P4 on Raspberry PI for Networking Education

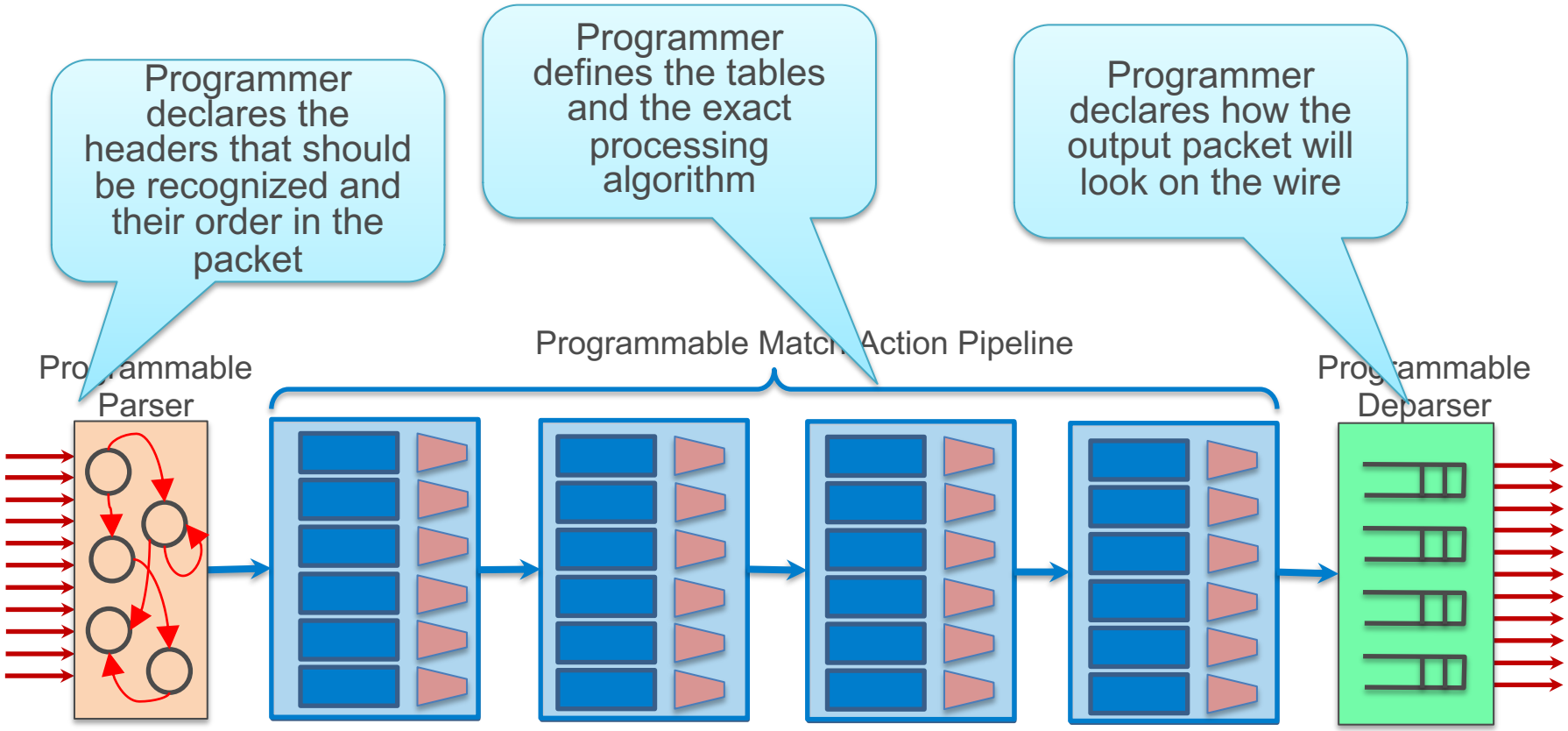
A P4 Education Workgroup Project

Noa Zilberman
Damu Ding
Fernando Ramos
Salvatore Signorello
Robert Soulé

Tutorial organization

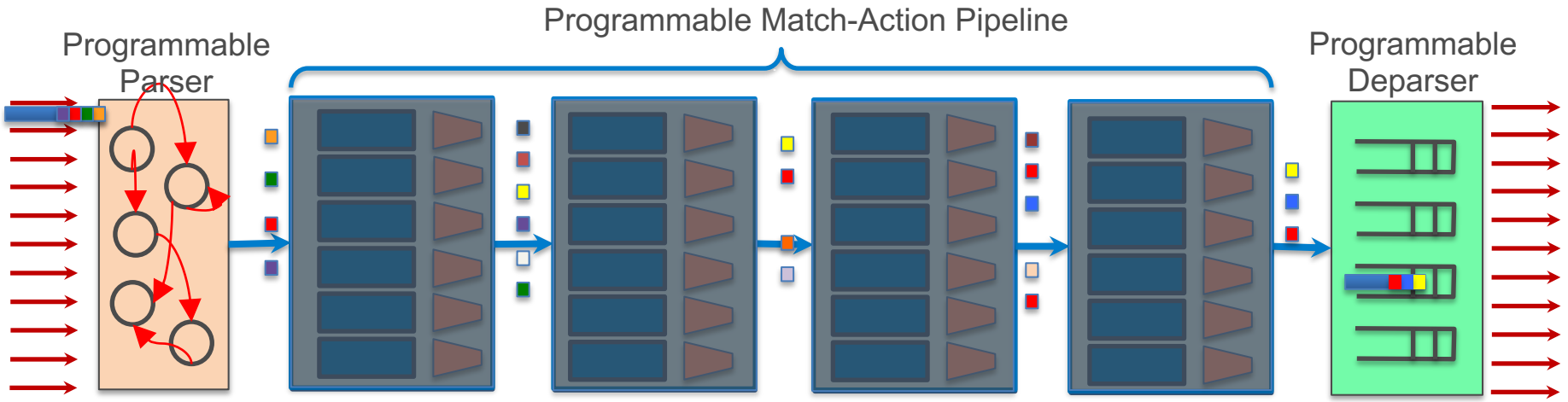
- An introduction to P4
- An introduction to P4Pi
- Hands-on exercises
- Use P4Pi for teaching
- Conclusion

PISA: Protocol-Independent Switch Architecture

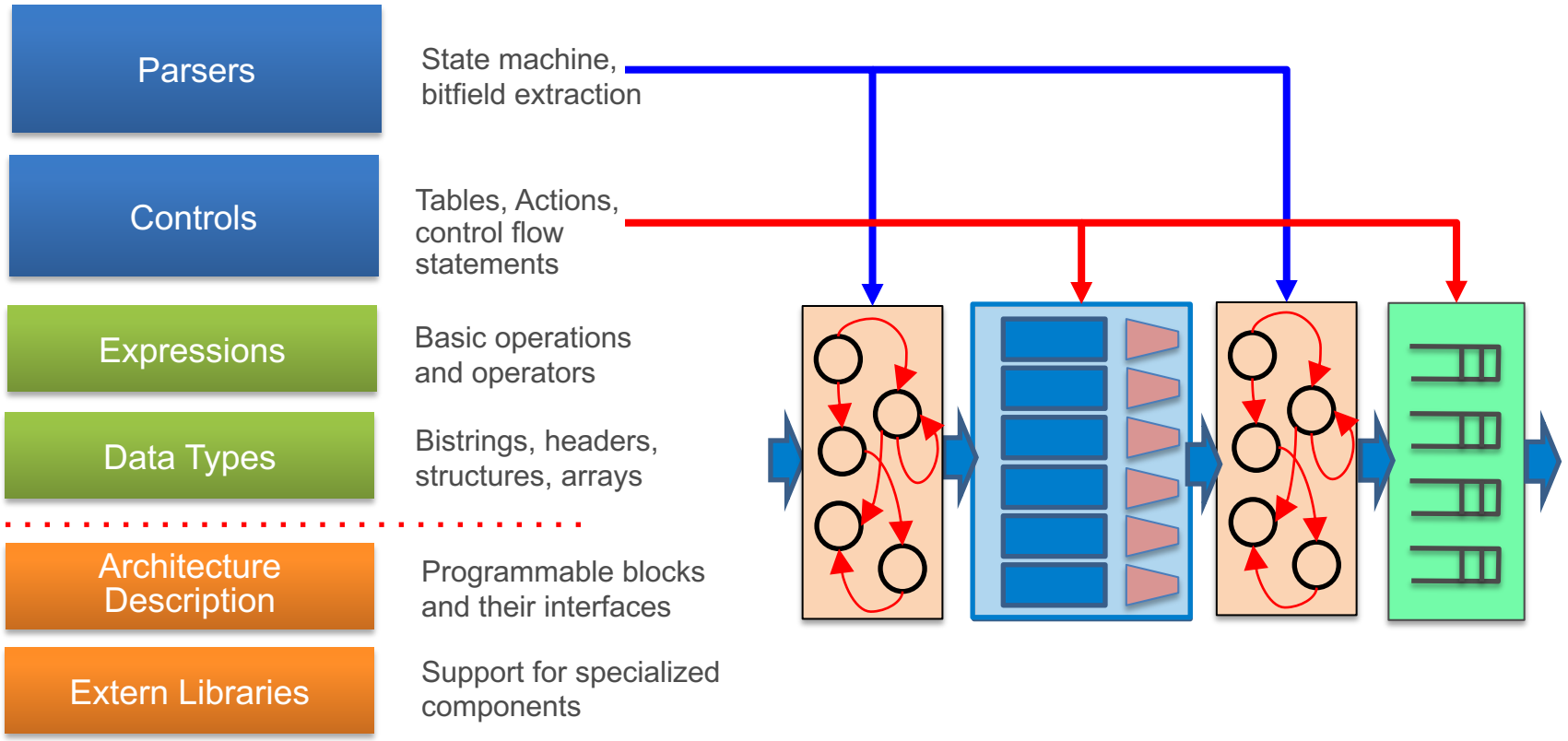


PISA in Action

- 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)



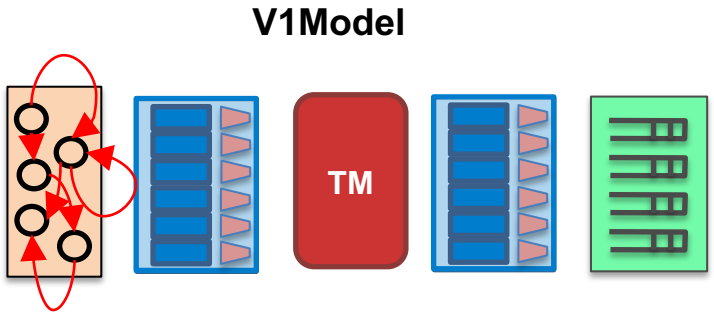
$P4_{16}$ Language Elements

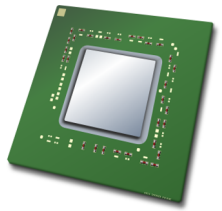


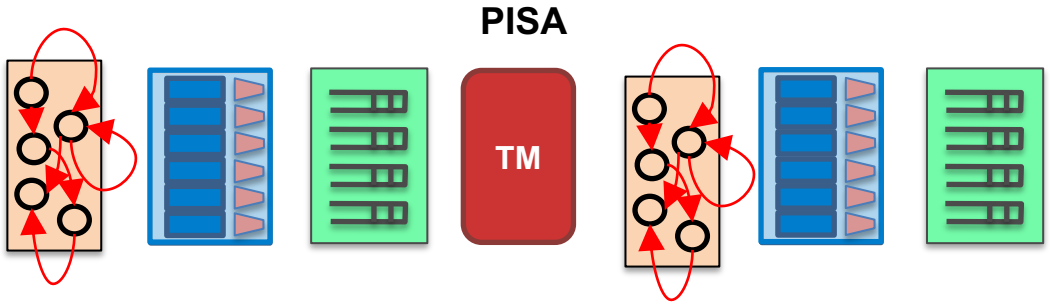
$P4_{16}$ Approach

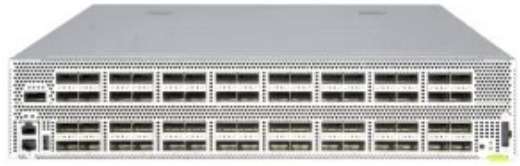
Term	Explanation
P4 Target	An embodiment of a specific hardware implementation
P4 Architecture	A specific set of P4-programmable components, externs, fixed components and their interfaces available to the P4 programmer
P4 Platform	P4 Architecture implemented on a given P4 Target

Example Architectures and Targets

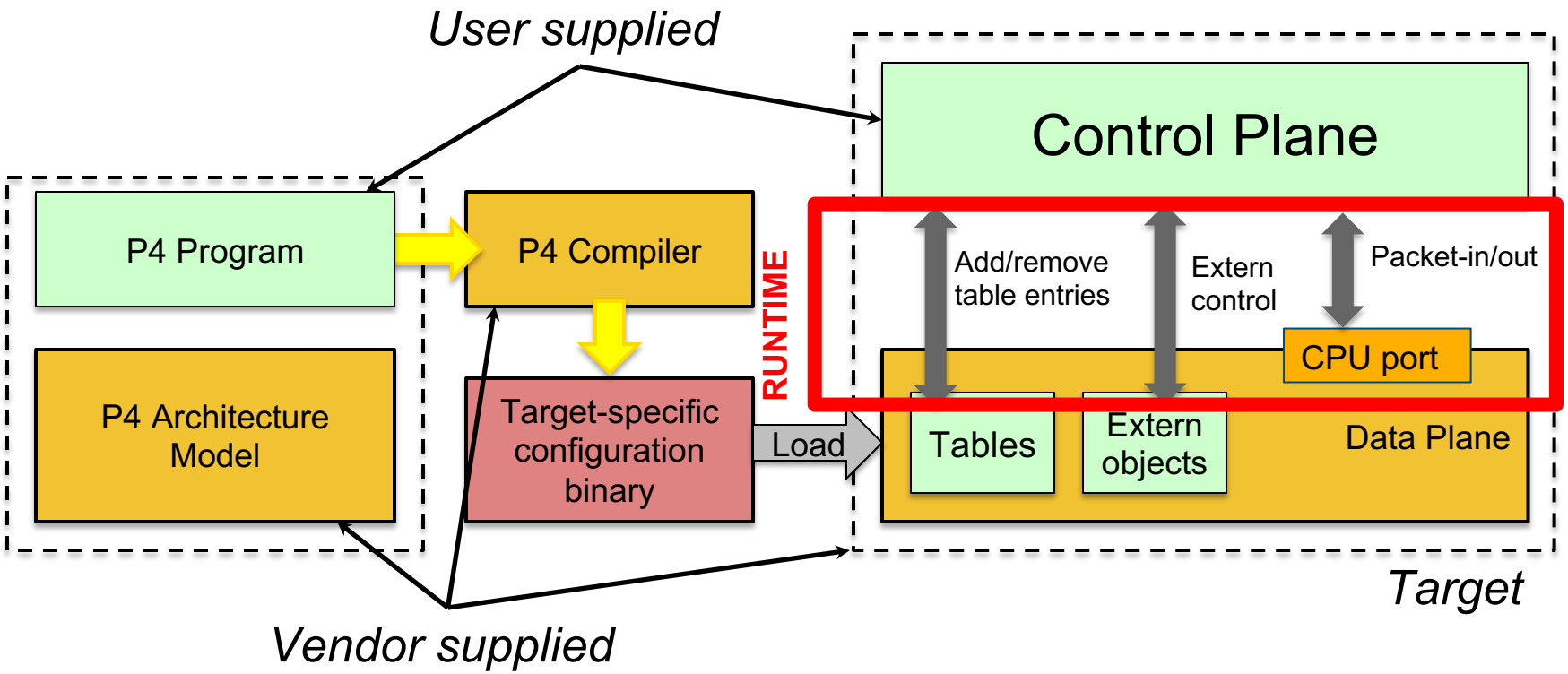






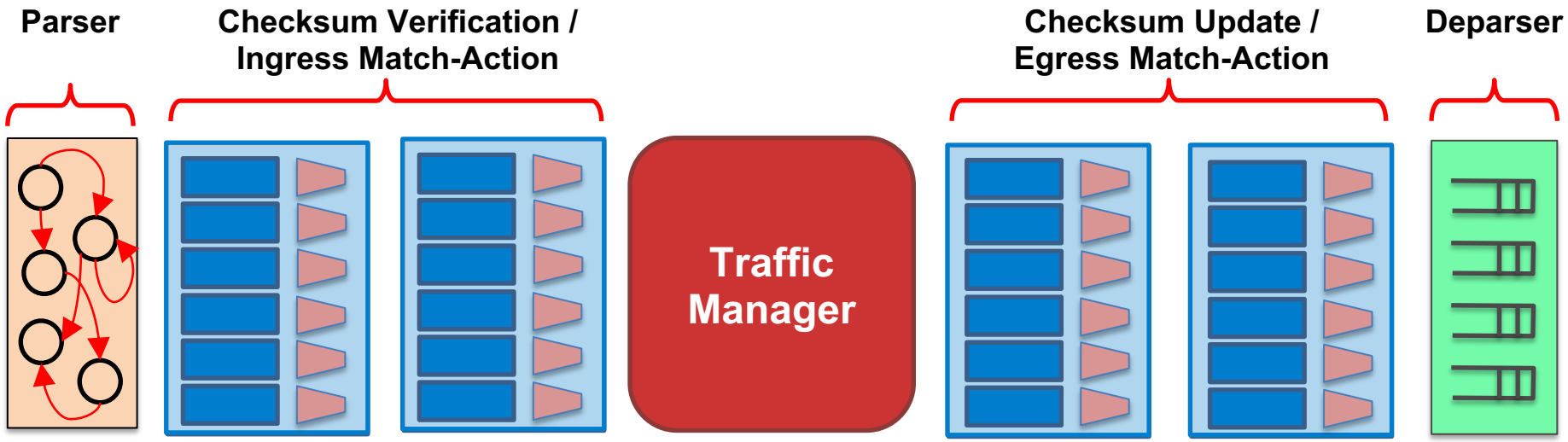


Programming a P4 Target



V1Model Architecture

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



V1model standard metadata

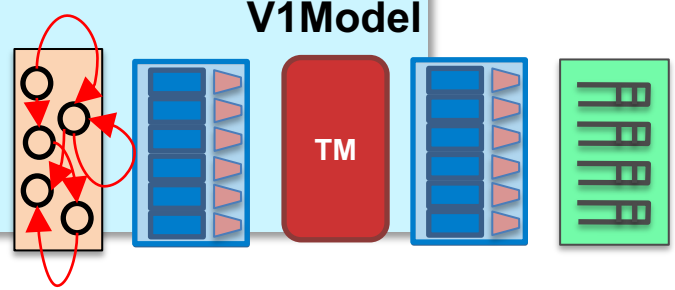
```
struct standard_metadata_t {
    bit<9>  ingress_port;
    bit<9>  egress_spec;
    bit<9>  egress_port;
    bit<32> clone_spec;
    bit<32> instance_type;
    bit<1>  drop;
    bit<16> recirculate_port;
    bit<32> packet_length;
    bit<32> enq_timestamp;
    bit<19> enq_qdepth;
    bit<32> deq_timedelta;
    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)

$P4_{16}$ 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;
```



$P4_{16}$ 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;
```

$P4_{16}$ 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

$P4_{16}$ 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

$P4_{16}$ 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

Intrinsic Metadata

The data that a P4-programmable components can use to interface with the rest of the system

- Defined in the files supplied by the vendor

Declaring and Initializing Variables

```
bit<16>       my_var;
bit<8>        another_var = 5;

const bit<16> ETHERTYPE_IPV4 = 0x0800;
const bit<16> ETHERTYPE_IPV6 = 0x86DD;

ethernet_t    eth;
vlan_tag_t    vtag = {3w0, 1w0, 12w13,
16w0x8847};
```

Safe constants with explicit widths

In $P4_{16}$ you can instantiate variables of both base and derived types

Variables can be initialized
Including the composite types

Constant declarations make for safer code

Infinite width and explicit width constants

P4$_{16}$ Parsers

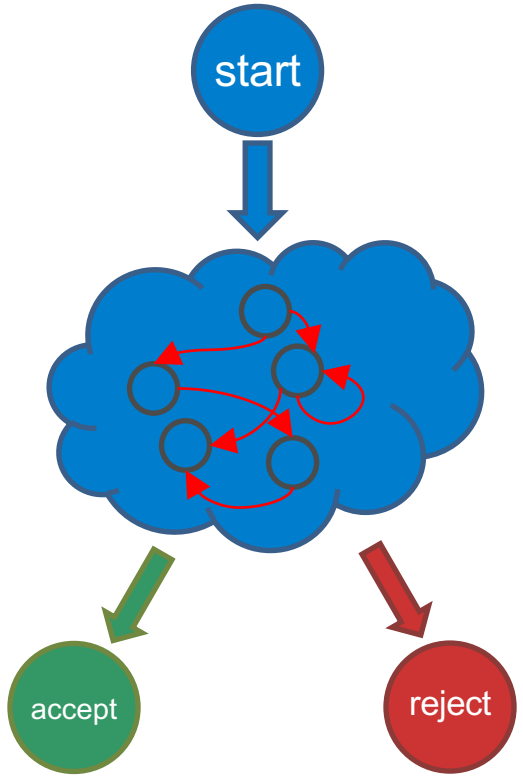
- Parsers are functions that map packets into

  headers and metadata,

  - written in a state machine style

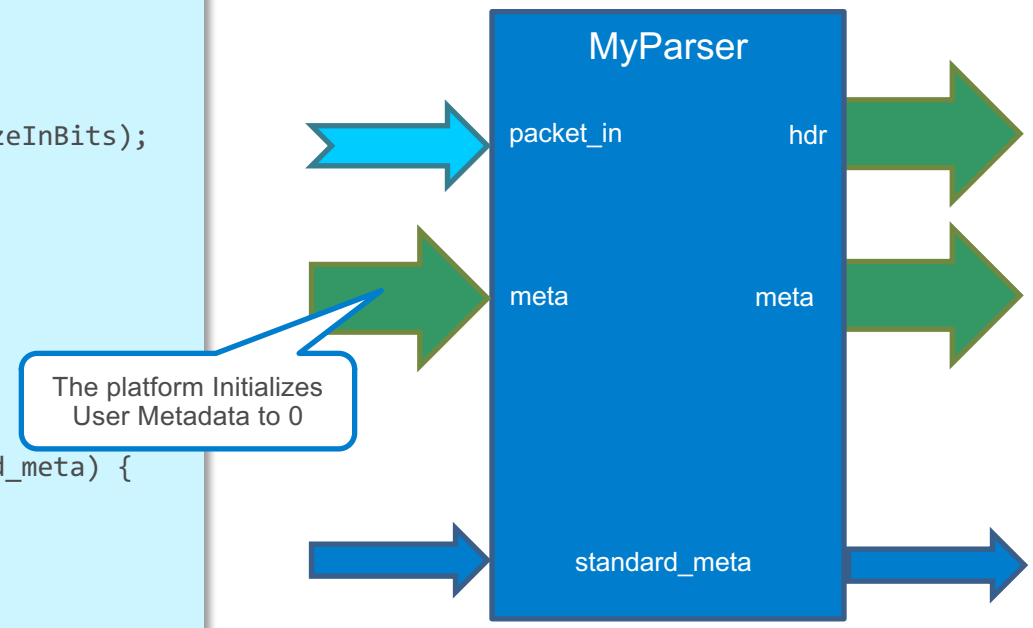
- Every parser has three predefined states
  - `start`
  - `accept`
  - `reject`
- Other states may be defined by the programmer
- In each state, execute zero or more statements,

  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;
  }

}
```



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

P4$_{16}$ Controls

- Similar to C functions (without loops and pointers)
- Can declare variables, create tables, instantiate externs, etc.
- Functionality specified by code in `apply` statement
- Represent all kinds of processing that are expressible as Directed acyclic graph:
  - 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) {
  /* Declarations region */
  bit<48> tmp;


  apply {
    /* Control Flow */
    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

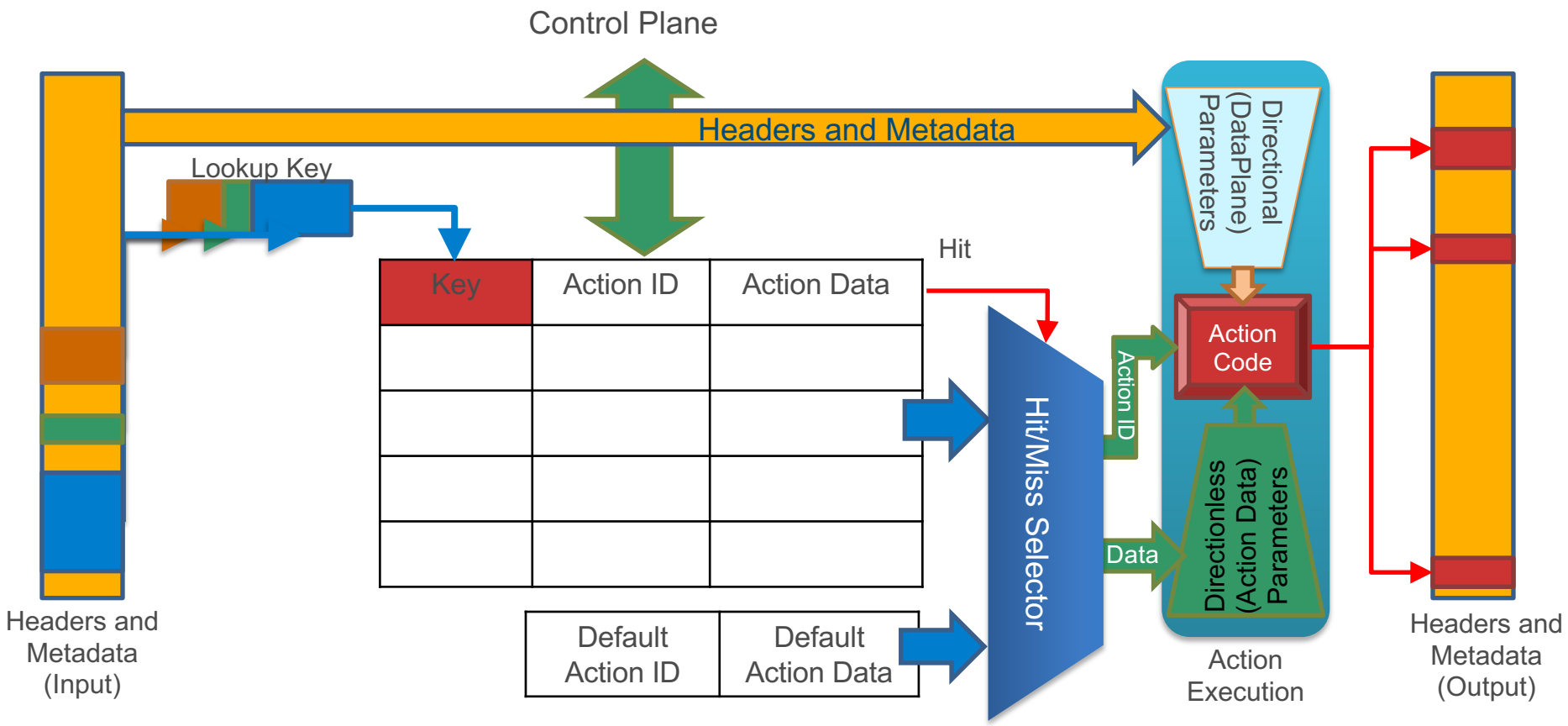
```
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t std_meta) {

  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: ++

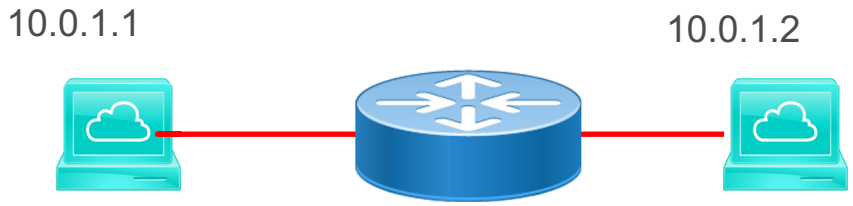
$P4_{16}$ Tables

- 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

Example: IPv4_LPM Table

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

Match Kinds

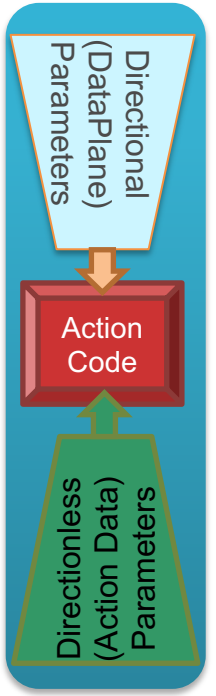
```
/* core.p4 */
match_kind {
    exact,
    ternary,
    lpm
}

/* 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
- P4-SDNet supports `exact, ternary, lpm and direct.`

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

Table Initialization

```
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t standard_metadata) {
  table ipv4_lpm {
    ...
  }
  apply {
     ...
  }
  const entries = {
   /*{ dstAddr, srcAddr, vlan_tag[0].isValid(), vlan_tag[1].isValid() } : action([action_data])*/
          { 48w000000000000, _,  _, _ } : malformed_ethernet(ETHERNET_ZERO_DA);
          { _, 48w000000000000, _, _ } : malformed_ethernet(ETHERNET_ZERO_SA);
          { _, 48w010000000000 &&& 48w010000000000, _, _ } : malformed_ethernet(ETHERNET_MCAST_SA);
          { 48wFFFFFFFFFFFF, _, 0, _ } : broadcast_untagged();
          { 48wFFFFFFFFFFFF, _, 1, 0 } : broadcast_single_tagged();
          { 48wFFFFFFFFFFFF, _, 1, 1 } : broadcast_double_tagged();
          { 48w010000000000 &&& 48w010000000000, _, 0, _ } : multicast_untagged();
          { _, _, 0, _ } : unicast_untagged();
          { _, _, 1, 0 } : unicast_single_tagged();
  }
}
```

$P4_{16}$ Deparsing

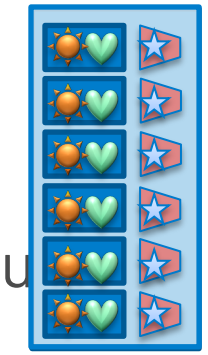
```
/* From core.p4 */
extern packet_out {
  void emit<T>(in T hdr);
}

/* 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

The Need for Externs

- Most platforms contain specialized facilities
  - Differ from vendor to vendor
  - Can't be expressed in the core language
  - Might have control-plane accessible state or configu
- The language should stay the same
  - In $P4_{16}$ all specialized objects use the same interface
- Objects can be used even if their implementation is hidden
  - Through instantiation and method calling



Stateless and Stateful Objects

- Stateless Objects: Reinitialized for each packet
  - Variables (metadata), packet headers, packet_in, packet_out
- Stateful Objects: Keep their state between packets
  - Tables
  - Externs
    - V1 architecture: Counters, Meters, Registers, Parser Value Sets, Selectors, etc.

Counters

```
/* Initialize a new counter*/
counter(512, CounterType.packets_and_bytes) my_counter;

/* Count packets and store them in the corresponding index of the counter*/
my_counter.count(index)

/* Counter types*/
packets
bytes
packets_and_bytes
```

Direct counter

```
control MyIngress(…) {
  direct_counter(CounterType.packets_and_bytes) direct_my_counter;
  table count_table {
    key = {
              standard_metadata.ingress_port: exact
        }
    actions = {
              NoAction;
        }
    counters = direct_my_counter;
  }
  apply {
     ...
     count_table.apply();
     ...
  }
}
```

Meters

```
/* Initialize a new meter*/
meter(1024, MeterType.packets) my_meter;

/* Assign a tag to the corresponding index of the meter*/
my_meter.execute_meter<bit<32>>(meter_index, meta.meter_tag);

/* Meter types*/
packets
bytes
packets_and_bytes
```

Using a meter for rate-limiting

```
control MyIngress(…) {
  meter(1024, MeterType.packets_and_bytes) my_meter;
  action meter_action(bit<32> meter_index){
     my_meter.execute_meter<bit<32>>(meter_index, meter.meter_tag);
  }
  table m_read{
        key = {hdr.ethernet.srcAddr: exact;}
        actions = {meter_action; NoAction;}
  }
  table m_filter{
        key = {meta.meter_tag: exact;}
        actions = {drop; NoAction;}
  }
  apply {
        m_read.apply();
        m_filter.apply();
        }}
```

Direct meter

```
control MyIngress(…) {
  direct_meter(1024, MeterType.packets_and_bytes) my_direct_meter;
  action meter_action(bit<32> meter_index){
     my_meter.read(meta.meter_tag);
  }
  table meter_table {
    key = {hdr.ethernet.srcAddr: exact}
    actions = { meter_action; NoAction;}
    meters = my_direct_meter;
  }
  apply {
     ...
     meter_table.apply();
     ...
  }
}
```

Registers

```
/* Initialize a new register*/
register<bit<32>>(1024) reg;

/* Read the value in the index of the register to a metadata called
meta.read_value*/
reg.read(meta.read_value, index);

/* Write metadata value meta.write_value to the indicated index of the
register */
reg.write(index, meta.write_value);
```

FAQs

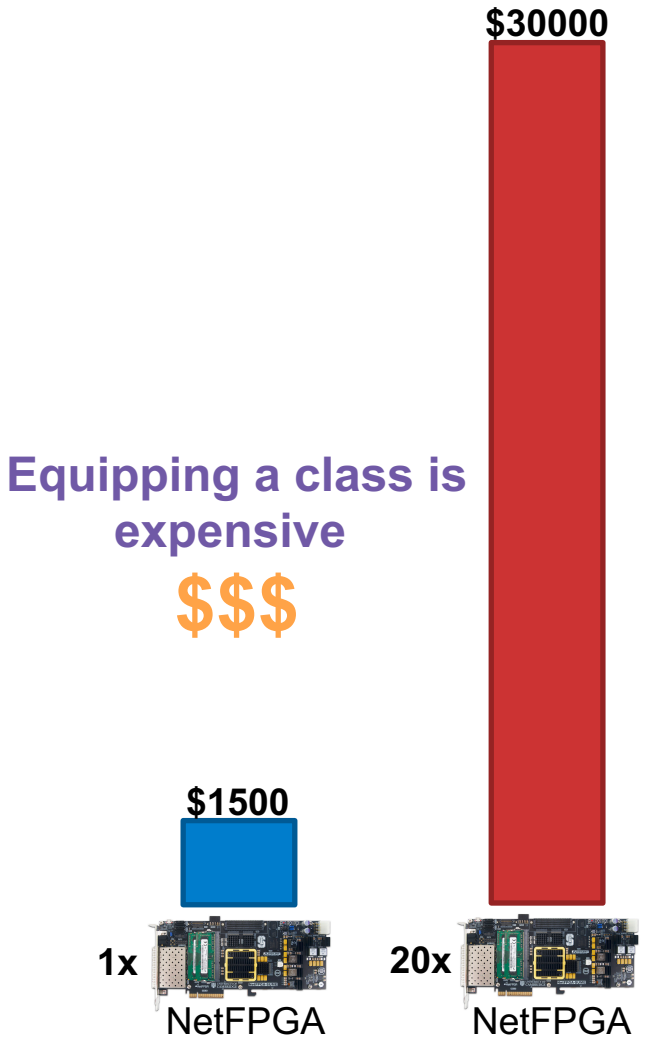
- Can I apply a table multiple times in my P4 Program?
  - No (except via resubmit / recirculate)
- Can I modify table entries from my P4 Program?
  - No (except for direct counters or const entries)
- What happens upon reaching the `reject` state of the parser?
  - Architecture dependent
- How much of the packet can I parse?
  - Architecture dependent

Behavioral model Bmv2

- The most famous P4 software switch target
  - The compiled P4 program by a Bmv2 compiler implements the packet-processing behavior
  - A tool for developing, testing and debugging P4 data planes and control plane software written for them
  - Lacks in performance

Teaching Networking Using P4 + Hardware - Today

- **P4 programmable switch-ASIC**
  - Too expensive for many (>1000's of $)
  - Closed / partially closed source
- **SmartNICs**
  - Cost $1000-$2000
  - Closed source
  - Requires micro-architecture knowledge
- **NetFPGA**
  - Cost ~$1500
  - Open-source
  - Requires FPGA design knowledge



What Is The Ideal Platform For Teaching?

- **Low cost**
  - Less than $100
- **Easy to learn**
  - And easy to use
- **Availability**
  - Worldwide, in-stock
  - Long term support
- **Open-source**
- **Training resources available**
- **Wireless + Wired connectivity**
  - Students can use their laptops
  - ...or existing lab machines
- **Network performance**
  - "Home" level

P4 on Raspberry PI

RaspberryPi 4 Model B
Low-cost platform
On-board WiFi

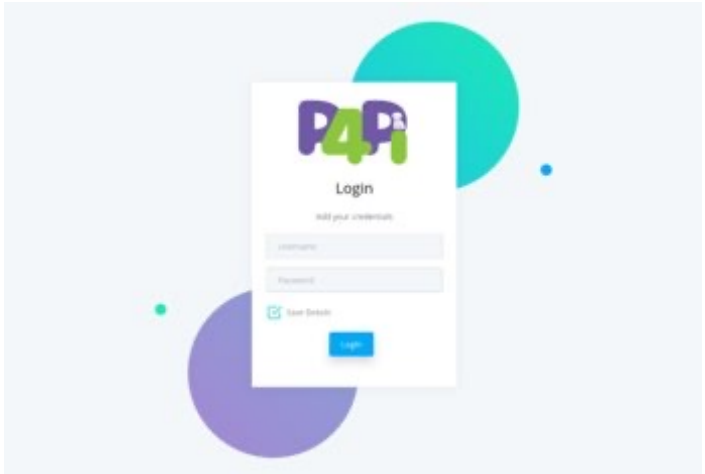
+

Bmv2

P4 on Raspberry PI

Raspberry Pi 4 model B

- 1.5GHz quad-core CPU
- 2.4 GHz and 5.0 GHz wireless
- 1x Gigabit Ethernet
- 2x USB 3.0; 2x USB 2.0



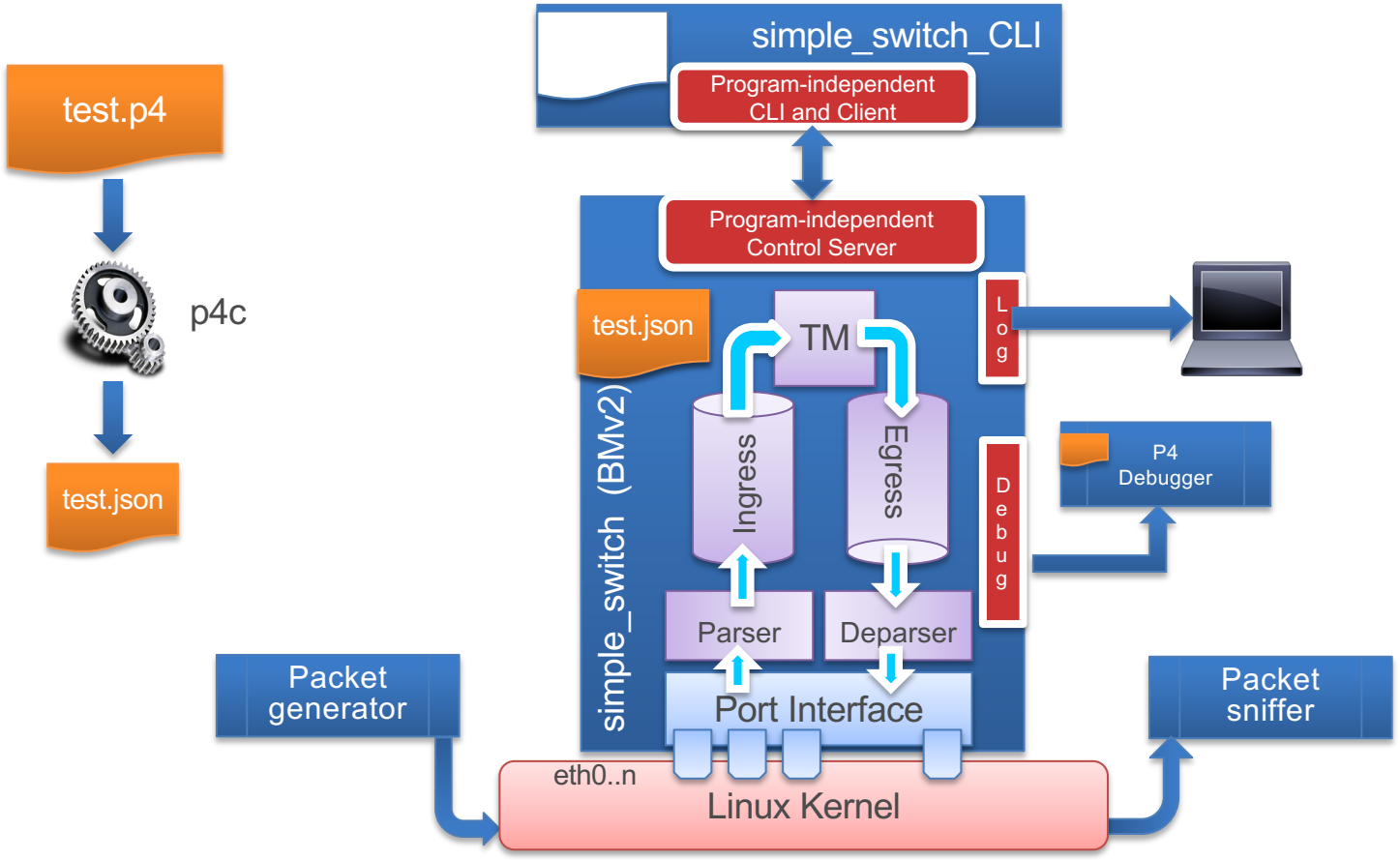
Web Interface

SSH

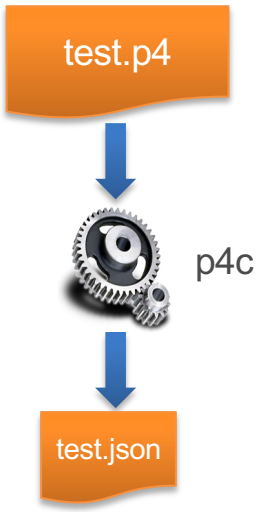
P4PI Is More Than a Hardware Platform

- **Training Materials**
  - Tutorials, exercises etc.
- **P4PI Repository**
  - P4PI source code, wiki, tools, reference programs.
- **Community engagement**

Workflow


test.p4
p4c
test.json
simple_switch_CLI
Program-independent CLI and Client
Program-independent Control Server
test.json
TM
Ingress
Egress
Parser
Deparser
Port Interface
simple_switch (BMv2)
Log
Debug
P4 Debugger
Packet generator
Packet sniffer
eth0..n
Linux Kernel

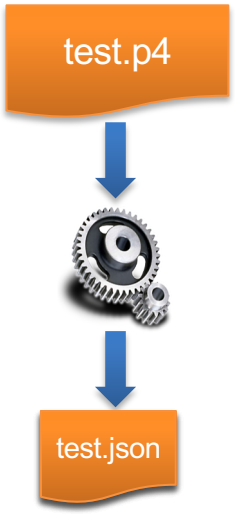
Step 1: P4 Program Compilation

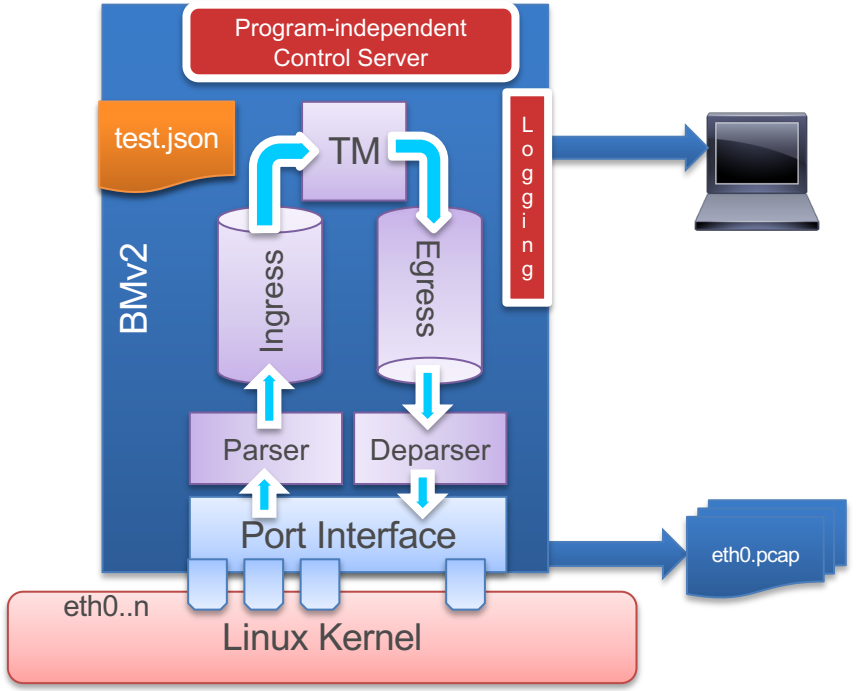



```
$ p4c --target bmv2 --arch v1model
--std p4-16 test.p4
```

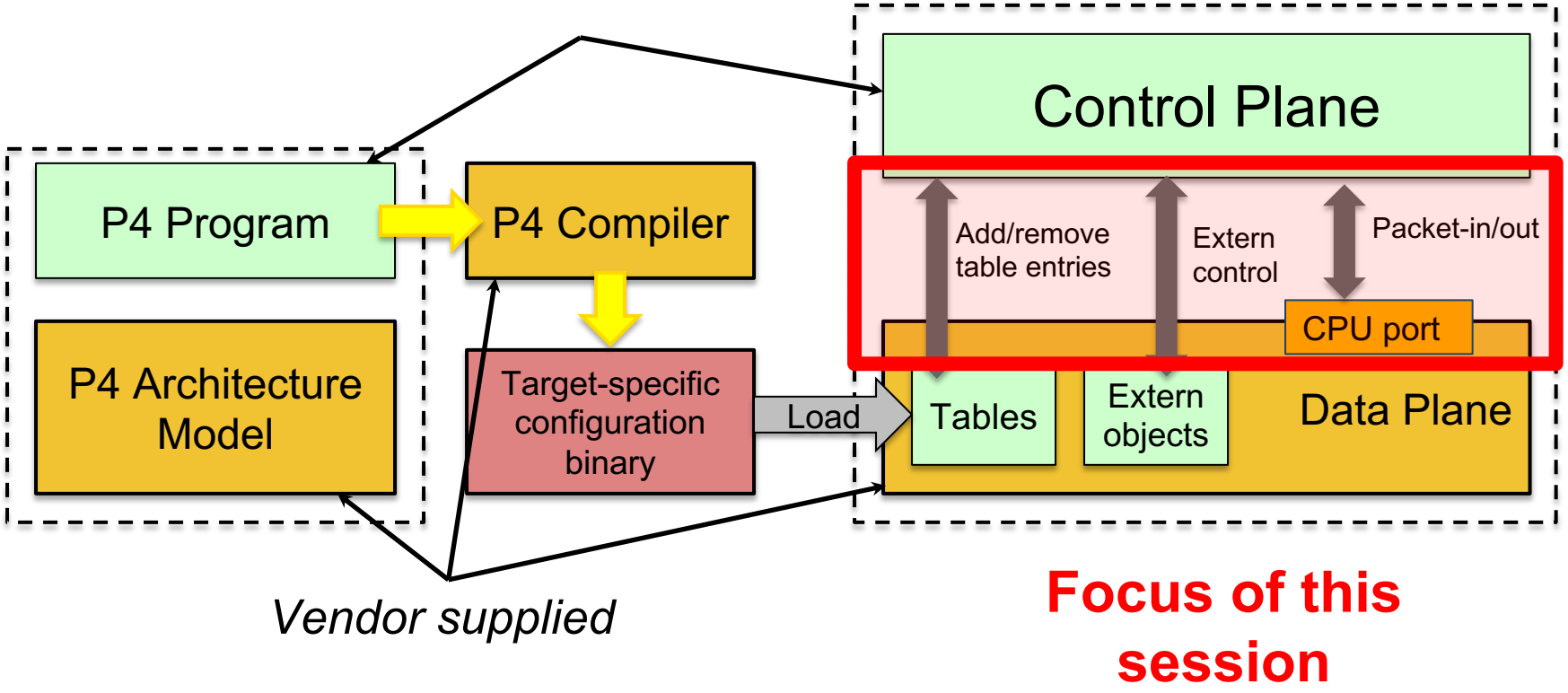
Step 2: Run your program

```
$ sudo simple_switch -i 0@eth0 test.json
```





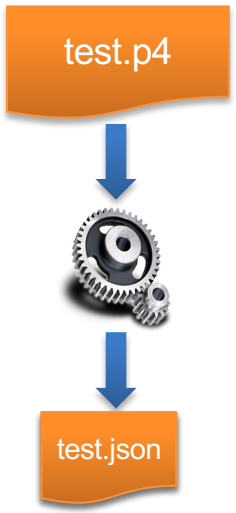
Runtime control of P4 data plane

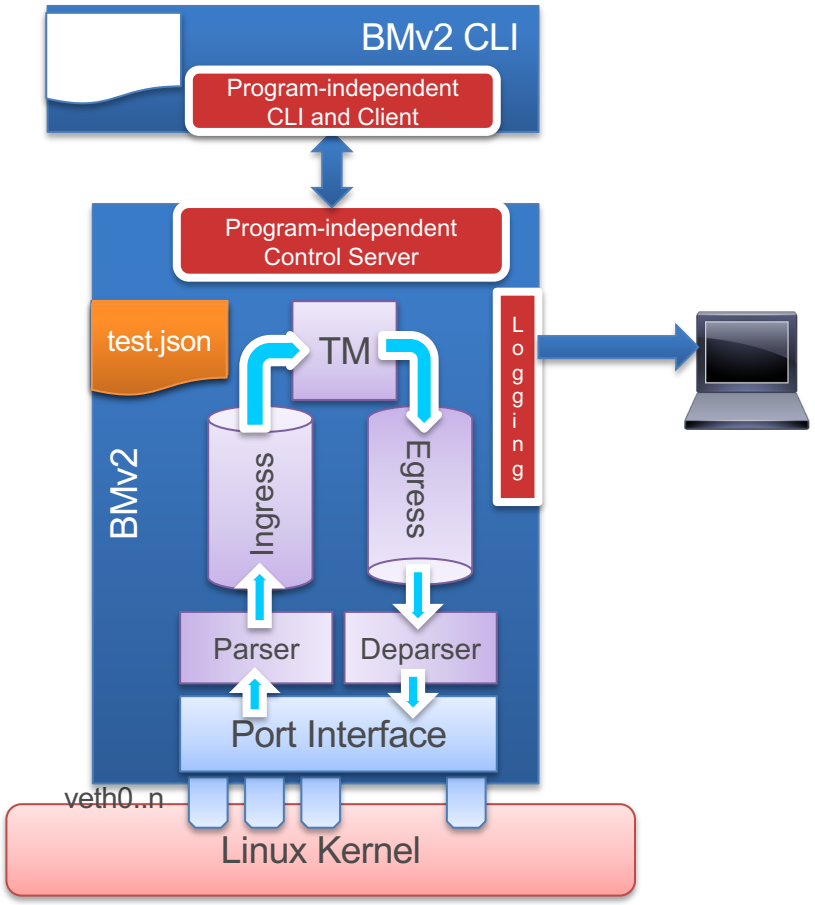


- **BMv2 CLI**
  - Program-independent, but target-specific -- control plane not portable!

Step 3: Start the CLI

```
$ simple_switch_CLI
```





Use 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: aaaaaaaaaaaa &&& ffffffffffff => 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

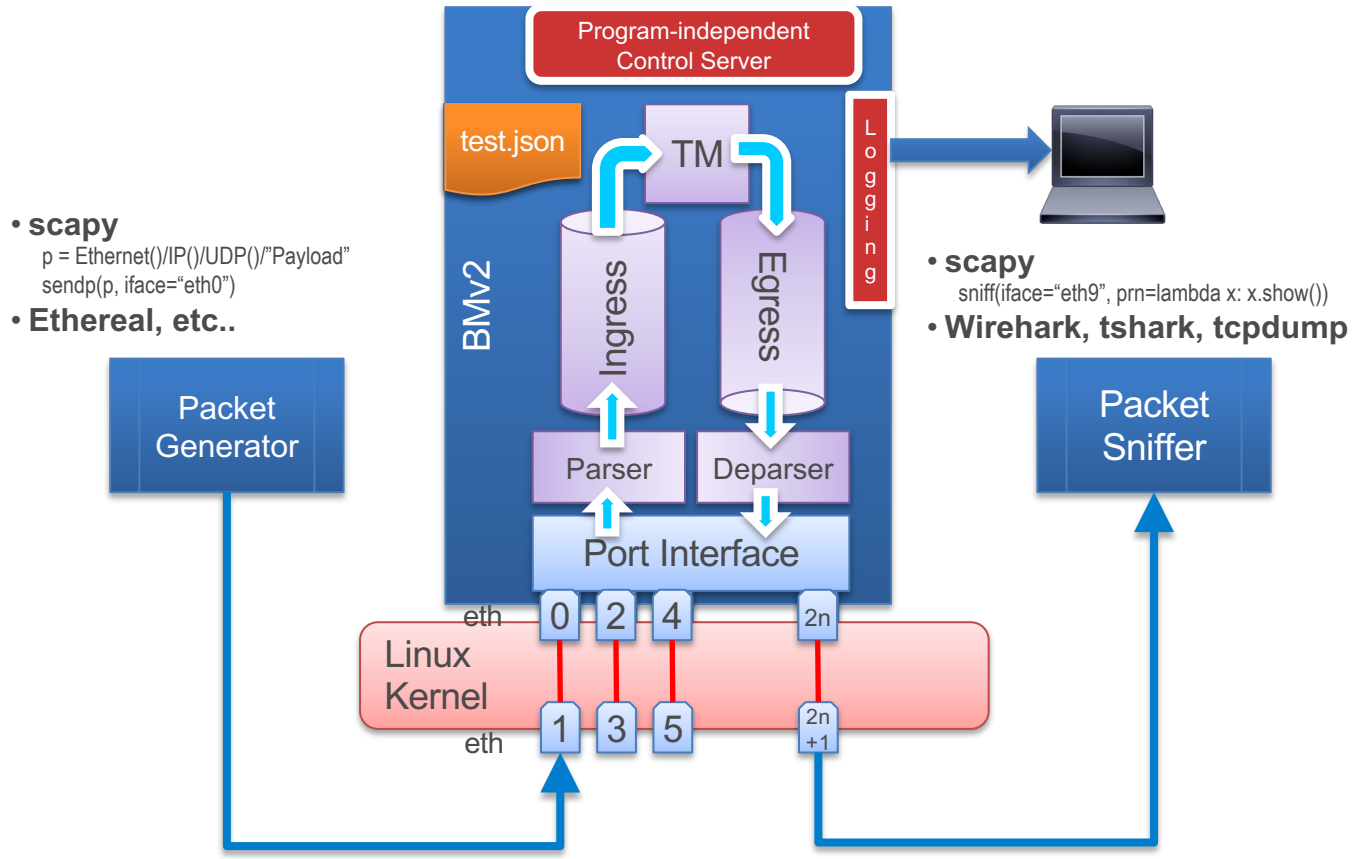
Store table rules

commands.txt

```
table_add m_table m_action 01:00:00:00:00:00&&&01:00:00:00:00:00 => 1 0
table_add m_table m_action 01:00:00:00:00:01&&&01:00:00:00:00:01 => 2 1
table_dump m_table
```

```
$ simple_switch_CLI < commands.txt
```

Step 4: Sending and Receiving Packets



Hands-on exercises

- **Platform**
  - P4app
  - Raspberry Pi

- **Exercises**
  - L2 switching
  - Calculator

Time to start programming!

https://github.com/ox-computing/P4Pi-Tutorial-NetSoft-2022

Build an Internet router

Part I: Data Plane

Students implement the data plane in P4

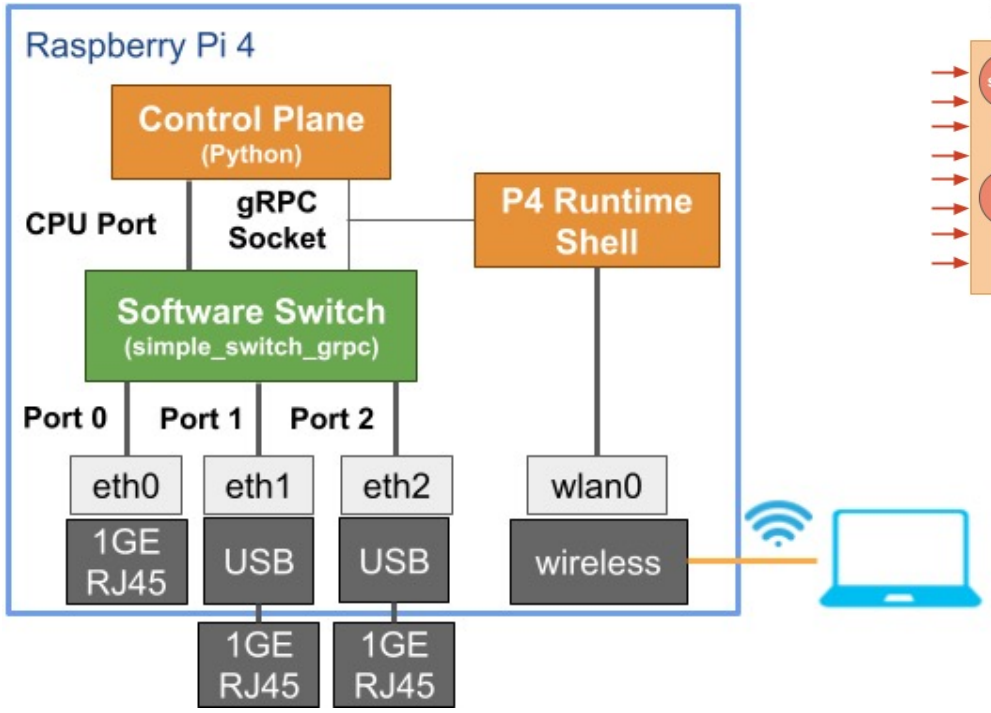
- IP Routing table
- Local IP address table
- ARP table
- L2 forwarding table

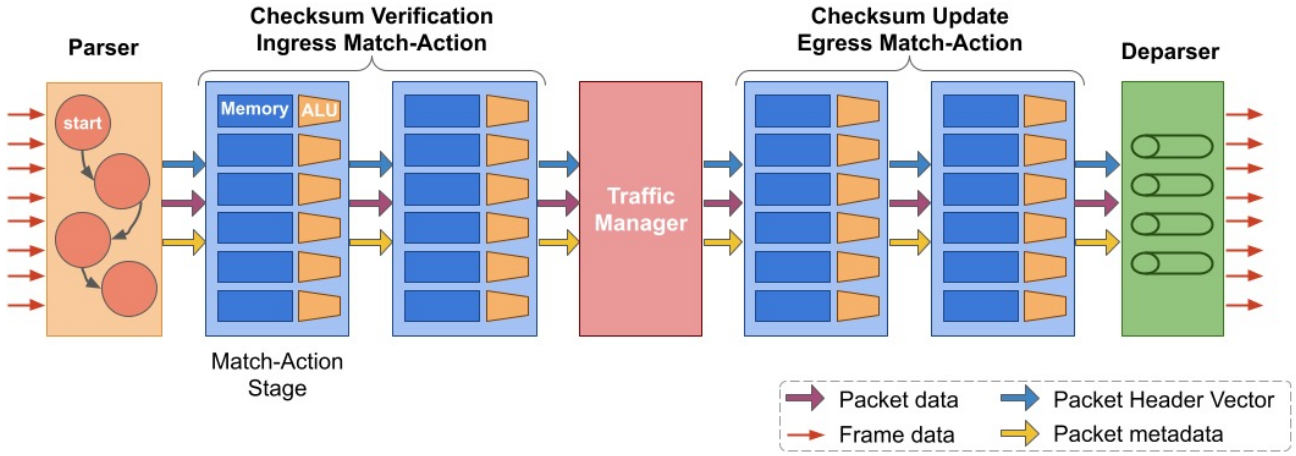
Part II: Control Plane

Students implement the control plane in python

- Pee-Wee OSPF Protocol
  - Simplified OSPFv2
  - HELLO Packets
  - Link State Update Packets
  - Topology Database

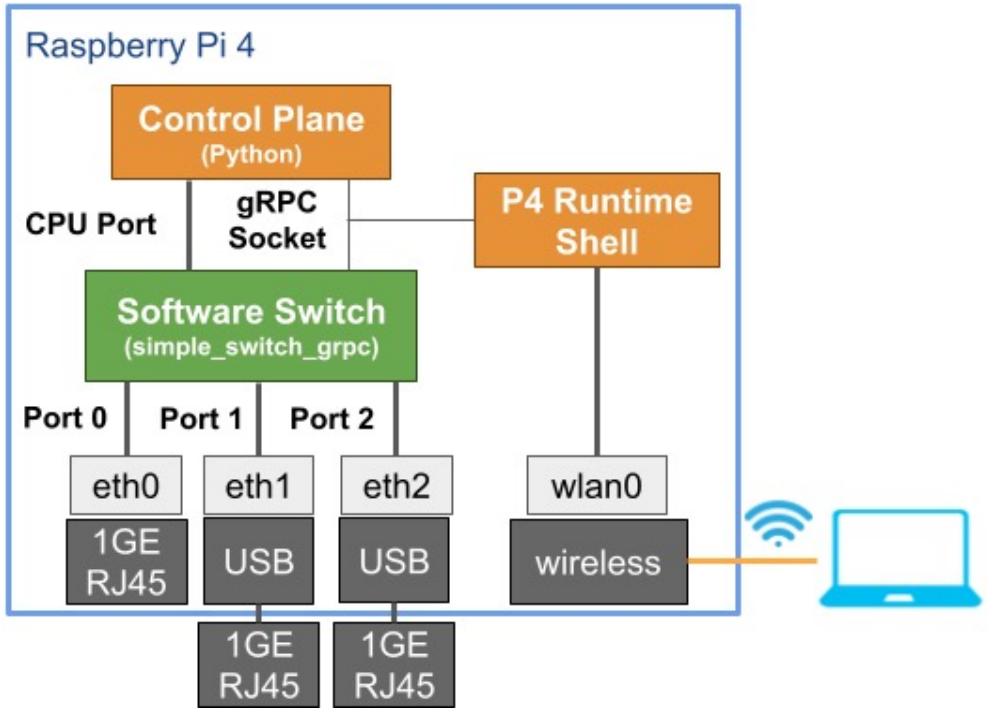
Router Architecture on P4Pi





- Architecture: v1model
- P4 Compiler: p4c-bm2-ss
- P4 Target: simple_switch_grpc

Router Architecture on P4Pi

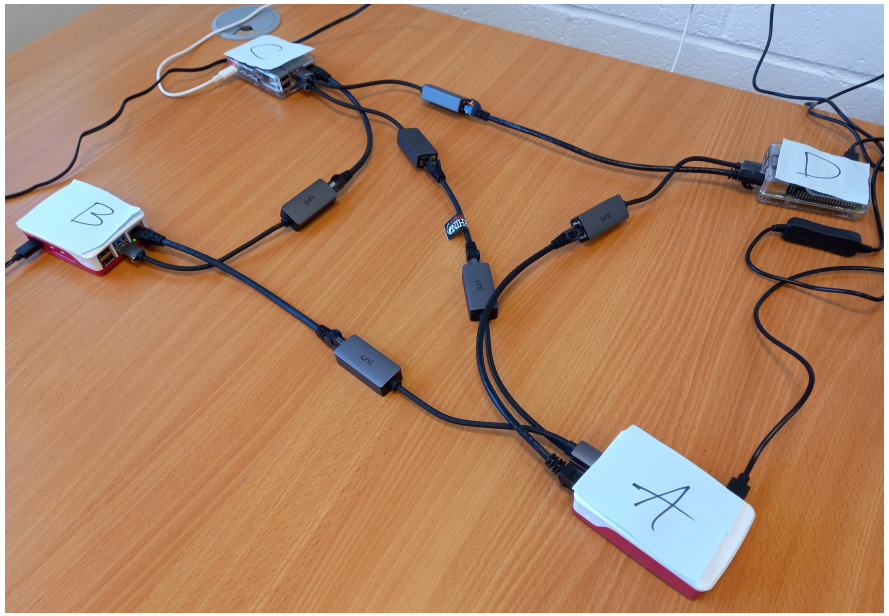


- Students develop the P4 data plane code and Python control plane code
  - Either on their laptop
    - Can test in the tutorials' environment
  - OR directly on P4Pi
- The router runs on P4Pi as any other project
- Use ssh to connect, configure and run

Testing is ***FUN!***

- Stand alone tests:
  - Using P4Pi and laptop(s)
  - Test ARP, Ping, iperf, …

- Interoprability tests
  - Multiple students projects, multiple P4Pi platforms
  - What happens when you add new routers?
  - What happens when you disconnect cables?



Conclusion

- P4Pi platform is currently **under development**
  - Contributions are welcome
  - https://github.com/p4lang/p4pi
- Developed for **networking education**
  - Cheap, available and open
  - Hands-on experience
- Making P4 available **for hobbyists**
  - Everyone needs a P4 access point at home
  - … and other crazy ideas

Available Resources

- **P4Pi Repository:** https://github.com/p4lang/p4pi/
- **P4Pi Wiki:** https://github.com/p4lang/p4pi/wiki
- **P4Pi Videos Playlist:** https://youtube.com/playlist?list=PLf7HGRMAlJBw2uudODtQT2B7JRcNgm-vV

Papers:

- “P4Pi: P4 on Raspberry Pi for Networking Education". Sándor Laki, Radostin Stoyanov, Dávid Kis, Robert Soulé, Péter Vörös and Noa Zilberman. ACM SIGCOMM Computer Communication Review, Volume 51, Number 3, July 2021
- "Building an Internet Router with P4Pi". Radostin Stoyanov, Adam Wolnikowski, Robert Soulé, Sándor Laki, and Noa Zilberman. 4th P4 Workshop in Europe (EuroP4) 2021, December 2021.

Acknowledgement

We thank the following people who contributed to the P4Pi project over time:

Robert Soulé (Yale), Noa Zilberman (Oxford), Sándor Laki (ELTE), Dávid Kis (ELTE), Péter Vörös (ELTE), Radostin Stoyanov (Oxford), Fernando Ramos (Lisbon), Damu Ding (Oxford), Changgang Zheng (Oxford), Xinpeng Hong (Oxford), Mingyuan Zang (DTU), Salvatore Signorello (Lisbon).

Some of the slides used in this presentation were adopted from https://github.com/p4lang/tutorials and from previous P4Pi presentations, tutorials and events by the above contributors.