UNIVERSITY OF WATERLOO

CS 856: Programmable Networks

Lecture 2: Programming the Data Plane with P4

Mina Tahmasbi Arashloo

Winter 2023

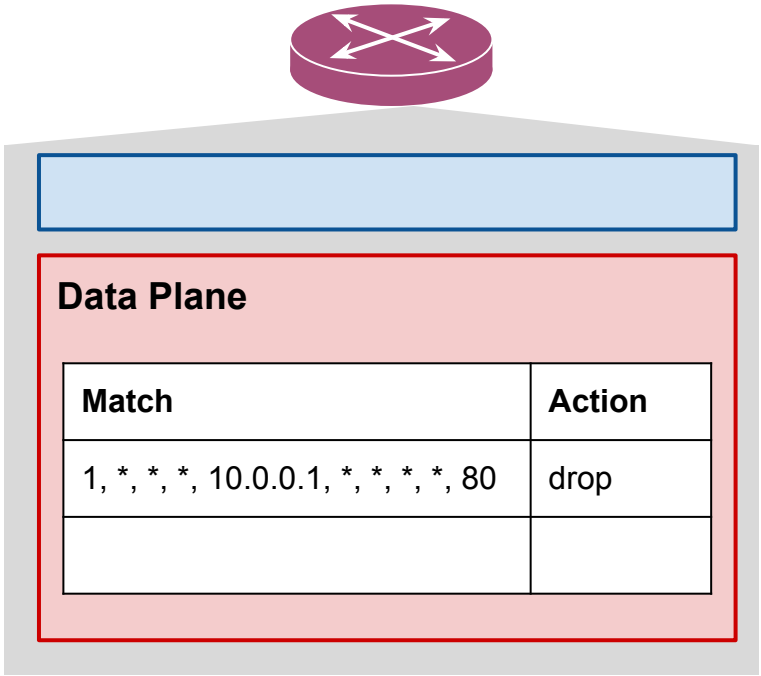
Logistics

- Presentations were assigned yesterday
- Reviews are due **Monday at 5pm**.
- Project proposal is due **Jan 31**.
    - There will be a dropbox on LEARN for submitting proposals.

So for in programmable networks…

- **2005:** 4D
  - Separating the "decision" plane from the data plane
- **2008:** OpenFlow
  - A simple yet general protocol for controller-switch communication
  - Abstracts the switch data plane as one big look-up table
- **2011:** Frenetic
  - Domain-specific network programming language
  - Raising the level of abstraction for network programming

OpenFlow started simple…



- Match
  - Input port
  - Ethernet header fields (src, dst, type)
  - Some IP header fields (src, dst, proto)
  - Some TCP header fields (src port, dst port)
- Action
  - drop
  - forward to port N
  - send to controller
  - modify the value of a field

But it grew more complex (and quickly)!

- More fields

Version	Date	Header Fields
OF 1.0	Dec 2009	12 fields (Ethernet, TCP/IPv4)
OF 1.1	Feb 2011	15 fields (MPLS, inter-table metadata)
OF 1.2	Dec 2011	36 fields (ARP, ICMP, IPv6, etc.)
OF 1.3	Jun 2012	40 fields
OF 1.4	Oct 2013	41 fields

Table 1: Fields recognized by the OpenFlow standard*

* From "P4: Programming Protocol-Independent Packet Processors", SIGCOMM CCR 2014

But it grew more complex (and quickly)!

- More fields
- Multiple tables

Version	Date	Header Fields
OF 1.0	Dec 2009	12 fields (Ethernet, TCP/IPv4)
OF 1.1	Feb 2011	15 fields (MPLS, inter-table metadata)
OF 1.2	Dec 2011	36 fields (ARP, ICMP, IPv6, etc.)
OF 1.3	Jun 2012	40 fields
OF 1.4	Oct 2013	41 fields

Table 1: Fields recognized by the OpenFlow standard*

* From "P4: Programming Protocol-Independent Packet Processors", SIGCOMM CCR 2014

Why multiple tables?

- Suppose you want to assign VLAN tags based on source and destination MAC addresses.

N entries

Match			**Action**
src MAC	dst MAC	everything else	
A	B	*	vlan = 2
C	B	*	vlan = 3
…	…	…	…

Why multiple tables?

- Now you also want to forward packets based on their source and destination IP address.

N entries

Match			**Action**
src MAC	dst MAC	everything else	
A	B	*	vlan = 2
C	B	*	vlan = 3
…	…	…	…

+

M entries

Match			**Action**
src IP	dst IP	everything else	
X	Y	*	outport = 1
Z	W	*	outport = 5
…	…	…	…

Why multiple tables?

- Now you also want to forward packets ba
  destination IP address.

You can write separate programs in, say, Frenetic, and compose them.

But what would the final OpenFlow table look like?

N entries

Match			Action
src MAC	dst MAC	everything else	
A	B	*	vlan = 2
C	B	*	vlan = 3
…	…	…	…

M entries

Match			Action
src IP	dst IP	everything else	
X	Y	*	outport = 1
Z	W	*	outport = 5
…	…	…	…

Why multiple tables?

M x N entries

Match					**Action**
src MAC	dst MAC	src IP	dst IP	everything else	
A	B	X	Y	*	vlan = 2; outport = 1
C	B	X	Y	*	vlan = 3; outport = 1
A	B	Z	W	*	vlan = 2; outport = 5
C	B	Z	W	*	vlan = 3; outport = 5
…	…	…	…	…	…

Why multiple tables?

- Network devices have multiple tables.
- For simplicity and uniformity, OpenFlow abstracted away everything as a single table.
- Why not allow the controller to program separate network logic in separate tables?

M x N entries

Match					**Action**
src MAC	dst MAC	src IP	dst IP	everything else	
A	B	X	Y	*	vlan = 2; outport = 1
C	B	X	Y	*	vlan = 3; outport = 1
A	B	Z	W	*	vlan = 2; outport = 5
C	B	Z	W	*	vlan = 3; outport = 5
…	…	…	…	…	…

OpenFlow kept getting extended

- To support more fields
- To expose more of the data-plane capabilities to the control-plane
  - e.g., multiple tables

OpenFlow kept getting extended

- To support more fields
- To expose more of the data-plane capabilities to the control-plane
  - e.g., multiple tables

This does not seem sustainable…

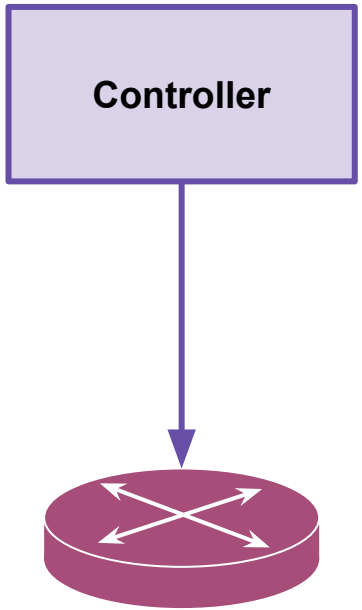
OpenFlow kept getting extended

- To support more fields
- To expose more of the data-plane capabilities to the control-plane
  - e.g., multiple tables

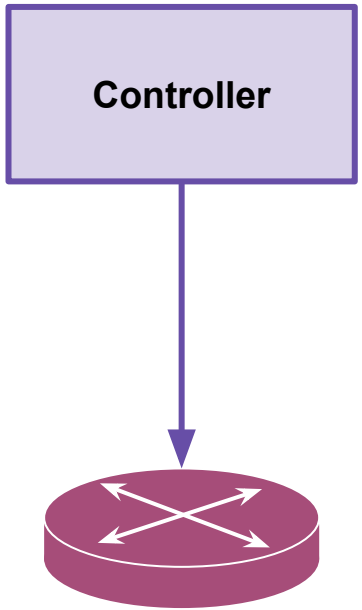
This does not seem sustainable…

Why don't we open up the controller-switch interface even more?

Controller to switch

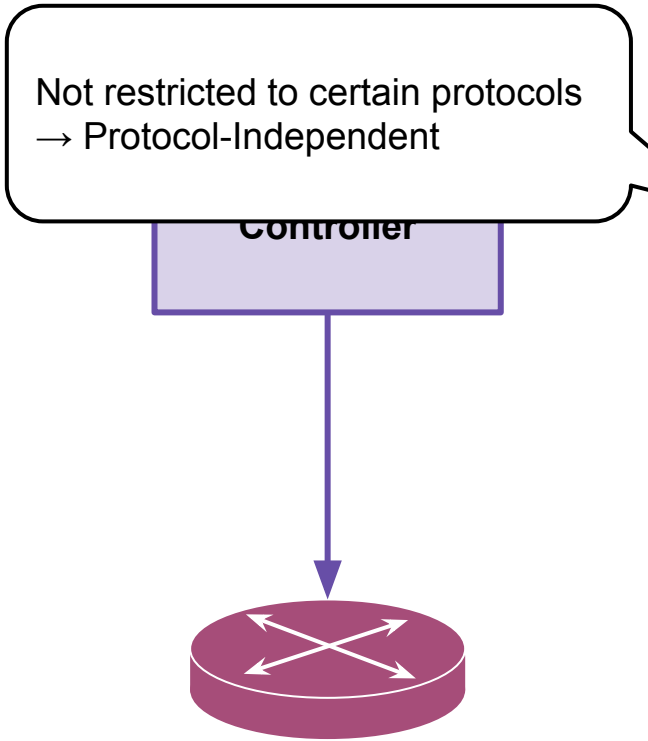


- **Runtime communication**
  - add/remove/modify table entries
  - send packet
  - request traffic statistics



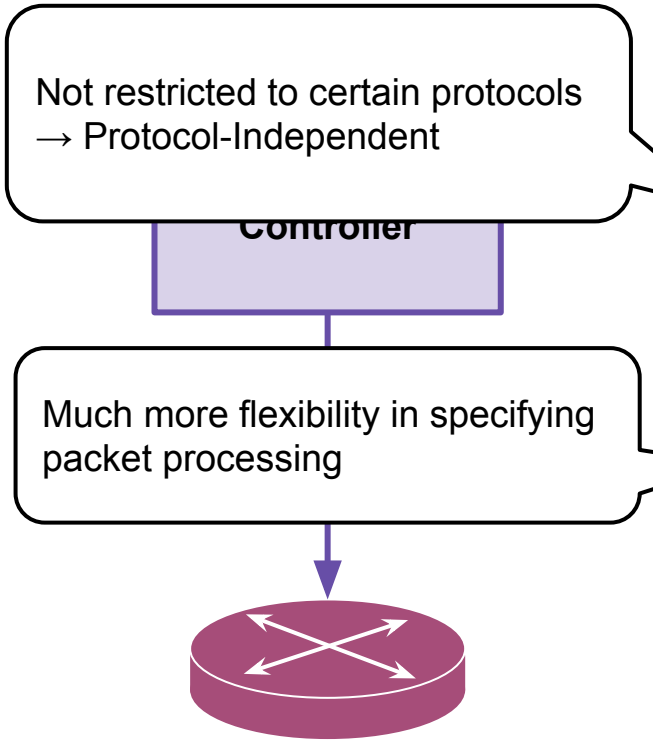
Controller to switch

- **Headers and Parsing**
  - Header X and Y look like this
  - To parse header X, look at the bytes B1 to B2 in the packet…
- **Table Configuration**
  - Table T1 should use X for match and A1 or A2 for actions.
  - Table T2 should use …
- **Runtime communication**
  - add/remove/modify table entries
  - send packet
  - request traffic statistics



Controller to switch

- **Headers and Parsing**
  - Header X and Y look like this
  - To parse header X, look at the bytes B1 to B2 in the packet…
- **Table Configuration**
  - Table T1 should use X for match and A1 or A2 for actions.
  - Table T2 should use …
- **Runtime communication**
  - add/remove/modify table entries
  - send packet
  - request traffic statistics



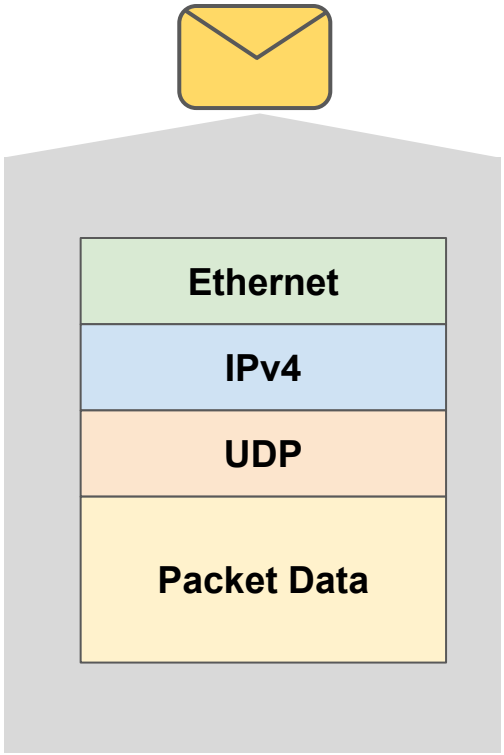
Controller to switch

- **Headers and Parsing**
  - Header X and Y look like this
  - To parse header X, look at the bytes B1 to B2 in the packet…
- **Table Configuration**
  - Table T1 should use X for match and A1 or A2 for actions.
  - Table T2 should use …
- **Runtime communication**
  - add/remove/modify table entries
  - send packet
  - request traffic statistics

P4: **P**rogramming **P**rotocol-Independent **P**acket **P**rocessors

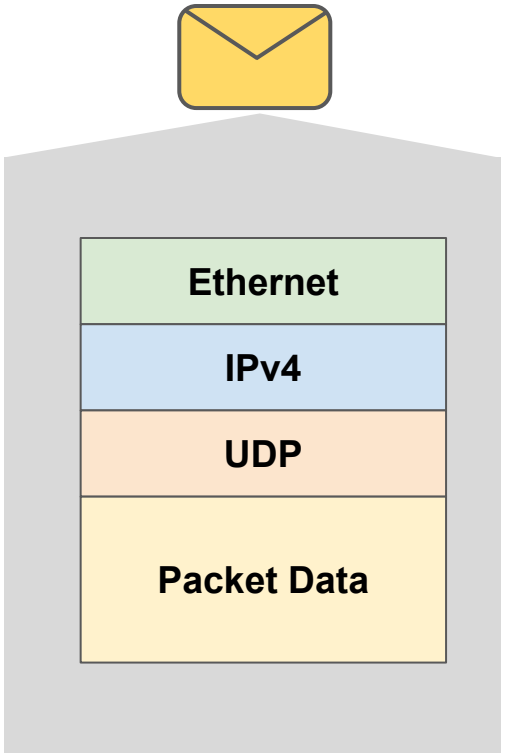
- A data-plane programming language (proposed in 2014)
- P4 programs specify
  - Headers and Parsing
  - Match-action tables
  - How packets are processed in the data plane using those tables

Example: Destination-based IP forwarding



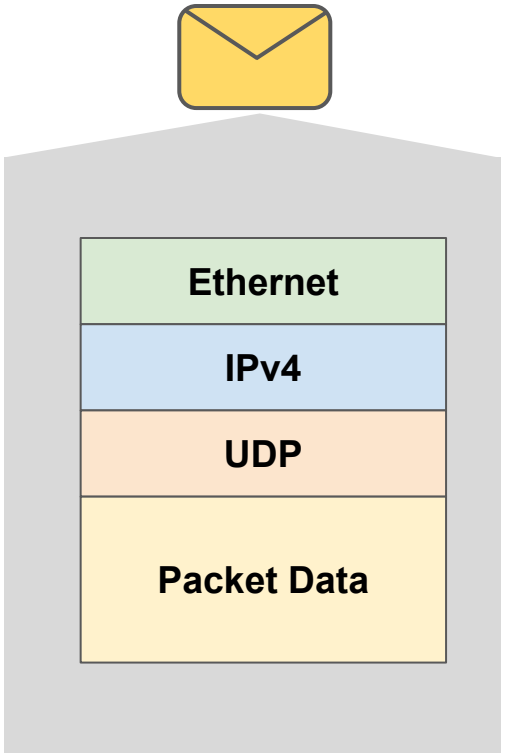
- Decide which port to forward the packet to based on the destination IP address.
- Destination IP is a field supported by OpenFlow.
- Can be implemented using OpenFlow rules.

Example: Destination-based IP forwarding in OpenFlow



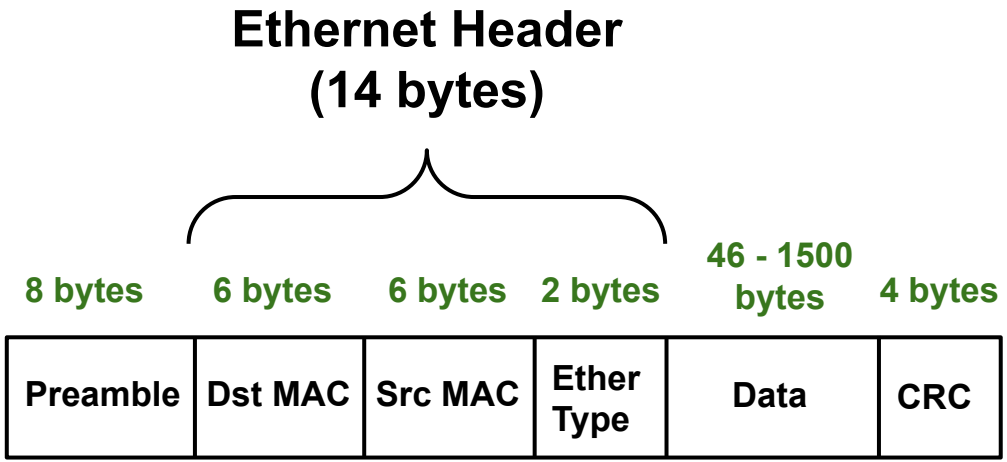
Match		Action
dst IP	everything else	
125.12.1.25/24	*	outport = 1
140.2.33.22/32	*	outport = 5
…	…	…

Example: Destination-based IP forwarding in P4



1. Define the headers that we need for processing incoming packets.
    - The Ethernet and IP header
    - Don't need anything else beyond that
2. Define how they should be parsed from the packet

Defining the Ethernet header



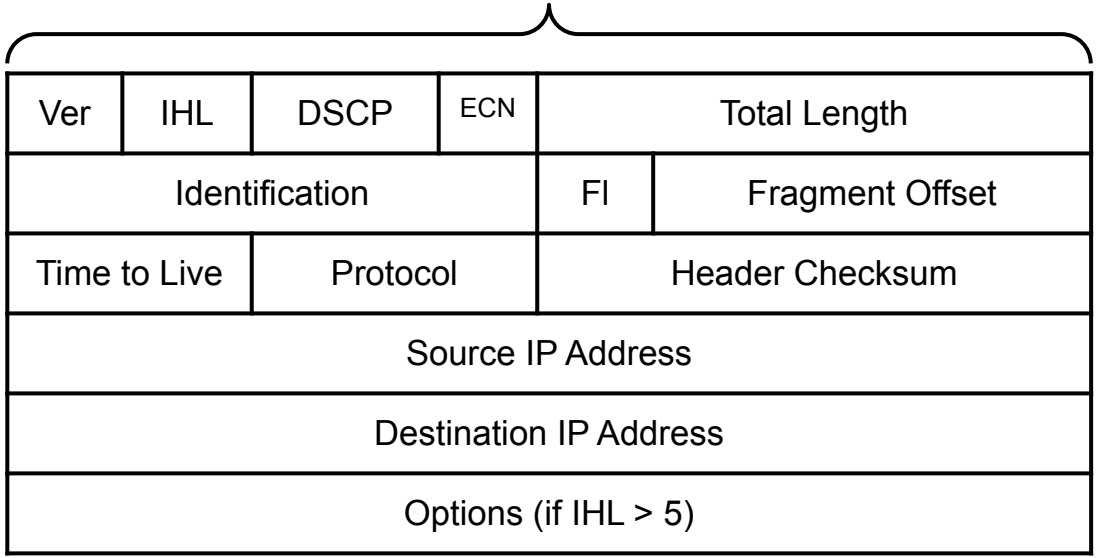
Definition in P4

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

Defining the IPv4 header

IPv4 Header

4*5 = 20 bytes (or more with options)



Definition in P4

```
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;
    bit<32> srcAddr;
    bit<32> dstAddr;
}
```

Instantiating the headers

- Next, we need to instantiate the headers. In this case, we expect our packets to have one Ethernet header and one IP header.

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

Instantiating the headers

- We can have multiple instances of a header if needed (e.g., IP in IP tunneling)

```
struct headers {
    ethernet_t ethernet;
    ipv4_t outer_ipv4;
    ipv4_t inner_ipv4;
}
```

Metadata

- Metadata are extra variables that accompany the packet as it is processed in the switch.
- You can read from and write to them in different parts of a P4 program.
- You can define your own metadata or use special ones that the underlying target makes available to you.

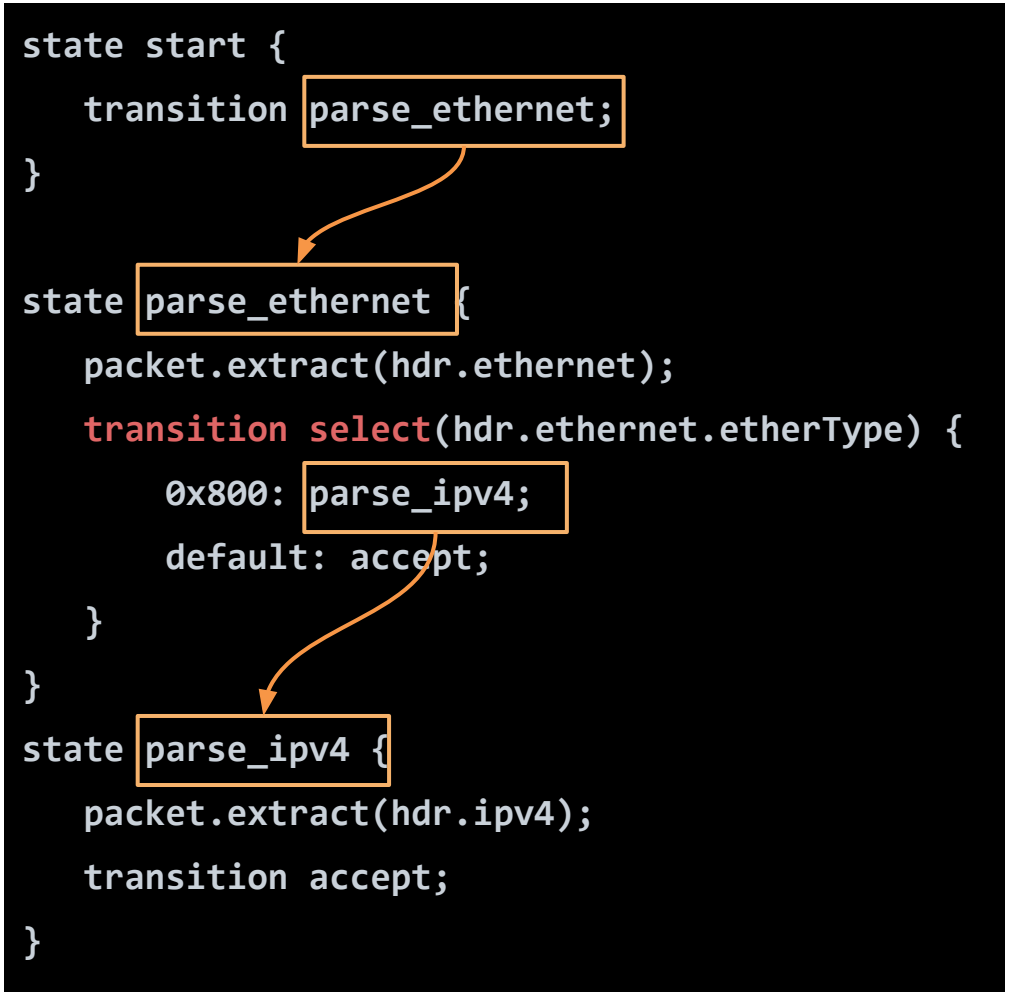
Parsing headers

- P4 parsers are state machines.
- The parser starts from the "start" state and transitions to user-defined states as it parses bits from the packet and puts them into headers.

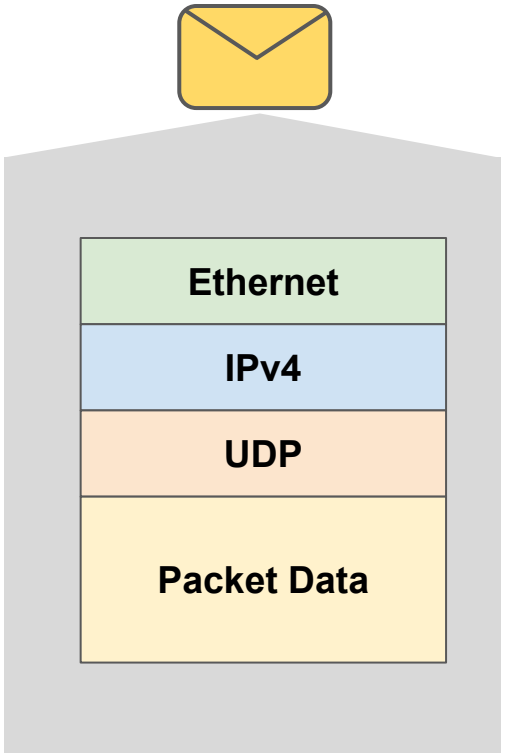
```
parser MyParser(packet_in packet,
                out headers hdr,
                inout metadata meta,
                inout standard_metadata_t standard_metadata){

    state machine describing how to parse headers
}
```

Parser States

- `extract` takes bits out of the packet and put them in the header instances.
- With `select`, we can pick which state to `transition` to next based on the other "variables" in the program.



Example: Destination-based IP forwarding in P4



- We have extracted bits from the packet into headers.
- We can read from and write to these headers in control blocks.

Controls

The body of a control looks like a simple imperative program.

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

    Declarations (e.g., tables, actions, etc.)

    apply {
        if (hdr.ipv4.isValid()) {
            ipv4_forward.apply();
        }
    }
}
```

Controls

You can declare variables, tables, and actions.

You can specify how these tables and actions should be applied to packets.

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

    Declarations (e.g., tables, actions, etc.)

    apply {
        if (hdr.ipv4.isValid()) {
            ipv4_forward.apply();
        }
    }
}
```

Defining Tables

- `key` specifies the set of fields that are used for matching
- `actions` specifies the set of possible actions that can be applied to packets in this table.

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

Defining Tables

- To add a rule to the table, the controller should specify
  - values for the match fields
  - which action to take for matched packets

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

Defining Actions

- Actions can modify metadata, packet fields, etc.
- When adding a rule, the controller should
  - specify which action to take
  - provide the arguments for that action

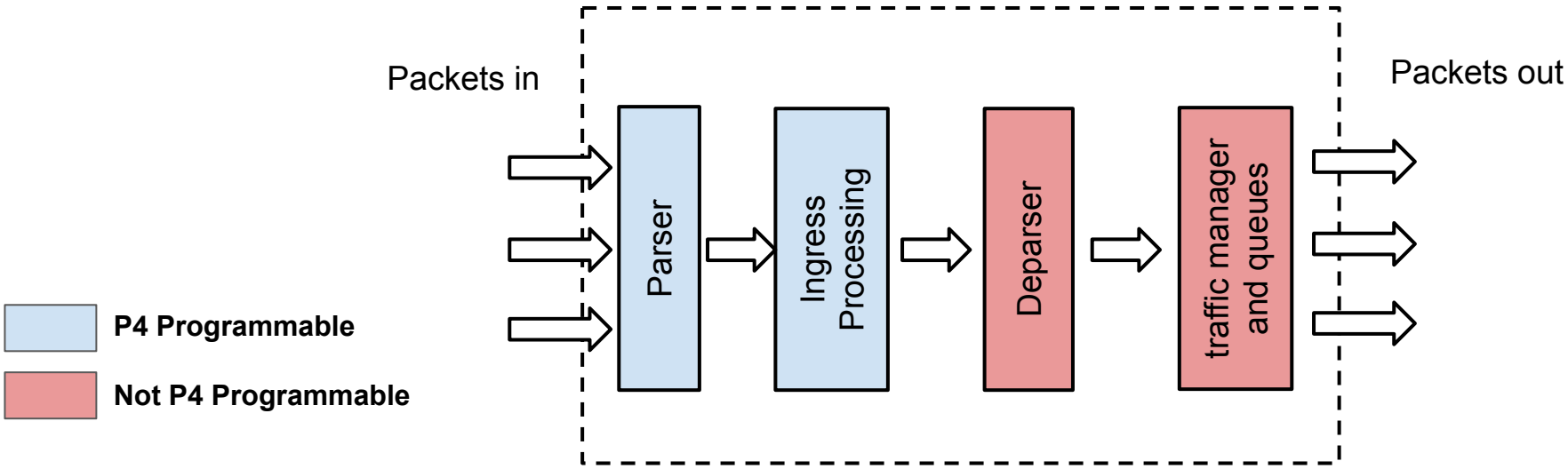
```
action drop() {
    mark_to_drop(standard_metadata);
}

action forward(macAddr_t dstAddr, egressSpec_t port) {
    standard_metadata.egress_spec = port;
    hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
    hdr.ethernet.dstAddr = dstAddr;
    hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
}
```

Tables & Actions in Controls

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

    action drop() { … }

    action forward(macAddr_t dstAddr, egressSpec_t port) {...}

    table ipv4_forward {
       …
    }
    apply {
        if (hdr.ipv4.isValid()) {
            ipv4_forward.apply();
        }
    }
}
```

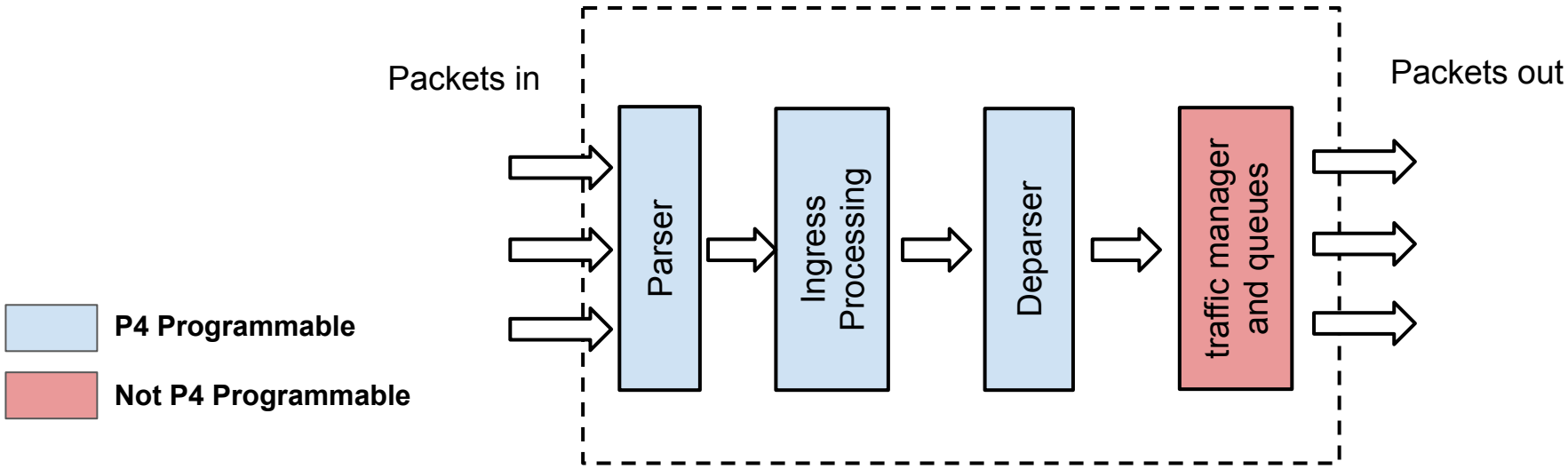
Architectures

- Data planes can have very different architectures and/or allow varying levels of programmability.



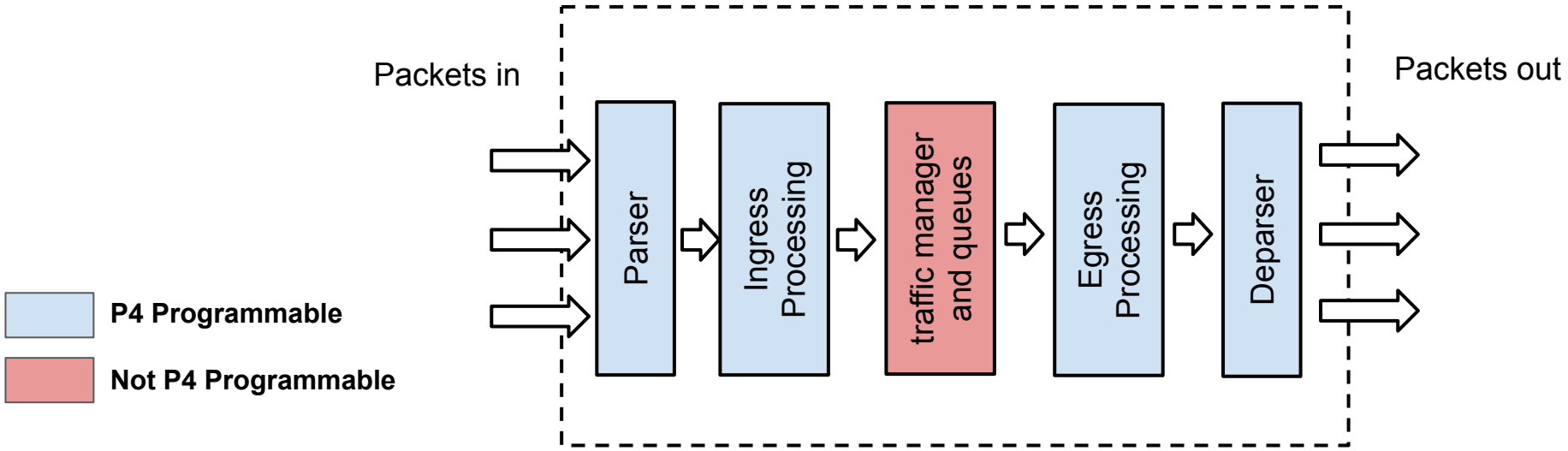
Architectures

- Data planes can have very different architectures and/or allow varying levels of programmability.



Architectures

- Data planes can have very different architectures and/or allow varying levels of programmability.



Architectures

- P4 architecture files describe the P4 programmable blocks in a data plane and their interface with the programmer.

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

Architectures

- P4 architecture files describe the P4 programmable blocks in a data plane and their interface with the programmer.

```
parser Parser<H, M>(packet_in b,
                    out H parsedHdr,
                    inout M meta,
                    inout standard_metadata_t standard_metadata);

control Ingress<H, M>(inout H hdr,
                      inout M meta,
                      inout standard_metadata_t standard_metadata);
```

Architectures

- They specify what kind of standard metadata they make available to the programmers.

```
struct standard_metadata_t {
    bit<9>      ingress_port;
    bit<9>      egress_spec;
    bit<9>      egress_port;
    bit<32>     instance_type;
    bit<32>     packet_length;
    ...
}
```

Architectures

- They also specify any "special" block that is not programmable but can be used as a blackbox in P4 programs.
- You can think of it as a special library of objects and functions.

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

extern void random<T>(out T result, in T lo, in T hi);
```

Let's look at actual programs

- forwarding.p4
- v1model.p4

What about the control plane?

- P4 programs do not specify the dynamics of how rules are added, modified, or removed from tables.
- The controller still needs to
  - populate the tables in P4 programs,
  - get statistics, or
  - send/receive packets to/from the data plane.
- Can't use OpenFlow out of the box.
  - The table definitions change from one program to another

P4 Runtime

- A controller platform for targets whose behavior is described by P4 programs
- Provides libraries in common programming languages to communicate with P4 switches.
  - similar to OpenFlow controllers (e.g., NOX)

P4 Today

- Since 2014, the community around P4 has only grown.
- It has seen widespread adoption by industry and academia
- Many papers that either use P4 for various new applications or improve the language itself and its compilers.
  - You'll see P4 pop up many times in papers in this class :)
- It has found applications in many places
  - Prototyping new hardware features
  - Offloading all sorts of functionality to the switch
  - Being used as a specification language for fixed-function switches
  - …

Paper 1: P4: Programming Protocol-Independent Packet Processors

- The original P4 paper, published in 2014
- The language has evolved since then, but the main philosophy and language constructs are mainly the same.
- In 2013, a subset of the authors published a paper (which we will read later!) on a switch architecture that is more reconfigurable than OpenFlow.
- So, this paper assumes that compiling P4 programs to actual hardware is possible and mostly focuses on language design.

Paper 2: The $P4_{16}$ Programming Language

- P4 was a great starting point for making data planes programmable.
- But, there was room for lots of improvement from a language design standpoint :)
- In 2017, the P4 language consortium released a major update to the P4 language.
- This paper describes the arguments behind the changes, the new version of the language, and sketches of a compiler design.

Additional Resources

- P4.org!
- Domino, Mantis, MicroP4, and P4All
  - Proposals for higher level data plane programming languages
  - Or extensions to P4

Logistics

- Presentations were assigned yesterday
- Reviews are due **Monday at 5pm**.
- Project proposal is due **Jan 31**.
    - There will be a dropbox on LEARN for submitting proposals.