SDNFV - Softwarization and Virtualization
Software Defined Networking

Prof. Chien-Chao Tseng

曾建超教授

Department of Computer Science
National Chiao Tung University
cctseng@cs.nctu.edu.tw

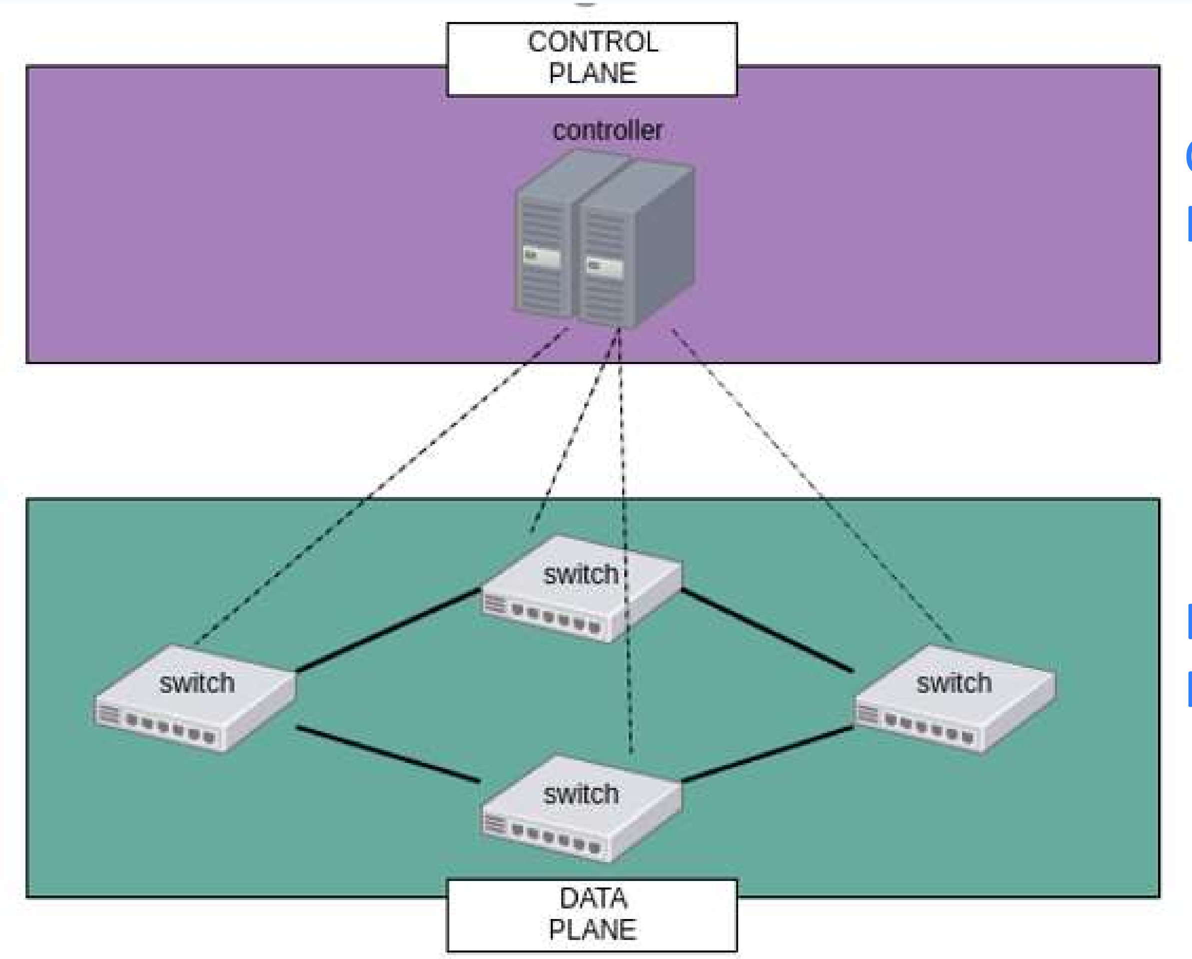
References:

1. Nick McKeown, Stanford University
- IEEE ICC 2018 // Keynote: Programmable Forwarding Planes Are Here To Stay
- https://www.youtube.com/watch?v=8ie0FcsN07U
- CS244 Advanced Topics in Networking- Lecture 7: Programmable Forwarding
2. Open Networking Foundation (ONF)
3. Software Defined Networking: Introduction to SDN & OpenFlow, Prof. James Won-Ki Hong

Software Defined Networks (SDN) and Network Function Virtualization (NFV)

Software Defined Networking (SDN)

- Centralized Programmable Network Control

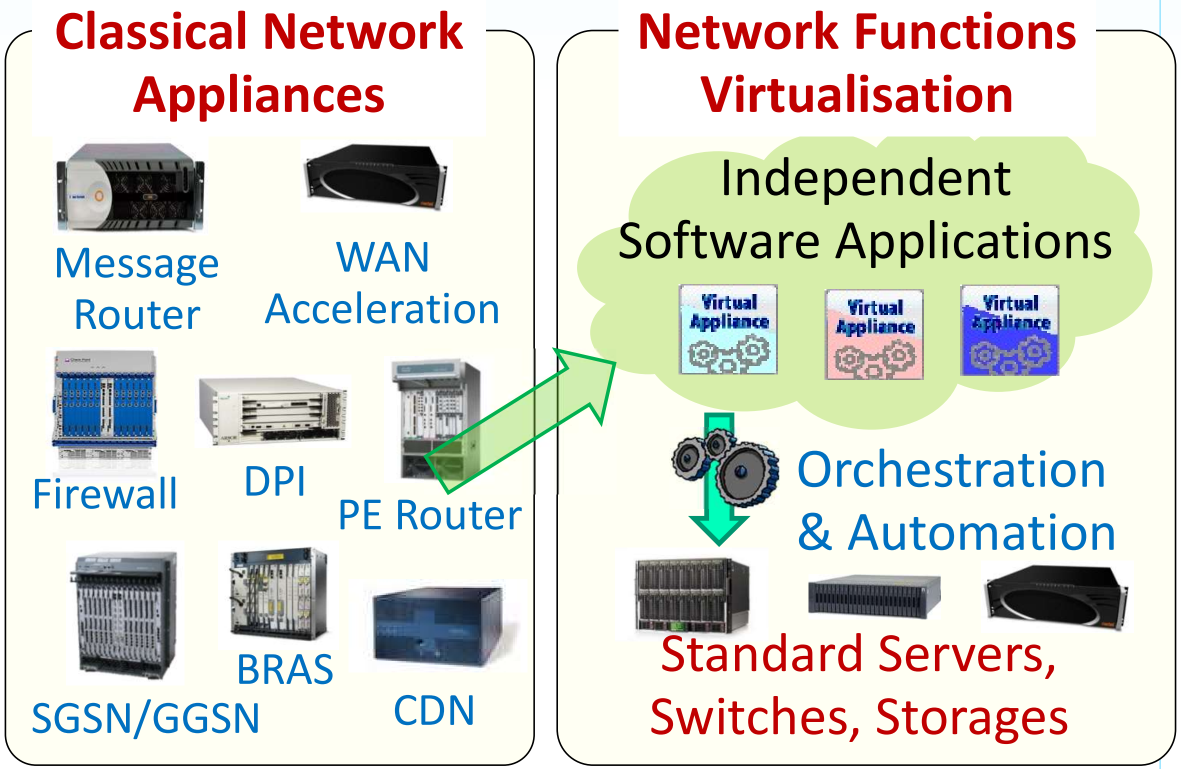


Control
Pane

Data
Plane

Network Functions Virtualisation

- **Network Functions Virtualization (NFV)**
  leveraging standard **IT virtualization** technology to consolidate many network equipment types onto **industry standard** high volume servers, switches and storage
  - which could be located in Datacenters, Network Nodes and End User Premises.



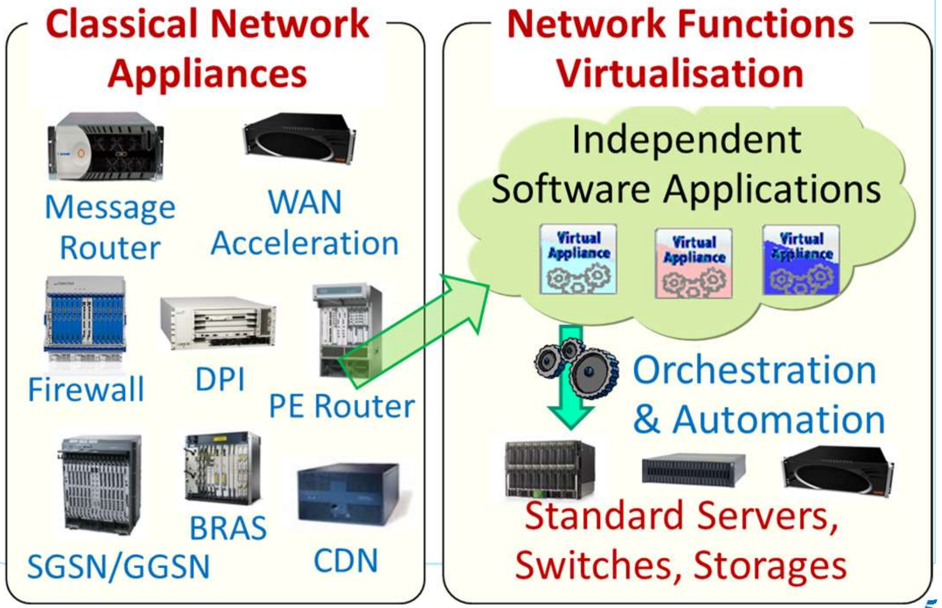
- PE Router: Provider Edge Router,
- DPI: Deep Packet Inspection
- GGSN: Gateway GPRS Support Node,
- SGSN: Serving GPRS Support Node,
- BRAS: Broadband Remote Access Server,
- CDN: Content Delivery Network

Source: Network Functions Virtualisation – Introductory White Paper
http://portal.etsi.org/NFV/NFV_White_Paper.pdf

NFV and SDN

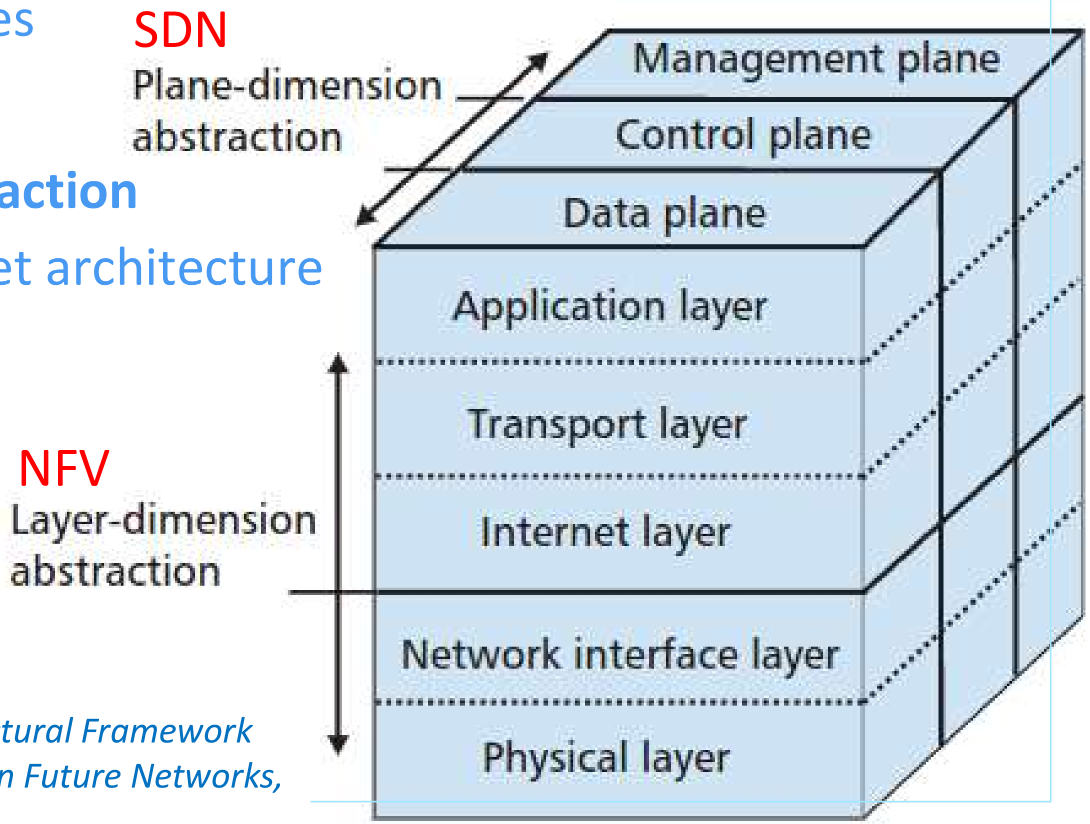
- NFV is applicable to
  - **Data plane packet processing** and **control plane function**
  - in **fixed** and **mobile** network infrastructures.
- **Highly Complementary** to Software Defined Networking (SDN).
  - **Mutually beneficial** but are **not dependent** on each other.
- **Network Functions** can be **virtualized** and **deployed** without an SDN and **vice-versa**.

✓ **Key principles** of both SDN and NFV are based on **abstraction** and **softwarization**



Two-dimensional Model of Layer-Plane Abstraction

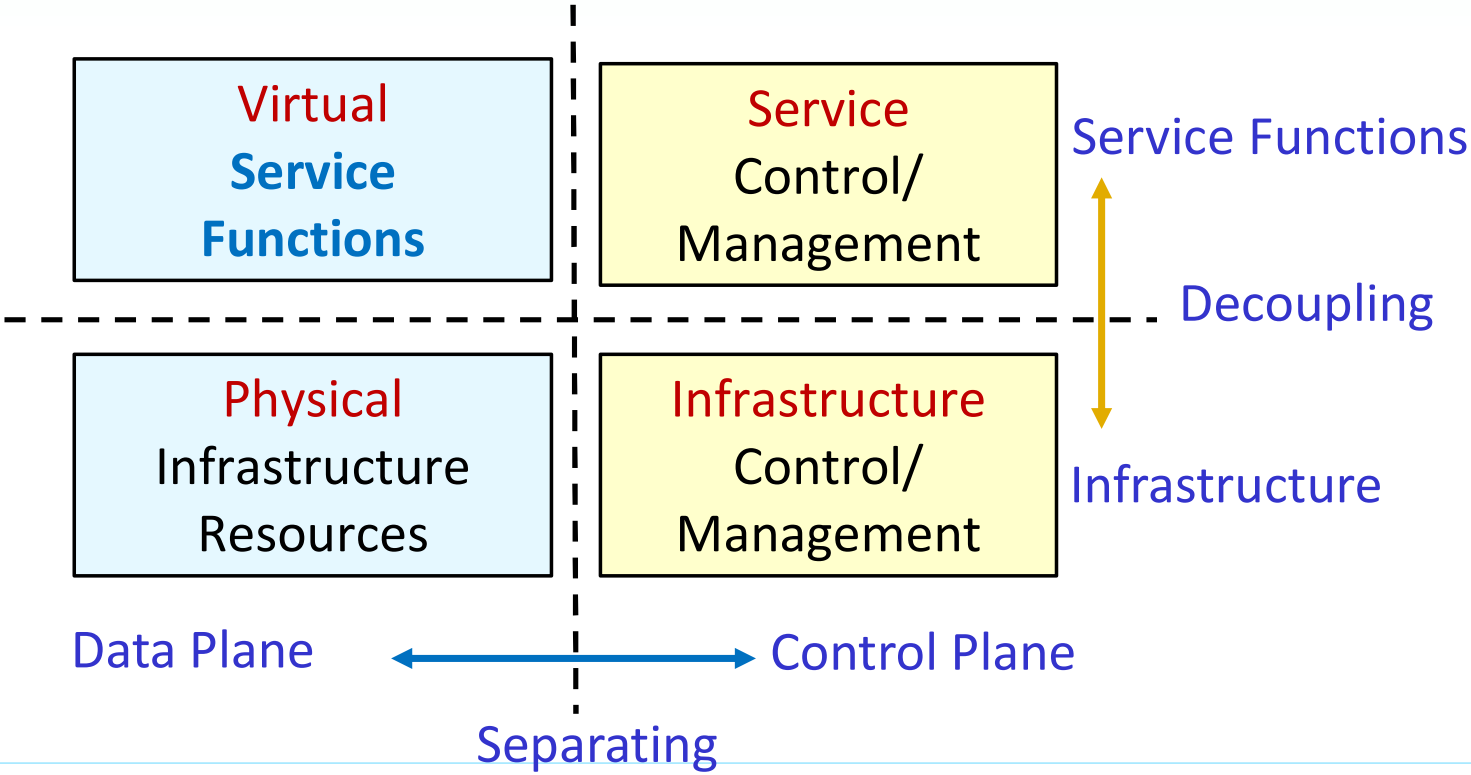
- **SDN: based on Plane-dimension Abstraction**
  - Plane abstraction of traditional circuit-switching-based telecom system
    - Data, Control, Management Planes
- **NFV: based on Layer-dimension Abstraction**
  - Layer abstraction of IP-based Internet architecture
    - TCP/IP Layer Stack
- Two abstraction dimensions are orthogonal
  - in principle are independent



Source: Software-Defined Network Virtualization: An Architectural Framework for Integrating SDN and NFV for Service Provisioning in Future Networks, IEEE Network, 2016

Integration of SDN and NFV

- Unified Network Architecture with components in four quadrants
  - Loosely coupled with abstract interfaces



SDN

Act 1

In which network owners take charge of their **control plane**

The Rise of Whitebox Servers and Switches

Servers

Applications
OS
CPU

Legacy

Applications
Linux
x86

Whitebox

Network switches

Applications
OS
CPU + ASIC

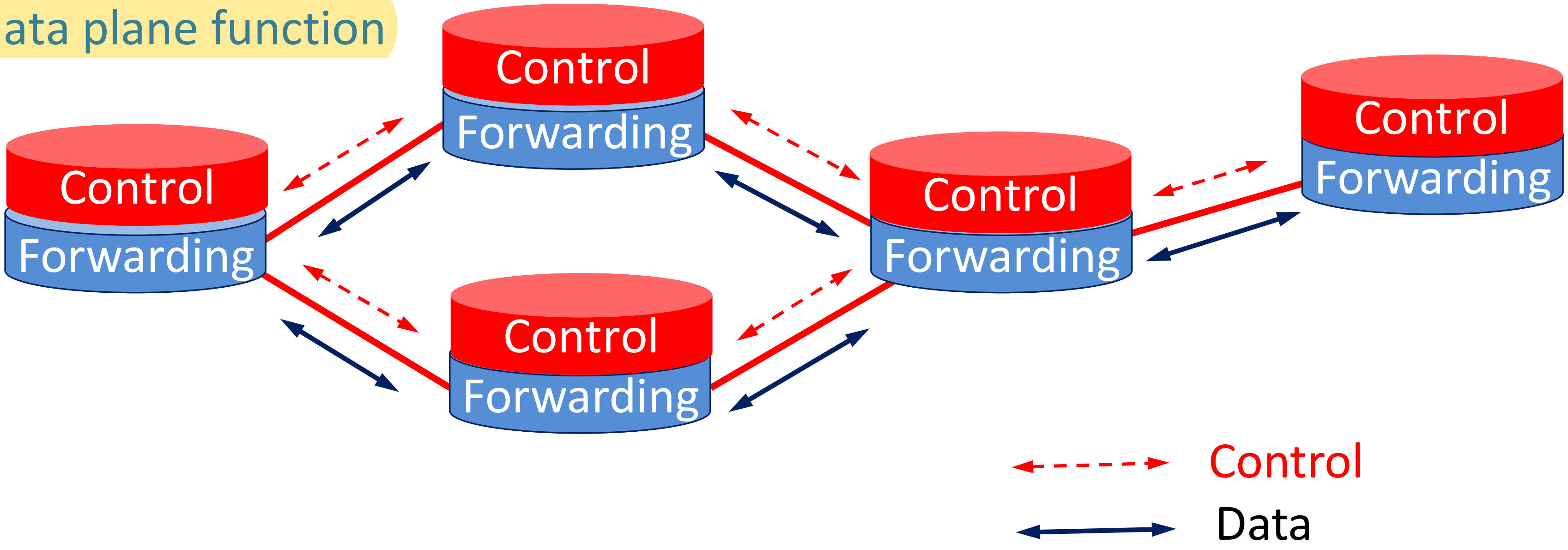
Legacy

Control Programs
Linux
x86 + ASIC

Whitebox

Traditional Networking

- Integrated Control and Data Planes
- Distributed Control
  - Distributed algorithm running between neighbors
  - Vender lock-in
- Fixed data plane function

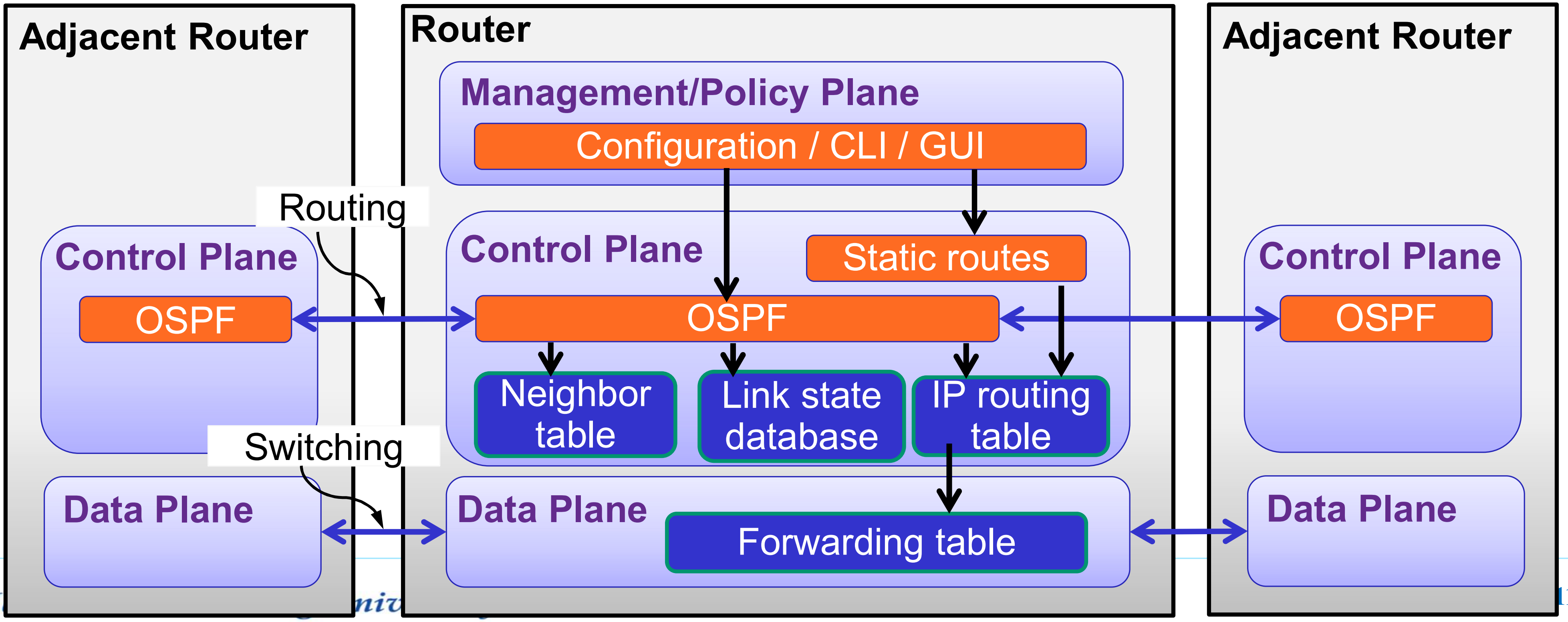


Credited to Prof. Scott Shenker (UC, Berkeley) and Prof.Nick Mckeown (Stanford University)

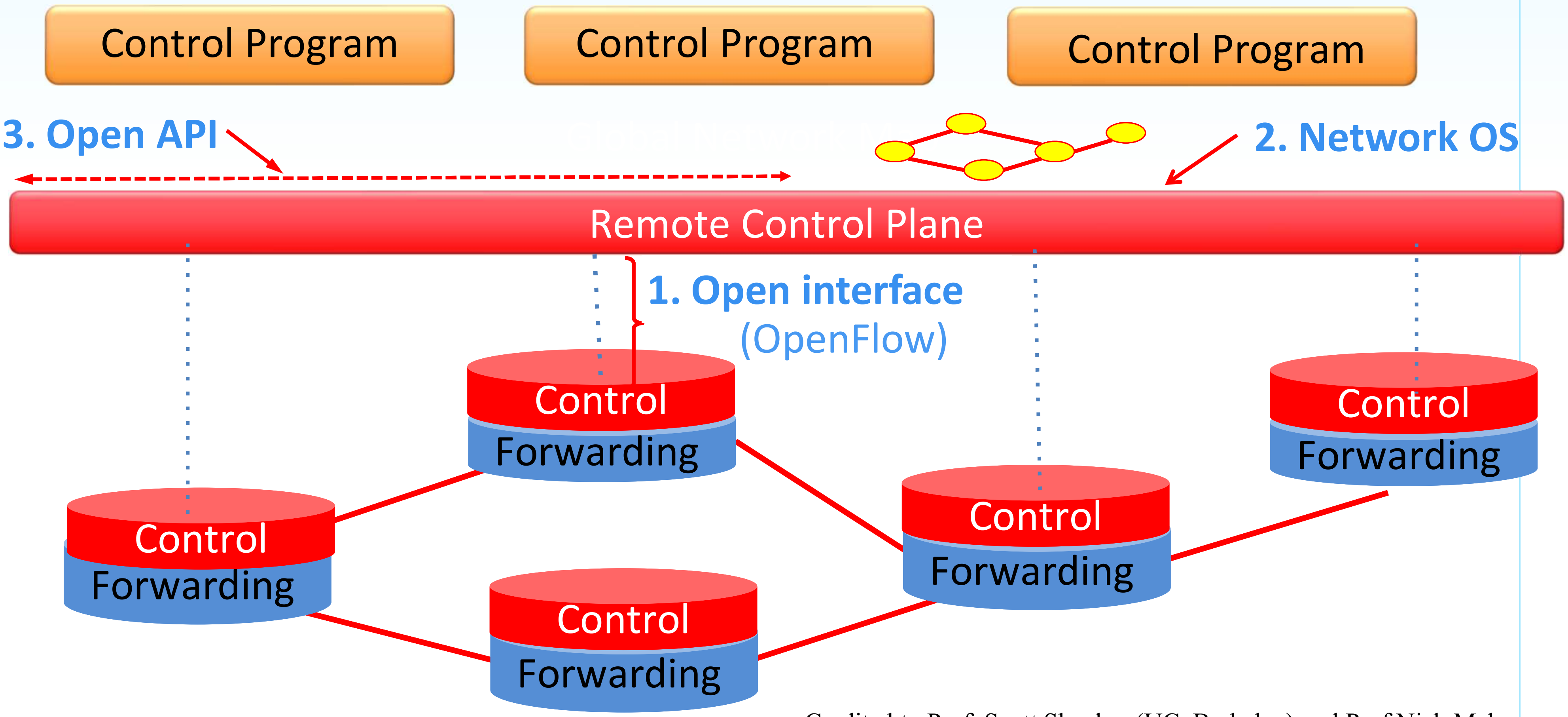
Traditional Network Node

- Router: can be partitioned into three planes
  1. Management plane → Configuration
  2. Control plane → Make decision for the route
  3. Data plane → Data forwarding

Source: James Won-Ki Hong, CSED702Y: Software Defined Networking, POSTECH



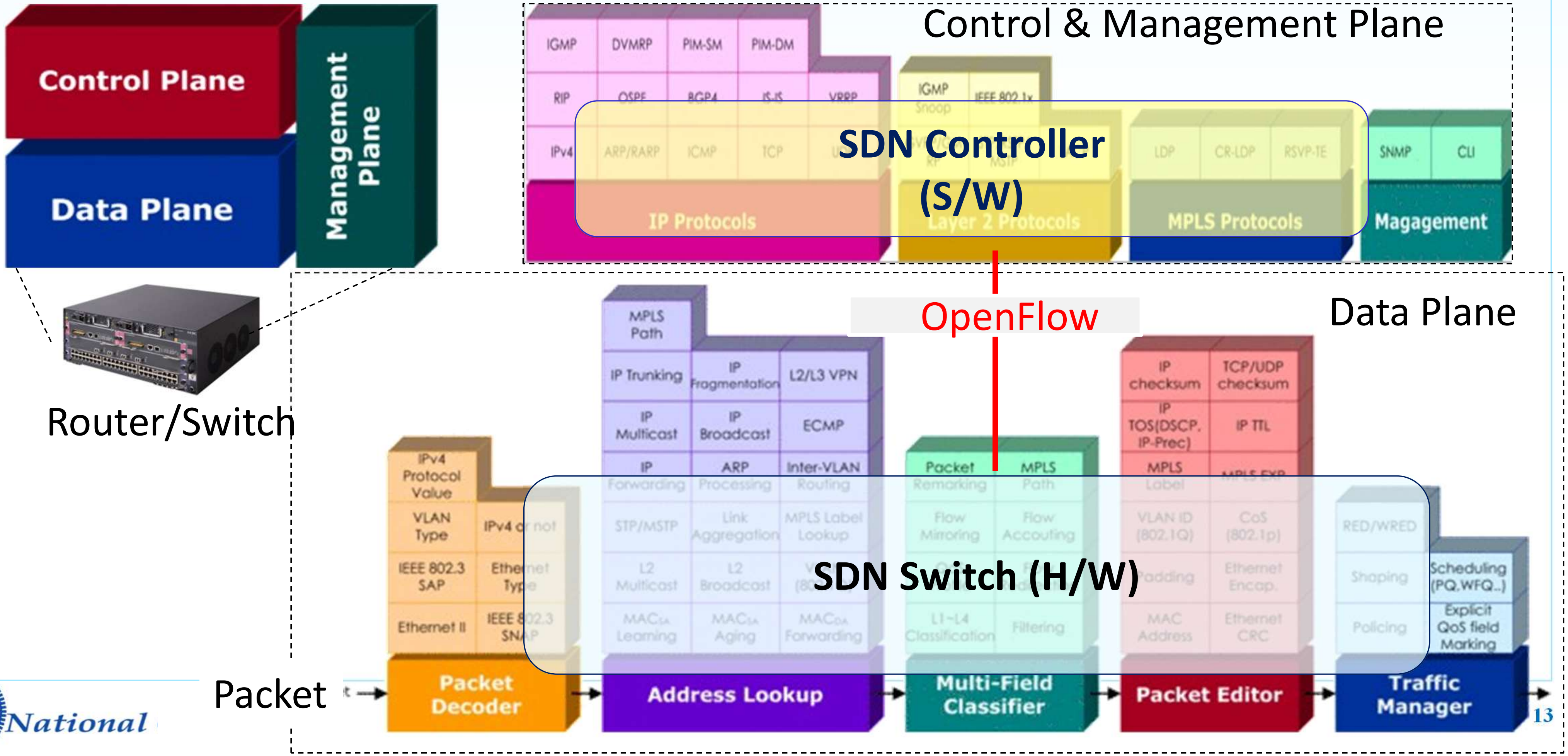
Software Defined Network (SDN)



Credited to Prof. Scott Shenker (UC, Berkeley) and Prof.Nick Mckeown (Stanford University)

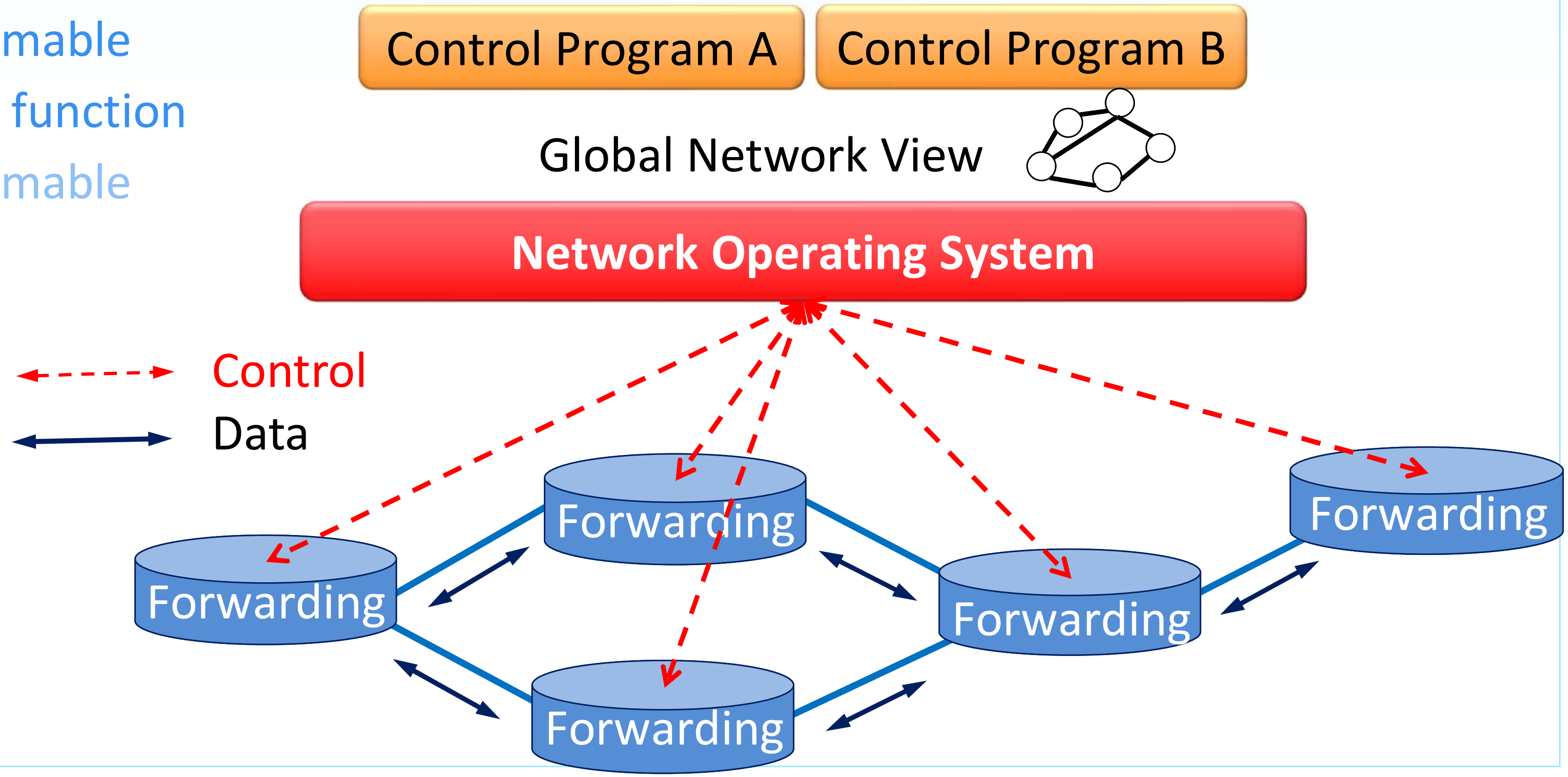
SDN Concept

- SDN separates control and data plane functions



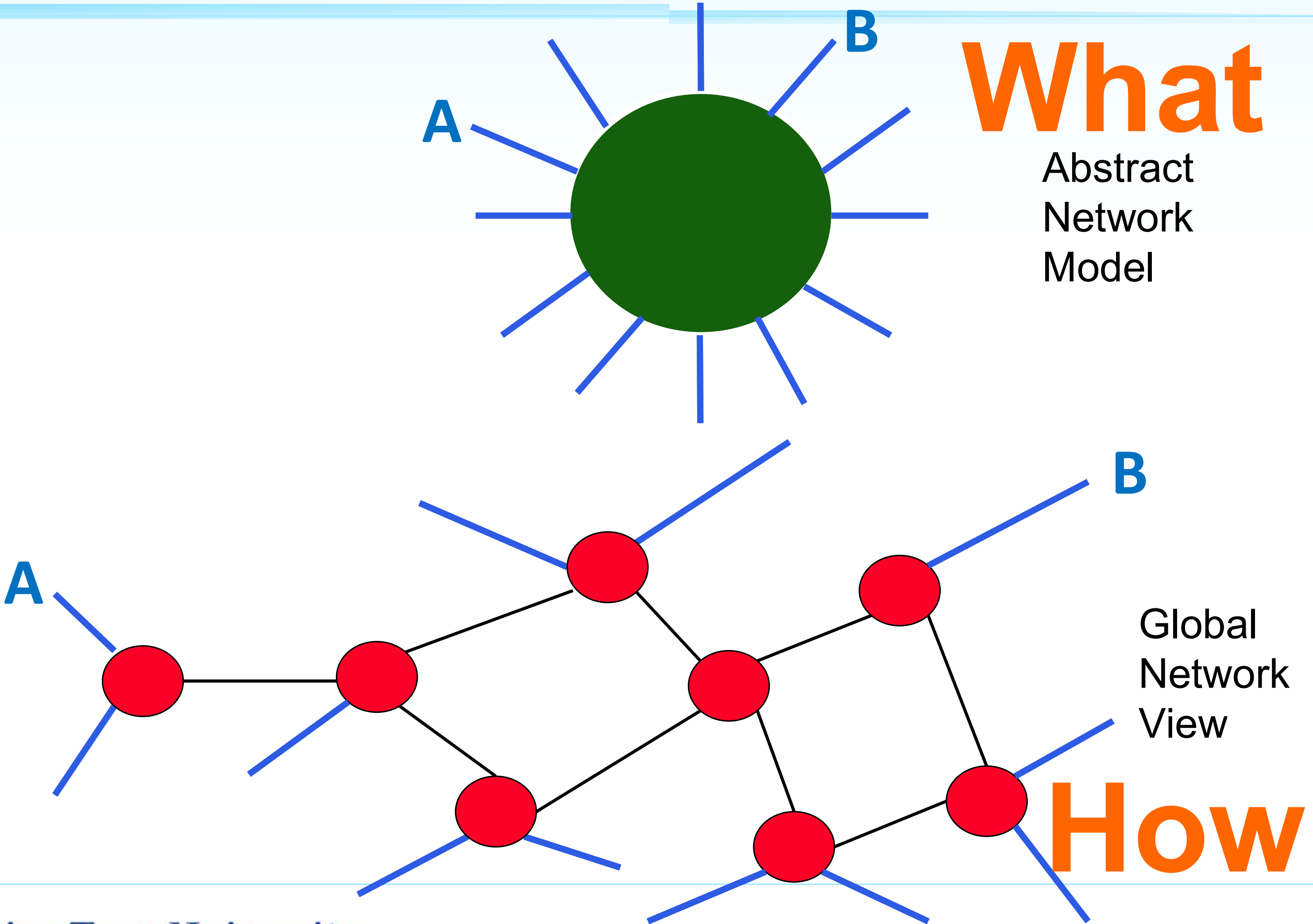
SDN – Separation of Control and Data Planes

- Separation of Control and Data Planes
- Centralized Control
  - Programmable
- Data plane function
  - Programmable

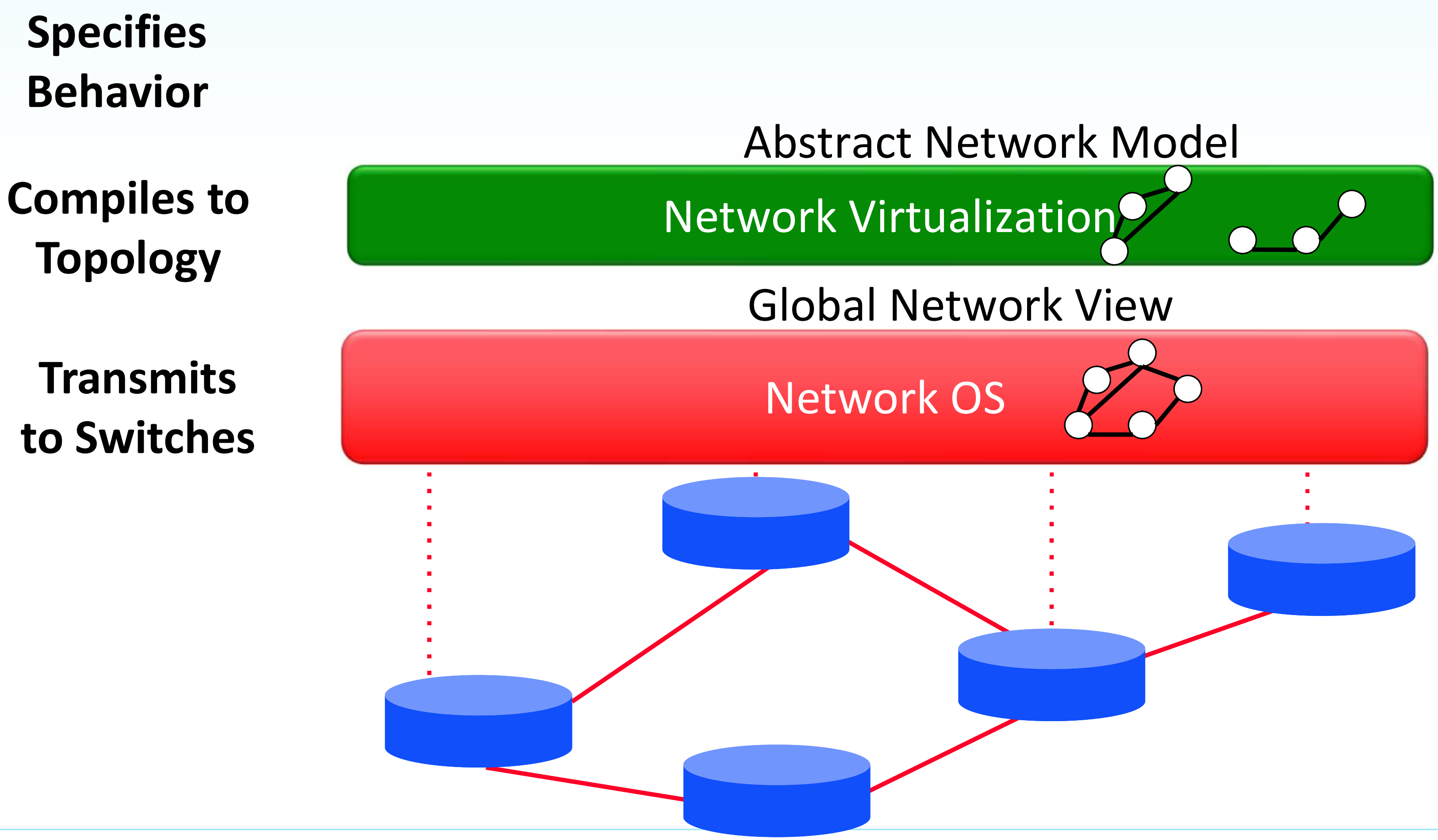


Credited to Prof. Scott Shenker (UC, Berkeley) and Prof.Nick Mckeown (Stanford University)

Intent Service



Software Defined Network: Take 2



Now I can tailor my network to meet my needs!

I can….

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…

In the Beginning…

- OpenFlow was simple

- A single rule table
  - Priority, pattern, actions, counters, timeouts
- Matching on any of 12 fields, e.g.,
  - MAC addresses
  - IP addresses
  - Transport protocol
  - Transport port numbers

Over Five Years…

- Proliferation of header fields

Version	Date	# Headers
OF 1.0	Dec 2009	12
OF 1.1	Feb 2011	15
OF 1.2	Dec 2011	36
OF 1.3	Jun 2012	40
OF 1.4	Oct 2013	41

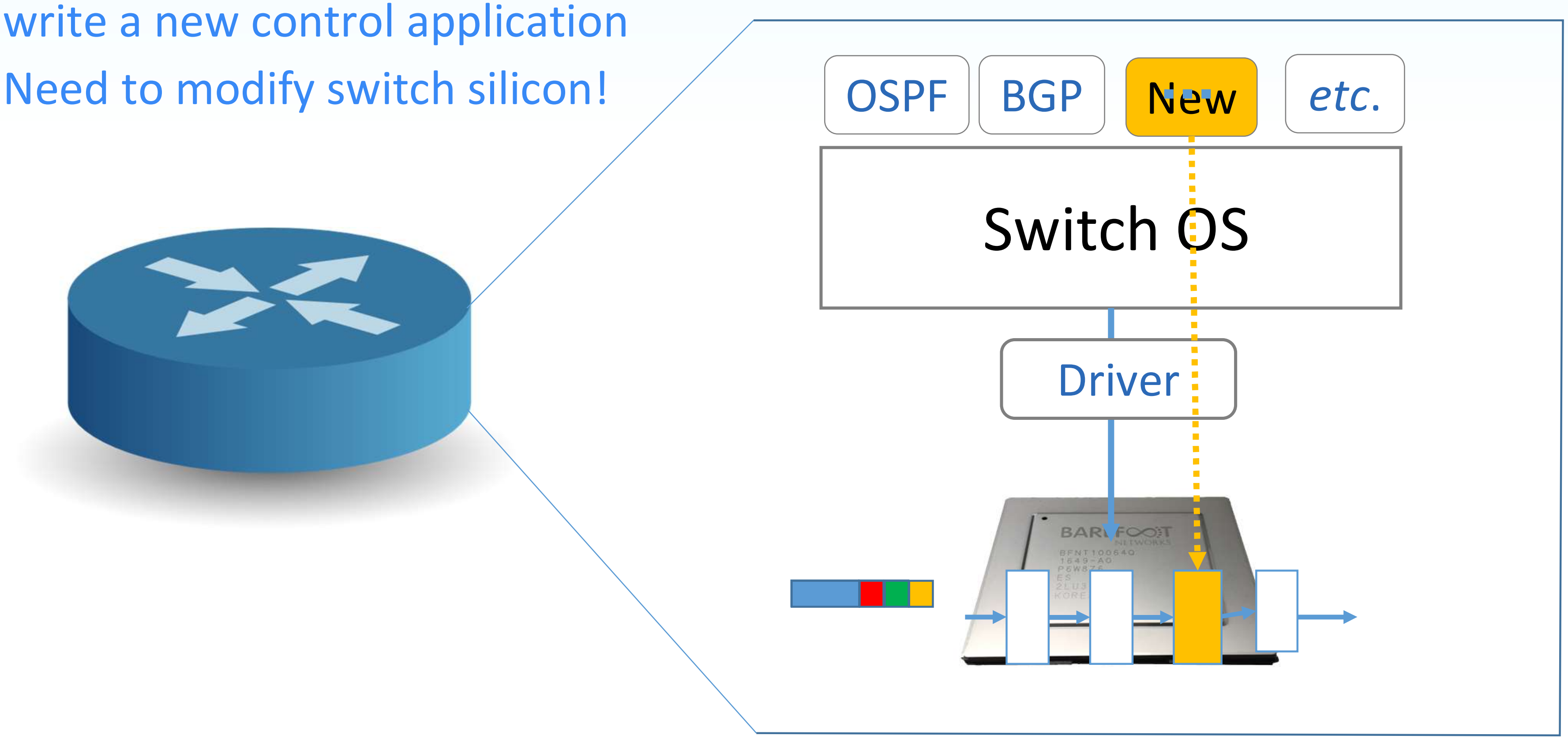
- Multiple stages of heterogeneous tables
- Still not enough (e.g., VXLAN, NVGRE, STT, …)

NCTU Computer Science

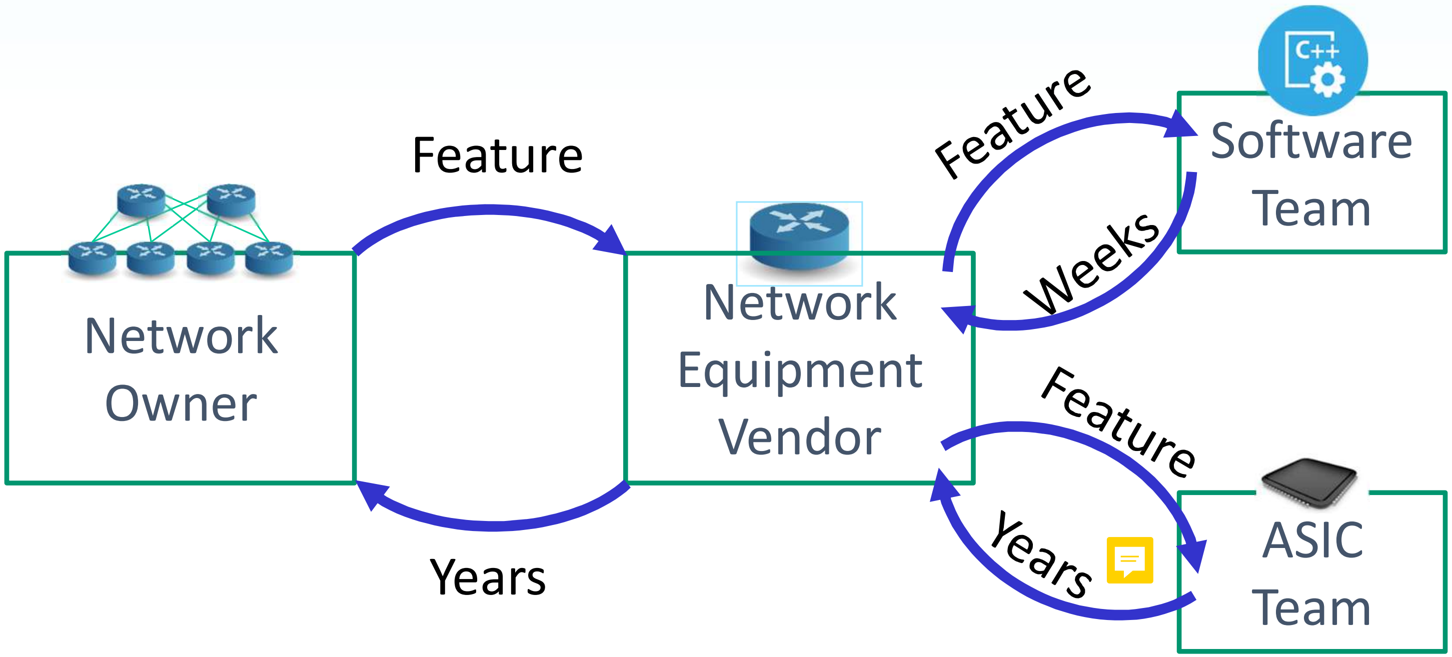
Where does it stop?!?

Need a New Feature on Network

- Need to
  - write a new control application
  - Need to modify switch silicon!



What SDN pioneers had realized …

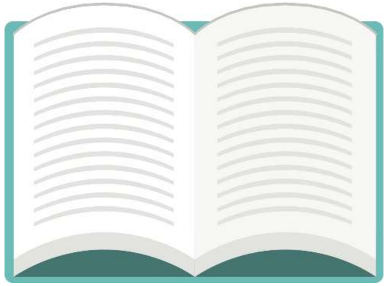


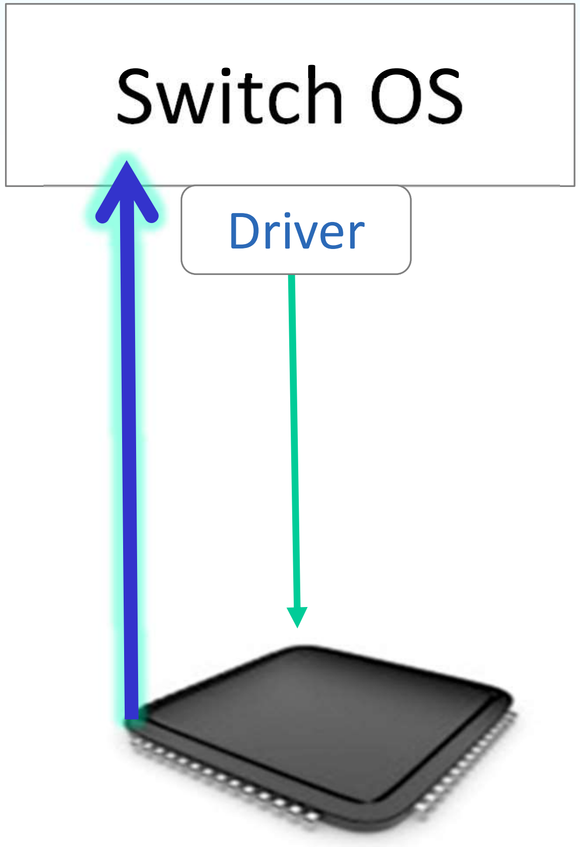
When you need a new feature…

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 "Bottoms-up"

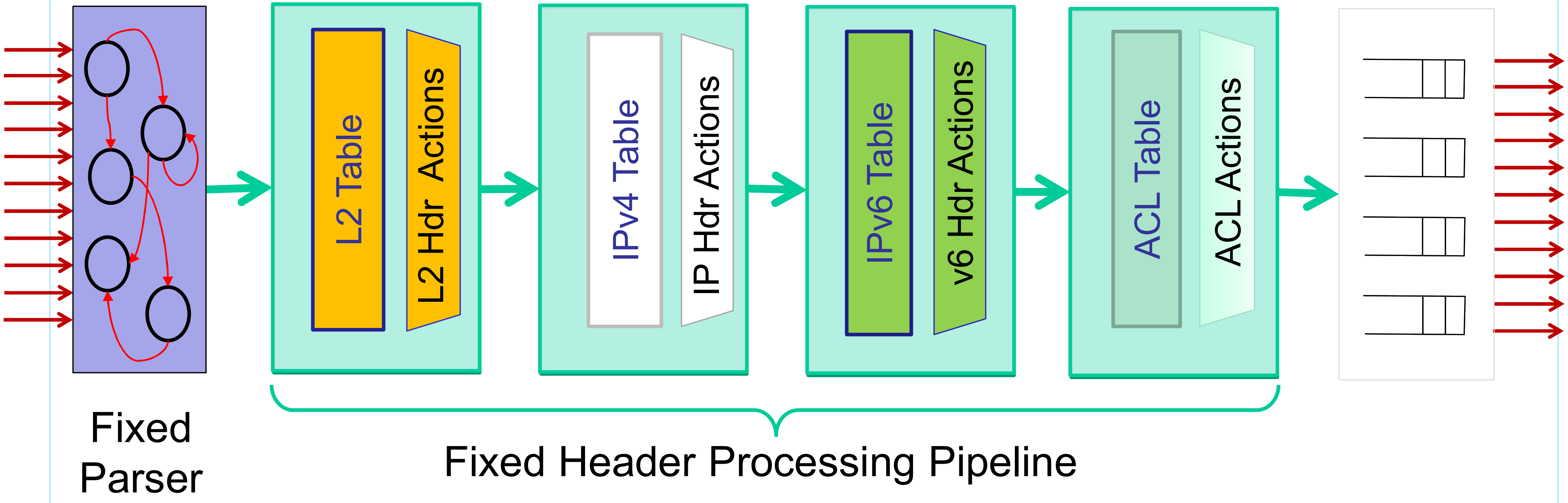
"This is how I process packets …"





Fixed-function switch

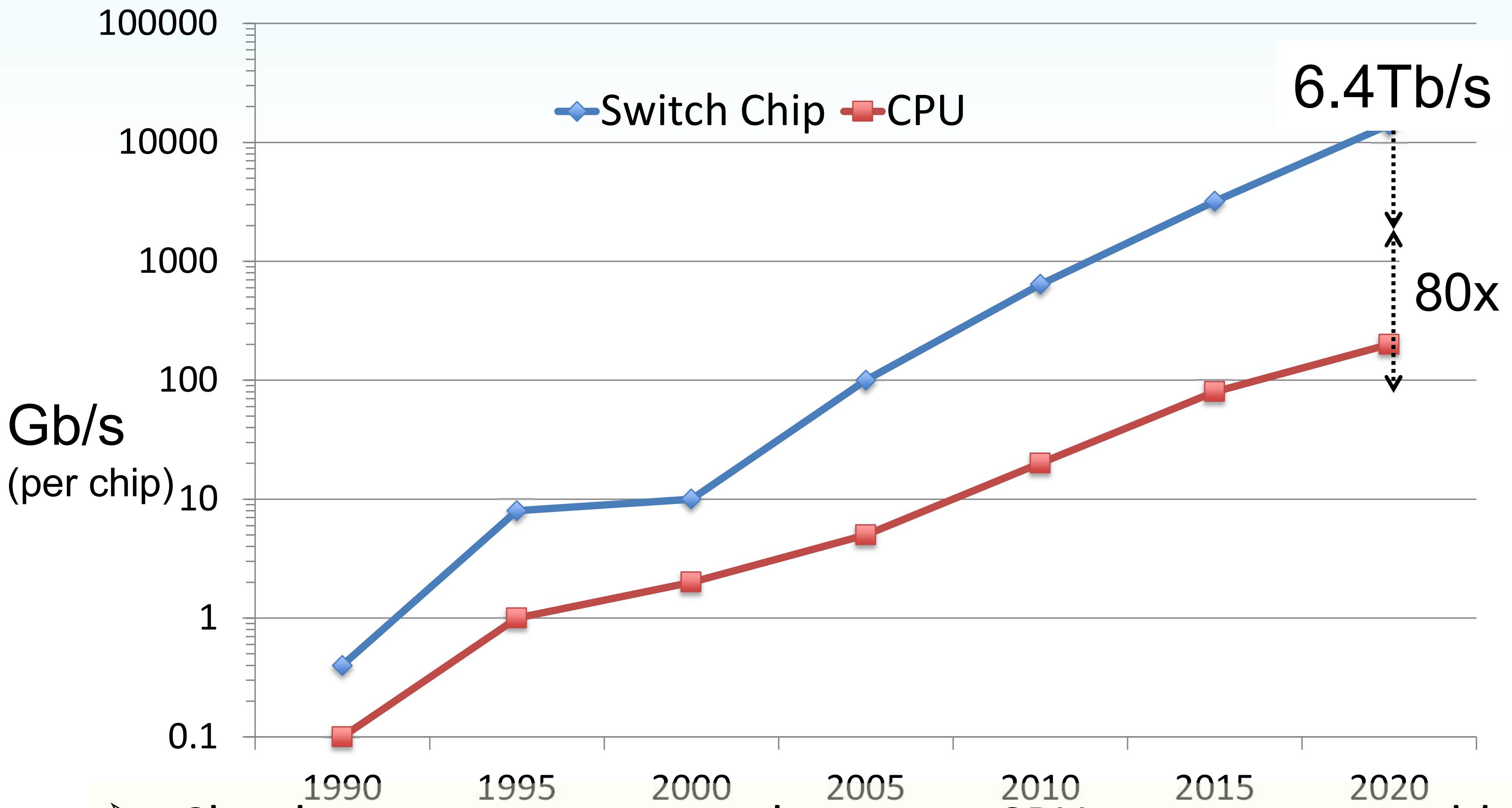
Switch with fixed function pipeline



“Programmable switches run 10x slower, consume more power and cost more.”

Conventional wisdom in Networking

Packet Forwarding Speeds



- ➢ Clearly, cannot use general purpose CPUs as programmable switch chips

Domain Specific Processors

Computers

Java

Compiler

CPU

Graphics

OpenCL

Compiler

GPU

Signal Processing

Matlab

Compiler

DSP

Machine Learning

TensorFlow

Compiler

TPU

>>>

Networking

Language

Compiler

?

SDN

Act 2

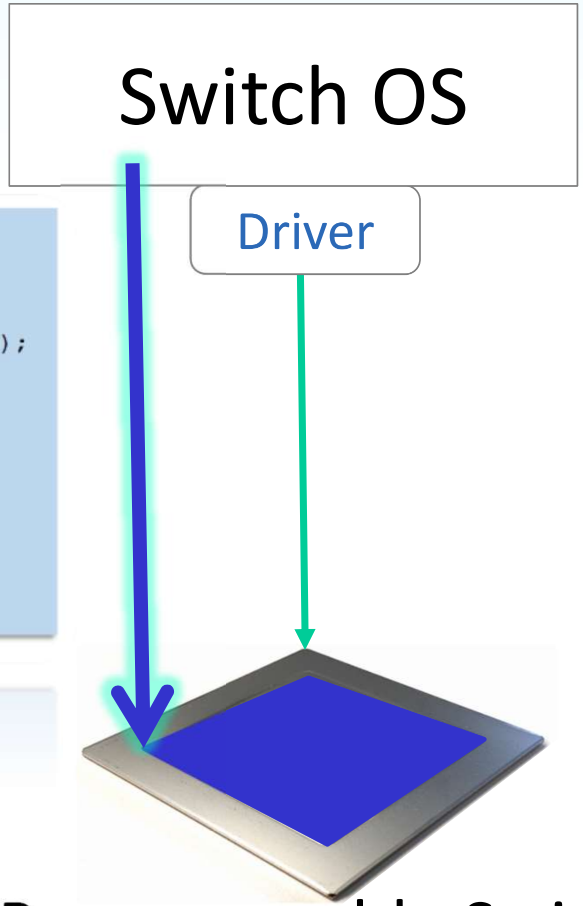
In which network system developers take charge of their forwarding plane too

Network systems are starting to be programmed "top-down"

"This is precisely how you must process packets"

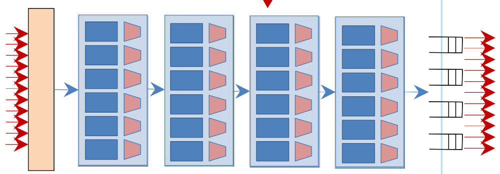
```
table int_table {
  reads {
    ip.protocol;
  }
  actions {
    export_queue_latency;
  }
}
```

```
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);
}
```

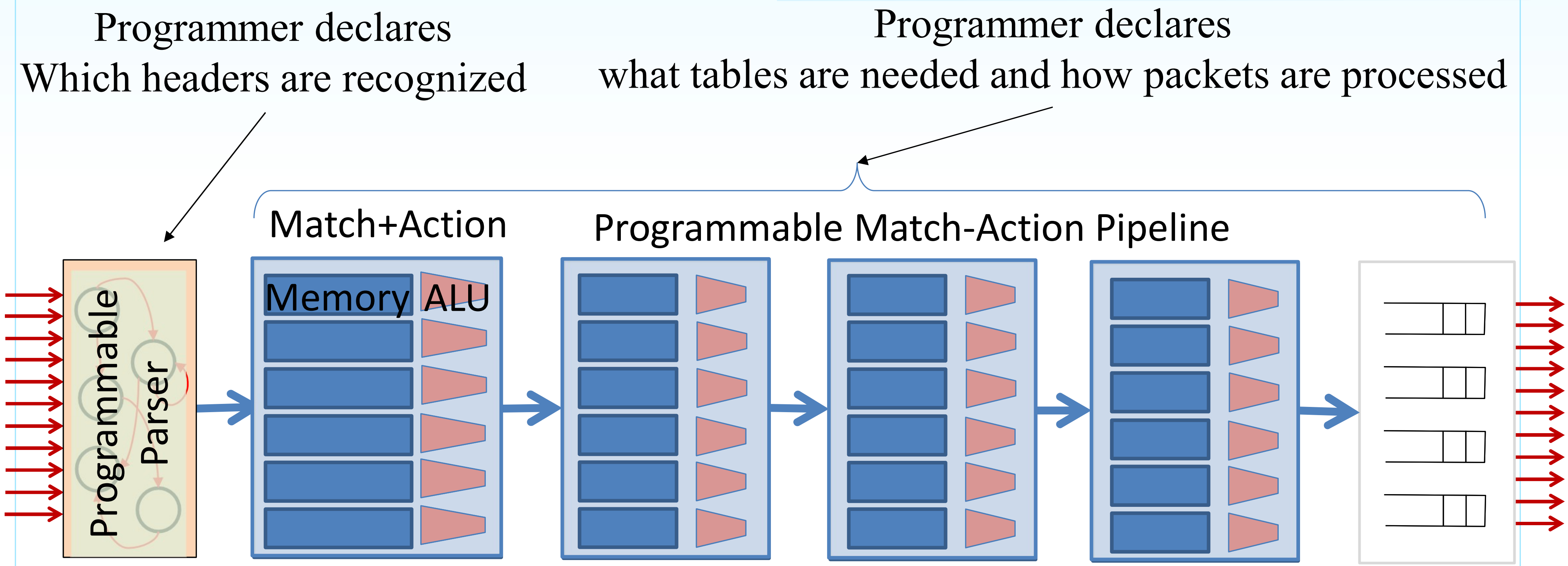


Programmable Switch

Domain Specific Processors

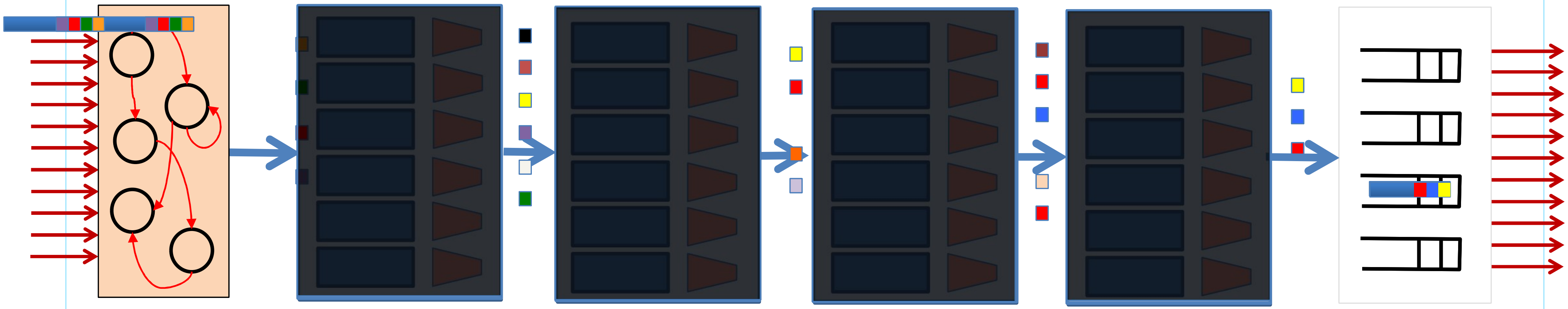
Computers	Graphics	Signal Processing	Machine Learning		Networking
Java	OpenCL	Matlab	TensorFlow	>>>	P4
Compiler	Compiler	Compiler	Compiler		Compiler
					
CPU	GPU	DSP	TPU		PISA (Protocol-Independent Switch Architecture)

PISA: Protocol Independent Switch Architecture

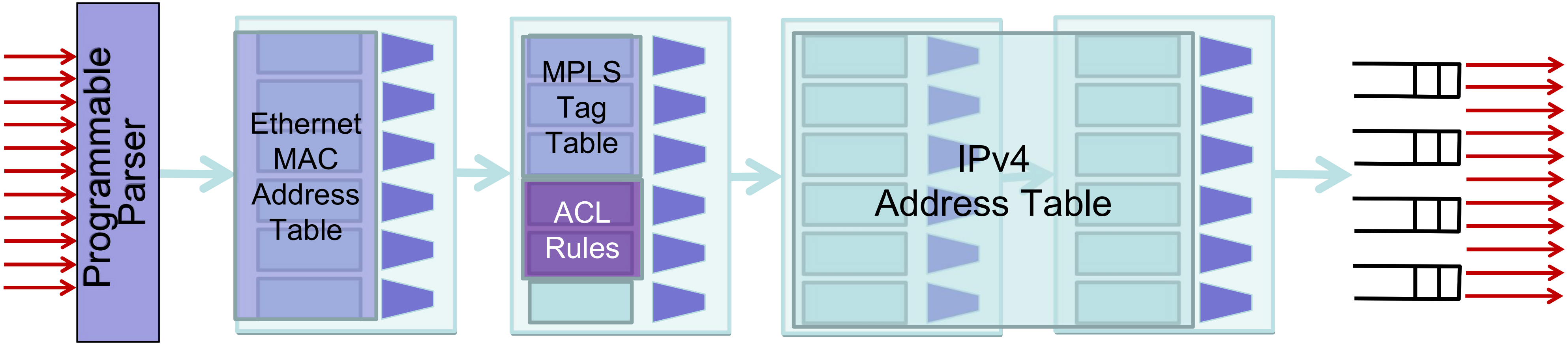


All stages are identical. A "compiler target".

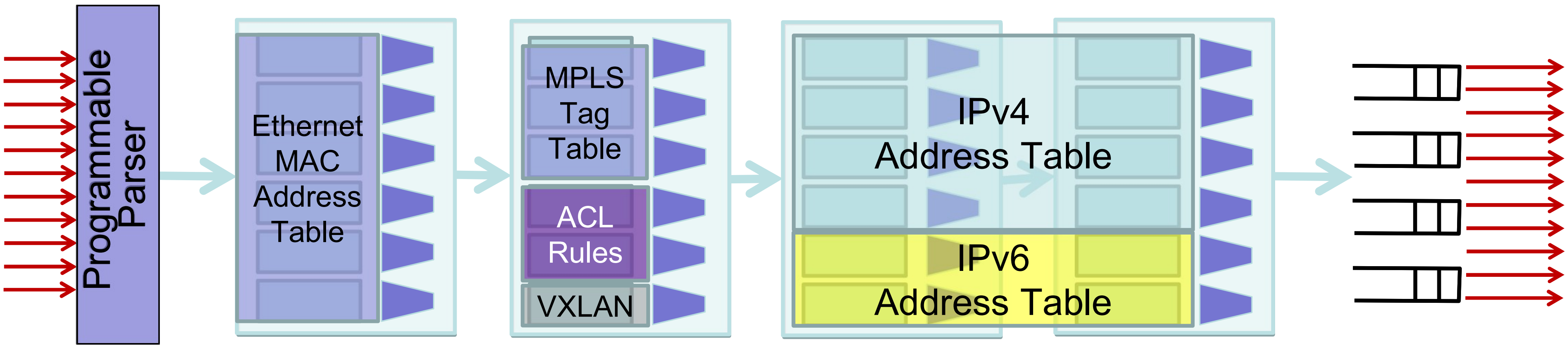
PISA: Protocol Independent Switch Architecture



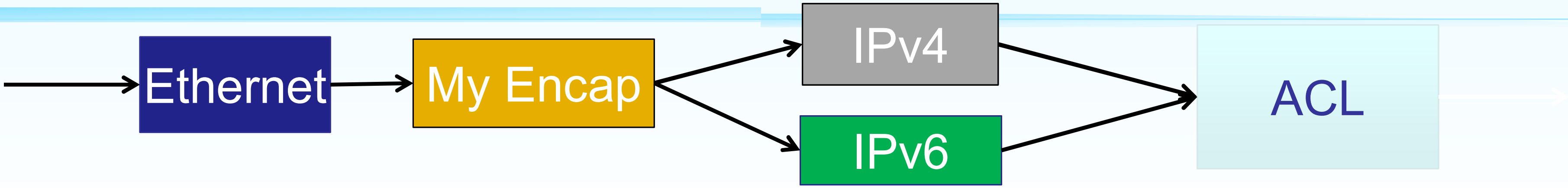
PISA – Programmable

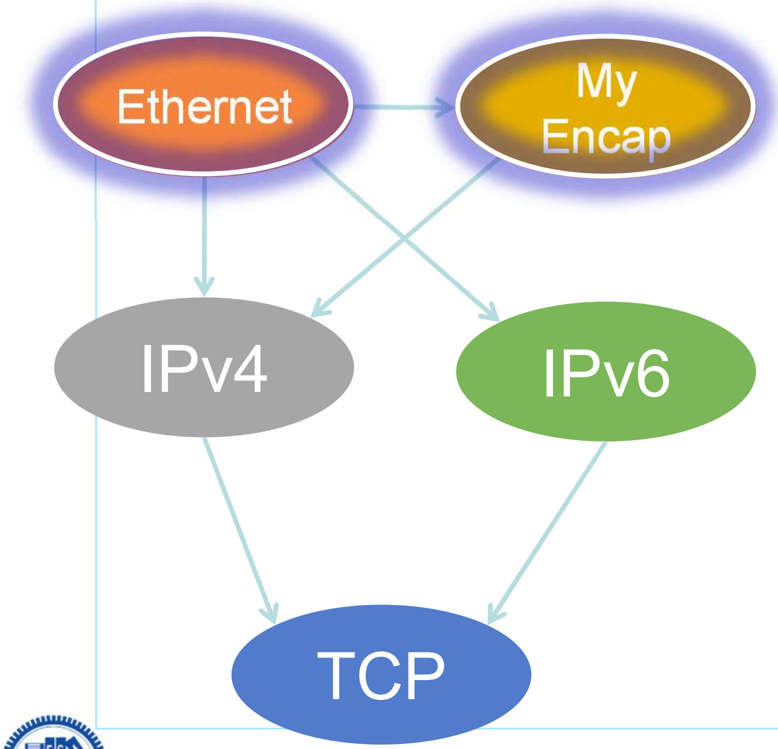


Re-program in the Field



P4 program example: Defining Headers

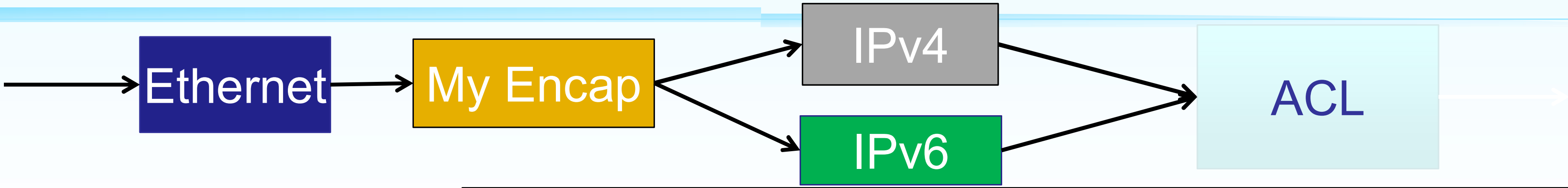


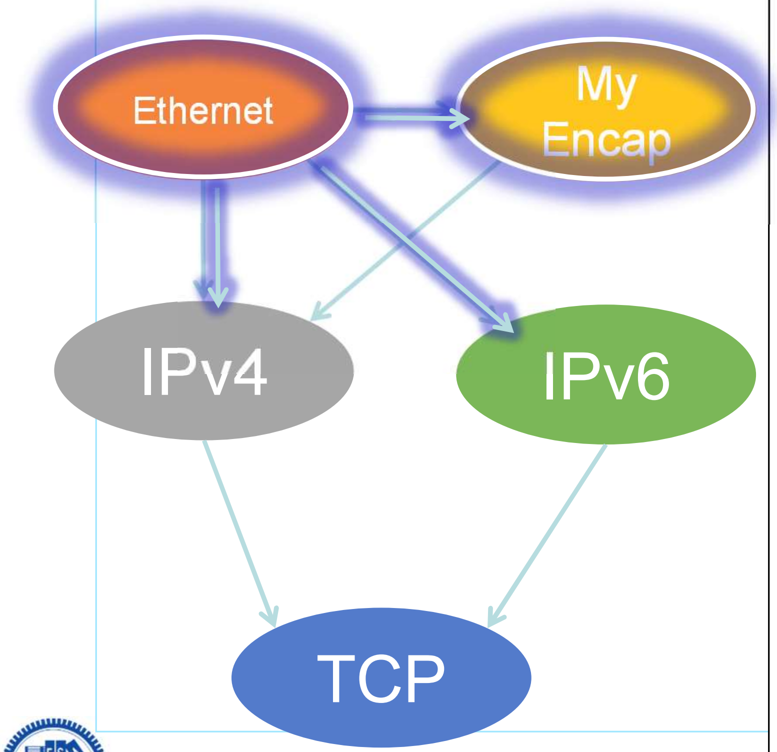



```
header ethernet_t {
        bit<48> dst_addr;
        bit<48> src_addr;
        bit<16> ether_type;
}
```

```
header my_encap_t {
        bit<12> foo;
        bit<8> bar;
        bit<4> baz;
        bit<4> qux;
        bit<4> next_protocol;
}
```

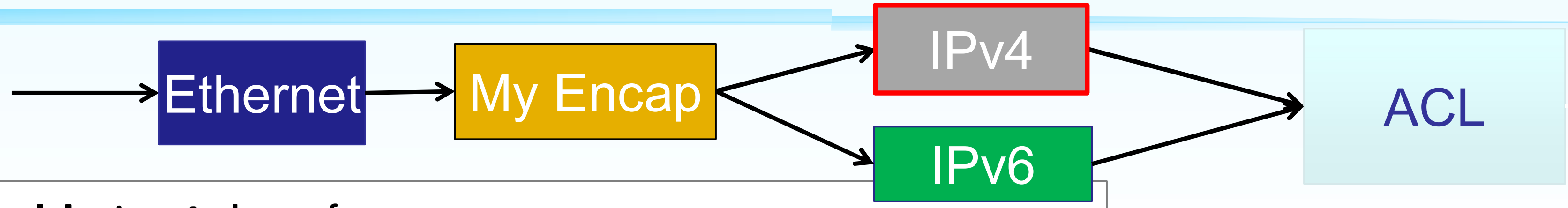
P4 Program Example: Parsing Headers






```
parser MyParser(packet_in packet,
          out headers_t hdr,
          inout my_metadata_t metadata,
          inout standard_metadata_t standard_metadata){
 state parse_ethernet {
          packet.extract(hdr.ethernet);
          transition select(hdr.ethernet.ether_type) {
              0x8100 : parse_my_encap;
              0x0800 : parse_ipv4;
              0x86DD : parse_ipv6;
               default: accept;
          }
 }
```

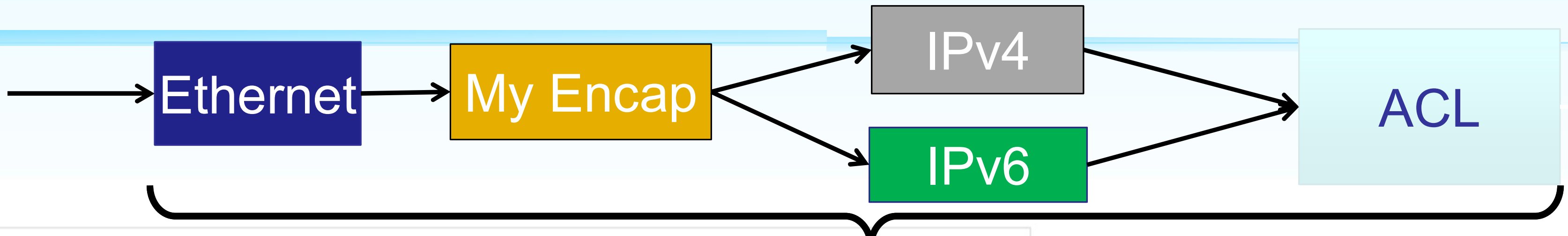
P4 Program Example – Table




```
table ipv4_lpm {
    key = {
        hdr.ipv4.dst_addr : lpm;
        }
    actions = {
        set_next_hop;
        drop;
        }
}
```

```
action set_next_hop(bit<32> nhop_ipv4_addr, bit<9> port) {
        metadata.nhop_ipv4_addr = nhop_ipv4_addr;
        standard_metadata.egress_port = port;
        hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
}
```

P4 Program Example – Control




```
table ipv4_lpm {
    key = {
        hdr.ipv4.ds
    }
    actions = {
        set_next_h
        drop;
    }
}
```

```
control Ingress (...) {
        action set_next_hop(...) {...};
        ...
        table L2 {...};
        ...
        apply {
                L2.apply();
                my_encap.apply();
                if (hdr.ipv4.IsValid())
                        ipv4_lpm.apply();
                else
                        ipv6_lpm.apply();
                acl.apply();
        }
}
```

port) {

To learn more, visit P4.org

Applications of P4

❶ 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~~

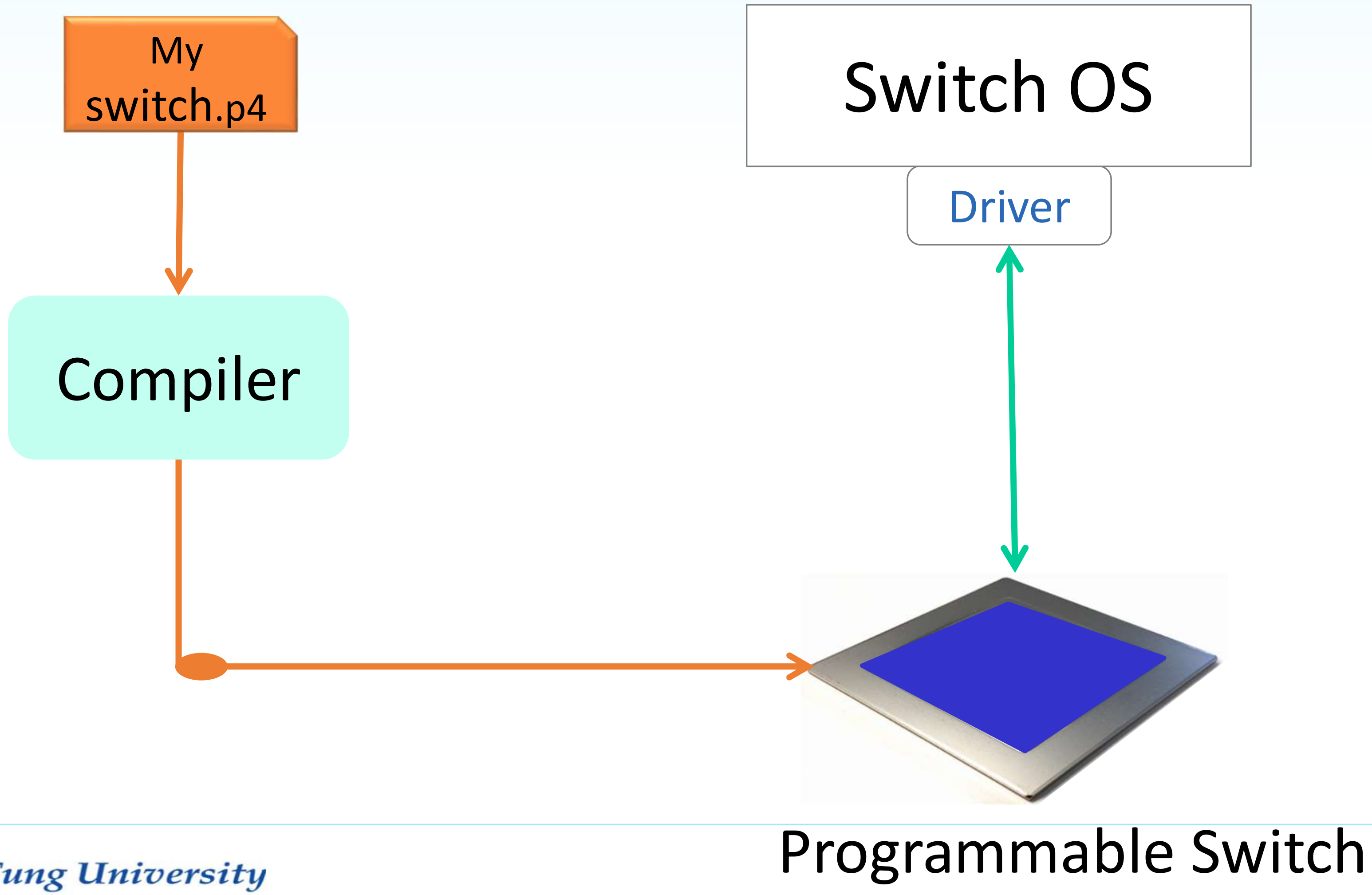
Monitoring & Telemetry
- ~~Ingress Mirroring and Egress Mirroring~~
- Negative Mirroring
- ~~Sflow~~
- INT

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

Protocol Offload
- BFD, OAM

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

Reducing complexity

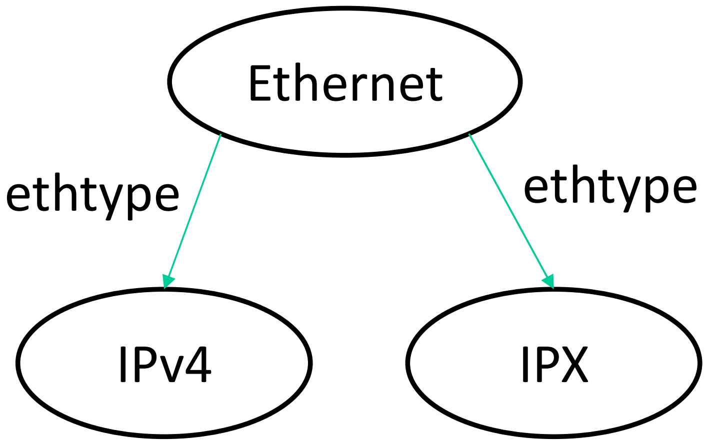


How programmability is being used

❷ Adding new features

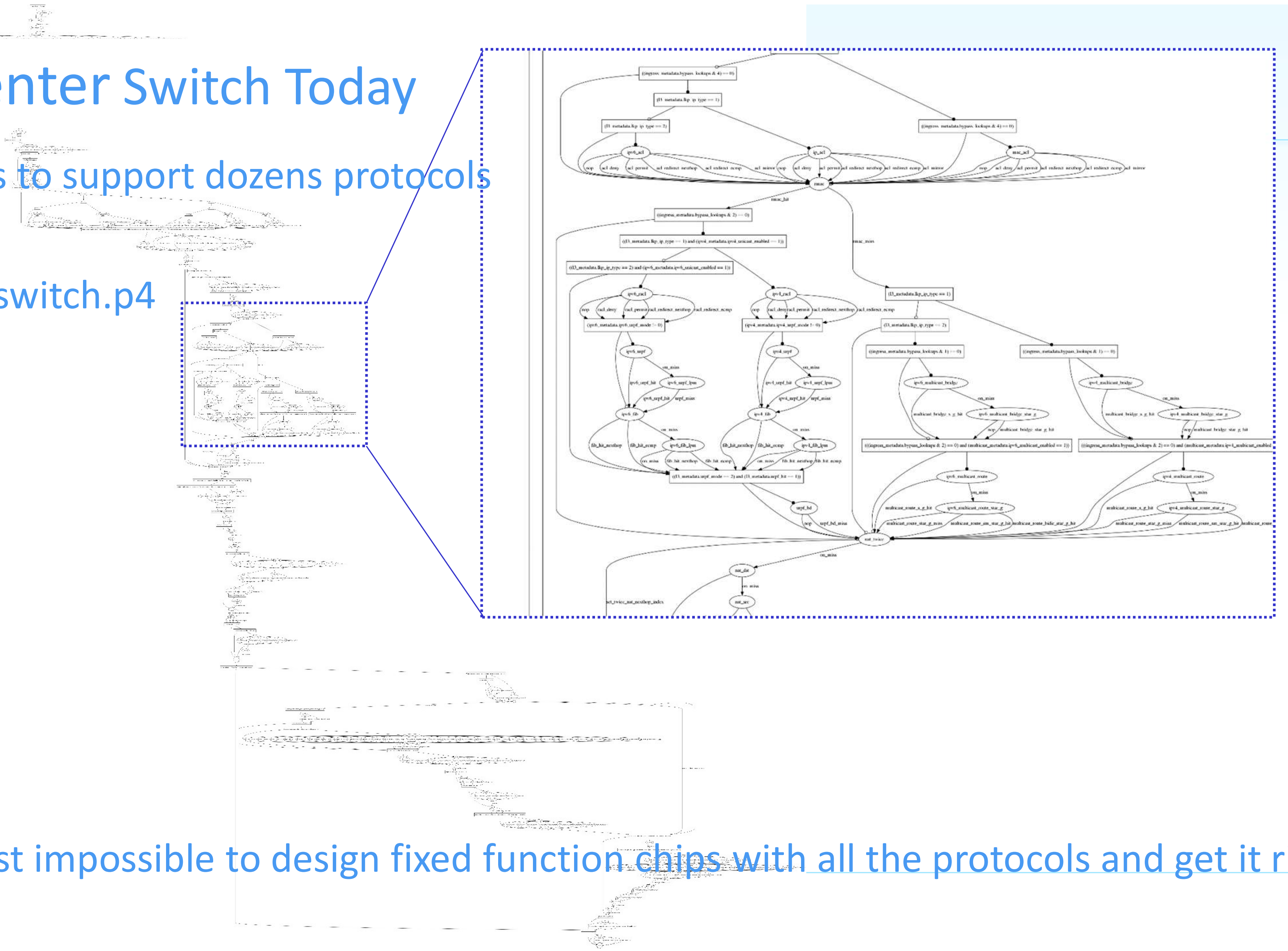
National Chiao Tung University

Protocol complexity 20 years ago



Datacenter Switch Today

- Needs to support dozens protocols
  - switch.p4



➢ Almost impossible to design fixed function chips with all the protocols and get it right.

Adding features: Some examples so far

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*

- MSDC: Massively Scalable Data Center

New applications: Some examples so far

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

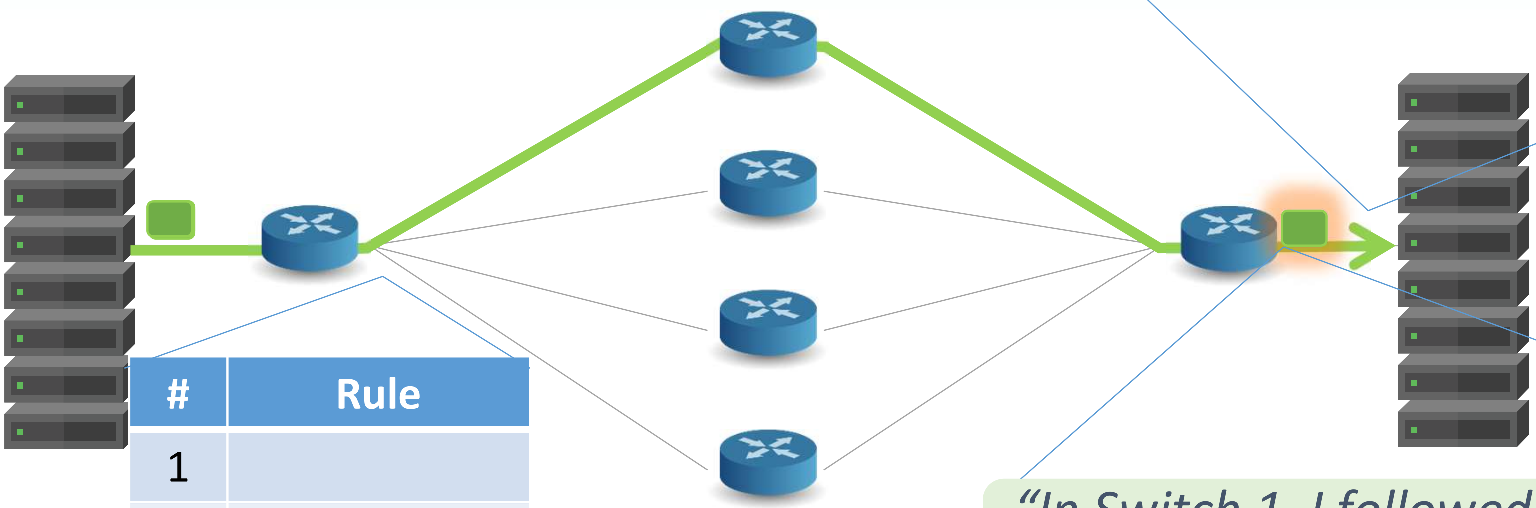
How programmability is being used

❸ Network telemetry

Path and Flow Rules Tracking

❶ *"Which path did my packet take?"*

"I visited Switch 1 @780ns, Switch 9 @1.3µs, Switch 12 @2.4µs"



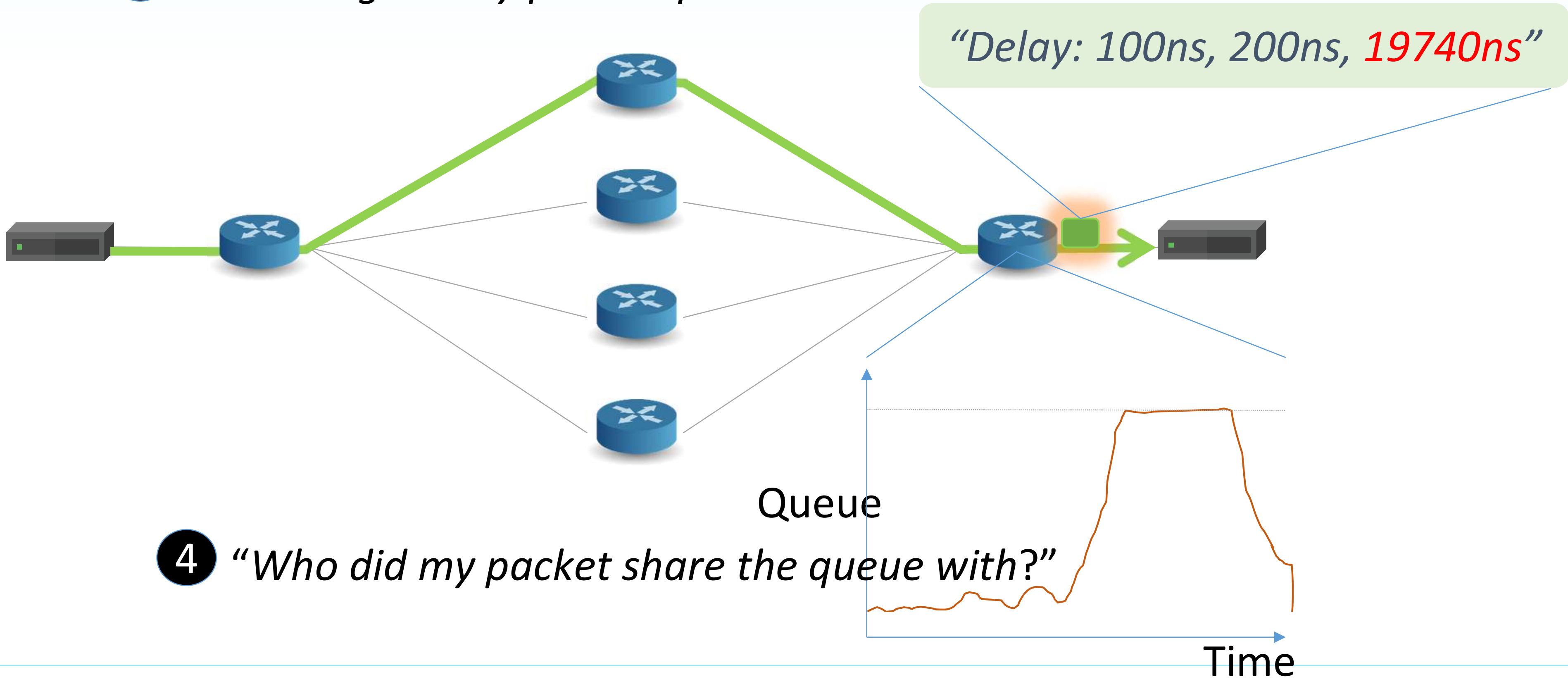
#	Rule
1	
2	
…	
75	192.168.0/24

"In Switch 1, I followed rules 75 and 250. In Switch 9, I followed rules 3 and 80. "

❷ *"Which rules did my packet follow?"*

Queuing Delay

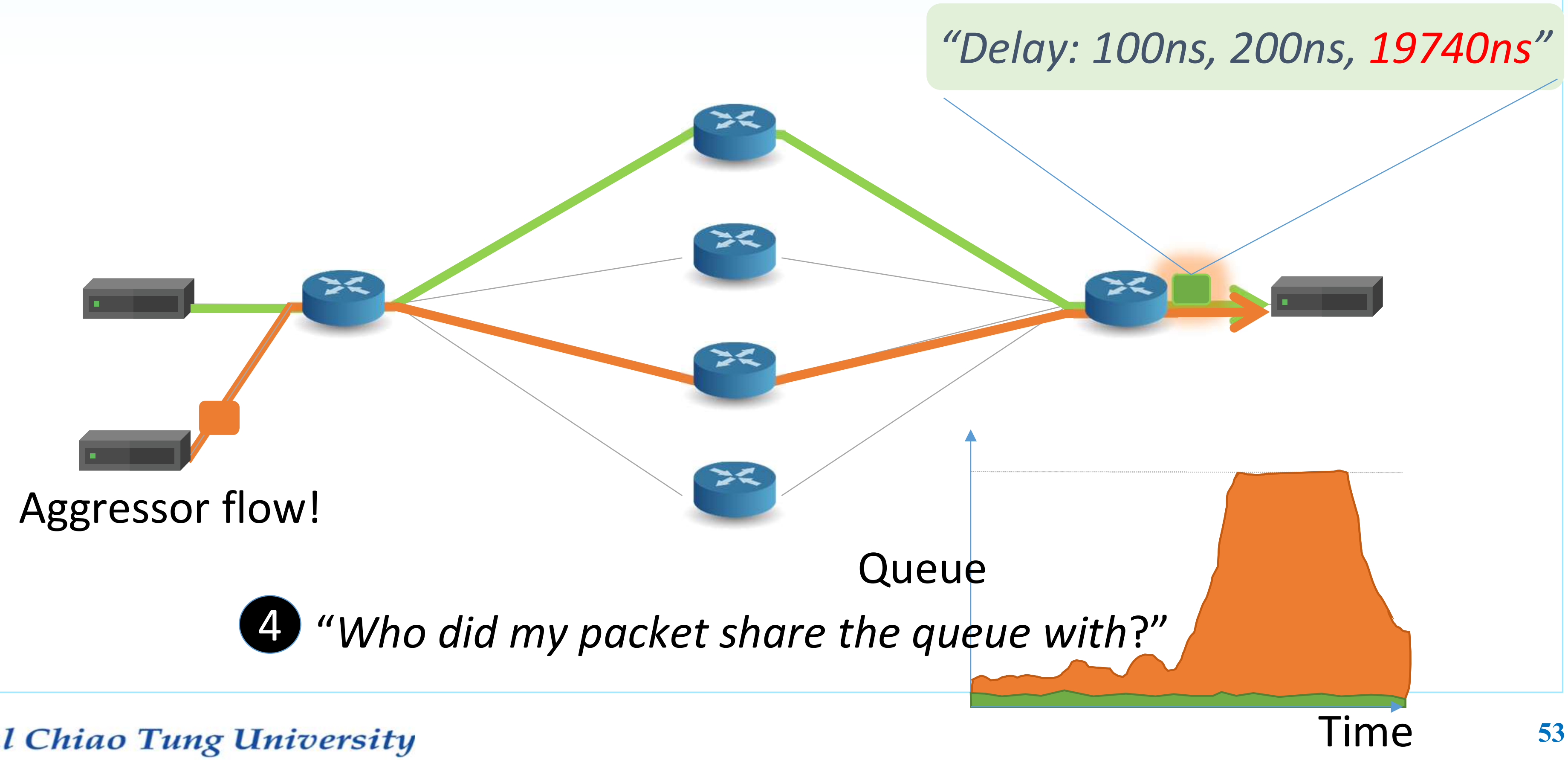
❸ *"How long did my packet queue at each switch?"*



❹ *"Who did my packet share the queue with?"*

Why Long Queuing Delay

❸ *"How long did my packet queue at each switch?"*



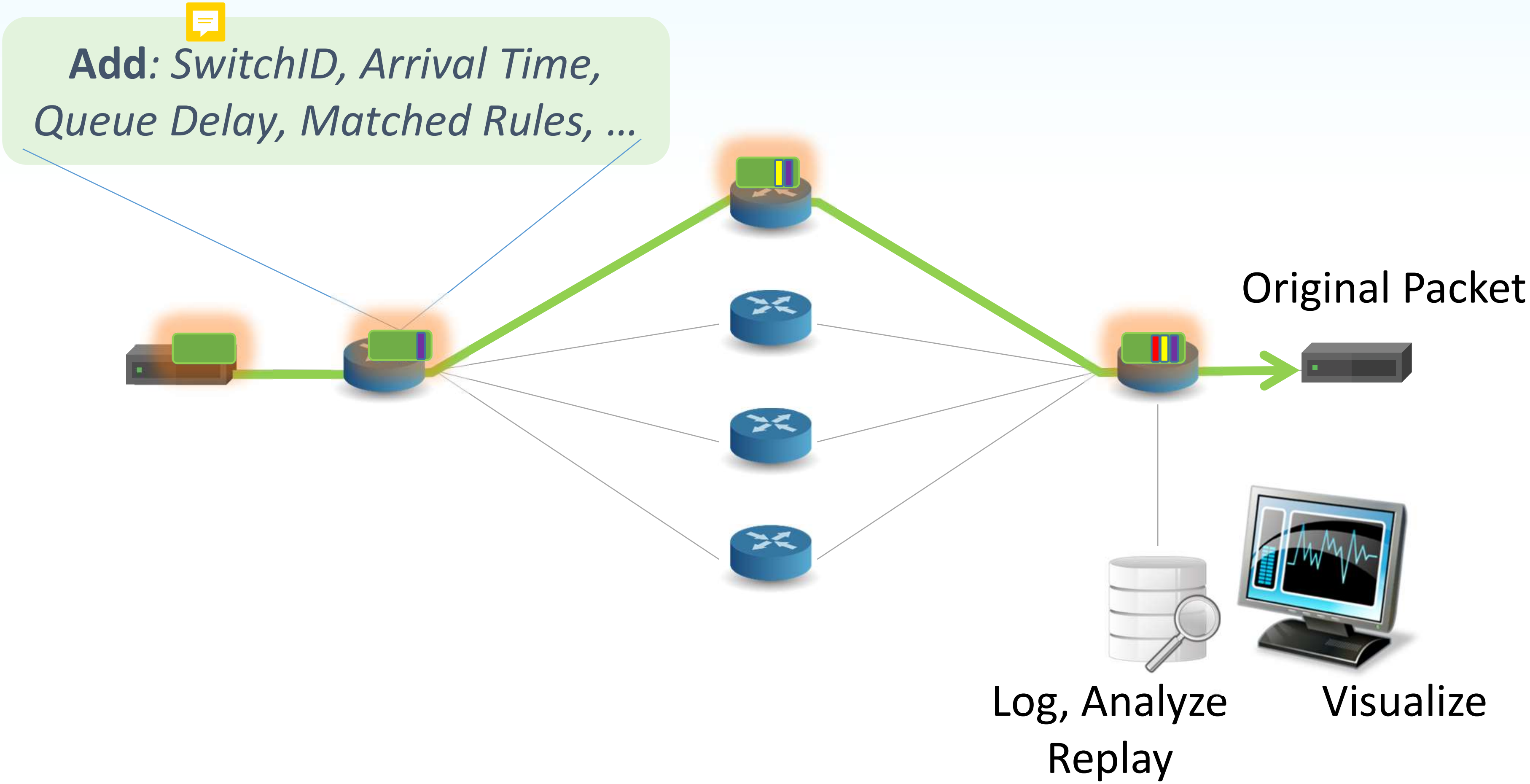
❹ *"Who did my packet share the queue with?"*

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



In summary…

1. **SDN is about who is in charge**!

   Act 1: Network owners and operators took charge of how their networks are controlled.

   Act 2: They also decide how packets are processed.

2. **Chip technology**: Programmable forwarding now has the same power, performance and cost as fixed function.

3. **New ideas**: Beautiful new ideas now owned by the programmer, not the chip designer.