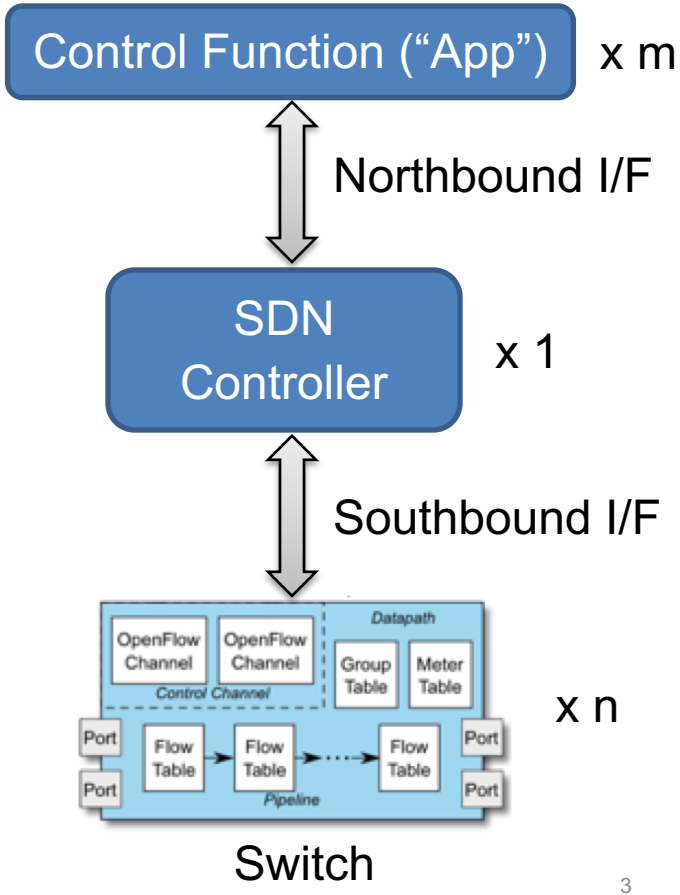
CS344 – Lecture 2
P4 Language Overview

What is Data Plane Programming?

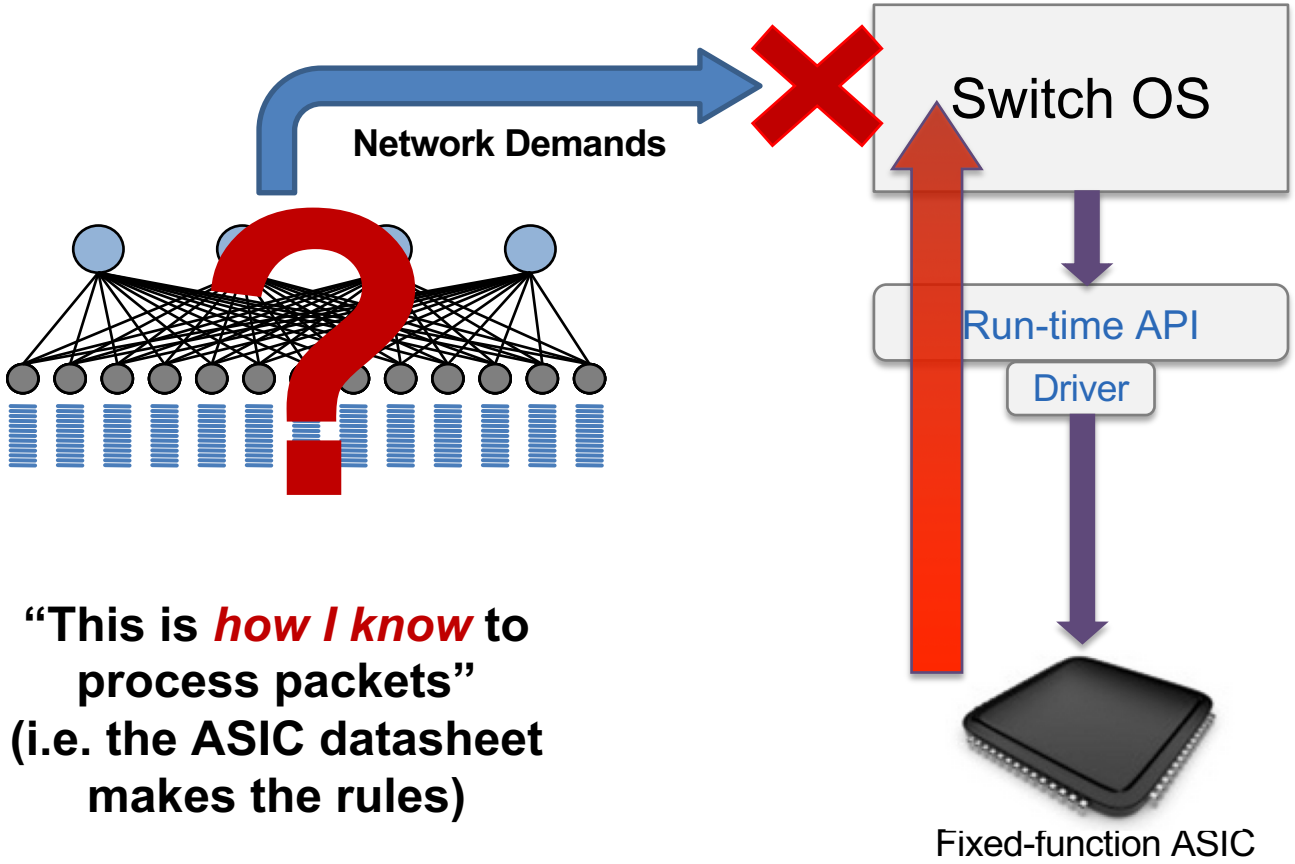
- **Why program the Data Plane?**

Software Defined Networking

- **Main contributions**
  - Match-Action abstraction
  - Standardized *protocol* to interact with switch
  - Logically centralized control via a single entity
- **Issues**
  - Data-plane protocol evolution requires changes to standards (12 → 40 OpenFlow match-fields)
  - Limited interoperability between vendors => southbound I/F differences handled at controller (OpenFlow / netconf / JSON / XML variants)
  - Limited programmability

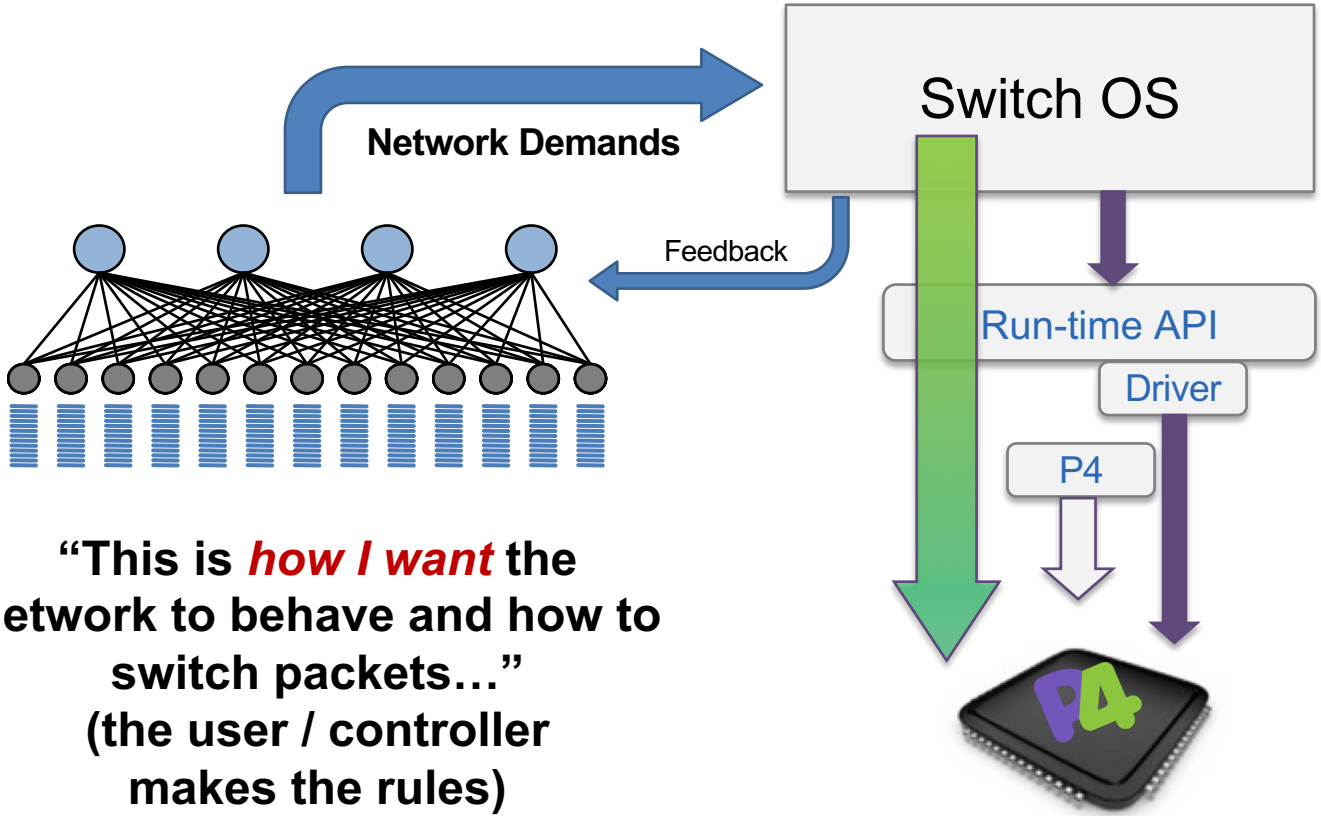


Status Quo: Bottom-up design



"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)

Benefits of Data Plane Programmability

- **New Features** – Add new protocols
- **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**

Think programming rather than protocols…

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.

What can you do with P4?

- **Layer 4 Load Balancer – SilkRoad[1]**
- **Low Latency Congestion Control – NDP[2]**
- **In-band Network Telemetry – INT[3]**
- **Fast In-Network cache for key-value stores – NetCache[4]**
- **Consensus at network speed – NetPaxos[5]**
- **Aggregation for MapReduce Applications [6]**
- **… 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] Dang, Huynh Tu, et al. "NetPaxos: Consensus at network speed." SIGCOMM, 2015.
[6] Sapio, Amedeo, et al. "In-Network Computation is a Dumb Idea Whose Time Has Come." *Hot Topics in Networks*. ACM, 2017.

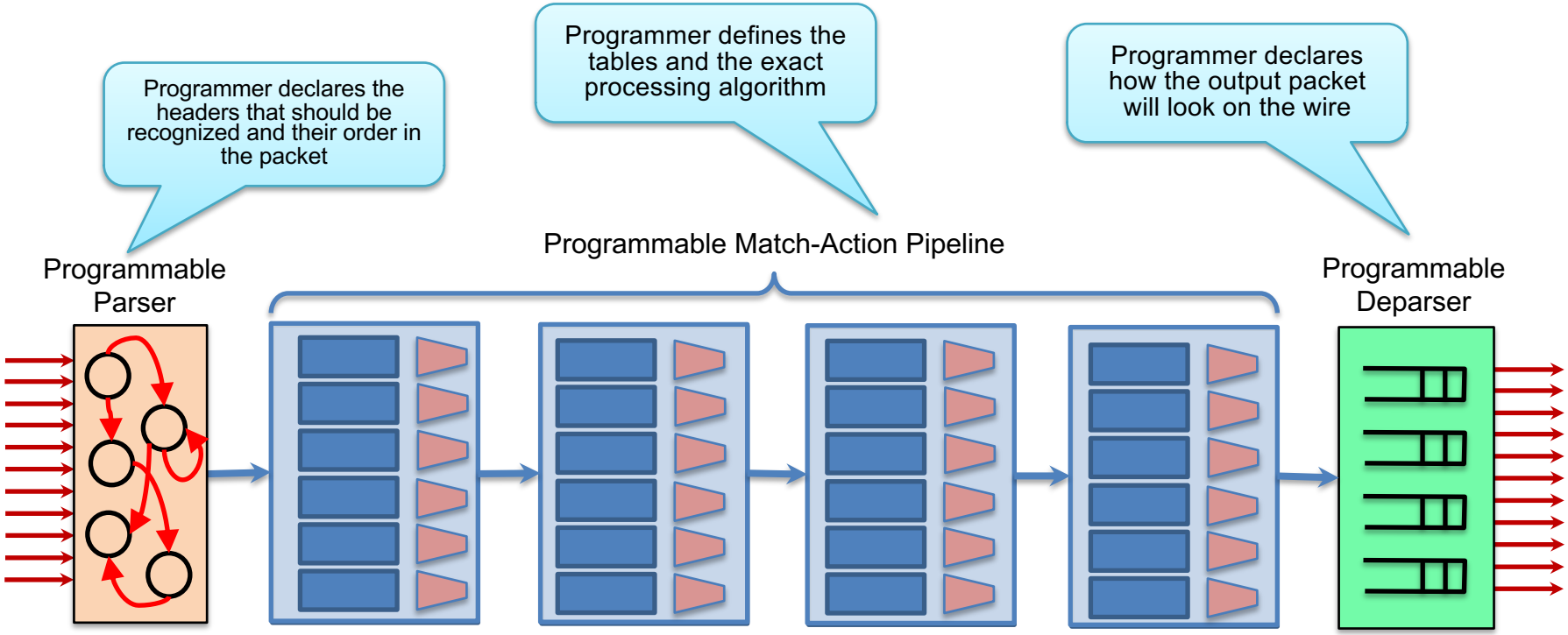
Brief History and Trivia

- 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

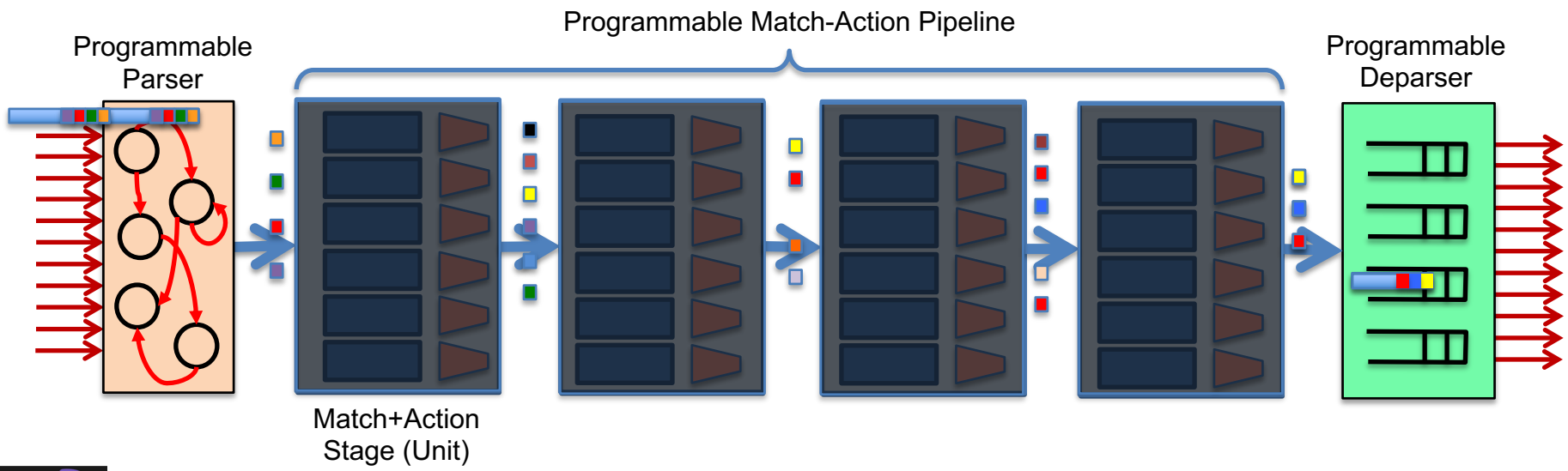
P4 Data Plane Model

PISA: Protocol-Independent Switch Architecture

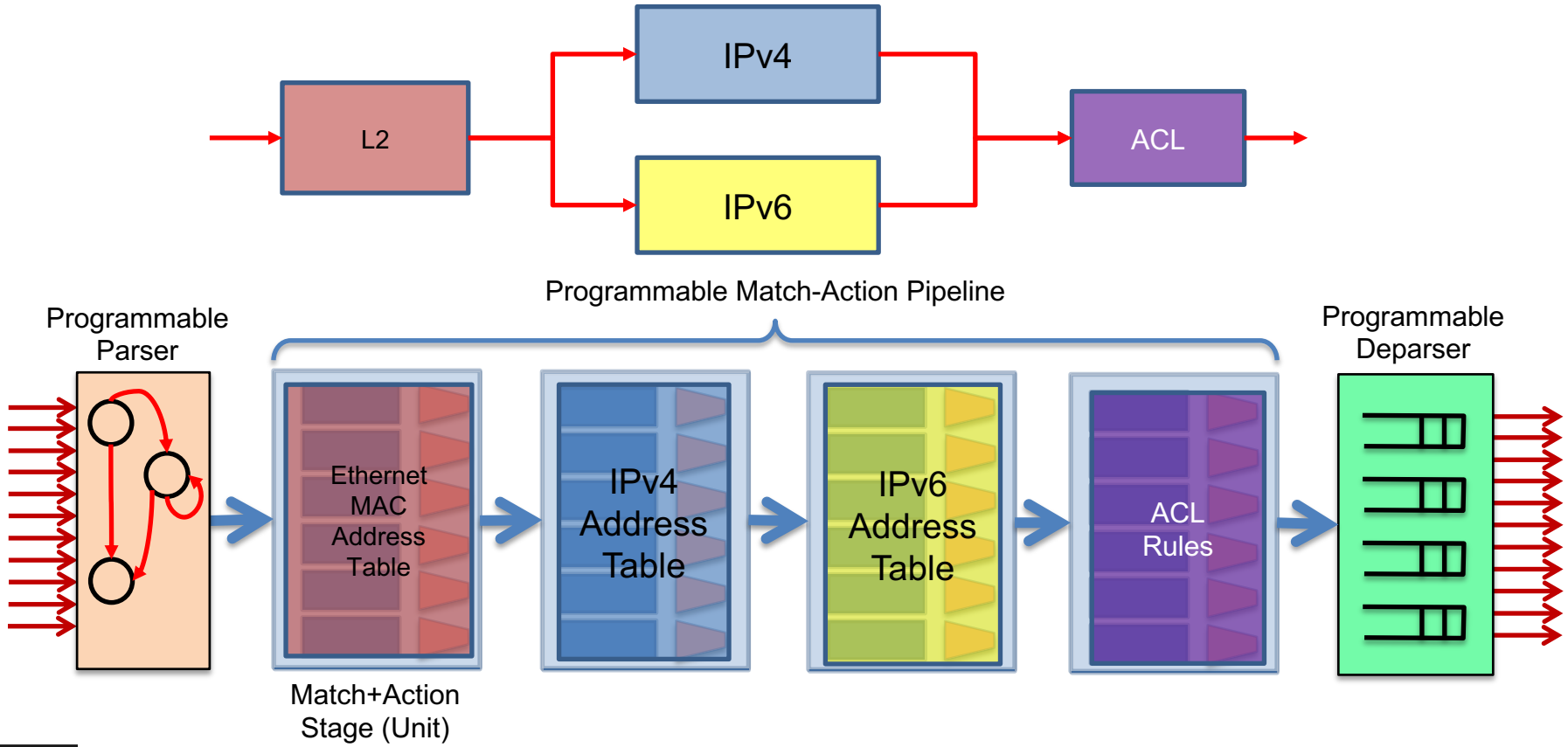


PISA in Action

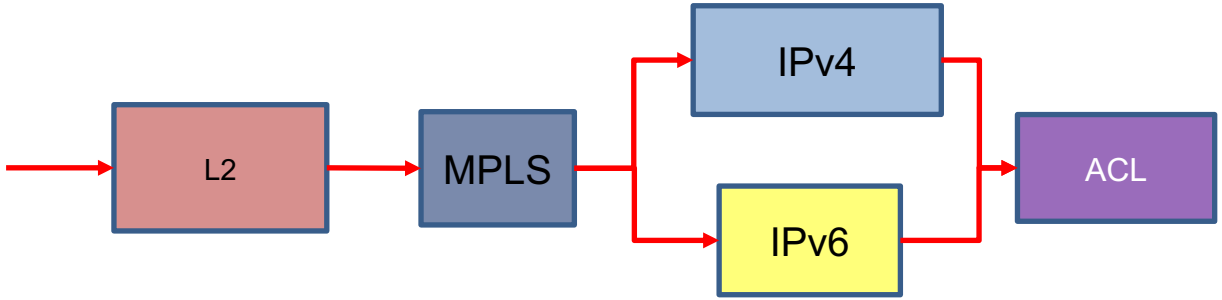
- Packet is parsed into individual headers (parsed representation)
- Headers and intermediate results can be used for matching and actions
- Headers can be modified, added or removed
- Packet is deparsed (serialized)

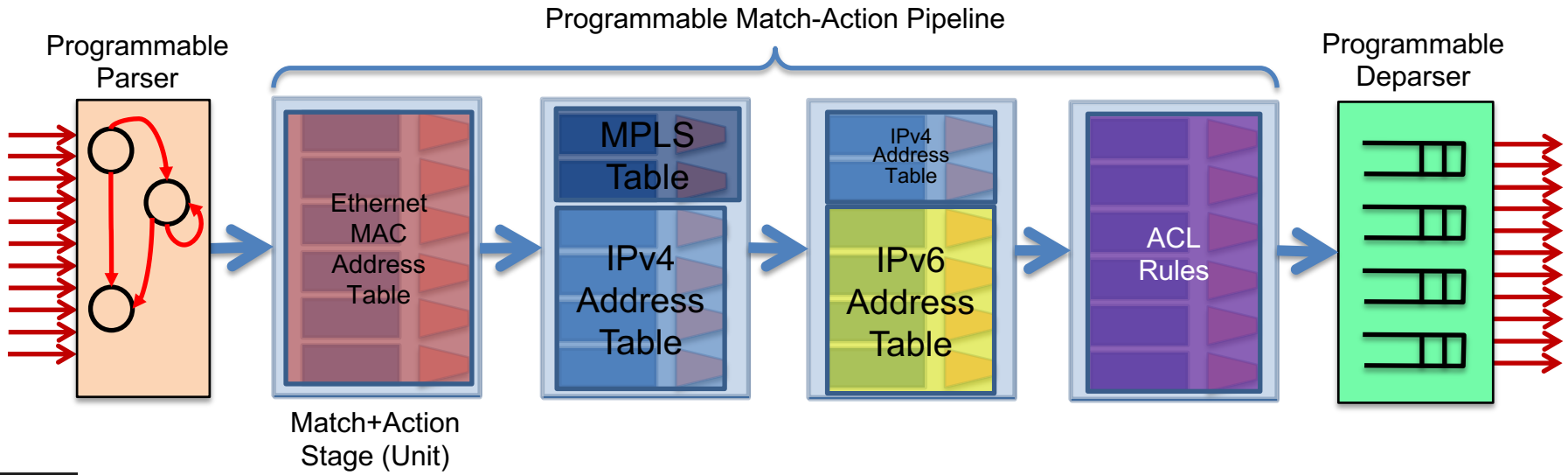


Mapping a Simple L3 Data Plane Program on PISA

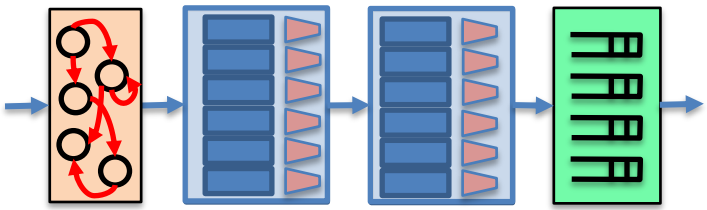


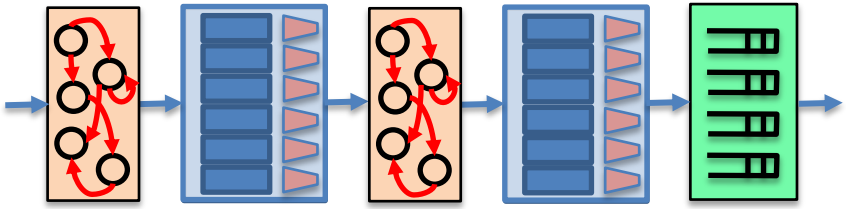
Mapping a More Complex Data Plane Program on PISA

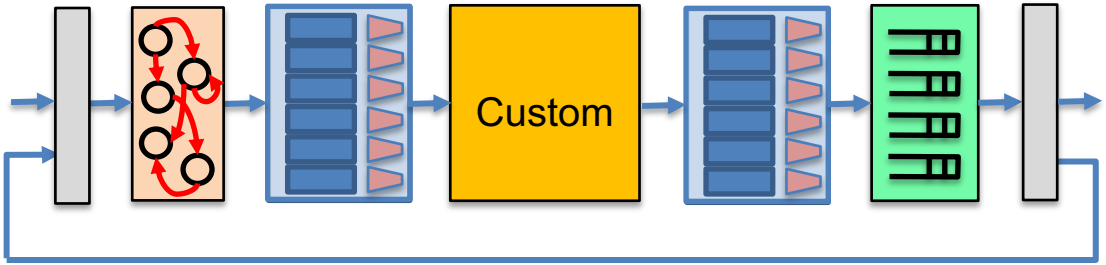




P4 Architectures

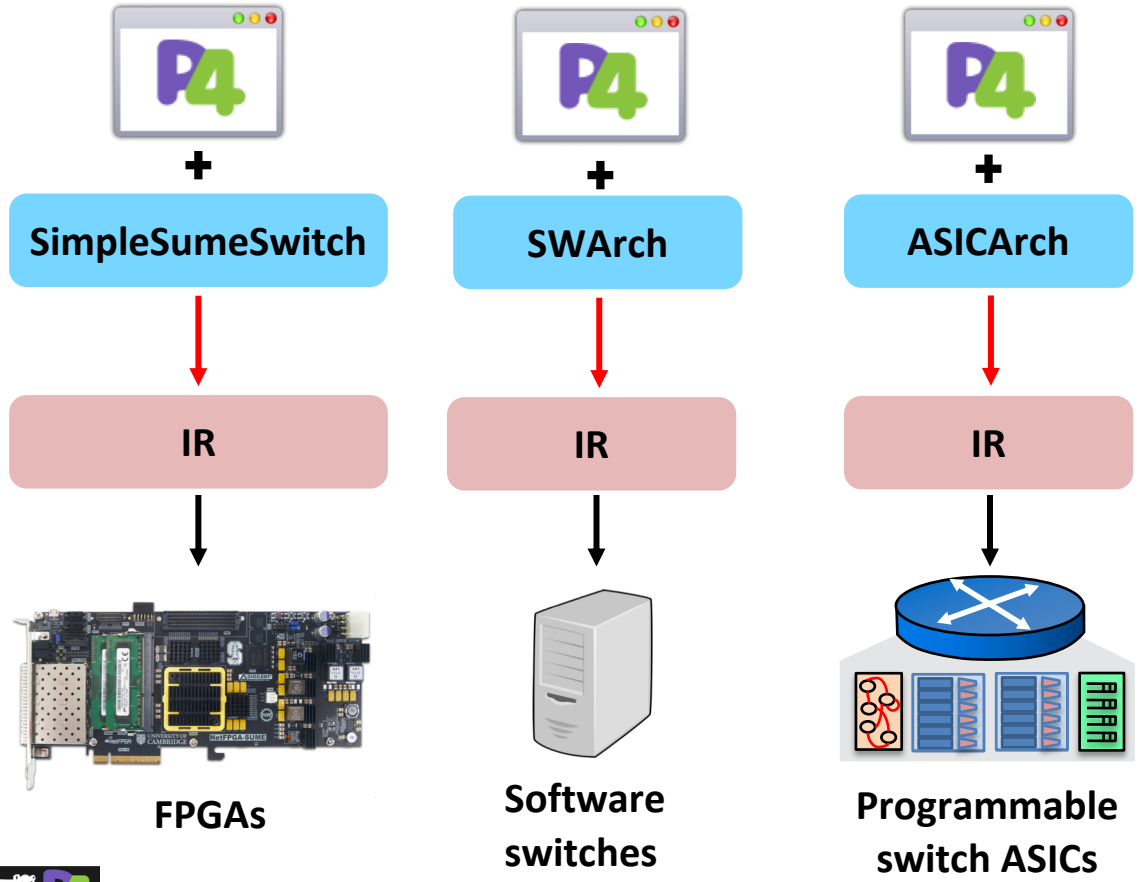






- Define how the P4 programmable elements are composed
- Define how to interact with non-programmable elements
- P4 programs are written for a specific architecture

P4 Approach



P4 Programs

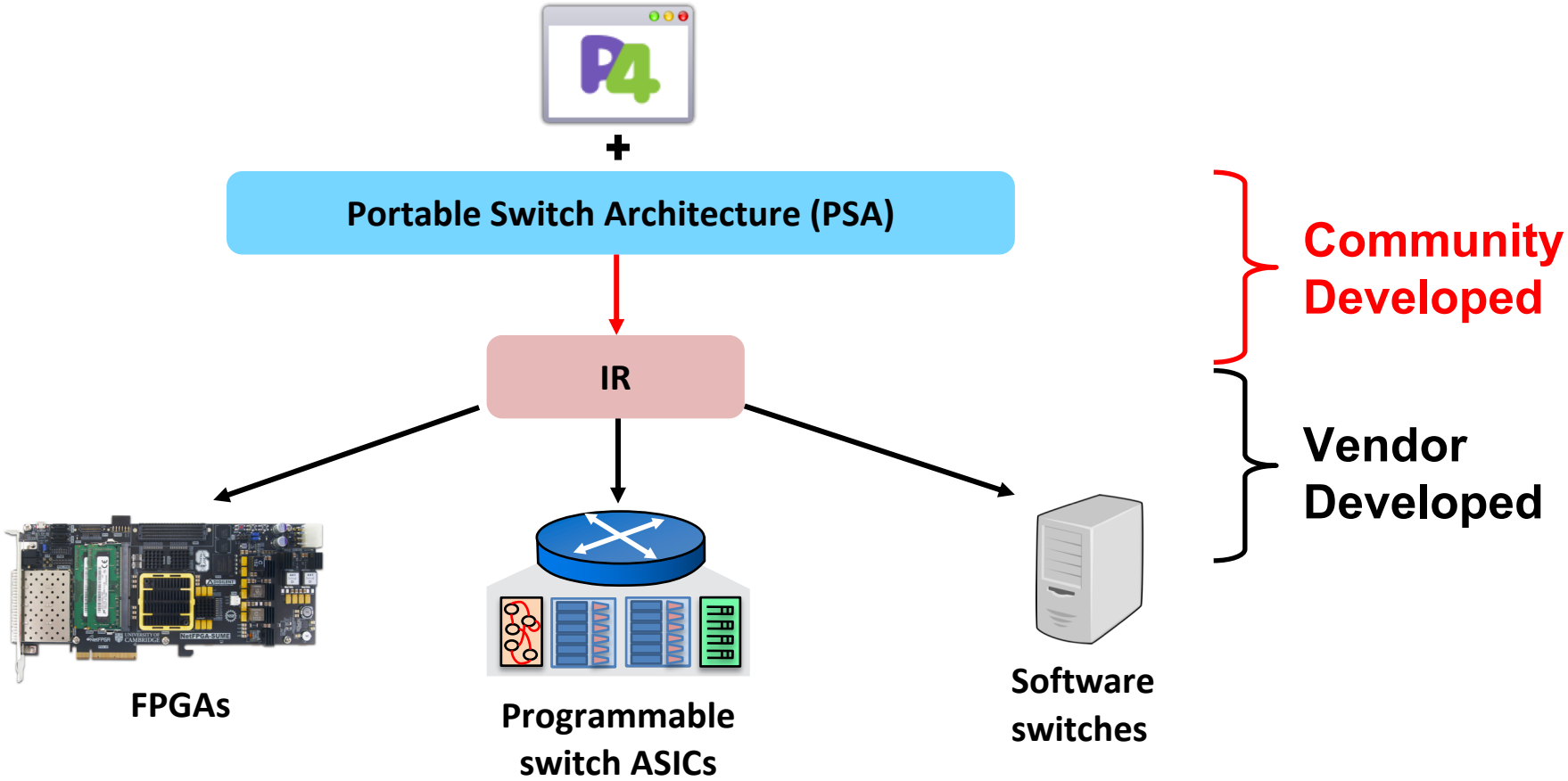
Architecture

Community Developed

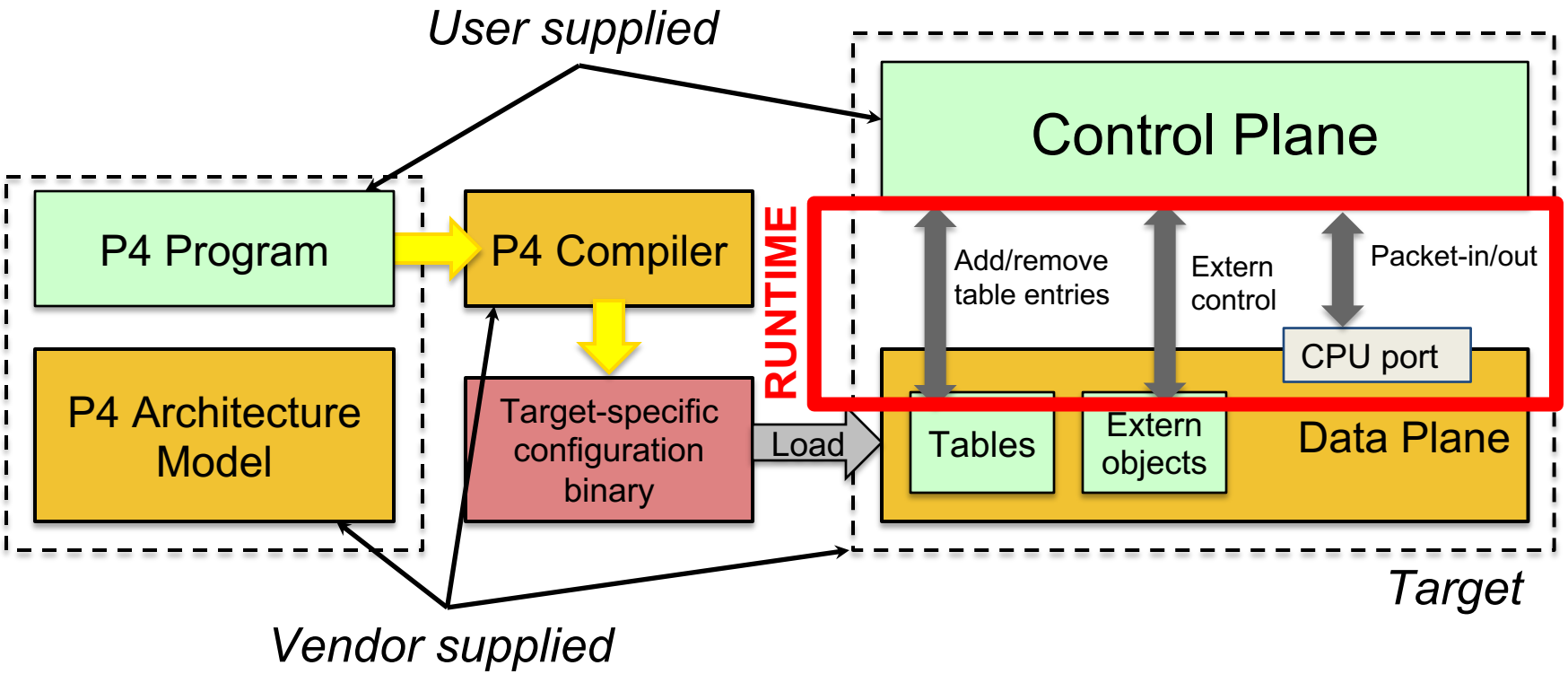
Vendor Developed

Target

P4 Goal



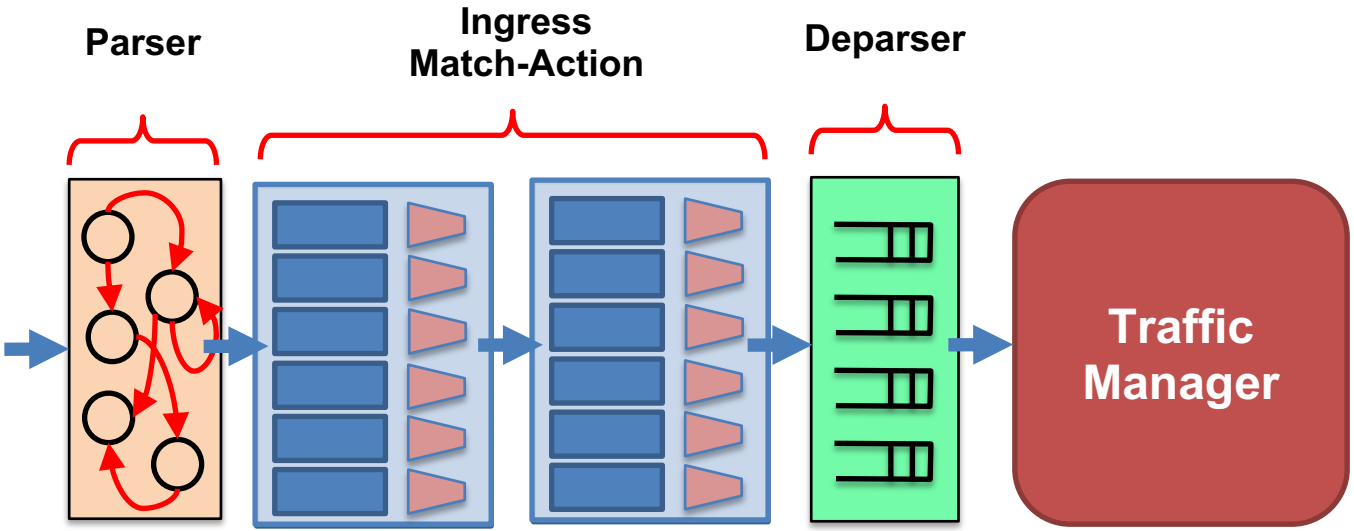
Programming a P4 Target



SimpleSumeSwitch Architecture

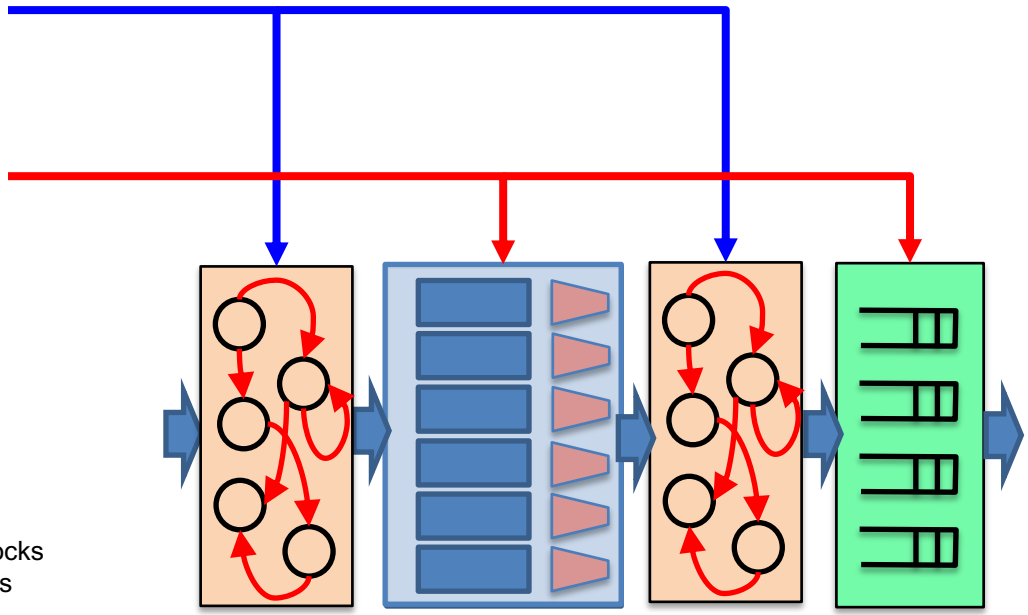
Supported on:

- NetFPGA SUME board
- BMv2 in Mininet



$P4_{16}$ Language Elements

Parsers	State machine, bitfield extraction
Controls	Tables, Actions, control flow statements
Expressions	Basic operations and operators
Data Types	Bistrings, headers, structures, arrays
Architecture Description	Programmable blocks and their interfaces
Extern Libraries	Support for specialized components



Standard Metadata in SimpleSumeSwitch Architecture

```
/* standard sume switch metadata */
struct sume_metadata_t {
    bit<16> dma_q_size;
    bit<16> nf3_q_size;
    bit<16> nf2_q_size;
    bit<16> nf1_q_size;
    bit<16> nf0_q_size;
    // send digest_data to CPU
    bit<8> send_dig_to_cpu;
    // ports are one-hot encoded
    bit<8> dst_port;
    bit<8> src_port;
    // pkt len is measured in bytes
    bit<16> pkt_len;
}
```

- **`src_port/dst_port`** – one-hot encoded, easy to do multicast
- **`*_q_size`** – size of each output queue, measured in terms of 32-byte words, when packet starts being processed by the P4 program
- **`send_dig_to_cpu`** – to send `digest_data` to control-plane

P4 Program Template (SimpleSumeSwitch)

```
#include <core.p4>
#include <sume_switch.p4>
/* HEADERS and METADATA */
struct user_data { ... }
struct digest_data { ... }
struct headers {
  ethernet_t   ethernet;
  ipv4_t       ipv4;
}
/* PARSER */
parser MyParser(packet_in packet,
                out headers hdr,
                out user_data_t user,
                out digest_data_t digest,
                inout sume_metadata_t smeta) {
  ...
}
/* MATCH-ACTION PROCESSING */
control MyIngress(inout headers hdr,
                  inout user_data_t user,
                  inout digest_data_t digest,
                  inout sume_metadata_t smeta) {
  ...
}
```

```
/* DEPARSER */
control MyDeparser(packet_out packet,
                   in headers hdr,
                   in user_data_t user,
                   inout digest_data_t digest,
                   inout sume_metadata_t smeta) {
  ...
}
/* SWITCH */
SimpleSumeSwitch(
  MyParser(),
  MyIngress(),
  MyDeparser()
) main;
```

P4 Hello World (SimpleSumeSwitch)

```
#include <core.p4>
#include <sume_switch.p4>
/* HEADERS and METADATA */
struct user_data { ... }
struct digest_data { ... }
struct headers { ... }
/* PARSER */
parser MyParser(packet_in packet,
                out headers hdr,
                out user_data_t user,
                out digest_data_t digest,
                inout sume_metadata_t smeta) {
 state start { transition accept; }
}
/* MATCH-ACTION PROCESSING */
control MyIngress(inout headers hdr,
                  inout user_data_t user,
                  inout digest_data_t digest,
                  inout sume_metadata_t smeta) {
  apply {
    if (smeta.src_port == 1) {
       smeta.dst_port = 2;
    } else if (smeta.src_port == 2) {
       smeta.dst_port = 1;
    }
  }
}
```

```
/* DEPARSER */
control MyDeparser(packet_out packet,
                   in headers hdr,
                   in user_data_t user,
                   inout digest_data_t digest,
                   inout sume_metadata_t smeta) {
  apply {}
}
/* SWITCH */
SimpleSumeSwitch(
  MyParser(),
  MyIngress(),
  MyDeparser()
) main;
```

P4 Hello World (SimpleSumeSwitch)

```
#include <core.p4>
#include <sume_switch.p4>
/* HEADERS and METADATA */
struct headers { ... }
/* PARSER */
parser MyParser(packet_in packet, out headers hdr, ...) {
 state start { transition accept; }
}
/* MATCH-ACTION PROCESSING */
control MyIngress(..., inout sume_metadata_t smeta) {

  action set_dst_port(bit<8> port) {
    smeta.dst_port = port;
  }

  table forward {
    key = { smeta.src_port : exact; }
    actions = {
      set_dst_port;
      NoAction;
    }
    size = 1024;
    default_action = NoAction();
  }

  apply {   forward.apply();   }
}
```

```
/* DEPARSER */
control MyDeparser(packet_out packet,
                   in headers hdr,
                   in user_data_t user,
                   inout digest_data_t digest,
                   inout sume_metadata_t smeta) {
  apply {}
}
/* SWITCH */
SimpleSumeSwitch(
  MyParser(),
  MyIngress(),
  MyDeparser()
) main;
```

forward table entries:

Key	Action Name	Action Data
1	set_dst_port	2
2	set_dst_port	1

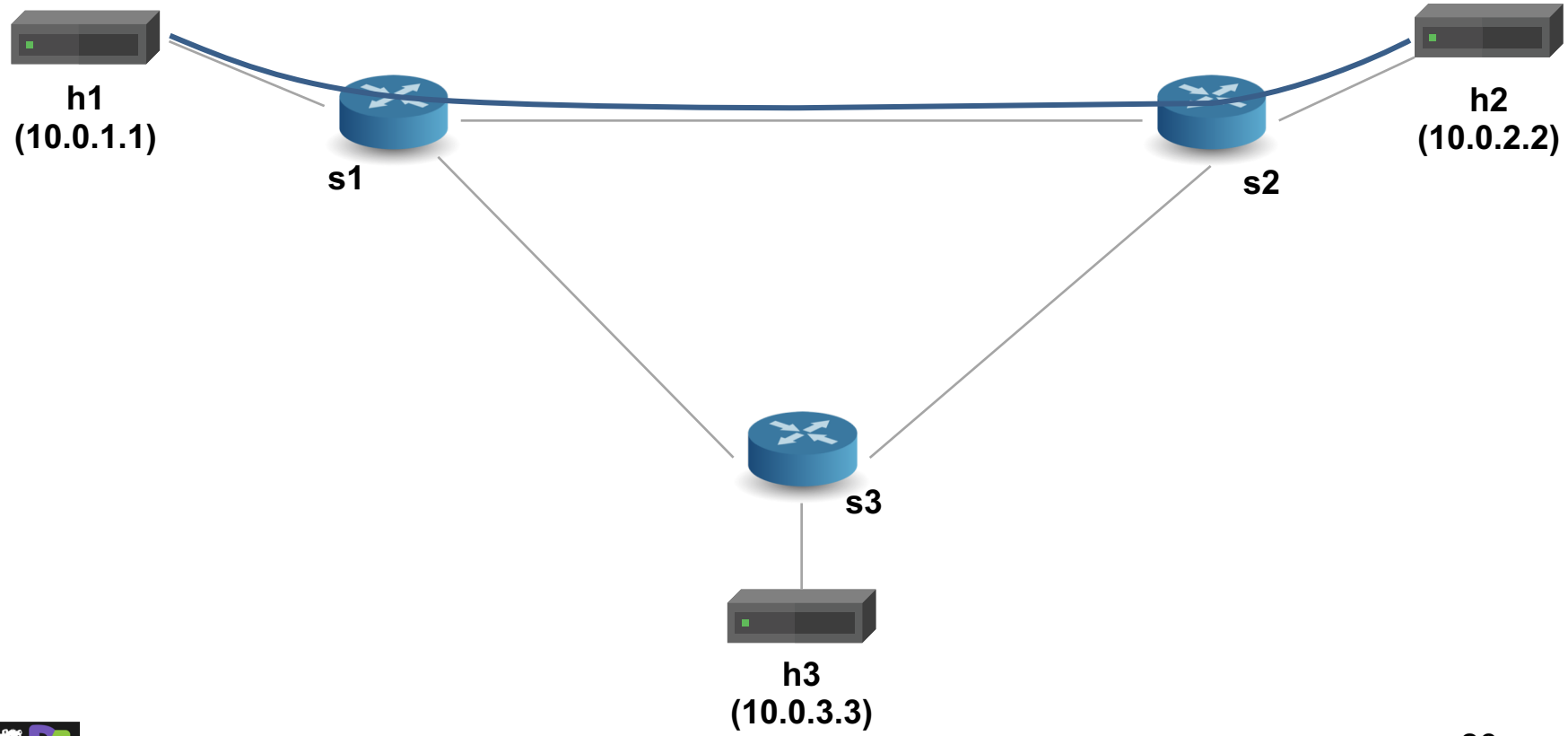
Running Example: Basic Forwarding

- **We'll use a simple application as a running example—a basic router—to illustrate the main features of $P4_{16}$**

- **Basic router functionality:**
  - Parse Ethernet and IPv4 headers from packet
  - Find destination in IPv4 routing table
  - Update source / destination MAC addresses
  - Decrement time-to-live (TTL) field & update IP checksum
  - Set the egress port
  - Deparse headers back into a packet

- **We've written some starter code for you (`basic.p4`) and implemented a static control plane**

Basic Forwarding: Topology



P4 Types (Basic and Header Types)

```
typedef bit<48> macAddr_t;
typedef bit<32> ip4Addr_t;
header ethernet_t {
  macAddr_t dstAddr;
  macAddr_t srcAddr;
  bit<16>   etherType;
}
header ipv4_t {
  bit<4>    version;
  bit<4>    ihl;
  bit<8>    diffserv;
  bit<16>   totalLen;
  bit<16>   identification;
  bit<3>    flags;
  bit<13>   fragOffset;
  bit<8>    ttl;
  bit<8>    protocol;
  bit<16>   hdrChecksum;
  ip4Addr_t srcAddr;
  ip4Addr_t dstAddr;
}
```

Basic Types

- **bit<n>**: Unsigned integer (bitstring) of size n
- **bit** is the same as **bit<1>**
- **int<n>**: Signed integer of size n (>=2)
- **varbit<n>**: Variable-length bitstring

Header Types: Ordered collection of members

- Can contain **bit<n>**, **int<n>**, and **varbit<n>**
- Byte-aligned
- Can be valid or invalid
- Provides several operations to test and set validity bit: **isValid()**, **setValid()**, and **setInvalid()**

Typedef: Alternative name for a type

P4 Types (Other Types)

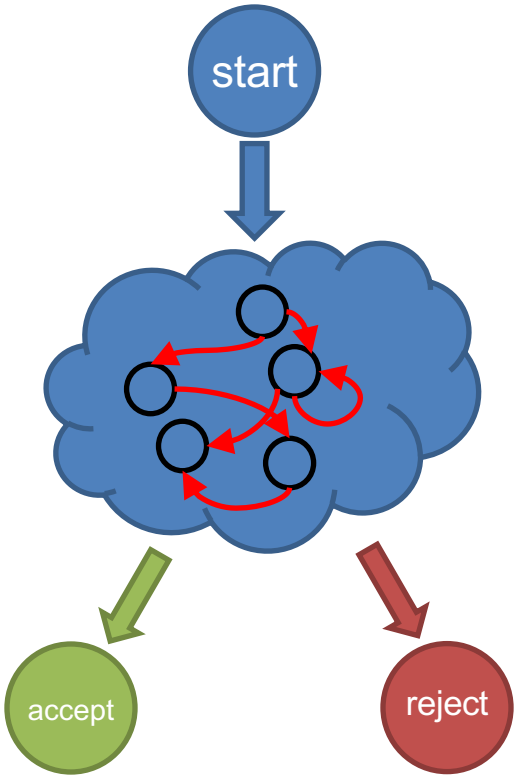
```
/* Architecture */
struct sume_metadata_t {
    bit<16> dma_q_size;
    bit<16> nf3_q_size;
    bit<16> nf2_q_size;
    bit<16> nf1_q_size;
    bit<16> nf0_q_size;
    bit<8> send_dig_to_cpu;
    bit<8> dst_port;
    bit<8> src_port;
    bit<16> pkt_len;
}

/* User program */
struct user_metadata {
  ...
}
struct headers {
  ethernet_t   ethernet;
  ipv4_t       ipv4;
}
```

Other useful types

- **Struct**: Unordered collection of members (with no alignment restrictions)
- **Header Stack:** array of headers
- **Header Union:** one of several headers

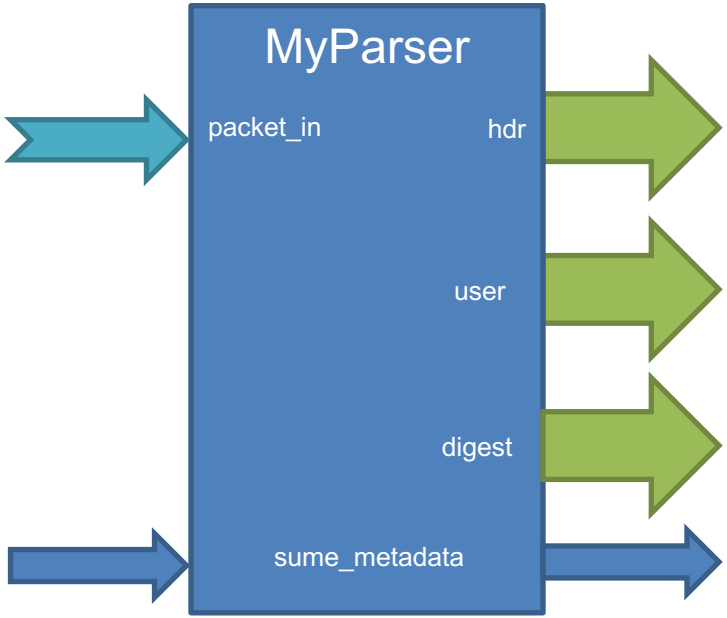
P4 Parsers

- **Parsers are functions that map packets into headers and metadata, written in a state machine style**
- **Every parser has three predefined states**
  - `start`
  - `accept`
  - `reject`
- **Other states may be defined by the programmer**
- **In each state, execute zero or more statements, and then transition to another state (loops are OK)**



Parsers (SimpleSumeSwitch)

```
/* From core.p4 */
extern packet_in {
  void extract<T>(out T hdr);
  void extract<T>(out T variableSizeHeader,
                  in bit<32> variableFieldSizeInBits);
  T lookahead<T>();
  void advance(in bit<32> sizeInBits);
  bit<32> length();
}
/* User Program */
parser MyParser(packet_in packet,
                out headers hdr,
                out user_data_t user,
                out digest_data_t digest,
                inout sume_metadata_t smeta) {

  state start {
     packet.extract(hdr.ethernet);
     transition accept;
  }

}
```



Select Statement

```
state start {
  transition parse_ethernet;
}

state parse_ethernet {
  packet.extract(hdr.ethernet);
  transition select(hdr.ethernet.etherType) {
    0x800: parse_ipv4;
    default: accept;
  }
}
```

$P4_{16}$ has a `select` statement that can be used to branch in a parser

Similar to `case` statements in C or Java, but without "fall-through behavior"—i.e., `break` statements are not needed

In parsers it is often necessary to branch based on some of the bits just parsed

For example, etherType determines the format of the rest of the packet

Match patterns can either be literals or simple computations such as masks

Coding Break

P4 Controls

- **Similar to C functions (without loops)**
- **Can declare variables, create tables, instantiate externs, etc.**
- **Functionality specified by code in `apply` statement**
- **Represent all kinds of processing that are expressible as DAG:**
  - Match-Action Pipelines
  - Deparsers
  - Additional forms of packet processing (updating checksums)
- **Interfaces with other blocks are governed by user- and architecture-specified types (typically headers and metadata)**

Example: Reflector (V1Model)

```
control MyIngress(inout headers hdr,
                  inout user_data_t user,
                  inout digest_data_t digest,
                  inout sume_metadata_t smeta) {

  bit<48> tmp;
  apply {
    tmp = hdr.ethernet.dstAddr;
    hdr.ethernet.dstAddr = hdr.ethernet.srcAddr;
    hdr.ethernet.srcAddr = tmp;
    smeta.dst_port = smeta.src_port;
  }
}
```

Desired Behavior:

- **Swap source and destination MAC addresses**
- **Bounce the packet back out on the physical port that it came into the switch on**

Example: Simple Actions

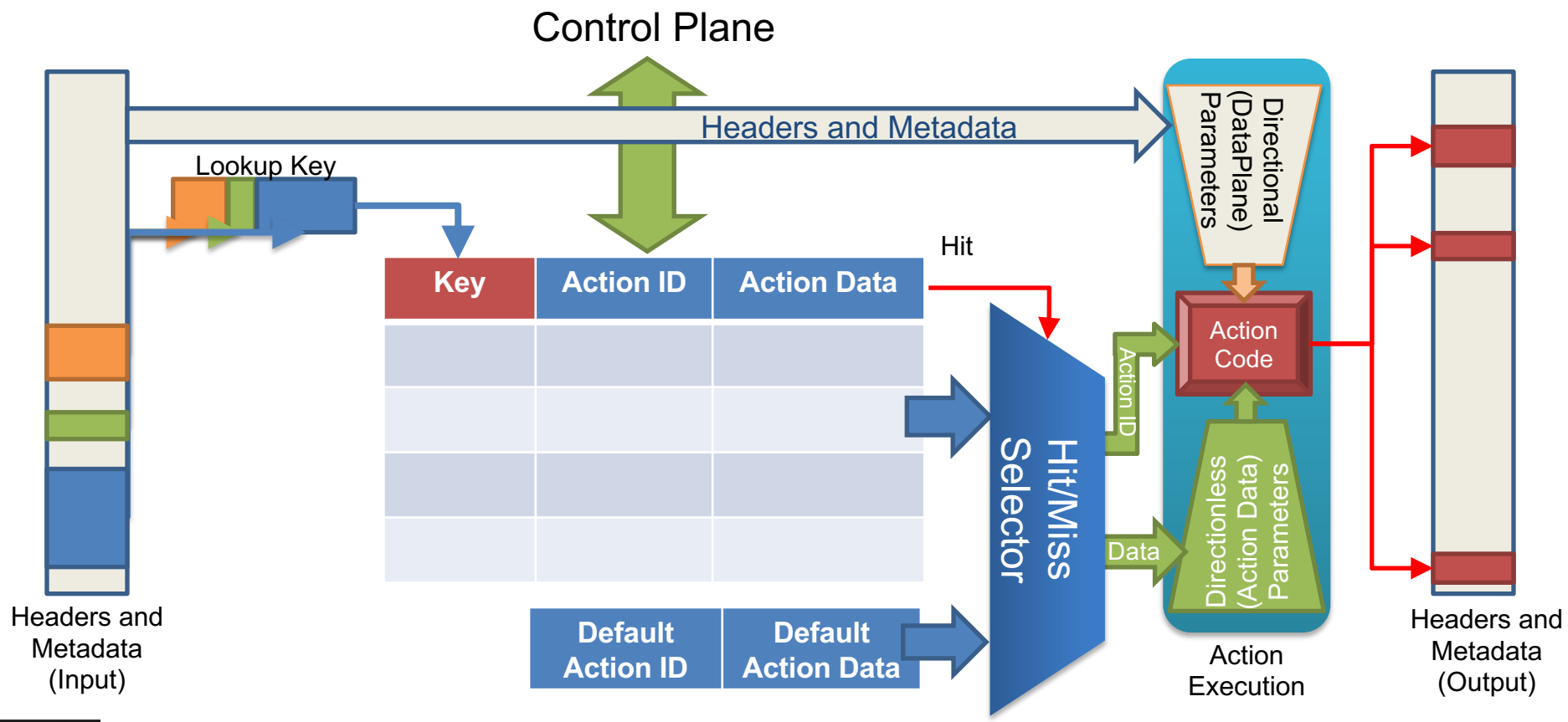
```
control MyIngress(inout headers hdr,
                  inout user_data_t user,
                  inout digest_data_t digest,
                  inout sume_metadata_t smeta) {

  action swap_mac(inout bit<48> src,
                  inout bit<48> dst) {
    bit<48> tmp = src;
    src = dst;
    dst = tmp;
  }

  apply {
    swap_mac(hdr.ethernet.srcAddr,
             hdr.ethernet.dstAddr);
    smeta.dst_port = smeta.src_port;
  }
}
```

- **Very similar to C functions**
- **Can be declared inside a control or globally**
- **Parameters have type and direction**
- **Variables can be instantiated inside**
- **Many standard arithmetic and logical operations are supported**
  - +, -, *
  - ~, &, |, ^, >>, <<
  - ==, !=, >, >=, <, <=
  - No division/modulo
- **Non-standard operations:**
  - Bit-slicing: [m:l] (works as l-value too)
  - Bit Concatenation: ++

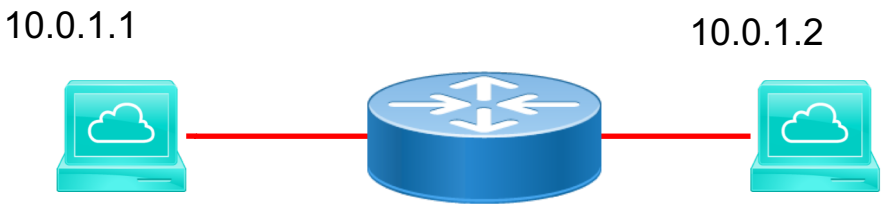
P4 Tables

- **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
    - Static entries
    - 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)

Tables: Match-Action Processing



Example: IPv4_LPM Table



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	

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

IPv4_LPM Table

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

Match Kinds

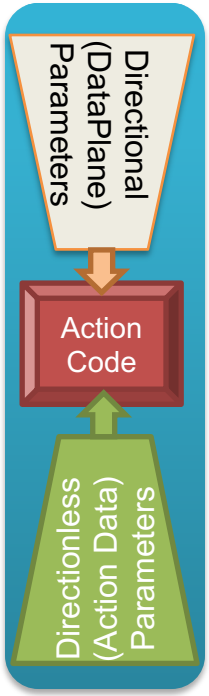
```
/* core.p4 */
match_kind {
    exact,
    ternary,
    lpm
}

/* v1model.p4 */
match_kind {
    range,
    selector
}

/* Some other architecture */
match_kind {
    regexp,
    fuzzy
}
```

- **The type `match_kind` is special in P4**
- **The standard library (`core.p4`) defines three standard match kinds**
  - Exact match
  - Ternary match
  - LPM match
- **The architecture (`v1model.p4`) defines two additional match kinds:**
  - range
  - selector
- **Other architectures may define (and provide implementation for) additional match kinds**

Defining Actions for L3 forwarding

```
/* core.p4 */
action NoAction() {
}

/* basic.p4 */
action drop() {
  mark_to_drop();
}

/* basic.p4 */
action ipv4_forward(macAddr_t dstAddr,
                    bit<9> port) {
  ...
}
```

- **Actions can have two different types of parameters**
  - Directional (from the Data Plane)
  - Directionless (from the Control Plane)
- **Actions that are called directly:**
  - Only use directional parameters
- **Actions used in tables:**
  - Typically use directionless parameters
  - May sometimes use directional parameters too



Action Execution

Applying Tables in Controls

```
control MyIngress(inout headers hdr,
                  inout user_data_t user,
                  inout digest_data_t digest,
                  inout sume_metadata_t smeta) {

  table ipv4_lpm {
    ...
  }
  apply {
     ...
     ipv4_lpm.apply();
     ...
  }
}
```

P4 Deparsing

```
/* From core.p4 */
extern packet_out {
  void emit<T>(in T hdr);
}
/* User Program */
control MyDeparser(packet_out packet,
                   in headers hdr,
                   in user_data_t user,
                   inout digest_data_t digest,
                   inout sume_metadata_t smeta)
{
  apply {
    ...
    packet.emit(hdr.ethernet);
    ...
  }
}
```

- **Assembles the headers back into a well-formed packet**
- **Expressed as a control function**
  - No need for another construct!
- **packet_out extern is defined in core.p4:** emit(hdr): serializes header if it is valid
- **Advantages:**
  - Makes deparsing explicit...
    ...but decouples from parsing

P4 Externs

```
// special SUME hash function
extern void sume_hash(in bit<64> data,
                      out bit<8> result);

control MyIngress(inout headers hdr,
                  inout user_data_t user,
                  inout digest_data_t digest,
                  inout sume_metadata_t smeta)
{


  apply {
    bit<8> flowID;
    sume_hash(hdr.ip.src++hdr.ip.dst, flowID);
    ...
  }
}
```

- **Black boxes for P4 programs**
- **Functionality is not described in P4**
- **Used to perform device/vendor specific functionality**
- **Can be stateless of stateful**
- **Can be accessed by the control-plane**
- **Set of supported externs is defined by architecture**

Fin!