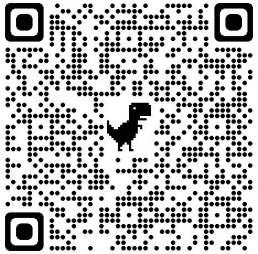


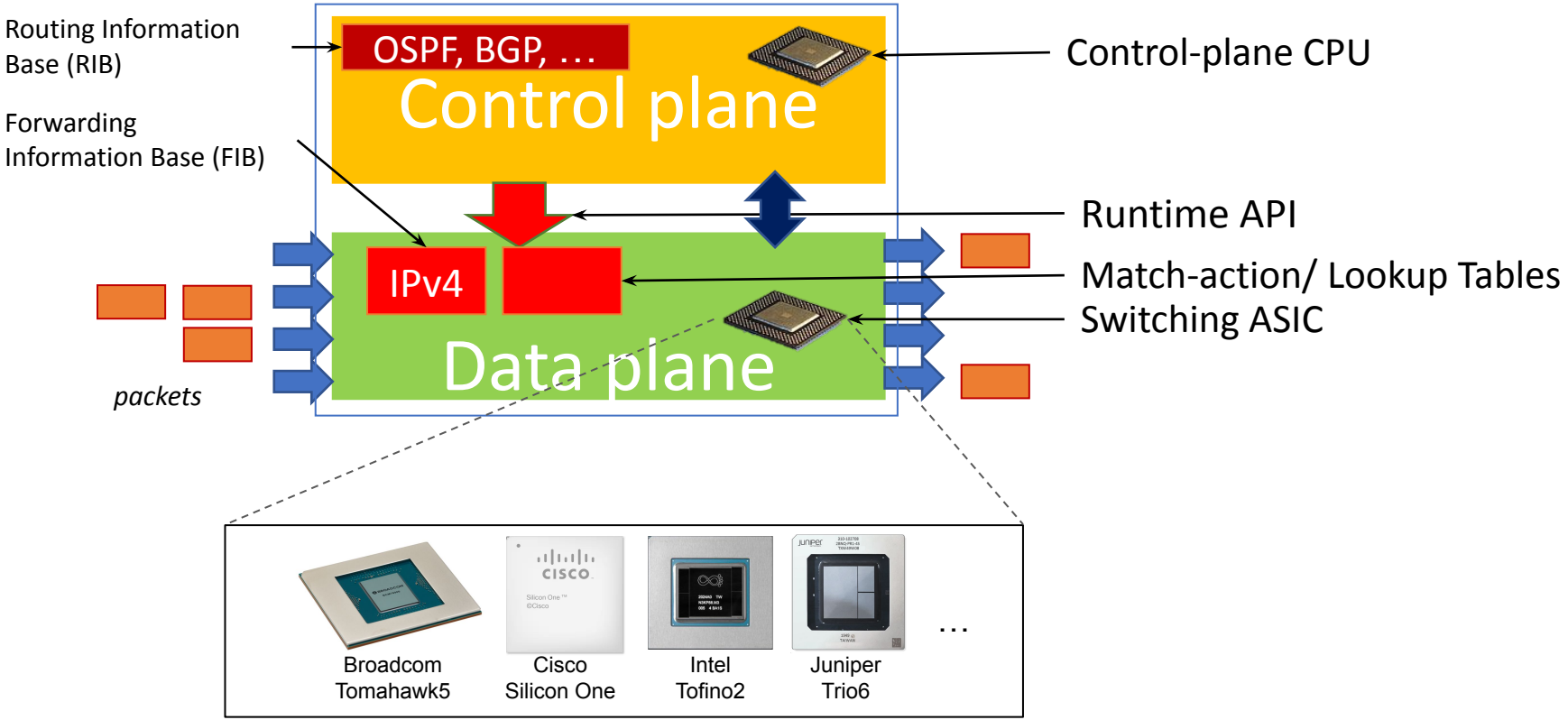
Introduction to P4 Programming

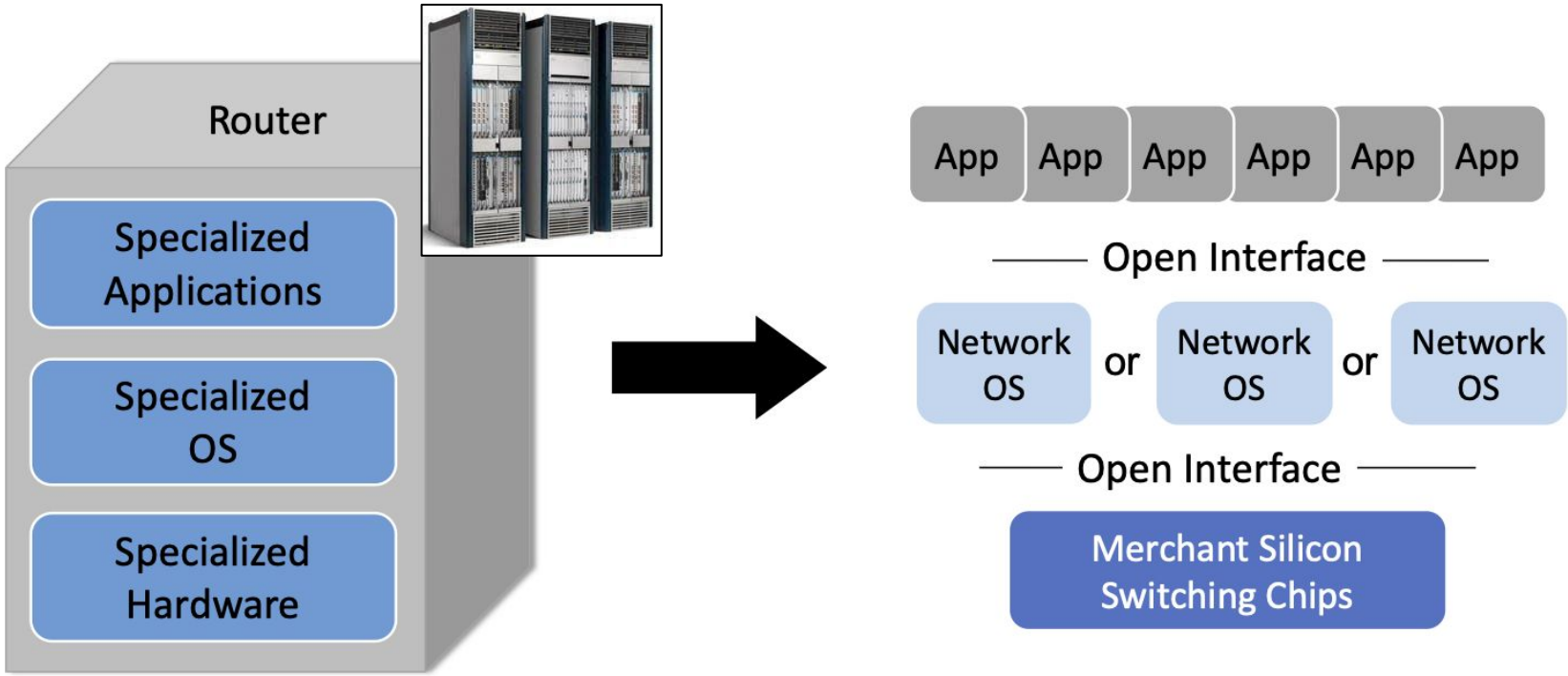
CS5229 Advanced Computer Networks

TA: Khooi Xin Zhe

Why Data Plane Programmability?

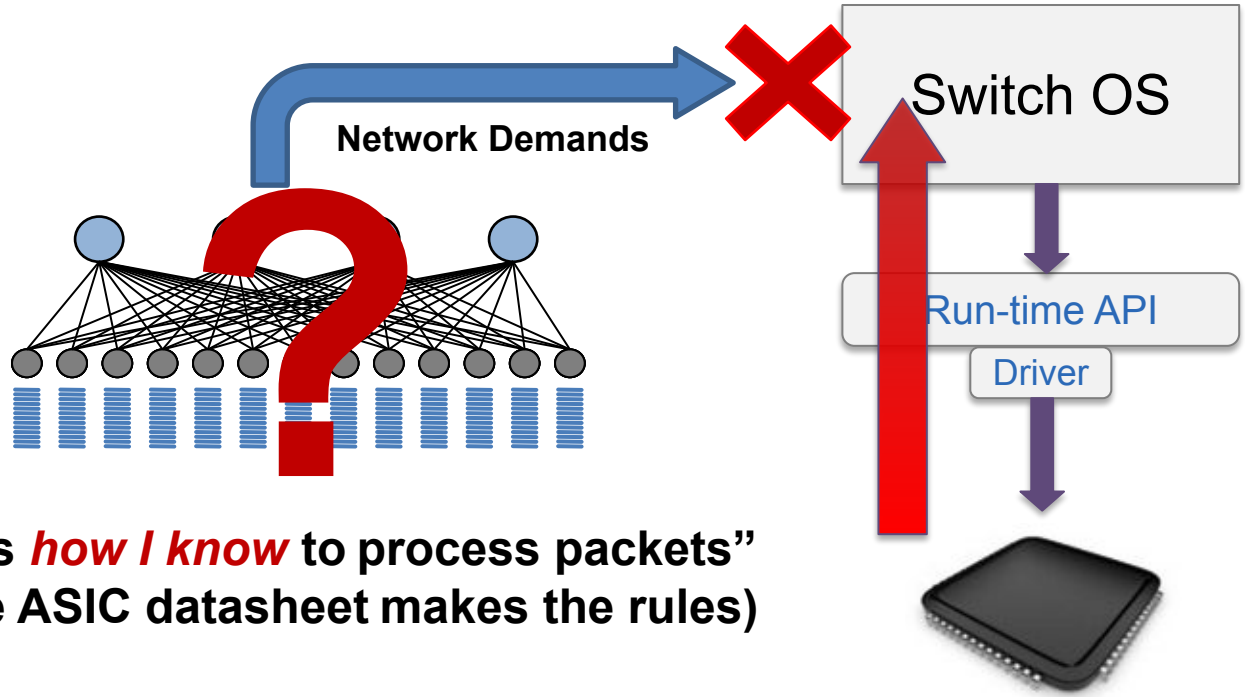
Recap: Switch/ Router Architectures





Source: https://sdn.systemsapproach.org/intro.html

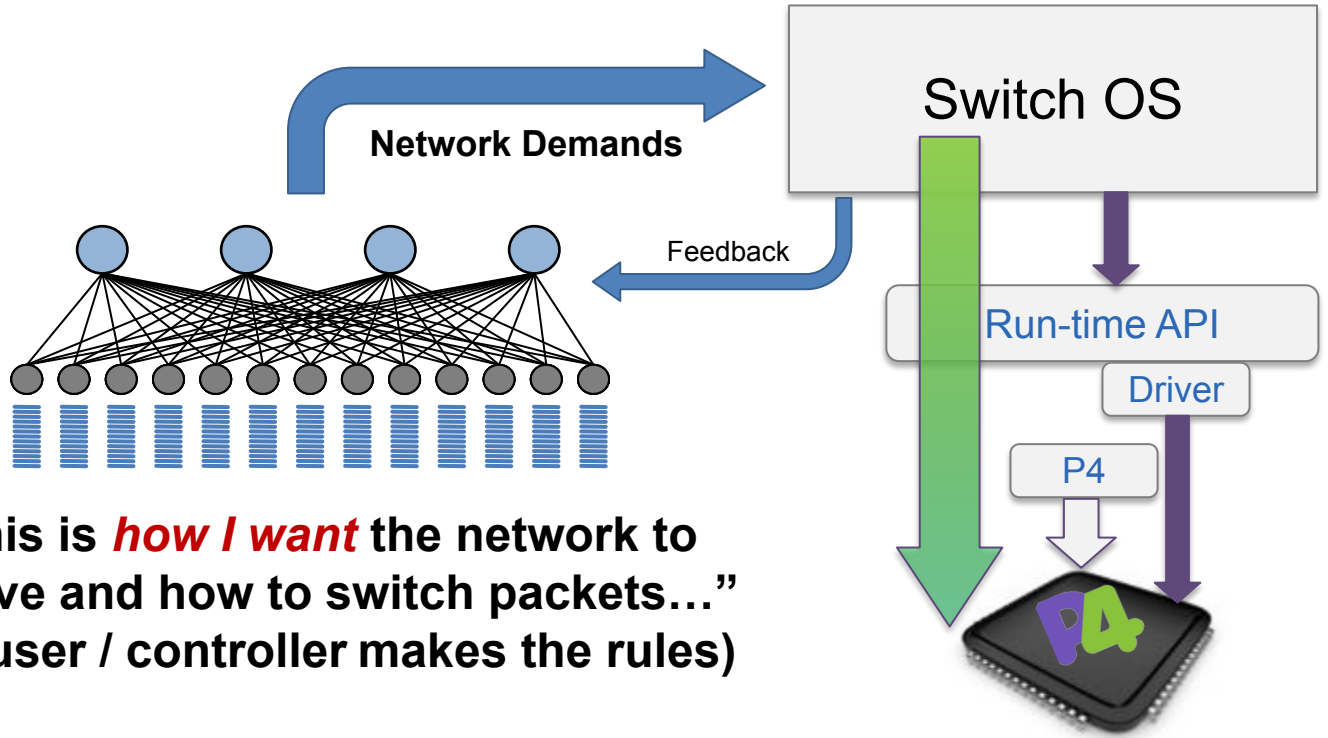
Bottom-up Design



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

Inflexible data plane, slows down development!

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

Reconfigurable Match-action Tables (RMT)

Forwarding Metamorphosis: Fast Programmable Match-Action Processing in Hardware for SDN

Pat Bosshart[†], Glen Gibb[‡], Hun-Seok Kim[†], George Varghese[§], Nick McKeown[‡],
Martin Izzard[†], Fernando Mujica[†], Mark Horowitz[‡]
[†]Texas Instruments [‡]Stanford University [§]Microsoft Research
pat.bosshart@gmail.com {grg, nickm, horowitz}@stanford.edu
varghese@microsoft.com {hkim, izzard, fmujica}@ti.com

ABSTRACT

In Software Defined Networking (SDN) the control plane is physically separate from the forwarding plane. Control software programs the forwarding plane (e.g., switches and routers) using an open interface, such as OpenFlow. This paper aims to overcomes two limitations in current switching chips and the OpenFlow protocol: i) current hardware switches are quite rigid, allowing "Match-Action" processing on only a fixed set of fields, and ii) the OpenFlow specification only defines a limited repertoire of packet processing actions. We propose the RMT (reconfigurable match tables) model, a new RISC-inspired pipelined architecture for switching chips, and we identify the essential minimal set of action primitives to specify how headers are processed in hardware. RMT allows the forwarding plane to be changed in the field without modifying hardware. As in OpenFlow, the programmer can specify multiple match tables of arbitrary width and depth, subject only to an overall resource limit, with each table configurable for matching on arbitrary fields. However, RMT allows the programmer to modify *all* header fields much more comprehensively than in OpenFlow. Our paper describes the design of a 64 port by 10 Gb/s switch chip implementing the RMT model. Our concrete design demonstrates, contrary to concerns within the community, that flexible OpenFlow hardware switch implementations are feasible at almost no additional cost or power.

1. INTRODUCTION

To improve is to change; to be perfect is to change often. — Churchill

Good abstractions—such as virtual memory and time-sharing—are paramount in computer systems because they allow systems to deal with change and allow simplicity of programming at the next higher layer. Networking has progressed because of key abstractions: TCP provides the abstraction of connected queues between endpoints, and IP provides a simple datagram abstraction from an endpoint to the network edge. However, routing and forwarding *within* the network remain a confusing conglomerate of routing protocols (e.g., BGP, ICMP, MPLS) and forwarding behaviors (e.g., routers, bridges, firewalls), and the control and forwarding planes remain intertwined inside closed, vertically integrated boxes.

Software-defined networking (SDN) took a key step in abstracting network functions by separating the roles of the control and forwarding planes via an *open* interface between them (e.g., OpenFlow [27]). The control plane is lifted up and out of the switch, placing it in external software. This programmatic control of the forwarding plane allows network owners to add new functionality to their network, while replicating the behavior of existing protocols. OpenFlow has become quite well-known as an interface between the con-

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**

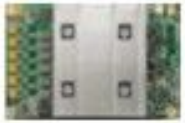
Think programming rather than protocols…

Innovations Enabled by Data Plane Programmability

- **Fine-grained RDMA Load Balancing – ConWeave[1]**
- **In-band Network Telemetry – INT[2]**
- **In-Network Caching – NetCache[3]**
- **In-Network Aggregation for Distributed ML [4]**
- **Masking Corrupted Fibre Optic Links[5]**
- **Byzantine Fault Tolerance Acceleration - NeoBFT[6]**
- **5G FrontHaul Network Slicing - FSA[7]**
- **(Very) Precise Time Synchronisation - DPTP[8]**
- **… and much more!**

[1] CH Song, et al., "Network Load Balancing with In-network Reordering Support for RDMA", *ACM SIGCOMM*, 2023.
[2] C Kim, et al. "In-band network telemetry via programmable data planes." *ACM SIGCOMM*. 2015.
[3] J Xin, et al. "NetCache: Balancing Key-Value Stores with Fast In-Network Caching." *ACM SOSP*. 2017.
[4] A Sapio, et al. "Scaling Distributed Machine Learning with In-Network Aggregation." *USENIX NSDI*, 2021.
[5] J Raj, et. al., "Masking Corruption Packet Losses in Datacenter Networks with Link-local Retransmission", *ACM SIGCOMM*, 2023.
[6] G Sun, et. al., "NeoBFT: Accelerating Byzantine Fault Tolerance Using Authenticated In-Network Ordering", *ACM SIGCOMM*, 2023.
[7] B Nishant, et. al., "FSA: Fronthaul Slicing Architecture for 5G using dataplane programmable switches", *ACM MOBICOM*, 2021.
[8] PG Kannan et. al., "Precise Time-synchronization in the Data-Plane using Programmable Switching ASICs", ACM SOSR, 2019.

General Industry Trend: Rise of the Domain Specific Architectures (DSAs)

Computers	Graphics	Signal Processing	Machine Learning		Networking
Java	OpenCL	Matlab	TensorFlow	>>>	P4
Compiler	Compiler	Compiler	Compiler		Compiler
					
CPU	GPU	DSP	TPU		Tofino & Tofino 2 (PISA)

Questions?

Introduction to P4

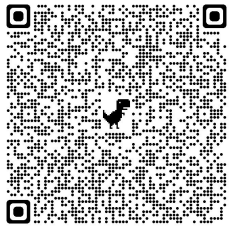
Language evolution

- P4: Programming Protocol-Independent Packet Processors
  - Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, David Walker *ACM SIGCOMM Computer Communications Review (CCR). Volume 44, Issue #3 (July 2014)*

- **P4_14**
  - Original version of the language
  - Assumed specific devices capabilities
  - Good only for a subset of programmable switch/ targets

- **P4_16**
  - More mature and stable language definition
  - Does not assume device capabilities, which instead are defined by target manufacturer via external libraries/ architecture definition
  - Good for many targets, e.g., switches/ NICs, whether programmable or fixed-function

P4$_{16}$ Language Specification



version 1.2.4

The P4 Language Consortium

2023-05-15

Abstract. P4 is a language for programming the data plane of network devices. This document provides a precise definition of the P4$_{16}$ language, which is the 2016 revision of the P4 language (http://p4.org).

The target audience for this document includes developers who want to write compilers, simulators, IDEs, and debuggers for P4 programs. This document may also be of interest to P4 programmers who are interested in understanding the syntax and semantics of the language at a deeper level.

Contents

Available Software Tools

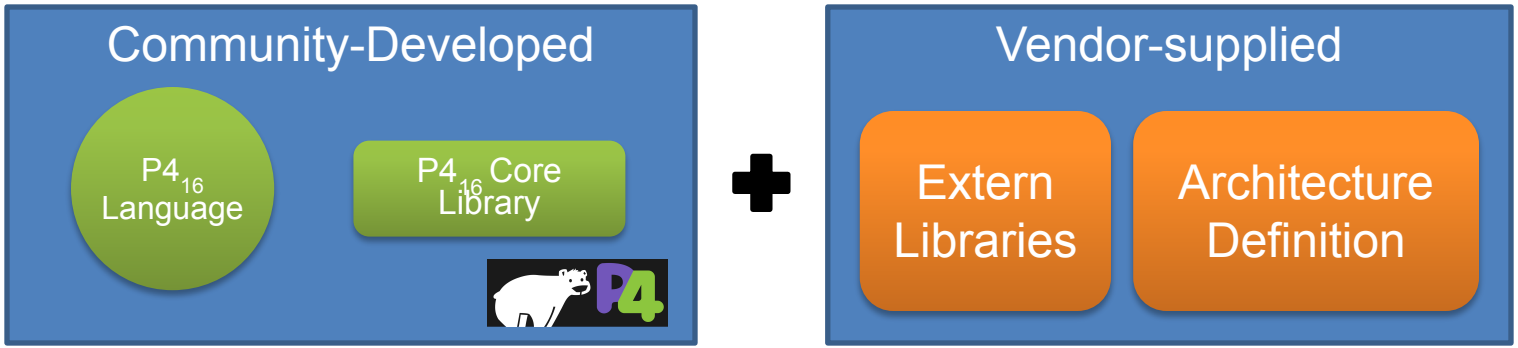
- Compilers for various back-ends
  - Intel Tofino ASIC, DPDK, eBPF, Xilinx FPGA (open-source and proprietary)
- Multiple control-plane implementations
  - SAI, OpenFlow, P4Runtime
- Software Simulators/ Models
- Testing tools (Packet Test Framework)
- Sample P4 programs
- Tutorials

DPDK
DATA PLANE DEVELOPMENT KIT

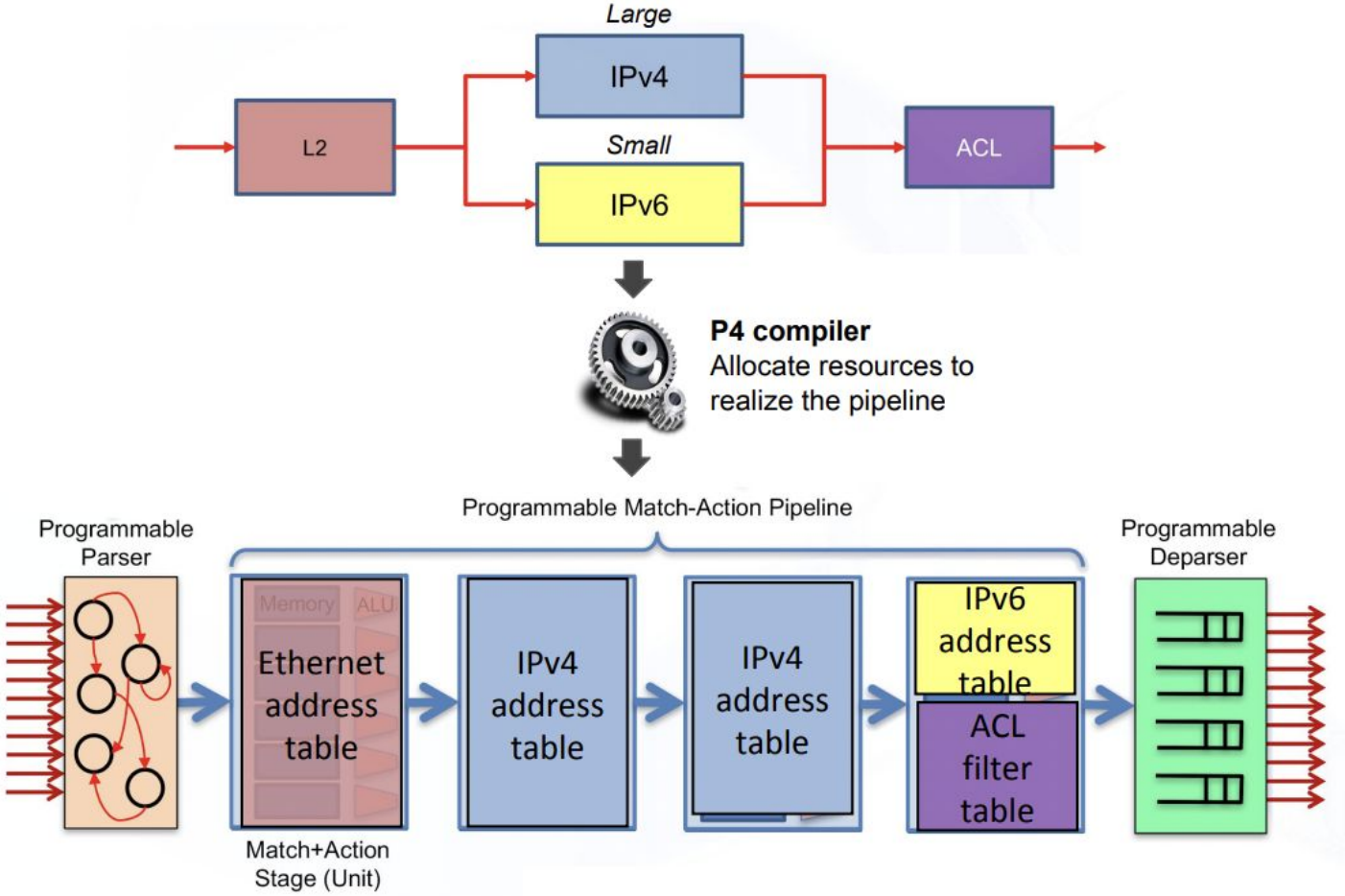
eBPF

P4 Approach

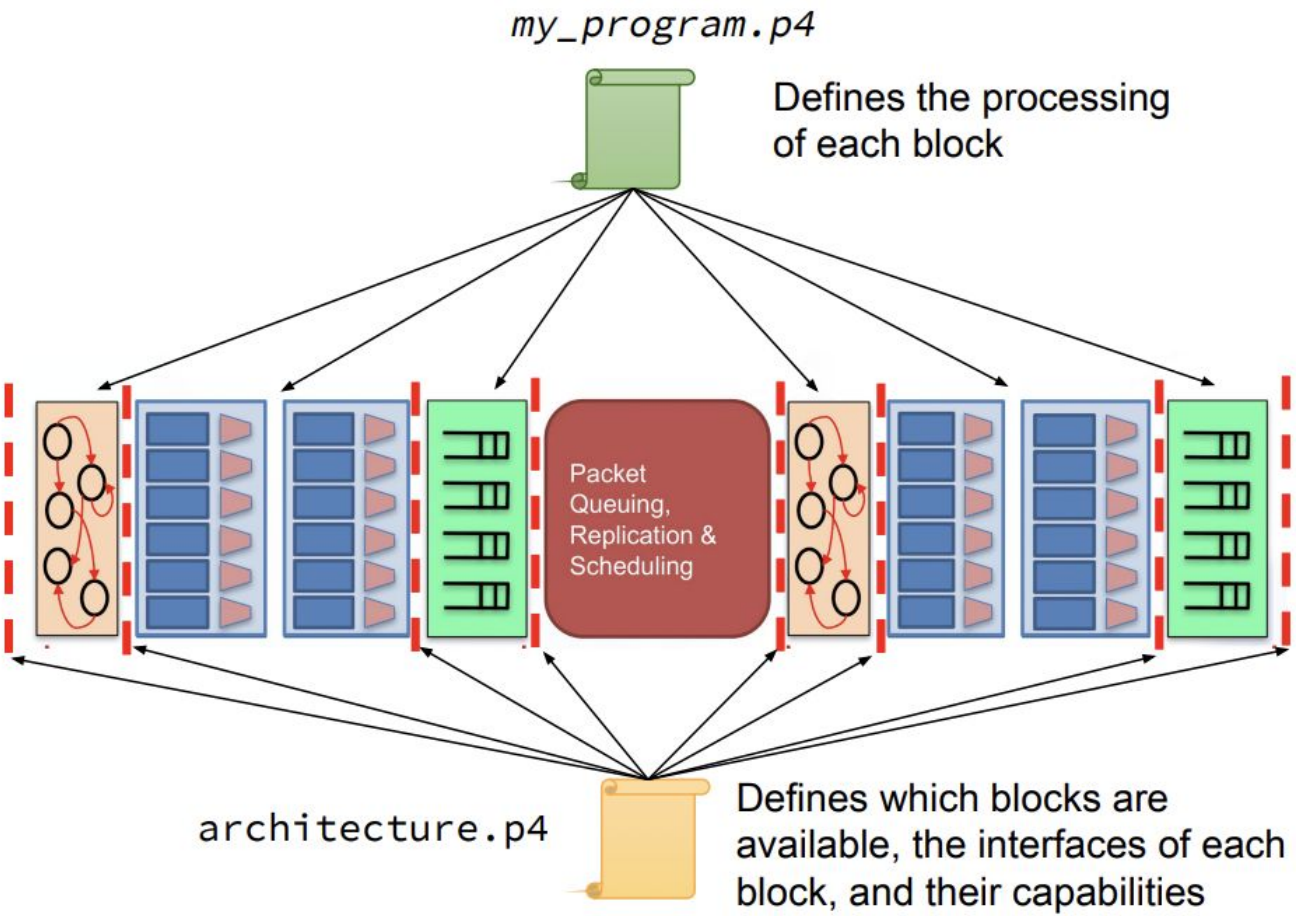
Term	Explanation
P4 Target	An embodiment of a specific hardware implementation
P4 Architecture	Provides an interface to program a target via some set of P4-programmable components, externs, fixed components



Mapping a simple logical pipeline on PISA

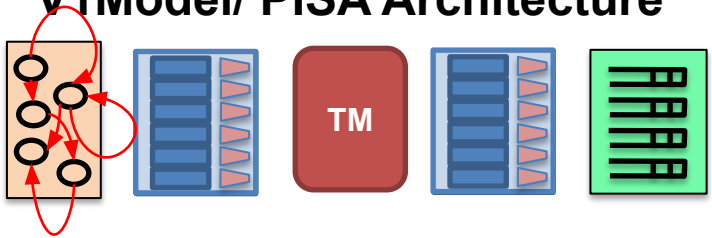


P4 programs and architectures



Example Architectures and Targets

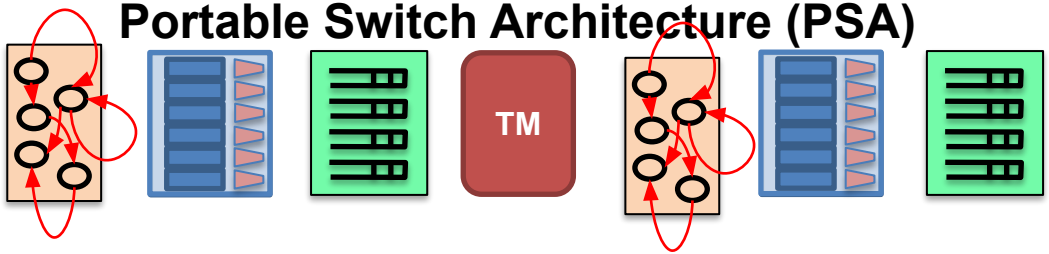
V1Model/ PISA Architecture



Software Switch

Tofino Native Architecture*
(P4_14)

Portable Switch Architecture (PSA)

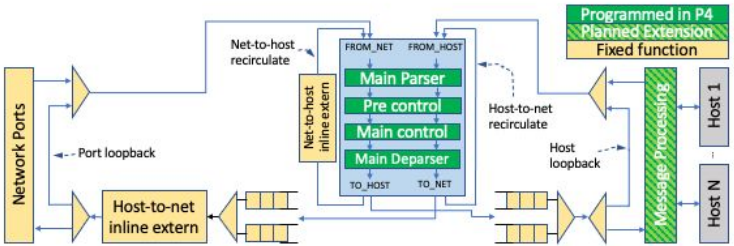


DPDK
DATA PLANE DEVELOPMENT KIT

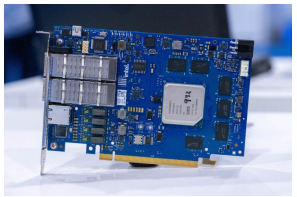
Software Switch

Tofino Native Architecture*

Portable NIC Architecture (PNA)



DPDK
DATA PLANE DEVELOPMENT KIT

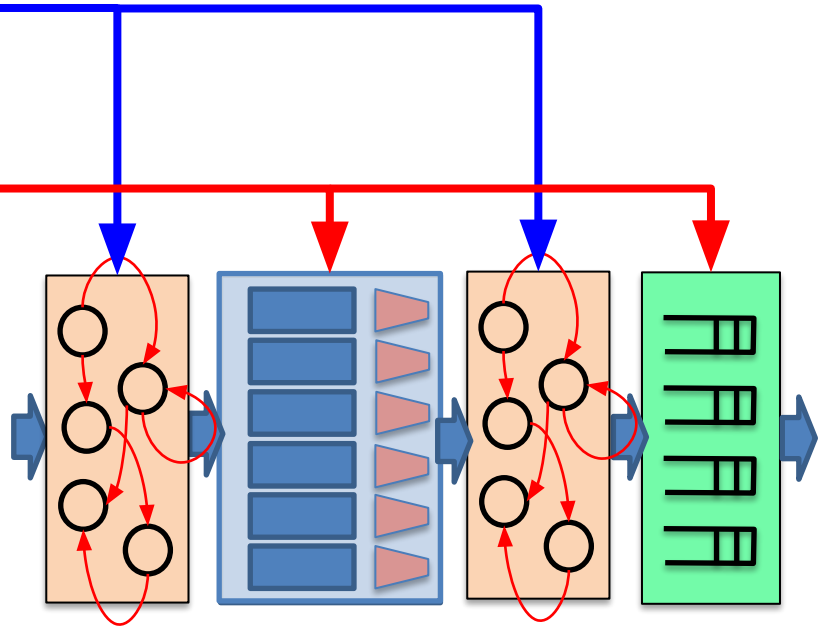


SmartNICs (IPUs, DPUs…)

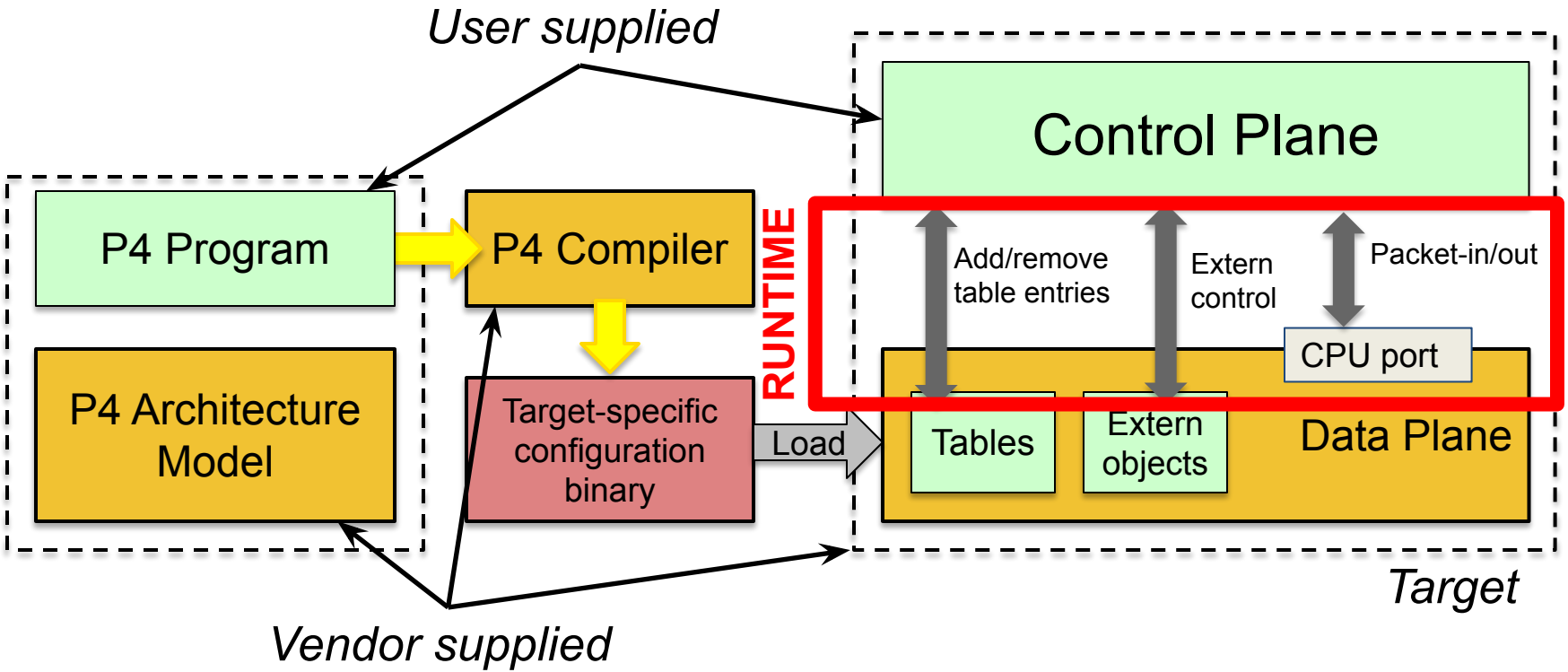
P4 Language Elements

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

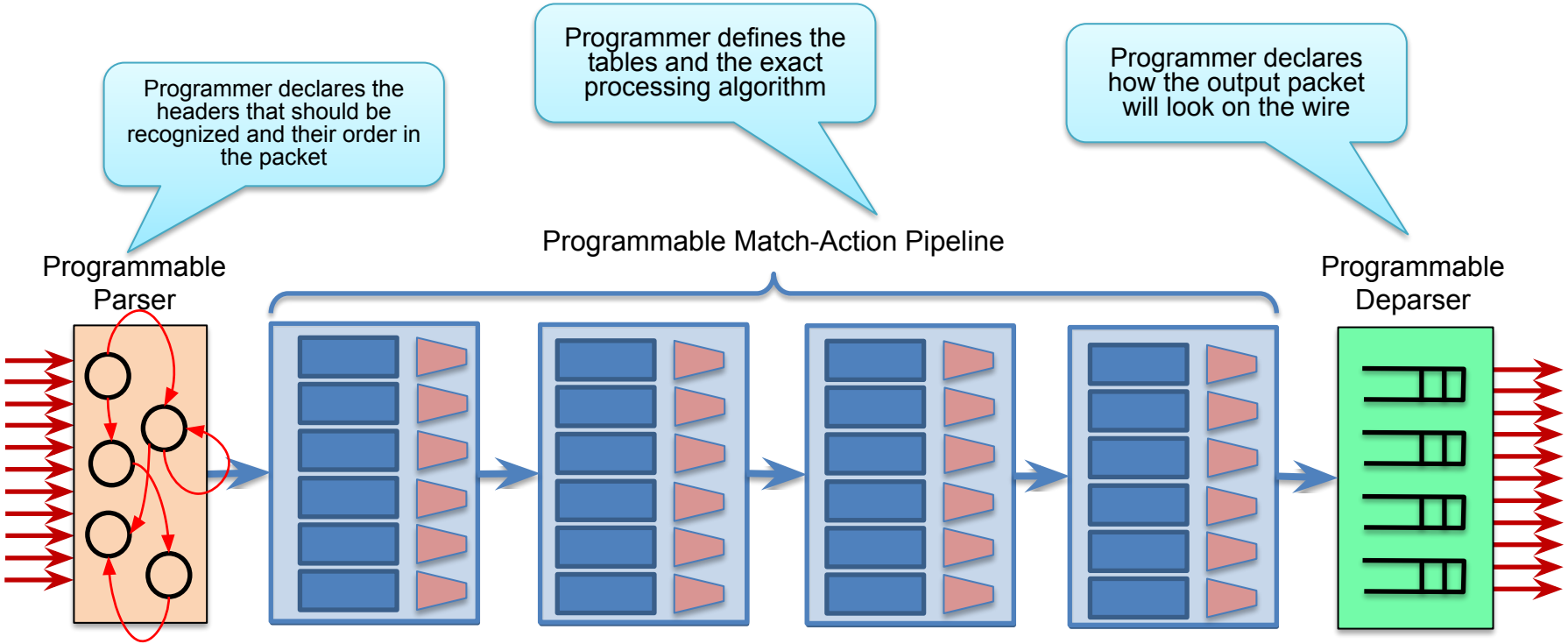
- **Architecture Description** — Programmable blocks and their interfaces
- **Extern Libraries** — Support for specialized components



Programming a P4 Target



Protocol-Independent Switch (PISA)/ V1Model Architecture



P4 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;
```

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)

Simple P4 Program Example

```
#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;
```

Questions?

Defining and (De-)Parsing Headers

P4 Types (Basic and Header Types)

```
typedef bit<48> macAddr_t;

header ethernet_t {
  macAddr_t dstAddr;
  macAddr_t srcAddr;
  bit<16>   etherType;
}
```

Basic Types

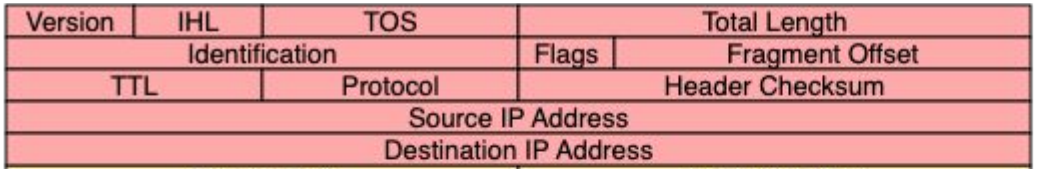
- **bit<n>**: Unsigned integer (bitstring) of size n
- **bit** is the same as **bit<1>**
- **int<n>**: Signed integer of size n (>=2)
- **varbit<n>**: Variable-length bitstring

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

Example: IPv4 Header




```
typedef bit<32> ip4Addr_t;

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;
}
```

P4 Types (Other Types)

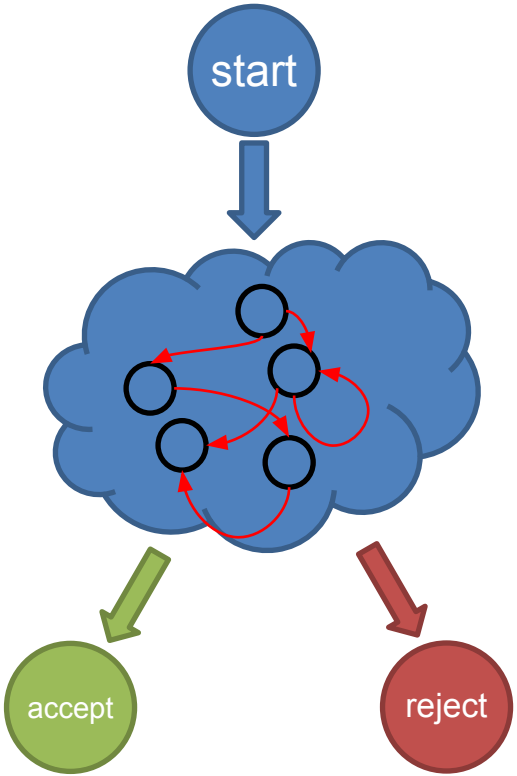
```
/* Architecture */
struct standard_metadata_t {
  bit<9>  ingress_port;
  bit<9>  egress_spec;
  bit<9>  egress_port;
  bit<32> clone_spec;
  bit<32> instance_type;
  bit<1>  drop;
  bit<16> recirculate_port;
  bit<32> packet_length;
  ...
}

/* 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

Programmable Parsers

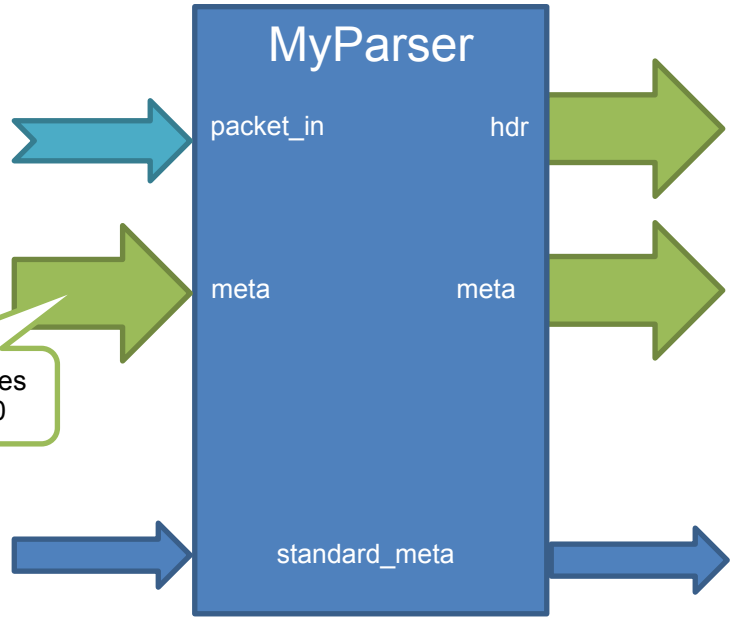
- Parsers are functions that map packets into headers and metadata, written in a state machine style
- Every parser has three predefined states
  - start
  - accept
  - reject
- Other states may be defined by the programmer
- In each state, execute zero or more statements, and then transition to another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



Intuition: Think of JSON Serialization!

V1Model Parser

```
/* User Program */
parser MyParser(packet_in packet,
          out headers hdr,
          inout metadata meta,
          inout standard_metadata_t std_meta) {

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

}
```



“select” statement

```
state start {
  transition parse_ethernet;
}

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

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

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

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

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

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

Deparsing

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

- Assembles the headers back into a well-formed packet
- **emit(hdr)**: serializes header if it is valid
- Advantages:
  - Makes deparsing explicit but decouples from parsing

Intuition: Think of JSON Deserialization!

Questions?

(Stateless) Packet Processing

P4 Control Blocks

- 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

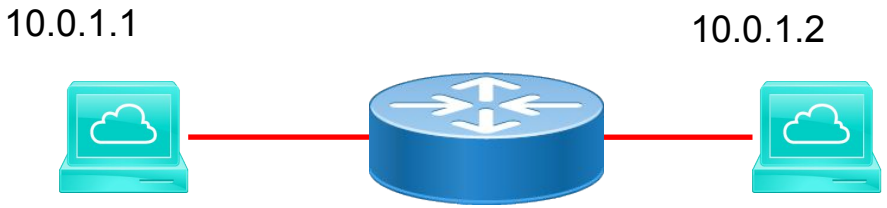
```
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t std_meta) {

  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 Match-action 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
    - …
- 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)

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

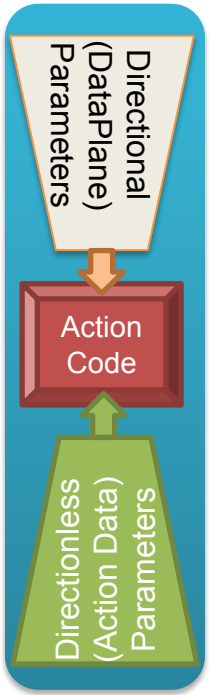
IPv4_LPM Table

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

Defining Actions

```
/* core.p4 */
action NoAction() {
}

/* basic.p4 */
action drop() {
  mark_to_drop();
}

/* basic.p4 */
action ipv4_forward(macAddr_t dstAddr,
                    bit<9> port) {
  ...
}
```

- **Actions can have two different types of parameters**
  - Directional (from the Data Plane)
  - Directionless (from the Control Plane)
- **Actions that are called directly:**
  - Only use directional parameters
- **Actions used in tables:**
  - Typically use directionless parameters
  - May sometimes use directional parameters too



Action Execution

Applying Tables in Controls

```
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t standard_metadata) {
  action drop(){
    …
  }

  table ipv4_lpm {
    ...
  }
  apply {
     ...
     ipv4_lpm.apply();
     ...
  }
}
```

Demo

Questions?

(Stateless) Packet Processing - cont.

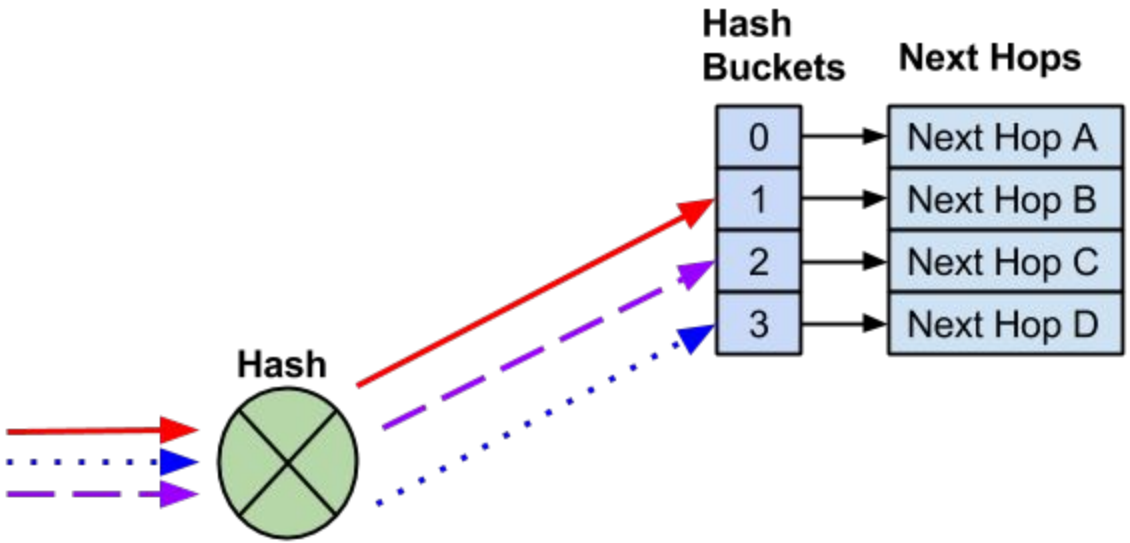
Hashing (V1Model)

```
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);

bit<10> variable_to_output;
hash (variable_to_output, HashAlgorithm.crc32, 0,
{hdr.ipv4.src_addr, … }, 1024);

// here we have the hash value ready
```

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.

ECMP as a Use Case



Mirroring (V1Model)

```
…
        clone(CloneType.I2E, 0);
…
```

Mirrors a copy of a packet for further processing

Primitives supported are:
CloneType.I2E
CloneType.E2E

Multicast (V1Model)

```
…

standard_metadata.mcast_grp = GROUP_NUM;
…
```

Replicates multiple copies of the packets, used for multicast/ broadcast

Group members are configured through the control plane to the Traffic Manager

Stateful Packet Processing

What is “Stateful Packet Processing”?

- Packet processing decisions made on a particular packet is dependent on states set by previous packets
- Examples:
  - Heavy-hitter detection
  - DDoS defense
  - L4 Firewall
  - NAT forwarding
  - Flowlet switching
  - …

Registers in V1 Model

Definition

```
/* Definition in v1model.p4 */

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

Usage (Calculating Inter-Packet Gap)

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

action get_inter_packet_gap(out bit<48> interval,
                            bit<14> flow_id)
{
   bit<48> last_pkt_ts;

   /* Get the time the previous packet was seen */
   last_seen.read((bit<32>)flow_id,
                  last_pkt_ts);

   /* Calculate the time interval */
   interval = standard_metadata.ingress_global_timestamp -
              last_pkt_ts;

   /* Update the register with the new timestamp */
   last_seen.write((bit<32>)flow_id,
                   standard_metadata.ingress_global_timestamp);
}
```

Note: TYPO, order of the arguments is wrong for “read”

Registers (V1Model)

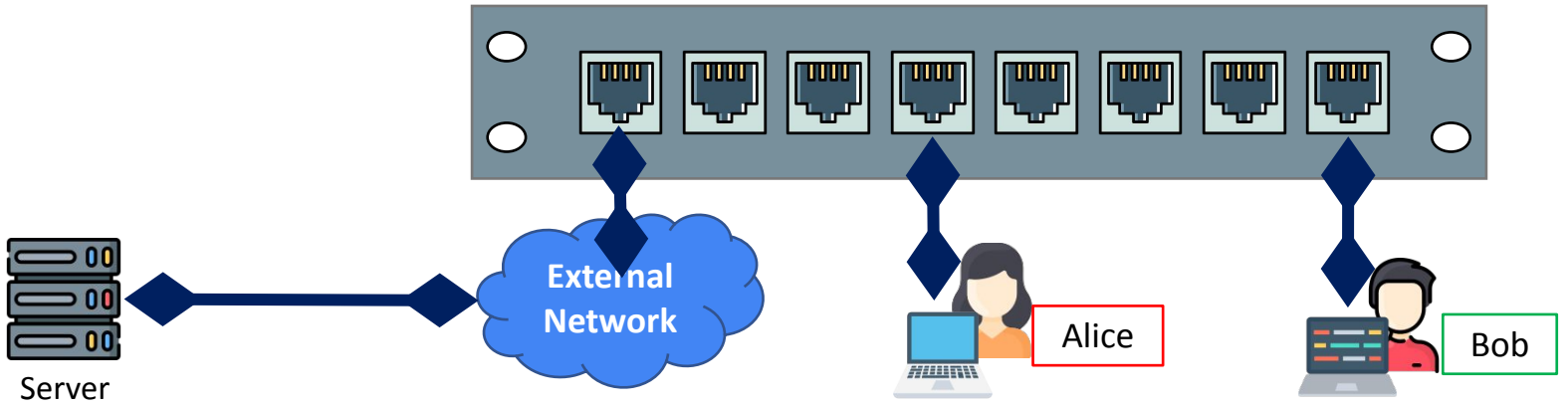
Registers are used maintain states in the data plane

Similar to arrays in programming languages like C/C++, Java etc.

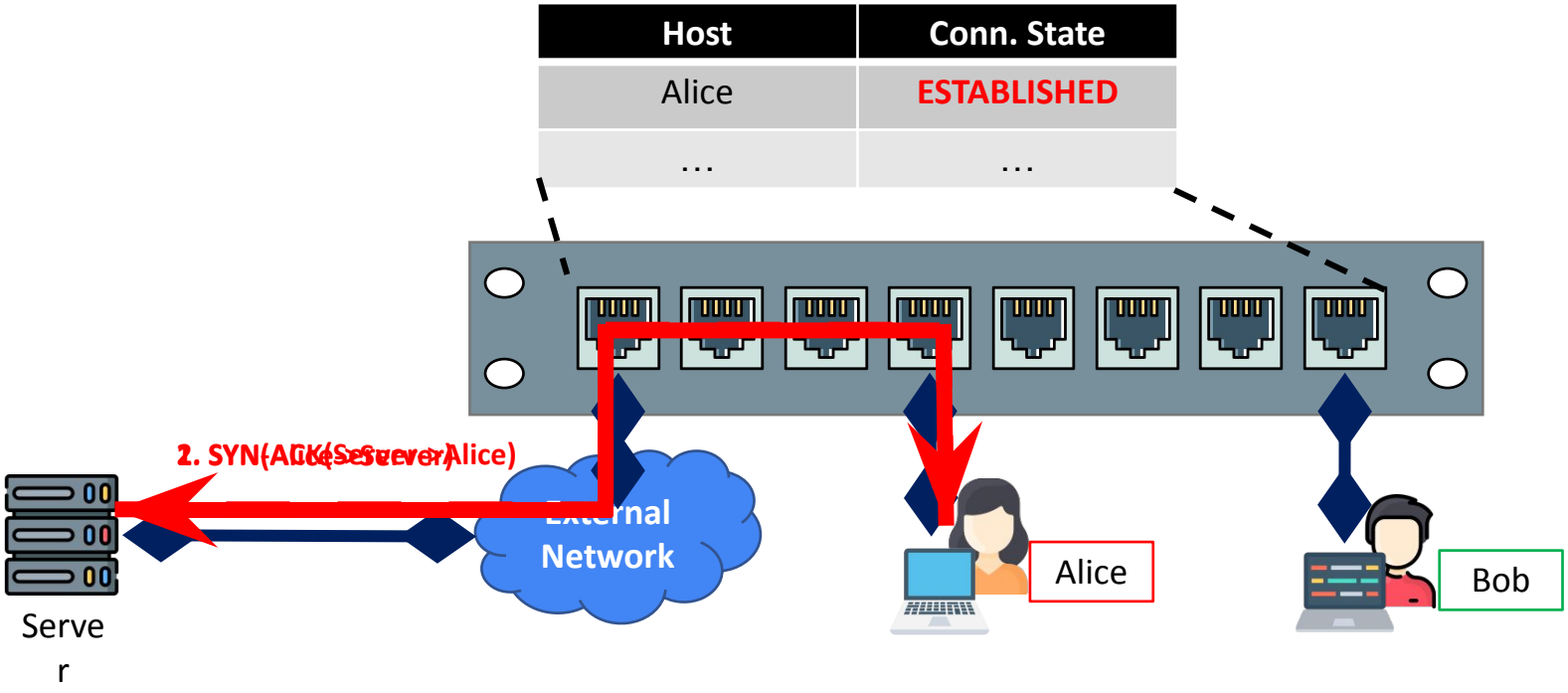
```
#define NUM_ENTRIES 1024
#define BIT_WIDTH 32
…
register<bit<BIT_WIDTH>>(1024) my_register;    // register definition
…
bit<32> index = hash(...);
bit<32> value;
my_register.read(value, index);               // register read action
…
value = value + 1;
…
my_register.write(index, value);              // register write action
```

Example: Stateful Firewall

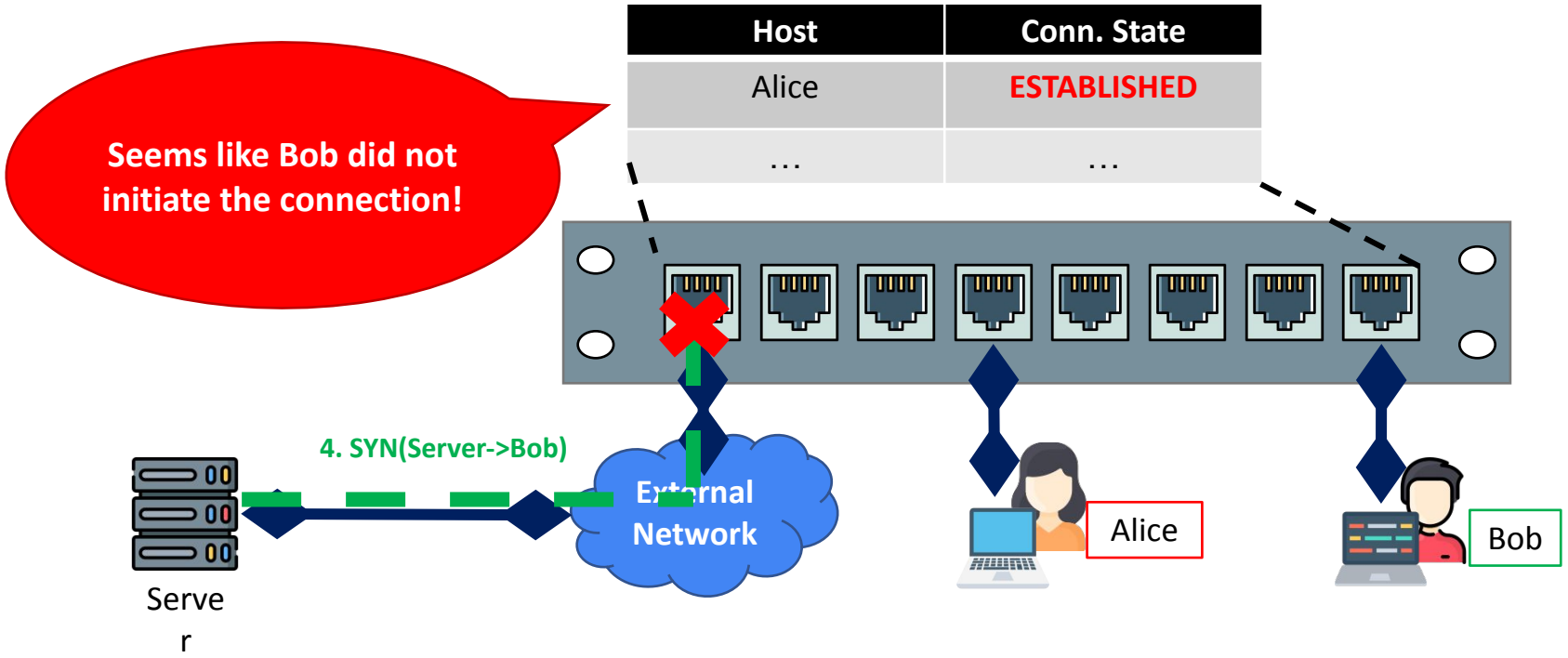
- Assume a network admin:
  - **Allows** connections to be initiated only from/ within an enterprise/home network
  - **Blocks** incoming connection requests from outside
- Typically used in stateful firewalls



Example: Stateful Firewall



Example: Stateful Firewall



Bloom Filters

Wikipedia:
"A Bloom filter is a space-efficient probabilistic data structure, conceived by Burton Howard Bloom in 1970, that is **used to test whether an element is a member of a set.**"

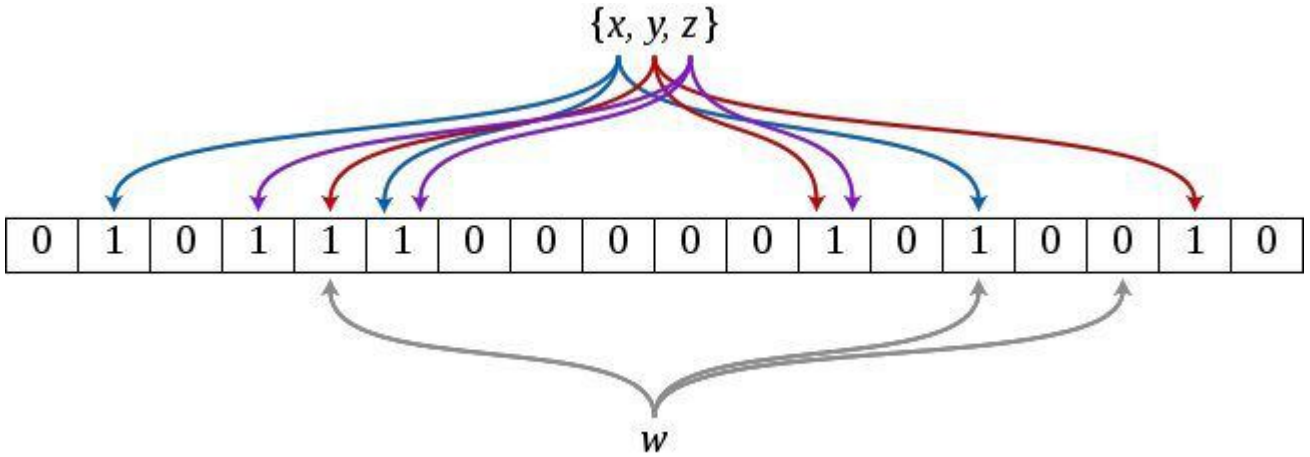


Image source: https://www.kdnuggets.com/2016/08/gentle-introduction-bloom-filter.html

Bloom Filters

- Stores information about a set of elements.
- Supports two operations:
  1. **add(x)** - adds x to bloom filter
  2. **contains(x)** - returns true if x in bloom filter, otherwise returns false
     a. If return false, **definitely** not in bloom filter.
     b. If return true, **possibly** in the structure (some false positives).

Slides from: https://courses.cs.washington.edu/courses/cse312/20au/files/slides/10-23-annotated.pdf

Meters in V1 Model Architecture

Definition

```
/* Definition in v1model.p4 */

enum MeterType {
    packets,
    bytes
}

extern meter {
    meter(bit<32> instance_count, MeterType type);
    void execute_meter<T>(in bit<32> index, out T result);
}
```

This is a template definition. The method will accept the parameter of any type

Color Coding:

0 – Green
1 – Yellow
2 -- Red

Usage

```
typedef bit<2> meter_color_t;

const meter_color_t METER_COLOR_GREEN  = 0;
const meter_color_t METER_COLOR_YELLOW = 1;
const meter_color_t METER_COLOR_RED    = 2;

meter(1024, MeterType.bytes) acl_meter;

action color_my_packets(bit<10> index) {
    acl_meter.execute_meter((bit<32>)index, meta.color);
}

table acl {
    key = { . . .}
    actions = { color_my_packets; . . .}
}

apply {
    acl.apply();
    if (meta.color == METER_COLOR_RED) {
        mark_to_drop();
    }
}
```

Counters in V1 Model Architecture

```
/* Definition in v1model.p4 */

enum CounterType {
    packets,
    bytes,
    packets_and_bytes
}

/* An array of counters of a given type */
extern counter {
    counter(bit<32> instance_count, CounterType type);
    void count(in bit<32> index);
}
```

- **Extern definition contains**
  - The instantiation method
    - Has the same name as the extern
    - Is evaluated at compile-time
  - Methods to access the extern
    - Very similar to actions
    - Can return values too
- **Enums in $P4_{16}$**
  - Abstract values
    - No specific (numerical representation)

Counters (V1Model)

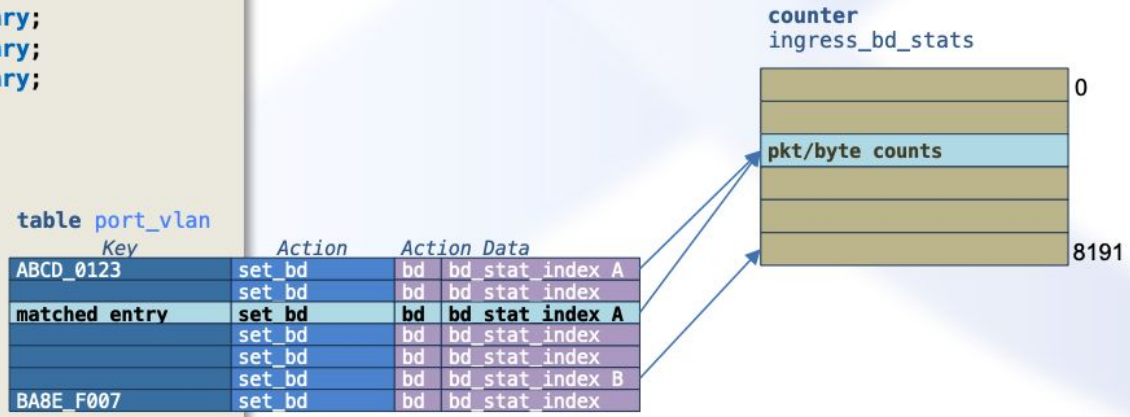
Counters can be updated from the P4 program, but can only be read from the control plane.

```
#define NUM_ENTRIES 1024
#define BIT_WIDTH 32
…
counter(NUM_ENTRIES, CounterType.packets) my_pkt_counts;
…
bit<32> index = hash(...);
…
my_pkt_counts.count((bit<32>) index);
…
```

Using the V1 Model Counters

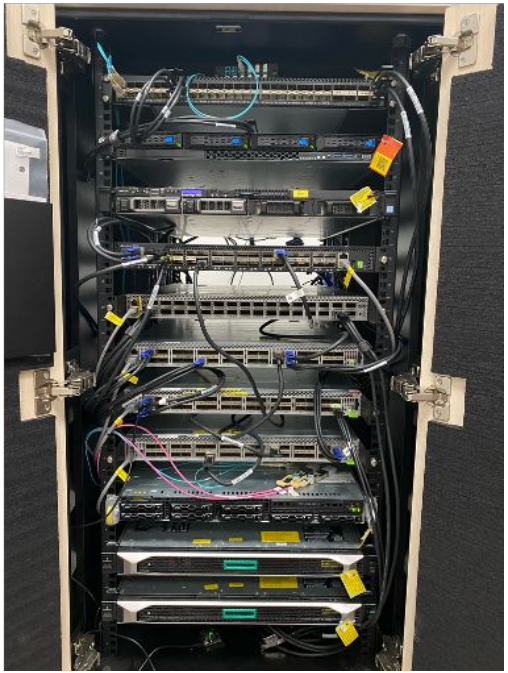
```
control MyIngress(inout my_headers_t        hdr,
                  inout my_metadata_t       meta,
                  inout standard_metadata_t standard_metadata)
{
   counter(8192, CounterType.packets_and_bytes) ingress_bd_stats;

   action set_bd(bit<16> bd, bit<13> bd_stat_index) {
       meta.l2.bd = bd;
       ingress_bd_stats.count((bit<32>)bd_stat_index);
   }

   table port_vlan {
       key = {
           standard_metadata.ingress_port : ternary;
           hdr.vlan_tag[0].isValid()      : ternary;
           hdr.vlan_tag[0].vid            : ternary;
       }
       actions = { set_bd; mark_for_drop; }
       default_action = mark_for_drop();
    }

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

- **Instantiate an extern inside the control**
  - Call the instantiation method
    - Parameters must be known at compile-time
- **Use extern's methods in actions or directly**



Questions?

Interested to working with us?



Quiz 3

Question 1:

In the V1Model (see: https://github.com/p4lang/p4c/blob/main/p4include/v1model.p4), which of the following variables in the "standard_metadata" do we set to forward a packet to a particular egress port?

- **egress_spec**
- ingress_port
- egress_port

Question 2:

Referring to the "calc" exercise from the P4 Tutorials (see: https://github.com/p4lang/tutorials/tree/master/exercises/calc), if you intend to extend the existing program to add support for multiplication (following the existing program's convention), what should the OpCode be?

0x2a
0x78
0x2f

Question 3:

Which of the follow match-action table "match_kind" is not included in the core P4 language (see: https://github.com/p4lang/p4c/blob/main/p4include/core.p4)?

range
ternary
lpm
exact

Question 4:

The state "parse_ethernet" handles IPv4 and VLAN (802.1Q).

To transition into the "parse_vlan" state, what should be the value for "hdr.ethernet.etherType"?

0x88A8
0x86DD
0x0806
0x8100

Question 5:

Consider the first 4 packets (#1 to 4) from this PCAP (see: https://www.cloudshark.org/captures/74a6deb7aa4e or download the attached "sample.pcap"), which of them will be dropped by the following parser (ignore the checksums)?

```
state parse_payload {
    packet.extract(hdr.payload);
    transition select(hdr.payload.byte_1, hdr.payload.byte_2) {
        (0x48, 0x45) : accept;
        default : reject;
    }
}
```

- all 4 packets
- packets 1 and 3
- packets 1 and 2
- packets 1 and 4
- **packets 2 and 4**
- packets 1 and 3
- packets 3 and 4
- packets 2 and 3
- No Answer

Question 6:

We install the following three table entries into the table:

1. key=10.0.0.1, action=drop()

2. key=10.0.0.2, action=NoAction();

3. key=10.0.0.3, action=ipv4_forward(1);

If a switch P4 program (unconditionally) applies the above table, and it receives an IPv4 packet from 10.0.0.1 destined towards 10.0.0.4, will there be a table **hit** (if yes - True, if no - False)?

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

True	14 respondents
False	85 respondents

References

1. ONF P4 Tutorial: https://opennetworking.org/wp-content/uploads/2020/12/P4_tutorial_01_basics.gslide.pdf
2. ONF ONS Tutorial: https://events19.linuxfoundation.org/wp-content/uploads/2017/12/Tutorial-P4-and-P4Runtime-Technical-Introduction-and-Use-Cases-for-Service-Providers-Carmelo-Cascone-Open-Networking-Foundation.pdf
3. Standford CS344 Lecture 2: https://cs344-stanford.github.io/lectures/Lecture-2-P4-tutorial.pdf
4. P4 Introduction: https://conferences.sigcomm.org/sigcomm/2018/files/slides/hda/paper_2.2.pdf
5. ETH Zurich Adv Topics in Communication Networks: https://polybox.ethz.ch/index.php/s/dP7zuZH5Y9amcGG

NetCache: Balancing Key-Value Stores with Fast In-Network Caching

Xin Jin[1], Xiaozhou Li[2], Haoyu Zhang[3], Robert Soule[2,4],
Jeongkeun Lee[2], Nate Foster[2,5], Changhoon Kim[2], Ion Stoica[6]

[1]John Hopkins University, [2]Barefoot Networks, [3]Princeton University,
[4]Università della Svizzera italiana, [5]Cornell University, [6]UC Berkeley

Slides adapted from: https://opennetworking.org/wp-content/uploads/2020/12/p4-ws-2017-netcache.pdf

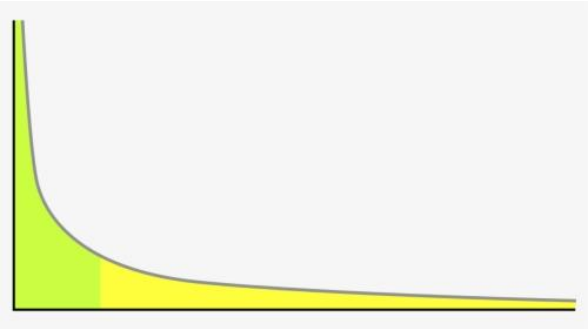
Goal: Fast and cost-efficient rack-scale KV stores

- Store, retrieve, manage key-value objects
  - Critical building block for large-scale cloud services

- Target workloads
  - Small objects
  - Read intensive
  - **Highly skewed and dynamic key popularity**

Key challenge:
Highly skewed and rapidly changing workloads

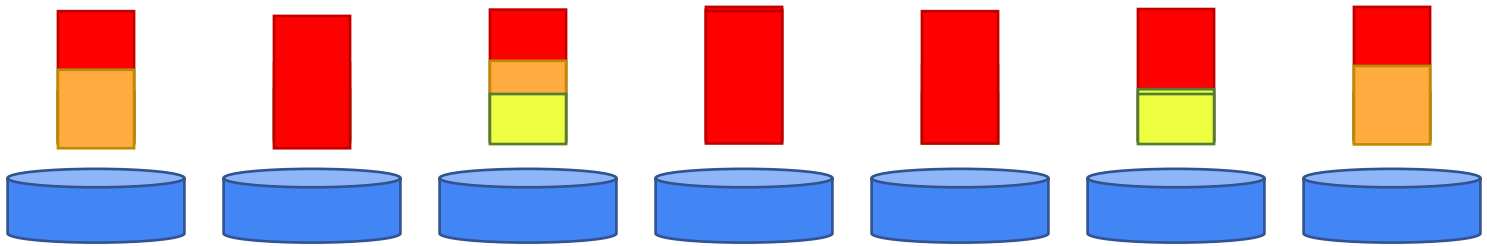
10% of items account for ~60-90% of queries[1]



Long tail
distribution

[1] Atikoglu et al., Workload Analysis of a Large-scale Key-value Store. 2012. ACM SIGMETRICS.

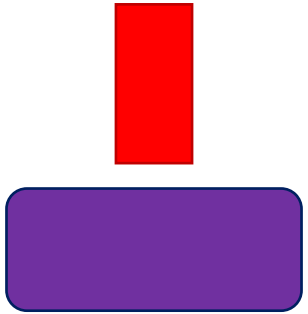
Key challenge:
Highly skewed and **rapidly changing** workloads



Q: How to provide effective dynamic load balancing?

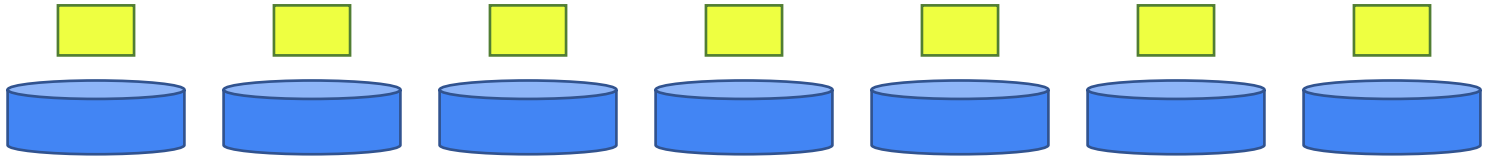
Opportunity:
Fast, **small** cache can ensure load balancing

Cache absorbs hottest queries



Cache ***O(N log N)*** hottest items[1]
N = # of servers

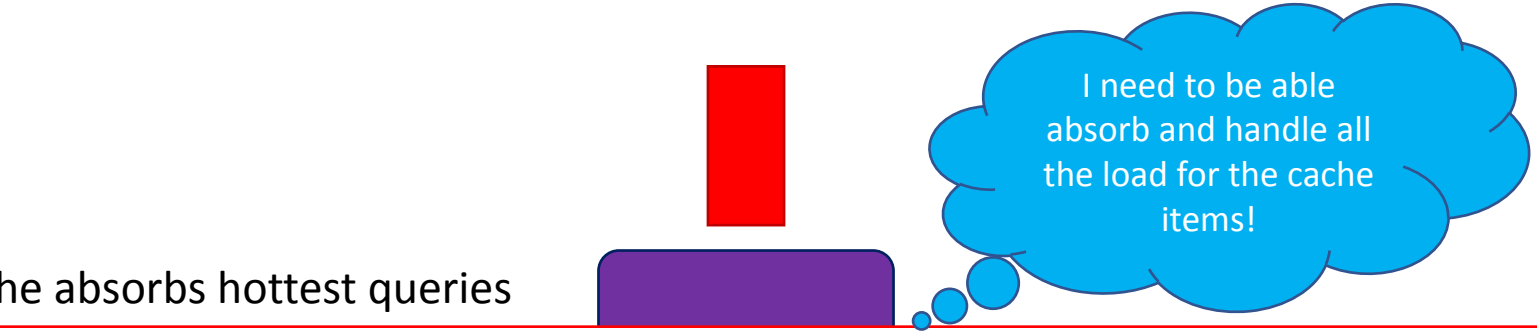
E.g., 10,000 hot objects for
100 servers with 100B items



Balanced load

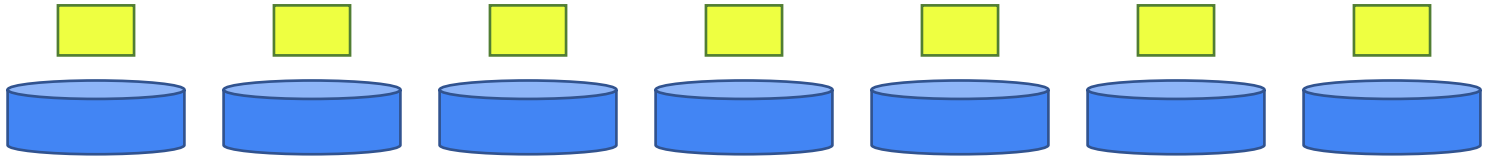
[1] Fan et al., Small Cache, Big Effect: Provable Load Balancing for Randomly Partitioned Cluster Services. 2011. ACM SOCC.

Opportunity:
Fast, **small** cache can ensure load balancing



Cache absorbs hottest queries

Requirement: cache throughput ≥ backend aggregate throughput

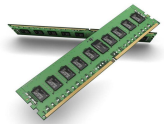


Balanced load

NetCache: Towards billions QPS KV store racks

Cache needs to provide the aggregate throughput of the storage layer

storage layer



In-memory

Each: O(10) MQPS

Total: O(1) BQPS

cache

cache layer

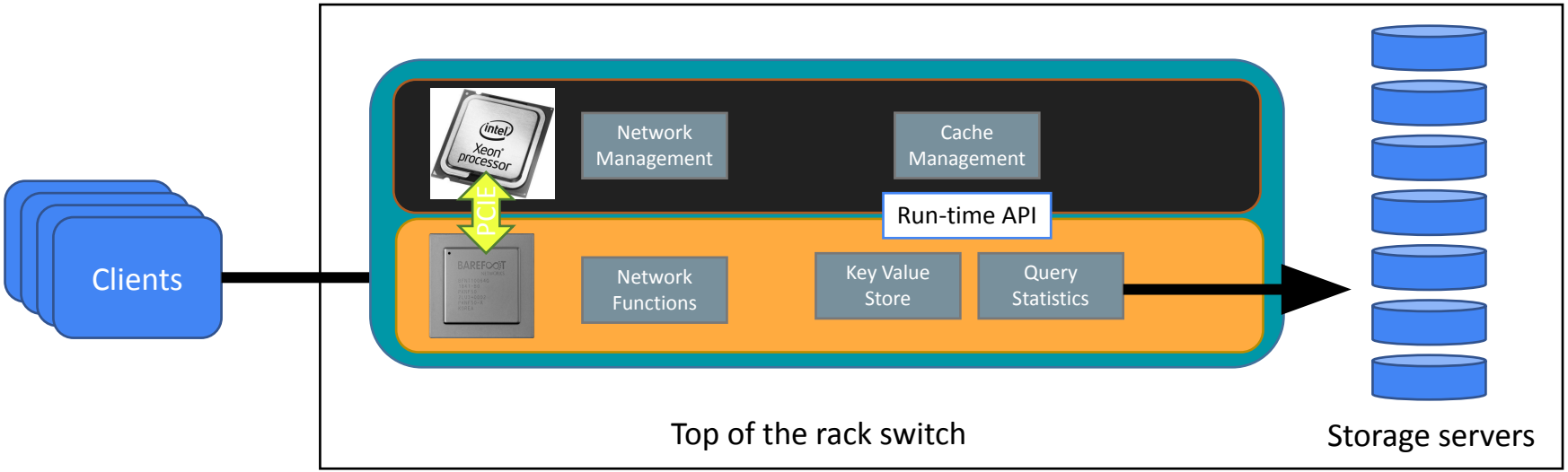
In-network

O(1) BQPS

Small on-chip memory? 10s of MB only.

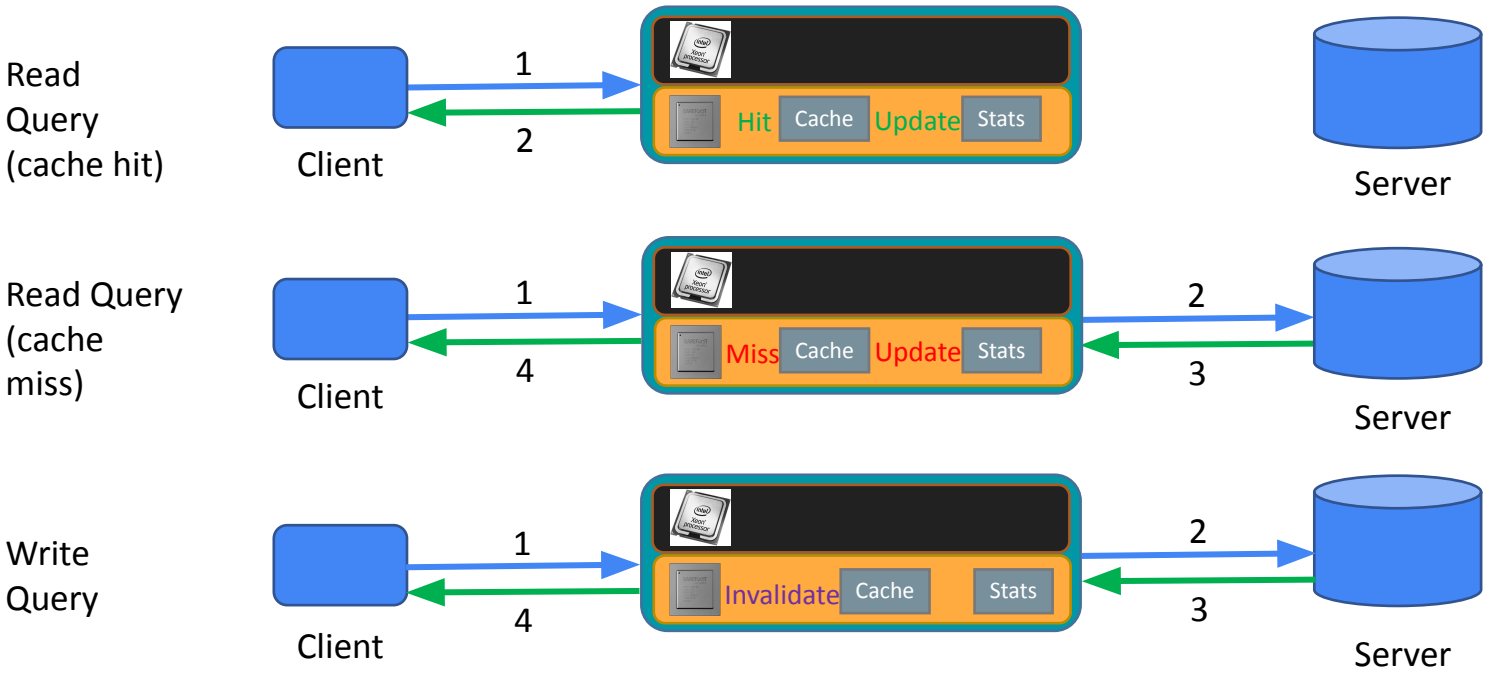
Only cache **O(N log N)** **<u>small</u>** items!

NetCache Rack-scale architecture

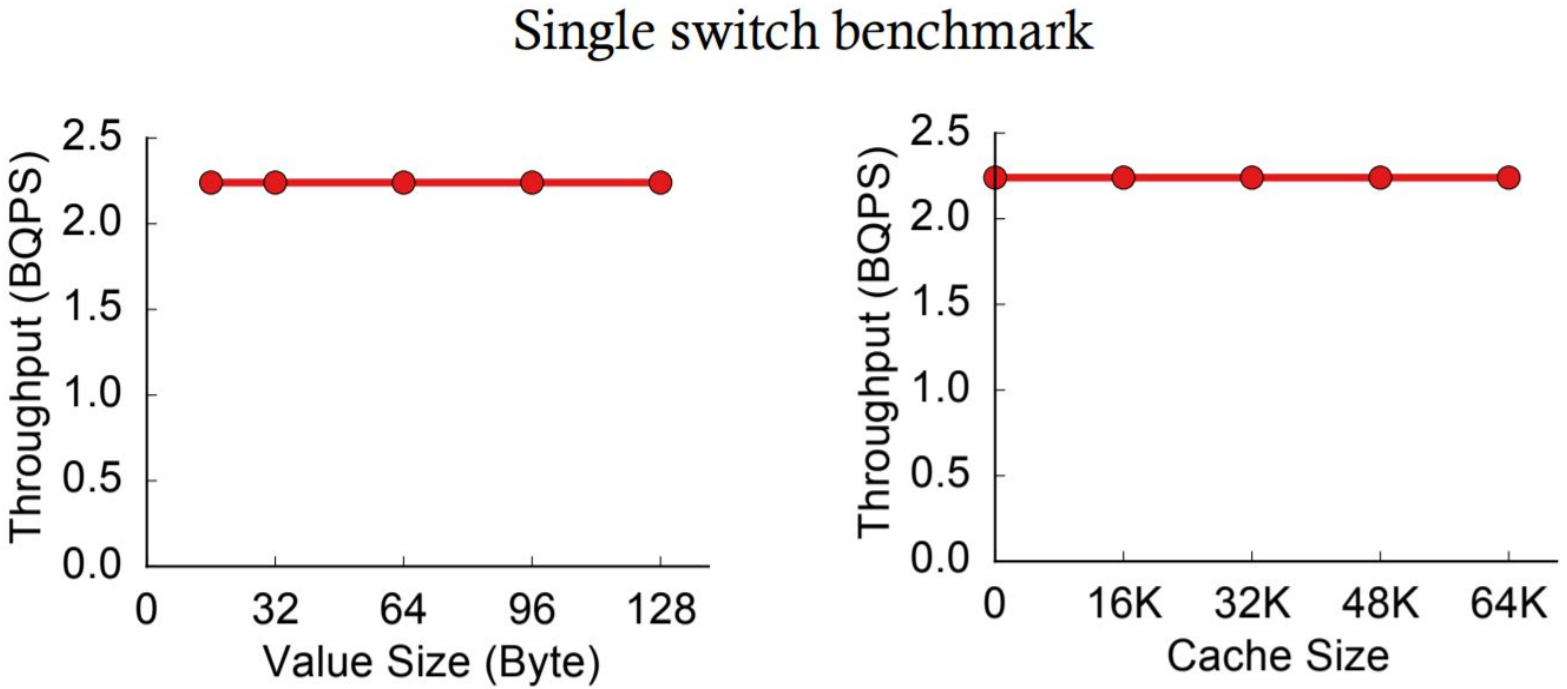


- Switch Data Plane
  - Key-value store to serve queries for cached keys
  - Query statistics to enable efficient cache updates
- Switch Control Plane
  - Insert hot items into the cache and evict less popular items
  - Manage memory allocation for on-chip key-value store

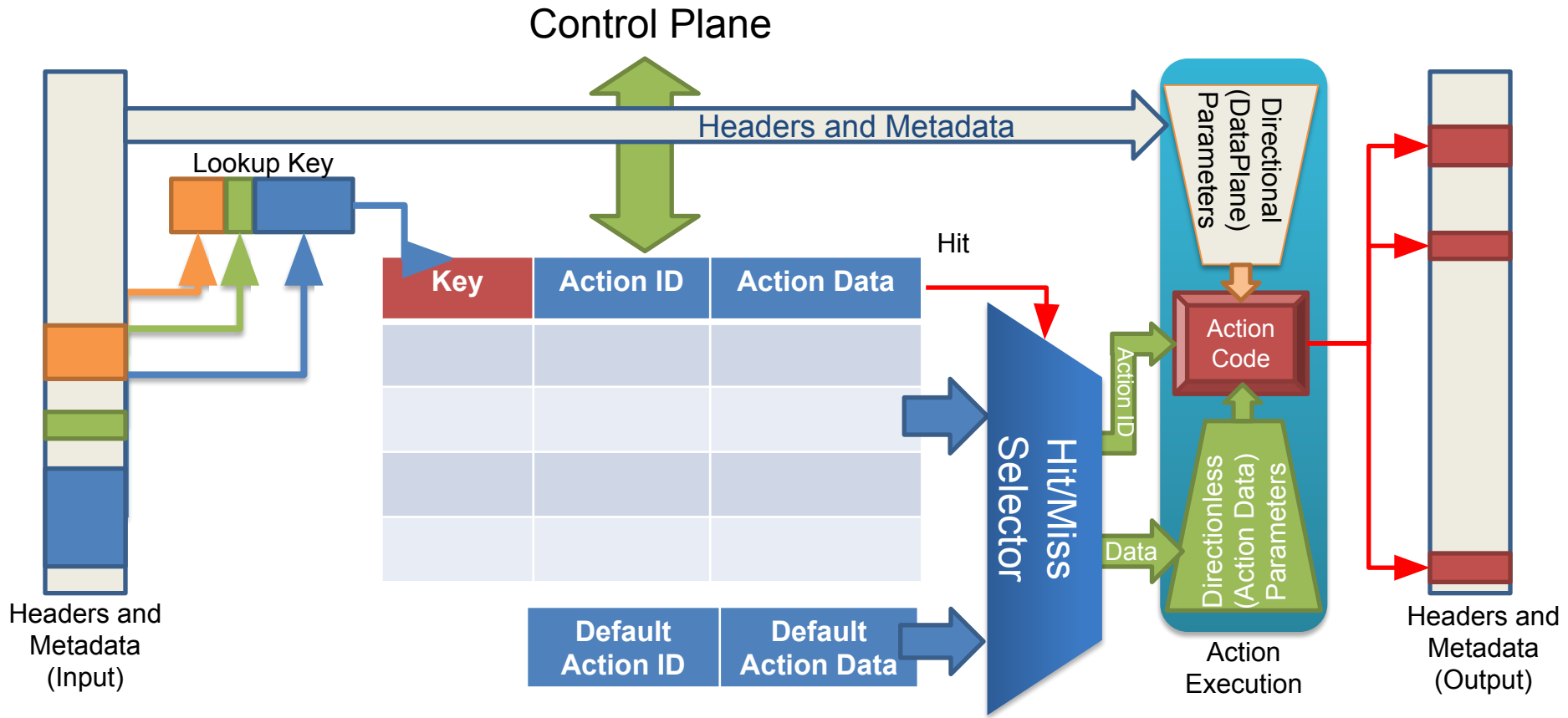
Data Plane Query Handling



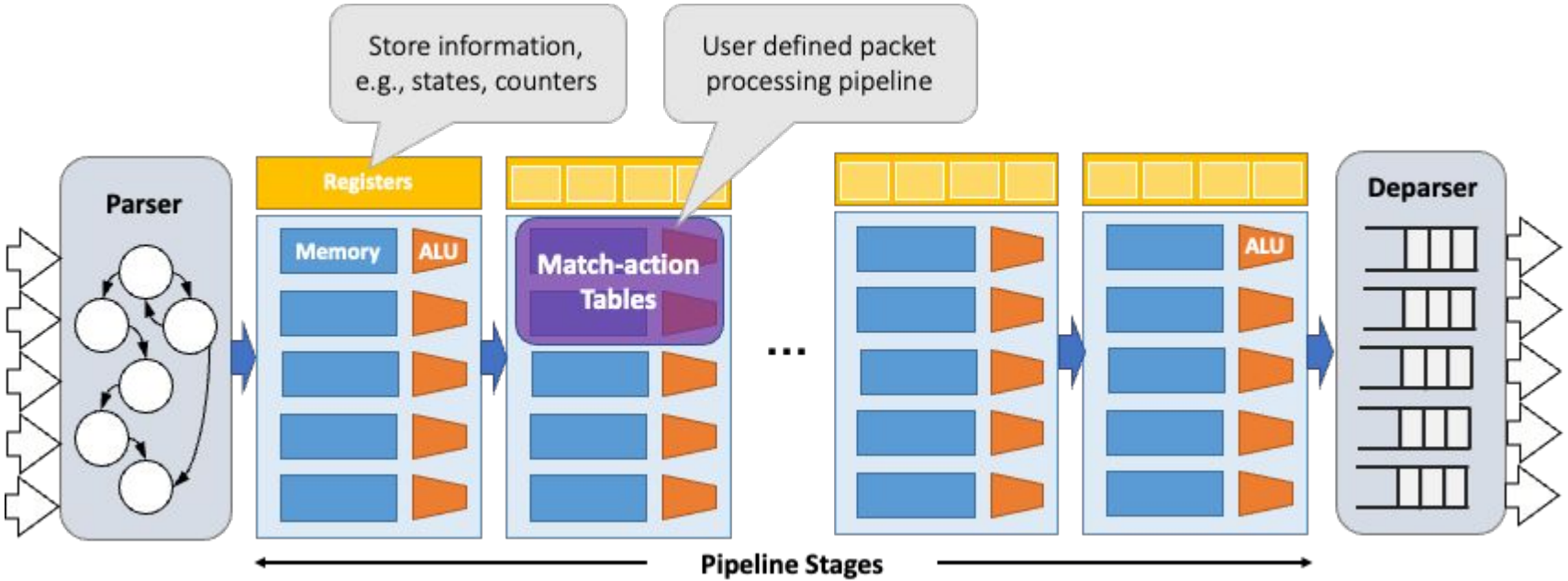
The “boring life” of a NetCache switch



Backup Slides


Control Plane
Headers and Metadata
Lookup Key
Key
Action ID
Action Data
Hit
Default Action ID
Default Action Data
Hit/Miss Selector
Action ID
Data
Directional (DataPlane) Parameters
Action Code
Directionless (Action Data) Parameters
Action Execution
Headers and Metadata (Input)
Headers and Metadata (Output)

Recap


Store information, e.g., states, counters
User defined packet processing pipeline
Parser
Registers
Memory
ALU
Match-action Tables
...
ALU
Deparser
Pipeline Stages

Bloom Filters: Example

bloom filter t with m = 5 that uses k = 3 hash functions

```
function INITIALIZE(k,m)
    for i = 1, . . . , k: do
        t_i = new bit vector of m 0's
```

Index →	0	1	2	3	4
t_1	0	0	0	0	0
t_2	0	0	0	0	0
t_3	0	0	0	0	0

Slides from: https://courses.cs.washington.edu/courses/cse312/20au/files/slides/10-23-annotated.pdf

Bloom Filters: Example

bloom filter t of length m = 5 that uses k = 3 hash functions

function ADD(x)
  for $i = 1, \ldots, k$: **do**
    $t_i[h_i(x)] = 1$

add("thisisavirus.com")

h_1("thisisavirus.com") → 2

h_2("thisisavirus.com") → 1

h_3("thisisavirus.com") → 4

Index →	0	1	2	3	4
t_1	0	0	**1**	0	0
t_2	0	**1**	0	0	0
t_3	0	0	0	0	**1**

Slides from: https://courses.cs.washington.edu/courses/cse312/20au/files/slides/10-23-annotated.pdf

Bloom Filters: Example

bloom filter t of length m = 5 that uses k = 3 hash functions

contains(“thisisavirus.com”)

h_1(“thisisavirus.com”) → 2

h_2(“thisisavirus.com”) → 1

h_3(“thisisavirus.com”) → 4

function CONTAINS(x)
return $t_1[h_1(x)] == 1 \wedge t_2[h_2(x)] == 1 \wedge \cdots \wedge t_k[h_k(x)] == 1$

True True True

Index →	0	1	2	3	4
t_1	0	0	1	0	0
t_2	0	1	0	0	0
t_3	0	0	0	0	1

Slides from: https://courses.cs.washington.edu/courses/cse312/20au/files/slides/10-23-annotated.pdf

Bloom Filters: Example

bloom filter t of length m = 5 that uses k = 3 hash functions

contains("thisisavirus.com")

h_1("thisisavirus.com") → 2

h_2("thisisavirus.com") → 1

h_3("thisisavirus.com") → 4

```
function CONTAINS(x)
    return t1[h1(x)] == 1 ∧ t2[h2(x)] == 1 ∧ ··· ∧ tk[hk(x)] == 1
```

True **True** **True**

Since all conditions satisfied, returns True (correctly)

Index →	0	1	2	3	4
t_1	0	0	**1**	0	0
t_2	0	**1**	0	0	0
t_3	0	0	0	0	**1**

Slides from: https://courses.cs.washington.edu/courses/cse312/20au/files/slides/10-23-annotated.pdf

Bloom Filters: Example

bloom filter t of length m = 5 that uses k = 3 hash functions

contains(“verynormalsite.com”)

h_1(“verynormalsite.com”) → 2

h_2(“verynormalsite.com”) → 0

h_3(“verynormalsite.com”) → 4

function CONTAINS(x)
return $t_1[h_1(x)] == 1 \wedge t_2[h_2(x)] == 1 \wedge \cdots \wedge t_k[h_k(x)] == 1$

True **True** **True**

Since all conditions satisfied, returns True (incorrectly)

Index →	0	1	2	3	4
t_1	0	1	**1**	0	0
t_2	**1**	1	0	0	0
t_3	0	0	0	0	**1**

Slides from: https://courses.cs.washington.edu/courses/cse312/20au/files/slides/10-23-annotated.pdf

Bloom Filters: Summary

- An empty bloom filter is an empty k x m bit array with all values initialized to zeros
  - k = number of hash functions
  - m = size of each array in the bloom filter
- add(x) runs in O(k) time
- contains(x) runs in O(k) time
- requires O(km) space (in bits!)
- Probability of false positives from collisions can be reduced by increasing the size of the bloom filter

Slides from: https://courses.cs.washington.edu/courses/cse312/20au/files/slides/10-23-annotated.pdf