Software Defined Networking (SDN)

Marco.Cello@unige.it

DITEN – Università di Genova

Talk @ IEIIT – Consiglio Nazionale delle Ricerche (CNR)

Genova 28 Marzo 2014

Presented in CSE 291 @ UCSD by Gregory Kesden

Material from:

- Scott Shenker (UC Berkeley), “Software-Defined Networking at the Crossroads”, Standford, Colloquium on Computer Systems Seminar Series (EE380), 2013.
- Scott Shenker (UC Berkeley), “A Gentle Introduction to Software Defined Networks”, Technion Computer Engineering Center, 2012. http://tce.technion.ac.il/files/2012/06/Scott-shenker.pdf
- Scott Shenker (UC Berkeley), “The Future of Networking, and the Past of Protocols”, Open Network Summit, 2011. http://www.opennetsummit.org/archives/oct11/shenker-tue.pdf
- Nick McKeown (Stanford), ITC Keynote, San Francisco, 2011. http://yuba.stanford.edu/~nickm/talks/ITC%20Keynote%20Sept%202011.ppt

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, Microsoft, Facebook, NTT
- **Members**: Cisco, Juniper, HP, Dell, Broadcom, IBM,…..

2013: Latest Open Networking Summit

- 1600 attendees, Google: SDN used for their WAN
- Commercialized, in production use (few places)

Why Was SDN Needed?

- Networks are hard to manage
  - Computation and storage have been virtualized
    - Creating a more flexible and manageable infrastructure
  - Networks are still notoriously hard to manage
    - Network administrators large share of sysadmin staff
- Networks are hard to evolve
  - Ongoing innovation in systems software
    - New languages, operating systems, etc.
  - Networks are stuck in the past
    - Routing algorithms change very slowly
    - Network management extremely primitive
- Networks design not based on formal principles
  - OS courses teach fundamental principles
    - Mutual exclusion and other synchronization primitives
    - Files, file systems, threads, and other building blocks
  - Networking courses teach a big bag of protocols
    - No formal principles, just general design guidelines

Networks design not based on formal principles

- Networks used to be simple
  - Basic Ethernet/IP straightforward, easy to manage
- New control requirements have led to complexity
  - ACLs, VLANs, TE, Middleboxes, DPI,…
- The infrastructure still works...
  - Only because of our great ability to master complexity
- Ability to master complexity both blessing and curse

How Programming Made the Transition

- 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,...

Abstractions simplify programming

Easier to write, maintain, reason about programs

Abstractions are the way we extracted simplicity

So, what role do abstractions play in networking?

The Two Networking "Planes"

- **Data plane**: processing and delivery of packets with local forwarding state
  - Forwarding state + packet header →forwarding decision
- **Control plane**: compute the state in routers (forwarding state)
  - Determines how and where packets are forwarded
  - Routing, traffic engineering, firewall state, …
  - Implemented with distributed protocols, manual configuration (and scripting) or centralized computation

- These different planes require different abstractions

Data Plane Abstractions: Layers

Applications

…built on…

Reliable (or unreliable) transport

…built on…

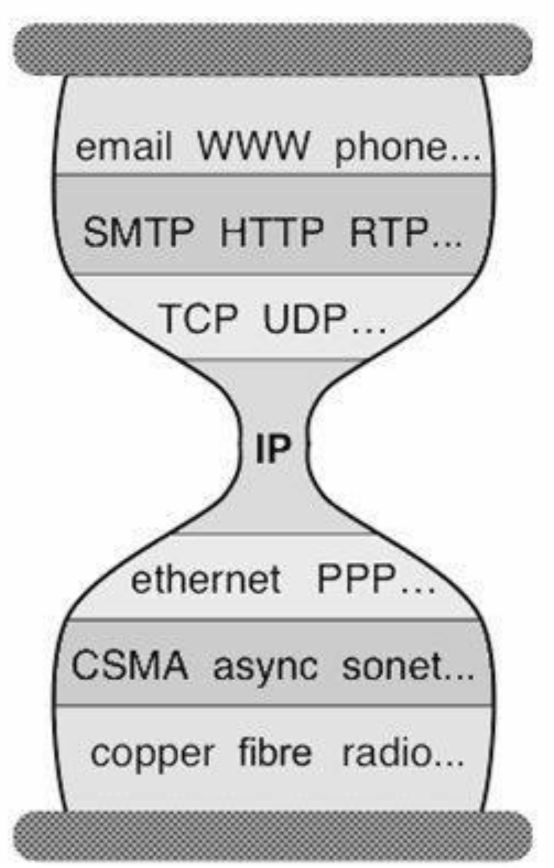
Best-effort global packet delivery

…built on…

Best-effort local packet delivery

…built on…

Local physical transfer of bits



Control Plane Abstractions

?

(Too) Many Control Plane Mechanisms

- Variety of goals:
  - **Routing:** distributed routing algorithms
  - **Isolation**: ACLs, VLANs, Firewalls,…
  - **Traffic engineering**: adjusting weights, MPLS,…

- No modularity, limited functionality

- **Control Plane: mechanism without abstraction**
  - ***Too many mechanisms, not enough functionality***

What abstractions should we apply to the control plane?

The Control Plane Problem

- Control plane must compute forwarding state. To accomplish its task, the control plane must:
  1. Figure out what network looks like (topology)
  2. Figure out how to accomplish goal on given topology
  3. Tell the swtiches what to do (configure forwarding state)
- We view this as a natural set of requirements....
  - And we require each new protocol to solve all three

This is crazy!

Programming Analogy

- What if you were told to write a program that must…
  - Be aware of the hardware you were running on
  - Specify where each bit was stored
- Programmer would immediately define abstractions:
  - Machine-independent interface
  - Virtual memory interface
- Programmers use abstractions to separate concerns
  - Network designers should too!

The Control Plane Problem

- Control plane must compute forwarding state. To accomplish its task, the control plane must:
  1. Figure out what network looks like (topology)
  2. Figure out how to accomplish goal on given topology
  3. Tell the swtiches what to do (configure forwarding state)

- What components do we want to reuse?
  1. Determining the topology information
  3. Configuring forwarding state on routers/switches

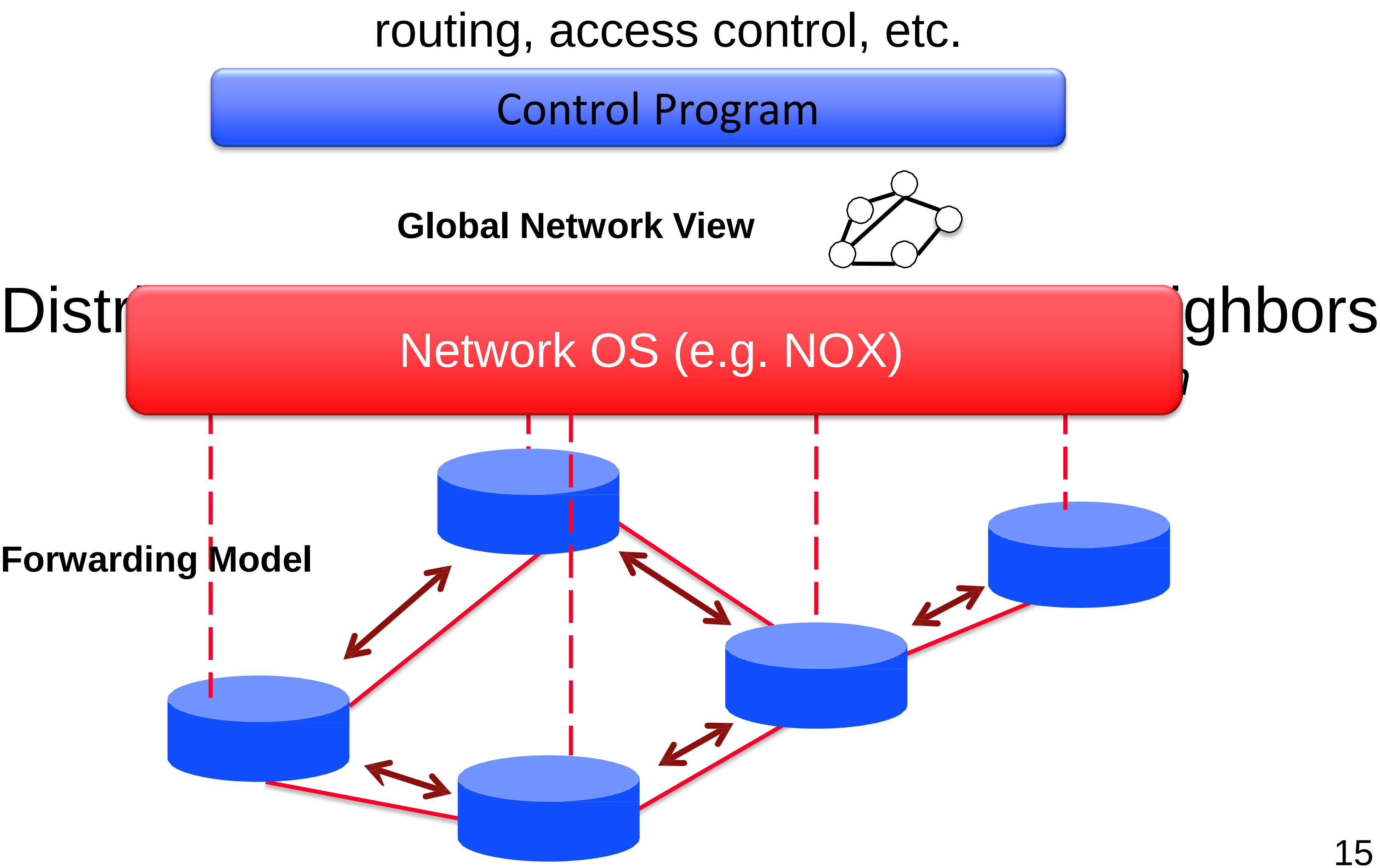
- You now know everthing you need about SDN:
  - It is the use of those two control planes abstractions

SDN: Two Control Plane Abstractions

- Abstraction: **global network view**
  - Provides information about current network
  - **Implementation**: “Network Operating System”
    - Runs on servers in network (replicated for reliability)

- Abstraction: **forwarding model**
  - Provides standard way of defining forwarding state
  - This is OpenFlow
    - Specification of <match,action> flow entries

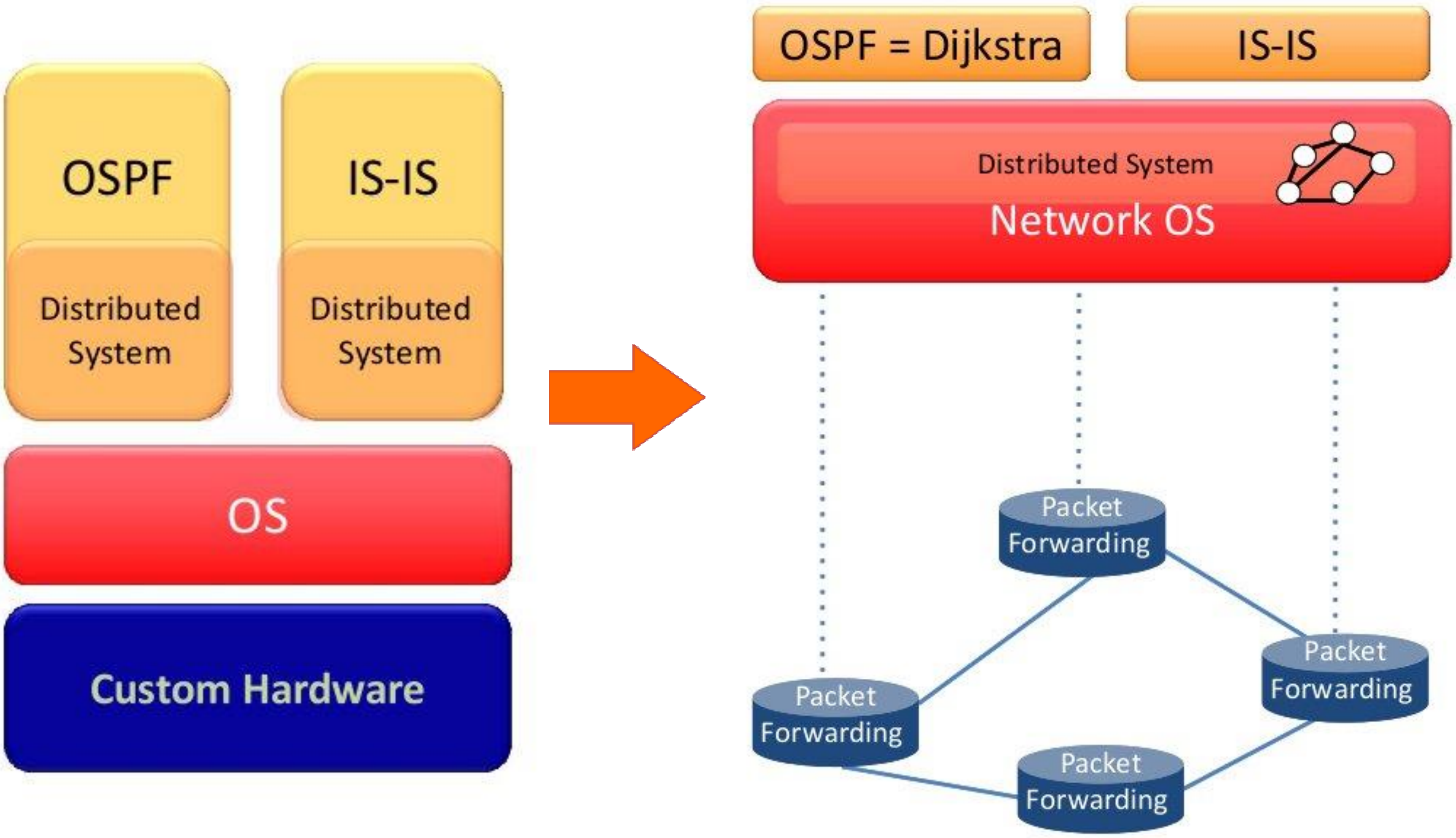
[illegible]


routing, access control, etc.
Control Program
Global Network View
Network OS (e.g. NOX)
Forwarding Model

Example1: OSPF and Dijkstra

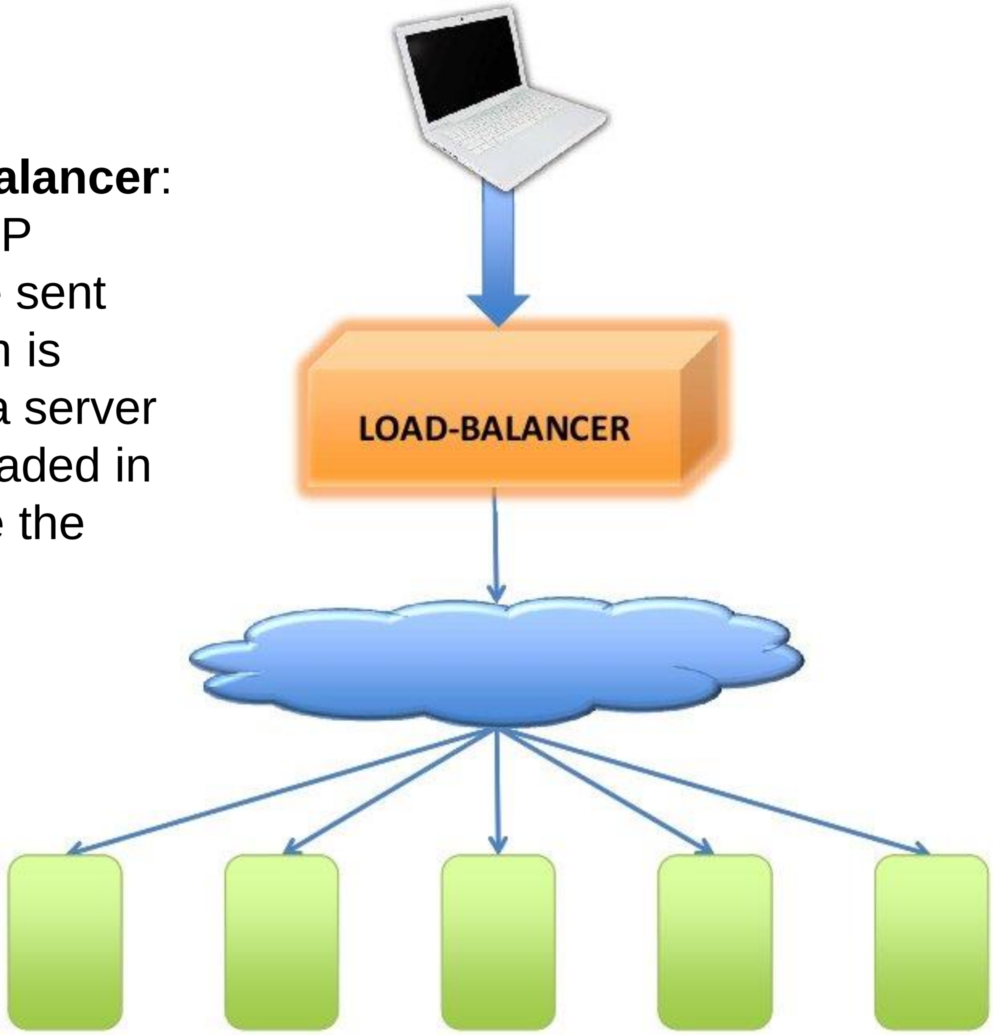
- OSPF
  - RFC 2328: 245 pages
- Distributed System
  - Builds consistent, up-to-date map of the network: 101 pages
- Dijkstra’s Algorithm
  - Operates on map: 4 pages

Example1: OSPF and Dijkstra

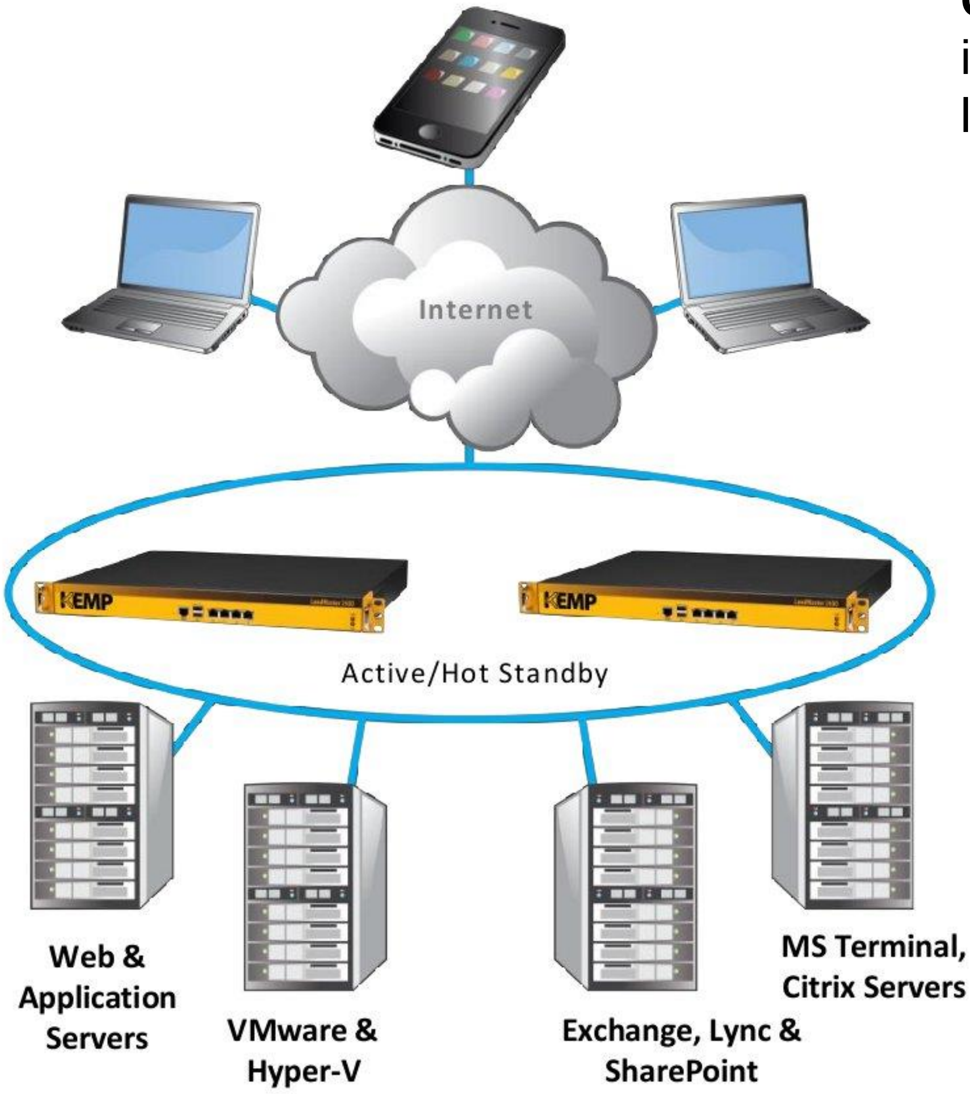


Example2: Load Balancing

Optimal Load Balancer: Ideally each HTTP request would be sent over a path which is lightly loaded to a server which is lightly loaded in order to minimize the request



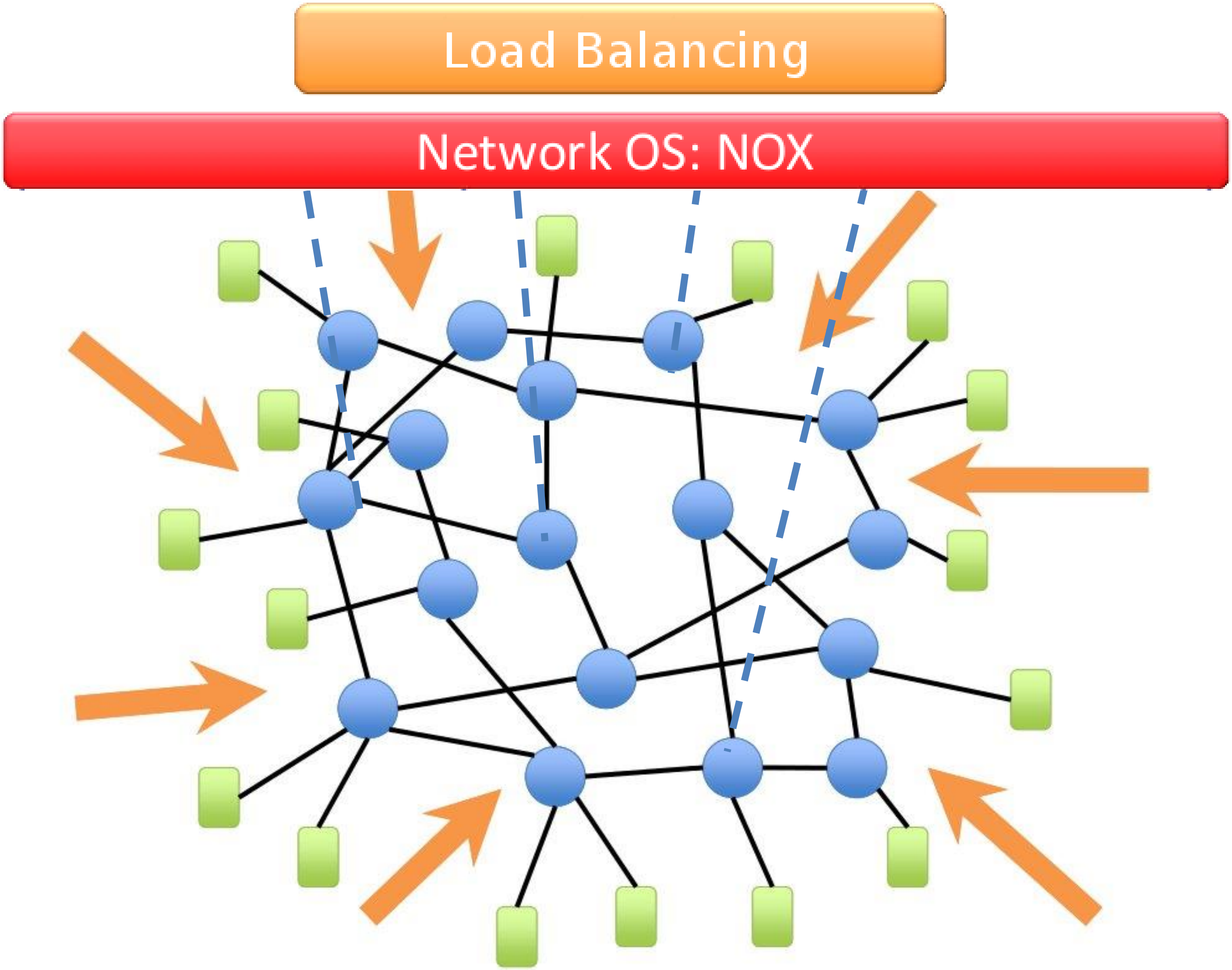
Example2: Load Balancing



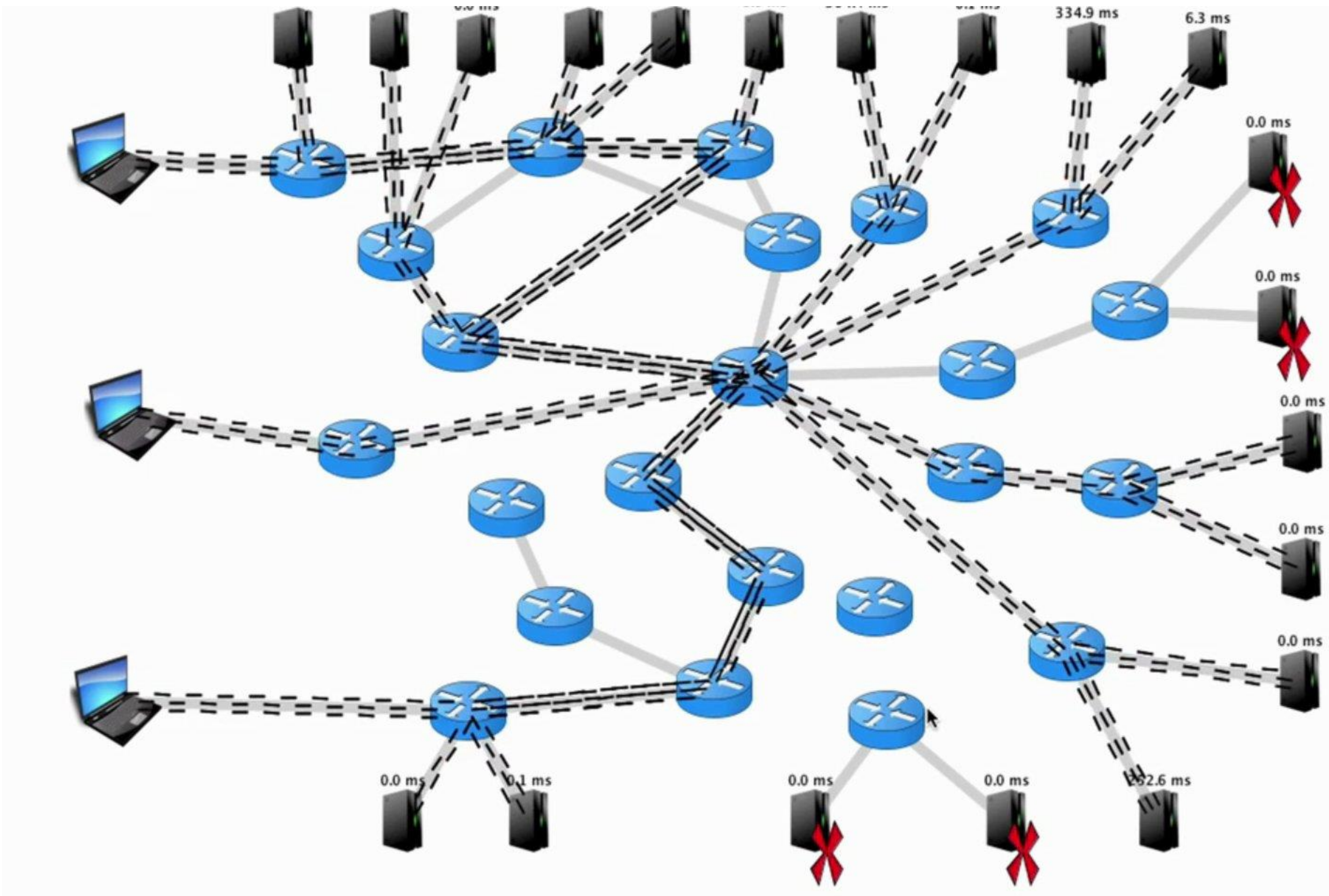
Current Load Balancer: it can choose only the lightly loaded server

KEMP Technologies LoadMasterTM 2400

Example2: Load Balancing



Example2: Load Balancing



N. Handigol, S. Seetharaman, M. Flajslik, R. Johari, and N. McKeown. Aster*x: Load-balancing as a network primitive. 9th GENI Engineering Conference (Plenary), November 2010

Specification Abstraction

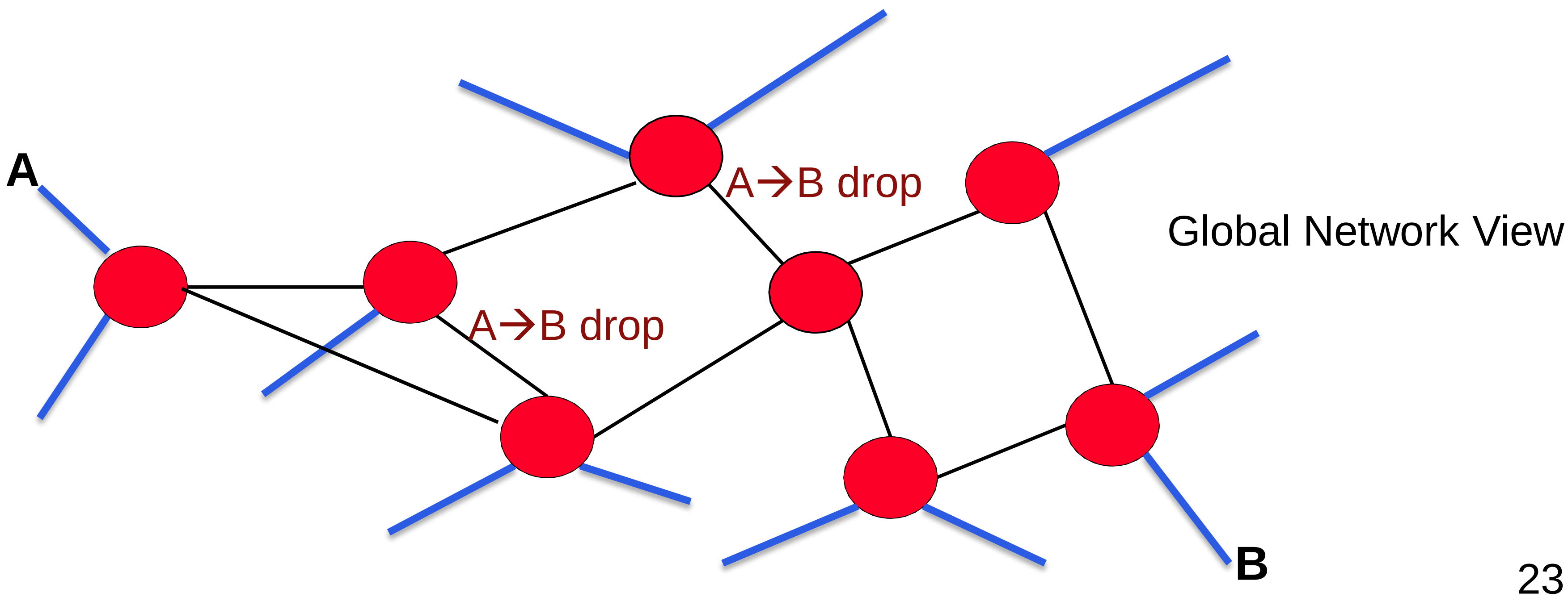
- Control program must ***express*** desired behavior
  - Whether it be isolation, access control, or QoS

- It should not be responsible for ***implementing*** that behavior on physical network infrastructure
  - Requires configuring the forwarding tables in each switch

- Proposed abstraction: **Virtual Topology** of network
  - Virtual Topology models only enough detail to *specify goals*
  - Will depend on task semantics

Simple Example: Access Control

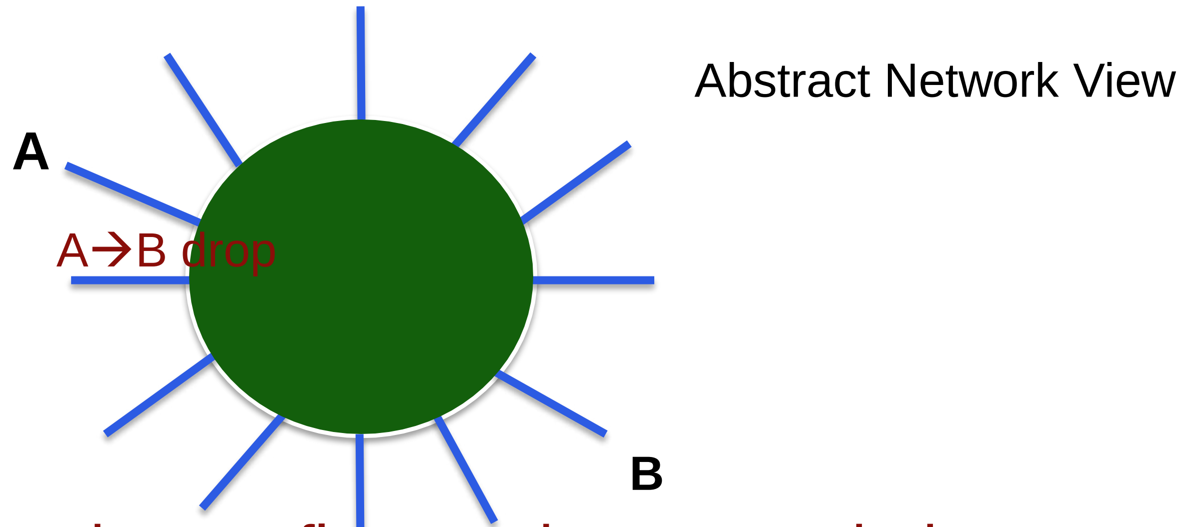
- Operator's goal: prevent A's packets from reaching B
- Control program does so with access control entries:
  - Control program must respond to topology/routing changes
  - Makes it hard to write correct control program



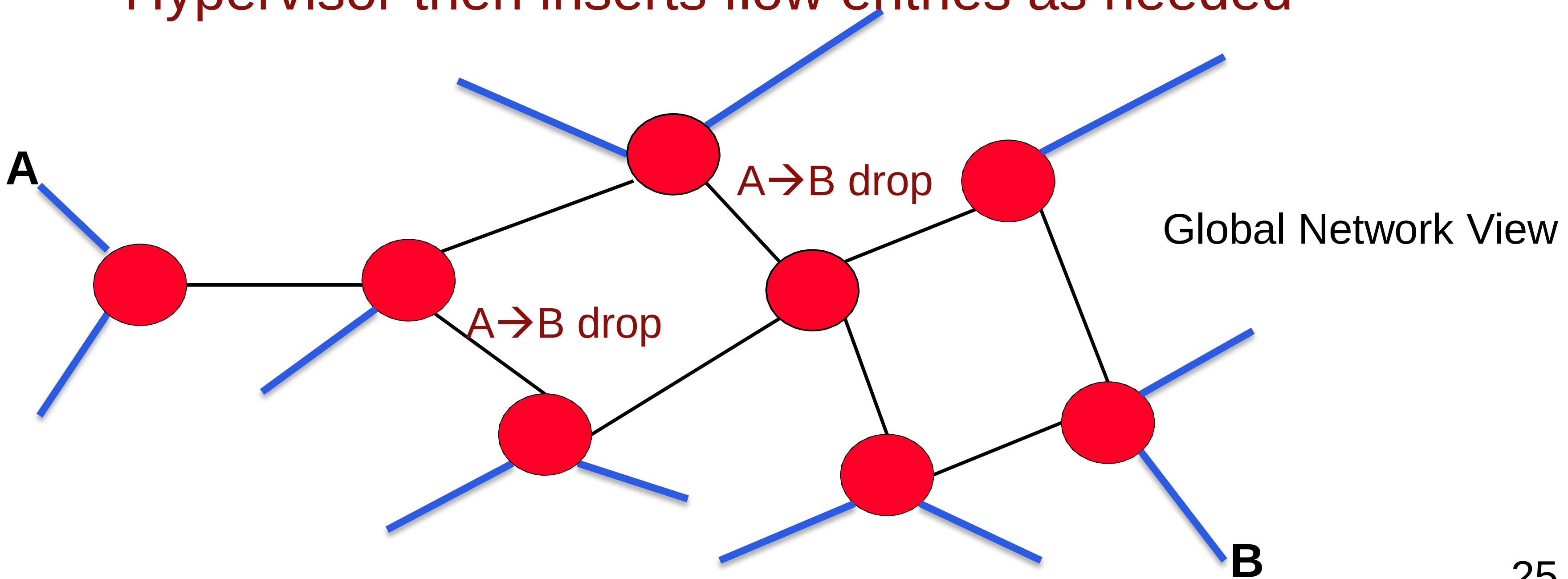
Network Virtualization

- Introduce new abstraction and new SDN layer
- Abstraction: **Virtual Topology**
  - Allows operator to express requirements and policies
  - Via a set of logical switches and their configurations
- Layer: **Network Hypervisor**
  - Translates those requirements into switch configurations
  - "Compiler" for virtual topologies

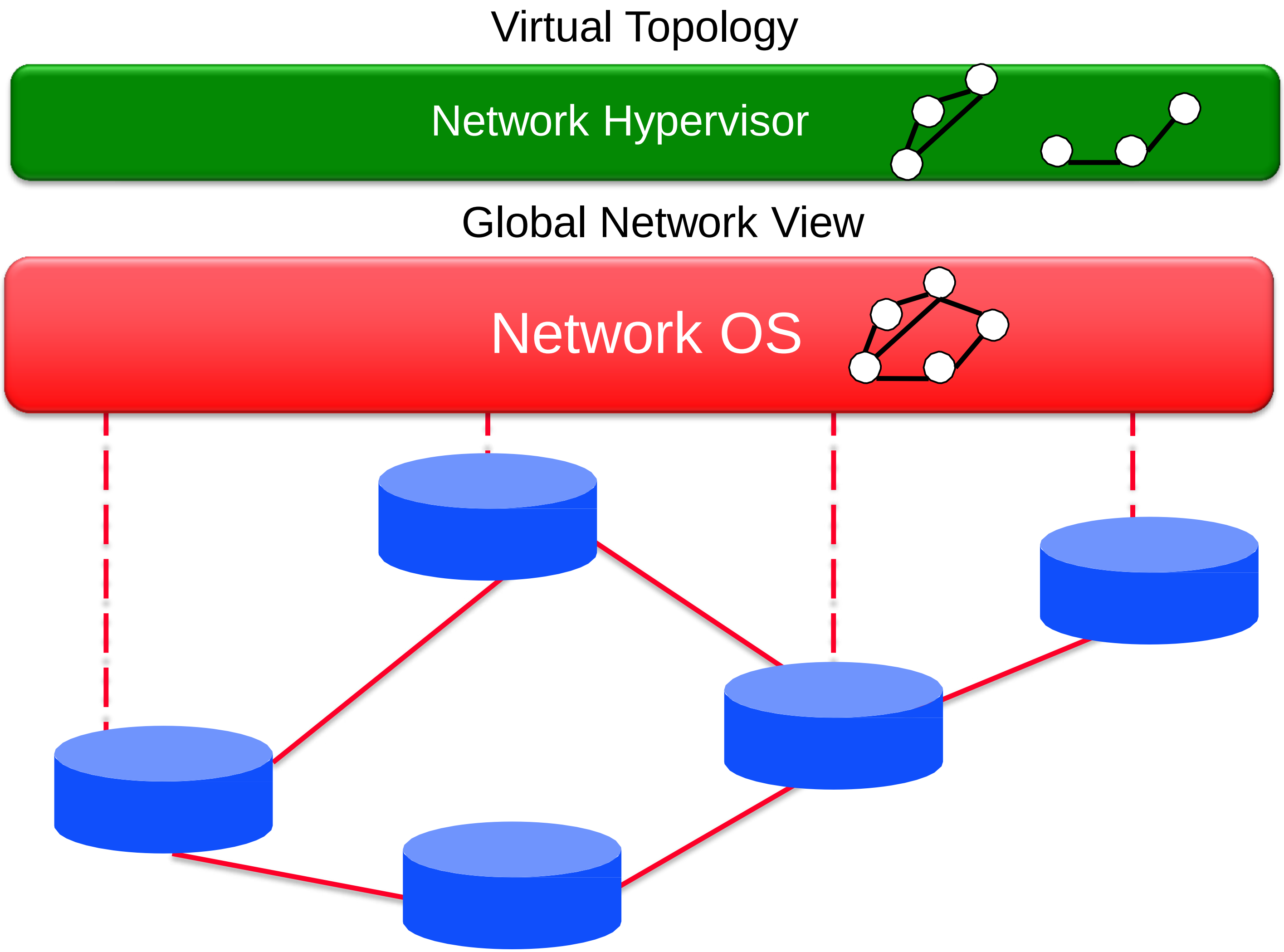
Virtualization Simplifies Control Program



Hypervisor then inserts flow entries as needed



Software Defined Network



Clean Separation of Concerns

- **Control program:** express goals on Virtual Topology
  - **Operator Requirements**
  - **Configuration = Function(view)**
  - Not a distributed protocol, now just a graph algorithm

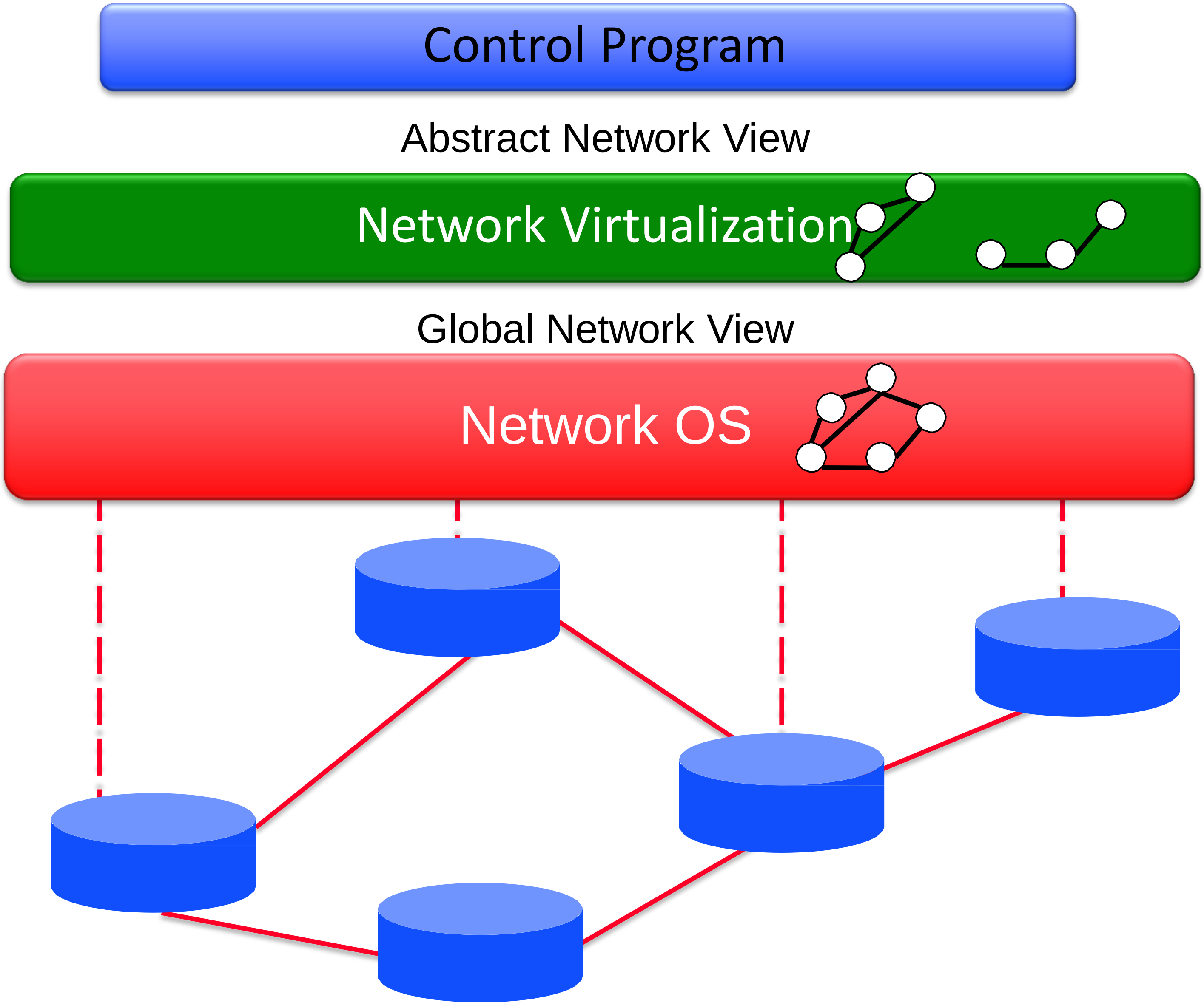
- **Network Hypervisor:** Virtual Topology ←→ Global Network View

- **Network OS:** Global Network View ←→ physical switches
  - Gathers information for global network view
  - Conveys configurations from control program to switches

- Router/switches: merely follow orders from NOS

- **Clean separation of control and data planes**
  - Not packaged togheter in proprietary boxes
  - Enables use of commodity hardware, 3rd party software
  - Easier to write, maintain, verify, reason about, …

SDN: Layers for the Control Plane



Abstractions Don't Eliminate Complexity

- Every component of system is tractable
  - NOS, Virtualization 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….

Virtualization is Killer App for SDN

- Consider a multi-tenant datacenter
  - Want to allow each tenant to specify virtual topology
  - This defines their individual policies and requirements

- Datacenter's network hypervisor compiles these virtual topologies into set of switch configurations
  - Takes 1000s of individual tenant virtual topologies
  - Computes configurations to implement all simultaneously

- ***This is what people are paying money for….***
  - ***Enabled by SDN's ability to virtualize the network***

What Should I Remember About SDN?

Four Crucial Points

- SDN is merely set of abstractions for control plane
  - Not a specific set of mechanisms
  - OpenFlow is least interesting aspect of SDN, technically

- SDN involves computing a function….
  - NOS handles distribution of state

- …on an abstract network
  - Can ignore actual physical infrastructure

- Network virtualization is the "killer app"
  - Already virtualized compute, storage; network is next

Does SDN have larger implications?

Aside from providing easier network management, how will SDN change the world of networking?

Control/Data Planes Become Separate

- Currently control plane tied to data plane
- NOS runs on servers: observes/controls data plane
- Changes the deployment and business models
  - Can buy the control plane separately from the switches
  - Enabling commodity hardware and 3rd party software
- Changes the testing model
  - Simulator to analyze large-scale control planes

Networking Becomes Edge-Oriented

- Can implement most control functionality at edge
  - Access control, QoS, mobility, migration, monitoring…
- Network core merely delivers packets edge-to-edge
  - Current protocols do a good job (mostly)
- **Let edge handle all complexity**
  - Complicated matching, actions
  - “Overlay” networking via tunnels
- This has two important implications

1. Makes SDN Incrementally Deployable

- Host software often has OpenFlow switch
  - Open vSwitch (OVS) in Linux, Xen,…
- The edge becomes a software switch
  - Core of network can be legacy hardware
- Enables incremental deployment of SDN
  - Might never need OpenFlow in hardware switches….

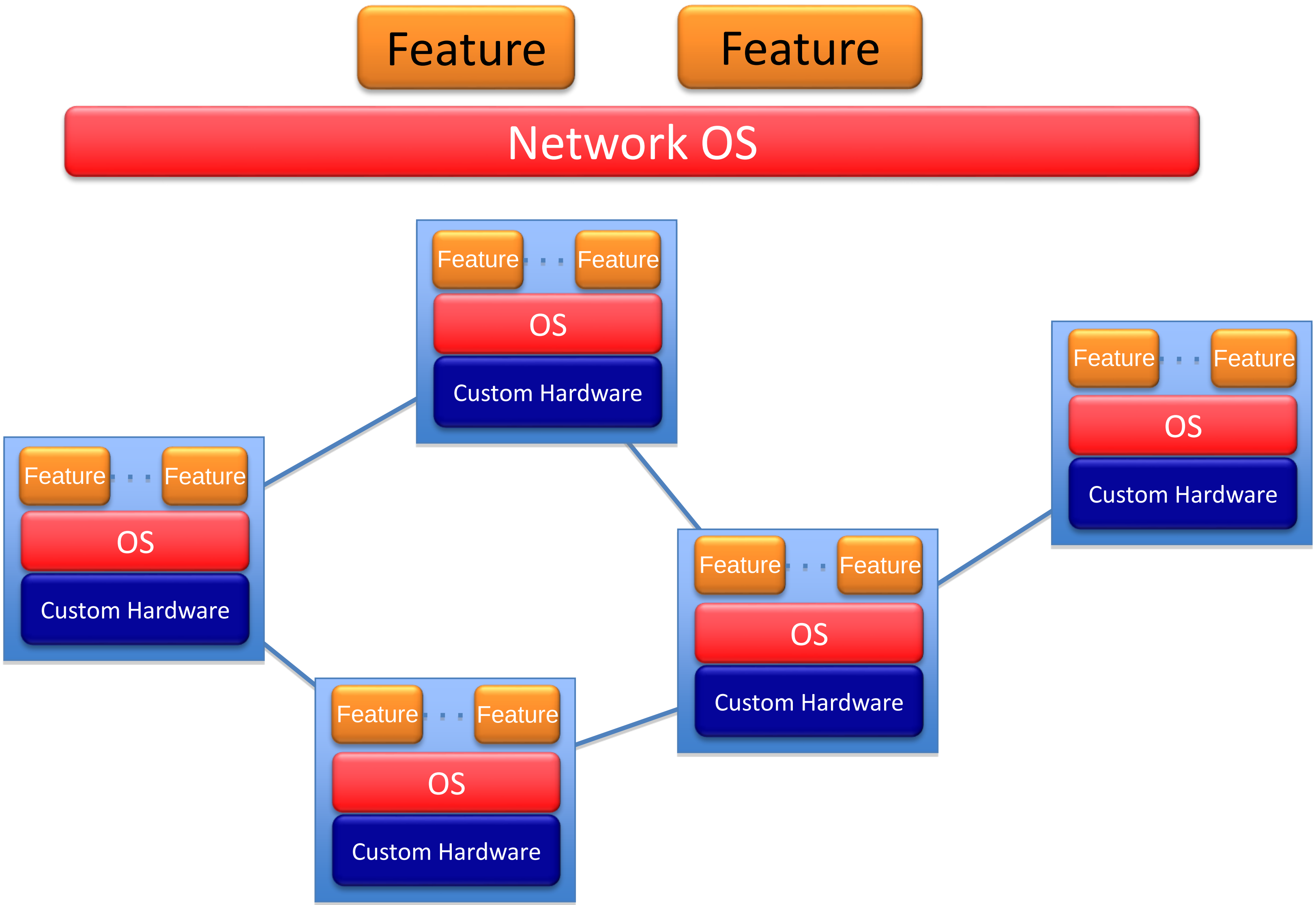
2. Networking Becomes Software-Oriented

- All complicated forwarding done in software (edge)
- And control plane is a program (on a server)…
  - …not a protocol (on a closed proprietary switch/router)
- We are *programming* the network, not designing it
  - Focus on modularity and abstractions, not packet headers
- **Innovation at software, not hardware, speeds**
- Software lends itself to clean abstractions

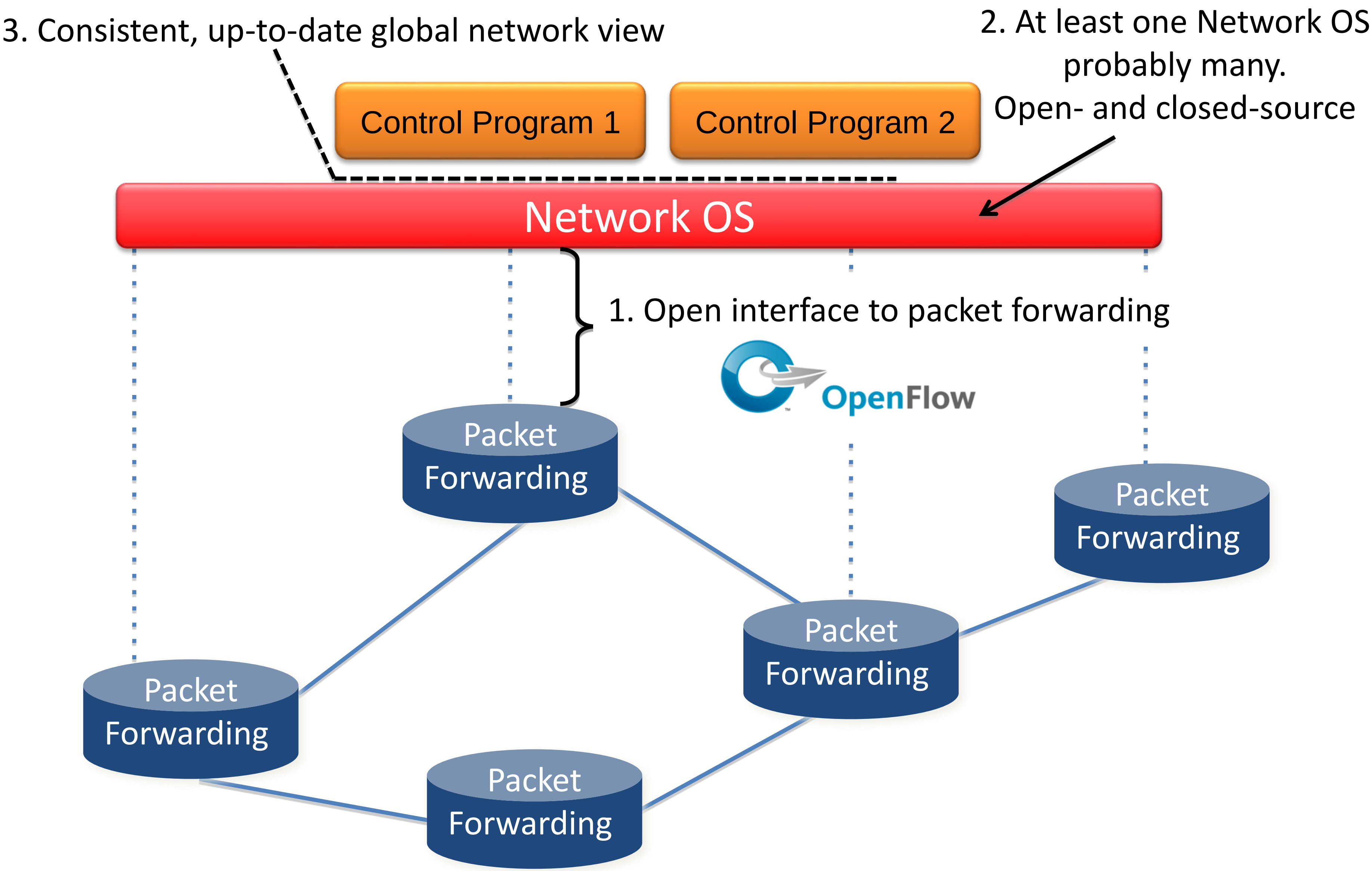
SDN Vision: Networks Become “Normal”

- Hardware: Cheap, interchangeable, Moore’s Law
- Software: Frequent releases, decoupled from HW
- Functionality: Mostly driven by SW
  - Edge (software switch)
  - Control program
- Solid intellectual foundations

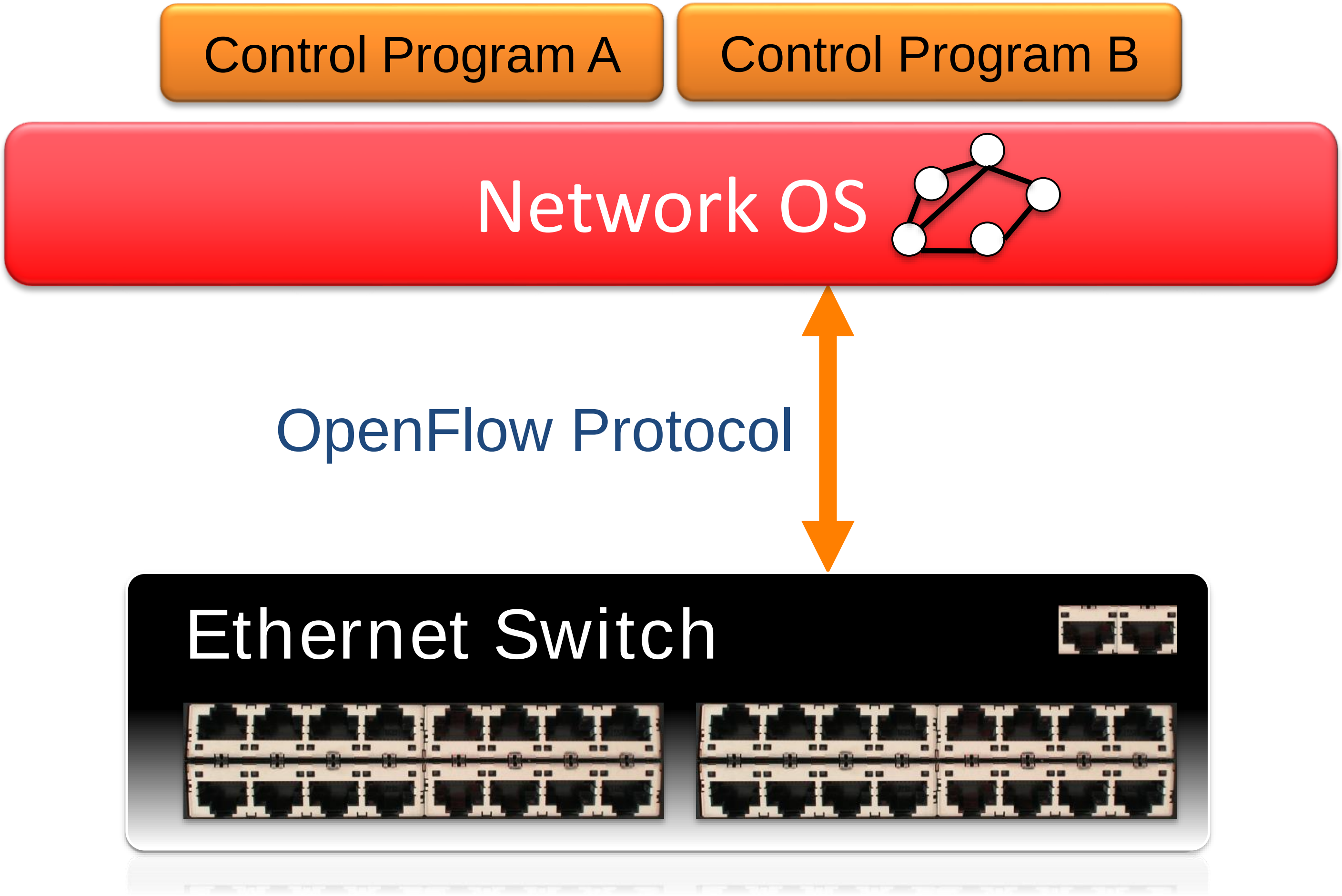
Recap - The network is changing



Recap - Software Defined Network (SDN)

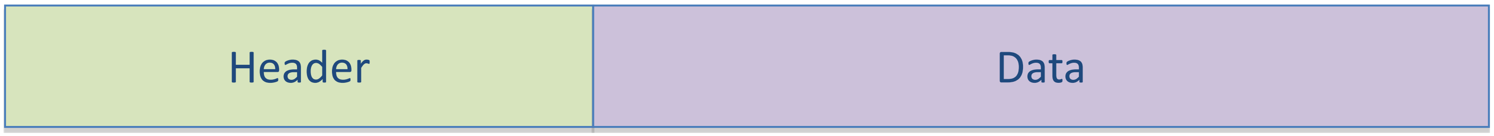


OpenFlow Basics



Primitives <Match, Action>

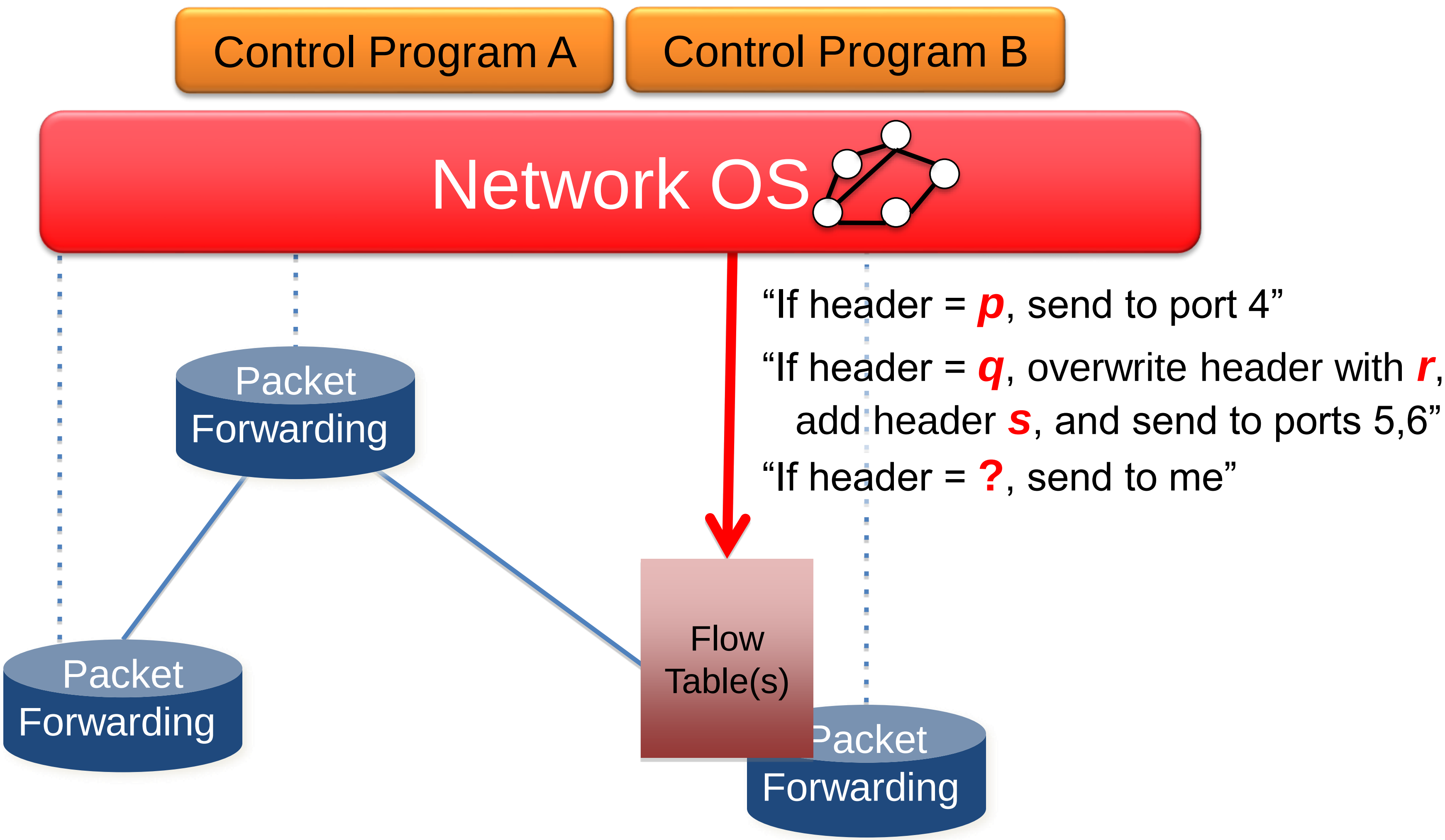
- **Match** arbitrary bits in headers:



Match: 1000x01xx0101001x

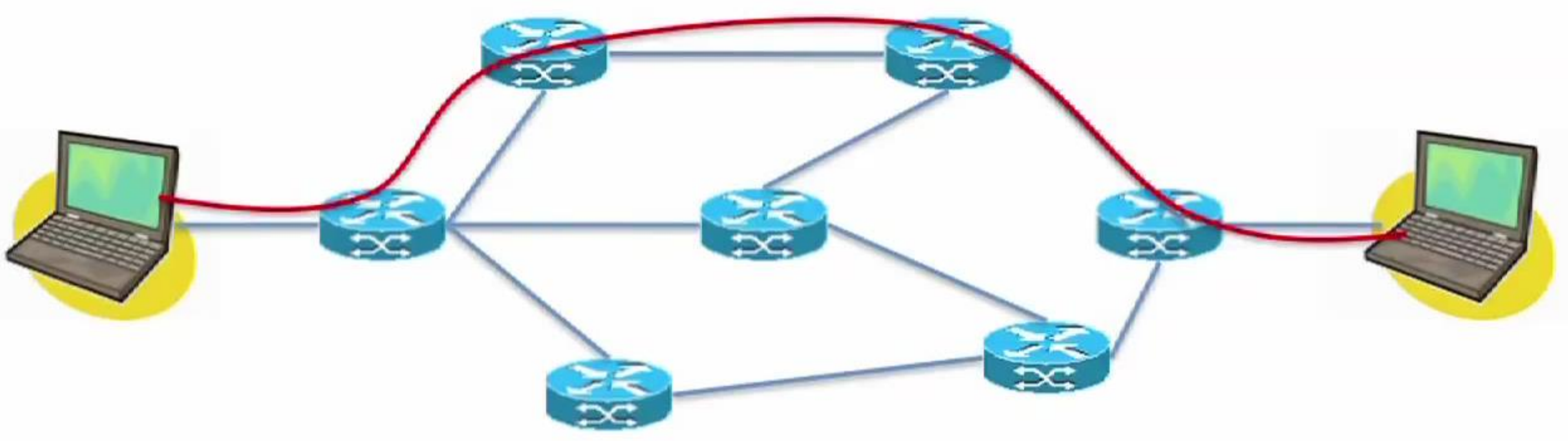
  - Match on any header, or new header
  - Allows any flow granularity
- **Action**
  - Forward to port(s), drop, send to controller
  - Overwrite header with mask, push or pop
  - Forward at specific bit-rate

OpenFlow Basics



More sophisticated flow identification

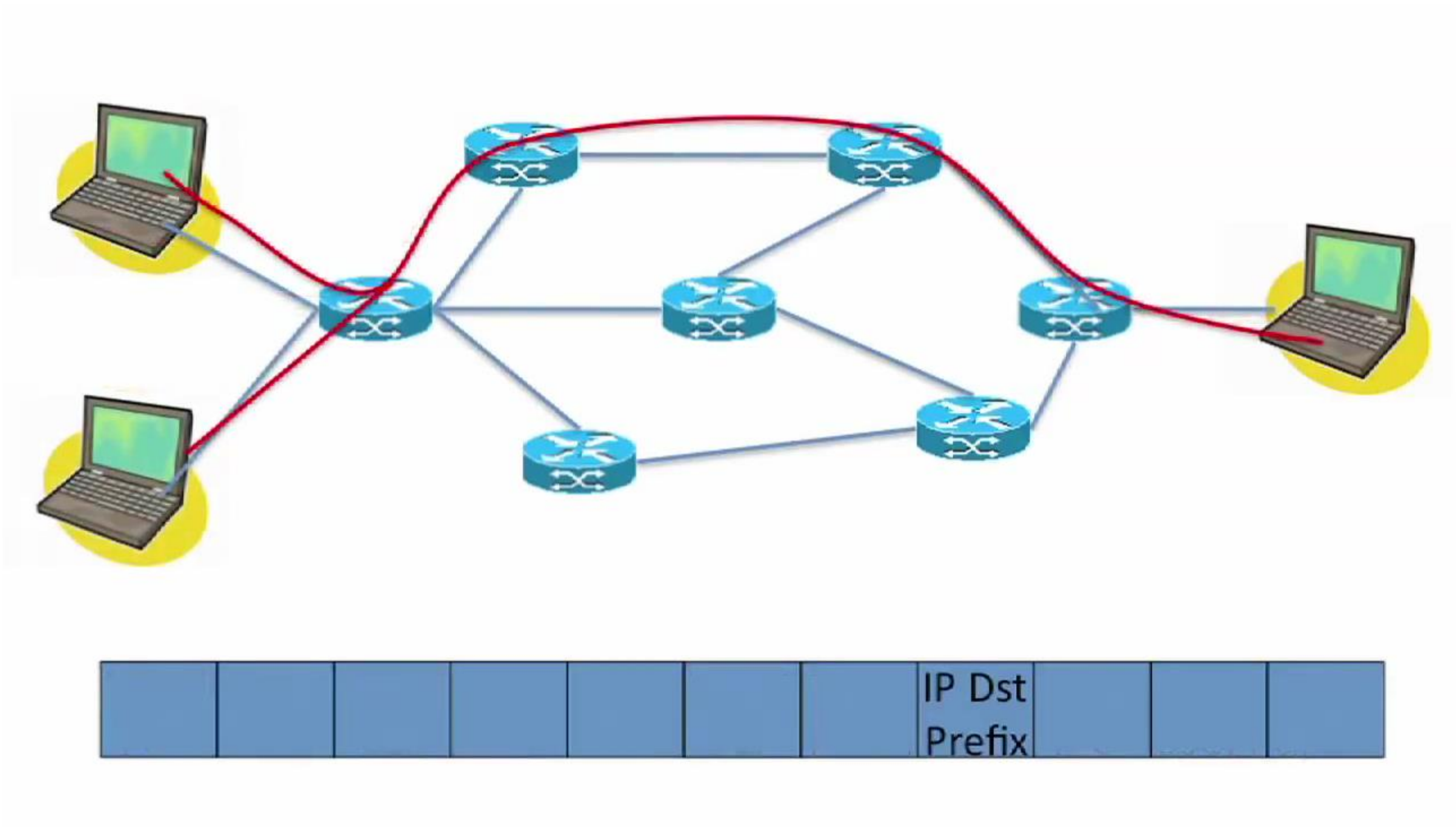
Application level flow



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

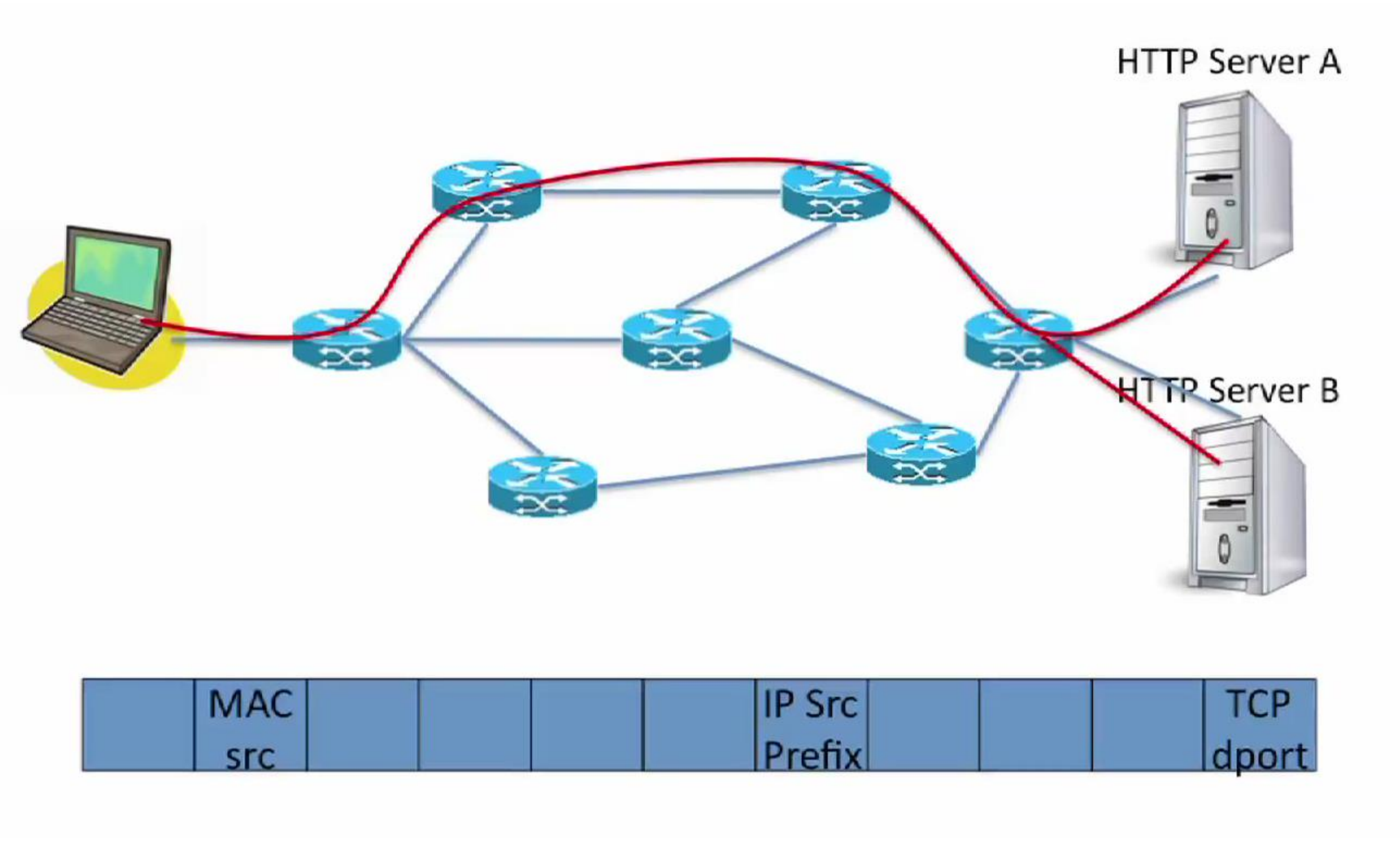
More sophisticated flow identification

IP flow



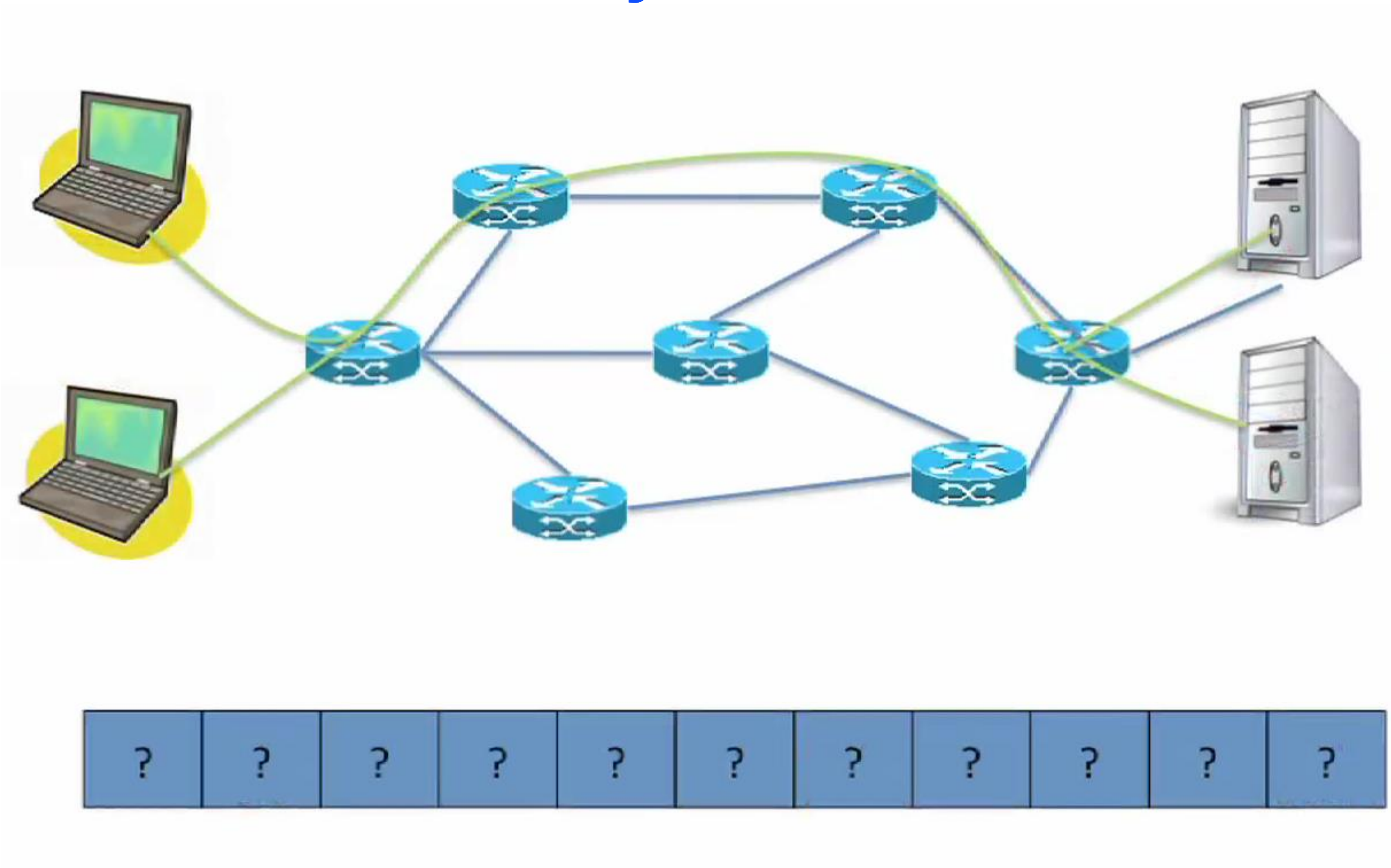
More sophisticated flow identification

Custom flow



More sophisticated flow identification

My flow



SDN "Implementations" – Software/Hardware

- Forwarding Model
  - OpenFlow
  - ForCES
- Software Switches compliant with OpenFlow std.
  - Open vSwitch
  - Pantou/OpenWRT
  - Ofsoftswitch13
  - Indigo
- Controller compliant with OpenFlow std.
  - POX
  - NOX
  - MUL
  - Maestro
- Available Commodity Switches compliant with OpenFlow std.
  - Hewlett-Packard 8200zl, 6600, 6200zl,
  - Brocade 5400zl, and 3500/3500yl
  - IBM NetIron CES 2000 Series

Bruno Astuto A. Nunes, Marc Mendonca, Xuan-Nam Nguyen, Katia Obraczka, and Thierry Turletti, "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks", Technical Report, http://hal.inria.fr/hal-00825087/PDF/bare_jrnl.pdf

SDN Literature - Sources

- Browsing on proceedings of:
  - ACM Sigcomm;
  - ACM Sigcomm Workshop HotSDN;
  - ACM Sigcomm Workshop HotNets;
  - ACM CoNEXT;
  - USENIX NSDI;
  - USENIX HotCloud;
  - USENIX Hot-ICE;
  - ONS;
- SDN reading list: http://www.nec-labs.com/~lume/sdn-reading-list.html

SDN research areas

SDN architecture

Controller scalability

- multi-controller
- reduce messages sent to controller
- switch/CPU design approaches

Network Updates

Programming

Testing/Debugging

SDN applications

Traffic Management/QoS

- flow scheduling
- Load balancing
- Transport protocol

Monitoring

Security