
# Hands-on Labs

INSTRUCTORS:

STEPHEN IBANEZ, GORDON BREBNER, ROBERT HALSTEAD,
CHRIS NEELY, TUSHAR SWAMY, SEAN CHOI

# Outline

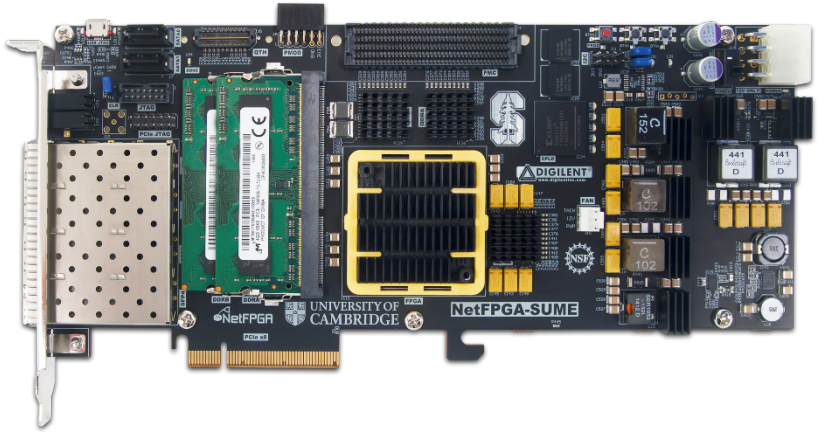
- P4-NetFPGA Workflow Overview
- P4 Compilation Using Xilinx P4-SDNet
- P4-NetFPGA Workflow Details
- Tutorial Assignments

# P4-NetFPGA Workflow Overview

# So what is NetFPGA?
# NetFPGA = Networked FPGA

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

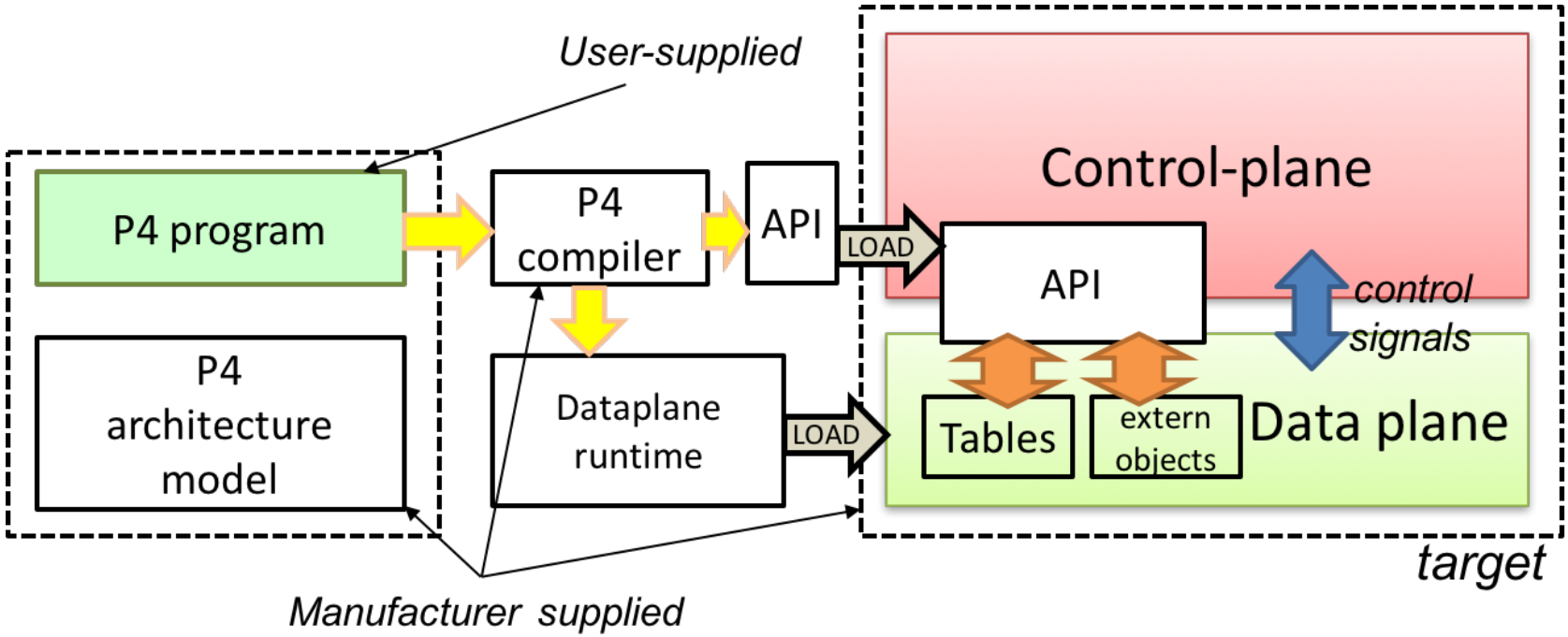
SUME



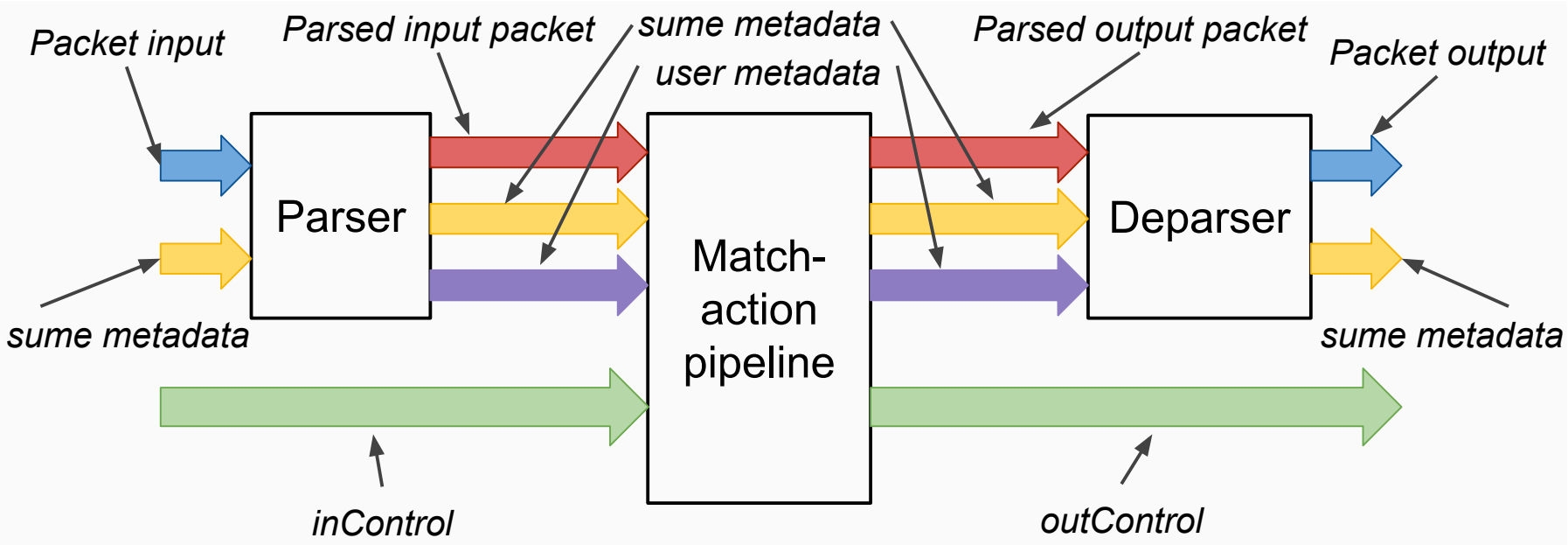

=

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

# General Process for Programming a P4 Target



# SimpleSumeSwitch Architecture Model for SUME Target

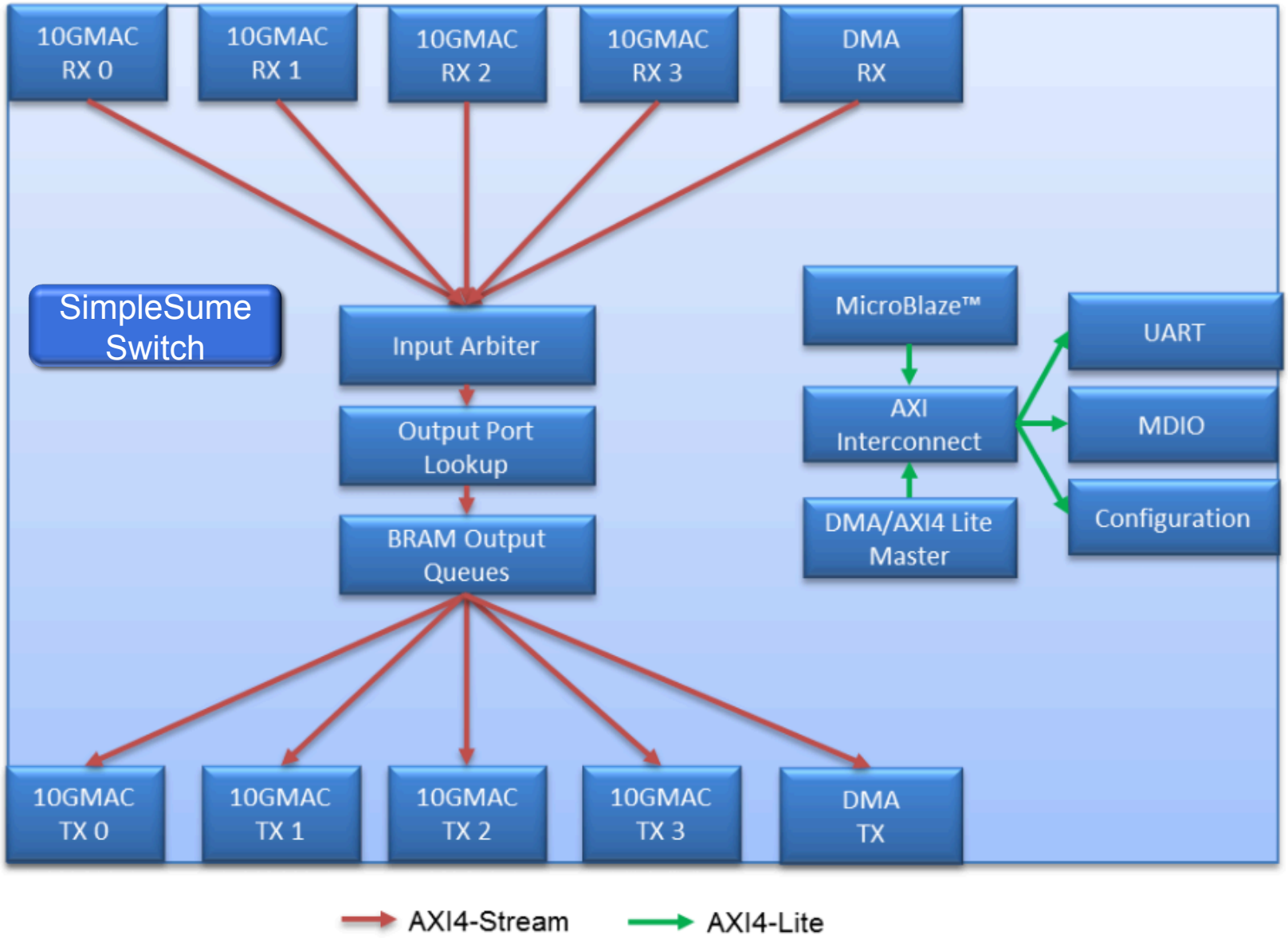



P4 used to describe parser, match-action pipeline, and deparser

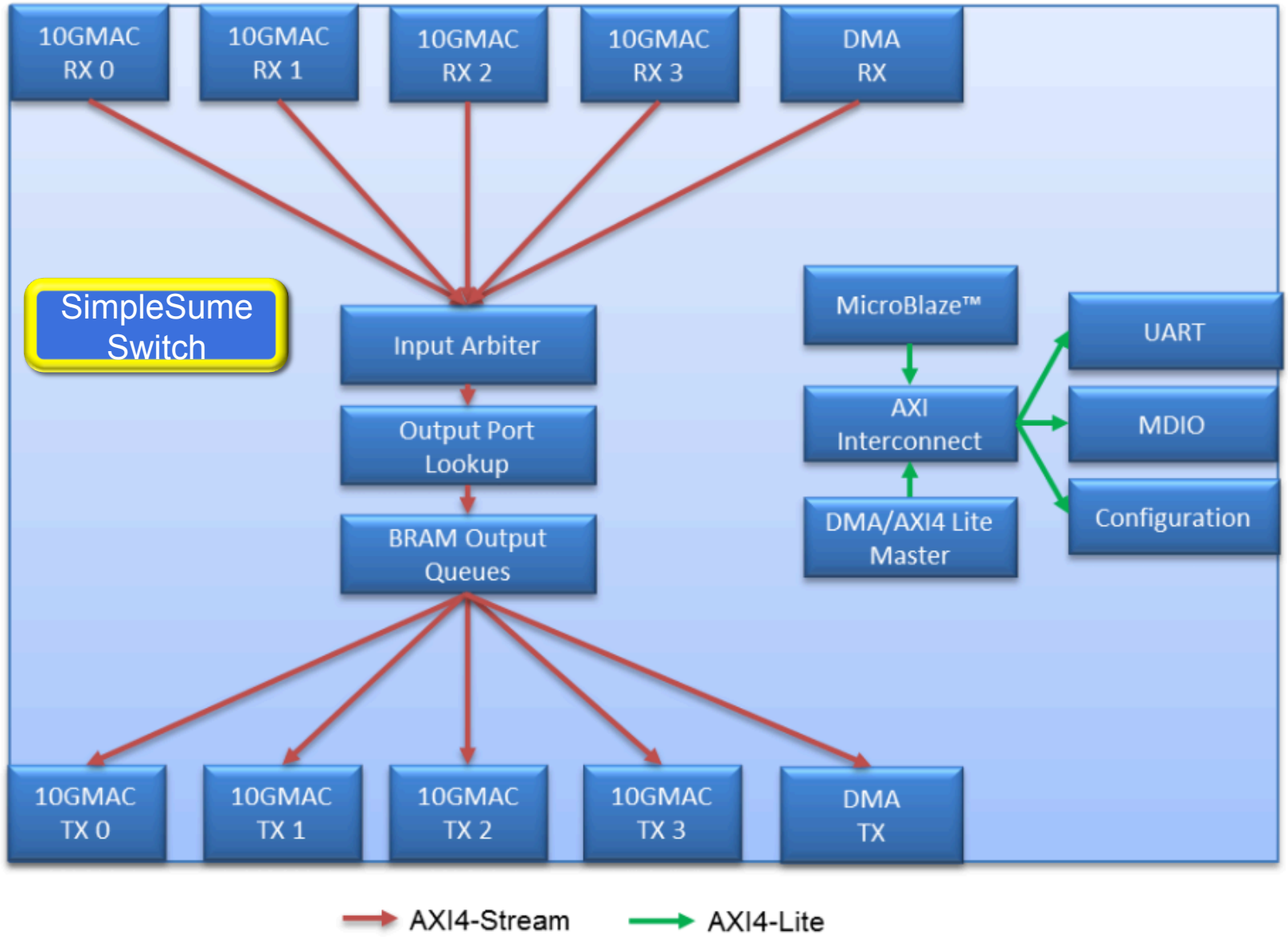
# Standard SUME Metadata in Architecture Model

```
/* 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

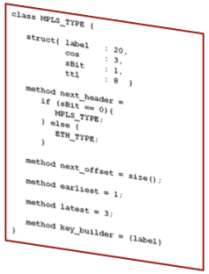


# P4 Compilation Using Xilinx P4-SDNet

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

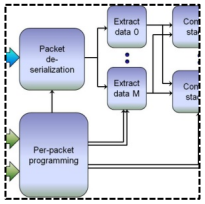
Headline feature set, uniquely enabled by FPGA ‘white box hardware’ target:

Scalable 1G to 100G line rate performance

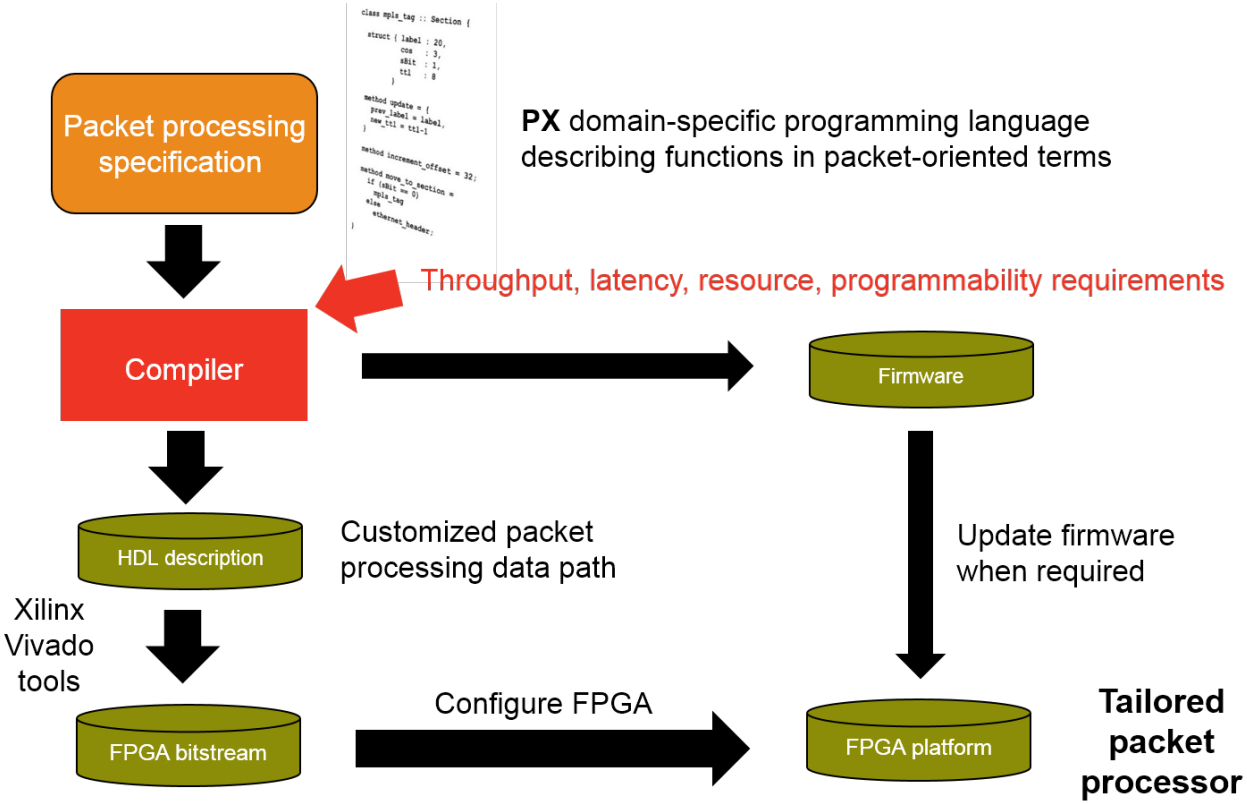

Domain-specific programming abstraction

Exact-fit hardware for reduced cost and power



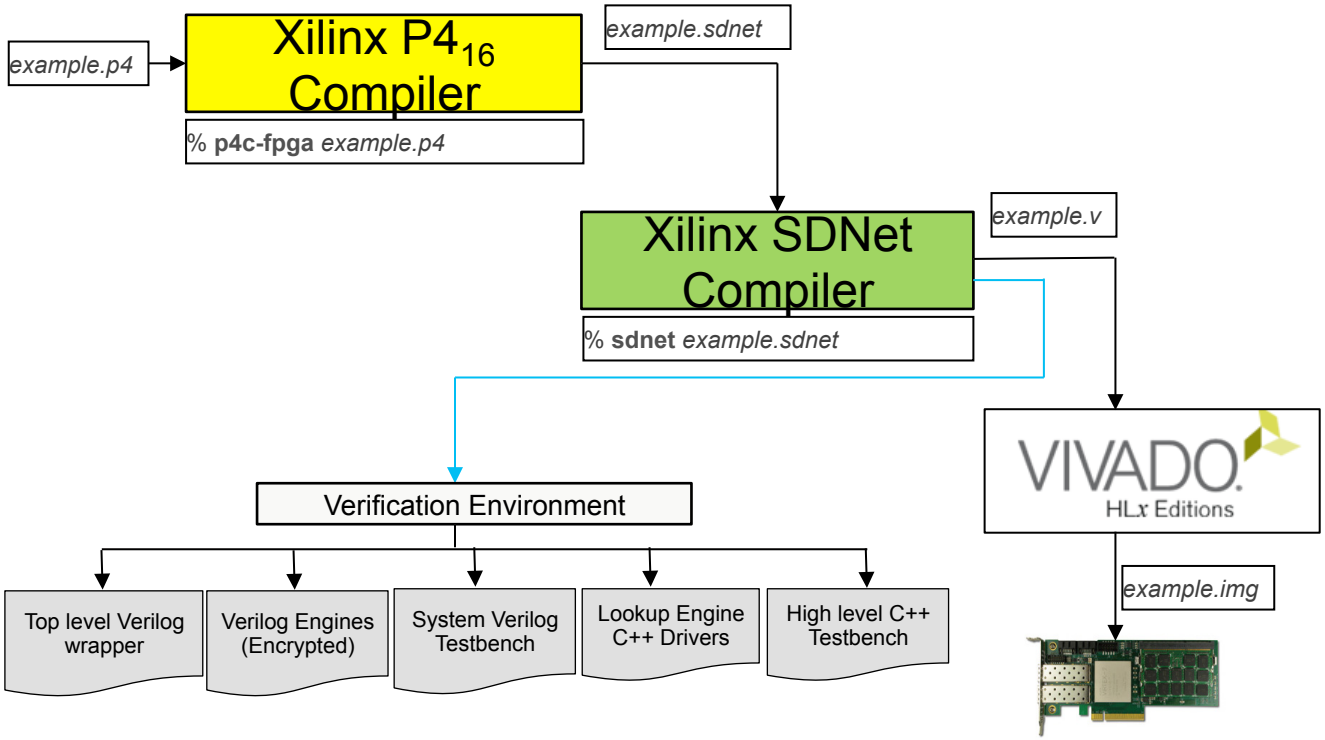

Firmware for run time programmability

# 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

# 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

# Directory Structure of $SUME_FOLDER

```
NetFPGA-SUME-SDNet/
|
|- contrib-projects/
|   |- sume-sdnet-switch/ → the main directory for P4 dev
|
|- lib/ → contains all of the SUME IP cores
|
|- tools/ → various NetFPGA scripts for test infra.
|
|- Makefile → builds all of the SUME IP cores
```

# Directory Structure of $SUME_SDNET

```
sume-sdnet-switch/
|
|- bin/ → scripts used to automate workflow
|
|- templates/ → templates for externs, wrapper module,
|                CLI tools, new projects
|
|- projects/ → all of the P4 project directories
|   |- switch_calc/
```

# Directory Structure of $P4_PROJECT_DIR

```
switch_calc/
|
|- src/ → P4 source files
|
|- testdata/ → scripts to generate testdata used for
|              verifying functionality of P4 program
|
|- simple_sume_switch/ → main SUME project directory,
|              top level HDL files and SUME sim scripts
|
|- sw/ → populated with API files and CLI tools
|
|- nf_sume_sdnet_ip/ → SDNet output directory
```

# 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

# 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 |

- LRC16 checksum function
- Timestamp generation
- 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!

# Tutorial Assignments

https://bitbucket.org/sibanez/netfpga-sume-sdnet/wiki/Tutorial%20Assignments

http://tinyurl.com/p4-netfpga-dev-day

Username: p4user
Password: p4Rocks!

# Assignment 1: Switch as a Calculator

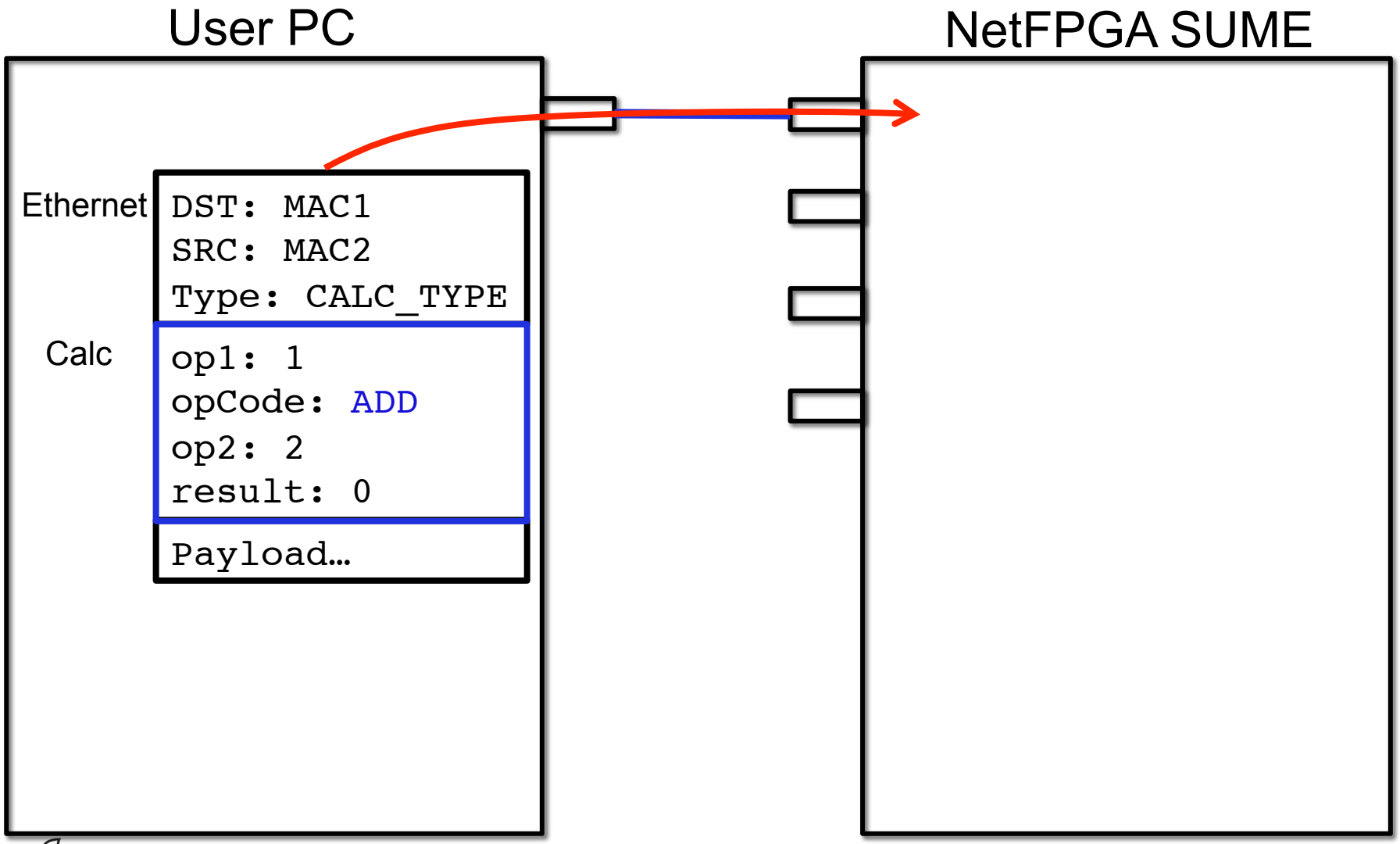
# Switch as Calculator

Supported Operations:

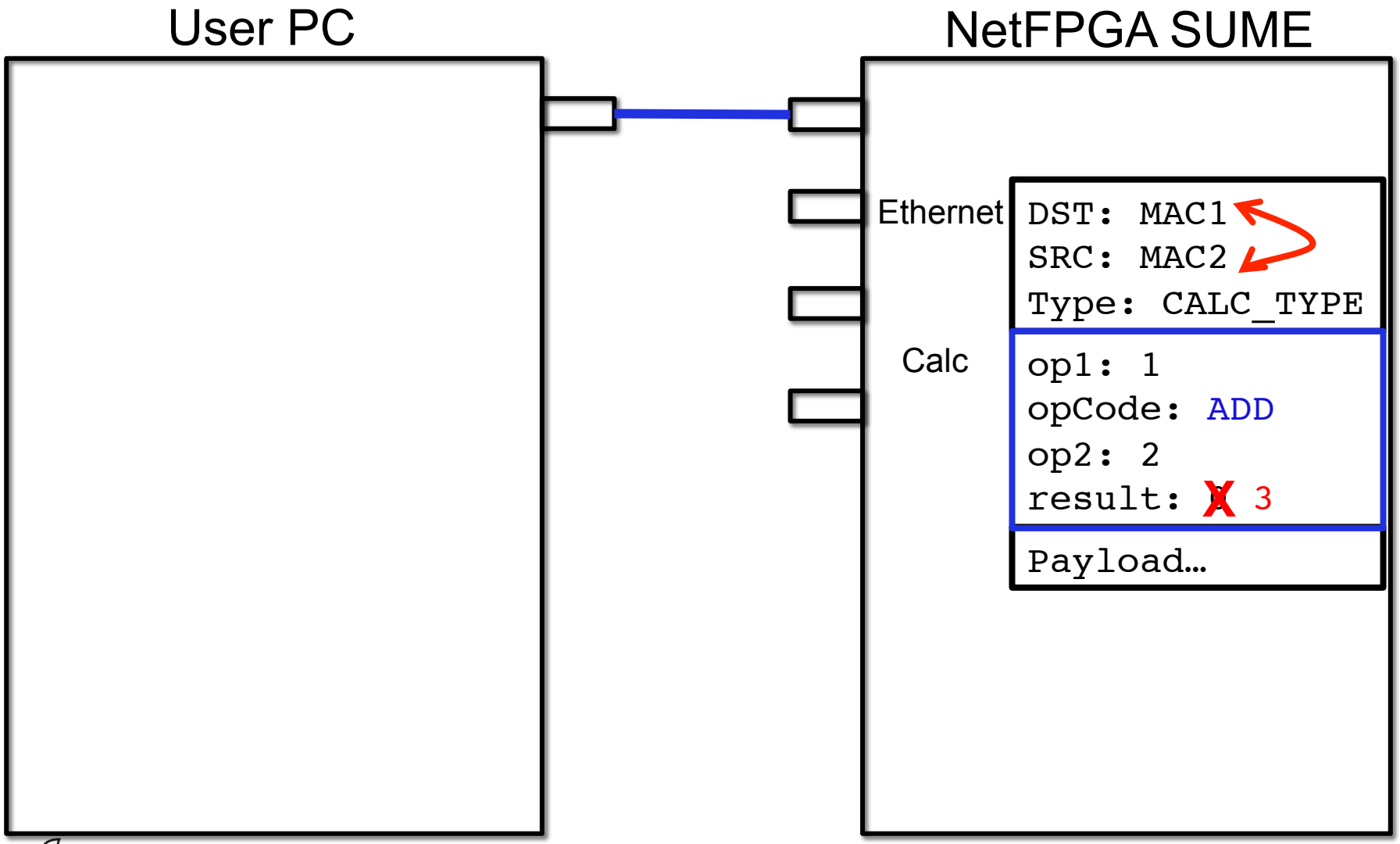
- ADD – add two operands
- SUBTRACT – subtract two operands
- ADD_REG – add constant to current value in register
- SET_REG – overwrite the current value of the register
- LOOKUP – Lookup the given key in the table

```
header Calc_h {
    bit<32> op1;
    bit<8> opCode;
    bit<32> op2;
    bit<32> result;
}
```

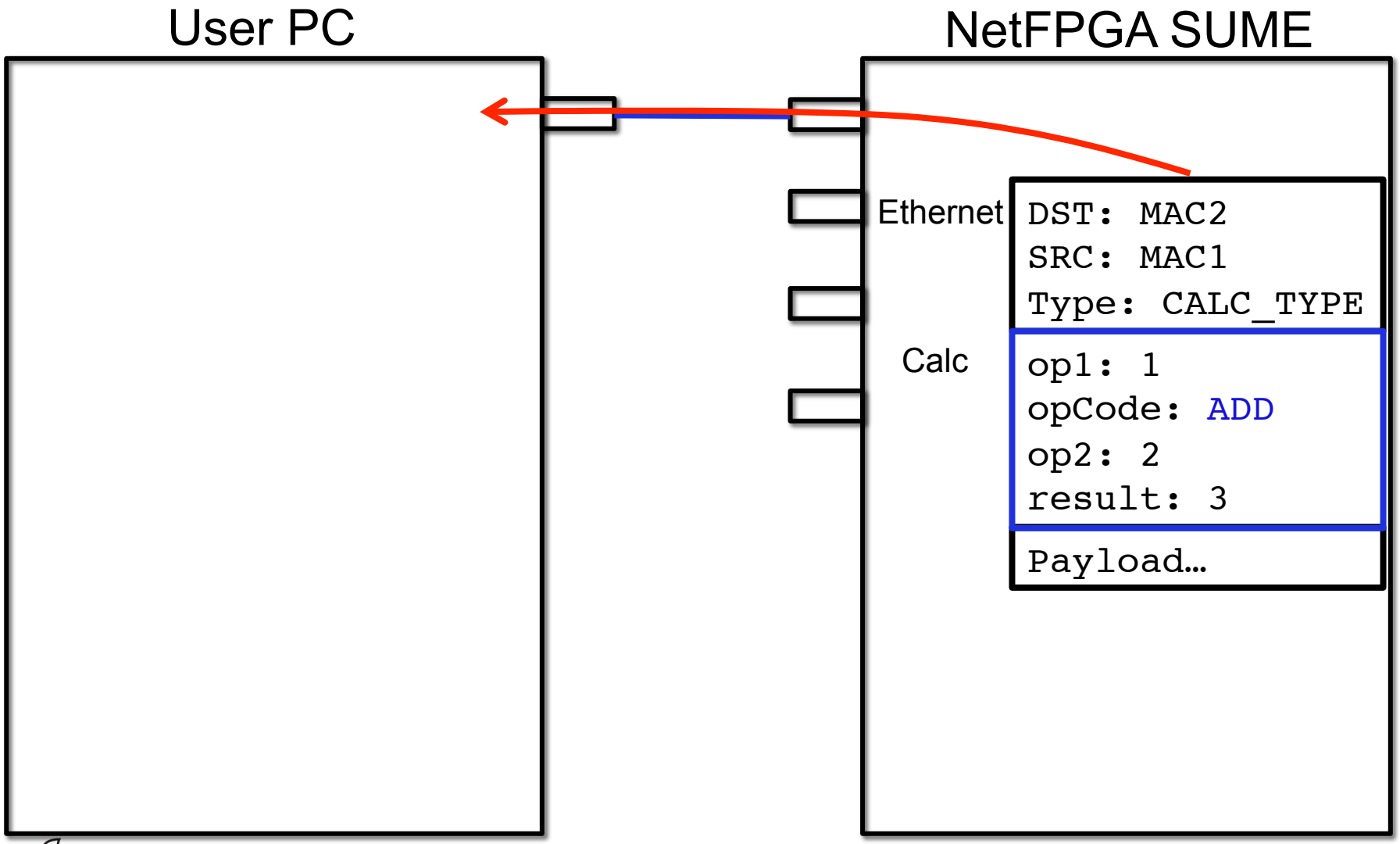
# Switch as Calculator



# Switch as Calculator

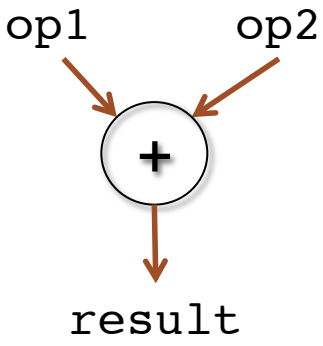


# Switch as Calculator

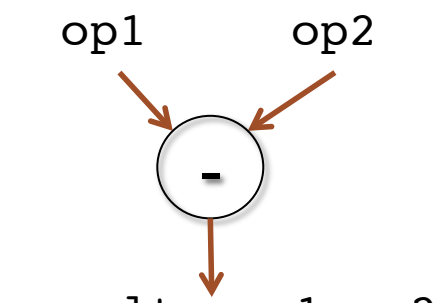


# Switch Calc Operations

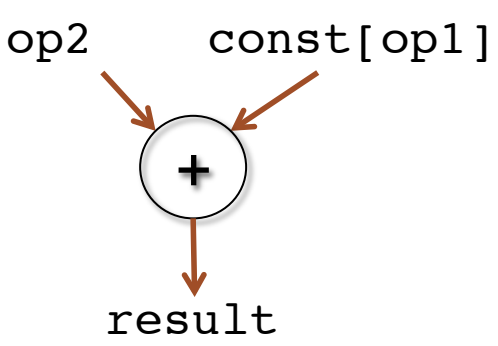
**ADD**



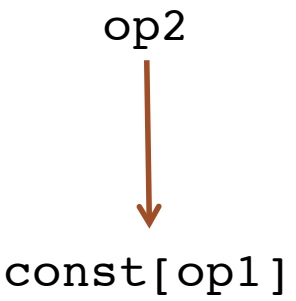

**SUB**




**ADD_REG**




**SET_REG**




**LOOKUP**

key: op1 →

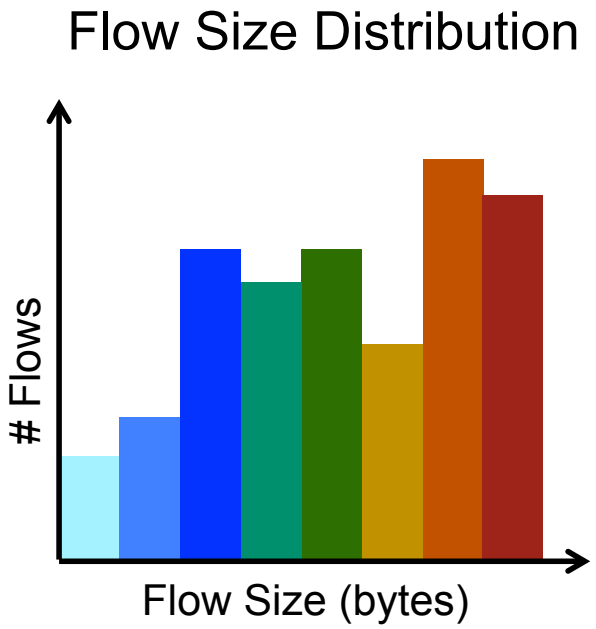
| key | val |
|---|---|
| 0 | 1 |
| 1 | 16 |
| 2 | $16^2$ |
| 3 | $16^3$ |

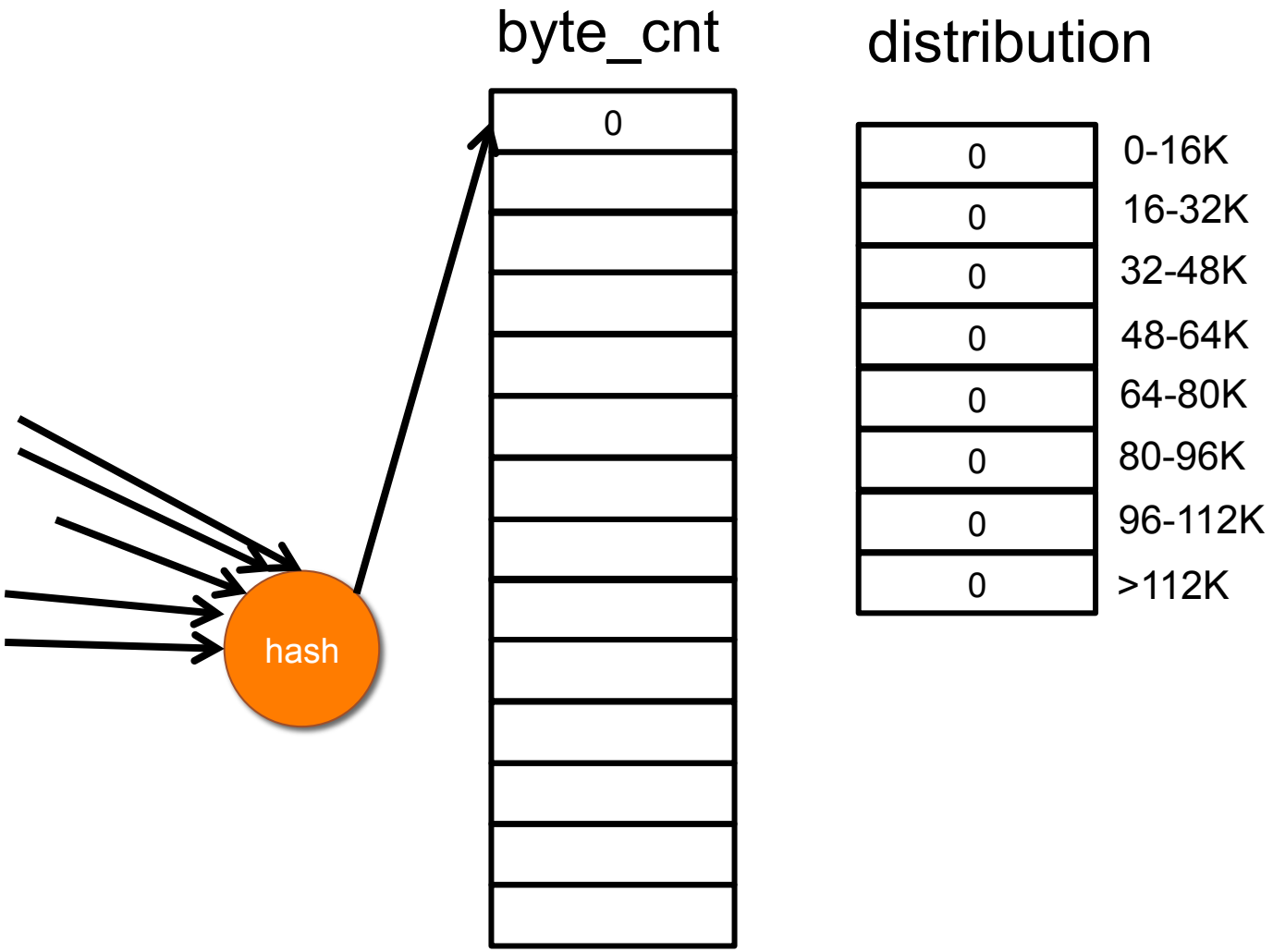
→ result: val

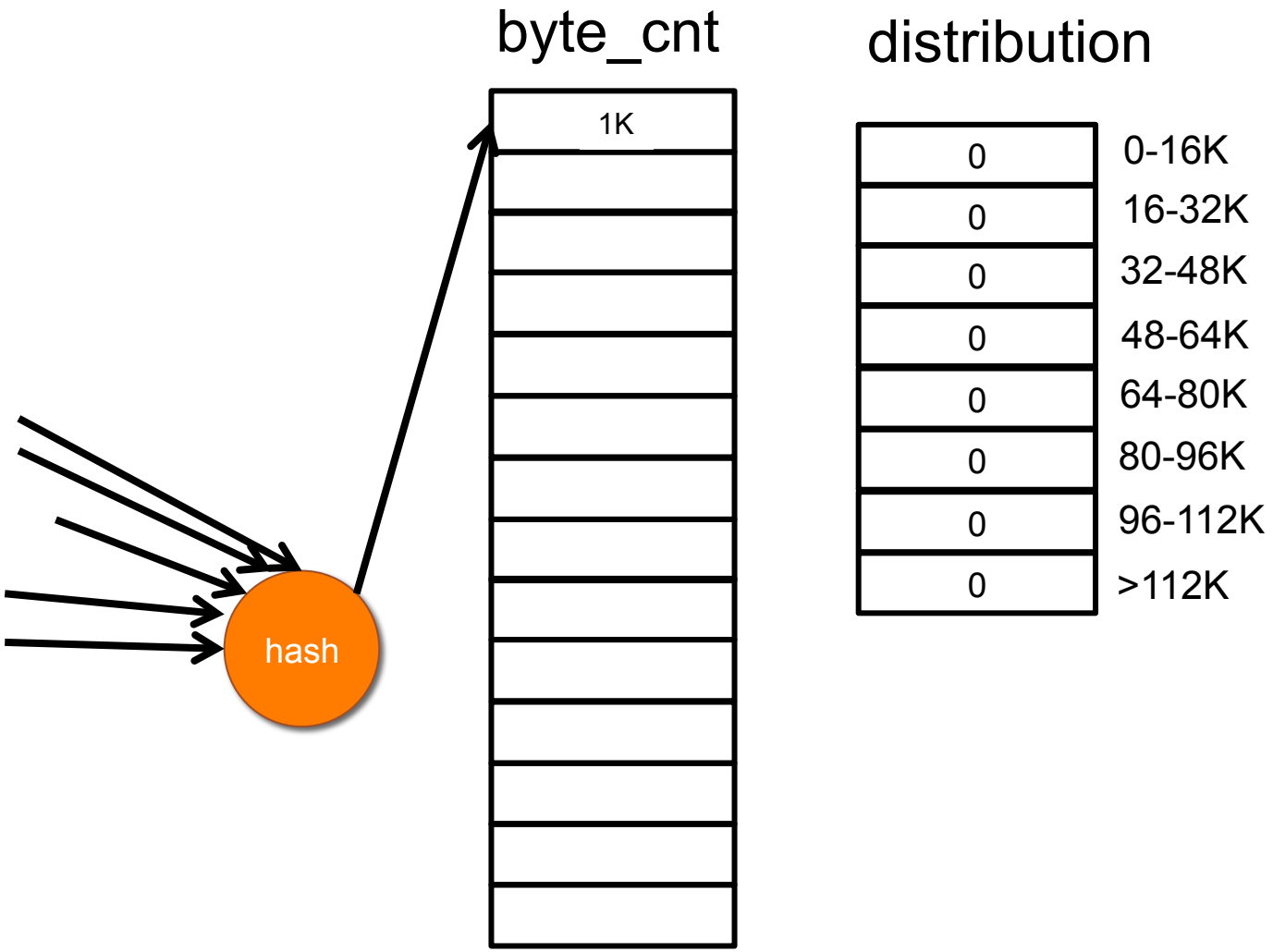
# Assignment 2: TCP Monitor

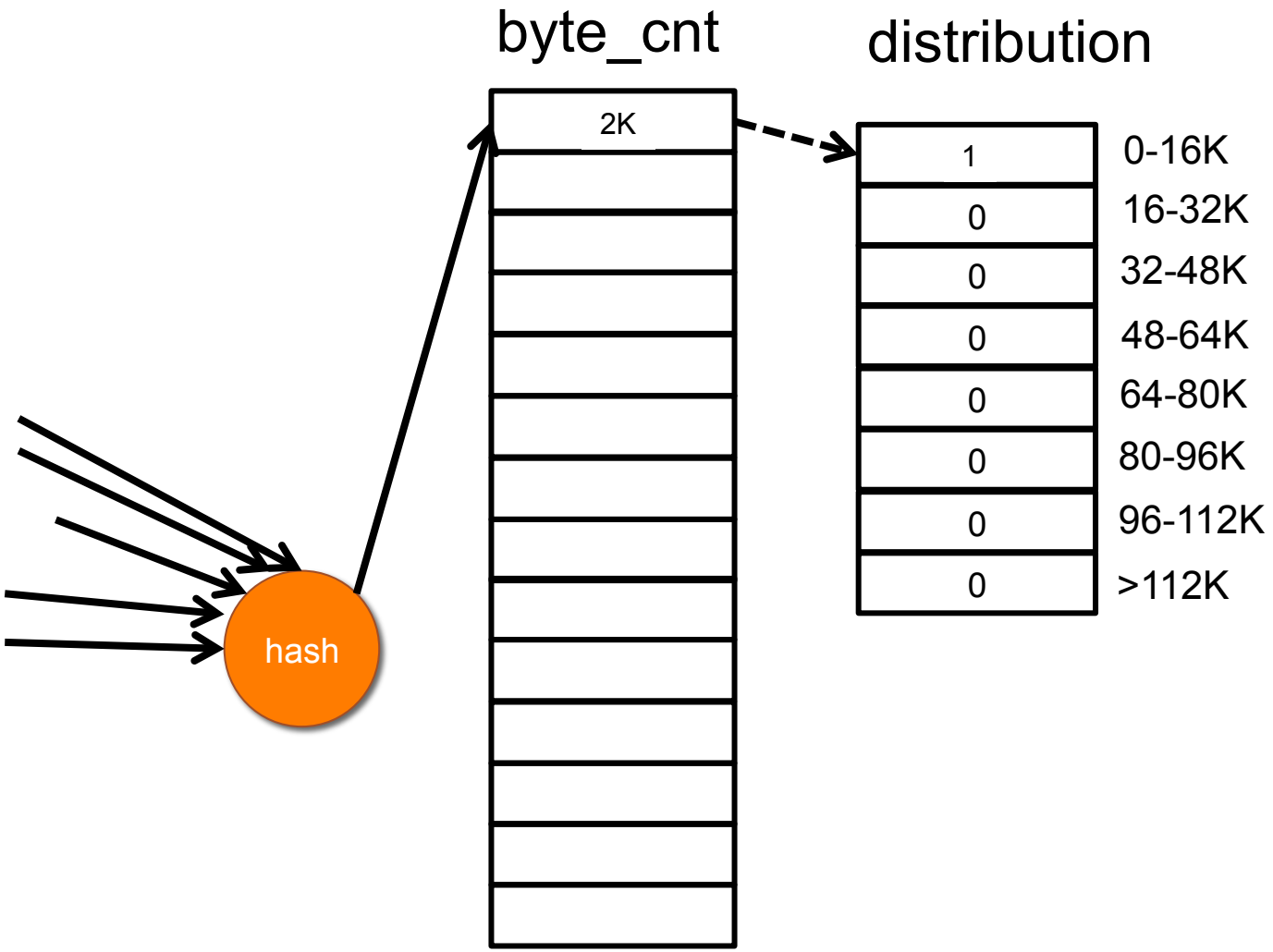
# TCP Monitor

- Practice writing stateful P4 programs for NetFPGA SUME
- Compute TCP flow size distribution in the data-plane
- Flow is determined by 5-tuple and delimited by SYN/FIN
- Fine grained flow monitoring capabilities with P4




byte_cnt
distribution
0
hash
0
0
0
0
0
0
0
0-16K
16-32K
32-48K
48-64K
64-80K
80-96K
96-112K
>112K


byte_cnt
distribution
1K
hash
0
0
0
0
0
0
0
0
0-16K
16-32K
32-48K
48-64K
64-80K
80-96K
96-112K
>112K


byte_cnt
distribution
2K
1
0
0
0
0
0
0
0
0-16K
16-32K
32-48K
48-64K
64-80K
80-96K
96-112K
>112K
hash

# Assignment 3: In-band Network Telemetry (INT)

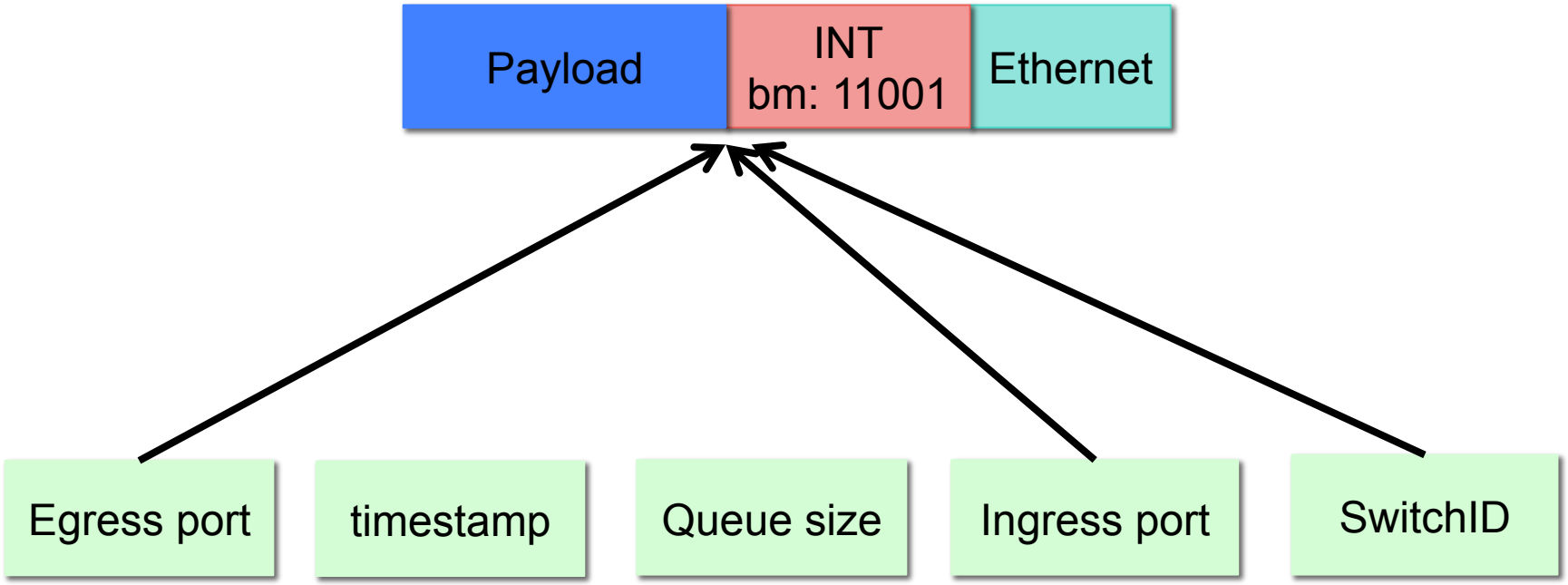
# In-band Network Telemetry (INT)

- One of the most popular applications for programmable data-planes
- All about gaining more visibility into network
- Basic idea:
  - Source requests each switch along path to insert some desired metadata into packet (using a bitmask)
- Example metadata:
  - Switch ID
  - Ingress Port
  - Egress Port
  - Timestamp
  - Queue Occupancy

# In-band Network Telemetry (INT)

- Bitmask format (5 bits):

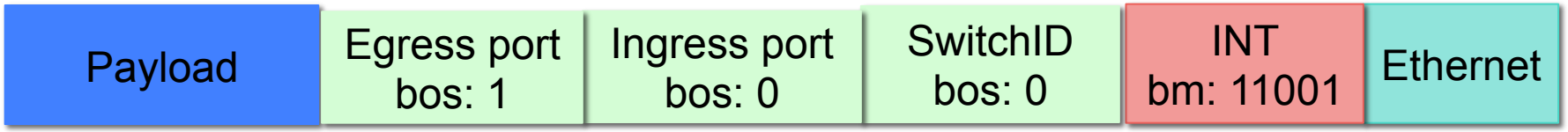
<SWITCH_ID><INGRESS_PORT><Q_SIZE><TSTAMP><EGRESS_PORT>

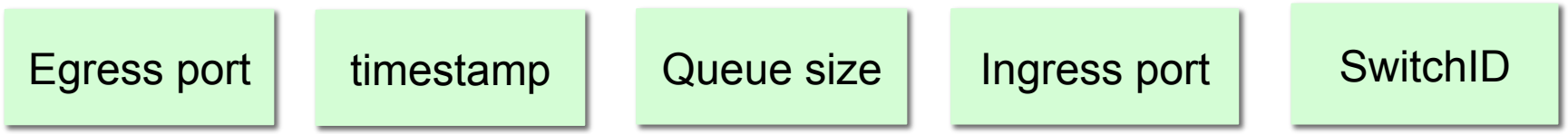


# In-band Network Telemetry (INT)

- Bitmask format (5 bits):

<SWITCH_ID><INGRESS_PORT><Q_SIZE><TSTAMP><EGRESS_PORT>



# 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)

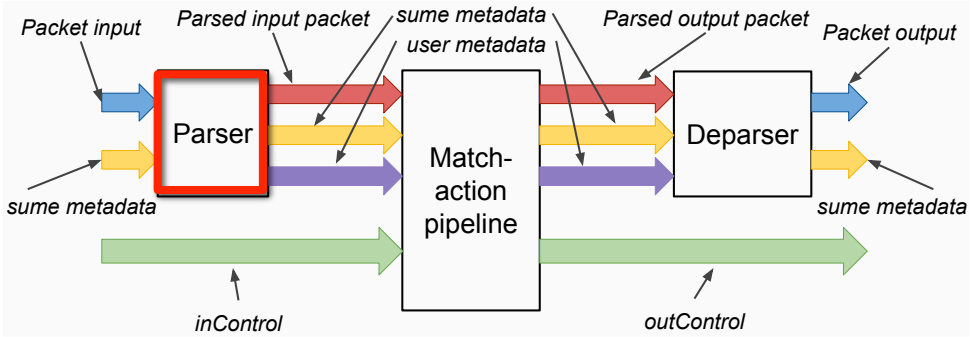
# Learning Switch – Header definitions

```
// 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

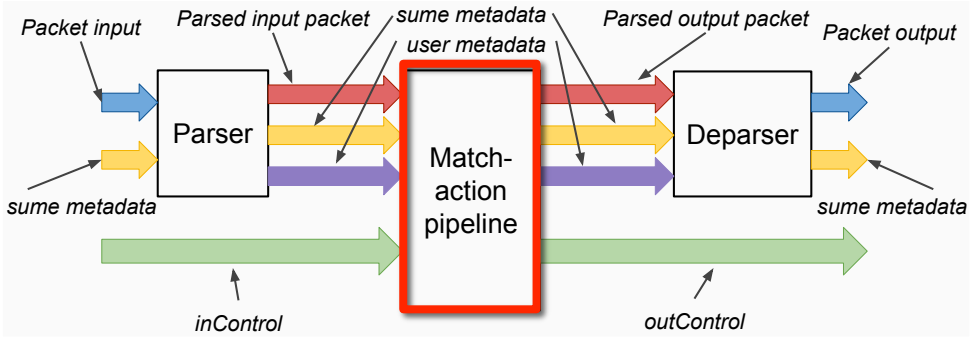
# Learning Switch – Parser




```
// 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

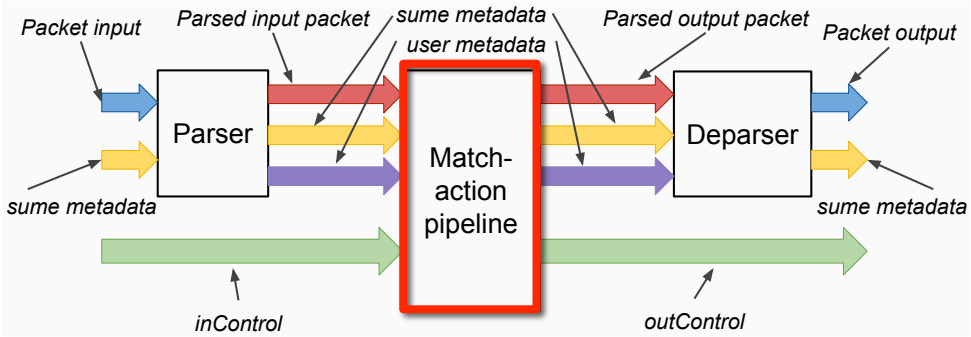
# Learning Switch – Control Flow




```
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

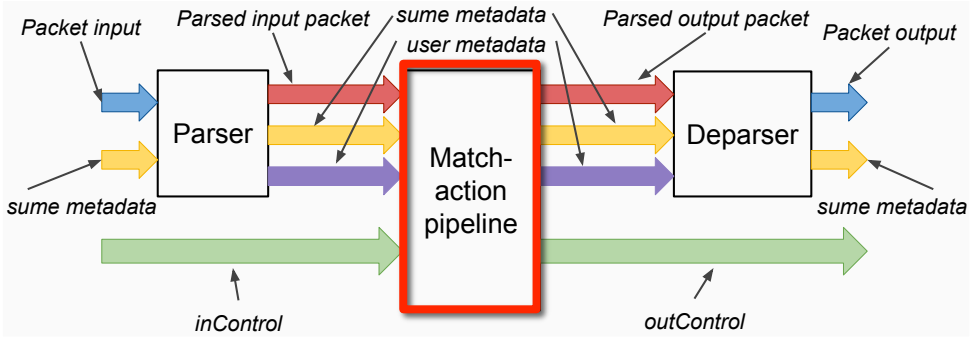



```
apply {
    // try to forward based on
    // destination Ethernet address
    if (!forward.apply().hit) {
        // miss in forwarding table
        broadcast.apply();
    }

    // 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

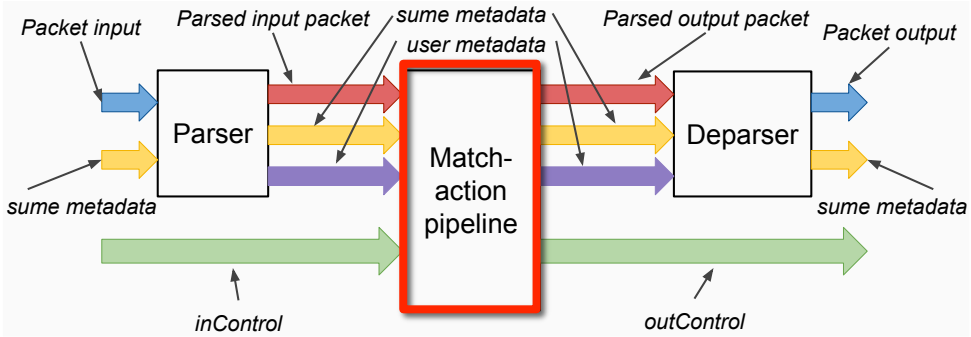



```
apply {
    // try to forward based on
    // destination Ethernet address
    if (!forward.apply().hit) {
        // miss in forwarding table
        broadcast.apply();
    }

    // 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

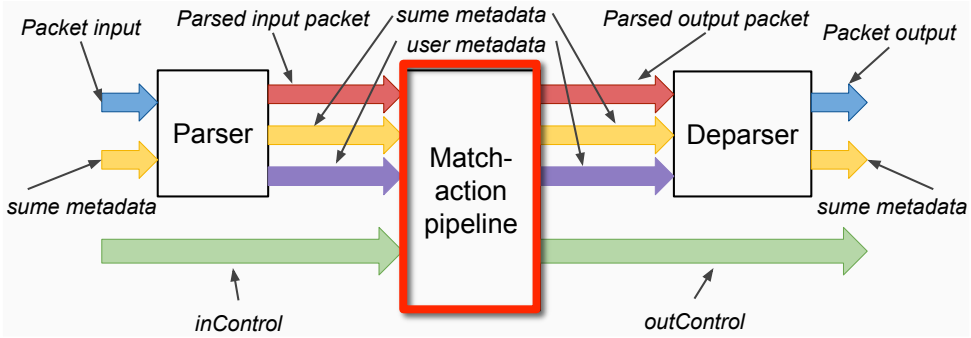
# Learning Switch – Control Flow




```
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;
}
```

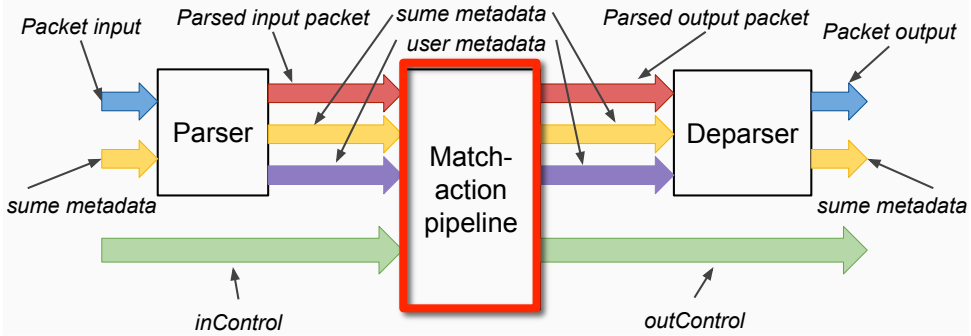
# Learning Switch – Control Flow




```
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;
}
```

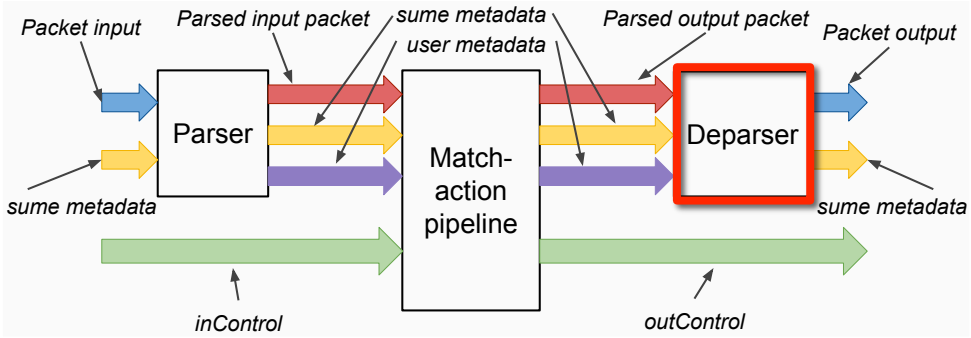
# Learning Switch – Control Flow




```
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