[S06] Introduction to P4

Programming Protocol Independent Packet Processors

Redes Software

Múltiples Grados

Curso 2024-2025

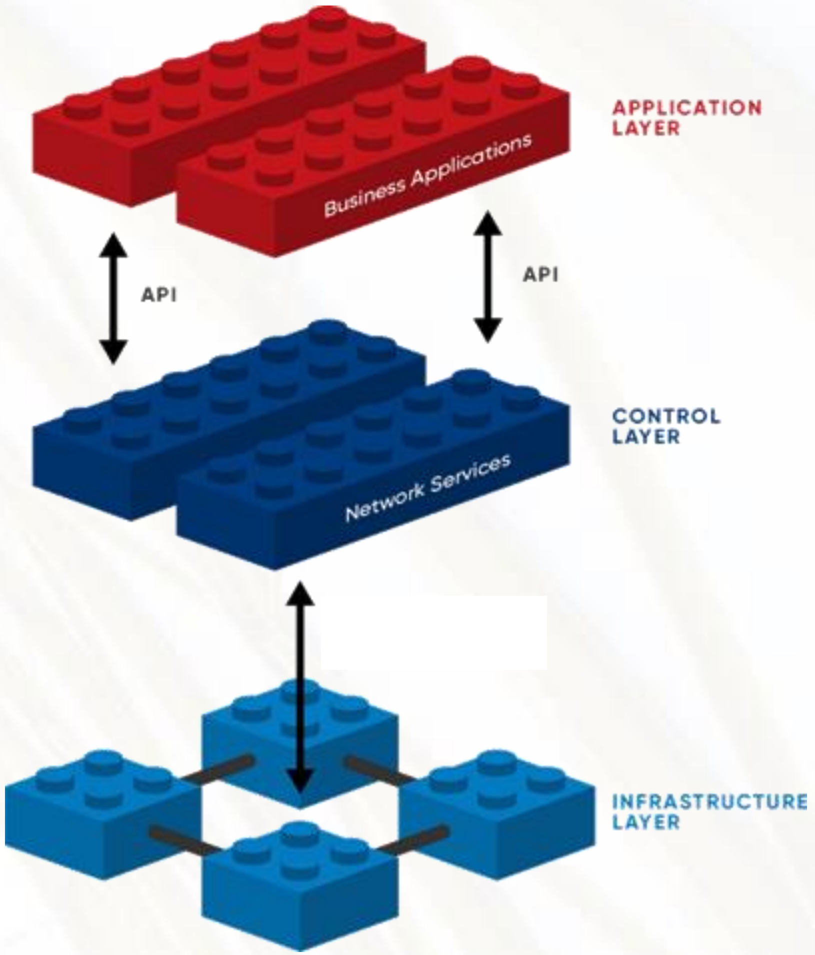
Based on slides provided by Pablo Molinero for the Master on SDN/NFV of UC3M

uc3m | Universidad Carlos III de Madrid

P4

Contents

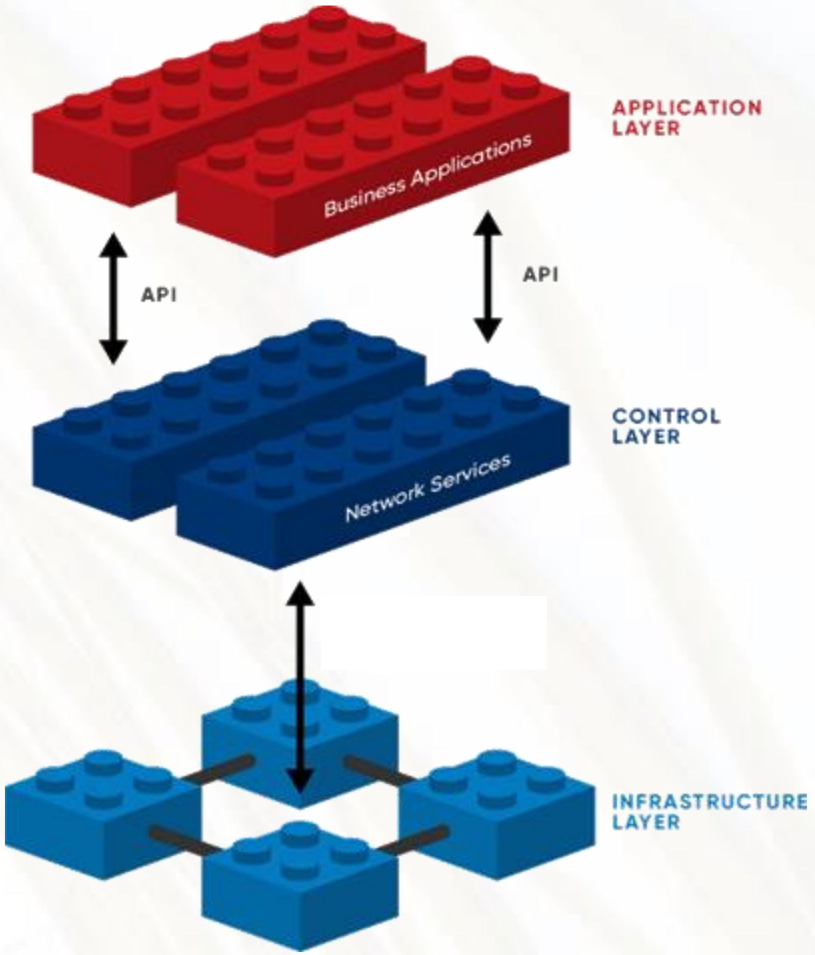
- Why P4?
- What is P4?
- Standards & History
- Use Cases
- P4 Overview
- P4 Language Elements
- Discussion



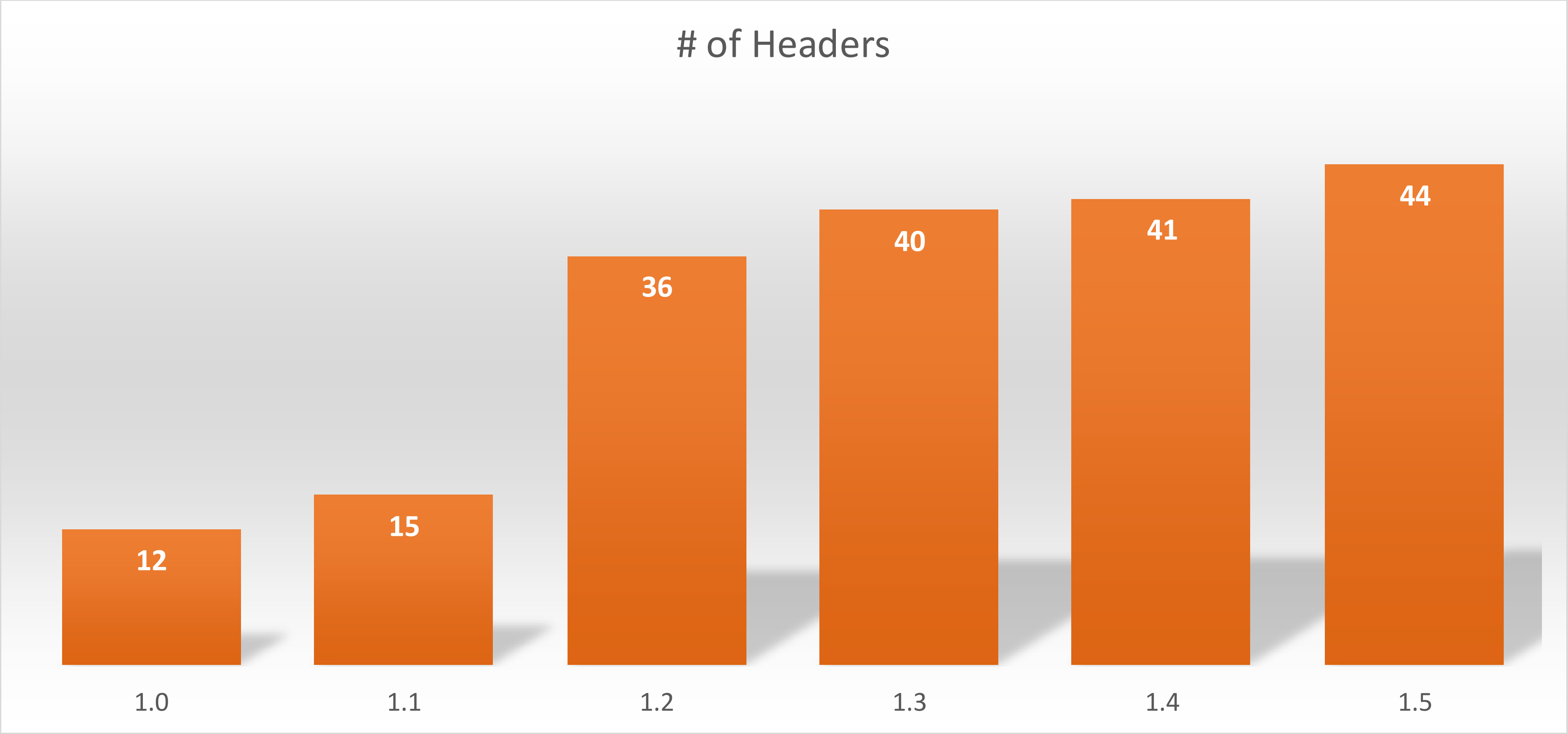
P4

Contents

- Why P4?
- What is P4?
- Standards & History
- Use Cases
- P4 Overview
- P4 Language Elements
- Discussion



Isn’t Open-Flow Enough?



Open-Flow version

Open-flow has *never* been enough: it keeps changing to describe new protocols

Why P4?

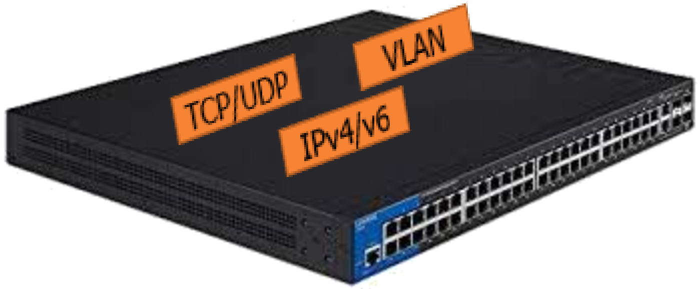
- What if I want to create a new protocol
  - Takes years to standardize it
  - High-performance implementations determined by chip manufacturing
    - E.g., VXLAN routing in Broadcom’s chips
- The P4 Language came as way to build Advanced switch architectures
  - Originally described in a 2014 SIGCOMM paper titled *“Programming Protocol-Independent Packet Processors”*
  - Programmable packet headers
  - Stateful packet processing

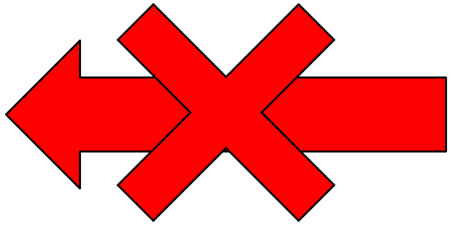
What P4 Brings

- Consequences of P4
  - HW performance with SW programmability
  - Device manufacturer ≠ device programmer
  - Many network capabilities exposed to software
- We can innovate in these areas without waiting for the device manufacturing
  - In-band Network Telemetry (INT)
  - Sub-flow load balancer
  - Service chaining
- Good for programmable switches, as well as fixed-function ones

Change of Paradigm

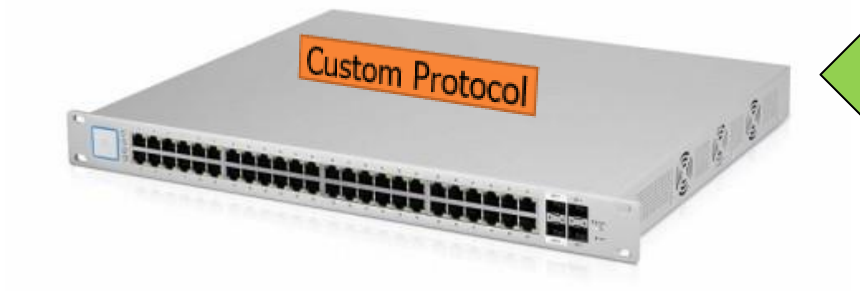
Current: Fixed-Function Switching Chip

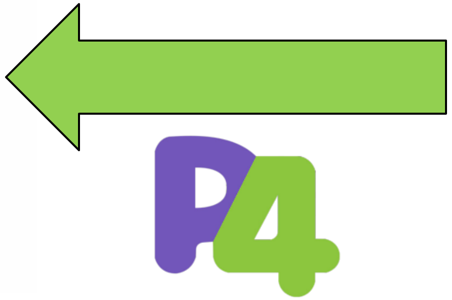




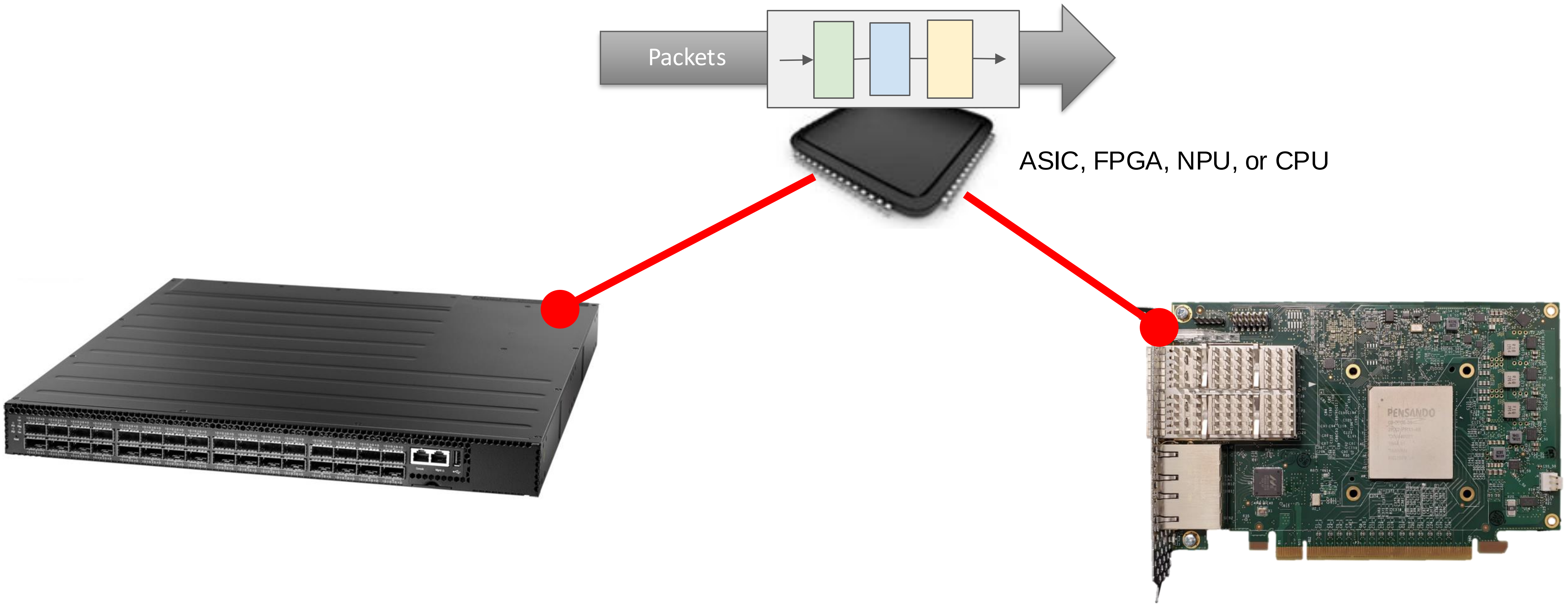
Custom protocol

Future: Programable Switching Chip





P4 in a Switch and SmartNIC



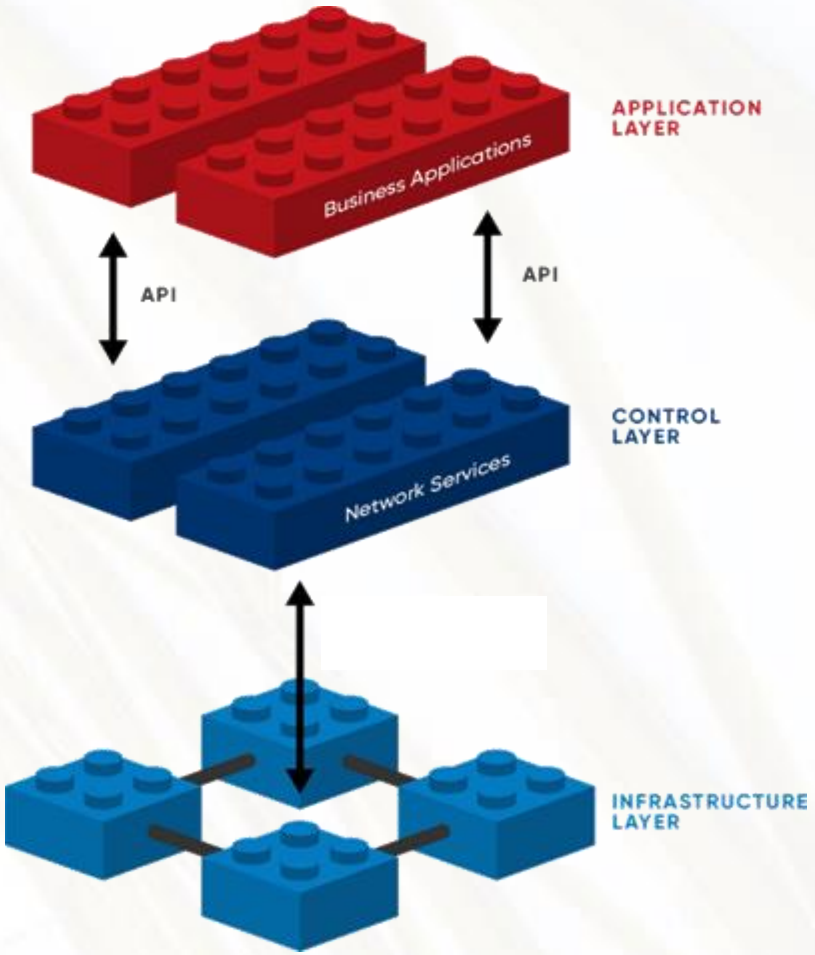
Pipeline of match-action tables

ASIC, FPGA, NPU, or CPU

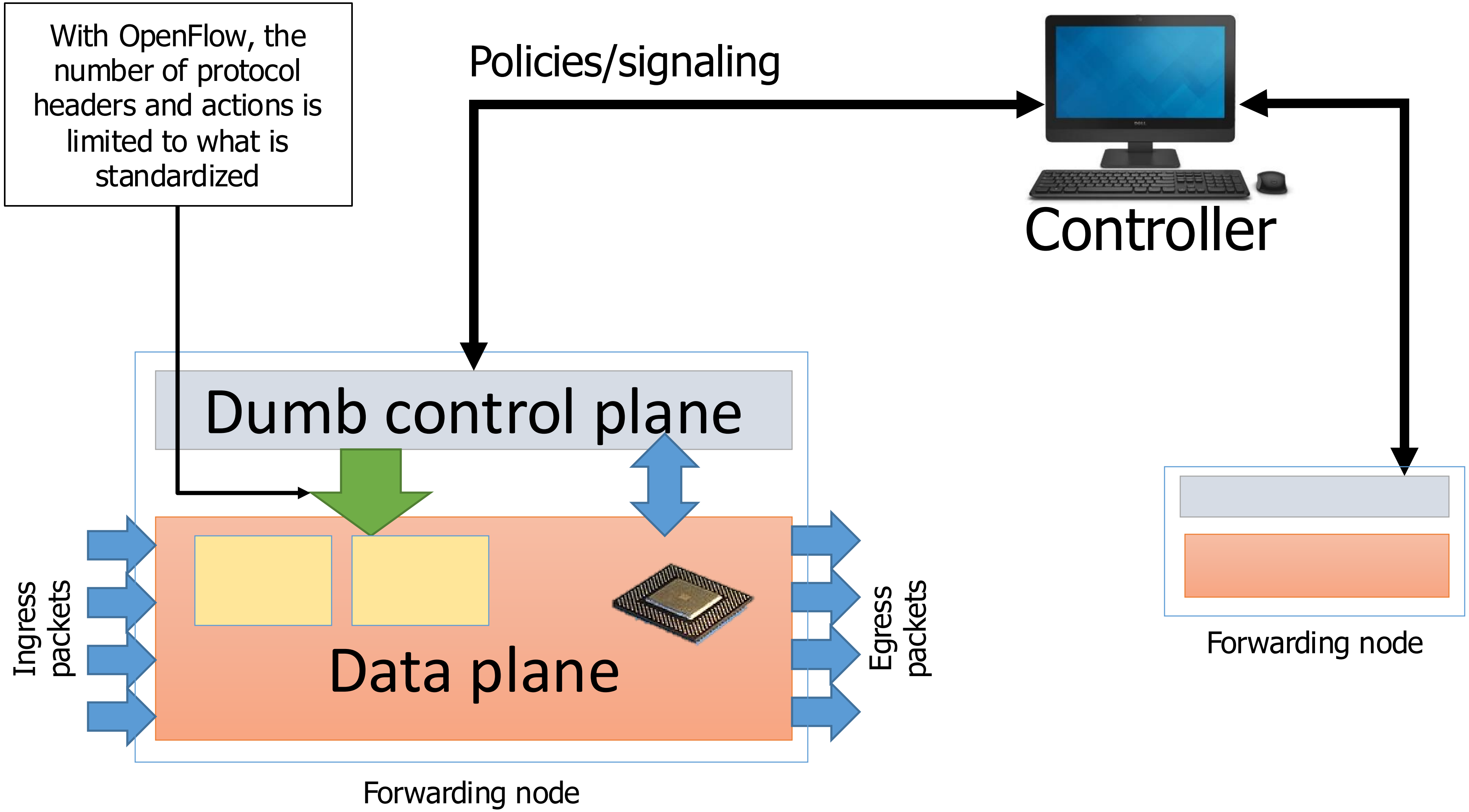
P4

Contents

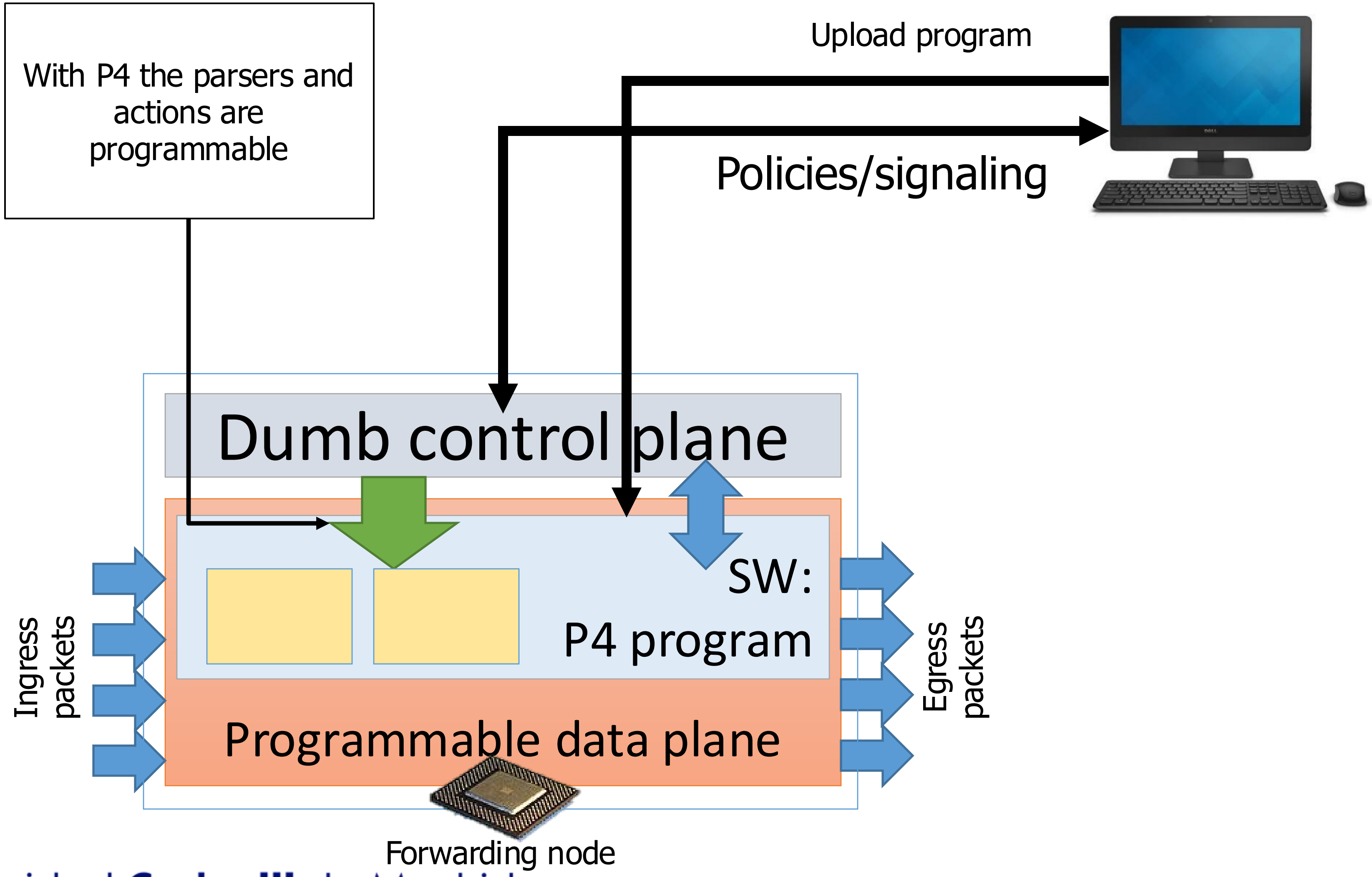
- Why P4?
- What is P4?
- Standards & History
- Use Cases
- P4 Overview
- P4 Language Elements
- Discussion



Software-Defined Networking



The P4 World



With P4 the parsers and actions are programmable

Upload program

Policies/signaling

Dumb control plane

SW:

P4 program

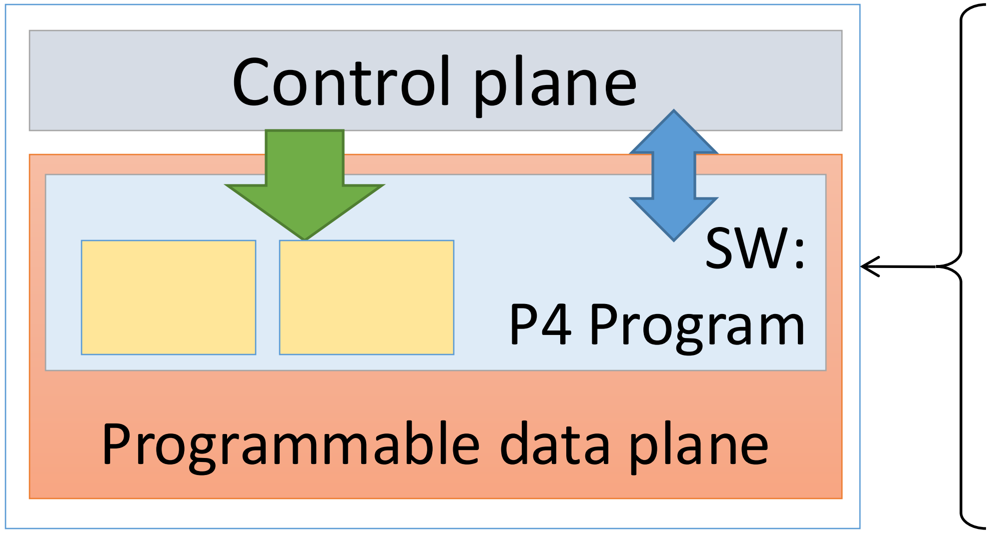
Programmable data plane

Ingress packets

Egress packets

Forwarding node

Not Just for Switches!



- Programmable switches
- FPGA switches
- Programmable network cards
- Software switches
- Hypervisor switches
- …

Result:

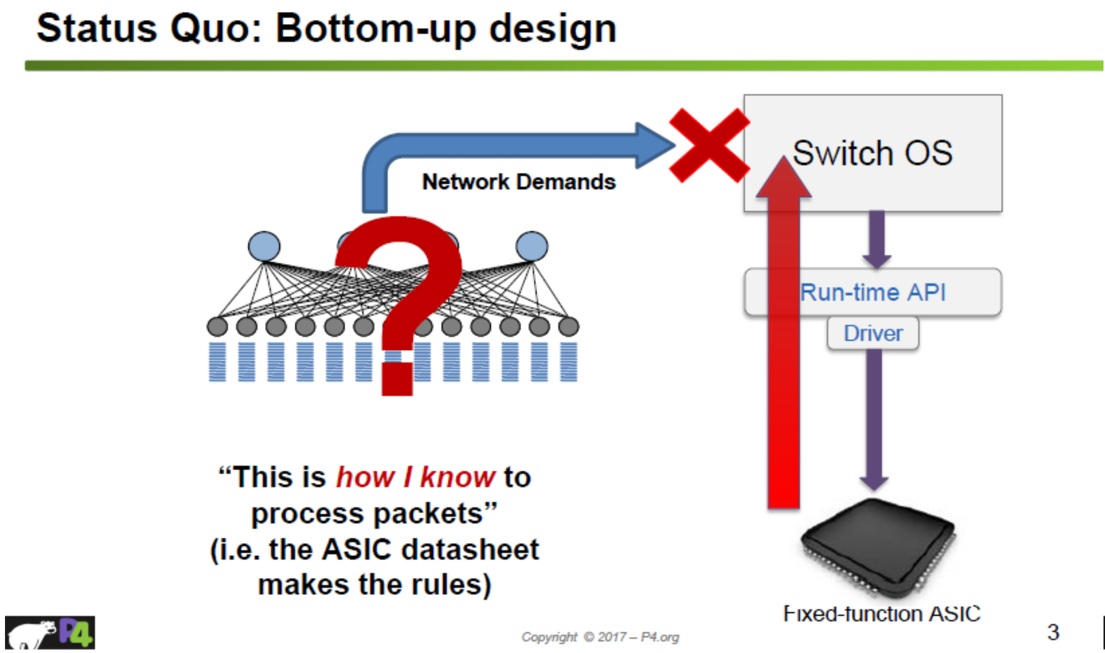
- The flexibility of a SW program (with limitations)
- The performance of HW-based forwarding (depending on the target architecture)

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, Altera, …
- These devices let us tell them how to process packets

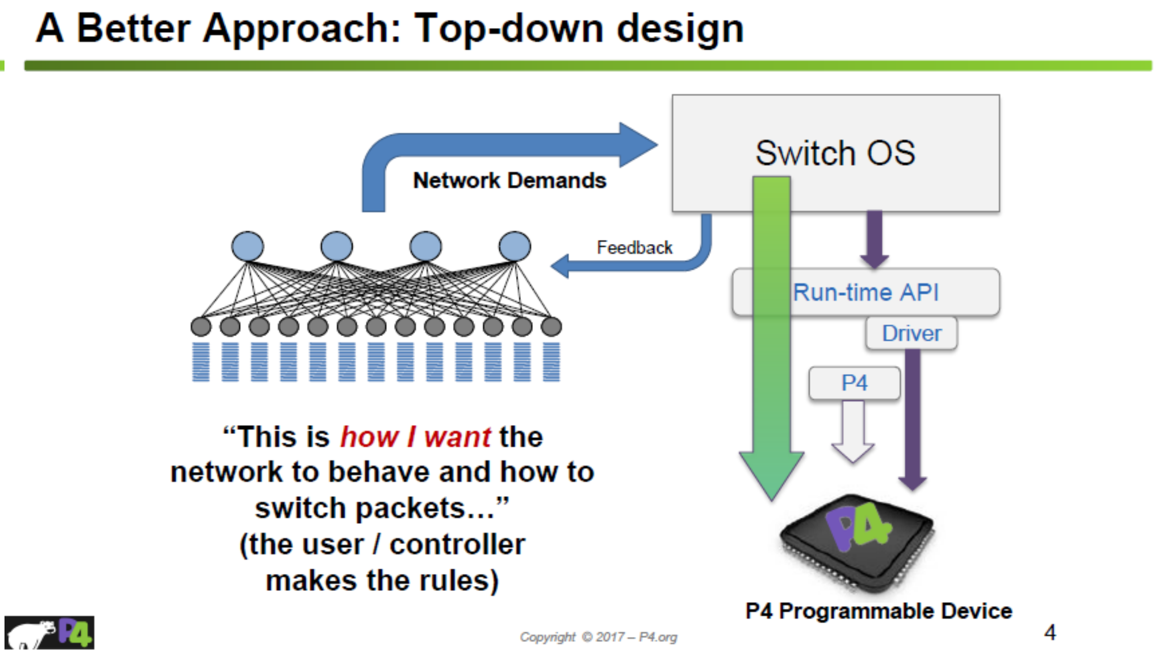
Intent-Based Programming

- Traditional network nodes expose their capabilities => not very programmable
- OpenFlow allows programmability, but still exposes a lot of internal details
- P4 programs using the desired behavior



Status Quo: Bottom-up design

"This is ***how I know*** to process packets" (i.e. the ASIC datasheet makes the rules)



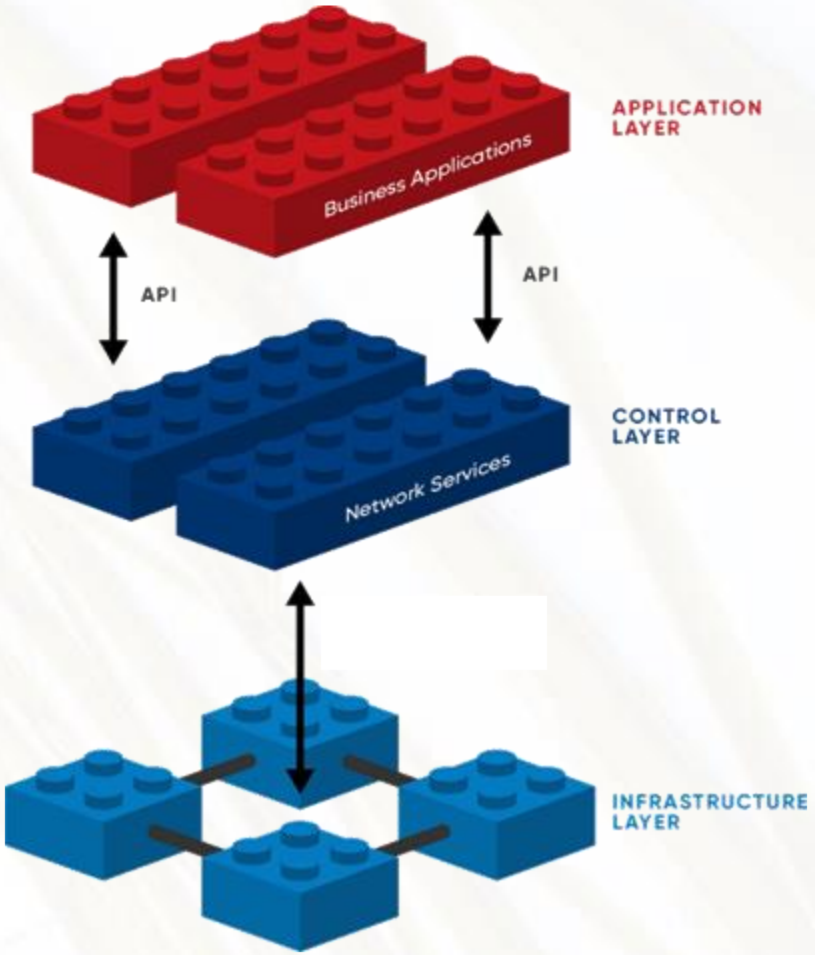
A Better Approach: Top-down design

"This is ***how I want*** the network to behave and how to switch packets…" (the user / controller makes the rules)

P4

Contents

- Why P4?
- What is P4?
- Standards & History
- Use Cases
- P4 Overview
- P4 Language Elements
- Discussion



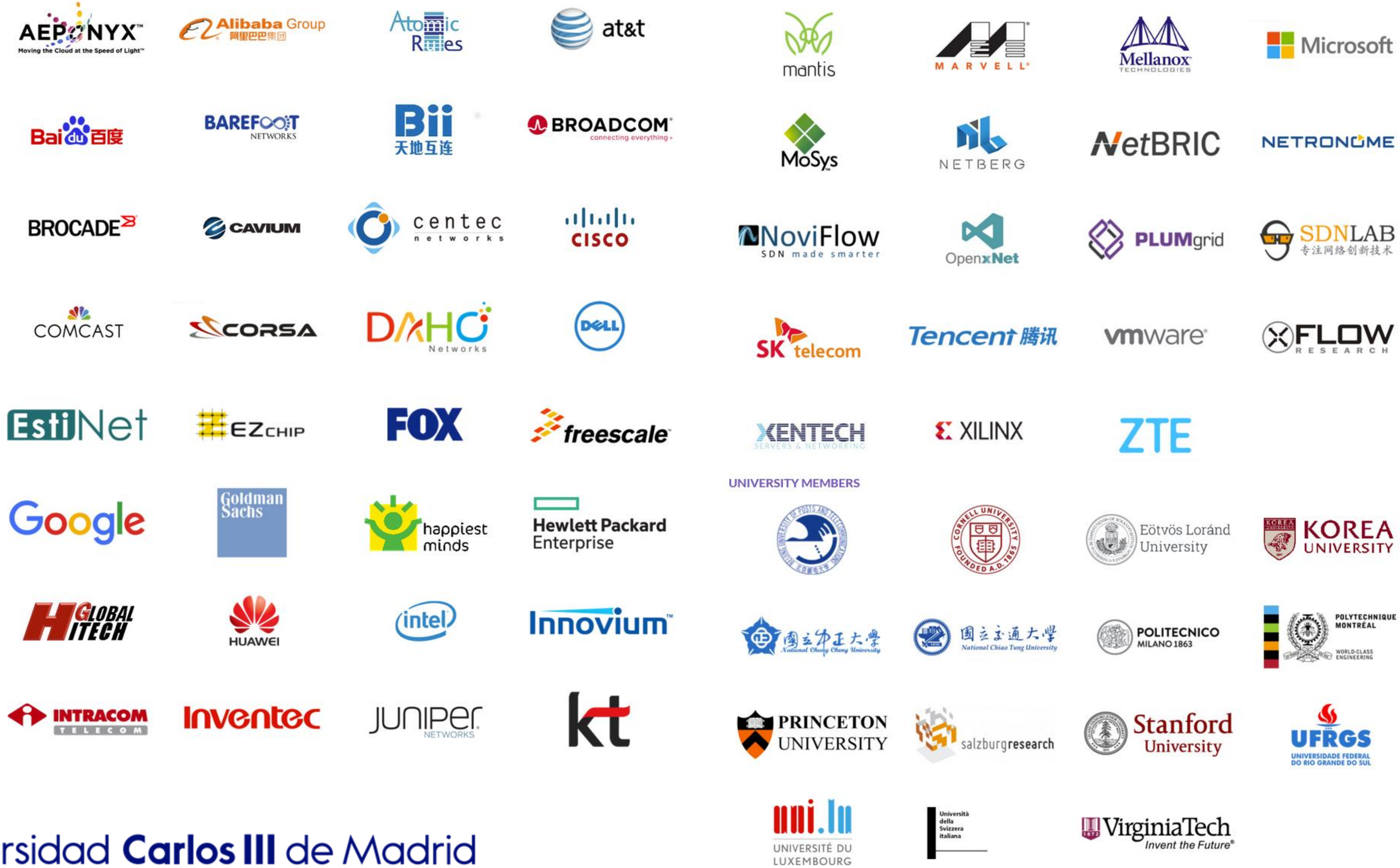
P4 Community

- http://github.com/p4lang
- http://p4.org

- P4 Language Consortium
  - Mailing lists
  - Workshops
  - P4 developer days
  - Now a Project under ONF and the Linux Foundation
- Academic papers (SIGCOMM, SOSR)

Carriers, cloud operators, chip vendors, networking, universities, start-ups

P4.org Consortium



P4 History

- 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)
- May 2017: $P4_{14}$ v1.0.4
- Apr 2016: $P4_{16}$ – first commits
- May 2017: $P4_{16}$ Specification released
- Oct 2019: $P4_{16}$ Latest Specification released

. . .

- Official Spelling P4_16 or $P4_{16}$

$P4_{16}$

- Most recent revision of P4
- Similar to C; strongly typed
- Currently in draft form
- Spec: https://p4.org/p4-spec/docs/P4-16-v1.2.0.pdf
- Reference compiler implementation (Apache 2 license): http://github.com/p4lang/p4c

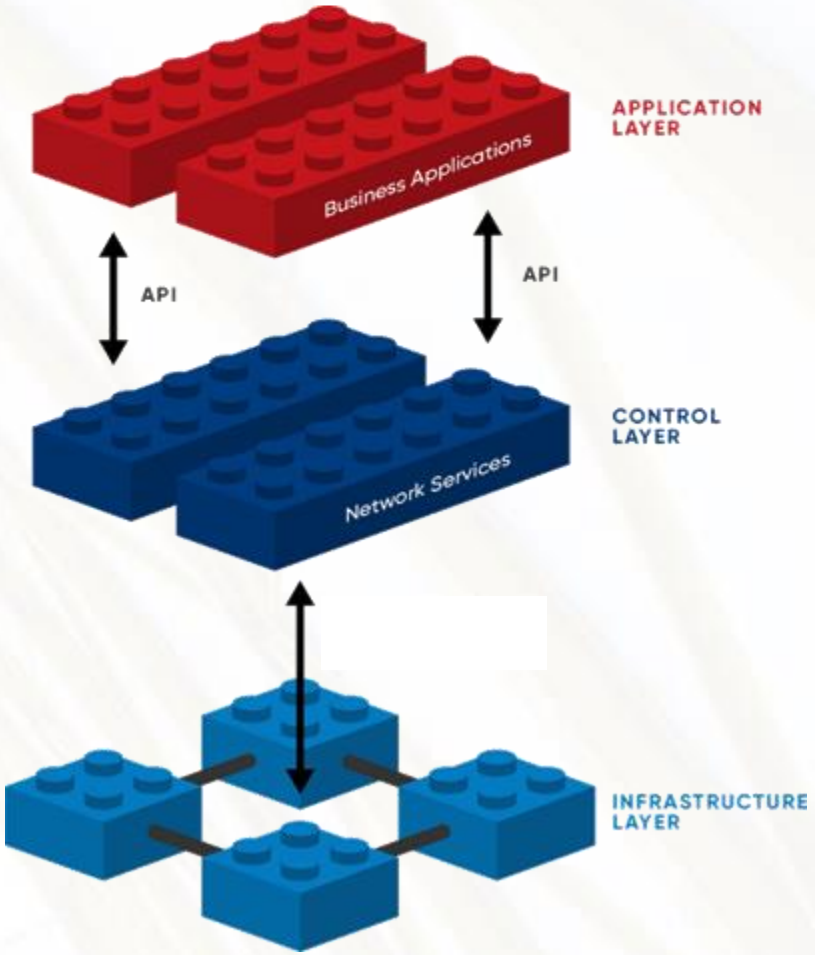
Available Software Tools

- Compilers for various back-ends
  - Netronome chip, Barefoot chip, BMv2, eBPF, Xilinx FPGA (open-source and proprietary)
- Multiple control-plane implementations
  - SAI, OpenFlow
- Simulators
- Testing tools
- Sample P4 programs
- Tutorials

P4

Contents

- Why P4?
- What is P4?
- Standards & History
- Use Cases
- P4 Overview
- P4 Language Elements
- Example
- Discussion



Use Only What You Need

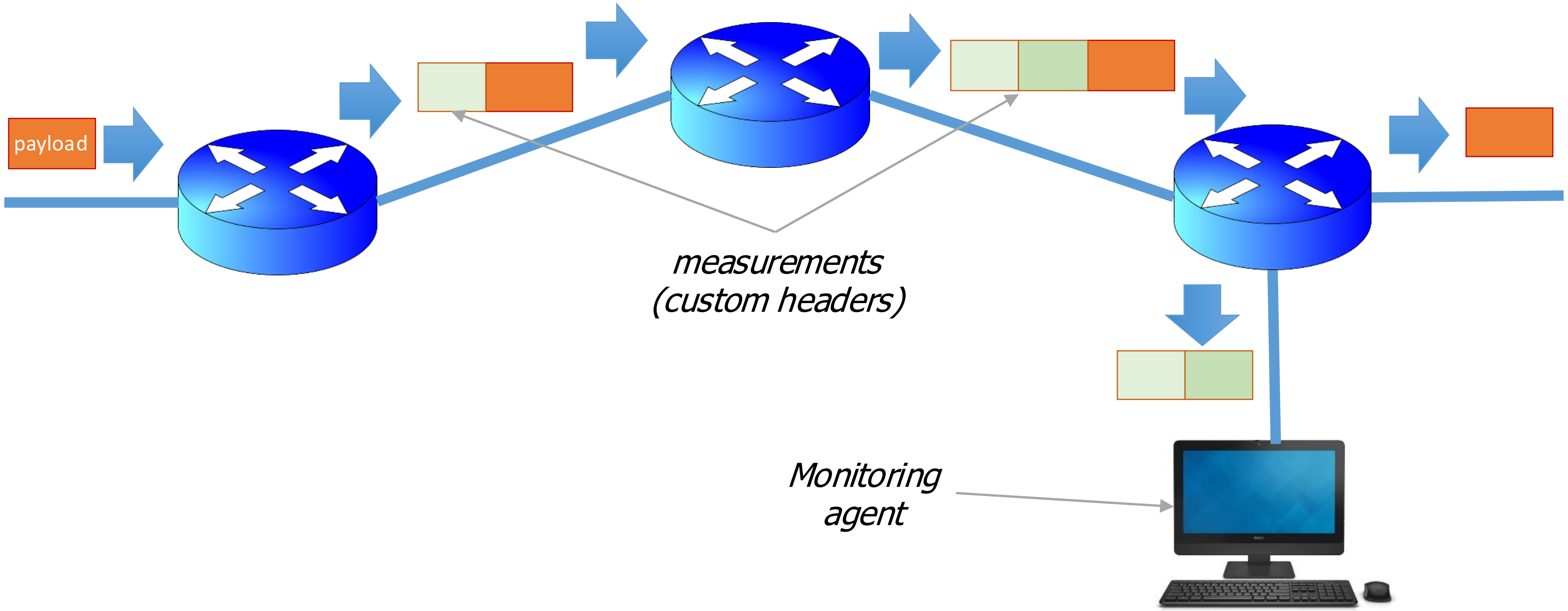
- IETF has issued thousands of RFCs
- Switch RAM and CPU is very expensive
- Network operators can *remove* protocols
- Simpler troubleshooting

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.

Network Monitoring

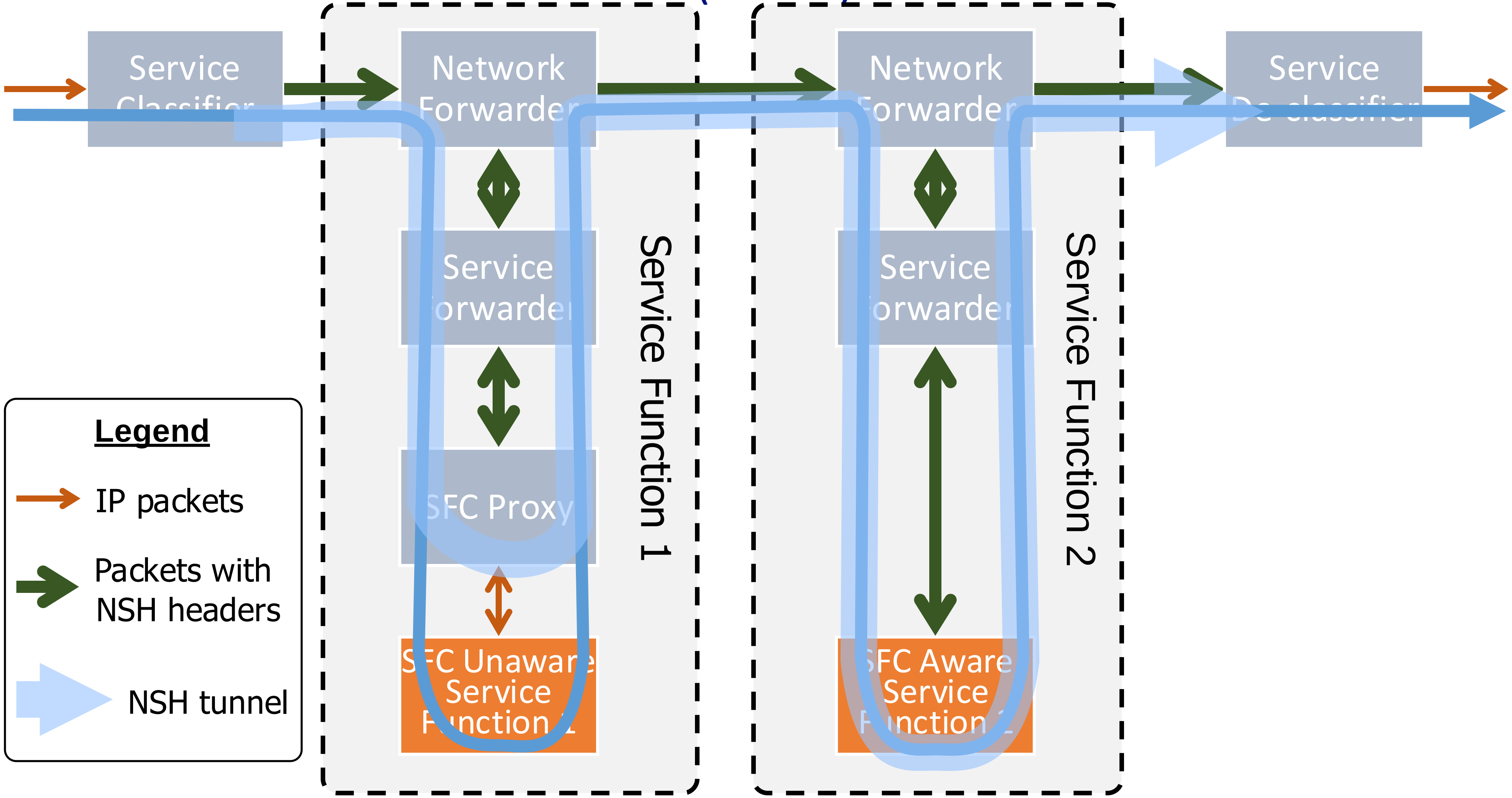


In-Band Network Telemetry (INT)

Improving Network Monitoring and Management with Programmable Data Planes

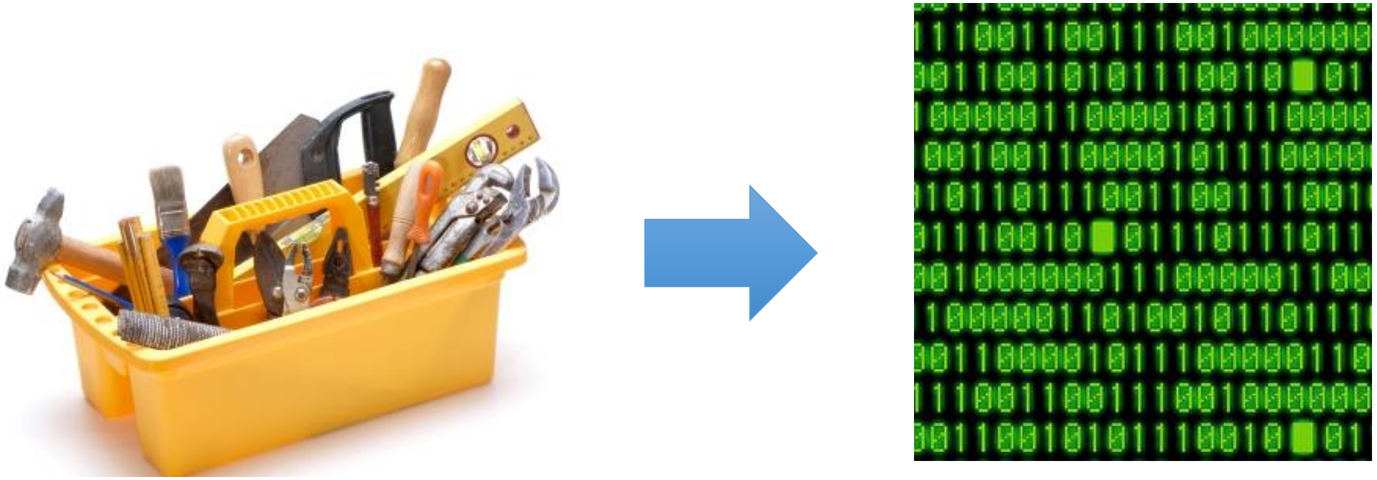
By Mukesh Hira & LJ Wobker

IETF Service Function Chaining: Network Services Header (NSH)



Network = Software

- Use ***software*** engineering principles and tools
- Upgrade your network at any time
- Protocols = intellectual property

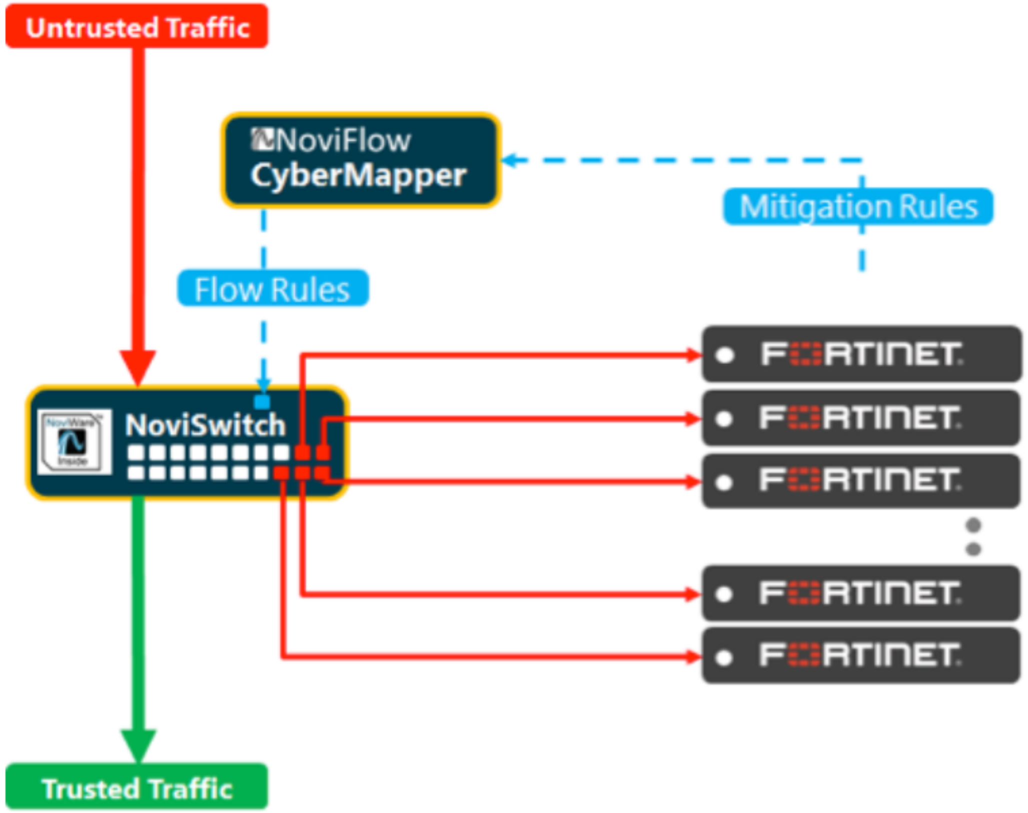


Protocols = Programs

- Implement (new) protocols
  - VxLAN: 175 lines of code
  - NVGRE: 183 lines of code
- Low overhead (high speed)
- Define your own packet processing policies
- Improved signaling, monitoring, and troubleshooting
- Change functionality with software upgrades
- Use only what you need

Scaling Network Security Services

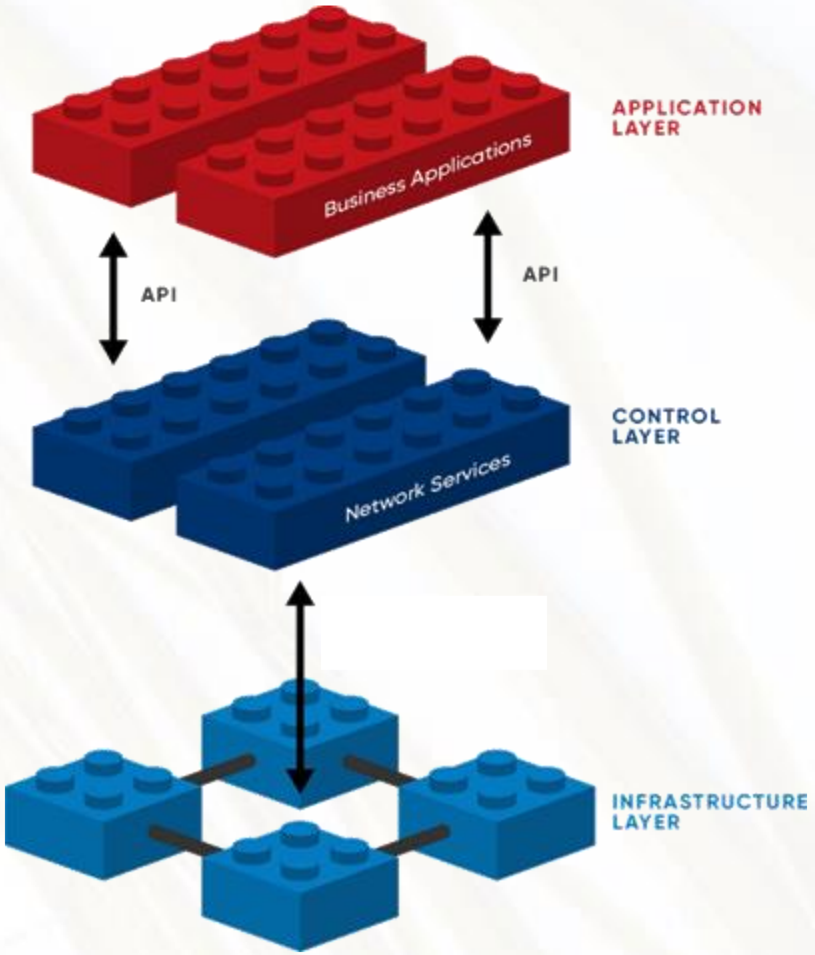
- Services
  - Threat Intelligence Gateway
  - Packet Broker
    - Filtering, mirroring, Port-pairing
  - Load Balancing
    - Sticky stateless load balancing
    - Proportional load balancing
    - Pool Scaling
  - Networking
    - In-band Network Telemetry (INT)
    - Service Chaining
    - Segment Routing



P4

Contents

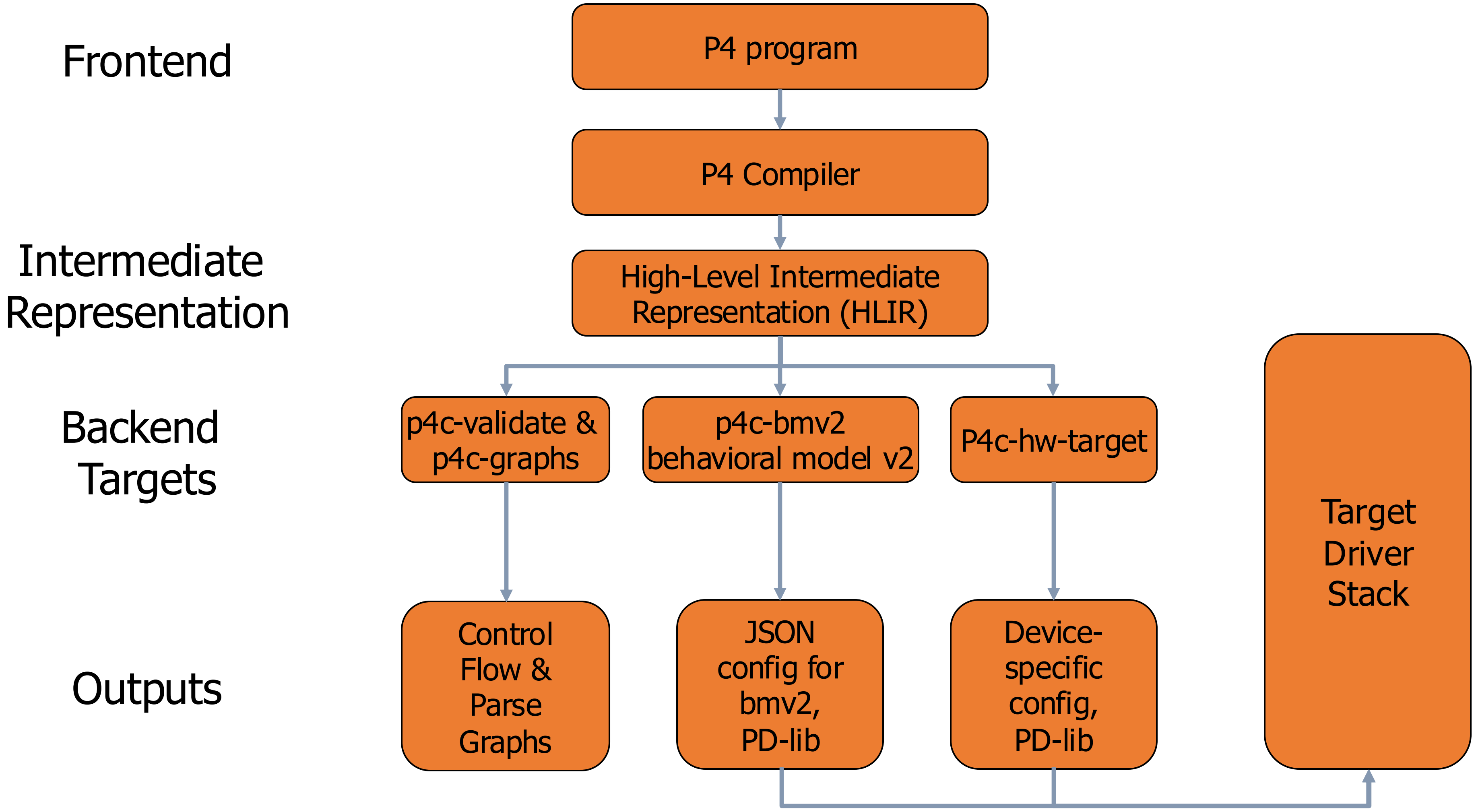
- Why P4?
- What is P4?
- Standards & History
- Use Cases
- P4 Overview
- P4 Language Elements
- Discussion



P4 - Protocol Independent Packet Processing

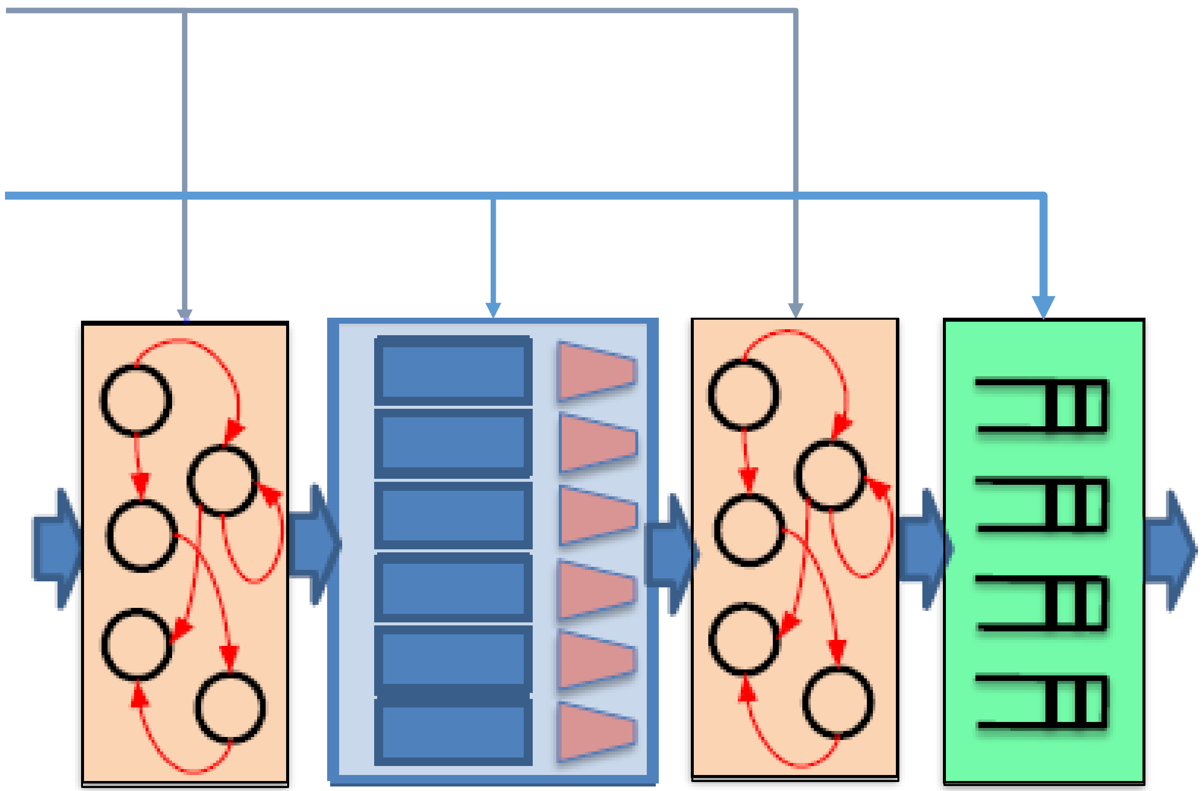
- Protocol independence
  - Configure packet parser. Not restricted to few protos
  - Define a set of typed match + action tables
- Target independence
  - Program without knowledge of switch details
  - Rely on compiler to configure the target switch
- Reconfigurability
  - Change parsing and processing in the field
- Programmable Network Devices
  - PISA: Protocol Independent Switch Architecture
  - ASIC, NPU, CPU, FPGA

P4 Modular Compilation

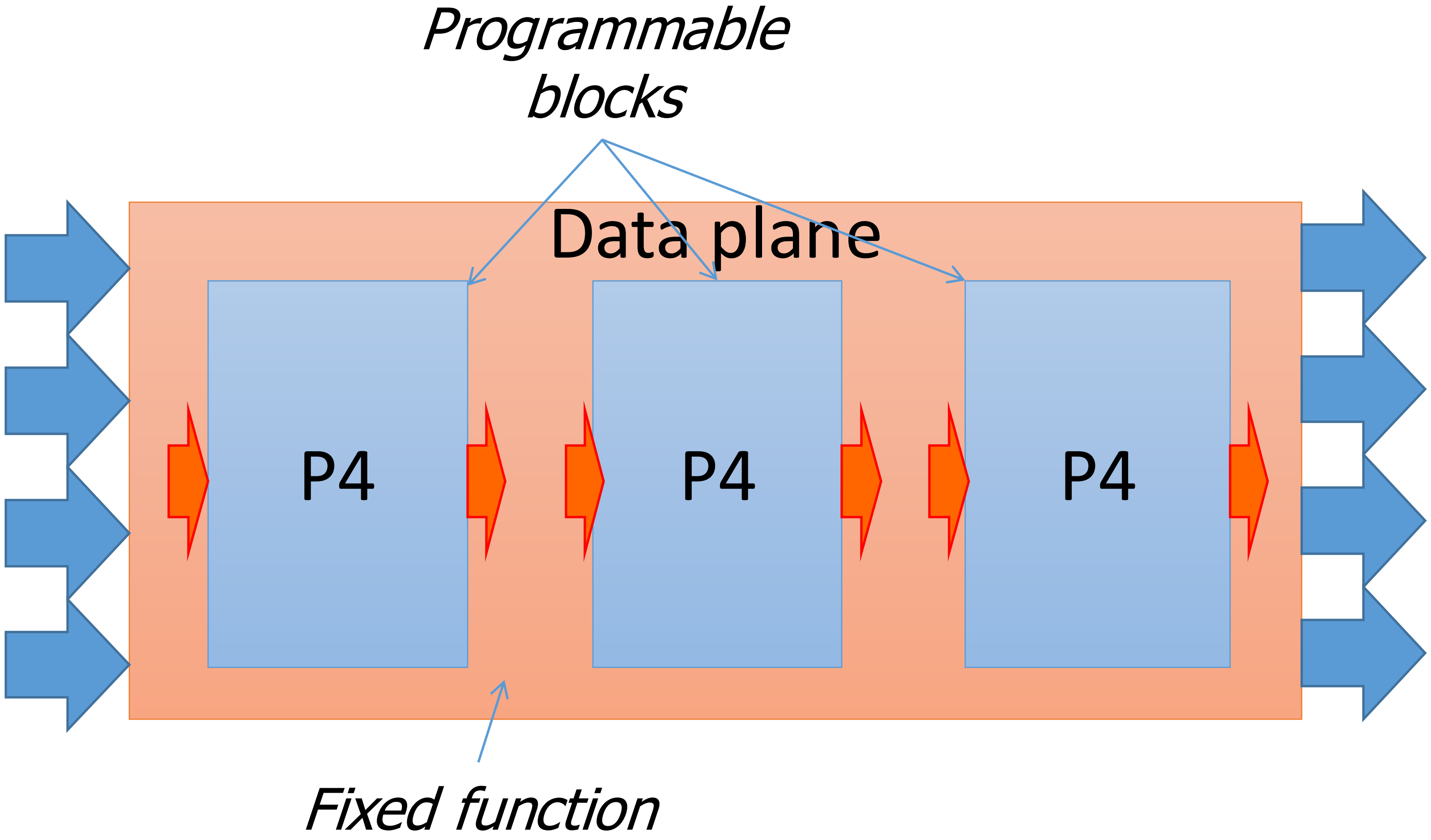

Frontend
P4 program
P4 Compiler
Intermediate Representation
High-Level Intermediate Representation (HLIR)
Backend Targets
p4c-validate & p4c-graphs
p4c-bmv2 behavioral model v2
P4c-hw-target
Target Driver Stack
Outputs
Control Flow & Parse Graphs
JSON config for bmv2, PD-lib
Device-specific config, PD-lib

P4$_{16}$ Language Elements

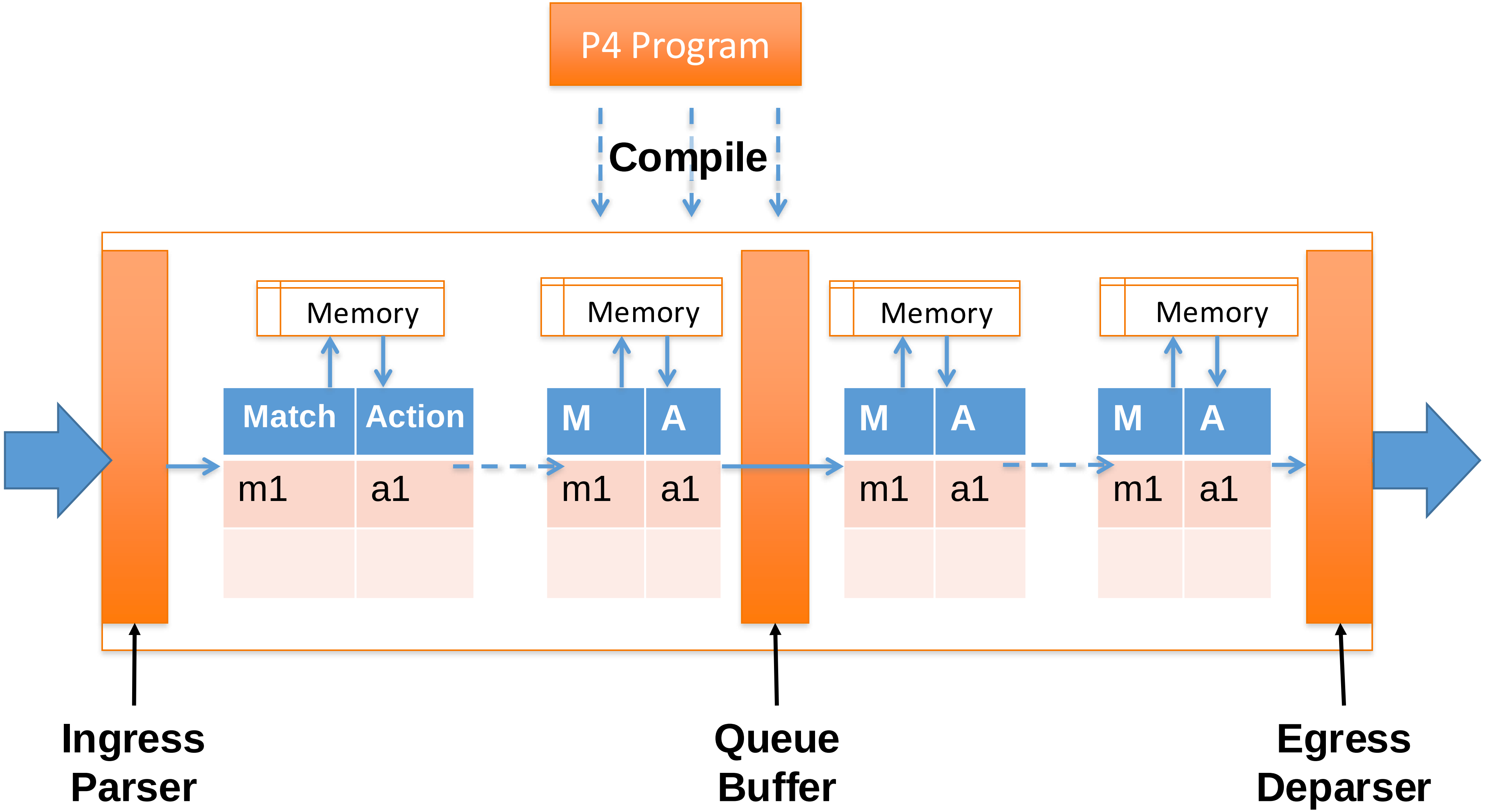
user	Parsers	State machine, bitfield extraction
	Controls	Tables, Actions, control flow statements
	Expresions	Logical expressions
	Data Types	Bistrings, headers, structures, arrays
Vendor-supplied target	Target description	Programmable blocks and their interfaces
	External libraries	Support for specialized components



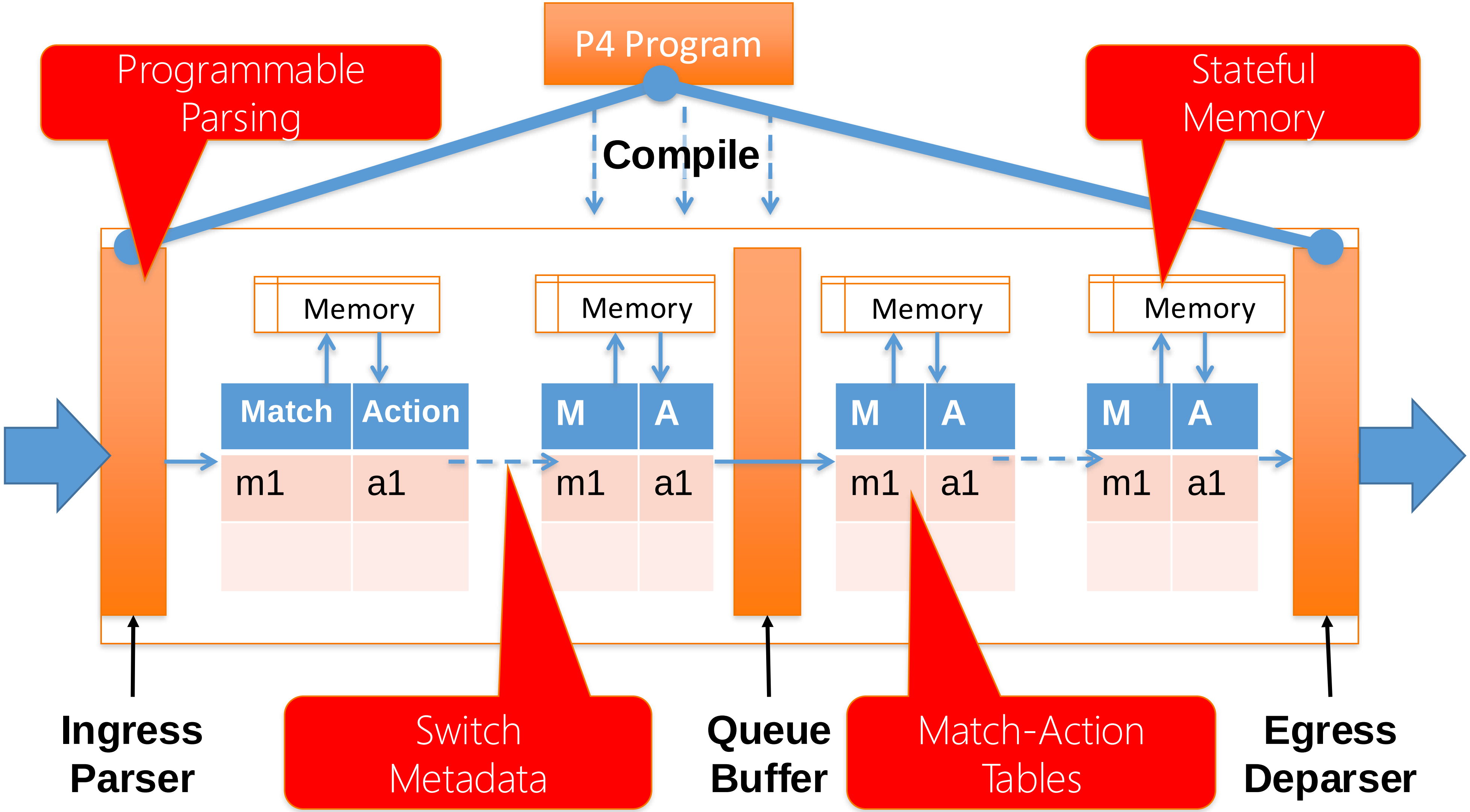
$P4_{16}$ Data Plane Model



Programmable Switches: Capabilities

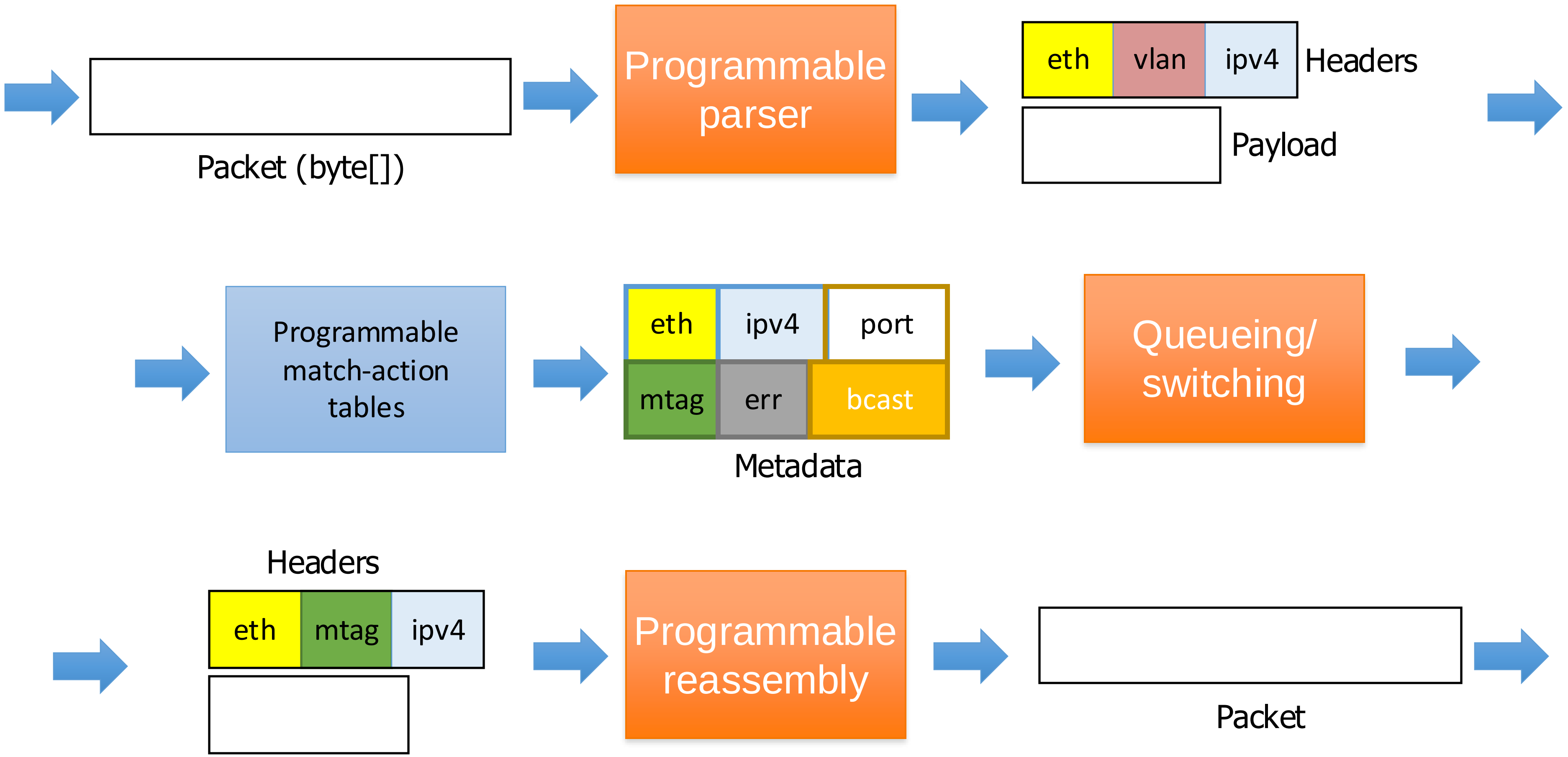


Programmable Switches: Capabilities



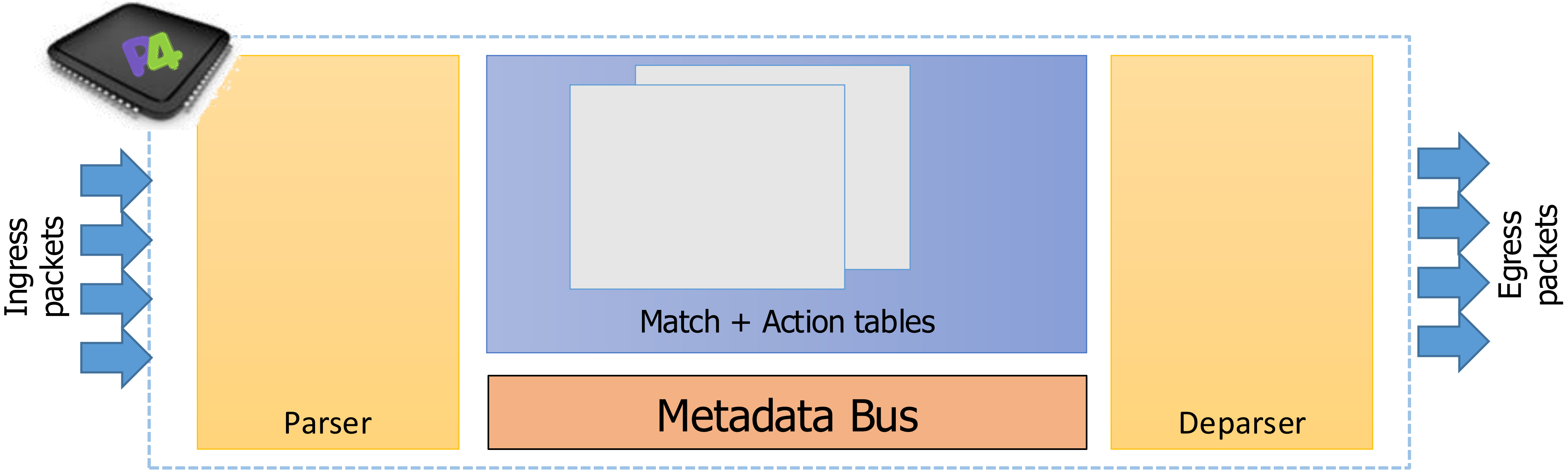
uc3m | Universidad Carlos III de Madrid

Example: Packet Processing Pipeline

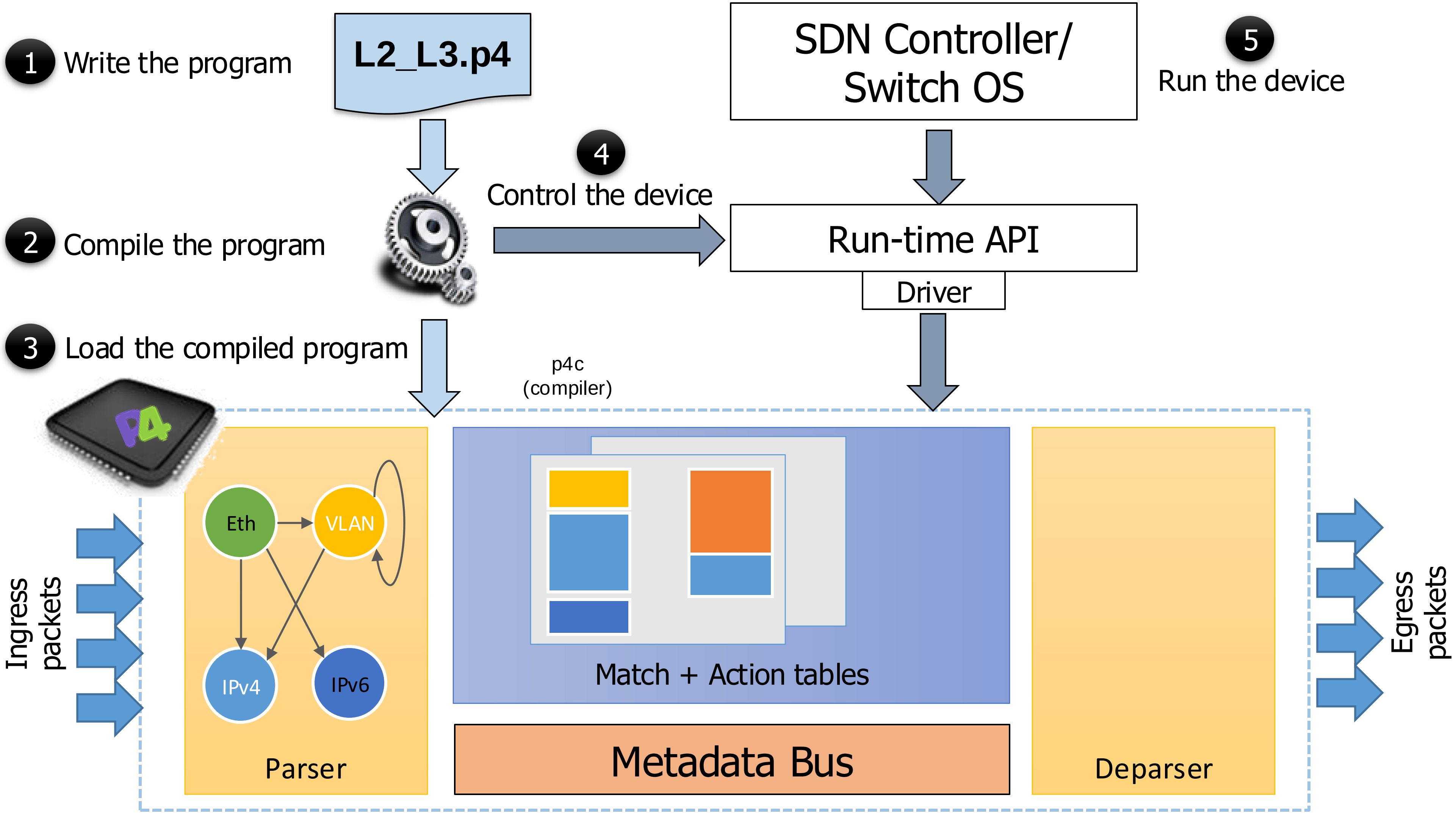


P4-Based Workflow

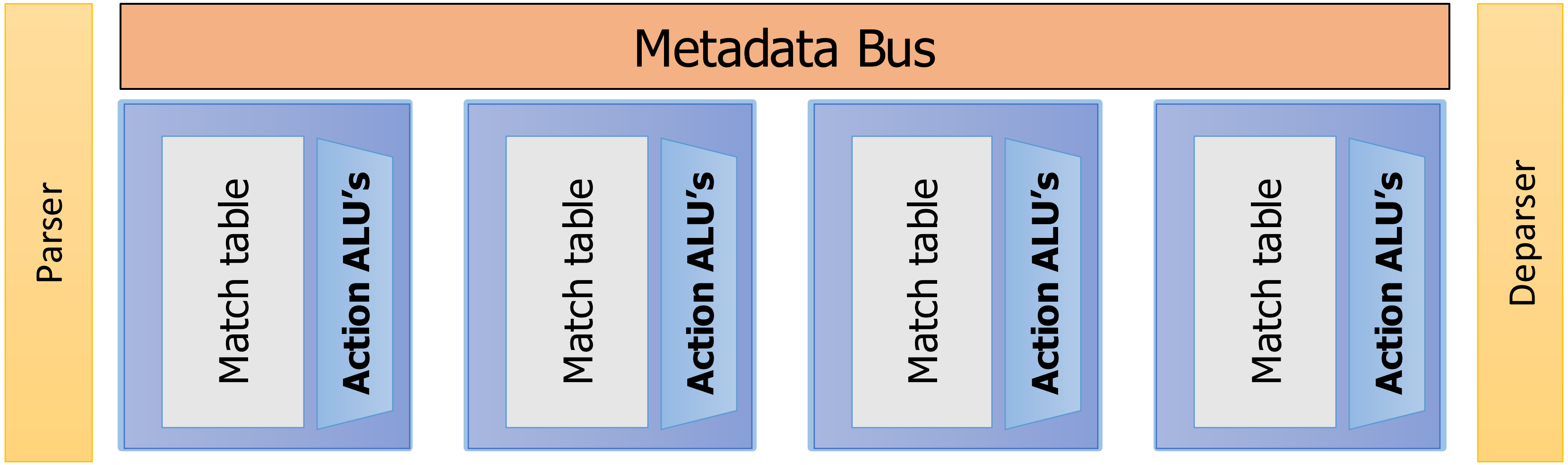
- Device is not programmed yet
  - Parser, tables and deparser are empty
  - Does not know about any packet formats or protocols



P4-Based Workflow



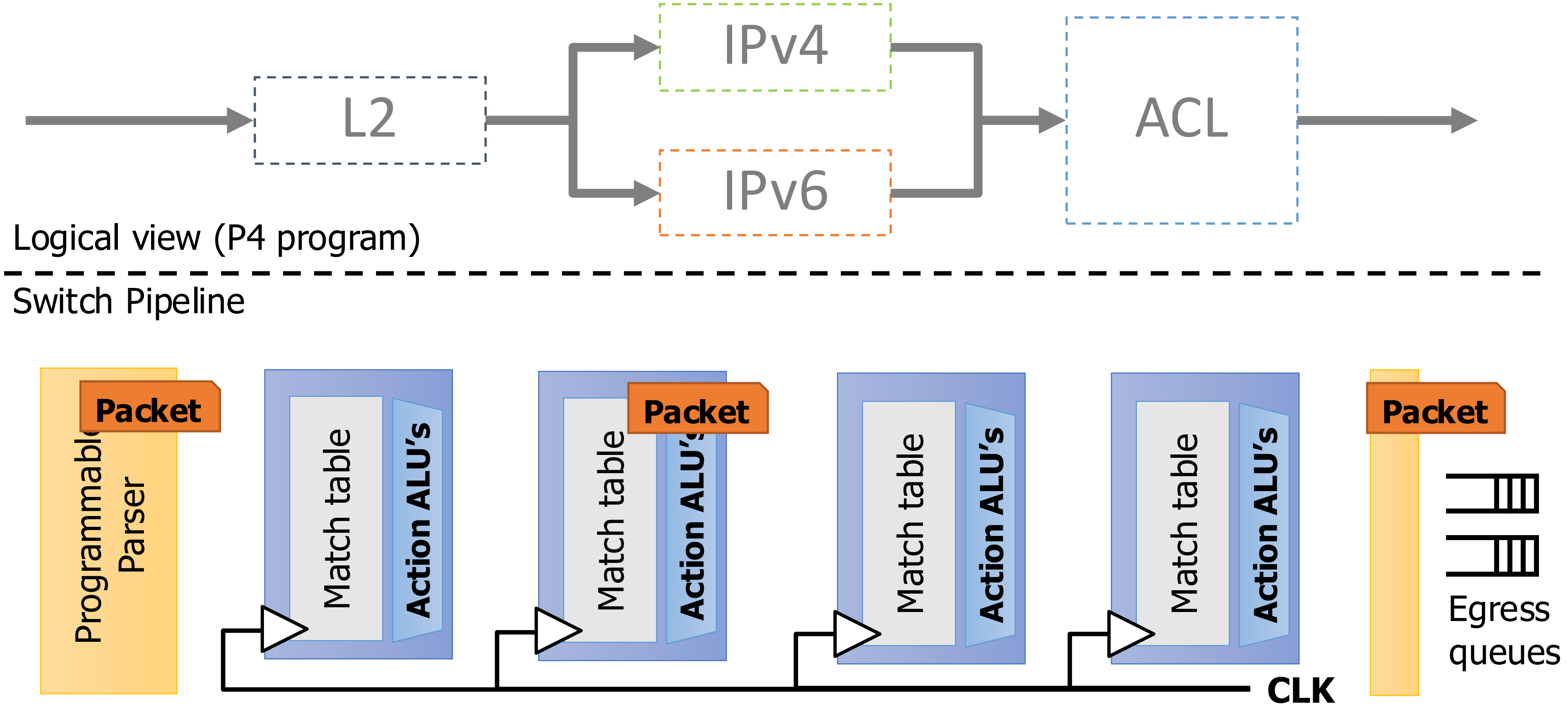
The anatomy of a basic pipeline



- Parser
  - Converts packet data into a metadata (Parsed representation)
- Match + Action Tables
  - Operate on metadata
- Deparser
  - Converts metadata back into a serialized packet
- Metadata Bus
  - Carries the information within the pipeline

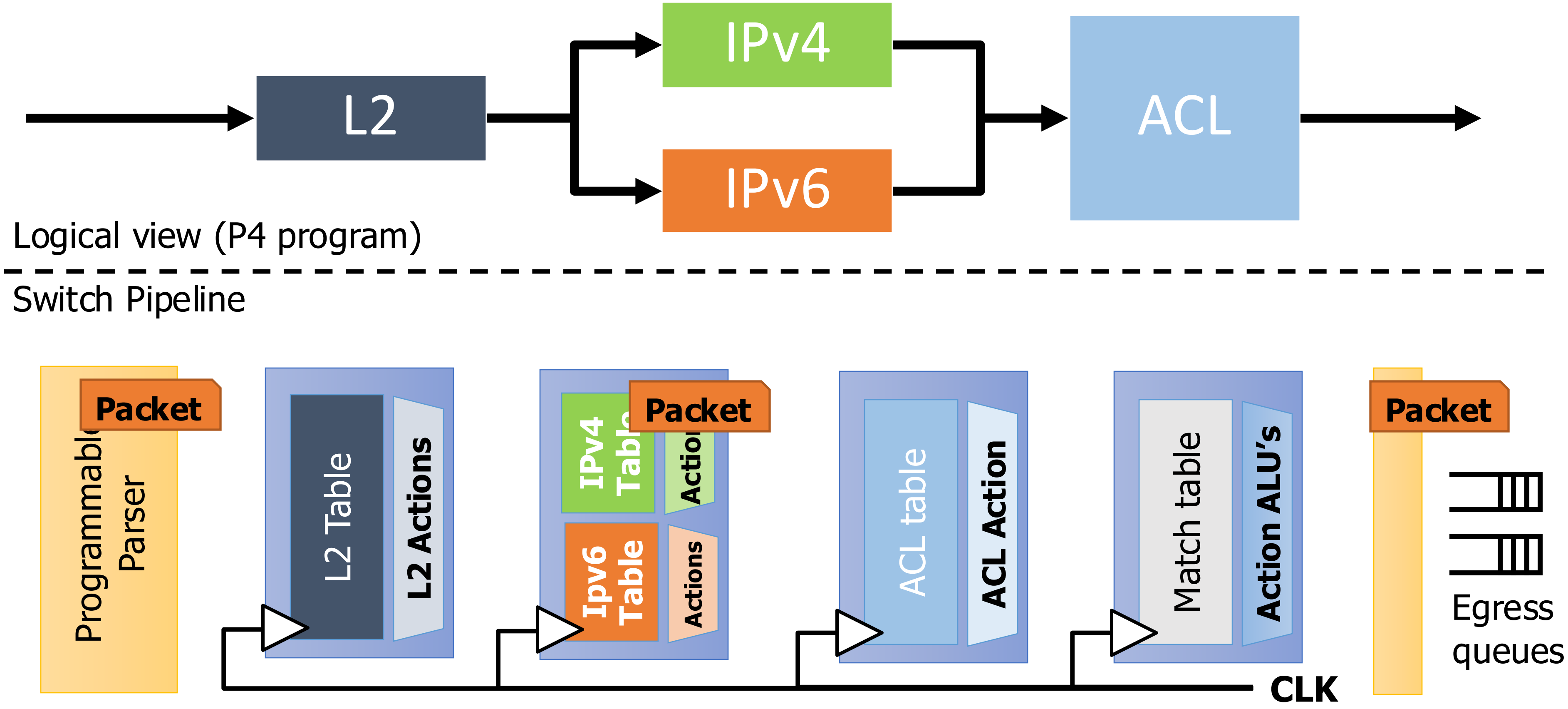
Mapping a P4 Program into a Pipeline

- No Program has been downloaded yet



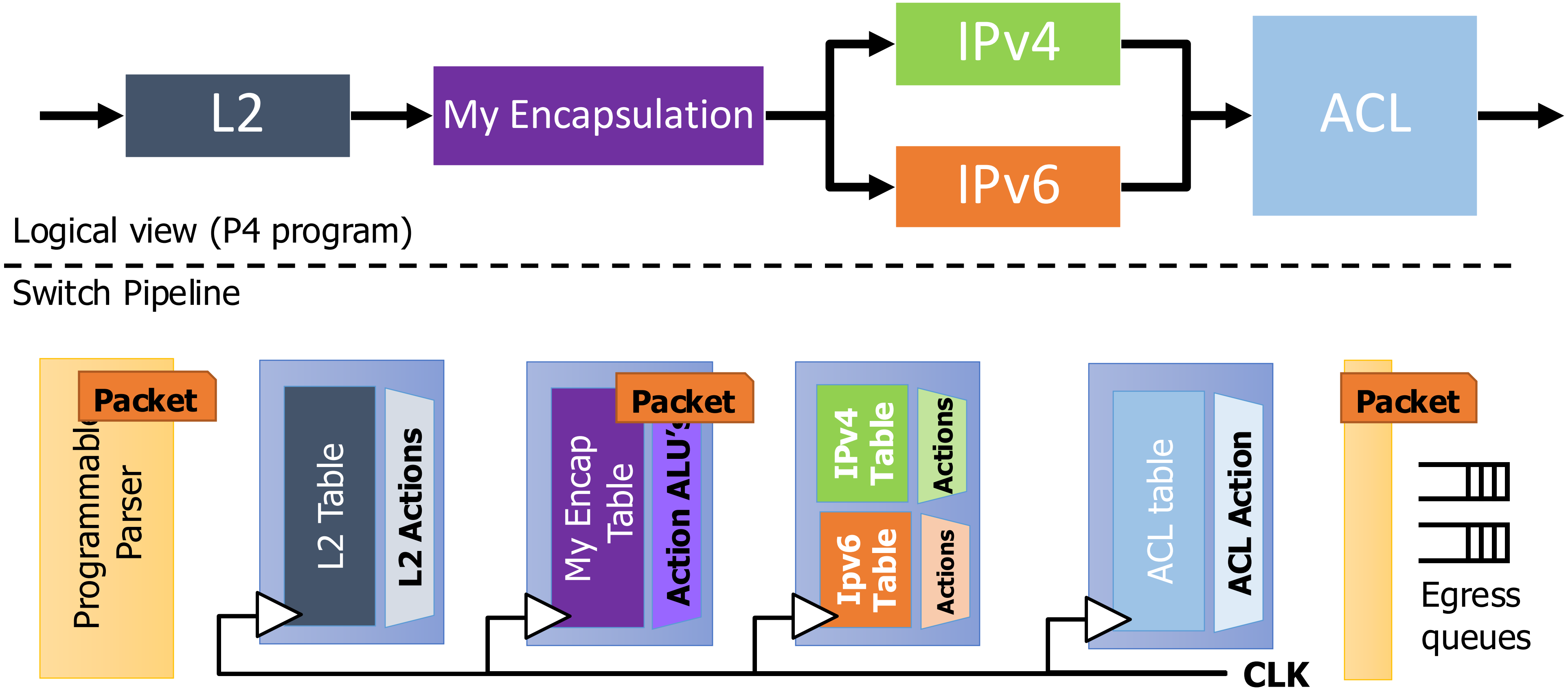
Mapping a P4 Program into a Pipeline

- Program has been compiled and downloaded

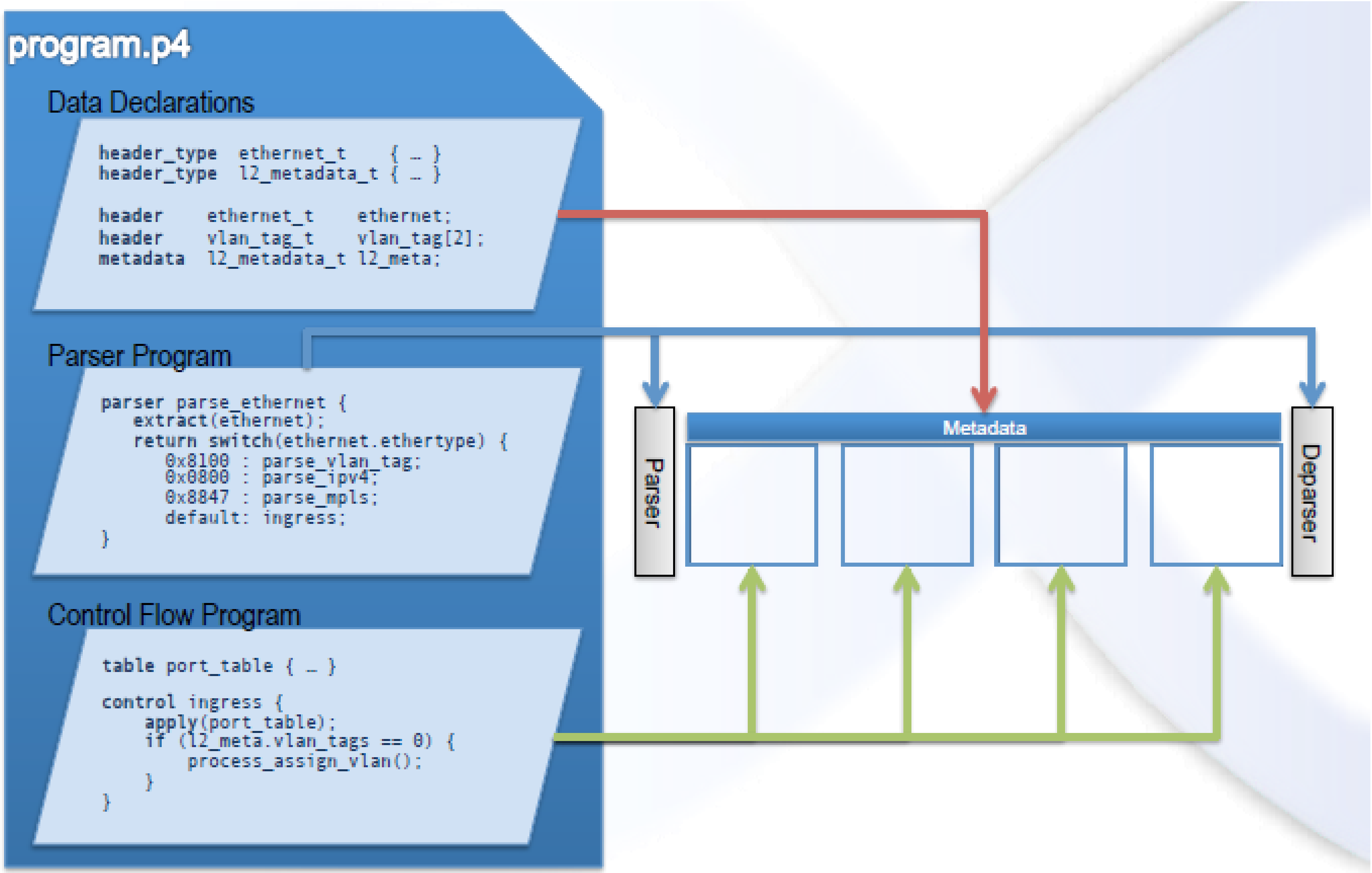


Mapping a P4 Program into a Pipeline

- Re-Program in the field



P4 Program Sections



Data Declarations

```
header_type  ethernet_t    { … }
header_type  l2_metadata_t { … }

header    ethernet_t    ethernet;
header    vlan_tag_t    vlan_tag[2];
metadata  l2_metadata_t l2_meta;
```

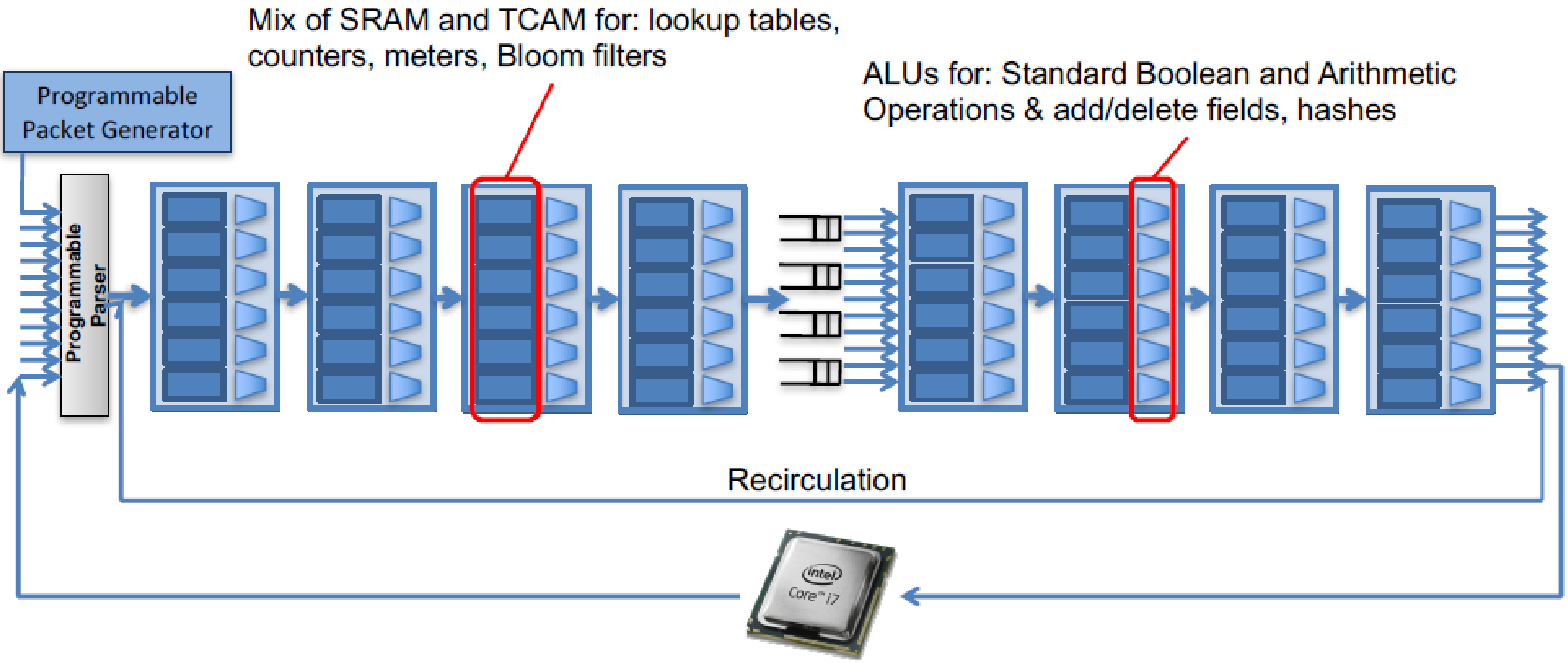
Parser Program

```
parser parse_ethernet {
   extract(ethernet);
   return switch(ethernet.ethertype) {
      0x8100 : parse_vlan_tag;
      0x0800 : parse_ipv4;
      0x8847 : parse_mpls;
      default: ingress;
}
```

Control Flow Program

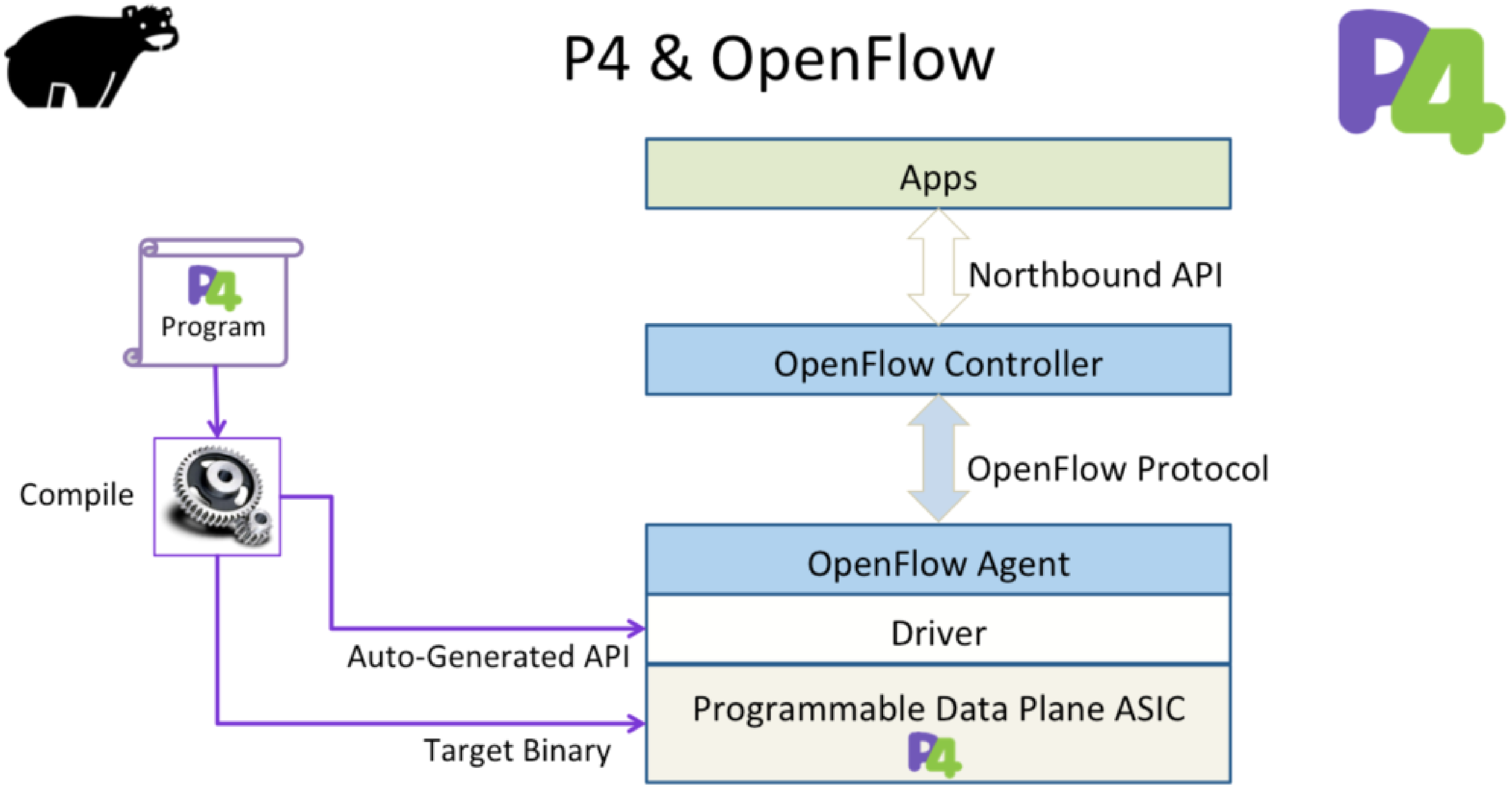
```
table port_table { … }

control ingress {
    apply(port_table);
    if (l2_meta.vlan_tags == 0) {
        process_assign_vlan();
    }
}
```

PISA: Protocol Independent Switch Architecture



Mix of SRAM and TCAM for: lookup tables, counters, meters, Bloom filters
ALUs for: Standard Boolean and Arithmetic Operations & add/delete fields, hashes
Programmable Packet Generator
Programmable Parser
Recirculation
intel
Core i7

P4 & OpenFlow

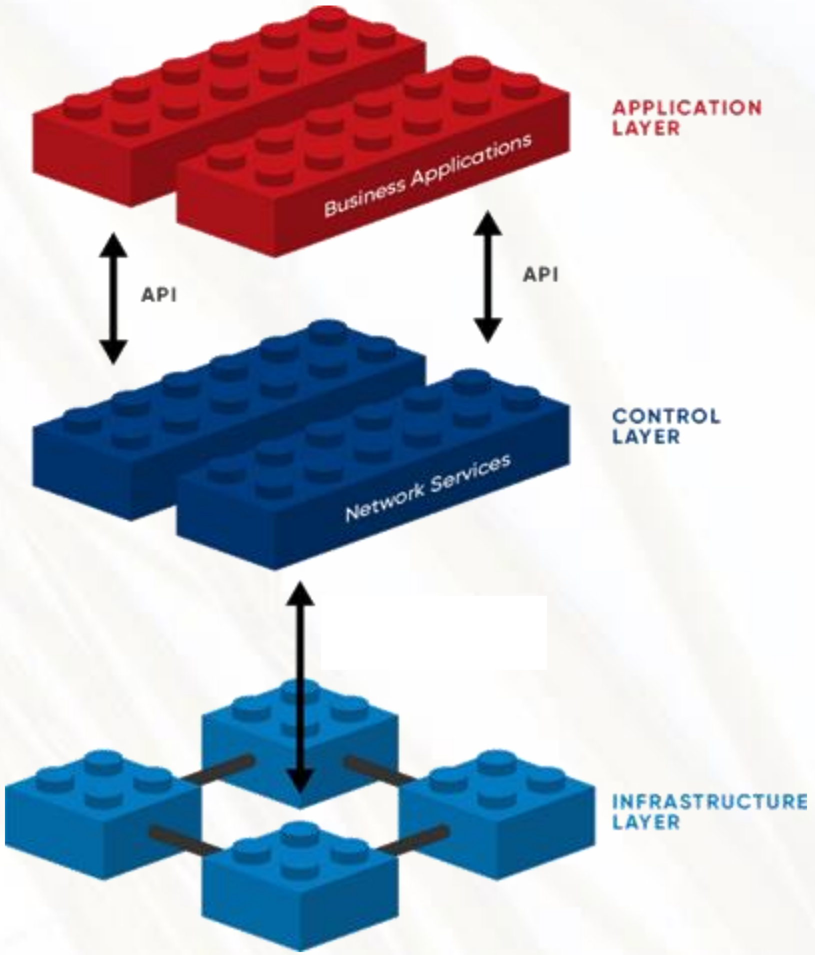


Copyright © 2016 P4 Language Consortium.

P4

Contents

- Why P4?
- What is P4?
- Standards & History
- Use Cases
- P4 Overview
- P4 Language Elements
- Discussion



Why $P4_{16}$

- Clearly defined semantics
  - You can describe what your data plane program is doing
- Expressive
  - Supports a wide range of architectures
- High-level, Target-independent
  - Uses conventional constructs
  - Compiler manages the resources and deals with the hardware
- Type-safe
  - Enforces good software design practices & eliminates “stupid” bugs
- Agility
  - High-speed networking devices become as flexible as any software
- Insight
  - Freely mixing packet headers and intermediate results

P4 Language Components

- Data declarations
  - Packet Headers and Metadata
- Parser Programming
  - Parser Functions (Parser states)
  - Checksum Units
- Packet Flow Programming
  - Actions
    - Primitive and compound actions
    - Counters, Meters, Registers
  - Tables
    - Match keys & Attributes
  - Control Functions (Imperative Programs)
- No: pointers, loops, recursion, floating point!!

Limitations of P4$_{16}$

- The core P4 language is very small
  - Highly portable among many targets
  - But very limited in expressivity
- Accelerators can provide additional functionality
  - May not be portable between different targets
  - Under construction: library of standard accelerators