

CS344 – Lecture 2

P4 Language Overview



What is Data Plane Programming?

- Why program the Data Plane?

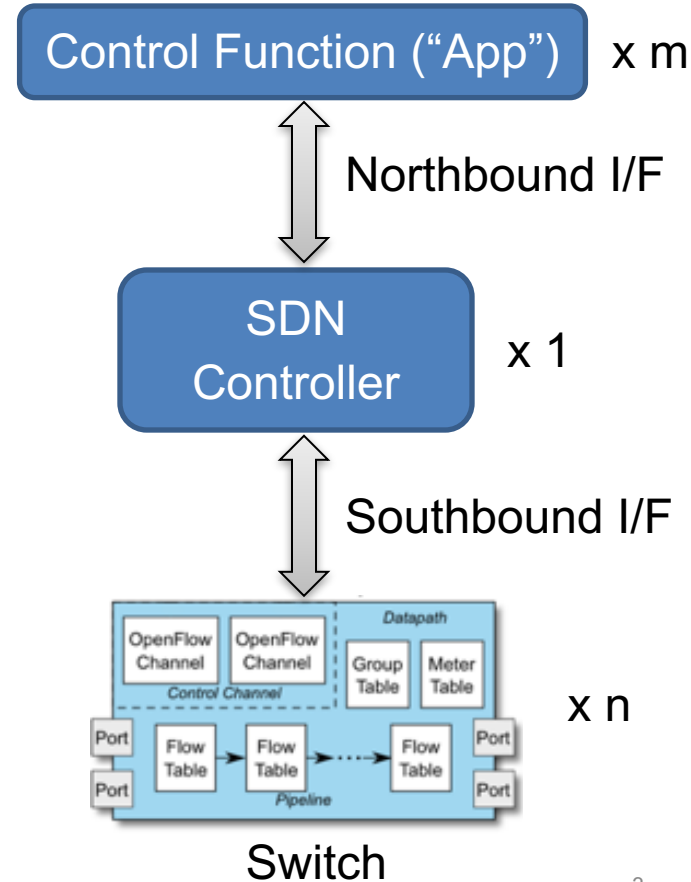
Software Defined Networking (SDN)

- **Main contributions**

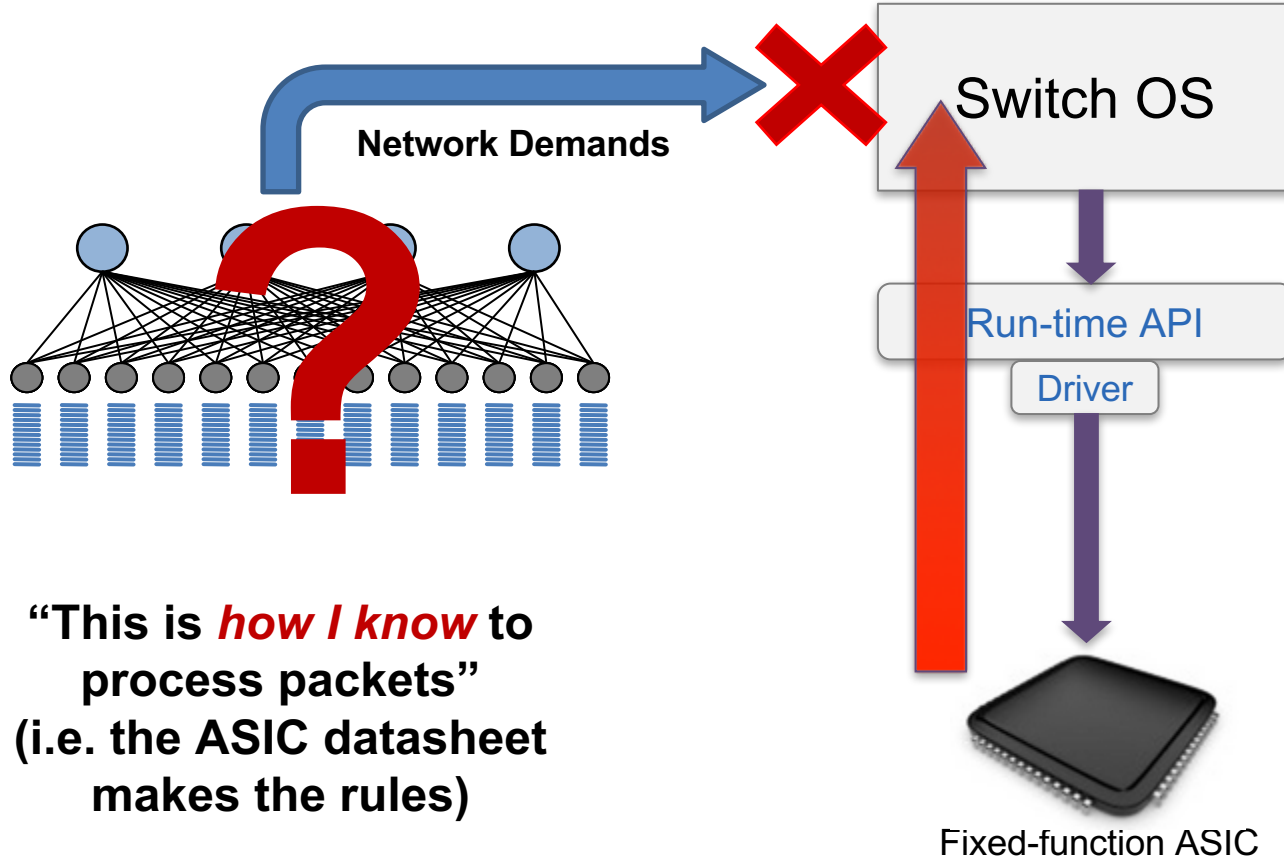
- OpenFlow = standardized *protocol* to interact with switch
 - download flow table entries, query statistics, etc.
- OpenFlow = standardized *model*
 - match/action abstraction
- *Concept of logically* centralized control via a single entity (“SDN controller”)
 - Simplifies control plane

- **Issues**

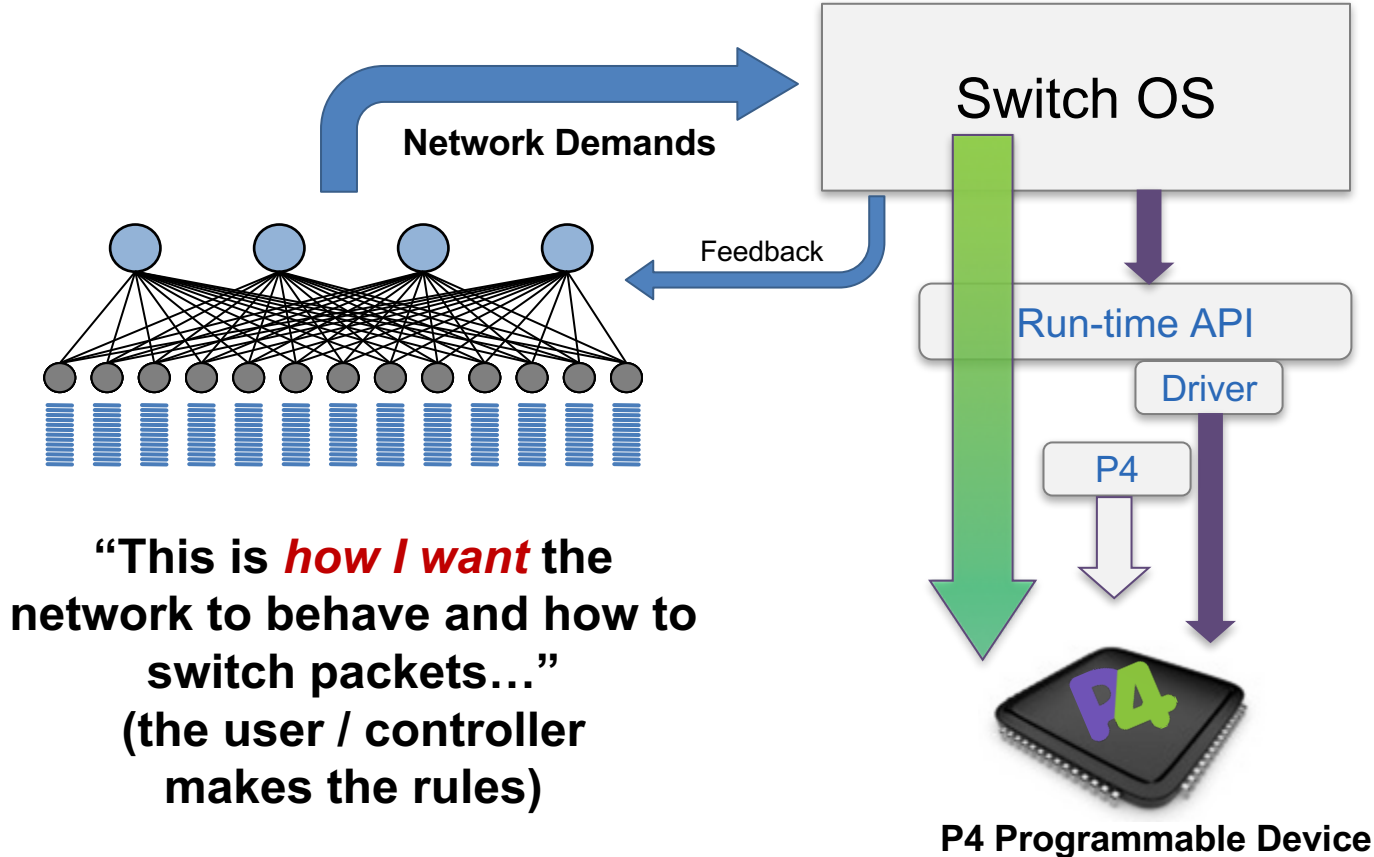
- Data-plane protocol evolution requires changes to standards (12 → 40 OpenFlow match fields)
- Limited interoperability between vendors (OpenFlow / netconf / JSON / XML variants)
- Limited programmability



Status Quo: Bottom-up design



A Better Approach: Top-down design



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 P4₁₄ Draft Specification (v0.9.8)
- **Sep 2014:** P4₁₄ Specification released (v1.0.0)
- **Jan 2015:** P4₁₄ v1.0.1
- **Mar 2015:** P4₁₄ v1.0.2
- **Nov 2016:** P4₁₄ v1.0.3
- **May 2017:** P4₁₄ v1.0.4

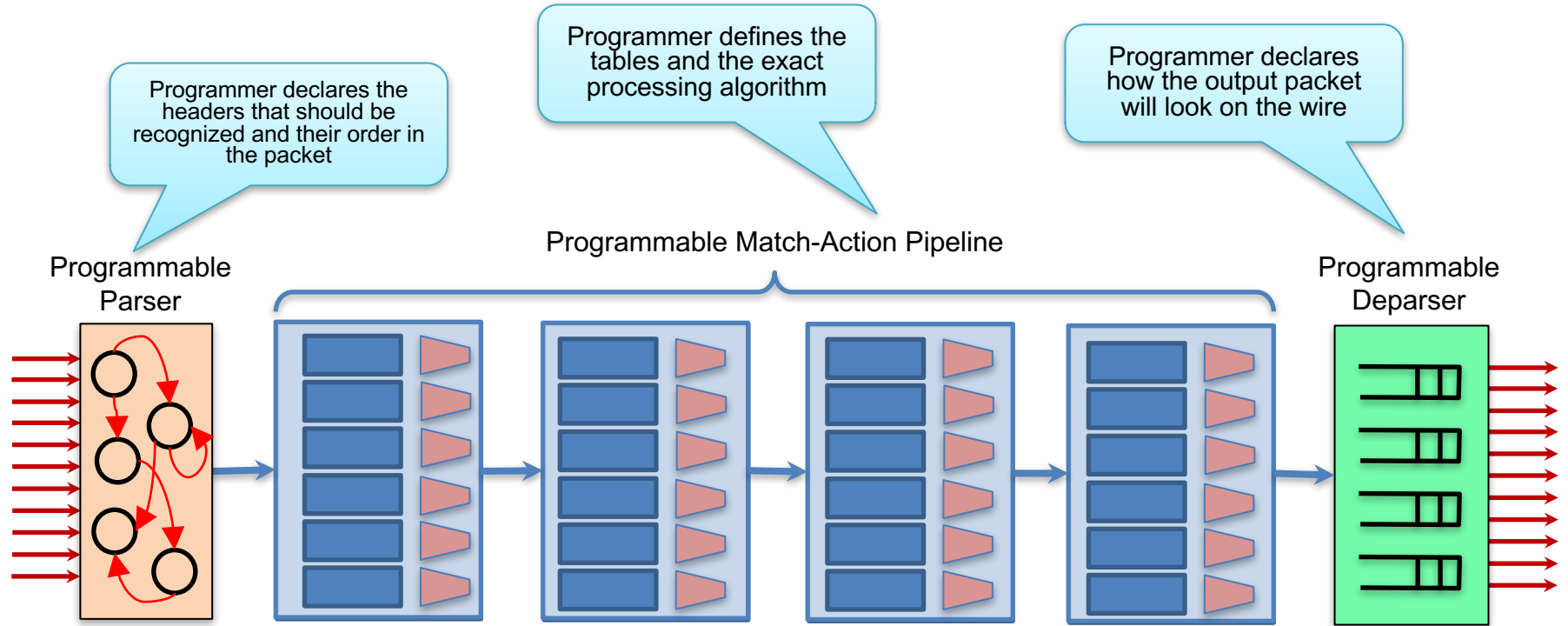
- **Apr 2016:** P4₁₆ – first commits
- **Dec 2016:** First P4₁₆ Draft Specification
- **May 2017:** P4₁₆ Specification released



P4_16 Data Plane Model

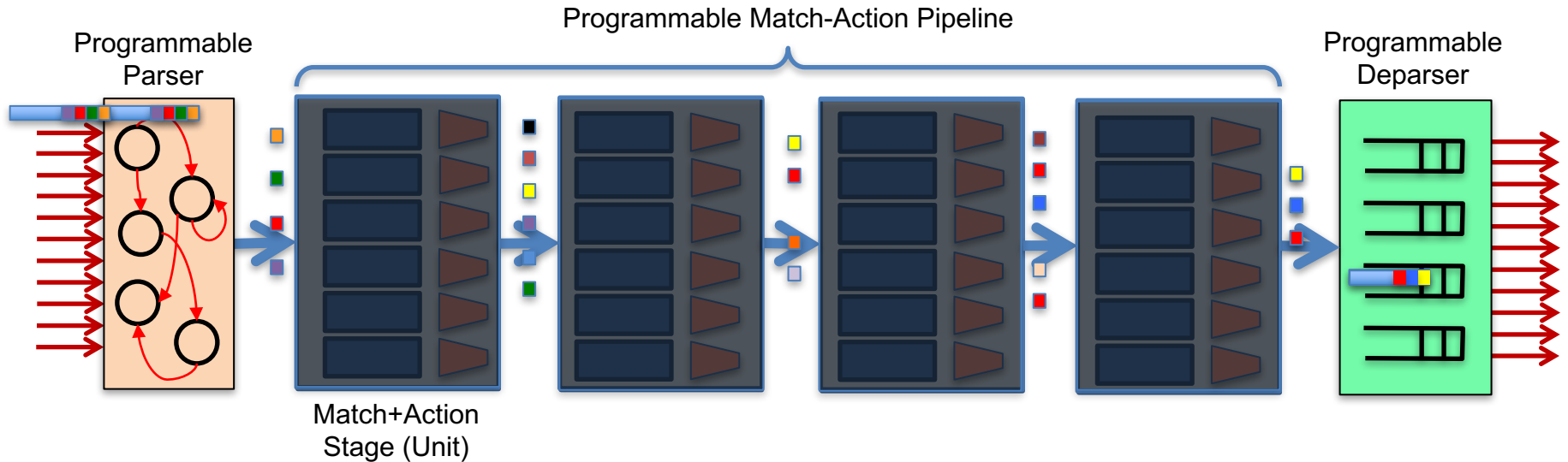


PISA: Protocol-Independent Switch Architecture



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)



P4₁₆ Language Elements

Parsers

State machine,
bitfield extraction

Controls

Tables, Actions,
control flow
statements

Expressions

Basic operations
and operators

Data Types

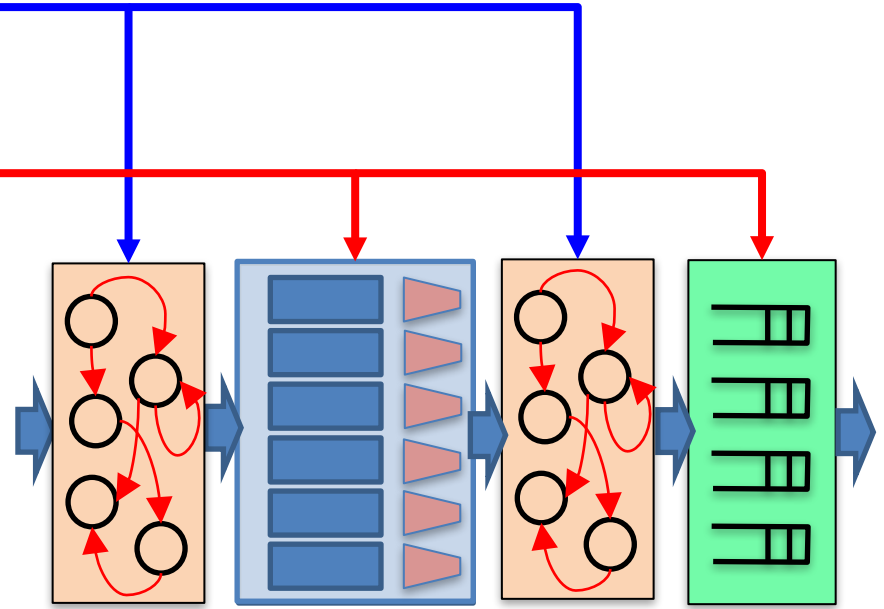
Bistrings, headers,
structures, arrays

Architecture
Description

Programmable blocks
and their interfaces

Extern Libraries

Support for specialized
components

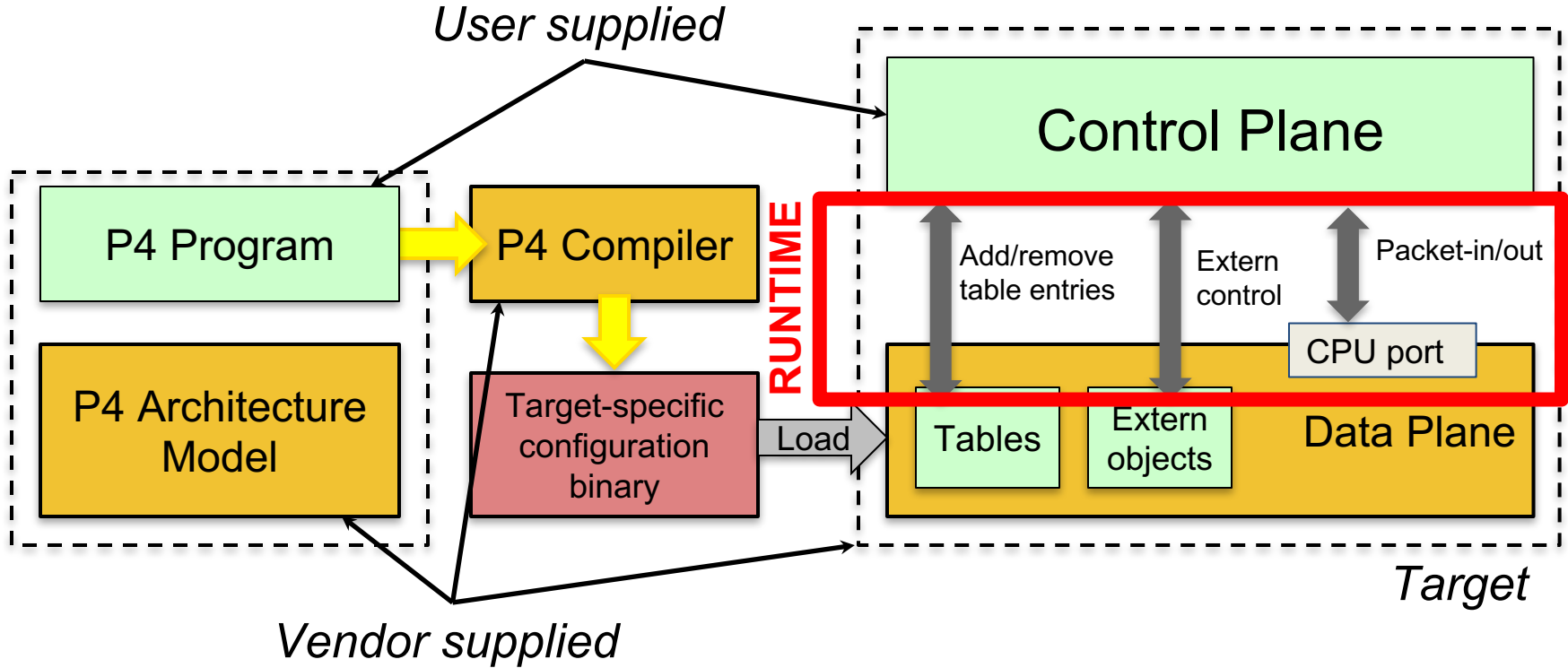


P4_16 Approach

Term	Explanation
P4 Target	An embodiment of a specific hardware implementation
P4 Architecture	Provides an interface to program a target via some set of P4-programmable components, externs, fixed components



Programming a P4 Target



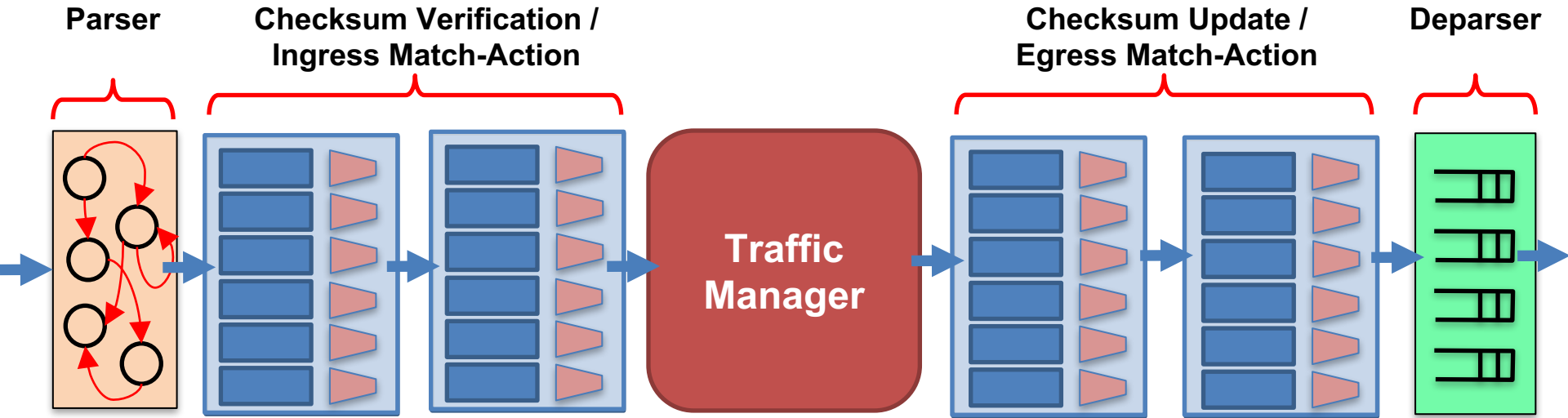
The Exercises

- **Install VM image**
- **We'll be using several software tools pre-installed on the VM**
 - Bmv2: a P4 software switch
 - p4c: the reference P4 compiler
 - Mininet: a lightweight network emulation environment
- **Each directory contains a few scripts**
 - `$ make` : compiles P4 program, execute on Bmv2 in Mininet, populate tables
 - `*.py`: send and receive test packets in Mininet
- **Exercises**
 - Each example comes with an incomplete implementation; your job is to finish it!
 - Look for “TODOs”



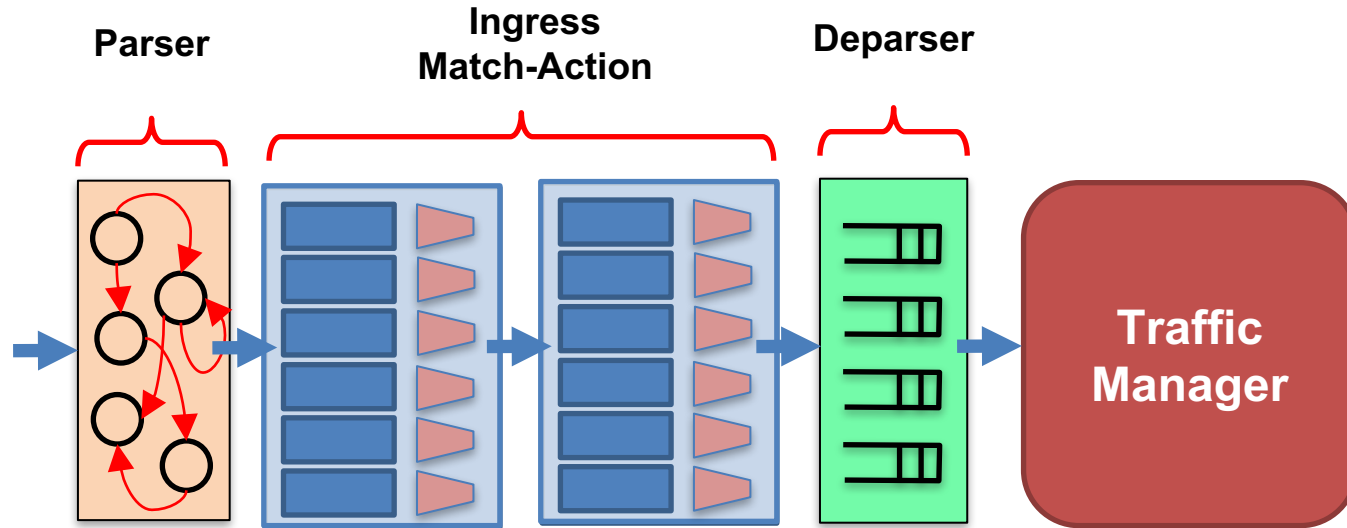
V1Model Architecture

- Implemented on top of Bmv2's `simple_switch` target



SimpleSumeSwitch Architecture

- Implemented on top NetFPGA SUME board



V1Model Standard Metadata

```
struct standard_metadata_t {
    bit<9>  ingress_port;
    bit<9>  egress_spec;
    bit<9>  egress_port;
    bit<32> clone_spec;
    bit<32> instance_type;
    bit<1>  drop;
    bit<16> recirculate_port;
    bit<32> packet_length;
    bit<32> enq_timestamp;
    bit<19> enq_qdepth;
    bit<32> deq_timedelta;
    bit<19> deq_qdepth;
    bit<48> ingress_global_timestamp;
    bit<32> lf_field_list;
    bit<16> mcast_grp;
    bit<1>  resubmit_flag;
    bit<16> egress_rid;
    bit<1>  checksum_error;
}
```

- **ingress_port** - the port on which the packet arrived
- **egress_spec** - the port to which the packet should be sent to
- **egress_port** - the port on which the packet is departing from (read only in egress pipeline)



P4₁₆ Program Template (V1Model)

```
#include <core.p4>
#include <v1model.p4>
/* HEADERS */
struct metadata { ... }
struct headers {
    ethernet_t    ethernet;
    ipv4_t        ipv4;
}
/* PARSER */
parser MyParser(packet_in packet,
                out headers hdr,
                inout metadata meta,
                inout standard_metadata_t smeta) {
    ...
}
/* CHECKSUM VERIFICATION */
control MyVerifyChecksum(in headers hdr,
                        inout metadata meta) {
    ...
}
/* INGRESS PROCESSING */
control MyIngress(inout headers hdr,
                 inout metadata meta,
                 inout standard_metadata_t std_meta) {
    ...
}
```

```
/* EGRESS PROCESSING */
control MyEgress(inout headers hdr,
                 inout metadata meta,
                 inout standard_metadata_t std_meta) {
    ...
}
/* CHECKSUM UPDATE */
control MyComputeChecksum(inout headers hdr,
                          inout metadata meta) {
    ...
}
/* DEPARSER */
control MyDeparser(inout headers hdr,
                  inout metadata meta) {
    ...
}
/* SWITCH */
V1Switch(
    MyParser(),
    MyVerifyChecksum(),
    MyIngress(),
    MyEgress(),
    MyComputeChecksum(),
    MyDeparser()
) main;
```



P4₁₆ Hello World (V1Model)

```
#include <core.p4>
#include <v1model.p4>
struct metadata {}
struct headers {}

parser MyParser(packet_in packet,
  out headers hdr,
  inout metadata meta,
  inout standard_metadata_t standard_metadata) {

  state start { transition accept; }
}

control MyVerifyChecksum(inout headers hdr, inout metadata
meta) {  apply { }  }

control MyIngress(inout headers hdr,
  inout metadata meta,
  inout standard_metadata_t standard_metadata) {
  apply {
    if (standard_metadata.ingress_port == 1) {
      standard_metadata.egress_spec = 2;
    } else if (standard_metadata.ingress_port == 2) {
      standard_metadata.egress_spec = 1;
    }
  }
}
```

```
control MyEgress(inout headers hdr,
  inout metadata meta,
  inout standard_metadata_t standard_metadata) {
  apply { }
}

control MyComputeChecksum(inout headers hdr, inout metadata
meta) {
  apply { }
}

control MyDeparser(packet_out packet, in headers hdr) {
  apply { }
}

V1Switch(
  MyParser(),
  MyVerifyChecksum(),
  MyIngress(),
  MyEgress(),
  MyComputeChecksum(),
  MyDeparser()
) main;
```



P4₁₆ Hello World (V1Model)

```
#include <core.p4>
#include <v1model.p4>
struct metadata {}
struct headers {}

parser MyParser(packet_in packet, out headers hdr,
  inout metadata meta,
  inout standard_metadata_t standard_metadata) {
  state start { transition accept; }
}

control MyIngress(inout headers hdr, inout metadata meta,
  inout standard_metadata_t standard_metadata) {
  action set_egress_spec(bit<9> port) {
    standard_metadata.egress_spec = port;
  }
}

table forward {
  key = { standard_metadata.ingress_port: exact; }
  actions = {
    set_egress_spec;
    NoAction;
  }
  size = 1024;
  default_action = NoAction();
}

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

```
control MyEgress(inout headers hdr,
  inout metadata meta,
  inout standard_metadata_t standard_metadata) {
  apply { }
}

control MyVerifyChecksum(inout headers hdr, inout metadata
  meta) { apply { } }

control MyComputeChecksum(inout headers hdr, inout metadata
  meta) { apply { } }

control MyDeparser(packet_out packet, in headers hdr) {
  apply { }
}

V1Switch( MyParser(), MyVerifyChecksum(), MyIngress(),
  MyEgress(), MyComputeChecksum(), MyDeparser() ) main;
```

Key	Action Name	Action Data
1	set_egress_spec	2
2	set_egress_spec	1

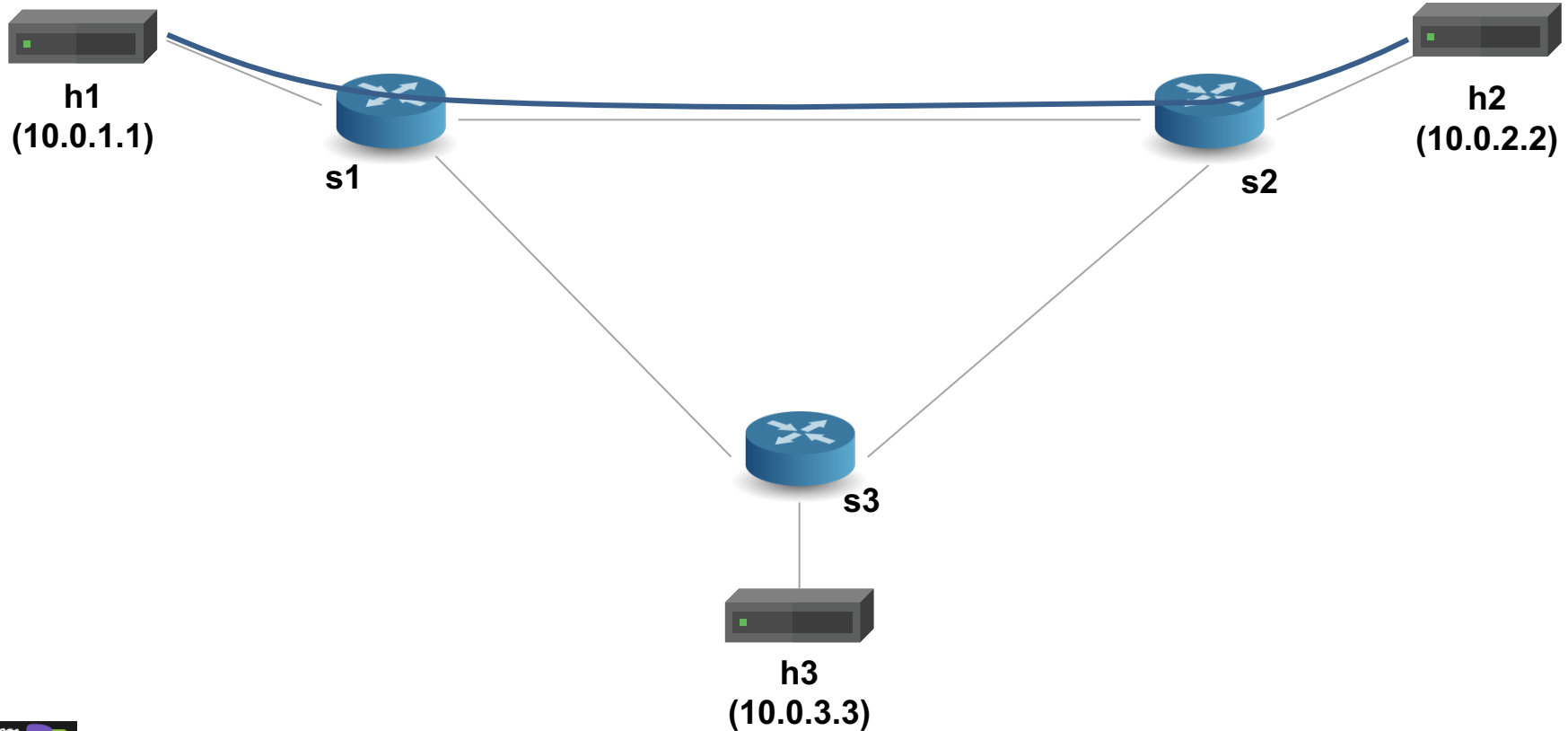


Running Example: Basic Forwarding

- We'll use a simple application as a running example—a basic router—to illustrate the main features of P4₁₆
- **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
 - 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



P4₁₆ 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



P4₁₆ Types (Other Types)

```
/* Architecture */
struct standard_metadata_t {
    bit<9>  ingress_port;
    bit<9>  egress_spec;
    bit<9>  egress_port;
    bit<32> clone_spec;
    bit<32> instance_type;
    bit<1>  drop;
    bit<16> recirculate_port;
    bit<32> packet_length;
    ...
}

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

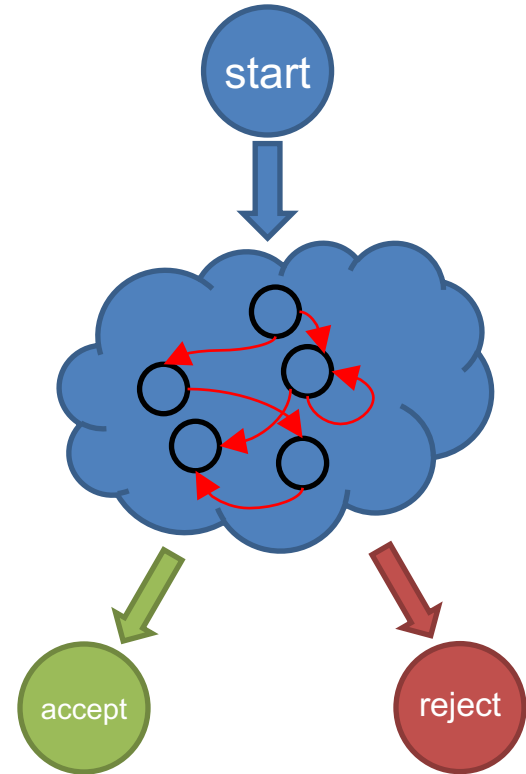
Other useful types

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



P4₁₆ Parsers

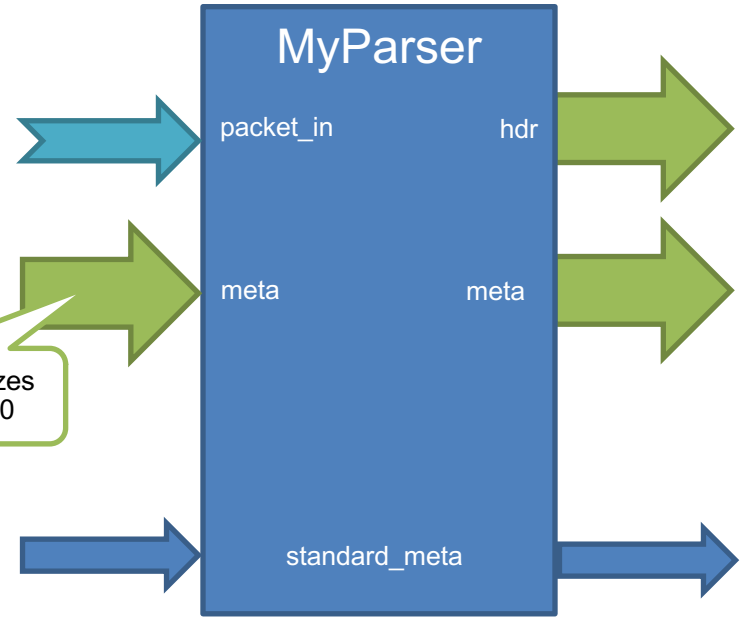
- **Parsers are functions that map packets into headers and metadata, written in a state machine style**
- **Every parser has three predefined states**
 - start
 - accept
 - reject
- **Other states may be defined by the programmer**
- **In each state, execute zero or more statements, and then transition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



Parsers (V1Model)

```
/* 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,
               inout metadata meta,
               inout standard_metadata_t std_meta) {

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



The platform Initializes User Metadata to 0



Select Statement

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

P4₁₆ 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



P4₁₆ 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 metadata meta,  
                 inout standard_metadata_t std_meta) {  
    bit<48> tmp;  
    apply {  
        tmp = hdr.ethernet.dstAddr;  
        hdr.ethernet.dstAddr = hdr.ethernet.srcAddr;  
        hdr.ethernet.srcAddr = tmp;  
        std_meta.egress_spec = std_meta.ingress_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 metadata meta,
                 inout standard_metadata_t std_meta) {

    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);
        std_meta.egress_spec = std_meta.ingress_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: ++



P4₁₆ 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.

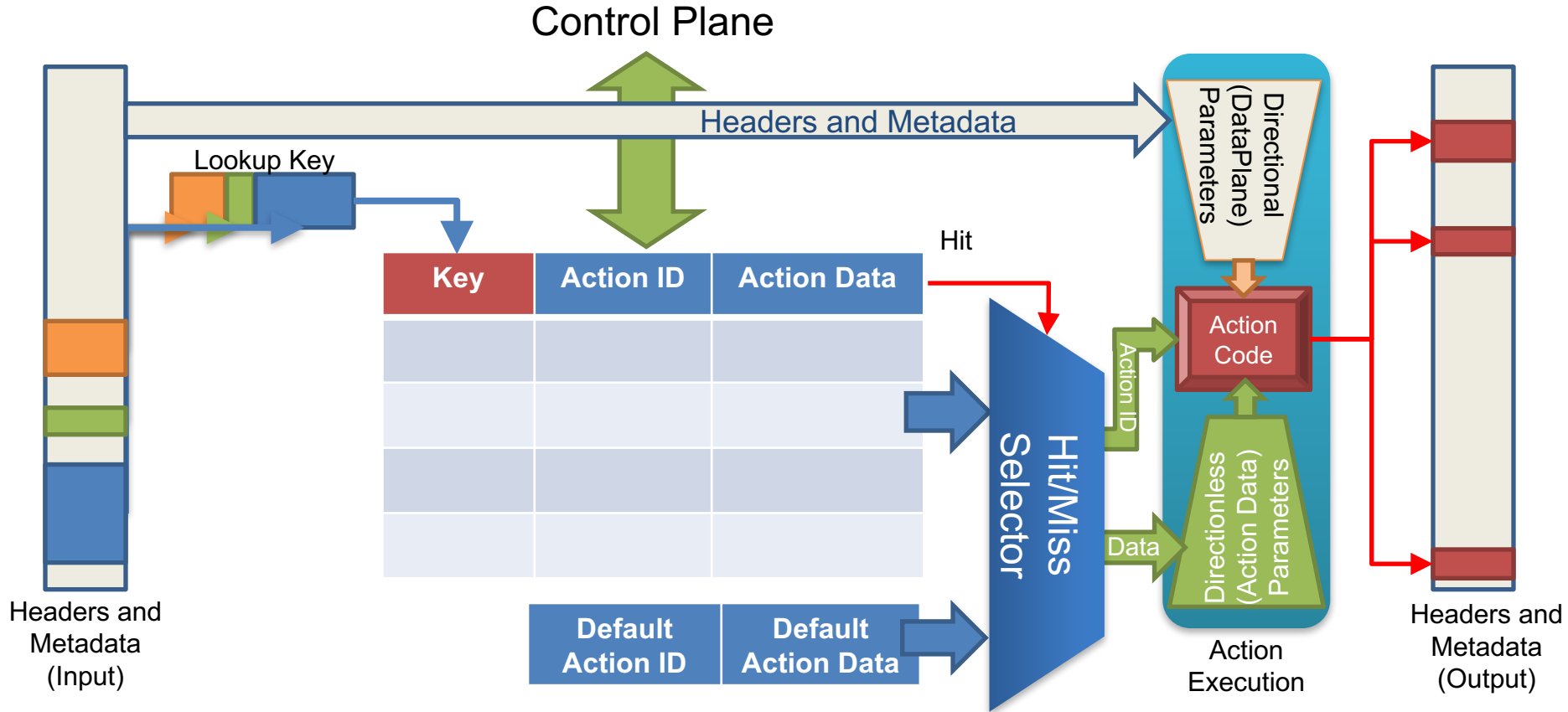
- **Each table contains one or more entries (rules)**

- **An entry contains:**

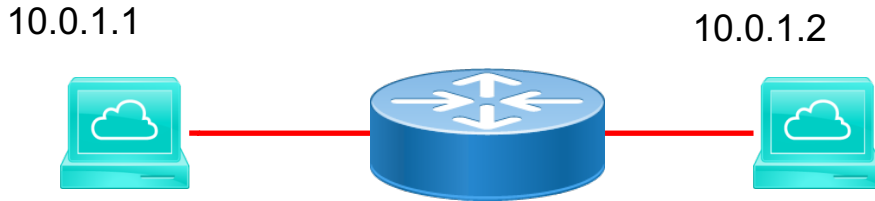
- A specific key to match on
- A **single** action that is executed when a packet matches the entry
- Action data (possibly empty)



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  
}
```

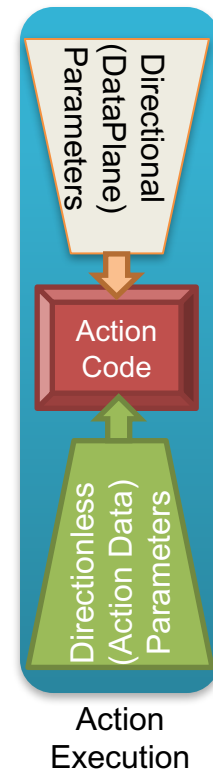
- The type `match_kind` is special in P4
- The standard library (`core.p4`) defines three standard match kinds
 - Exact match
 - Ternary match
 - LPM match
- The architecture (`v1model.p4`) defines two additional match kinds:
 - range
 - selector
- Other architectures may define (and provide implementation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



Applying Tables in Controls

```
control MyIngress(inout headers hdr,  
                 inout metadata meta,  
                 inout standard_metadata_t standard_metadata) {  
  table ipv4_lpm {  
    ...  
  }  
  apply {  
    ...  
    ipv4_lpm.apply();  
    ...  
  }  
}
```



P4₁₆ Deparsing

```
/* From core.p4 */
extern packet_out {
    void emit<T>(in T hdr);
}

/* User Program */
control DeparserImpl(packet_out packet,
                    in headers hdr) {
    apply {
        ...
        packet.emit(hdr.ethernet);
        ...
    }
}
```

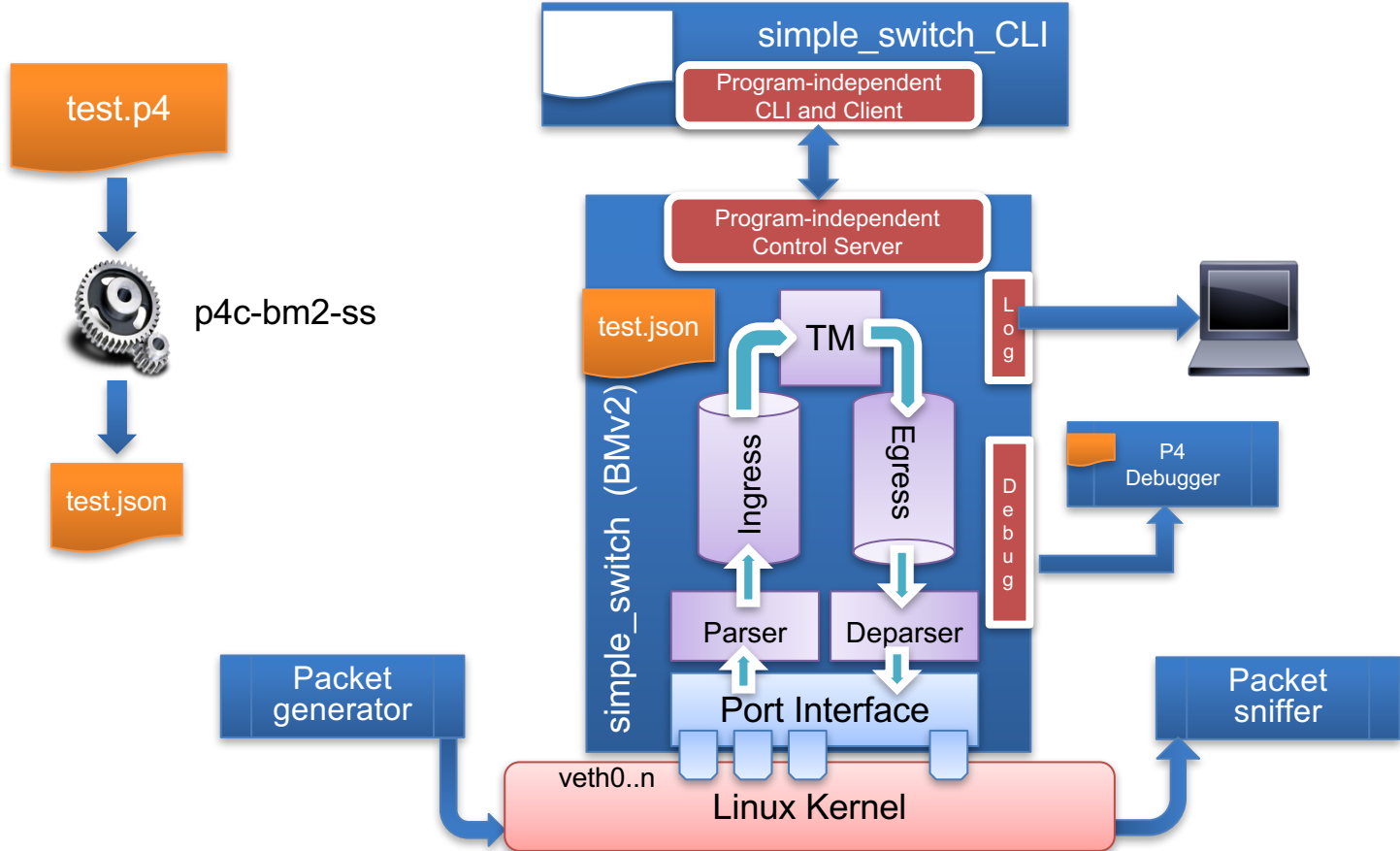
- **Assembles the headers back into a well-formed packet**
- **Expressed as a control function**
 - No need for another construct!
- **packet_out extern is defined in core.p4**: emit(hdr): serializes header if it is valid
- **Advantages:**
 - Makes deparsing explicit...
...but decouples from parsing



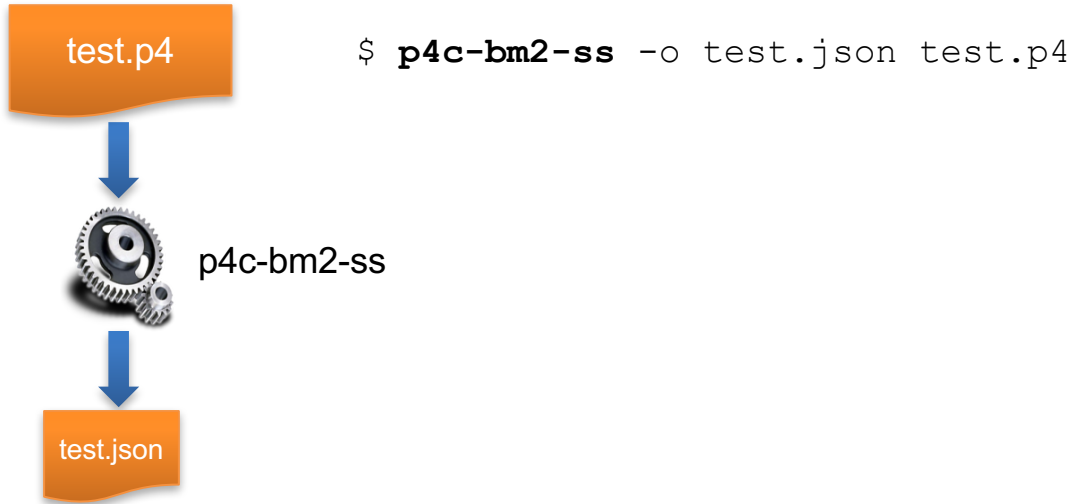
Coding Break



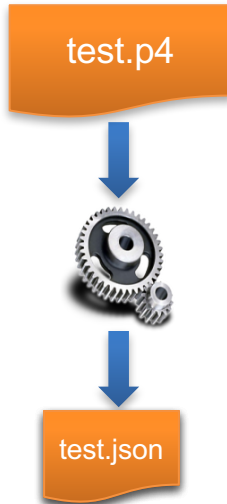
Makefile: under the hood



Step 1: P4 Program Compilation

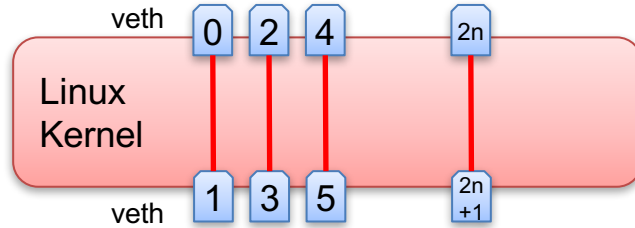


Step 2: Preparing veth Interfaces



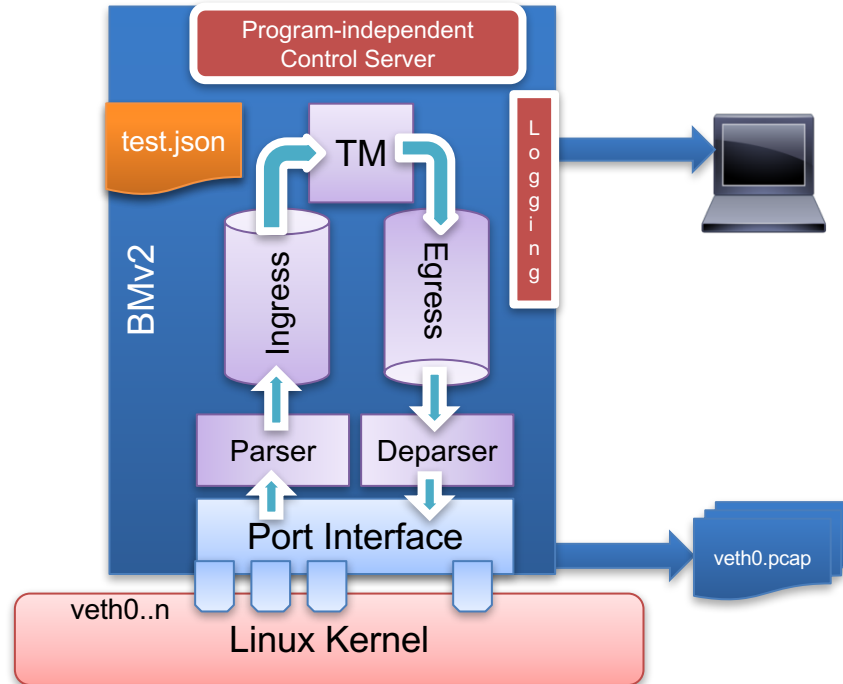
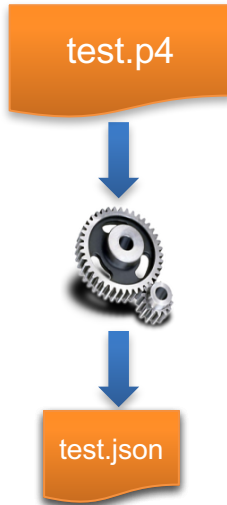
```
$ sudo ~/p4lang/tutorials/examples/veth_setup.sh

# ip link add name veth0 type veth peer name veth1
# for iface in "veth0 veth1"; do
    ip link set dev ${iface} up
    sysctl net.ipv6.conf.${iface}.disable_ipv6=1
    TOE_OPTIONS="rx tx sg tso ufo gso gro lro rxvlan txvlan rxhash"
    for TOE_OPTION in $TOE_OPTIONS; do
        /sbin/ethtool --offload $intf "$TOE_OPTION"
    done
done
```



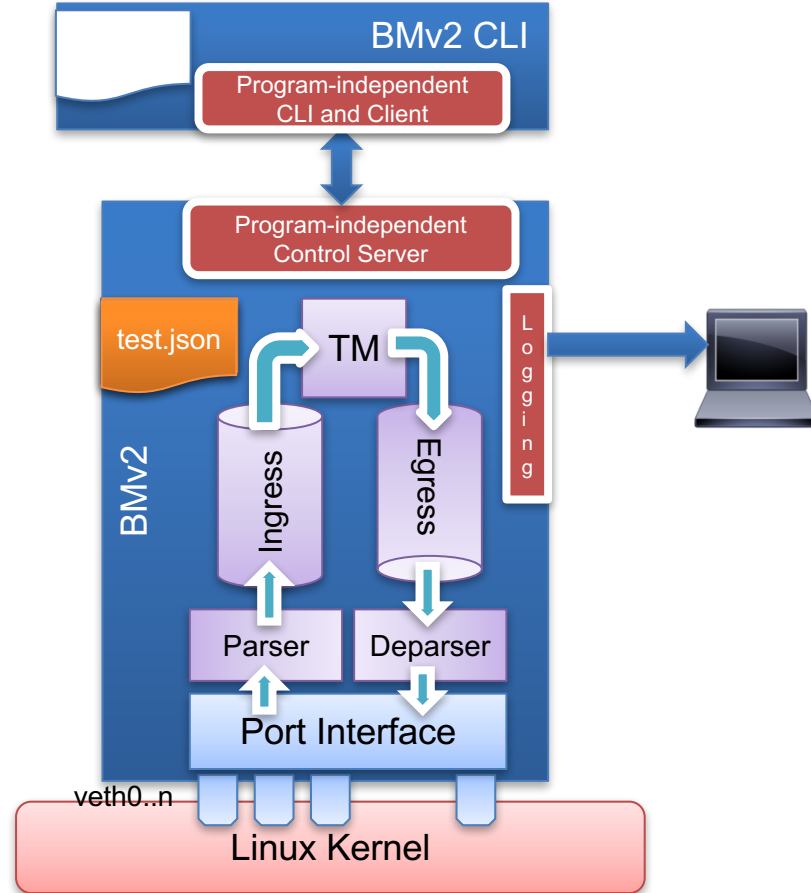
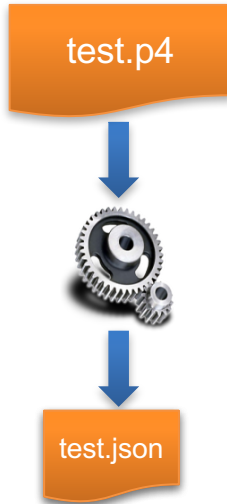
Step 3: Starting the model

```
$ sudo simple_switch --log-console --dump-packet-data 64 \  
-i 0@veth0 -i 1@veth2 ... [--pcap] \  
test.json
```



Step 4: Starting the CLI

```
$ simple_switch_CLI
```



Working with Tables in simple_switch_CLI

RuntimeCmd: **show_tables**

```
m_filter          [meta.meter_tag(exact, 32)]
m_table          [ethernet.srcAddr(ternary, 48)]
```

RuntimeCmd: **table_info m_table**

```
m_table          [ethernet.srcAddr(ternary, 48)]
*****
_nop
[ ]m_action      [meter_idx(32)]
```

RuntimeCmd: **table_dump m_table**

```
m_table:
0: aaaaaaaaaa &&& ffffffff => m_action - 0,
SUCCESS
```

RuntimeCmd: **table_add m_table m_action 01:00:00:00:00:00&&&01:00:00:00:00:00 => 1 0**

```
Adding entry to ternary match table m_table
match key:      TERNARY-01:00:00:00:00:00 &&& 01:00:00:00:00:00
action:         m_action
runtime data:   00:00:00:05
SUCCESS
entry has been added with handle 1
```

RuntimeCmd: **table_delete 1**

Value and mask for ternary matching. No spaces around "&&&"

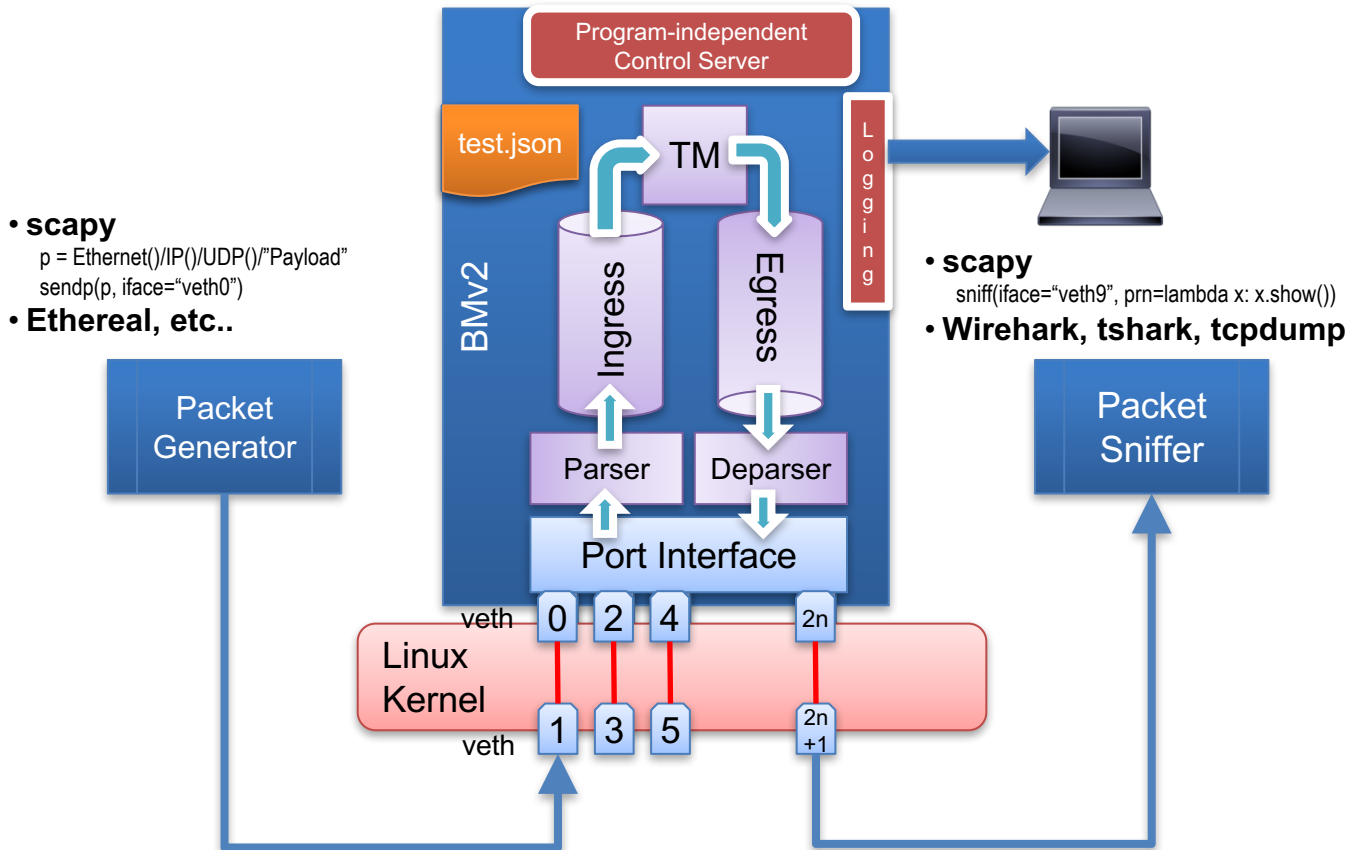
Entry priority

"=>" separates the key from the action data

All subsequent operations use the entry handle



Step 5: Sending and Receiving Packets



Basic Tunneling

- **Add support for basic tunneling to the basic IP router**
- **Define a new header type (`myTunnel`) to encapsulate the IP packet**
- **`myTunnel` header includes:**
 - **`proto_id` : type of packet being encapsulated**
 - **`dst_id` : ID of destination host**
- **Modify the switch to perform routing using the `myTunnel` header**

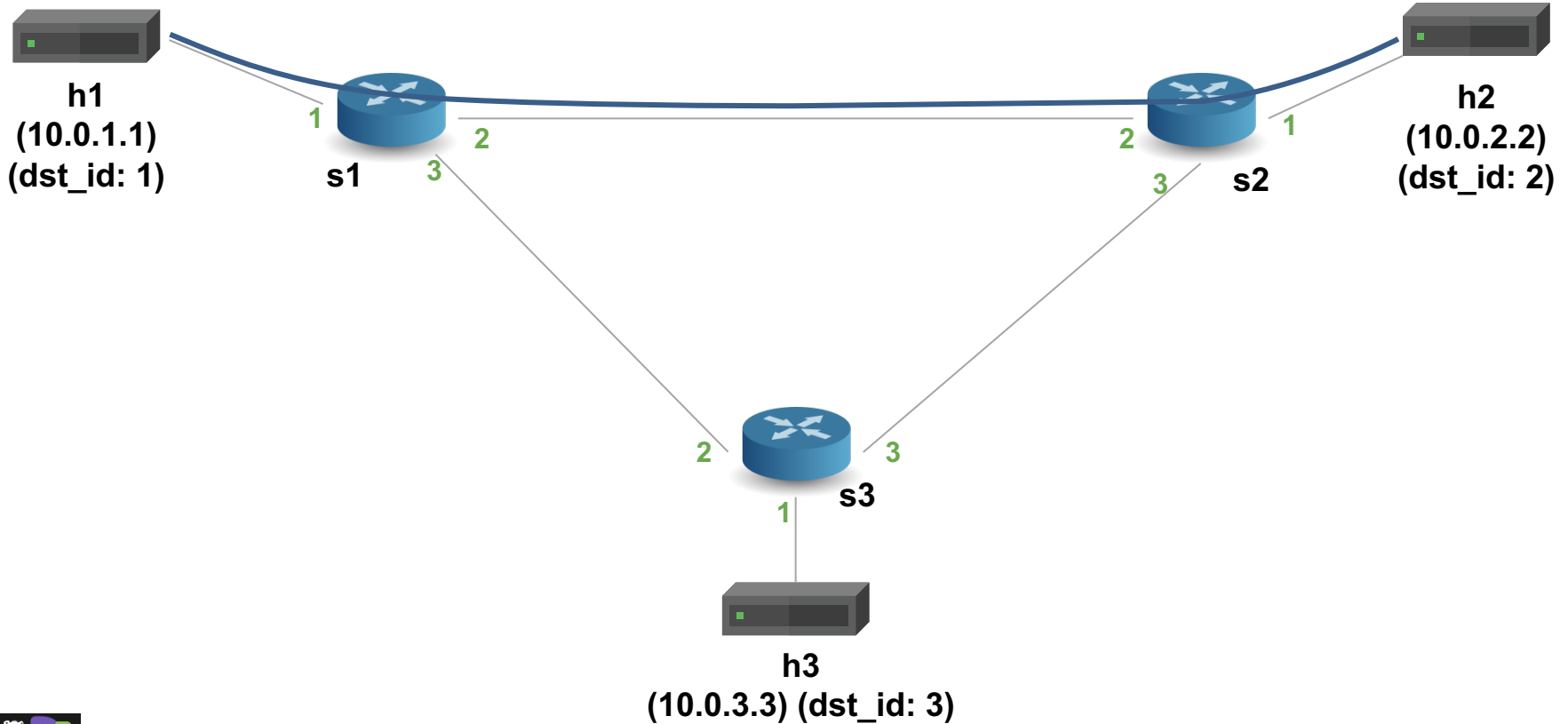


Basic Tunneling TODO List

- **Define myTunnel_t header type and add to headers struct**
- **Update parser**
- **Define myTunnel_forward action**
- **Define myTunnel_exact table**
- **Update table application logic in MyIngress apply statement**
- **Update deparser**
- **Adding forwarding rules**



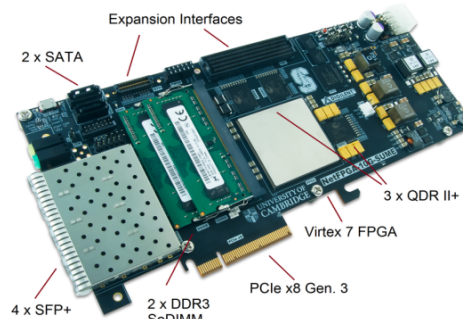
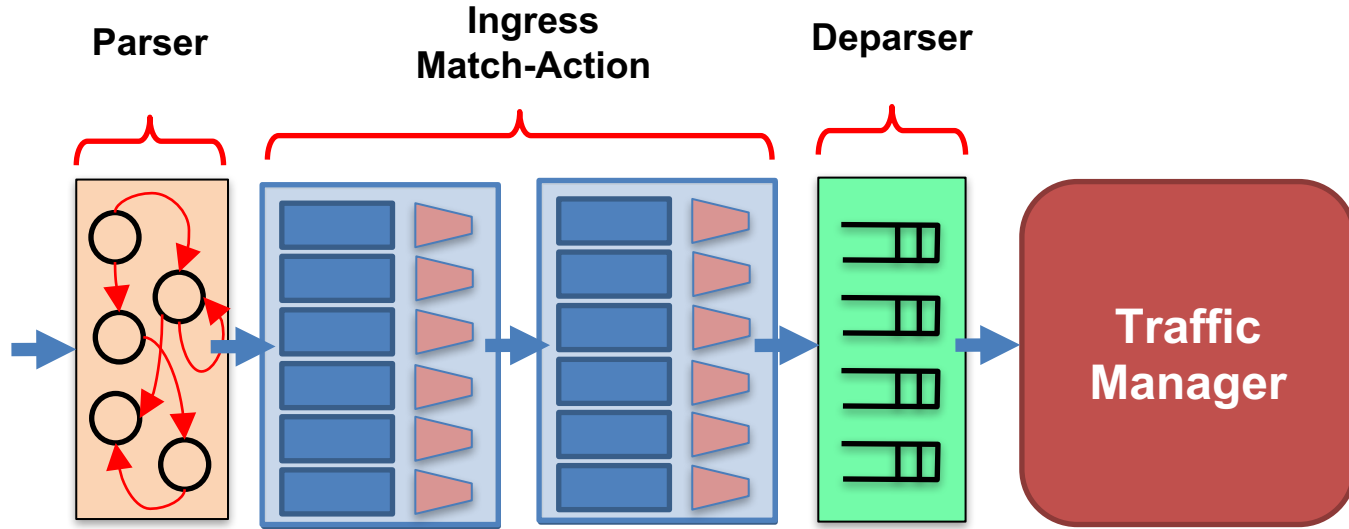
Basic Forwarding: Topology



Coding Break



SimpleSumeSwitch Architecture



Fin!
