Software Defined Networks

- A quick overview
  - Based primarily on the presentations of Prof. Scott Shenker of UC Berkeley

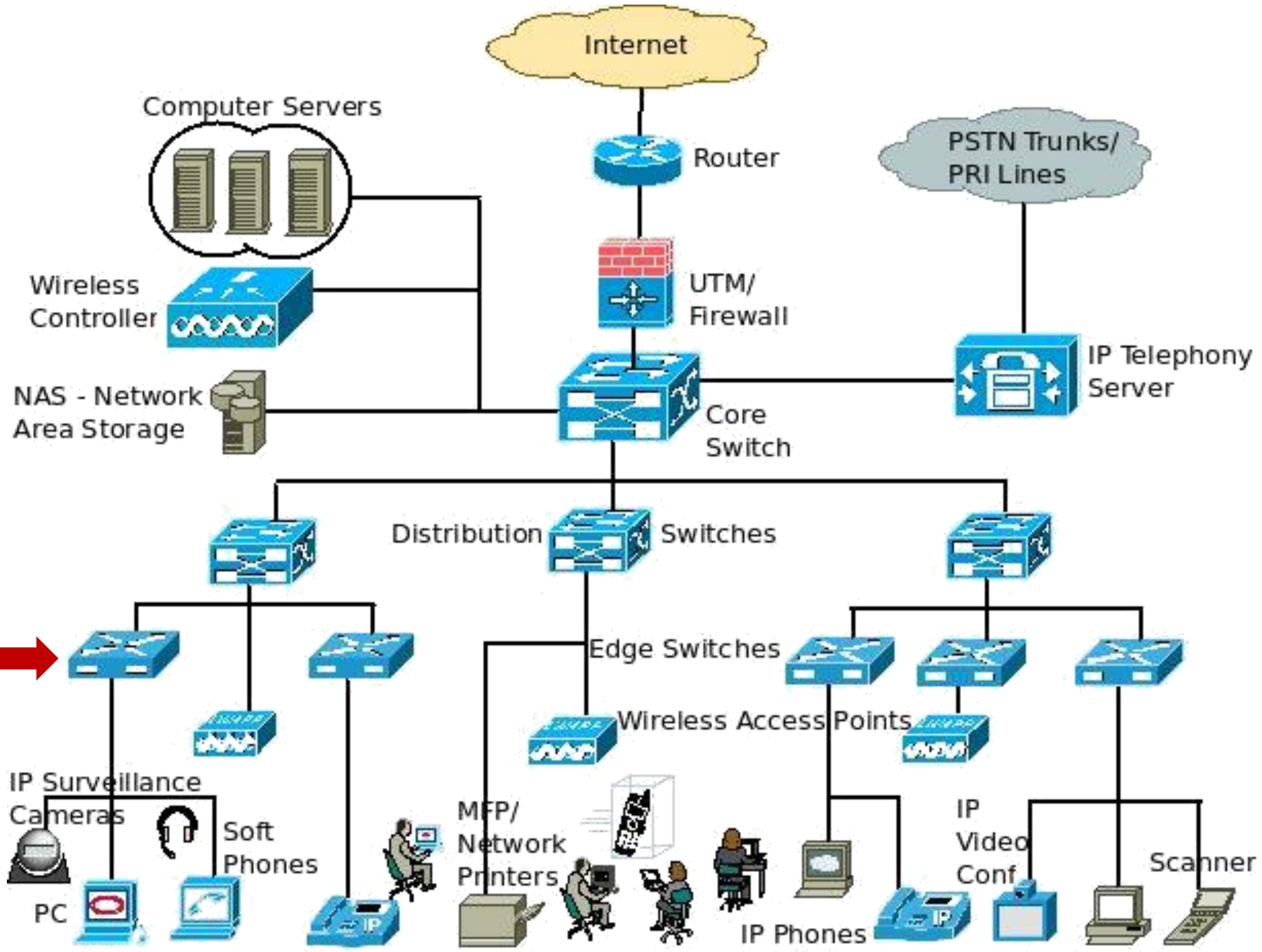
"The Future of Networking, and the Past of Protocols"

- Please watch the YouTube video of Shenker's talk
- with a short intro to Openflow basics at the end

Two Key Definitions

- **Data Plane**: processing and delivery of packets
  - Based on state in routers and endpoints
  - E.g., IP, TCP, Ethernet, etc.
  - Fast timescales (per-packet)
- **Control Plane**: establishing the state in routers
  - Determines how and where packets are forwarded
  - Routing, traffic engineering, firewall state, …
  - Slow time-scales (per control event)
- These different planes require *different abstractions*

Limitations of Current Networks



Switches

http://www.excitingip.net/27/a-basic-enterprise-lan-network-architecture-block-diagram-and-components/

Limitations of Current Networks

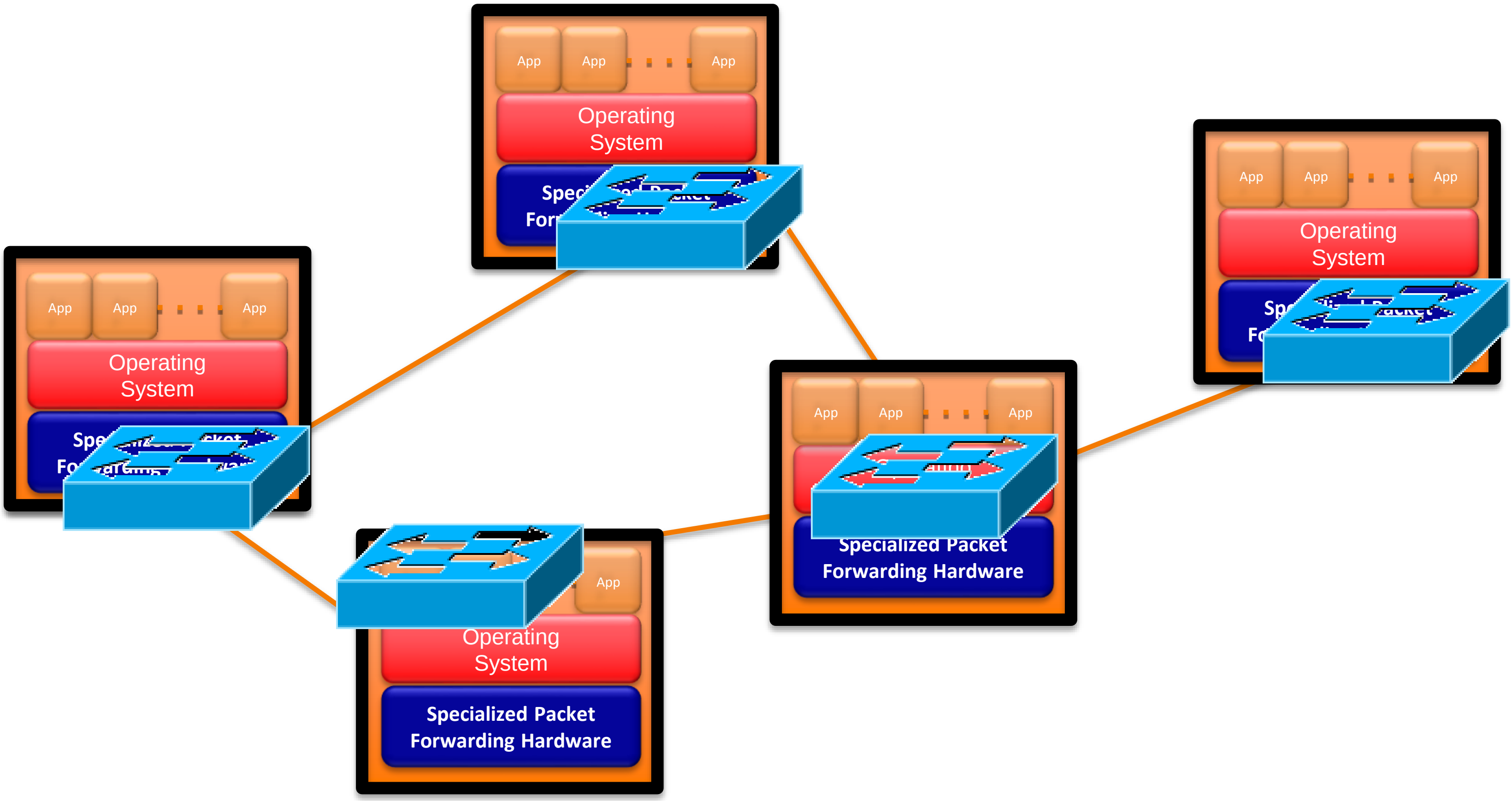
- **Enterprise networks are difficult to manage**

- **“New control requirements have arisen”:**
  - Greater scale
  - Migration of VMS

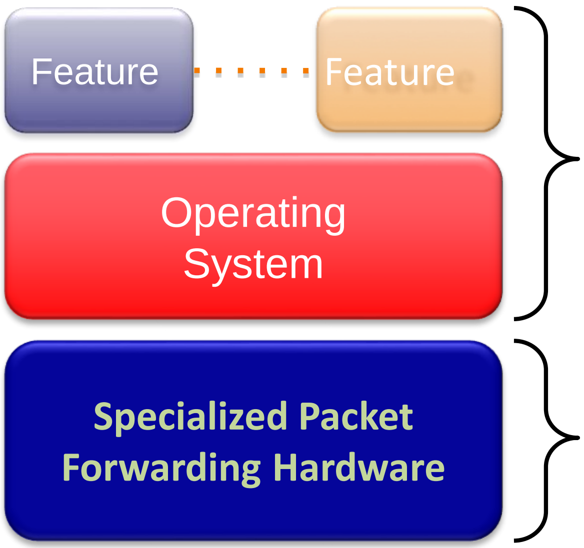
- **How to easily configure huge networks?**

Limitations of Current Networks

- Old ways to configure a network



Limitations of Current Networks



Million of lines of source code

Billions of gates

Many complex functions baked into infrastructure

OSPF, BGP, multicast, differentiated services, Traffic Engineering, NAT, firewalls, …

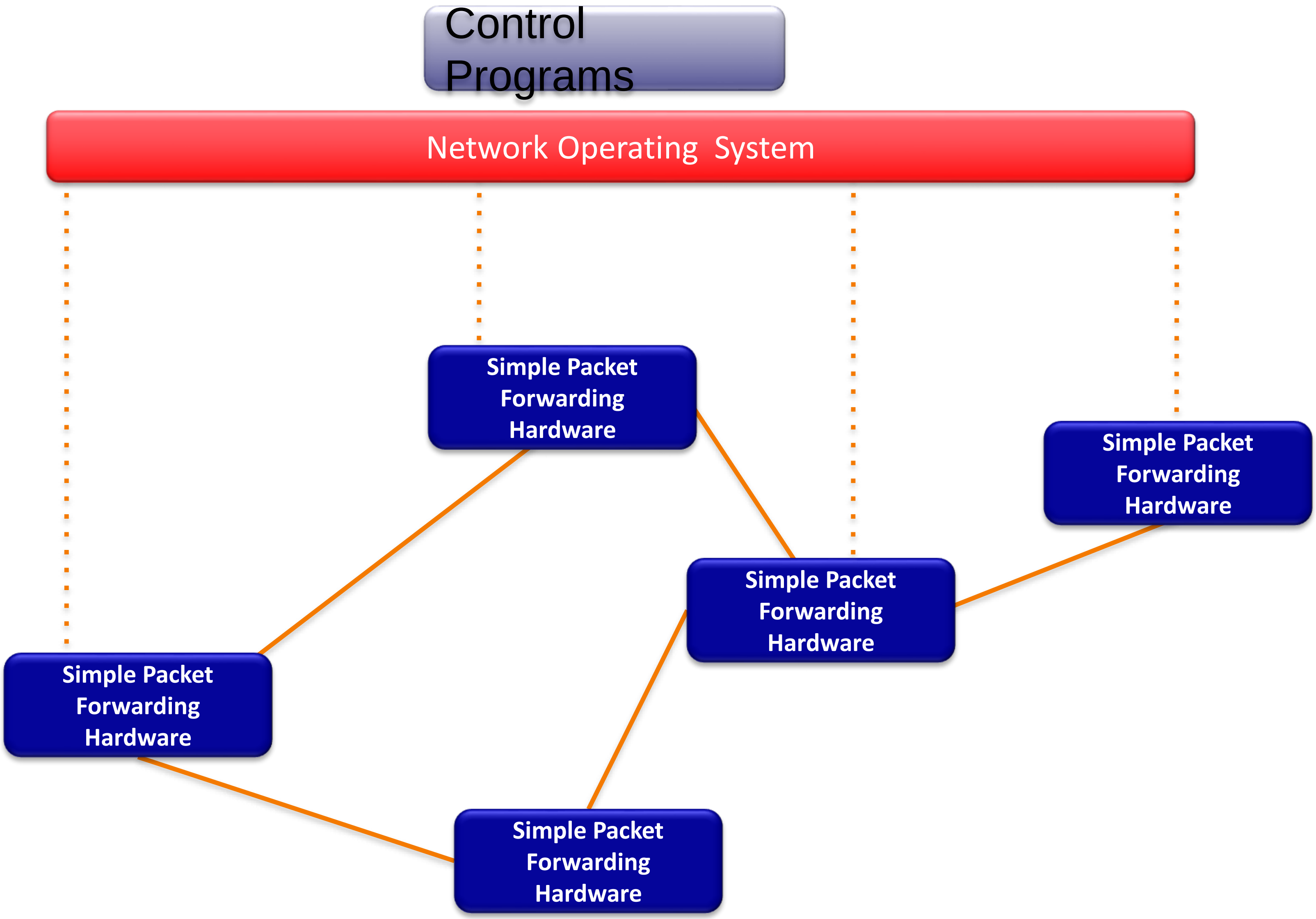
Cannot dynamically change according to network conditions

Limitations of Current Networks

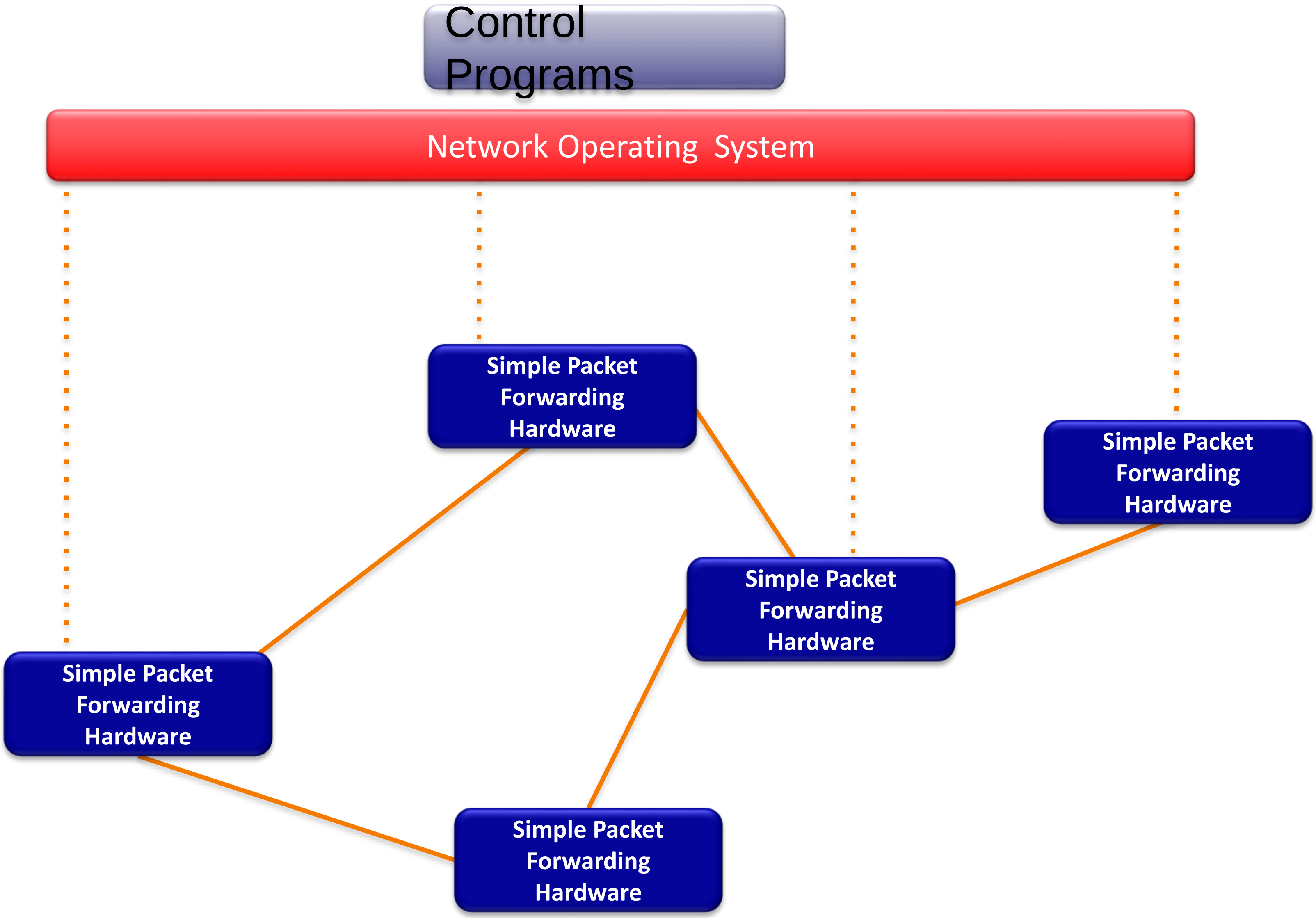
- No control plane abstraction for the whole network!
- It's like old times – when there was no OS…

Wilkes with the EDSAC, 1949

Idea: An OS for Networks



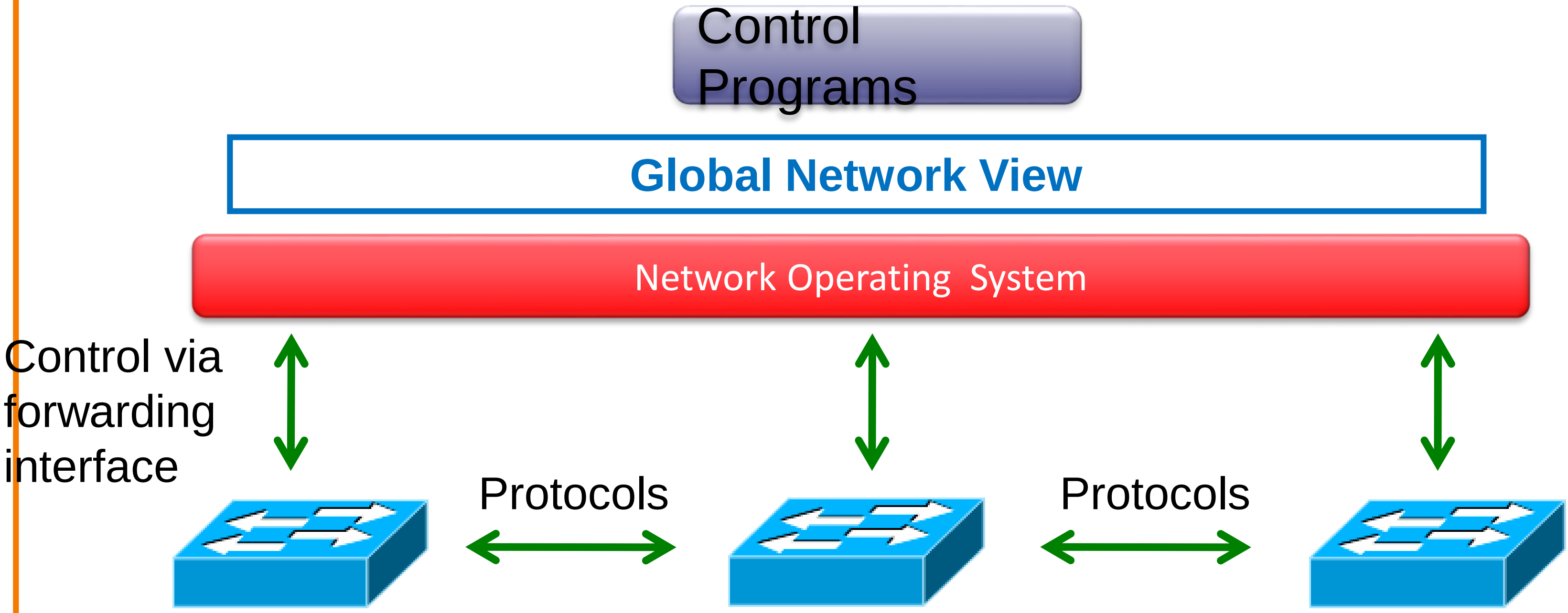
Idea: An OS for Networks



Idea: An OS for Networks

- **"NOX: Towards an Operating System for Networks"**

Software-Defined Networking (SDN)



Software Defined Networking

- **No longer designing distributed control protocols**
- **Much easier to write, verify, maintain, …**
  – An interface for programming
- **NOS serves as fundamental control block**
  – With a global view of network

Software Defined Networking

- **Questions:**
  - How to obtain global information?
  - What are the configurations?
  - How to implement?
  - How is the scalability?
  - How does it really work?

A Short History of SDN

- ~2004: Research on new management paradigms
  - RCP, 4D [Princeton, CMU,….]
  - SANE, Ethane [Stanford/Berkeley]

- 2008: Software-Defined Networking (SDN)
  - NOX Network Operating System [Nicira]
  - OpenFlow switch interface [Stanford/Nicira]

- 2011: Open Networking Foundation (~69 members)
  - **Board**: Google, Yahoo, Verizon, DT, Msoft, F'book, NTT
  - **Members**: Cisco, Juniper, HP, Dell, Broadcom, IBM,…..

- 2012: Latest Open Networking Summit
  - Almost 1000 attendees, Google: SDN used for their WAN
  - Commercialized, in production use (few places)

The Future of Networking,

and the Past of Protocols

Scott Shenker

Key to Internet Success: Layers

Applications

…built on…

Reliable (or unreliable) transport

…built on…

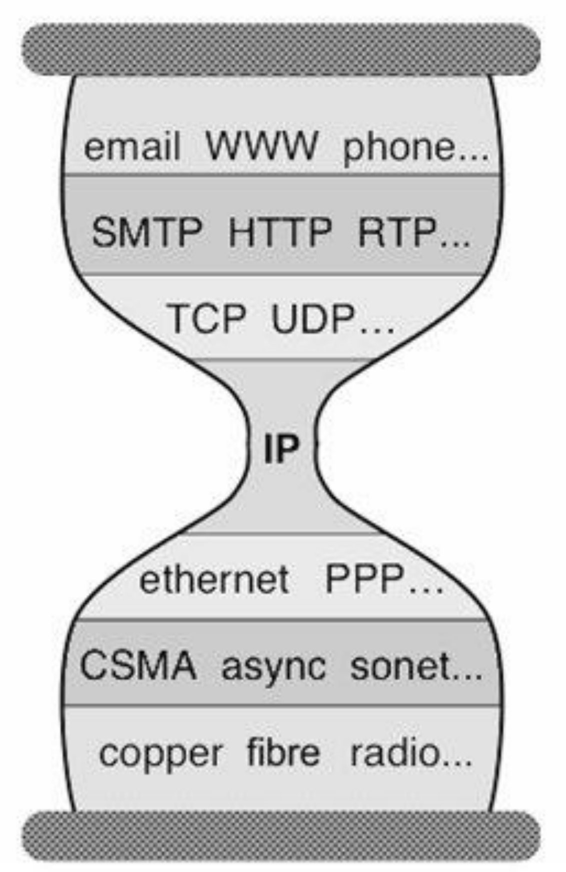
Best-effort global packet delivery

…built on…

Best-effort local packet delivery

…built on…

Physical transfer of bits



Why Is Layering So Important?

- Decomposed delivery into fundamental components
- Independent but compatible **innovation** at each layer
- A practical success of unprecedented proportions…
- **…but an academic failure**

Built an Artifact, Not a Discipline

- Other fields in “systems”: OS, DB, DS, etc.
  - Teach basic principles
  - Are easily managed
  - Continue to evolve
- Networking:
  - Teach big bag of protocols
  - Notoriously difficult to manage
  - Evolves very slowly

Why Does Networking Lag Behind?

- Networks used to be simple: Ethernet, IP, TCP….
- New **control** requirements led to great complexity
  - Isolation ➔ VLANs, ACLs
  - Traffic engineering ➔ MPLS, ECMP, Weights
  - Packet processing ➔ Firewalls, NATs, middleboxes
  - Payload analysis ➔ Deep packet inspection (DPI)
  - …..
- Mechanisms designed and deployed independently
  - Complicated "control plane" design, primitive functionality
  - Stark contrast to the elegantly modular "data plane"

Infrastructure Still Works!

- **Only** because of “our” ability to master complexity

- This ability to master complexity is both a blessing…
  - **…and a curse!**

A Better Example: Programming

- Machine languages: no abstractions
  - Mastering complexity was crucial
- Higher-level languages: OS and other abstractions
  - File system, virtual memory, abstract data types, ...
- Modern languages: even more abstractions
  - Object orientation, garbage collection,…

Abstractions key to extracting simplicity

“The Power of Abstraction”

“Modularity based on abstraction is the way things get done”

–

Barbara Liskov

Abstractions → Interfaces → Modularity

What abstractions do we have in networking?

Abstractions ~ Problem Decomposition

- Decompose problem into basic components (tasks)
- Define an abstraction for each component
- Implementation of abstraction can focus on one task
- If tasks still too hard to implement, return to step 1

Layers are Great Abstractions

- Layers only deal with the **data plane**
- We have no powerful ***control plane*** abstractions!
- How do we find those control plane abstractions?
- Two steps: ***define*** problem, and then ***decompose*** it.

The Network Control Problem

- Compute the configuration of each physical device
  - E.g., Forwarding tables, ACLs,…
- Operate without communication guarantees
- Operate within given network-level protocol

Only people who love complexity would find this a reasonable request

Programming Analogy

- What if programmers had to:
  - Specify where each bit was stored
  - Explicitly deal with all internal communication errors
  - Within a programming language with limited expressability

- Programmers would redefine problem:
  - Define a higher level abstraction for memory
  - Build on reliable communication abstractions
  - Use a more general language

- **Abstractions** divide problem into tractable pieces
  - And make programmer's task easier

From Requirements to Abstractions

1. Operate without communication guarantees

   Need an abstraction for **distributed state**

2. Compute the configuration of each physical device

   Need an abstraction that **simplifies configuration**

3. Operate within given network-level protocol

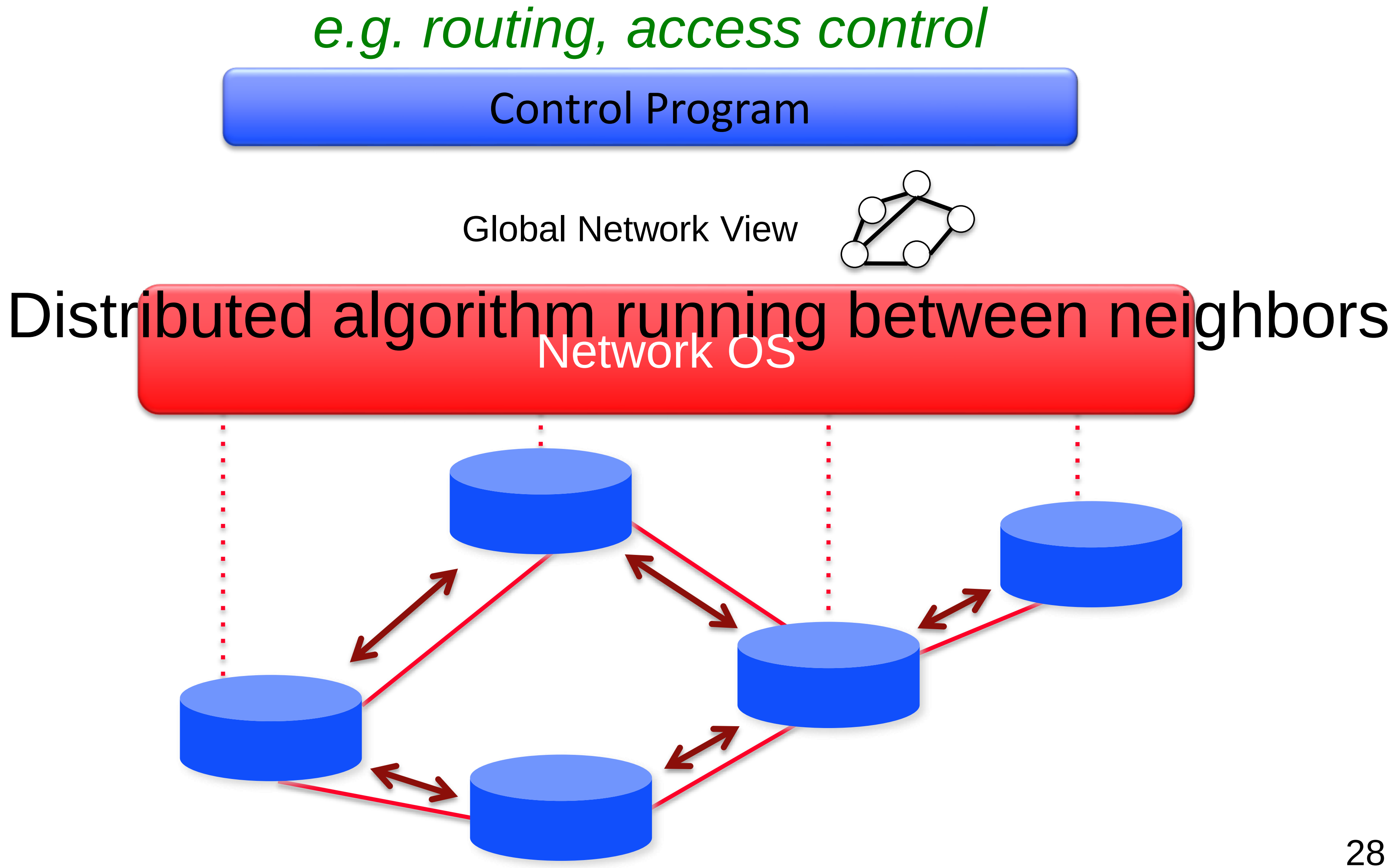
   Need an abstraction for general **forwarding model**

Once these abstractions are in place, control mechanism has a much easier job!

1. Distributed State Abstraction

- Shield control mechanisms from state distribution
  - While allowing access to this state
- Natural abstraction: ***global network view***
  - Annotated network graph provided through an API
- Implemented with "Network Operating System"
- Control mechanism is now program using API
  - No longer a distributed protocol, now just a graph algorithm
  - E.g. Use Dijkstra rather than Bellman-Ford

Network of Switches and/or Routers
Software Defined Network (SDN)
Traditional Control Mechanisms



Major Change in Paradigm

- No longer designing distributed control protocols
  - Design one distributed system (NOS)
  - Use for all control functions

- Now just defining a centralized control ***function***

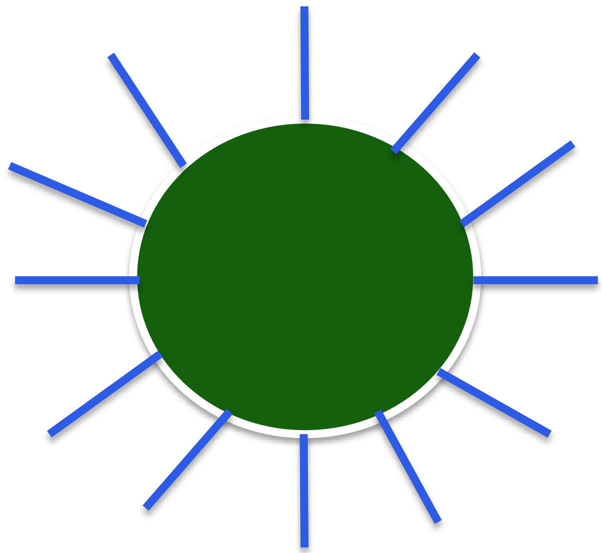
Configuration = Function(view)

- If you understand this, raise your hand.

2. Specification Abstraction

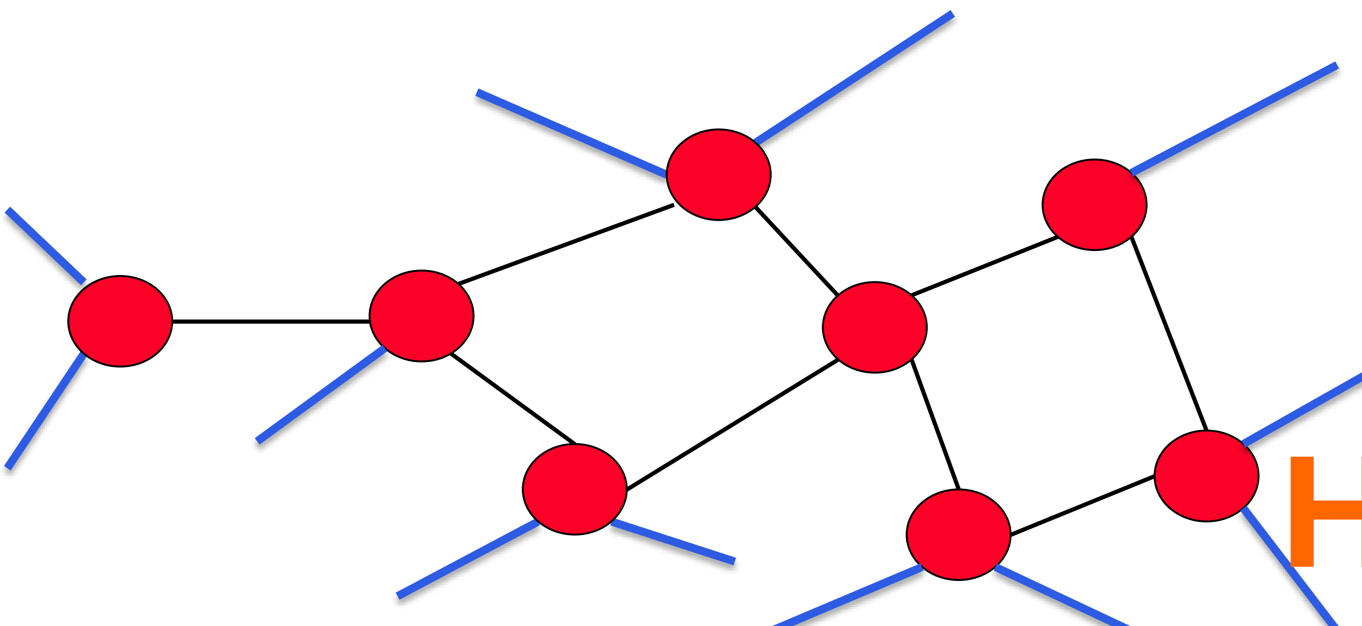
- Control program should express desired behavior
- It should not be responsible for implementing that behavior on physical network infrastructure
- Natural abstraction: **simplified model** of network
  - Simple model with only enough detail to specify goals
- Requires a new shared control layer:
  - **Map abstract configuration to physical configuration**
- This is “network virtualization”

Simple Example: Access Control



What

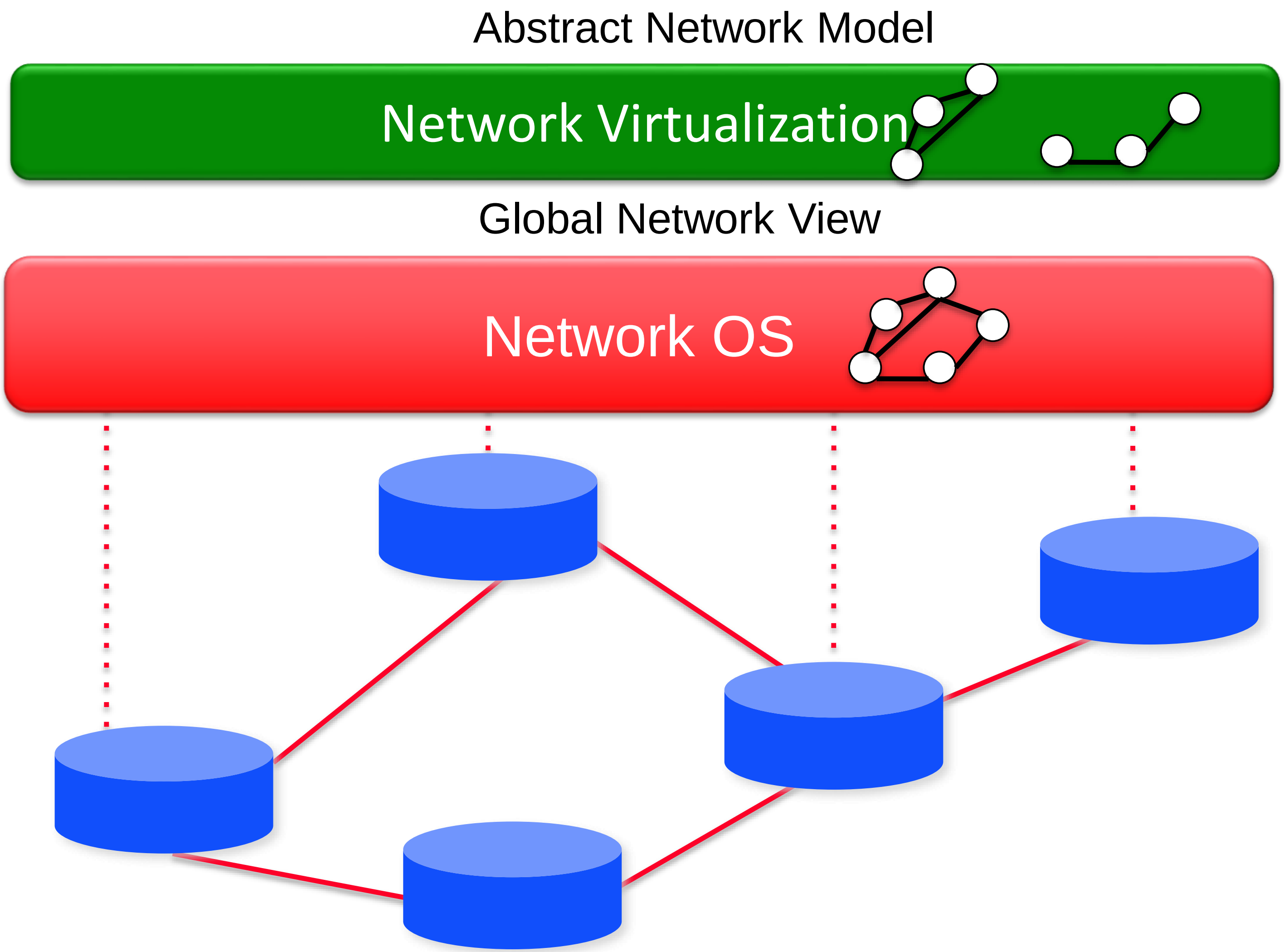
Abstract
Network
Model



Global
Network
View

How

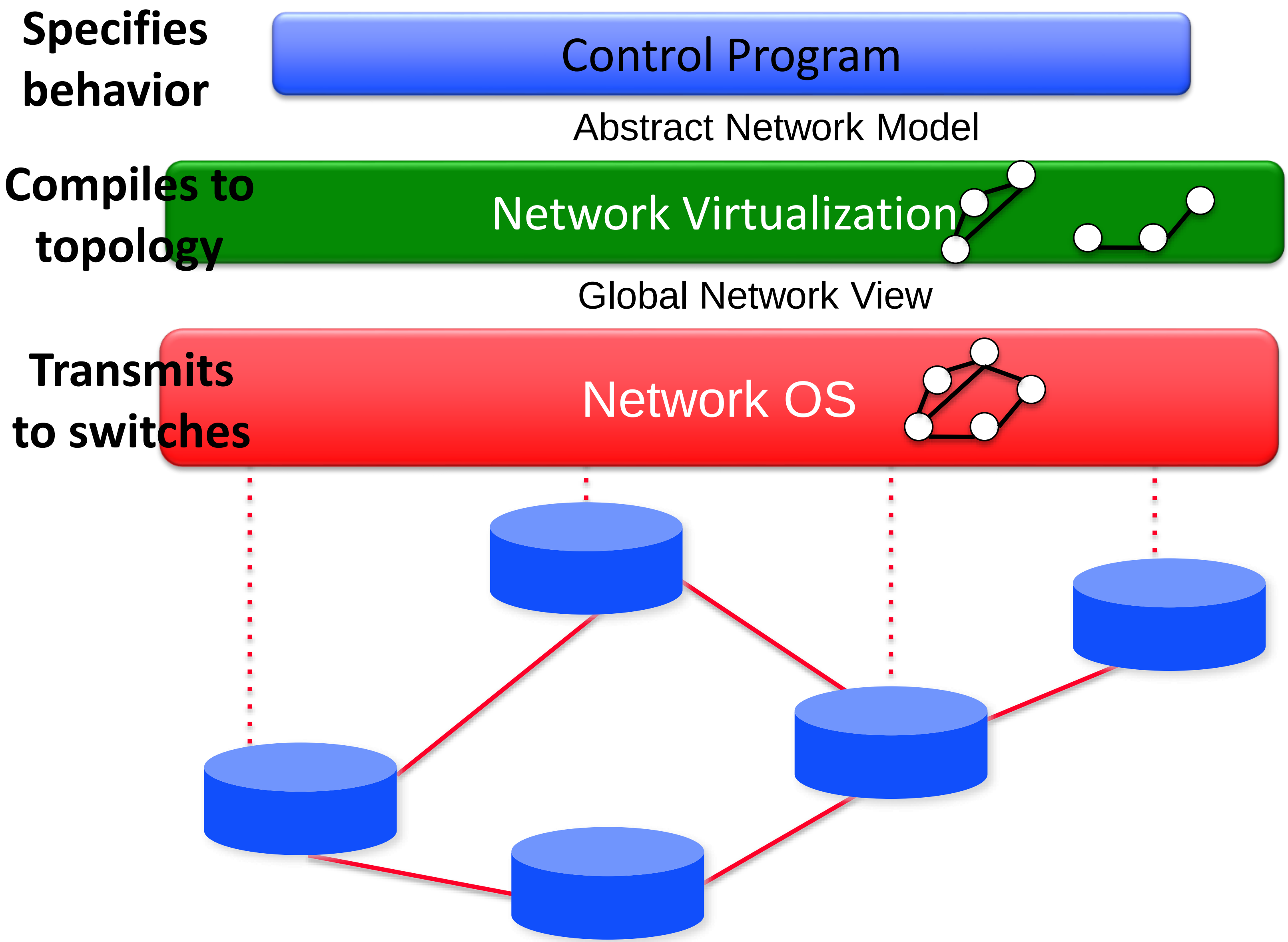
Software Defined Network: Take 2



What Does This Picture Mean?

- Write a simple program to configure a simple model
  - Configuration merely a way to specify what you want
- Examples
  - ACLs: who can talk to who
  - Isolation: who can hear my broadcasts
  - Routing: only specify routing to the degree you care
    - Some flows over satellite, others over landline
  - TE: specify in terms of quality of service, not routes
- Virtualization layer “compiles” these requirements
  - Produces suitable configuration of actual network devices
- NOS then transmits these settings to physical boxes

Software Defined Network: Take 2



Two Examples Uses

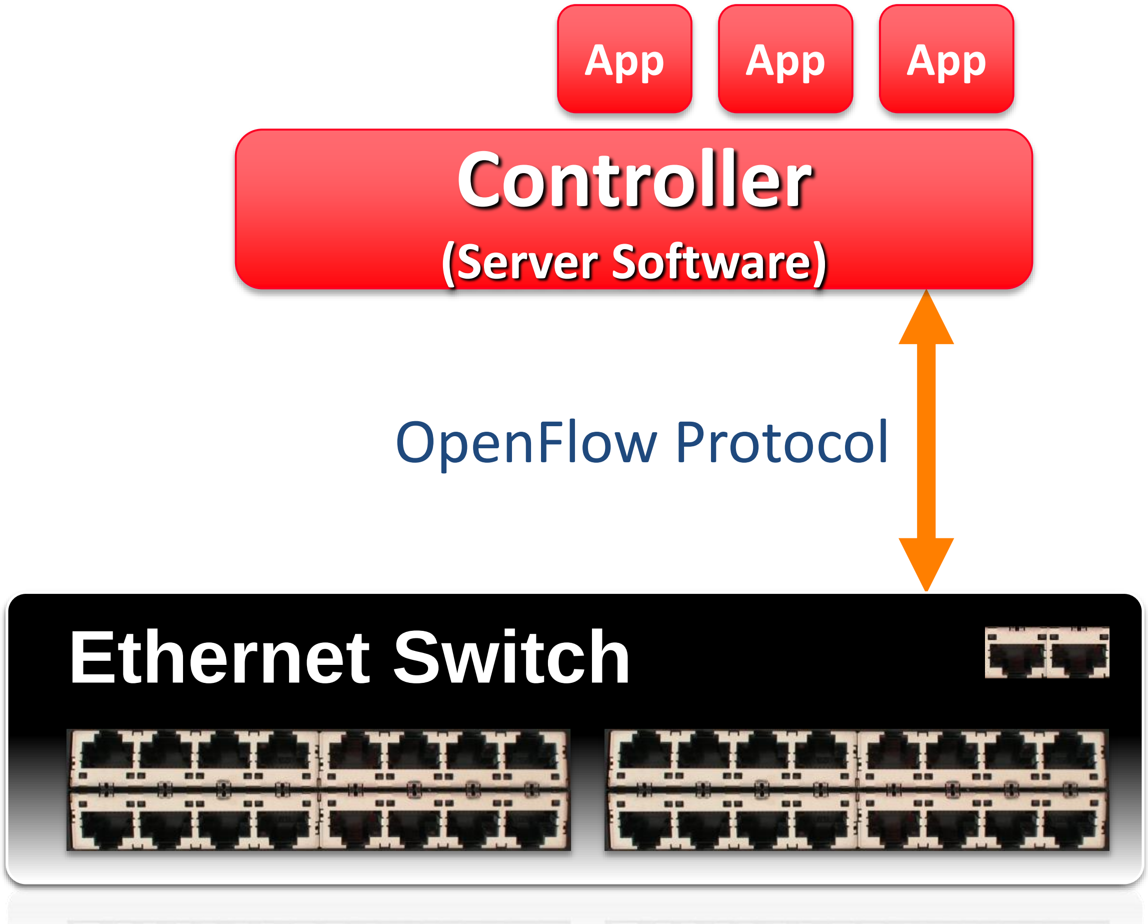
- Scale-out router:
  - Abstract view is single router
  - Physical network is collection of interconnected switches
  - Allows routers to “scale out, not up”
  - Use standard routing protocols on top

- Multi-tenant networks:
  - Each tenant has control over their “private” network
  - Network virtualization layer compiles all of these individual control requests into a single physical configuration

- **Hard to do without SDN, easy** *(in principle)* **with SDN**

3. Forwarding Abstraction

- Switches have two “brains”
  - Management CPU (smart but slow)
  - Forwarding ASIC (fast but dumb)
- Need a forwarding abstraction for both
  - CPU abstraction can be almost anything
- ASIC abstraction is much more subtle: **OpenFlow**
- OpenFlow:
  - Control switch by inserting <header;action> entries
  - Essentially gives NOS remote access to forwarding table
  - Instantiated in OpenvSwitch


App
App
App
Controller
(Server Software)
OpenFlow Protocol
Ethernet Switch

Plumbing Primitives
<Match, Action>

Match arbitrary bits in headers:

- Match on any header, or new header

Match: 1000x01xx0101001x

- 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

38

OpenFlow Table Entry

Rule	Action	Stats

Packet + byte counters

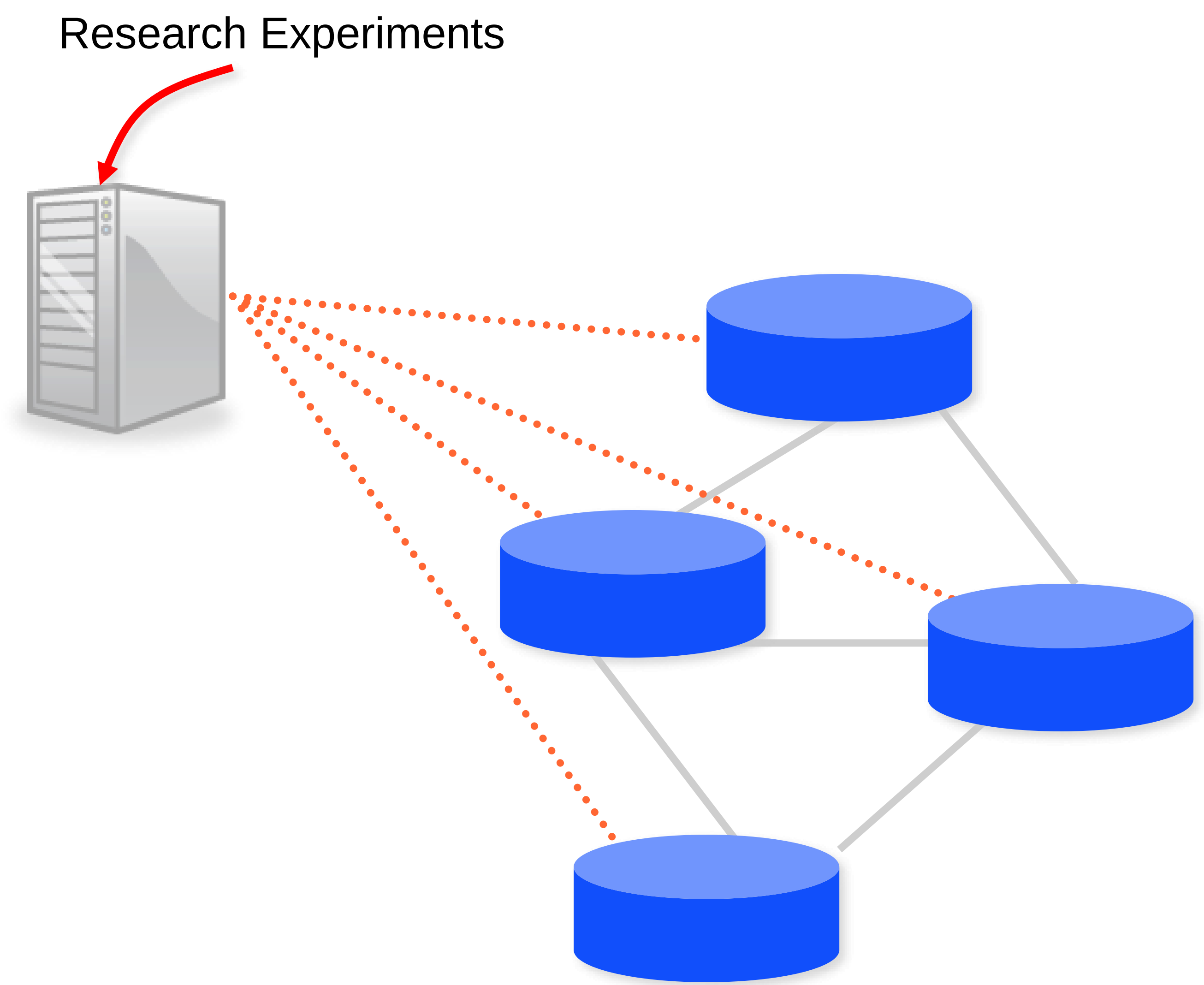
1. Forward packet to port(s)
2. Encapsulate and forward to controller
3. Drop packet
4. Send to normal processing pipeline
5. …

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

+ mask

3
9

Step 1:
Separate Control from Datapath

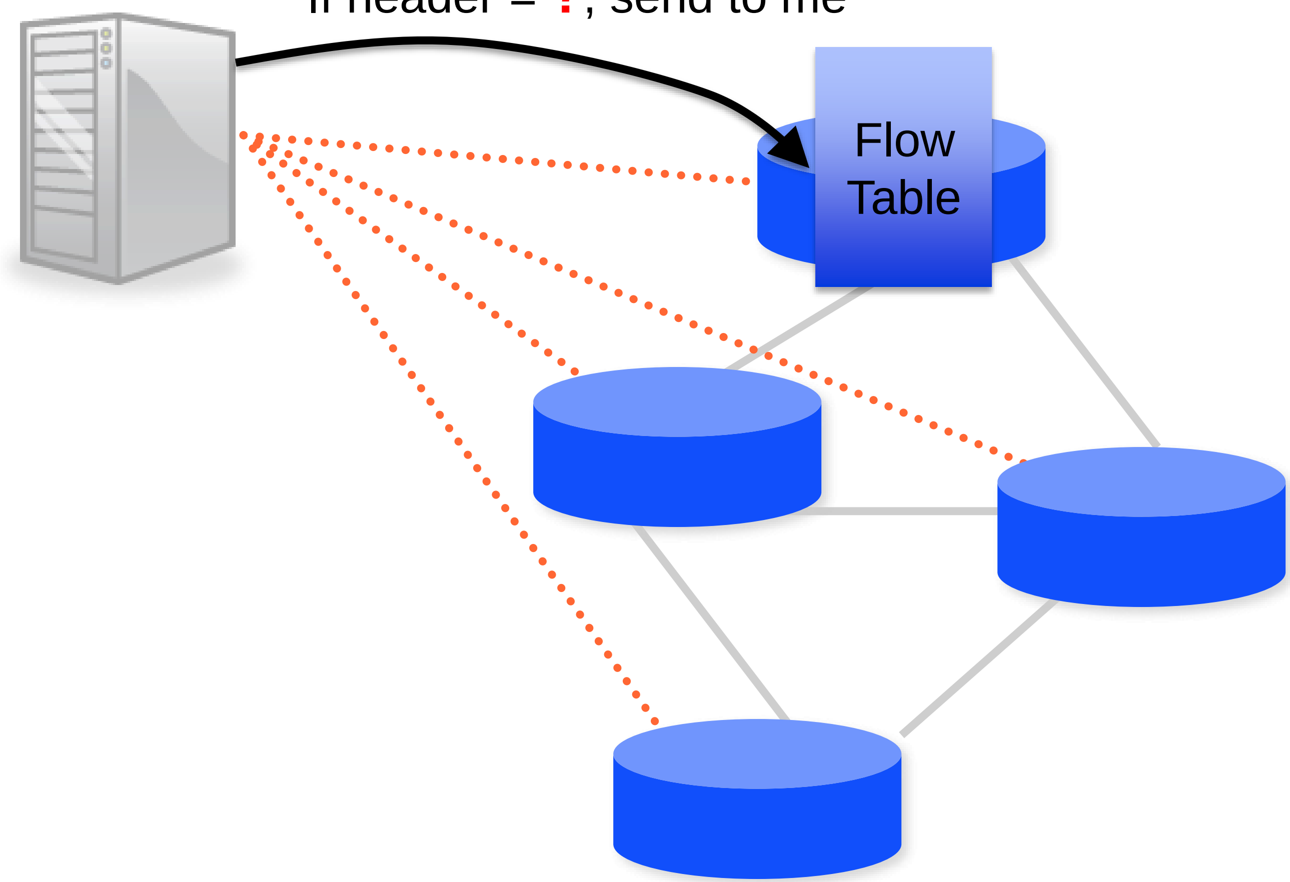


Step 2:
Cache flow decisions in datapath

“If header = ***x***, send to port 4”

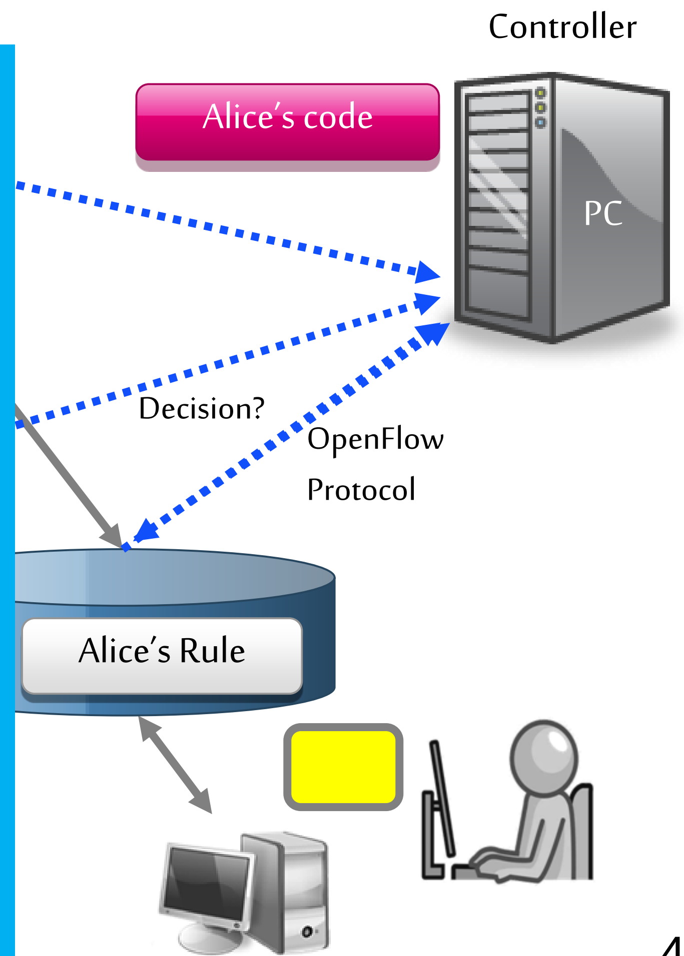
“If header = ***y***, overwrite header with ***z***, send to ports 5,6”

“If header = **?**, send to me”



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



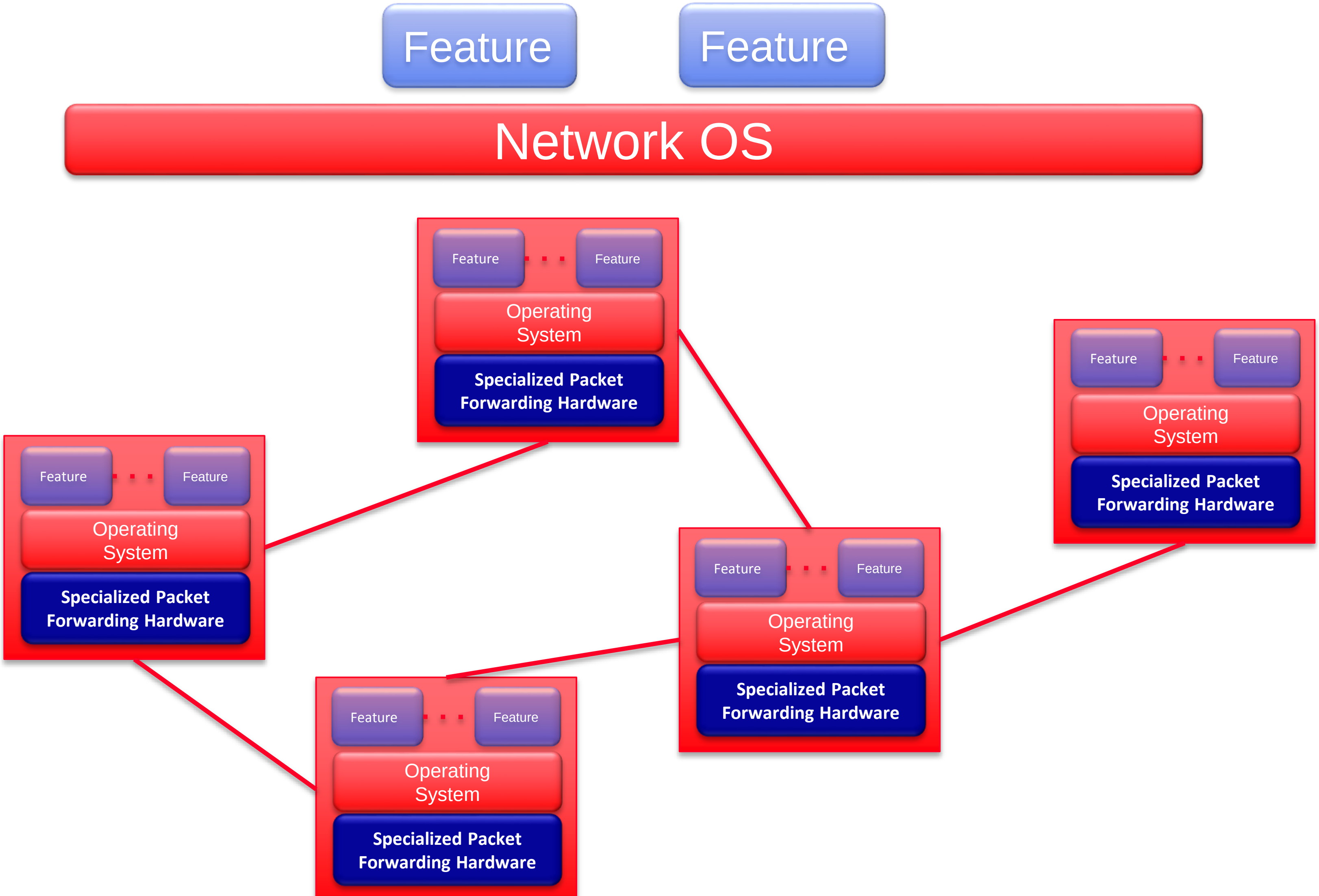
OpenFlow Standardization

Version 1.0: Most widely used version

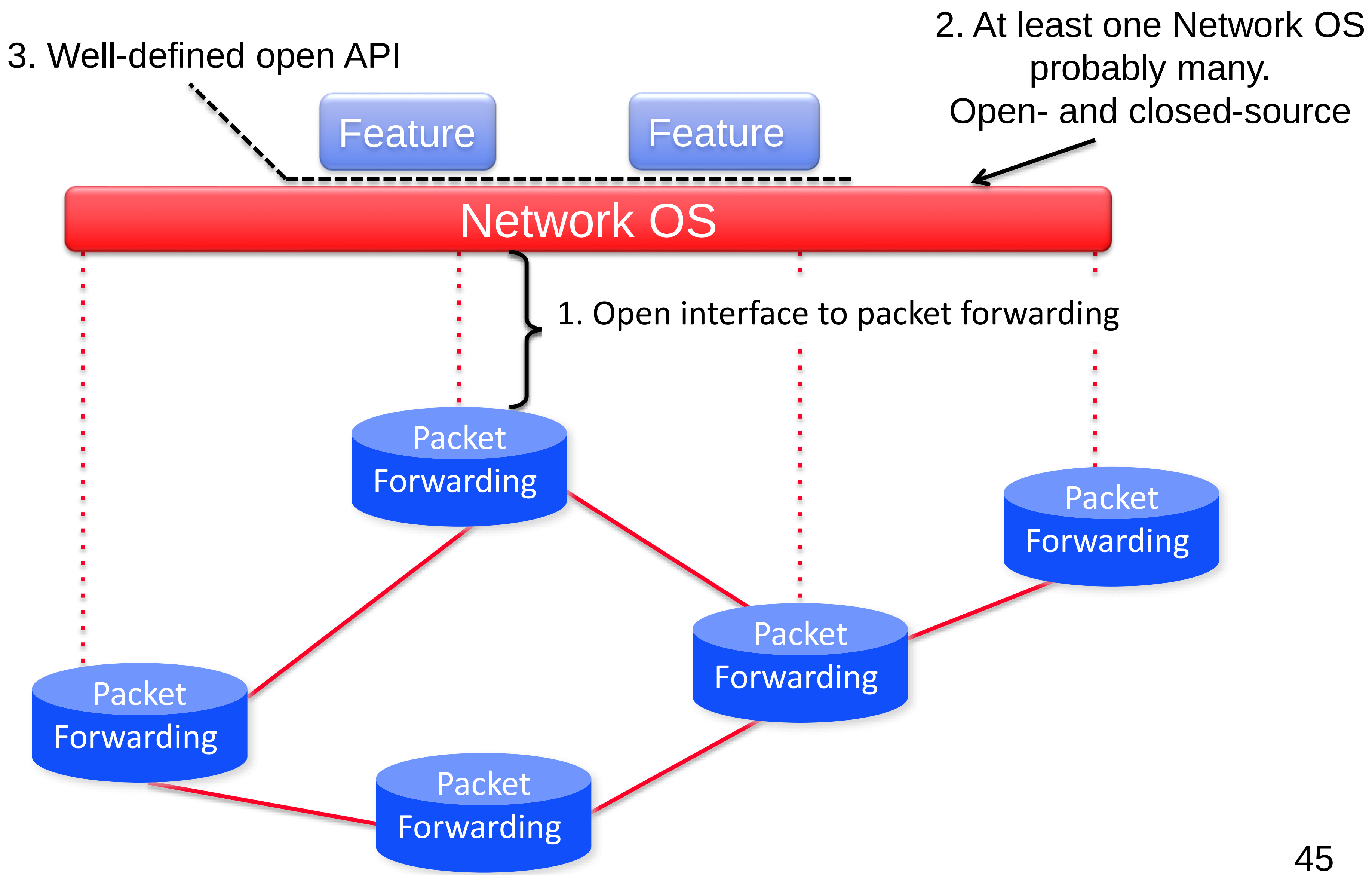
Version 1.1: Released in February 2011.

OpenFlow transferred to ONF in March 2011.

Restructured Network



Software-Defined Network



Does SDN Work?

• Is it scalable?	**Yes**
• Is it less responsive?	**No**
• Does it create a single point of failure?	**No**
• Is it inherently less secure?	**No**
• Is it incrementally deployable?	**Yes**

SDN: Clean Separation of Concerns

- **Control prgm: specify behavior on abstract model**
  - Driven by **Operator Requirements**
- **Net Virt’n: map abstract model to global view**
  - Driven by **Specification Abstraction**
- **NOS: map global view to physical switches**
  - API: driven by **Distributed State Abstraction**
  - Switch/fabric interface: driven by **Forwarding Abstraction**

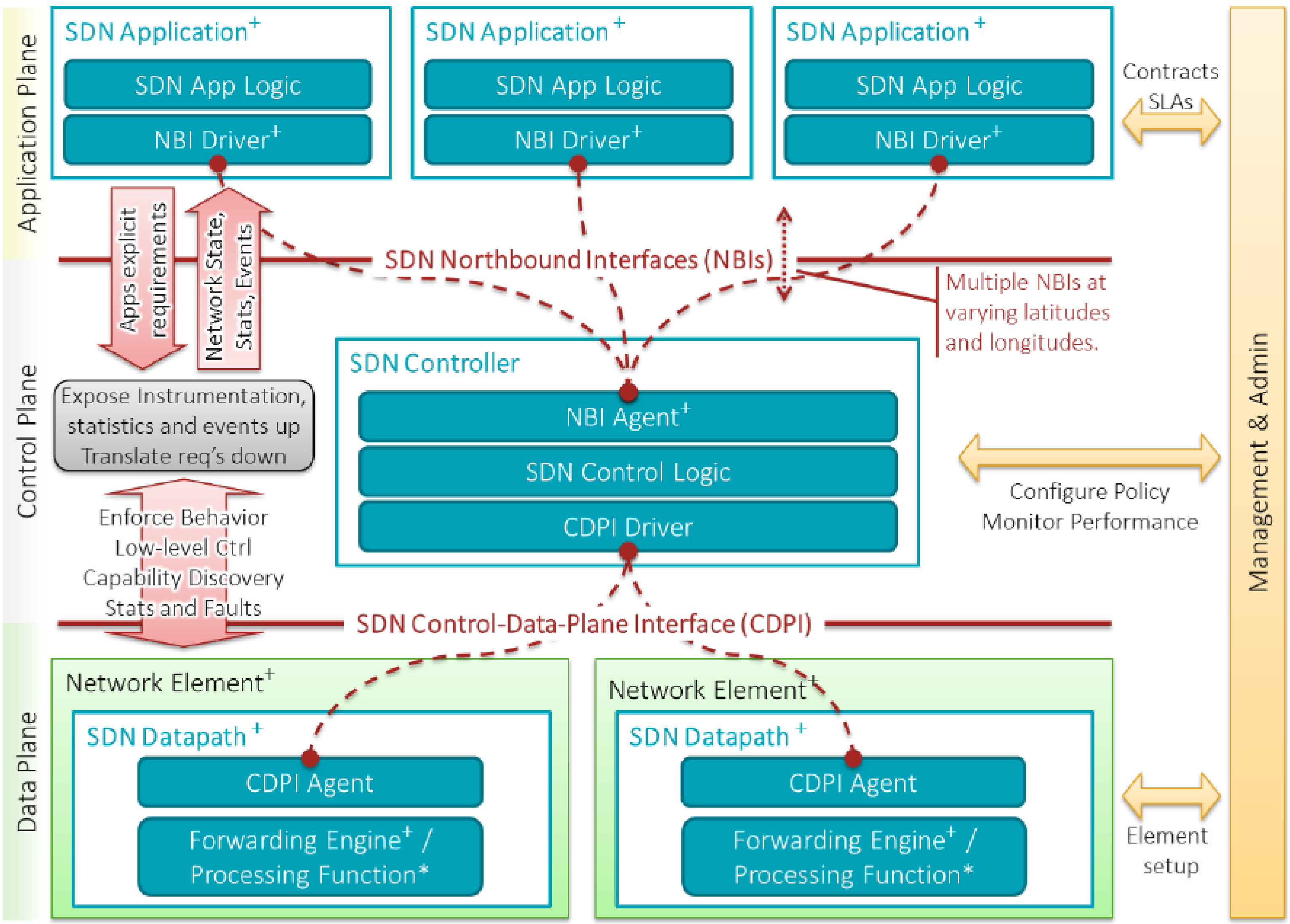
We Have Achieved Modularity!

- Modularity enables independent innovation
  - Gives rise to a thriving ecosystem

- Innovation is the true value proposition of SDN
  - SDN doesn’t allow you to do the impossible
  - It just allows you to do the possible much more easily

- ***This is why SDN is the future of networking…***

SDN Architecture Overview (ONF v1.0)



+ indicates one or more instances | * indicates zero or more instances