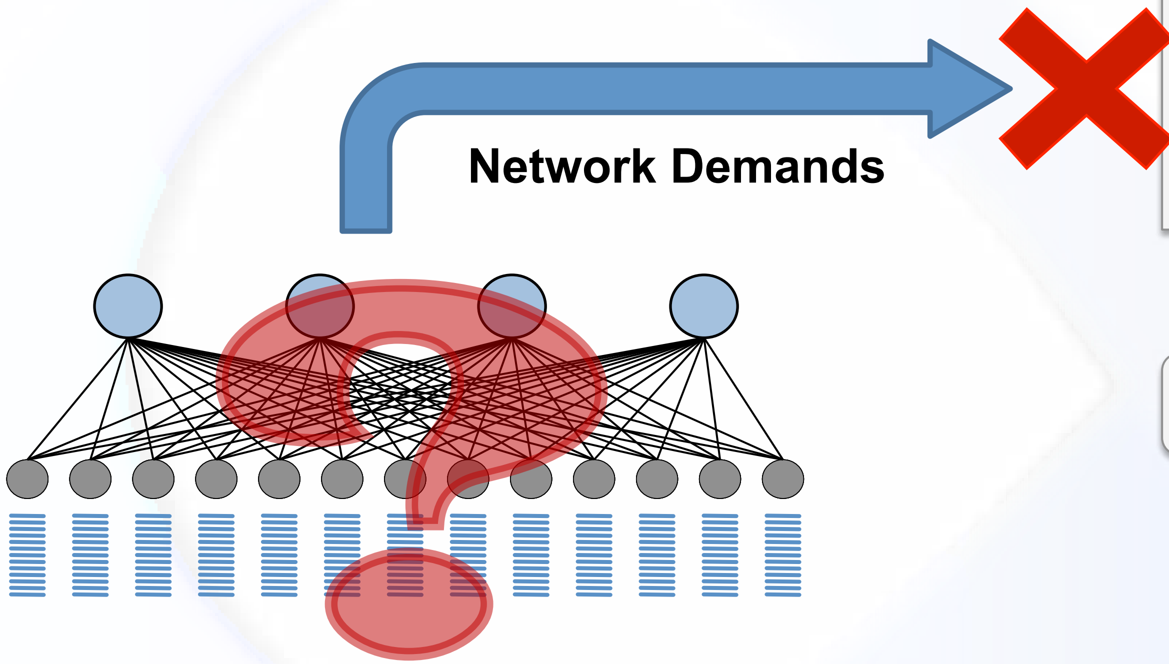
P4 Tutorial

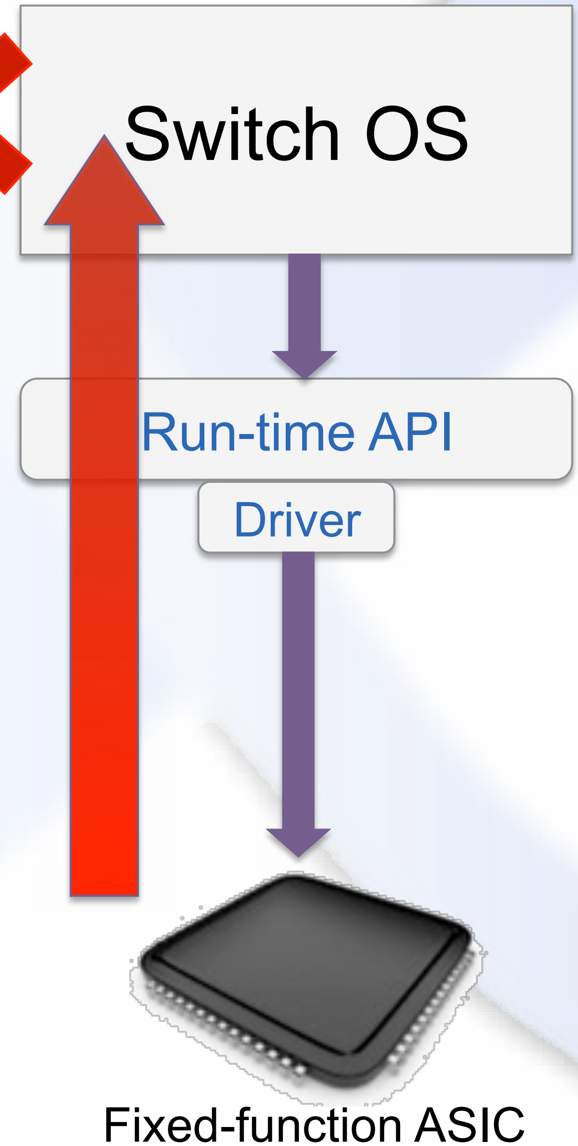
Vladimir Gurevich
November 2015

P4 Introduction

BAREFOOT NETWORKS

Status Quo: Bottom-up design

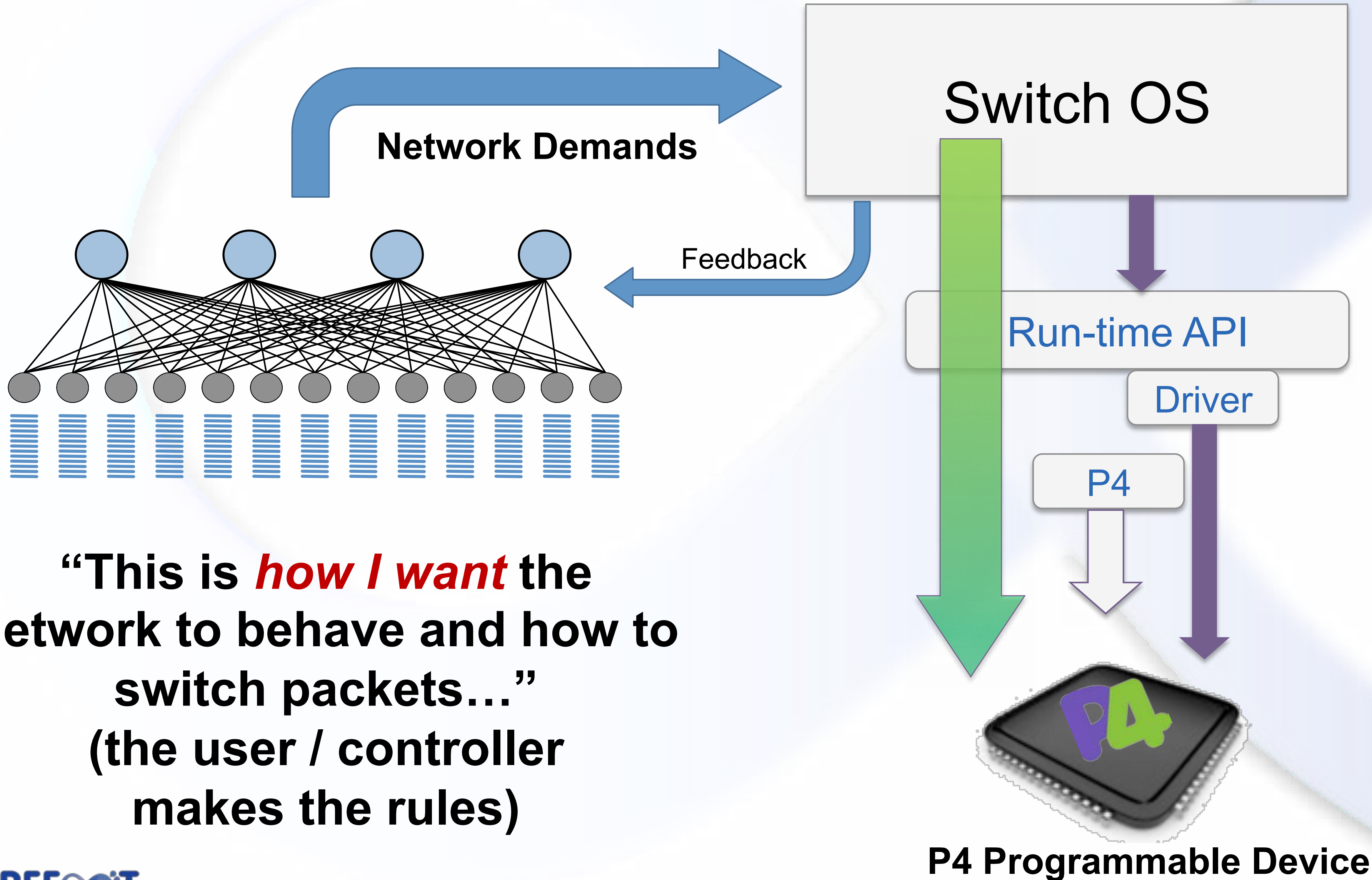




"This is *how I know* to process packets"
(i.e. the ASIC datasheet makes the rules)

BAREFOOT NETWORKS

A Better Approach: Top-down design



"This is *how I want* the network to behave and how to switch packets…"
(the user / controller makes the rules)

Programmable Network Devices

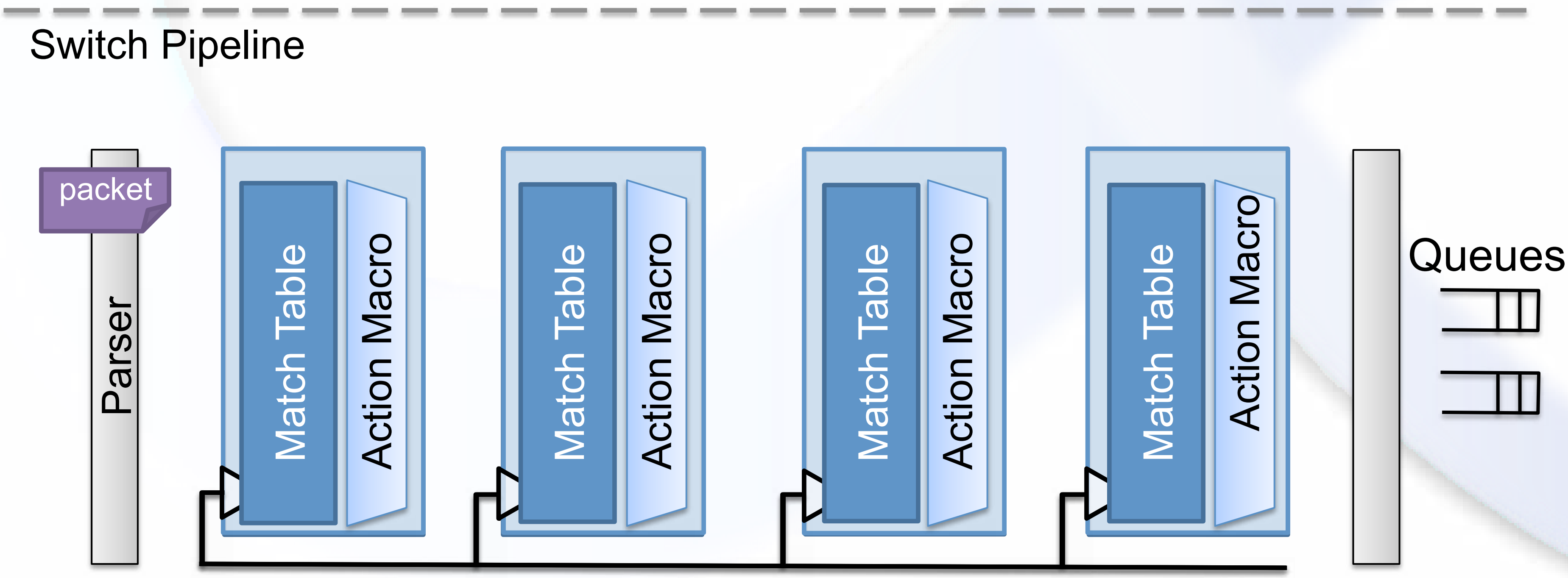
- **PISA: Flexible Match+Action ASICs**
  - Intel Flexpipe, Cisco Doppler, Cavium (Xpliant), …
- **NPU**
  - EZchip, Netronome, …
- **CPU**
  - Open Vswitch, ...
- **FPGA**
  - Xilinx, ...

“Top-down” These devices let us tell them how to process packets.

BAREFOOT NETWORKS

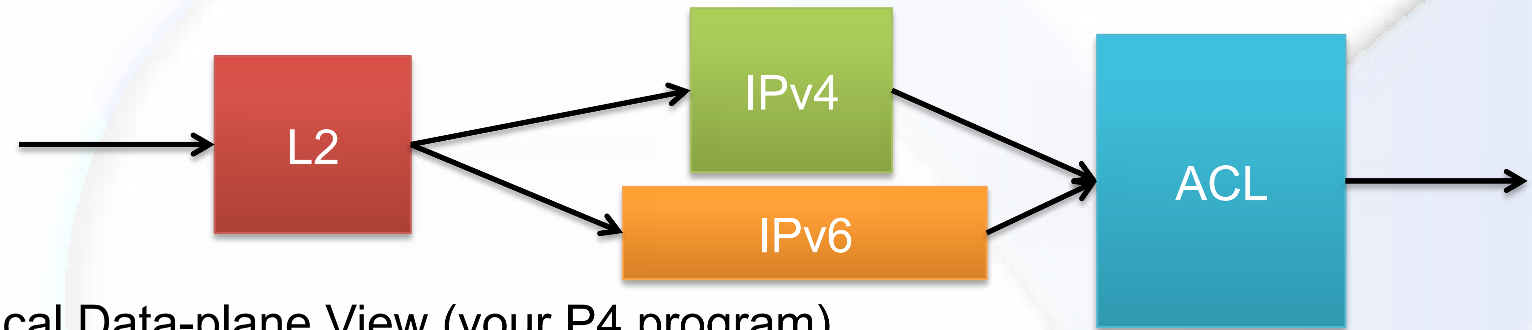
Why we call it Protocol Independent Packet Processing

Protocol-Independent Switch Architecture (PISA)



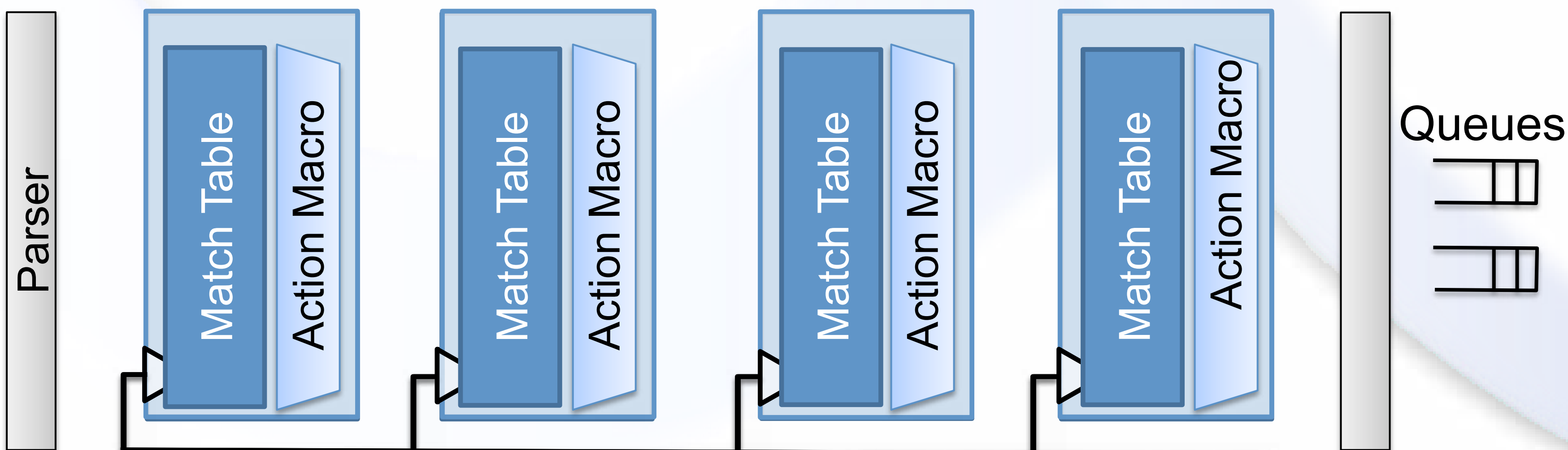
BAREFOOT NETWORKS

Protocol-Independent Switch Architecture (PISA)



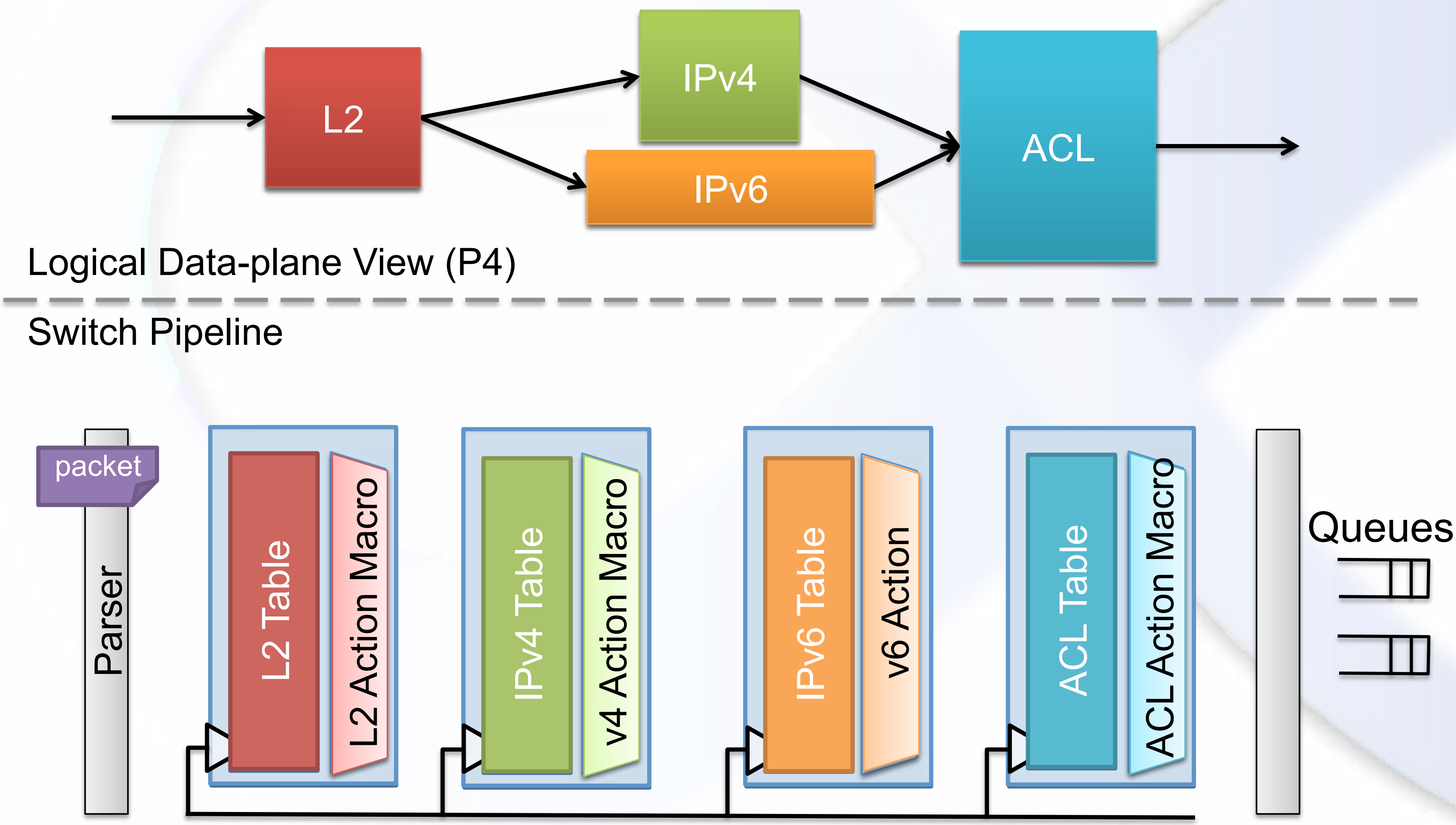
Logical Data-plane View (your P4 program)

Switch Pipeline



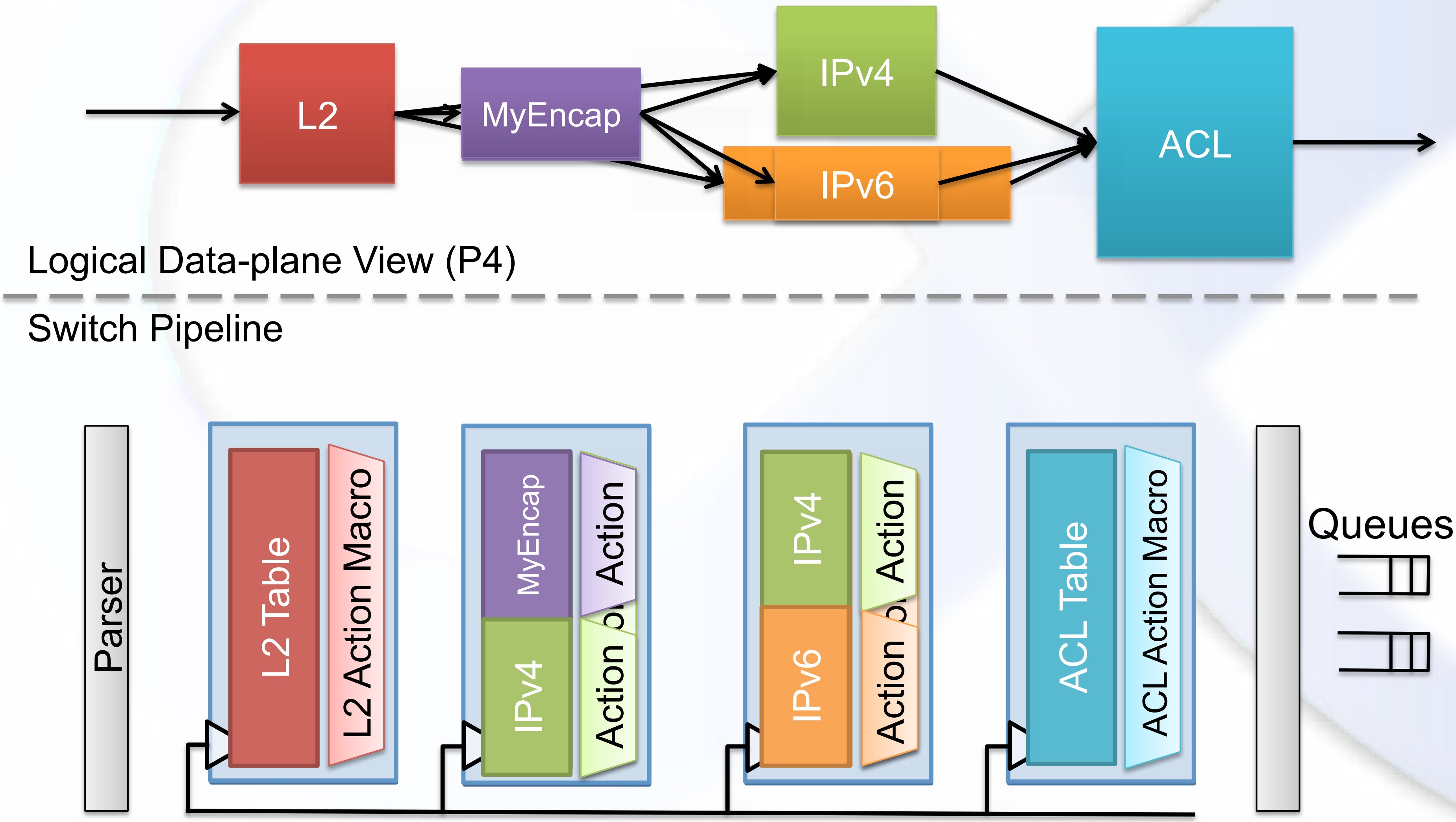
BAREFOOT NETWORKS

Mapping to Physical Resources



BAREFOOT NETWORKS

Re-configurability



P4: Three Goals

Protocol independence

- Configure a packet parser
- Define a set of typed match+action tables

Target independence

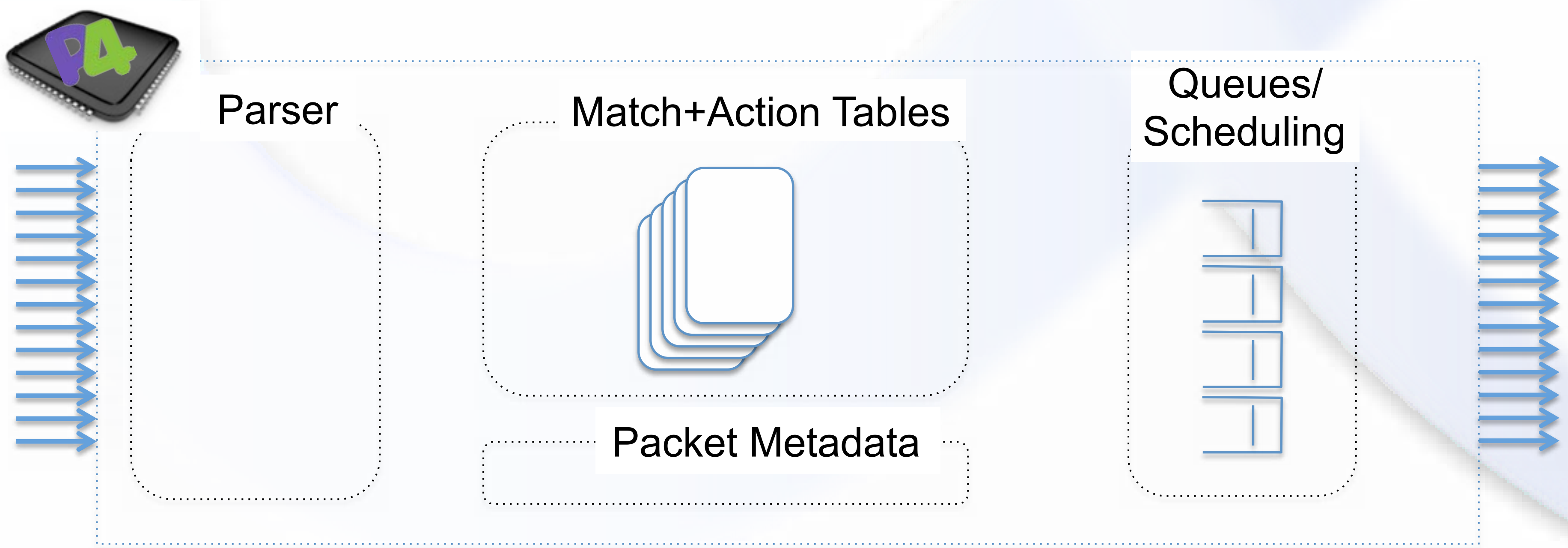
- Program without knowledge of switch details
- Rely on compiler to configure the target switch

Reconfigurability

- Change parsing and processing in the field

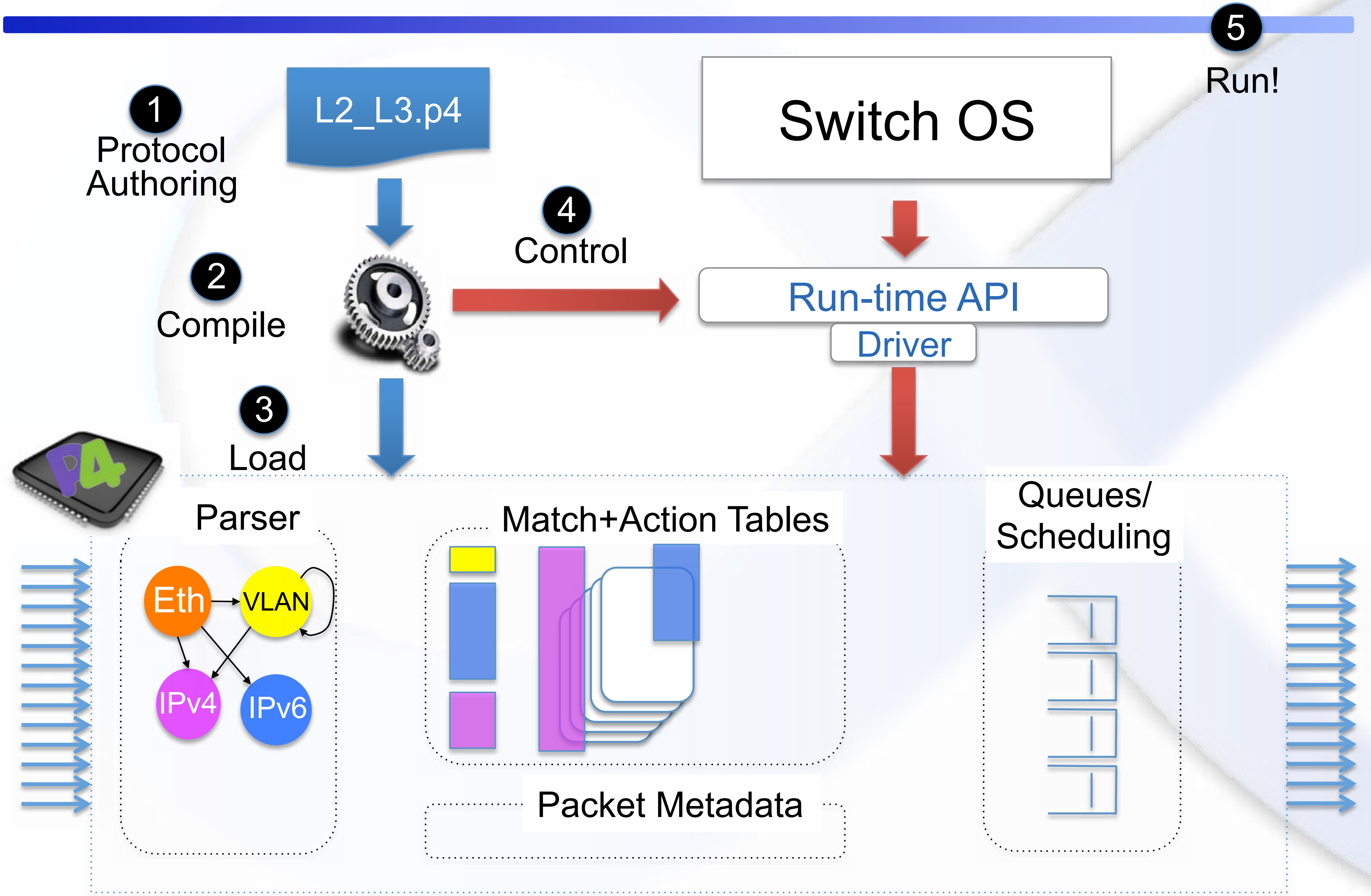
P4-Based Workflow

- **Device is not yet programmed**
  - Does not know about any packet formats or protocols



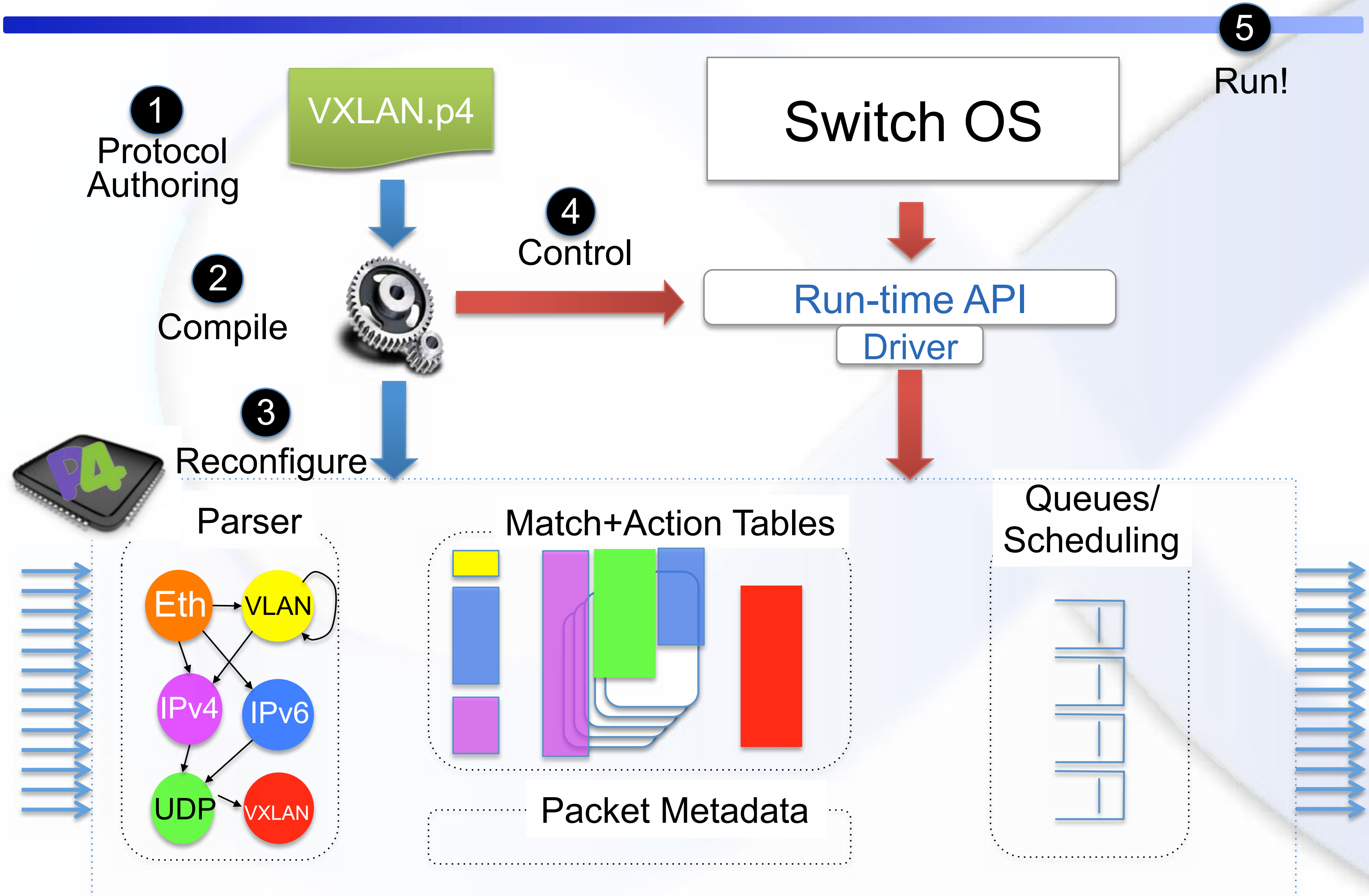
BAREFOOT NETWORKS

P4-Based Workflow



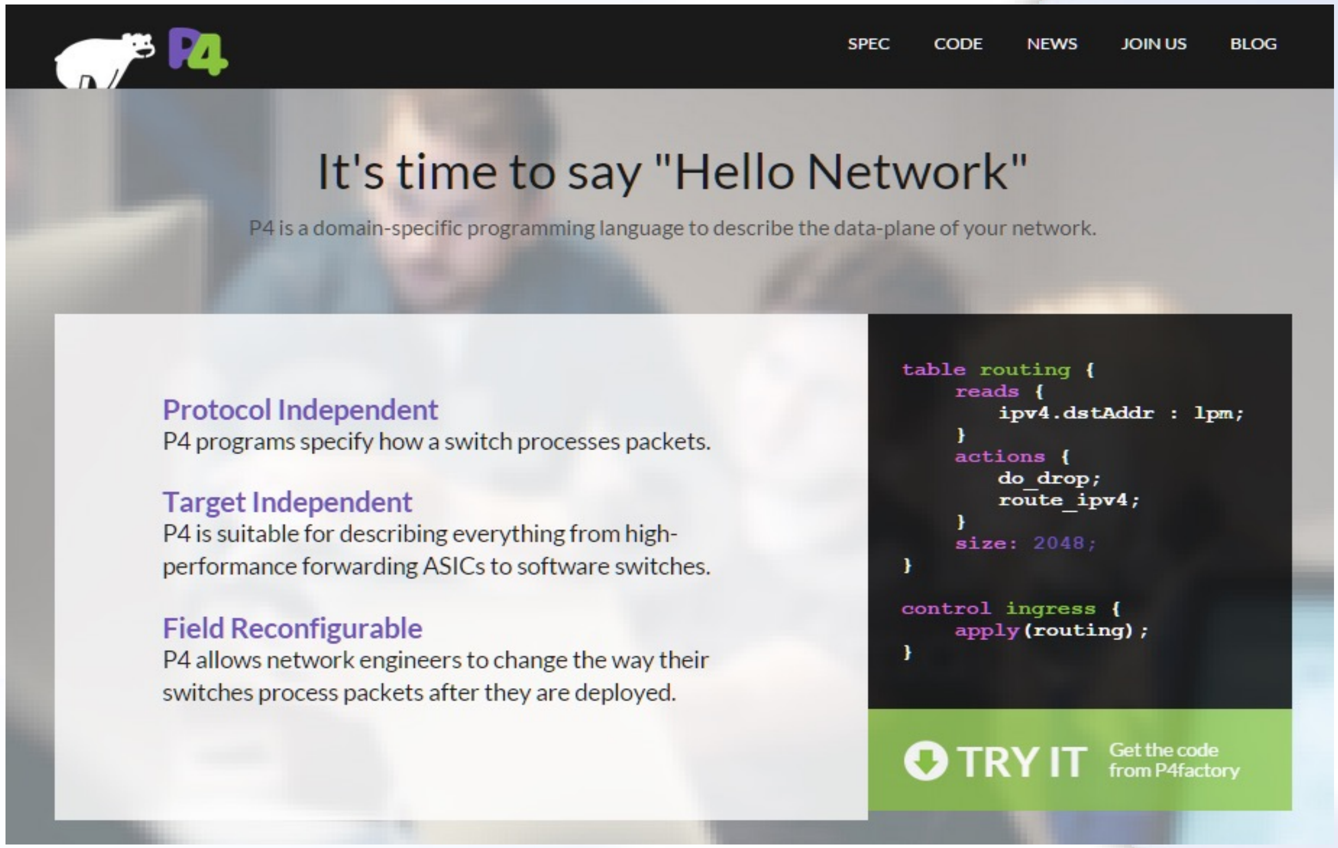
BAREFOOT NETWORKS

P4-Based Workflow



The P4 Language Consortium

- Consortium of academic and industry members
- Open source, evolving, domain-specific language
- Permissive Apache license, code on GitHub today
- Membership is free: contributions are welcome
- Independent, set up as a California nonprofit



BAREFOOT NETWORKS

P4.org Membership

Operators
Alibaba Group 阿里巴巴集团 · at&t · Bai du 百度 · 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 · POLITECNICO MILANO 1863 · PRINCETON UNIVERSITY · Stanford University · uni.lu UNIVERSITÉ DU LUXEMBOURG · Università della Svizzera italiana

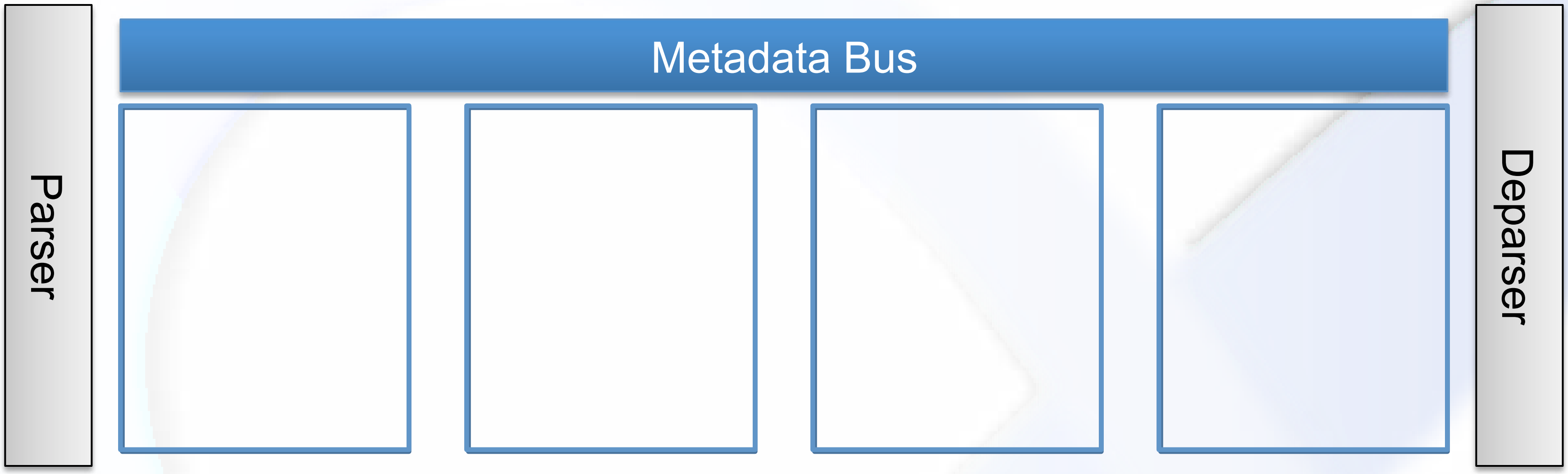
BAREFOOT NETWORKS

P4 Concepts

- **Pipeline**
  - Parser / Deparser
  - Match-Action Tables

BAREFOOT NETWORKS

The anatomy of a basic pipeline



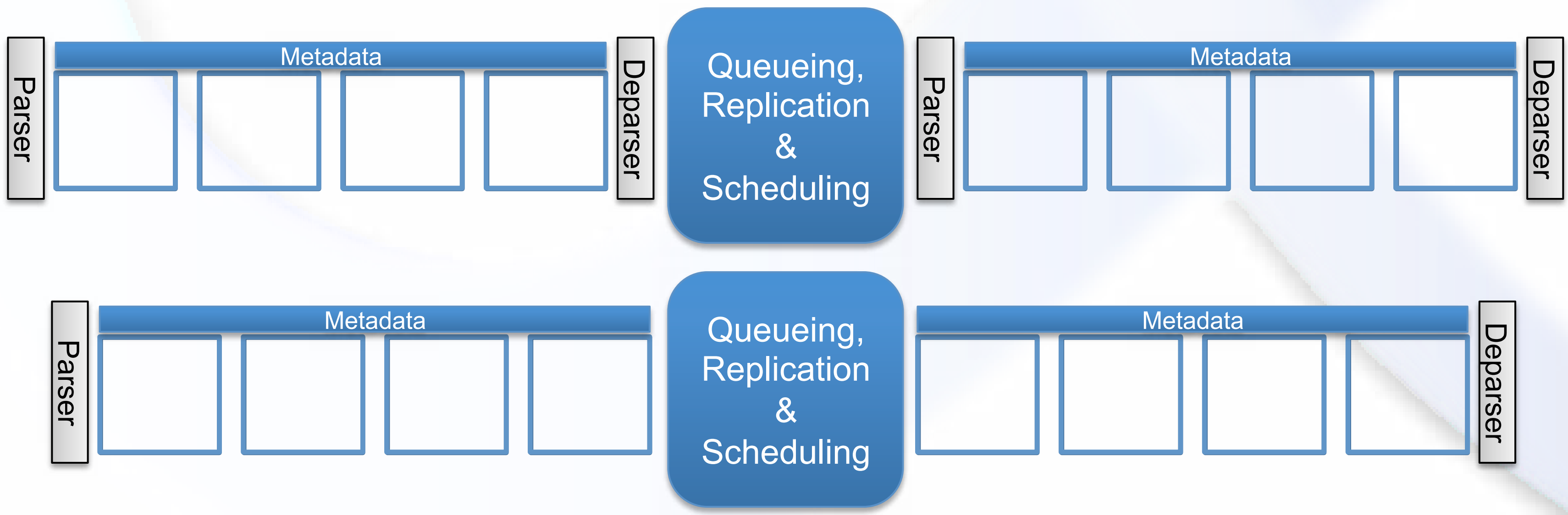
- **Parser**
  - Converts packet data into a metadata (Parsed Representation)
- **Match+Action Tables**
  - Operate on metadata
- **Deparser**
  - Converts metadata back into a serialized packet
- **Metadata Bus**
  - Carries the information within the pipeline

All are optional

BAREFOOT NETWORKS

Anatomy of a Switch

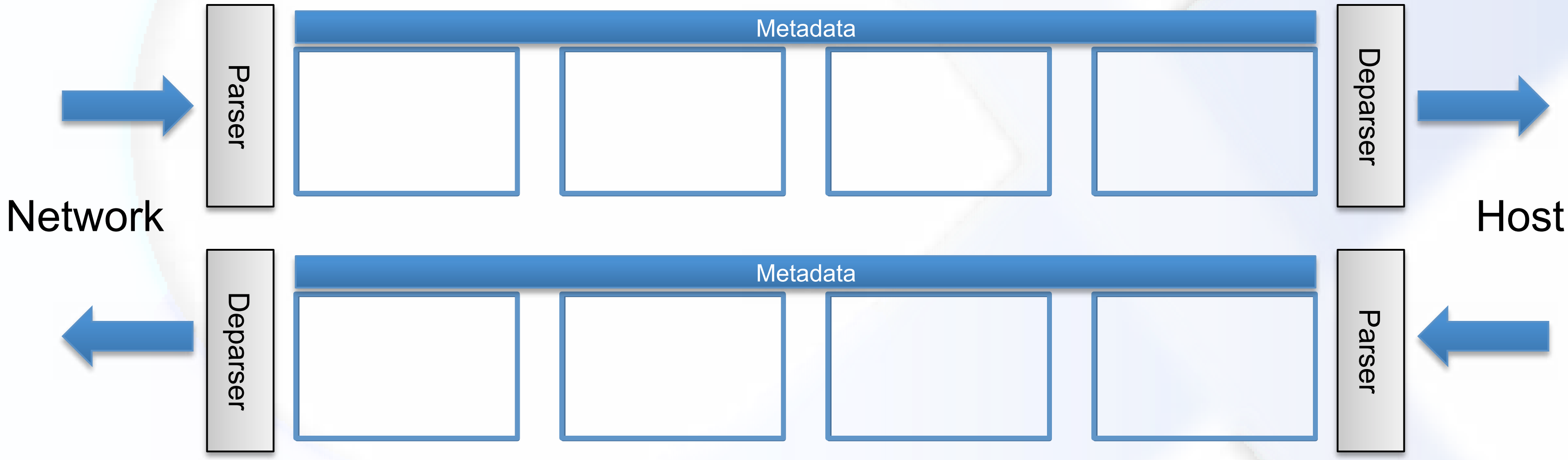
- **Ingress Pipeline**
- **Egress Pipeline**
- **Traffic Manager**
  - N:1 Relationships: Queueing, Congestion Control
  - 1:N Relationships: Replication
  - Scheduling



BAREFOOT NETWORKS

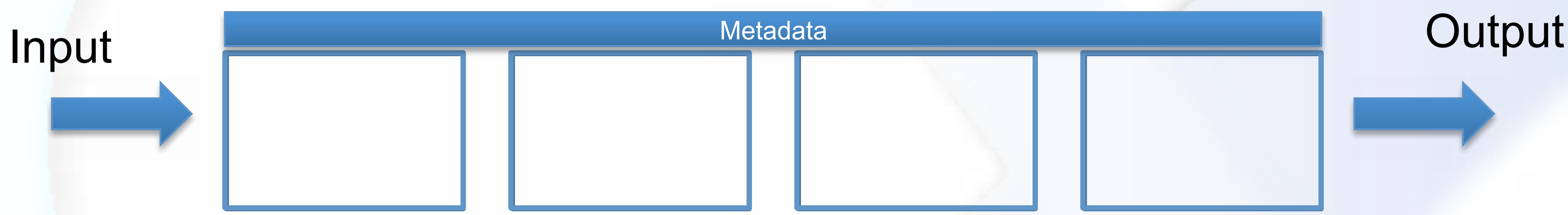
Anatomy of a NIC

- **Single or Dual Pipeline**



Anatomy of Protocol Plugin

- **Single, “Bare” Pipeline**
  - No parsing/deparsing, just processing



P4 Program Sections

program.p4

Data Declarations

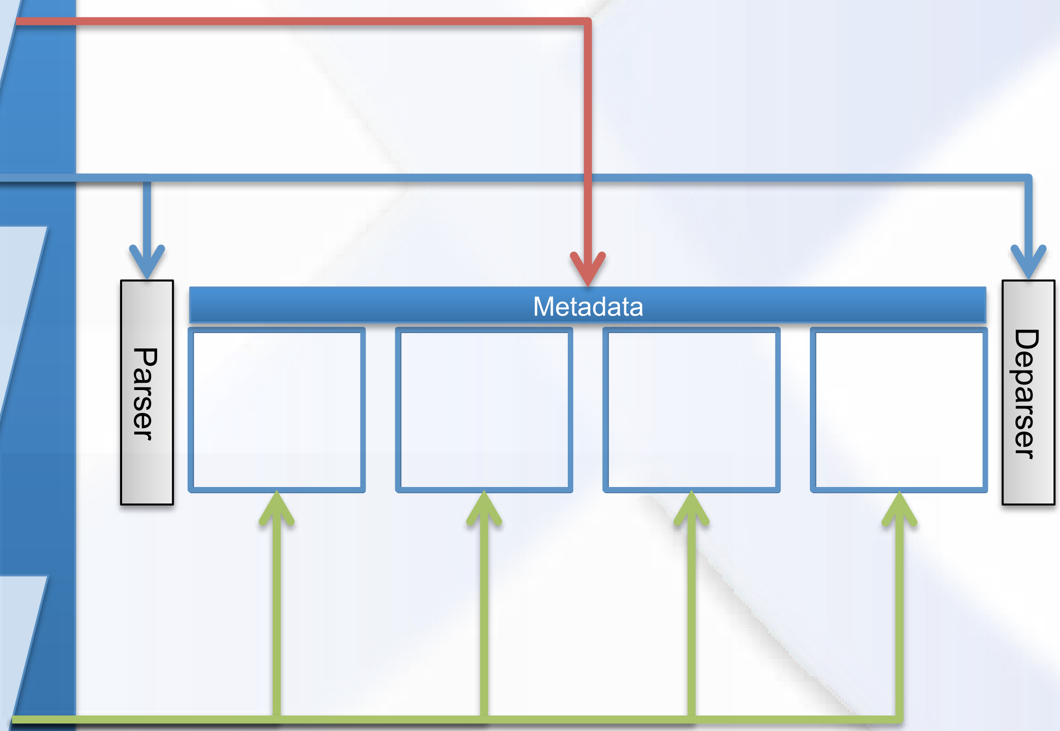
```
header_type  ethernet_t    { … }
header_type  l2_metadata_t { … }

header     ethernet_t     ethernet;
header     vlan_tag_t     vlan_tag[2];
metadata   l2_metadata_t  l2_meta;
```

Parser Program

```
parser parse_ethernet {
   extract(ethernet);
   return switch(ethernet.ethertype) {
      0x8100 : parse_vlan_tag;
      0x0800 : parse_ipv4;
      0x8847 : parse_mpls;
      default: ingress;
}
```

Control Flow Program

```
table port_table { … }

control ingress {
    apply(port_table);
    if (l2_meta.vlan_tags == 0) {
        process_assign_vlan();
    }
}
```



BAREFOOT NETWORKS

P4 Constructs

- **P4 Spec 1.0.2+**

P4 Language Components

- **Data declarations**
  - Packet Headers and Metadata
- **Parser Programming**
  - Parser Functions (Parser states)
  - Checksum Units
- **Packet Flow Programming**
  - Actions
    - Primitive and compound actions
    - Counters, Meters, Registers
  - Tables
    - Match keys
    - Attributes
  - Control Functions (Imperative Programs)

No: pointers, loops, recursion, floating point

BAREFOOT NETWORKS

Headers and Fields (Packet)

<u>Example:</u> Declaring packet headers

```
header_type ethernet_t {
    fields {
        dstAddr   : 48;
        srcAddr   : 48;
        etherType : 16;
    }
}

header_type vlan_tag_t {
    fields {
        pcp       : 3;
        cfi       : 1;
        vid       : 12;
        etherType : 16;
    }
}

header ethernet_t ethernet;
header vlan_tag_t vlan_tag[3];
```

Header Type Declarations

Actual Header Instantiation

Handy Arrays for Header Stacks

Headers and Fields (Metadata)

<u>Example:</u> Declaring Metadata

```
header_type ingress_metadata_t {
    fields {
     /* Inputs */
        ingress_port          : 9;   /* Available prior to parsing    */
        packet_length         : 16;  /* Might not be always available */
        instance_type         : 2;   /* Normal, clone, recirculated   */
        ingress_global_tstamp : 48;
        parser_status         : 8;   /* Parsing Error */

     /* Outputs from Ingress Pipeline */
        egress_spec           : 16;
        queue_id              : 9;
   }
}

metadata ingress_metadata_t ingress_metadata;
```

Metadata is a header too

Actual Metadata Instantiations

Metadata vs. Packet Headers

- **Layout definition**
  - Packet header declarations define both the fields and the actual layout in the packet.
  - Layout is not defined for metadata
- **Byte Alignment**
  - Packet header length must be a multiple of 8 bits
  - No special requirements for metadata
- **Validity**
  - Packet headers are valid only if present in the packet
  - Metadata is ALWAYS valid
    - Default value is either 0 or can be specified explicitly
- **Acceptable fields**
  - Packet headers can contain calculated and variable length fields

Variable-Length Fields

Example: Declaring IPv4 packet header

```
header_type ipv4_t {
    fields {
        version        : 4;
        ihl            : 4;
        diffserv       : 8;
        totalLen       : 16;
        identification : 16;
        flags          : 3;
        fragOffset     : 13;
        ttl            : 8;
        protocol       : 8;
        hdrChecksum    : 16;
        srcAddr        : 32;
        dstAddr        : 32;
        options        : *;
    }
    length     : (ihl << 2);
    max_length : 60;
}
```

Variable-length Field

Calculated, based on another field

Defining a Parser Tree

<u>Example:</u> Simple Parser for L2/L3 Packets

```
header ethernet_t ethernet;
header vlan_tag_t vlan_tag[2];
header ipv4_t ipv4;
header ipv6_t ipv6;

parser start {
    extract(ethernet);
    return select(latest.etherType) {
        0x8100, 0x9100 : parse_vlan_tag;
        0x0800         : parse_ipv4;
        0x86DD         : parse_ipv6;
        default        : ingress;
    }
}
parser parse_vlan_tag {
    extract(vlan_tag[next]);
    return select(latest.etherType) {
        0x8100 mask 0xEFFF : parse_vlan_tag;
        0x0800             : parse_ipv4;
        0x86DD             : parse_ipv6;
        default            : ingress;
    }
}
```

Transitions to the next parser states

Depending on the state, it can be:
- ethernet.ethertype
- vlan_tag[0].ethertype
- vlan_tag[1].ethertype

The loop is bounded by the number of elements in vlan_tag[] array

This is not a reserved word, but a name of the Control Flow Function

Defining a Parser Tree

Example: **Simple Parser for L2/L3 Packets**

```
header ethernet_t ethernet;
header vlan_tag_t vlan_tag[2];
header ipv4_t ipv4;
header ipv6_t ipv6;

parser start {
    extract(ethernet);
    return select(latest.etherType) {
        0x8100, 0x9100 : parse_vlan_tag;
        0x0800         : parse_ipv4;
        0x86DD         : parse_ipv6;
        default        : ingress;
    }
}
parser parse_vlan_tag {
    extract(vlan_tag[next]);
    return select(latest.etherType) {
        0x8100 mask 0xEFFF : parse_vlan_tag;
        0x0800             : parse_ipv4;
        0x86DD             : parse_ipv6;
        default            : ingress;
    }
}
```

```
parser parse_ipv4 {
    extract(ipv4);
    return ingress;
}

parser parse_ipv6 {
    extract(ipv6);
    return ingress;
}
```

BAREFOOT NETWORKS

Using Calculated Fields

Example: **Calculated fields for IPv4**

```
field_list ipv4_checksum_list {
        ipv4.version;
        ipv4.ihl;
        ipv4.diffserv;
        ipv4.totalLen;
        ipv4.identification;
        ipv4.flags;
        ipv4.fragOffset;
        ipv4.ttl;
        ipv4.protocol;
        ipv4.srcAddr;
        ipv4.dstAddr;
}
field_list_calculation ipv4_checksum {
    input        { ipv4_checksum_list; }
    algorithm    : csum16;
    output_width : 16;
}

calculated_field ipv4.hdrChecksum  {
    verify ipv4_checksum;
    update ipv4_checksum;
}
```

```
parser parse_ipv4 {
    extract(ipv4);
    return ingress;
}
```

BAREFOOT NETWORKS

Multi-field select statement

<u>Example:</u> Ipv4 Header Parsing

```
parser parse_ipv4 {
    extract(ipv4);
    set_metadata(ipv4_metadata.lkp_ipv4_sa, ipv4.srcAddr);
    set_metadata(ipv4_metadata.lkp_ipv4_da, ipv4.dstAddr);
    set_metadata(l3_metadata.lkp_ip_proto, ipv4.protocol);
    set_metadata(l3_metadata.lkp_ip_ttl, ipv4.ttl);

    return select(latest.fragOffset, latest.ihl, latest.protocol) {
        0x0000_5_01 : parse_icmp;
        0x0000_5_06 : parse_tcp;
        0x0000_5_11 : parse_udp;
        default     : ingress;
}
```

Metadata can be set from the parser

Fields are joined for a match

"_" are ignored in numerical constants

Deparsing (Serializing packet headers)

- **Fundamental assumption of P4**
  - The device must be able to parse any packet it can produce
- **Consequence**
  - Packet headers can be reassembled using the parser definition

BAREFOOT NETWORKS

Actions

- **Primitive actions**
  - no_op, drop
  - modify_field, modify_field_with_hash_based_index
  - add_header, remove_header, copy_header
  - push/pop (a header)
  - count, execute_meter
  - generate_digest, truncate
  - resubmit, recirculate, clone{_i2i, _e2i, _i2e, _e2e}
- **Compound actions**

```
action route_ipv4(dst_port, dst_mac, src_mac, vid) {
    modify_field(standard_metadata.egress_spec, dst_port);
    modify_field(ethernet.dst_addr, dst_mac);
    modify_field(ethernet.src_addr, src_mac);
    modify_field(vlan_tag.vid, vid);
    modify_field(ipv4.ttl, ipv4.ttl - 1);
}
```

BAREFOOT NETWORKS

Action Execution Semantics

- **All actions within a compound action are assumed to be executed sequentially**

```
action parallel_test() {
    modify_field(hdr.fieldA, 1);
    modify_field(hdr.fieldB, hdr.fieldA);
}
```

	Sequential Semantics	Parallel Semantics
fieldA	1	1
fieldB	1	fieldA before action

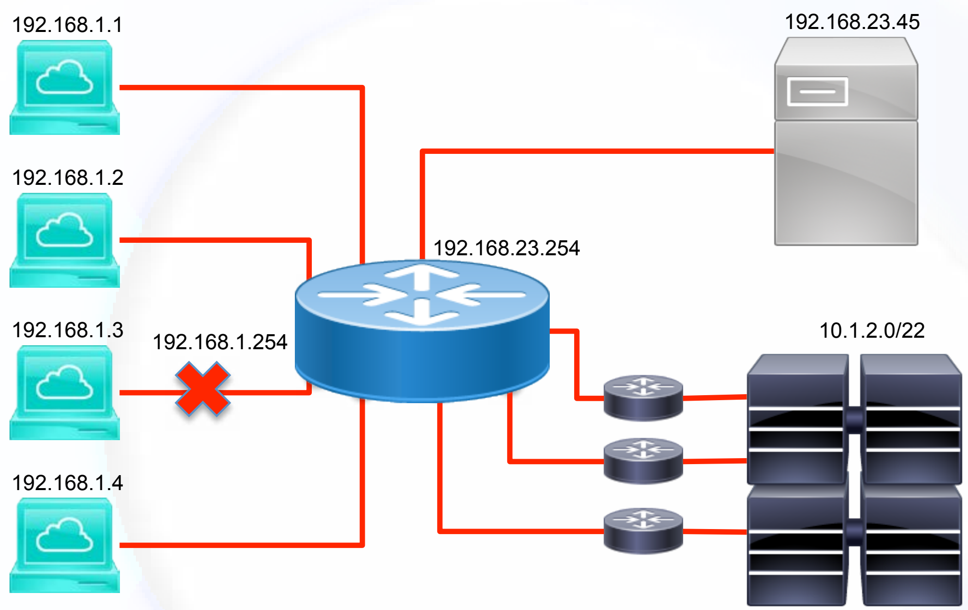
- **This is an important specification change**
  - Up to version 1.0.2 action execution was parallel
  - After 1.0.2 action execution is sequential
- **The maximum number of steps supported for a compound action is target-dependent**

Match-Action Tables

- **The most fundamental units of the Match-Action Pipeline**
  - What to match on and match type
  - A list of possible actions
  - Additional attributes
    - Size
    - What to do on miss
- **Each table contains one or more entries (rows)**
- **An entry contains:**
  - A specific key to match on
  - A **single** action
    - to be executed when a packet matches the entry
  - (Optional) action data

BAREFOOT NETWORKS

Example: IPv4 Processing



Key	Action	Action Data
192.168.1.1	l3_switch	port= mac_da= mac_sa= vlan=
192.168.1.2	l3_switch	port= mac_da= mac_sa= vlan=…
192.168.1.3	l3_drop	
192.168.1.254	l3_l2_switch	port=
192.168.1.0/24	l3_l2_switch	port=
10.1.2.0/22	l3_switch_ecmp	ecmp_group=

BAREFOOT NETWORKS

Defining Actions

```
action l3_switch(port, mac_da, mac_sa, vlan) {
    modify_field(metadata.egress_spec, port);
    modify_field(ethernet.dstAddr, mac_da);
    modify_field(ethernet.srcAddr, mac_sa);
    modify_field(vlan_tag[0].vlanid, vlan);
    modify_field(ipv4.ttl, ipv4.ttl - 1);
}

action l3_l2_switch(port) {
    modify_field(metadata.egress_spec, port);
}

action l3_drop() {
    drop();
}

action l3_switch_nexthop(nexthop_index) {
    modify_field(l3_metadata.nexthop, nexthop_index);
    modify_field(l3_metadata.nexthop_type, NEXTHOP_TYPE_SIMPLE);
}
action l3_switch_ecmp(ecmp_group) {
    modify_field(l3_metadata.nexthop, ecmp_group);
    modify_field(l3_metadata.nexthop_type, NEXTHOP_TYPE_ECMP);
}
```

Match-Action Table (Exact Match)

<u>Example:</u> A typical L3 (IPv4) Host table

```
table ipv4_host {
    reads {
        ingress_metadata.vrf     : exact;
        ipv4.dstAddr             : exact;
    }
    actions {
        nop;
        l3_switch;
        l3_l2_switch;
        l3_switch_nexthop;
        l3_switch_ecmp;
        l3_drop;
    }
    size : HOST_TABLE_SIZE;
}
```

These are the only possible actions. Each particular entry can have only ONE of them.

vrf	ipv4.dstAddr	action	data
1	192.168.1.10	l3_switch	port_id= mac_da= mac_sa=
100	192.168.1.10	l3_l2_switch	port_id=<CPU>
1	192.168.1.3	l3_drop	
5	10.10.1.1	l3_switch_ecmp	ecmp_group=127

BAREFOOT NETWORKS

Match-Action Table (LPM)

<u>Example:</u> A typical L3 (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;
}
```

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_switch_nexthop	nexthop_index=451
1	0.0.0.0 / 0	l3_switch_nexthop	nexthop_index=1

BAREFOOT NETWORKS

Match-Action Table (TCAM-based)

<u>Example:</u> A typical L3 (IPv4) Routing table

```
table ipv4_lpm {
    reads {
        ingress_metadata.vrf     : ternary;
        ipv4.dstAddr             : ternary;
    }
    actions {
        nop;
        l3_l2_switch;
        l3_multicast;
        l3_nexthop;
        l3_ecmp;
        l3_drop;
   }
   size : 65536;
}
```

Prio	vrf	ipv4.dstAddr / mask	action	data
100	0x001/0xFFF	192.168.1.5 / 255.255.255.255	l3_swith_nexthop	nexthop_index=10
10	0x000/0x000	192.168.2.0/255.255.255.0	l3_switch_ecmp	ecmp_index=25
10	0x000/0x000	192.168.3.0/255.255.255.0	l3_switch_nexthop	nexthop_index=31
5	0x000/0x000	0.0.0.0/0.0.0.0.	l3_l2_switch	port_id=64

Match-Action Table (dmac actions)

<u>Example:</u> A typical L2 table

```
/* Possible Actions */
action unicast_send(port_id) {
    modify_field(ingress_metadata.egress_ifindex, port_id);
}

action multicast_send(mc_index) {
    modify_field(ig_intr_md_for_tm.mcast_grp_b, mc_index)
}

action dmac_redirect_nexthop(nexthop_index) {
    modify_field(l2_metadata.l2_redirect, TRUE);
    modify_field(l2_metadata.l2_nexthop, nexthop_index);
    modify_field(l2_metadata.l2_nexthop_type, NEXTHOP_TYPE_SIMPLE);}

action dmac_redirect_ecmp(ecmp_index) {
    modify_field(l2_metadata.l2_redirect, TRUE);
    modify_field(l2_metadata.l2_nexthop, ecmp_index);
    modify_field(l2_metadata.l2_nexthop_type, NEXTHOP_TYPE_ECMP);
}

action dmac_drop() {
    drop();
}
```

Match-Action Table (Exact Match)

Example: **A typical L2 table**

```
/* Actual Table */
table dmac {
    reads {
        ingress_metadata.bd    : exact;
        l2_metadata.lkp_mac_da : exact;
    }
    actions {
        nop;
        dmac_hit;
        dmac_multicast_hit;
        dmac_redirect_nexthop;
        dmac_redirect_ecmp;
        dmac_drop;
    }
    size : MAC_TABLE_SIZE;
}
```

These are the only possible actions. Each particular entry can have only ONE of them.

bd (vlan)	lkp_mac_da	action	data
1	00:00:01:02:03:04	dmac_hit	port_id
100	01:22:33:44:55:66	dmac_multicast_hit	mc_index=465
10	00:11:11:11:11:11	dmac_hit	ifindex=31
5	00:12:13:00:00:01	dmac_redirect_nexthop	nexthop_index=17

Match-Action Table (TCAM-based)

Example: TCAM-based L2 Lookup

```
/* Actual Table */
table dmac_cache {
    reads {
        ingress_metadata.bd    : ternary;
        l2_metadata.lkp_mac_da : ternary;
    }
    actions {
        nop;
        dmac_hit;
        dmac_multicast_hit;
        dmac_redirect_nexthop;
        dmac_redirect_ecmp;
        dmac_drop;
    }
    size : 65536;
}
```

Prio	bd /bd_mask	lkp_mac_da/mask	action	data
100	0x001/0xFFF	00:00:01:02:03:04/FF:FF:FF:FF:FF:FF	dmac_hit	ifindex=10
10	0x000/0x000	01:22:33:44:55:66/FF:FF:FF:FF:FF:FF	dmac_multicast_hit	mc_index=465
10	0x000/0x000	00:11:11:00:00:00/FF:FF:FF:FF:FF:FF	dmac_hit	ifindex=31
5	0x000/0x000	00:12:13:00:00:00/FF:FF:FF:00:00:00	dmac_redirect_nexthop	nexthop_index=17

Types of Match

- **Exact**
  - port_index : exact
- **Ternary**
  - ethernet.srcAddr : ternary
- **Valid**
  - vlan_tag[0] : valid
- **LPM (special kind of ternary match)**
  - ipv4.dstAddr : lpm
- **Range**
  - udp.dstPort : range

Table Miss

- **Each table can have a Default Action**
  - Chosen by the Control Path at runtime from the list of table Actions
    - P4 Program does not have an indication which action (and which action data) will be the default
- **Default Action (with the default action data) is executed in the event when no matching entries are found**
- **If no Default Action is specified, it is no_op()**

BAREFOOT NETWORKS

Tables without a match

- **If a table has no reads{} section, it always produces a miss**
- **Control plane can enable execution of the action by setting it as a default for the table**

```
action increment_counters() {
    count(bd_counter, metadata.bd_counter_index);
    count(vrf_counter, metadata.vrf_counter_index);
}

table do_counting {
    actions {
        increment_counters;
    }
}
```

```
P4_CLI>>> table_set_default do_counting increment_counters
```

Direct Counters

- **A counter per table entry**

```
counter ip_acl_stats {
    type : packets_and_bytes;
    direct : ip_acl;
}

table ip_acl {
    reads {
        ipv4_metadata.lkp_ipv4_sa : ternary;
        ipv4_metadata.lkp_ipv4_da : ternary;
        l3_metadata.lkp_ip_proto  : ternary;
        l3_metadata.lkp_l4_sport  : ternary;
        l3_metadata.lkp_l4_dport  : ternary;
    }
    actions {
        nop;
        acl_log;
        acl_deny;
        acl_permit;
        acl_mirror;
        acl_redirect_nexthop;
        acl_redirect_ecmp;
    }
    size : INGRESS_IP_ACL_TABLE_SIZE;
}
```

table ip_acl | **counter** ip_acl_stats

Match Fields	*Action Sel*	*Action Data*	*Counter*
ABCD_xxxx_0123	acl_deny		counter A
matched entry	**acl_permit**	**8b 8b**	**pkt/byte counts**
BA8E_F007_xxxx	nop		counter Z

Indirect Counters

- **Flexibly linked counters**

```
counter ingress_bd_stats {
    type : packets_and_bytes;
    instance_count : BD_STATS_TABLE_SIZE;
}

action set_bd(bd, bd_stat_index) {
    modify_field(l2_metadata.bd, bd);
    count(ingress_bd_stats, bd_stat_index);
}

table port_vlan {
    reads {
        ingress_metadata.ingress_port : exact;
        vlan_tag[0]                   : valid;
        vlan_tag[0].vlan_id           : exact;
    }
    actions {
        set_bd;
    }
}
```

Different VLANs (BDs) can share the same counter

Other tables can also reference these counters

table port_vlan

Match Fields	Action Sel	Action Data	
ABCD_0123	set_bd	bd	bd_stat_index
	set_bd	bd	bd_stat_index
matched entry	**set_bd**	**bd**	**bd_stat_index A**
	set_bd	bd	bd_stat_index
	set_bd	bd	bd_stat_index
	set_bd	bd	bd_stat_index A
BA8E_F007	set_bd	bd	bd_stat_index

counter ingress_bd_stats

pkt/byte counts

Meters

- **Declaration is similar to counters**
  - Action: execute_meter()

```
meter acl_meter {
    type:    packets;
     direct: ip_acl;
     result: metadata.color;
}

meter bd_meter {
    type: bytes;
    instance_count: 1000;
}

action do_acl_meter(meter_index) {
    execute_meter(acl_meter, meter_index, metadata.color);
}
```

Meters calculate packet color and deposit it into the specified field

Color Coding:

0 – Green
1 – Yellow
2 -- Red

Registers

- **Declaration is similar to indirect counters**
  - Actions: read_register(), write_register()

```
register last_syn {
    width: 32;
    statidc: flow_table;
    instance_count:  1024;
}

action get_flow_age(flow_index) {
    read_register(last_syn, flow_index, metadata.flow_start_time);
    modify_field(metadata.flow_age,
                 metadata.flow_start_time - metadata.ingress_global_stamp);
}

action start_new_flow(flow_index) {
    write_register(last_syn, flow_index, metadata.ingress_global_timestamp);
}
```

BAREFOOT NETWORKS

Action Profiles

- **Actions can be complex**

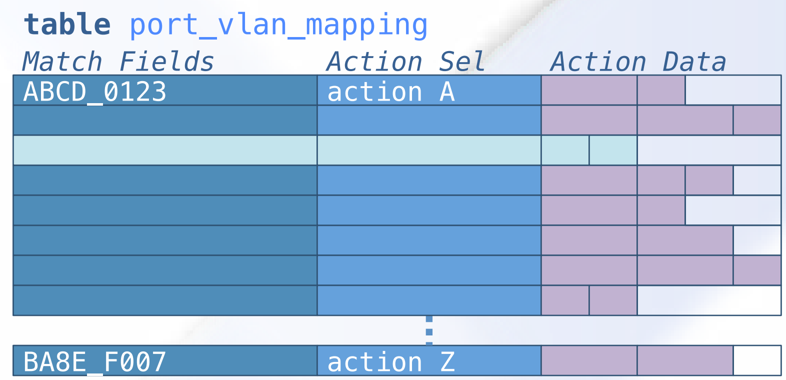
```
action set_bd(bd, vrf, rmac_group,
        ipv4_unicast_enabled, ipv6_unicast_enabled,
        ipv4_urpf_mode, ipv6_urpf_mode,
        igmp_snooping_enabled, mld_snooping_enabled,
        bd_label, stp_group, stats_idx,
        exclusion_id)
{
    modify_field(l3_metadata.vrf, vrf);
    modify_field(ipv4_metadata.ipv4_unicast_enabled, ipv4_unicast_enabled);
    modify_field(ipv6_metadata.ipv6_unicast_enabled, ipv6_unicast_enabled);
    modify_field(ipv4_metadata.ipv4_urpf_mode, ipv4_urpf_mode);
    modify_field(ipv6_metadata.ipv6_urpf_mode, ipv6_urpf_mode);
    modify_field(l3_metadata.rmac_group, rmac_group);
    modify_field(acl_metadata.bd_label, bd_label);
    modify_field(ingress_metadata.bd, bd);
    modify_field(ingress_metadata.outer_bd, bd);
    modify_field(l2_metadata.stp_group, stp_group);
    modify_field(l2_metadata.bd_stats_idx, stats_idx);

    modify_field(multicast_metadata.igmp_snooping_enabled,
                 igmp_snooping_enabled);
    modify_field(multicast_metadata.mld_snooping_enabled,
                 mld_snooping_enabled);
    modify_field(ig_intr_md_for_tm.level1_exclusion_id, exclusion_id);
}
```

60-70 bits for the parameters

Naïve implementation

- **Each entry has its own action**

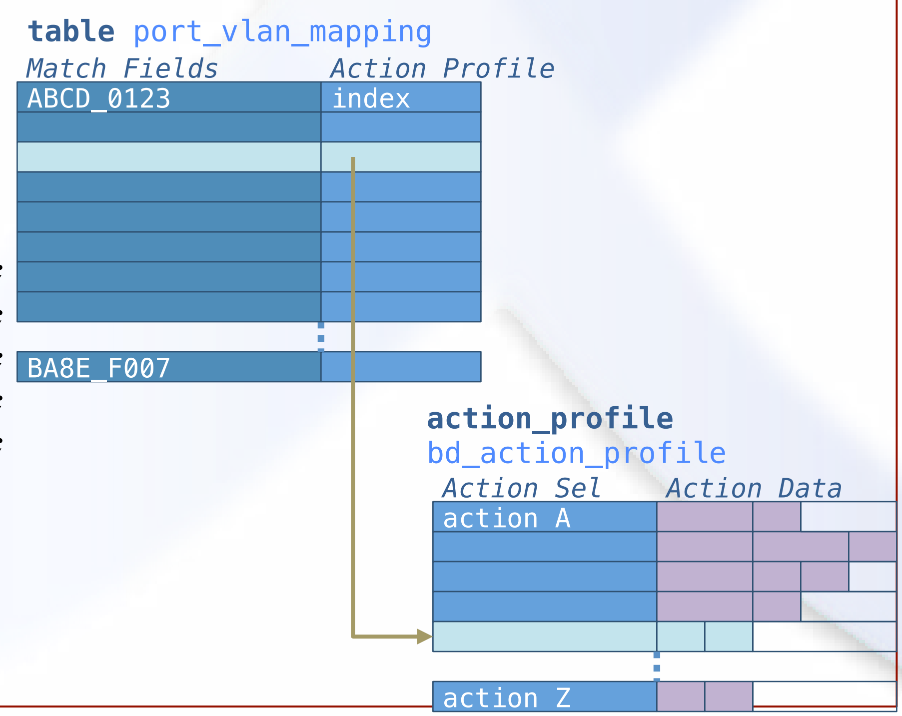
```
table port_vlan_mapping {
    reads {
        ingress_metadata.ifindex : exact;
        vlan_tag_[0]             : valid;
        vlan_tag_[0].vid         : exact;
        vlan_tag_[1]             : valid;
        vlan_tag_[1].vid         : exact;
    }
    actions {
        set_bd;
        set_bd_ipv4_mcast_switch_ipv6_mcast_switch_flags;
        set_bd_ipv4_mcast_switch_ipv6_mcast_route_flags;
        set_bd_ipv4_mcast_route_ipv6_mcast_switch_flags;
        set_bd_ipv4_mcast_route_ipv6_mcast_route_flags;
    }
    size : 32768;
}
```



Using the profiles

- **Sharing the same action with multiple entries**

```
action_profile bd_action_profile {
    actions {
        set_bd;
        set_bd_ipv4_mcast_switch_ipv6_mcast_switch_flags;
        set_bd_ipv4_mcast_switch_ipv6_mcast_route_flags;
        set_bd_ipv4_mcast_route_ipv6_mcast_switch_flags;
        set_bd_ipv4_mcast_route_ipv6_mcast_route_flags;
    }
    size : 8192;
}

table port_vlan_mapping {
    reads {
        ingress_metadata.ifindex : exact;
        vlan_tag_[0]             : valid;
        vlan_tag_[0].vid         : exact;
        vlan_tag_[1]             : valid;
        vlan_tag_[1].vid         : exact;
    }
    action_profile : bd_action_profile;
    size : 32768;
}
```



Using the profiles for LAG and ECMP

```
action_selector ecmp_selector {
    selection_key : ecmp_hash;
}

action_profile ecmp_action_profile {
    actions {
        nop;
        set_ecmp_nexthop_details;
    }
    size : ECMP_SELECT_TABLE_SIZE;
    dynamic_action_selection : ecmp_selector;
}

table ecmp_group {
    reads {
        l3_metadata.nexthop_index : exact;
    }
    action_profile: ecmp_action_profile;
    size : ECMP_GROUP_TABLE_SIZE;
}
```

Chooses a particular entry within a group

Chooses a GROUP of profile entries

Using the profiles for LAG and ECMP

```
action_selector ecmp_selector {
    selection_key : ecmp_hash;
}

action_profile ecmp_action_profile {
    actions {
        nop;
        set_ecmp_nexthop_details;
    }
    size : ECMP_SELECT_TABLE_SIZE;
    dynamic_action_selection :
          ecmp_selector;
}

table ecmp_group {
    reads {
        l3_metadata.nexthop_index : exact;
    }
    action_profile: ecmp_action_profile;
    size : ECMP_GROUP_TABLE_SIZE;
}
```

```
field_list l3_hash_fields {
    ipv4_metadata.lkp_ipv4_sa;
    ipv4_metadata.lkp_ipv4_da;
    l3_metadata.lkp_ip_proto;
    l3_metadata.lkp_l4_sport;
    l3_metadata.lkp_l4_dport;
}

field_list_calculation ecmp_hash {
    input {
        l3_hash_fields;
    }
    algorithm : crc16;
    output_width : ECMP_BIT_WIDTH;
}
```

BAREFOOT NETWORKS

Control Flow Functions

- **Primitives**
  - Perform a table lookup: **apply**
  - **if/else** statement
  - **apply** with the case clause
- **Sequential Execution Semantics**
- **User-defined control functions**

```
control perform_basic_l2 {
    apply(port_vlan_maping);
    apply(dmac);
}
```

- **Standard control functions: ingress() and egress()**

BAREFOOT NETWORKS

If/Else Branching

Example: Separate Ipv4 and IPv6 Processing Paths

```
if ((l3_metadata.lkp_ip_type == IPTYPE_IPV4) and
    (ipv4_metadata.ipv4_unicast_enabled == TRUE)) {
    /* router ACL/PBR */
    process_ipv4_racl();
    process_nat();
    process_ipv4_urpf();
    process_ipv4_fib();
} else {
    if ((l3_metadata.lkp_ip_type == IPTYPE_IPV6) and
        (ipv6_metadata.ipv6_unicast_enabled == TRUE)) {
        /* router ACL/PBR */
        process_ipv6_racl();
        process_ipv6_urpf();
        process_ipv6_fib();
    }
}
```

BAREFOOT NETWORKS

Hit/Miss Branching

<u>Example:</u> Use Route Lookup if Host Lookup Fails

```
control process_ipv4_fib {
    apply(ipv4_fib) {
        miss {
            apply(ipv4_fib_lpm);
        }
    }
}
```

Action Branching

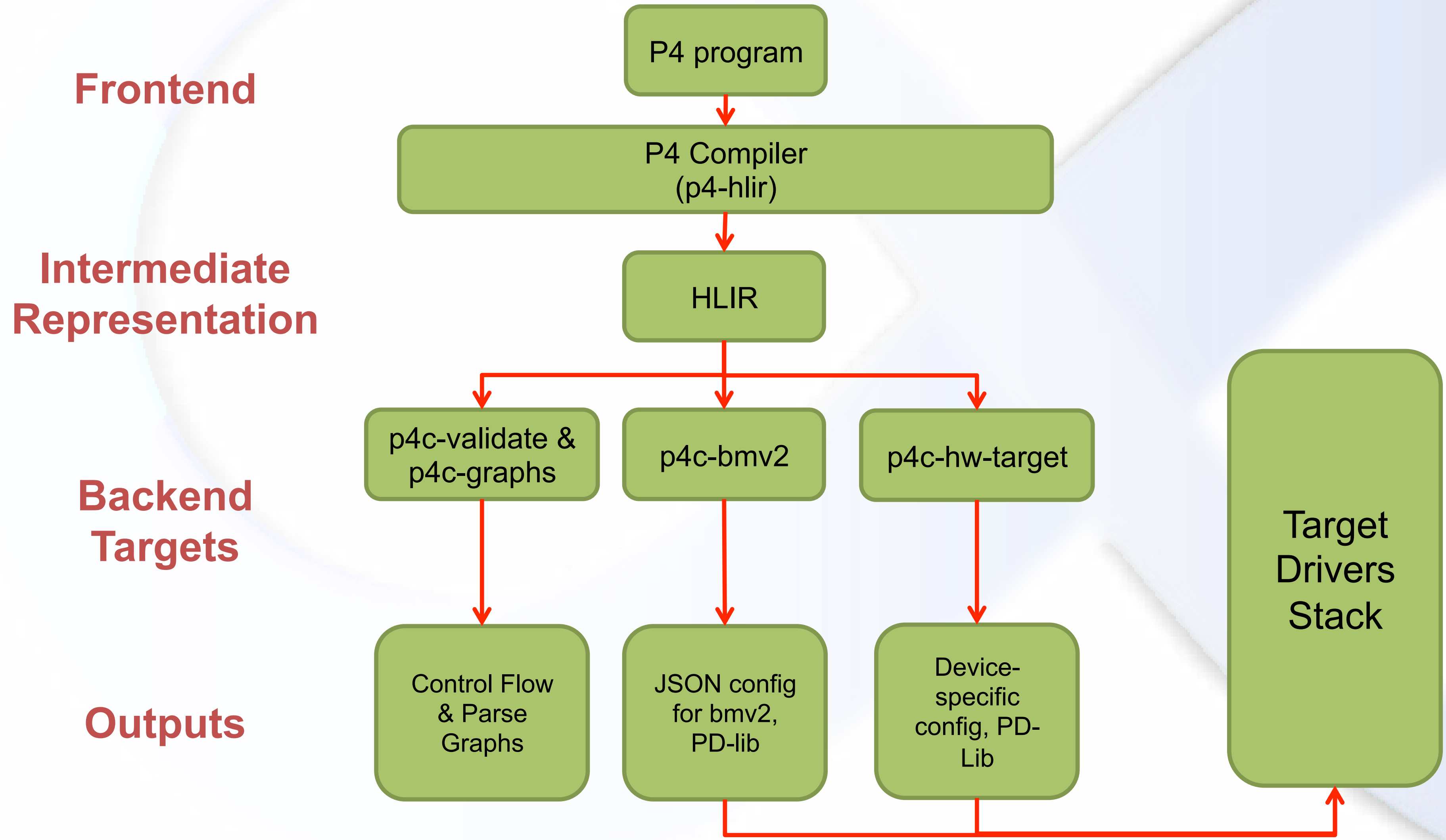
<u>Example:</u> Use per-router mac decapsulation

```
table router_mac {
    reads {
        l2_metadata.lkup_dst_mac : ternary;
        l2_metadata.bd           : ternary;
        ingress_metadata.src_port: ternary;
    }
    actions {
        nop;
        enable_ipv4_lookup;
        enable_ipv6_lookup;
        enable_mpls_decap;
        enable_mim_decap;
}
control process_router_mac_lookup {
    apply(router_mac) {
        enable_ipv4_lookup { process_ipv4_fib(); }
        enable_ipv6_lookup { process_ipv6_fib(); }
        enable_mpls_decap  { process_mpls_label_lookup(); }
           /* etc. */
    }
}
```

BAREFOOT NETWORKS

P4 Compiler Overview

P4 Modular Compilation



Modular Compiler Overview

- **P4 code is translated to High-Level Intermediate Representation (HLIR)**
  - Similar to AST (Abstract Syntax Trees)
  - Currently represented as a hierarchy of Python objects
  - Frees backend developers from the burden of syntax analysis and target-independent semantic checks
- **Multiple backends**
  - Code generators for various targets
    - software switches
    - network interface cards
    - packet processors / NPUs
    - FPGAs, GPUs, ASICs
  - Validators and graph generators
  - Run-time API generators

Dependency Analysis

BAREFOOT NETWORKS

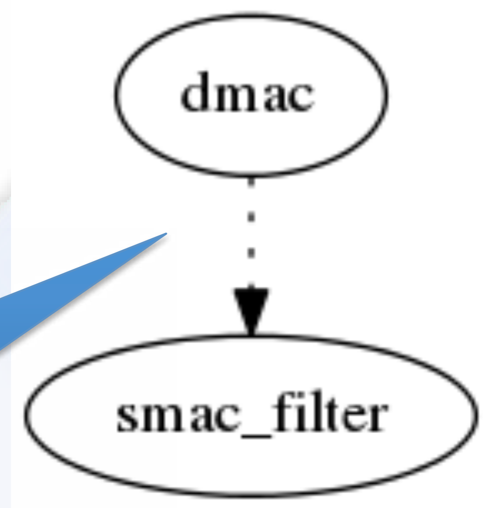
Types of dependencies

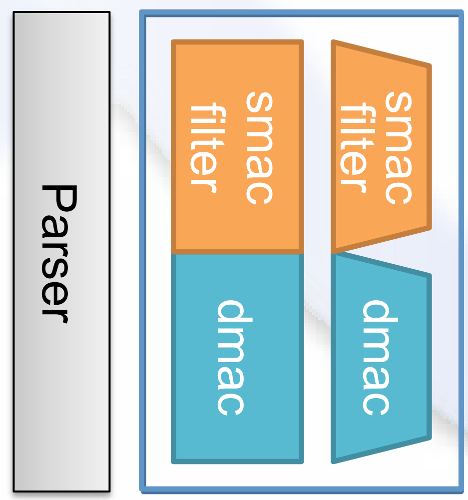
- Dependencies are inferred from target-independent P4 program analysis
- Independent tables
- Match Dependency
- Action Dependency
- Successor Dependency
- Reverse Read Dependency

Independent Tables

```
action ing_drop() {
    modify_field(ing_metadata.drop, 1);
}

action set_egress_port(egress_port) {
    modify_field(ing_metadata.egress_spec,
                 egress_port);
}

table dmac {
    reads {
        ethernet.dstAddr : exact;
    }
    actions {
        nop;
        set_egress_port;
    }
}

table smac_filter {
    reads {
        ethernet.srcAddr : exact;
    }
    actions {
        nop;
        ing_drop;
    }
}

control ingress {
    apply(dmac);
    apply(smac_filter);
}
```

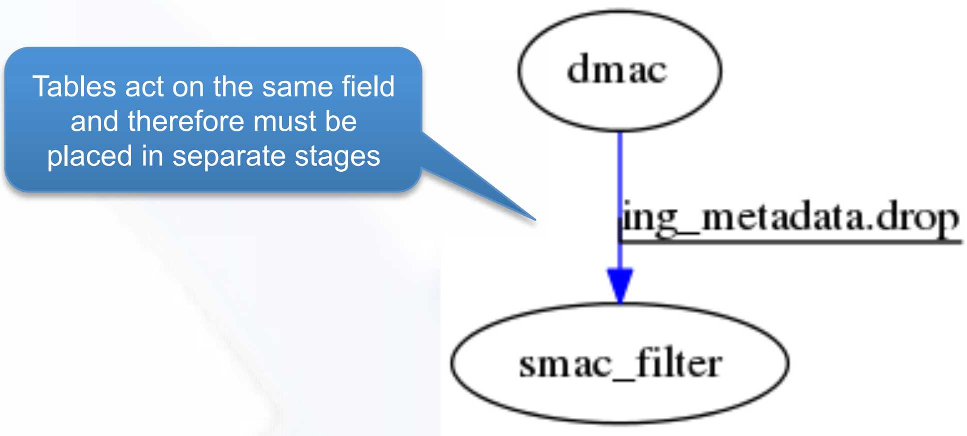
Tables are independent: both matching and action execution can be done in parallel

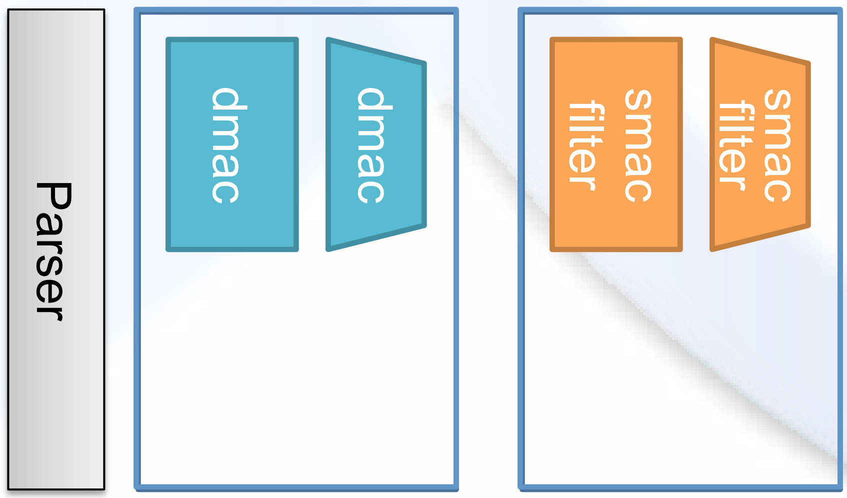




Action Dependency

```
action ing_drop() {
    modify_field(ing_metadata.drop, 1);
}

action set_egress_port(egress_port) {
    modify_field(ing_metadata.egress_spec,
                 egress_port);
}

table dmac {
    reads {
        ethernet.dstAddr : exact;
    }
    actions {
        nop;
        ing_drop;
        set_egress_port;
    }
    size : 131072;
}

table smac_filter {
    reads {
        ethernet.srcAddr : exact;
    }
    actions {
        nop;
        ing_drop;
    }
}

control ingress {
    apply(dmac);
    apply(smac_filter);
}
```

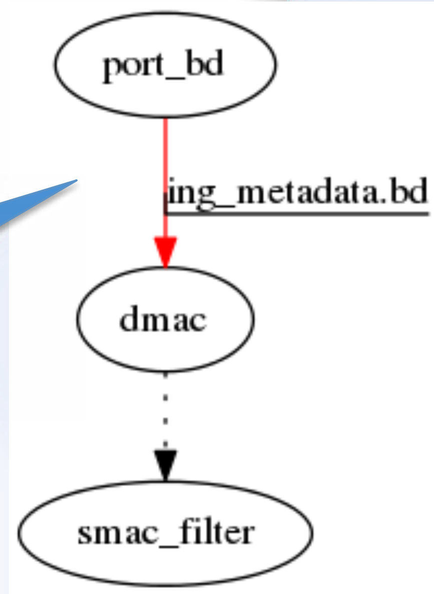


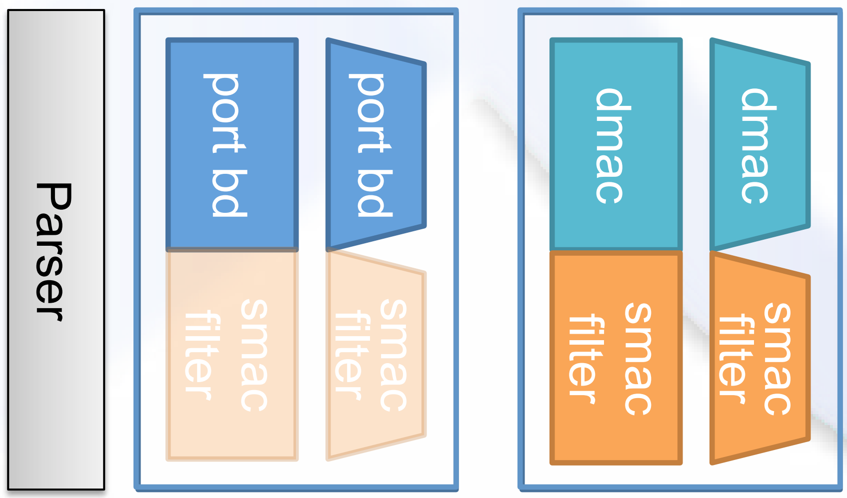


Match Dependency

```
action set_bd(bd) {
    modify_field(ing_metadata.bd, bd);
}

table port_bd {
    reads {
        ing_metadata.ingress_port : exact;
    }
    actions {
        set_bd;
    }
    size : 256;
}

table dmac {
    reads {
        ethernet.dstAddr : exact;
        ing_metadata.bd  : exact;
    }
    actions {
        nop;
        set_egress_port;
    }
    size : 131072;
}

table smac_filter {
    reads {
        ethernet.srcAddr : exact;
    }
    actions {
        nop;
        ing_drop;
    }
}

control ingress {
    apply(port_bd);
    apply(dmac);
    apply(smac_filter);
}
```

The second table matches on the field, modified by the first.

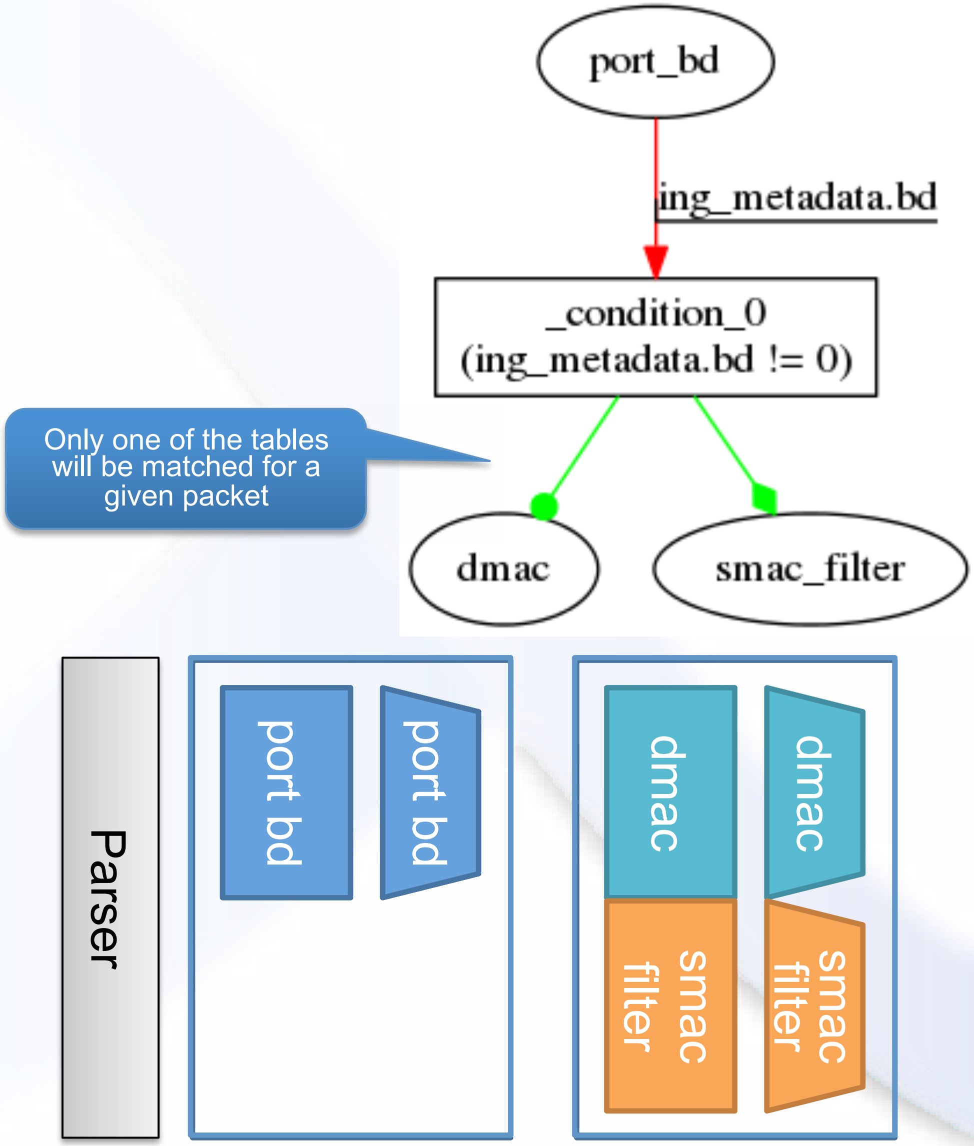




BAREFOOT NETWORKS

Successor Dependency

```
action set_bd(bd) {
    modify_field(ing_metadata.bd, bd);
}

table port_bd {
    reads {
        ing_metadata.ingress_port : exact;
    }
    actions {
        set_bd;
    }
    size : 256;
}

table dmac {
    reads {
        ethernet.dstAddr : exact;
        ing_metadata.bd  : exact;
    }
    actions {
        nop;
        ing_drop;
        set_egress_port;
    }
    size : 131072;
}

table smac_filter {
    reads {
        ethernet.srcAddr : exact;
    }
    actions {
        nop;
        ing_drop;
    }
}

control ingress {
    apply(port_bd);

    if (ing_metadata.bd != 0) {
        apply(dmac);
    } else {
        apply(smac_filter);
    }
}
```

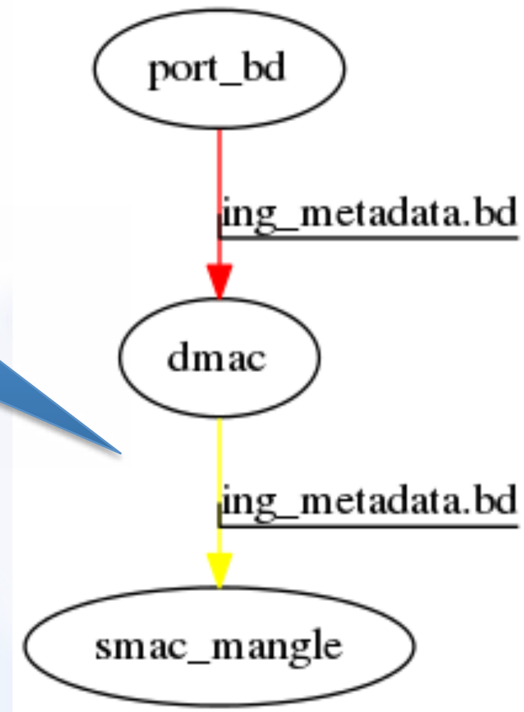


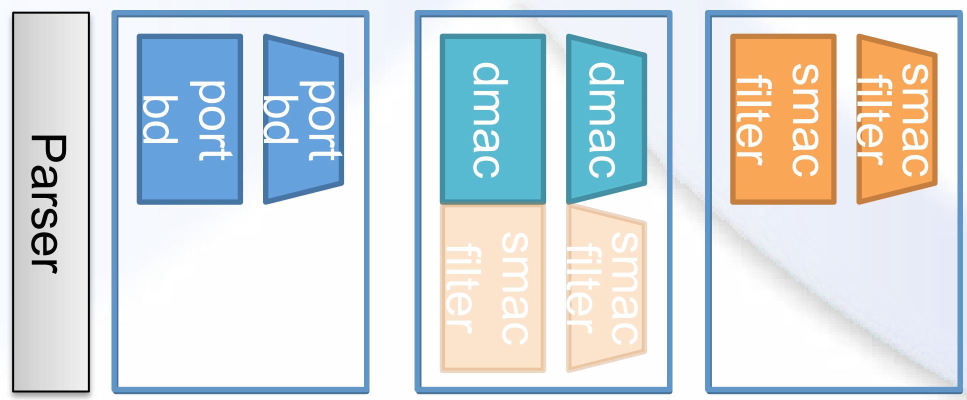
BAREFOOT NETWORKS

Reverse Read Dependency

```
action set_bd(bd) {
    modify_field(ing_metadata.bd, bd);
}

table port_bd {
    reads {
        ing_metadata.ingress_port : exact;
    }
    actions {
        set_bd;
    }
    size : 256;
}

table dmac {
    reads {
        ethernet.dstAddr : exact;
        ing_metadata.bd  : exact;
    }
    actions {
        nop;
        ing_drop;
        set_egress_port;
    }
    size : 131072;
}

table smac_mangle {
    reads {
        ethernet.srcAddr : exact;
    }
    actions {
        nop;
        set_bd;
    }
}

control ingress {
    apply(port_bd);
    apply(dmac);
    apply(smac_mangle);
}
```

The third table modifies a field used by the second table for matching. Therefore the action cannot take place before the second table matches.

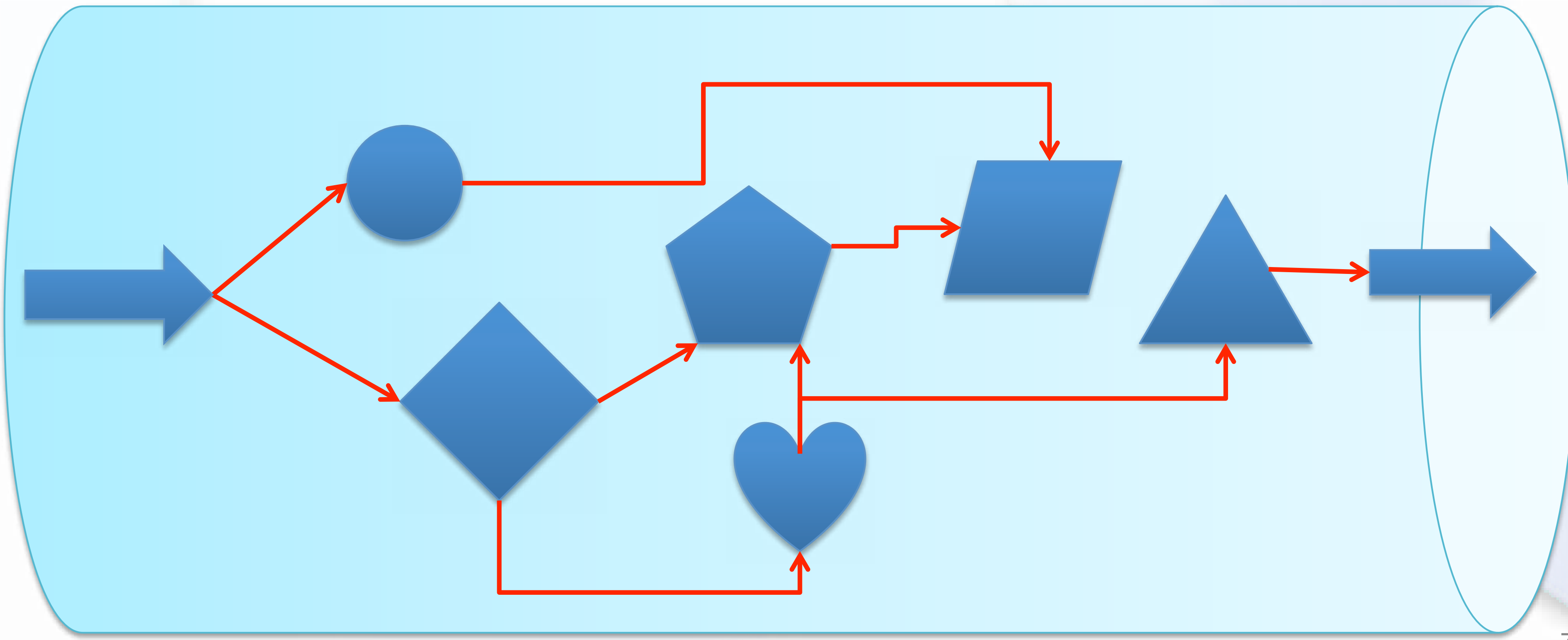




Automatic API Generation

Network Device API Basics

- **Object Definitions (Schema)**
  - Reflects the object properties and methods
- **Object Relationships (Behavior)**
  - The quality of the API is directly dependent on how well the object relationships are specified



BAREFOOT NETWORKS

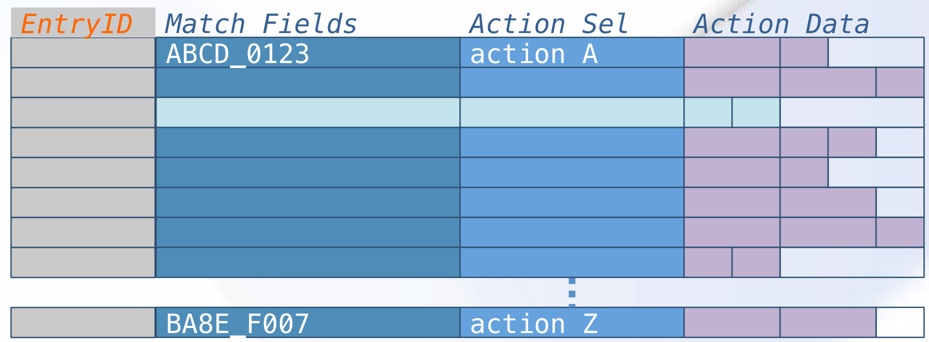
P4 is an Ideal Base for a Network APIs

- **Clearly defined objects**
  - Tables
  - Counters
  - Meters
  - Registers
- **Unambiguously defined relationships**
  - Control Flow Functions
- **Idea:**
  - Each of fundamental P4 objects has a “natural” schema

BAREFOOT NETWORKS

Tables

- **Uniform representation**
  - Primary key: Entry ID
  - Match Fields
  - Action
  - Action Data
    - Depends on the action
- **Operations**
  - Entry Add
    - (Match Fields, Action, Action Data) → Entry ID
  - Entry Get
    - (Entry ID) → (Match Fields, Action, Action Data)
  - Entry Delete
    - (Entry ID) →
  - Entry Modify
    - (Entry ID, Action, Action Data) →
  - Entry Lookup
    - (Match Fields, [Action, Action Data]) → Entry ID
  - Table Traverse
    - → [EntryID0, EntryID1, … EntryIDn]
  - Table Default_Action Set
    - (Action, Action Data) →
  - Table Default_Action Get
    - → (Action, Action Data)
  - Table Default Action Clear
    - →



- **Other Operations**
  - Table Size Get
  - Table Occupancy Get
  - Table Clear

Example API. Match & Action Specs

myprog.p4

```
action a1(p11, p12) {…}
action a2(p21, p22, p23) {…}
action a3() {…}

table t1 {
    reads {
       meta.f1 : exact;
       meta.f2 : ternary;
       h1      : valid;
    }
```

pd_myprog.h

```
typedef struct p4_pd_myprog_a1_action_spec {
    <type> p11;
    <type> p12;
} p4_pd_myprog_a1_action_spec_t;

typedef struct p4_pd_myprog_a2_action_spec {
    <type> p21;
    <type> p22;
    <type> p23;
} p4_pd_myprog_a2_action_spec_t;

typedef struct p4_pd_myprog_a3_action_spec {
} p4_pd_myprog_a3_action_spec_t;

typedef struct p4_pd_myprog_t1_match_spec {
    <type>   meta_f1;
    <type>   meta_f2;
    <type>   meta_f2_mask;
    uint8_t  h1_valid;
} p4_pd_myprog_t1_match_spec_t;
```

exact: f
ternary: f and f_mask
lpm: f and f_prefix_len
valid: f_valid
range: f_min and f_max

Example API. Entry Add

myprog.p4

```
action a1(p11, p12) {…}
action a2(p21, p22, p23) {…}
action a3() {…}

table t1 {
    reads {
       meta.f1 : exact;
       meta.f2 : ternary;
       h1      : valid;
    }
    actions {
       a1;
       a2;
       a3;
    }
}
```

pd_myprog.h

```
p4_pd_status_t p4_pd_myprog_t1_entry_add_with_a1(
    p4_pd_target_t                      device_target,
    p4_pd_session_t                     session_handle,
    p4_pd_priority_t                    priority,
    const p4_pd_myprog_t1_match_spec_t  *match_spec,
    const p4_pd_myprog_a1_action_spec_t *action_spec,
    p4_pd_entry_handle_t                *entry_hdl);

p4_pd_status_t p4_pd_myprog_t1_entry_add_with_a2(
    p4_pd_target_t                      device_target,
    p4_pd_session_t                     session_handle,
    p4_pd_priority_t                    priority,
    const p4_pd_myprog_t1_match_spec_t  *match_spec,
    const p4_pd_myprog_a2_action_spec_t *action_spec,
    p4_pd_entry_handle_t                *entry_hdl);

p4_pd_status_t p4_pd_myprog_t1_entry_add_with_a3(
    p4_pd_target_t                      device_target,
    p4_pd_session_t                     session_handle,
    p4_pd_priority_t                    priority,
    const p4_pd_myprog_t1_match_spec_t  *match_spec,
    const p4_pd_myprog_a3_action_spec_t *action_spec,
    p4_pd_entry_handle_t                *entry_hdl);
```

Example API. Entry Modify

myprog.p4

```
action a1(p11, p12) {…}
action a2(p21, p22, p23) {…}
action a3() {…}

table t1 {
    reads {
       meta.f1 : exact;
       meta.f2 : ternary;
       h1      : valid;
    }
    actions {
       a1;
       a2;
       a3;
    }
}
```

pd_myprog.h

```
p4_pd_status_t p4_pd_myprog_t1_entry_modify_with_a1(
    p4_pd_target_t                      device_target,
    p4_pd_session_t                     session_handle,
    p4_pd_entry_handle_t                entry_hdl,
    const p4_pd_myprog_a1_action_spec_t  *action_spec);

p4_pd_status_t p4_pd_myprog_t1_entry_modify_with_a2(
    p4_pd_target_t                      device_target,
    p4_pd_session_t                     session_handle,
    p4_pd_entry_handle_t                entry_hdl,
    const p4_pd_myprog_a2_action_spec_t  *action_spec);

p4_pd_status_t p4_pd_myprog_t1_entry_modify_with_a3(
    p4_pd_target_t                      device_target,
    p4_pd_session_t                     session_handle,
    p4_pd_entry_handle_t                entry_hdl,
    const p4_pd_myprog_a3_action_spec_t  *action_spec);
```

Example API. Entry Delete and Lookup

myprog.p4

```
action a1(p11, p12) {…}
action a2(p21, p22, p23) {…}
action a3() {…}


table t1 {
    reads {
       meta.f1 : exact;
       meta.f2 : ternary;
       h1      : valid;
    }
    actions {
       a1;
       a2;
       a3;
    }
}
```

pd_myprog.h

```
p4_pd_status_t p4_pd_myprog_t1_entry_delete(
    p4_pd_target_t                         device_target,
    p4_pd_session_t                        session_handle,
    p4_pd_entry_handle_t                   entry_hdl);

p4_pd_status_t p4_pd_myprog_t1_entry_lookup(
    p4_pd_target_t                         device_target,
    p4_pd_session_t                        session_handle,
    const p4_pd_myprog_t1_match_spec_t    *match_spec,
    p4_pd_entry_handle_t                  *entry_hdl);
```

BAREFOOT NETWORKS

Example API. Entry Get

myprog.p4

```
action a1(p11, p12) {…}
action a2(p21, p22, p23) {…}
action a3() {…}

table t1 {
    reads {
       meta.f1 : exact;
       meta.f2 : ternary;
       h1      : valid;
    }
    actions {
       a1;
       a2;
       a3;
    }
}
```

pd_myprog.h

```
typedef enum {
   P4_PD_MYPROG_ACTION_A1,
   P4_PD_MYPROG_ACTION_A2,
   P4_PD_MYPROG_ACTION_A3,
   …
   P4_PD_MYPROG_ACTION_COUNT;
} p4_pd_myprog_actions_t;

typedef union {
   p4_pd_myprog_a1_action_spec_t a1;
   p4_pd_myprog_a2_action_spec_t a2;
   . . .
} p4_pd_myprog_action_spec_t;

p4_pd_status_t p4_pd_myprog_t1_entry_get(
    p4_pd_target_t                      device_target,
    p4_pd_session_t                     session_handle,
    p4_pd_entry_handle_t                entry_hdl,
    p4_pd_myprog_t1_match_spec_t       *match_spec,
    p4_pd_myprog_actions_t             *action,
    p4_pd_myprog_action_spec_t         *action_spec_t);
```

Example API. Default Action APIs

myprog.p4

```
action a1(p11, p12) {…}
action a2(p21, p22, p23) {…}
action a3() {…}

table t1 {
    reads {
       meta.f1 : exact;
       meta.f2 : ternary;
       h1      : valid;
    }
    actions {
       a1;
       a2;
       a3;
    }
}
```

pd_myprog.h

```
p4_pd_status_t p4_pd_myprog_t1_set_default_action_a1(
    p4_pd_target_t                      device_target,
    p4_pd_session_t                     session_handle,
    const p4_pd_myprog_a1_action_spec_t  *action_spec);

p4_pd_status_t p4_pd_myprog_t1_set_default_action_a2(
    p4_pd_target_t                      device_target,
    p4_pd_session_t                     session_handle,
    const p4_pd_myprog_a2_action_spec_t  *action_spec);

p4_pd_status_t p4_pd_myprog_t1_set_default_action_a3(
    p4_pd_target_t                      device_target,
    p4_pd_session_t                     session_handle,
    const p4_pd_myprog_a3_action_spec_t  *action_spec);

p4_pd_status_t p4_pd_myprog_t1_clear_default_action(
    p4_pd_target_t                      device_target,
    p4_pd_session_t                     session_handle);
```

Example API. Default Action APIs

myprog.p4

```
action a1(p11, p12) {…}
action a2(p21, p22, p23) {…}
action a3() {…}

table t1 {
    reads {
       meta.f1 : exact;
       meta.f2 : ternary;
       h1      : valid;
    }
    actions {
       a1;
       a2;
       a3;
    }
}
```

pd_myprog.h

```
p4_pd_status_t p4_pd_myprog_t1_set_default_action_a1(
    p4_pd_target_t                      device_target,
    p4_pd_session_t                     session_handle,
    const p4_pd_myprog_a1_action_spec_t  *action_spec);

p4_pd_status_t p4_pd_myprog_t1_set_default_action_a2(
    p4_pd_target_t                      device_target,
    p4_pd_session_t                     session_handle,
    const p4_pd_myprog_a2_action_spec_t  *action_spec);

p4_pd_status_t p4_pd_myprog_t1_set_default_action_a3(
    p4_pd_target_t                      device_target,
    p4_pd_session_t                     session_handle,
    const p4_pd_myprog_a3_action_spec_t  *action_spec);

p4_pd_status_t p4_pd_myprog_t1_clear_default_action(
    p4_pd_target_t                      device_target,
    p4_pd_session_t                     session_handle);
```

Counters

- **Individual Counter Operations**
  - Get
    - (Counter Index or Entry ID) → Value
  - Clear
    - (Counter Index or Entry ID) →
  - Set (optional)
    - (Counter Index or Entry ID, Value) →
- **Counter Array Operations**
  - Width Get
  - Array size Get
  - Get All
  - Clear All

Example API

myprog.p4

```
counter c1 {
    type:  packets_and_bytes;
  direct:  t1;
};

counter c2 {
    type:           bytes;
    instance_count: 1000;
}
```

pd_myprog.h

```
p4_pd_status_t p4_pd_myprog_c1_get(
    p4_pd_target_t                    device_target,
    p4_pd_session_t                   session_handle,
    p4_pd_entry_handle_t              entry_hdl,
    uint64_t                         *packets,
    uint64_t                         *bytes);

p4_pd_status_t p4_pd_myprog_c2_get(
    p4_pd_target_t                    device_target,
    p4_pd_session_t                   session_handle,
    uint32_t                          counter_idx,
    uint64_t                         *bytes);
```

Meters

- **Individual Meter Operations**
  - Set
    - (Meter Index or EntryID,
      Committed Rate, Committed Birst, Peak Rate, Peak Birst)
    - Is that the only option?
    - What about different meter types (color-blind/color-aware, single rate?)
      - Are all meters in the array of the same type?
    - Who standardizes the units (bits, bytes, kbits, Mbytes, etc.)?
    - Who standardizes the colors?
  - Get
    - (Meter Index or Entry ID) → (Settings)

Registers

- **Operations**
  - Set
    - (Register Index or Entry ID, value) →
  - Get
    - (Register Index or Entry ID) → value
- **C type for the value depends on register definition**
- **Optional Operations**
  - Width Get
  - Get All
  - Set All

Discussion

- **Pros**
  - Easy to understand and use
  - Some type checking
- **Cons**
  - N*3 + m functions per table
  - Very inconvenient for CLI implementation

BAREFOOT NETWORKS

Example API

myprog.p4

```
action a1(p11, p12) {…}
action a2(p21, p22, p23) {…}
action a3() {…}

table t1 {
    reads {
       meta.f1 : exact;
       meta.f2 : ternary;
       h1      : valid;
    }
    actions {
       a1;
       a2;
       a3;
    }
}
```

pd_myprog_style2.h

```
typedef enum {
    P4_PD_MYPROG_ACTION__A1,
    P4_PD_MYPROG_ACTION__A2,
    . . .
    P4_PD_MYPROG_ACTION__MAX
} p4_pd_myprog_actionid_t;

typedef enum {
    P4_PD_MYPROG_TABLE__T1,
    . . .
    P4_PD_MYPROG_TABLE__MAX
} p4_pd_myprog_table_id_t;

typedef enum {
    P4_PD_MYPROG_FIELD__META_F1,
    P4_PD_MYPROG_FIELD__META_F2,
    P4_PD_MYPROG_FIELD__META_F2__MASK,
    P4_PD_MYPROG_FIELD__H1__VALID,
    P4_PD_MYPROG_FIELD__P11,
    P4_PD_MYPROG_FIELD__P12,
    P4_PD_MYPROG_FIELD__P22,
    P4_PD_MYPROG_FIELD__P23,
    . . .
    P4_MD_MYPROG_FIELD__MAX
} p4_pd_myprog_field_id_t;
```

Example API

myprog.p4

```
action a1(p11, p12) {…}
action a2(p21, p22, p23) {…}
action a3() {…}

table t1 {
    reads {
       meta.f1 : exact;
       meta.f2 : ternary;
       h1      : valid;
    }
    actions {
       a1;
       a2;
       a3;
    }
}
```

pd_myprog_style2.h

```
const char *p4_pi_action2str(
    p4_pi_target_t     program_indicator,
    p4_pi_action_t     action_id);

p4_pi_action_t p4_pi_str2action(
    p4_pi_target_t     program_indicator,
    const char        *action_str);

const char *p4_pi_table2str(
    p4_pi_target_t     program_indicator,
    p4_pi_table_t     table_id);

p4_pi_table_t p4_pi_str2table(
    p4_pi_target_t     program_indicator,
    const char        *table_str);

const char *p4_pi_field2str(
    p4_pi_target_t     program_indicator,
    p4_pi_field_t      field_id);

p4_pi_field_t p4_pi_str2field(
    p4_pi_target_t     program_indicator,
    const char        *field_str);
```

BAREFOOT NETWORKS

Example API

myprog.p4

```
action a1(p11, p12) {…}
action a2(p21, p22, p23) {…}
action a3() {…}

table t1 {
    reads {
       meta.f1 : exact;
       meta.f2 : ternary;
       h1      : valid;
    }
    actions {
       a1;
       a2;
       a3;
    }
}
```

pd_myprog_style2.h

```
typedef struct {
 . . .
} p4_pi_entry_t;

p4_pi_status_t p4_pi_entry_init(
    p4_pi_target_t  program_indicator,
    p4_pi_table_t   table_id,
    p4_pi_entry_t  *entry);

p4_pi_status_t p4_pi_entry_action_set(
    p4_pi_entry_t    *entry,
    p4_pi_action_id_t  action_id);

p4_pi_status_t p4_pi_entry_field_set(
    p4_pi_entry_t    *entry,
    p4_pi_field_id_t  field_id,
    const void       *value);

p4_pi_status_t p4_pi_entry_add(
    p4_pi_target_t           device_target,
    p4_pi_session_handle_t  session_hdl,
    const p4_pi_entry_t     *entry,
    p4_pi_entry_id_t        *entry_id);
```

Another Option:

```
struct p4_pi_value {
   p4_pi_value_type_t type;
   union {
      uint8_t u8;
      uint16_t u16;
      uint32_t u32;
      uint64_t u64;
      . . .
   }
}
```

BAREFOOT NETWORKS

Example API

myprog.p4

```
action a1(p11, p12) {…}
action a2(p21, p22, p23) {…}
action a3() {…}

table t1 {
    reads {
       meta.f1 : exact;
       meta.f2 : ternary;
       h1      : valid;
    }
    actions {
       a1;
       a2;
       a3;
    }
}
```

pd_myprog_style2.h

```
p4_pi_status_t p4_pi_entry_add(
    p4_pi_target_t          device_target,
    p4_pi_session_handle_t  session_hdl,
    const p4_pi_entry_t    *entry,
    p4_pi_entry_id_t       *entry_id);

p4_pi_status_t p4_pi_entry_modify(
    p4_pi_target_t          device_target,
    p4_pi_session_handle_t  session_hdl,
    p4_pi_entry_id_t        entry_id
    const p4_pi_entry_t    *entry);

p4_pi_status_t p4_pi_entry_delete(
    p4_pi_target_t          device_target,
    p4_pi_session_handle_t  session_hdl,
    p4_pi_entry_id_t        entry_id);

p4_pi_status_t p4_pi_entry_get(
    p4_pi_target_t          device_target,
    p4_pi_session_handle_t  session_hdl,
    p4_pi_entry_id_t        entry_id
    p4_pi_entry_t          *entry);

p4_pi_status_t p4_pi_entry_lookup(
    p4_pi_target_t          device_target,
    p4_pi_session_handle_t  session_hdl,
    const p4_pi_entry_t    *entry,
    p4_pi_entry_id_t       *entry_id);
```

Other APIs

- **Get a list of Tables defined for a target**
- **Get a list of Actions for a Table**
- **Get a list of applicable fields for a Table+Action**
  - Clearly separated in match fields and action data fields
- **Get Table Attributes (size, etc.)**
- **Get Field Attributes (size, type, etc.)**
- **. . .**

Discussion

- **Pros**
  - Extremely flexible and doesn't depend on a program
  - Implementation can be fully data driven
  - P4 Program replacement possible
  - Potentially, new tables/actions/fields can be added on the fly for incremental compilation
  - Very Easy CLI and other tool implementation
- **Cons**
  - Almost no type checking
  - More function calls required
    - Separate calls to set each field

Conclusions

- **Uniform structure and small number of P4 objects allow APIs to be generated automatically**
  - In some cases, not all information can be derived
    - P4 v.1.1 typing will help with action_spec
  - P4 evolution need to include mechanisms to allow full, unambiguous API generation
- **Many API styles are possible**
  - API generators should be implemented as separate backends

Thank you