Programmable Networking
P4 for All

Rinku Shah[1], Assistant Professor, IIIT Delhi
Neeraj Kumar Yadav[1], Research Scholar, IIIT Delhi
Keshav Gambhir[2], Microsoft India
Harish S A[3], Research Scholar (PMRF), IIT Hyderabad

[1] IIIT Delhi
India

[2] Microsoft
India

[3] IIT Hyderabad
India

Brief Bio

Rinku Shah

Assistant Professor, IIIT Delhi

rinku@iiitd.ac.in

https://faculty.iiitd.ac.in/~rinku

Neeraj Kumar Yadav

Research scholar, IIIT Delhi

neerajy@iiitd.ac.in

Keshav Gambhir

Microsoft India

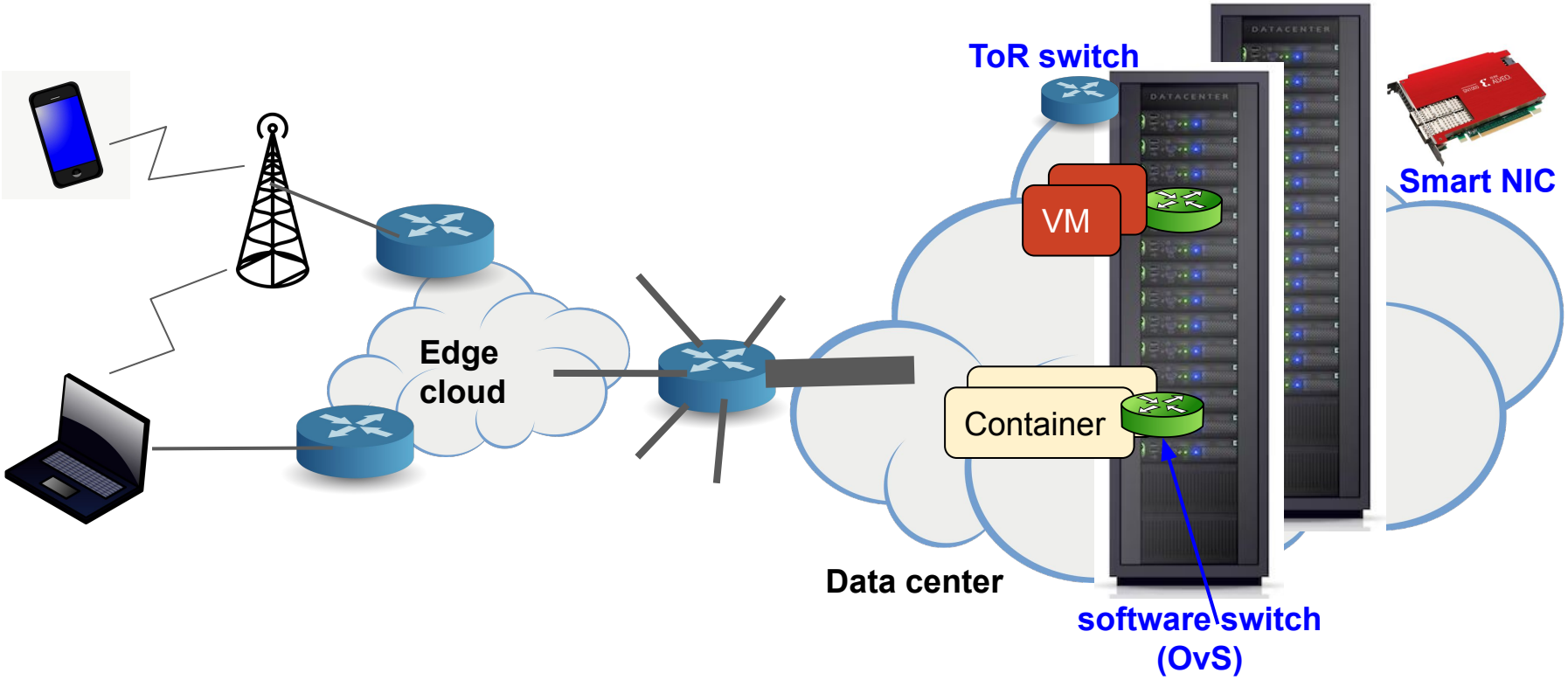
keshav19249@iiitd.ac.in

Harish S A

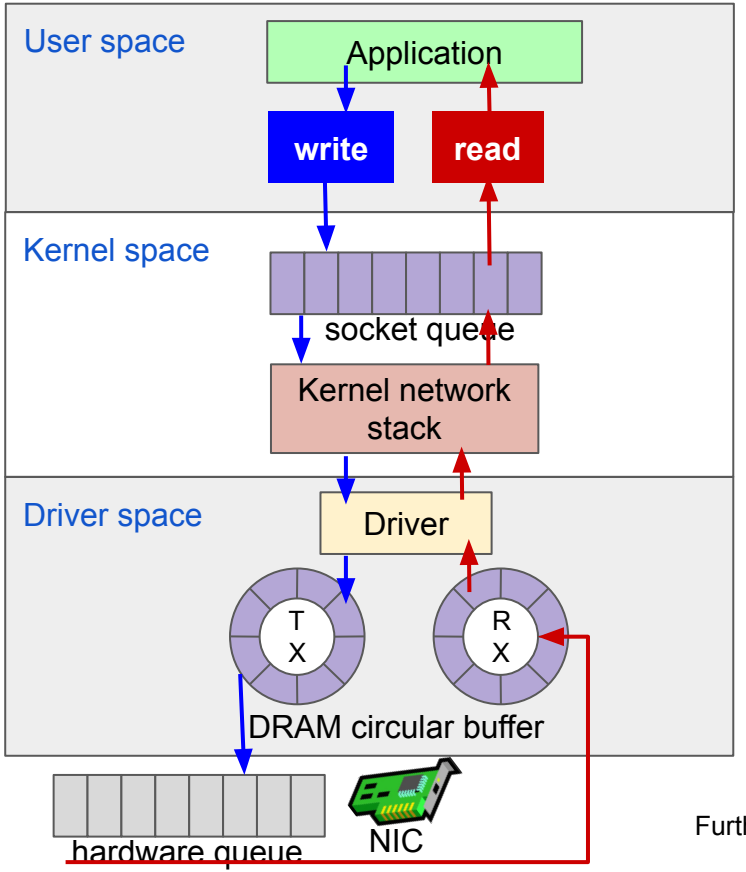
Research scholar (PMRF), IIT Hyderabad

cs21resch11009@iith.ac.in

The Big Picture



Network packet processing: Traditional network stack



Sources of overhead

Mode switching

Context switching
Lock/unlock

Packet copy
Dynamic alloc/dealloc

Per packet interrupts

Dynamic alloc/dealloc

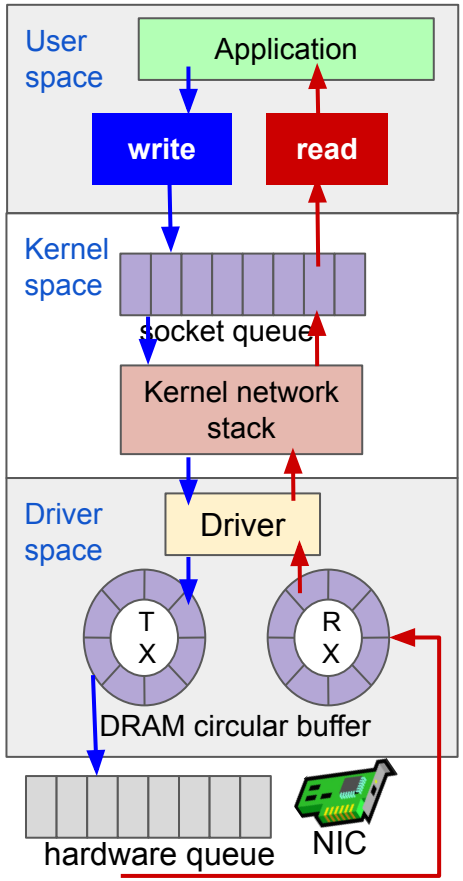
100 Gbps NIC[1]
- **RX: 20 CPUs**
- **TX: 10 CPUs**

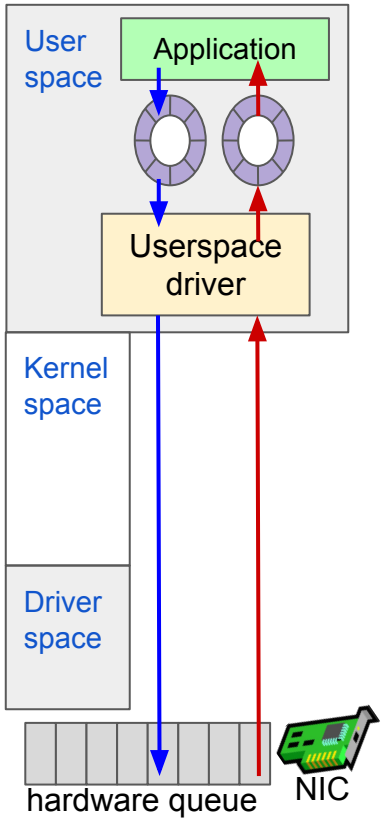
[1] Understanding Host Network Stack Overheads

Further details @ Kernel-bypass techniques for high-speed network packet processing

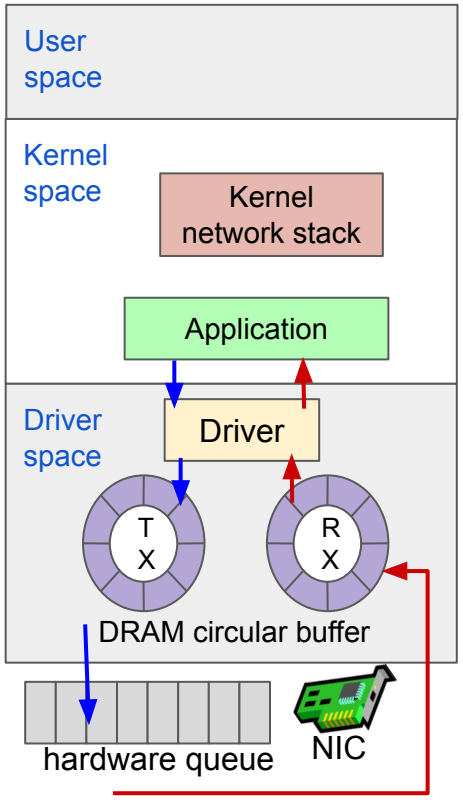
* Slide abstracts details for simplicity

Evolution of network packet processors

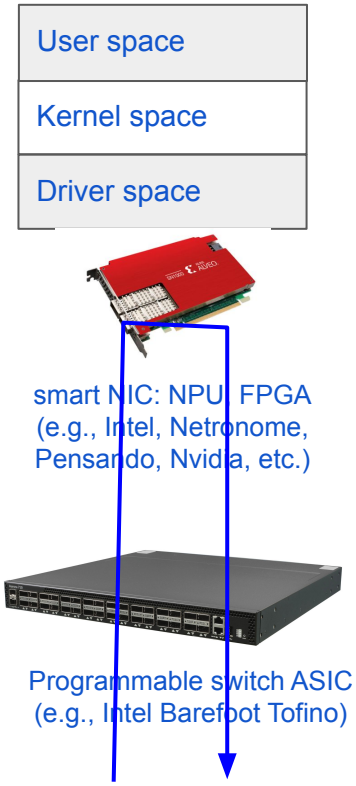




Kernel bypass (e.g. netmap, DPDK)



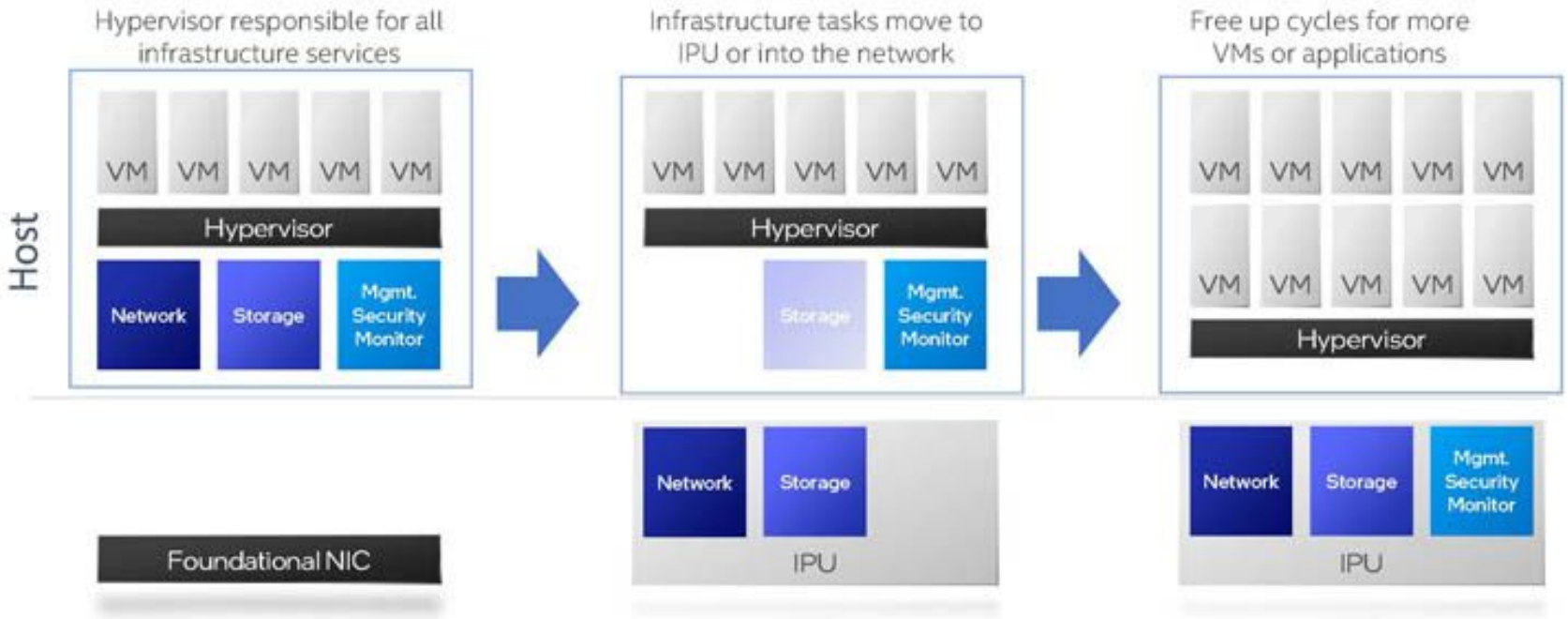
in-kernel compute (e.g. eBPF, XDP)



in-network compute (NIC/switch)

* Slide abstracts details for simplicity

Migrating Infrastructure Workloads to smart NICs (IPU/DPU/…)



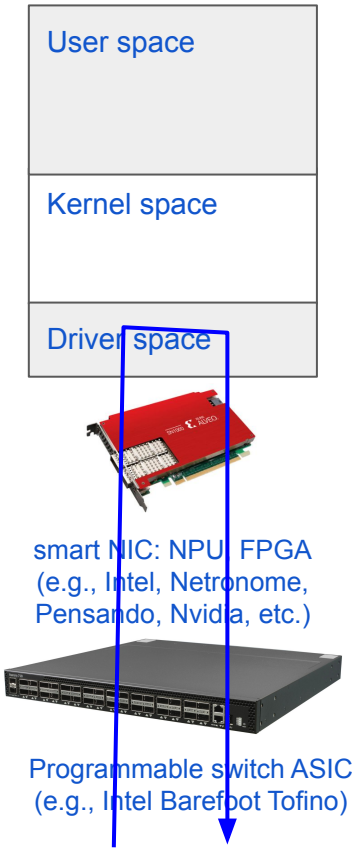
Ref: [Intel Mount Evans DPU IPU Arm Accelerator at Hot Chips 33]

Which is the best network packet processor?

Each packet processor has
its strengths, limitations and constraints!!

- Instruction set limitations
- Limited memory
- No loops
- No stalls -> Line rate processing
- …

Writing network packet processing programs



Traditional prog.

- **Socket API**

Kernel bypass

- **DPDK APIs**

in-kernel compute (e.g. eBPF, XDP)

- **Restricted “C”**

in-network compute

- **Restricted “C” OR target specific**

Sample code snippets for network targets

```
l2fwd_main_loop(void)
{
        struct rte_mbuf *pkts_burst[MAX_PKT_BURST];
        struct rte_mbuf *m;
        int sent;
        unsigned lcore_id;
        uint64_t prev_tsc, diff_tsc, cur_tsc, timer_tsc;
        unsigned i, j, portid, nb_rx;
        struct lcore_queue_conf *qconf;
        const uint64_t drain_tsc = (rte_get_tsc_hz() + US_PER_S - 1) / US_PER_S *
                        BURST_TX_DRAIN_US;
        struct rte_eth_dev_tx_buffer *buffer;
```

Kernel bypass: DPDK snippet

```
SEC("xdp_lb")
int xdp_load_balancer(struct xdp_md *ctx)
{
    void *data = (void *)(long)ctx->data;
    void *data_end = (void *)(long)ctx->data_end;

    bpf_printk("got something");

    struct ethhdr *eth = data;
    if (data + sizeof(struct ethhdr) > data_end)
        return XDP_ABORTED;

    if (bpf_ntohs(eth->h_proto) != ETH_P_IP)
        return XDP_PASS;
```

In-kernel compute: eBPF/XDP

```
table tbl_forward {
    key = {
        ig_intr_md.ingress_port : exact;
    }

    actions = {
        act_forward;
        // nop;
    }
```

In-network compute: Programmable switch

```
int pif_plugin_crc32_custom_hash(EXTRACTED_HEADERS_T *headers, ACTION_DATA_T *action_data){

    PIF_PLUGIN_crc32_hash_T *hash = pif_plugin_hdr_get_crc32_hash(headers);
    uint32_t hv = hash_calculation(headers);
    //hv = hash_me_crc32c((void *) hk, sizeof(hk), 1);
    PIF_HEADER_SET_crc32_hash___hash(hash,hv);

    return PIF_PLUGIN_RETURN_FORWARD;
}
```

In-network compute: Netronome NIC

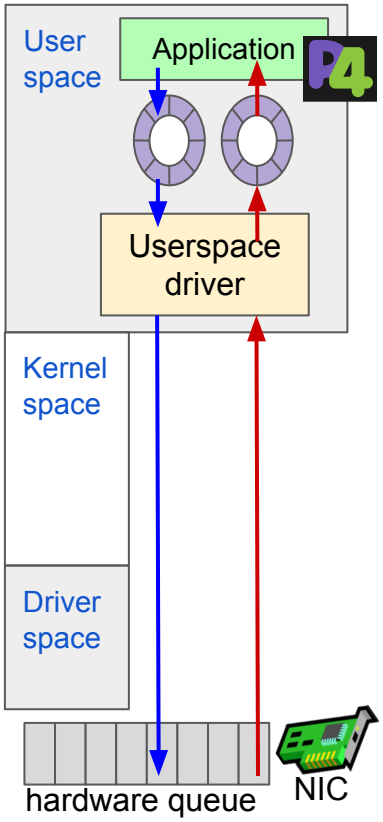
```
void AES_cipher(aes_byte_t in[16],aes_byte_t key[240],aes_byte_t cp_t[16])
{
        state_t pt_mat;
        #pragma HLS ARRAY_PARTITION variable=pt_mat complete dim=0
        aes_byte_t i,j;


        from_in_to_statematrix:for(i=0;i<4;i++)
        {
                #pragma HLS UNROLL
                for(j=0;j<4;j++)
                {
                        #pragma HLS UNROLL
                        pt_mat[i][j]=in[i*4+j];
                }
```

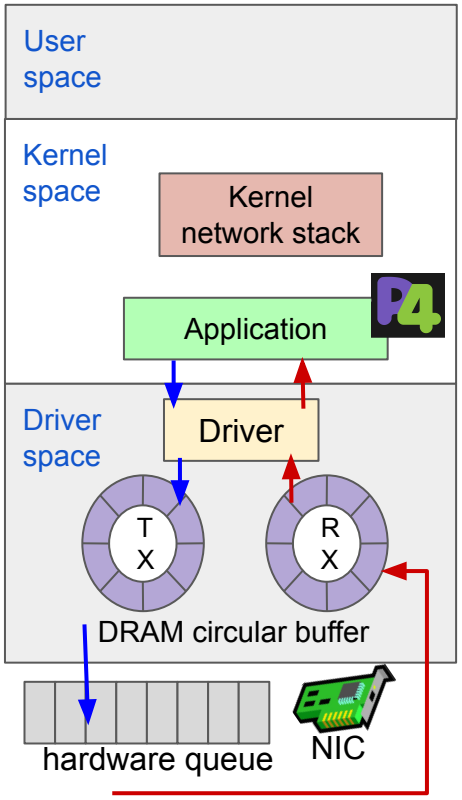
In-network compute: FPGA NIC

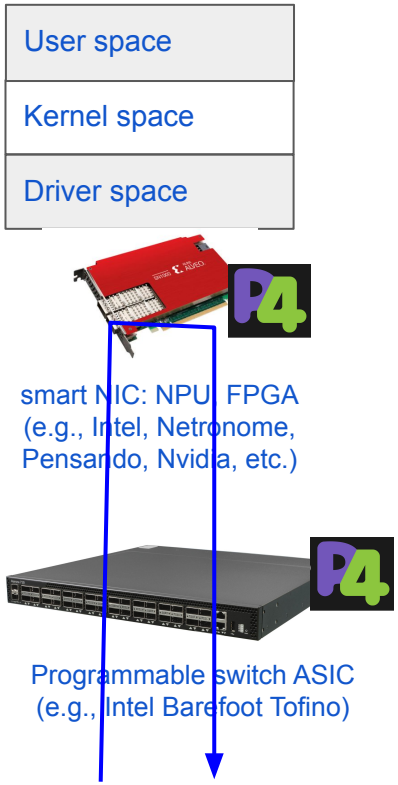
Need for a
standard network programming language!!

Is P4 the right answer?

P4 for ALL!







P4 is

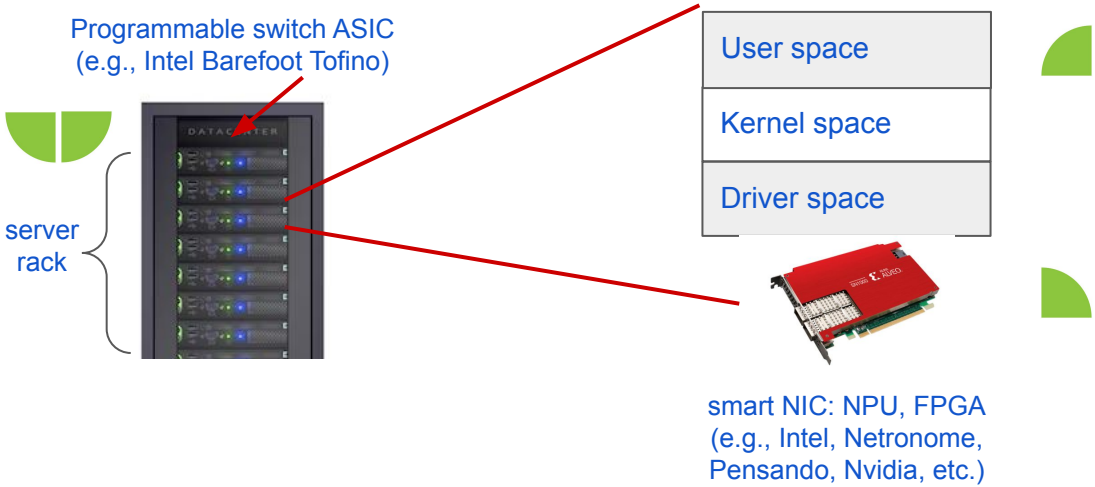
- Simpler than “C”
- Cross compilable across targets

Kernel bypass: DPDK

in-kernel compute: eBPF

in-network compute: smart NIC/ Tofino switch)

Use case: Key-value store



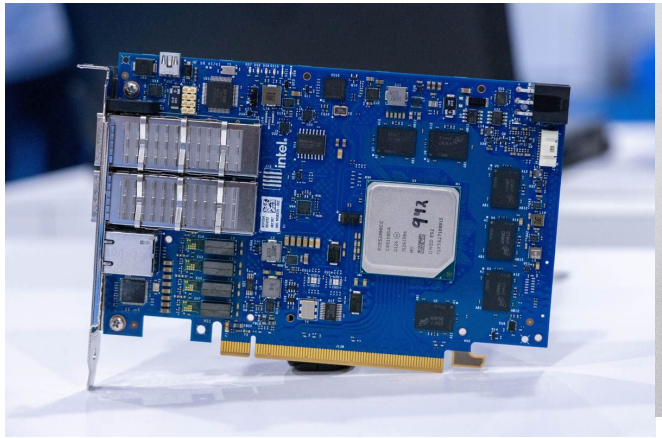
Where would you deploy this application?

Open research challenges

- **Program composition**
- **Program disaggregation**

Can P4 help solve them?

Industry trends



Intel Mounts Evan IPU (ASIC+FPGA)

Pensando DPU DSC-200

Xilinx SN1022 FPGA smart NIC

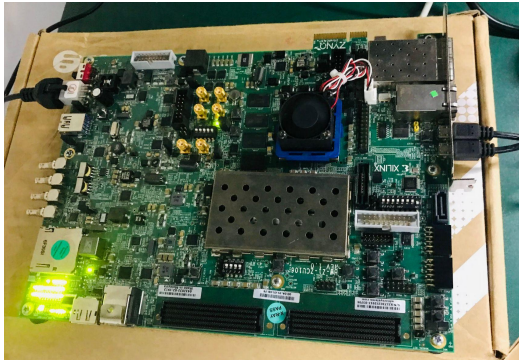
Other vendors: Broadcom, Nvidia, …

Programmable Netwoking Lab @ IIIT Delhi

https://github.com/pnl-iiitd

Intel Tofino switch

Netronome smart NIC
CX4000



Xilinx MPSoC FPGA board
ZCU106

Xilinx FPGA smart NIC
SN1022

What is P4?

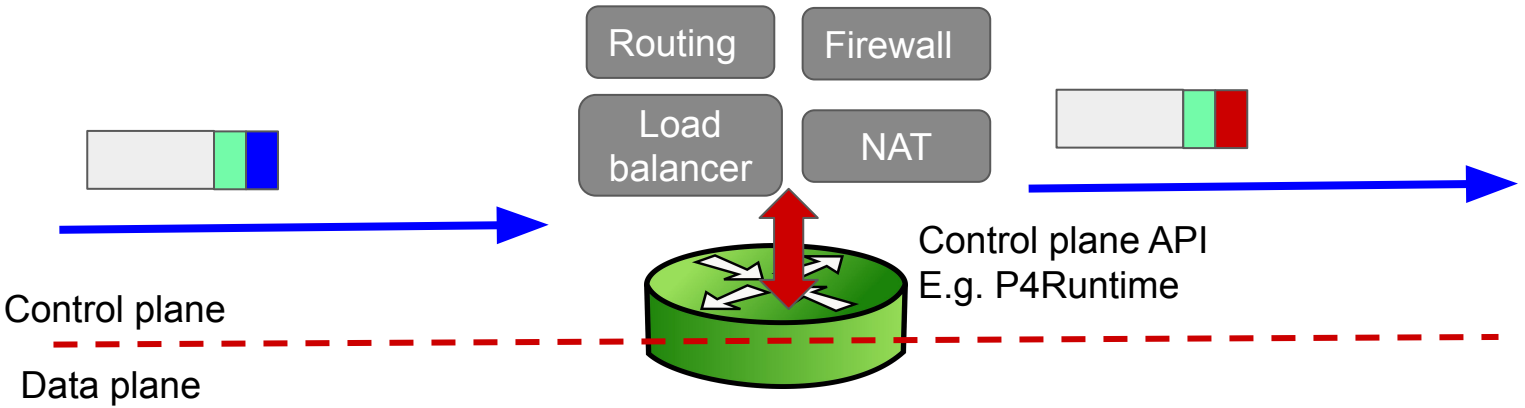
Portable Protocol-independent Packet Processors

Domain-specific language for network programming

Further slides on P4 theory are taken / inspired / adapted from:
P4 Language Tutorial, p4.org http://bit.ly/p4d2-2018-spring

Other reference material:
https://github.com/p4lang/tutorials
https://github.com/p4lang/p4c

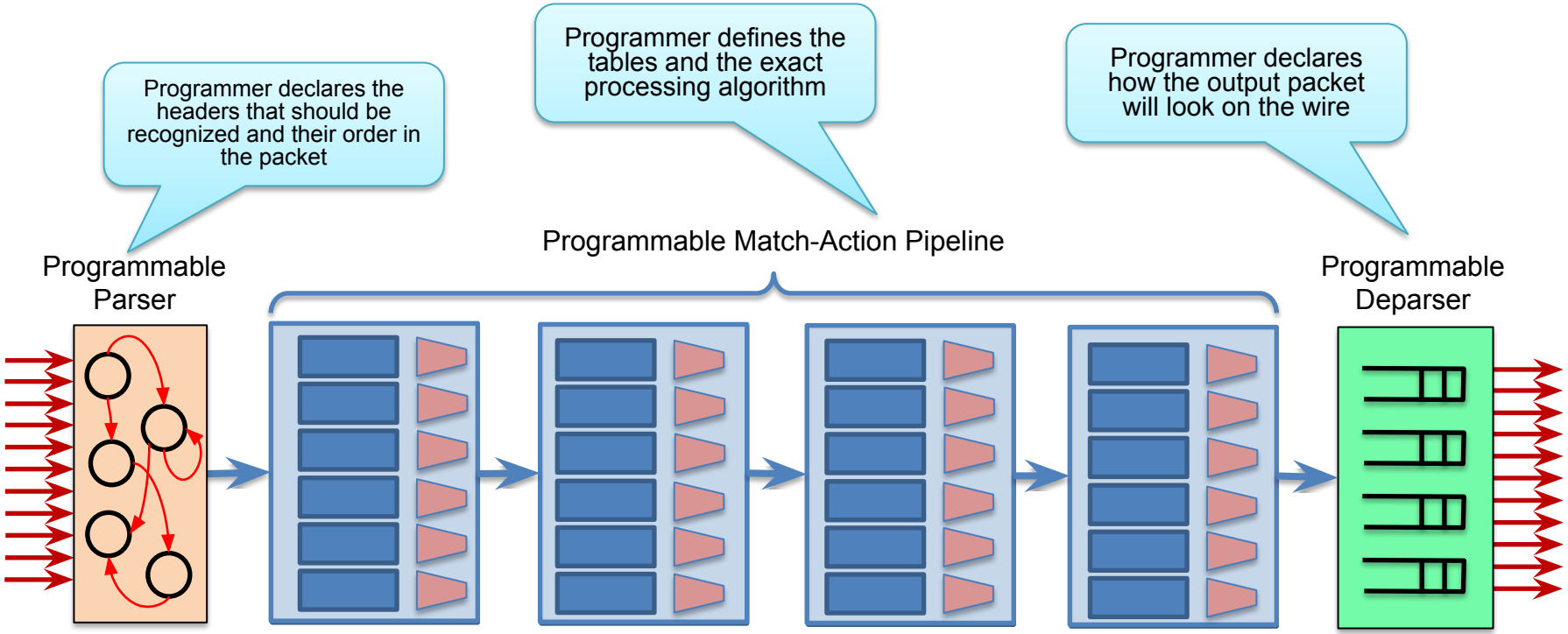
Packet processing flow



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	

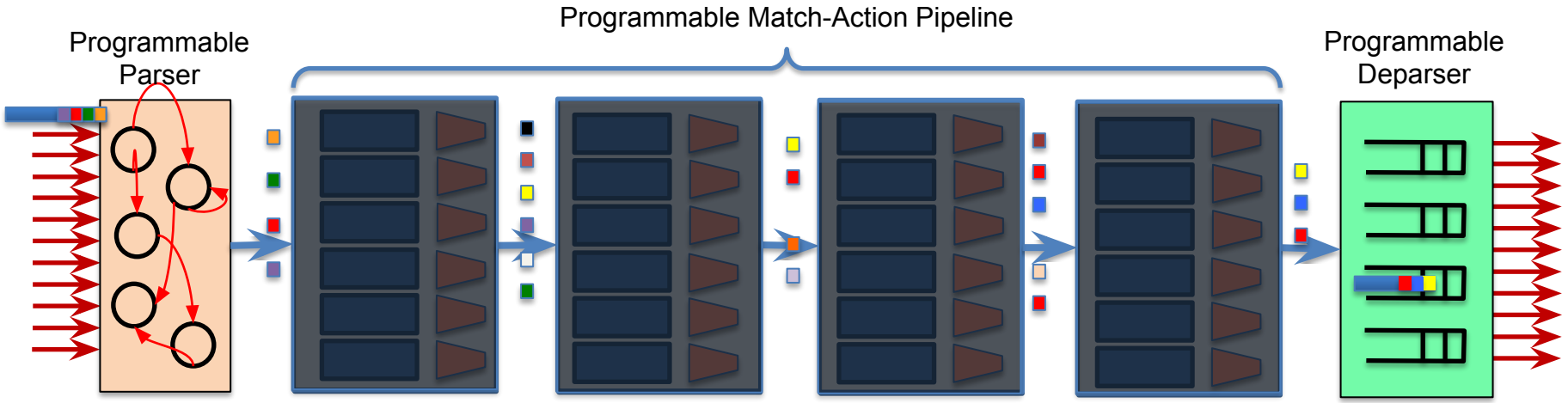
Match Action Table (MAT)

PISA: Protocol-Independent Switch Architecture



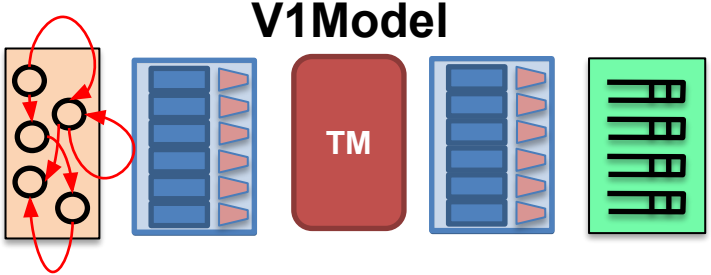
PISA in Action

- 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)

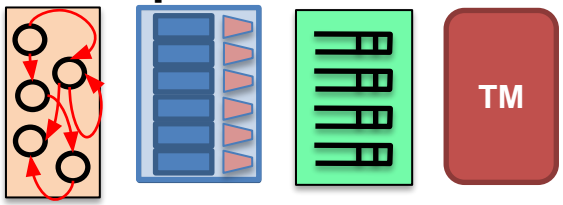


Example Architectures and Targets

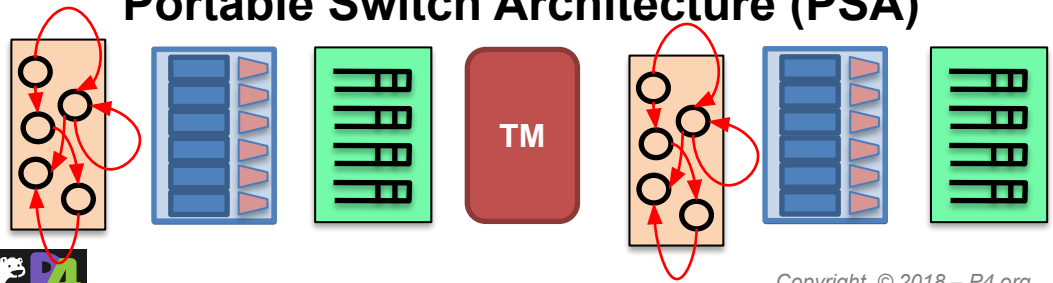
V1Model



SimpleSumeSwitch

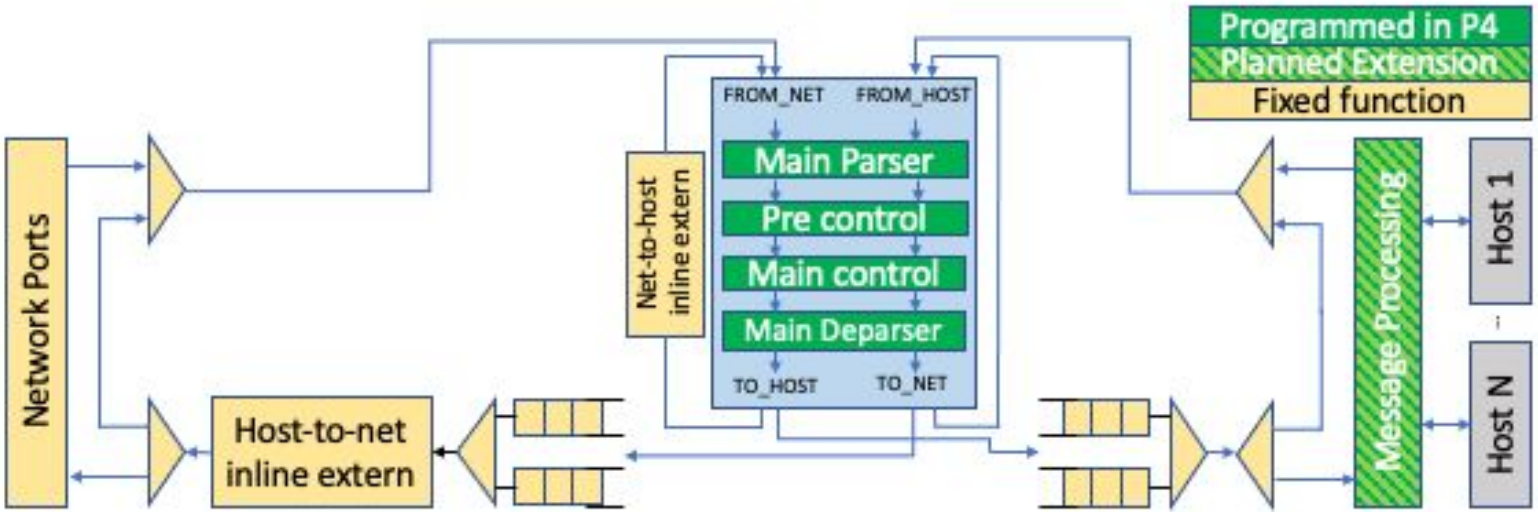


Portable Switch Architecture (PSA)



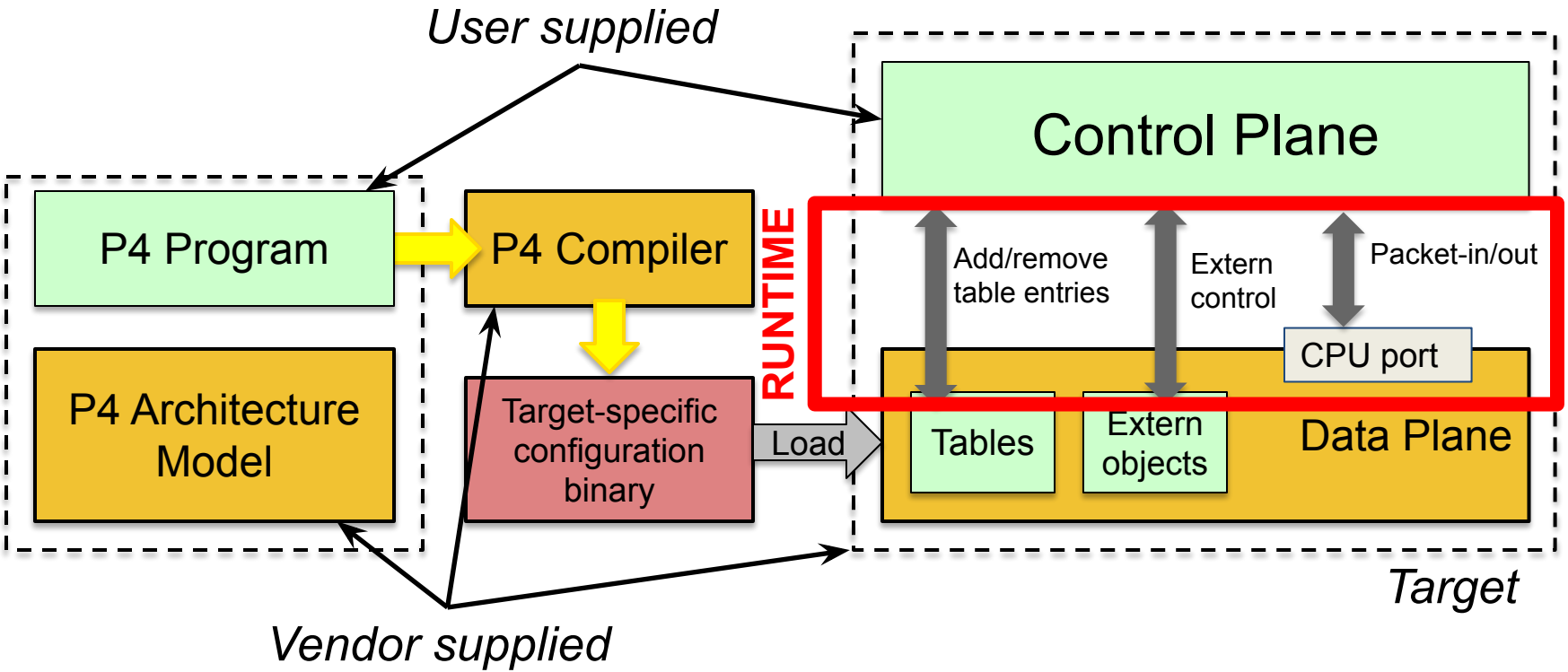
Anything

Portable NIC Architecture (PNA)



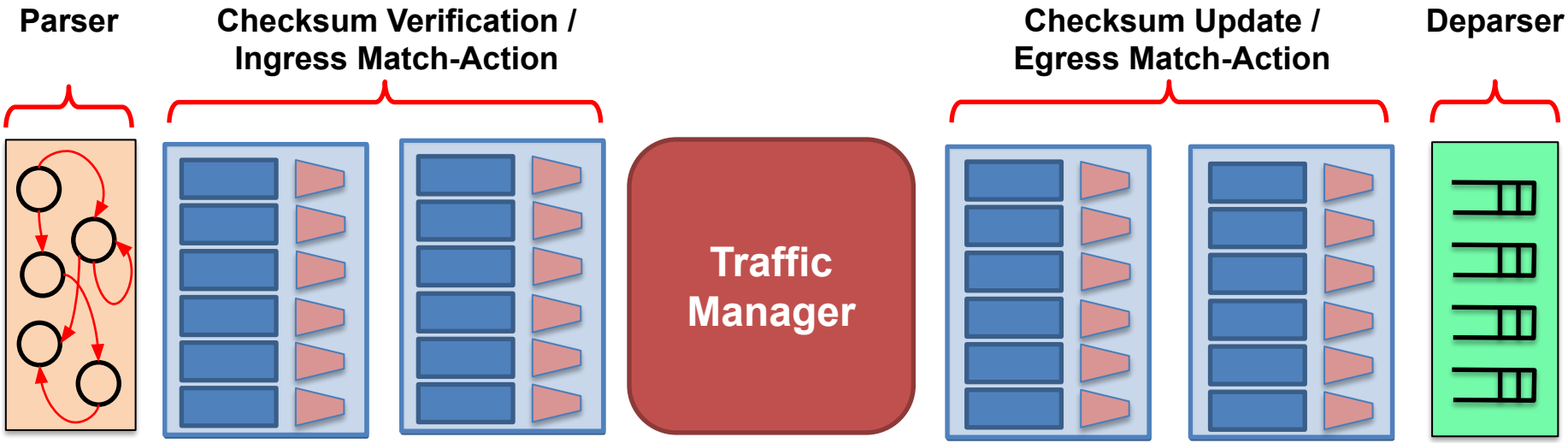
Ref: https://p4.org/p4-spec/docs/PNA.html

Programming a P4 Target



V1Model Architecture

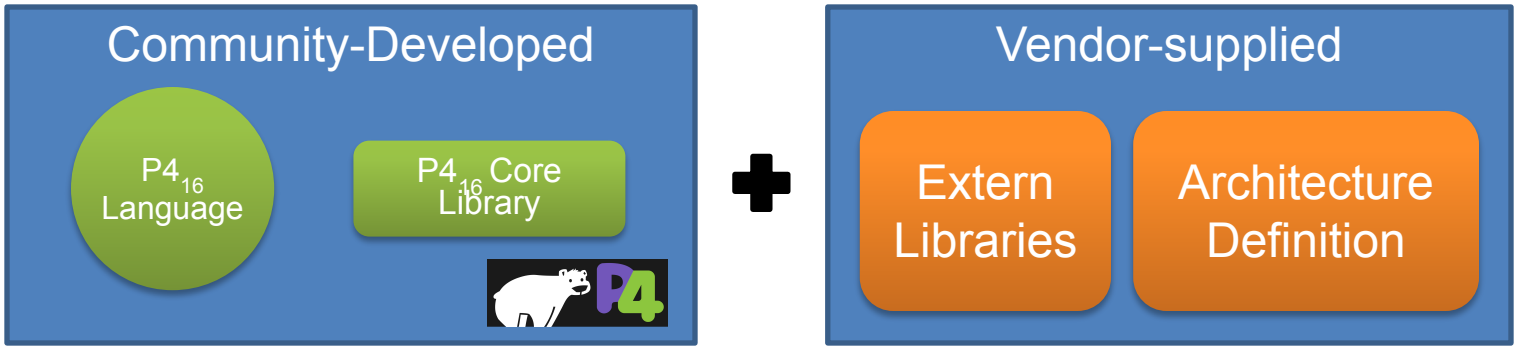
- **Implemented on top of Bmv2's `simple_switch` target**



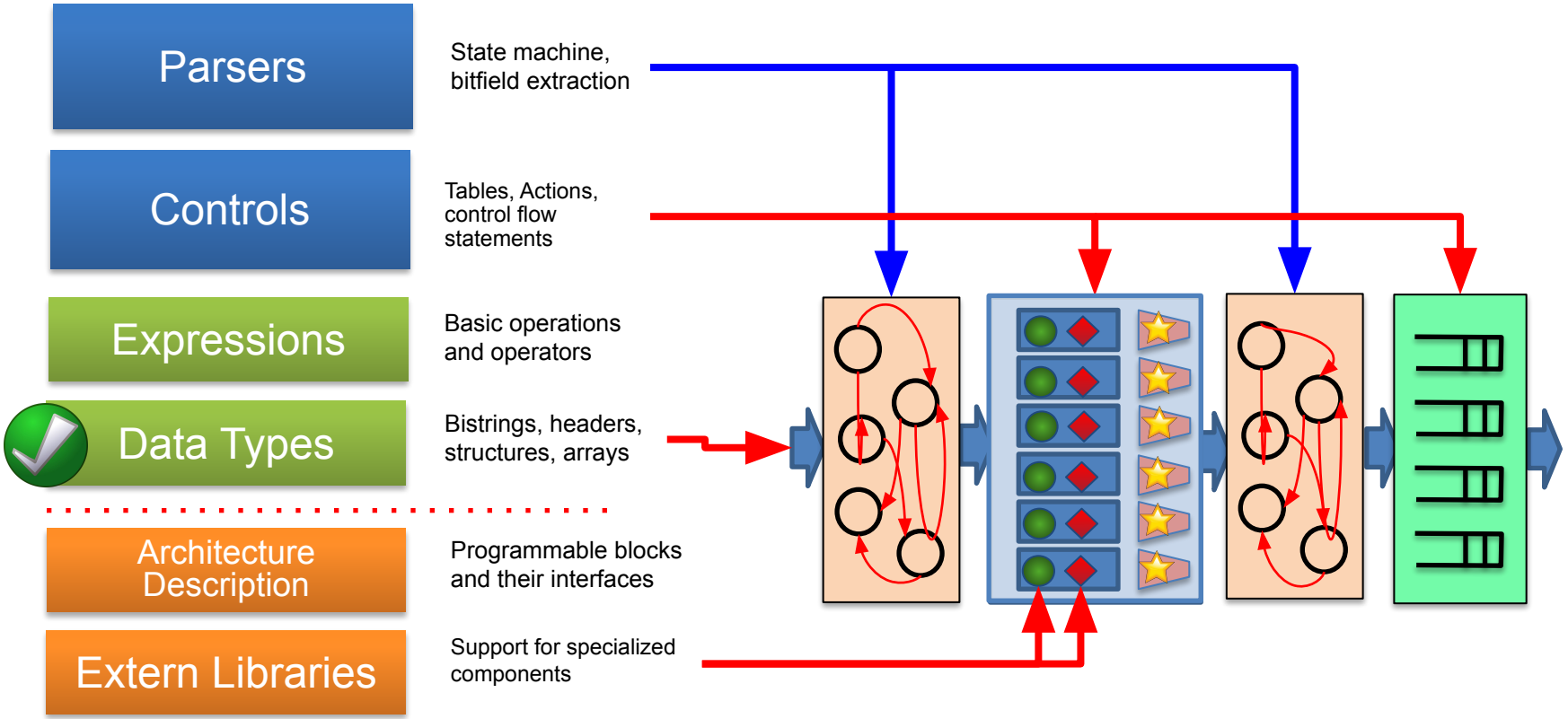
Reference:
https://docs.google.com/presentation/d/1zliBqsS8IOD4nQUboRRmF_19poeLLDLadD5zLzrTkVc/edit#slide=id.g37fca2850e_6_141

P4_16 Approach

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



P4$_{16}$ Language Elements



$P4_{16}$ Data Types

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

P4$_{16}$ header definitions

```
header ethernet_t {
  bit<48> dstAddr;
  bit<48> srcAddr;
  bit<16> etherType;
}
header ipv4_t {
  bit<4>    version;
  bit<4>    ihl;
  bit<8>    diffserv;
  bit<16>   totalLen;
  bit<16>   identification;
  bit<3>    flags;
  bit<13>   fragOffset;
  bit<8>    ttl;
  bit<8>    protocol;
  bit<16>   hdrChecksum;
  ip4Addr_t srcAddr;
  ip4Addr_t dstAddr;
}
struct my_headers_t {
  ethernet_t ethernet;
  ipv4_t ipv4;
}
```

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

P4$_{16}$ header definitions

```
header ethernet_t {
  bit<48> dstAddr;
  bit<48> srcAddr;
  bit<16> etherType;
}
header ipv4_t {
  bit<4>    version;
  bit<4>    ihl;
  bit<8>    diffserv;
  bit<16>   totalLen;
  bit<16>   identification;
  bit<3>    flags;
  bit<13>   fragOffset;
  bit<8>    ttl;
  bit<8>    protocol;
  bit<16>   hdrChecksum;
  ip4Addr_t srcAddr;
  ip4Addr_t dstAddr;
}
struct my_headers_t {
  ethernet_t ethernet;
  ipv4_t ipv4;
}
```

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

Derived Types

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

P4$_{16}$ header definitions

```
header ethernet_t {
  bit<48> dstAddr;
  bit<48> srcAddr;
  bit<16> etherType;
}
header ipv4_t {
  bit<4>    version;
  bit<4>    ihl;
  bit<8>    diffserv;
  bit<16>   totalLen;
  bit<16>   identification;
  bit<3>    flags;
  bit<13>   fragOffset;
  bit<8>    ttl;
  bit<8>    protocol;
  bit<16>   hdrChecksum;
  ip4Addr_t srcAddr;
  ip4Addr_t dstAddr;
}
struct my_headers_t {
  ethernet_t ethernet;
  ipv4_t ipv4;
}
```

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

Derived Types

- **header** — Ordered collection of members
  - Byte-aligned
  - Can be valid or invalid
  - Can contain **bit<n>**, **int<n>**, and **varbit<n>**
  - Provides several operations to test and set validity bit: **isValid()**, **setValid()**, and **setInvalid()**
- **struct** — Unordered collection of members
  - No alignment restrictions
  - Can contain any derived data type

P4$_{16}$ header definitions

```
header ethernet_t {
  bit<48> dstAddr;
  bit<48> srcAddr;
  bit<16> etherType;
}
header ipv4_t {
  bit<4>    version;
  bit<4>    ihl;
  bit<8>    diffserv;
  bit<16>   totalLen;
  bit<16>   identification;
  bit<3>    flags;
  bit<13>   fragOffset;
  bit<8>    ttl;
  bit<8>    protocol;
  bit<16>   hdrChecksum;
  ip4Addr_t srcAddr;
  ip4Addr_t dstAddr;
}
struct my_headers_t {
  ethernet_t ethernet;
  ipv4_t ipv4;
}
```

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

Derived Types

- **header** — Ordered collection of members
  - Byte-aligned
  - Can be valid or invalid
  - Can contain **bit<n>**, **int<n>**, and **varbit<n>**
  - Provides several operations to test and set validity bit: **isValid()**, **setValid()**, and **setInvalid()**
- **struct** — Unordered collection of members
  - No alignment restrictions
  - Can contain any derived data type
- **Header stacks:** arrays of headers
- **Header Union:** one of several headers

P4$_{16}$ header definitions

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

Basic Types

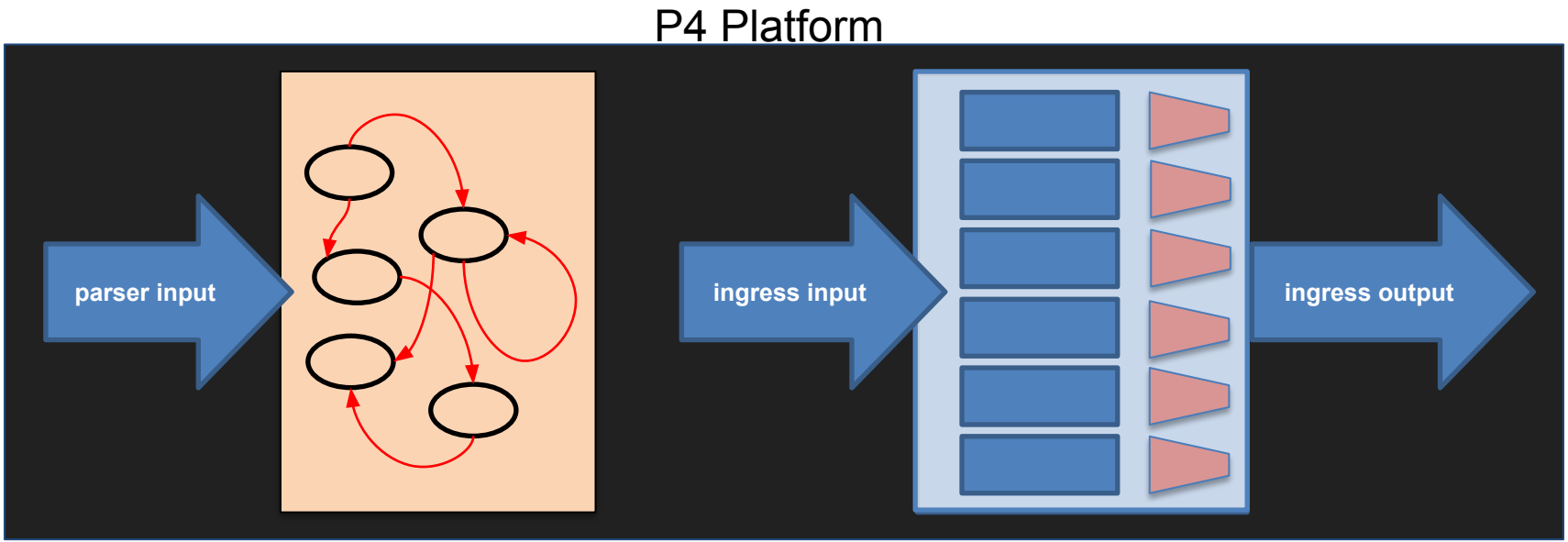
- **bit<n>**: Unsigned integer (bitstring) of size n
  - **bit** is the same as **bit<1>**
- **int<n>**: Signed integer of size n (>=2)
- **varbit<n>**: Variable-length bitstring

Derived Types

- **header** — Ordered collection of members
  - Byte-aligned
  - Can be valid or invalid
  - Can contain **bit<n>**, **int<n>**, and **varbit<n>**
  - Provides several operations to test and set validity bit: **isValid()**, **setValid()**, and **setInvalid()**
- **struct** — Unordered collection of members
  - No alignment restrictions
  - Can contain any derived data type
- **Header stacks:** arrays of headers
- **Header Union:** one of several headers
- **typedef** — Alternative name for a type

Using structs for standard metadata

- Standard 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



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)

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

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

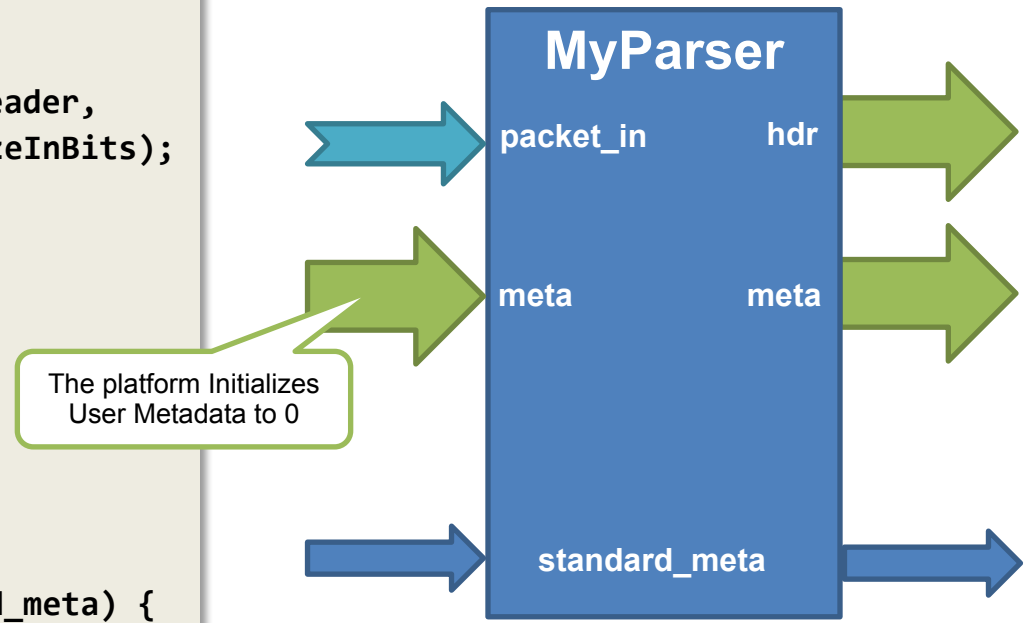
```
#include <core.p4>
#include <v1model.p4>
/* HEADERS */
struct metadata { ... }
struct headers {
  ethernet_t   ethernet;
  ipv4_t       ipv4;
}
/* PARSER */
parser MyParser(packet_in packet,
          out headers hdr,
          inout metadata meta,
          inout standard_metadata_t smeta) {
  ...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

Programming the Parser

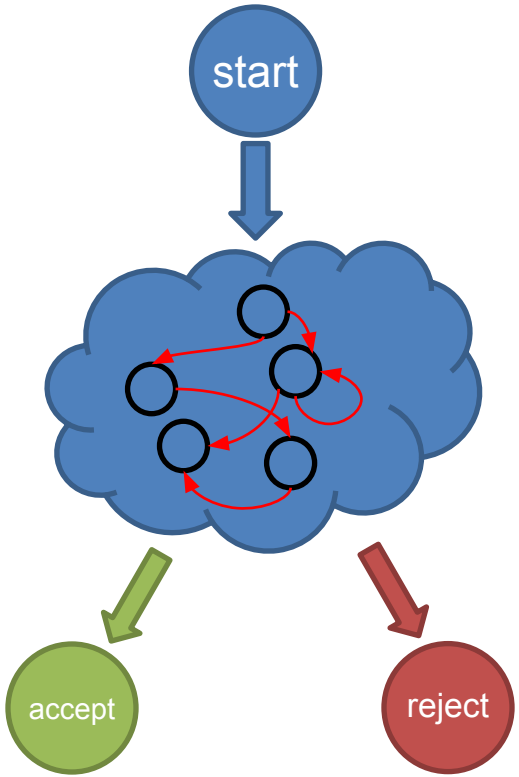
Parsers (V1Model)

```
/* From core.p4 */
extern packet_in {
  void extract<T>(out T hdr);
  void extract<T>(out T variableSizeHeader,
                  in bit<32> variableFieldSizeInBits);
  T lookahead<T>();
  void advance(in bit<32> sizeInBits);
  bit<32> length();
}

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

  ...
}
```



P4$_{16}$ Parsers

- Parsers are special functions that map packets into headers and metadata
- Written in a state machine style
- Every parser has three predefined states
  - start
  - accept
  - reject
    - can be reached explicitly or implicitly
    - What happens in reject state is defined by the architecture
      - For V1Model; packet is not dropped
- 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)



Select Statement

```
state start {
  transition parse_ethernet;
}

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

state parse_ipv4 {
     packet.extract(hdr.ipv4);
     transition select(hdr.ipv4.ihl) {
     0 .. 4: reject;
     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

The Lookahead method

```
state start {
    transition select(packet.lookahead<bit<8>>()) {
        0: parse_tcp_option_end;
        1: parse_tcp_option_nop;
        2: parse_tcp_option_ss;
        3: parse_tcp_option_s;
        5: parse_tcp_option_sack;
    }
}

…

state parse_tcp_option_sack {
…
}
```

- Provided by the **packet_in** packet abstraction
- Evaluates to a set of bits from the input packet **without advancing** the **nextBitIndex** pointer.
- Similar to extract, it will transition to reject and set the error if there are not enough bits in the packet.

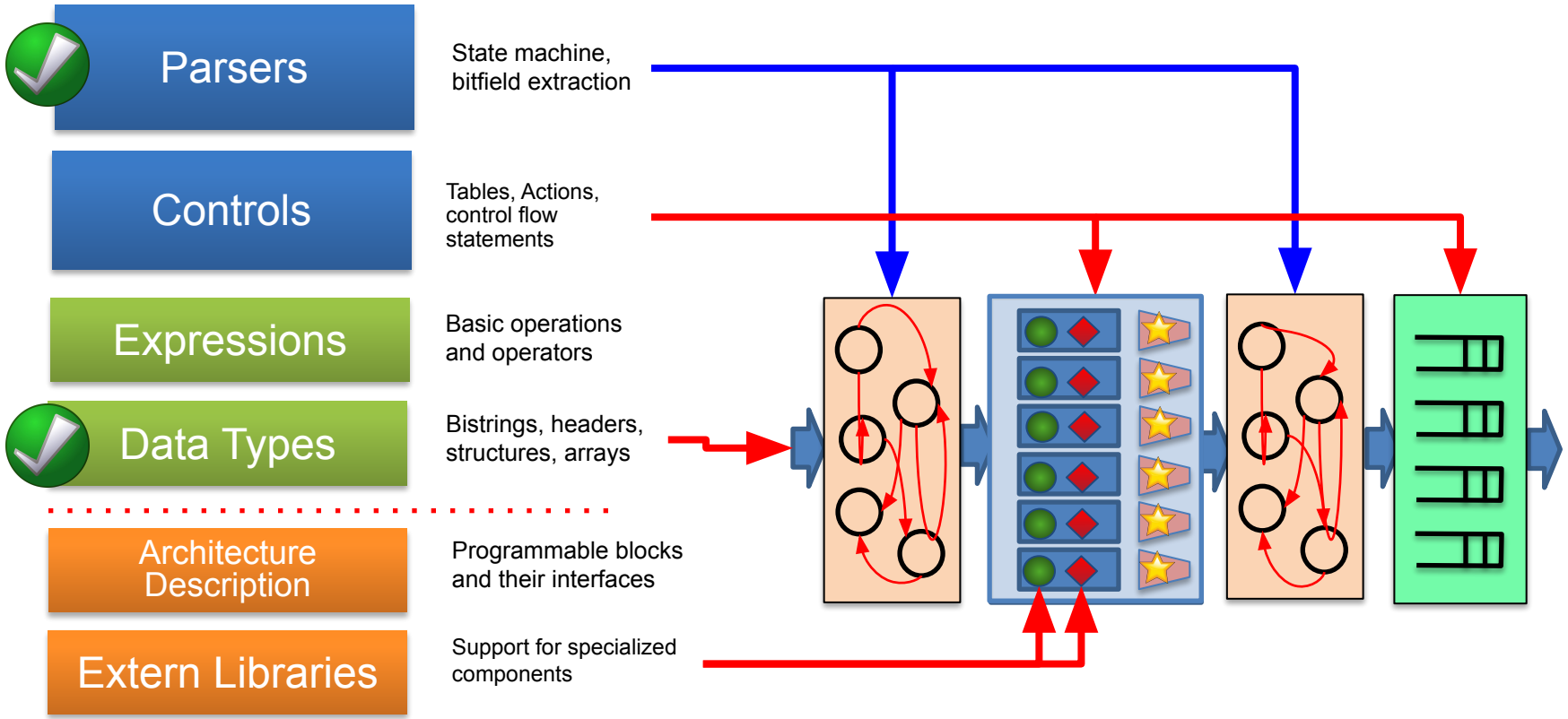
Packet Deparsing

P4$_{16}$ Deparsing

```
/* From core.p4 */
extern packet_out {
  void emit<T>(in T hdr);
}

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

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

P4$_{16}$ Language Elements

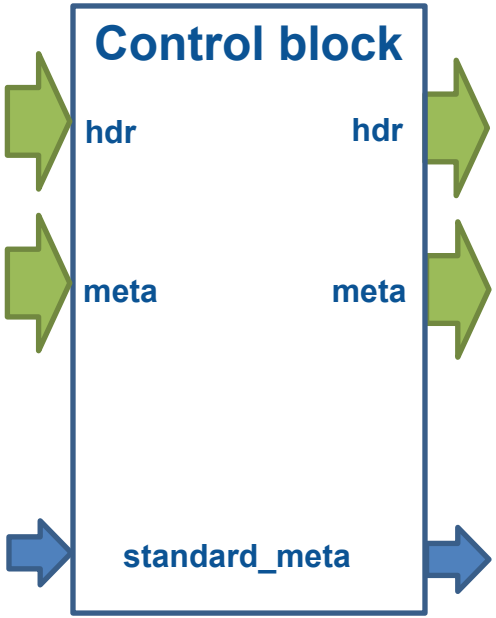


Programming the Match-Action Pipeline

- Controls
- Actions
- Tables

P4$_{16}$ Controls

- **Similar to C functions (without loops)**
  - should be representable as Directed Acyclic Graphs (DAG)
- **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)**



Lab 0: Hello World!

$ git clone https://github.com/pnl-iiitd/2023-P4-for-all-tutorial.git

Hello World: A wire that connects two ports

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;
```

Example: Simple Actions and Expressions

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

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

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

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

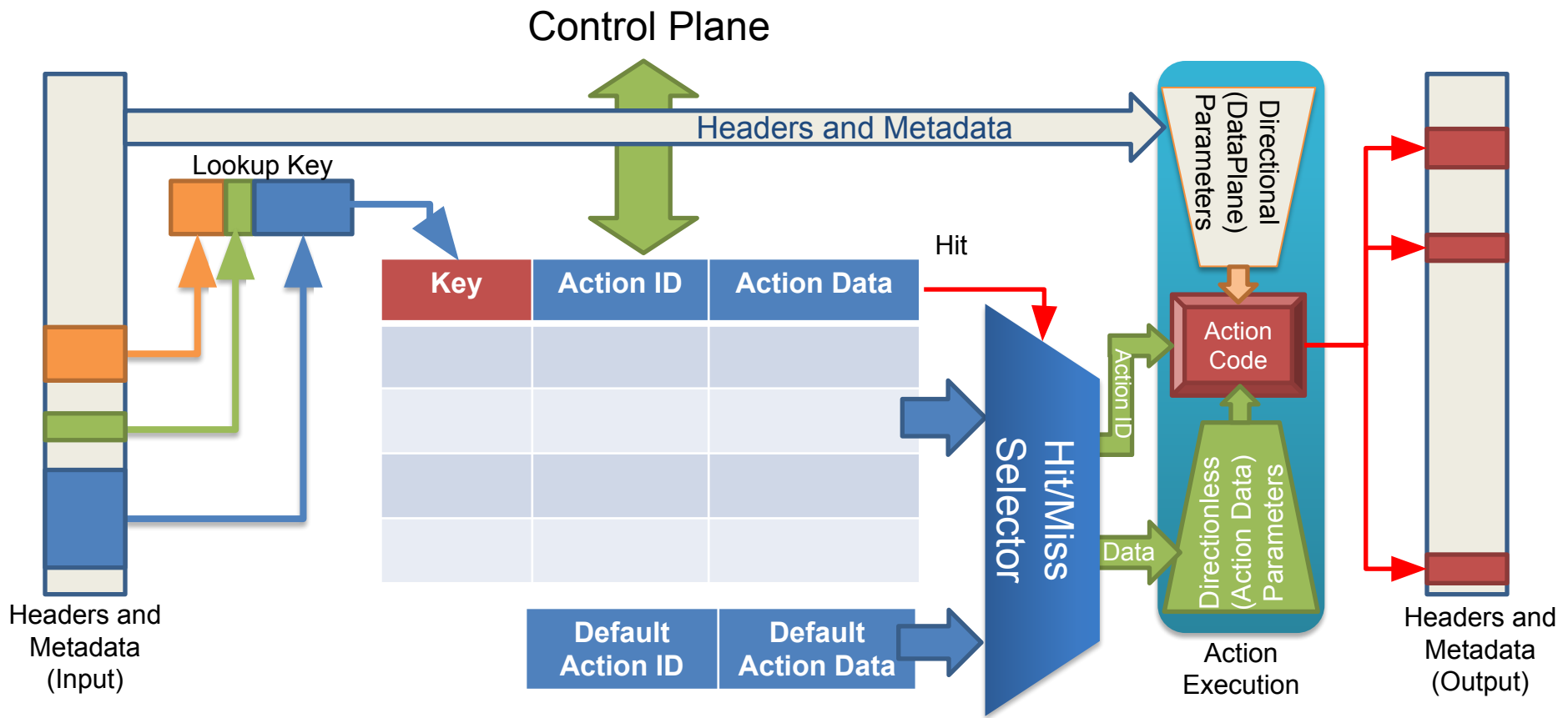
- **Many standard arithmetic and logical operations are supported**
  - +, -, *
  - ~, &, |, ^, >>, <<
  - ==, !=, >, >=, <, <=
  - No division/modulo
- **Non-standard operations:**
  - Bit-slicing: [m:l] (works as l-value too)
  - Bit Concatenation: ++

P4$_{16}$ Tables

- **Fundamental unit of a Match-Action Pipeline**
  - Specifies what data to match on and match kind
  - Specifies a list of *possible* actions
  - Optionally specifies a number of table **properties**
    - Size
    - Default action
    - Static entries
    - etc.
- **Each table contains one or more entries (rules)**
- **An entry contains:**
  - A specific key to match on
  - A **single** action that is executed when a packet matches the entry
  - Action data (possibly empty, optional)

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

P4$_{16}$ Tables: Match-Action Processing



Match Kinds

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Lab 1: Basic Forwarding

Exercise Link

Want to try more P4 exercises?

Visit [P4 lang tutorials]