Software-Defined Networking Architecture

Brighten Godfrey

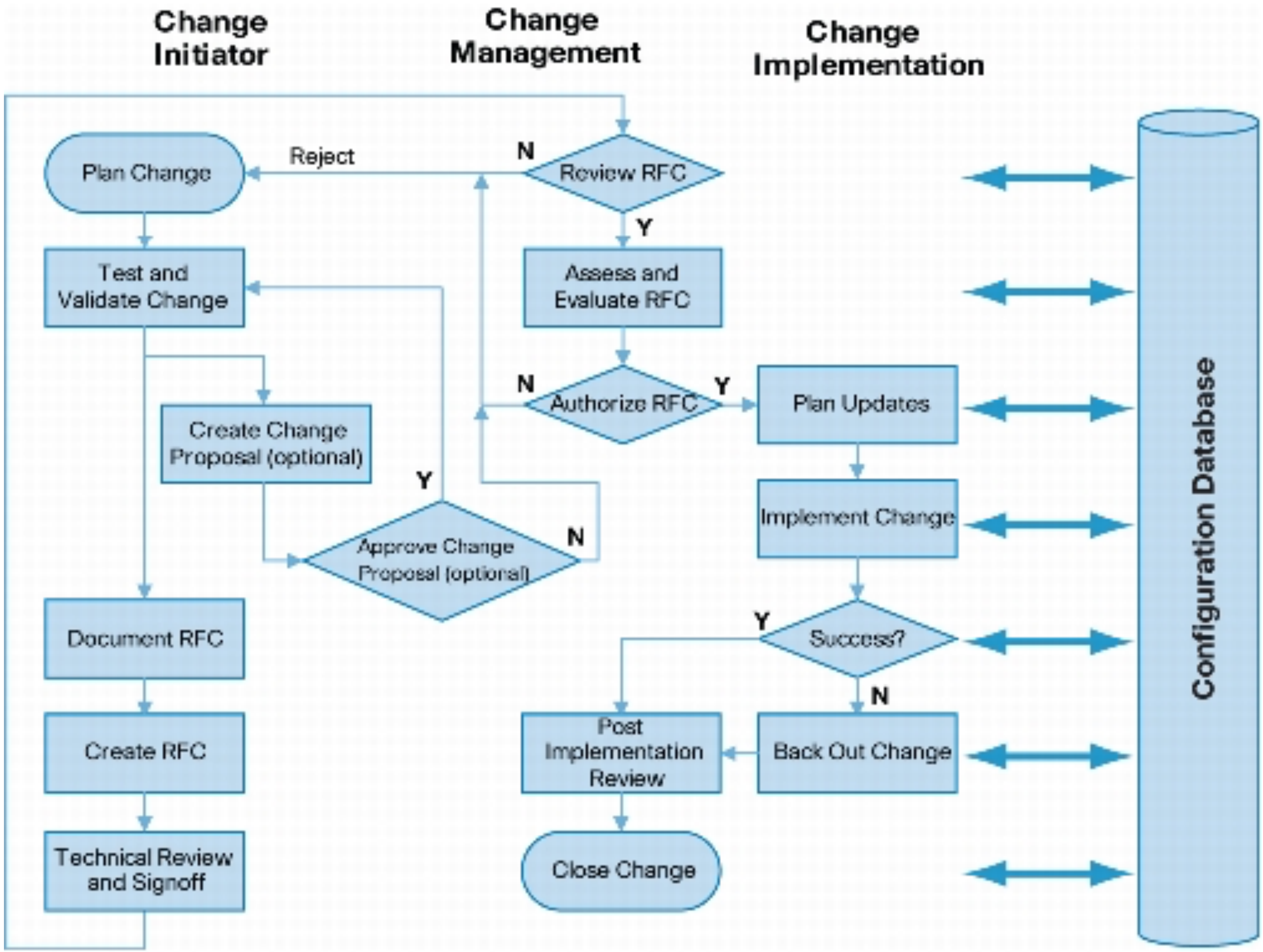
CS 538 February 27 2017

slides ©2010-2017 by Brighten Godfrey

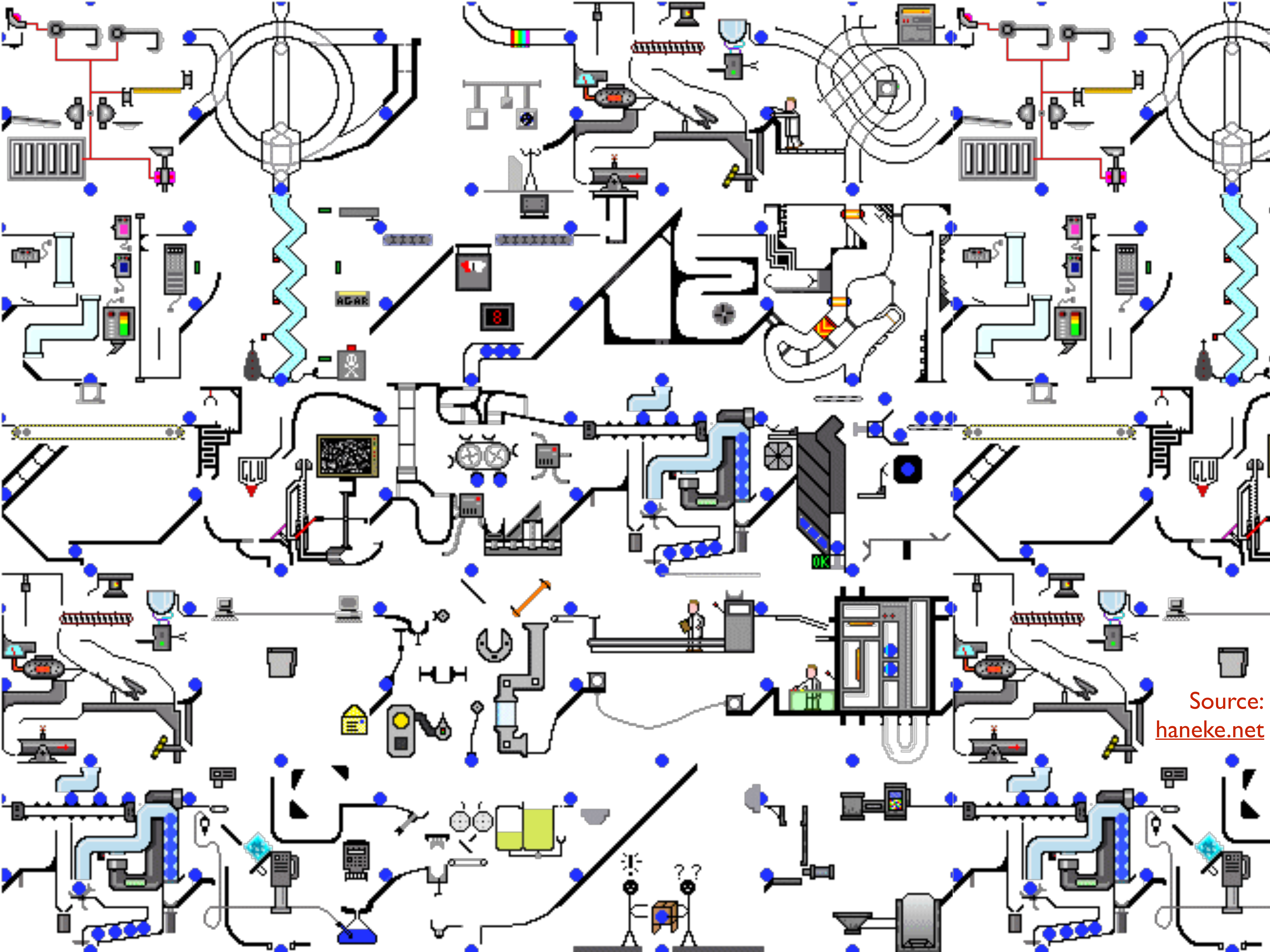
The Problem

Networks are complicated

- Just like any computer system
- Worse: it's distributed
- Even worse: no clean programming APIs, only "knobs and dials"



http://www.cisco.com/c/en/us/products/collateral/services/high-availability/white_paper_c11-458050.html



Source:
haneke.net

The Problem

Networks are complicated

- Just like any computer system
- Worse: it's distributed
- Even worse: no clean programming APIs, only "knobs and dials"

Network equipment is proprietary

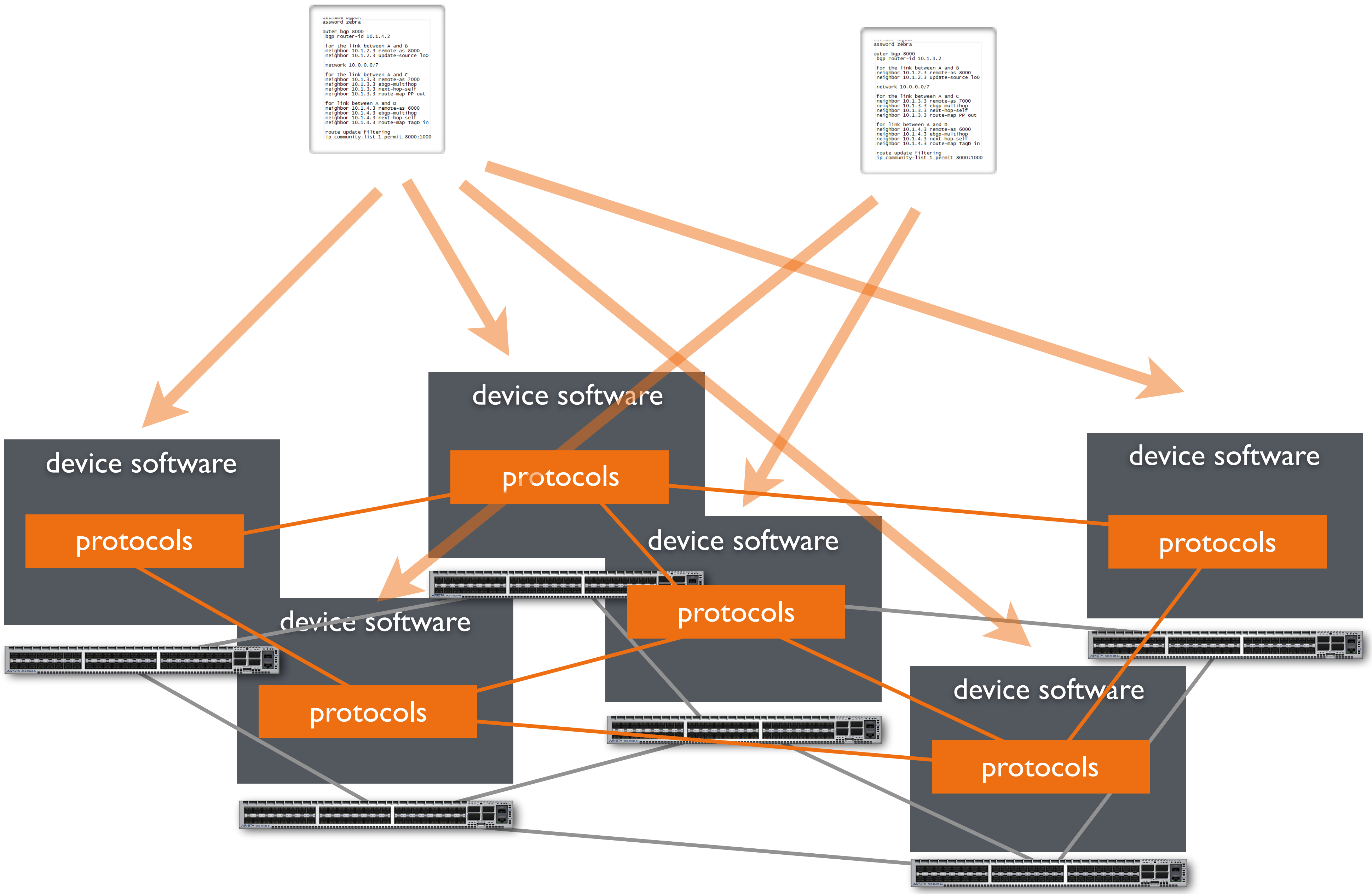
- Integrated solutions (software, configuration, protocol implementations, hardware) from major vendors

Result: Hard to innovate and modify networks

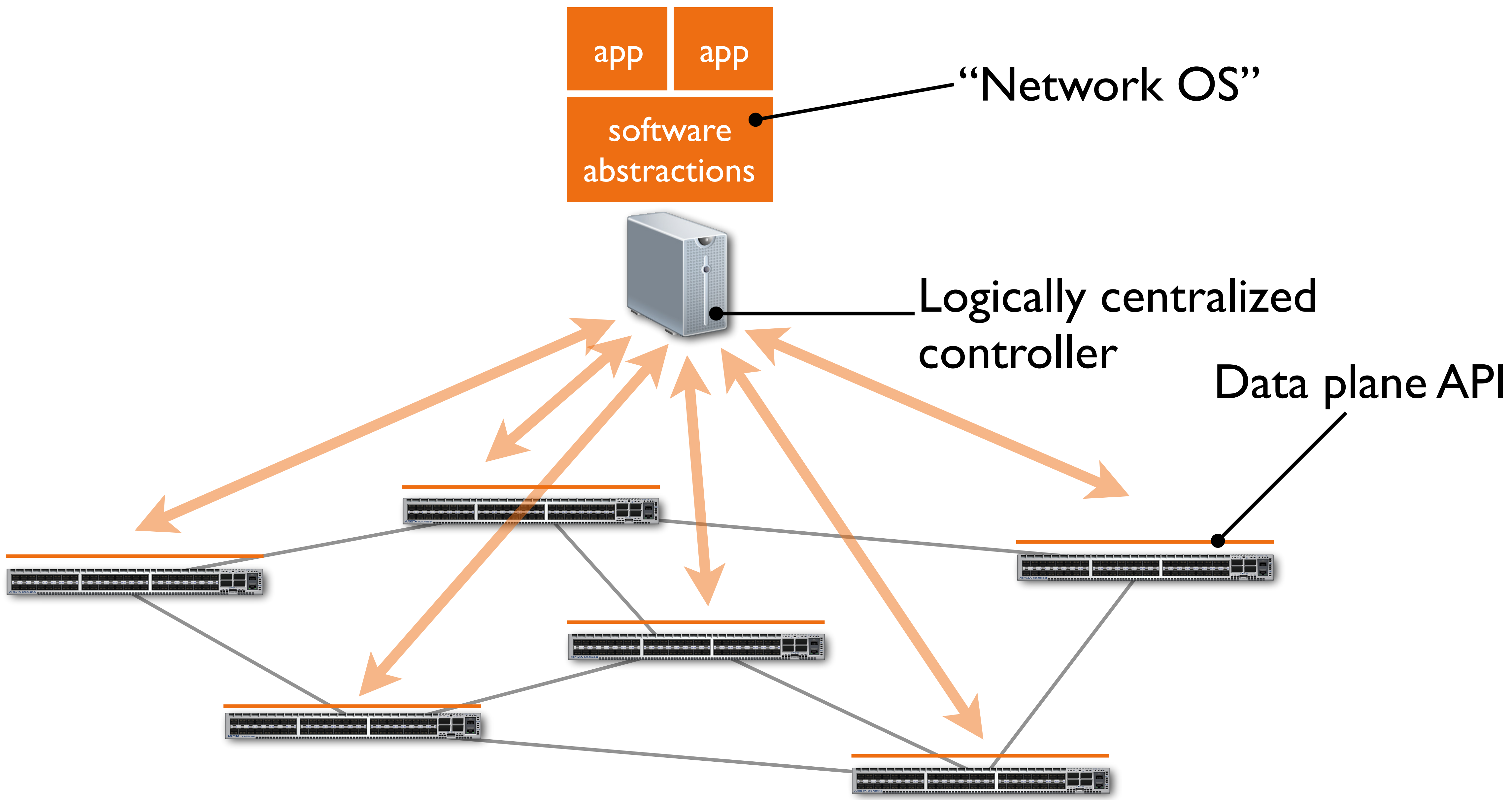
Traditional network

```
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
    attr_in[DL_DST] = nox.reverse_resolve(h
    attr_in[DL_VLAN] = vlanid
    action_in = [(nox.OUTPUT, (0, nox.FLOOD
    install_datapath_flow(dp, attr_in, acti

nox.register_for_user_authentication(setup_
```

Match specific set of packets

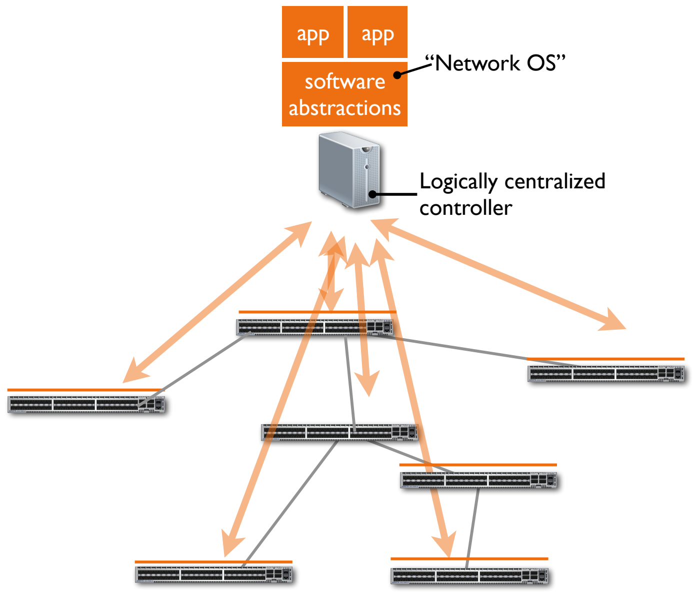
Construct action

Install (match, action) in a specific switch

Common primitives:

- Match packets, execute actions (rewrite, forward packet)
- Topology discovery
- Monitoring

Evolution of SDN



[Graphic: José-Manuel Benitos]

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

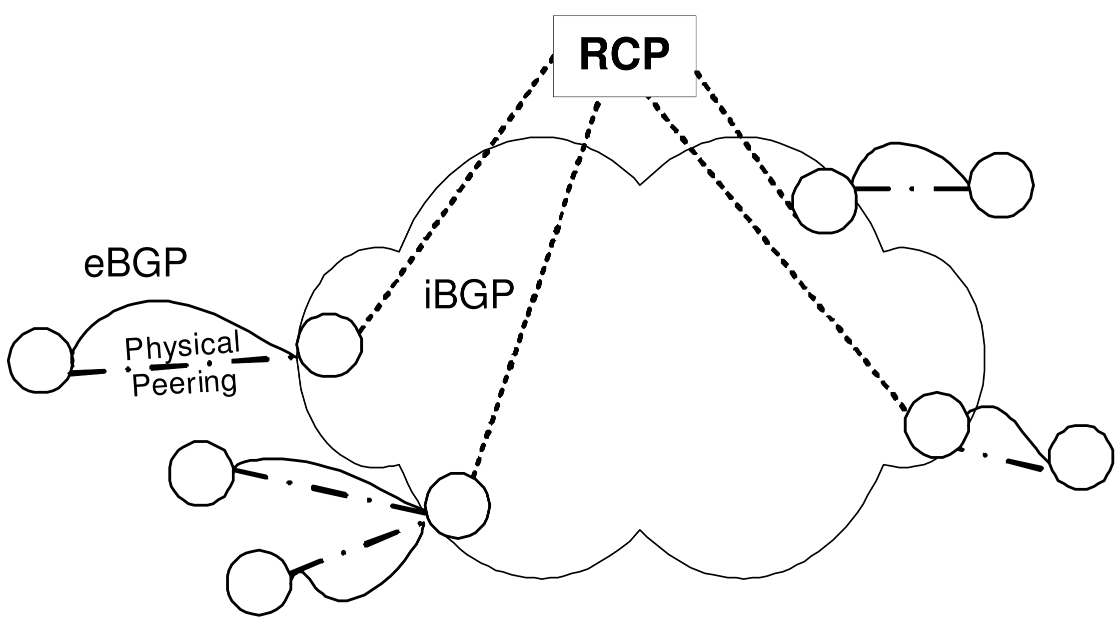
Active Networks (1999)

- Packet header carries (pointer to) program code

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

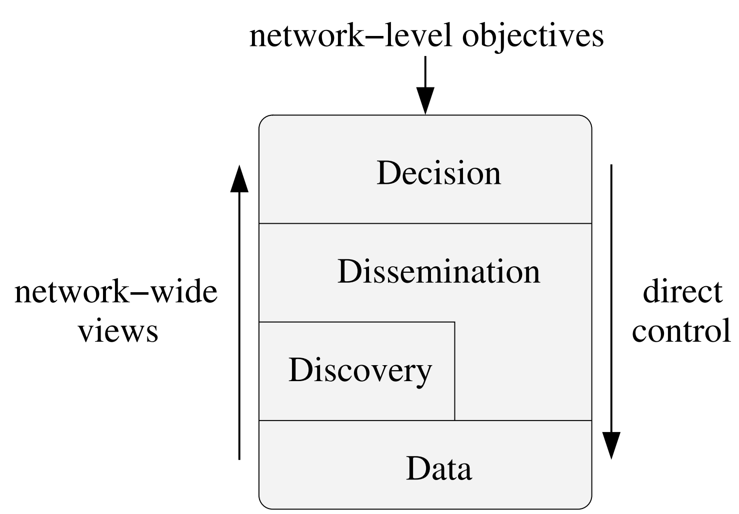


Logically Centralized Control

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



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

Logically Centralized Control

Routing Control Platform (2005)

4D architecture (2005)

Ethane (2007)

- [Casado, Freedman, Pettit, L
SIGCOMM 2007]
- Centralized controller enfo
Ethernet forwarding policy

```
# 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*

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:

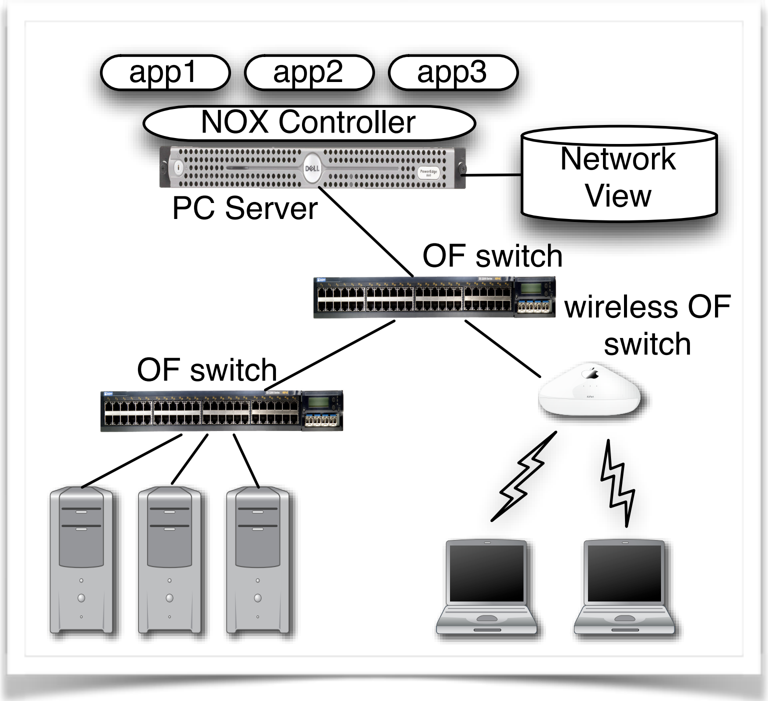
Routing Control Platform (2005)

4D architecture (2005)

Ethane (2007)

OpenFlow (2008)

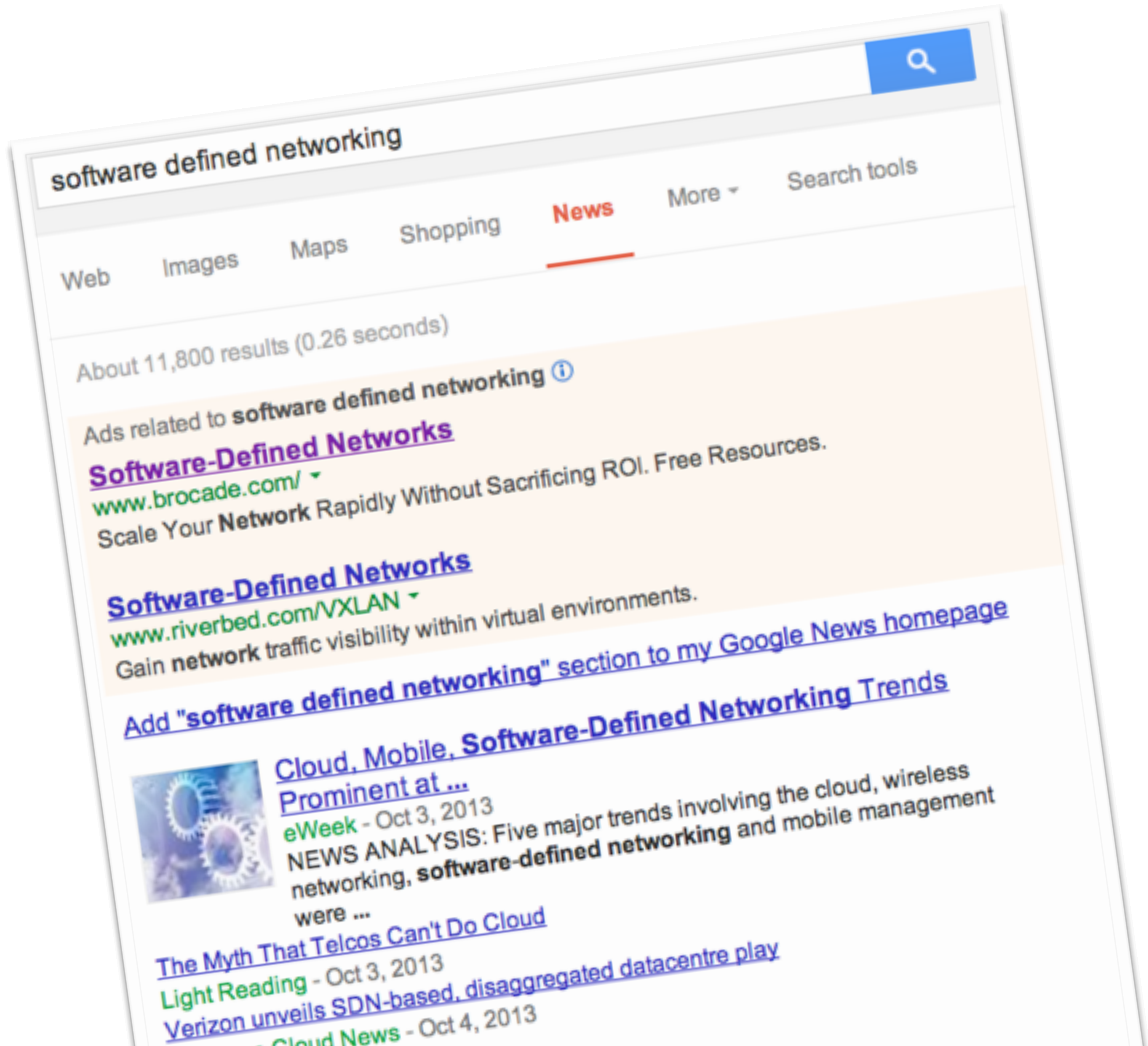
NOX (2008)



- [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: 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 dist. sys. problems once, then just write algorithms
- Libraries/languages to help programmers write net apps
- Systems to write high level policy instead of programming

Opportunities

Open data plane interface

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

Centralized controller

- Direct programmatic control of network

Software abstractions on the cont

- Solve distributed systems proble write algorithms
- Libraries/languages to help programm

All active areas of current research!

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? Whitebox switching? Programmable data planes?

Devising the right control abstractions

- Programming OpenFlow: far too low level
- But what are the right high-level abstractions to cover important use cases?

Q: When do you control the net?

When does the SDN controller send instructions to switches?

- ...in the OpenFlow paper?
- ...other options?

Q: When do you control the net?

When does the SDN controller send instructions to switches?

- ...in the OpenFlow paper? Reactive (when packet arrives needing forwarding rule)
- ...other options? Proactive (in advance of need)

Q: How does SDN affect reliability?

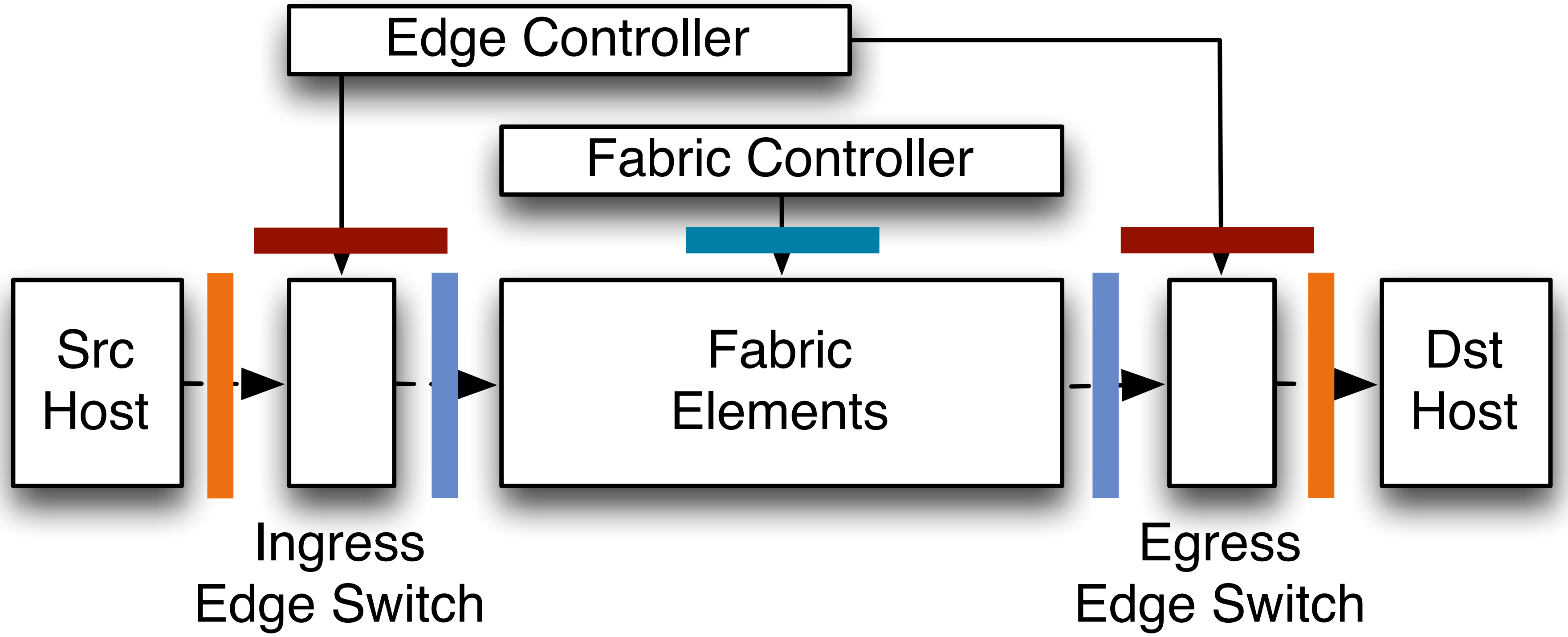
More bugs in the network, or fewer?

From SDN to Fabric

[Casado, Koponen, Shenker, Tootoonchian, HotSDN'12]

Separate interfaces:

- Host-network (external-to-internal data plane)
- Operator-network
- Packet-switch (internal data plane)



Fabric discussion

Q: "Host-Network and Packet-Switch interfaces were identical" in the Internet. How is this a simplification?

Q: Does OF meet the net's goals of:

- Simplified hardware
- Vendor-neutral hardware
- "Future-proof" hardware
- Flexible software

Q: Drivers of early deployment?

What drove early deployment of OpenFlow & SDN?

Access control in enterprises? Net research?

- Good ideas, are already valuable
- But not the "killer apps" for initial large-scale deployment

The first "Killer Apps" for SDN

Inter-datacenter traffic engineering

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

Cloud virtualization

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

Key characteristics of the above use cases

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

Next up

Wednesday: SDN in the WAN

Monday: SDN in the virtualized data center