Programmable networks

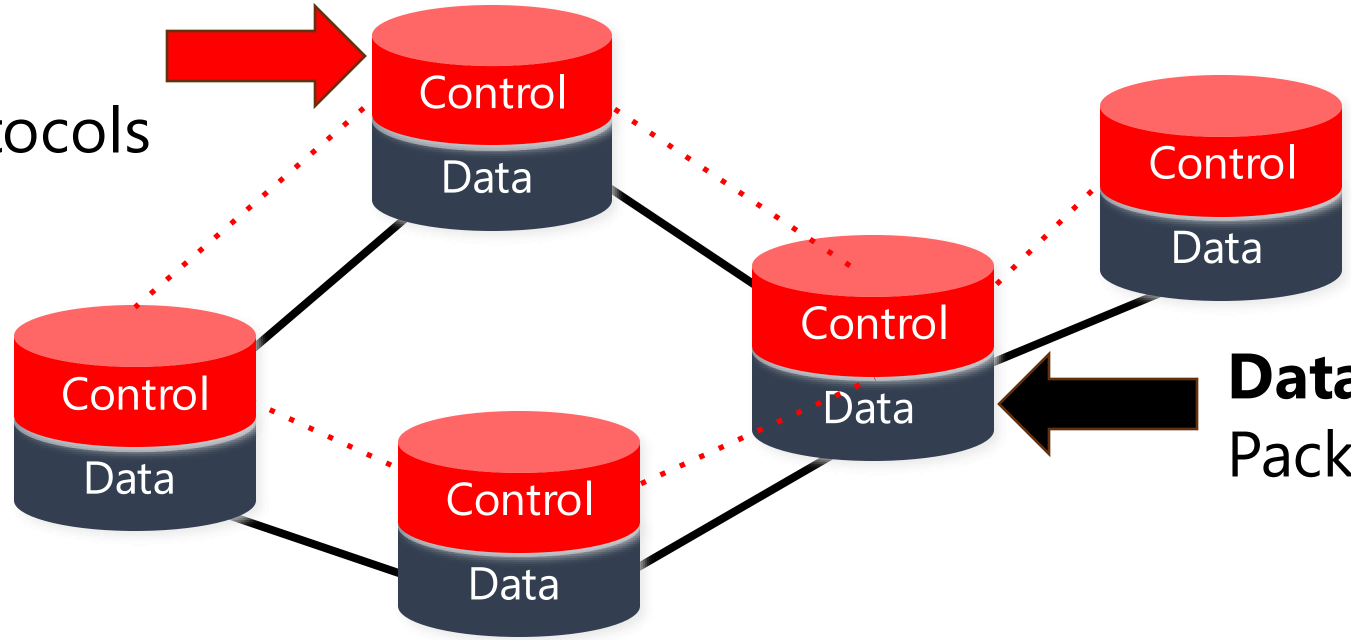
CS356: Computer Networks, Fall 2024

11/19/2024

Guest instructor: Daehyeok Kim

Three "planes" in networks

Control plane:
Distributed routing protocols



Data plane:
Packet forwarding

Management plane:
Has to reverse engineer what the control plane does

Network management becomes complex

Need for expressing "network-wide" management policies

Q: What are examples of network-wide policies?

Traffic engineering

- "Keep all links below 70% utilization"
- "Balancing utilization across links"

Reachability (or security)

- "Do not allow hosts in subnet B to access servers in subnet A"

➔ Hard to achieve using low-level configuration on each router

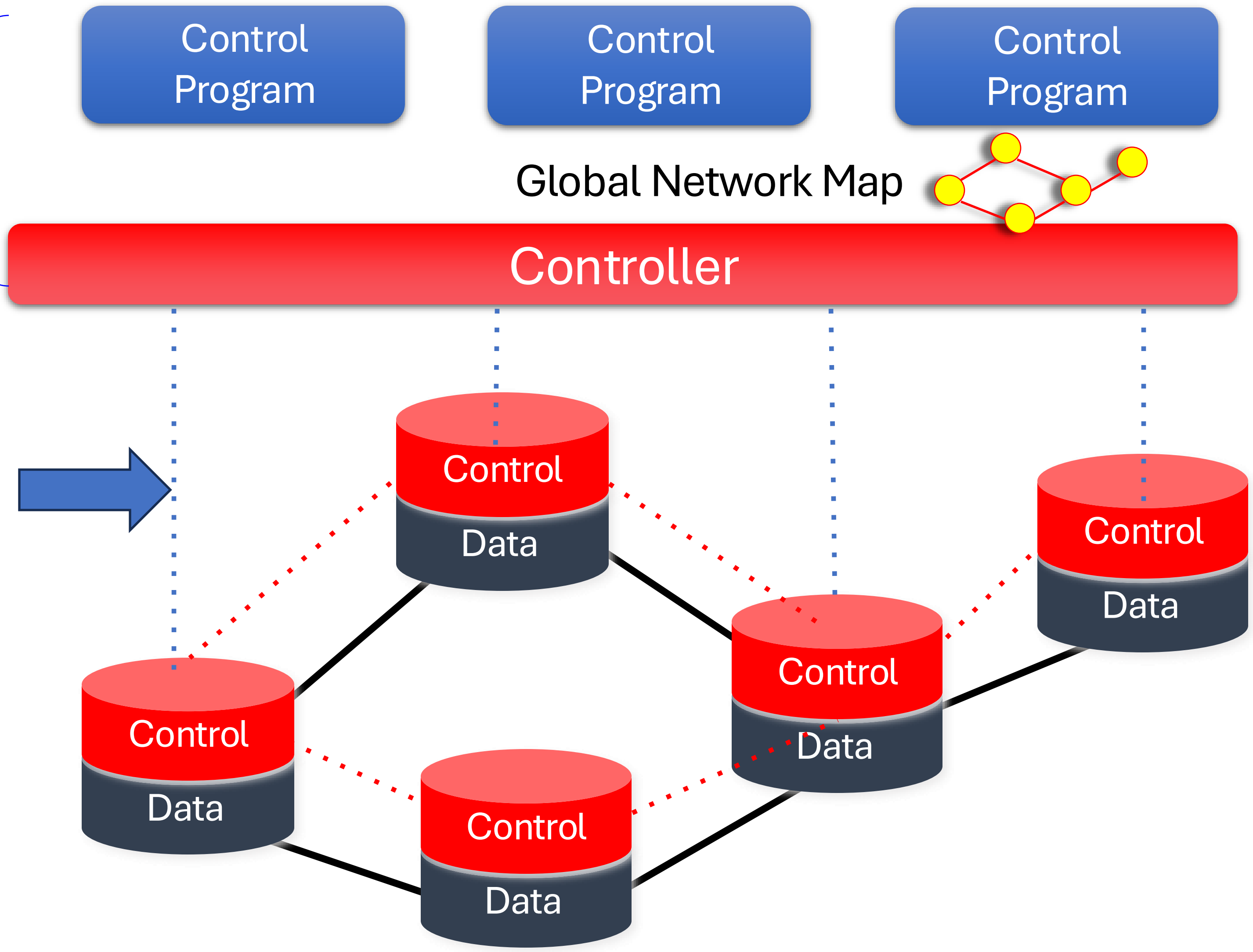
Challenges with IP networks

- Lack of abstractions
- Inability to express intent
- Unpredictable outcome from complex distributed algorithms
- Interactions among protocols (e.g., IGP & EGP)
- Can't manage a device unless it's properly configured
  - Bootstrap issue – control & management plane dependent on correct data plane
  - Fragility, risk of change

Root cause: IP networks bundle control logic and packet handling

Software-defined networking (SDN): Decoupling the control and data planes

Separate the control from the data plane



OpenFlow

A single (logically centralized) control plane

Motivation of OpenFlow: Innovating campus wiring closets

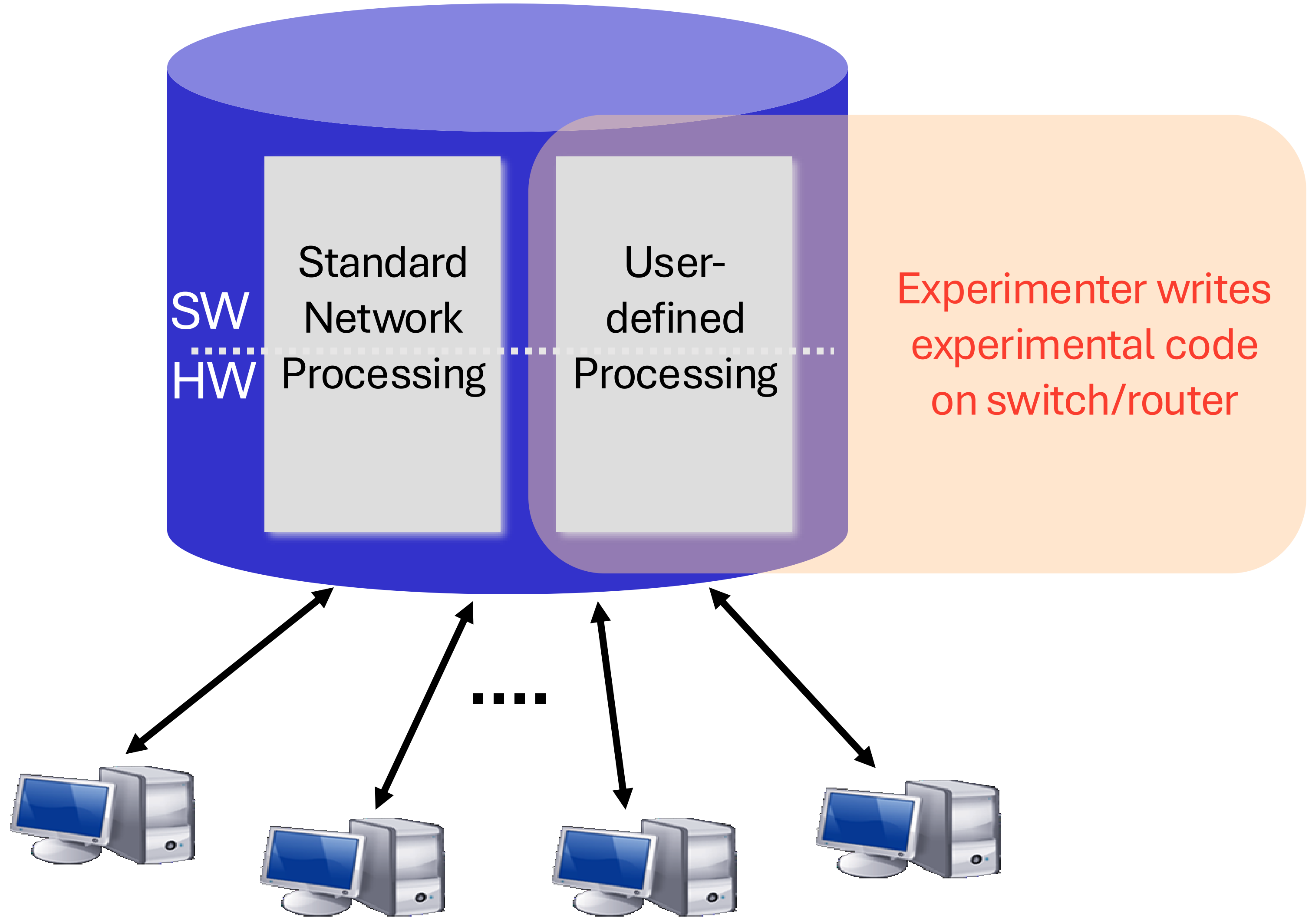
Experiments we'd like to do

- Mobility management
- Network-wide energy management
- New naming/addressing schemes
- Network access control

Problem with traditional networks

- Paths are fixed (by the network)
- IP-only
- Addresses dictated by DNS, DHCP, etc
- No means to add our own processing

Experimenter's dream (Vendor's Nightmare...)



No obvious way

Vendors won't open SW and HW development environment

- Complexity of support
- Market protection and barrier to entry

Hard to build my own

- Software only (e.g., Click): Too slow
- Hardware/software (e.g., NetFPGA): Fanout too small (need >100 ports for wiring closet)

Furthermore, we want…

Isolation: Regular production traffic untouched

Virtualized and programmable: Different flows processed in different ways

Open development environment for all researchers

Flexible definitions of a "flow"

- Individual application traffic
- Aggregated flows
- Alternatives to IP running side-by-side
- …

OpenFlow switching

A way to run experiments in the networks we use everyday

A "pragmatic" compromise

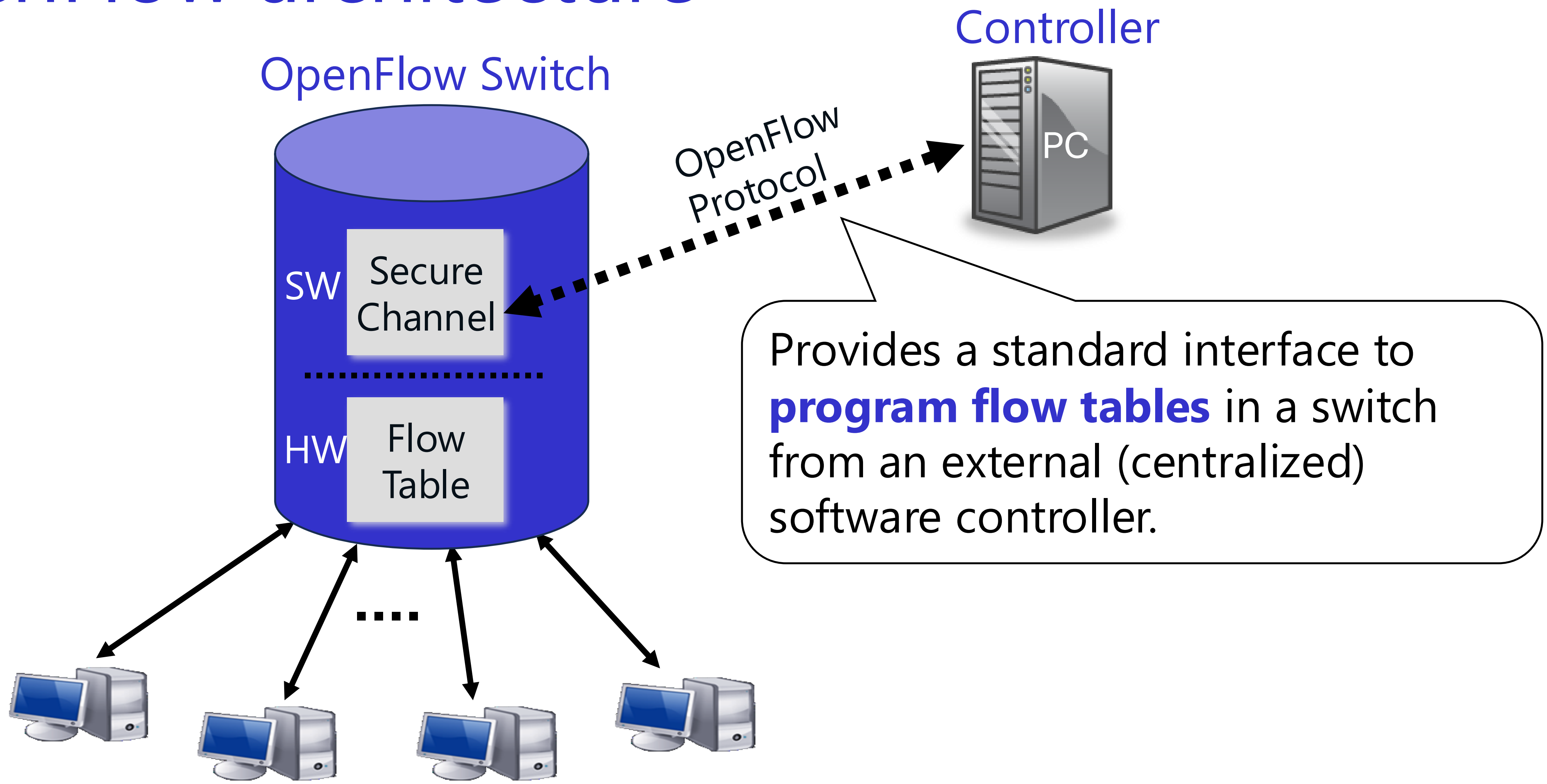
Allow researchers to run experiments in their network…

…without requiring vendors to expose internal workings.

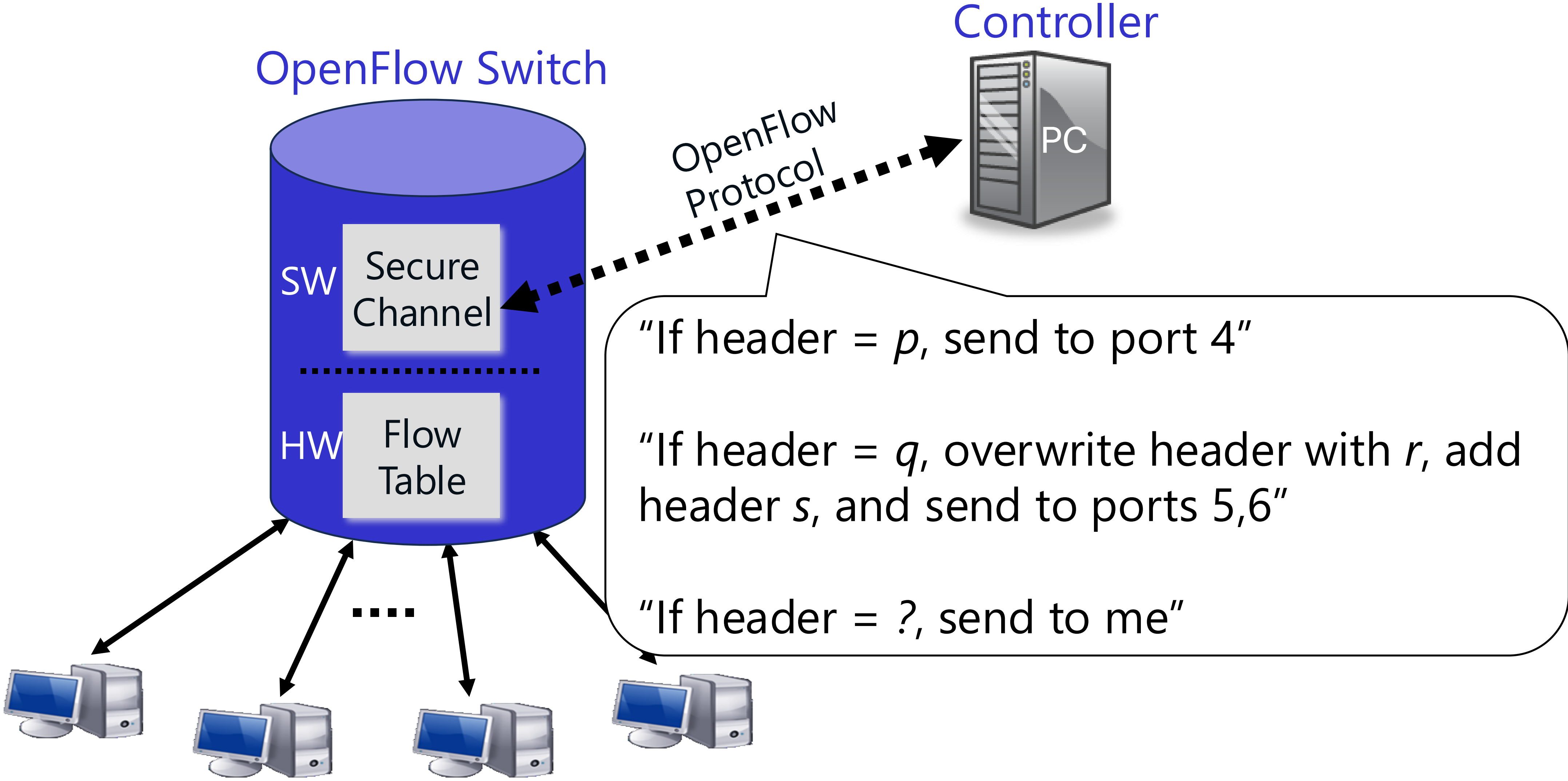
Basic ideas

- An Ethernet switch (e.g., 128-ports of 1GE)
- An open protocol to remotely add/remove flow entries

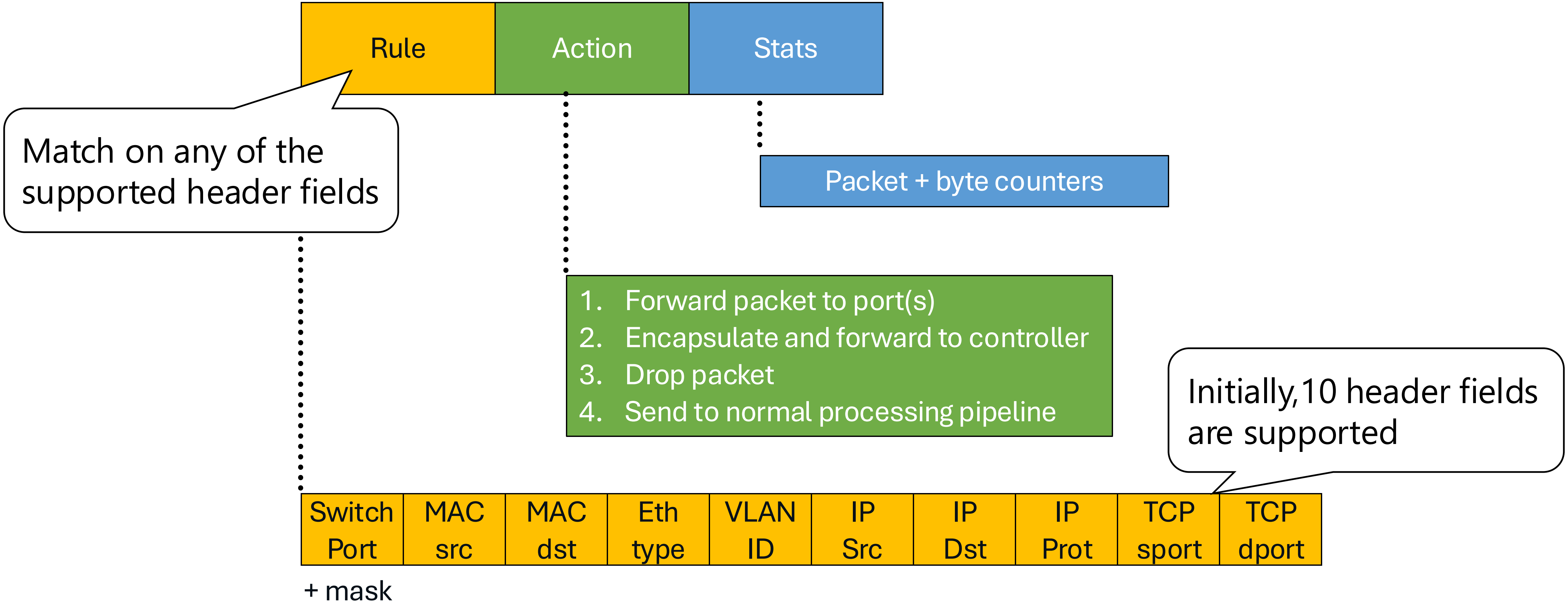
OpenFlow architecture



OpenFlow architecture



Flow table entry
"Type 0" OpenFlow Switch



Example rules

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
Port 3	00:2e:..	00:1f:..	0x0800	vlan1	1.2.3.4	5.6.7.8	4	8888	80	port4

Stateless Firewall

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

What is nice

- Fits well with the **TCAM abstraction**
- Most vendors already have this
- They can just expose this without exposing internals

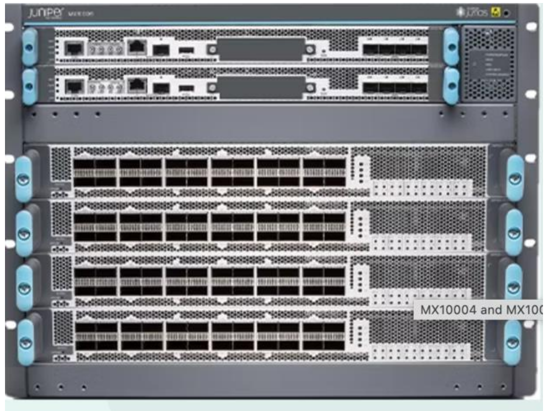
Supported header fields

Version	Date	# of headers
OF 1.0	Dec 2009	12
OF 1.1	Feb 2011	15
OF 1.2	Dec 2011	36
OF 1.3	June 2012	40
OF 1.4	Oct 2013	41
OF 1.5	Mar 2015	45

OpenFlow supported switches

Cisco Catalyst 9000 Series

HP Aruba Series

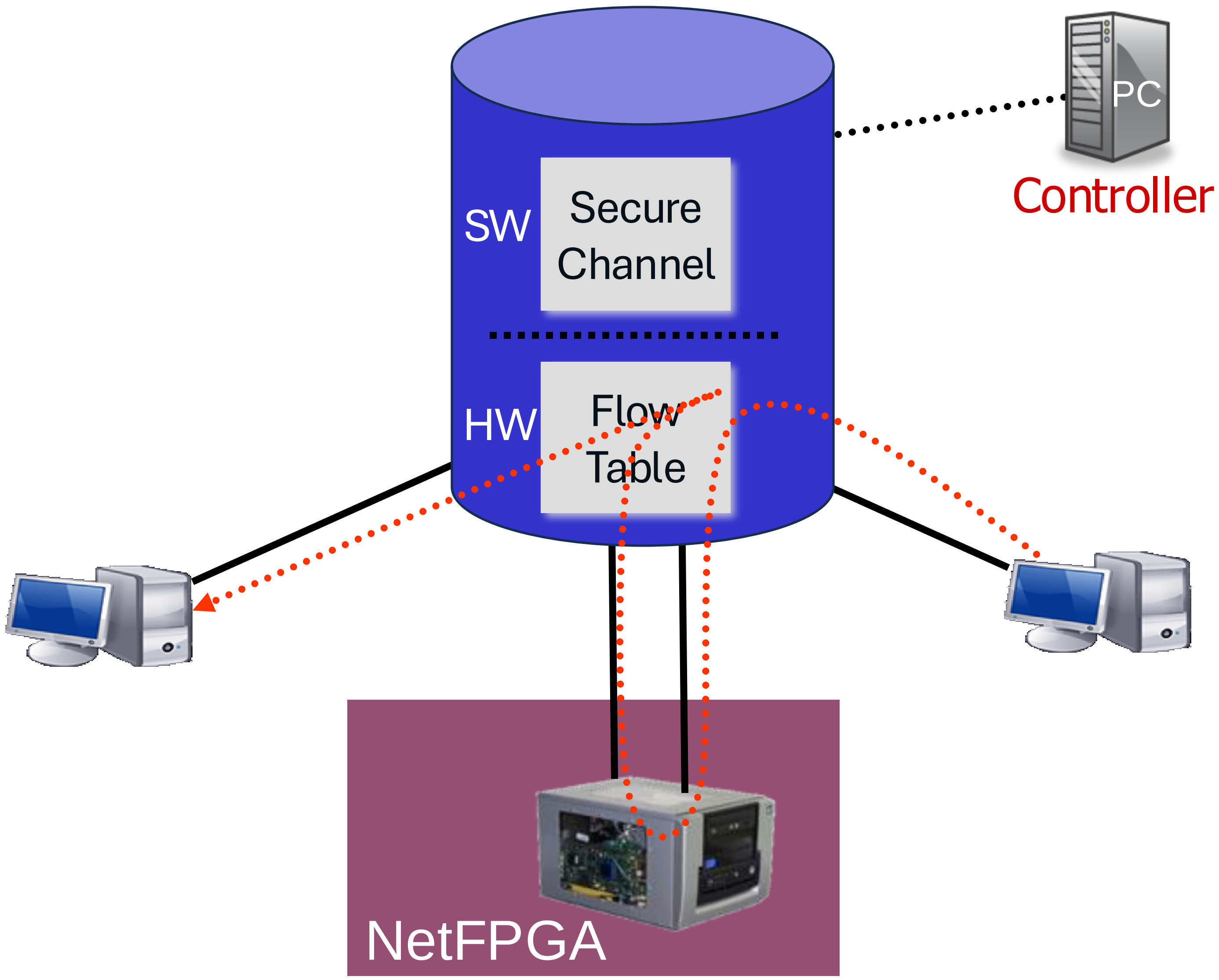


Juniper MX Series

Many others including ones from Big Switch networks, Pica8, Huawei, …

DC operators (e.g., Google, Meta) often build their own white box switches

Experiments at the packet level



SDN used today

It’s been widely used in production networks

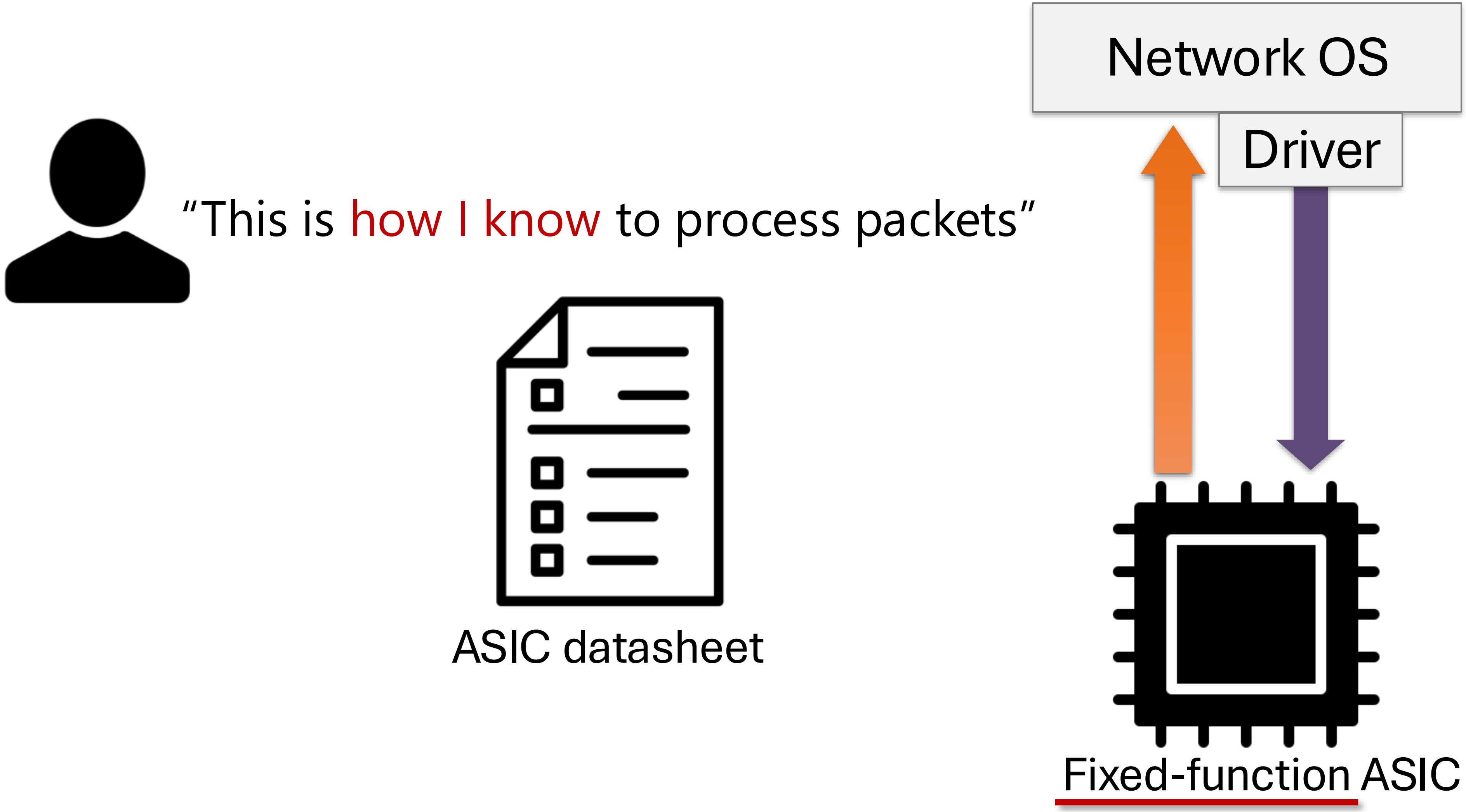
- E.g., Google and Microsoft’s Software-defined WAN

OpenFlow is an instance of protocol enabling the SDN approach

- It’s not actively developed today

Network operators (e.g., Google, MSFT) use their own protocols

Programmable *control plane* is not enough



Can we make data plane programmable?

Traditionally: Switching chip provides fixed function/size/number of match-action tables

➔Highly optimized to provide line-rate processing speed

Q: how can we make the switching chip flexible while not sacrificing its performance?

Q: How can we map a switch program to a programmable switch pipeline?

Technology Advance: Programmable *data plane*

P4: Programming Protocol-Independent Packet Processors

Pat Bosshart[†], Dan Daly[*], Glen Gibb[†], Martin Izzard[†], Nick McKeown[‡], Jennife
Cole Schlesinger[**], Dan Talayco[†], Amin Vahdat[¶], George Varghese[§], David

← ***P4 language - ACM SIGCOMM CCR (2014)***

Forwarding Metamorphosis: Fast Programmable Match-Action Processing in Hardware for SDN

Pat Bosshart[†], Glen Gibb[‡], Hun-Seok Kim[†], George Varghese[§], Nick McKeown[‡], Martin Izzard[†], Fernando Mujica[†], Mark Horowitz[‡]

RMT architecture - ACM SIGCOMM (2013) →

Realization in industry

Barefoot Networks unveils high-speed, programmable network switch

d with its g

Increased programmability brings more options to networks

Arista's 7170 programmable network switch with Barefoot
customers to deploy one system in multiple ways.

INDUSTRIAL AUTOMATION

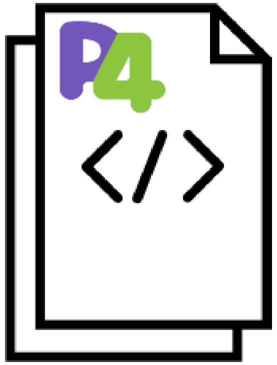
Broadcom Opens Up Programming for New Switch Chip

News

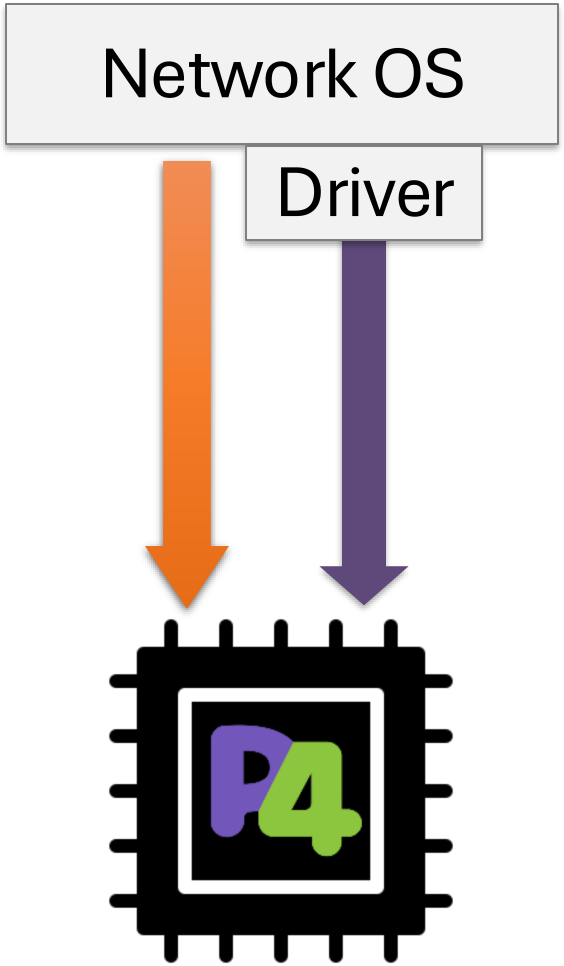
Cisco Rolls Out New Silicon, Router for 'Internet for the Future'

“Top-down” design with programmable data plane

“This is how you must process packets!”



User-defined data plane program



Programmable data plane ASIC

Benefits of data plane programmability

- New Features – Add new protocols
- Reduce complexity – Remove unused protocols
- Efficient use of resources – flexible use of tables
- Greater visibility – New diagnostic techniques, telemetry, etc.
- SW style development – rapid design cycle, fast innovation, fix data plane bugs in the field

Use case: Accelerating network functions

- Moving virtualized network functions to programmable switches
  - Similar cost compared to traditional networking devices.
  - Cheap per-Gbps cost.
  - High-performance.

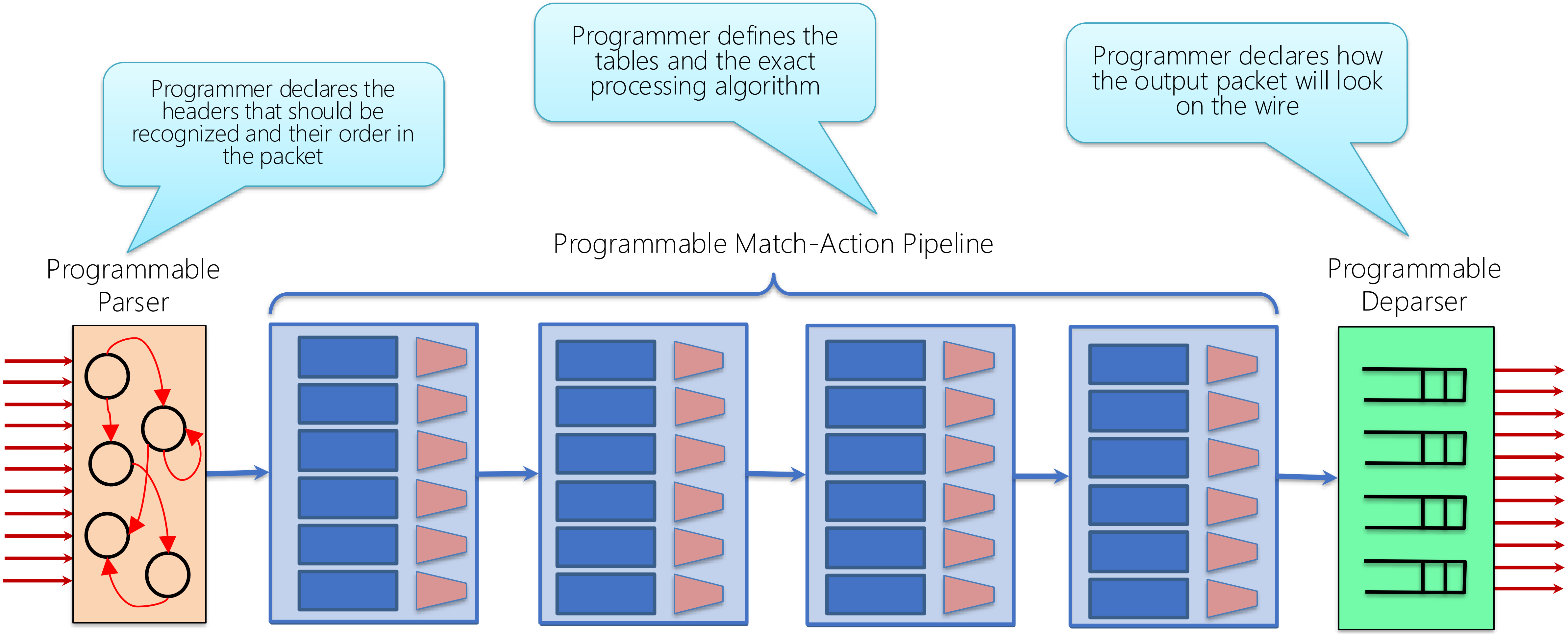
- 5G packet Core
  - Accelerating user-plane functions by running them on switches.
- Cloud-scale data center
  - Accelerating cloud-scale load balancer, firewall, gateway functions, etc.

More innovative use cases

- Low Latency Congestion Control – NDP[1]
- In-band Network Telemetry – INT[2]
- In-Network caching and coordination – NetCache[3] / NetChain[4]
- In-Network consensus protocol[5]
- In-Network parameter server for distributed ML – SwitchML [6]
- … and many more

[1] Handley, Mark, et al. "Re-architecting datacenter networks and stacks for low latency and high performance." SIGCOMM, 2017.
[2] Kim, Changhoon, et al. "In-band network telemetry via programmable dataplanes." SIGCOMM. 2015.
[3] Xin Jin et al. "NetCache: Balancing Key-Value Stores with Fast In-Network Caching." To appear at SOSP 2017
[4] Jin, Xin, et al. "NetChain: Scale-Free Sub-RTT Coordination." *NSDI*, 2018.
[5] Dang, Huynh Tu, et al. "NetPaxos: Consensus at network speed." SIGCOMM, 2015.
[6] Sapio, Amedeo, et al. "Scaling Distributed Machine Learning with In-Network Aggregation." *Arxiv, 2019.*

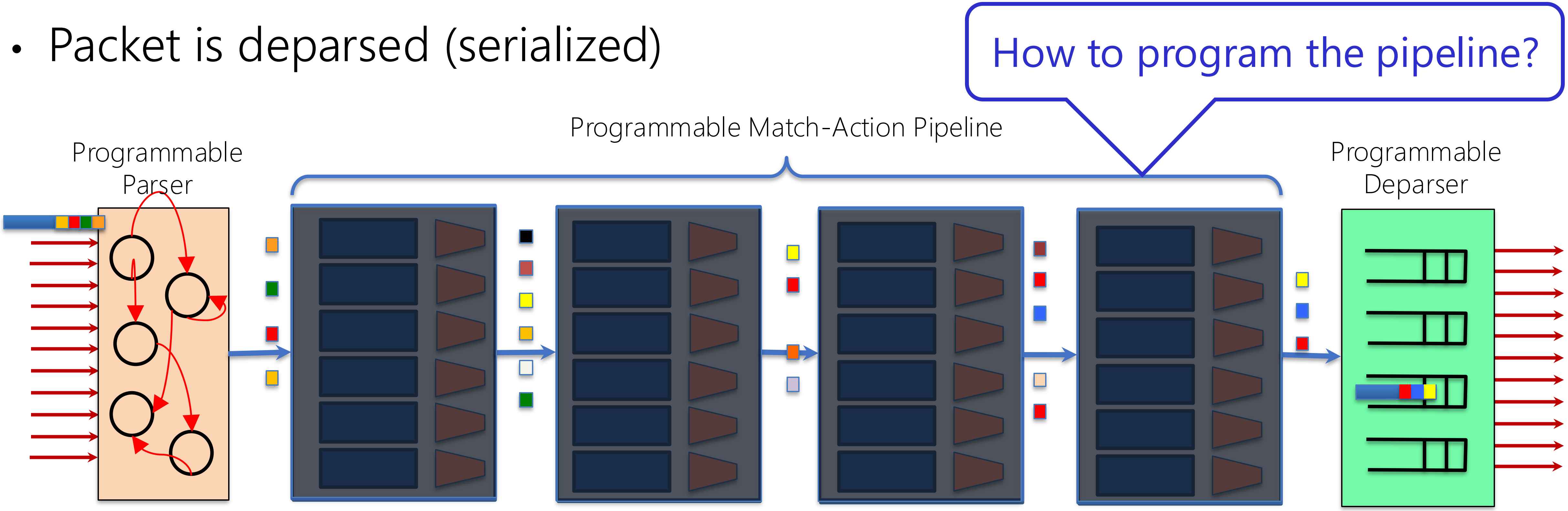
PISA: Protocol-Independent Switch Architecture



*Slides from P4 tutorial (https://github.com/p4lang/tutorials/blob/master/P4_tutorial.pdf)

PISA in action

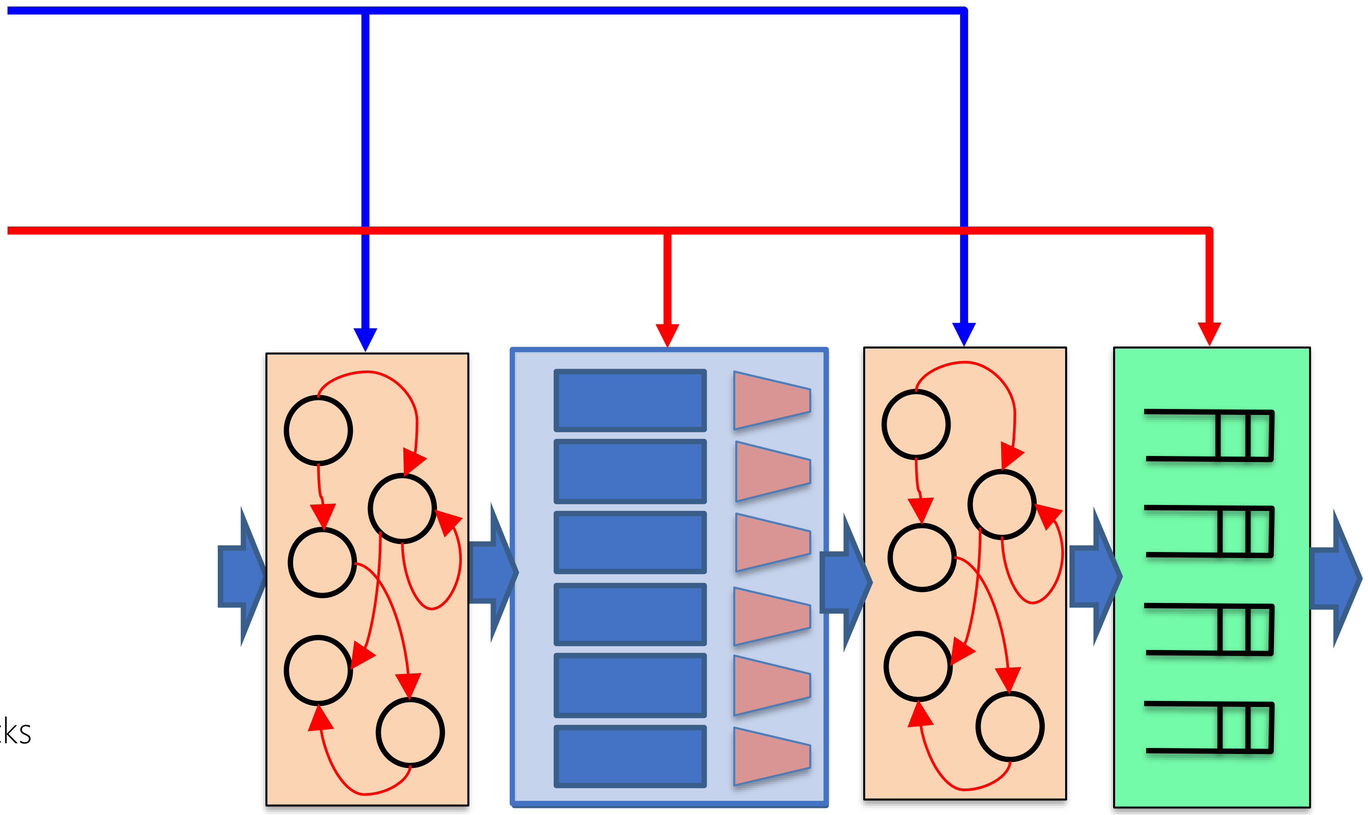
- Packet is parsed into individual headers (parsed representation)
- Headers and intermediate results can be used for matching and actions
- Headers can be modified, added or removed
- Packet is departed (serialized)



*Slides from P4 tutorial (https://github.com/p4lang/tutorials/blob/master/P4_tutorial.pdf)

$P4_{16}$ language elements

Parsers	State machine, bitfield extraction
Controls	Tables, Actions, control flow statements
Expressions	Basic operations and operators
Data Types	Bistrings, headers, structures, arrays
Architecture Description	Programmable blocks and their interfaces
Extern Libraries	Support for specialized components



*Slides from P4 tutorial (https://github.com/p4lang/tutorials/blob/master/P4_tutorial.pdf)

P4$_{16}$ Program Template (V1Model)

```
#include <core.p4>
#include <v1model.p4>
/* HEADERS */
struct metadata { ... }
struct headers {
  ethernet_t    ethernet;
  ipv4_t        ipv4;
}
/* PARSER */
parser MyParser(packet_in packet,
             out headers hdr,
             inout metadata meta,
             inout standard_metadata_t smeta) {
  ...
}
/* CHECKSUM VERIFICATION */
control MyVerifyChecksum(in headers hdr,
                         inout metadata meta) {

   ...
}
/* INGRESS PROCESSING */
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t std_meta) {
  ...
}
```

```
/* EGRESS PROCESSING */
control MyEgress(inout headers hdr,
                 inout metadata meta,
                 inout standard_metadata_t std_meta) {
  ...
}
/* CHECKSUM UPDATE */
control MyComputeChecksum(inout headers hdr,
                          inout metadata meta) {
  ...
}
/* DEPARSER */
control MyDeparser(inout headers hdr,
                   inout metadata meta) {
  ...
}
/* SWITCH */
V1Switch(
  MyParser(),
  MyVerifyChecksum(),
  MyIngress(),
  MyEgress(),
  MyComputeChecksum(),
  MyDeparser()
) main;
```

*Slides from P4 tutorial (https://github.com/p4lang/tutorials/blob/master/P4_tutorial.pdf)

$P4_{16}$ Hello World (V1Model)

```
#include <core.p4>
#include <v1model.p4>
struct metadata {}
struct headers {}

parser MyParser(packet_in packet,
   out headers hdr,
   inout metadata meta,
   inout standard_metadata_t standard_metadata) {

    state start { transition accept; }
}

control MyVerifyChecksum(inout headers hdr, inout
metadata meta) {   apply {  }   }

control MyIngress(inout headers hdr,
   inout metadata meta,
   inout standard_metadata_t standard_metadata) {
apply {
        if (standard_metadata.ingress_port == 1) {
            standard_metadata.egress_spec = 2;
        } else if (standard_metadata.ingress_port == 2) {
            standard_metadata.egress_spec = 1;
        }
    }
}
```

```
control MyEgress(inout headers hdr,
   inout metadata meta,
   inout standard_metadata_t standard_metadata) {
    apply {  }
}

control MyComputeChecksum(inout headers hdr, inout metadata
meta) {
     apply { }
}

control MyDeparser(packet_out packet, in headers hdr) {
    apply { }
}

V1Switch(
   MyParser(),
   MyVerifyChecksum(),
   MyIngress(),
   MyEgress(),
   MyComputeChecksum(),
   MyDeparser()
) main;
```

*Slides from P4 tutorial (https://github.com/p4lang/tutorials/blob/master/P4_tutorial.pdf)

$P4_{16}$ Hello World (V1Model)

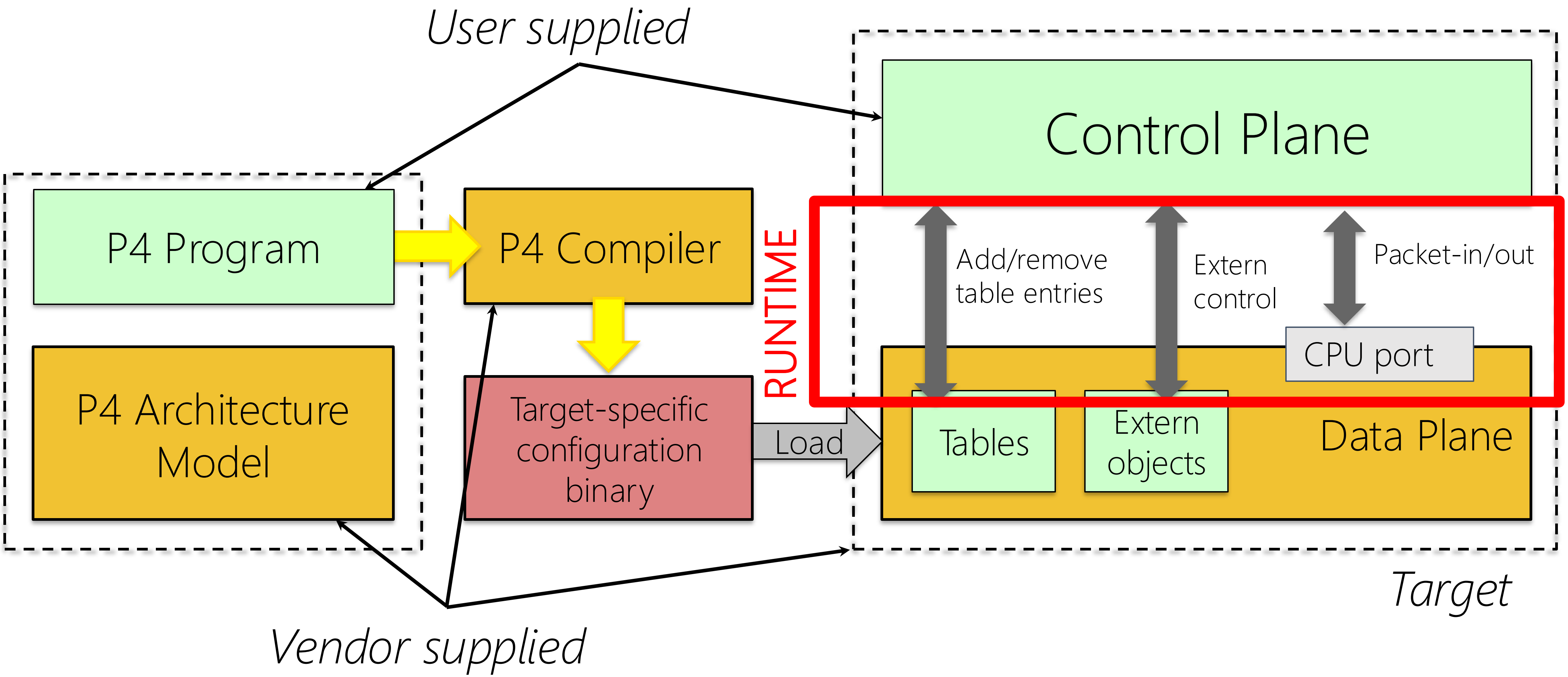
```
#include <core.p4>
#include <v1model.p4>
struct metadata {}
struct headers {}

parser MyParser(packet_in packet, out headers hdr,
   inout metadata meta,
   inout standard_metadata_t standard_metadata) {
    state start { transition accept; }
}

control MyIngress(inout headers hdr, inout metadata meta,
   inout standard_metadata_t standard_metadata) {
    action set_egress_spec(bit<9> port) {
        standard_metadata.egress_spec = port;
    }
    table forward {
        key = { standard_metadata.ingress_port: exact; }
        actions = {
            set_egress_spec;
            NoAction;
        }
        size = 1024;
        default_action = NoAction();
    }
    apply {   forward.apply();   }
}
```

```
control MyEgress(inout headers hdr,
   inout metadata meta,
   inout standard_metadata_t standard_metadata) {
    apply {   }
}

control MyVerifyChecksum(inout headers hdr, inout metadata
meta) {   apply { }   }

control MyComputeChecksum(inout headers hdr, inout metadata
meta) {   apply { }   }

control MyDeparser(packet_out packet, in headers hdr) {
    apply { }
}

V1Switch( MyParser(), MyVerifyChecksum(), MyIngress(),
MyEgress(), MyComputeChecksum(), MyDeparser() ) main;
```

Key	Action ID	Action Data
1	set_egress_spec ID	2
2	set_egress_spec ID	1

*Slides from P4 tutorial (https://github.com/p4lang/tutorials/blob/master/P4_tutorial.pdf)

Programming a P4 target



*Slides from P4 tutorial (https://github.com/p4lang/tutorials/blob/master/P4_tutorial.pdf)

Various P4 targets

Programmable switches

- E.g., Intel’s Tofino chip-based switches

Programmable network interface cards (NICs)

- E.g., AMD’s Alveo FPGA-based NICs

Software dataplanes running on CPUs

- E.g., eBPF programs running inside the Linux kernel

Summary

Traditionally, networks are hard to be managed due to tight coupling between the control and data plane

SDN decouples of control and data plane

- Centralized controller controls the network with network-wide view
- OpenFlow: a protocol between the controller to individual devices

Programmable data plane further improves flexibility

- Data plane logic can be programmable with P4
- Many innovative use cases beyond packet processing