ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

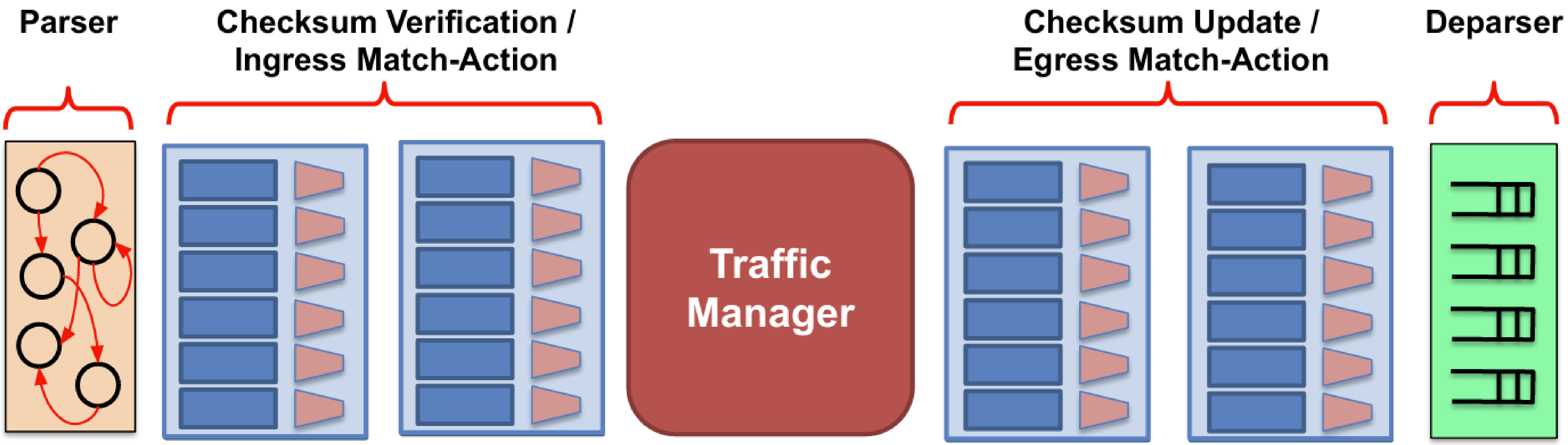
P4 Blocks and Smart NICs

Khanh Nam Chu

ONE LOVE. ONE FUTURE.

CONTENTS

V1 Model Architecture



I. P4 Program Syntax

1. $P4_{16}$ Language Elements

- Parsers: state machine, bitfield extraction.
- Controls: tables, actions, control flow statements.
- Expressions: basic operations and operators.
- Data Types: bitstrings, headers, structures, arrays.
- Architecture Description: programmable blocks and their interfaces.
- Extern Libraries: support for specialized components.

I. P4 Program Syntax

2. $P4_{16}$ Types

Basic Types:

- bit<n> : unsigned integer (bitstring) of size n (bit is the same as bit<1>).
- int<n> : signed integer of size n (>=2).
- varbit<n> : variable-length bitstring.

Header Types: ordered collection of members

- Contains bit<n>, int<n>, and varbit<n>.
- Byte-aligned.
- Provides several operations to test and set validity bit: isValid(), setValid(), and setInvalid().

Struct: unordered collection of members (with no alignment restrictions).

2. $P4_{16}$ Types

```
header ethernet_t {
  macAddr_t r;     dstAdd
  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;
}
```

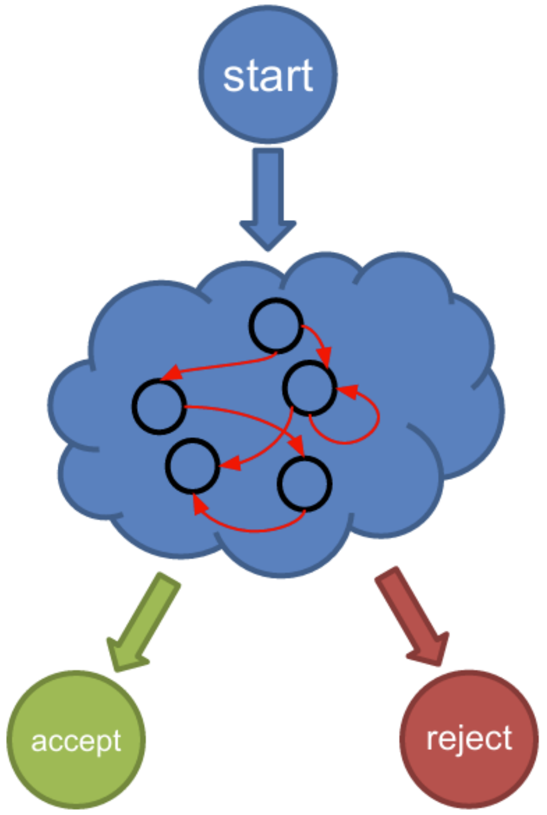
Ordered restrictions

```
struct headers {
  ethernet_t   ethernet;
  ipv4_t       ipv4;
}
```

No ordered restrictions

3. $P4_{16}$ Parser

- Parsers are functions that map packets into headers and metadata, written in a state machine style.
- Every parser has 3 predefined states: start, accept, reject.
- Other states may be defined by the programmer.



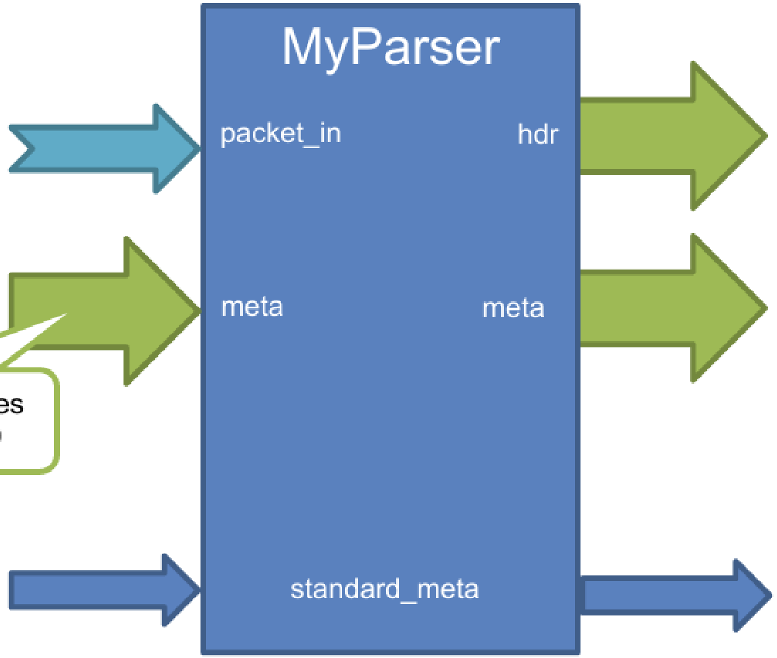
3. $P4_{16}$ Parser

```
/* From core.p4 */
extern packet_in {
  void extract<T>(out T hdr);
  void extract<T>(out T variableSizeHeader,
                  in bit<32> variableFieldSizeInBits);
  T lookahead<T>();
  void advance(in bit<32> sizeInBits);
  bit<32> length();
}
/* User Program */
parser MyParser(packet_in packet,
          out headers hdr,
          inout metadata meta,
          inout standard_metadata_t std_meta) {

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

}
```

The platform Initializes User Metadata to 0



3. $P4_{16}$ Parser

packet.extract(hdr.ethernet);

- takes the beginning of the incoming packet and extract the Ethernet header.
- reads a fixed number of bits (defined by the Ethernet header).
- save the extracted values into the **hdr.ethernet** structure.

transition accept;

done parsing this packet, no more headers needed to be extracted.

Furthermore, we can parse a header depending on the value of the other headers.

3. $P4_{16}$ Parser

```
state start {
  transition parse_ethernet;
}

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

- The parameter inside **transition select()** is a particular value in hex of **etherType**.
- If **etherType** equals 0x0800, we'll transition to state **parse_ipv4**
- If not, we are done parsing, no transition

4. $P4_{16}$ Controls

Processes: Match-Action Pipelines, Deparsers, and Additional forms of packet processing.

An example of swapping source MAC address and dst MAC address

Declaration region, in this case, the function **action swap_mac**

Using the function **apply**, the function **swap_mac** will be executed

This line of code means send the packet back to the same port it came port

```
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;
  }
}
```

4. $P4_{16}$ Tables

- The fundamental unit of a Match-Action Pipeline:
  - +) specifies what data to match on and match kind.
  - +) specifies a list of possible actions.
  - +) optionally specifies a number of table properties: size, default action, static entries,…
- Each table contains one or more entries(rules).
- An entry contains:
  - +) a specific key to match on.
  - +) a single action that is executed when a packet matches the entry.
  - +) action data (possibly empty).

4. $P4_{16}$ Tables

`<field>:<match_kind>`
match_kind can be: exact, lpm, ternary, range,…

3 built-in functions

maximum number of entries the table can hold is 1024

if no match, do nothing

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

4. $P4_{16}$ Tables

Applying Tables in Controls

```
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t standard_metadata) {
  table ipv4_lpm {
    ...
  }
  apply {
     ...
     ipv4_lpm.apply();
     ...
  }
}
```

5. $P4_{16}$ Deparser

- Assembles the headers back into a well-formed packet.
- Expressed as a control function.
- **emit(hdr)**: serializes header if it is valid.

Note that the order of emitting matters, the correct order should be:

packet.emit(hdr.ethernet);
packet.emit(hdr.ipv4);
packet.emit(hdr.tcp);

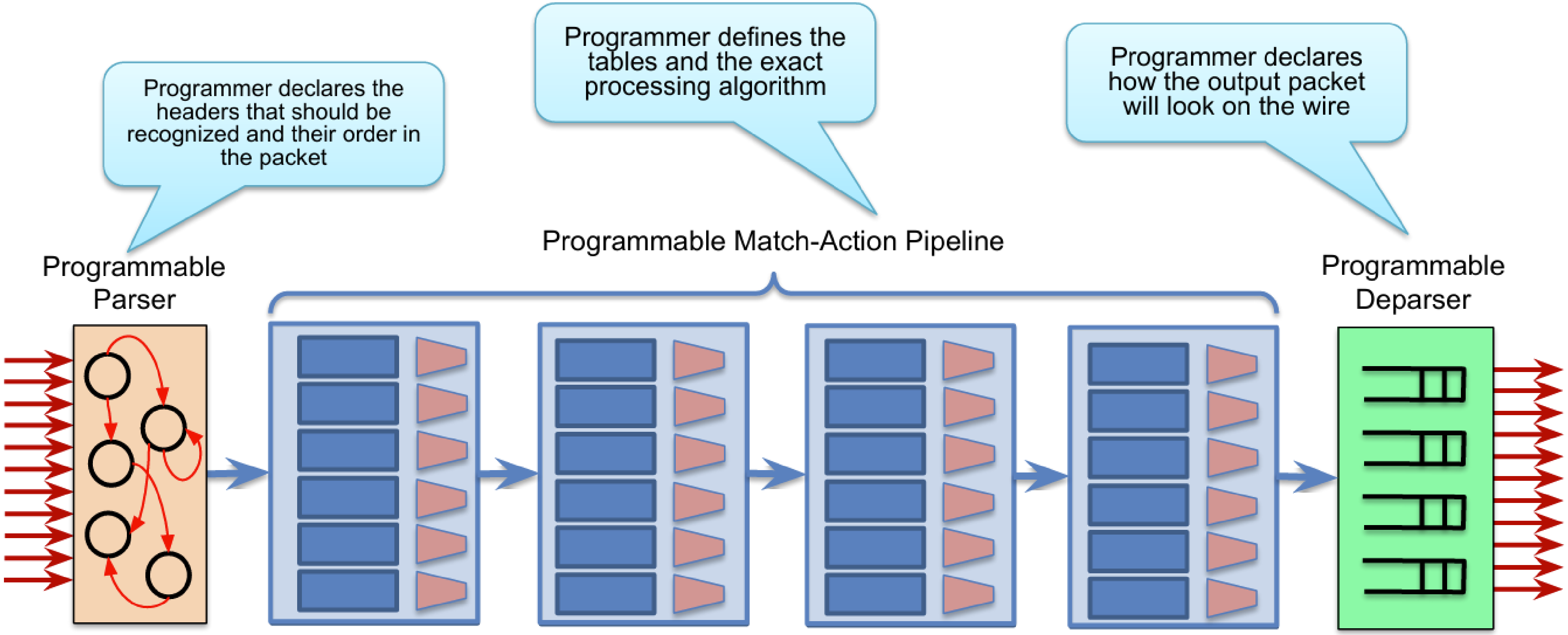
```
/* User Program */
control DeparserImpl(packet_out packet,
                     in headers hdr) {
  apply {
    ...
    packet.emit(hdr.ethernet);
    ...
  }
}
```

Assignment 1

- The link to the assignment: minmit/p4-exercises

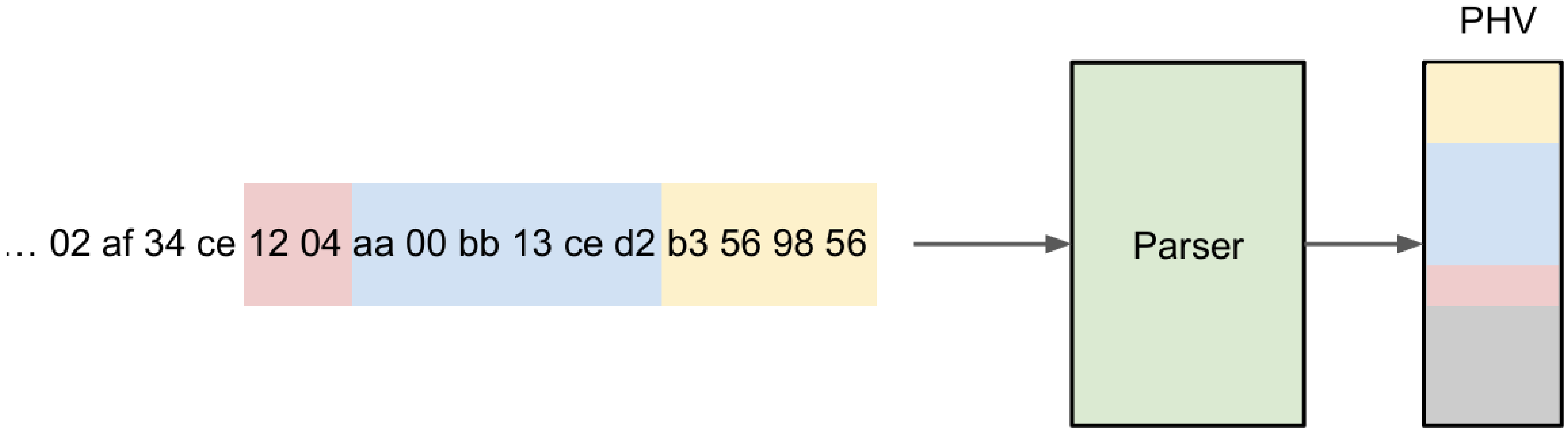
- The solution of the assignment: **https://docs.google.com/document/d/1QKshFH0B1N6YS1ReLrxOCiFQygA8AA2e/edit**

1. PISA model


Programmer declares the headers that should be recognized and their order in the packet
Programmer defines the tables and the exact processing algorithm
Programmer declares how the output packet will look on the wire
Programmable Parser
Programmable Match-Action Pipeline
Programmable Deparser

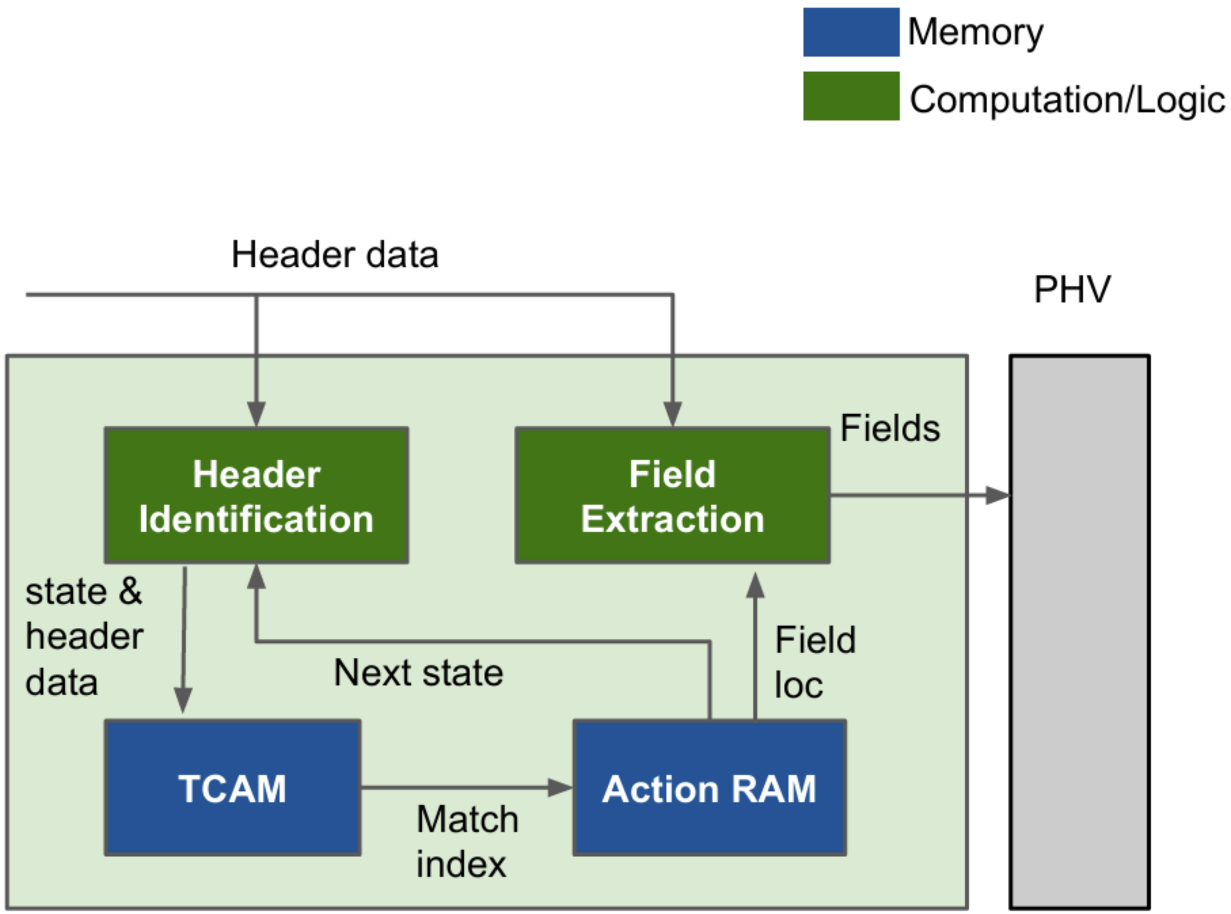
2. Programmable Parser

- Takes bits from a packet and outputs a Packet Header Vector (PHV).
- PHV: the collection of all the header fields.



2. Programmable Parser

Structure Programmable Parser



II. Programmable Switch Architecture

2. Programmable Parser

Header Identification

•**Logic**: Uses computation to detect protocol boundaries (e.g., Ethernet → IPv4 → TCP).
•**Purpose**: Determines what kind of header comes next in the packet.
•**Input**: Receives raw header data from the packet and the current state of parsing.
•**Output**: Passes the header type and data to the TCAM (for decision-making on what to do next).

Field Extraction

•**Purpose**: Extracts specific fields from the header (e.g., source IP, destination port).
•**Input**: Gets header data from the packet and lookup results from the Action RAM.
•**Output**: Fields are extracted and placed into the **PHV (Packet Header Vector)**.
•**Logic**: This unit understands how to break down a recognized header into usable fields.

2. Programmable Parser

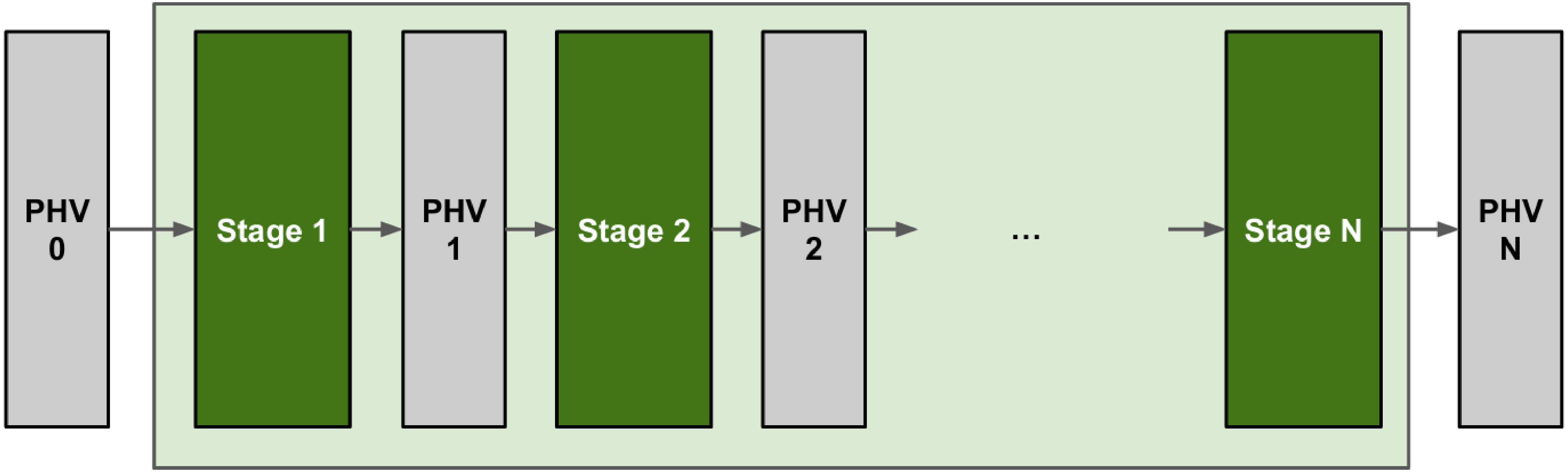
TCAM (Ternary Content-Addressable Memory)

•**Purpose**: Matches incoming header/state information to known patterns.
•**Input**: State and header data from the Header Identification block.
•**Output**: Match index sent to the Action RAM.
•**Memory**: Stores matching rules.

Action RAM

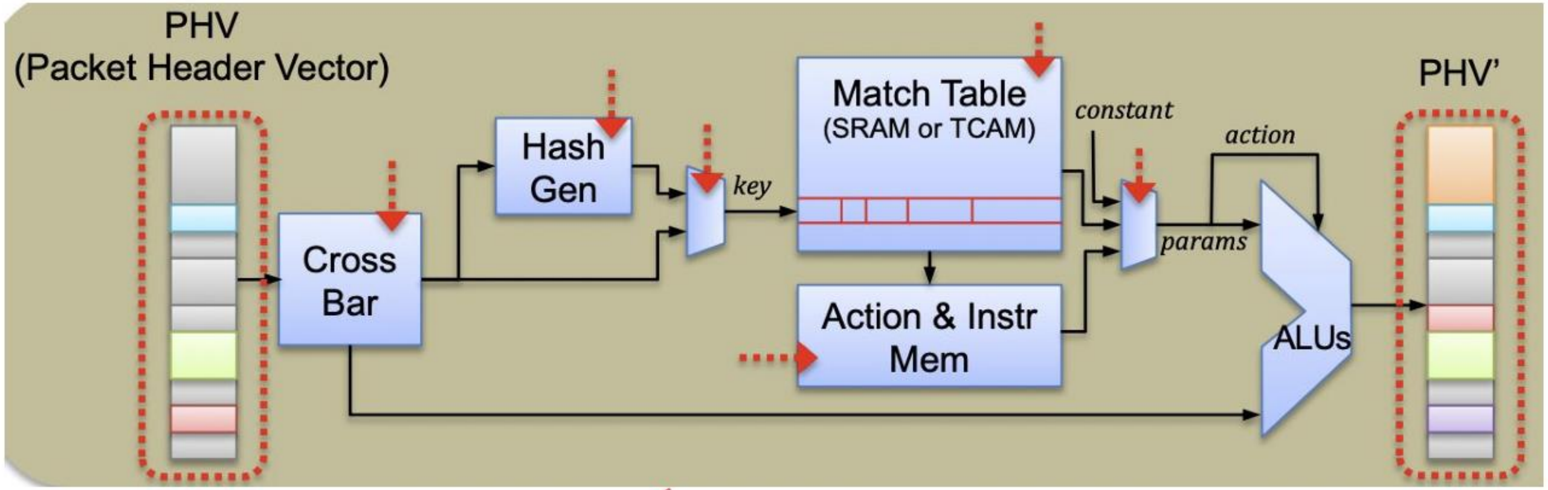
•**Purpose**: Stores the instructions for what to do next (like which fields to extract and what the next parser state is).
•**Input**: Match index from TCAM.
•**Output**: Field location (for Field Extraction), and next state (for Header Identification).
•**Memory**: Provides actions like "extract 32 bits at this offset," or "go to state X".

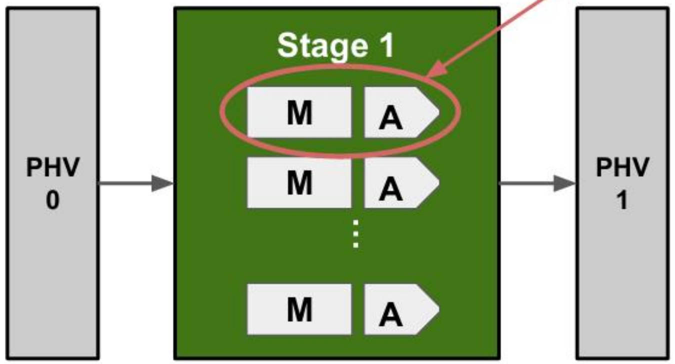
3. Ingress Processing



3. Ingress Processing

Inside a Match-Action unit





Match-action units are "programmed" by configuring the components marked with red dotted arrows.

3. Ingress Processing

Inside a Match-Action unit

Crossbar

- Selects which parts of the PHV (e.g., destination IP, protocol, etc.) are used as the **key** for the match operation.
- Think of this like selecting fields for lookup.

Hash Generator (Hash Gen)

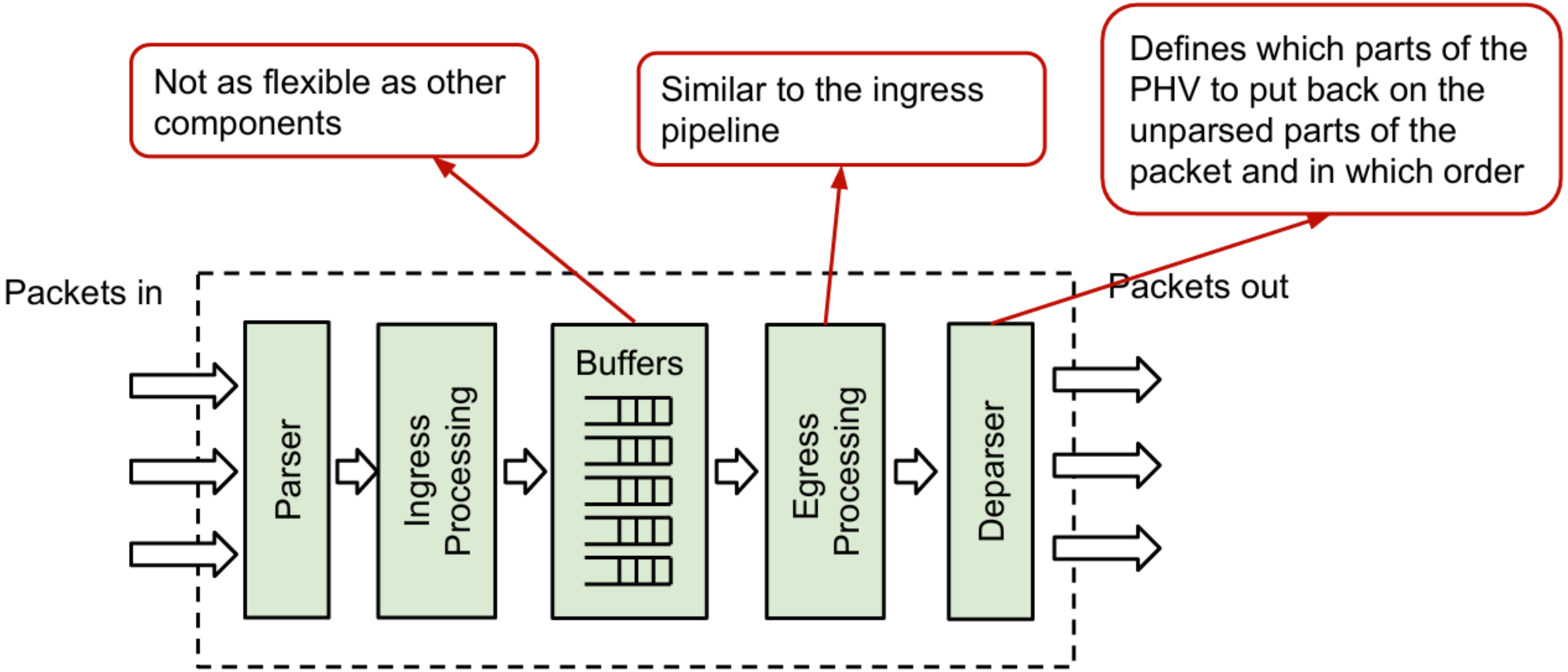
- Optional step depending on the type of match (e.g., **exact match** vs **ternary match**).
- Hashes the selected key for efficient lookup in the match table (mainly in SRAM-based match tables).

Action & Instruction Memory

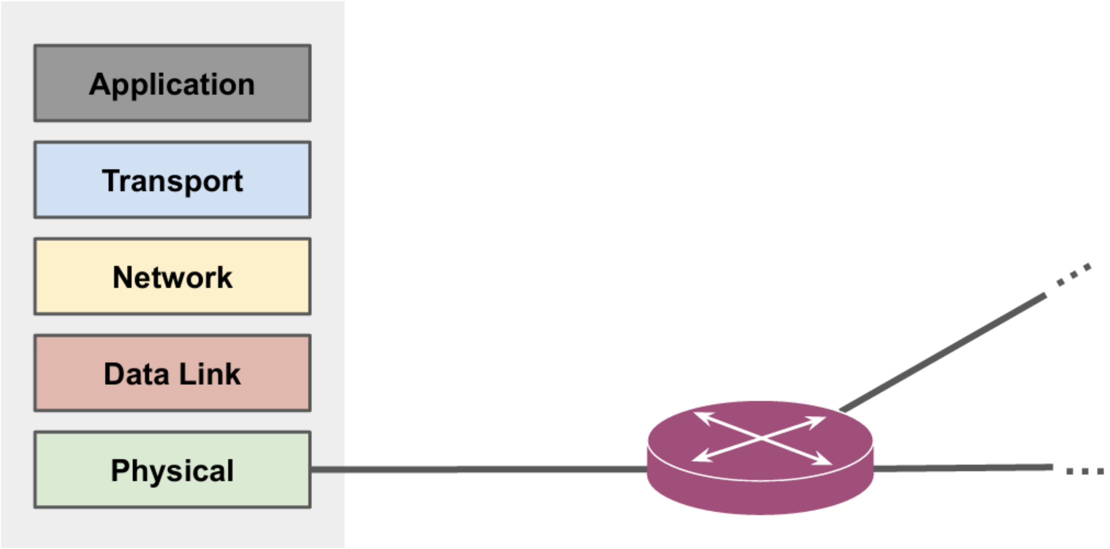
- Based on the **action ID** from the match table, it tells the system **what action to execute** (e.g., forward, drop, modify header).
- Also supplies any needed **parameters** (e.g., new output port, new MAC address).

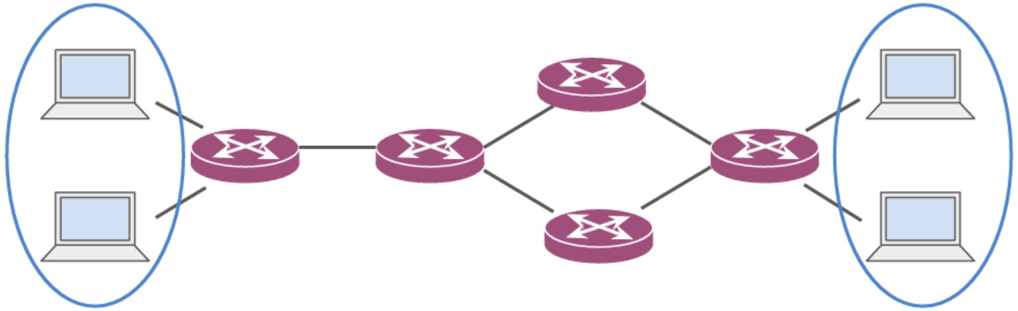
3. Ingress Processing

PISA: **P**rotocol-**I**ndependent **S**witch **A**rchitecture

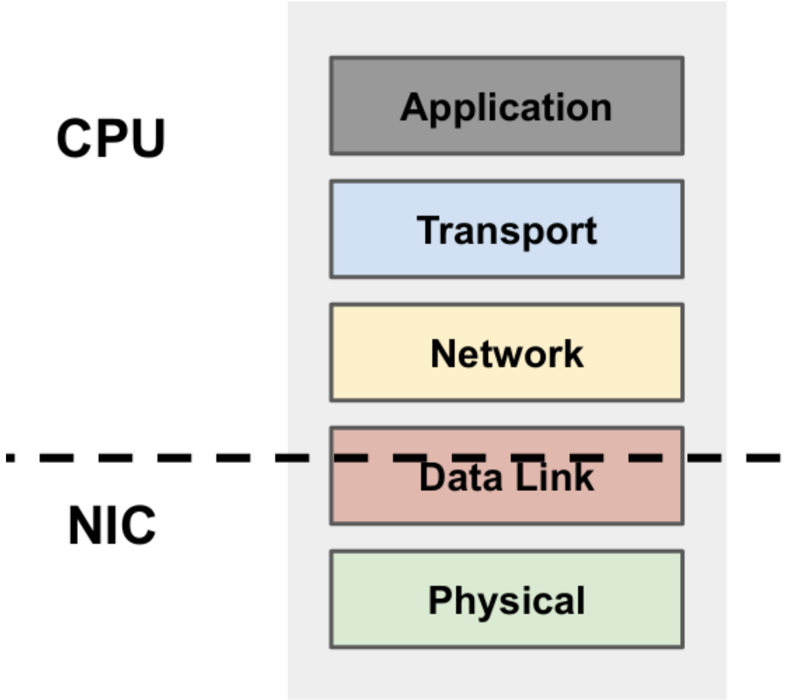


1. Network Programmability at the end points





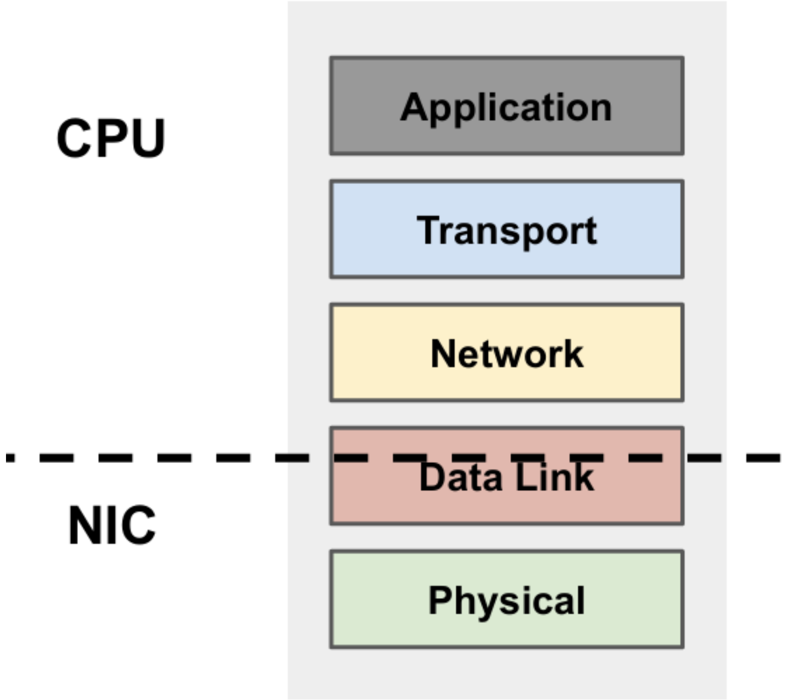
2. Network Interface Cards (NICs)



On transmit (egress):

- The host CPU generates packets on application request.
- Packets are sent to the NIC over PCIe.
- The NIC transforms packets to bits and sends them over the link.

2. Network Interface Cards (NICs)



On receive (ingress):

- The NIC turns bits into packets.
- Packets are sent to the host over PCIe.
- The host CPU processes packets and delivers them to applications.

3. Smart NICs

A regular NIC

+

A programmable domain-specific hardware

The hardware can be:
- Field Programmable Gate Arrays (FPGA).
- Multi-Core Systems on Chip (SoC).
- P4 programmable pipelines.

→The Smart NIC can even do the CPU's traditional job like processing packets and deliver them to applications.

THANK YOU !

hust.edu.vn fb.com/dhbkhn