ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Laboratory of Network Programmability and Automation

ONOS+P4 Tutorial Controlling P4 data planes with ONOS

Andrea Melis

Dipartimento di Informatica – Scienza e Ingegneria

Copyright

The following slides belong to:

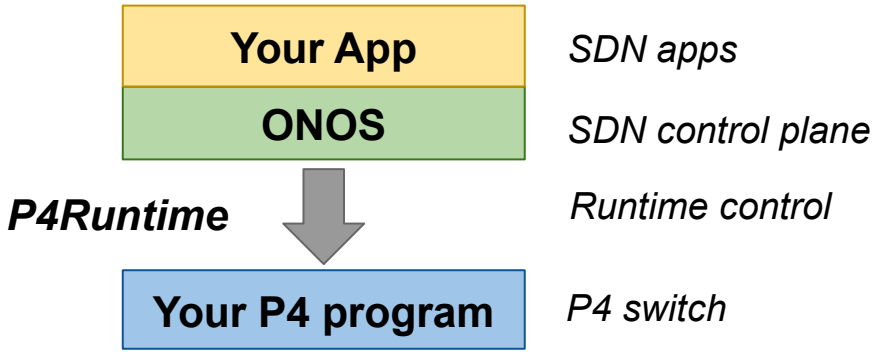
The Open Network Foundtion

Link to slides:

http://bit.ly/onos-p4-tutorial-slides

Goal of this session

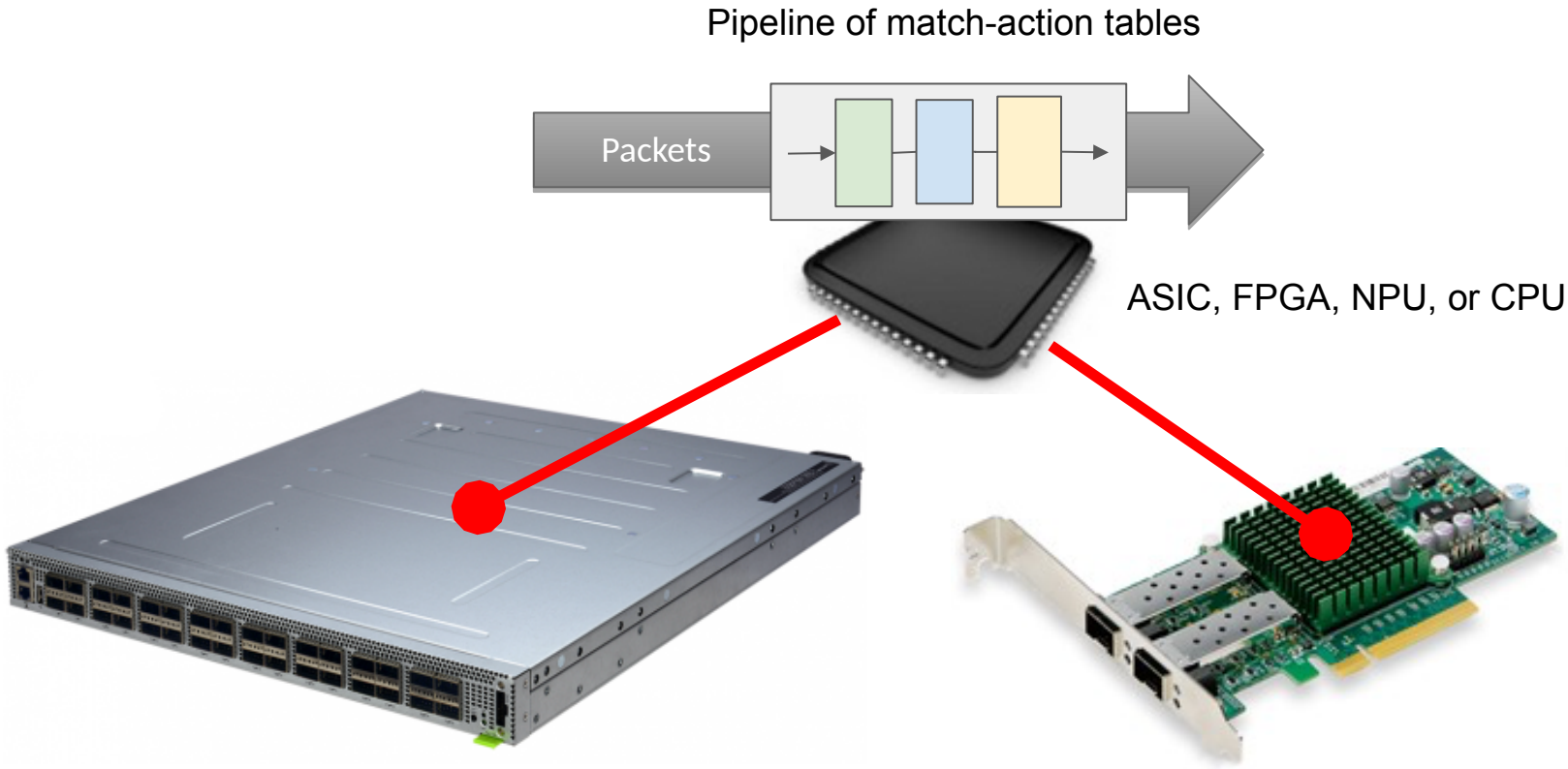
- **Learn the basics of P4, P4Runtime and ONOS**
- **Show you the "big picture" of P4**
  - Acquire enough knowledge to build full-stack network applications
  - Go from a P4 idea to an end-to-end solution
- **Learn the tools to practically experiment with it**



P4

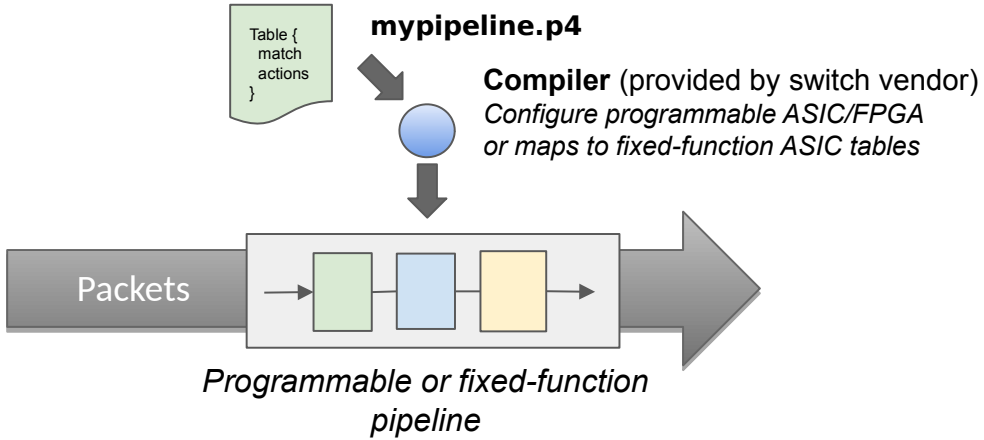
The Data Plane Programming Language

What is a pipeline?



P4

- **Domain-specific language to formally define the data plane pipeline**
  - Describe protocol headers, lookup tables, actions, counters, etc.
  - Can describe fast pipelines (e.g ASIC, FPGA) as well as a slower ones (e.g. SW switch)
- **Good for programmable switches, as well as fixed-function ones**
  - Defines “contract” between the control plane and data plane for runtime control



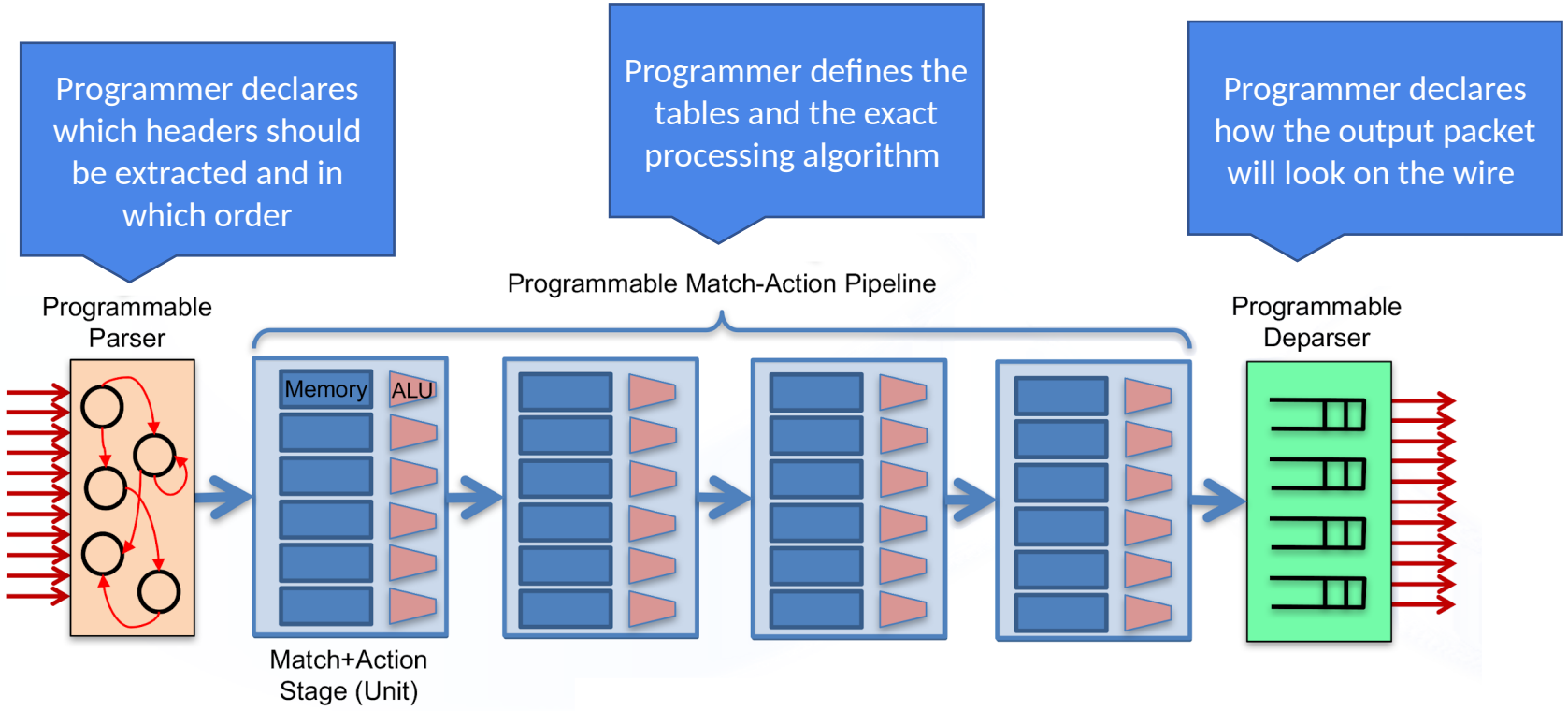
Evolution of the language

- **P4$_{14}$**
  - Original version of the language
  - Assumed specific device capabilities
  - Good only for a subset of programmable switch/targets

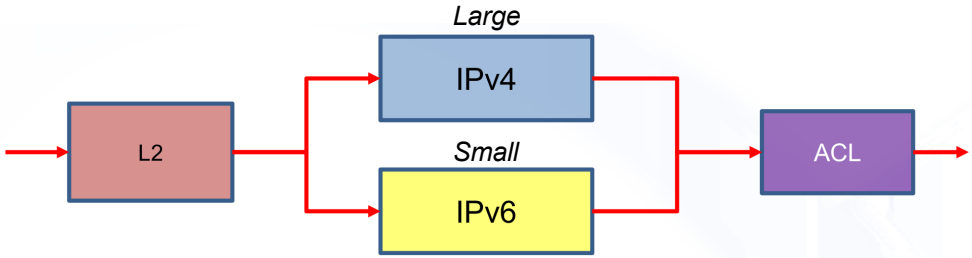
- **P4$_{16}$ (Focus of this tutorial)**
  - More mature and stable language definition
  - Does not assume device capabilities
    - Defined via external libraries/architecture definition
  - Good for many targets
    - E.g., switches or NICs, programmable or fixed-function

PISA: Protocol-Independent Switch Architecture

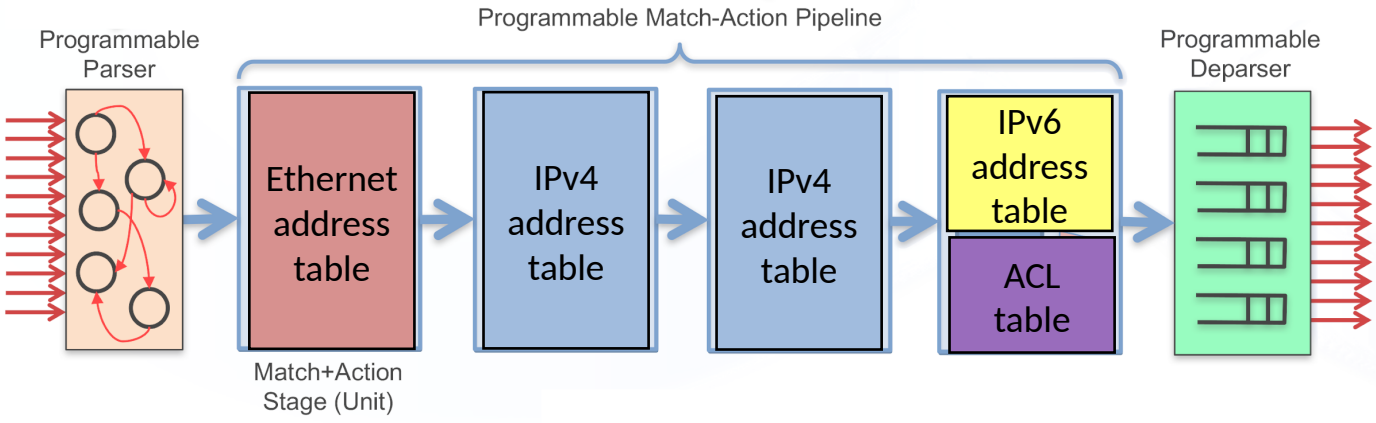
Abstract machine model of a high-speed programmable switch architecture



Compiling a simple logical pipeline on PISA

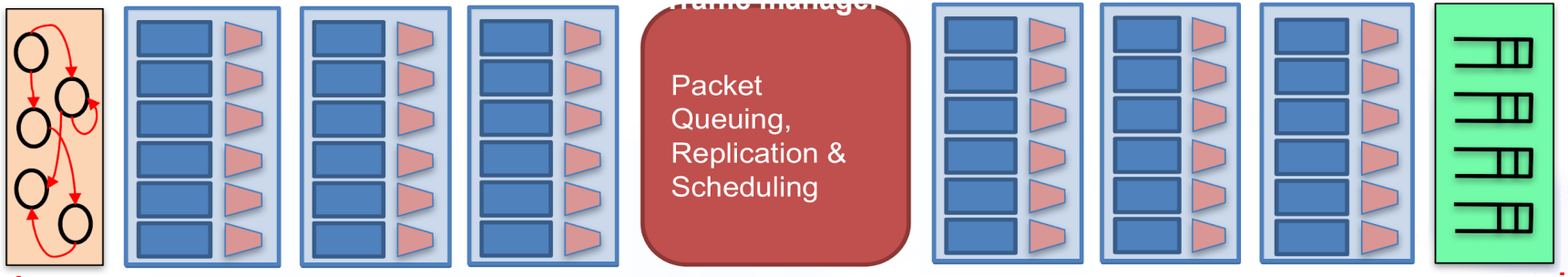


P4 compiler
Allocate resources to realize the pipeline



V1Model P4 Switch Architecture

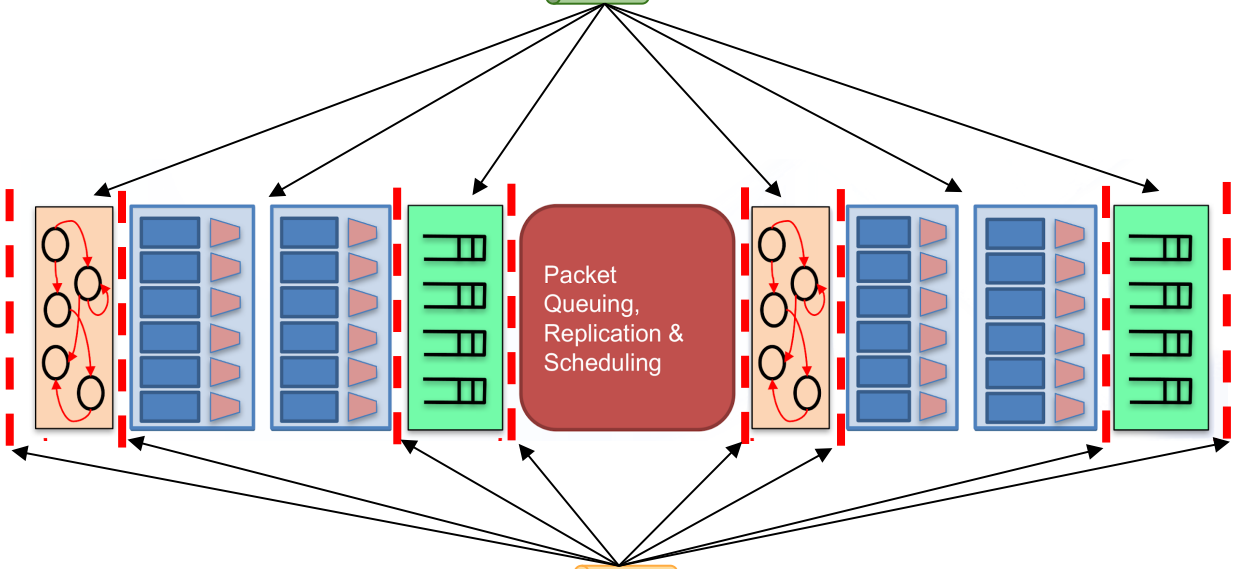
- Parser/deparser → P4 programmable
- Checksum verification/update → P4 programmable
- Ingress Pipeline → P4 programmable
- Egress Pipeline → P4 programmable
  - Match on egress port
- Traffic Manager → Fixed function
  - Queueing, Replication (multicast), scheduling



P4 architectures

my_program.p4

Written against a specific architecture
Defines the processing of each block



architecture.p4
Provided by switch vendor

Defines which blocks are available, the interfaces of each block, and their capabilities

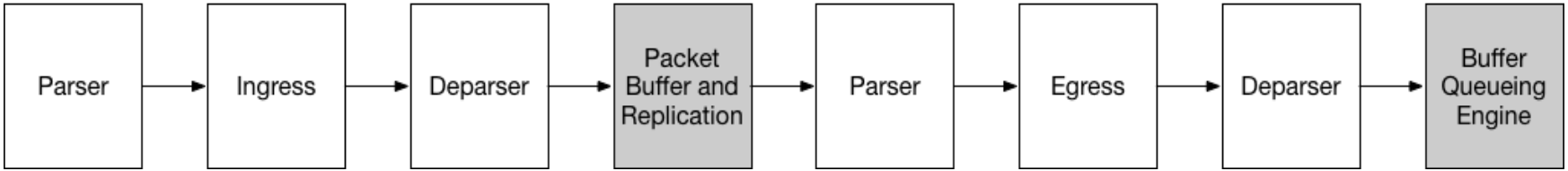
Preliminary takeaways

- **Can I implement/describe this or that feature with P4?**
  - It depends on the architecture
  - The P4 language is flexible enough to express almost any behavior based on match-action tables
  - Specific capabilities depend on the architecture
    - e.g. ternary match vs. longest-prefix match vs. exact match
- **Can I execute my P4 program on a switch *X* from vendor *Y*?**
  - Probably yes, if vendor *Y* provides a P4 compiler for the specific architecture used

Architectures enable portability of P4 programs across different HW and SW targets

PSA - Portable Switch Architecture

- **Community-developed architecture (P4.org Arch WG)**
  - https://github.com/p4lang/p4-spec/tree/master/p4-16/psa
- **Describes common capabilities of a network switch**
- **6 programmable P4 blocks + 2 fixed-function blocks**
- **Defines capabilities beyond match+action tables**
  - Counters, meters, stateful registers, hash functions, etc.



P4 program template (V1Model)

```
#include <core.p4>
#include <v1model.p4>
/* HEADERS */
struct metadata { ... }
struct headers {
  ethernet_t   ethernet;
  ipv4_t       ipv4;
}
/* PARSER */
parser MyParser(packet_in packet,
          out headers hdr,
          inout metadata meta,
          inout standard_metadata_t smeta) {
  ...
}
/* CHECKSUM VERIFICATION */
control MyVerifyChecksum(in headers hdr,
                         inout metadata meta) {
   ...
}
/* INGRESS PROCESSING */
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t std_meta) {
  ...
}
```

```
/* EGRESS PROCESSING */
control MyEgress(inout headers hdr,
                 inout metadata meta,
                 inout standard_metadata_t std_meta) {
  ...
}
/* CHECKSUM UPDATE */
control MyComputeChecksum(inout headers hdr,
                          inout metadata meta) {
  ...
}
/* DEPARSER */
control MyDeparser(inout headers hdr,
                   inout metadata meta) {
  ...
}
/* SWITCH */
V1Switch(
  MyParser(),
  MyVerifyChecksum(),
  MyIngress(),
  MyEgress(),
  MyComputeChecksum(),
  MyDeparser()
) main;
```

P4 program example: simple_router.p4

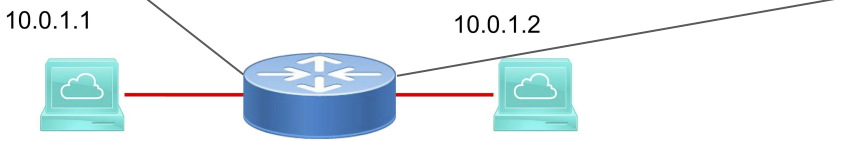
```
header ethernet_t {
  bit<48> dst_addr;
  bit<48> src_addr;
  bit<16> eth_type;
}

header ipv4_t {
  bit<4>  version;
  bit<4>  ihl;
  bit<8>  diffserv;
  …
}

parser parser_impl(packet_in pkt, out headers_t hdr) {
  /* Parser state machine to extract header fields */
}
```

```
action set_next_hop(bit<48> dst_addr) {
  ethernet.dst_addr = dst_addr;
  ipv4.ttl = ipv4.ttl - 1;
}
…
table ipv4_routing_table {
  key = {
        ipv4.dst_addr : LPM; // longest-prefix match
  }
  actions = {
        set_next_hop();
        drop();
   }
  size = 4096; // table entries
}
```

Runtime: simple_router.p4

- **Data plane program (P4)**
  - Defines the match-action tables
  - Performs the lookup
  - Executes the chosen action
- **Control plane (runtime)**
  - Populates table entries with specific information
  - Based on configuration, automatic discovery, protocol calculations

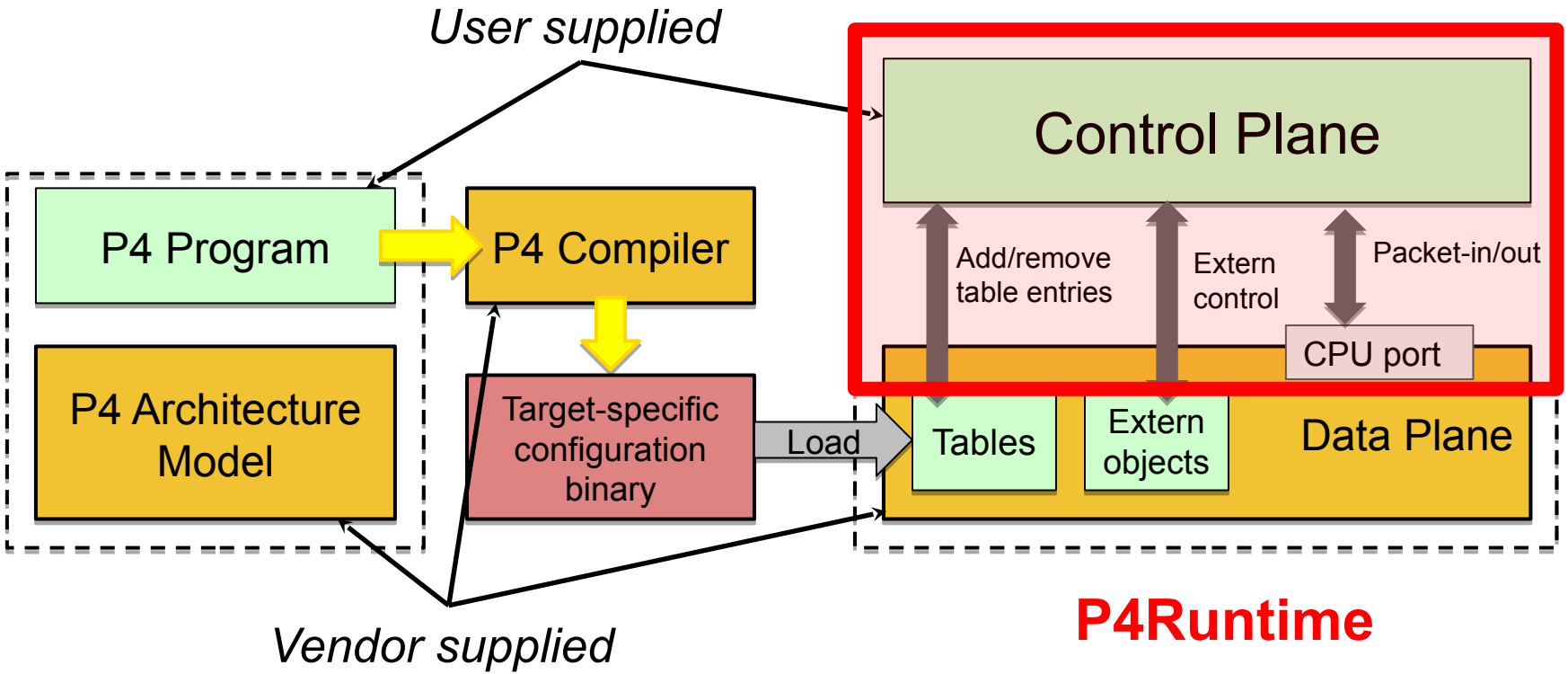
```
action ipv4_forward(bit<48> dst_addr, bit<9> port) {
  ethernet.dst_addr = dst_addr;
  standard_metadata.egress_spec = port;
  ipv4.ttl = ipv4.ttl - 1;
}
table ipv4_routing_table {
 key = {
        ipv4.dst_addr : LPM; // longest-prefix match
 }
 actions = {
        ipv4_forward();
        drop();
  }
}
```



Control plane populates table entries

Key	Action	Action Data
10.0.1.1/32	ipv4_forward	dstAddr=00:00:00:00:01:01 port=1
10.0.1.2/32	drop	
*`	NoAction	

P4 workflow summary

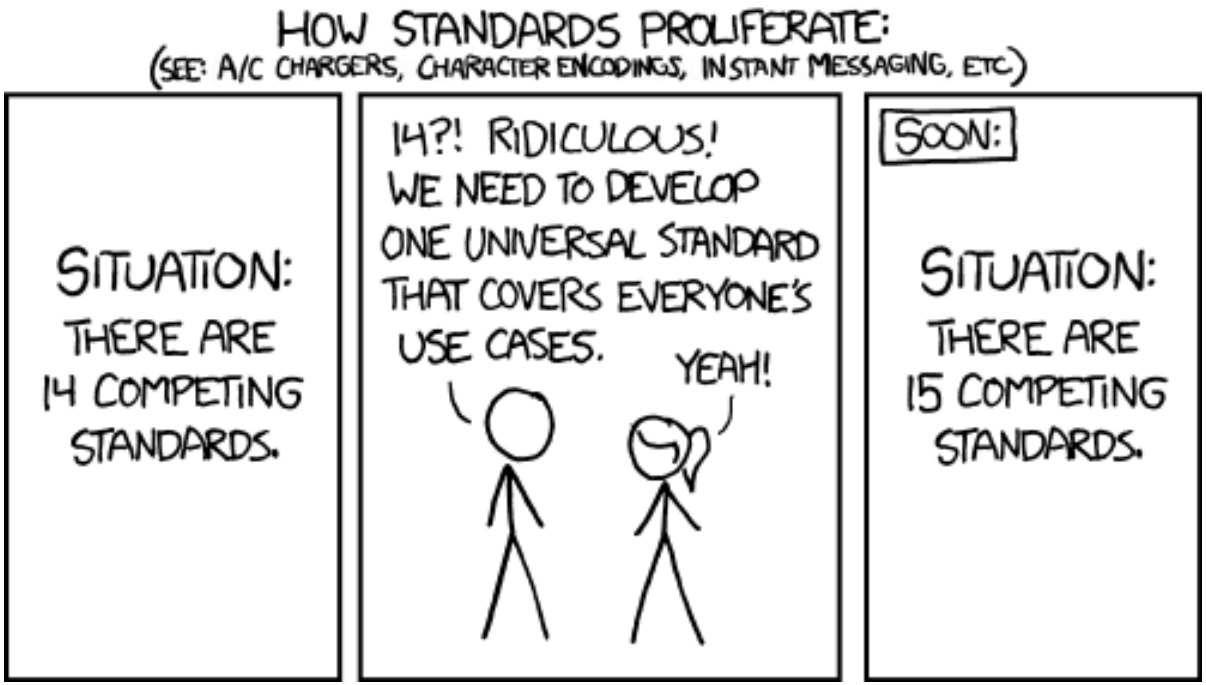


P4Runtime

Runtime control API for P4-defined data planes

Do we need yet another data plane control API?

Can’t we re-use existing APIs such as OpenFlow or SAI?



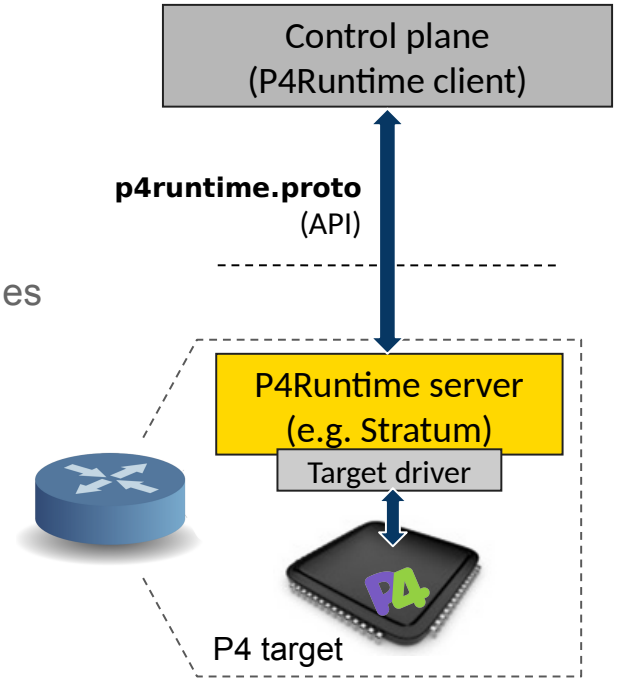
Credits XKCD

Yes, we need P4Runtime

API	Target-independent Same API works with different switches from different vendors	Protocol-independent Same API allows control of any data plane protocol, standard or custom	Pipeline-independent Same API allows control of many arbitrary pipelines formally specified
OpenFlow		✘ Protocol headers and actions hard-coded in the spec	(with TTP)
Switch Abstraction Interface (SAI)		✘ Designed for legacy forwarding pipelines (L2/L3/ACL)	✘ Implicit fixed-function pipeline
P4Runtime			(with P4)

P4Runtime overview

- **API for runtime control of P4-defined switches**
  - Designed around PSA architecture but can be extended to others
- **Community-developed (p4.org API WG)**
  - Initial contribution by Google and Barefoot
  - RC of version 1.0 available: https://p4.org/p4-spec/
- **gRPC/protobuf-based API definition**
  - Automatically generate client/server code for many languages
- **P4 program-independent**
  - API doesn't change with the P4 program
- **Enables field-reconfigurability**
  - Ability to push new P4 program, i.e. re-configure the switch pipeline, without recompiling the switch software stack



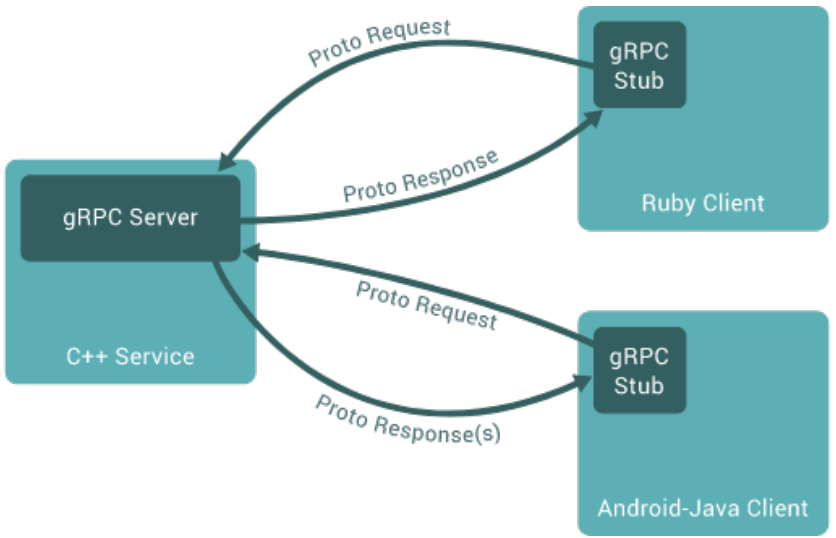
Protocol Buffers (protobuf) Basics

- Language for describing data for serialization in a structured way
- Strongly typed
- Auto-generate code to serialize/deserialize messages in:
  - Code generators for: *Action Script, C, C++, C#, Clojure, Lisp, D, Dart, Erlang, Go, Haskell, Java, Javascript, Lua, Objective C, OCaml, Perl, PHP, Python, Ruby, Rust, Scala, Swift, Visual Basic, ...*
- Platform-neutral
- Extensible and backwards compatible

```
syntax = "proto3";

message Person {
  string name = 1;
  int32 id = 2;
  string email = 3;

  enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
  }

  message PhoneNumber {
    string number = 1;
    PhoneType type = 2;
  }

  repeated PhoneNumber phone = 4;
}
```

gRPC Basics

- Use Protobuf to define service API and messages
- Automatically generate native client and server code in:
  - C / C++, C#, Dart, Go, Java, Node.js, PHP, Python, Ruby
- Transport over HTTP/2.0 and TLS
  - Efficient single TCP connection implementation that supports bidirectional streaming



p4runtime.proto (gRPC service)

Enables a local or remote control plane to

- Load the pipeline/P4 program
- Write/read pipeline state (e.g. table entries, meters, groups, etc.)
- Send/receive packets to/from controller (packet-in/out)
- Arbitrate mastership (for distributed control planes)

```
service P4Runtime {
   rpc Write(WriteRequest) returns (WriteResponse) {}
   rpc Read(ReadRequest) returns (stream ReadResponse) {}
   rpc
SetForwardingPipelineConfig(SetForwardingPipelineConfigRequest)
       returns (SetForwardingPipelineConfigResponse) {}
   rpc
GetForwardingPipelineConfig(GetForwardingPipelineConfigRequest)
       returns (GetForwardingPipelineConfigResponse) {}
   rpc StreamChannel(stream StreamMessageRequest)
       returns (stream StreamMessageResponse) {}
```

From: https://github.com/p4lang/p4runtime/blob/master/proto/p4/v1/p4runtime.proto

P4Runtime `Write` Request

```
message WriteRequest {
 uint64 device_id = 1;
 uint64 role_id = 2;
 Uint128 election_id = 3;
 repeated Update updates = 4;
}
```

```
message Update {
 enum Type {
   UNSPECIFIED = 0;
   INSERT = 1;
   MODIFY = 2;
   DELETE = 3;
 }
 Type type = 1;
 Entity entity = 2;
}
```

```
message Entity {
 oneof entity {
   ExternEntry extern_entry = 1;
   TableEntry table_entry = 2;
   ActionProfileMember
        action_profile_member = 3;
   ActionProfileGroup
        action_profile_group = 4;
   MeterEntry meter_entry = 5;
   DirectMeterEntry direct_meter_entry = 6;
   CounterEntry counter_entry = 7;
   DirectCounterEntry direct_counter_entry = 8;
   PacketReplicationEngineEntry
        packet_replication_engine_entry = 9;
   ValueSetEntry value_set_entry = 10;
   RegisterEntry register_entry = 11;
 }
}
```

P4Runtime TableEntry

p4runtime.proto simplified excerpts:

```
message TableEntry {
  uint32 table_id;
  repeated FieldMatch match;
  Action action;
  int32 priority;
  ...
}
```

```
message Action {
  uint32 action_id;
  message Param {
    uint32 param_id;
    bytes value;
  }
  repeated Param params;
}
```

```
message FieldMatch {
  uint32 field_id;
  message Exact {
    bytes value;
  }
  message Ternary {
    bytes value;
    bytes mask;
  }
  message LPM {
    bytes value;
    uint32 prefix_length;
  }
  ...
  oneof field_match_type {
    Exact exact;
    Ternary ternary;
    LPM lpm;
    ...
  }
}
```

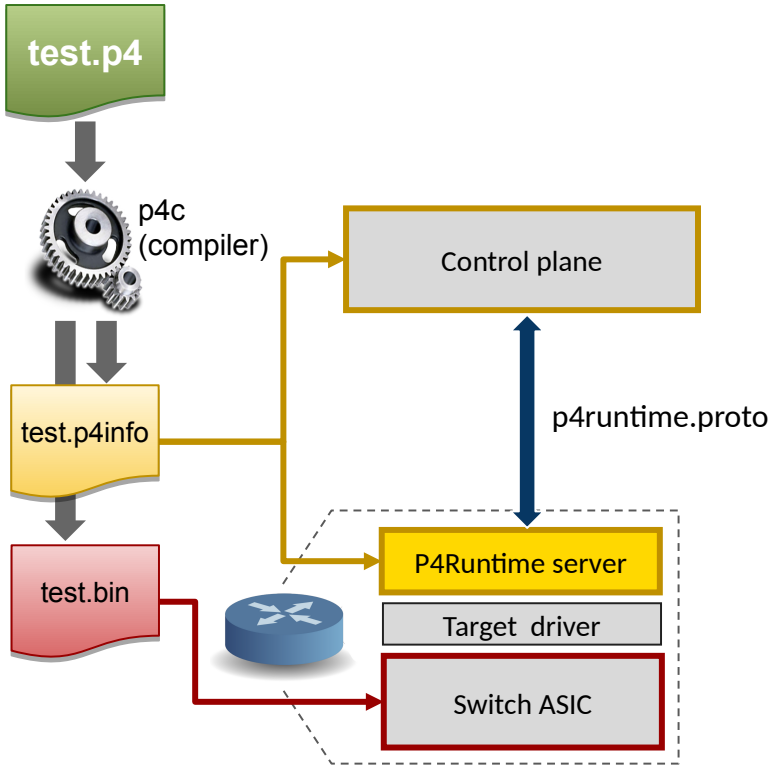
To add a table entry, the control plane needs to know:

- **IDs of P4 entities**
  - Tables, field matches, actions, params, etc.
- **Field matches for the particular table**
  - Match type, bit width, etc.
- **Parameters for the particular action**
- **Other P4 program attributes**

P4 compiler workflow

P4 compiler generates 2 outputs:

1. **Target-specific binaries**
   - Used to realize switch pipeline
     (e.g. binary config for ASIC, bitstream for FPGA, etc.)
2. **P4Info file**
   - Describes “schema” of pipeline for runtime control
     - Captures P4 program attributes such as tables, actions, parameters, etc.
   - Protobuf-based format
   - Target-independent compiler output
     - Same P4Info for SW switch, ASIC, etc.



Full P4Info protobuf specification:
https://github.com/p4lang/p4runtime/blob/master/proto/p4/config/v1/p4info.proto

P4Info example

basic_router.p4

```
...

action ipv4_forward(bit<48> dstAddr,
                    bit<9> port) {
  /* Action implementation */
}

...

table ipv4_lpm {
  key = {
     hdr.ipv4.dstAddr: lpm;
  }
  actions = {
     ipv4_forward;
     ...
  }
  ...
}
```

P4 compiler

basic_router.p4info

```
actions {
  id: 16786453
  name: "ipv4_forward"
  params {
    id: 1
    name: "dstAddr"
    bitwidth: 48
    ...
    id: 2
    name: "port"
    bitwidth: 9
  }
}
...
tables {
  id: 33581985
  name: "ipv4_lpm"
  match_fields {
    id: 1
    name: "hdr.ipv4.dstAddr"
    bitwidth: 32
    match_type: LPM
  }
  action_ref_id: 16786453
}
```

P4Runtime table entry example

basic_router.p4

```
action ipv4_forward(bit<48> dstAddr,
                    bit<9>  port) {
  /* Action implementation */
}
table ipv4_lpm {
  key = {
     hdr.ipv4.dstAddr: lpm;
  }
  actions = {
     ipv4_forward;
     ...
  }
  ...
}
```

Control plane generates

Logical view of table entry

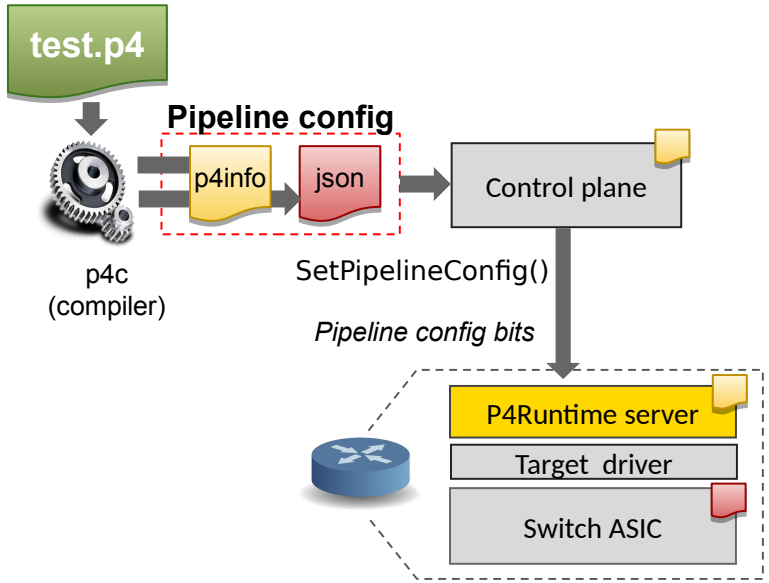
```
hdr.ipv4.dstAddr=10.0.1.1/32
        -> ipv4_forward(00:00:00:00:00:10, 7)
```

Protobuf message

```
table_entry {
  table_id: 33581985
  match {
    field_id: 1
    lpm {
      value: "\n\000\001\001"
      prefix_len: 32
    }
  }
  action {
    action_id: 16786453
    params {
      param_id: 1
      value: "\000\000\000\000\000\n"
    }
    params {
      param_id: 2
      value: "\000\007"
    }
  }
}
```

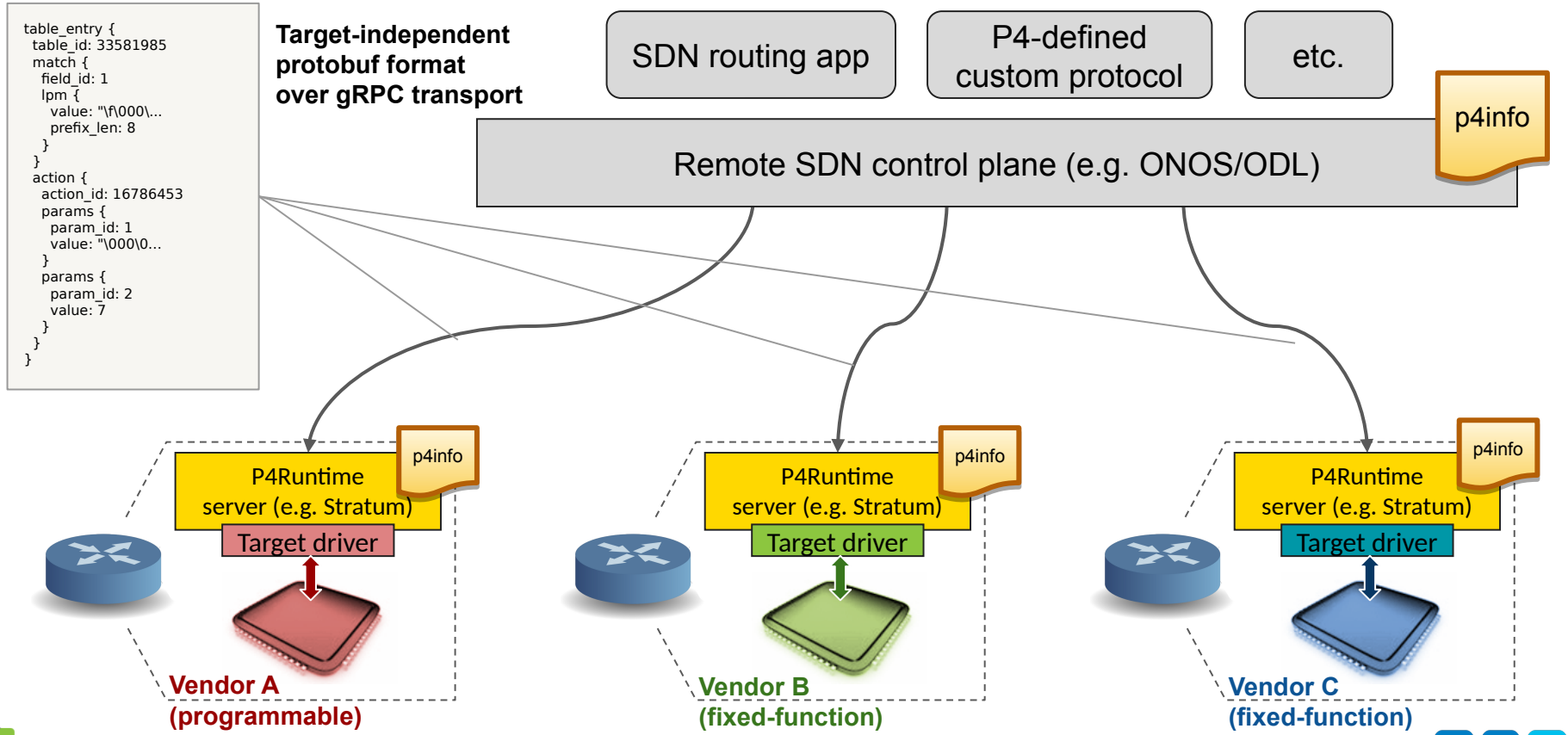
P4Runtime SetPipelineConfig


```
message
SetForwardingPipelineConfigRequest {
 enum Action {
   UNSPECIFIED = 0;
   VERIFY = 1;
   VERIFY_AND_SAVE = 2;
   VERIFY_AND_COMMIT = 3;
   COMMIT = 4;
   RECONCILE_AND_COMMIT = 5;
 }
 uint64 device_id = 1;
 uint64 role_id = 2;
 Uint128 election_id = 3;
 Action action = 4;
 ForwardingPipelineConfig config = 5;
}
```



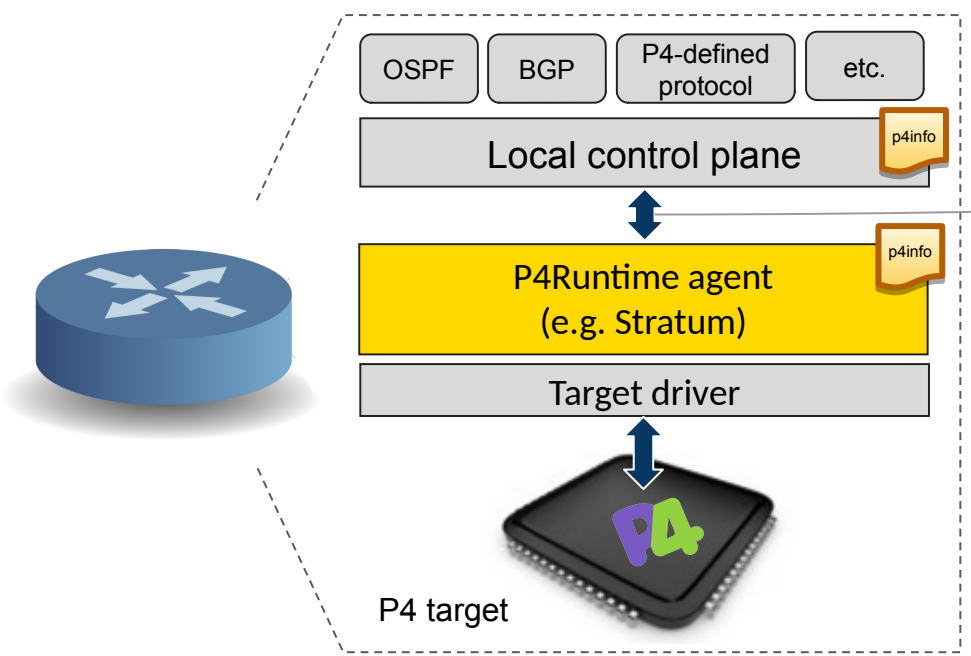

```
message ForwardingPipelineConfig {
 config.P4Info p4info = 1;
 // Target-specific P4
configuration.
 bytes p4_device_config = 2;
}
```

Silicon-independent remote control



Slide courtesy P4.org

Portability of local control plane



Same target-independent protobuf format (gRPC or other mechanism)

```
table_entry {
  table_id: 33581985
  match {
    field_id: 1
    lpm {
      value: "\f\000\...
      prefix_len: 8
    }
  }
  action {
    action_id: 16786453
    params {
      param_id: 1
      value: "\000\0...
    }
    params {
      param_id: 2
      value: 7
    }
  }
}
```

P4Runtime summary

- **P4Runtime is an improvement over previous data plane APIs**
  - Realizes the vision of OpenFlow 2.0
  - Provides protocol and pipeline-independence
  - Protocols supported and pipeline are formally specified using P4
- **Based on protobuf and gRPC**
  - Makes it easy to implement a P4Runtime client/server by auto-generating code
- **P4Info as a contract between control and data plane**
  - Generated by P4 compiler
  - Needed by the control plane to format the body of P4Runtime messages (e.g. to add table entry)

ONOS

A control plane for P4Runtime devices

What is ONOS?

- **Open Network Operating System (ONOS)**
- **Provides the control plane for a software-defined network**
  - Logically centralized remote controller
  - Provides APIs to make it easy to create apps to control a network
- **Focus on service provider for access/edge applications**
- **Runs as a distributed system across many servers**
  - For scalability, high-availability, and performance
- **Open source project**
  - Created by ON.Lab and currently hosted by the Linux Foundation
  - Active community: 8,152 commits and 276 authors in last 2 years

ONOS releases

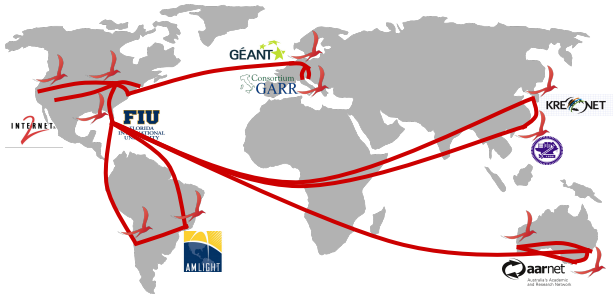
4-month release cycles

- **Avocet** (1.0.0) 2014-12
- …
- **Loon** (1.11.0) 2017-08 *(Initial P4Runtime support)*
- **Magpie** (1.12.0) 2017-12
- ***Nightingale** (1.13.0)* ***2018-04***
- ***Owl** (1.14.0)* ***2018-08***
- ***Peacock** (1.15.0)* ***2018-12*** *(latest release)*

Deployments

- **Access network for residential customers**
  - In production with a major US telecom provider
  - Edge leaf-spine fabric: ~10K clients, ~150K routes, ~1.5M flows
  - OpenFlow

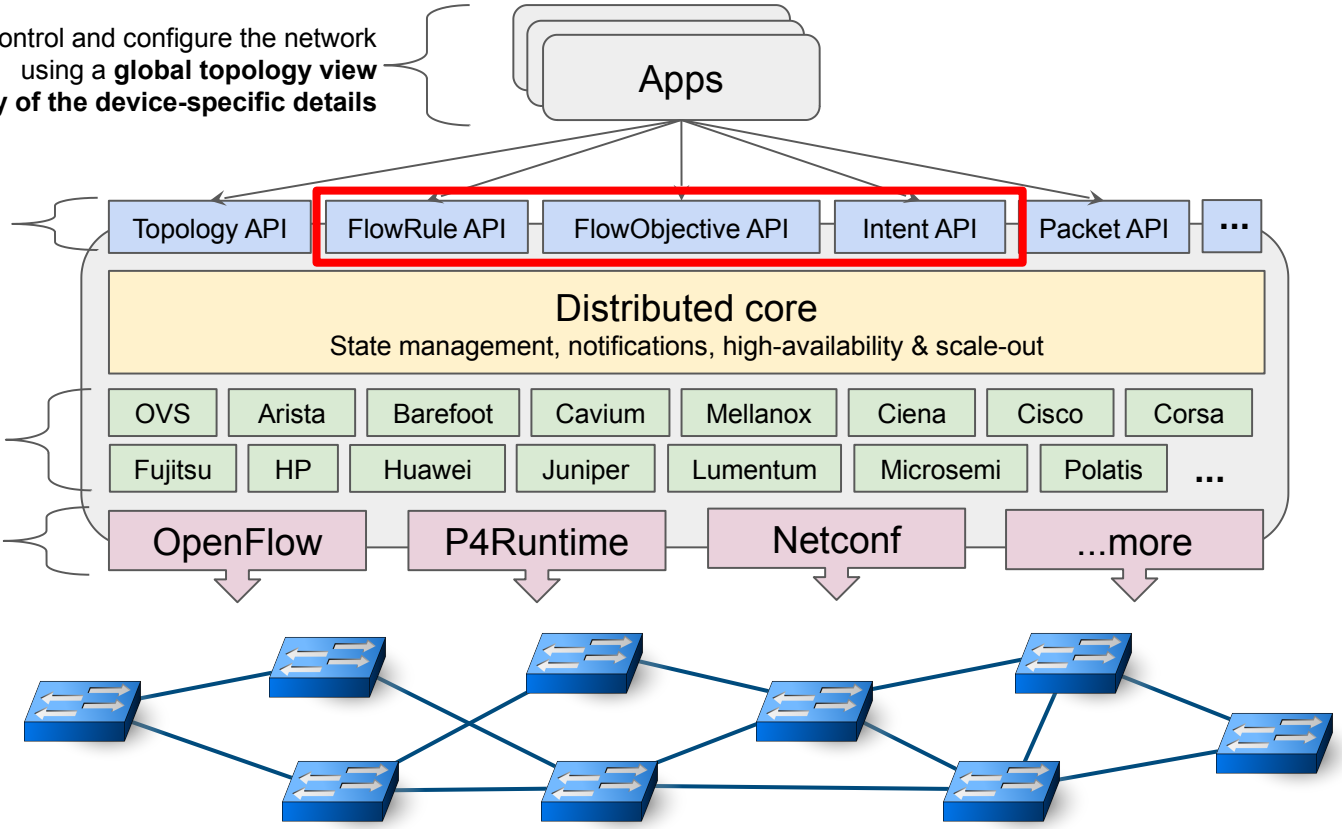
- **Research & Education**
  - SDN-IP, VPLS apps



- **SDN in Air-Traffic Management**
  - Safety-critical, ATM-grade deployment in Brazil (~22M km^2)
  - Radar relays, remote control towers, pilot voice, etc.
  - NetBroker from Frequentis developed on ONOS
  - Brown-field & OpenFlow

ONOS architecture

Control and configure the network
using a **global topology view**
and **independently of the device-specific details**

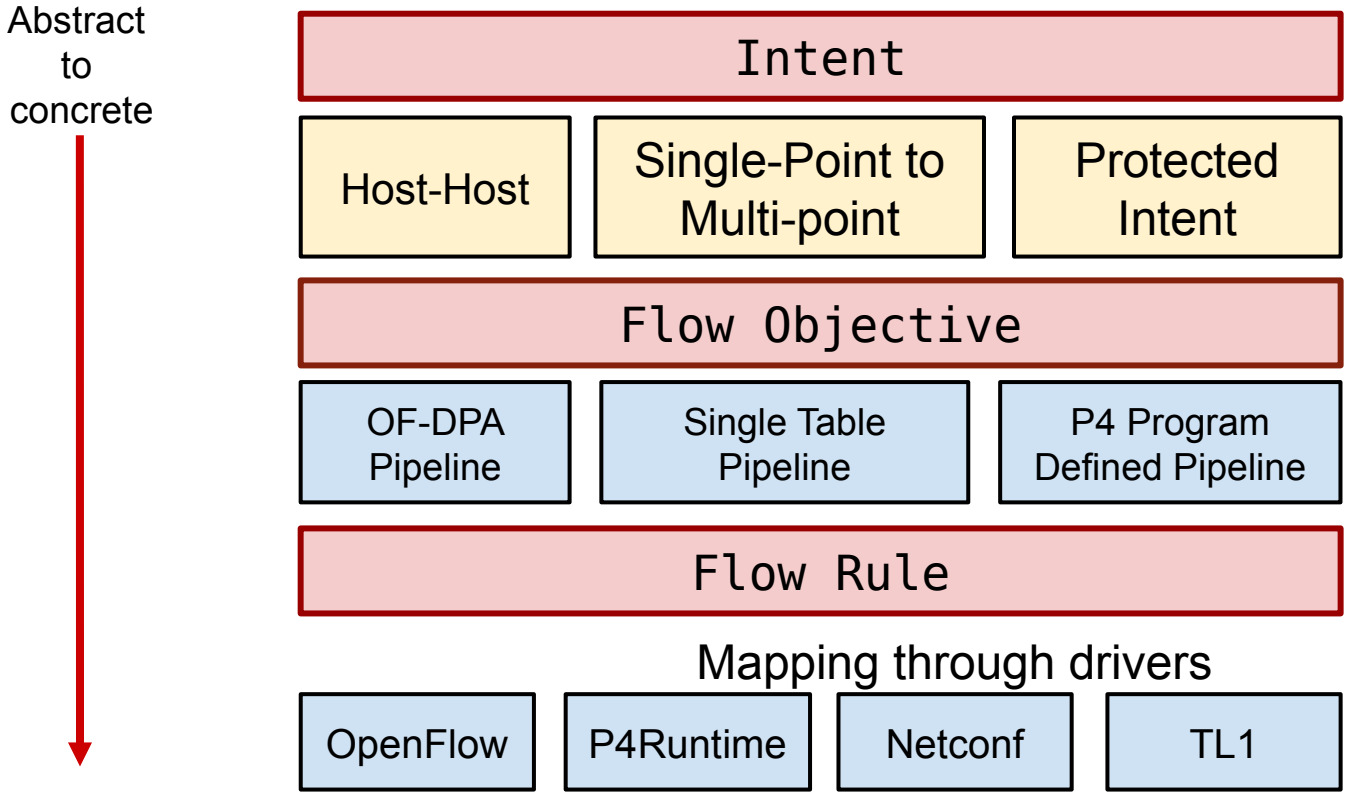


Northbound API
Device/protocol-agnostic
Java, REST, CLI, gRPC

Device driver
Allow device-specific variants
of standard protocols

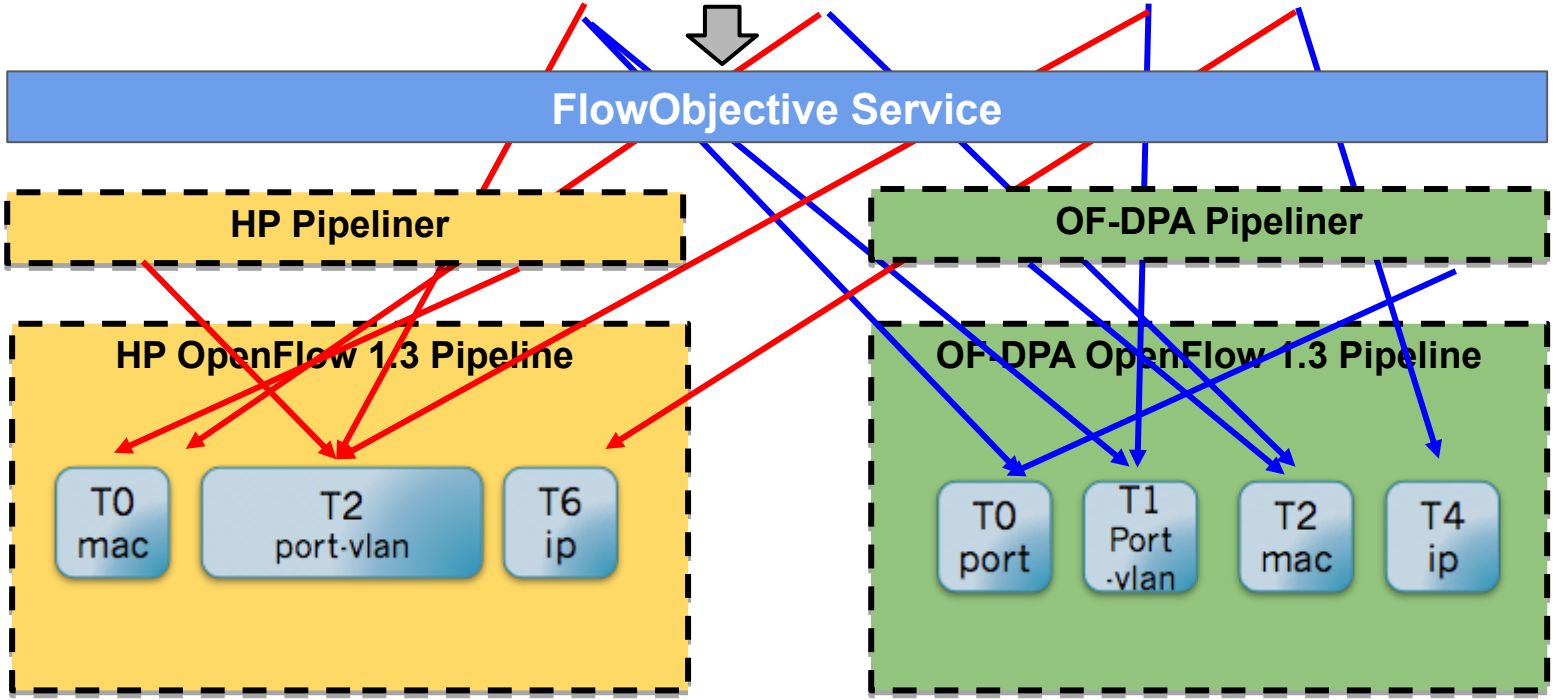
Shared protocol libraries

Network programming API



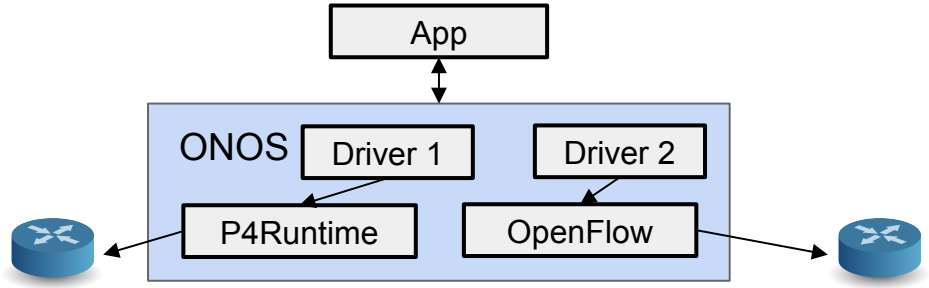
Flow objective example

Peering Router Match on Switch port, MAC address, VLAN, IP



Driver behaviors in ONOS

- **ONOS defines APIs to interact with device called “behaviors”**
  - DeviceDescriptionDiscovery → Read device information and ports
  - FlowRuleProgrammable → Write/read flow rules
  - PortStatisticsDiscovery → Statistics of device ports (e.g. packet/byte counters)
  - Pipeliner → FlowObjective-to-FlowRules mapping logic
  - Etc.
- **Behavior = Java interface**
- **Driver = collection of one or more behavior implementations**
  - Implementations use ONOS protocol libraries to interact with device



ONOS key takeaways

- **Apps are independent from switch control protocols**
  - Same app can work with OpenFlow and P4Runtime devices
- **Different network programming APIs**
  - FlowRule API – pipeline-dependent
  - FlowObjective API – pipeline-independent
    - Drivers translate 1 FlowObjective to many FlowRule
- **FlowObjective API enables application portability**
  - App using FlowObjectives can work with switches with different pipelines
  - For example, switches with different P4 programs

P4 and P4Runtime support in ONOS

P4 and P4Runtime support in ONOS

Goals:

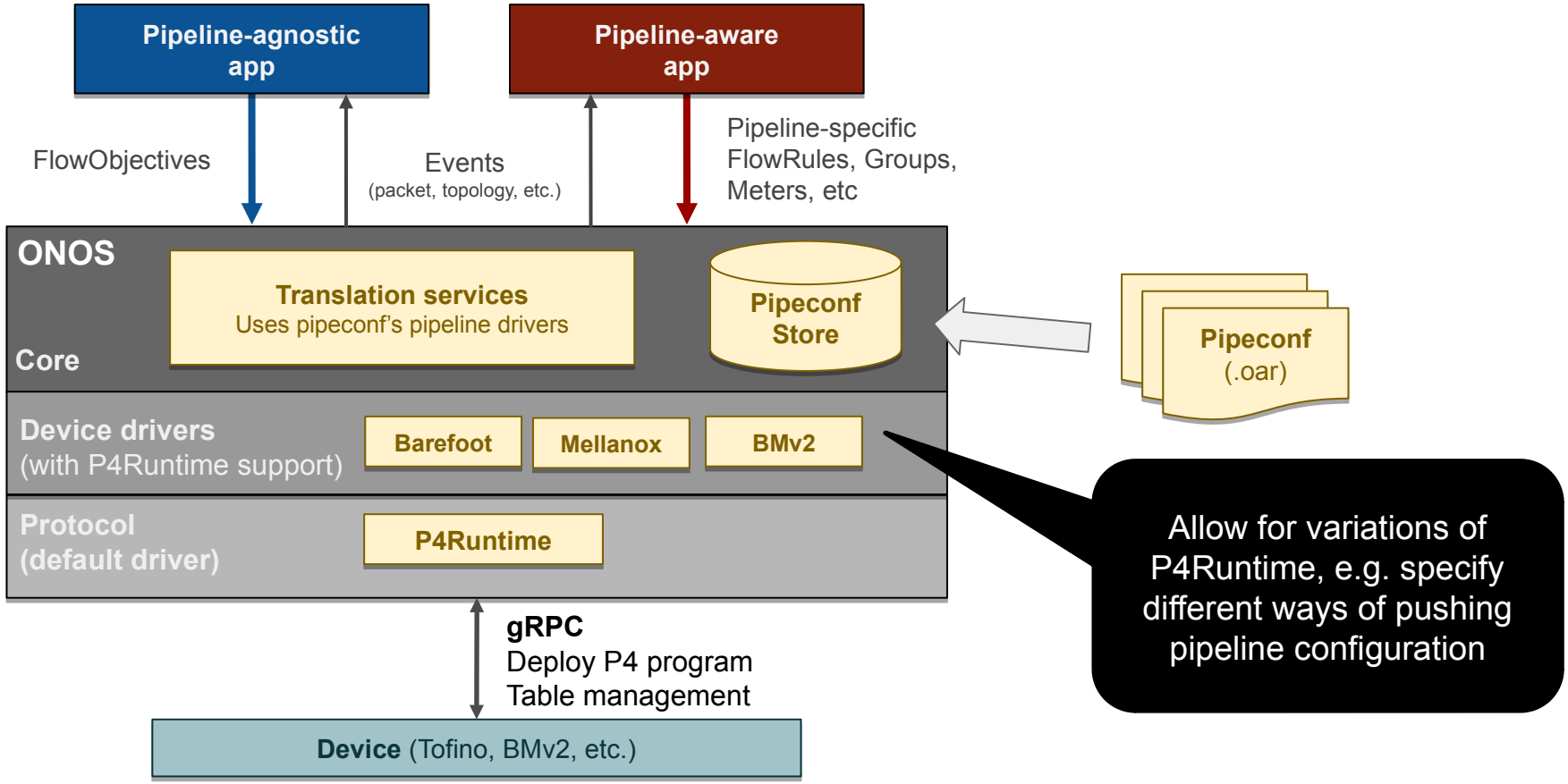
1. **Allow ONOS users to bring their own P4 program**
2. **Allow *existing* apps to control *any* P4 pipeline without changing the app**
    - i.e. enable app portability across many P4 pipelines
3. **Allow apps to control custom/new protocols as defined in the P4 program**

Pipeconf - Bring your own pipeline!

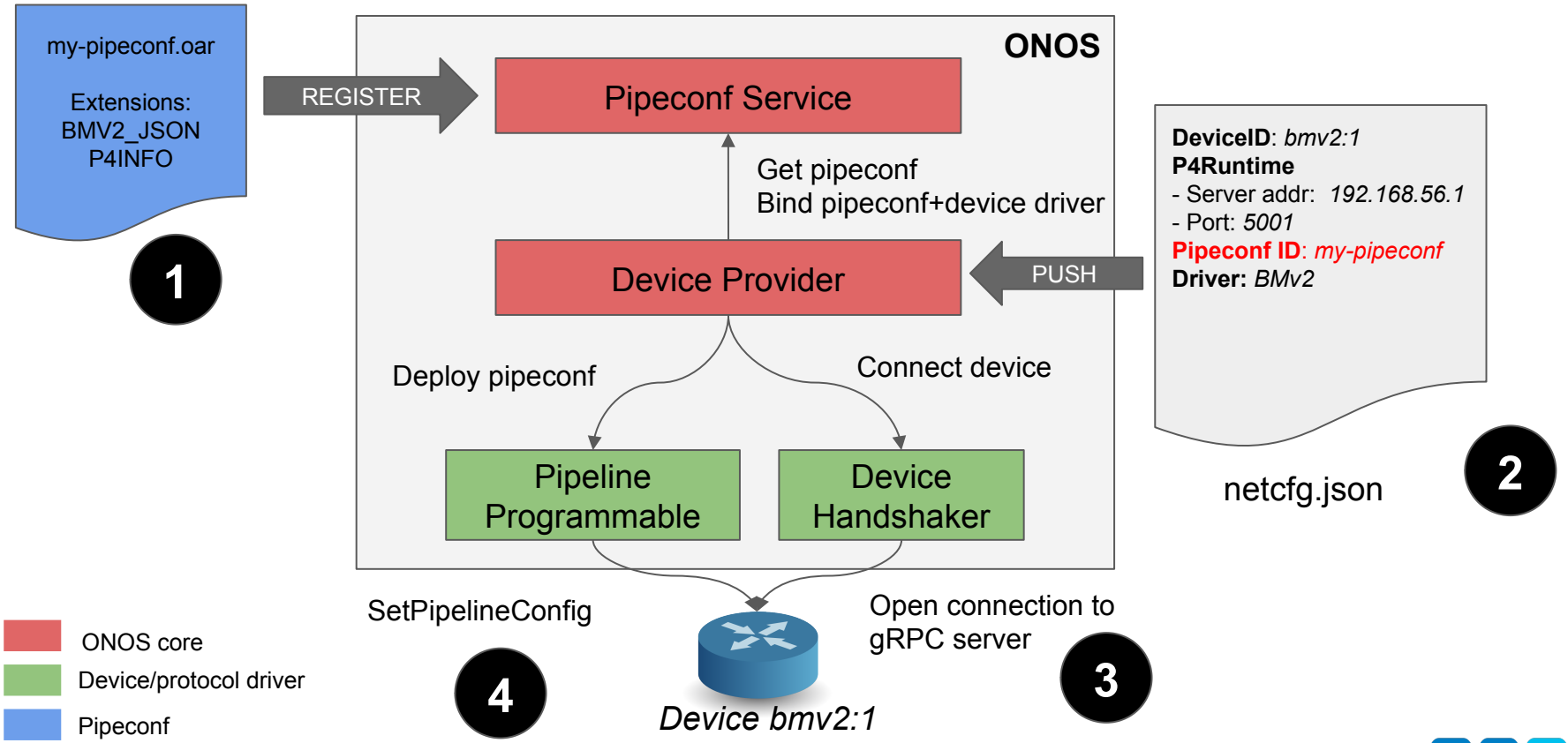
- **Package together everything necessary to let ONOS understand, control, and deploy an arbitrary pipeline**
- **Provided to ONOS as an app**
  - Can use `.oar` binary format for distribution

1. **Pipeline model**
   - Description of the pipeline understood by ONOS
   - Automatically derived from P4Info
2. **Target-specific binaries to deploy pipeline to device**
   - E.g. BMv2 JSON, Tofino binary, FPGA bitstream, etc.
3. **Pipeline-specific driver behaviors**
   - E.g. “Pipeliner” implementation: logic to map FlowObjectives to P4 pipeline

Pipeconf support in ONOS



Pipeconf deploy and device discovery



Flow operations

Define flow rules using *same headers/action names as in the P4 program*. E.g match on "hdr.my_protocol.my_field"

Pipeconf-based 3 phase translation:

1. Flow Objective → Flow Rule

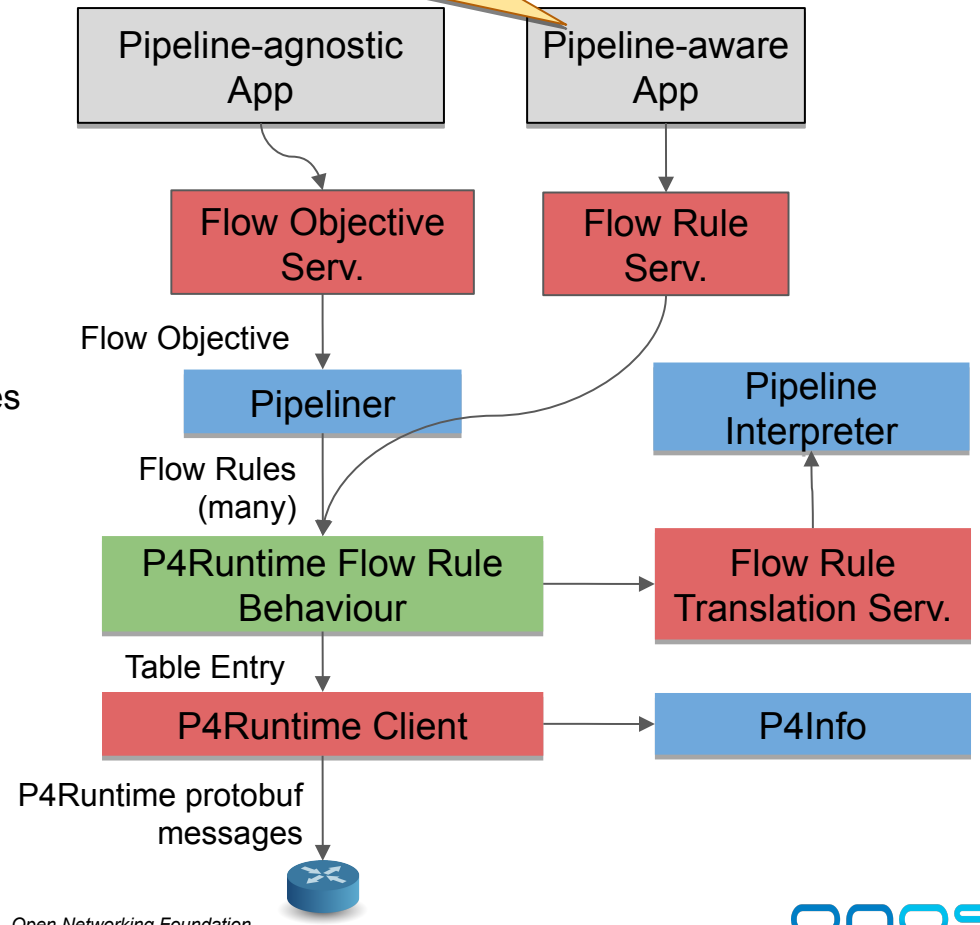
- Maps 1 flow objective to many flow rules

2. Flow Rule → Table entry

- Maps standard headers/actions to P4-defined ones
  E.g. ETH_DST→"hdr.ethernet.dst_addr"

3. Table Entry → P4Runtime message

- Maps P4 names to P4Info numeric IDs



ONOS Core
Device/protocol driver
Pipeconf driver

Pipeline Interpreter

- **Driver behavior**
  - Provides mapping between ONOS well-known headers/actions and P4 program-specific entities
- **Example: flow rule mapping**
  - Match
    - 1:1 mapping between ONOS known headers and P4 header names
    - E.g. ETH_DST → “ethernet.dst_addr” (name defined in P4 program)
  - Action
    - ONOS defines standard actions as in OpenFlow (output, set field, etc)
    - Problem: P4 allows only one action per table entry, ONOS many (as in OpenFlow)
    - E.g. header rewrite + output: 2 actions in ONOS, 1 action with 2 parameters in P4
    - How to map many actions to one? Need interpretation logic (i.e. Java code)!

P4Runtime support in ONOS 1.14 (Owl)

P4Runtime control entity	ONOS API
Table entry	Flow Rule Service, Flow Objective Service Intent Service
Packet-in/out	Packet Service
Action profile group/members, PRE multicast groups	Group Service
Meter	Meter Service (indirect meters only)
Counters	Flow Rule Service (direct counters) P4Runtime Client (indirect counters)
Pipeline Config	Pipeconf

Unsupported features - community help needed!
Parser value sets, registers, digests, clone sessions

ONOS+P4 workflow recap

- **Write P4 program and compile it**
  - Obtain P4Info and target-specific binaries to deploy on device
- **Create pipeconf**
  - Implement pipeline-specific driver behaviours (Java):
    - Pipeliner (optional - if you need FlowObjective mapping)
    - Pipeline Interpreter (to map ONOS known headers/actions to P4 program ones)
    - Other driver behaviors that depend on pipeline
- **Use existing pipeline-agnostic apps**
  - Apps that program the network using FlowObjectives
- **Or... write new pipeline-aware apps**
  - Apps can use same string names of tables, headers, and actions as in the P4 program
- **Enjoy!**