P4 for an FPGA target

Gordon Brebner
Xilinx Labs
San José, USA

P4 Workshop, Stanford University, 4 June 2015

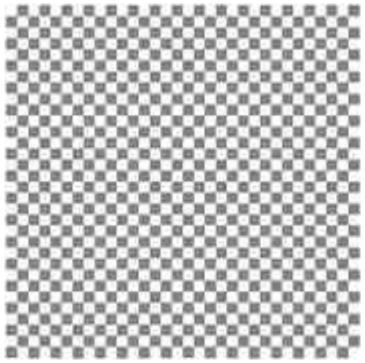
What this talk is about

- FPGAs and packet processing languages
- Xilinx SDNet data plane builder featuring the PX language
- P4 and PX comparison
- P4-to-FPGA compilation using SDNet
- Protocol Independent Forwarding (PIF) project
- Summary

FPGA: the “white box” hardware chip

- **Attributes of Xilinx Ultrascale FPGAs in 2015:**

Up to 2.5m tiles



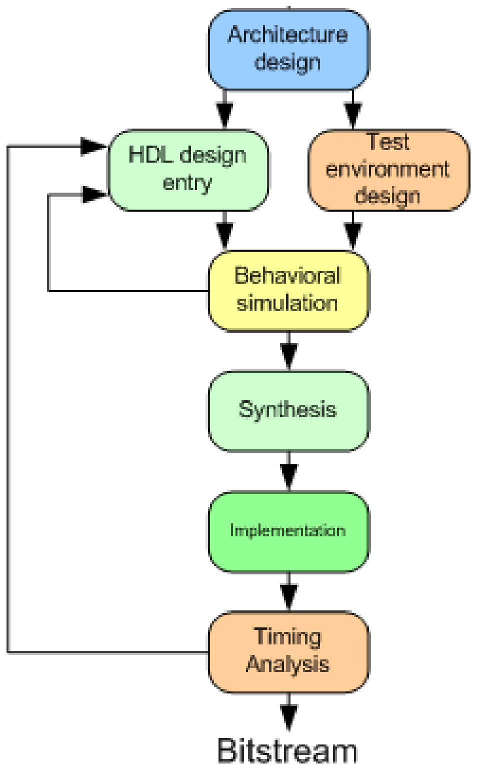
Tile is 6-input logic gate and 2 flip-flops
+ Embedded function blocks and memories
+ 100G Ethernet and 150G Interlaken interfaces

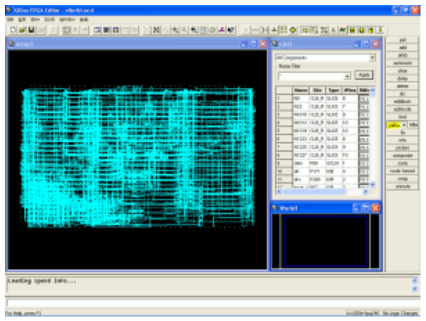
Local and long-distance wiring between components

All **programmable** by writing to memory

- **Can now implement complex packet processing on single chip**
  - Open platform used by researchers: NetFPGA
- **Beyond single chips:**
  - Multiple FPGAs, e.g., Corsa SDN data planes
  - FPGA fast paths, CPU slow paths and control
  - FPGA smart paths, ASIC dumb switches

Generic FPGA programming





- Chip design style experience
- Hardware Description Language (HDL)
- Cryptic results sometimes
  - Behavior and performance
- Enhancements (for 'hardware guys'):
  - Libraries of blocks
    - Allow re-use and sharing
    - In HDL, or pre-synthesized
  - High-level synthesis
    - Typically superset of subset of C
    - Translated into HDL
- Abstraction needed for 'software guys'
  - Current emphasis at Xilinx: **SDx environments**

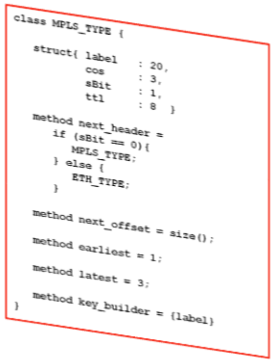
Packet processing research in Xilinx Labs

- **First generation (1 to 2 Gb/s line rates, 2001-2005)**
  - Initially, fine-grain Click as domain-specific language [DAC 2004]
  - Then, coarse-grain Click plus HAEC state machine language [FPT 2004]
- **Second generation (10 to 20 Gb/s line rates, 2005-2009)**
  - G language for packet processing functions …
  - … plus Click (with extended semantics) for composition of functions
  - Eventually published late in life [NetFPGA 2009, FPT 2009]
- **Third generation (100+ Gb/s line rates, 2009-2013)**
  - PX language for packet processing functions and their composition
  - First version published using PP as name [ANCS 2011]
  - Enhanced version published using PX as name [IEEE Micro 2014]
  - Productized as **Xilinx SDNet** in 2014

Xilinx SDNet Programmable Packet Processor (www.xilinx.com/sdnet)

- **Headline feature set, facilitated by FPGA white box target:**

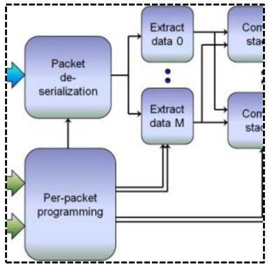
100G+ packet rate performance



```
class MPLS_TYPE {

  struct( label     : 20,
          cos       : 3,
          sBit      : 1,
          ttl       : 8  }
  method next_header =
    if (sBit == 0){
      MPLS_TYPE;
    } else {
      ETH_TYPE;
    }

  method next_offset = size();

  method earliest = 1;

  method latest = 3;

  method key_builder = (label)
}
```

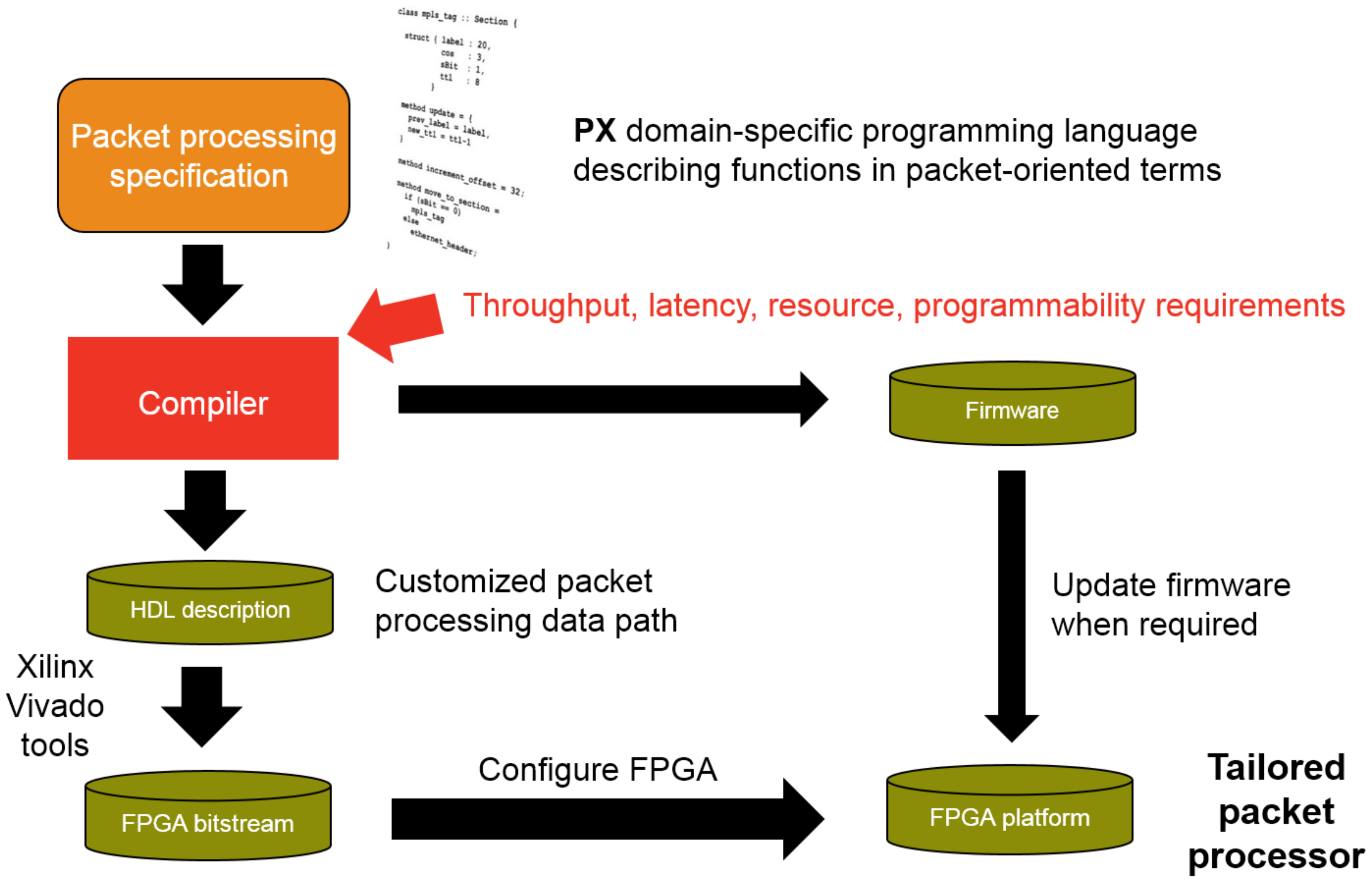
Domain-specific programming abstraction

Exact-fit hardware for reduced cost and power



Firmware for run time programmability

Design flow and use model



```
class mpls_tag :: Section {

  struct { label : 20,
           cos   : 3,
           sBit  : 1,
           ttl   : 8
         }

  method update = {
    prev_label = label,
    new_ttl = ttl-1
  }

  method increment_offset = 32;

  method move_to_section =
    if (sBit == 0)
      mpls_tag
    else
      ethernet_header;
}
```

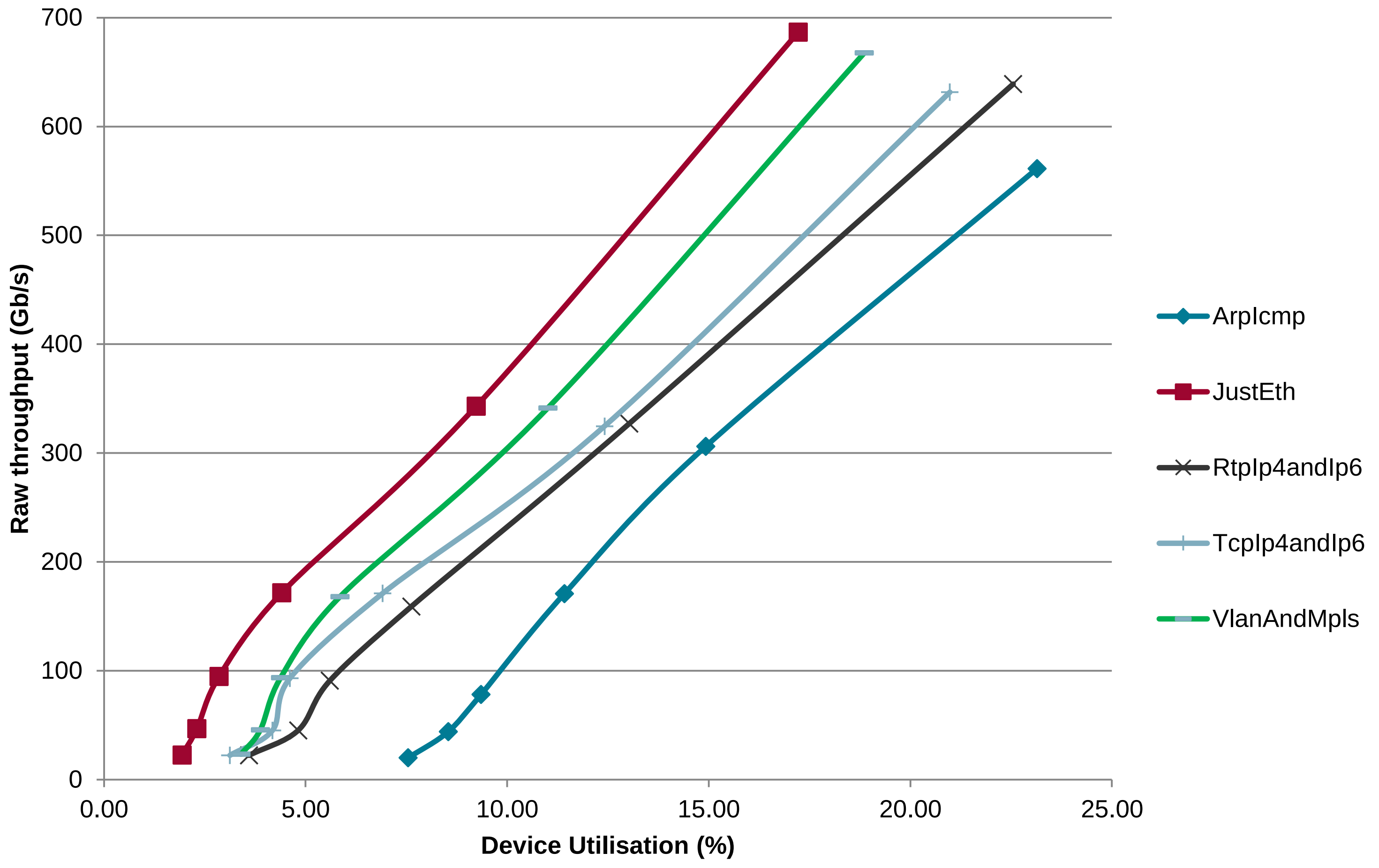
Packet processing specification
PX domain-specific programming language describing functions in packet-oriented terms
Throughput, latency, resource, programmability requirements
Compiler
Firmware
HDL description
Customized packet processing data path
Update firmware when required
Xilinx Vivado tools
FPGA bitstream
Configure FPGA
FPGA platform
Tailored packet processor

A glimpse of PX

```
struct flow_s {                                       // OpenFlow 1.0 12-tuple
       port : 3,                                      // Arrival port
       dmac : 48, smac : 48, type : 16,               // Ethernet
       vid : 12, pcp : 3,                             // VLAN
       sa : 32, da : 32, proto : 8, tos : 6,          // IP
       sp : 16, dp : 16                               // TCP or UDP
}

class OF_parser :: ParsingEngine (9216*8, 4, ETH_header) {

    class OF :: Tuple(inout) { struct flow_s; }
    OF fields;

    // Section sub-class for an Ethernet header
    class ETH_header :: Section {
        struct { dmac : 48, smac : 48, type : 16 }
        method update = {
            fields.dmac = dmac, fields.smac = smac, fields.type = type
        }
        method move_to_section =
            if (type == 0x8100) VLAN_header
            else if (type == 0x0800) IP_header else done(1);
    }

    // Section sub-class for a VLAN header
    class VLAN_header :: Section {
        struct { pcp : 3, cfi : 1, vid : 12, tpid : 16 }
        method update = { fields.vid = vid, fields.pcp = pcp }
        method move_to_section = if (tpid == 0x0800) IP_header else done(1);
    }

    // Section sub-class for an IP header ...
```

Throughput can be traded off with FPGA utilization

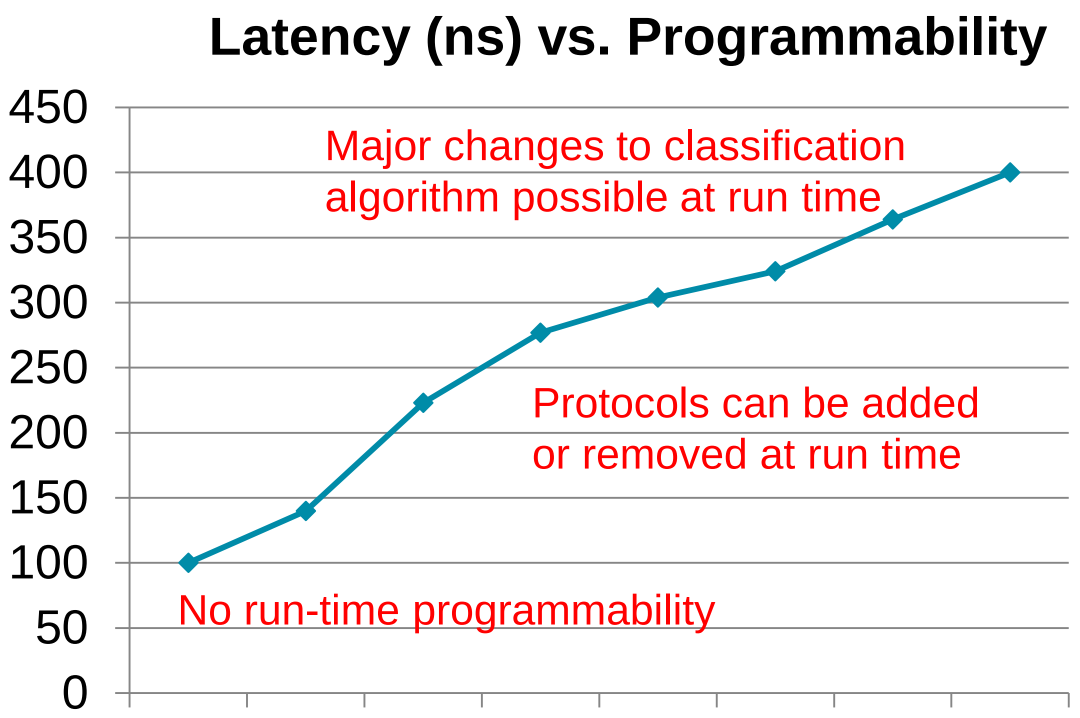
- Different raw throughputs through varying data path width parameter



XILINX ALL PROGRAMMABLE.

Latency can be traded off with programmability

- High-level description specifies the packet processing functions
- Compiler optimizes implementation characteristics by providing just the required run-time programmability



Example of trade-offs for 30-protocol packet classification design

Current P4 and PX: Many similarities

- Domain-specific: fast path packet processing
- Declarative: based on packet-centric abstractions
- Describe per-packet processing in isolation
- Restricted computation model: no loops, pointers, etc.
- Abstracted from physical implementation detail
- *Not designed by language experts*

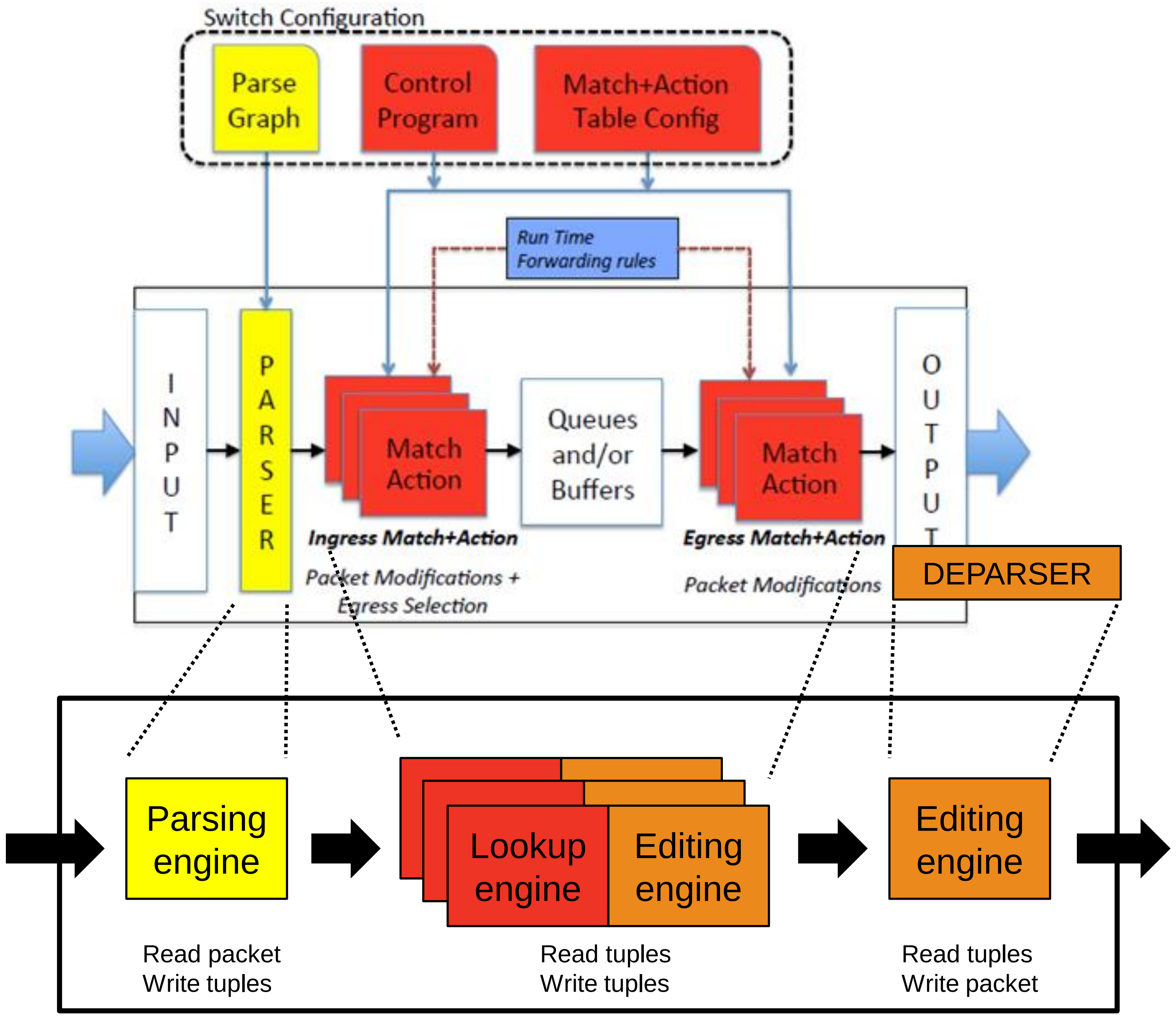
- Headers ("sections" in PX) specified using structs
- Packet metadata specified using structs
- Protocol-agnostic packet parsing via programmed transitions
- Exact match, LPM, and ternary style matching
- Protocol-agnostic packet editing

Current P4 and PX: Some differences

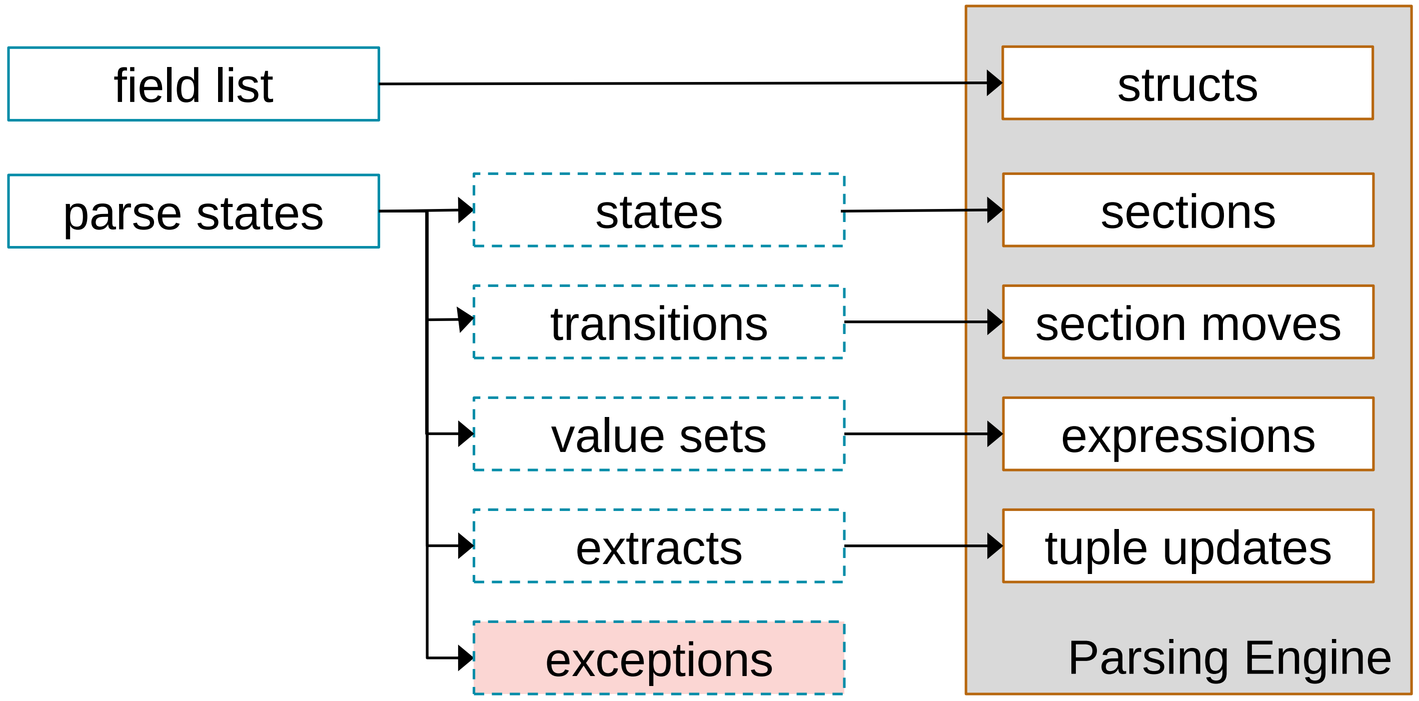
- **PX features:**
  - Object-oriented semantics, and modular specifications
  - More general parsing transitions, and programmable offset calculation
  - Allows incorporation of autonomous (library or user) black box engines
  - Hierarchical data flow-based composition of engines to build data planes
  - Packet update model based on editing operations (not deparsing)

- **P4 features:**
  - Match-action table semantics
  - Imperative control flow functions
  - Built-in complex calculations (e.g., checksums, hashes)
  - Explicit inter-packet state: counters, meters, registers
  - Resubmit, recirculate, and clone, actions

Mapping P4 to PX

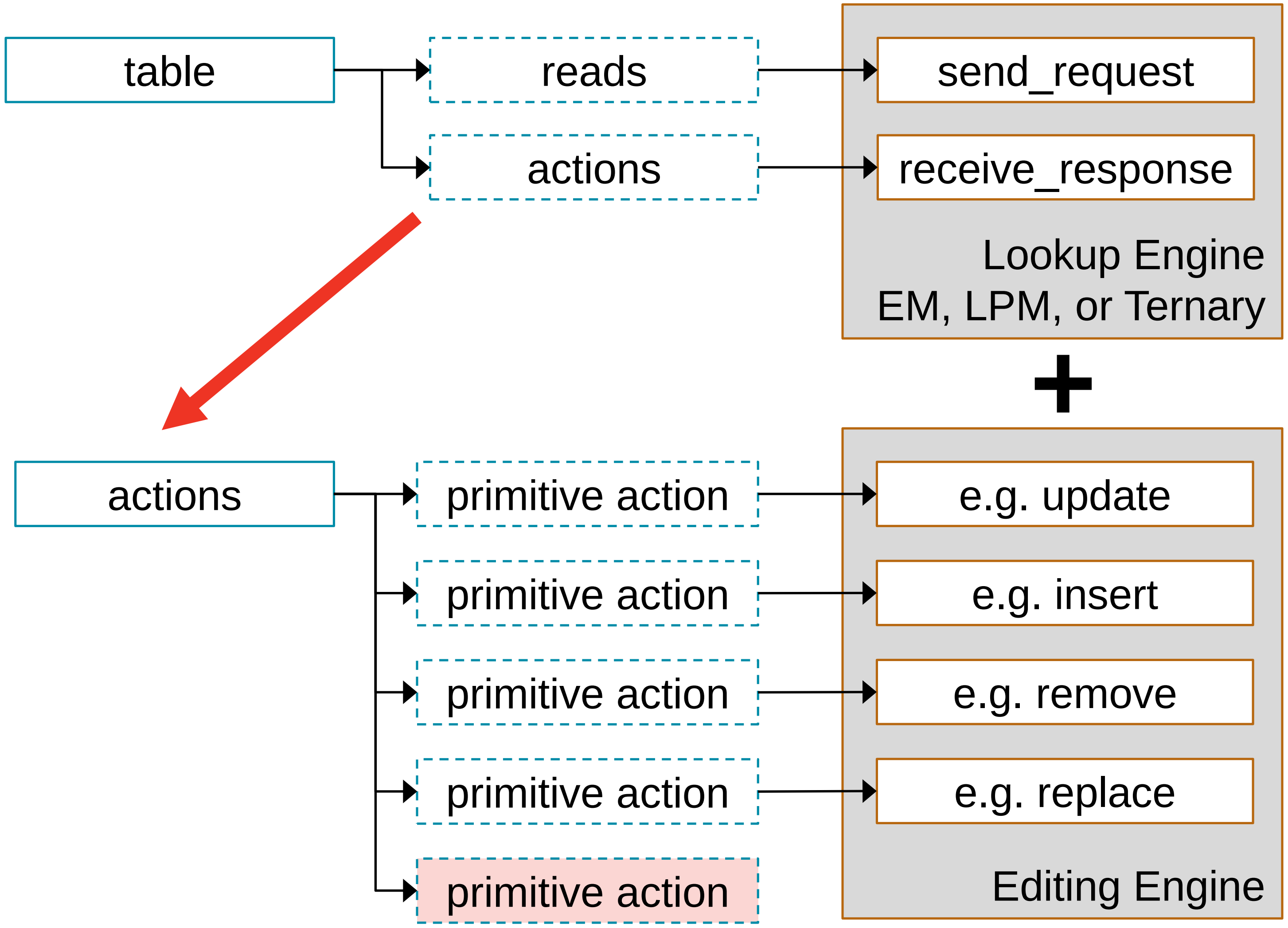


Parser

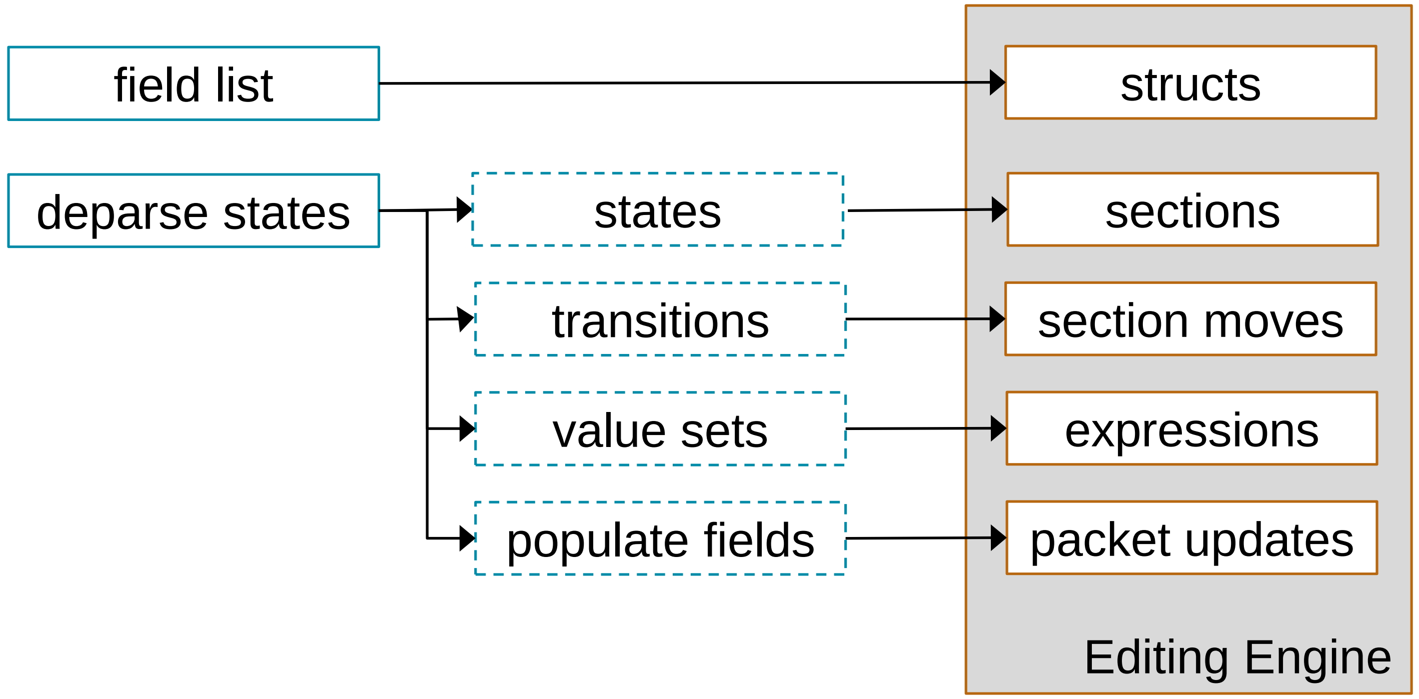

field list
parse states
states
transitions
value sets
extracts
exceptions
structs
sections
section moves
expressions
tuple updates
Parsing Engine

Match-Action table

Mapped to Lookup Engine and Editing Engine pair



Deparser


field list
deparse states
states
transitions
value sets
populate fields
structs
sections
section moves
expressions
packet updates
Editing Engine

Optimization experiment 1

- **Direct mapping to PX:**
  - Each MAT mapped to a LE-EE pair
  - Ternary LEs used for generality
- **Optimized mapping to PX:**
  - Small MAT functions included within parsing states (“POF-style”)
  - Lookup uses in-expression CAM plus operator-based wildcarding
  - Actions coded by tuple updates
  - Deparser editing engine does all packet update actions
- **Example impact: 23% resource reduction**

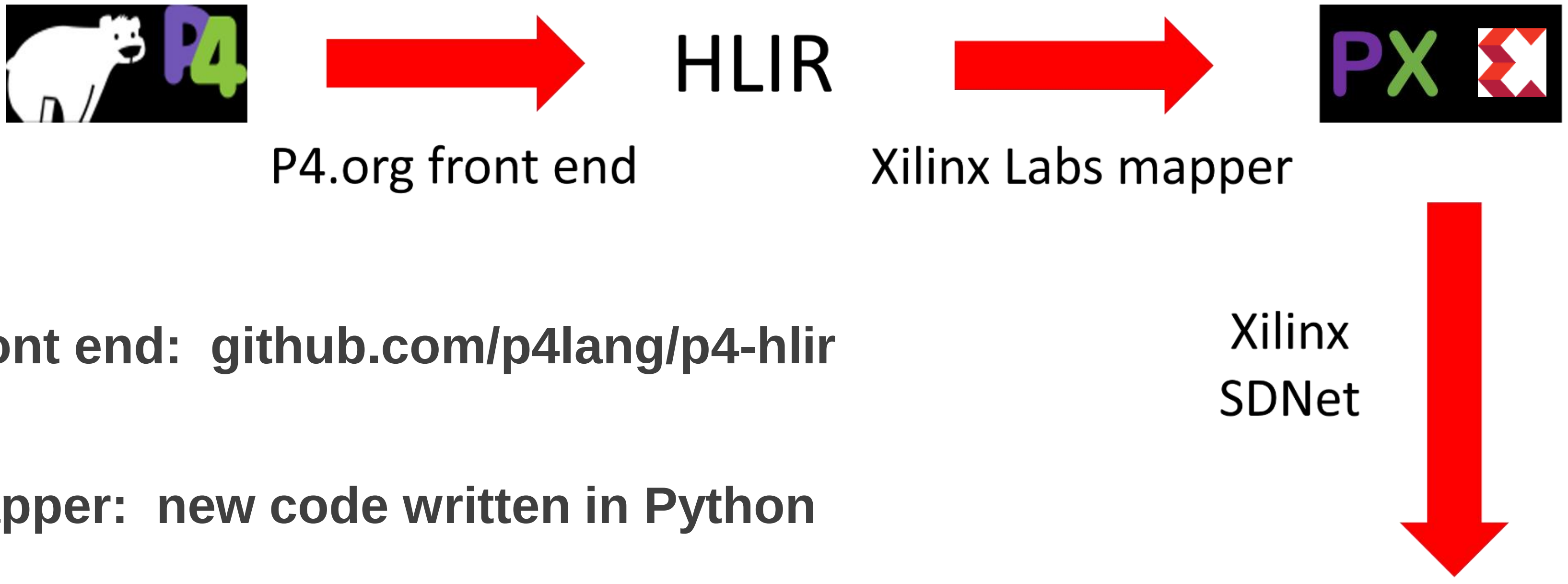
Optimization experiment 2

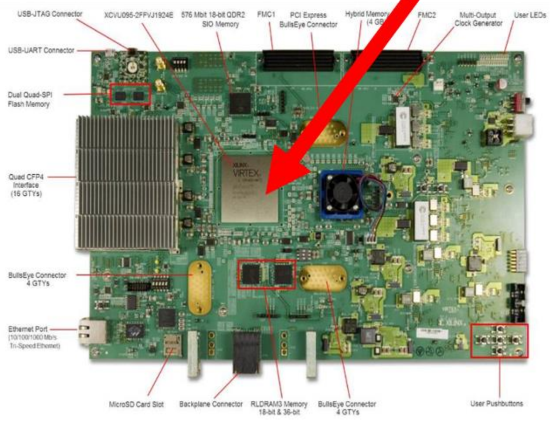
- **Direct mapping to PX:**
  - Parser extracts all fields, and Deparser reconstructs all fields
    - All fields passed along data path
  - MATs have deferred actions on packets:
    - Either direct updates to fields being sent along data path …
    - … or coded deferred actions passed down data path

- **Optimized mapping to PX:**
  - Parser extracts only fields needed for MAT keys, or other expressions
    - Packets passed along data path
  - MATs have immediate actions on packets:
    - Either direct updates to packets being sent along data path …
    - … or coded 'end of processing' actions passed down data path

- **Example impact: 41% resource reduction**

Prototype demonstrated today




USB-JTAG Connector
XCVU095-2FFVJ1924E
576 Mbit 18-bit QDR2 SIO Memory
FMC1
PCI Express BullsEye Connector
Hybrid Memory (4 GB
FMC2
Multi-Output Clock Generator
User LEDs
USB-UART Connector
Dual Quad-SPI Flash Memory
Quad CFP4 Interface (16 GTYs)
BullsEye Connector 4 GTYs
Ethernet Port (10/100/1000 Mb/s Tri-Speed Ethernet)
MicroSD Card Slot
Backplane Connector
RLDRAM3 Memory 18-bit & 36-bit
BullsEye Connector 4 GTYs
User Pushbuttons

- **Front end: github.com/p4lang/p4-hlir**
- **Mapper: new code written in Python**
- **SDNet: next product release version**
- **Xilinx VCU109 development board:**
  - Carries Virtex Ultrascale XCVU095 FPGA
  - Has quad CFP4 interface
    - Being used for 400G Ethernet demos

Prototype demonstrated today

- **Normally, connect board to Ixia tester via single CFP4:**
  - Tester generates input minimum-size packets at 100G line rate
  - Packets pass through fast path generated from P4 description
  - Tester captures output packets at 100G line rate

- **Today, the compact and quiet version:**
  - Test packets written to FPGA
  - On-FPGA block generates input minimum-size packets at 100G line rate
  - Packets pass through fast path generated from P4 description
  - On-FPGA block captures and stores output packets at 100G line rate
  - Captured packets read from FPGA
  - Wireshark shows packets before and after processing

- **MicroBlaze soft processor on FPGA is the on-chip controller**

Protocol Independent Forwarding (PIF)

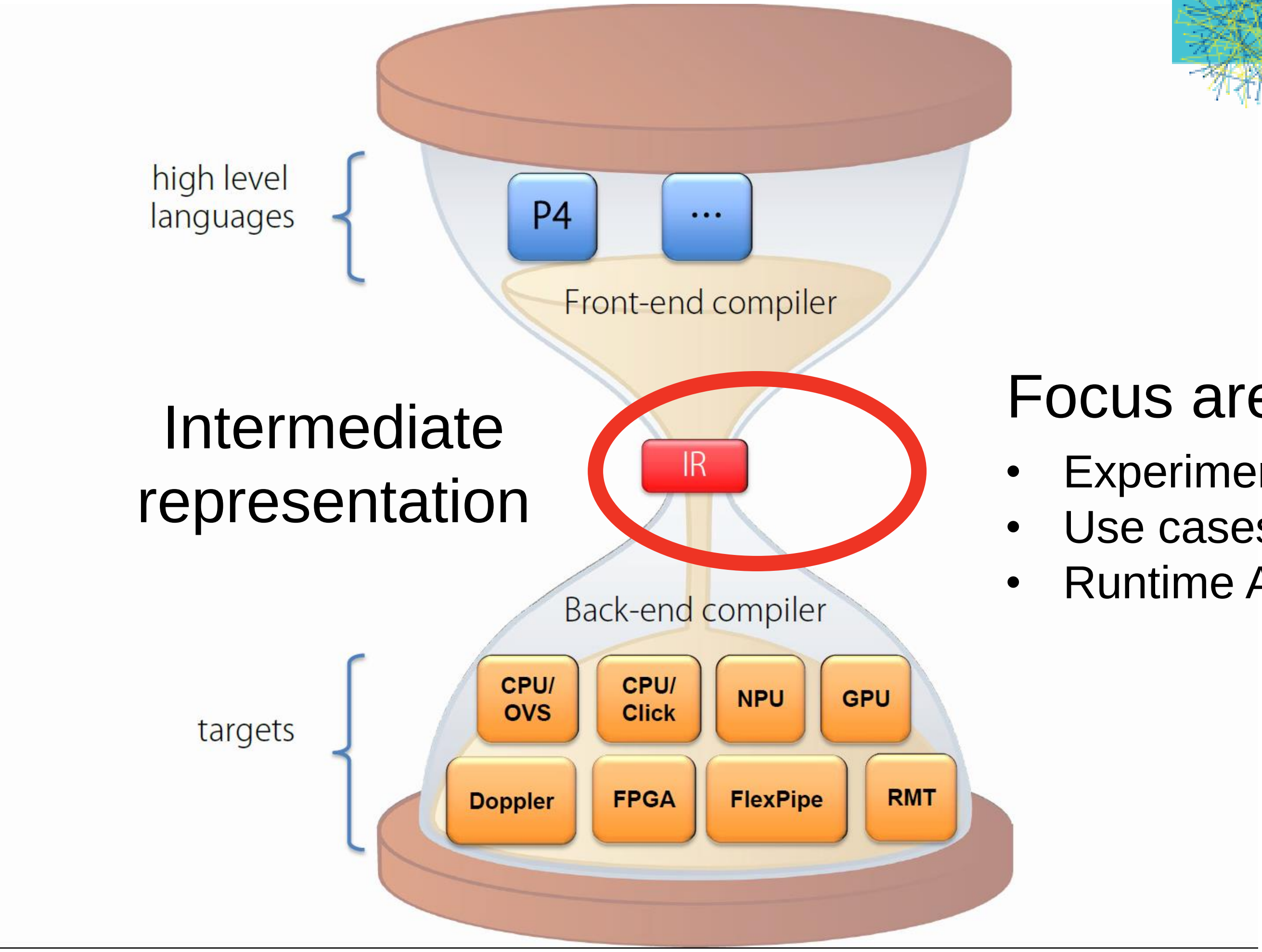
- **"How to tell your plumbing what to do"**
  - Nick McKeown, ONF member workday keynote, September 2014
- **Nick's observation:**
  - Some switches are more programmable than fixed-function ASICs
- **Proposed PIF approach:**
  - **Phase 0.** Initially, the switch does not know what a protocol is, or 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)

PIF open source software project

Within ONF Oct 2014 to Apr 2015, open to all since May 2015

OPEN SOURCE SDN

opensourcesdn.org



Intermediate representation

Focus areas of PIF project:

- Experimenting with IR elements
- Use cases
- Runtime APIs

↓

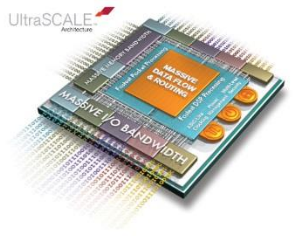
Feeds into ONF "OpenFlow Next Gen" specification activity

P4-to-Xilinx FPGA direct compilation flow

PIF IR

Open source front end

- Moving target …
- Anticipated …
- Moving target …
- Refactoring …
- Range of Xilinx FPGAs



Summary

- FPGA is compelling target technology
  - Combination of programmability with performance

- Demonstration of P4-to-Xilinx flow generating 100G fast path
  - Prototype leveraging existing Xilinx SDNet methodology

- Active participation in P4 evolution
- Active participation in PIF project

- Xilinx invites researchers to undertake collaborative projects

The End
Metro Goldwyn Mayer
THE END
A Warner Bros.-First National PICTURE
The End
FIN