Software defined networking

Ankit Singla

ETH Zürich Spring 2017

[Many slides are from Brighten Godfrey, UIUC]

This lecture …

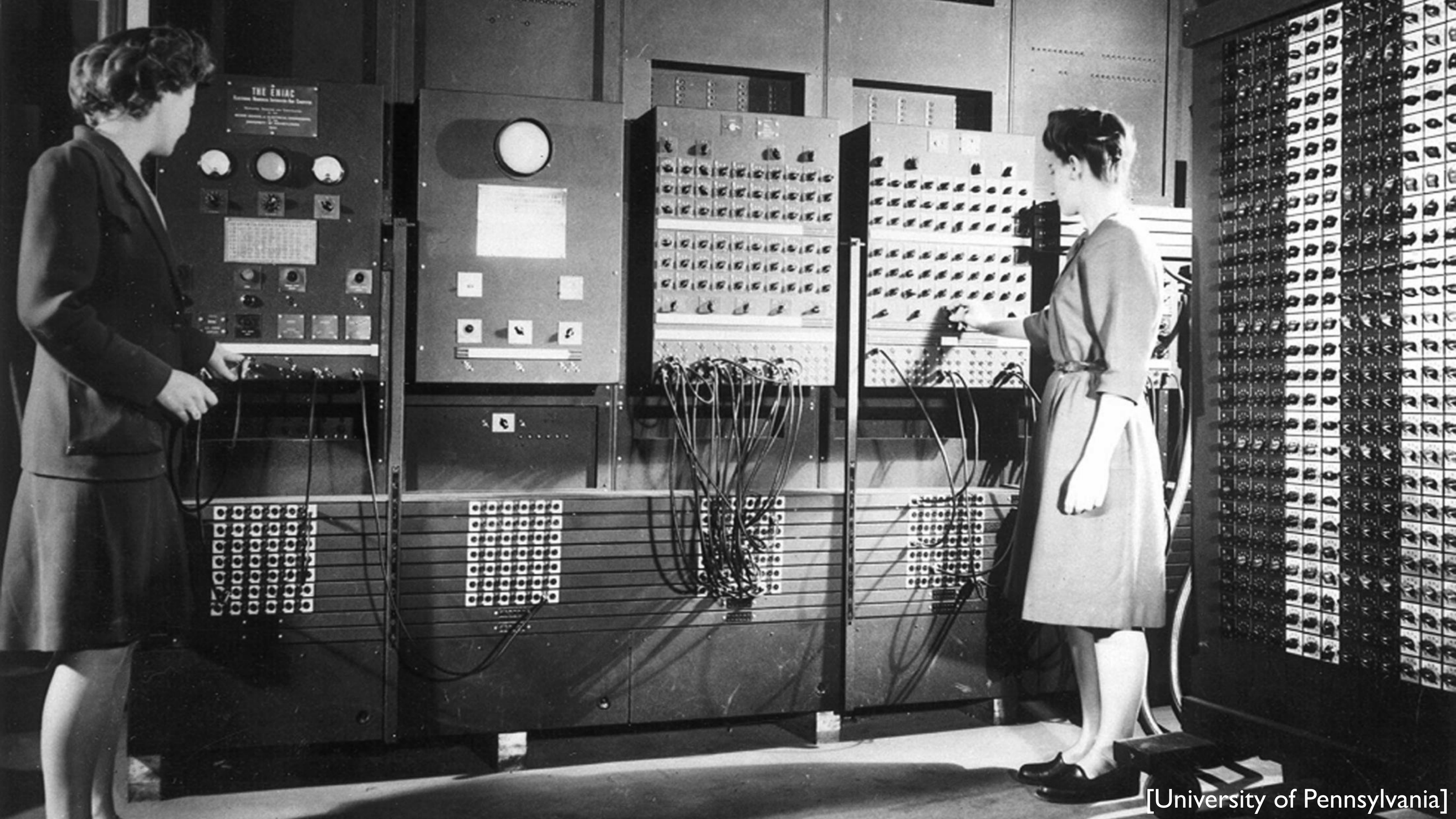
- How do we manage the network?
- How do we make things more programmable?

- Intro: "Host-level networking" — via Patrick
  - *Safe and Effective Fine-grained TCP Retransmissions for Datacenter Communication*

The problem

Networks are complicated

- Just like any computer system
- Worse: distributed
- Even worse: only “knobs and dials”



[University of Pennsylvania]

Complexity causes bugs ...

Operator error the root cause of Microsoft Azure failure

Cliff Saran
Managing Editor

20 Nov 2014 17:10

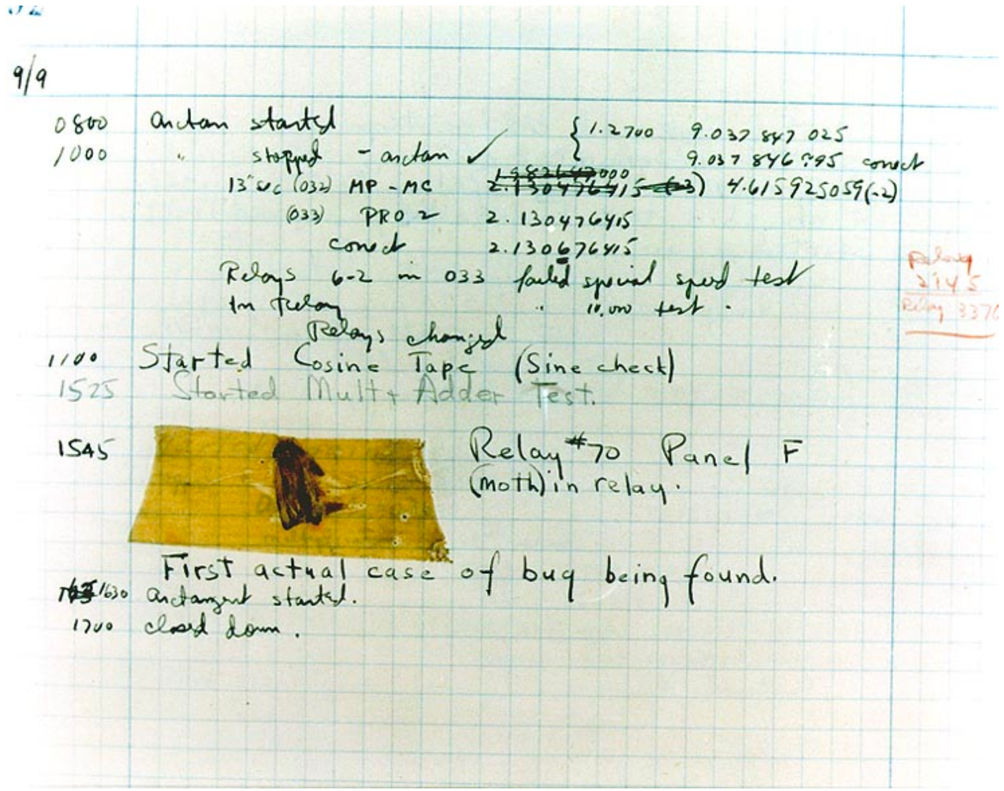
Operator error was the root cause of the catastrophic failure of Microsoft's Azure cloud computing platform on 19 November 2014

Telia engineer error to blame for massive net outage

Lesson 1: Don't confuse Europe with Hong Kong

20 Jun 2016 at 20:27, Kieren McCarthy

Historical side-note



The First "Computer Bug"

“Amazing Grace” Hopper

First compiler!


CRAY
HOPPER


70

The problem

Networks are complicated

- Just like any computer system
- Worse: distributed
- Even worse: only “knobs and dials”

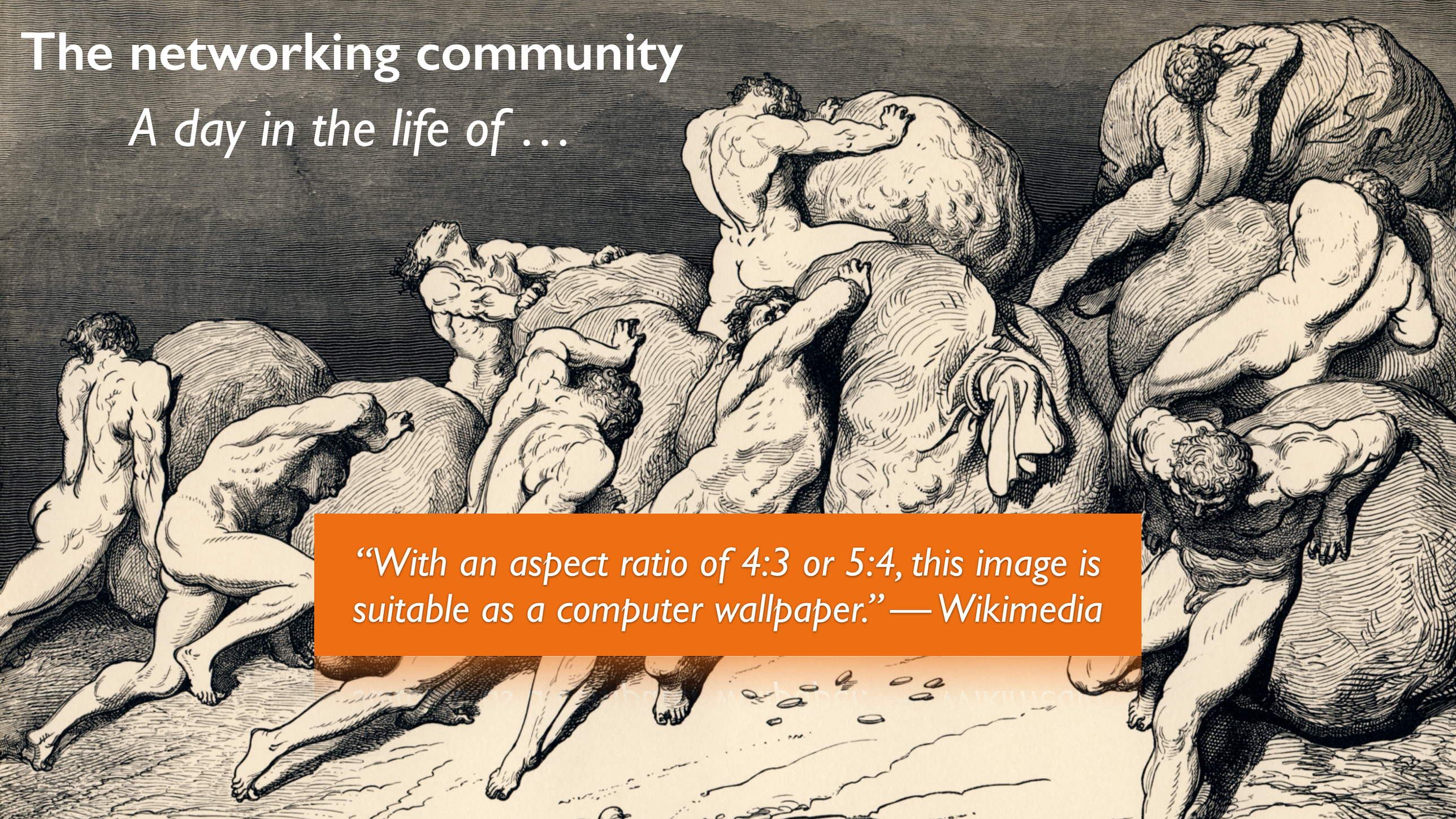
Network equipment is proprietary

- “Integrated solutions” from major vendors


Integrated
Solution

The networking community

A day in the life of …

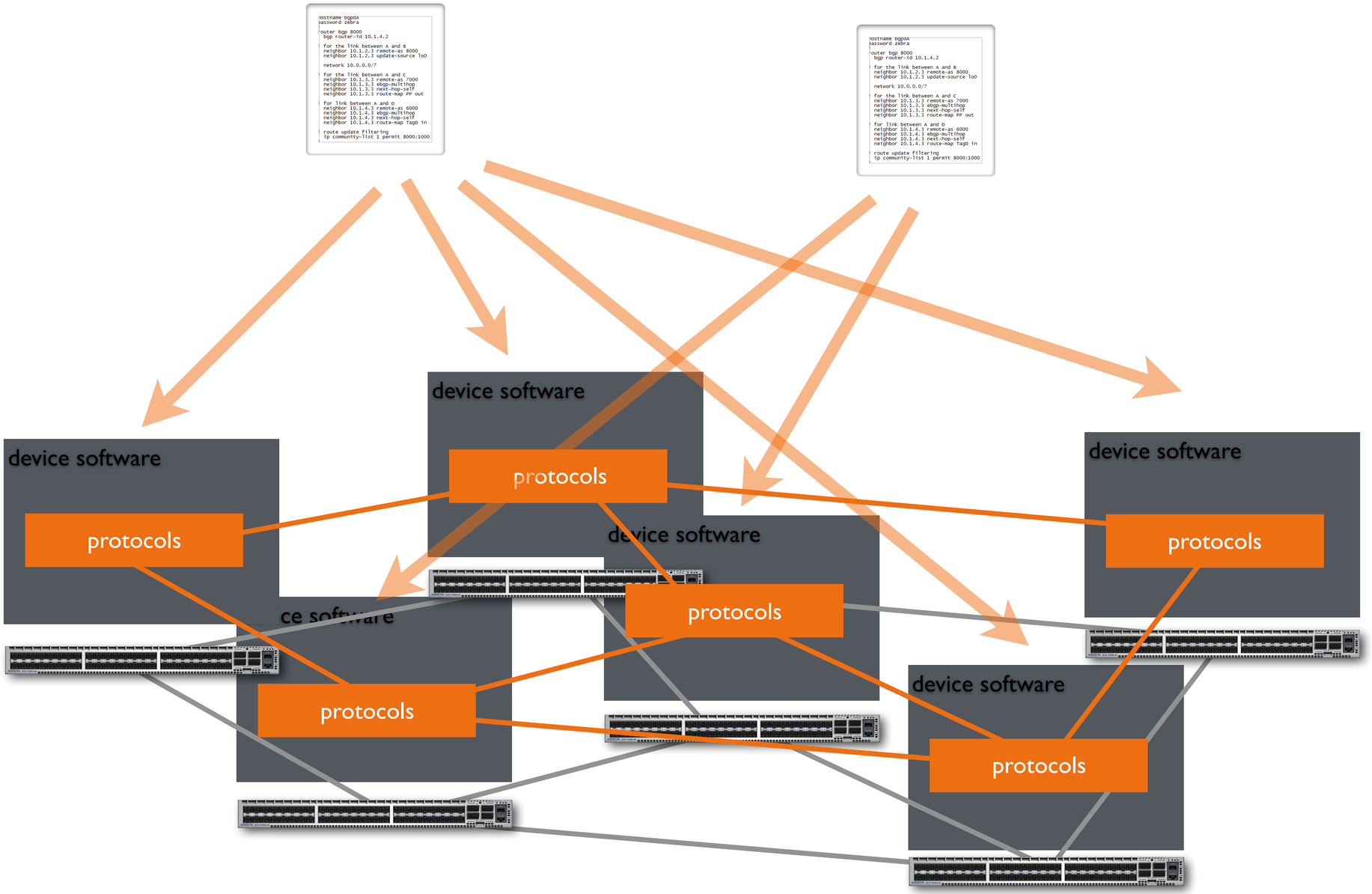


"With an aspect ratio of 4:3 or 5:4, this image is suitable as a computer wallpaper." — Wikimedia

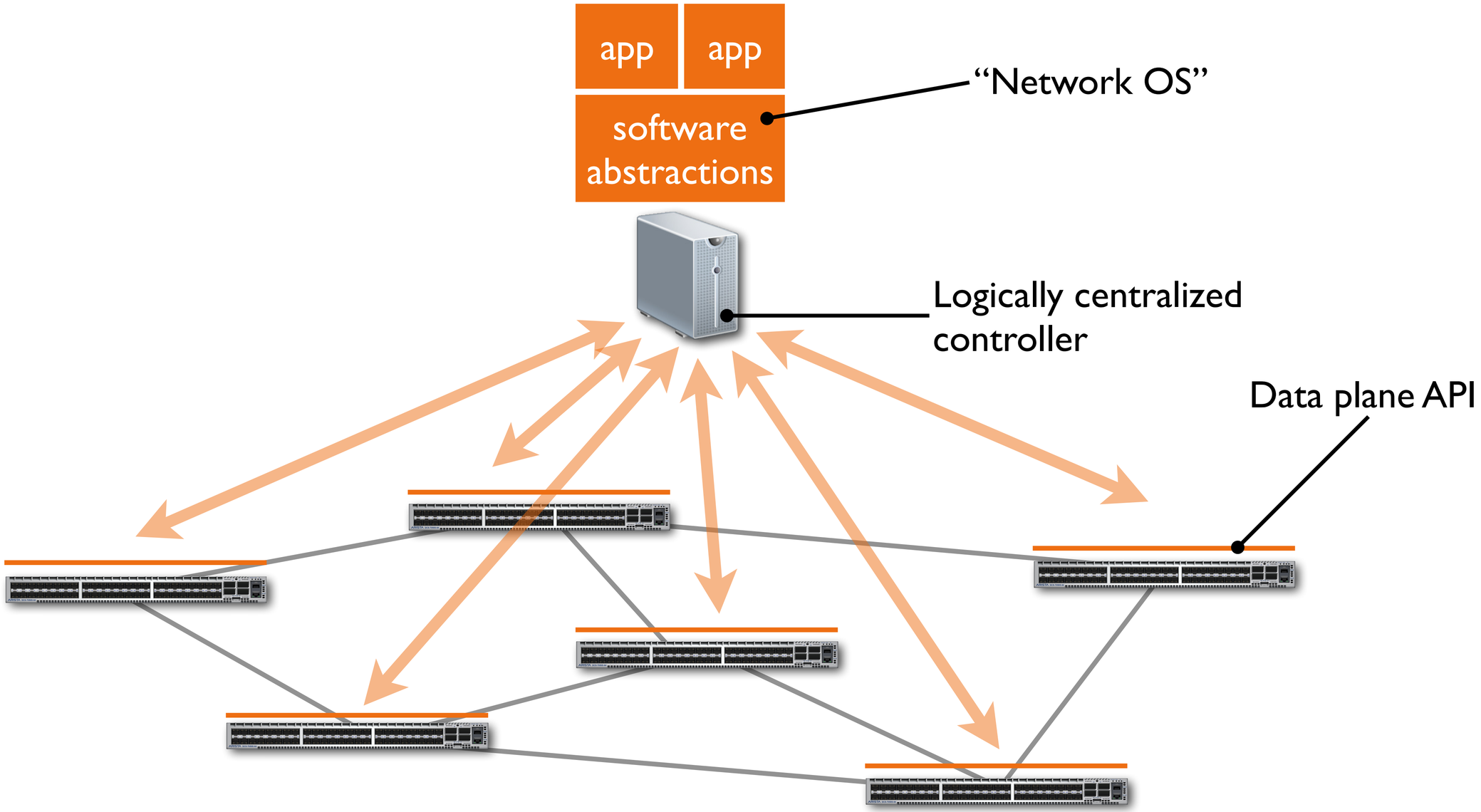
Traditional network

```
hostname bgpdA
password zebra
!
router bgp 8000
  bgp router-id 10.1.4.2

! for the link between A and B
  neighbor 10.1.2.3 remote-as 8000
  neighbor 10.1.2.3 update-source lo0

  network 10.0.0.0/7

! for the link between A and C
  neighbor 10.1.3.3 remote-as 7000
  neighbor 10.1.3.3 ebgp-multihop
  neighbor 10.1.3.3 next-hop-self
  neighbor 10.1.3.3 route-map PP out

! for link between A and D
  neighbor 10.1.4.3 remote-as 6000
  neighbor 10.1.4.3 ebgp-multihop
  neighbor 10.1.4.3 next-hop-self
  neighbor 10.1.4.3 route-map TagD in

! route update filtering
  ip community-list 1 permit 8000:1000
!
```

Traditional network



Software-defined network



Example

From NOX [Gude, Koponen, Pettit, Pfaff, Casado, McKeown, Shenker, CCR 2008]

```
# On user authentication, statically setup VLAN tagging
# rules at the user's first hop switch
def setup_user_vlan(dp, user, port, host):
    vlanid = user_to_vlan_function(user)

    # For packets from the user, add a VLAN tag
    attr_out[IN_PORT] = port
    attr_out[DL_SRC] = nox.reverse_resolve(host).mac
    action_out = [(nox.OUTPUT, (0, nox.FLOOD)),
        (nox.ADD_VLAN, (vlanid))]
    install_datapath_flow(dp, attr_out, action_out)

    # For packets to the user with the VLAN tag, remove it
    attr_in[DL_DST] = nox.reverse_resolve(host).mac
    attr_in[DL_VLAN] = vlanid
    action_in = [(nox.OUTPUT, (0, nox.FLOOD)), (nox.DEL_VLAN)]
    install_datapath_flow(dp, attr_in, action_in)

nox.register_for_user_authentication(setup_user_vlan)
```

Example

From NOX [Gude, Koponen, Pettit, Pfaff, Casado, McKeown, Shenker, CCR 2008]

```
# On user authentication, statically setup VLAN tagging
# rules at the user's first hop switch
def setup_user_vlan(dp, user, port, host):
    vlanid = user_to_vlan_function(user)

    # For packets from the user, add a VLAN tag
    attr_out[IN_PORT] = port
    attr_out[DL_SRC] = nox.reverse_resolve(host).mac
    action_out = [(nox.OUTPUT, (0, nox.FLOOD)),
        (nox.ADD_VLAN, (vlanid))]
    install_datapath_flow(dp, attr_out, action_out)

    # For packets to the user with the VLAN tag, remove it
    attr_in[DL_DST] = nox.reverse_resolve(host).mac
    attr_in[DL_VLAN] = vlanid
    action_in = [(nox.OUTPUT, (0, nox.FLOOD)), (nox.DEL_VLAN)]
    install_datapath_flow(dp, attr_in, action_in)

nox.register_for_user_authentication(setup_user_vlan)
```

Match specific set of packets

Example

From NOX [Gude, Koponen, Pettit, Pfaff, Casado, McKeown, Shenker, CCR 2008]

```
# On user authentication, statically setup VLAN tagging
# rules at the user's first hop switch
def setup_user_vlan(dp, user, port, host):
    vlanid = user_to_vlan_function(user)

    # For packets from the user, add a VLAN tag
    attr_out[IN_PORT] = port
    attr_out[DL_SRC] = nox.reverse_resolve(host).mac
    action_out = [(nox.OUTPUT, (0, nox.FLOOD)),
        (nox.ADD_VLAN, (vlanid))]
    install_datapath_flow(dp, attr_out, action_out)

    # For packets to the user with the VLAN tag, remove it
    attr_in[DL_DST] = nox.reverse_resolve(host).mac
    attr_in[DL_VLAN] = vlanid
    action_in = [(nox.OUTPUT, (0, nox.FLOOD)), (nox.DEL_VLAN)]
    install_datapath_flow(dp, attr_in, action_in)

nox.register_for_user_authentication(setup_user_vlan)
```

Match specific set of packets

Construct action

Example

From NOX [Gude, Koponen, Pettit, Pfaff, Casado, McKeown, Shenker, CCR 2008]

```
# On user authentication, statically setup VLAN tagging
# rules at the user's first hop switch
def setup_user_vlan(dp, user, port, host):
    vlanid = user_to_vlan_function(user)

    # For packets from the user, add a VLAN tag
    attr_out[IN_PORT] = port
    attr_out[DL_SRC] = nox.reverse_resolve(host).mac
    action_out = [(nox.OUTPUT, (0, nox.FLOOD)),
        (nox.ADD_VLAN, (vlanid))]
    install_datapath_flow(dp, attr_out, action_out)

    # For packets to the user with the VLAN tag, remove it
    attr_in[DL_DST] = nox.reverse_resolve(host).mac
    attr_in[DL_VLAN] = vlanid
    action_in = [(nox.OUTPUT, (0, nox.FLOOD)), (nox.DEL_VLAN)]
    install_datapath_flow(dp, attr_in, action_in)

nox.register_for_user_authentication(setup_user_vlan)
```

Match specific set of packets

Construct action

Install (match, action) in a specific switch

Example

From NOX [Gude, Koponen, Pettit, Pfaff, Casado, McKeown, Shenker, CCR 2008]

```
# On user authentication, statically setup VLAN tagging
# rules at the user's first hop switch
def setup_user_vlan(dp, user, port, host):
    vlanid = user_to_vlan_function(user)

    # For packets from the user, add a VLAN tag
    attr_out[IN_PORT] = port
    attr_out[DL_SRC] = nox.reverse_resolve(host).mac
    action_out = [(nox.OUTPUT, (0, nox.FLOOD)),
        (nox.ADD_VLAN, (vlanid))]
    install_datapath_flow(dp, attr_out, action_out)

    # For packets to the user with the VLAN tag, remove it
    attr_in[DL_DST] = nox.reverse_resolve(host).mac
    attr_in[DL_VLAN] = vlanid
    action_in = [(nox.OUTPUT, (0, nox.FLOOD)), (nox.DEL_
    install_datapath_flow(dp, attr_in, action_in)

nox.register_for_user_authentication(setup_user_vlan)
```

Match specific set of packets

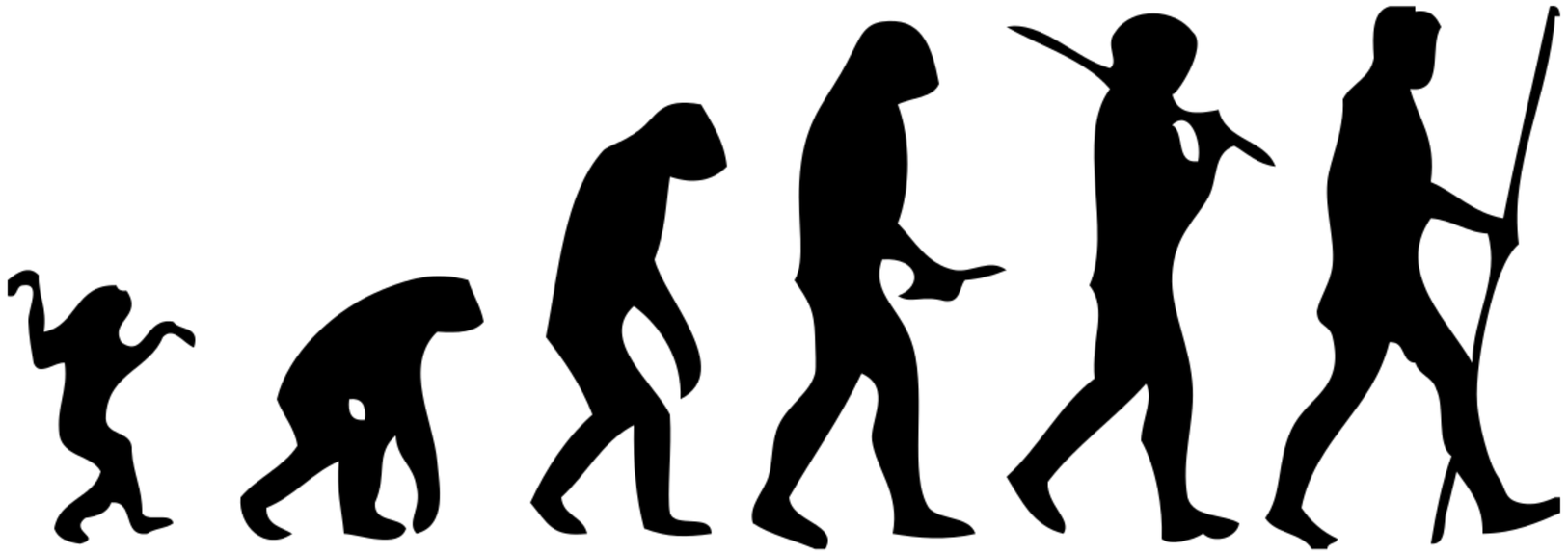
Construct action

Install (match, action) in a specific switch

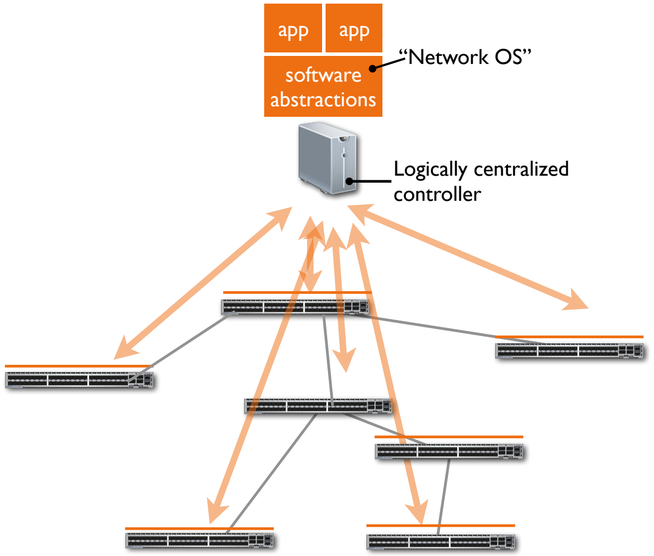
Common primitives:

- Match packets, execute actions
- Topology discovery
- Monitoring

Evolution of SDN



[José-Manuel Benitos]



Evolution of SDN

ACM CCR, 2014

The Road to SDN: An Intellectual History of Programmable Networks

Nick Feamster
Georgia Tech
feamster@cc.gatech.edu

Jennifer Rexford
Princeton University
jrex@cs.princeton.edu

Ellen Zegura
Georgia Tech
ewz@cc.gatech.edu

Evolution of SDN: Flexible Data Planes

Label switching / MPLS (1997)

- "Tag Switching Architecture Overview", [Rekhter, Davie, Rose, Swallow, Farinacci, Katz, Proc. IEEE, 1997]
- Set up explicit paths for classes of traffic

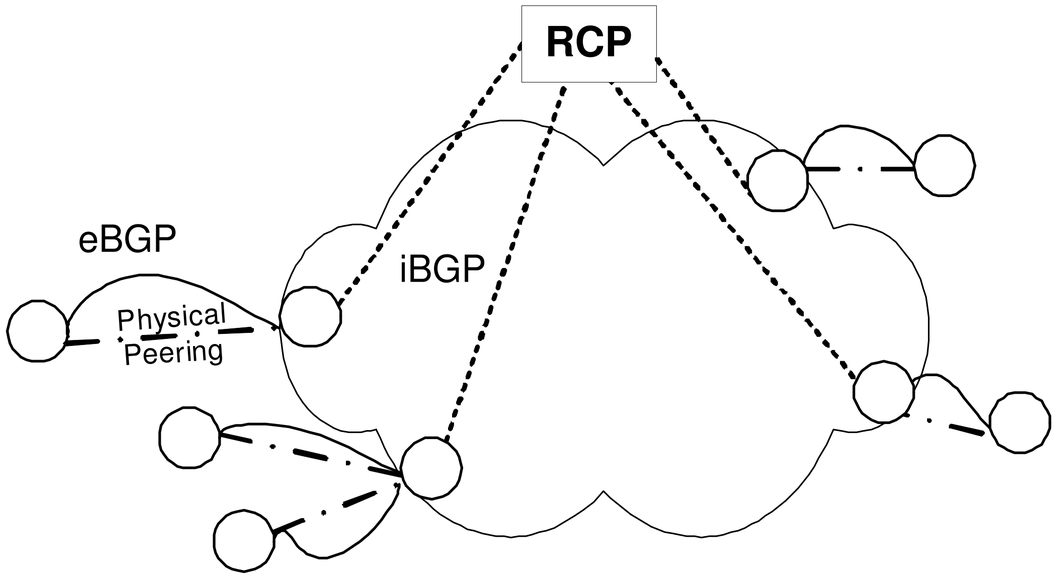
Active Networks (1999)

- Packet header carries (pointer to) program code

Evolution of SDN: Logically Centralized Control

Routing Control Platform (2005)

- [Caesar, Caldwell, Feamster, Rexford, Shaikh, van der Merwe, NSDI 2005]
- Centralized computation of BGP routes, pushed to border routers via iBGP

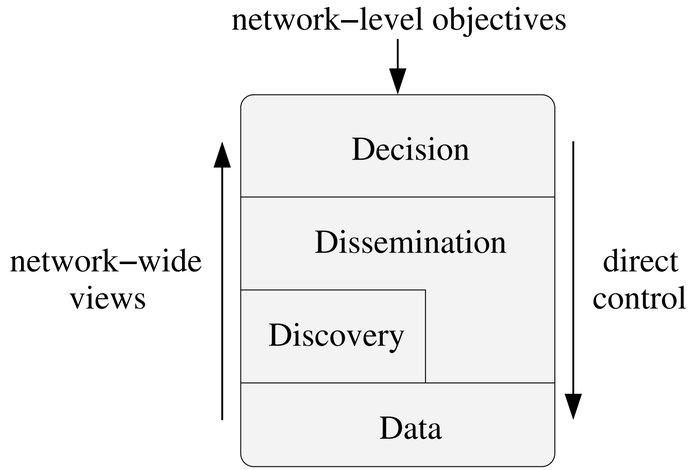


Evolution of SDN: Logically Centralized Control

Routing Control Platform (2005)

4D architecture (2005)

- A Clean Slate 4D Approach to Network Control and Management [Greenberg, Hjalmtysson, Maltz, Myers, Rexford, Xie, Yan, Zhan, Zhang, CCR Oct 2005]
- Logically centralized "decision plane" separated from data plane



Evolution of SDN: Logically Centralized Control

Routing Control Platform (2005)

4D architecture (2005)

Ethane (2007)

- [Casado, Freedman, Pettit, Luo, McKeown, Shenker, SIGCOMM 2007]
- Centralized controller enforces enterprise network Ethernet forwarding policy using existing hardware

Evolution of SDN: Logically Centralized Control

Routing Control Platform (2005)

4D architecture (2005)

Ethane (2007)

- [Casado, Freedman, Pettit, Luo, McKe
  SIGCOMM 2007]
- Centralized controller enforces enter
  Ethernet forwarding policy using exist

```
# Groups —
desktops = ["griffin","roo"];
laptops = ["glaptop","rlaptop"];
phones = ["gphone","rphone"];
server = ["http_server","nfs_server"];
private = ["desktops","laptops"];
computers = ["private","server"];
students = ["bob","bill","pete"];
profs = ["plum"];
group = ["students","profs"];
waps = ["wap1","wap2"];
%%
# Rules —
[(hsrc=in("server")∧(hdst=in("private"))] : deny;
# Do not allow phones and private computers to communicate
[(hsrc=in("phones")∧(hdst=in("computers"))] : deny;
[(hsrc=in("computers")∧(hdst=in("phones"))] : deny;
# NAT-like protection for laptops
[(hsrc=in("laptops")] : outbound-only;
# No restrictions on desktops communicating with each other
[(hsrc=in("desktops")∧(hdst=in("desktops"))] : allow;
# For wireless, non-group members can use http through
# a proxy. Group members have unrestricted access.
[(apsrc=in("waps"))∧(user=in("group"))] :allow;
[(apsrc=in("waps"))∧(protocol="http)] : waypoints("http-proxy");
[(apsrc=in("waps"))] : deny;
[]: allow; # Default-on: by default allow flows
```

Figure 4: A sample policy file using *Pol-Eth*

Evolution of SDN: Logically Centralized Control

Routing Control Platform (2005)

4D architecture (2005)

Ethane (2007)

OpenFlow (2008)

- [McKeown, Anderson, Balakrishnan, Parulkar, Peterson, Rexford, Shenker, Turner, CCR 2008]
- Thin, standardized interface to data plane
- General-purpose programmability at controller

Evolution of SDN: Logically Centralized Control

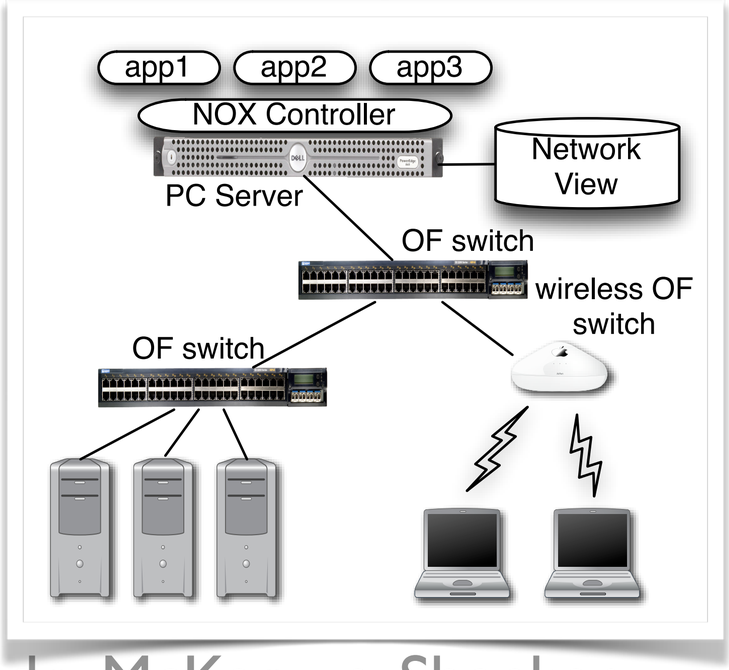
Routing Control Platform (2005)

4D architecture (2005)

Ethane (2007)

OpenFlow (2008)

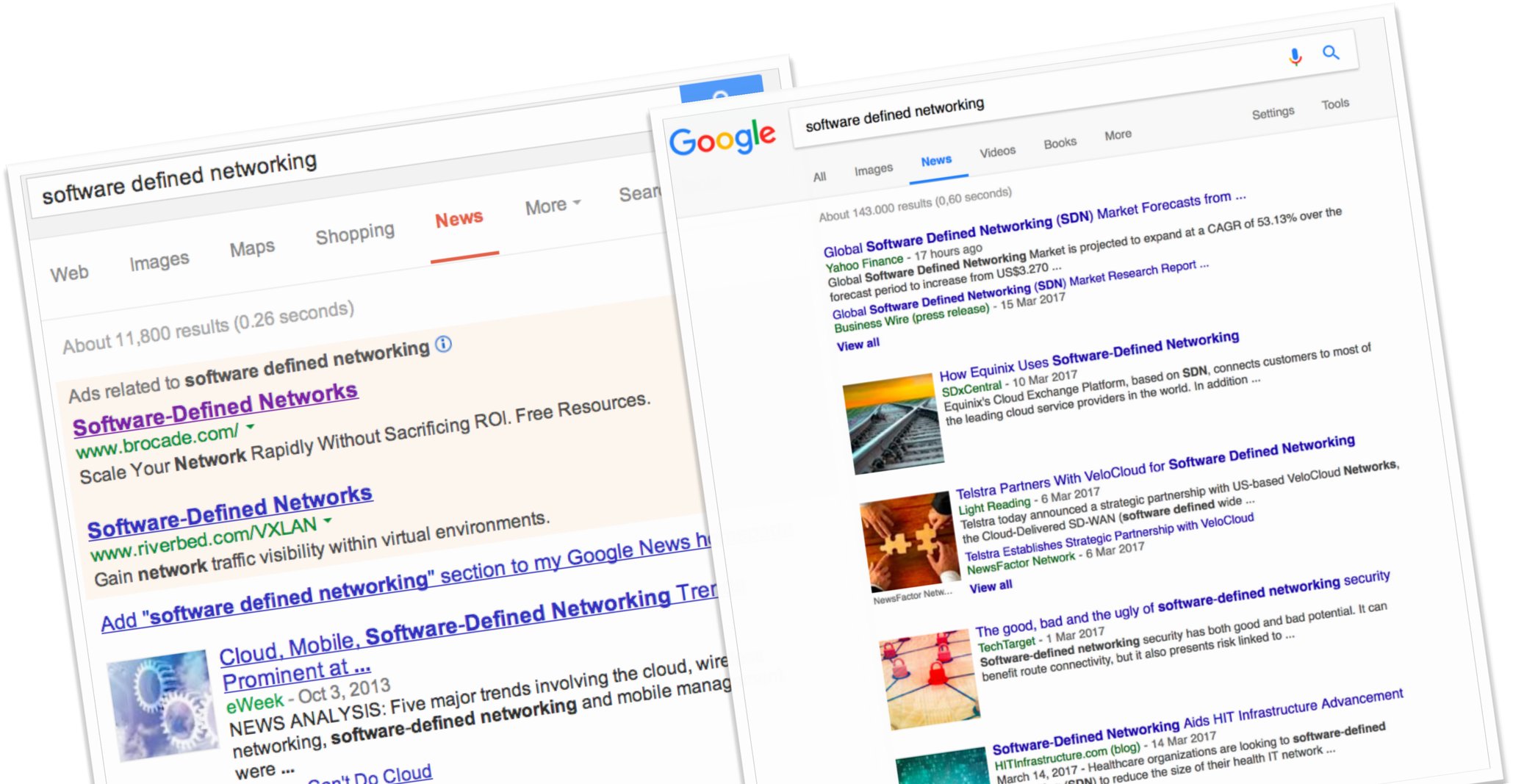
NOX (2008)



- [Gude, Koponen, Pettit, Pfaff, Casado, McKeown, Shenker, CCR 2008]
- First OF controller: centralized network view provided to multiple control apps as a database
- Behind the scenes, handles state collection & distribution

Evolution of SDN

Industry explosion (~2010+)



Opportunities

Open data plane interface

- Hardware: with standardized API, easier for operators to change hardware, and for vendors to enter market
- Software: can more directly access device behavior

Centralized controller

- Direct programmatic control of network

Software abstractions on the controller

- Solve distributed problems *once*, then just write algorithms
- Libraries / languages to help programmers write net apps
- Systems to write high level policy instead of programming

Challenges for SDN

Performance and scalability

Distributed system challenges still present

- Resilience of "logically centralized" controller
- Imperfect knowledge of network state
- Consistency issues between controllers

Challenges for SDN

Reaching agreement on data plane protocol

- OpenFlow? NFV functions? Programmable data planes?

Devising the right control abstractions

- Programming OpenFlow: far too low level
- But what are the right high-level abstractions?

The first "Killer Apps" for SDN

Cloud virtualization

- Create separate virtual networks for tenants
- Allow flexible placement and movement of VMs

Inter-datacenter traffic engineering

- Drive utilization to near 100% when possible
- Protect critical traffic from congestion

Key characteristics of the above use cases

- Special-purpose deployments with less diverse hardware
- Existing solutions aren't just inconvenient, *they don't work!*

Multi-Tenant Data Centers: The Challenges

Key Needs

Agility

Strength

Constitution

Dexterity

Charisma

Key Needs

Agility

Location independent addressing

Performance uniformity

Security

Network semantics

Agility

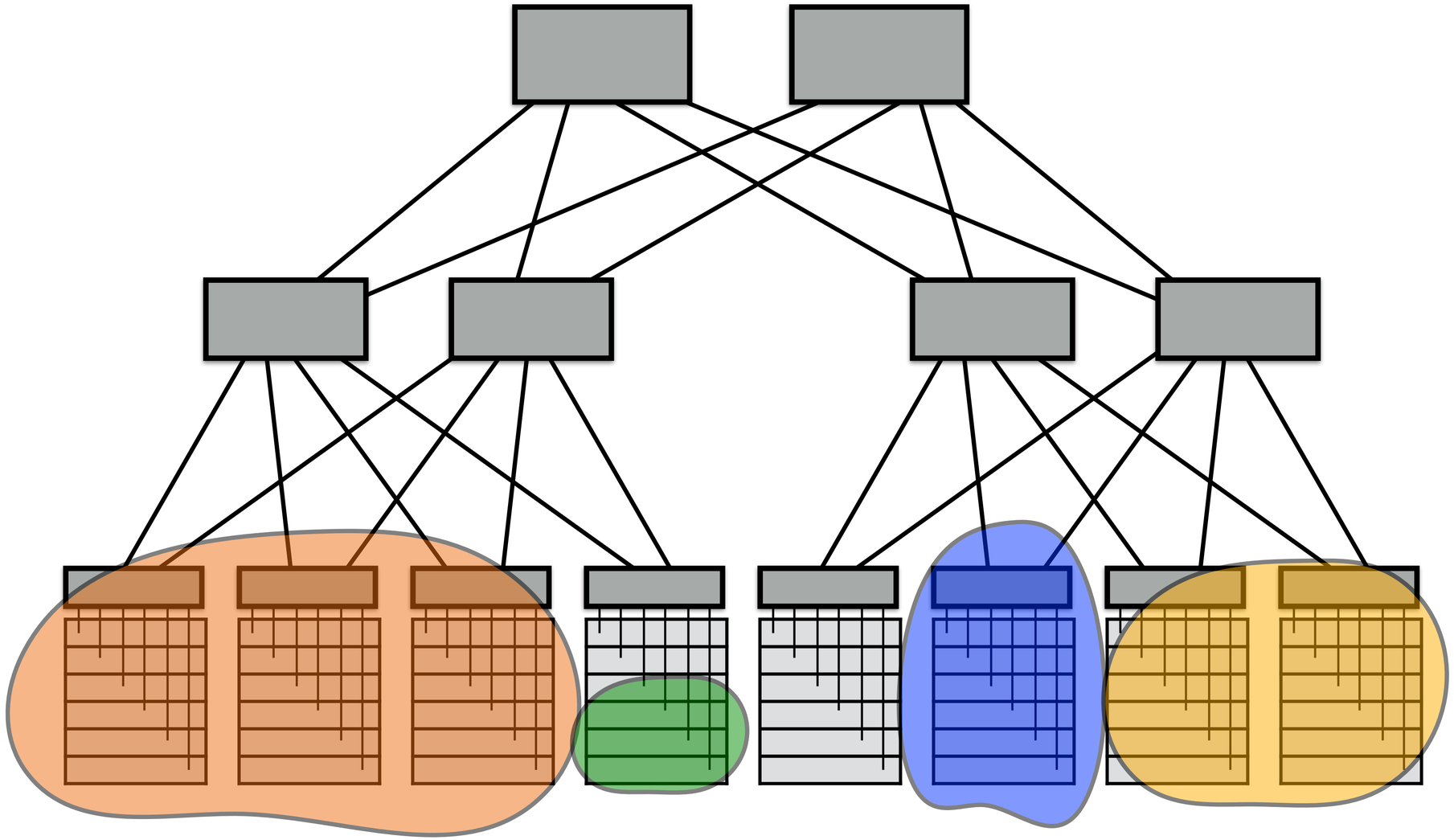
Agility: Use any server for any service at any time

- Better economy of scale through increased utilization
- Improved reliability

Service / tenant

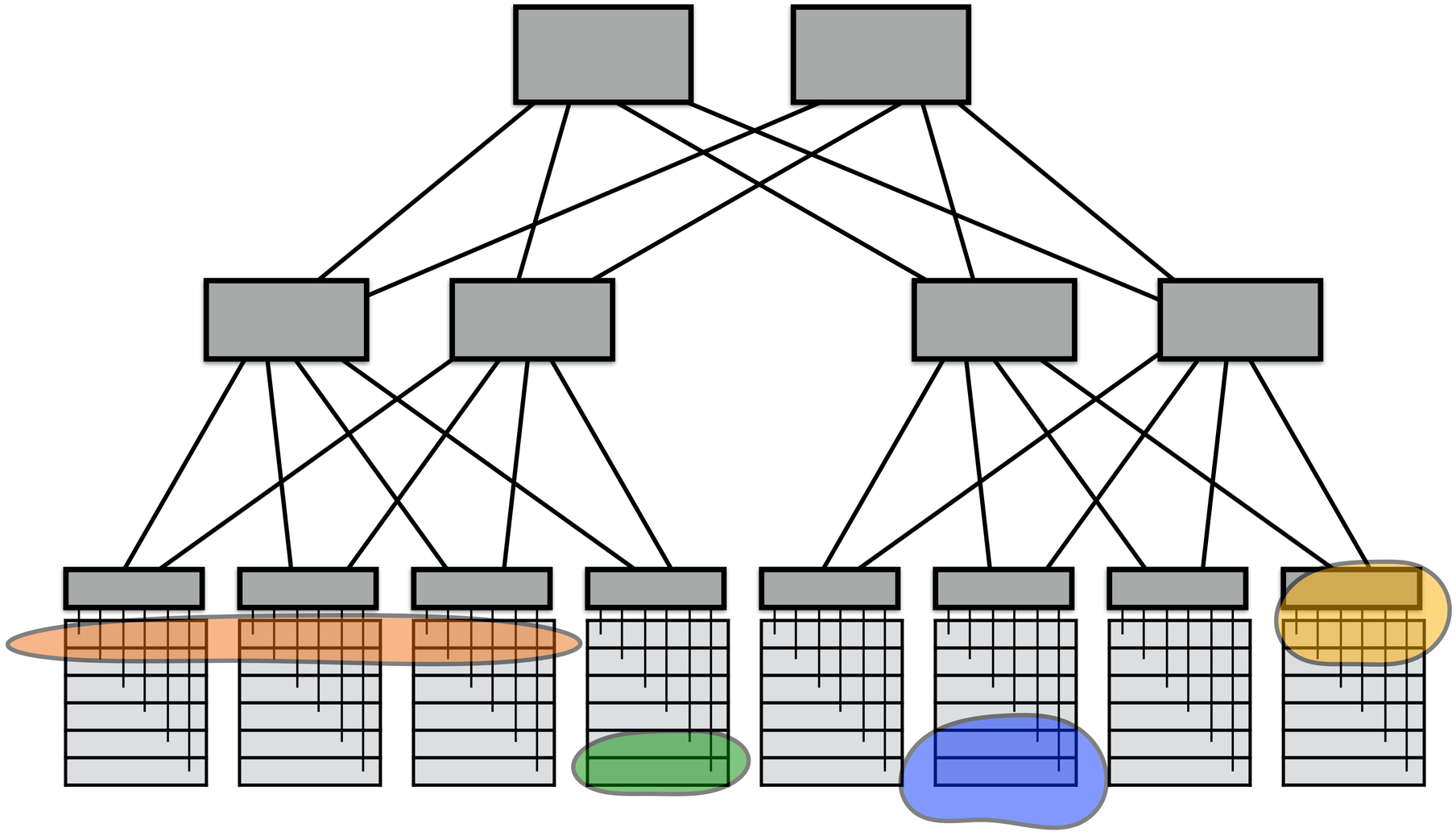
- Customer renting space in a public cloud
- Application or service in a private cloud (internal customer)

Lack of Agility in Traditional DCs



Tenants in “silos”

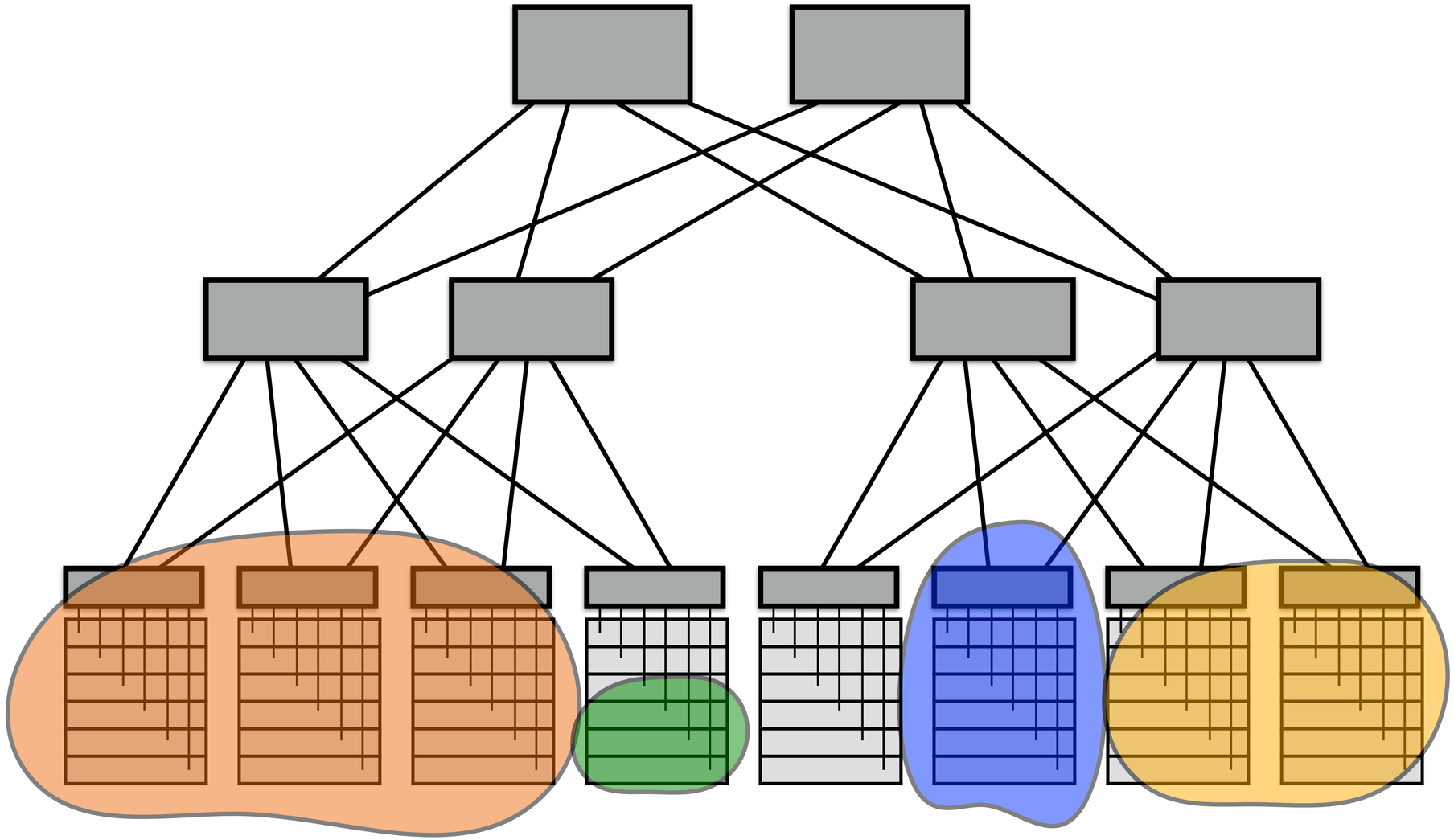
Lack of Agility in Traditional DCs



Tenants in “silos”

Poor utilization

Lack of Agility in Traditional DCs

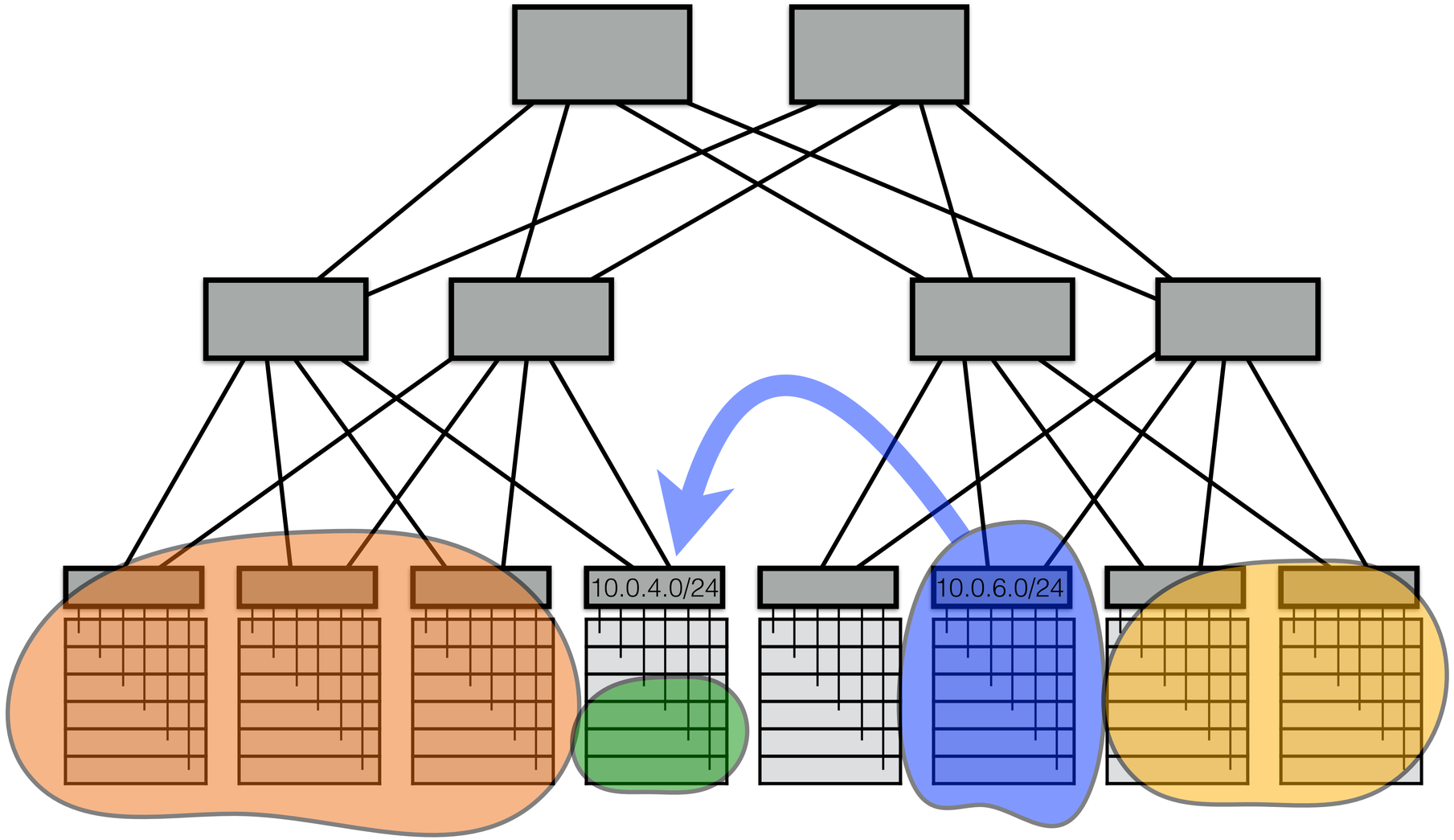


Tenants in “silos”

Poor utilization

Inability to expand

Lack of Agility in Traditional DCs



IP addresses locked to topological location!

Key Needs

Agility

Location independent addressing

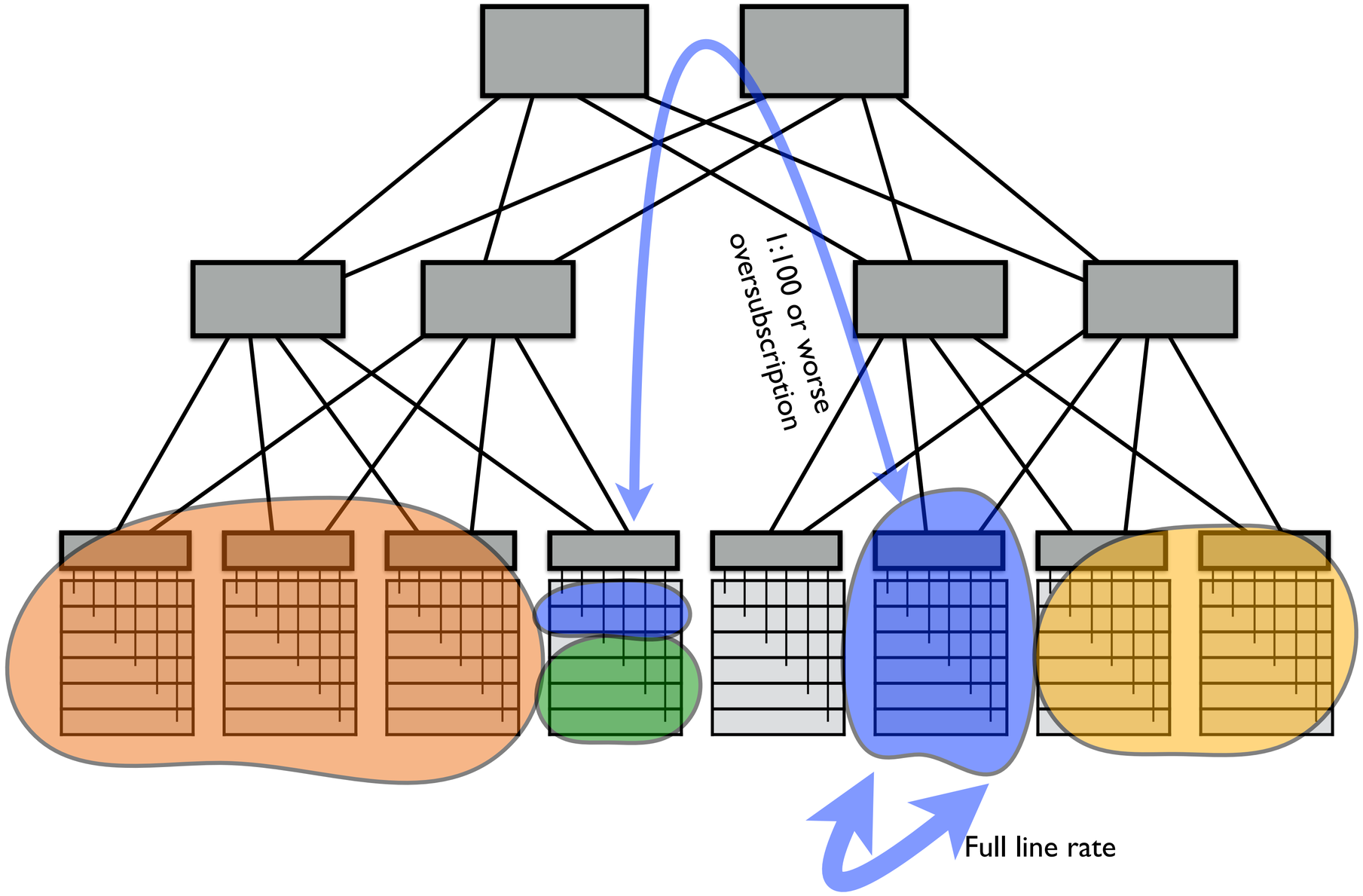
- Tenant's IP addresses can be taken anywhere

Performance uniformity

Security

Network semantics

Lack of Agility in Traditional DCs



Nonuniform
performance

Key Needs

Agility

Location independent addressing

- Tenant's IP addresses can be taken anywhere

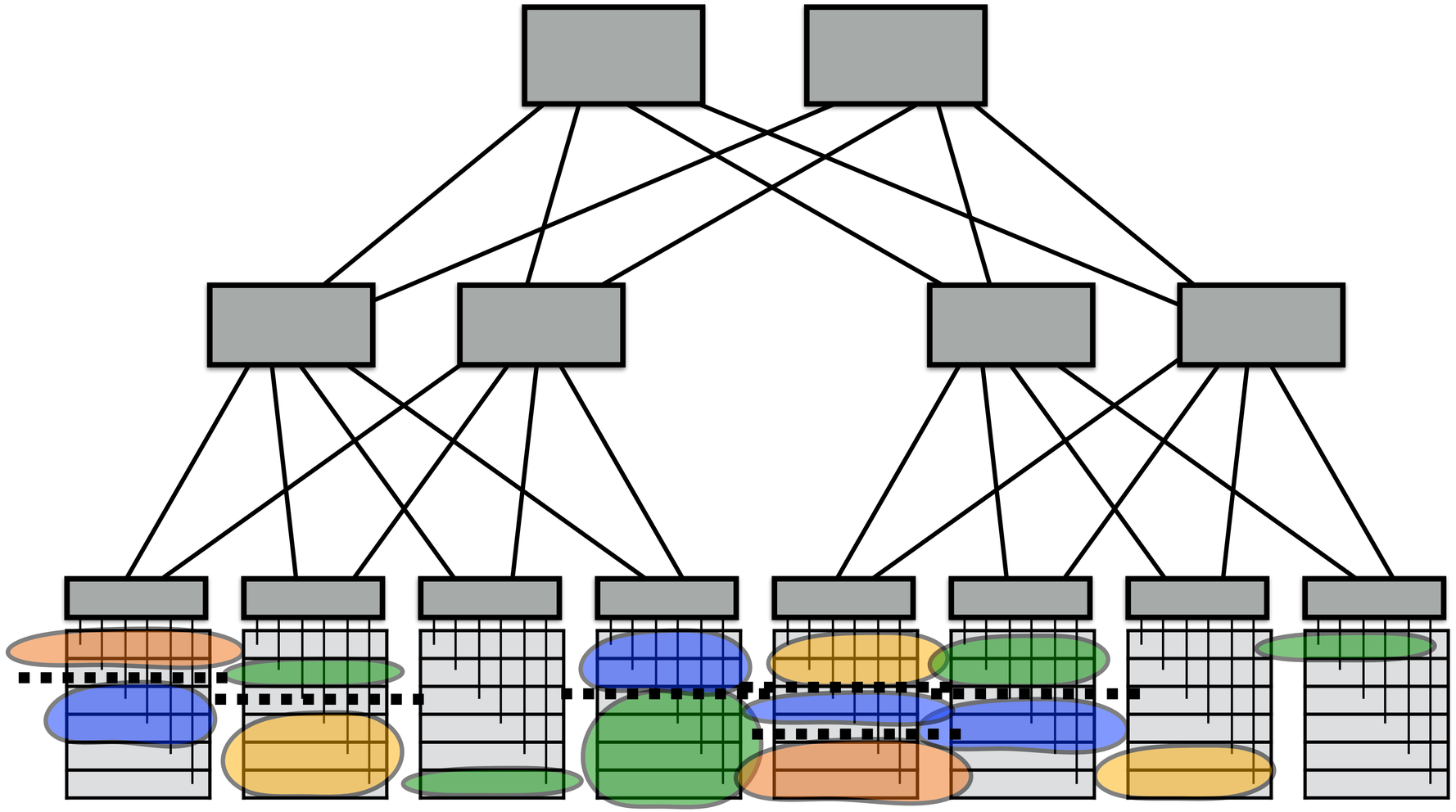
Performance uniformity

- VMs receive same throughput regardless of placement

Security

Network semantics

Lack of Agility in Traditional DCs



Untrusted environment

Key Needs

Agility

Location independent addressing

- Tenant's IP addresses can be taken anywhere

Performance uniformity

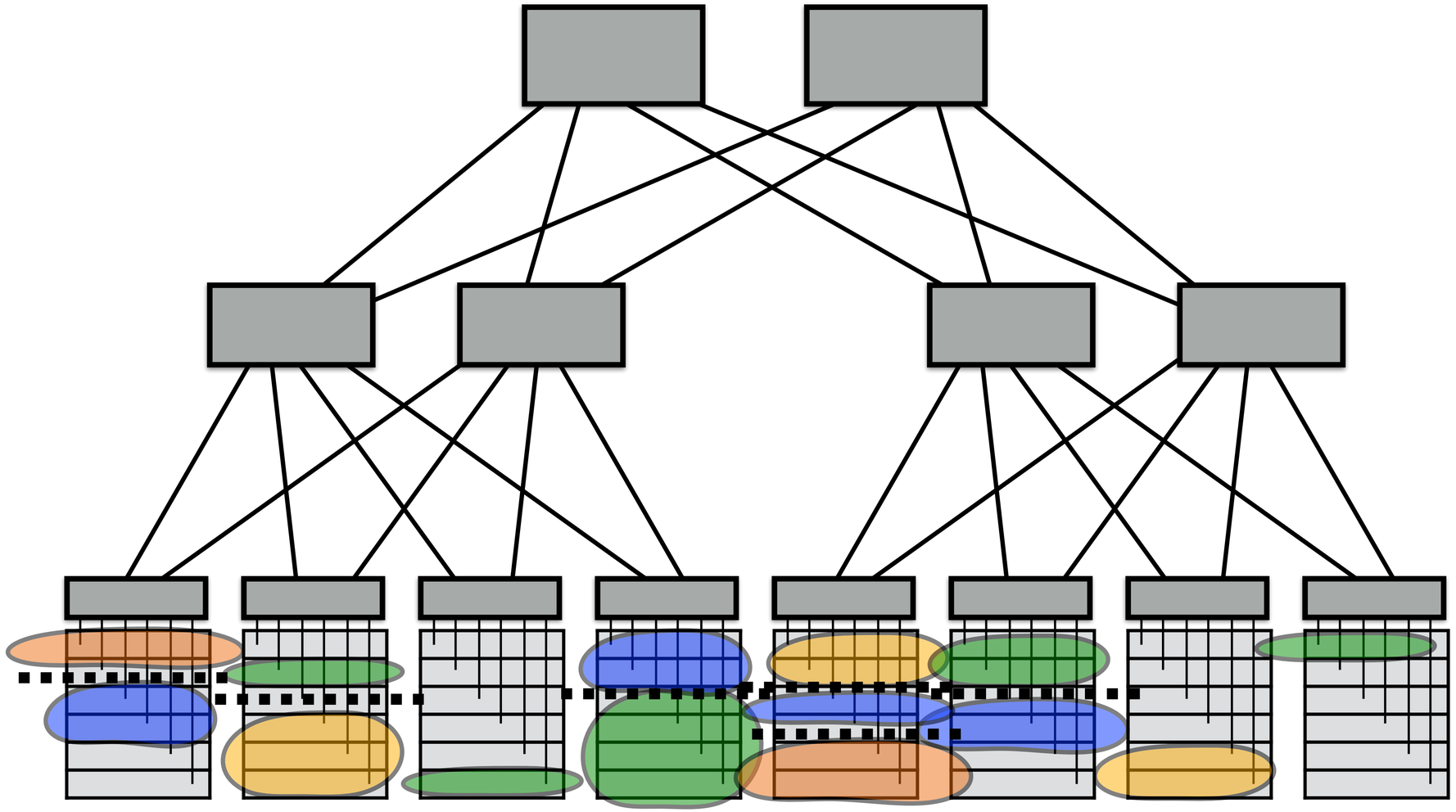
- VMs receive same throughput regardless of placement

Security

- Micro-segmentation: isolation at tenant granularity

Network semantics

Lack of Agility in Traditional DCs



x 1000s of legacy apps in a large enterprise

Key Needs

Agility

Location independent addressing

- Tenant's IP addresses can be taken anywhere

Performance uniformity

- VMs receive same throughput regardless of placement

Security

- Micro-segmentation: isolation at tenant granularity

Network semantics

- Layer 2 service discovery, multicast, broadcast, …

Network Virtualization
Case Study: VL2

Case Study

[ACM SIGCOMM 2009]

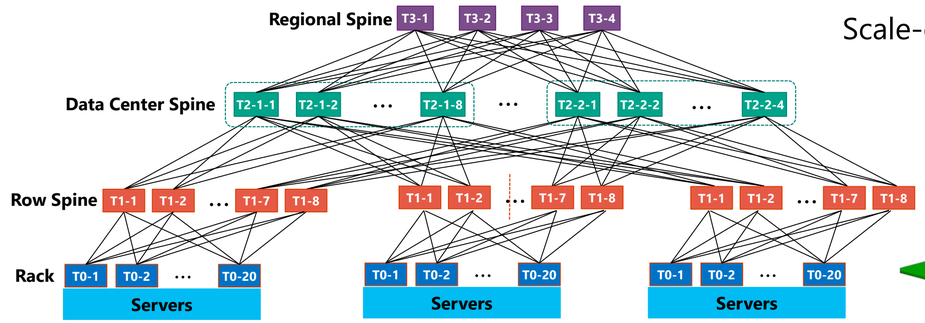
VL2: A Scalable and Flexible Data Center Network

Albert Greenberg, James R. Hamilton, Navendu Jain
Srikanth Kandula, Changhoon Kim, Parantap Lahiri
David A. Maltz, Parveen Patel, Sudipta Sengupta

Microsoft Research

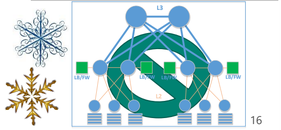
Influenced architecture of Microsoft Azure

VL2 → Azure Clos Fabrics with 40G NICs



Scale-out, active-active

Scale-up, active-passive



Outcome of >10 years of history, with major revisions every six months

Microsoft

16

[Albert Greenberg's keynote, SIGCOMM 2015]

Virtualization

"All problems in computer science can be solved by another level of indirection."

– *David Wheeler*

App / Tenant layer	• Application Addresses (AAs): Location independent • Illusion of a single big Layer 2 switch connecting the app
Virtualization layer	• Directory server: Maintain AA to LA mapping • Server agent: Query server, wrap AAs in outer LA header
Physical network layer	• Locator Addresses (LAs): Tied to topology, used to route • Layer 3 routing via OSPF / BGP

Did we achieve agility?

Location independent addressing

- AAs are location independent

L2 network semantics

- Agent intercepts and handles L2 broadcast, multicast

- Require "layer 2.5" shim agent running on host; but, concept transfers to hypervisor-based virtual switch

Did we achieve agility?

Performance uniformity

- Clos network is built at low over-subscription
- Uniform capacity everywhere
- ECMP provides good (though not perfect) load balancing
- But, performance isolation among tenants depends on TCP

Security

- Directory system can allow/deny connections by choosing whether to resolve an AA to a LA
- But, segmentation not explicitly enforced at hosts

Where's the SDN?

Directory servers: Logically centralized control

- Orchestrate application locations
- Control communication policy

Host agents: dynamic "programming" of data path

VL2 enduring take-aways

Scale-out Clos network

ECMP for traffic-oblivious routing

Separation of virtual and physical addresses

Centralized control plane

Network Virtualization
Case Study: NVP

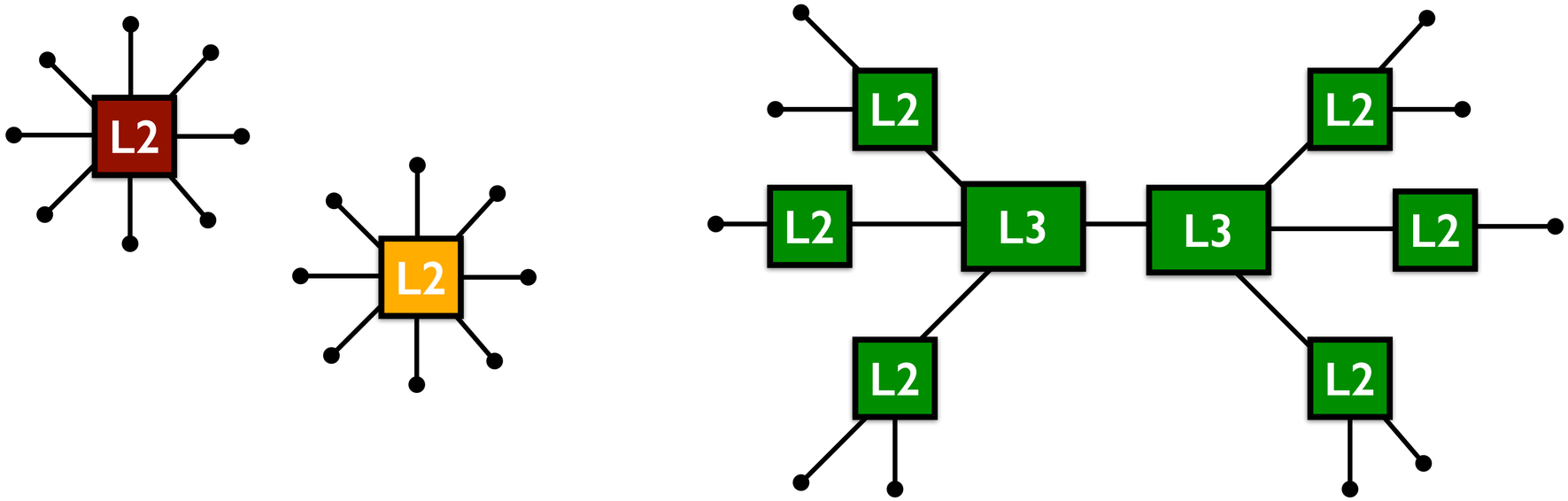
Case study: NVP

[USENIX NSDI 2014]

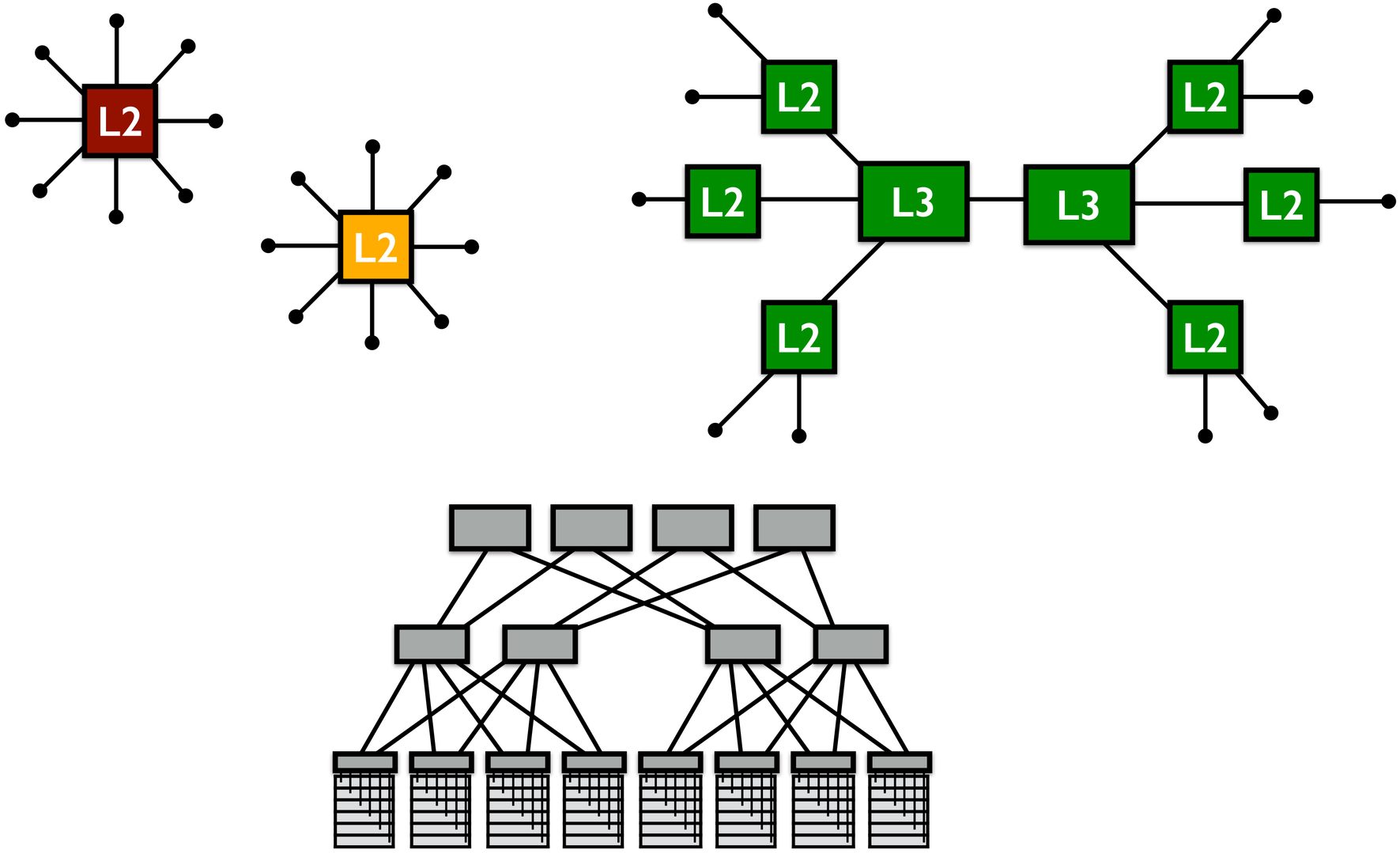
Network Virtualization in Multi-tenant Datacenters

Teemu Koponen, Keith Amidon, Peter Balland, Martín Casado, Anupam Chanda, Bryan Fulton, Igor Ganichev, Jesse Gross, Natasha Gude, Paul Ingram, Ethan Jackson, Andrew Lambeth, Romain Lenglet, Shih-Hao Li, Amar Padmanabhan, Justin Pettit, Ben Pfaff, and Rajiv Ramanathan, *VMware;* Scott Shenker, *International Computer Science Institute and the University of California, Berkeley;* Alan Shieh, Jeremy Stribling, Pankaj Thakkar, Dan Wendlandt, Alexander Yip, and Ronghua Zhang, *VMware*

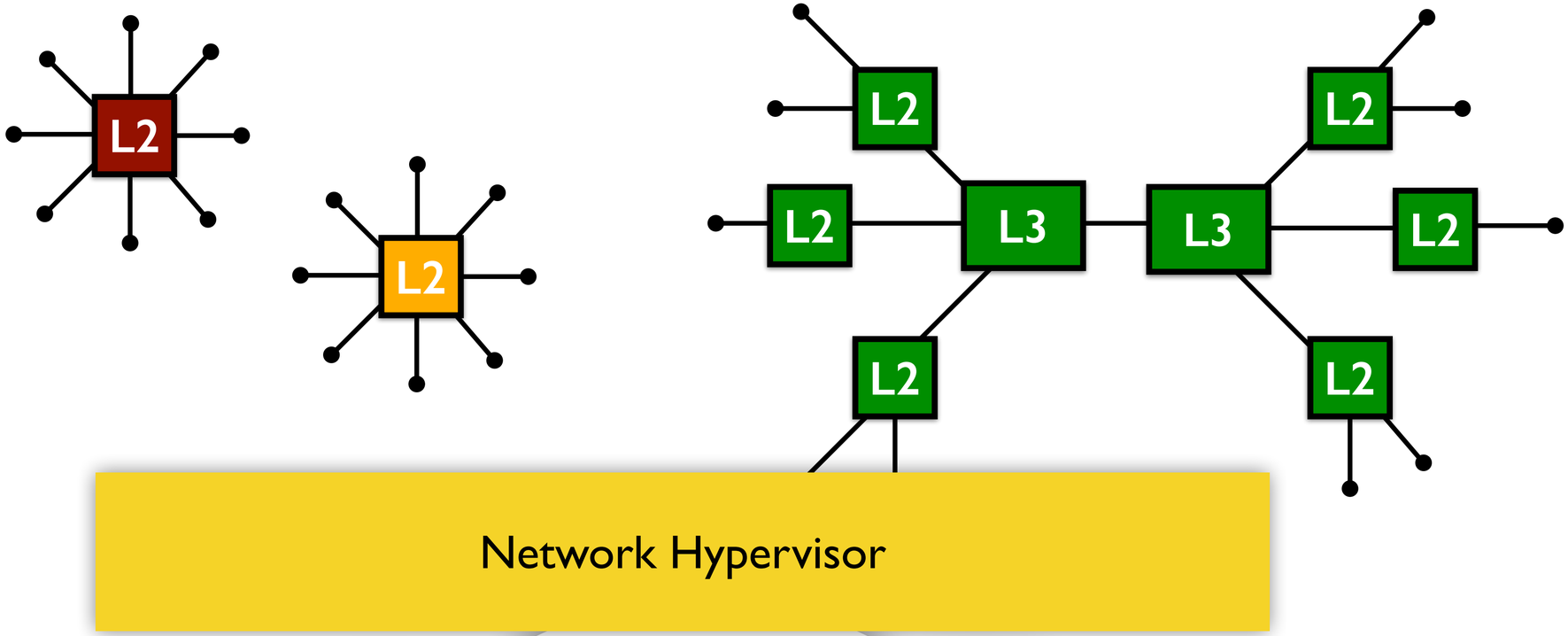
NVP approach to virtualization



NVP approach to virtualization



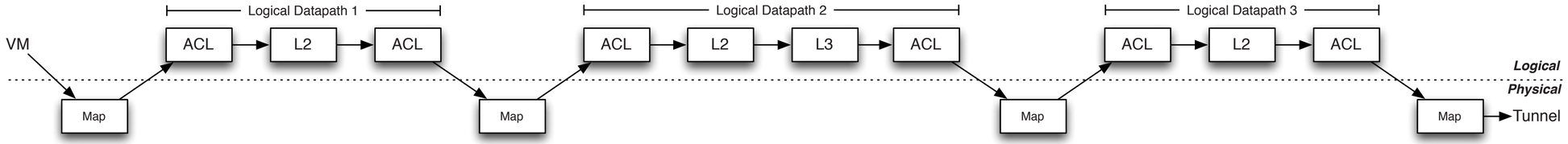
NVP approach to virtualization



Virtual network service

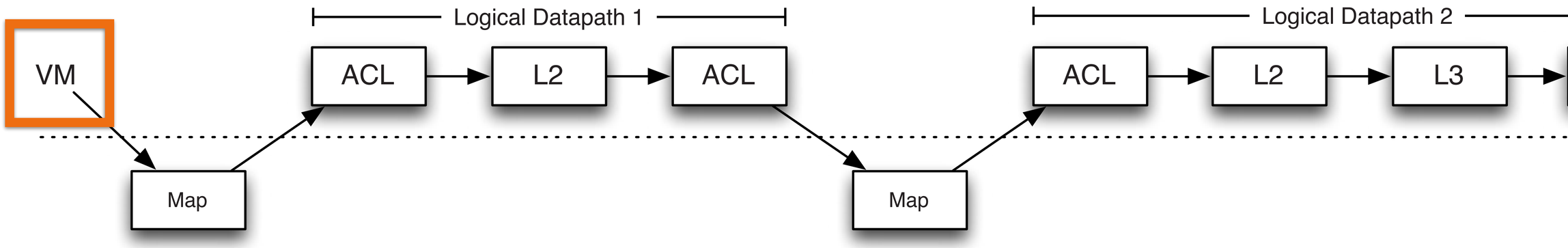
[Figure: Koponen et al.]

Virtual network service

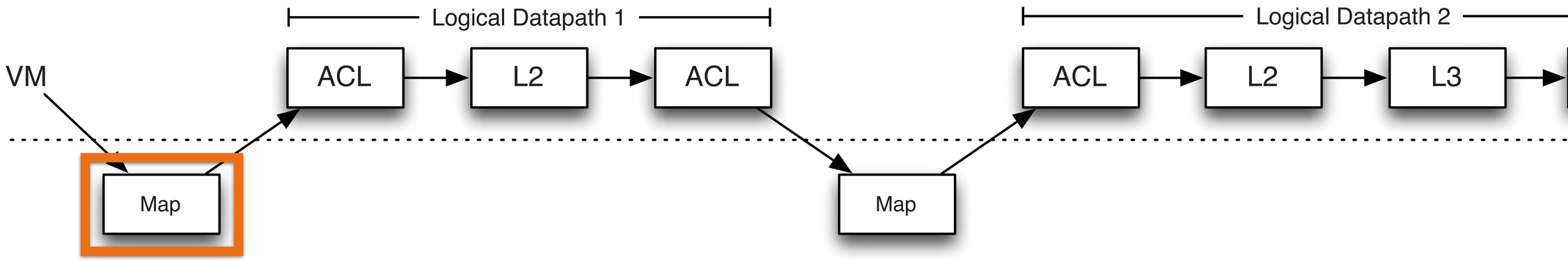


[Figure: Koponen et al.]

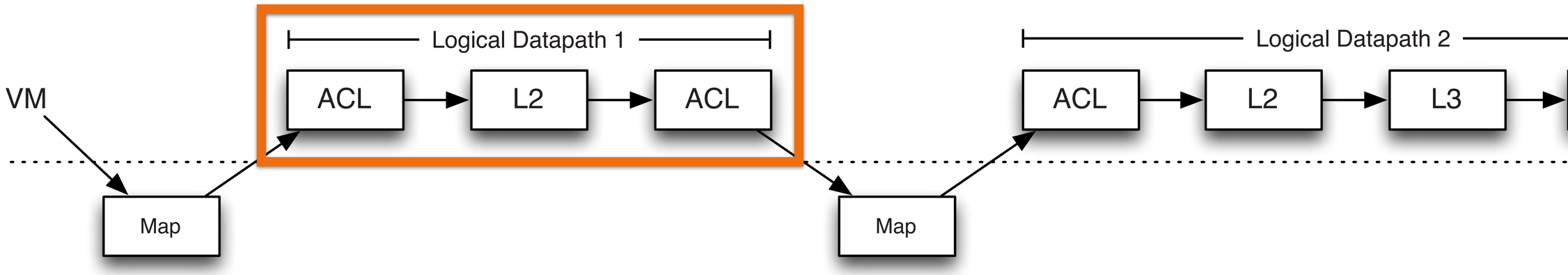
Virtual network service



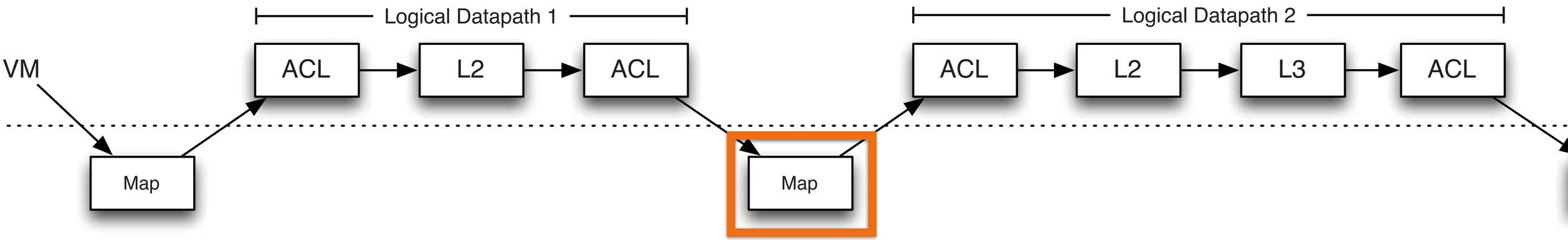
Virtual network service



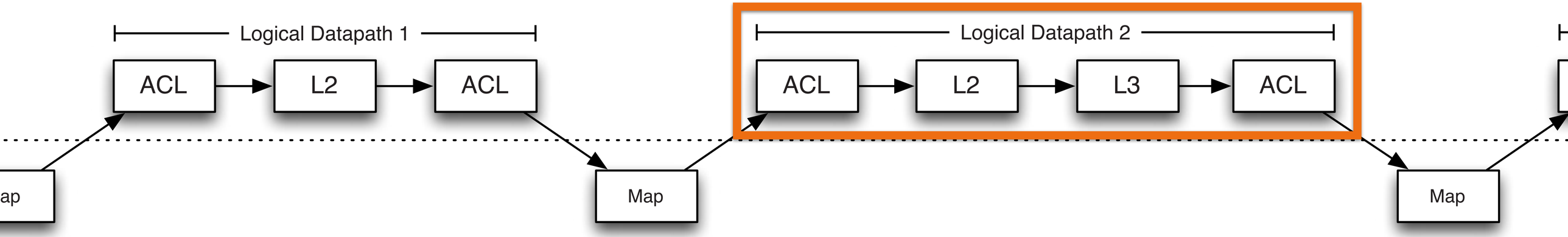
Virtual network service



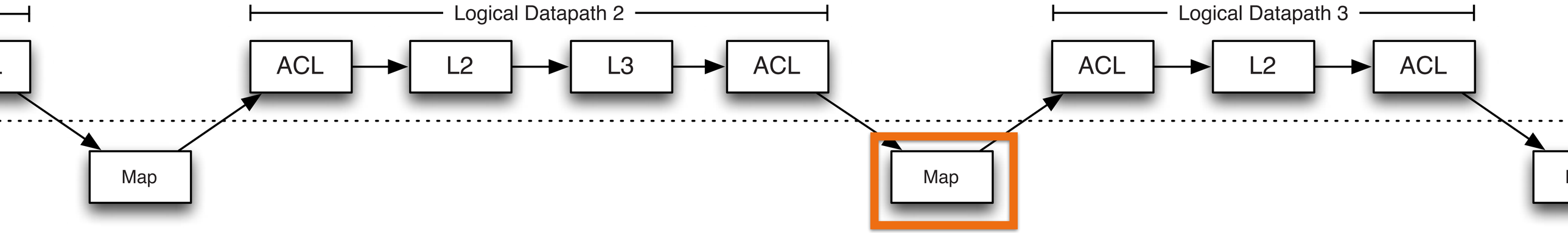
Virtual network service



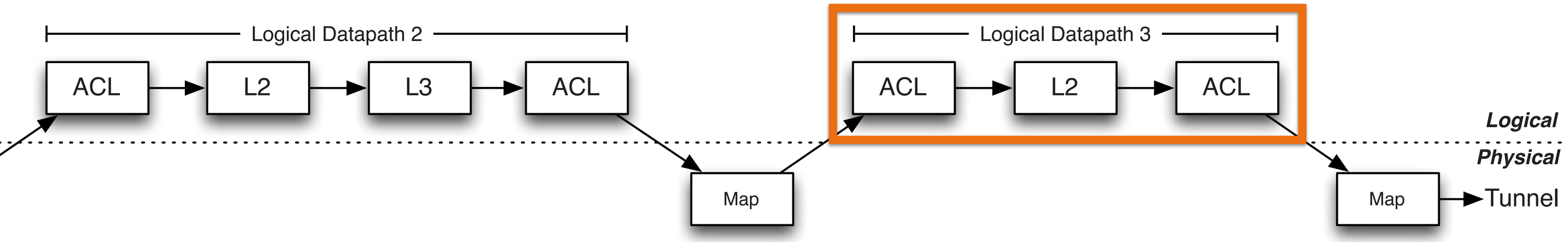
Virtual network service



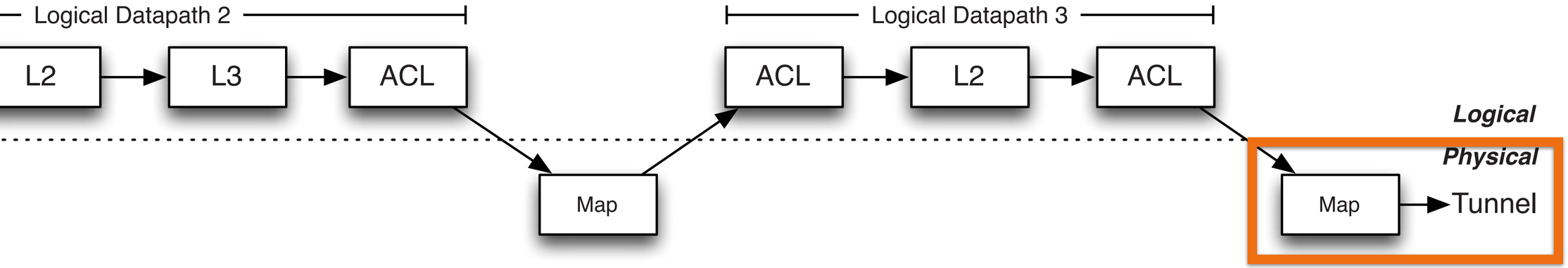
Virtual network service



Virtual network service

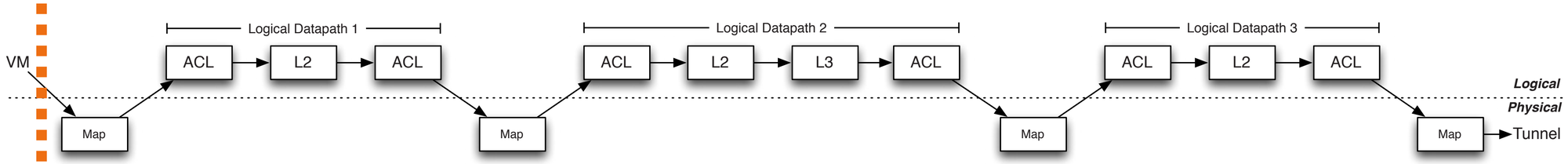


Virtual network service



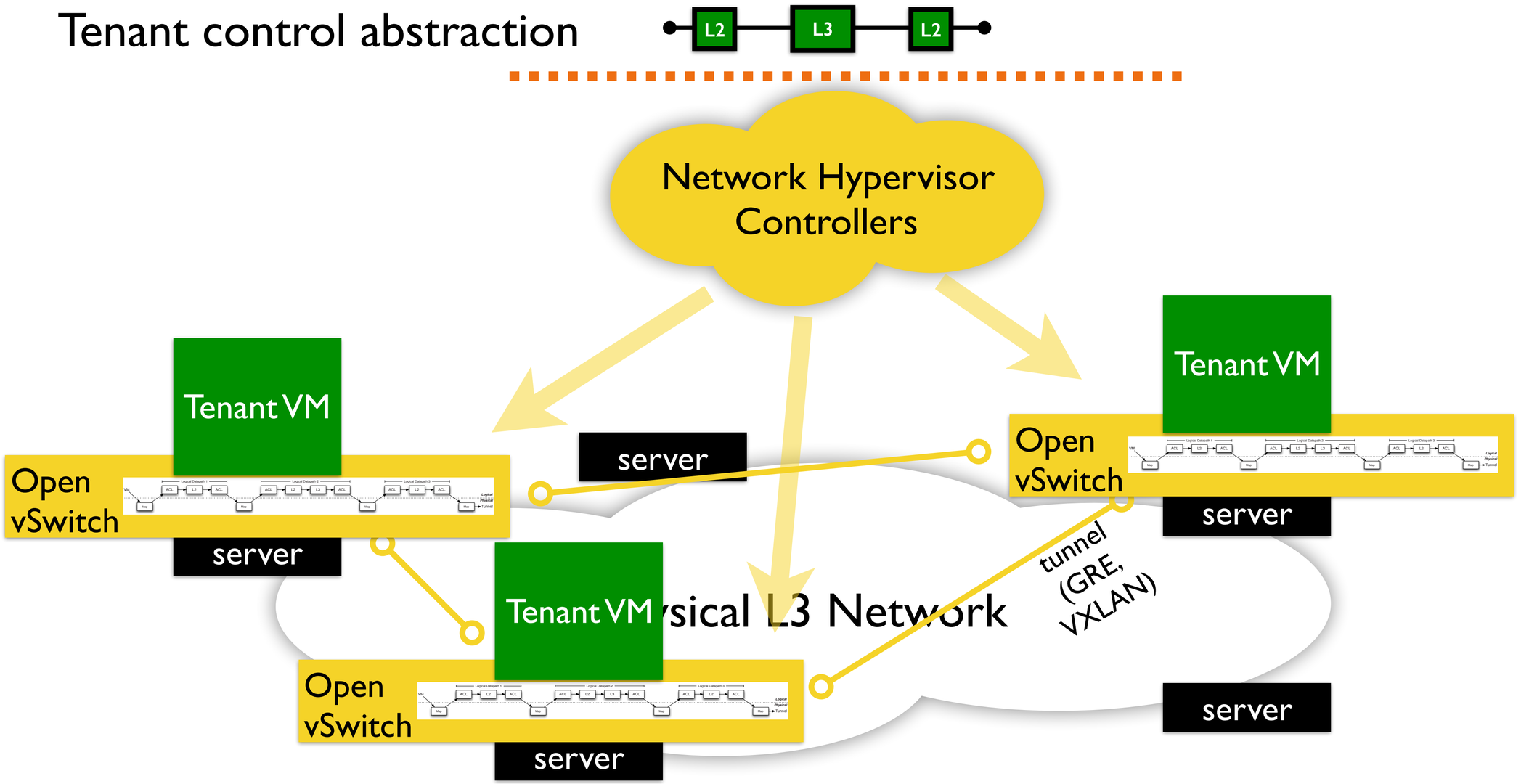
Virtual network service

Control abstraction
(sequence of OpenFlow flow tables)



Packet
abstraction

Big system picture



Challenge: Performance

Large amount of state to compute

- Full virtual network state at every host with a tenant VM!
- $O(n^2)$ tunnels for tenant with n VMs
- Solution 1: Automated incremental state computation with *nlog* declarative language
- Solution 2: Logical controller computes single set of universal flows for a tenant, translated more locally by "physical controllers"

Challenge: Performance

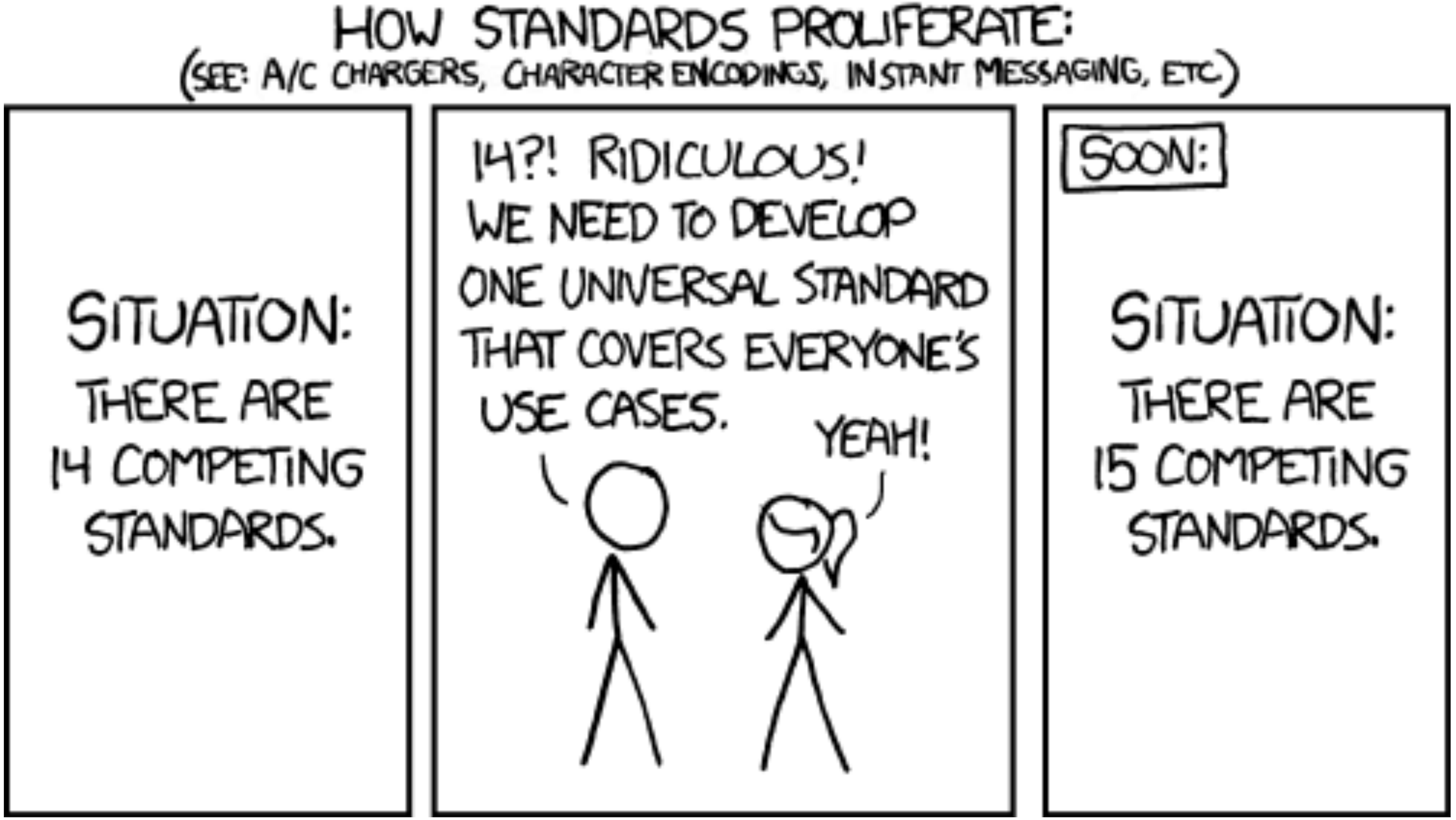
Pipeline processing in virtual switch can be slow

- Solution: Send first packet of a flow through the full pipeline; thereafter, put an exact-match packet entry in the kernel

Tunneling interferes with TCP Segmentation Offload (TSO)

- NIC can't see TCP outer header
- Solution: STT tunnels adds "fake" outer TCP header

So, are we there, yet?


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)
SITUATION:
THERE ARE
14 COMPETING
STANDARDS.
14?! RIDICULOUS!
WE NEED TO DEVELOP
ONE UNIVERSAL STANDARD
THAT COVERS EVERYONE'S
USE CASES.
YEAH!
SOON:
SITUATION:
THERE ARE
15 COMPETING
STANDARDS.

[xkcd.com]

How many extensions do you need?

Version	Date	Header Fields
OF 1.0	Dec 2009	12 fields (Ethernet, TCP/IPv4)
OF 1.1	Feb 2011	15 fields (MPLS, inter-table metadata)
OF 1.2	Dec 2011	36 fields (ARP, ICMP, IPv6, etc.)
OF 1.3	Jun 2012	40 fields
OF 1.4	Oct 2013	41 fields

Table 1: Fields recognized by the OpenFlow standard

Encapsulation, tunneling, STT …

Network programmability

ACM CCR, 2014

P4: Programming Protocol-Independent Packet Processors

Pat Bosshart[†], Dan Daly[*], Glen Gibb[†], Martin Izzard[†], Nick McKeown[‡], Jennifer Rexford[**], Cole Schlesinger[**], Dan Talayco[†], Amin Vahdat[¶], George Varghese[§], David Walker[**]
[†]Barefoot Networks [*]Intel [‡]Stanford University [**]Princeton University [¶]Google [§]Microsoft Research

- A new stage of the SDN trend
- Programmable switches!
- Hot topic now: papers, startups, industrial involvement

Weekly reading guide

Ankit Singla

TCP in the data center

ACM SIGCOMM, 2009

Safe and Effective Fine-grained TCP Retransmissions for Datacenter Communication

Vijay Vasudevan[1], Amar Phanishayee[1], Hiral Shah[1], Elie Krevat[1],
David G. Andersen[1], Gregory R. Ganger[1], Garth A. Gibson[1,2], Brian Mueller[2]

- TCP was not designed for this
- What are the failure cases?
- How do we modify it for the data center?

TCP in the data center

ACM SIGCOMM, 2009

Safe and Effective Fine-grained TCP Retransmissions for Datacenter Communication

Vijay Vasudevan[1], Amar Phanishayee[1], Hiral Shah[1], Elie Krevat[1],
David G. Andersen[1], Gregory R. Ganger[1], Garth A. Gibson[1,2], Brian Mueller[2]

Patrick Stuedi

- TCP was not designed for this
- What are the failure cases?
- How do we modify it for the data center?