The Evolution of Software-Defined Networking: From Academic Roots to Industry Revolution

Marc Bruyere @ IIJ Lab

IIJ
Internet Initiative Japan

Agenda

- **Nothing is created ex-nihilo "何も虚無から創造されることはない"**
  - An historical overview
  - Protocol Independent Switch Architecture and P4 Lang
- **A short P4 tutorial**
  - PISA – P4 Key elements
  - L3 routing short introduction
  - Learning by yourself
- **A couple of projects from IIJ lab**
  - The IIJ Lab testbed
  - P4 HolistIX : IXP switching fabric with a zero-touch approach
  - UmbrellaDC : Data-Center switching fabric fast flow completion - no overhead
  - Gandalf : Packet amplification and nano-sec latency timestamping
- **Outcomes and conclusion**

Open Signaling (OPENSIG)

- **Year: 1995**
- Initiated to make networks (ATM, Internet, mobile) more open and programmable.
- Aimed for separation between communication hardware and control software.
- Led to the development of the General Switch Management Protocol (GSMP), concluded in June 2002.
- **From : ITU-T**

Active Networking

- **Mid-1990s**
- Proposed programmable network infrastructure for customized services.
- Two main approaches: user-programmable switches and capsules (program fragments in messages).
- Despite considerable activity, it faced challenges in widespread use and deployment.

Towards an Active Network Architecture (1996),

David L. Tennenhouse, et al.,

Computer Communication Review ACM

ForCES

- **Early 2000s**
- An approach parallel to SDN, focusing on separation within network devices.
- **Key concepts for Separation of IP Control and Forwarding are described in RFC 3654 & 5810**
- **IETF WG : Forwarding and Control Element Separation**
- Ongoing standardization with various published documents including architectural framework and communication protocols.

Real adoption of the concepts

But no vendor have adopted ForCes RFCs

4D Project

- **Year: 2005**
- Advocated a clean slate design emphasizing separation between routing decision logic and network interaction protocols.
- Influenced the thinking towards decoupled control and data planes in networking.
- Inspired later works like NOX, proposing an "operating system for networks" for OpenFlow-enabled networks.
- Research on a radical design project aimed at shaping industry practices.

Albert Greenberg, Gisli Hjalmtysson, David A. Maltz, Andy Myers, Jennifer Rexford, Geoffrey Xie, Hong Yan, Jibin Zhan, and Hui Zhang. 2005. A clean slate 4D approach to network control and management. SIGCOMM Comput. Commun. Rev. 35, 5 (October 2005), 41–54. https://doi.org/10.1145/1096536.1096541

Ethane

- **Year: 2007**
- Predecessor to OpenFlow, designed for enterprise network architecture.
- Focused on centralized control for managing policy and security.
- Consisted of two components: a controller and an Ethane switch with a flow table and secure channel.
- Implemented on HW NetFPGA and NEC Ethernet switch with Broadcom ASIC and deployed at William Gates Building @ Stanford University.

Martin Casado, Michael J. Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. 2007. Ethane: taking control of the enterprise. SIGCOMM Comput. Commun. Rev. 37, 4 (October 2007), 1–12. https://doi.org/10.1145/1282427.1282382

The Open Networking Foundation

- **Year: 2011**
- A full networking ecosystem and growing community.
- Google adoption of OpenFlow.
- Various start-up created (Nicira Networks).
- 60 members in less than a year

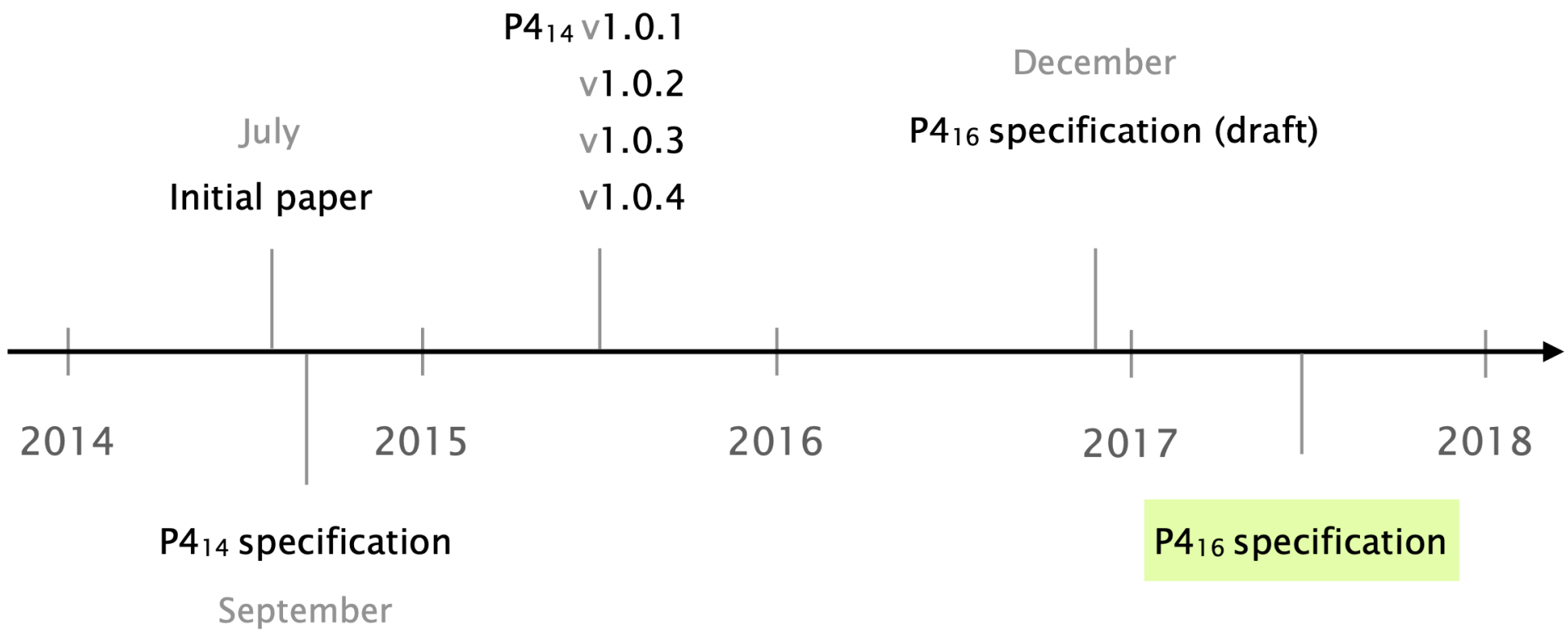
P4 Programming Protocol-Independent Packet Processors

- **Year: 2014**
- Enhances SDN flexibility (no fixed fields as in OF).
- A Protocol-Independent Switch Architecture
- Introduce a domain specific language with a compiler for programmable parsers and control flow mechanisms.
- Hardware target-independent

P. Bosshart *et al.*, “P4: programming protocol-independent packet processors,” *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014, doi: 10.1145/2656877.2656890.

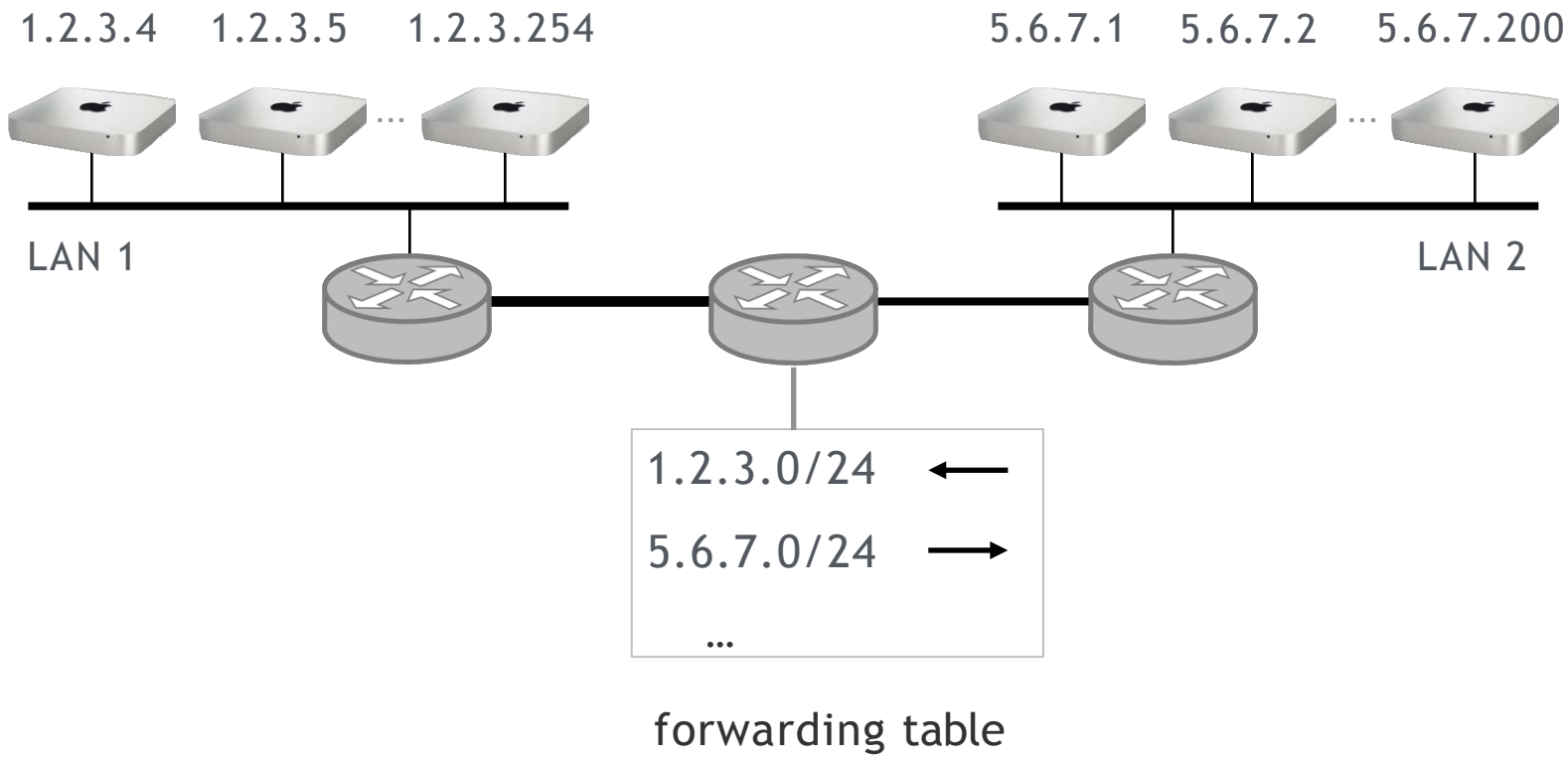
P4 Short Tutorial

- P4-LANG versions.
- How to program a simple L3 routing.
- Do yourself the full tutorial to learn how to implement. many others features and invent new ones.
- Thanks for the nice tutorial works from ETH Zurich.


P4$_{14}$ v1.0.1
v1.0.2
v1.0.3
v1.0.4
December
P4$_{16}$ specification (draft)
July
Initial paper
2014
2015
2016
2017
2018
P4$_{14}$ specification
September
P4$_{16}$ specification

IP forwarding

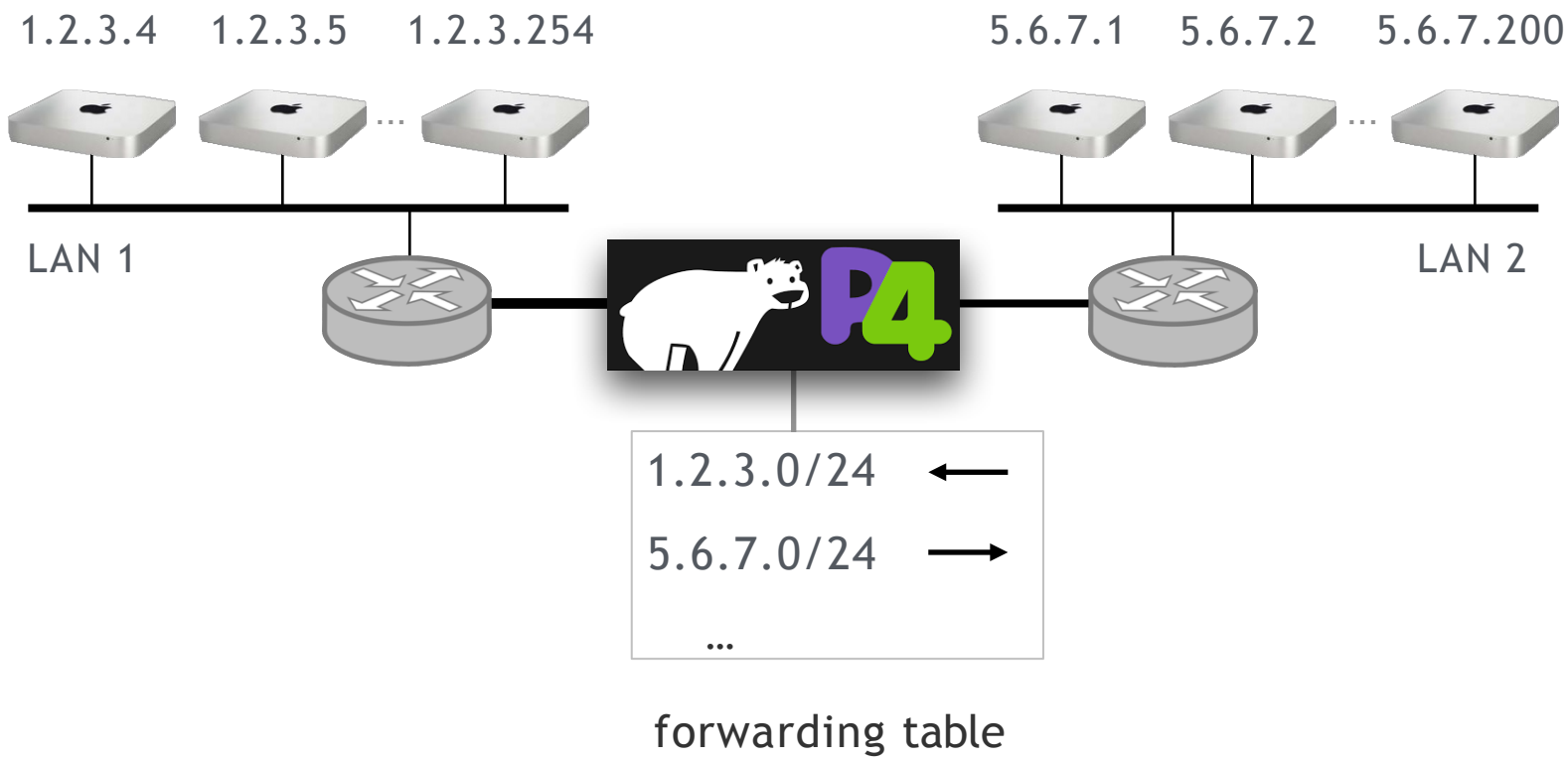
in a traditional router



forwarding table

IP forwarding

in P4

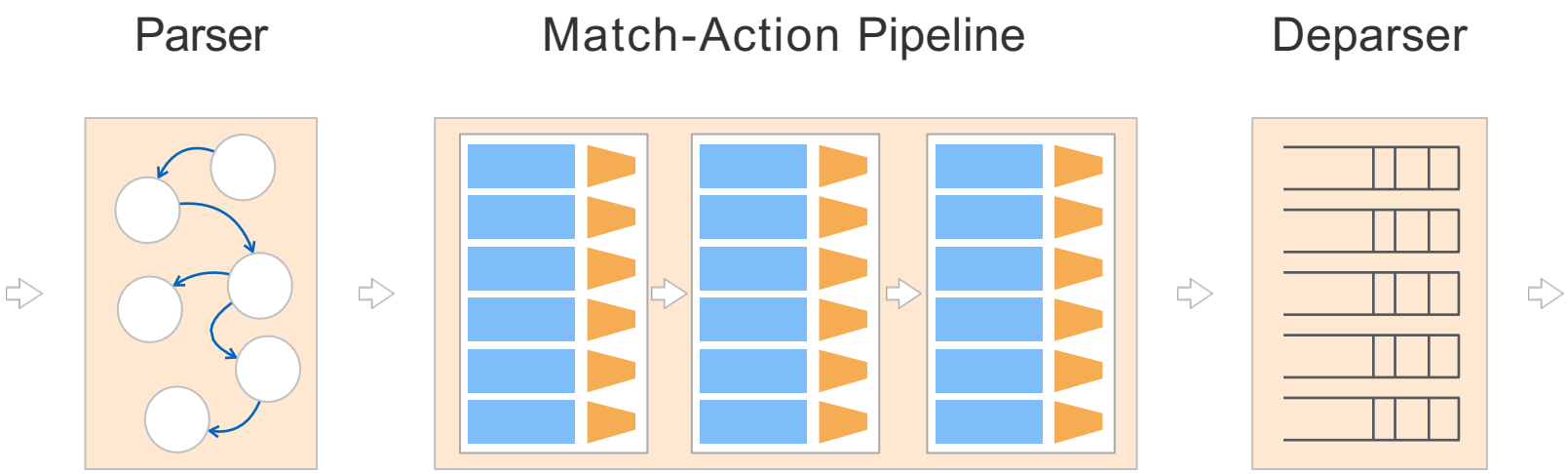


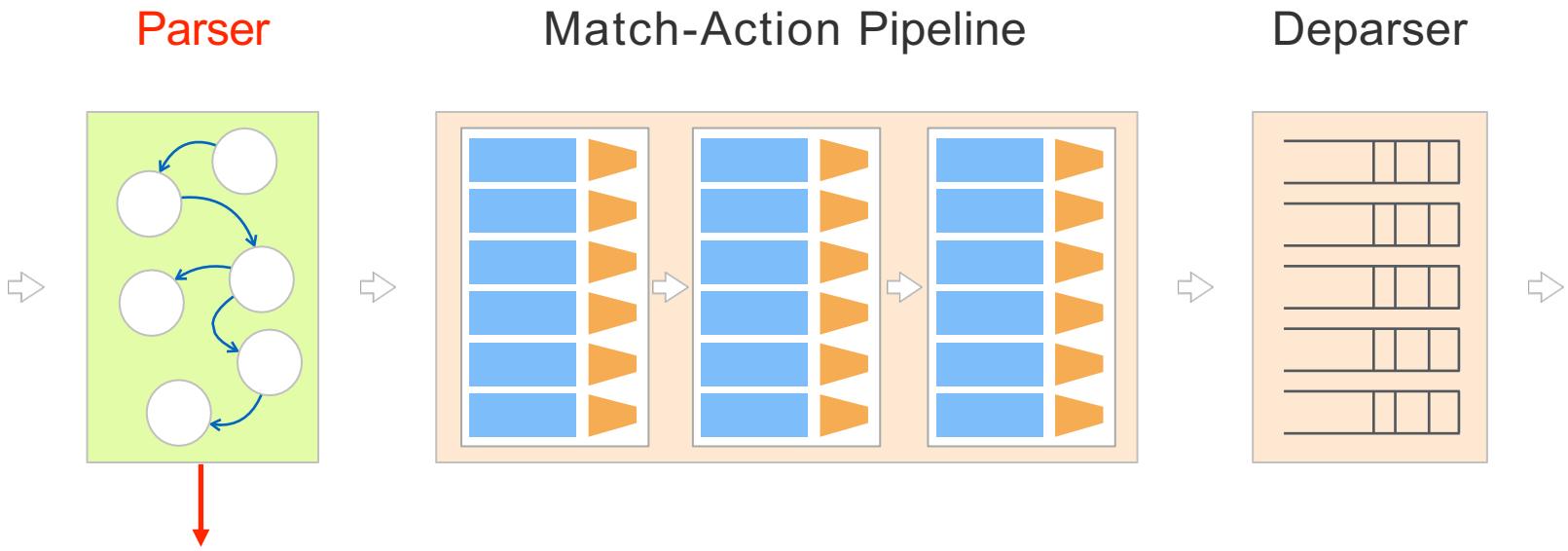
When forwarding an IP packet,
an IP router performs four actions:

- Lookup the next hop to use
- Update the MAC addresses
- Decrement the TTL
- Forward the packet to the output port

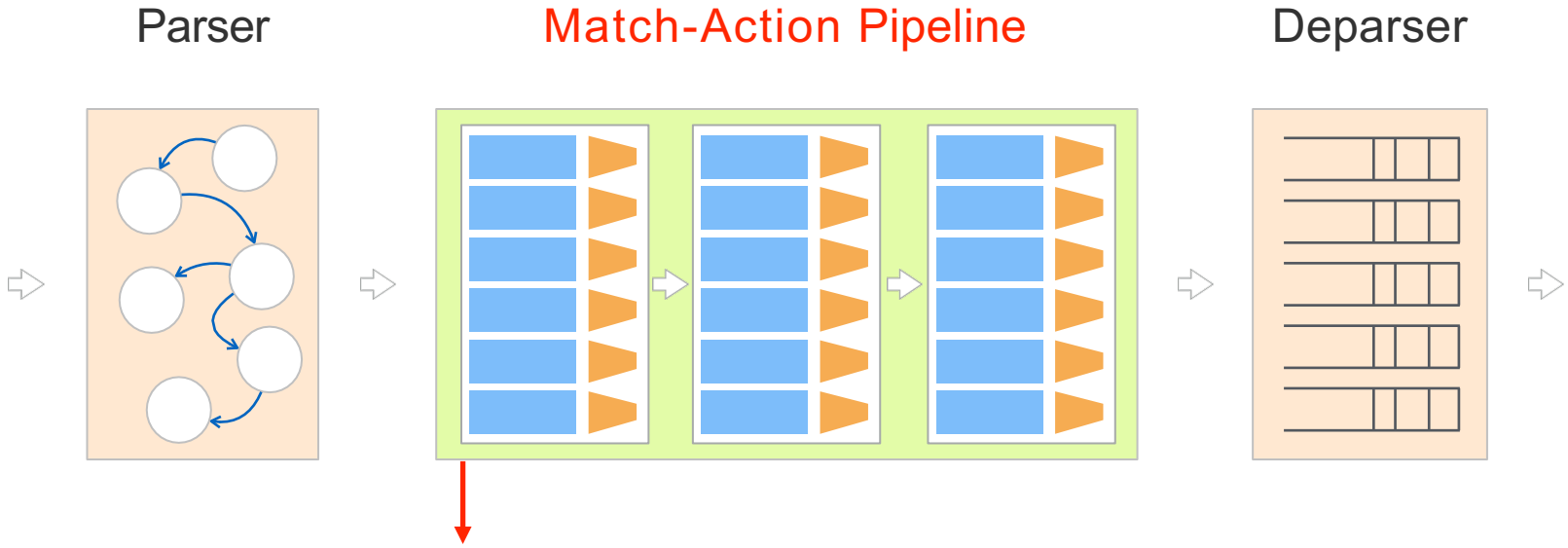
Each of them should be in implemented in P4

A P4 program consists of three basic parts

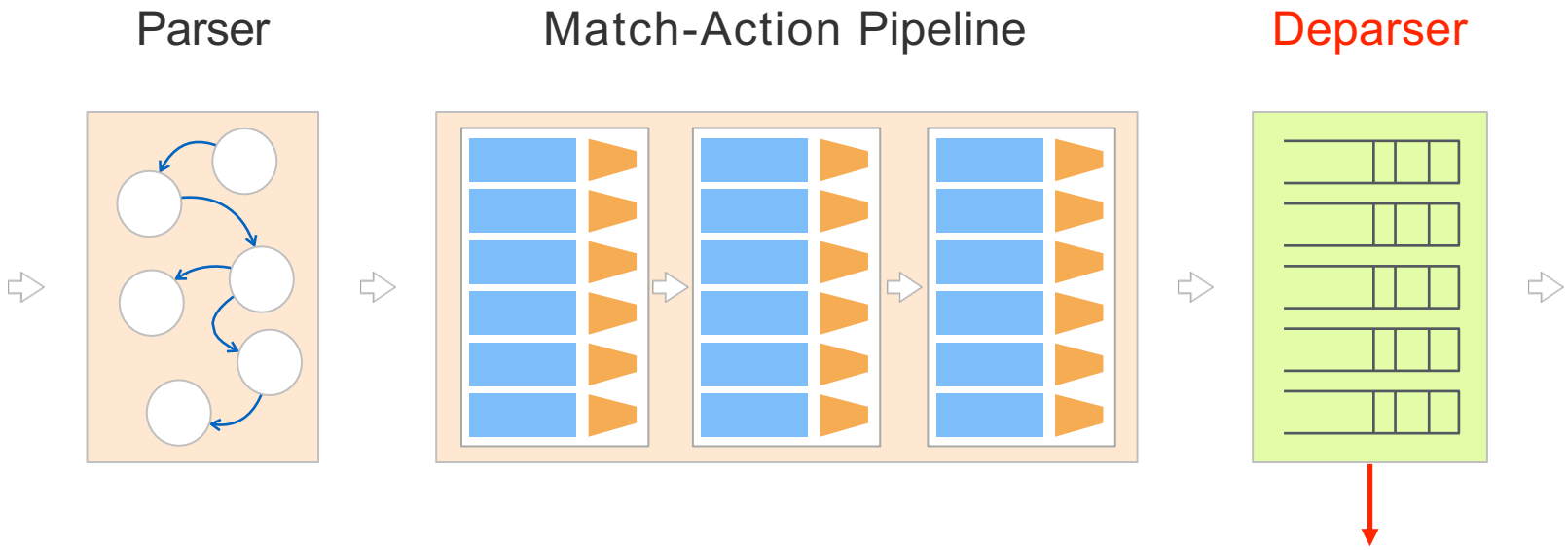

Parser
Match-Action Pipeline
Deparser

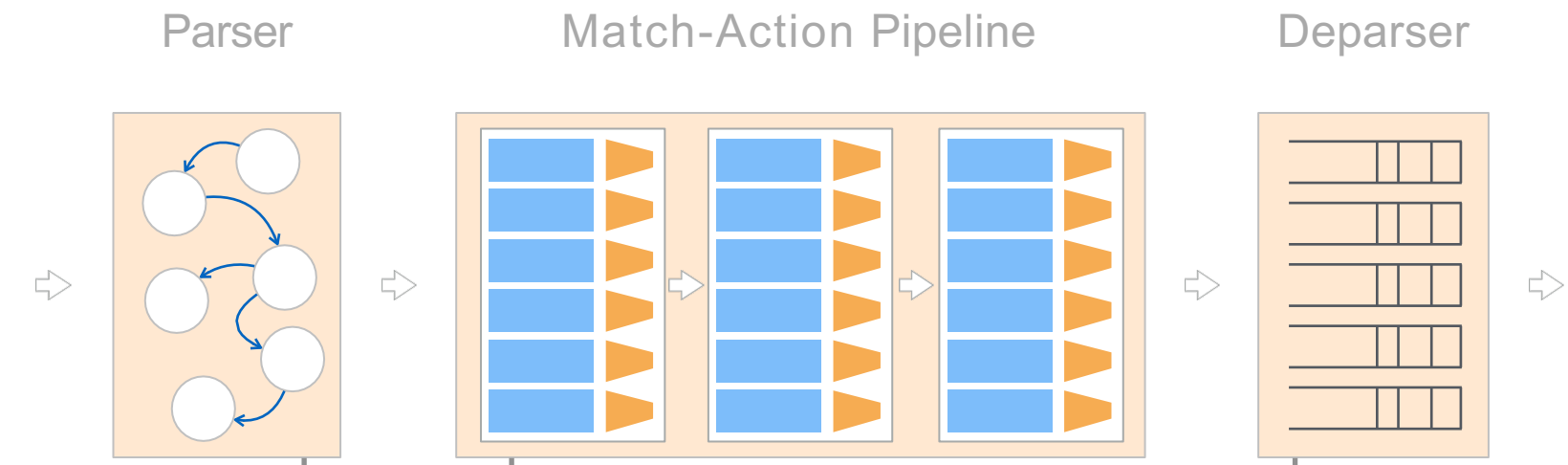


Specifies the headers to extract
from the incoming packet

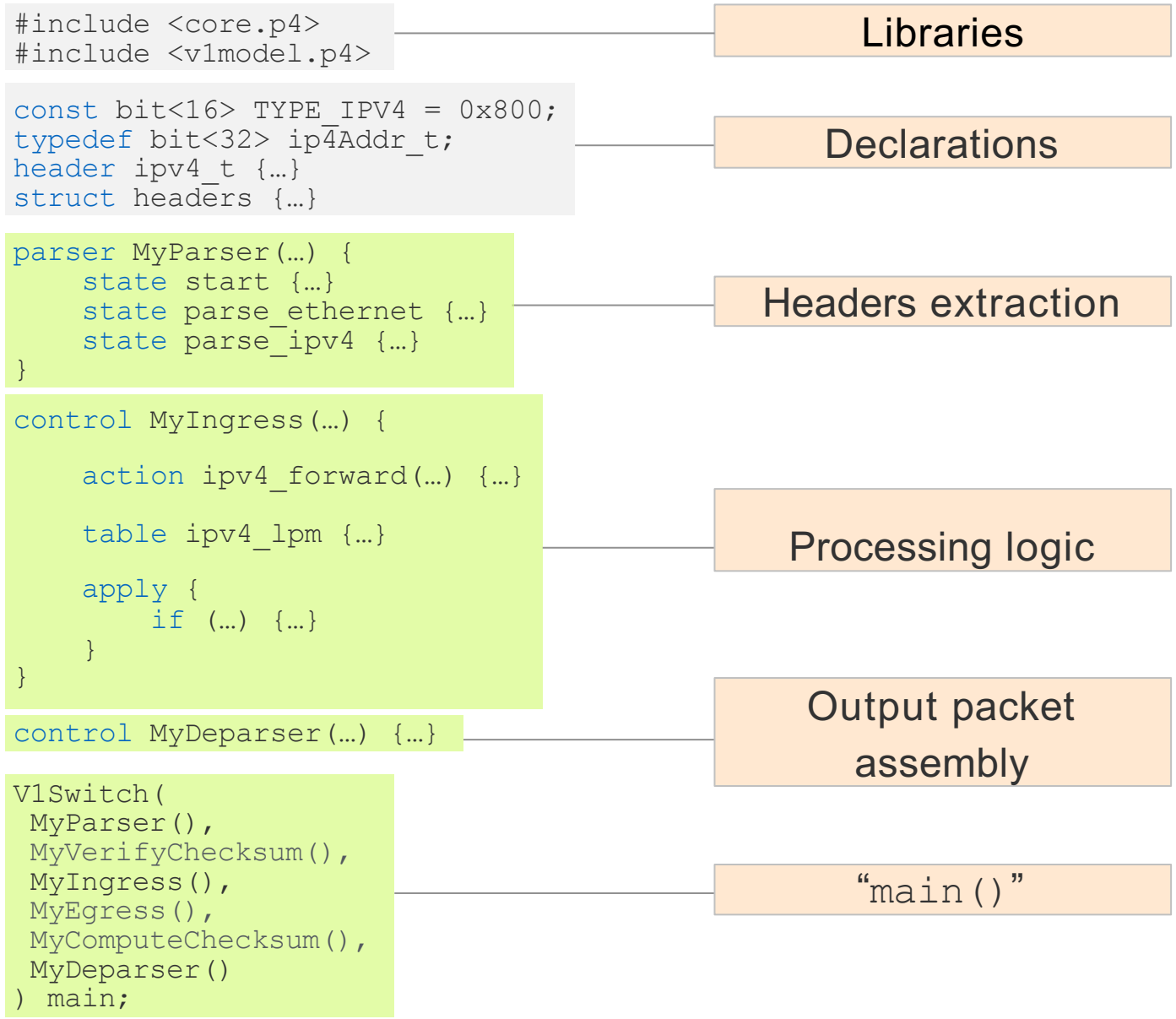


Specifies the processing logic to apply on the extracted headers

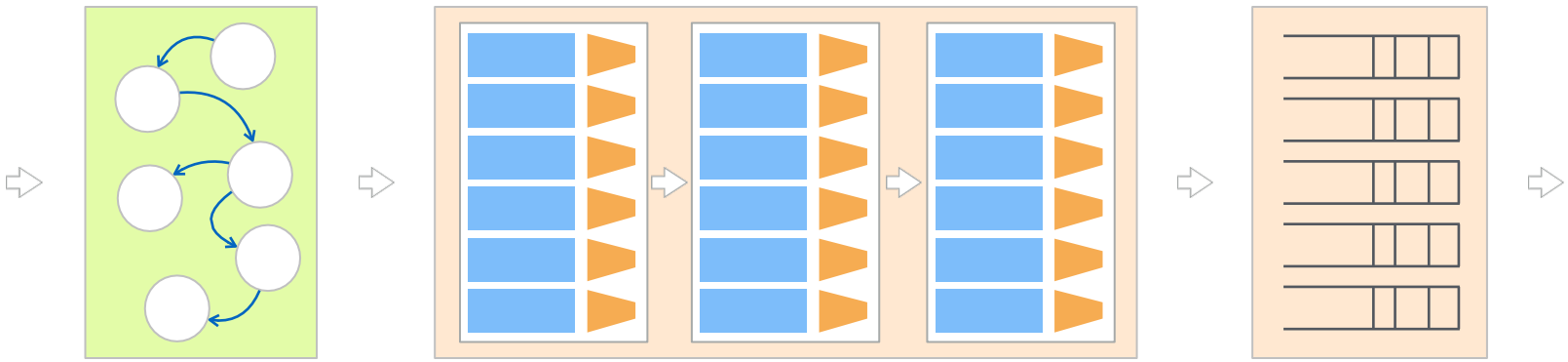

Parser
Match-Action Pipeline
Deparser
Specifies how
the output packet
will look on the wire



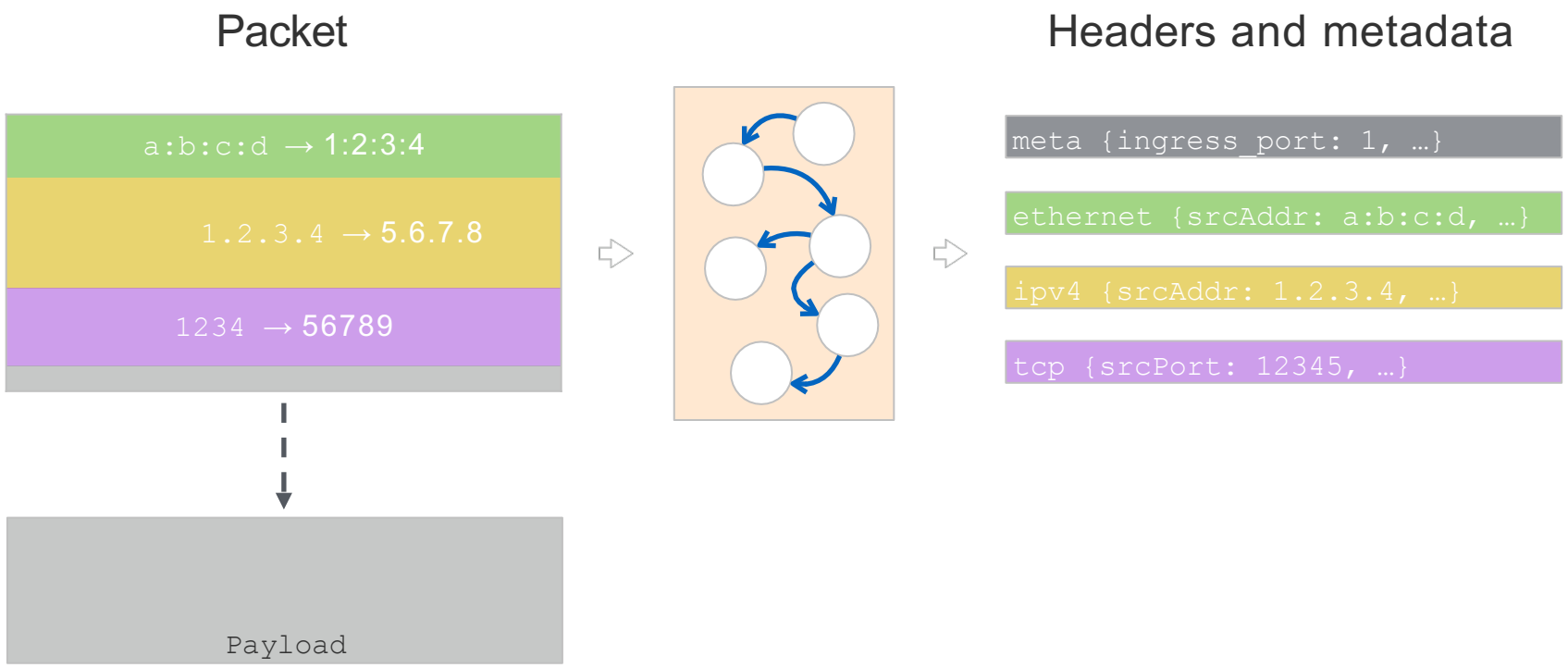

```
V1Switch(
 MyParser(),
 MyVerifyChecksum(),
 MyIngress(),
 MyEgress(),
 MyComputeChecksum(),
 MyDeparser()
) main;
```


#include <core.p4>
#include <v1model.p4>
Libraries
const bit<16> TYPE_IPV4 = 0x800;
typedef bit<32> ip4Addr_t;
header ipv4_t {…}
struct headers {…}
Declarations
parser MyParser(…) {
state start {…}
state parse_ethernet {…}
state parse_ipv4 {…}
}
Headers extraction
control MyIngress(…) {
action ipv4_forward(…) {…}
table ipv4_lpm {…}
apply {
if (…) {…}
}
}
Processing logic
control MyDeparser(…) {…}
Output packet assembly
V1Switch(
MyParser(),
MyVerifyChecksum(),
MyIngress(),
MyEgress(),
MyComputeChecksum(),
MyDeparser()
) main;
“main()”

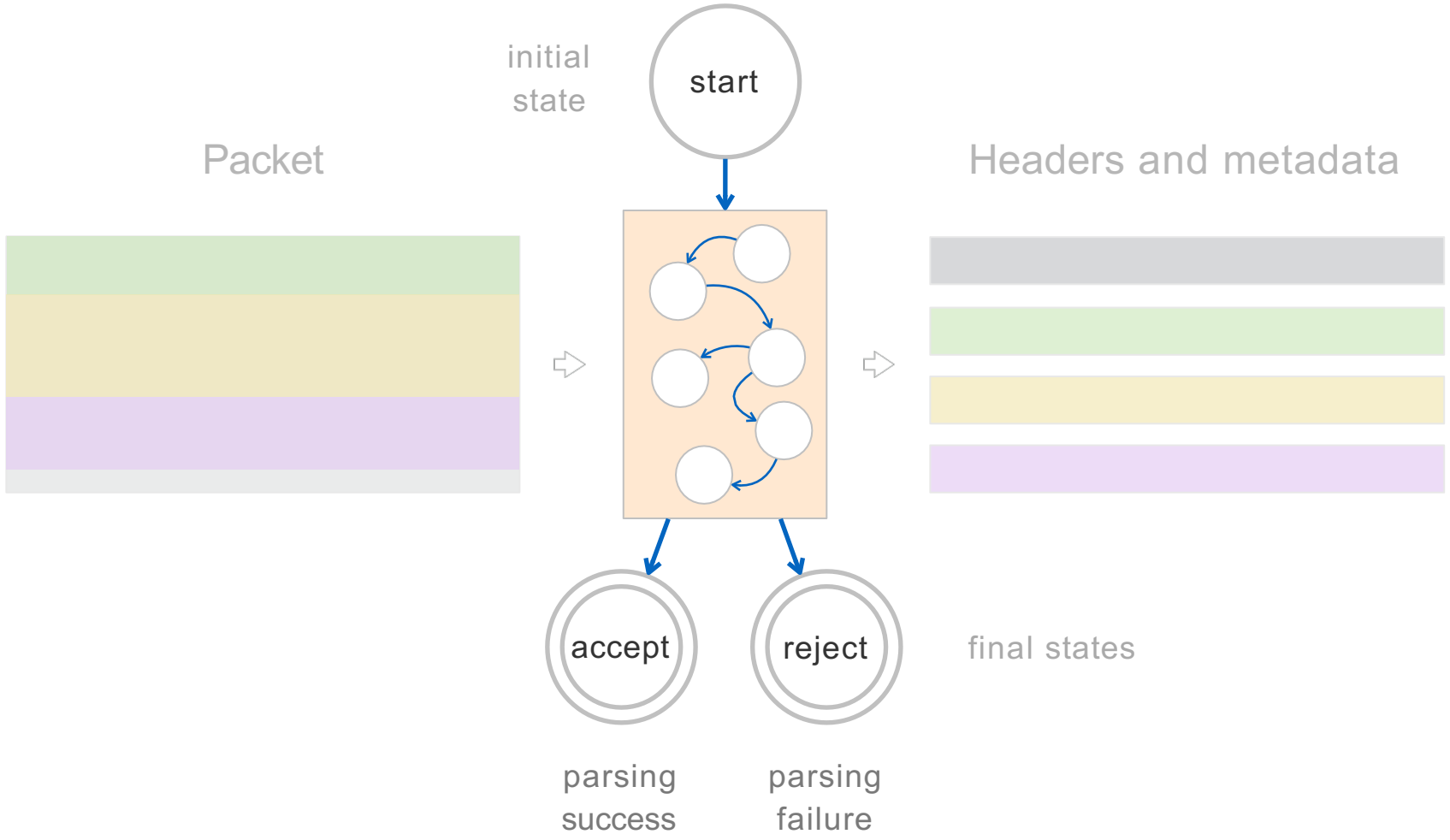
Parser

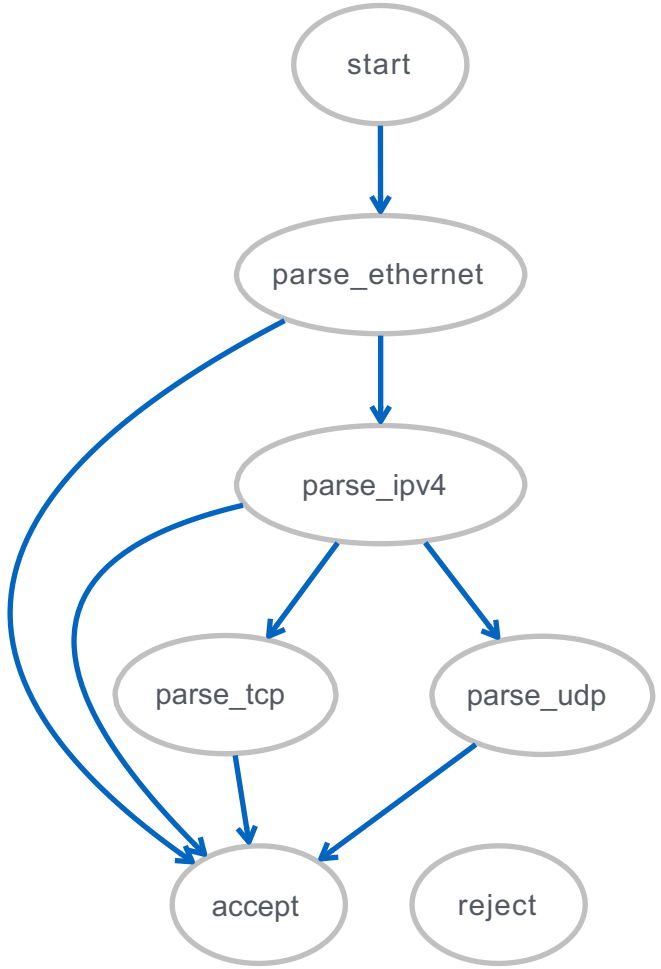


The parser relies on a programmable state machine which maps packets into headers and metadata



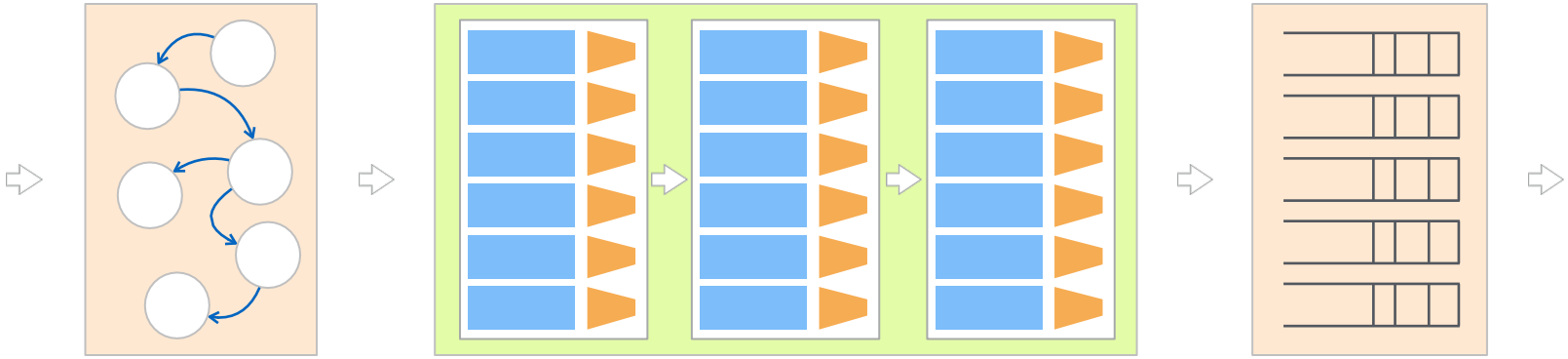
The state machine has three predefined states: start, accept and reject






```
parser MyParser(…) {
 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.protocol) {
   6: parse_tcp;
   17: parse_udp;
   default: accept;
  }
 }
 state parse_tcp {
  packet.extract(hdr.tcp);
   transition accept;
 }
 state parse_udp {
  packet.extract(hdr.udp);
   transition accept;
 }
}
```

Match-Action Pipeline



In the match-action pipeline,
control blocks manipulate headers and metadata

A control block is an imperative program
which describes how to process packets

A control block is an imperative program which describes how to process packets

Headers and metadata from parser

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

 bit<9> port;

 apply {
  port = 1
  std_meta.egress_spec = port;
  hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
  hdr.ethernet.dstAddr = 0x2;
  hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
 }
}
```

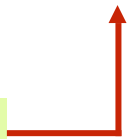
Variable declaration

Control flow

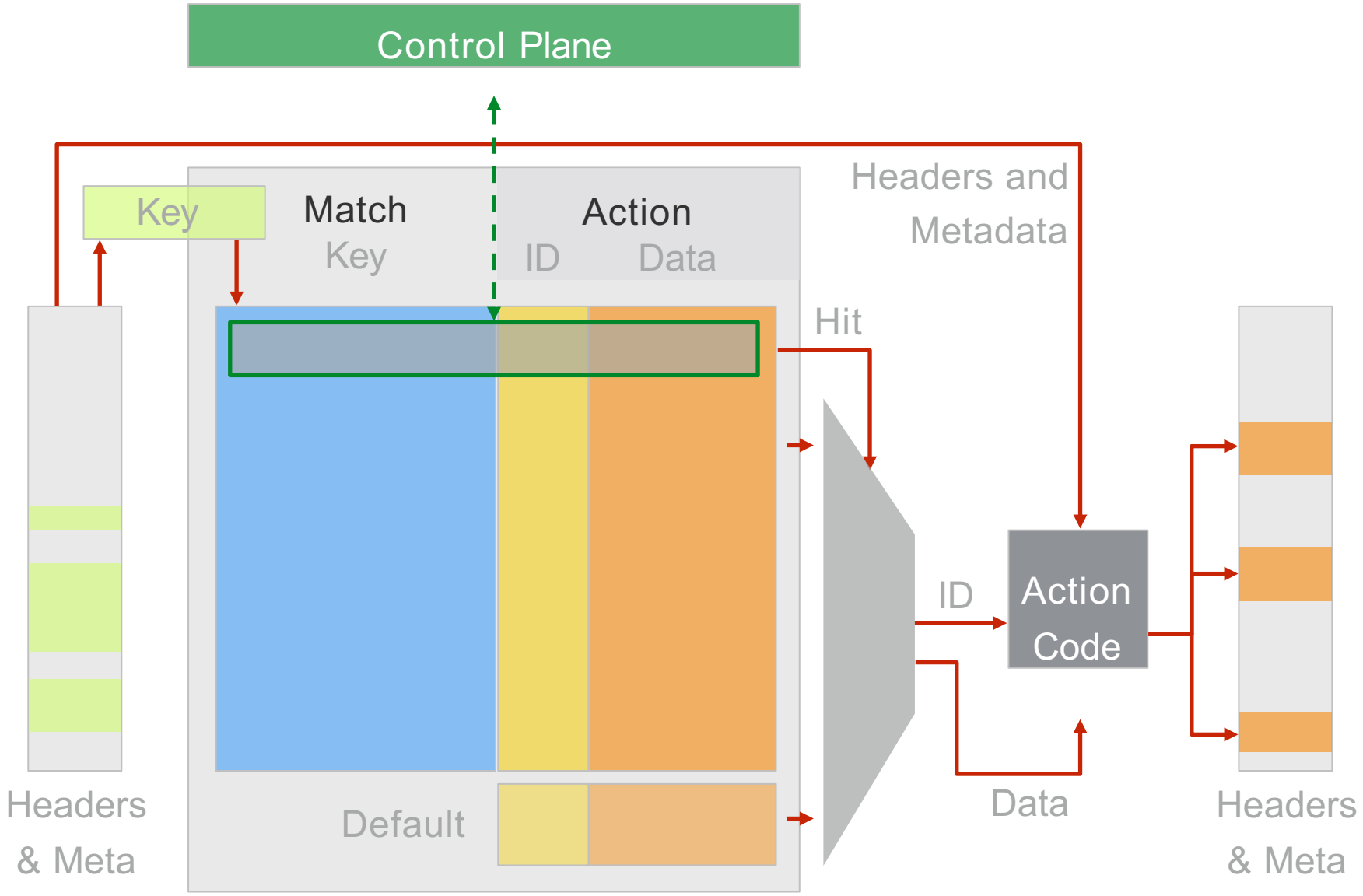
Actions allow to re-use code

in a block or globally, they are similar to C functions

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

 action ipv4_forward(macAddr_t dstAddr,
                     egressSpec_t port) {
  std_meta.egress_spec = port;
  hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
  hdr.ethernet.dstAddr = dstAddr;
  hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
 }

 apply {
  ipv4_forward(0x123, 1);
 }
}
```



Control blocks often rely on tables which map specific headers values to specific actions


Control Plane
Key
Match
Key
Action
ID
Data
Headers and
Metadata
Hit
ID
Action
Code
Data
Default
Headers
& Meta
Headers
& Meta

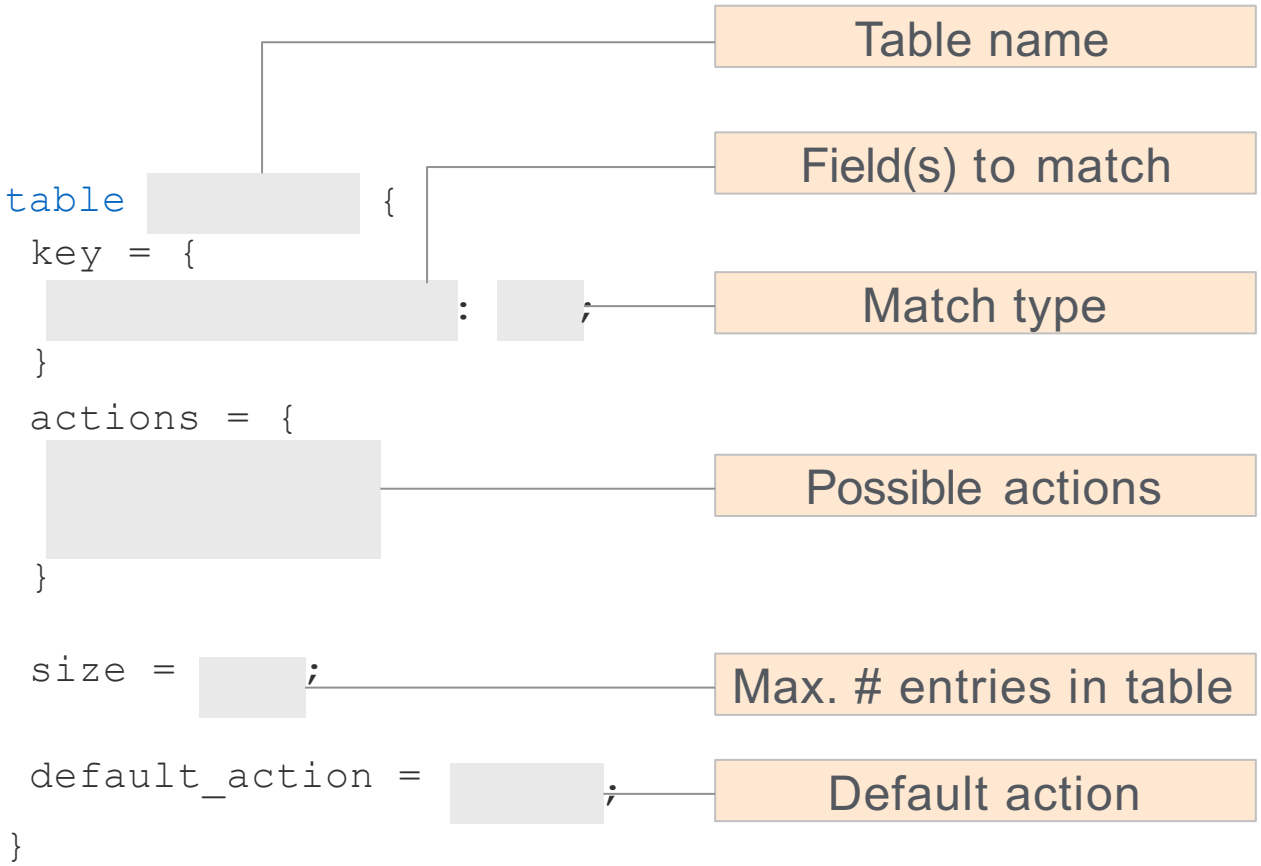
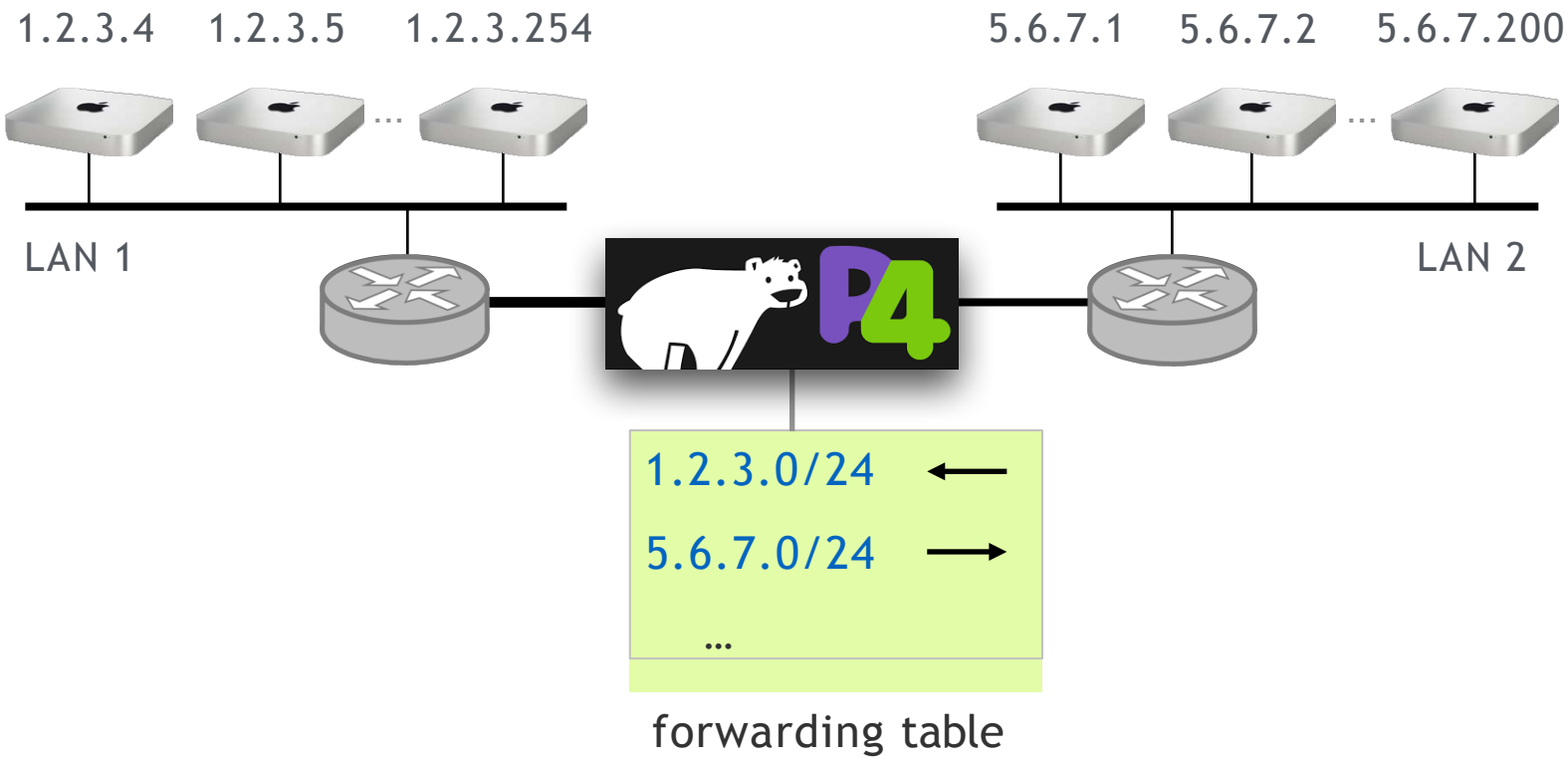
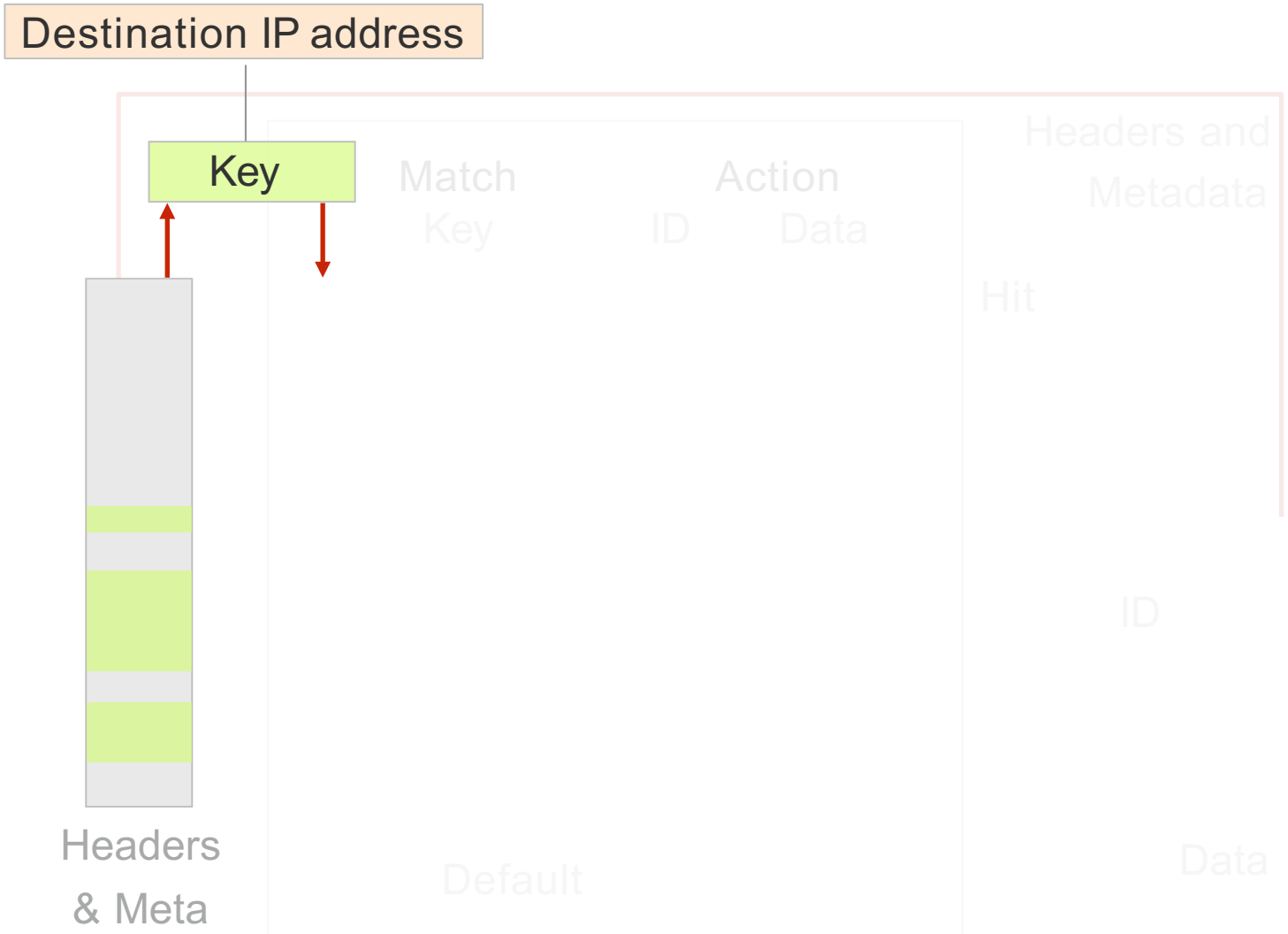
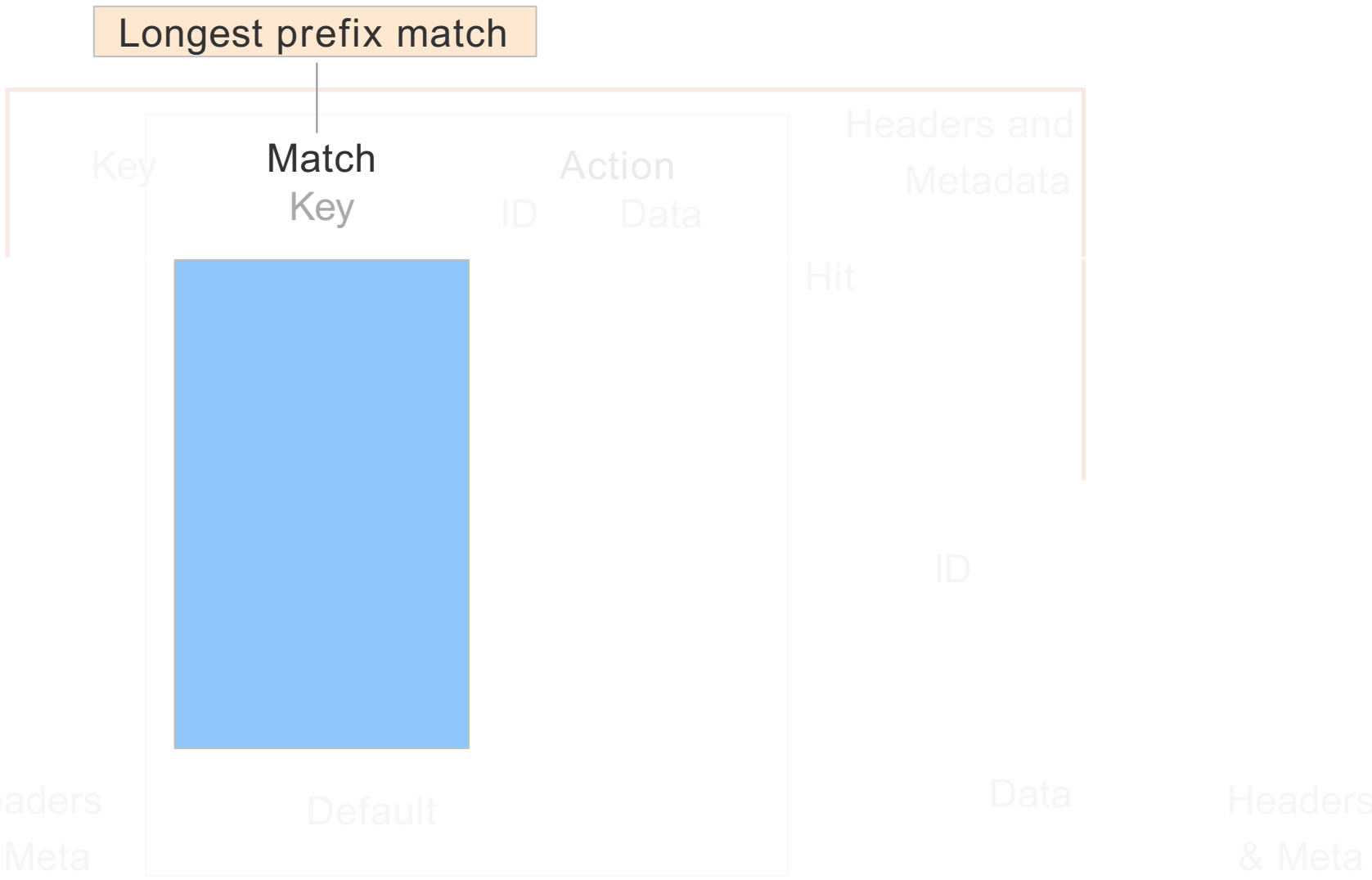


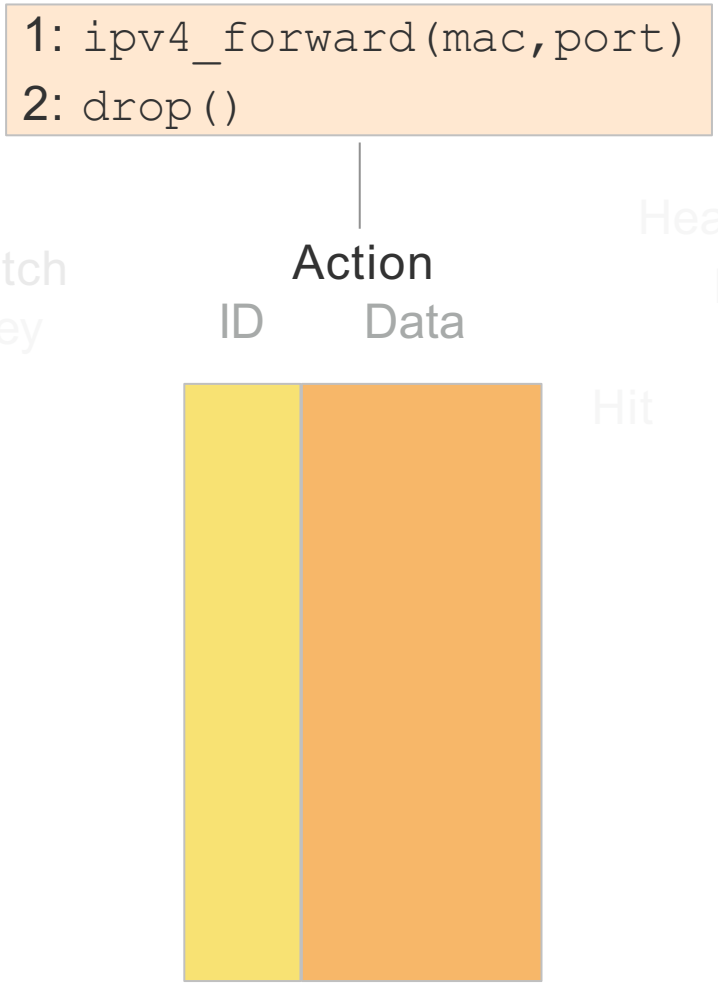
table {
key = {
: ;
}
actions = {
}
size = ;
default_action = ;
}
Table name
Field(s) to match
Match type
Possible actions
Max. # entries in table
Default action

Example: IP forwarding table




Destination IP address
Key
Headers
& Meta


Longest prefix match
Match
Key
Key
Action
ID
Data
Headers and
Metadata
Hit
ID
Default
Data
Headers
& Meta
Headers
& Meta


1: ipv4_forward(mac,port)
2: drop()
Action
ID
Data

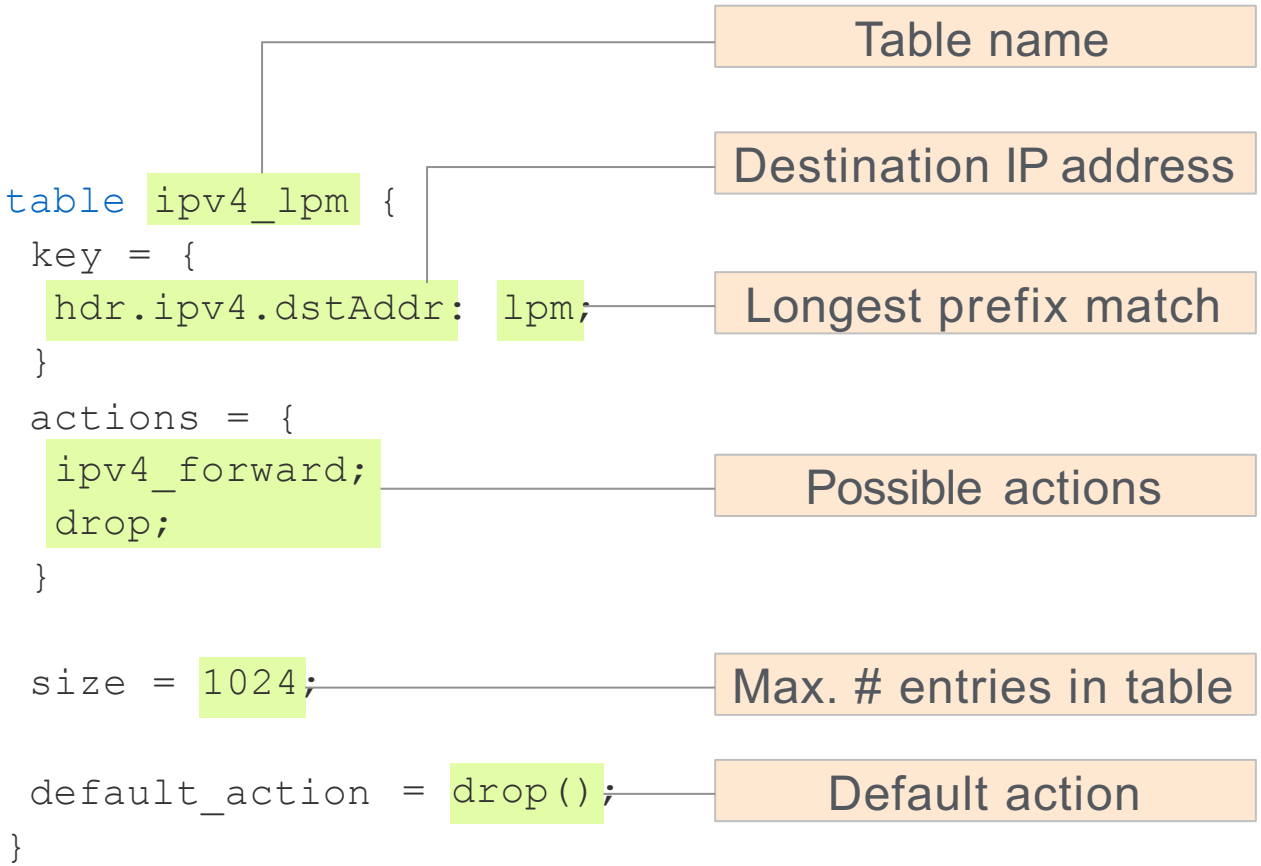
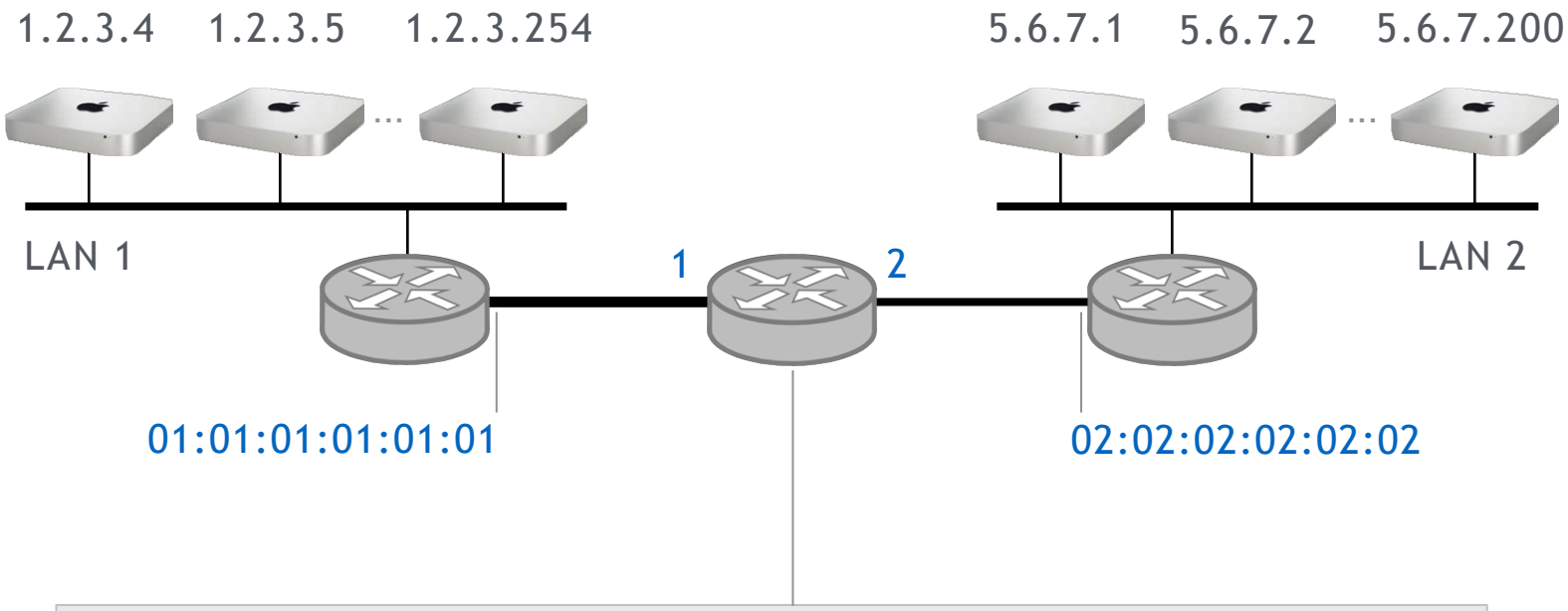


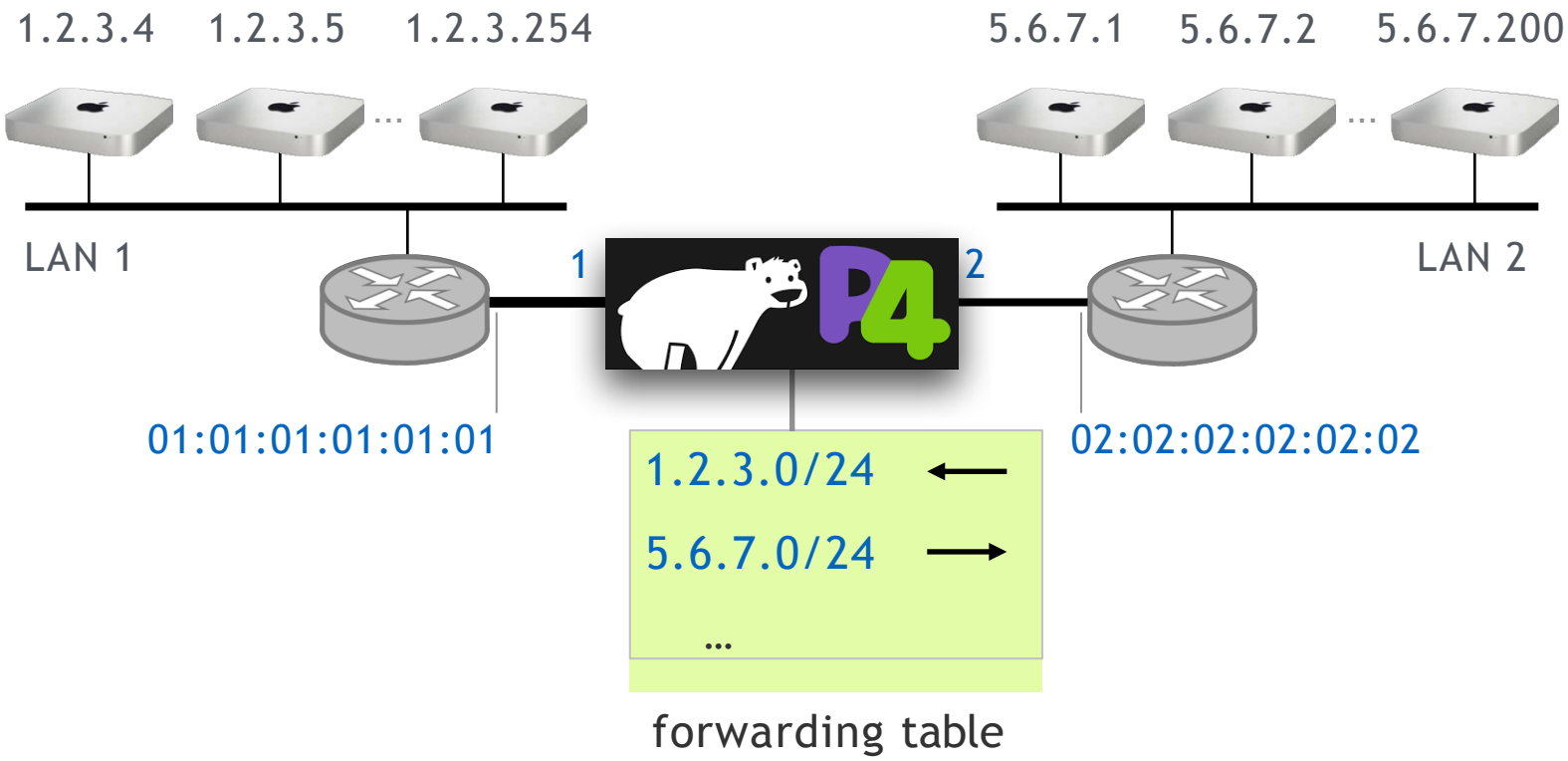
table ipv4_lpm {
key = {
hdr.ipv4.dstAddr: lpm;
}
actions = {
ipv4_forward;
drop;
}
size = 1024;
default_action = drop();
}
Table name
Destination IP address
Longest prefix match
Possible actions
Max. # entries in table
Default action

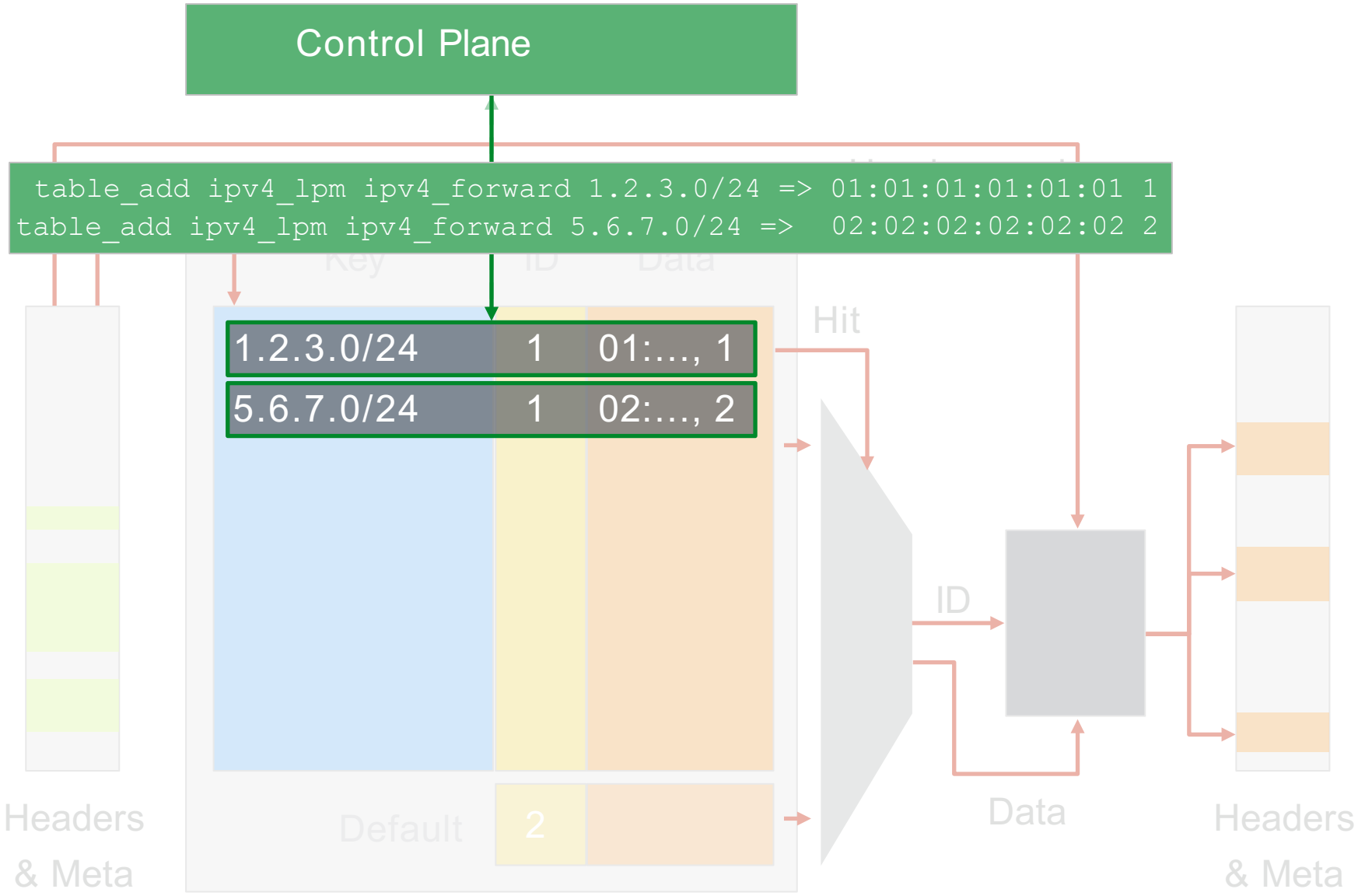
Example: IP forwarding table

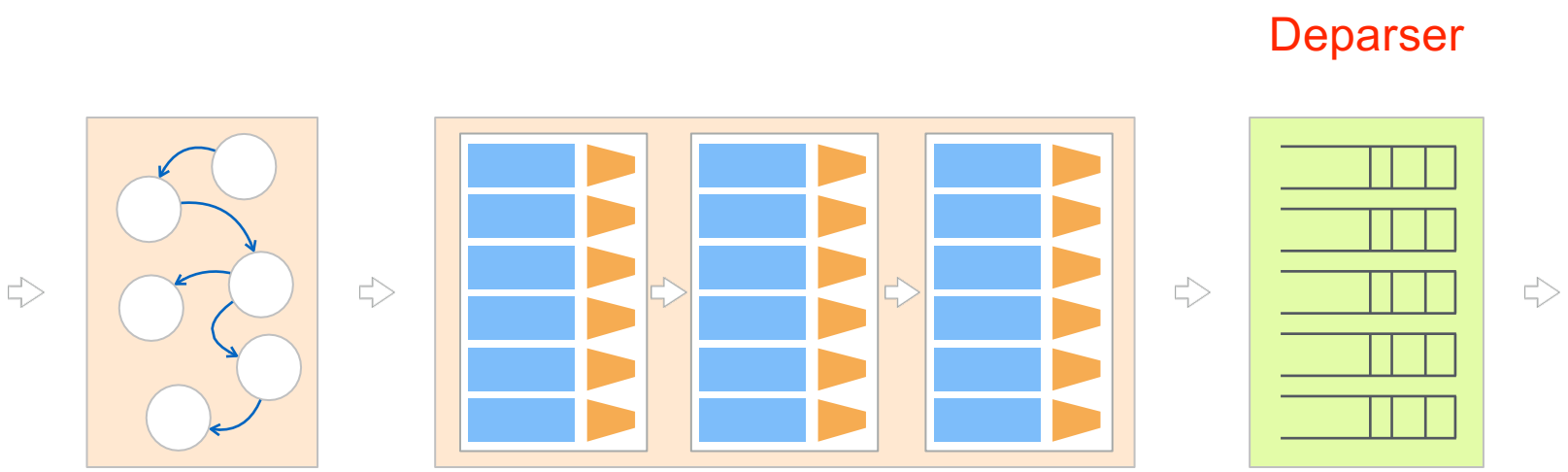



```
action ipv4_forward(macAddr_t dstAddr, egressSpec_t port) {
    hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
    hdr.ethernet.dstAddr = dstAddr;
    hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
    standard_metadata.egress_spec = port;
}
```

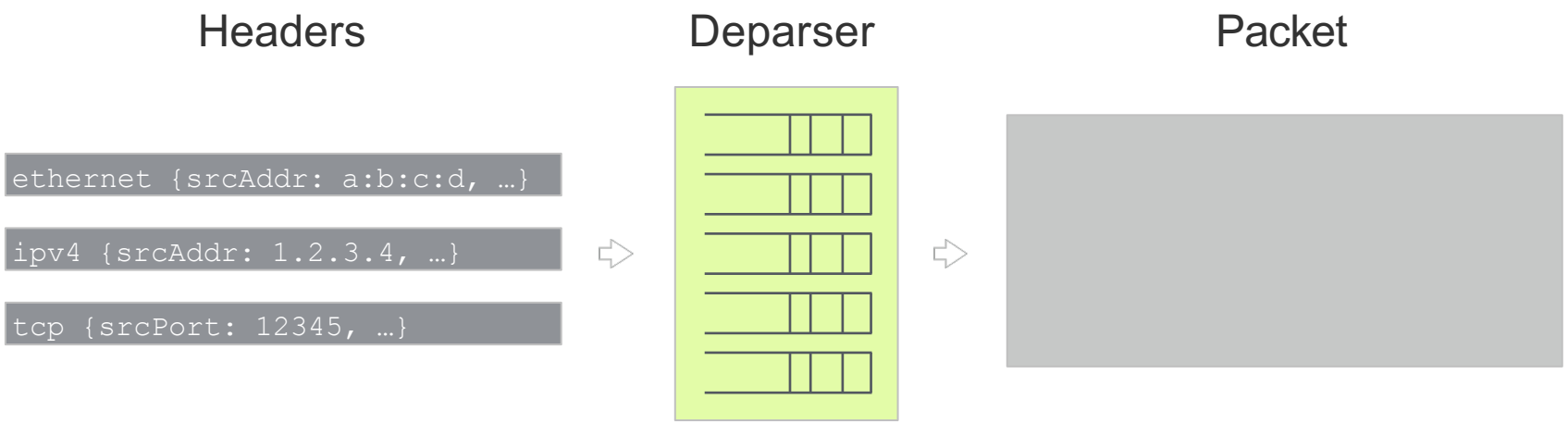
Example: IP forwarding table

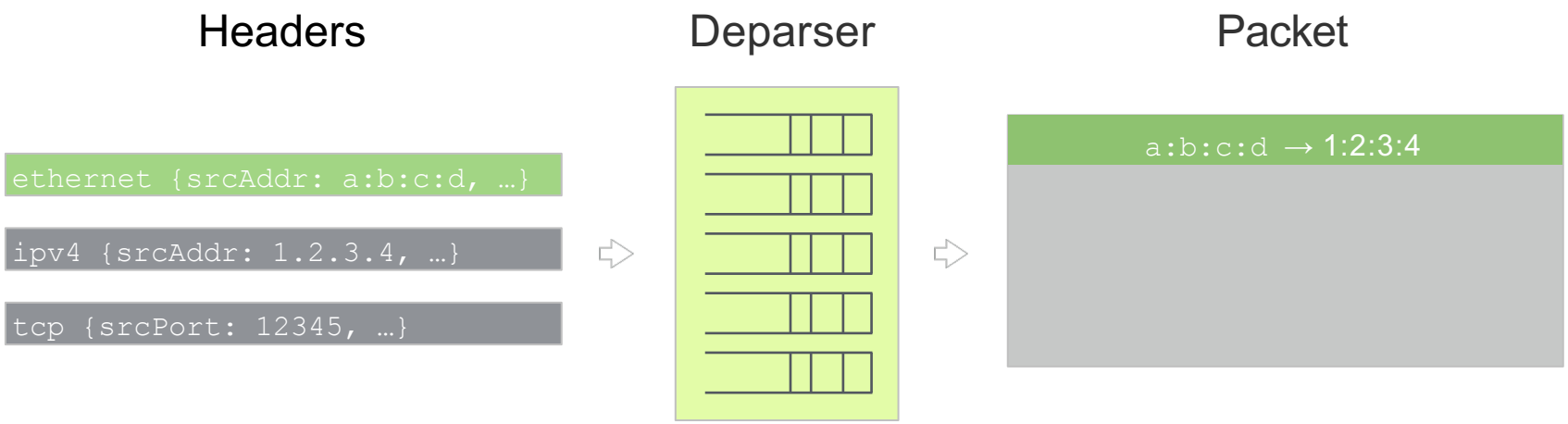



Control Plane
table_add ipv4_lpm ipv4_forward 1.2.3.0/24 => 01:01:01:01:01:01 1
table_add ipv4_lpm ipv4_forward 5.6.7.0/24 => 02:02:02:02:02:02 2
1.2.3.0/24 1 01:..., 1
5.6.7.0/24 1 02:..., 2
Default 2
Hit
ID
Data
Headers & Meta
Headers & Meta

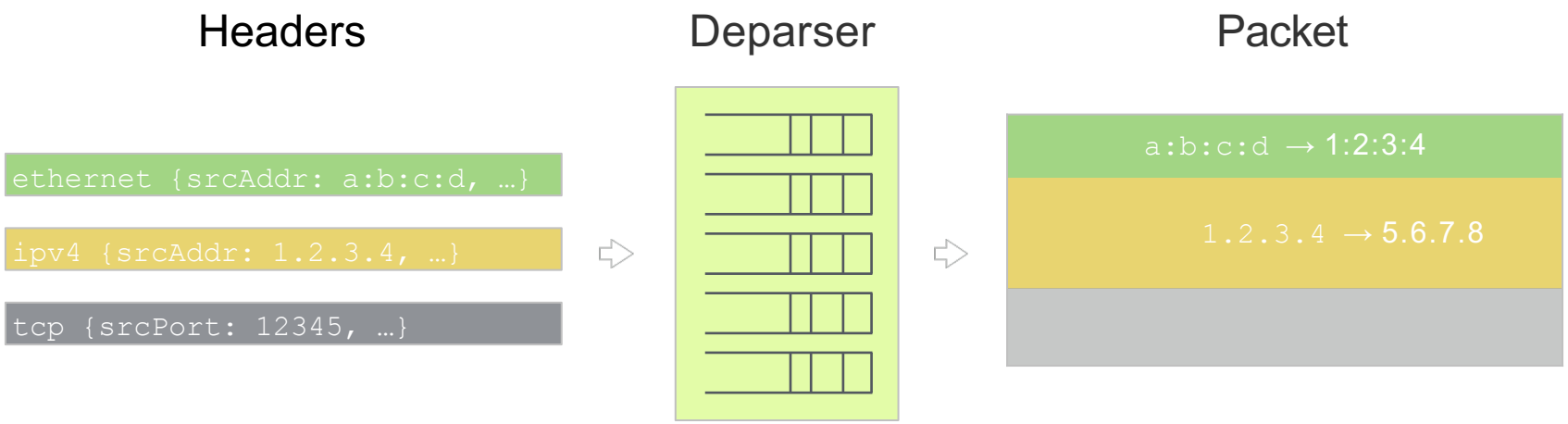

Deparser

The Deparser assembles the headers back into a well-formed packet

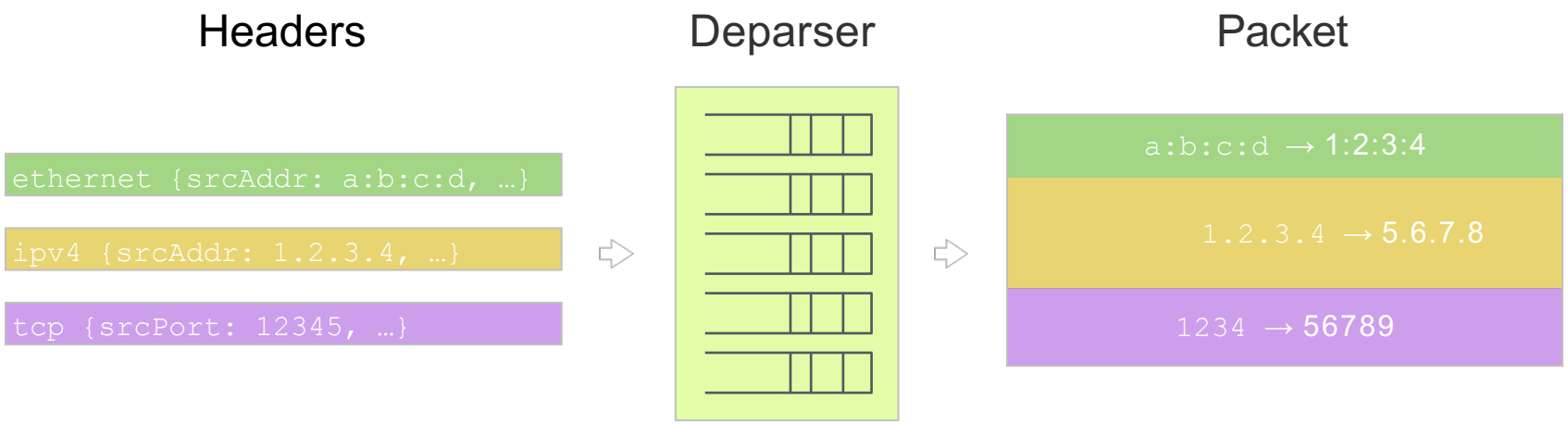




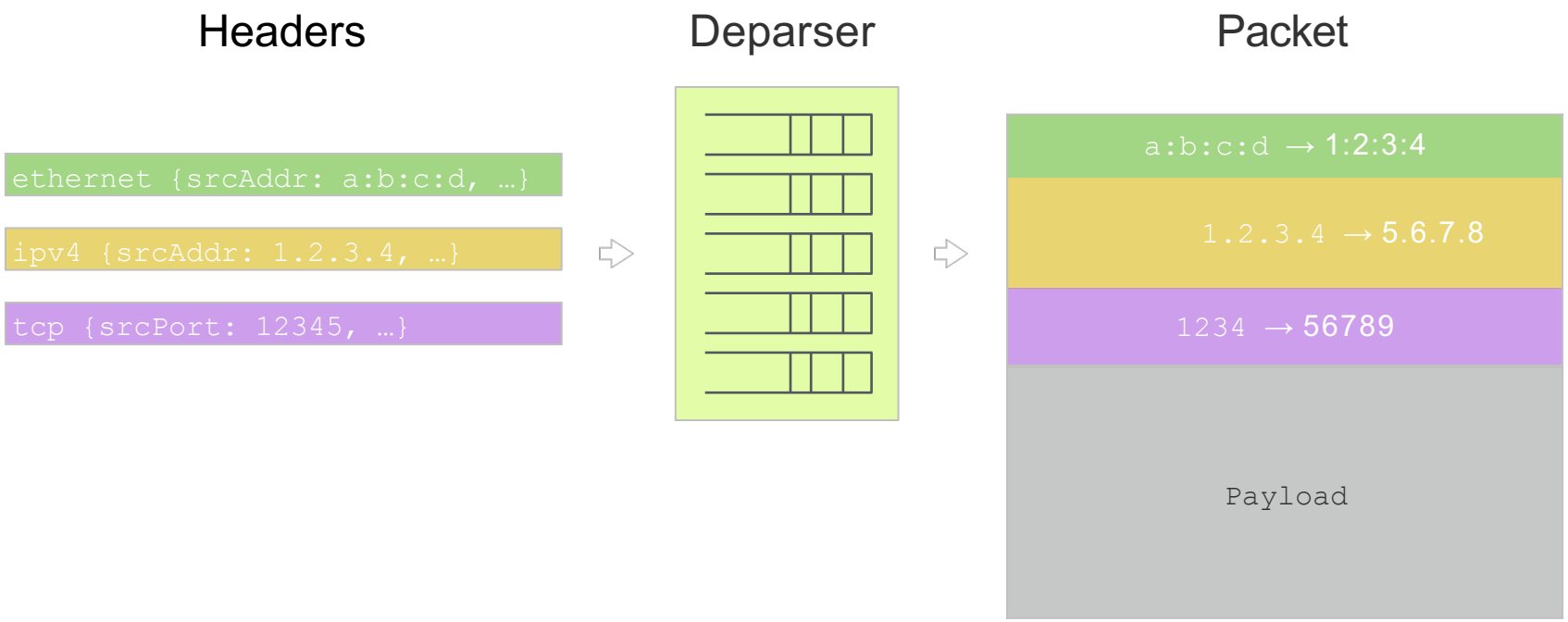

```
control MyDeparser(packet_out packet, in headers hdr) {
    apply {
        packet.emit(hdr.ethernet);


    }
}
```



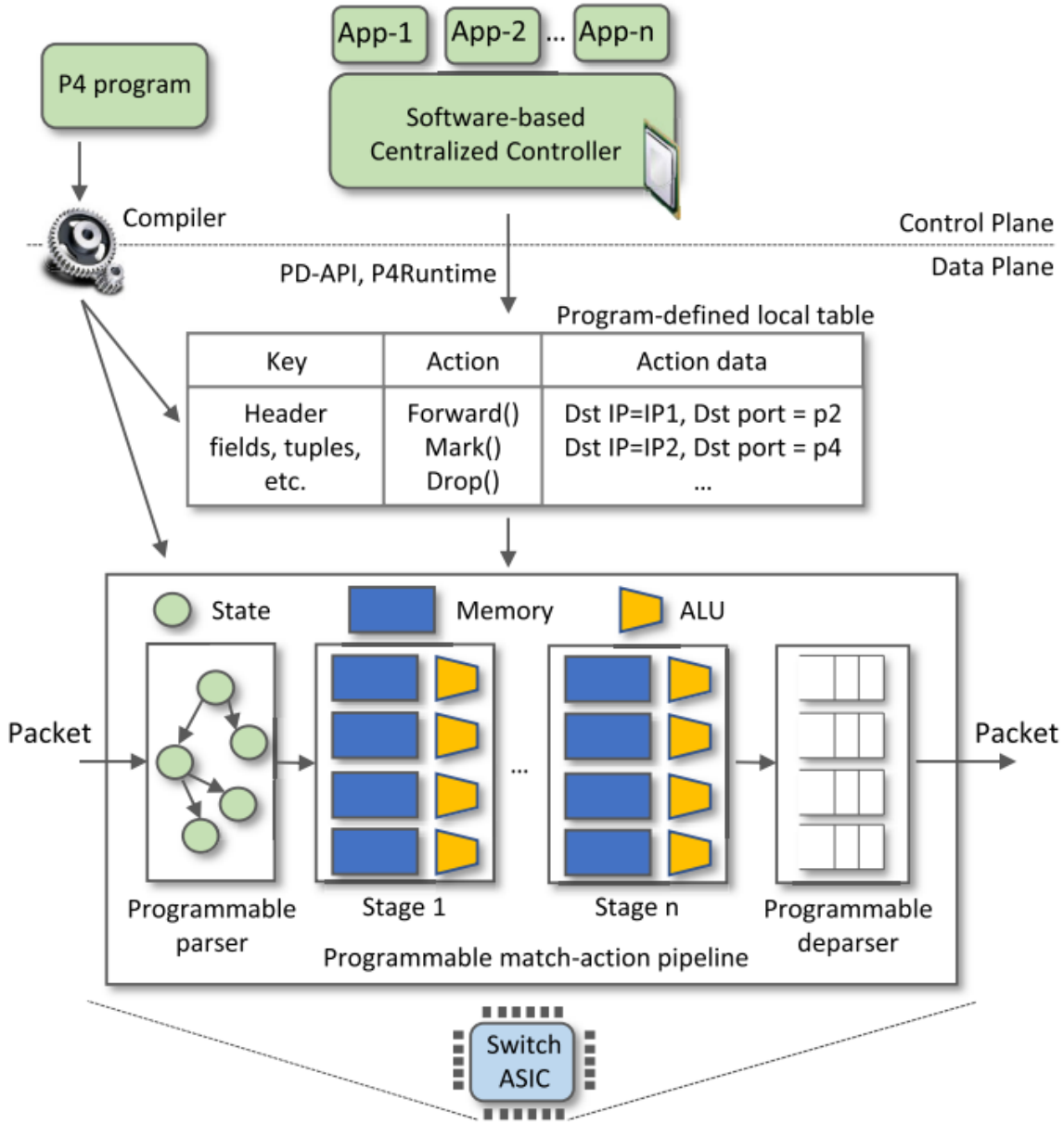

```
control MyDeparser(packet_out packet, in headers hdr) {
    apply {
        packet.emit(hdr.ethernet);
        packet.emit(hdr.ipv4);

    }
}
```



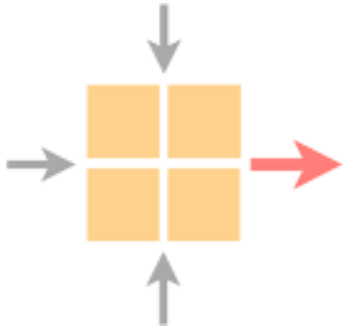

```
control MyDeparser(packet_out packet, in headers hdr) {
    apply {
        packet.emit(hdr.ethernet);
        packet.emit(hdr.ipv4);
        packet.emit(hdr.tcp);
    }
}
```



We have implemented a simple L3 routing in a P4 pipeline


App-1
App-2
...
App-n
P4 program
Software-based
Centralized Controller
Compiler
Control Plane
Data Plane
PD-API, P4Runtime
Program-defined local table
Key
Action
Action data
Header
fields, tuples,
etc.
Forward()
Mark()
Drop()
Dst IP=IP1, Dst port = p2
Dst IP=IP2, Dst port = p4
...
State
Memory
ALU
Packet
Packet
Programmable
parser
Stage 1
Stage n
Programmable
deparser
Programmable match-action pipeline
Switch
ASIC

Learning by yourself – Continue the experience



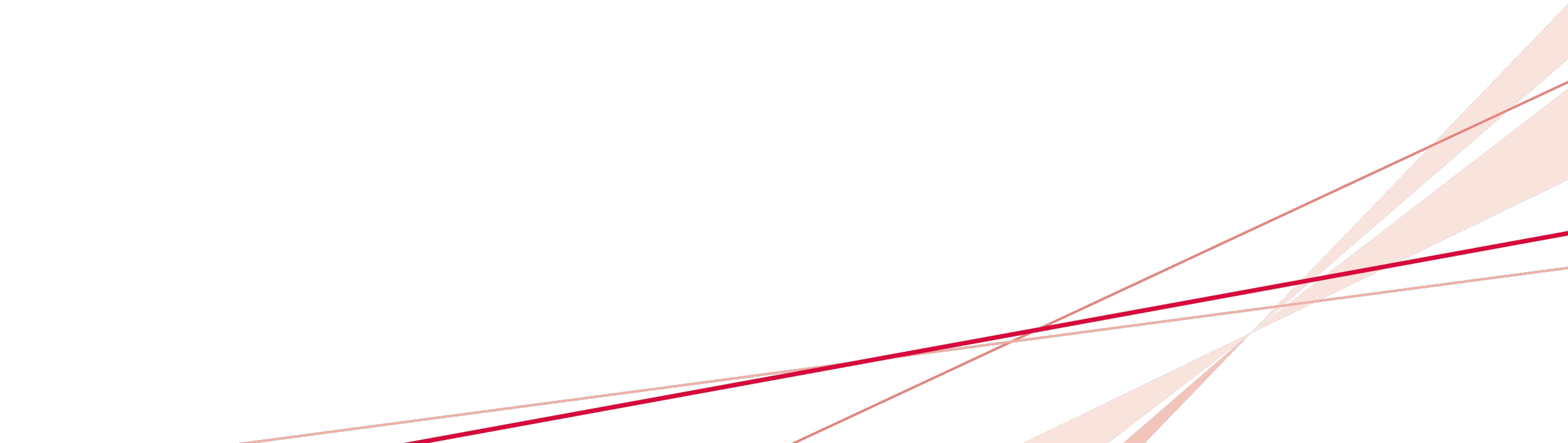
Networked Systems

ETH Zürich

https://github.com/nsg-ethz/p4-learning

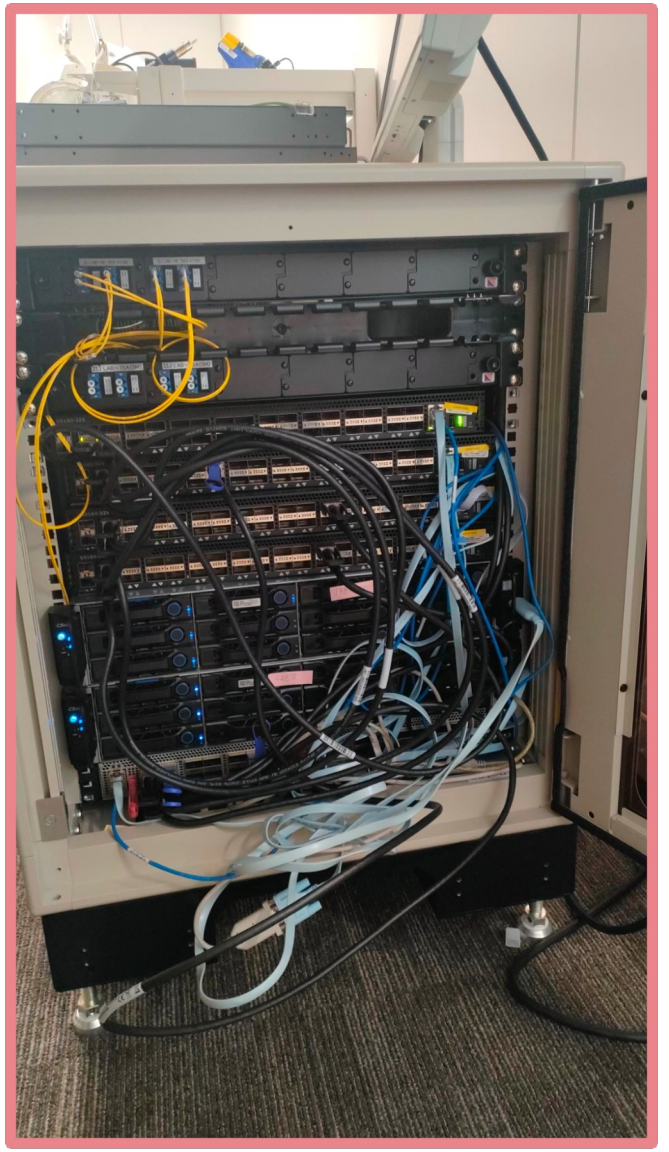
https://p4.org/learn

IIJ Lab – P4 SDN projects



IIJ Lab – P4 Projects – The Testbed in the WorkShop Room

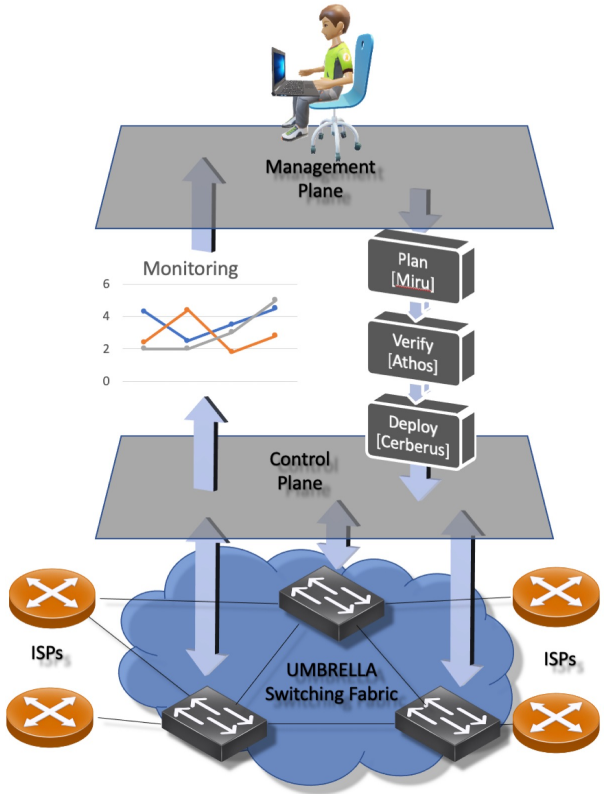
- 5 x Tofino1 switch of 32 x 100G QSFP28 ports
- 2 x Server with 10Gs and 100G Interfaces



IIJ Lab – P4 Projects – HolistIX

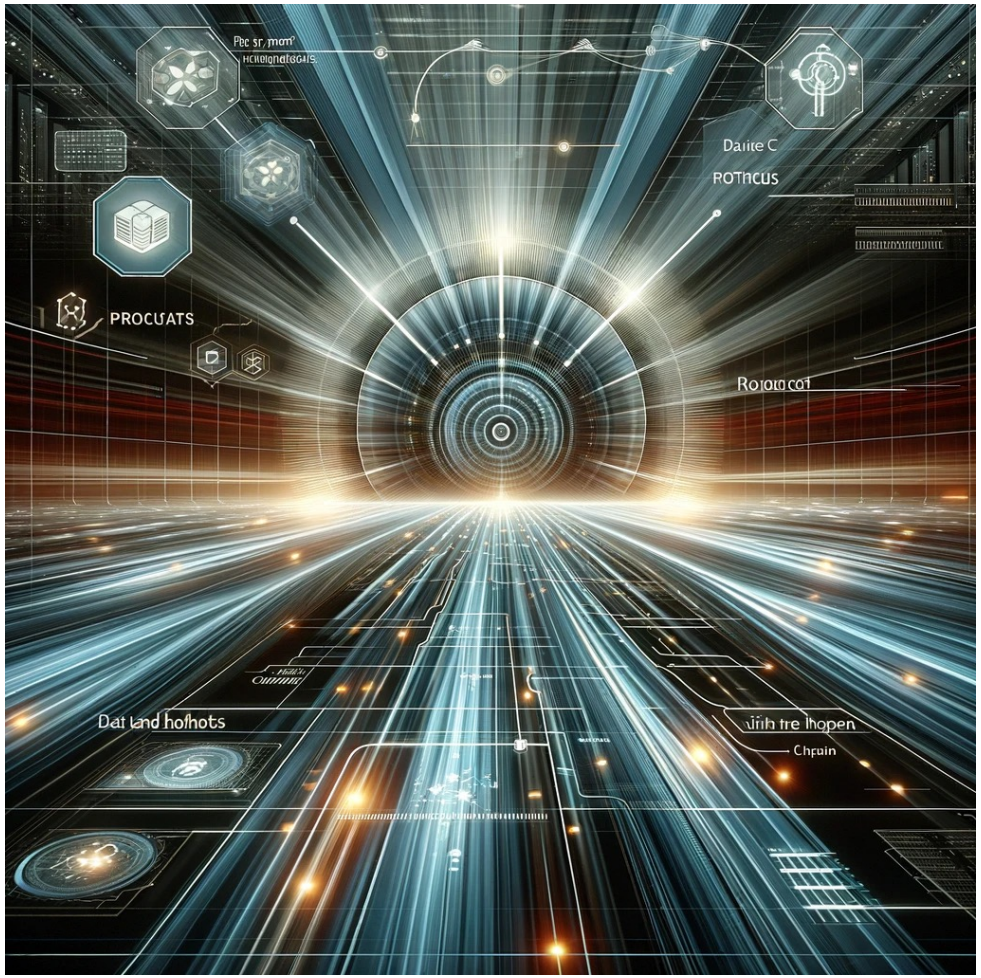
HOLISTIX

- Introduce automation from the top down for IXPs
  - Plan > Verify > Deploy
- Based on the Software Designed Network Umbrella switching fabric implemented in P4
- Change broadcast packets to unicast ones
- No more quarantine time – fast deployement
- No overhead like EVPN – MPLS based



IIJ Lab – P4 Projects – UmbrellaDC

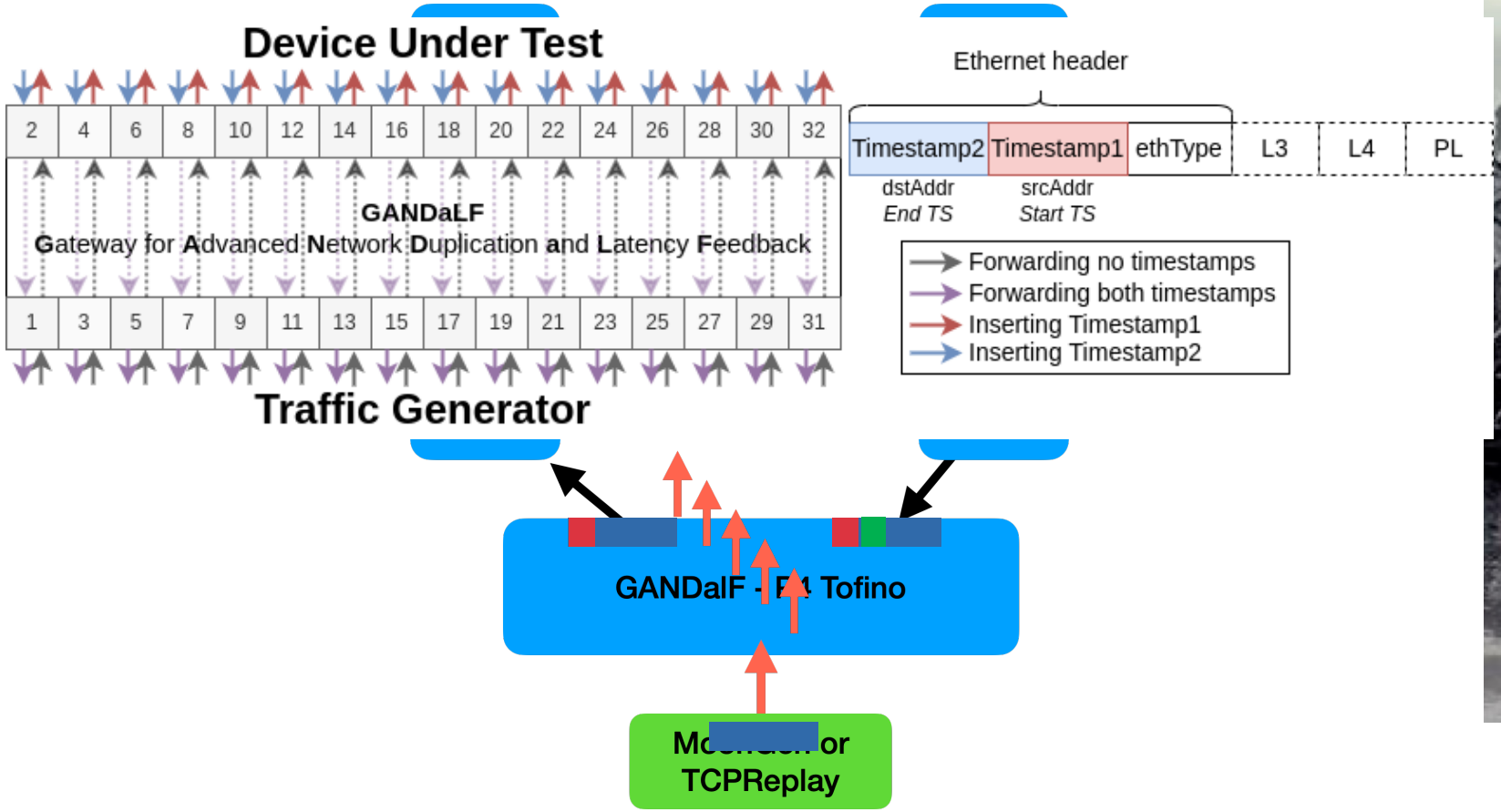
- Focus on Optimizing flow completion time for RPC based protocols [FlowLet load-balancing] supporting TCP and Homa
- Improved latency with no Overhead and minimal pipeline steps (Homa focus on l4-l7 and not on l2-l3)
- Today DC minimal properties:
  - Seamless migration (No @IPs change) > L2 Flat
  - Top-to-bottom network management (e.g HolistIX approach) - Single Source of Truth centralized management
  - Any end host should be able to communicate with any other via all available physical paths



IIJ Lab – P4 Projects – GANDaLF
Gateway for Advanced Network Duplication and Latency Feedback

- Nano-second measurement
- 100Gps from 64B to Jumbo
- No cost and hardware generated

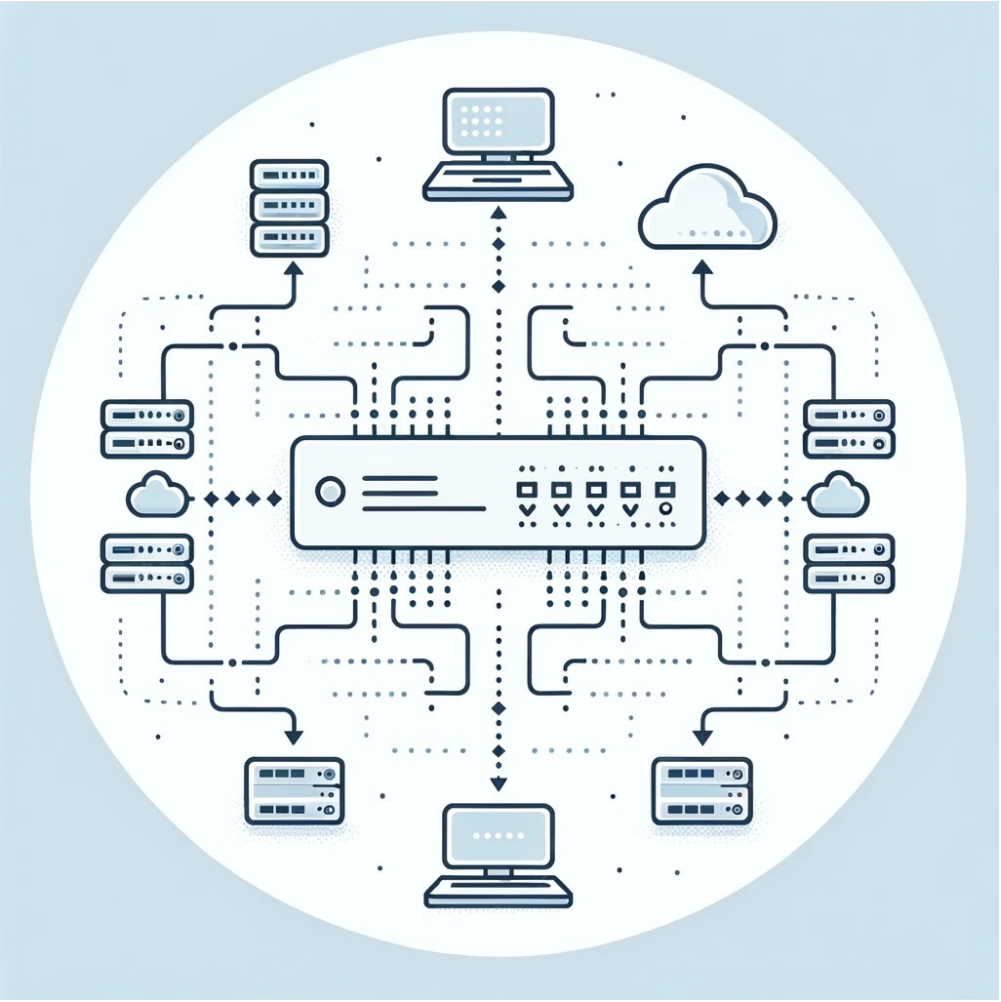
IIJ Lab – P4 Projects – GANDaLF
Gateway for Advanced Network Duplication and Latency Feedback



Outcomes, Thoughts and Conclusion

SDN - Research and Industry combined evolution

- P4-Lang is powerful language to describe match-action data-plane pipeline !
- Even if Intel Barefoot as stopped to invest in Tofino3 ASIC, various organization are using P4-Lang as their reference model language.
- Others hardware chipset may support it in the future.
- P4 was made for high-performance hardware, does SmartNIC or soft switch could keep P4 community active ?
- Self-learning is key for innovation – Have fun
- I am pleased to share more in-depth tech details !



Thanks !

Questions – Discussions

IIJ
Internet Initiative Japan