QCon SAN FRANCISCO

Programming The Network Data Plane

Changhoon Kim

P4.org

BAREFOOT NETWORKS

Beautiful ideas: What if you could …

- Realize a small, but super-fast DNS cache
- Perform TCP SYN authentication for billions of SYNs per sec
- Build a replicated key-value store ensuring RW ops in a few usecs
- Improve your consensus service performance by ~100x
- Boost your Memcached cluster’s throughput by ~10x
- Speed up your DNN training dramatically by realizing parameter servers

… using ***switches*** in your network?

You couldn't do any of those so far because …

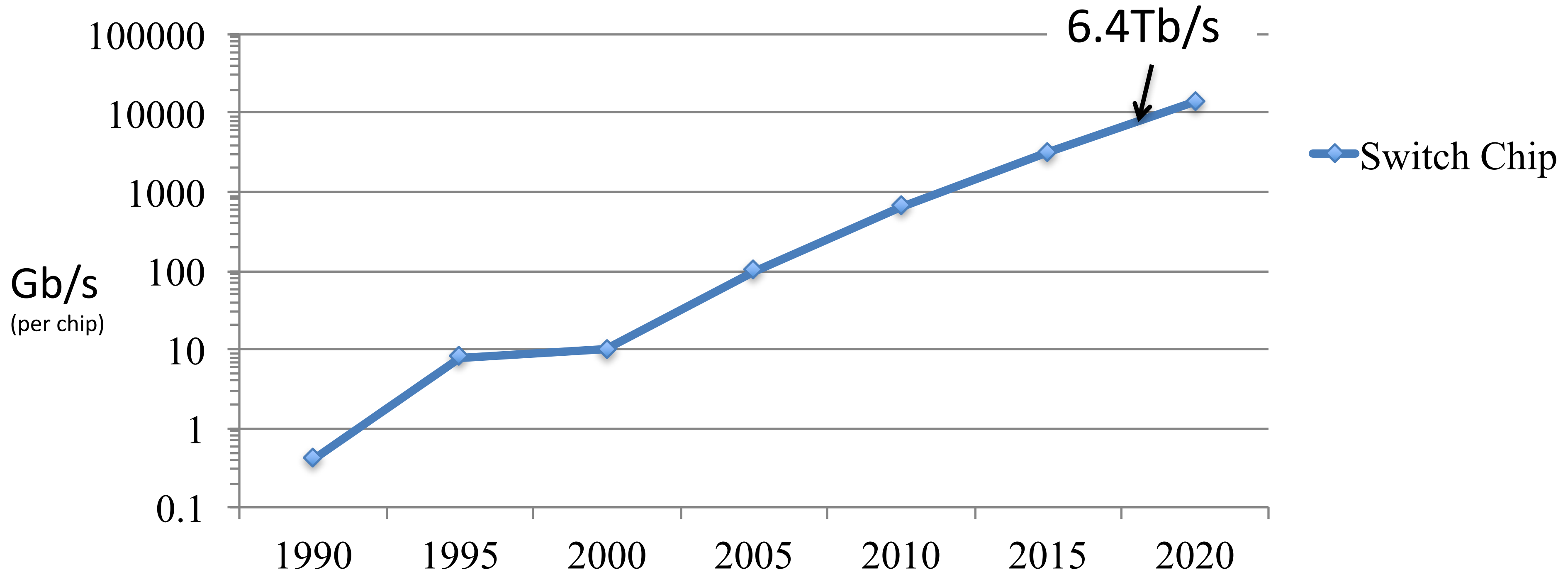
- No DIY – must work with vendors at feature level
- Excruciatingly complicated and involved process to build consensus and pressure for features
- Painfully long and unpredictable lead time
- To use new features, you must get new switches
- What you finally get != what you asked for

This is very unnatural to developers

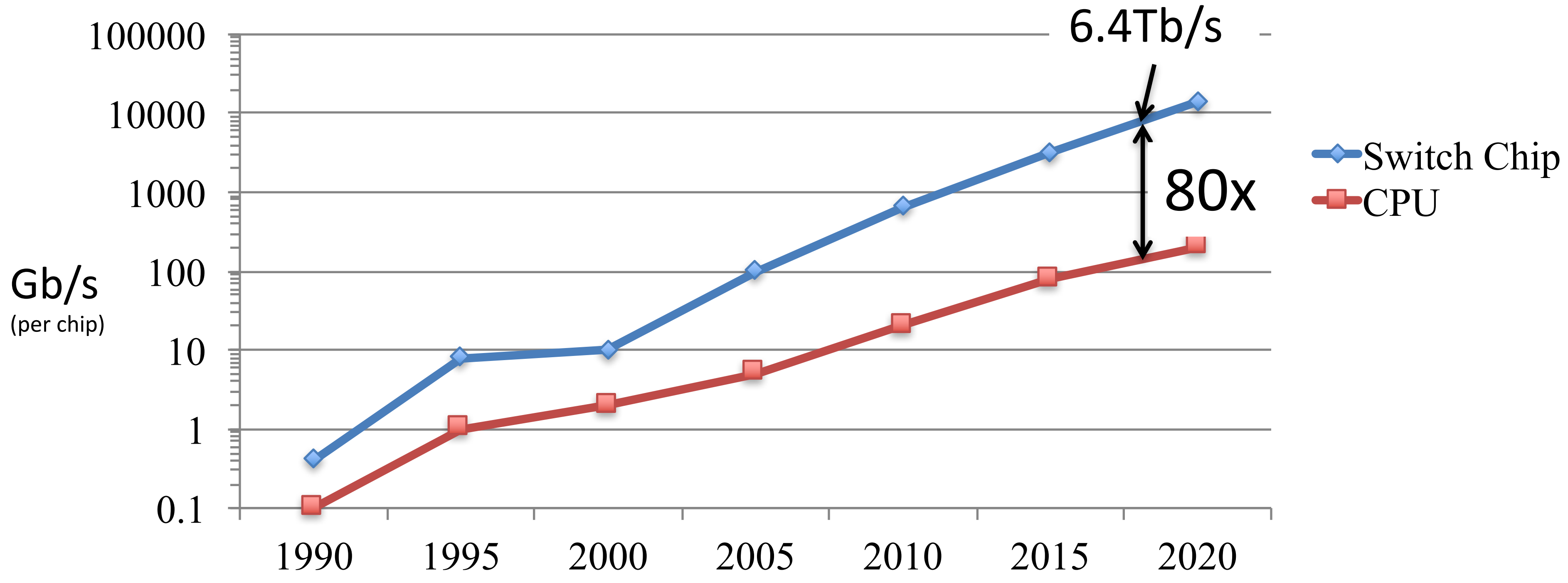
- Because you all know how to realize your own ideas by "programming" CPUs
  - Programs used in every phase (implement, test, and deploy)
  - Extremely fast iteration and differentiation
  - You own your own ideas
  - A sustainable ecosystem where all participants benefit

Can we replicate this healthy, sustainable ecosystem for networking?

Reality: Packet forwarding speeds

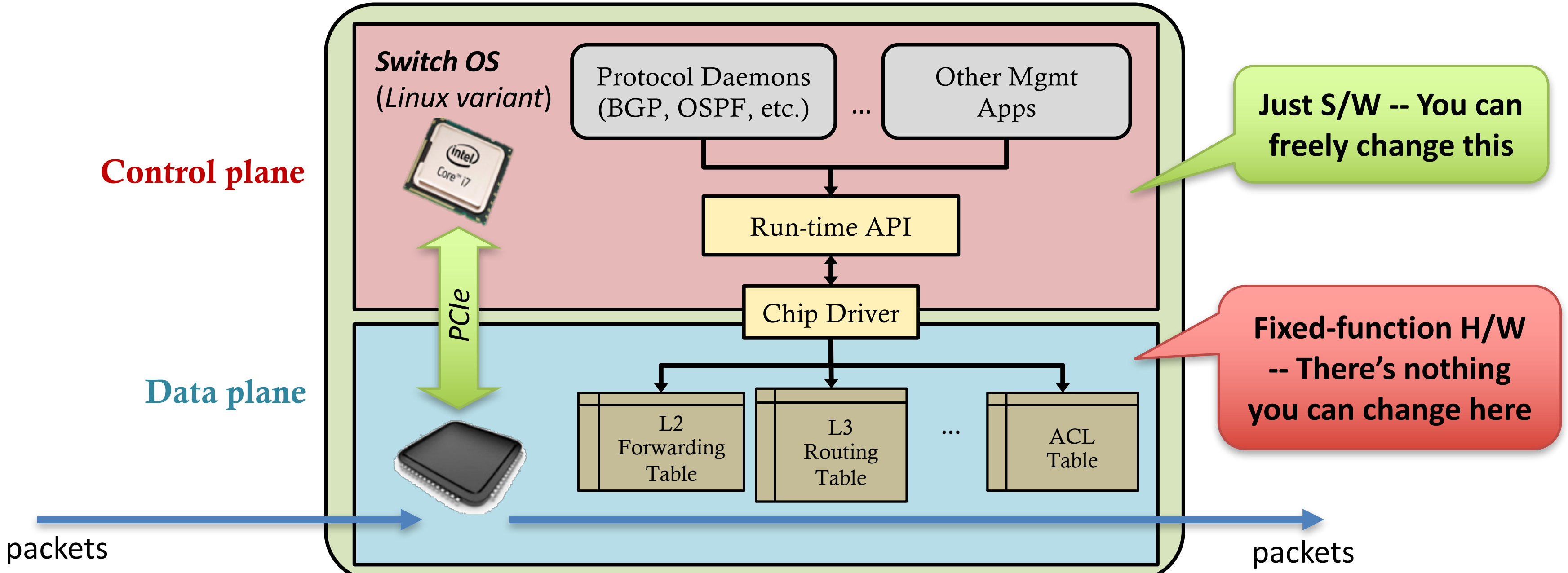


Reality: Packet forwarding speeds



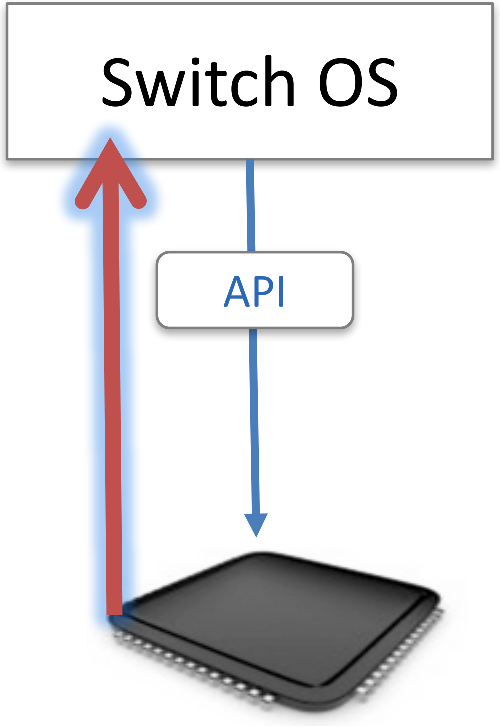
What does a typical switch look like?

A switch is just a Linux box with a high-speed switching chip



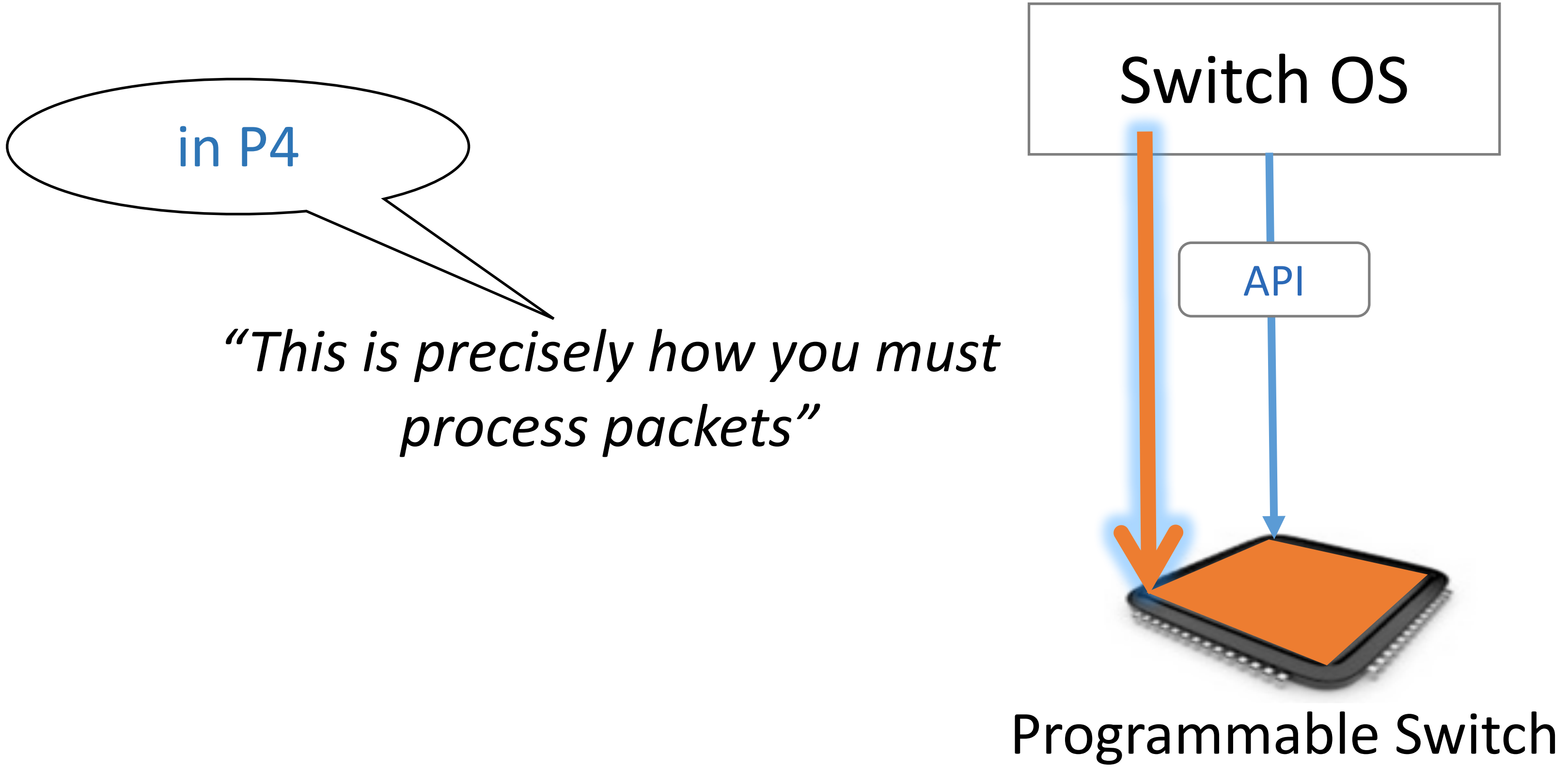
Networking systems have been built “bottoms-up”

“This is roughly how I process packets …”



Fixed-function switch

Turning the tables “top-down”



“This is precisely how you must process packets”

“Programmabl
-100x slower
than fixed-
ey cost more
and
wer.”

tional wisdom in networking

Not true anymore! ☺

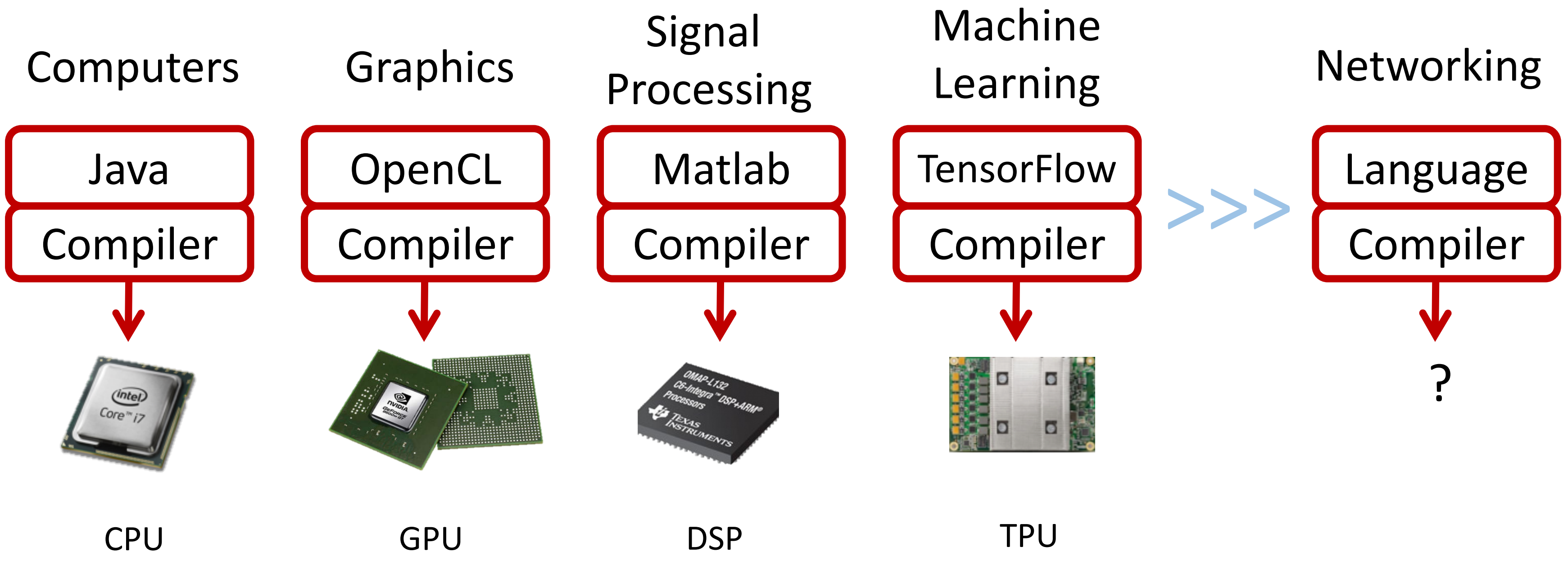
Evidence: Tofino 6.5Tb/s switch (arrived Dec 2016)

The world's fastest <u>and</u> most programmable switch.

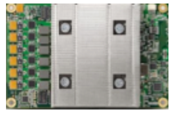
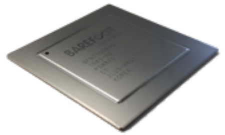
No power, cost, or power penalty compared to fixed-function switches.

An incarnation of **PISA (Protocol Independent Switch Architecture)**

Domain-specific processors

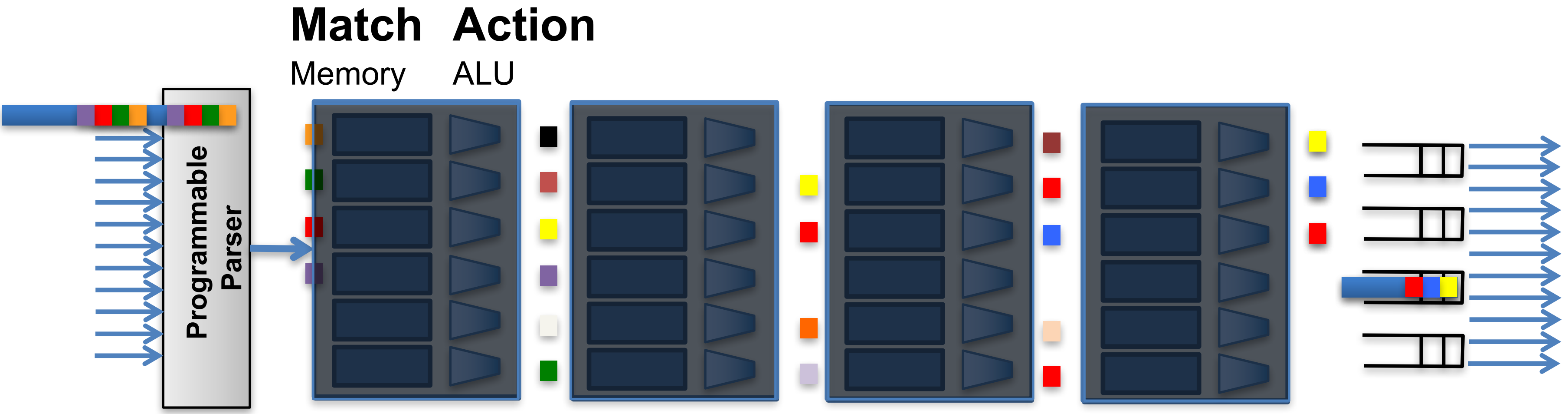


Domain-specific processors

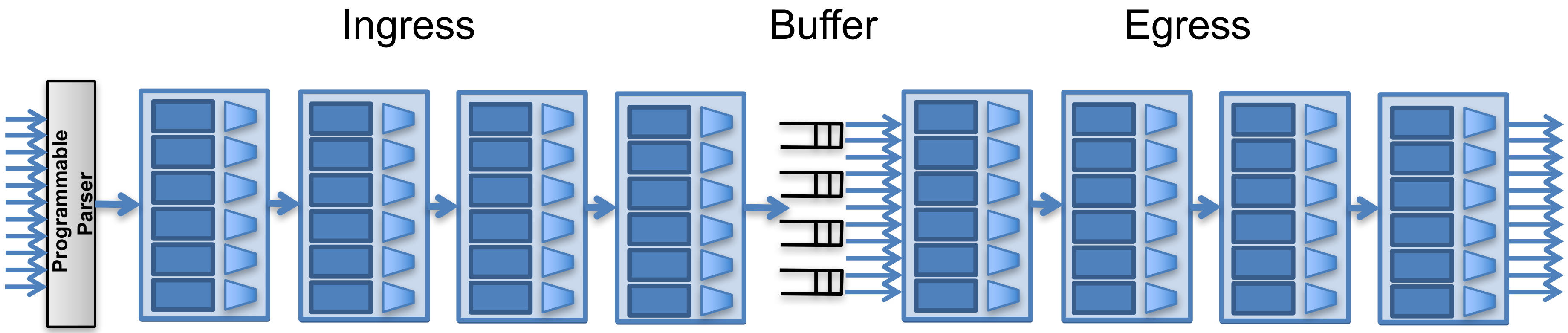
Computers	Graphics	Signal Processing	Machine Learning		Networking
Java	OpenCL	Matlab	TensorFlow		P4
Compiler	Compiler	Compiler	Compiler	>>>	Compiler
					
CPU	GPU	DSP	TPU		**PISA**

PISA: An architecture for high-speed programmable packet forwarding

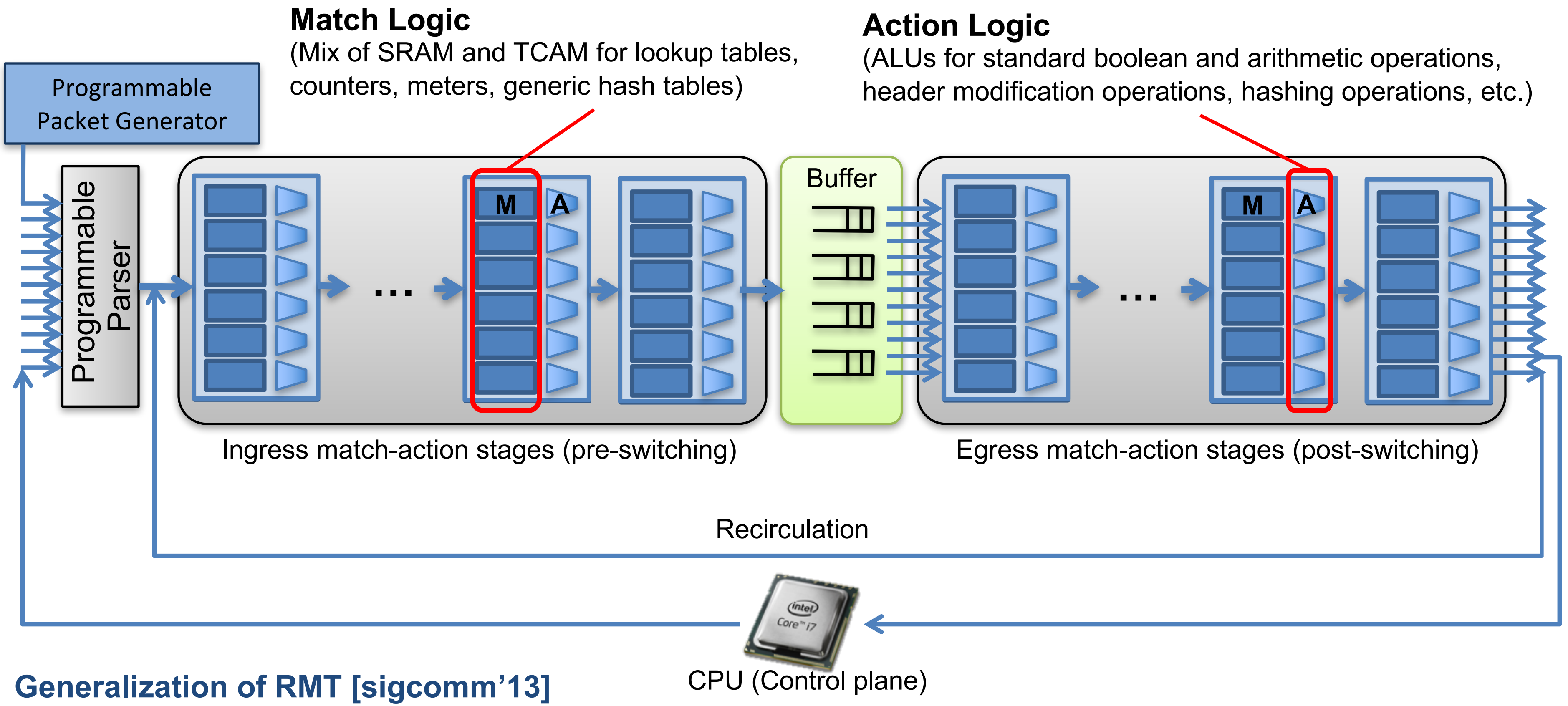
PISA: Protocol Independent Switch Architecture



PISA: Protocol Independent Switch Architecture



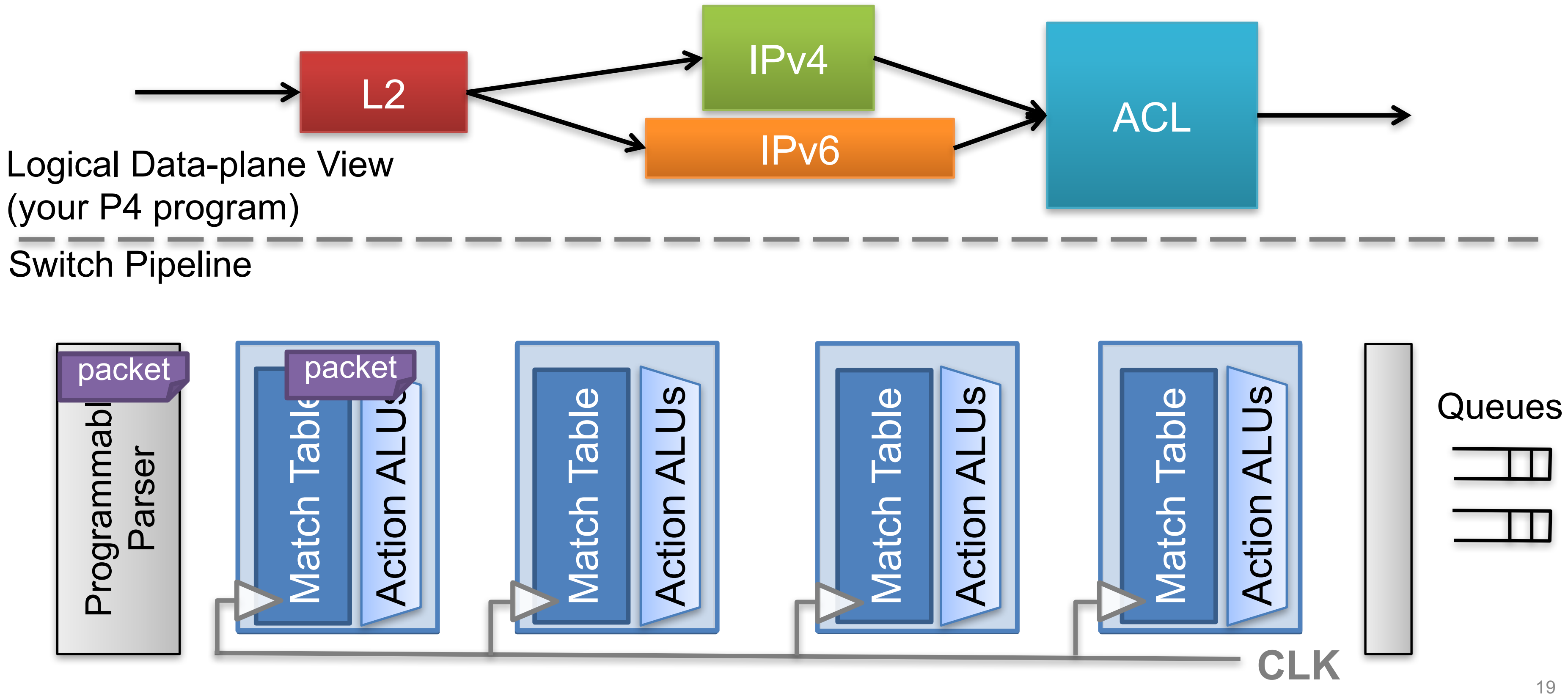
PISA: Protocol Independent Switch Architecture


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.)
Programmable Packet Generator
Programmable Parser
M
A
…
Buffer
M
A
…
Ingress match-action stages (pre-switching)
Egress match-action stages (post-switching)
Recirculation
intel
Core™ i7
CPU (Control plane)

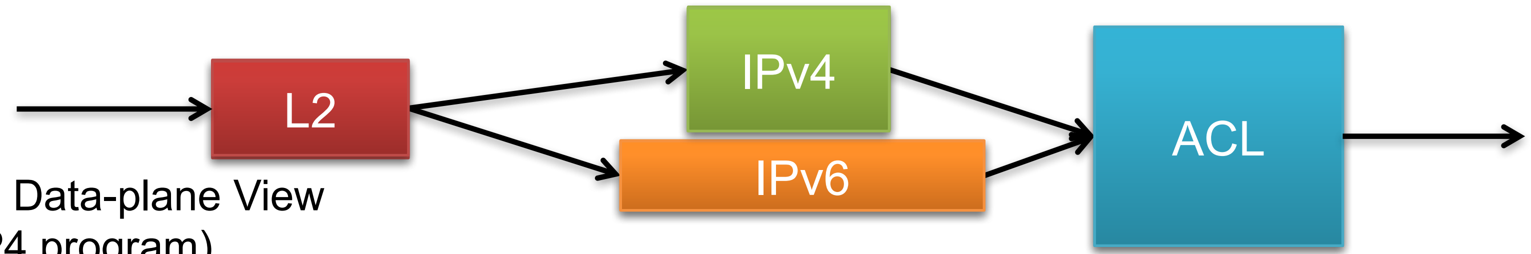
Generalization of RMT [sigcomm'13]

Why we call it protocol-independent packet processing

Device does not understand any protocols until it gets programmed

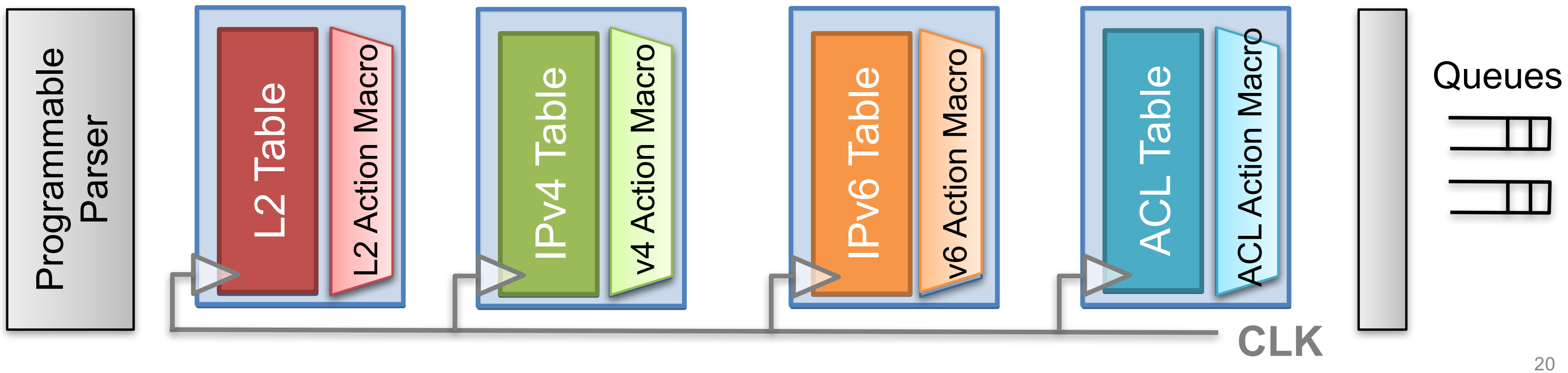


Mapping logical data-plane design to physical resources

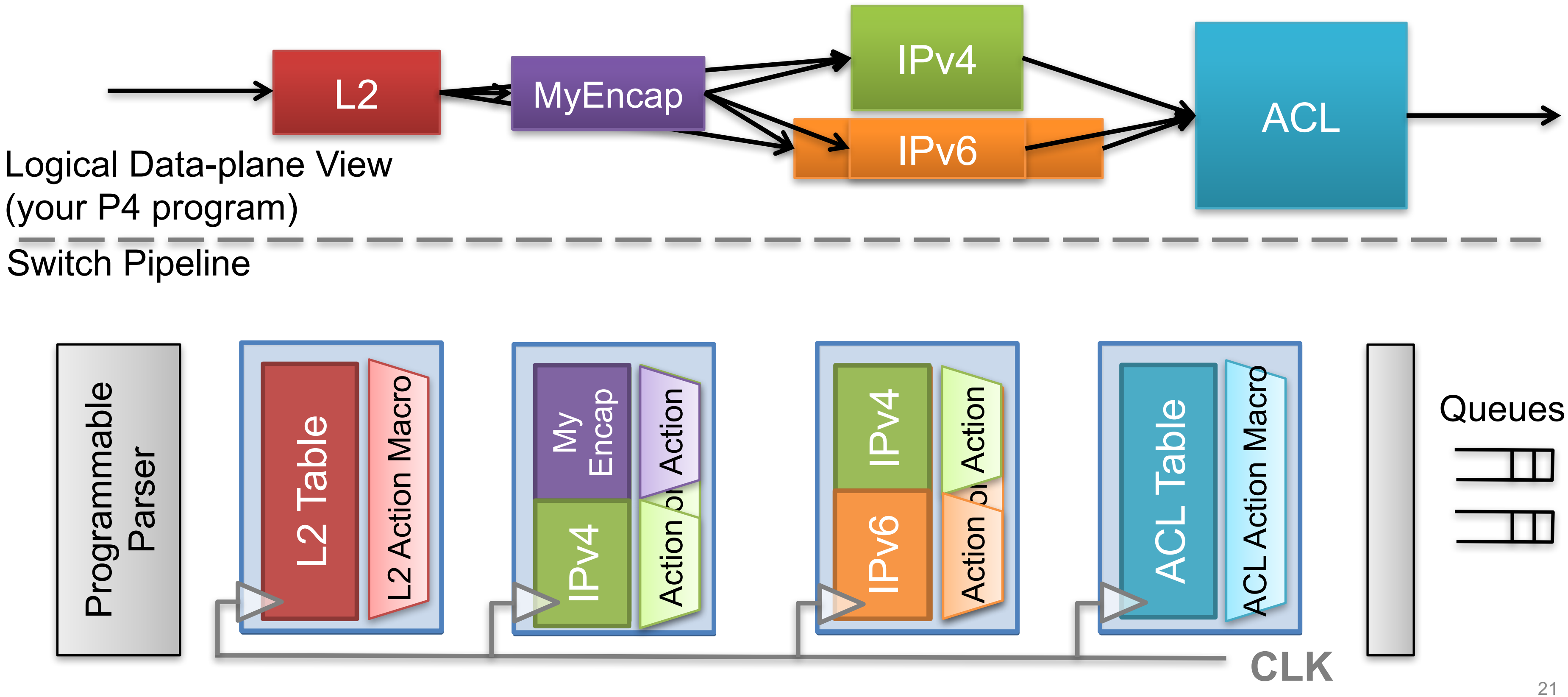


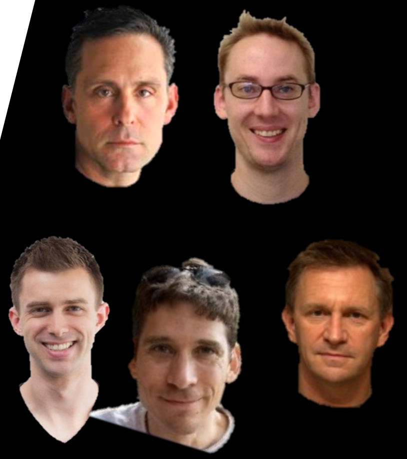
Logical Data-plane View
(your P4 program)

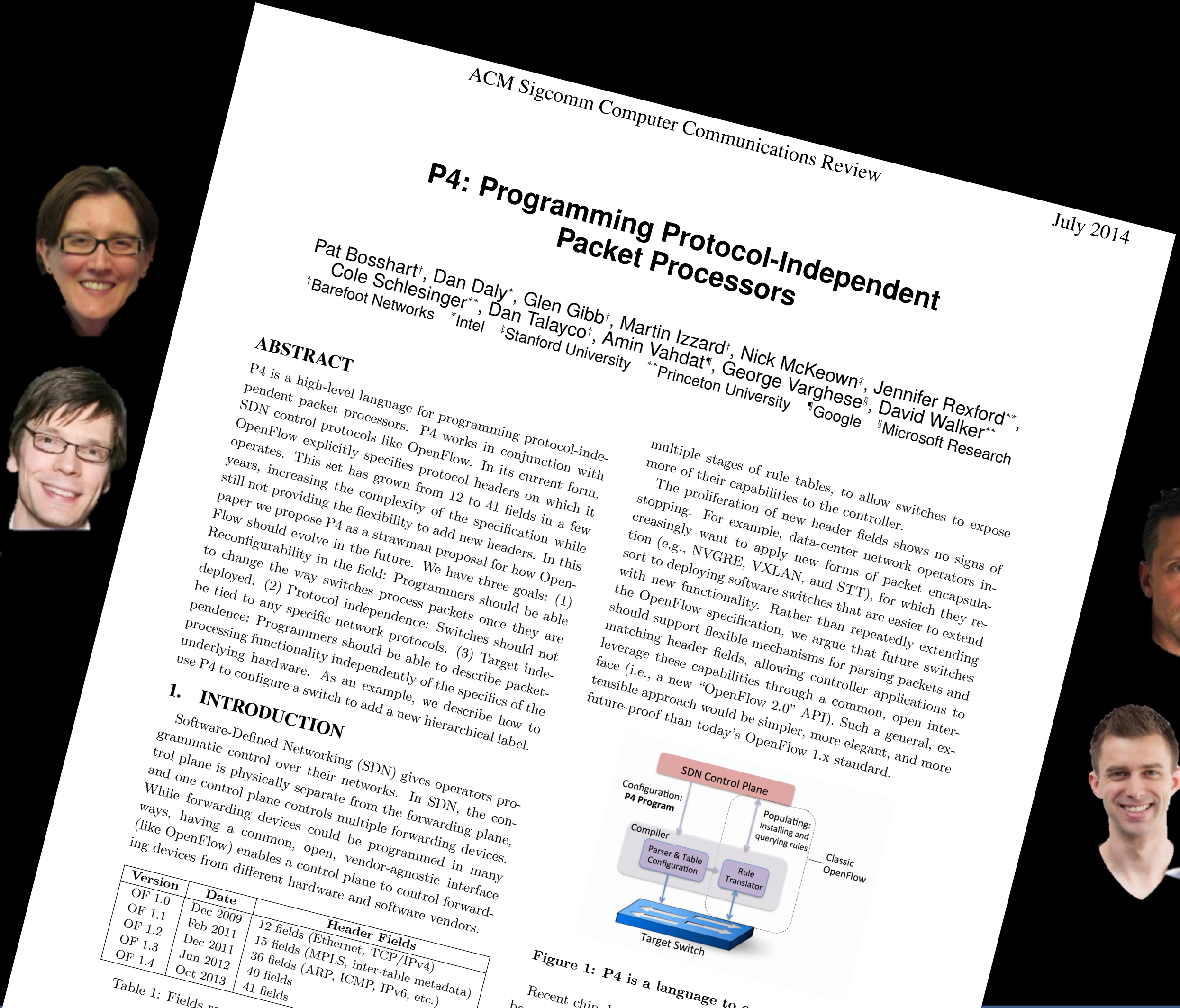
Switch Pipeline



Re-program in the field







ACM Sigcomm Computer Communications Review

July 2014

P4: Programming Protocol-Independent Packet Processors

Pat Bosshart[†], Dan Daly[*], Glen Gibb[†], Martin Izzard[†], Nick McKeown[‡], Jennifer Rexford[**], Cole Schlesinger[**], Dan Talayco[†], Amin Vahdat[¶], George Varghese[§], David Walker[**]
[†]Barefoot Networks [*]Intel [‡]Stanford University [**]Princeton University [¶]Google [§]Microsoft Research

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 not 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.

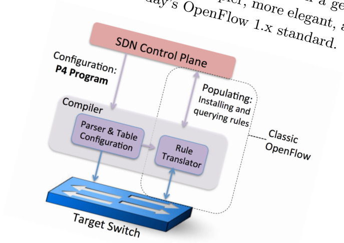
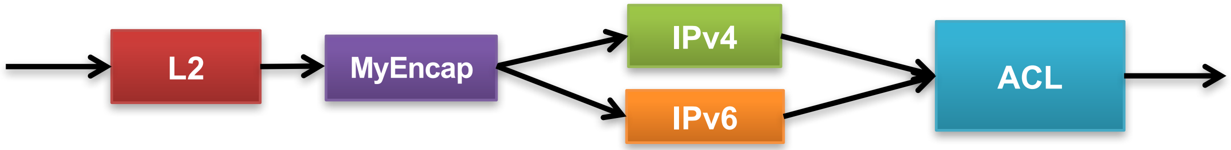

SDN Control Plane
Configuration: P4 Program
Populating: Installing and querying rules
Compiler
Parser & Table Configuration
Rule Translator
Classic OpenFlow
Target Switch

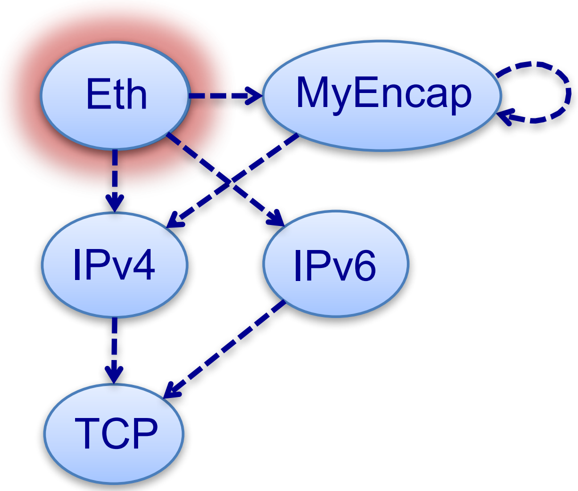
Figure 1: P4 is a language to

Recent chip

be

What does a P4 program look like?



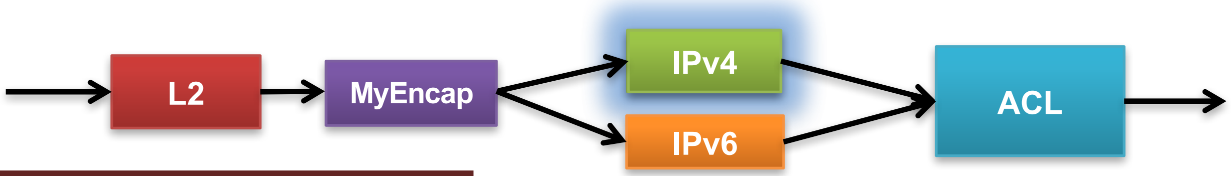


```
header_type ethernet_t {
     fields {
          dstAddr : 48;
          srcAddr : 48;
}
```

```
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);
}
```

P4.org (http://p4.org)

- **Open-source community to nurture the language**
  - Open-source software – Apache license
  - A common language: $P4_{16}$
  - Support for various types of devices and targets
- **Enable a wealth of innovation**
  - Diverse “apps” (including proprietary ones) running on commodity targets
- **With no barrier to entry**
  - Free of membership fee, free of commitment, and simple licensing

So, what kinds of exciting new opportunities are arising?

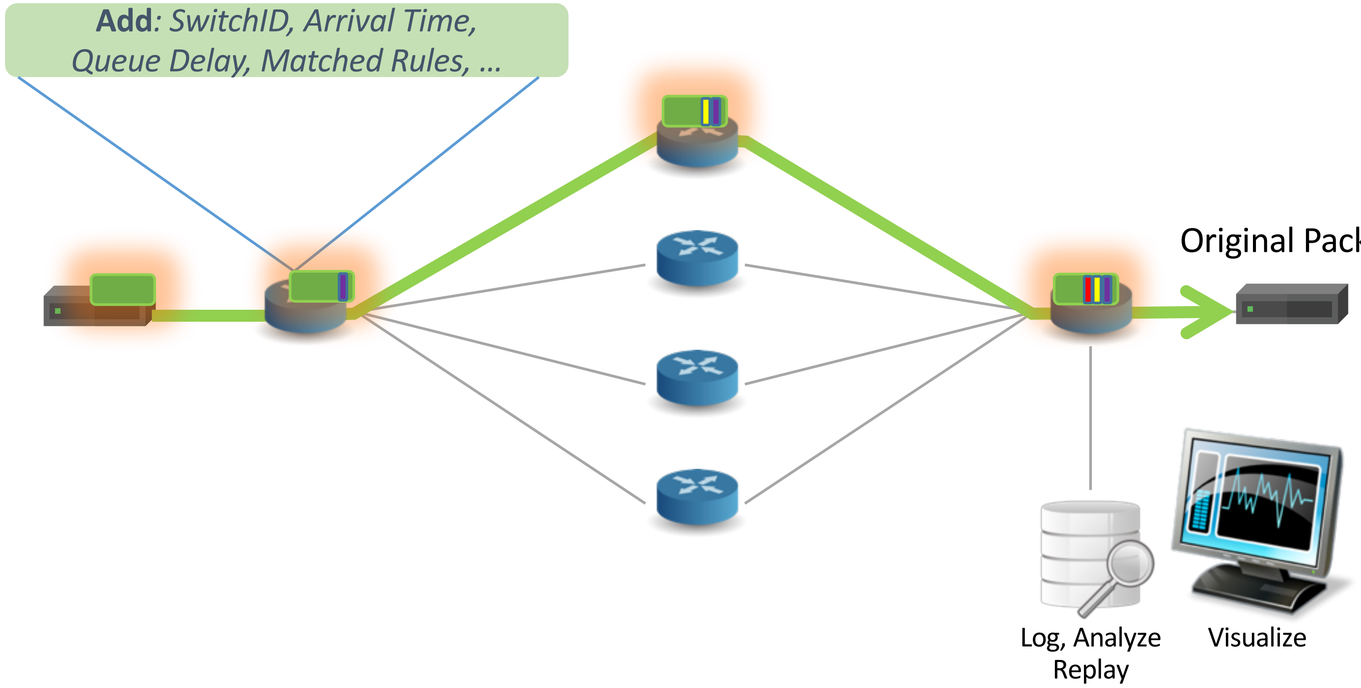
The network should answer these questions

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*?”

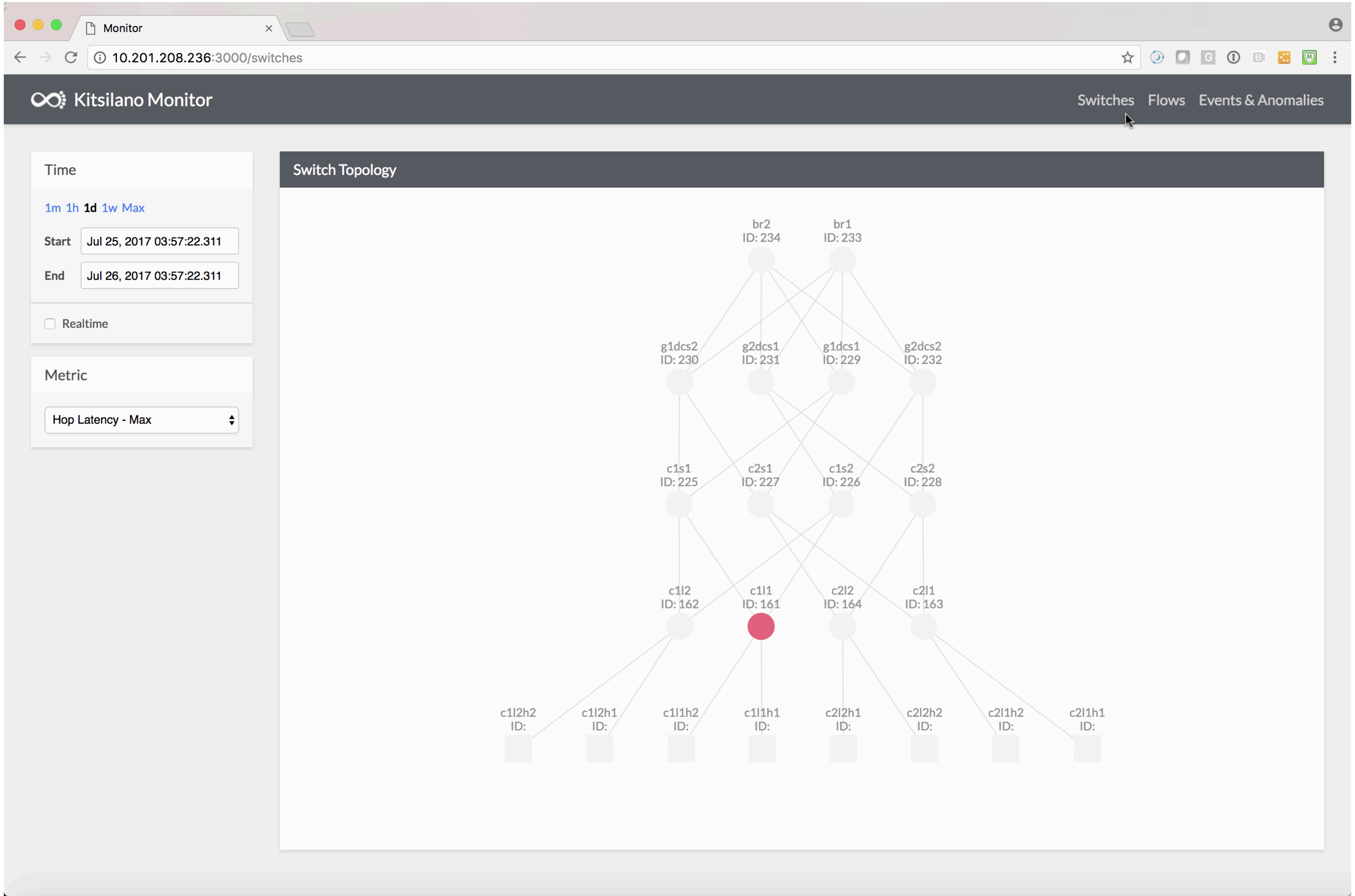
PISA + P4 can answer all four questions for the first time.
At full line rate. Without generating any additional packets!

In-band Network Telemetry (INT)

A read-only version of Tiny Packet Programs [sigcomm'14]


Add: *SwitchID, Arrival Time,*
Queue Delay, Matched Rules, …
Original Packet
Log, Analyze
Replay
Visualize

A quick demo of INT!


Monitor

10.201.208.236:3000/switches

Kitsilano Monitor

Switches Flows Events & Anomalies

Time

1m 1h 1d 1w Max

Start Jul 25, 2017 03:57:22.311

End Jul 26, 2017 03:57:22.311

Realtime

Metric

Hop Latency - Max

Switch Topology

What does this mean to you?

- Improve your distributed apps' performance with telemetry data
- Ask the four key questions regarding your packets to network admins or cloud providers
- Huge opportunities for Big-data processing and machine-learning experts
- "Self-driving" network is not hyperbole

PISA: An architecture for high-speed programmable ~~packet forwarding~~ event processing

What we have seen so far: Adding new networking features

1. New encapsulations and tunnels
2. New ways to tag packets for special treatment
3. New approaches to routing: e.g., source routing in data-center networks
4. New approaches to congestion control
5. New ways to manipulate and forward packets: e.g. splitting ticker symbols for high-frequency trading

What we have seen so far: World’s fastest middle boxes

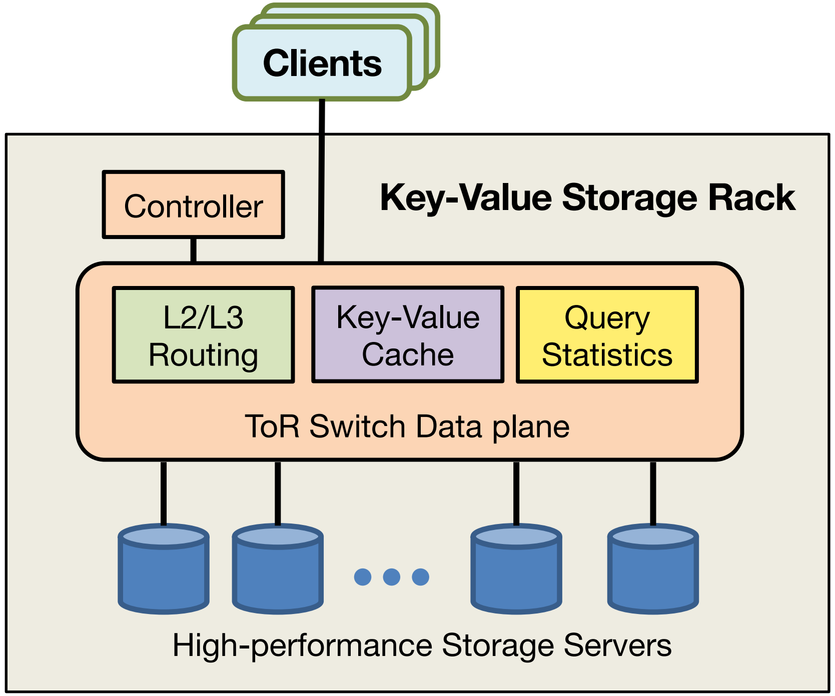
1. Layer-4 load connection balancing at Tb/s
   - Replace 100s of servers or 10s of dedicated appliances with one PISA switch
   - Track and maintain mappings for 5 ~ 10 million HTTP connections

2. Stateless firewall or DDoS detector
   - Add/delete and track 100s of thousands of new connections per second
   - Include other stateless line-rate functions
     (e.g., TCP SYN authentication, sketches, or Bloomfilter-based whitelisting)

What we have seen so far: Offloading part of computing to network

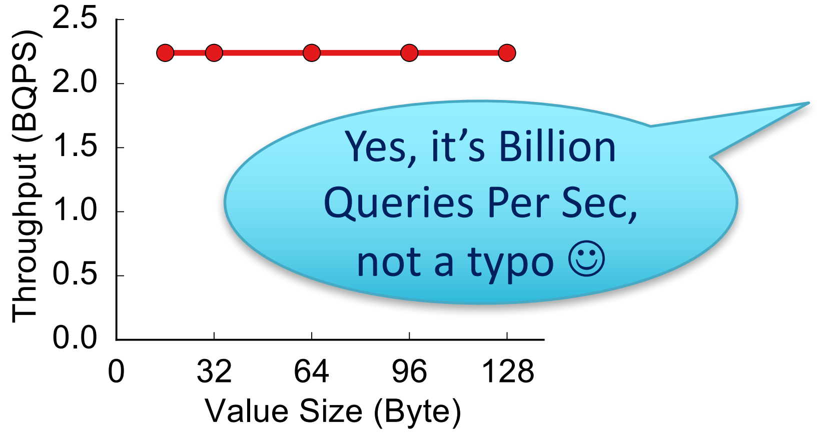
1. DNS cache
2. Key-value cache [ACM SOSP'17]
3. Chain replication
4. Paxos [ACM CCR'16] and RAFT
5. Parameter service for DNN training

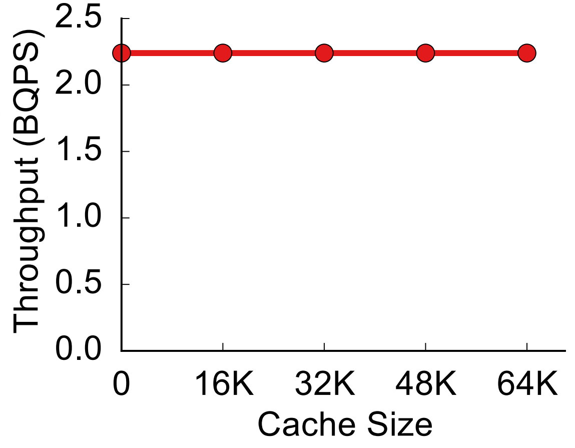
Example: NetCache



- **Non-goal**
  - Maximize the cache hit rate
- **Goal**
  - **Balance the workloads of backend servers by serving only O(*N*log*N*) hot items** -- *N* is the number of backend servers
  - Make the "fast, small-cache" theory viable for modern in-memory KV servers [Fan et. al., SOCC'11]
- **Data plane**
  - Unmodified routing
  - **Key-value cache built with on-chip SRAM**
  - **Query statistics to detect hot items**
- **Control plane**
  - Update cache with hot items to handle dynamic workloads

The "boring life" of a NetCache switch

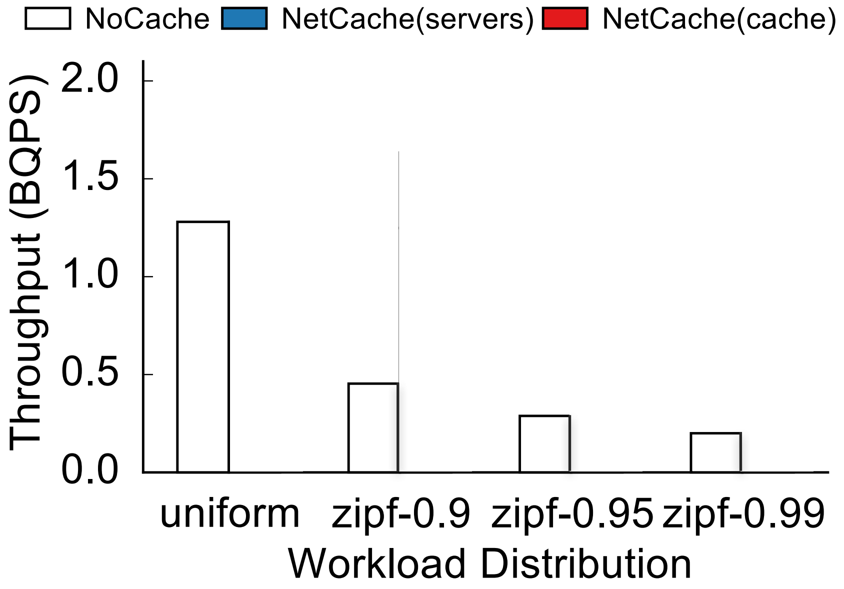




One can further increase the value sizes with more stages, recirculation, or mirroring.

And its "not so boring" benefits

Throughput of a key-value storage rack with one Tofino switch and 128 storage servers.



NetCache provides **3-10x throughput improvements**.

NetCache is a **key-value** store that leverages

in-network caching to achieve

Billions of queries/sec & a few usec latency

even under

highly-skewed & rapidly-changing

workloads.

Summing it up …

Why data-plane programming?

1. **New features**: Realize your beautiful ideas very quickly
2. **Reduce complexity**: Remove unnecessary features and tables
3. **Efficient use of H/W resources**: Achieve biggest bang for buck
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 ideas**: No need to share your ideas with others

"Protocols are being lifted off chips and into software"

– Ben Horowitz

My observations

- PISA and P4: The first attempt to define a machine architecture and programming models for networking in a disciplined way
- Network is becoming yet another programmable platform
- It’s fun to figure out the best workloads for this new machine architecture

Want to find more resources or follow up?

- Visit http://p4.org and http://github.com/p4lang
  - P4 language spec
  - P4 dev tools and sample programs
  - P4 tutorials
- Join P4 workshops and P4 developers' days
- Participate in P4 working group activities
  - Language, target architecture, runtime API, applications
- Need more expertise across various fields in computer science
  - To enhance PISA, P4, dev tools (e.g., for formal verification, equivalence check, and many more …)

Thanks.
Let’s develop your beautiful ideas
in P4!