Lab 1: Introduction

- **GitHub:** https://bit.ly/p4-sigcomm17
- **On VM:** `cd tutorials/SIGCOMM_17/exercises/basic`

Before we start…

- **Please make sure that your VM is up to date**
  - `cd ~/tutorials`
  - `git pull`
- **We'll be using several software tools are 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**
  - `run.sh`: compile program, populate tables, and execute on Bmv2
  - `*.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" (or peek at the P4 code in `solution/` if you must)

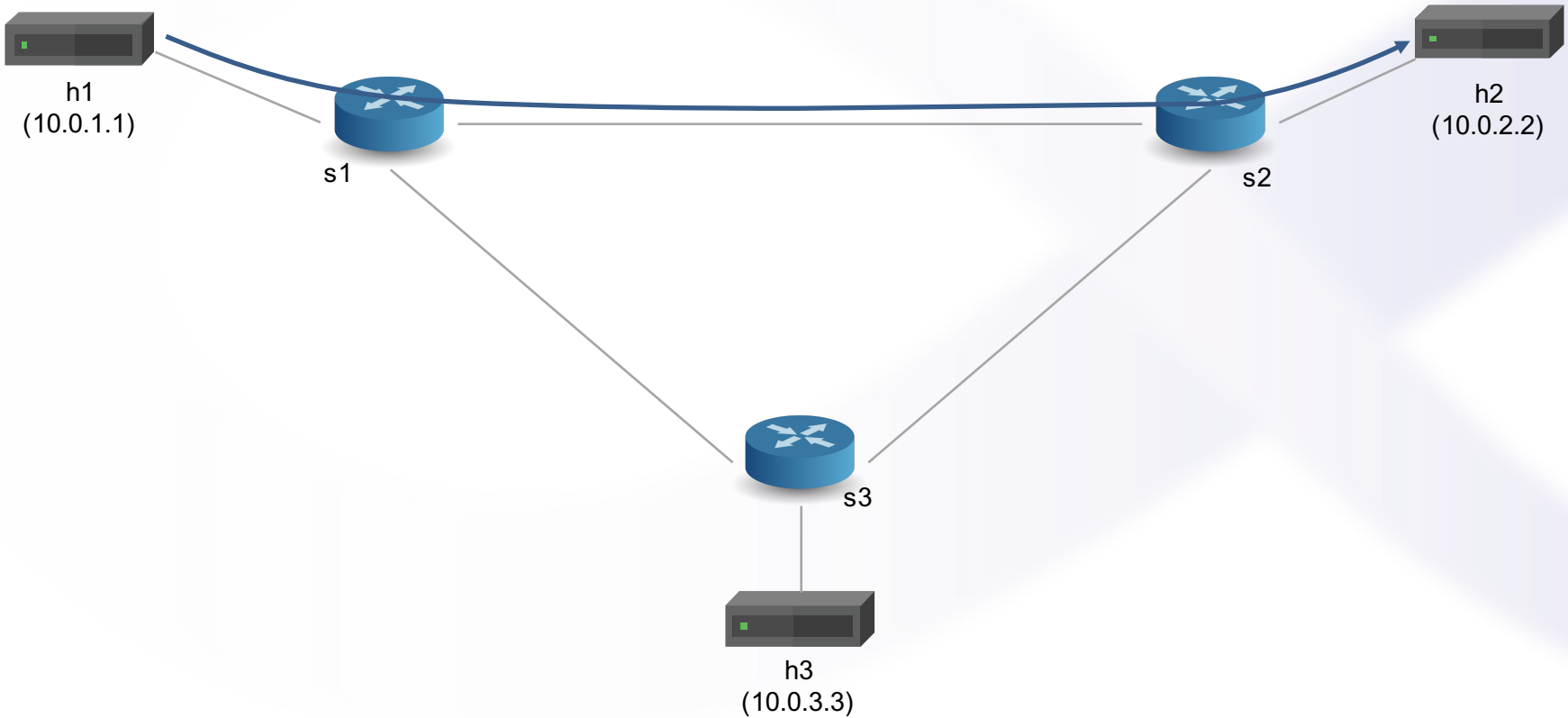
Running Example: Basic Forwarding

- **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
  - 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**

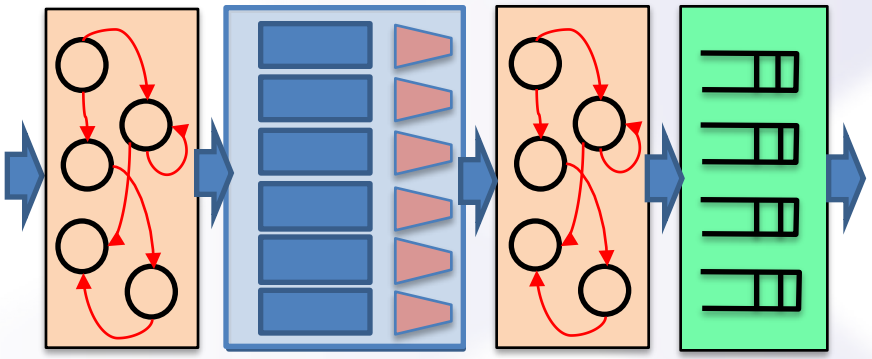
Basic Forwarding: Topology



P4$_{16}$ Language Elements

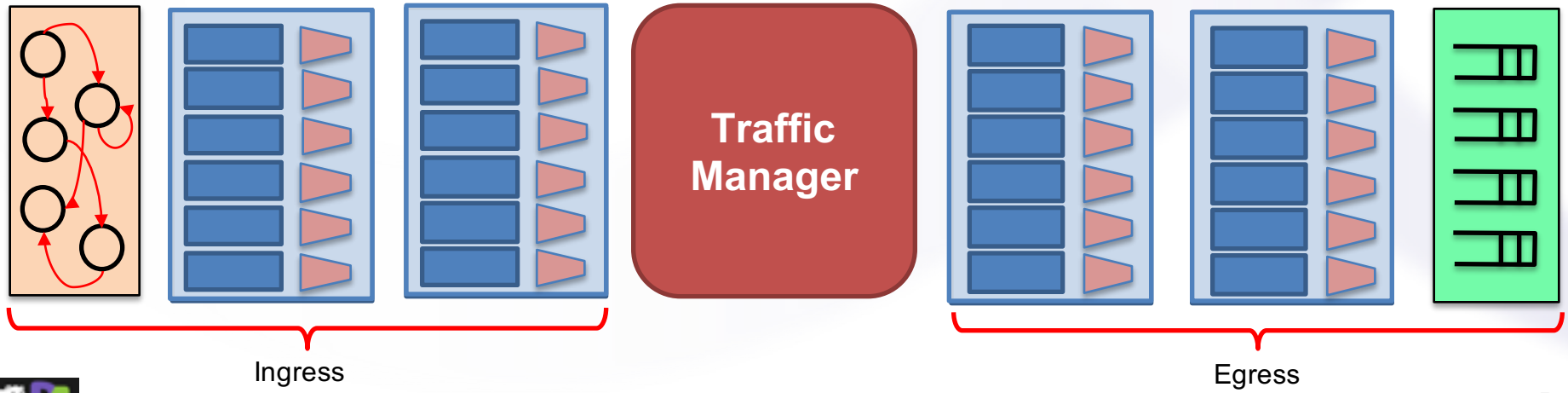
- **Parsers** — State machine, bitfield extraction
- **Controls** — Tables, Actions, control flow statements
- **Expressions** — Basic operations and operators
- **Data Types** — Bistrings, headers, structures, arrays

- **Architecture Description** — Programmable blocks and their interfaces
- **Extern Libraries** — Support for specialized components



V1Model Architecture

- **Implemented on top of Bmv2's `simple_switch` target**
- **Components:**
  - Parser/Deparser
  - Verify/Update Checksum
  - Ingress/Egress Pipelines
  - Traffic Manager (not programmable in P4)



$P4_{16}$ 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$_{16}$ 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

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

P4$_{16}$ Types (Other Types)

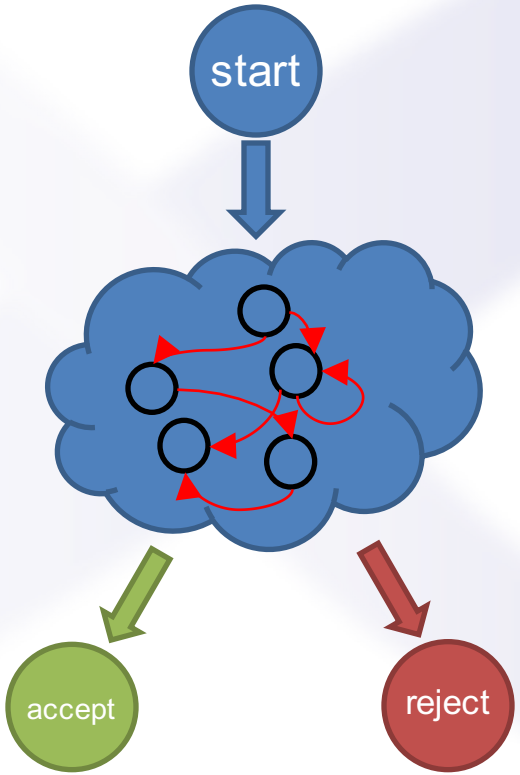
```
/* Architecture */
struct standard_metadata_t {
  bit<9>  ingress_port;
  bit<9>  egress_spec;
  bit<9>  egress_port;
  bit<32> clone_spec;
  bit<32> instance_type;
  bit<1>  drop;
  bit<16> recirculate_port;
  bit<32> packet_length;
  ...
}

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

Other useful types

- **Standard Metadata:** maintained by switch
- **Header Stack:** array of headers
- **Header Union:** one of several headers
- **Struct**: Unordered collection of members (with no alignment restrictions)
- **Typedef**: Alternative name for a type

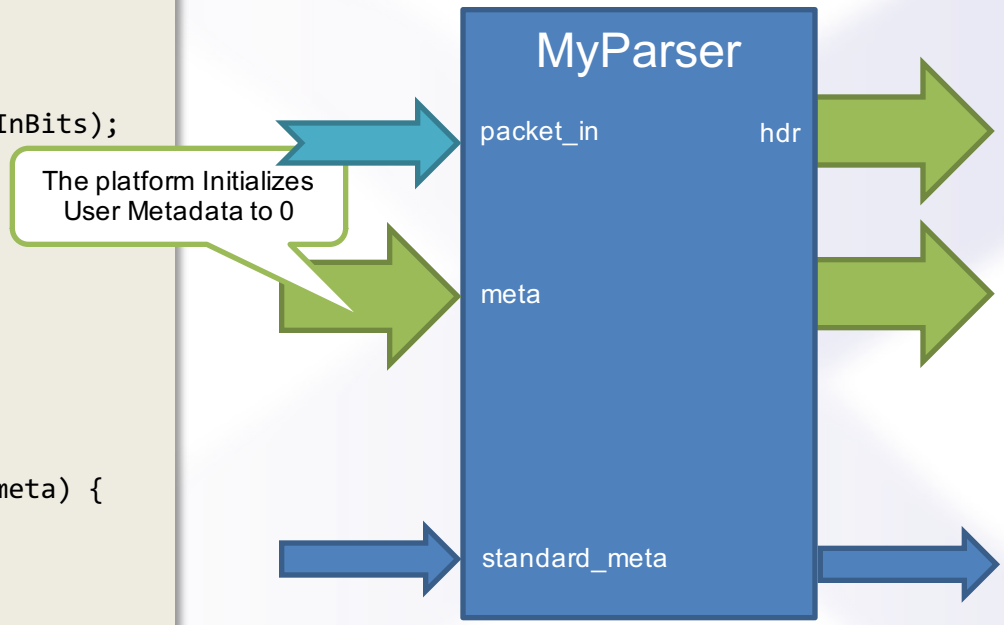
P4$_{16}$ Parsers

- 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 {
     parser.extract(hdr.ethernet);
     transition accept;
  }
}
```



Coding Break

Select Statement

```
state start {
  transition parse_ethernet;
}

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

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

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

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

P4$_{16}$ Controls

- **Similar to C functions (without loops)**
- **Can declare variables, create tables, instantiate externs, etc.**
- **Functionality specified by code in `apply` block**
- **Represent all kinds of processing that are expressible as DAG:**
  - Match-Action Pipelines
  - Deparsers
  - Additional forms of packet processing (updating checksums)
- **Interfaces between controls 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

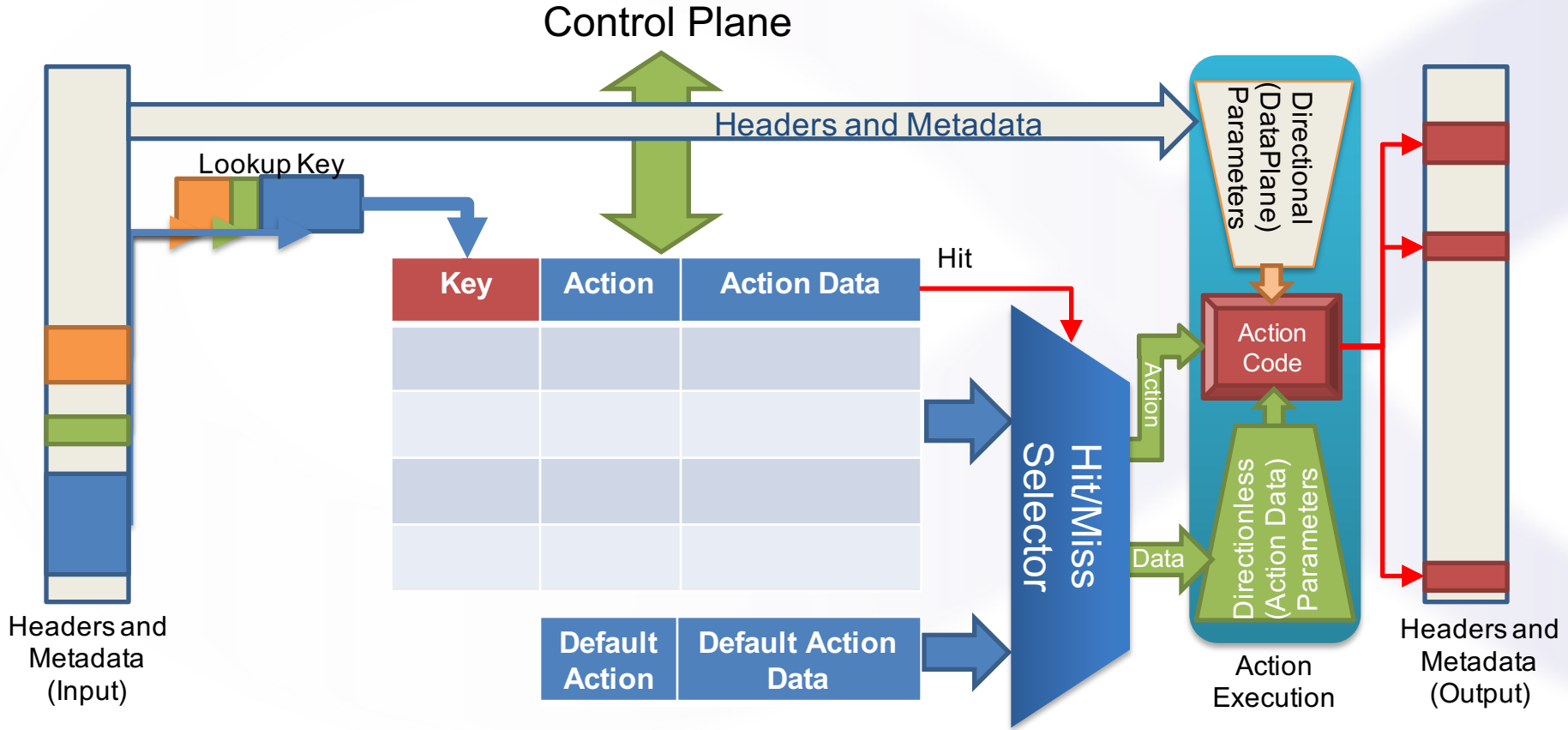
```
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t std_meta) {
  action swap_mac(inout bit<48> src,
                  inout bit<48> dst) {
    bit<48> tmp = src;
    src = dst;
    dst = tmp;
  }
  apply {
    swap_mac(hdr.ethernet.srcAddr,
             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: ++

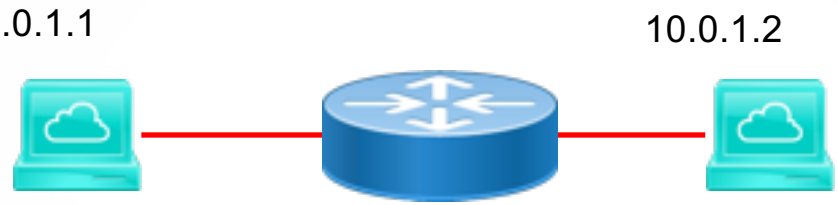
$P4_{16}$ Tables

- **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

IPv4_LPM Table

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

Match Kinds

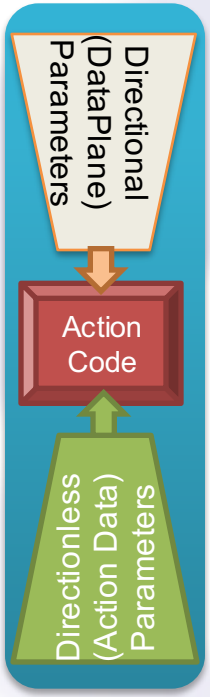
```
/* core.p4 */
match_kind {
    exact,
    ternary,
    lpm
}
/* v1model.p4 */
match_kind {
    range,
    selector
}
/* Some other architecture */
match_kind {
    regexp,
    fuzzy
}
```

- **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() {
}

/* v1model.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 direction-less 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();
     ...
  }
}
```

Coding Break

Debugging

```
control MyIngress(...) {
  table debug {
    key = {
      std_meta.egress_spec : exact;
    }
    actions = { }
  }
  apply {
     ...
     debug.apply();
  }
}
```

- **Bmv2 maintains logs that keep track of how packets are processed in detail**
  - `build/logs/s1.log`
  - `build/logs/s2.log`
  - `build/logs/s3.log`
- **Can manually add information to the logs by using a dummy debug table that reads headers and metadata of interest**

```
[15:16:48.145] [bmv2] [D]
[thread 4090] [96.0] [cxt 0]
Looking up key:
* std_meta.egress_spec : 2
```

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);
    ...
  }
}
```

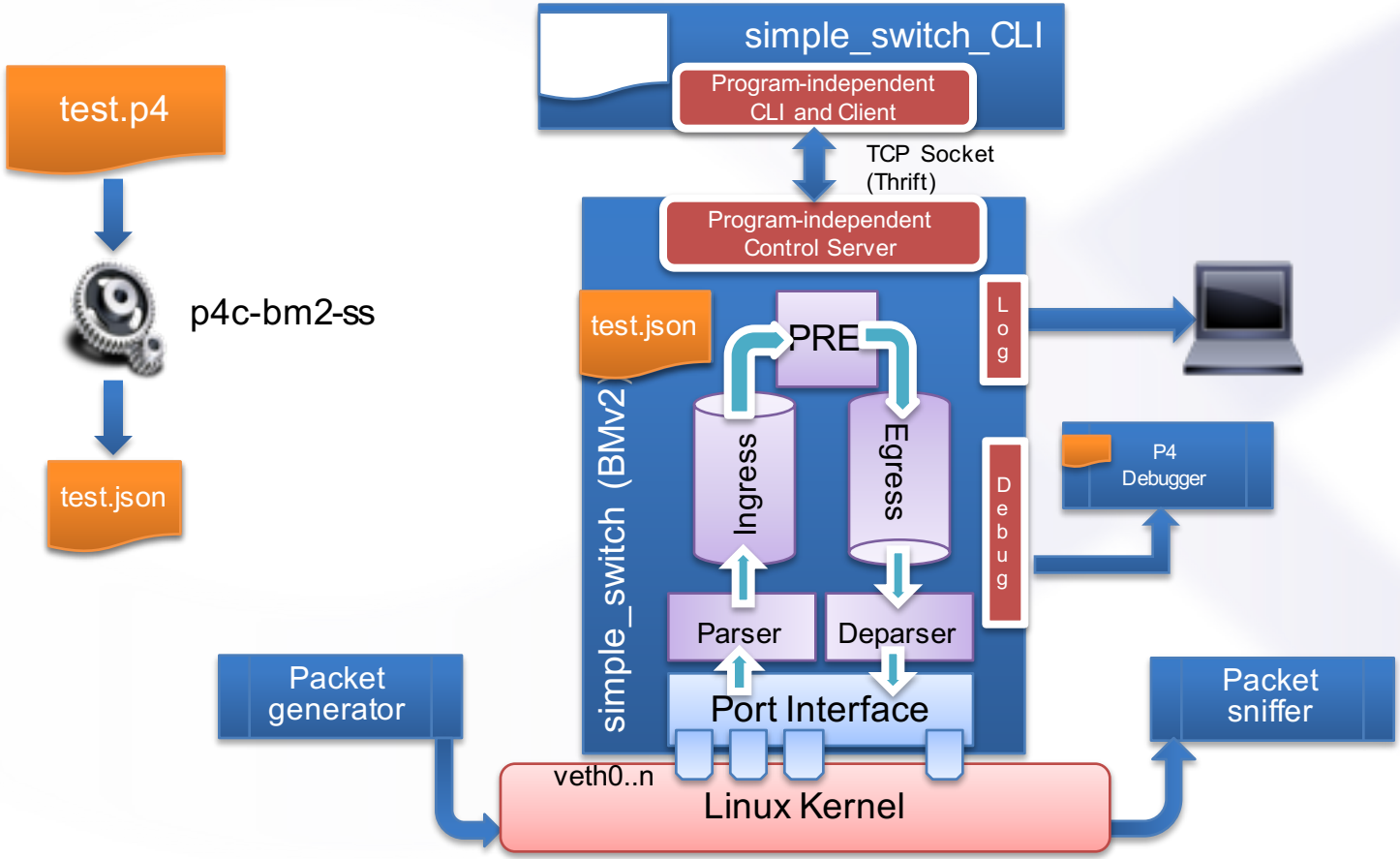
- **Assembles the headers back into a well-formed packet**
- **Expressed as a control function**
  - No need for another construct!
- **packet_out extern is defined in core.p4:** emit(hdr): serializes header if it is valid
- **Advantages:**
  - Makes deparsing explicit…
  - …but decouples from parsing

Scrambler

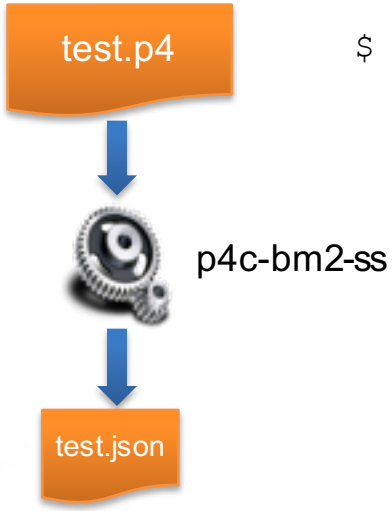
- In this exercise, we'll extend the basic forwarding program to scramble headers by inverting the Ethernet and IPv4 source and destination addresses
- As long as the control & data planes are synchronized, it will still deliver packets correctly
- In addition, as long as a packet traverses an even-number of hops, even the receiver won't notice a difference
- Hint: ~ operator computes the bit-wise complement of a value

Coding Break

run.sh: under the hood

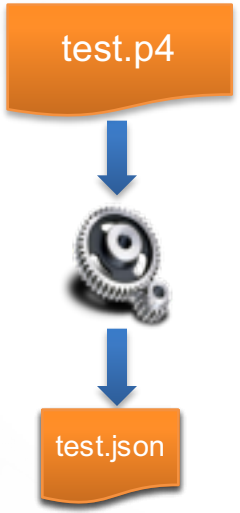


Step 1: P4 Program Compilation

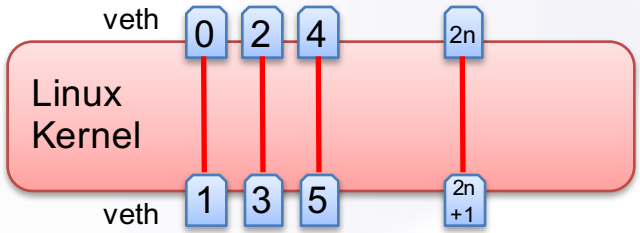



```
$ p4c-bm2-ss -o test.json test.p4
```

Step 2: Preparing veth Interfaces

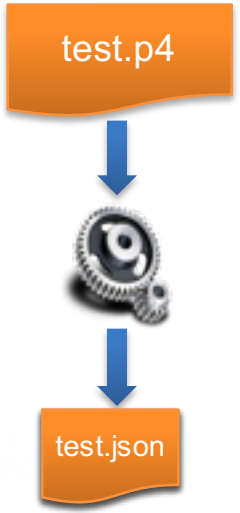


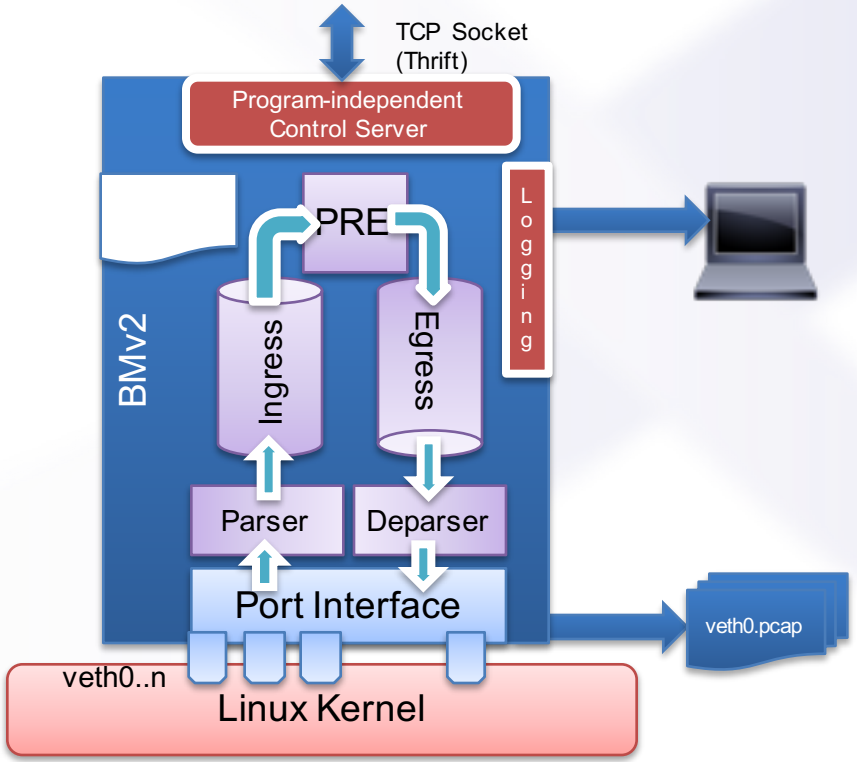

```
$ sudo ~/p4lang/tutorials/examples/veth_setup.sh

# 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

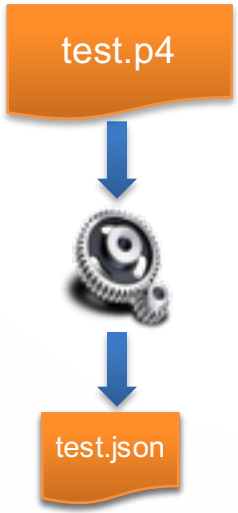
```
$ sudo simple_switch --log-console --dump-packet-data 64 \
       -i 0@veth0 -i 1@veth2 …                          \
       test.json
```

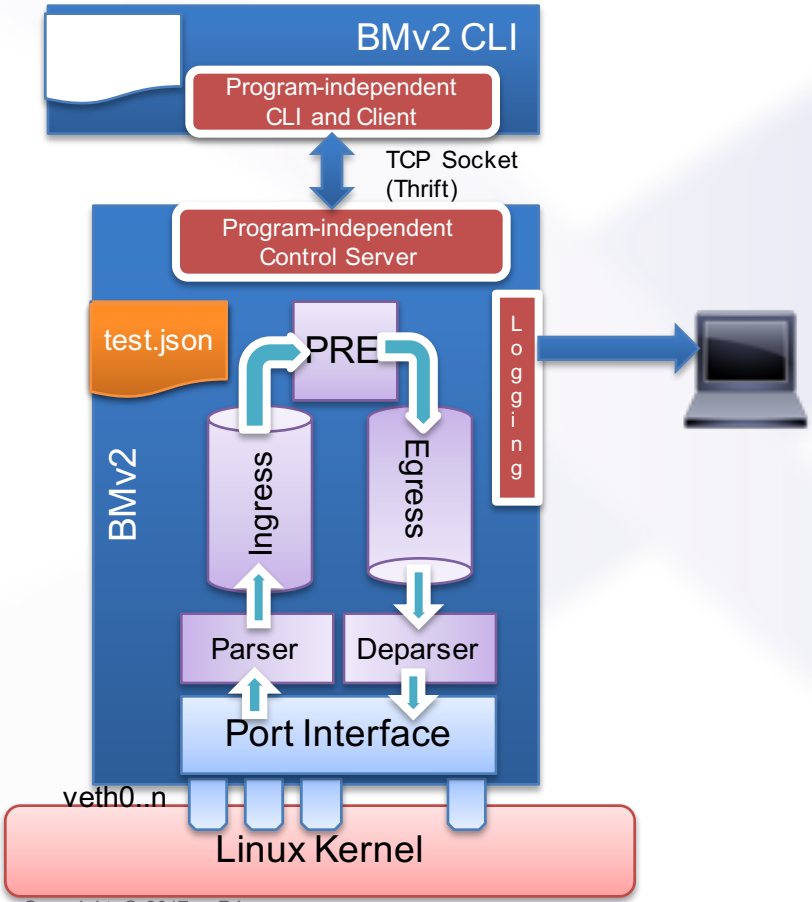




Step 4: Starting the CLI

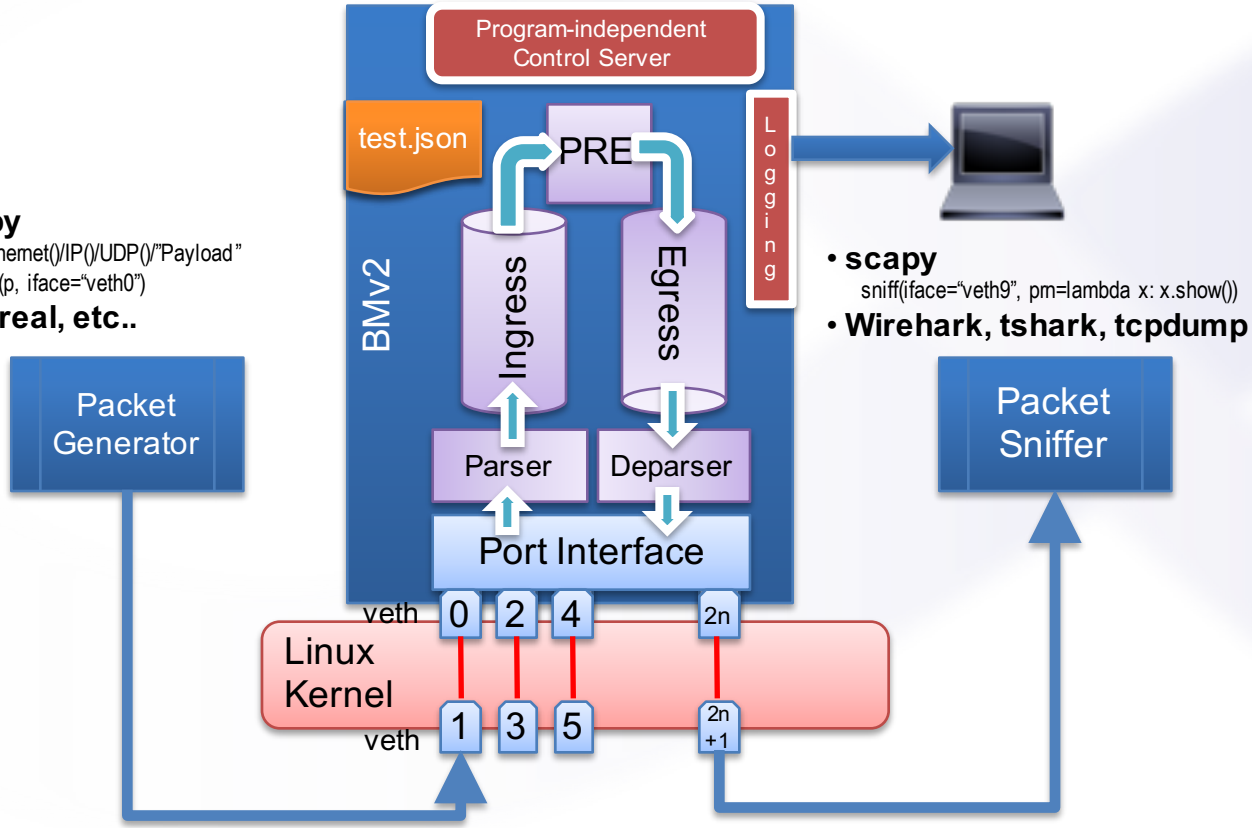
```
$ simple_switch_CLI
```





Step 5: Sending and Receiving Packets

- **scapy**
  p = Ethernet()/IP()/UDP()/"Payload"
  sendp(p, iface="veth0")
- **Ethereal, etc..**



- **scapy**
  sniff(iface="veth9", pm=lambda x: x.show())
- **Wirehark, tshark, tcpdump**

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:  dump_table  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

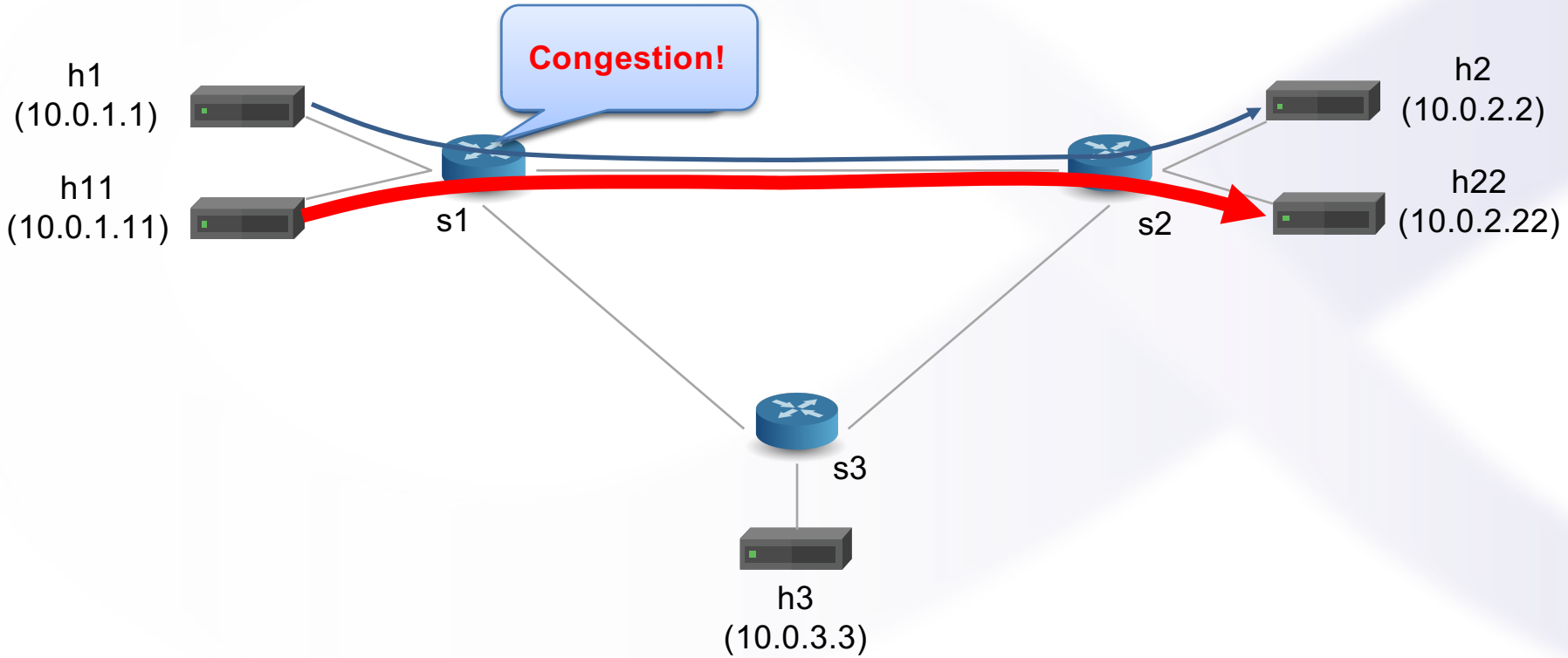
Fin!

Plan

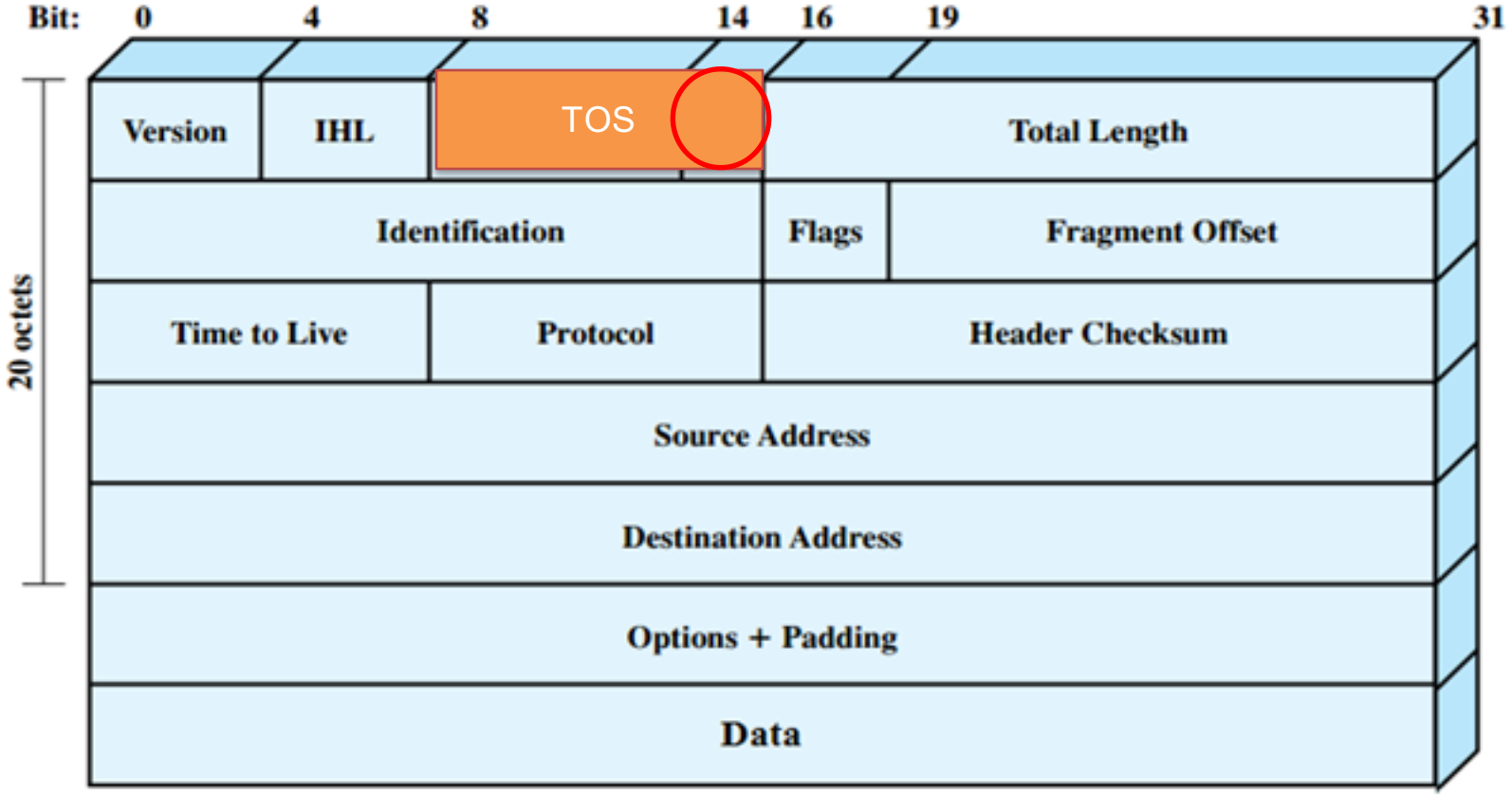
- **The rest of the day is organized into a series of labs**
- **Each lab contains a pair of thematically-similar exercises**
  - 1. Introduction ✔
    - Basic forwarding & Scrambler
  - 2. Monitoring & Debugging
    - ECN & MRI (pun intended)
  - 3. Advanced Data Structures
    - Source routing & Calculator
  - 4. Dynamic forwarding
    - Load balancing & HULA

Lab 2: Monitoring & Debugging

Monitoring & Debugging



Explicit Congestion Notification

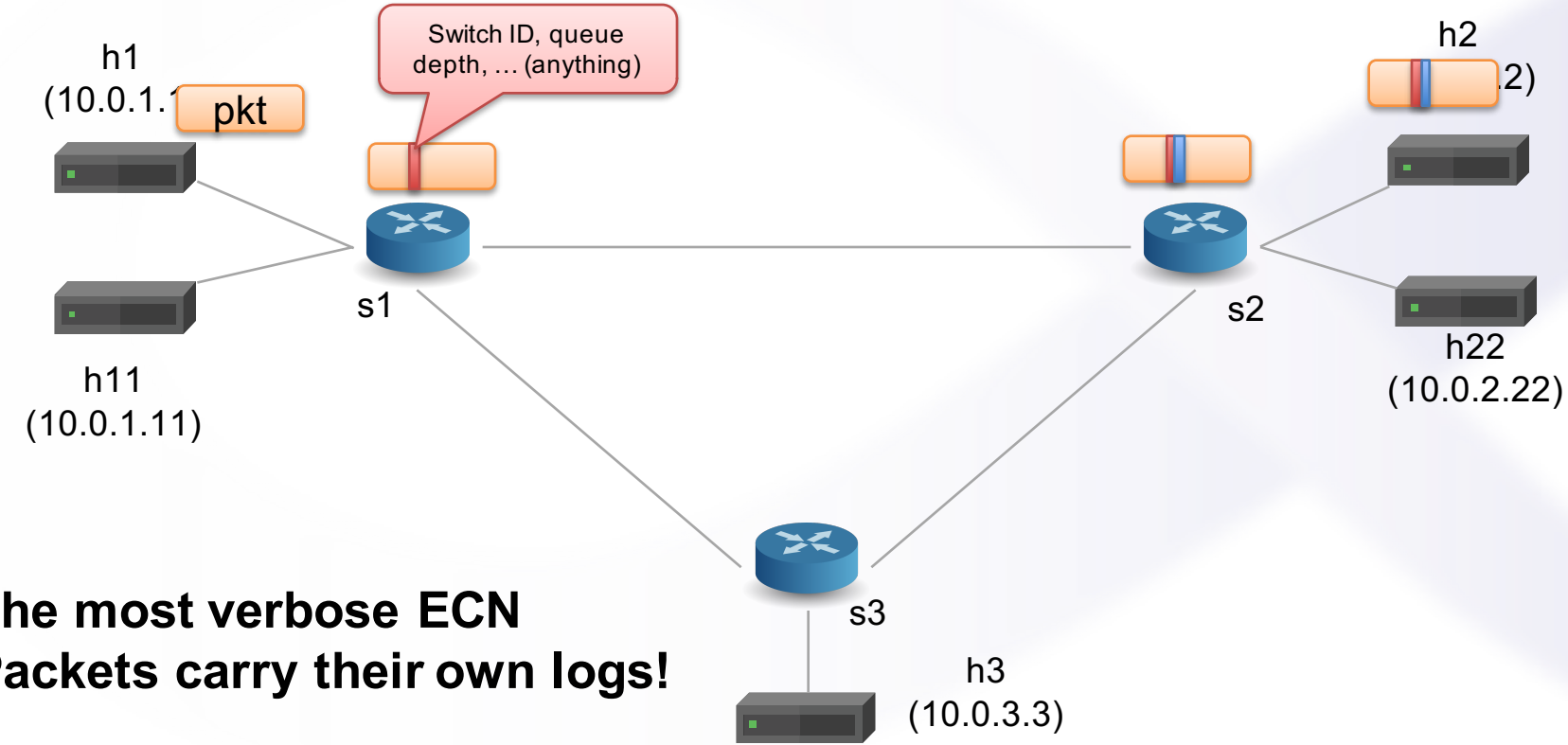


Explicit Congestion Notification

- **Explicit Congestion Notification**
  - `00`: Non ECN-Capable Transport, Non-ECT
  - `10`: ECN Capable Transport, ECT(0)
  - `01`: ECN Capable Transport, ECT(1)
  - `11`: Congestion Encountered, CE
- **For packets originating from ECT, ECN-capable switches set the CE bit upon congestion instead of dropping the packet**
  - E.g., observed queue depth > threshold
- **The standard data for the V1Model includes the queue depth:**

```
bit<19> standard_metadata.enq_qdepth
```

Multi-Route Inspection



The most verbose ECN
Packets carry their own logs!

Multi-Route Inspect: Packet Format

```
header mri_t {
  bit<16>  count;
}

header switch_t {
  switchID_t  swid;
  qdepth_t    qdepth;
}

struct headers {
  ethernet_t         ethernet;
  ipv4_t             ipv4;
  ipv4_option_t      ipv4_option;
  mri_t              mri;
  switch_t[MAX_HOPS] swtraces;
}
```

- **Recall header validity opertions:**
  - `hdr.setValid()`: add_header
  - `hdr.setInvalid()`: remove_header
  - `hdr.isValid()`:test validity
- **Header Assignment**
  - `hdr = { f1, f2, ..., fn }`
  - `hdr1 = hdr2`
- **Header Stacks**

  Parsers:
  - `stk.next`
  - `stk.last`
  - `stk.lastIndex`

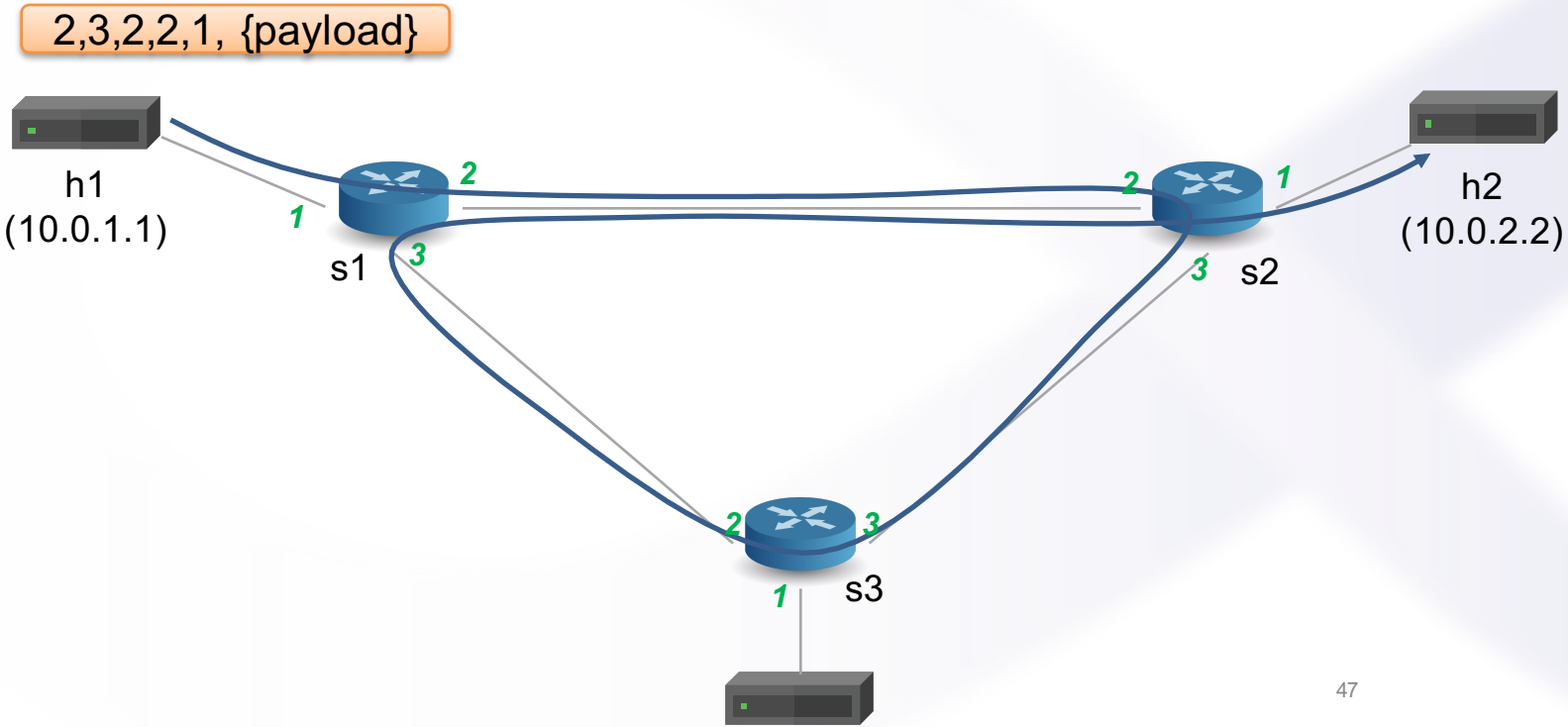
  Controls:
  - `stk[i]`
  - `stk.size`
  - `stk.push_front(int count)`
  - `stk.pop_front(int count)`

Header verification

```
/* Standard errors, defined in core.p4 */
error {
   NoError,           // no error
   PacketTooShort,    // not enough bits in packet for extract
   NoMatch,           // match expression has no matches
   StackOutOfBounds,  // reference to invalid element of a header stack
   OverwritingHeader, // one header is extracted twice
   HeaderTooShort,    // extracting too many bits in a varbit field
   ParserTimeout      // parser execution time limit exceeded
}
/* Additional error added by the programmer */
error { IPv4BadHeader }
...
state parse_ipv4 {
  packet.extract(hdr.ipv4);
  verify(hdr.ipv4.version == 4, error.IPv4BadHeader);
  transition accept;
}
```

Lab 3: Advanced Data Structures

Source Routing

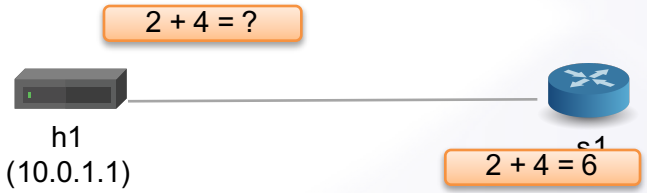


Source Routing: Packet Format

```
#define MAX_HOPS 9

const bit<16> TYPE_IPV4 = 0x800;
const bit<16> TYPE_SRCROUTING = 0x1234;

header srcRoute_t {
  bit<1>    bos;
  bit<15>   port;
}

struct headers {
  ethernet_t           ethernet;
  srcRoute_t[MAX_HOPS] srcRoutes;
  ipv4_t               ipv4;
}
```

- Parse source routes only if etherType is 0x1234
- The special value bos == 0 indicates the “bottom of stack”
- Forward packets using source routes, and also decrement IPv4 TTL
- Drop the packet if source routes are not valid

Calculator



Calculator: Packet Format

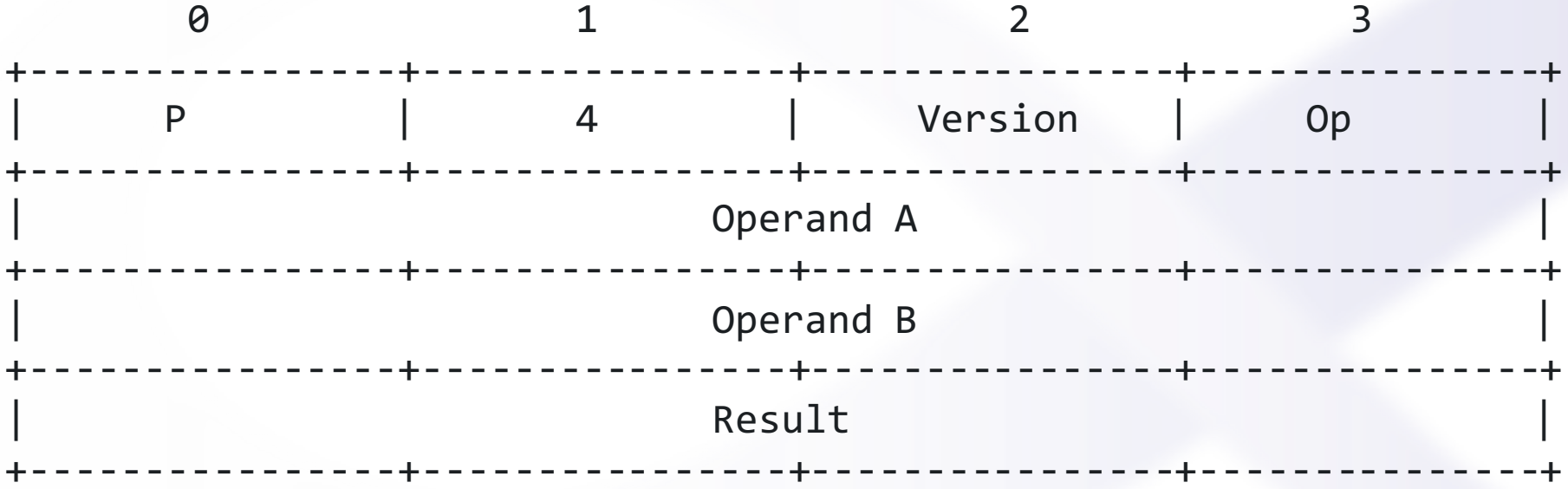


Table Initializers

```
table tbl {
  key = { hdr.h.f : exact }
  actions = { a1; a2; a3 }
  entries = {
   { 0x01 } : a1(1);
   { 0x02 } : a1(2);
   { _ } : NoAction();
  }
}
```

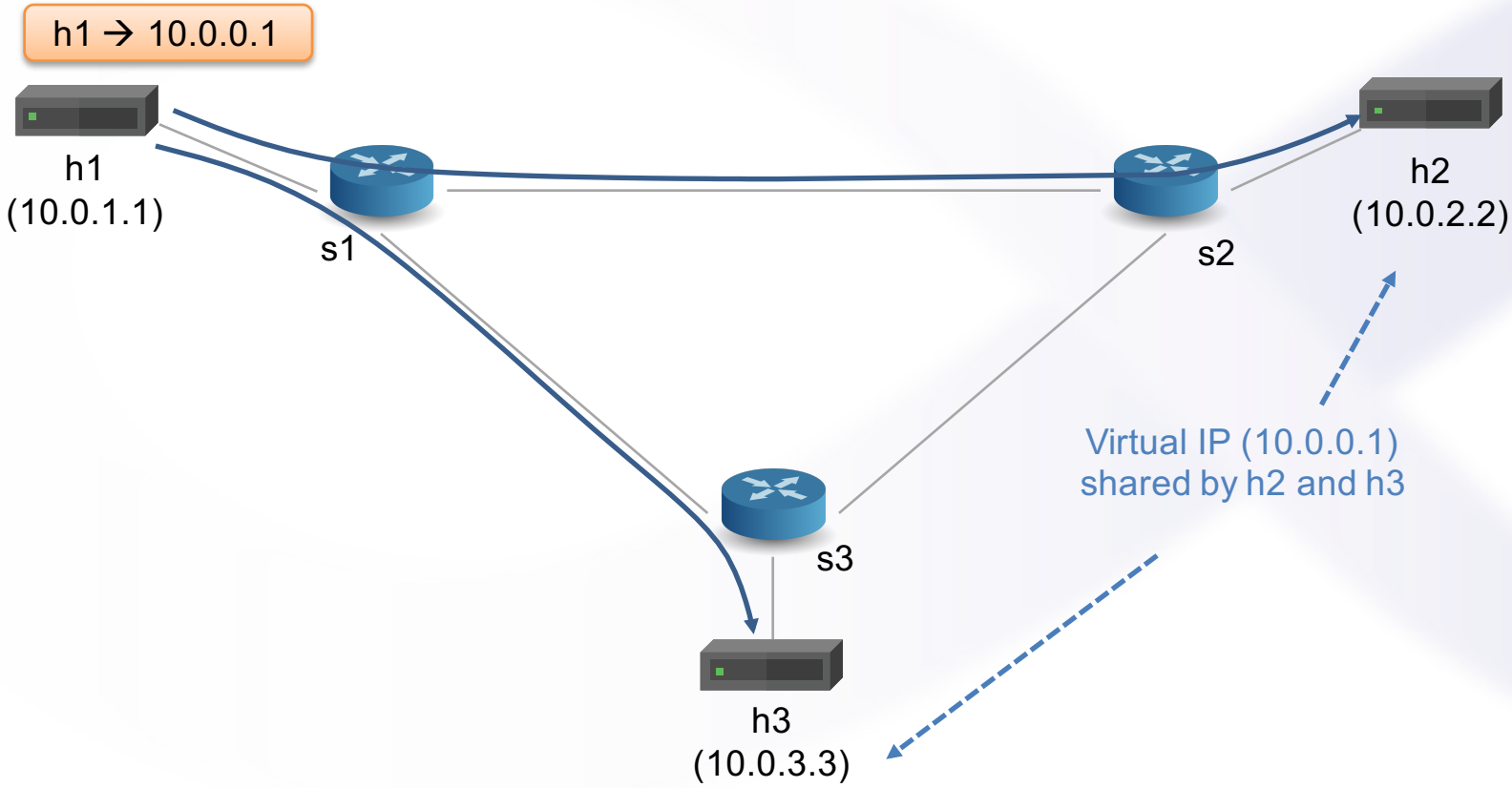
Can initialize tables with constant entries

Must fully specify the value of all action data, including values that are normally supplied by the control-plane

Hint: for the calculator, use a table that matches on the op-code

Lab 4: Dynamic Forwarding

Simple Load Balancing



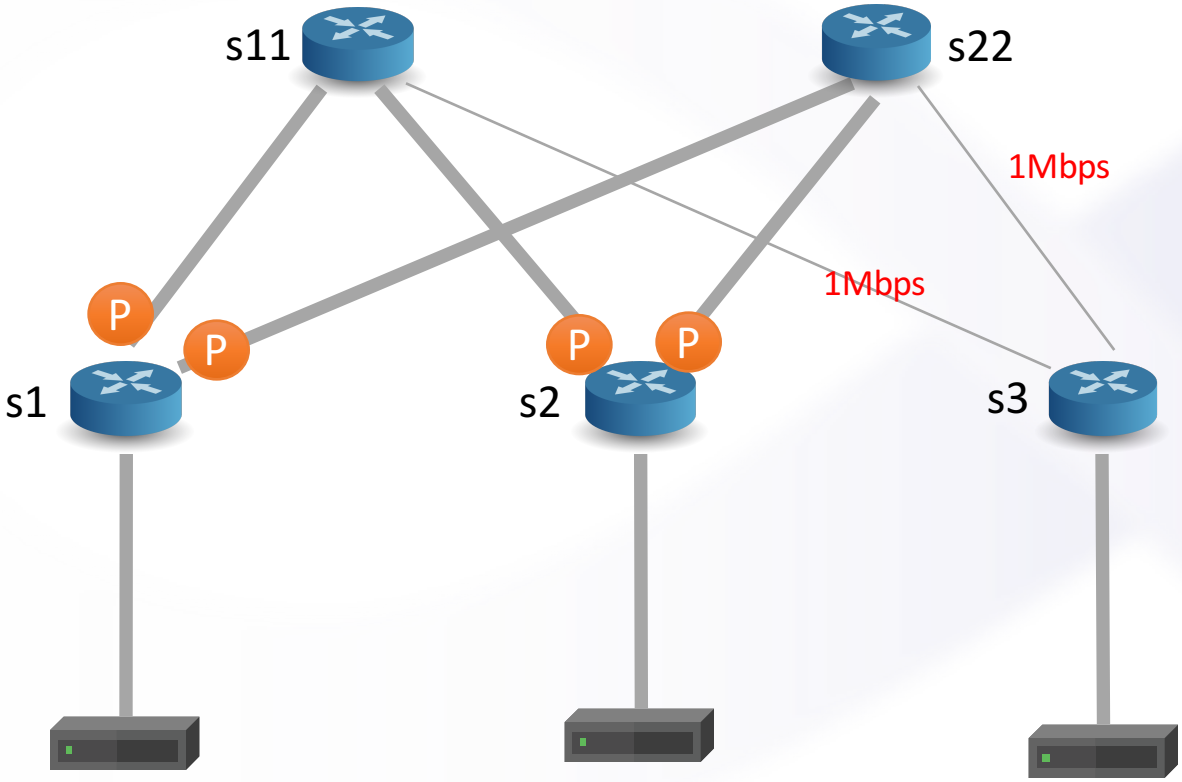
Hashing (V1Model)

```
enum HashAlgorithm {
  csum16,
  xor16,
  crc32,
  crc32_custom,
  crc16,
  crc16_custom,
  random,
  identity
}
extern void hash<O, T, D, M>(
  out O result,
  in HashAlgorithm algo,
  in T base,
  in D data,
  in M max);
```

Computes the hash of `data` (using `algo`) modulo `max` and adds it to `base`

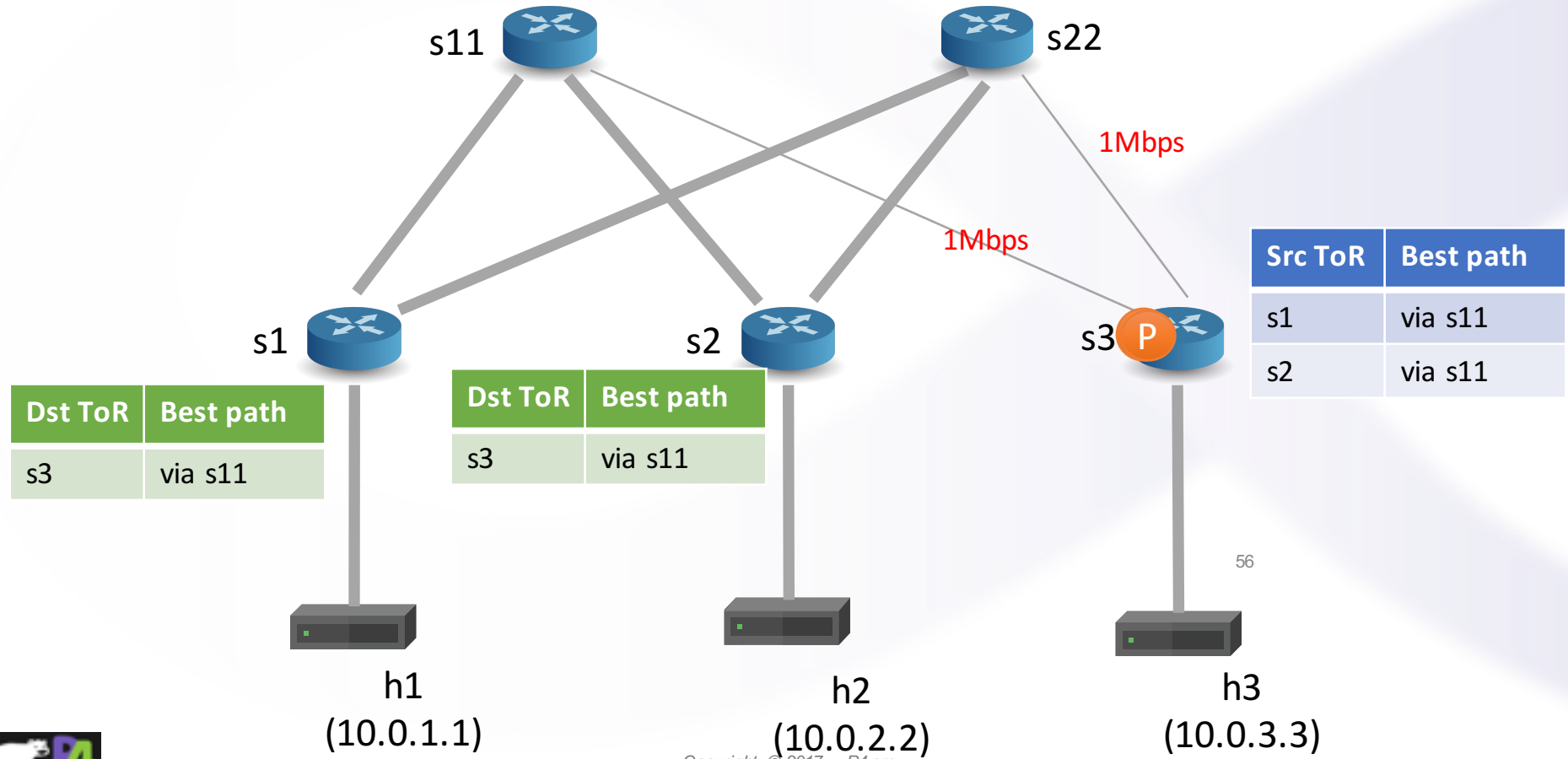
Uses type variables (like C++ templates / Java Generics) to allow hashing primitive to be used with many different types.

Dynamic Forwarding: HULA

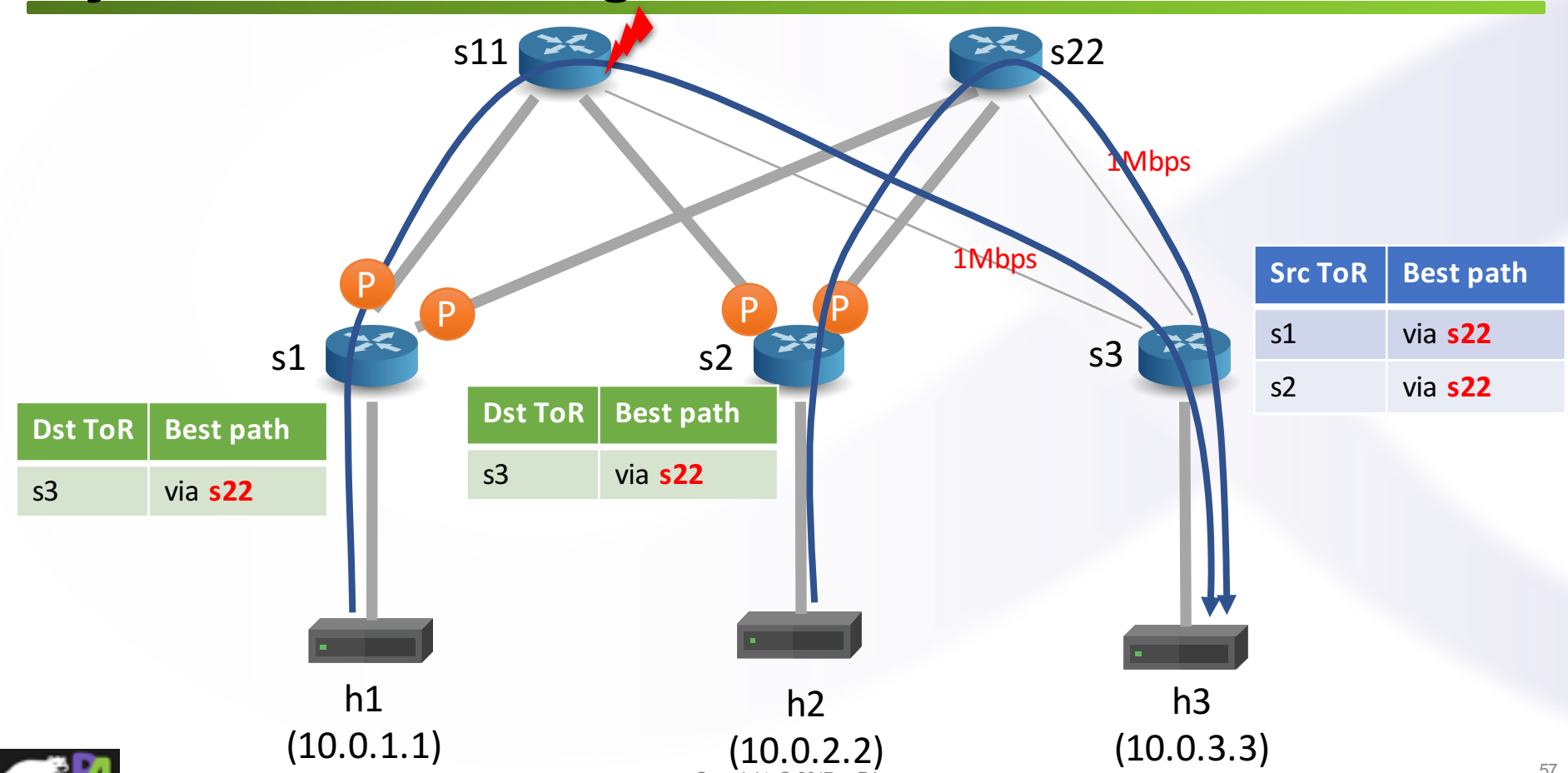


Src ToR	Best path
s1	via s11
s2	via s11

Dynamic Forwarding: HULA



Dynamic Forwarding: HULA



Dst ToR	Best path
s3	via **s22**

Dst ToR	Best path
s3	via **s22**

Src ToR	Best path
s1	via **s22**
s2	via **s22**

Using Match Results

```
apply {
    ...
    if (hdr.ipv4.isValid()) {
        if (!ipv4_host.apply().hit) {
            ipv4_lpm.apply();
        }
    }
}

apply {
    ...
    switch (ipv4_lpm.apply().action_run) {
        l3_switch_nexthop: { nexthop.apply(); }
           l3_switch_ecmp: { ecmp.apply(); }
                  l3_drop: { exit; }
                  default: { /*Do nothing */ }
    }
}
```

- **Apply method returns a special result:**
  - A boolean, indicating whether the table hit
  - An enum, representing the selected action
- `switch(...)` **statement**
  - Only used with results of applying a table
  - Each case should be a block statement
  - Default case is optional
    - Means "any action" not "default action"
- `exit` **and** `return` **Statements**
  - `return` – go to the end of the current control
  - `exit` – go to the end of the top-level control
  - Useful to skip additional processing

Registers (V1Model)

Definition

```
/* From v1model.p4 */

extern register<T> {
  register(bit<32> instance_count);
  void read(out T result, in bit<32> idx);
  void write(in bit<32> idx, in T value);
}
```

Usage (Calculating Inter-Packet Gap)

```
register<bit<48>>(16384) last_seen;

action get_inter_packet_gap(out bit<48> interval,
                            bit<14> flow_id) {
  bit<48> last_pkt_ts;
   /* Get the time previous packet was seen */
   last_seen.read((bit<32>)flow_id, last_pkt_ts);
   /* Calculate interval */
   interval = std_meta.ingress_global_timestamp -
              last_pkt_ts;
   /* Update the register with the new timestamp */
   last_seen.write((bit<32>)flow_id,
                   std_meta.ingress_global_timestamp);
}
```

Wrapping up & Next Steps

Why $P4_{16}$?

- **Clearly defined semantics**
  - You can describe what your data plane program is doing
- **Expressive**
  - Supports a wide range of architectures through standard methodology
- **High-level, Target-independent**
  - Uses conventional constructs
  - Compiler manages the resources and deals with the hardware
- **Type-safe**
  - Enforces good software design practices and eliminates “stupid” bugs
- **Agility**
  - High-speed networking devices become as flexible as any software
- **Insight**
  - Freely mixing packet headers and intermediate results

Things we covered

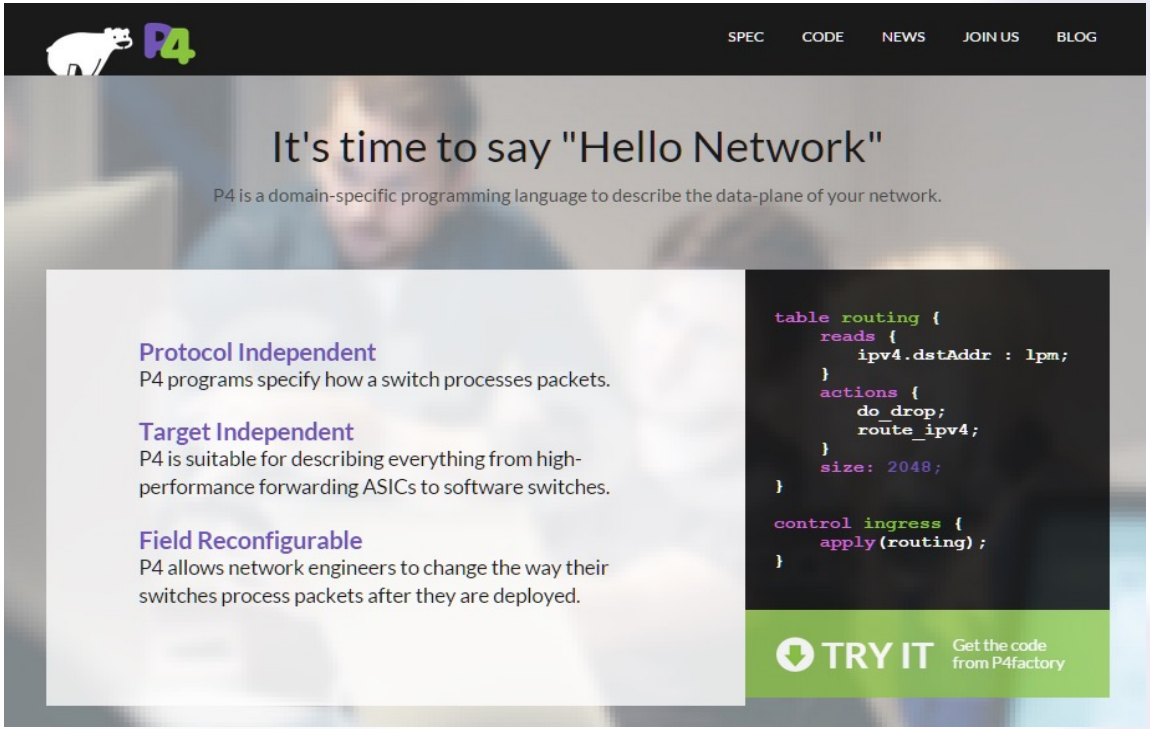
- **The P4 ”world view”**
  - Protocol-Independent Packet Processing
  - Language/Architecture Separation
  - If you can interface with it, it can be used
- **Key Data Types**
- **Constructs for packet parsing**
  - State machine-style programming
- **Constructs for packet processing**
  - Actions, tables and controls
- **Packet deparsing**
- **Architectures & Programs**

Things we didn't cover

- **Mechanisms for modularity**
  - Instantiating and invoking parsers or controls
- **Details of variable-length field processing**
  - Parsing and deparsing of options and TLVs
- **Architecture definition constructs**
  - How these "templated" definitions are created
- **Advanced features**
  - How to do learning, multicast, cloning, resubmitting
  - Header unions
- **Other architectures**
- **Control plane interface**

The P4 Language Consortium

- Consortium of academic and industry members
- Open source, evolving, domain-specific language
- Permissive Apache license, code on GitHub today
- Membership is free: contributions are welcome
- Independent, set up as a California nonprofit



P4.org Membership

- **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

Thank you