

# CS452/552

# Software Defined Networking

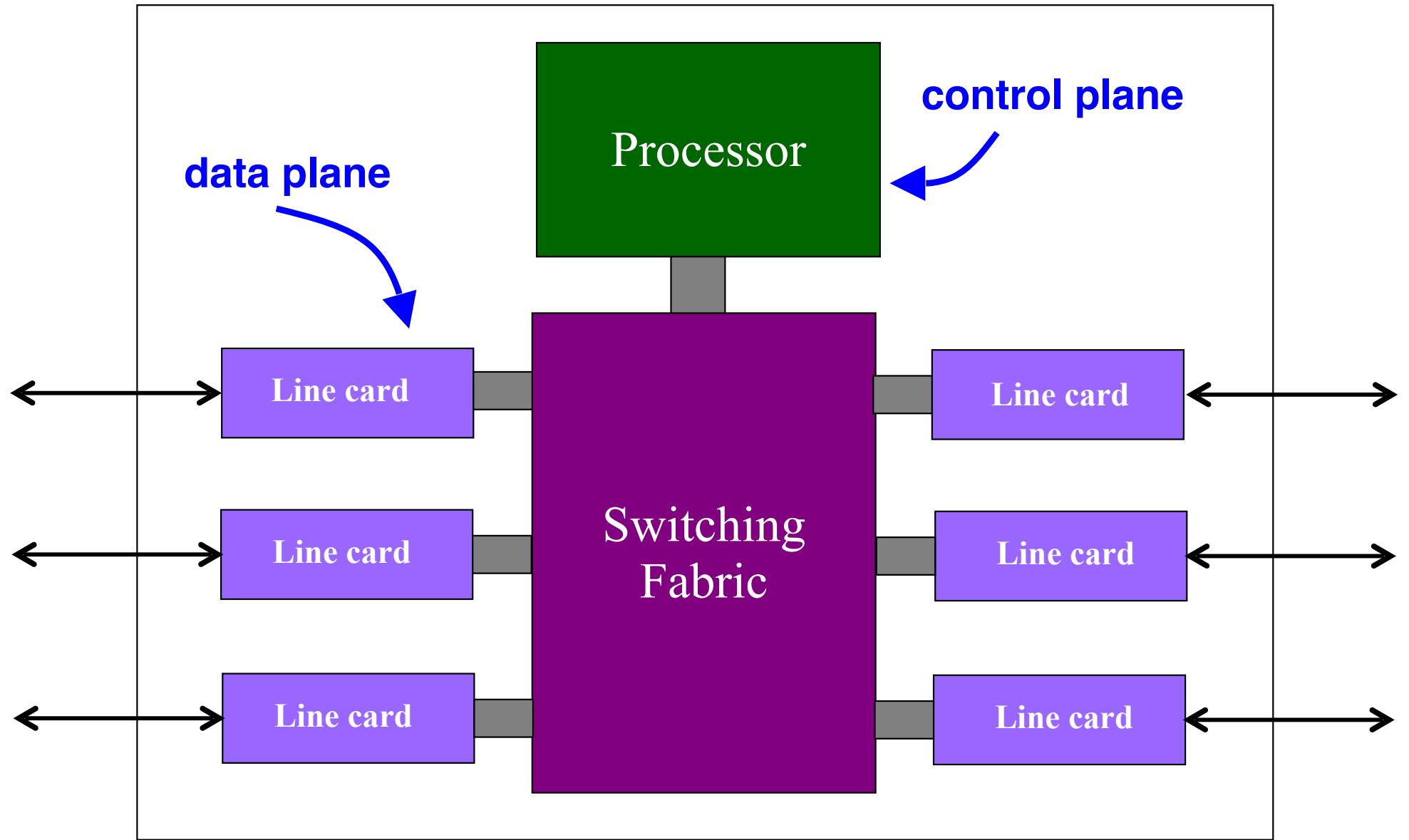
Based on slides by J. Rexford @ Princeton & N. McKeown @  
Stanford & S. Shenker @ Berkeley & P. Gill @ UMass

# Data, Control, and Management Planes

# Timescales

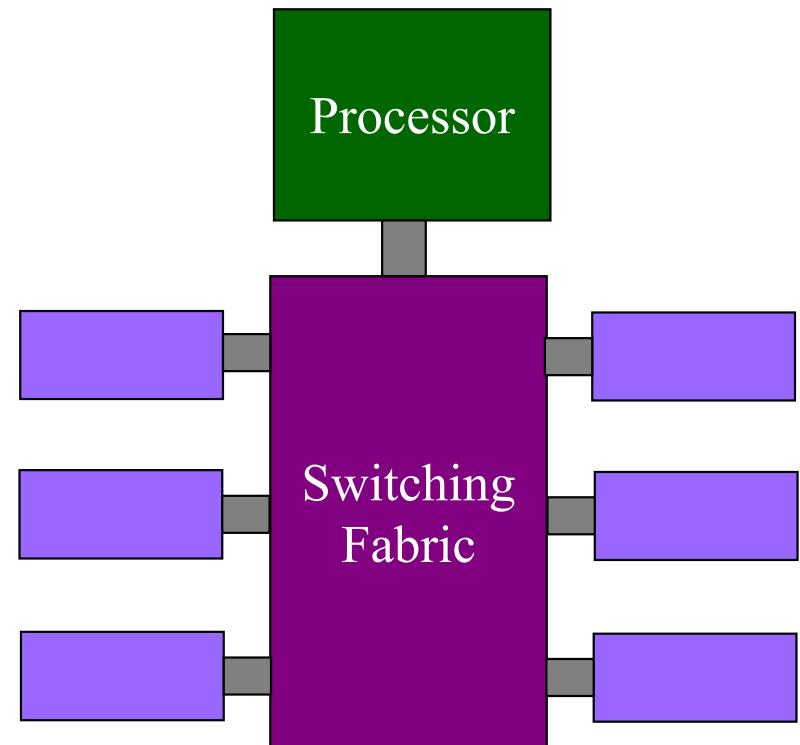
	Data	Control	Management
Time-scale	Packet (nsec)	Event (10 msec to sec)	Human (min to hours)
Tasks	Forwarding, buffering, filtering, scheduling	Routing, circuit set-up	Analysis, configuration
Location	Line-card hardware	Router software	Humans or scripts

# Data and Control Planes



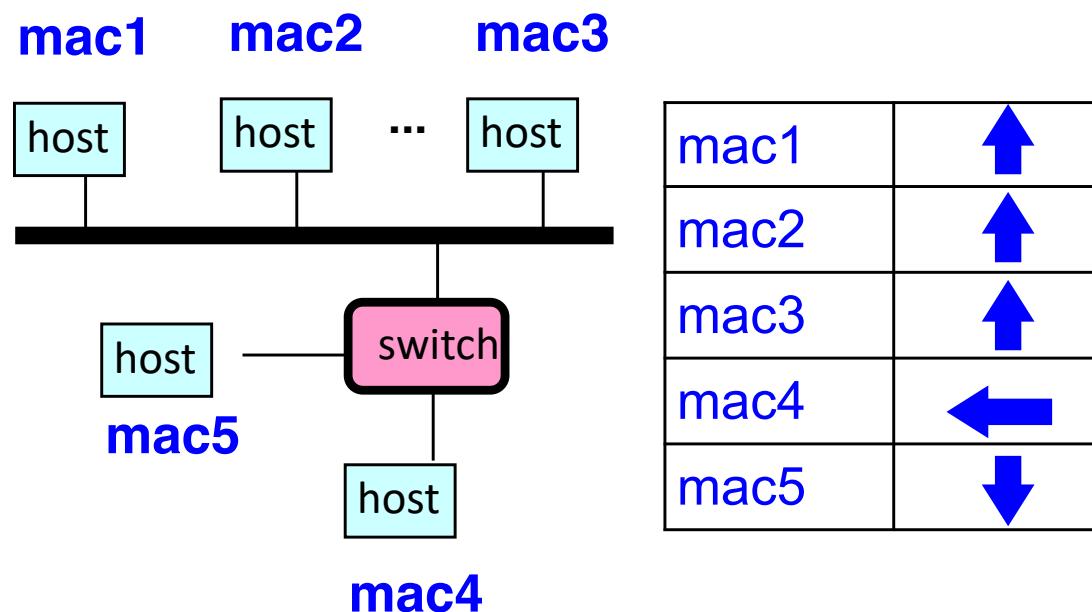
# Data Plane

- Streaming algorithms on packets
  - Matching on some bits
  - Perform some actions
- Wide range of functionality
  - Forwarding
  - Access control
  - Mapping header fields
  - Traffic monitoring
  - Buffering and marking
  - Shaping and scheduling
  - Deep packet inspection



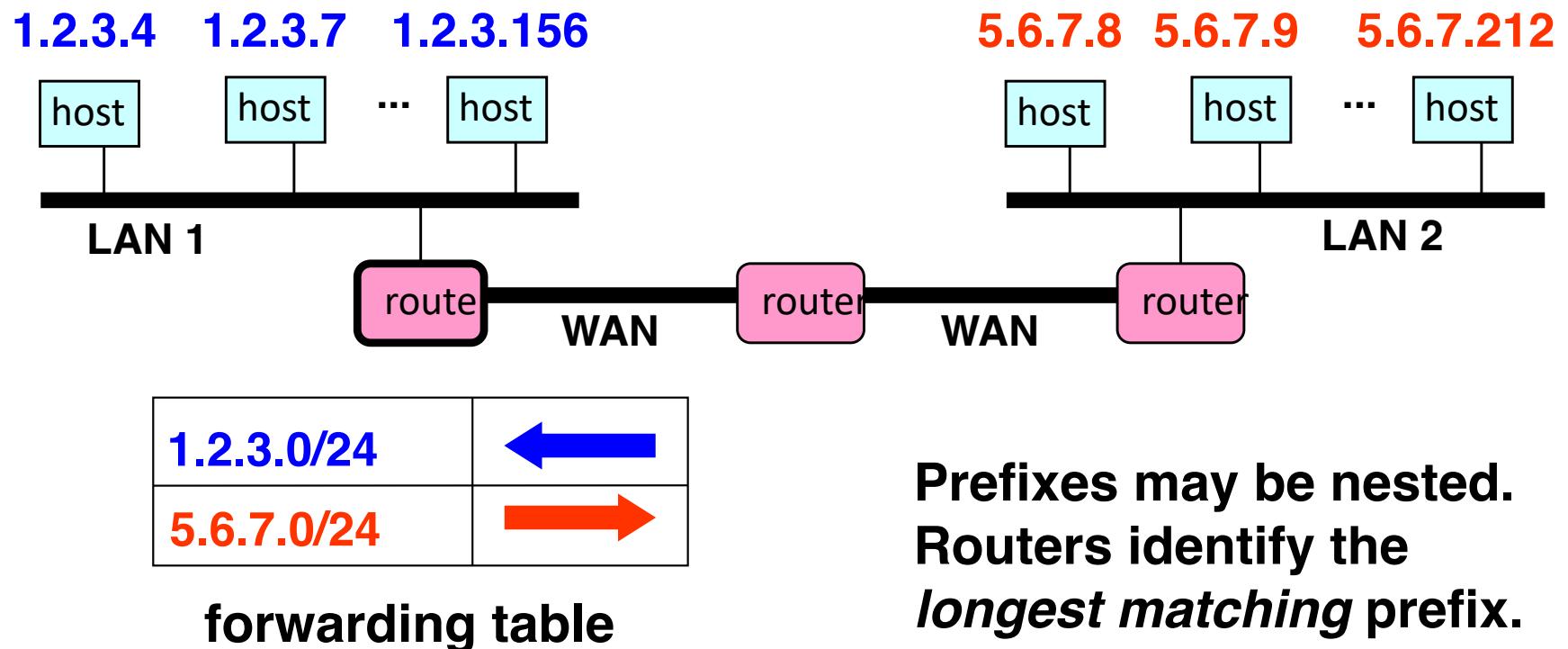
# Switch: Match on Destination MAC

- MAC addresses are location independent
  - Assigned by the vendor of the interface card
  - Cannot be aggregated across hosts in LAN



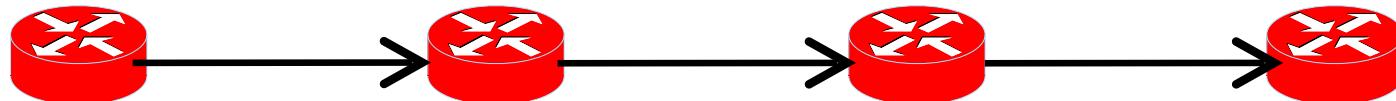
# Router: Match on IP Prefix

- IP addresses grouped into common subnets
  - Allocated by ICANN, regional registries, ISPs, and within individual organizations
  - Variable-length prefix identified by a mask length



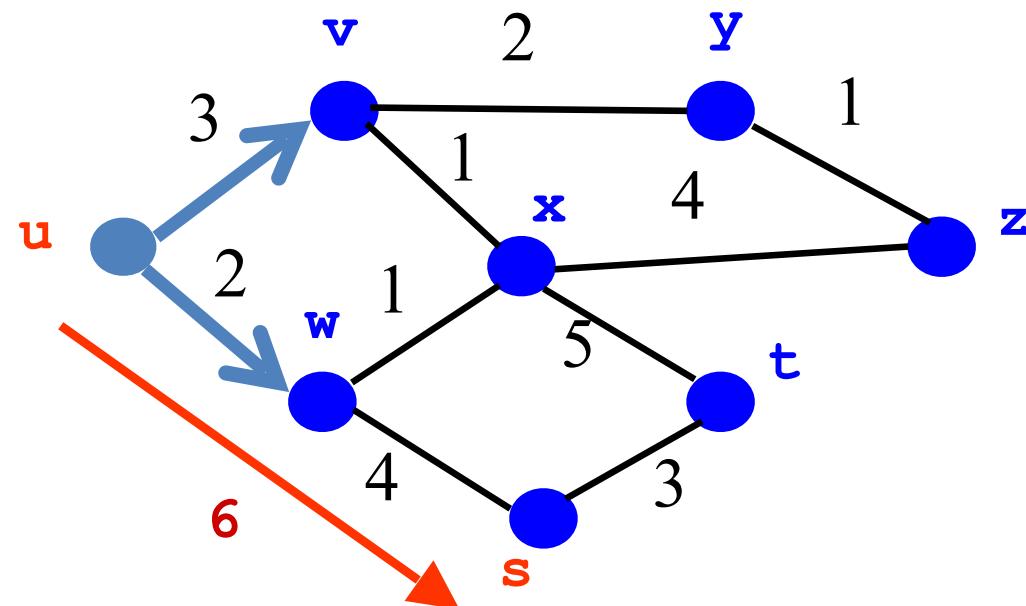
# Forwarding vs. Routing

- **Forwarding:** data plane
  - Directing a data packet to an outgoing link
  - Individual router *using* a forwarding table
- **Routing:** control plane
  - Computing paths the packets will follow
  - Routers talking amongst themselves
  - Individual router *creating* a forwarding table



# Routing Example: Shortest-Path Routing

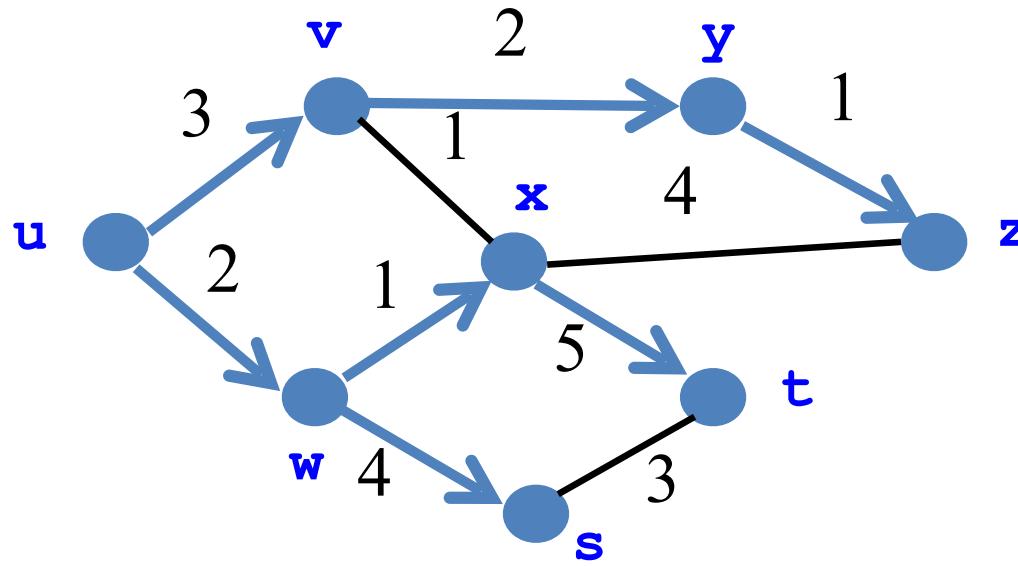
- Compute: *path costs* to all nodes
  - From a source  $u$  to all other nodes
  - Cost of the path through each link
  - Next hop along least-cost path to  $s$



Dest	link
v	$(u, v)$
w	$(u, w)$
x	$(u, w)$
y	$(u, v)$
z	$(u, v)$
s	$(u, w)$
t	$(u, w)$

# Distributed Control Plane

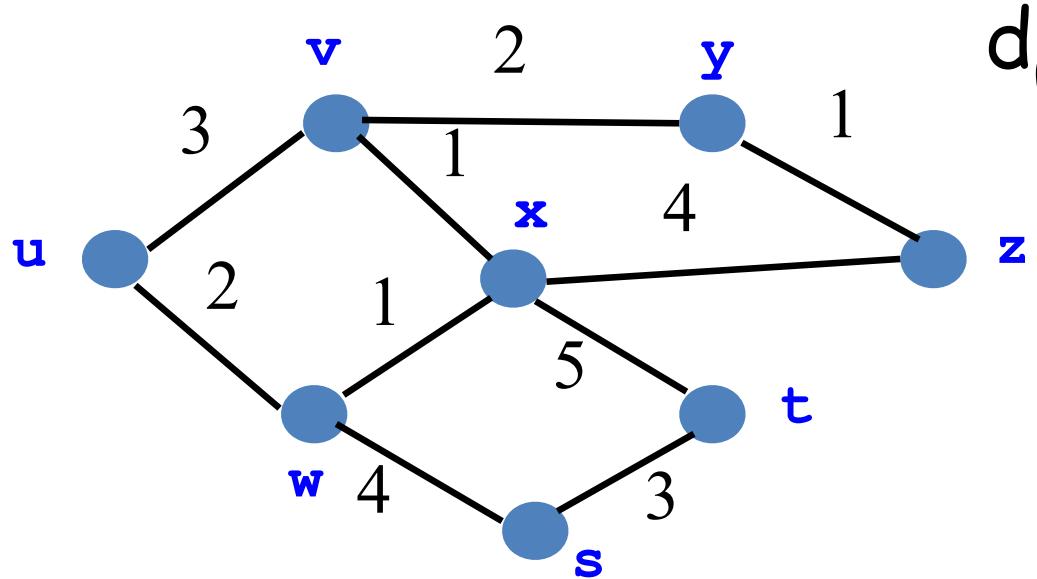
- **Link-state routing:** OSPF, IS-IS
  - Flood the entire topology to all nodes
  - Each node computes shortest paths
  - Dijkstra's algorithm



Dest	link
v	(u,v)
w	(u,w)
x	(u,w)
y	(u,v)
z	(u,v)
s	(u,w)
t	(u,w)

# Distributed Control Plane

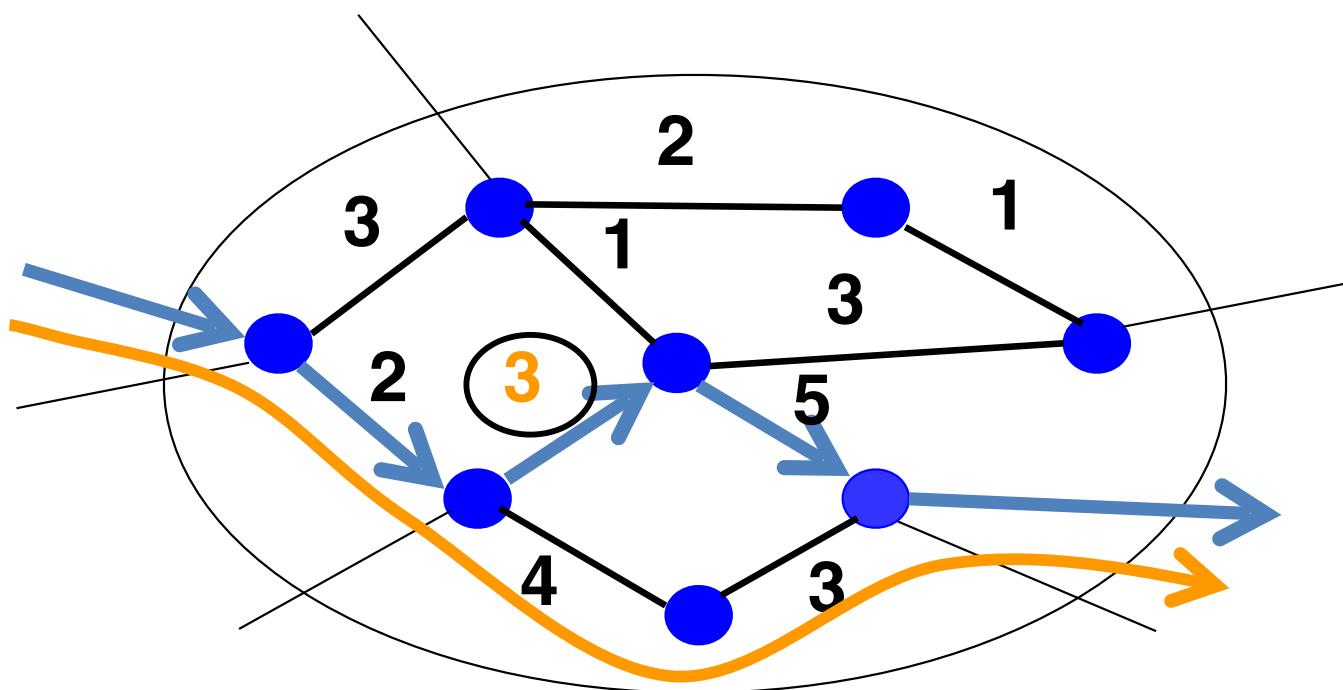
- Distance-vector routing: RIP, EIGRP
  - Each node computes path cost
  - ... based on each neighbors' path cost
  - Bellman-Ford algorithm



$$d_u(z) = \min\{c(u,v) + d_v(z), c(u,w) + d_w(z)\}$$

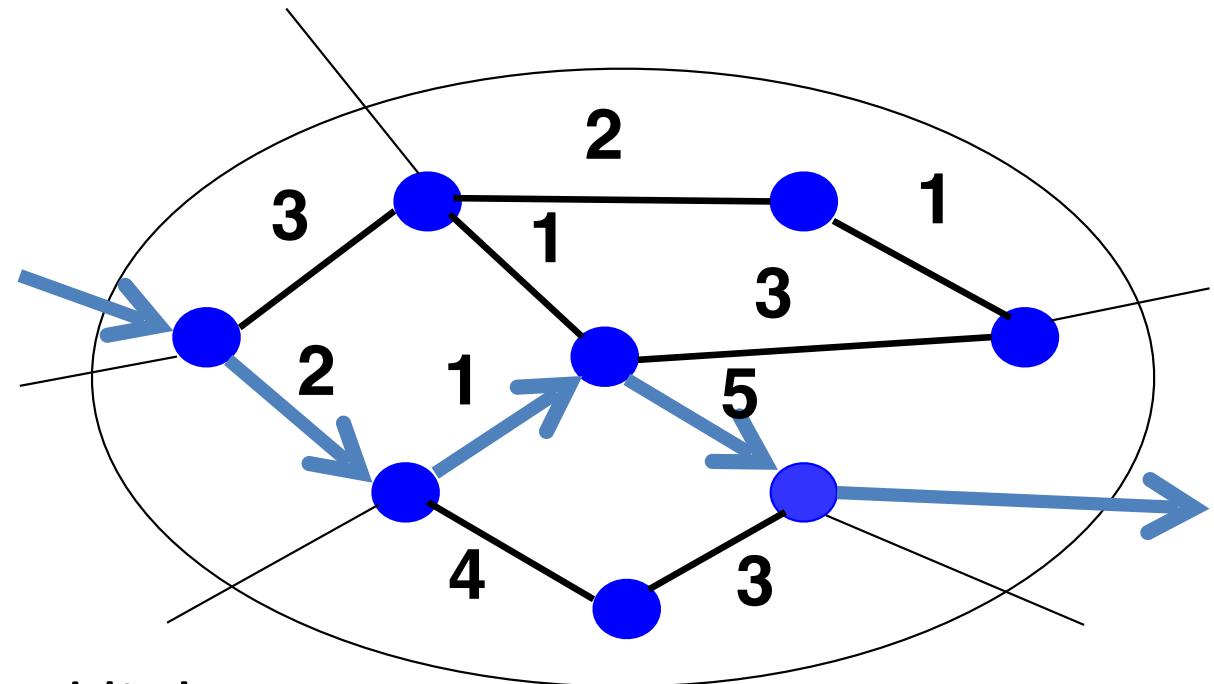
# Traffic Engineering Problem

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



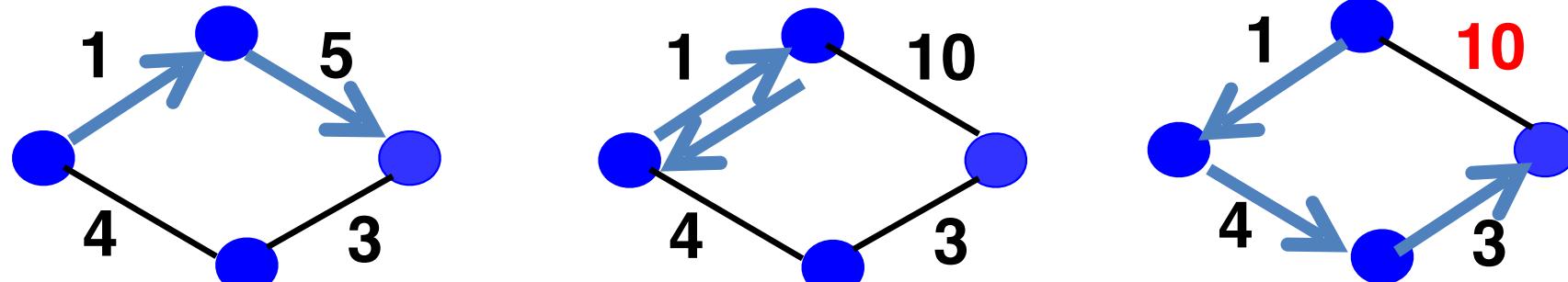
# Traffic Engineering: Optimization

- Inputs
  - Network topology
  - Link capacities
  - Traffic matrix
- Output
  - Link weights
- Objective
  - Minimize max-utilized link
  - Or, minimize a sum of link congestion
  - Or, maximize QoS for highest paying customers
  - Or, something else



# Transient Routing Disruptions

- Topology changes
  - Link weight change
  - Node/link failure or recovery
- Routing convergence
  - Nodes temporarily disagree how to route
  - Leading to transient loops and blackholes



# Management Plane Challenges

- Indirect control
  - Changing weights instead of paths
  - Complex optimization problem
- Uncoordinated control
  - Cannot control which router updates first
- Interacting protocols and mechanisms
  - Routing and forwarding
  - Naming and addressing
  - Access control
  - Quality of service
  - ...

# Software Defined Networking (high level view)

# Virtualization of networks

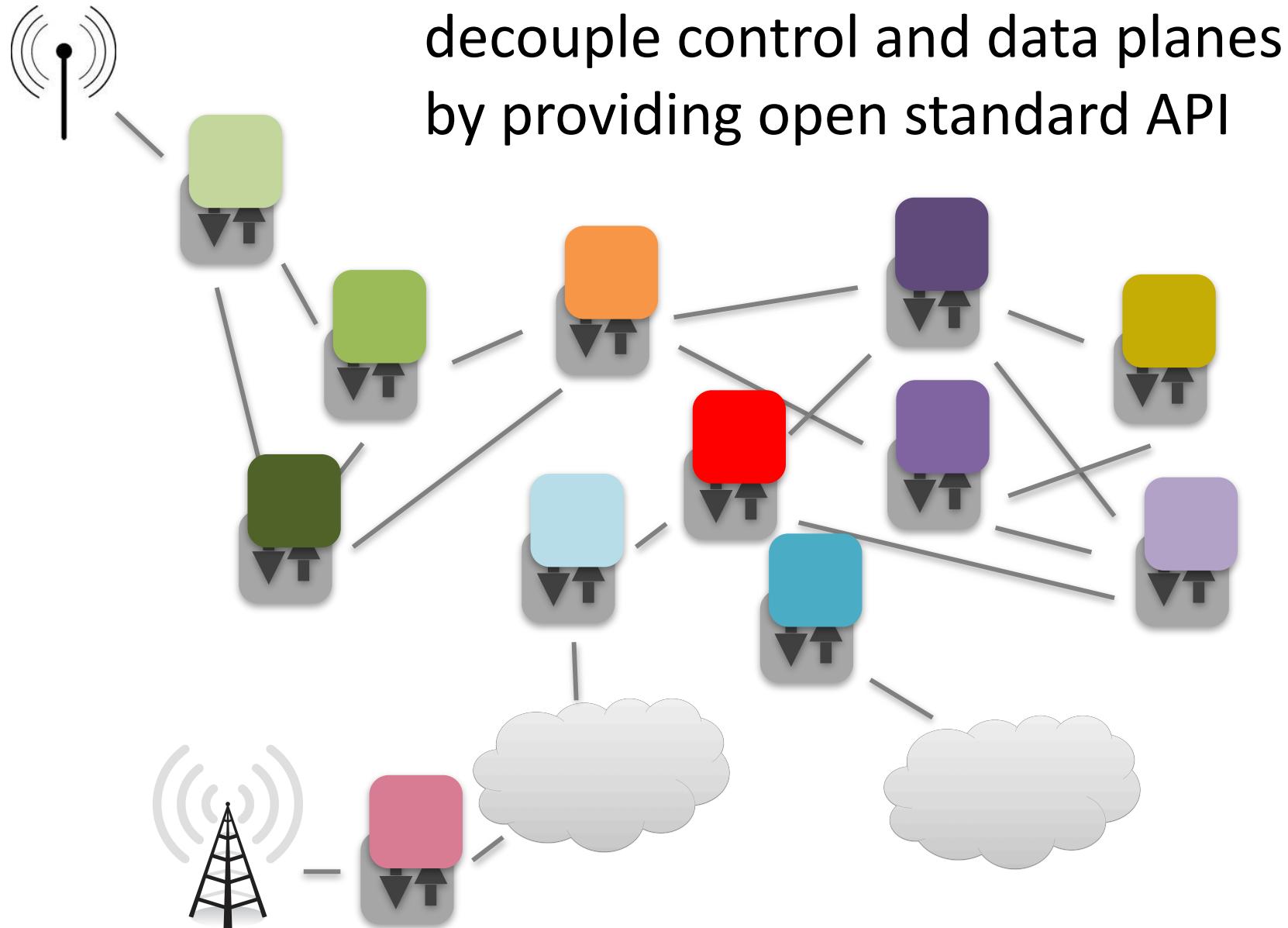
Virtualization of resources: a powerful abstraction in systems engineering:

- computing examples: virtual memory, virtual devices
  - Virtual machines: e.g., java, vmware
  - IBM VM os from 1960's/70's
- layering of abstractions: don't sweat the details of the lower layer, only deal with lower layers abstractly

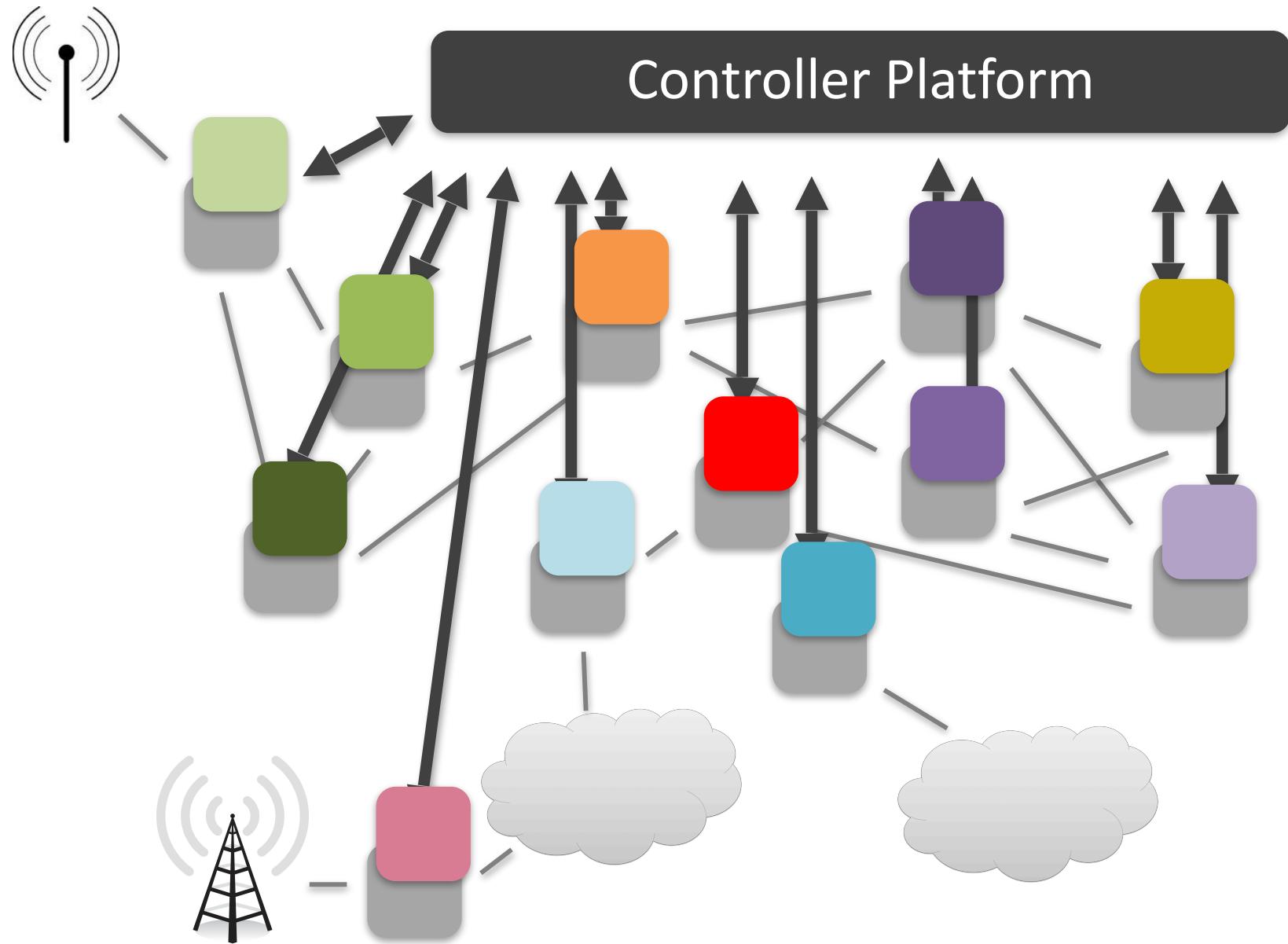
# Recap: Timescales

	Data	Control	Management
Time-scale	Packet (nsec)	Event (10 msec to sec)	Human (min to hours)
Tasks	Forwarding, buffering, filtering, scheduling	Routing, circuit set-up	Analysis, configuration
Location	Line-card hardware	Router software	Humans or scripts

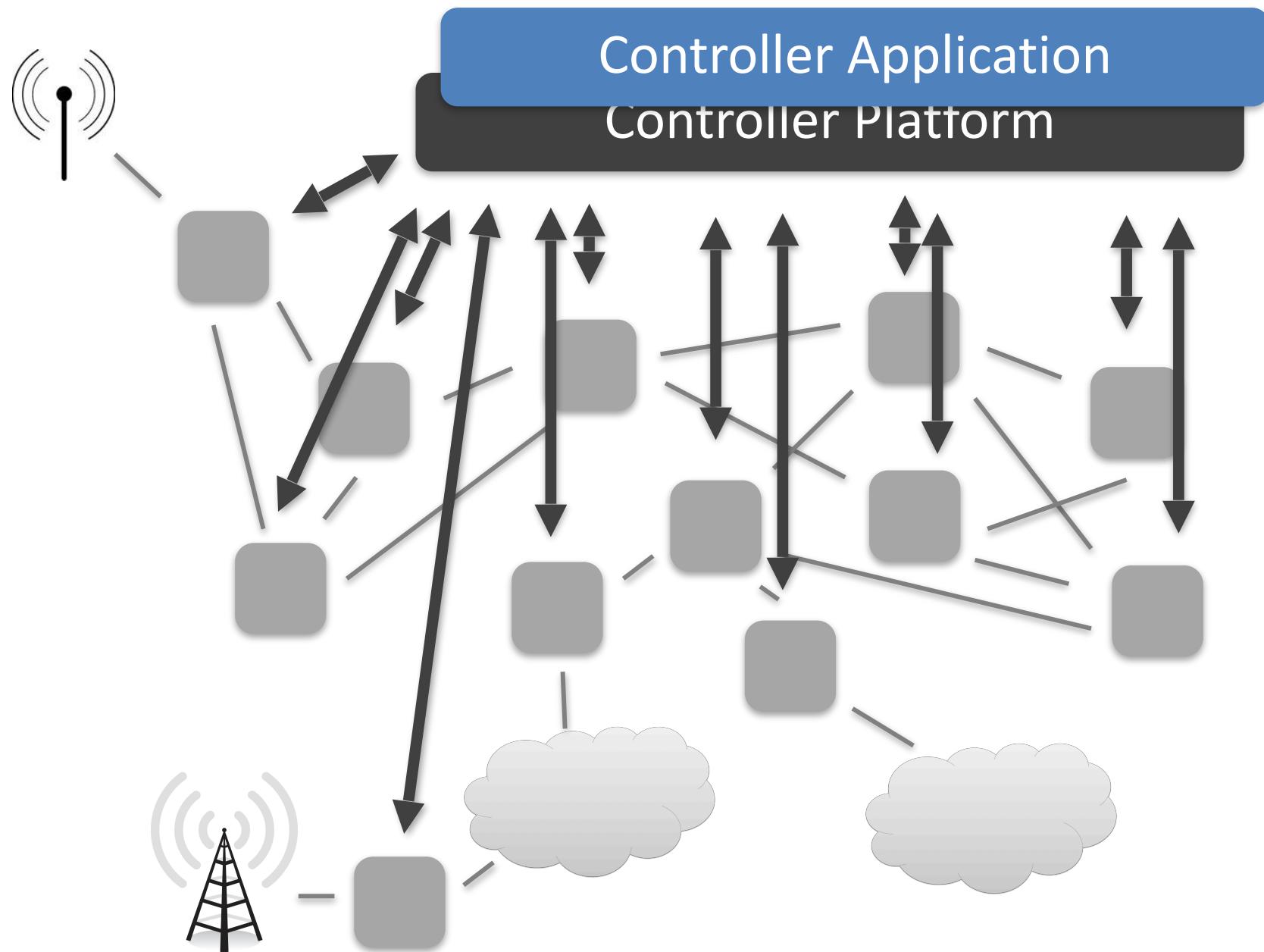
# Control/Data Separation



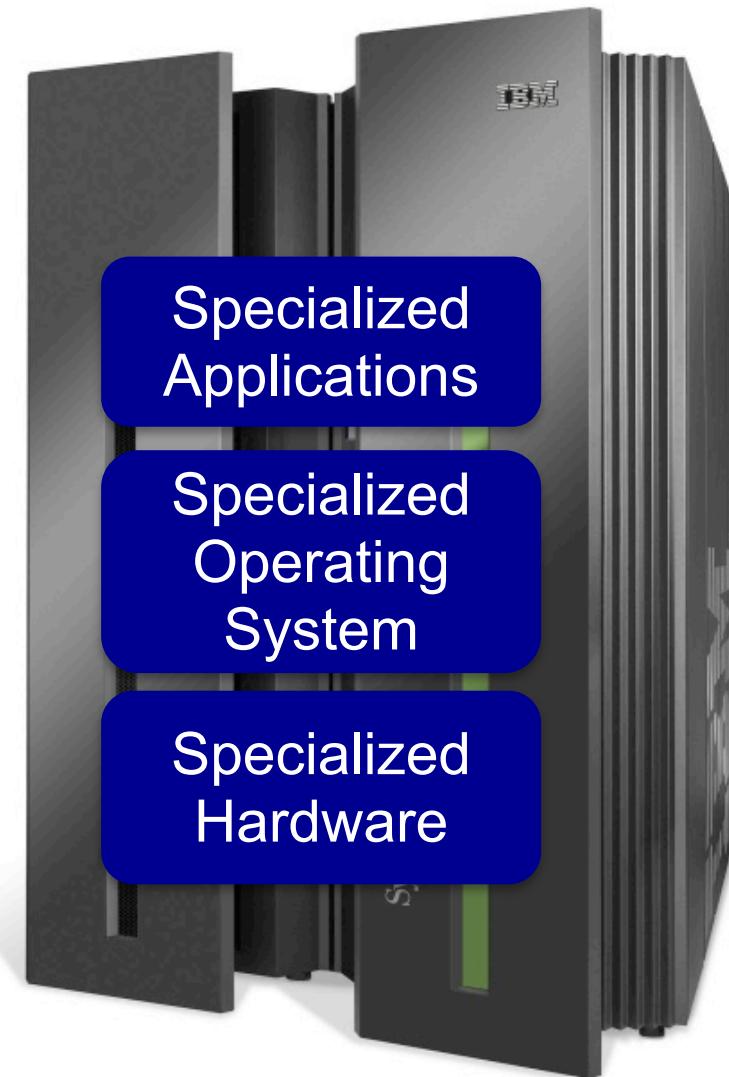
# Logically Centralized Controller



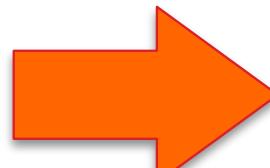
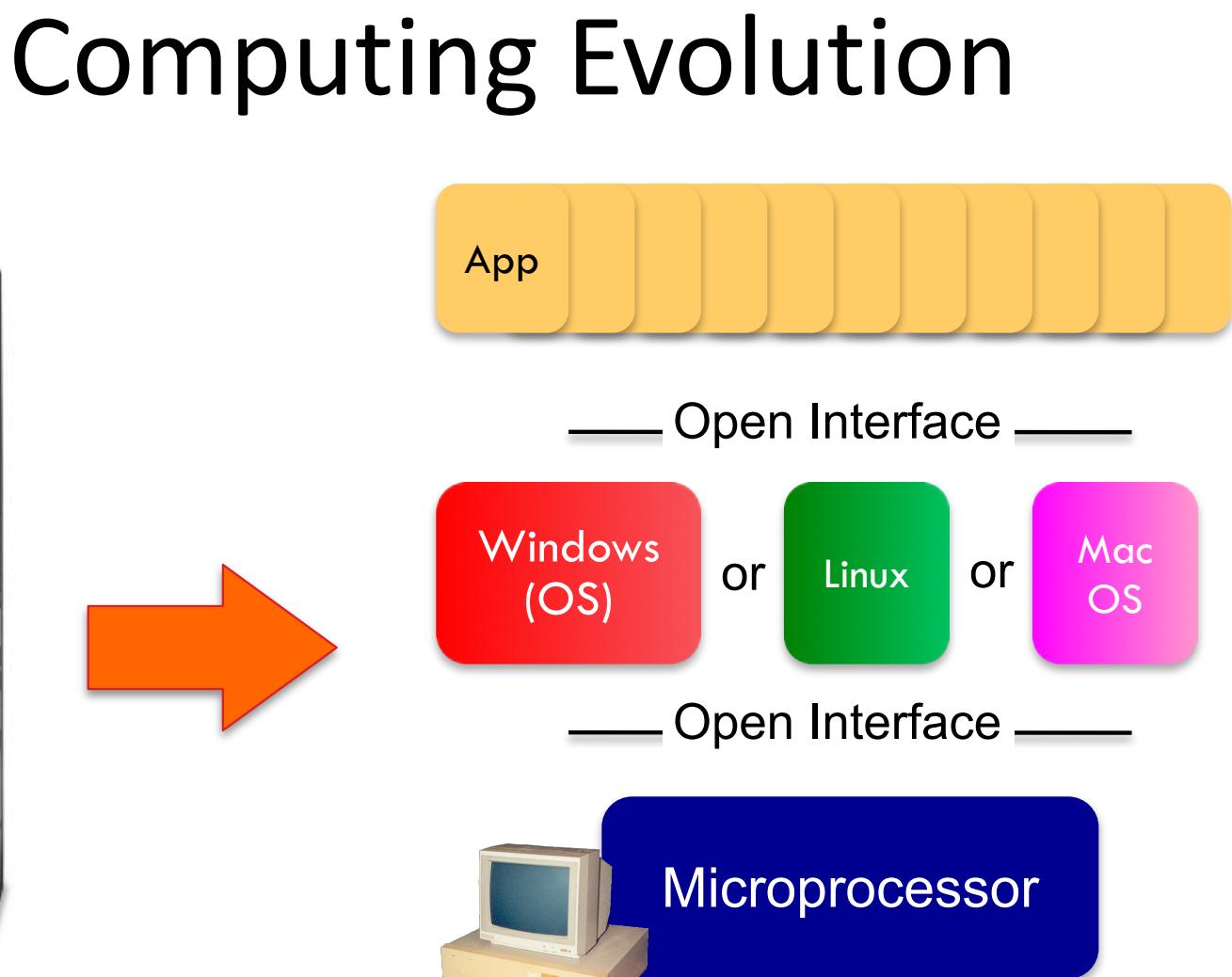
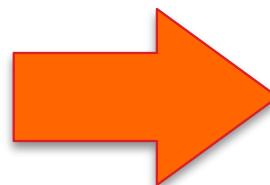
# Protocols → Applications



# Computing Evolution



Vertically integrated  
Closed, proprietary  
Slow innovation  
Small industry



Horizontal  
Open interfaces  
Rapid innovation  
Huge industry

# Networking Evolution

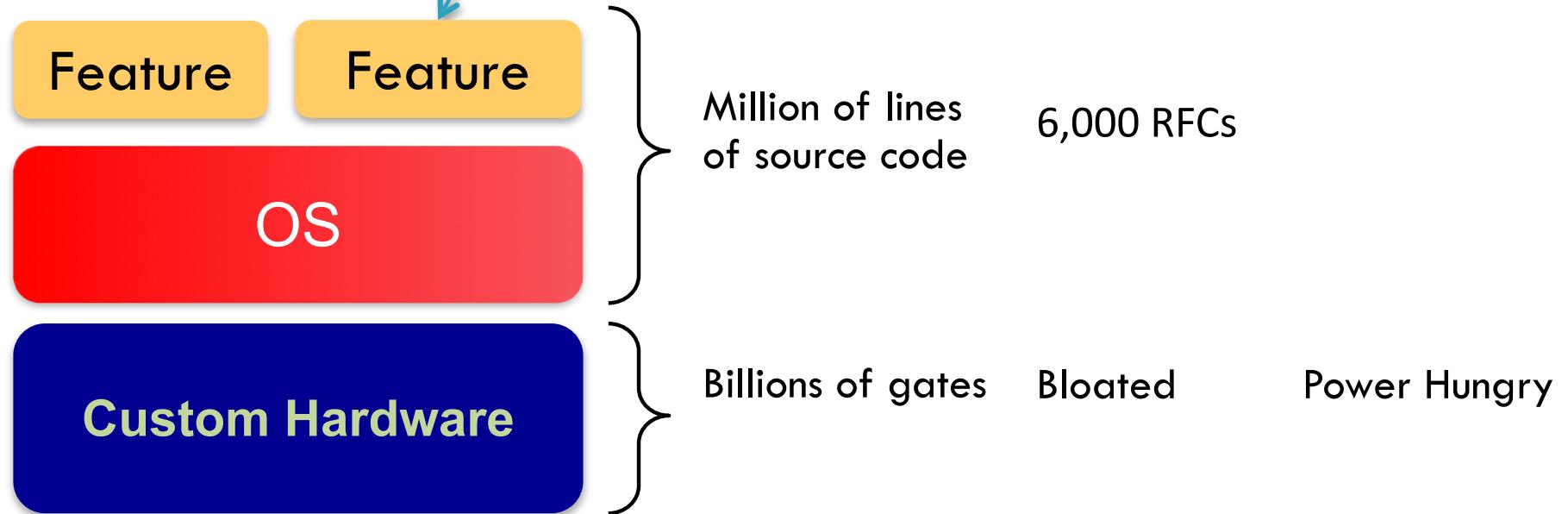


Vertically integrated  
Closed, proprietary  
Slow innovation

Horizontal  
Open interfaces  
Rapid innovation

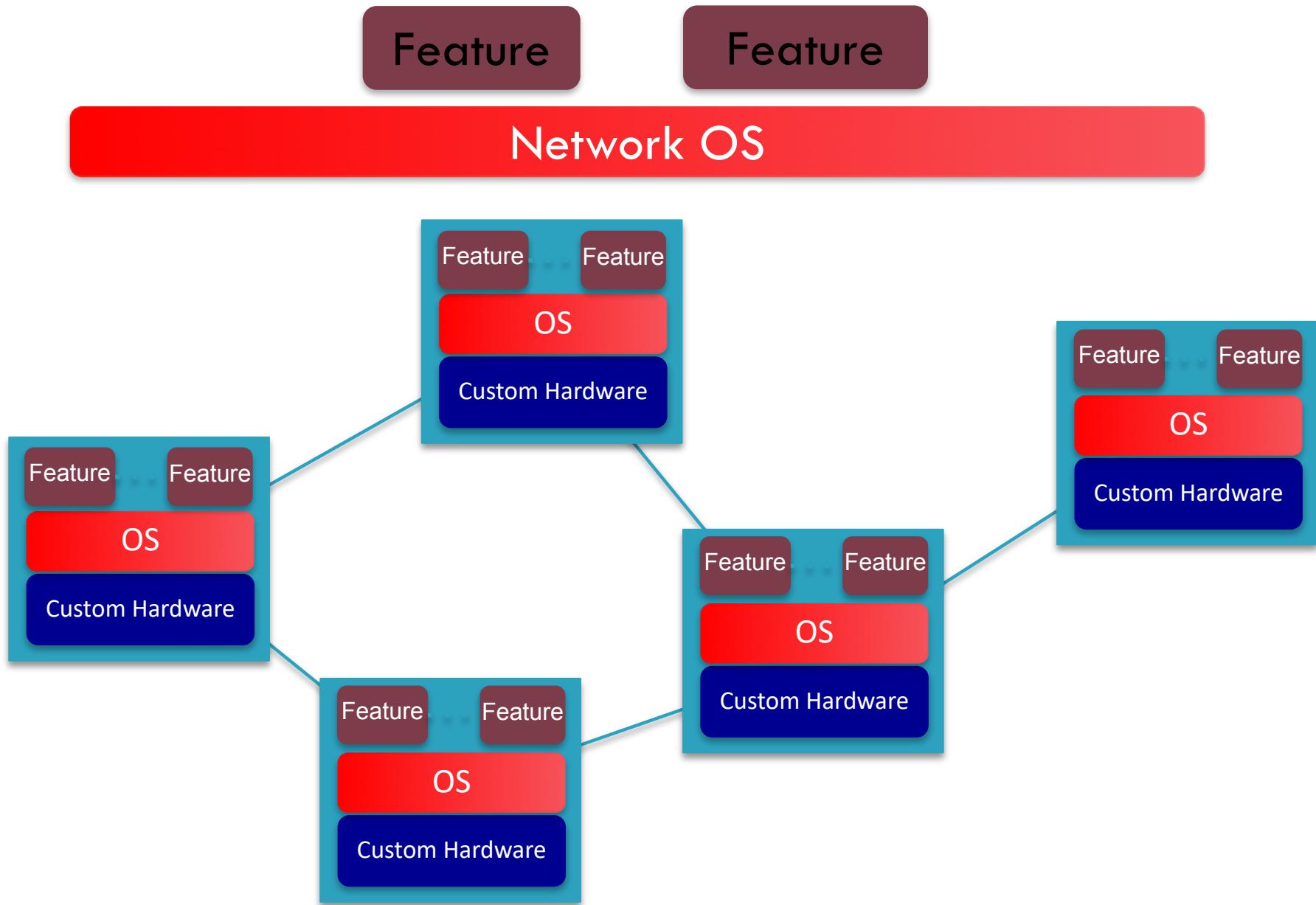


Routing, management, mobility management,  
access control, VPNs, ...



- Vertically integrated, complex, closed, proprietary
- Networking industry with “mainframe” mind-set

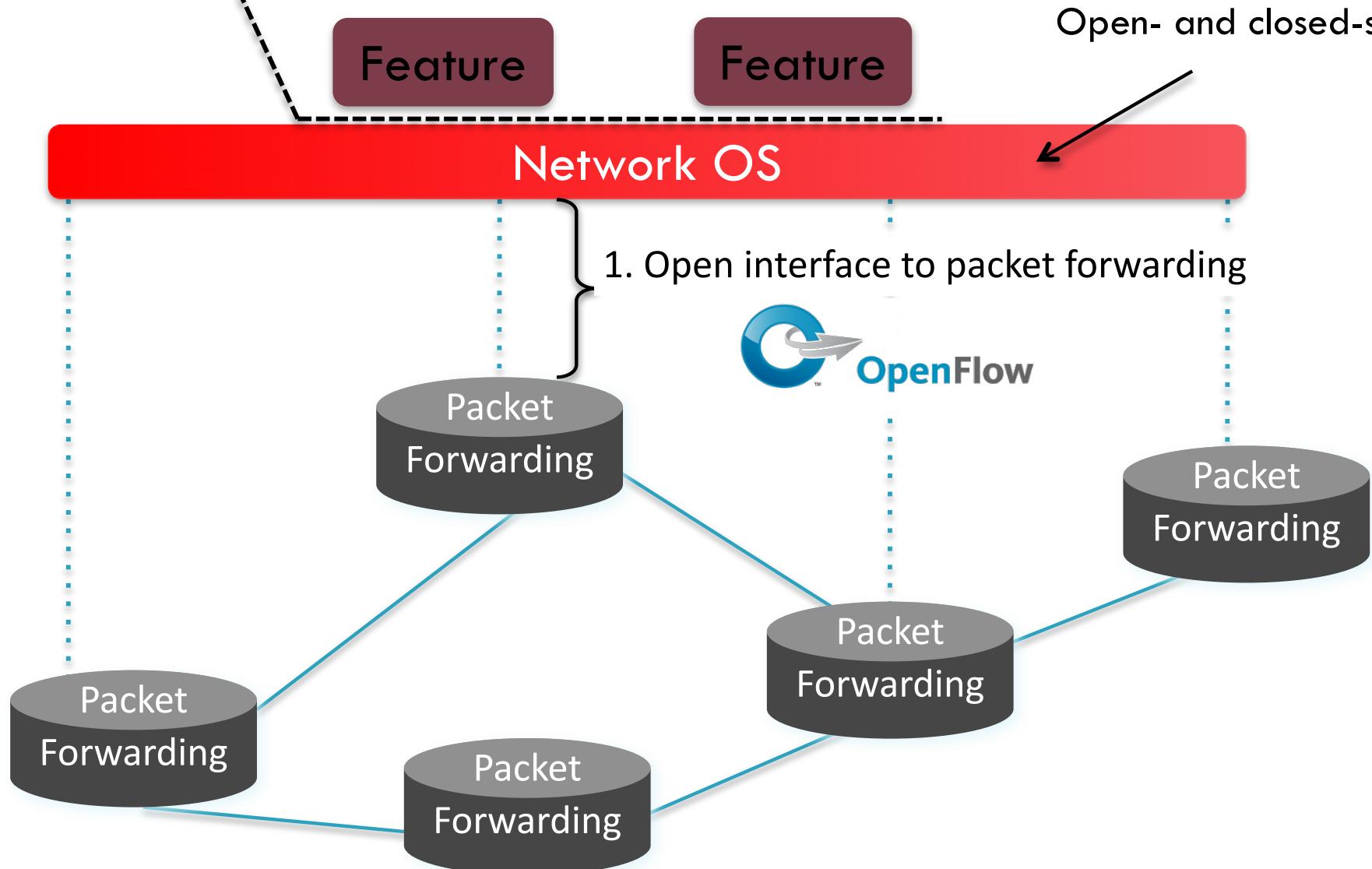
# The network is changing



# Software Defined Network (SDN)

3. Consistent, up-to-date global network view

2. At least one Network OS  
probably many.  
Open- and closed-source



# Network OS

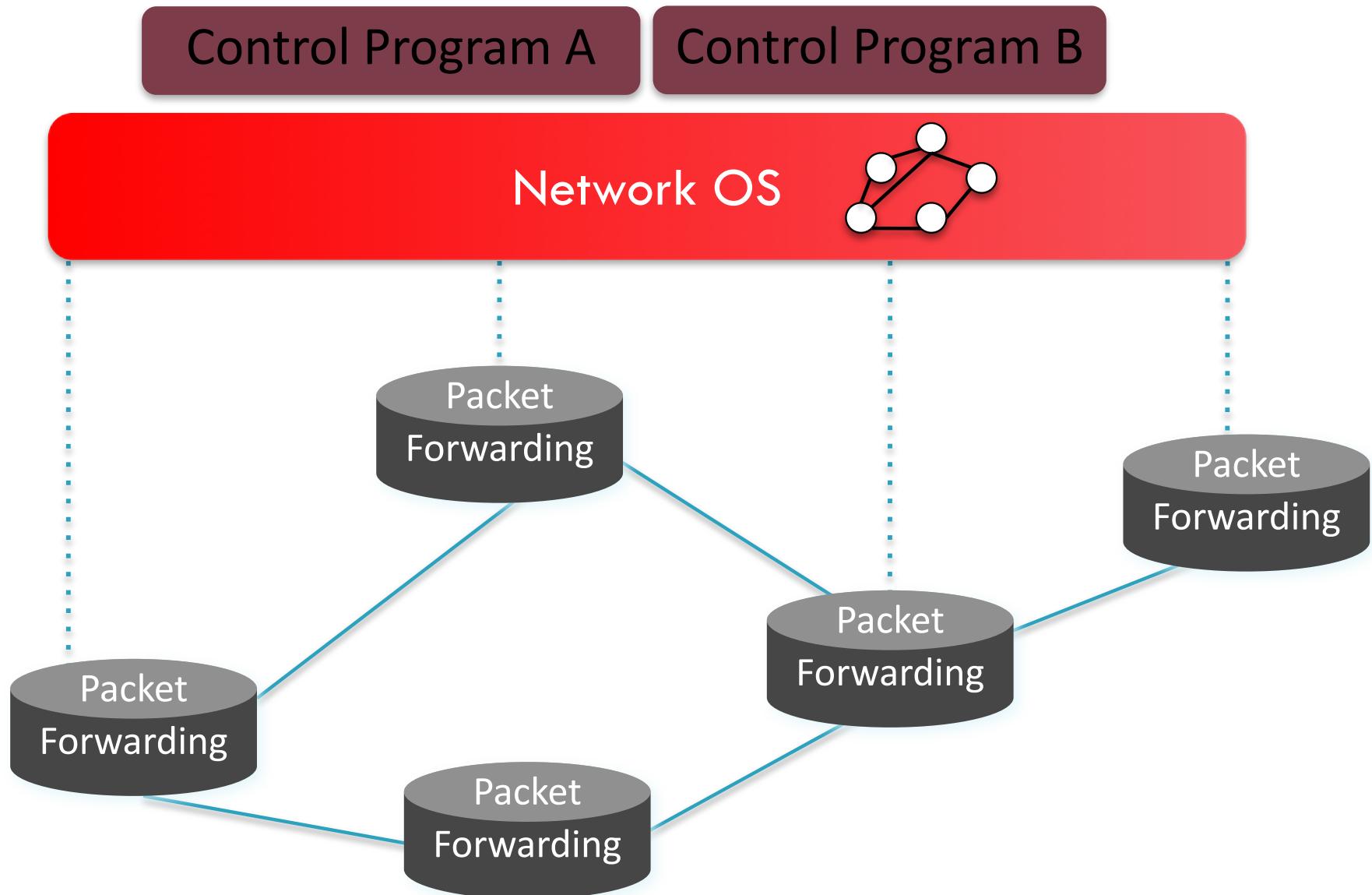
**Network OS:** distributed system that creates a consistent, up-to-date network view

- ❑ Runs on servers (controllers) in the network
- ❑ NOX, ONIX, Trema, Beacon, Maestro, ... + more

Uses forwarding abstraction to:

- ❑ Get state information **from** forwarding elements
- ❑ Give control directives **to** forwarding elements

# Software Defined Network (SDN)



# Control Program

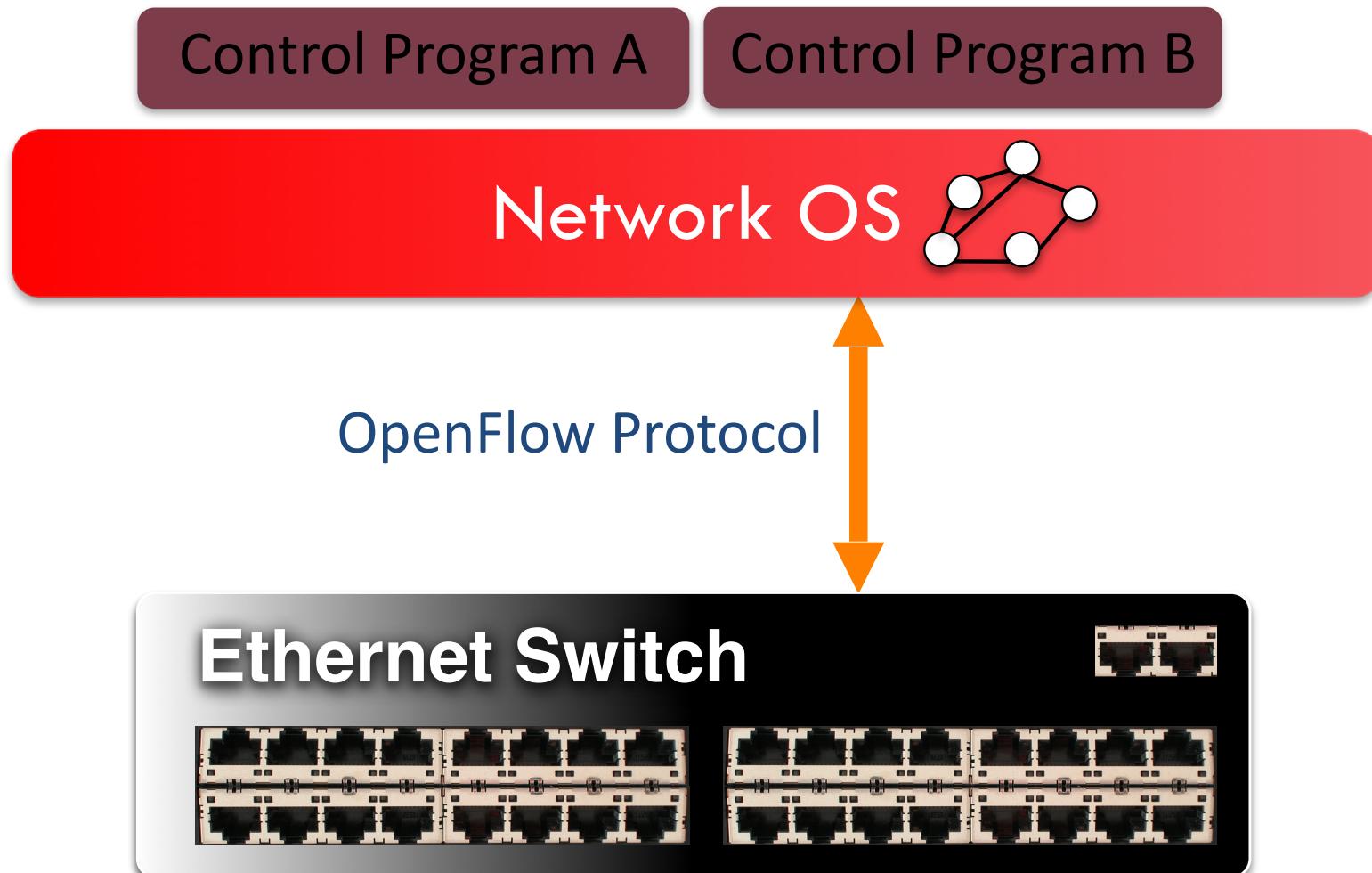
- Control program operates on view of network
  - ▣ **Input:** global network view (graph/database)
  - ▣ **Output:** configuration of each network device
- Control program is not a distributed system
  - ▣ Abstraction hides details of distributed state

# Forwarding Abstraction

**Purpose:** Abstract away forwarding hardware

- 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

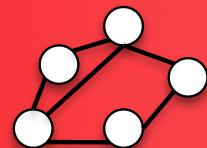


# OpenFlow Basics

Control Program A

Control Program B

Network OS



Packet  
Forwarding

Packet  
Forwarding

“If header =  $p$ , send to port 4”

“If header =  $q$ , overwrite header with  $r$ ,  
add header  $s$ , and send to ports 5,6”

“If header = ?, send to me”

Flow  
Table(s)

Packet  
Forwarding

# Plumbing Primitives

- Primitive is *<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

# General Forwarding Abstraction

Small set of primitives  
“Forwarding instruction set”

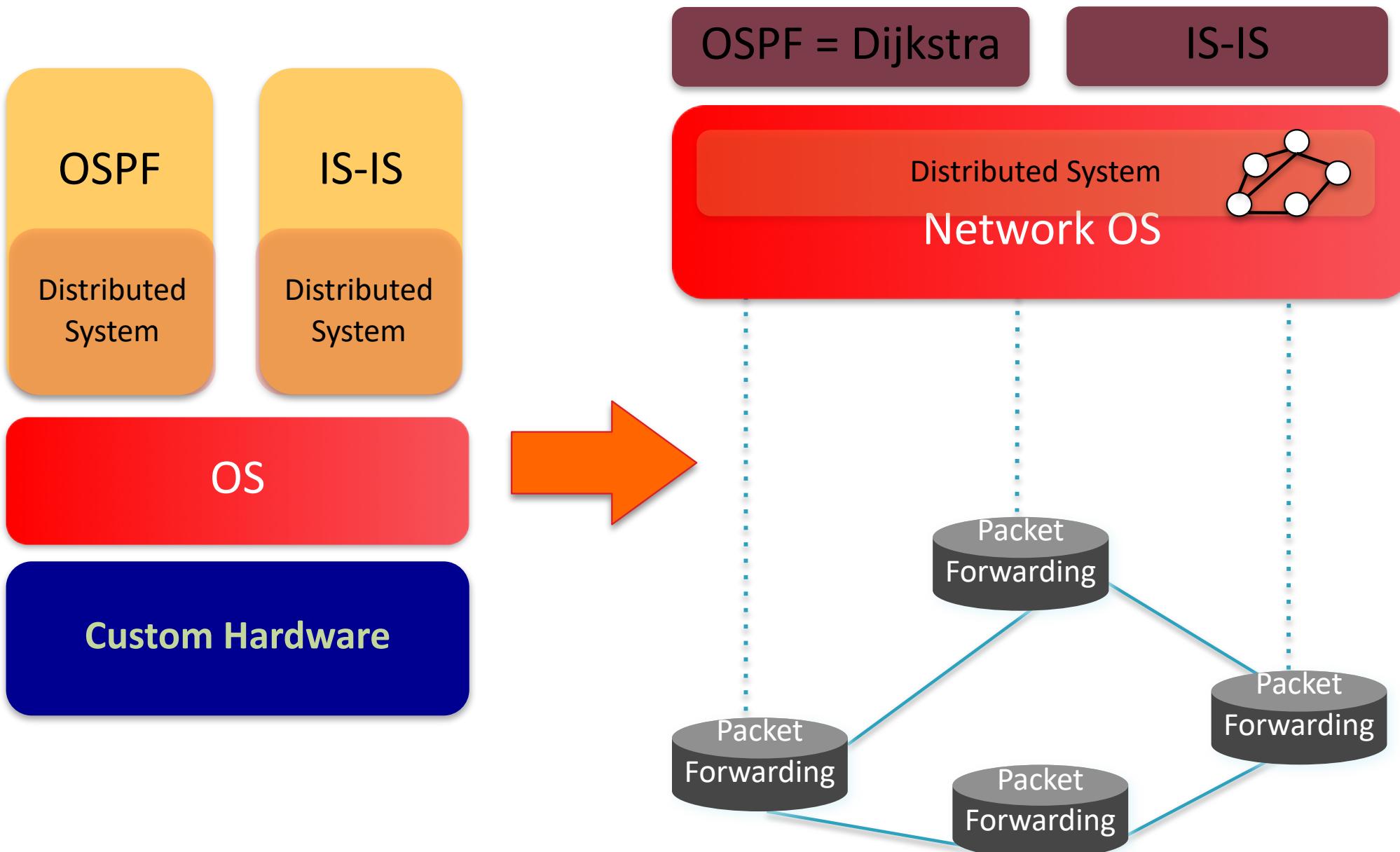
Protocol independent  
Backward compatible

Switches, routers, WiFi APs,  
basestations, TDM/WDM

# Example 1: OSPF and Dijkstra

- OSPF
  - RFC 2328: **245 pages**
- Distributed System
  - Builds consistent, up-to-date map of the network: **101 pages**
- Dijkstra's Algorithm
  - Operates on map: **4 pages**

# Example



Why SDN?

Great talk by Scott Shenker

<http://www.youtube.com/watch?v=WVs7Pc99S7w>

# Networking and Abstractions

- Networking is
  - ▣ “Intellectually Weak”
  - ▣ behind other fields
  - ▣ about the mastery of complexity

Good abstractions tame complexity

- ▣ Interfaces are instances of those abstractions

No abstraction => increasing complexity

- ▣ We are now at the complexity limit

# By comparison: Programming

- Machine languages: no abstractions
  - ▣ Had to deal with low-level details
- Higher-level languages: OS and other abstractions
  - ▣ File system, virtual memory, abstract data types, ...
- Modern languages: even more abstractions
  - ▣ Object orientation, garbage collection,...

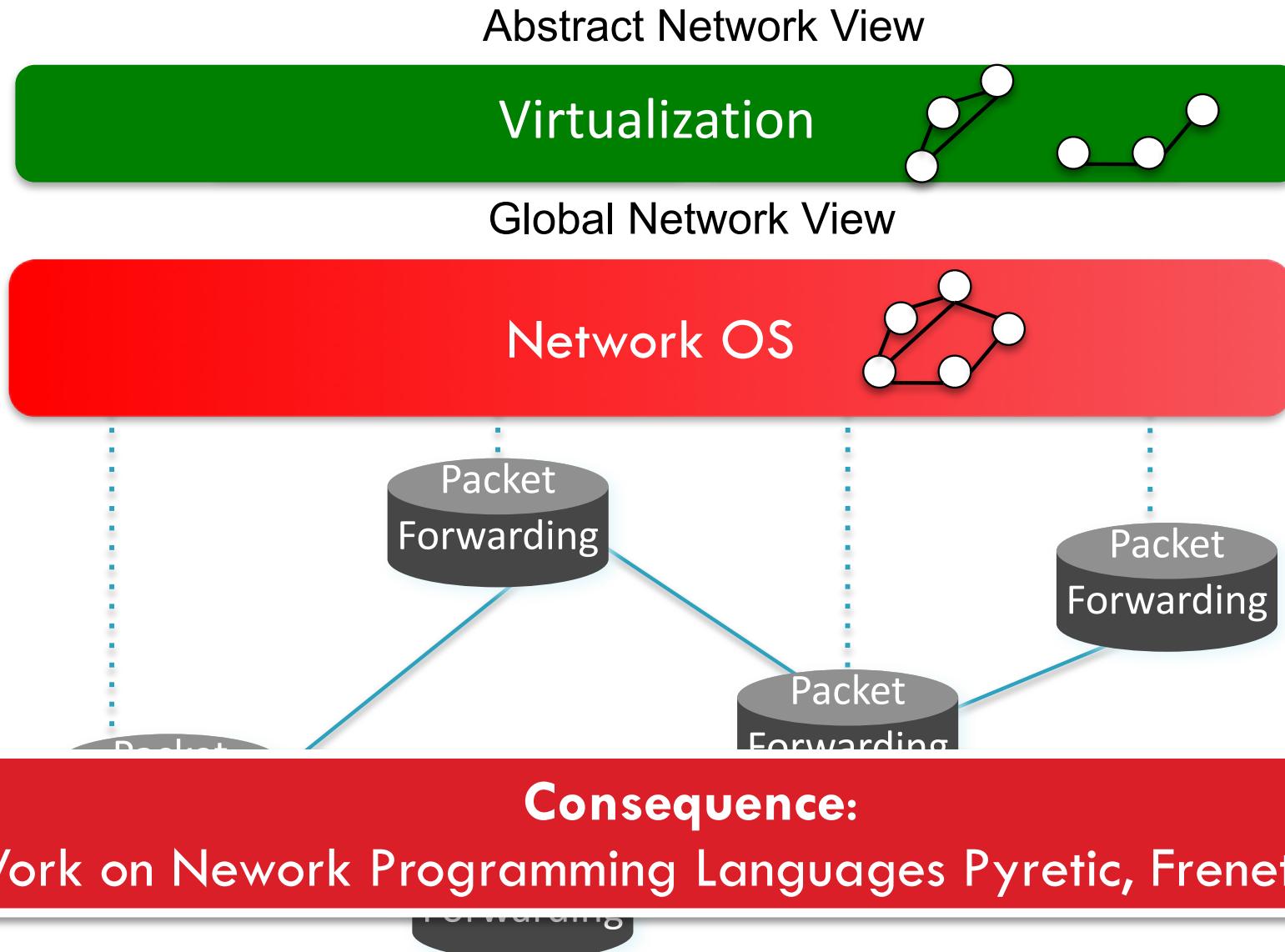
# Programming Analogy

- What if programmers had to:
  - ▣ Specify where each bit was stored
  - ▣ Explicitly deal with internal communication errors
  - ▣ Within a programming language with limited expressibility
  
- Programmers would redefine problem by:
  - ▣ Defining higher level abstractions for memory
  - ▣ Building on reliable communication primitives
  - ▣ Using a more general language

# Specification Abstraction

- Network OS eases implementation
  - ▣ E.g., Helps manage distributed state
- Next step is to ease specification
  - ▣ E.g., How do you specify what the system should do?
- Key goals
  - ▣ Provide abstract view of network map
  - ▣ Control program operates on abstract view
  - ▣ Develop means to simplify specification

# Software Defined Network (SDN)



# Open Networking Foundation (ONF)

New non-profit standards organization (Mar 2011)

Defining standards for SDN, starting with OpenFlow

Board of Directors

Google, Facebook, Microsoft, Yahoo, DT, Verizon

39 Member Companies

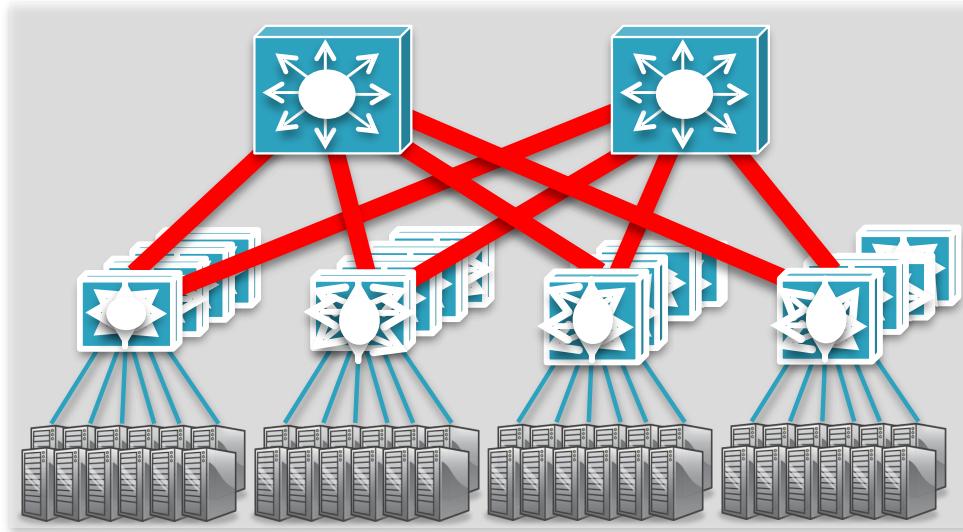
Cisco, VMware, IBM, Juniper, HP, Broadcom, Citrix, NTT, Intel,  
Ericsson, Dell, Huawei, ...

# Telco Operators

- Global IP traffic growing 40-50% per year
- End-customer monthly bill remains unchanged
- Therefore, CAPEX and OPEX need to reduce 40-50% per Gb/s per year
- But in practice, reduces by ~20% per year

SDN enables industry to reduce OPEX and CAPEX  
...and to create new differentiating services

# Example: New Data Center



## Cost

200,000 servers

Fanout of 20 → 10,000 switches

\$5k vendor switch = \$50M

\$1k commodity switch = \$10M

Savings in 10 data centers = **\$400M**

## Control

More flexible control

Tailor network for services

Quickly improve and innovate

# Summary

- Networks becoming
  - More programmatic
  - Defined by owners and operators, not vendors
  - Faster changing, to meet operator needs
  - Lower opex, capex and power
- Abstractions
  - Will shield programmers from complexity
  - Make behavior more provable