High-Speed FPGA Packet Processing using the new P4 Programming Language

FPGA 2018

Presenters

- **Gordon Brebner, Xilinx Labs**
- **Stephen Ibanez, Stanford University**

Agenda

- **Network data plane programming with P4**
  - What is data plane programming?
  - P4: the data plane programming language

- ***Break***

- **Standard P4 design flow for software simulation target**
- **FPGA implementation of P4**
  - Xilinx P4-SDNet
  - P4→NetFPGA design flow

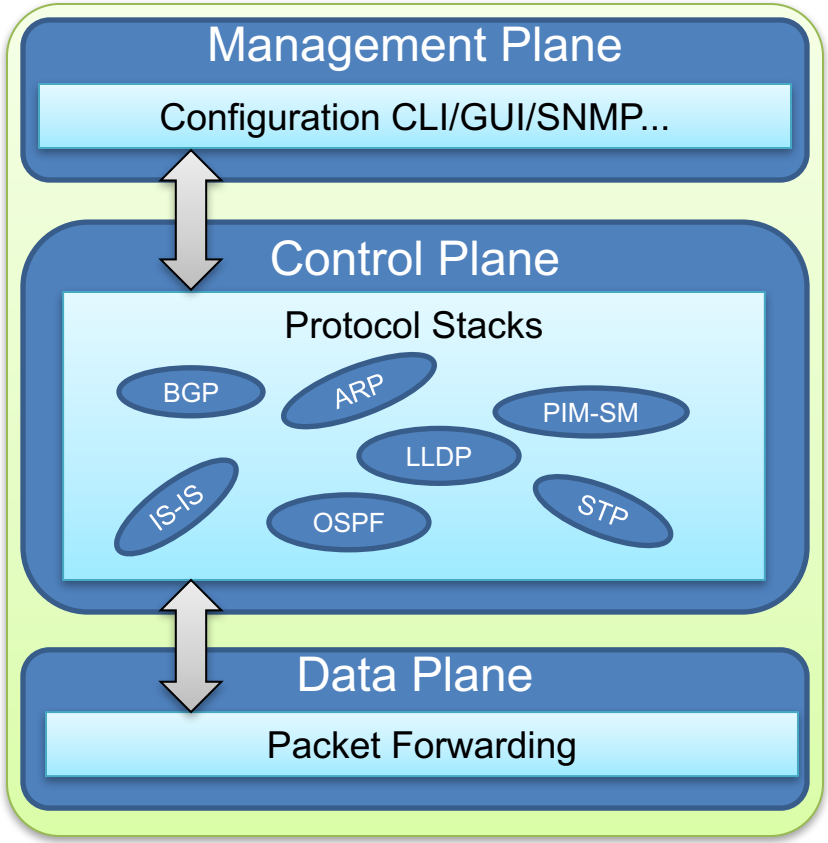
- **Future research directions**

What is Data Plane Programming?

- **Why program the Data Plane?**

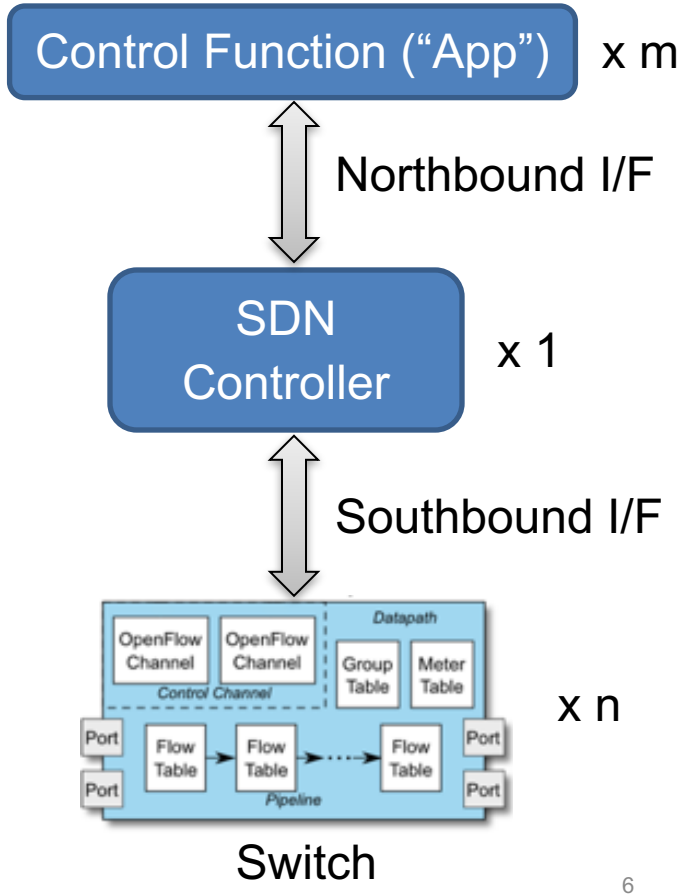
Standard Telecommunications Architecture

- **Traditional architecture consists of the three planes**
- **A Plane is a group of algorithms**
- **These algorithms**
  - Process different kinds of traffic
  - Have different performance requirements
  - Are designed using different methodologies
  - Are implemented using different programming languages
  - Run on different hardware



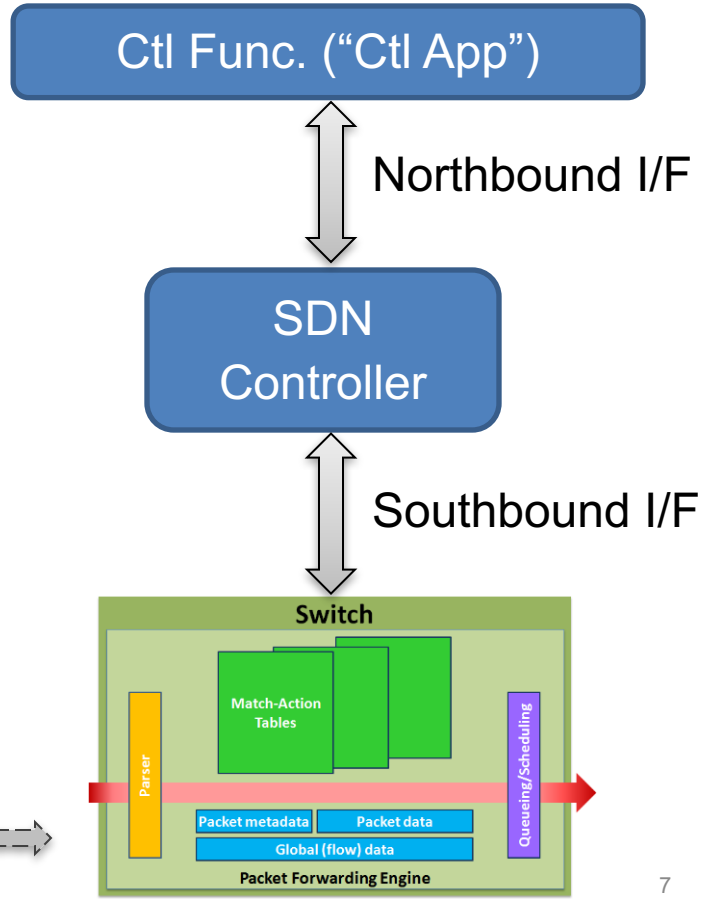
Software Defined Networking: Logically Centralized Control

- **Main contributions**
  - OpenFlow = standardized *model* (“architecture”) defining switch behavior (match / action)
  - OpenFlow = standardized *protocol* to interact with switch (download flow table entries, query statistics etc.)
  - *Concept* of *logically* centralized control via a single entity (“SDN controller”)
    - Simplifies control plane – e.g. compute optimal paths at one location (controller), vs. waiting for distributed routing algorithms to converge
- **Issues**
  - Limited interoperability between vendors => southbound I/F differences handled at controller (OpenFlow / netconf / JSON / XML variants)
  - Dataplane protocol evolution requires changes to standards (protocol and behavior)

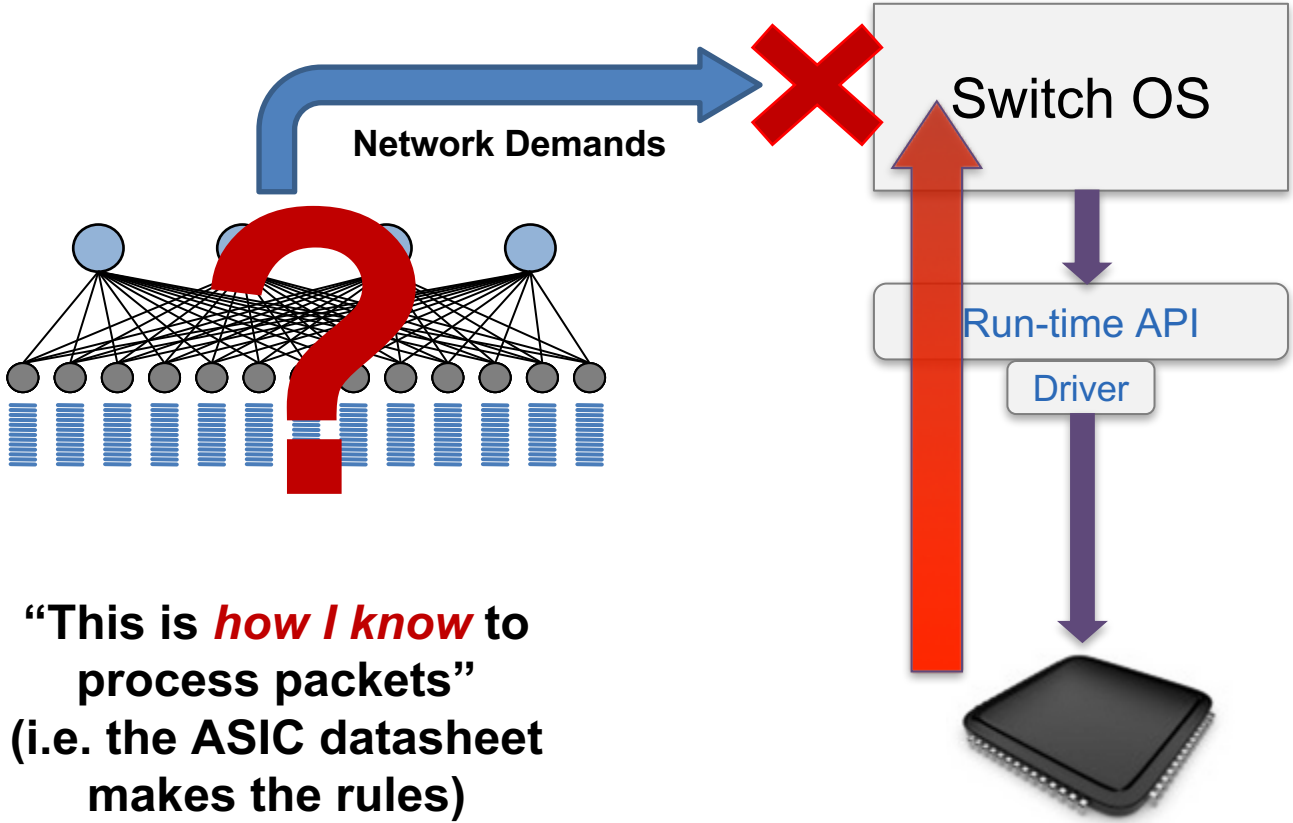


SDN Evolved: (Dataplane) Protocol Independence

- **Main contributions**
  - Programs explicitly define dataplane protocols (parse tree)
  - Programs explicitly specify datapath behavior (matching, actions, queueing, global state…)
- **Issues**
  - Enabling SDN controller (or its equivalent, e.g. OpenStack) to take advantage of new protocols / behaviors: more flexible NBI+SBI
  - Introduces new entities: DP program, DP programmer => new lifecycle issues arise

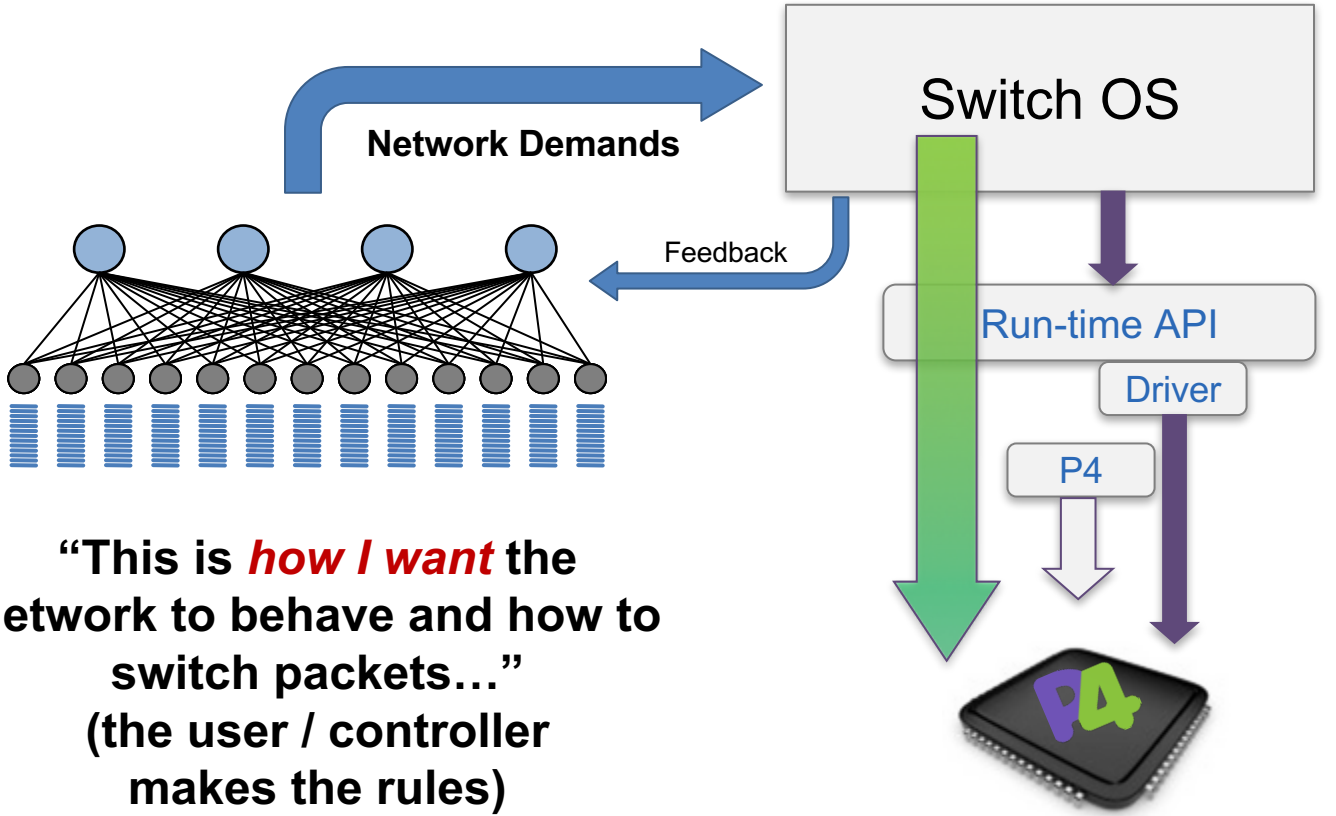


Status Quo: Bottom-up design



"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

- **C** • Control and Customization. Make the device behave exactly as you want
- **R** • Reliability. Reduce the risk by removing unused features
- **E** • Efficiency. Reduce energy consumption and expand scale by doing only what you need
- **A** • Add new features on your schedule
- **T** • Telemetry. Be able to see inside the Data Plane
- **E** • Exclusivity and Differentiation. No need to share your IP with the chip vendor

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, Intel, …

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]
- Fast In-Network cache for key-value stores – NetCache[3]
- In-band Network Telemetry – INT[4]
- Consensus at network speed – NetPaxos[5]
- … 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] Xin Jin et al. "NetCache: Balancing Key-Value Stores with Fast In-Network Caching." SOSP, 2017
[4] Kim, Changhoon, et al. "In-band network telemetry via programmable dataplanes." SIGCOMM, 2015.
[5] Dang, Huynh Tu, et al. "NetPaxos: Consensus at network speed." SIGCOMM, 2015.

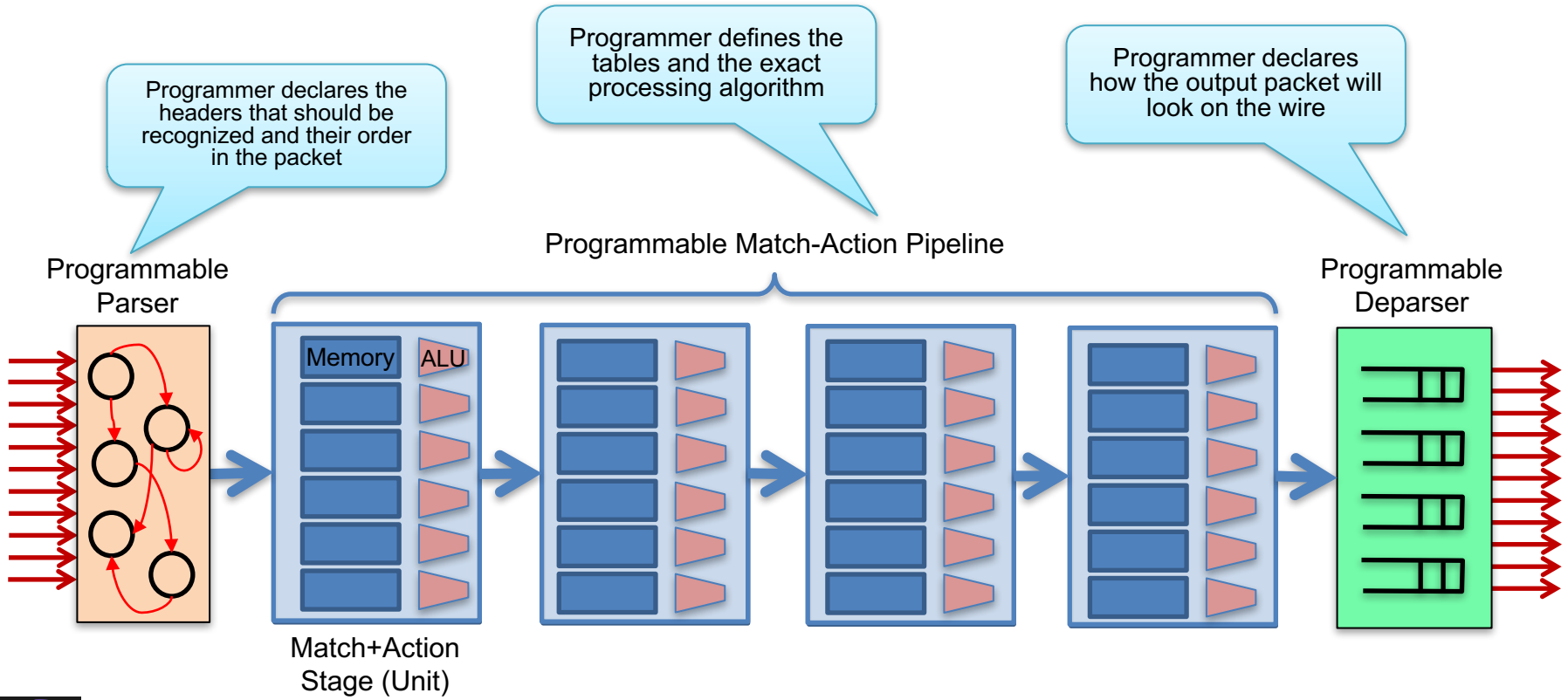
Brief History and Trivia

- May 2013: Initial idea and the name “P4”
- July 2014: First paper (SIGCOMM ACR)
- Aug 2014: First $P4_{14}$ Draft Specification (v0.9.8)
- Sep 2014: $P4_{14}$ Specification released (v1.0.0)
- Jan 2015: $P4_{14}$ v1.0.1
- Mar 2015: $P4_{14}$ v1.0.2
- Nov 2016: $P4_{14}$ v1.0.3
- May 2017: $P4_{14}$ v1.0.4

- Apr 2016: $P4_{16}$ – first commits
- Dec 2016: First $P4_{16}$ Draft Specification
- May 2017: $P4_{16}$ Specification released

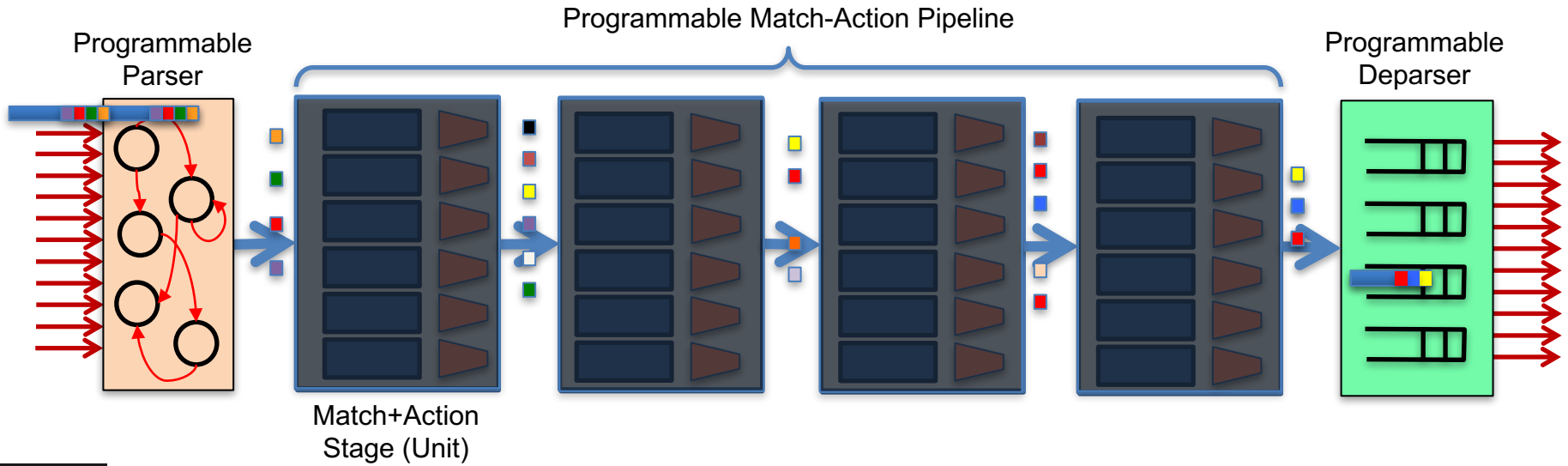
$P4_{16}$ Data Plane Model

PISA: Protocol-Independent Switch Architecture

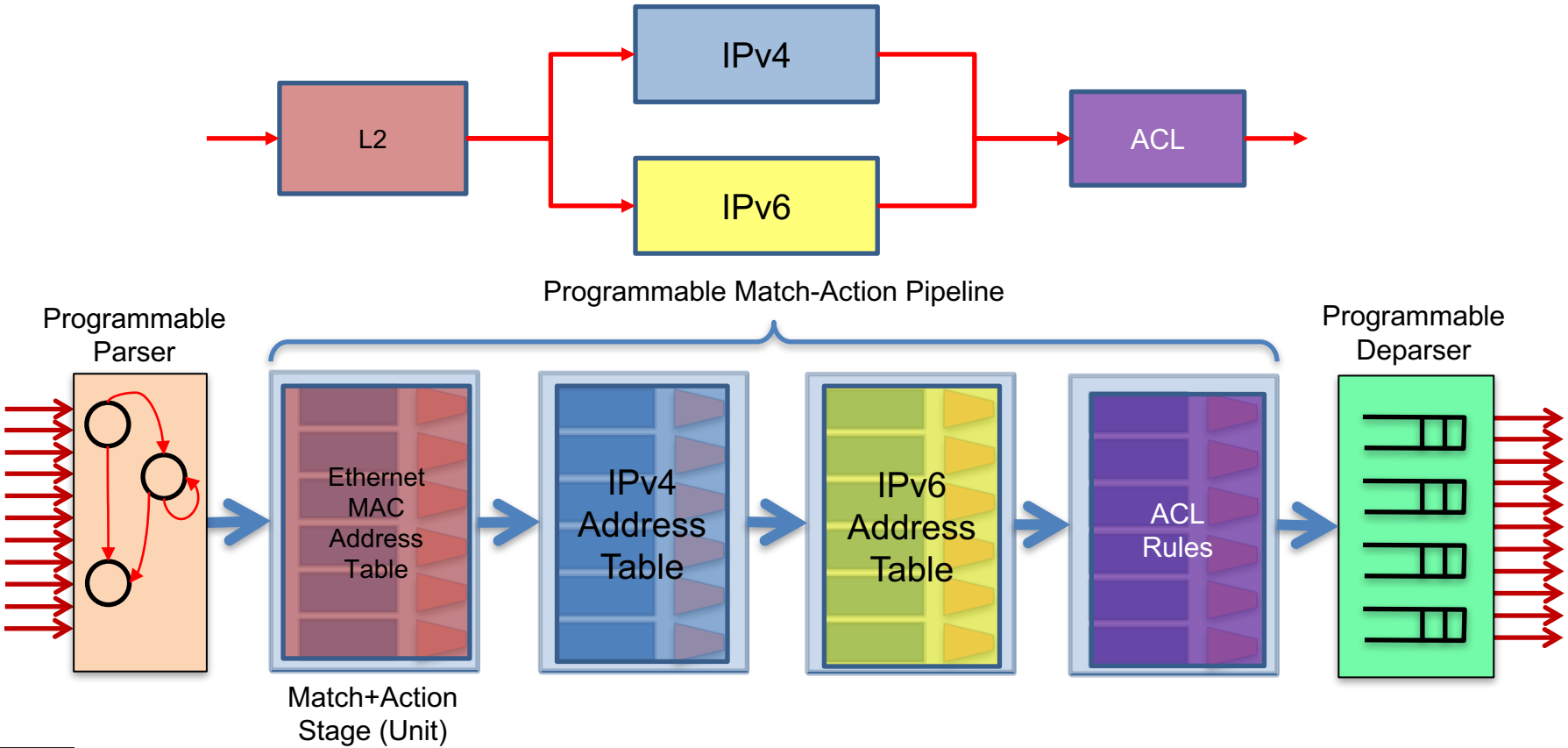


PISA in Action

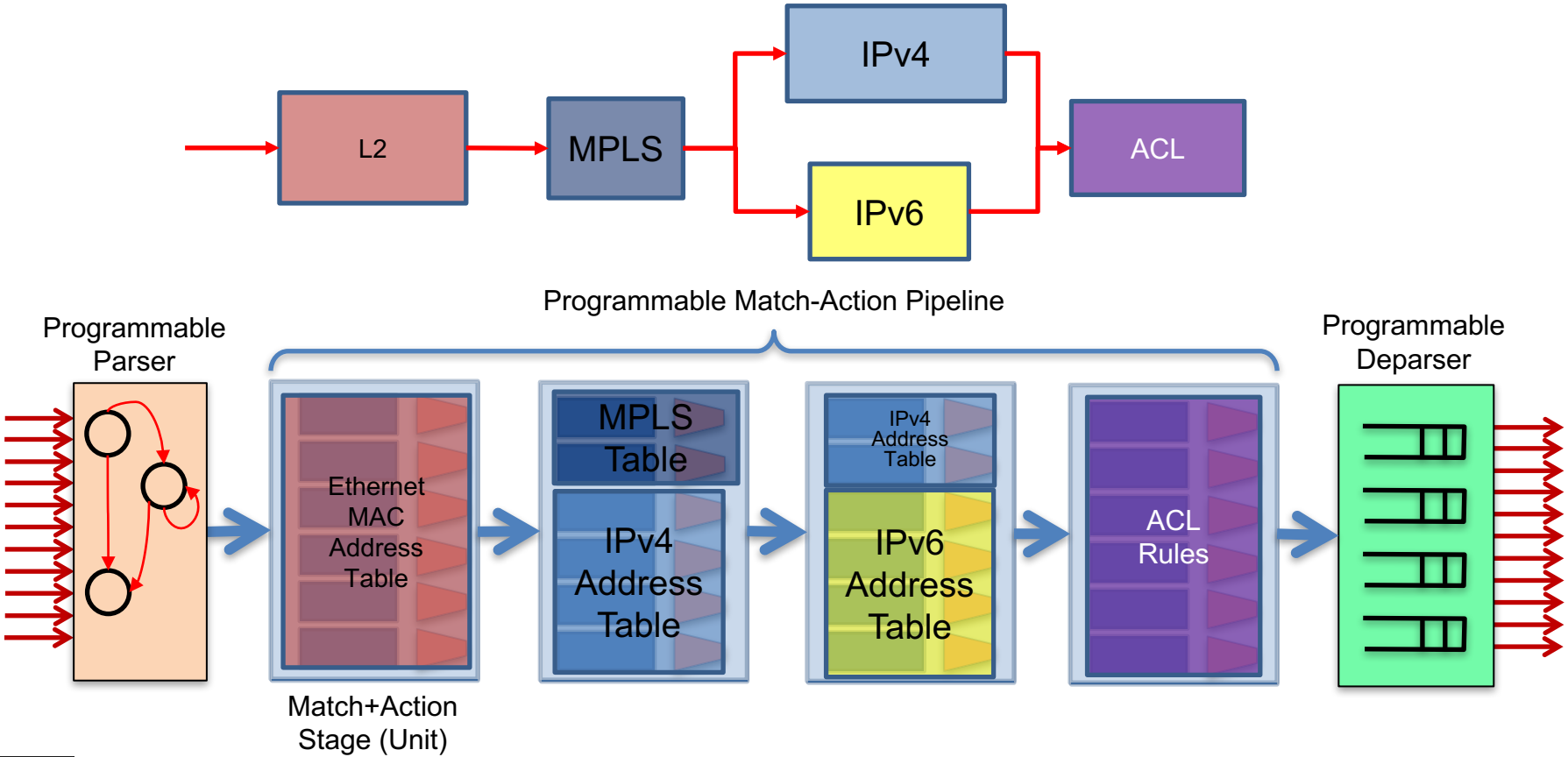
- Packet is parsed into individual headers (parsed representation)
- Headers and intermediate results can be used for matching and actions
- Headers can be modified, added or removed
- Packet is deparsed (serialized)



Mapping a Simple L3 Data Plane Program on PISA

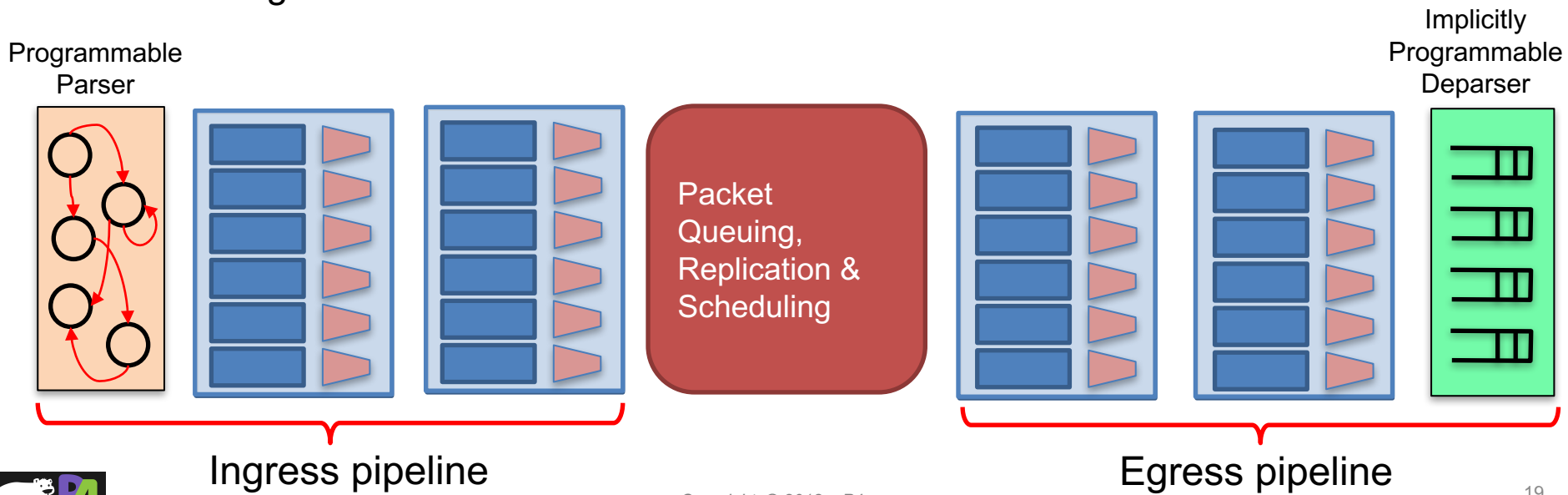


Mapping a More Complex Data Plane Program on PISA



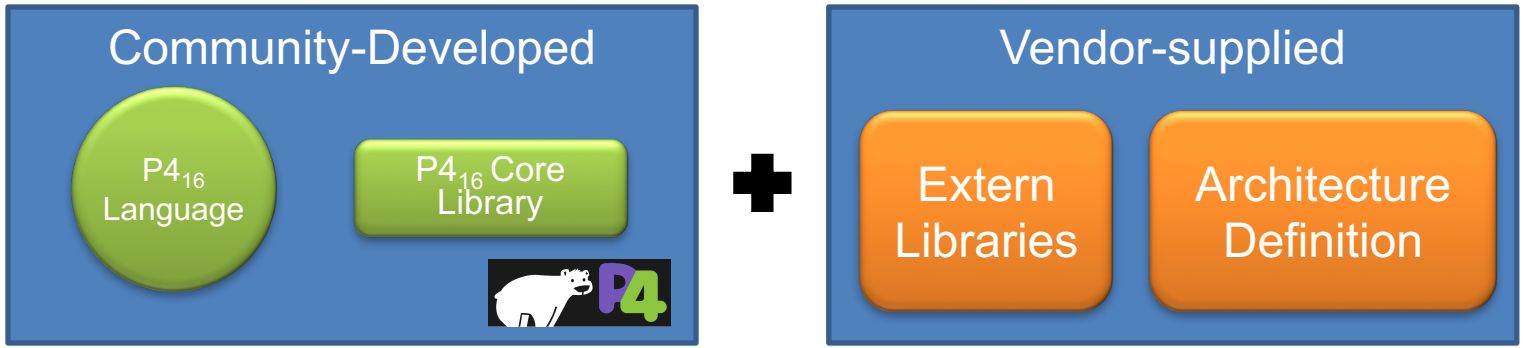
$P4_{14}$ Switch Model (V1 Architecture)

- **Ingress Pipeline**
- **Egress Pipeline**
- **Traffic Manager**
  - N:1 Relationships: Queueing, Congestion Control
  - 1:N Relationships: Replication
  - Scheduling

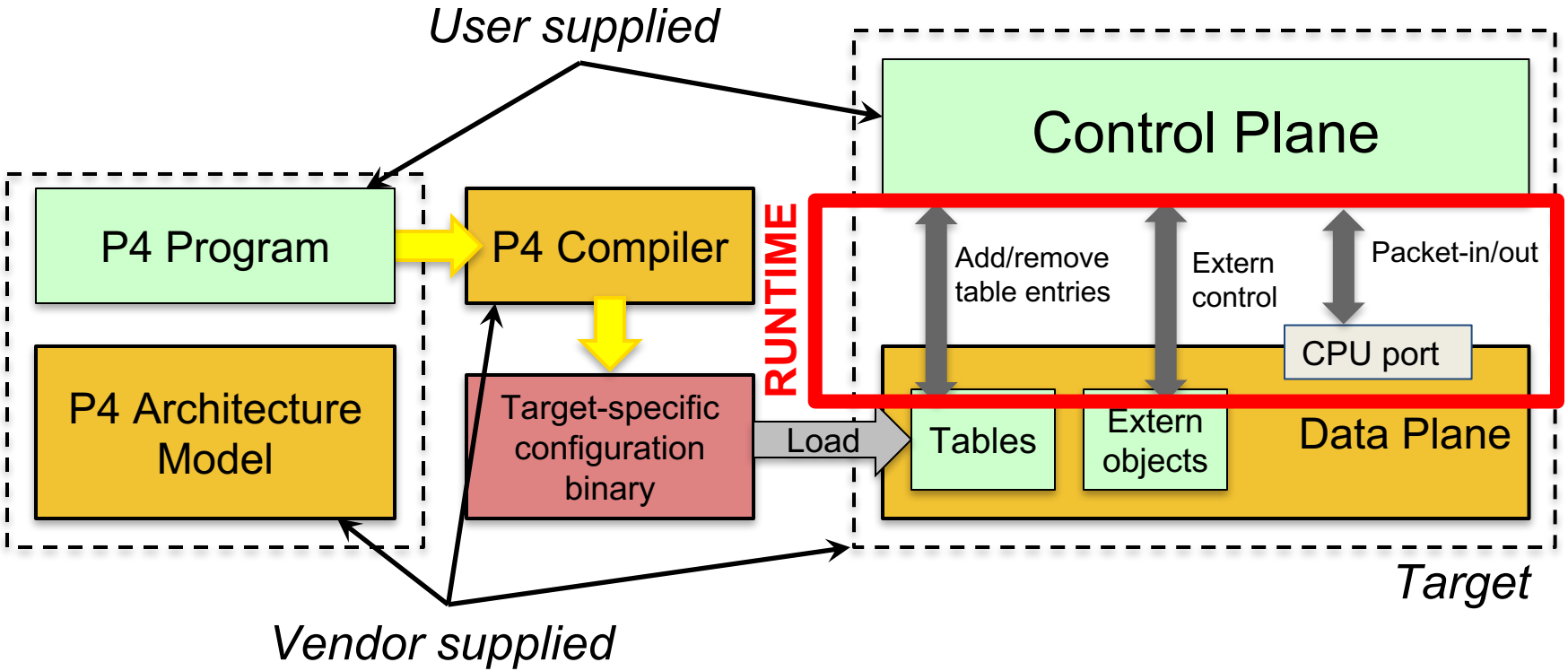


$P4_{16}$ Approach

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



Programming a P4 Target



$P4_{16}$ Basics

P4$_{16}$ Language Elements

Parsers

State machine, bitfield extraction

Controls

Tables, Actions, control flow statements

Expressions

Basic operations and operators

Data Types

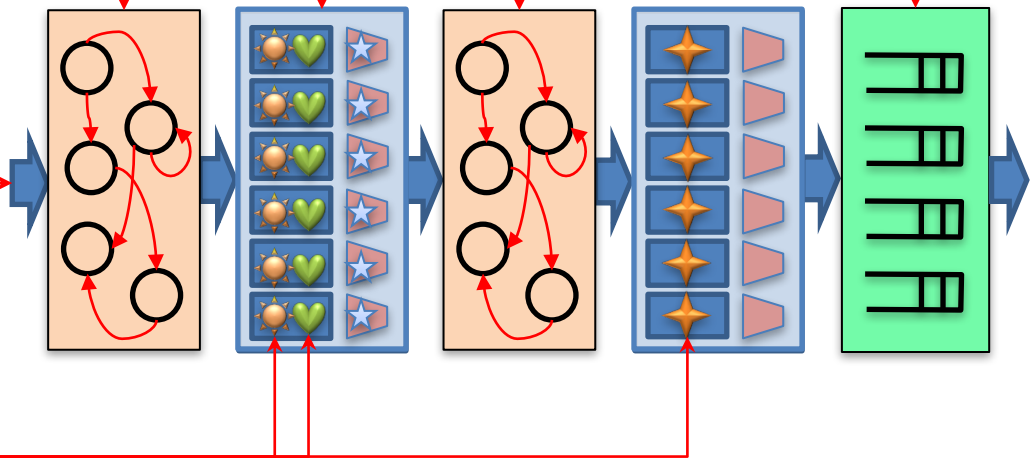
Bistrings, headers, structures, arrays

Architecture Description

Programmable blocks and their interfaces

Extern Libraries

Support for specialized components



V1Model Standard Metadata

```
struct standard_metadata_t {
   bit<9>  ingress_port;
   bit<9>  egress_spec;
   bit<9>  egress_port;
   bit<32> clone_spec;
   bit<32> instance_type;
   bit<1>  drop;
   bit<16> recirculate_port;
   bit<32> packet_length;
   bit<32> enq_timestamp;
   bit<19> enq_qdepth;
   bit<32> deq_timedelta;
   bit<19> deq_qdepth;
   bit<48> ingress_global_timestamp;
   bit<32> lf_field_list;
   bit<16> mcast_grp;
   bit<1>  resubmit_flag;
   bit<16> egress_rid;
   bit<1>  checksum_error;
}
```

- **`ingress_port`** - the port on which the packet arrived
- **`egress_spec`** - the port to which the packet should be sent to
- **`egress_port`** - the port on which the packet is departing from (read only in egress pipeline)

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

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

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

$P4_{16}$ Hello World (V1Model)

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

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

    state start { transition accept; }
}

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Key	Action Name	Action Data
1	set_egress_spec	2
2	set_egress_spec	1

$P4_{16}$ Data Types

- **Basic Data Types**
- **Derived Data Types**
  - Headers and metadata

Simple Header Definitions

Example: Declaring L2 headers

```
header ethernet_t {
    bit<48>     dstAddr;
    bit<48>     srcAddr;
    bit<16>     etherType;
}

header vlan_tag_t {
    bit<3>       pri;
    bit<1>       cfi;
    bit<12>      vid;
    bit<16>      etherType;
}

struct my_headers_t {
    ethernet_t    ethernet;
    vlan_tag_t[2] vlan_tag;
}
```

- **Basic Types**
  - **bit<n>** – Unsigned integer (bitsrting) of length n
    - **bit** is the same as **bit<1>**
  - **int<n>** – Signed integer of length n (>=2)
  - **varbit<n>** – Variable-length bitstring
- **Derived Types**
  - **header** – Ordered collection of members
    - Byte-aligned
    - Can be valid or invalid
    - Can contain bit<n>, int<n> and varbit<n>
  - **struct** – Unordered collection of members
    - No alignment restrictions
    - Can contain any basic or derived types
  - Header Stacks -- arrays of headers

Typedef

Example: Declaring a type for MAC address

```
typedef bit<48> mac_addr_t;

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

- **Basic Types**
  - **bit<n>** – Unsigned integer (bitsrting) of length n
    - **bit** is the same as **bit<1>**
  - **int<n>** – Signed integer of length n (>=2)
  - **varbit<n>** – Variable-length bitstring
- **Derived Types**
  - **header** – Ordered collection of members
    - Byte-aligned
    - Can be valid or invalid
    - Can contain bit<n>, int<n> and varbit<n>
  - **struct** – Unordered collection of members
    - No alignment restrictions
    - Can contain any basic or derived types
  - Header Stacks -- arrays of headers
  - **typedef** – An alternative name for a type

Varbit

Example: Declaring IPv4 header

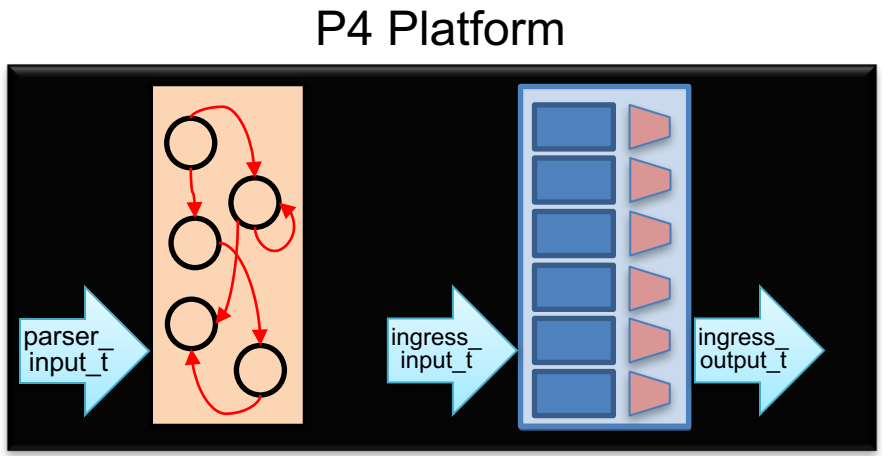
```
typedef bit<32> ipv4_addr_t;

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;
    ipv4_addr_t  srcAddr;
    ipv4_addr_t  dstAddr;
}

header ipv4_options_t {
    varbit<320>  options;
}
```

- **Basic Types**
  - **bit<n>** – Unsigned integer (bitsrting) of length n
    - **bit** is the same as **bit<1>**
  - **int<n>** – Signed integer of length n (>=2)
  - **varbit<n>** – Variable-length bitstring
- **Derived Types**
  - **header** – Ordered collection of members
    - Byte-aligned
    - Can be valid or invalid
    - Can contain bit<n>, int<n> and varbit<n>
  - **struct** – Unordered collection of members
    - No alignment restrictions
    - Can contain any derived types
  - Header Stacks -- arrays of headers
  - **typedef** – An alternative name for a type

Using structs for Intrinsic Metadata

```
typedef bit<9>  port_id_t;  /* Switch port     */
typedef bit<16> mgid_t;     /* Multicast Group */
typedef bit<5>  qid_t;      /* Queue  ID       */

struct parser_input_t {
    port_id_t ingress_port;
    bit<1>    resubmit_flag;
}

struct ingress_input_t {
    portid_t       ingress_port;
    timestamp_t    ingress_timestamp;
}

struct ingress_output_t {
    portid_t       egress_port;
    mgid_t         mcast_group;
    bit<1>         drop_flag;
    qid_t          egress_queue;
}
```

- **Intrinsic Metadata is the data that a P4-programmable components can use to interface with the rest of the system**
- **These definitions come from the files, supplied by the vendor**



Declaring and Initializing Variables

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

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

ethernet_t      eth;
vlan_tag_t      vtag = { 3w2, 0, 12w13, 16w0x8847 };
```

Better than #define!

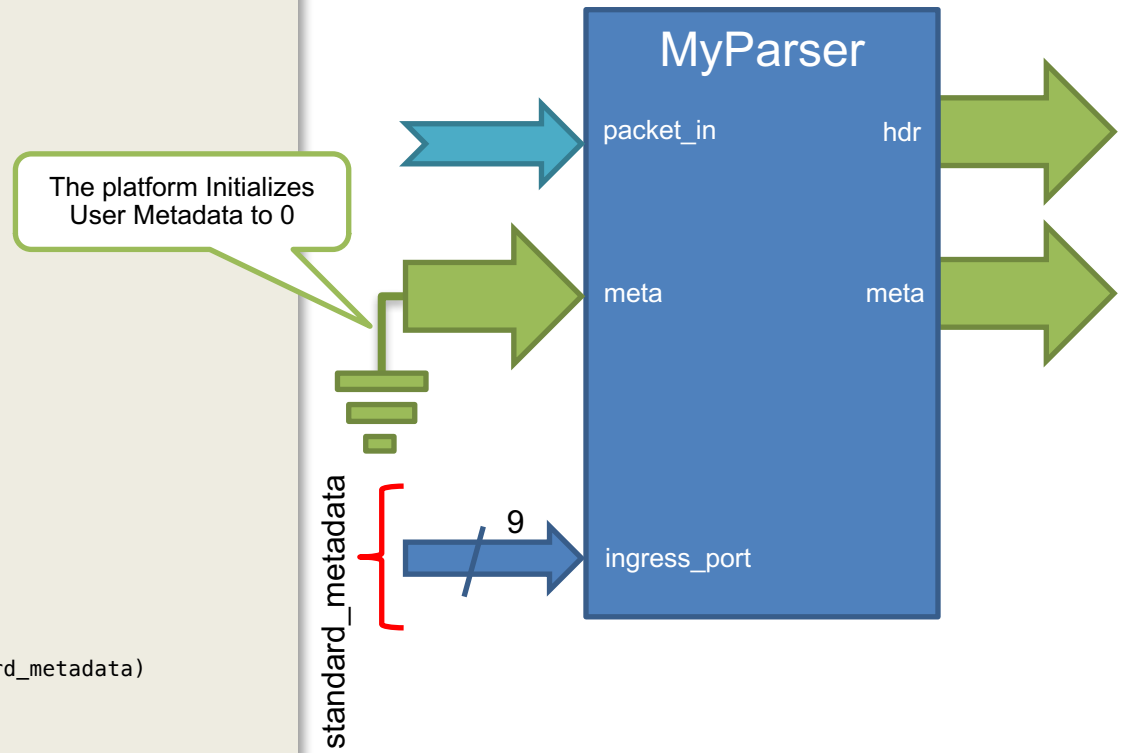
Safe constants with explicit widths

- **In P4$_{16}$ you can instantiate variables of both base and derived types**
- **Variables can be initialized**
  - Including the composite types
- **Constant declarations make for safer code**
- **Infinite width and explicit width constants**

Programming the Parser

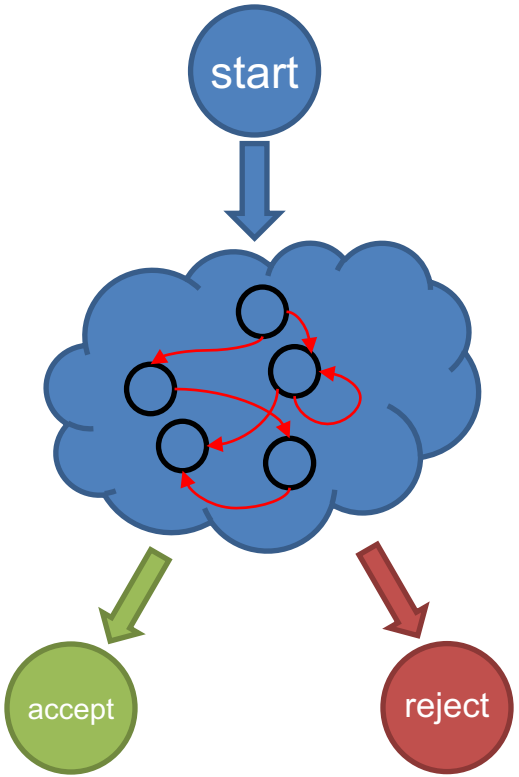
Parser Model (V1 Architecture)

```
#include <core.p4>
#include <v1model.p4>

/* User-defined inputs and outputs */
struct my_headers_t {
    ethernet_t    ethernet;
    vlan_tag_t[2] vlan_tag;
    ipv4_t        ipv4;
    ipv6_t        ipv6;
}

struct my_metadata_t {
    /* Nothing yet */
}

/* System-provided inputs. Others to follow */
struct standard_metadata_t {
    bit<9>   ingress_port;
    . . .
}

/* Parser Declaration */
parser MyParser(packet_in                packet,
                out   my_headers_t       hdr,
                inout my_metadata_t      meta,
                inout standard_metada_t  standard_metadata)
{
   . . .
```



Parsers in P4$_{16}$

- **Parsers are special functions written in a state machine style**
- **Parsers have three predefined states**
  - start
  - accept
  - reject
    - Can be reached explicitly or implicitly
    - What happens in reject state is defined by an architecture
- **Other states are user-defined**



Implementing Parser State Machine

```
parser MyParser(packet_in                 packet,
                out   my_headers_t        hdr,
                inout my_metadata_t       meta,
                in    standard_metadata_t standard_metadata)
{
    state start {
        packet.extract(hdr.ethernet);
        transition select(hdr.ethernet.etherType) {
            0x8100 &&& 0xEFFF : parse_vlan_tag;
            0x0800  : parse_ipv4;
            0x86DD  : parse_ipv6;
            0x0806  : parse_arp;
            default : accept;
        }
    }

    state parse_vlan_tag {
        packet.extract(hdr.vlan_tag.next);
        transition select(hdr.vlan_tag.last.etherType) {
             0x8100 : parse_vlan_tag;
             0x0800 : parse_ipv4;
             0x86DD : parse_ipv6;
             0x0806 : parse_arp;
            default : accept;
        }
    }
```

```
    state parse_ipv4 {
        packet.extract(hdr.ipv4);
        transition select(hdr.ipv4.ihl) {
            0 .. 4: reject;
                 5: accept;
           default: parse_ipv4_options;
    }

    state parse_ipv4_options {
        packet.extract(hdr.ipv4.options,
                       (hdr.ipv4.ihl - 5) << 2);
        transition accept;
    }

    state parse_ipv6 {
        packet.extract(hdr.ipv6);
        transition accept;
    }
```

Selecting on Multiple fields. Parsing ARP

```
const bit<16> ARP_HTYPE_ETHERNET = 0x0001;
const bit<16> ARP_PTYPE_IPV4     = 0x0800;
const bit<8>  ARP_HLEN_ETHERNET  = 6;
const bit<8>  ARP_PLEN_IPV4      = 4;

    state parse_arp {
        packet.extract(hdr.arp);
        transition select(hdr.arp.htype,
                          hdr.arp.ptype,
                          hdr.arp.hlen,
                          hdr.arp.plen) {
            (ARP_HTYPE_ETHERNET,
             ARP_PTYPE_IPV4,
             ARP_HLEN_ETHERNET,
             ARP_PLEN_IPV4) : parse_arp_ipv4;
                   default : accept;
        }
    }

    state parse_arp_ipv4 {
        packet.extract(hdr.arp_ipv4);
        transition accept;
    }
}
```

These definitions should go in the beginning of the parser function or in the top-level

hdr.arp

hdr.arp_ipv4

Internet Protocol (IPv4) over Ethernet ARP packet		
octet offset	0	1
0	Hardware type (HTYPE). (Ethernet = 0x0001)	
2	Protocol type (PTYPE) (IPv4 = 0x0800)	
4	Hardware address length (HLEN)	Protocol address length (PLEN)
6	Operation (OPER)	
8	Sender hardware address (SHA) (first 2 bytes)	
10	(next 2 bytes)	
12	(last 2 bytes)	
14	Sender protocol address (SPA) (first 2 bytes)	
16	(last 2 bytes)	
18	Target hardware address (THA) (first 2 bytes)	
20	(next 2 bytes)	
22	(last 2 bytes)	
24	Target protocol address (TPA) (first 2 bytes)	
26	(last 2 bytes)	

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);
        verify(hdr.ipv4.ihl >= 5,     error.IPv4BadHeader);
        transition select(hdr.ipv4.ihl) {
                  5: accept;
            default: parse_ipv4_options;
   }
   . . .
```

Lookahead

Example: Typical MPLS Heuristic

```
header ip46_t {    /* Common for both IPv4 and IPv6 */
    bit<4> version;
    bit<4> reserved;
}

state parse_mpls {
    packet.extract(hdr.mpls.next);
    transition select(hdr.mpls.last.bos) {
        0: parse_mpls;
        1: guess_mpls_payload;
    }
}

state guess_mpls_payload {
    transition select(packet.lookahead<ip46_t>().version) {
              4 : parse_inner_ipv4;
              6 : parse_inner_ipv6;
        default : parse_inner_ethernet;
    }
}
```

- **lookahead is a generic method**
  - Compiler generates the method with the proper return type (ipv4_t) at the compile time
- **Returns the packet data without advancing the cursor**
  - The packet length is still checked
- **Much safer and easier to use than bit offsets**

Programming the Match-Action Pipeline

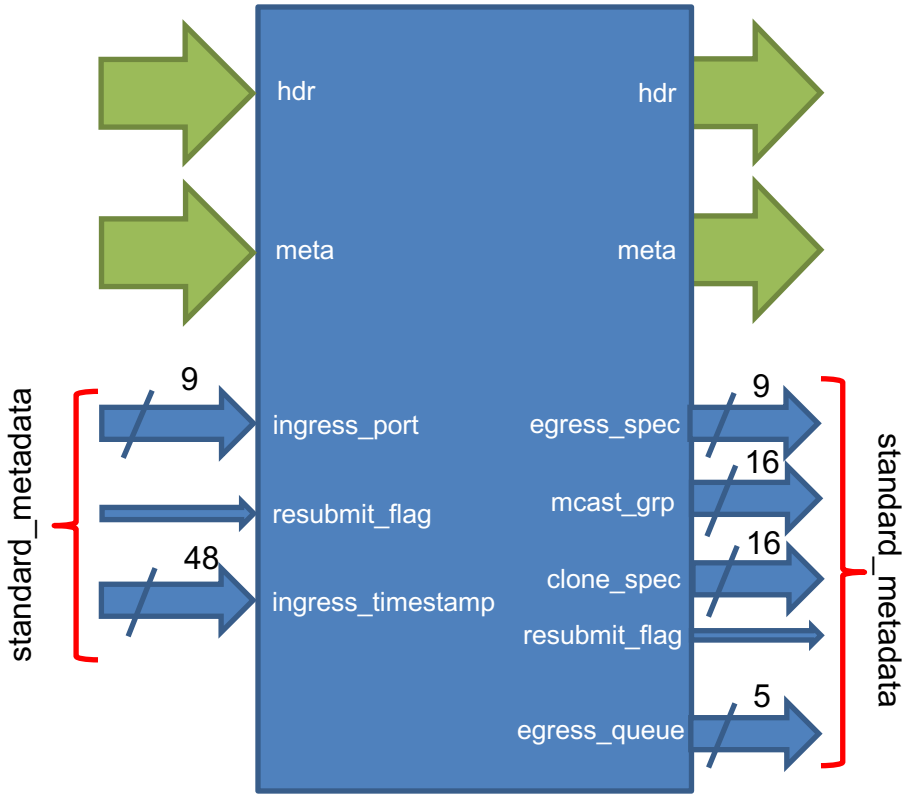
- Controls
- Actions
- Tables

Controls in P4

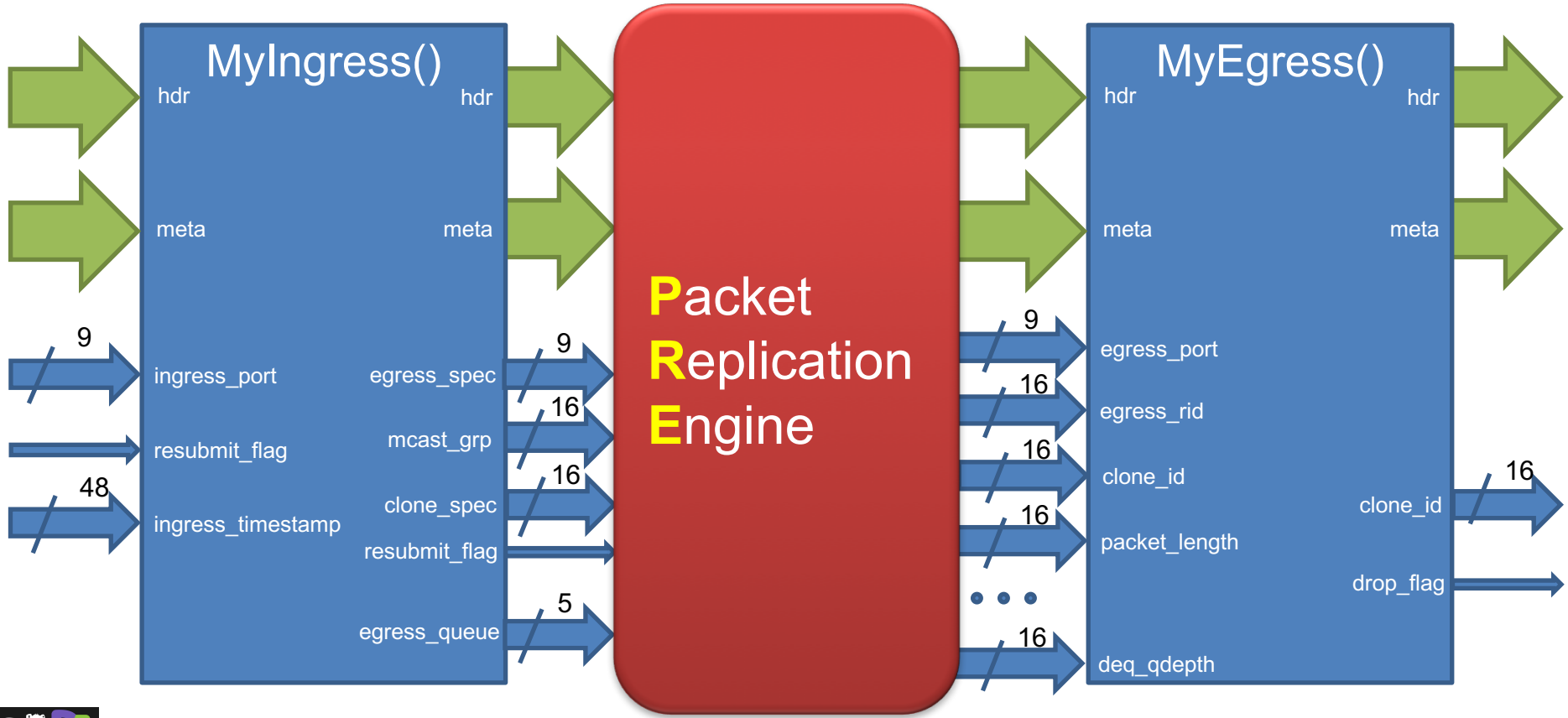
- **Very similar to C functions without loops**
  - Algorithms should be representable as Direct Acyclic Graphs (DAG)
- **Represent all kinds of processing that are expressible as DAG:**
  - Match-Action Pipelines
  - Deparsers
  - Additional processing (checksum updates)
- **Interface with other blocks via user- and architecture-defined data**

Simple Reflector (V1 Architecture)

```
control MyIngress(
    inout my_headers_t          hdr,
    inout my_metadata_t         meta,
    inout standard_metadata_t   standard_metadata)
{
    bit<48> tmp;

    apply {
        tmp = hdr.ethernet.dstAddr;
        hdr.ethernet.dstAddr = hdr.ethernet.srcAddr;
        hdr.ethernet.srcAddr = tmp;

        standard_metadata.egress_spec =
                            standard_metadata.ingress_port;
    }
}
```



How does it work? (V1 Architecture)



Simple Actions and Expressions

```
const bit<9> DROP_PORT = 511; /* Specific to V1 architecture */

action mark_to_drop() {     /* Already defined in v1model.p4 */
    standard_metadata.egress_spec = DROP_PORT;
    standard_metadata.mgast_grp   = 0;
}

control MyIngress(inout my_headers_t         hdr,
                  inout my_metadata_t        meta,
                  inout standard_metadata_t  standard_metadata)
{
    /* Local Declarations */

    action swap_mac(inout bit<48> dst, inout bit<48> src) {
        bit<48> tmp;
        tmp = dst; dst = src; src = tmp;
    }

    action reflect_to_other_port() {
        standard_metadata.egress_spec =
                        standard_metadata.ingress_port ^ 1;
    }
```

- **Very similar to C functions**
- **Can be declared inside a control or globally**
- **Parameters have type and direction**
- **Variables can be instantiated inside**

- **Standard Arithmetic and Logical operations are supported**
  - +, -, *
  - ~, &, |, ^, >>, <<
  - ==, !=, >, >=, <, <=
  - No division/modulo
- **Additional operations:**
  - Bit-slicing: [m:l]
    - Works as l-value too
  - Bit Concatenation: ++

Simple Actions and Expressions

```
const bit<9> DROP_PORT = 511;  /* V1 Architecure-specific */

action mark_to_drop() {  /* Already defined in v1model.p4 */
    standard_metadata.egress_spec = DROP_PORT;
    standard_metadata.mgast_grp   = 0;
}

control MyIngress(inout my_headers_t        hdr,
                  inout my_metadata_t       meta,
                  inout standard_metadata_t standard_metadata)
{
    /* Local Declarations */

    action swap_mac(inout bit<48> dst, inout bit<48> src) {
        bit<48> tmp;
        tmp = dst; dst = src; src = tmp;
    }

    action reflect_to_other_port() {
        standard_metadata.egress_spec =
                          standard_metadata.ingress_port ^ 1;
    }
```

```
    /* The body of the control */

    apply {
        if (hdr.ethernet.dstAddr[40:40] == 0x1) {
            mark_to_drop();
        } else {
            swap_mac(hdr.ethernet.dstAddr,
                     hdr.ethernet.srcAddr);
            reflect_to_other_port();
        }
    }
}
```

Actions Galore: Operating on Headers

Example: Encapsulating IPv4 into a 2-label MPLS packet

```
header mpls_t {
    bit<20> label;
    bit<3>  exp;
    bit<1>  bos;
    bit<8>  ttl;
}

action ipv4_in_mpls(in bit<20> label1, in bit<20> label2) {
    hdr.mpls[0].setValid();
    hdr.mpls[0].label = label1;
    hdr.mpls[0].exp   = 0;
    hdr.mpls[0].bos   = 0;
    hdr_mpls[0].ttl   = 64;

    hdr.mpls[1].setValid();
    hdr.mpls[1] = { label2, 0, 1, 128 };

    if (hdr.vlan_tag.isValid()) {
        hdr.vlan_tag.etherType = 0x8847;
    } else {
        hdr.ethernet.etherType = 0x8847;
    }
}
```

if() statements are allowed in actions too!

- **Header Validity bit manipulation:**
  - `header.setValid() – add_header`
  - `header.setInvalid() – remove_header`
  - `header.isValid()`
- **Header Assignment**
  - From tuples

Actions Galore: Operating on Headers

```
action decap_ip_ip() {
    hdr.ipv4 = hdr.inner_ipv4;
    hdr.inner_ipv4.setInvalid();
}

action pop_mpls_label() {
    hdr.mpls.pop_front(1);
}

action push_mpls_label(in bit<20> label, in bit<3> exp) {
    hdr.mpls.push_front(1);
    hdr.mpls[0].setValid();
    hdr.mpls[0] = { label, exp, 0, 64 };
}
```

- **Header Validity bit manipulation:**
  - `header.setValid() – add_header`
  - `header.setInvalid() – remove_header`
  - `header.isValid()`
- **Header Assignment**
  - `header  = { f1, f2, ..., fn }`
  - `header1 = header2`
- **Special operations on Header Stacks**
  - In the parsers
    - `header_stack.next`
    - `header_stack.last`
    - `header_stack.lastIndex`
  - In the controls
    - `header_stack[i]`
    - `header_stack.size`
    - `header_stack.push_front(int count)`
    - `header_stack.pop_front(int count)`

Actions Galore: Bit Manipulation

Example: **Forming Ethernet MAC address for IPv4 Multicast Packets**

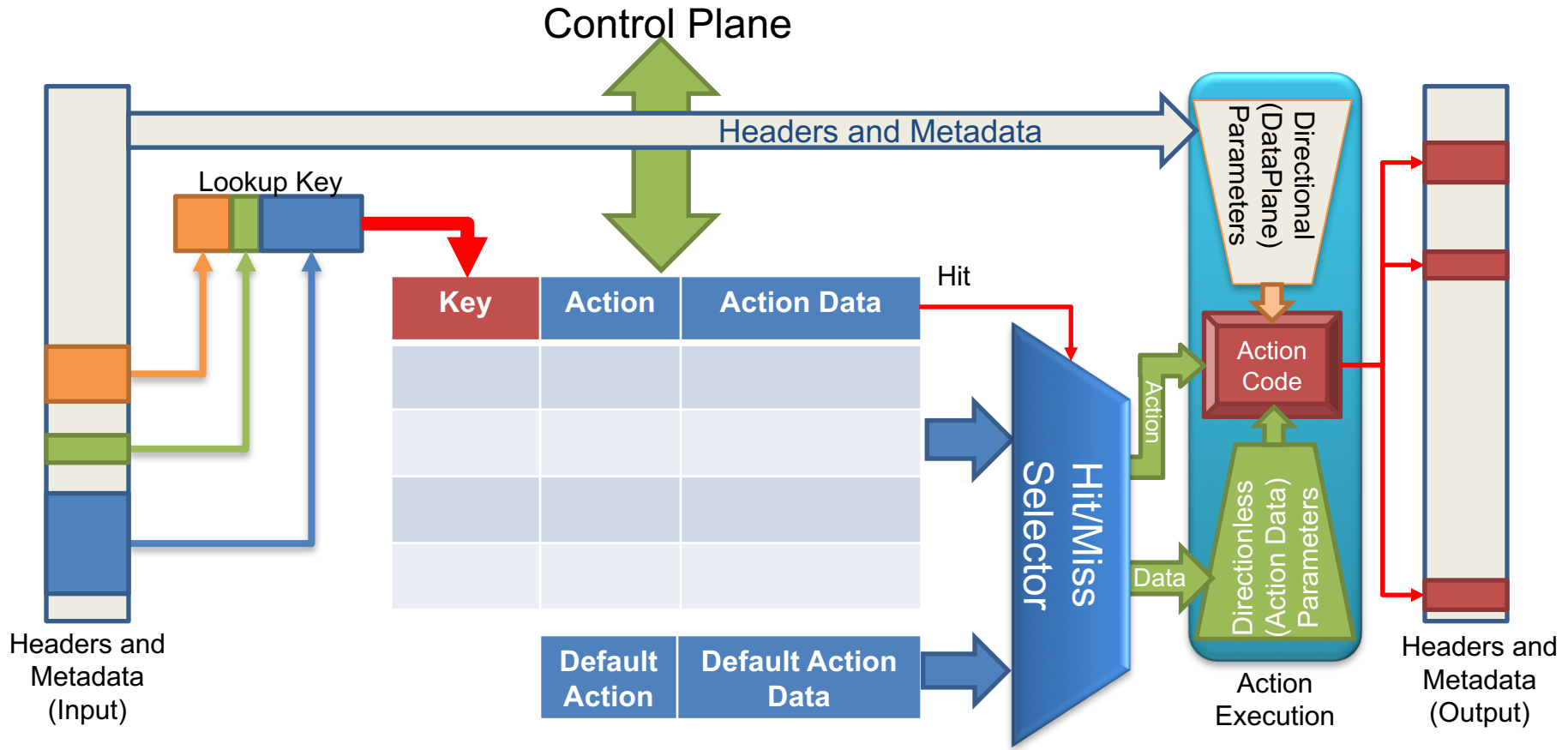
```
action set_ipmcv4_mac_da_1() {
    hdr.ethernet.dstAddr = 24w0x01005E ++ 1w0 ++
                           hdr.ipv4.dstAddr[22:0];
}

action set_ipmcv4_mac_da_2() {
    hdr.ethernet.dstAddr[47:24] = 0x01005E;
    hdr.ethernet.dstAddr[23:23] = 0;
    hdr.ethernet.dstAddr[22:0]  = hdr.ipv4.dstAddr[22:0];
}
```

- **Special Operations for bit manipulation:**
  - Bit-string concatenation
  - Bit-slicing

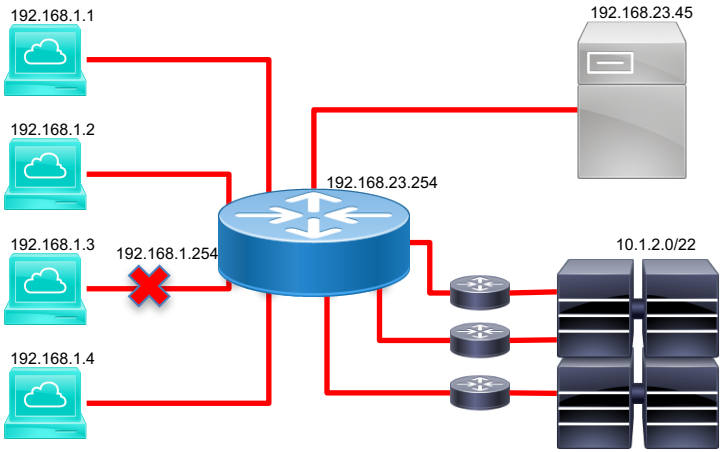
Match-Action Tables

- **The fundamental units of the Match-Action Pipeline**
  - What to match on and match type
  - A list of *possible* actions
  - Additional **properties**
    - Size
    - Default Action
    - Entries
    - etc.
- **Each table contains one or more entries (rows)**
- **An entry contains:**
  - A specific key to match on
  - A **single** action
    - to be executed when a packet matches the entry
  - (Optional) action data

Tables: Match-Action Processing



Example: Basic IPv4 Forwarding



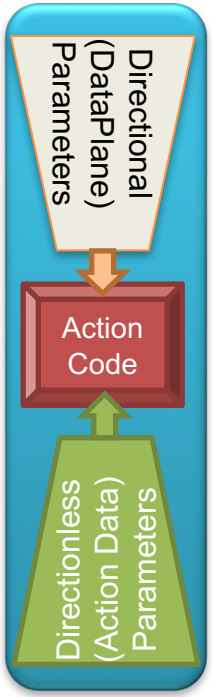
Key	Action	Action Data
192.168.1.1	l3_switch	port= mac_da= mac_sa= vlan=
192.168.1.2	l3_switch	port= mac_da= mac_sa= vlan=…
192.168.1.3	l3_drop	
192.168.1.254	l3_l2_switch	port=
192.168.1.0/24	l3_l2_switch	port=
10.1.2.0/22	l3_switch_ecmp	ecmp_group=

- **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

Defining Actions for L3 forwarding

```
action l3_switch(bit<9>  port,
                 bit<48> new_mac_da,
                 bit<48> new_mac_sa,
                 bit<12> new_vlan)
{
    /* Forward the packet to the specified port */
    standard_metadata.metadata.egress_spec = port;

    /* L2 Modifications */
    hdr.ethernet.dstAddr   = new_mac_da;
    hdr.ethernet.srcAddr   = mac_sa;
    hdr.vlan_tag[0].vlanid = new_vlan;

    /* IP header modification (TTL decrement) */
    hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
}

action l3_l2_switch(bit<9> port) {
    standard_metadata.metadata.egress_spec = port;
}

action l3_drop() {
    mark_to_drop();
}
```

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



Action Execution

Match-Action Table (Exact Match)

Example: A typical L3 (IPv4) Host table

```
table ipv4_host {
    key = {
        meta.ingress_metadata.vrf   : exact;
        hdr.ipv4.dstAddr            : exact;
    }
    actions = {
        l3_switch;  l3_l2_switch;
        l3_drop;    noAction;
    }
    default_action = noAction();
    size = 65536;
}
```

These are the only possible actions. Each particular entry can have only ONE of them.

```
/* Defined in core.p4 */
action noAction() { }
```

vrf	ipv4.dstAddr	action	data
1	192.168.1.10	l3_switch	port_id= mac_da= mac_sa=
100	192.168.1.10	l3_l2_switch	port_id=<CPU>
1	192.168.1.3	l3_drop	
	DEFAULT	noAction	

Match-Action Table (Longest Prefix Match)

Example: A typical L3 (IPv4) Routing table

```
table ipv4_lpm {
    key = {
        meta.ingress_metadata.vrf    : exact;
        hdr.ipv4.dstAddr             : lpm;
    }
    actions = {
        l3_switch;
        l3_l2_switch;
        l3_switch_nexthop(meta.l3.nexthop_info);
        l3_switch_ecmp(meta.l3.nexthop_info);
        l3_drop;  noAction;
   }
   const default_action = l3_l2_switch(CPU_PORT);
   size = 16384;
}
```

Different fields can use different match types

Prefix length also serves as a priority indicator

vrf	ipv4.dstAddr / prefix	action	data
1	192.168.1.0 / 24	l3_l2_switch	port_id=3
10	10.0.16.0 / 22	l3_ecmp	ecmp_index=12
1	192.168.0.0 / 16	l3_switch_nexthop	nexthop_index=451
1	0.0.0.0 / 0	l3_switch_nexthop	nexthop_index=1
	DEFAULT	l3_l2_switch	port_id=CPU_PORT

Match-Action Table (Ternary Match)

Example: A more powerful L3 (IPv4) Routing table

```
table ipv4_lpm {
    key = {
        meta.ingress_metadata.vrf   : ternary;
        hdr.ipv4.dstAddr            : ternary;
    }
    actions = {
        l3_switch;
        l3_l2_switch;
        l3_switch_nexthop(meta.l3.nexthop_info);
        l3_switch_ecmp(meta.l3.nexthop_info);
        l3_drop;
        noAction;
   }
   const default_action = l3_l2_switch(CPU_PORT);
   size = 16384;
}
```

Explicitly Specified Priority

Prio	vrf / mask	ipv4.dstAddr / mask	action	data
100	0x001/0xFFF	192.168.1.5 / 255.255.255.255	l3_swith_nexthop	nexthop_index=10
10	0x000/0x000	192.168.2.0/255.255.255.0	l3_switch_ecmp	ecmp_index=25
10	0x000/0x000	192.168.3.0/255.255.255.0	l3_switch_nexthop	nexthop_index=31
5	0x000/0x000	0.0.0.0/0.0.0.0	l3_l2_switch	port_id=64

Using Tables in the Controls

```
control MyIngress(inout my_headers_t        hdr,
                  inout my_metadata_t       meta,
                  inout standard_metadata_t standard_metadata)
{
    /* Declarations */
    action l3_switch(...) { . . . }
    action l3_l2_switch(...) { . . . }
    . . .
    table assign_vrf { . . . }
    table ipv4_host  { . . . }
    table ipv6_host  { . . . }

    /* Code */
    apply {
        assign_vrf.apply();
        if (hdr.ipv4.isValid()) {
            ipv4_host.apply();
        }
    }
}
```

- **Declare Actions**
  - Declaration is instantiation
- **Declare Tables**
  - Declaration is instantiation
- **Apply() Tables – Perform Match-Action**
  - Make sure the table matches on valid headers

Using the Match Results

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

apply {
    . . .
    switch (ipv4_lpm.apply().action_run) {
        l3_switch_nexthop: { nexthop.apply(); }
           l3_switch_ecmp: { ecmp.apply(); }
                  l3_drop: { exit; }
                  default: { /* Not needed. Do nothing */ }
    }
}
```

- **Apply method returns a special result:**
  - A boolean, representing the hit
  - An enum, representing a selected action
- **Switch() statement**
  - Only used for the results of match-action
  - Each case should be a block statement
  - Default case is optional
    - Means “any action” not “default action”
- **Exit and Return Statements**
  - `return` – go to the end of the current control
  - `exit` – go to the end of the top-level control
  - Useful to skip further processing

Match Kinds (types)

```
/* core.p4 */

match_kind {
    exact,
    ternary,
    lpm
}

/* v1model.p4 */

match_kind {  /* Augments the standard definition }
    range,
    selector
}

/* Some other architecture */

match_kind {
    regexp,
    range,
    fuzzy,
    telepathy
}
```

- **match_kind is a special type in P4**
- **core.p4 defines three basic match kinds**
  - Exact match
  - Ternary match
  - LPM match
- **Architectures can add their own match kinds**

Advanced Matching

```
table classify_ethernet {
    key = {
        hdr.ethernet.dstAddr      : ternary;
        hdr.ethernet.srcAddr      : ternary;
        hdr.vlan_tag[0].isValid() : ternary;
        hdr.vlan_tag[1].isValid() : ternary;
    }
    actions = {
        malformed_ethernet;
        unicast_untagged;
        unicast_single_tagged;
        unicast_double_tagged;
        multicast_untagged;
        multicast_single_tagged;
        multicast_double_tagged;
        broadcast_untagged;
        broadcast_single_tagged;
        broadcast_double_tagged;
    }
```

- **Tables keys can be arbitrary expressions**
- **Check header validity**
  - `header.isValid()`
- **Use only important bits**
  - `header.field[msb:lsb]`
    - `hdr.ipv6.dstAddr[127:64] : lpm;`
- **Fantasy is your limit:**
  - `hdr.ethernet.srcAddr ^ hdr.ethernet.dstAddr`

Table Initialization

```
    /* Continued from previous slide */
    const entries = {
          /* { dstAddr, srcAddr, vlan_tag[0].isValid(), vlan_tag[1].isValid() } : action([action_data]) */
        { 48w000000000000,                    _,              _, _ } : malformed_ethernet(ETHERNET_ZERO_DA);
        { _,                                  48w000000000000, _, _ } : malformed_ethernet(ETHERNET_ZERO_SA);
        { _,              48w010000000000 &&& 48w010000000000, _, _ } : malformed_ethernet(ETHERNET_MCAST_SA);
        { 48wFFFFFFFFFFFF,                    _,              0, _ } : broadcast_untagged();
        { 48wFFFFFFFFFFFF,                    _,              1, 0 } : broadcast_single_tagged();
        { 48wFFFFFFFFFFFF,                    _,              1, 1 } : broadcast_double_tagged();
        { 48w010000000000 &&& 48w010000000000, _,             0, _ } : multicast_untagged();
        { 48w010000000000 &&& 48w010000000000, _,             1, 0 } : multicast_single_tagged();
        { 48w010000000000 &&& 48w010000000000, _,             1, 1 } : multicast_double_tagged();
        { _,                                  _,              0, _ } : unicast_untagged();
        { _,                                  _,              1, 0 } : unicast_single_tagged();
        { _,                                  _,              1, 1 } : unicast_double_tagged();
    }
}
```

Packet Deparsing

Deparsing

```
control MyDeparser(packet_out      packet,
                   in my_headers_t hdr)
{
    apply {
        /* Layer 2 */
        packet.emit(hdr.ethernet);
        packet.emit(hdr.vlan_tag);

        /* Layer 2.5 */
        packet.emit(hdr.mpls);

        /* Layer 3 */
           /* ARP */
        packet.emit(hdr.arp);
        packet.emit(hdr.arp_ipv4);
           /* IPv4 */
        packet.emit(hdr.ipv4);
           /* IPv6 */
        packet.emit(hdr.ipv6);

        /* Layer 4 */
        packet.emit(hdr.icmp);
        packet.emit(hdr.tcp);
        packet.emit(hdr.udp);
    }
}
```

- **Assembling the packet from headers**
- **Expressed as another control function**
  - No need for another construct
- **packet_out – defined in core.p4**
  - emit(header) – serialize the header **if** it is valid
  - emit(header_stack) – serialize the valid elements in order
- **Advantages:**
  - Decoupling of parsing and deparsing

Simplified Deparsing

```
struct my_headers_t {
    ethernet_t     ethernet;
    vlan_tag_t [2] vlan_tag;
    mpls_t     [5] mpls;
    arp_t          arp;
    arp_ipv4_t     arp_ipv4;
    ipv4_t         ipv4;
    ipv6_t         ipv6;
    icmp_t         icmp;
    tcp_t          tcp;
    udp_t          udp;
}

control MyDeparser(packet_out      packet,
                   in my_headers_t hdr)
{
    apply {
        packet.emit(hdr);
    }
}
```

- **Simply keep the header struct organized**
  - Headers will be deparsed in order

Externs

The Need for Externs

- **Most platforms contain specialized facilities**
  - They differ from vendor to vendor
  - They can't be expressed in the core language
    - Specialized computations
  - They might have control-plane accessible state or configuration
- **The language should stay the same**
  - In $P4_{14}$ almost 1/3 of all the constructs were dedicated to specialized processing
  - In $P4_{16}$ all specialized objects use the same interface
- **Objects can be used even if their implementation is hidden**
  - Through instantiation and method calling

Stateless and Stateful Objects

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

Object	Data Plane Interface		Control Plane Can	
	Read State	Modify/Write State	Read	Modify/Write
Table	apply()	---	Yes	Yes
Parser Value Set	get()	---	Yes	Yes
Counter	---	count()	Yes	Yes*
Meter	execute ()		Configuration Only	Configuration Only
Register	read()	write()	Yes	Yes

Counters in V1 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);
}
```

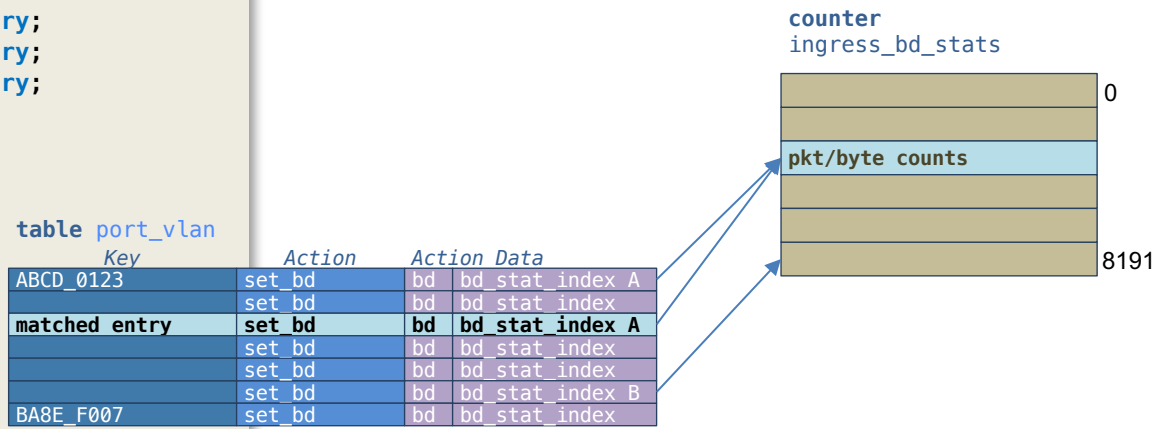
- **Extern definition contains**
  - The instantiation method
    - Has the same name as the extern
    - Is evaluated at compile-time
  - Methods to access the extern
    - Very similar to actions
    - Can return values too

- **Enums in $P4_{16}$**
  - Abstract values
    - No specific (numerical representation)

Using the V1 Architecture 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**



Meters in V1 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();
    }
}
```

Registers in V1 Architecture

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(last_pkt_ts,
                  (bit<32>)flow_id);

   /* 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);
}
```

Header Checksums (V1 Architecture)

IPv4 Checksum Verification

```
control MyVerifyChecksum(in    my_headers_t   hdr,
                         inout my_metadata_t  meta)
{
    Checksum16() ipv4_checksum;
    bit<16>      ck;

    apply {
        if (hdr.ipv4.isValid()) {
            ck = ipv4_checksum.get(
                {
                  hdr.ipv4.version,        hdr.ipv4.ihl,
                  hdr.ipv4.diffserv,       hdr.ipv4.totalLen,
                  hdr.ipv4.identification, hdr.ipv4.flags,
                  hdr.ipv4.fragOffset,     hdr.ipv4.ttl,
                  hdr.ipv4.protocol,       hdr.ipv4.srcAddr,
                  hdr.ipv4.dstAddr
                });
            if (hdr.ipv4.hdrChecksum != ck) {
                mark_to_drop();
            }
        }
    }
}
```

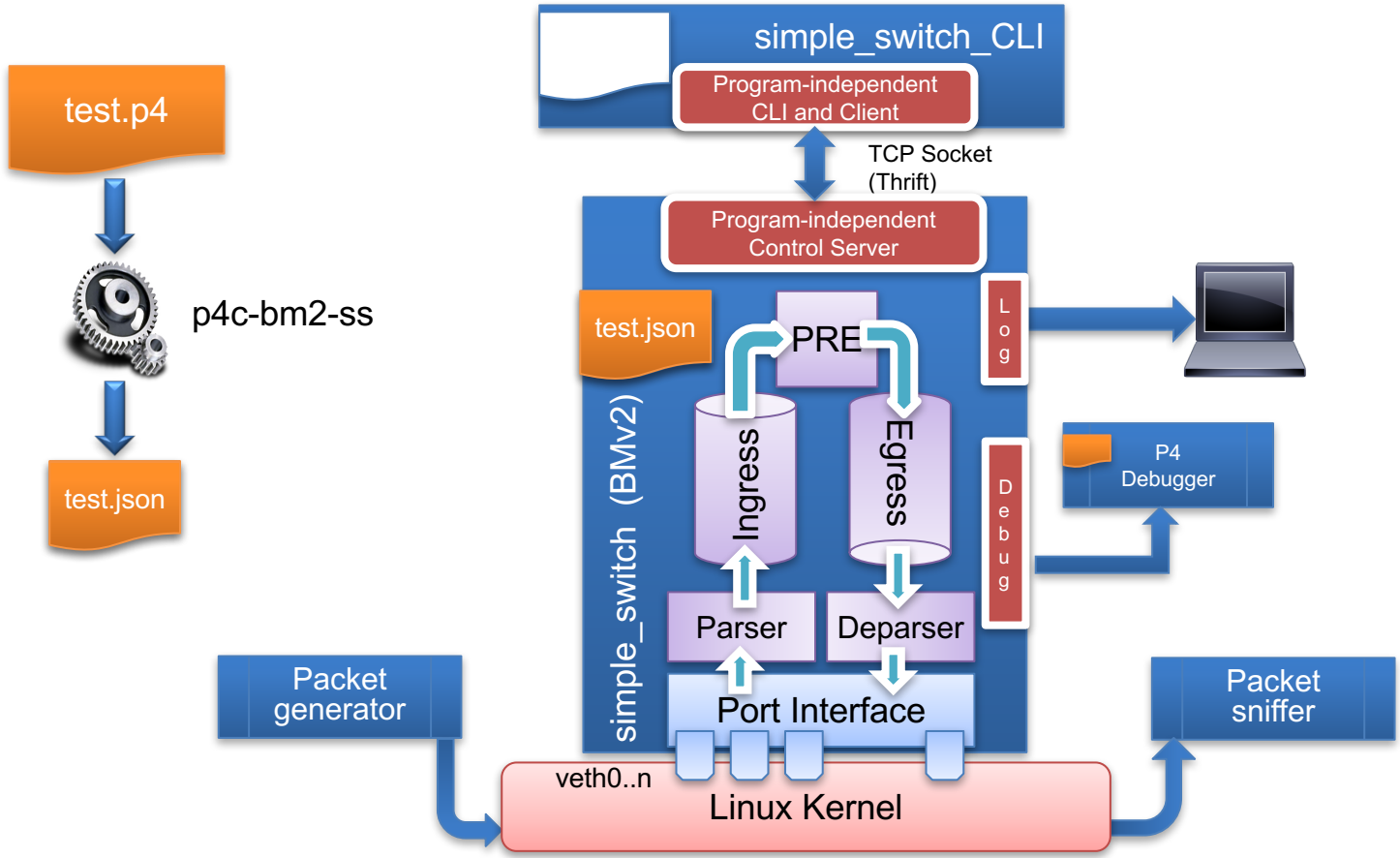
IPv4 Checksum Update

```
control MyComputeChecksum(inout my_headers_t  hdr,
                          inout my_metadata_t meta)
{
    Checksum16() ipv4_checksum;

    apply {
        if (hdr.ipv4.isValid()) {
            hdr.ipv4.hdrChecksum = ipv4_checksum.get(
                {
                  hdr.ipv4.version,        hdr.ipv4.ihl,
                  hdr.ipv4.diffserv,       hdr.ipv4.totalLen,
                  hdr.ipv4.identification, hdr.ipv4.flag,
                  hdr.ipv4.fragOffset,     hdr.ipv4.ttl,
                  hdr.ipv4.protocol,       hdr.ipv4.srcAddr,
                  hdr.ipv4.dstAddr
                });
        }
    }
}
```

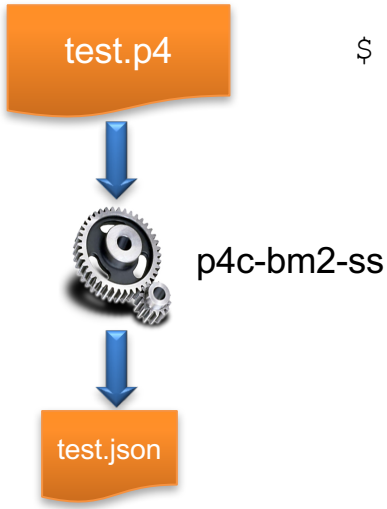
Break

P4 Toolchain for BMv2 software simulation

Basic Workflow

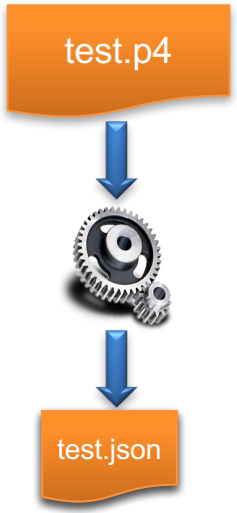


Step 1: P4 Program Compilation

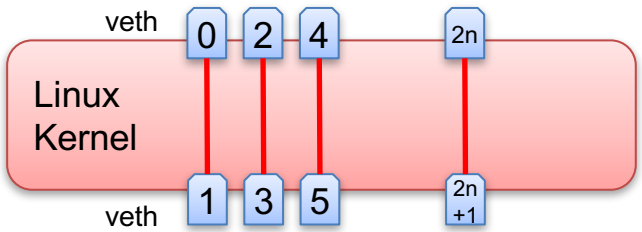



```
$ p4c-bm2-ss -o test.json test.p4
```

Step 2: Preparing veth Interfaces

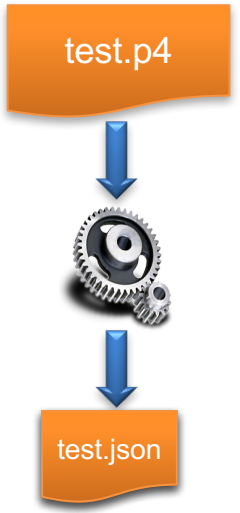


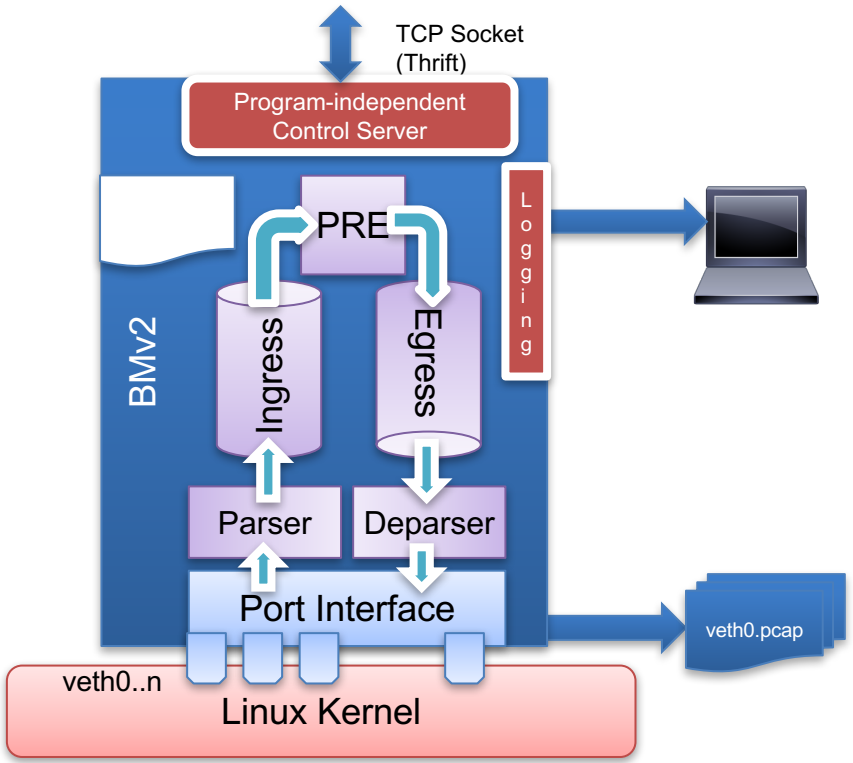

```
$ sudo ~/p4lang/tutorials/examples/veth_setup.sh

# ip link add name veth0 type veth peer name veth1
# for iface in "veth0 veth1"; do
    ip link set dev ${iface} up
    sysctl net.ipv6.conf.${iface}.disable_ipv6=1
    TOE_OPTIONS="rx tx sg tso ufo gso gro lro rxvlan txvlan rxhash"
    for TOE_OPTION in $TOE_OPTIONS; do
        /sbin/ethtool --offload $intf "$TOE_OPTION"
    done
  done
```



Step 3: Starting the model

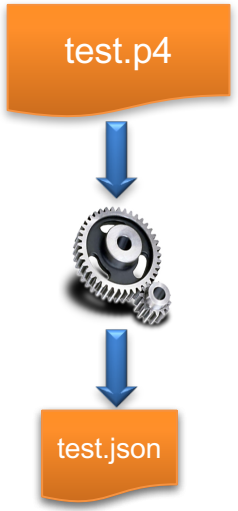
```
$ sudo simple_switch --log-console --dump-packet-data 64 \
       -i 0@veth0 -i 1@veth2 …                           \
       test.json
```

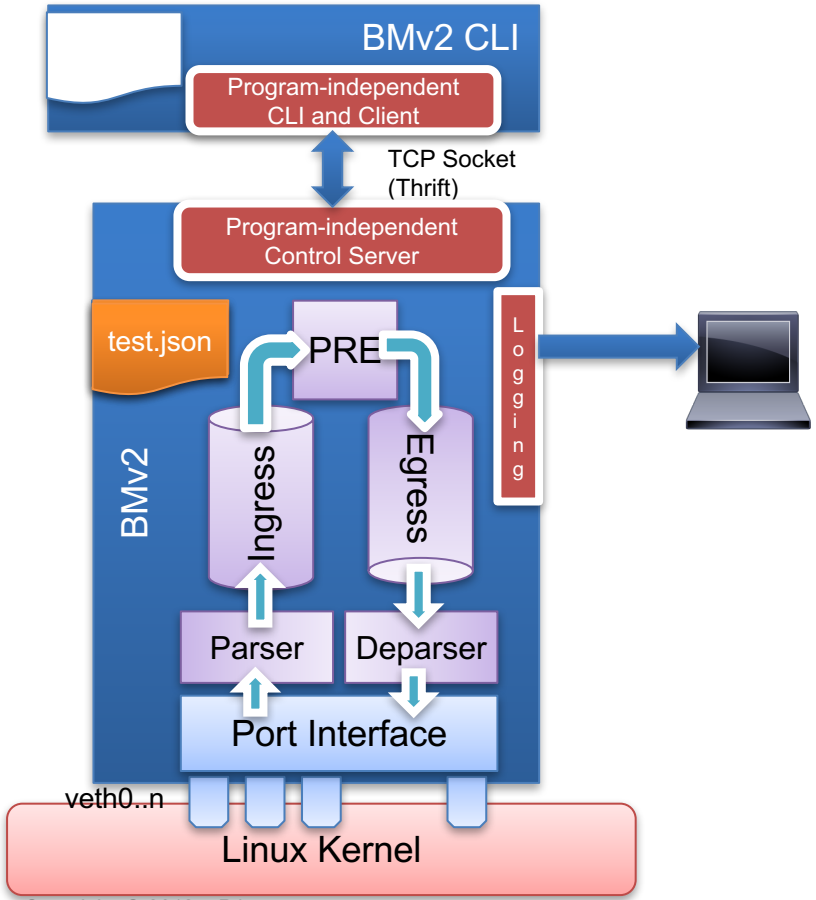




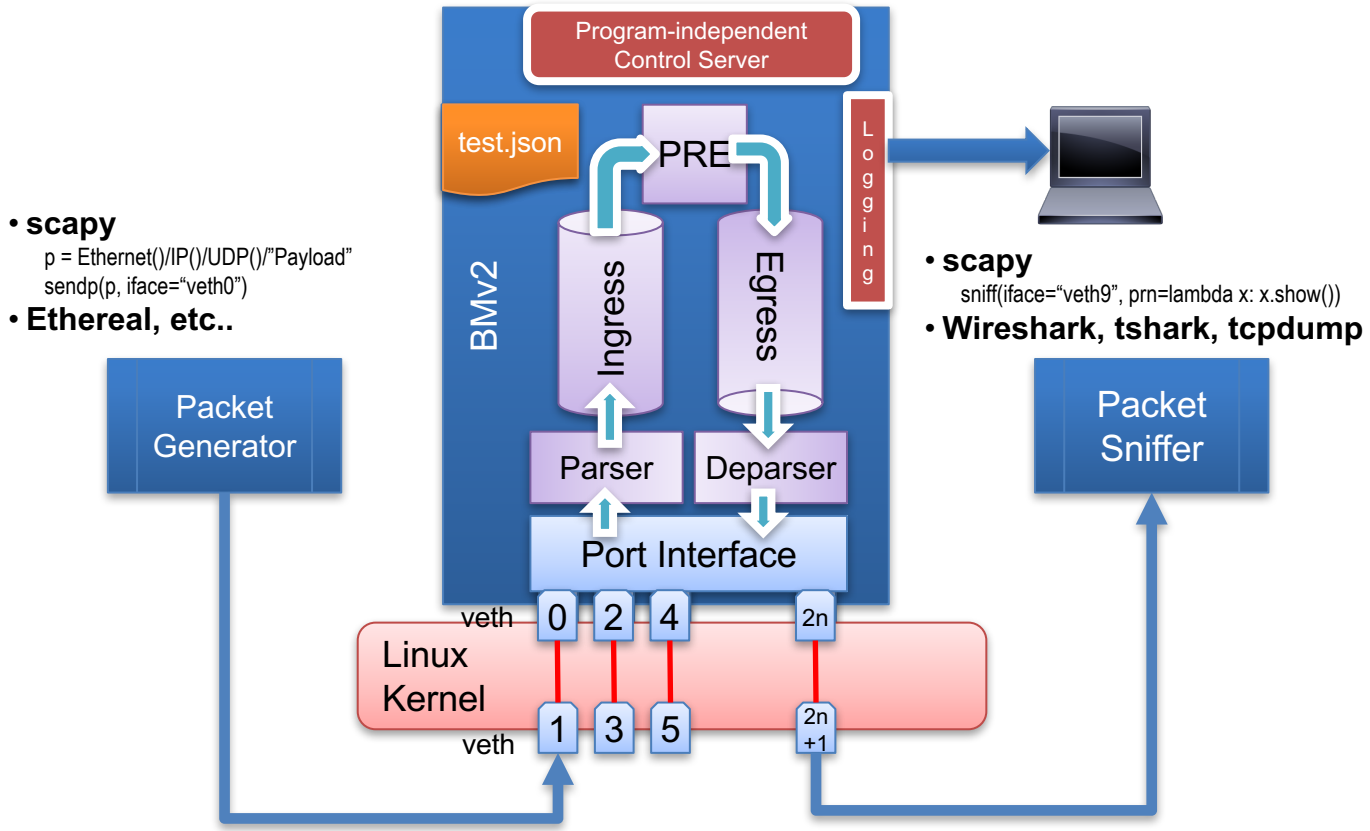
Step 4: Starting the CLI

```
$ simple_switch_CLI
```





Step 5: Sending and Receiving Packets



Other tools

- **Mininet**
  - On a single host, emulate a **network** of multiple devices with a set of interconnecting links that you configure.

- **P4 Runtime - A program independent (PI) API for**
  - Loading P4 programs into devices
  - Adding and removing table entries
  - Configuring meters and other externs, reading counter statistics
  - Either locally or remotely, over a TCP socket using Google Protocol Buffers
  - p4-api working group is writing a spec and developing the code

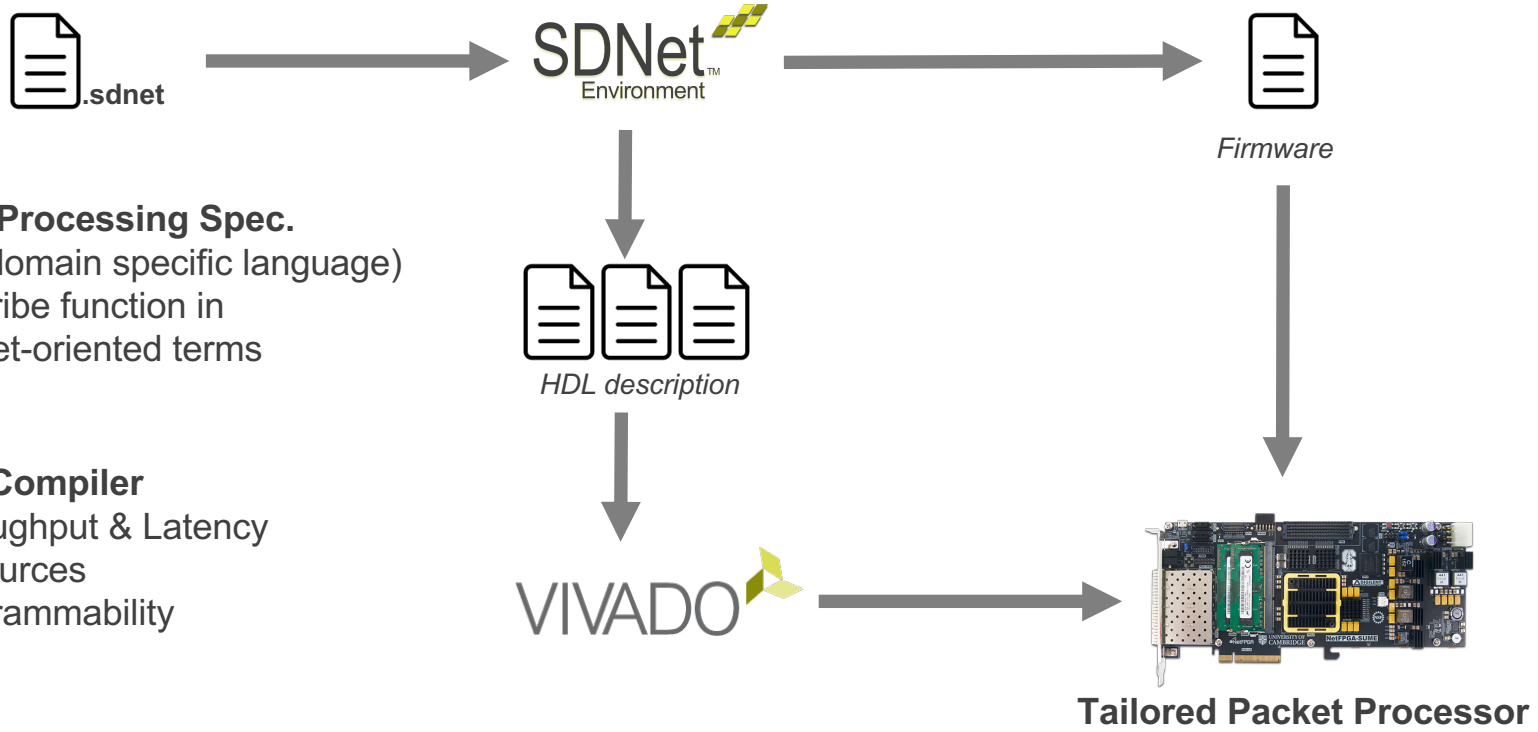
Extending the tools

- **All are open source - https://github.com/p4lang**
- **P4c compiler has multiple back ends already:**
  - bmv2
  - EBPF – Extended Berkeley Packet Filter, running in Linux kernel
  - Xilinx FPGAs
- **Designed to add back ends for additional devices**
  - Compiler internal documentation in **docs** directory of p4c Github repo
- **Portable Switch Architecture (PSA)**
  - p4-arch working group is developing many useful P4_16 externs
  - Good reference for how to add your own

XILINX
ALL PROGRAMMABLE™

The p4c-sdnet Compiler

Xilinx SDNet Design Flow & Use Model



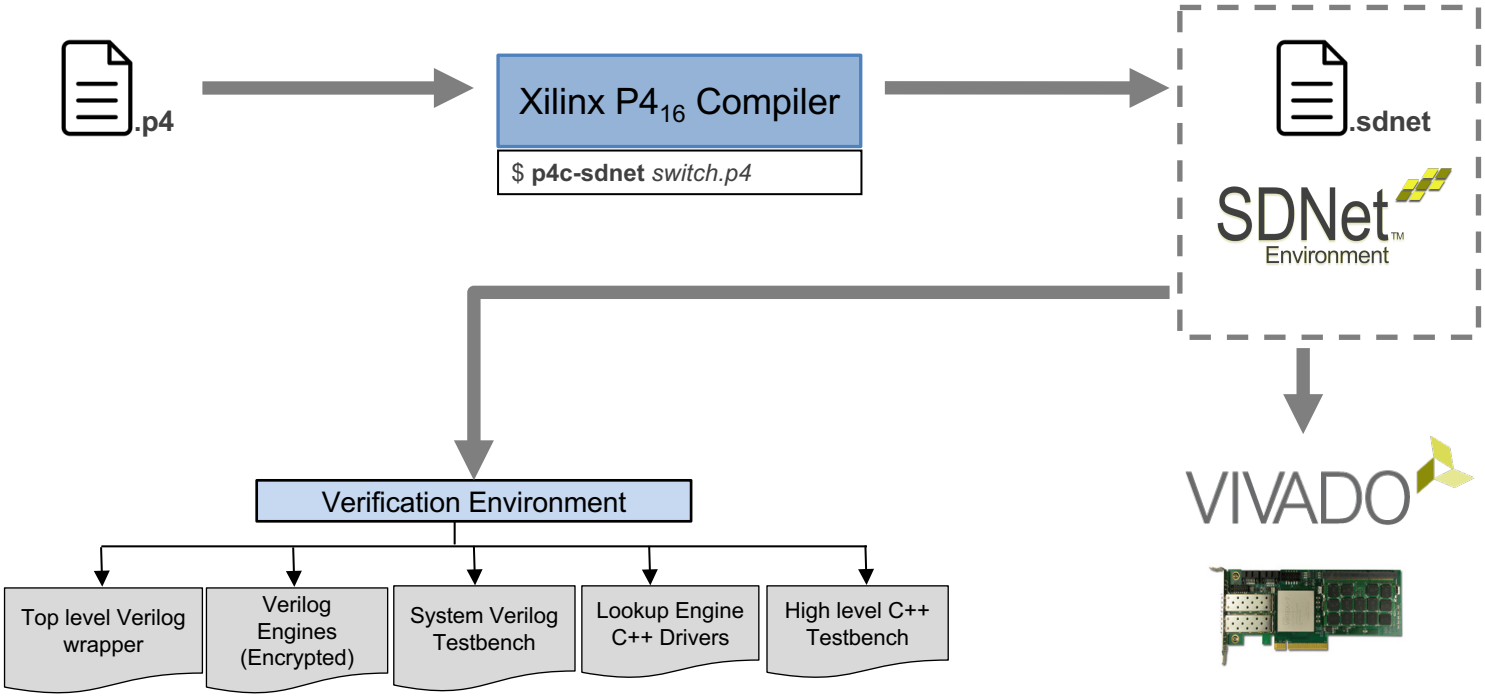
Packet Processing Spec.

- PX (domain specific language)
- describe function in packet-oriented terms

SDNet Compiler

- Throughput & Latency
- Resources
- Programmability

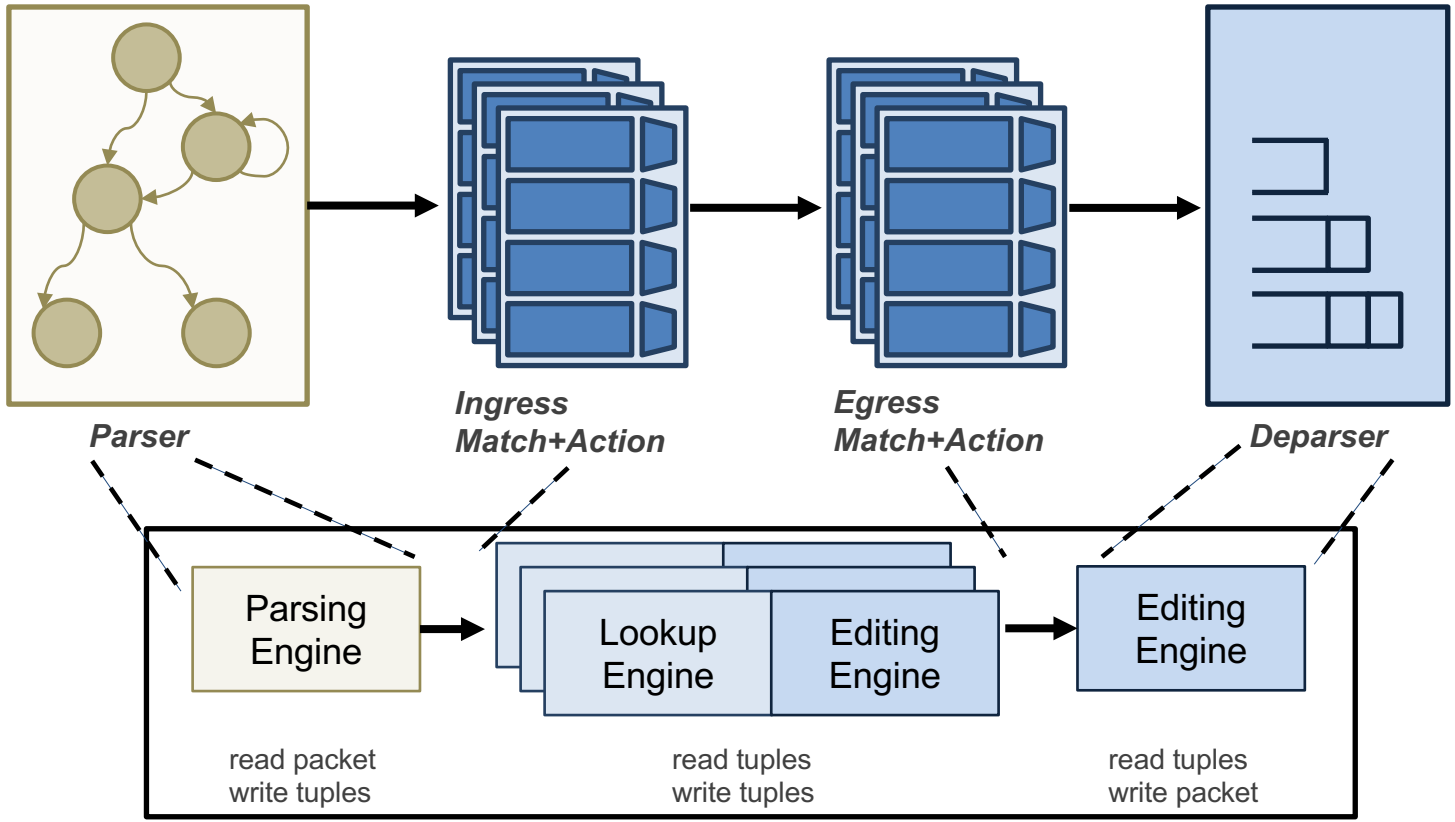
Xilinx P4 Design Flow & Use Model



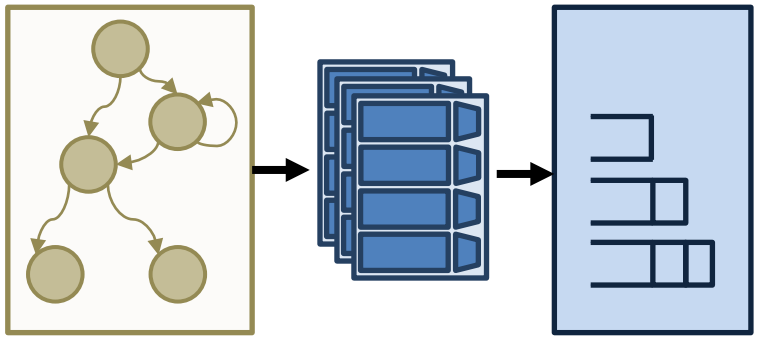
Considerations When Mapping to SDNet

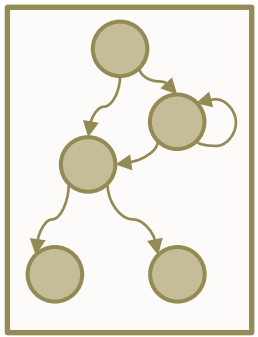
- **Identifying parallelism within P4 parser and control blocks**
  - table lookups
  - actions
  - etc.
- **P4 packet processing model**
  - extract entire header from packet
  - updates apply directly to header
  - deparser re-inserts header back into packet
- **SDNet packet processing model**
  - stream packet through “engines”
  - modify header values in-line without removing and re-inserting

Mapping P4 Architectures to SDNet



Support for Multiple Architectures



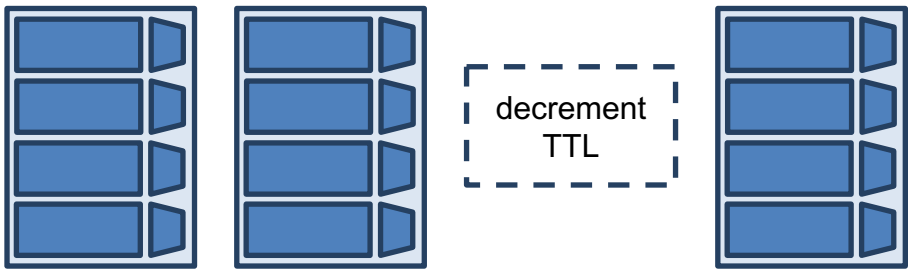


- Single Match+Action Pipeline
  - simple updates to packet headers
- Only Parser
  - pull information from packet w/o updates

Providing Externs using Custom User Logic

```
header ipv4_t { … }

extern void decrement_ttl (inout ipv4_t ip);

Control MyIngress (
   inout my_headers_t          hdr,
   inout my_metadata_t         meta,
   inout standard_metadata_t  std_meta)
{

   ...

   apply
   {
      ...

      if (hdr.ip.isValid())
         decrement_ttl(hdr.ip);

      ...

   }
}
```

- **User writes a custom Verilog component**
  - decrement_ttl.v
- **use P4's extern construct in the switch description file**
- **p4c-sdnet flow generates hook so module can be added into the bitstream**



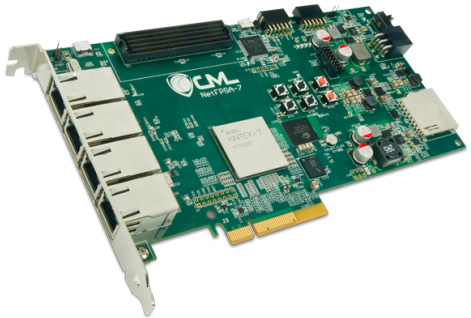
P4-SDNet availability

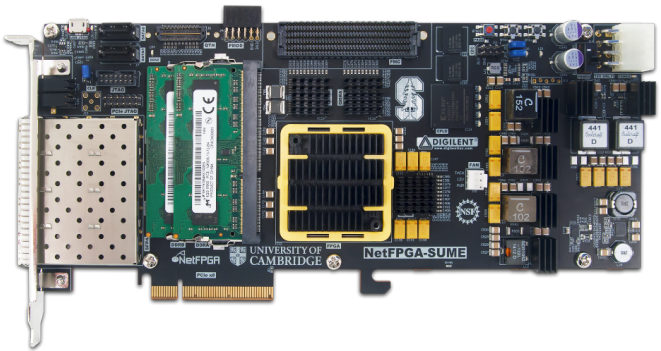
- **Currently SDNet 2017.4 – releases aligned with Vivado**
- **Documentation and download: https://www.xilinx.com/sdnet**
- **License required**
  - Donation possible via Xilinx University Program request
  - Around 40 research groups worldwide have licenses today

Overview

NetFPGA = Networked FPGA

- **A line-rate, flexible, open networking platform for teaching and research**



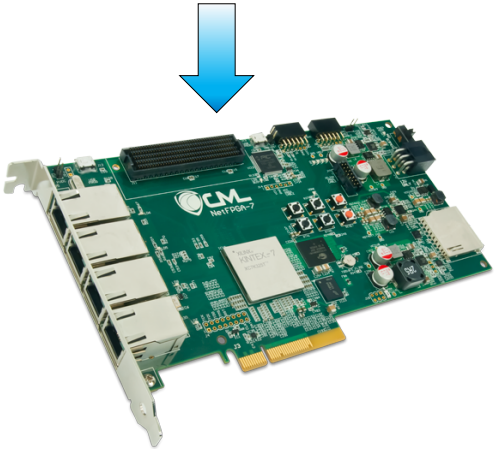


=

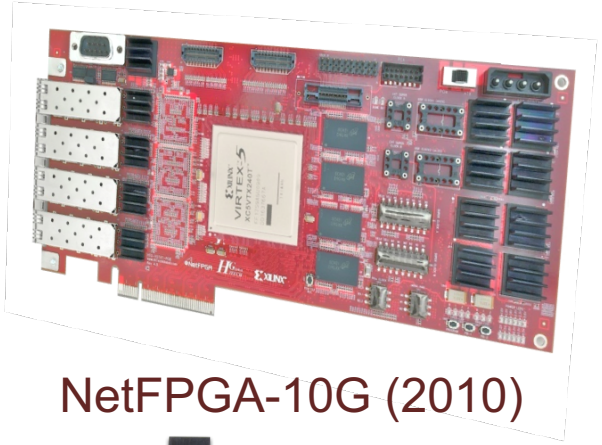
- Network Interface Card
- Hardware Accelerated Linux Router
- IPv4 Reference Router
- Traffic Generator
- Openflow Switch
- More Projects
- Add Your Project

NetFPGA Family of Boards

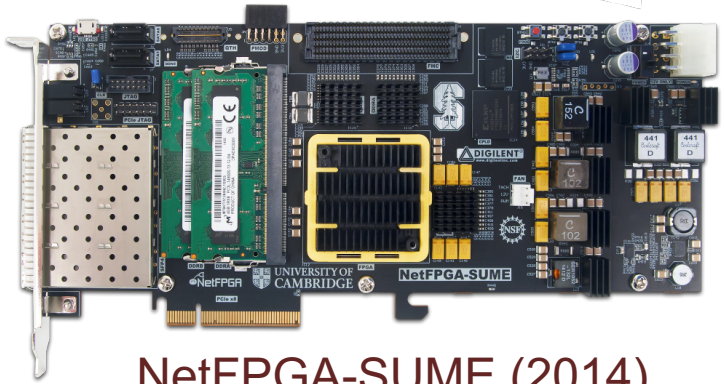
NetFPGA-1G (2006)



NetFPGA-1G-CML (2014)



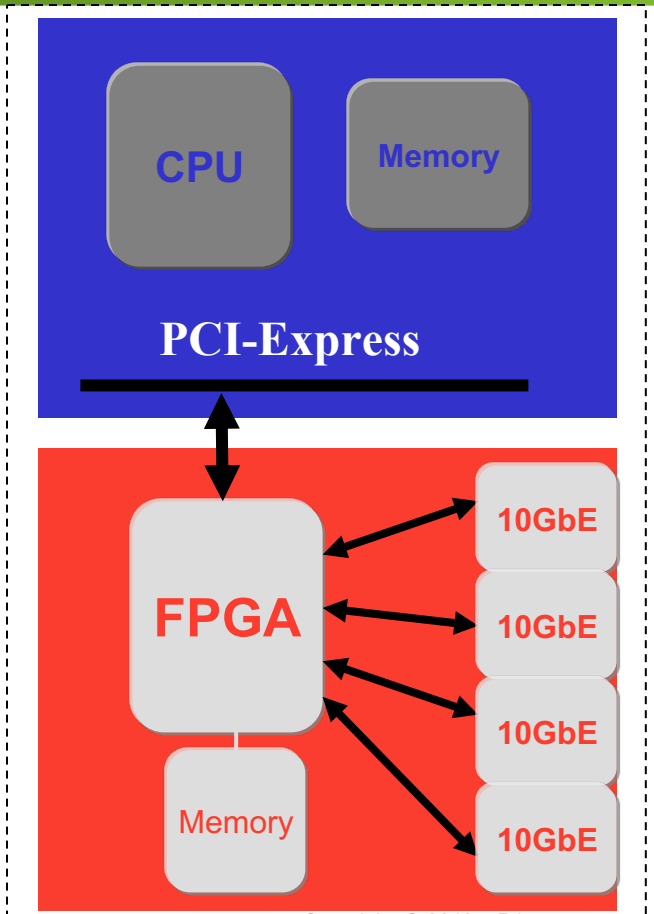
NetFPGA-10G (2010)



NetFPGA-SUME (2014)

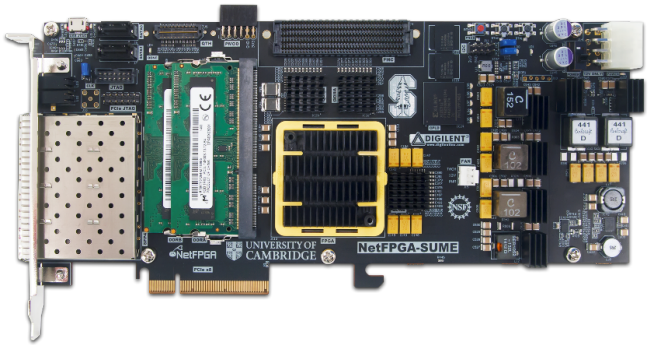
NetFPGA board

Networking Software running on a standard PC



PC with NetFPGA

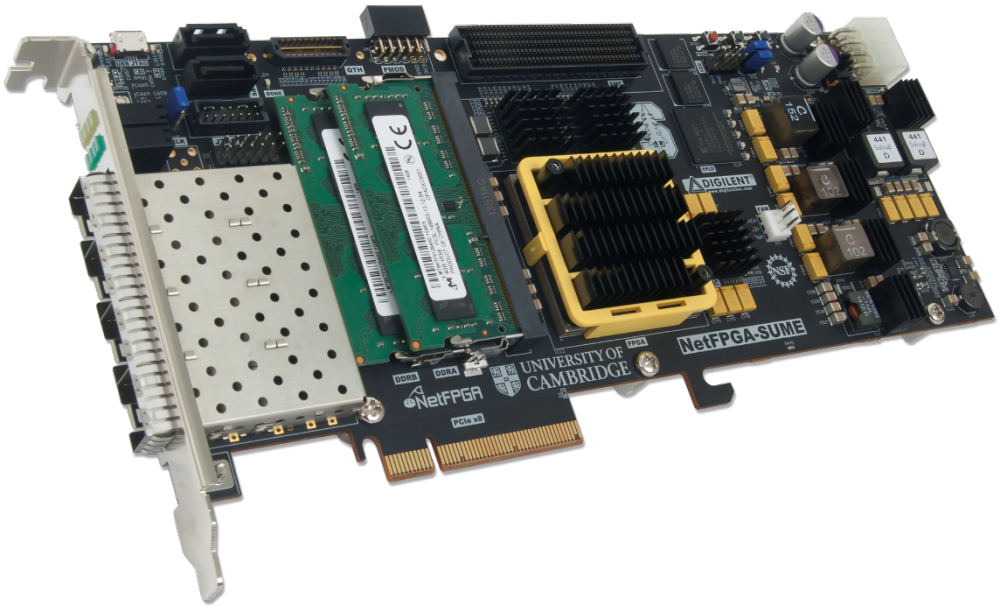
A hardware accelerator built with FPGA driving 1/10/ 100Gb/s network links



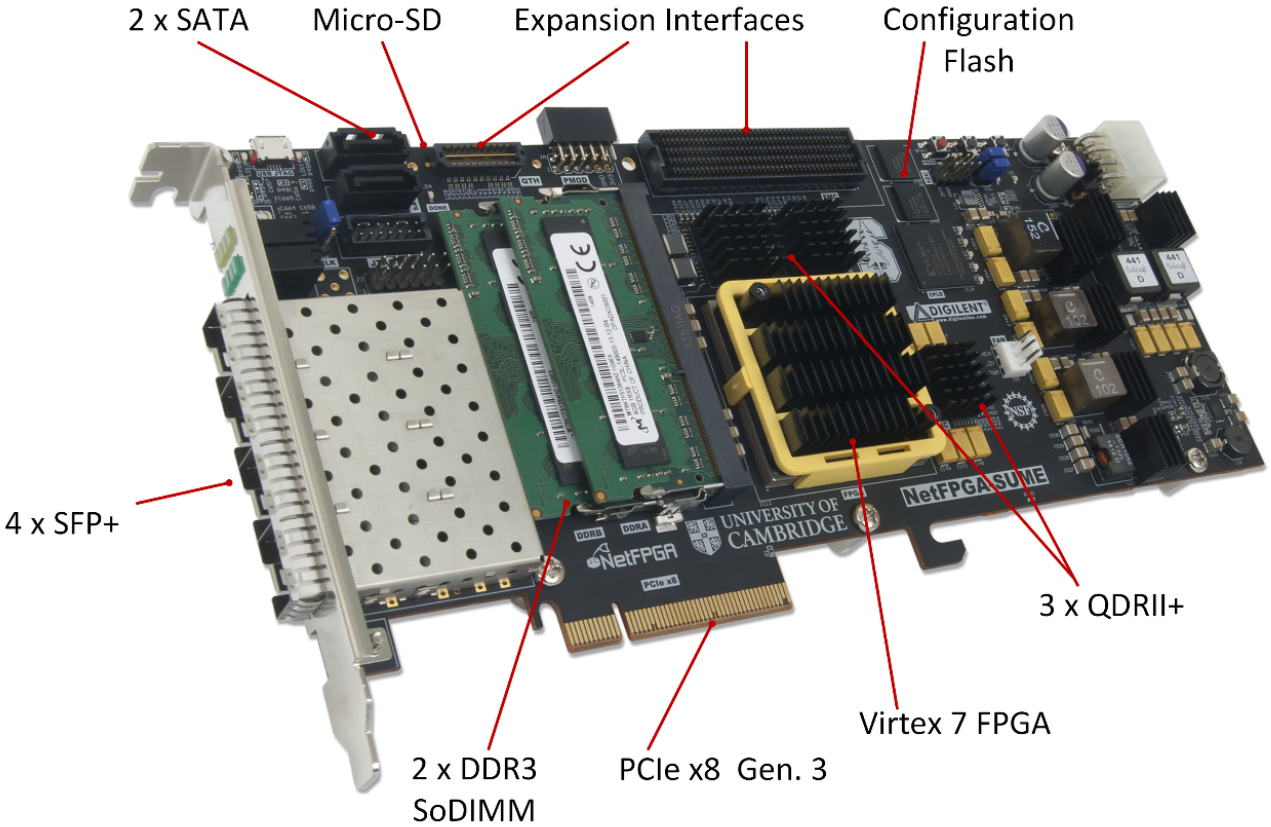
NetFPGA consists of …

Four elements:

- **NetFPGA board**
- **Tools + reference designs**
- **Contributed projects**
- **Community**

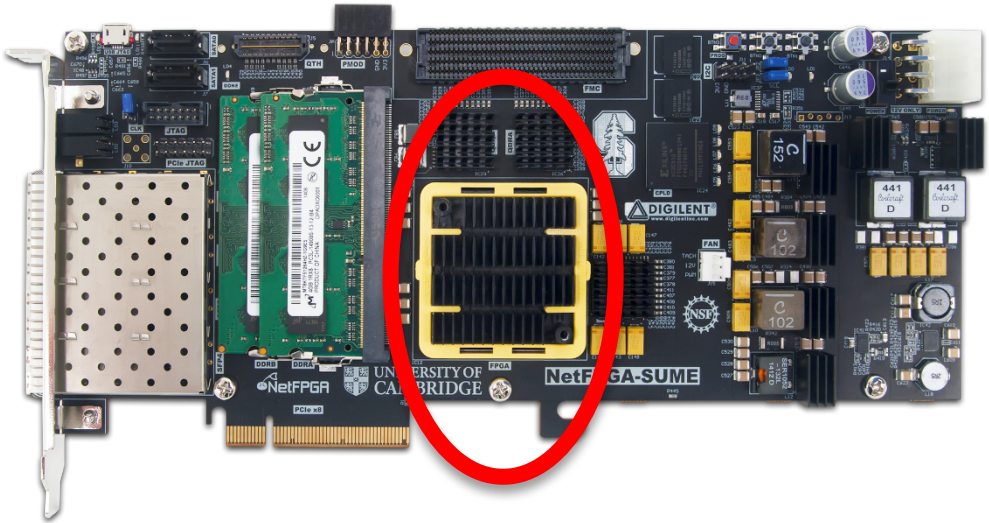


NetFPGA-SUME



Xilinx Virtex 7 690T

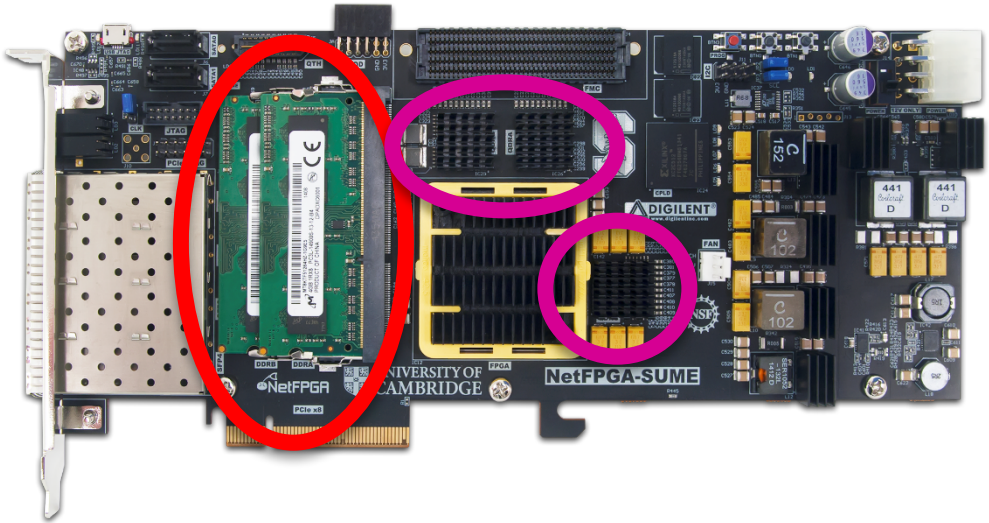
- Optimized for high-performance applications
- 690K Logic Cells
- 52Mb RAM
- 3 PCIe Gen. 3 Hard cores



Memory Interfaces

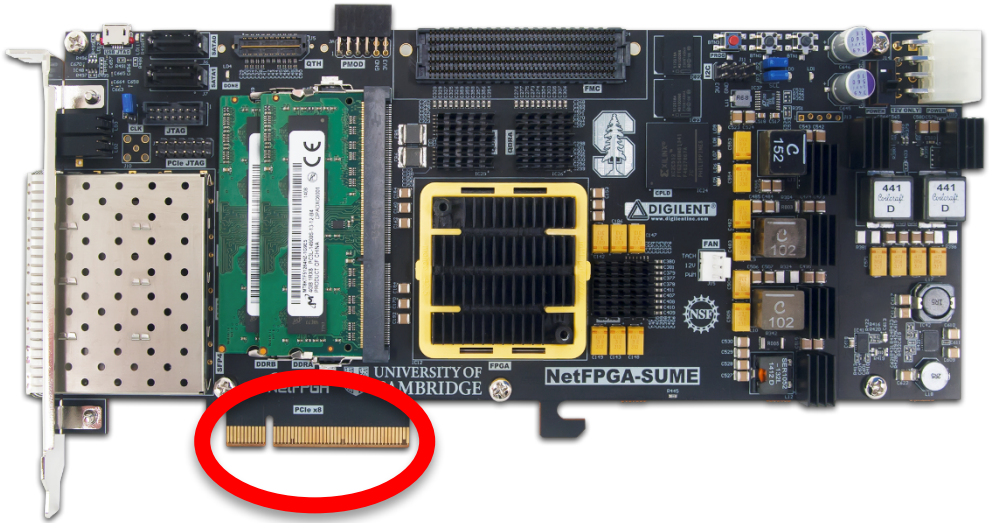
- **DRAM:**
  2 x DDR3 SoDIMM
  1866MT/s, 4GB

- **SRAM:**
  3 x 9MB QDRII+, 500MHz



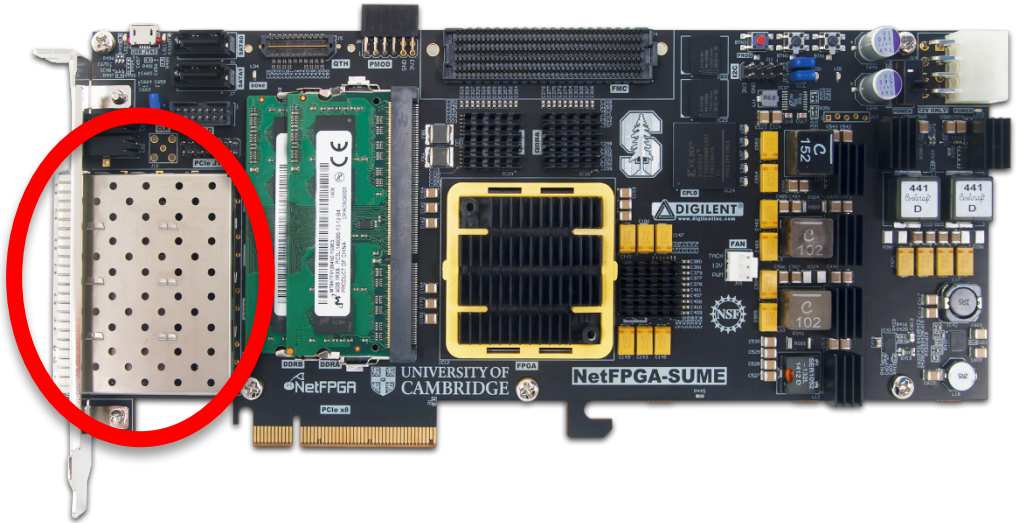
Host Interface

- PCIe Gen. 3
- x8 (only)
- Hardcore IP



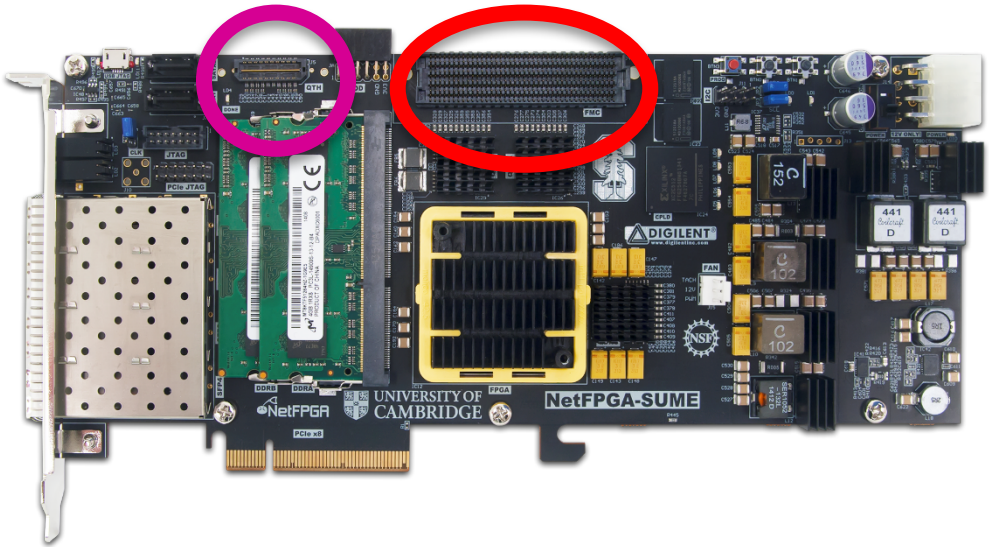
Front Panel Ports

- 4 SFP+ Cages
- Directly connected to the FPGA
- Supports 10GBase-R transceivers (default)
- Also Supports 1000Base-X transceivers and direct attach cables



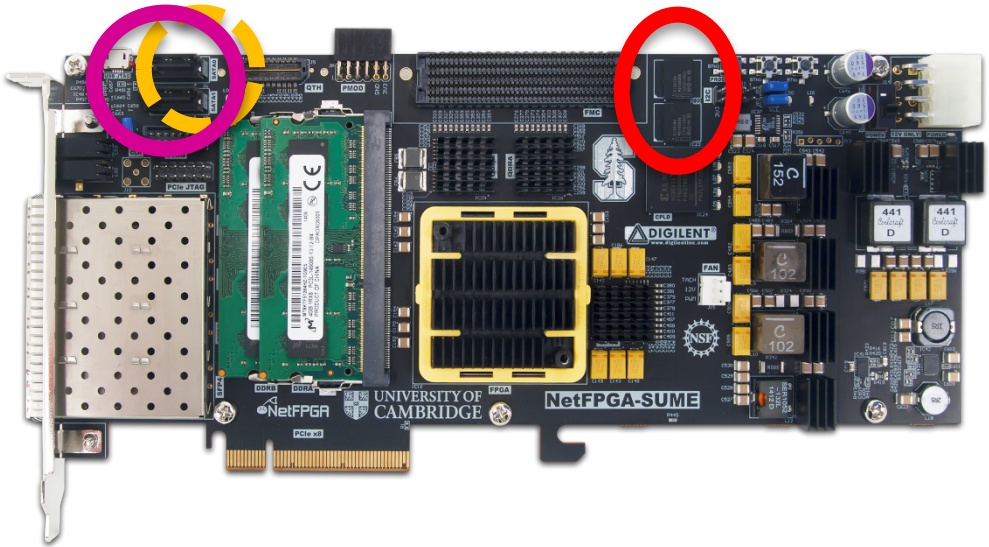
Expansion Interfaces

- **FMC HPC connector**
  - VITA-57 Standard
  - Supports Fabric Mezzanine Cards (FMC)
  - 10 x 12.5Gbps serial links
- **QTH-DP**
  - 8 x 12.5Gbps serial links

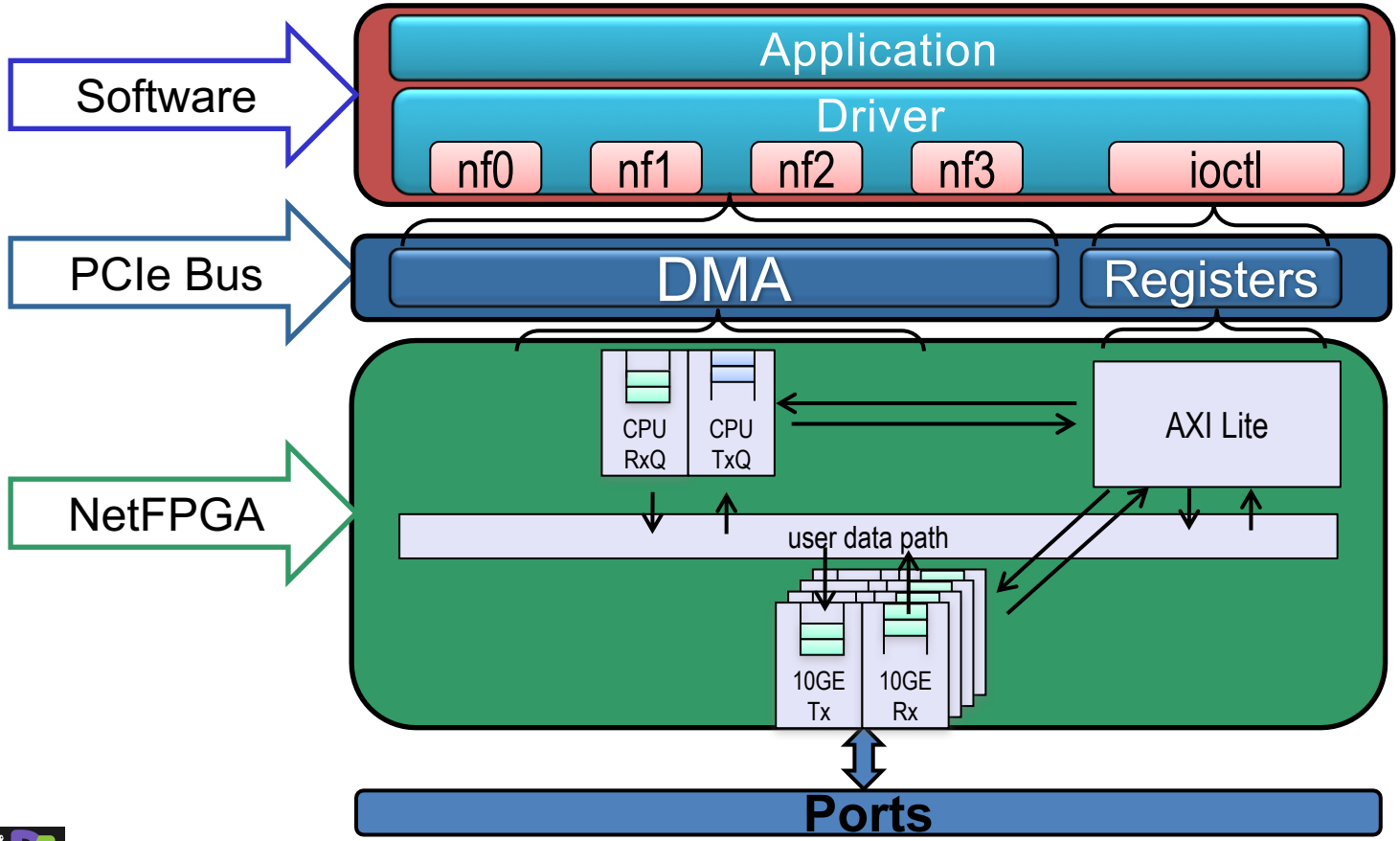


Storage

- 128MB FLASH
- 2 x SATA connectors
- Micro-SD slot
- Enable standalone operation

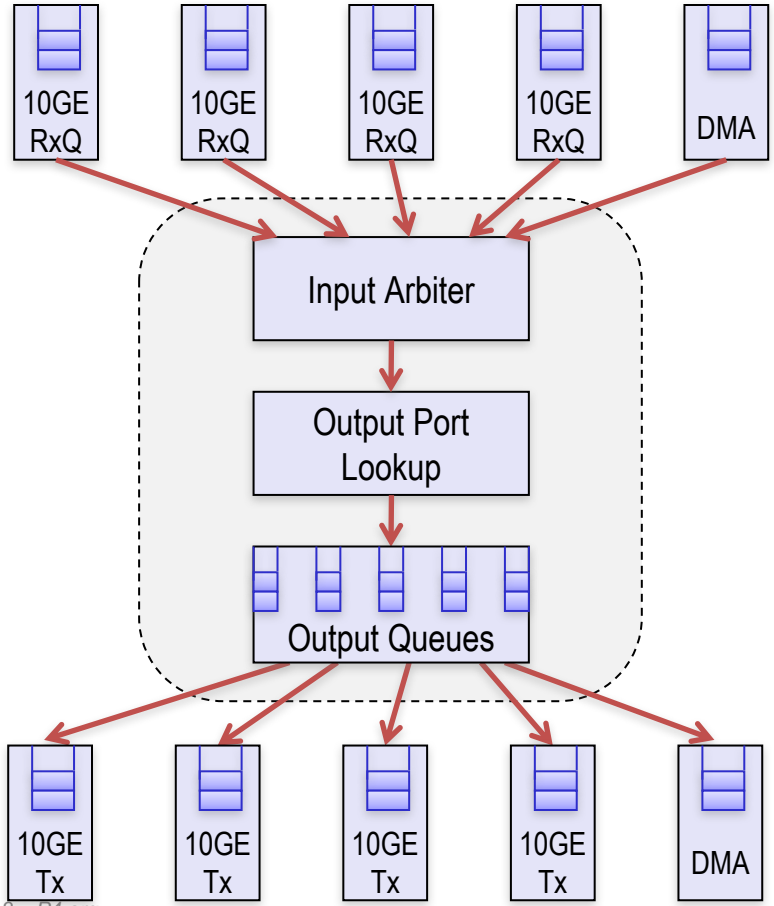


Full System Components



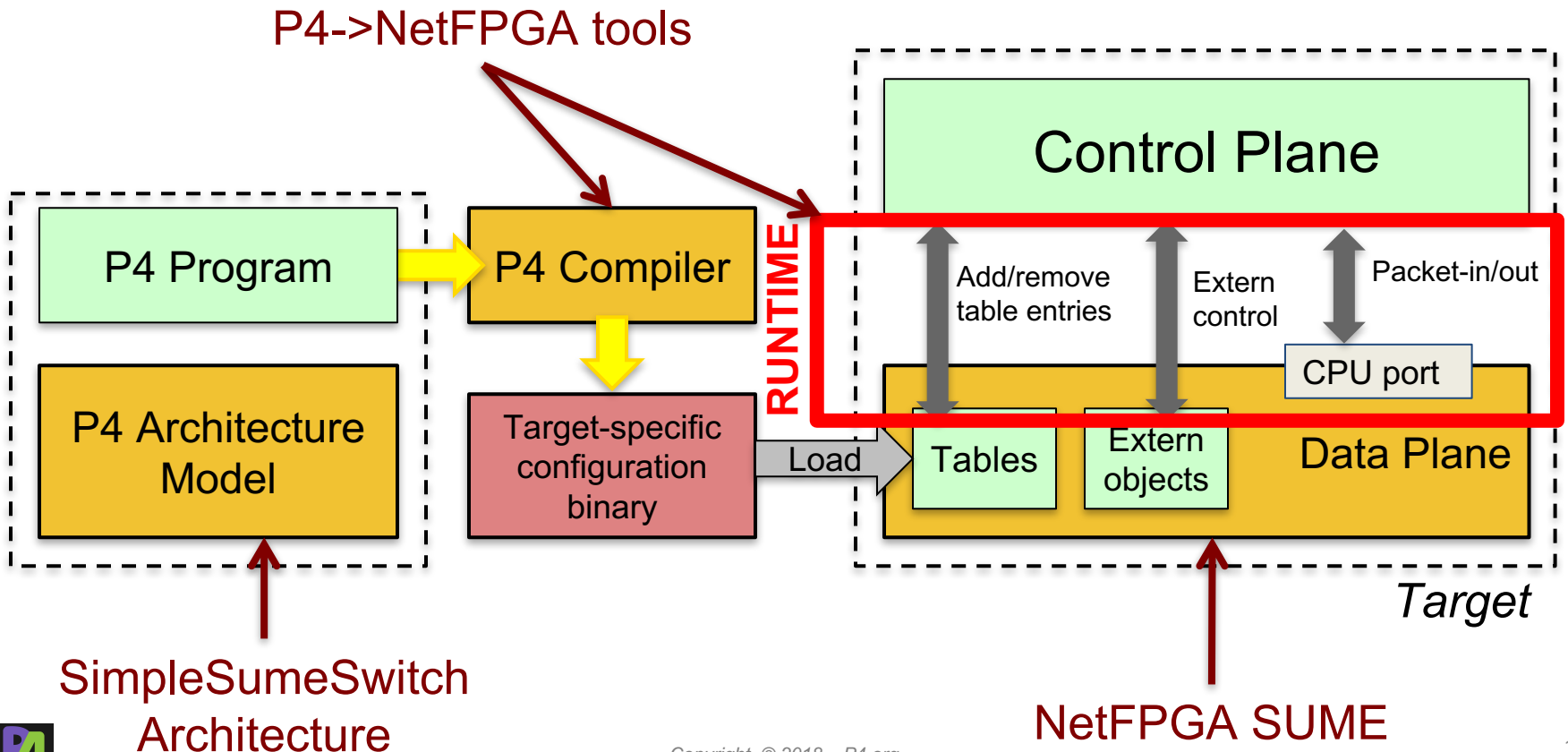
Reference Switch Pipeline

- **Five stages**
  - Input port
  - Input arbitration
  - Forwarding decision and packet modification
  - Output queuing
  - Output port
- **Packet-based module interface**
- **Pluggable design**

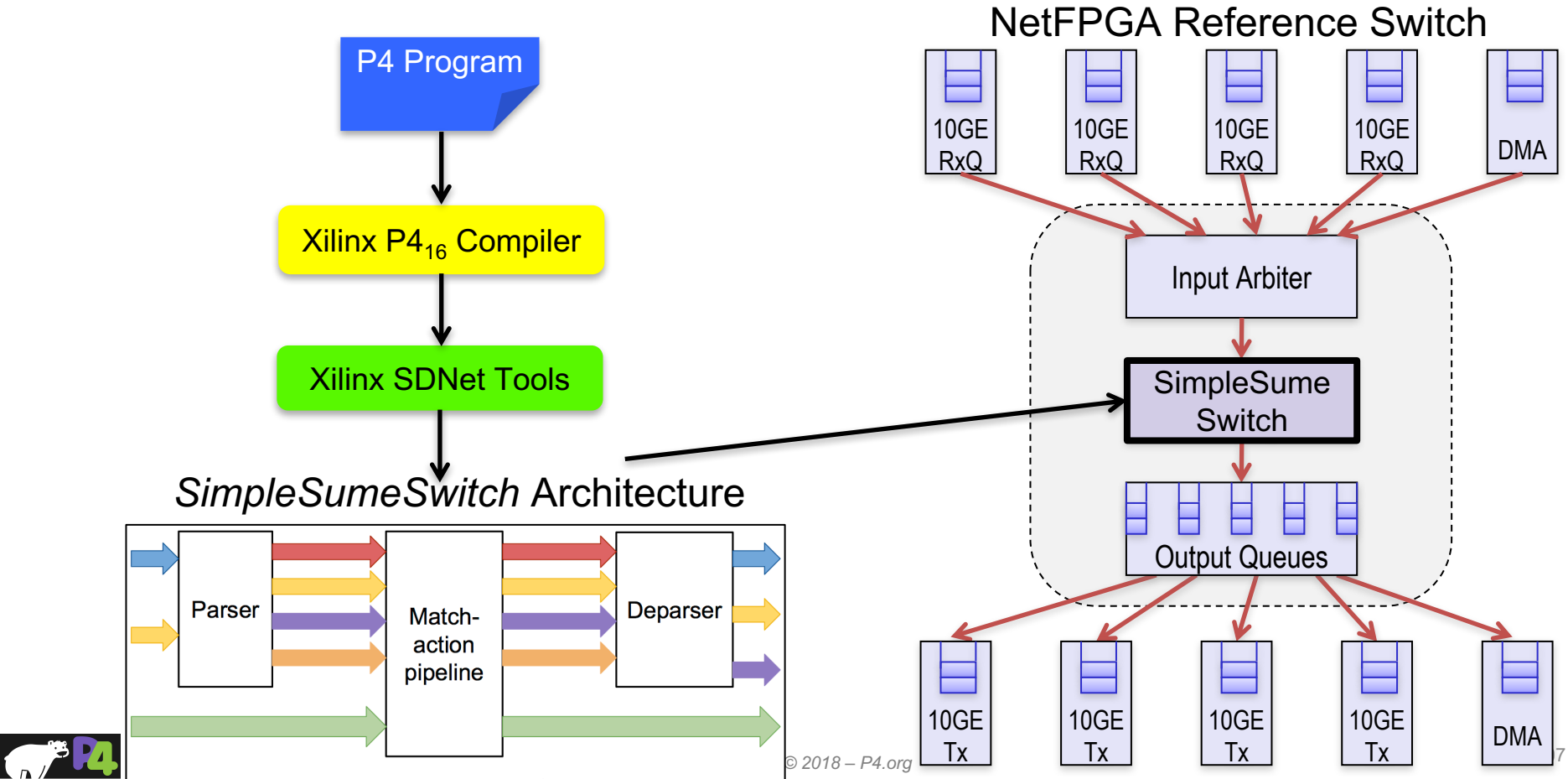


Overview

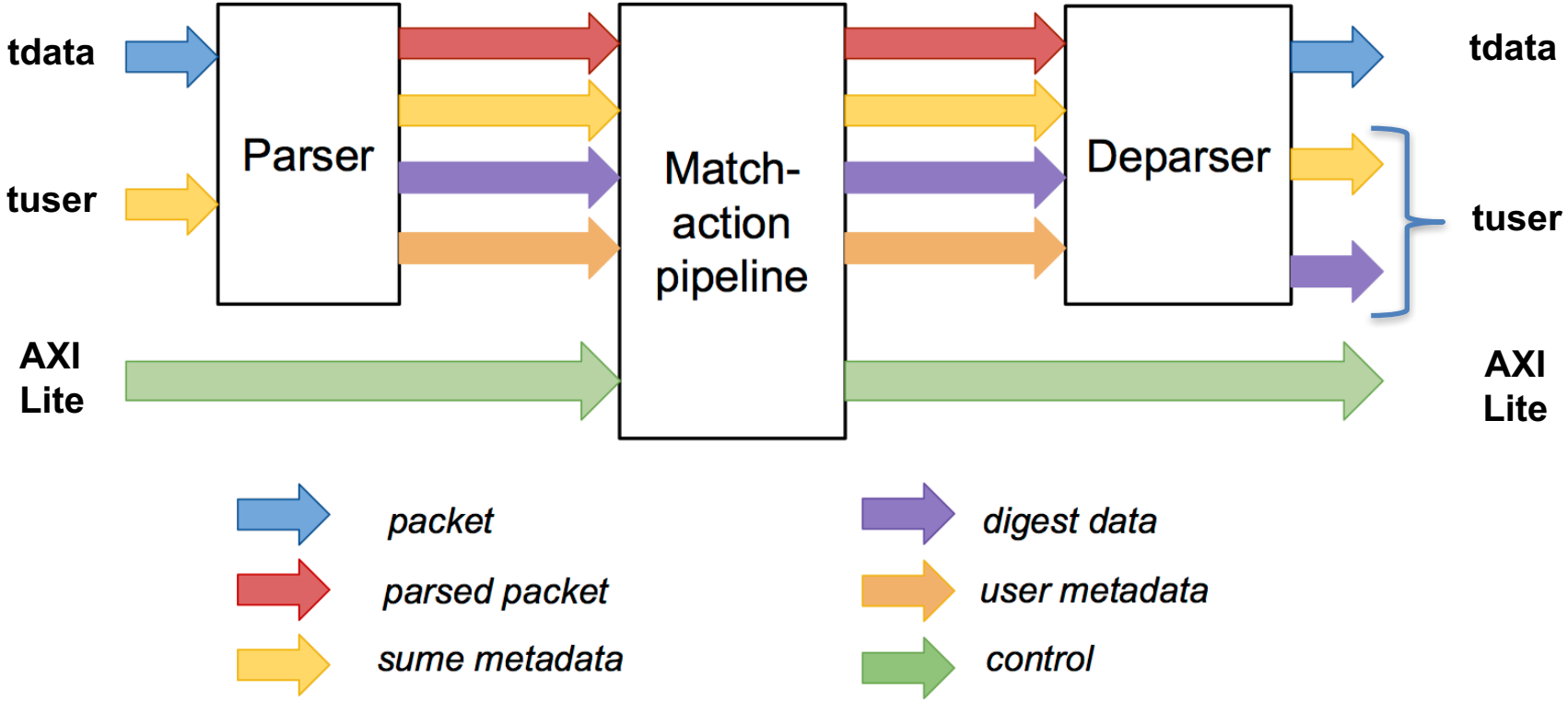
General Process for Programming a P4 Target



P4→NetFPGA Compilation Overview



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;
    bit<8> send_dig_to_cpu; // send digest_data to CPU
    bit<8> dst_port; // one-hot encoded
    bit<8> src_port; // one-hot encoded
    bit<16> pkt_len; // unsigned int
}
```

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

P4→NetFPGA Extern Function library

- **Implement platform specific functions**
  - Black box to P4 program
- **Implemented in HDL**
- **Stateless – reinitialized for each packet**
- **Stateful – keep state between packets**
- **Xilinx Annotations**
  - `@Xilinx_MaxLatency()` – maximum number of clock cycles an extern function needs to complete
  - `@Xilinx_ControlWidth()` – size in bits of the address space to allocate to an extern function

P4→NetFPGA Extern Function library

- HDL modules invoked from within P4 programs
- Stateful Atoms [1]

Atom	Description
R/W	Read or write state
RAW	Read, add to, or overwrite state
PRAW	Predicated version of RAW
ifElseRAW	Two RAWs, one each for when predicate is true or false
Sub	IfElseRAW with stateful subtraction capability

- Stateless Externs

Atom	Description
IP Checksum	Given an IP header, compute IP checksum
LRC	Longitudinal redundancy check, simple hash function
timestamp	Generate timestamp (granularity of 5 ns)

- Add your own!

[1] Sivaraman, Anirudh, et al. "Packet transactions: High-level programming for line-rate switches." *Proceedings of the 2016 ACM SIGCOMM Conference*. ACM, 2016.

Using Atom Externs in P4 – Resetting Counter

Packet processing pseudo code:

```
count[NUM_ENTRIES];

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

Using Atom Externs in P4 – Resetting Counter

```
#define REG_READ    0
#define REG_WRITE   1
#define REG_ADD     2
// count register
@Xilinx_MaxLatency(1)
@Xilinx_ControlWidth(3)
extern void count(in bit<3> index, in bit<32> newVal,
                  in bit<32> incVal,in bit<8> opCode,
                  out bit<32> result);

bit<16> index = pkt.hdr.index;
bit<32> newVal; bit<32> incVal; bit<8> opCode;
if(pkt.hdr.reset == 1) {
    newVal = 0;
    incVal = 0; // not used
    opCode = REG_WRITE;
} else {
    newVal = 0; // not used
    incVal = 1;
    opCode = REG_ADD;
}

bit<32> result; // the new value stored in count reg
count_reg_raw(index, newVal, incVal, opCode, result);
```

- State can be accessed exactly *1 time*
- Using RAW atom here

Instantiate atom

Set metadata for state access

Single state access!

API & Interactive CLI Tool Generation

- **Both Python API and C API**
  - Manipulate tables and stateful elements in P4 switch
  - Used by control-plane program
- **CLI tool**
  - Useful debugging feature
  - Query various compile-time information
  - Interact directly with tables and stateful elements in at run time

P4→NetFPGA Workflow

1. Write P4 program
2. Write externs
3. Write python gen_testdata.py script

All of your effort will go here

4. Compile to Verilog / generate API & CLI tools
5. Run simulations
6. Build bitstream
7. Check implementation results
8. Test the hardware

fail

pass

Get the Tools!

- **Docs: https://github.com/NetFPGA/P4-NetFPGA-public/wiki**
- **Request P4->NetFPGA tools and licenses from Xilinx**
- **NetFPGA SUME Board**
  - (Academic users) Special price request [1]
  - (Industry users) Purchase from Digilent [2]

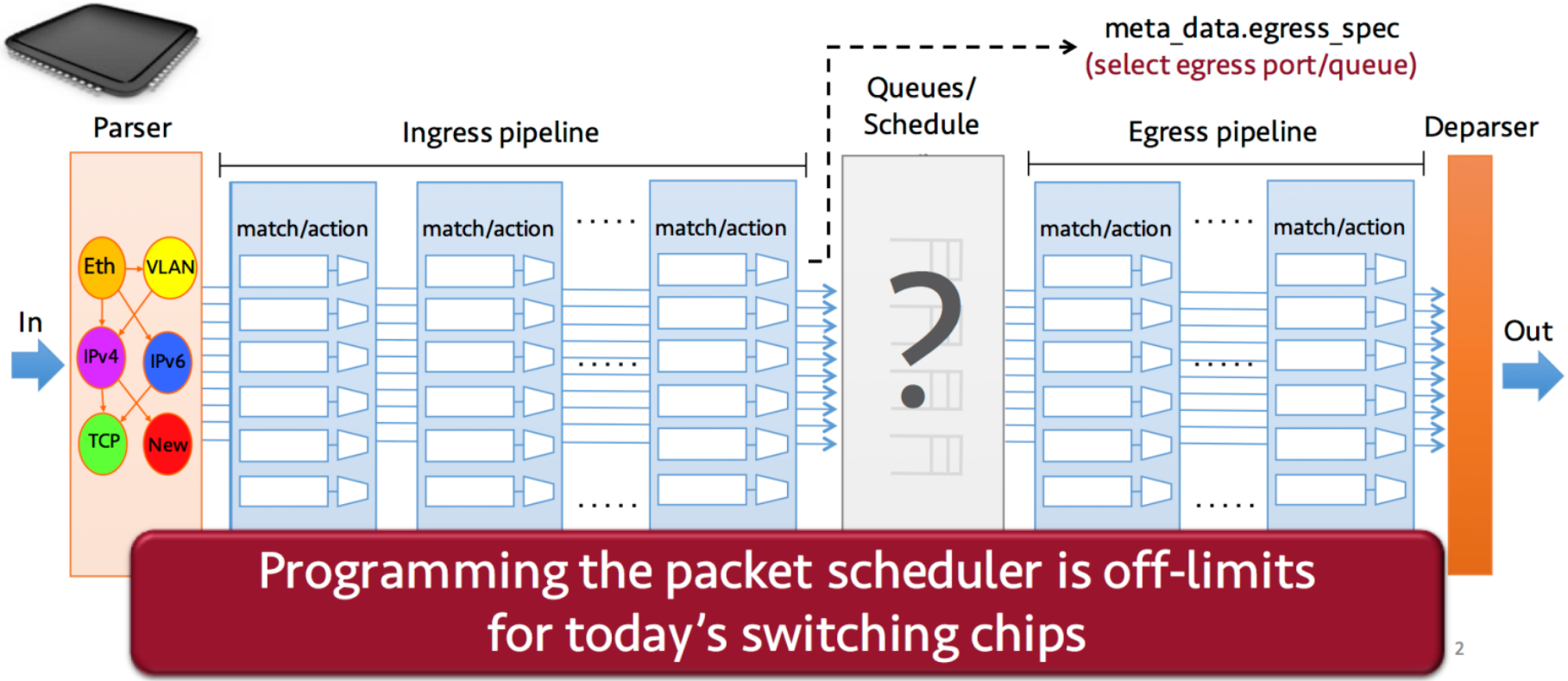
[1] https://netfpga.wufoo.com/forms/netfpga-special-pricing-request/
[2] http://store.digilentinc.com/netfpga-sume-virtex-7-fpga-development-board/

Future research topics

Examples of ongoing P4 Research Topics

- **P4 Infrastructure**
  - **Programmable scheduling**
  - **Programmable target architectures**
  - PacketMod
- **Data-plane Programs**
  - **In-band network telemetry**
  - **Congestion control**
  - Load balancing
- **Networking-Offloading Applications**
  - **Aggregation for MapReduce applications**
  - Key-value caching
  - Consensus

Programmable Scheduling



Sivaraman, Anirudh, et al. "Programmable packet scheduling at line rate." *Proceedings of the 2016 ACM SIGCOMM Conference*. ACM, 2016.

Why scheduler is not programmable ... so far

- **Plenty of scheduling algorithms, but no consensus on right abstractions. Contrast to:**
  - Parse graphs for parsing
  - Match-Action tables for forwarding

- **Scheduler has tight timing requirements**
  - One decision every few ns

What does the Scheduler do?

Decides:

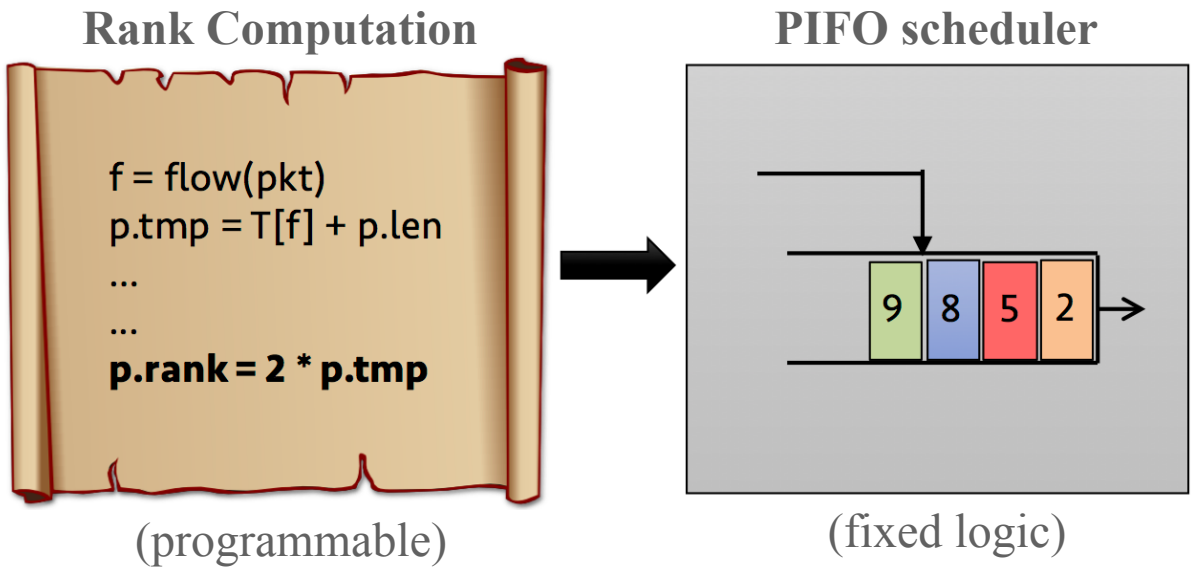
- **In what order are packets sent?**
  - Ex: FCFS, Priorities, WFQ
- **At what time are packets sent?**

Key observation:

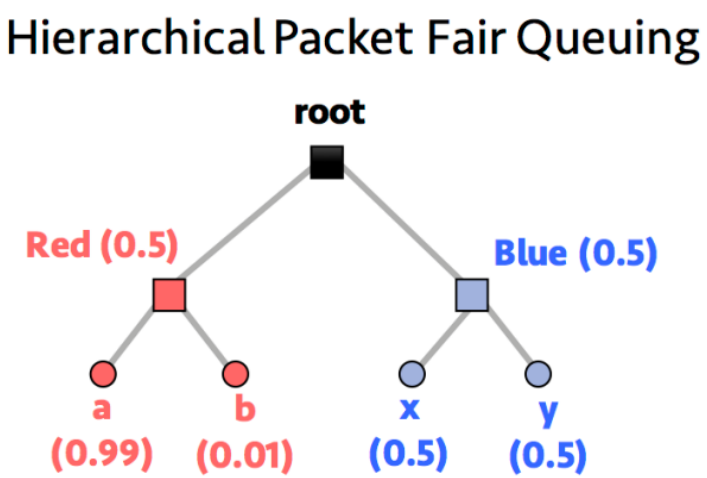
- **For many algorithms, the relative order in which packets are sent does not change with future arrivals**
  - i.e. scheduling order can be determined before enqueue

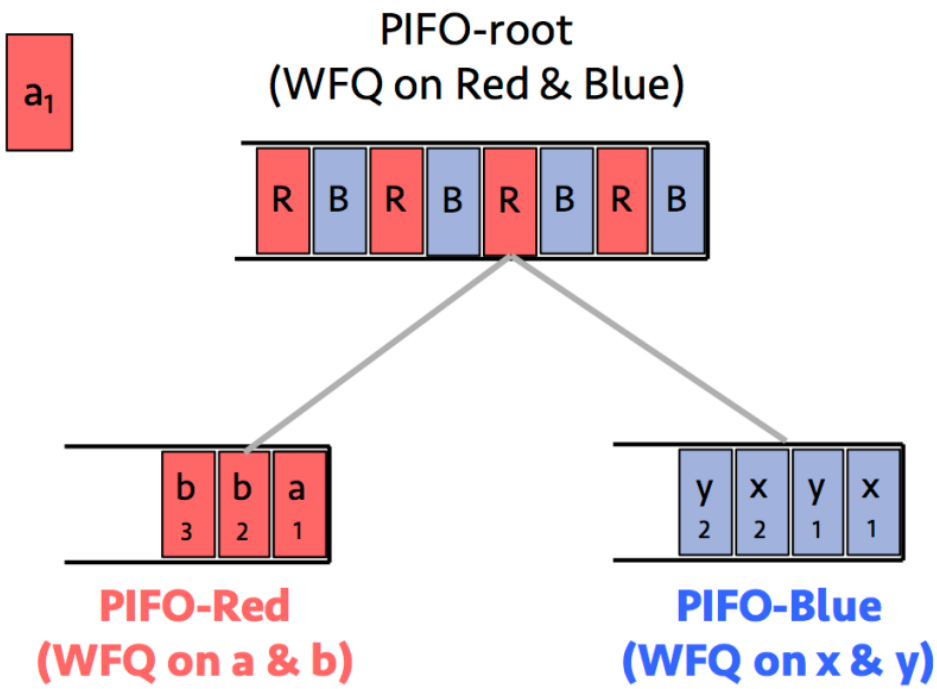
PIFO

- PIFO - proposed abstraction that can be used to implement many scheduling algorithms
- Packets are pushed into an arbitrary location based on computed rank



PIFO Tree





PIFO Remarks

- **Very limited scheduling in modern switching chips**
  - Deficit Round Robin, traffic shaping, strict priorities
- **Scheduling algorithms that can be implemented with PIFO**
  - Weighted Fair Queueing, Token Bucket Filtering, Hierarchical Packet Fair Queueing, Least-Slack Time-First, the Rate Controlled Service Disciplines, and fine-grained priority scheduling (e.g., Shortest Job First)
- **PIFO cannot implement algorithms that require**
  - Changing the scheduling order of all packets of a flow
  - Output rate limiting
- **PIFO implementation feasibility?**

Programmable Target Architectures

Observations:

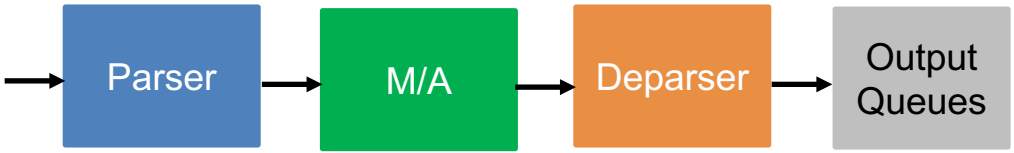
- Current P4 expectation: target architectures are *fixed,* specified in English
- FPGAs can support many different architectures

Idea:

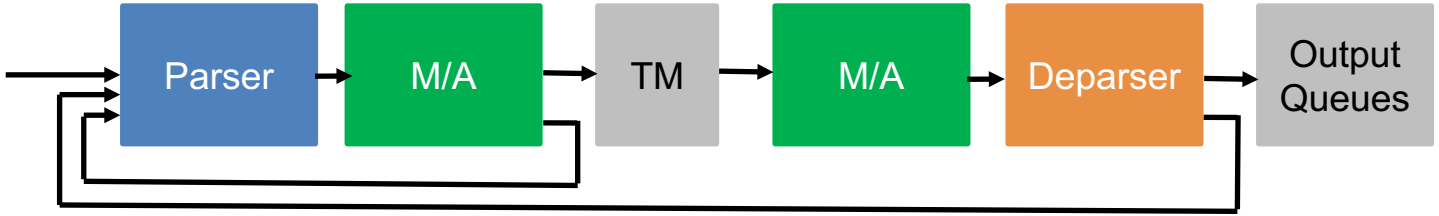
- Extend P4 to allow description of target architectures
  - More precise definition than English description
- Generate implementation on FPGA
- Easily integrate custom modules
- Explore performance tradeoffs of different architectures

Many Possible Architectures…

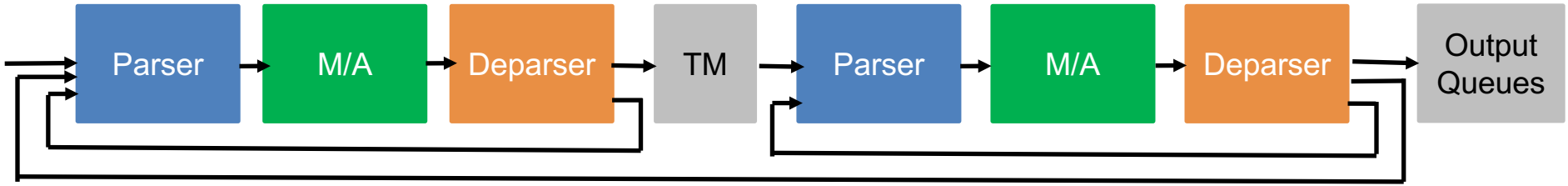
SimpleSumeSwitch



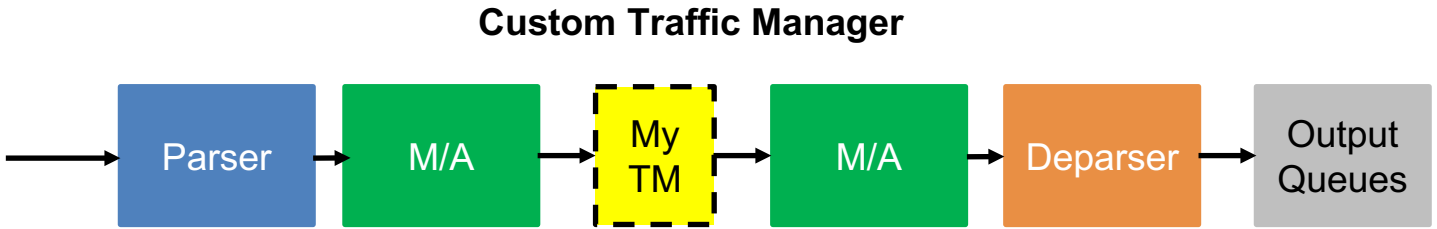
V1 Model

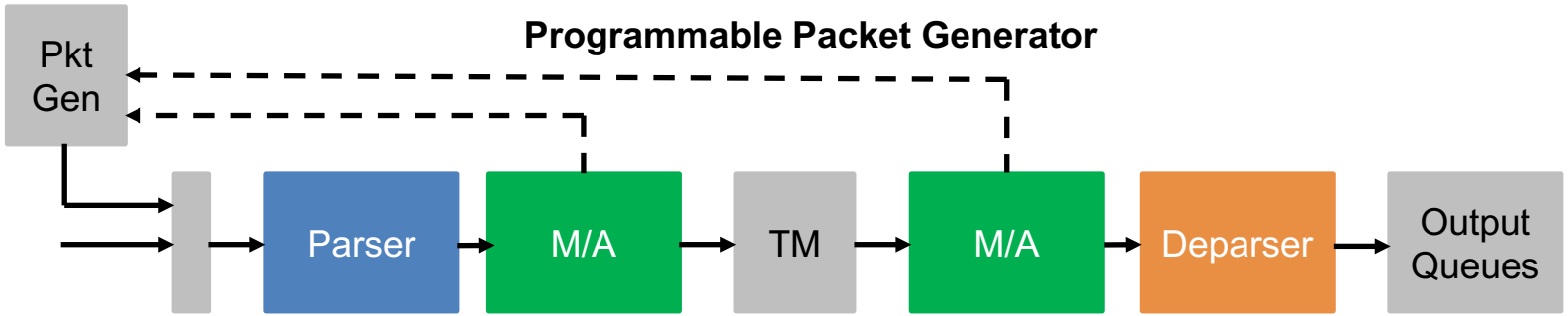


Portable Switch Architecture



Many Possible Architectures…





Programmable Target Architectures

```
package SimpleSumeSwitch<H, M, D>(
                  Parser<H, M, D> TopParser,
                  Pipe<H, M, D> TopPipe,
                  Deparser<H, M, D> TopDeparser) {

 // Top level I/O
 packet_in instream;
 inout sume_metadata_t sume_metadata;
 out D digest_data;
 packet_out outstream;

 // Connectivity of the architecture
 connections {
   // TopParser input connections
   TopParser.b                  = instream;
   TopParser.sume_metadata      = sume_metadata;

   // TopPipe <-- TopParser
   TopPipe.p                = TopParser.p;
   TopPipe.user_metadata    = TopParser.user_metadata;
   TopPipe.digest_data      = TopParser.digest_data;
   TopPipe.sume_metadata    = TopParser.sume_metadata;
```

```
   // TopDeparser <-- TopPipe
   TopDeparser.p                = TopPipe.p;
   TopDeparser.user_metadata    = TopPipe.user_metadata;
   TopDeparser.digest_data      = TopPipe.digest_data;
   TopDeparser.sume_metadata    = TopPipe.sume_metadata;

   // TopDeparser output connections
   digest_data      = TopDeparser.digest_data;
   sume_metadata    = TopDeparser.sume_metadata;
   outstream        = TopDeparser.b;
 }
}
```

Workflow

- **Two Actors: (1) Target Architecture Designer, (2) P4 Programmer**

Provides:
- $P4_+$ architecture declaration

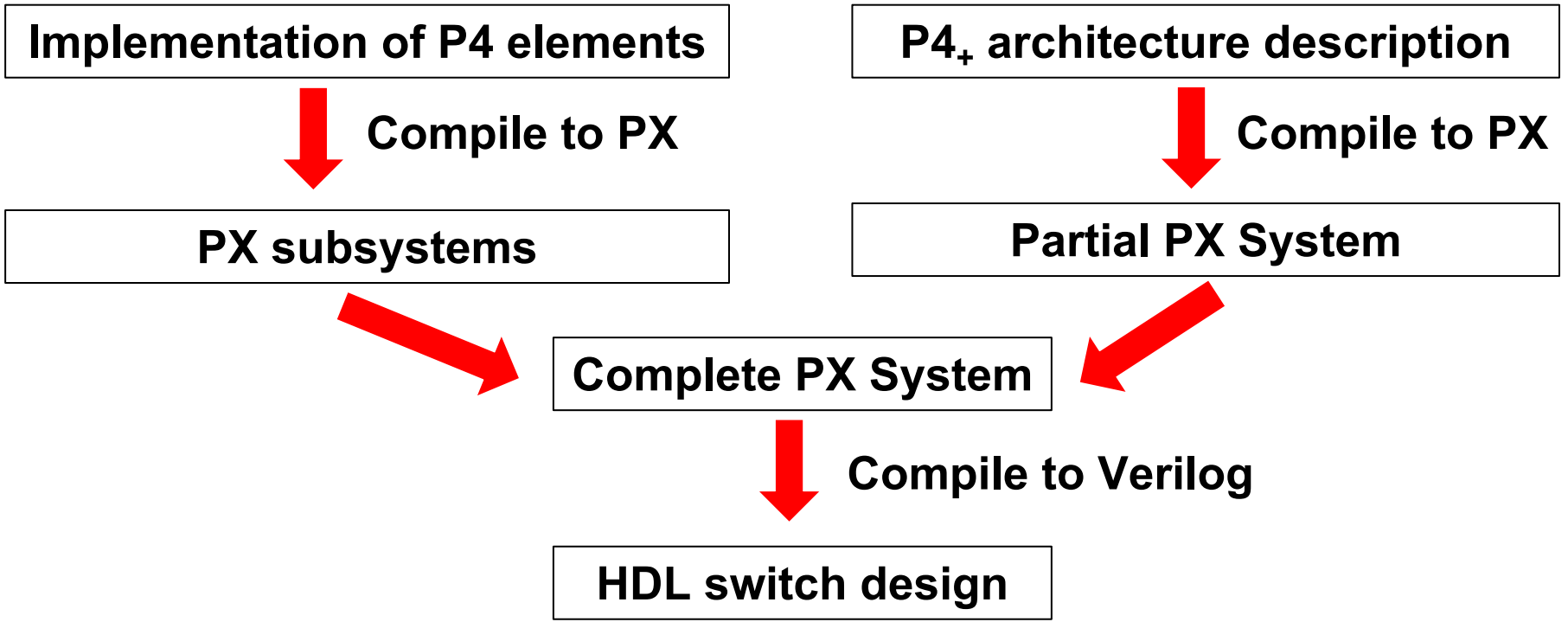
Implements:
- non-P4 elements
- externs

in target architecture

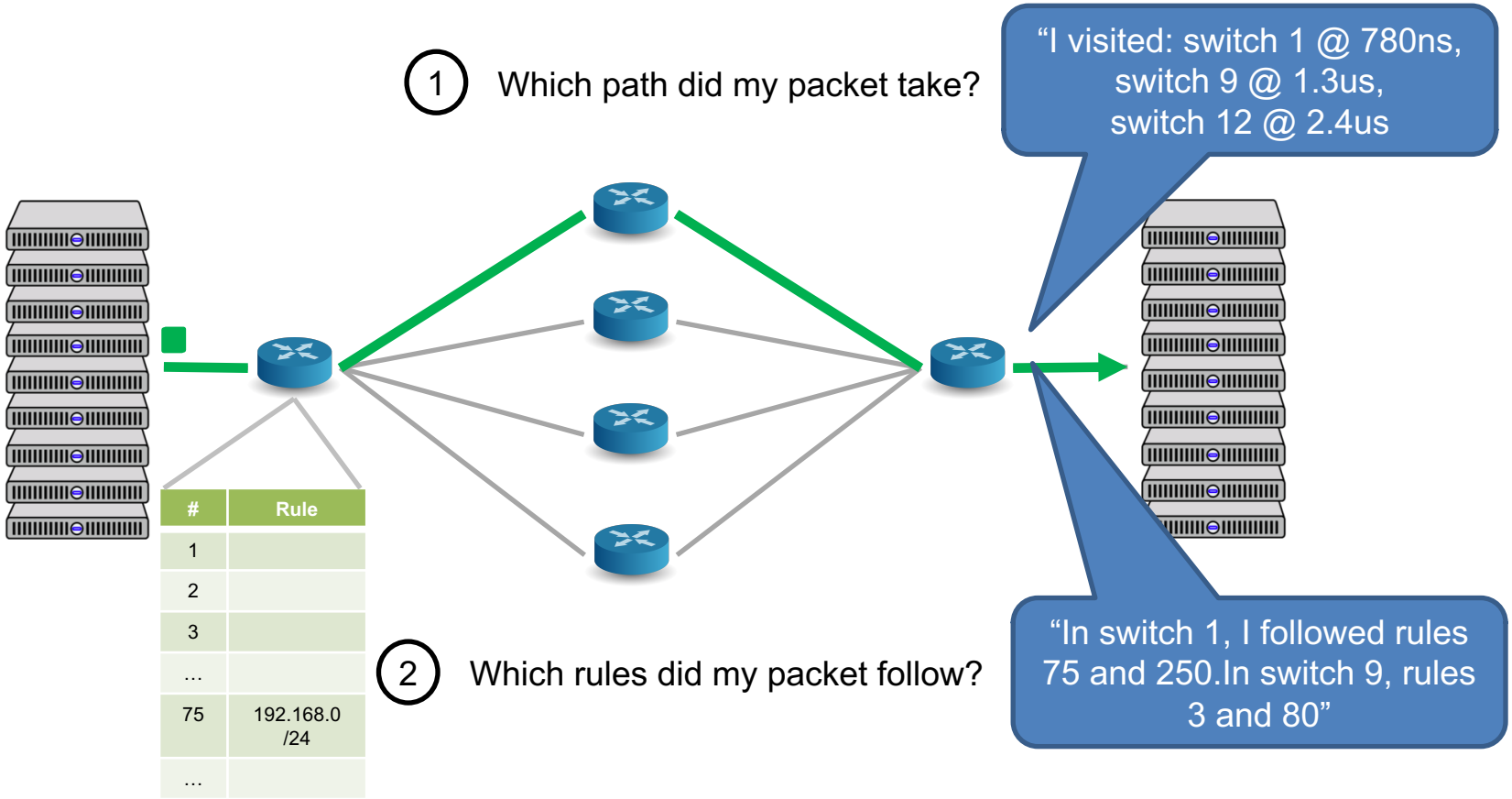
- **Someone who is more familiar with FPGA development**

Workflow

- Two Actors: (1) Target Architecture Designer, (2) P4 Programmer



In-band Network Telemetry (INT)

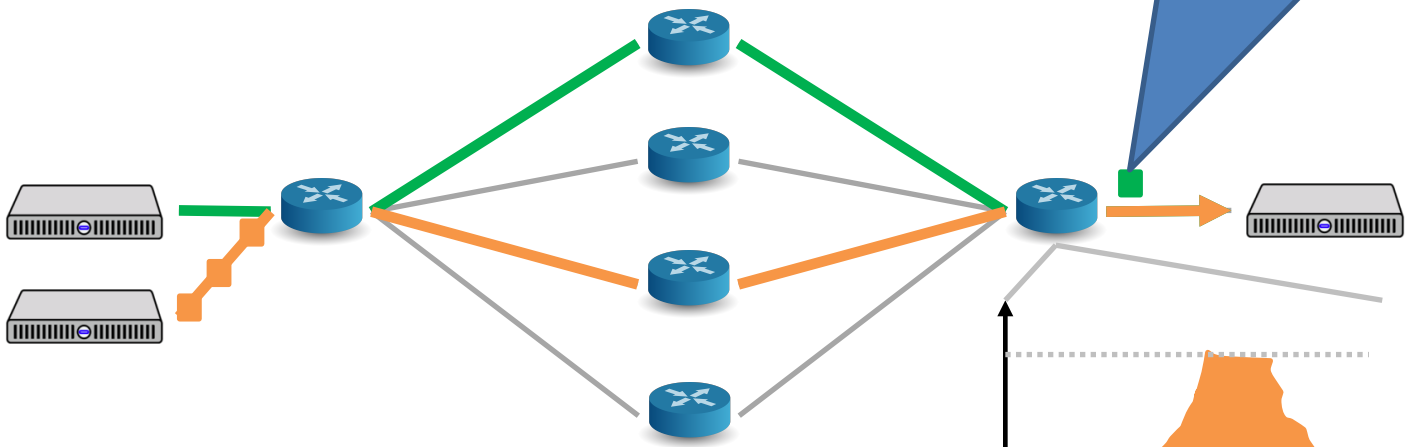


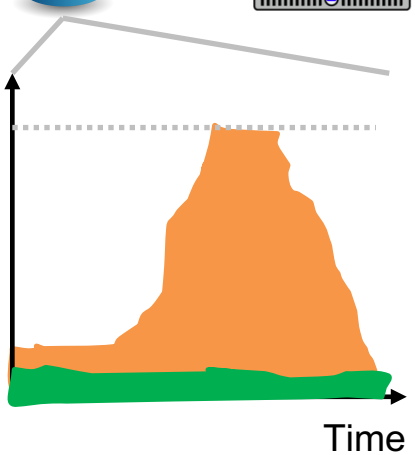
INT Slides courtesy of Nick McKeown

In-band Network Telemetry (INT)

3 How long did my packet queue at each s

"Delay: 100ns, 200ns, 19740ns"





4 Who did my packet share the queue with?

INT Slides courtesy of Nick McKeown

In-band Network Telemetry (INT)

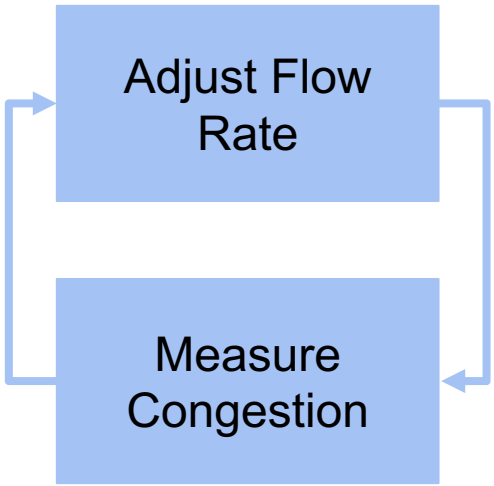
1. Which path did my packet take?
2. Which rules did my packet follow?
3. How long did my packet queue at each switch?
4. Who did my packet share the queue with?

No need to add a single additional packet!

INT Slides courtesy of Nick McKeown

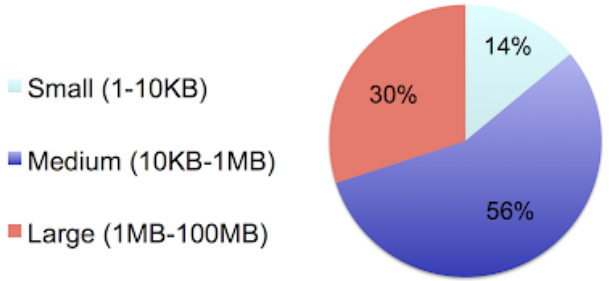
Congestion Control

Reactive Congestion Control



- No use of explicit information about traffic matrix
- Can only react and move in right direction
- Reactive techniques are slow to converge (10s-100s of RTTs)

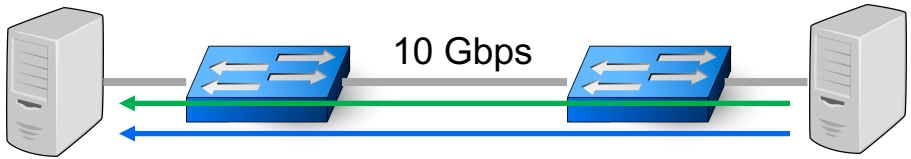
Fraction of Total Flows in Bing Workload

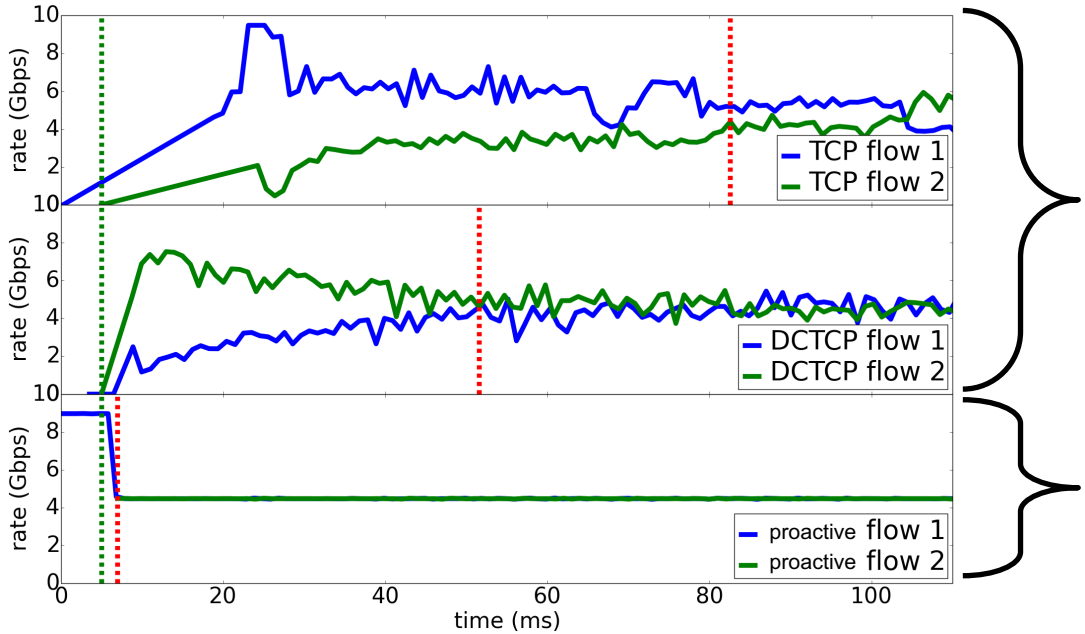


	Typical Flow Completion Times
10 Gb/s	70-80 RTTs
40 Gb/s	17-20 RTTs
100 Gb/s	7-8 RTTs

- Typical flows will finish in just a few RTTs as we move towards higher link speeds

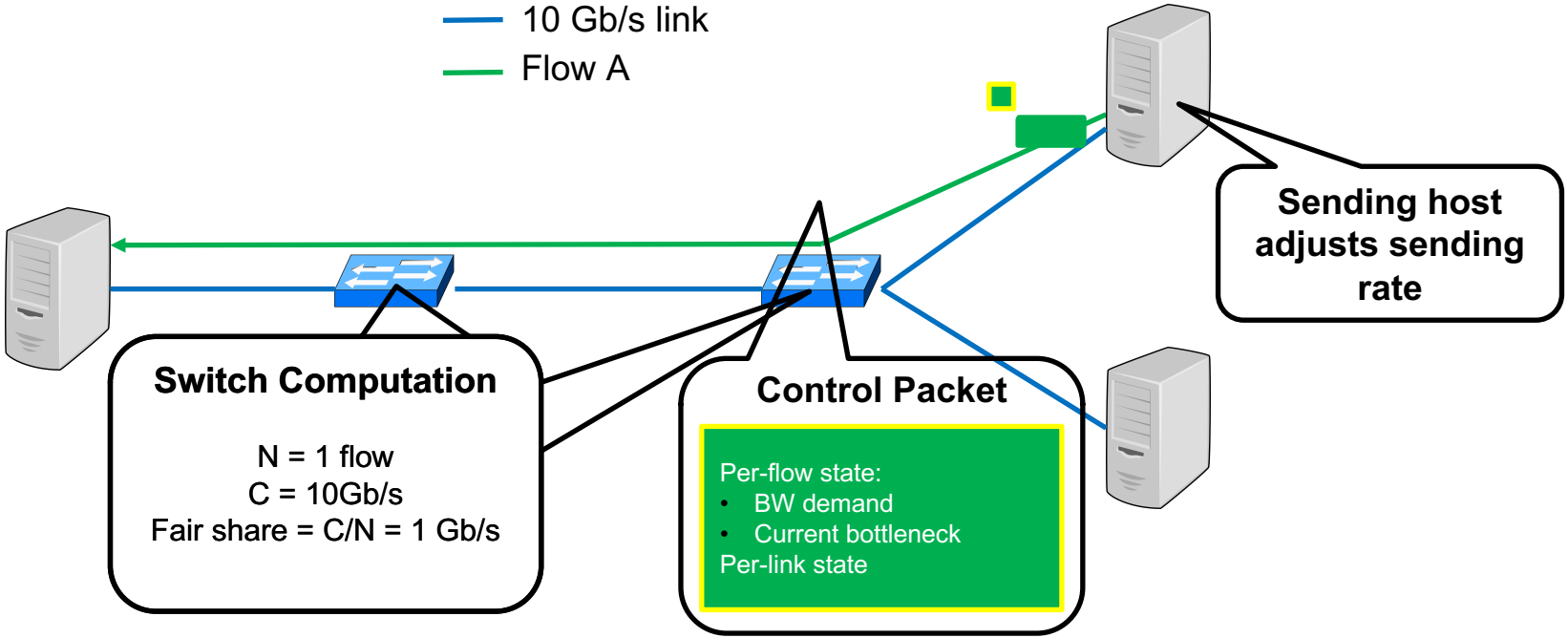
Proactive Congestion Control



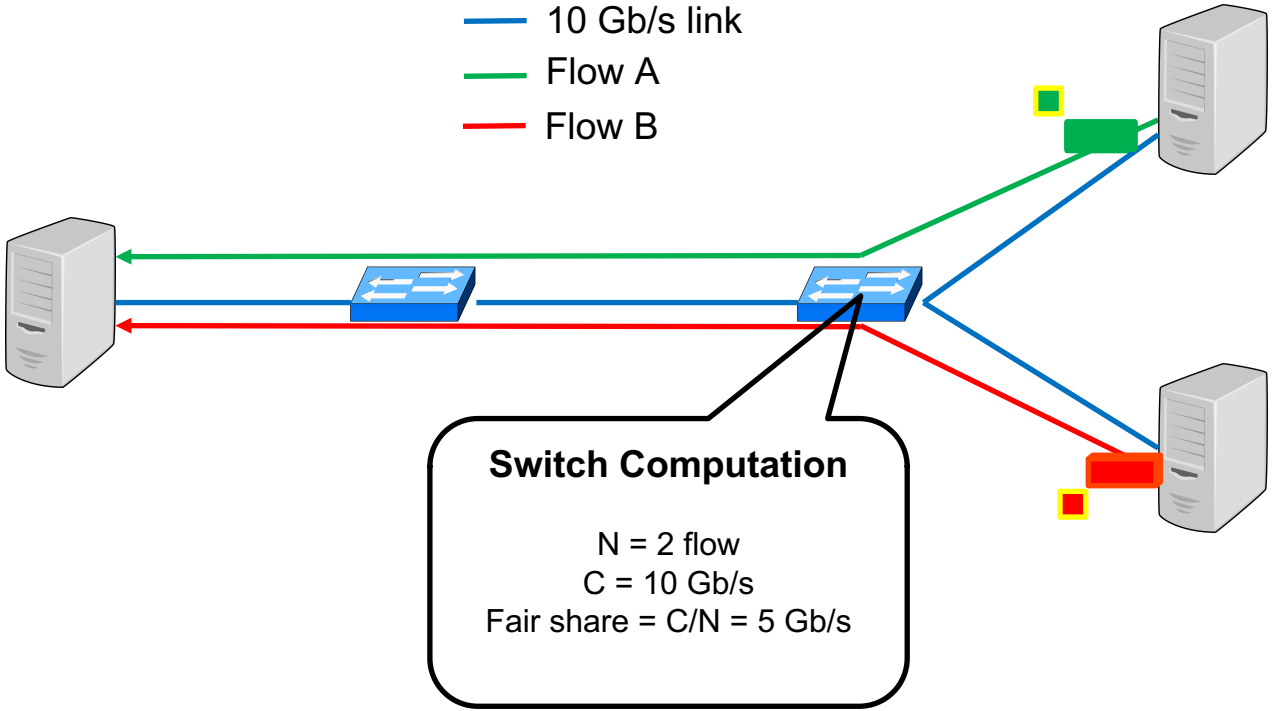


- **Proactive techniques converge much more quickly than reactive**
- **Faster convergence times lead to lower flow completion times**

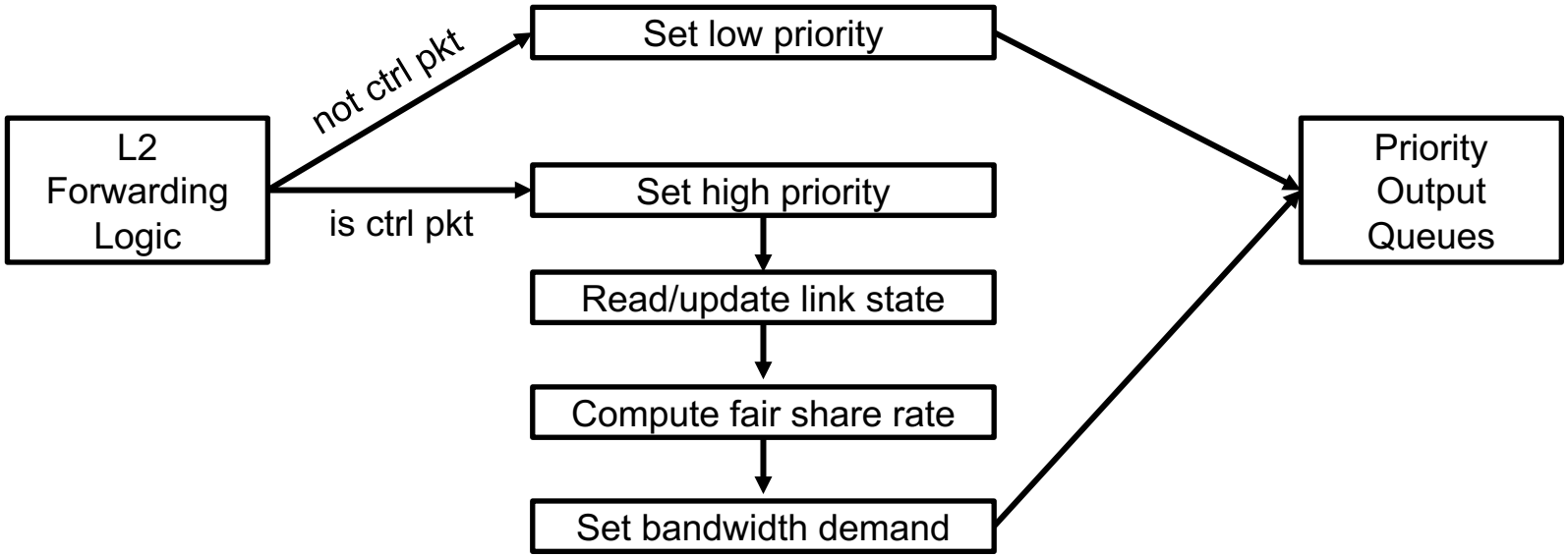
An example proactive scheme



An example proactive scheme



Proactive Algorithm in P4



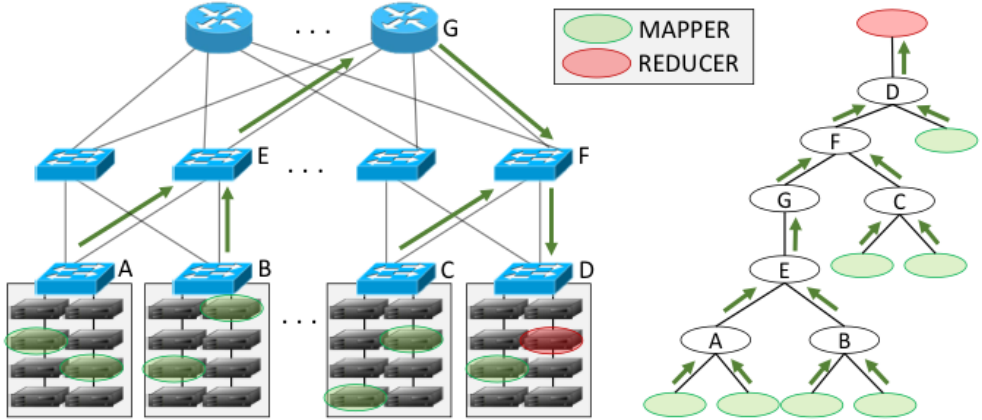
In-Network Computation

- **Programmable data plane hardware → opportunity to reconsider division of computation**
- ***What kinds of computation should be delegated to network?***
- **Network computations are constrained:**
  - *Limited memory size* (10's of MB of SRAM)
  - *Limited set of actions* (simple arithmetic, hashing, table lookups)
  - *Few operations per packet* (10's of ns to process each packet)
- **Goals:**
  - Reduce: application runtime, load on servers, network congestion
  - Increase: application scalability

Sapio, Amedeo, et al. "In-Network Computation is a Dumb Idea Whose Time Has Come." *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*. ACM, 2017.

In-Network Aggregation

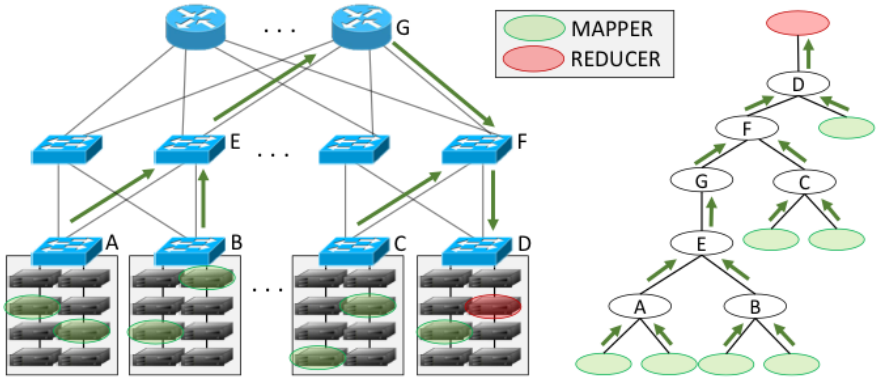
- **Aggregate data at intermediate network nodes to reduce network traffic**
- **Simple arithmetic operations at switches**
- **Widely applicable to many distributed applications**
  - Machine learning training
  - Graph analytics
  - MapReduce applications



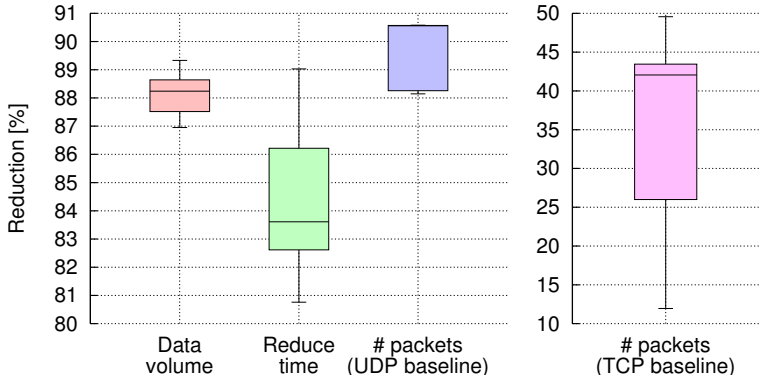
In-Network Aggregation

- **Network controller is informed of MapReduce job**
  - Configures switches in aggregation tree to perform aggregation
- **Significant network traffic reduction → reduced run time**
- **How to make robust to loss? Encryption?**

Aggregation Tree



Reduction Results



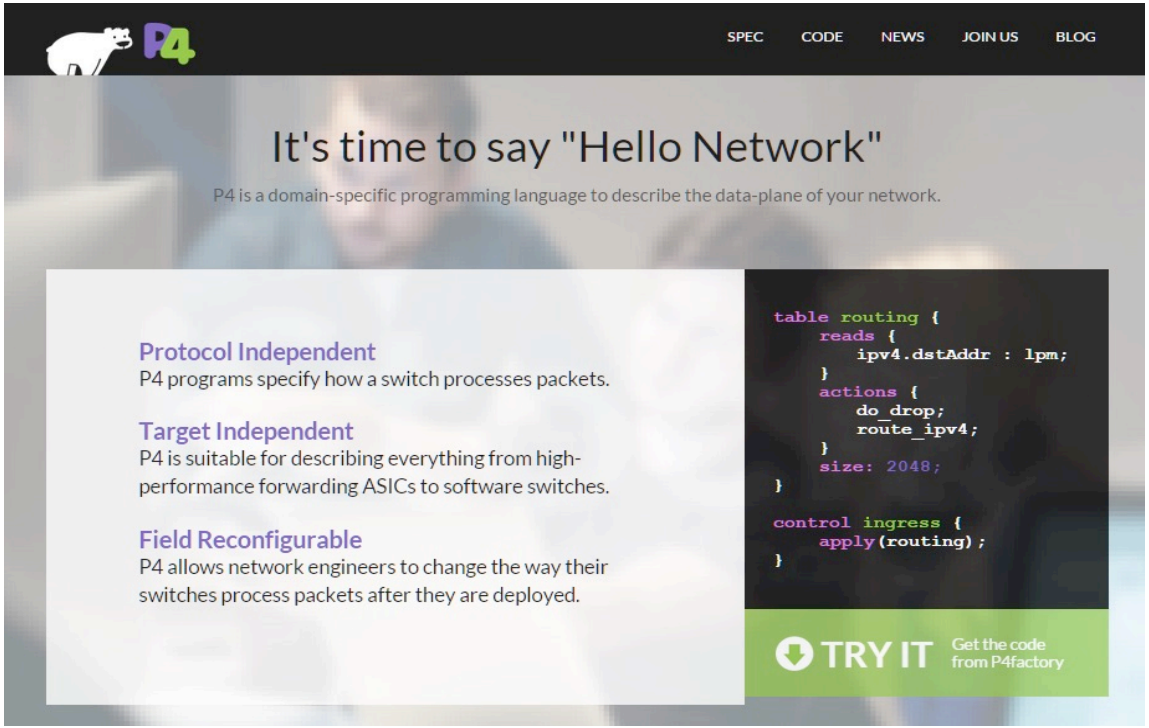
Two relevant papers at FPGA 2018

16:50 Monday	**P4-compatible High-level Synthesis of Low Latency 100 Gb/s Streaming Packet Parsers in FPGAs** **(short paper)**
	Jeferson Santiago da Silva; François-Raymond Boyer; J.M. Pierre Langlois (Polytechnique Montreal)

15:15 Tuesday	**Configurable FPGA Packet Parser for Terabit Networks with Guaranteed Wire-Speed Throughput**
	Jakub Cabal (1); Pavel Benáček (1); Lukáš Kekely(1); Michal Kekely (2); Viktor Puš (2); Jan Kořenek (3) (1 CESNET a.l.e.; 2 Netcope Technologies ; 3 Brno Unive of Tech)

The P4 Language Consortium

- http://p4.org
- Consortium of academic and industry members
- Open source, evolving, domain-specific language
- Permissive Apache license, code on GitHub today
- Membership is free: contributions are welcome
- Independent, set up as a California nonprofit



P4.org current membership

Original P4 Paper Authors:
BAREFOOT NETWORKS, Google, intel, Microsoft, PRINCETON UNIVERSITY, Stanford University

Operators/ End Users
Alibaba Group 阿里巴巴集团, Baidu 百度, FOX, Goldman Sachs, at&t, COMCAST, Google, kt, Microsoft, Tencent 腾讯, SK telecom

Systems
ARISTA, BROCADE, CISCO, DAHO Networks, CORSA, DELL, Hewlett Packard Enterprise, HUAWEI, Inventec, INTRACOM TELECOM, juniper NETWORKS, mantis, NETBERG, NetBRIC, NoviFlow SDN made smarter, OpenxNet, ZTE, XENTECH

Targets
AEPONYX Moving the Cloud at the Speed of Light™, Atomic Rules, BAREFOOT NETWORKS, CAVIUM, EZCHIP, intel, NETCOPE TECHNOLOGIES, Mellanox TECHNOLOGIES, BROADCOM connecting everything, centec networks, freescale, Innovium, MoSys, NETRONOME, PLUMgrid, vmware, MARVELL, STORDIS The High Performance Distributor, XILINX

Solutions/ Services
EstiNet, happiest minds, HGLOBAL HITECH, SDNLAB 专注网络创新技术, FLOW RESEARCH

Academia/ Research
Bii 天地互连, Cornell University Founded A.D. 1865, 國立交通大學 National Chiao Tung University, KOREA UNIVERSITY, PRINCETON UNIVERSITY, salzburgresearch, uni.lu UNIVERSITÉ DU LUXEMBOURG, Università della Svizzera italiana, Eötvös Loránd University, POLITECNICO MILANO 1863, POLYTECHNIQUE MONTRÉAL, Stanford University, UFRGS UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL, VirginiaTech Invent the Future

The end …

- … over to you now