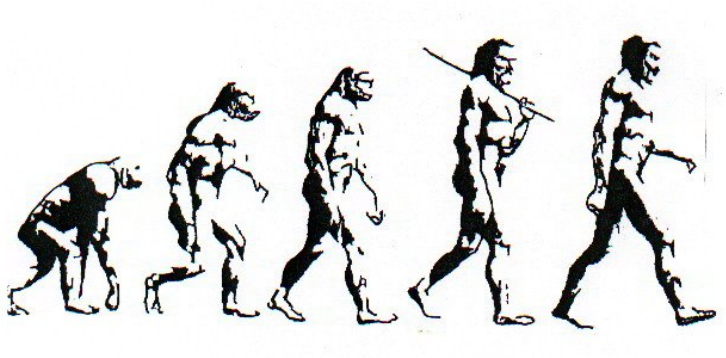


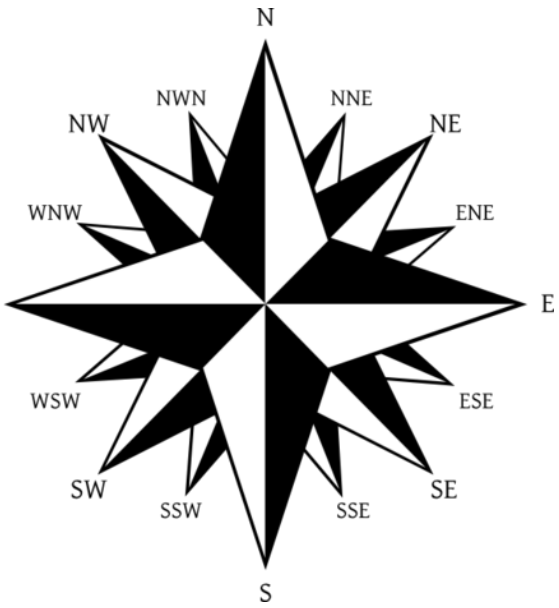
Evolving P4
- a proposal -

Mihai Budiu
P4.org and BAREFOOT NETWORKS

P4 Workshop, Stanford
June 4, 2015

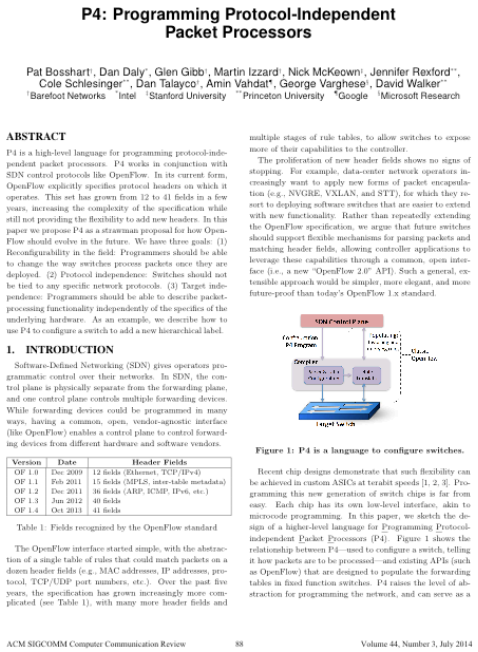
With input from

- Gordon Brebner
- Dan Talayco
- George Varghese
- Jennifer Rexford
- David Walker
- Nick McKeown



- Chris Dodd
- Leo Alterman
- Ajeer Pudyiapura
- Alexei Storovaitov
- Dan Daly
- Changhoon Kim

A Brief History of P4



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 more header fields and 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). Figure 1 shows the relationship between P4—used to configure a switch, telling it how packets are to be processed—and existing APIs (such as OpenFlow) that are designed to populate the forwarding tables in fixed function switches. P4 raises the level of abstraction for programming the network, and can serve as a

published July 2014
SIGCOMM CCR

The P4 Language Specification

Version 1.0.2

March 3, 2015

The P4 Language Consortium

1 Introduction

P4 is a declarative language for expressing how packets are processed by the pipeline of a network forwarding element such as a switch, NIC, router or network function appliance. It is based upon an abstract forwarding model consisting of a parser and a set of match+action table resources, divided between ingress and egress. The parser identifies the headers present in each incoming packet. Each match+action table performs a lookup on a subset of header fields and applies the actions corresponding to the first match within each table. Figure 1 shows this model.

P4 itself is protocol independent but allows for the expression of forwarding plane protocols. A P4 program specifies the following for each forwarding element.

- *Header definitions:* the format (the set of fields and their sizes) of each header within a packet.
- *Parse graph:* the permitted header sequences within packets.
- *Table definitions:* the type of lookup to perform, the input fields to use, the actions that may be applied, and the dimensions of each table.
- *Action definitions:* compound actions composed from a set of primitive actions.
- *Pipeline layout and control flow:* the layout of tables within the pipeline and the packet flow through the pipeline.

P4 addresses the configuration of a forwarding element. Once configured, tables may be populated and packet processing takes place. These post-configuration operations are referred to as "run time" in this document. This does not preclude updating a forwarding element's configuration while it is running.

1.1 The P4 Abstract Model

The following diagram shows a high level representation of the P4 abstract model.

The P4 machine operates with only a few simple rules.

March 2015
p4.org

FOSS release (Apache2 license)

1. P4 front-end compiler
2. Executable P4 software switch
3. P4 code examples

?
p4.org

Why P4 has momentum

- Natural for network programmers
- Right abstraction level
  - Parser, MAU, deparser, table, headers, metadata
- High-level, yet expressive
  - No loops, no recursion, no pointers, no FP
  - Bit-level packet manipulation
  - Tables as primitives

The P4 tension

Universal

Customizable

P4 v1:
Fixed Abstract Forwarding Model

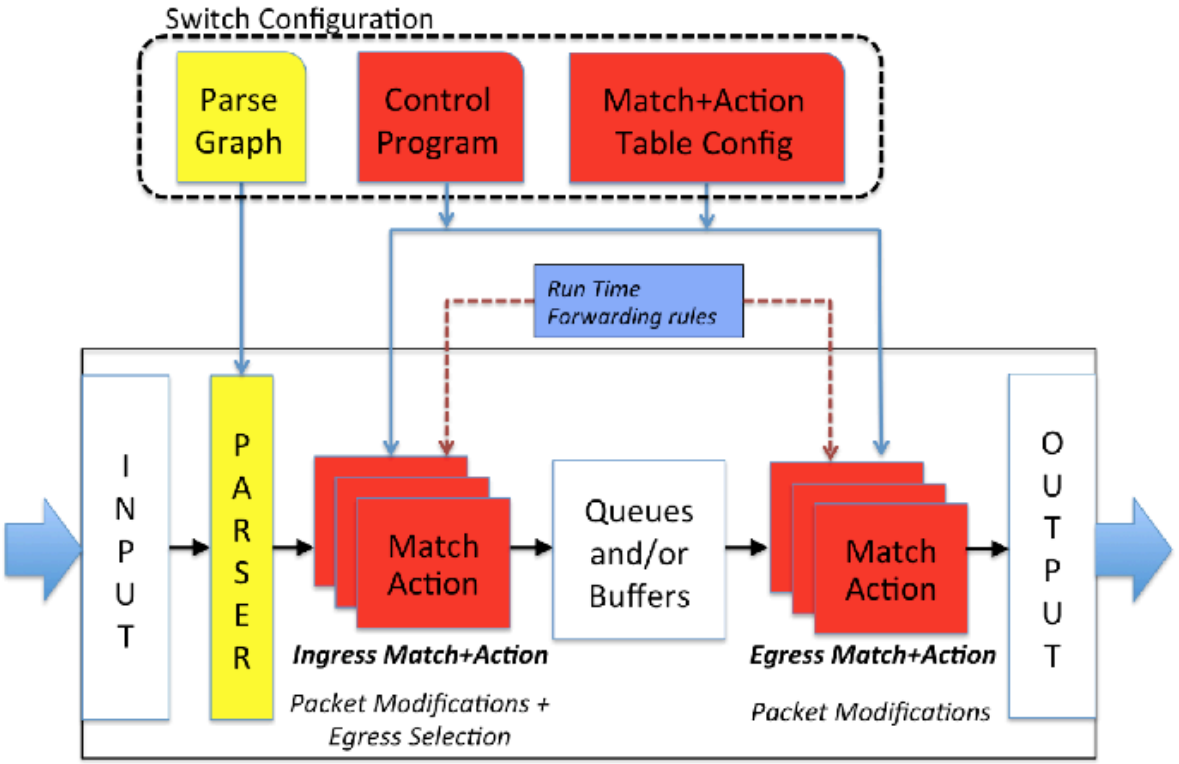


Figure 1: Abstract Forwarding Model

P4 v1: Details

Identifier	Exception Event
p4_pe_index_out_of_bounds	A header stack array in… clared bound.
p4_pe_out_of_packet	There were no… m- plete an e…
p4_pe_header_too_long	A c… the declared
p4_pe_header_too_short	…was less than the min- …ixed length portion of the
p4_pe_unh…	…ment had no default specified but the …on value was not in the case list.
…	…ecksum error was detected.
	This is not an exception itself, but allows the pro- grammer to define a handler to specify the default behavior if no handler for the condition exists.

Parser exceptions

Table 2: Standard Parser Exceptions

Action profiles

Mirroring

field_list_calculation

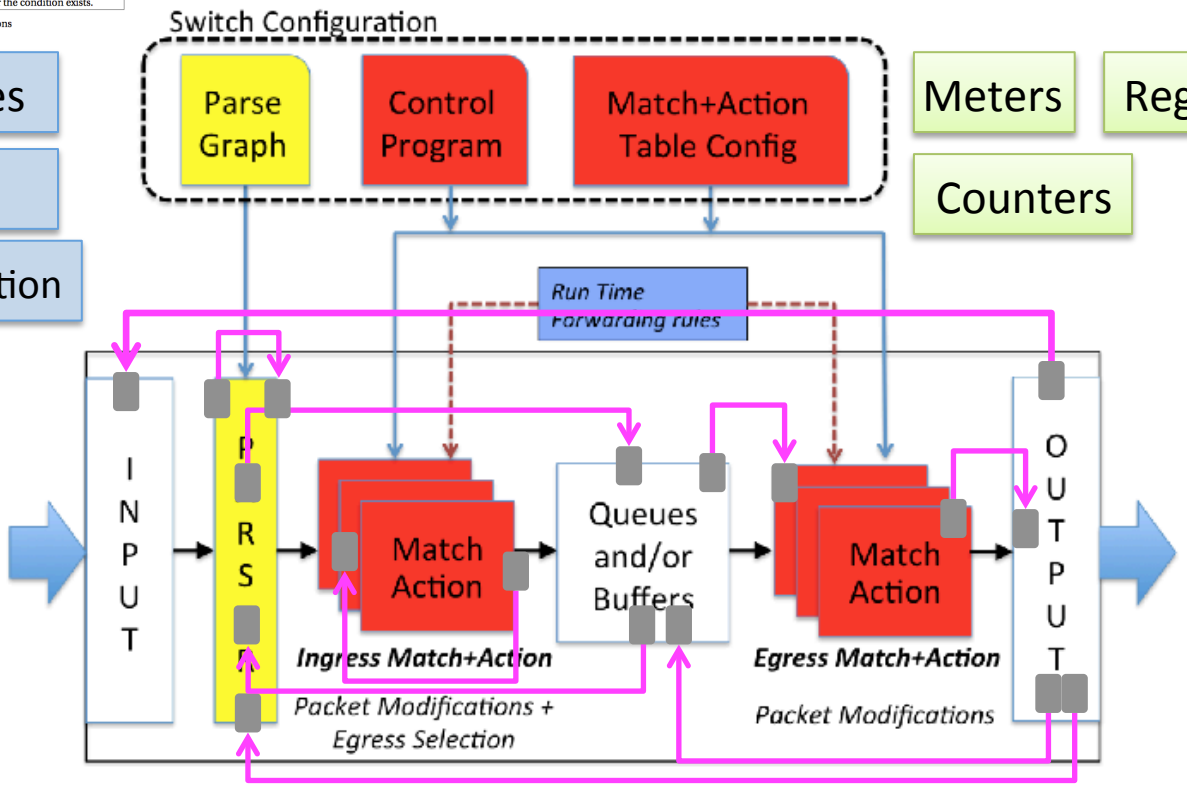
xor16

csum16

crc16

crc32

programmable

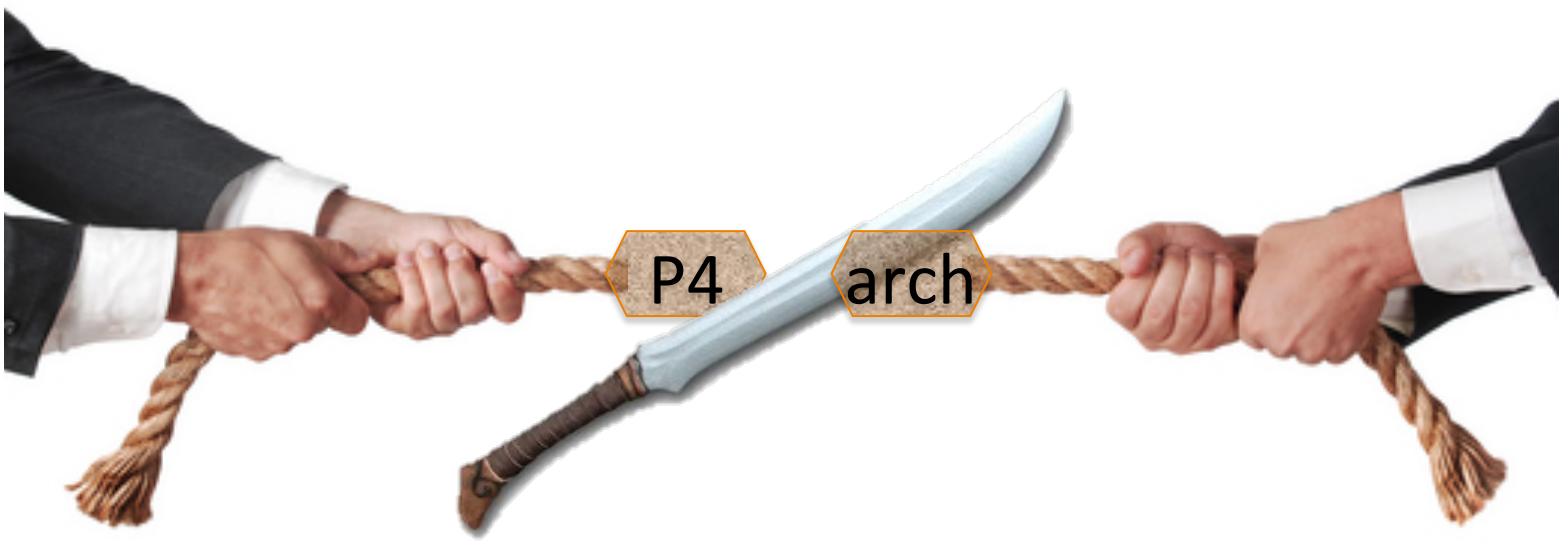


Field	Notes
ingress_port	The port on which the packet arrived. Set prior to parsing. Always defined. Read only
packet_length	The number of bytes in the par… Ethernet, does not include the CRC. Set … Can- not be used for matching or … if the switch is in "cut-throug…
egress_spec	Specification of an egres… match+action during i… the "intended" egress… physical port(s) (ser… physical port, a lo… LAG, a route, or … group.
egress_port	The physica… committed… by the B… egress … Read
egress_instance	An …
instance_type	… of instance of the packet: … one … cated … or flags or type value; for example, could a … recirculated? Do we need a counter too?
parser_…	… of the parser. 0 means no error. Otherwise, …alue indicates what error occurred during pars- …g. Specific representation is TBD.
pars…	…f a parser error occurred, this is an indication of the location in the parser program where the error oc- curred. Specific representation is TBD.

Intrinsic metadata

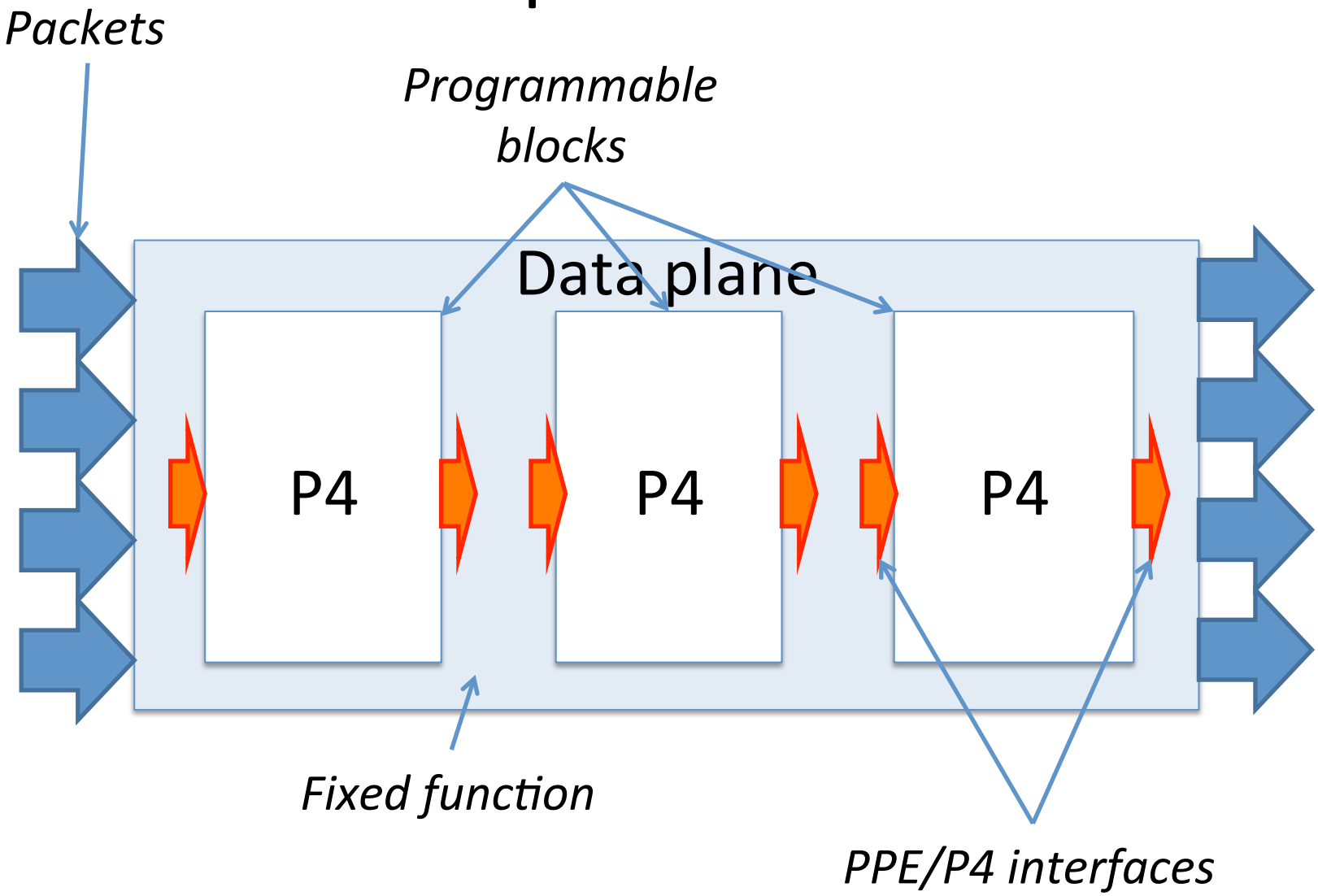
Table 3: Standard Intrinsic Metadata Fields

Divide and conquer

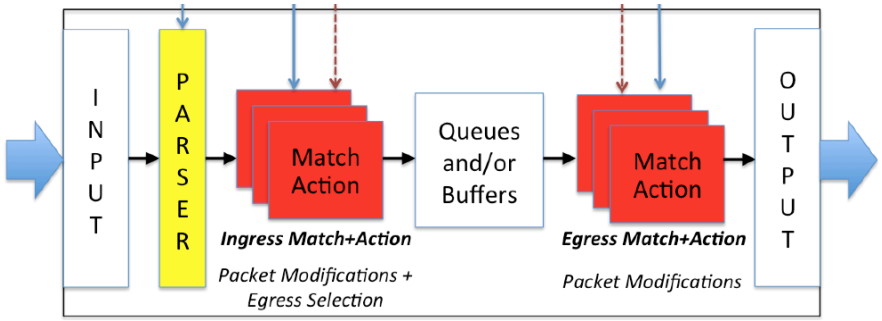


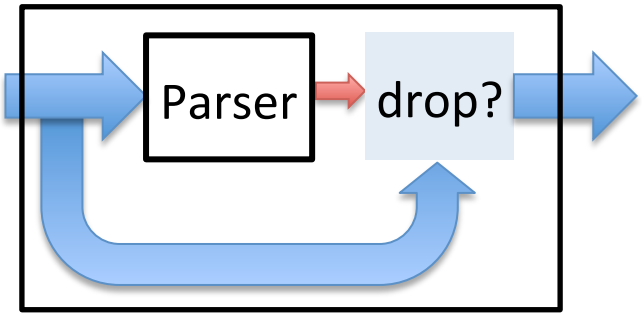
- Separate language definition from architecture definition
- Evolve them independently

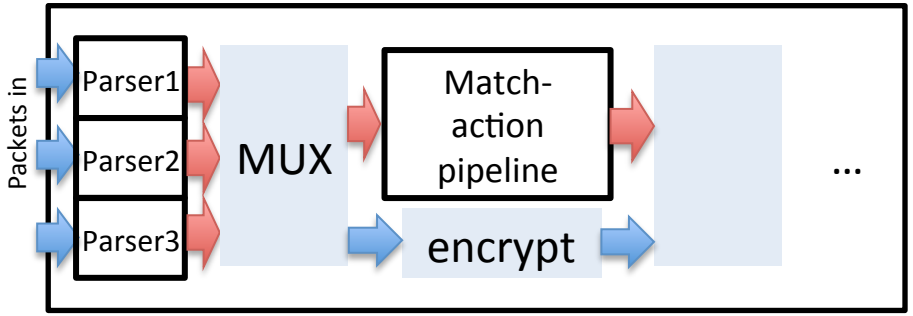
Generic Programmable Packet Engine (PPE) Dataplane Model



Bonus:
P4 Support for multiple architectures

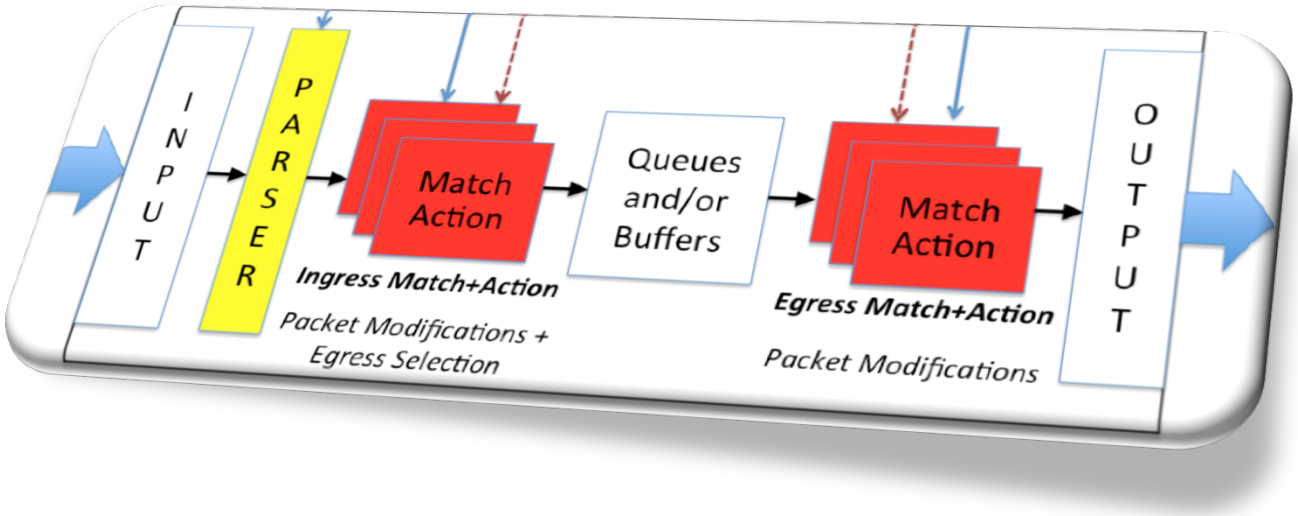






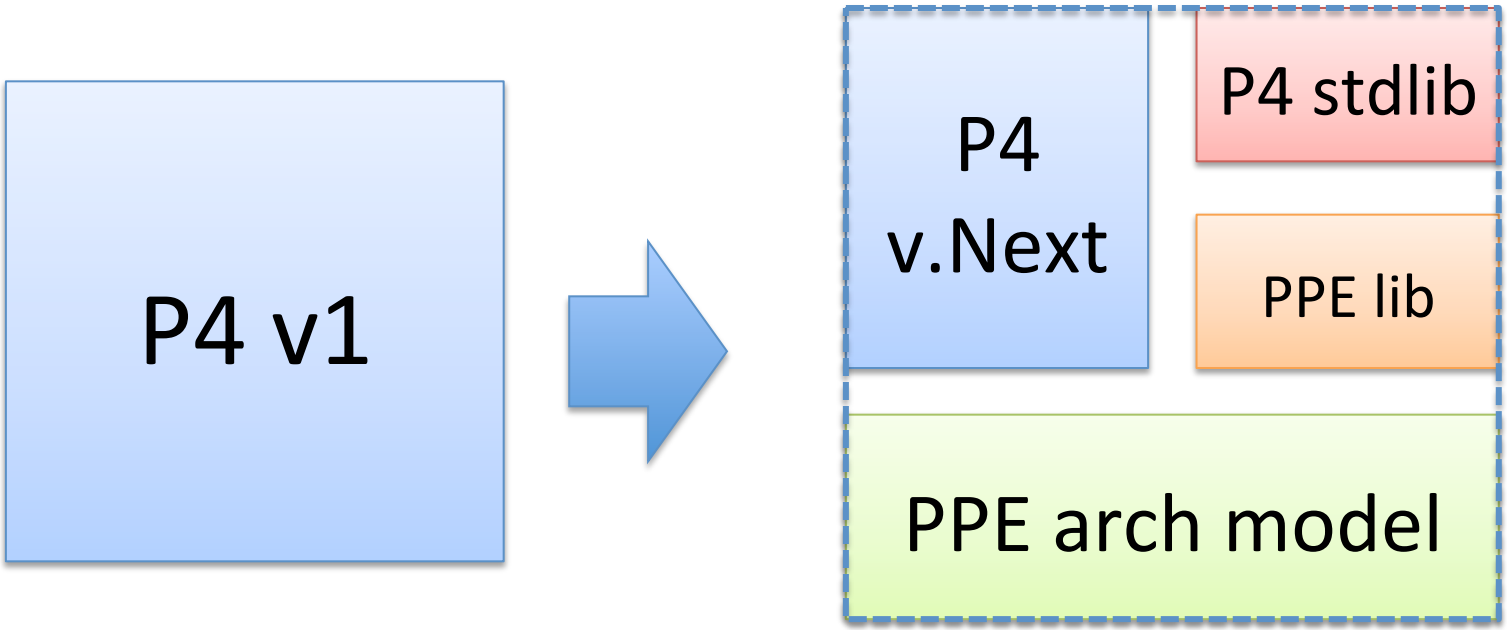
Portability

- P4 programs are portable *between architectures that implement the same PPE model*

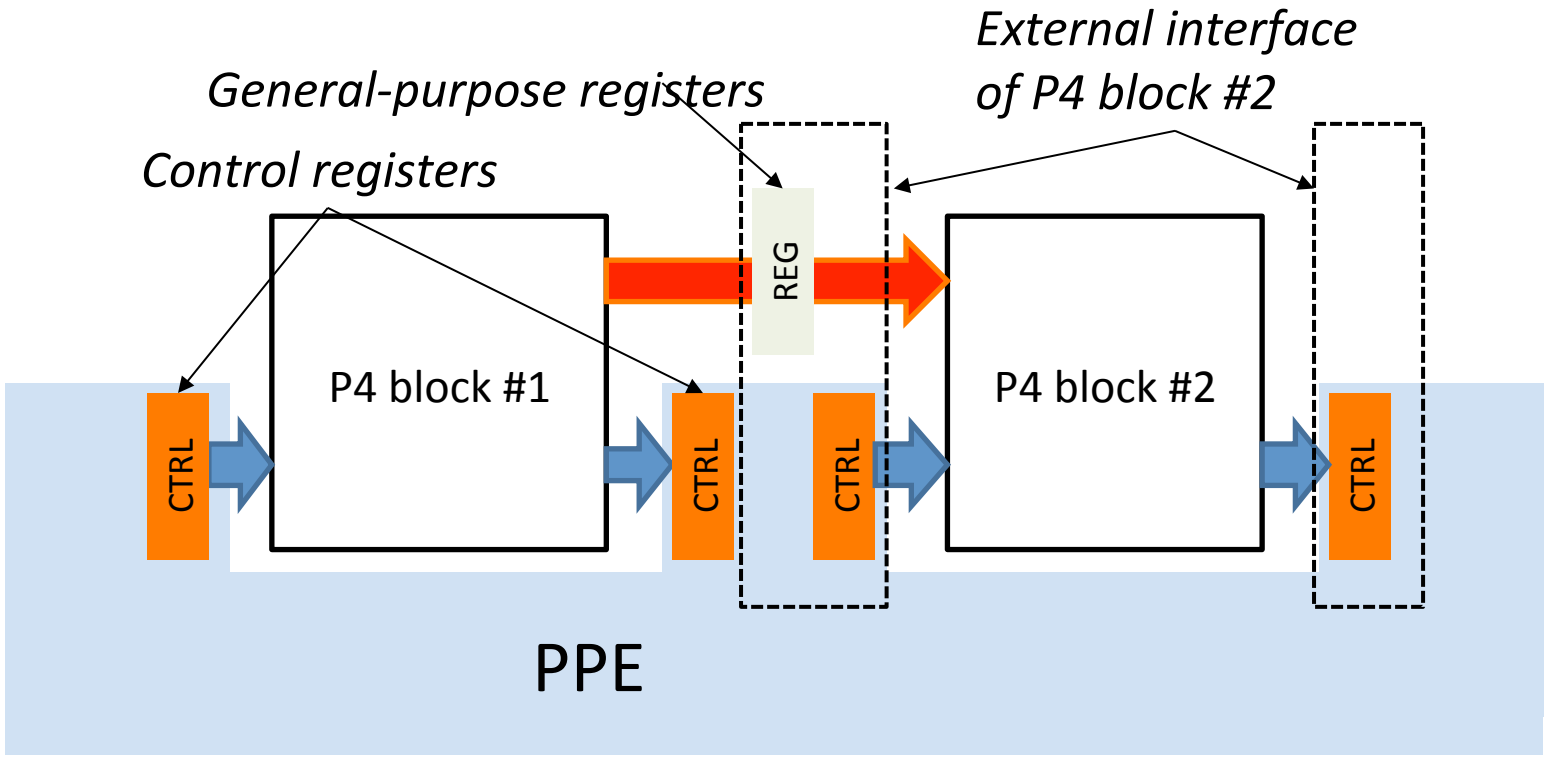


- *Community should evolve a standard model, starting from the P4 v1 switch model*

Evolution roadmap



PPE arch model



P4 standard library

- Packet operations (extract, deparse, etc.)
- Checksums
- Error codes
- Table types (e.g. hash-table)
- Standard actions (e.g. nop)

- Stdlib evolution driven by P4 community

PPE libraries

- Written by target manufacturers
- Define hardware-software interfaces
- Target-specific hardware block interfaces
  - Counters, meters, etc.
  - Custom tables (e.g. Tries)
  - Custom actions

P4 v1. => P4 v.next

- Parser
- Match-Action Units
- Tables
- Actions
- Headers
- Types
- Control-flow

P4 v.next

- Typing
- Modularity
- Error handling
- Simplicity

P4.org members:
Please contribute your ideas!

Mailing list: p4-design@p4.org

(P4.org membership is free and open to anyone)