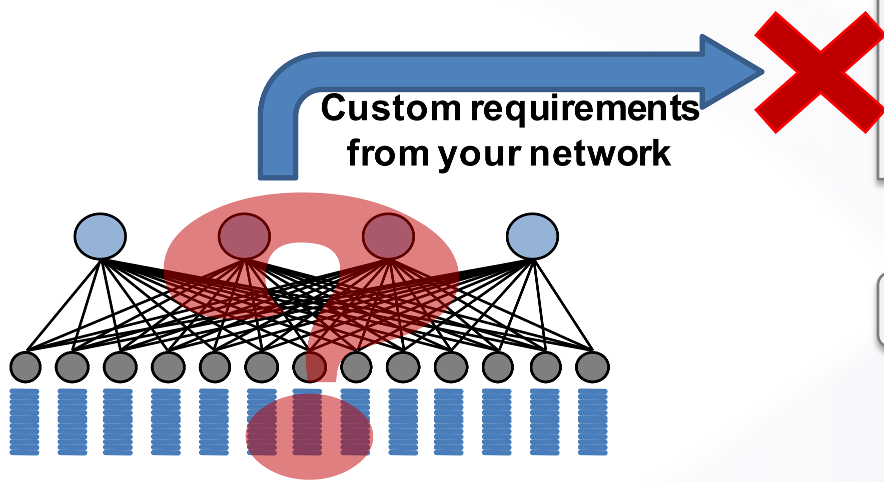
Programming The Network Data Plane in P4

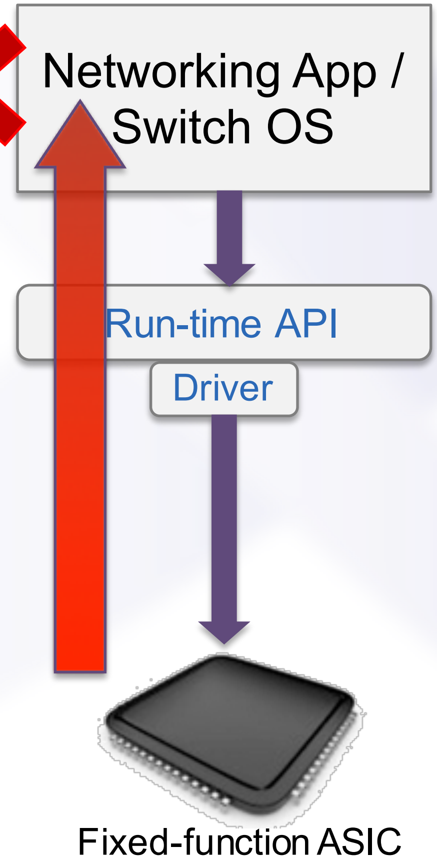
SIGCOMM'16
Tutorial

Barefoot Networks
Aug 2016

P4 Introduction

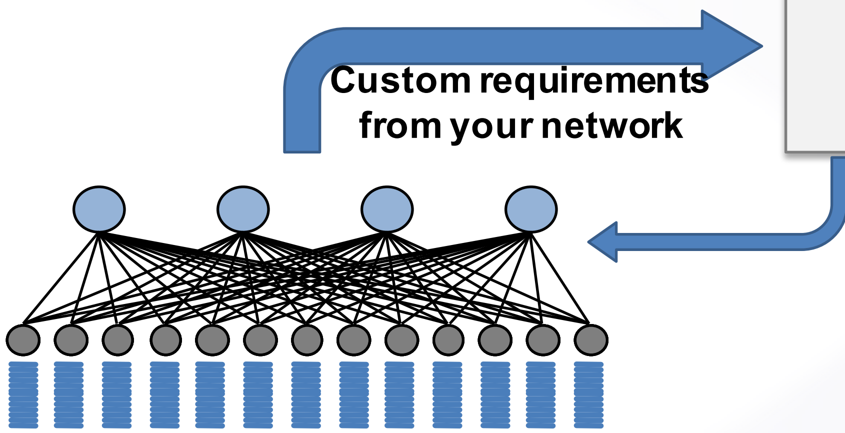
Status Quo: Bottom-up design

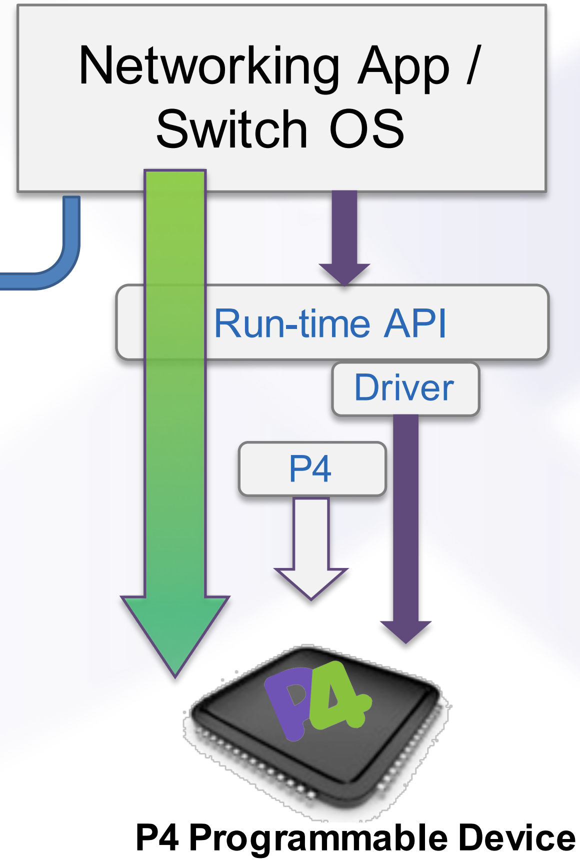




"This is *how I know* to process packets"
(i.e. the ASIC datasheet makes the rules)

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)

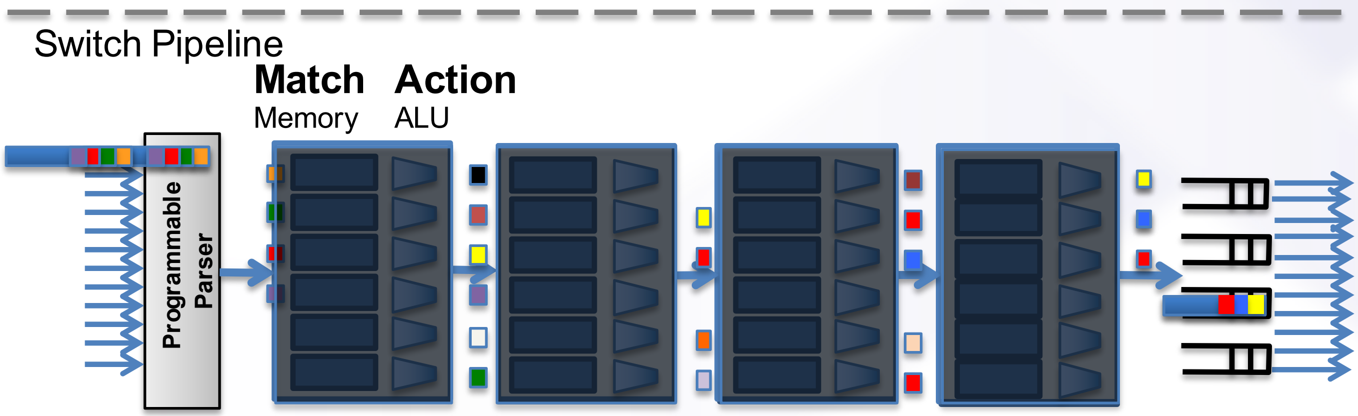
Programmable Network Devices

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

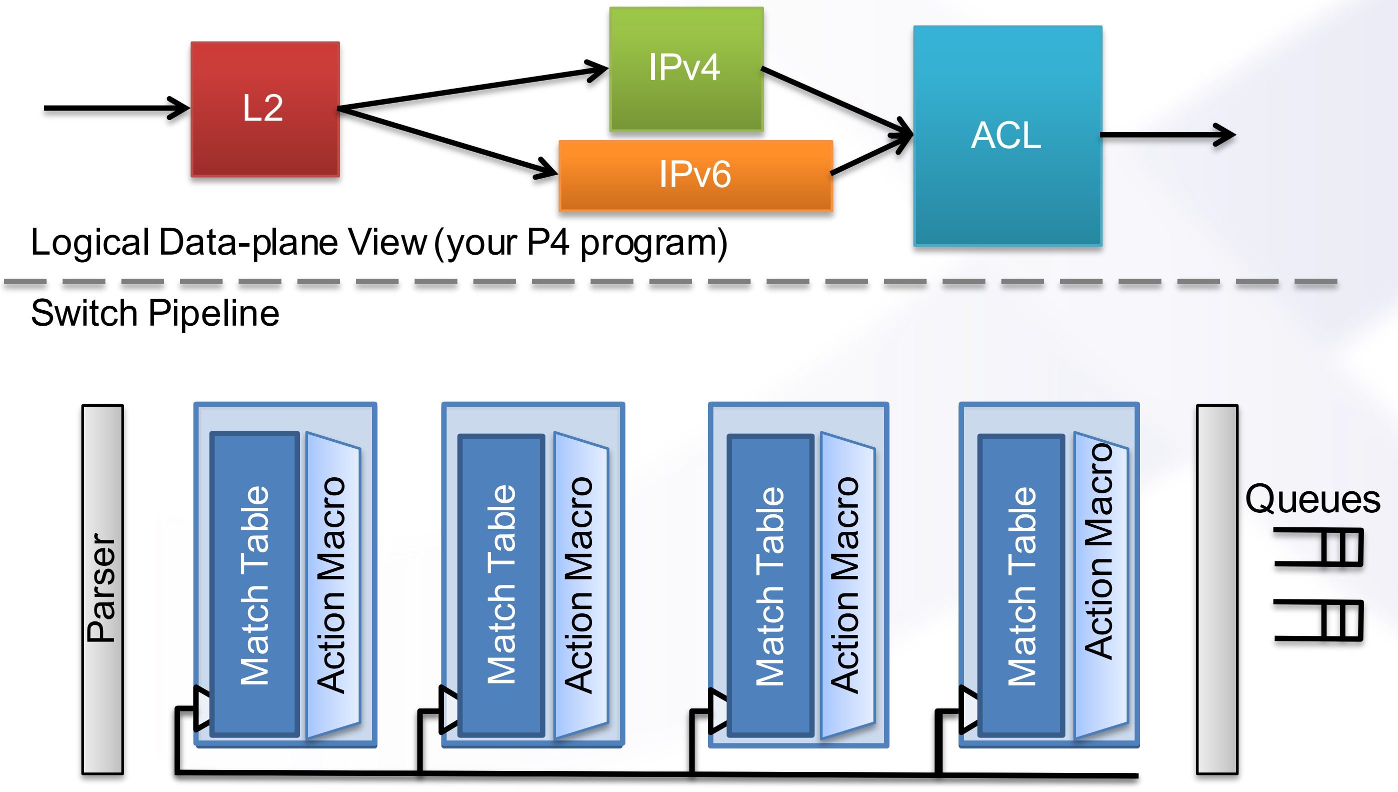
These devices let us tell them how to process packets.

Why we call it
Protocol Independent Packet Processing

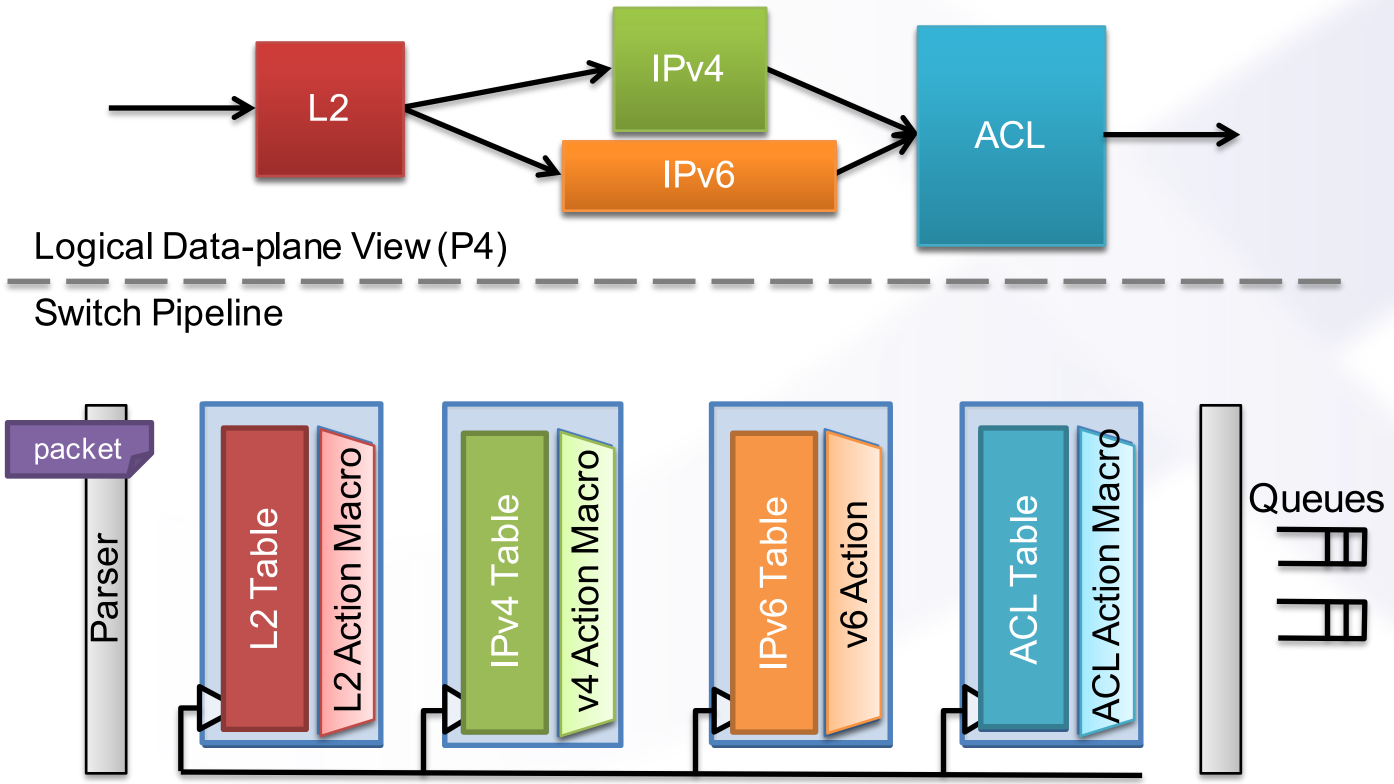
Protocol-Independent Switch Architecture (PISA)



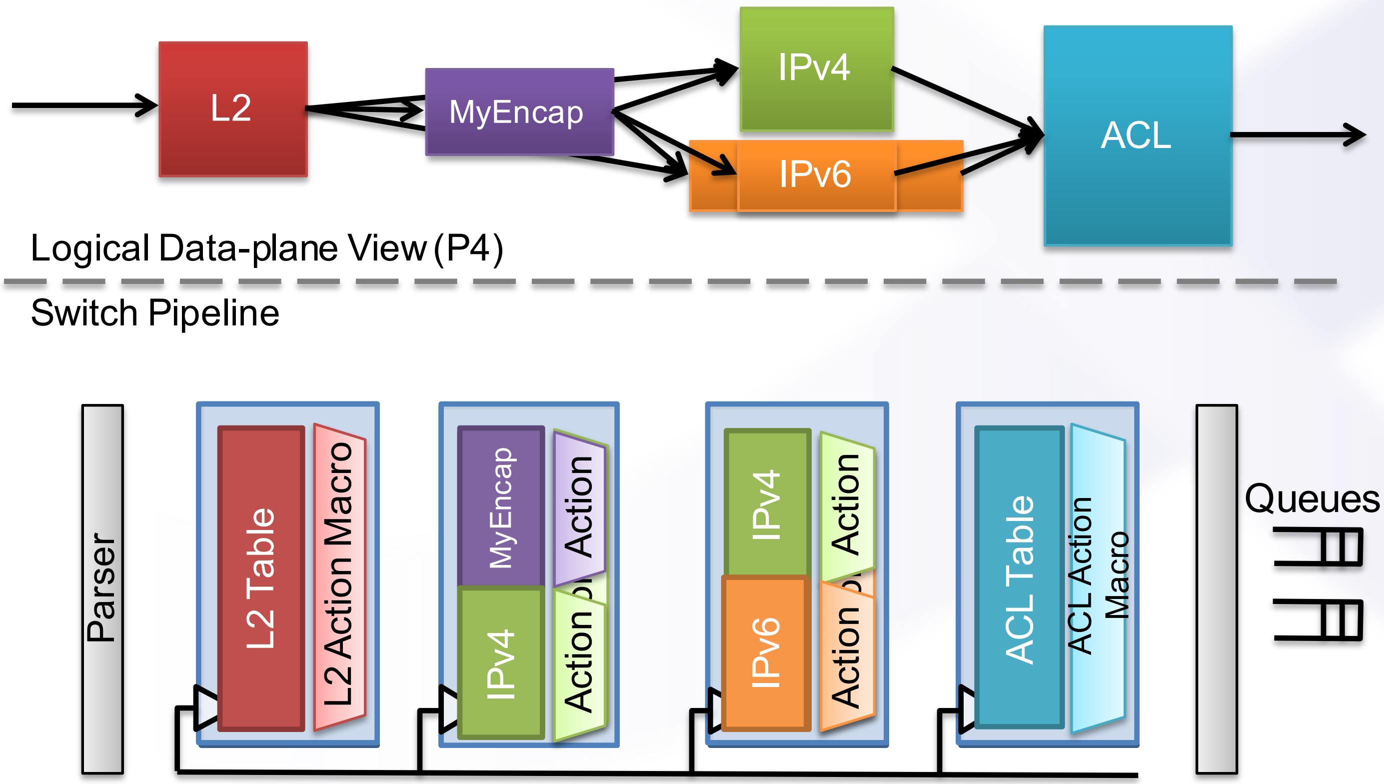
Protocol-Independent Switch Architecture (PISA)



Mapping to Physical Resources



Re-configurability



P4: Three Goals

Protocol independence

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

Target independence

- Program without knowledge of packet-processing device details
- Let compilers configure the target device

In-field Re-configurability

- Allow the users to change parsing and processing program in the field

What does this mean?

- **To network device vendors**
  - S/W programming practices and tools used in every phase
  - Extremely fast iteration and feature release
  - Differentiation in capabilities and performance
  - Can fix even data-plane bugs in the field

You wear both hats ☺

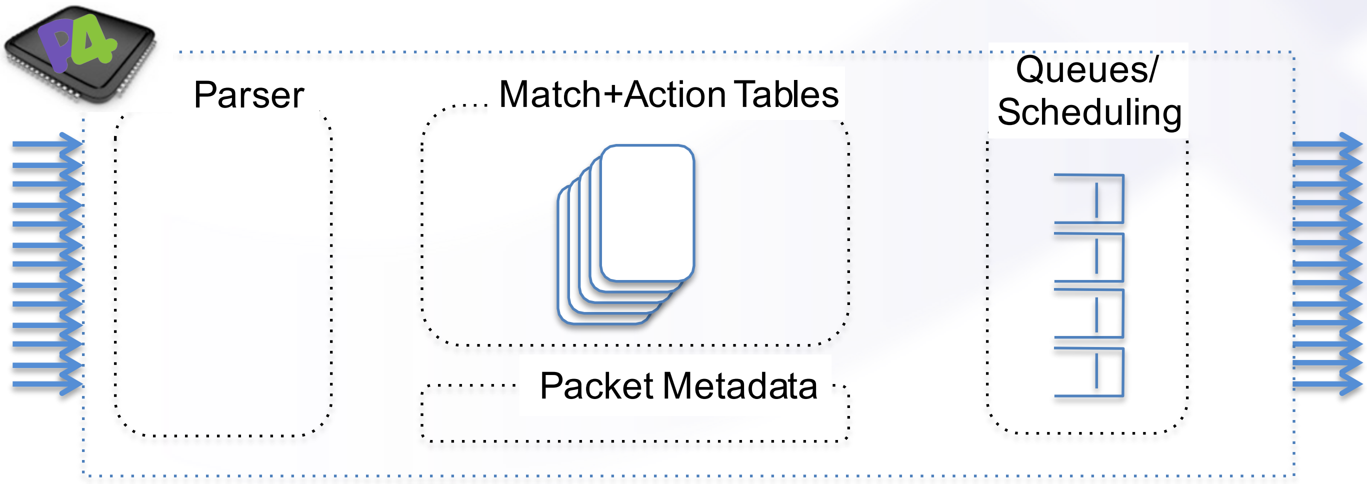
- **To large on-line service providers and carriers**
  - No more “black boxes” in the “white boxes”
  - Your devs can program, test, and debug your network devices all the way down
  - You keep your own ideas

Key benefits of programmable forwarding

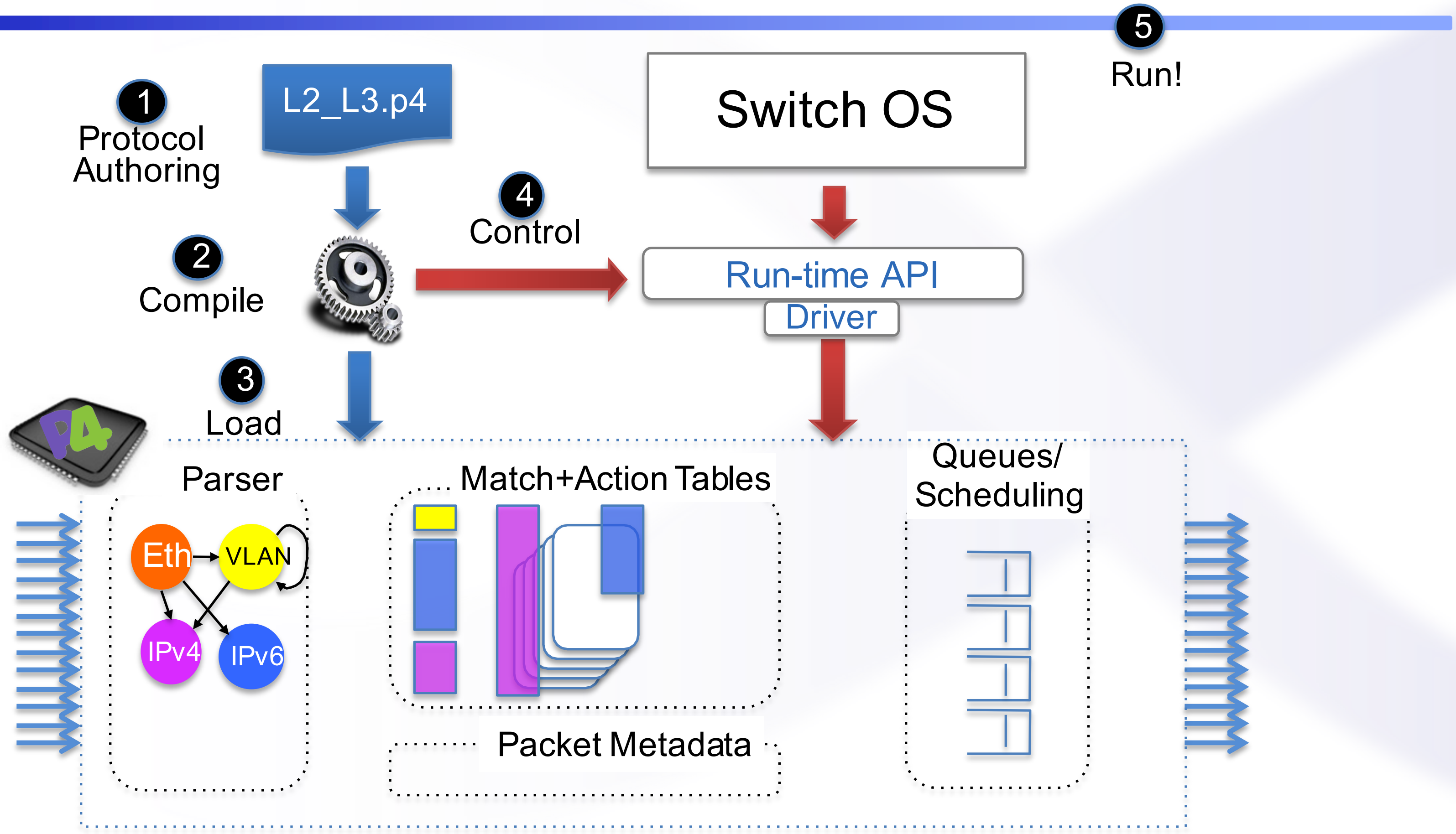
1. **New features**: Realize new protocols and behaviors very quickly
2. **Reduce complexity**: Remove unnecessary features and tables
3. **Efficient use of H/W resources**: Achieve biggest bang for buck
4. **Greater visibility**: New diagnostics, telemetry, OAM, etc.
5. **Modularity**: Compose forwarding behavior from libraries
6. **Portability**: Specify forwarding behavior once; compile to many devices
7. **Own your own network**: No need to wait for next chips or systems

P4-Based Workflow

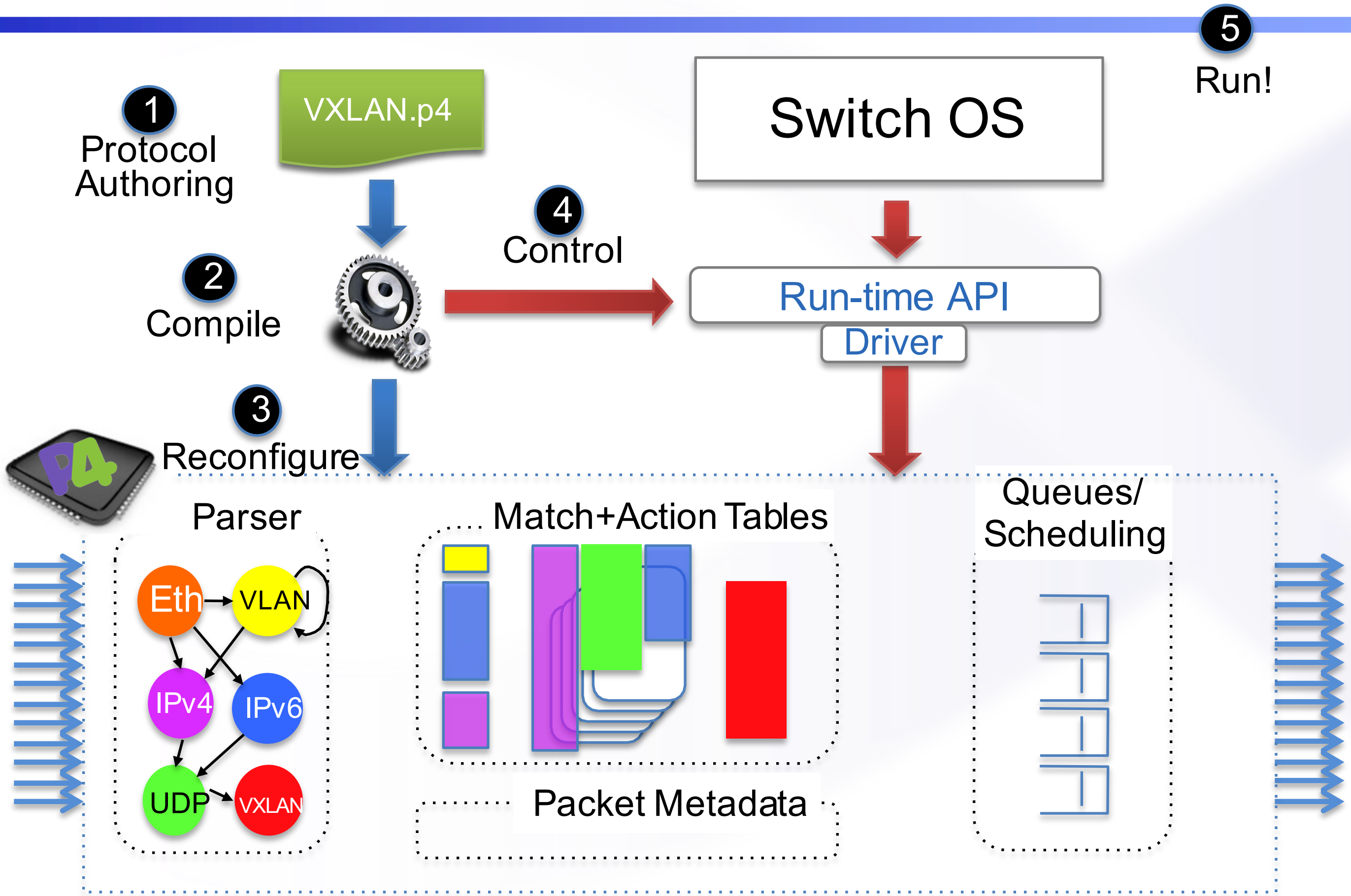
- **Device is not yet programmed**
  - Does not know about any packet formats or protocols



P4-Based Workflow

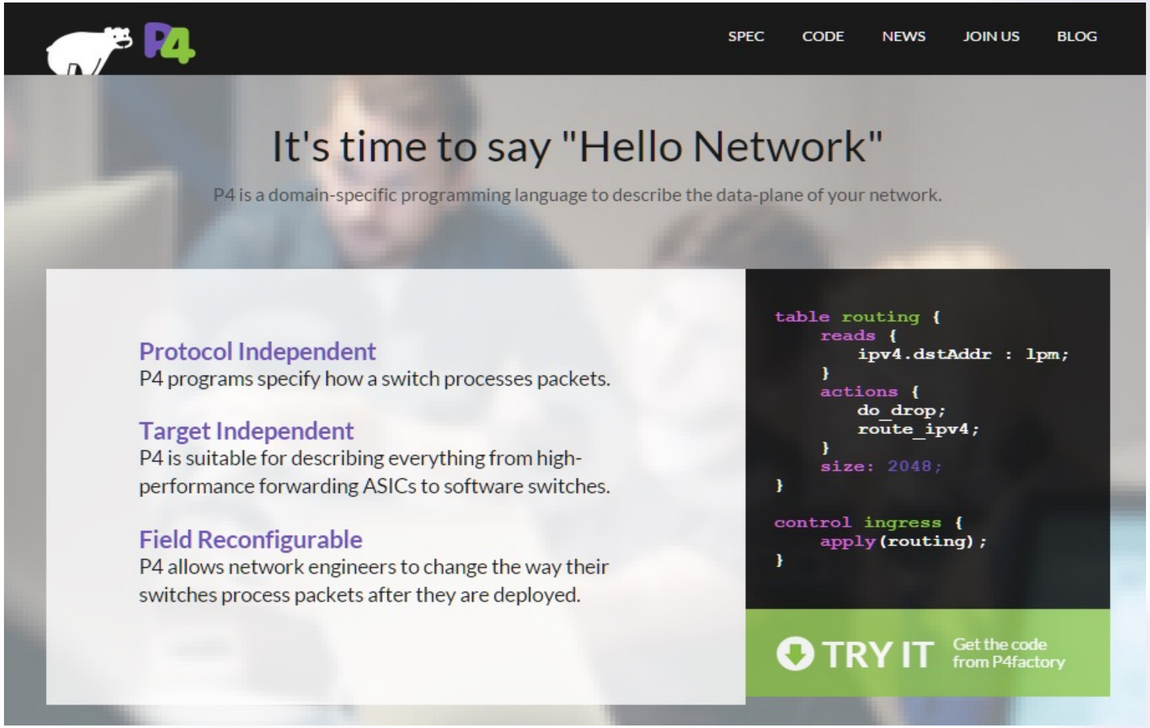


P4-Based Workflow



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**



P4.org Membership

Original P4 Paper Authors: Barefoot Networks, Google, intel, Microsoft, Princeton University, Stanford University

Operators: Alibaba Group 阿里巴巴集团, at&t, Bai du 百度, COMCAST, kt, Microsoft, SK telecom, Tencent 腾讯

Systems: BROCADE, CISCO, CORSA, DELL, Hewlett Packard Enterprise, HUAWEI, Inventec, Juniper Networks

Targets: AEPONYX (Moving the Cloud at the Speed of Light), Barefoot Networks, CAVIUM, EZchip, Global Hitech, Atomic Rules, Broadcom (connecting everything), centec networks, freescale, intel, Mellanox Technologies, Marvell, MoSys, Netronome, PLUMgrid, XILINX, NoviFlow (SDN made smarter), vmware

Academia: Cornell University, Politecnico Milano 1863, Princeton University, Stanford University, uni.lu Université du Luxembourg, Università della Svizzera italiana

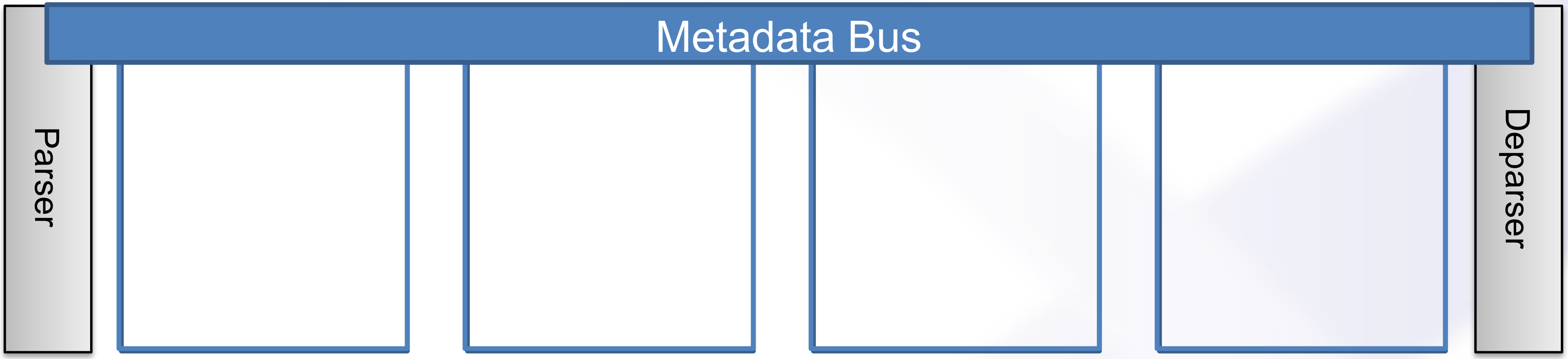
- Open source, evolving, domain-specific language
- Permissive Apache license, code on GitHub today

- Membership is free: contributions are welcome
- Independent, set up as a California nonprofit

P4 Concepts

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

The anatomy of a basic pipeline

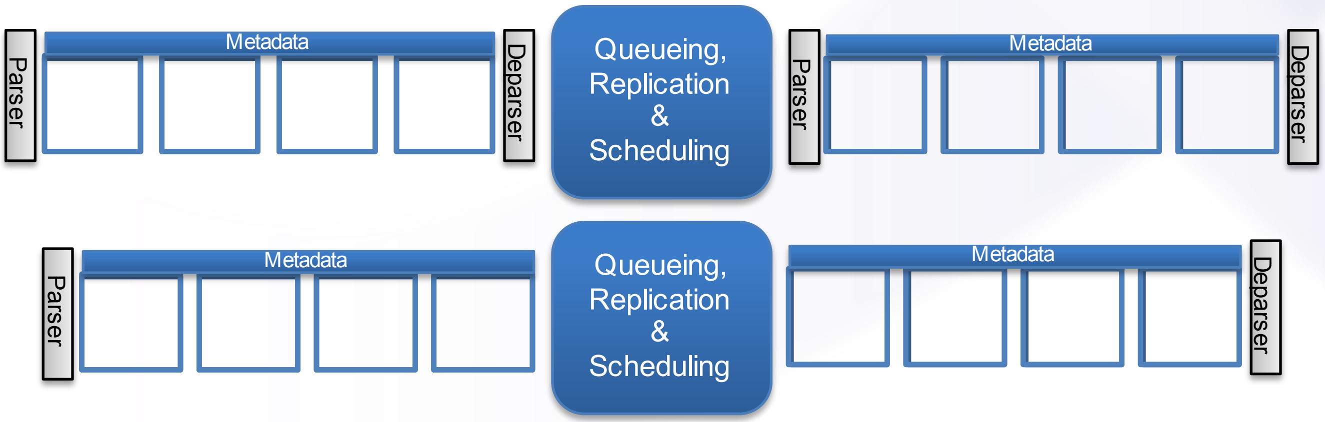


- **Parser**
  - Converts packet data into a metadata (Parsed Representation)
- **Match+Action Tables**
  - Operate on metadata
- **Deparser**
  - Converts metadata back into a serialized packet
- **Metadata Bus**
  - Carries the information within the pipeline

All are optional

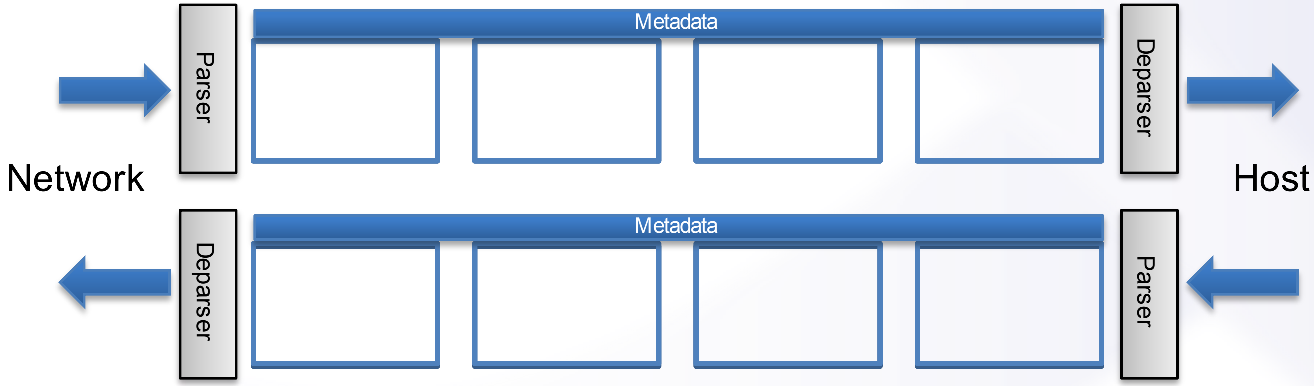
Anatomy of a Switch

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



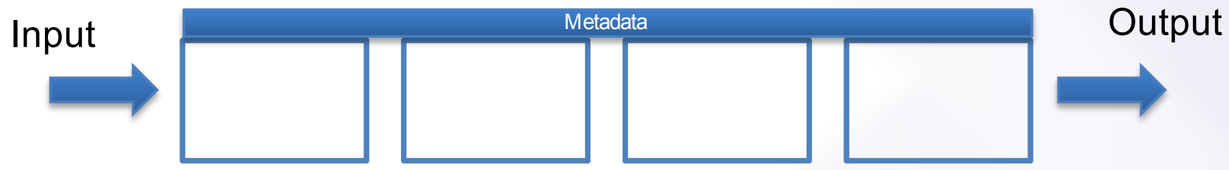
Anatomy of a NIC

- **Single or Dual Pipeline**



Anatomy of Protocol Plugin

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



P4 Program Sections

program.p4

Data Declarations

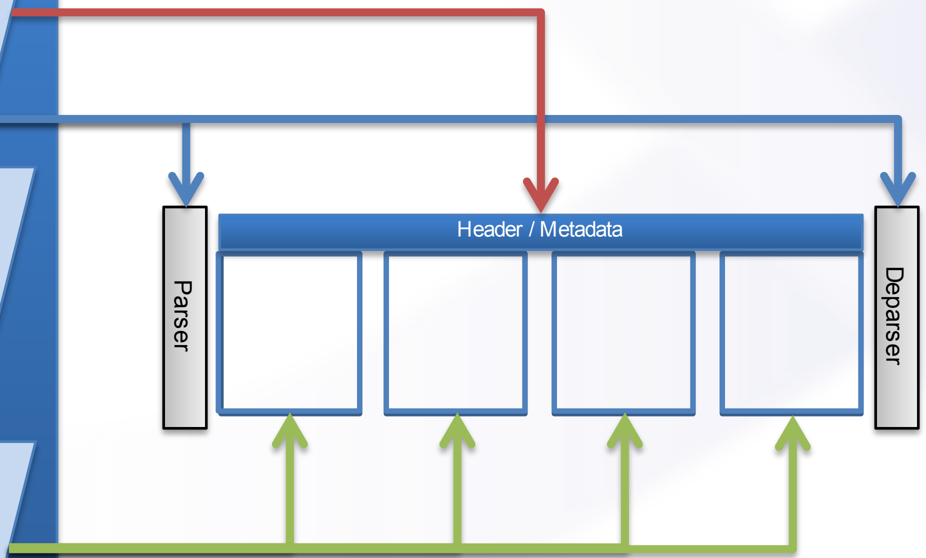
```
header_type   ethernet_t      { … }
header_type   l2_metadata_t   { … }

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;
}
```

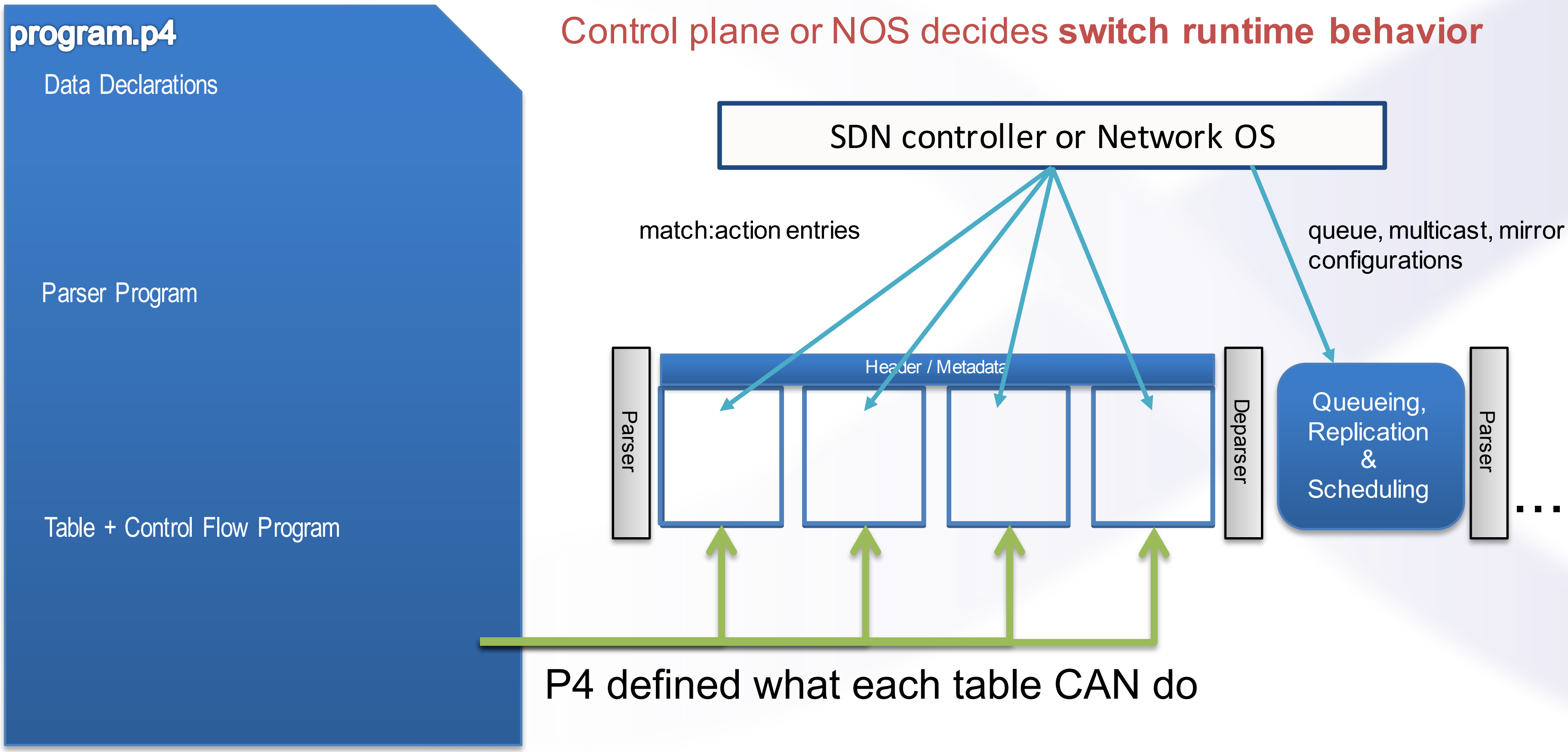
Table + Control Flow Program

```
table port_table { … }

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

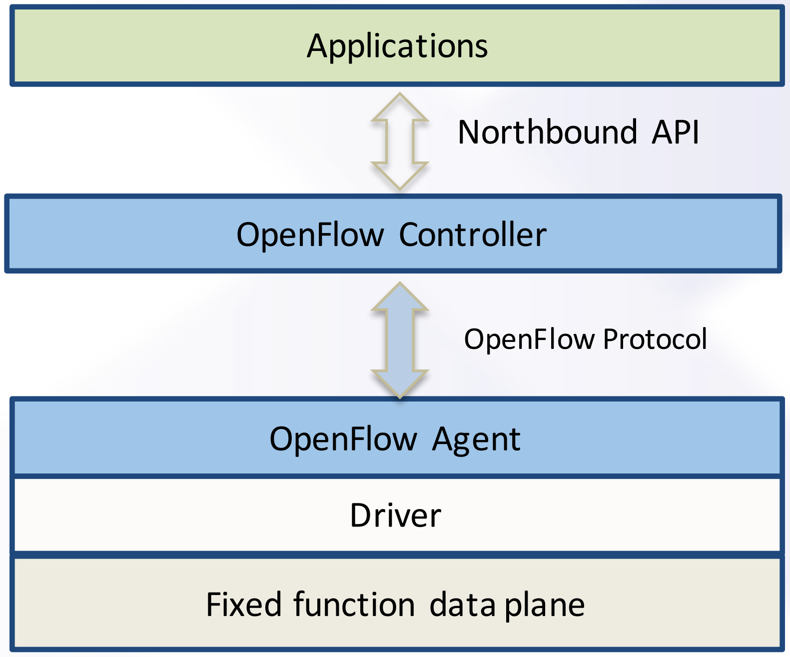


P4 program defines what each table CAN do

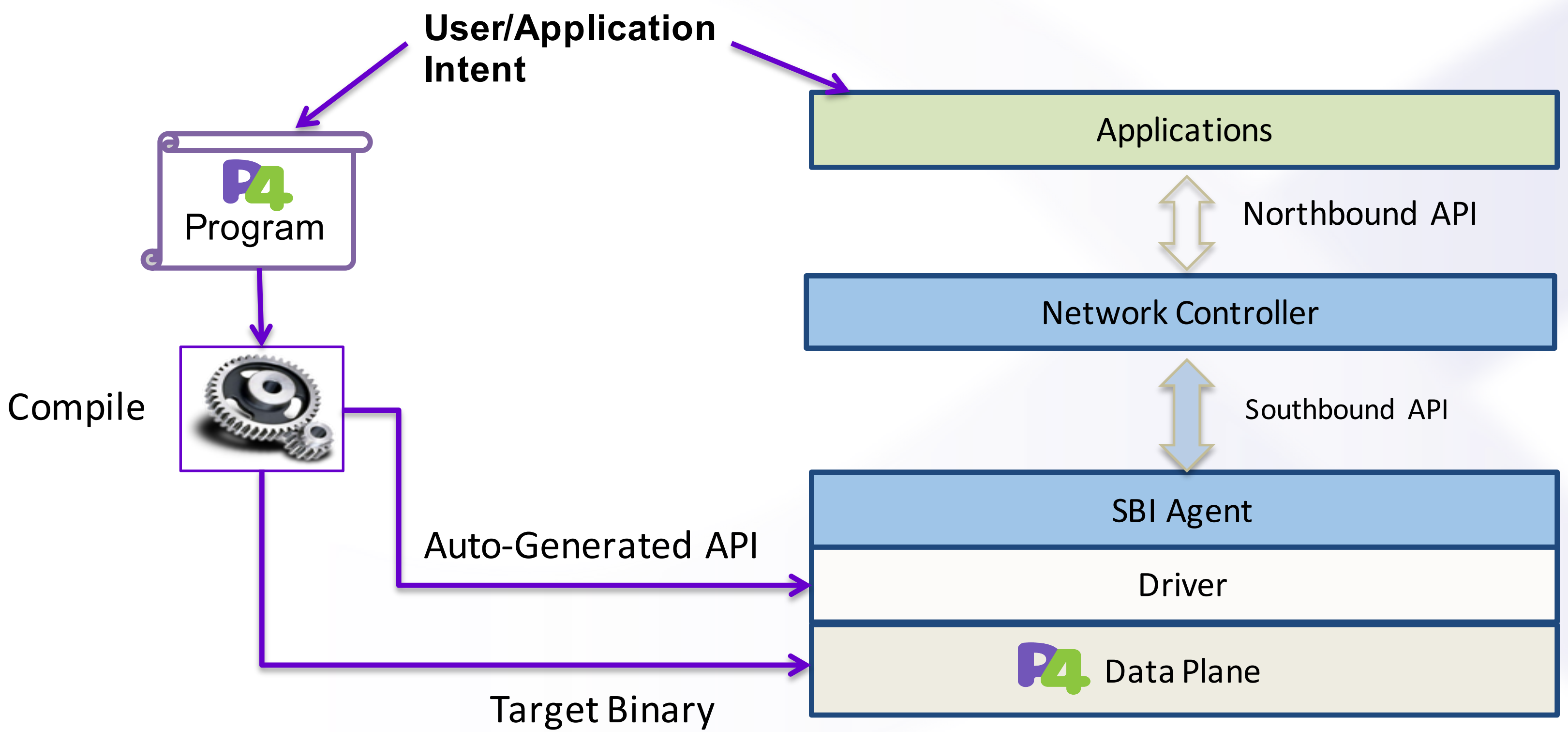
Control Plane Roles



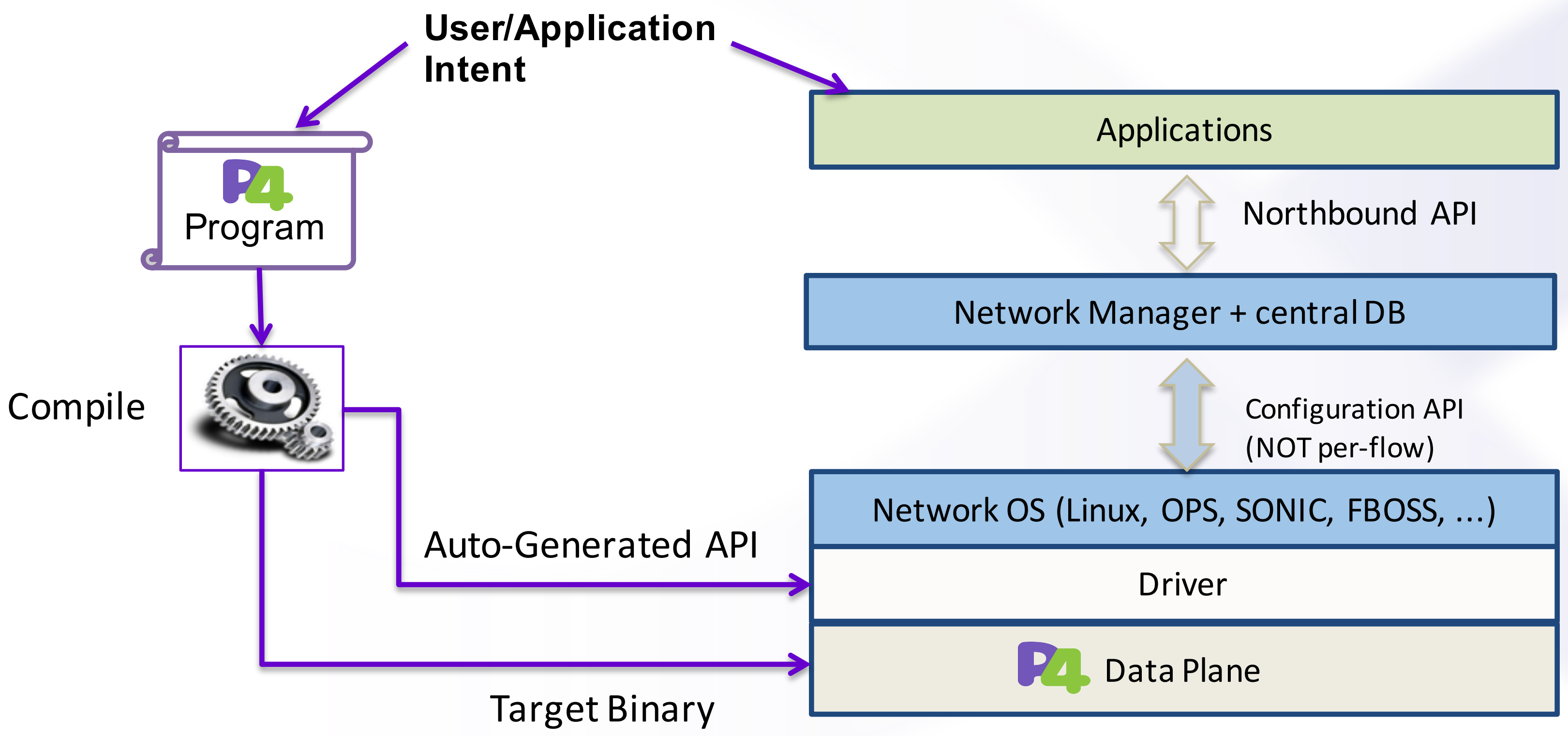
P4 & OpenFlow: Traditional SDN before P4



P4 with OpenFlow



P4 with Network OS



P4 Constructs

- **P4 Spec v1.0.2+, v1.1.0-**

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

Headers and Fields (Packet)

Example: 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)

Example: 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 Instantiation

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

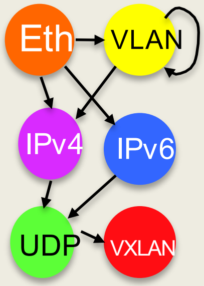
Defining a Parser Tree

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;
    }
}
```

Transitions to the next parser states. Prioritized by order

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

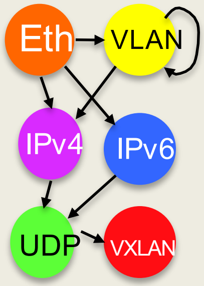


Defining a Parser Tree (cont.)

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;
}
```



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;
}

parser_exception p4_pe_checksum {
    return parser_drop;
}
```

Predefined parser state

Multi-field select statement

Example: 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) {
        0x0000501 : parse_icmp;
        0x0000506 : parse_tcp;
        0x0000511 : parse_udp;
        default   : ingress;
}
```

Metadata can be initialized by the parser

Fields are joined for a match

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

  - When the device only need to insert a header but shouldn't actually parse it

- **Example: insert my_header after udp**

```
parser parse_udp {
     extract(udp);
     return select(latest.dst_port) {
         0x0 mask 0x00 : ingress;
         default       : parse_my_header;
}
```

Ingress parser will always transit to ingress

Parser tree has a branch to my_header for deparsing

P4 Language Components

- Data declarations
- Parser Programming

- **Packet Flow Programming**
  - Actions
    - Primitive and compound actions
    - Counters, Meters, Registers
  - Tables
    - Match keys
    - Attributes
  - Control Functions

Actions

- **Primitive actions**
  - no_op, drop
  - modify_field, modify_field_with_hash_based_offset
  - add, add_to_field
  - 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);
    add_to_field(ipv4.ttl, -1);
}
```

Arithmetic and Logical Primitives

- **The current standard (v1.0.2)**
  - Primitive actions
    - Standard: add(), add_to_field()
    - Additional: subtract(), subtract_from_field(), bit_and(), bit_or(), bit_xor(), shift_left(), shift_right(), …
    - `add_to_field(ipv4.ttl, -1)`
  - Partial support for expressions exists in some compilers
- **Developing standard (p4-16)**
  - Expressions with +, -, &, |, ^, ~, <<, >>, etc.
    - `modify_field(ipv4.ttl, ipv4.ttl - 1)`
  - Specific targets might restrict expression complexity

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);
}
```

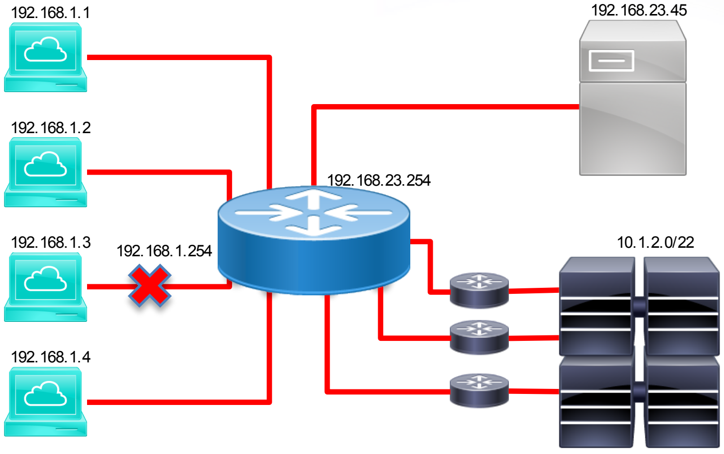
	Sequential Semantics	Parallel Semantics
fieldA	1	1
fieldB	1	fieldA before action

- **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**
- **P4 defines**
  - What to match on and match type
  - A list of *possible* actions
  - Additional attributes
    - Size
- **In runtime, 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

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=

- **P4 Program**
  - Defines the format of the table
    - Key Fields
    - Actions
    - Action Data
- **Control Plane (IP stack, Routing protocols)**
  - Populates table entries with specific information
    - Based on the configuration
    - Based on automatic discovery
    - Based on protocol calculations
- **Data Plane (populated table)**
  - Performs the lookup
  - Executes the chosen action

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)

Example: A typical L3 (IPv4) Host table

```
table ipv4_host {
    reads {
        ingress_metadata.vrf    : exact;
        ipv4.dstAddr            : exact;
    }
    actions {
        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

Match-Action Table (Longest Prefix Match)

Example: A typical L3 (IPv4) Routing table

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

Different fields can use different match types

Prefix also serves as a priority indicator

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

Match-Action Table (Ternary Match)

Example: A typical L3 (IPv4) Routing table

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

Ternary tables require an explicit specification of entry priority

Prio	vrf / mask	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 Types

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

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
- **When no matching entries are found**
  - Default Action with the default action data is executed **if** it has been set by the control path
  - If no Default Action has been specified, it is **no_op**()

Stateful Objects

- Counters, Meters, Registers

What are stateful objects

- **Stateful objects keep their state between packets**
  - Metadata and packet headers are **stateless**
    - They are re-initialized for each packet
  - Counters, Meters and Registers are **stateful**
    - Counters are incremented with each packet
    - Meters keep their bucket state
    - Registers store arbitrary data

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

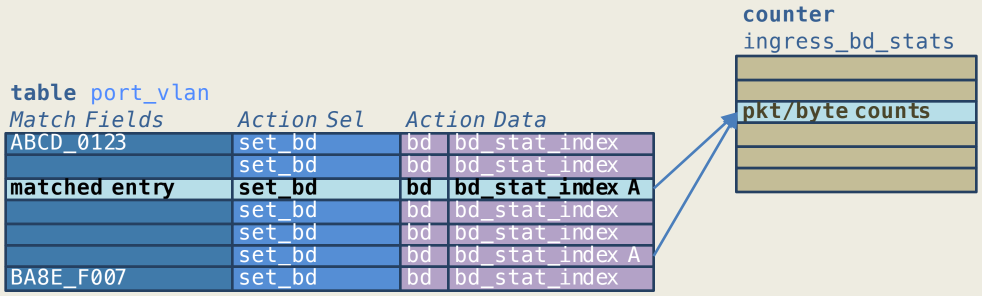
Indirect Counters

Flexibly linked counters

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

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



Meters

- **Declaration is similar to counters**
  - Action: execute_meter(meter_array, meter_index, color_destination)

```
/* Direct Meter */
meter acl_meter {
     type:   packets;
     direct: ip_acl;
     result: metadata.color;
}

/* Indirect Meter Array */
meter bd_meter {
    type: bytes;
    instance_count: 1000;
}

action do_bd_meter(meter_index) {
    execute_meter(bd_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
    - register_read(register_array, register_index, destination_field)
    - register_write(register_array, register_index, value)

```
/* Register Array (Indirect) */
register last_syn {
    width: 32;
    static: flow_table;
    instance_count:  1024;
}

action get_flow_age(flow_index) {
    register_read(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) {
    register_write(last_syn, flow_index, metadata.ingress_global_timestamp);
}
```

Control Flow Functions

- **Primitives**
  - Perform a table lookup: **apply**
  - **if/else** statement
  - **apply** with the case clause
- **Sequential Execution Semantics**
  - Compiler is doing parallelization automatically
- **Standard control functions**
  - ingress() – Ingress Pipeline processing
  - egress() – Egress Pipeline processing
- **User-defined control functions**

Standard Control Functions

- **ingress() control function starts processing**
  - remember “return ingress;” statement in the parser functions
- **egress() control function is called implicitly from the Packet Replication Engine**

```
control ingress {
    apply(port_vlan_mapping);
    apply(smac);
    apply(dmac);
}

control egress {
    apply(vlan_tag_removal);
}
```

User-Defined Control Functions

- **Help improve code readability**
  - No specific performance advantages: the code is flattened by the compiler
- **No parameters are accepted**

```
control assign_vlan {
    apply(subnet_vlan);
    apply(mac_vlan);
    apply(protocol_vlan);
    apply(port_vlan);
    apply(resolve_vlan);
}

control ingress {
    . . .
    if (!valid(vlan_tag[0]) {
        assign_vlan();
    }
    . . .
}
```

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)) {
    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)) {
        process_ipv6_racl();
        process_ipv6_urpf();
        process_ipv6_fib();
    }
}
```

Action Branching

Example: 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. */
    }
}
```

Miss branching

```
action on_miss() {}

table ipv4_fib {
   reads {
   . . .
   }
   actions {
       l3_switch;
       l3_l2_switch;
       l3_switch_nexthop;
       l3_switch_ecmp;
       on_miss;
   }
}

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

We choose to use only this action as the default

Executing actions in the control flow

```
control process_counters {
    if (my_meta.drop_packet == 0) {
        count(bd_counter,
              metadata.bd_counter_index);
        count(vrf_counter,
              metadata.vrf_counter_index);
    }
}
```

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

table do_process_counters {
    actions {
        update_counters;
    }
}

control process_counters {
    if (my_meta.drop_packet == 0) {
        apply(do_process_counters);
    }
}
```

This action must be set as a default for this table by the control plane

- **Actions cannot be directly referenced in the control flow functions**
  - Instead, they need to be “wrapped” into tables
- **Tables without keys can be used to implement unconditional execution**
  - They always miss and hence the desired action needs to be set as a default

Advanced Concepts

- Action Profiles
- Packet Digests
- Packet Resubmit/Recirculation
- Packet Cloning

Action Profiles

- **Separate table match entries from actions and action data**
- **Allow multiple entries to share same action data**
  - Saves space
  - Allows quick update of multiple entries
- **Allow multiple actions/action_data per entry**
  - This is called “dynamic action selection”
  - Used to implement LAG or ECMP
- **Can be more efficient compared to explicit implementation**

Action Profiles

Actions can be complex

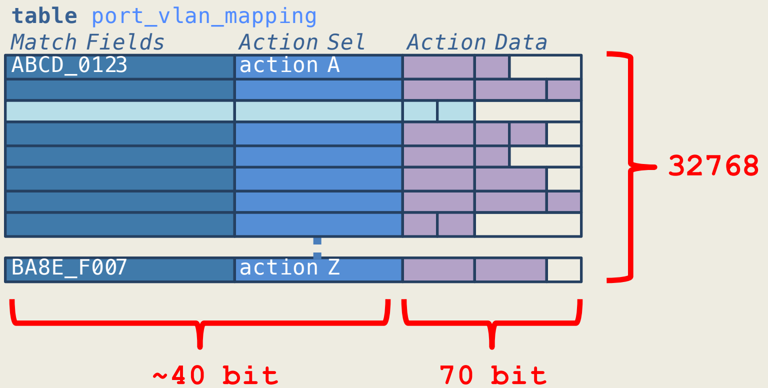
```
action set_bd(bd, vrf, rmac_group,
        ipv4_unicast_enabled, ipv6_unicast_enabled,
        ipv4_urpf_mode, ipv6_urpf_mode,
        igmp_snooping_enabled, mld_snooping_enabled,
        bd_label, stp_group, stats_idx,
        exclusion_id)
{
    modify_field(l3_metadata.vrf, vrf);
    modify_field(ipv4_metadata.ipv4_unicast_enabled,       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, many entries do the same action – the table uses too much space

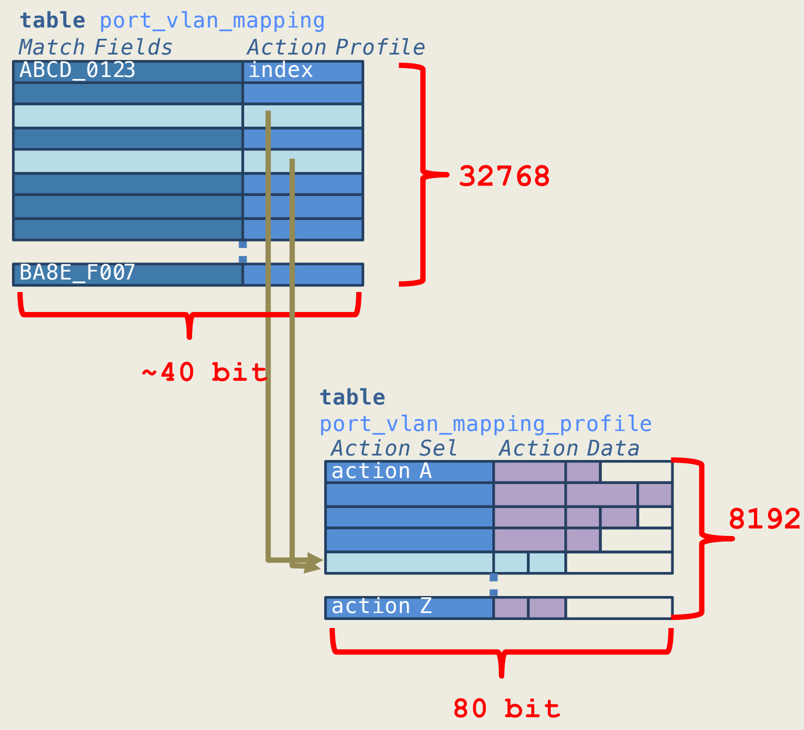
```
table port_vlan_mapping {
    reads {
        ingress_metadata.ifindex : exact;
        vlan_tag[0]              : valid;
        vlan_tag[0].vid          : exact;
        vlan_tag[1]              : valid;
        vlan_tag[1].vid          : exact;
    }
    actions {
        set_bd;
        set_bd_ipv4_mcast_switch_ipv6_mcast_switch_flags;
        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;
}
```



Manual Profile Implementation

Share the same action data bits with multiple entries

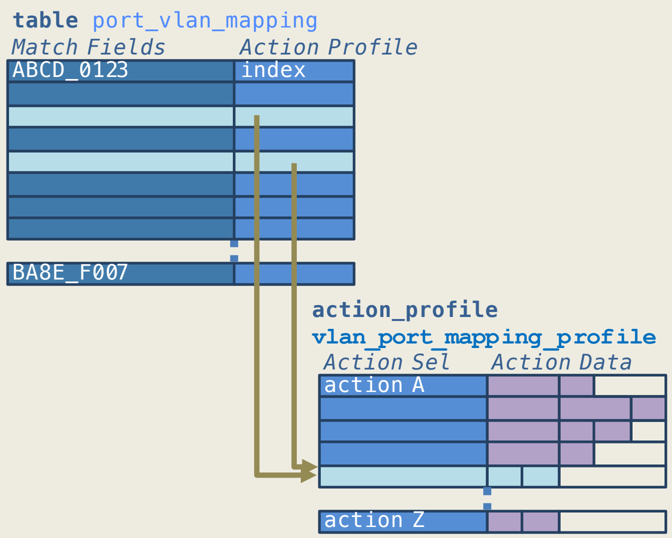
```
table port_vlan_mapping_profile {
    reads {
        metadata.profile_index : exact;
    actions {
        set_bd;
        set_bd_ipv4_mcast_switch_ipv6_mcast_switch_flags;
        set_bd_ipv4_mcast_switch_ipv6_mcast_route_flags;
        set_bd_ipv4_mcast_route_ipv6_mcast_switch_flags;
        set_bd_ipv4_mcast_route_ipv6_mcast_route_flags;
    }
    size : 8192;
}

action set_profile_index(index) {
    modify_field(metadata.profile_index,
                 index);
}

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_profile_index;
    }
    size : 32768;
}
```



Using P4 profiles

Share the same action with multiple entries. Action profile can be shared by <u>multiple tables</u> too.

```
action_profile vlan_port_mapping_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 : vlan_port_mapping_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;
}
```

Packet Digests – Notify control plane of data plane events

- **Action**
  - generate_digest(receiver, field_list)
- **Sends the specified metadata to a target-specific receiver**
  - Software-based
  - Hardware-Based

Implementing MAC Learning

Source MAC Lookup

```
action smac_hit(src_port, static) {
    modify_field(l2_metadata.src_port, src_port);
    modify_field(l2_metadata.is_static, static);
    modify_field(l2_metadata.src_move,
                 standard_metadata.igress_port ^ src_port);
}

action smac_miss() {
    modify_field(l2_metadata.src_miss, TRUE);
}

table smac {
    reads {
        vlan_tag[0].vid : exact;
        ethernet.srcAddr: exact;
    }
    actions {
       smac_hit;
       smac_miss;
    }
}
```

Learn Notification Generation

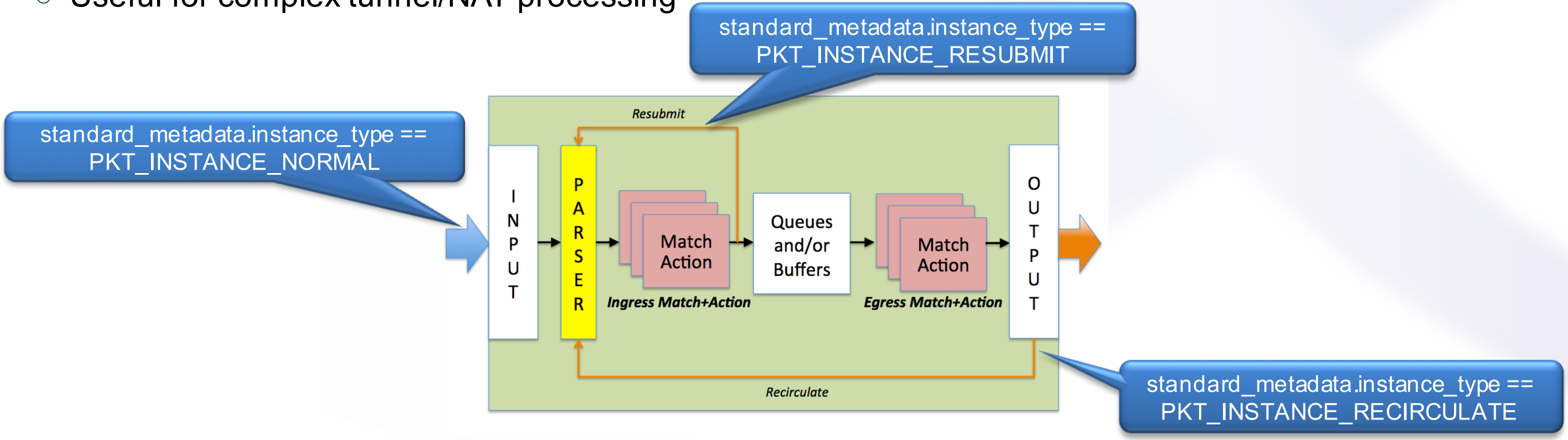
```
field_list l2_learn_digest {
    ethernet.srcAddr,
    vlan_tag[0].vid,
    standard_metadata.ingress_port,
    l2_metadata.src_port,
    l2_metadata.src_move
}

action send_learn_notification() {
    generate_digest(L2_RECV, l2_learn_digest);
}

table learn_notification {
    reads {
        l2_metadata.src_move : exact;
        l2_metadata.src_miss : exact;
        l2_metadata.is_static: exact;
    }
    actions {
       send_learn_notification;
    }
}

control process_l2_learning {
    apply(smac);
    apply(learn_notification);
}
```

Target-specific number. Can be SW or HW receiver

Other fields can influence sending of learn notifications

Resubmit and Recirculate

- **Implementing complex processing**
- **resubmit(field_list)**
  - **Unmodified** packet is returned back to the parser within ingress pipeline, with additional metadata
  - Useful for complex parsing
- **recirculate(field_list)**
  - **Fully modified packet** is returned back to the ingress parser after completing all the processing
  - Useful for complex tunnel/NAT processing



MPLS Processing (Simple)

```
parser parse_mpls {
    extract(mpls[next]);
    return select(latest.bos) {
        0       : parse_mpls;
        1       : parse_mpls_bos;
        default : ingress;
    }
}

parser parse_mpls_bos {
    return select(current(0, 4)) {
        0x4     : parse_inner_ipv4;
        0x6     : parse_inner_ipv6;
        default: parse_inner_ethernet;
    }
}
```

- **This parser is a very effective hack**
  - MPLS does not formally carry any indication of the payload type
- **The right way to do parsing is by using label lookup**

MPLS Processing (Full)

Parser

```
#define  IPV4      0
#define  IPV6      1
#define  ETHERNET  2

header_type  my_metadata  {
    fields  {
        mpls_payload  : 2;
    }
}

parser  parse_mpls_bos  {
    return  select(
                standard_metadata.resubmit_flag,
                my_metadata.mpls_payload,
                current(0,  4))  {
        0x04   : parse_inner_ipv4;
        0x06   : parse_inner_ipv6;
        0x40  mask  0x70  : parse_inner_ipv4
        0x50  mask  0x70  : parse_inner_ipv6;
        0x60  mask  0x70  : parse_inner_ethernet;
        default:  parse_inner_ethernet;
    }
}
```

Control Flow

```
action set_mpls_payload_type(mpls_payload) {
    modify_field(my_metadata.mpls_payload, mpls_payload);
}

table mpls {
    reads {
        mpls[0].label : exact;
        mpls[1].label : exact;
    }
    actions {
        set_mpls_payload_type;
    }
}

control ingress {
    . . .
    apply(mpls);
    /* If our guess was wrong... */
    if (my_metadata.mpls_payload == ETHERNET and !valid(inner_ethernet)) {
        apply(mpls_resubmit);
    }
    . . .
}
```

MPLS Processing (Resubmit)

```
field_list mpls_resubmit_list {
    my_metadata.mpls_payload
}

action do_mpls_resubmit() {
    resubmit(mpls_resubmit_list);
}

table mpls_resubmit {
    actions {
        do_mpls_resubmit;
    }
}
```

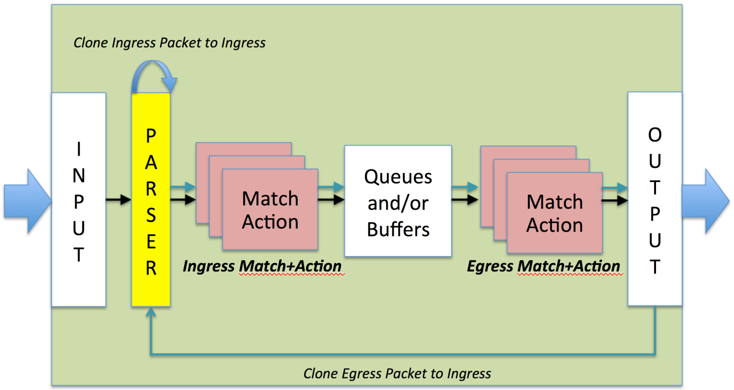
- **The packet still goes through the rest of the ingress pipeline**
    - Use the if() statement around the rest of the code to avoid further processing
- **All metadata computed at the first pass will be lost**
    - Put all the fields that will be needed in the second pass into the resubmit field_list

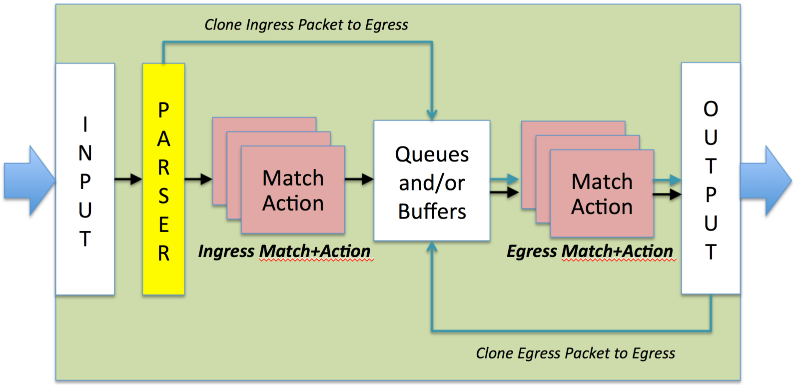
Cloning

- **A new copy of a packet is created**
- **The original and the clone are processed independently**
- **Two cloning sources**
  - Original packet (before ingress)
  - Fully processed (after egress)
- **Two cloning destinations**
  - Beginning of the ingress pipeline
  - Packet Queuing/Replication Engine
    - Before the beginning of the egress pipeline

Source / Dest	Packet before Ingress	Packet After Egress
Ingress Pipeline	clone_i2i()	clone_e2i()
Egress Pipeline	clone_i2e()	clone_e2e()

clone_x2y(clone_spec, field_list)





Implementing Negative Mirroring (aka mirror on drop)

Creating a mirrored copy

```
field_list mirror_info {
    i2e_metadata.mirror_session_id;
    i2e_metadata.mirror_drop_flags;
}

action do_negative_mirror(session_id) {
    modify_field(i2e_metadata.mirror_session_id, session_id);
    modify_field(i2e_metadata.mirror_drop_flags,
                 ingress_metadata.drop_flags);
    clone_i2e(session_id, mirror_info);
}

table negative_mirror {
    actions {
        do_negative_mirror;
    }
}

control ingress {
    . . .
    if (ingress_metadata.drop_flags != 0) {
        apply(negative_mirror);
    }
}
```

Processing the mirrored copy

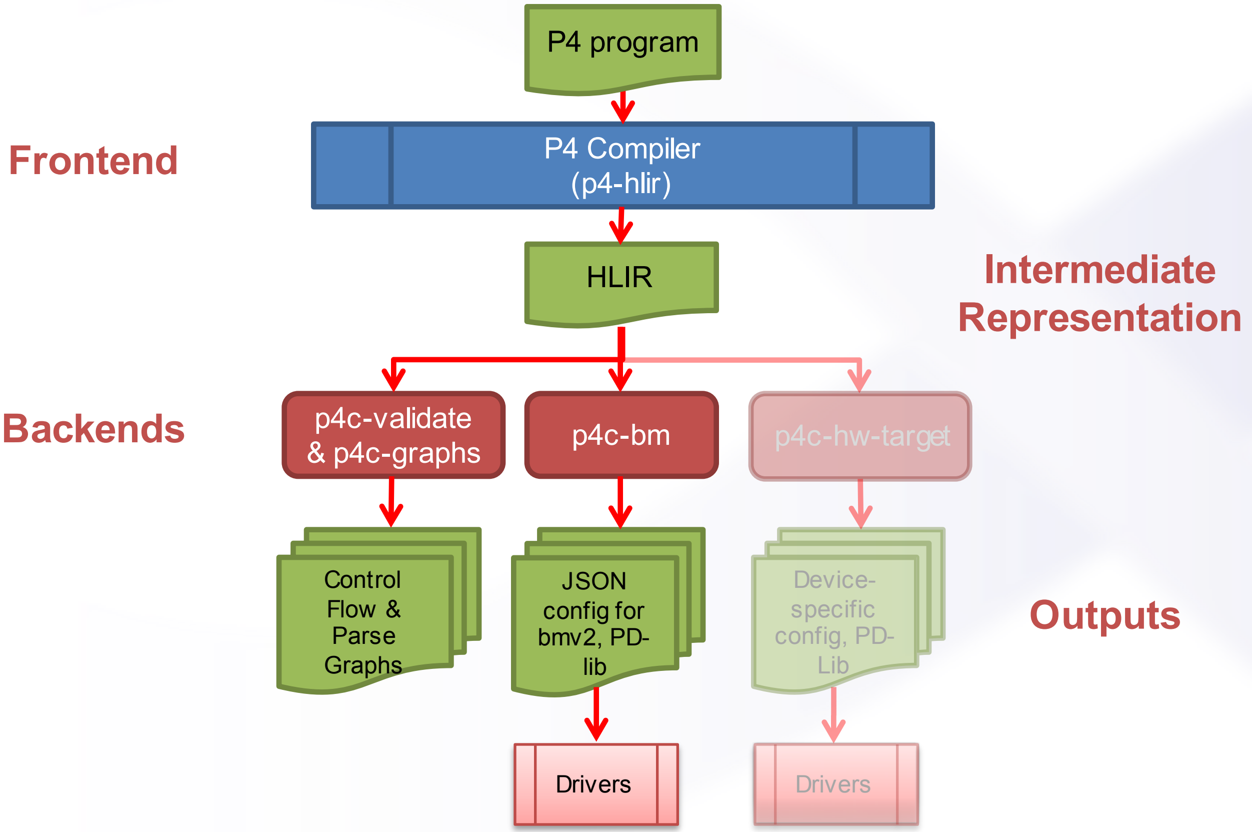
```
action set_mirror_nhop(nhop_idx) {
    modify_field(l3_metadata.nexthop_index, nhop_idx);
}

action set_mirror_bd(bd) {
    modify_field(egress_metadata.bd, bd);
}

table mirror {
    reads {
        i2e_metadata.mirror_session_id : exact;
    }
    actions {
        nop;
        set_mirror_nhop;
        set_mirror_bd;
    }
    size : MIRROR_SESSIONS_TABLE_SIZE;
}

control egress {
    if (standard_metadata.instance_type == PKT_INSTANCE_TYPE_INGRESS_CLONE) {
        apply(mirror);
    else {
        . . .
    }
}
```

P4 Compiler Overview

Modular Compiler Overview

- **Single Front-End (p4-hlir)**
  - Translates P4 code into 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
    - HLIR documentation is supplied with the frontend code
- **Multiple backends**
  - Code generators for various targets
    - Software Switch Model (p4c-bm)
    - Network Interface Cards
    - Packet Processors / NPUs
    - FPGAs, GPUs, ASICs
  - Validators and graph generators
  - Run-time API generators

P4 Modular Compiler



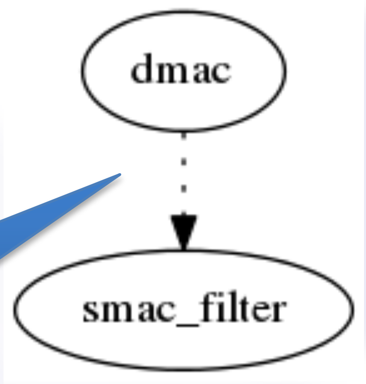
Dependency Analysis

Types of dependencies

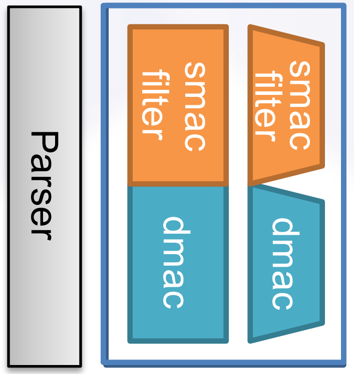
- Dependencies are inferred from target-independent P4 program analysis
- Independent tables
- Match Dependency
- Action Dependency
- Successor Dependency
- Reverse Read Dependency

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);
}
```

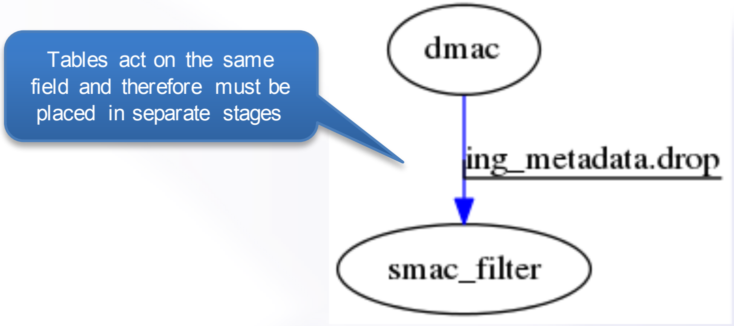


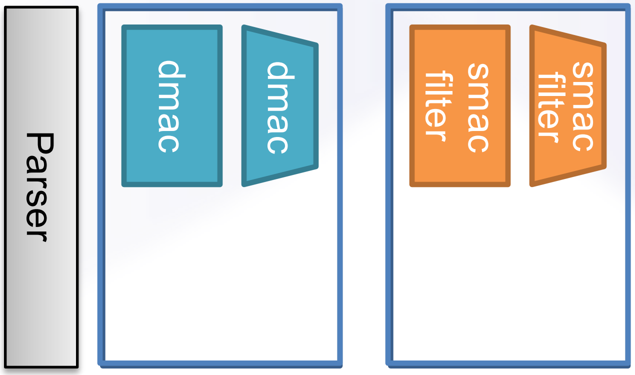
Tables are independent: both matching and action execution can be done in parallel



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;
    }
}

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

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

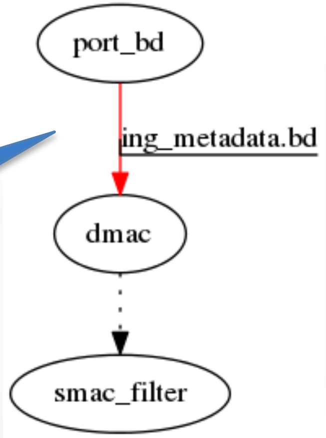


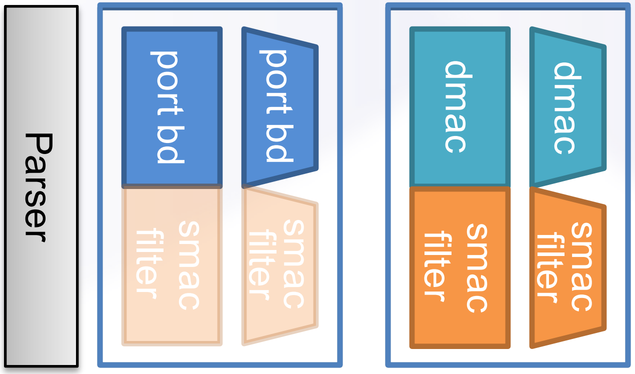


Match Dependency

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

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

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

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

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

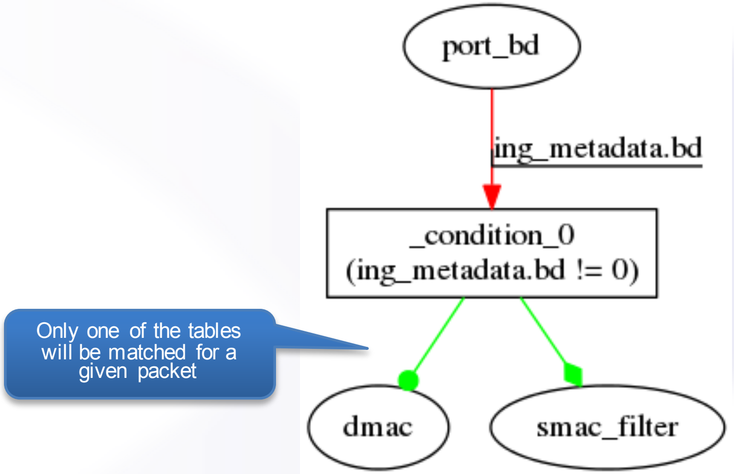
The second table matches on the field, modified by the first.

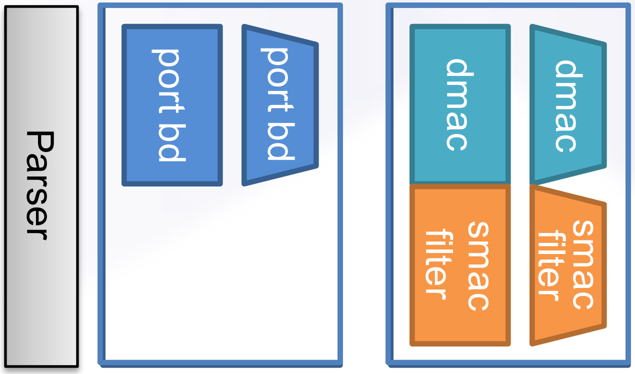




Successor Dependency

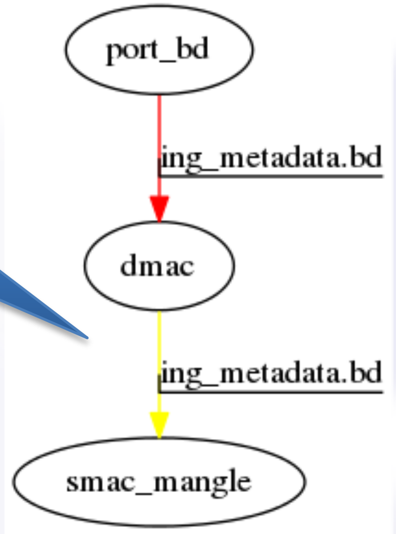
```
action set_bd(bd) {
    modify_field(ing_metadata.bd, bd);
}

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

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

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);
    }
}
```



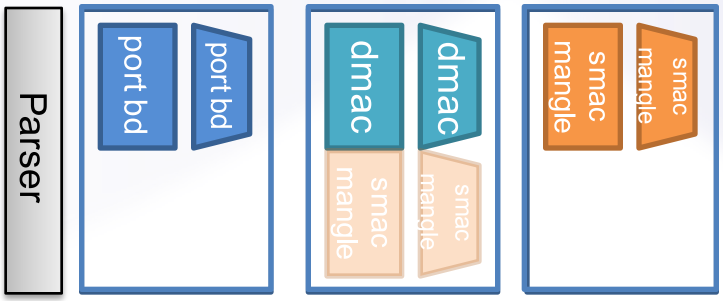


Reverse Read Dependency

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

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

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

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

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



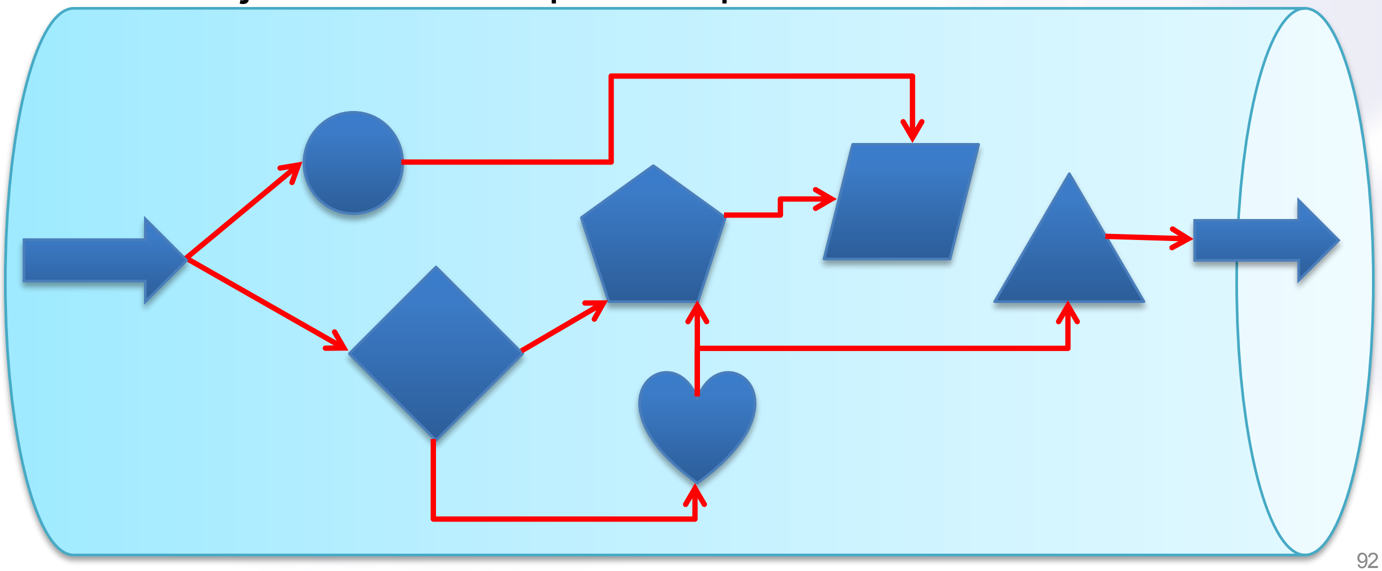
The third table modifies a field used by the second table for matching. Therefore the action cannot take place before the second table matches.



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

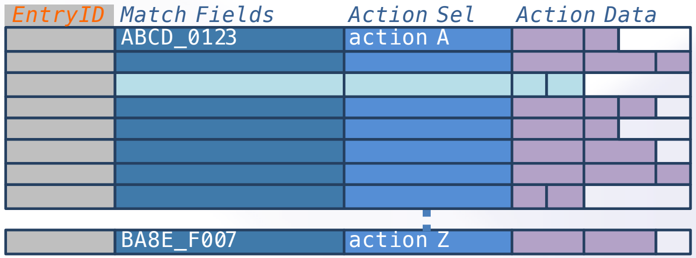


P4 is an Ideal Base for a Network APIs

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

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);
```

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);
```

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

github.com/p4lang

- **switch.p4: reference p4 program**
- **BMv2 (Behavioral Model v2): s/w switch runs p4**
- **BMv2 compiler**
- **supporting tools, scripts**

- **Project summaries: [link]**

BMv2 Primitives

- standard primitives https://github.com/p4lang/p4-hlir/blob/master/p4_hlir/frontend/primitives.json
- bmv2 specific primitives https://github.com/p4lang/p4c-bm/blob/master/p4c_bm/primitives.json

Intrinsic Metadata, provided by BMv2 switch

- If one defines all these fields, all the simple_switch features will be supported, so it is recommended to define these fields in every program (to avoid a headache).

```
header_type intrinsic_metadata_t {
    fields {
        ingress_global_timestamp : 48;  // ingress timestamp, in microseconds
        mcast_grp : 4;  // to be set in the ingress pipeline for multicast
        egress_rid : 4;  // replication id, available on egress if packet was multicast
        mcast_hash : 16;  // unused
        lf_field_list : 32;  // set by generate_digest primitive, not to be used directly by P4 programmer
        resubmit_flag : 16;  // used internally
        recirculate_flag : 16;  // used internally
    }
}

metadata intrinsic_metadata_t intrinsic_metadata;

header_type queueing_metadata_t {
    fields {
        enq_timestamp : 48;  // in microseconds
        enq_qdepth : 16;
        deq_timedelta : 32;
        deq_qdepth : 16;
    }
}

metadata queueing_metadata_t queueing_metadata;
```

Thank you