Welcome.

The P4 Community

Nick McKeown
Stanford University

At some level, we all know *why* we want programmability….

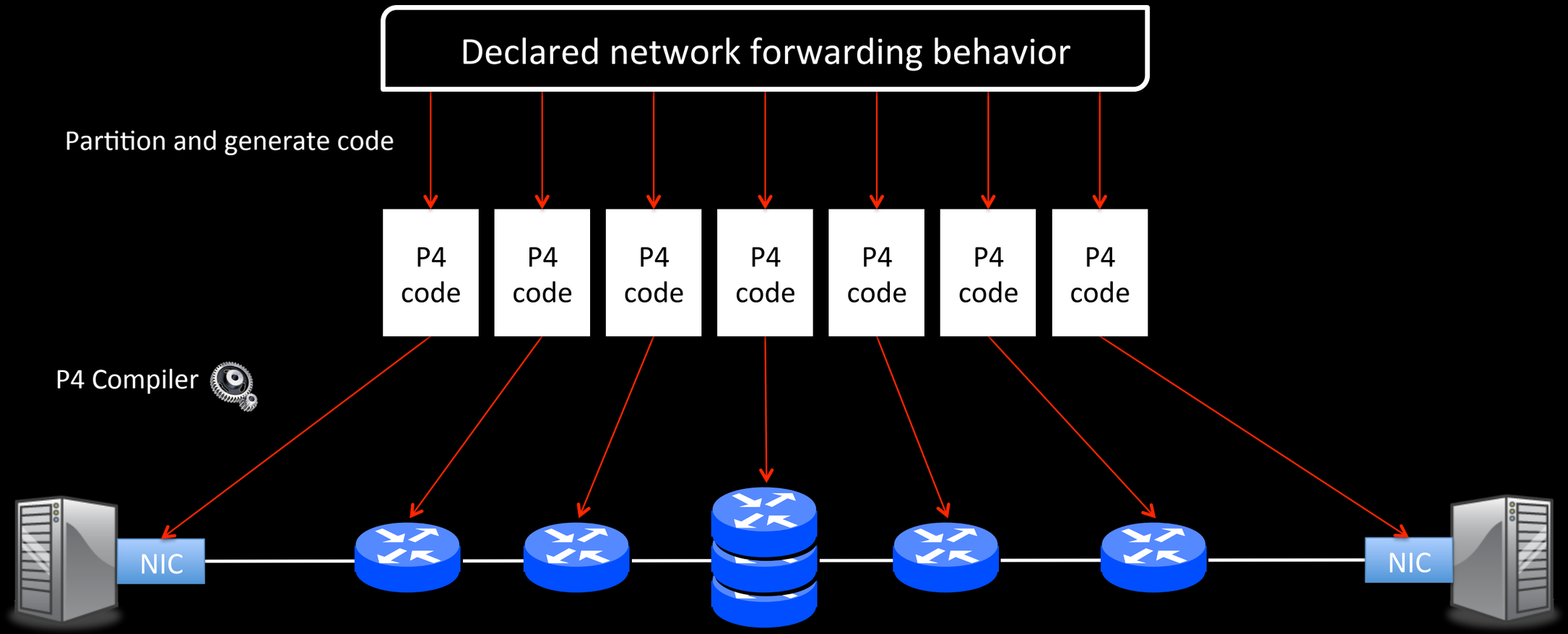
Programmable Forwarding

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. A new abstraction: Programming rather than protocols

A long-term aspiration



Why now?

Domain Specific Processors

1 Computers

Applications

Compiler

CPU

2 Graphics

Applications

Compiler

GPU

3 Signal Processing

Applications

Compiler

DSP

4 Networks

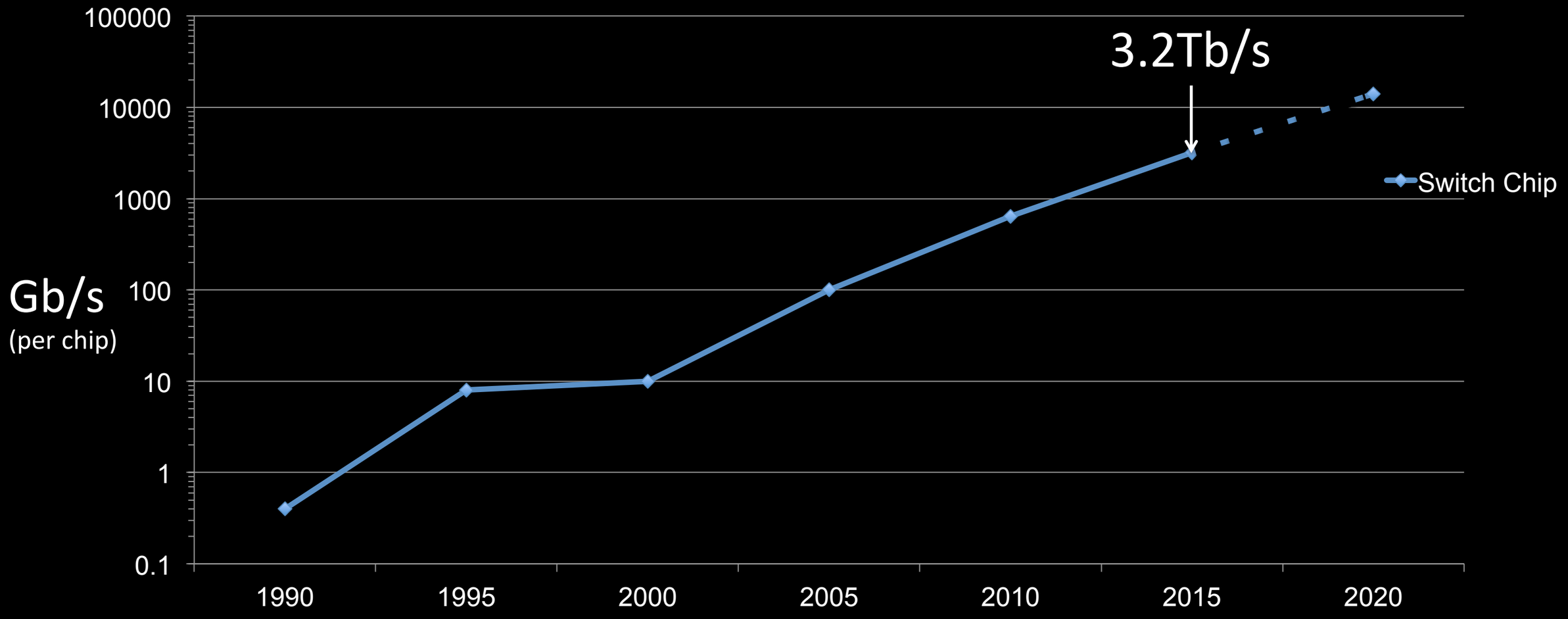
Applications

Compiler

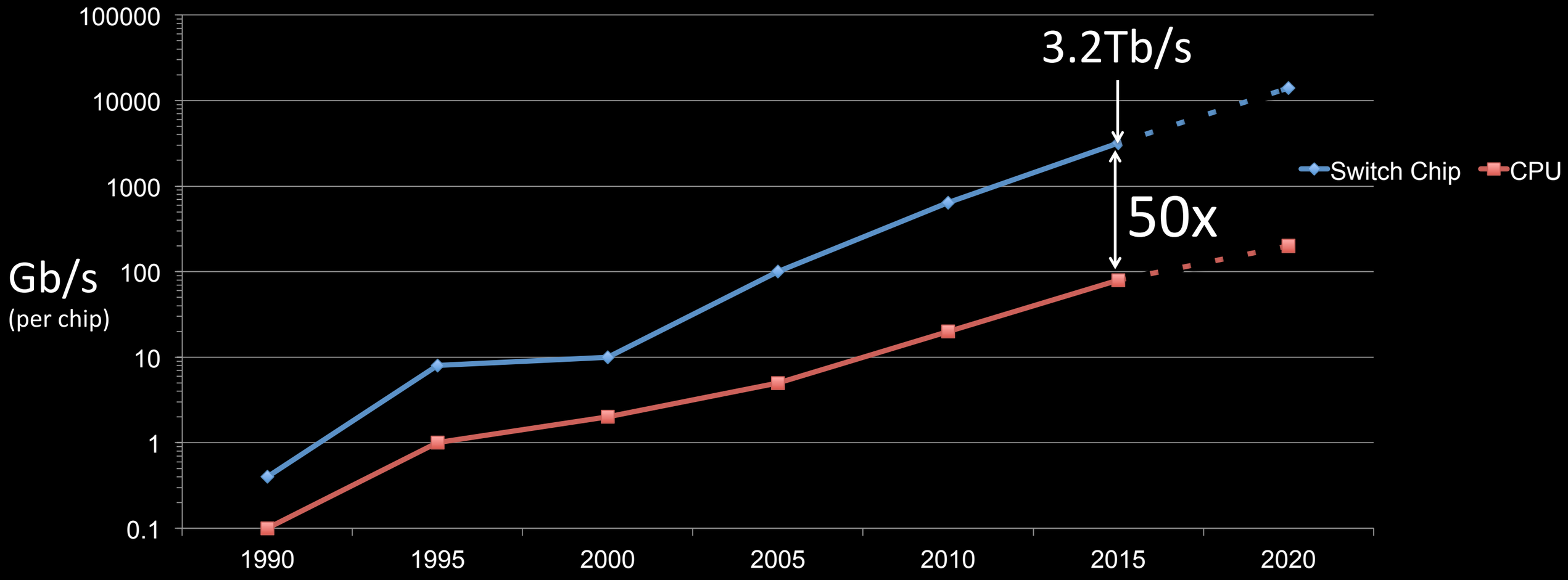
?

But isn’t programmability too expensive in networking?

Packet Forwarding Speeds



Packet Forwarding Speeds



What has been happening

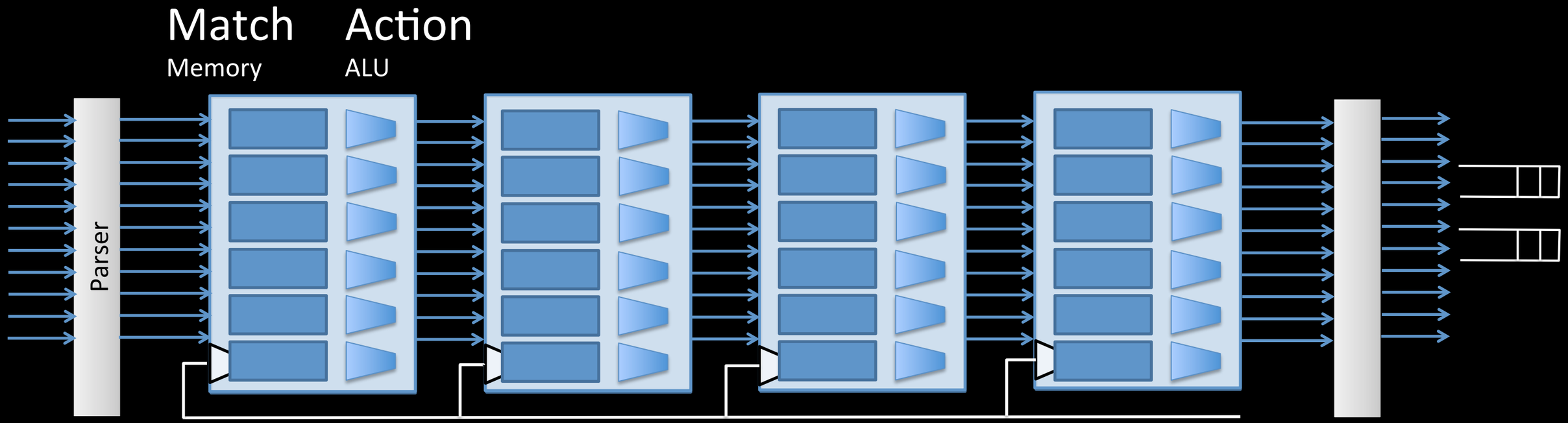
A collective stepping back

- Recognized the value of the “match + action” abstraction
- Picked the basic *plumbing primitives*
- Created more flexible pipelines, less protocol dependent
- Exploited the inherent parallelism

Fulcrum/Intel, Cisco Doppler, Netronome, Xpliant/Cavium, Huawei, “RMT”, etc.

A more general architectural approach is emerging

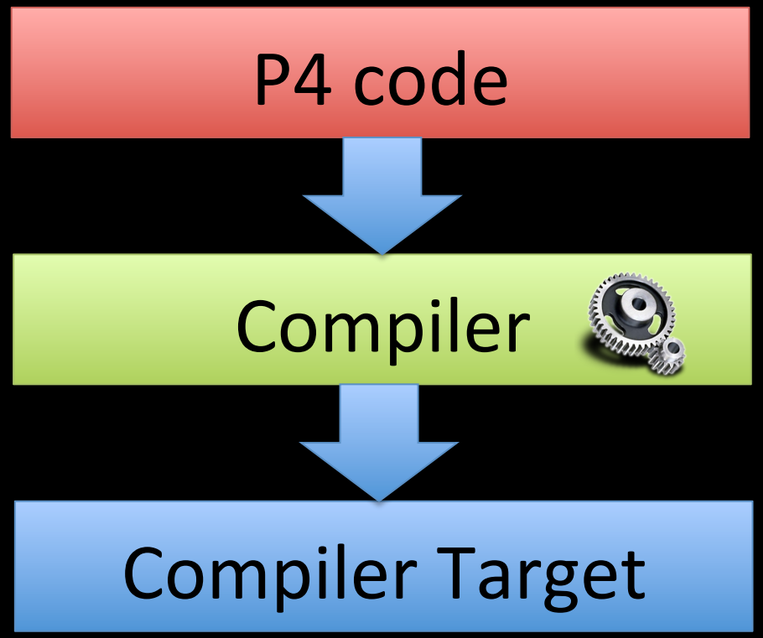
PISA: Protocol Independent Switch Architecture

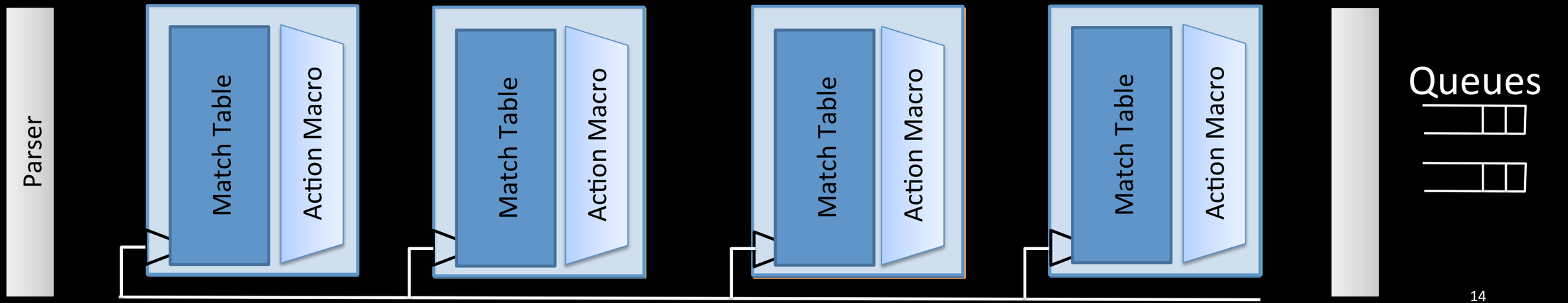


Abstract forwarding model (for P4 v1.0)

Compiler target for P4 programs

P4 and PISA


P4 code
Compiler
Compiler Target


Parser
Match Table
Action Macro
Match Table
Action Macro
Match Table
Action Macro
Match Table
Action Macro
Queues

P4 and PISA

Parser

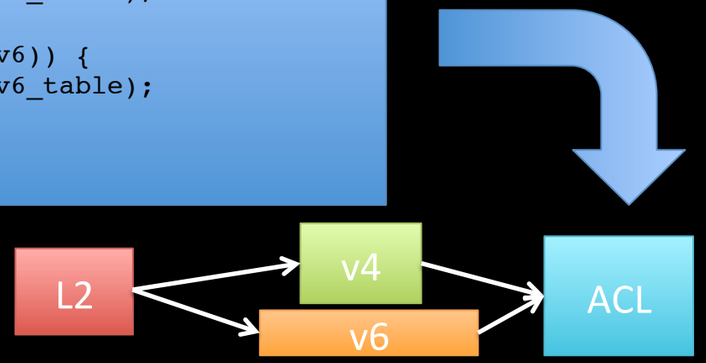
```
parser parse_ethernet {
  extract(ethernet);
select(latest.etherType) {
    0x800   : parse_ipv4;
    0x86DD : parse_ipv6;
  }
}
```

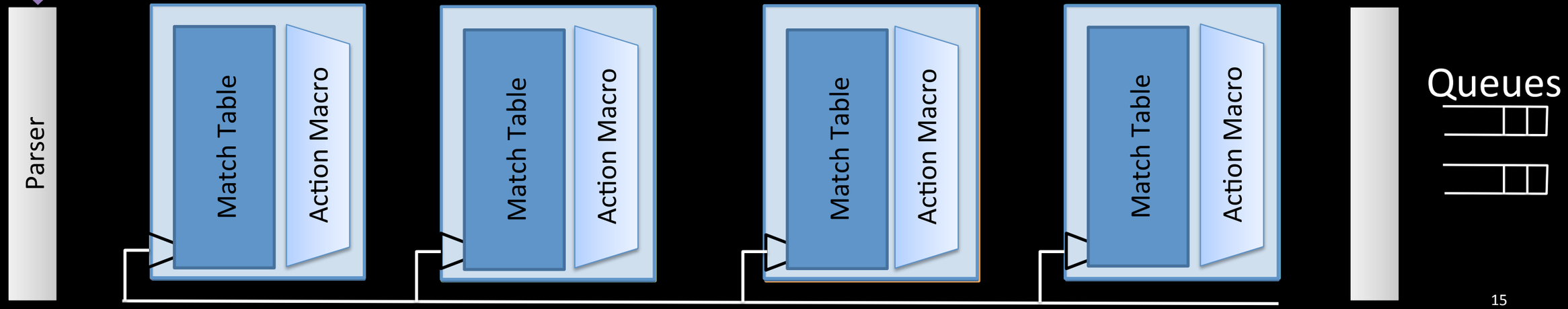
Match Action Tables

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

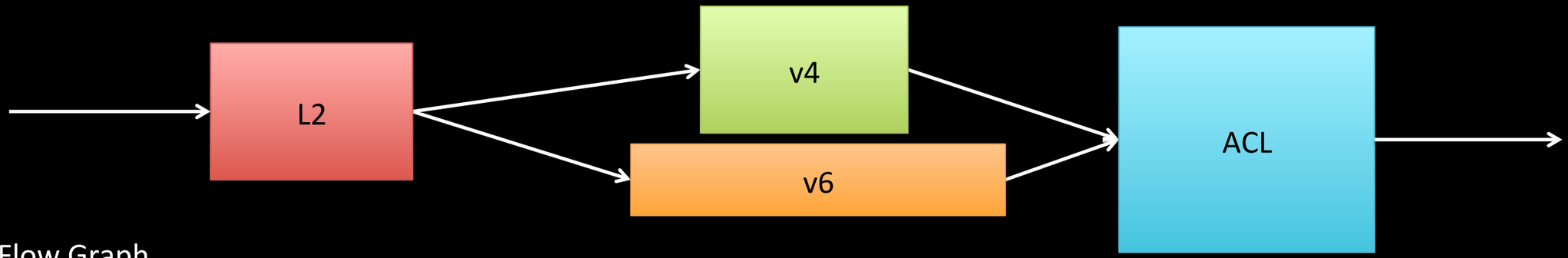
Control Flow Graph

```
control ingress
{
    apply(l2_table);
    if (valid(ipv4)) {
        apply(ipv4_table);
    }
    if (valid(ipv6)) {
        apply(ipv6_table);
    }
    apply (acl);
}
```



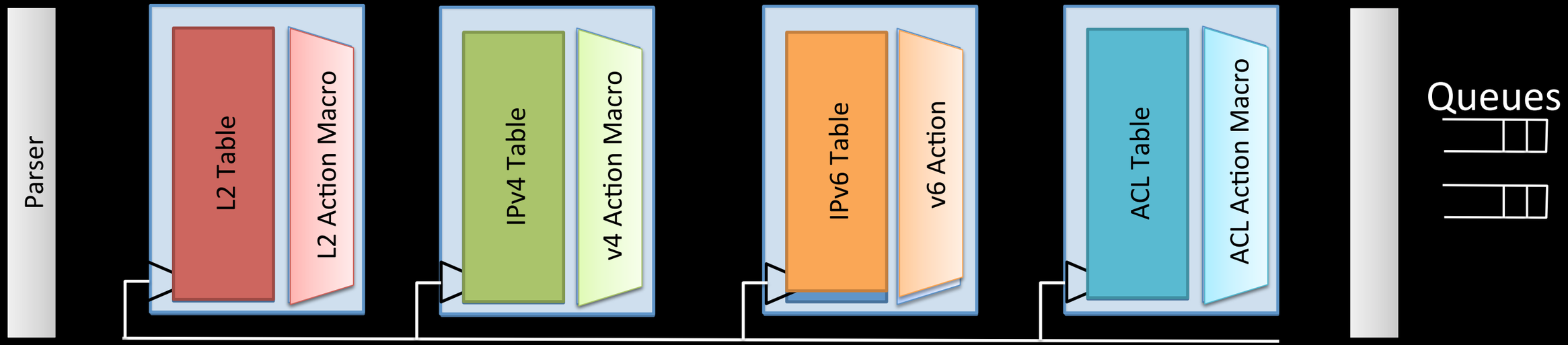


Mapping Control Flow to PISA Target



Control Flow Graph

Switch Pipeline

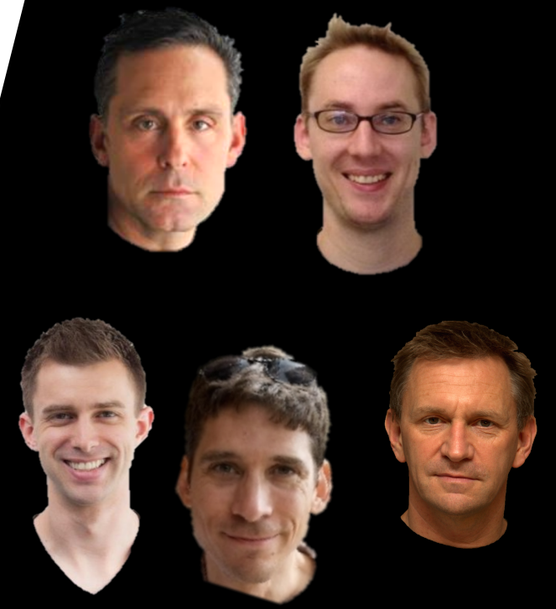


Why “Protocol Independence” ?

- Programmable in the field
- IP belongs to the programmer
- Breaks the stranglehold of the “lock-in” APIs
- Faster innovation

Can we create a common language?

Existing work: packetC, Netcore, Frenetic, Pyretic, NetKAT, …



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 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.

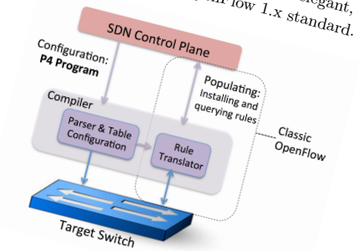
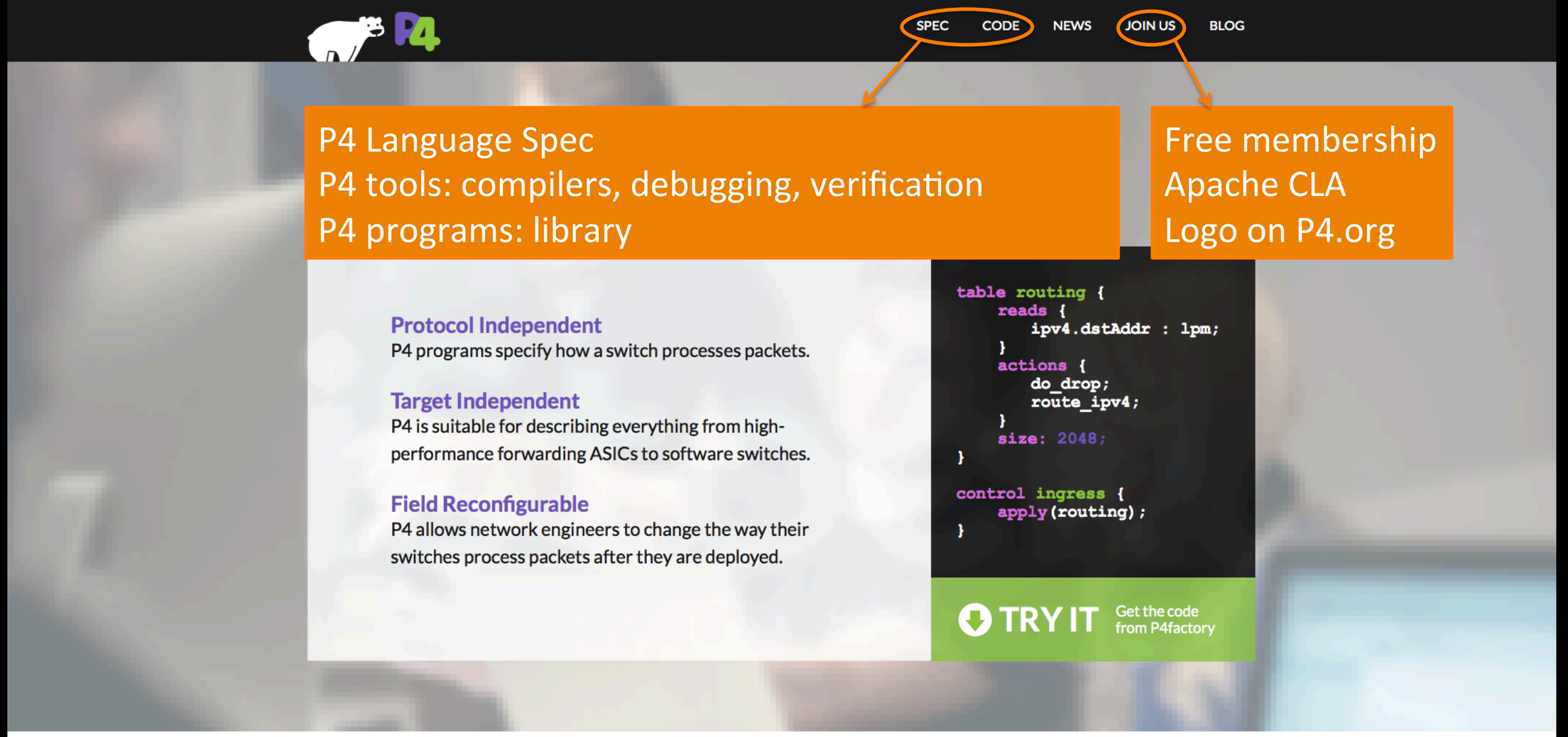


Figure 1: P4 is a language to configure switches.

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). P4 has three goals: (1) it how packets a


SPEC
CODE
NEWS
JOIN US
BLOG
P4 Language Spec
P4 tools: compilers, debugging, verification
P4 programs: library
Free membership
Apache CLA
Logo on P4.org
Protocol Independent
P4 programs specify how a switch processes packets.
Target Independent
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.
table routing {
reads {
ipv4.dstAddr : lpm;
}
actions {
do_drop;
route_ipv4;
}
size: 2048;
}
control ingress {
apply(routing);
}
TRY IT
Get the code from P4factory
SEE HOW P4 WORKS
- 12 MIN
STAY CONNECTED
Subscribe to our announcements mailing list
SUBSCRIBE
LEGAL
PRIVACY
©2015 P4 Language Consortium

INDUSTRY MEMBERS

UNIVERSITY MEMBERS

<The End>