Protocol Independent Forwarding

ONF PIF Project

2015-02-13

Underlined text records discussion

Agenda

- Overview (30 min)
  - Motivation and terminology
  - Projects and responsibilities
- Tour of AIR-IRI (30 min)
  - Sample datapath program
  - Overview of interpreter + infrastructure
- Work items and infrastructure (40 min)
  - Summary of work so far
  - Future work areas - IR development (interpreter+tools), IR samples, runtime interacting with IR
  - Software infrastructure and licensing
- Next steps - how to participate (20 min)

Why OpenFlow Next Generation / PIF etc?

- Vision - refactor architecture / design (*technology* push)
  - Support new approaches, e.g. more flexibly configured datapath => *Protocol Independent Forwarding*
  - Support new operations, e.g. stateful, delegated to switch, nested encapsulation
  - Optimize capacity, throughput - e.g. simpler, more parallel

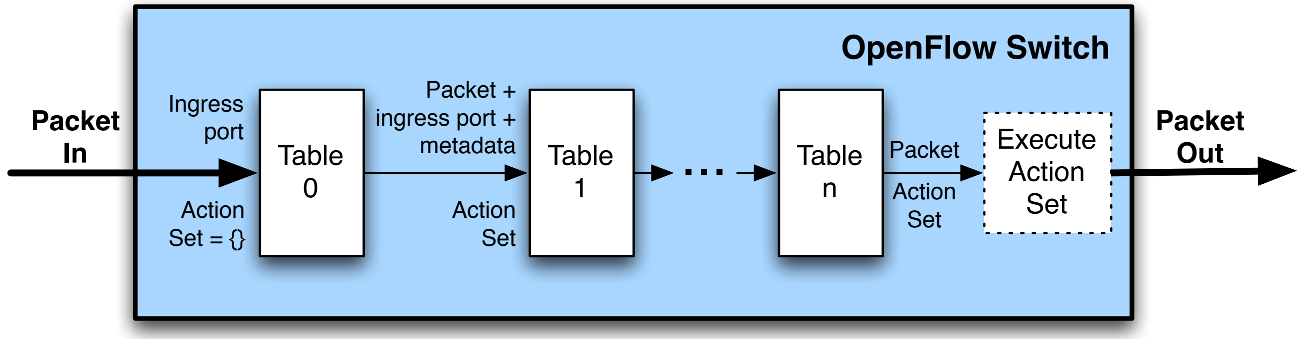
- Optimize processes / lifecycles (facilitate *ecosystem*)
  - Standards creation process (help ONF working group participants)
  - Development and deployment processes (help switch, controller, app vendors and users / operators)
    - Avoid cross product of controllers / apps and switch types

- Address specific requirements (*market* pull)
  - Support new use cases / market segments e.g. L4-L7 services / NFV
  - Refinement of capabilities / improved performance etc. for existing use cases

Interpretations of Protocol Independence

- Restructuring the existing OpenFlow *specification* - result:
  - Modular specification
  - Core or base does not refer to protocols
  - Each protocol / layer (e.g. Ethernet or IP) documented in its own add on
  - Result can express the same semantics as e.g. OpenFlow 1.3 - no impact on switches / controllers
  - Easier to add support for protocols in future - just write an add on module
- Introducing support for “*user defined” protocols* - result:
  - Match fields not limited to existing set of 40-ish OXMs
  - “Users” (vendors / operators…) describe new field as length + offset from already defined field etc.
  - Can be accomplished by extending existing OpenFlow 1.x specification
- More ambitious Protocol Independent Forwarding project - result:
  - More than just ability to define new protocol fields
  - Enables defining arrangement of OpenFlow pipeline in a more flexible way
  - “Datapath program” (in effect forwarding model) describes pipeline arrangement (matching tables, actions, QoS TM elements etc.) and behavior
  - Toolchains (compilers etc) enable configuring switches to “run” these “programs”

Elements in PIF World - New + Improved

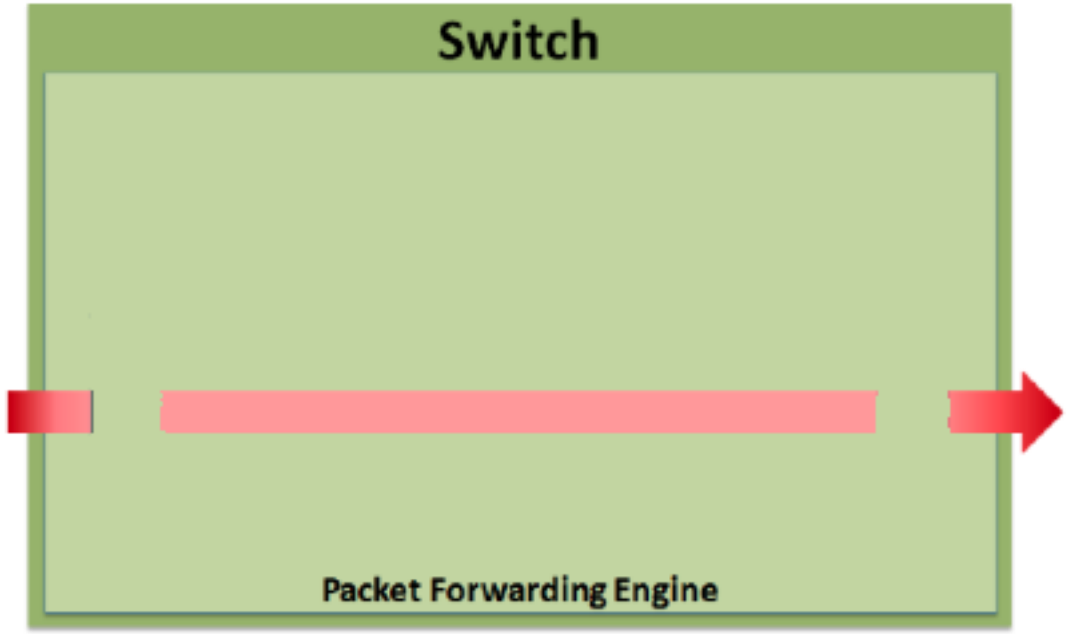


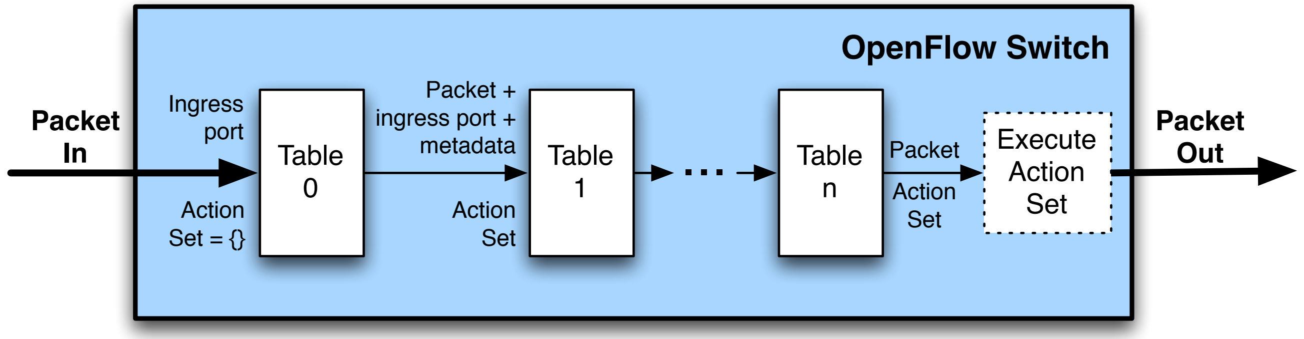
- Predefined Protocol Forwarding
  - OpenFlow specification defines protocols / fields, match/action behavior, overall control flow (tables can influence)
  - Set of supported protocols fixed by implementation

Elements in PIF World - New + Improved

- Config time

Datapath Program *in HL lang*

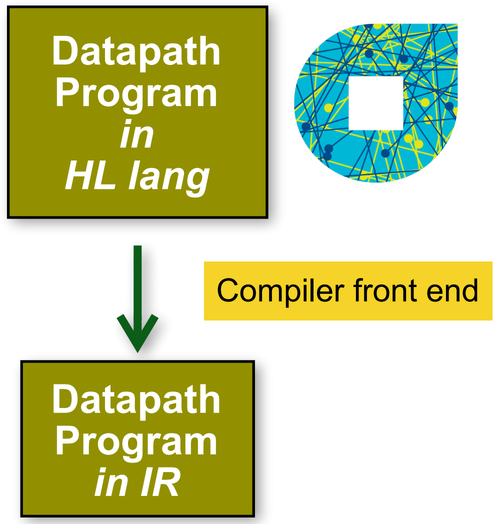


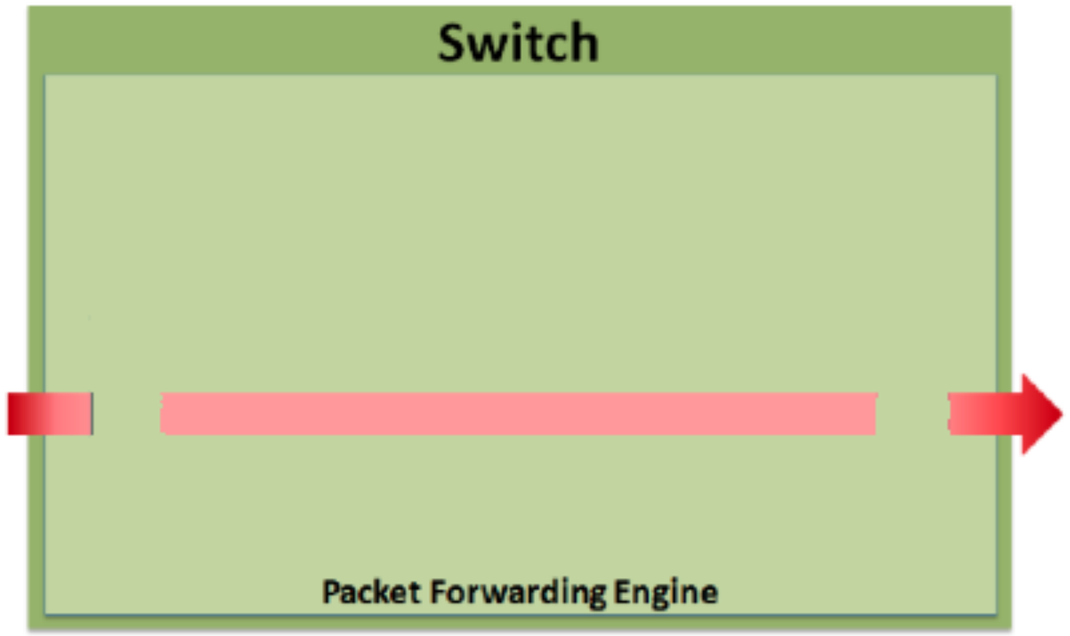


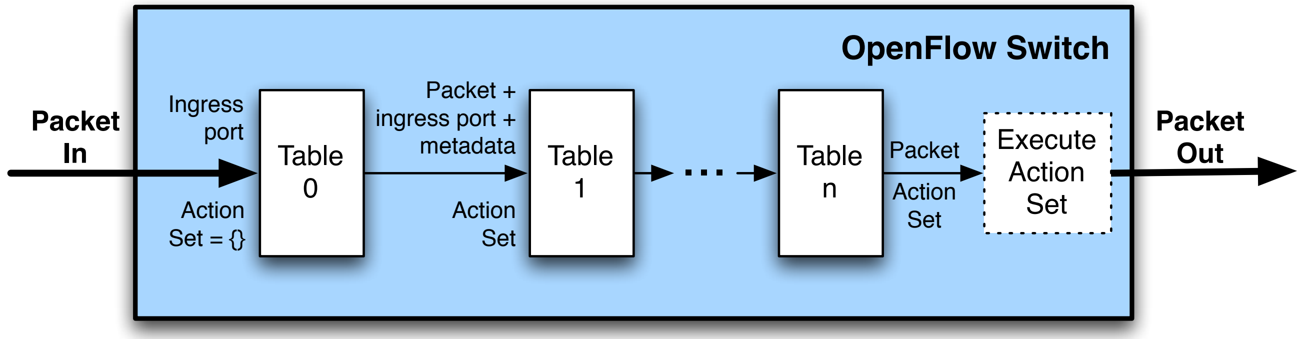
- Predefined Protocol Forwarding
  - OpenFlow specification defines protocols / fields, match/action behavior, overall control flow (tables can influence)
  - Set of supported protocols fixed by implementation

Elements in PIF World - New + Improved

- Config time

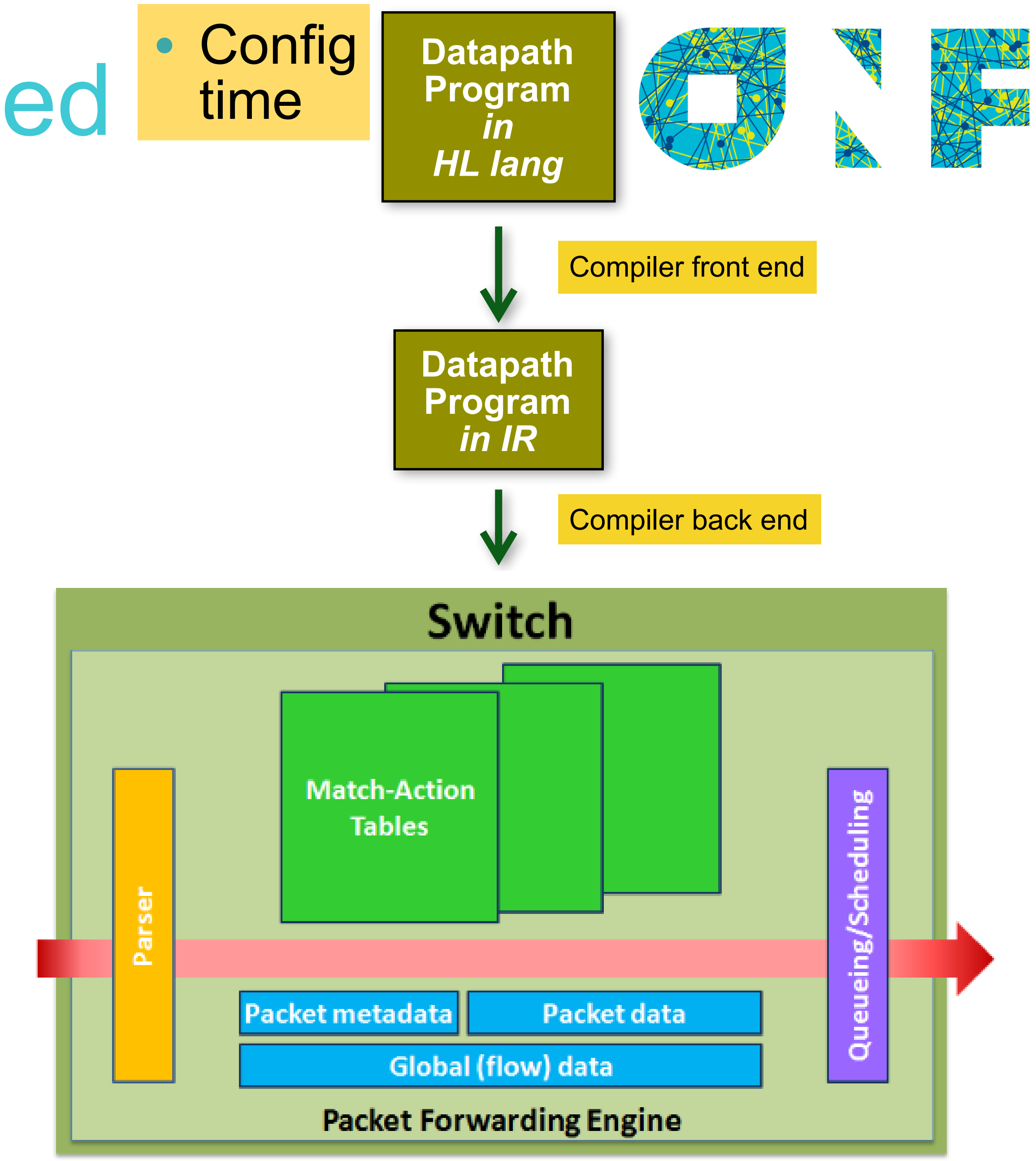


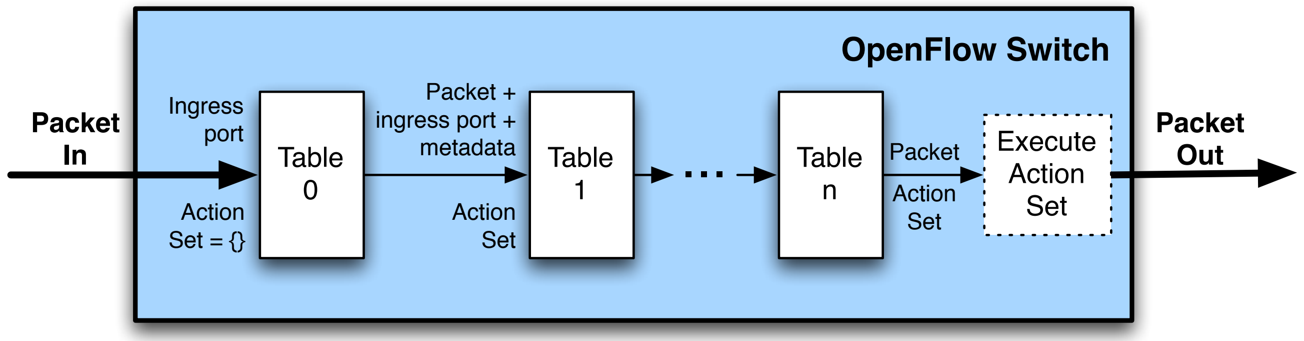




- Predefined Protocol Forwarding
  - OpenFlow specification defines protocols / fields, match/action behavior, overall control flow (tables can influence)
  - Set of supported protocols fixed by implementation

Elements in PIF World - New + Improved

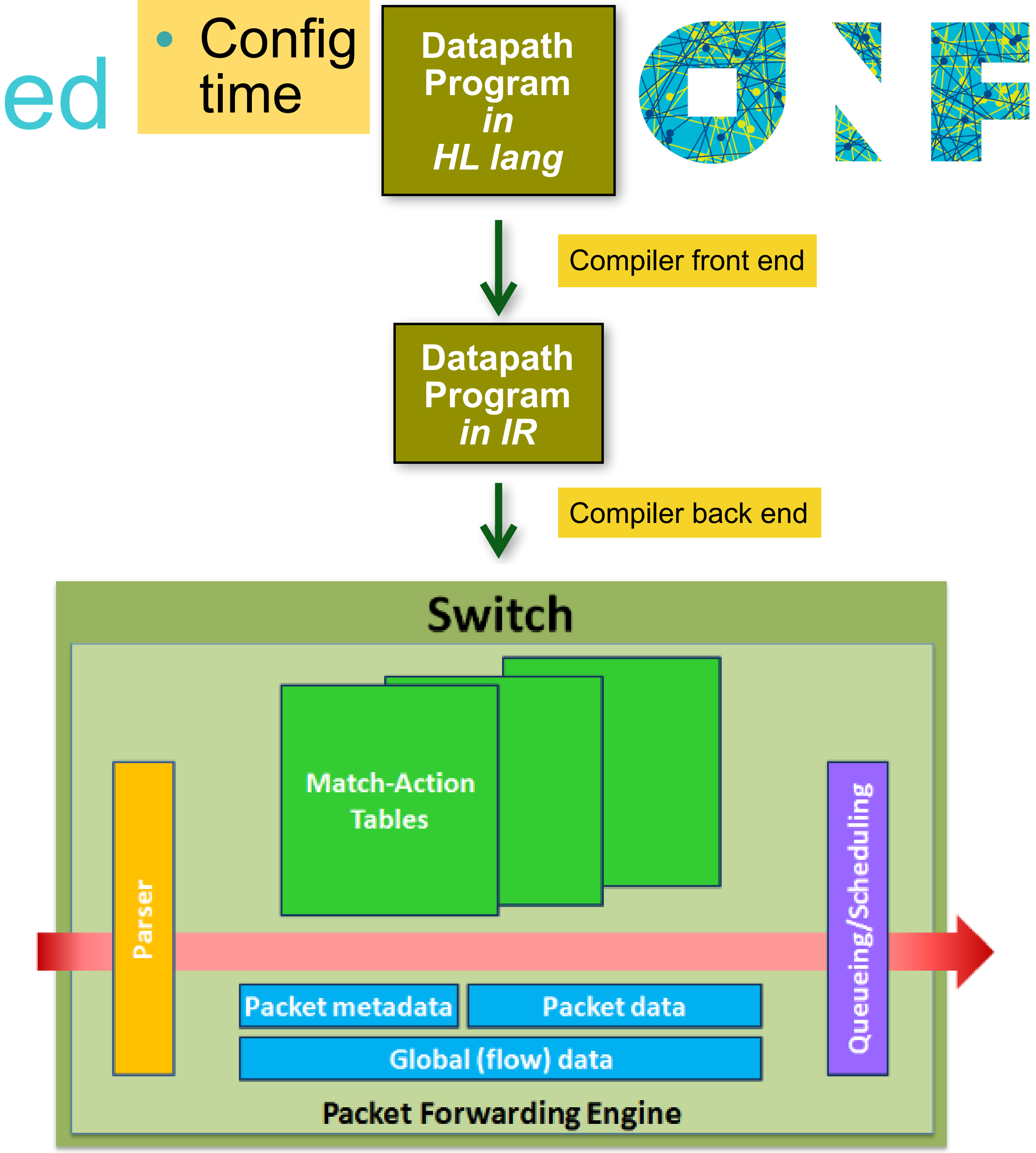


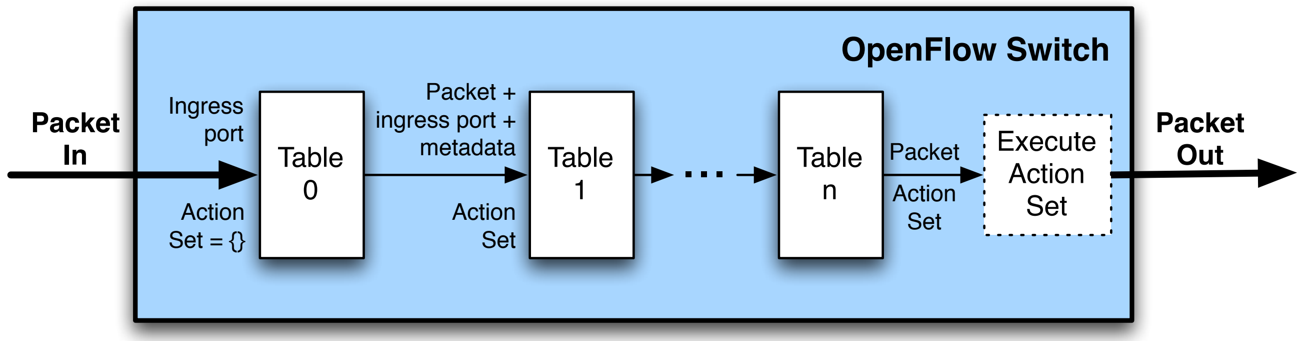


- Predefined Protocol Forwarding
  - OpenFlow specification defines protocols / fields, match/action behavior, overall control flow (tables can influence)
  - Set of supported protocols fixed by implementation

Elements in PIF World - New + Improved

- Config time



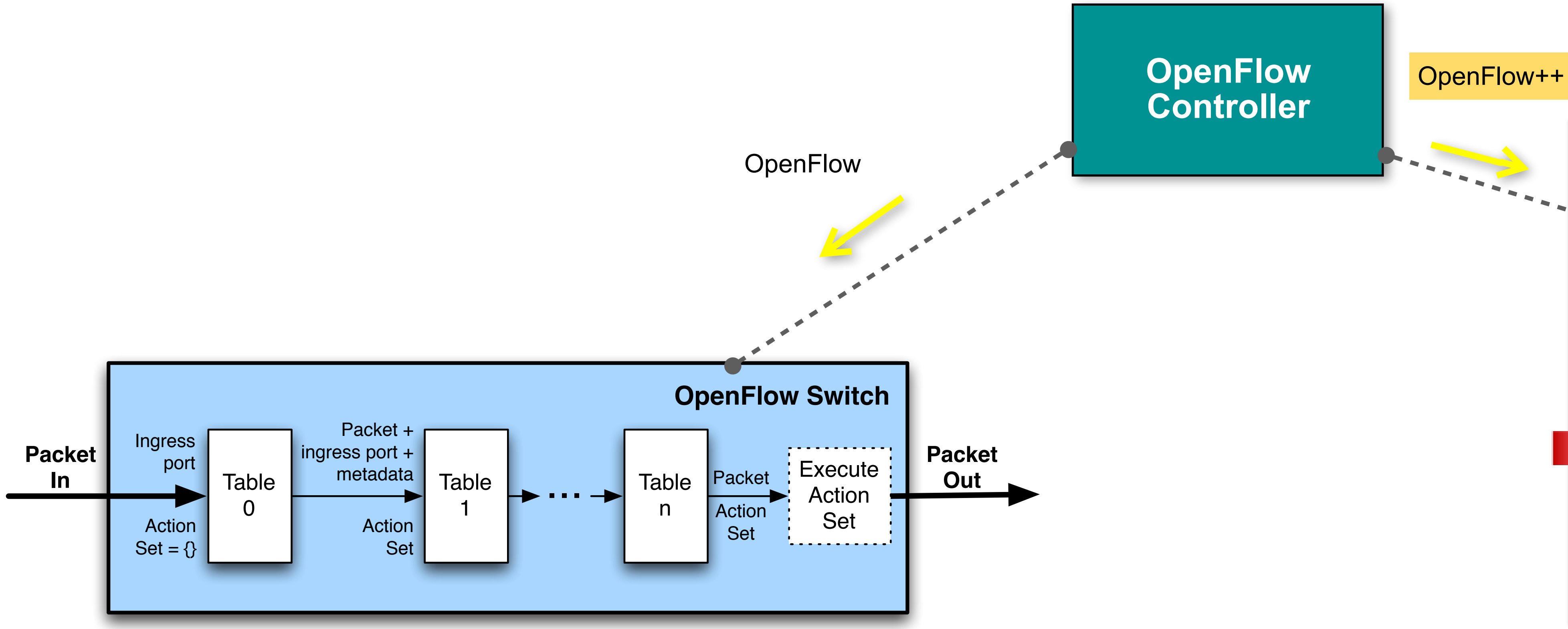


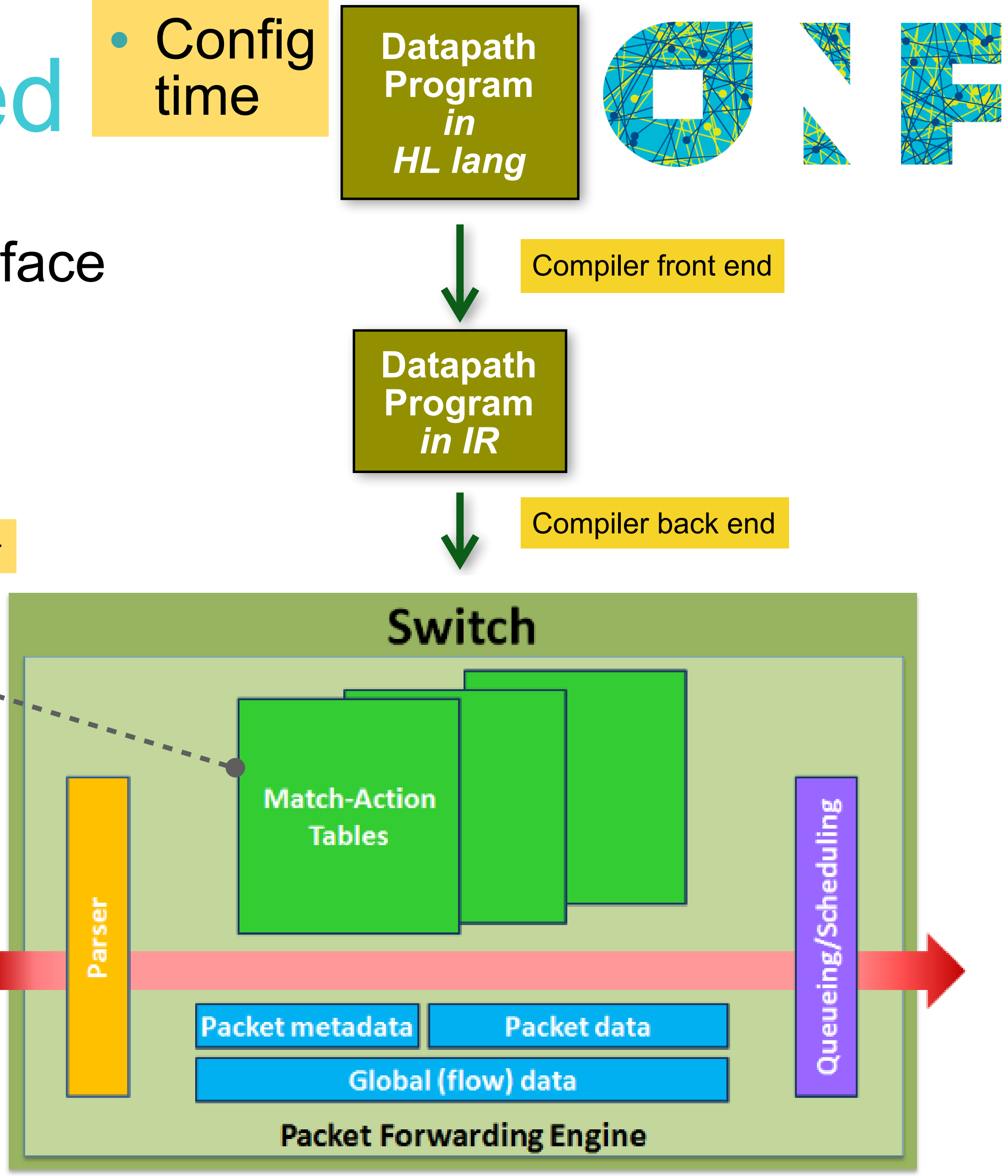
- Predefined Protocol Forwarding
  - OpenFlow specification defines protocols / fields, match/action behavior, overall control flow (tables can influence)
  - Set of supported protocols fixed by implementation

- Protocol Independent Forwarding (PIF) Configured ("Programmed") Datapath
  - Programs in language(s) describe datapath
    - Parse tree => protocol independent
    - Match/action tables (control flow arranges table sequence)
    - Packet metadata, per table or global state
    - QoS

Elements in PIF World - New + Improved

- Either way - need “southbound” interface for run-time interaction with switch...
  - Populate tables, receive statistics + events



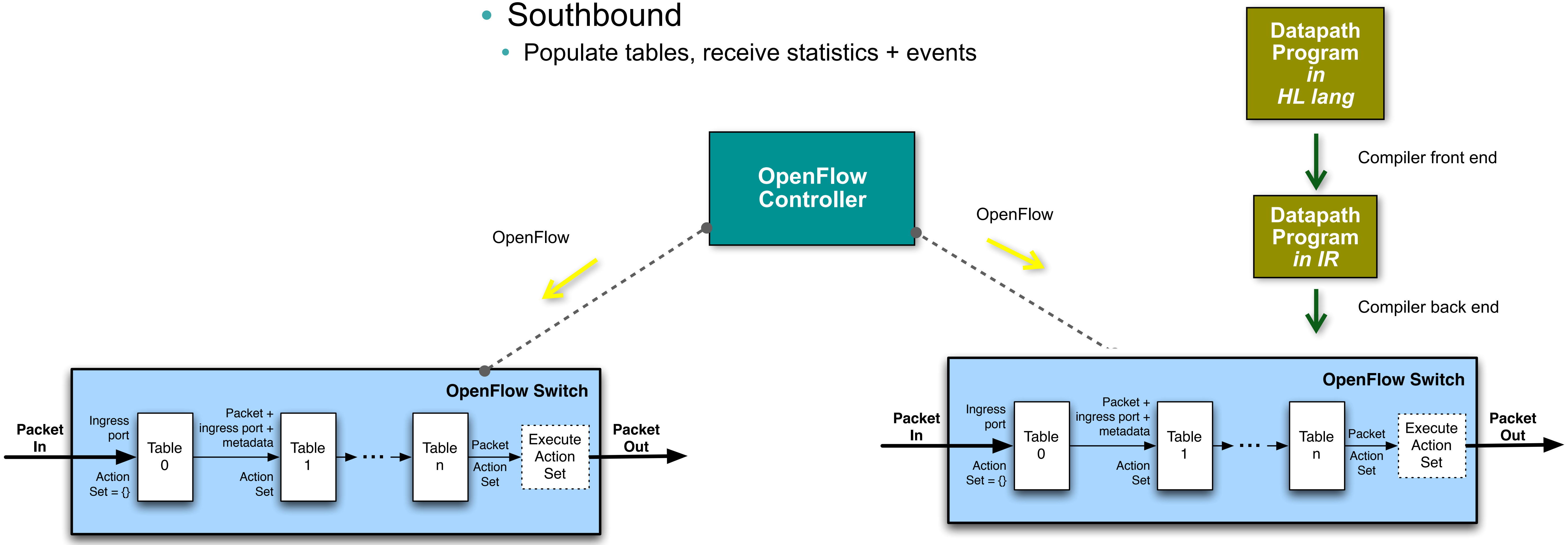


- Predefined Protocol Forwarding
  - OpenFlow specification defines protocols / fields, match/action behavior, overall control flow (tables can influence)
  - Set of supported protocols fixed by implementation

- Protocol Independent Forwarding (PIF) Configured (“Programmed”) Datapath
  - Programs in language(s) describe datapath
    - Parse tree => protocol independent
    - Match/action tables (control flow arranges table sequence)
    - Packet metadata, per table or global state
    - QoS

Emulating OpenFlow 1.x

- Southbound
  - Populate tables, receive statistics + events



Datapath Program *in HL lang*

Compiler front end

Datapath Program *in IR*

Compiler back end

OpenFlow Controller

OpenFlow

OpenFlow

OpenFlow Switch

Packet In

Ingress port

Action Set = {}

Table 0

Packet + ingress port + metadata

Action Set

Table 1

Table n

Packet

Action Set

Execute Action Set

Packet Out

OpenFlow Switch

Packet In

Ingress port

Action Set = {}

Table 0

Packet + ingress port + metadata

Action Set

Table 1

Table n

Packet

Action Set

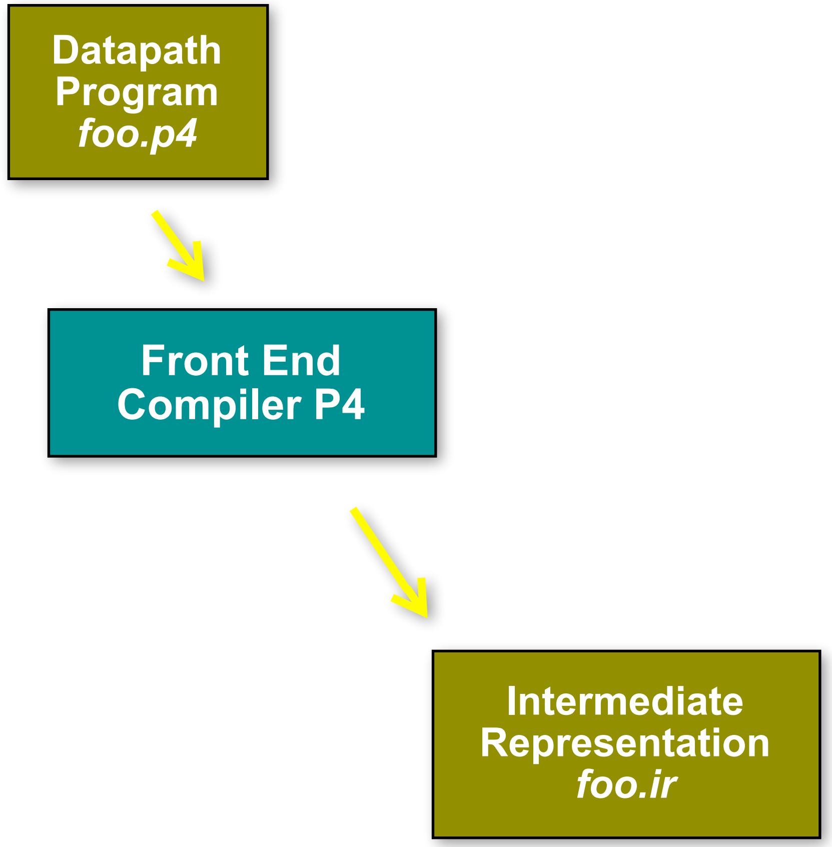
Execute Action Set

Packet Out

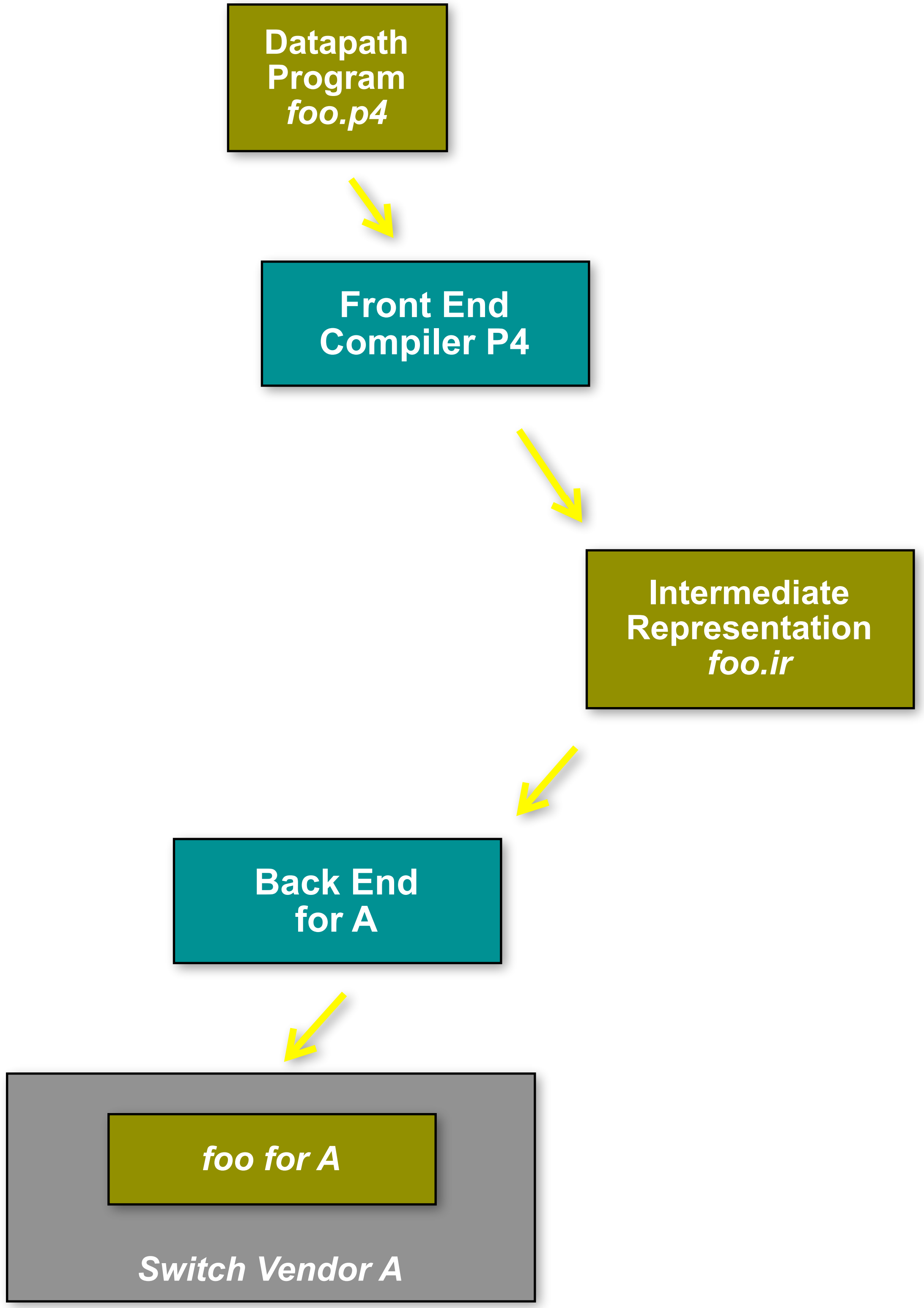
- Predefined Protocol Forwarding
  - Implements OpenFlow 1.x spec manually

- Configured (“Programmed”) Datapath
  - Program implements OpenFlow 1.x spec

Elements and Responsibilities



Elements and Responsibilities



Datapath Program *foo.p4*

Front End Compiler P4

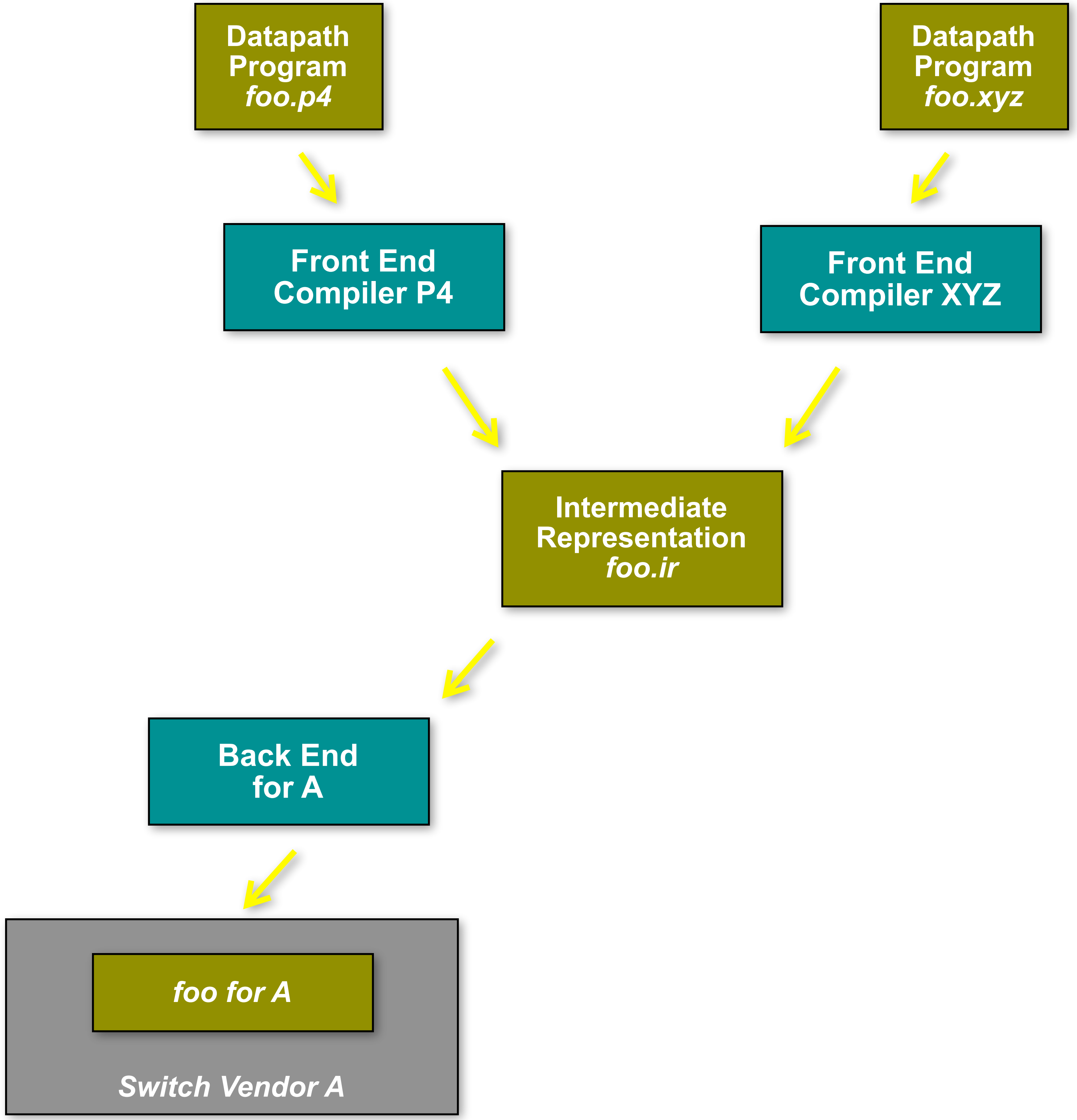
Intermediate Representation *foo.ir*

Back End for A

foo for A

Switch Vendor A

Elements and Responsibilities



Datapath Program *foo.p4*

Datapath Program *foo.xyz*

Front End Compiler P4

Front End Compiler XYZ

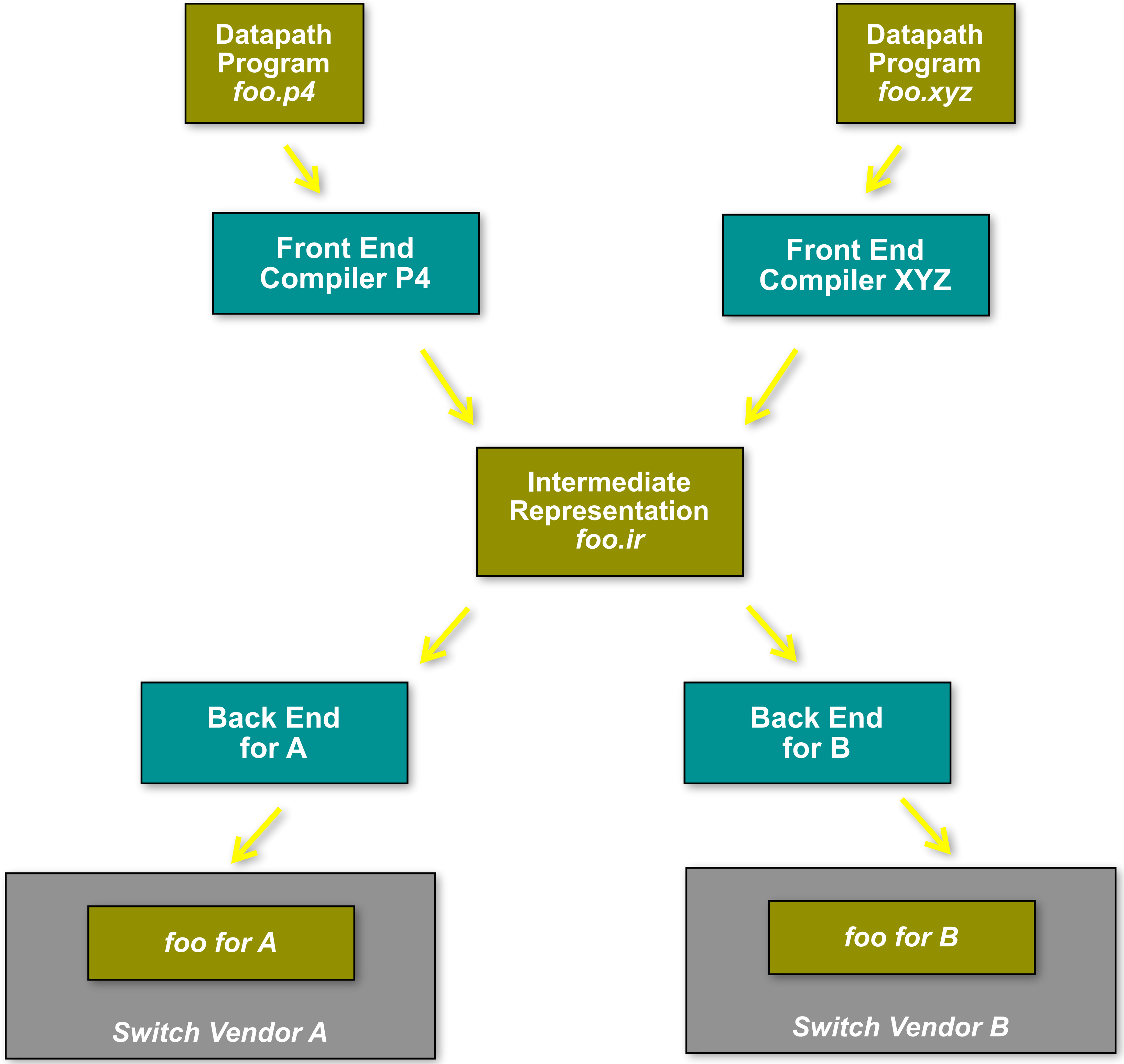
Intermediate Representation *foo.ir*

Back End for A

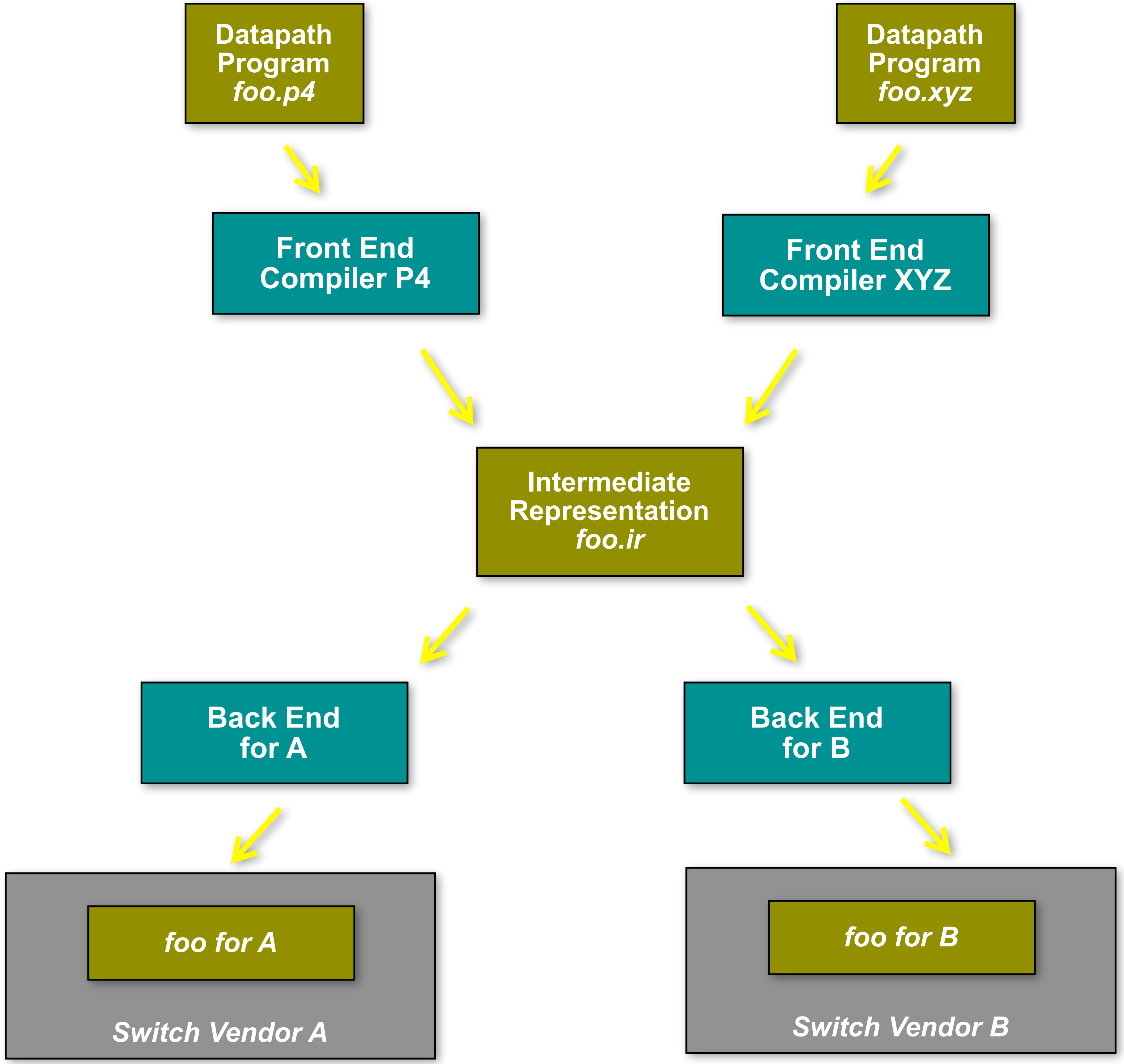
foo for A

Switch Vendor A

Elements and Responsibilities



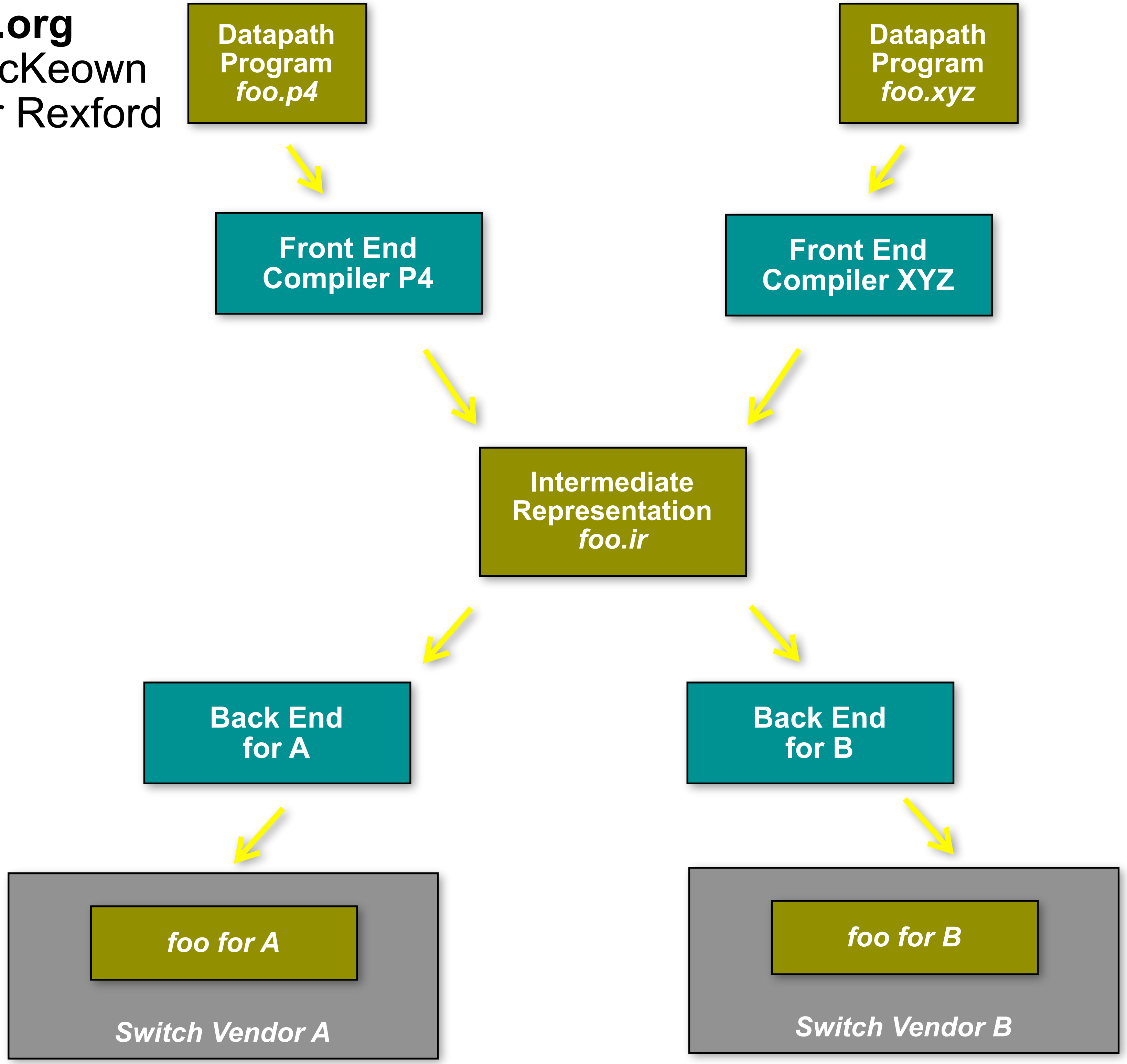
Elements and Responsibilities



- Target platforms: various pipeline types, FPGAs, processors - CPU/NPU

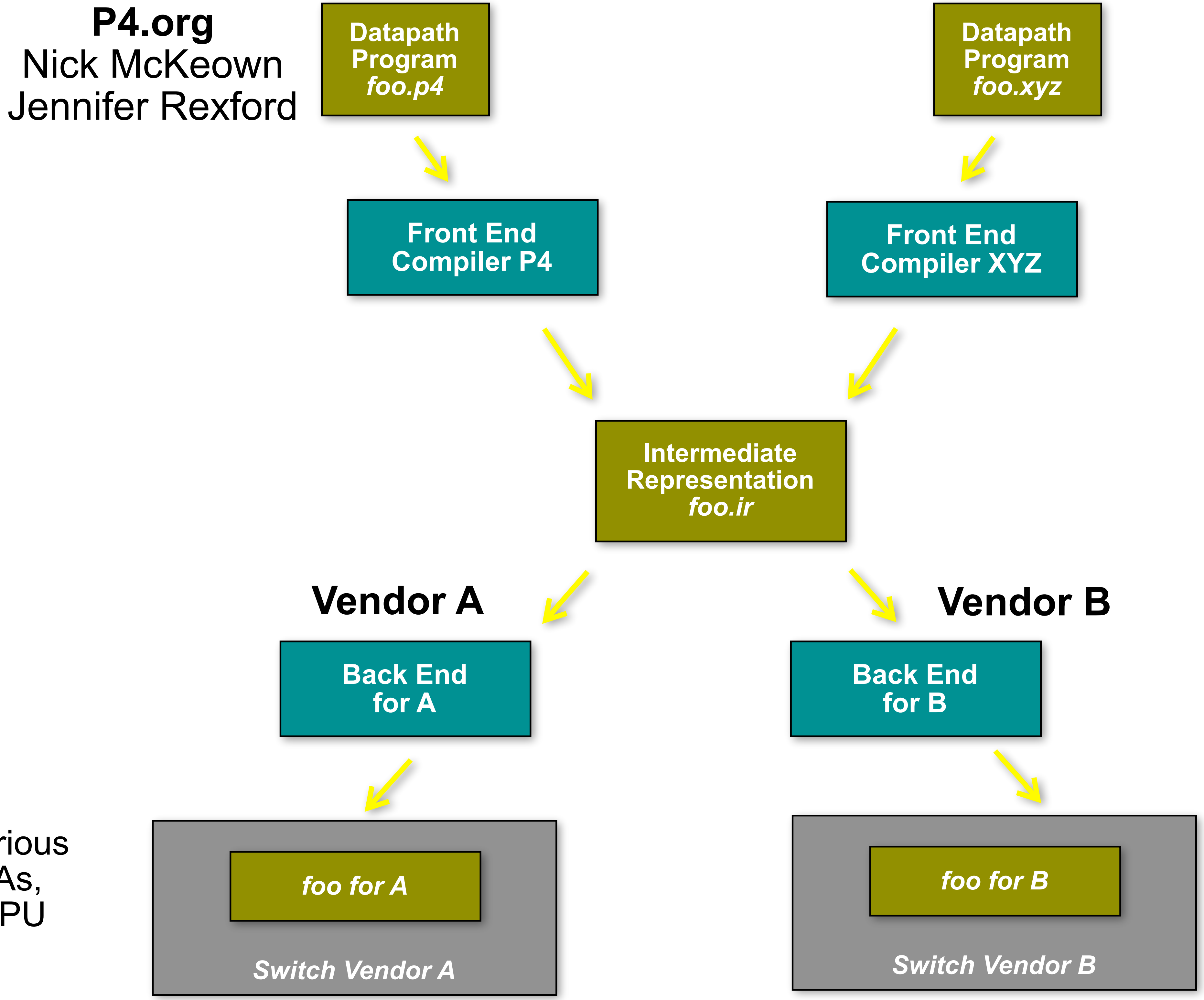
Elements and Responsibilities

P4.org
Nick McKeown
Jennifer Rexford



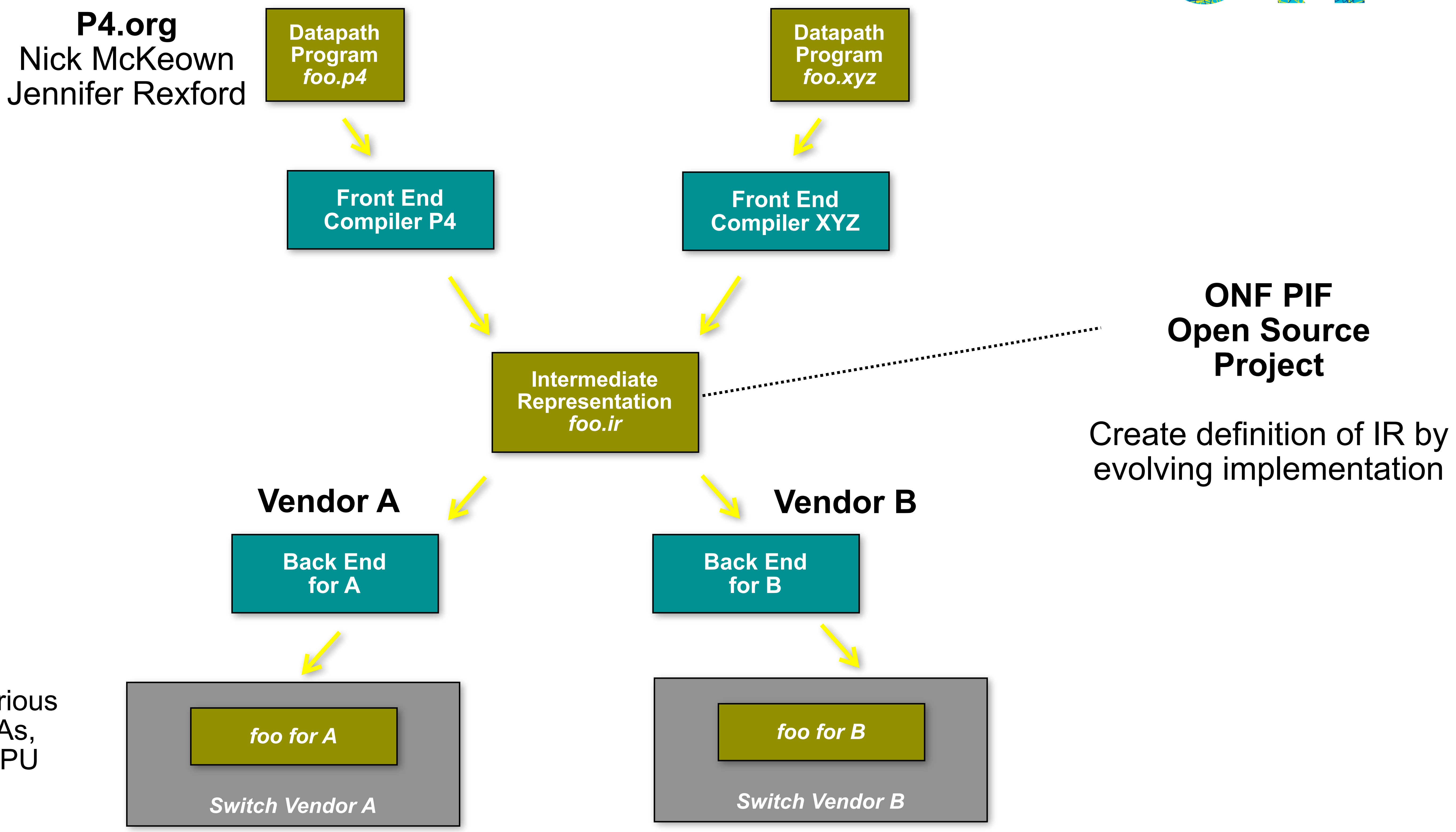
- Target platforms: various pipeline types, FPGAs, processors - CPU/NPU

Elements and Responsibilities



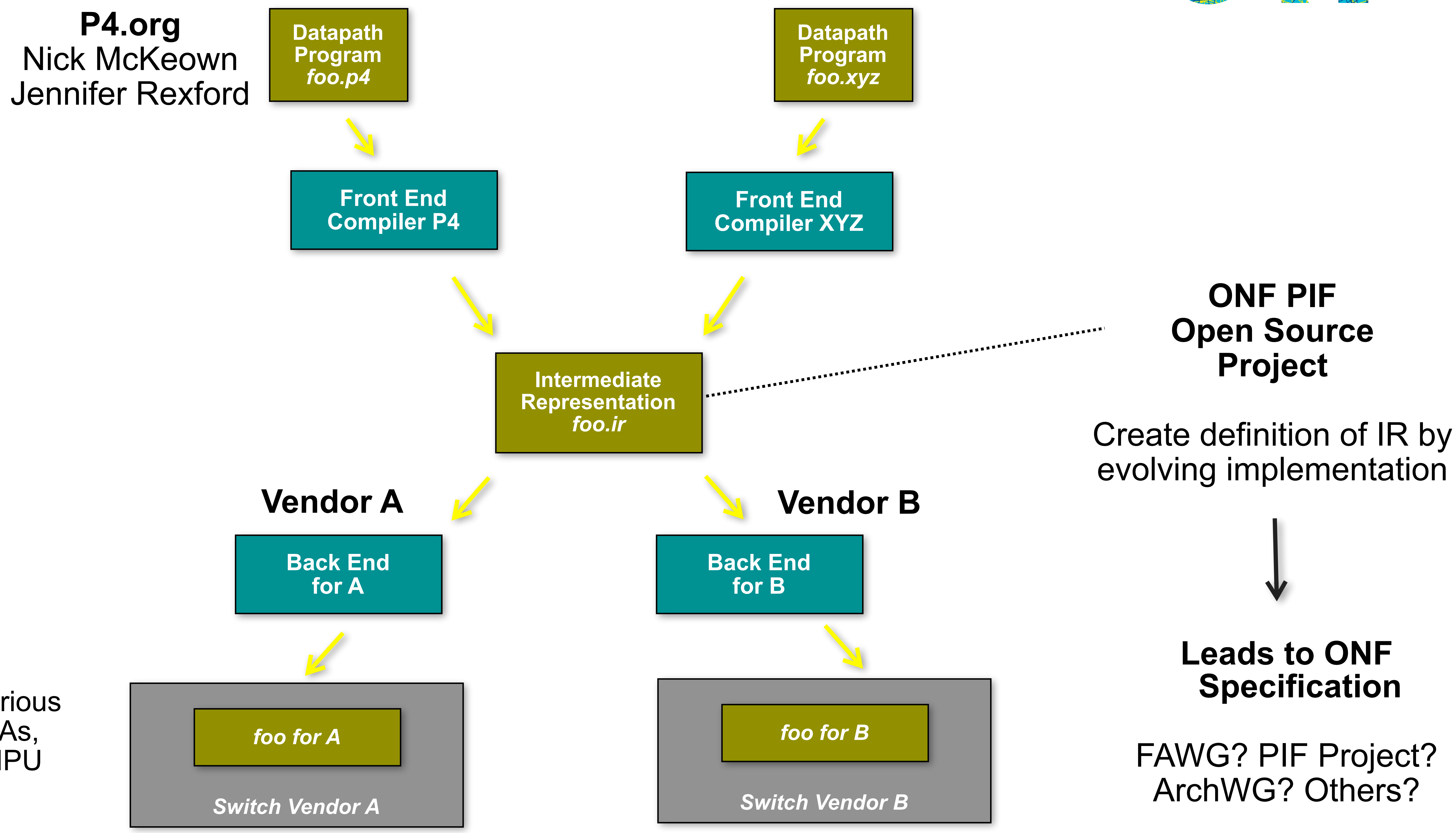
- Target platforms: various pipeline types, FPGAs, processors - CPU/NPU

Elements and Responsibilities



- Target platforms: various pipeline types, FPGAs, processors - CPU/NPU

Elements and Responsibilities



P4.org
Nick McKeown
Jennifer Rexford

Datapath Program
foo.p4

Datapath Program
foo.xyz

Front End Compiler P4

Front End Compiler XYZ

Intermediate Representation
foo.ir

ONF PIF Open Source Project

Create definition of IR by evolving implementation

Leads to ONF Specification

FAWG? PIF Project? ArchWG? Others?

Vendor A

Back End for A

foo for A

Switch Vendor A

Vendor B

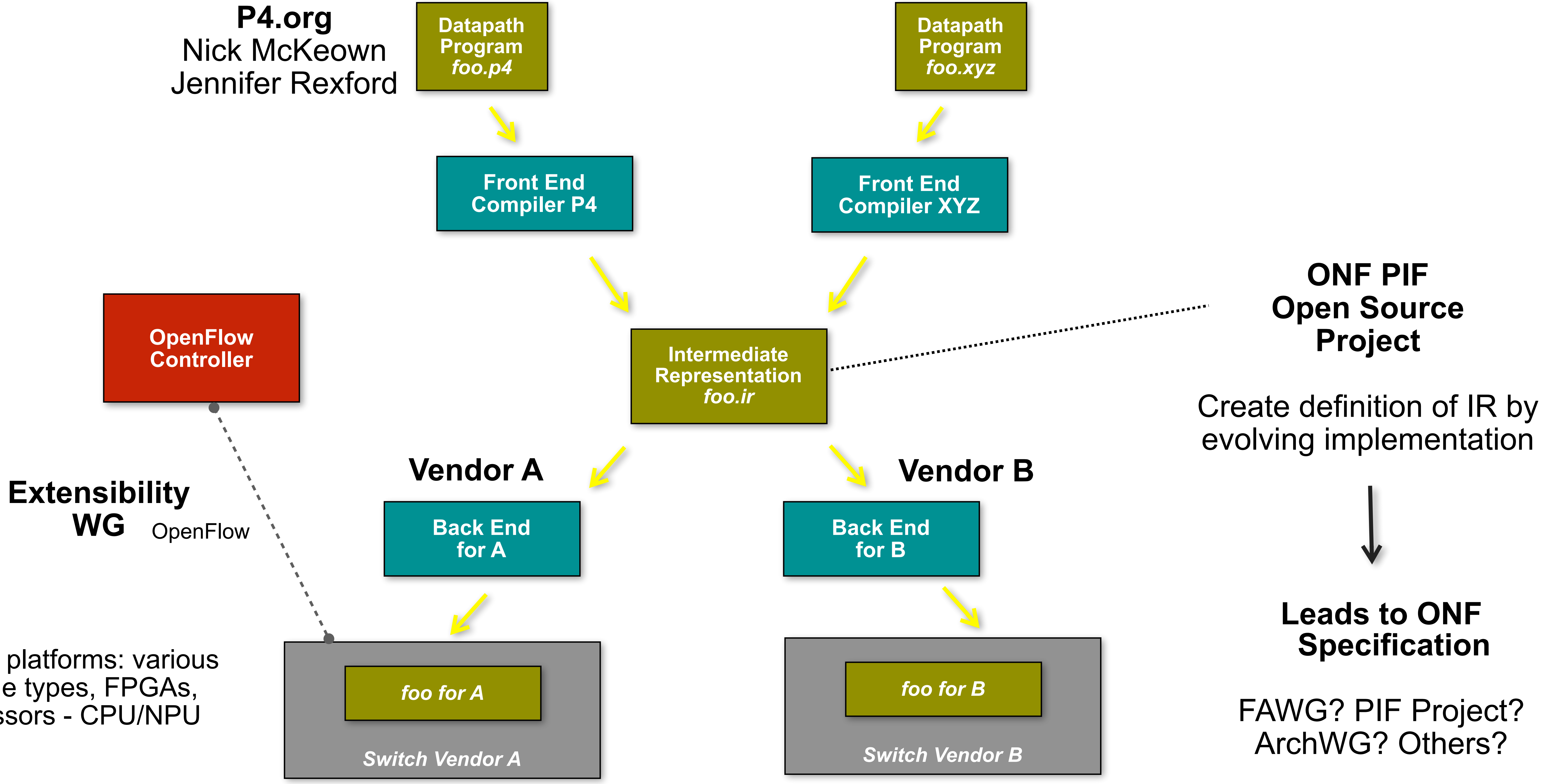
Back End for B

foo for B

Switch Vendor B

- Target platforms: various pipeline types, FPGAs, processors - CPU/NPU

Elements and Responsibilities



P4.org
Nick McKeown
Jennifer Rexford

Datapath Program *foo.p4*

Datapath Program *foo.xyz*

Front End Compiler P4

Front End Compiler XYZ

OpenFlow Controller

Intermediate Representation *foo.ir*

ONF PIF Open Source Project

Create definition of IR by evolving implementation

Leads to ONF Specification

FAWG? PIF Project? ArchWG? Others?

Extensibility WG

OpenFlow

Vendor A

Back End for A

foo for A

Switch Vendor A

Vendor B

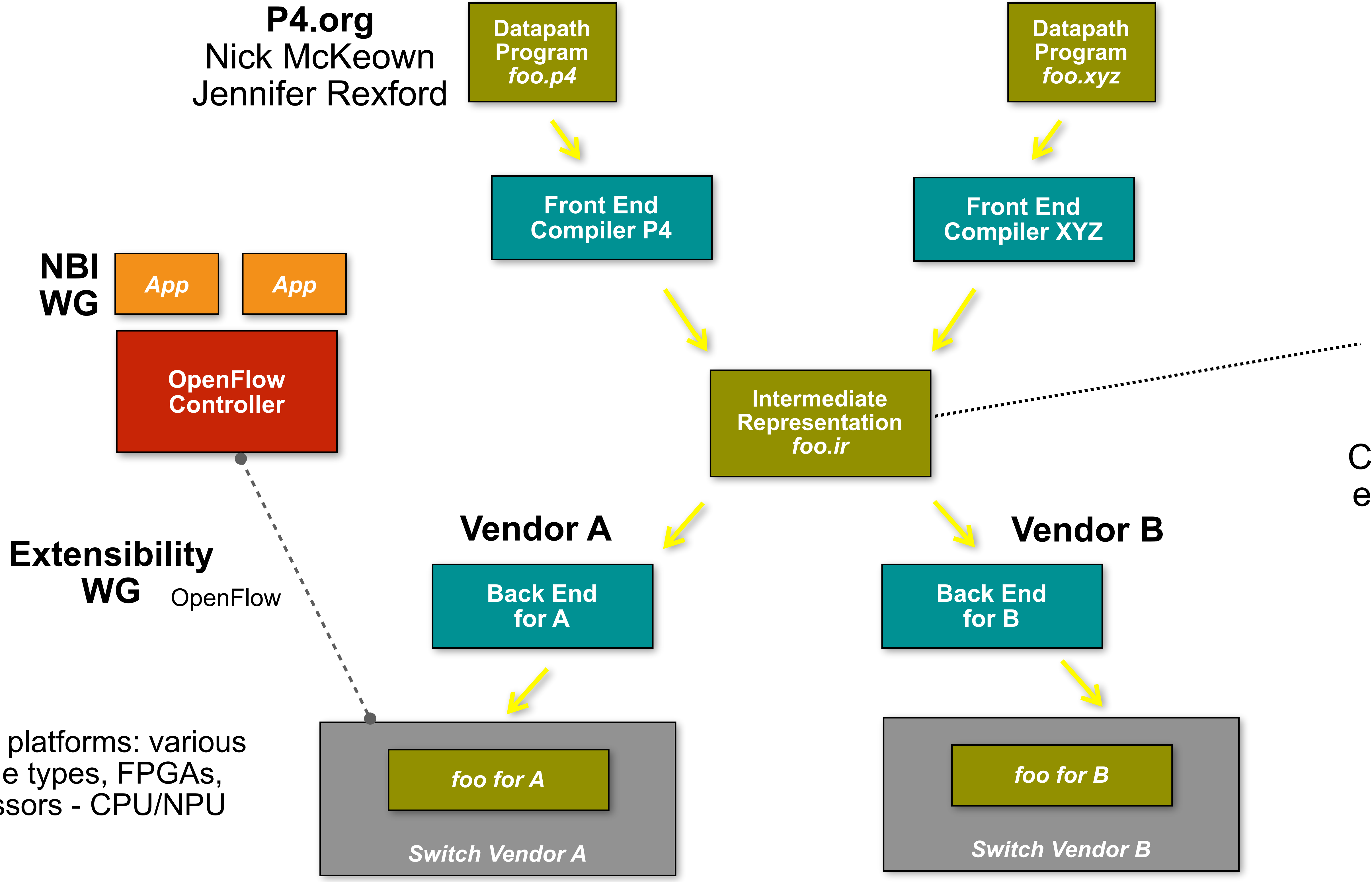
Back End for B

foo for B

Switch Vendor B

- Target platforms: various pipeline types, FPGAs, processors - CPU/NPU

Elements and Responsibilities



- Target platforms: various pipeline types, FPGAs, processors - CPU/NPU

ONF PIF Open Source Project

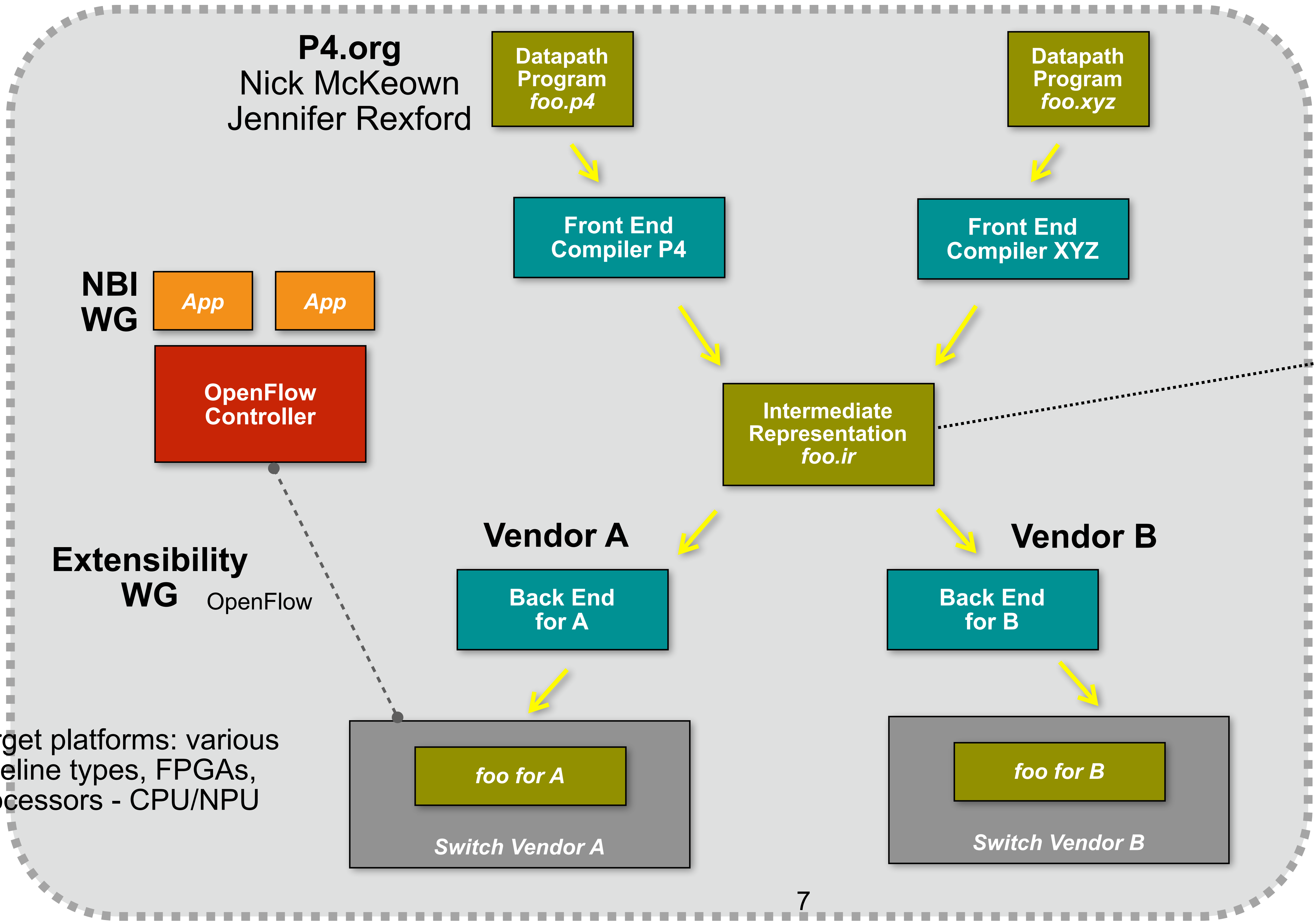
Create definition of IR by evolving implementation

↓

Leads to ONF Specification

FAWG? PIF Project? ArchWG? Others?

Elements and Responsibilities



- Target platforms: various pipeline types, FPGAs, processors - CPU/NPU

FAWG
Wider OpenFlow-NG concerns e.g. lifecycles / forwarding models / capability profiles…

ONF PIF Open Source Project

Create definition of IR by evolving implementation

Leads to ONF Specification

FAWG? PIF Project? ArchWG? Others?

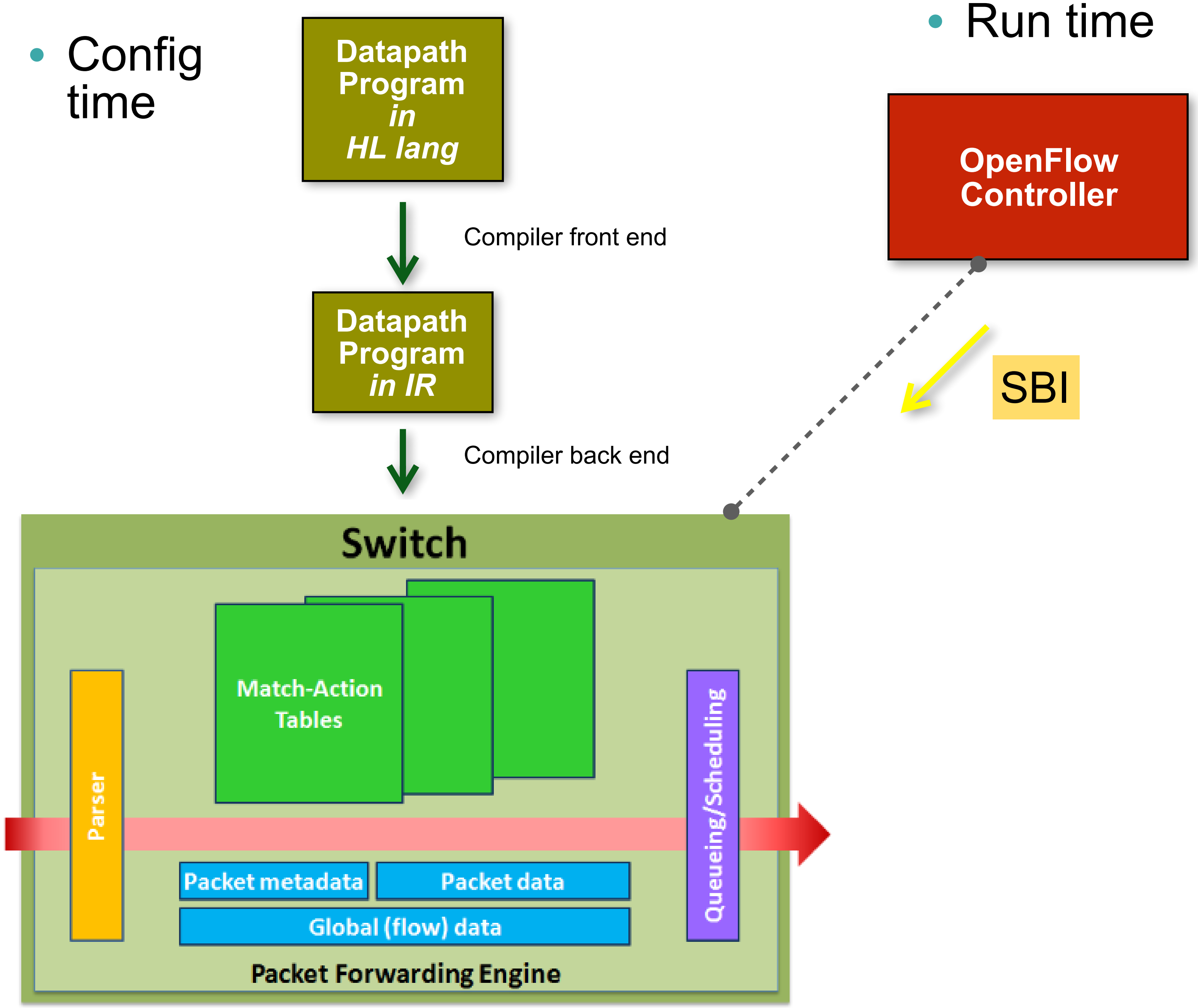
New Elements (Draft)

- Datapath program
  - Who writes it? vendor / operator / end user / their agents
  - What is it? monolitic / modular program (libraries) => ecosystem
- Front end compiler
  - Where does this run? — developer workstation or controller?
  - Who supplies it? — multi-vendor / de-facto standard?
- Back end compiler
  - Where does this run? — switch or controller?
  - Who supplies it? — almost certainly vendor specific

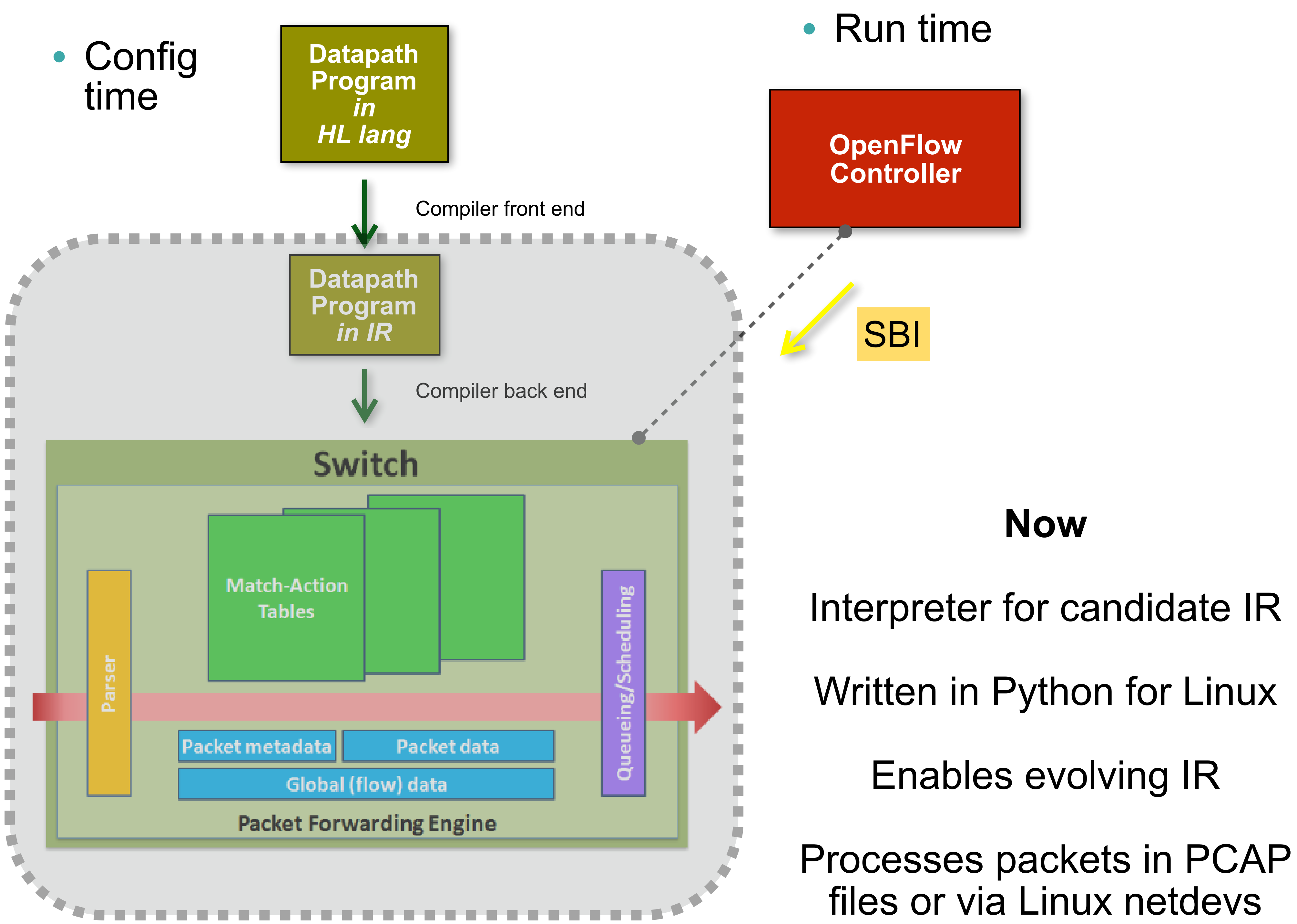
AIR-IRI: Prototype IR

- Config time

- Run time



AIR-IRI: Prototype IR



Now

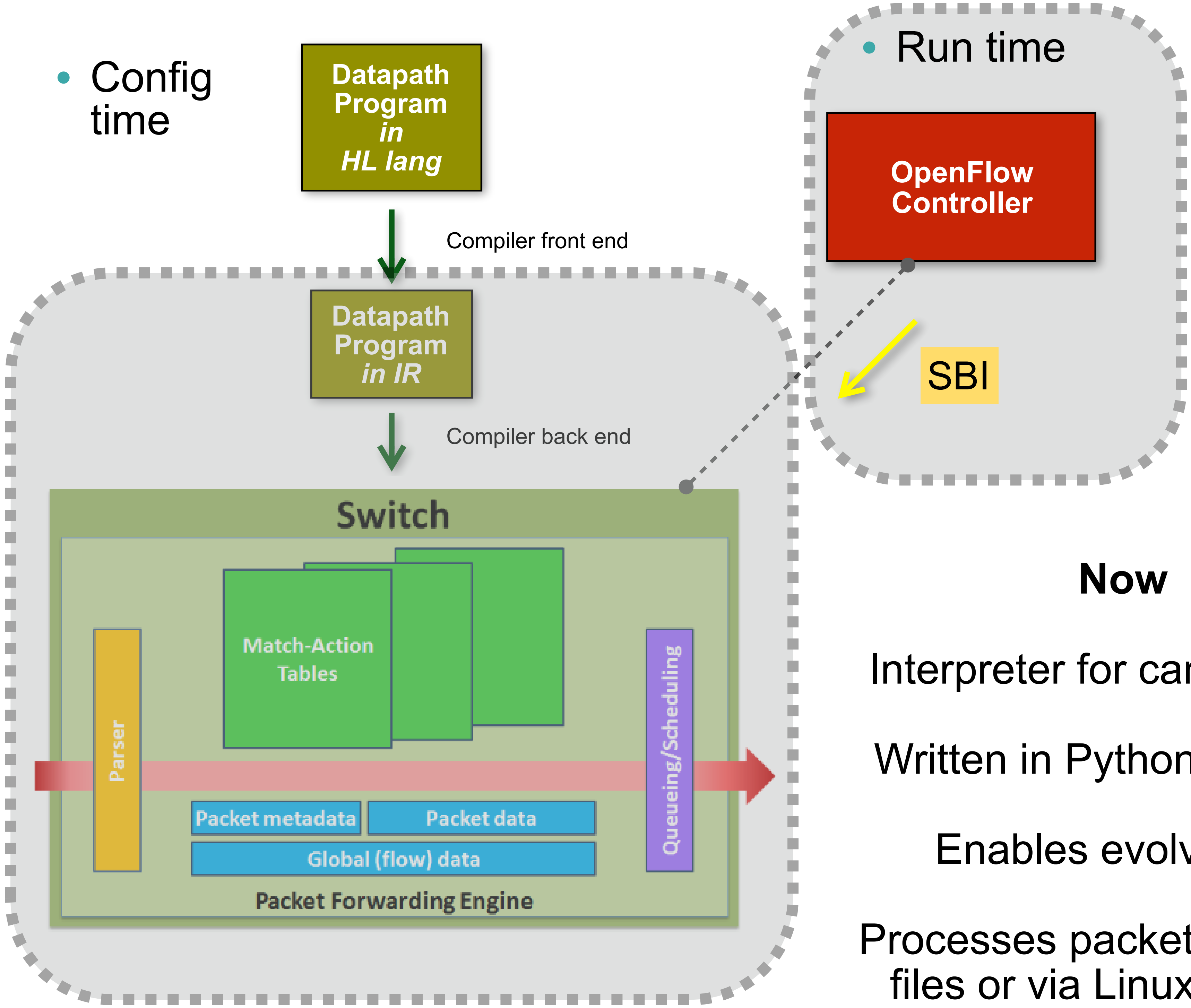
Interpreter for candidate IR

Written in Python for Linux

Enables evolving IR

Processes packets in PCAP files or via Linux netdevs

AIR-IRI: Prototype IR



- Config time

Datapath Program *in HL lang*

Compiler front end

Datapath Program *in IR*

Compiler back end

- Run time

OpenFlow Controller

SBI

Soon

Hook up to run-time interface
- Apache Thrift?
- OpenFlow 1.x?

Now

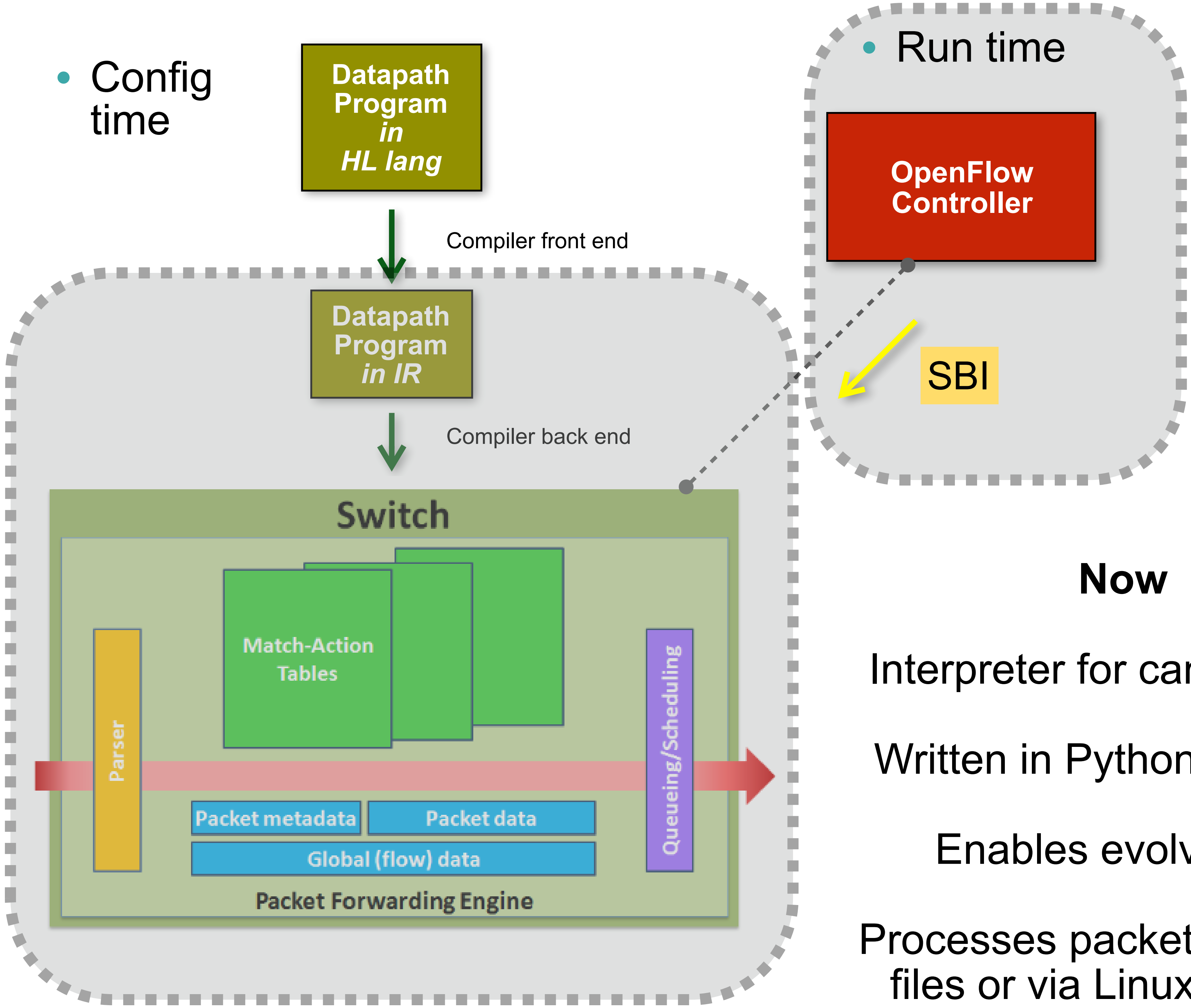
Interpreter for candidate IR

Written in Python for Linux

Enables evolving IR

Processes packets in PCAP files or via Linux netdevs

AIR-IRI: Prototype IR



Soon

Hook up to run-time interface
- Apache Thrift?
- OpenFlow 1.x?

Now

Interpreter for candidate IR

Written in Python for Linux

Enables evolving IR

Processes packets in PCAP files or via Linux netdevs

ToDo for YOU!

Define new IR variant, improve interpreter to support it

Implement IR back end for your platform

Meta IR Facilitates Evolution / Experimentation

- AIR = framework for creating IR interpreters
  - Facilitates introducing classes of “objects”, e.g. match tables, action blocks, TM blocks
  - Each “object” has named attributes (like C structure)
  - air_meta.yml defines the objects and their attributes (declares classes/types)

- IRI = instance of an IR
  - Specific set of “objects” with implementation of each
  - Currently defined objects: value_set, value_map, table, header, metadata, action, parse_state, parser, control_flow, traffic_manager, processor_layout
  - Community invited to extend / modify this

IRI Elements: Types

```
# Complete list of types:
  - table
  - header
  - metadata
  - action
  - parse_state
  - parser
  - control_flow
  - traffic_manager
  - processor_layout
```

```
# Types with a process method:
  - control_flow
  - parser
  - traffic_manager
```

IRI Elements: Attributes per Type

```
# All support type and doc
air_attributes :
  table :
    - match_on
  header :
    - fields
    - max_depth # hdr stack if > 1
  metadata :
    - fields
    - initial_values
  action :
    - format
    - parameter_list
    - implementation
  parse_state :
    - extracts
    - select_value # Optional
```

```
# CONTINUED
  control_flow : *graph_attributes
  parser :
    - format
    - implementation
    - start_state
  traffic_manager : # Experimental
    - queues_per_port
    - dequeue_discipline
    - egress_spec_map
  processor_layout:
    - format
    - implementation
    - port_count
```

Layout: Top Level Pipeline Structure

```
layout:
  type : processor_layout
  doc  : "The layout specification for the switch instance"
  port_count : 4
  format : list
  implementation :
    - parser
    - ingress_flow
    - tm_queues
    - egress_flow
```

- Changes for each category of “datapath program” (e.g. with / without QoS)
- Simple list of processors currently implemented
  - Future: more complex topologies

Protocol / Metadata Fields and Actions

```
# Header object
ethernet :
  type : header
  doc : "The L2 header"
  fields :
    - dst_mac : 48
    - src_mac : 48
    - ethertype : 16
```

```
# Metadata object
pkt_md : # General metadata
  type : metadata
  doc : "General metadata for the packet"
  fields :
    # Virtual network instance identifier
    - vni : 16
```

```
# Action object
set_vni_a :
  type : action
  doc : "Set the VNI in metadata"
  format : action_set
  parameter_list :
    - vni_id
  implementation : >-
    modify_field(pkt_md.vni, vni_id);
```

- Protocol fields vary according to targeted network protocols
- Metadata fields and actions vary according to required behavior
- =>“Datapath programming” commencing in earnest

Protocol Details - Complete Parser

```
ethernet_p :
  type : parse_state
  doc : "Parse state for ethernet"
  extracts :
    - ethernet
  select_value :
    - ethernet.ethertype

vlan_p :
  type : parse_state
  doc : "Parse state for vlan tag"
  extracts :
    - vlan_tag
```

```
parser :
  type : parser
  doc : "Implementation of primary parser"
  format : dot
  start_state : ethernet_p
  implementation : >-
    digraph {
      ethernet_p -> vlan_p [value="0x8100"]
      ethernet_p -> vlan_p [value="0x9100"]
    }
```

- Parse tree currently specified in dedicated parser object (parsing before matching)
- Being considered: match protocol ID using a table, then trigger parsing next header (parse - match - parse - match)

Tables and Control Flow

```
vni :
  type : table
  doc : "Map VLAN to VNI"
  match_on :
    vlan_tag.vlan_id : ternary
    ethenet.src_mac : ternary

forward :
  type : table
  doc : "Forward based on L2 dest addr"
  match_on :
    pkt_md.vni : exact
    ethenet.dst_mac : exact

acl :
  type : table
  doc : "Perform ACL operations"
  match_on :
    pkt_md.vni : exact
    ethenet.dst_mac : exact
    ethenet.src_mac : exact
```

```
ingress_flow :
  type : control_flow
  doc : "The control flow for ingress"
  format : dot
  implementation : >-
    digraph {
      vni -> forward [action=set_vni_a]
      forward -> exit_control_flow [action=set_egress_a]
      forward -> exit_control_flow [action=drop_pkt_a]
    }

egress_flow :
  type : control_flow
  doc : "The control flow for egress"
  format : dot
  implementation : >-
    digraph {
      acl -> exit_control_flow [action=set_dst_mac_a]
      acl -> exit_control_flow [action=drop_pkt_a]
    }
```

AIR / IRI Evolution

- Evolve IR language interpreter
  - Enhance existing objects e.g. matching, actions, QoS
  - New concepts e.g. statefulness
- Tools operating on IR datapath programs
  - Visualization
  - Import / Export e.g. create NDM / TTP from IR program
  - Predict performance / capacity requirements
  - Transform - e.g. parse everything then match vs. incremental distributed parsing
- Introduce run-time
  - Callable auto-marshalled, e.g. Apache Thrift
  - Traditional protocol based, e.g. OpenFlow 1.x
  - Sample controller + sample applications on controller
- Samples
  - Sample datapath applications (in IR? in high level language once front end compilers ready?)
  - Libraries for common protocols, actions (+ infrastructure for this - templating)

Discussion + Next Steps