Advanced Network Programming

Programmable Data Plane

Lin Wang

Fall 2020, Period 1

Part of the content is adapted from Laurent Vanbever

Part 2: network infrastructure

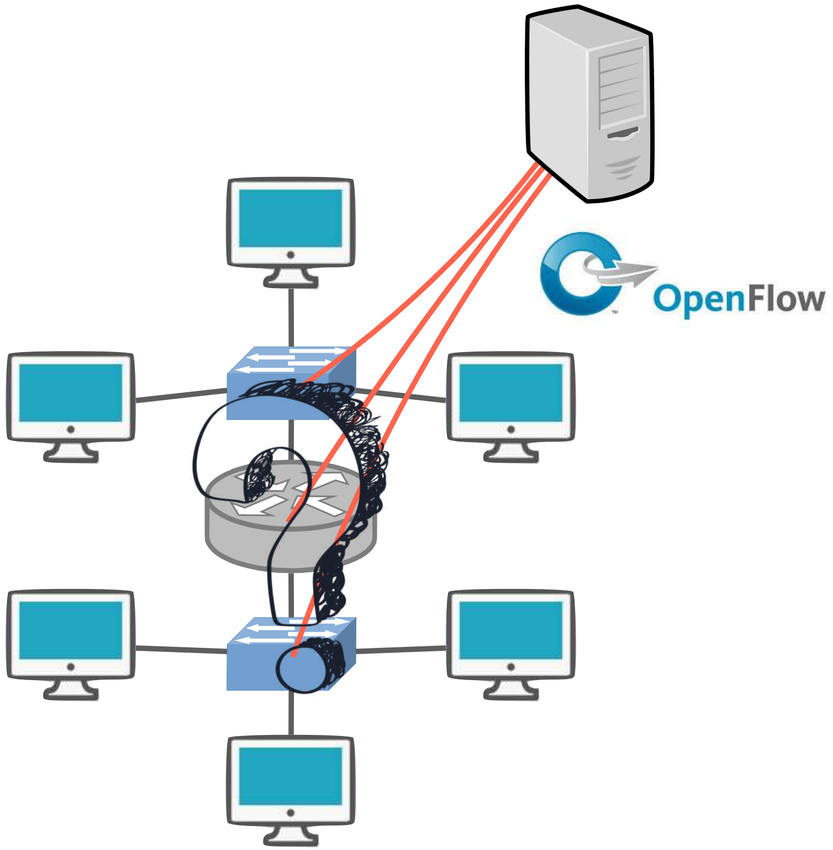
Lecture 7: Network forwarding and routing

Lecture 8: Software defined networking

Lecture 9: Programmable data plane

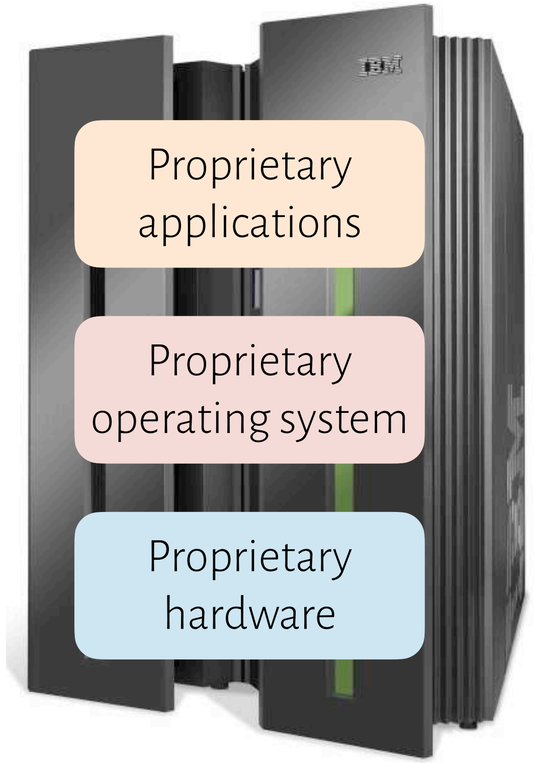
Lecture 10: Cloud networking

Lecture 11: Beyond networking

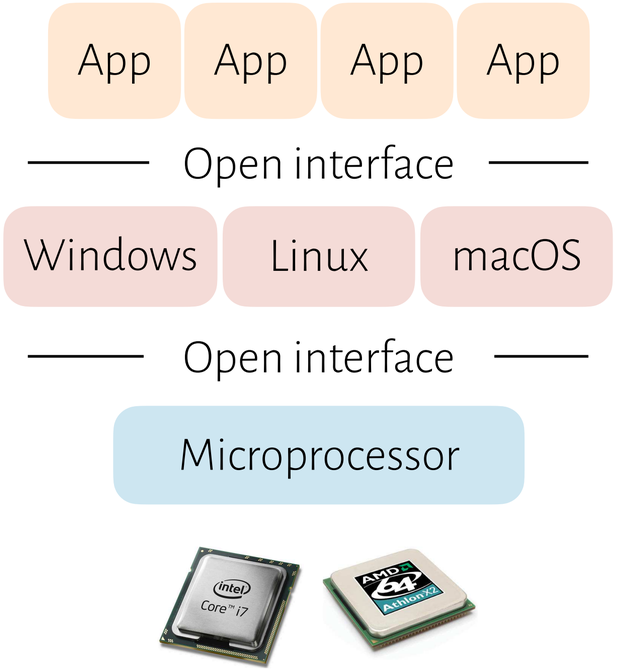


Part of the slides are adapted from materials by Laurent Vanbever (ETH)

Evolution of the computer industry

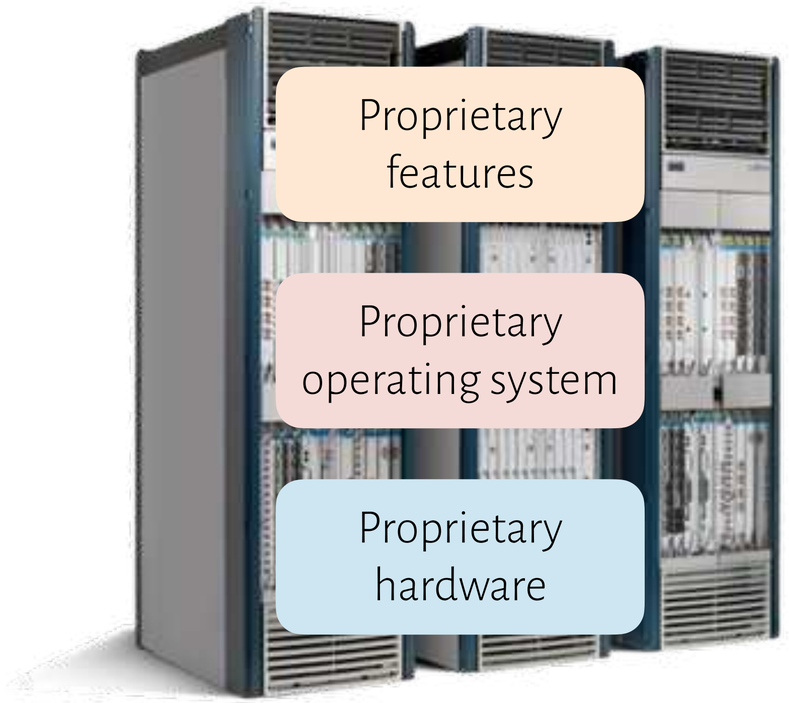


Disaggregation



The computing industry has been evolving from proprietary hardware/software towards more **general-purpose** hardware/software with **open standards/interfaces**.

Evolution of networking industry



Disaggregation

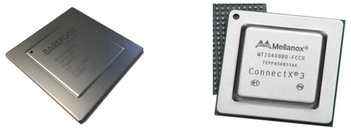
App | App | App | App

Open interface

Floodlight | Ryu | ONOS

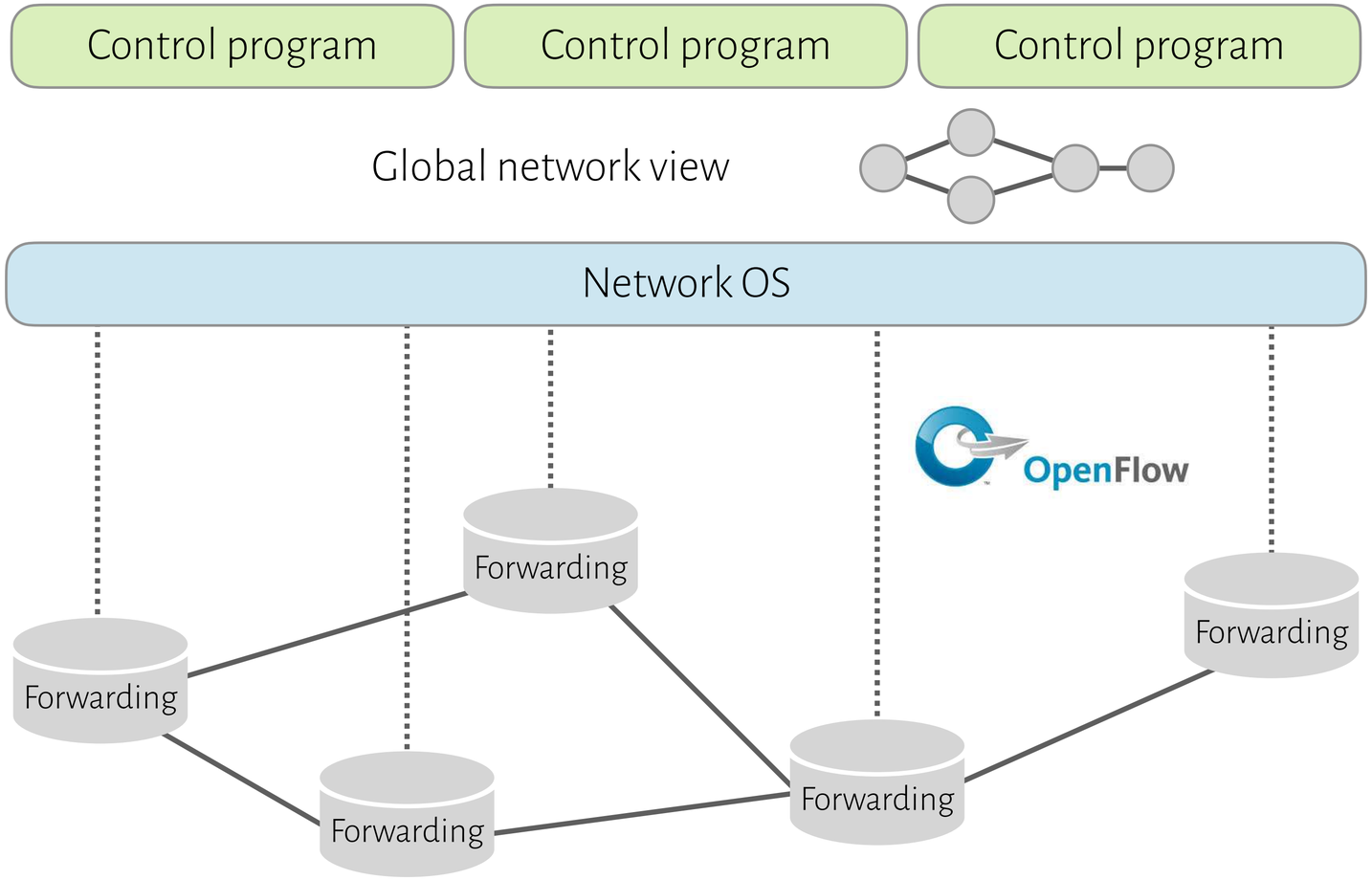
Open interface

Merchant switch chips

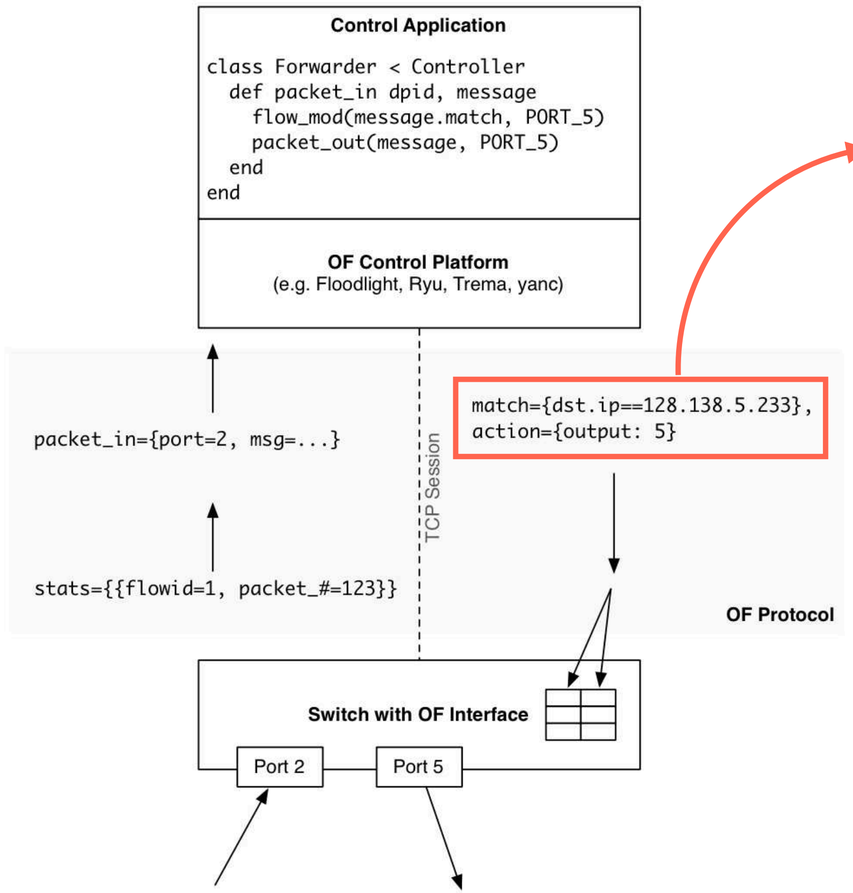


The networking industry has also been evolving from proprietary hardware/software towards more **general-purpose** hardware/software with **open standards/interfaces**.

Recap: software define networking



A deep dive into OpenFlow



OpenFlow is designed around the **match+action abstraction**: a set of header match fields and forwarding actions

OpenFlow v1.5: 41 match header fields

Most hardware/software switches only support limited match/action set (Ethernet, IP, TCP, MPLS) due to ASIC limitations.

Match

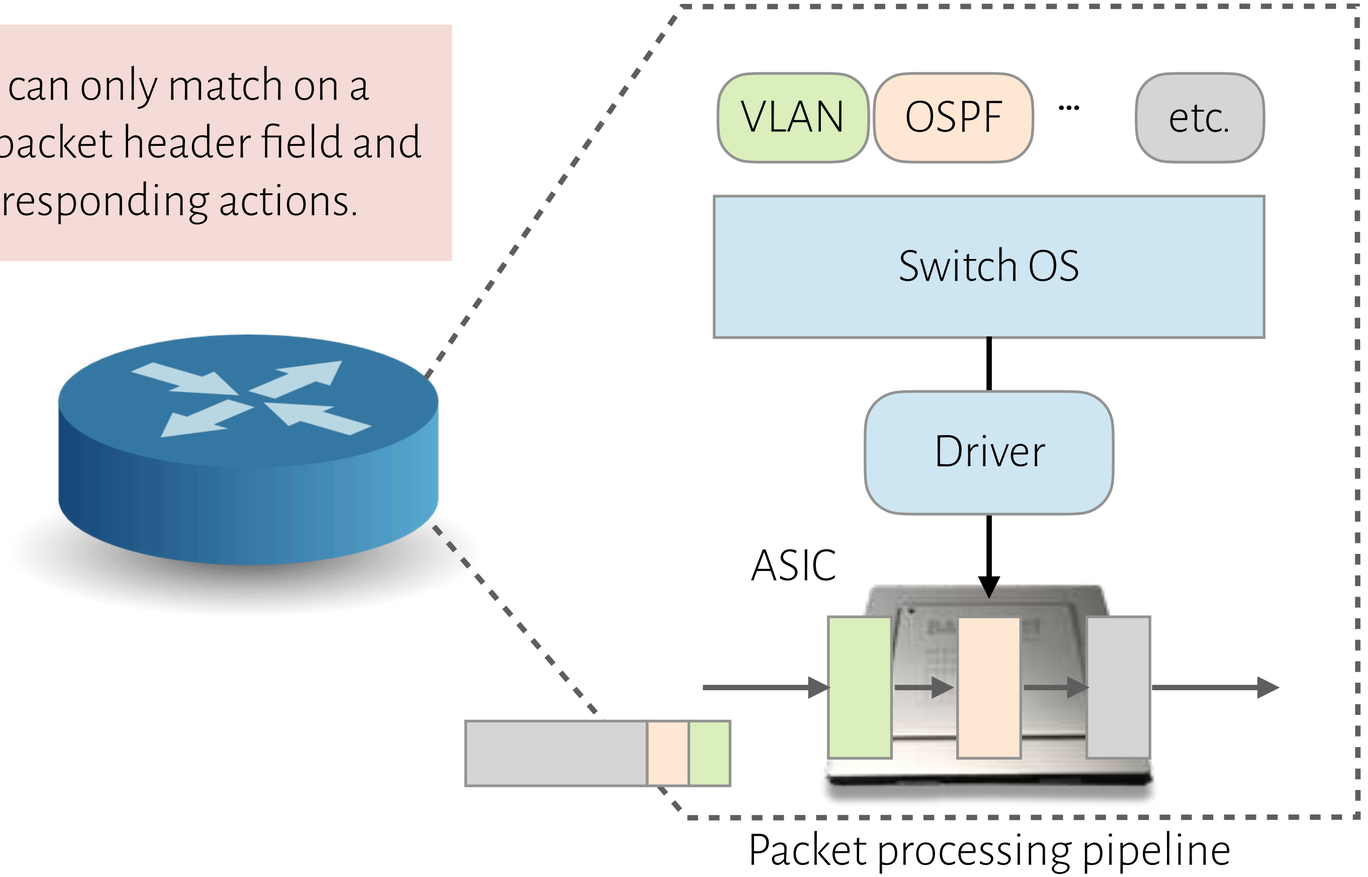
```
enum oxm_ofb_match_fields {
  OFPXMT_OFB_IN_PORT,
  OFPXMT_OFB_IN_PHY_PORT,
  OFPXMT_OFB_METADATA,
  OFPXMT_OFB_ETH_DST ,
  OFPXMT_OFB_ETH_SRC ,
  OFPXMT_OFB_ETH_TYPE ,
  OFPXMT_OFB_VLAN_VID ,
  OFPXMT_OFB_VLAN_PCP ,
  OFPXMT_OFB_IP_DSCP ,
  OFPXMT_OFB_IP_ECN ,
  OFPXMT_OFB_IP_PROTO ,
  OFPXMT_OFB_IPV4_SRC ,
  OFPXMT_OFB_IPV4_DST ,
  OFPXMT_OFB_TCP_SRC ,
  OFPXMT_OFB_TCP_DST ,
  OFPXMT_OFB_UDP_SRC ,
  OFPXMT_OFB_UDP_DST ,
  OFPXMT_OFB_SCTP_SRC ,
  OFPXMT_OFB_SCTP_DST ,
  OFPXMT_OFB_ICMPV4_TYPE ,
  OFPXMT_OFB_ICMPV4_CODE ,
  OFPXMT_OFB_ARP_OP ,
  OFPXMT_OFB_ARP_SPA ,
  OFPXMT_OFB_ARP_TPA ,
  OFPXMT_OFB_ARP_SHA ,
  OFPXMT_OFB_ARP_THA ,
  OFPXMT_OFB_IPV6_SRC ,
  OFPXMT_OFB_IPV6_DST ,
  OFPXMT_OFB_IPV6_FLABEL ,
  OFPXMT_OFB_ICMPV6_TYPE ,
  OFPXMT_OFB_ICMPV6_CODE ,
  OFPXMT_OFB_IPV6_ND_TARGET
  OFPXMT_OFB_IPV6_ND_SLL,
  OFPXMT_OFB_IPV6_ND_TLL,
  OFPXMT_OFB_MPLS_LABEL,
  OFPXMT_OFB_MPLS_TC,
  OFPXMT_OFP_MPLS_BOS,
  OFPXMT_OFB_PBB_ISID,
  OFPXMT_OFB_TUNNEL_ID,
  OFPXMT_OFB_IPV6_EXTHDR,
  OFPXMT_OFB_PBB_UCA
};
```

Action

```
enum ofp_action_type {
  OFPAT_OUTPUT,
  OFPAT_COPY_TTL_OUT,
  OFPAT_COPY_TTL_IN,
  OFPAT_SET_MPLS_TTL,
  OFPAT_DEC_MPLS_TTL,
  OFPAT_PUSH_VLAN,
  OFPAT_POP_VLAN,
  OFPAT_PUSH_MPLS,
  OFPAT_POP_MPLS,
  OFPAT_SET_QUEUE,
  OFPAT_GROUP,
  OFPAT_SET_NW_TTL,
  OFPAT_DEC_NW_TTL,
  OFPAT_SET_FIELD,
  OFPAT_PUSH_PBB,
  OFPAT_POP_PBB,
  OFPAT_EXPERIMENTER
};
```

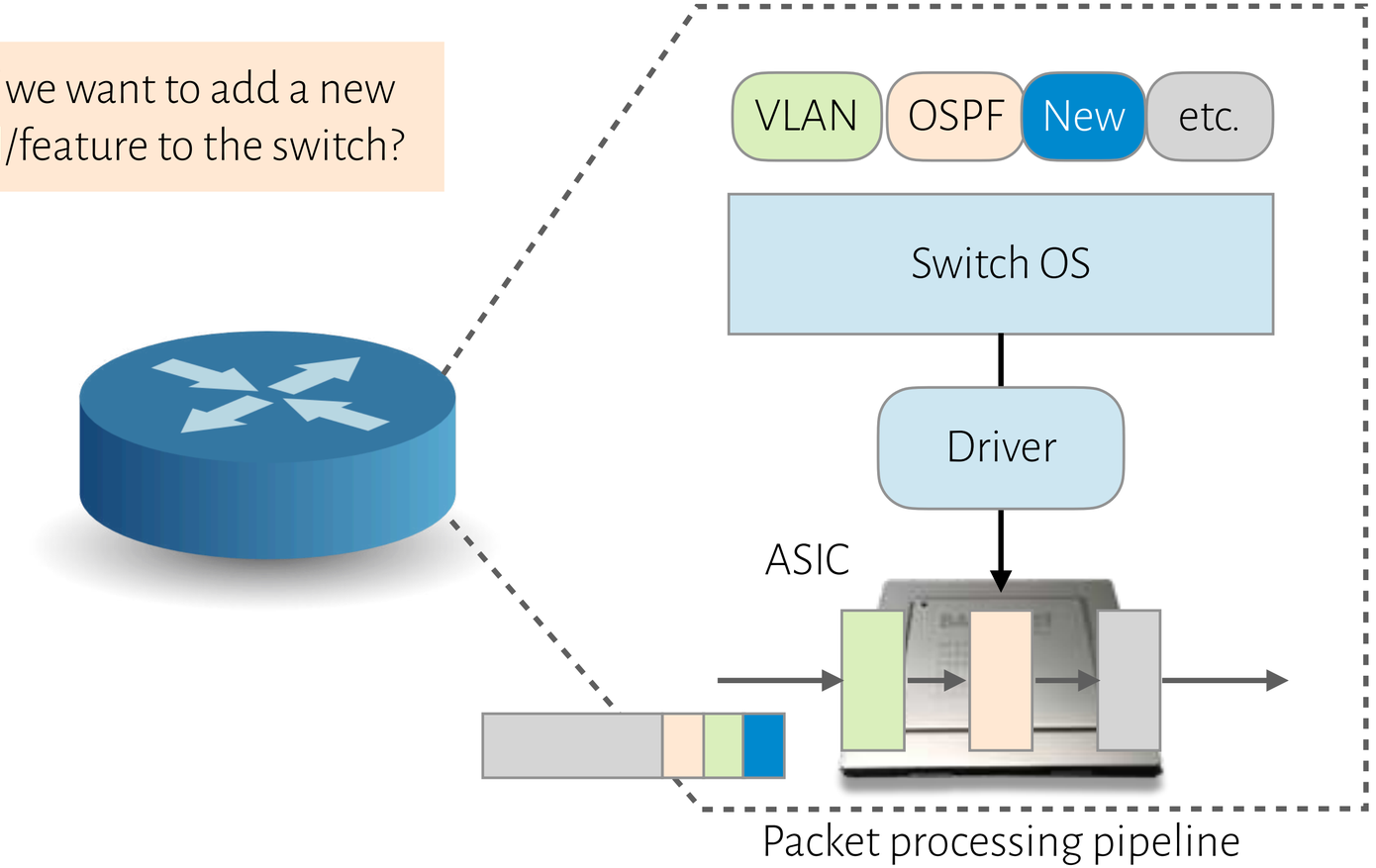
Switch architecture

A switch can only match on a supported packet header field and take corresponding actions.



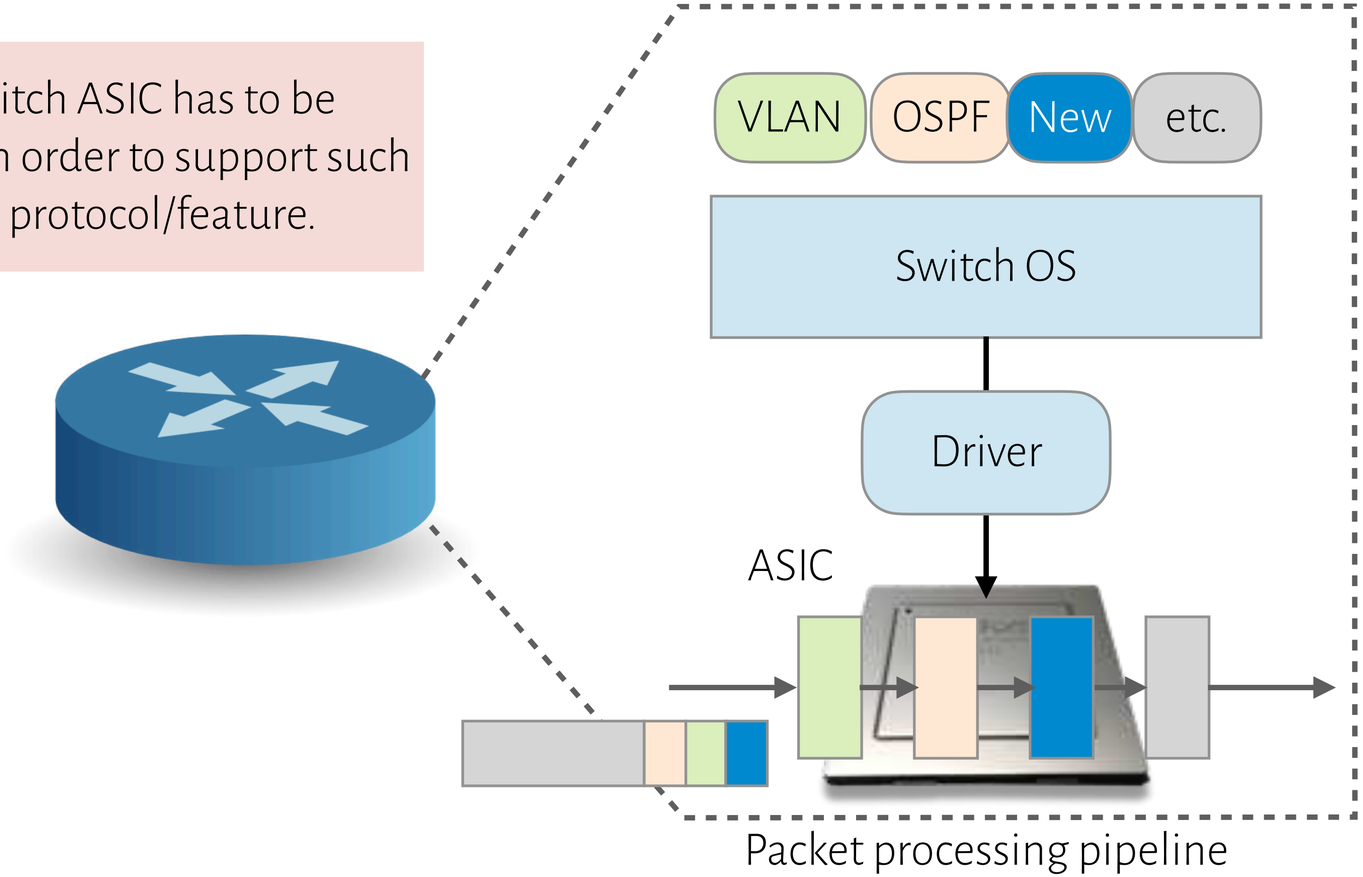
Switch architecture: adding a new protocol

What if we want to add a new protocol/feature to the switch?

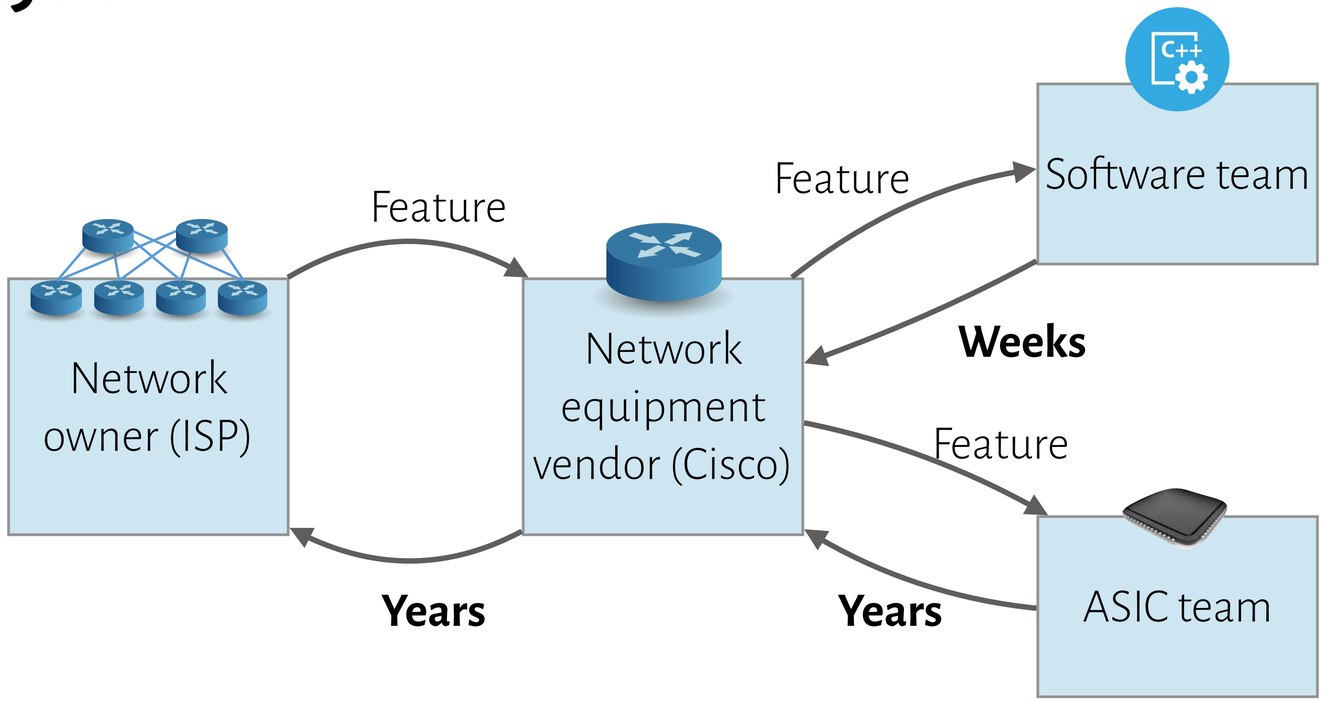


Switch architecture: required modifications

The switch ASIC has to be modified in order to support such a new protocol/feature.



Development cycle



It takes **years** for the new ASIC to be developed, fully tested, and finally deployed!! When the upgrade is available:

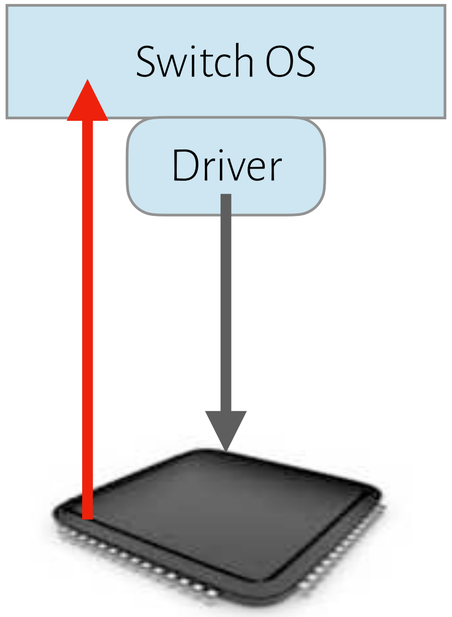
- It either **no longer solves your problem**
- You need **a fork-lift upgrade** at huge expenses

What is the root cause of all this?

The "bottom-up" mentality

The network systems are built following the bottom-up approach: all network features are centered around **the capabilities of the ASIC**.

"This is how I process packet..."



Fixed function switch

Any other ideas? If so, why you think it is better?

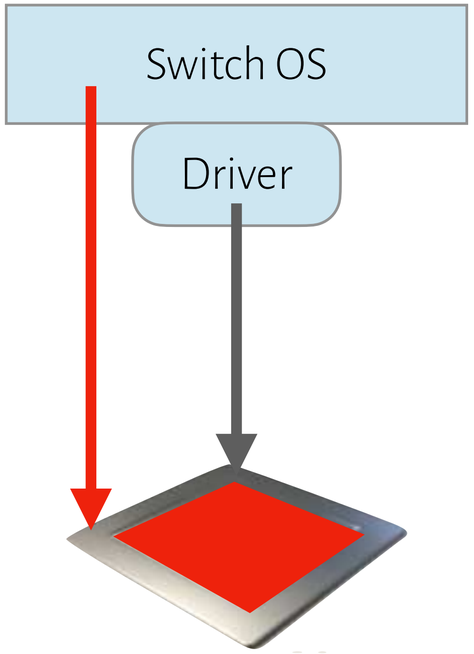
The "top-down" approach

Make the ASIC **programmable**, and let your features to tell the ASIC what to support!

```
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);
}
```

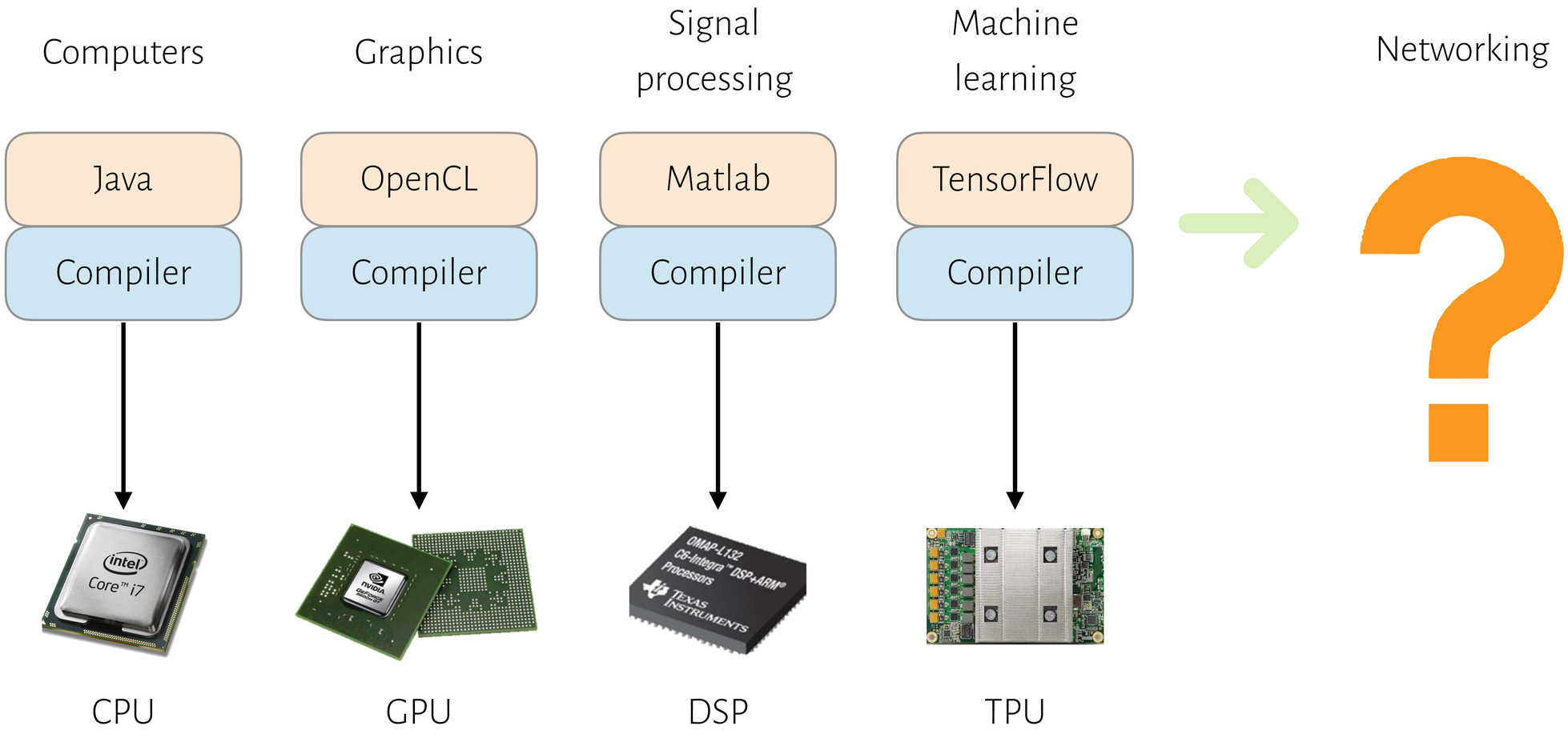
"This is precisely how you must process packets…"



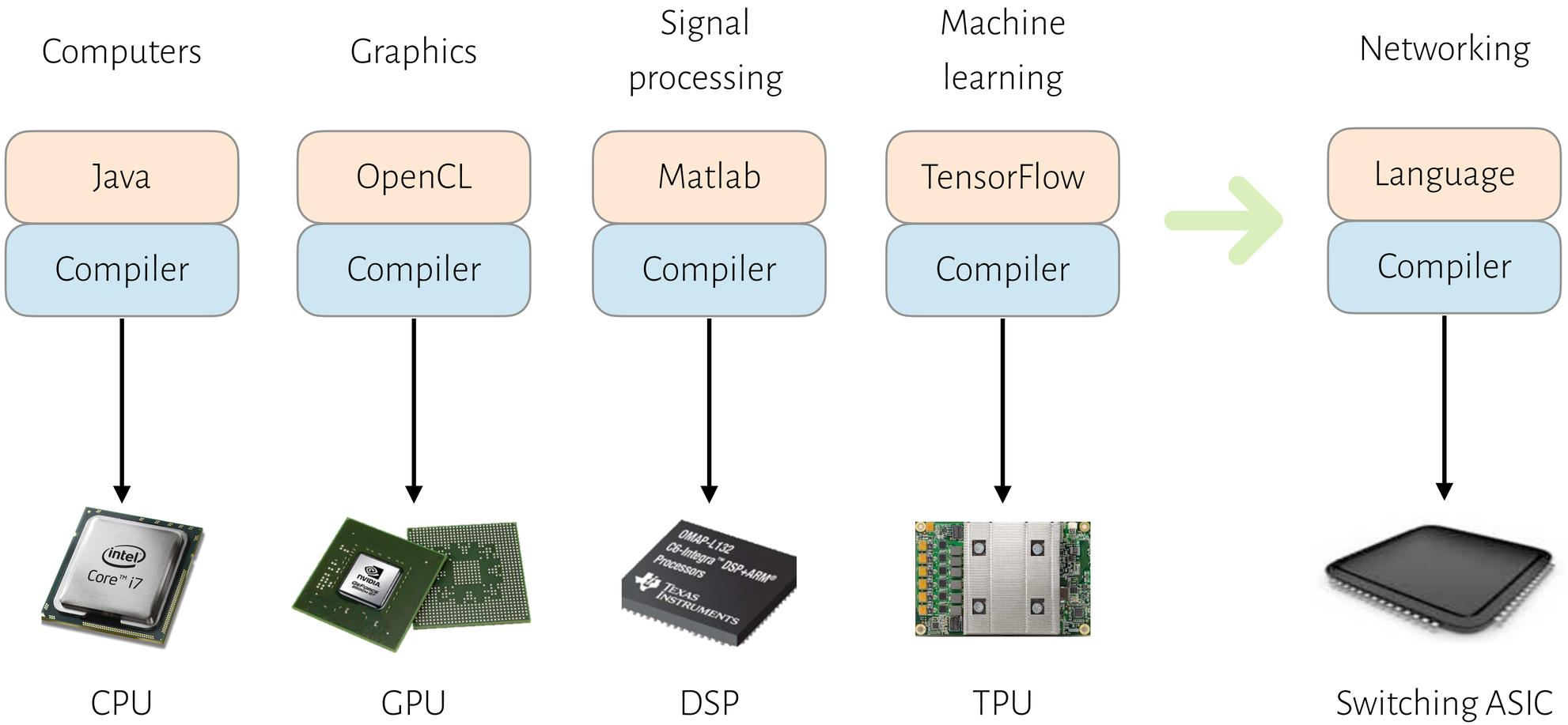
Customizable switching ASIC

How to support programmability?

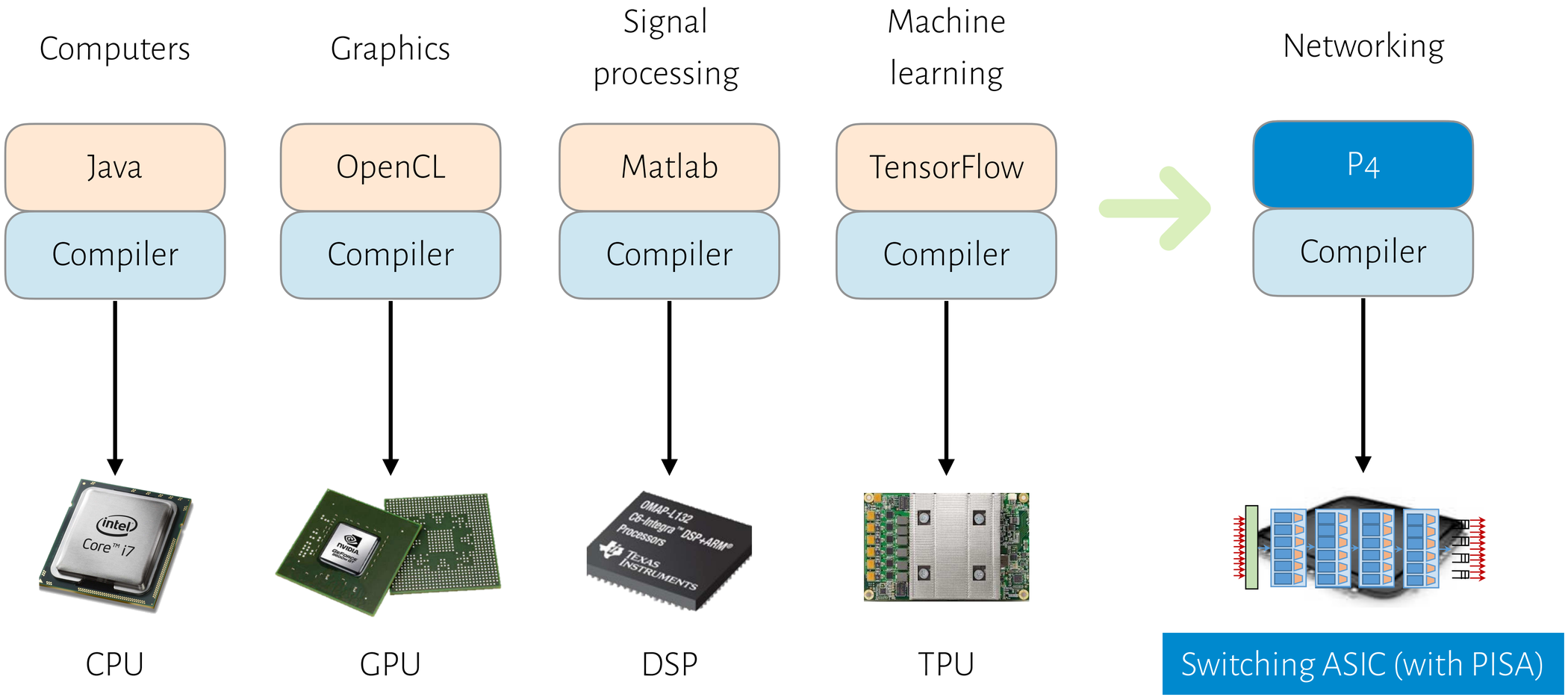
Domain-specific processors



Domain-specific processors



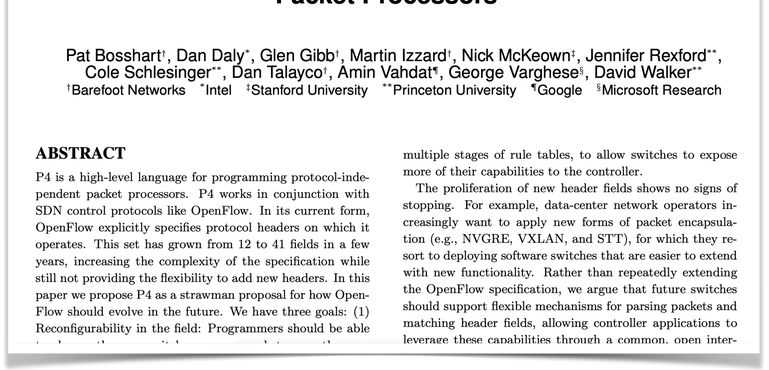
Programmable switch ASIC abstraction: PISA



PISA and P4

PISA: **p**rotocol **i**ndependent **s**witch **a**rchitecture

P4: **p**rogramming **p**rotocol-independent **p**acket **p**rocessors — a high-level language for programming protocol-independent packet processors



P4: Programming Protocol-Independent Packet Processors

Pat Bosshart†, Dan Daly*, Glen Gibb†, Martin Izzard†, Nick McKeown‡, Jennifer Rexford**, Cole Schlesinger**, Dan Talayco†, Amin Vahdat¶, George Varghese§, David Walker**
†Barefoot Networks *Intel ‡Stanford University **Princeton University ¶Google §Microsoft Research

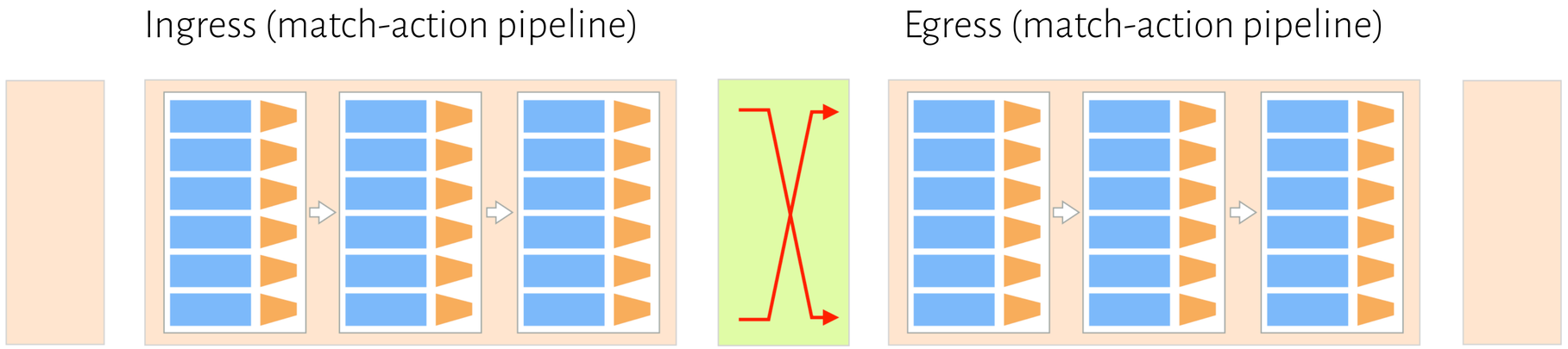
ABSTRACT

P4 is a high-level language for programming protocol-independent packet processors. P4 works in conjunction with SDN control protocols like OpenFlow. In its current form, OpenFlow explicitly specifies protocol headers on which it operates. This set has grown from 12 to 41 fields in a few years, increasing the complexity of the specification while still not providing the flexibility to add new headers. In this paper we propose P4 as a strawman proposal for how OpenFlow should evolve in the future. We have three goals: (1) Reconfigurability in the field: Programmers should be able

multiple stages of rule tables, to allow switches to expose more of their capabilities to the controller.

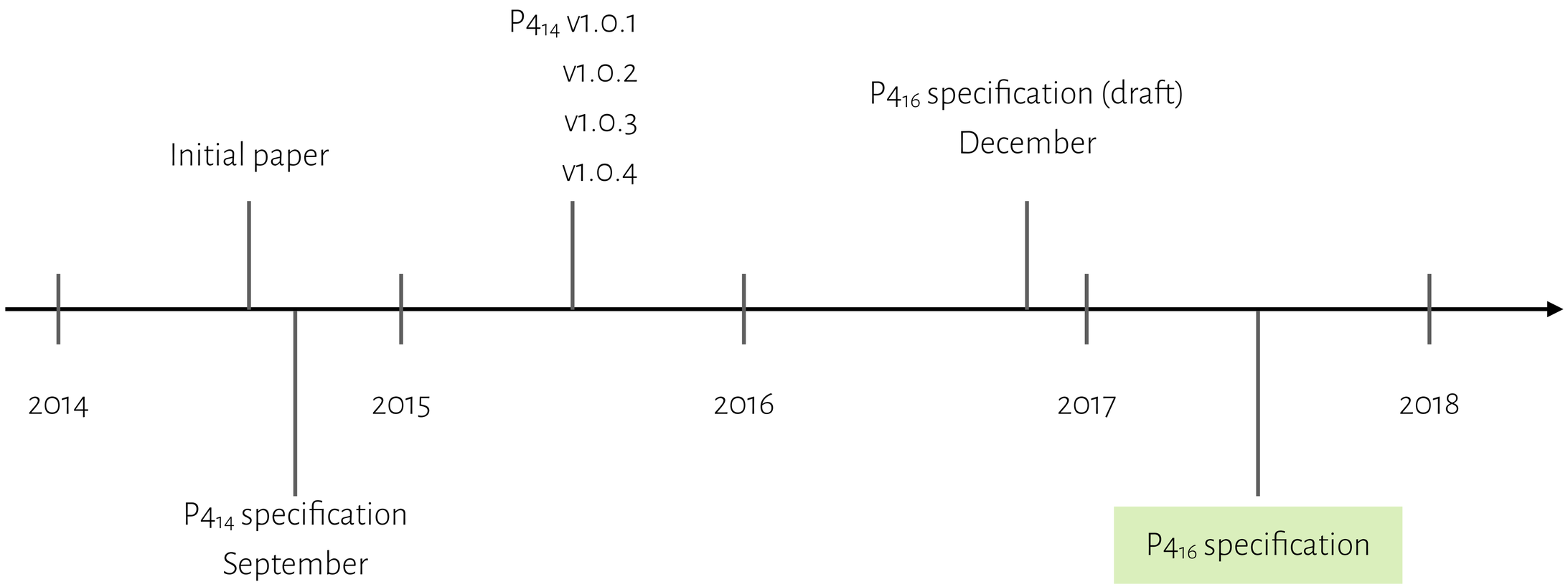
The proliferation of new header fields shows no signs of stopping. For example, data-center network operators increasingly want to apply new forms of packet encapsulation (e.g., NVGRE, VXLAN, and STT), for which they resort to deploying software switches that are easier to extend with new functionality. Rather than repeatedly extending the OpenFlow specification, we argue that future switches should support flexible mechanisms for parsing packets and matching header fields, allowing controller applications to leverage these capabilities through a common, open inter-

ACM SIGCOMM CCR 2014



https://p4.org

P4 development



P4$_{16}$ introduces the concept of an architecture

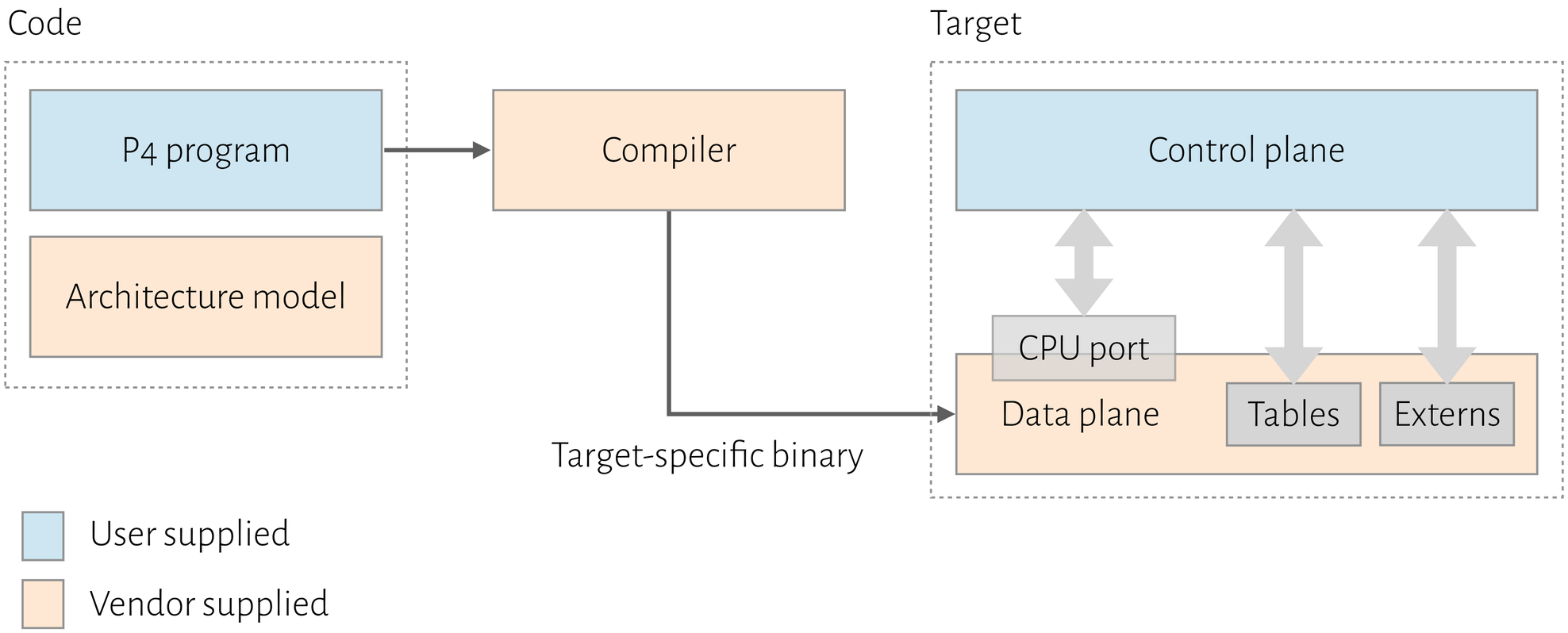
P4 target

A model of a specific hardware implementation

P4 architecture

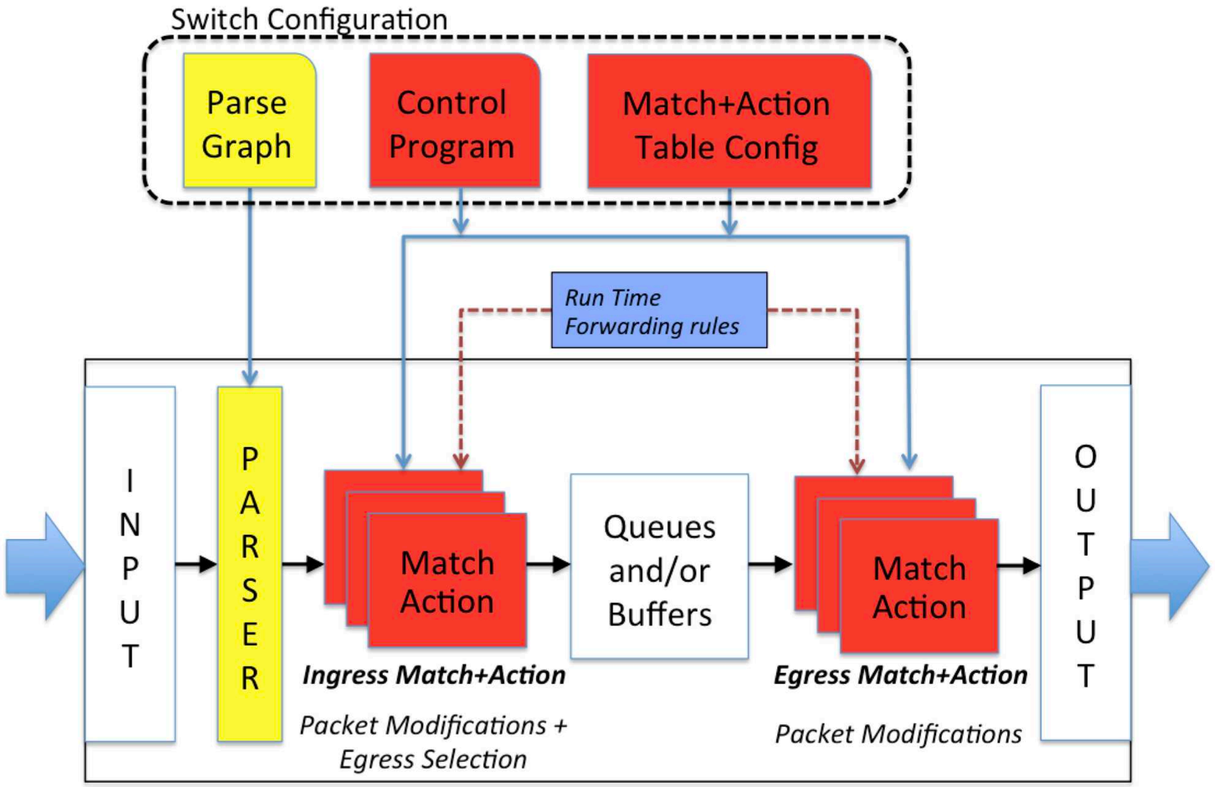
An API to program a target

Programming a P4 target



A simple $P4_{16}$ switch architecture: v1model

Roughly equivalent to "PISA"



https://p4.org/p4-spec/p4-14/v1.0.4/tex/p4.pdf

v1model architecture

Defines the **metadata** it supports, including both *standard* and *intrinsic* ones

```
struct standard_metadata_t {
    bit<9> ingress_port;
    bit<9> egress_spec;
    bit<9> egress_port;
    bit<32> clone_spec;
    bit<32> instance_type;
    bit<1> drop;
    bit<16> recirculate_port;
    bit<32> packet_length;
    bit<32> enq_timestamp;
    bit<19> enq_qdepth;
    bit<32> deq_timedelta;
    bit<19> deq_qdepth;
    error parser_error;
```

Standard

```
    bit<48> ingress_global_timestamp;
    bit<48> egress_global_timestamp;
    bit<32> lf_field_list;
    bit<16> mcast_grp;
    bit<32> resubmit_flag;
    bit<16> egress_rid;
    bit<1> checksum_error;
    bit<32> recirculate_flag;
}
```

Intrinsic

Architecture-specific constructs

Each architecture defines a list of "**externs**"

- Blackbox functions whose interface is know

Most targets contain specialised components, which cannot be expressed in P4

On the other hand, $P4_{16}$ aims to be **target-independent**

- $P4_{14}$ has almost 1/3 of the constructs target-dependent: not portable to different targets

v1model architecture-specific externs

```
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);
extern void hash<O, T, D, M>(out O result,
    in HashAlgorithm algo, in T base, in D data, in M max);
extern void update_checksum<T, O>(in bool condition,
    in T data, inout O checksum, HashAlgorithm algo);
```

https://github.com/p4lang/p4c/blob/master/p4include/v1model.p4

Questions

P4 language overview

```
#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 {...}
}
```

Parse packet headers

```
control MyIngress(...) {
    action ipv4_forward(...) {...}
    table ipv4_lpm {...}
    apply {
        if (...) {...}
    }
}
```

Control flow to modify packets

```
control MyDeparser(...) {...}
```

Assemble modified packet

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

“main()”

P4 language basics: data types

$P4_{16}$ is a **statically-typed** language with base types and operators to derive composed ones

Base types examples:

bool	Boolean value
bit<W>	Bit-string of width W
int<W>	Signed integer of width W
varbit<W>	Bit-string of dynamic length <= W
match_kind	Describes ways to match table keys
error	Used to signal errors
void	No values, used in few restricted circumstances
float ✗	Not supported
string ✗	Not supported

P4 language basics: composed data types

Header

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

Header stack

```
header Mpls_h {
    bit<20> label;
    bit<3> tc;
    bit bos;
    bit<8> ttl;
}

Mpls_h[10] mpls;
```

Array of up to 10 MPLS headers

Header union

```
header_union Ip_h {
    IPv4_h v4;
    IPv6_h v6;
}
```

Either IPv4 or IPv6 header is present

A successful extract() sets to true the validity bit of the extracted header

Parsing a packet using extract() fills in the fields of the header from a network packet

P4 language basics: composed data types

Struct: unordered collection of named members

```
struct standard_metadata_t {
    bit<9> ingress_port;
    bit<9> egress_spec;
    bit<9> egress_port;
    …
}
```

Tuple: unordered collection of unnamed members

```
tuple<bit<32>, bool> x;
x = {10, false}
```

Other data types:

- enum: `enum Priority {High, Low}`
- Type specification: `typedef bit<48> macAddr_t;`
- extern, parser, control, package…

https://p4.org/p4-spec/docs/P4-16-v1.0.0-spec.html

P4 language basics: operations

P4 operations are similar to C operations and vary depending on the types (unsigned/signed integers,...)

- Arithmetic operations: **+**, **-**, *****
- Logical operations: **~**, **&**, **|**, **^**, **>>**, **<<**
- Non-standard operations: **[m:l]** bit slicing, **++** bit concatenation
- No division and modulo: can be approximated

https://p4.org/p4-spec/docs/P4-16-v1.0.0-spec.html

P4 language basics: variables and constants

Constants, variable declarations and instantiations are pretty much the same as in C too

Variable

```
bit<8> x = 123;

typedef bit<8> MyType;
MyType x;
x = 123;
```

Constant

```
const bit<8> x = 123;

typedef bit<8> MyType;
const MyType x = 123;
```

Important

Variables cannot be used to maintain state between different network packets

Instead, we can only use **two stateful constructs, i.e., tables and extern objects,** to maintain state

P4 language basics: statements

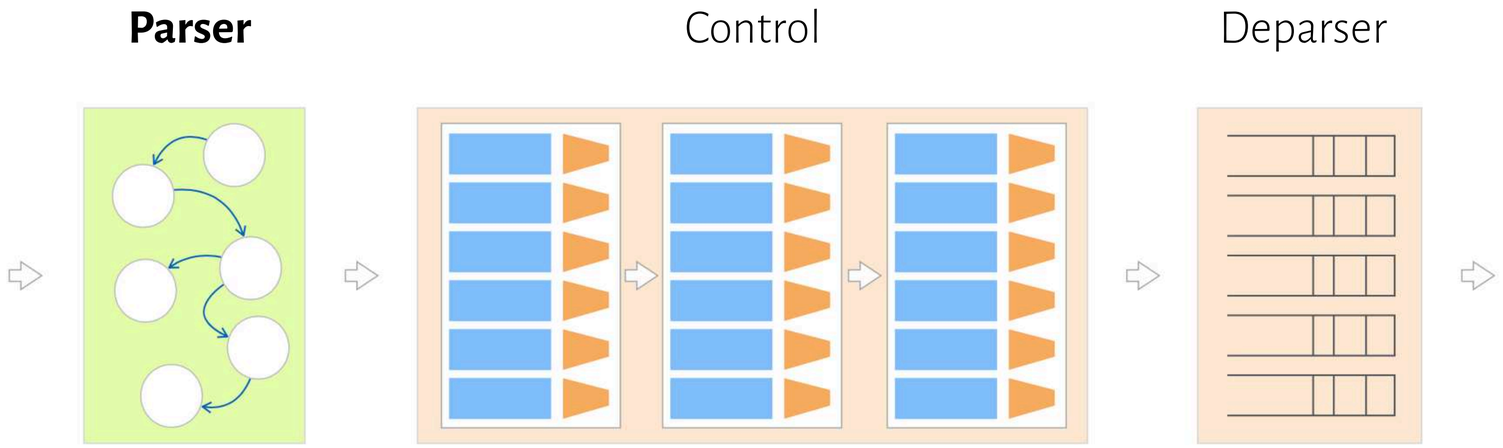
P4 statements are pretty classical too

- Some restrictions may apply depending on the statement location

return	Terminates the execution of the action of control containing it	
exit	Terminates the execution of all the blocks currently executing	
Conditions	`if (x==123) {…} else {…}`	Not in parser
Switch	`switch (t.apply().action_run) { action1: {…} action2: {…} }`	Only in control blocks

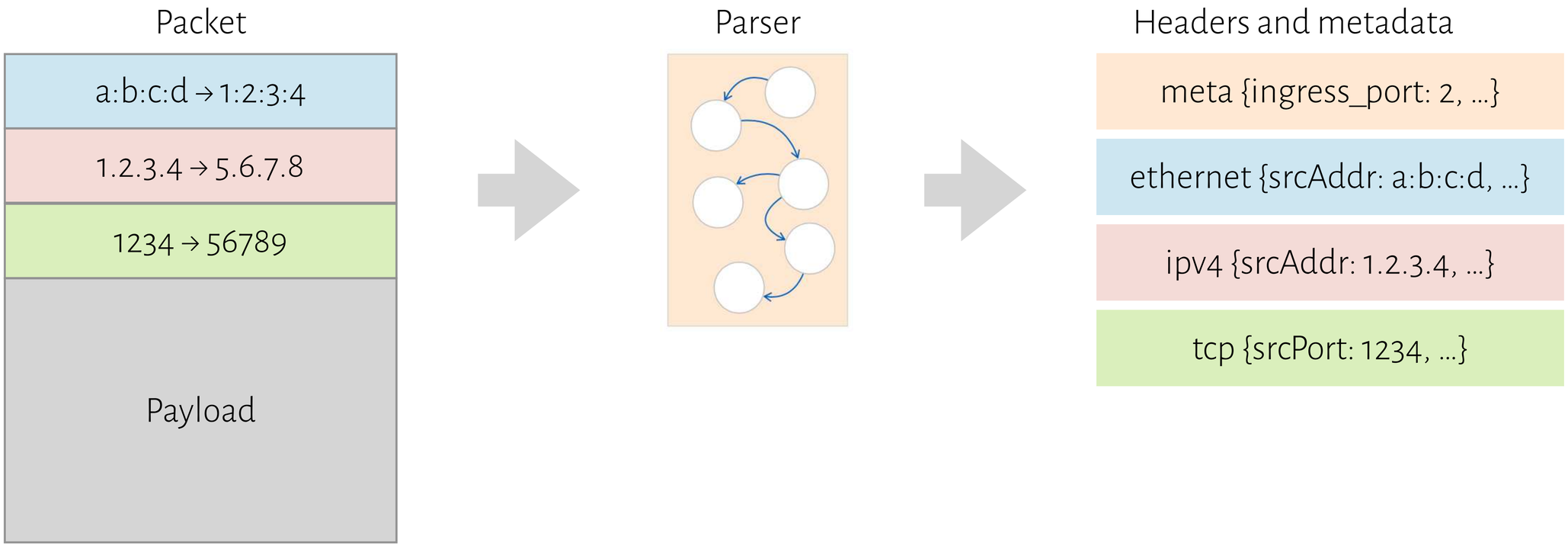
Questions

P4 processing overview

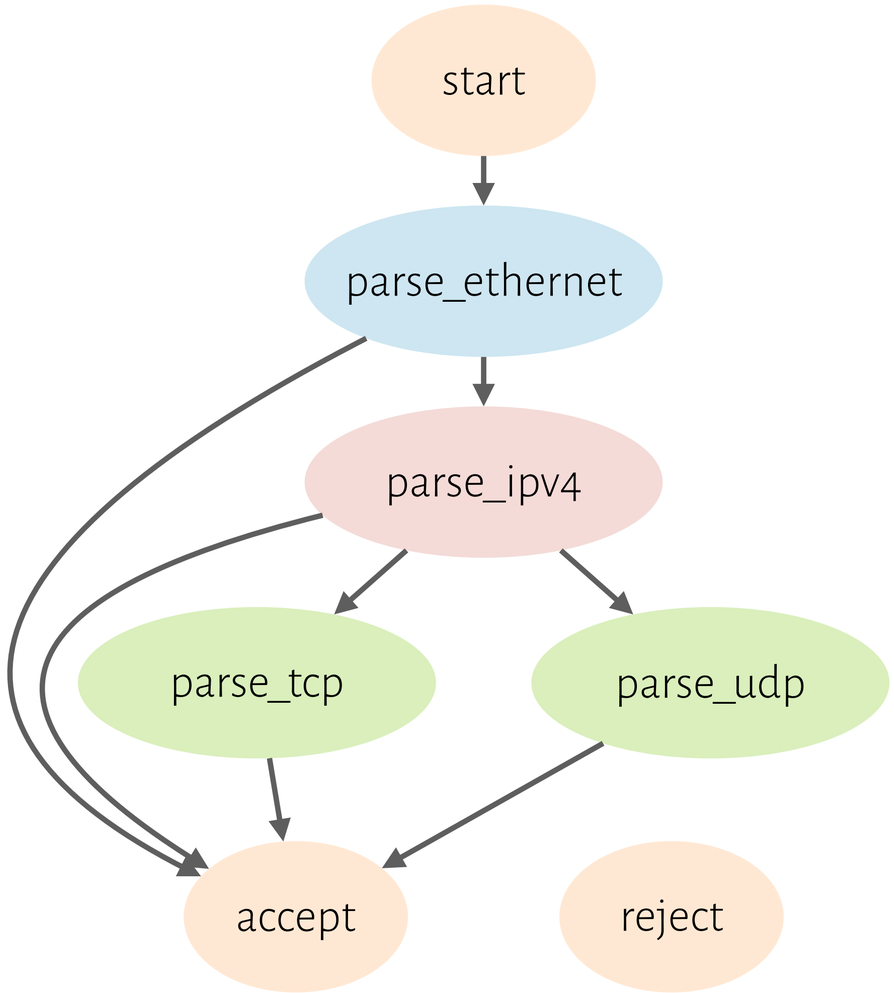


P4 parser

The parser uses a **state machine** to map packets into headers and metadata



P4 parser: example



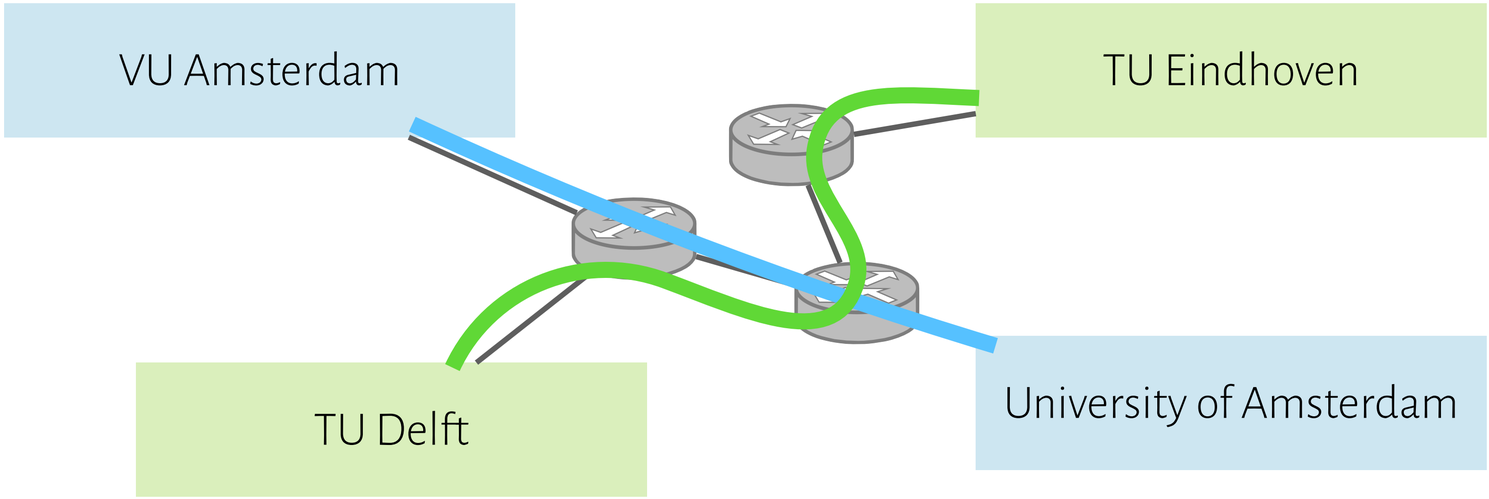

```
parser MyParser(…) {
    state start {
        transition parse_ethernet;
    }
    state parse_ethernet {
        packet.extract(hdr.ethernet);
        transition select(hdr.ethernet.etherType) {
            0x800: parse_ipv4;
            default: accept;
        }
    }
    state parse_ipv4 {
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

Transition between states

P4 for tunnelling

P4 allows for defining and parsing custom packet headers

- Allows you to implement your own protocols



How to define a packet header for packet tunnelling and parse the packets properly? Think about it!

P4 parser: variable-width header extraction

```
header IPv4_no_options_h {
    …
    bit<32> srcAddr;
    bit<32> dstAddr;
}

header IPv4_options_h {
    varbit<32> options;
}

parser MyParser(…) {
    state parse_ipv4 {
        packet.extract(hdr.ipv4);
        transition select(hdr.ipv4.ihl) {
            5: dispatch_on_protocol;
            default: parse_ipv4_options;
        }
    }
    state parse_ipv4_options {
        packet.extract(hdr.ipv4options, (hdr.ipv4.ihl - 5) <<
        transition dispatch_on_protocol;
    }
}
```

Fixed-width fields

Variable-width fields

ihl determines the length of options field

P4 parser: more advanced concepts

Parsing a header stack requires the parser to loop

- The only “loops” that are possible in P4
- Example in source routing: popping up all the headers to determine the next hop

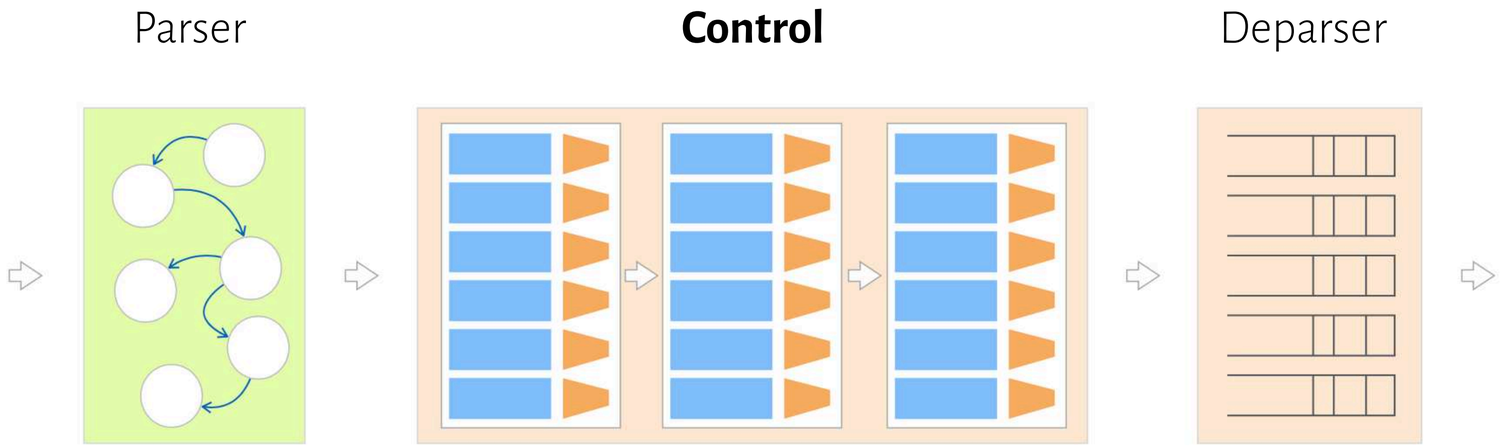
Why should we be cautious about loops?

Other concepts in P4 parser:

- Verify: error handling in the parser
- Lookahead: access bits that are not parsed yet
- Sub-parsers: like subroutines

https://p4.org/p4-spec/docs/P4-16-v1.0.0-spec.html

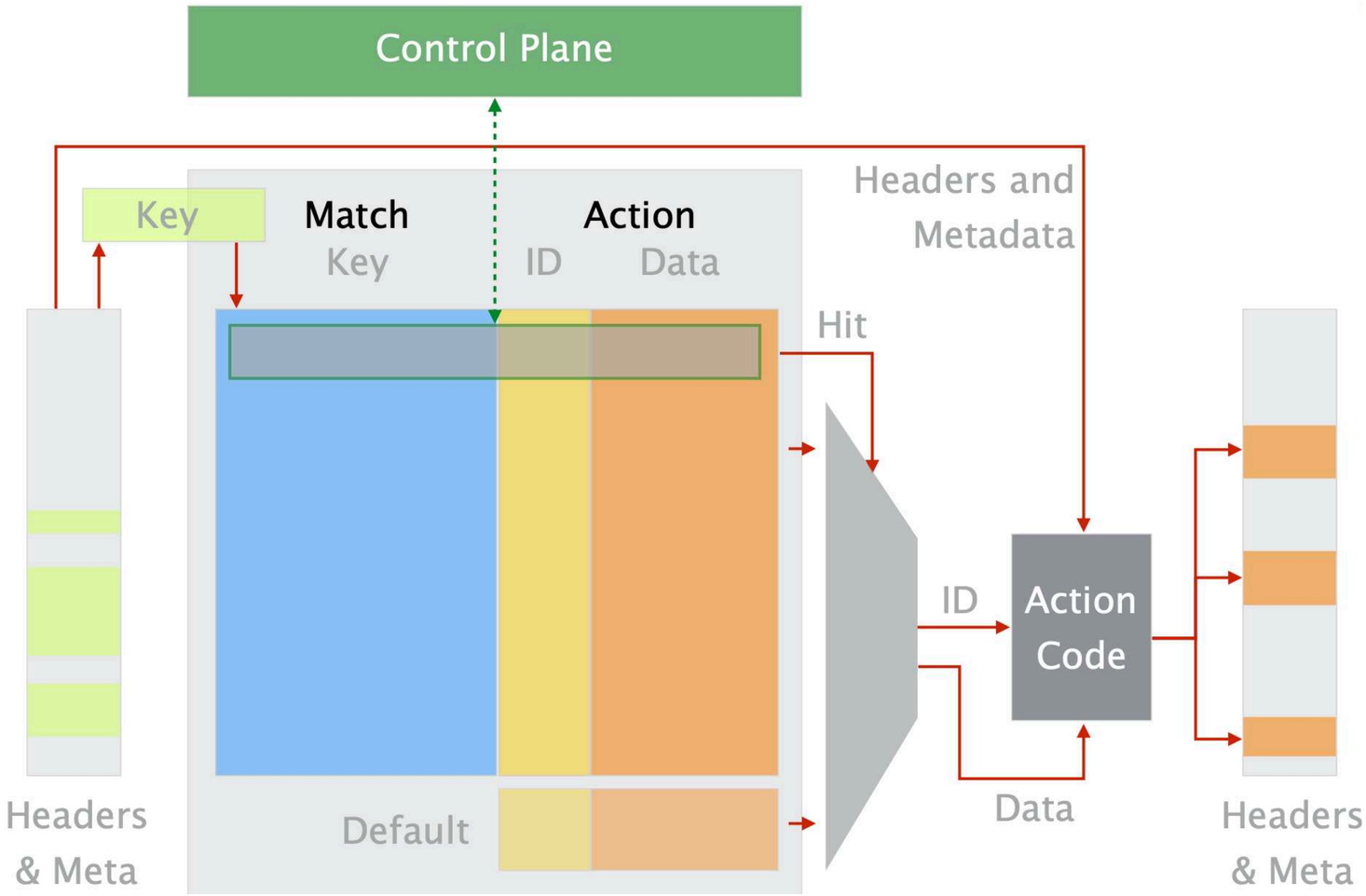
P4 processing overview



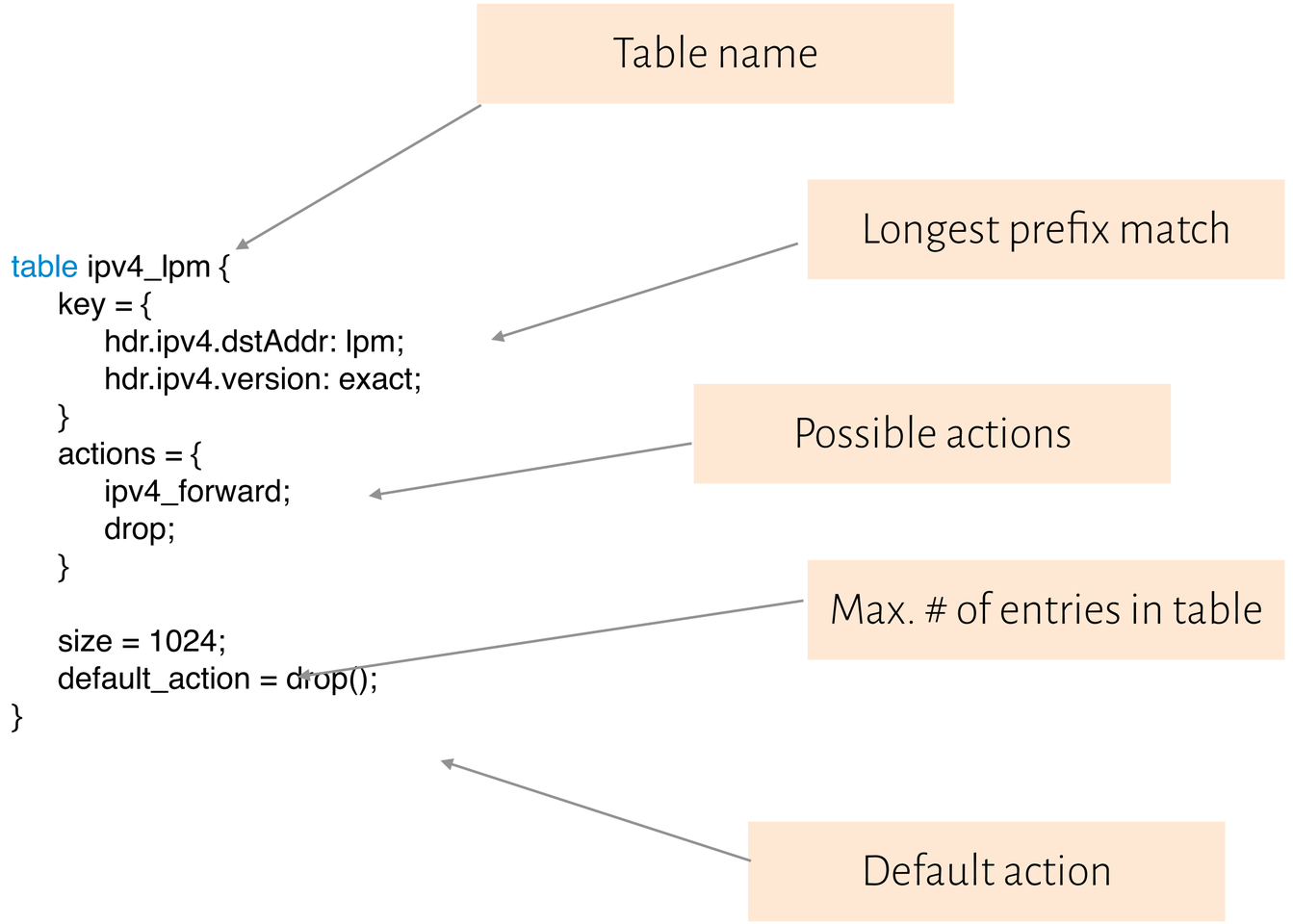
P4 control

Tables	Match a key and return an action
Actions	Similar to functions in C
Control flow	Similar to C but without loops

P4 control: tables



P4 control: tables



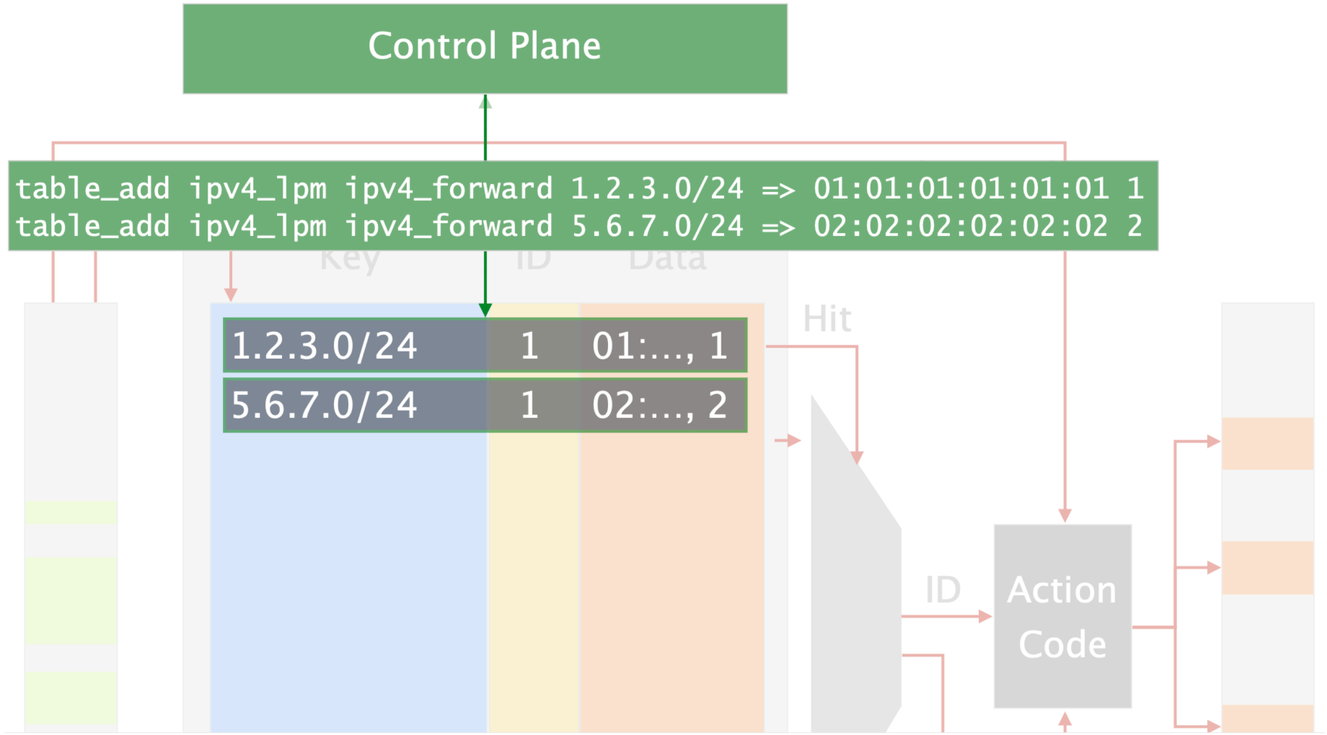
P4 control: match types

core.p4	exact	Exact comparison: 0x01020304
	ternary	Compare with mask: 0x01020304 & 0x0F0F0F0F
	lpm	Longest prefix match
v1model.p4	range	Check if in range: 0x01020304 - 0x010203FF
Other architectures	...	

P4 control: table entries

Table entries are added through the control plane

- Recall the SDN architecture



P4 control: actions

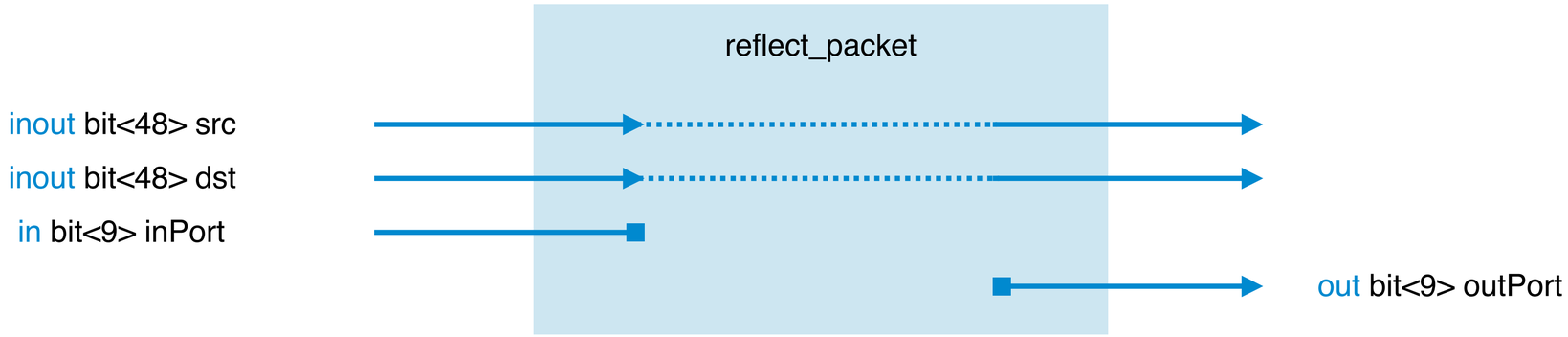
Actions are

- Blocks of statements that possibly modify the packets
- Usually take directional parameters indicating how the corresponding value is treated within the block

```
action reflect_packet(inout bit<48> src,
                      inout bit<48> dst,
                      in bit<9> inPort;
                      out bit<9> outPort;
                     ) {
    bit<48> tmp = src;
    src = dst;
    dst = tmp;
    outPort = inPort;
}

reflect_packet(hdr.ethernet.srcAddr, hdr.ethernet.dstAddr,
    standard_metadata.ingress_port, standard_metadata.egress_spec);
```

in: read only inside the action
out: uninitiated, write inside the action
inout: combination of in and out

P4 control: actions




```
action set_egress_port(bit<9> port) {
    standard_metadata.egress_spec = port;
}
```

Action parameters resulting from a table lookup do not take a direction

P4 control: control flow

Apply a table

```
ipv4_lpm.apply()
```

Check if there was a hit

```
if (ipv4_lpm.apply().hit) {…}
else {…}
```

Check which action was executed

```
switch (ipv4_lpm.apply().action_run) {
    ipv4_forward: {…}
}
```

v1model.p4

```
extern void verify_checksum<T, O>( in bool condition
                                   in T data
                                   inout O checksum
                                   HashAlgorithm algo
                                 );
```

```
extern void update_checksum<T, O>( in bool condition
                                   in T data
                                   inout O checksum
                                   HashAlgorithm algo
                                 );
```

P4 control: re-computing checksum

```
control MyComputeChecksum {
    apply {
        update_checksum(
            hdr.ipv4.isValid(),
            { hdr.ipv4.version,
              hdr.ipv4.ihl,
              hdr.ipv4.diffserv,
              hdr.ipv4.totalLen,
              hdr.ipv4.identification,
              hdr.ipv4.flags,
              hdr.ipv4.fragOffset,
              hdr.ipv4.ttl,
              hdr.ipv4.protocol,
              hdr.ipv4.srcAddr,
              hdr.ipv4.dstAddr },
            hdr.ipv4.hdrChecksum,
            HashAlgorithm.csum16);
    }
}
```

Pre-condition

Fields list

Checksum field

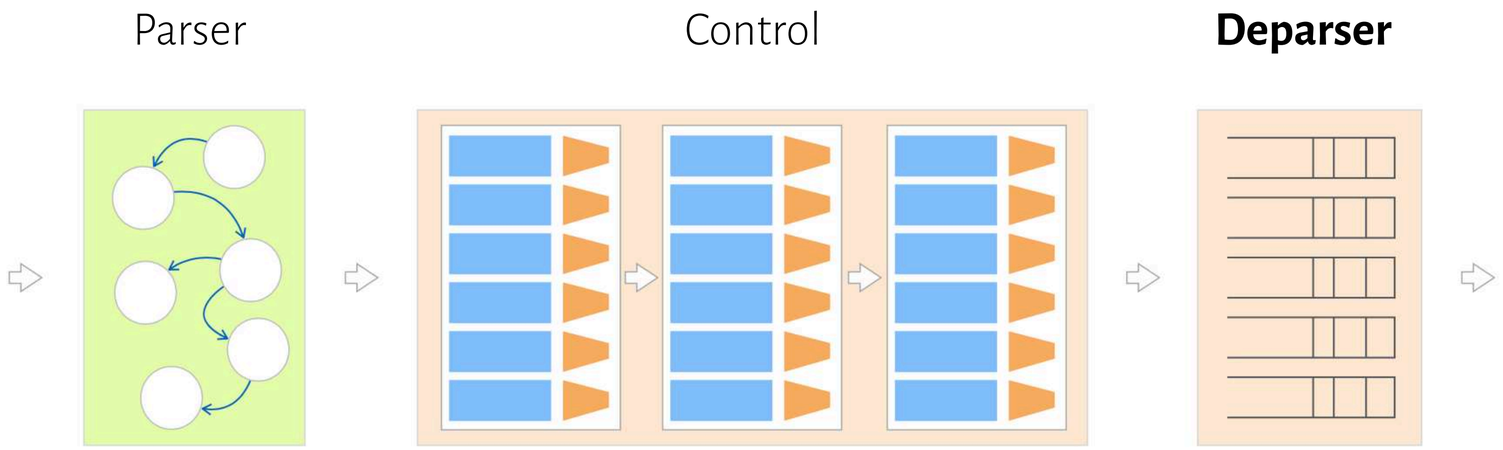
Checksum algorithm

P4 control: more advanced concepts

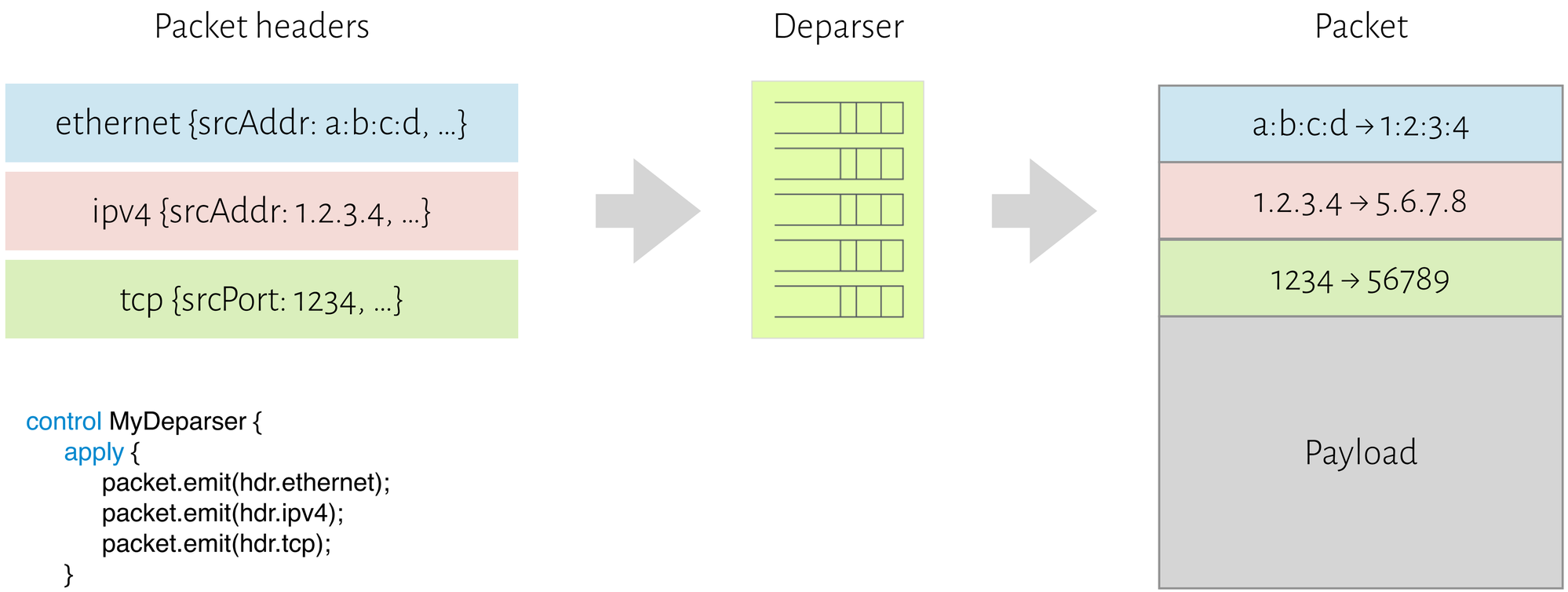
Cloning packets	Create a clone of a packet
Sending packets to control plane	Use dedicated Ethernet port, or target-specific mechanisms
Recirculating	Send packet through pipeline multiple times

Be cautious about recirculating!

P4 processing overview

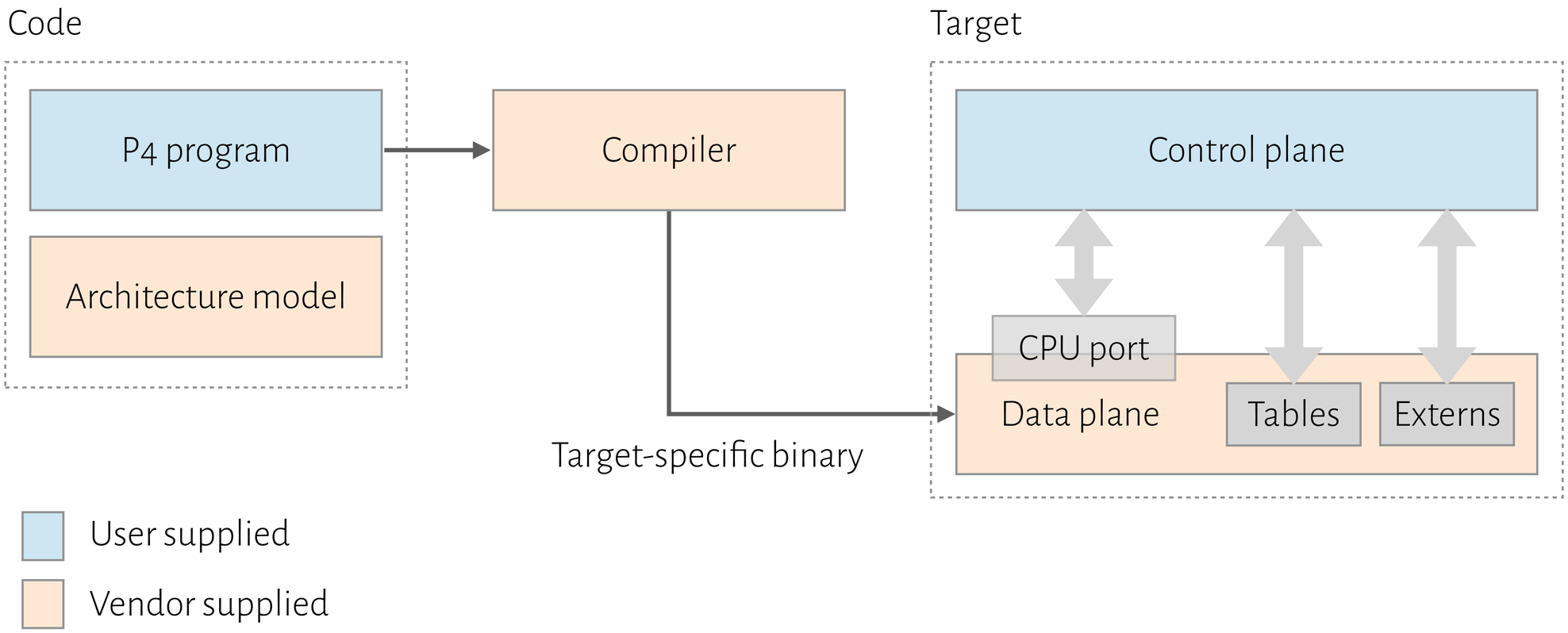


P4 deparser




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

P4 full circle



Questions

Hardware implementation of PISA architecture

Forwarding Metamorphosis: Fast Programmable Match-Action Processing in Hardware for SDN

Pat Bosshart†, Glen Gibb‡, Hun-Seok Kim†, George Varghese§, Nick McKeown‡,
Martin Izzard†, Fernando Mujica†, Mark Horowitz‡
†Texas Instruments ‡Stanford University §Microsoft Research
pat.bosshart@gmail.com {grg, nickm, horowitz}@stanford.edu
varghese@microsoft.com {hkim, izzard, fmujica}@ti.com

ABSTRACT

In Software Defined Networking (SDN) the control plane is physically separate from the forwarding plane. Control software programs the forwarding plane (e.g., switches and routers) using an open interface, such as OpenFlow. This paper aims to overcomes two limitations in current switching chips and the OpenFlow protocol: i) current hardware switches are quite rigid, allowing "Match-Action" processing on only a fixed set of fields, and ii) the OpenFlow specification only defines a limited repertoire of packet processing actions. We propose the RMT (reconfigurable match tables) model, a new RISC-inspired pipelined architecture for switching chips, and we identify the essential minimal set of action primitives to specify how headers are processed in hardware. RMT allows the forwarding plane to be changed in the field without modifying hardware. As in OpenFlow, the programmer can specify multiple match tables of arbitrary width and depth, subject only to an overall resource limit, with each table configurable for matching on arbitrary

1. INTRODUCTION

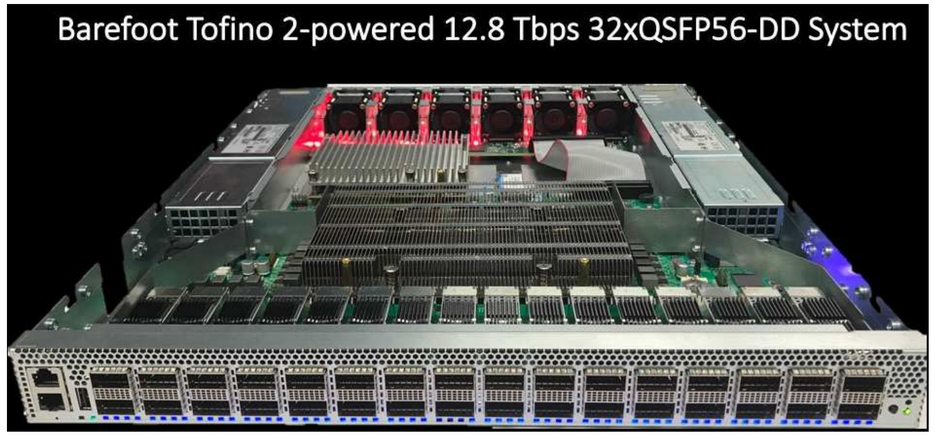
To improve is to change; to be perfect is to change often. — Churchill

Good abstractions—such as virtual memory and time-sharing—are paramount in computer systems because they allow systems to deal with change and allow simplicity of programming at the next higher layer. Networking has progressed because of key abstractions: TCP provides the abstraction of connected queues between endpoints, and IP provides a simple datagram abstraction from an endpoint to the network edge. However, routing and forwarding *within* the network remain a confusing conglomerate of routing protocols (e.g., BGP, ICMP, MPLS) and forwarding behaviors (e.g., routers, bridges, firewalls), and the control and forwarding planes remain intertwined inside closed, vertically integrated boxes.

Software-defined networking (SDN) took a key step in abstracting network functions by separating the roles of the

ACM SIGCOMM 2013

BAREFOOT NETWORKS



https://www.barefootnetworks.com

Research landscape surrounding programmable data plane

Data plane programmability for

Performance (e.g., load balancing)
Network monitoring
Applications offloading (caching, consensus, locking, aggregation)

Platforms
Program synthesis
Verification
Management

for data plane programmability

Application: network monitoring

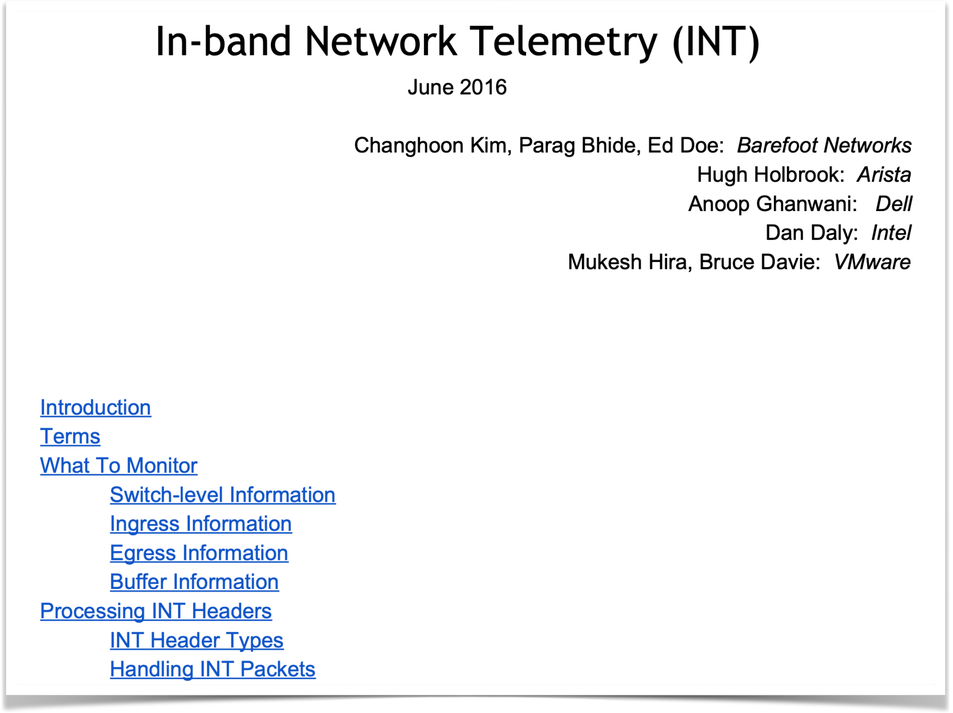
Limitations of current network monitoring solutions:

Not fast enough

- Involve CPU and control planes
- Network state changes rapidly

Do not provide end-to-end state

- Difficult to correlate per-element state with the actual path of a flow



In-band Network Telemetry (INT)

June 2016

Changhoon Kim, Parag Bhide, Ed Doe: *Barefoot Networks*
Hugh Holbrook: *Arista*
Anoop Ghanwani: *Dell*
Dan Daly: *Intel*
Mukesh Hira, Bruce Davie: *VMware*

https://p4.org/assets/INT-current-spec.pdf

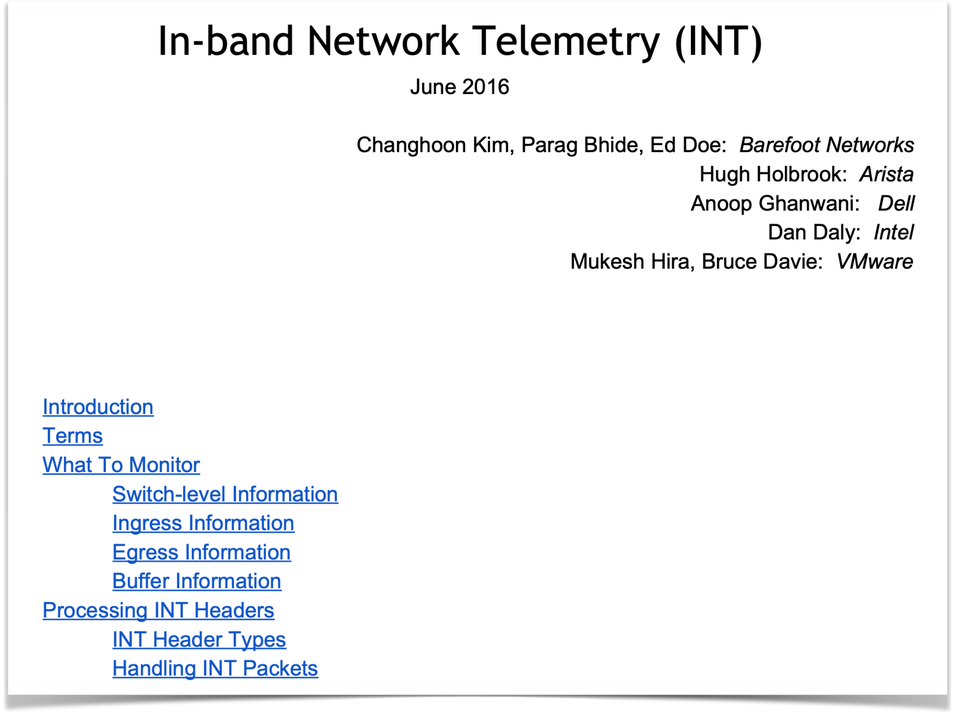
In-band network telemetry (INT)

Mechanisms for **collecting network state in the network data plane**

- As close to real-time as possible
- At current and future line rates
- With a framework that can adapt over time

Examples of **network state**

- Switch ID, ingress port ID, egress port ID
- Egress link utilisation
- Hop latency
- Egress queue occupancy
- Egress queue congestion status



In-band Network Telemetry (INT)

June 2016

Changhoon Kim, Parag Bhide, Ed Doe: *Barefoot Networks*
Hugh Holbrook: *Arista*
Anoop Ghanwani: *Dell*
Dan Daly: *Intel*
Mukesh Hira, Bruce Davie: *VMware*

https://p4.org/assets/INT-current-spec.pdf

INT using P4

P4 enables flexible packet parsing and modification for INT

P4 allows INT to adapt to

- Any encapsulation format
- Any state required to be collected
- Any feature, protocol – current and future

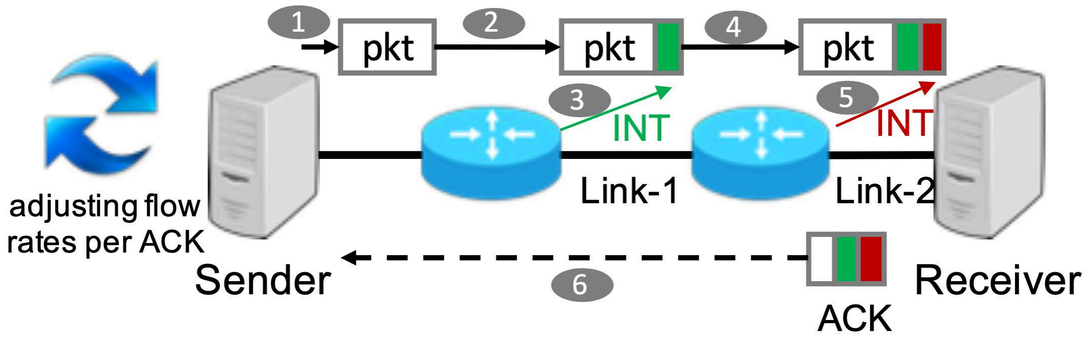
```
table int_inst {
    reads {
        int_header.instruction_mask: exact;
    }
    actions {
        int_set_header_i0;
        int_set_header_i1;
        int_set_header_i2;
        int_set_header_i3;
        ……
    }
}
```

Exact match table definition

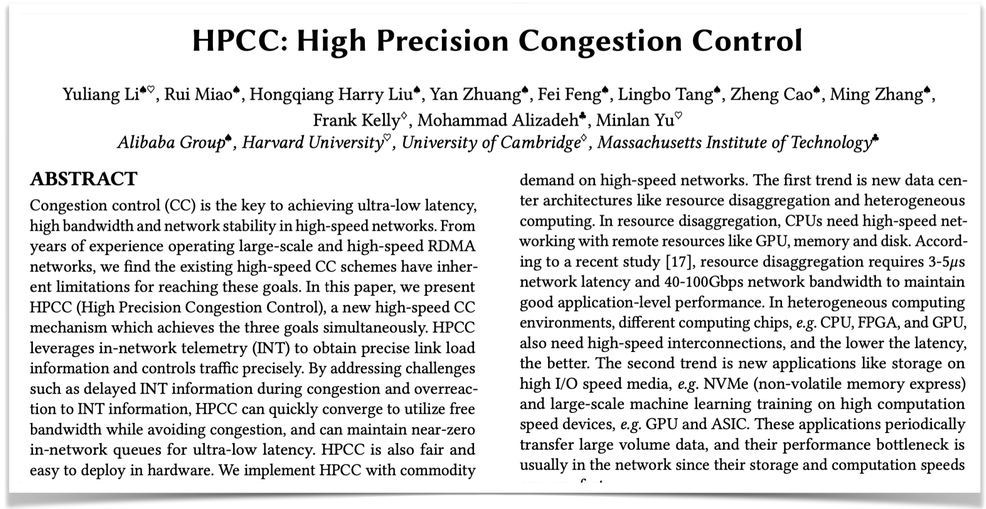
```
action int_set_header_i0() {}
action int_set_header_i1() {
    int_set_header_3();
}
action int_set_header_i2() {
    int_set_header_2();
}
action int_set_header_i3() {
    int_set_header_3();
    int_set_header_2();
}
```

Action definitions

Application: congestion control



Use INT to **obtain precise network link status information** and adjust sending rate based on such information



HPCC: High Precision Congestion Control

Yuliang Li[♠♡], Rui Miao[♠], Hongqiang Harry Liu[♠], Yan Zhuang[♠], Fei Feng[♠], Lingbo Tang[♠], Zheng Cao[♠], Ming Zhang[♠], Frank Kelly[♢], Mohammad Alizadeh[♣], Minlan Yu[♡]
Alibaba Group[♠], Harvard University[♡], University of Cambridge[♢], Massachusetts Institute of Technology[♣]

ABSTRACT

Congestion control (CC) is the key to achieving ultra-low latency, high bandwidth and network stability in high-speed networks. From years of experience operating large-scale and high-speed RDMA networks, we find the existing high-speed CC schemes have inherent limitations for reaching these goals. In this paper, we present HPCC (High Precision Congestion Control), a new high-speed CC mechanism which achieves the three goals simultaneously. HPCC leverages in-network telemetry (INT) to obtain precise link load information and controls traffic precisely. By addressing challenges such as delayed INT information during congestion and overreaction to INT information, HPCC can quickly converge to utilize free bandwidth while avoiding congestion, and can maintain near-zero in-network queues for ultra-low latency. HPCC is also fair and easy to deploy in hardware. We implement HPCC with commodity

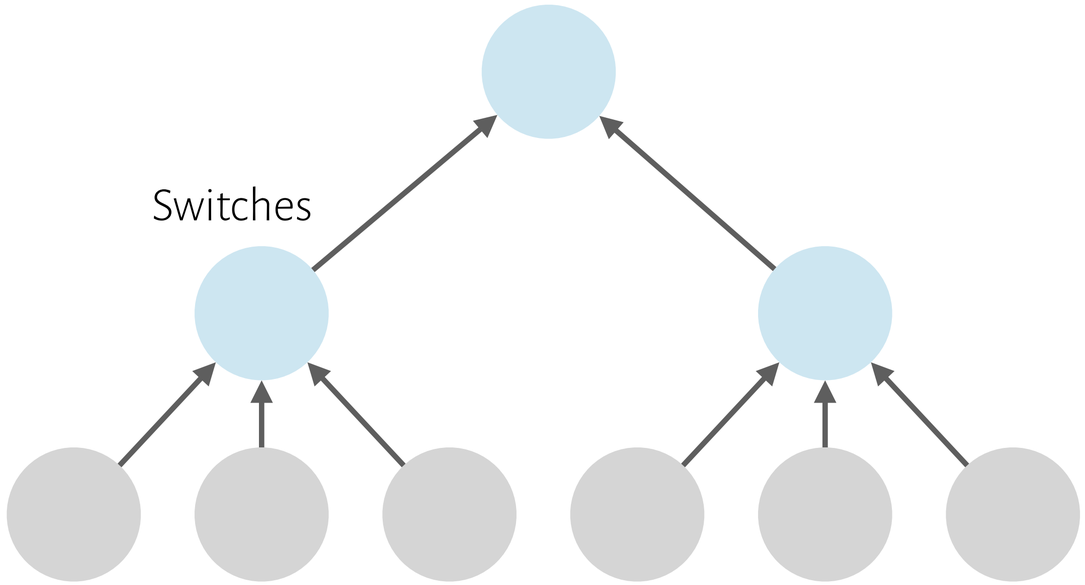
demand on high-speed networks. The first trend is new data center architectures like resource disaggregation and heterogeneous computing. In resource disaggregation, CPUs need high-speed networking with remote resources like GPU, memory and disk. According to a recent study [17], resource disaggregation requires 3-5μs network latency and 40-100Gbps network bandwidth to maintain good application-level performance. In heterogeneous computing environments, different computing chips, *e.g.* CPU, FPGA, and GPU, also need high-speed interconnections, and the lower the latency, the better. The second trend is new applications like storage on high I/O speed media, *e.g.* NVMe (non-volatile memory express) and large-scale machine learning training on high computation speed devices, *e.g.* GPU and ASIC. These applications periodically transfer large volume data, and their performance bottleneck is usually in the network since their storage and computation speeds

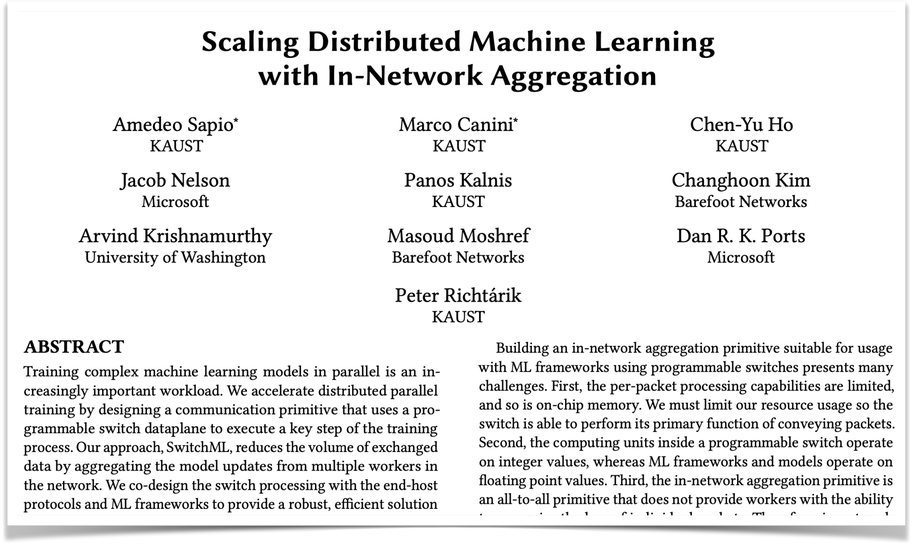
ACM SIGCOMM 2019

Think about the difference to ECN

Application: aggregation

Application-specific aggregation tree
(Typical for MapReduce, parameter server in machine learning training)





Scaling Distributed Machine Learning with In-Network Aggregation

Amedeo Sapio*
KAUST

Marco Canini*
KAUST

Chen-Yu Ho
KAUST

Jacob Nelson
Microsoft

Panos Kalnis
KAUST

Changhoon Kim
Barefoot Networks

Arvind Krishnamurthy
University of Washington

Masoud Moshref
Barefoot Networks

Dan R. K. Ports
Microsoft

Peter Richtárik
KAUST

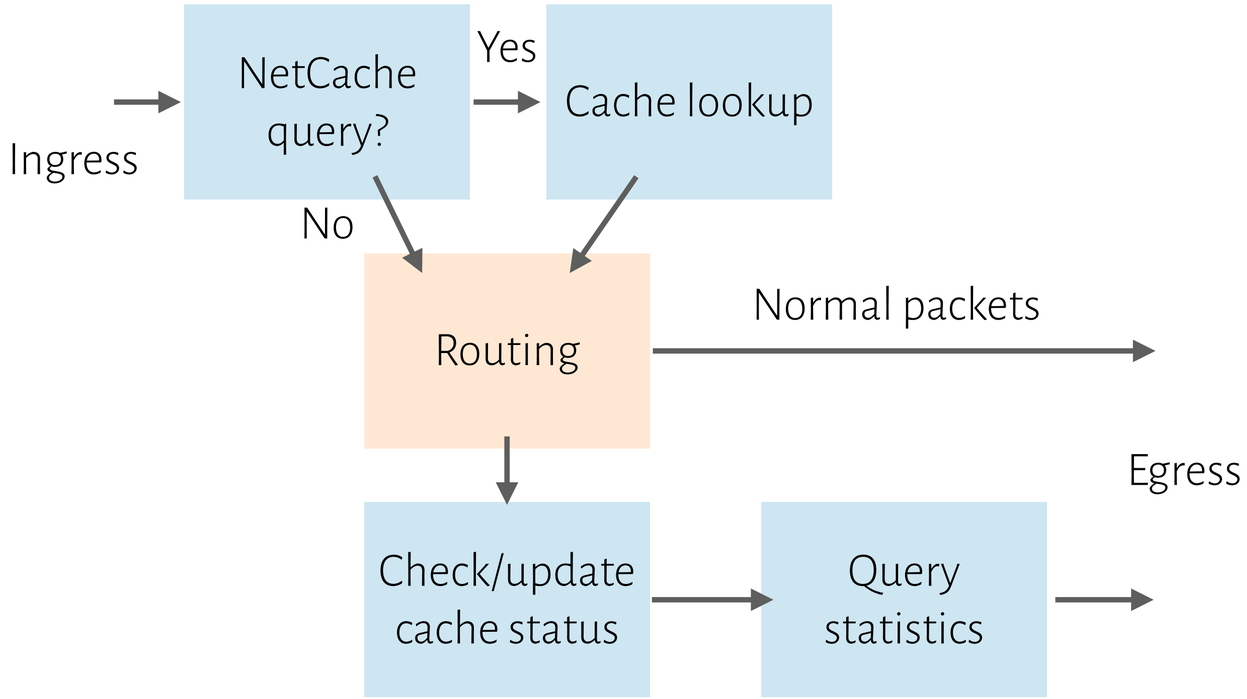
ABSTRACT

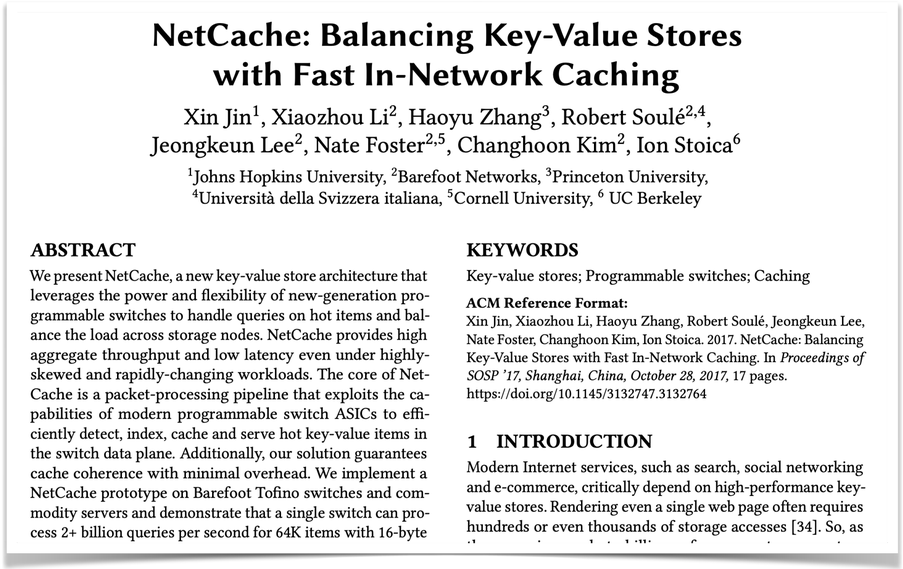
Training complex machine learning models in parallel is an increasingly important workload. We accelerate distributed parallel training by designing a communication primitive that uses a programmable switch dataplane to execute a key step of the training process. Our approach, SwitchML, reduces the volume of exchanged data by aggregating the model updates from multiple workers in the network. We co-design the switch processing with the end-host protocols and ML frameworks to provide a robust, efficient solution

Building an in-network aggregation primitive suitable for usage with ML frameworks using programmable switches presents many challenges. First, the per-packet processing capabilities are limited, and so is on-chip memory. We must limit our resource usage so the switch is able to perform its primary function of conveying packets. Second, the computing units inside a programmable switch operate on integer values, whereas ML frameworks and models operate on floating point values. Third, the in-network aggregation primitive is an all-to-all primitive that does not provide workers with the ability

Application: NetCache

Cache layer implemented in switches to accelerate operations in key-value store





NetCache: Balancing Key-Value Stores with Fast In-Network Caching

Xin Jin[1], Xiaozhou Li[2], Haoyu Zhang[3], Robert Soulé[2,4], Jeongkeun Lee[2], Nate Foster[2,5], Changhoon Kim[2], Ion Stoica[6]

[1]Johns Hopkins University, [2]Barefoot Networks, [3]Princeton University, [4]Università della Svizzera italiana, [5]Cornell University, [6] UC Berkeley

ABSTRACT

We present NetCache, a new key-value store architecture that leverages the power and flexibility of new-generation programmable switches to handle queries on hot items and balance the load across storage nodes. NetCache provides high aggregate throughput and low latency even under highly-skewed and rapidly-changing workloads. The core of NetCache is a packet-processing pipeline that exploits the capabilities of modern programmable switch ASICs to efficiently detect, index, cache and serve hot key-value items in the switch data plane. Additionally, our solution guarantees cache coherence with minimal overhead. We implement a NetCache prototype on Barefoot Tofino switches and commodity servers and demonstrate that a single switch can process 2+ billion queries per second for 64K items with 16-byte

KEYWORDS

Key-value stores; Programmable switches; Caching

ACM Reference Format:

Xin Jin, Xiaozhou Li, Haoyu Zhang, Robert Soulé, Jeongkeun Lee, Nate Foster, Changhoon Kim, Ion Stoica. 2017. NetCache: Balancing Key-Value Stores with Fast In-Network Caching. In *Proceedings of SOSP '17, Shanghai, China, October 28, 2017,* 17 pages.
https://doi.org/10.1145/3132747.3132764

1 INTRODUCTION

Modern Internet services, such as search, social networking and e-commerce, critically depend on high-performance key-value stores. Rendering even a single web page often requires hundreds or even thousands of storage accesses [34]. So, as

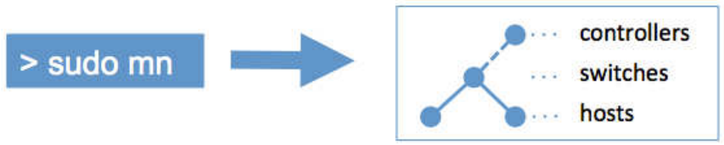
ACM SOSP 2017

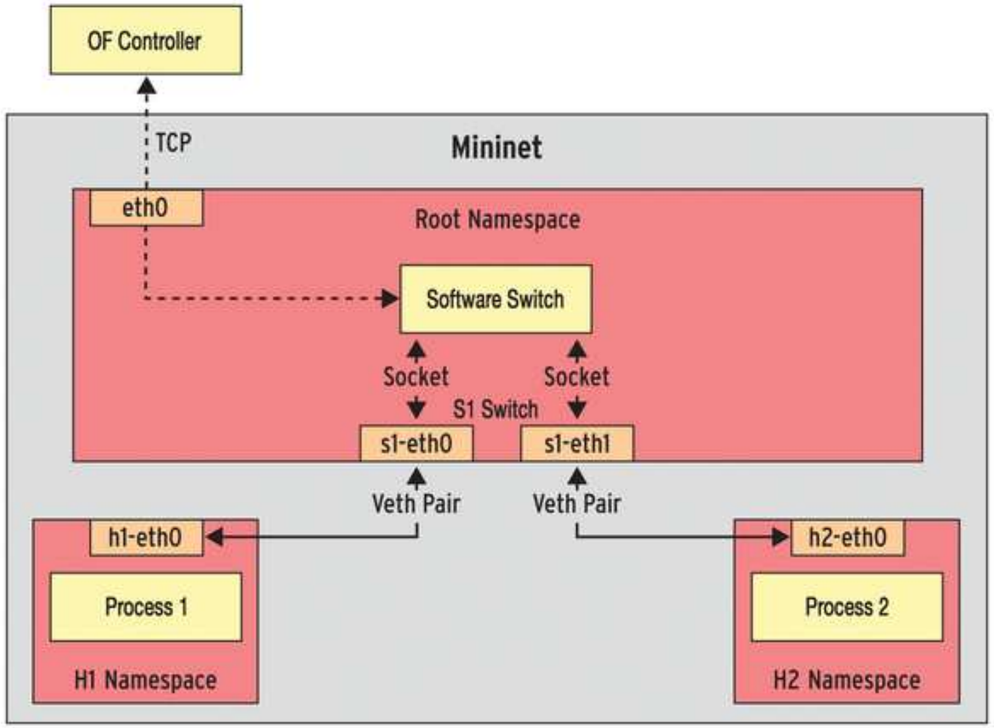
Questions

How to try SDN on your computer?

Mininet

Mininet creates a **realistic virtual network**, running **real kernel**, **switch**, and **application code**, on **a single machine**, in **seconds**, with **a single command**...





Installation

Download the Mininet VM

- https://github.com/mininet/mininet/wiki/Mininet-VM-Images

Native installation from source

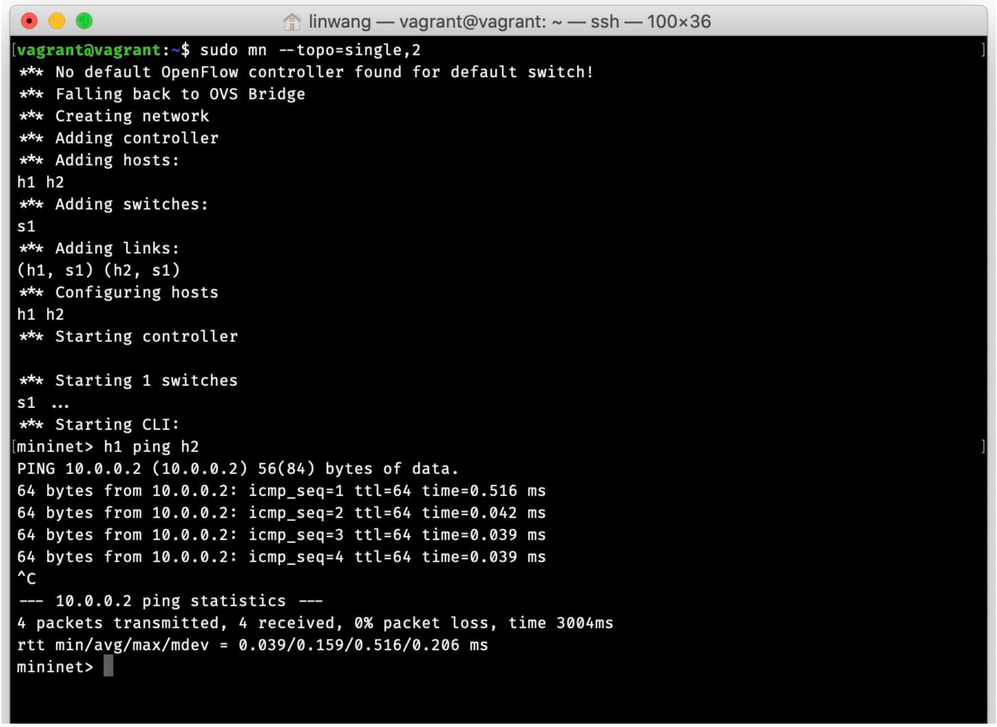
- git clone git://github.com/mininet/mininet
- mininet/util/install.sh -a

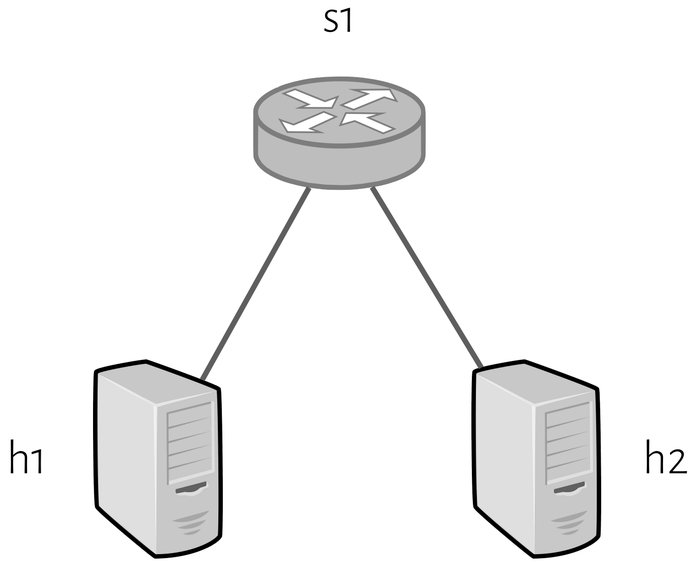
Install from packages

- sudo apt install mininet

Play with Mininet

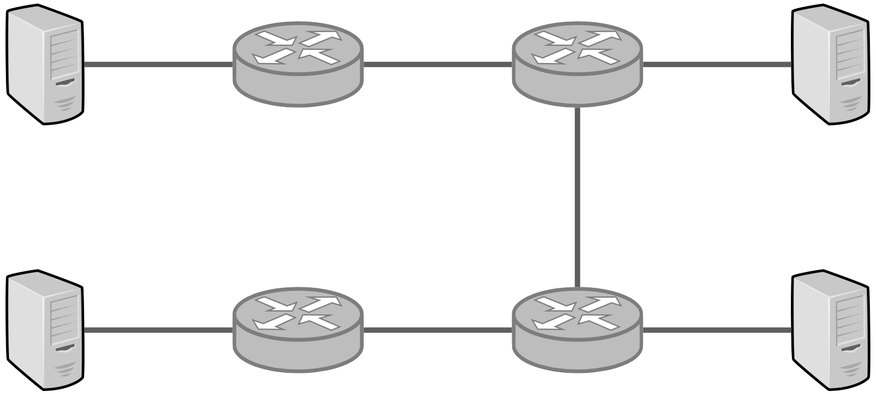
sudo mn --topo=single,2
h1 ping -c5 h2
xterm h2
tcpdump -n -i h2-eth0 arp





Programming with Mininet

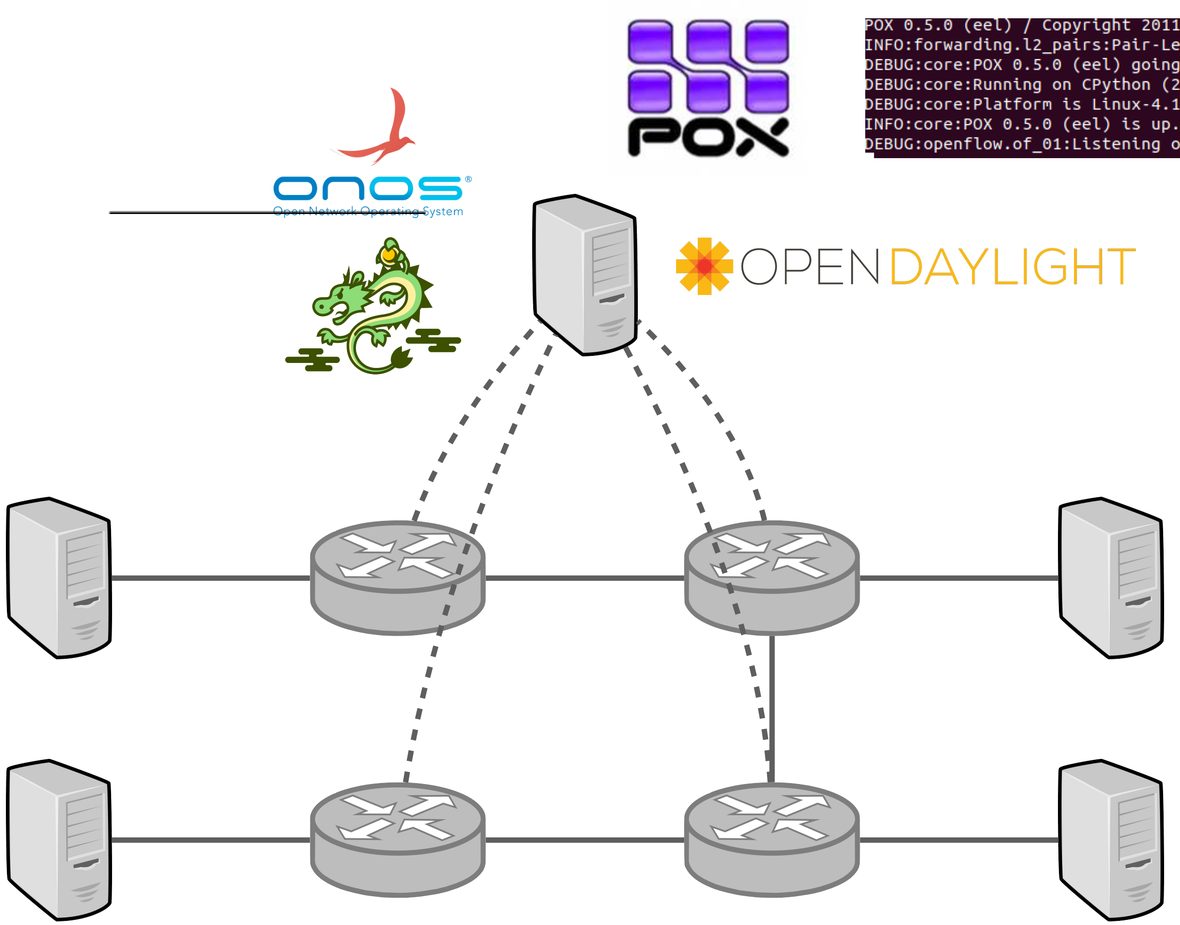
Create a (broken) rectangle network topology in Python



```
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import OVSSwitch, Controller, RemoteController

class RectTopo(Topo):
    "Rectangular topology example."

    def __init__(self):

        Topo.__init__(self)

        h1 = self.addHost('h1')
        h2 = self.addHost('h2')
        h3 = self.addHost('h3')
        h4 = self.addHost('h4')

        s1 = self.addSwitch('s1')
        s2 = self.addSwitch('s2')
        s3 = self.addSwitch('s3')
        s4 = self.addSwitch('s4')

        self.addLink(h1, s1)
        self.addLink(h2, s2)
        self.addLink(h3, s3)
        self.addLink(h4, s4)

        self.addLink(s1, s2)
        self.addLink(s2, s3)
        self.addLink(s3, s4)

topos = {'recttopo': (lambda: RectTopo())}
```

OpenFlow controller

git clone git@github.com:noxrepo/pox.git
sudo ./pox/pox.py --verbose forwarding.l2_pairs
sudo mn --custom brectangle.py --topo recttop --**controller remote** --switch ovsk --mac



How to try P4 and data plane programming on your laptop?

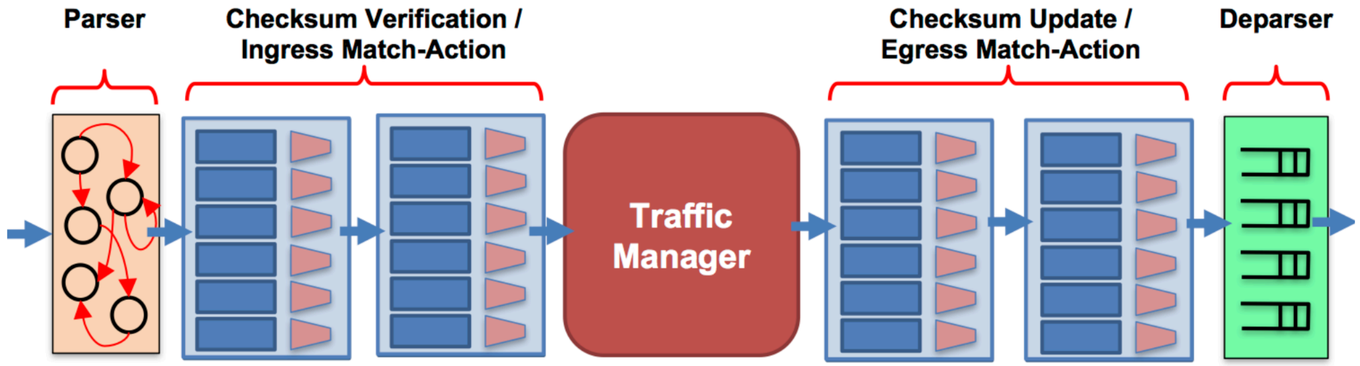
Use Mininet to set up the network environment

Use software switches **bmv2**: https://github.com/p4lang/behavioral-model

See P4 tutorials: https://github.com/p4lang/tutorials

Working with P4 in Mininet on BMV2

P4.org has developed an open source software switch called BMV2 (behavioral model version 2) designed to be a target for P4 programs. That is, P4 programs can be compiled onto it to configure how it processes packets. Every P4 target supports one or more P4 target architectures. The target architecture supported by BMV2 that we will be using for these introductory exercises is called the V1Model. A diagram of the V1Model is shown below:

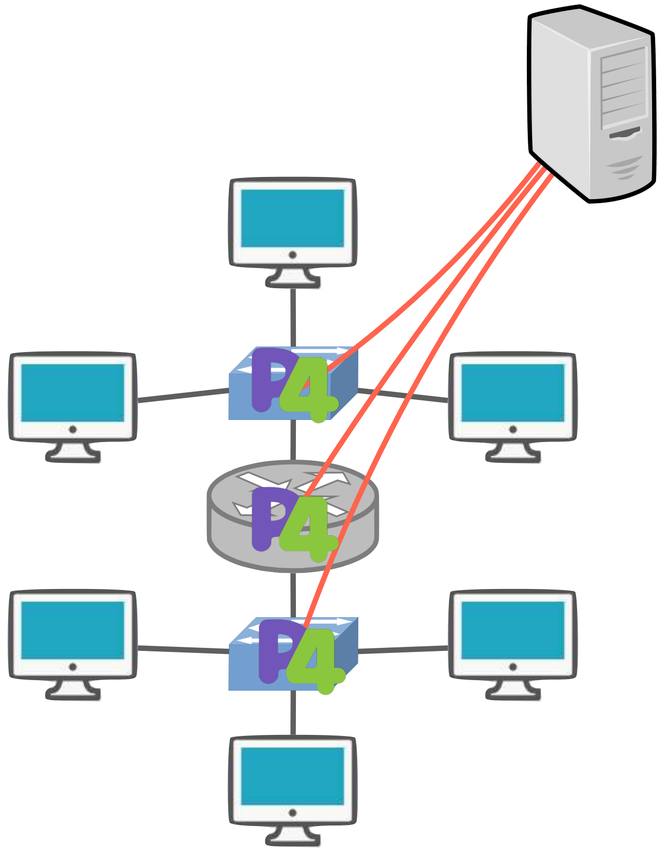


https://build-a-router-instructors.github.io/deliverables/p4-mininet/

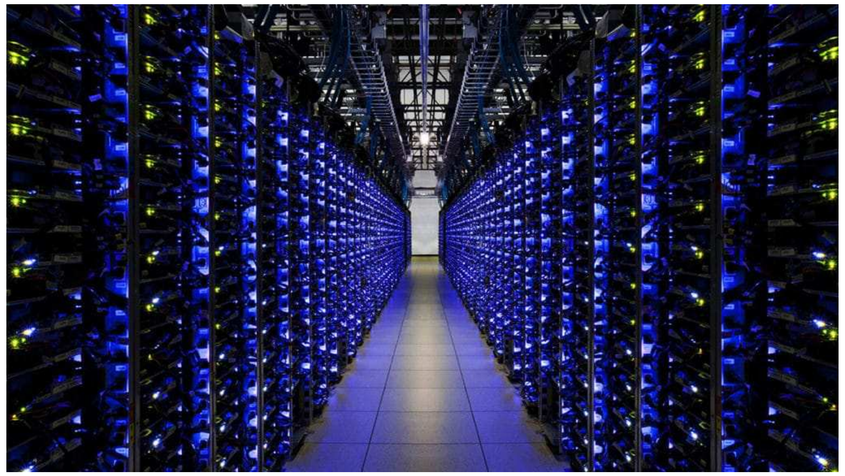
Summary

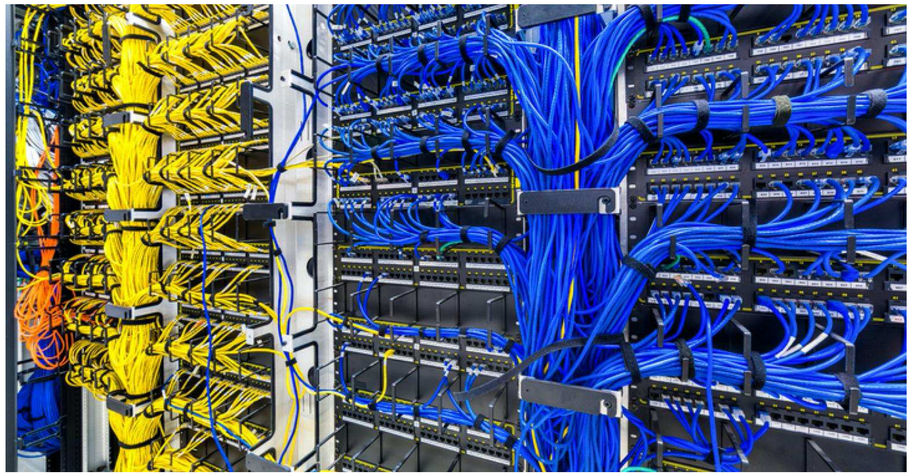
Lecture 9: Programmable data plane

- Limitations of OpenFlow
- How to enable data plane programmability
- PISA abstraction and P4
- P4 language basics
- P4 applications
- How to try these cool things on your computer? Mininet



Next week: cloud networking





How to build a high-performance network for data centres?