STEPHEN IBANEZ (STANFORD UNIVERSITY)
GORDON BREBNER (XILINX LABS)

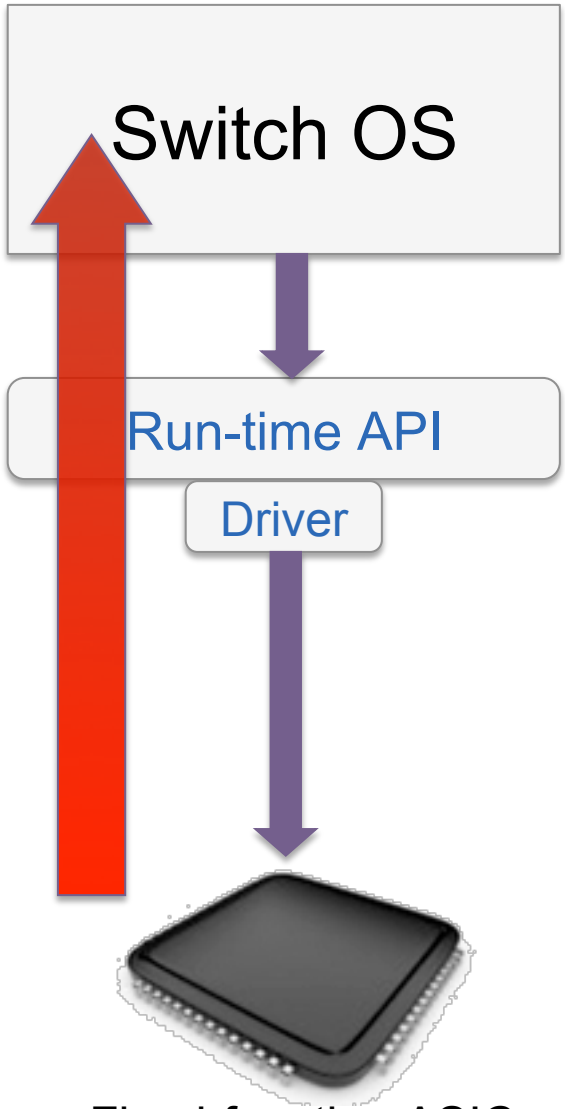
Outline

- P4 Introduction
- P4-NetFPGA Workflow Overview
- P4 Compilation Using Xilinx P4-SDNet
- Future Work
- P4-NetFPGA Example

Packet Processing Status Quo

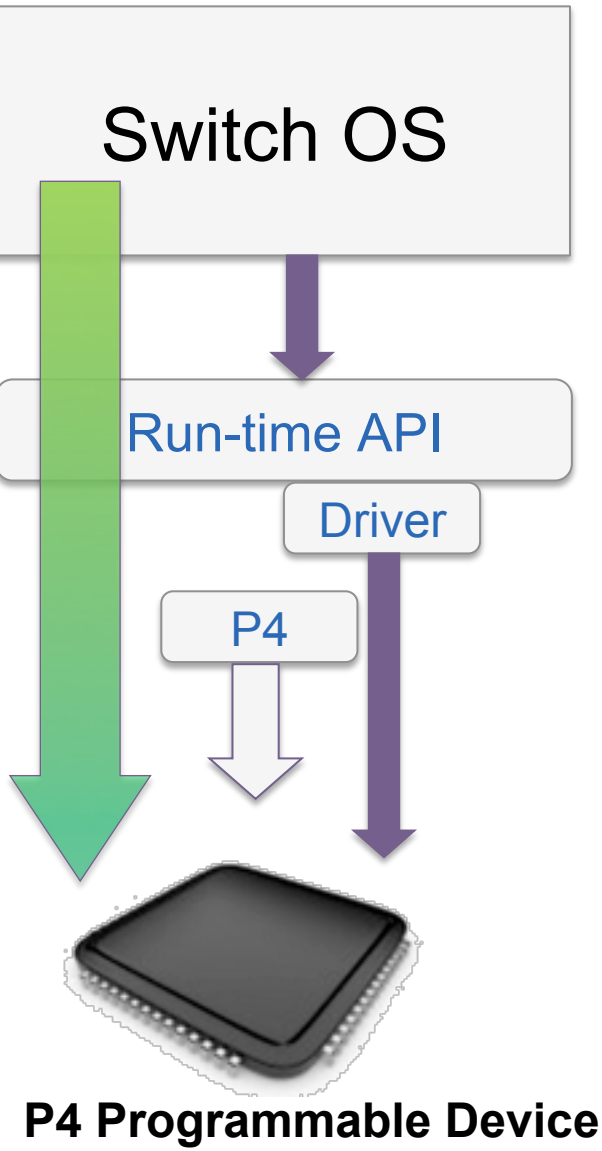
- Bottom-up design philosophy
- Prone to bugs
- Long and unpredictable lead time
- Protocols evolve very quickly

"This is roughly how I process packets…"
– chip datasheet



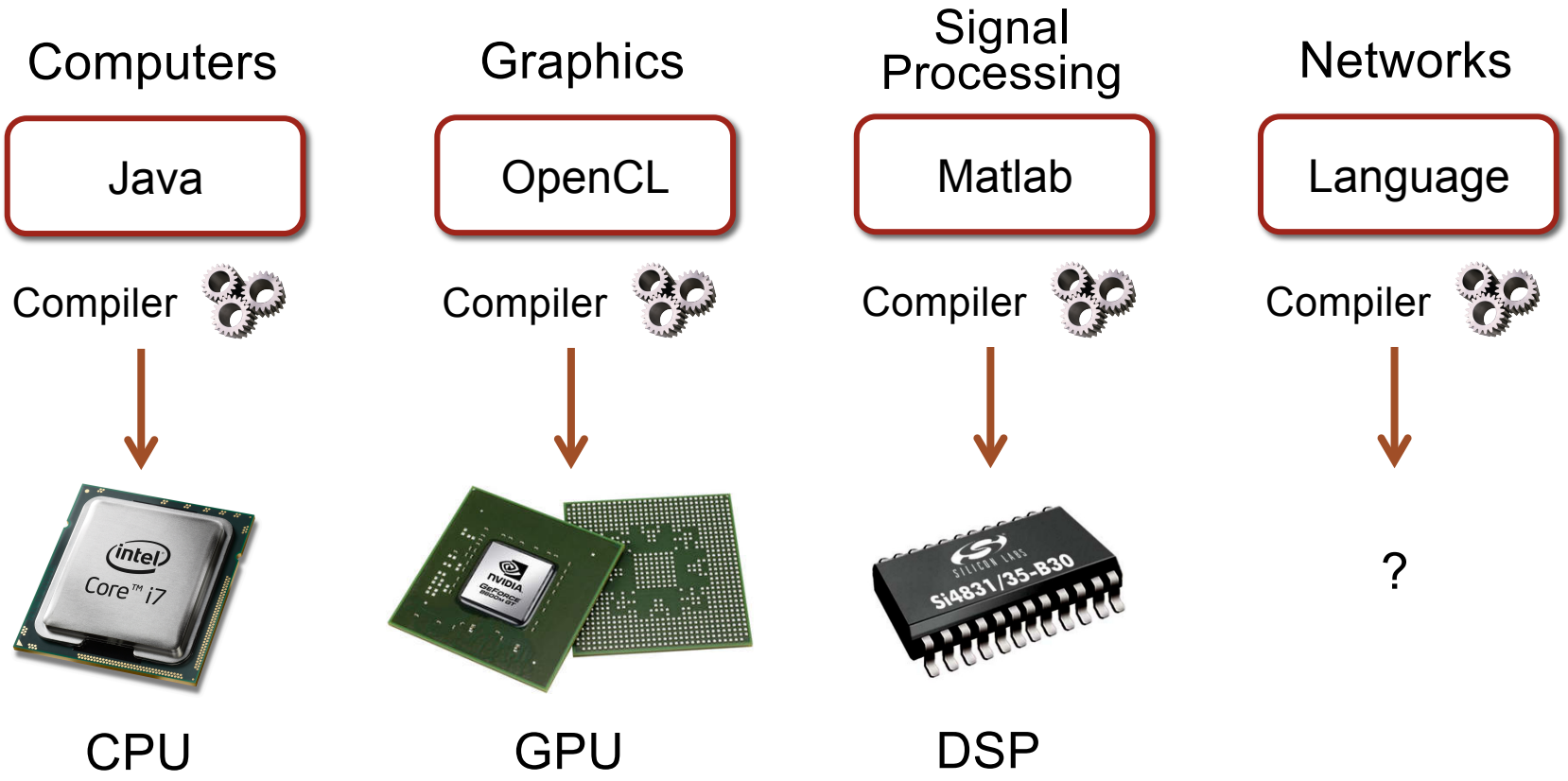
P4 Design Philosophy

- Top-down approach
- User / Controller makes the rules

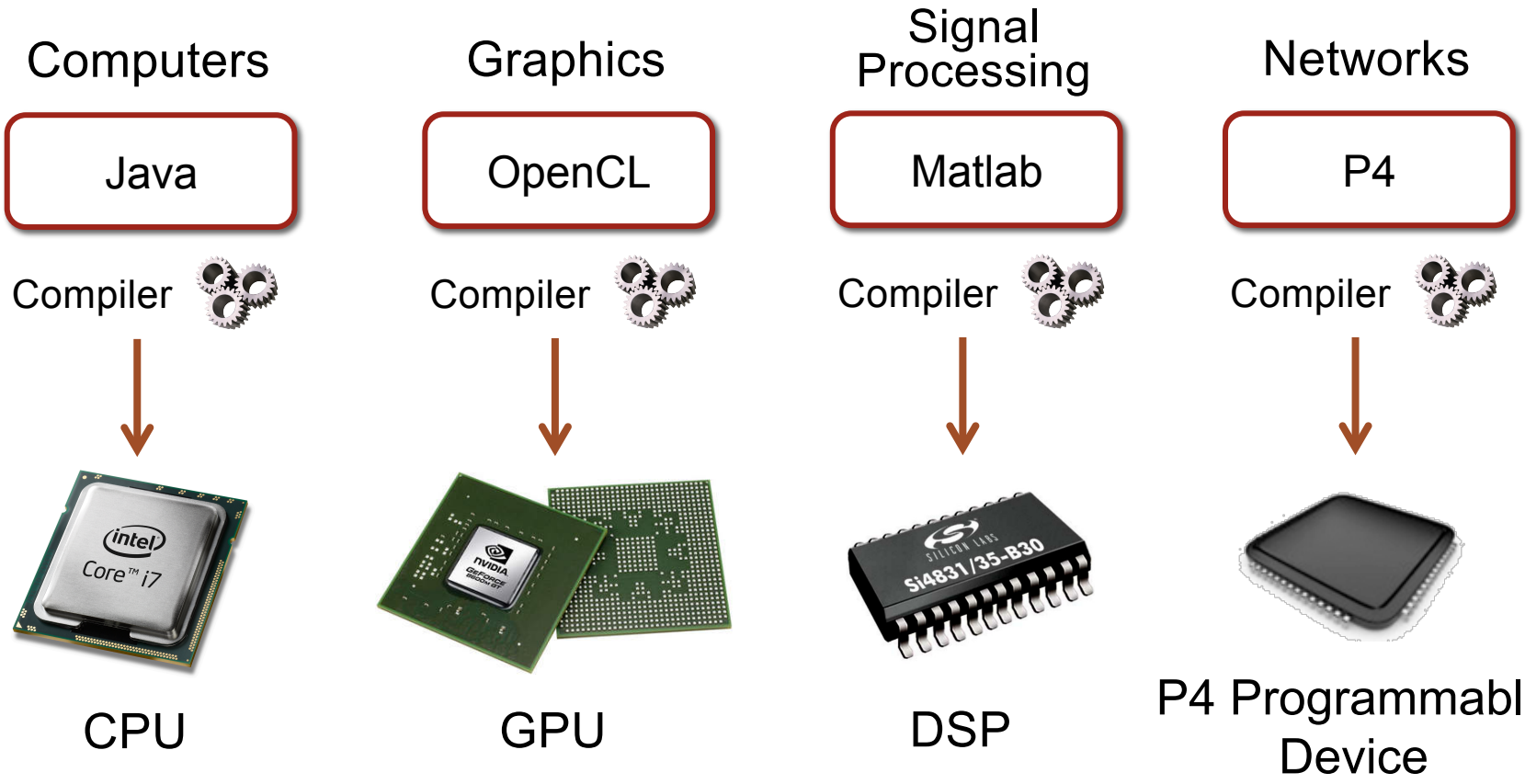


"This is exactly how you must process packets"
– P4 Program/Control Plane

Domain Specific Processors



Domain Specific Processors



What is P4?

- An open source language allowing the specification of packet processing logic, as well as API generation
- First appeared in paper published in ACM CCR July 2014
  - Motivated by limitations of OpenFlow static specification

- Current version: $P4_{14}$; Imminent new version (May 2017): $P4_{16}$
- Open P4 language consortium at P4.org
  - Currently has around 60 members
- P4 events at Stanford in Jun and Nov 2015, and May and Nov 2016
  - All oversubscribed

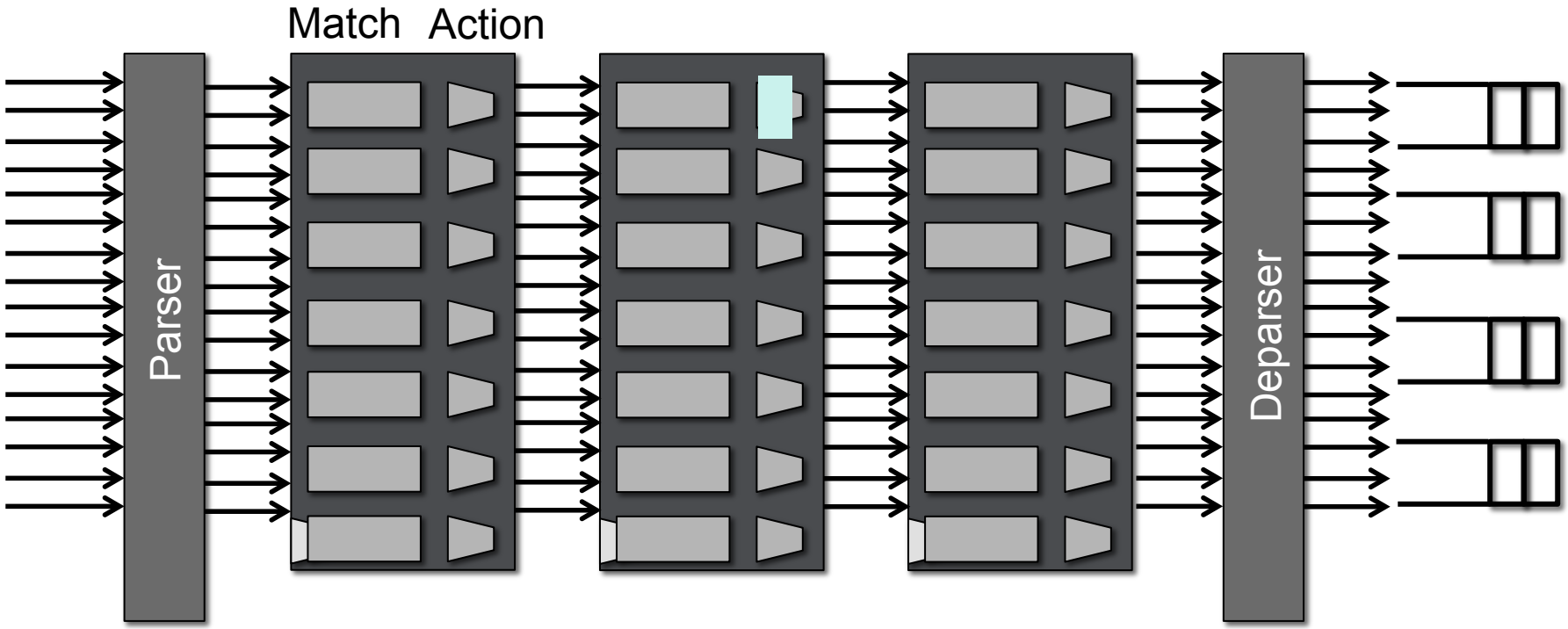
P4 Language Components

State-machine;
Field extraction

Table lookup and update;
Field manipulation;
Control Flow

Field assembly

Abstract Forwarding Model



Programmable Network Devices as P4 Targets

- PISA: Flexible Match+Action ASICs
  - Barefoot Tofino, Intel Flexpipe, Cisco Doppler, etc.
- NPU
  - Netronome, EZchip, …
- CPU
  - Open vSwitch, VPP, …
- FPGA
  - Xilinx P4-SDNet, P4FPGA, …

Benefits of Programmable Forwarding

- New Features – Add new protocols
- Reduce complexity – Remove unused protocols
- Efficient use of resources – flexible use of tables
- Greater visibility – New diagnostic techniques, telemetry, etc.
- SW style development – rapid design cycle, fast innovation, fix data-plane bugs in the field
- You keep your own ideas

Think programming rather than protocols…

Many P4 Applications Already

- In-band Network Telemetry — the killer app
- Network state aware adaptive routing – CLOVE, HULA
- Paxos
- Heavy hitter detection
- … and many more

In-band Network Telemetry (INT)

1. “Which path did my packet take?”
2. “Which rules did my packet follow?”
3. “How long did it queue at each switch?”
4. “Who did it share the queue with?”

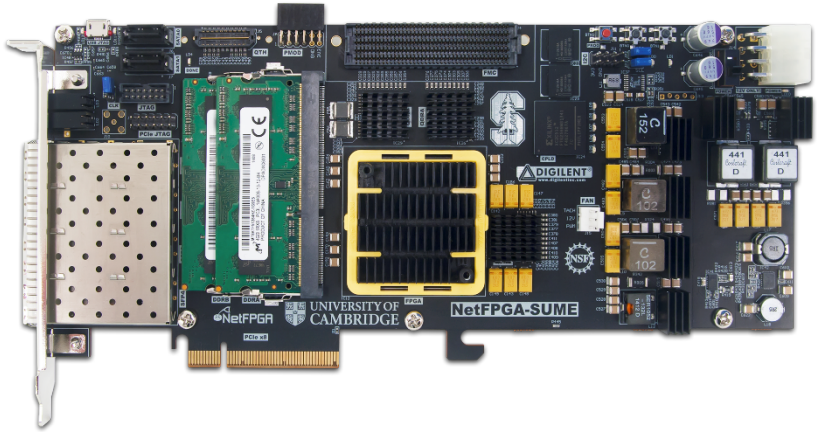
Can questions at full line-rate, with no additional packets!

P4-NetFPGA Workflow Overview

So what is NetFPGA?
NetFPGA = Networked FPGA

- A line-rate, flexible, open networking platform for teaching and research

SUME



=

- Network Interface Card
- Hardware Accelerated Linux Router
- IPv4 Reference Router
- Traffic Generator
- Openflow Switch
- More Projects
- Add Your Project

P4-NetFPGA Workflow Goals

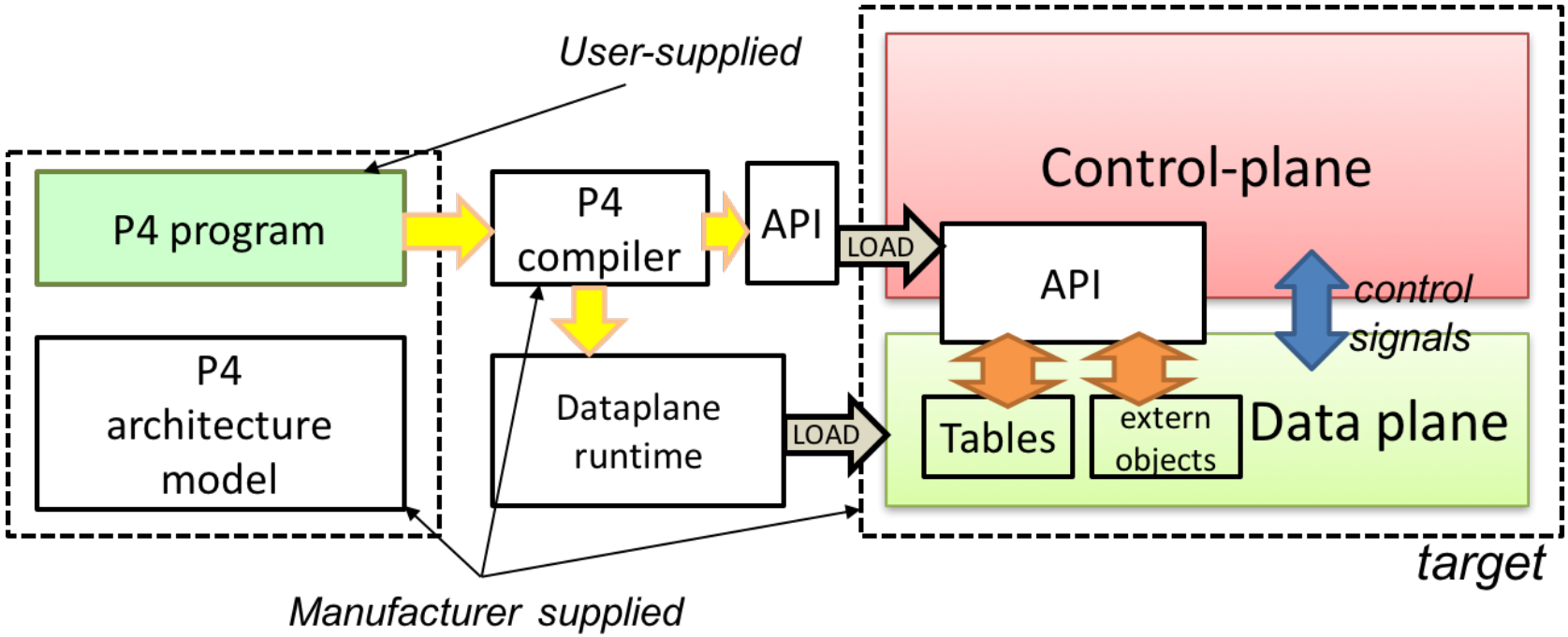
- No need for developers to know HDL or FPGA details
- Easy to write P4 programs for NetFPGA SUME
- Easy to debug
- Easy to compile/run those programs on SUME
- Easy for community to contribute (e.g. externs, architecture elements)
- Experience P4 on real hardware at line-rate

P4-NetFPGA Availability

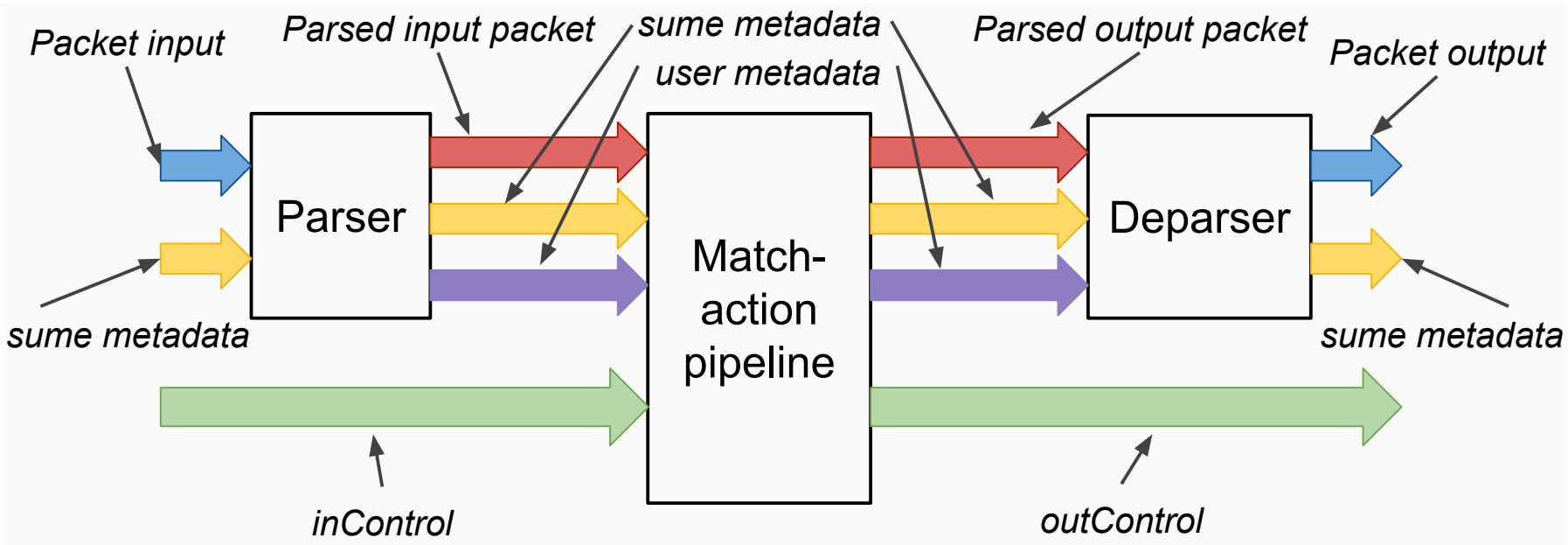
- General access release June 2017, after P4 Workshop
- When $P4_{16}$ language specification is released
- When P4-SDNet 2017.1 released

- Tutorial Events:
  - P4 Developer Day @ Stanford – May 16, 2017
  - NetFPGA Summer Camp @ Cambridge – Summer 2017
  - SIGCOMM @ UCLA – August 21-25

General Process for Programming a P4 Target



SimpleSumeSwitch Architecture Model for SUME Target

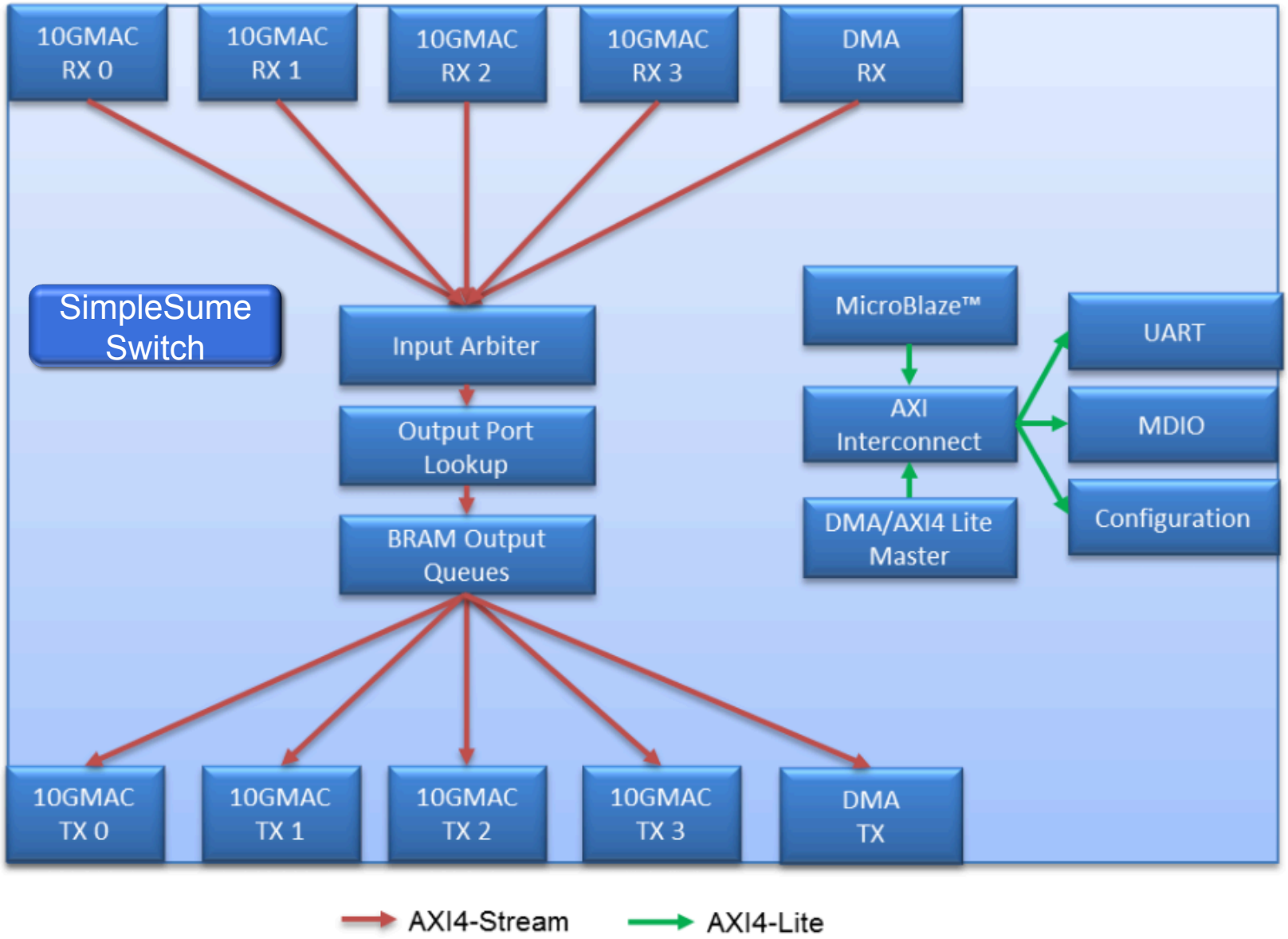


P4 used to describe parser, match-action pipeline, and deparser

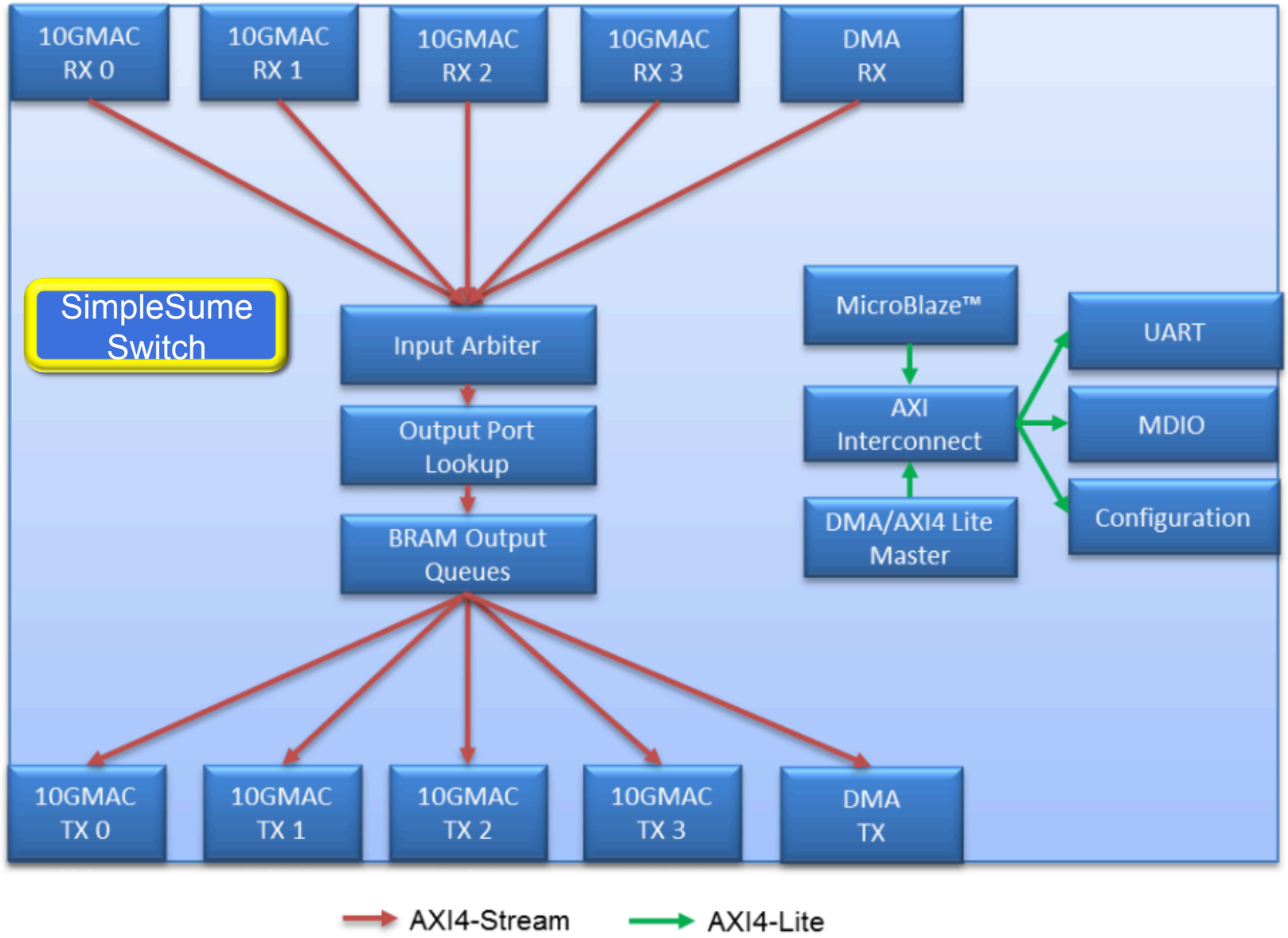
Standard SUME Metadata in Architecture Model

```
/* standard sume switch metadata */
struct sume_metadata_t {
    bit<16> pkt_len; // unsigned int
    port_t src_port; // one-hot encoded
    port_t dst_port; // one-hot encoded
    bit<8> drop;
    bit<8> send_dig_to_cpu; // send digest_data to CPU
    digest_metadata_t digest_data;
}


// digest metadata to send to CPU
struct digest_metadata_t {
    bit<8> src_port;
    bit<48> eth_src_addr;
    bit<24> unused;
}
```

P4 Architecture Model Plugs Into SUME Reference Switch



P4 Architecture Model Plugs Into SUME Reference Switch

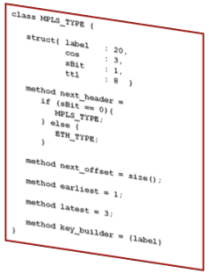


P4 Compilation Using Xilinx P4-SDNet

Xilinx SDNet programmable packet processor
(www.xilinx.com/sdnet)

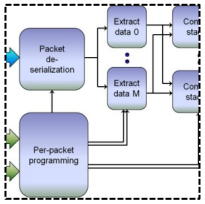
Headline feature set, uniquely enabled by FPGA ‘white box hardware’ target:

Scalable 1G to 100G line rate performance



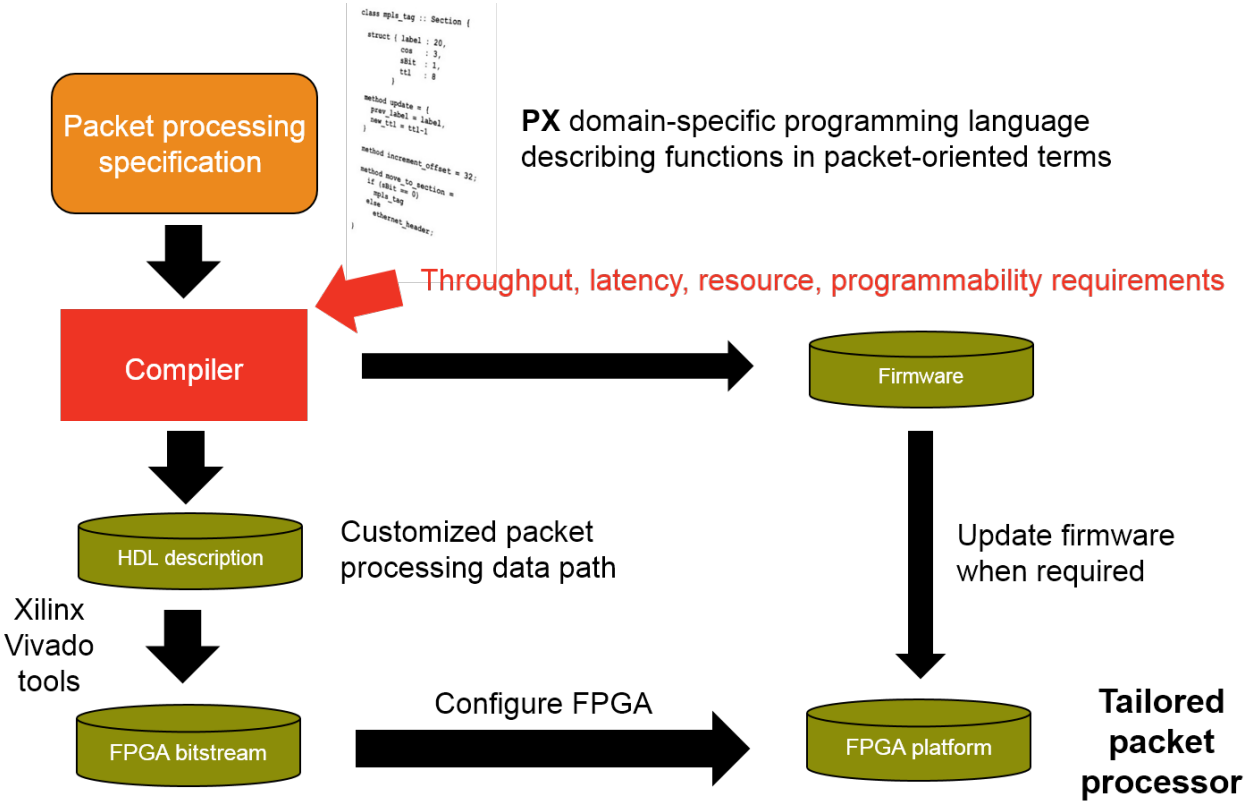
Domain-specific programming abstraction

Exact-fit hardware for reduced cost and power



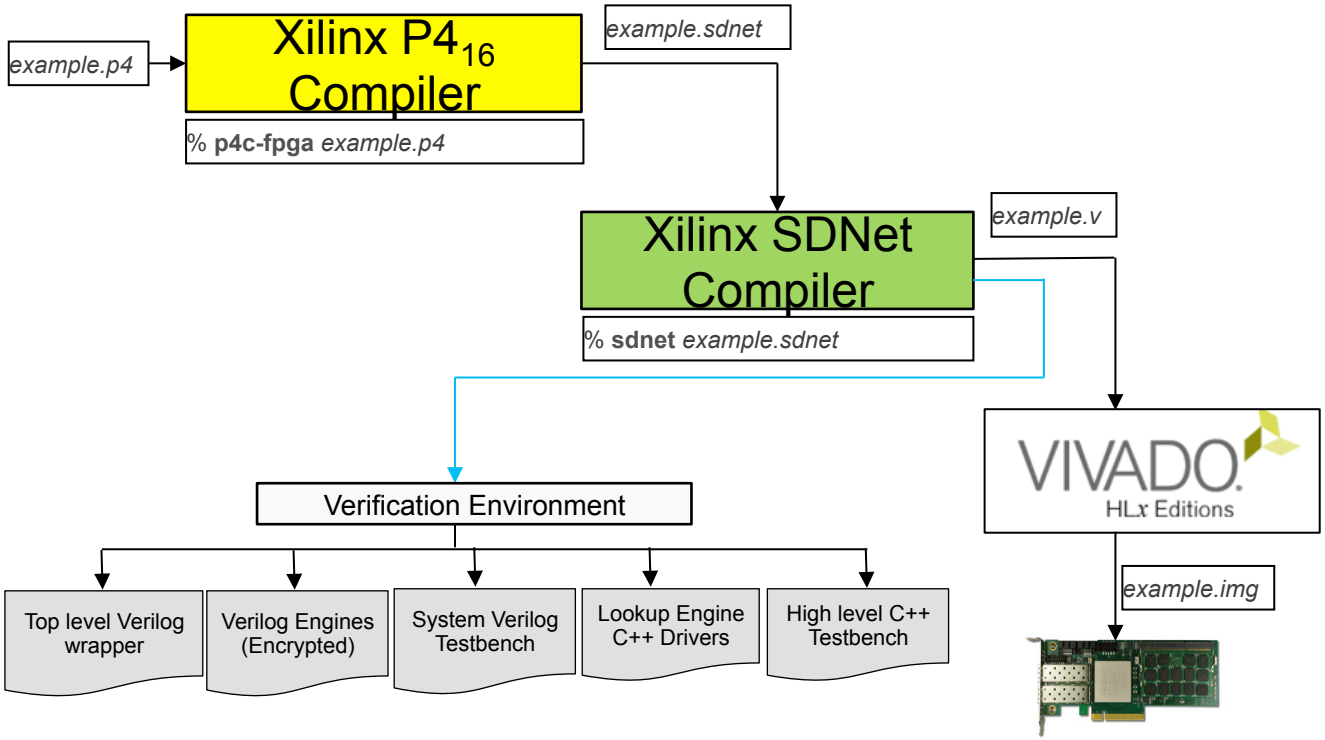
Firmware for run time programmability

SDNet Design Flow and Use Model using PX language



Xilinx P4-SDNet Design Flow

Included in SDNet 2016.4 release, February 2017



P4-SDNet Compilation in P4-NetFPGA Workflow

- User P4 code is compiled with respect to SimpleSumeSwitch Architecture Model:
  - Code for Parser, Match-Action Pipeline, and Deparser
- Compiler outputs Verilog module for whole P4-described system
- Module has standard AXI-S packet input/output interfaces
- Output is engineered for 100G rate >> SUME switch aggregate 40G rate
- Supports P4 extern feature for user defined logic

Future Work

Programmable Target Architectures for NetFPGA

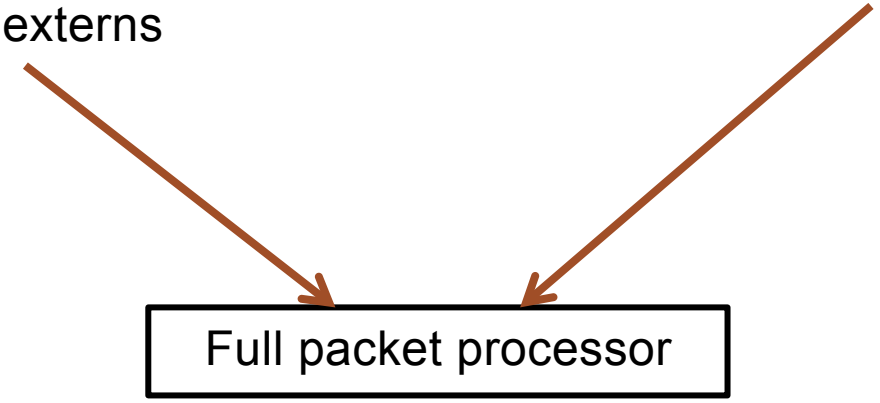
- P4 Language extensions
  - Explicitly define connections between architecture elements ($P4_{16+}$)
- Will be two actors:

Target Architecture Designer

- $P4_{16+}$ architecture description
- Implementation of non-P4-defined elements and externs

P4 programmer

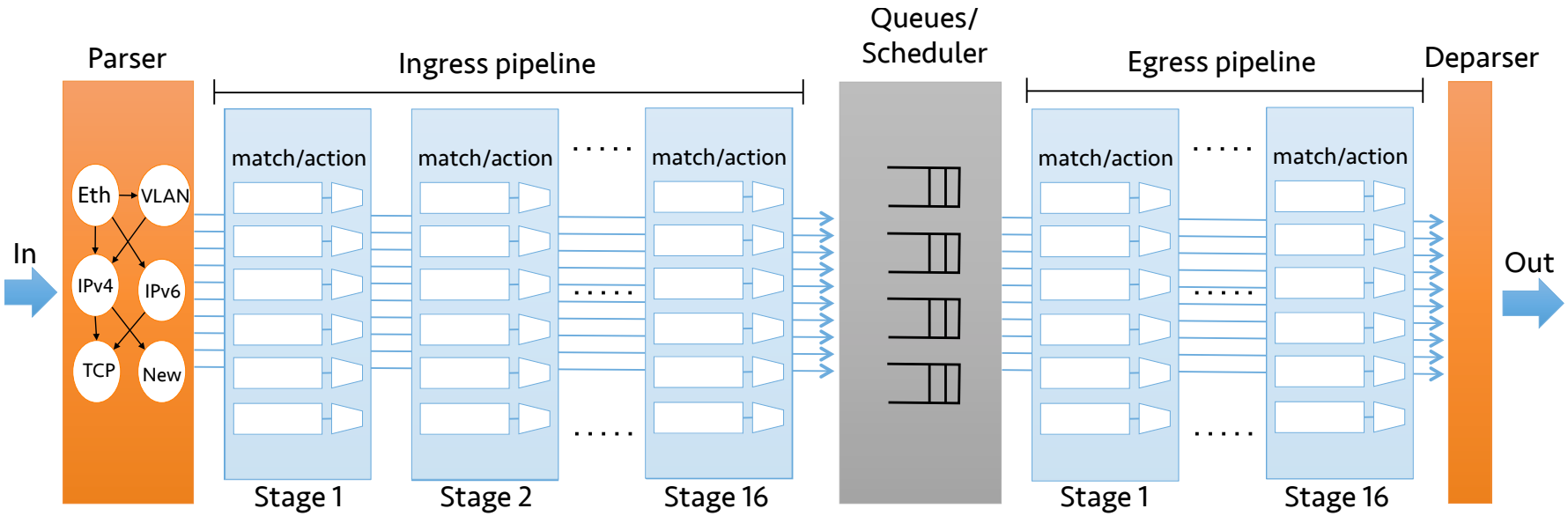
- Implementation of P4 elements in target architecture



Programmable Target Architectures for NetFPGA

- Motivation:
  - Take advantage of FPGA's ability to support multiple architectures
  - Study performance tradeoffs of various architectures
  - Allow incorporation of experimental architecture components and (P4-style or not) tool flows for creating them

Another possible architecture:

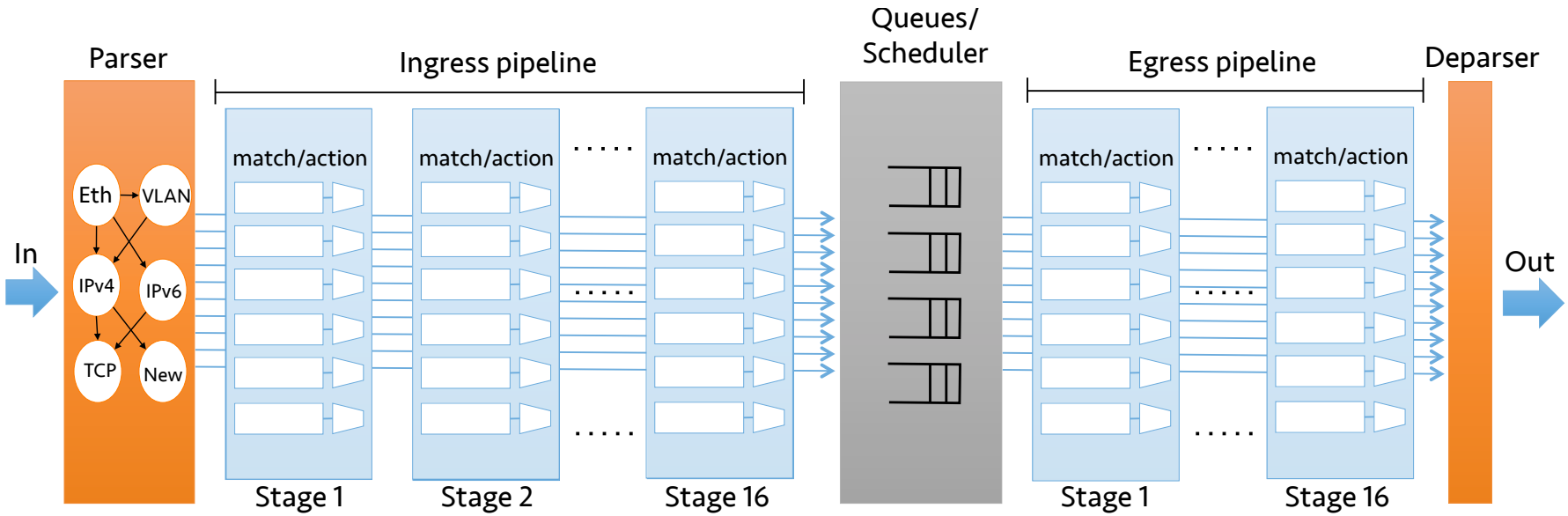


Integrate with Open Source Network Tester (OSNT)

- Need robust system for performing hardware tests
- Software based solutions (such as MoonGen) can be great at traffic generation, but have poor monitoring capabilities
- OSNT can provide both, but needs programmable parser
- Instantiate both switch architecture and OSNT architecture with P4 modules

Programmable Scheduling

- Experimental P4 extensions to describe traffic management
- Based on PIFO model (SIGCOMM 2016)
- Potential drop-in replacement for SUME output queueing module



FIN

P4-NetFPGA Example

Learning by example – L2 Learning Switch

- Parses Ethernet frames
- Forwards based on Ethernet destination address
- Frame broadcast (with ingress port filtering) if address not in forwarding database
- Learns based on Ethernet source address
- If source address is unknown, the address and the source port are sent to the control-plane (which will add an entry to the forwarding database)

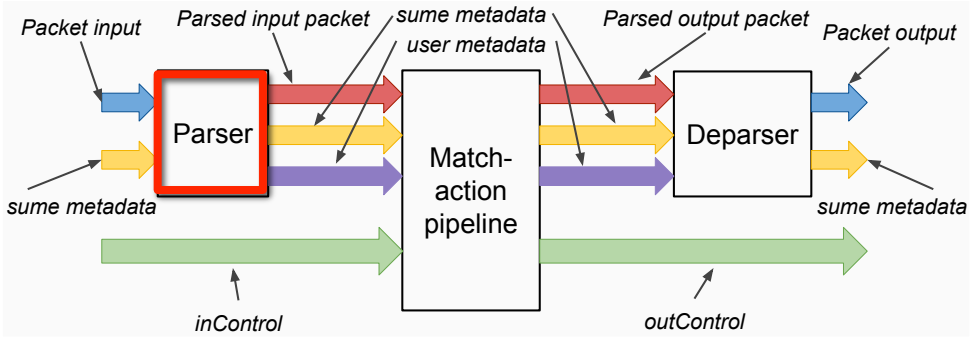
Learning Switch – Header definitions

```
// standard Ethernet header
header Ethernet_h {
    EthernetAddress dstAddr;
    EthernetAddress srcAddr;
    bit<16> etherType;
}

// IPv4 header without options
header IPv4_h {
    bit<4> version;
    bit<4> ihl;
    bit<8> diffserv;
    bit<16> totalLen;
    bit<16> identification;
    bit<3> flags;
    bit<13> fragOffset;
    bit<8> ttl;
    bit<8> protocol;
    bit<16> hdrChecksum;
    IPv4Address srcAddr;
    IPv4Address dstAddr;
}
```

```
// List of all recognized headers
struct Parsed_packet {
    Ethernet_h ethernet;
    IPv4_h ip;
}
```

- Explicit specification of headers, fields, and their bit widths
- The headers that can be parsed, manipulated, or created by the switch

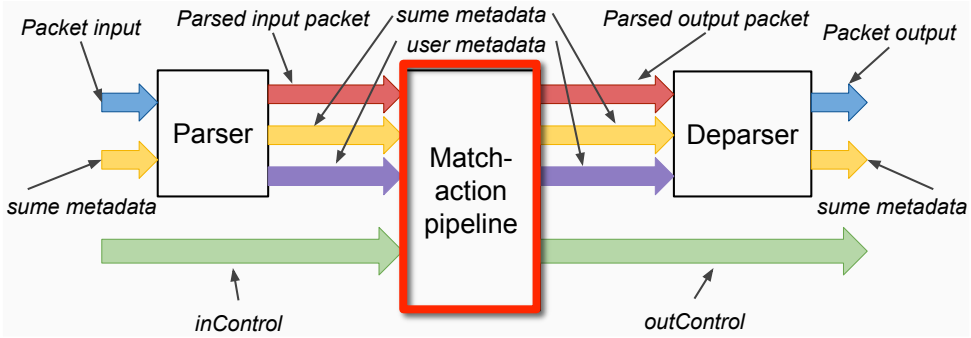
Learning Switch – Parser




```
// Parser Implementation
parser TopParser(packet_in b,
                 out Parsed_packet p,
                 out user_metadata_t user_metadata,
                 inout sume_metadata_t sume_metadata) {
    state start {
        b.extract(p.ethernet);
        transition select(p.ethernet.etherType) {
            IPV4_TYPE: parse_ipv4;
            default: reject;
        }
    }

    state parse_ipv4 {
        b.extract(p.ip);
        transition accept;
    }
}
```

- State machine
- Extracts headers from incoming packets
- Produces parsed representation of packet for use in match-action pipeline

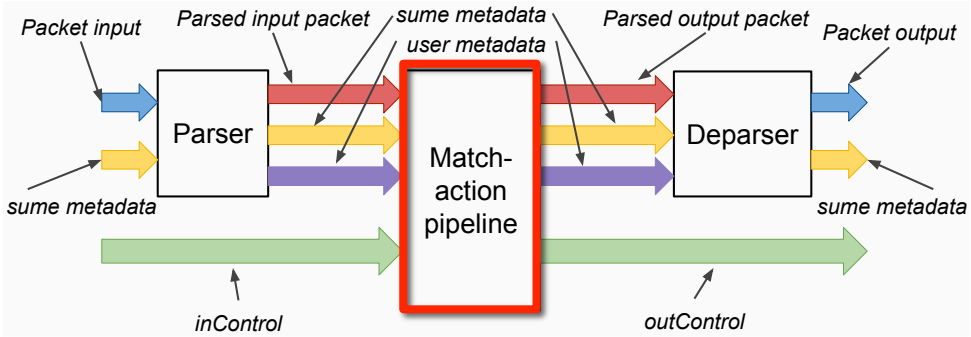
Learning Switch – Control Flow




```
apply {
    // try to forward based on
    // destination Ethernet address
    if (!forward.apply().hit) {
        // miss in forwarding table
        broadcast.apply();
    }

    // check if src Ethernet address
    // is in the forwarding database
    if (!smac.apply().hit) {
        // unknown source MAC address
        send_to_control();
    }
}
```

- Apply match-action tables
- Invoke actions directly
- Control flow may depend on:
  - Hit/miss in table
  - Which action the table invoked

Learning Switch – Control Flow

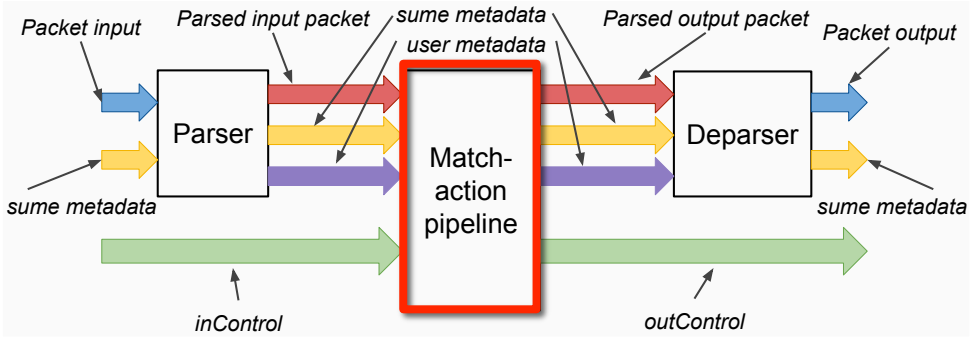



```
apply {
    // try to forward based on
    // destination Ethernet address
    if (!forward.apply().hit) {
        // miss in forwarding table
        broadcast.apply();
    }

    // check if src Ethernet address
    // is in the forwarding database
    if (!smac.apply().hit) {
        // unknown source MAC address
        send_to_control();
    }
}
```

```
action set_output_port(port_t port) {
    sume_metadata.dst_port = port;
}

table forward() {
    key = {
      headers.ethernet.dstAddr: exact;
    }

    actions = {
      set_output_port;
    }
    size = 64;
    default_action = nop;
}
```

- **Tables**:
  - Which fields (header and/or metadata) to match on
  - List of valid actions that can be applied
  - Resources to allocate to table
  - Match types: exact, ternary, LPM

Learning Switch – Control Flow

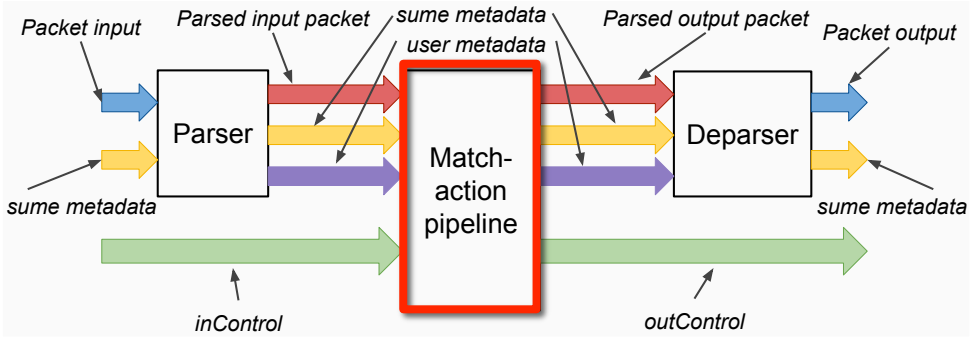



```
apply {
    // try to forward based on
    // destination Ethernet address
    if (!forward.apply().hit) {
        // miss in forwarding table
        broadcast.apply();
    }

    // check if src Ethernet address
    // is in the forwarding database
    if (!smac.apply().hit) {
        // unknown source MAC address
        send_to_control();
    }
}
```

```
action set_output_port(port_t port) {
    sume_metadata.dst_port = port;
}

table forward() {
    key = {
      headers.ethernet.dstAddr: exact;
    }

    actions = {
      set_output_port;
    }
    size = 64;
    default_action = nop;
}
```

- **Actions**:
  - Modify header/metadata fields
  - Parameters may be provided by data-plane or control-plane

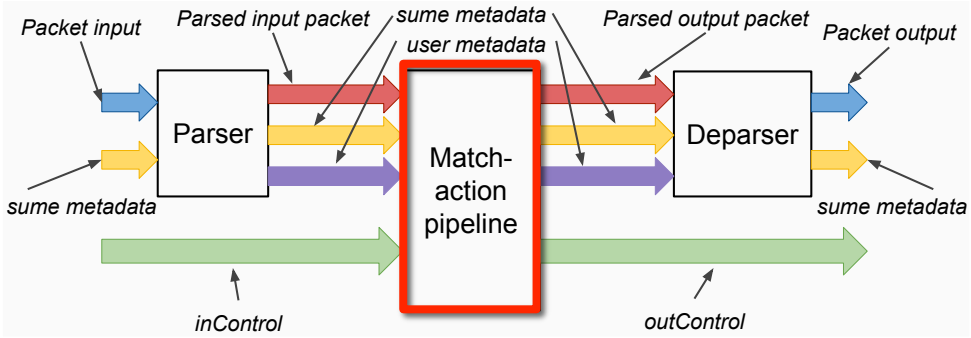
Learning Switch – Control Flow




```
apply {
    // try to forward based on
    // destination Ethernet address
    if (!forward.apply().hit) {
        // miss in forwarding table
        broadcast.apply();
    }

    // check if src Ethernet address
    // is in the forwarding database
    if (!smac.apply().hit) {
        // unknown source MAC address
        send_to_control();
    }
}
```

```
action set_broadcast(port_t port) {
    sume_metadata.dst_port = port;
}

table broadcast() {
    key = {
      sume_metadata.src_port: exact;
    }

    actions = {
        set_broadcast;
        nop;
    }
    size = 64;
    default_action = nop;
}
```

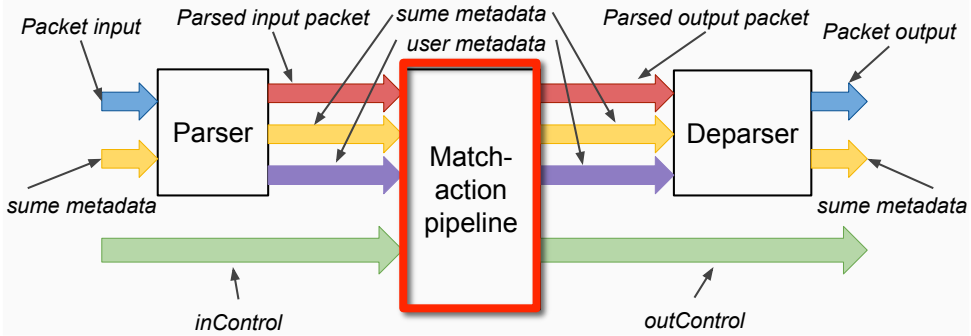
Learning Switch – Control Flow




```
apply {
    // try to forward based on
    // destination Ethernet address
    if (!forward.apply().hit) {
        // miss in forwarding table
        broadcast.apply();
    }

    // check if src Ethernet address
    // is in the forwarding database
    if (!smac.apply().hit) {
        // unknown source MAC address
        send_to_control();
    }
}
```

```
table smac() {
    key = {
      headers.ethernet.srcAddr: exact;
    }

    actions = {
        nop;
    }
    size = 64;
    default_action = nop;
}
```

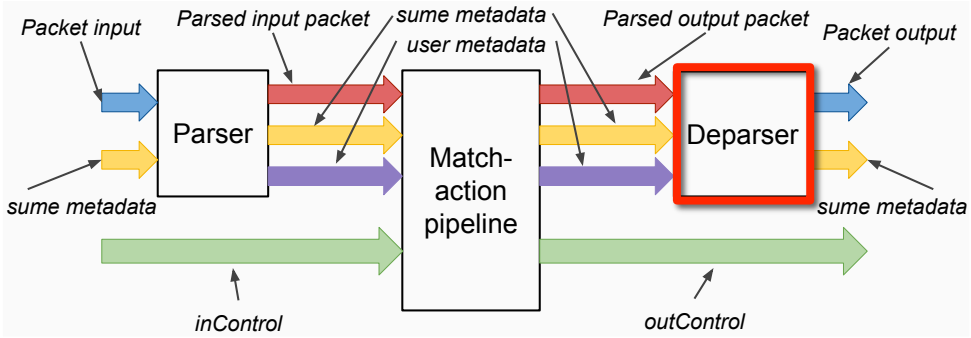
Learning Switch – Control Flow




```
apply {
    // try to forward based on
    // destination Ethernet address
    if (!forward.apply().hit) {
        // miss in forwarding table
        broadcast.apply();
    }

    // check if src Ethernet address
    // is in the forwarding database
    if (!smac.apply().hit) {
        // unknown source MAC address
        send_to_control();
    }
}
```

```
action send_to_control() {
    sume_metadata.digest_data.src_port = sume_metadata.src_port;
    sume_metadata.digest_data.eth_src_addr = headers.ethernet.srcAddr;
    sume_metadata.send_dig_to_cpu = 1;
}
```

Learning Switch – Deparser




```
// Deparser Implementation
control TopDeparser(packet_out b,
                    in Parsed_packet p,
                    in user_metadata_t user_metadata,
                    inout sume_metadata_t sume_metadata) {
    apply {
        b.emit(p.ethernet);
        b.emit(p.ip);
    }
}
```

- Reconstruct the packet
- May append headers or arbitrary data

Learning Switch – Control-Plane

```
def learn_digest(pkt):
    dig_pkt = Digest_data(str(pkt))
    add_to_tables(dig_pkt)

def add_to_tables(dig_pkt):
    src_port = dig_pkt.src_port
    src_addr = dig_pkt.eth_src_addr
    (found, val) = table_cam_read_entry('forward', [src_addr])
    if (found == 'False'):
        table_cam_add_entry('forward', [src_addr], 'set_output_port', [src_port])
        table_cam_add_entry('smac', [src_addr], 'nop', [])

def main():
    sniff(iface=DMA_IFACE, prn=learn_digest, count=0)
```

- Auto generated Python API
- Some other API functions:
  - table_cam_delete_entry()
  - table_cam_get_size()
  - reg_read()
  - reg_write()
- C API also available

Sources

- Special thanks to Nick McKeown for various P4 Intro slides!
- http://p4.org/wp-content/uploads/2015/03/p4-tutorial-12201423.pdf
- https://www.slideshare.net/opennetsummit/p4-webinarintrodemooct2015chang
- https://schd.ws/hosted_files/p4workshop2015/c9/NickM-P4-Workshop-June-04-2015.pdf
- https://schd.ws/hosted_files/p4workshop2015/4b/ChangK-P4-Workshop-June-04-2015.pdf
- http://p4.org/wp-content/uploads/2016/12/P4_16-prerelease-Dec_16.html
- https://www.youtube.com/watch?v=zR88Nlg3n3g

P4-NetFPGA Workflow Details

P4-NetFPGA Workflow

1. Write P4 program
2. Write python gen_testdata.py script
3. Compile to verilog / generate API & CLI tools

   $ make
4. Run initial SDNet simulation

   $ ./vivado_sim.bash
5. Install SDNet output as SUME library core

   $ make install_sdnet
6. Run NetFPGA simulation

   $ ./nf_test sim –major switch –minor default
7. Build bitstream

   $ make
8. Test the hardware

All of your effort will go here

API & Interactive CLI Tool Generation

- Both Python API and C API
  - Manipulate tables and stateful elements in P4 switch
  - Used by control-plane program
- CLI tool
  - Useful debugging feature
  - Query various compile-time information
  - Interact directly with tables and stateful elements in real time
  - Come to demo!

P4-NetFPGA Extern Function Library

- Verilog modules invoked from within P4 programs

Examples:

- Atoms for writing stateful P4 programs based on packet transactions (SIGCOMM 2016)

Atom	Description
R/W	Read or write state
RAW	Read, add to, or overwrite state
PRAW	Predicated version of RAW

- CRC16 checksum function
- More to come …

Using Atom Externs in P4 – Resetting Counter

Packet processing pseudo code:

```
count[NUM_ENTRIES];

if (pkt.hdr.reset == 1):
    count[pkt.hdr.index] = 0
else:
    count[pkt.hdr.index]++
```

Using Atom Externs in P4 – Resetting Counter

```
#define REG_READ    0
#define REG_WRITE   1
#define REG_ADD     2
// count register
@CYCLES(1)
@CONTROLBITS(16)
extern void count_reg_raw(in bit<16> index,
                          in bit<32> newVal,
                          in bit<32> incVal,
                          in bit<8> opCode,
                          in bit<32> result);

bit<16> index = pkt.hdr.index;
bit<32> newVal;
bit<32> incVal;
bit<8> opCode;

if(pkt.hdr.reset == 1) {
    newVal = 0;
    incVal = 0; // not used
    opCode = REG_WRITE;
} else {
    newVal = 0; // not used
    incVal = 1;
    opCode = REG_ADD;
}

bit<32> result; // the new value stored in count
count_reg_raw(index, newVal, incVal, opCode, result);
```

- State can be accessed exactly ***1 time***
- Using RAW atom here

- Instantiate atom

- Set metadata for state access

- State access!

P4-NetFPGA Extern Function Library

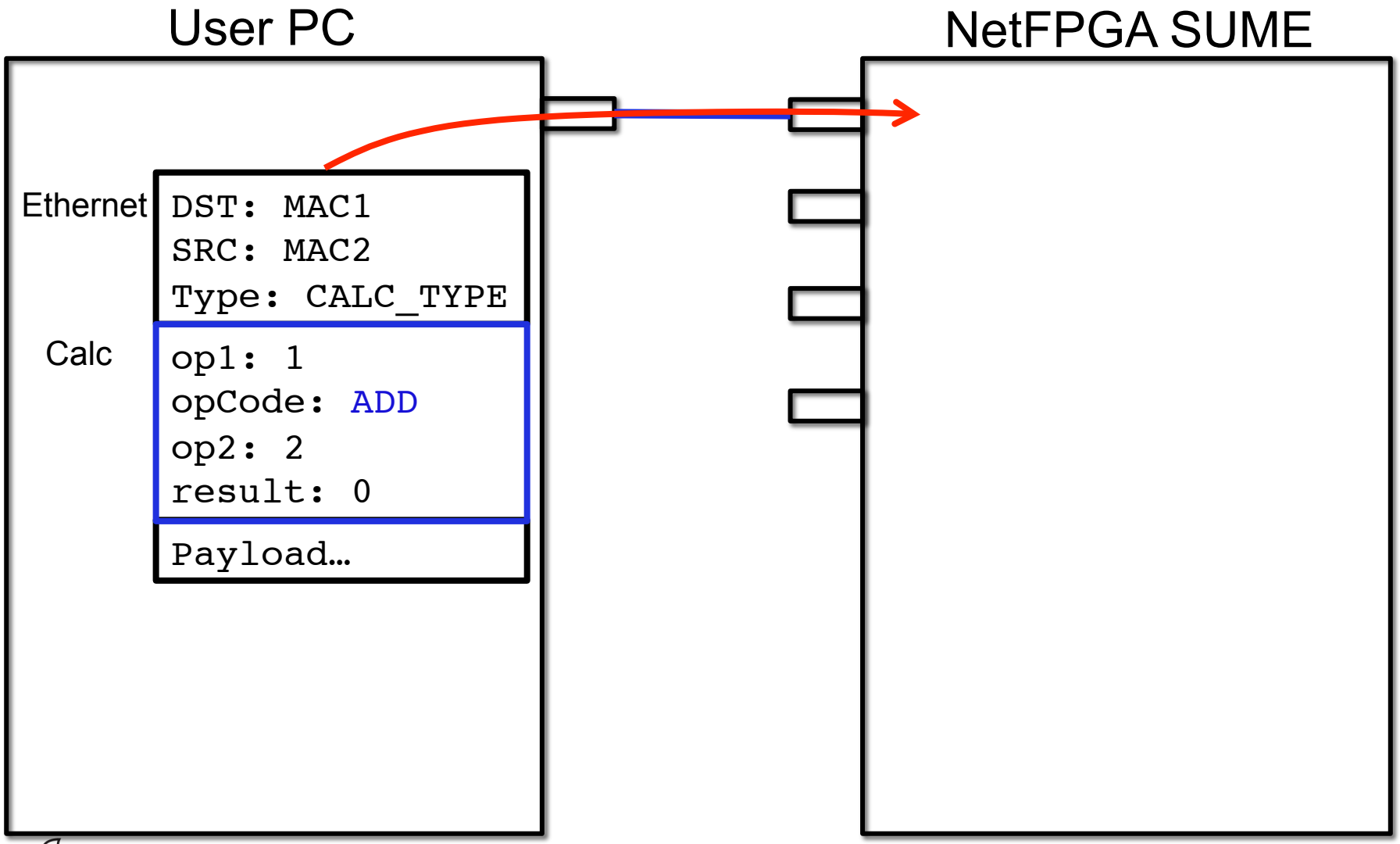
- What will you contribute? ☺
  - Functions to write to external memory
  - Hash functions
  - Time-based functions
  - Meters
  - Etc…

P4-NetFPGA Sample Projects

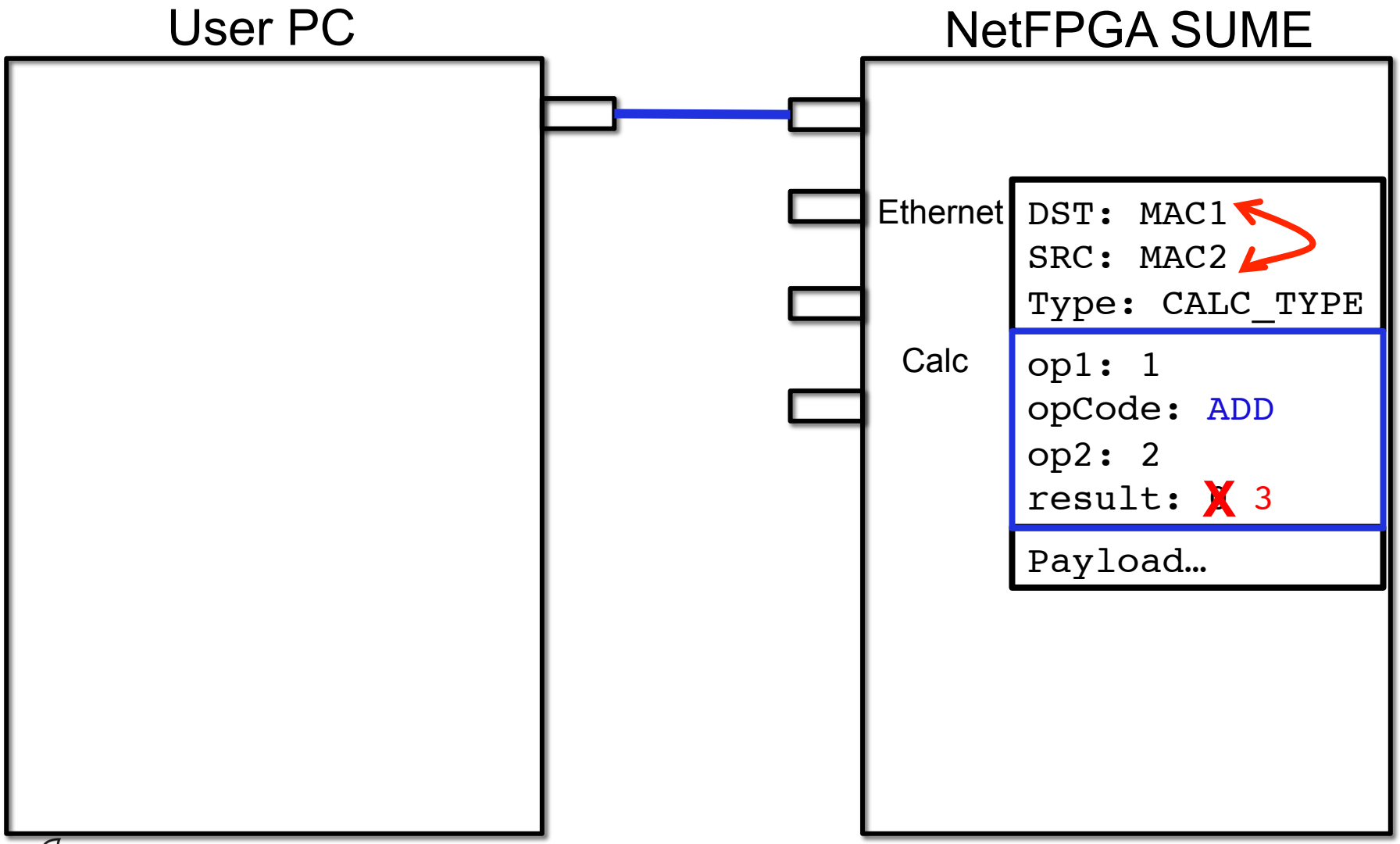
- L2 Learning Switch
- Switch as calculator (demo)
- In-band network telemetry
- SYN flood detection (work in progress)
- … what will you contribute? ☺

P4-NetFPGA Demo

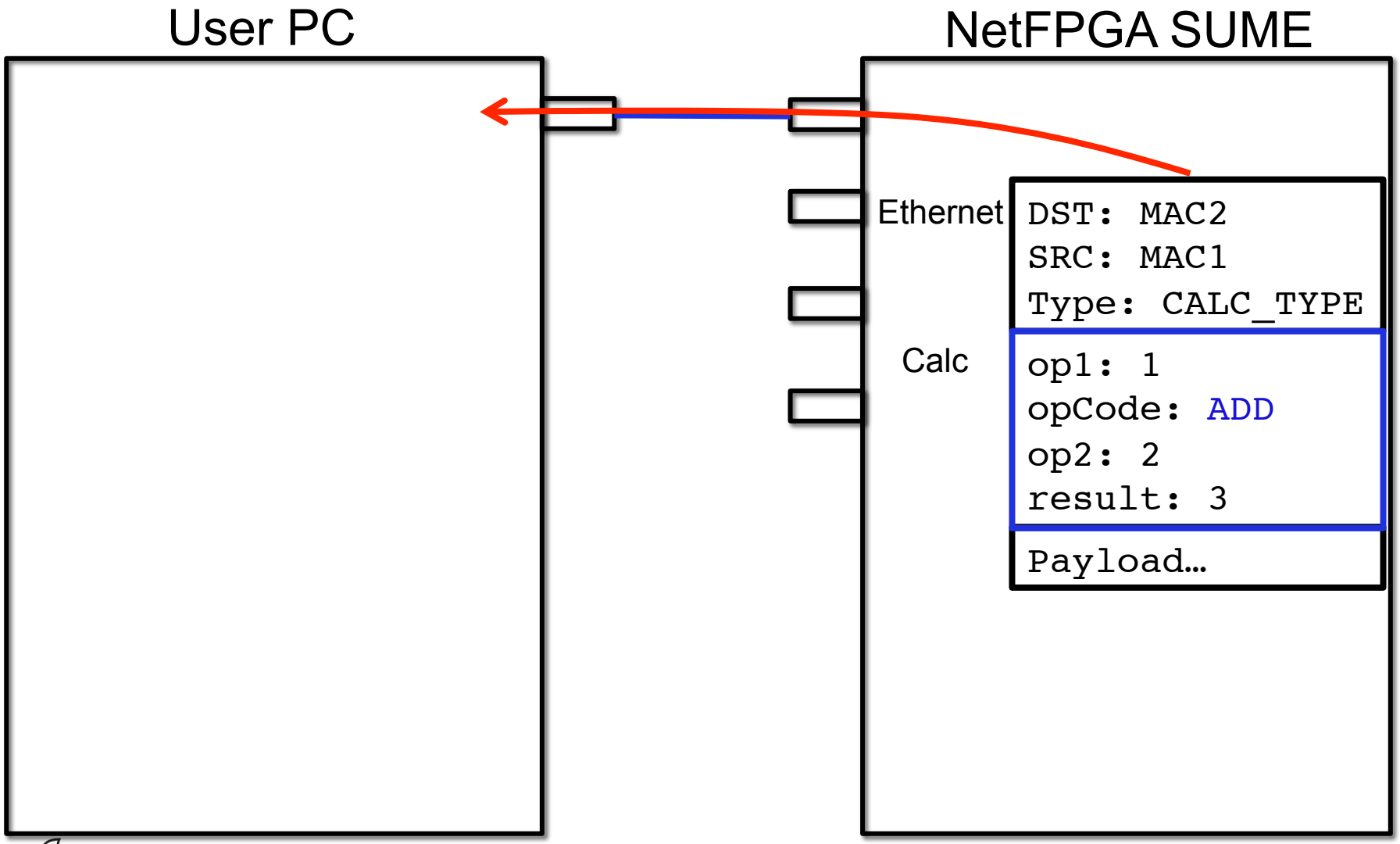
Switch as Calculator



Switch as Calculator



Switch as Calculator



Switch as Calculator

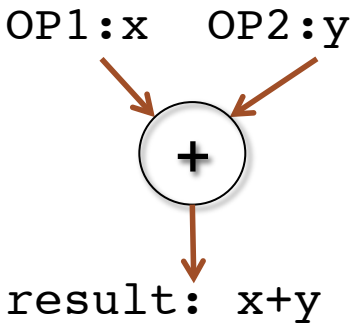
Supported Operations:

- ADD – add two operands
- SUBTRACT – subtract two operands
- ADD_REG – add constant to current value in register
- SET_REG – overwrite the current value of the register
- LOOKUP – Lookup the given key in the table

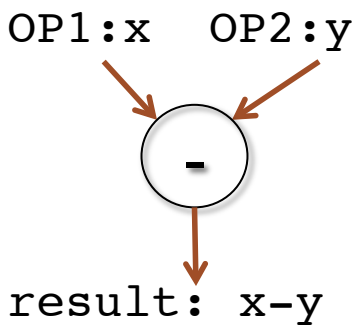
```
header Calc_h {
    calcField_t op1;
    bit<8> opCode;
    calcField_t op2;
    calcField_t result;
}
```

Switch Calc Operations

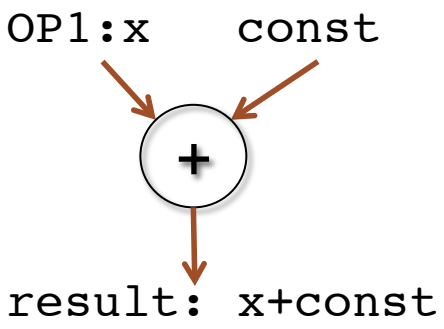
ADD



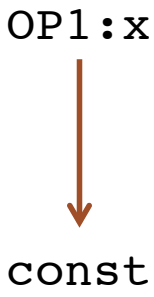
SUB



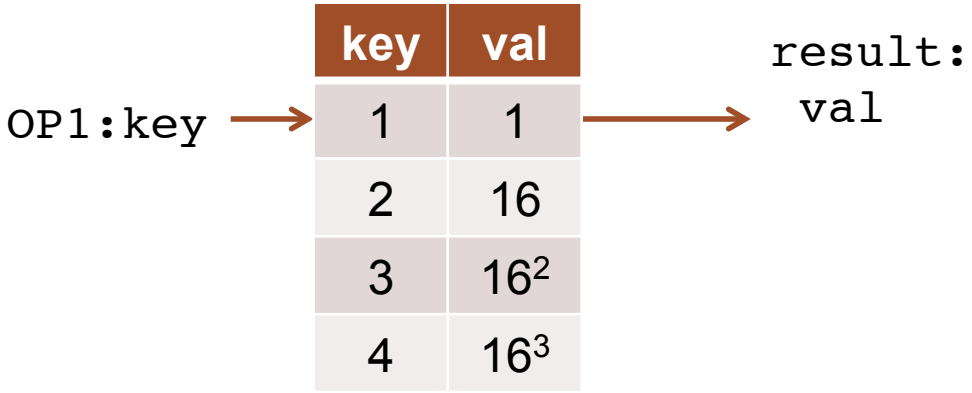
ADD_REG



SET_REG



LOOKUP



key	val
1	1
2	16
3	16^2
4	16^3