Thinking Architecturally (80 Minutes Inside Scott's Head)

EE122 Fall 2012

Scott Shenker

http://inst.eecs.berkeley.edu/~ee122/

Materials with thanks to Jennifer Rexford, Ion Stoica, Vern Paxson and other colleagues at Princeton and UC Berkeley

Announcements

- Roughly 60 of you have not yet participated
  - If you aren't sure, check bspace
- I will not schedule special office hours for you
  - Show up at normal hours, or ask a question in class
- I only have office hours tonight and Thursday
  - And I will be doing some reviews next week
  - Details on Thursday….

Agenda

- MPLS primer
- Architecture vs Engineering
- Review of SDN
  - And its implications
- Another functional decomposition
  - And its implications

Warning!

- These are recent architectural developments
- Still trying to work them through
- Will present three different approaches
- Not yet clear how they fit together
- *If you understand this lecture, you will know more about Internet architecture than 99% of networking researchers….*

Rules of Engagement….

- If you don’t understand, ask….

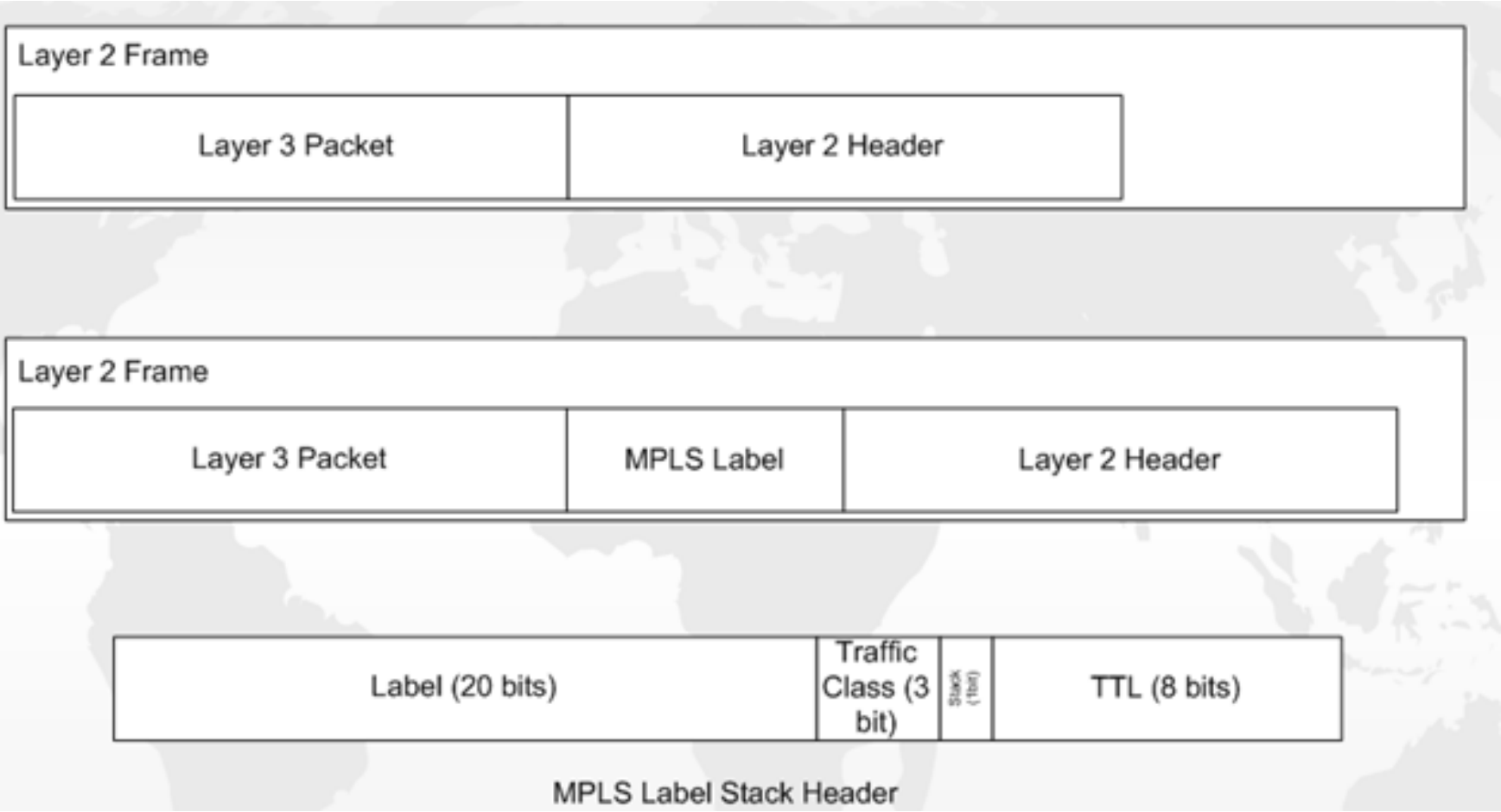
MPLS

Necessary background for today

Multiprotocol Label Switching (MPLS)

- Operators wanted more flexibility in routing traffic
  - Normal routing just used destination address…
- Wanted ability to route on larger aggregates
  - First decide if flow belongs to aggregate, then route agg.
  - Example: all traffic from LA to NY follow same path
- Wanted ability to route at finer granularity
  - Not all packets with same destination take same path
- Solution: insert a “label” before IP header
  - Route based on that label

MPLS Header


Layer 2 Frame
Layer 3 Packet
Layer 2 Header
Layer 2 Frame
Layer 3 Packet
MPLS Label
Layer 2 Header
Label (20 bits)
Traffic Class (3 bit)
Stack (1bit)
TTL (8 bits)
MPLS Label Stack Header

Using MPLS

- Make a distinction between edge and core routers
- Edge routers inspect IP header, insert MPLS label
- Core routers route based on MPLS label
- Must set up forwarding state for MPLS labels
  - Done in a variety of ways, for a variety of goals
  - Supporting failover paths, TE,…

MPLS is widely used

- Sometimes used for traffic engineering
- But mostly used for VPNs
  - Where all a customer's traffic is given a single label
- Extremely important practically, not intellectually
  - Because it is not tightly tied to a single purpose
  - Each use is ad hoc, rather than an overall paradigm
    - Sort of like the IPv6 flow ID: all mechanism, no policy

Architecture

Net Management and the E2E Principle

- The E2E principle assumes that hosts have goals
  - But completely ignores network operators
- There are some tasks that can't be implemented in the host because:
  - They require knowledge of the network (TE)
  - The host may not share the operator's goal (ACLs)
  - The host may be poorly configured (ACLs)
  - ….
- Network management requires new principles
  - SDN's layering is one such principle (reviewed today)

Architecture vs Engineering [Old Slide]

- Architecture:
  - The allocation of functionality and definition of interfaces among elements
- The Internet "architecture" is the decision about *what* tasks get done, and *where*:
  - In the network, or in the hosts
  - Engineering is more about *how* tasks get done
- These architectural decisions play a crucial role in scaling, heterogeneity, robustness, etc…
  - **This is what I spend my life worrying about**

Architecture in this course…

- E2E principle:
  - Do things in hosts when you can
- Fate sharing:
  - Store state where it is needed
- Layering:
  - Arises from functional decomposition of network delivery
  - *All modularity follows from decomposition, and modularity is the key to any overall design*

Decomposing Network Delivery

- First get bits on wire locally
  - No packets or packet headers, just low-level bits
- Then find a way to deliver packets locally
  - Requires packet headers, but only local addressing
- Then find a way to deliver packets globally
  - Requires packet headers with global addresses
- Then find a way to deliver packets reliably
  - Requires ACKing, etc.

Decomposition leads to layers

Applications

…built on…

Reliable (or unreliable) transport

…built on…

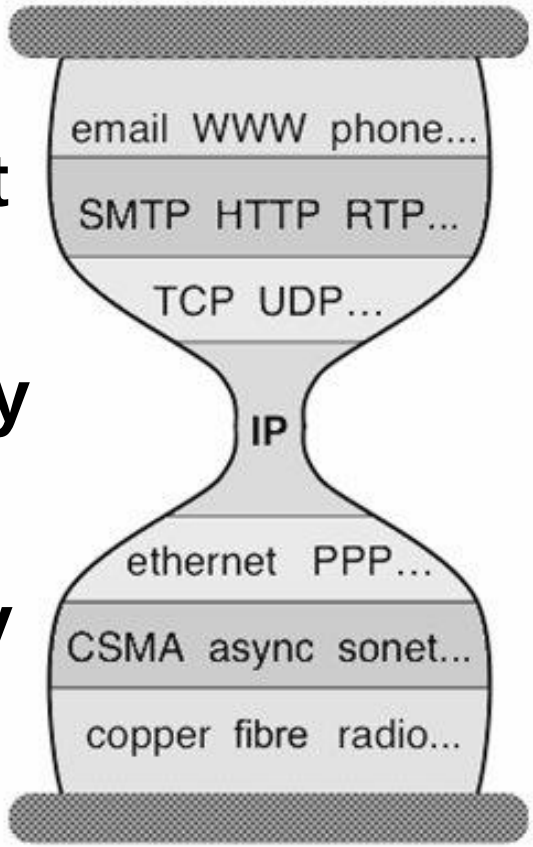
Best-effort global packet delivery

…built on…

Best-effort local packet delivery

…built on…

Physical transfer of bits



Decomposing the Control Plane

Control Plane Decomposition

- Goal: what are you trying to accomplish?
  - Access control: who can reach who?
  - Isolation: what are the virtual networks?
- Switch configurations needed to implement goal
  - What forwarding state is needed to achieve goals?
- Gathering information about topology
  - Need topology in order to compute forwarding state
- Telling switches what to do
  - Switches must be given the computed forwarding state

Controlling Switches

- Need protocol to communicate with switches
- Need standard way to specify configuration
- OpenFlow does both:
  - General controller-switch protocol
  - Specification of < header ; action > flow entries
    - What matches and actions are allowed
    - How are they expressed

Gathering Information About Topology

- Need a “network operating system”
  - NOX was the first
  - Many exist now: ONIX, Floodlight, Trema, POX,….
- The NOS runs on servers
- Communicates with each switch
- Can construct topology from switch information

Determining Appropriate Configuration

- Compute appropriate switch configurations
- Given specified control goals (next slide)…
- …and topology (previous slide)
- Computation depends on details of goals:
  - Access control
  - Traffic engineering
  - …

Deciding on your goals

- Operator decides what needs to be done
- Expresses these desires to control program
  - Through some UI
  - (not standardized, specific to control program)
- Which then communicates them to SDN platform
- The SDN platform then implements these goals
  - First computing appropriate configurations
  - Then communicating those configurations to switches

Interface for control program

- Control program needs interface to SDN platform
  - Must have some way of communicating intent
- Currently, we use an “abstract network view”
  - Intent is communicated by configuring simple network

Abstractions for Control Plane

Expression of Intent

…built on…

Abstract Network View

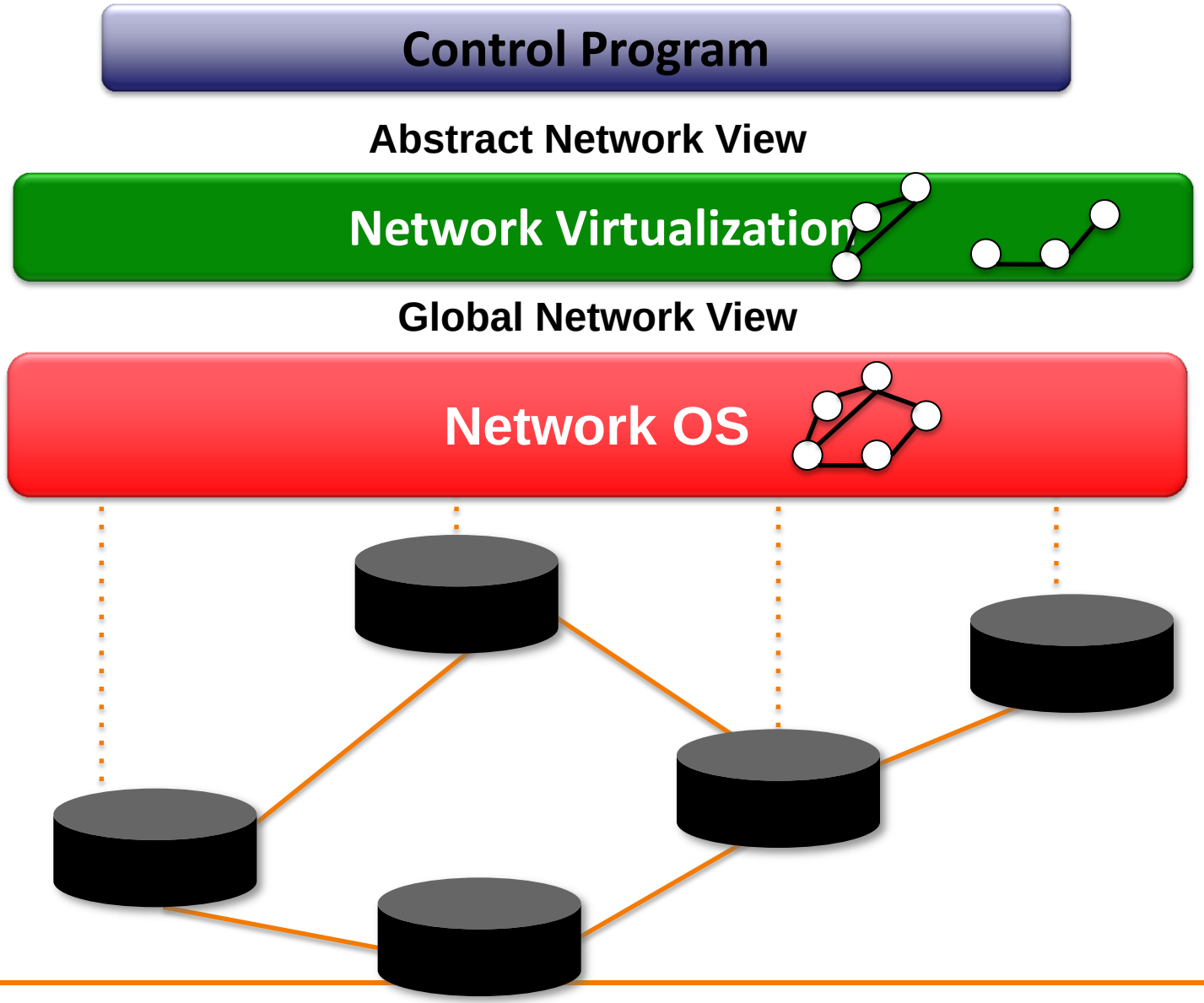
…built on…

Global Network View

…built on…

Physical Topology

SDN: *Layers* for the Control Plane



Clean Separation of Concerns

- **Control program:** express goals on abstract view
- **VirtualizationLayer:** abstract view ⬅➡ global view
- **NOS:** global view ⬅➡ physical switches

Abstractions for Control Plane

Expression of Intent (Control Program)

…built on…

Abstract Network View (Virtualization Layer)

…built on…

Global Network View (NOS)

…built on…

Physical Topology (Real Network)

This view is greatly oversimplified

- But I won’t tell you in what ways…
- …because it would just confuse the picture

Implications of SDN

Separation of Control/Data Plane

- Today, routers implement both
  - They forward packets
  - And run the control plane software
- SDN networks
  - Data plane implemented by switches
    - Switches act on local forwarding state
  - Control plane implemented by controllers
    - All forwarding state computed by SDN platform
- This is a technical change, with broad implications

Old Economic Model

- Bought HW/SW from single vendor
  - Closed control SW, proprietary forwarding HW
- HW deployment needs to be all from one vendor
- “Vendor lock-in”

New Economic Model

- Can buy HW from anyone (theoretically)
  - HW becomes interchangeable, if it supports OpenFlow
- Can buy SW from anyone
  - Runs on controllers, so doesn't need to run on switch
- SDN platform sold separately from control pgrms?
  - Would require stable and open platform interface
  - Currently a debate within ONF….
- Much less lock in (we hope)

Role of OpenFlow

- Architecturally: boring
  - Just a switch specification….zzzzzz
- Economically: hugely important
  - Could help break HW vendor lock-in

Changes the Testing Model

- Unit test hardware
  - The forwarding abstraction is well specified
  - Completely separate from control plane
- Use simulator to analyze large-scale control planes
  - External events are input to simulation
  - Can do regression testing
- Much wider testing coverage than is possible today
  - Today use physical testbeds, limited in size/scope

Which Then Changes Deployment…

- Before SDN, upgrades were scary and rare events
  - Code was not well tested (because it was hard to test)
  - And could only come from hardware vendor
- With SDN, upgrades can be easy and frequent
  - Much better testing
  - And the software can come from anyone
- Rate of innovation will be much greater!
  - Old: innovation mainly to get customers to buy new HW
  - SDN: innovation to make customers happier!

What is SDN's "Killer App"?

Current Networks: Topology = Policy

- Network topology determines:
  - Broadcast domains
  - Effectiveness of ACLs and firewalls
  - Equivalence classes
  - …
- When moving net to cloud, want to retain policy
  - But often don't have abstract formulation of that policy

SDN and Multitenant Datacenters

- SDN allows users to specify virtual topology
  - Each tenant can specify own topology (i.e., policy)
- SDN-enabled cloud network implements policies
  - SDN compiles set of policies into low-level configuration
- Tenants can migrate VMs to/from own network
  - Because policies driven from same topology
- ***This is what people are paying money for….***
  - ***Enabled by SDN's ability to virtualize the network***

Research Topics in SDN

Troubleshooting (now)

Scaling (later in lecture)

Troubleshooting SDNs?

- SDN networks introduce layers of abstraction
  - **Good**: makes it easier to write control programs
  - **Bad**: finding the problem when things go wrong
- "Going wrong" means packet behaviors do not agree with goals set by control program
  - Routing, access control, etc.
- Example: control pgrm forbids A from contacting B
  - But operators sees packets from A reaching B!

Must Answer Two Questions

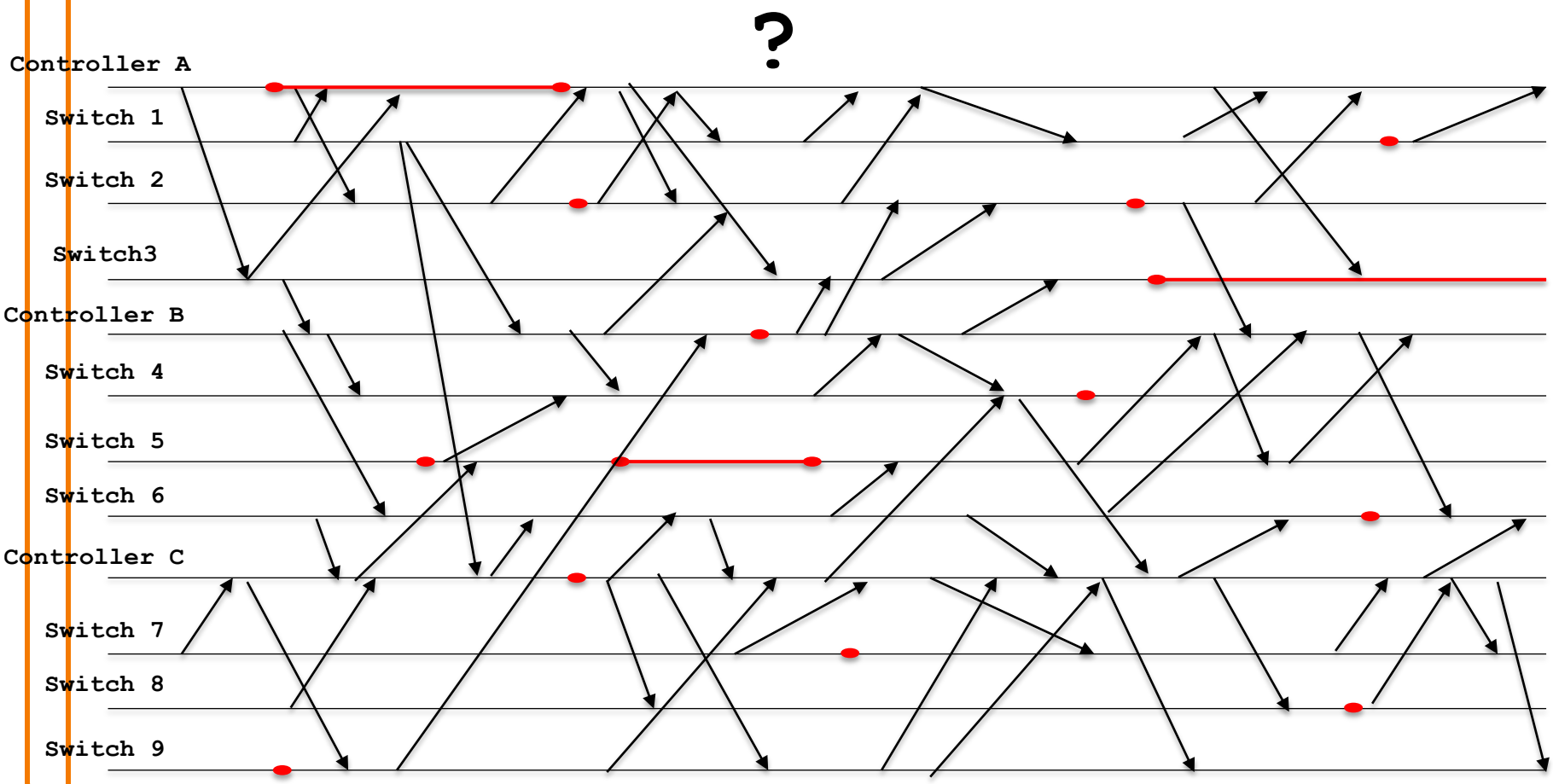
- What layer of abstraction caused this error?
  - Is the control program wrong?
  - Did the SDN platform introduce the error?
    - If so, in the NOS or the virtualization layer?
  - Did the physical switches malfunction?

Must Answer Two Questions

- What layer of abstraction caused this error?
  - Is the control program wrong? *[Other work]*
  - **Did the SDN platform introduce the error?**
    - **If so, in the NOS or the virtualization layer?**
    - **Or at what level of the hierarchy?**
  - Did the physical switches malfunction? *[Other work]*

- What external events triggered this error?
  - SDN platform is a complex distributed system
  - Subtle errors in responding to controller failures, etc.
  - ….

Identifying the Problematic Events



Leverage Control/Data Plane Split

- Replay control plane in simulator
- Iteratively remove external events from log
  - See if error still occurs
- Can programmatically identify ***a*** minimal causal set
- Focus debugging efforts on a small log of events
- *Lots of technical difficulties swept under the rug!*

Another Decomposition

Thinking architecturally

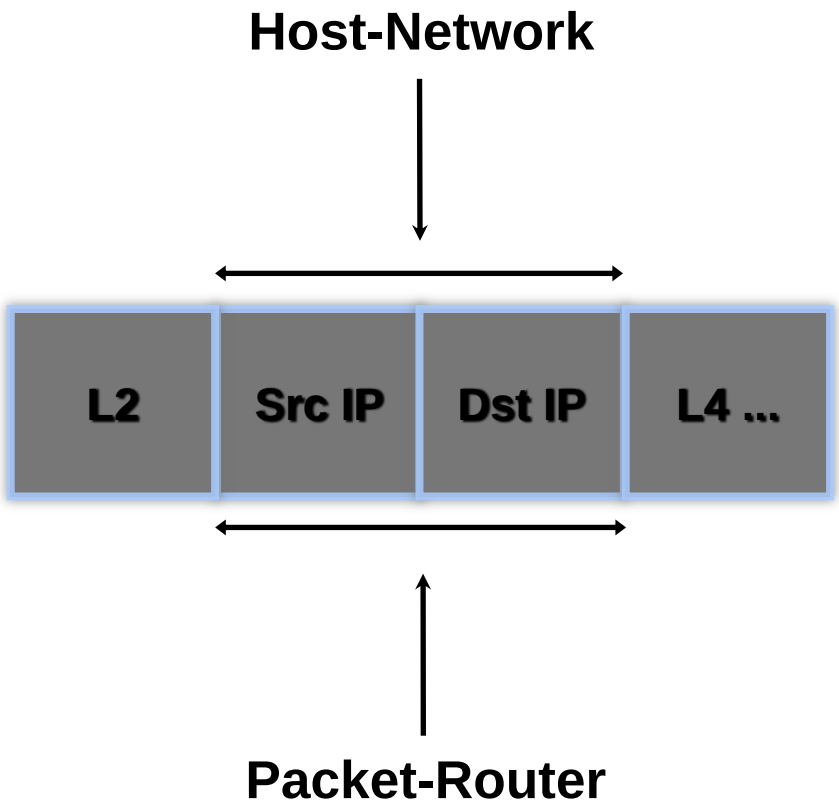
- Seeing how network services can be decomposed
- But there are many aspects to network service
- Dataplane considers how packets are handled
- Our control plane discussion only considers at how operator controls network
- What about another decomposition that takes hosts and forwarding into account

Three Important Logical Interfaces

- **Host to network**: destination, QoS requirements
- **Packet to router**: lookup key into forwarding state
- **Operator to network**: isolation, access control,…

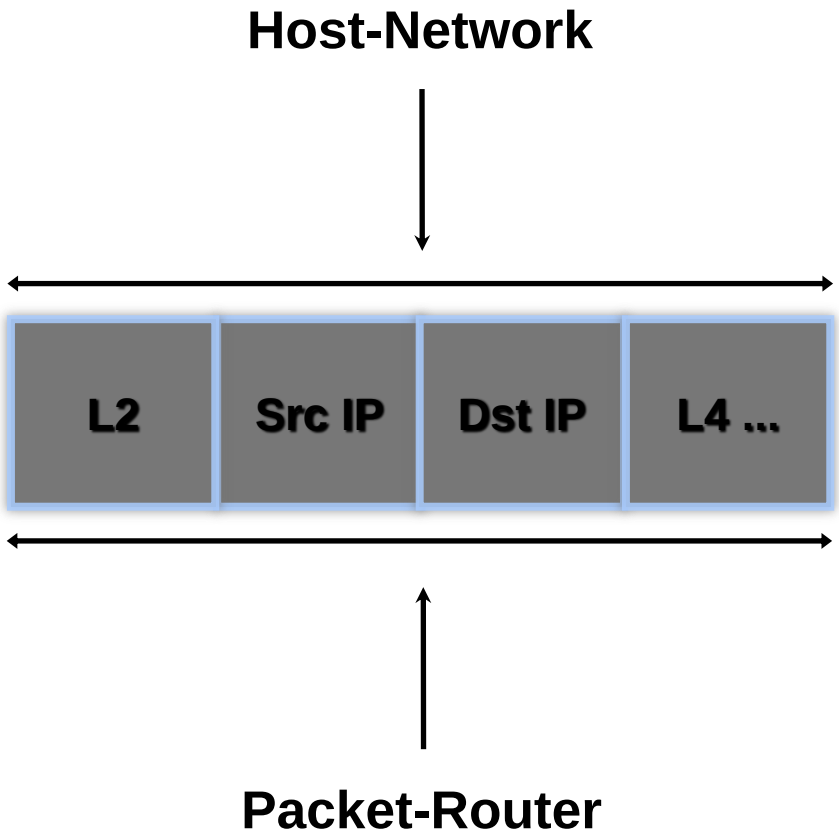
Interfaces in IP

- Host-network and packet-router interfaces are conflated.
  - They use same fields
  - Every router must interpret host demands
- Minimal operator-network interface
  - Routing algorithms, ACLs
  - Tacked on, not integral to overall design



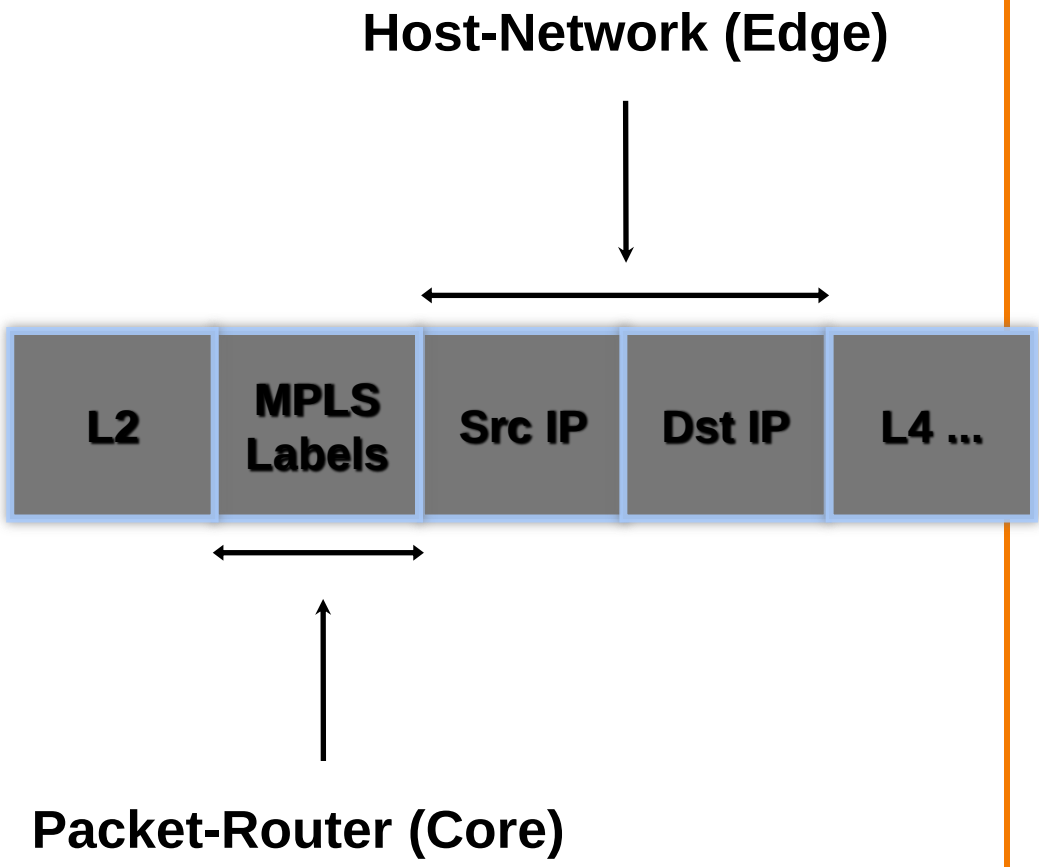
Interfaces in OpenFlow

- Host-network and packet-router interfaces still conflated
  - Same fields still used for both (just more fields)
  - All routers have fully general OF matching
- But SDN provides programmatic network-operator interface.
  - Huge step forward!



Interfaces in MPLS

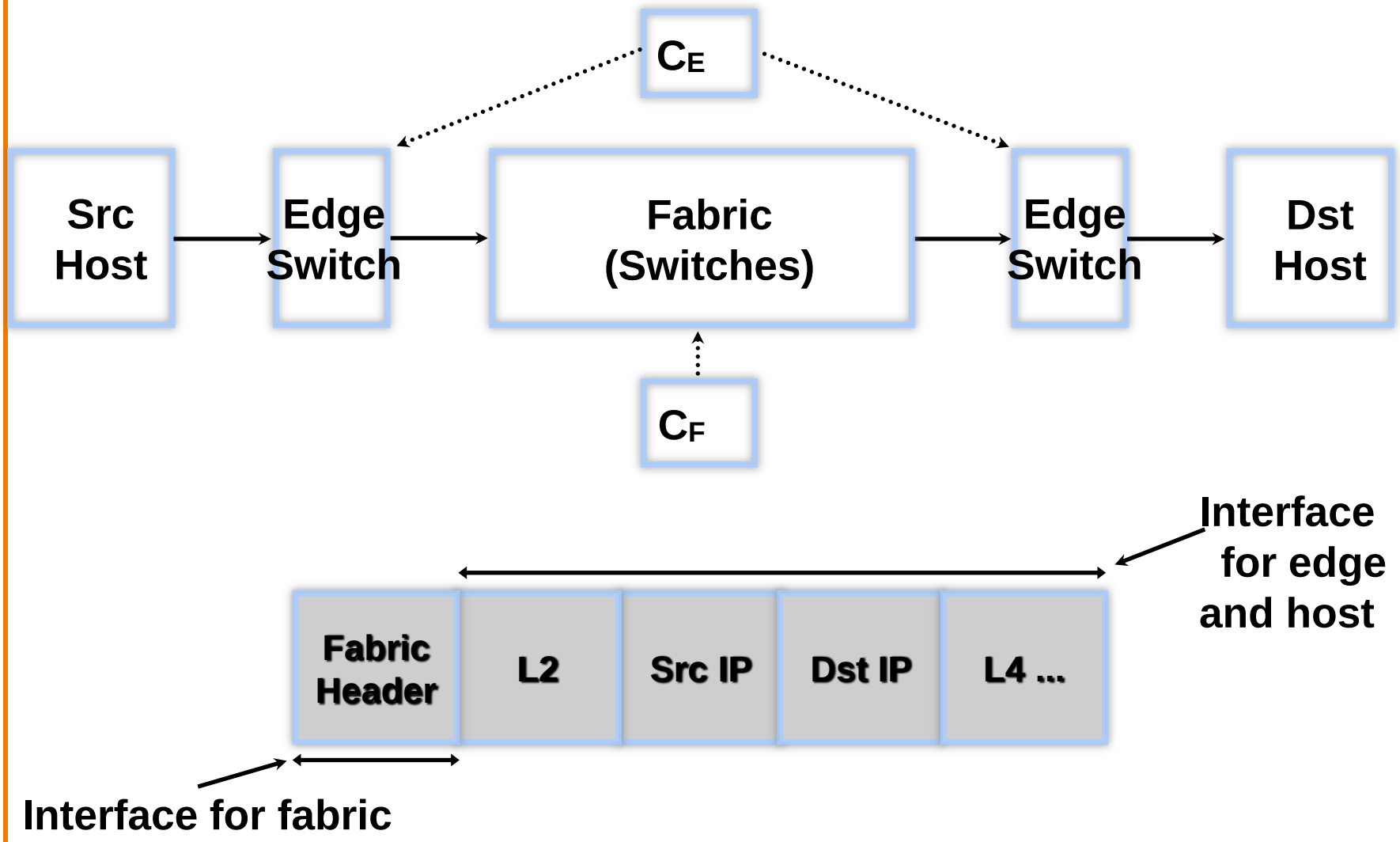
- Host-network and packet-router interfaces are distinct
  - Edge router looks at IP header
  - Internal switches look at MPLS label
- No fully general operator-network interface



Need to combine SDN and MPLS

- MPLS makes the important distinction between
  - Host-network interface
  - Packet-switch interface
- SDN provides full operator-network interface
- The combination of SDN and MPLS looks like this:
  - Network core is a “fabric” that delivers packets e2e
  - Network edge uses full IP header to make decisions
    - Then uses fabric interface to forward across network

SDN + MPLS = “Fabric”



Fabric Interface

- Fabric functions much like a distributed switch
  - Delivers packets to appropriate edge ports
  - “Transactional” changes for internal forwarding state
  - We will call these a “logical cross-bar” (LXB)
- Internal hardware is simple and future-proof:
  - Fabric only requires simple label forwarding internally
  - Only edge switches tied to host protocols
- Normal SDN needs general matching everywhere
  - This SDN approach only uses general matching at edge

Core-Edge Distinction

- Has far-reaching implications for SDN's future
  - Scalability
  - Deployability
  - Evolvability
  - ….

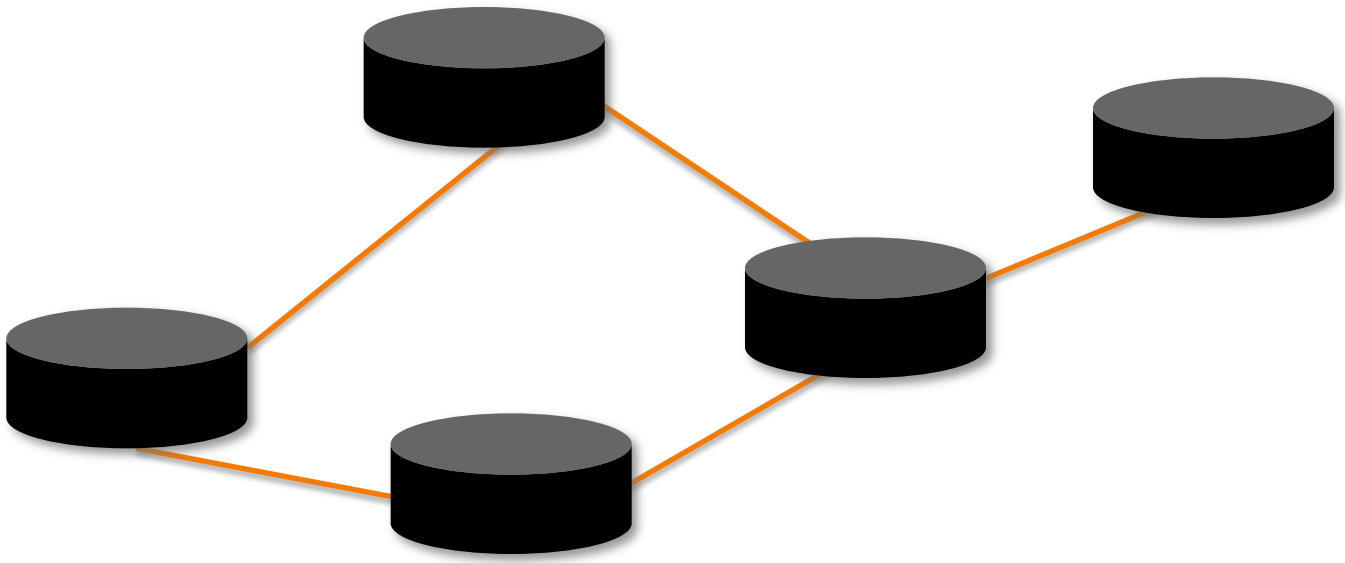
Scalability

Scaling SDN

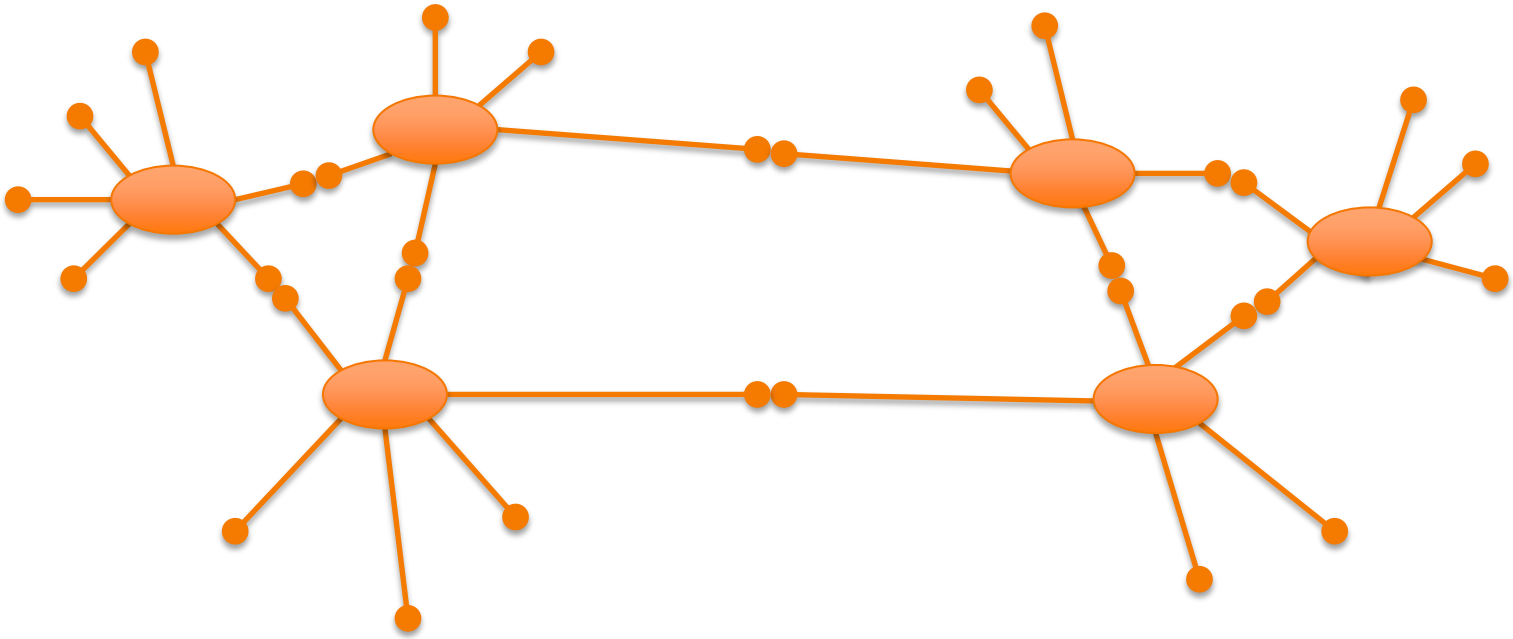
- What about global networks?
  - Can't have a flat organization of replicated controllers
  - Need more locality of control
- Can organize network into a hierarchical structure
  - Each unit of the hierarchy is a logical xbar **(LXB)**
- Each LXB
  - Given forwarding table by parent
  - Gathers topology from children
  - Controller implements **Table** on **Topology**

Example

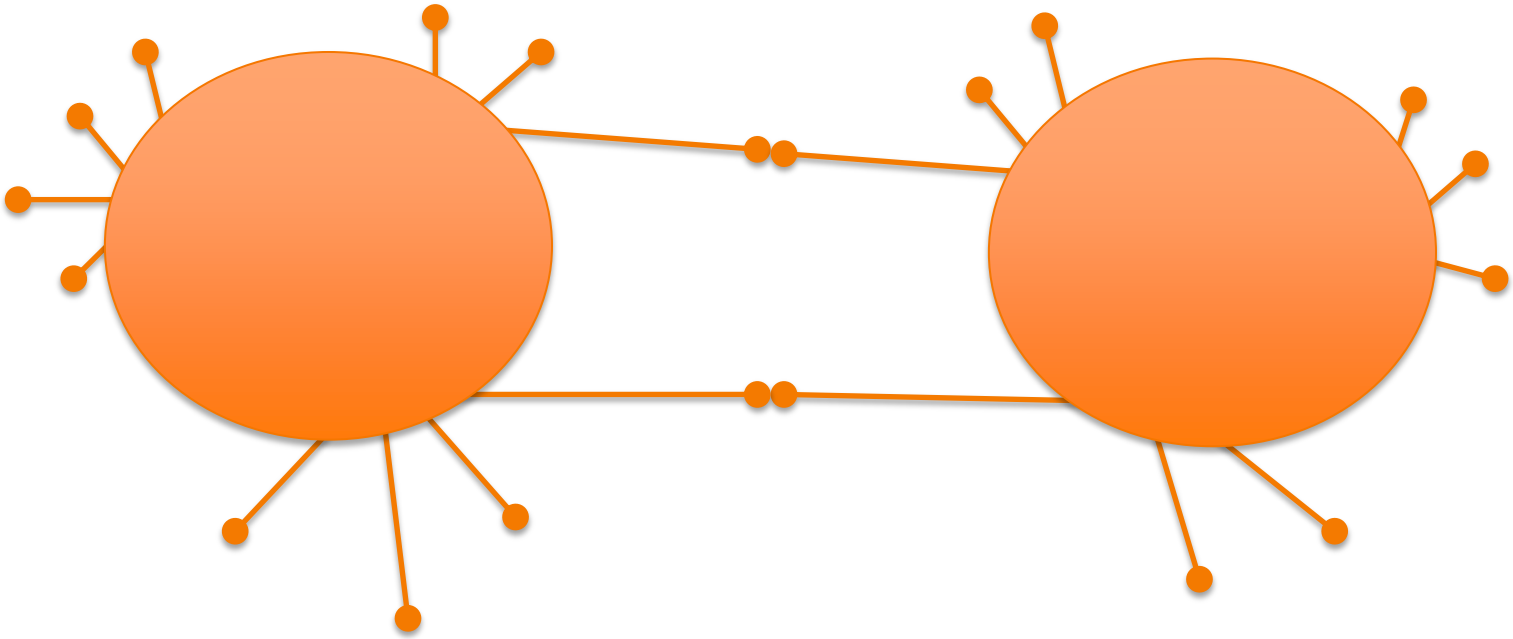
- Level 0 LXBs: physical switches



Level 1 LXBs: PoPs

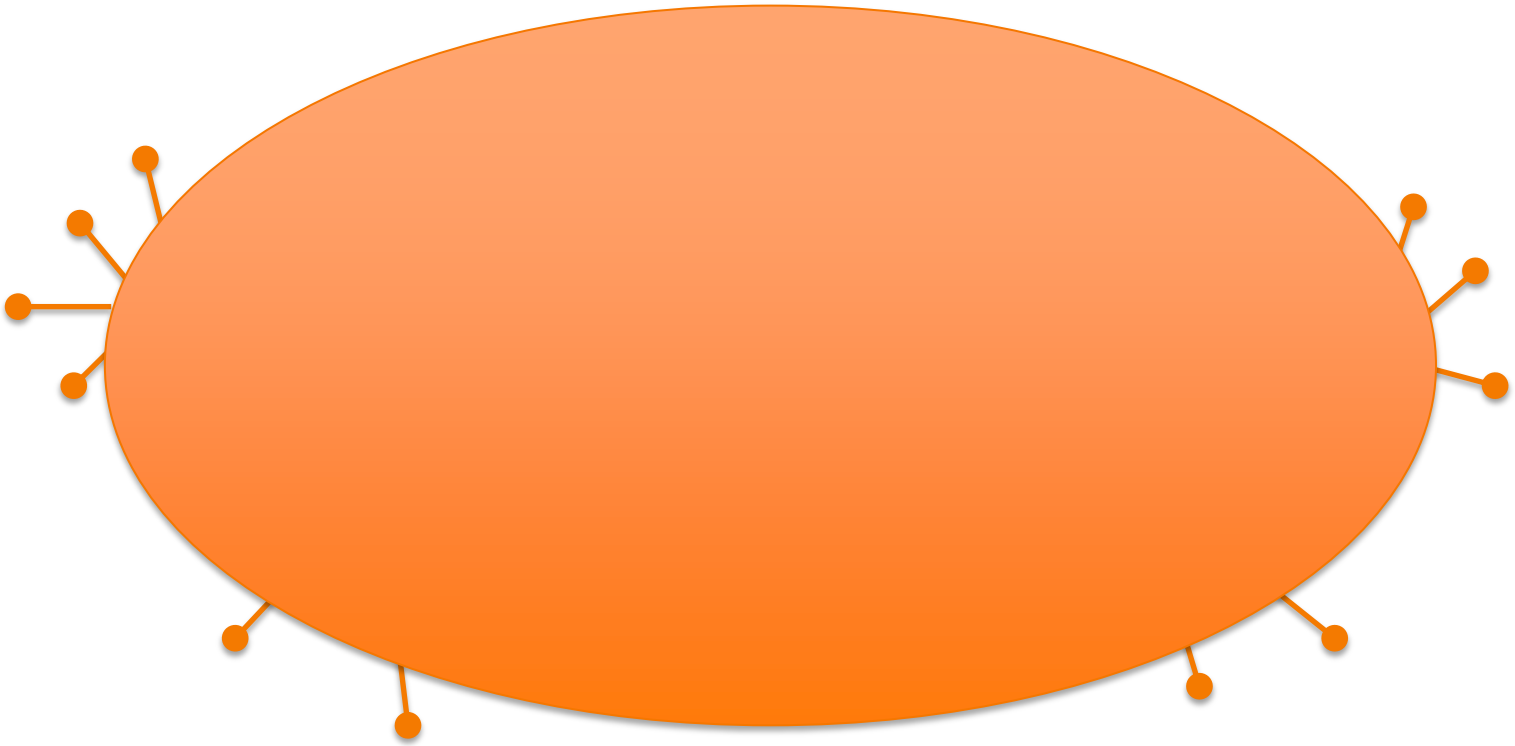


Level 2 LXBs: Regional Networks



Level 3 LXB: Global Network

- Ports attached to stub networks
- LXB forwards between them



What's New Here?

- Recursive application of basic "fabric" control
- Can use same controller codebase for each level
  - Topology state from children
  - Forwarding table from parent
  - Controller figures out to implement Table on Topology
- Overall structure has desirable properties:
  - Transactional model for network state changes
  - Bounded depth of computation for state changes

Deployability

Fact

- Most functionality can be implemented at edge
  - Access control
  - Isolation
  - ….
- This is implicit in the notion of “fabrics”
  - Fabric has simple service model
  - But makes sophisticated decisions at edge
    - Whether to forward
    - Where to forward
    - …

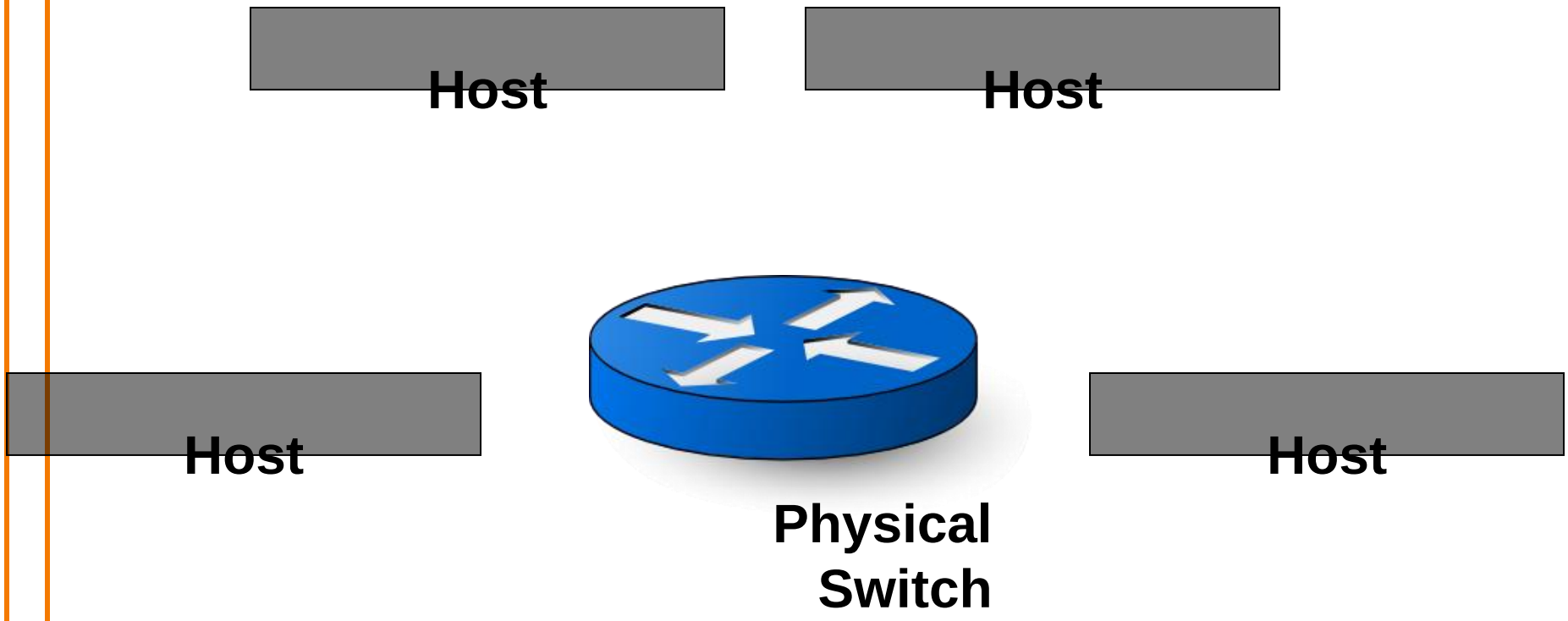
Current Networks Are Like Fabrics

- They provide edge-to-edge delivery
- To provide SDN-like benefits, merely add SDN-based switches at edge
- Use legacy equipment in core of network
- Simplifies deployment

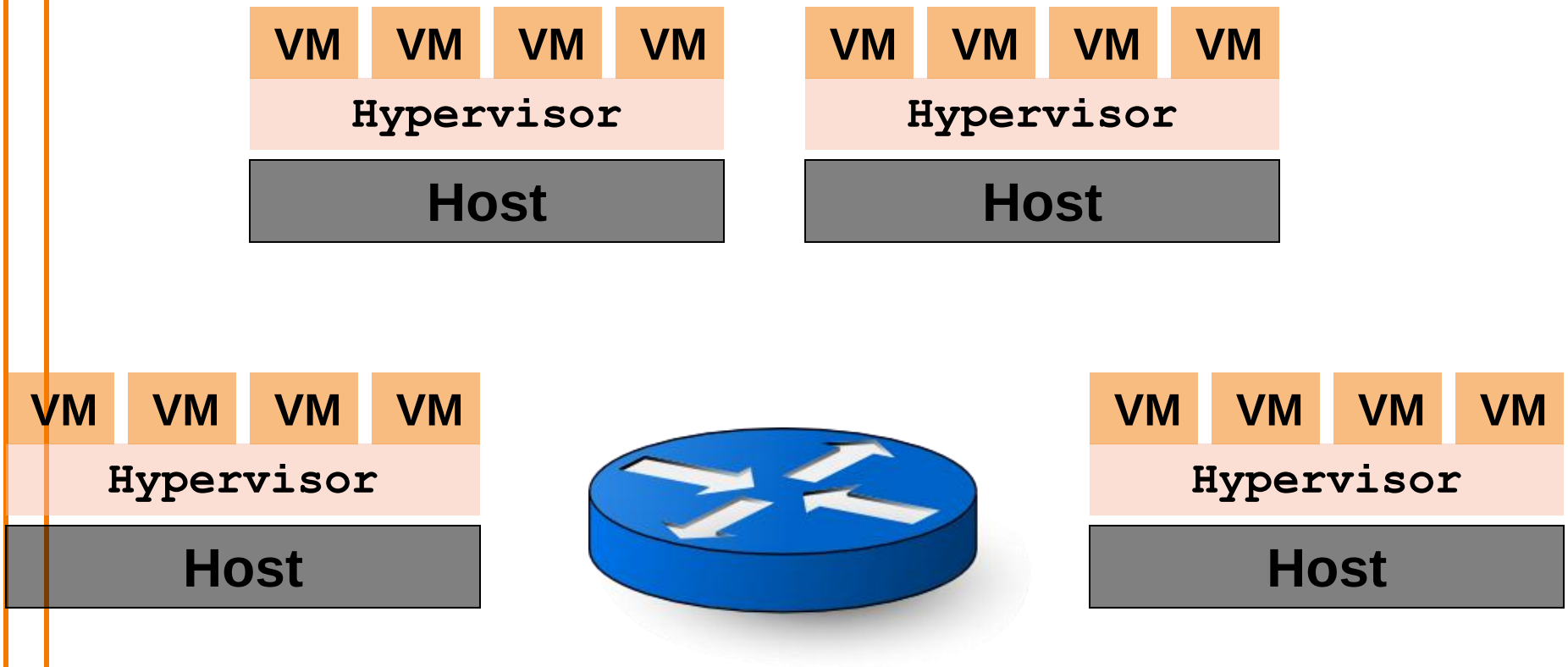
Virtualization makes this even easier…

- Virtualization (VMs) is supported by hypervisors
- Hypervisors use software switch to connect VMs
- Make this software switch SDN-compatible
  - And you'll be able to deploy SDN without any new HW
- Open vSwitch is now in Linux, Xen, and coming soon to other Oses and hypervisors
- SDN now deployable without any HW deployment!

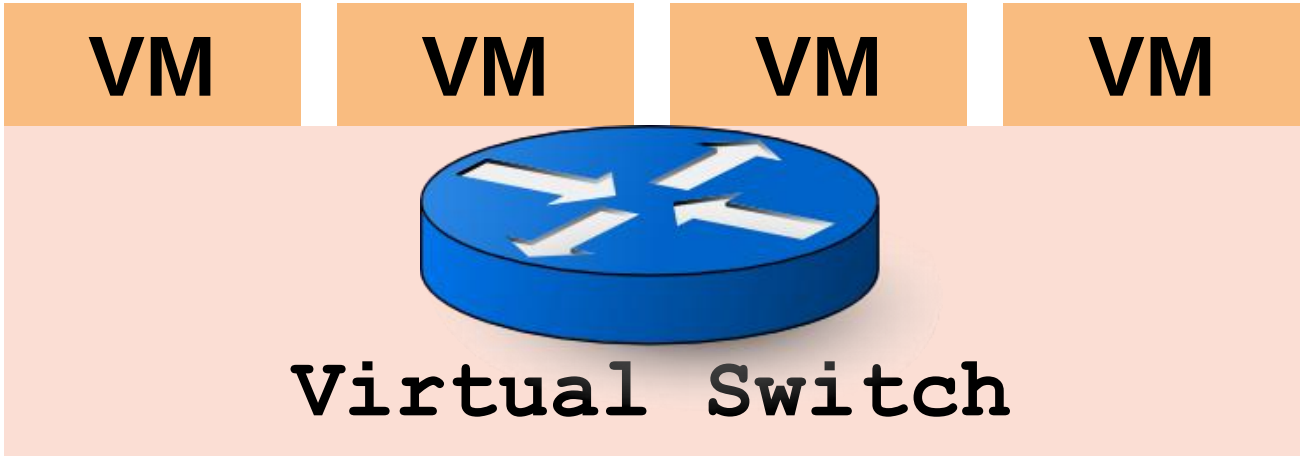
Network in Regular Setting



Network in Virtualized Setting

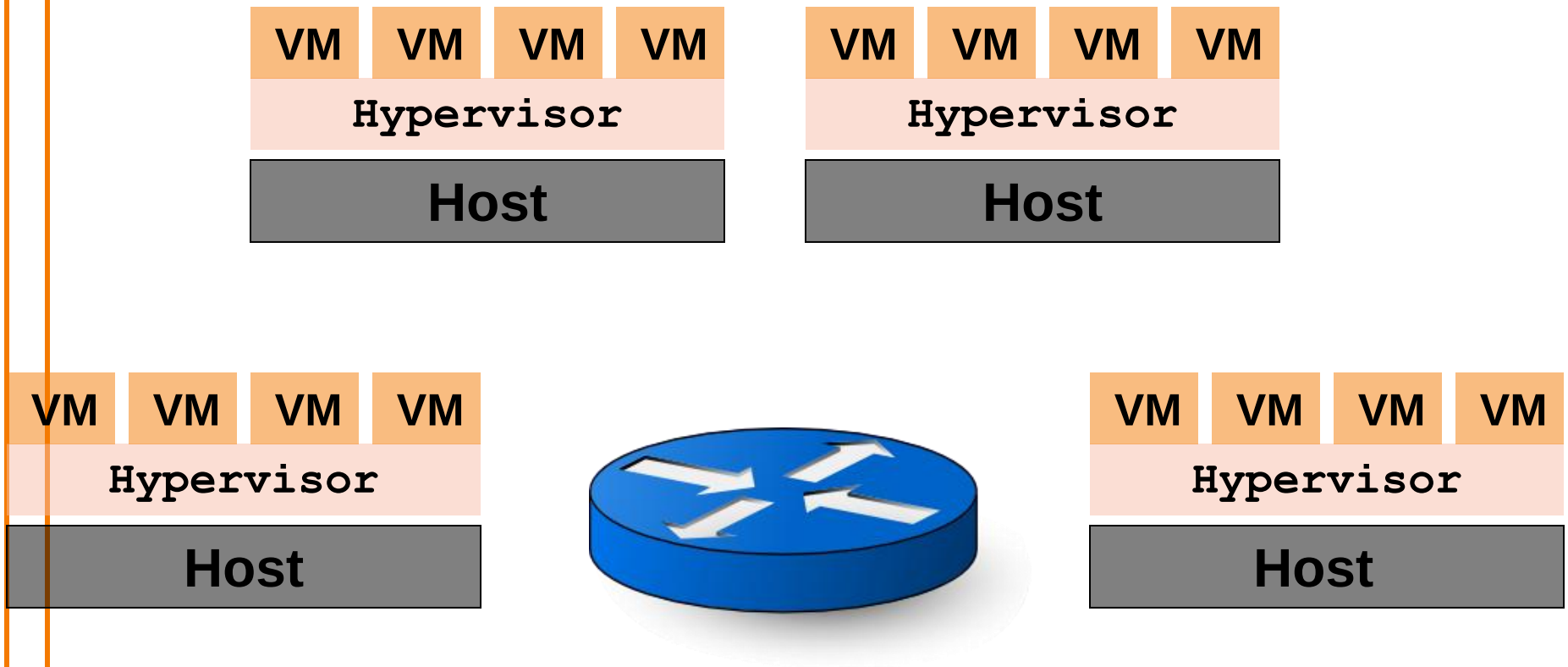


Virtual Switches (Vswitch)

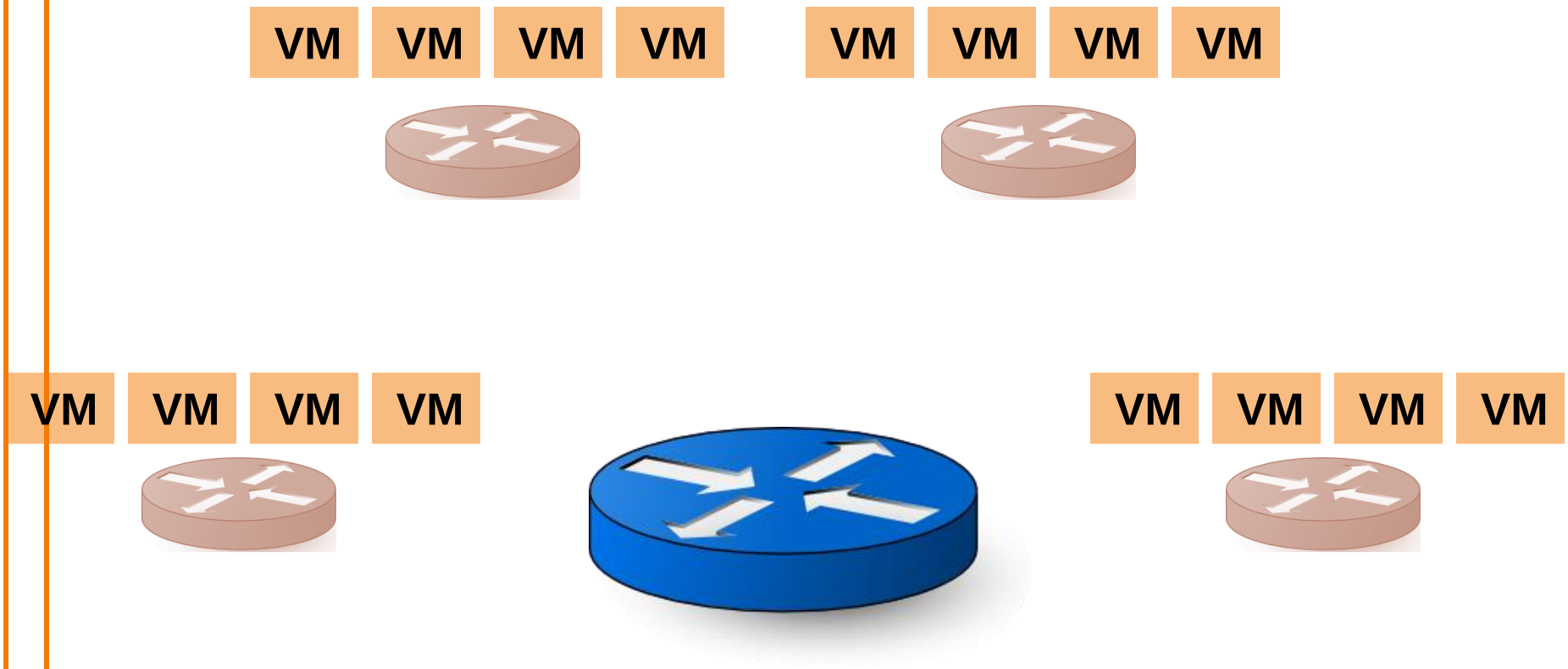


- Vswitch is first-hop switch for all VMs
  - Software switch (speed is not an issue)
  - Should support OpenFlow
  - Can be controlled by Network Operating System

Physical View of Virtualized Network



Logical View of Virtualized Network



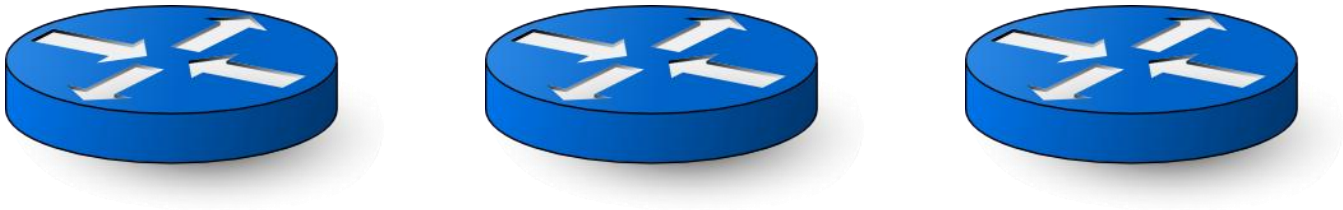
All edge switches are Vswitches

Vswitches are Sufficient

- Vswitches are enough to implement most functions
  - Access control, QoS, mobility, migration, monitoring,…
- Physical network becomes static crossbar
  - Simple to implement and manage
  - Components can be legacy and static

Managing Physical Network

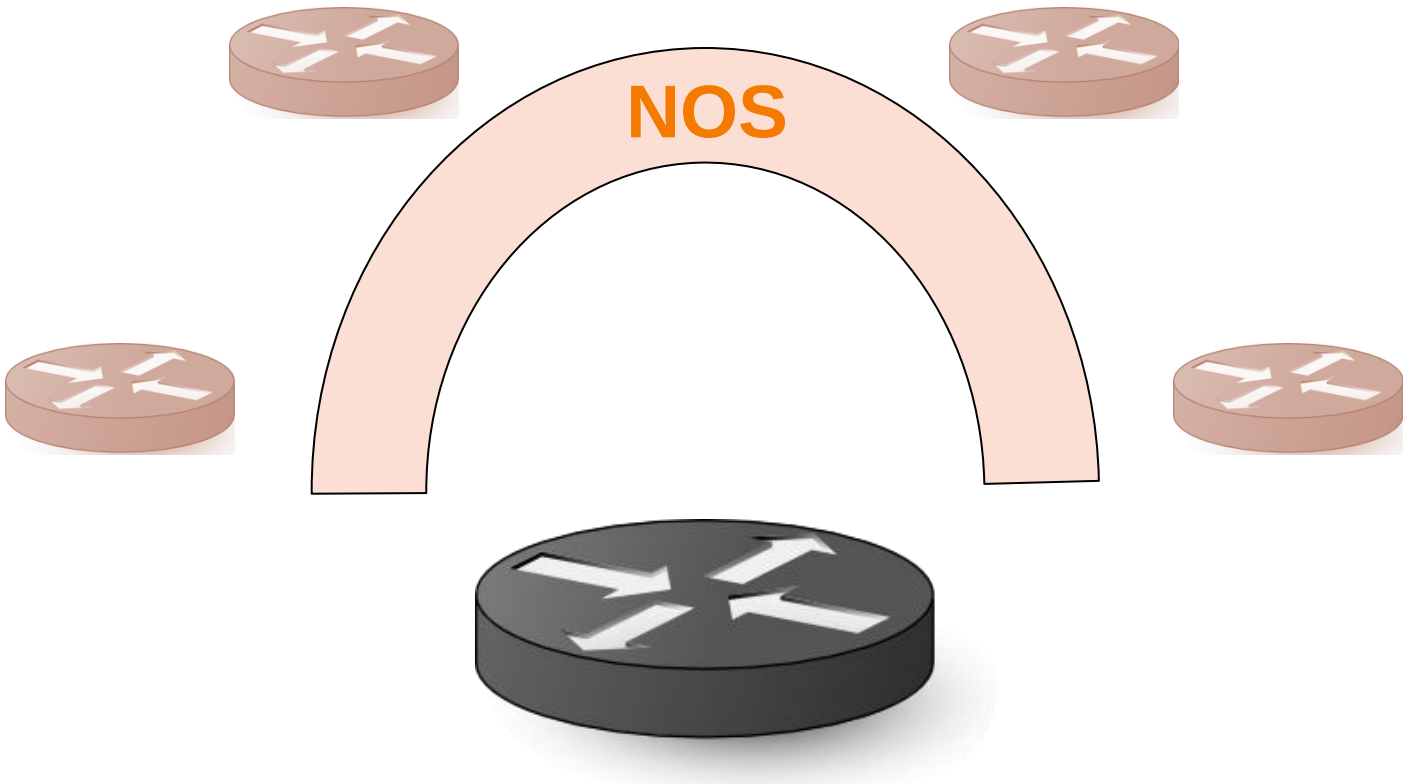
NOS



Physical Switches

Managing Virtualized Network

Only Vswitches need to be controlled by NOS



Physical network is logical crossbar

Vswitches as Insertion Point

- Can insert new functionality into old networks with:
  - Hypervisors with OpenFlow-enabled Vswitch
  - Network Operating System (on servers)
- No change to physical infrastructure
  - Legacy hosts
  - Legacy network components

Evolvability

Implications

- Domains can be build like “fabrics”
- Edge uses software forwarding
  - Controlled by edge controller
- Core uses hardware label-based forwarding
- Only edge understands interdomain delivery
  - Change in “architecture” (e.g., IPv4 to IPv6) only requires change to edge controller software
  - Software-defined Internet Architecture

Software-Defined Internet Architecture

- Major architectural changes via edge software
- Software switches are fast enough to handle load
  - Single core can do ~10Gbps
- No need for architectural support inside domain
  - At least not initially
- No need for a single global internetworking layer
  - A repudiation of the core Internet belief
  - Can have several such layers coexist