Sharif University of Technology

Computer Engineering Department

Software-Defined Networking

Ali Movaghar

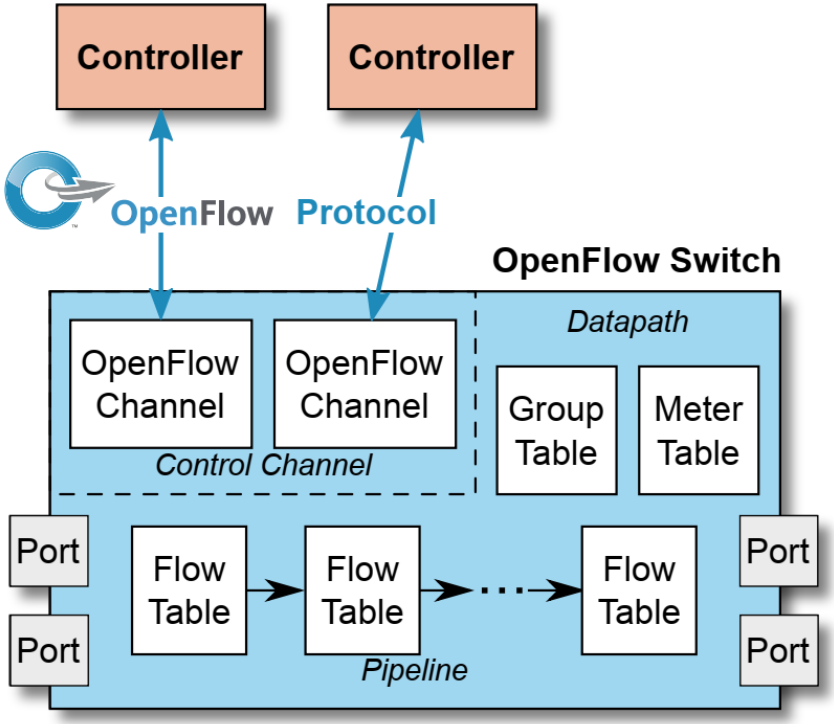
Mohammad Hosseini

Data Plane Programming
Part 2

TA: Iman Rahmati & Farbod Shahinfar

SDN and OpenFlow

- SDN has changed the way we operate big networks.
  - Disaggregation
  - Logically centralized control



- OpenFlow is a key protocol that enabled SDN.
  - Standardized protocol to interact with switch
  - Add/remove table entries, query statistics, etc.
  - Configuring specialized objects (e.g. meters)
  - Runtime control over the network devices.

- To this point,
  - the initial idea of SDN was data plane agnostic
  - SDN was entirely focused on opening the control plane to programmability

SDN and OpenFlow

Can OpenFlow solve all the problems of networks?

Has SDN achieved its aims?

Dependency on Forwarding Pipeline

- Control applications were implicitly tied to a particular forwarding pipeline.
  - We don't want to limit ourselves to a single vendor's pipeline
  - This is analogous to writing a Java program that can run on an x86 processor and is not easily ported to an ARM processor.

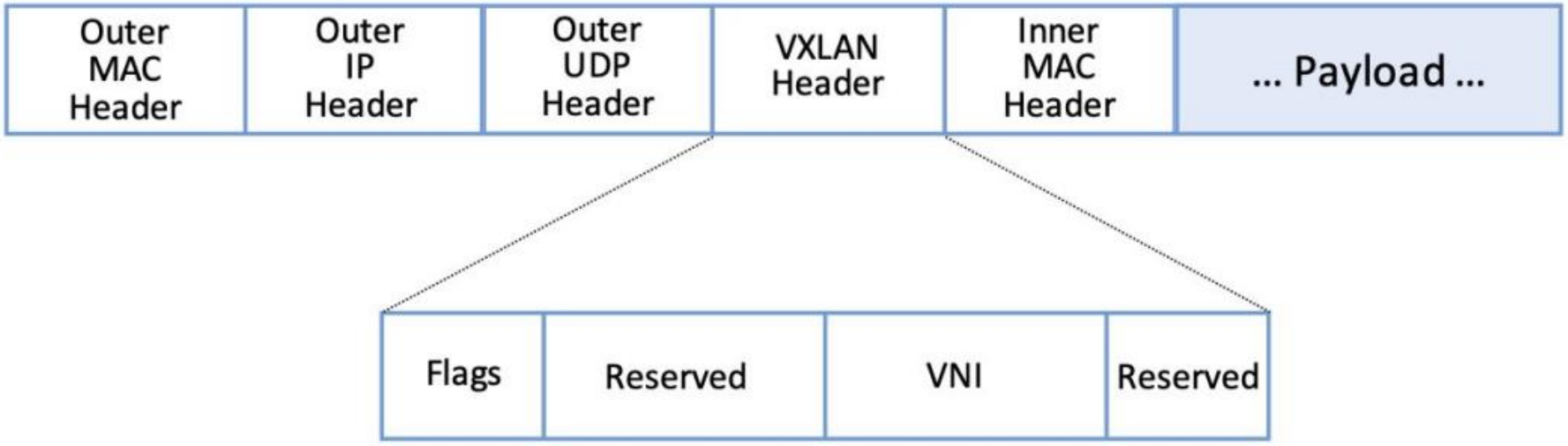
- We need an **abstract way to specify a pipeline's behavior**, that can in turn be mapped onto the physical pipeline of any given switch.

Proliferation of Header 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

New Headers

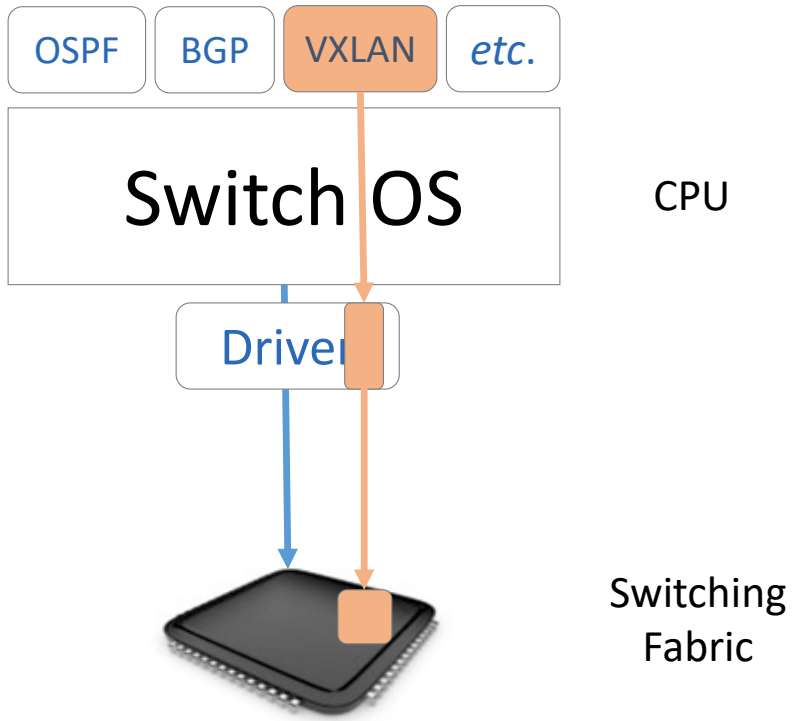
- The protocol stack changed in unexpected ways.
- The assumption that "*all header fields we might need to match against are well-known*" is flawed.



- Hardware needs to change

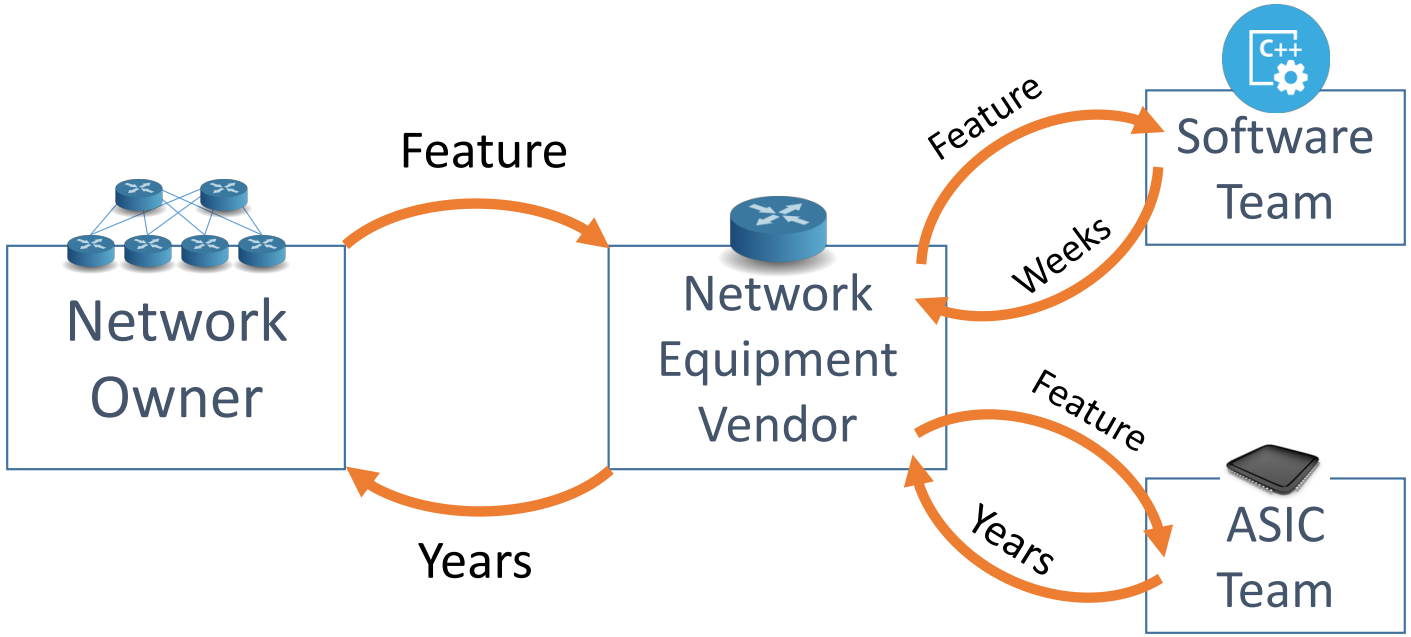
Developing a New Network Feature

- If existing hardware meets the requirements of the new feature, the development is done in the control plane.
- Otherwise, data plane needs to be changed as well.



Developing a New Network Feature

❖Many new network features and protocols require data plane changes



❖Network protocol/feature evolution is too slow

❖You need a fork-lift upgrade, at huge expense

OpenFlow and Hardware Constraints

- OpenFlow is protocol dependent (only supports a set of existing protocols) because of constraints of traditional switching chips.
  - The OpenFlow protocol has had to map its functions onto the capabilities of existing chips.
  - Mapping to existing switching chips enabled quick adoption.

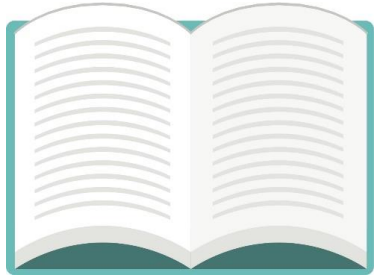
What do we want from SDN?

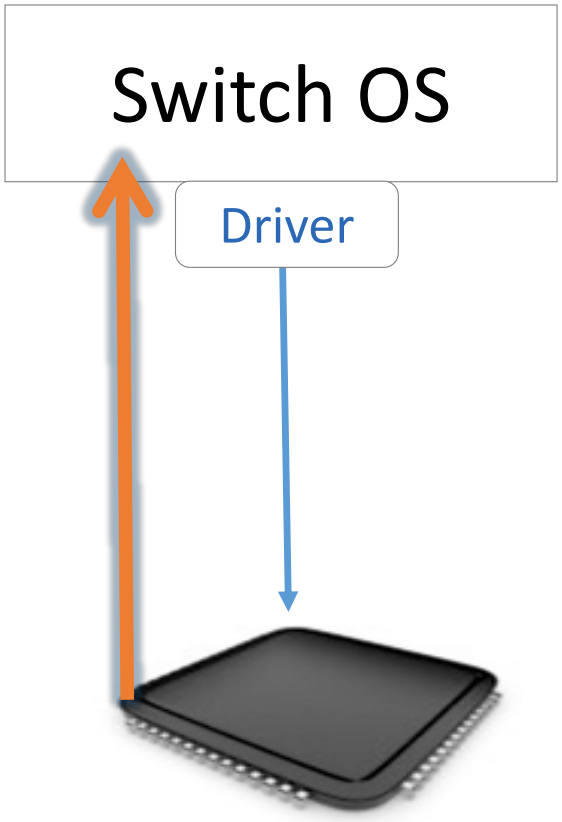
- Program networks and their elements
- Protocol-independent processing
- Control and repurpose in the field

Status Quo

❖Network systems are built "bottom-up"

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





Fixed-function switch

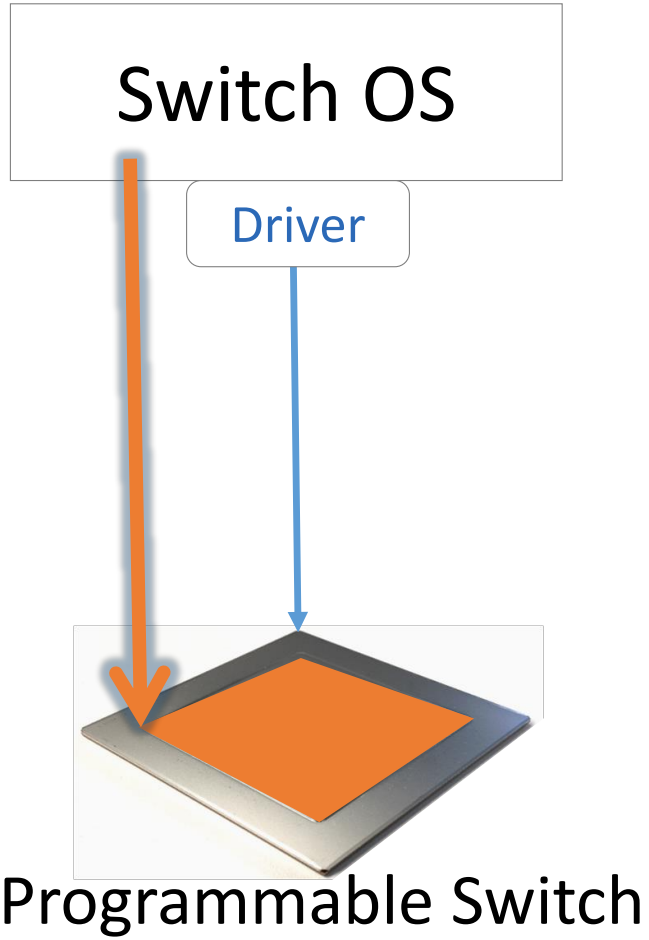
A Better Approach

❖"Top-down design"

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

```
table int_table {
  reads {
    ip.protocol;
  }
  actions {
    export_queue_latency;
  }
}
```

```
action export_queue_latency (sw_id) {
  add_header(int_header);
  modify_field(int_header.kind, TCP_OPTION_INT);
  modify_field(int_header.len, TCP_OPTION_INT_LEN);
  modify_field(int_header.sw_id, sw_id);
  modify_field(int_header.q_latency,
               intrinsic_metadata.deq_timedelta);
  add_to_field(tcp.dataOffset, 2);
  add_to_field(ipv4.totalLen, 8);
  subtract_from_field(ingress_metadata.tcpLength,
                      12);
}
```



Domain Specific Processors

General Computing

Java

Compiler

CPU

Graphics

OpenCL

Compiler

GPU

Signal Processing

Matlab

Compiler

DSP

Machine Learning

TensorFlow

Compiler

TPU

Networking

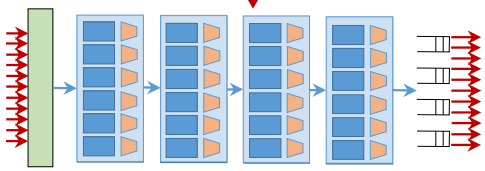
Language

Compiler

?

P4

Compiler



PISA

Benefits of Data Plane Programming

- **New features**
  - Realize new protocols and behaviors very easily and quickly

- **You keep your own ideas**
  - Proprietary features

Benefits of Data Plane Programming

❖Reducing complexity

➢ Remove unnecessary features

IPv4 and IPv6 routing
- Unicast Routing
  - Routed Ports & SVI
  - VRF
- Unicast RPF
  - Strict and Loose
- ~~Multicast~~
- ~~PIM-SM/DM & PIM-Bidir~~

Ethernet switching
- ~~VLAN Flooding~~
- MAC Learning & Aging
- STP state
- ~~VLAN Translation~~

Load balancing
- ~~LAG~~
- ECMP & WCMP
- Resilient Hashing
- ~~Flowlet Switching~~

Fast Failover
– LAG & ECMP

Tunneling
- IPv4 and IPv6 Routing & Switching
- ~~IP-in-IP (6in4, 4in4)~~
- VXLAN, NVGRE, GENEVE & GRE
- ~~Segment Routing, ILA~~

~~MPLS~~
- ~~LER and LSR~~
- ~~IPv4/v6 routing (L3VPN)~~
- ~~L2 switching (EoMPLS, VPLS)~~
- ~~MPLS over UDP/GRE~~

ACL
- MAC ACL, IPv4/v6 ACL, RACL
- ~~QoS ACL, System ACL, PBR~~
- Port Range lookups in ACLs

QOS
- QoS Classification & marking
- ~~Drop profiles/WRED~~
- ~~RoCE v2 & FCoE~~
- CoPP (Control plane policing)

~~NAT and L4 Load Balancing~~

Security Features
- ~~Storm Control, IP Source Guard~~

Monitoring & Telemetry
- ~~Ingress Mirroring and Egress Mirroring~~
- Negative Mirroring
- ~~Sflow~~
- INT

Counters
- Route Table Entry Counters
- ~~VLAN/Bridge Domain Counters~~
- Port/Interface Counters

Protocol Offload
- BFD, OAM

Multi-chip Fabric Support
- ~~Forwarding, QOS~~

Benefits of Data Plane Programming

❖Efficient use of resources

- ➢ Flexible use of tables
- ➢ Achieve the biggest bang of your buck

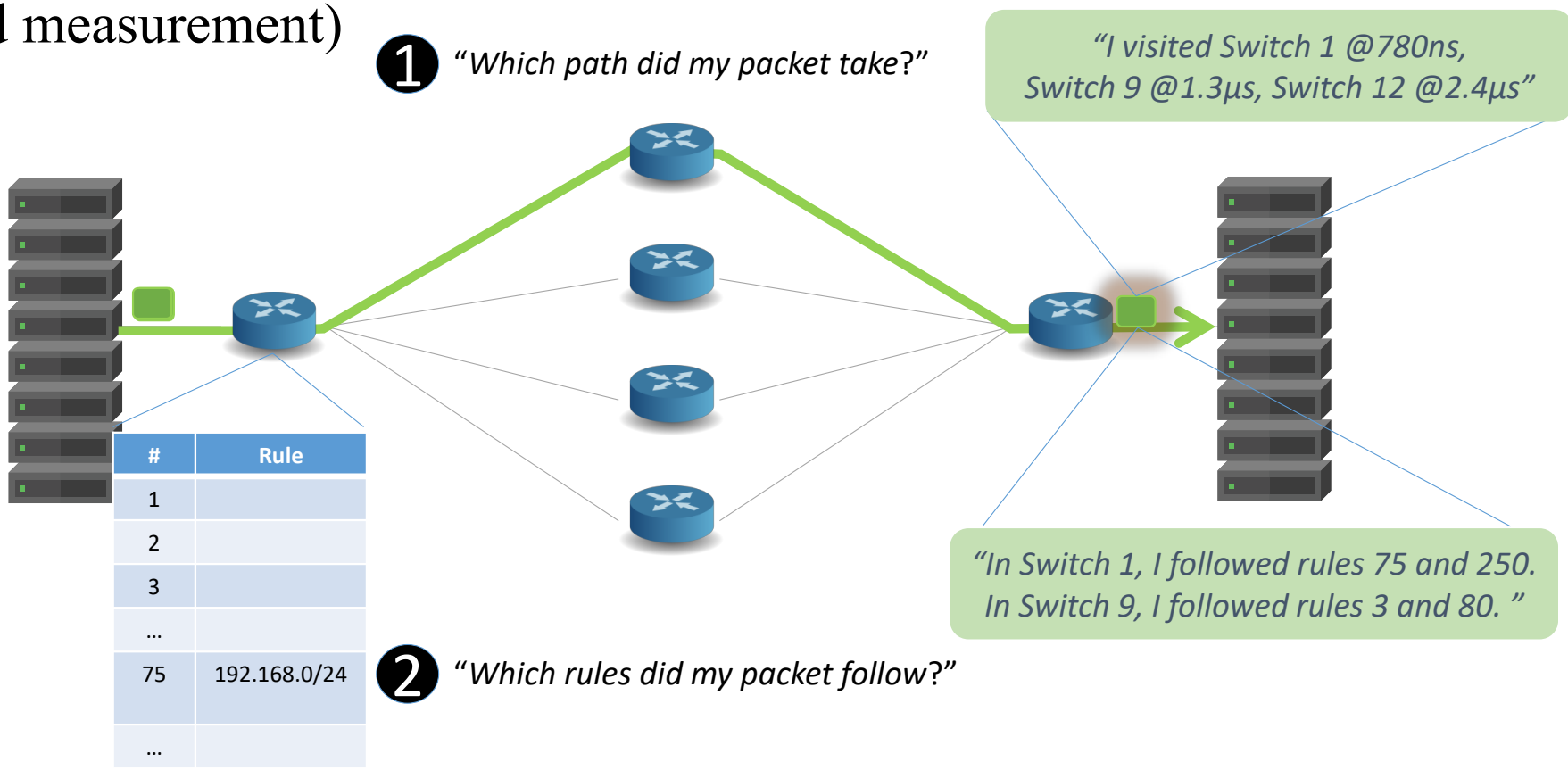
Benefits of Data Plane Programming

- **Modularity**
  - Compose forwarding behavior from libraries
- **Portability**
  - Specify forwarding behavior once, compile to many devices (ASICs, FPGAs, software switches, NPU)
- **Software style development**
  - Rapid design cycle, fast innovation, fix data plane bugs in the field

Benefits of Data Plane Programming

❖Greater visibility

- Visibility into what our network is exactly doing (monitoring and measurement)



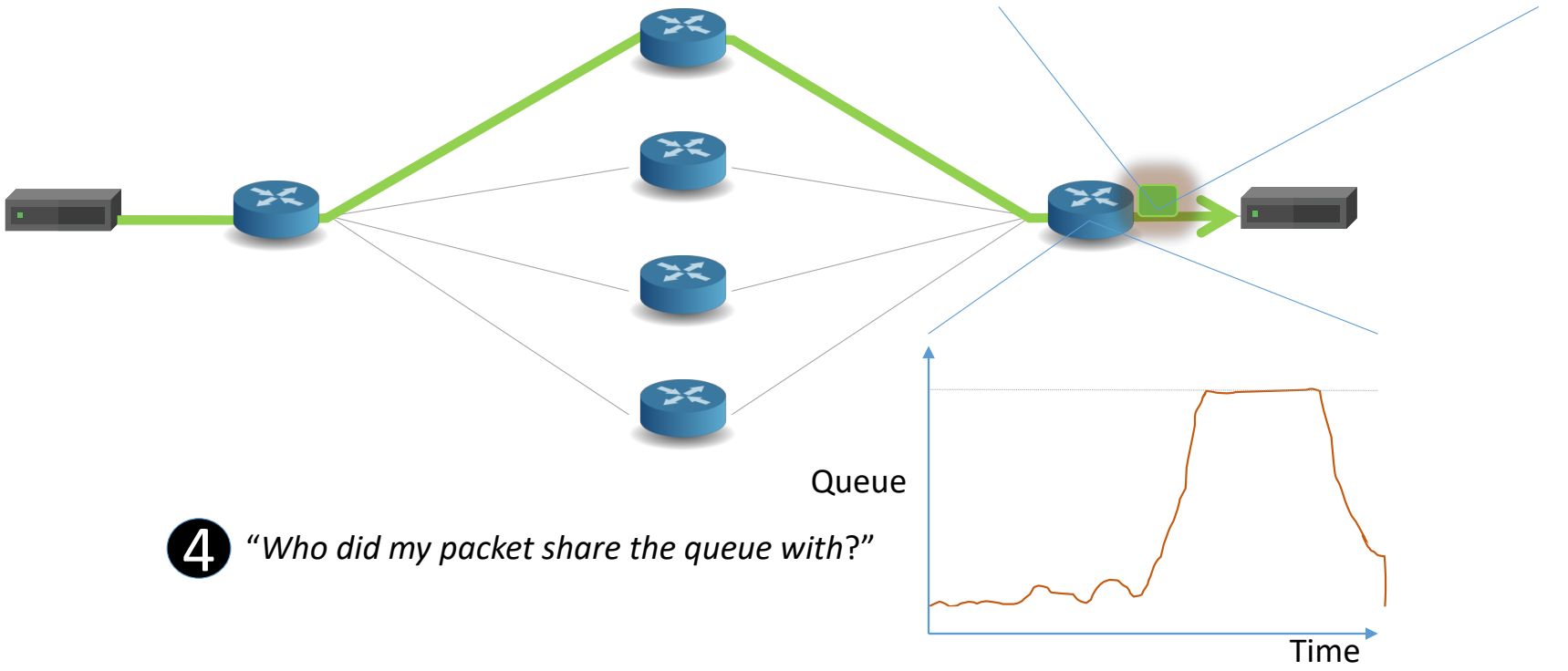
Benefits of Data Plane Programming

❖Greater visibility

- ➢ Visibility into what our network is exactly doing (monitoring and measurement)

❸ *"How long did my packet queue at each switch?"*

"Delay: 100ns, 200ns, 19740ns"

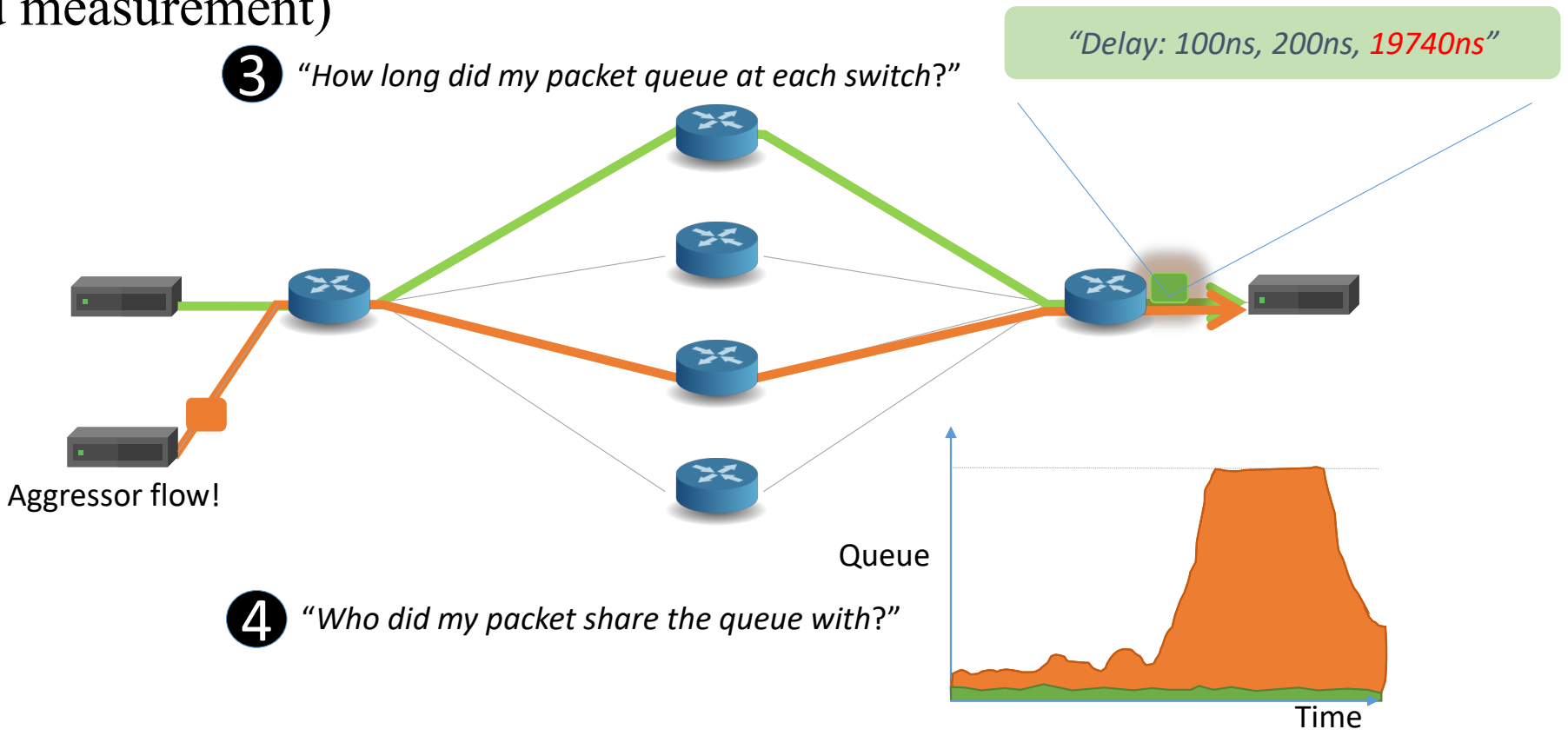


❹ *"Who did my packet share the queue with?"*

Benefits of Data Plane Programming

❖Greater visibility

- Visibility into what our network is exactly doing (monitoring and measurement)



Benefits of Data Plane Programming

❖Greater visibility

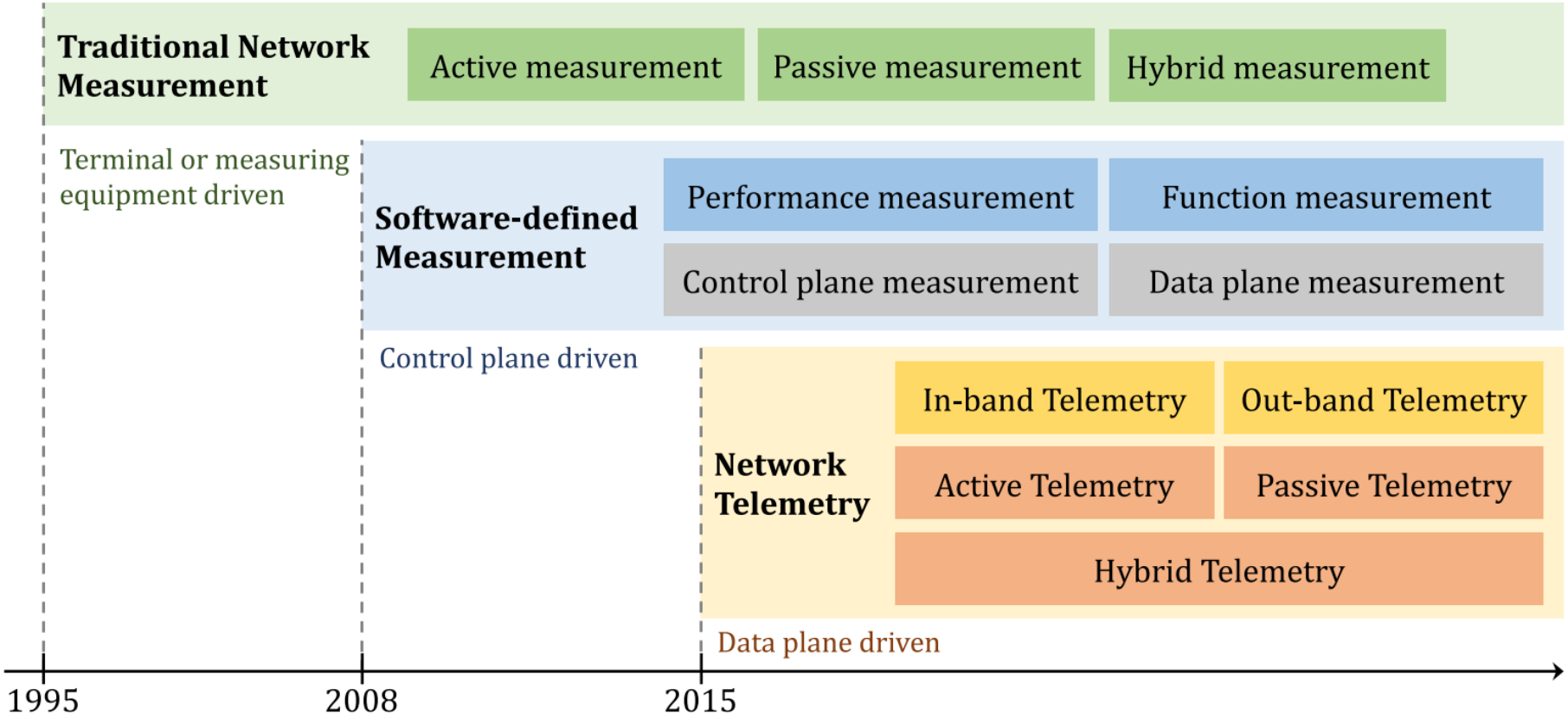
- Visibility into what our network is exactly doing (monitoring and measurement)
  - *"Which path did my packet take?"*
  - *"Which rules did my packet follow?"*
  - *"How long did it queue at each switch?"*
  - *"Who did it share the queues with?"*

  - They represent the complete ground truth of what's happening in the network.

- New techniques for diagnostic and telemetry

Benefits of Data Plane Programming

Network Measurement

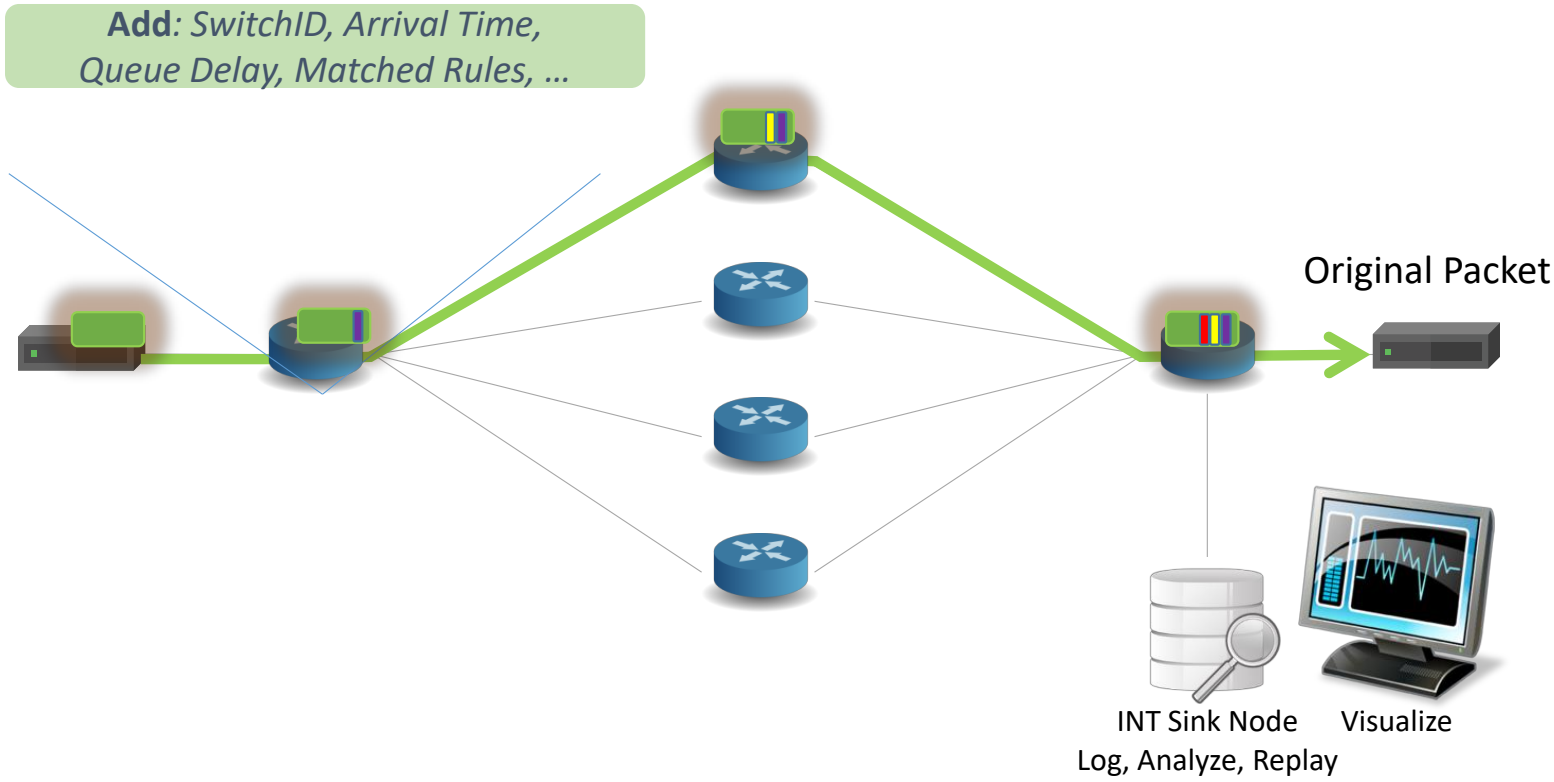


"In-Band Network Telemetry: a Survey", Computer Networks, 2021.

Benefits of Data Plane Programming

❖INT: Inband Network Telemetry

- INT combines data packet forwarding with network measurement
- INT data and user data usually share the same link or even the same packet



P4 Concepts

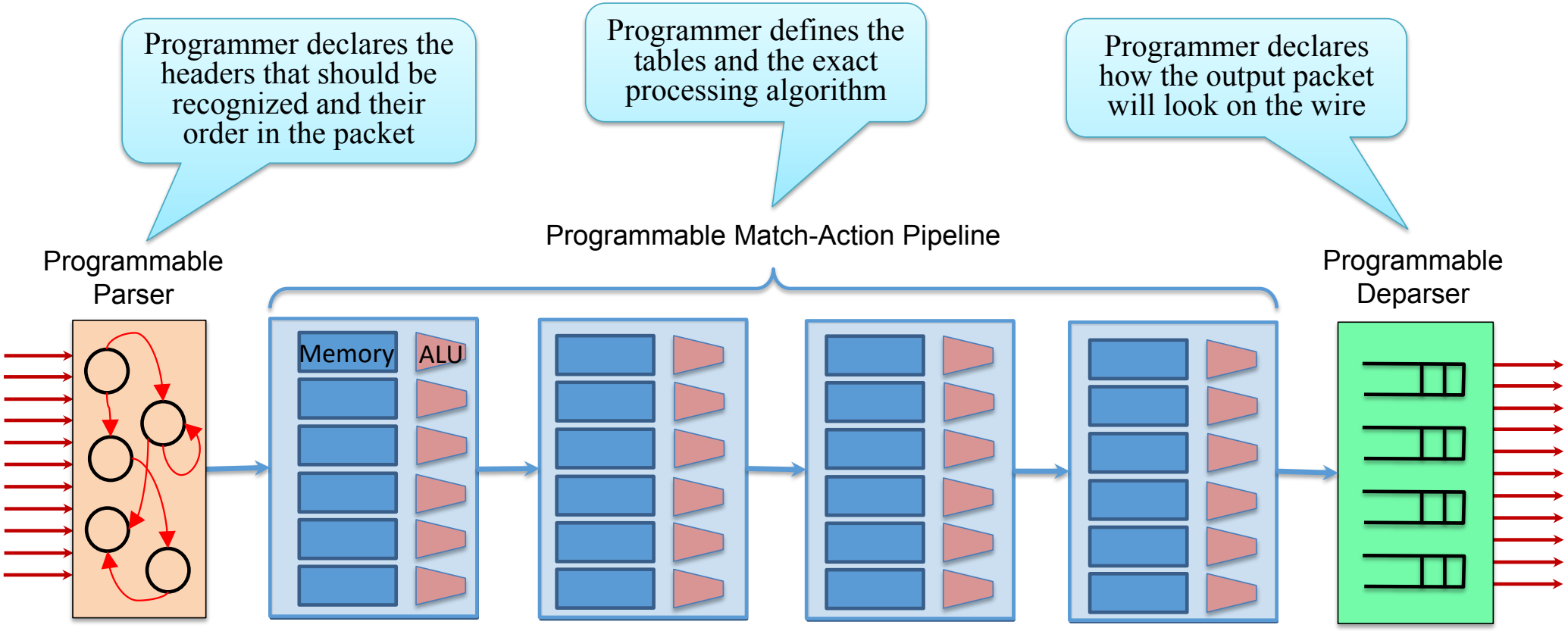
Brief History and Trivia

- May 2013: Initial idea and the name "P4"
- July 2014: First paper (SIGCOMM CCR)
- Aug 2014: First $P4_{14}$ Draft Specification (v0.9.8)
- Sep 2014: $P4_{14}$ Specification released (v1.0.0)
- Jan 2015: $P4_{14}$ v1.0.1
- Mar 2015: $P4_{14}$ v1.0.2
- Nov 2016: $P4_{14}$ v1.0.3
- May 2017: $P4_{14}$ v1.0.4
- Nov 2018: $P4_{14}$ v1.1.0

- Apr 2016: $P4_{16}$ – first commits
- Dec 2016: First $P4_{16}$ Draft Specification
- May 2017: $P4_{16}$ Specification released (v1.0.0)
- Nov 2018: $P4_{16}$ v1.1.0
- Oct 2019: $P4_{16}$ v1.2.0
- May 2021: $P4_{16}$ v1.2.2

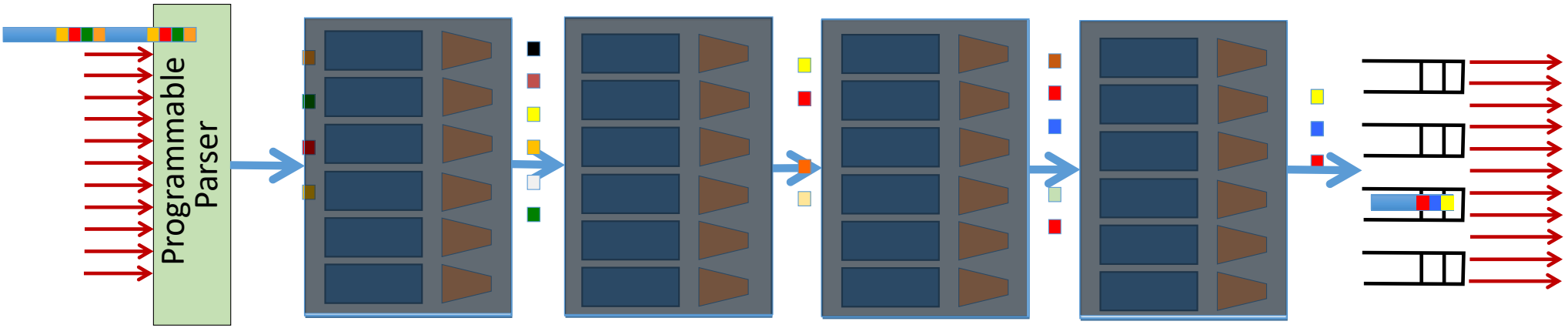
PISA: Protocol-Independent Switch Architecture

- Abstract machine model of programmable switch architecture
- $P4_{14}$ targeted PISA-like devices

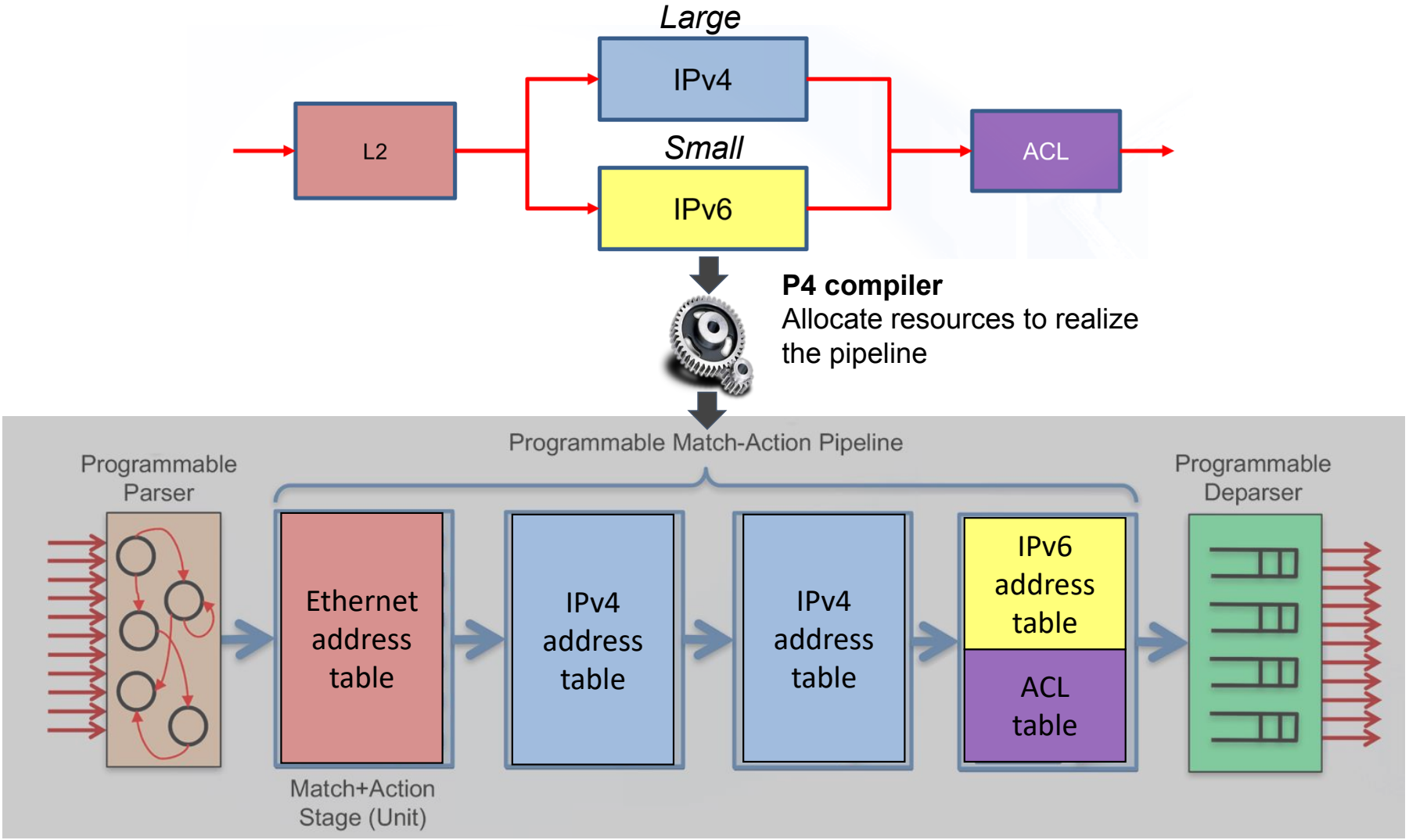


PISA: Protocol-Independent Switch Architecture

- 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)



Compiling a simple logical pipeline on PISA



P4 Compiler Workflow

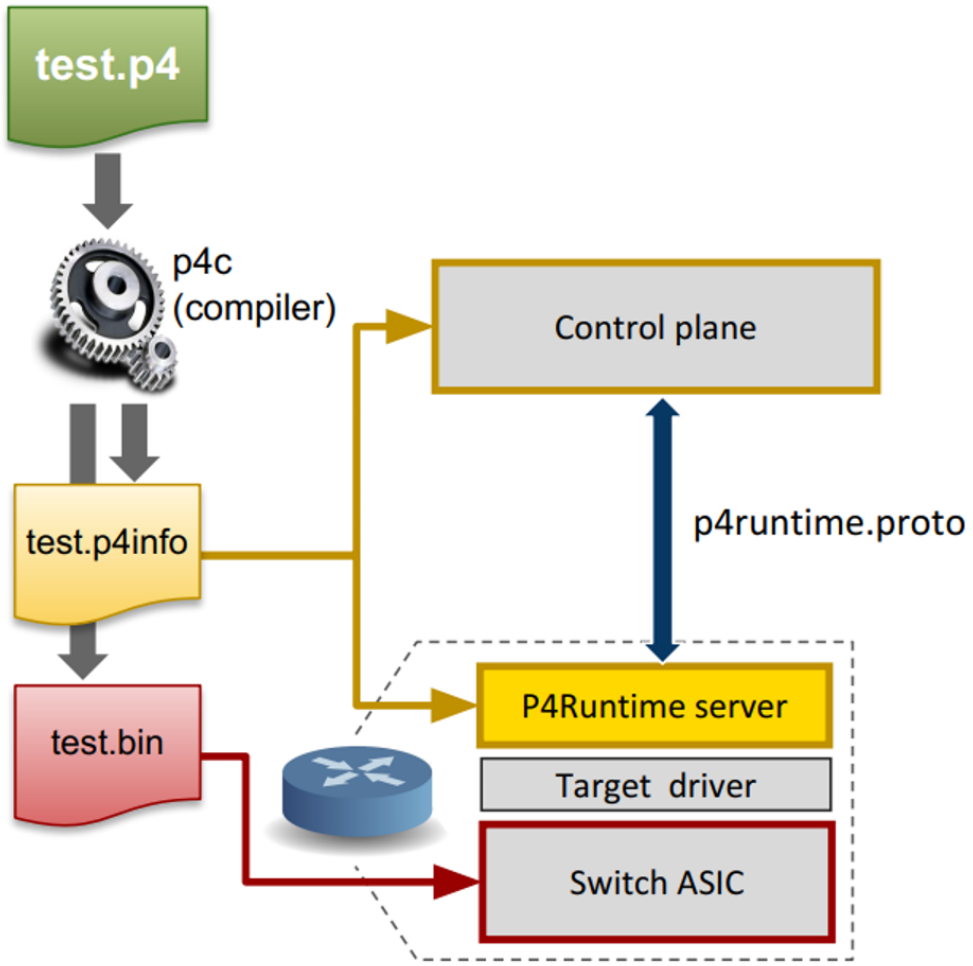
P4 compiler generates 2 files

1- Target-specific binaries

- Used to configure switch pipeline (data plane)
  (e.g. binary config for ASIC, bitstream for FPGA, etc,)

2- P4Info file

- Describes "schema" of pipeline for runtime control
- P4 program attributes
  - IDs of tables, actions, parameters, etc.
  - Table structure, action parameters, etc.
- Target-independent compiler output
  (same P4Info for SW switch, ASIC, etc.)



Evolution of the Language

- $P4_{14}$ was based on a reference forwarding model inspired by PISA

- It fits with most switching ASICs, but it doesn't fit with some other platforms like
  - FPGAs which have more freedom,
  - NPUs which are structured differently,
  - or just regular CPUs.

- There are targets that may want to use different kinds of models

- $P4_{16}$ separated the target-architecture from the language itself

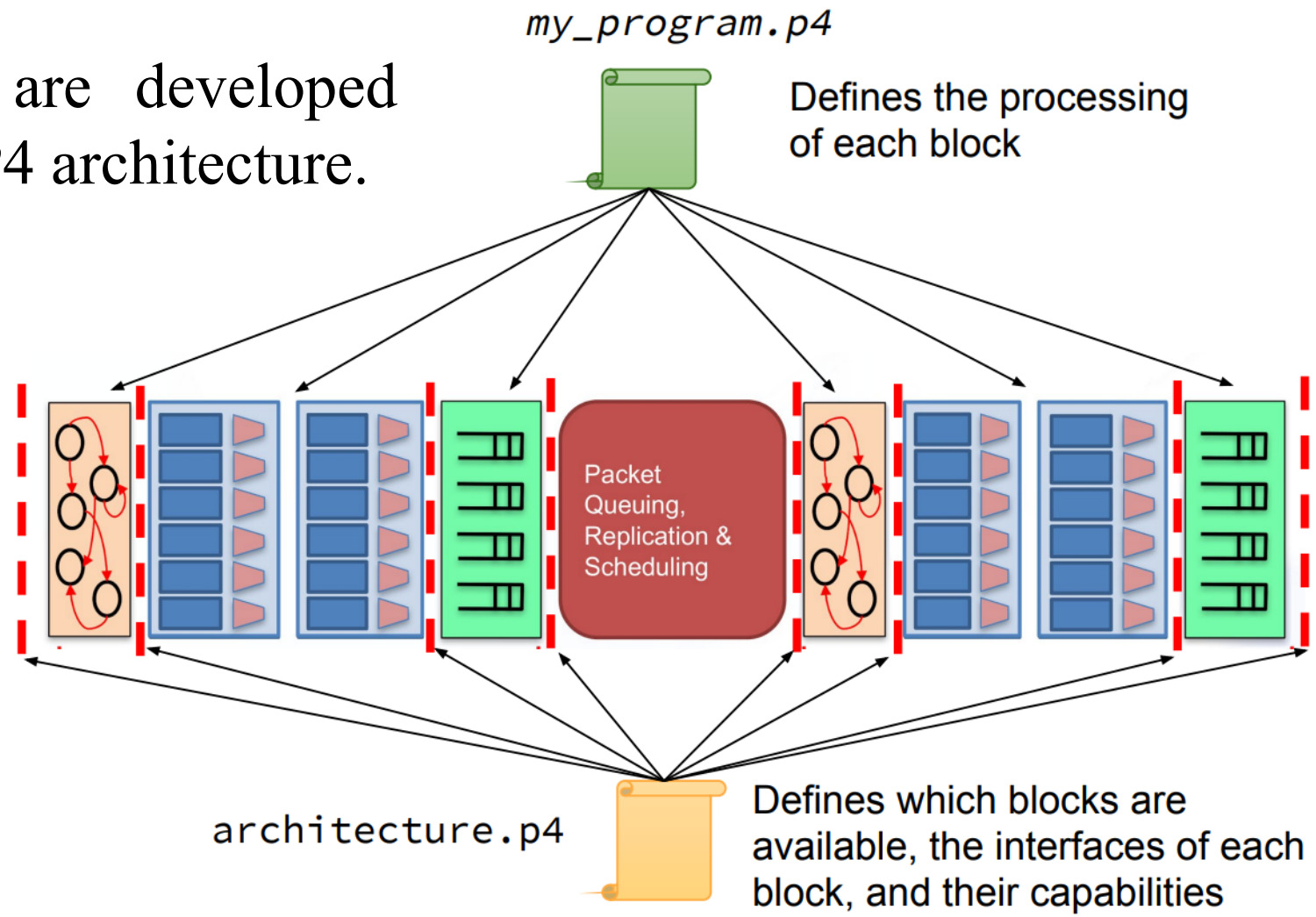
$P4_{16}$ Approach

- **Target:** A packet-processing system capable of executing a P4 program

- **Architecture**: A set of P4-programmable components and the data plane interfaces between them.
  - An intermediate layer between the core P4 language and the targets which insulate programmers from the hardware details
  - Represents the capabilities and the logical view of a target's P4 processing pipeline by identifying the P4-programmable blocks (e.g., parser, ingress control flow, egress control flow, deparser, etc.) and their data plane interfaces.
- Each manufacturer must provide both a P4 compiler as well as an accompanying architecture definition for their target.

$P4_{16}$ Approach

- P4 programs are developed for a specific P4 architecture.



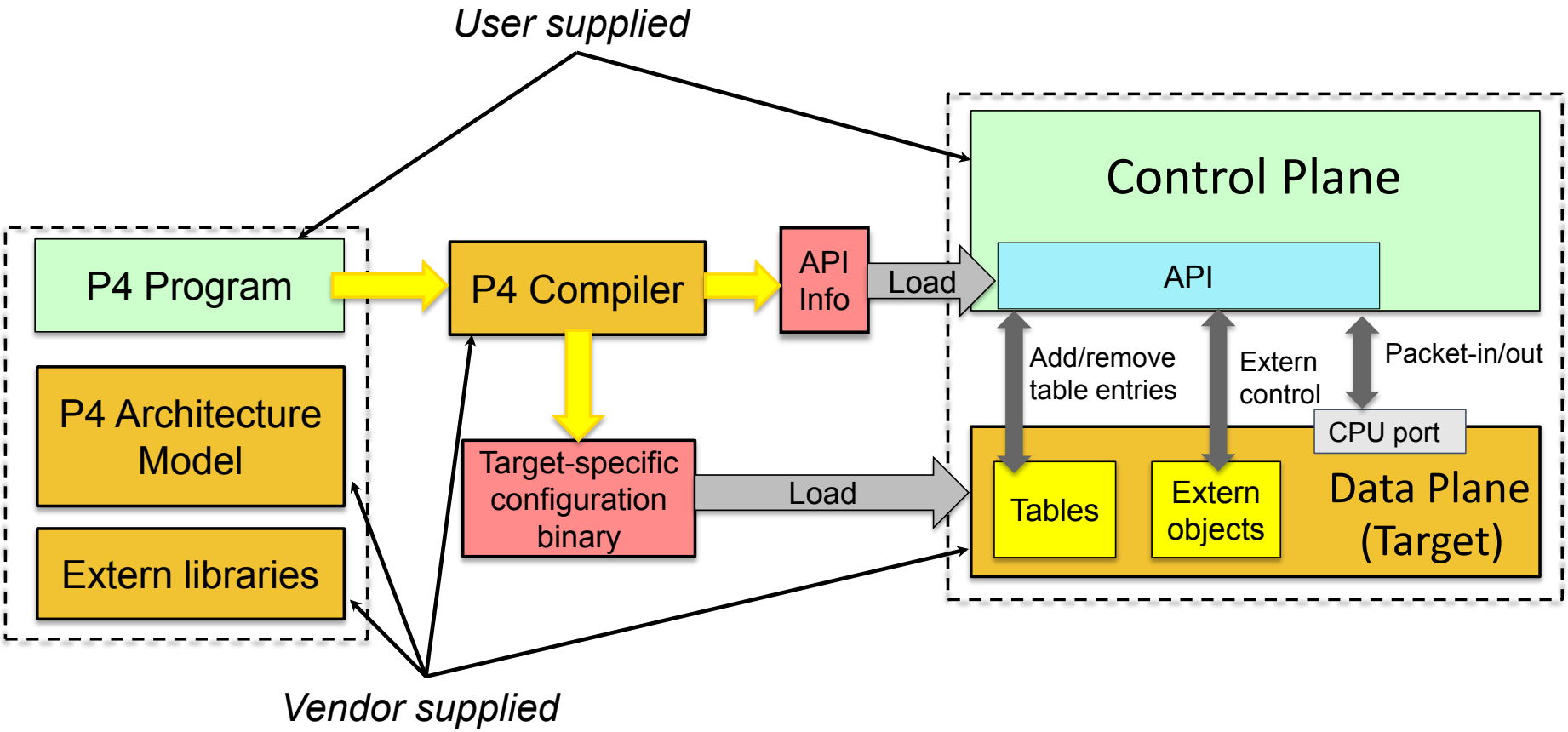
$P4_{16}$ Approach

- In general, P4 programs are not expected to be portable across different architectures.
- However, P4 programs written for a given architecture should be portable across all targets that faithfully implement the corresponding model, provided there are sufficient resources.
- The manufacturers of P4 targets provide P4 compilers that compile architecture-specific P4 programs into target-specific configuration binaries.

$P4_{16}$ Approach

- **Extern:** additional fixed-function services that are exposed by the target and can be invoked by a P4 programmer.
  - Functionalities that are not part of the P4 language core.
  - e.g. checksum, hash computation units, random number generators, packet and byte counters, meters, registers, ciphers to encrypt/decrypt the packet payload, etc.
  - The implementation of an extern unit is not specified in P4, but its interface is specified.
  - A P4 program can include a target-provided extern library, and make use of its extern objects by instantiating them.

$P4_{16}$ Workflow



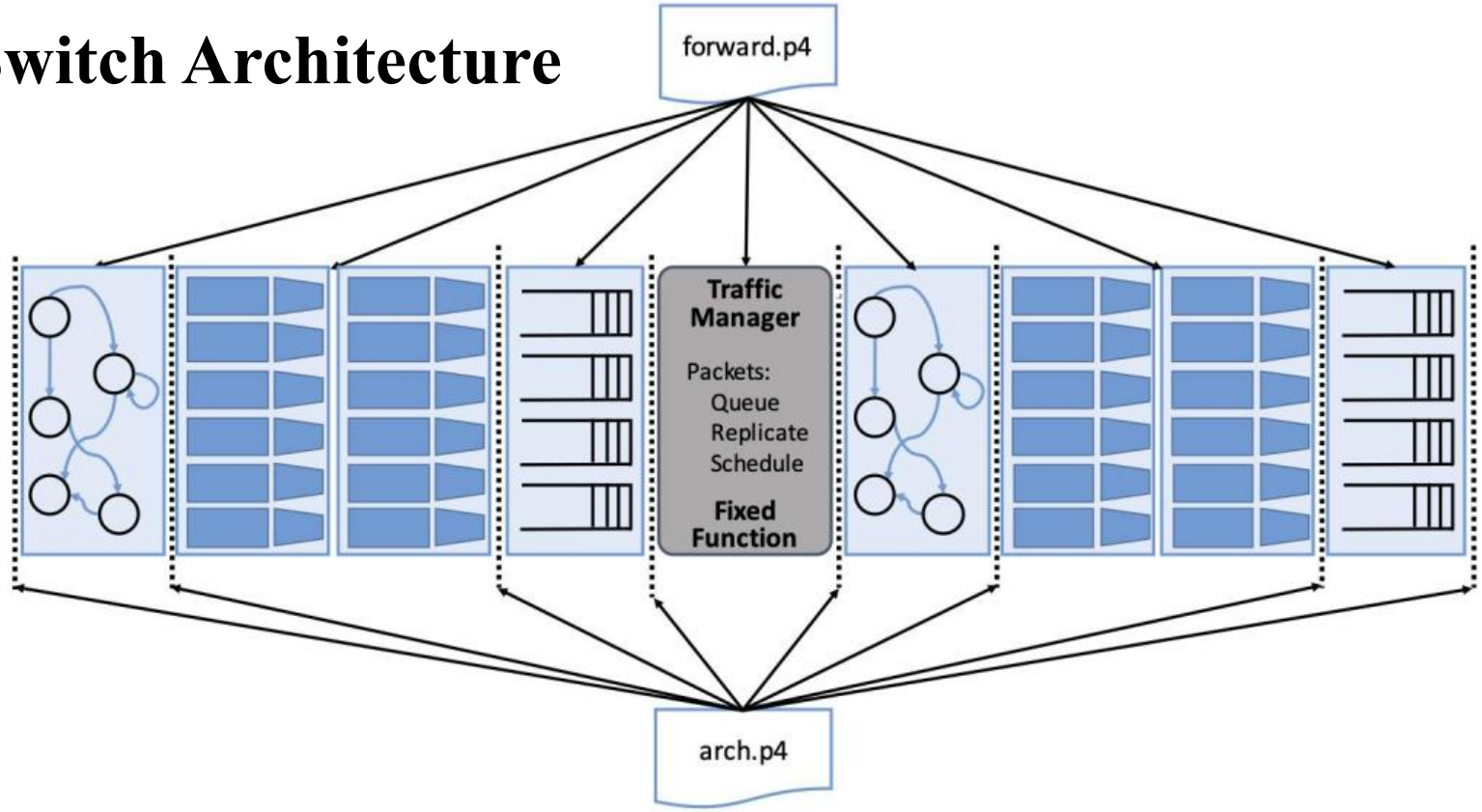
Well-known Architectures

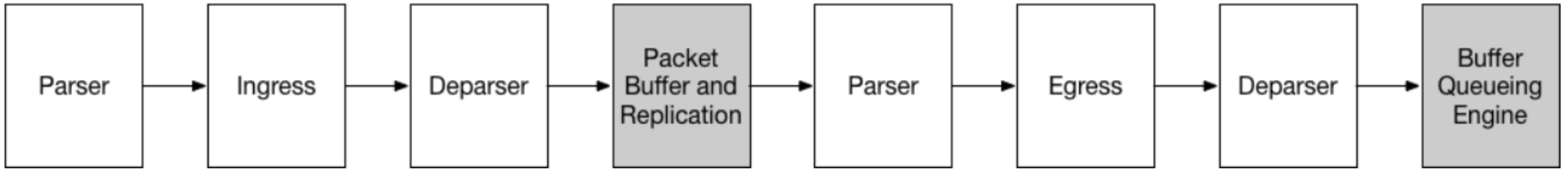
❖ **PSA: Portable Switch Architecture**

- Developed by P4 community
- Describes common capabilities of network switches
- 6 programmable P4 blocks + 2 fixed-function blocks in the pipeline
- Externs such as checksum, Counter, Meter, Hash, Random, Register, etc.

Well-known Architectures

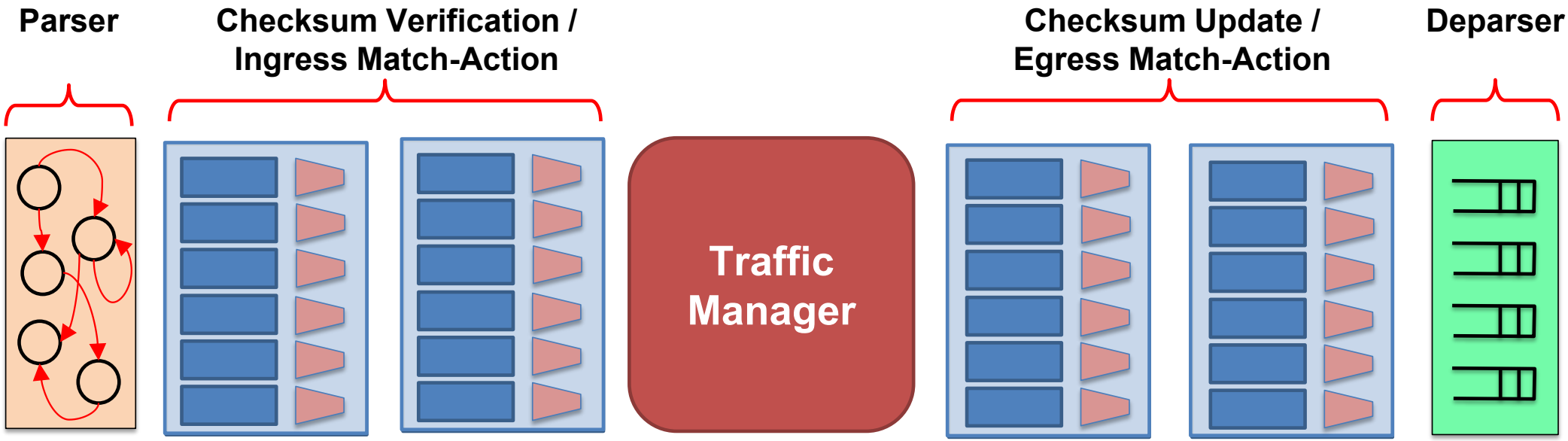
❖ **PSA: Portable Switch Architecture**





Well-known Architectures

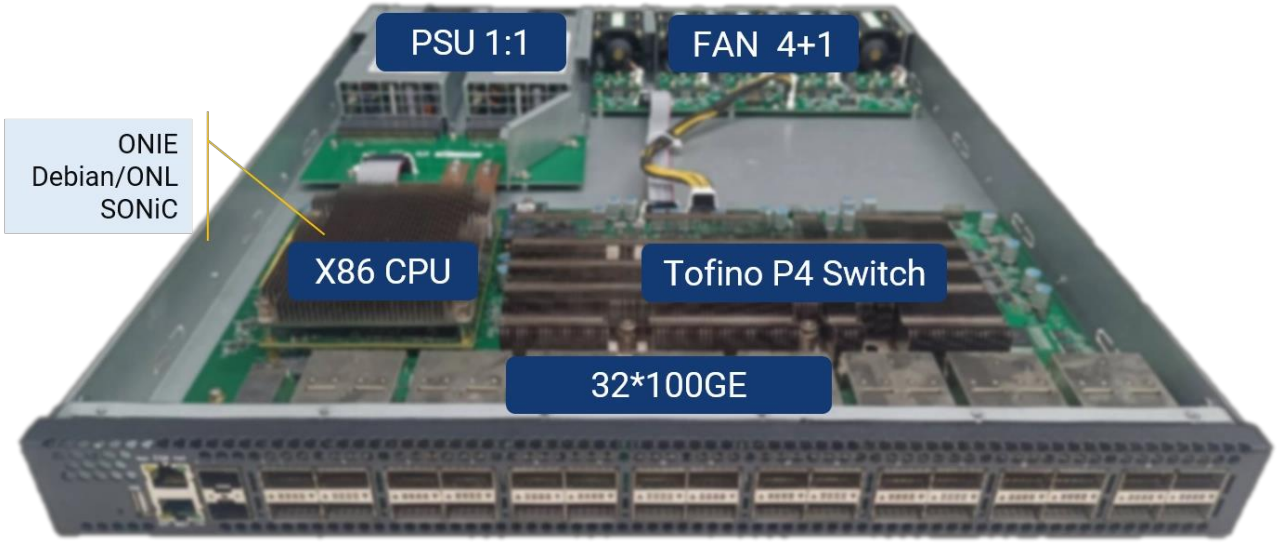
- **V1Model**
  - Implemented on top of Bmv2's *simple_switch* target
  - Bmv2 is a software switch



Well-known Architectures

❖ Tofino Native Architecture (TNA)

- The architecture model defined by *Barefoot* for their family of programmable switching chips.

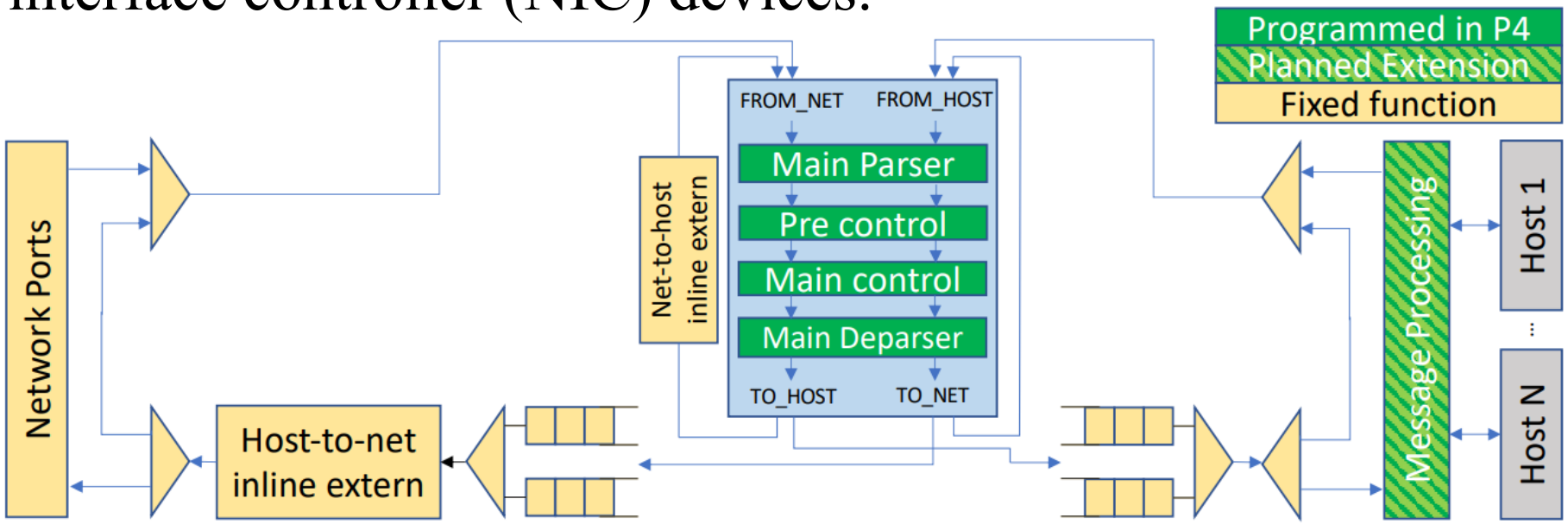


6.5 Tb/s Switching Chip
65 x 100GE (or 260 x 25GE)

Well-known Architectures

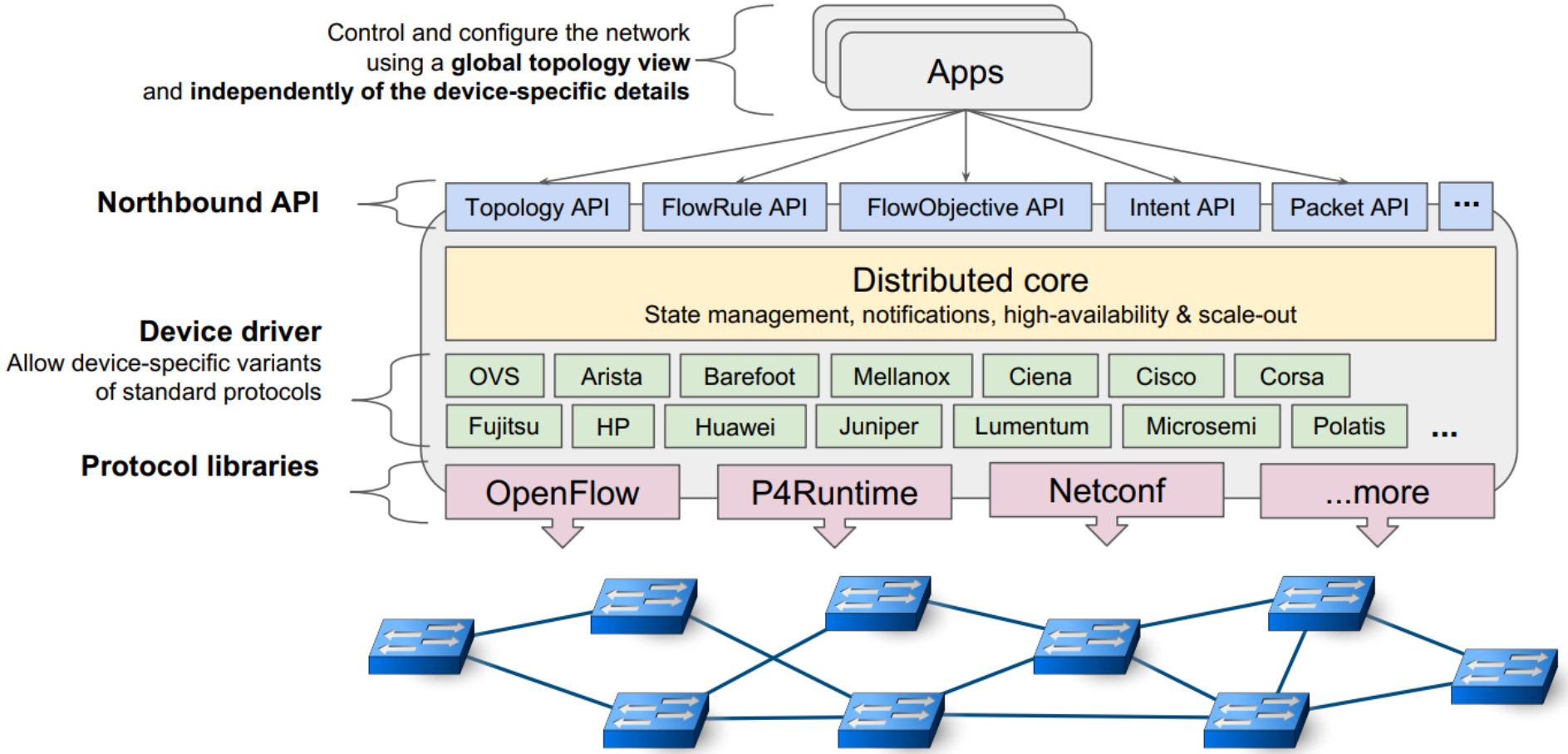
❖ Portable NIC Architecture (PNA)

- ➢ Is being developed by P4 community
- ➢ Defines the structure and common capabilities for network interface controller (NIC) devices.



Example of P4-programmable NIC: Netronome Agilio FX SmartNICs

P4 and P4runtime Support in ONOS



ONOS users can bring their own P4 program and control custom/new protocols.
ONOS apps can control any P4 pipeline (app portability across many P4 pipelines)

Examples of Applications

- Layer 4 load balancer (SilkRoad)

"Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics", SIGCOMM 2017

- Connection-persistent load balancer (Cheetah)

"A High-SpeedLoad-Balancer Design with Guaranteed Per-Connection-Consistency", NSDI 2020

- Low latency congestion control (NDP)

"Re-architecting datacenter networks and stacks for low latency and high performance", SIGCOMM 2017

- Fast In-Network cache for key-value stores (NetCache)

"Netcache: Balancing key-value stores with fast in-network caching", SOSP 2017

- In-band network telemetry (INT)

"In-band network telemetry via programmable dataplanes", SIGCOMM 2015

- Intrusion detection (P4ID)

"P4ID: P4 enhanced intrusion detection", IEEE NFV-SDN 2019