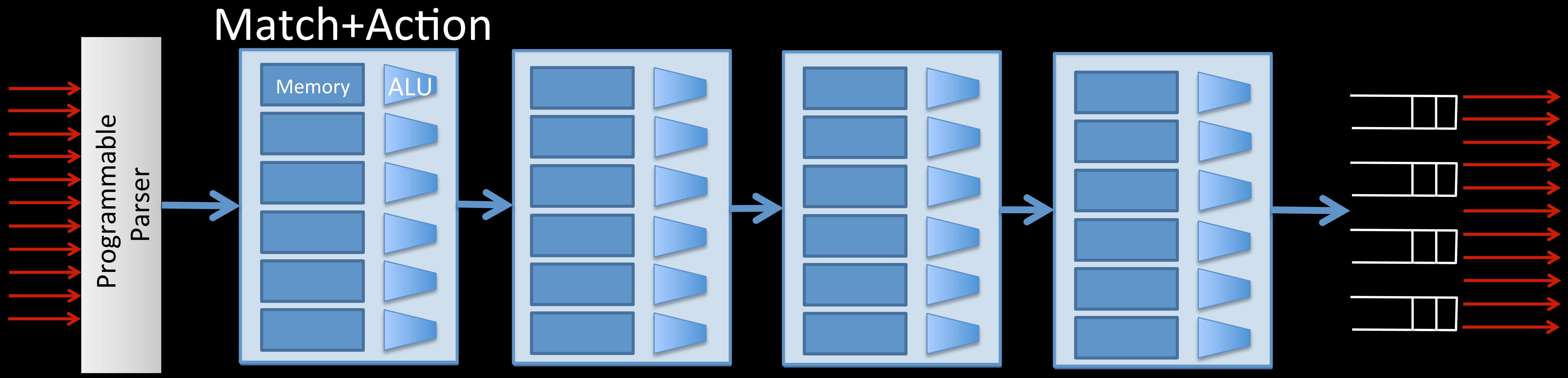
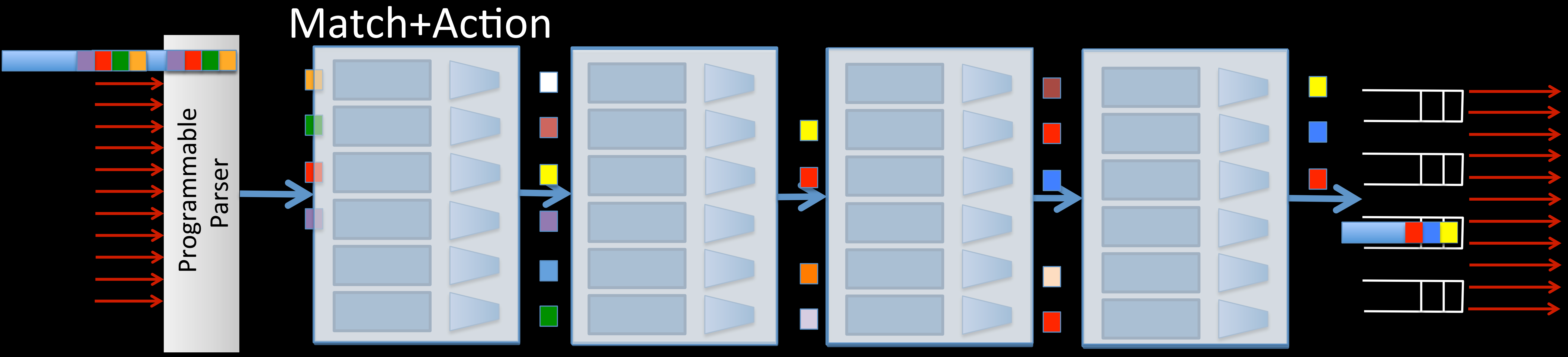
Programming the Forwarding Plane

Nick McKeown

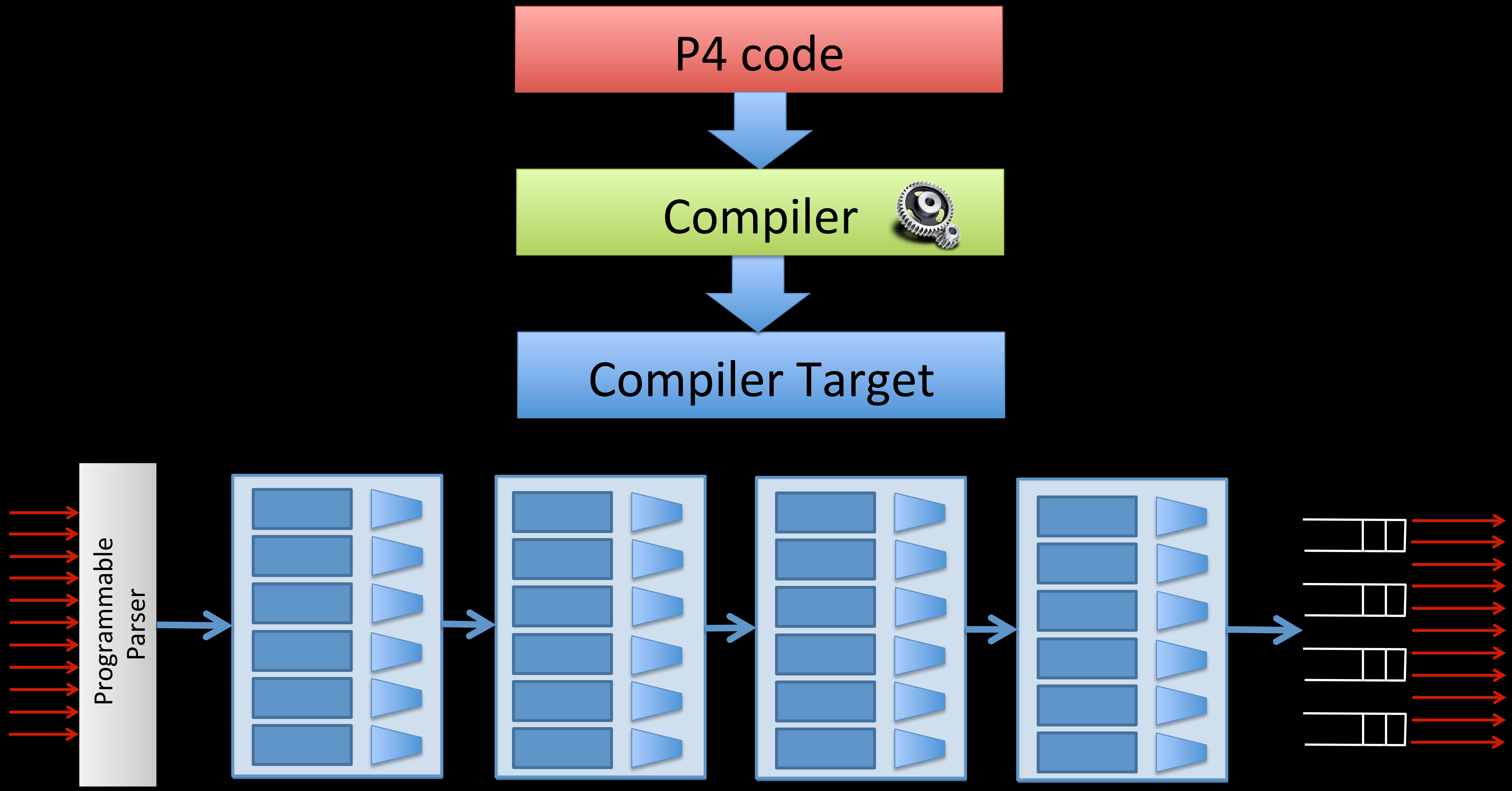
PISA: Protocol Independent Switch Architecture
[Sigcomm 2013]



PISA: Protocol Independent Switch Architecture



P4 and PISA



[ACM CCR 2014]
“Best Paper 2014”

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 recognized by the OpenFlow standard

The OpenFlow interface started simple, with the abstraction of a single table of rules that could match packets on a dozen header fields (e.g., MAC addresses, IP addresses, protocol, TCP/UDP port numbers, etc.). Over the past five years, the specification has grown increasingly more complicated (see Table 1), with many

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 configure switches.

Recent chip designs demonstrate that such flexibility can be achieved in custom ASICs at terabit speeds [1, 2, 3]. Programming this new generation of switch chips is far from easy. Each chip has its own low-level interface, akin to microcode programming. In this paper, we sketch the design of a higher-level language for Programming Protocol-independent Packet Processors (P4).

George Varghese

Amin Vahdat

Jen Rexford

Dan Daly

Pat Bosshart

Glen Gibb

Martin Izzard

Dave Walker

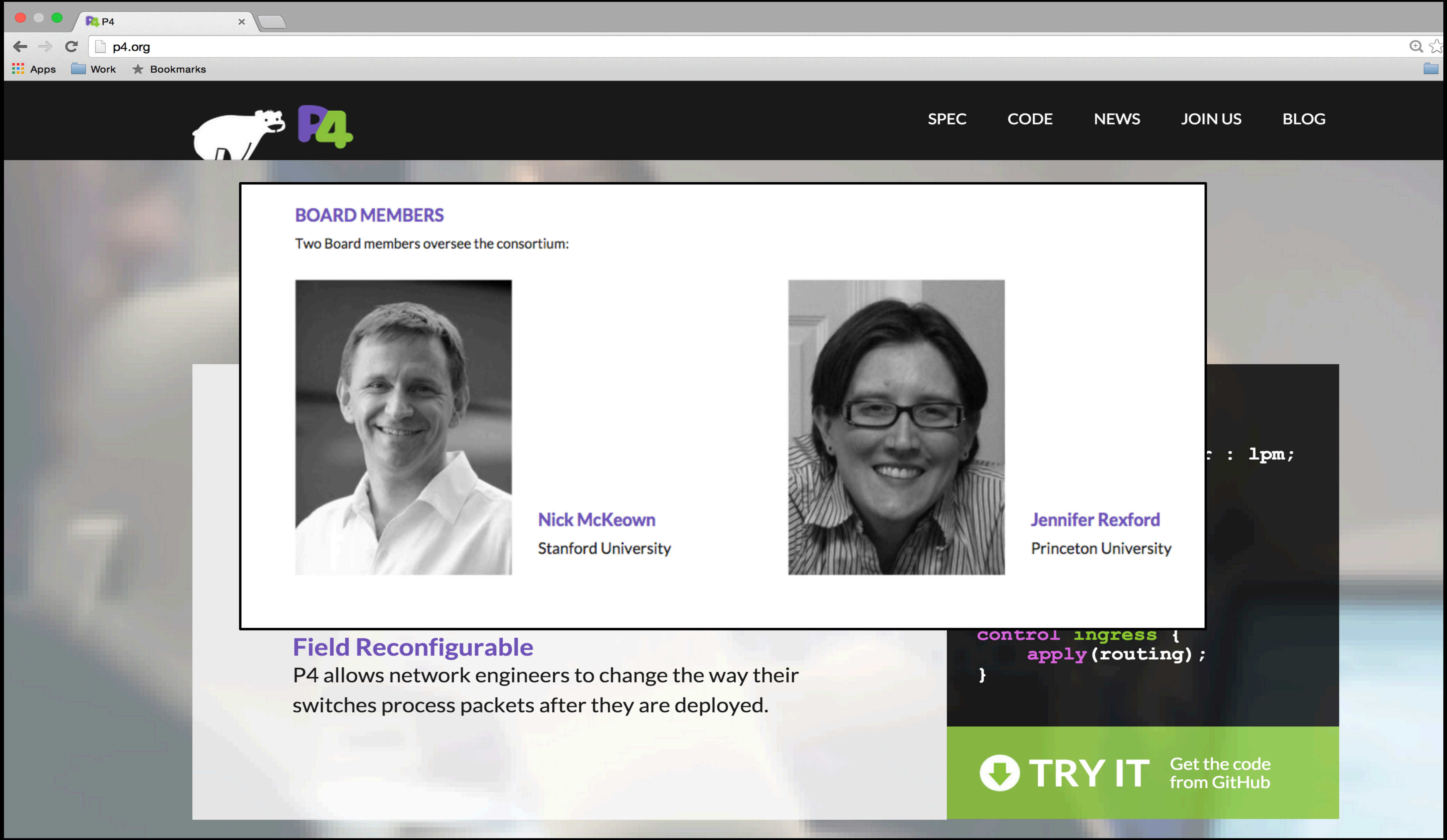
Cole Schlesinger

Dan Talayco

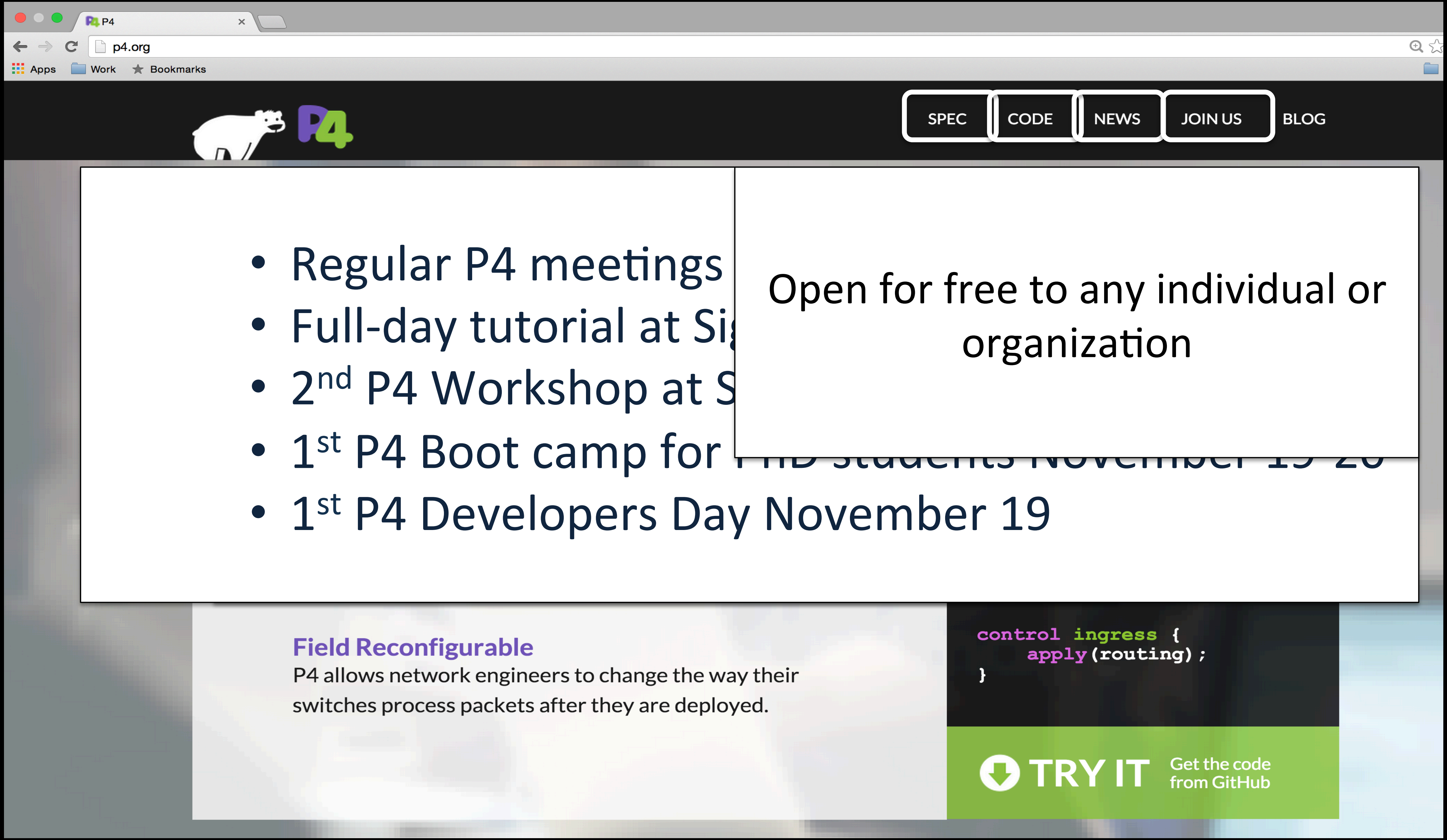
Nick McKeown

Update on P4 Language Ecosystem

P4.org – P4 Language Consortium



P4.org – P4 Language Consortium



- Regular P4 meetings
- Full-day tutorial at Si
- 2nd P4 Workshop at S
- 1st P4 Boot camp for
- 1st P4 Developers Day November 19

Open for free to any individual or organization

P4 Consortium – P4.org

Operators Alibaba Group 阿里巴巴集团, at&t, Baidu 百度, Comcast, Microsoft, SK telecom

Systems BROCADE, CISCO, CORSA, DELL, Hewlett Packard Enterprise, HUAWEI, JUNIPER NETWORKS

Targets AEPONYX Moving the Cloud at the Speed of Light™, Atomic Rules, BAREFOOT NETWORKS, BROADCOM. Connecting everything®, CAVIUM, centec networks, EZCHIP, freescale, intel, Mellanox TECHNOLOGIES, MARVELL, NETRONOME, PLUMgrid, XILINX

Academia Cornell University Founded A.D. 1865, POLITECNICO MILANO 1863, PRINCETON UNIVERSITY, Stanford University, uni.lu UNIVERSITÉ DU LUXEMBOURG, Università della Svizzera italiana

Mapping P4 programs to compiler target

Lavanya Jose, Lisa Yan, George Varghese, NM

[NSDI 2015]

Naïve Mapping: Control Flow Graph

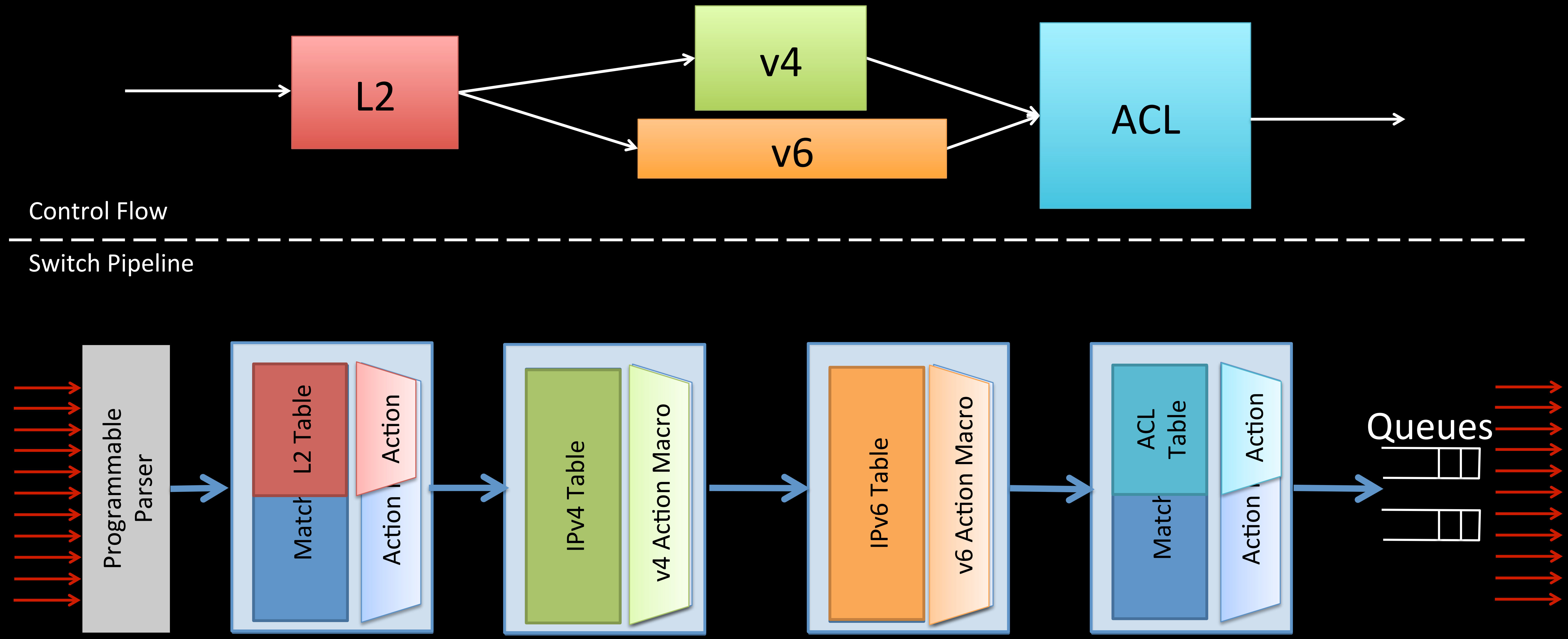
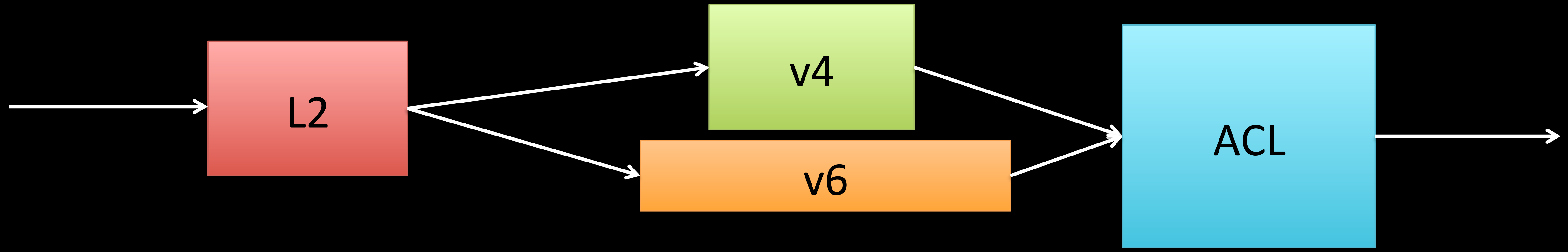
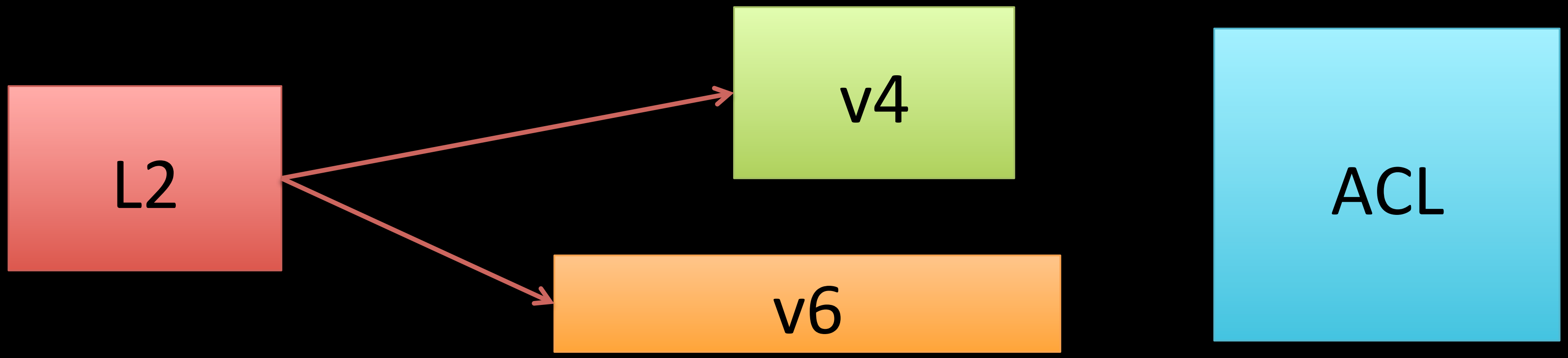


Table Dependency Graph (TDG)

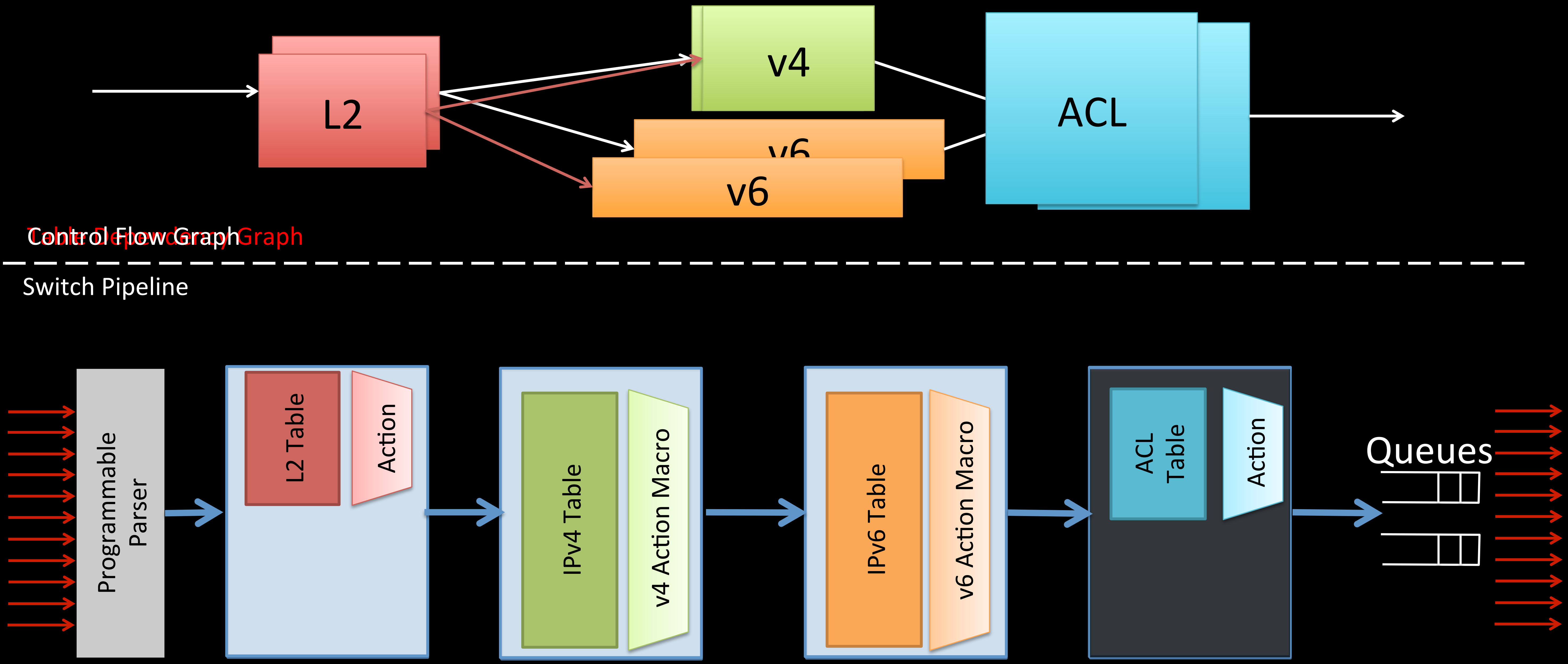


Control Flow Graph

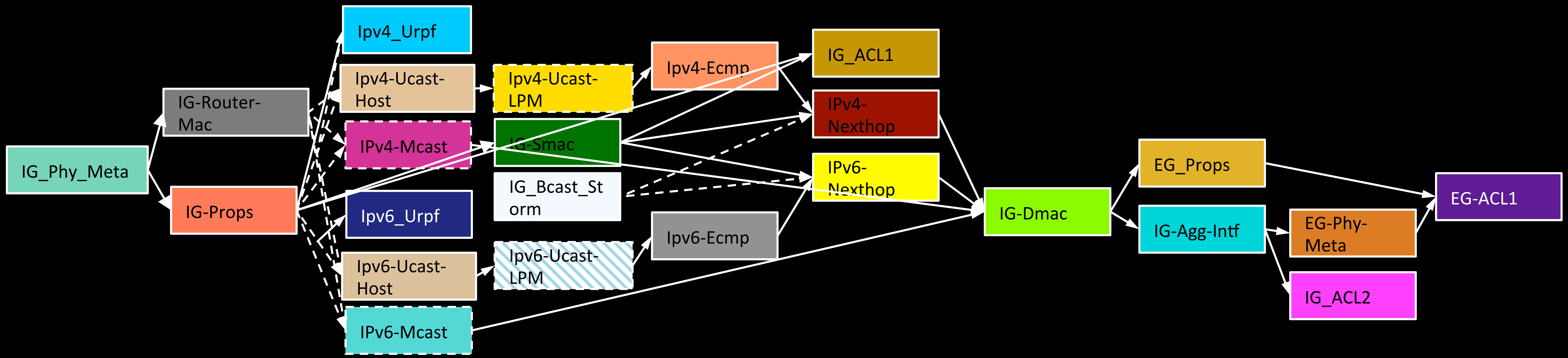
Table Dependency Graph



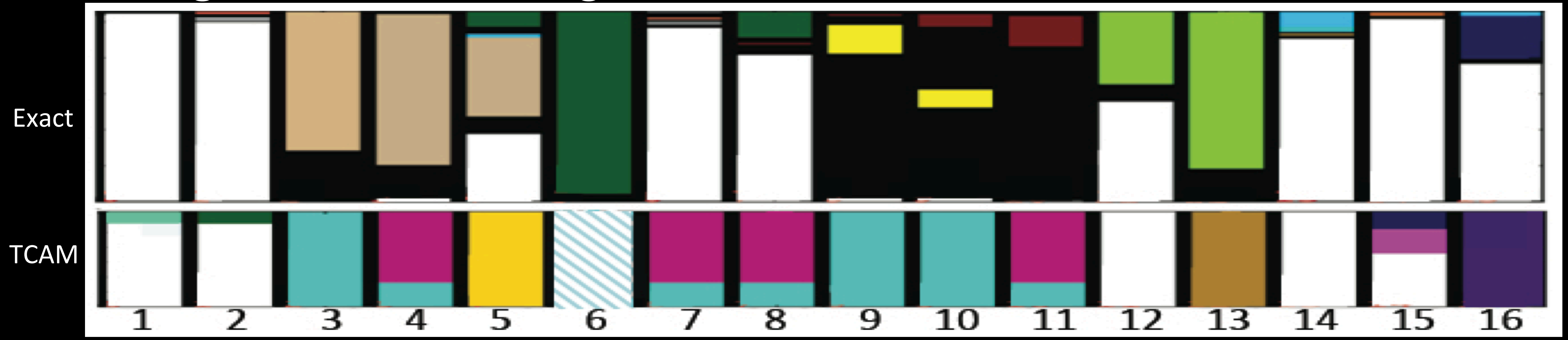
Efficient Mapping: TDG



Example Use Case: Typical TDG



Configuration for 16-stage PISA



Mapping Techniques
[NSDI 2015]

Compare: Greedy Algorithm versus Integer Linear Programming (ILP)

Greedy Algorithm runs 100-times faster

ILP Algorithm uses 30% fewer stages

Recommendations:

1. If enough time, use ILP
2. Else, run ILP offline to find best parameters for Greedy algorithm

PISCES: Protocol Independent Software Hypervisor Switch

Mohammad Shahbaz*, Sean Choi, Jen Rexford*, Nick Feamster*, Ben Pfaff, NM

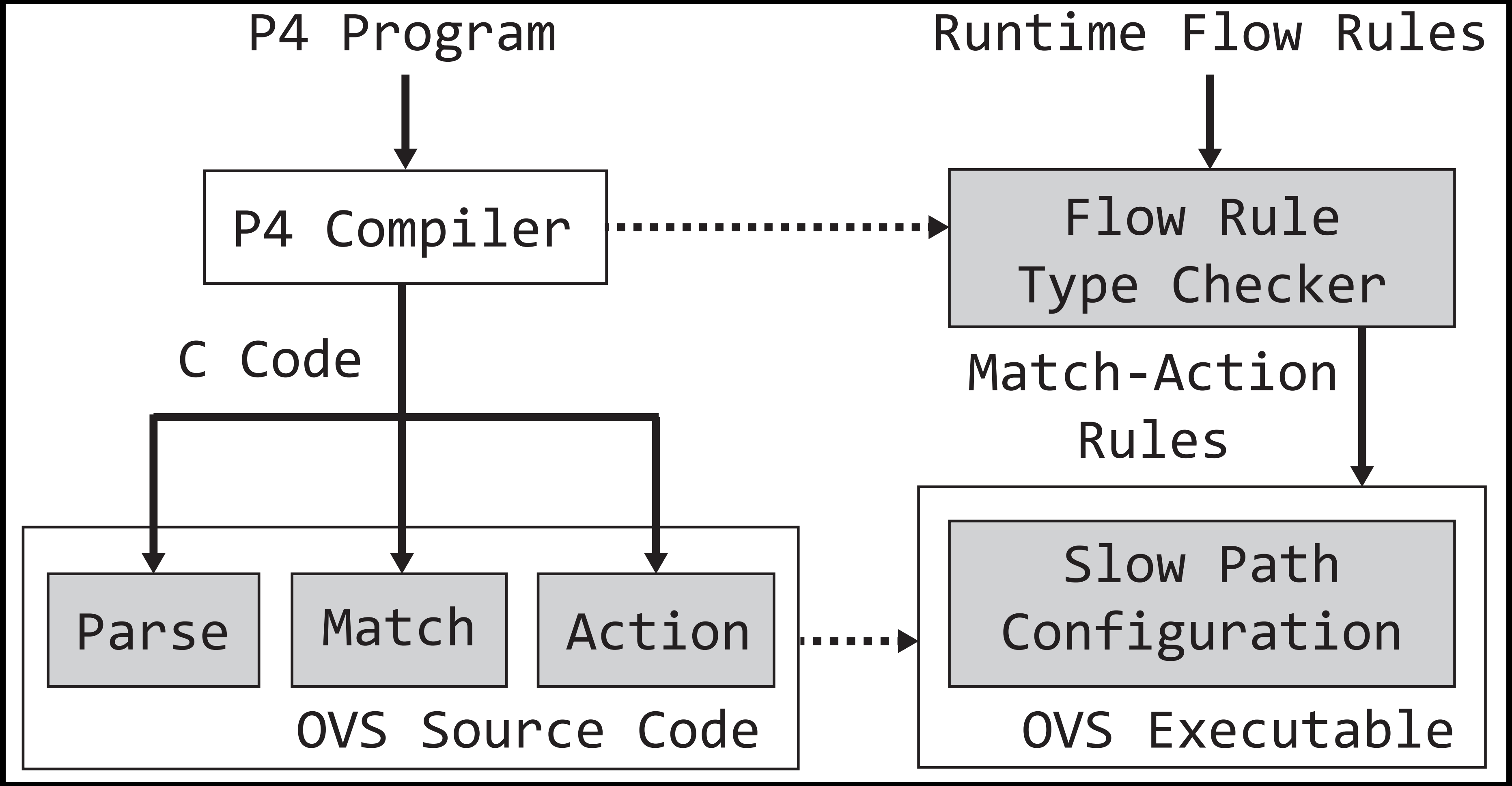
<u>Problem</u>: Adding new protocol feature to OVS is complicated

- Requires domain expertize in kernel programming *and* networking
- Many modules affected
- Long QA and deployment cycle: typically 9 months

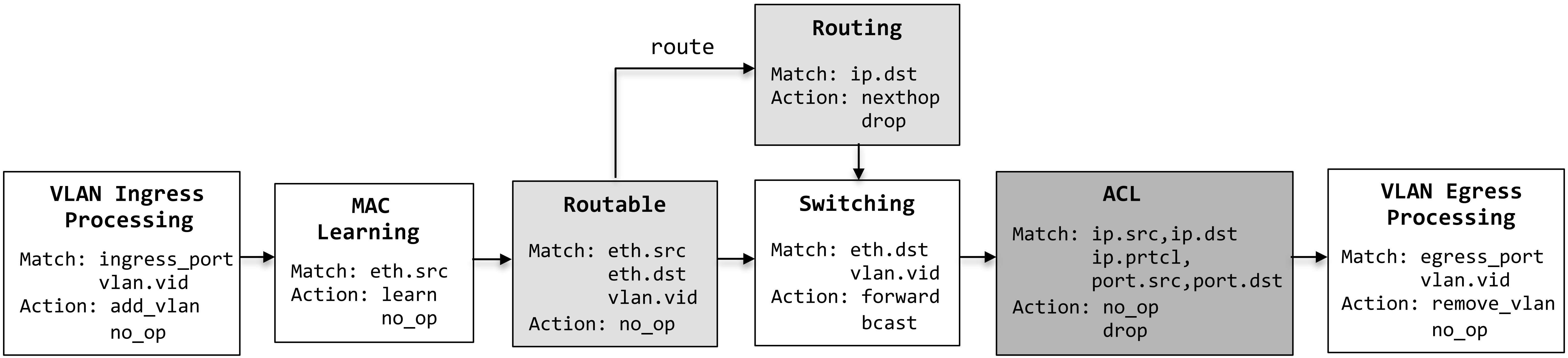
<u>Approach</u>: Specify forwarding behavior in P4; compile to modify OVS

<u>Question</u>: How does the PISCES switch performance compare to OVS?

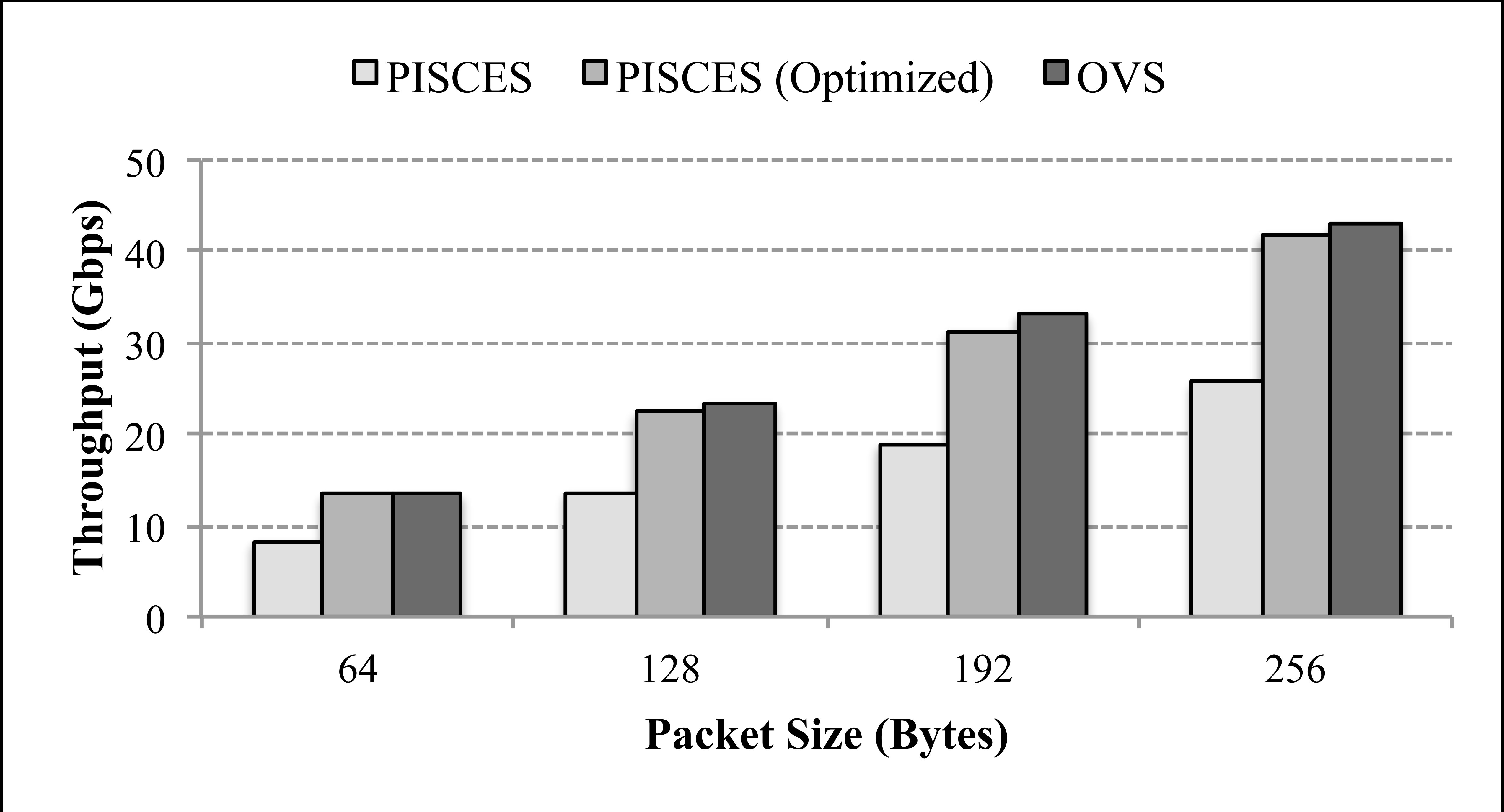
PISCES Architecture



Native OVS expressed in P4



PISCES vs Native OVS



Complexity Comparison

	LOC	Methods	Method Size
Native OVS	14,535	106	137.13
ovs.p4	341	40	8.53

40x reduction in LOC
20x reduction in method size

		Files Changed	Lines Changed
Connection Label	OVS	28	411
	ovs.p4	1	5
Tunnel OAM Flag	OVS	18	170
	ovs.p4	1	6
TCP Flags	OVS	20	370
	ovs.p4	1	4

Code mastery no longer needed

Next Steps

1. Make PISCES available as open-source (May 2016)
2. Accumulate experience, measure reduction in deployment time
3. Develop P4-to-eBPF compiler for kernel forwarding

PERC: Proactive Explicit Rate Control

Lavanya Jose, Stephen Ibanez, Mohammad Alizadeh, George Varghese, Sachin Katti, NM

Problem: Congestion control algorithms in DCs are "reactive"

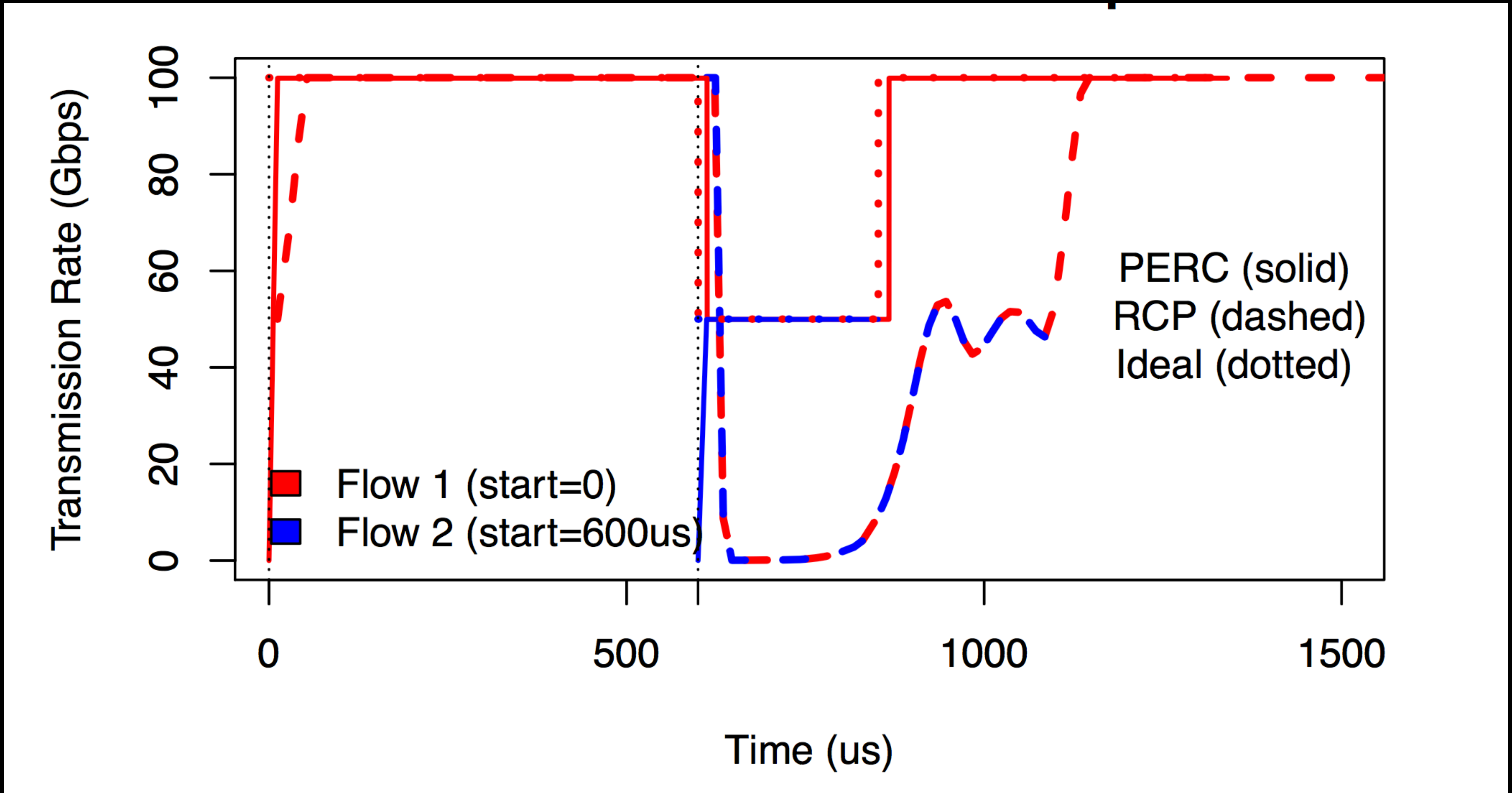
- Typically takes 100 RTTs to converge to fair-share rates (e.g. TCP, RCP, DCTCP)
- The algorithm it doesn't know the answer; it uses successive approximation

Approach: Explicitly calculate the fair-share rates in the forwarding plane

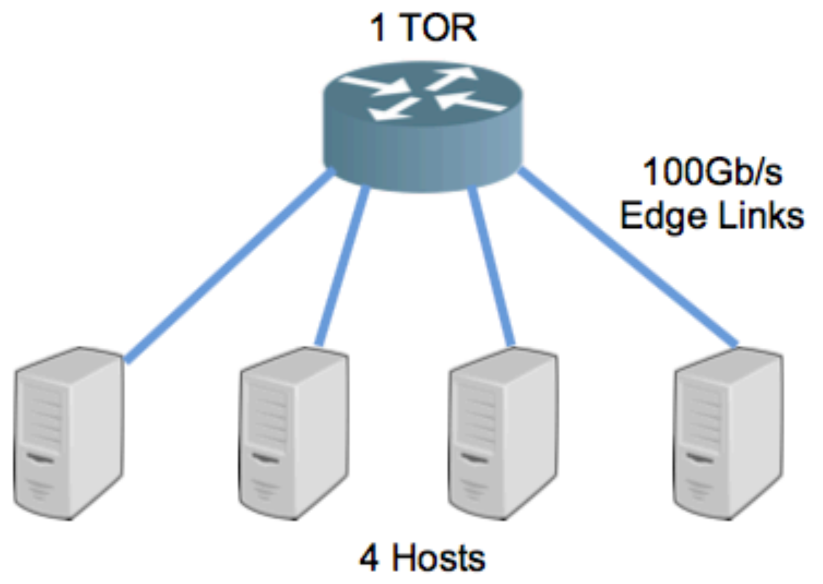
Question: Does it converge much faster? Is it practical?

[Hotnets 2015]

Reactive vs Proactive Algorithms



Performance Results



	RCP	PERC
Median	14 RTTs	4 RTTs
Tail (99^{th})	71 RTTs	10 RTTs

Convergence time determined by dependency chain

Next Steps

Convergence time

- Proof that convergence time equals length of dependency chain
- Reduce measured time to provable minimum

Develop practical algorithm

- Resilient to imperfect and lost update information
- Calculated in PISA-style forwarding plane

<The End>