

# LABORATORY OF NETWORK PROGRAMMABILITY AND AUTOMATION

## PROGRAMMABLE DATA PLANE WITH P4

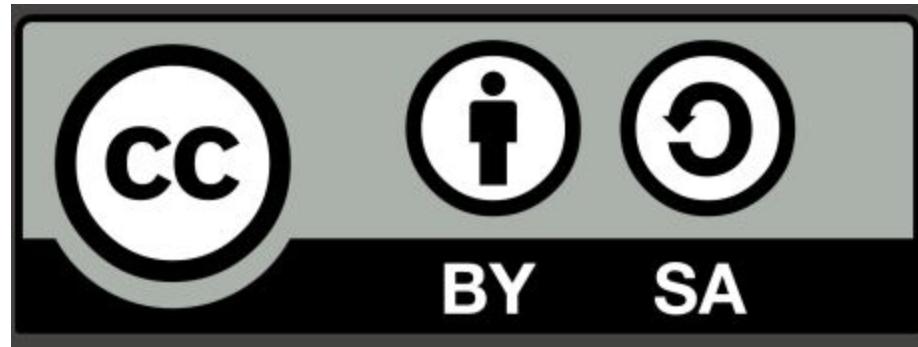
---

Andrea Melis

Computer Science and Engineering Department, University of Bologna

# License

Released under CC-BY-SA License



<https://creativecommons.org/licenses/by-sa/4.0/>

# Agenda

- Data Plane Introduction
  - What it is?
  - What does it do?
  - Data Plane in a SDN
- Why we need a Programmable Data Plane
  - Challenges
  - Benefits
- P4 Language
  - History
  - WorkFlow
  - Architecture
  - Goal

# What is the data plane?

- Processing packet streams
  - Large volume, packets come in streams, algorithms process them
    - super fast → small time to process single packet
    - matching bitfields, simple actions
    - at end hosts → NIC
    - inside the network → router, switch, firewall
- Bunch of different functionality
  - packet forwarding (switch)
  - access control (firewall)
  - tunneling
  - traffic monitoring
  - buffering and marking
  - shaping and scheduling
  - Deep packet inspection (DPI box)



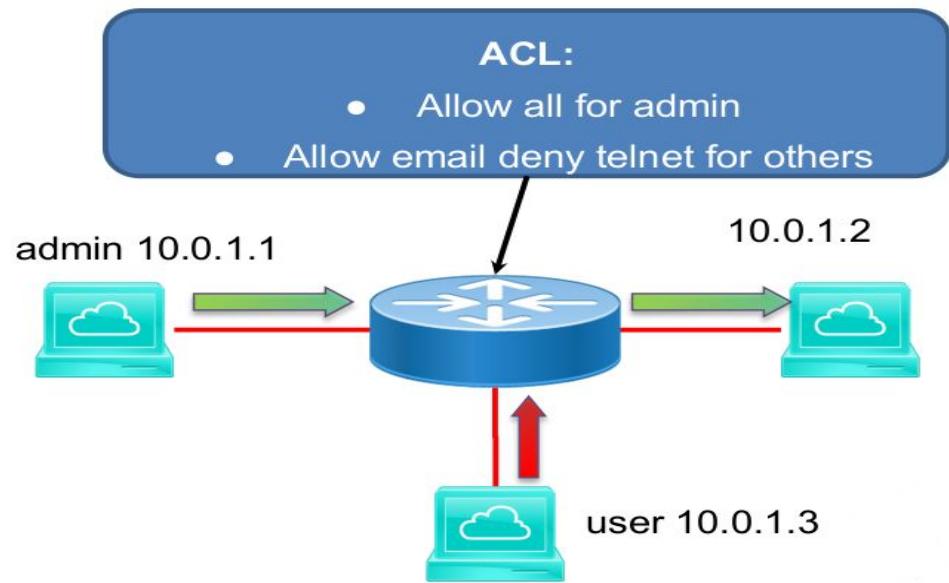
# Access Control

- **Packet Filtering → Access Control Lists (ACL)**

- Src, Dst IP address
  - Src, Dst ports
  - Protocol ID

- **Stateful operations**

- also for security,  
e.g. attacks
  - e.g. block all TCP syn  
packets from outside
  - requires to parse TCP  
headers and maintain  
flow state



# Access Control List

- Accept/Drop actions
  - ordered and list
  - Wildcard rules possible
  - list entries can overlap → priority

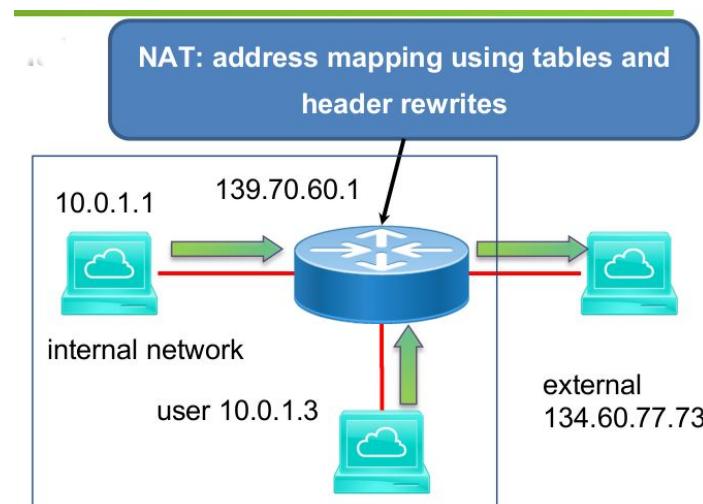
- Packet classification
  - match header fields
  - identify match with highest priority

- Different approaches
  - multi-dimensional classification algorithms
  - Use TCAMs: ternary content addressable memory

Src=1.2.3.4, Dest=5.6.7.8	accept
Dest=1.2.3.*	drop
Dest=1.2.3.8, Dport!=53	accept
Src=1.2.3.7, Dport=100	accept
Dport=100	drop

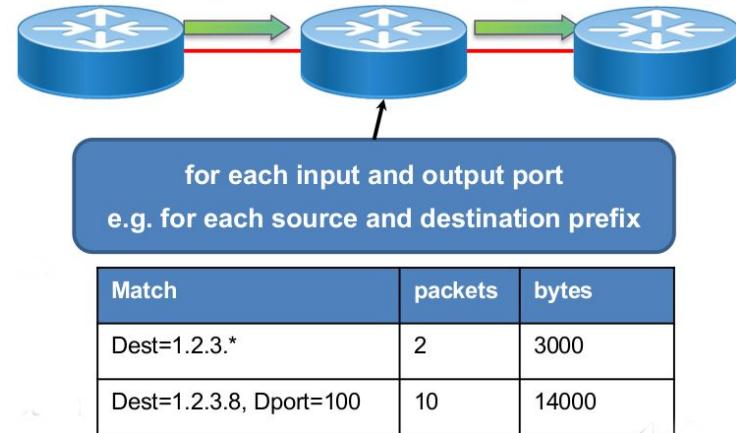
# Network Address Translation - NAT

- Mapping between internal and external addresses
  - IP-addresses: between end-hosts and NAT
  - ports: each connection needs to be unique
- NAT Table
  - entries are dynamically created
  - when to remove entries?
  - what if both ends are behind NAT?



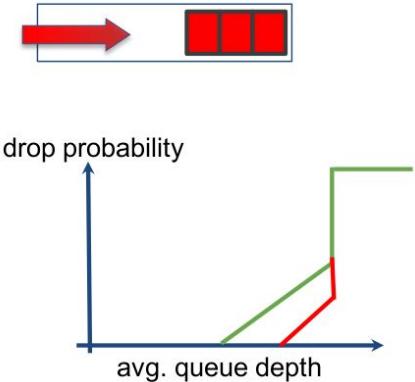
# Traffic Monitoring

- Why Traffic Monitoring?
  - volume based charging, traffic engineering,
  - anomaly detection, ...
- How?
  - matching header fields
  - updating counter of packets/bytes
- Challenges
  - identify correct aggregates: proactive vs. reactive
  - more information, e.g. time in queue, congestion states,..
  - some packets of a flow might pass through other nodes, e.g. MPTCP



# Buffering and Queue Management

- First In First Out (FIFO) → Drop Tail
  - packets served in arriving order
  - if queue is full, arriving packet is dropped
- Random Early Detection (RED)
  - drop earlier (function of buffer size)
  - or mark to signal congestion to end hosts
  - different traffic classes can be handled differently
- Multiple Traffic Classes
  - separate FIFO queue
    - for each flow or traffic class (e.g. voice, video)
    - need scheduler to decide serving order
- Active Queue Management (AQM)
  - queue autotunes itself to e.g. latency target
    - CoDel, PIE, FqCoDel,...
    - Packet Value based dropping

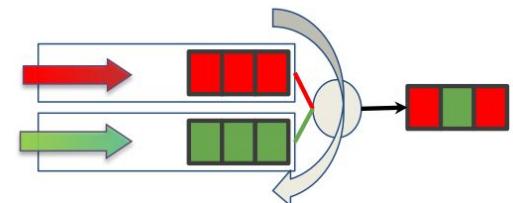
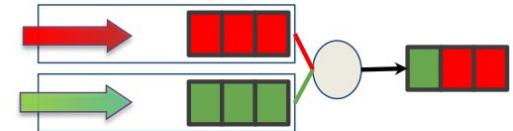


# Packet Marking and Traffic Classification

- **Mark a packet to signal downstream or end-hosts**
  - ECN - Early Congestion notification
  - Reuse some of the IP header fields, e.g. Type of Service (ToS) bits can use bufferstate → RED, CoDel,...
- **How to mark?**
  - End hosts based on applications
    - how can the network trust the endhosts?
  - Network nodes based on traffic classes
    - how can the network infer application requirements?
- **How to identify traffic classes?**
  - using flow specification based on five tuple
  - rate limitations, what conforms to profile, what is out of profile

# Packet Scheduling

- Determines the serving order of packets
  - when there are multiple queues to serve
  - multiple algorithms, different complexity
- Strict priority
  - assign to each queue a priority number
  - serve always the queue with highest priority first if it has packets
- Round Robin
  - go through queues in round robin way
  - if packet in the queue, serve, otherwise check next one
- Weighted Fair Scheduling
  - assign weights to queues
  - serve proportionally many packets



- WHAT ABOUT SDN?

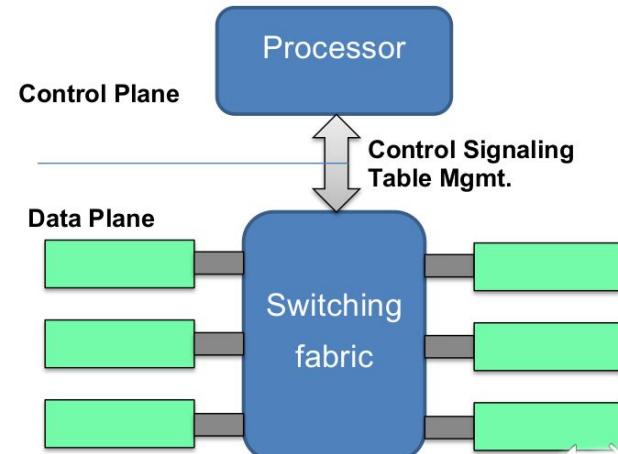
# SDN

- Control Plane

- calculates the forwarding table
- determines output port based on
- destination address

- Data Plane

- manages individual incoming packets
- matches destination address
  - switch: Dst MAC addr
  - routers: longest IP Prefix
- lookup the output port
- action: the packet is sent to output port
- switching fabric: directs packets from input to output



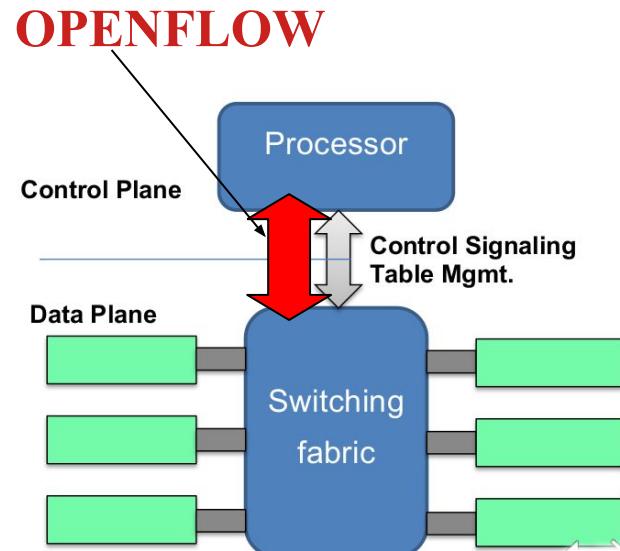
# SDN

- Control Plane

- calculates the forwarding table
  - determines output port based on
  - destination address

- Data Plane

- manages individual incoming packets
  - matches destination address
    - switch: Dst MAC addr
    - routers: longest IP Prefix
  - lookup the output port
  - action: the packet is sent to output port
  - switching fabric: directs packets from input to output



# OpenFlow

- Match

- Header fields
  - Subset
  - IP, MAC, MPLS
- Ports
- supported by most TCAM implementations

- Actions

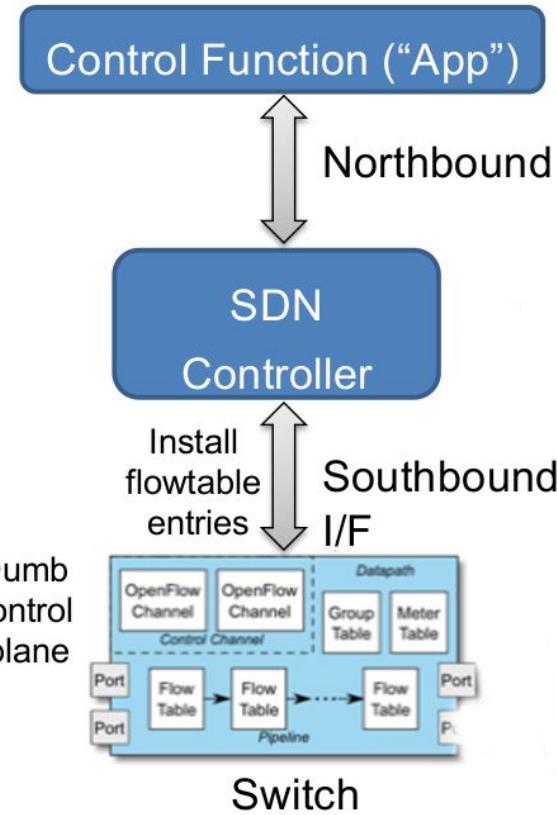
- If match found, perform action on
  - packet
    - drop
    - rewrite header fields
    - forward on port X
    - count packets
    - flood

Prio	Match	Action
1	Src=1.2.3.4, Dest=5.6.7.8	forward (1)
2	Dest=1.2.3.*	drop
3	Dest=1.2.3.8, Dport!=53	forward (2)
4	Src=1.2.3.7, Dport=100	forward (3)
5	Dport=100	forward controller

# OpenFlow and SDN

- Pros

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



# OpenFlow ... in the Beginning

- OpenFlow was simple
- A single rule table
  - Priority, pattern, actions, counters, timeouts
- Matching on any of 12 fields, e.g.,
  - MAC addresses
  - IP addresses
  - Transport protocol
  - Transport port numbers

# OpenFlow ... then

- Proliferation of header fields

Version	Date	# Headers
OF 1.0	Dec 2009	12
OF 1.1	Feb 2011	15
OF 1.2	Dec 2011	36
OF 1.3	Jun 2012	40
OF 1.4	Oct 2013	41

- Things are changing...

# Future SDN Switches

- Configurable packet parser
  - Not tied to a specific header format
- Flexible match+action tables
  - Multiple tables (in series and/or parallel)
  - Able to match on all defined fields
- General packet-processing primitives
  - Copy, add, remove, and modify
  - For both header fields and meta-data

# Need Hardware

- Configurable switch needs a programmable switch
- Programmable switches “always” existed
- Issues
  - Hard to program
  - Lack of standard
  - Lack of common interface
  - Definitely not performant
  - Definitely not supported

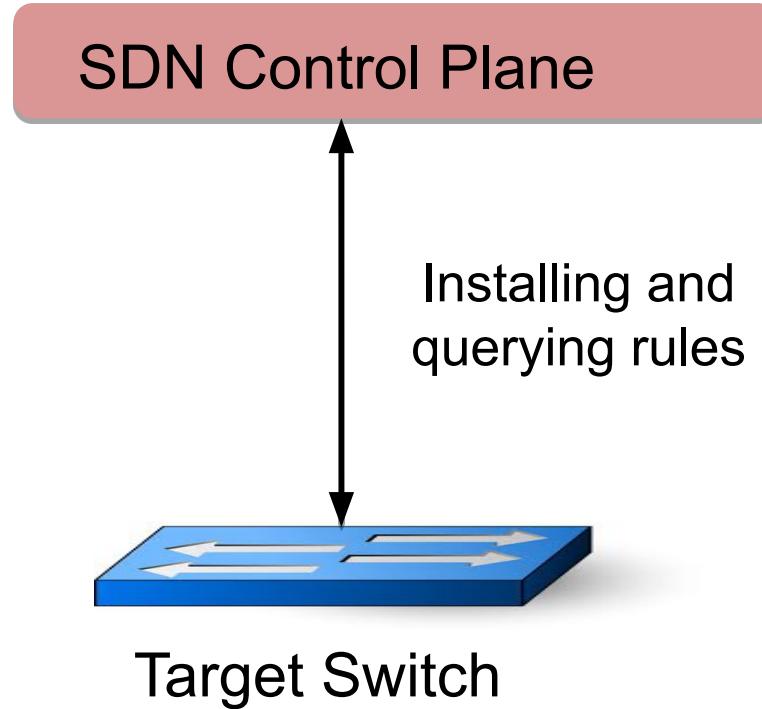
# Need Hardware

- Things changed
- New generation of switch ASICs
  - Intel FlexPipe
  - Cisco Doppler
  - Tofino
- Pros
  - Performances:
    - <http://conferences.sigcomm.org/sigcomm/2013/papers/sigcomm/p99.pdf>
- Cons
  - Still need a standard interface.
    - Custom, vendor-specific interfaces
    - Low-level, akin to microcode programming

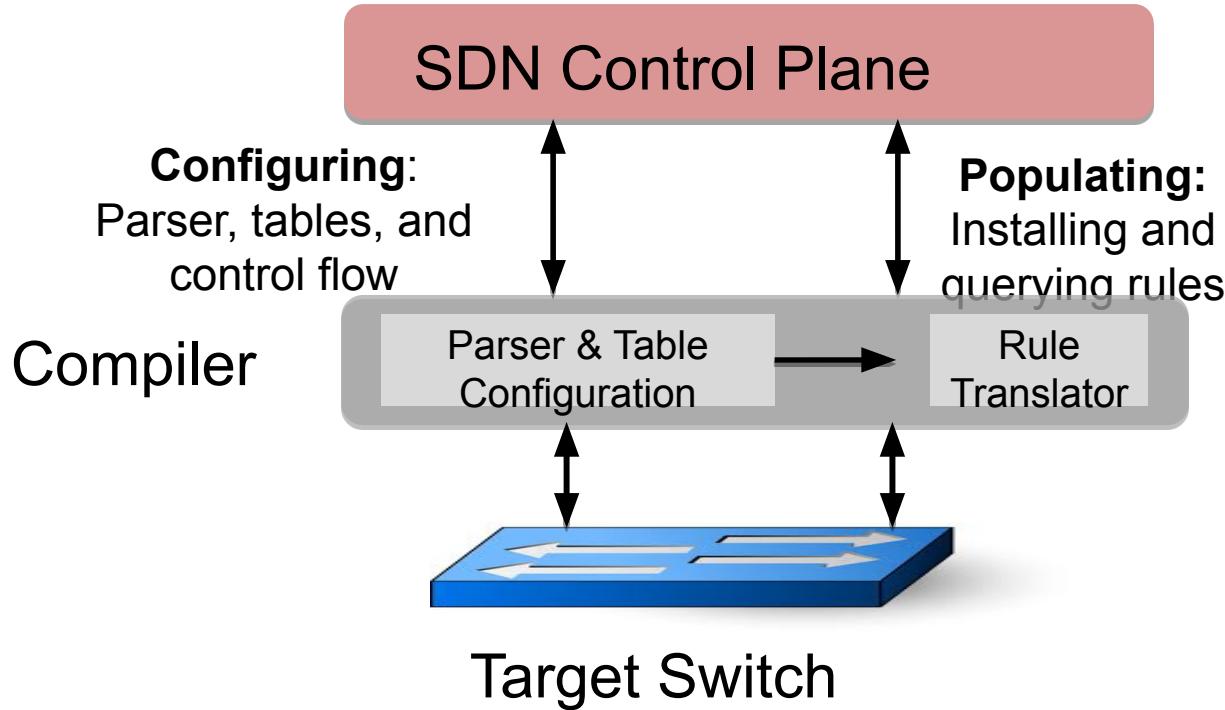
# Need a Programmable Interface

- Three goals
- 1) Protocol independence
  - Configure a packet parser
  - Define a set of typed match+action tables
- 2) Target independence
  - Program without knowledge of switch details
  - Rely on compiler to configure the target switch
- 3) Reconfigurability
  - Change parsing and processing in the field

# OpenFlow



# New OpenFlow



# **What is Data Plane Programming?**

- **Why program the Data Plane?**

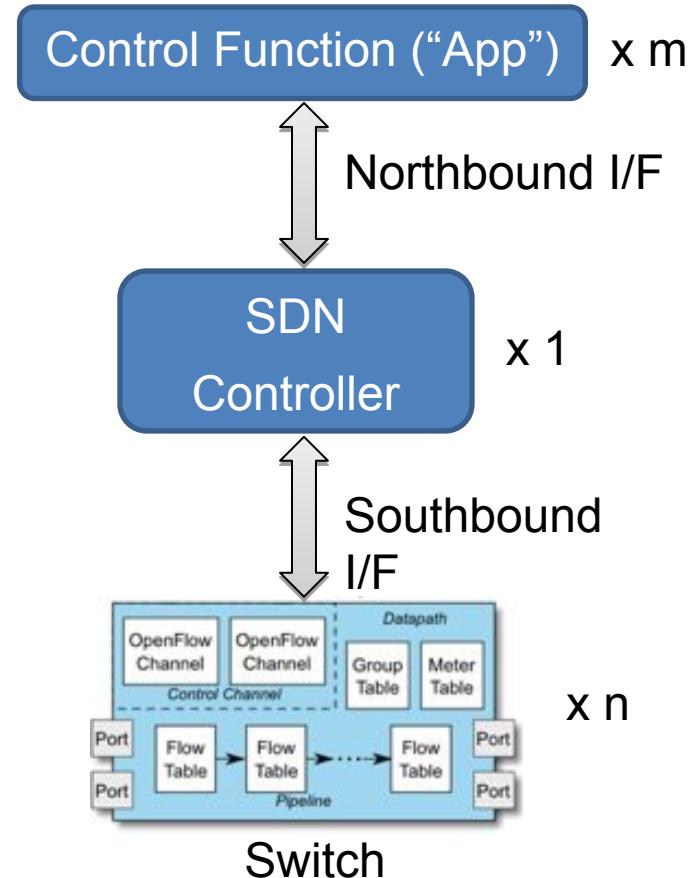
# Software Defined Networking (SDN)

- **Main contributions**

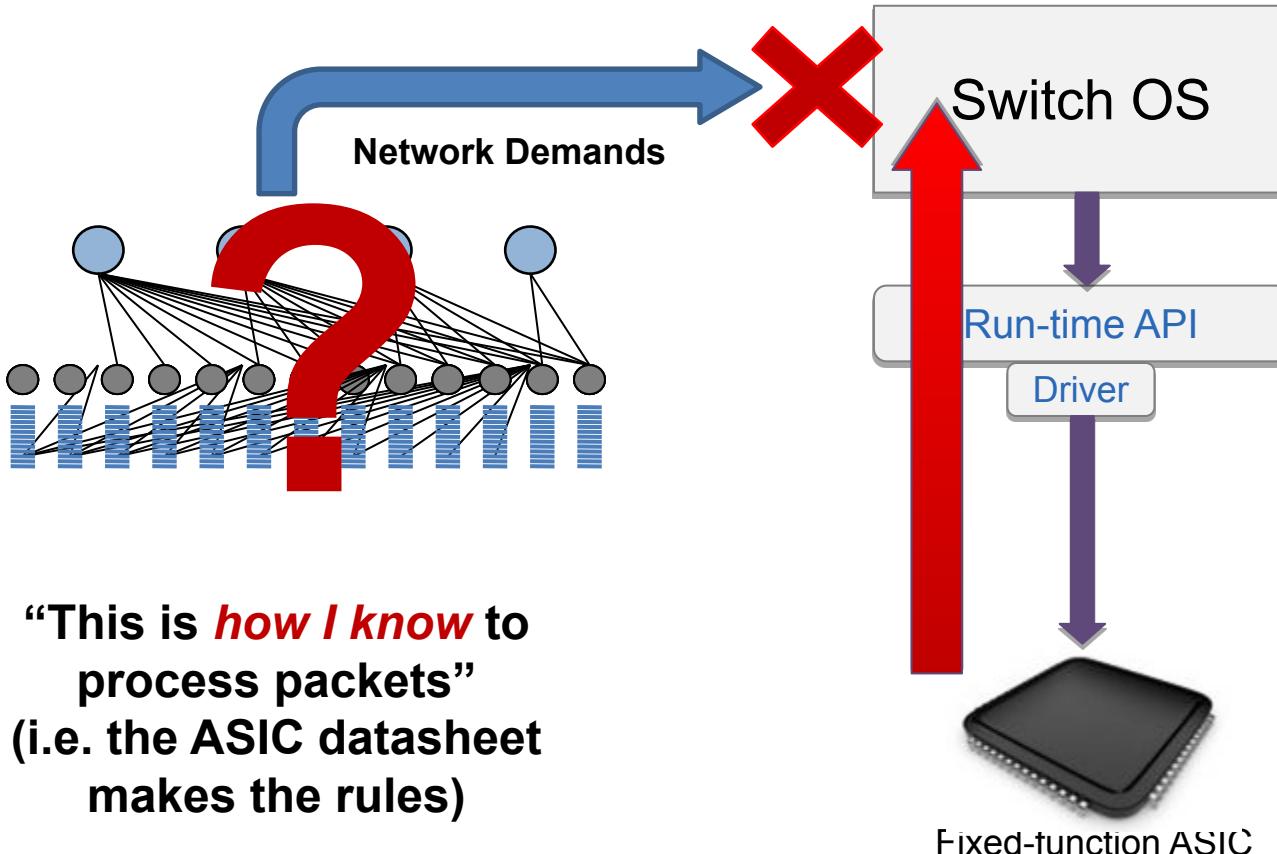
- OpenFlow = standardized *protocol* to interact with switch
  - download flow table entries, query statistics, etc.
- OpenFlow = standardized *model*
  - match/action abstraction
- *Concept of logically centralized control via a single entity (“SDN controller”)*
  - Simplifies control plane

- **Issues**

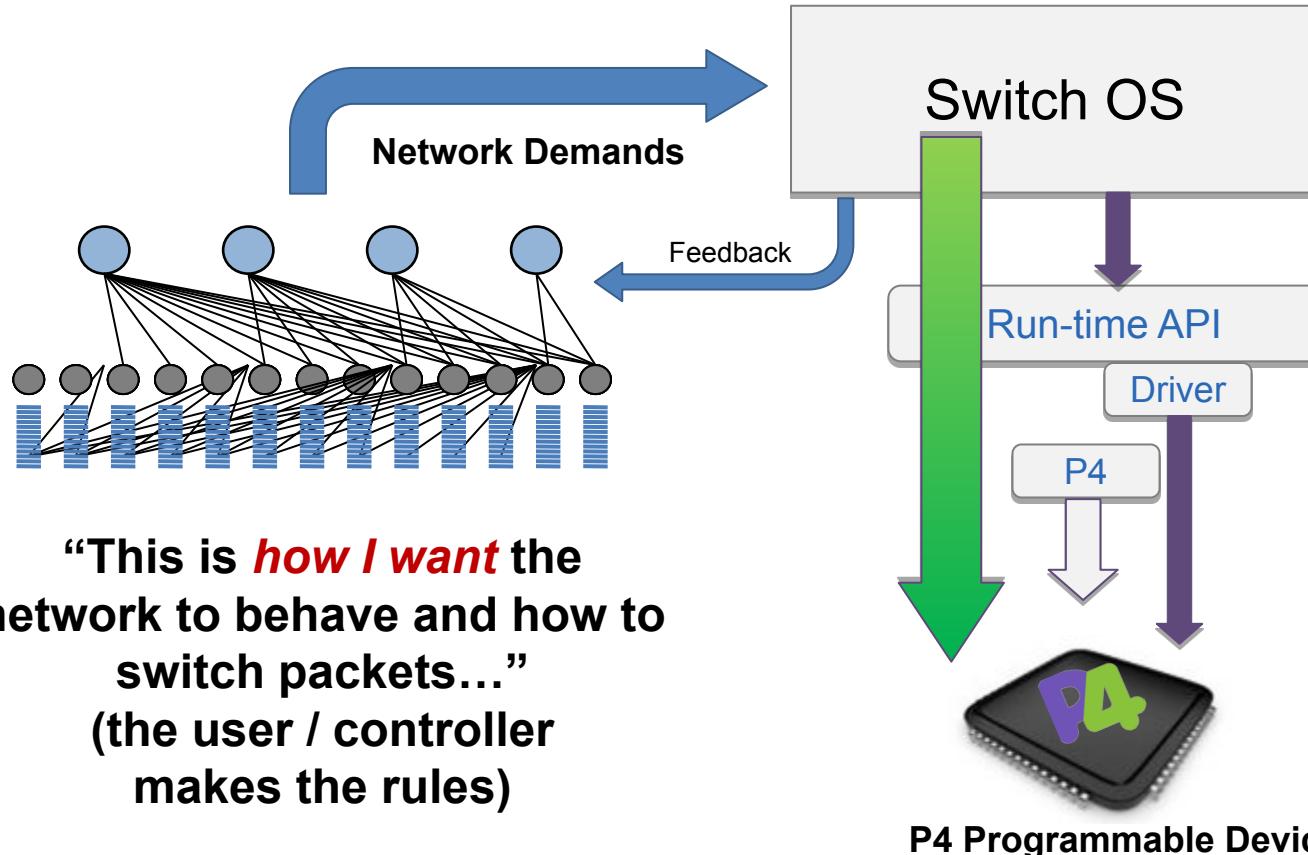
- Data-plane protocol evolution requires changes to standards (12 → 40 OpenFlow match fields)
- Limited interoperability between vendors (OpenFlow / netconf / JSON / XML variants)
- Limited programmability



# 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]
- In-Network caching and coordination – NetCache[4] / NetChain[5]
- Aggregation for MapReduce Applications [7]
- ... and much more

[1] Miao, Rui, et al. "SilkRoad: Making Stateful Layer-4 Load Balancing Fast and Cheap Using Switching ASICs." SIGCOMM, 2017.

[2] Handley, Mark, et al. "Re-architecting datacenter networks and stacks for low latency and high performance." SIGCOMM, 2017.

[3] Kim, Changhoon, et al. "In-band network telemetry via programmable dataplanes." SIGCOMM. 2015.

[4] Xin Jin et al. "NetCache: Balancing Key-Value Stores with Fast In-Network Caching." To appear at SOSP 2017

[5] Jin, Xin, et al. "NetChain: Scale-Free Sub-RTT Coordination." NSDI, 2018.

[6] Dang, Huynh Tu, et al. "NetPaxos: Consensus at network speed." SIGCOMM, 2015.

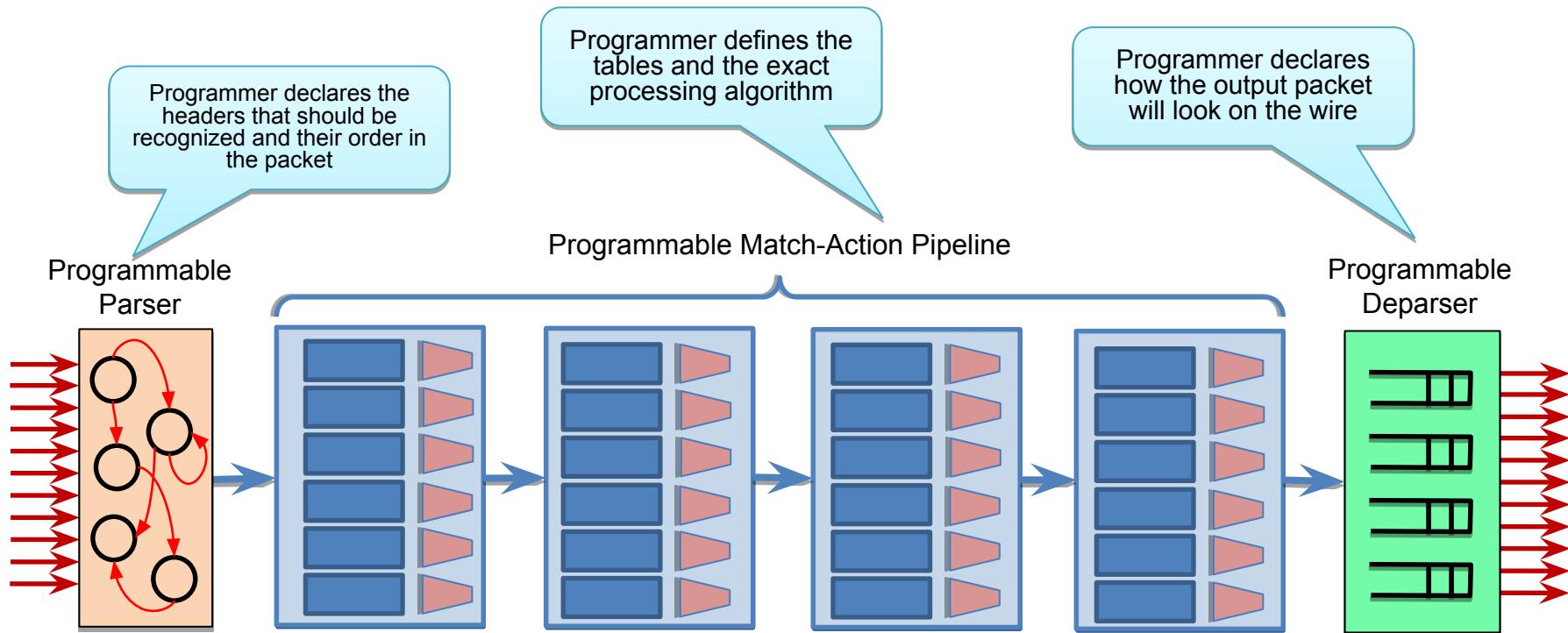
[7] Sapiro, Amedeo, et al. "In-Network Computation is a Dumb Idea Whose Time Has Come." *Hot Topics in Networks*. ACM, 2017.

# Brief History and Trivia

- May 2013: Initial idea and the name “P4”
- July 2014: First paper (SIGCOMM CCR)
- Aug 2014: First P4<sub>14</sub> Draft Specification (v0.9.8)
- Sep 2014: P4<sub>14</sub> Specification released (v1.0.0)
- Jan 2015: P4<sub>14</sub> v1.0.1
- Mar 2015: P4<sub>14</sub> v1.0.2
- Nov 2016: P4<sub>14</sub> v1.0.3
- May 2017: P4<sub>14</sub> v1.0.4
- Apr 2016: P4<sub>16</sub> – first commits
- Dec 2016: First P4<sub>16</sub> Draft Specification
- May 2017: P4<sub>16</sub> Specification released

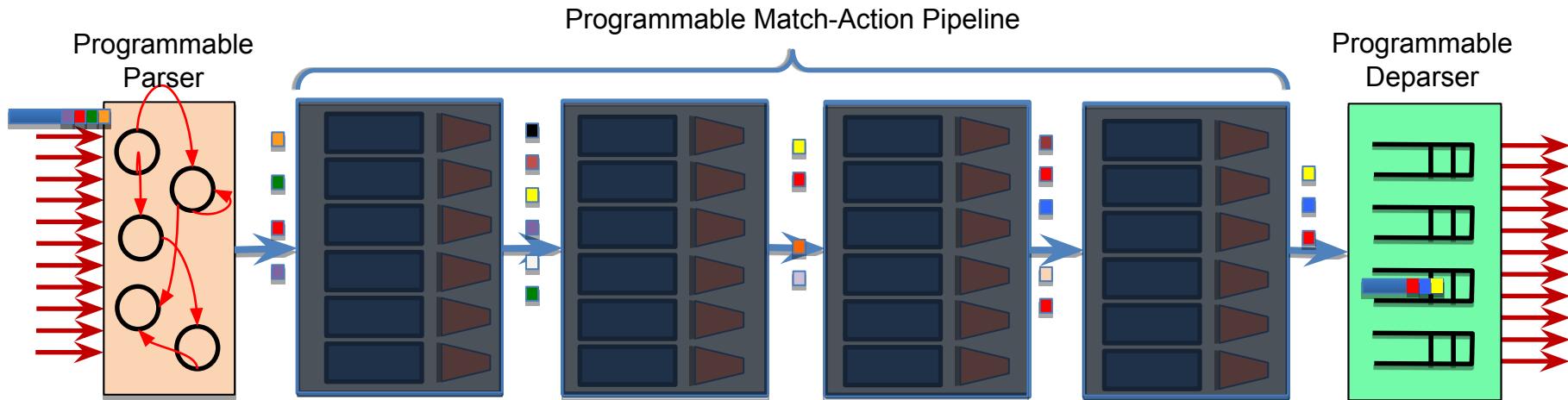
# **P4\_16 Data Plane Model**

# PISA: Protocol-Independent Switch Architecture

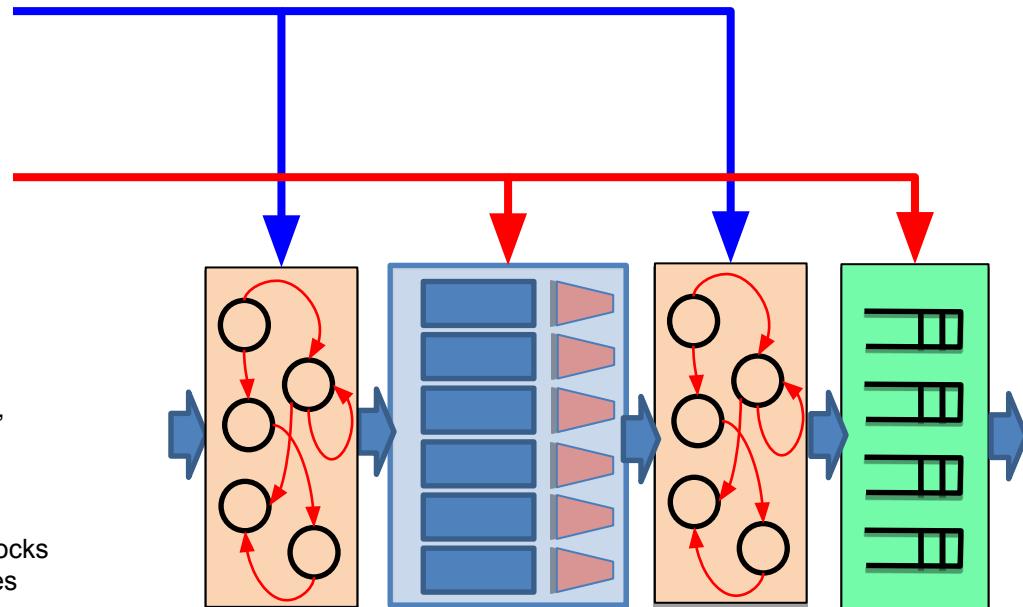


# PISA in Action

- 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)

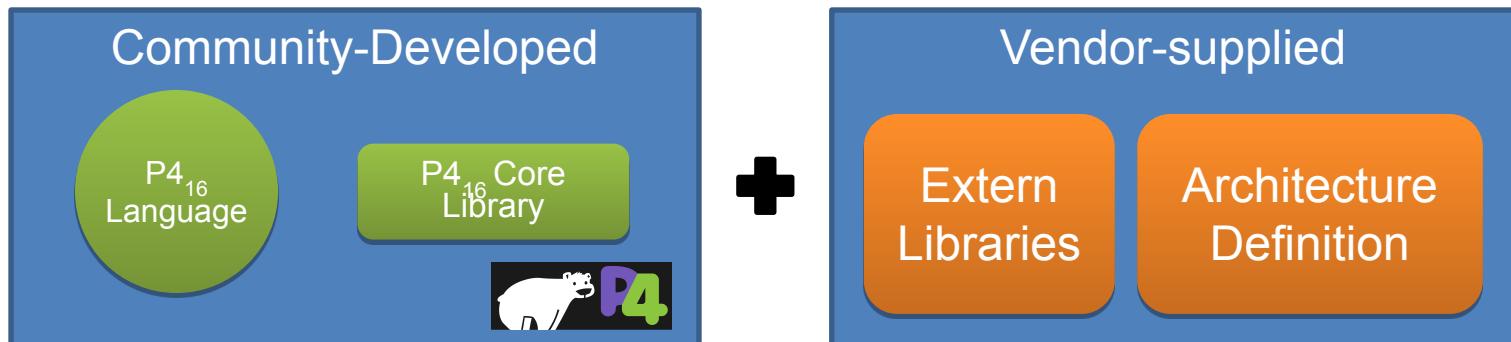


# P4<sub>16</sub> Language Elements

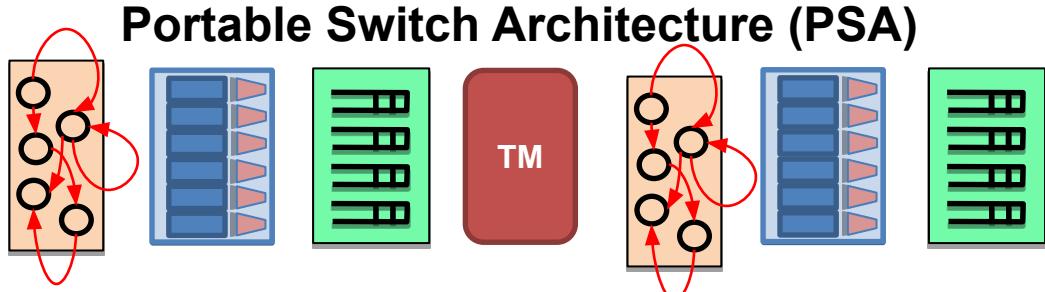
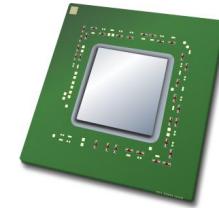
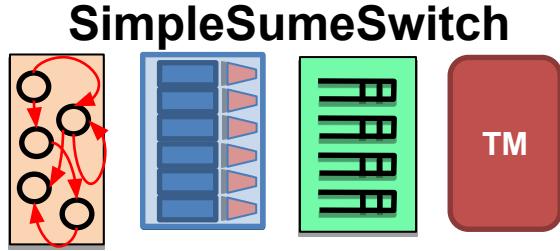
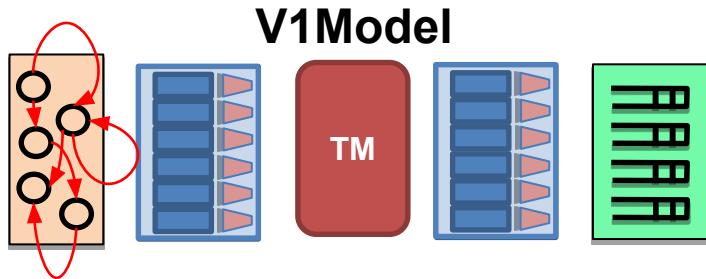


# P4\_16 Approach

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

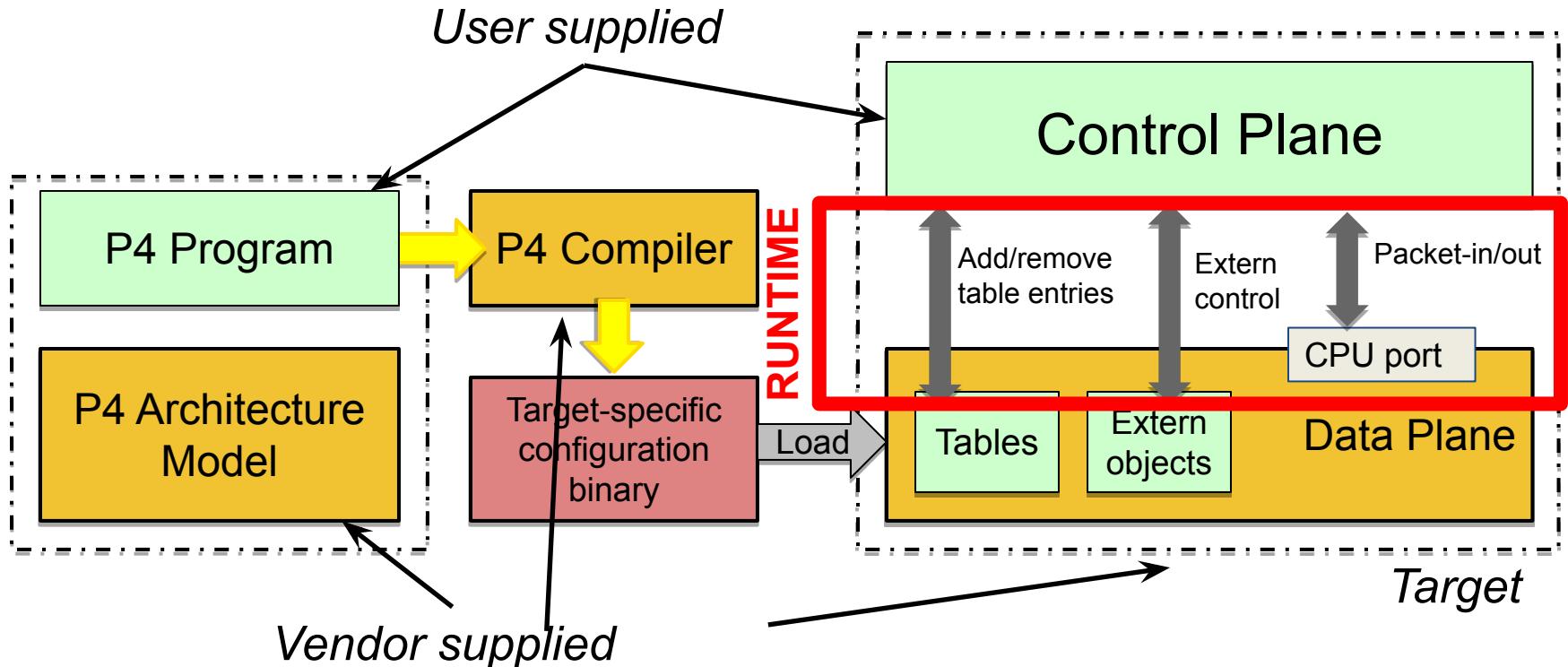


# Example Architectures and Targets



Anything

# Programming a P4 Target



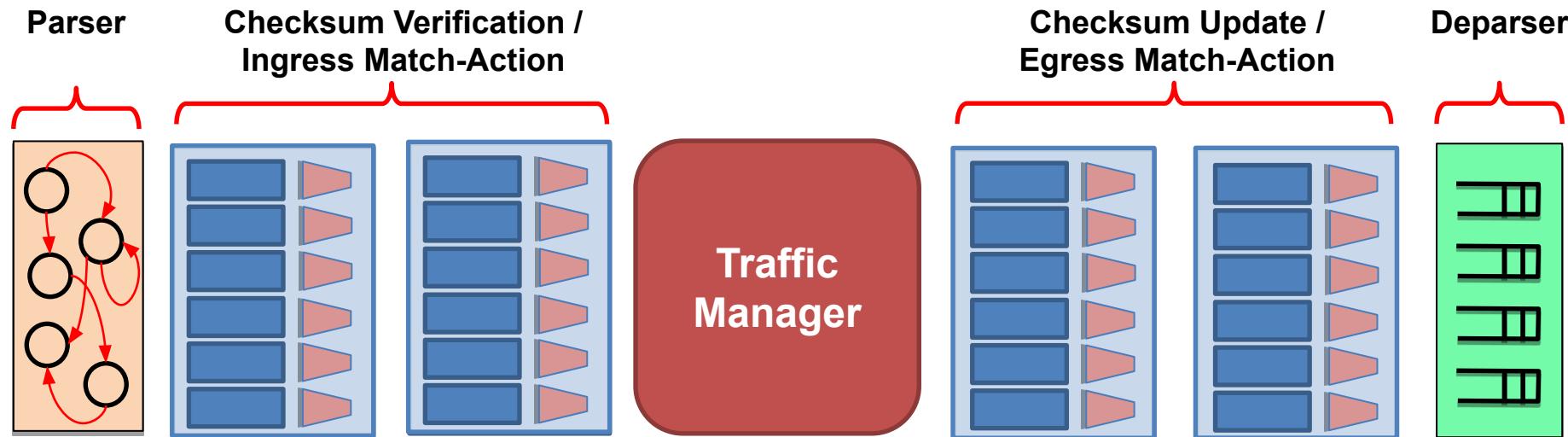
# **Lab 1: Basics**

# **Before we start...**

- **Install VM image**
- **Exercises under folder**
  - **/vt19/..**

# 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 that the packet will be sent out of (read only in egress pipeline)

# P4<sub>16</sub> 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<sub>16</sub> 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<sub>16</sub> 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;
    }
}






```

```
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 {   }
}

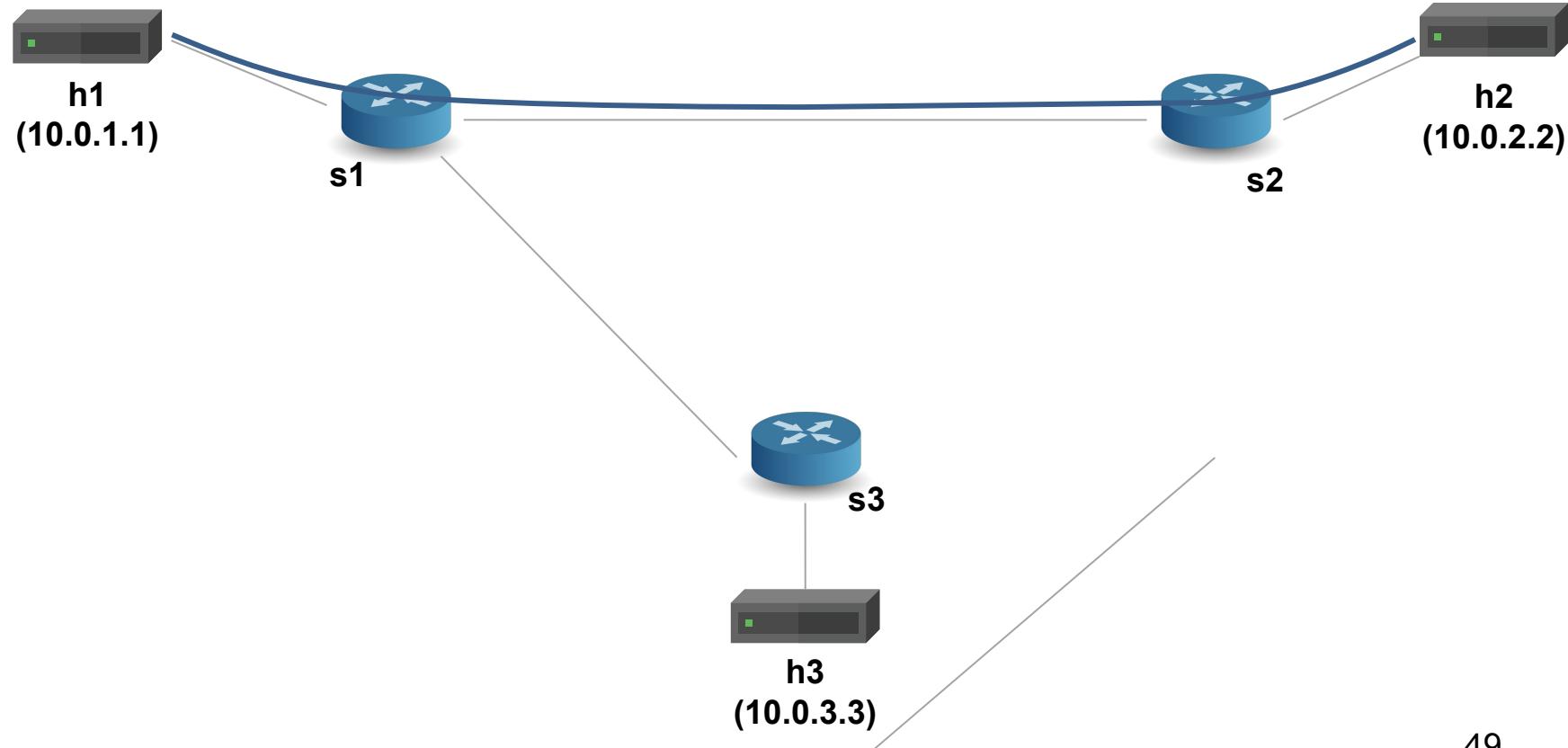
V1Switch( MyParser(), MyVerifyChecksum(), MyIngress(),
MyEgress(), MyComputeChecksum(), MyDeparser() ) main;
```

Key	Action ID	Action Data
1	set_egress_spec ID	2
2	set_egress_spec ID	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<sub>16</sub>
- 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<sub>16</sub> 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 ( $\geq 2$ )
- **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<sub>16</sub> 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;
    ...
}

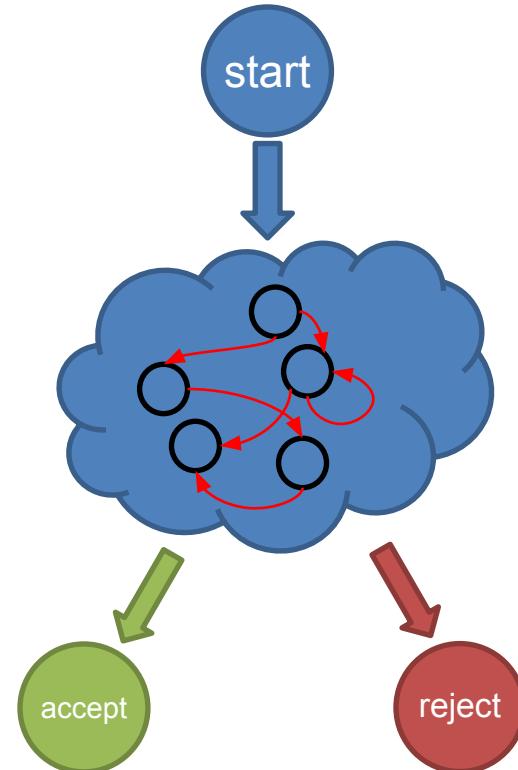
/* User program */
struct metadata {
    ...
}
struct headers {
    ethernet_t   ethernet;
    ipv4_t        ipv4;
}
```

## Other useful types

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

# P4<sub>16</sub> Parsers

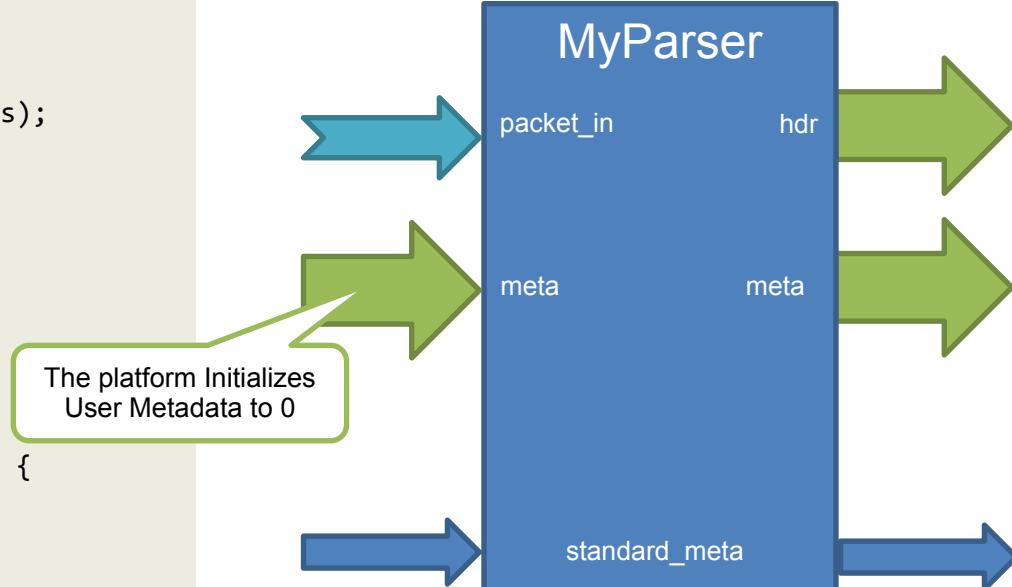
- **Parsers are functions that map packets into headers and metadata, written in a state machine style**
- **Every parser has three predefined states**
  - start
  - accept
  - reject
- **Other states may be defined by the programmer**
- **In each state, execute zero or more statements, and then transition 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_ip4;  
        default: accept;  
    }  
}
```

P4<sub>16</sub> 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<sub>16</sub> 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) {
    /* 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;
    }

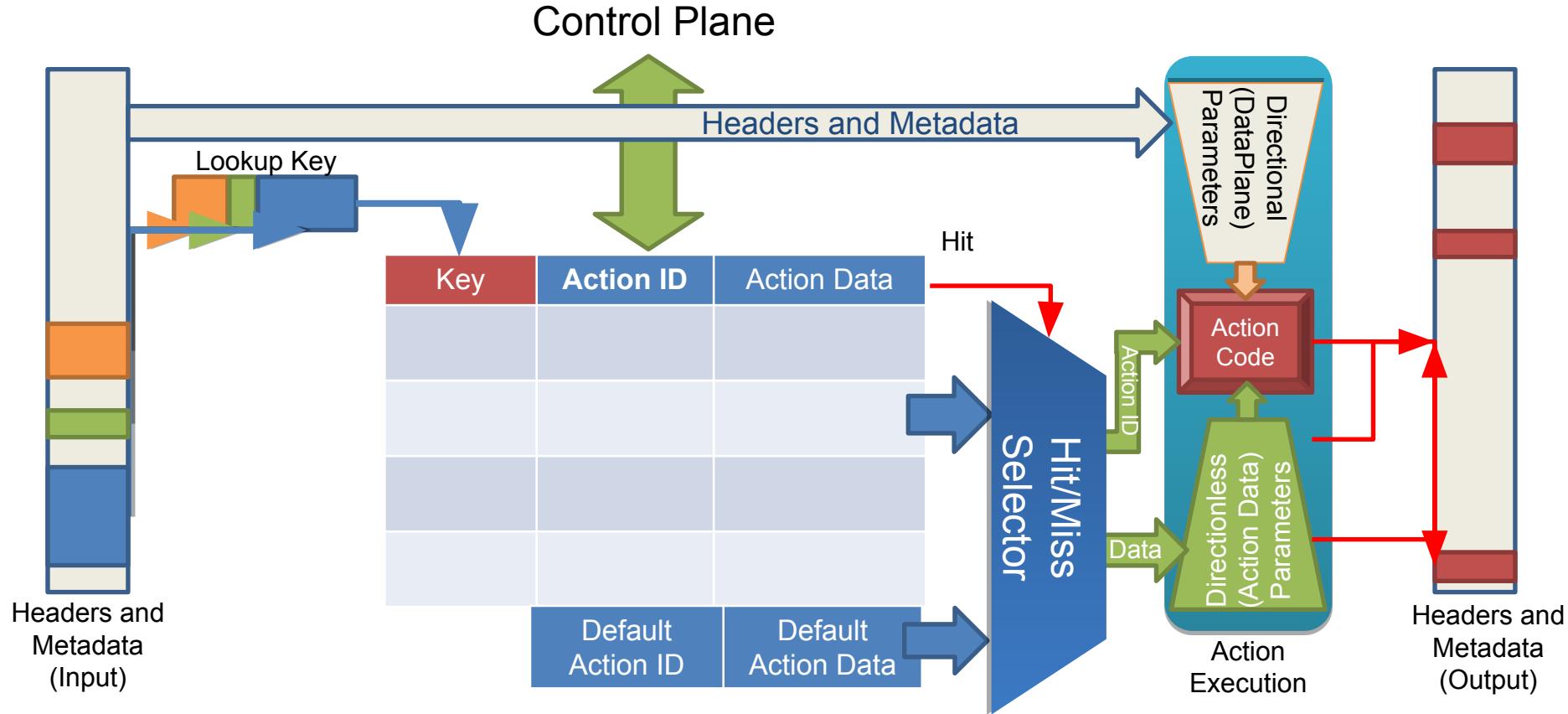
    apply {
        swap_mac(hdr.ethernet.srcAddr,
                  hdr.ethernet.dstAddr);
        std_meta.egress_spec = std_meta.ingress_port;
    }
}
```

- **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: ++

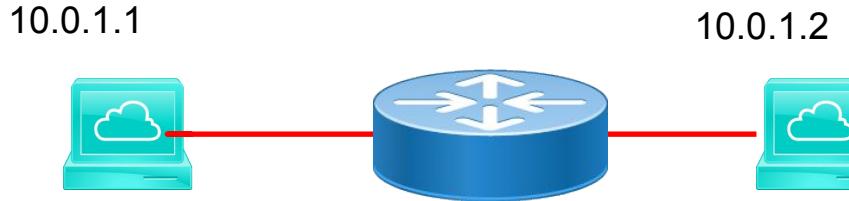
# P4<sub>16</sub> Tables

- **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();
        ...
    }
}
```

# P4<sub>16</sub> 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

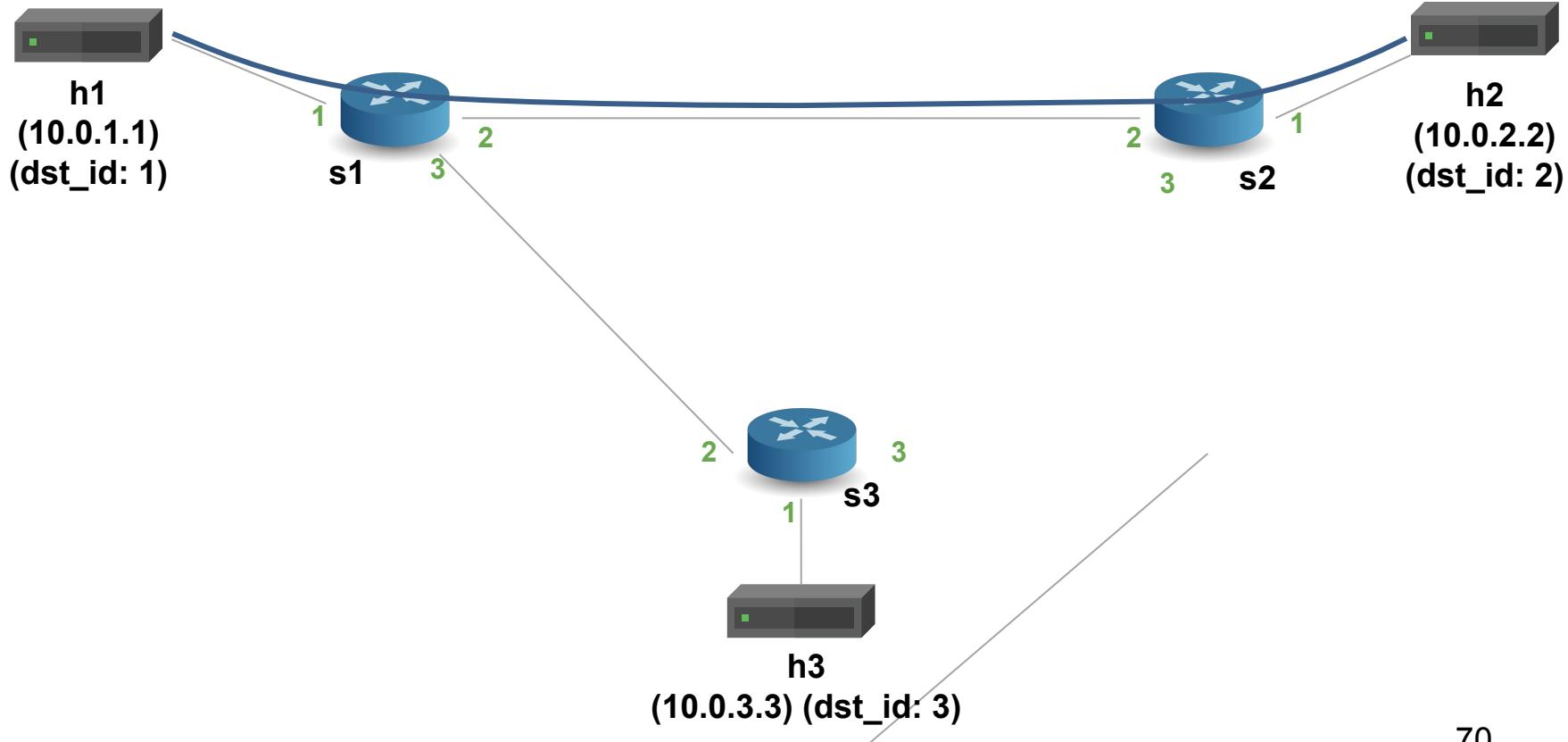
# Basic Tunneling

- 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



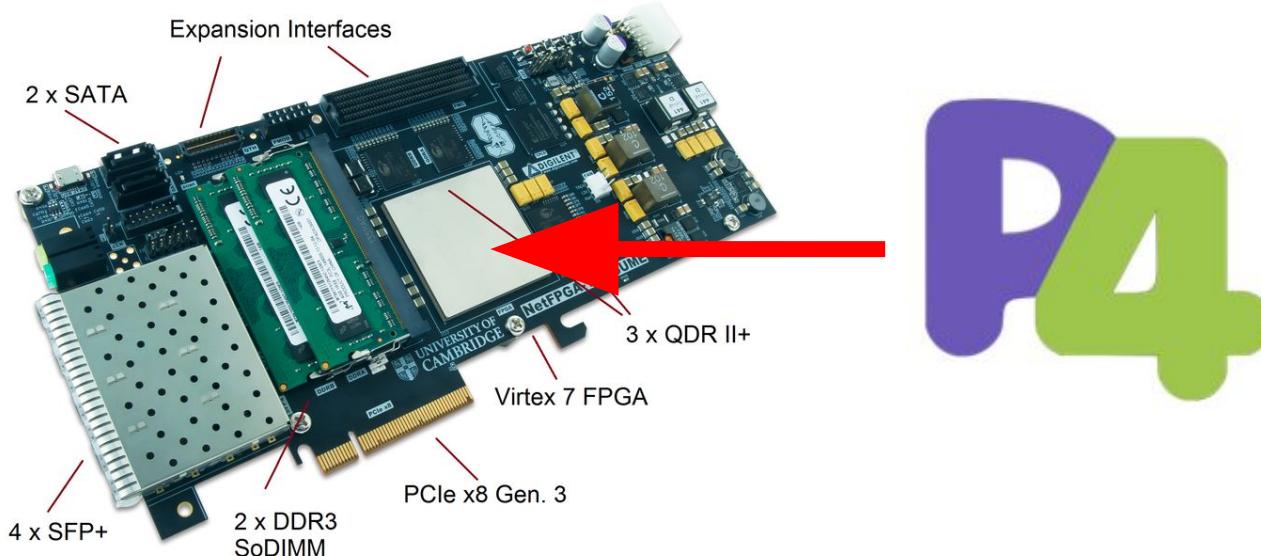
# Coding Break

# 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)
- **What happens upon reaching the `reject` state of the parser?**
  - Architecture dependent
- **How much of the packet can I parse?**
  - Architecture dependent

# P4→NetFPGA

- Prototype and evaluate P4 programs in real hardware!
- 4x10G network interfaces
- Special price for academic users :)
- <https://github.com/NetFPGA/P4-NetFPGA-public/wiki>



P4

# Debugging

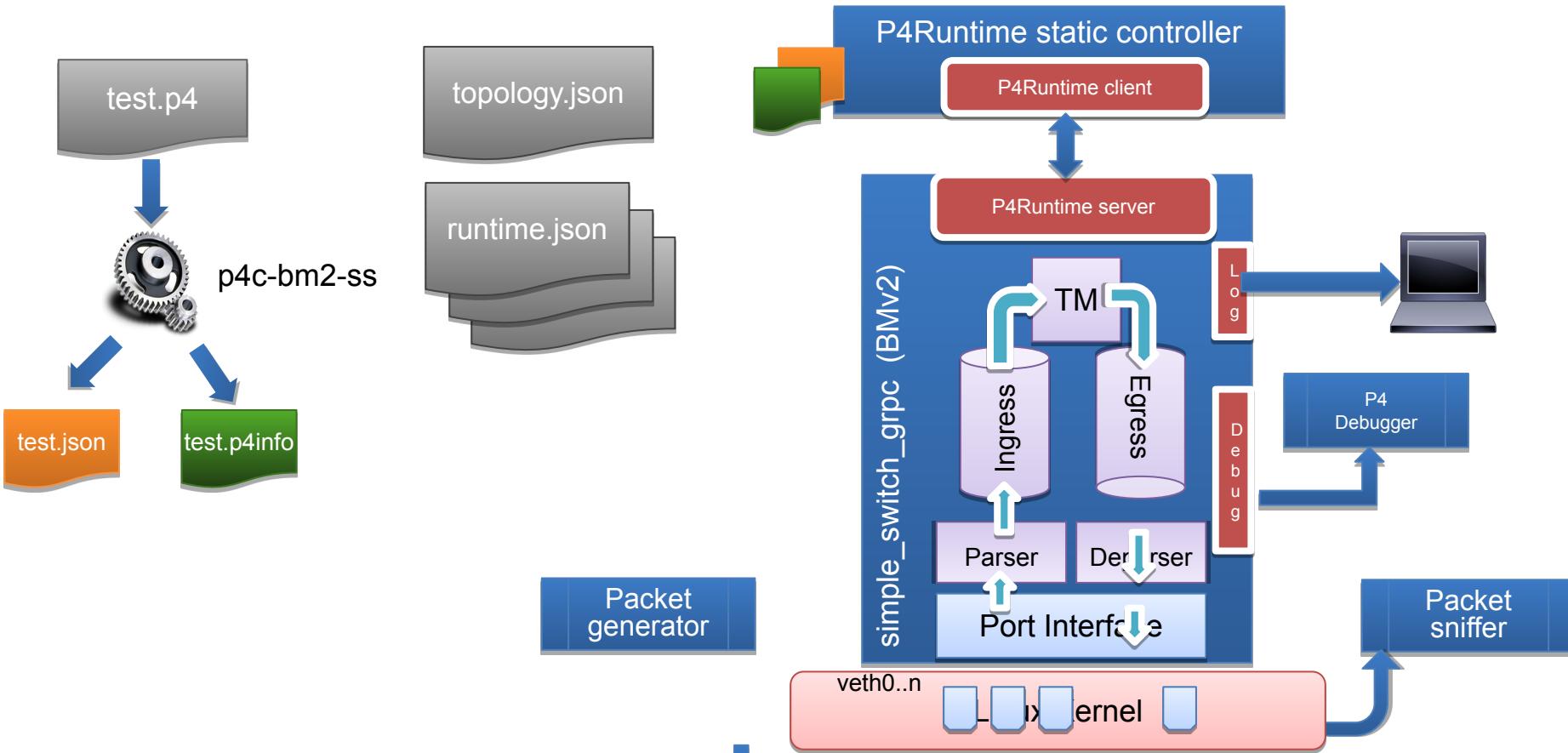
```
control MyIngress(...) {
    table debug {
        key = {
            std_meta.egress_spec : exact;
        }
        actions = { }
    }
    apply {
        ...
        debug.apply();
    }
}
```

- **Bmv2 maintains logs that keep track of how packets are processed in detail**
  - /tmp/p4s.s1.log
  - /tmp/p4s.s2.log
  - /tmp/p4s.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

# **Lab 2: P4Runtime**

# P4 Software Tools

# Makefile: under the hood



# Makefile: under the hood (in pseudocode)

```
P4C_ARGS = --p4runtime-file $(basename $@).p4info  
          --p4runtime-format text  
RUN_SCRIPT = ../../utils/run_exercise.py  
TOPO = topology.json
```

## dirs:

```
mkdir -p build pcaps logs
```

## build: for each P4 program, generate BMv2 json file

```
p4c-bm2-ss --p4v 16 $(P4C_ARGS) -o $@ $<
```

## run: build, then [default target]

```
sudo python $(RUN_SCRIPT) -t $(TOPO)
```

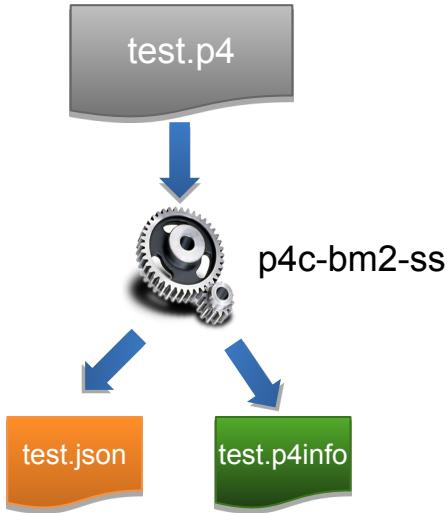
## stop: sudo mn -c

## clean: stop, then

```
rm -f *.pcap
```

```
rm -rf build pcaps logs
```

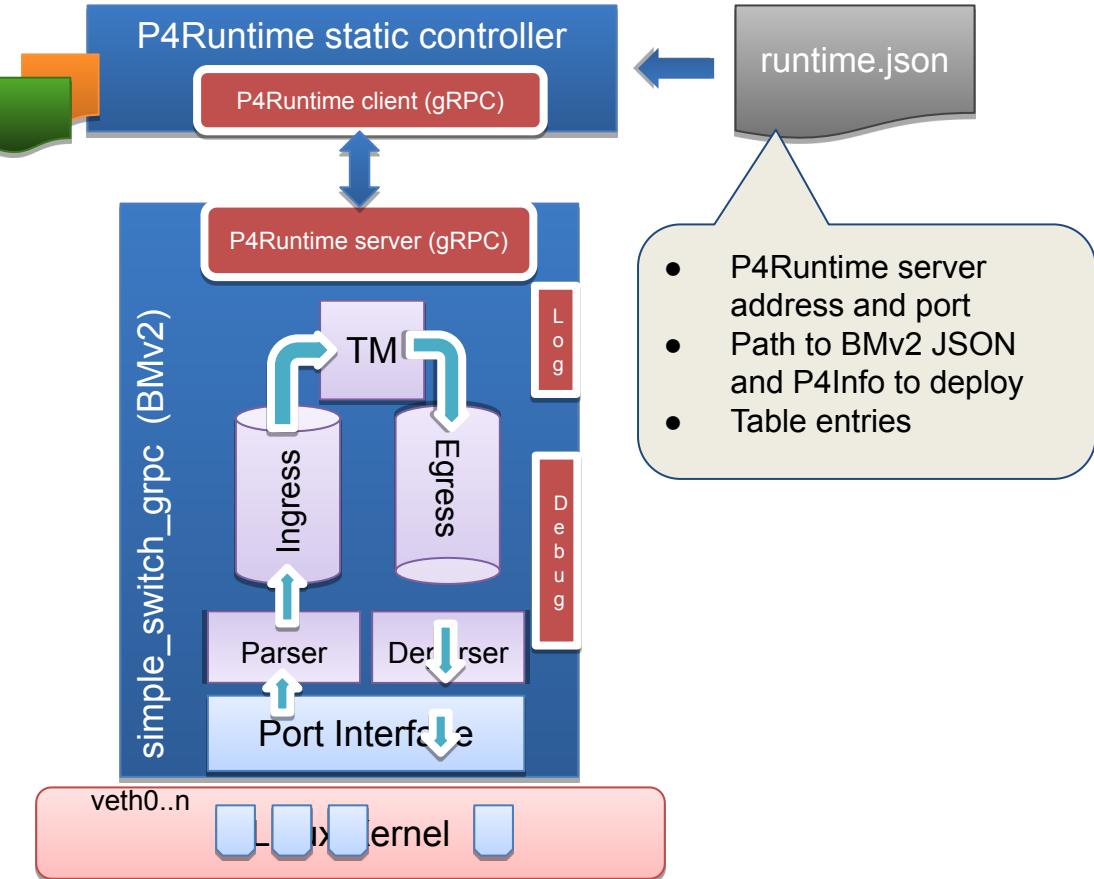
# Step 1: P4 Program compilation [build phase]



```
$ p4c-bm2-ss --p4v 16 \
-o test.json \
--p4runtime-file test.p4info \
--p4runtime-format text \
test.p4
```

## Step 2: run\_exercise.py [run phase]

- Create network based on topology.json
- Start simple\_switch\_grpc instance for each switch
- Use P4Runtime to push the P4 program (P4Info and BMv2 JSON)
- Add the static rules defined in runtime.json



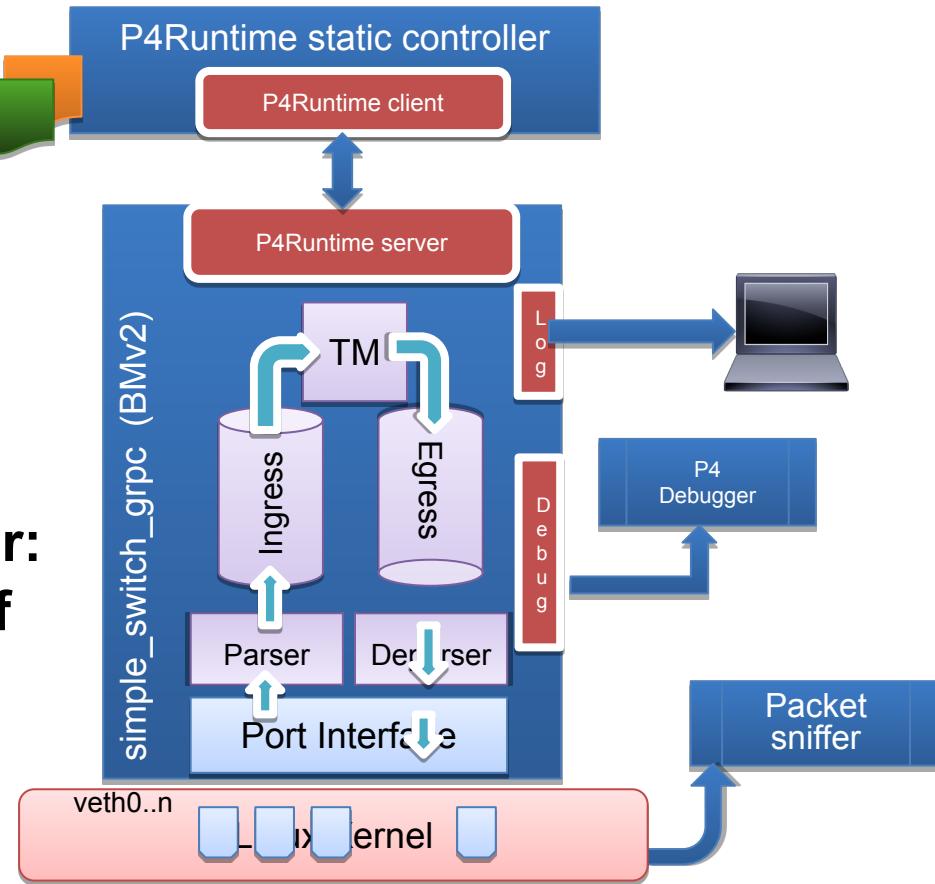
# Step 3: Run the traffic generator and sniffer

In some exercises, this is  
send.py and receive.py

In others, we use standard  
Linux programs, like ping

In the p4runtime exercise,  
we also run our own controller:  
mycontroller.py instead of  
the static one

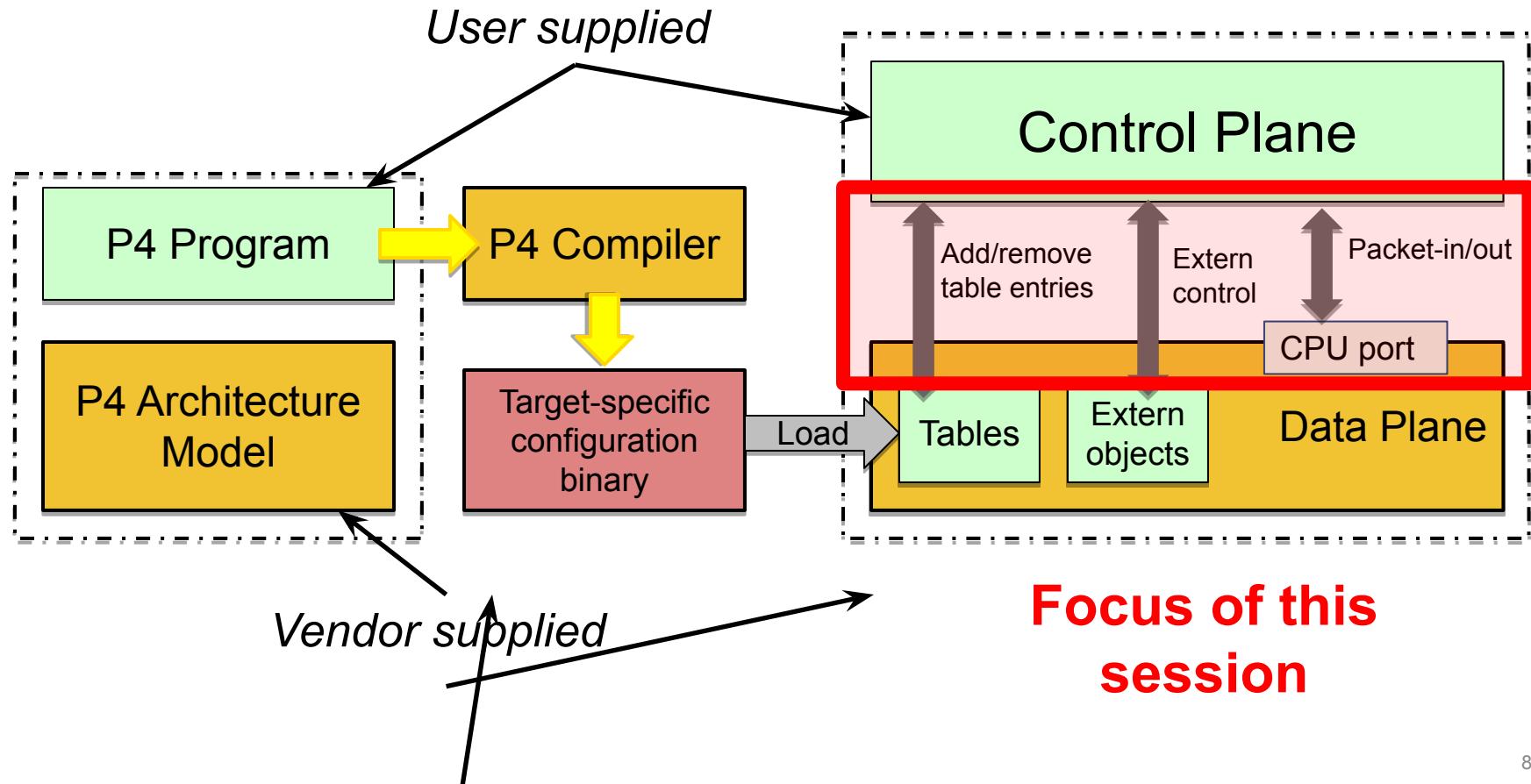
Packet  
generator



# P4Runtime

- API overview
- Workflow
- Exercise - Tunneling
- Why P4Runtime?

# Runtime control of P4 data planes



# Existing approaches to runtime control

- **P4 compiler auto-generated runtime APIs**
  - Program-dependent -- hard to provision new P4 program without restarting the control plane!
- **BMv2 CLI**
  - Program-independent, but target-specific -- control plane not portable!
- **OpenFlow**
  - Target-independent, but protocol-dependent -- protocol headers and actions baked in the specification!
- **OCP Switch Abstraction Interface (SAI)**
  - Target-independent, but protocol-dependent

# Why do we need another data plane control API?

HOW STANDARDS PROLIFERATE:  
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)

SITUATION:  
THERE ARE  
14 COMPETING  
STANDARDS.

14?! RIDICULOUS!  
WE NEED TO DEVELOP  
ONE UNIVERSAL STANDARD  
THAT COVERS EVERYONE'S  
USE CASES.

YEAH!



SOON:

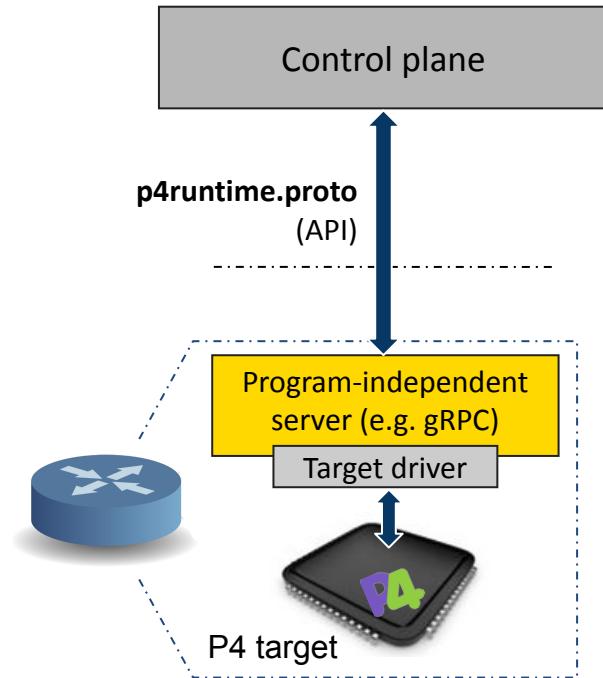
SITUATION:  
THERE ARE  
15 COMPETING  
STANDARDS.

# Properties of a runtime control API

API	Target-independent	Protocol-independent
P4 compiler auto-generated	✓	✗
BMv2 CLI	✗	✓
OpenFlow	✓	✗
SAI	✓	✗
P4Runtime	✓	✓

# What is P4Runtime?

- **Framework for runtime control of P4 targets**
  - Open-source API + server implementation
    - <https://github.com/p4lang/PI>
  - Initial contribution by Google and Barefoot
- **Work-in-progress by the p4.org API WG**
  - Draft of version 1.0 available
- **Protobuf-based API definition**
  - p4runtime.proto
  - gRPC transport
- **P4 program-independent**
  - API doesn't change with the P4 program
- **Enables field-reconfigurability**
  - Ability to push new P4 program without recompiling the software stack of target switches



# Protocol Buffers Basics

- Language for describing data for serialization in a structured way
- Common binary wire-format
- Language-neutral
  - Code generators for: *Action Script, C, C++, C#, Clojure, Lisp, D, Dart, Erlang, Go, Haskell, Java, Javascript, Lua, Objective C, OCaml, Perl, PHP, Python, Ruby, Rust, Scala, Swift, Visual Basic, ...*
- Platform-neutral
- Extensible and backwards compatible
- Strongly typed

```
syntax = "proto3";

message Person {
    string name = 1;
    int32 id = 2;
    string email = 3;

    enum PhoneType {
        MOBILE = 0;
        HOME = 1;
        WORK = 2;
    }

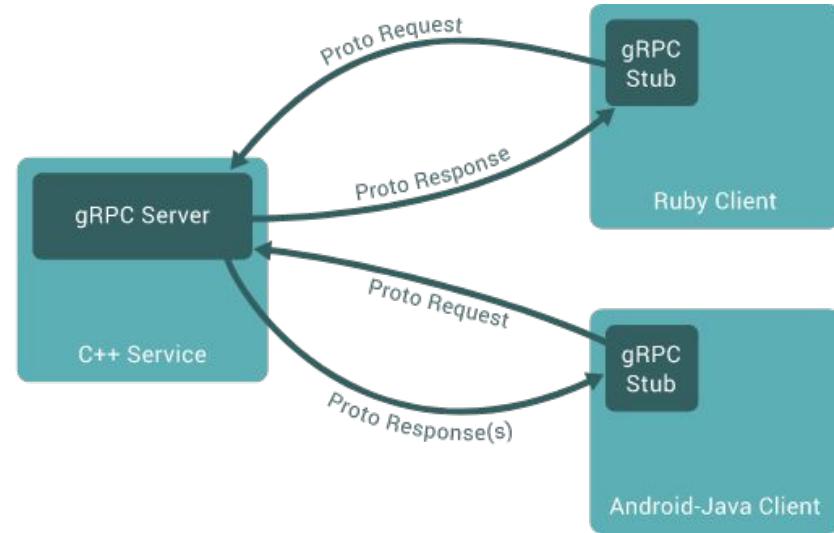
    message PhoneNumber {
        string number = 1;
        PhoneType type = 2;
    }

    repeated PhoneNumber phone = 4;
}
```

# gRPC Basics

- Use Protocol Buffers to define service API and messages
- Automatically generate native stubs in:

- C / C++
- C#
- Dart
- Go
- Java
- Node.js
- PHP
- Python
- Ruby



- Transport over HTTP/2.0 and TLS
  - Efficient single TCP connection implementation that supports bidirectional streaming

# gRPC Service Example

```
// The greeter service definition.  
service Greeter {  
    // Sends a greeting  
    rpc SayHello (HelloRequest) returns (HelloReply) {}  
}  
  
// The request message containing the user's name.  
message HelloRequest {  
    string name = 1;  
}  
  
// The response message containing the greetings  
message HelloReply {  
    string message = 1;  
}
```

More details here: <https://grpc.io/docs/guides/>

# P4Runtime Service

**Enables a local or remote entity to load the pipeline/program, send/receive packets, and read and write forwarding table entries, counters, and other chip features.**

```
service P4Runtime {
    rpc Write(WriteRequest) returns (WriteResponse) {}
    rpc Read(ReadRequest) returns (stream ReadResponse) {}
    rpc SetForwardingPipelineConfig(SetForwardingPipelineConfigRequest)
        returns (SetForwardingPipelineConfigResponse) {}
    rpc GetForwardingPipelineConfig(GetForwardingPipelineConfigRequest)
        returns (GetForwardingPipelineConfigResponse) {}
    rpc StreamChannel(stream StreamMessageRequest)
        returns (stream StreamMessageResponse) {}
}
```

# P4Runtime Service

**Protobuf Definition:**

<https://github.com/p4lang/PI/blob/master/proto/p4/v1/p4runtime.proto>

**Service Specification:**

***Working draft of version 1.0 is available now***

<https://p4.org/p4-spec/docs/P4Runtime-v1.0.0.pdf>

# P4Runtime Write Request

```
message WriteRequest {  
    uint64 device_id = 1;  
    uint64 role_id = 2;  
    Uint128 election_id = 3;  
    repeated Update updates = 4;  
}
```

```
message Update {  
    enum Type {  
        UNSPECIFIED = 0;  
        INSERT = 1;  
        MODIFY = 2;  
        DELETE = 3;  
    }  
    Type type = 1;  
    Entity entity = 2;
```

```
message Entity {  
    oneof entity {  
        ExternEntry extern_entry = 1;  
        TableEntry table_entry = 2;  
        ActionProfileMember  
            action_profile_member = 3;  
        ActionProfileGroup  
            action_profile_group = 4;  
        MeterEntry meter_entry = 5;  
        DirectMeterEntry direct_meter_entry = 6;  
        CounterEntry counter_entry = 7;  
        DirectCounterEntry direct_counter_entry = 8;  
        PacketReplicationEngineEntry  
            packet_replication_engine_entry = 9;  
        ValueSetEntry value_set_entry = 10;  
        RegisterEntry register_entry = 11;  
    }  
}
```

# P4Runtime Table Entry

p4runtime.proto simplified excerpts:

```
message TableEntry {  
    uint32 table_id;  
    repeated FieldMatch match;  
    Action action;  
    int32 priority;  
    ...  
}
```

```
message Action {  
    uint32 action_id;  
    message Param {  
        uint32 param_id;  
        bytes value;  
    }  
    repeated Param params;  
}
```

```
message FieldMatch {  
    uint32 field_id;  
    message Exact {  
        bytes value;  
    }  
    message Ternary {  
        bytes value;  
        bytes mask;  
    }  
    ...  
    oneof field_match_type {  
        Exact exact;  
        Ternary ternary;  
        ...  
    }  
}
```

Full protobuf definition:

<https://github.com/p4lang/PI/blob/master/proto/p4/p4runtime.proto>

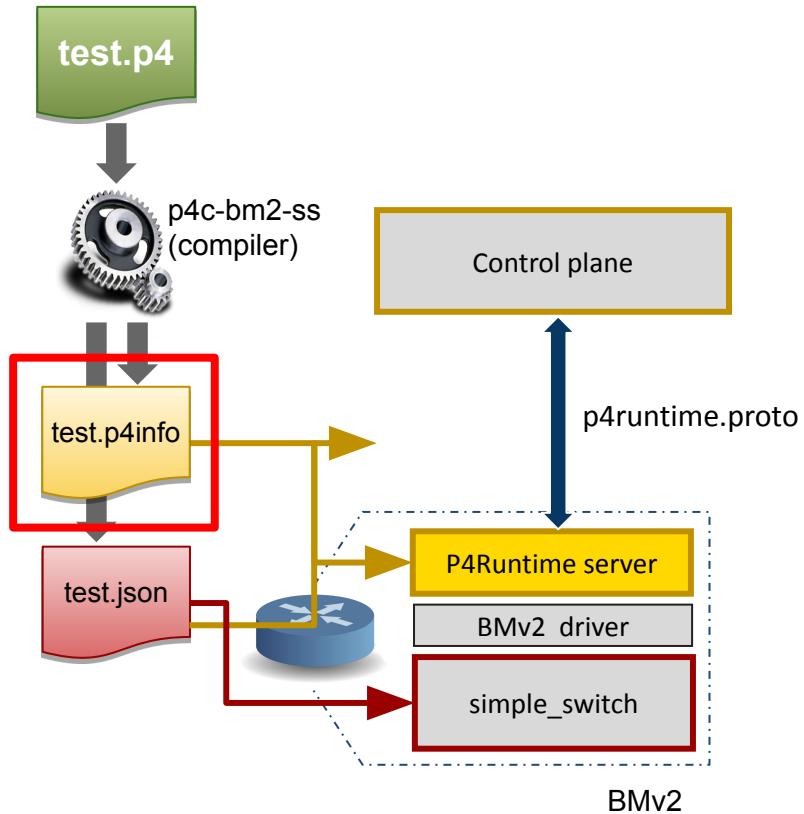
To add a table entry, the control plane needs to know:

- **IDs of P4 entities**
  - Tables, field matches, actions, params, etc.
- **Field matches for the particular table**
  - Match type, bitwidth, etc.
- **Parameters for the particular action**
- **Other P4 program attributes**

# P4Runtime workflow

## P4Info

- **Captures P4 program attributes needed at runtime**
  - IDs for tables, actions, params, etc.
  - Table structure, action parameters, etc.
- **Protobuf-based format**
- **Target-independent compiler output**
  - Same P4Info for BMv2, ASIC, etc.



Full P4Info protobuf specification:

<https://github.com/p4lang/PI/blob/master/proto/p4/config/v1/p4info.proto>

# P4Info example

## basic\_router.p4

```
...
action ipv4_forward(bit<48> dstAddr,
                    bit<9> port) {
    /* Action implementation */
}

...
table ipv4_lpm {
    key = {
        hdr.ipv4.dstAddr: lpm;
    }
    actions = {
        ipv4_forward;
        ...
    }
    ...
}
```

## basic\_router.p4info

```
actions {
    id: 16786453
    name: "ipv4_forward"
    params {
        id: 1
        name: "dstAddr"
        bitwidth: 48
        ...
        id: 2
        name: "port"
        bitwidth: 9
    }
}
...
tables {
    id: 33581985
    name: "ipv4_lpm"
    match_fields {
        id: 1
        name: "hdr.ipv4.dstAddr"
        bitwidth: 32
        match_type: LPM
    }
    action_ref_id: 16786453
}
```



P4 compiler

# P4Runtime Table Entry Example

## basic\_router.p4

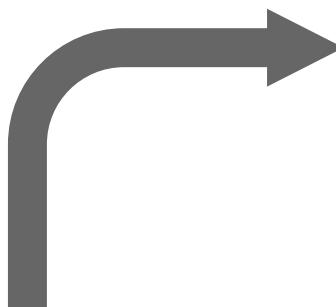
```
action ipv4_forward(bit<48> dstAddr,  
                    bit<9> port) {  
    /* Action implementation */  
}  
  
table ipv4_lpm {  
    key = {  
        hdr.ipv4.dstAddr: lpm;  
    }  
    actions = {  
        ipv4_forward;  
        ...  
    }  
    ...  
}
```



Logical view of table entry

```
hdr.ipv4.dstAddr=10.0.1.1/32  
-> ipv4_forward(00:00:00:00:00:10, 7)
```

Control plane  
generates

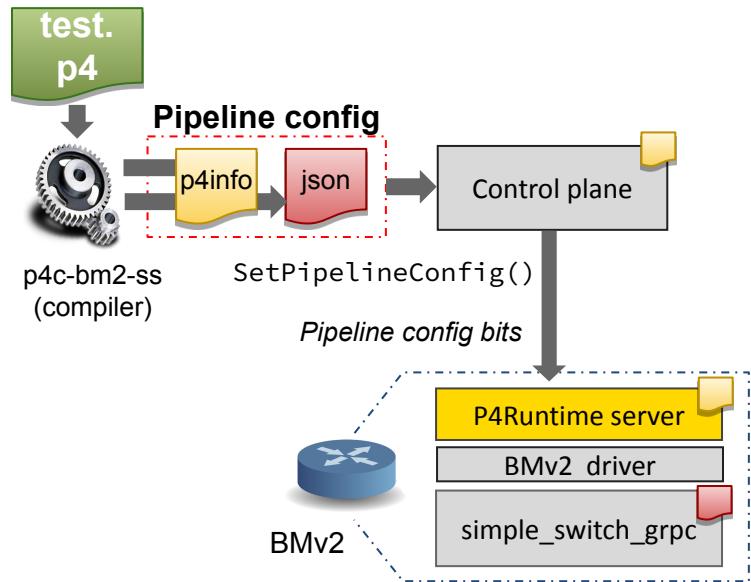


## Protobuf message

```
table_entry {  
    table_id: 33581985  
    match {  
        field_id: 1  
        lpm {  
            value: "\n\000\001\001"  
            prefix_len: 32  
        }  
    }  
    action {  
        action_id: 16786453  
        params {  
            param_id: 1  
            value: "\000\000\000\000\000\n"  
        }  
        params {  
            param_id: 2  
            value: "\000\007"  
        }  
    }  
}
```

# P4Runtime SetPipelineConfig

```
message SetForwardingPipelineConfigRequest {  
    enum Action {  
        UNSPECIFIED = 0;  
        VERIFY = 1;  
        VERIFY_AND_SAVE = 2;  
        VERIFY_AND_COMMIT = 3;  
        COMMIT = 4;  
        RECONCILE_AND_COMMIT = 5;  
    }  
    uint64 device_id = 1;  
    uint64 role_id = 2;  
    Uint128 election_id = 3;  
    Action action = 4;  
    ForwardingPipelineConfig config = 5;  
}
```



```
message ForwardingPipelineConfig {  
    config.P4Info p4info = 1;  
    // Target-specific P4 configuration.  
    bytes p4_device_config = 2;  
}
```

# P4Runtime StreamChannel

```
message StreamMessageRequest {  
    oneof update {  
        MasterArbitrationUpdate  
            arbitration = 1;  
        PacketOut packet = 2;     
    }  
}
```

```
// Packet sent from the controller to the switch.  
  
message PacketOut {  
    bytes payload = 1;  
    // This will be based on P4 header annotated as  
    // @controller_header("packet_out").  
    // At most one P4 header can have this annotation.  
    repeated PacketMetadata metadata = 2;  
}
```

```
message StreamMessageResponse {  
    oneof update {  
        MasterArbitrationUpdate  
            arbitration = 1;  
        PacketIn packet = 2;     
    }  
}
```

```
// Packet sent from the switch to the controller.  
  
message PacketIn {  
    bytes payload = 1;  
    // This will be based on P4 header annotated as  
    // @controller_header("packet_in").  
    // At most one P4 header can have this annotation.  
    repeated PacketMetadata metadata = 2;  
}
```

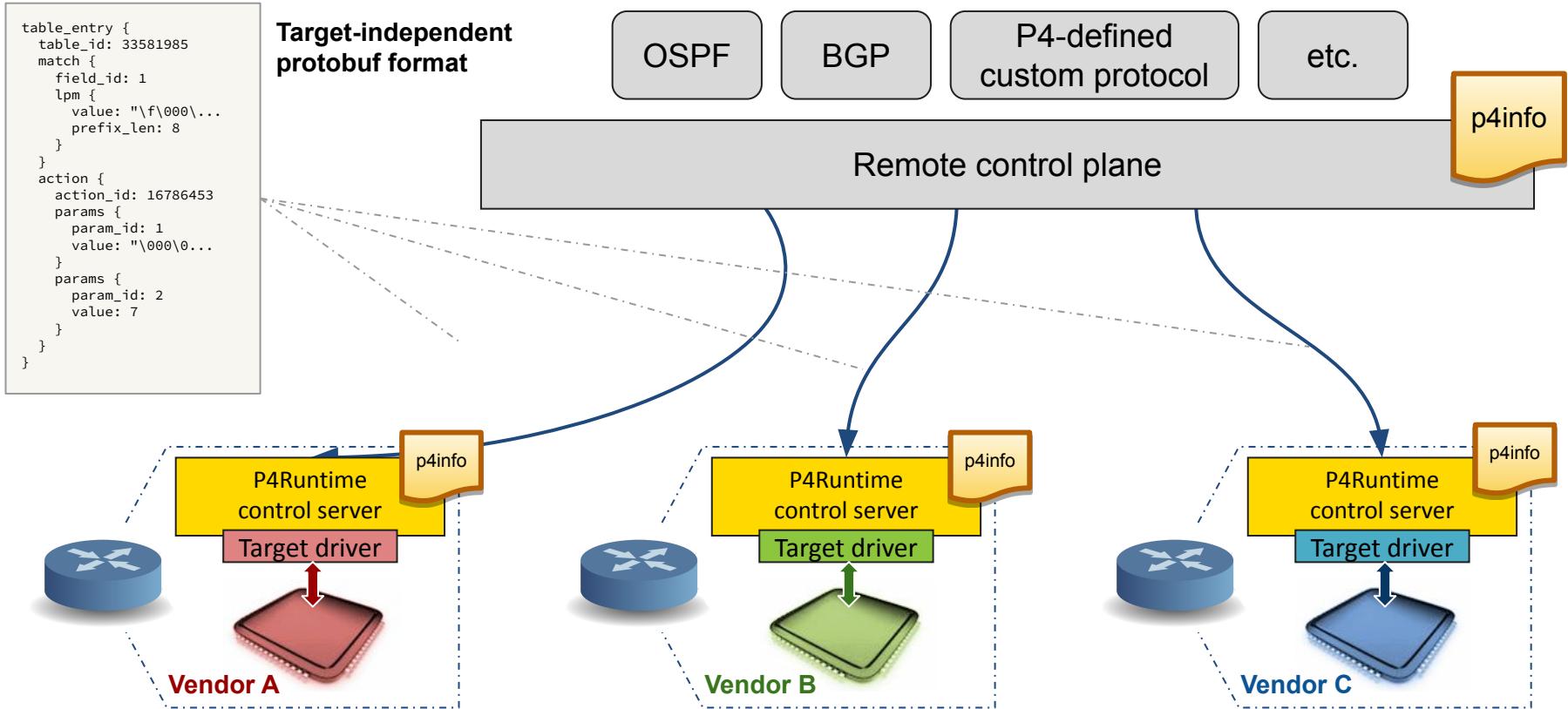
# P4Runtime Common Parameters

- **device\_id**
  - Specifies the specific forwarding chip or software bridge
  - **Set to 0 for single chip platforms**
- **role\_id**
  - Corresponds to a role with specific capabilities (i.e. what operations, P4 entities, behaviors, etc. are in the scope of a given role)
  - Role definition is currently agreed upon between control and data planes offline
  - **Default role\_id (0) has full pipeline access**
- **election\_id**
  - P4Runtime supports mastership on a per-role basis
  - Client with the highest election ID is referred to as the "master", while all other clients are referred to as "slaves"
  - **Set to 0 for single instance controllers**

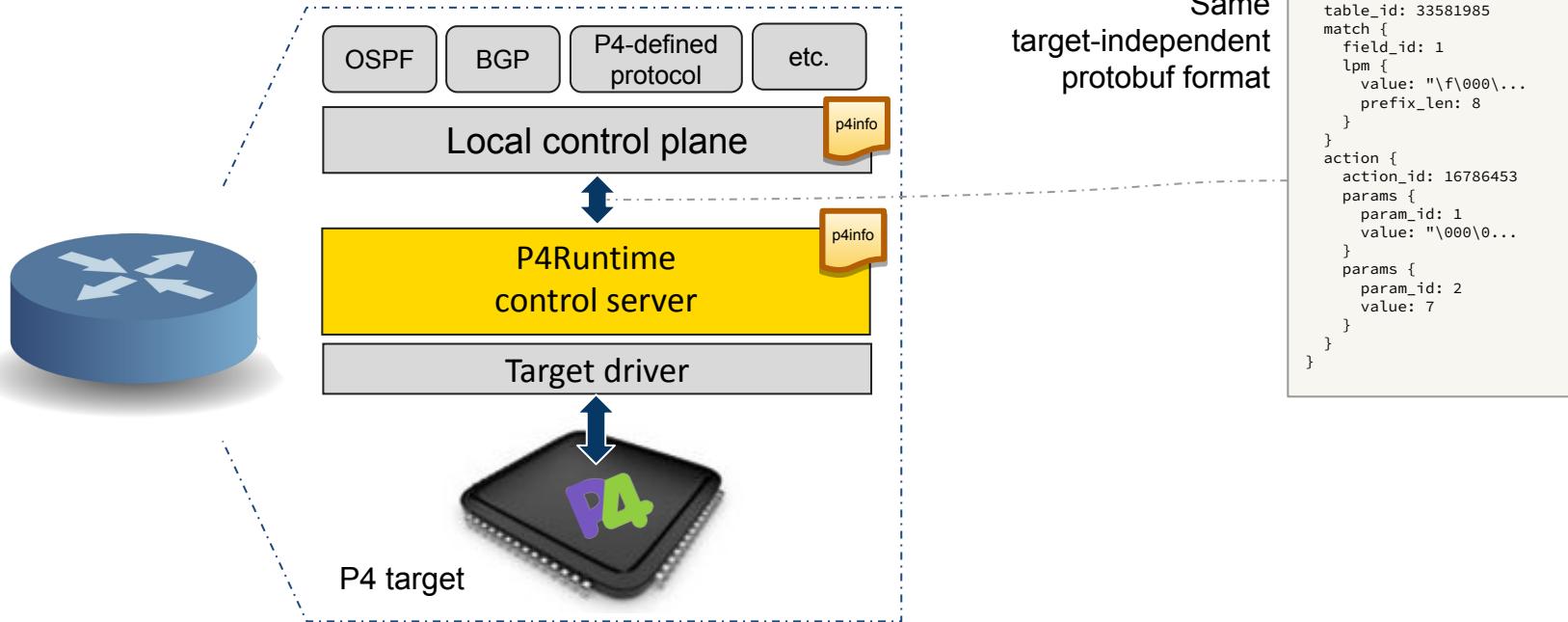
# Mastership Arbitration

- Upon connecting to the device, the client (e.g. controller) needs to open a StreamChannel
- The client must advertise its role\_id and election\_id using a MasterArbitrationUpdate message
  - If role\_id is not set, it implies the default role and will be granted full pipeline access
  - The election\_id is opaque to the server and determined by the control plane (can be omitted for single-instance control plane)
- The switch marks the client for each role with the highest election\_id as master
- Master can:
  - Perform Write requests
  - Receive PacketIn messages
  - Send PacketOut messages

# Remote control



# Local control



Same  
target-independent  
protobuf format

```
table_entry {  
    table_id: 33581985  
    match {  
        field_id: 1  
        lpm {  
            value: "\f\000\..."  
            prefix_len: 8  
        }  
    }  
    action {  
        action_id: 16786453  
        params {  
            param_id: 1  
            value: "\000\0..."  
        }  
        params {  
            param_id: 2  
            value: 7  
        }  
    }  
}
```

P4Runtime can be used equally well  
by a remote or local control plane

# P4Runtime API recap

## Things we covered:

- **P4Runtime definition**
- **P4Info**
- **Table entries**
- **Set pipeline config**
- **Packet-in/out support**
- **Controller replication**
  - Via master-slave arbitration

## What we didn't cover:

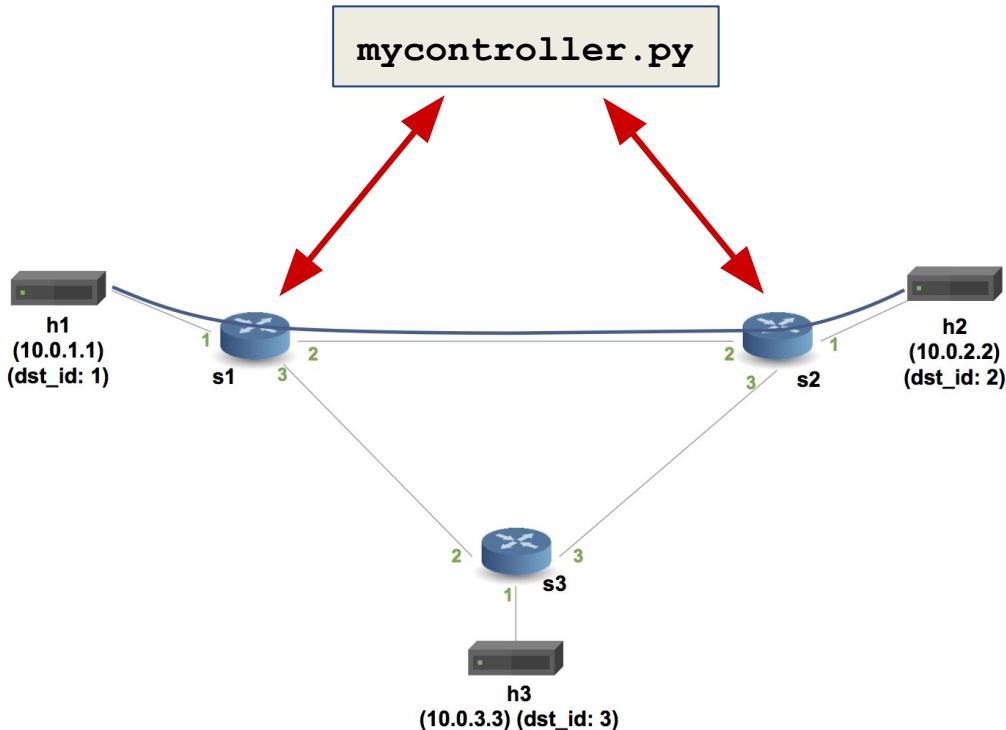
- **How to control other P4 entities**
  - Externs, counters, meters
- **Batched reads/writes**
- **Switch configuration**
  - Outside the P4Runtime scope
  - Achieved with other mechanisms
    - e.g., OpenConfig and gNMI

# **P4Runtime exercise**

# Exercise Overview

Controller's responsibilities:

1. Establish a gRPC connection to the switches for the P4Runtime service
2. Push the P4 program to each switch
3. Write the tunnel forwarding rules:
  - a. **myTunnel\_ingress** rule to encapsulate packets on the ingress switch
  - b. **myTunnel\_forward** rule to forward packets on the ingress switch
  - c. **myTunnel\_egress** rule to decapsulate and forward packets on the egress switch
4. Read the tunnel ingress and egress counters every 2 seconds



# Getting started

The source code has already been downloaded on your VM:

`~/tutorials/exercises/p4runtime`

You should start by reading the `README.md`

In this exercise, you will need to complete the implementation of `writeTunnelRules` in `mycontroller.py`

You will need two Terminal windows: one for your dataplane network (Mininet) that you will start using `make`, and the other is for your controller program.

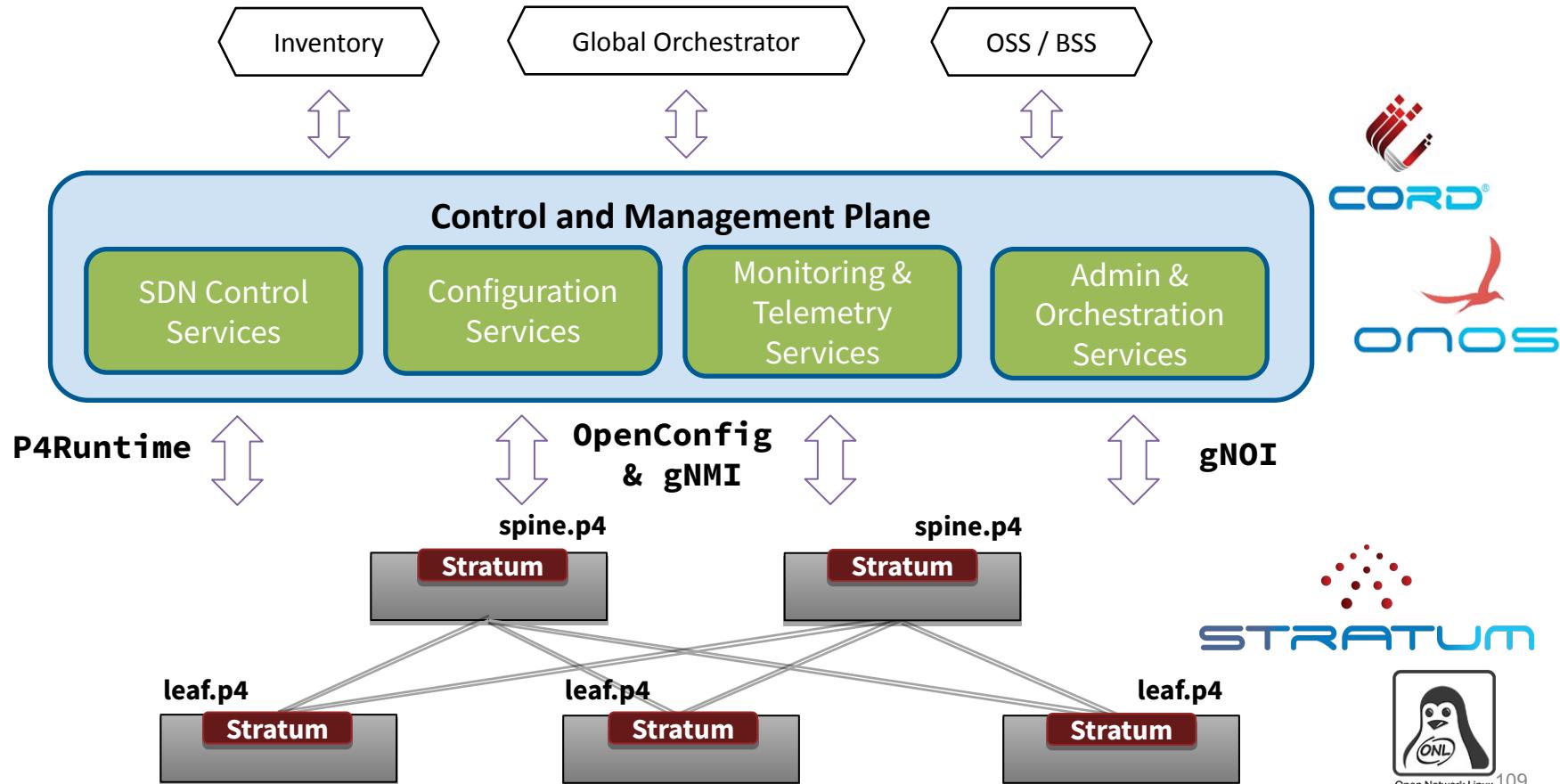
To find the source code:

<https://github.com/p4lang/tutorials/>

The screenshot shows a portion of a `README.md` file. At the top, there's a header for "Implementing a Control Plane using P4 Runtime". Below it is a section titled "Introduction". A note states: "In this exercise, we will be using P4 Runtime to send flow entries to the switch instead of using the switch's CLI. We will be building on the same P4 program that you used in the `basic_tunnel` exercise. The P4 program has been renamed to `advanced_tunnel.py` and has been augmented with two counters (`ingressTunnelCounter`, `egressTunnelCounter`) and two new actions (`myTunnel_ingress`, `myTunnel_egress`). You will use the starter program, `mycontroller.py`, and a few helper libraries in the `p4runtime_lib` directory to create the table entries necessary to tunnel traffic between host 1 and 2." A "Spoiler alert" note mentions a reference solution in the `solution` sub-directory. Below this, a "Step 1: Run the (incomplete) starter code" section describes the starter code file `mycontroller.py` and its purpose. It also instructs to compile the P4 program and start the network using `mycontroller.py`. A numbered list item 1. In your shell, run: `make` is shown at the bottom.

# **P4Runtime in the wild**

# P4Runtime-enabled Open Source SDN Stack



# Some Pointers

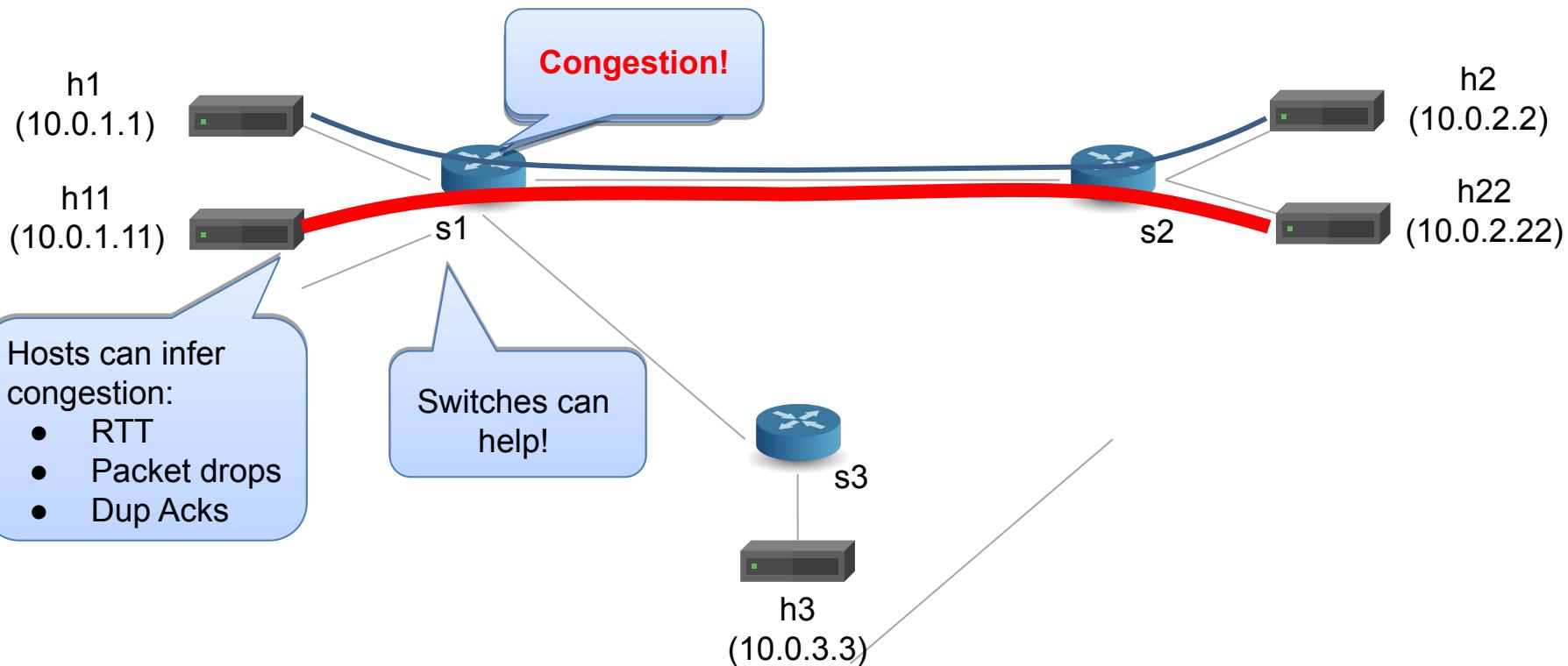
<https://stratumproject.org/>

<https://onosproject.org/>

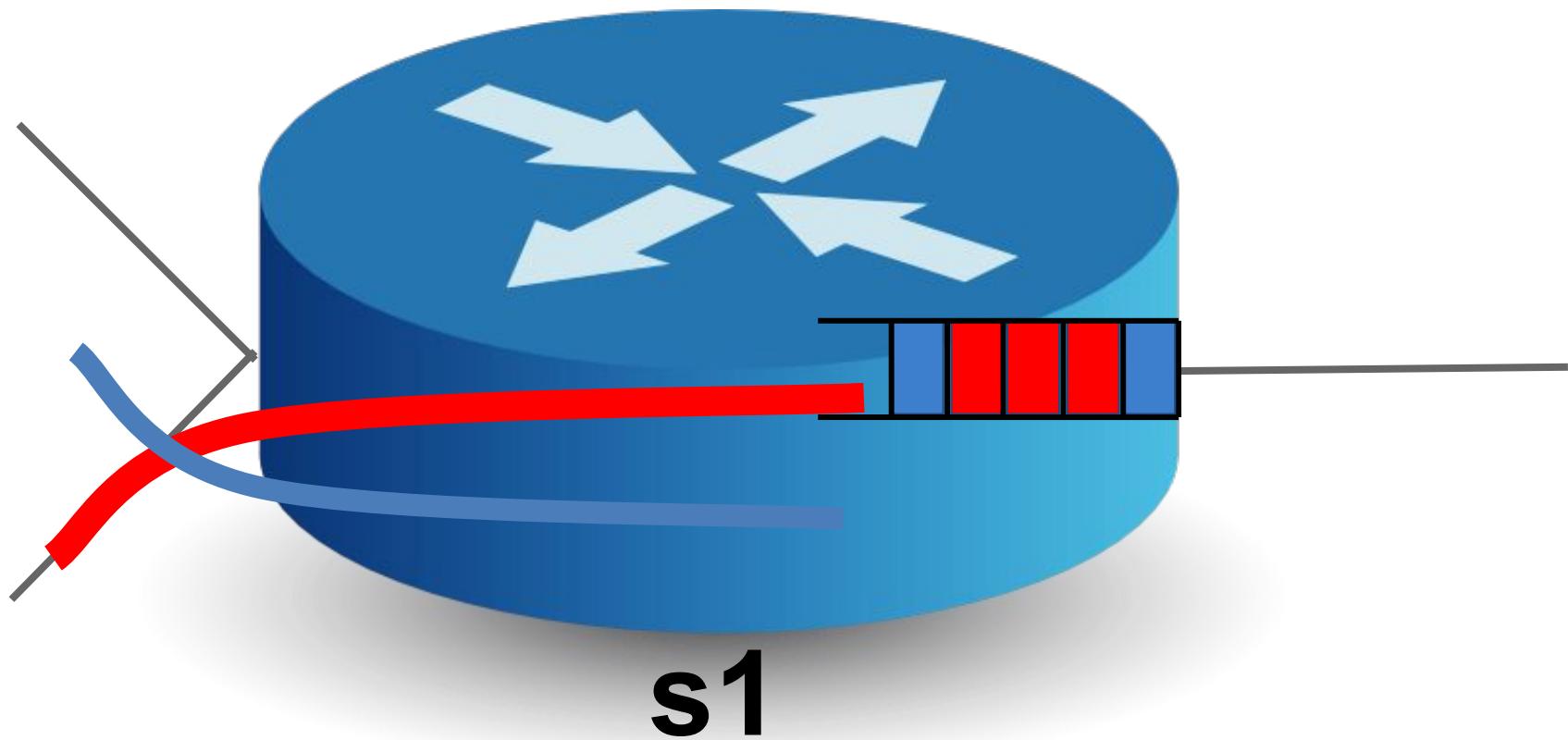
<https://wiki.onosproject.org/display/ONOS/P4+brigade>

# **Lab 3: Monitoring & Debugging**

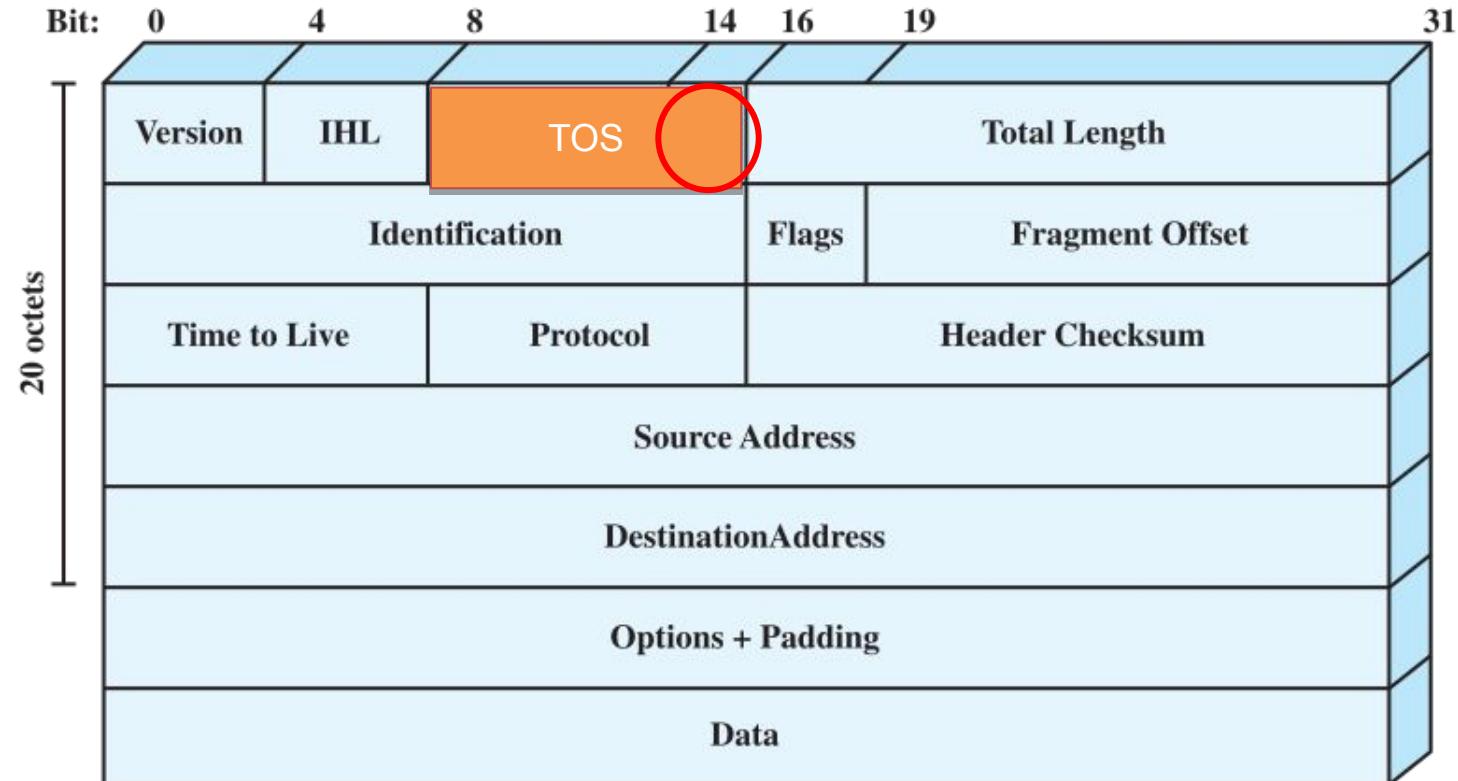
# Monitoring & Debugging



# 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

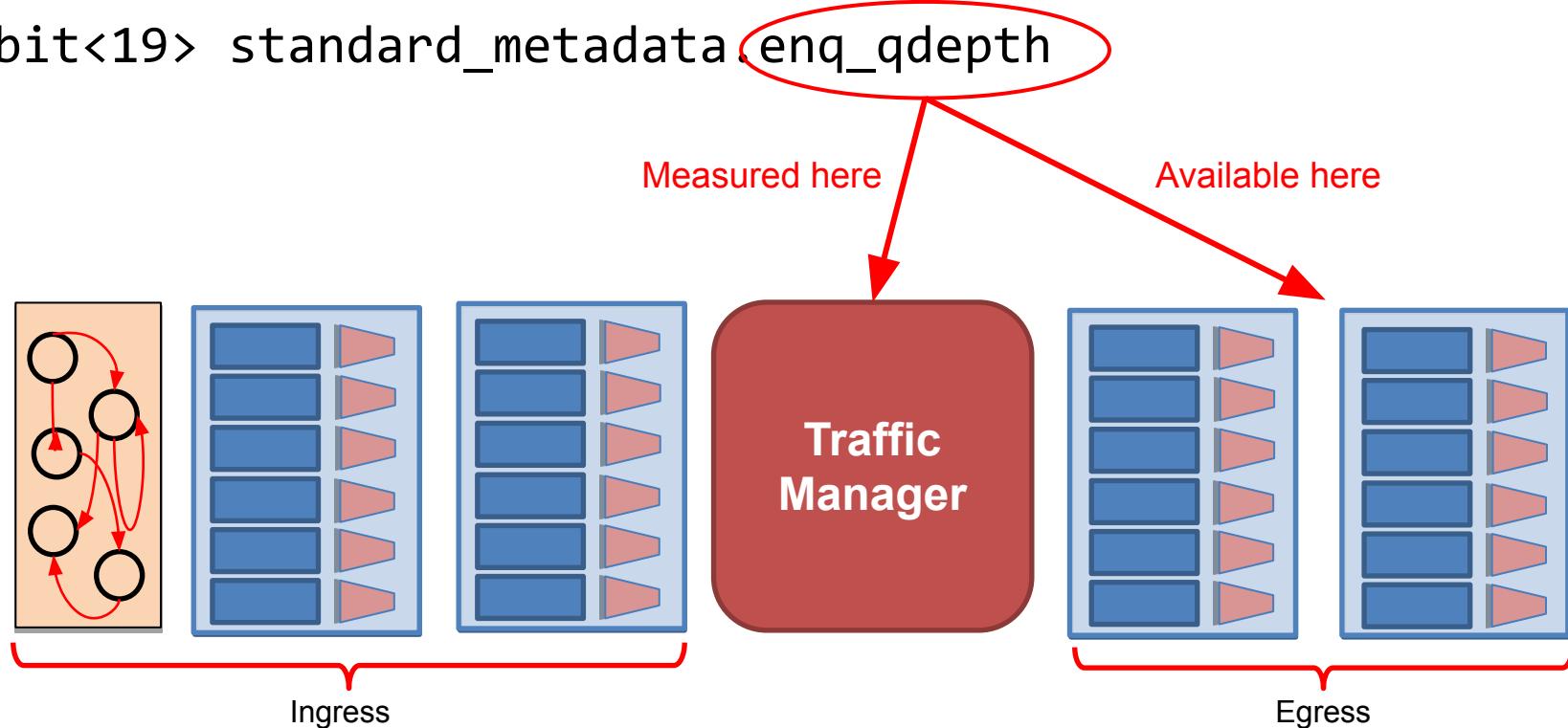
- **For packets originating from ECT, ECN-capable switches set the CE bit upon congestion**

- E.g., observed queue depth > threshold

# Explicit Congestion Notification in P4

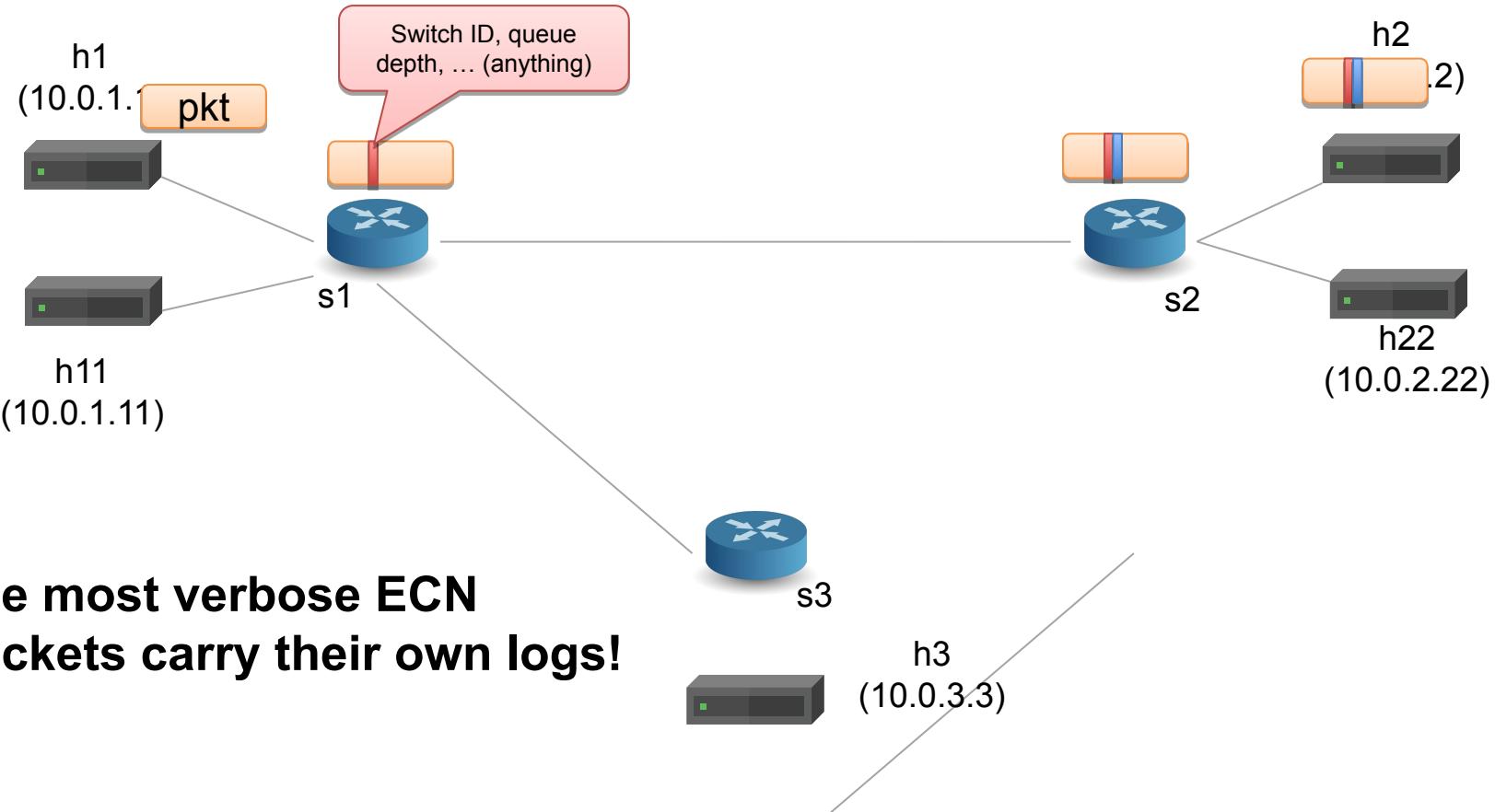
- The standard data for the V1Model includes the queue depth:

bit<19> standard\_metadata.enq\_qdepth



# Coding Break

# Multi-Route Inspection



# 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;  
}
```

- **Header validity operations:**

- `hdr.setValid() : add_header`
- `hdr.setInvalid() : remove_header`
- `hdr.isValid() : test validity`

- **Header Stacks**

- `hdr[CNT] stk;`

- **Header Stacks in Parsers**

- `stk.next`
- `stk.last`
- `stk.lastIndex`

- **Header Stacks in 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;
}
```

# Coding Break

# Questions and Answers



<https://bit.ly/join-p4-lang-slack>

Join #d2-2018-spring channel



<https://pigeonhole.at>

Event code: P4D2



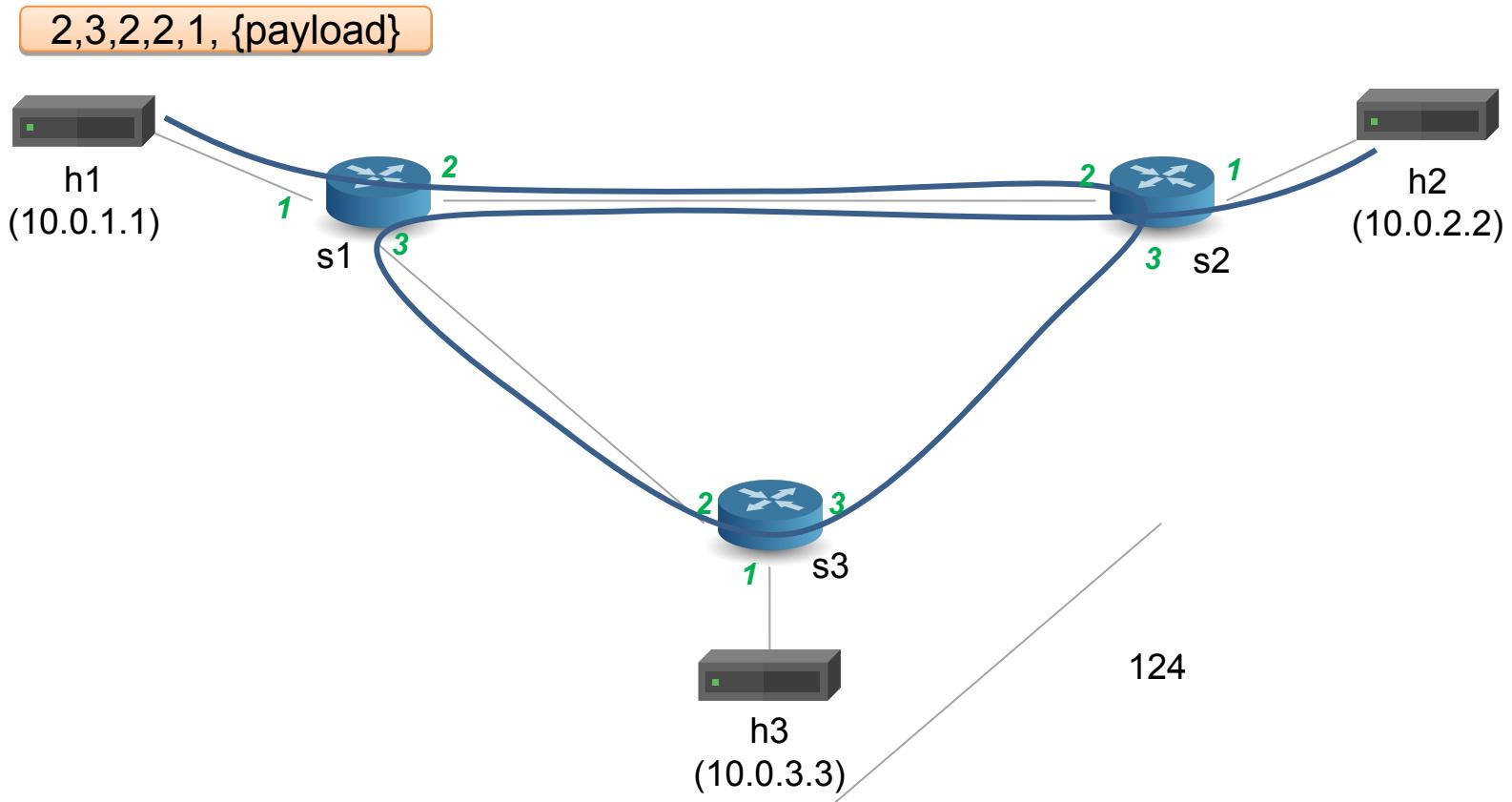
*Questions for TAs*



*Questions for Panel*

# **Lab 4: Advanced Behavior**

# Source Routing



# Source Routing: Packet Format

```
#define MAX_HOPS 9

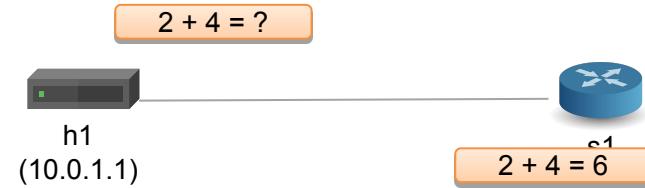
const bit<16> TYPE_IPV4 = 0x800;
const bit<16> TYPE_SRCROUTING = 0x1234;
header srcRoute_t {
    bit<1>    bos;
    bit<15>   port;
}

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

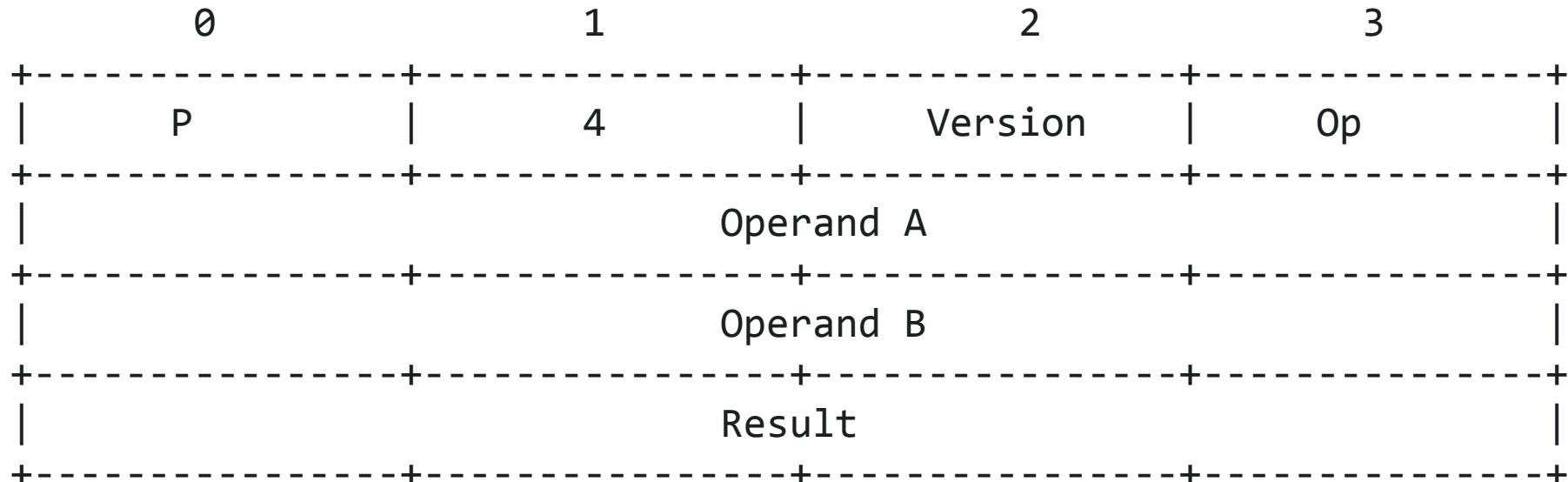
- Parse source routes only if etherType is **0x1234**
- The special value **bos == 1** indicates the “bottom of stack”
- Forward packets using source routes, and also decrement IPv4 TTL
- Drop the packet if source routes are not valid
- Hint: Use the next, pop\_front primitives  
`packet.extract(hdr.srcRoutes.next)`  
`hdr.srcRoutes.pop_front(1)`

# Coding Break

# 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();  
    }  
}
```

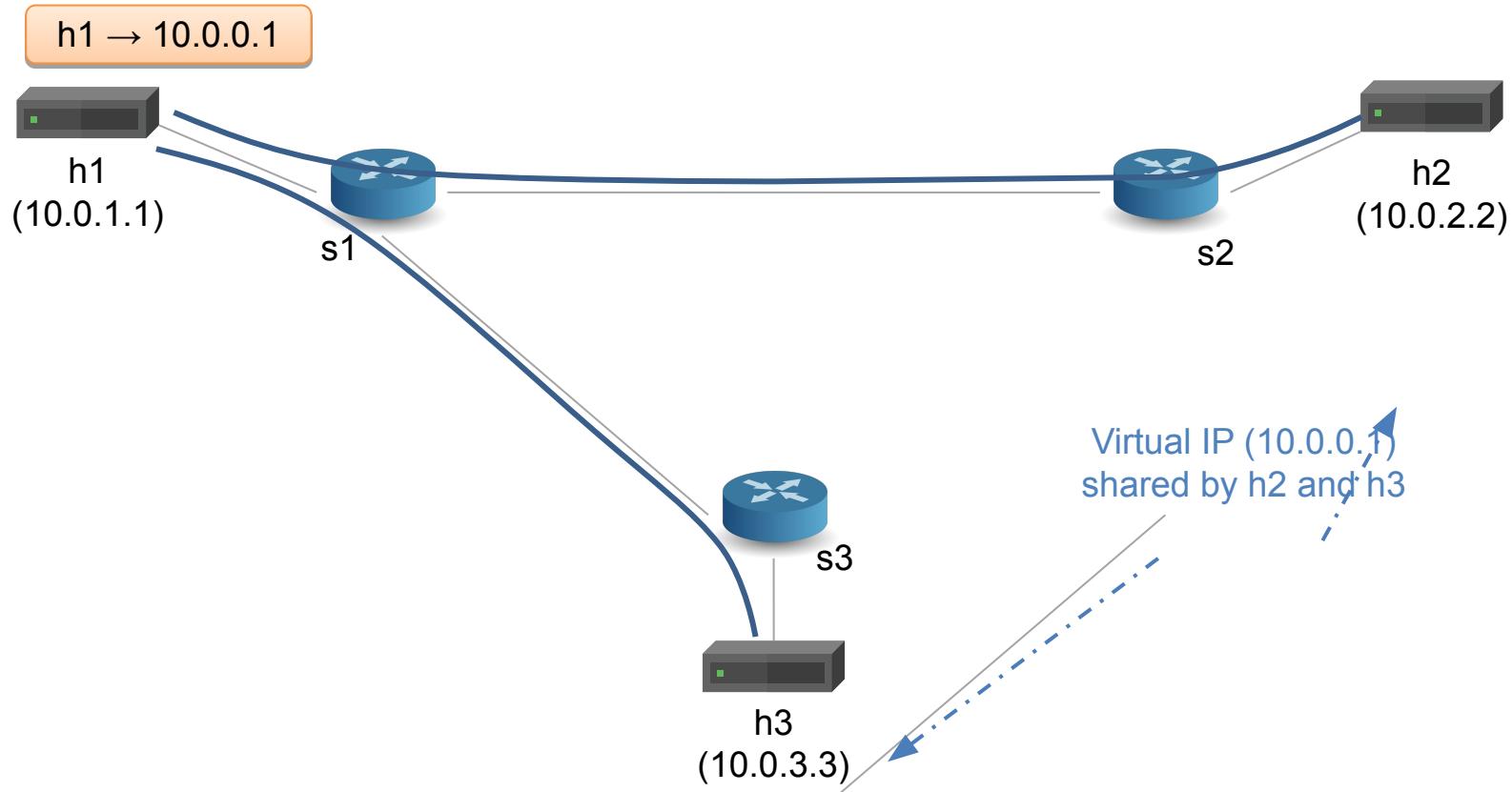
**Can initialize tables with constant entries**

**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**

# Coding Break

# 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.**

# **Wrapping up & Next Steps**

# Why P4<sub>16</sub>?

- **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

The screenshot shows the homepage of the P4 Language Consortium at https://p4.org. The page features a large banner image of a person working on a computer. Overlaid on the banner are four sections: "Protocol Independent" (describing how P4 programs specify packet processing), "Target Independent" (describing its use from ASICs to software switches), "Field Reconfigurable" (allowing network engineers to change switch behavior after deployment), and a code snippet for a "routing" table. A "TRY IT! GET THE CODE ON GITHUB" button is visible at the bottom right.

P4 programs specify how a switch processes packets.

```
table routing {  
    key = { ipv4.dstAddr : lpm; }  
    actions = { drop; route; }  
    size : 2048;  
}  
control.ingress() {  
    apply {  
        routing.apply();  
    }  
}
```

TRY IT! GET THE CODE ON GITHUB