

Programming The Network Data Plane In P4



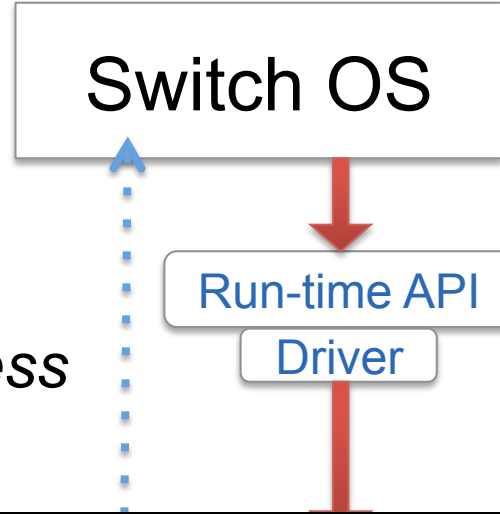
Changhoon Kim

chang@barefootnetworks.com

Status quo



“This is roughly how I process packets ...”



- **Prone to bugs**
- **Very long and unpredictable lead time**

Fixed-function ASIC

Extremely limited way of turning “beautiful ideas” into reality

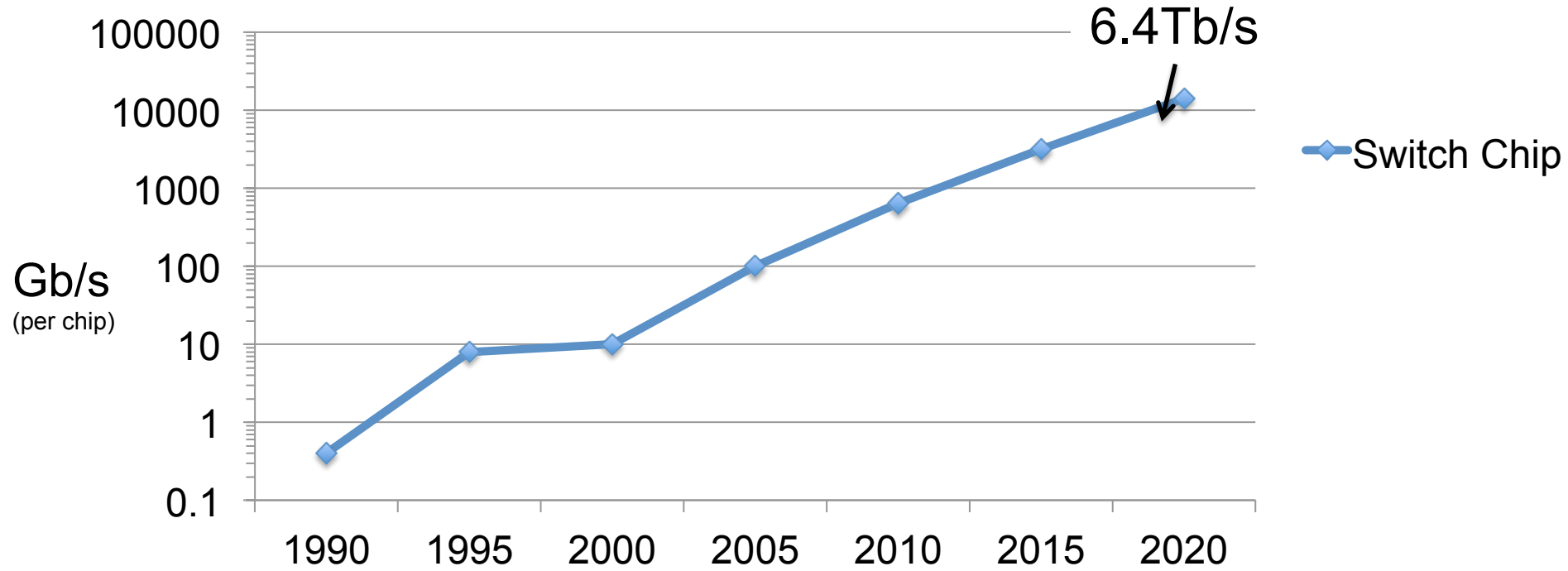
- No DIY – must work with vendors at feature level
- Excruciatingly complicated and involved process to build consensus and pressure for *features*
- Painfully long and unpredictable lead time
- To use new features, you must get new switches
- What you finally get != what you asked for



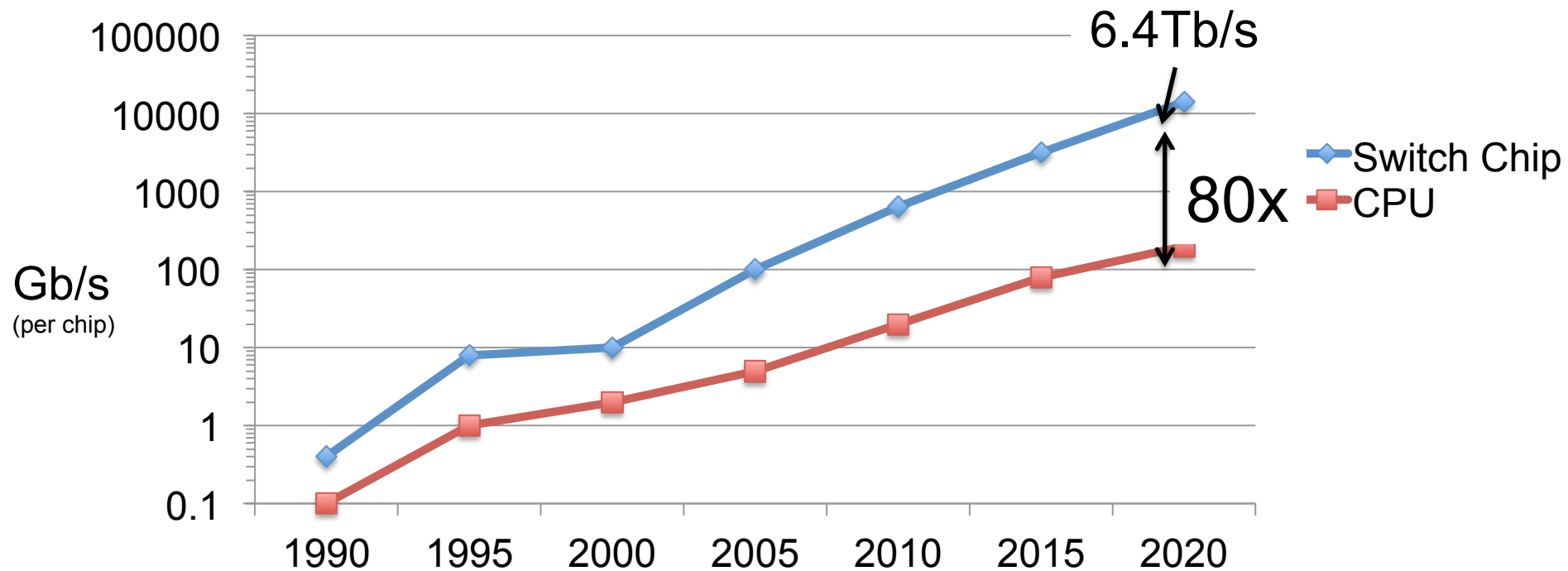
Programmable network devices come to rescue

- CPUs: 10s of Gb/s
- FPGAs, NPUs: 100s of Gb/s
- **Protocol-Independent Switch Architecture (PISA) chips: a few Tb/s**
 - Initial architecture introduced in RMT [SIGCOMM'13]
 - Packet processing machine with fully programmable parser and generic match-action logic
 - No penalty in size, cost, and power compared to fixed-function ASICs
 - PISA products available now -- in next few years this kind of silicon will dominate

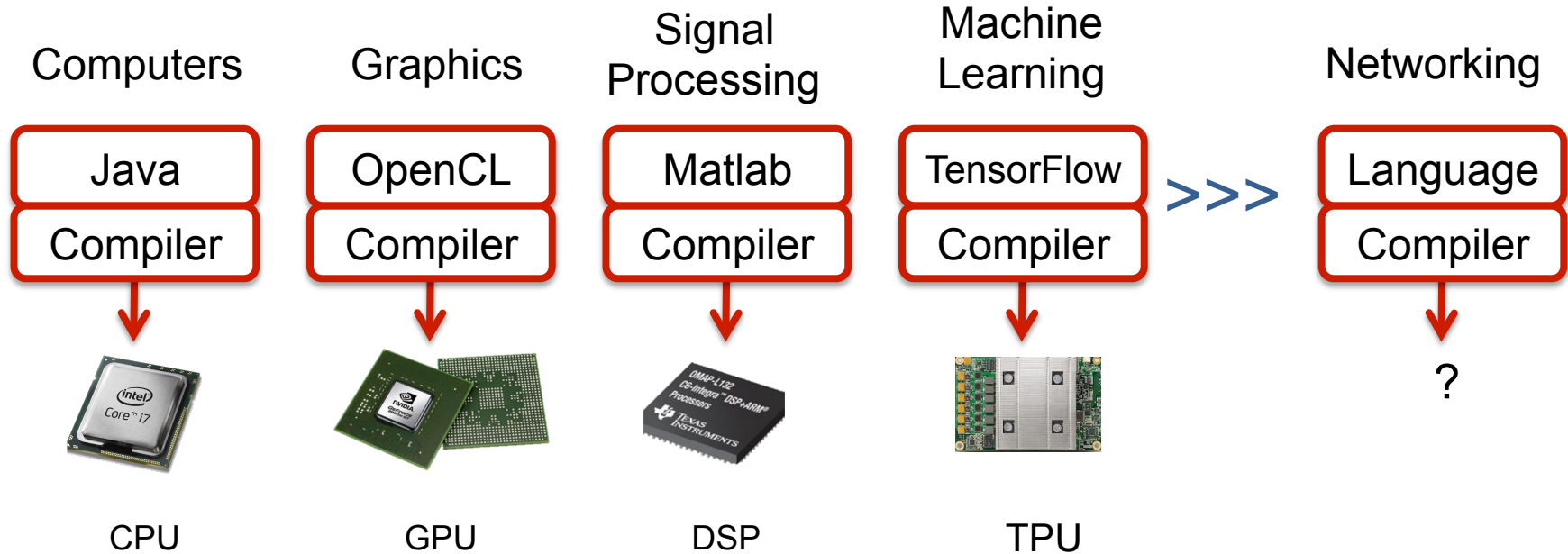
Packet forwarding speeds



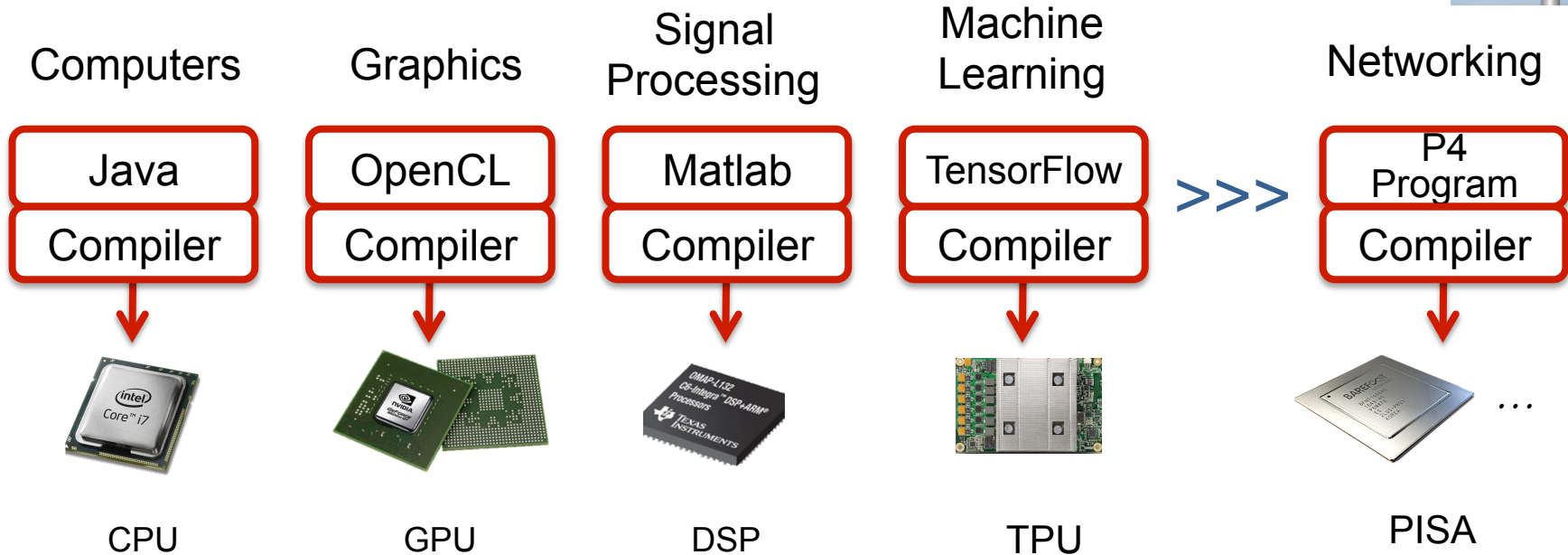
Packet forwarding speeds



Domain-specific processors



Domain-specific processors



P4: Programming Protocol-Independent Packet Processors

Pat Bosshart¹, Dan Daly², Glen Gibb³, Martin Izzard⁴, Nick McKeown⁵, Jennifer Rexford⁶, Cole Schlesinger⁷, Dan Talayco⁸, Amin Vahdat⁹, George Varghese¹⁰, David Walker¹¹, Barefoot Networks¹, Intel², Stanford University³, Princeton University⁴, Google⁵, Microsoft Research⁶

ABSTRACT

P4 is a high-level language for programming protocol-independent packet processors. P4 works in conjunction with SDN control protocols like OpenFlow. In its current form, OpenFlow explicitly specifies protocol headers on which it operates. This set has grown from 12 to 41 fields in a few years, increasing the complexity of the specification while still not providing the flexibility to add new headers. In this paper we propose P4 as a strawman proposal for how OpenFlow should evolve in the future. We have three goals: (1) Reconfigurability in the field: Programmers should be able to change the way switches process packets once they are deployed. (2) Protocol independence: Switches should be tied to any specific network protocols. (3) Target independence: Programmers should be able to describe packet-processing functionality independently of the specifics of the underlying hardware. As an example, we describe how to use P4 to configure a switch to add a new hierarchical label.

1. INTRODUCTION

Software-Defined Networking (SDN) gives operators programmatic control over their networks. In SDN, the control plane is physically separate from the forwarding plane, and one control plane controls multiple forwarding devices. While forwarding devices could be programmed in many ways, having a common, open, vendor-agnostic interface (like OpenFlow) enables a control plane to control forwarding devices from different hardware and software vendors.

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 r...

multiple stages of rule tables, to allow switches to expose more of their capabilities to the controller. The proliferation of new header fields shows no signs of stopping. For example, data-center network operators increasingly want to apply new forms of packet encapsulation (e.g., NVGRE, VXLAN, and STT), for which they resort to deploying software switches that are easier to extend with new functionality. Rather than repeatedly extending the OpenFlow specification, we argue that future switches should support flexible mechanisms for parsing packets and matching header fields, allowing controller applications to leverage these capabilities through a common, open interface (i.e., a new “OpenFlow 2.0” API). Such a general, extensible approach would be simpler, more elegant, and more future-proof than today’s OpenFlow 1.x standard.

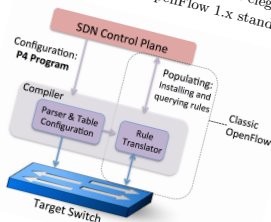
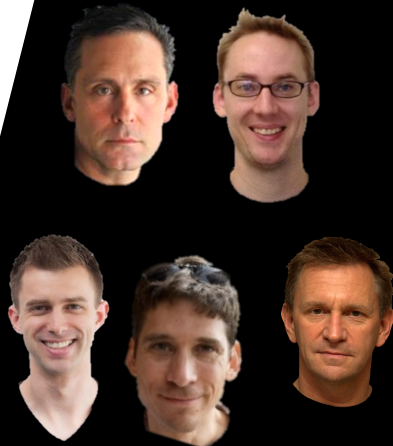
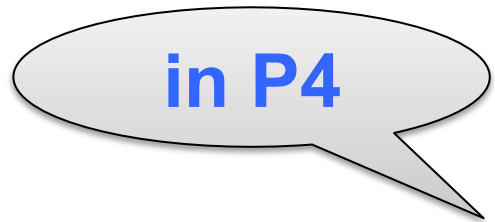


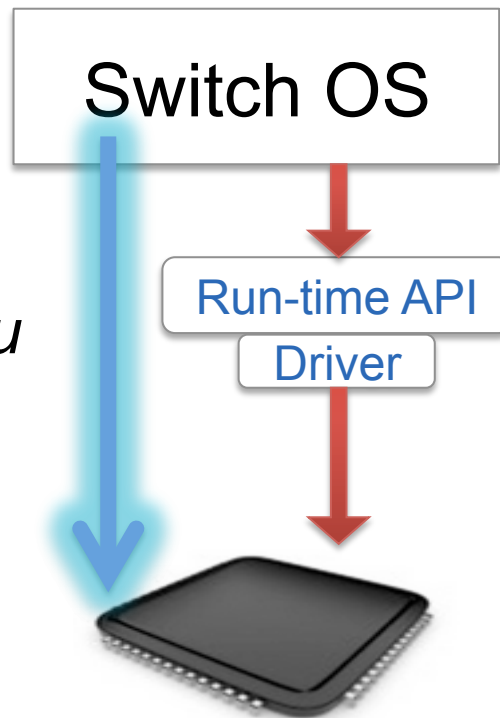
Figure 1: P4 is a language to...



Turning the tables “upside down”



“This is precisely how you must process packets”



PISA device

(Protocol-Independent Switch Architecture)

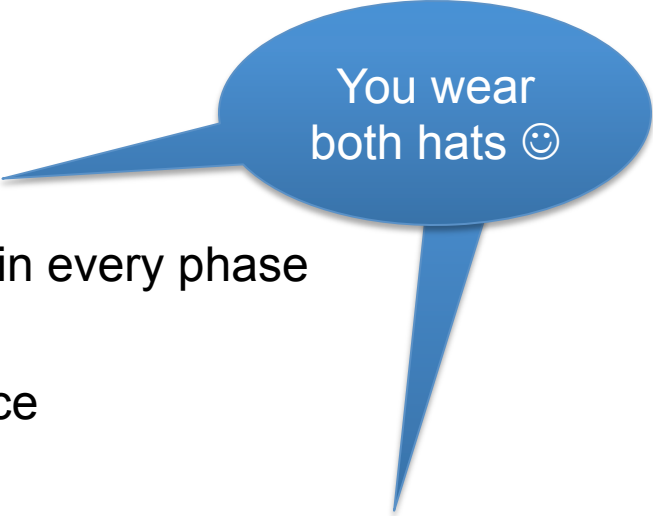
What does this mean?

■ To network device vendors

- S/W programming practices and tools used in every phase
- Extremely fast iteration and feature release
- Differentiation in capabilities and performance
- Can fix even data-plane bugs in the field

■ To large on-line service providers and carriers

- No more “black boxes” in the “white boxes”
- Your devs can program, test, and debug your network devices all the way down
- You keep your own ideas



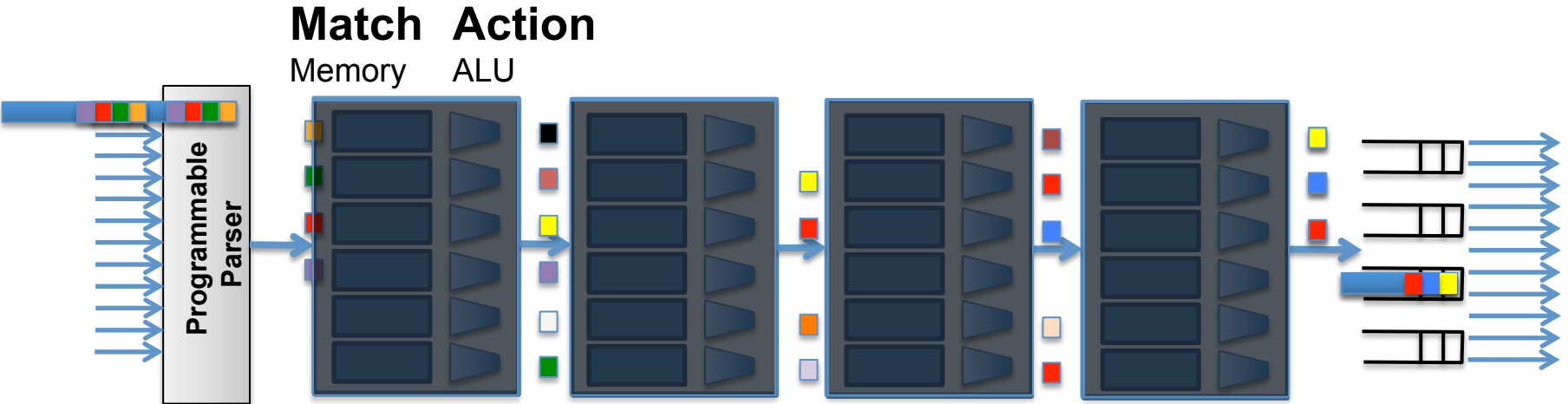
You wear
both hats 😊

The rest of the talk:

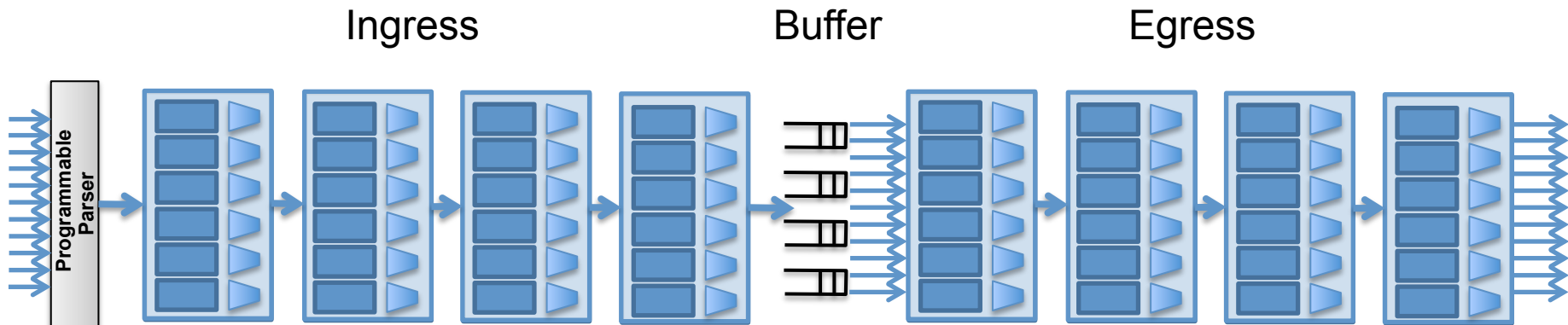
- *How PISA works*
- *Why we call it protocol-independent forwarding*
- *What kind of cool things you can do with PISA and P4*
- *Demo!*

PISA: An architecture for high-speed programmable packet forwarding

PISA: Protocol Independent Switch Architecture



PISA: Protocol Independent Switch Architecture



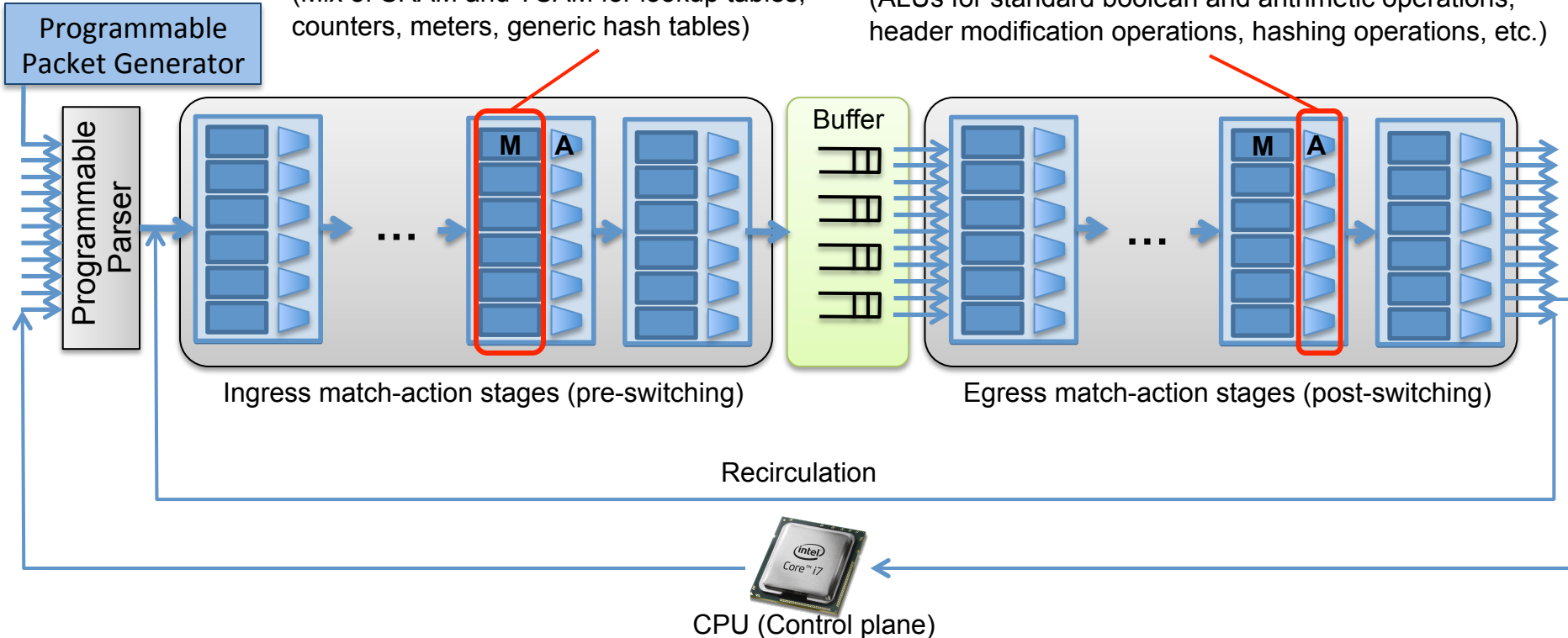
PISA: Protocol Independent Switch Architecture

Match Logic

(Mix of SRAM and TCAM for lookup tables, counters, meters, generic hash tables)

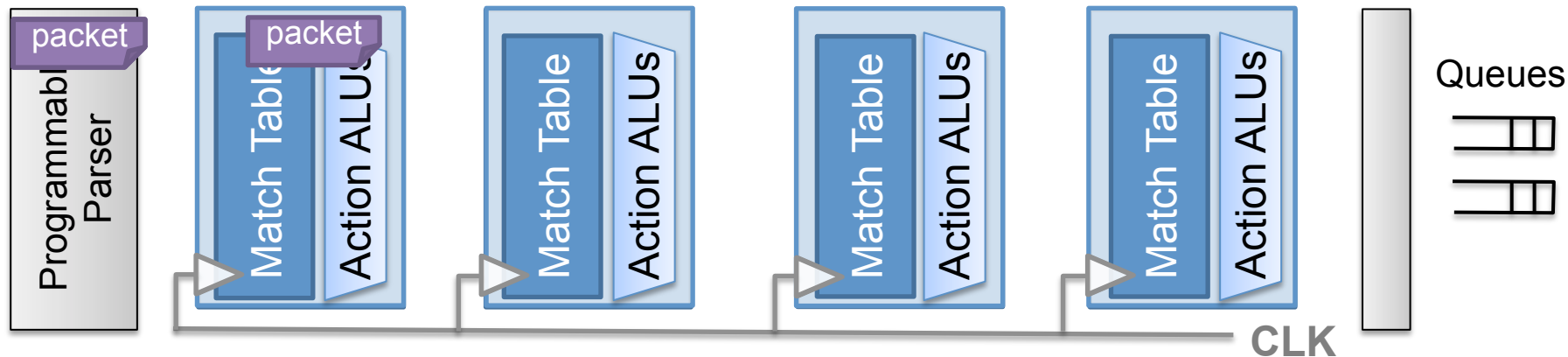
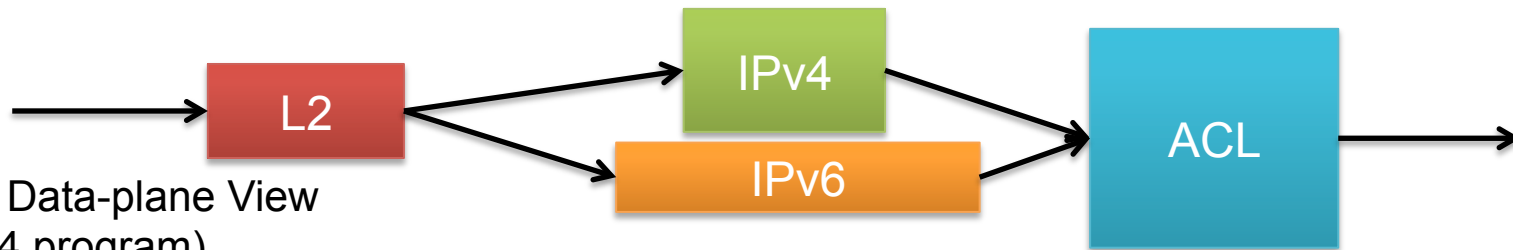
Action Logic

(ALUs for standard boolean and arithmetic operations, header modification operations, hashing operations, etc.)

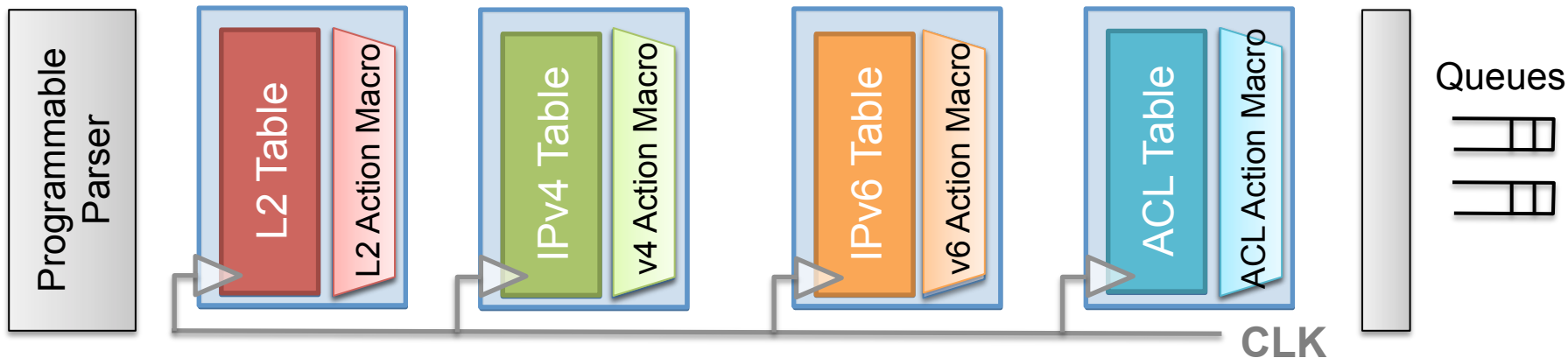
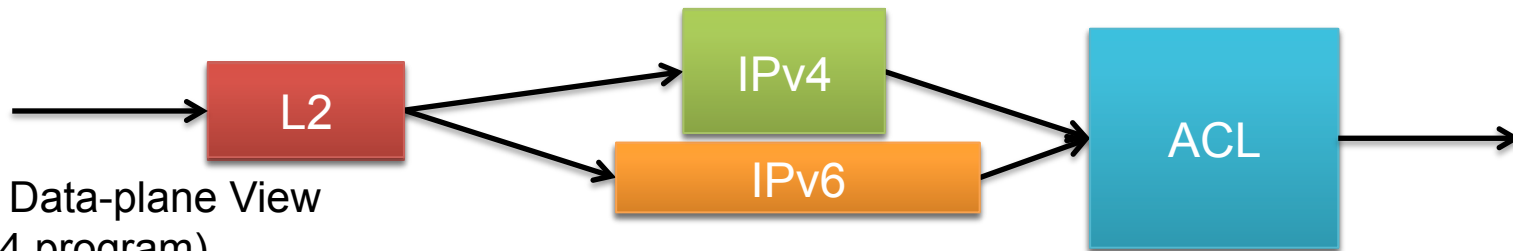


**Why we call it
protocol-independent packet
processing**

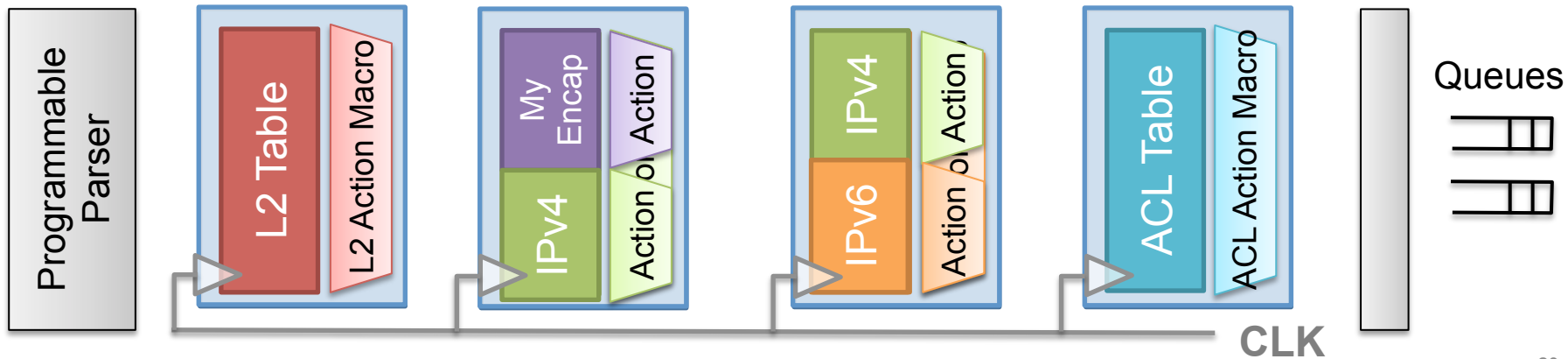
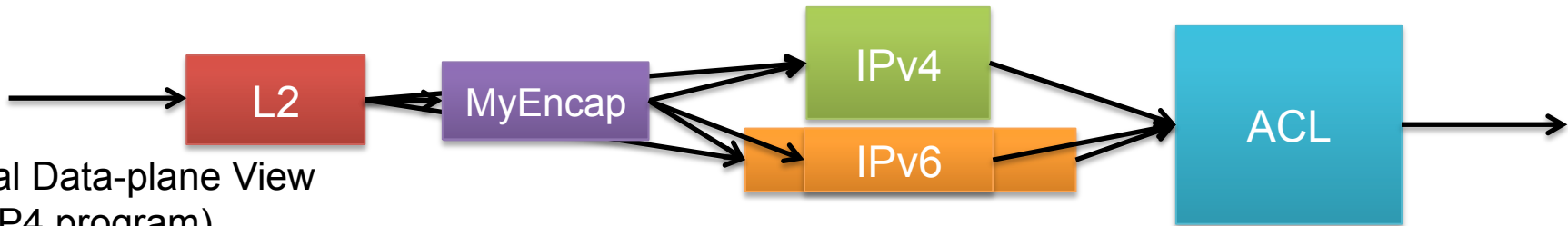
Device does not understand any protocols until it gets programmed



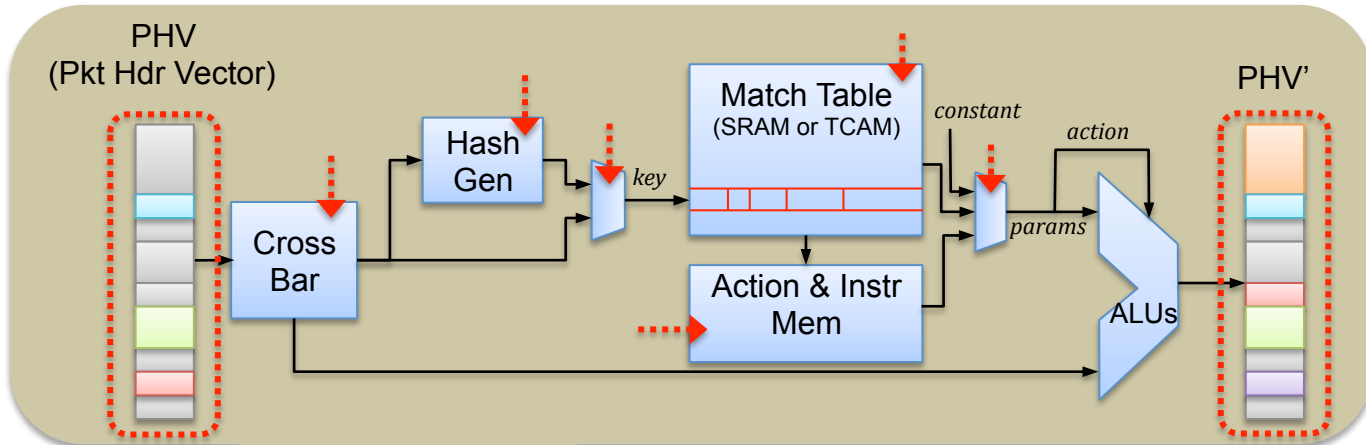
Mapping logical data-plane design to physical resources



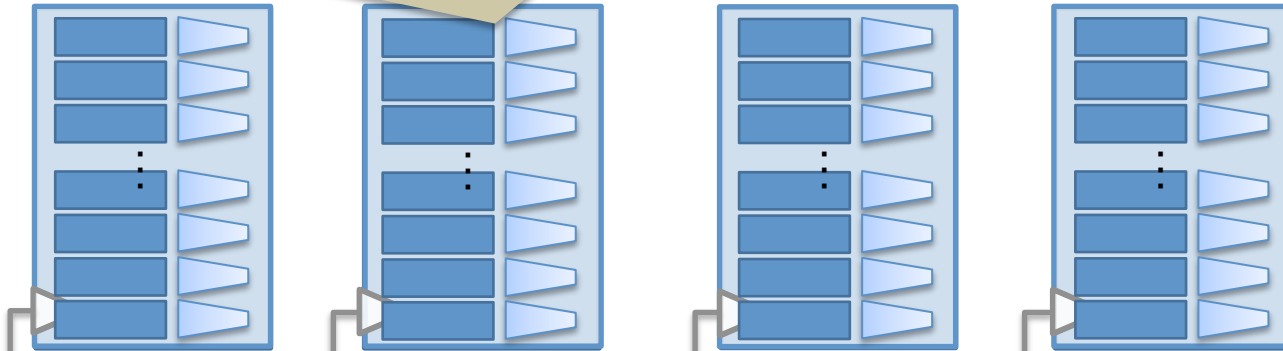
Re-program in the field



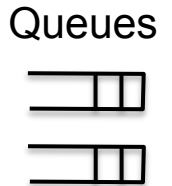
What exactly does the compiler do?



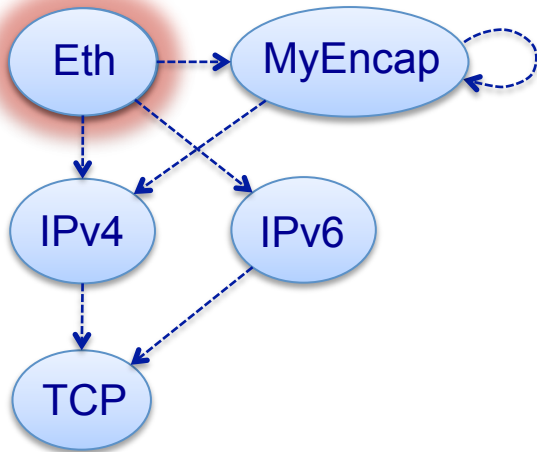
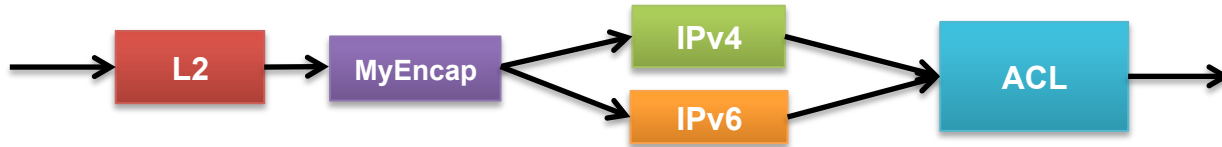
Programmable
Parser



Queues



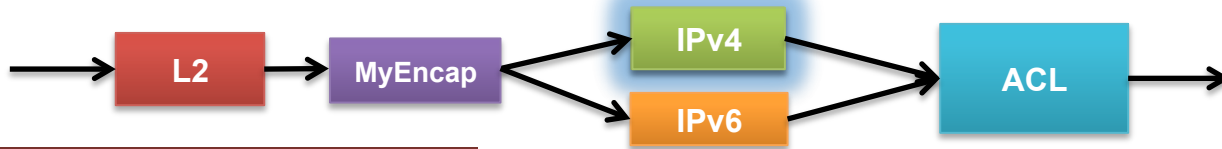
What does a P4 program look like?



```
header_type ethernet_t {  
  fields {  
    dstAddr : 48;  
    srcAddr : 48;  
  }  
  
  parser parse_ethernet {  
    extract(ethernet);  
    return select(latest.etherType) {  
      0x8100 : parse_vlan;  
    }  
  }  
}
```

```
header_type my_encap_t {  
  fields {  
    foo : 12;  
    bar : 8;  
    baz : 4;  
    qux : 4;  
    next_protocol : 4;  
  }  
}
```

What does a P4 program look like?



```
table ipv4_lpm
{
  reads {
    ipv4.dstAddr
  }
  actions {
    set_next_hop;
  }
}
```

```
control ingress
{
  apply(l2);
  apply(my_encap);
  if (valid(ipv4) {
    apply(ipv4_lpm);
  } else {
    apply(ipv6_lpm);
  }
  apply(acl);
}
```

```
action set_next_hop(nhop_ip)
{
  modify_field(metadata.nhop_ipv4_addr, nhop_ipv4_addr);
  modify_field(standard_metadata.egress_port, port);
  add_to_field(ipv4.ttl, -1);
}
```

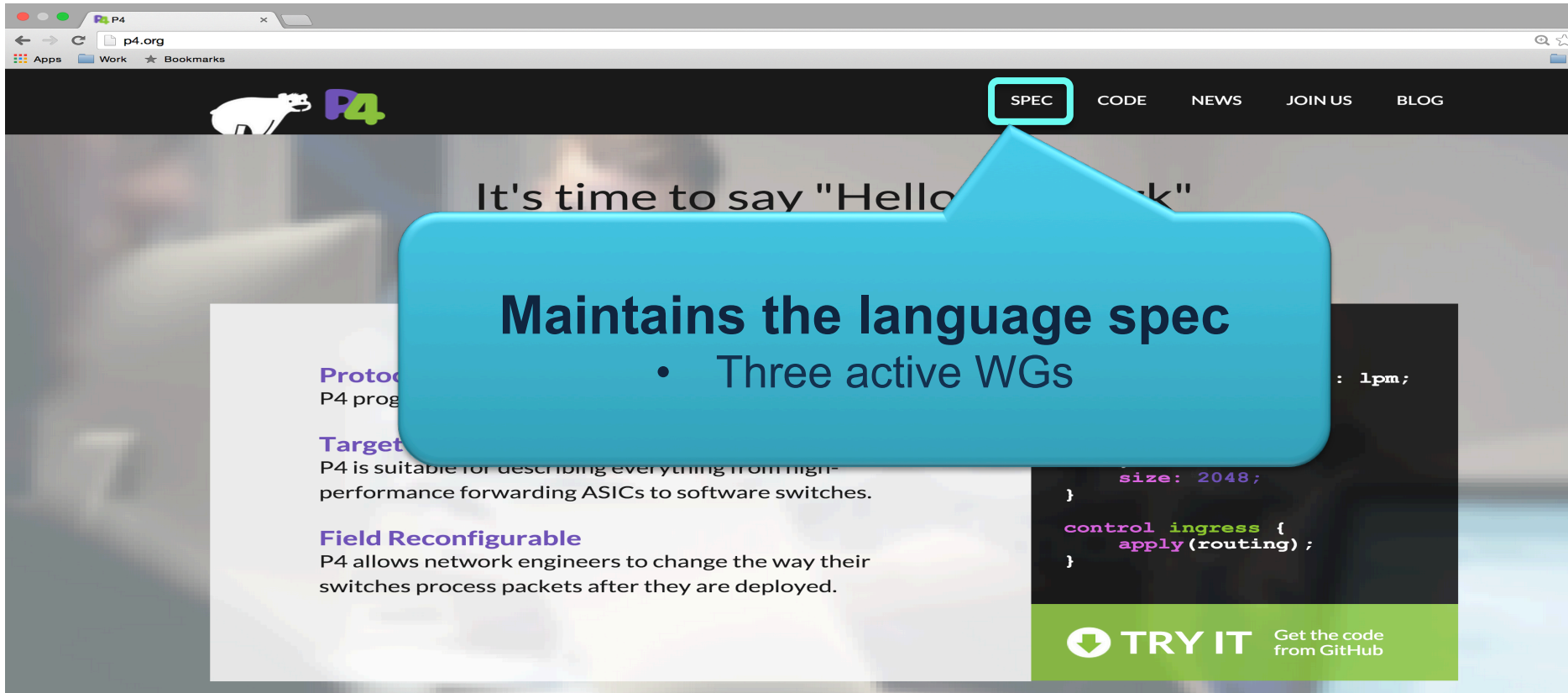
What does a P4 program look like?

```
/* Example: A typical IPv4 routing table */
table ipv4_lpm {
  reads {
    ingress_metadata.vrf      : exact;
    ipv4.dstAddr              : lpm;
  }
  actions {
    nop;
    l3_l2_switch;
    l3_multicast;
    l3_nexthop;
    l3_ecmp;
    l3_drop;
  }
  size : 65536;
}
```

These are the only possible actions.
Each particular entry in the table is
associated with ONE of these

vrf	ipv4.dstAddr / prefix	action	data
1	192.168.1.0 / 24	l3_l2_switch	port_id=64
10	10.0.16.0 / 22	l3_ecmp	ecmp_index=12
1	192.168.0.0 / 16	l3_nexthop	nexthop_index=451
1	0.0.0.0 / 0	l3_nexthop	nexthop_index=1

P4.org – P4 Language Consortium



The image shows a screenshot of the P4.org website. The browser's address bar shows 'p4.org'. The navigation menu includes 'SPEC', 'CODE', 'NEWS', 'JOIN US', and 'BLOG'. A blue callout box points to the 'SPEC' link and contains the text 'Maintains the language spec' and a bullet point '• Three active WGs'. The main content area features a headline 'It's time to say "Hello P4"', a 'Protocol' section, a 'Target' section, and a 'Field Reconfigurable' section. A code snippet is visible on the right side of the page.

It's time to say "Hello P4"

Protocol
P4 program

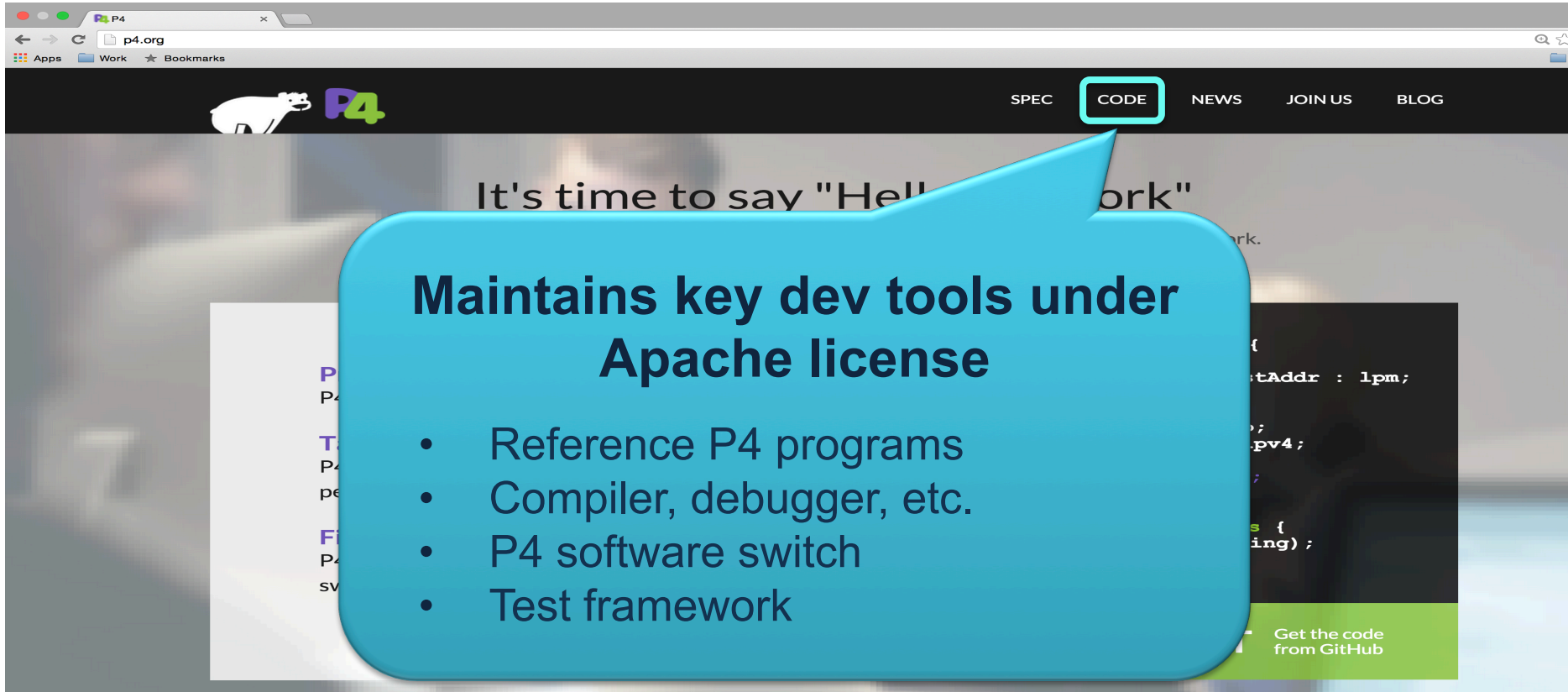
Target
P4 is suitable for describing everything from high-performance forwarding ASICs to software switches.

Field Reconfigurable
P4 allows network engineers to change the way their switches process packets after they are deployed.

```
size: 2048;
}
control ingress {
  apply(routing);
}
```

TRY IT Get the code from GitHub

P4.org – P4 Language Consortium



The image shows a screenshot of the P4.org website. The browser's address bar displays 'p4.org'. The navigation menu includes links for 'SPEC', 'CODE', 'NEWS', 'JOIN US', and 'BLOG'. The 'CODE' link is highlighted with a red box. A blue callout box is overlaid on the page, containing the text 'Maintains key dev tools under Apache license' and a bulleted list of tools. The background of the website shows a header with a cow logo and the text 'It's time to say "Hello P4"'. A code snippet is visible on the right side of the page.

CODE

It's time to say "Hello P4"

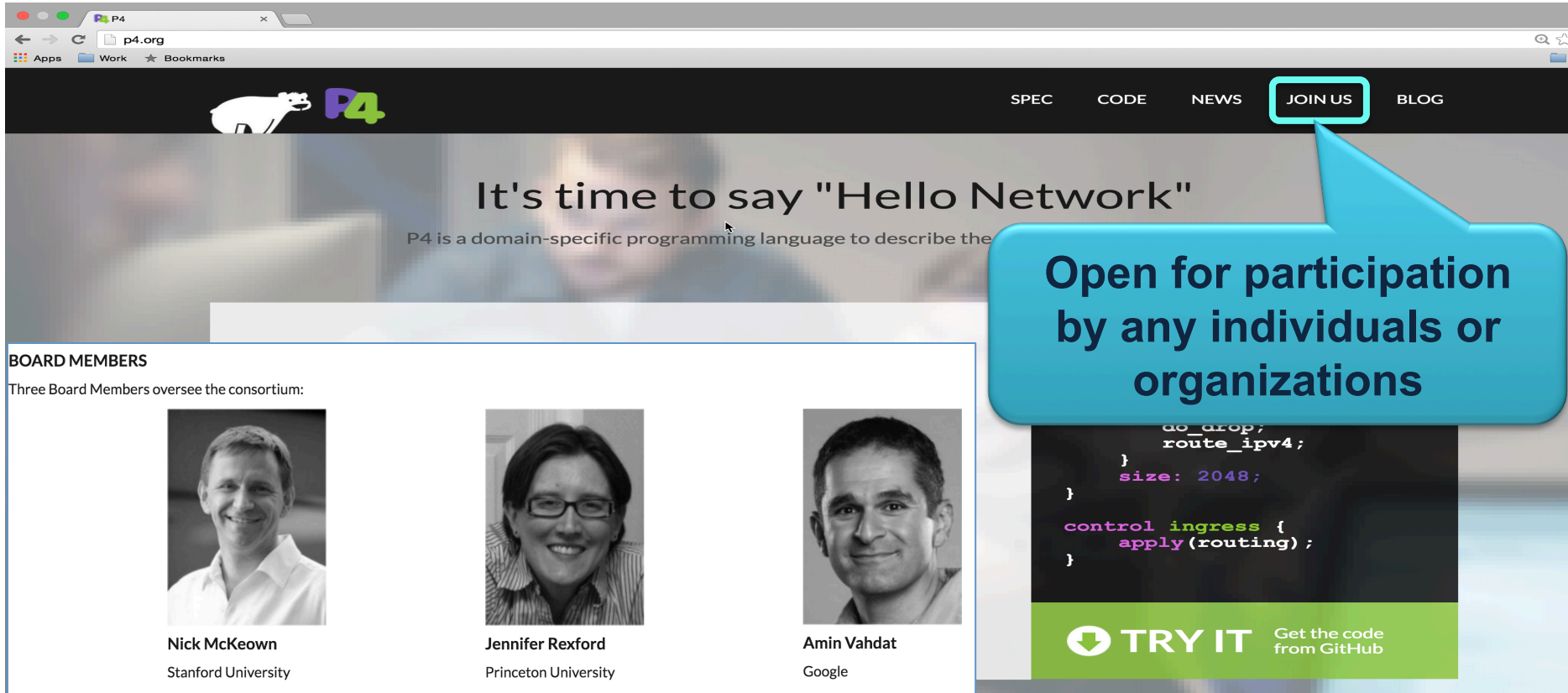
Maintains key dev tools under Apache license

- Reference P4 programs
- Compiler, debugger, etc.
- P4 software switch
- Test framework

```
stAddr : lpm;  
;  
pv4;  
;  
s {  
ing);
```

Get the code from GitHub

P4.org – P4 Language Consortium



The screenshot shows the P4.org website. The navigation bar includes links for SPEC, CODE, NEWS, JOIN US (highlighted with a blue callout box), and BLOG. The main heading reads "It's time to say 'Hello Network'" followed by the subtext "P4 is a domain-specific programming language to describe the". Below this, a section titled "BOARD MEMBERS" lists three individuals: Nick McKeown (Stanford University), Jennifer Rexford (Princeton University), and Amin Vahdat (Google). To the right, there is a code snippet for a network configuration and a "TRY IT" button with the text "Get the code from GitHub".


JOIN US

It's time to say "Hello Network"


P4 is a domain-specific programming language to describe the

BOARD MEMBERS


Three Board Members oversee the consortium:



Nick McKeown
Stanford University



Jennifer Rexford
Princeton University



Amin Vahdat
Google

```
do_drop;
route_ipv4;
}
size: 2048;
}
control ingress {
  apply(routing);
}
```

TRY IT Get the code from GitHub

Open for participation by any individuals or organizations



P4.org Members

Original P4 Paper Authors:



Operators/
End Users



Systems



Targets



Solutions/
Services



Academia/
Research



P4 development environment

- **Open-source P4 development tools**
 - P4 compilers & dev tools, reference P4 programs, P4-programmable S/W switch, test framework, etc.
 - Apache 2.0 license
 - Available at <http://github.com/p4lang>
- **Several other programmable forwarding targets**
 - Both hardware and software devices (OVS, eBPF, VPP etc.)
 - Switches, NICs, etc.

What kind of cool things can you do by programming data planes?

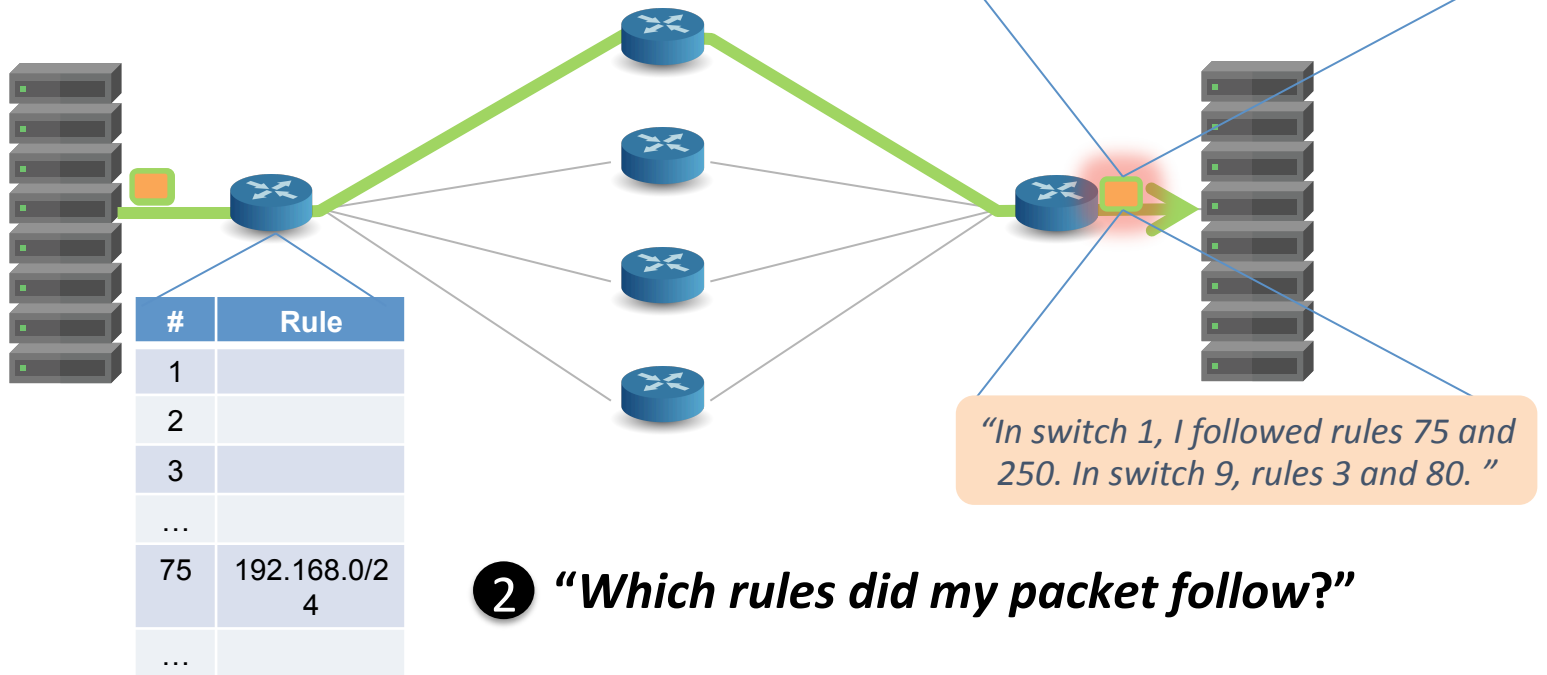
- **Advanced network monitoring, analysis, and diagnostics**
- **Custom traffic monitoring and filtering**
 - FlowRadar [NSDI'16]
- **Various modes of congestion control**
 - RCP, XCP, TeXCP, DCQCN++, Timely++
- **Dynamic source routing**
 - Flowlet switching, CONGA [SIGCOMM'15], HULA [SOSR'16]
- **Embedding middlebox functions into switches**
 - In-switch L4 connection load balancing, TCP SYN authentication, etc.
- **Offloading parts of the distributed apps**
 - SwitchPaxos [SOSR'15, ACM CCR'16] , SwitchKV [NSDI'16]
- **Jointly optimizing network and the apps running on it**
- **And many more ... -- we're just starting to scratch the surface!**

Very natural questions

*“Is every packet delivered correctly and timely?
If not, when and why? If yes, how well?”*

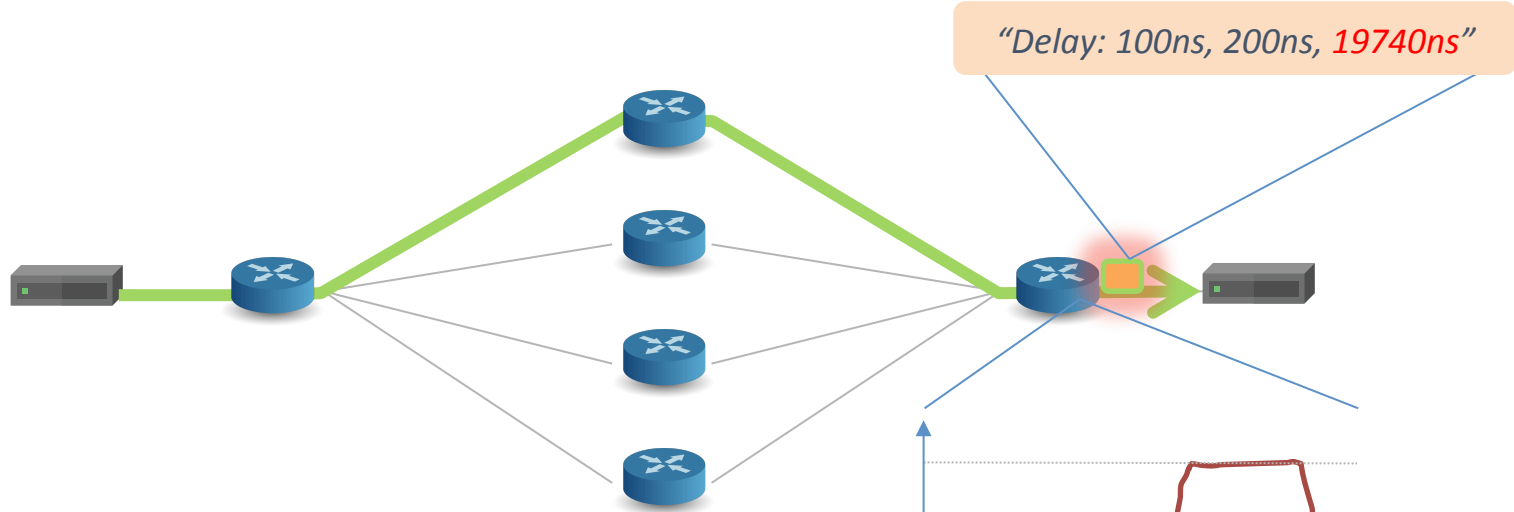
No switches today can answer these basic questions.

1 "Which path did my packet take?"

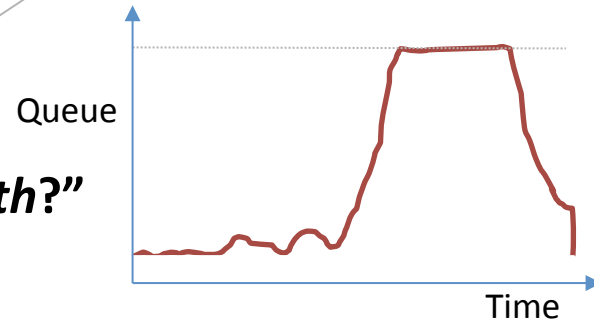


2 "Which rules did my packet follow?"

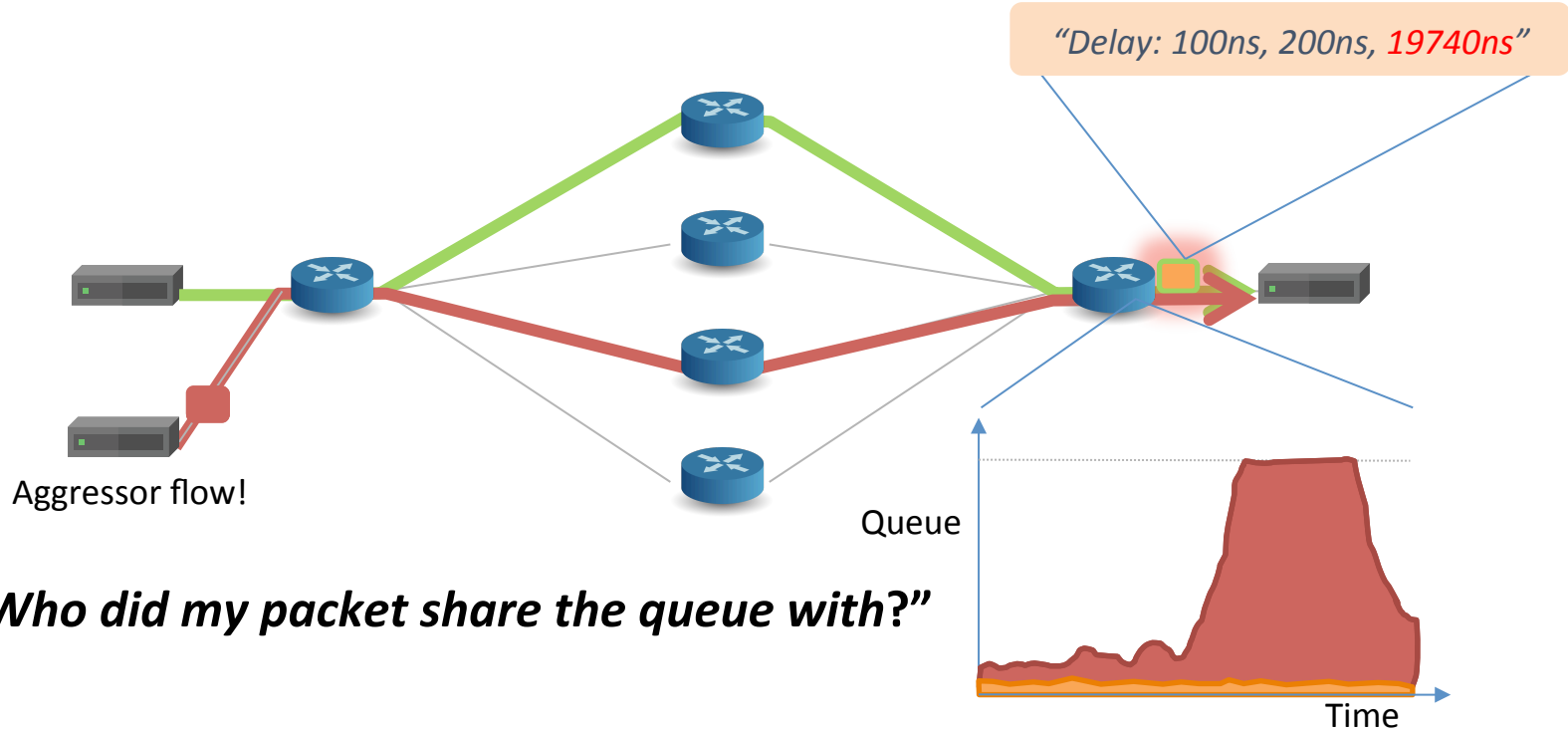
3 *“How long did my packet queue at each switch?”*



4 *“Who did my packet share the queue with?”*



3 "How long did my packet queue at each switch?"



4 "Who did my packet share the queue with?"

The network should 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?”*



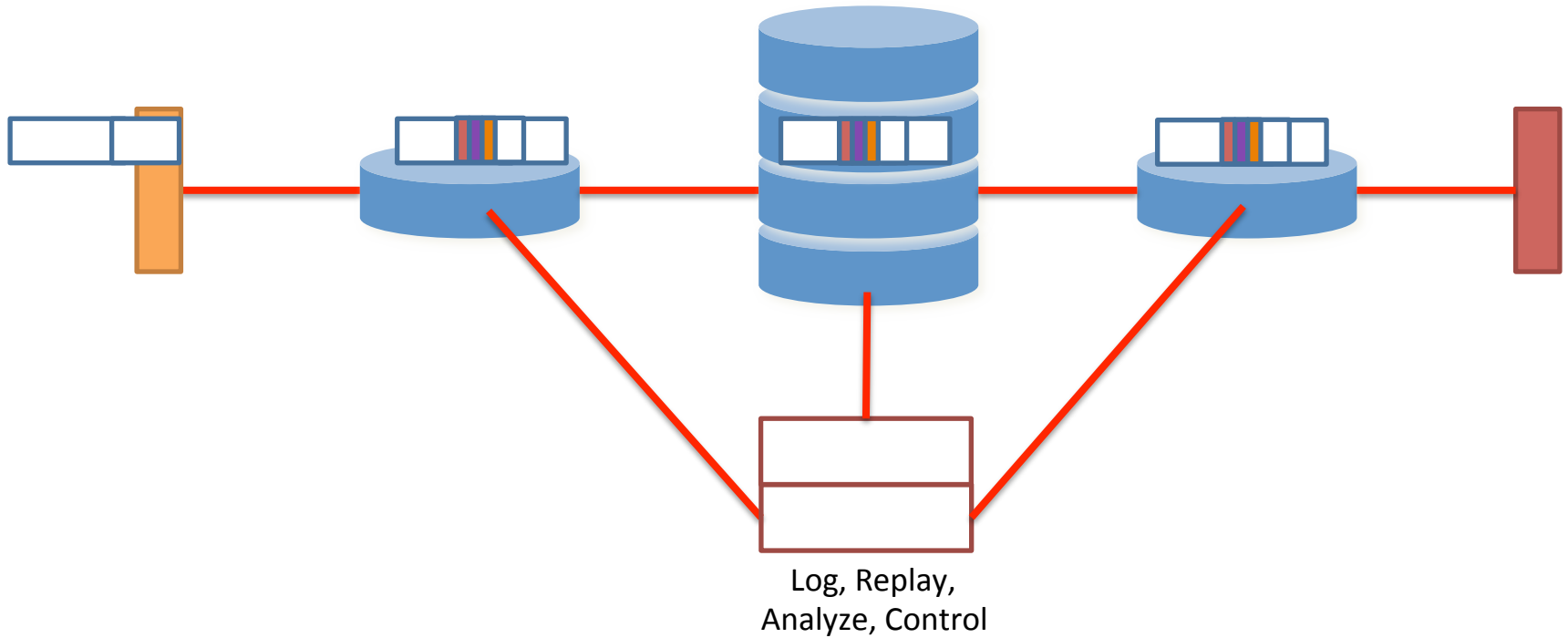
With programmable data plane, we can now answer all these questions, at line rate, without any latency penalty!

Two approaches (each is a P4 program)

1. Packet postcards

- Switch generates a small time-stamped digest for every packet
- Sends to server(s) for logging and processing

Packet postcards



Two approaches (each is a P4 program)

1. Packet postcards

- Switch generates a small time-stamped digest for every packet
- Sends to server(s) for logging and processing
- **Pros:** Can replay network history. Packet sizes unchanged.
- **Cons:** Lots of extra traffic.

Two approaches (each is a P4 program)

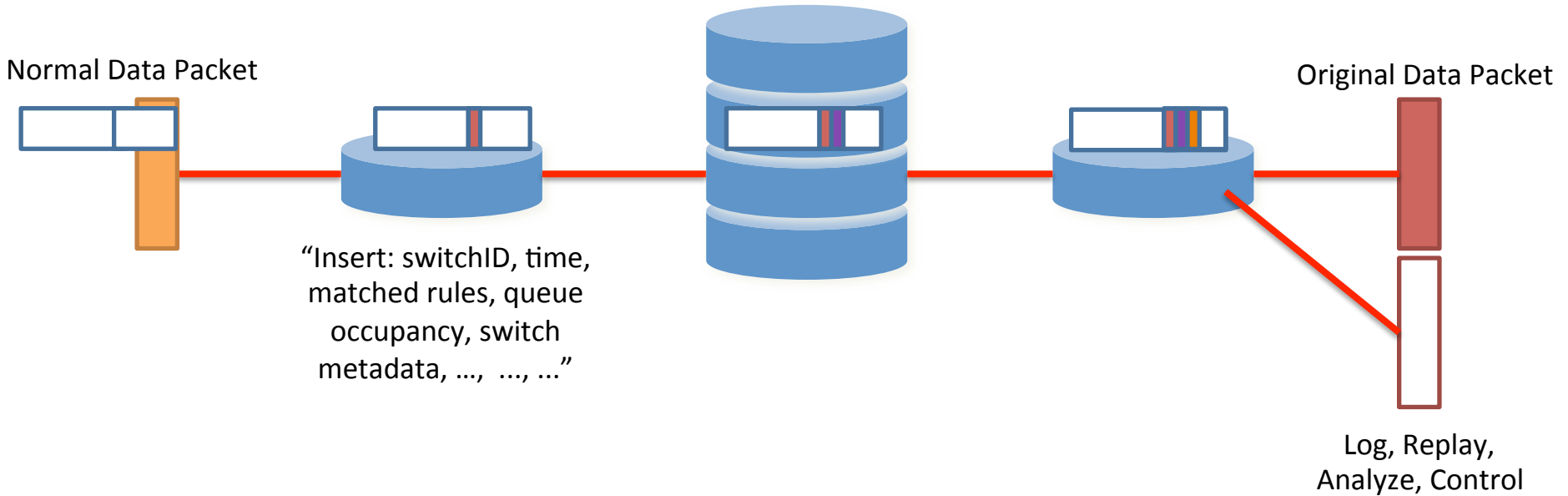
1. Packet postcards

- Switch generates a small time-stamped digest for every packet
- Sends to server(s) for logging and processing
- **Pros:** Can replay network history. Packet sizes unchanged.
- **Cons:** Lots of extra traffic.

2. Inband Network telemetry (INT)

- Data packets carry instructions to insert state into packet header

In-band Network Telemetry (INT)



Two approaches (each is a P4 program)

1. Packet postcards

- Switch generates a small time-stamped digest for every packet
- Sends to server(s) for logging and processing
- **Pros:** Can replay network history. Packet sizes unchanged.
- **Cons:** Lots of extra traffic.

2. Inband Network telemetry (INT)

- Data packets carry instructions to insert state into packet header
- **Pros:** No additional packets. Can replay network history.
- **Cons:** Packet size increases.

In-band Network Telemetry in P4

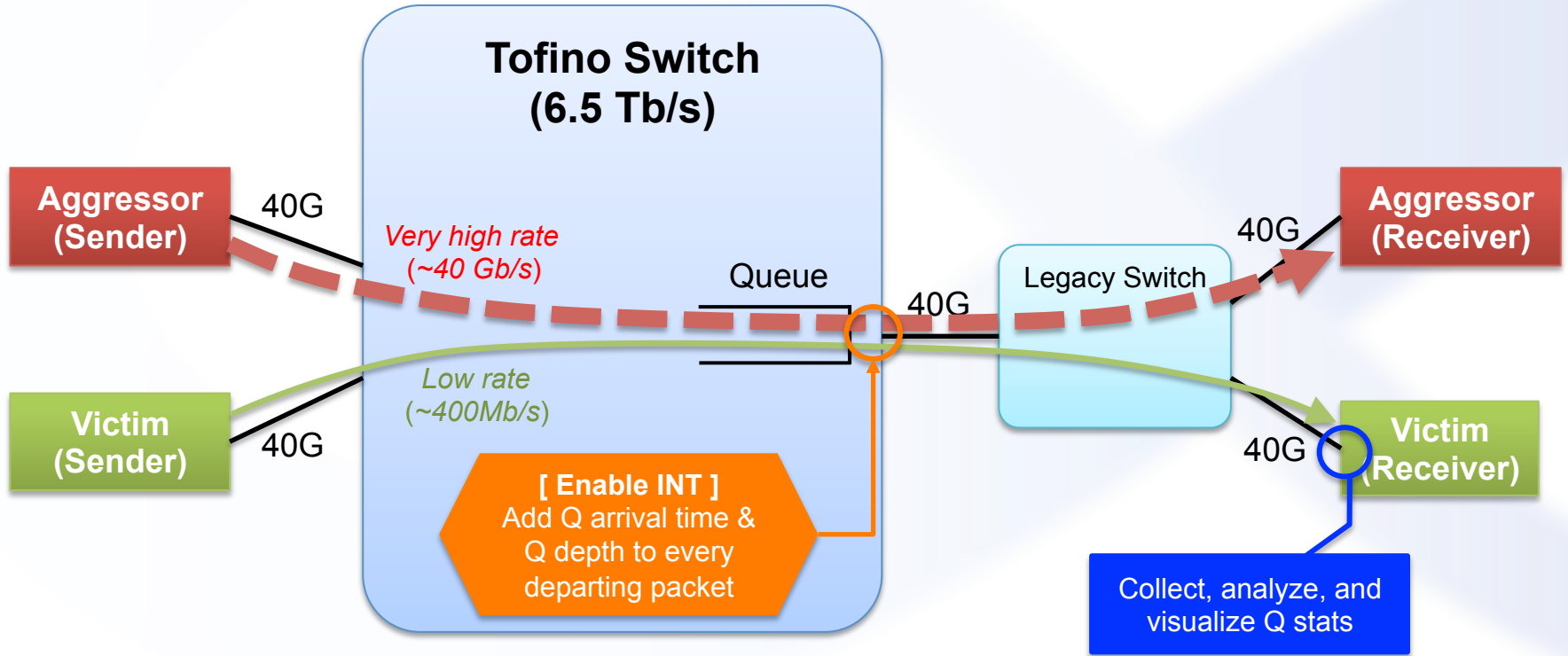
```
table int_table {
  reads {
    ip.protocol;
  }
  actions {
    export_queue_latency;
  }
}
```

```
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);
}
```

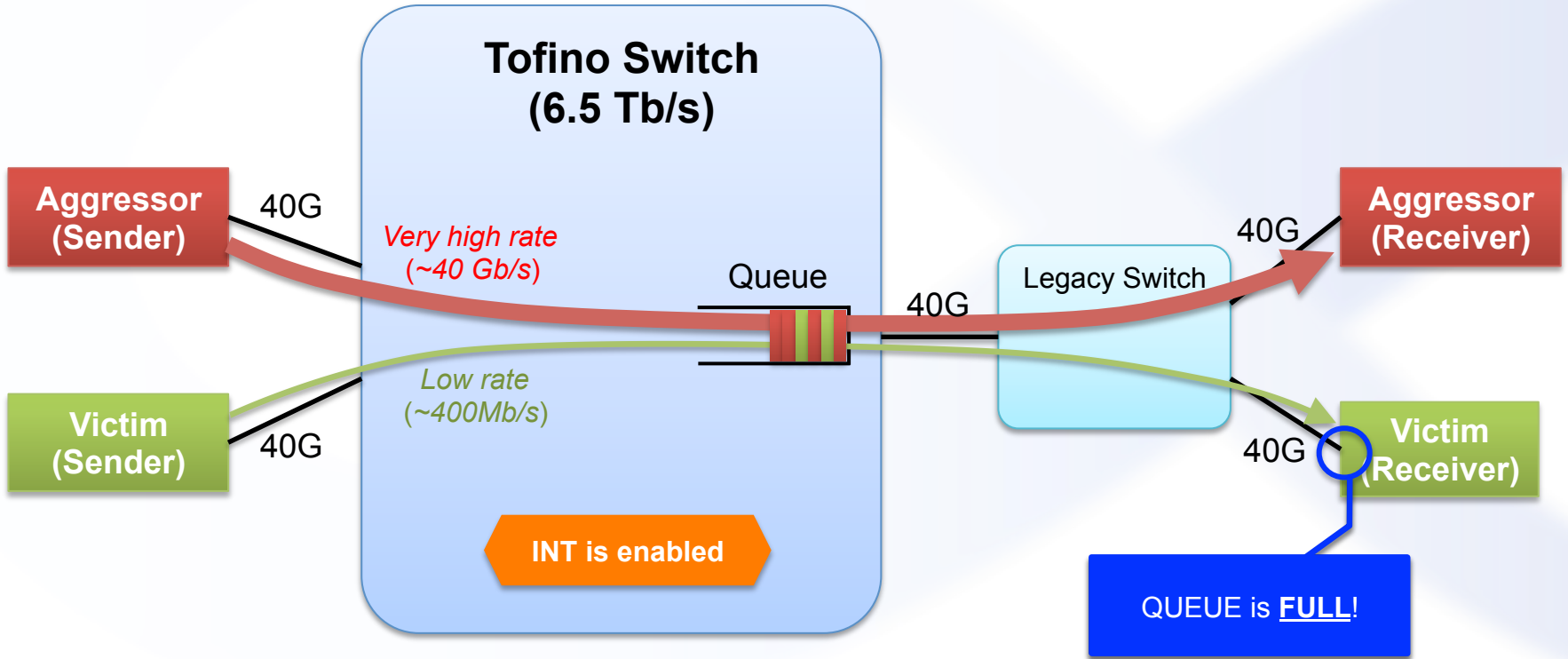
Add TCP Options &
copy switch ID and queue latency
Into the options

Demo!

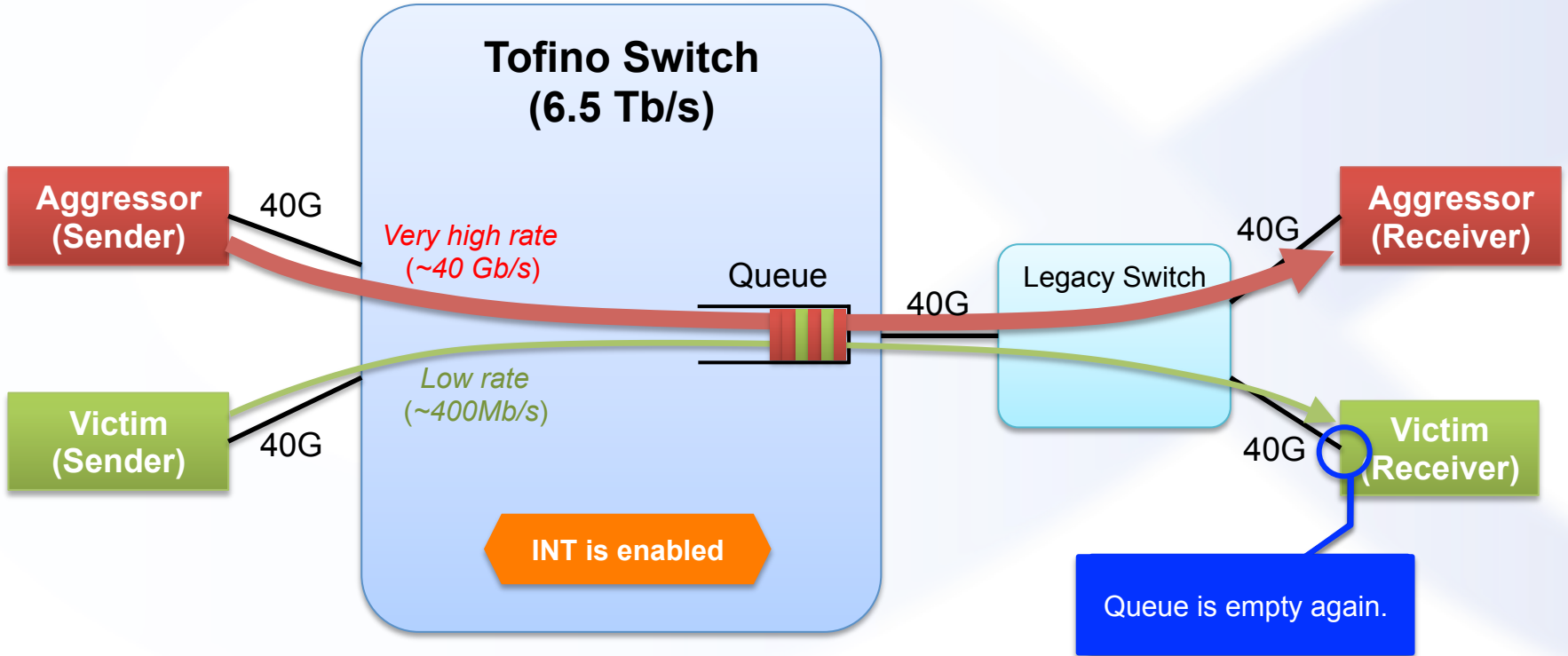
Demo environment



What's going to happen ...

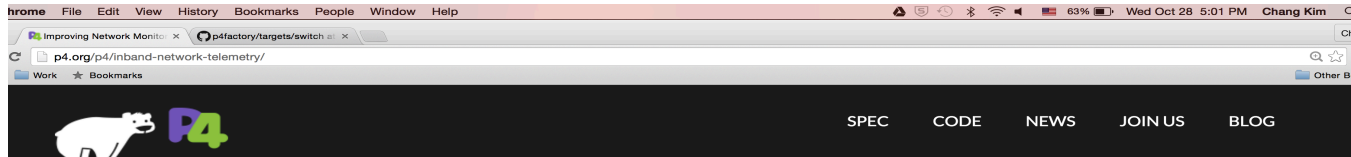


What's going to happen ...



INT open-source code and spec

- <http://p4.org/p4/inband-network-telemetry/>



Improving Network Monitoring and Management with Programmable Data Planes

By Mukesh Hira & LJ Wobker

Quicklinks: [The INT specification](#) --- [INT GitHub repository](#) --- [INT demo video](#)

Compute virtualization and the widespread deployment of virtual machines led to an extension of the network into the hypervisor. This extension is a natural consequence of the need to support virtual network services — logical switches, load balancers, and firewalls — over virtual network interfaces. VMware Network I/O virtualization (vswi) is a technology that decouples the virtual network from the physical network so as to enable deployment of virtual services over any physical network infrastructure, the only requirement from the physical network being IP connectivity between the hypervisors.

While the decoupling of physical and virtual topologies has advantages, it is important to have some interaction between the physical and virtual switches to allow for end-to-end monitoring of the entire physical + virtual network from a “single pane of glass” and to help in troubleshooting and fault isolation in complex physical + virtual topologies.

We propose methods for various network elements to collect and report their state in real-time, allowing for improved cooperation between the virtual and physical layers without requiring intermediate layers such as CPU driven control planes. The general term we have applied to these methods is INT: Inband Network Telemetry.

Some examples of useful network state to be reported by network elements are –

- (i) <Switch-ID, Input port ID, output port ID> – This allows for determination and monitoring of the different paths between a pair of end-points. Current mechanisms for determining multiple paths between a pair of end-points are based on IP traceroute and can only discover equal-cost layer 3 paths (ECMP routes), but cannot discover or report the existence of

Search the Site...

Recent Posts

- [Improving Network Monitoring and Management with Programmable Data Planes](#)
- [P4 goes to England: SIGCOMM 2015](#)
- [P4 language evolution](#)
- [P4 and Open vSwitch](#)
- [1st P4 Workshop on June 4, 2015](#)

Archives

- [September 2015](#)
- [July 2015](#)
- [June 2015](#)
- [May 2015](#)

Categories

- [API](#)

Recapping: Why is data-plane programmability a big deal?

Key benefits of programmable forwarding

1. **New features**: Add new protocols
2. **Reduce complexity**: Remove unused protocols
3. **Efficient use of resources**: Flexible use of tables
4. **Greater visibility**: New diagnostics, telemetry, OAM etc.
5. **Modularity**: Compose forwarding behavior from libraries
6. **Portability**: Specify forwarding behavior once; compile to many devices
7. **Own your own IP**: No need to tell the chip vendor your features

Closing remark

*Network is becoming a programmable platform,
repeating the same evolution pattern that already took
place in computing and storage industries*

**Join the 4th P4 Workshop and Developer Day,
May/16 - 17 at Stanford (more info at <http://p4.org>)**

Thanks!

Q&A

BACKUP SLIDES