# Programmable Target Architectures for P4

**Gordon Brebner**
**Xilinx Labs**
**San José, USA**

**P4 Workshop, Stanford University, 18 November 2015**

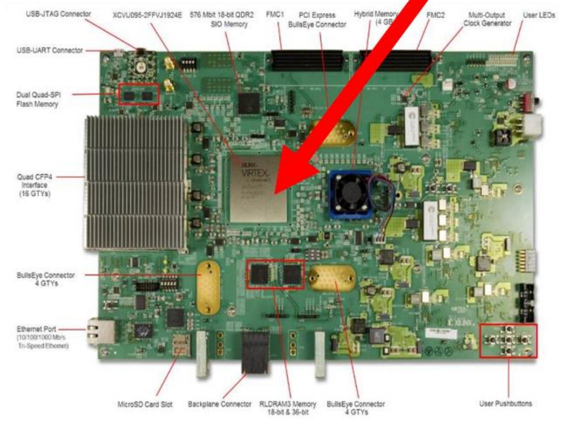
## What this talk is about

- Recap: P4-to-FPGA compilation demonstrated at June workshop
- Architecture/language decoupling discussion
- Click, and extended port/connection semantics
- Xilinx SDNet’s PX data plane building
- Prototype 1:  Generation of P4 includes from PX description
- Prototype 2:  Generation of PX from Extended P4 description

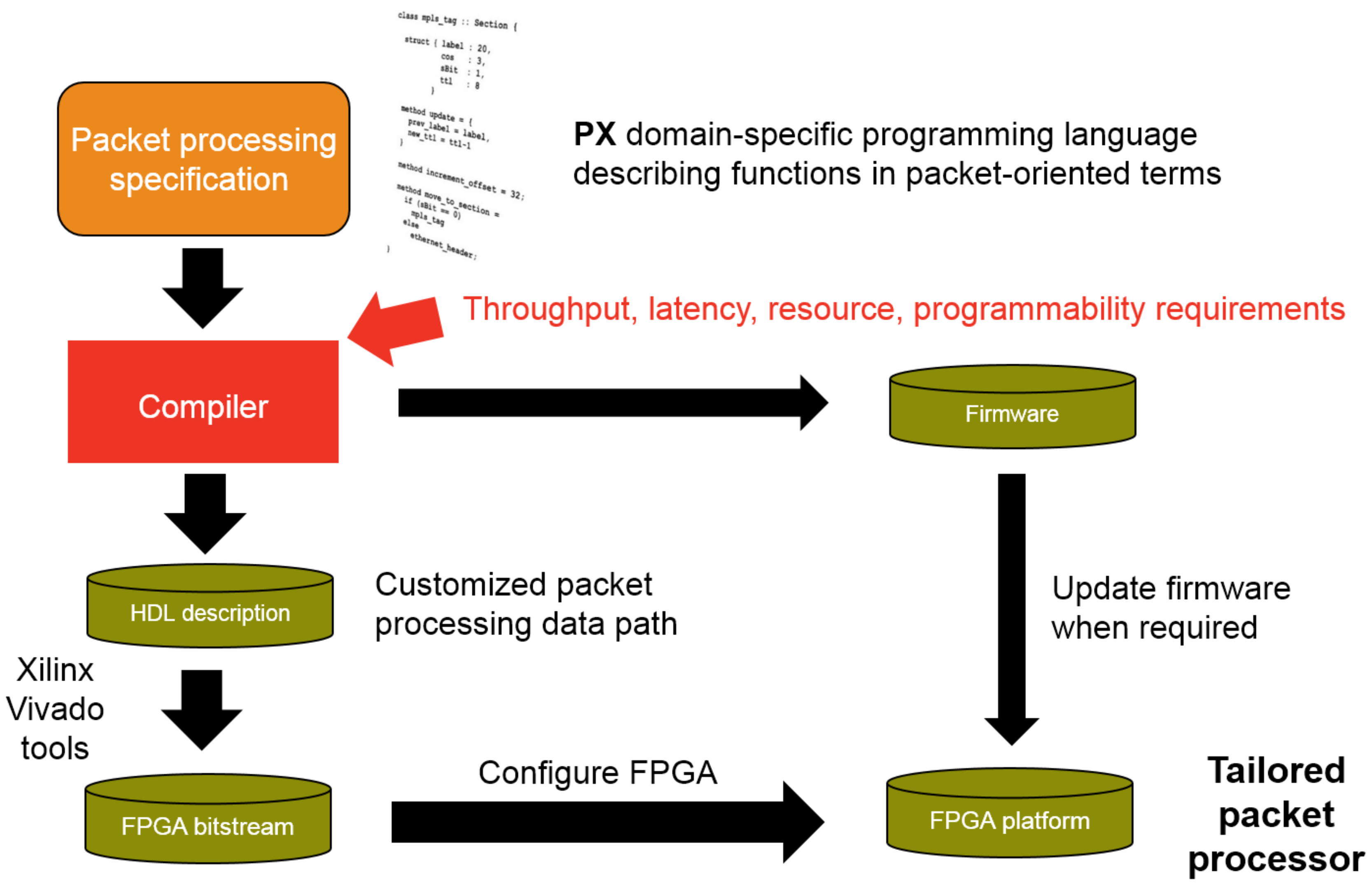
# Prototype demonstrated at June P4 Workshop

$$\begin{tikzcd}
\text{P4} \arrow[r, "\text{P4.org front end}"'] & \text{HLIR} \arrow[r, "\text{Xilinx Labs mapper}"'] & \text{PX} \arrow[d, "\text{Xilinx SDNet}"'] \\
 & & \text{VCU109}
\end{tikzcd}$$

- 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


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

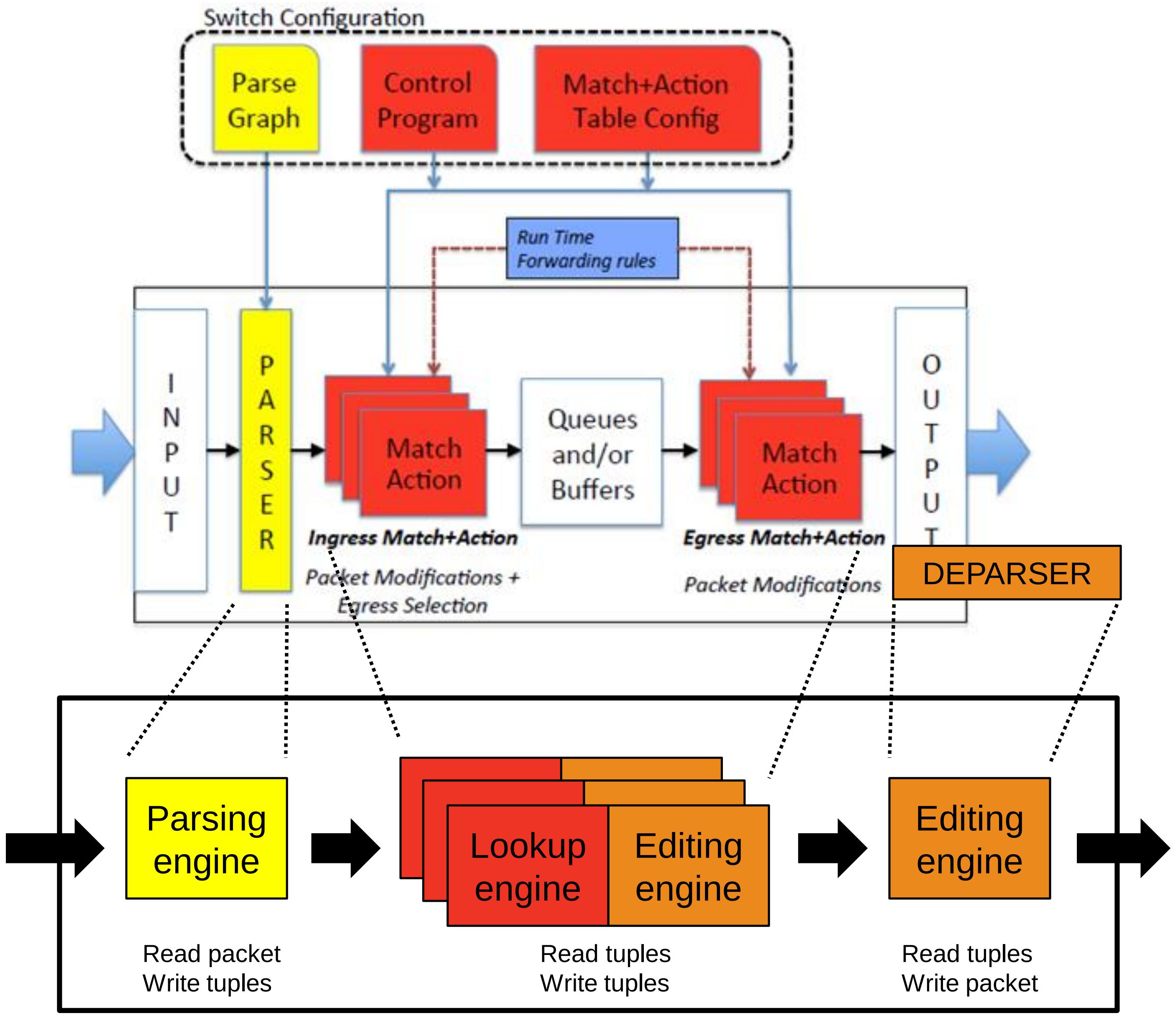
# Xilinx SDNet design flow and use model



```
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;
}
```

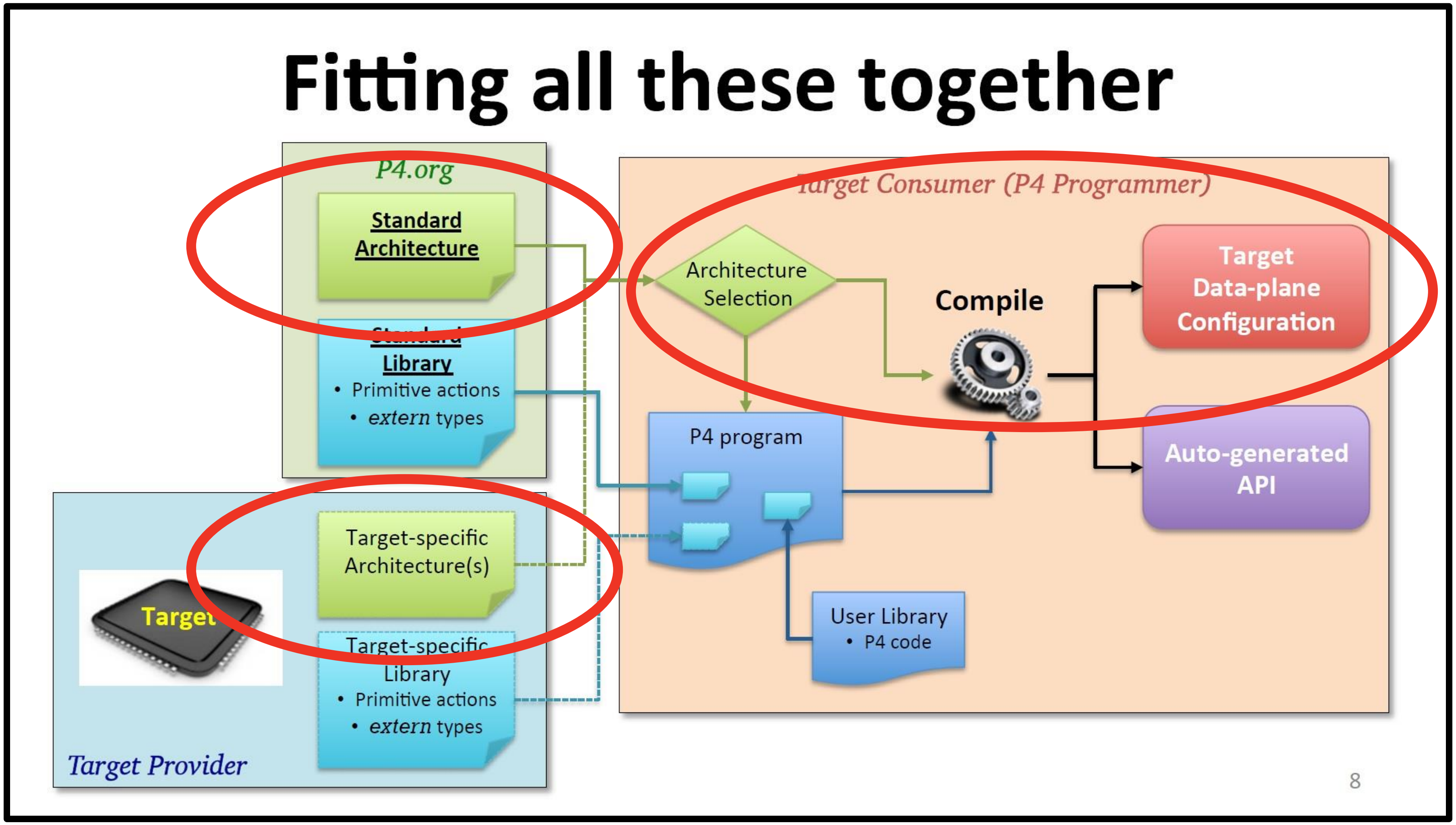
Packet processing specification
**PX** domain-specific programming language describing functions in packet-oriented terms
Throughput, latency, resource, programmability requirements
Compiler
Firmware
HDL description
Customized packet processing data path
Update firmware when required
Xilinx Vivado tools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

# Mapping P4 to PX



# Proposed P4 architecture/language separation

- Slide from Chang's earlier talk today:



# Standard Click, and describing architectures

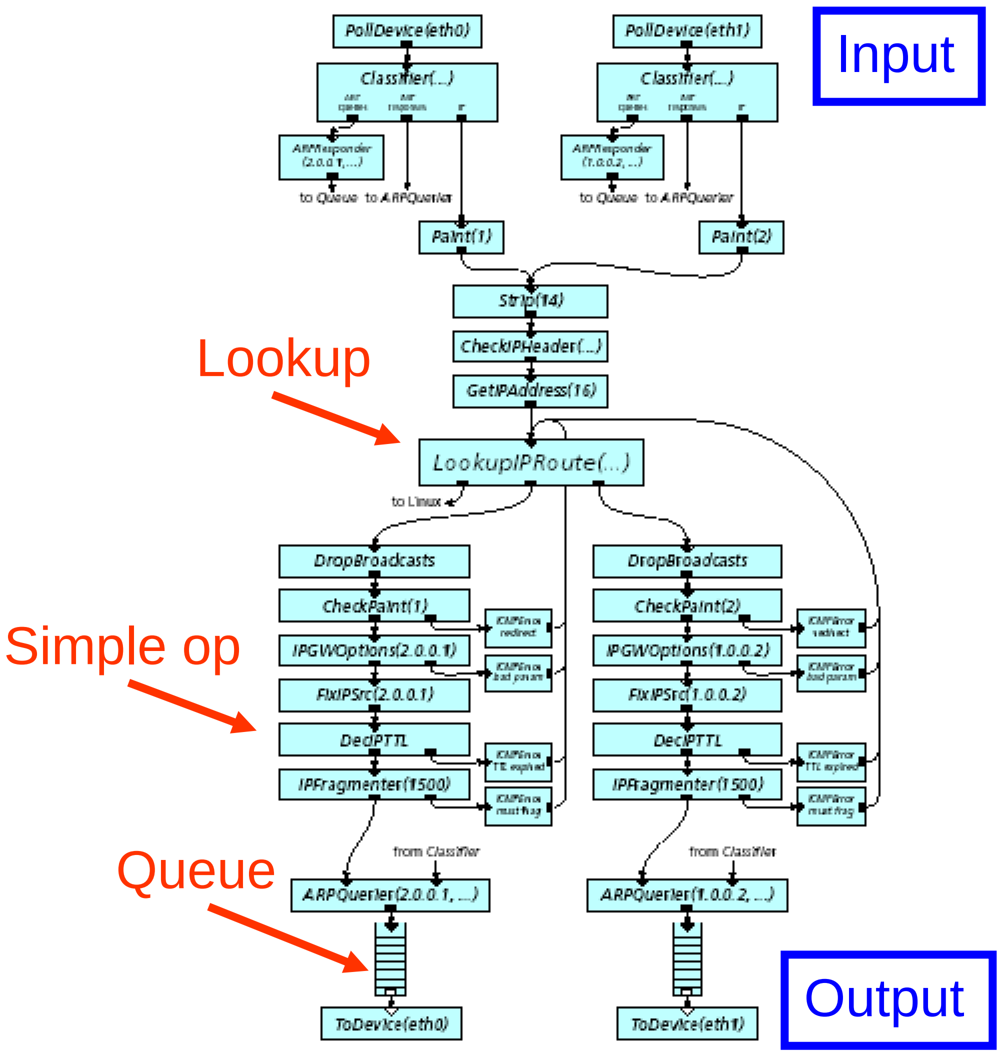
Each box is an instance of a pre-defined Click *element*

*Connections* between elements, and packets ‘pushed’ and ‘pulled’ through the resulting graph

Some adverse characteristics:

- Fixed-function elements
- Fine-grain element functions
- Inter-element interaction:
    - data flow for packets
    - *method calls otherwise*

Click-to-FPGA research (DAC 2004)



## Extended Click connection semantics

- **Xilinx Labs work (FCCM 2010, ACM TRETS 2012)**

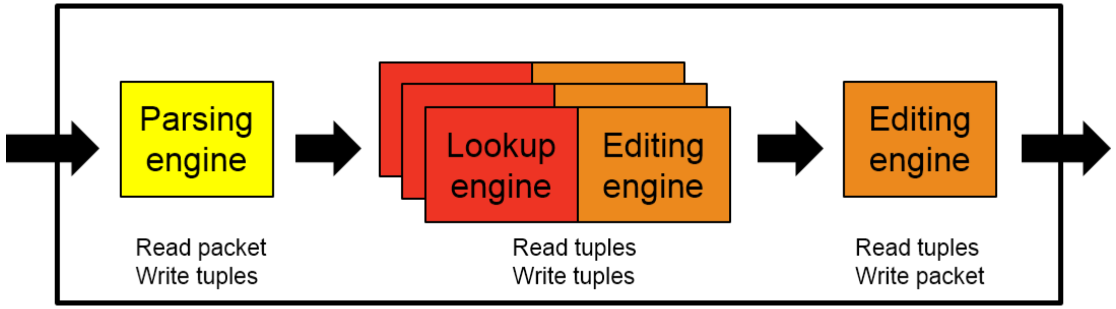
- **Five types of connection between ports:**
  - Packet (as in Standard Click)
  - Tuple (as used for metadata in P4, for example)
  - Access (store access: read and write operations)
  - Compute (procedure call and return)
  - Plain (no semantics, just wiring – low-level escape mechanism)

- **Featured with Click syntax in Xilinx *Packet Xpress* (FPT 2009)**

- **Features with PX syntax in Xilinx SDNet**
  - Only packet and tuple types available to user

# Example: Data plane building in PX
## (Auto-generated from P4 1.0 description by P4-to-PX mapper)

```
class p4_processor :: System {
  Packet_input instream;
  Packet_output outstream;

  Parser parsing_engine1;
  Deparser editing_engine2;
  update_eth_dest lookup_engine3;
  process_ipv4 editing_engine6;
  next_hop_ipv4 lookup_engine5;
  process_eth editing_engine4;

  method connect = {
    parsing_engine1.packet_in = instream,
    outstream = editing_engine2.packet_out,
    editing_engine4.wire2_tupleIn = lookup_engine3.response,
    lookup_engine3.request = parsing_engine1.update_eth_dest_tupleOut,
    editing_engine6.wire1_tupleIn = lookup_engine5.response,
    lookup_engine5.request = parsing_engine1.next_hop_ipv4_tupleOut,
    editing_engine4.standard_metadata_tupleInOut = parsing_engine1.standard_metadata_tupleOut,
    editing_engine4.ethernet_tupleInOut = parsing_engine1.ethernet_tupleOut,
    editing_engine4.ipv4_tupleInOut = parsing_engine1.ipv4_tupleOut,
    editing_engine4.packet_in = parsing_engine1.packet_out,
    editing_engine6.standard_metadata_tupleInOut = editing_engine4.standard_metadata_tupleInOut,
    editing_engine6.ethernet_tupleInOut = editing_engine4.ethernet_tupleInOut,
    editing_engine6.ipv4_tupleInOut = editing_engine4.ipv4_tupleInOut,
    editing_engine6.packet_in = editing_engine4.packet_out,
    editing_engine2.standard_metadata_tupleIn = editing_engine6.standard_metadata_tupleInOut,
    editing_engine2.ethernet_tupleIn = editing_engine6.ethernet_tupleInOut,
    editing_engine2.ipv4_tupleIn = editing_engine6.ipv4_tupleInOut,
    editing_engine2.packet_in = editing_engine6.packet_out
  }
}
```



## Prototype 1:  Generate P4 includes from PX  (1)

- Mark certain PX components with P4 significance

- P4 engine: a component described in P4

```
class Parser :: P4Engine (parser) {
    StandardOut standard;
    EthernetOut ethernet;
    IPv4Out ipv4;

    method classify = { standard=metadata, ethernet=header, ipv4=header }
}
```

- P4 procedure: an extern made available to P4 components

```
class IPv4Checksum :: P4Procedure {
    IPv4InOut ipv4;

    ChecksumEngine checksum;

    method apply = {
        checksum.ipv4 = ipv4
    }
}
```

## Prototype 1: Generate P4 includes from PX (2)

- Target-specific P4 includes generated by PX compiler
- Header_type declarations (e.g. Standard_t)
- From P4 engine (P4-discuss syntax):

```
whitebox_type Parser (
    out metadata Standard_t standard,
    out header Ethernet_t ethernet,
    out header IPv4_t ipv4
);
```

- From P4 procedure (P4 1.1 syntax):

```
extern_type IPv4Checksum {
      method apply(IPv4_t ipv4);
}
```

## Prototype 2: Generate PX from Extended P4 (1)

- **Target architecture described in P4 "1.arch":**
  - Header type declarations (standard P4 1.1 syntax)
  - *Element* type declarations (Click-inspired: target architecture components)
    - Note: what was generated from PX in Prototype 1
  - Extern type declarations (standard P4 1.1 syntax)
    - Note: what was generated from PX in Prototype 1
  - *Architecture* type declaration (Click-inspired: overall target architecture)
    - Input and output ports
    - User-provided *intern* elements
    - Target-supplied elements
    - Connections between elements
    - Target-provided *extern* objects

## Prototype 2:  Generate PX from Extended P4  (2)

- **Example element type declarations**

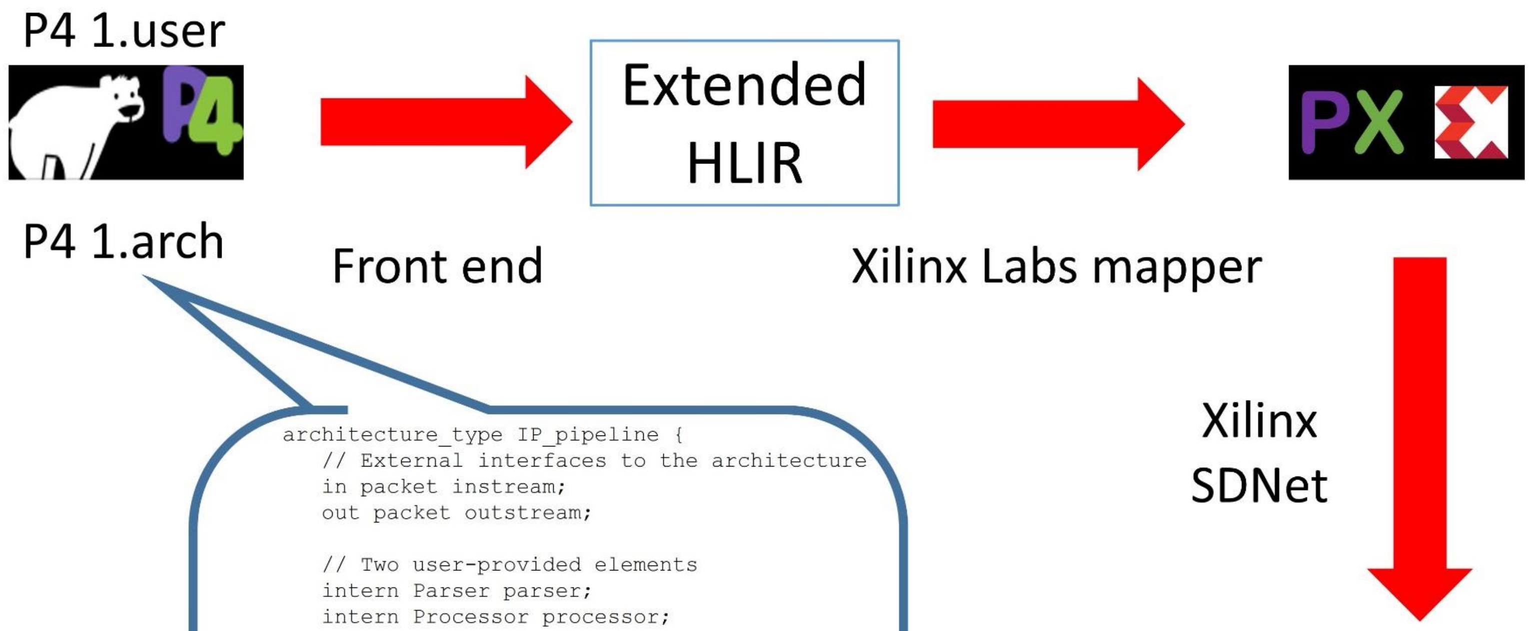
```
element_type Parser (
    in packet packet_in,
    out metadata standard_metadata_t standard,
    out header ethernet_t ethernet,
    out header ipv4_t ipv4
);


element_type Processor (
    inout metadata standard_metadata_t standard,
    inout header ethernet_t ethernet,
    inout header ipv4_t ipv4
);
```

## Prototype 2: Generate PX from Extended P4 (3)

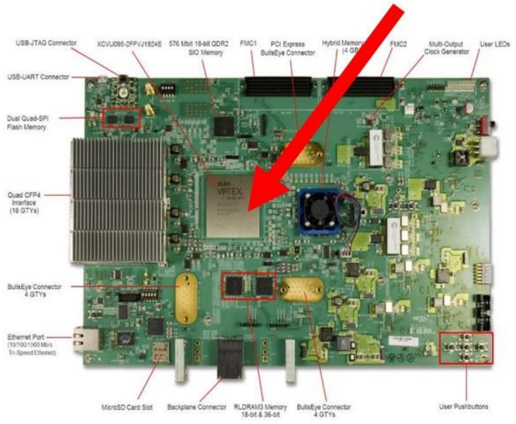
- **Example architecture type declaration**

```
architecture_type IP_pipeline {
   // External interfaces to the architecture
   in packet instream;
   out packet outstream;

   // Two user-provided elements
   intern Parser parser;
   intern Processor processor;

   // One supplied element
   Deparser deparser;

   // Connectivity of the architecture
   connections {
      (instream, parser.packet_in),                 // Input to parser
      (parser.standard, processor.standard),        // \
      (parser.ethernet, processor.ethernet),        //  Parser to processor
      (parser.ipv4, processor.ipv4),                // /
      (processor.standard, deparser.standard),      // \
      (processor.ethernet, deparser.ethernet),      //  Processor to deparser
      (processor.ipv4, deparser.ipv4),              // /
      (deparser.packet_out, outstream)              // Deparser to output
   }

   // One provided extern
   extern checksum ipv4_checksum;
}
```

# Prototype 2 demonstrated today at 100 Gb/s rate



```
architecture_type IP_pipeline {
   // External interfaces to the architecture
   in packet instream;
   out packet outstream;

   // Two user-provided elements
   intern Parser parser;
   intern Processor processor;

   // One supplied element
   Deparser deparser;

   // Connectivity of the architecture
   connections {
      (instream, parser.packet_in),
      (parser.standard, processor.standard),
      (parser.ethernet, processor.ethernet),
      (parser.ipv4, processor.ipv4),
      (processor.standard, deparser.standard),
      (processor.ethernet, deparser.ethernet),
      (processor.ipv4, deparser.ipv4),
      (deparser.packet_out, outstream)
   }

   // One provided extern
   extern IPv4Checksum checksummer;

}
```


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

## Summary

- **P4 heading towards architecture/language separation**
  - As explained by Chang earlier today
- **Strawman proposal for describing architecture using P4 “1.arch”**
  - As opposed to using English – more precise, and machine processible
  - Inspired by the well-known Click element/connection paradigm
- **Demonstration of compiling P4 “1.arch” to FPGA at 100G rate**
  - Including user P4 components written in P4 “1.user” (more-or-less 1.1)
- **Xilinx Labs invites researchers to undertake collaborative projects**

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