  - Hosts: arbitrary applications
- Enables innovation in applications
  - Web, P2P, VoIP, social networks, virtual worlds
- But, change is easy only at the edge…

Inside the ‘Net: A Different Story…

- Closed equipment
  - Software bundled with hardware
  - Vendor-specific interfaces
- Over specified
  - Slow protocol standardization
- Few people can innovate
  - Equipment vendors write the code
  - Long delays to introduce new features

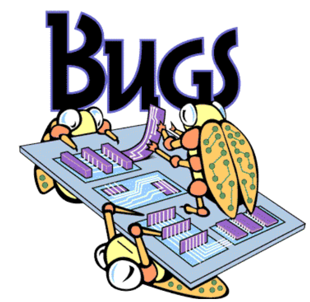


Impacts performance, security, reliability, cost…

Networks are Hard to Manage

- Operating a network is expensive
  - More than half the cost of a network
  - Yet, operator error causes most outages

- Buggy software in the equipment
  - Routers with 20+ million lines of code
  - Cascading failures, vulnerabilities, etc.



- The network is “in the way”
  - Especially a problem in data centers
  - … and home networks

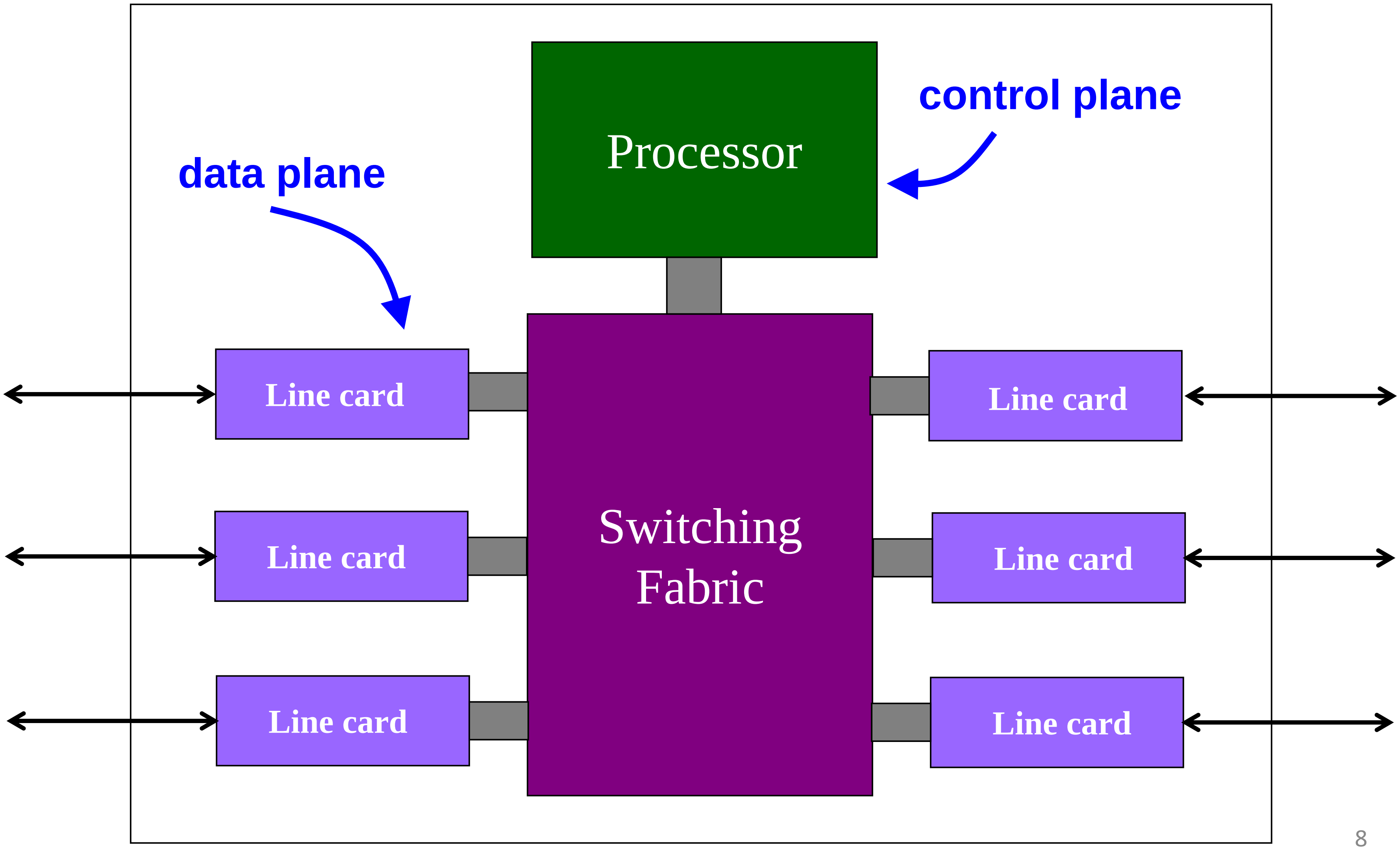
The Networking "Planes"

- **Data plane**: processing and delivery of packets with local forwarding state
  - Forwarding state + packet header → forwarding decision
  - Filtering, buffering, scheduling

- **Control plane**: computing the forwarding state in routers
  - Determines how and where packets are forwarded
  - Routing, traffic engineering, failure detection/recovery, …

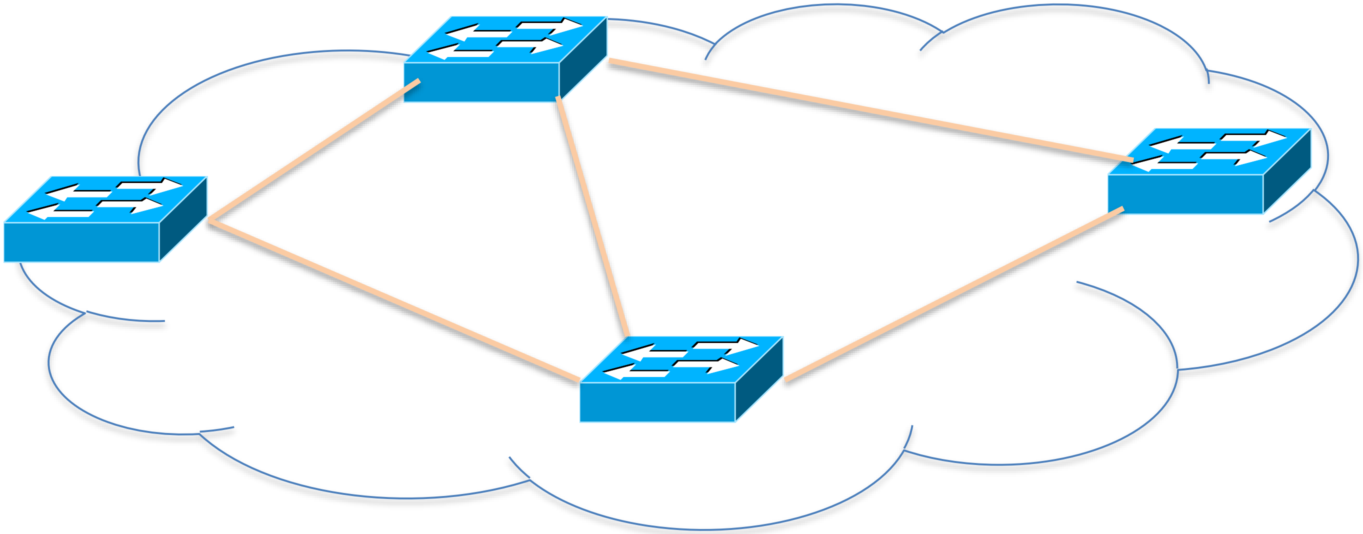
- **Management plane**: configuring and tuning the network
  - Traffic engineering, ACL config, device provisioning, …

Data and Control Planes



TRADITIONAL COMPUTER NETWORKS

Traditional Computer Networks

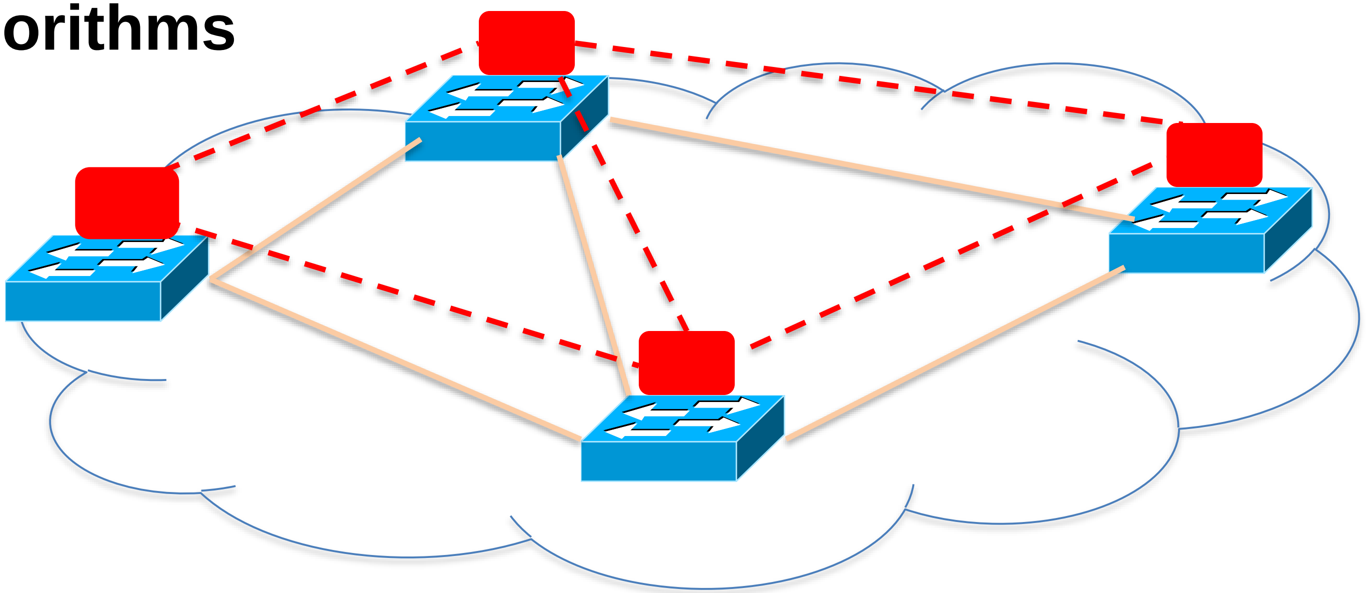


Data plane: **Packet streaming**

Forward, filter, buffer, mark, rate-limit, and measure packets

Traditional Computer Networks

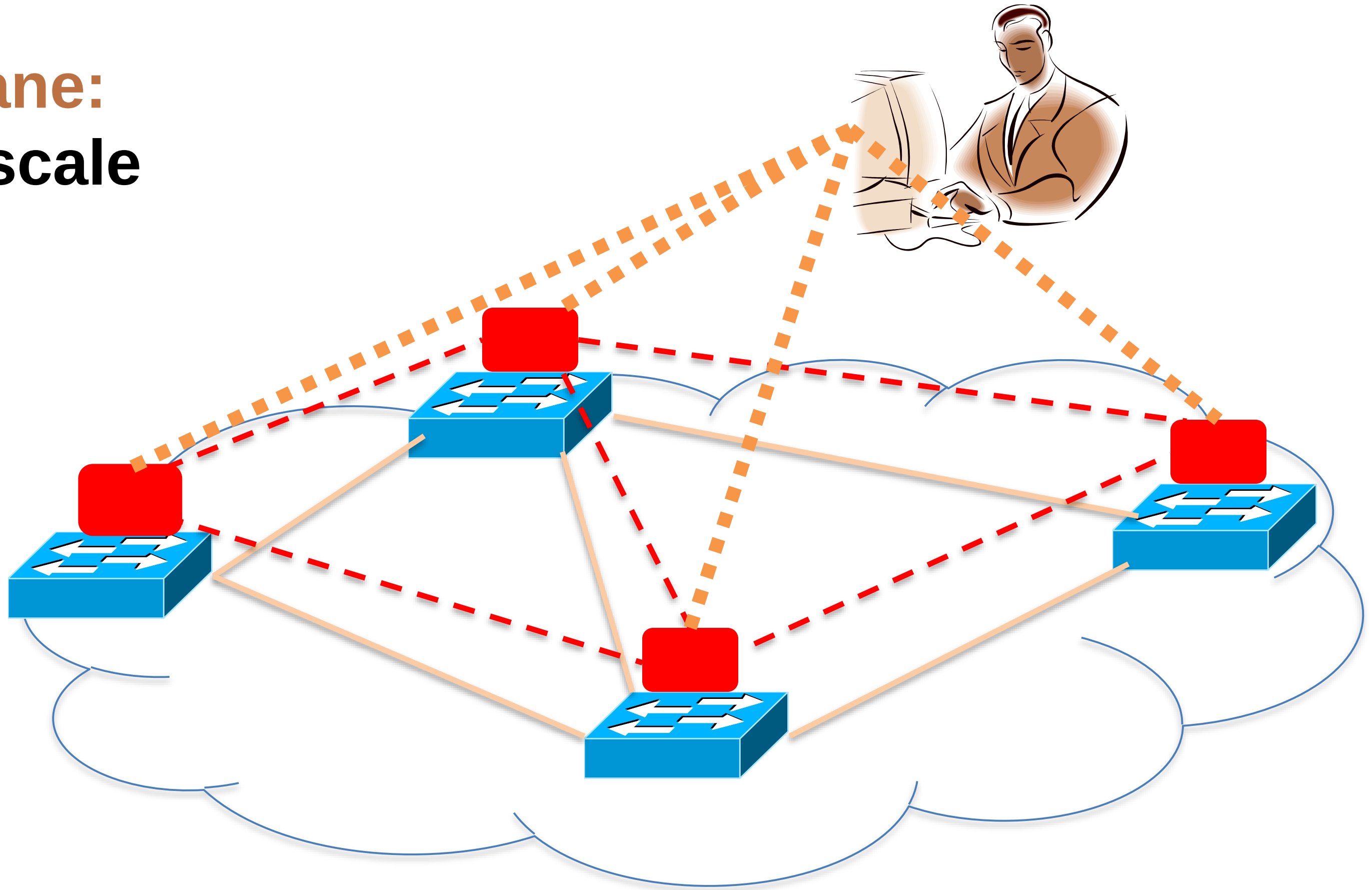
Control plane:
Distributed algorithms



Track topology changes, compute routes, install forwarding rules

Traditional Computer Networks

Management plane:
Human time scale



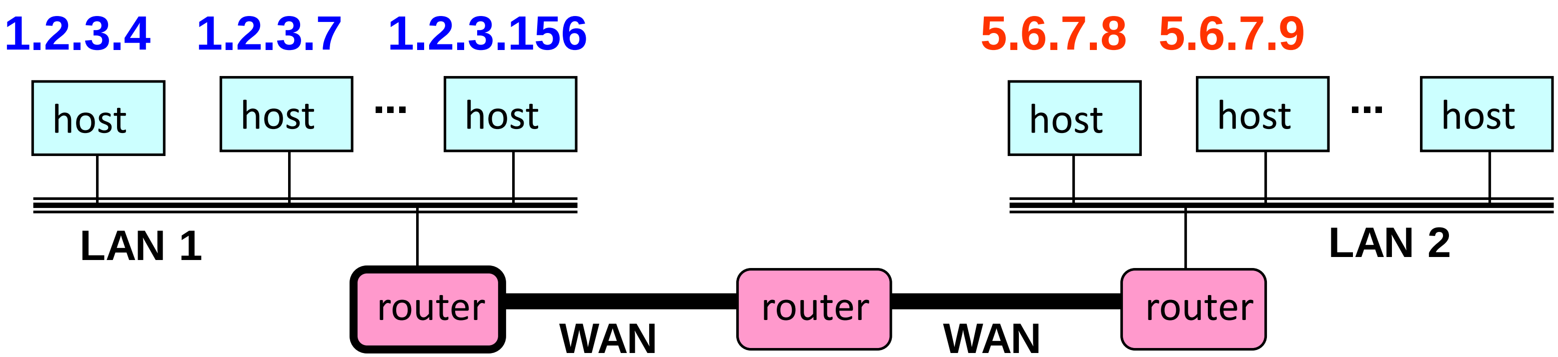
Collect measurements and configure the equipment

Timescales

	Data	Control	Management
Time-scale	Packet (nsec)	Event (10 msec to sec)	Human (min to hours)
Location	Linecard hardware	Router software	Humans or scripts

Data Plane

- Streaming algorithms on packets
  - Matching on some header bits
  - Perform some actions
- Example: IP Forwarding



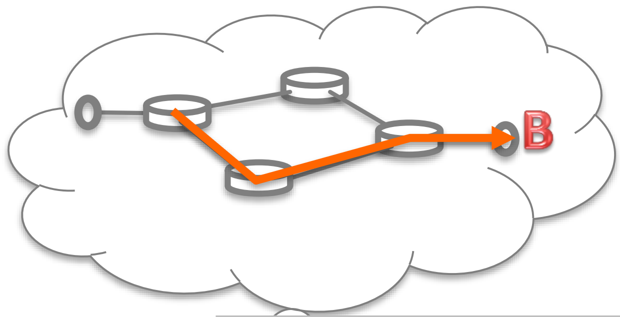
1.2.3.0/24	←
5.6.7.0/24	→

forwarding table

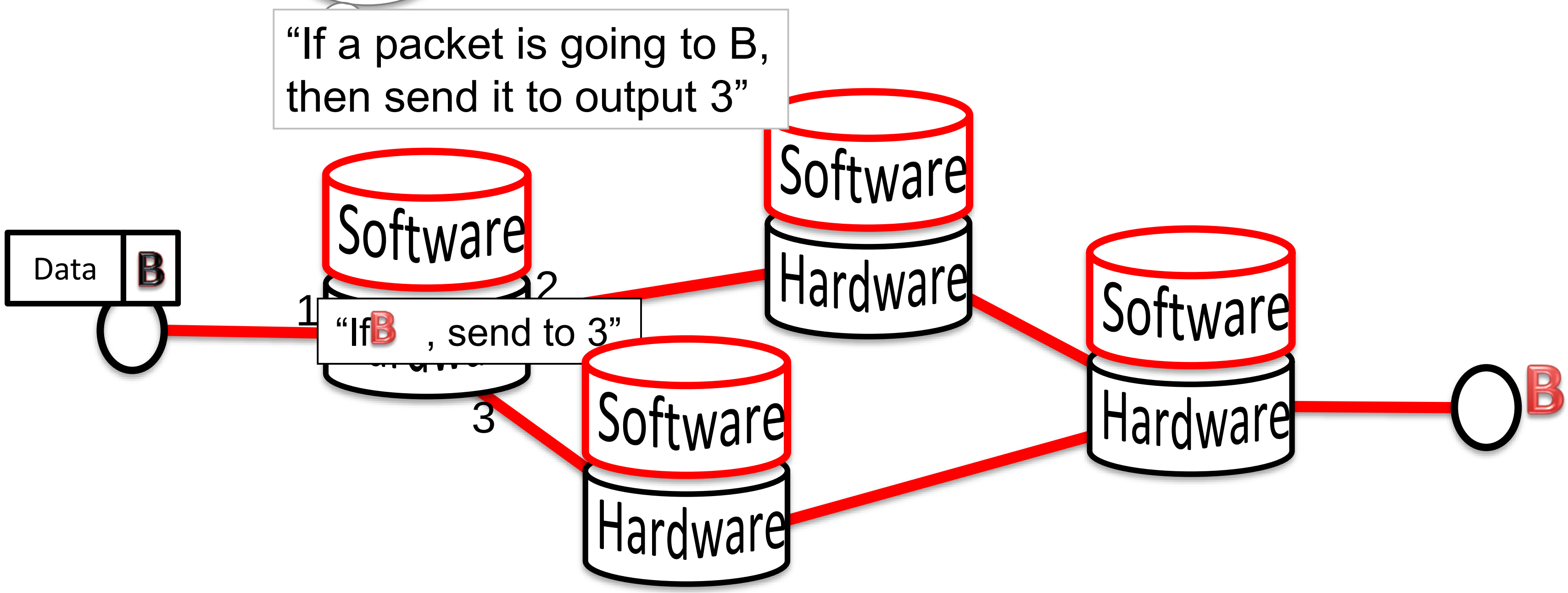
Control Plane

- Compute paths the packets will follow
  - Populate forwarding tables
  - Traditionally, a distributed protocol

- Example: Link-state routing (OSPF, IS-IS)
  - Flood the entire topology to all nodes
  - Each node computes shortest paths
  - Dijkstra’s algorithm

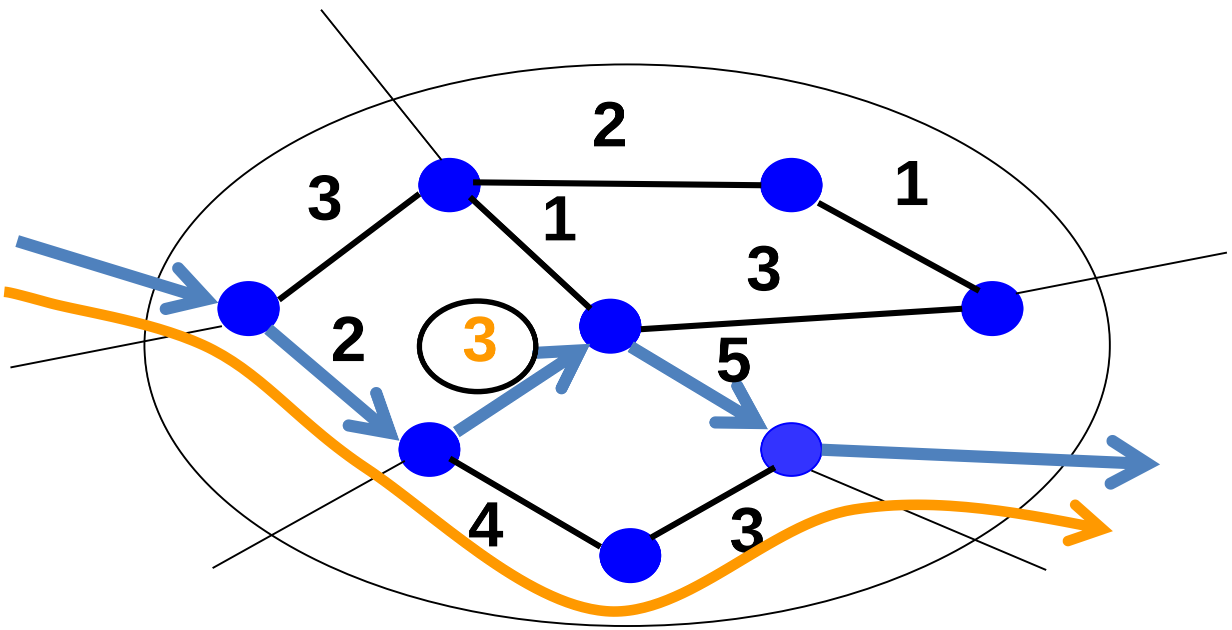


1. Figure out which routers and links are present.
2. Run Dijkstra's algorithm to find shortest paths.



Management Plane

- Traffic Engineering: setting the weights
  - Inversely proportional to link capacity?
  - Proportional to propagation delay?
  - Network-wide optimization based on traffic?



Traditional Networking Issues

(Too) many task-specific control mechanisms

- No modularity, limited functionality

Indirect control

- Must invert protocol behavior, “coax” it to do what you want
- Ex. Changing weights instead of paths for TE

Uncoordinated control

- Cannot control which router updates first

Interacting protocols and mechanisms

- Routing, addressing, access control, QoS

Traditional Networking Issues

The network is

- Hard to reason about
- Hard to evolve
- Expensive

WHAT IS SDN?

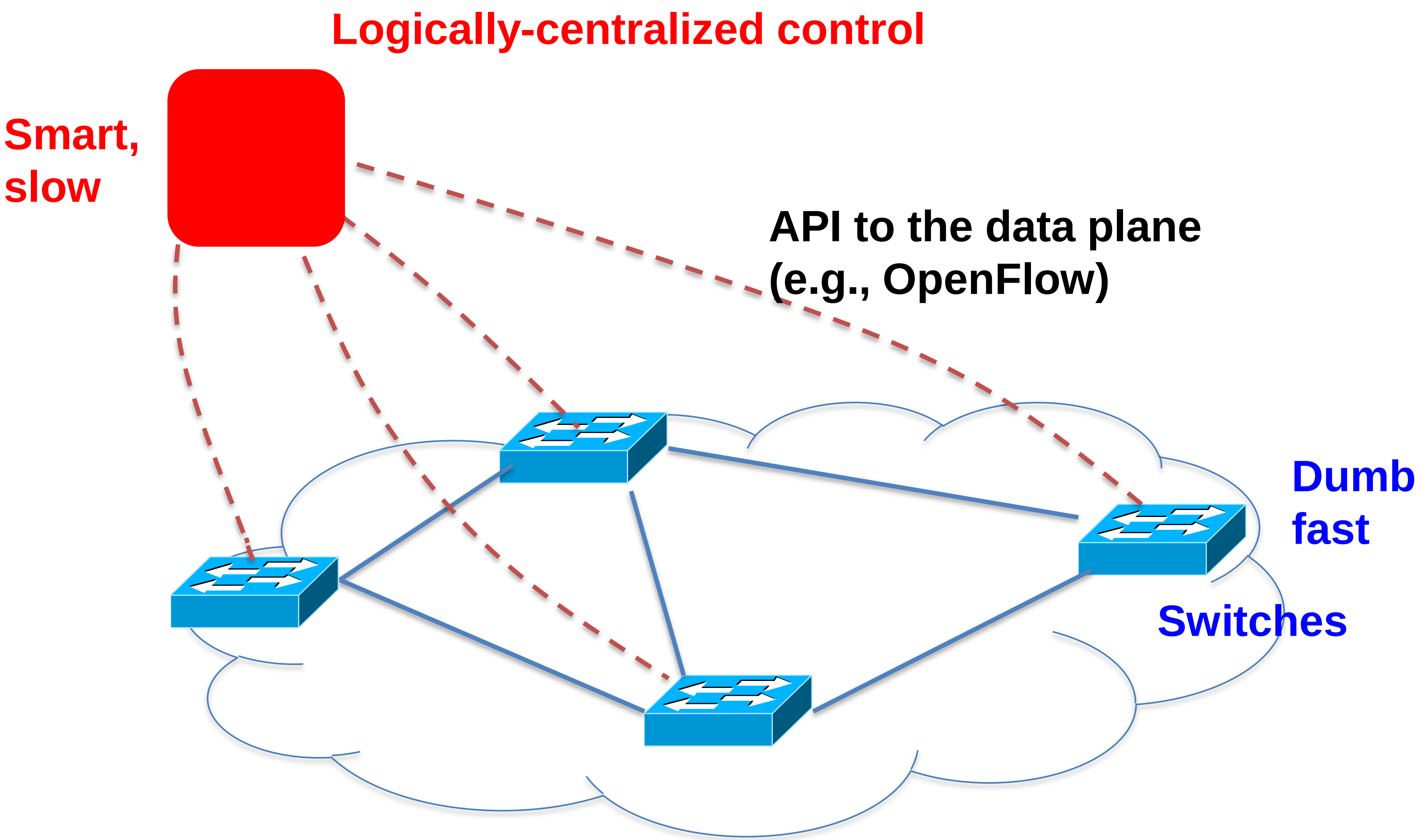
Software Defined Networks

A network in which the control plane is physically separate from the data plane.

and

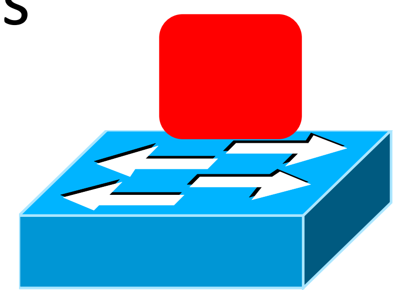
A single (logically centralized) entity controls several forwarding devices.

Software Defined Networks



Centralization of the Control Plane

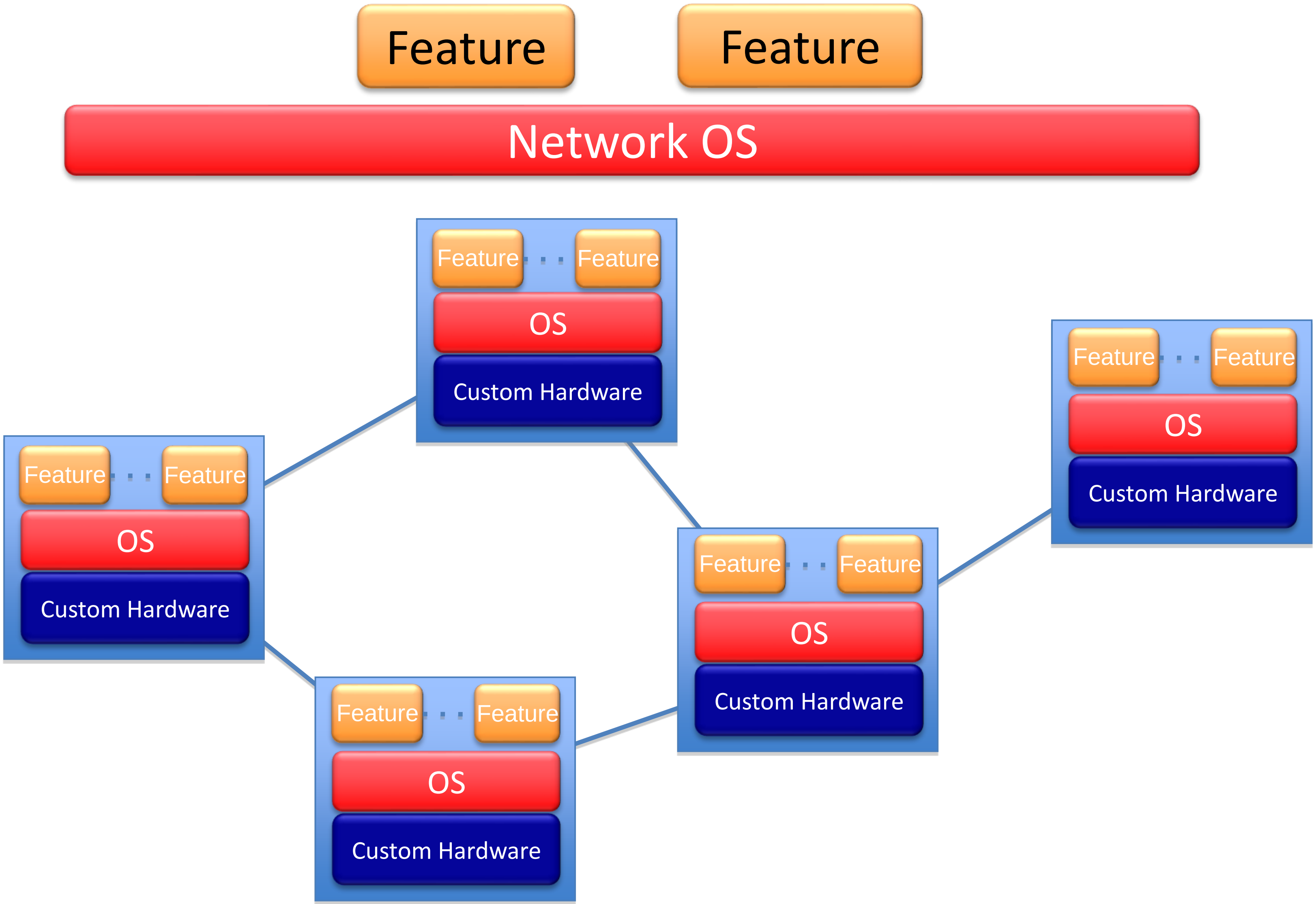
- Simpler management
  - No need to "invert" control-plane operations
- Faster pace of innovation
  - Less dependence on vendors and standards
- Easier interoperability
  - Compatibility only in "wire" protocols
- Simpler, cheaper equipment
  - Minimal software



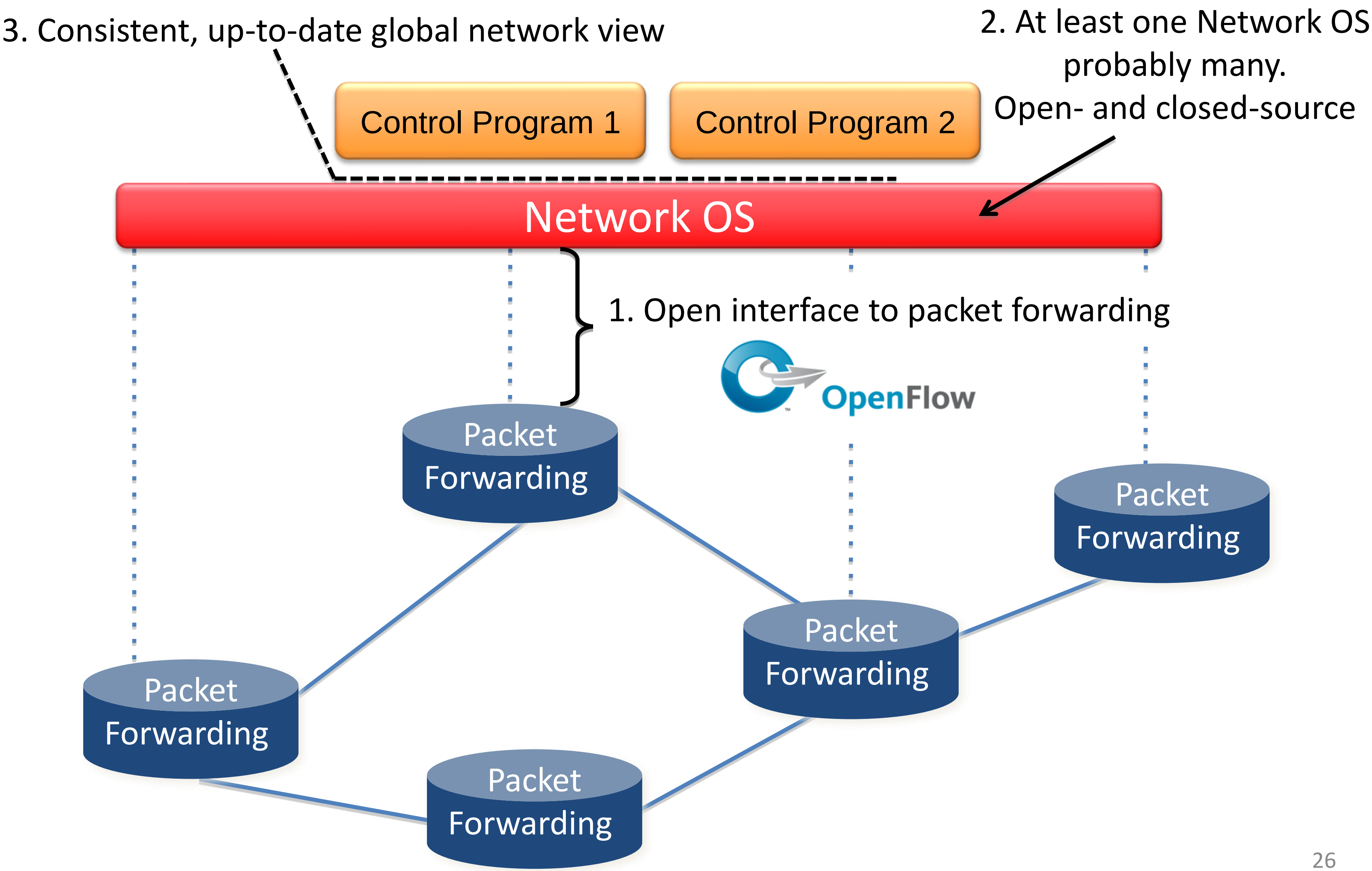
SDN advantages

- **Flexibility:** topologies, routing and forwarding architecture, independent configuration
- **Manageability:** separate policy and mechanism
- **Scalability:** maximize number of co-existing virtual networks
- **Security and Isolation:** isolate both the logical networks and the resources
- **Programmability:** programmable routers, etc.
- **Heterogeneity:** support for different technologies

How SDN changes the network



How SDN changes the network

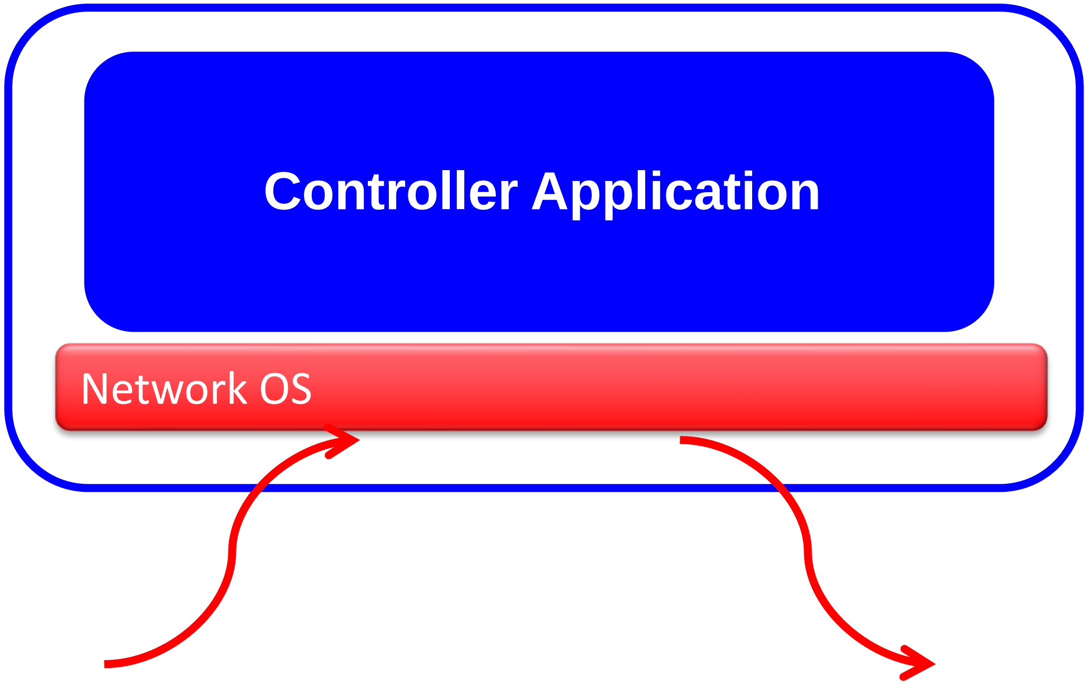


Control Program (controller app)

Control program operates on view of network

- **Input**: global network view (graph/database)
- **Output**: configuration of each network device

Controller: Programmability



<u>**Events from switches**</u>

Topology changes,
Traffic statistics,
Arriving packets

<u>**Commands to switches**</u>

(Un)install rules,
Query statistics,
Send packets

Control Program (controller app)

Control program is not a distributed system

– Abstraction hides details of distributed state

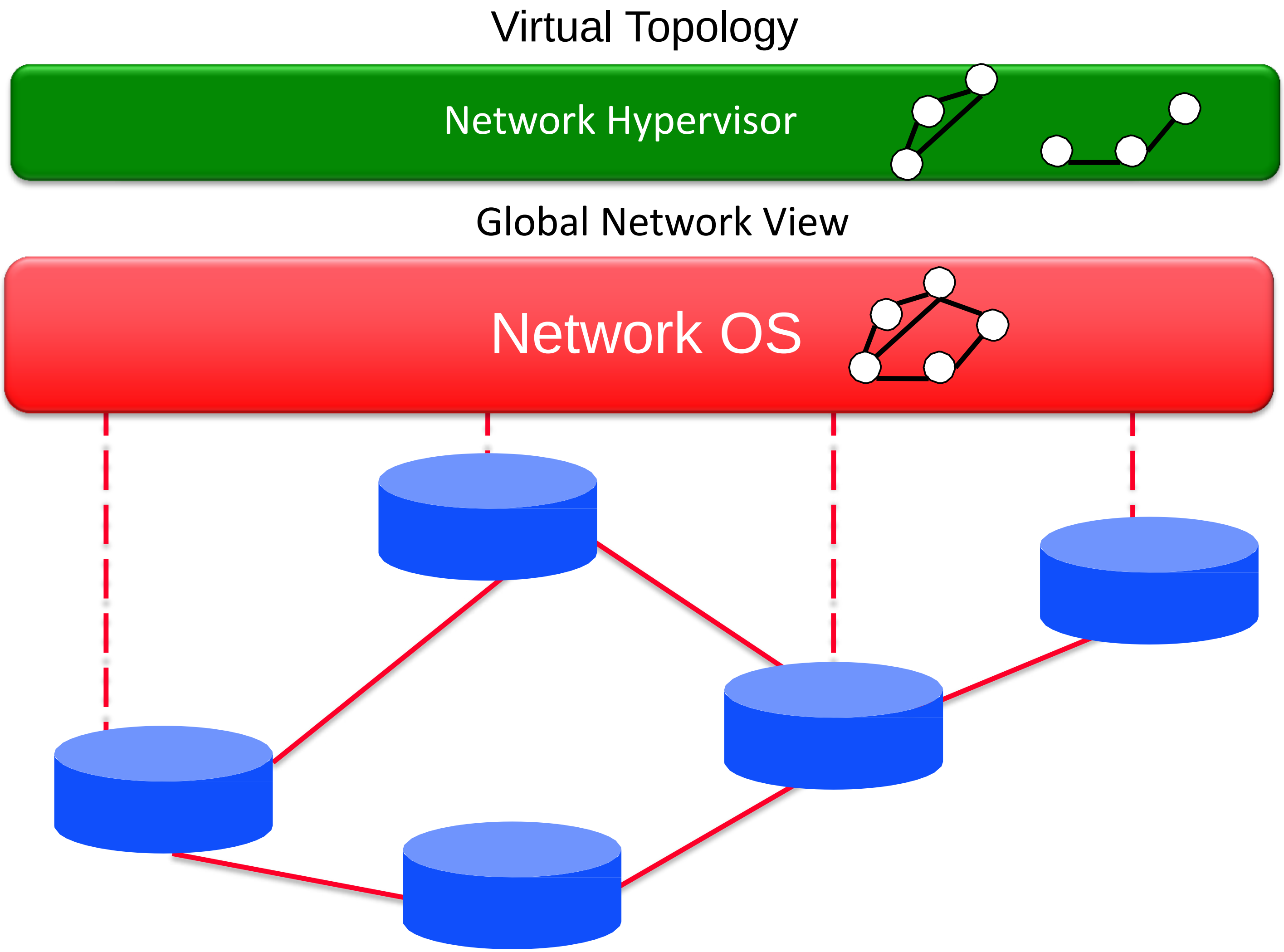
Abstraction of the physical network

– Support for multiple logical networks running on a common shared physical substrate

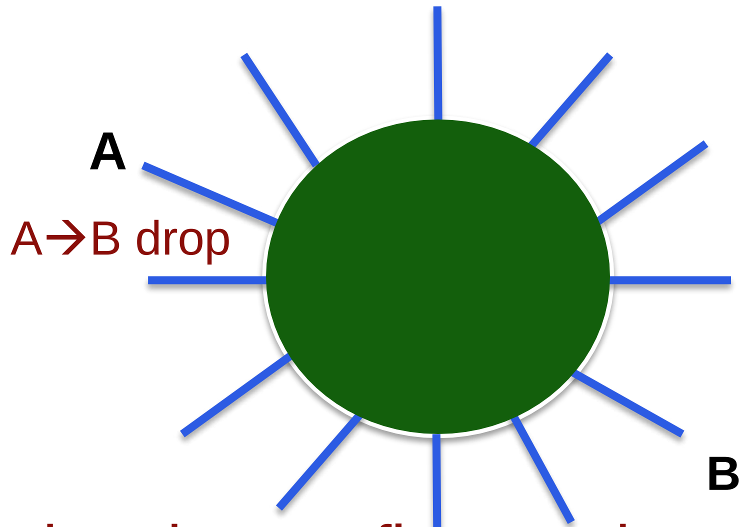
Aspects of the network that can be virtualized

– **Nodes:** Virtual machines
– **Links:** Tunnels (e.g., Ethernet GRE)
– Storage

Network virtualization

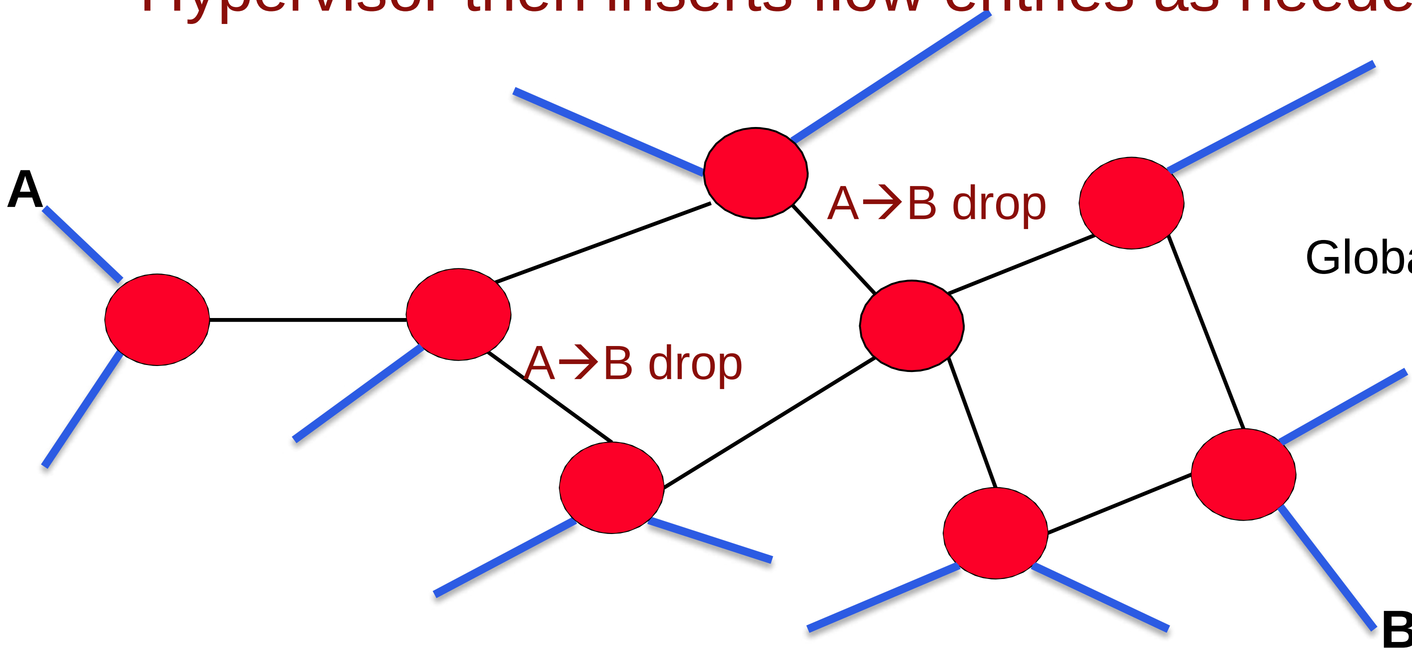


Virtualization simplifies control program



Abstract Network View

Hypervisor then inserts flow entries as needed



Global Network View

Does SDN Simplify the Network?

Abstraction doesn’t eliminate complexity

- Network OS and Hypervisor are still complicated pieces of code

SDN main achievements

- Simplifies interface for control program (user-specific)
- Pushes complexity into reusable code (SDN platform)

Just like compilers….

Forwarding Abstraction

Purpose: Standard way of defining forwarding state

– Flexible
  - Behavior specified by control plane
  - Built from basic set of forwarding primitives
– Minimal
  - Streamlined for speed and low-power
  - Control program not vendor-specific

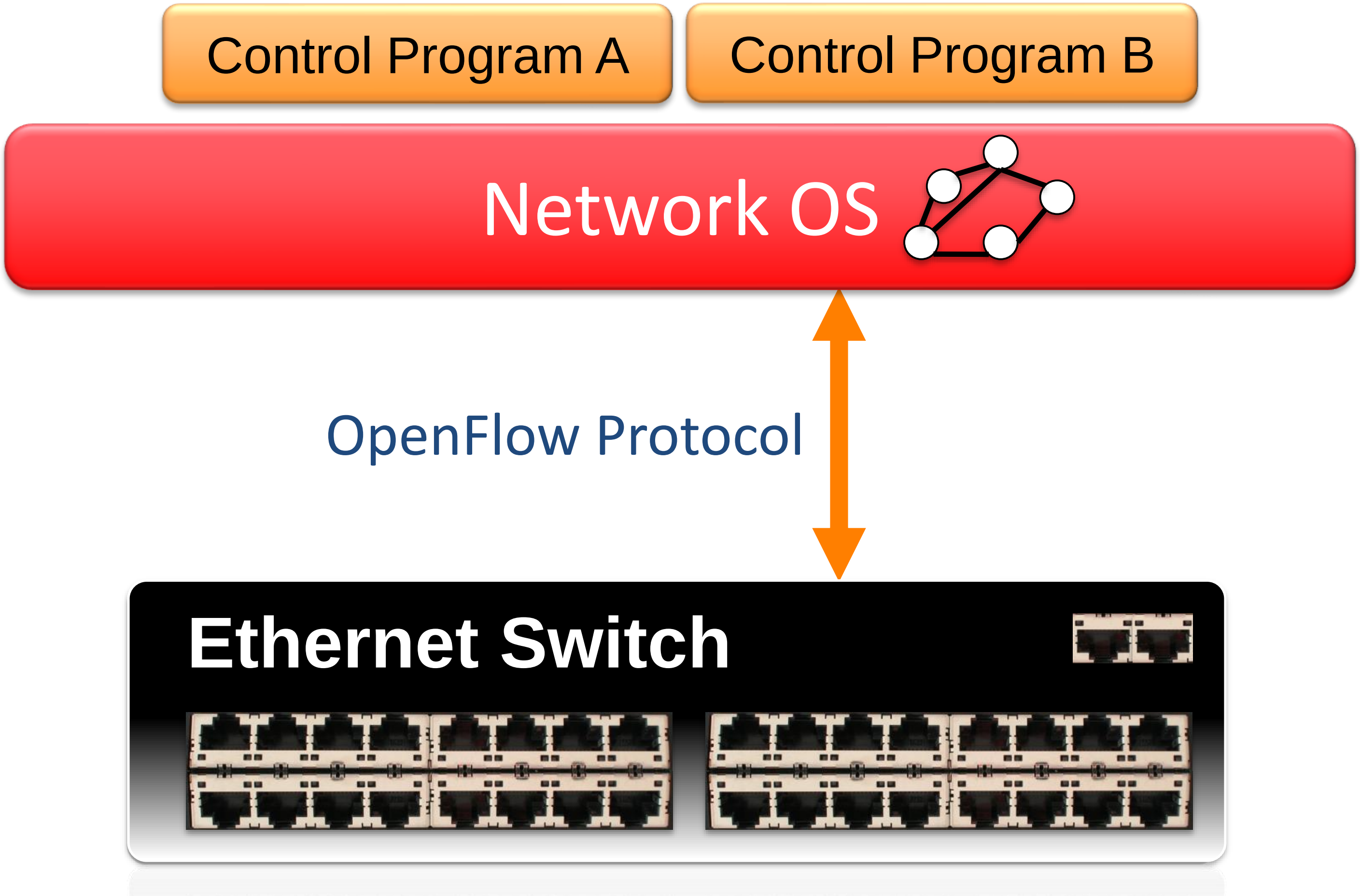
- OpenFlow is an example of such an abstraction

OPENFLOW BASICS

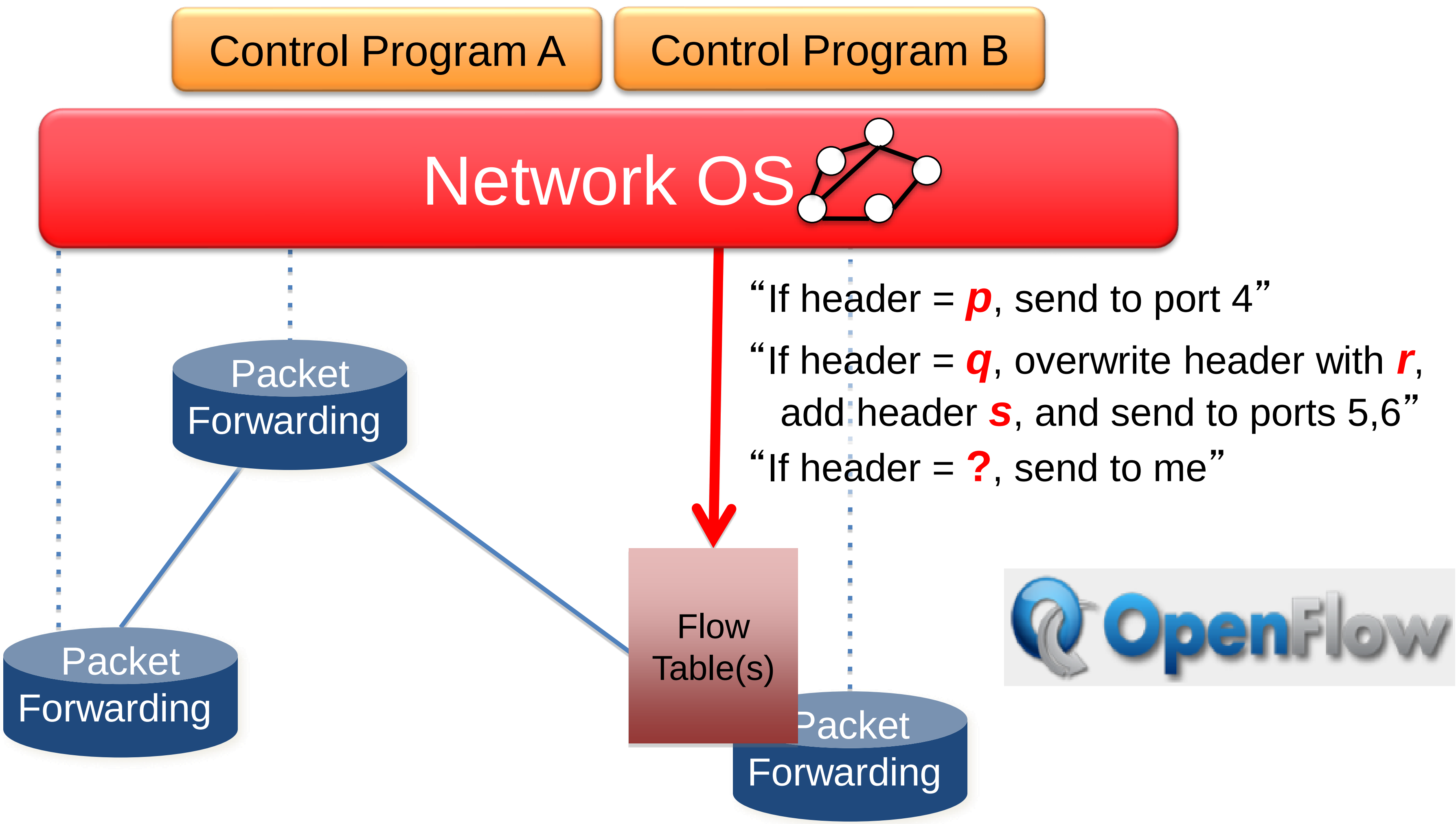
OpenFlow in the Wild

- Open Networking Foundation
  – Google, Facebook, Microsoft, Yahoo, Verizon, Deutsche Telekom, and many other companies
- Commercial OpenFlow switches
  – HP, NEC, Quanta, Dell, IBM, Juniper, …
- Network operating systems
  – NOX, Beacon, ONIX, Floodlight, Ryu, OpenDaylight
- Network deployments
  – Campuses, and research backbone networks
  – Commercial deployments (e.g., Google backbone)

OpenFlow Basics



OpenFlow Basics



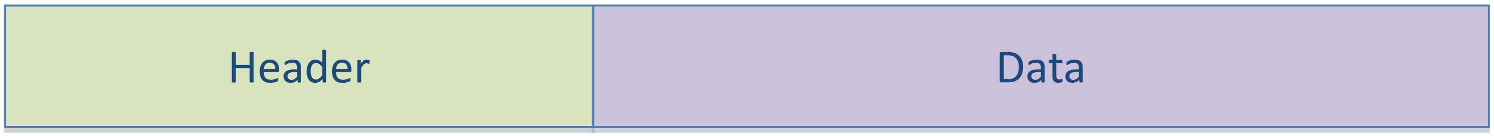
OpenFlow Rules

Exploit the flow table in switches, routers, and chipsets

Flow 1.	Rule (exact & wildcard)	Action	Statistics
Flow 2.	Rule (exact & wildcard)	Action	Statistics
Flow 3.	Rule (exact & wildcard)	Action	Statistics
Flow N.	Rule (exact & wildcard)	Default Action	Statistics

Primitives <match, action>

Match arbitrary bits in headers:



Match: 1000x01xx0101001x

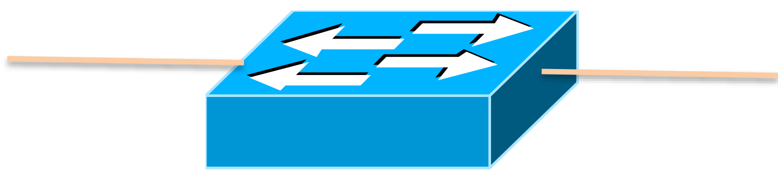
- Match on any header, or new header
- Allows any flow granularity

Action

- Forward to port(s), drop, send to controller
- Overwrite header with mask, push or pop
- Forward at specific bit-rate

Data-Plane: Simple Packet Handling

- Simple packet-handling rules
  - Pattern: match packet header bits
  - Actions: drop, forward, modify, send to controller
  - Priority: fixes 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

Unifies Different Kinds of Boxes

- Router
  - Match: 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

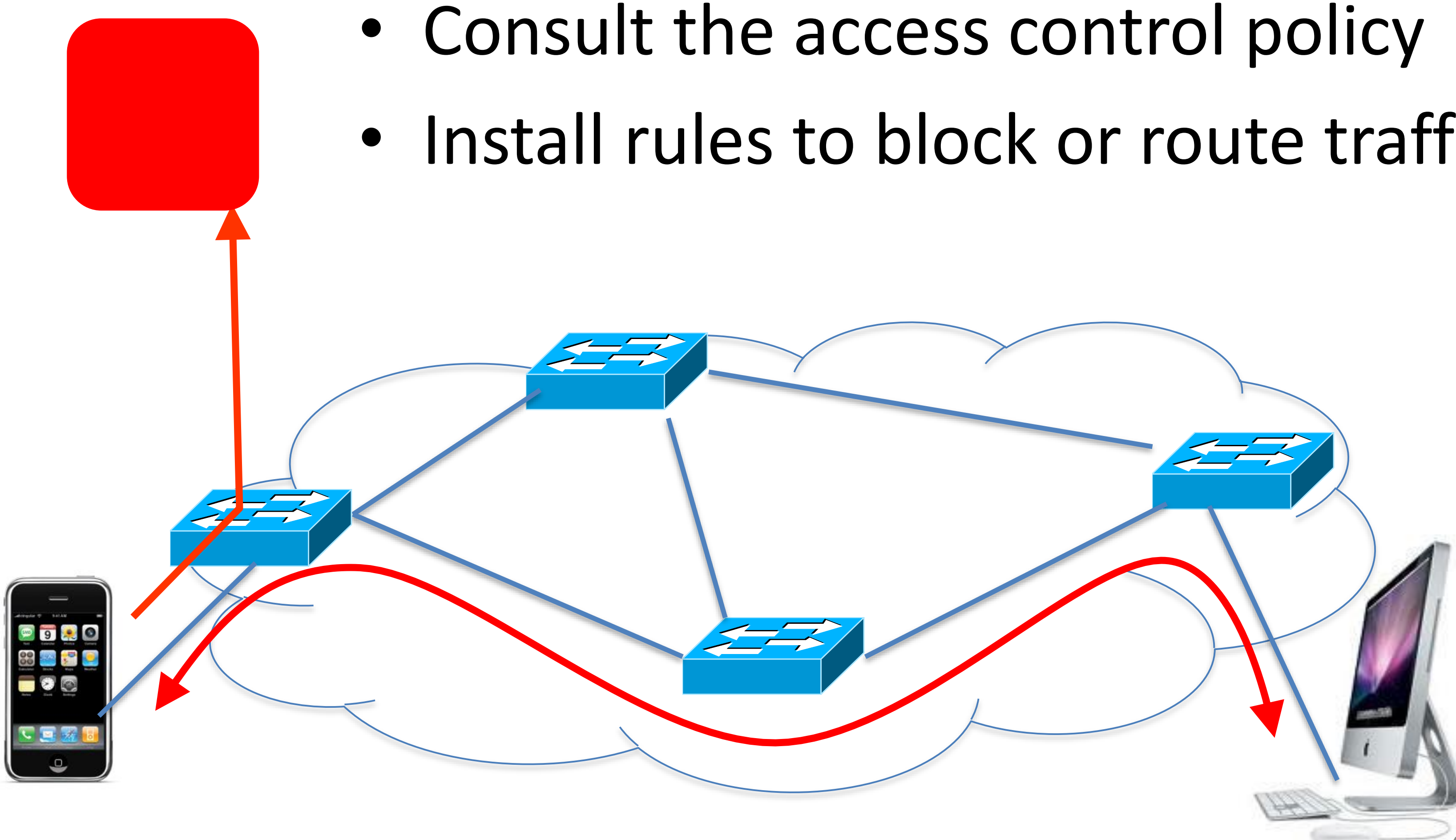
Example OpenFlow Applications

- **Dynamic access control**
- **Seamless mobility/migration**
- **Server load balancing**
- **Network virtualization**
- Using multiple wireless access points
- Energy-efficient networking
- Adaptive traffic monitoring
- Denial-of-Service attack detection

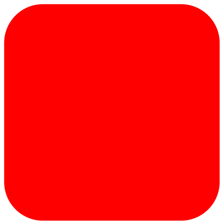
See http://www.openflow.org/videos/

E.g.: Dynamic Access Control

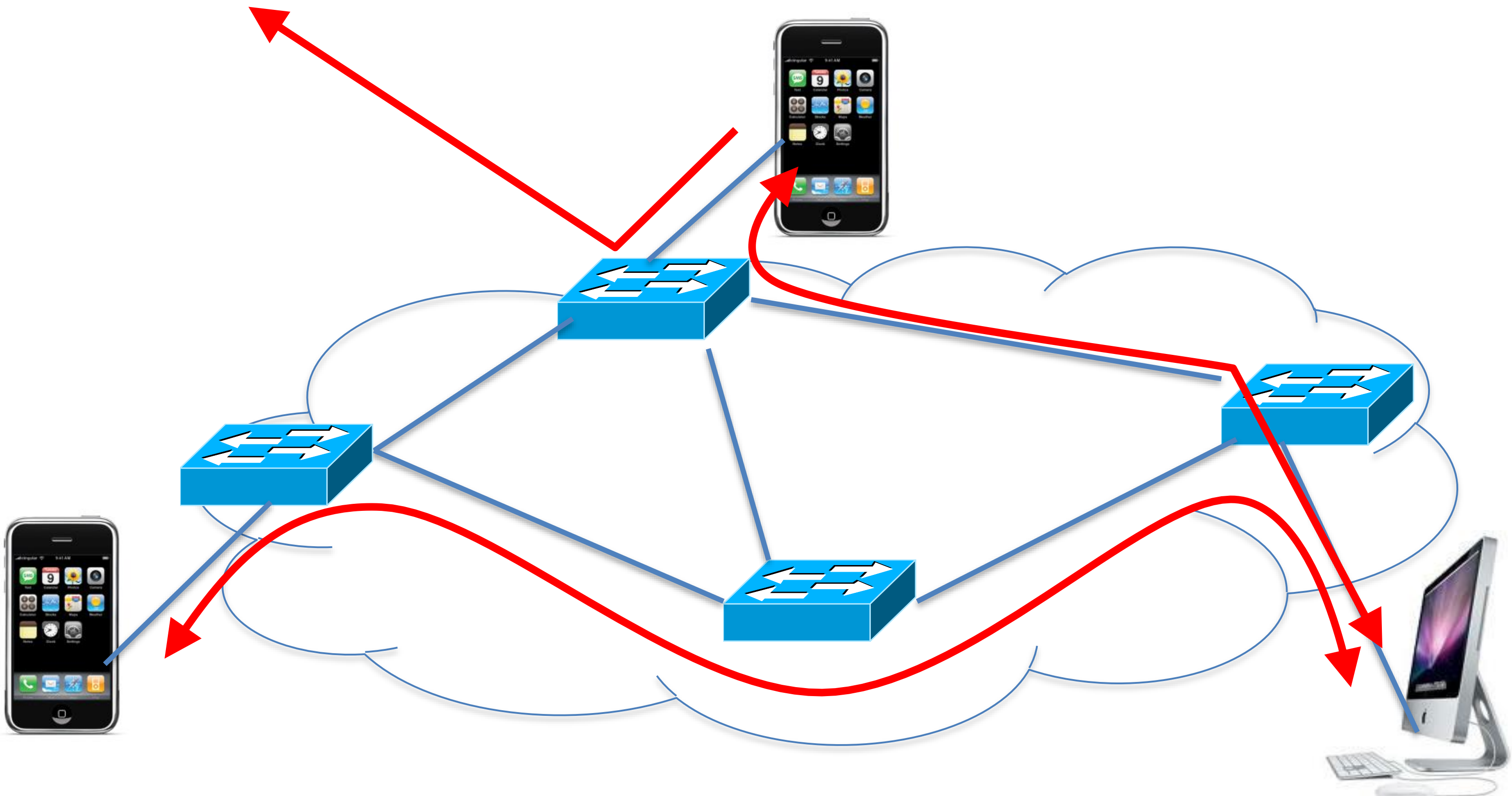
- Inspect first packet of a connection
- Consult the access control policy
- Install rules to block or route traffic



E.g.: Seamless Mobility/Migration

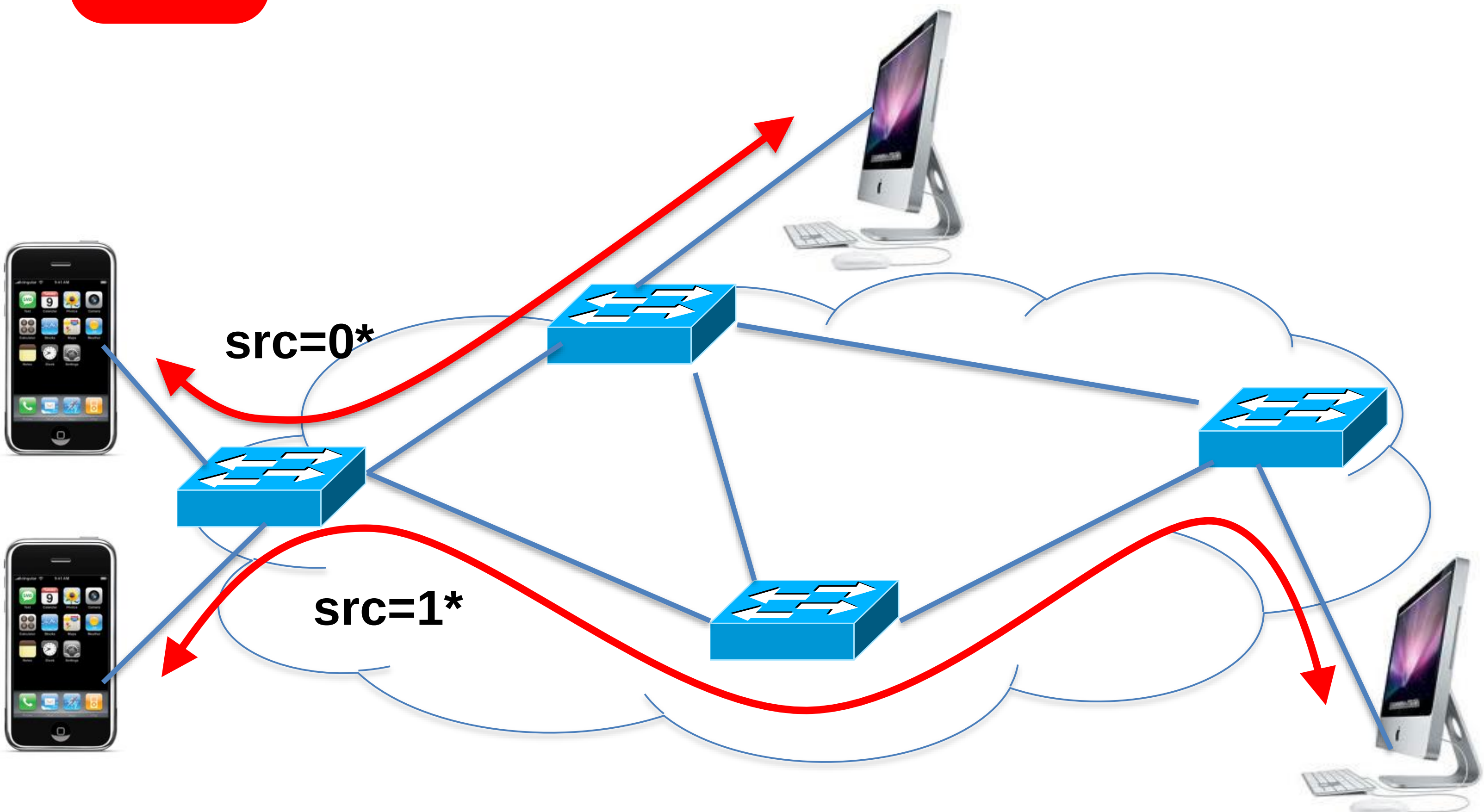


- See host send traffic at new location
- Modify rules to reroute the traffic



E.g.: Server Load Balancing

- Pre-install load-balancing policy
- Split traffic based on source IP



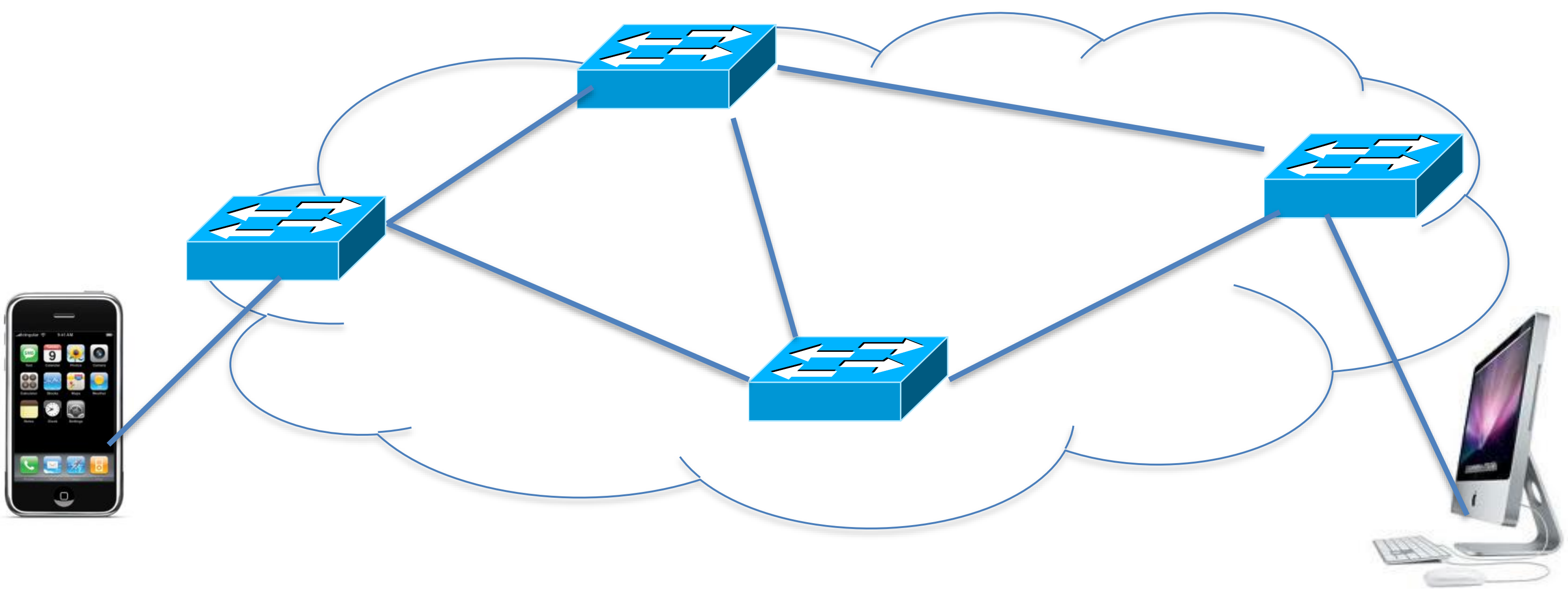
E.g.: Network Virtualization

Controller #1

Controller #2

Controller #3

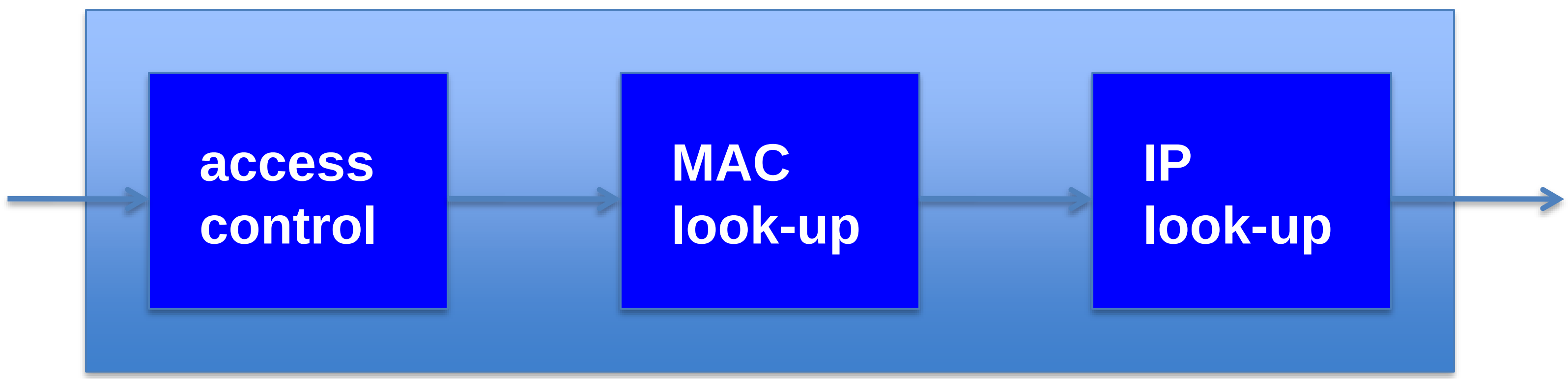
Partition the space of packet headers



SDN CHALLENGES

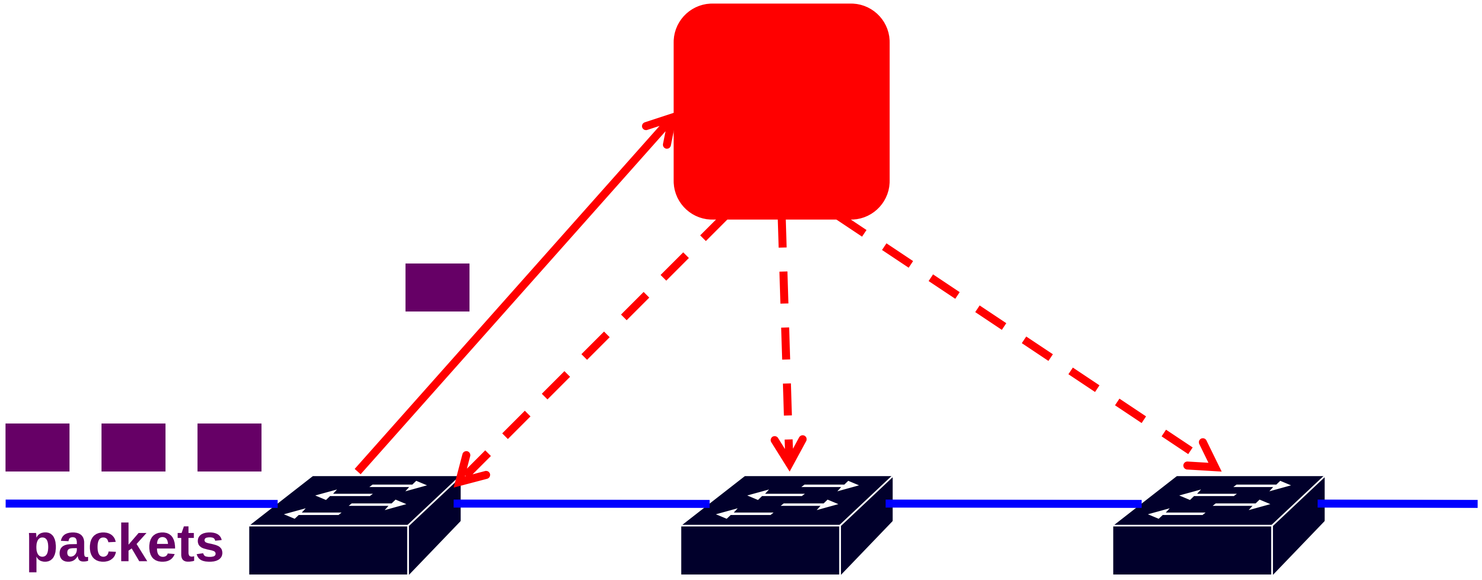
Heterogeneous Switches

- Number of packet-handling rules
- Range of matches and actions
- Multi-stage pipeline of packet processing
- Offload some control-plane functionality (?)

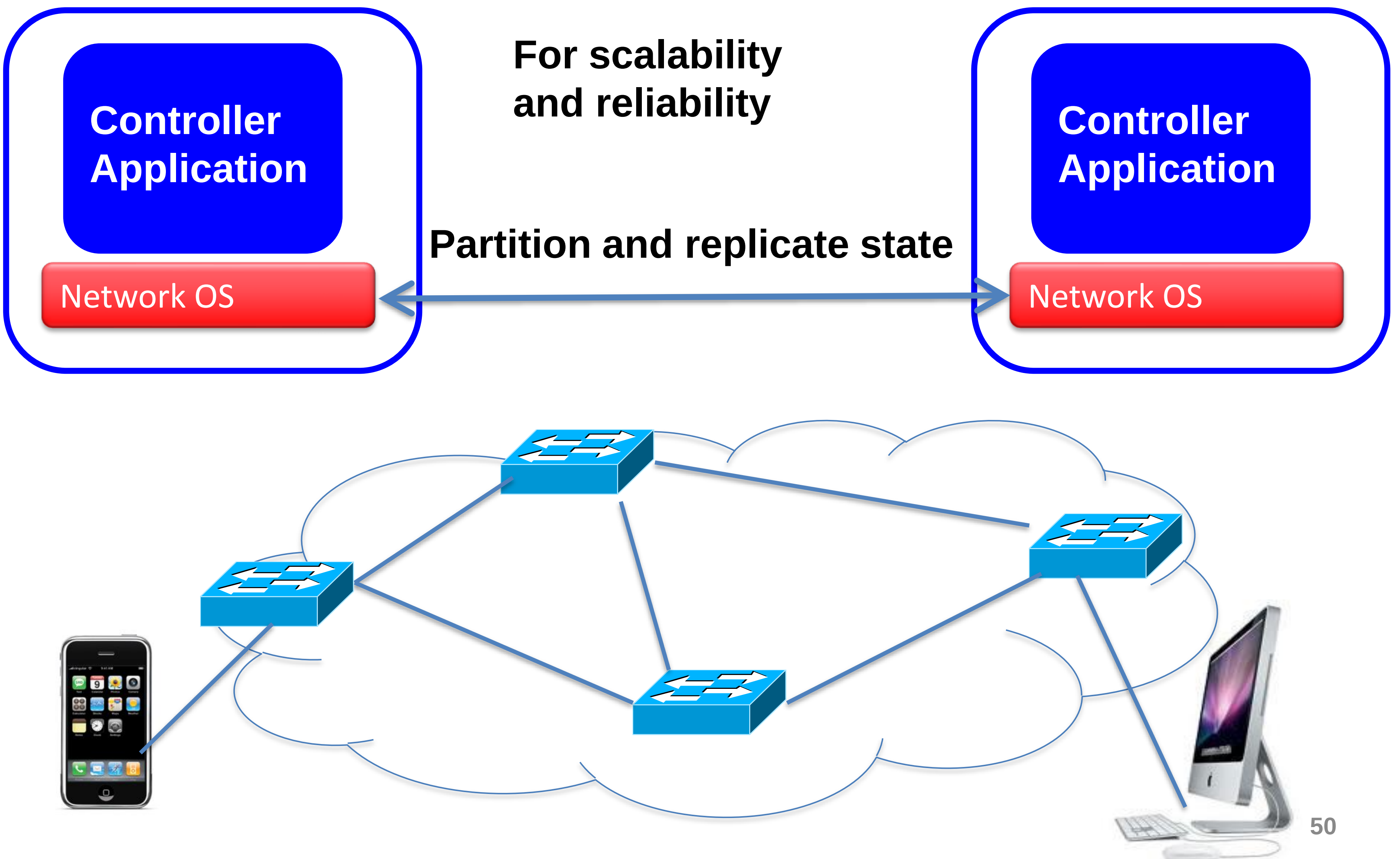


Controller Delay and Overhead

- Controller is much slower the the switch
- Processing packets leads to delay and overhead
- Need to keep most packets in the "fast path"



Distributed Controller



Testing and Debugging

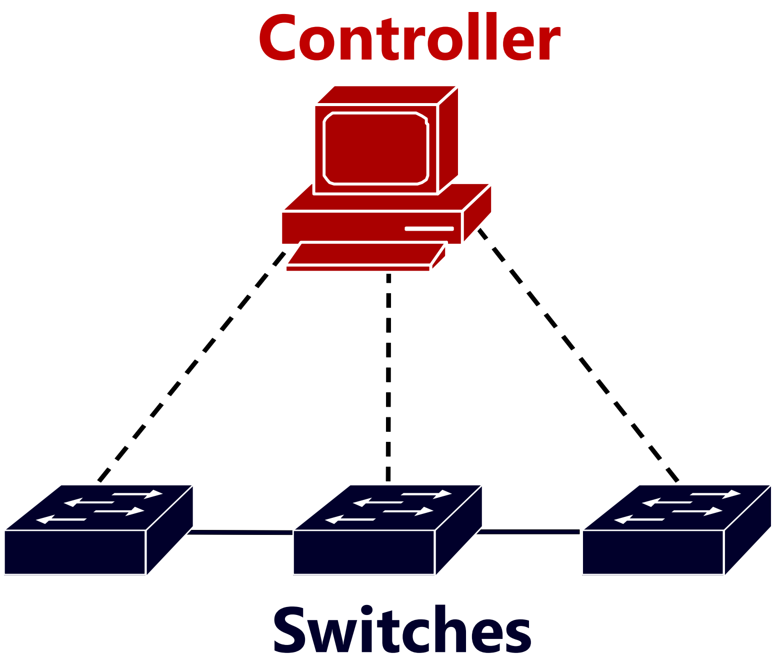
- OpenFlow makes programming the network possible
  – Network-wide view at controller
  – Direct control over data plane

- Plenty of room for bugs
  – Still a complex, distributed system

- Need for testing techniques
  – Controller applications
  – Controller and switches
  – Rules installed in the switches

Programming Abstractions

- Controller APIs are low-level
  - Thin veneer on the underlying hardware
- Need better languages
  - Composition of modules
  - Managing concurrency
  - Querying network state
  - Network-wide abstractions



Conclusion

- Rethinking networking
  - Open interfaces to the data plane
  - Separation of control and data
  - Leveraging techniques from distributed systems

- Significant momentum
  - In both research and industry