How to tell your plumbing what to do

Protocol Independent Forwarding

Nick McKeown

With thanks to many others…

ONF PIF: Curt Beckmann (Brocade), Justin Dustzadeh (Huawei), Yatish Kumar (Corsa), Ben Mack-Crane (Huawei), NM, Ben Pfaff (VMware), Jennifer Rexford (Princeton), Haoyu Song (Huawei), Dan Talayco (BFN).

P4: Pat Bosshart (BFN), Dan Daly (Intel), Glen Gibb (BFN), Martin Izzard (BFN), NM, Jennifer Rexford (Princeton), Cole Schlesinger (Princeton), Dan Talayco (BFN), Amin Vahdat (Google), George Varghese (MSFT), David Walker (Princeton).

Special shout out to Navindra Yadav

OpenFlow is a balancing act

A vendor-agnostic forwarding abstraction, balancing…

1. General match+action (hence TCAM model)
2. Fixed-function switch ASICs (hence 12 fixed fields)

Balance was necessary because

1. Big investments in ASICs and long development cycles requires clear, static guidelines
2. Fixed-function ASICs are faster and lower power (ASIC:NPU:CPU = 100:10:1)

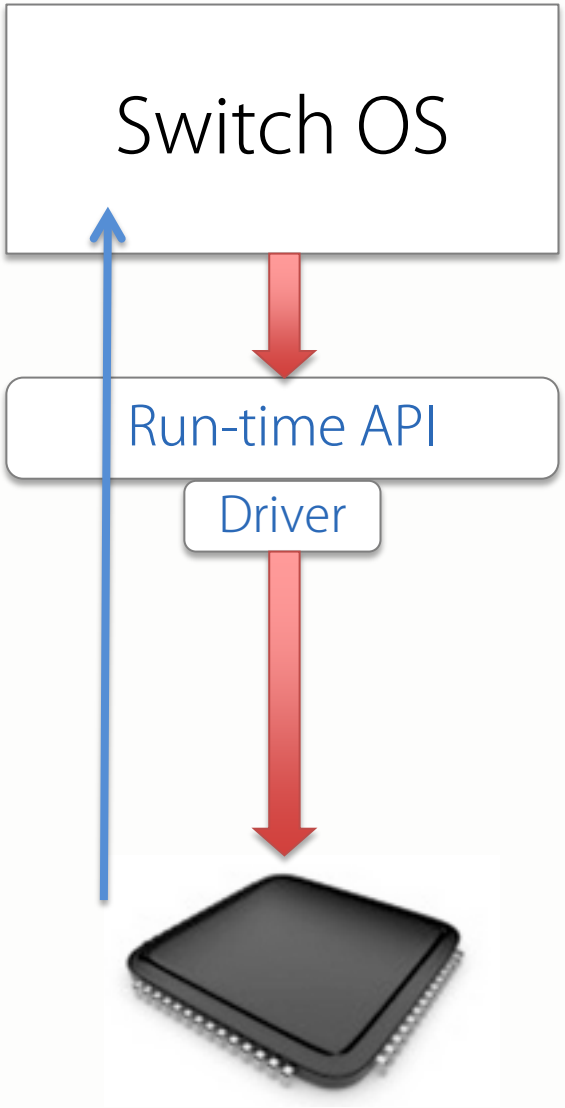
We have learned

We learned the importance of

- Multiple stages of match+action
- Not changing the specs too fast
- Extensibility
  1. So OpenFlow can be used in many networks
  2. So switch vendors can differentiate

Hence, the importance of TTPs.

"This is how I process packets"
(TTP)



Fixed-function ASIC

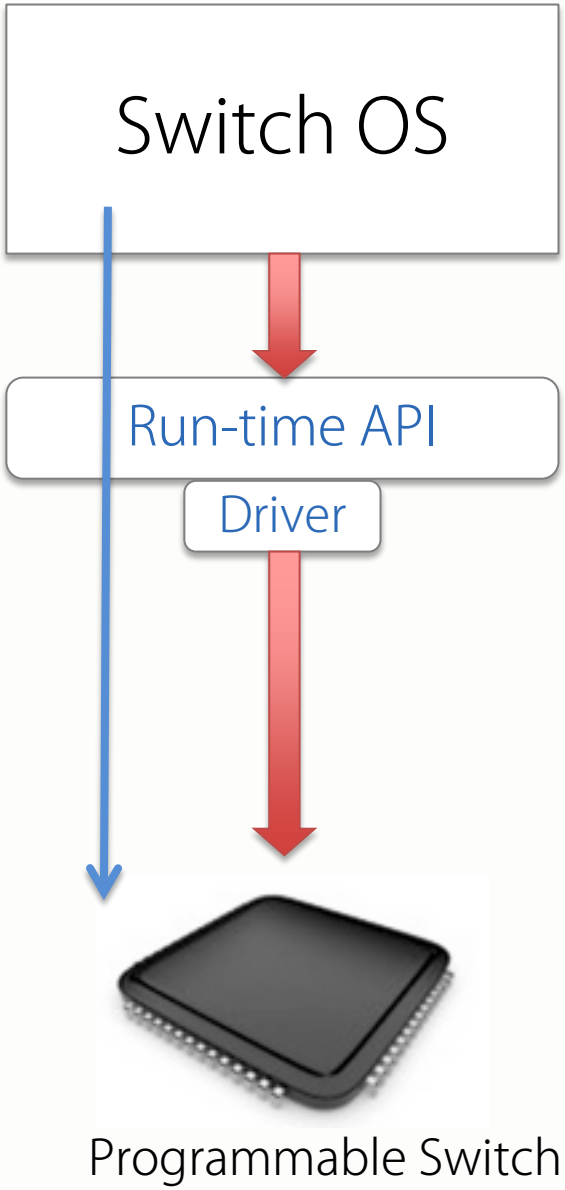
Programmable switches

Some switches are more programmable than fixed-function ASICs

- CPU (OVS, etc)
- NPU (Ezchip, Netronome etc)
- FPGA (Xilinx, Altera, Corsa)
- Flexible Match+Action ASICs
  (Intel Flexpipe, Cisco Doppler, Xpliant, …)

"Top-down" These switches let us tell them how to process packets.

"This is how the switch must process packets"
(PIF)



Why we call it
Protocol Independent Forwarding

Three phases

Phase 0. Initially, the switch does not know what a protocol is, or how to process packets (Protocol Independence).

Phase 1. We tell the switch how we want it to process packets (Configuration).

Phase 2. The switch runs (Run-time).

Three Goals

Protocol independence

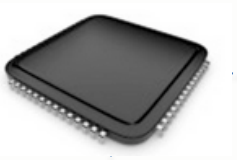
- Configure a packet parser
- Define a set of typed match+action tables

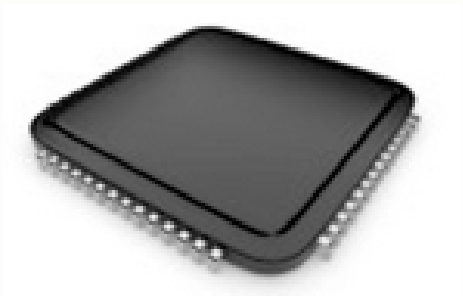
Target independence

- Program without knowledge of switch details
- Rely on compiler to configure the target switch

Reconfigurability

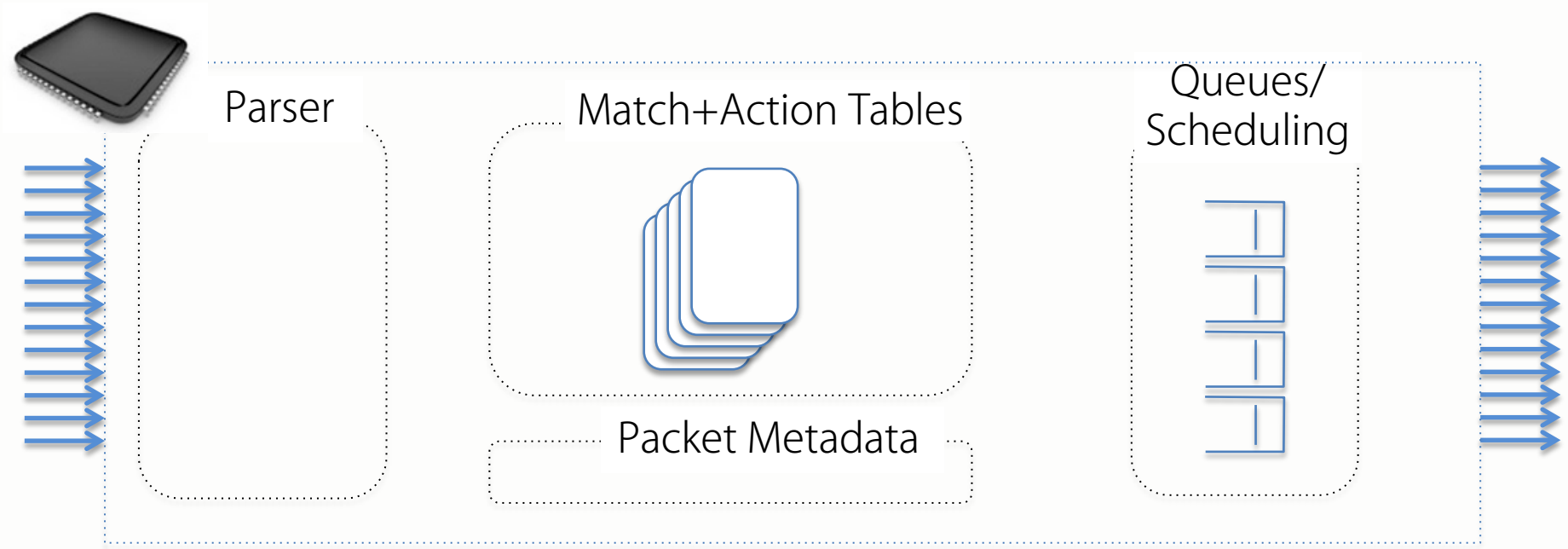
- Change parsing and processing in the field

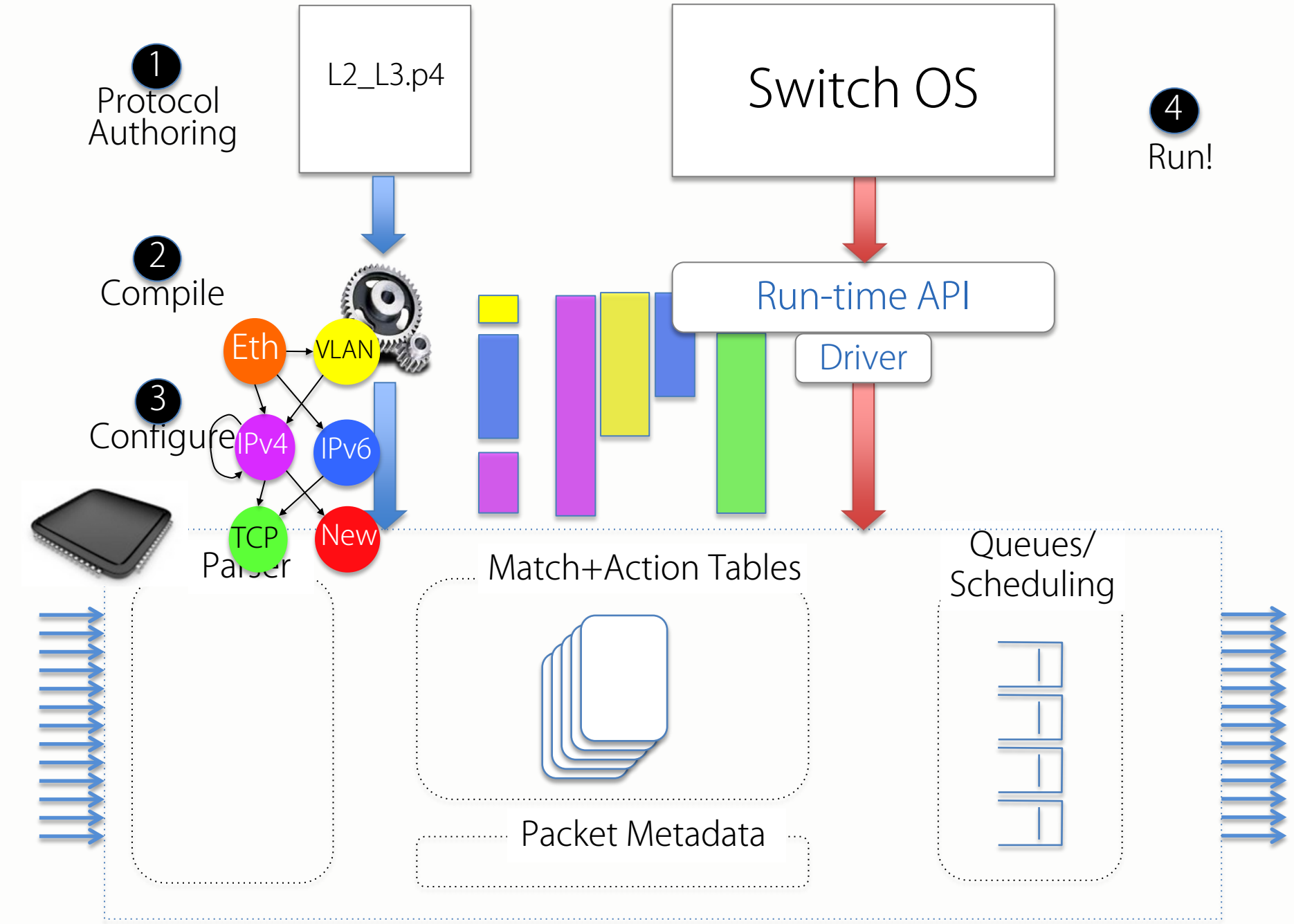


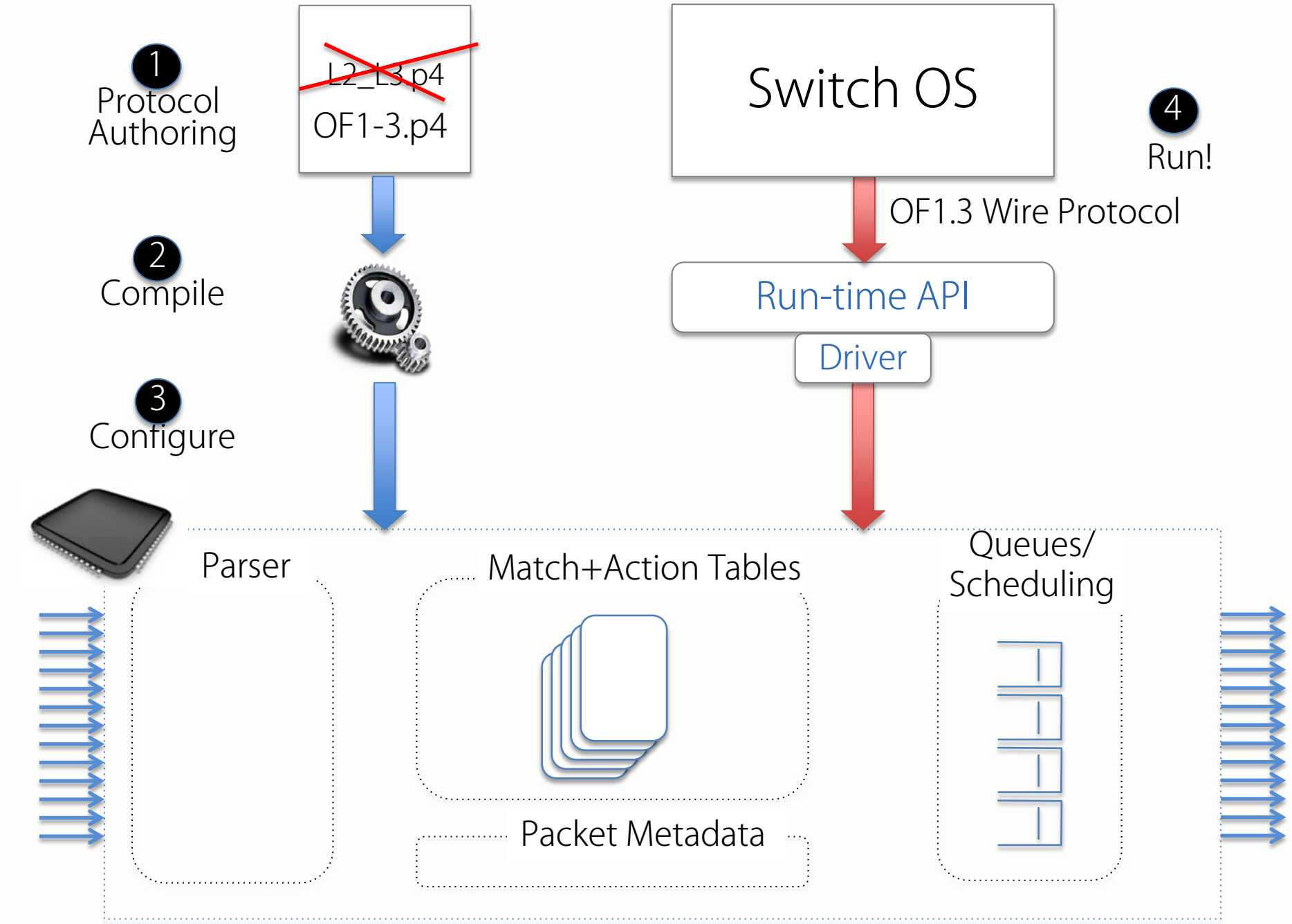


The Abstract Forwarding Model

Initially, a switch is unprogrammed
and does not know any protocols.




1
Protocol Authoring
L2_L3.p4
Switch OS
4
Run!
2
Compile
Run-time API
Driver
Eth
VLAN
3
Configure
IPv4
IPv6
TCP
New
Parser
Match+Action Tables
Queues/ Scheduling
Packet Metadata


1
Protocol Authoring
L2_L3.p4
OF1-3.p4
Switch OS
4
Run!
OF1.3 Wire Protocol
2
Compile
Run-time API
Driver
3
Configure
Parser
Match+Action Tables
Queues/ Scheduling
Packet Metadata

P4 Code Examples

- Headers and Fields
- The Parser
- Match+Action Tables
- Control flow

Headers and Fields

Header Fields: Ethernet

```
header_type ethernet_t {
  fields {
    dstAddr    : 48;
    srcAddr    : 48;
    etherType  : 16;
  }
}

/* Instance of eth header */
header ethernet_t first_ethernet;
```

Header Fields: VLAN

```
header_type vlan_tag_t {
  fields {
    pcp        : 3;
    cfi        : 1;
    vid        : 12;
    etherType  : 16;
  }
}

header vlan_tag_t vlan_tag[NUM];
```

Metadata

```
header_type standard_metadata_t {
  fields {
    ingress_port      : 32;
    packet_length     : 32;
    ingress_timestamp : 32;
    egress_spec       : 32;
    egress_port       : 32;
    egress_instance   : 32;
  }
}

metadata standard_metadata_t std_metadata;
```

The Parser

Parser: Ethernet

```
parser parse_ethernet {
  extract(ethernet);
  return switch(latest.etherType) {
        ETHERTYPE_VLAN : parse_vlan;
        ETHERTYPE_MPLS : parse_mpls;
        ETHERTYPE_IPV4 : parse_ipv4;
        ETHERTYPE_IPV6 : parse_ipv6;
        ETHERTYPE_ARP  : parse_arp_rarp;
        ETHERTYPE_RARP : parse_arp_rarp;
  }
}
```

Parser: IPv4

```
parser parse_ipv4 {
  extract(ethernet);
  return switch(latest.etherType) {
        PROTO_TCP : parse_tcp;
        PROTO_UDP : parse_udp;
        ...
  }
}
```

Match+Action Tables

Specifies

- Which fields to examine in each packet
- Actions that may be applied (by rule)
- Table size (optional)

Match+Action Table: VLAN

```
table port_vlan {
    reads {
        std_metadata.ingress_port : exact;
        vlan_tag[OUTER_VLAN].vid : exact;
    }
    actions {
        drop, ing_lif_extract;
    }
    size 16384;
}
```

Match+Action Table: Unicast RPF

```
table urpf_check {
    reads {
        routing_metadata.bd : ternary;
        ipv4.dstAddr : ternary;
    }
    actions {
        urpf_clear, urpf_set;
    }
}
```

Actions

Built from primitives

- modify field (packet header or metadata)
- add/remove header
- clone/recirculate
- counter/meter/stateful memory operations

Parallel semantics

Actions: LIF Extract

```
/* Ingress logical interface setup */
action ingress_lif_extract(i_lif, bd, vrf, v4term, v6term, igmp_snoop) {
  modify_field(route_md.i_lif, i_lif);
  modify_field(route_md.bd, bd);
  modify_field(route_md.vrf, vrf);
  modify_field(route_md.ipv4_term, v4term, 0x1);
  modify_field(route_md.ipv6_term, v6term, 0x1);
  modify_field(route_md.igmp_snoop, igmp_snoop, 0x1);
}
```

Control Flow

Control Flow: Ingress

```
control ingress {
    apply_table(port);
    apply_table(bcast_storm);
    apply_table(ip_sourceguard);
    if (valid(vlan_tag[0])) {
        apply_table(port_vlan);
    }
    apply_table(bridge_domain);
    if (valid(mpls_bos)) {
        apply_table(mpls_label);
    }
    retrieve_tunnel_vni();
    if (valid(vxlan) or valid(genv) or valid(nvgre)) {
        apply_table(dest_vtep);
        apply_table(src_vtep);
    }
    . . . .
}
```

Use Cases

- P4 can be used to describe switches, routers, firewalls, gateways, load-balancers, etc.
- Mostly-stateless applications
- “P4 is for plumbers”

P4

P4: Programming Protocol-Independent Packet Processors

ACM CCR. Volume 44, Issue #3 (July 2014)

Pat Bosshart, Glen Gibb, Martin Izzard, and Dan Talayco (Barefoot Networks), Dan Daly (Intel), Nick McKeown (Stanford), Cole Schlesinger and David Walker (Princeton), Amin Vahdat (Google), and George Varghese (Microsoft)

www.p4.org

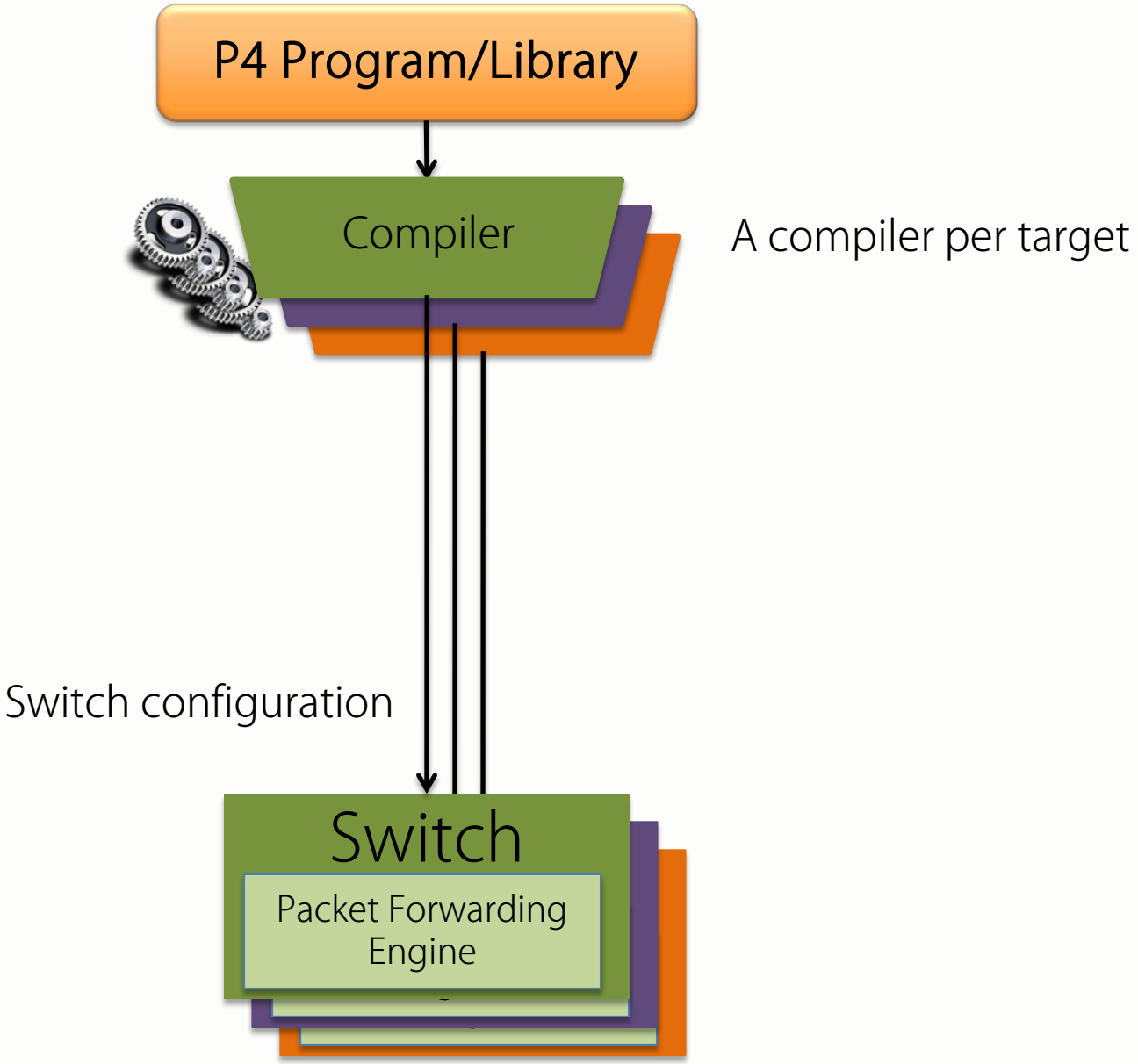
How to contribute to P4.org

Contribute to the P4 language spec

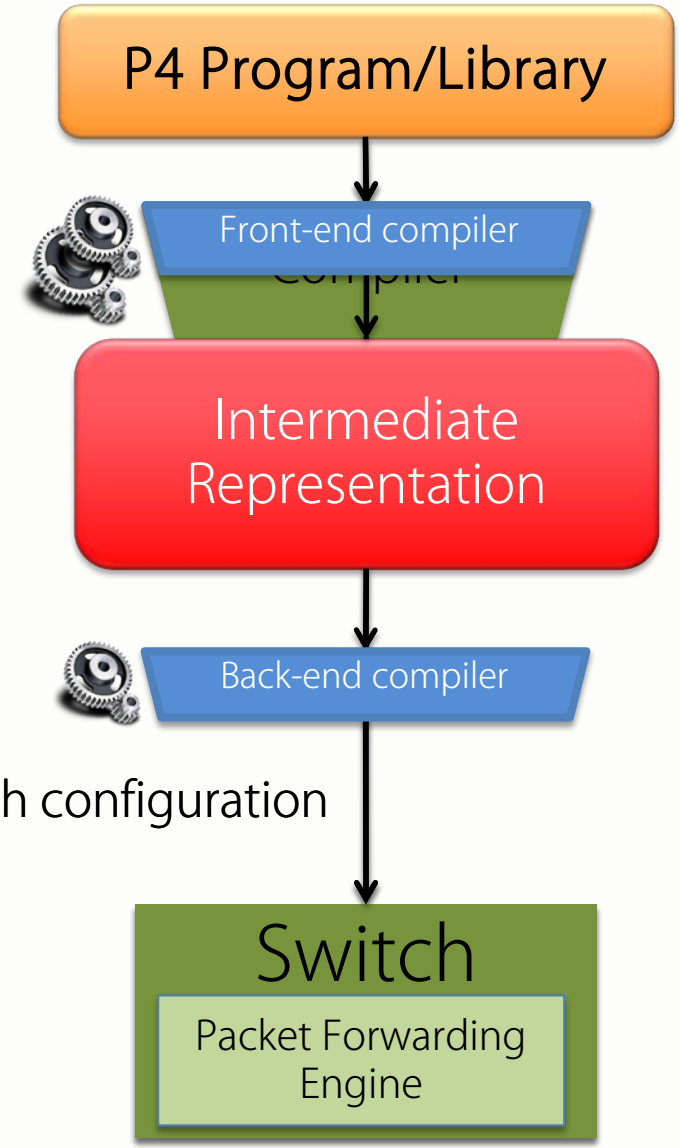
Contribute a compiler to your target

Contribute new tools (e.g. debugger)

The P4 View of the World



The P4 View of the World



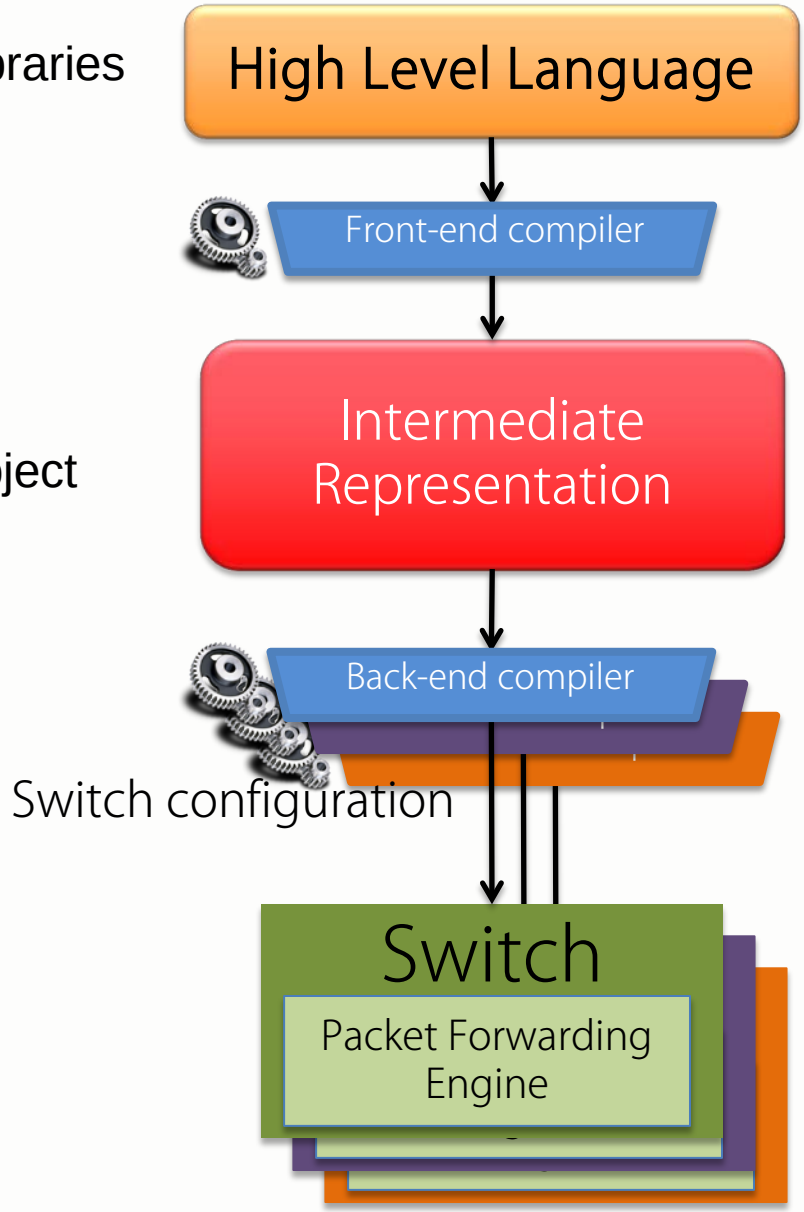
A compiler per target

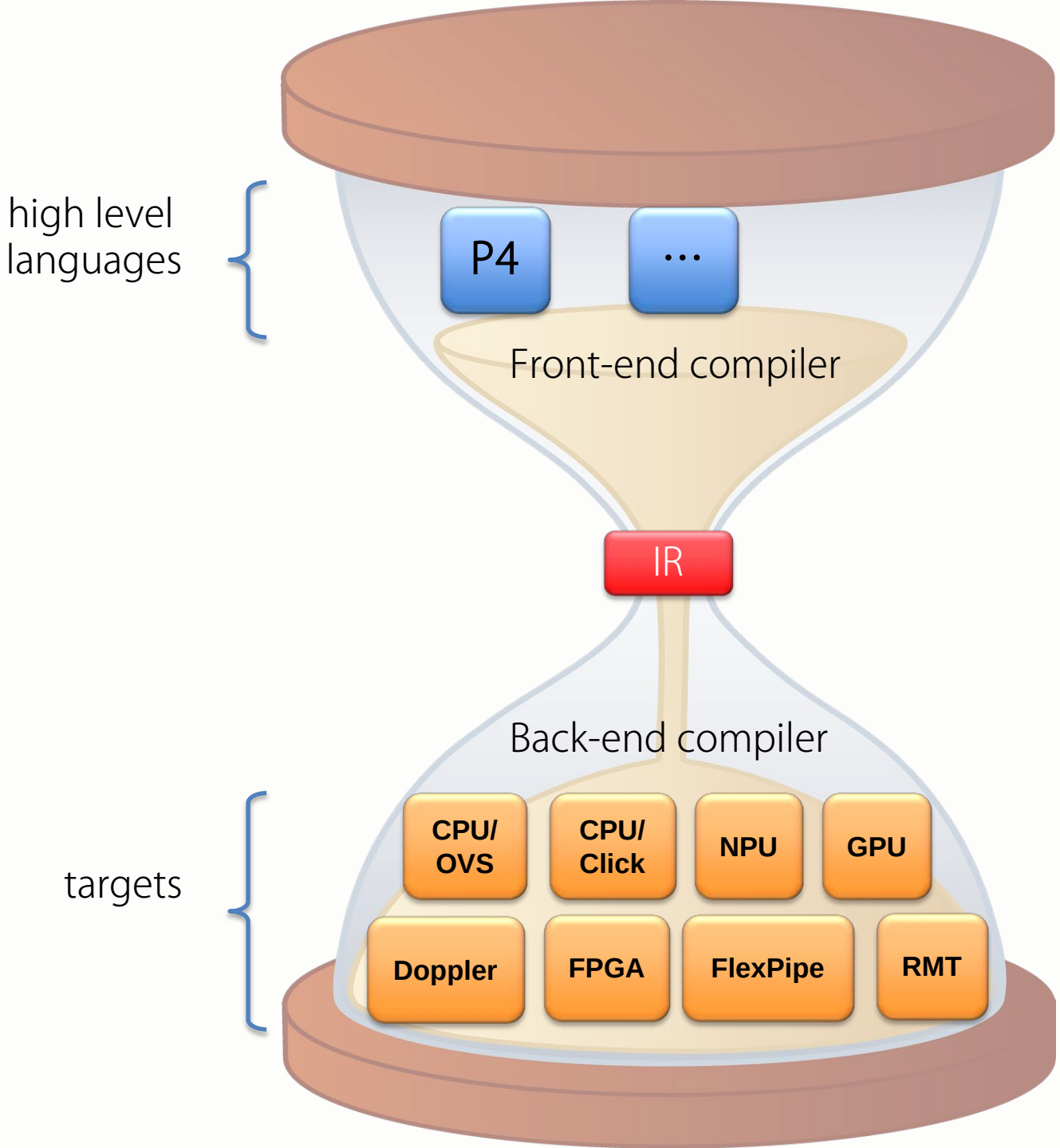
New ONF
open-source project

Many languages, many targets

e.g. P4 program/libraries

New ONF
open-source project




high level languages
P4
…
Front-end compiler
IR
Back-end compiler
targets
CPU/ OVS
CPU/ Click
NPU
GPU
Doppler
FPGA
FlexPipe
RMT

A Fourth Goal

Protocol independence

Target independence

Reconfigurability

Language independence

Goals of an IR

- Allow new languages above
- Compile to many devices below
- Uniquely define how packets are to be processed
- Keep it simple
- Action primitives: minimal “Plumbing Instruction Set (IS)”
- Later: Could add DPI IS, Security IS

Components of IR

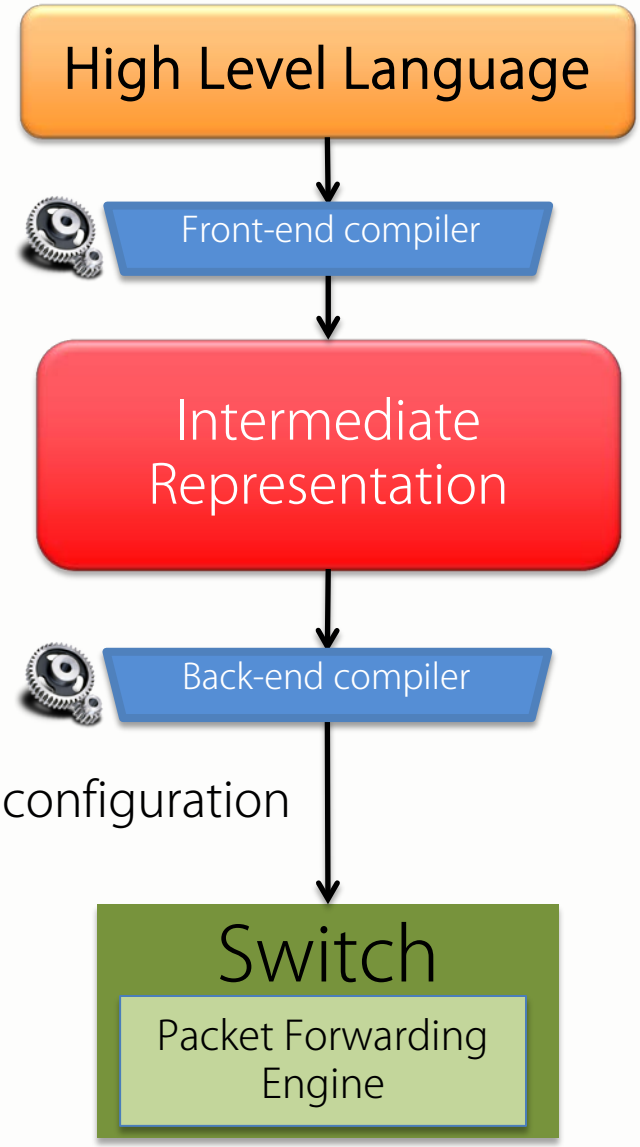
1. Plumbing Instruction Set
2. Parse graph
3. Table graph, edges annotated with actions.

Optional: Table graph, annotated with dependencies to allow back-end compiler to exploit concurrency.

In summary…

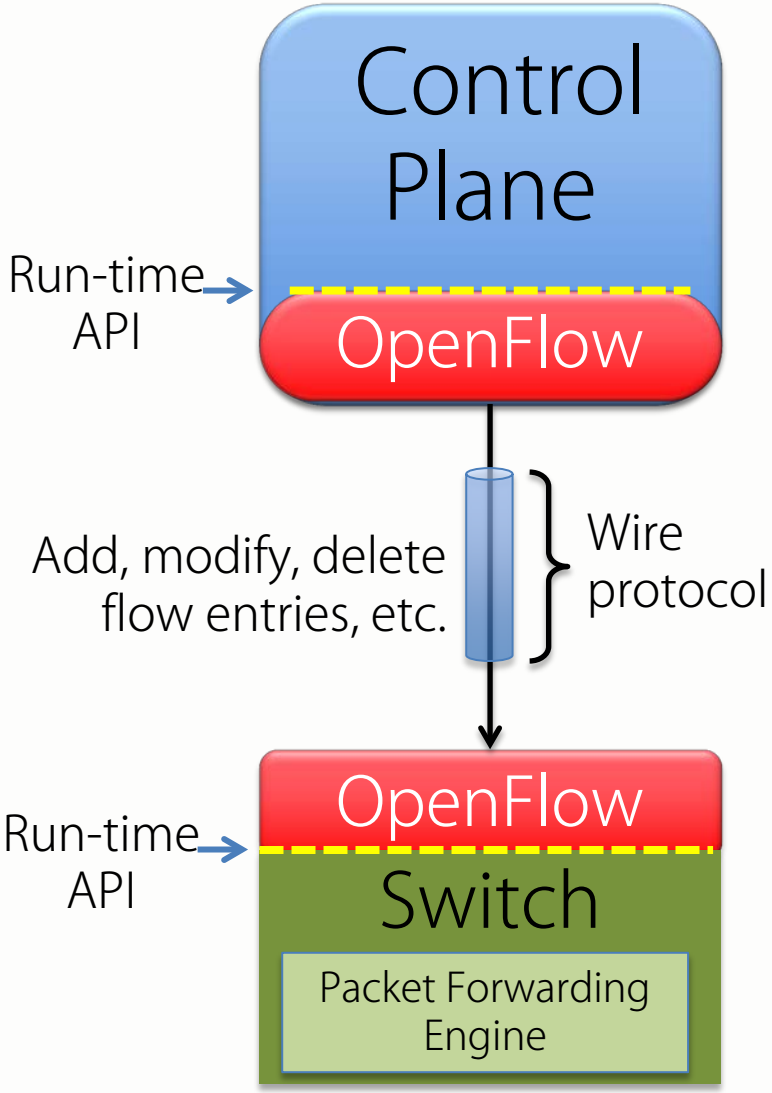
Step 1. Configuration

e.g. P4 program/libraries



New ONF
open-source project

Stage 2. Run-time



Contribute

ONF PIF Open-Source Project

- Approved by ONF Board
- Apache 2.0, open to all
- ONF needs to put open-source governance/license in place (will take a few weeks)

Contribute code to IR

Contribute back-end compiler:
(e.g. to OVS, FPGA, NPU, ASIC, …)

"So, what did the Plumber think of your suggestions dear...?"