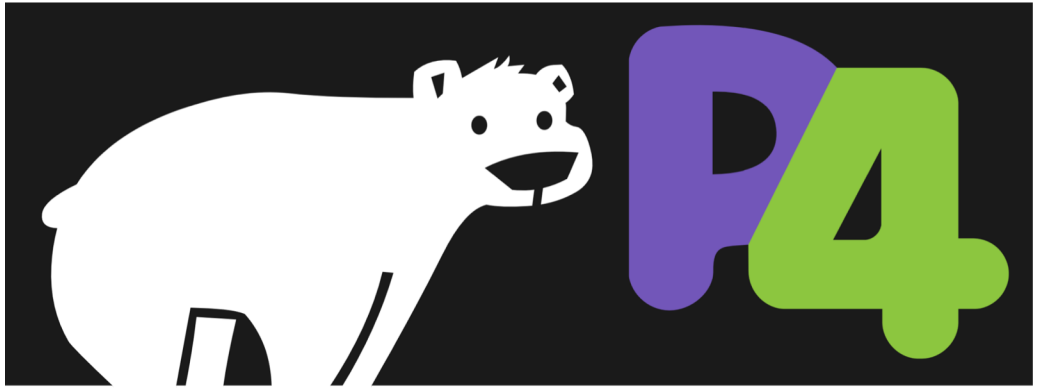


P4 Language Tutorial

Setting Up

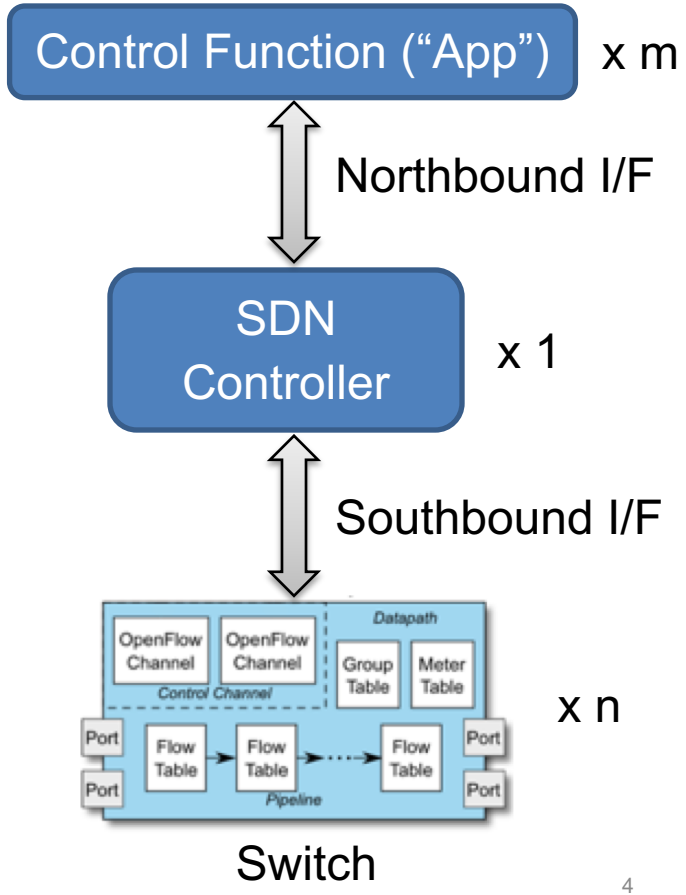
- **We will use virtual machine for hands-on lab exercises**
- **If you've already built or downloaded the VM – great!**
- **Otherwise, look for instructors with USB sticks**
- **Login Credentials:**
  - User: p4
  - Password: p4

What is Data Plane Programming?

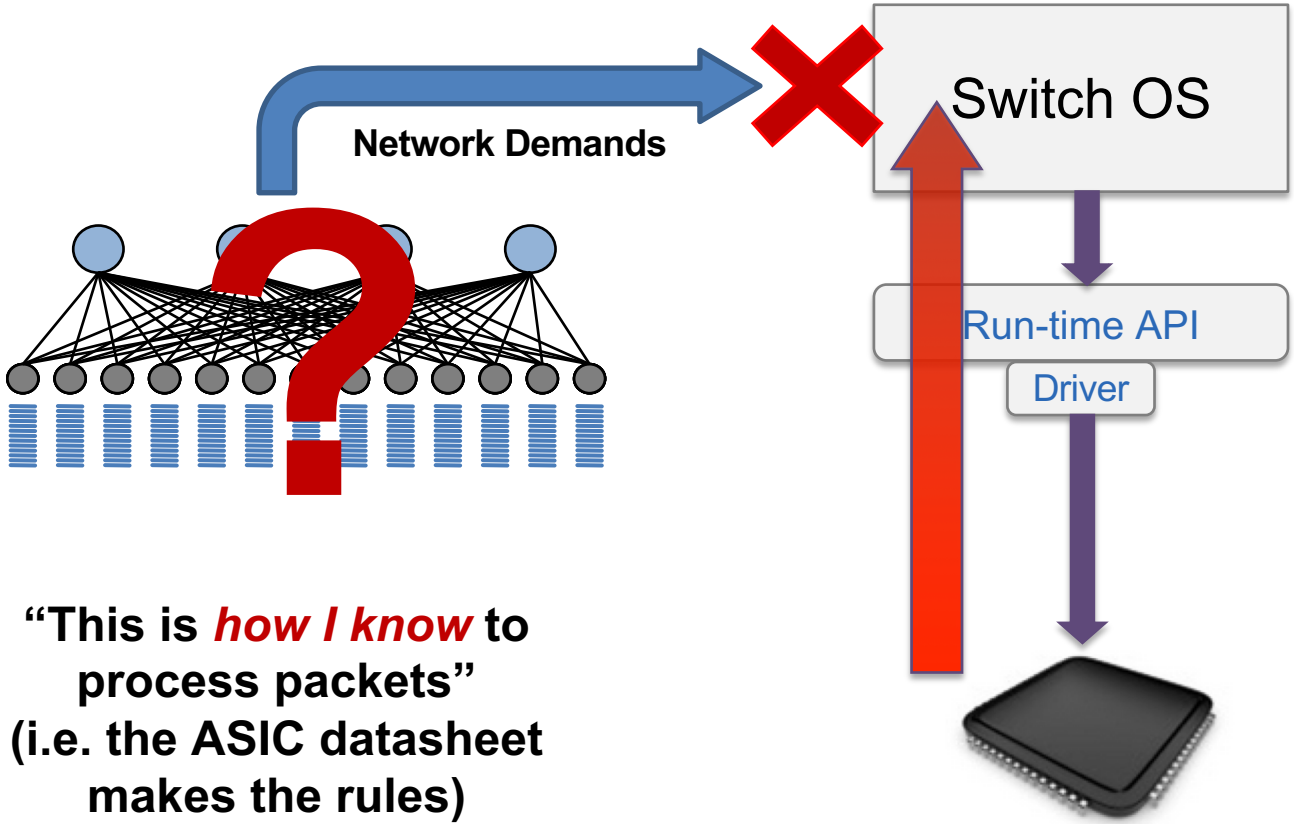
- **Why program the Data Plane?**

Software Defined Networking (SDN)

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

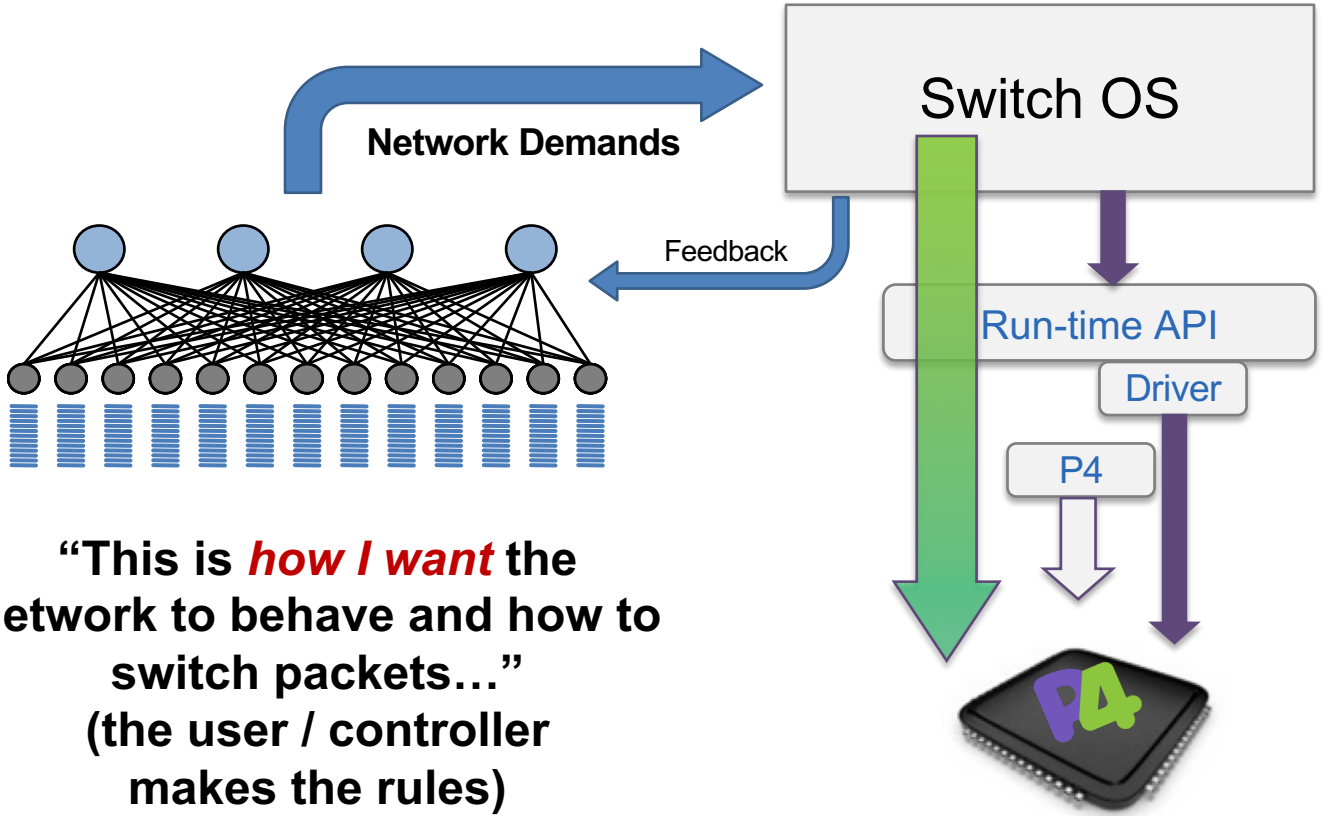


Status Quo: Bottom-up design



"This is *how I know* to process packets" (i.e. the ASIC datasheet makes the rules)

A Better Approach: Top-down design



"This is *how I want* the network to behave and how to switch packets…" (the user / controller makes the rules)

P4 Programmable Device

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…

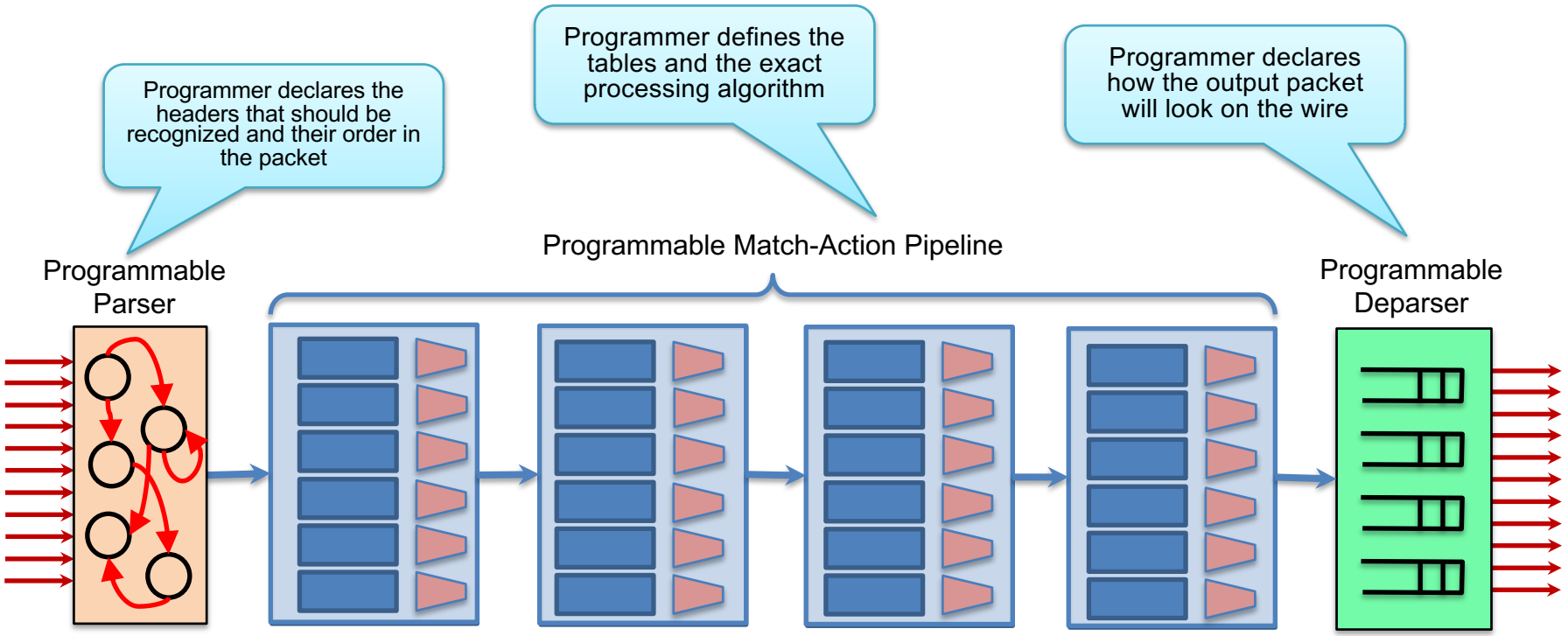
Programmable Network Devices

- **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.

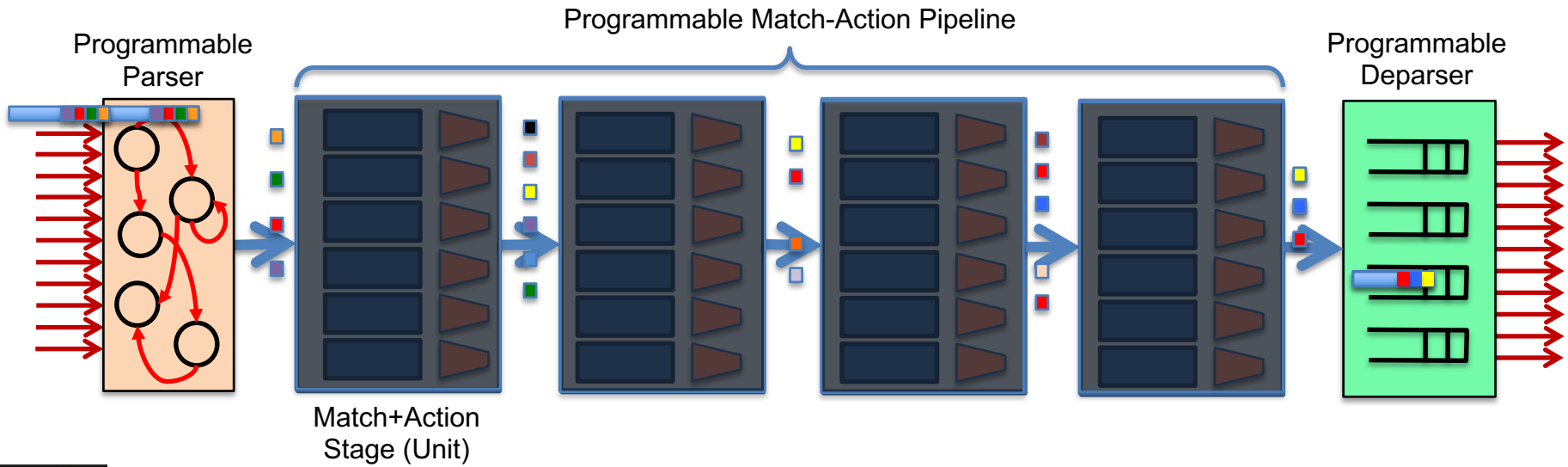
P4 Basics

PISA: Protocol-Independent Switch Architecture

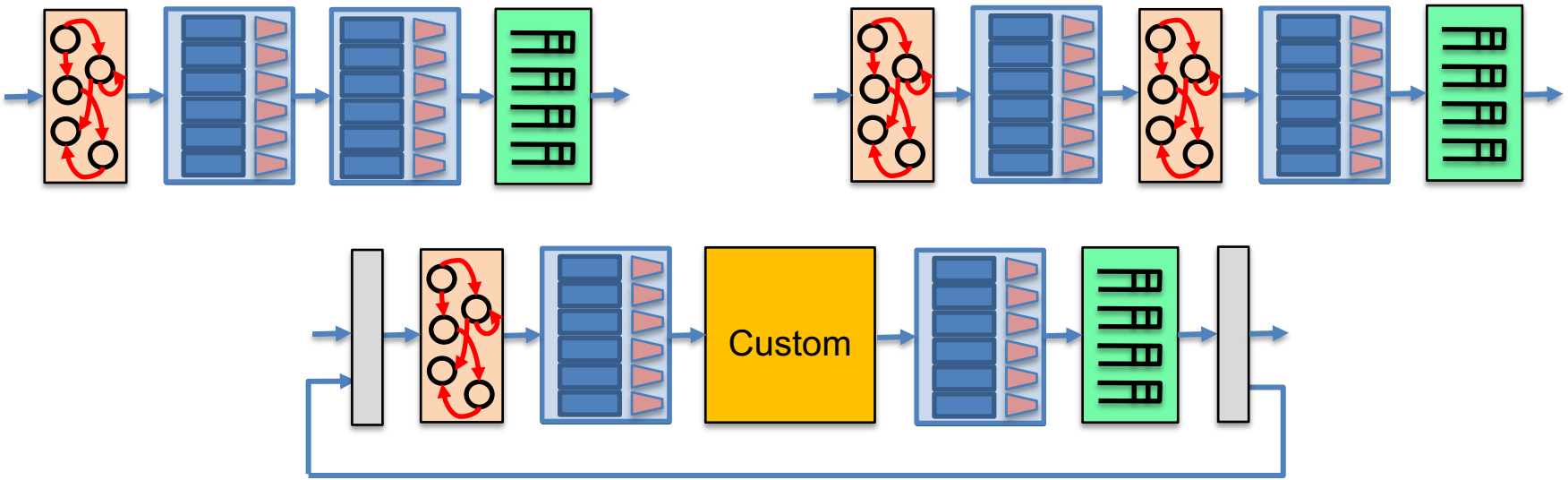


PISA in Action

- Packet is parsed into individual headers
- Headers and metadata are used for match/action processing
- Headers can be modified, added or removed
- Packet is deparsed (serialized)

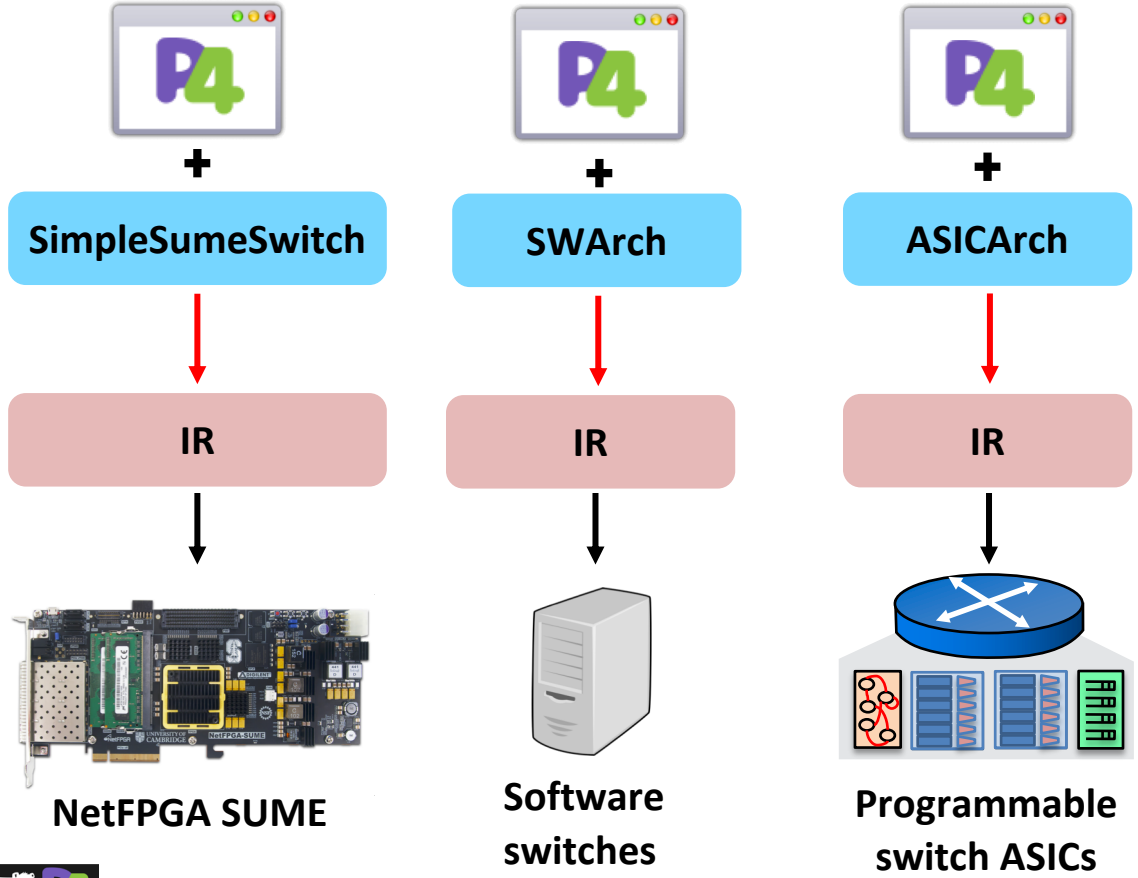


$P4_{16}$ Architectures



- Define how the P4 programmable elements are composed together
- Define how to interact with non-programmable elements
- P4 programs are written for a specific architecture
- Can think of architecture definition as an abstract base class in OOP

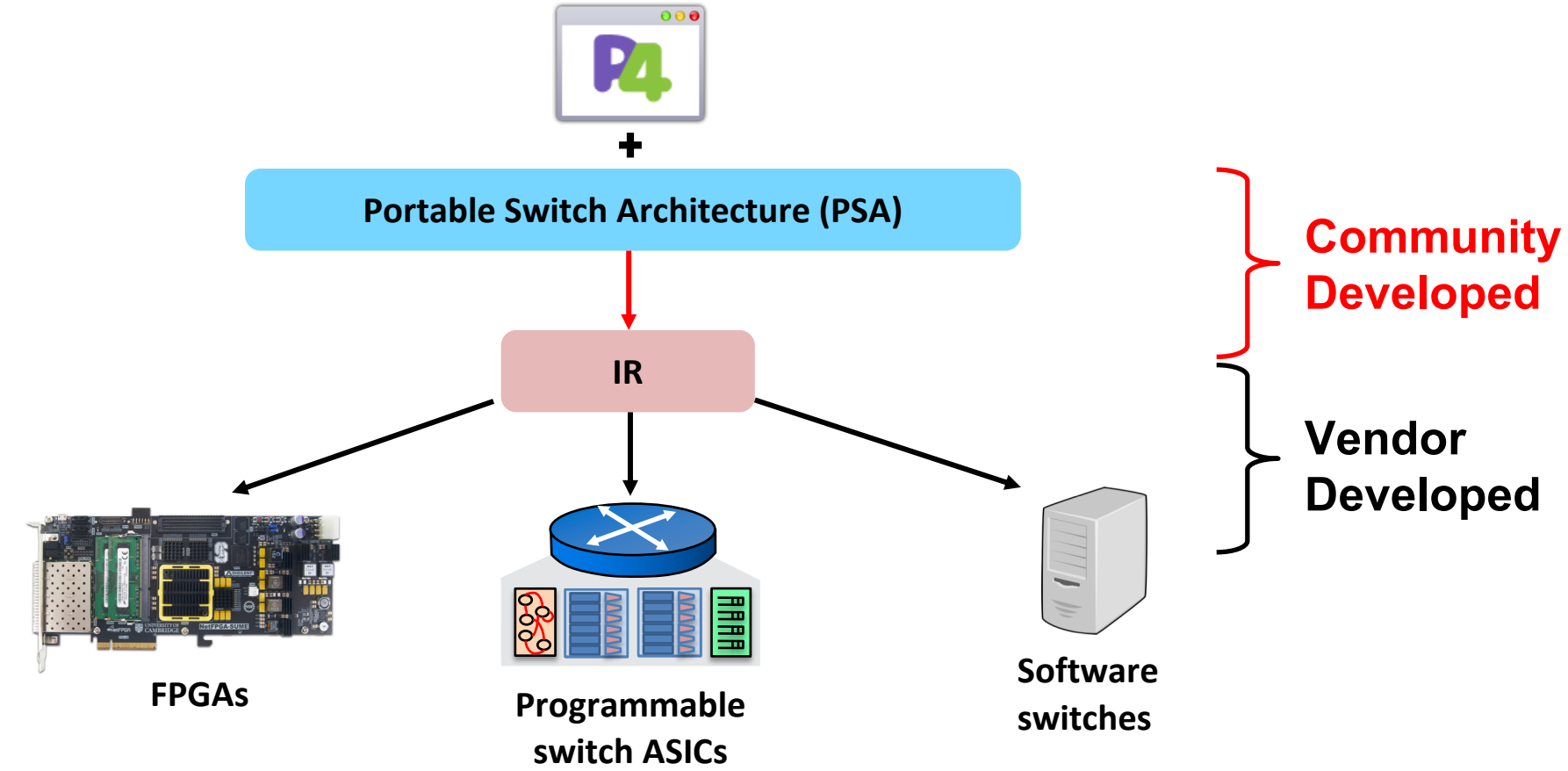
$P4_{16}$ Approach



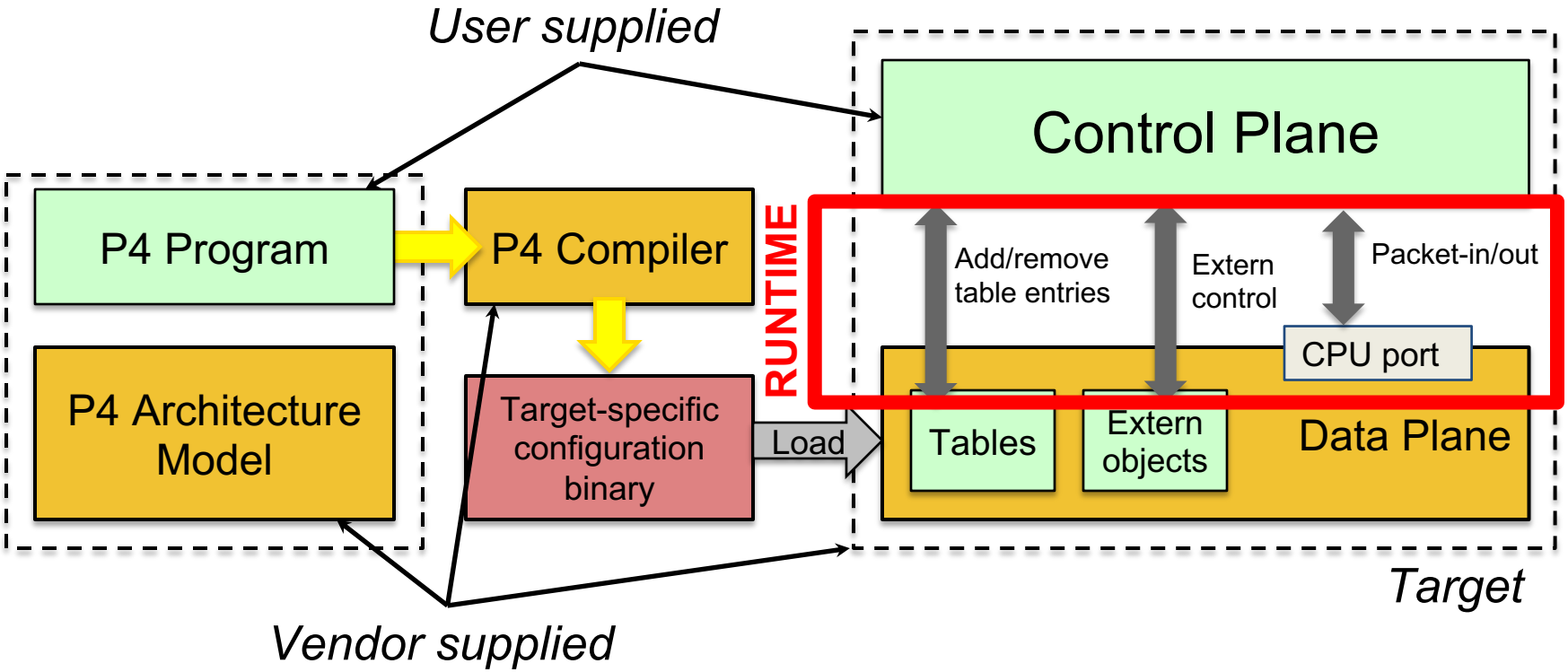
Why the Language/Architecture/Target Separation?

- **Core $P4_{16}$ language should be target independent**
- **Different targets may want to expose programmability in different ways**
- **Why is it important to keep this separation in mind?**
  - Affects portability of programs

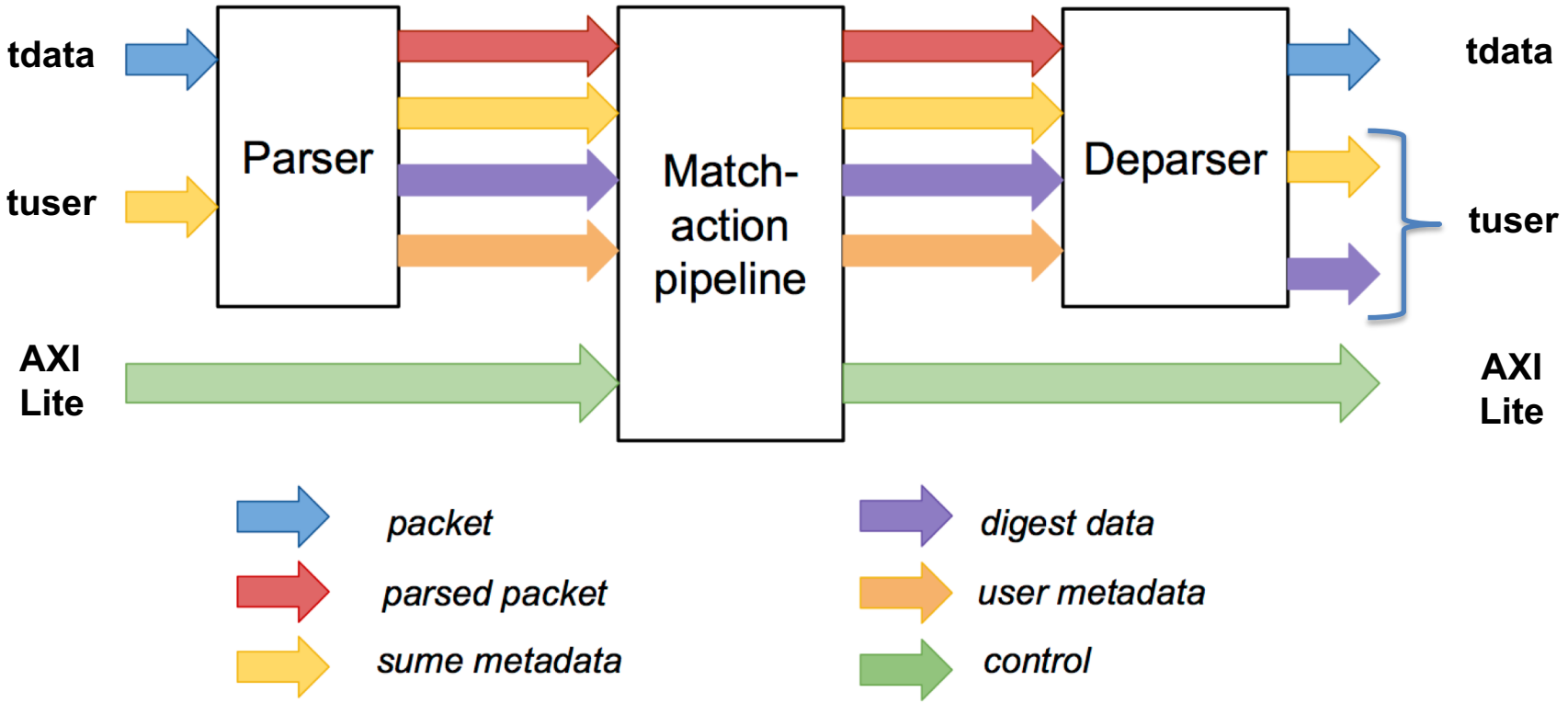
Portable Switch Architecture (PSA)



Programming a P4 Target



SimpleSumeSwitch Architecture Model for SUME Target



P4 used to describe parser, match-action pipeline, and deparser

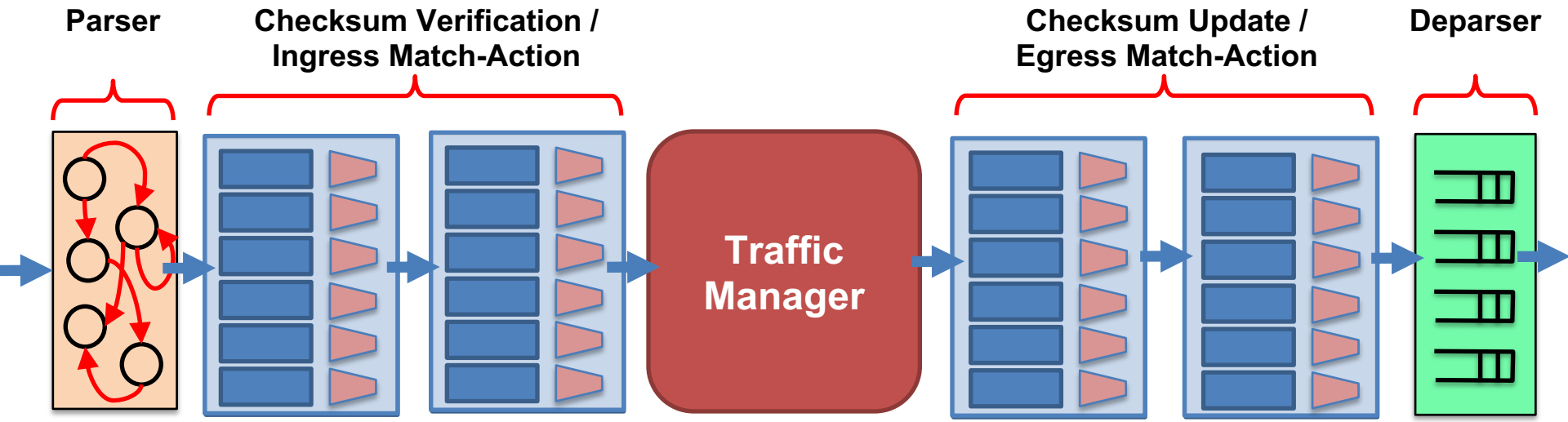
Standard Metadata in SimpleSumeSwitch Architecture

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

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

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)

$P4_{16}$ V1Model Architecture File (v1model.p4)

```
#include <core.p4>

/* SUPPORTED EXTERNS */
...

/* STANDARD METADATA */
struct standard_metadata_t { ... }

/* PARSER INTERFACE */
parser Parser<H, M>(packet_in packet,
                    out H hdr,
                    inout M meta,
                    inout standard_metadata_t smeta);

/* CHECKSUM VERIFICATION INTERFACE */
control VerifyChecksum<H, M>(in H hdr,
                             inout M meta);

/* INGRESS PROCESSING INTERFACE */
control Ingress<H, M>(inout H hdr,
                      inout M meta,
                      inout standard_metadata_t std_meta);
```

```
/* EGRESS PROCESSING INTERFACE */
control Egress<H, M>(inout H hdr,
                     inout M meta,
                     inout standard_metadata_t std_meta);

/* CHECKSUM UPDATE INTERFACE */
control ComputeChecksum<H, M>(inout H hdr,
                              inout M meta);

/* DEPARSER INTERFACE */
control Deparser<H>(packet_out packet,
                    inout H hdr);

/* SWITCH */
Package V1Switch<H, M> (
  Parser<H, M> p,
  VerifyChecksum<H, M> vr,
  Ingress<H, M> ig,
  Egress<H, M> eg,
  ComputeChecksum<H, M> ck,
  Deparser<H> dep
);
```

$P4_{16}$ 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(packet_out packet,
                   inout headers hdr) {
  ...
}
/* SWITCH */
V1Switch(
  MyParser(),
  MyVerifyChecksum(),
  MyIngress(),
  MyEgress(),
  MyComputeChecksum(),
  MyDeparser()
) main;
```

Exercise 1 – Basic IPv4 Forwarding

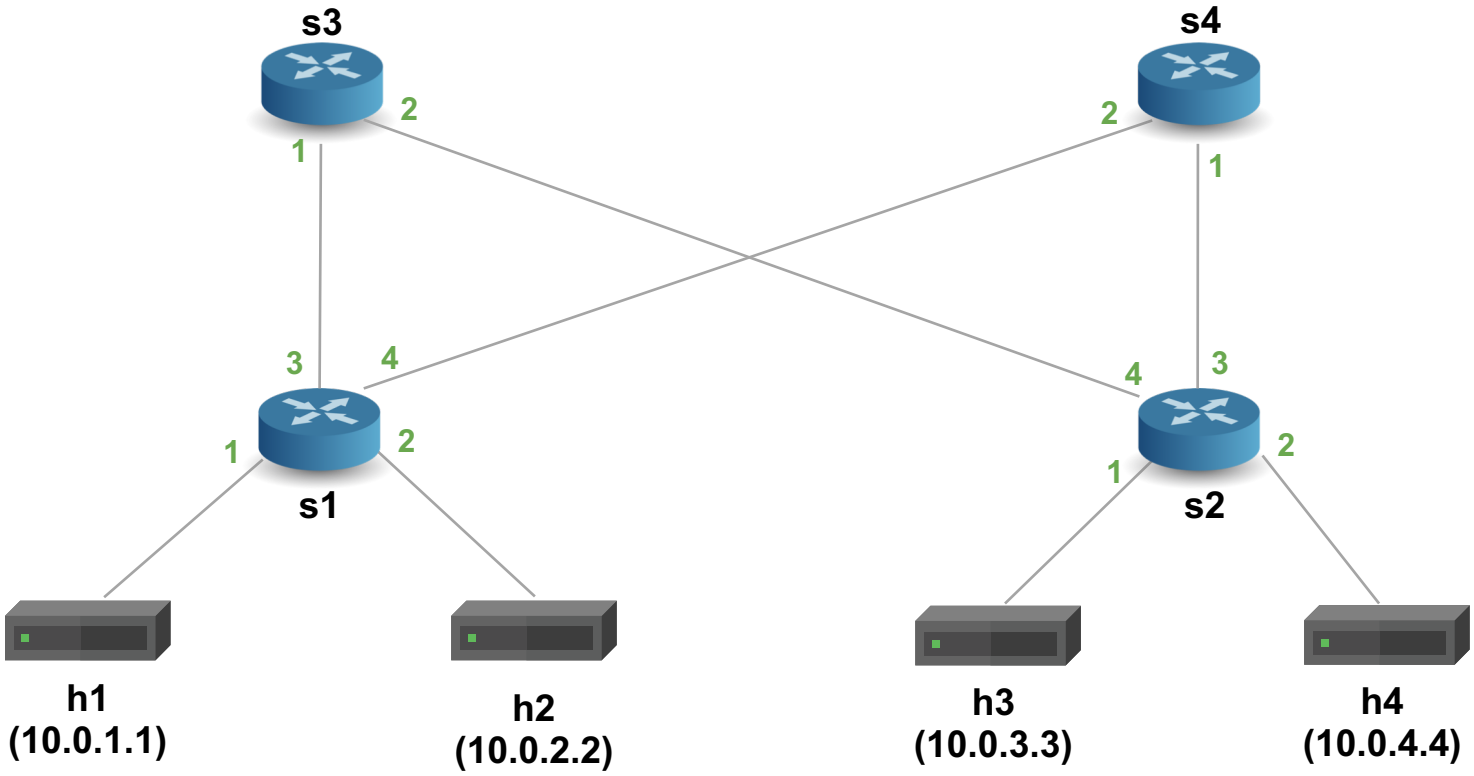
Basic IPv4 Forwarding

- **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**

Tutorial Topology



P4$_{16}$ Types (Basic and Header Types)

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

Basic Types

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

Header Types: Ordered collection of members

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

Typedef: Alternative name for a type

P4$_{16}$ Parsers

- Parsers map packets into headers and metadata, written as a state machine
- 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)

```
/* 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();
}
/* From v1model.p4 */
parser Parser(packet_in packet,
              out headers hdr,
              inout metadata meta,
              inout standard_metadata_t std_meta);

/* From simple_sume_switch.p4 */
parser Parser(packet_in packet,
              out headers hdr,
              out user_data_t user,
              out digest_data_t digest,
              inout sume_metadata_t smeta);
```

Parsers (V1Model)

```
/* 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 select(hdr.ethernet.type) {
      0x800   : parse_ipv4;
      default : accept;
    }
  }

  state parse_ipv4 {
    packet.extract(hdr.ipv4);
    transition accept;
  }
}
```

P4$_{16}$ has a `select` statement that can be used to branch in a parser

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

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

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

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

Coding Break

P4$_{16}$ 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)**

```
/* v1model.p4 */
control Ingress(inout headers hdr,
                inout metadata meta,
                inout standard_metadata_t std_meta);
```

```
/* simple_sume_switch.p4 */
control Ingress(inout headers hdr,
                inout user_data_t user,
                inout digest_data_t digest,
                inout sume_metadata_t smeta);
```

P4$_{16}$ Tables

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

- **The fundamental unit of a Match-Action Pipeline**
  - Specifies: data to match on & match kind
  - Specifies a list of *possible* actions
  - Optionally specifies:
    - Size
    - Default action
    - Static entries
    - etc.
- **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)

Match Kinds

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

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

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

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

P4 Match-Action Processing

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

  action set_egress_spec(bit<8> port) {
    std_meta.egress_spec = port;
  }

  table forward {
    key = { std_meta.ingress_port : exact; }
    actions = { set_egress_spec; }
    size = 1024;
    default_action = set_egress_spec(0);
  }

  apply {
    if (hdr.ipv4.isValid()) {
      forward.apply();
    }
    else {
      /* defined in v1model.p4 */
      mark_to_drop(std_meta);
    }
  }
}
```

- Declare tables and actions
- Similar to C functions
- No loops
- Functionality specified by code in `apply` statement
- Table entries populated by control-plane

`forward` table entries:

Key	Action Name	Action Data
1	set_egress_spec	2
2	set_egress_spec	1

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);
    packet.emit(hdr.ipv4);
    ...
  }
}
```

- **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

Coding Break

Externs

- Most targets contain specialized components which cannot be expressed in P4 (e.g. complex computations).
- The core $P4_{16}$ language should be target-independent.
- In $P4_{14}$ almost 1/3 of the constructs were target-dependent.
- Often one of the major differences between architectures.

Externs

```
/* Defined in v1model.p4 */
extern void random<T>(out T result, in T lo, in T hi);
extern void mark_to_drop(inout std_meta_t std_meta);

/* User program */
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout std_meta_t std_meta) {


  apply {
    bit<8> rand_val;

    /* extern function invocation */
    random<bit<8>>(rand_val, 0, 255);

    if (rand_val > THRESH) {
        /* extern function invocation */
        mark_to_drop(std_meta);
    }
  }
}
```

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

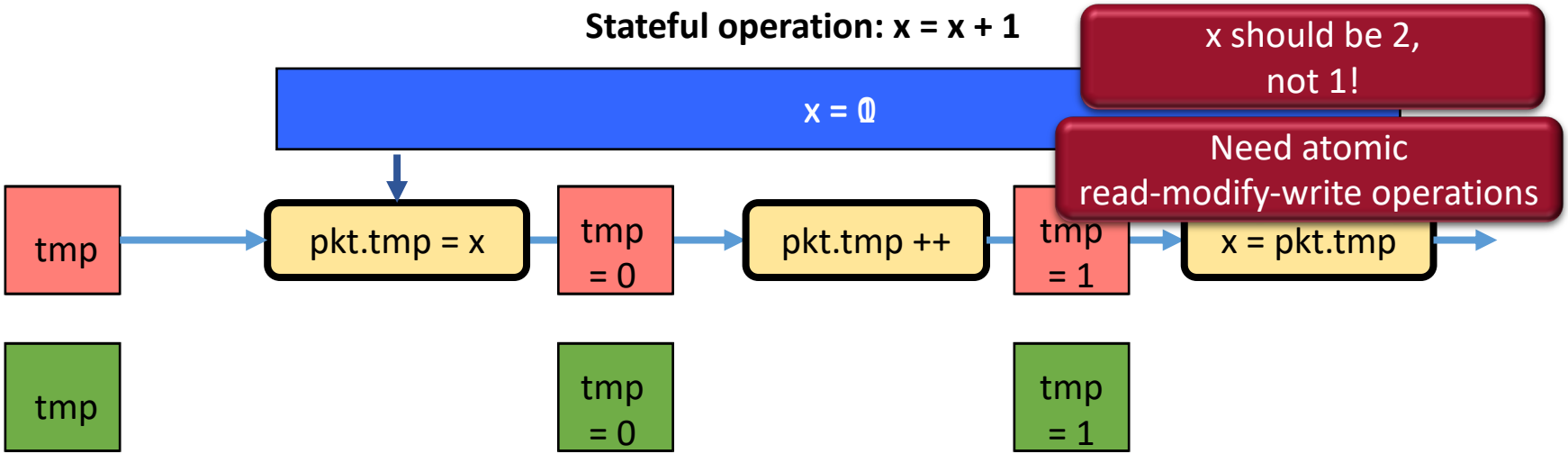
Stateful Packet Processing

Line-Rate Constraints

- **Low line-rate**
  - E.g. software switch
  - Relaxed constraints
  - Can expose very flexible stateful externs
- **High line-rate**
  - Multi-Tbps → process one packet every ~700ps → must use SRAM
  - Limited amount of memory (tens of MB)
  - Limited number of memory operations (multi-ported SRAM is too expensive)
  - Often expose stateful externs in a constrained way …

Stateful Externs

- **Stateless – reinitialized for each packet**
- **Stateful – keep state between packets**
  - Often cannot pipeline stateful operations



Slide Credit: Anirudh Sivaraman

Stateful Externs: V1Model & SimpleSumeSwitch

V1Model

```
/* General purpose state */
extern register<T> {
    register(bit<32> size);
    void read(out T result, in bit<32> index);
    void write(in bit<32> index, in T value);
}

/* Count bytes and/or packets */
extern counter {
    counter(bit<32> size, CounterType type);
    void count(in bit<32> index);
}


/* Metering flows of packets based on RFC 2697 and RFC
2698 */
extern meter {
    meter(bit<32> size, MeterType type);
    void execute_meter<T>(in bit<32> index,
                          out T result);
}
```

SimpleSumeSwitch

```
/* Atomically read or write a state variable */
extern void <name>_reg_rw<T, D>(in T index,
                                in D newVal,
                                in bit<8> opCode,
                                out D result);

/*
va
ex

/*
variable based on a predicate */
extern void <name>_reg_praw<T, D>(in T index,
                                  in D data,
                                  in bit<8> opCode,
                        in D compVal, in bit<8> relCode,
              out D result, out bit<1> predicate_result);
```

Various
Atomic Read/Modify/Write
Operations

Stateful Externs: V1Model & SimpleSumeSwitch

- **A simple example: a resetting counter**

```
count[256];
if (pkt.hdr.reset == 1):
    count[pkt.hdr.index] = 0
else:
    count[pkt.hdr.index]++
```

V1Model

```
register<bit<32>>(256) count_reg;

count_reg.read(count, pkt.hdr.index);
if (pkt.hdr.reset == 1) {
  count = 0;
} else {
  count = count + 1;
}
count_reg.write(pkt.hdr.index, count);
```

SimpleSumeSwitch

```
extern void count_reg_raw(in bit<8> index,
                          in bit<32> data,
                          in bit<8> opCode,
                          out bit<32> result);

bit<8> index = pkt.hdr.index;
bit<32> data; bit<8> opCode;
if(pkt.hdr.reset == 1) {
  data = 0;
  opCode = REG_WRITE;
} else {
  data = 1;
  opCode = REG_ADD;
}

bit<32> result;
/* Perform the atomic read/modify/write operation */
count_reg_raw(index, data, opCode, result);
```

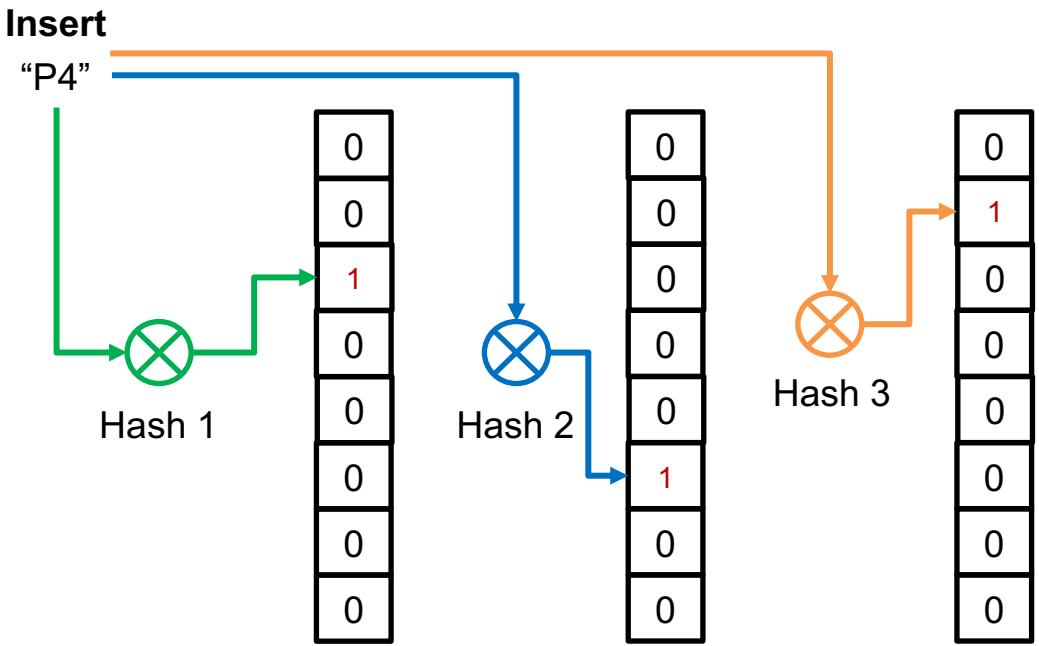
Bloom Filters & Sketches

- **Compact, fixed-size data structures that can summarize traffic statistics while bounding an error margin**

- **Can help us answer questions such as:**
  - *“Have I seen this flow before?”*
  - *“How many unique flows are there?”*
  - *“Which are the top-N flows or elephant flows?”*
  - *“Which are top-N spreaders?”*
  - *“How varied are certain header fields?”*
  - *“What does the flow-size distribution look like?”*

Bloom Filter

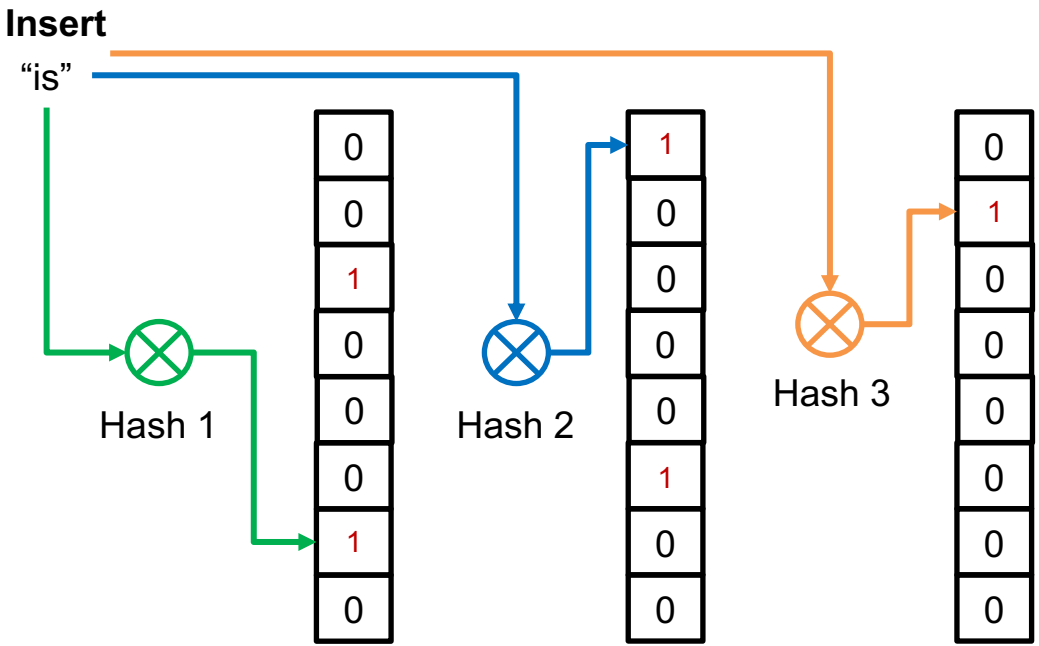
- **A memory efficient approach for insertions and membership queries**



Slide credit: Laurent Vanbever

Bloom Filter

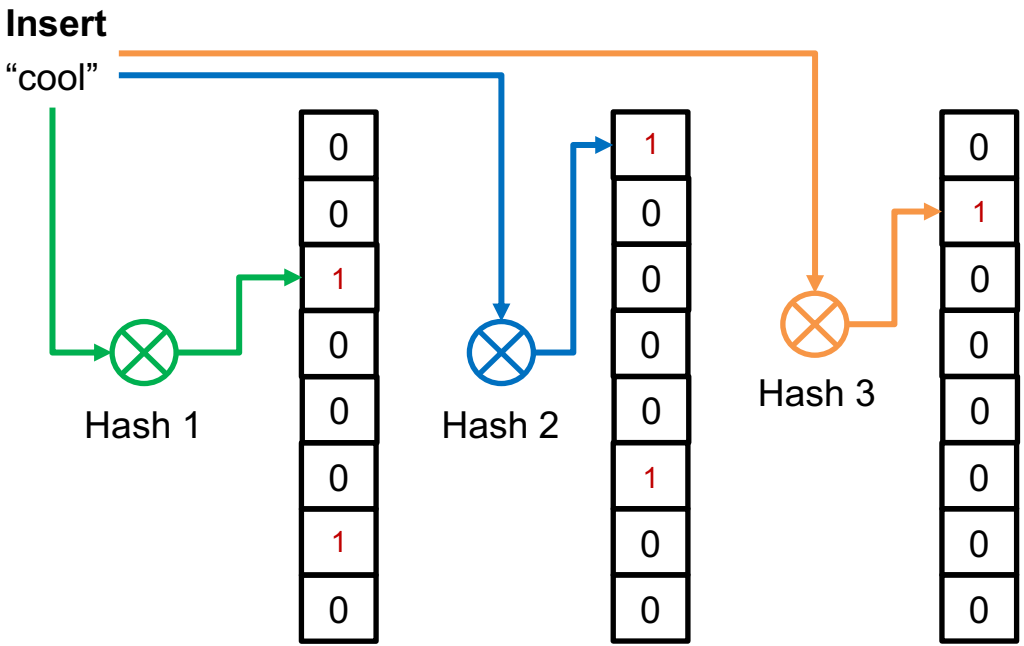
- **A memory efficient approach for insertions and membership queries**



Slide credit: Laurent Vanbever

Bloom Filter

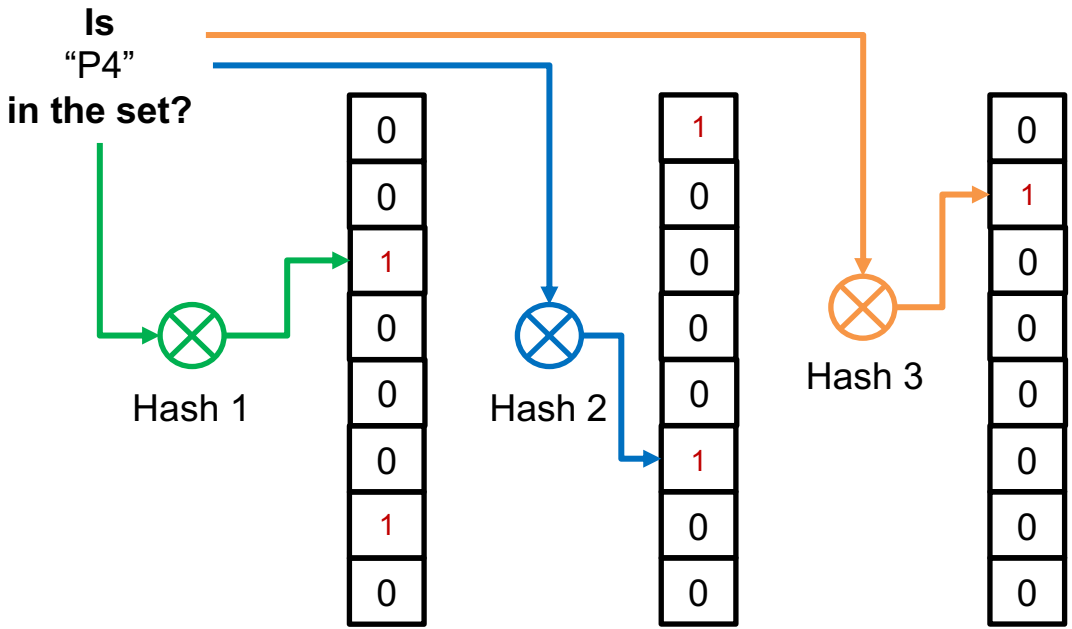
- **A memory efficient approach for insertions and membership queries**



Slide credit: Laurent Vanbever

Bloom Filter

- **A memory efficient approach for insertions and membership queries**



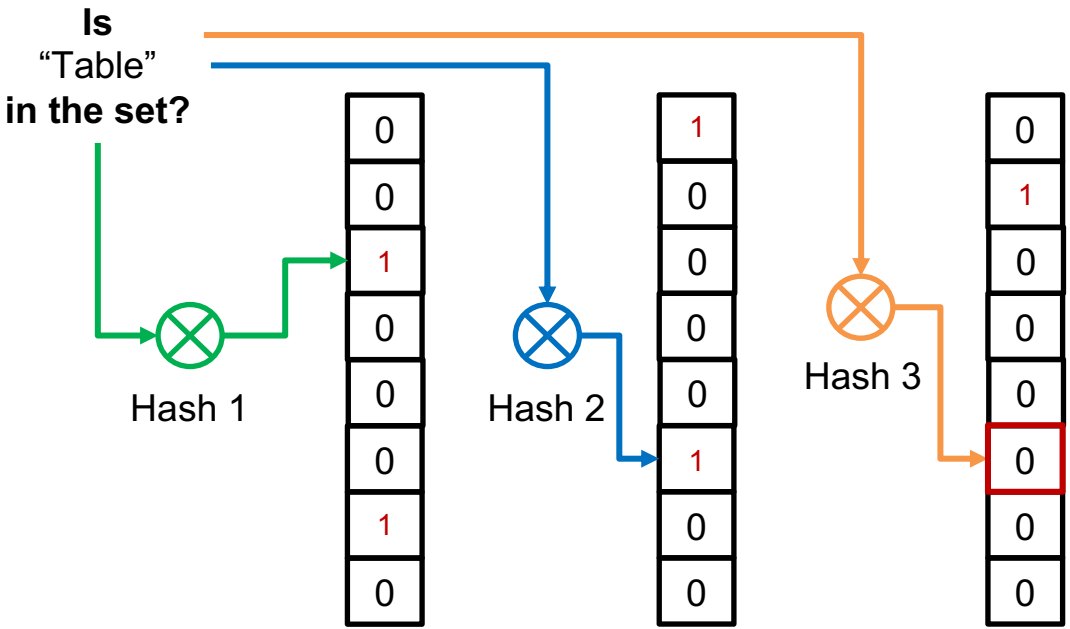
- An element is ***in the set*** if **all** the hash values map to a cell with 1

- An element ***is not in the set*** if ***at least*** one hash value maps to a cell with 0

Slide credit: Laurent Vanbever

Bloom Filter

- **A memory efficient approach for insertions and membership queries**



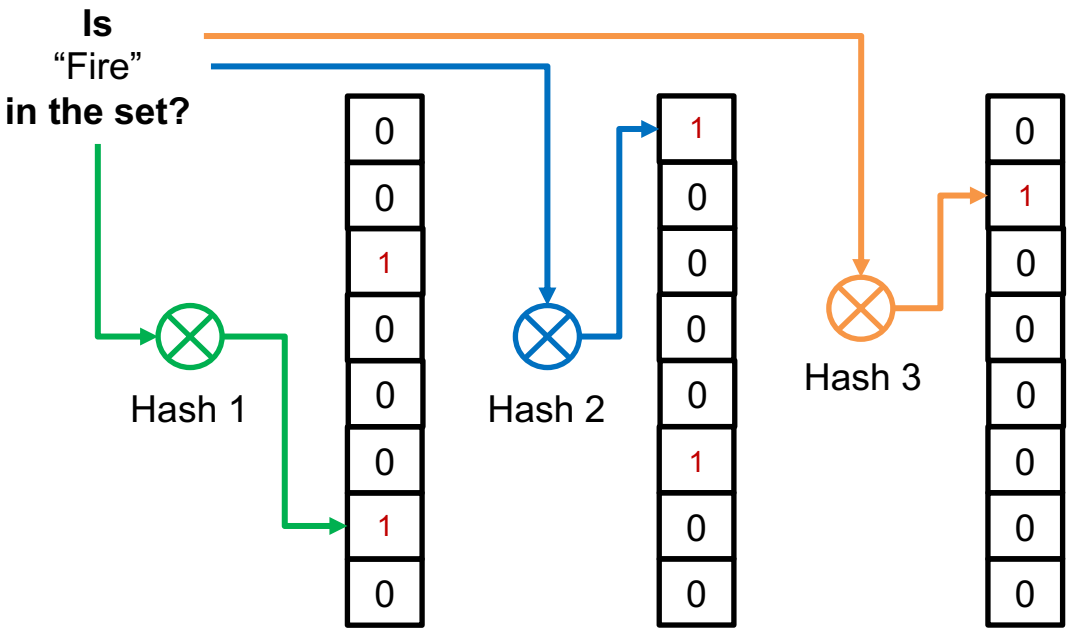
- An element is ***in the set*** if **all** the hash values map to a cell with 1

- An element ***is not in the set*** if ***at least*** one hash value maps to a cell with 0

Slide credit: Laurent Vanbever

Bloom Filter

- **A memory efficient approach for insertions and membership queries**



- An element is ***in the set*** if **all** the hash values map to a cell with 1
- An element ***is not in the set*** if ***at least*** one hash value maps to a cell with 0

FALSE POSITIVE

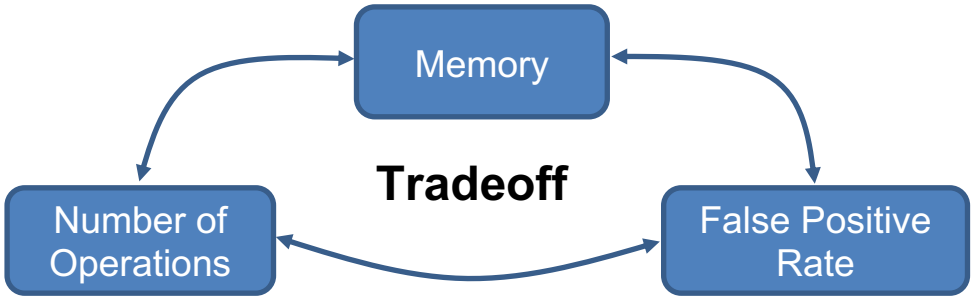
Slide credit: Laurent Vanbever

Bloom Filter

- ***N* elements, *M* cells, and *K* hash functions**
- **False Positive Rate =** $\left(1-\left(1-\frac{1}{M}\right)^{KN}\right)^{K}$

# of elements	# of cells	# of hash functions	False Positive Rate
1000	10000	7	0.82%
1000	100000	7	≈ 0%

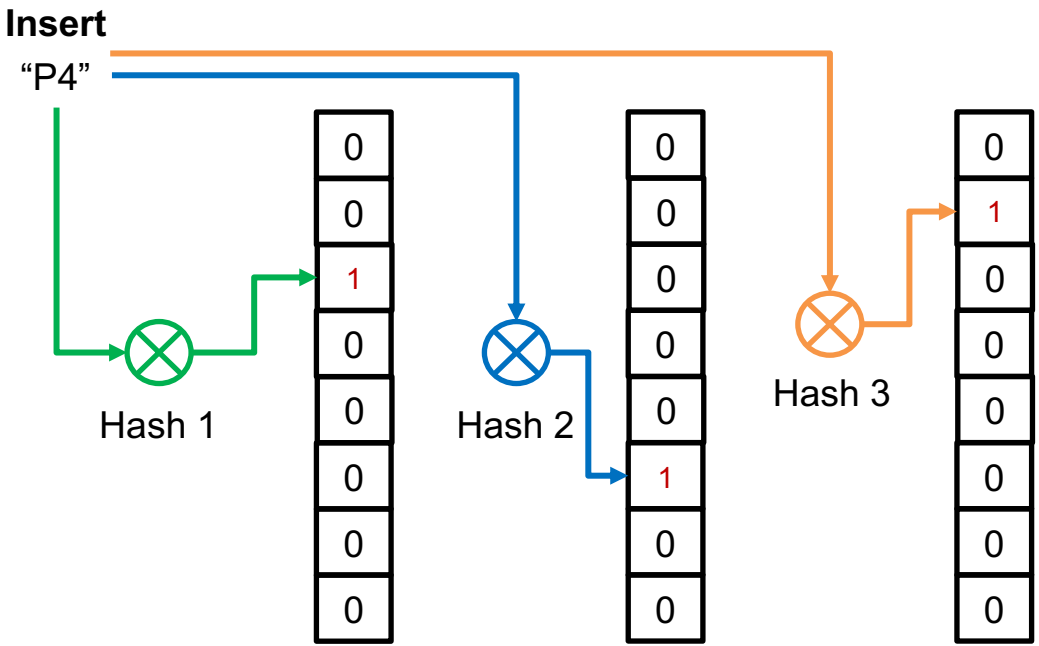
- **Can be easily extended to handle deletions – Counting Bloom Filter**



Slide credit: Laurent Vanbever

CountMin-Sketch

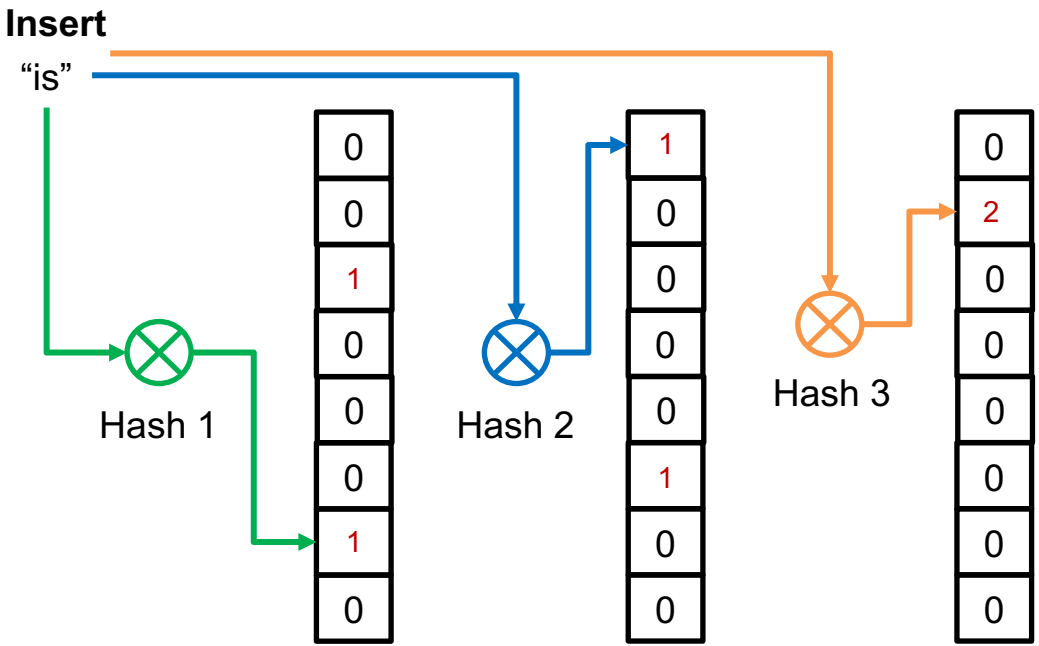
- A memory efficient approach for counting approximate frequencies of elements in a stream



result = min (1, 1, 1) = 1

CountMin-Sketch

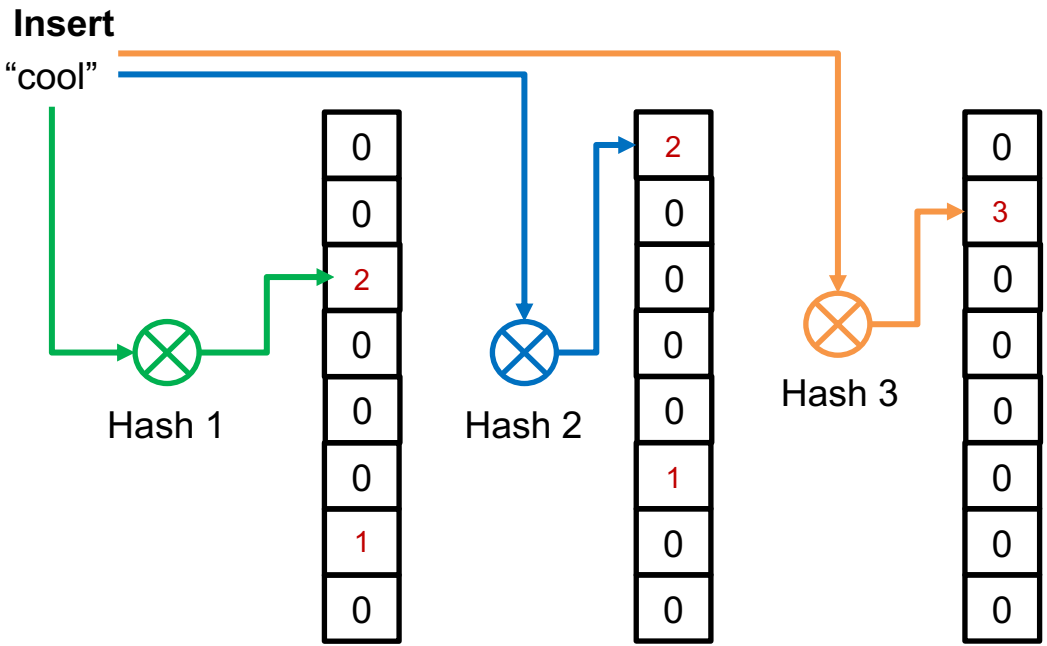
- **A memory efficient approach for counting approximate frequencies of elements in a stream**



result = min (1, 1, 2) = 1

CountMin-Sketch

- A memory efficient approach for counting approximate frequencies of elements in a stream



result = min (2, 2, 3) = 2

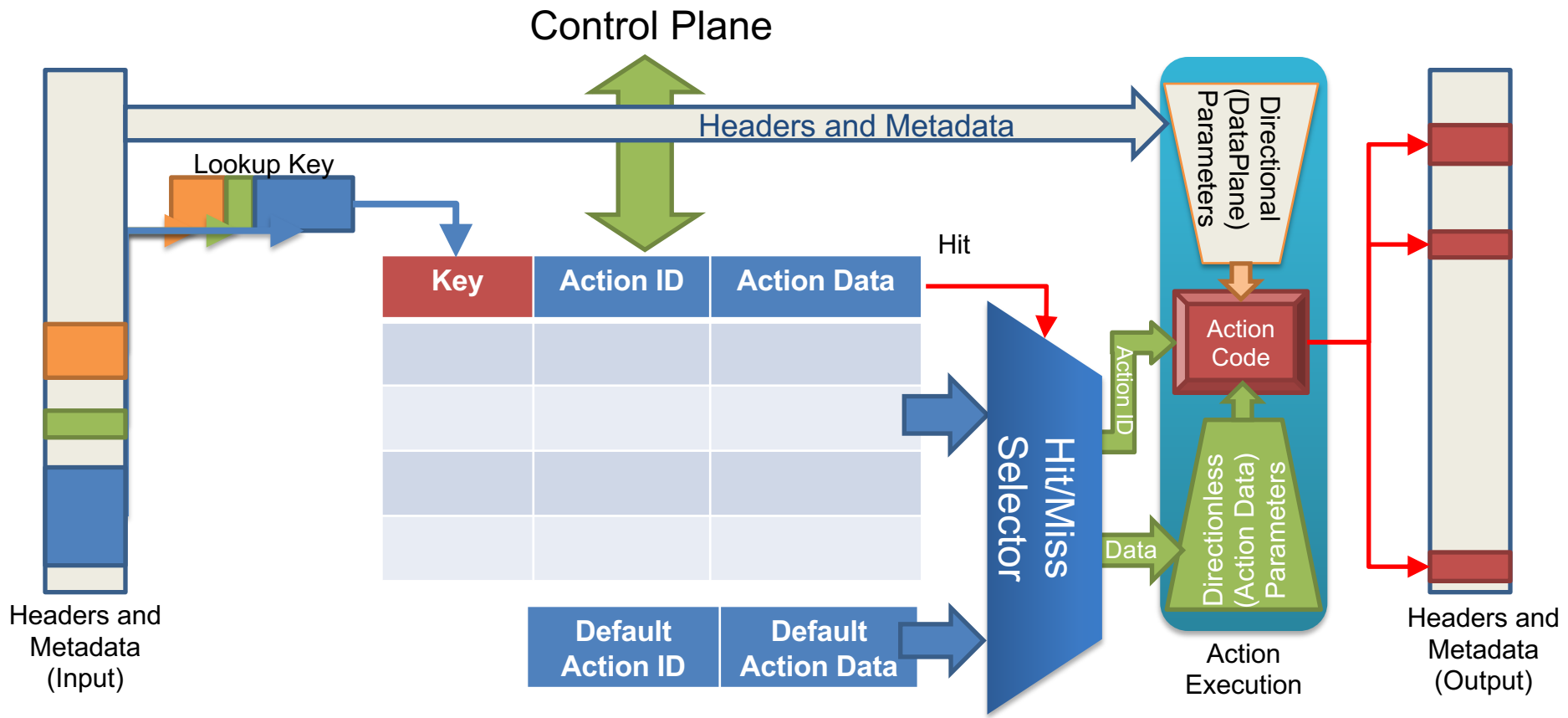
- Hash collisions cause over counting
- Use minimum value to minimize error
- Provable L1 error bound

Many other Bloom Filters and Sketches

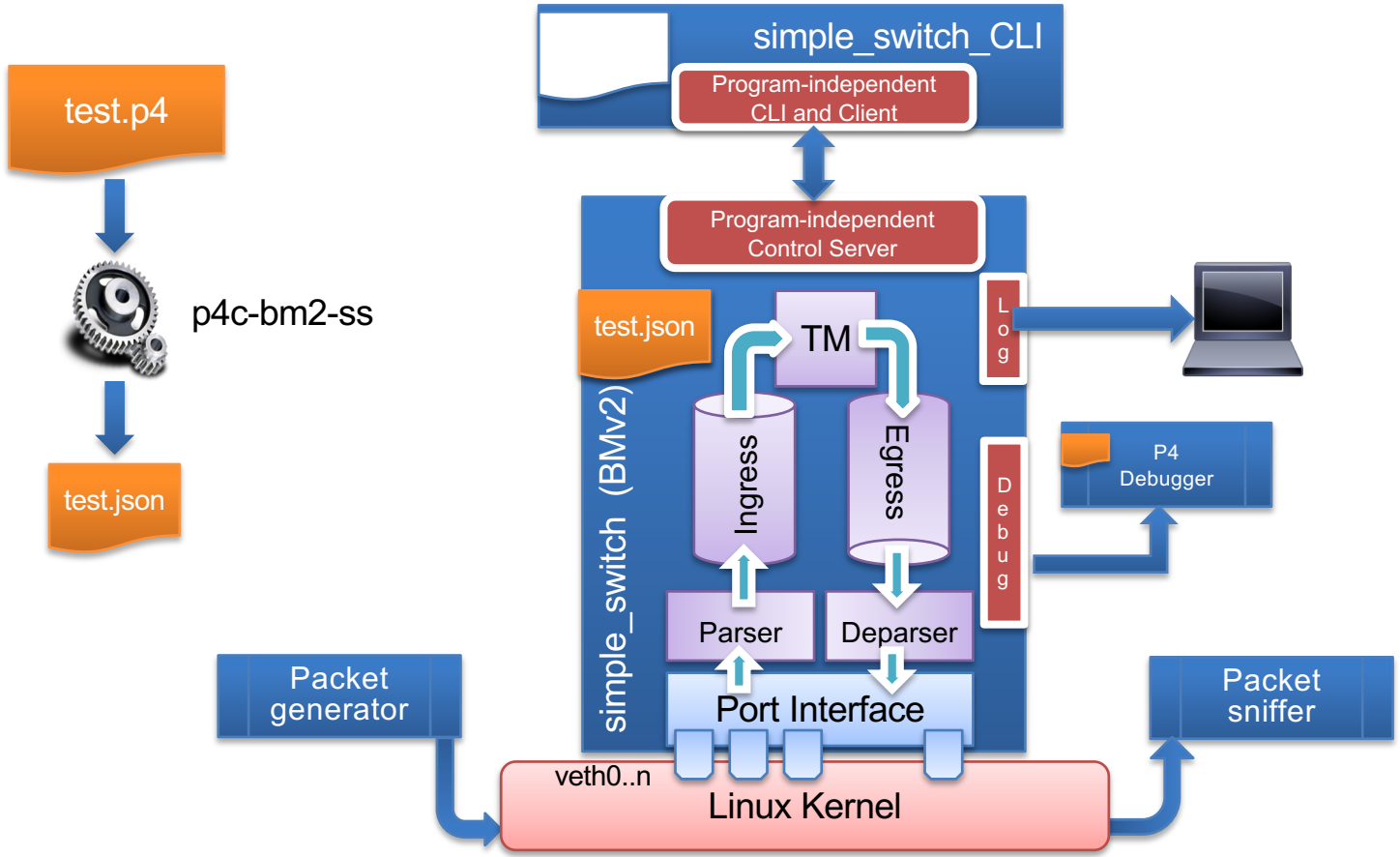
- Counting Bloom Filter
- Invertible Bloom Lookup Table
- Count Sketch
- Hyper Log-Log Sketch
- OpenSketch (NSDI 13)
- UnivMon (SIGCOMM 16)
- FlowRadar (NSDI 16)
- SketchLearn (SIGCOMM 18)

Backup Slides

Tables: Match-Action Processing



Makefile: under the hood



Working with Tables in simple_switch_CLI

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

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

RuntimeCmd: table_dump m_table
m_table:
0: 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

P4$_{16}$ Program Template (V1Model)

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

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

$P4_{16}$ Hello World (V1Model)

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

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

    state start { transition accept; }
}

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

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

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

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

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

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

P4$_{16}$ Hello World (V1Model)

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

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

control MyIngress(inout headers hdr, inout metadata meta,
   inout standard_metadata_t standard_metadata) {
    action set_egress_spec(bit<9> port) {
        standard_metadata.egress_spec = port;
    }
    table forward {
        key = { standard_metadata.ingress_port: exact; }
        actions = {
            set_egress_spec;
            NoAction;
        }
        size = 1024;
        default_action = NoAction();
    }
    apply {   forward.apply();   }
}
```

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

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

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

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

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

Key	Action ID	Action Data
1	set_egress_spec ID	2
2	set_egress_spec ID	1

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**