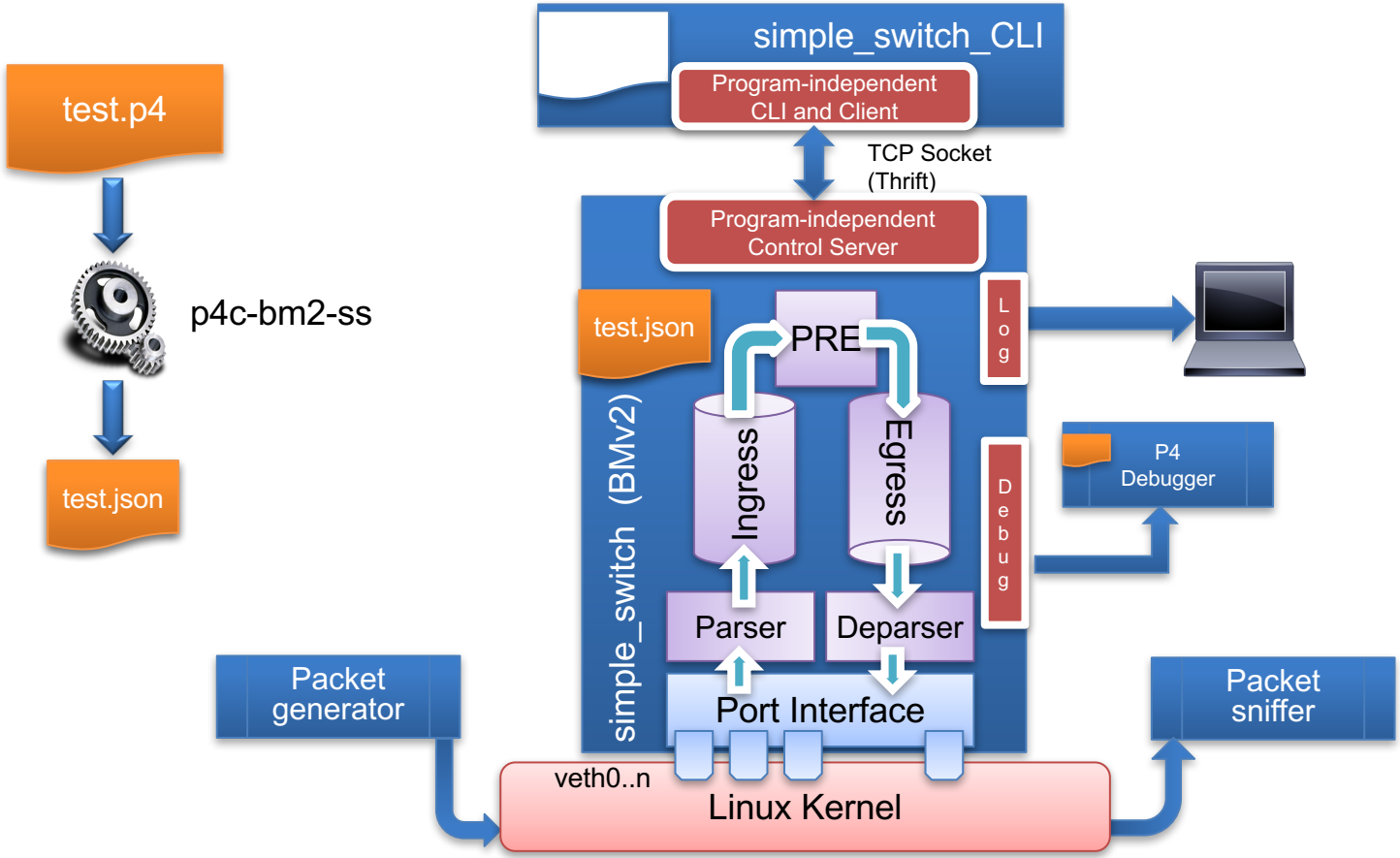
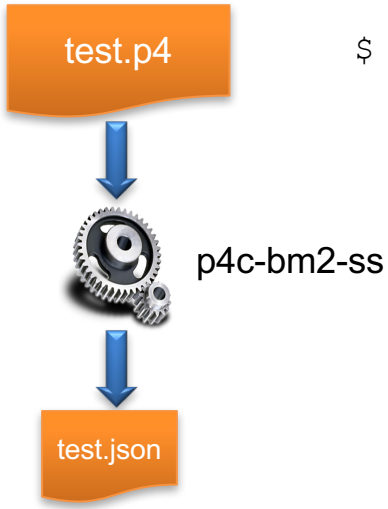
CS344 – Lecture 3

P4 Toolchain for BMv2 software simulation

Basic Workflow

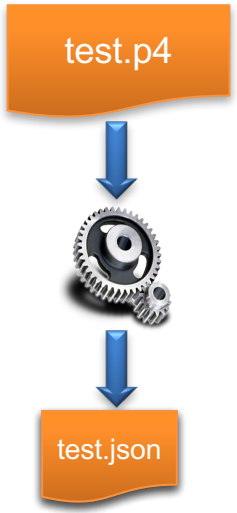


Step 1: P4 Program Compilation

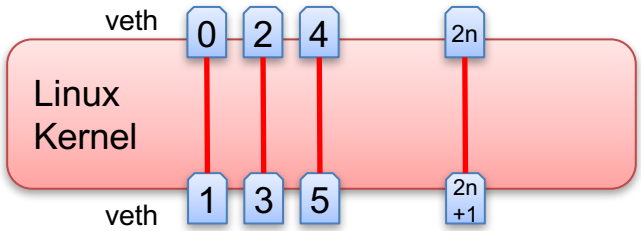



```
$ p4c-bm2-ss -o test.json test.p4
```

Step 2: Preparing veth Interfaces

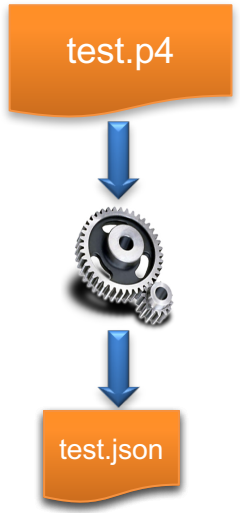


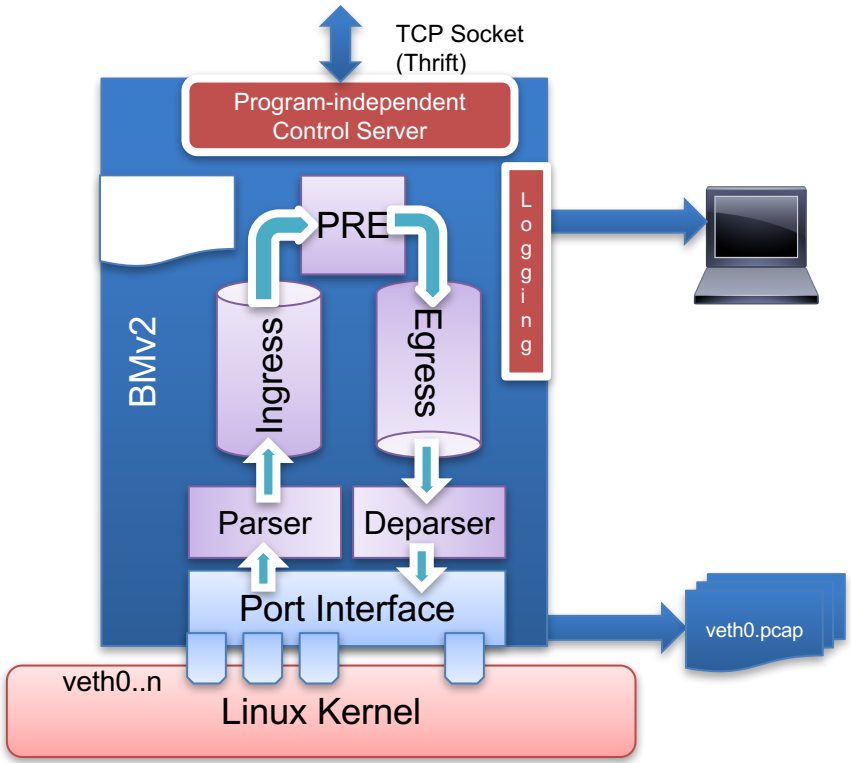

```
$ sudo ~/p4lang/tutorials/examples/veth_setup.sh

# ip link add name veth0 type veth peer name veth1
# for iface in “veth0 veth1”; do
    ip link set dev ${iface} up
    sysctl net.ipv6.conf.${iface}.disable_ipv6=1
    TOE_OPTIONS="rx tx sg tso ufo gso gro lro rxvlan txvlan rxhash”
    for TOE_OPTION in $TOE_OPTIONS; do
        /sbin/ethtool --offload $intf "$TOE_OPTION”
    done
  done
```



Step 3: Starting the model

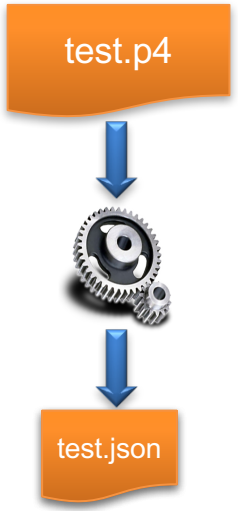
```
$ sudo simple_switch --log-console --dump-packet-data 64 \
       -i 0@veth0 -i 1@veth2 …                           \
       test.json
```

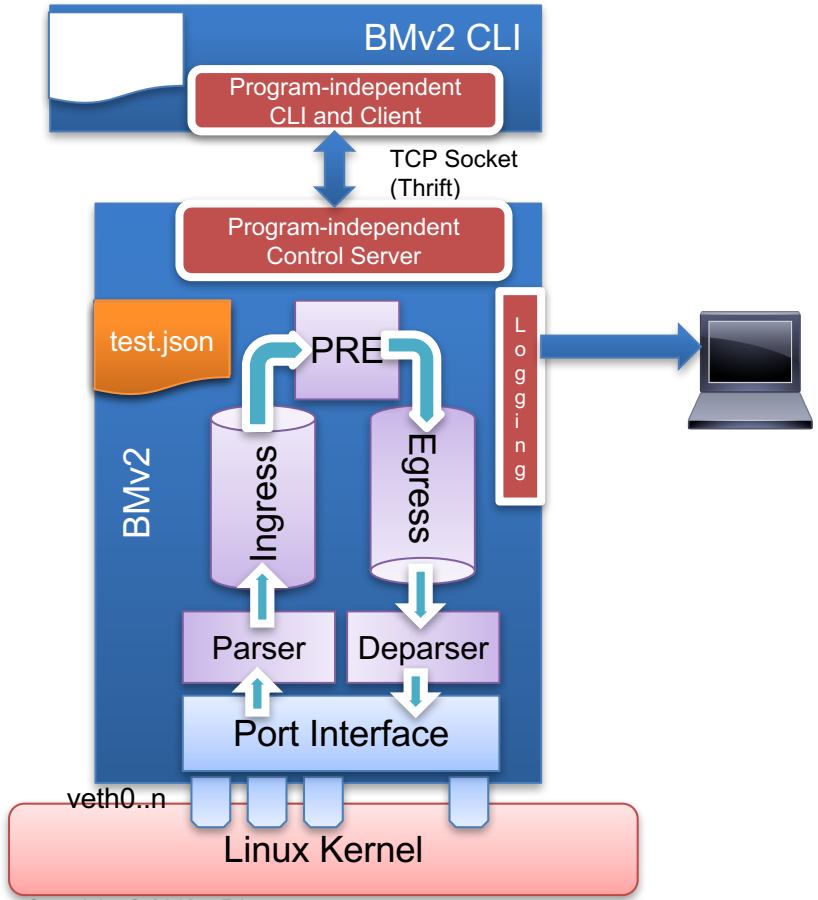




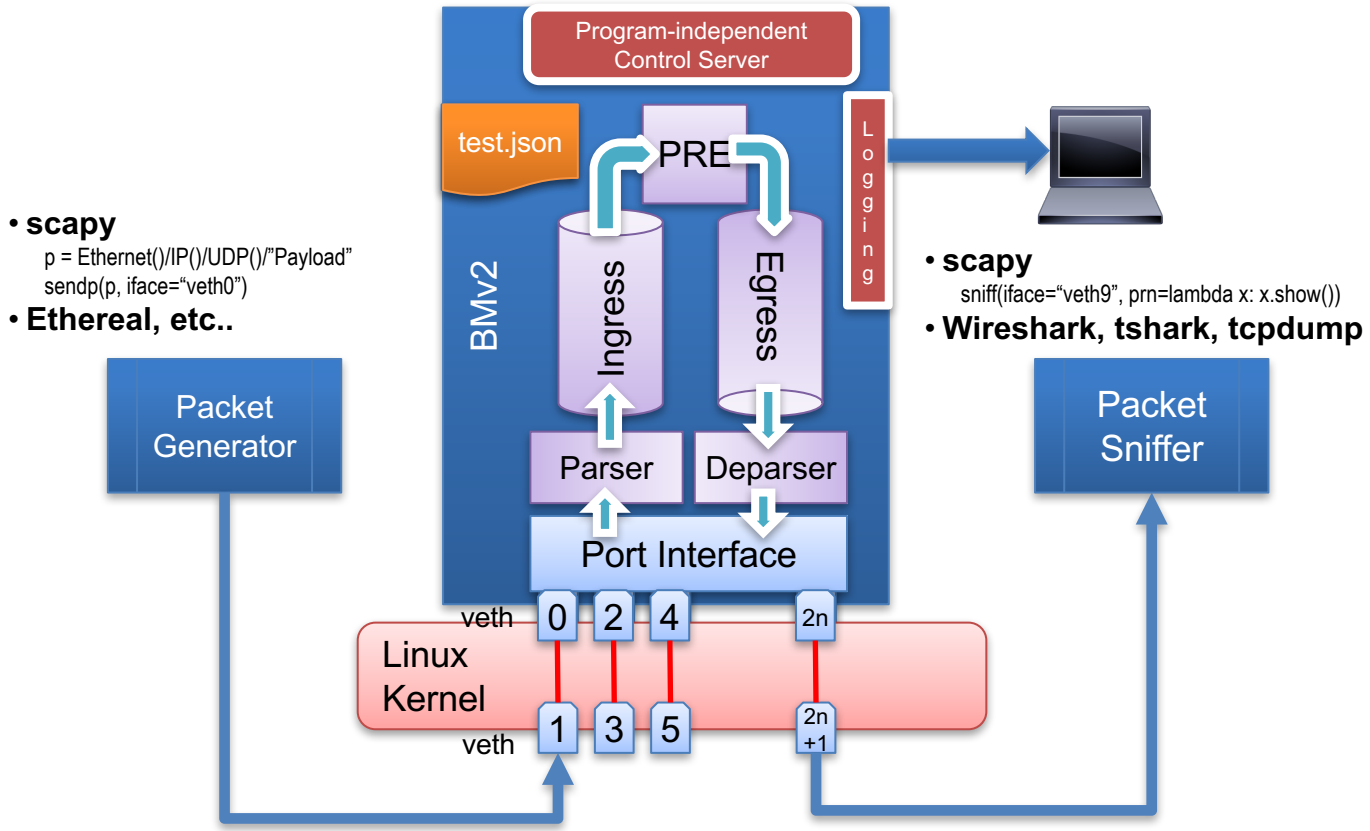
Step 4: Starting the CLI

```
$ simple_switch_CLI
```





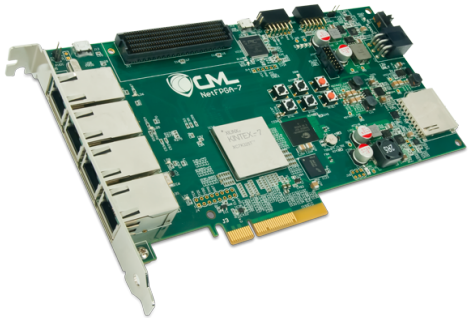
Step 5: Sending and Receiving Packets

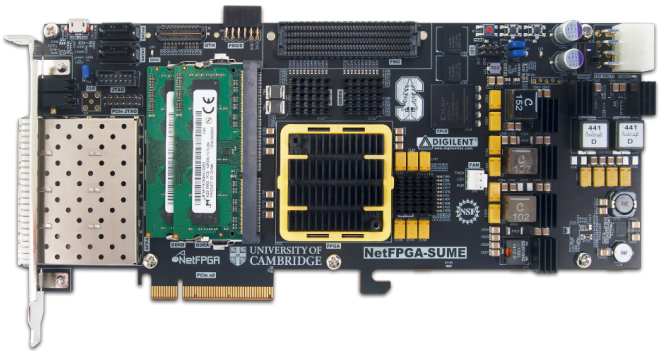


Overview

NetFPGA = Networked FPGA

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



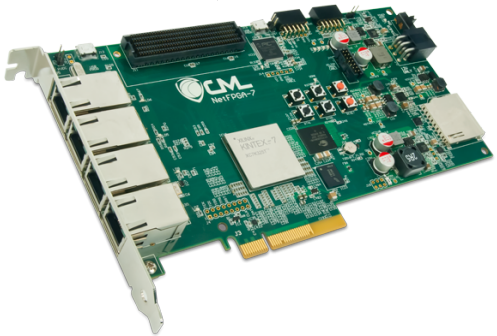


=

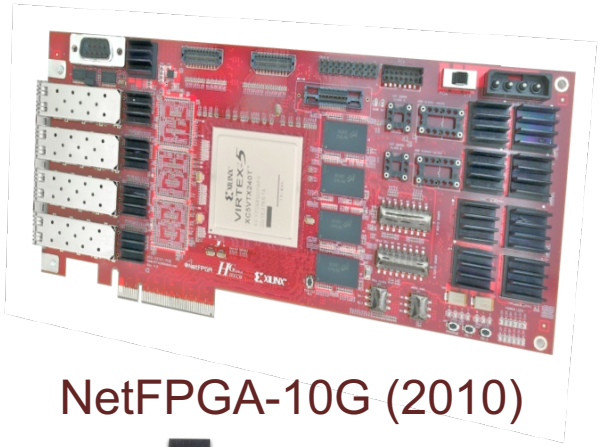
- Network Interface Card
- Hardware Accelerated Linux Router
- IPv4 Reference Router
- Traffic Generator
- Openflow Switch
- More Projects
- Add Your Project

NetFPGA Family of Boards

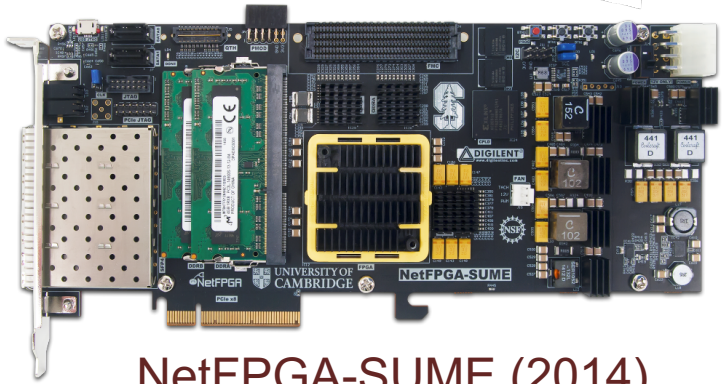
NetFPGA-1G (2006)



NetFPGA-1G-CML (2014)



NetFPGA-10G (2010)



NetFPGA-SUME (2014)

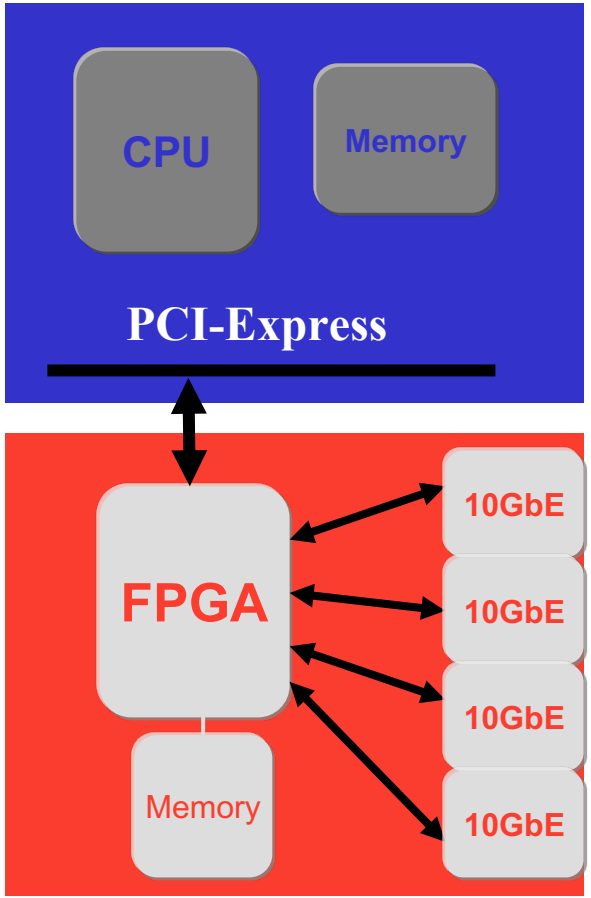
International Community

- **Over 1,200 users, using over 3500 cards at 200 universities in over 47 countries**

- **Join the mailing list: cl-netfpga-sume-beta@lists.cam.ac.uk**

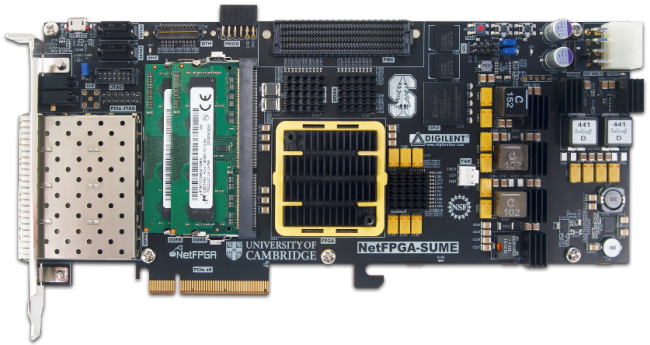
NetFPGA board

Networking Software running on a standard PC



PC with NetFPGA

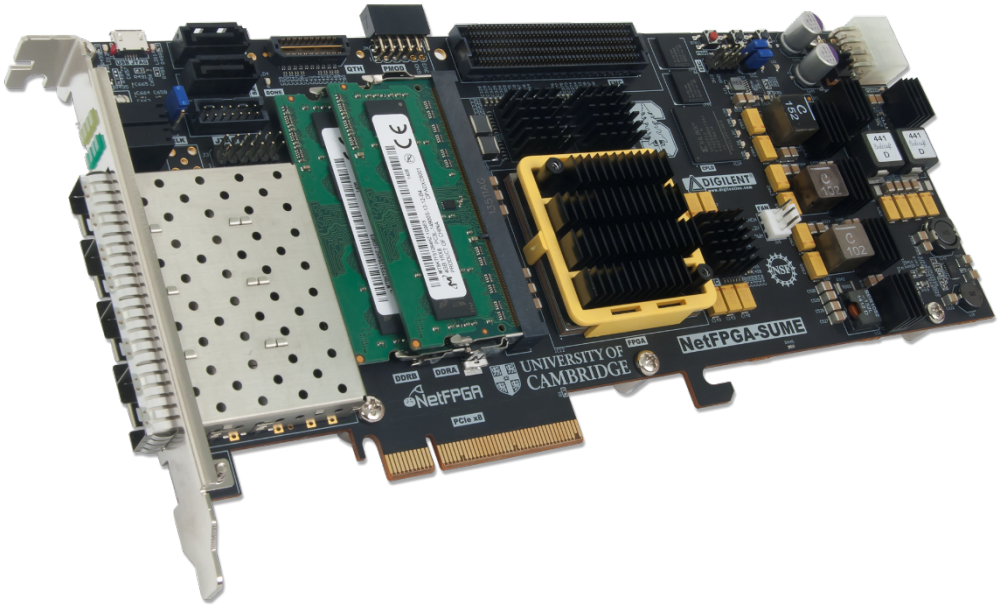
A hardware accelerator built with FPGA driving 1/10/ 100Gb/s network links



NetFPGA consists of …

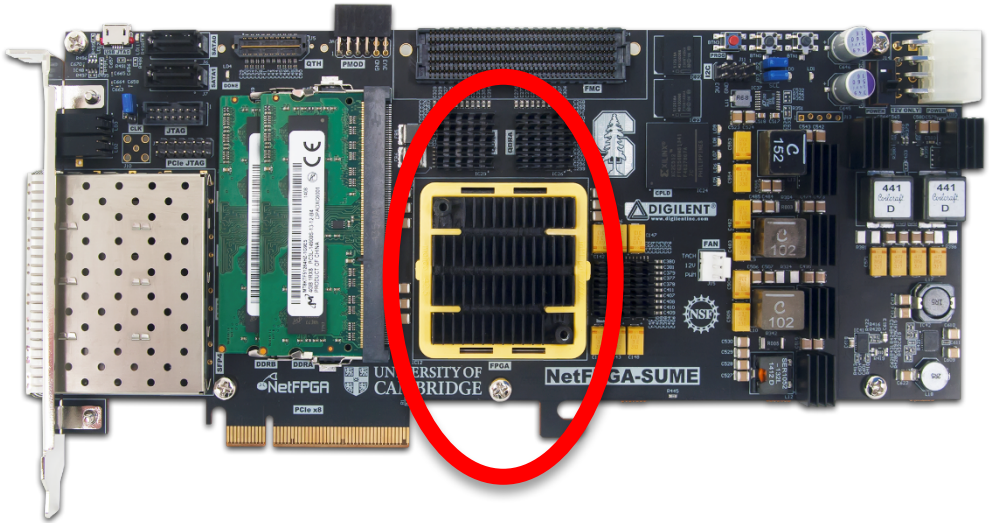
Four elements:

- **NetFPGA board**
- **Tools + reference designs**
- **Contributed projects**
- **Community**



Xilinx Virtex 7 690T

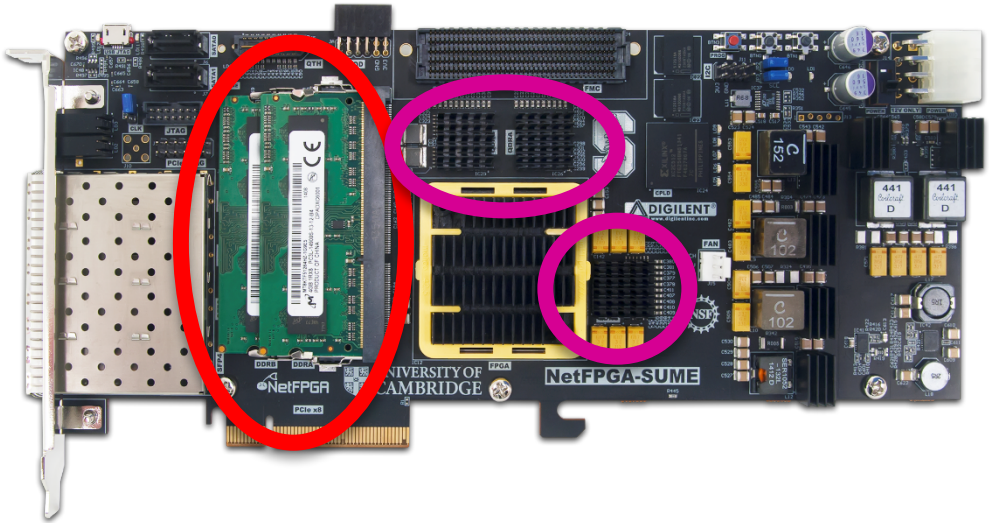
- Optimized for high-performance applications
- 690K Logic Cells
- 52Mb RAM
- 3 PCIe Gen. 3 Hard cores



Memory Interfaces

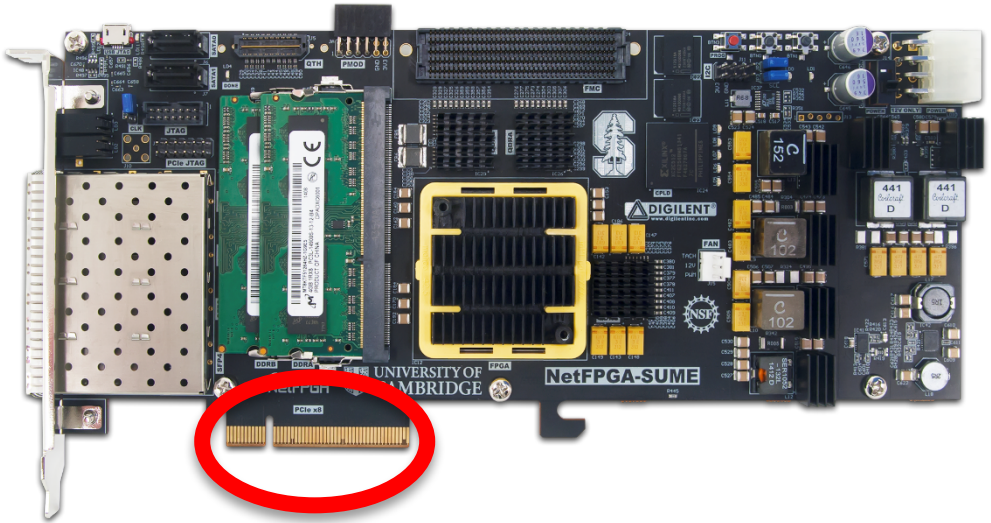
- **DRAM:**
  2 x DDR3 SoDIMM
  1866MT/s, 4GB

- **SRAM:**
  3 x 9MB QDRII+, 500MHz



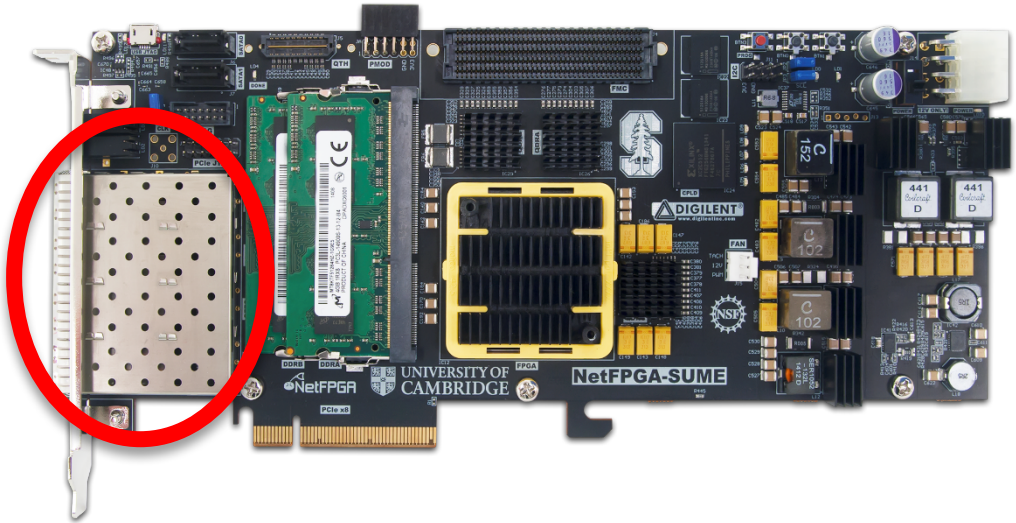
Host Interface

- PCIe Gen. 3
- x8 (only)
- Hardcore IP



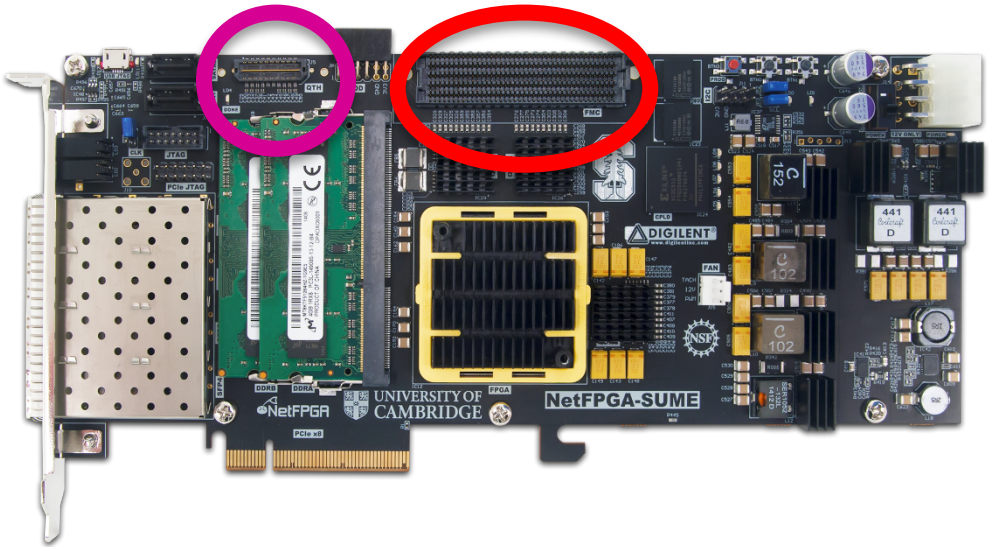
Front Panel Ports

- 4 SFP+ Cages
- Directly connected to the FPGA
- Supports 10GBase-R transceivers (default)
- Also Supports 1000Base-X transceivers and direct attach cables



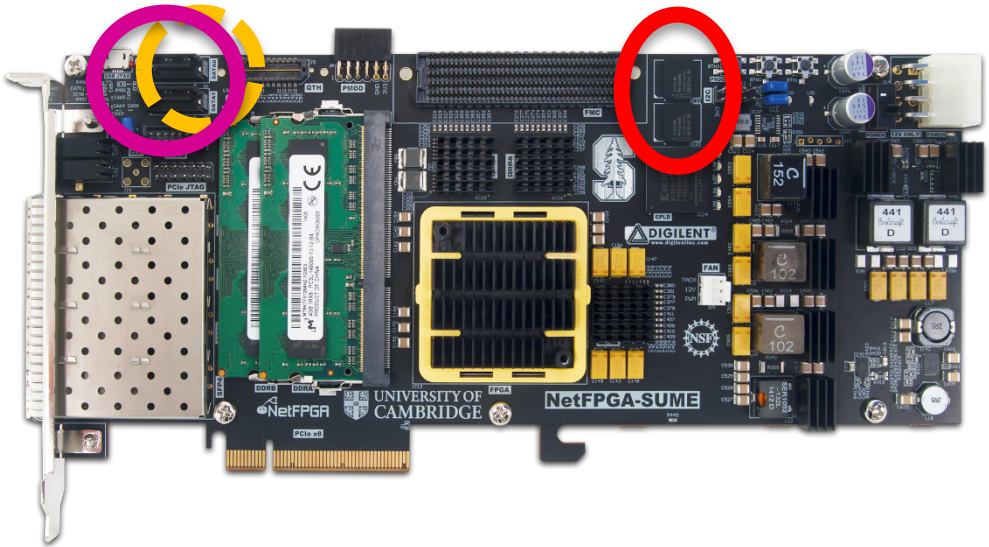
Expansion Interfaces

- **FMC HPC connector**
  - VITA-57 Standard
  - Supports Fabric Mezzanine Cards (FMC)
  - 10 x 12.5Gbps serial links
- **QTH-DP**
  - 8 x 12.5Gbps serial links



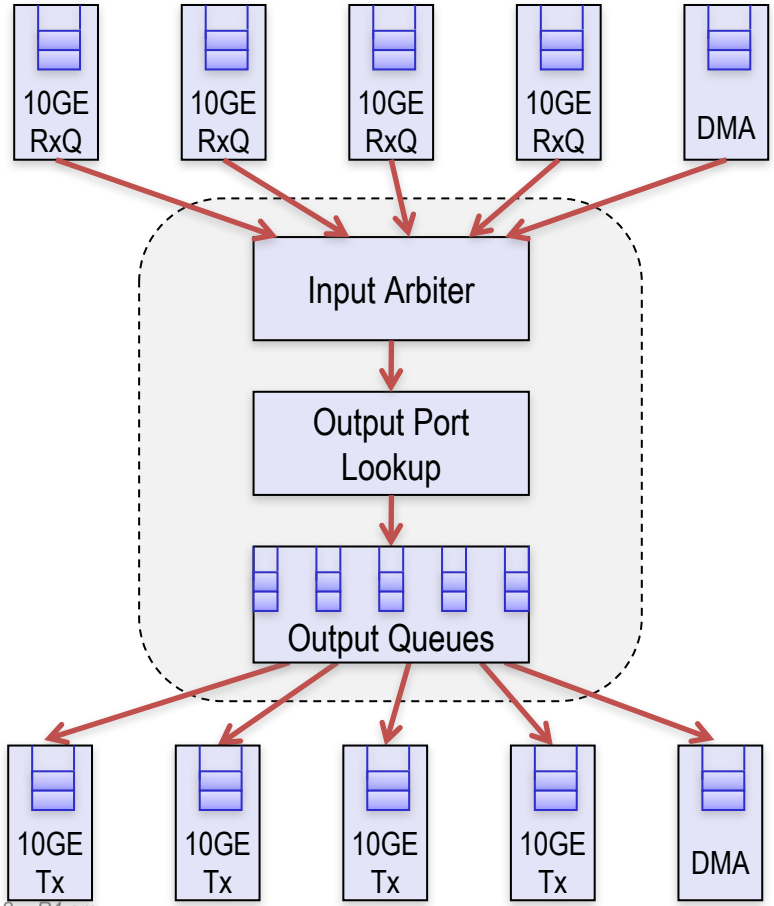
Storage

- 128MB FLASH
- 2 x SATA connectors
- Micro-SD slot
- Enable standalone operation

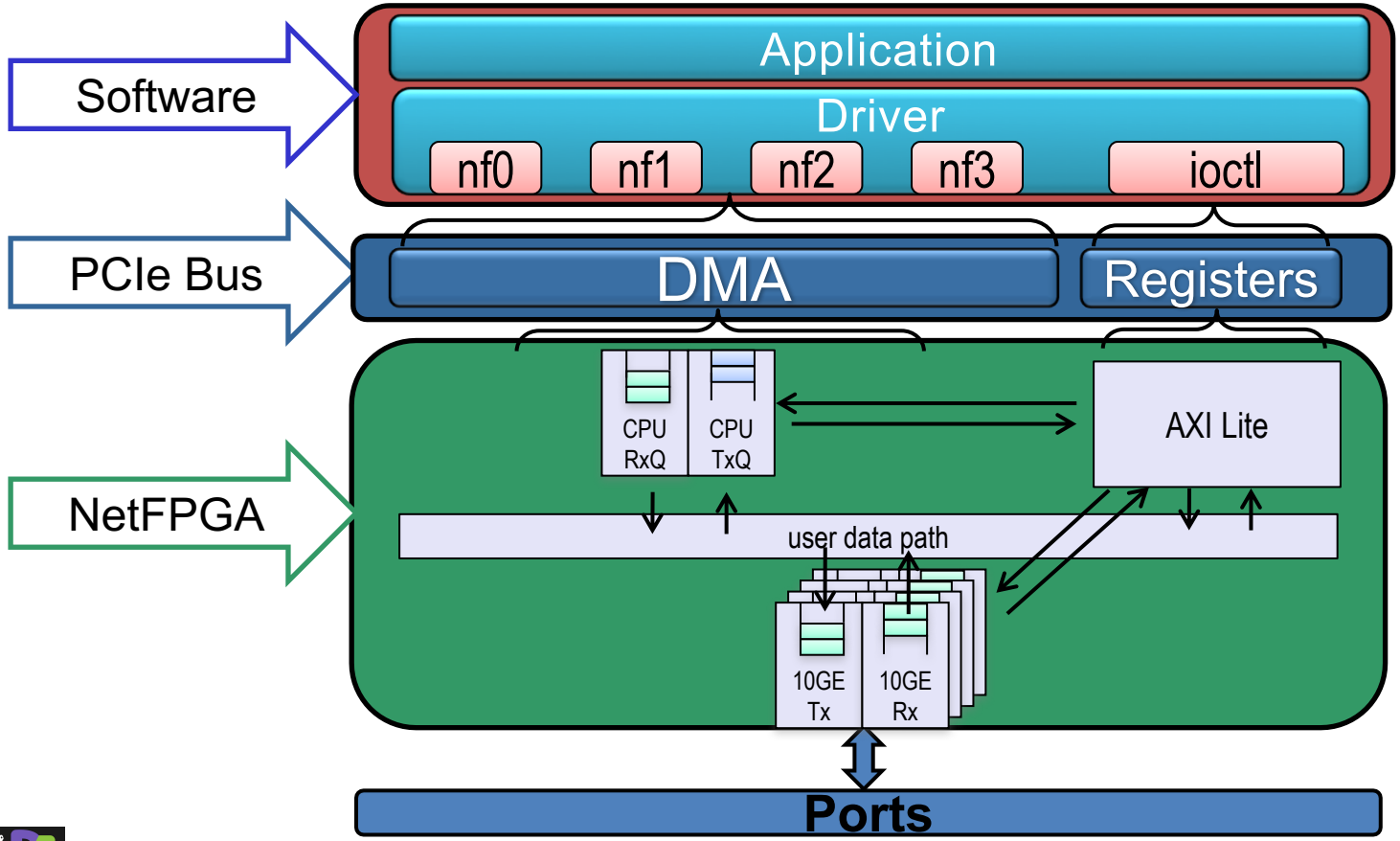


Reference Switch Pipeline

- **Five stages**
  - Input port
  - Input arbitration
  - Forwarding decision and packet modification
  - Output queuing
  - Output port
- **Packet-based module interface**
- **Pluggable design**



Full System Components

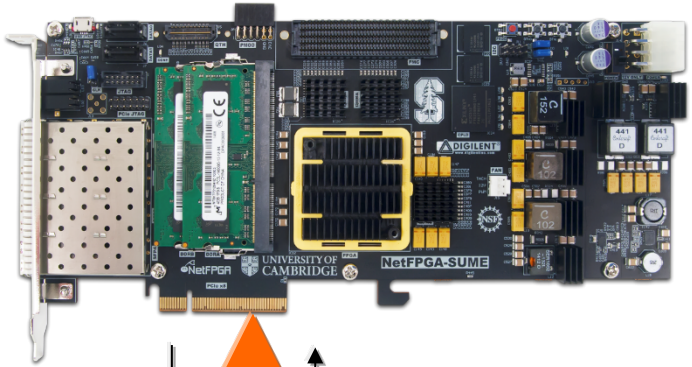


NetFPGA – Host Interaction

- **Linux driver interfaces with hardware**
  - Packet interface via standard Linux network stack
  - Register reads/writes voa ioctl system call with wrapper functions
    - `rwaxi(int address, unsigned *data);`
    - Eg: `rwaxi(0x7d4000000, &val)`

NetFPGA to Host Packet Transfer

1. Packet arrives – forwarding table sends to DMA queue

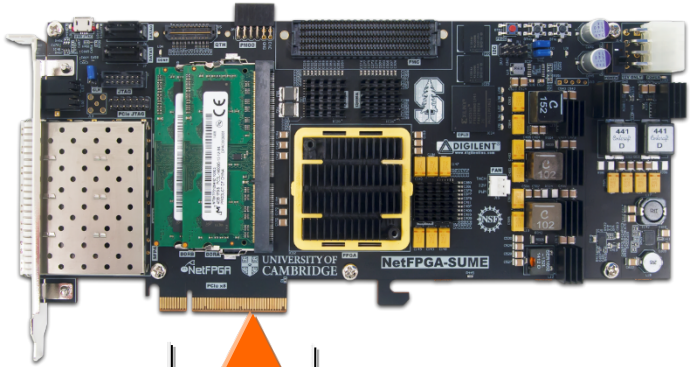


2. Interrupt notifies driver of packet arrival

PCIe Bus

3. Driver sets up and initiates DMA transfer

NetFPGA to Host Packet Transfer



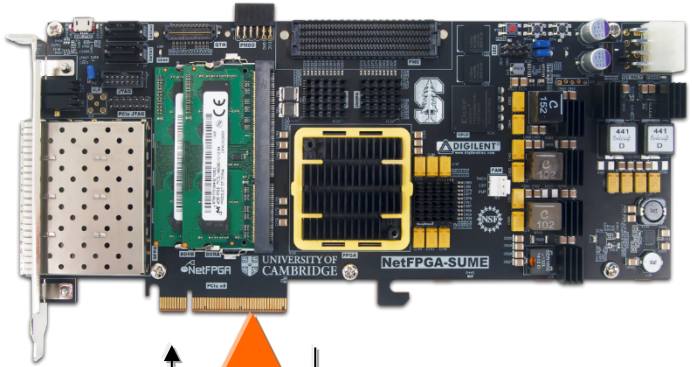
4. NetFPGA transfers packet via DMA

PCIe Bus

5. Interrupt signals completion of DMA

6. Driver passes packet to network stack

Host to NetFPGA Packet Transfer



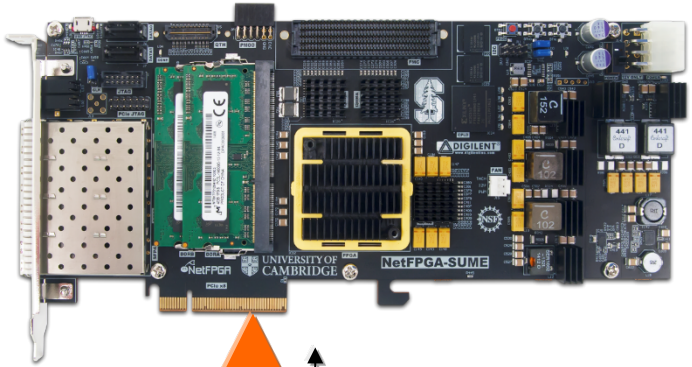
2. Driver sets up and initiates DMA transfer

PCIe Bus

3. Interrupt signals completion of DMA

1. Software sends packet via network sockets. Packet delivered to driver

NetFPGA Register Access



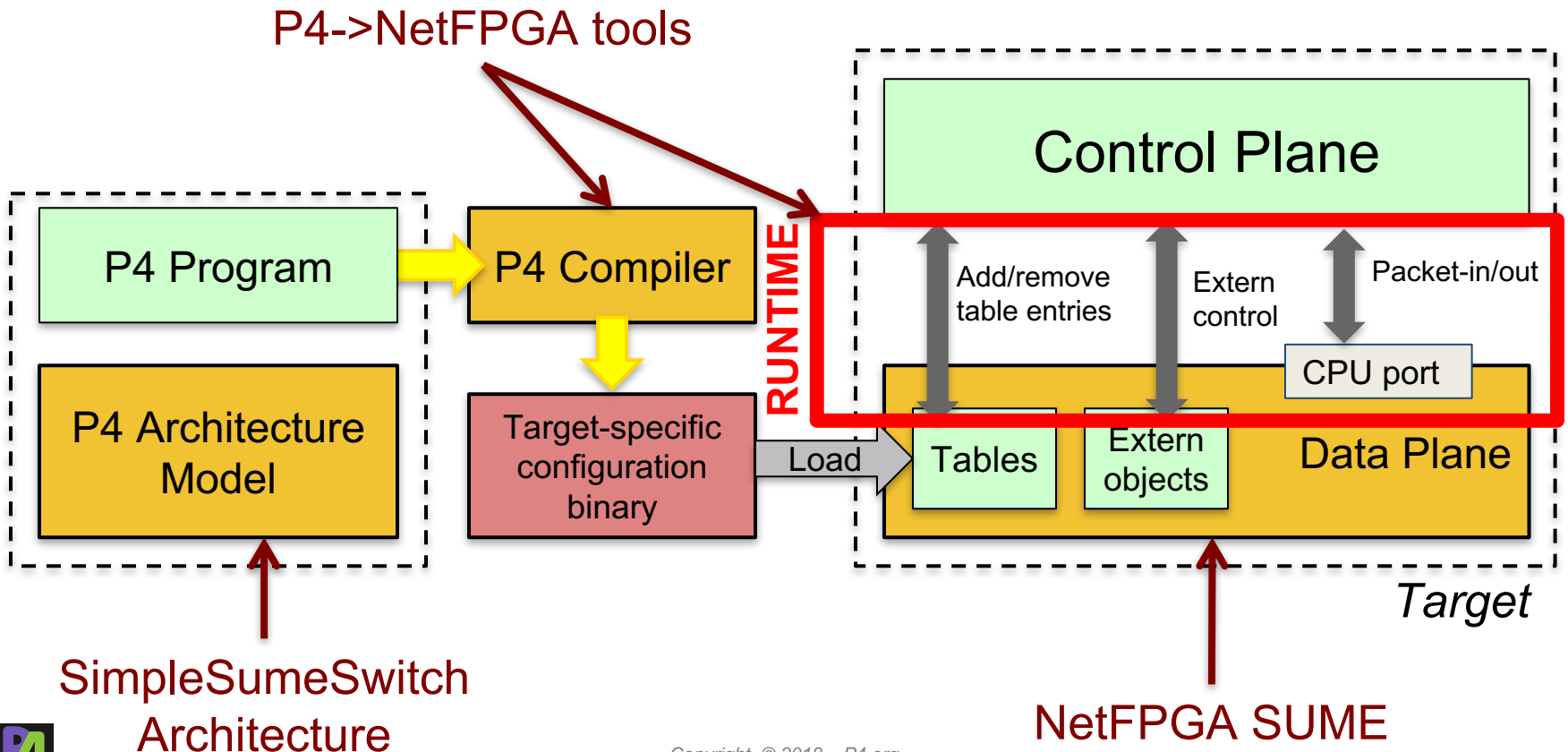
PCIe Bus

2. Driver performs PCIe memory read/write

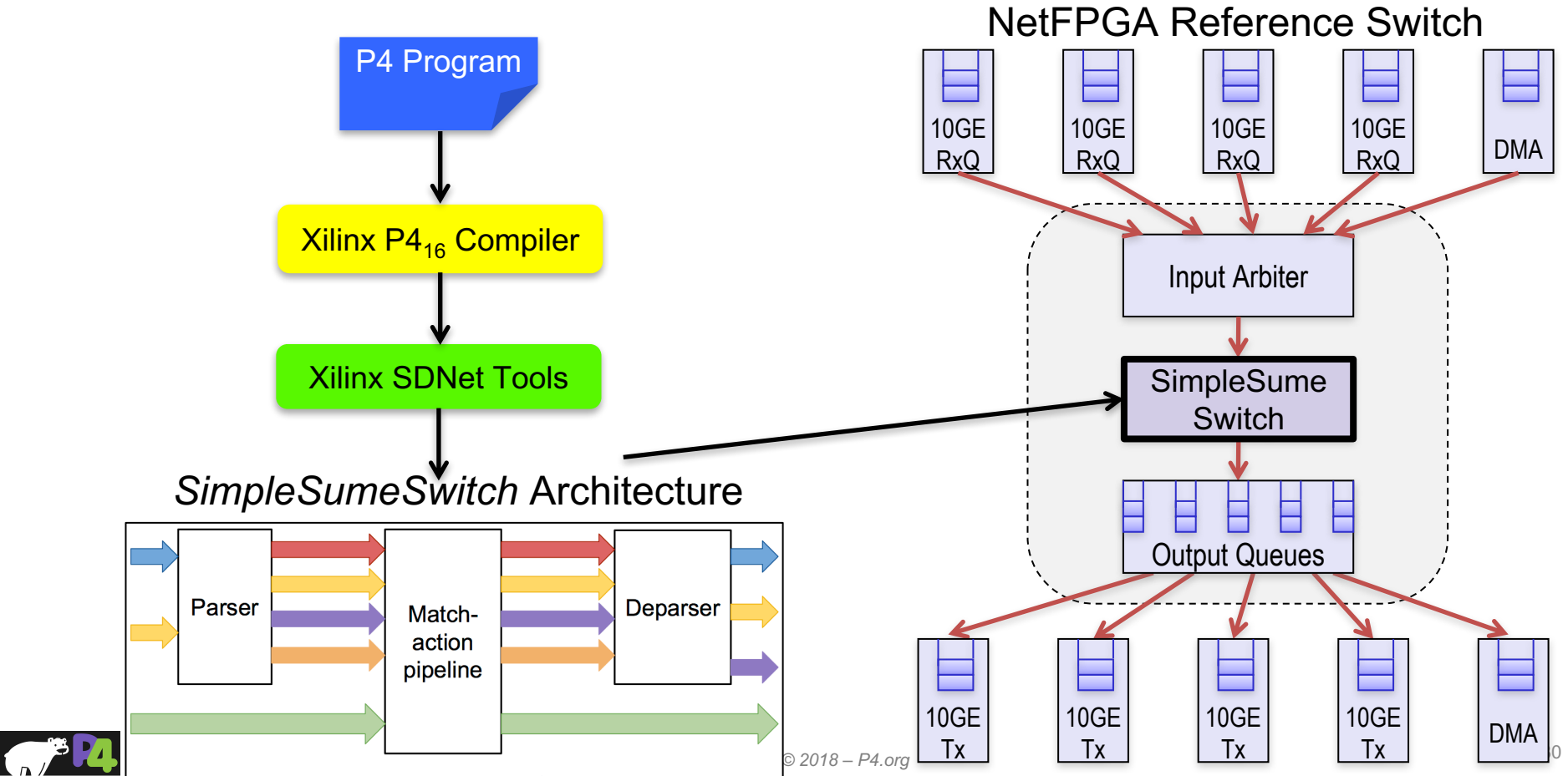
1. Software makes ioctl call on network socket. ioctl passed to driver

Overview

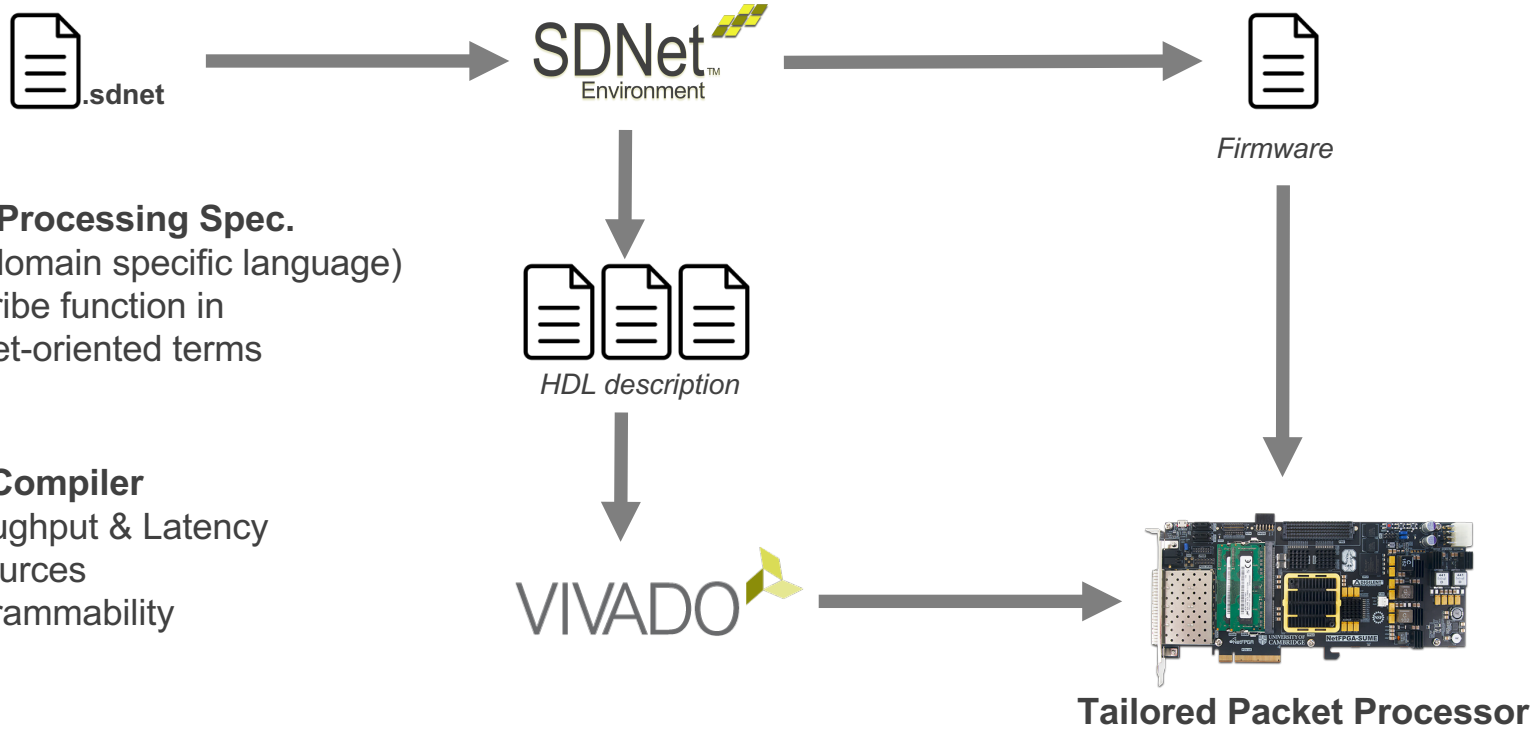
General Process for Programming a P4 Target



P4→NetFPGA Compilation Overview



Xilinx SDNet Design Flow & Use Model



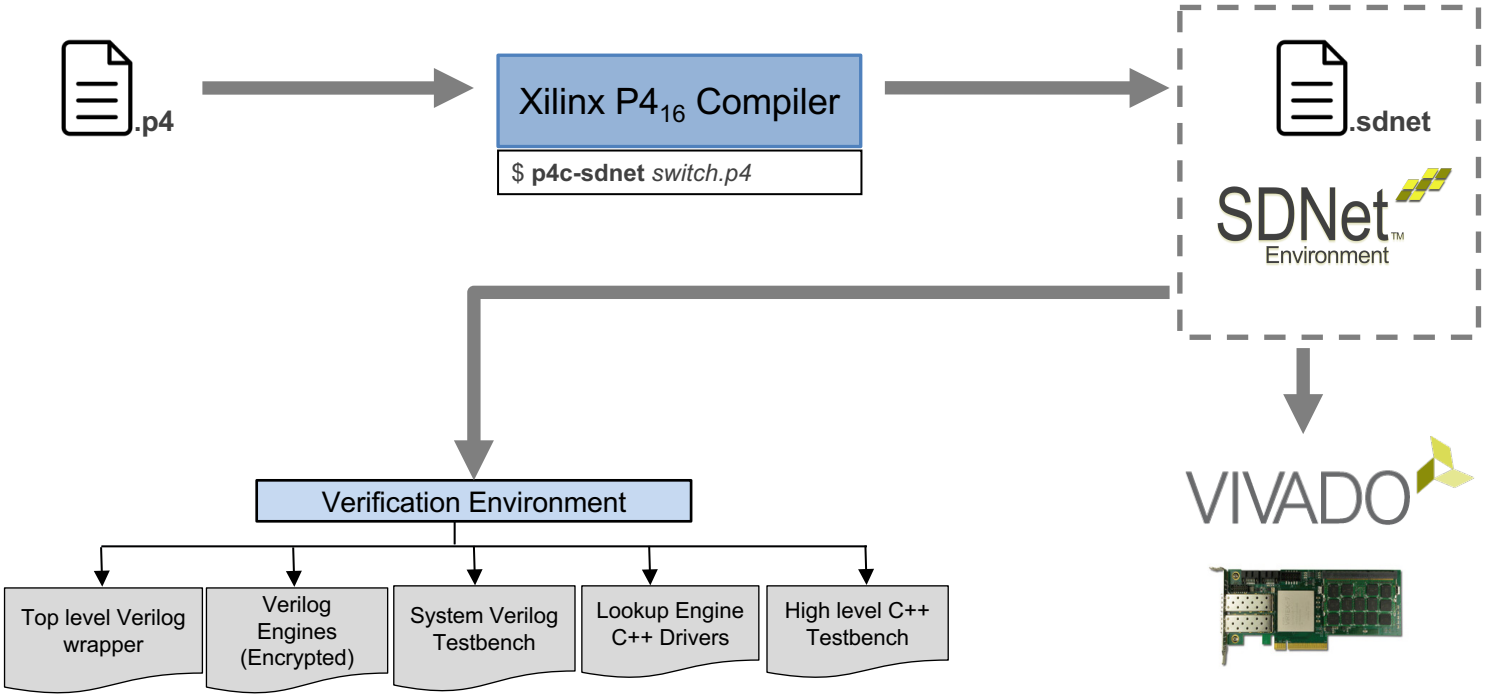
Packet Processing Spec.

- PX (domain specific language)
- describe function in packet-oriented terms

SDNet Compiler

- Throughput & Latency
- Resources
- Programmability

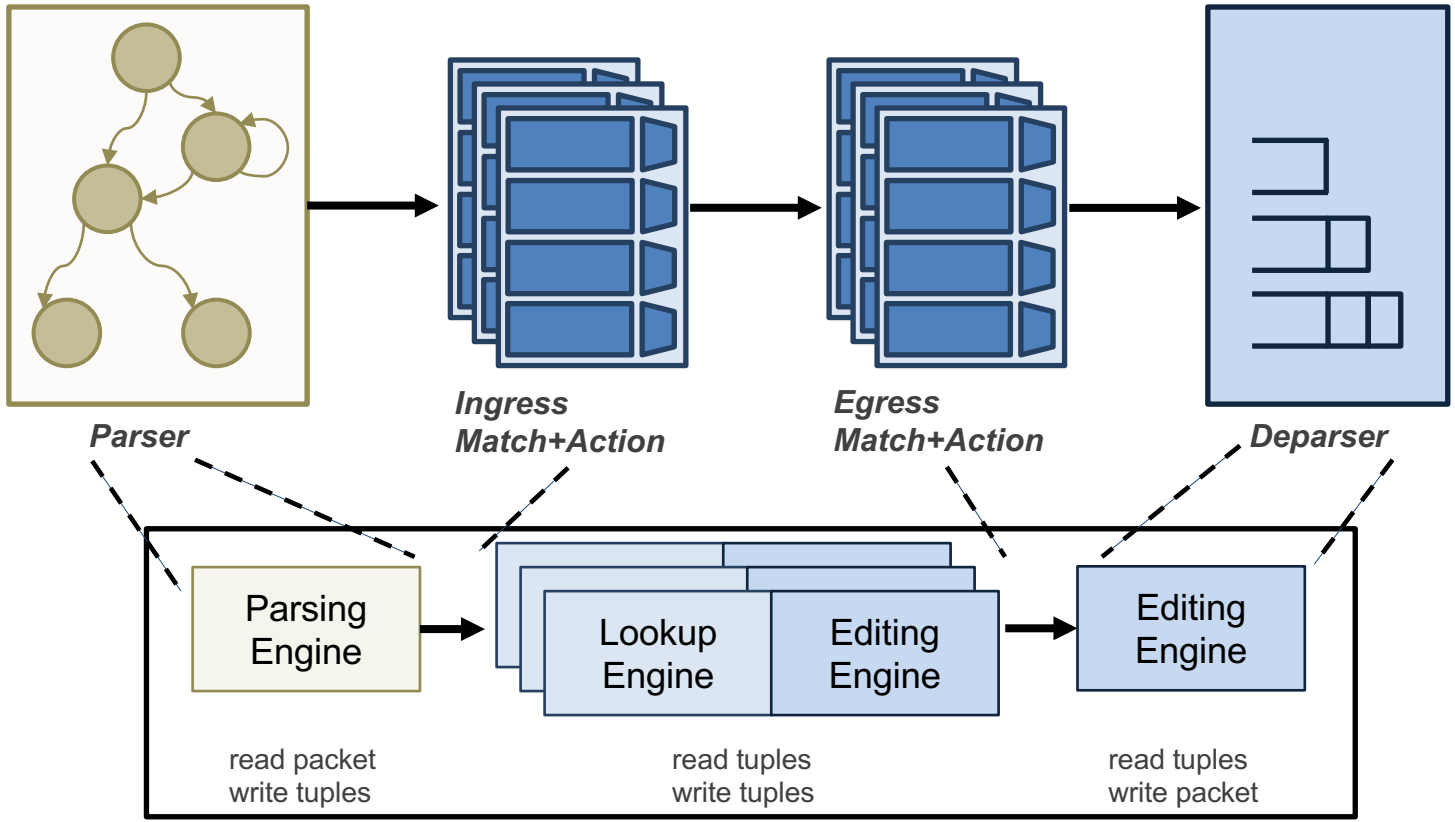
Xilinx P4 Design Flow & Use Model



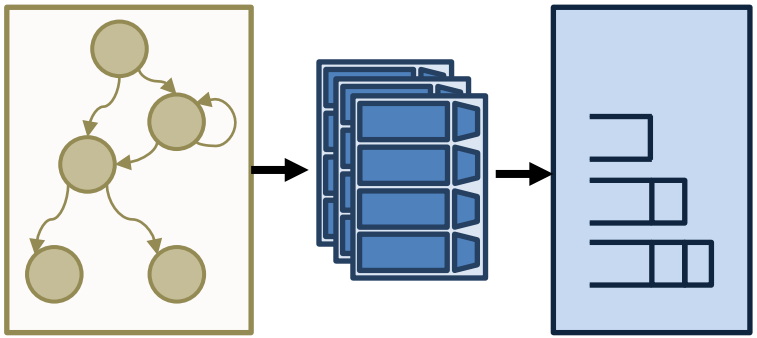
Considerations When Mapping to SDNet

- **Identifying parallelism within P4 parser and control blocks**
  - table lookups
  - actions
  - etc.
- **P4 packet processing model**
  - extract entire header from packet
  - updates apply directly to header
  - deparser re-inserts header back into packet
- **SDNet packet processing model**
  - stream packet through “engines”
  - modify header values in-line without removing and re-inserting

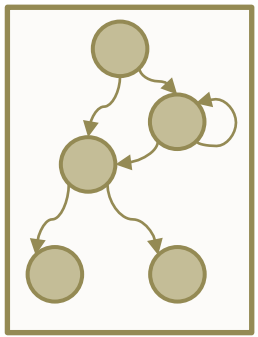
Mapping P4 Architectures to SDNet



Support for Multiple Architectures



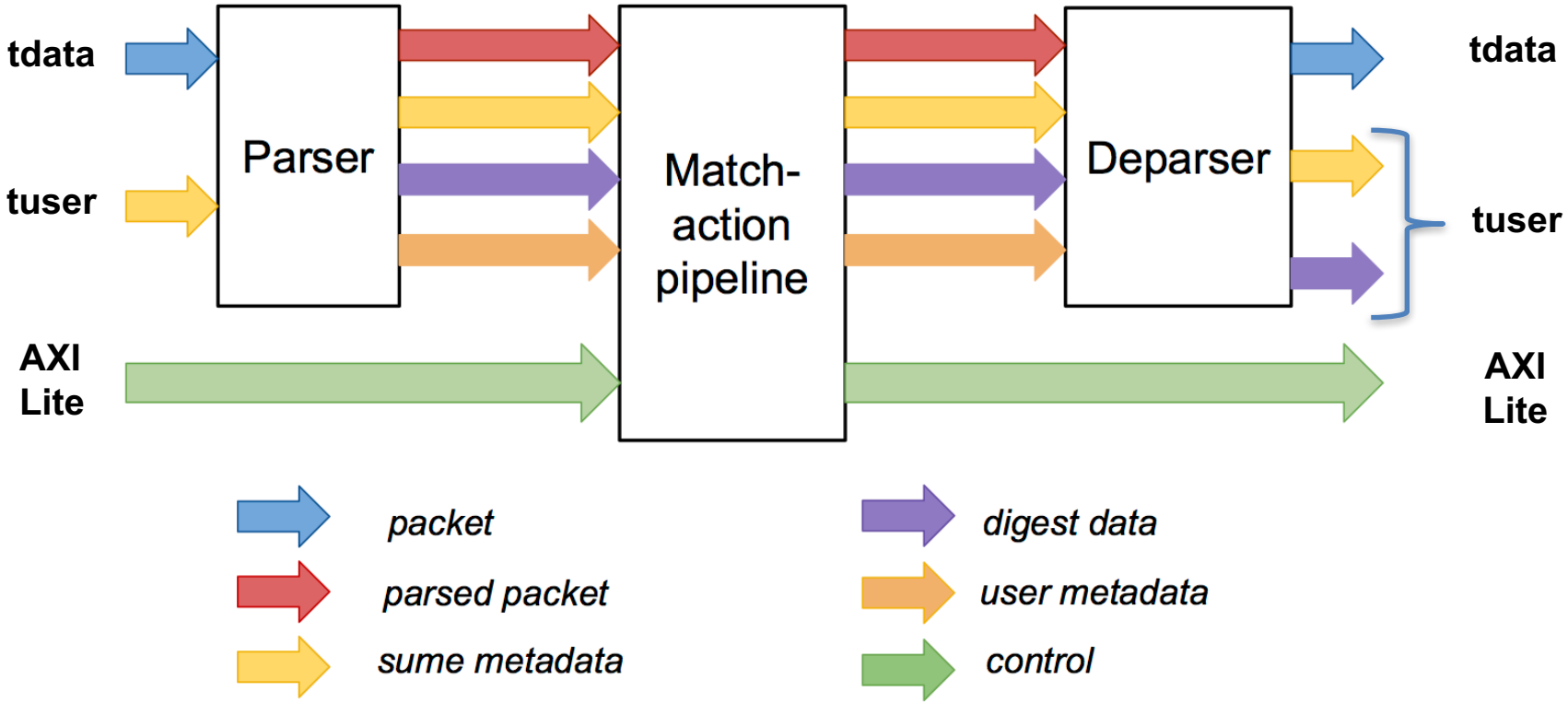
SimpleSumeSwitch



Only Parser

- Pull information from packet w/o updates

SimpleSumeSwitch Architecture Model for SUME Target



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

Standard 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> 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

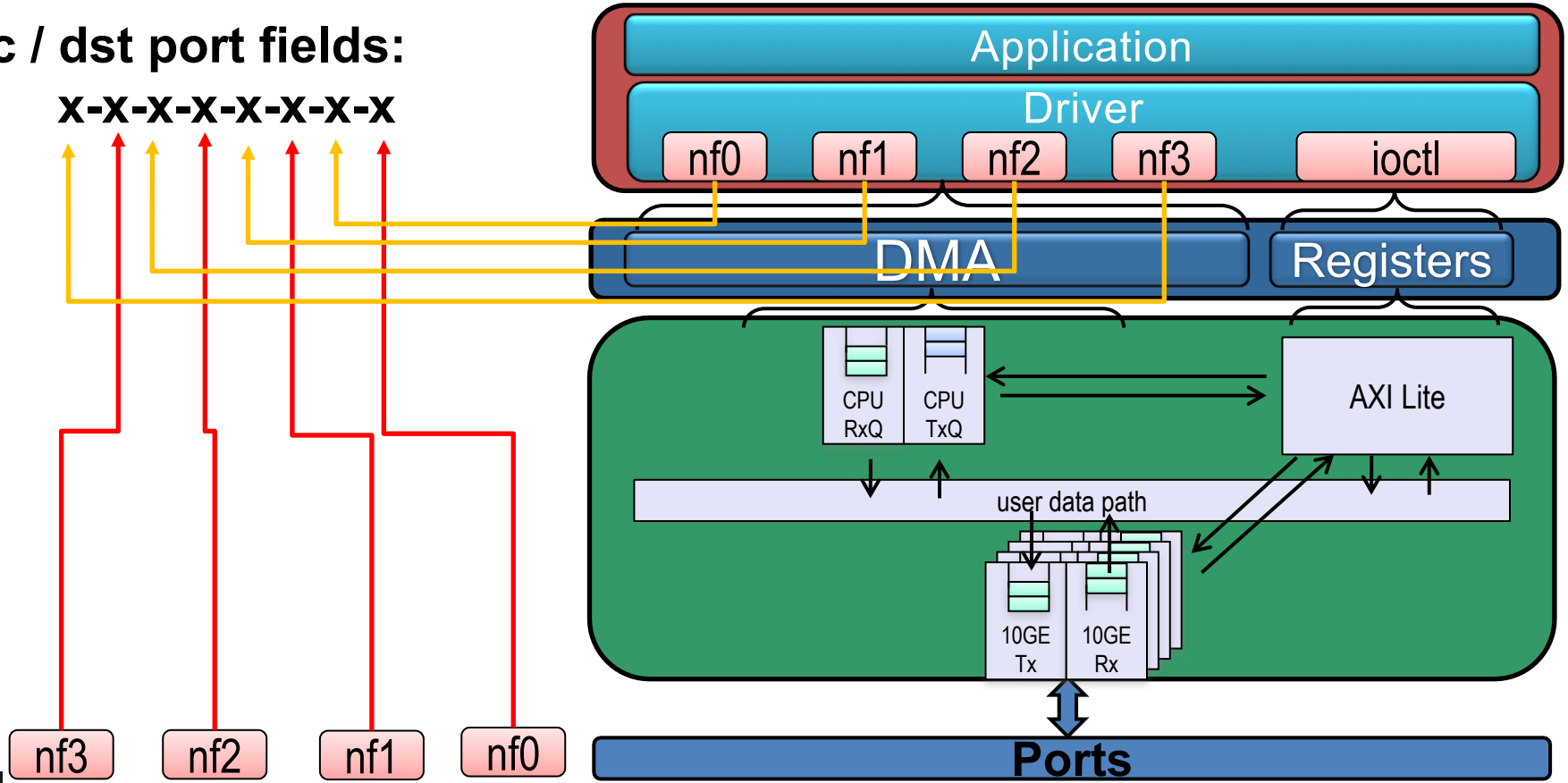
`src_port`/`dst_port` – one-hot encoded

`user_metadata`/`digest_data` – structs defined by the user

Interface Naming Conventions

src / dst port fields:

x-x-x-x-x-x-x-x



Overall P4 Program Structure

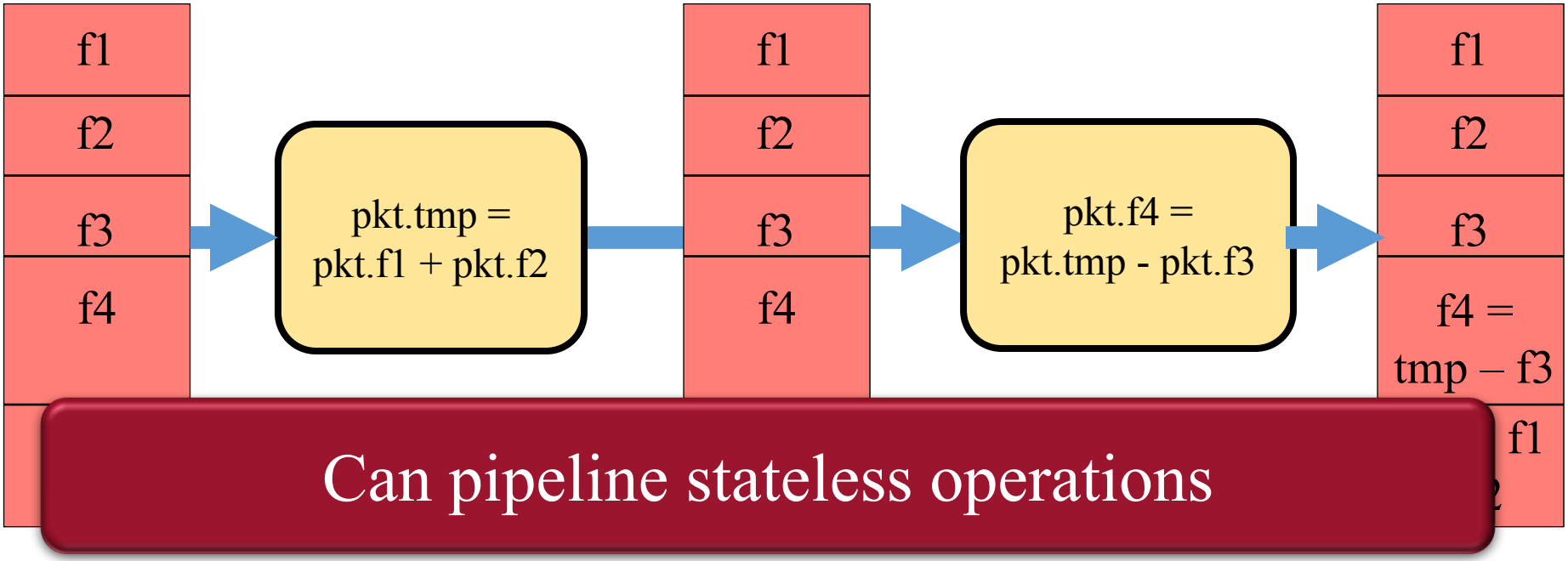
```
#include <core.p4>
#include <sume_switch.p4>

/******** 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;
```

P4→NetFPGA Extern Function library

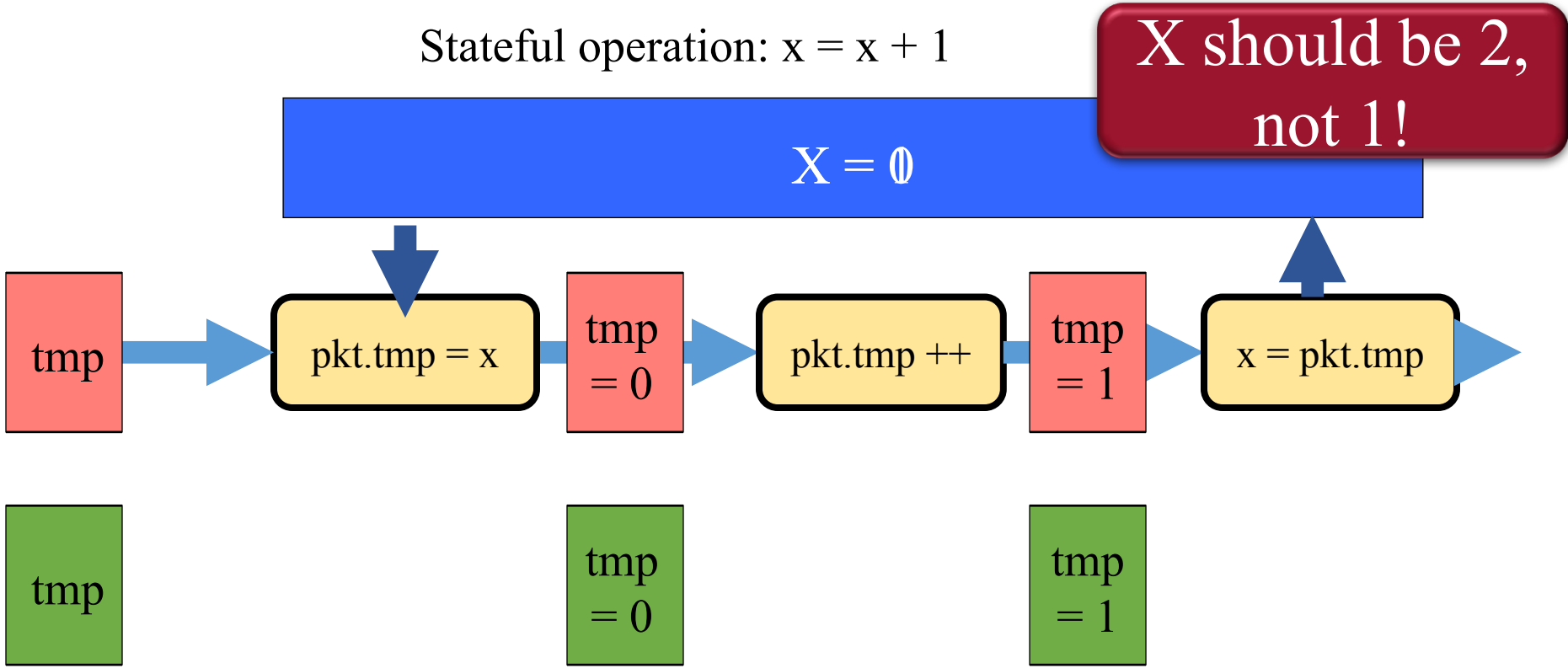
- **Implement platform specific functions**
  - Black box to P4 program
- **Implemented in HDL**
- **Stateless – reinitialized for each packet**
- **Stateful – keep state between packets**
- **Xilinx Annotations**
  - `@Xilinx_MaxLatency()` – maximum number of clock cycles an extern function needs to complete
  - `@Xilinx_ControlWidth()` – size in bits of the address space to allocate to an extern function

Stateless vs. stateful operations

Stateless operation: pkt.f4 = pkt.f1 + pkt.f2 – pkt.f3

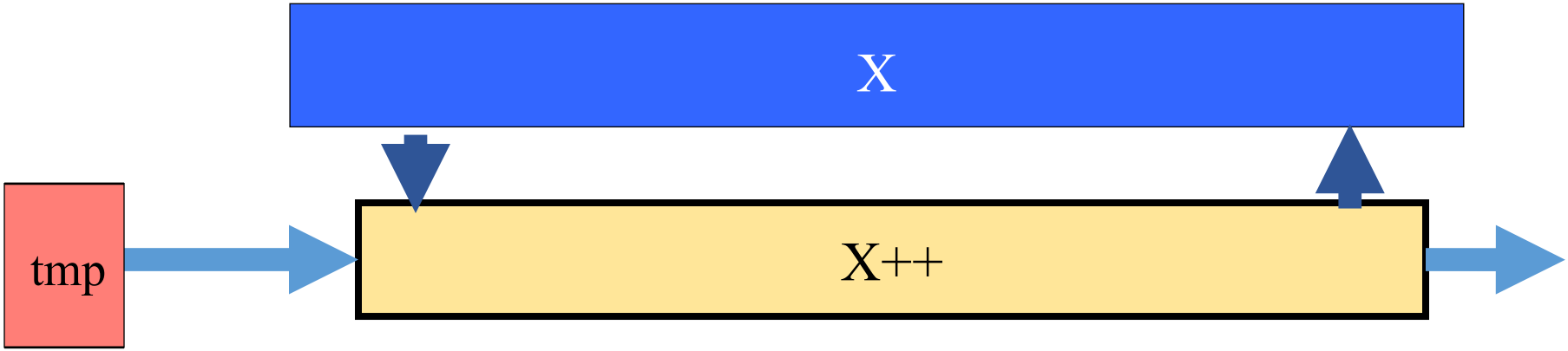


Stateless vs. stateful operations



Stateless vs. stateful operations

Stateful operation: x = x + 1



Cannot pipeline, need atomic operation in h/w

P4→NetFPGA Extern Function library

- HDL modules invoked from within P4 programs
- Stateful Atoms [1]

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)

- Add your own!

[1] Sivaraman, Anirudh, et al. "Packet transactions: High-level programming for line-rate switches." *Proceedings of the 2016 ACM SIGCOMM Conference*. ACM, 2016.

Adding Custom Externs

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

- No need to modify and existing code
- AXI Lite control interface module auto generated

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(64)
@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 reg
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!

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 externs in at run time

P4→NetFPGA Workflow

1. Write P4 program
2. Write externs
3. Write python gen_testdata.py script

All of your effort will go here

4. Compile to Verilog / generate API & CLI tools
5. Run simulations
6. Build bitstream
7. Check implementation results
8. Test the hardware

fail

pass

Debugging P4 Programs

- **SDNet HDL Simulation**
- **SDNet C++ simulation**
  - Verbose packet processing info
  - Output PCAP file
- **Full SUME HDL simulation**
- **Custom Python Model**

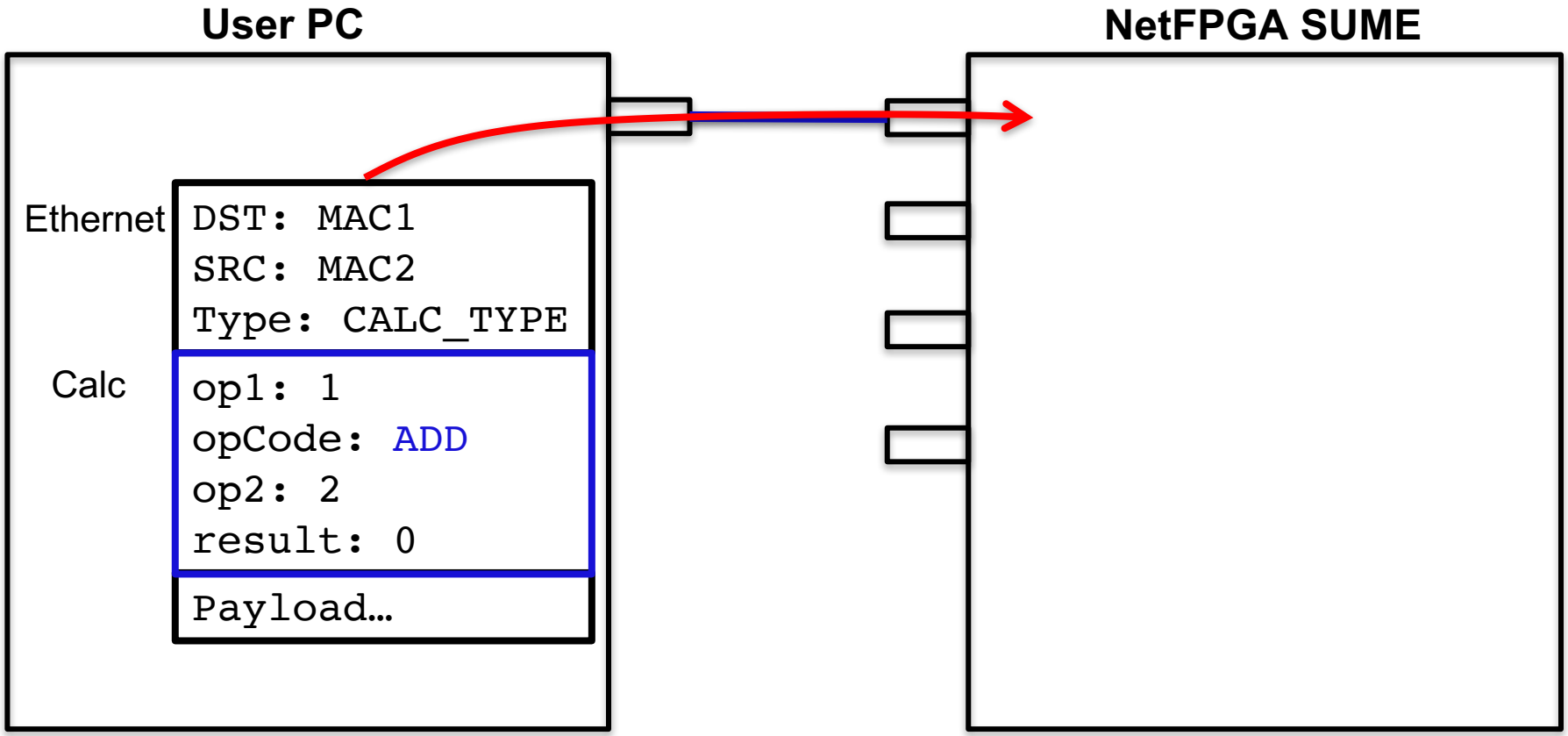
Assignment 1: Switch as a Calculator

Switch as a Calculator

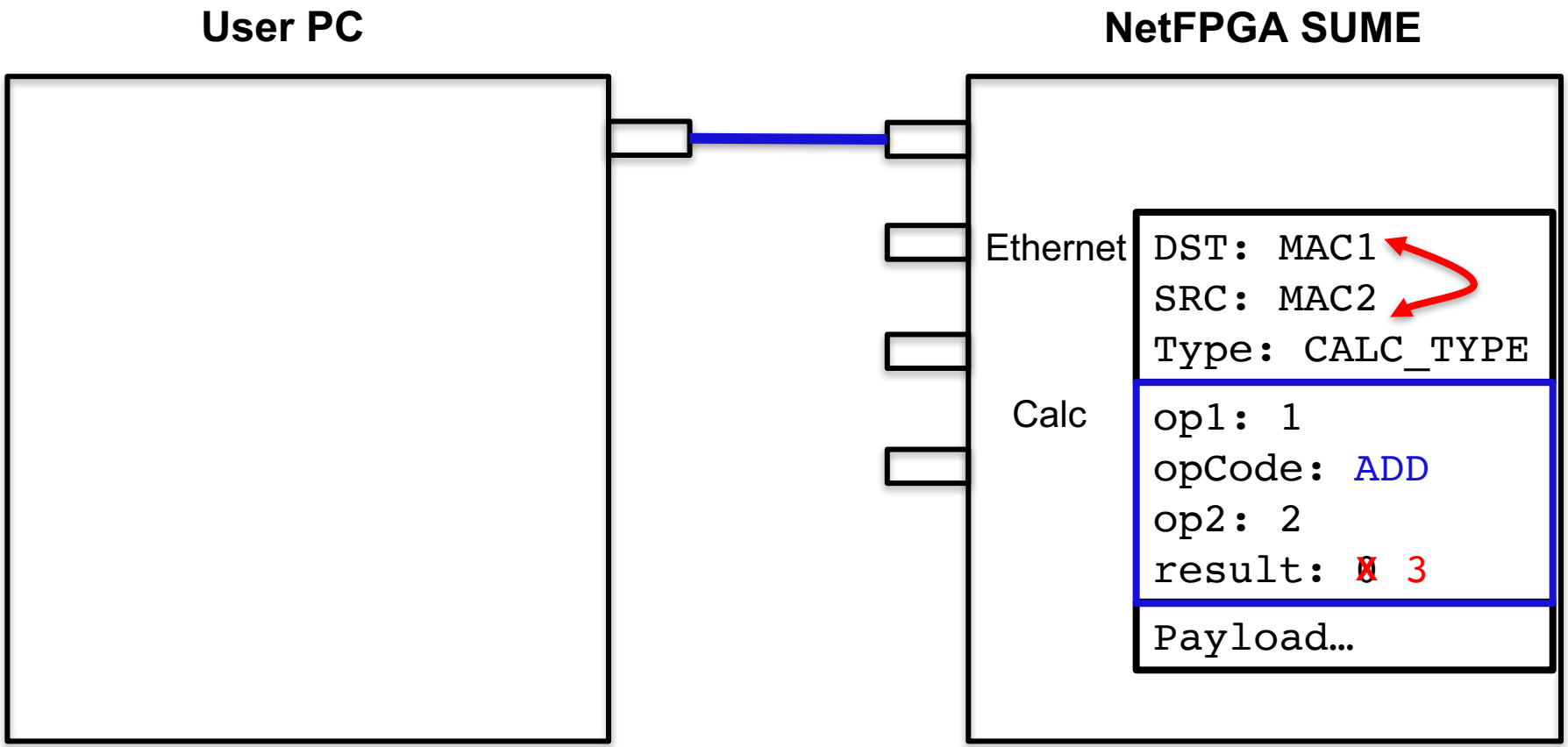
- **Supported Operations**
  - ADD – add two operands
  - SUBTRACT – subtract two operands
  - ADD_REG – add operand to current value in the register
  - SET_REG – overwrite the current value in the register
  - LOOKUP – Lookup the given key in the table

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

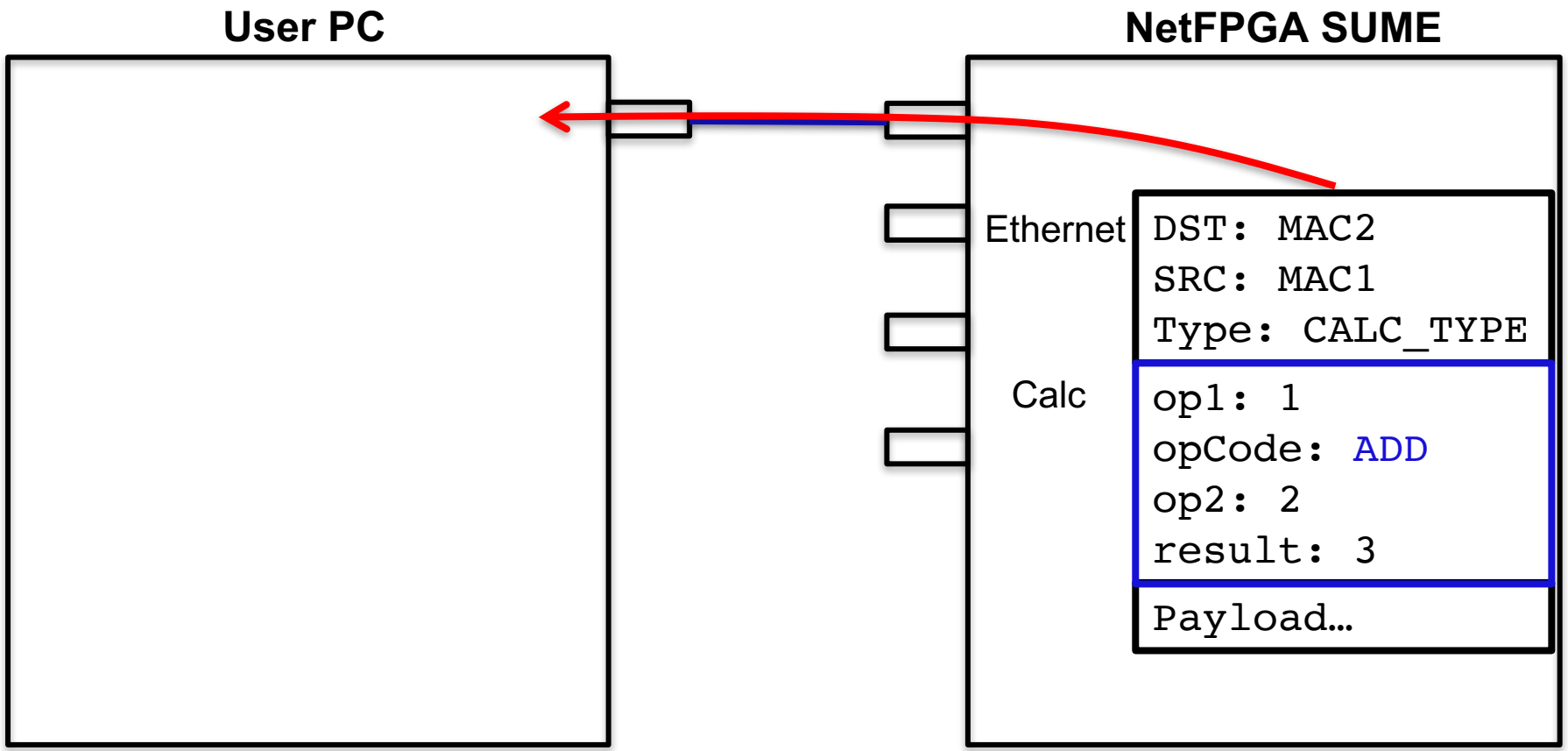
Switch as a Calculator



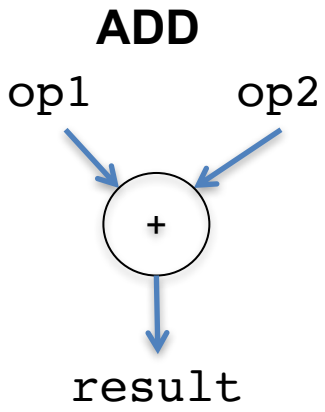
Switch as a Calculator

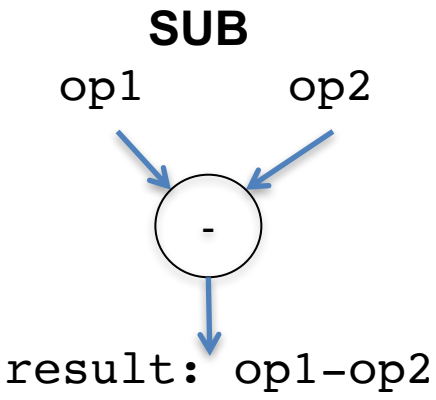


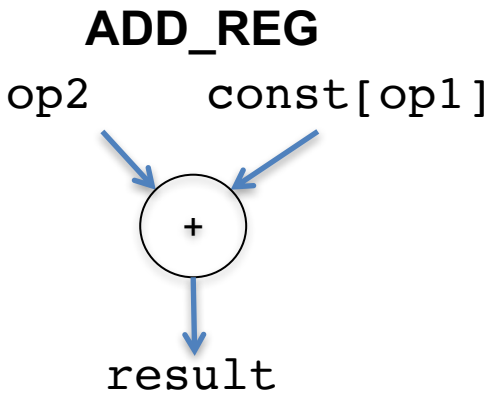
Switch as a Calculator

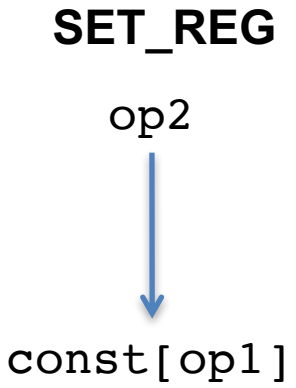


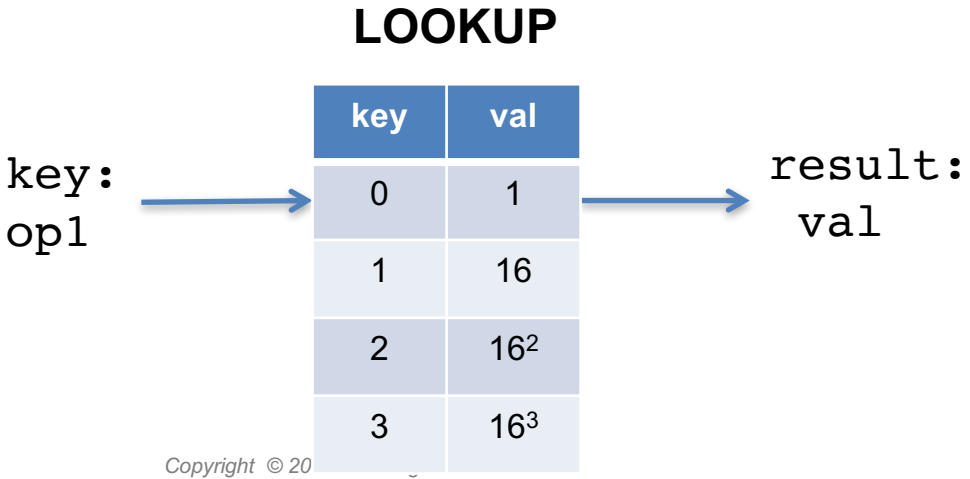
Switch Calc Operations











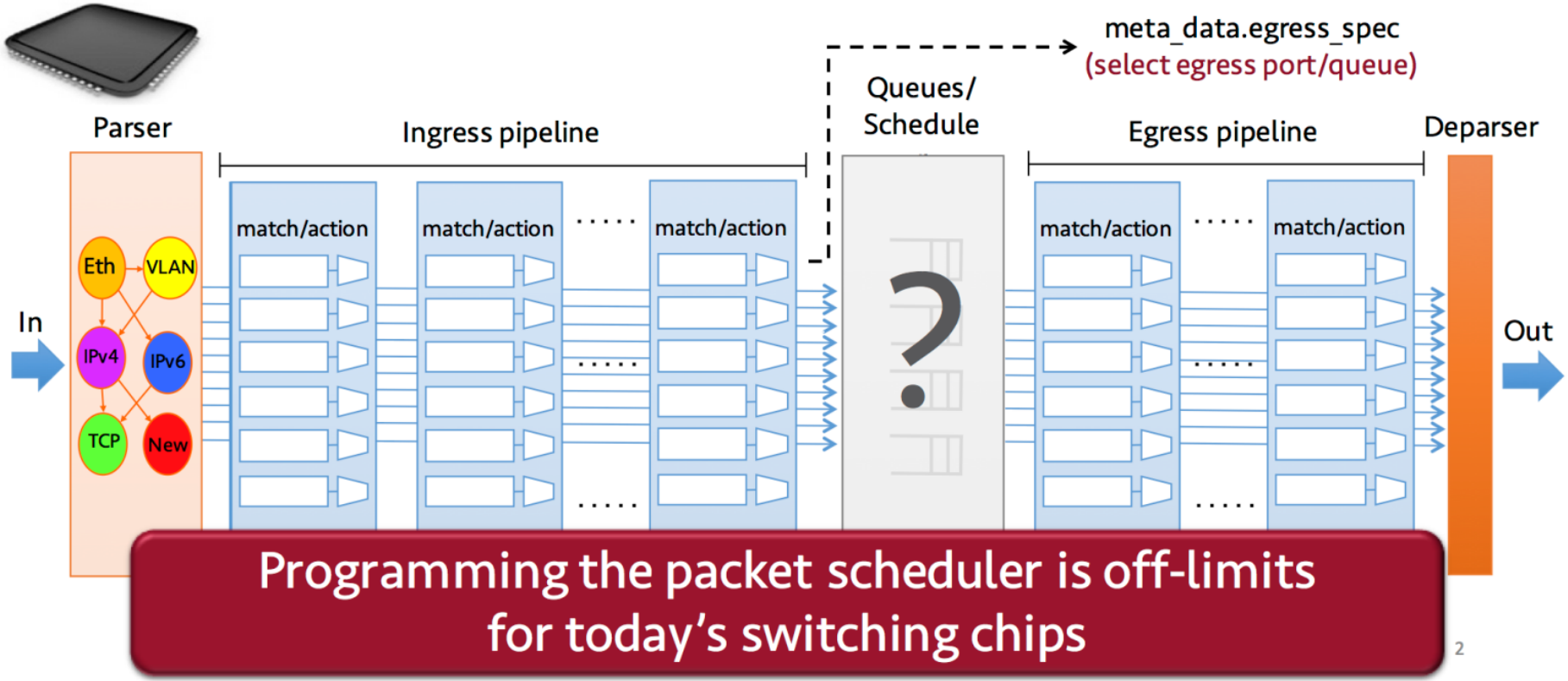
FIN

Research topics

Examples of ongoing P4 Research Topics

- **P4 Infrastructure**
  - **Programmable scheduling**
  - **Programmable target architectures**
  - PacketMod
- **Data-plane Programs**
  - **In-band network telemetry**
  - **Congestion control**
  - Load balancing
- **Networking-Offloading Applications**
  - **Aggregation for MapReduce applications**
  - Key-value caching
  - Consensus

Programmable Scheduling



Sivaraman, Anirudh, et al. "Programmable packet scheduling at line rate." *Proceedings of the 2016 ACM SIGCOMM Conference*. ACM, 2016.

Why scheduler is not programmable ... so far

- **Plenty of scheduling algorithms, but no consensus on right abstractions. Contrast to:**
  - Parse graphs for parsing
  - Match-Action tables for forwarding

- **Scheduler has tight timing requirements**
  - One decision every few ns

What does the Scheduler do?

Decides:

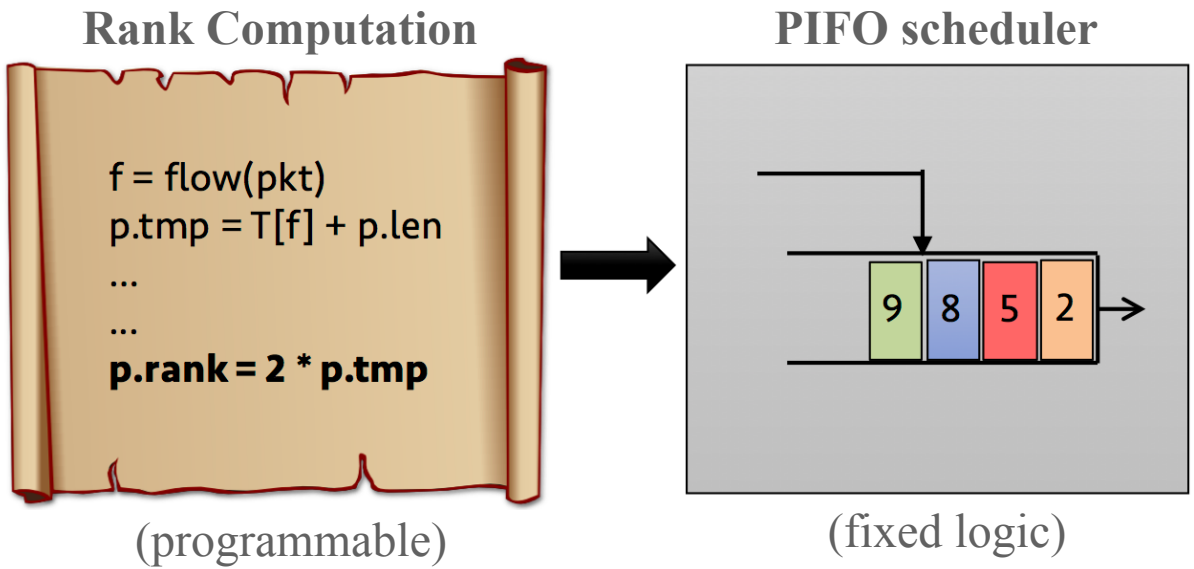
- **In what order are packets sent?**
  - Ex: FCFS, Priorities, WFQ
- **At what time are packets sent?**

Key observation:

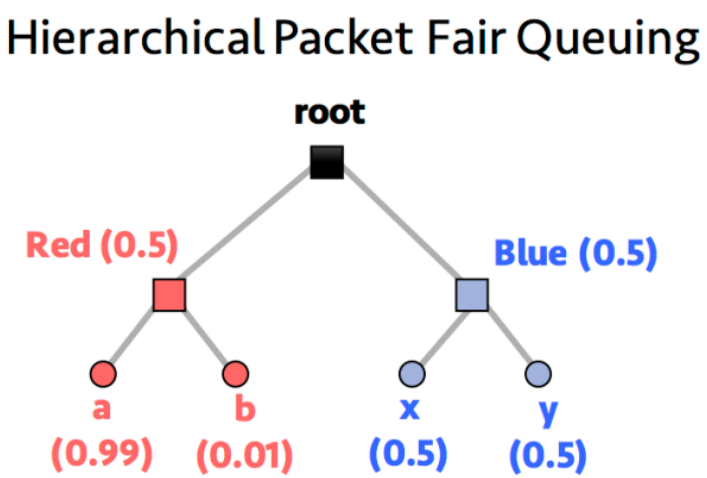
- **For many algorithms, the relative order in which packets are sent does not change with future arrivals**
  - i.e. scheduling order can be determined before enqueue

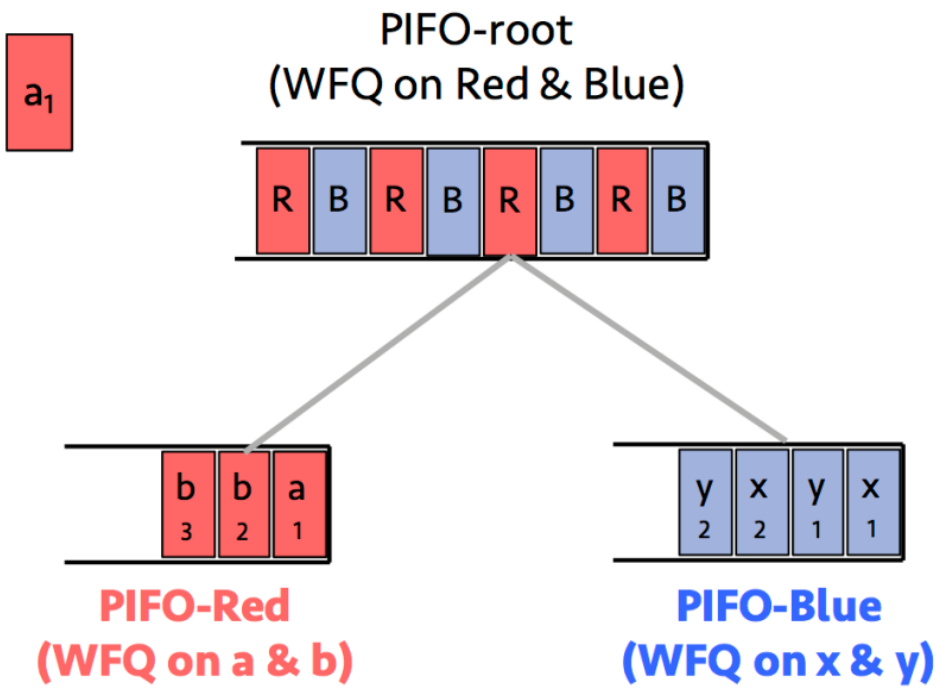
PIFO

- PIFO - proposed abstraction that can be used to implement many scheduling algorithms
- Packets are pushed into an arbitrary location based on computed rank



PIFO Tree





PIFO Remarks

- **Very limited scheduling in modern switching chips**
  - Deficit Round Robin, traffic shaping, strict priorities
- **Scheduling algorithms that can be implemented with PIFO**
  - Weighted Fair Queueing, Token Bucket Filtering, Hierarchical Packet Fair Queueing, Least-Slack Time-First, the Rate Controlled Service Disciplines, and fine-grained priority scheduling (e.g., Shortest Job First)
- **PIFO cannot implement algorithms that require**
  - Changing the scheduling order of all packets of a flow
  - Output rate limiting
- **PIFO implementation feasibility?**

Programmable Target Architectures

Observations:

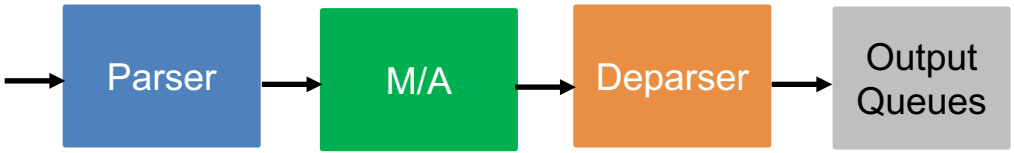
- Current P4 expectation: target architectures are *fixed,* specified in English
- FPGAs can support many different architectures

Idea:

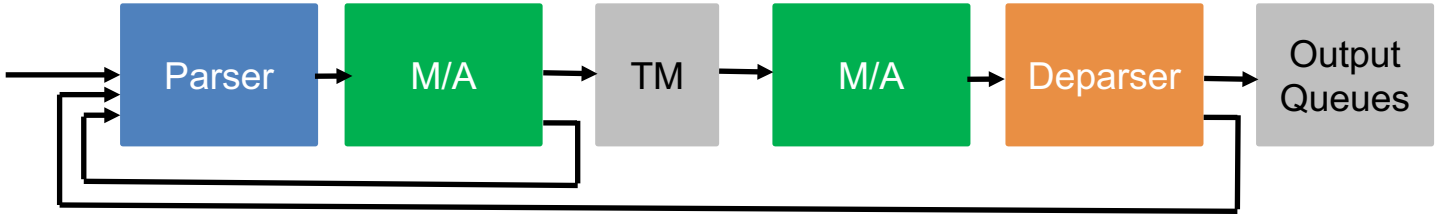
- Extend P4 to allow description of target architectures
  - More precise definition than English description
- Generate implementation on FPGA
- Easily integrate custom modules
- Explore performance tradeoffs of different architectures

Many Possible Architectures…

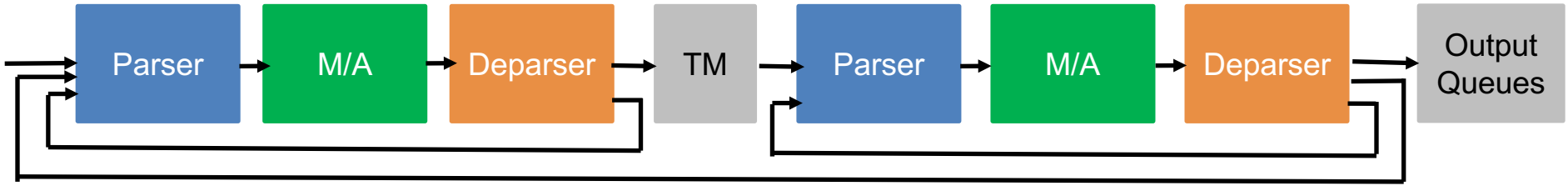
SimpleSumeSwitch



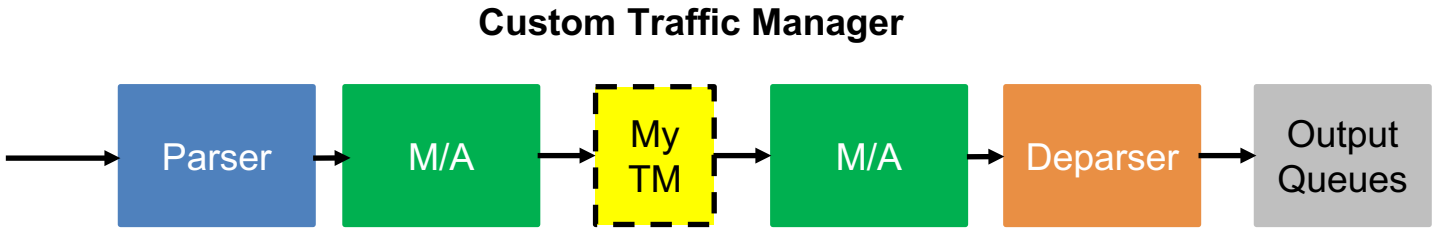
V1 Model

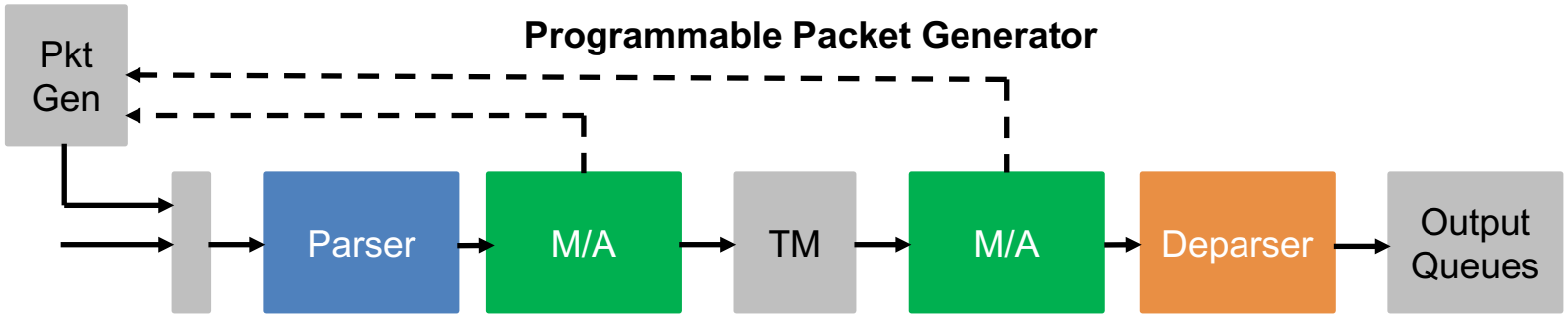


Portable Switch Architecture



Many Possible Architectures…





Programmable Target Architectures

```
package SimpleSumeSwitch<H, M, D>(
                  Parser<H, M, D> TopParser,
                  Pipe<H, M, D> TopPipe,
                  Deparser<H, M, D> TopDeparser) {

 // Top level I/O
 packet_in instream;
 inout sume_metadata_t sume_metadata;
 out D digest_data;
 packet_out outstream;

 // Connectivity of the architecture
 connections {
   // TopParser input connections
   TopParser.b                  = instream;
   TopParser.sume_metadata      = sume_metadata;

   // TopPipe <-- TopParser
   TopPipe.p                = TopParser.p;
   TopPipe.user_metadata    = TopParser.user_metadata;
   TopPipe.digest_data      = TopParser.digest_data;
   TopPipe.sume_metadata    = TopParser.sume_metadata;
```

```
   // TopDeparser <-- TopPipe
   TopDeparser.p                = TopPipe.p;
   TopDeparser.user_metadata    = TopPipe.user_metadata;
   TopDeparser.digest_data      = TopPipe.digest_data;
   TopDeparser.sume_metadata    = TopPipe.sume_metadata;

   // TopDeparser output connections
   digest_data      = TopDeparser.digest_data;
   sume_metadata    = TopDeparser.sume_metadata;
   outstream        = TopDeparser.b;
 }
}
```

Workflow

- **Two Actors: (1) Target Architecture Designer, (2) P4 Programmer**

Provides:
- $P4_+$ architecture declaration

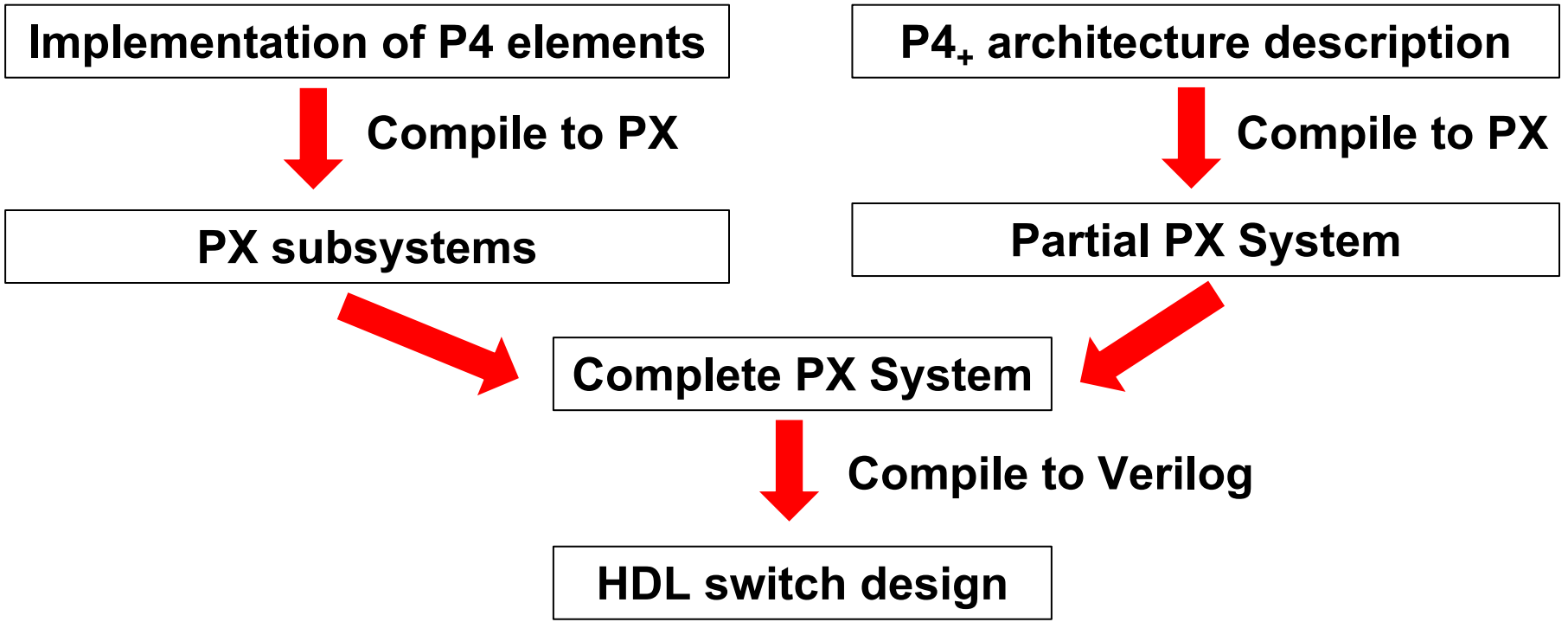
Implements:
- non-P4 elements
- externs

in target architecture

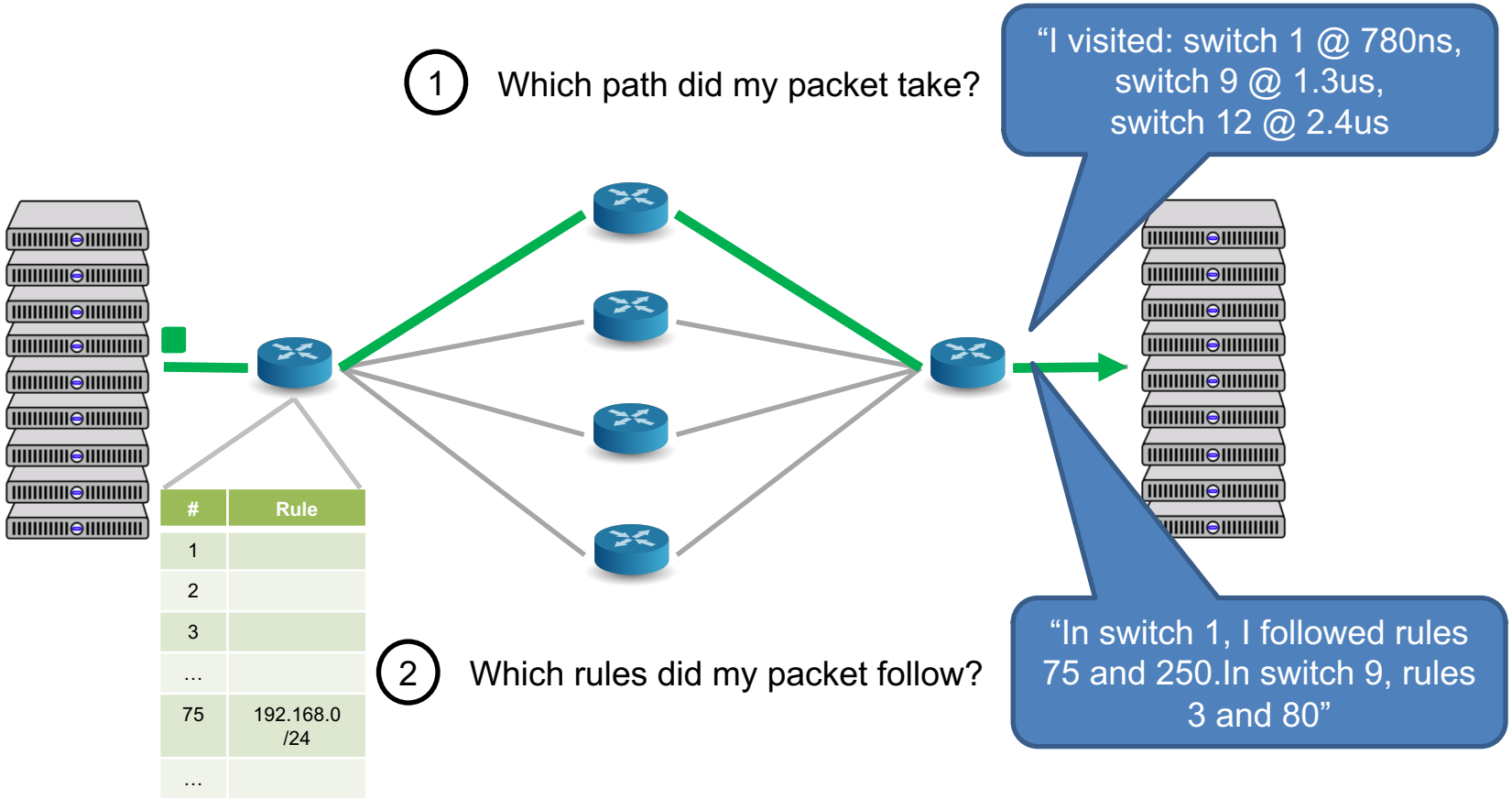
- **Someone who is more familiar with FPGA development**

Workflow

- Two Actors: (1) Target Architecture Designer, (2) P4 Programmer



In-band Network Telemetry (INT)

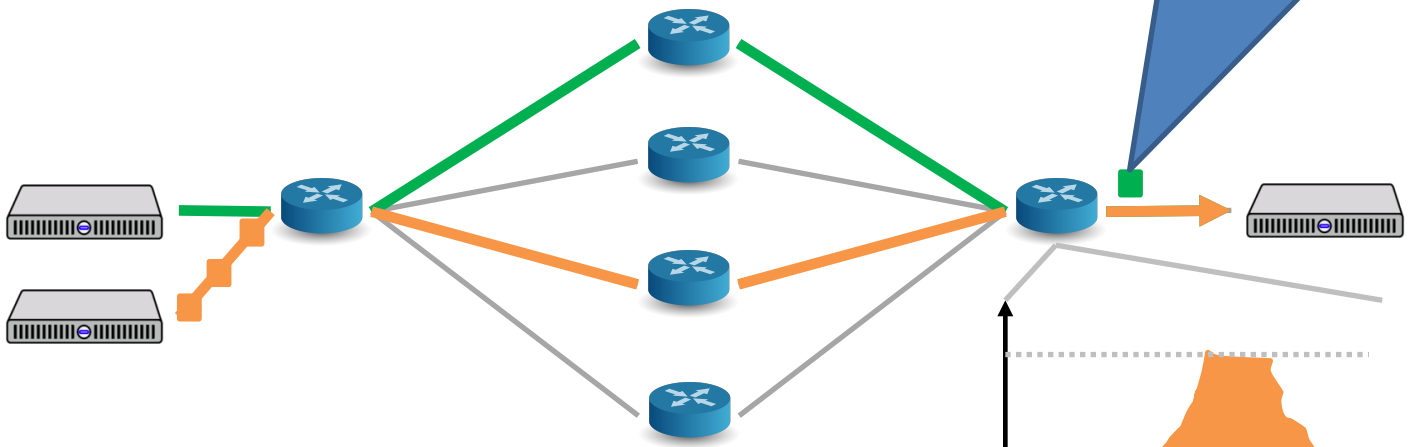


INT Slides courtesy of Nick McKeown

In-band Network Telemetry (INT)

3 How long did my packet queue at each s

"Delay: 100ns, 200ns, 19740ns"



Queue

Time

4 Who did my packet share the queue with?

INT Slides courtesy of Nick McKeown

In-band Network Telemetry (INT)

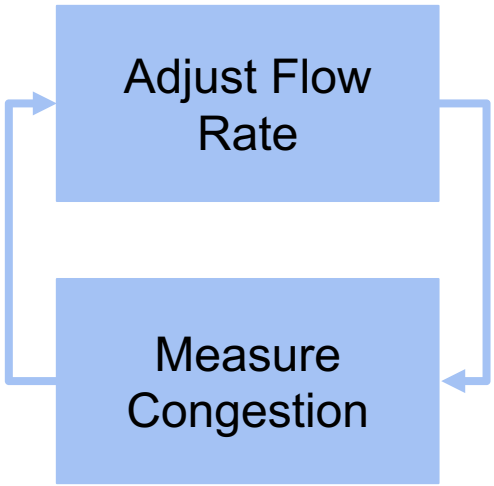
1. Which path did my packet take?
2. Which rules did my packet follow?
3. How long did my packet queue at each switch?
4. Who did my packet share the queue with?

No need to add a single additional packet!

INT Slides courtesy of Nick McKeown

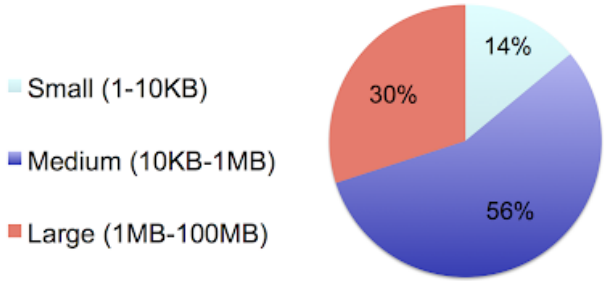
Congestion Control

Reactive Congestion Control



- No use of explicit information about traffic matrix
- Can only react and move in right direction
- Reactive techniques are slow to converge (10s-100s of RTTs)

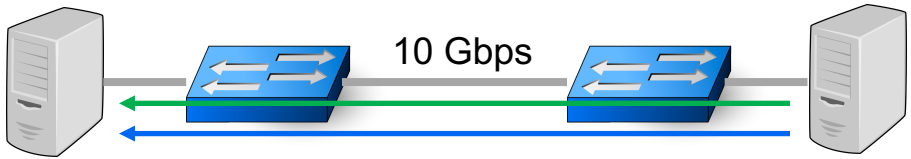
Fraction of Total Flows in Bing Workload

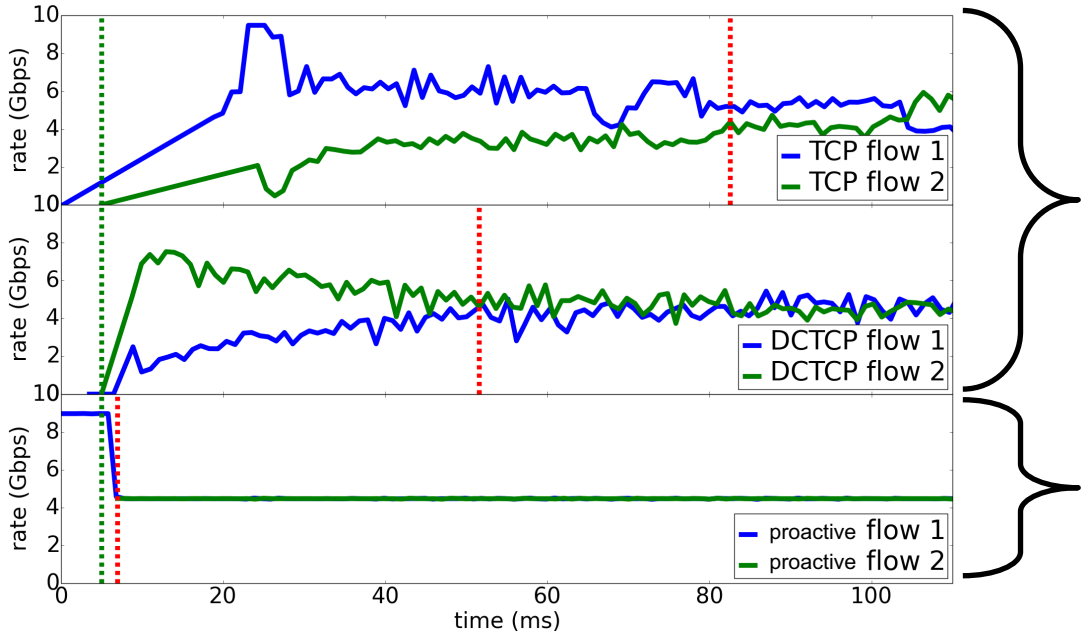


	Typical Flow Completion Times
10 Gb/s	70-80 RTTs
40 Gb/s	17-20 RTTs
100 Gb/s	7-8 RTTs

- Typical flows will finish in just a few RTTs as we move towards higher link speeds

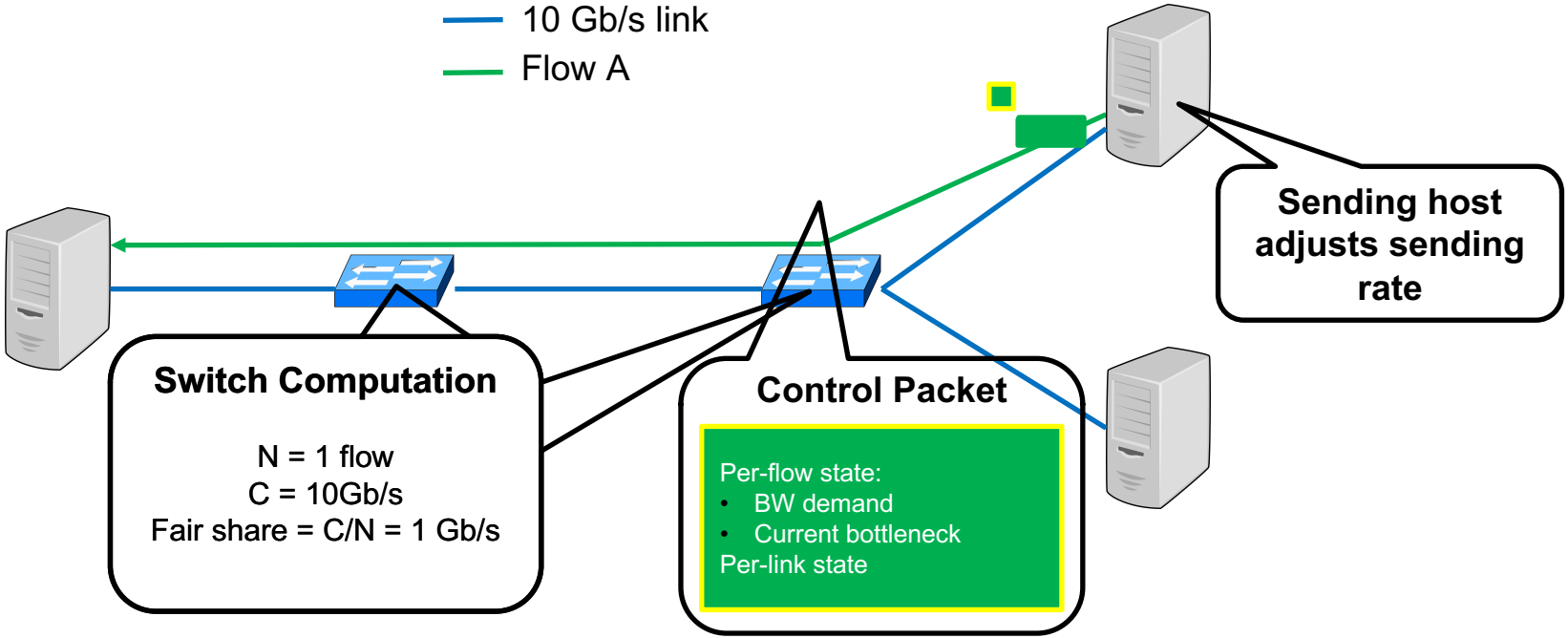
Proactive Congestion Control



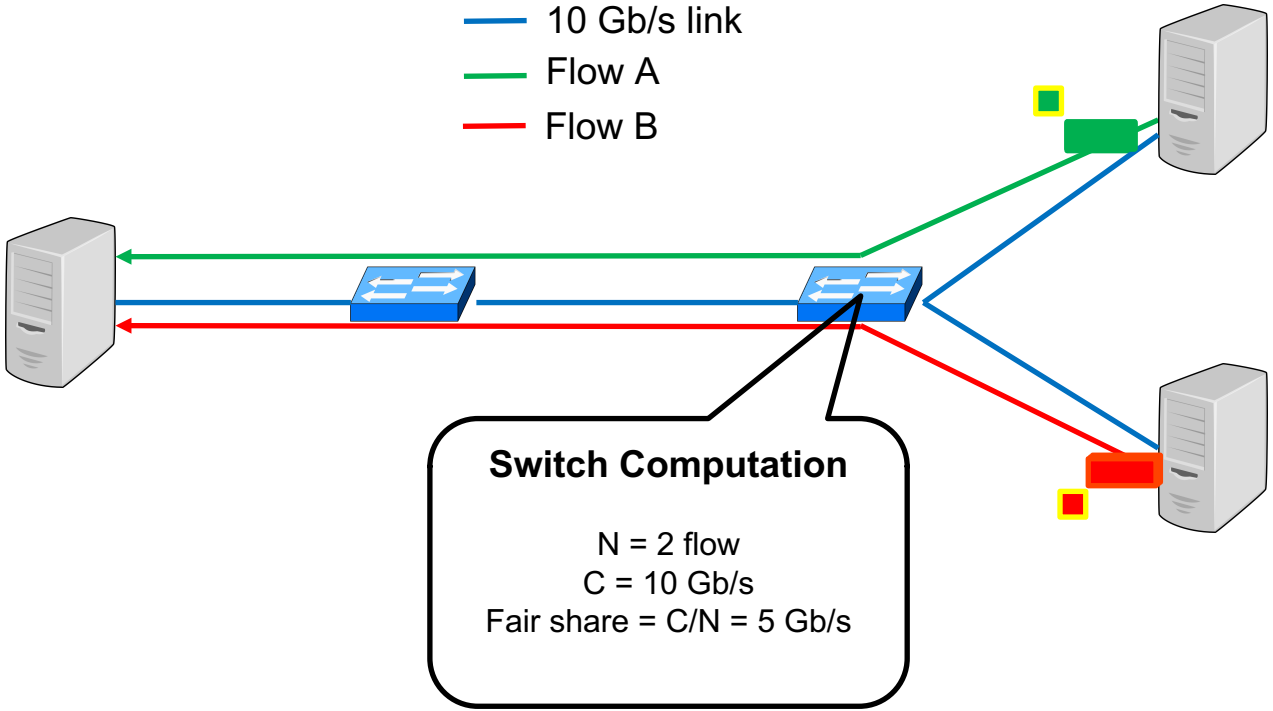


- **Proactive techniques converge much more quickly than reactive**
- **Faster convergence times lead to lower flow completion times**

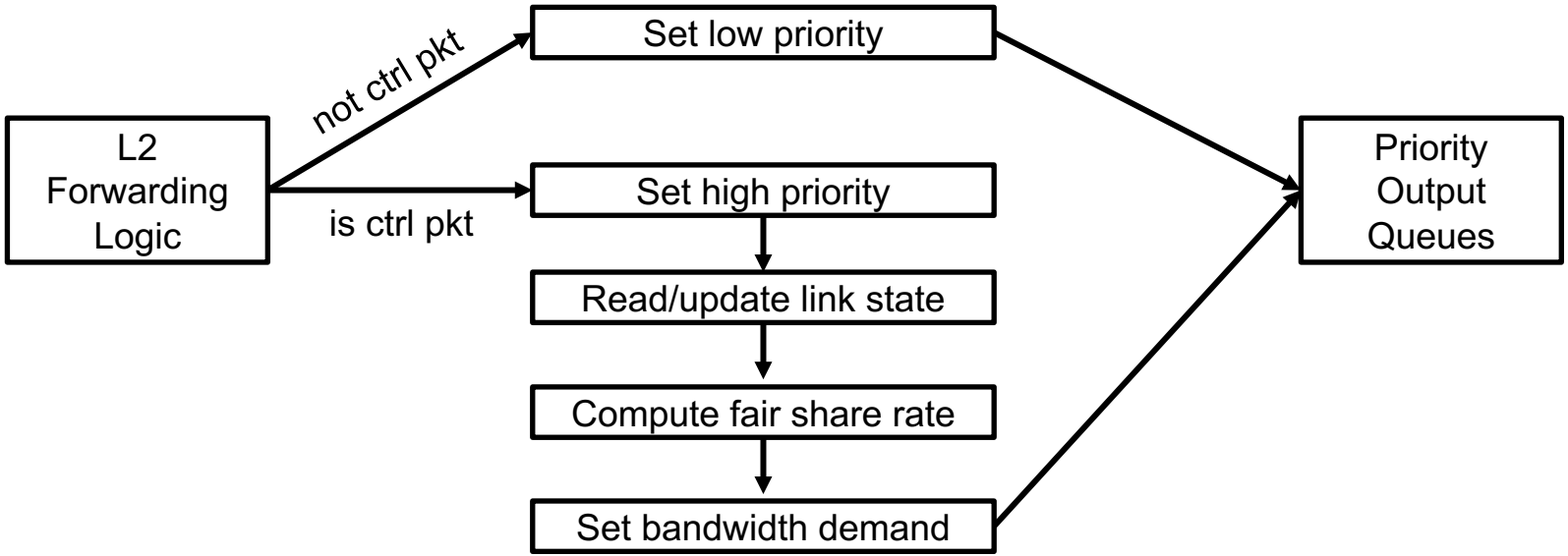
An example proactive scheme



An example proactive scheme



Proactive Algorithm in P4



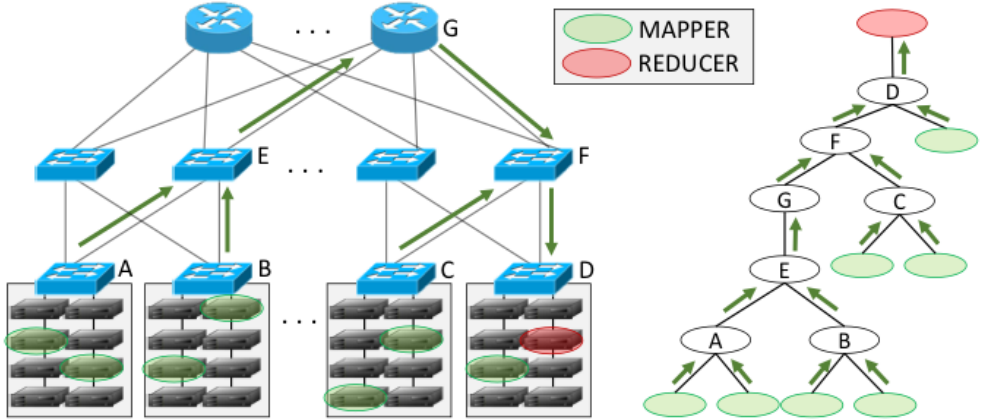
In-Network Computation

- **Programmable data plane hardware → opportunity to reconsider division of computation**
- ***What kinds of computation should be delegated to network?***
- **Network computations are constrained:**
  - *Limited memory size* (10's of MB of SRAM)
  - *Limited set of actions* (simple arithmetic, hashing, table lookups)
  - *Few operations per packet* (10's of ns to process each packet)
- **Goals:**
  - Reduce: application runtime, load on servers, network congestion
  - Increase: application scalability

Sapio, Amedeo, et al. "In-Network Computation is a Dumb Idea Whose Time Has Come." *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*. ACM, 2017.

In-Network Aggregation

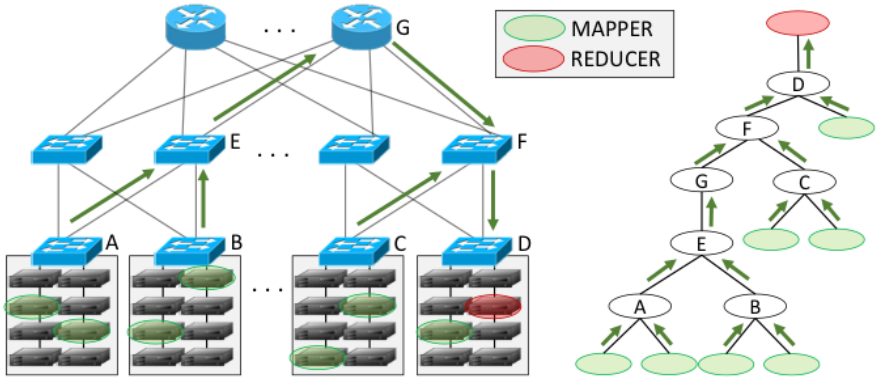
- **Aggregate data at intermediate network nodes to reduce network traffic**
- **Simple arithmetic operations at switches**
- **Widely applicable to many distributed applications**
  - Machine learning training
  - Graph analytics
  - MapReduce applications



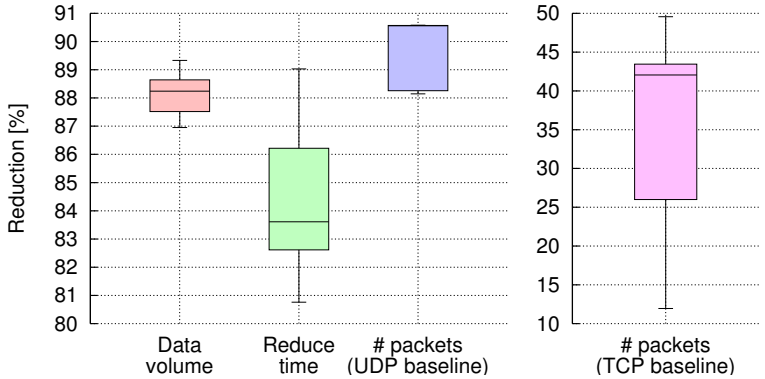
In-Network Aggregation

- **Network controller is informed of MapReduce job**
  - Configures switches in aggregation tree to perform aggregation
- **Significant network traffic reduction → reduced run time**
- **How to make robust to loss? Encryption?**

Aggregation Tree



Reduction Results



The P4 Language Consortium

- http://p4.org
- Consortium of academic and industry members
- Open source, evolving, domain-specific language
- Permissive Apache license, code on GitHub today
- Membership is free: contributions are welcome
- Independent, set up as a California nonprofit