P4 Language And Art of Data Plane Programming

Paul Ho
Arista Network

Agenda

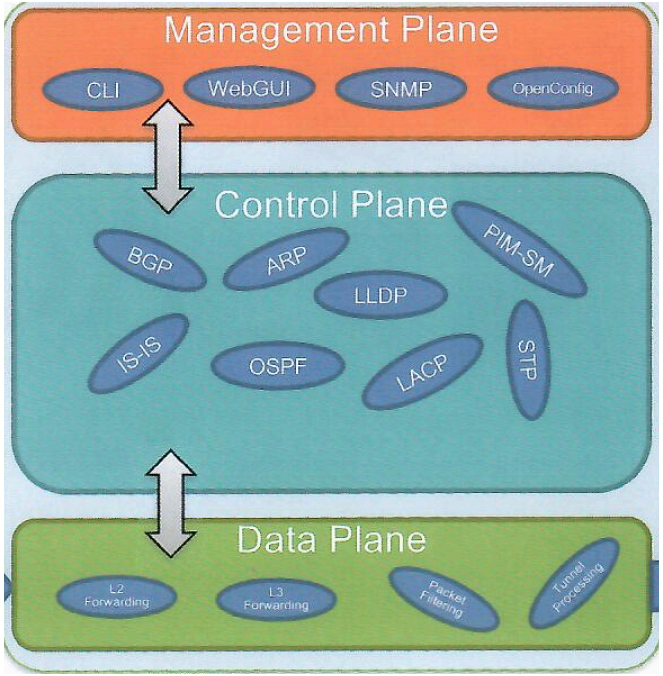
- What is Data Plane Programming?
- Introduction to P4
- P4 Example and Demo

What is Data Plane Programming?

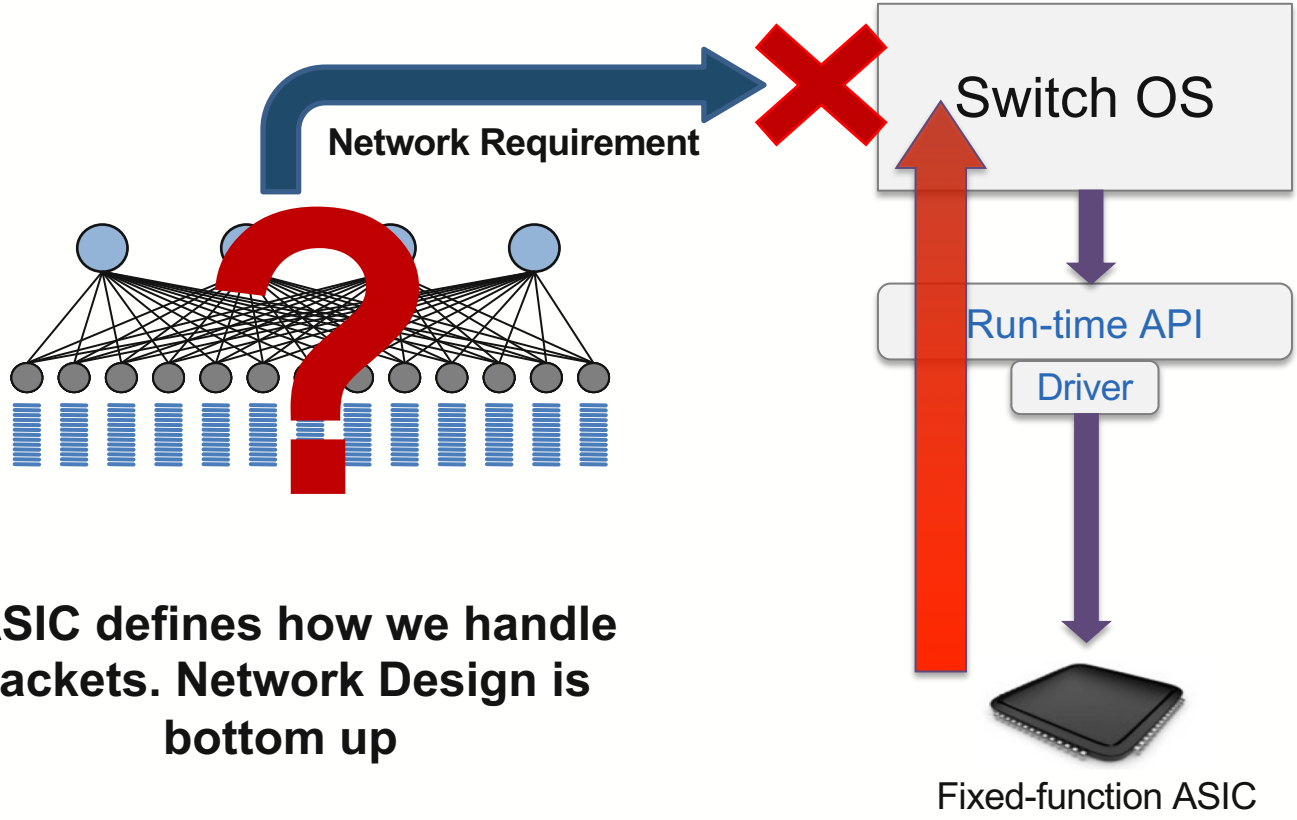
- **Why program the Data Plane?**

Standard Telecommunications Architecture

- The Three Planes
- Planes are Classes of Algorithms
  - Data Plane : Packet Forwarding
  - Control Plane : Routing, Discovery ...
    - Controlling the Data Plane
  - Management Plane : CLI, GUI
    - Managing the Control Plane (and the system)
- Different requirements
- Different design
- Different languages
- Different hardware
- Well-defined boundaries and interfaces

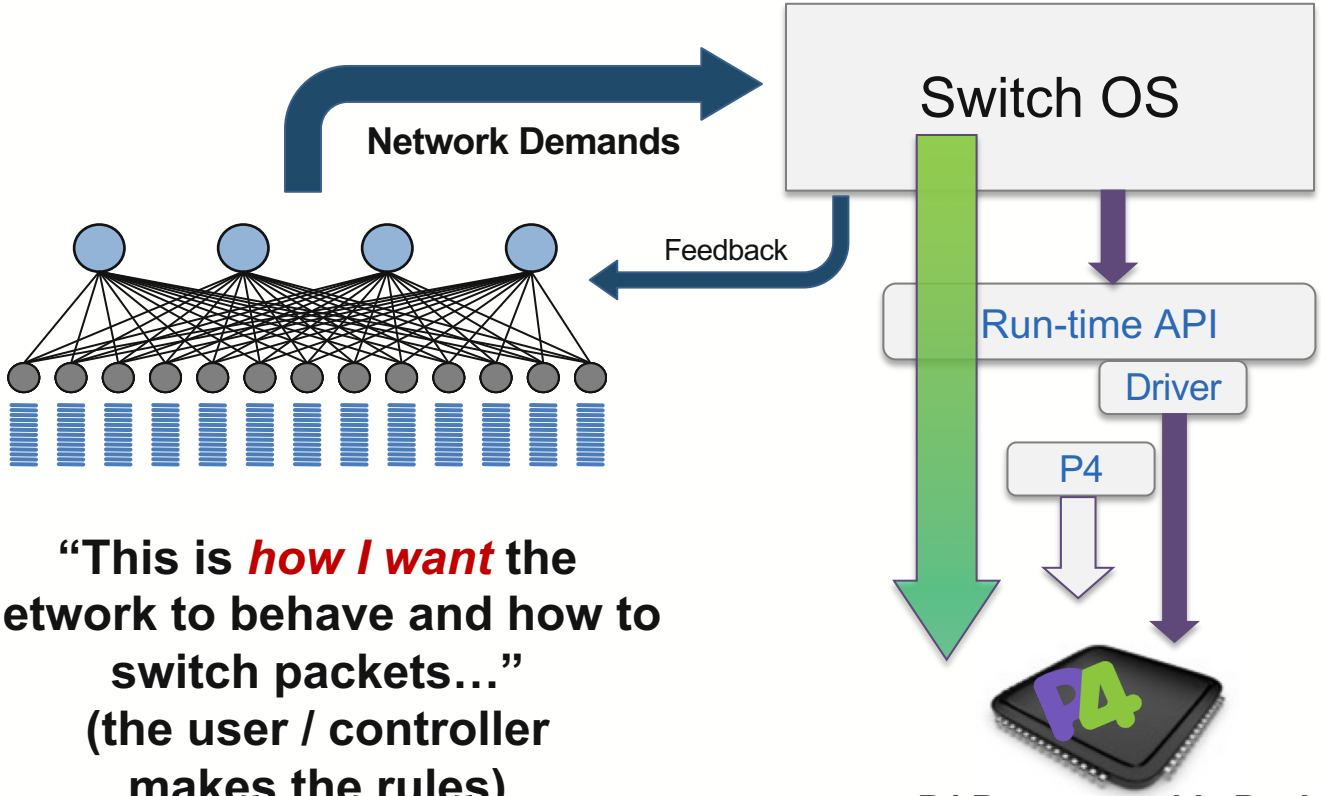


Status Quo:



“ASIC defines how we handle packets. Network Design is bottom up

A Better Approach



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

P4 Design Goals

- **Data Plane Programming Language**
  - Simple Packet Transformations and External Component Interfaces
- **Protocol Independence**
  - Programmer defines packet formats and processing algorithms
- **Target Independence**
  - No knowledge of low-level hardware organization is required
  - Compiler compiles the program for the target device
- **Consistent Control Plane Interface**
  - Control Plane APIs are automatically generated by the compiler
- **Reconfigurability**
  - Data Plane program can be changed in the field
- **Community-driven Design**
  - http://p4.org

Benefits of Data Plane Programmability

- **Customized** – Add new protocols/new features easily
- **Reduce complexity** – Remove unused protocols
- **Efficient use of resources** – flexible use of tables
- **Greater visibility** – New diagnostic techniques, telemetry, etc.
- **SW style development** – rapid design cycle, fast innovation, fix data plane bugs in the field
- **You keep your own ideas**

Brief History and Trivia

- May 2013: Initial idea and the name “P4”
- July 2014: First paper (SIGCOMM CCR)
- Aug 2014: First $P4_{14}$ Draft Specification (v0.9.8)
- Sep 2014: $P4_{14}$ Specification released (v1.0.0)
- Jan 2015: $P4_{14}$ v1.0.1
- Mar 2015: $P4_{14}$ v1.0.2
- Nov 2016: $P4_{14}$ v1.0.3
- May 2017: $P4_{14}$ v1.0.4

- Apr 2016: $P4_{16}$ – first commits
- Dec 2016: First $P4_{16}$ Draft Specification
- May 2017: $P4_{16}$ Specification released

P4.org Membership

Operators/ End Users

Alibaba Group · Baidu 百度 · FOX · Goldman Sachs · Microsoft · Tencent 腾讯 · at&t · COMCAST · Google · kt · SK telecom

Systems

ARISTA · BROCADE · cisco · DANTEL · CORSA · DELL · Hewlett Packard Enterprise · HUAWEI · Inventec · INTRACOM TELECOM · juniper · mantis · NETSERO · NetBRIC · NoviFlow · ZTE · OpenNet · XENTECH

Targets

AEPONYX · Atomic Rules · BAREFOOT NETWORKS · CAVIUM · EZchip · intel · NETCOPE TECHNOLOGIES · Mellanox · BROADCOM · centec networks · freescale · Innovium · MoSys · NETRONOME · PLUMgrid · vmware · MARVELL · STORDIS · XILINX

Solutions/ Services

EstiNet · happiest minds · HGLOBAL HITECH · SDNLAB · XFLOW RESEARCH

Academia/ Research

Bii · KOREA UNIVERSITY · PRINCETON UNIVERSITY · salzburgresearch · Eötvös Loránd University · POLITECNICO MILANO 1863 · Stanford University · UFRGS · VirginiaTech Invent the Future

- 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_14 and P4_16

- **P4_16 is a Logical, Long-Term Evolution of P4_14**
  - Same basic building blocks and concepts
  - More formally defined semantics
  - Explicit Language/Architecture separation
  - More convenient to use
  - More difficult to use
- **The fundamentals are still the same**
  - The power of the languages is exactly the same
  - All target can execute programs written in either P4_14 or P4_16
  - The new compiler (p4c) can perform automatic P4_14_to_P4_16 conversion

P4_14 and P4_16

- **P4_16 is a Logical, Long-Term Evolution of P4_14**
  - Same basic building blocks and concepts
  - More formally defined semantics
  - Explicit Language/Architecture separation
  - More convenient to use
  - More difficult to use

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, …

What can you do with P4?

- Layer 4 Load Balancer – SilkRoad[1]
- Low Latency Congestion Control – NDP[2]
- In-band Network Telemetry – INT[3]
- In-Network caching and coordination – NetCache[4] / NetChain[5]
- Aggregation for MapReduce Applications [7]
- … and much more

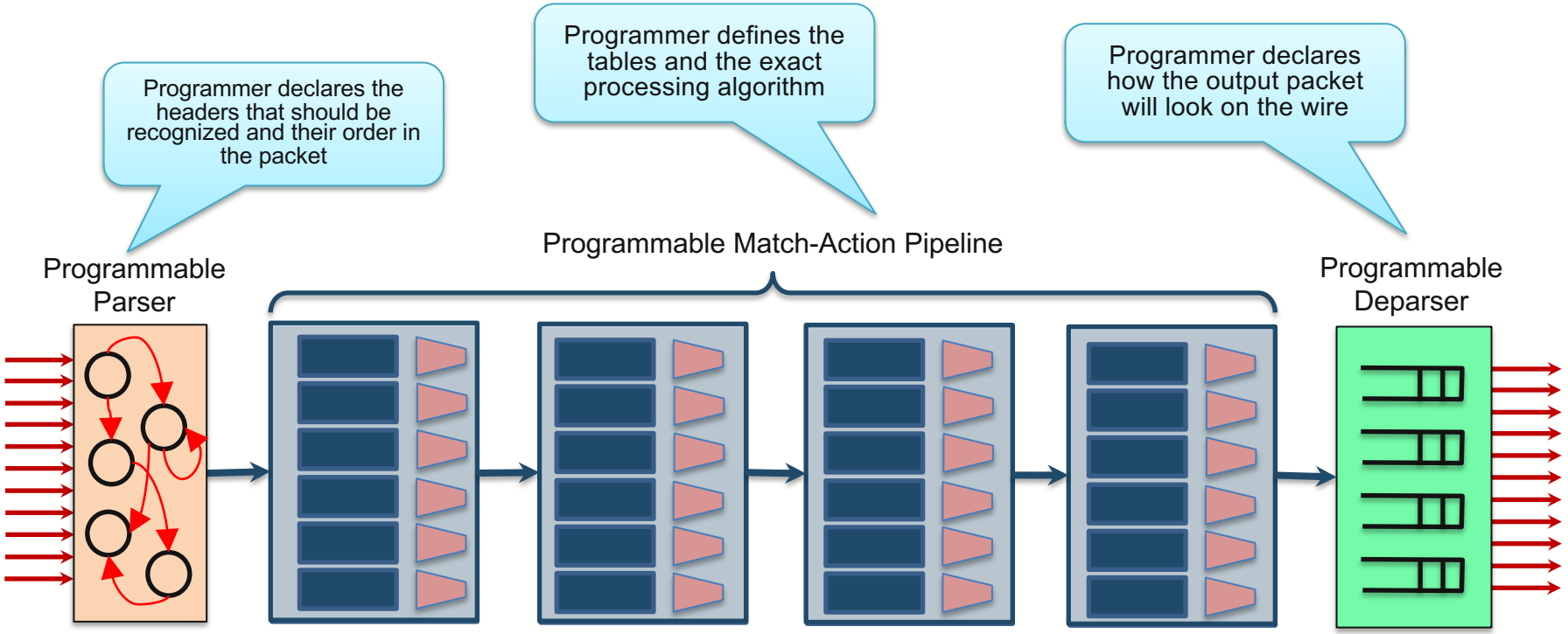
[1] Miao, Rui, et al. "SilkRoad: Making Stateful Layer-4 Load Balancing Fast and Cheap Using Switching ASICs." SIGCOMM, 2017.
[2] Handley, Mark, et al. "Re-architecting datacenter networks and stacks for low latency and high performance." SIGCOMM, 2017.
[3] Kim, Changhoon, et al. "In-band network telemetry via programmable dataplanes." SIGCOMM. 2015.
[4] Xin Jin et al. "NetCache: Balancing Key-Value Stores with Fast In-Network Caching." To appear at SOSP 2017
[5] Jin, Xin, et al. "NetChain: Scale-Free Sub-RTT Coordination." *NSDI*, 2018.
[6] Dang, Huynh Tu, et al. "NetPaxos: Consensus at network speed." SIGCOMM, 2015.
[7] Sapio, Amedeo, et al. "In-Network Computation is a Dumb Idea Whose Time Has Come." *Hot Topics in Networks*. ACM, 2017.

Introduction to P4

(**P**)rogramming (**P**)rotocol-Independent (**P**)acket (**P**)rocessors

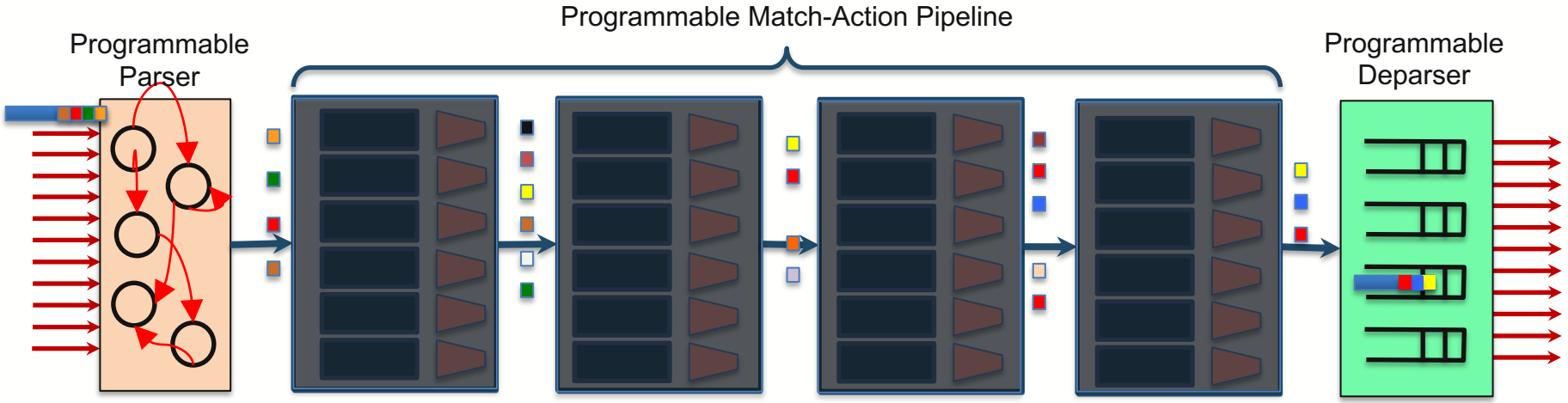
ARISTA

PISA: Protocol-Independent Switch Architecture



PISA in Action

- Packet is parsed into individual headers (parsed representation)
- Headers and intermediate results can be used for matching and actions
- Headers can be modified, added or removed
- Packet is deparsed (serialized)



P4$_{14}$ Program Skeleton

```
/*********************  H E A D E R S  ***************/
header_type ethernet_t {
    fields {
        dstAddr   : 48;
        srcAddr   : 48;
        etherType : 16;
    }
}
/*****************. P A R S E R  ********************/
header ethernet_t ethernet;

parser start {
    extract(ethernet);
    return select(ethernet.etherType) {
        0x8100 : parse_vlan_tag;
        0x0800 : parse_ipv4;
        default: ingress;
}
```

```
/*************  I N G R E S S   P R O C E S S I N G ********/
action send(port) {
   modify_field(ig_intr_md_for_tm.ucast_egress_port, port);
}

table ipv4_host {
   reads {
      ipv4.dstAddr : exact;
   }
   actions {
      send;
   }
}

control ingress {
      if (valid(ipv4)) {
            apply(ipv4_host) {
            miss {
                  apply(ipv4_lpm);
                  }
            }
      }
}

/******************* E G R E S S  P R O C E S S I N G  ***********************/
control egress {
}
```

Header Types

```
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_type my_metadata {
    fields {
        L3_nexthop : 14;
        next_ttl   : 8 (saturating)
        mtu_adjust : 16 (signed)
}
```

P4_14 base types

- Unsigned integer (bitstring)
- Signed integer
- Saturated Integers

There are no “standalone” variables

- Everything is organized into header_type structures
  - Packet Headers
  - Metadata
- Byte-aligned (multiple of 8-bits)
- Can be valid or invalid
- Provides several operations to test and set validity bit: **isValid()**, **setValid()**, and **setInvalid()**

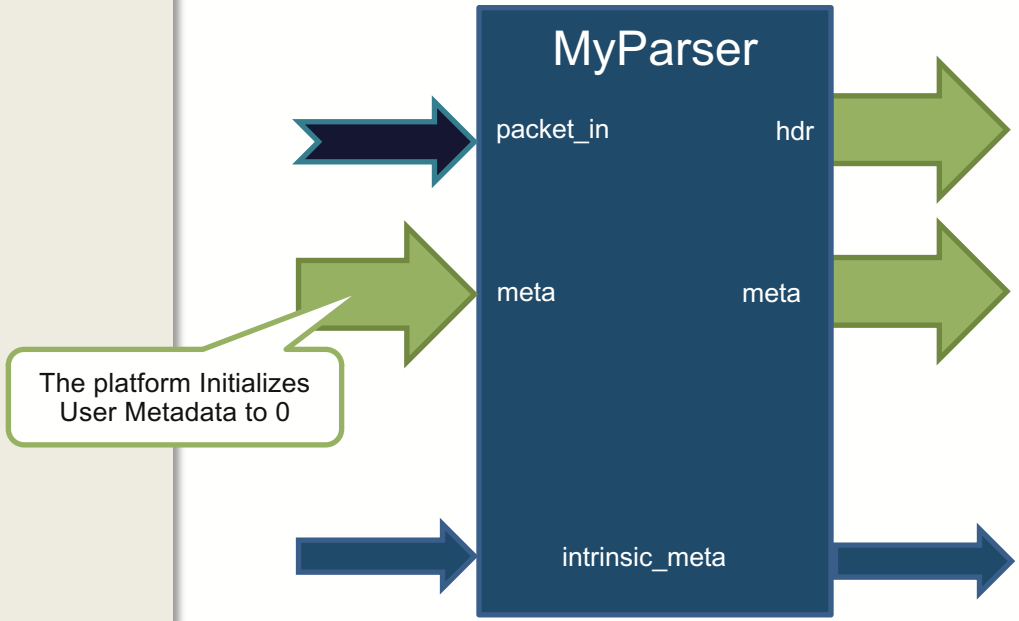
Parsers (MyParser)

```
header ethernet_t ethernet;
header vlan_tag_t vlan_tag[2];
header ipv4_t     ipv4;
Header my_metadata_t my_metadata;

parser start {
    extract(ethernet);
    return select(ethernet.etherType) {
        0x8100 : parse_vlan_tag;
        0x0800 : parse_ipv4;
        default: ingress;}
}

parser parse_vlan_tag {
    extract(vlan_tag[next]);
    return select(latest.etherType) {
        0x8100 : parse_vlan_tag;
        0x0800 : parse_ipv4;
        default: ingress;}
}

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

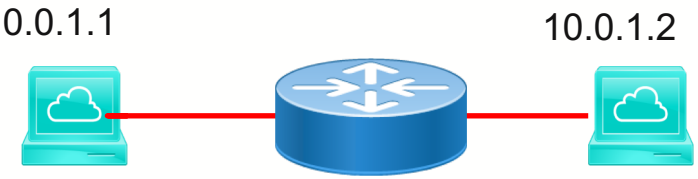


$P4_{14}$ Tables

- **The fundamental unit of a Match-Action Pipeline**
  - Specifies what data to match on and match kind
  - Specifies a list of *possible* actions
  - Optionally specifies a number of table **properties**
    - Size
    - Default action
    - etc.
- **Each table contains one or more entries (rules)**
- **An entry contains:**
  - A specific key to match on
  - A **single** action that is executed when a packet matches the entry
  - Action data (possibly empty)

IPv4_LPM Table

```
table ipv4_lpm {
    key = {
      hdr.ipv4.dstAddr: lpm;
    }
    actions = {
      ipv4_forward;
      drop;
      NoAction;
    }
    size = 1024;
    default_action = NoAction();
}
```



Key	Action	Action Data
10.0.1.1/32	ipv4_forward	dstAddr=00:00:00:00:01:01 port=1
10.0.1.2/32	drop	
*`	NoAction	

Actions

```
action nop {}

action remove_vlan_tag() {
     modify_field(ethernet.etherType, vlan_tag.etherType);
     remove_header(vlan.tag);
}

action ipv4_modify(dst_mac, src_mac, vid) {
     modify_field(ethernet.dstAddr, dst_mac);
     modify_field(ethernet.srcAddr, src_mac);
     modify_field(vlan_tap.vid, vid);
     add_to_field(ipv4.ttl, -1);
}

action simple_ipv4_route(dst_port) {
     modify_field(ig_intr_md_for_tm.ucast_egress_port,
dst_port);
     ipv4_modify(nexthop.dstmac, nexthop.src_mac,
l3_intf.vid);
```

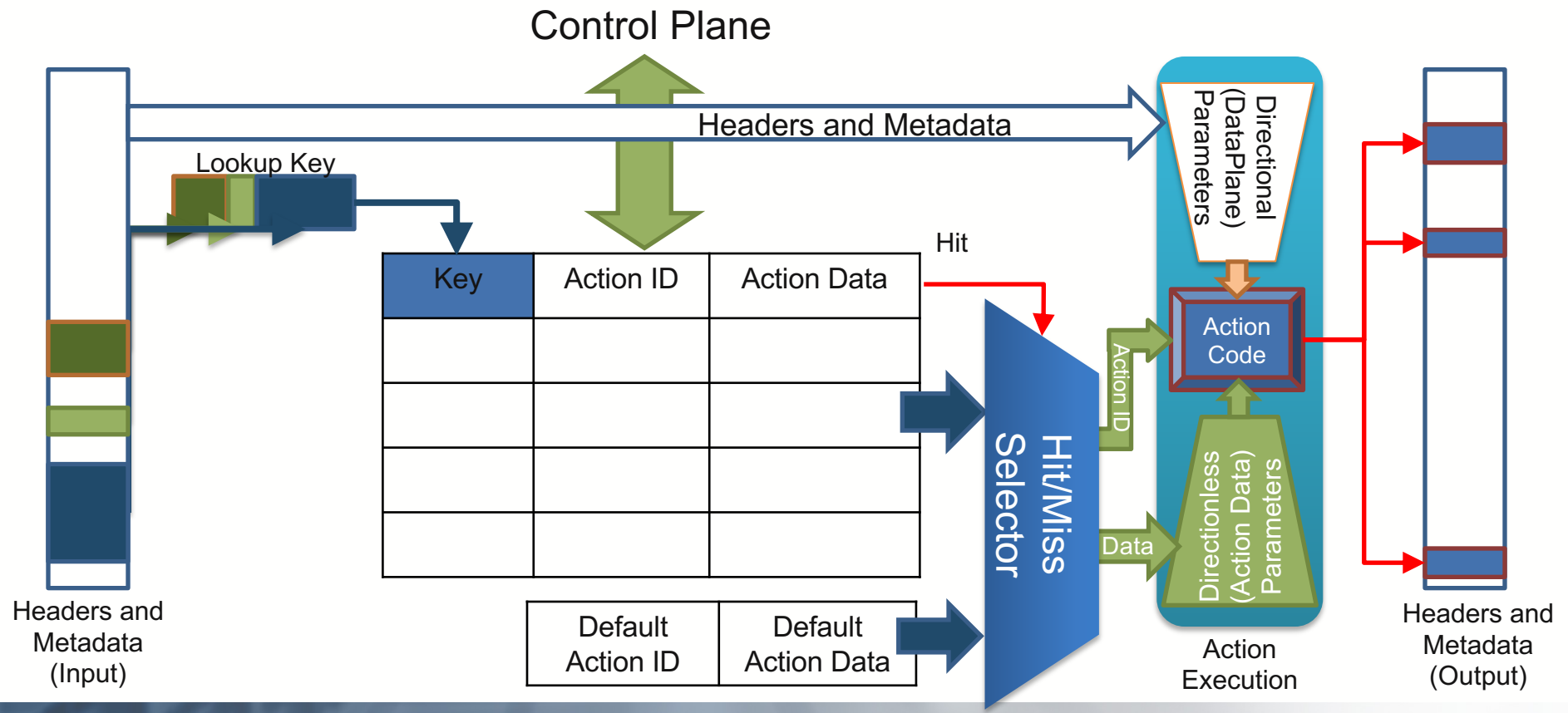
- **Defined by the programmer**
- **Consist of:**
  - calls to primitives which can't be used "standalone"
  - Calls to other actions (inlined by compiler)
- **Actions operate on**
  - Header and Metadata Fields
  - Constants (immediate values)
  - Action Data (provided by the control plane)
- **Sequential Execution Semantics**
  - Compiler will parallelize the code automatically
  - Compiler will reject the code it can't paraellize

$P4_{14}$ Controls

- **Similar to C functions (without loops)**
- **Functionality specified by code in `apply` statement**
- **Represent all kinds of processing :**
  - Match-Action Pipelines
  - Tables
- **Ingress Process**
- **Egress Process**

```
control ingress {
    if (valid(ipv4)) {
            apply(ipv4_host) {
        miss {
                apply(ipv4_lpm);
            }
            }
    }
}
```

Tables: Match-Action Processing



ARISTA

P4_14 Simple example

Running Simple_l3 Example: Basic Forwarding

- **We'll use a simple l3 application as a running example—a basic router—to illustrate the main features of $P4_{14}$**
- **Basic functionality:**
  - Parse Ethernet and IPv4 headers from packet
  - Match entry in ipv4_host (exact) and ipv4_lpm (lpm) tables
  - Perform action send() (set the egress port) or discard()
  - Deparse headers back into a packet
- **Example**

IP Address	Action	Parameters
192.168.1.1	Send	Port = 1
192.168.1.3	Discard	
192.168.0.0 / 16	Send	Port = 3
0.0.0.0 / 0	Send	Port = 64

Typical Workflow

- Write P4 program
- Compile P4 Program
- Start the target
  - Run the ASIC model
  - Load Kernel modules for the real ASIC
- Start the user-space driver process
- Program the tables
  - CLI, Control Plane Runtime
- Inject Packets
- Enjoy

Programming a P4 Target