CS 168
Software-Defined Networking (SDN)

Fall 2022

Guest Lecture: Scott Shenker

CS168.io

Be Forewarned....

- First classroom lecture in four years...
  - ...so this could be very rough
- Please ask questions when I’m floundering
  - To save us all the embarrassment
- When **I** ask a question, not looking for an answer
  - I’m asking you to think!
- Lecture will start slowly, with lots of generalities
  - This is to establish the necessary context
  - But things get more specific towards the end

Goal For Today

- Provide the “why” of software-defined networking
  - Why was it needed, why did it come about,...
  - Some history, some gossip, and the post-hoc rationale
  - ***An exercise in retrospective architectural thinking***
- Almost none of the real “how”
  - Go read papers (RCP, 4D, NOX, ONIX, NetVirt, Fabric, …)
  - Sylvia is actively working on new ways of doing SDN
  - But the main point of SDN isn’t in the details of how...
- I am presenting the “canonical” version of SDN
  - Which absolutely no one uses in its pure form
  - But is the best way to understand what SDN is conceptually

We Begin With Two Questions

Q#1: What Is SDN?

- SDN is a way of managing networks
  - Will clarify what that means later in lecture
- However, SDN is not a revolutionary technology…
  - …just a way of organizing network functionality
- But that's all the Internet architecture is….
  - The Internet architecture isn't clever, but it is deeply wise
- *SDN isn't clever, and we can only hope it is wise*
  - *We'll find out in thirty or forty years…*

Q#2: Where did SDN come from?

- ~2004: Research on new management paradigms
  - RCP, 4D [Princeton, CMU,….]
  - SANE, Ethane [Stanford/Berkeley]
  - Industrial efforts with similar flavor (most not published)

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

- 2011: Open Networking Foundation (ONF)
  - **Board**: Google,Yahoo,Verizon,DT,Msft,Fbook,NTT,GS
  - **Members**: Cisco, Juniper, HP, Dell, Broadcom, IBM,…..

Where did SDN really come from?

Martín Casado

Current Status Of SDN

- SDN accepted as **right way to do networking**
  - Commercialized, in production use, growing revenue
    - E.g., in use at Google/MSoft/Amazon, carriers partially adopted
  - Not fully adopted by router vendors, so this talk will often refer to pre-SDN practices in the present tense
- Was an insane level of SDN hype, and backlash…
  - SDN doesn't work miracles, merely makes things easier
- But the real question is: *why the rapid adoption?*
  - 2004: **idea**, 2008: **design**, 2011: industry **frenzy**
  - *This is incredibly fast for networking!*

Why The Rapid Adoption?

- When a technology is adopted so quickly, it must be addressing a significant pain point.
  - Especially true in networking, which changes very slowly

- SDN was addressing ***two*** huge pain points

- #1: Cisco's extreme market power
  - Explains why ***vendors*** jumped on SDN

- #2: The poor state of network management
  - Explains why ***customers*** cared about SDN

Network Management

What is network management?

- Recall the two “planes” of networking
- **Data plane**: forwarding packets
  - Based on local forwarding state
- **Control plane**: computing that forwarding state
  - Involves coordination with rest of system
- Broad definition of “network management”:
  - ***Everything having to do with the control plane***

Original Goals For The Control Plane

- **Basic connectivity**: route packets to destination
  - Forwarding state computed by routing protocols
  - Globally (intradomain) distributed algorithms

- **Interdomain policy**: find policy-compliant paths
  - Done by globally (interdomain) distributed BGP

- For long time, these were the only relevant goals!

- ***What other goals are relevant now?***

- Here are a few examples…

Isolation Of Logical LANs

- L2 bcast protocols often used for discovery
  - Useful, unscalable, invasive
- Want multiple logical LANs on a physical network
  - Retain usefulness, cope with scaling, provide isolation
- Use VLANs (virtual LANs) tags in L2 headers
  - Controls where broadcast packets go
  - Can create multiple logical L2 networks
  - Routers connect these logical L2 networks
- No universal method for setting VLAN state

Access Control

- Operators want to limit access to various hosts
  - "Don't let laptops access backend database machines"
  - Crucial for security
- This can be imposed by routers using ACLs
  - ACL: Access Control List
- Example entry in ACL: <header template; drop>
  - If not port 80, drop
  - If source address = X, drop
- These are typically configured manually
  - And often implemented in firewalls

Traffic Engineering

- Choose routes to spread traffic load across links

- Two main methods:
  - Setting up MPLS tunnels *(MPLS is layer 2.5)*
  - Adjusting weights in OSPF

- Often done with centralized computation
  - Take snapshot of topology and load
  - Compute appropriate MPLS/OSPF state
  - Send state to network

Net management now has many goals

- Achieving these goals is job of the control plane…

- …which currently involves many mechanisms

- **Globally distributed:** routing algorithms

- **Manual/scripted configuration:** ACLs, VLANs

- **Centralized computation:** Traffic engineering

Bottom Line

- Many different control plane mechanisms
- Each designed from scratch for their intended goal
- Encompassing a wide variety of implementations
    - Distributed, manual, centralized,…
- And none of them particularly well designed
- **Network control plane was a complicated mess!**
    - **With mediocre functionality…**
- Big contrast with simple and functional dataplane

Questions?

How Have We Managed To Survive?

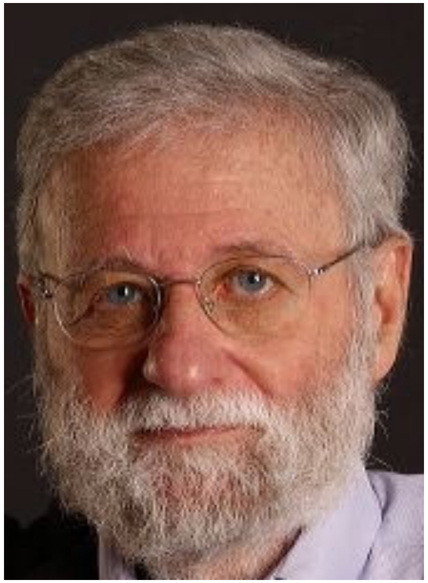
- Network admins must master this complexity
  - Understand all aspects of networks
  - Must keep myriad details in mind

- Networks require large expert admin staffs
  - Much larger than compute admin staffs
  - This is how we survive, by mastering complexity

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

A Simple Story About Complexity...

- ~1985: Don Norman visits Xerox PARC
  - Talks about user interfaces and stick shifts
  - Do you even know what a stick shift is?



What Was His Point?

- The ability to **master complexity** is valuable
  - But not the same as the ability to **extract simplicity**
- Each has its role:
  - When first getting systems to work, *master complexity*
    - ***Stick shifts!***
  - When making system easy to use, *extract simplicity*
    - ***Automatic transmissions!***
- You will never succeed in extracting simplicity
  - ***If you don't recognize it is a different skill set than mastering complexity!***

What Is *My* Point?

- Networking had never made the distinction…
  - And therefore never made the transition from mastering complexity to extracting simplicity for control plane

- Until SDN, focused on mastering complexity
  - Networking "experts" are those that know all the details

- Network management had suffered as a result

- *Simplify network mngmt requires extracting simplicity*
  - And we had never bothered to do that for control plane

Forcing People To Make Transition?

- We are really good at mastering complexity
  - And it had worked for us for decades, why change?
- How do you make people change?
  - Make them cry!
- A personal story about algebra and complexity
  - School problems:

    $3x + 2y = 8 \qquad x + y = 3$
  - My father's problems:

    $327x + 26y = 8757 \qquad 45x + 57y = 7776$
  - My response: (1) I cried, (2) I learned algebra

How Do You Make Network Operators Cry?

What convinced network operators
that they needed SDN?

Step 1: Large datacenters

- 100,000s machines; 10,000s switches
- Pushing the limits of what we could handle….

Step 2: Multiple tenancy

- Large datacenters can host many customers
  - Gave rise to the modern public cloud

- Each customer gets their own logical network
  - Customer should be able to set policies on this network
  - ACLs, VLANs, etc.

- If there are 1000 customers, that adds 3 oom
  - Where oom = orders of magnitude

- This went *way* beyond what we could handle
  - Because our control plane is so primitive!

Net Operators Were Now Weeping…

- They had been beaten by complexity
- The era of ad hoc control mechanisms was over
- We needed a simpler, more systematic design
  - We needed algebra, not arithmetic...
- But note the contrast between banks and multitenant datacenters:
  - One group willing to continue mastering complexity
  - The other was defeated, and needed something new
    - And they were desperate, which is why the rapid adoption

What Do We Do Now?

- We had been defeated by complexity in DCs
- So we had to “extract simplicity”!
- ***So how do you “extract simplicity”?***

An Example Transition: Programming

- Machine languages: no abstractions
  - Had to deal with low-level details
  - 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 About Network Abstractions?

- Consider the data and control planes separately
- Different tasks, so naturally different abstractions

Abstractions for Data Plane: Layers

Applications

…built on…

Reliable (or unreliable) transport

…built on…

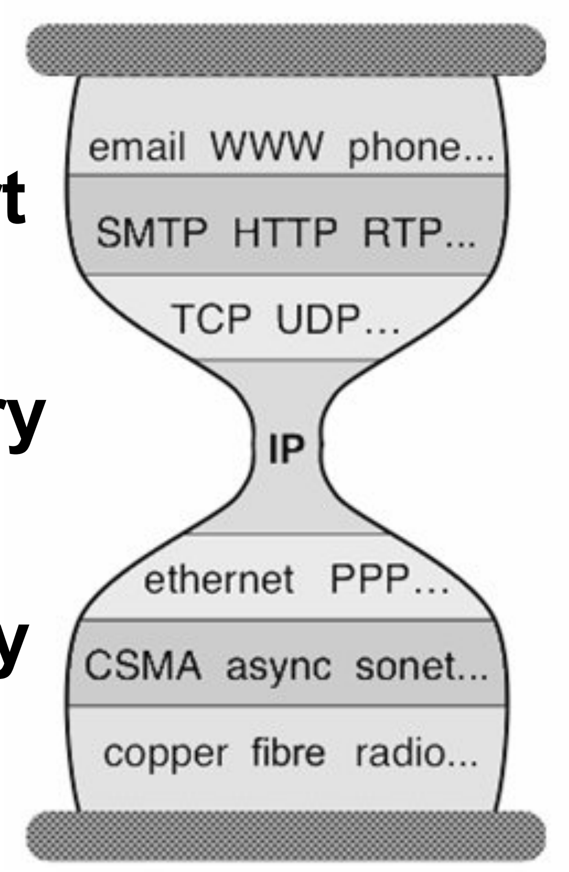
Best-effort global packet delivery

…built on…

Best-effort local packet delivery

…built on…

Physical transfer of bits



Control Plane Abstractions

?

Many Control Plane Mechanisms

- Variety of goals, no modularity:
  - **Routing:** distributed routing algorithms
  - **Isolation**: ACLs, VLANs, Firewalls,…
  - **Traffic engineering**: adjusting weights, MPLS,…

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

SDN: An Exercise in Finding Control Plane Abstractions

Lecture So Far

- We have motivated the need for SDN
  - Networks admins were using arithmetic...
  - ...but suddenly needed algebra

- We now talk about how SDN met that need
  - What is the “algebra” of network management

How do you find abstractions?

- You start with a task you need to perform
- You then decompose the task….
- …and define abstractions for each subtask
- Let’s do that for the control plane
- Basic task is to compute forwarding state
- But this task has several subtasks or constraints

Task: Compute Forwarding State

- Consistent with low-level hardware/software
  - Which might depend on particular vendor

- Based on entire network topology
  - Because many control decisions depend on topology

- For all routers/switches in network
  - Every router/switch needs forwarding state

The Pre-SDN Approach

- Design one-off mechanisms that deal with all three
    - E.g., routing protocols deal with all three subproblems

- A sign of how much we love complexity

- No other field would do it this way!

- They would define abstractions to handle each subtask independently

- …and so should we!

- And that is what leads to SDN

Separate Concerns With Abstractions

1. Be compatible with low-level hardware/software

   Need an abstraction for general **forwarding model**

2. Make decisions based on entire network

   Need an abstraction for **network state**

3. Compute configuration of each physical device

   Need an abstraction that **simplifies configuration**

Abs#1: Forwarding Abstraction

- Express intent independent of implementation
  - Don’t want to deal with proprietary HW and SW
- OpenFlow is one proposal for forwarding
  - Standardized interface to switch
  - Configuration in terms of flow entries: <header, action>
- Design details concern exact nature of:
  - Header matching
  - Allowed actions

Separate Concerns With Abstractions

1. Be compatible with low-level hardware/software

   Need an abstraction for general **forwarding model**

2. **Make decisions based on entire network**

   Need an abstraction for network state

3. Compute configuration of each physical device

   Need an abstraction that simplifies configuration

Abs#2: Network State Abstraction

- Abstract away various distributed mechanisms

- Abstraction: **global network view**
  - Annotated network graph provided through an API

- Implementation: “Network Operating System”
  - Runs on servers in network (“controllers”)
  - Replicated for reliability

- Information flows both to and from NOS
  - Information *from* routers/switches to form “view”
  - Configurations *to* routers/switches to control forwarding

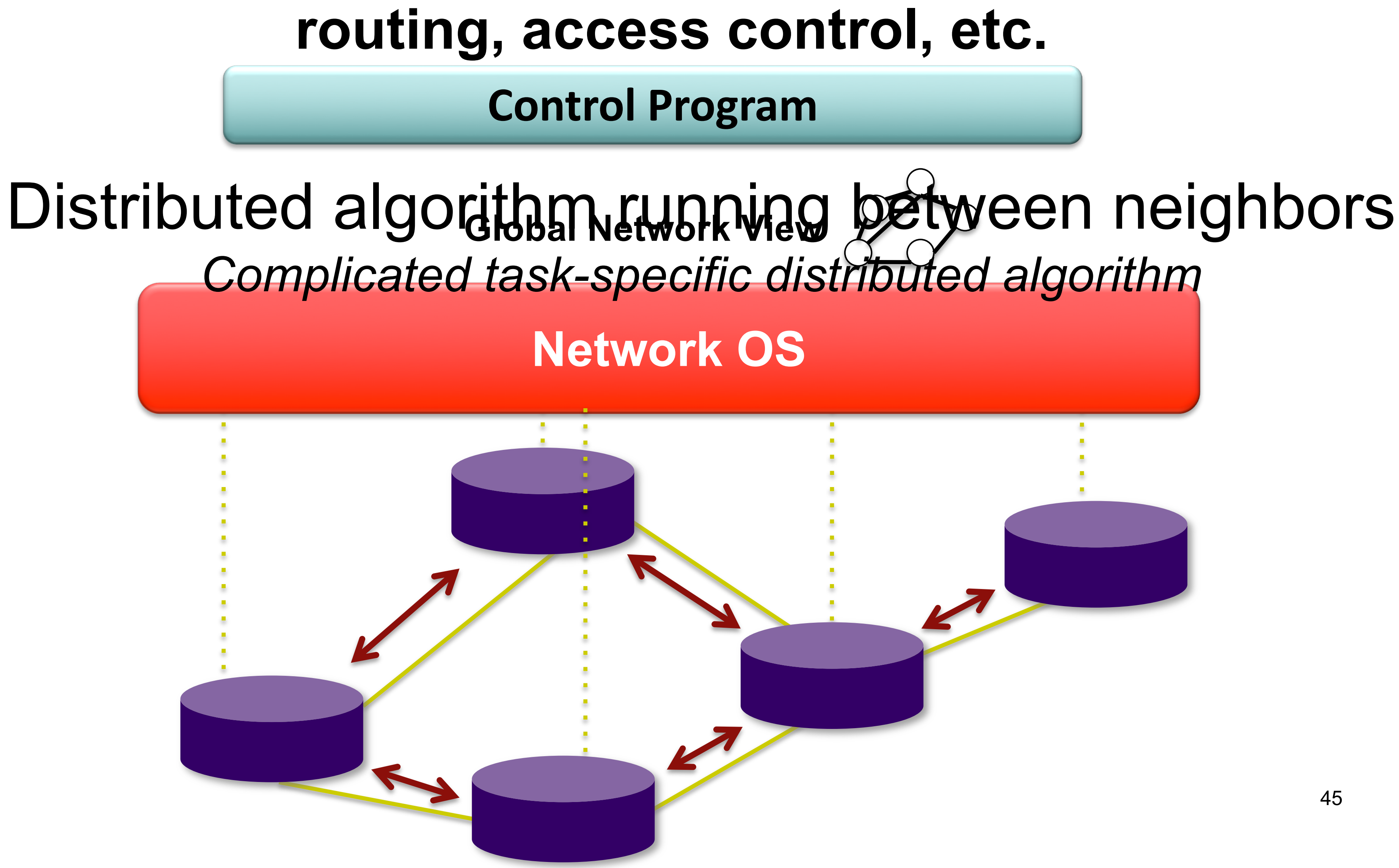
Network Operating System

- Think of it as a centralized link-state algorithm
- Switches send connectivity info to controller
- Controller computes forwarding state
  - Some control program that uses the topology as input
- Controller sends forwarding state to switches
  - Using forwarding abstraction (OpenFlow)
- Controller is replicated for resilience
  - System is only “logically centralized”

Network of Switches and/or Routers

Software Defined Network (SDN)

Traditional Control Mechanisms



routing, access control, etc.

Control Program

Distributed algorithm running between neighbors

Global Network View

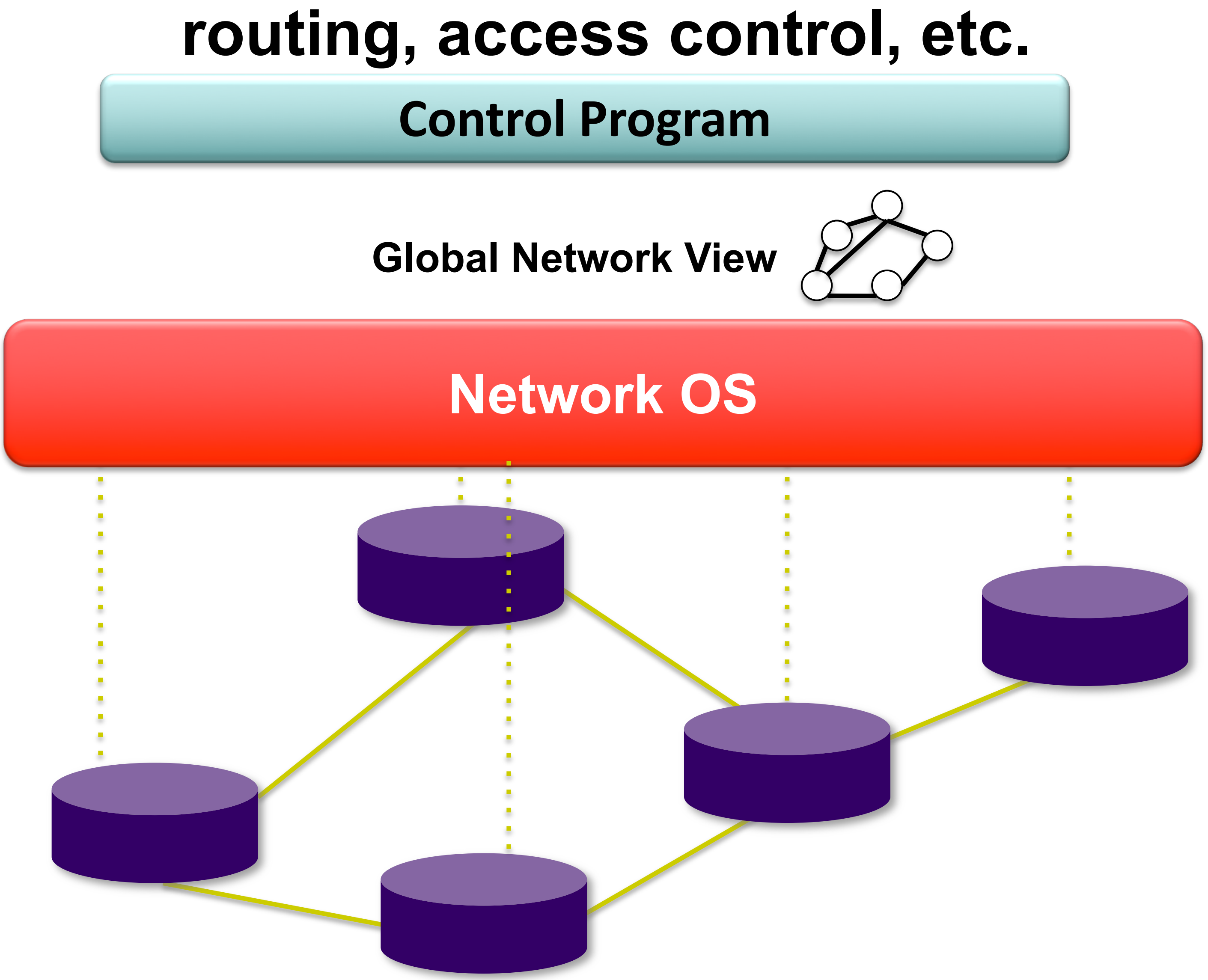
Complicated task-specific distributed algorithm

Network OS

Major Change In Paradigm

- Control program: **Configuration = Function(view)**
  - Configuration means set of forwarding entries
- Control mechanism now program using NOS API
- Not a distributed protocol, just a graph algorithm
  - All distributed algorithms in NOS
- Configurations are passed to switches by NOS

Software Defined Network (So Far)



routing, access control, etc.

Control Program

Global Network View

Network OS

Questions?

Separate Concerns With Abstractions

1. Be compatible with low-level hardware/software

   Need an abstraction for general forwarding model

2. Make decisions based on entire network

   Need an abstraction for network state

3. **Compute configuration of each physical device**

   Need an abstraction that simplifies configuration

Otherwise, control program must compute forwarding entries for every switch in network….

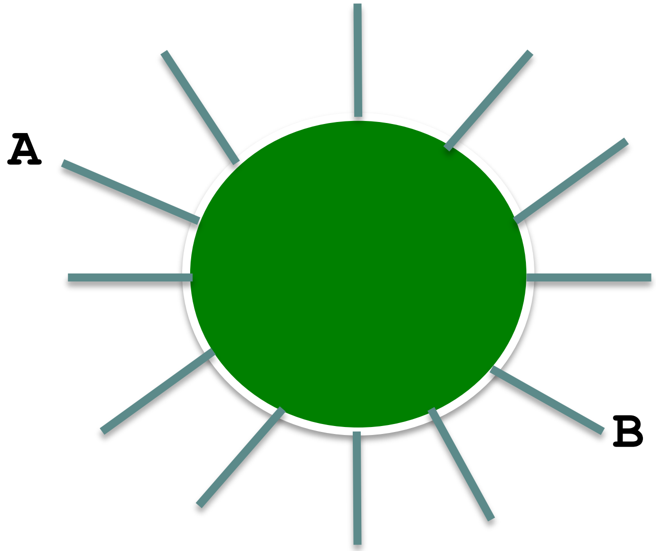
Abs#3: Specification Abstraction

- Control mechanism specifies desired behavior
  - Whether it be isolation, access control, or QoS

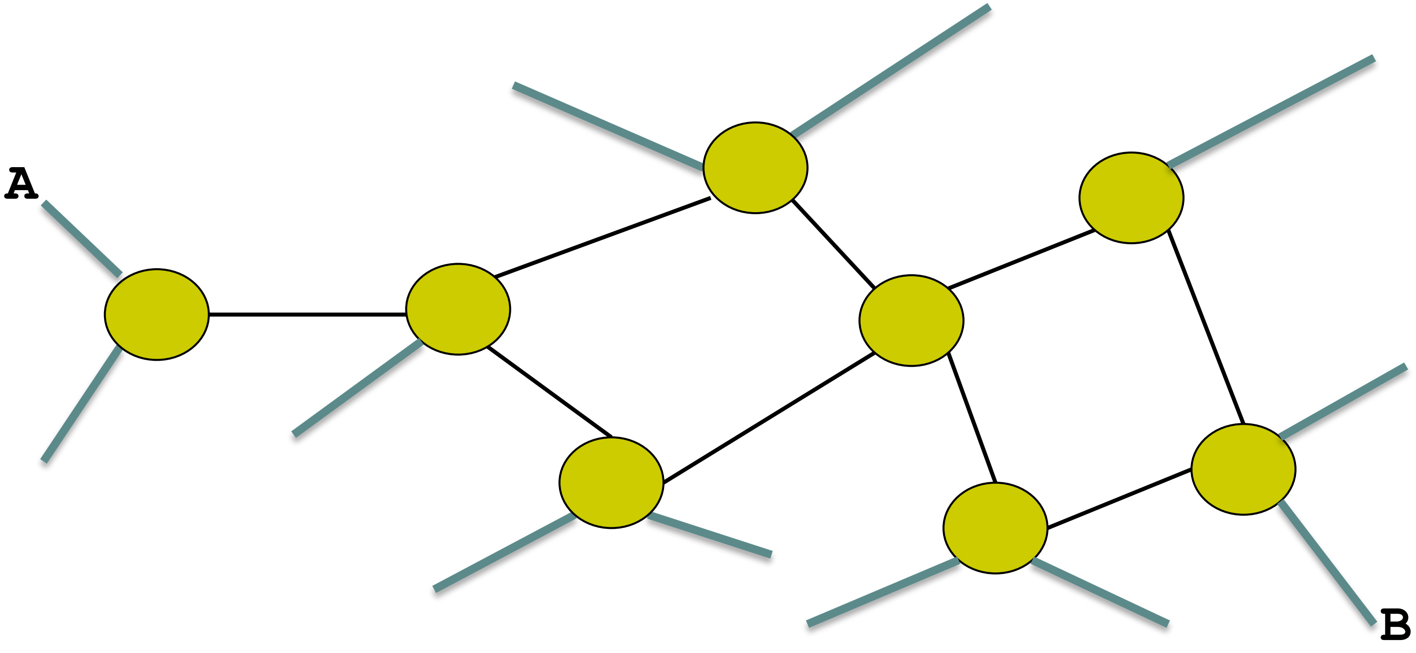
- It should not be responsible for ***implementing*** that behavior on physical network infrastructure
  - Requires configuring the forwarding tables in each switch

- Proposed abstraction: **abstract view** of network
  - Abstract view models only enough detail to *specify goals*
  - Will depend on task semantics
  - Now called "intention-based" networking
  - Think of this as providing a compiler...

Simple Example: Access Control

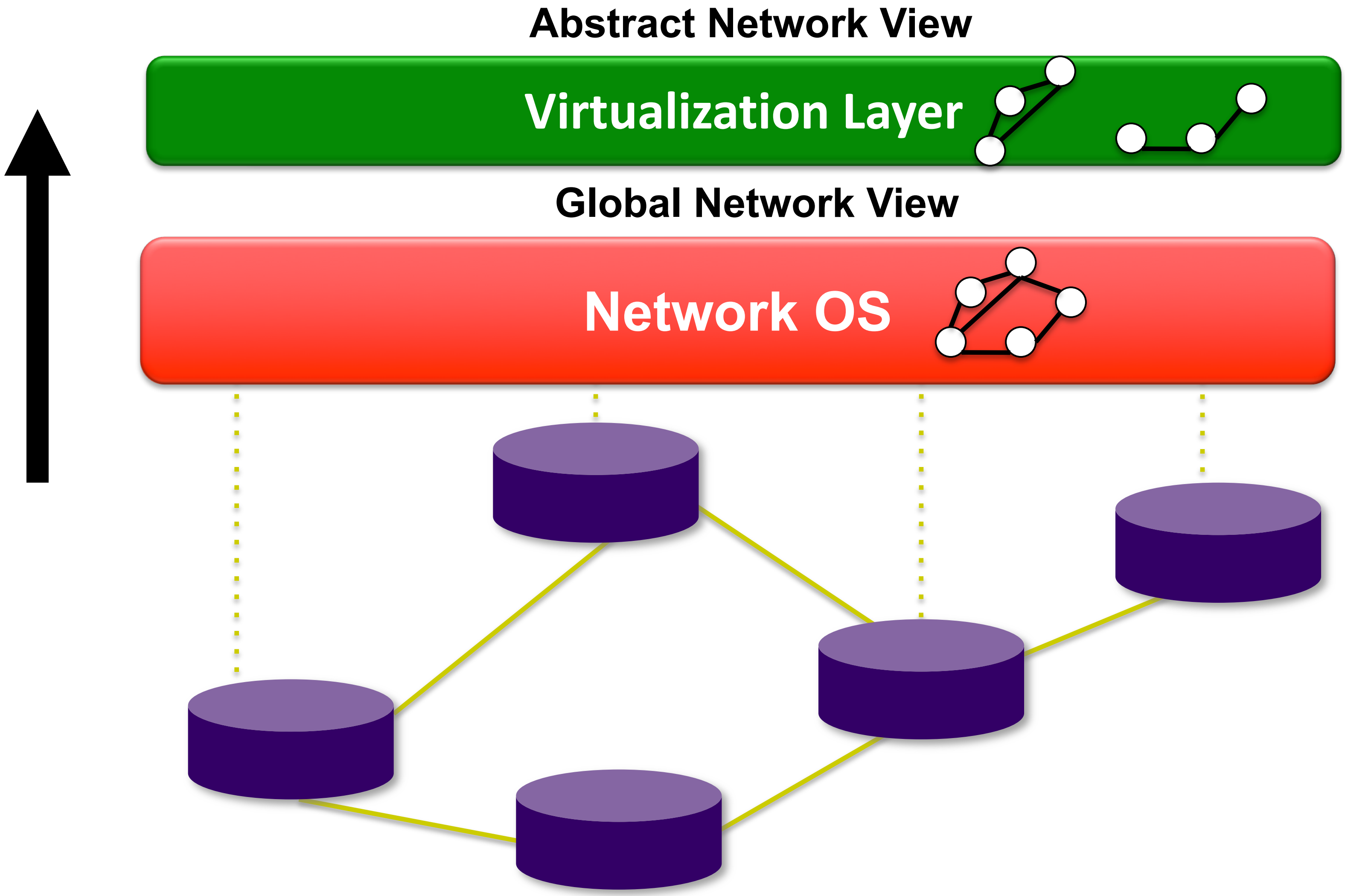


Abstract
Network
View

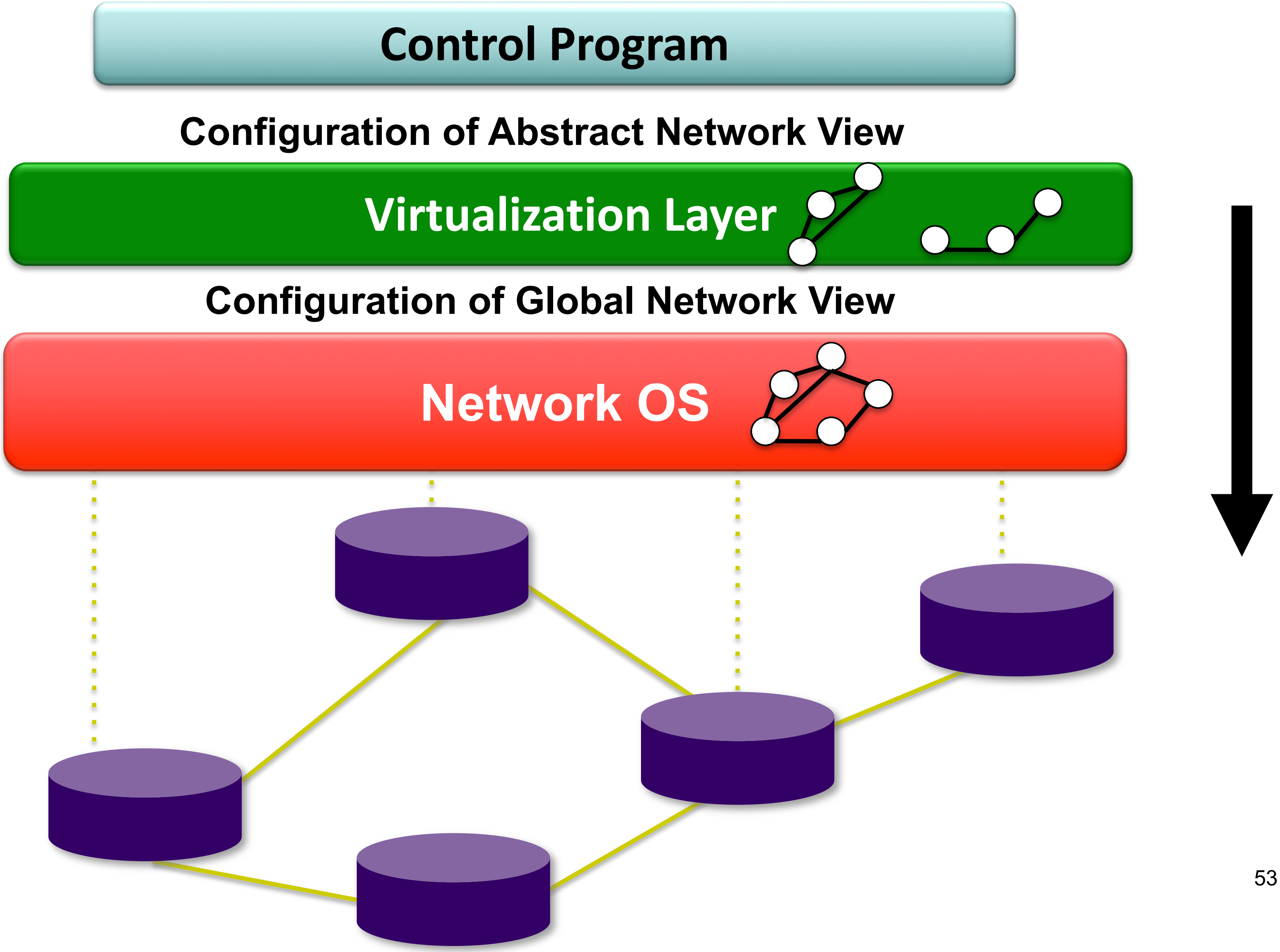


Global
Network
View

Software Defined Network



Software Defined Network



Control Program

Configuration of Abstract Network View

Virtualization Layer

Configuration of Global Network View

Network OS

What This Really Means

Routing Application

- Look at graph of network
- Compute routes
- Give to SDN platform, which passes on to switches
- *Graph algorithm, not distributed protocol*

Access Control Application

- Control program decides who can talk to who
  - E.g., based on security category
- Pass this information to SDN platform
- Appropriate ACL flow entries are added to network
  - In the right places (based on the topology)
- *The control program that decides who can talk to whom doesn't care what the network looks like!*

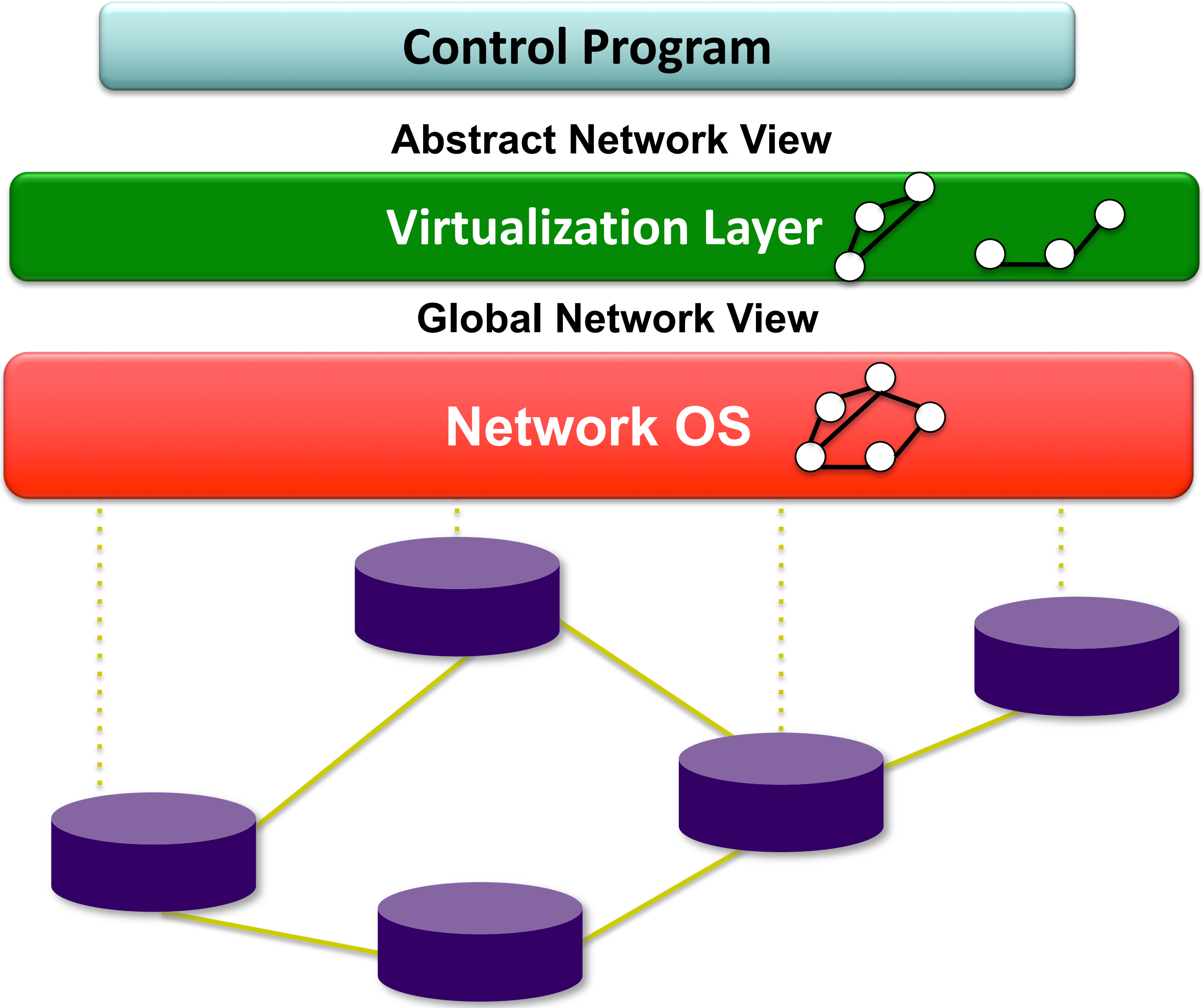
Clean Separation Of Concerns

- **Control program:** specify goals on abstract view
  - Driven by **Operator Requirements**

- **Virt. Layer:** abstract view **←→** global view
  - Implements goals on network (as in global view)
  - Driven by **Specification Abstraction** for particular task

- **NOS:** global view **←→** physical switches
  - API: driven by **Network State Abstraction**
  - Switch interface: driven by **Forwarding Abstraction**

SDN: *Layers* For The Control Plane



Questions?

Abstrns Don’t Remove Complexity

- NOS, Virtualization are complicated pieces of code
- SDN merely localizes the complexity:
  - Simplifies interface for control program (use-specific)
  - Pushes complexity into ***reusable*** code (SDN platform)
  - This is the trajectory of computer science
- This is the big payoff of SDN: modularity!
  - The core distribution mechanisms can be reused
  - Control programs only deal with their specific function

Why Is SDN Important?

- As a design:
  - It is more modular, enabling faster innovation
  - Control programs become very simple!
- As an academic endeavor:
  - Provides abstractions that enable systematic reasoning
  - Can reason about control program, without looking at each switch…
- As a change in the ecosystem:
  - Open switch interfaces reduce vendor lock-in
  - Not clear that this will happen (why?)

Common Questions About SDN?

- Is SDN less scalable, secure, resilient,…?
- Can SDN be extended to the WAN?
- Is OpenFlow the right fwding abstraction?
- Is SDN incrementally deployable?

Common Questions About SDN?

- Is SDN less scalable, secure, resilient,…? **No**
- Can SDN be extended to the WAN? **Yes**
- Is OpenFlow the right fwding abstraction? **No**
- Is SDN incrementally deployable? **Yes**

  How can this be?

What About Deployment?

- Most of SDN’s design is in software on servers
  - NOS and virtualization layer run on servers
  - Deploying these components is easy!
- But all routers must support OpenFlow
  - To provide information to the SDN controllers
  - To receive flow entries from the SDN controllers
- Requires replacing all routers in network
  - Routers are closed/proprietary, vendors won’t upgrade
- So the question is…

How Did We Get This Deployed?

- Get everyone to buy new OpenFlow switches?

- That is a completely ludicrous approach
  - *Though one we believed in at Nicira for a while*

- So, how did we deploy SDN?
  - Without them buying new switching hardware
  - And in some cases not even talking to the networking team at the company….

- Think about it....

Fact #1

- Most additional control plane functionality can be implemented at the edge
  - Access control, LAN Isolation, traffic engineering,…
  - Think about this for a second..
- Network core merely needs to deliver packets
  - Pre-SDN networking technologies pretty good at this
  - i.e., control plane for core only has its original task
- So only need to add SDN at network edge…
- This edge/core split arises in other contexts
  - E.g., MPLS, which has been widely adopted

Fact #2

- The operators who were crying were from large multitenant datacenters….

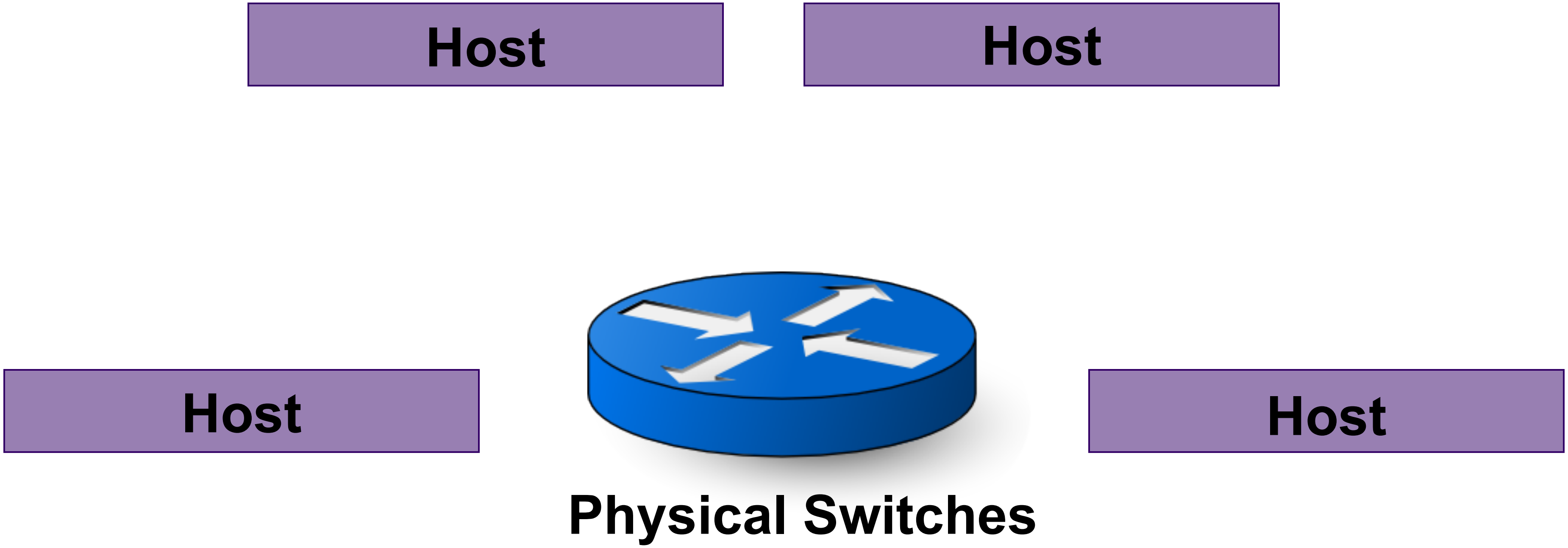
- They run hypervisors on their hosts, to support VMs initiated by tenants

- These two facts gave us an opening….

Deployment In Virtualized Datacenters

- Virtualization (VMs) is supported by hypervisors
- Hypervisors use virtual switches to connect VMs
- Make this virtual (software) switch SDN-compatible
  - And you'll be able to deploy SDN without any new HW
- Open vSwitch was an OpenFlow-capable vSwitch
  - Developed by Nicira, inserted into in Linux, Xen, etc.
- SDN now deployable without any HW deployment!
- This applies only to multitenant datacenters
  - ***But they were our only customers!***

Network in Regular Setting



Host

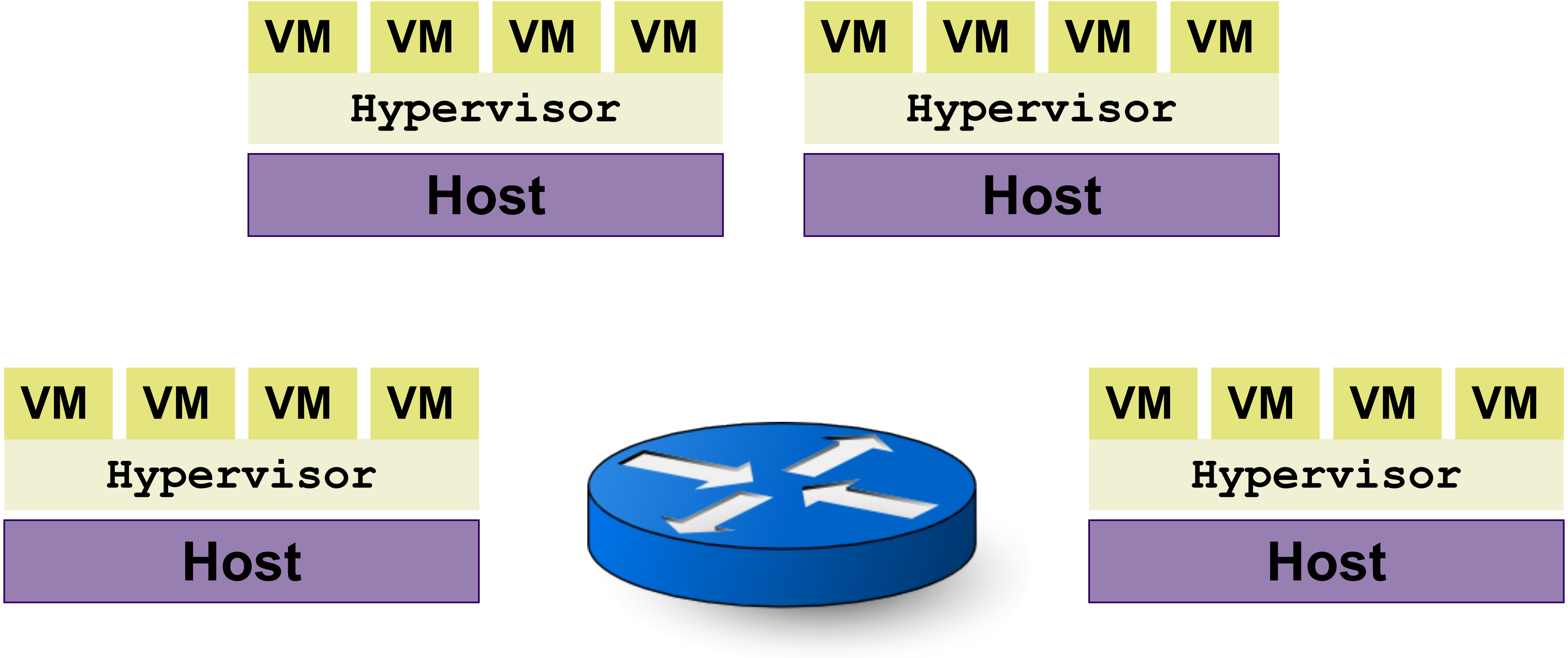
Host

Host

Host

Physical Switches

Network in Virtualized Setting



VM VM VM VM

Hypervisor

Host

VM VM VM VM

Hypervisor

Host

VM VM VM VM

Hypervisor

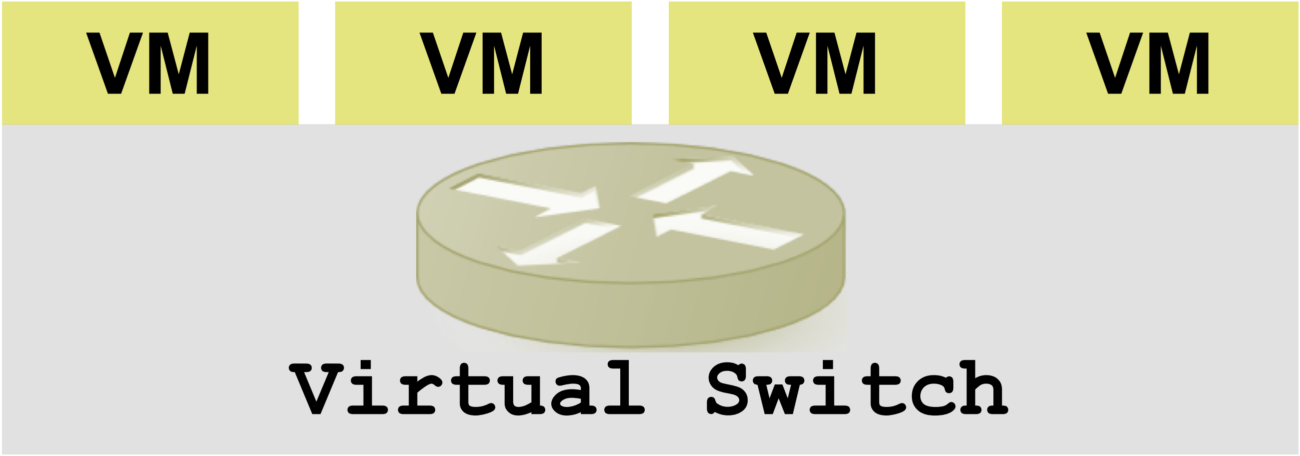
Host

VM VM VM VM

Hypervisor

Host

Virtual Switches (vSwitch)

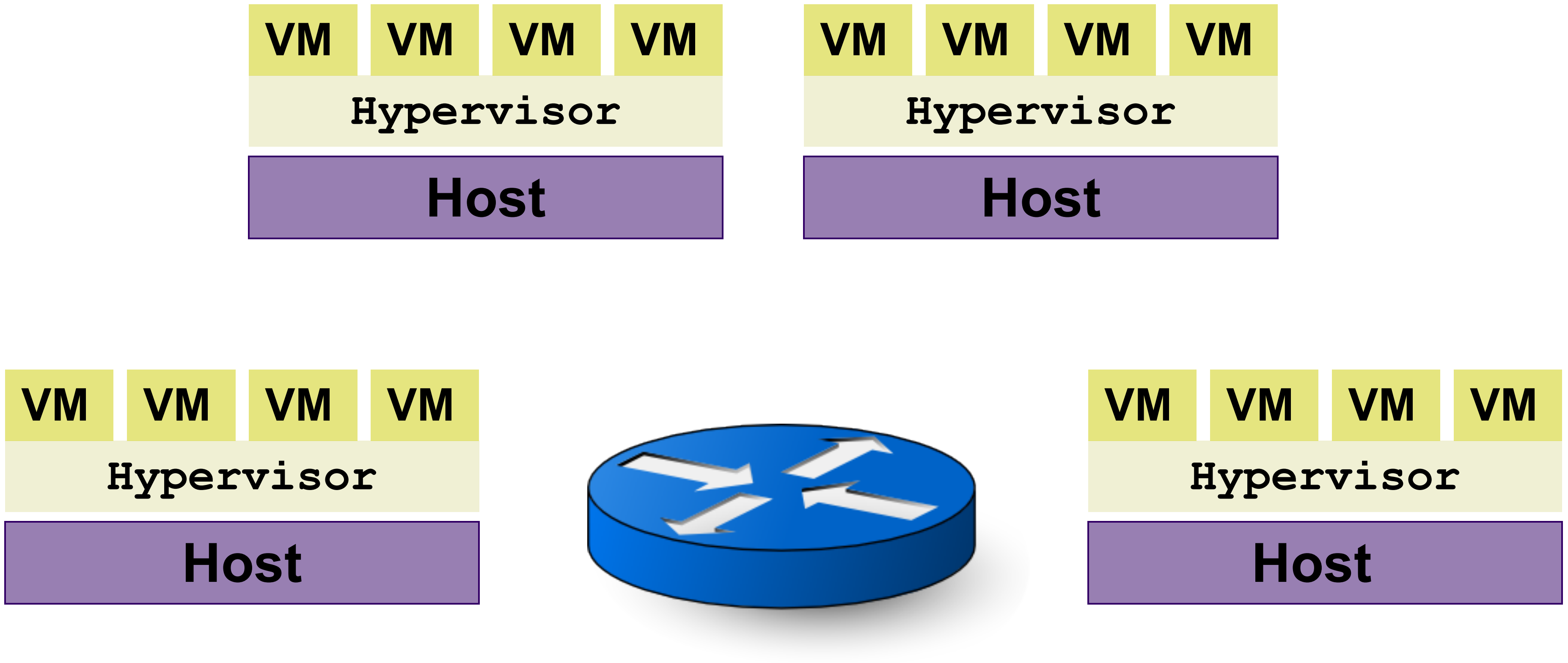


VM VM VM VM

Virtual Switch

- vSwitch is first-hop switch for all VMs
  - vSwitch sends packet to other VMs, or to physical network
- vSwitch is a software switch
  - If it supports OpenFlow, can be controlled by NOS

Physical View of Virtualized Network



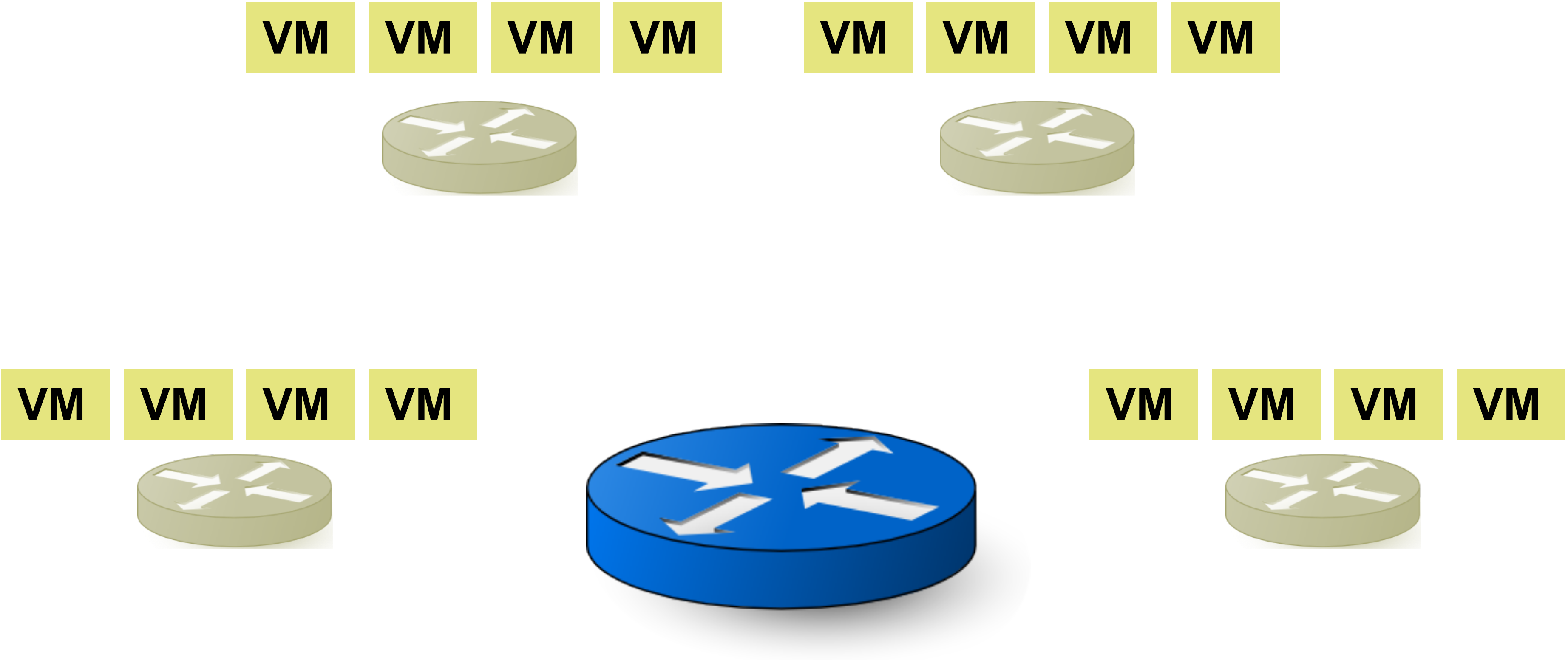
VM VM VM VM
Hypervisor
Host

VM VM VM VM
Hypervisor
Host

VM VM VM VM
Hypervisor
Host

VM VM VM VM
Hypervisor
Host

Logical View of Virtualized Network



All edge switches are vSwitches

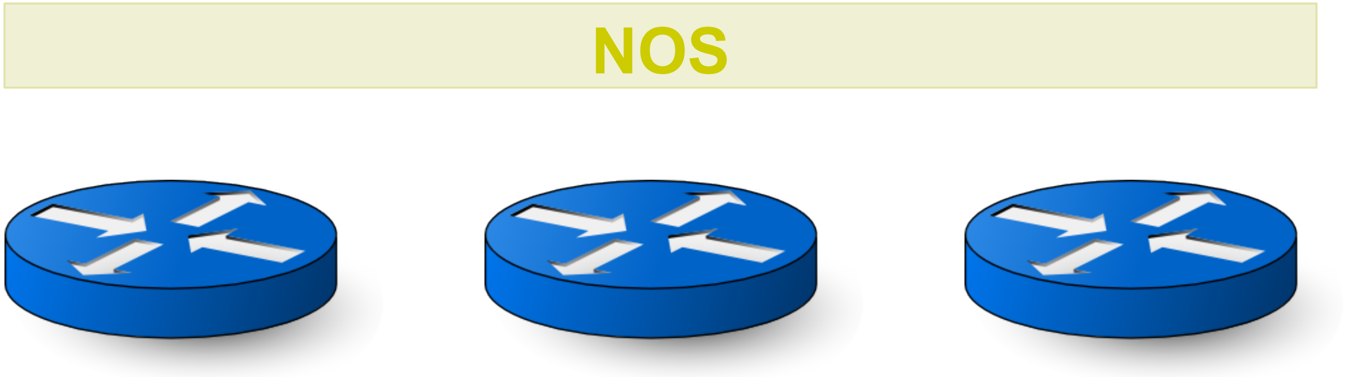
vSwitches are Sufficient in VDCs

- vSwitches enough to implement most CP functions
  - Access control, QoS, mobility, migration, monitoring,…

- Physical network becomes static crossbar
  - Crossbar: just delivers packets from edge-to-edge
  - Simple to implement and manage
  - Mostly static (only responds to changes inside core)

- Edge handles all dynamic/configured functions
  - Tracking VM movement
  - Access control policies
  - …

Managing Physical Network

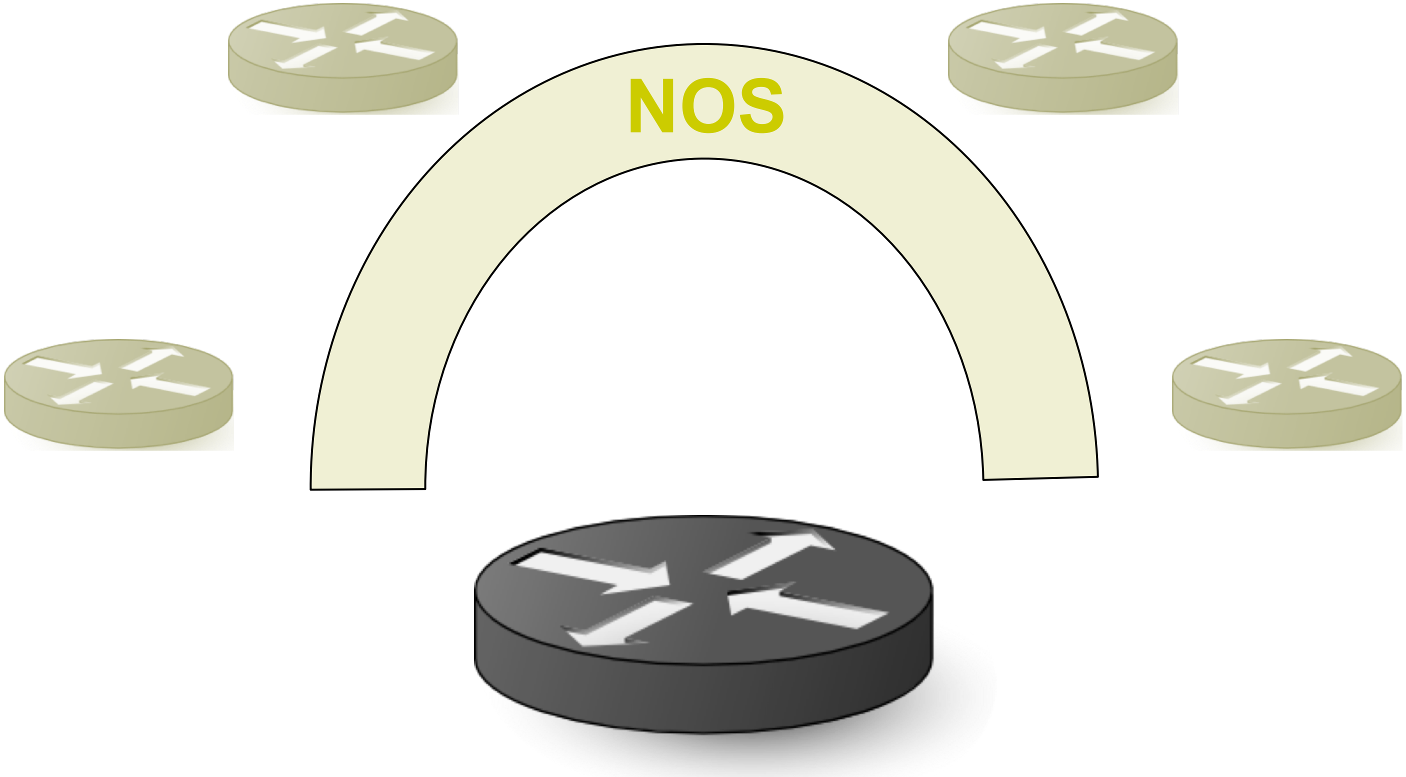


NOS

Physical Switches

Managing Virtualized Network

NOS only needs to control vSwitches at edge



Physical network is logical crossbar

vSwitches as Insertion Point

- Can insert new functionality into datacenters with
  - Hypervisors with OpenFlow-enabled vSwitch
  - Network Operating System (on servers)
- No change to physical infrastructure
  - Legacy hosts
  - Legacy network components
- This last issue isn’t just a technical point
  - The network remaining completely unchanged is huge!

Deploying SDN in VDCs

- Because the network is completely unchanged, the deployment can be managed by the compute team, not the networking team
  - Which have very different perspectives

- Networking team:
  - Very conservative, need to “not fail”, HW-oriented
- Compute team:
  - More nimble, need to deliver functionality, SW-oriented

- Initial SDN deployments were not network-driven
  - Which is what made them possible!

Questions?

About this lecture, or anything else...