CSC2229 – Computer Networks for Machine Learning

Handout # 3: Data Center Networks, Data Path vs. Control Programming

Professor Yashar Ganjali
Department of Computer Science
University of Toronto

ganjali7@cs.toronto.edu
http://www.cs.toronto.edu/~yganjali

Announcements

- List of papers for Week 4 are posted.
  - 10% bonus for Week 4 (Jan 31st) volunteers.
  - Volunteers for Week 5?
    - 5% bonus
    - Next week 0%
    - The week after …. ☺

- Each presentation is 20 minutes.
  - Followed by 10 minutes of discussion and Q&A.

- Please read the papers before each class.

Final Project

- List of suggested projects posted on class web site.
  - Very brief and high level.

- Start choosing your team members
  - Use Piazza if needed

- Final Project Topic
  - Consult with the instructor to choose a problem
    - Choose 1-2 problems from the offered list
  - Replicate & Improve
    - Choose a paper related to the course topic, replicate existing solution, and improve (bonus)
  - Survey of existing solutions
    - Create new insights by looking at certain problems/solutions from different perspectives
  - Apply your background/ideas
    - Identify challenges and opportunities in the areas covered in the course to apply your own ideas.

- Please discuss your ideas with me before submitting a proposal.

Outline

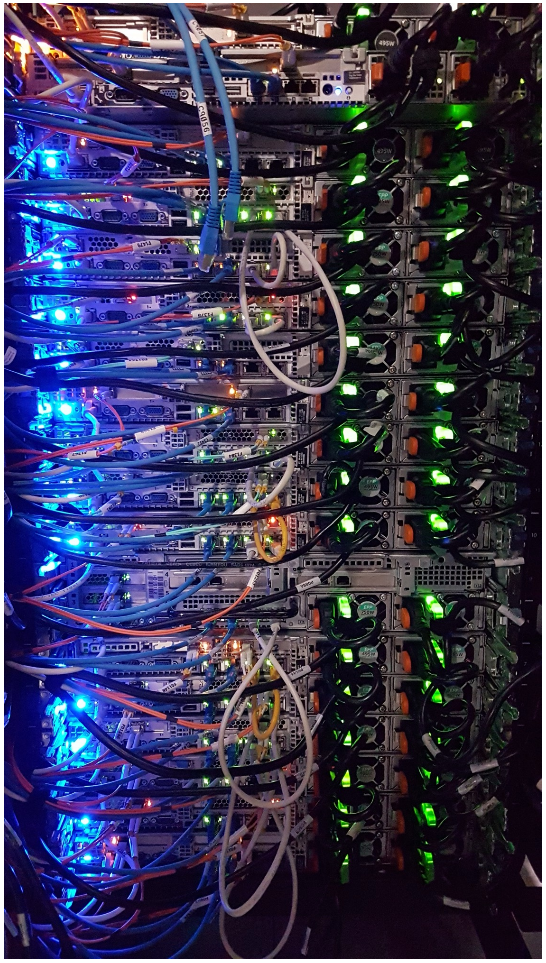
- Data-Center Networks
  - Design
  - Architecture

- Programming Computer Networks
  - Software-Defined Networking
  - Programmable Switches

- Next week:
  - DCN Transport
  - DCN and ML

Data Center Network (DCN)

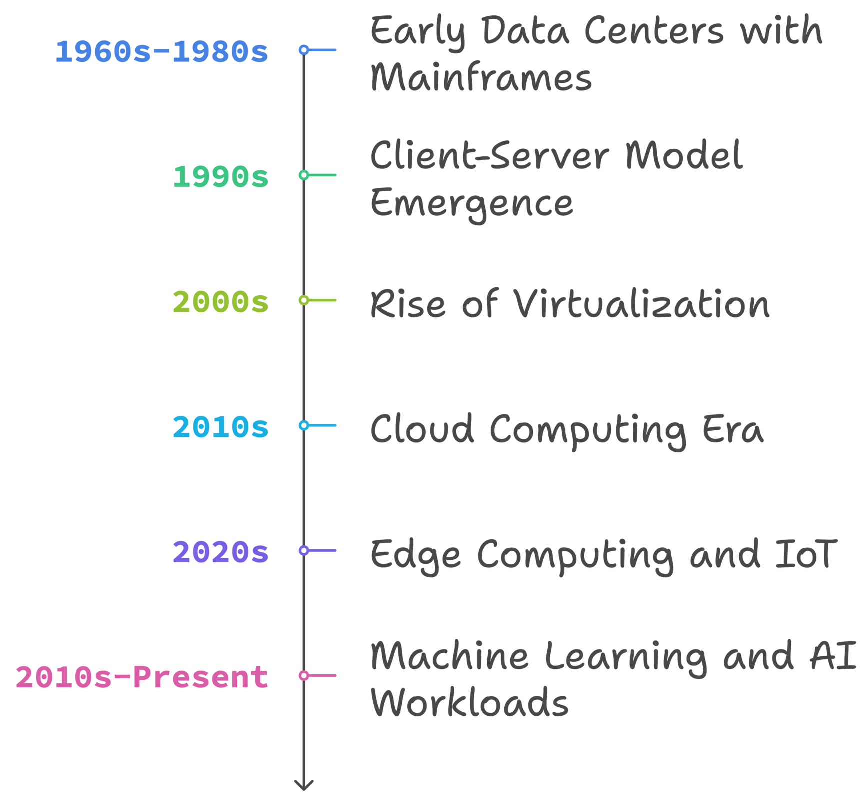
- A network of computing and storage resources
  - Proximity of components (within the data center) facilitates communication, i.e., high-performance
  - Can lead to reduced cost and overheads
    - Major cost upfront but less cost in long run.



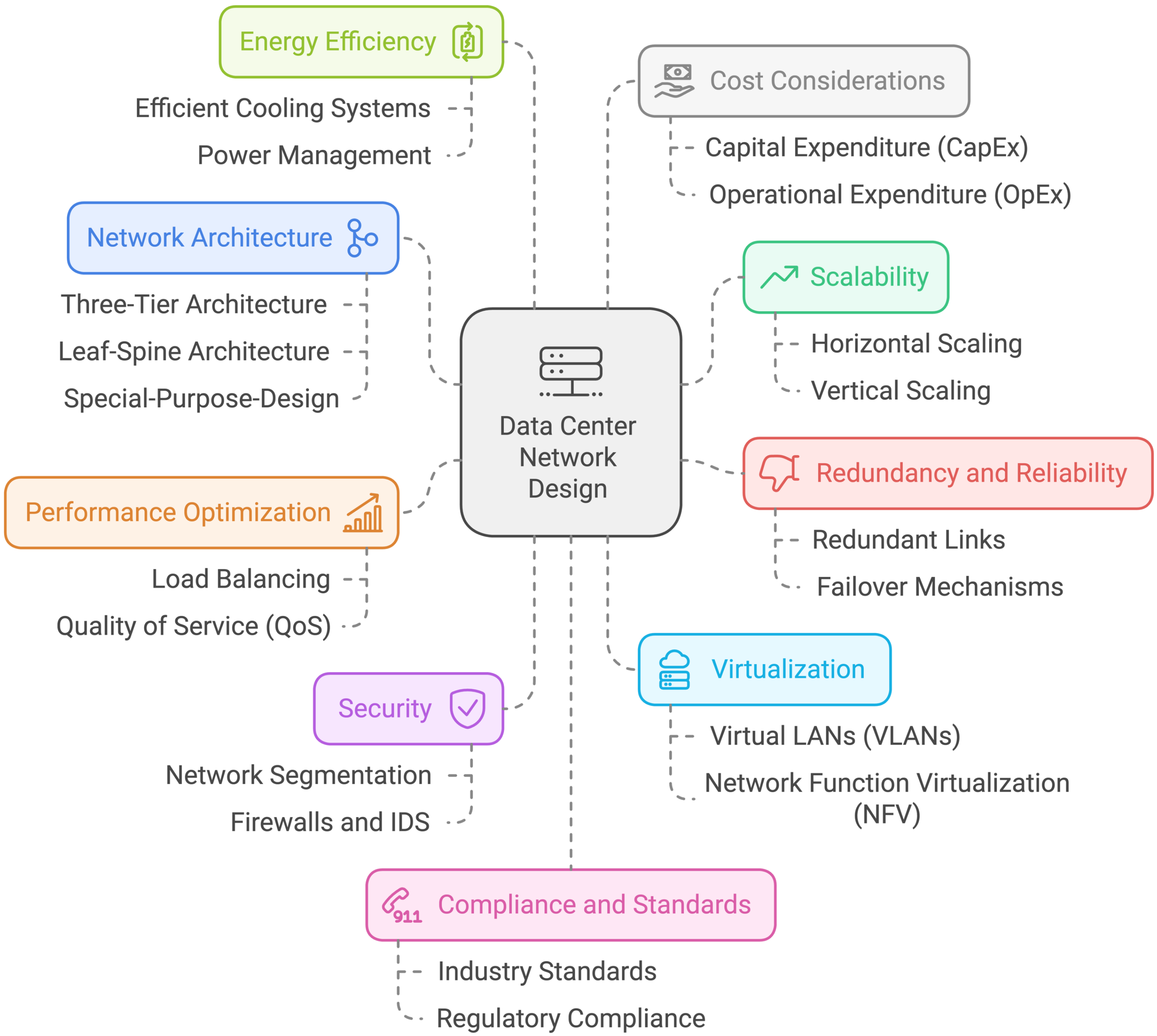
- Functions
  - Data Storage and Management:
    - Security, efficiency, reliability
  - Application Hosting
    - End-users and business applications, cloud computing and SaaS models
  - Data Processing:
    - Large volumes of data for analytics and processing, big data and AI workloads
  - And more …

Evolution of DCN

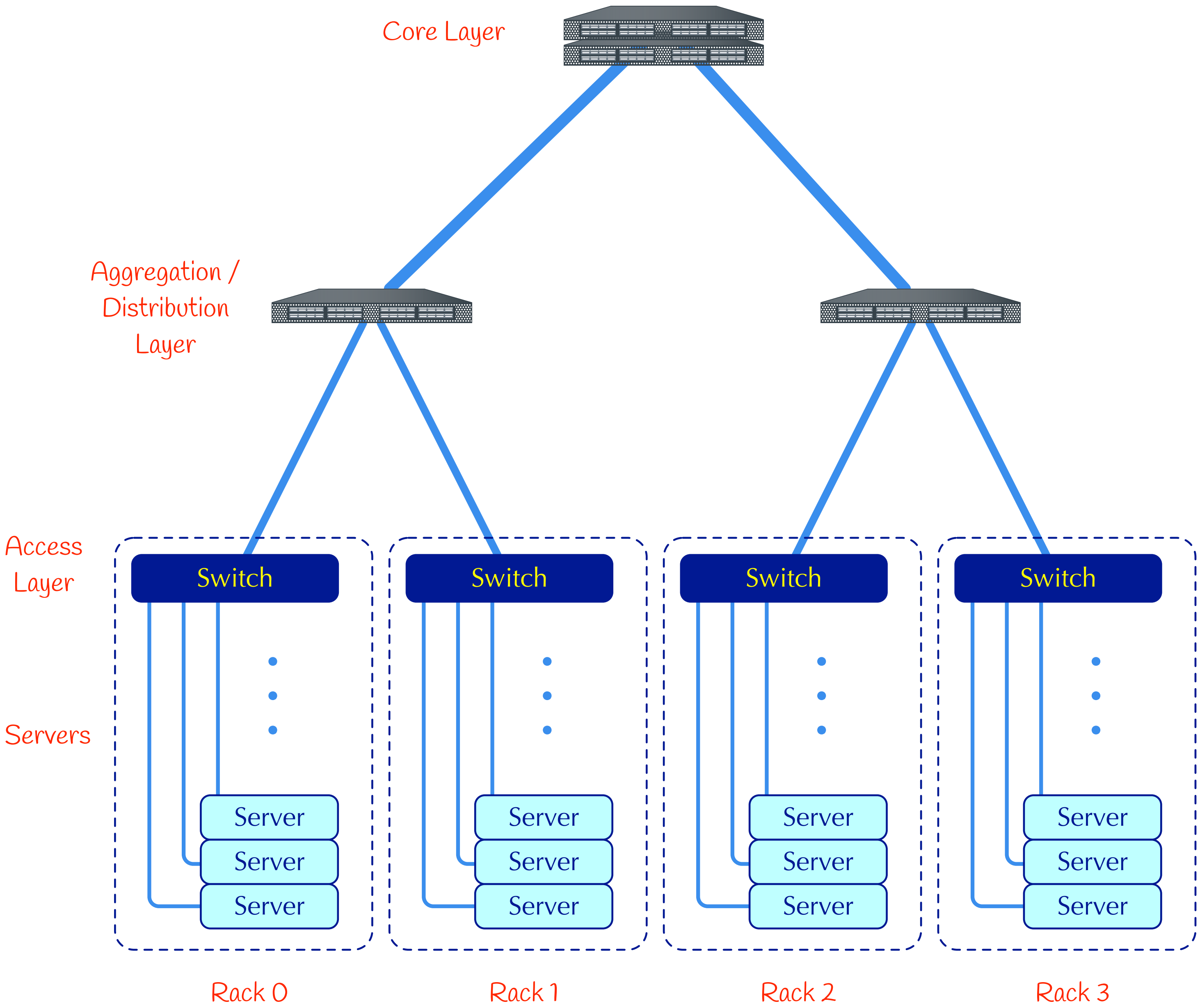
- Early Data Centers (1960s-1980s)
  - Mainframes
  - Centralized computing: limited networking
  - Point-to-point and proprietary connections
- Client-Server Model (1990s)
  - Distributed computing
  - Ethernet, TCP/IP
- Rise of Virtualization (2000s)
  - Virtual machines
  - Efficiency and scalability
  - VLANs, network segmentation
- Cloud Computing Era (2010s)
  - Cloud services
  - SDN: Scalability and automation
- Edge Computing and IoT (2020s)
  - Demand for low-latency, distributed network architectures
  - Micro data centers closer to data sources
- Machine Learning and AI (2010s-Present)
  - Exascale high-performance computing
  - Network as the bottleneck: stringent performance requirements



DCN Design Dimensions



Three-Tier Architecture



Hierarchical tree network topology

- Commonly used in traditional DCNs

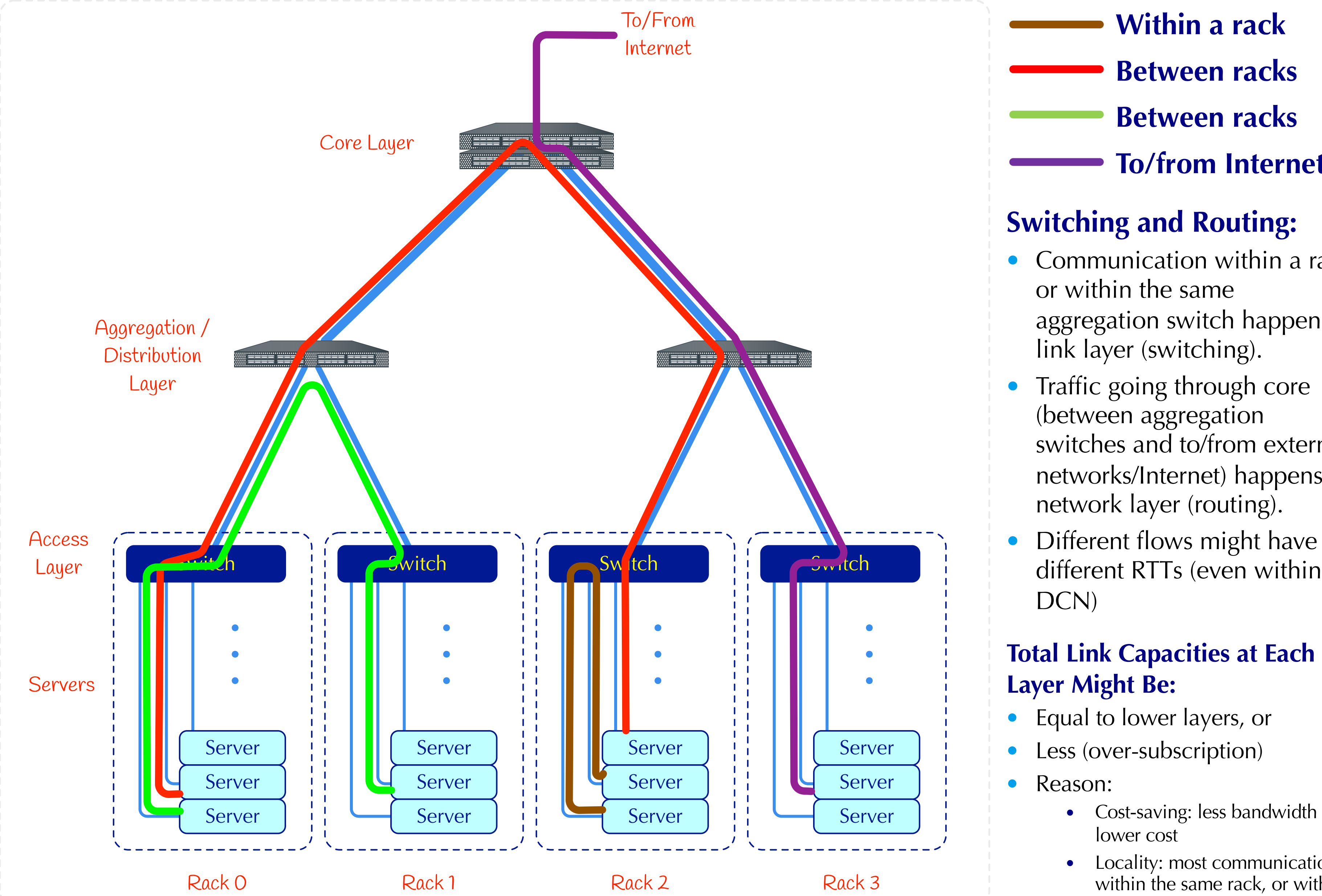
Three layers:

- Core:
  - Layer 3 (network layer)
  - Fully connected high-speed mesh of multiple routers
  - Connect to the external networks
- Aggregation:
  - Layer 3 and 2 (network and link layers)
  - Connect to core with few high-speed links (e.g., 100 Gb/s links), to access with many low-speed links (e.g., 10Gb/s) → simplify cabling
  - Middleboxes sit here (firewall, load balancer, …)
- Access: layer 2 (link)
  - Connect to each server in the rack (e.g., through one or two 10Gb/s links)
  - VLANs used to limit broadcast

Modular design

- Easy to expand

Traffic Direction in 3-Tier Architecture



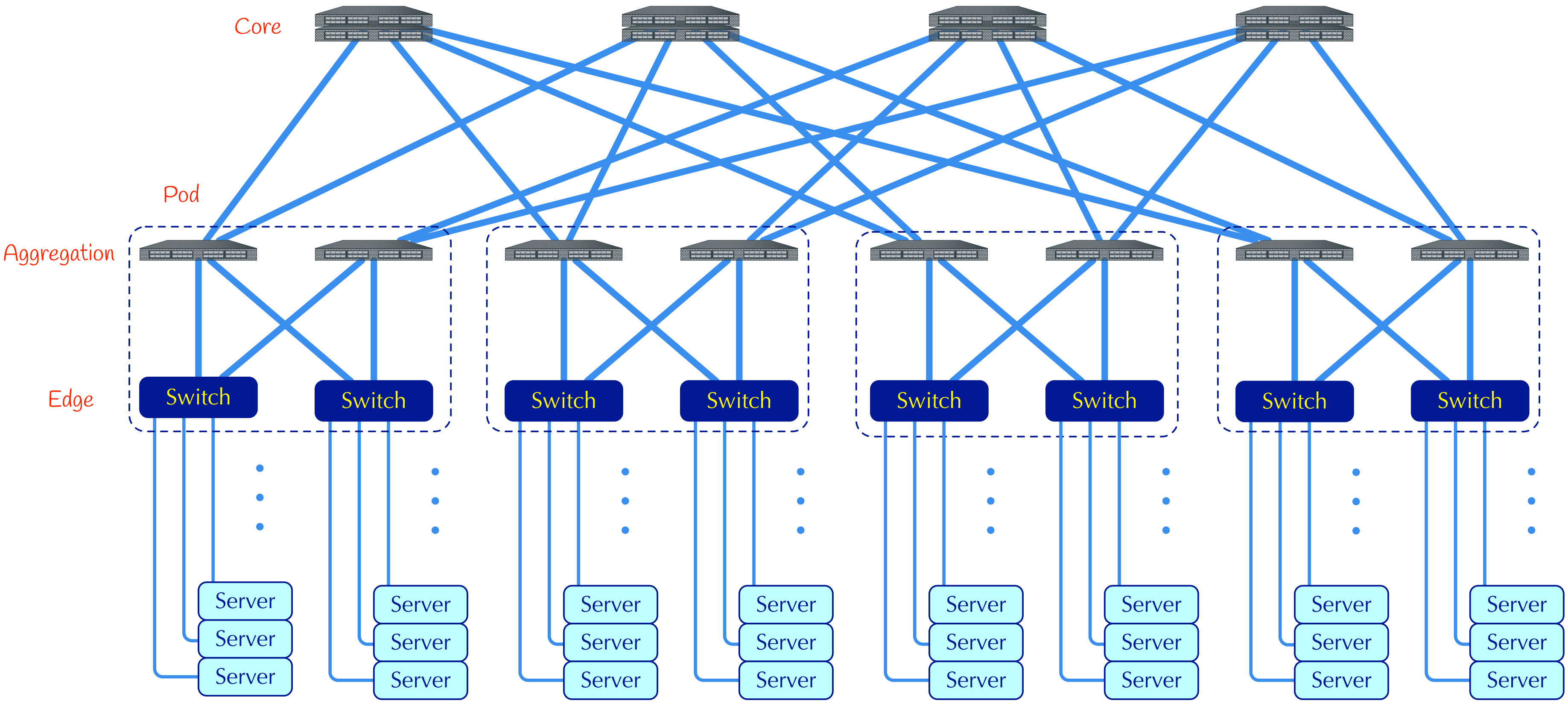
Switching and Routing:

- Communication within a rack or within the same aggregation switch happens at link layer (switching).
- Traffic going through core (between aggregation switches and to/from external networks/Internet) happens at network layer (routing).
- Different flows might have different RTTs (even within DCN)

Total Link Capacities at Each Layer Might Be:

- Equal to lower layers, or
- Less (over-subscription)
- Reason:
  - Cost-saving: less bandwidth → lower cost
  - Locality: most communication within the same rack, or within the same cluster.

Fat-Tree Architecture



Properties of Fat-Tree Architecture:

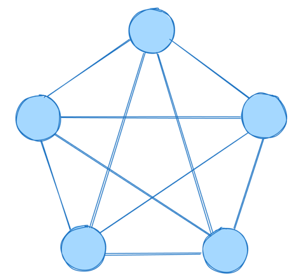
- Scalable: easily expandable
- Low latency, high throughput
- Cost-effective
  - Commodity hardware
  - Lower operational costs

Reliability and Improved Performance:

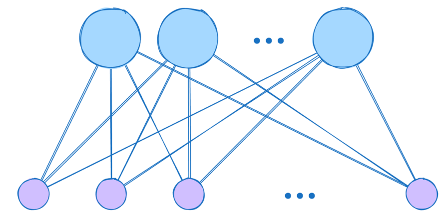
- Edge and aggregation switches are grouped into "pods".
  - Multiple-paths (choice of aggregation and core)
  - Automatic failover
- Leads to better load balance, redundancy, and thus
  - Reliability and high availability, and
  - Improved performance

Other DCN Architectures/Topologies

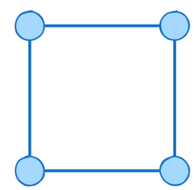
- Mesh: every node is connected to every other node
  - Direct communication, costly, but high performance
- Leaf-Spine topology: two-tier structure, servers and storage node connect directly to leaf switches
  - Switched environment, VLANs to limit broadcasts
- Hyper-cube: multi-dimensional cube structure
  - Used in high-performance computing and ML solutions
- ToR (Top-of-Rack) vs. EoR (End-of-Row)
- Hybrid: combine two or more topologies
  - Tailored to specific requirements of DCNs
- And many more …
- Question: what are the properties of each of these topologies?
  - End-to-end latency?
  - Simplicity?
  - Scalability?
  - …



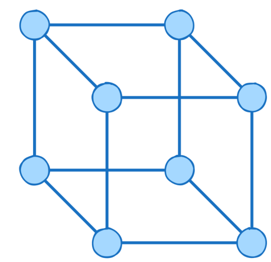
Mesh Topolog



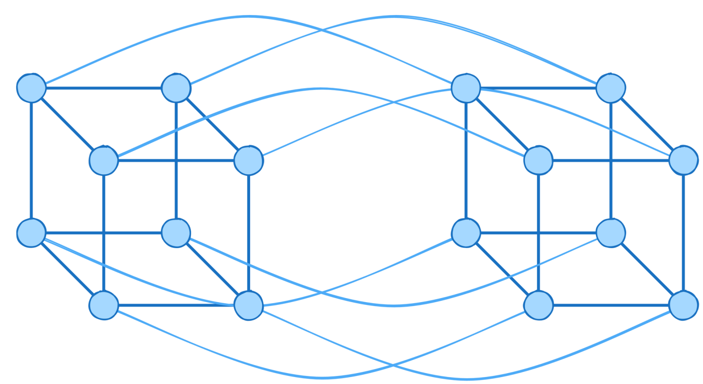
Leaf-Spine Topology



Two Dimensional Hypercube



Three Dimensional Hypercube



Four Dimensional Hypercube

Programming Data Center Networks

- DCN Primary Objective:
  - Transfer Packets

- Also, provide high performance (bandwidth, latency, loss requirements)
  - As well as new functions (e.g., network virtual functions or NFV, …),

- Traditionally, forwarding, routing, load balancing, … implemented in a combination of hardware, software running on network devices
  - Extremely difficult to change
  - New functions take years

Traditional Networks

Routing, management, mobility management, access control, VPNs, …

Feature

Feature

OS

Custom Hardware

Million of lines of source code

9,000 RFCs

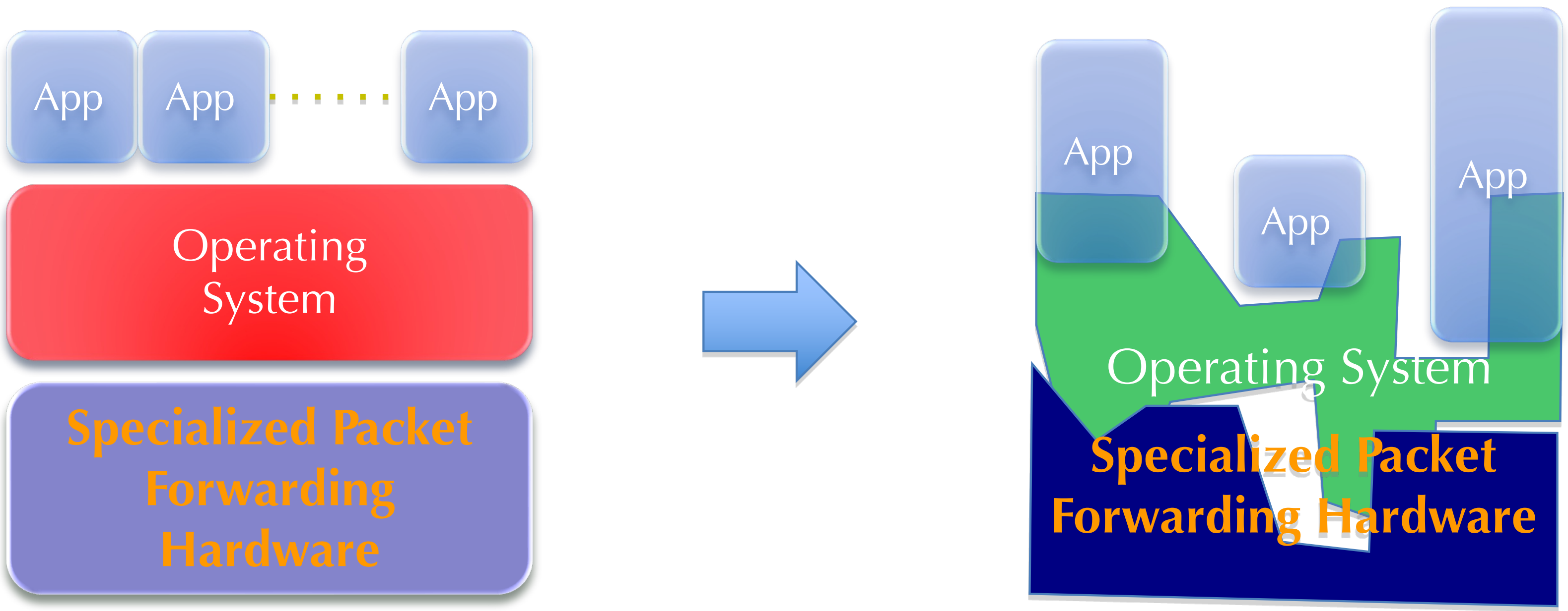
Billions of gates

Bloated

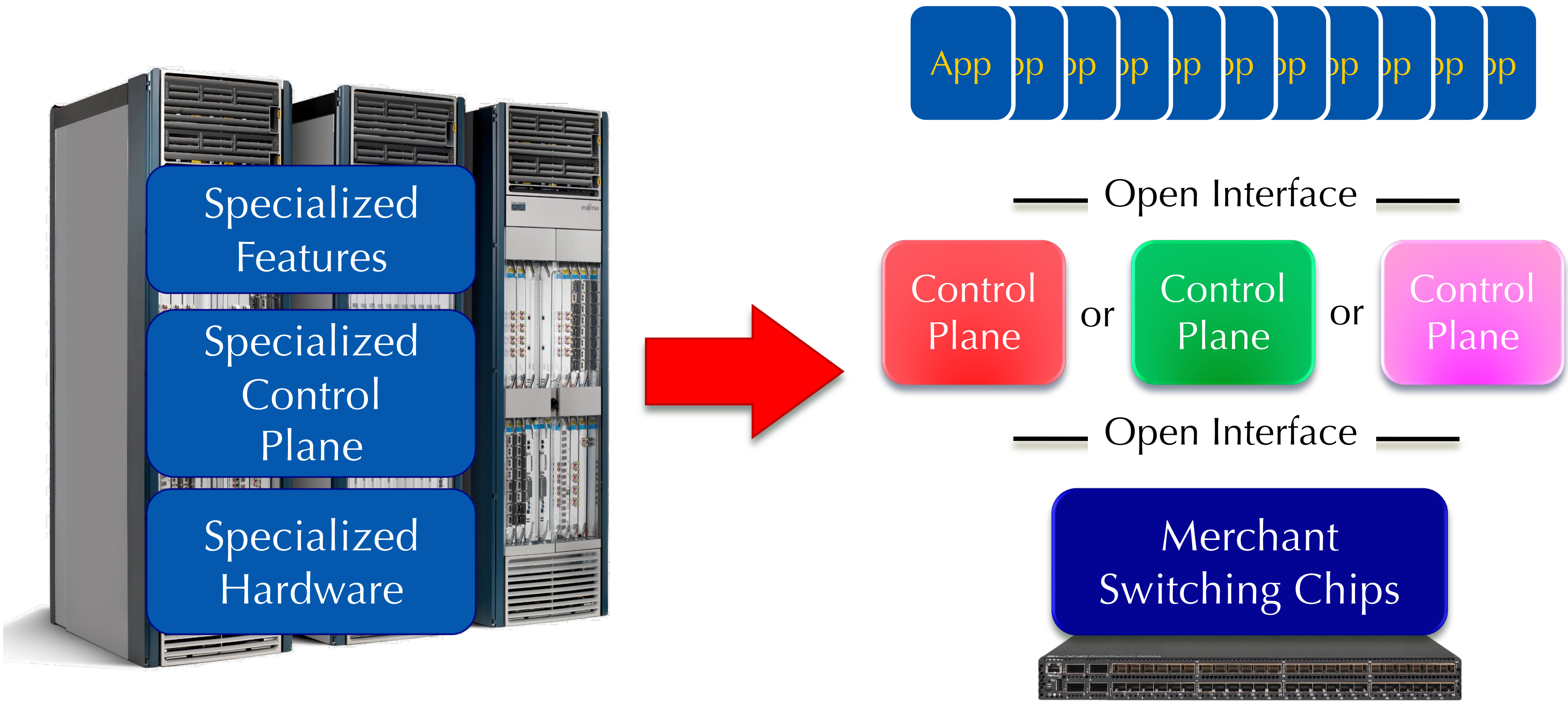
Power Hungry

- Vertically integrated, complex, closed, proprietary
- Networking industry with “mainframe” mind-set

Reality is Even Worse



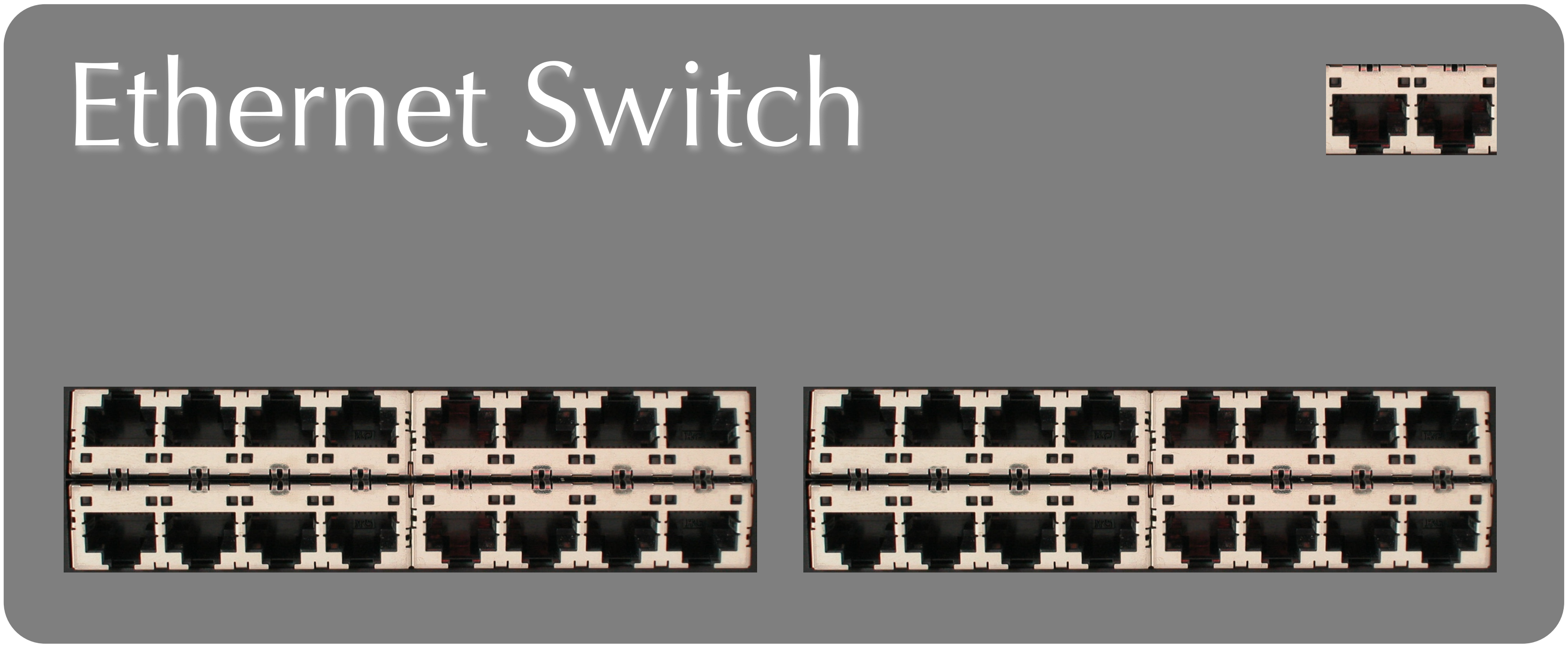
- Lack of competition means glacial innovation
- Closed architecture means blurry, closed interfaces



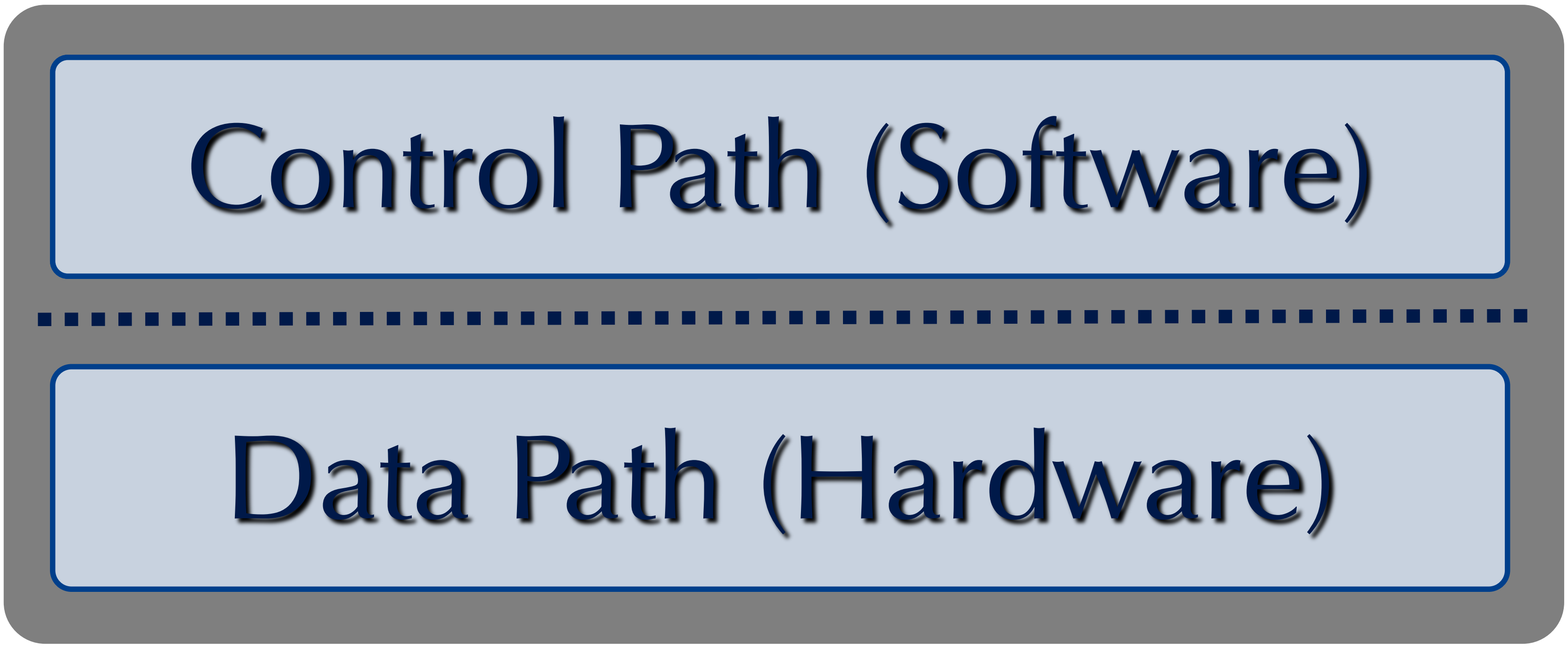
Vertically integrated
Closed, proprietary
Slow innovation

Horizontal
Open interfaces
Rapid innovation

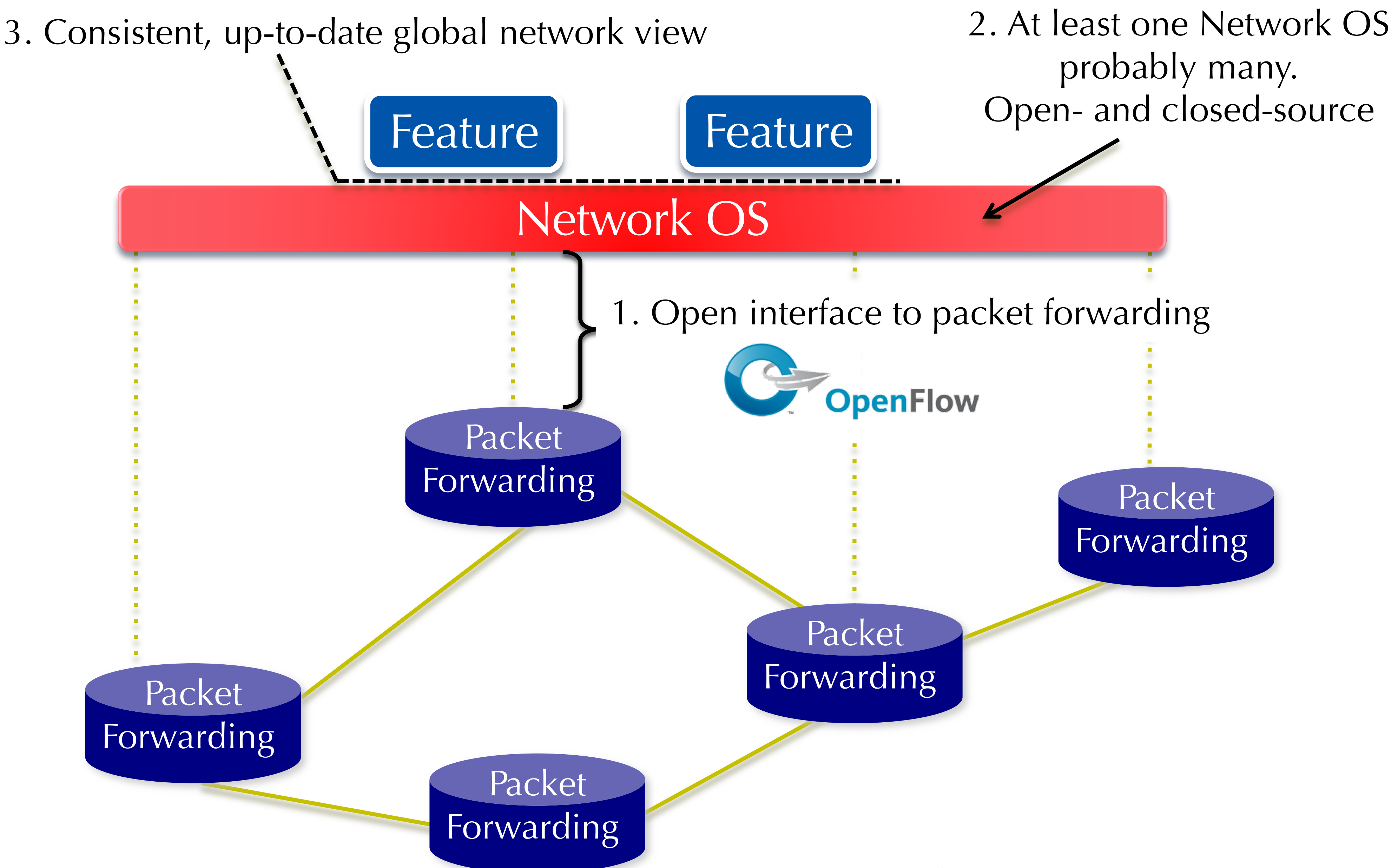
Traditional Switch



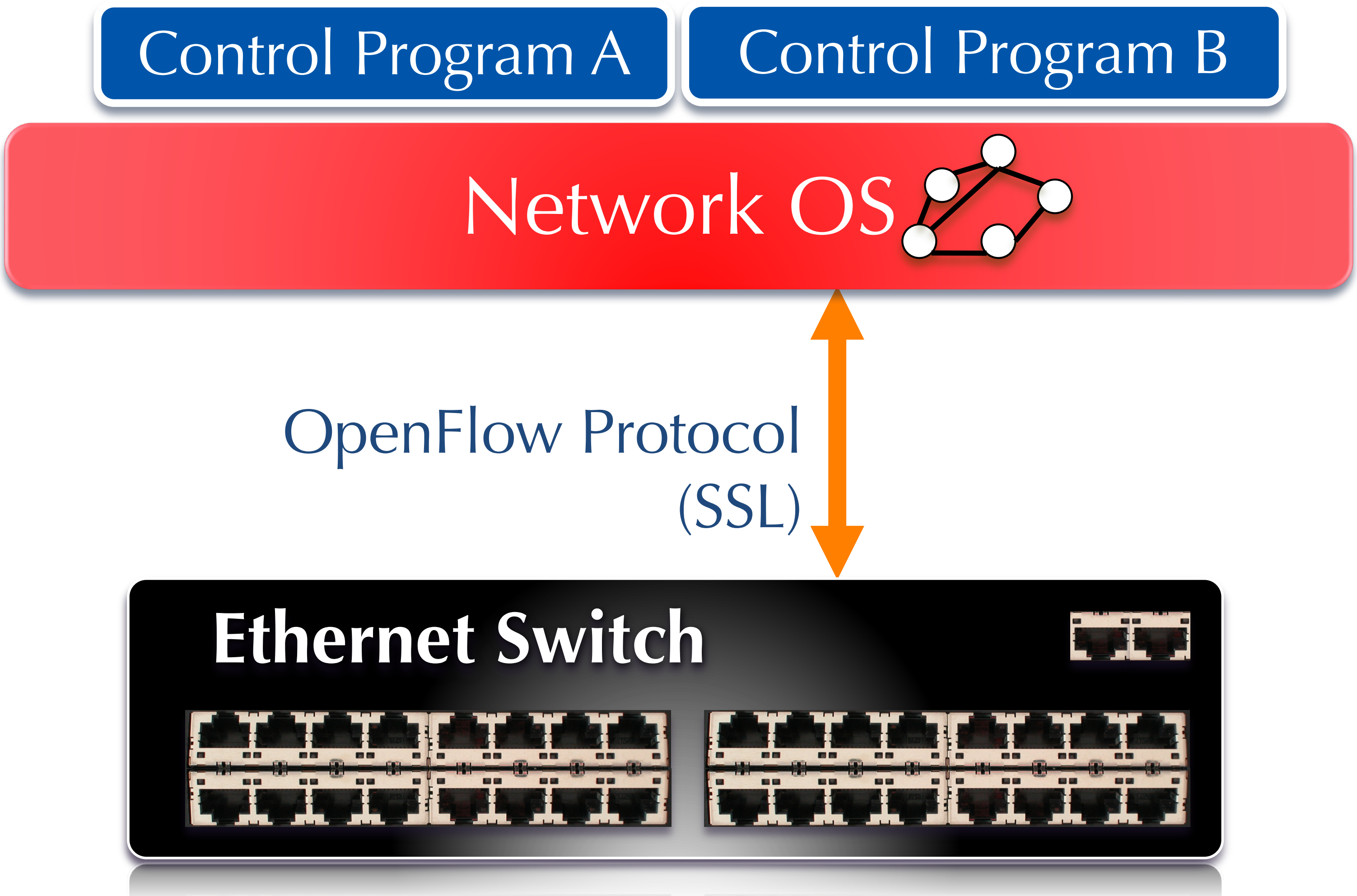
Traditional Switch



Software Defined Network (SDN)



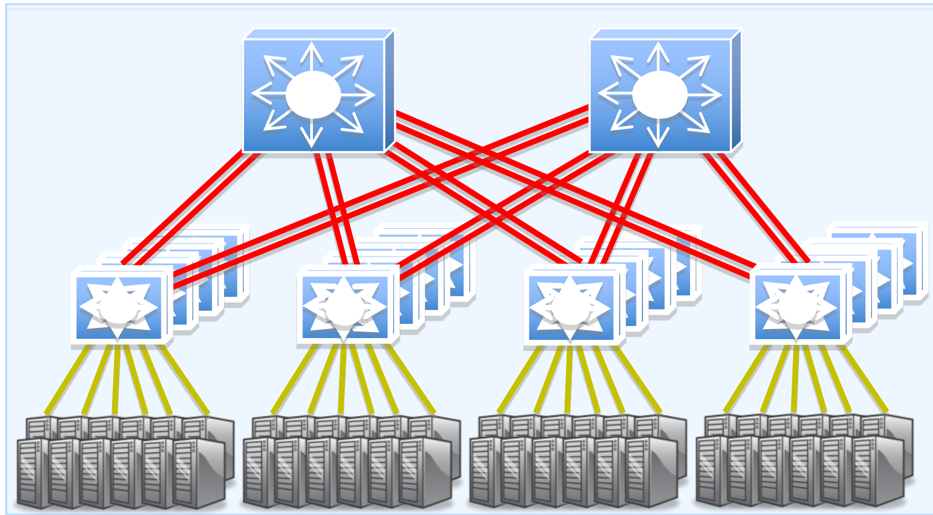
OpenFlow Switch



Consequences

- More innovation in network services
  - Owners, operators, 3rd party developers, researchers can improve the network
  - E.g. energy management, data center management, policy routing, access control, denial of service, mobility
- Lower barrier to entry for competition
  - Healthier marketplace, new players
- Lower cost
  - Infrastructure
  - Management

Example: New Data Center



Cost

200,000 servers

Fanout of 20 a 10,000 switches

$5k commercial switch a $50M

$1k custom-built switch a $10M

Savings in 10 data centers = $400M

Control

1. Optimize for features needed
2. Customize for services & apps
3. Quickly improve and innovate

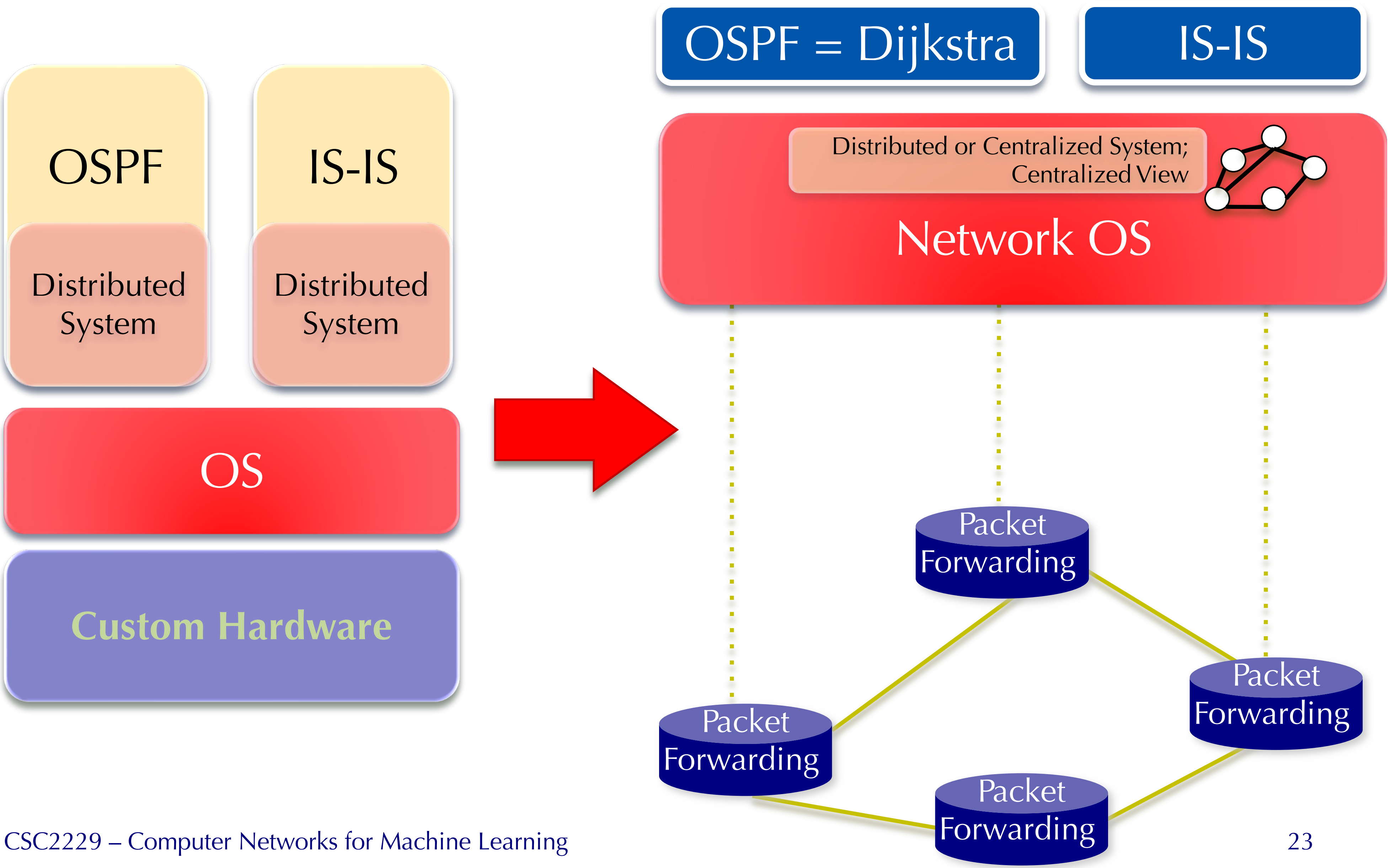
SDN can significantly reduce the CapEx and OpEx of data center networks.

Example: Routing

- OSPF
  - RFC 2328: 245 pages
- Distributed System
  - Builds consistent, up-to-date map of the network: 101 pages

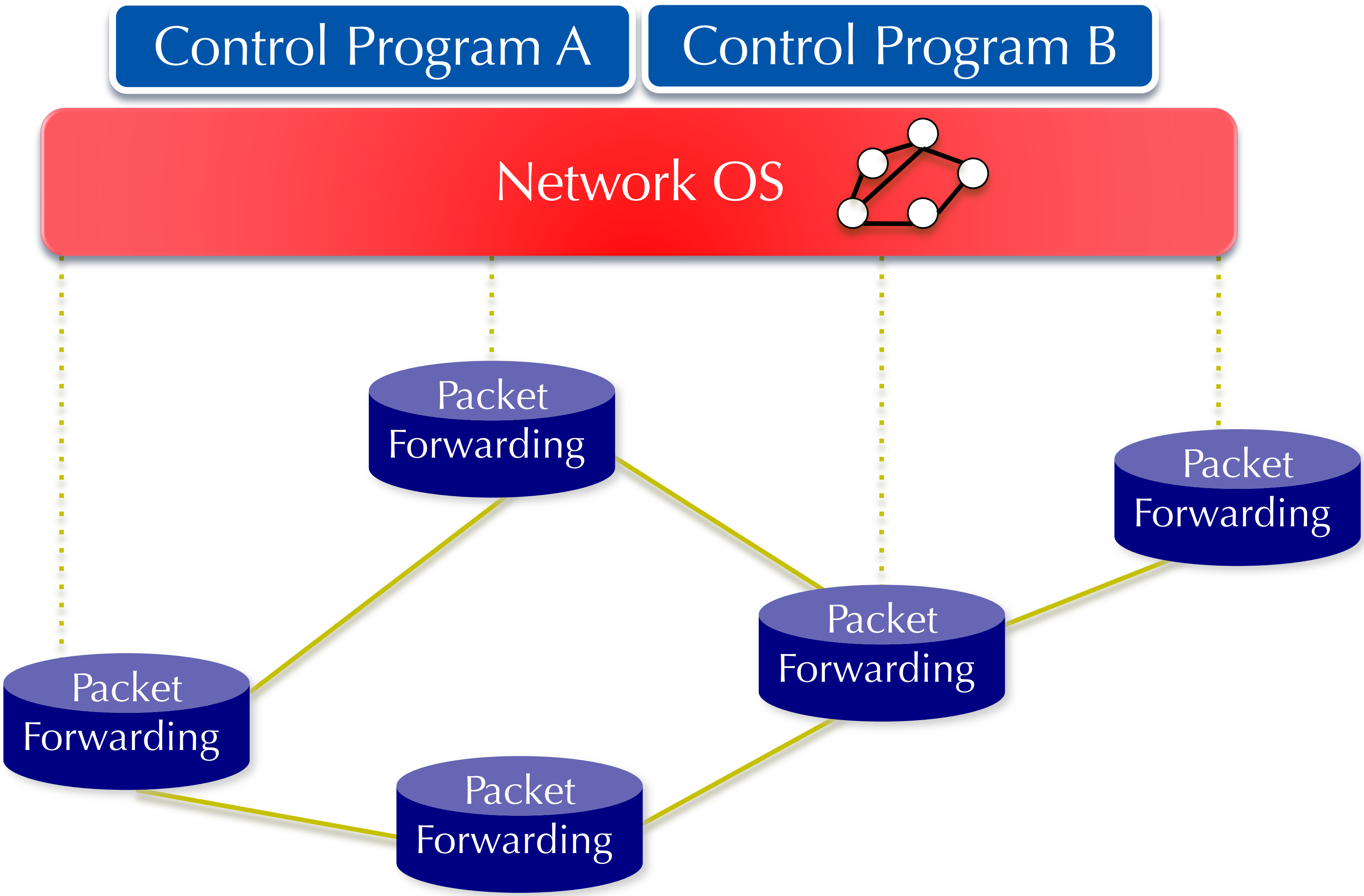
- Dijkstra's Algorithm
  - Operates on map: 4 pages

Example: Routing



Back to the story …

Software Defined Network (SDN)

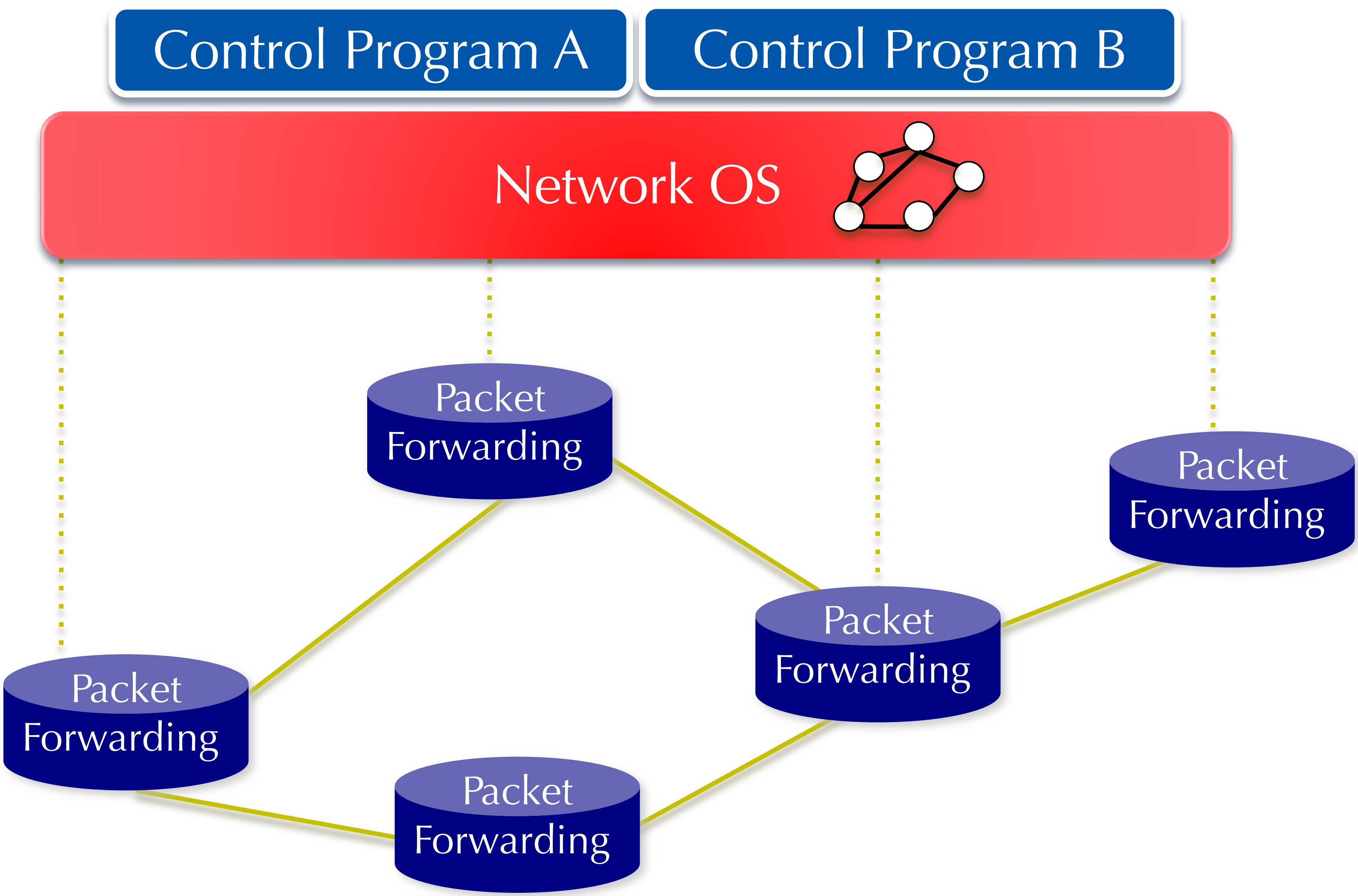


Network OS

- **Network OS**: distributed system that creates a consistent, up-to-date network view
  - Runs on servers (controllers) in the network
  - NOX, ONIX, HyperFlow, Kandoo, Floodlight, Trema, Beacon, Maestro, Beehive, OpenDayLight, ONOS, … + more

- Uses forwarding abstraction to:
  - Get state information from forwarding elements
  - Give control directives to forwarding elements

Software Defined Network (SDN)



Control Program

- Control program operates on view of network
  - Input: global network view (graph/database)
  - Output: configuration of each network device

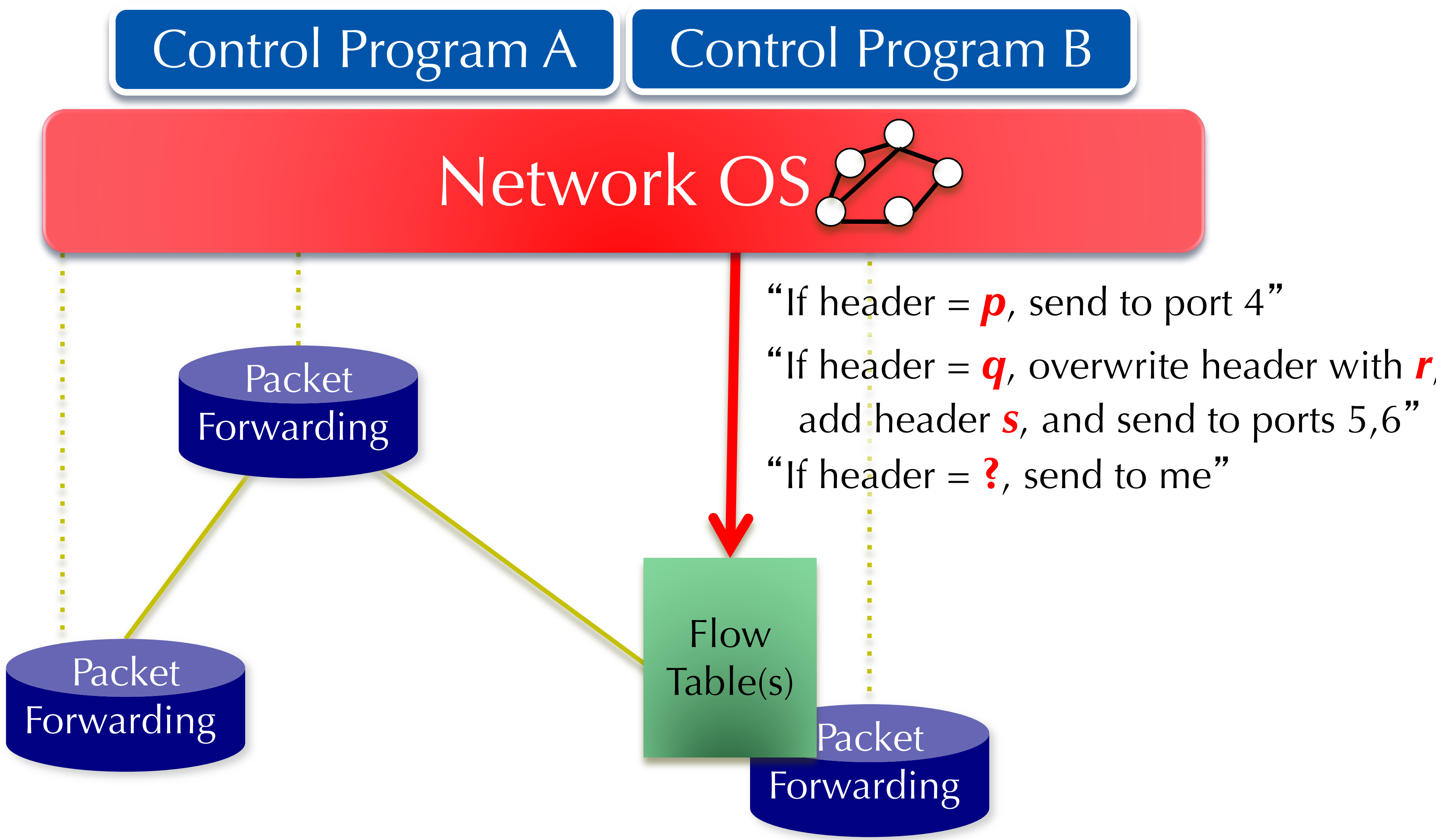
- Control program is not necessarily a distributed system
  - Ideally, the abstraction hides details of distributed state
  - Lots of practical challenges though.

OpenFlow

- OpenFlow
  - Started as open standard to run experimental protocols in production networks
    - API between the forwarding elements and the network OS
  - Started in Stanford, later Open Network Foundation (ONF)
    - Various companies (Cisco, Juniper, HP, NEC, …)

- Later, many similar (sometimes proprietary) interfaces used by various companies

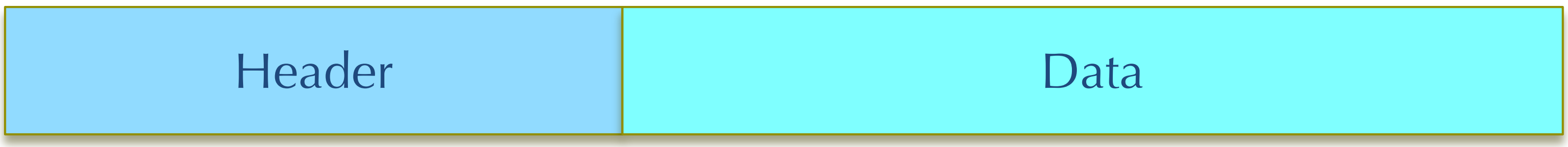
OpenFlow Rules



Plumbing 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 Rules – Cont'd

- Exploit the flow table in switches, routers, and chipsets

Flow 1.	Rule (exact & wildcard)	Action	Statistics
Flow 2.	Rule (exact & wildcard)	Action	Statistics
Flow 3.	Rule (exact & wildcard)	Action	Statistics
	....		
Flow N.	Rule (exact & wildcard)	Default Action	Statistics

Flow Table Entry

- OpenFlow Protocol Version 1.0

Rule	Action	Stats

Packet + byte counters

1. Forward packet to port(s)
2. Encapsulate and forward to controller
3. Drop packet
4. Send to normal processing pipeline

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

+ mask what fields to match

Examples

Switching

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	00:1f:..	*	*	*	*	*	*	*	port6

Flow Switching

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
port3	00:2e..	00:1f..	0800	vlan1	1.2.3.4	5.6.7.8	4	17264	80	port6

Firewall

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Forward
*	*	*	*	*	*	*	*	*	22	drop

Examples

Routing

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	*	*	*	*	5.6.7.8	*	*	*	port6

VLAN

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	*	*	vlan1	*	*	*	*	*	port6, port7, port9

Data Plane Functionalities

- So far, we have focused on *control plane*
  - It distinguishes SDN from traditional networks
  - Source of many (*perceived*) challenges …
  - … and opportunities

- What about the data plane?
  - Which features should be provided in the data plane?

- What defines the boundary between control and data planes?

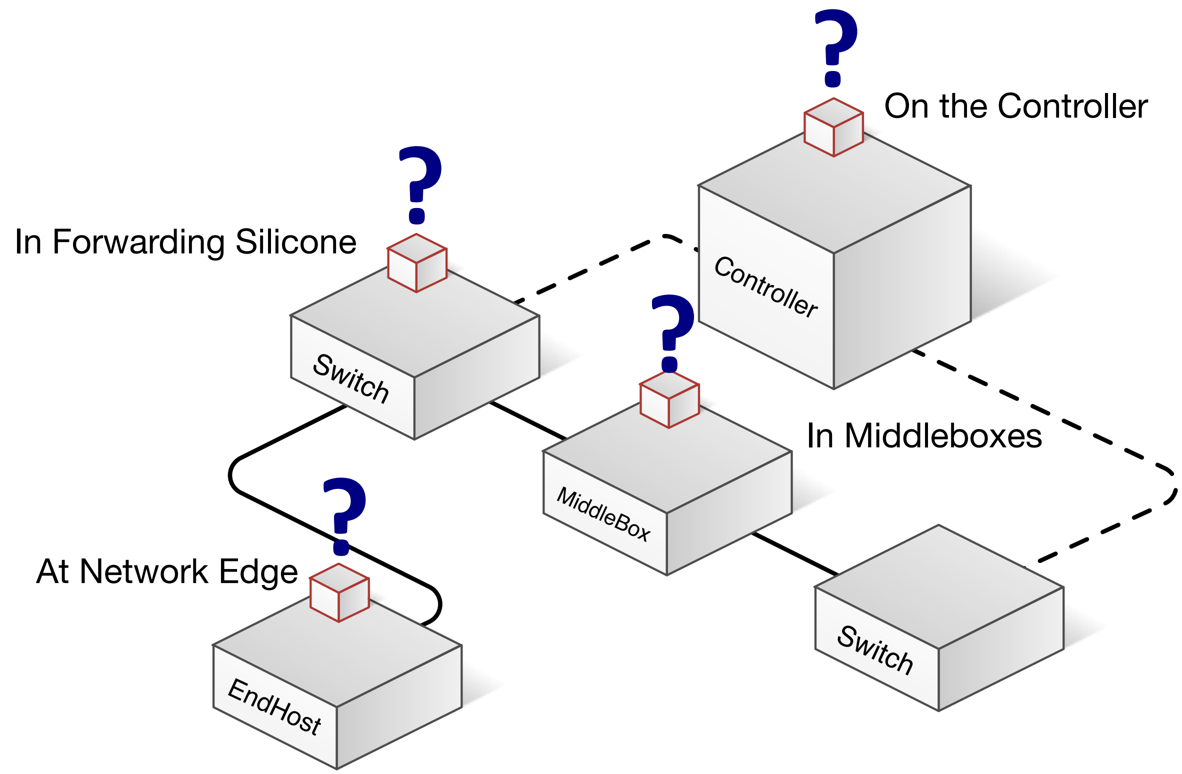
OpenFlow and Data Plane

- Historically, OpenFlow defined data plane's role as *forwarding*.
  - Other functions are left to middleboxes and edge nodes in this model.

- **Question.** Why only forwarding?
  - Other data path functionalities exist in today's routers/switches too.
  - What distinguishes forwarding from other functions?

Adding New Functionalities

- Consider a new functionality we want to implement in a software-defined network.
- Where is the best place to add that function?
  - Controller?
  - Switches?
  - Middleboxes?
  - Endhosts?



Adding New Functionalities

- What metrics/criteria do we have to decide where new functionalities should be implemented?

- **Example:** Elephant flow detection
  - **Data plane:** change forwarding elements (e.g., DevoFlow)
  - **Control plane:** push functionality close to the path (e.g., Kandoo)

- What does the development process look like?
  - Changing ASIC expensive and time-consuming
  - Changing software fast

Alternative View

- Decouple based on development cycle
  - Fast: mostly software
  - Slow: mostly hardware

- Development process
  - Start in software
  - As function matures, move towards hardware

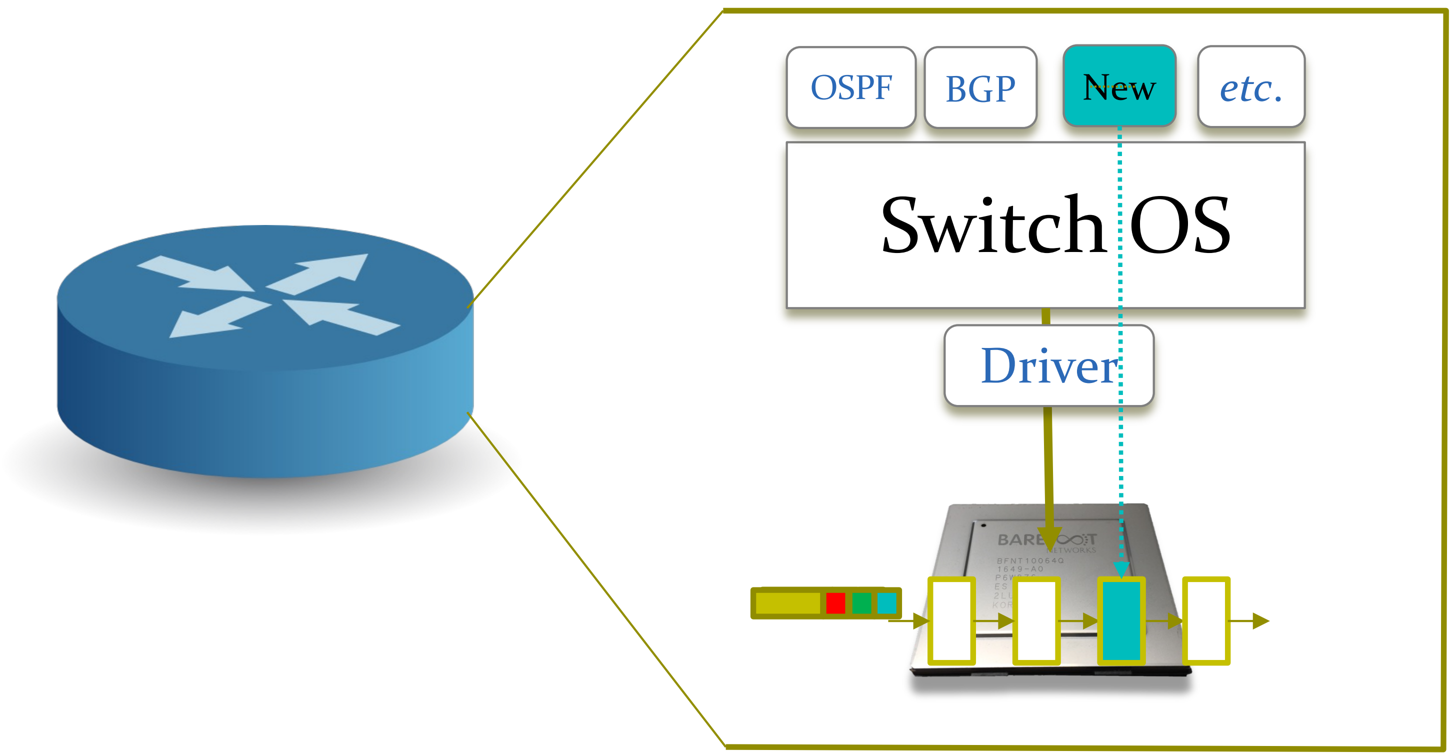
- Not a new idea
  - Model has been used for a long-time in industry.

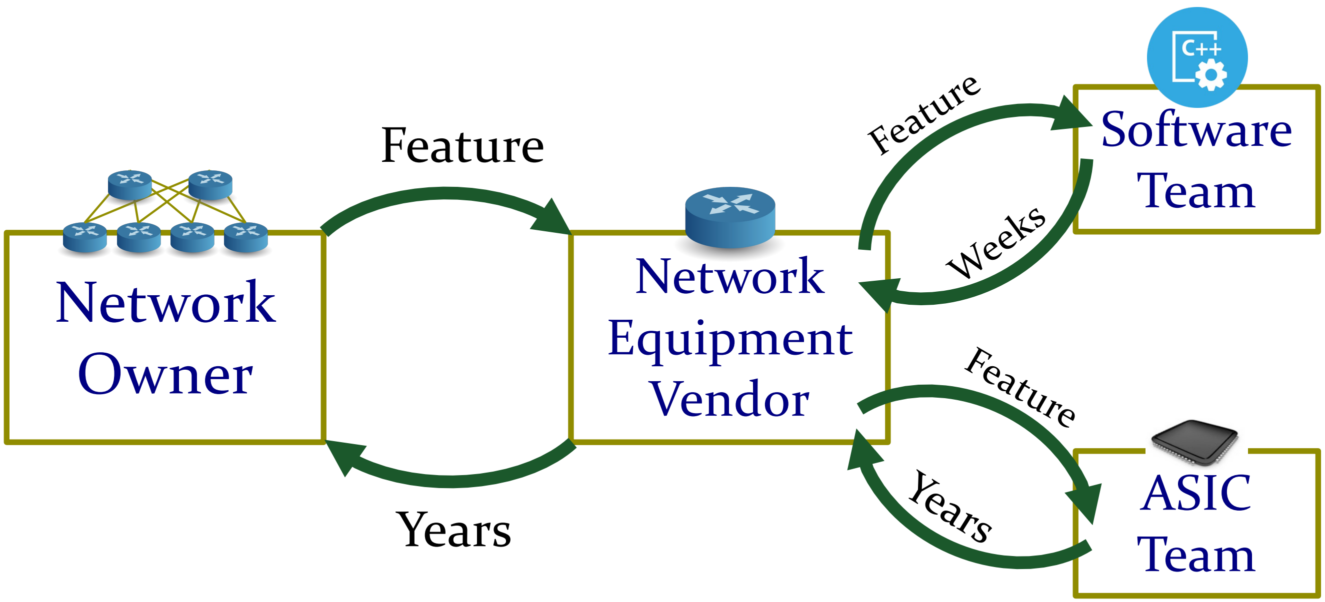
Changing the Control Plane

I can tailor my network to meet my needs …

1. Quickly deploy new protocols.
2. Monitor precisely what my forwarding plane is doing.
3. Fold expensive middlebox functions into the network, for free.
4. Try out beautiful new ideas. Tailor my network to meet my needs.
5. Differentiate. Now I own my intellectual property.

But wait a minute…


OSPF
BGP
New
etc.
Switch OS
Driver

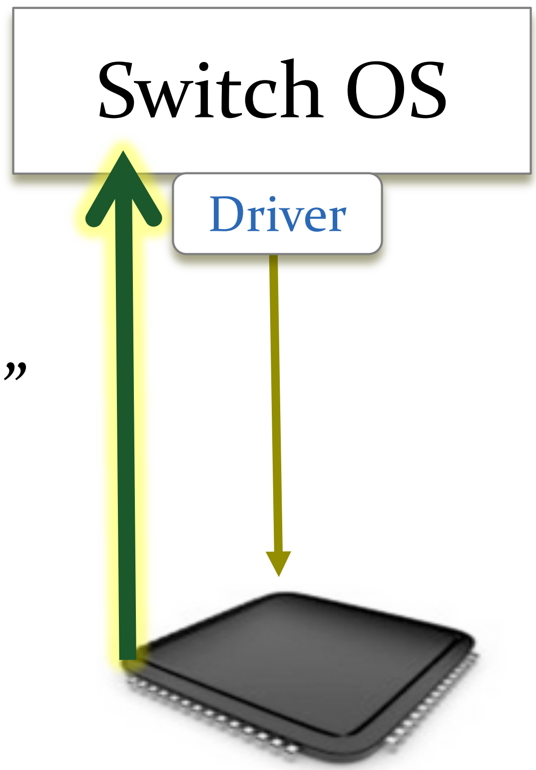

Network
Owner
Feature
Years
Network
Equipment
Vendor
Feature
Weeks
Software
Team
C++
Feature
Years
ASIC
Team

When you need a new feature…

1. Equipment vendor can't just send you a software upgrade
2. New forwarding features take years to develop
3. By then, you've figured out a kludge to work around it
4. Your network gets more complicated, more brittle
5. Eventually, when the upgrade is available, it either
   - No longer solves your problem, or
   - You need a fork-lift upgrade, at huge expense.

Network Systems Are Built "Bottom-up"

"This is how I process packets …"



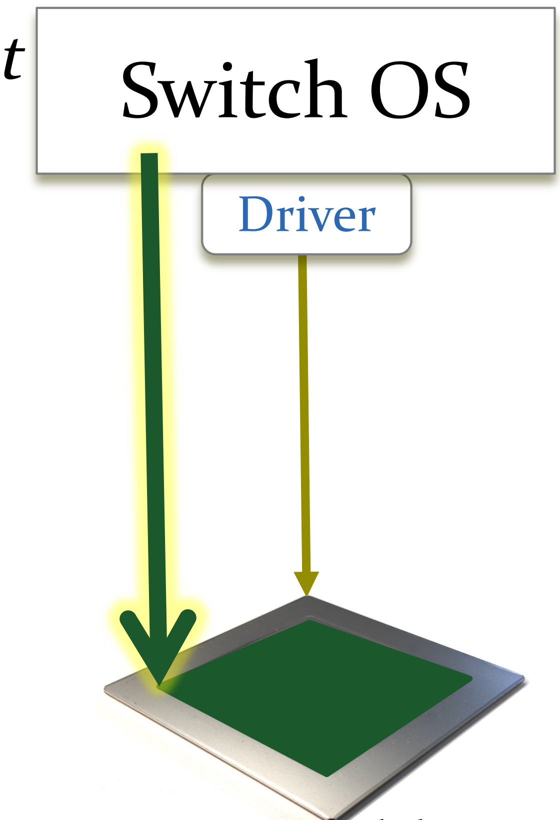
Fixed-function switch

"Top-Down" Network Programming

"This is precisely how you must process packets"

```
table int_table {
  reads {
    ip.protocol;
  }
  actions {
    export_queue_latency;
  }
}
```

```
action export_queue_latency (sw_id) {
  add_header(int_header);
  modify_field(int_header.kind, TCP_OPTION_INT);
  modify_field(int_header.len, TCP_OPTION_INT_LEN);
  modify_field(int_header.sw_id, sw_id);
  modify_field(int_header.q_latency,
               intrinsic_metadata.deq_timedelta);
  add_to_field(tcp.dataOffset, 2);
  add_to_field(ipv4.totalLen, 8);
  subtract_from_field(ingress_metadata.tcpLength,
                      12);
}
```



Programmable Switch

Domain Specific Processors

Computers

Java

Compiler

CPU

Graphics

OpenCL

Compiler

GPU

Signal Processing

Matlab

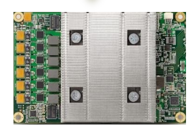
Compiler

DSP

Machine Learning

TensorFlow

Compiler



TPU

>>>

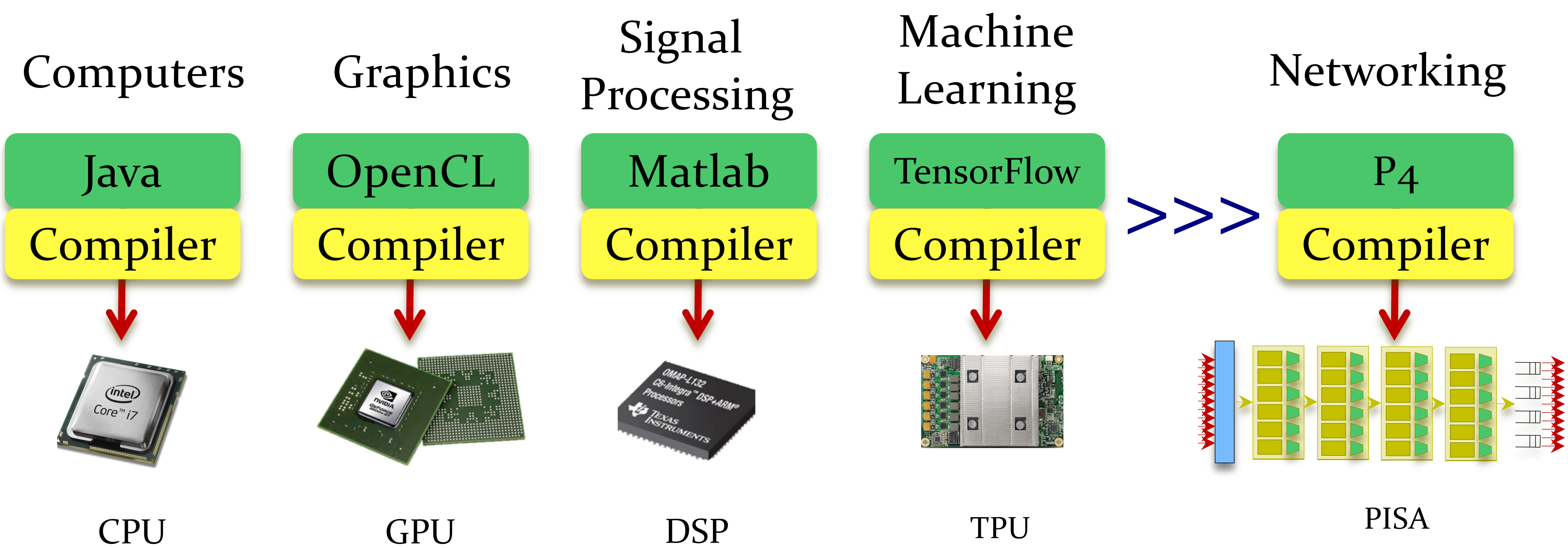
Networking

Language

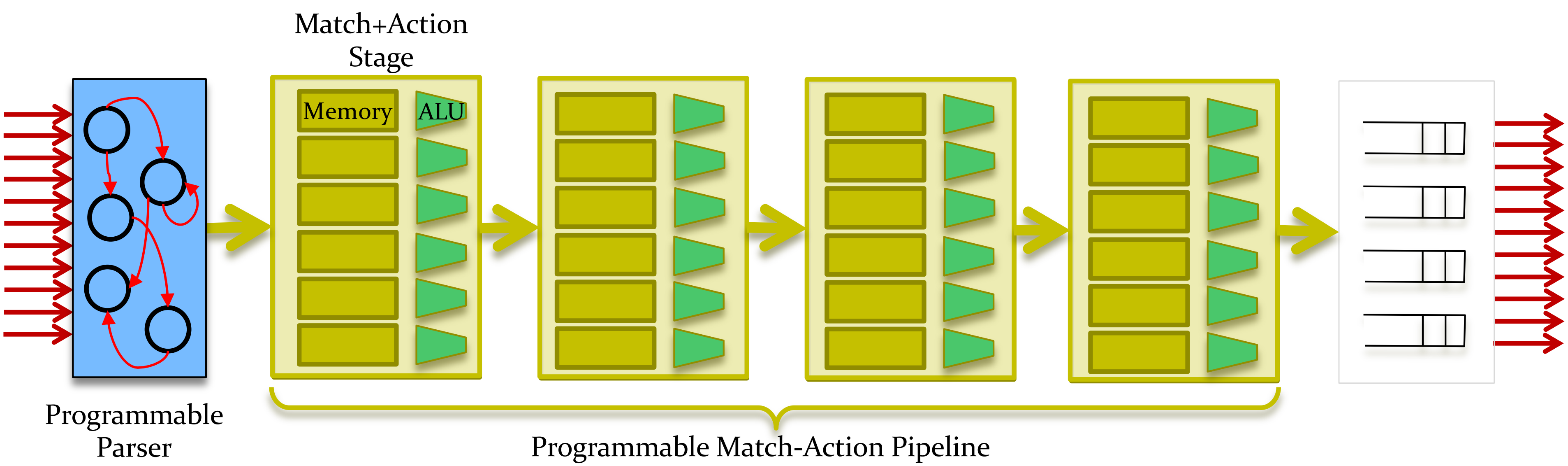
Compiler

?

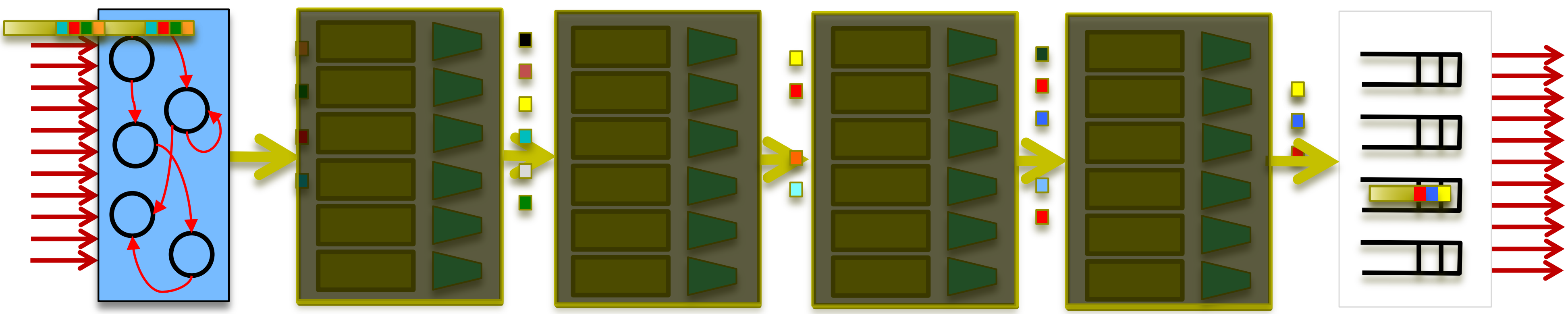
Domain Specific Processors



PISA: Protocol Independent Switch Architecture



PISA: Protocol Independent Switch Architecture



Example P4 Program

Parser Program

```
parser parse_ethernet {
   extract(ethernet);
   return switch(ethernet.ethertype) {
      0x8100 : parse_vlan_tag;
      0x0800 : parse_ipv4;
      0x8847 : parse_mpls;
      default: ingress;
}
```

Header and Data Declarations

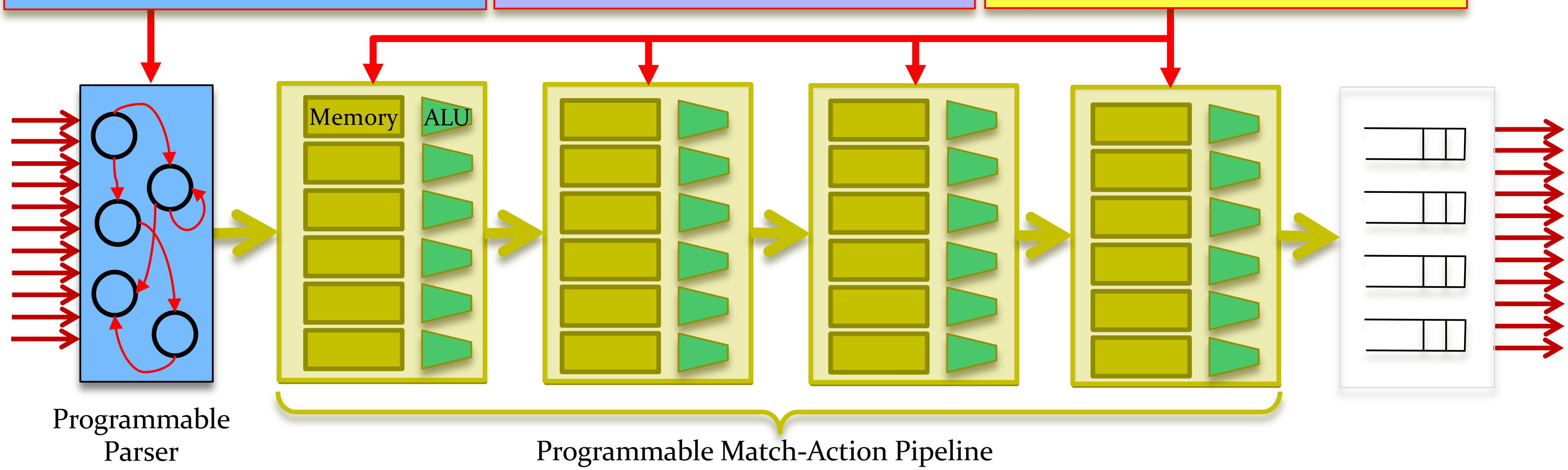
```
header_type  ethernet_t     { … }
header_type  l2_metadata_t { … }

header      ethernet_t     ethernet;
header      vlan_tag_t     vlan_tag[2];
metadata    l2_metadata_t  l2_meta;
```

Tables and Control Flow

```
table port_table { … }

control ingress {
    apply(port_table);
    if (l2_meta.vlan_tags == 0) {
        process_assign_vlan();
    }
}
```



Example: Barefoot Tofino 6.5Tb/s Switch

Same power.
Same cost.

Forwarding defined in software (P4).
Programs <u>always</u> run at line-rate.

Network Programmability: Consequences

1 **Reducing Complexity**

Reducing Complexity

switch.p4

Switch OS

IPv4 and IPv6 routing
- Unicast Routing
  - Routed Ports & SVI
  - VRF
- Unicast RPF
  - Strict and Loose
- ~~Multicast~~
  - ~~PIM-SM/DM & PIM-Bidir~~

Ethernet switching
- ~~VLAN Flooding~~
- MAC Learning & Aging
- STP state
- ~~VLAN Translation~~

Load balancing
- ~~LAG~~
- ECMP & WCMP
- Resilient Hashing
- ~~Flowlet Switching~~

Fast Failover
– LAG & ECMP

Tunneling
- IPv4 and IPv6 Routing & Switching
  - ~~IP in IP (6in4, 4in4)~~
  - VXLAN, NVGRE, GENEVE & GRE
  - ~~Segment Routing, ILA~~

~~**MPLS**~~
- ~~LER and LSR~~
- ~~IPv4/v6 routing (L3VPN)~~
- ~~L2 switching (EoMPLS, VPLS)~~
- ~~MPLS over UDP/GRE~~

ACL
- MAC ACL, IPv4/v6 ACL, RACL
- ~~QoS ACL, System ACL, PBR~~
- Port Range lookups in ACLs

QOS
- QoS Classification & marking
- ~~Drop profiles/WRED~~
- ~~RoCE v2 & FCoE~~
- CoPP (Control plane policing)

~~**NAT and L4 Load Balancing**~~

Security Features
- ~~Storm Control, IP Source Guard~~

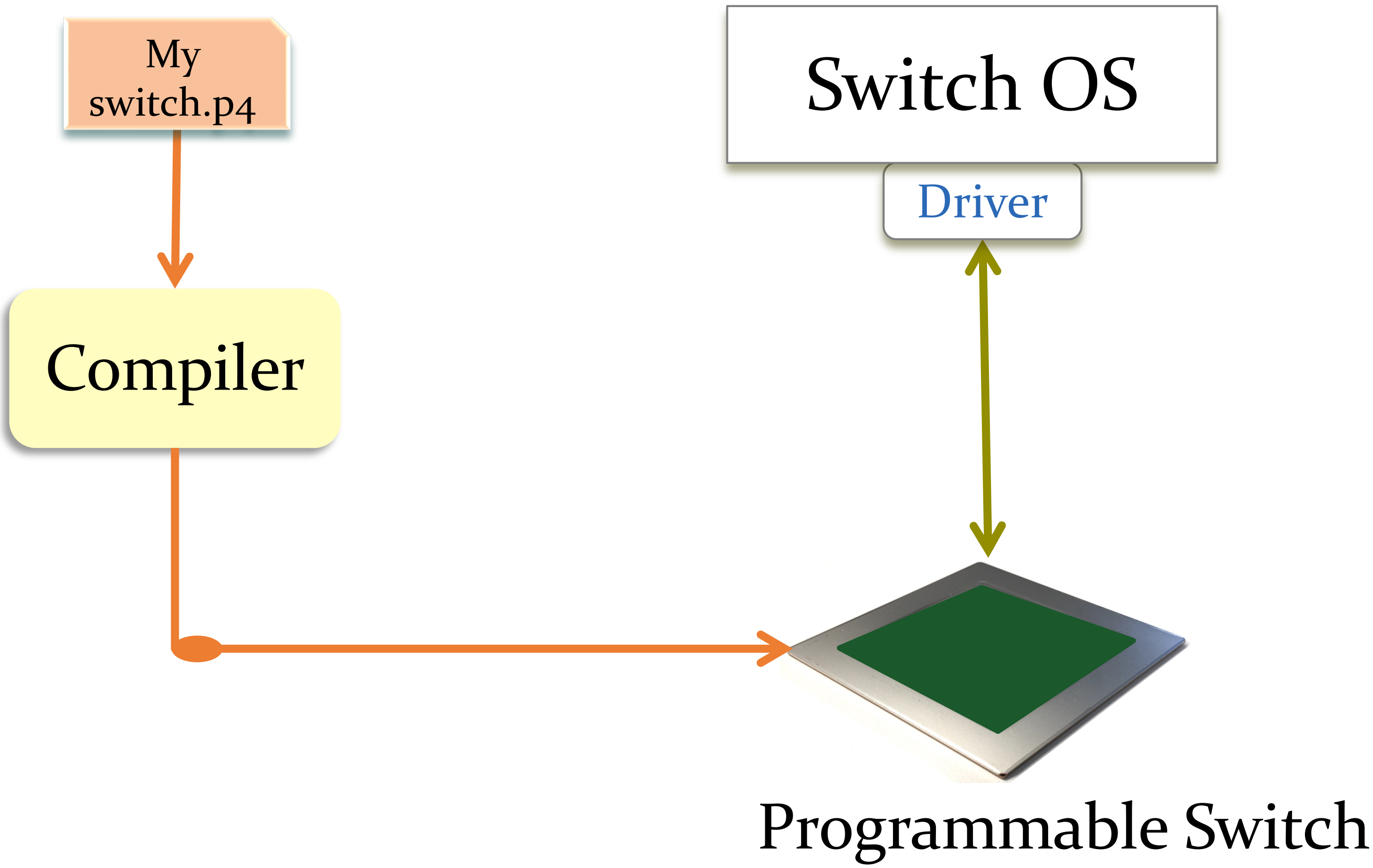
Monitoring & Telemetry
- ~~Ingress Mirroring and Egress Mirroring~~
- Negative Mirroring
- ~~Sflow~~
- INT

Counters
- Route Table Entry Counters
- ~~VLAN/Bridge Domain Counters~~
- Port/Interface Counters

Protocol Offload
- BFD, OAM

Multi-chip Fabric Support
- ~~Forwarding, QOS~~

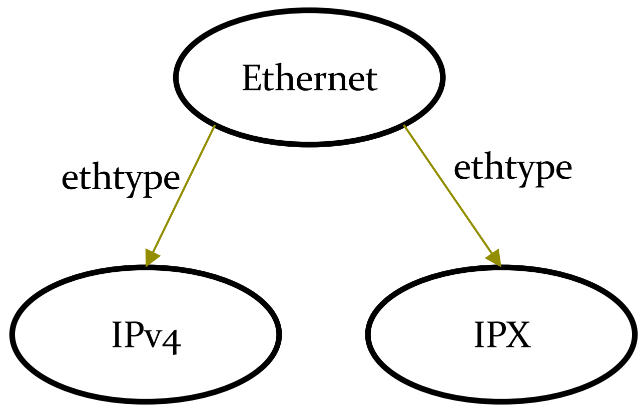
Reducing Complexity



Network Programmability: Consequences

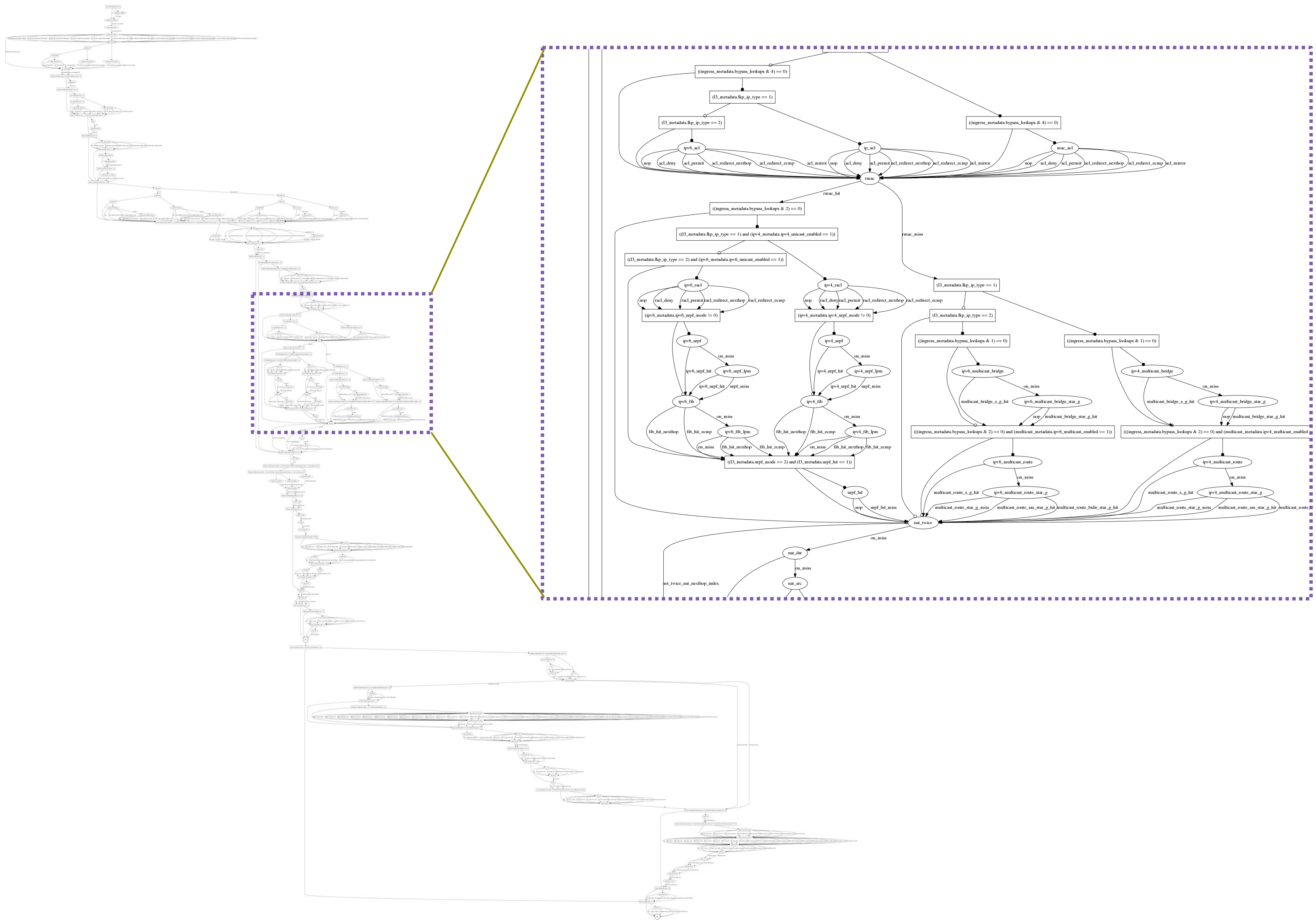
2 **Adding New Features**

Protocol Complexity 30 Years Ago



Programmable DCN Switch

Switch.p4



Adding Features: Some Examples

1. New encapsulations and tunnels
2. New ways to tag packets for special treatment
3. New approaches to routing: *e.g. source routing in MSDCs*
4. New approaches to congestion control
5. New ways to process packets: *e.g. processing ticker-symbols*

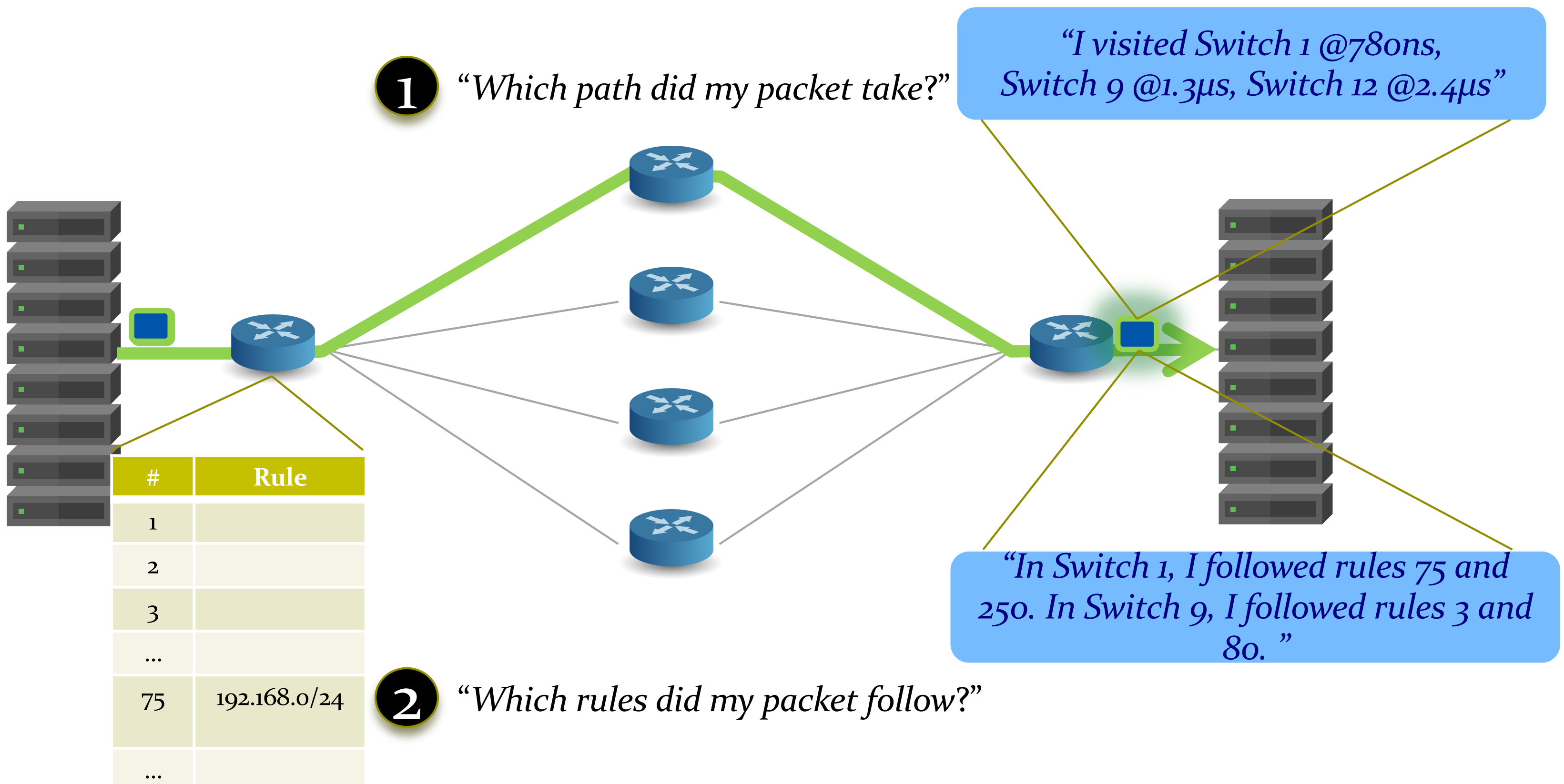
New applications: Some Examples

1. Layer-4 Load Balancer[1]
   - Replace 100 servers or 10 dedicated boxes with one programmable switch
   - Track and maintain mapping for 5-10 million http flows
2. Fast stateless firewall
   - Add/delete and track 100s of thousands of new connections per second
3. Cache for Key-value store[2]
   - Memcache in-network cache for 100 servers
   - 1-2 billion operations per second

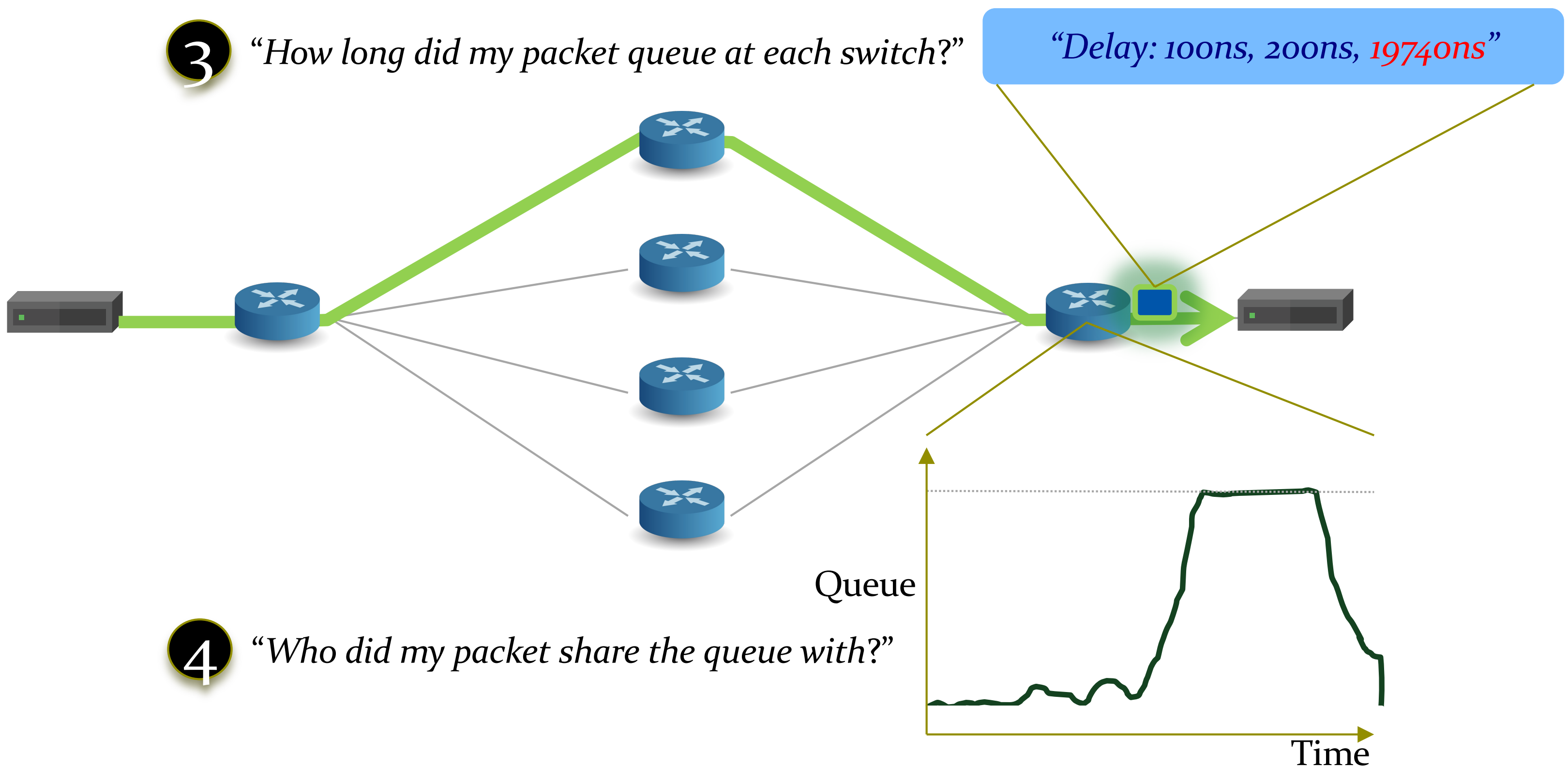
[1] "SilkRoad: Making Stateful Layer-4 Load Balancing Fast and Cheap Using Switching ASICs." Rui Miao et al. SIGCOMM 2017.
[2] "NetCache: Balancing Key-Value Stores with Fast In-Network Caching", Xin Jin et al. SOSP 2017

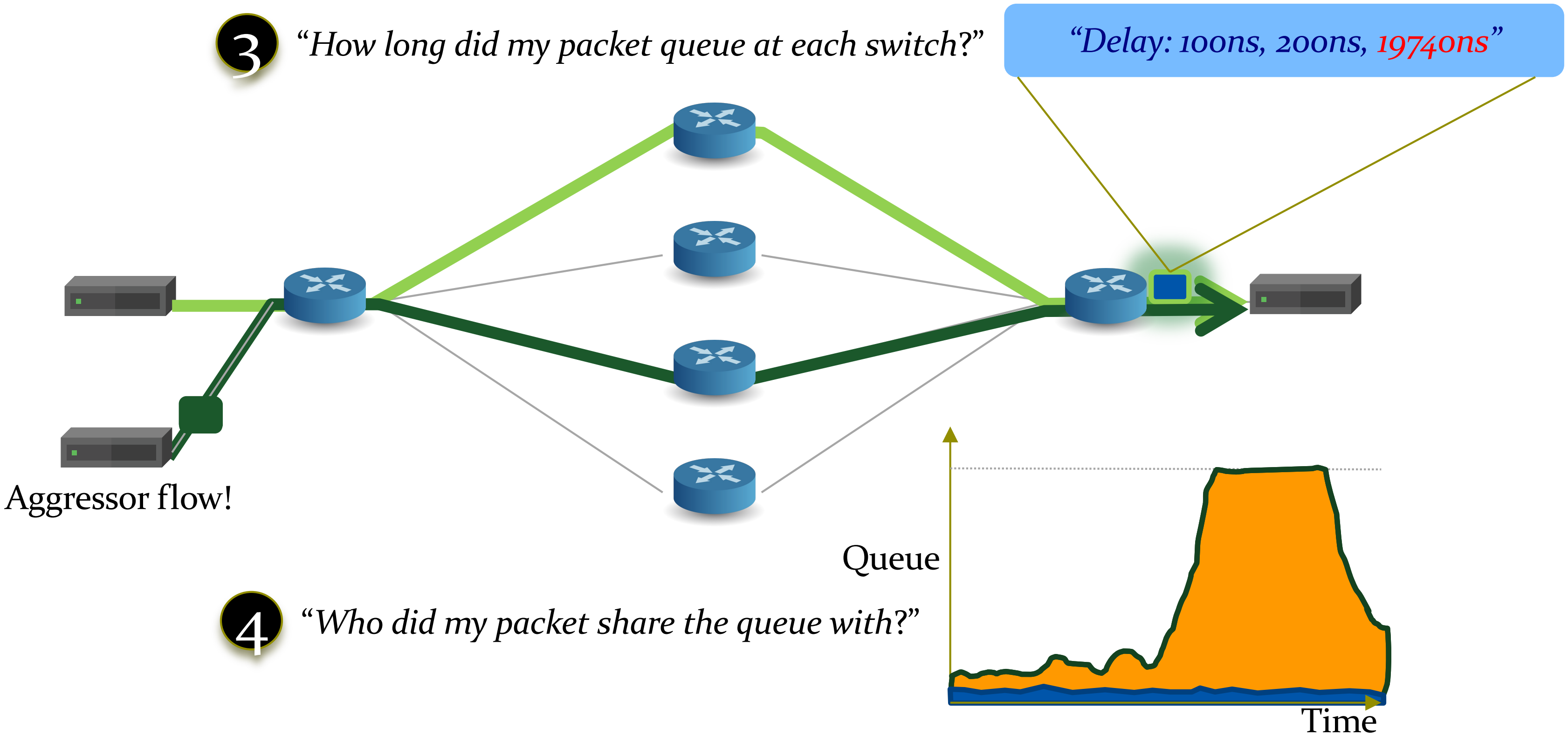
Network Programmability: Consequences

3 **Network Telemetry**



#	Rule
1	
2	
3	
…	
75	192.168.0/24
…	


3
"How long did my packet queue at each switch?"
"Delay: 100ns, 200ns, 19740ns"
Queue
Time
4
"Who did my packet share the queue with?"

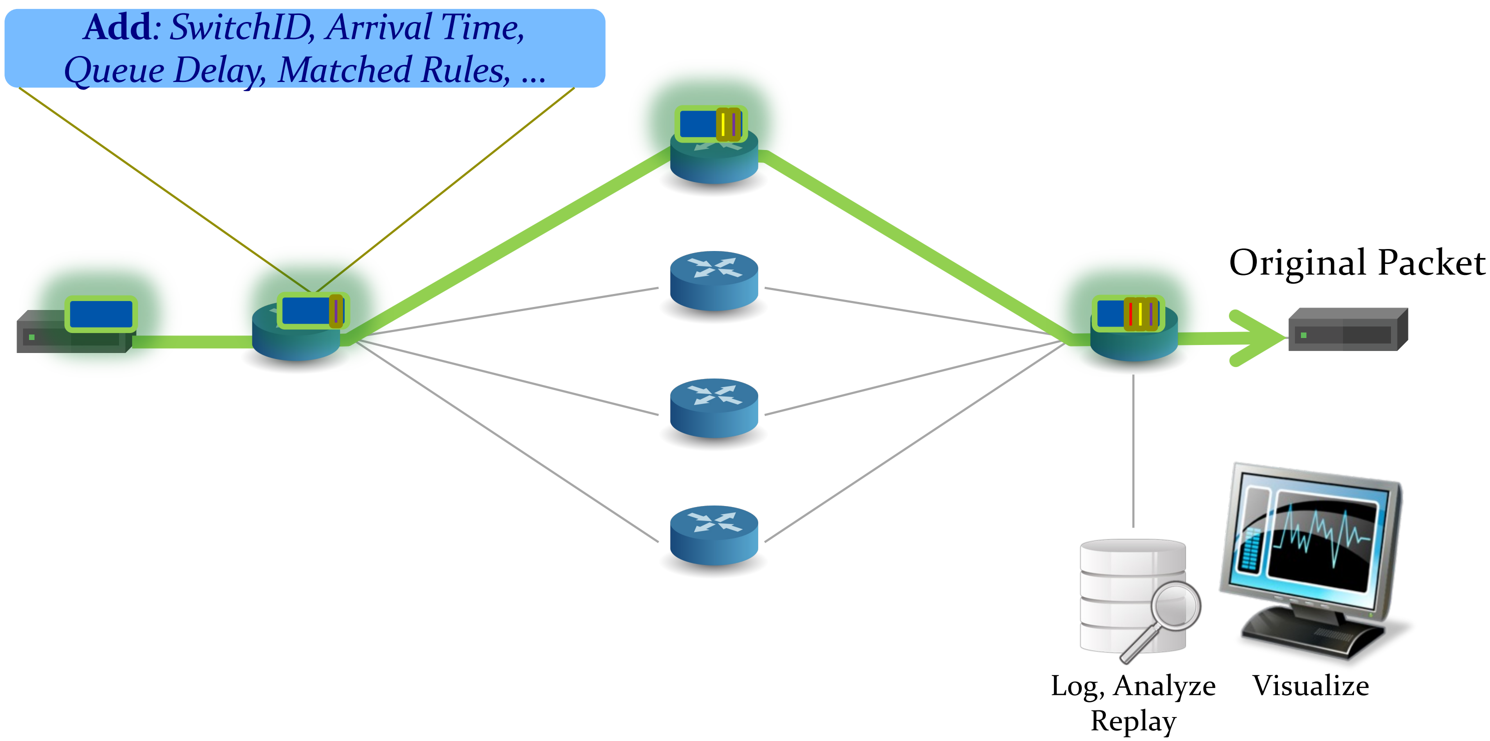

3
"How long did my packet queue at each switch?"
"Delay: 100ns, 200ns, 19740ns"
Aggressor flow!
Queue
Time
4
"Who did my packet share the queue with?"

We'd like the network to answer these questions

1. *"Which path did my packet take?"*
2. *"Which rules did my packet follow?"*
3. *"How long did it queue at each switch?"*
4. *"Who did it share the queues with?"*

A PISA device programmed using P4 can answer all four questions at line rate, for the first time. Without generating additional packets.

INT: Inband Network Telemetry

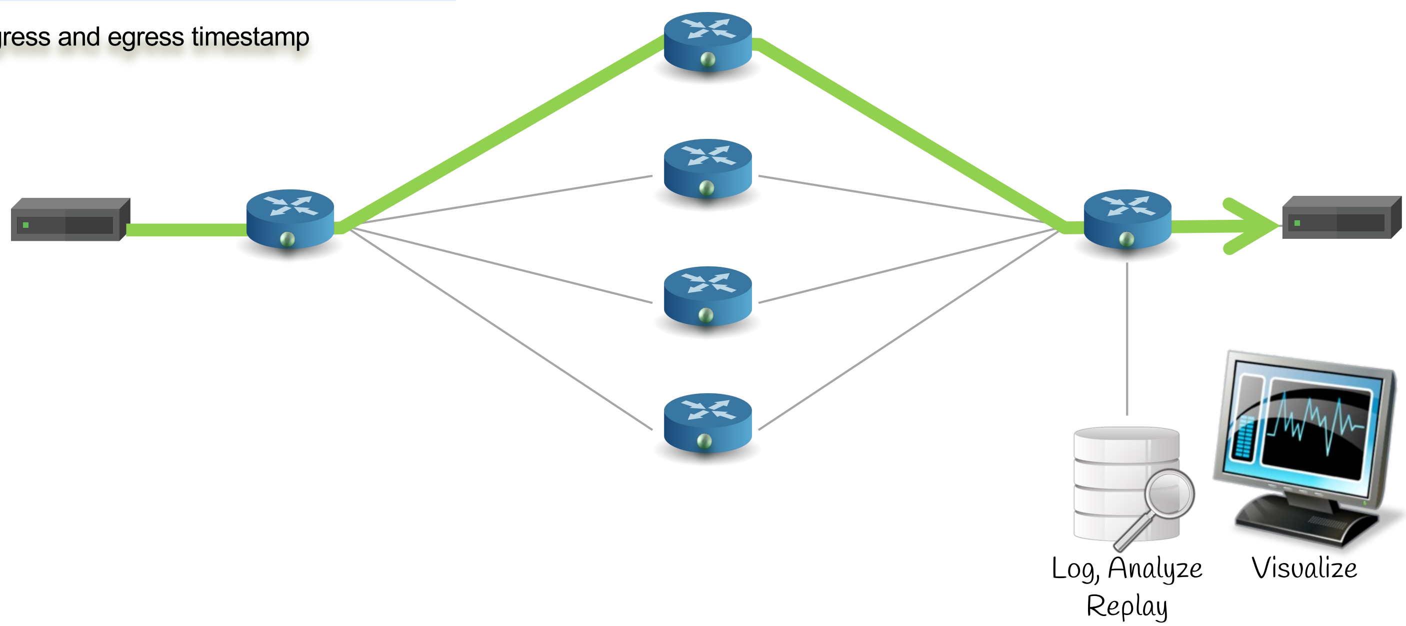


```
/* INT: add switch id */
action int_set_header_0() {
  add_header(int_switch_id_header);
  modify_field(int_switch_id_header.switch_id,
               global_config_metadata.switch_id);
}

/* INT: add ingress timestamp */
action int_set_header_1() {
  add_header(int_ingress_tstamp_header);
  modify_field(int_ingress_tstamp_header.ingress_tstamp,
i2e_metadata.ingress_tstamp);
}

/* INT: add egress timestamp */
action int_set_header_2() {
  add_header(int_egress_tstamp_header);
  modify_field(int_egress_tstamp_header.egress_tstamp,
                eg_intr_md_from_parser_aux.egress_global_tstamp);
}
```

P4 code snippet: Insert switch ID, ingress and egress timestamp



Network Programmability: Consequences

1. Reducing Complexity
2. Adding New Features
3. Network Telemetry

Final Comments

- Network programmability:
  - Significantly powerful, makes change feasible
  - Innovation in “control plane” and “data path”

- Academia vs. industry
  - Programmability allows development and testing of new ideas.
  - Has had a major impact in academic research
    - Creating proof-of-concept solutions in days/weeks
  - Opens doors for innovation and high impact research
  - Programmable switches in production: limited use cases
    - Programmable core vs. edge: which one do you think is more useful?