Tutorial

STEPHEN IBANEZ

Outline

- P4 Motivation
- P4 for NetFPGA Overview
- P4->NetFPGA Workflow Overview
- Tutorial Assignments

What is P4?

- Programming language to describe packet processing logic
- Used to implement forwarding-plane of network elements (i.e. switches, routers, NICs, etc.)

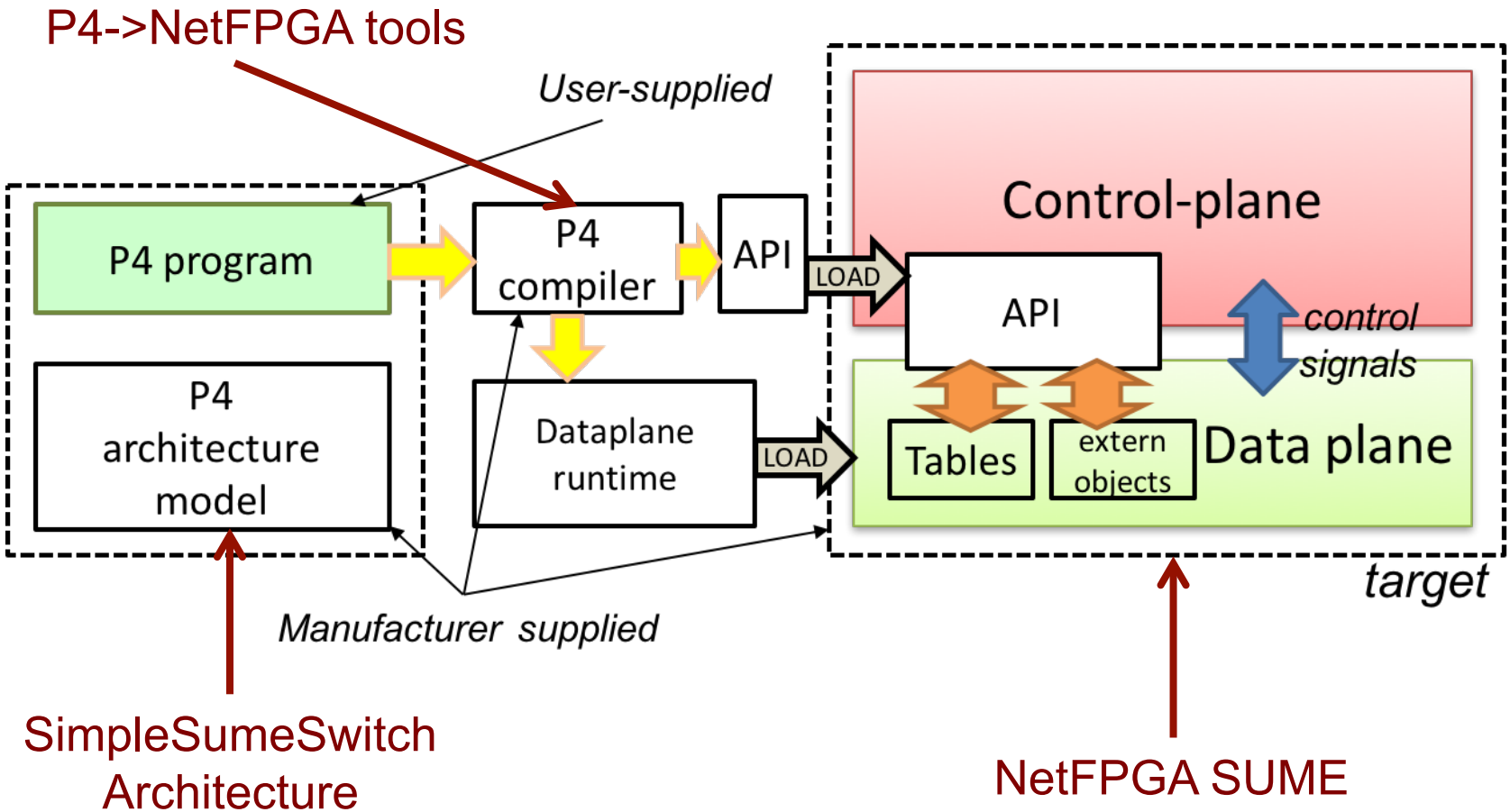
Benefits of Programmable Forwarding

- 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

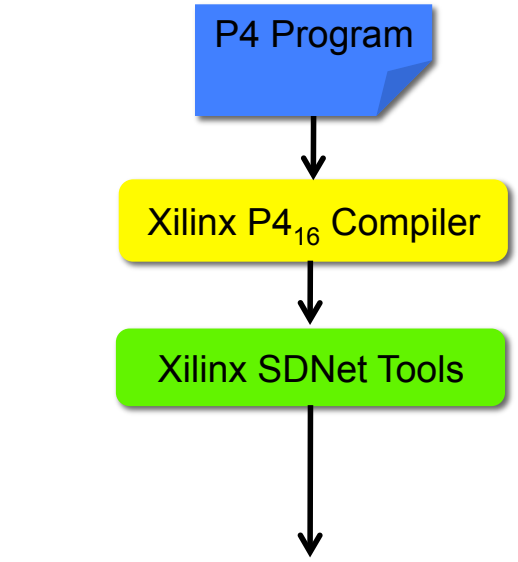
Think programming rather than protocols…

P4 for NetFPGA Overview

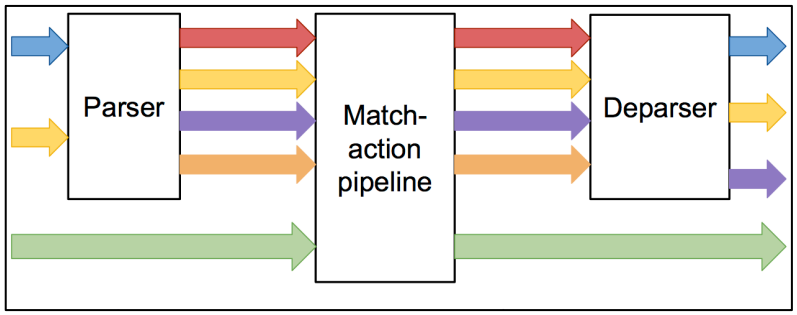
General Process for Programming a P4 Target



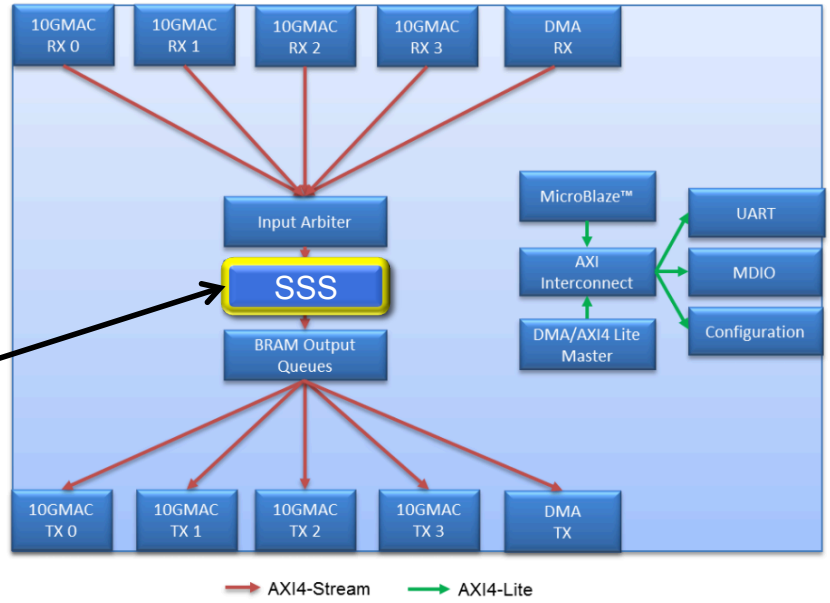
P4->NetFPGA Compilation Overview



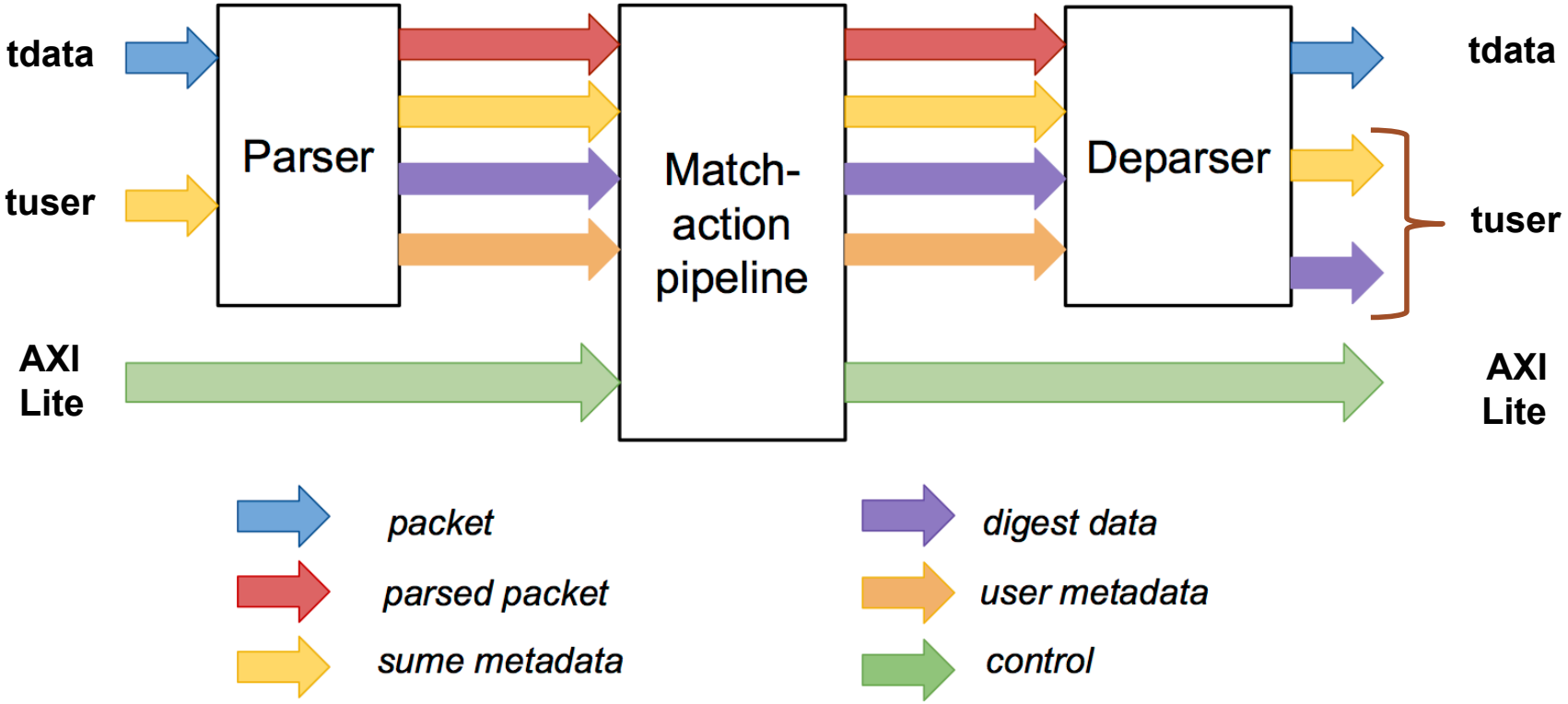
SimpleSumeSwitch Architecture



NetFPGA Reference Switch



SimpleSumeSwitch Architecture Model for SUME Target



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

Metadata in SimpleSumeSwitch Architecture

```
/* standard sume switch metadata */
struct sume_metadata_t {
    bit<16> dma_q_size;
    bit<16> nf3_q_size;
    bit<16> nf2_q_size;
    bit<16> nf1_q_size;
    bit<16> nf0_q_size;
    bit<8> send_dig_to_cpu; // send digest_data to CPU
    bit<8> drop;
    bit<8> dst_port; // one-hot encoded
    bit<8> src_port; // one-hot encoded
    bit<16> pkt_len; // unsigned int
}
```

- `*_q_size` – size of each output queue, measured in terms of 32-byte words, when packet starts being processed by the P4 program
- `src_port`/`dst_port` – one-hot encoded, easy to do multicast
- `user_metadata`/`digest_data` – structs defined by the user

P4->NetFPGA Compilation Overview

- 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
  - Standard AXI-S packet input/output interfaces
  - Standard AXI Lite control interface
- Supports P4 extern feature for user defined logic

P4 Language Components

State-machine;
Field extraction

Table lookup and update;
Field manipulation;
Control Flow

Field assembly

Overall P4 Program Structure

```
#include <core.p4>
#include <sume_switch.p4>

/******** CONSTANTS ********/
#define IPV4_TYPE   0x0800

/******** TYPES ********/
typedef bit<48> EthAddr_t;
header Ethernet_h {...}
struct Parsed_packet {...}
struct user_metadata_t {...}
struct digest_data_t {...}

/******** EXTERN FUNCTIONS ********/
extern void const_reg_rw(...);

/******** PARSERS and CONTROLS ********/
parser TopParser(...) {...}
control TopPipe(...) {...}
control TopDeparser(...) {...}

/******** FULL PACKAGE ********/
SimpleSumeSwitch(TopParser(), TopPipe(), TopDeparser()) main;
```

Learning by example – L2 Learning Switch

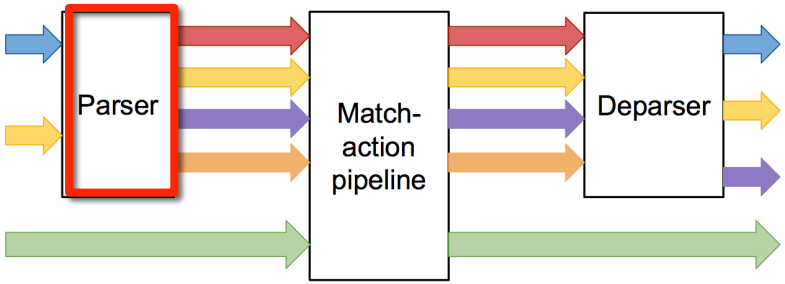
- Parses Ethernet frames
- Forwards based on Ethernet destination address
- Frame broadcasted (with ingress port filtering) if address not in forwarding database
- Learns based on Ethernet source address
- If source address is unknown, the address and the source port are sent to the control-plane (which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

Learning Switch – Header/Metadata definitions

```
typedef bit<48> EthernetAddress;

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

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

// data to send to cpu if desired
// MUST be 80 bits
struct digest_data_t {
    port_t src_port;
    EthernetAddress eth_src_addr;
    bit<24> unused;
}
```

- **typedef** – alternative name for a type
- **header** – ordered collection of members
  - Can be valid or invalid
  - Byte-aligned
- **struct** – collection of members
  - May contain any derived types
- Header stacks - array of headers
- Parsed_packet
  - The headers that can be parsed, manipulated, or created by the switch
- digest_data_t
  - Data that may be sent to the control-plane if desired.

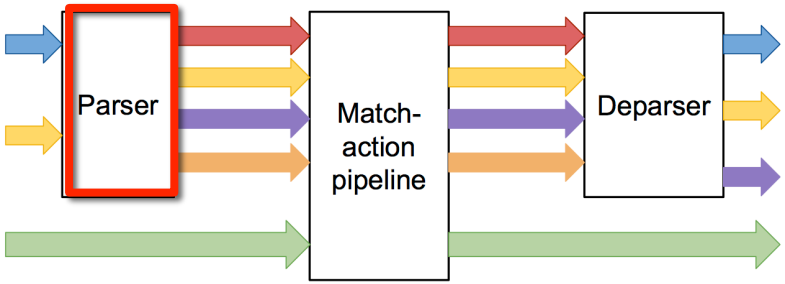
Learning Switch – Parser




```
@Xilinx_MaxPacketRegion(16384)
parser TopParser(packet_in b,
        out Parsed_packet p,
        out user_metadata_t user_metadata,
        out digest_data_t digest_data,
        inout sume_metadata_t sume_metadata) {

  state start {
    b.extract(p.ethernet);
    digest_data.src_port = 0;
    digest_data.eth_src_addr = 0;
    digest_data.unused = 0;
    transition accept;
  }
}
```

- State machine
- 3 predefined states:
  - start
  - accept
  - reject
- User defined states
- Extracts headers from incoming packets
- Produces parsed representation of packet for use in match-action pipeline
- `@Xilinx_MaxPacketRegion` – for parser/deparser; declares the largest packet size (in bits) to support

Ethernet + IPv4 – Parser




```
parser TopParser(packet_in b,
        out Parsed_packet p,
        out user_metadata_t user_metadata,
        out digest_data_t digest_data,
        inout sume_metadata_t sume_metadata) {

  state start {
    b.extract(p.ethernet);
    digest_data.src_port = 0;
    digest_data.eth_src_addr = 0;
    digest_data.unused = 0;
    transition select(p.ethernet.etherType) {
        IPV4_TYPE : parse_ipv4;
        default : accept;
    }
  }

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

- State machine
- 3 predefined states:
  - start
  - accept
  - reject
- User defined states
- Extracts headers from incoming packets
- Produces parsed representation of packet for use in match-action pipeline
- `@Xilinx_MaxPacketRegion` – for parser/deparser; declares the largest packet size (in bits) to support

Control Blocks

```
control TopPipe(inout Parsed_packet p,
    inout user_metadata_t user_metadata,
    inout digest_data_t digest_data,
    inout sume_metadata_t sume_metadata) {

    // Define tables
    // Define actions
    // Define global metadata

    apply {
        // Apply tables
        // Call actions
        // Define local metadata
    }
}
```

- Similar to C functions without loops
  - Algorithms should be representable as Directed Acyclic Graphs (DAG)
- Standard arithmetic and logical operations:
  - +, -, *
  - ~, &, |, ^, >>, <<
  - ==, !=, >, >=, <, <=
  - No division/modulo
- Additional operations:
  - Bit-slicing: [m:l]
  - Bit Concatenation: ++

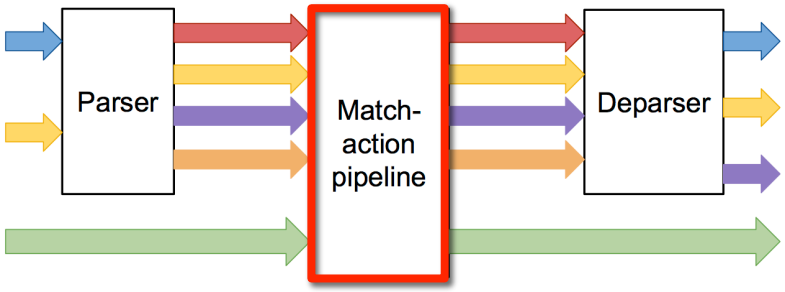
Match-Action Tables

- Which fields (header and/or metadata) to match on
  - Match type: exact, ternary, LPM
- List of valid actions that can be applied
- Resources to allocate to table
- Single entry includes:
  - Specific key to match on
  - A **single** action
    - To be executed when a packet matches the entry
  - (Optional) action data
- apply() method returns:
  - hit (boolean) – hit in table
  - action_run (enum) – the action invoked by the table

Actions

- Modify header/metadata
- Used in tables or invoked directly
- May contain if/else statements
- Action Parameters
  - Directional – from data-plane
  - Directionless – from control-plane
- Header validity bit manipulation
  - header.setValid()
  - header.setInvalid()
  - header.isValid()

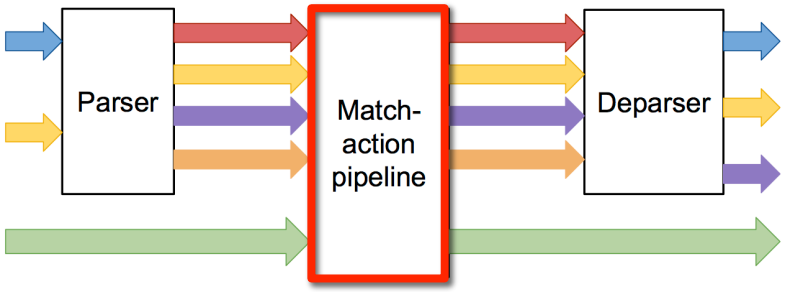
Learning Switch – Control Flow




```
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();
  }
}
```

- Try to forward based on dst MAC
- Broadcast packet if dst MAC is not known
- Check if src MAC is known
- Send src port and src MAC to control-plane if src MAC is unknown

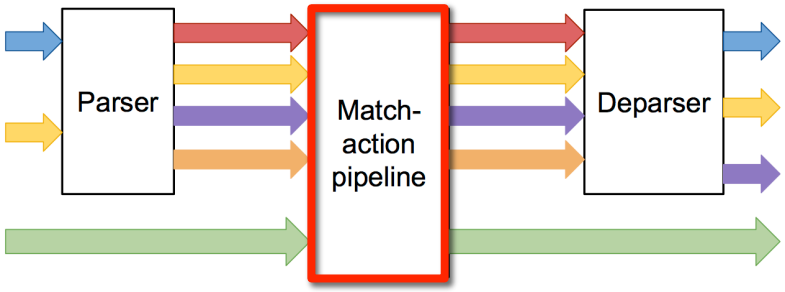
Learning Switch – Control Flow




```
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 = {
      p.ethernet.dstAddr: exact;
    }

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

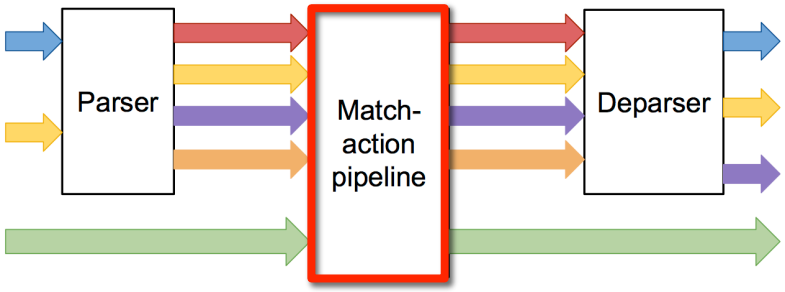
Learning Switch – Control Flow




```
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;
        NoAction;
    }
    size = 64;
    default_action = NoAction;
}
```

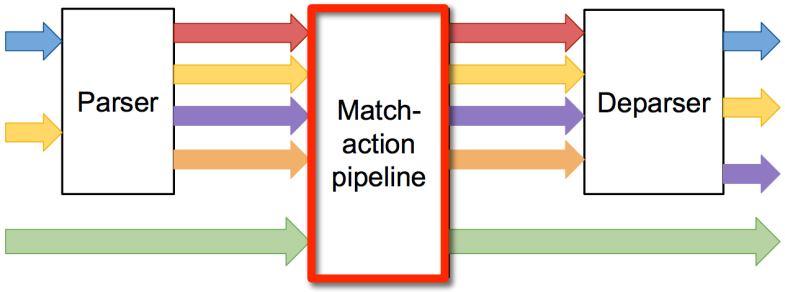
Learning Switch – Control Flow




```
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 = {
      p.ethernet.srcAddr: exact;
    }

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

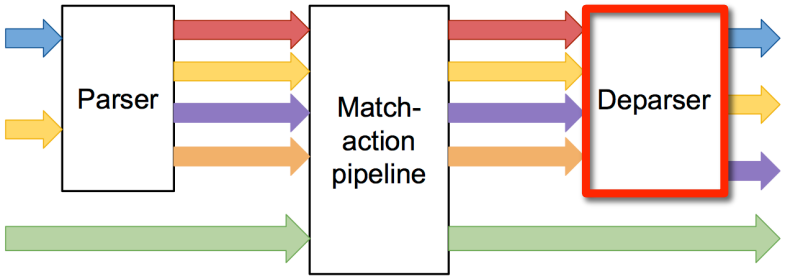
Learning Switch – Control Flow




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

Learning Switch – Deparser




```
// Deparser Implementation
@Xilinx_MaxPacketRegion(16384)
control TopDeparser(packet_out b,
    in Parsed_packet p,
    in user_metadata_t user_metadata,
    inout digest_data_t digest_data,
    inout sume_metadata_t sume_metadata) {

    apply {
        b.emit(p.ethernet);
    }
}
```

- Reconstruct the packet
- emit(header) – serialize the header **if** it is valid
- `@Xilinx_MaxPacketRegion` – for parser/deparser; declares the largest packet size (in bits) to support

Learning Switch – Control-Plane

```
DMA_IFACE = 'nf0'

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

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
    eth_src_addr = dig_pkt.eth_src_addr
    (found, val) = table_cam_read_entry('forward', ...
                                        [eth_src_addr])
    if (found == 'False'):
        table_cam_add_entry('forward', ...
                            [eth_src_addr], ...
                            'set_output_port', ...
                            [src_port])
        table_cam_add_entry('smac', ...
                             [eth_src_addr], ...
                            'NoAction_3', [])
```

- Monitor DMA interface
- Convert DMA data to appropriate format
- Check if entry already exists in the forwarding table
- Add entry to forwarding table
- Add entry to smac table

API

- Auto generated Python API
- Exact match table API functions:
  - table_cam_add_entry()
  - table_cam_delete_entry()
  - table_cam_get_size()
  - table_cam_read_entry()
- Ternary match table API functions:
  - table_tcam_add_entry()
  - table_tcam_clean()
  - table_tcam_erase_entry()
  - table_tcam_verify_entry()
- LPM table API functions:
  - table_lpm_load_dataset()
  - table_lpm_set_active_lookup_bank()
  - table_lpm_verify_dataset()
- Register API functions:
  - reg_read()
  - reg_write()
- C API also available

Externs

- Implement platform specific functions
  - Can't be expressed in the core P4 language
- Stateless – reinitialized for each packet
- Stateful – keep state between packets
- Xilinx Annotations
  - `@Xilinx_MaxLatency()` – maximum number of clock cycles an extern function needs to complete
  - `@Xilinx_ControlWidth()` – size in bits of the address space to allocate to an extern function

P4->NetFPGA Extern Function Library

- Verilog modules invoked from within P4 programs
- Stateful Atoms

Atom	Description
R/W	Read or write state
RAW	Read, add to, or overwrite state
PRAW	Predicated version of RAW
ifElseRAW	Two RAWs, one each for when predicate is true or false
Sub	IfElseRAW with stateful subtraction capability

- Stateless Externs

Atom	Description
IP Checksum	Given an IP header, compute IP checksum
LRC	Longitudinal redundancy check, simple hash function
timestamp	Generate timestamp (granularity of 5 ns)

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
@Xilinx_MaxLatency(1)
@Xilinx_ControlWidth(3)
extern void count(in bit<3> index,
                  in bit<32> newVal,
                  in bit<32> incVal,
                  in bit<8> opCode,
                  out 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

- **Single** state access!

P4->NetFPGA Workflow Overview

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. Check Timing
9. Test the hardware

All of your effort will go here

Directory Structure of $SUME_FOLDER

```
P4-NetFPGA-live/
|
|- 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

```
$P4_PROJECT_DIR/
|
|- src/ → P4 source files and commands.txt
|
|- 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 and any
|         user software for the project
|
|- 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

Tutorial Assignments

Tutorial Assignments

- Assignments described at this link:
  - https://github.com/NetFPGA/P4-NetFPGA-public/wiki/Tutorial-Assignments
- Short version:
  - https://tinyurl.com/P4-NetFPGA-Camp

Assignment 1: Switch as a Calculator

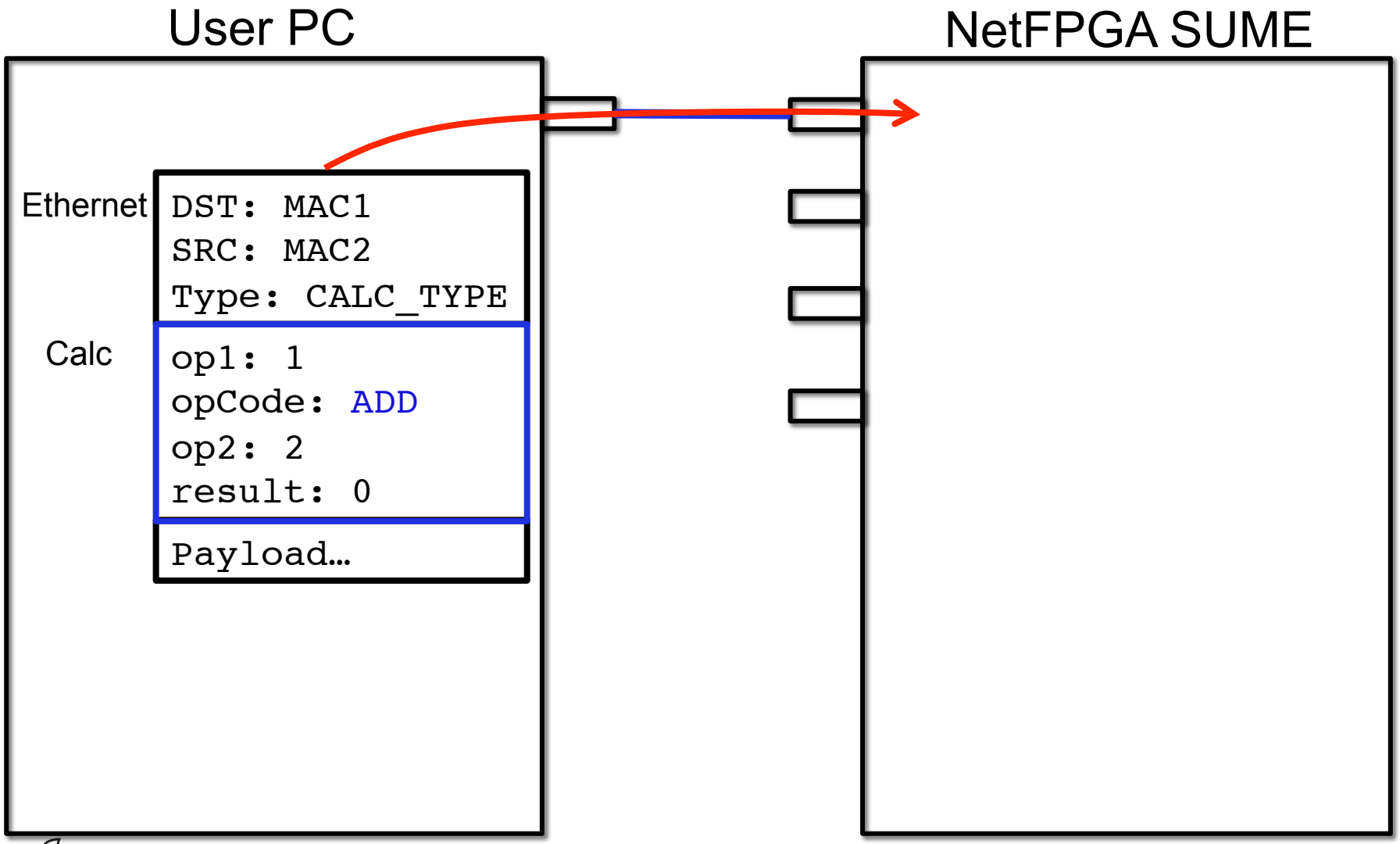
Switch as Calculator

Supported Operations:

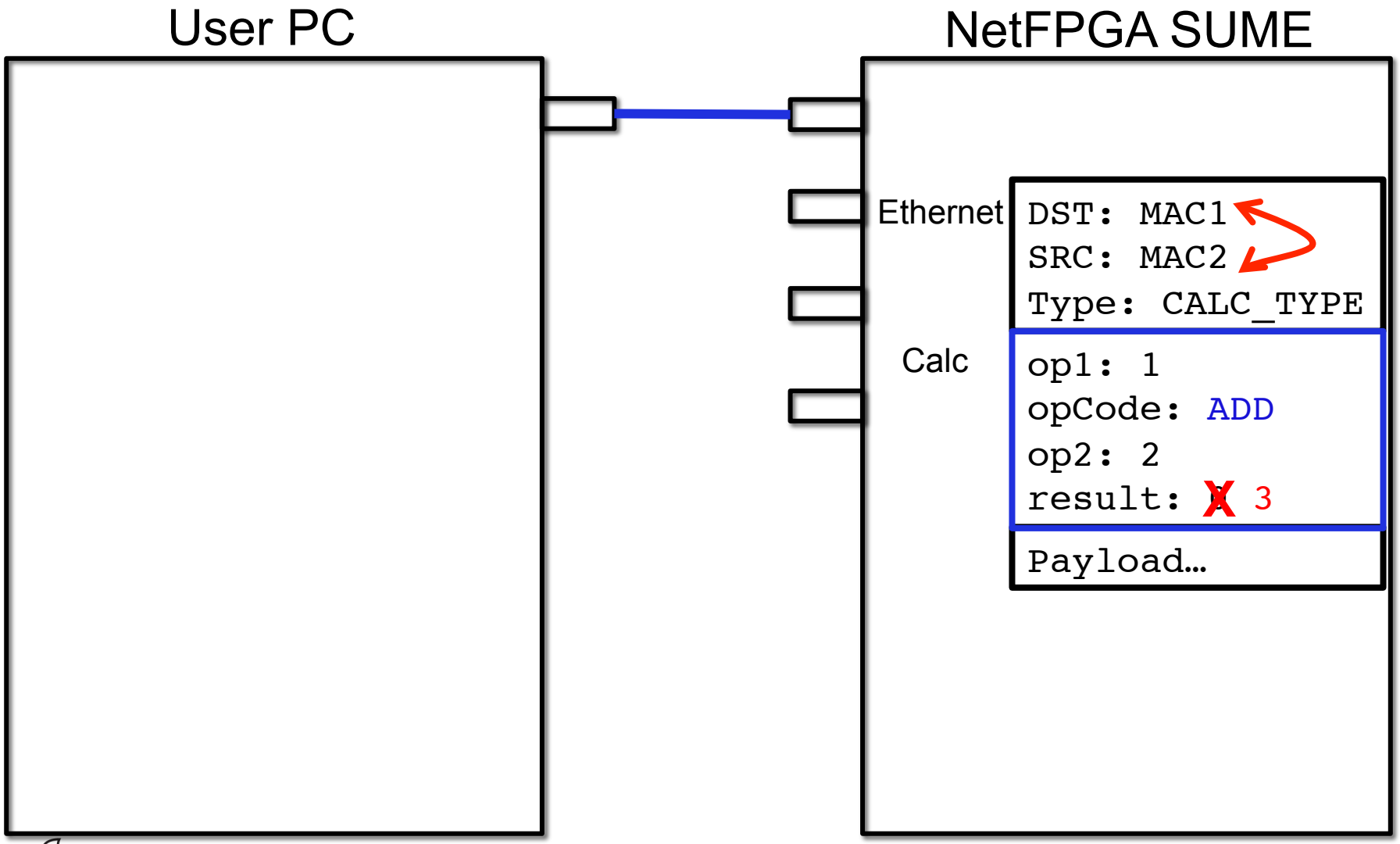
- ADD – add two operands
- SUBTRACT – subtract two operands
- ADD_REG – add operand to current value in register
- SET_REG – overwrite the current value of the register
- LOOKUP – Lookup the given key in the table

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

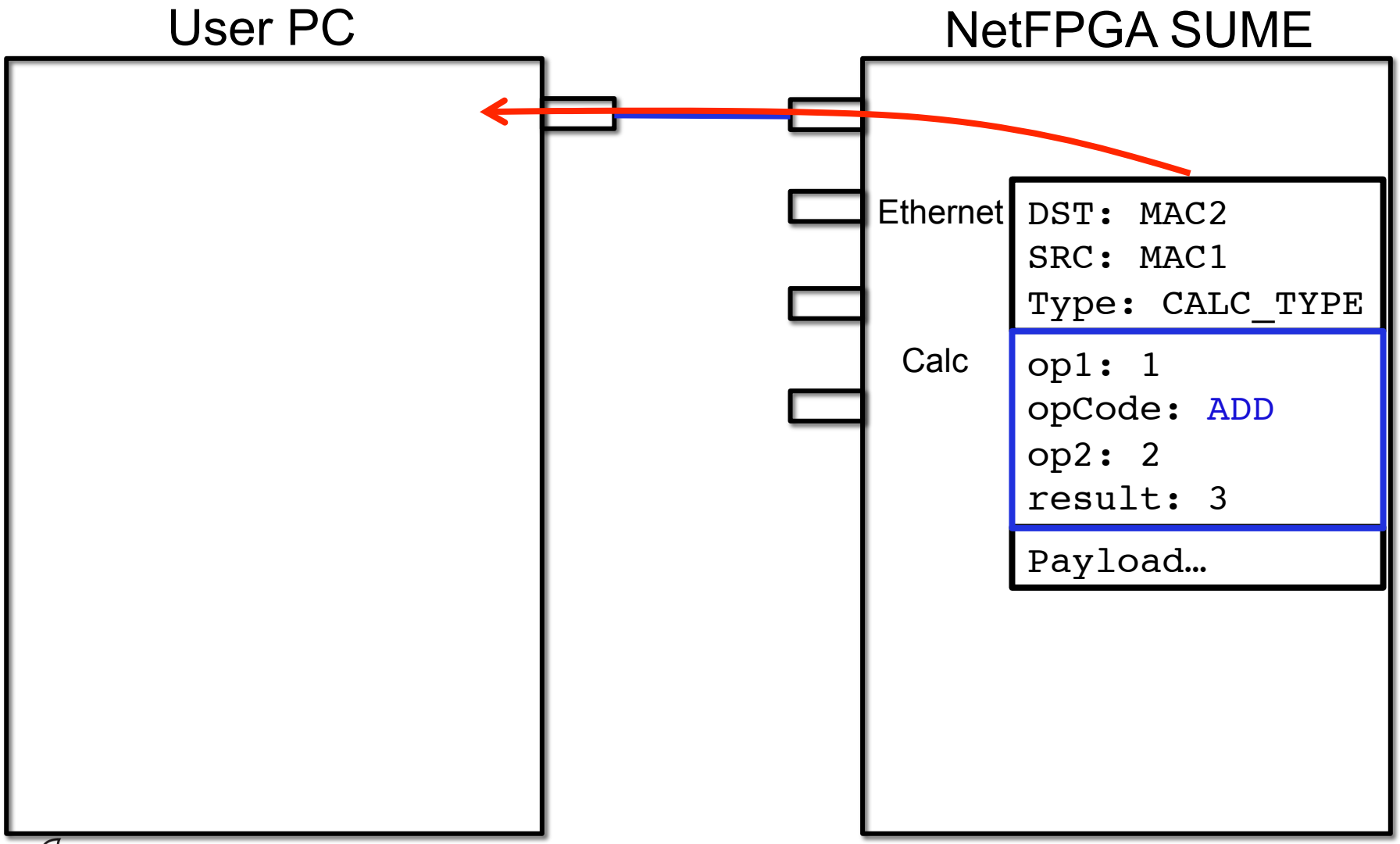
Switch as Calculator



Switch as Calculator

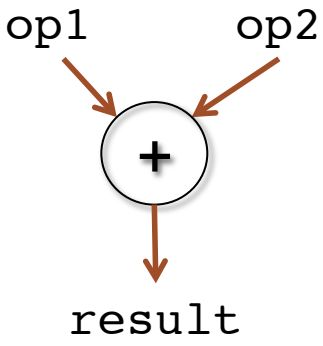


Switch as Calculator

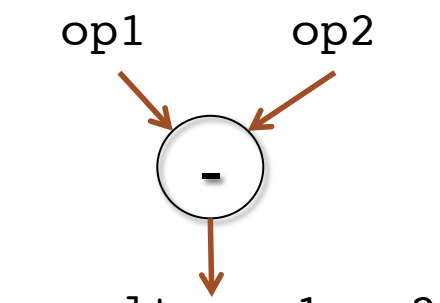


Switch Calc Operations

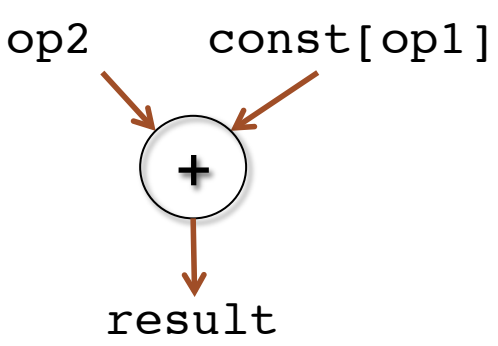
ADD



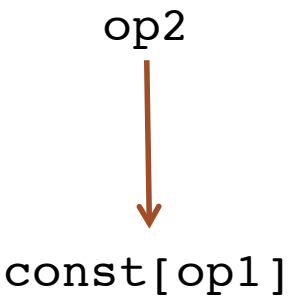
SUB



ADD_REG



SET_REG



LOOKUP

key: op1 →

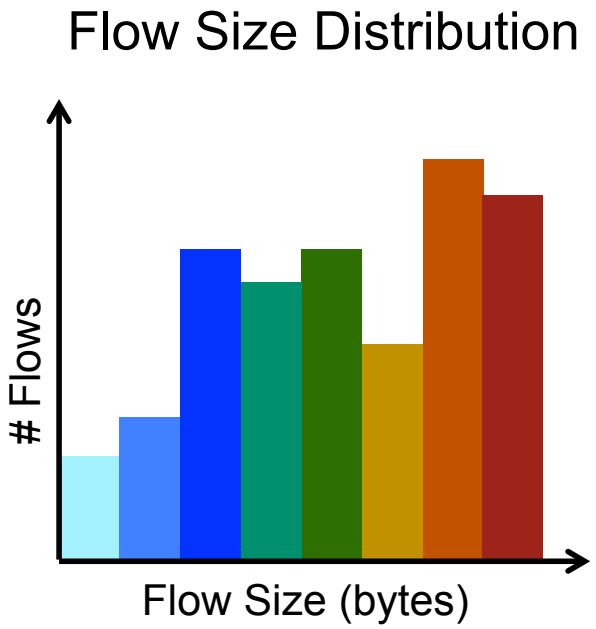
key	val
0	1
1	16
2	16^2
3	16^3

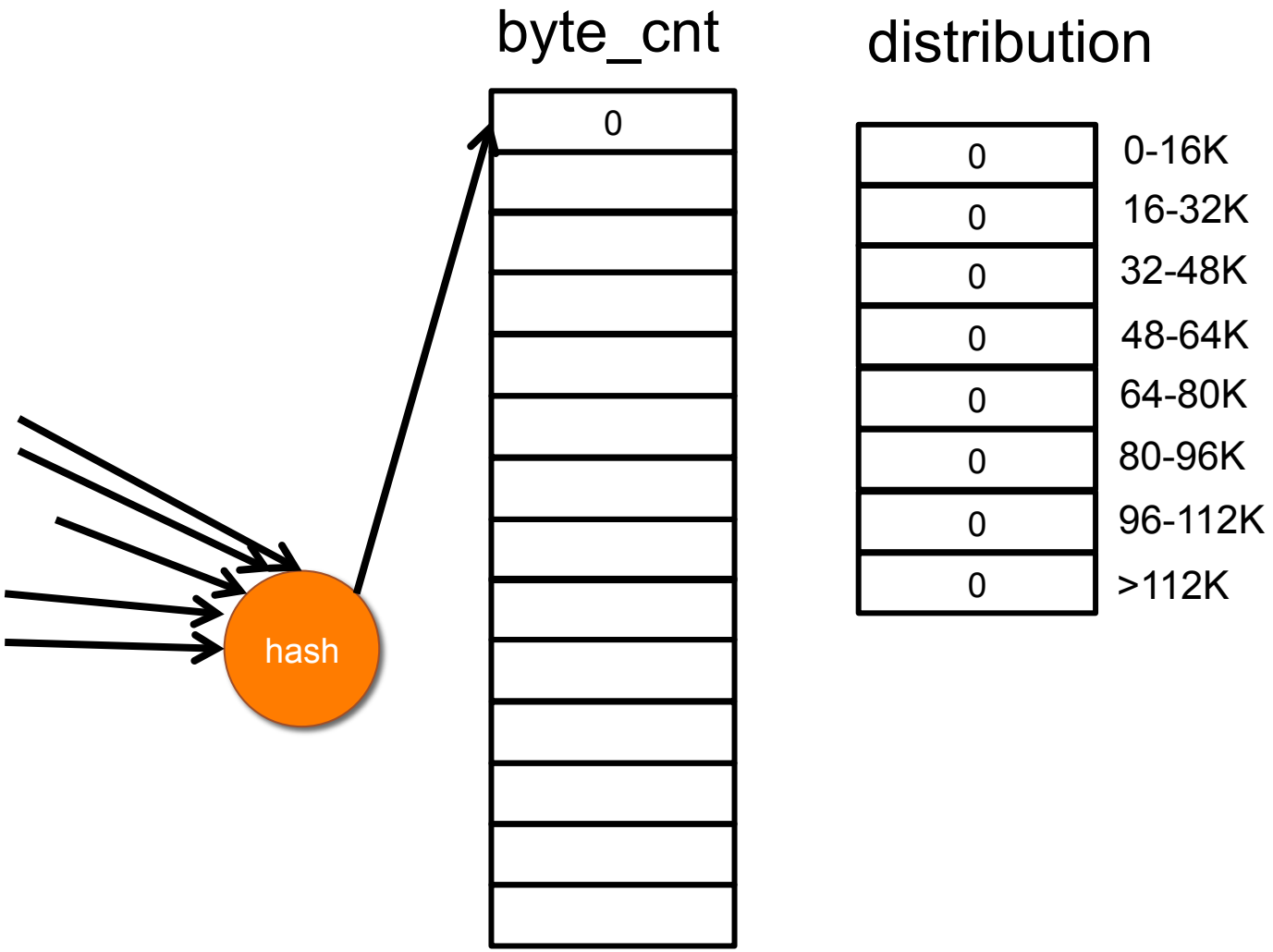
→ result: val

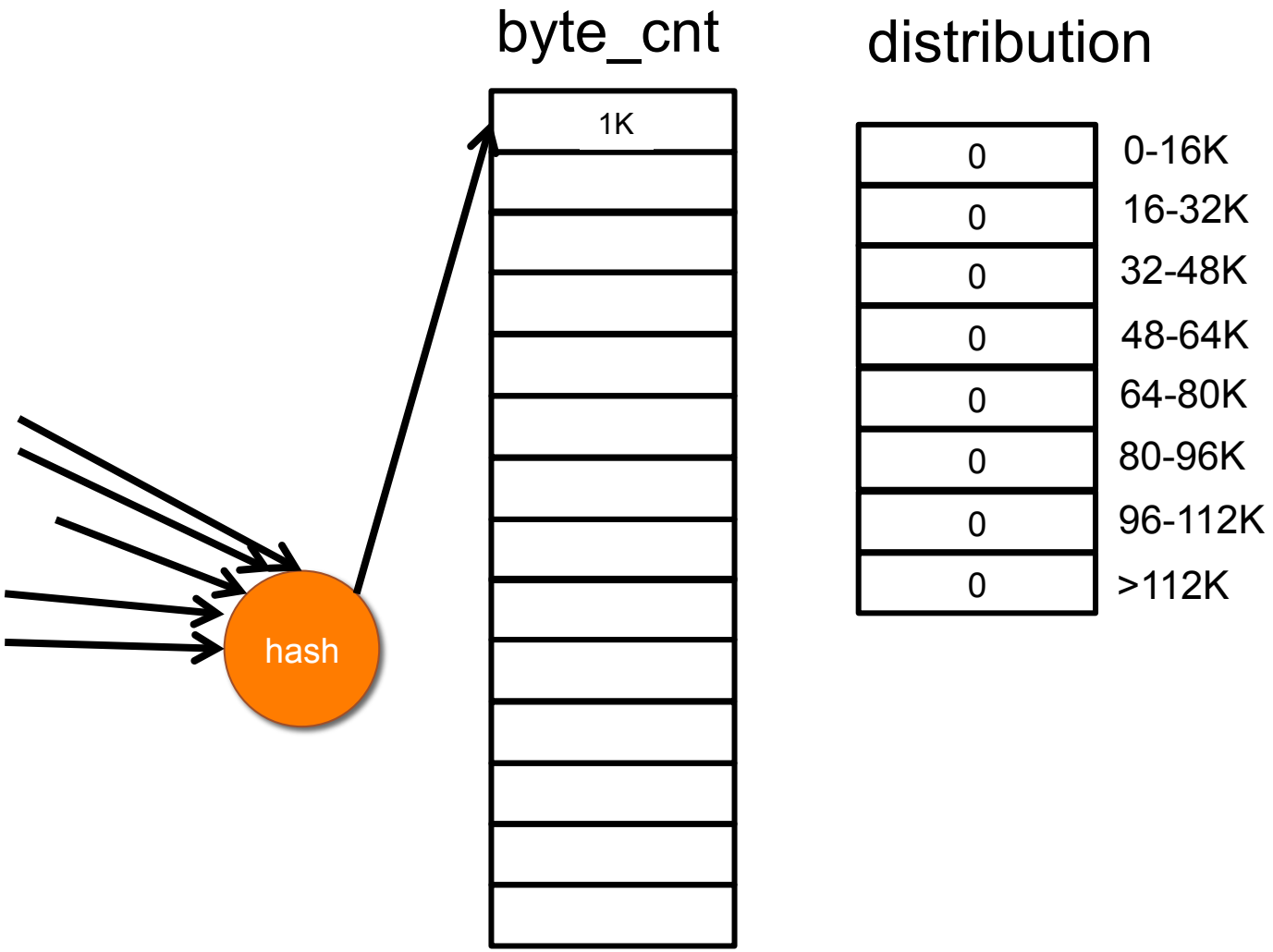
Assignment 2: TCP Monitor

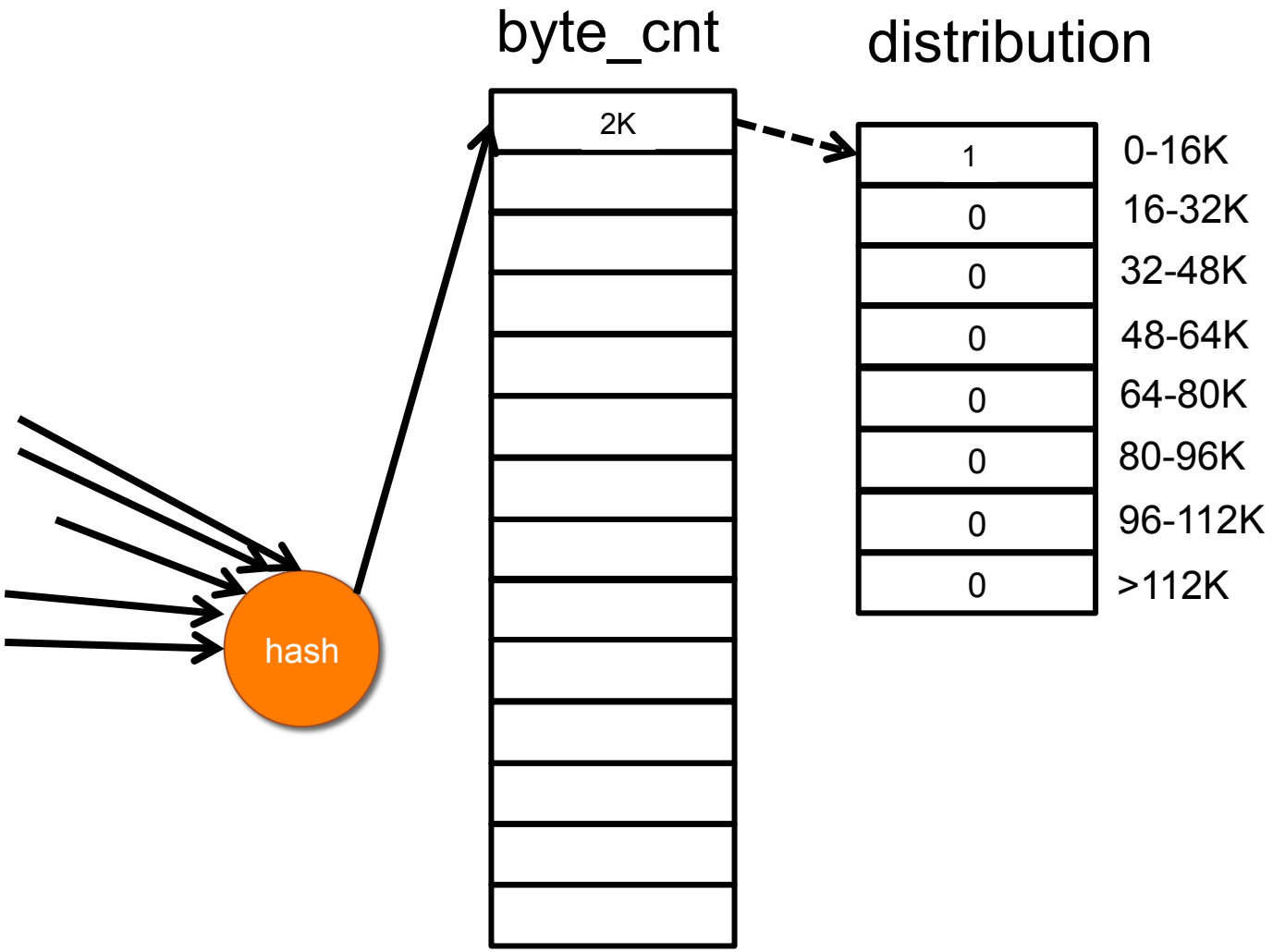
TCP Monitor

- 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)

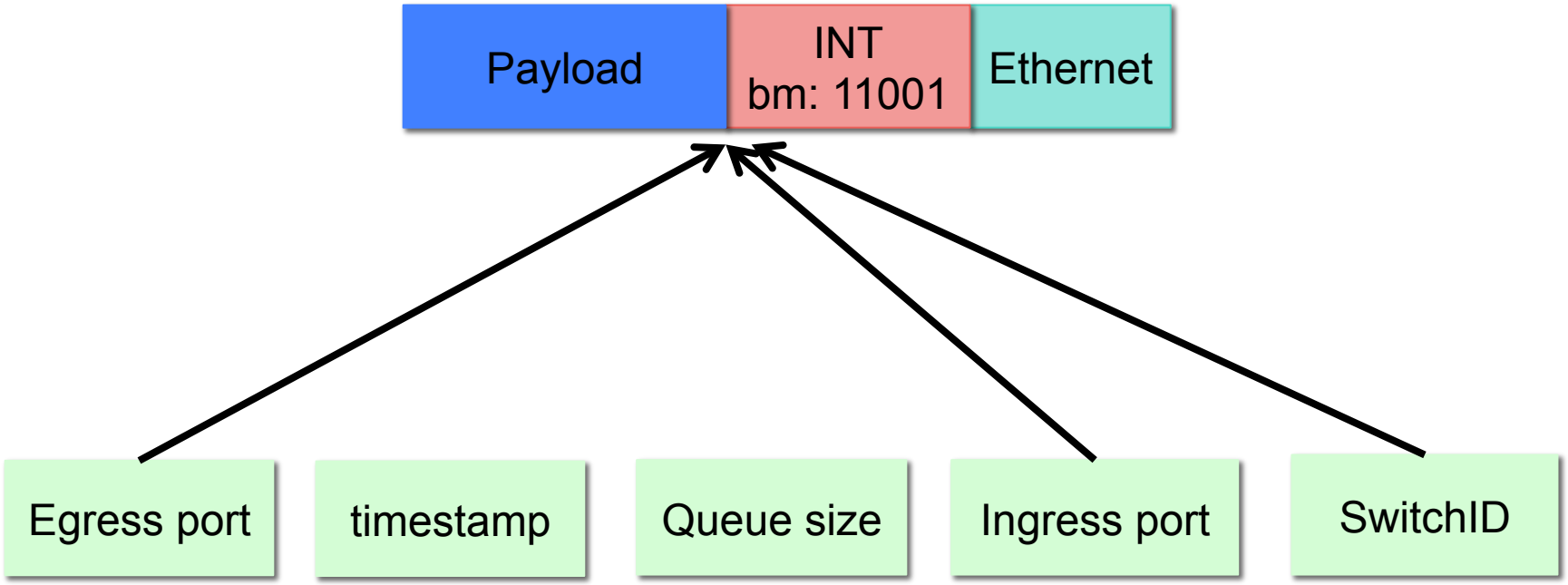
In-band Network Telemetry (INT)

- 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):

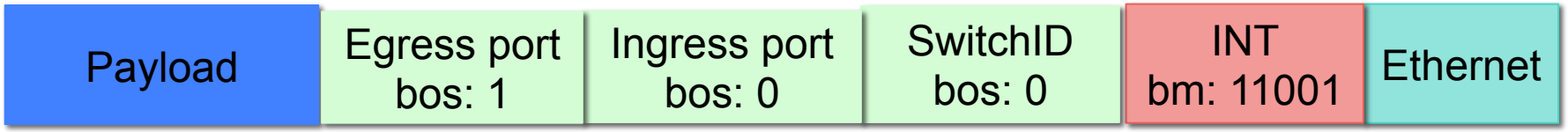
<SWITCH_ID><INGRESS_PORT><Q_SIZE><TSTAMP><EGRESS_PORT>

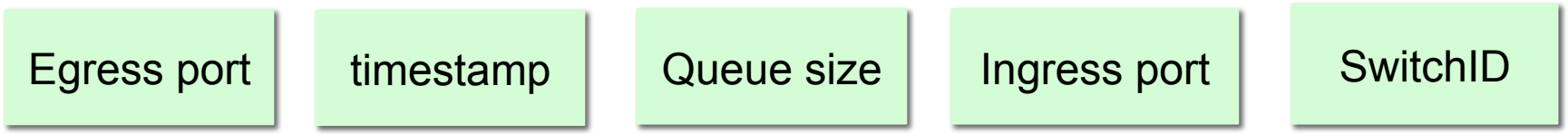


In-band Network Telemetry (INT)

- Bitmask format (5 bits):

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





Additional Details

Implementing Extern Functions

1. Implement verilog extern module in $SUME_SDNET/templates/externs/
2. Add entry to $SUME_SDNET/bin/extern_data.py

- No Need to modify any existing code
- If the extern has a control interface a cpu_regs module will be generated automatically to easily expose control_registers using the AXI Lite interface

Debugging SDNet Simulation

- HDLSimulation
  - Xsim or Questa sim
  - Error:

```
expected < tuple_out_sume_metadata > = < 000000000000000000000100040041 >
actual   < tuple_out_sume_metadata > = < 000000000000000000000100040040 >

$ $P4_PROJECT_DIR/testdata/sss_sdnet_tuples.py --parse
   000000000000000000000100040041 sume
Parsed Tuple:
-----------------------
dma_q_size  =  0
nf3_q_size  =  0
nf2_q_size  =  0
nf1_q_size  =  0
nf0_q_size  =  0
send_dig_to_cpu  =  0
drop  =  1
dst_port  = 00000000
src_port  = 00000100
pkt_len  =  65
```

Debugging SDNet Simulation

- C++ model
  - Prints debug info
    - Hit or miss in tables
    - Header/Metadata field modification along the way
  - Generates Packet_expect.pcap to compare to dst.pcap
- -skipEval option
  - Compare C++ model output to HDL simulation – make sure SDNet compiler is working properly
- Future Improvements:
  - SDNet C++ model implementation of extern functions

Debugging SUME Simulation

- What to do if SDNet simulation passes but SUME simulation fails?
  - Make sure you've:
    - $ cd $P4_PROJECT_DIR && make config_writes
    - $ cd $NF_DESIGN_DIR/test/sim_major_minor && make
    - $ cd $P4_PROJECT_DIR && make install_sdnet
  - Run SUME simulation longer?
  - Make sure SDNet module has finished reset before applying configuration writes
  - Check nf_*_expected.axi and nf_*_logged.axi
  - Add signals to simulation waveform
  - Perhaps issue with wrapper module?

SDNet Module HDL Wrapper

- Located in: $SUME_SDNET/templates/sss_wrapper/
- Tasks:
  - Generate tuple_in_VALID signal for s_axis_tuser
  - Reverse reset polarity
  - Reverses endianness of digest_data and inserts into last 80 bits of m_axis_tuser