P4FPGA Expedition

Han Wang

Ki Suh Lee, Vishal Shrivastav,

Hakim Weatherspoon, Nate Foster, Robert Soule[1]

Cornell University [1]Università della Svizzera italiana

Networking and Programming Language Workshop (NetPL'16)

P4 is awesome

- Exciting new applications
  - Telemetry : network measurement at network speed
  - NetPaxos : distributed consensus at network speed
  - and more ..
- We can deploy application across targets
  - Software : flexible and performance limitation
  - ASIC : Fast but could be expensive

Flexible, efficient and at reasonable cost?

P4 is awesome

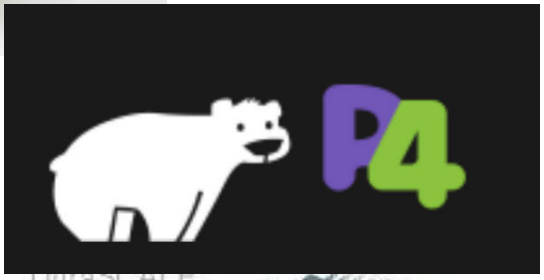
- Exciting new applications
  - Telemetry : network measurement at network speed
  - NetPaxos : distributed consensus at network speed
  - and more ..
- We can deploy application across targets
  - Software : flexible and performance limitation
  - ASIC : Fast but could be expensive

NetFPGA for P4 Era?

FPGAs are awesome

- Low cost
  - $2000 for NetFPGA SUME at academic price
- Good performance
  - 40G to 100G on a single FPGA

P4 and FPGA are perfect for each other!



But programming FPGA is hard!

- Language? Timing Error?
- What is the right architecture?
- How to deal with resource constraint?

We build a flexible P4 backend and compiler for FPGA

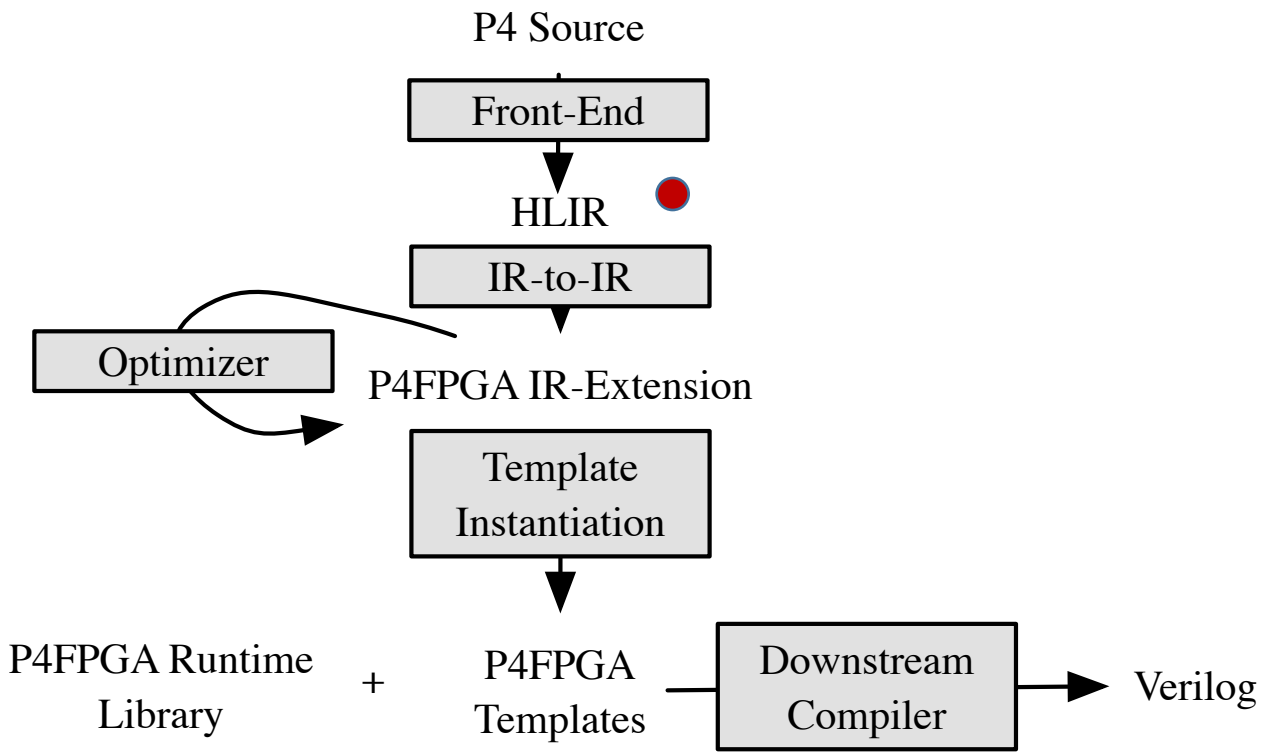
Contributions

- A new P4 to FPGA compiler
- Implementation in high level synthesis language
- Flexible and efficient architecture

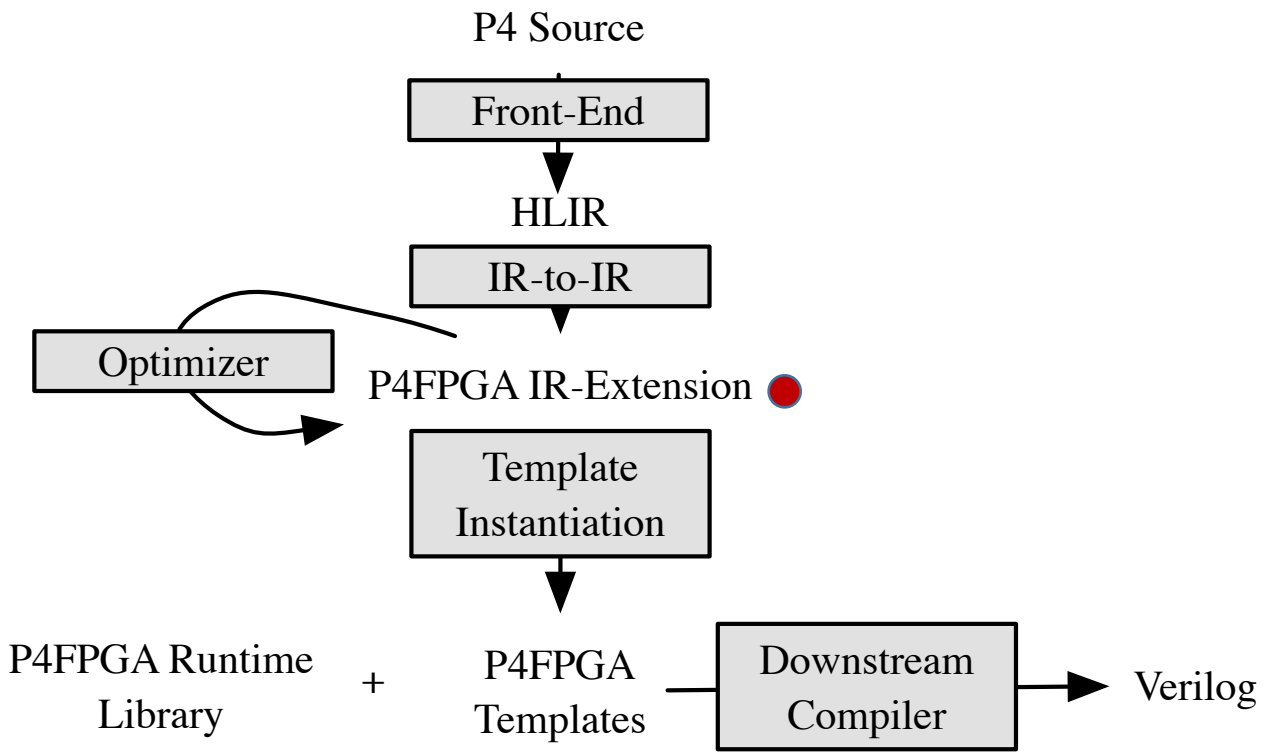
Outline

- Introduction
- Design overview
- Implementation
- Evaluation
- Demo
- Related work and Conclusion

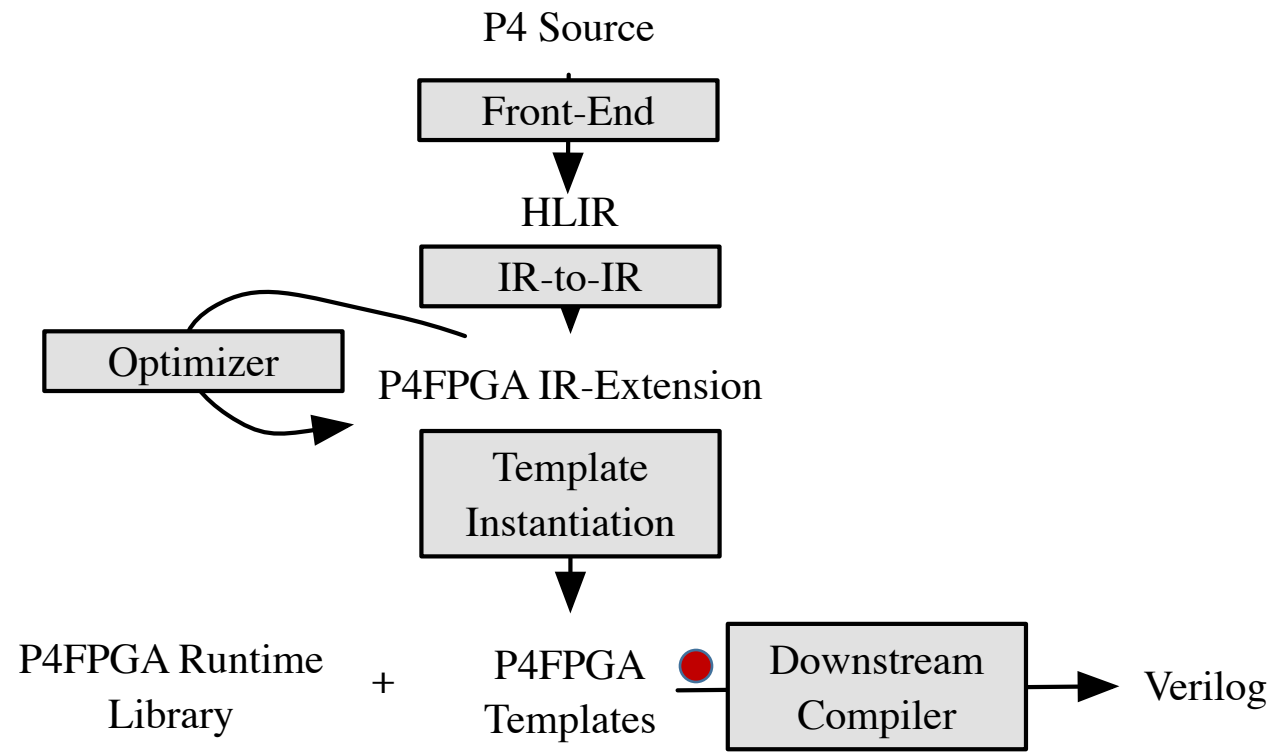
P4 to FPGA in 10000 foot view



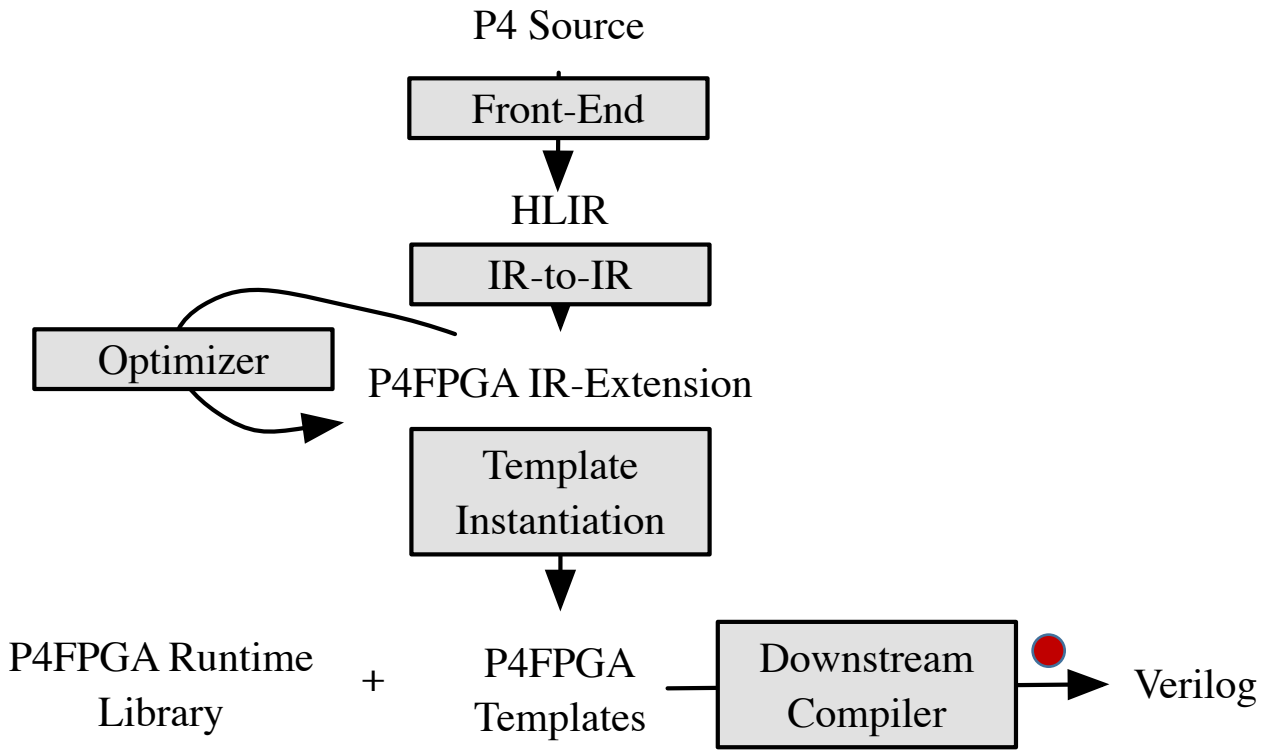
P4 to FPGA in 10000 foot view



P4 to FPGA in 10000 foot view

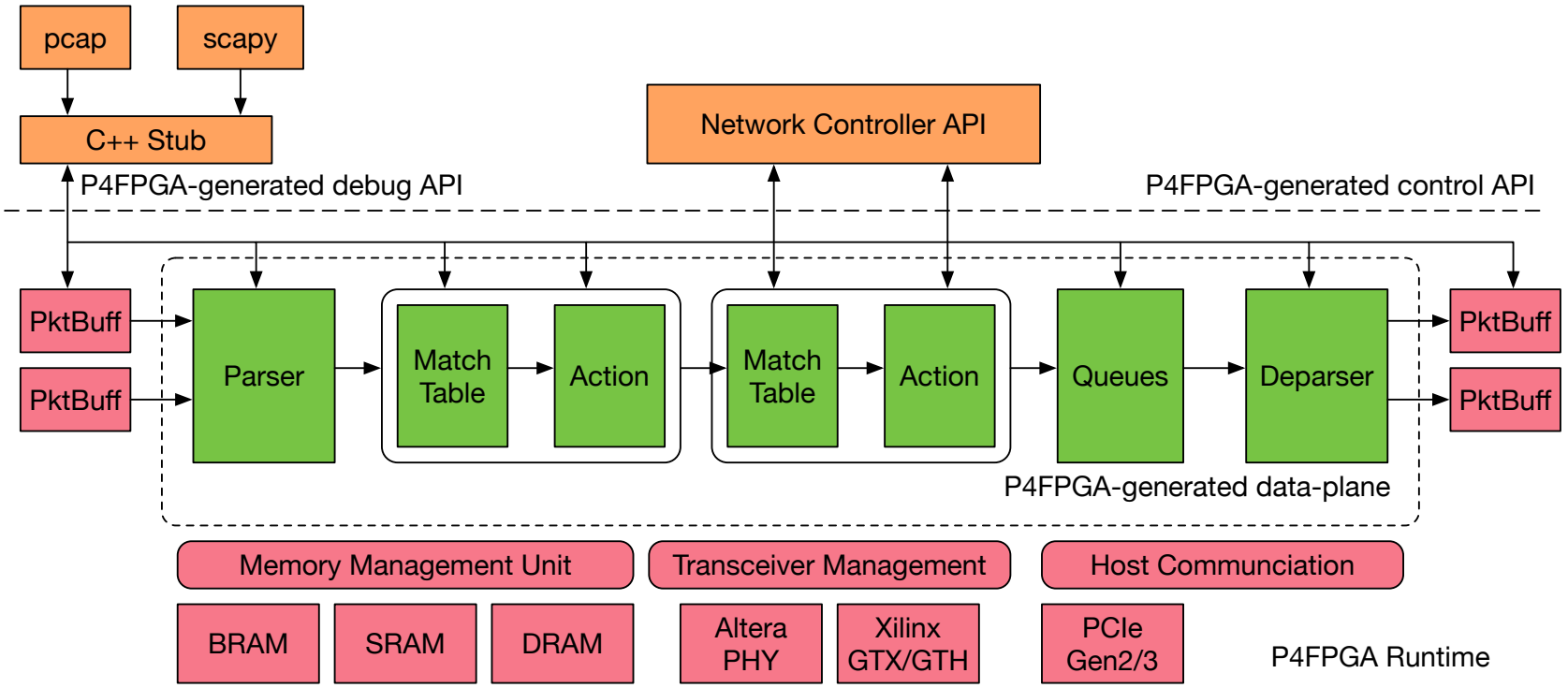


P4 to FPGA in 10000 foot view



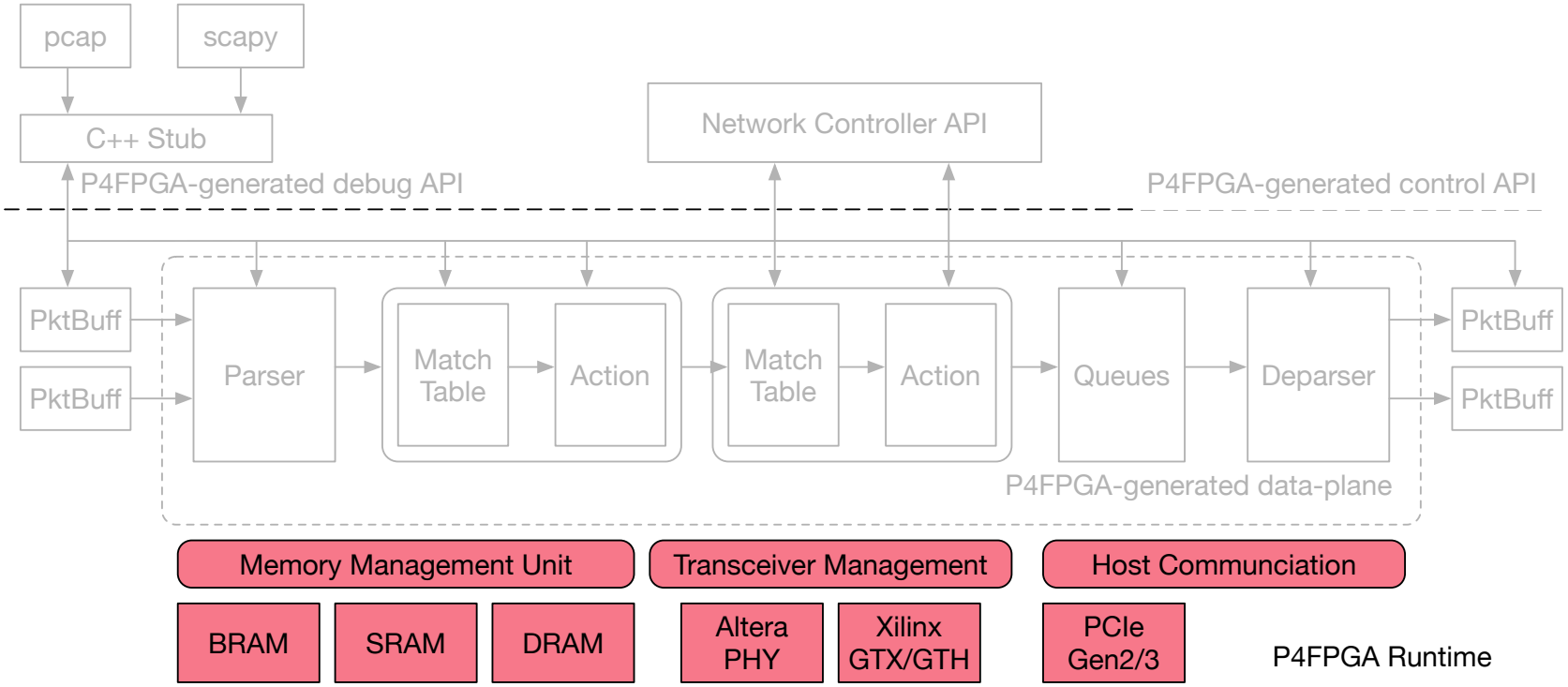
Design Overview

- P4FPGA Runtime
- P4FPGA Template: Code Generation
- P4FPGA API and debug interface



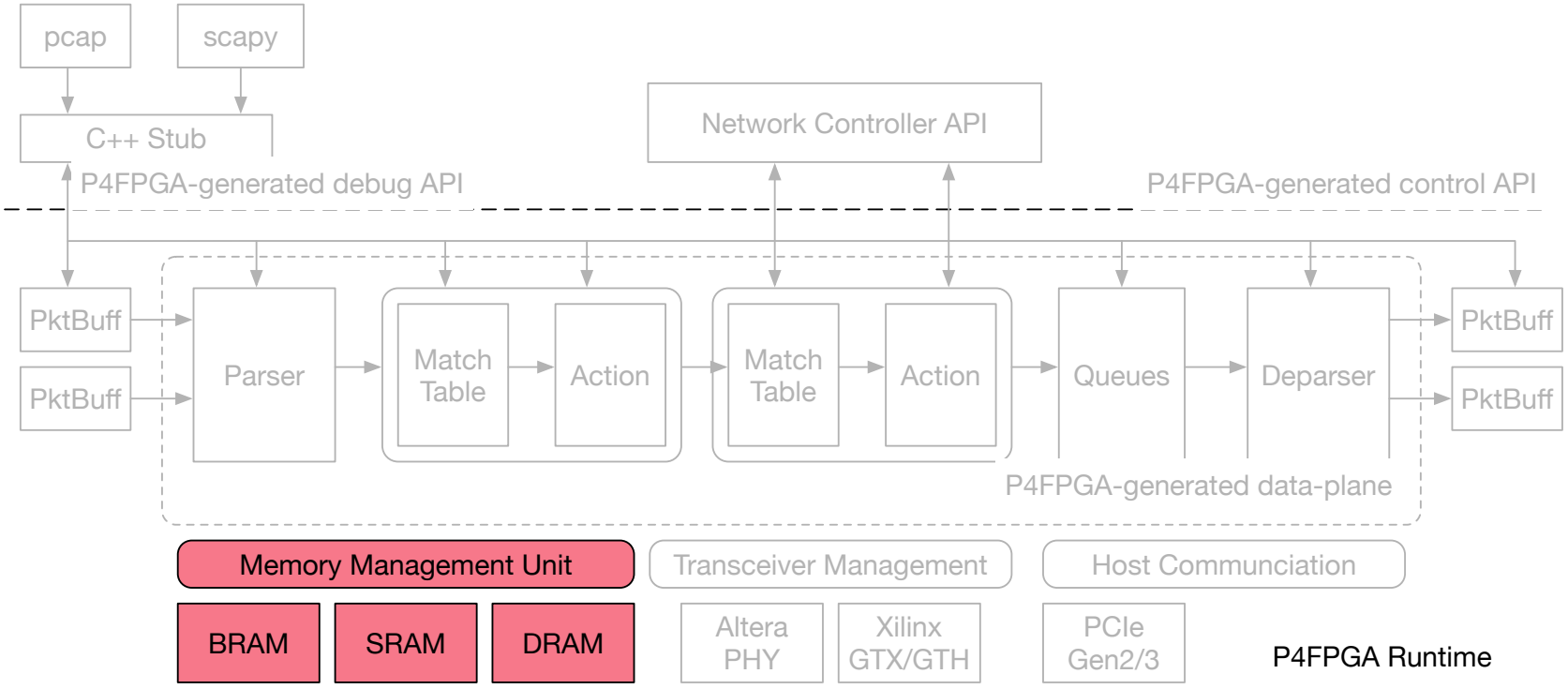
Design Overview

- P4FPGA Runtime
- P4FPGA Template: Code Generation
- P4FPGA API and debug interface



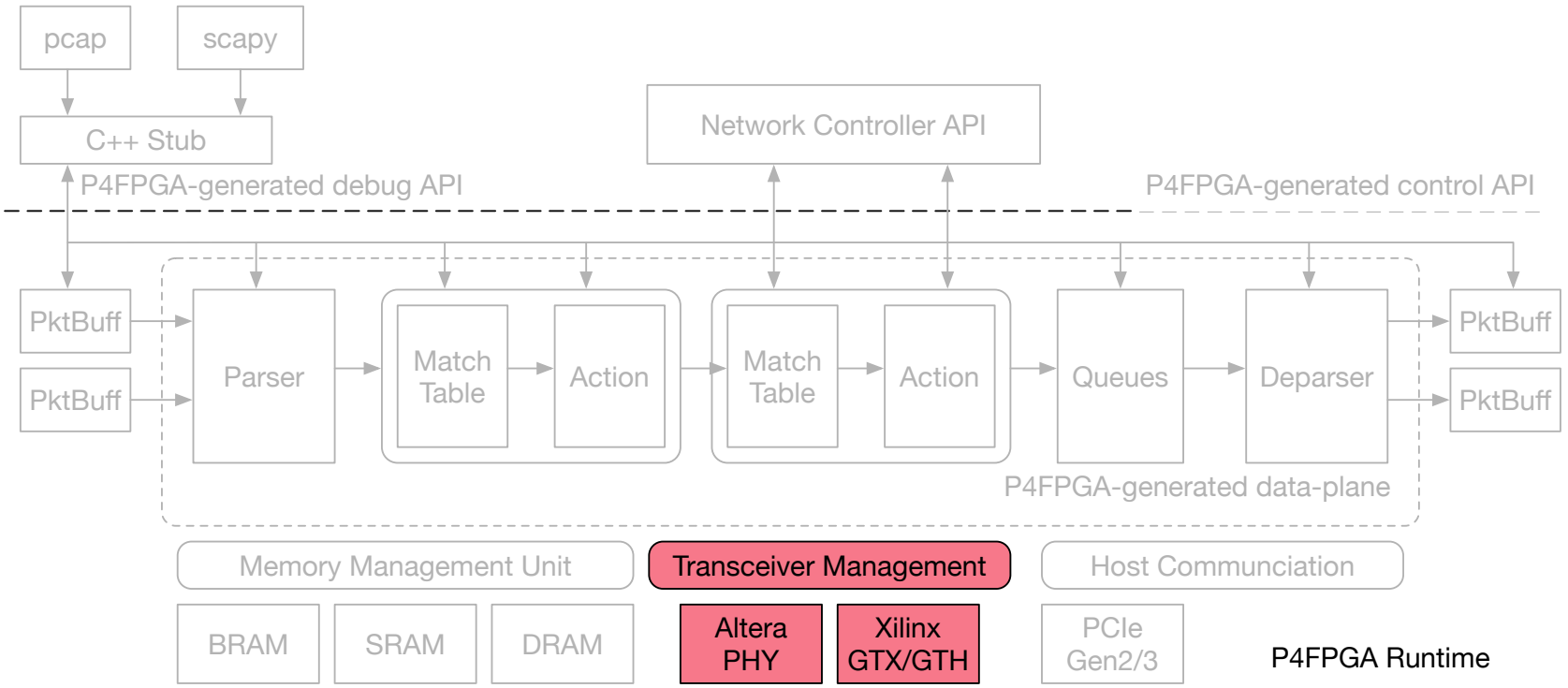
P4FPGA Runtime

- Shared packet memory
  - Represent packet with a unique token ID
  - Expose consistent interface to pipeline
  - Hardware managed heap, 256 bytes per page



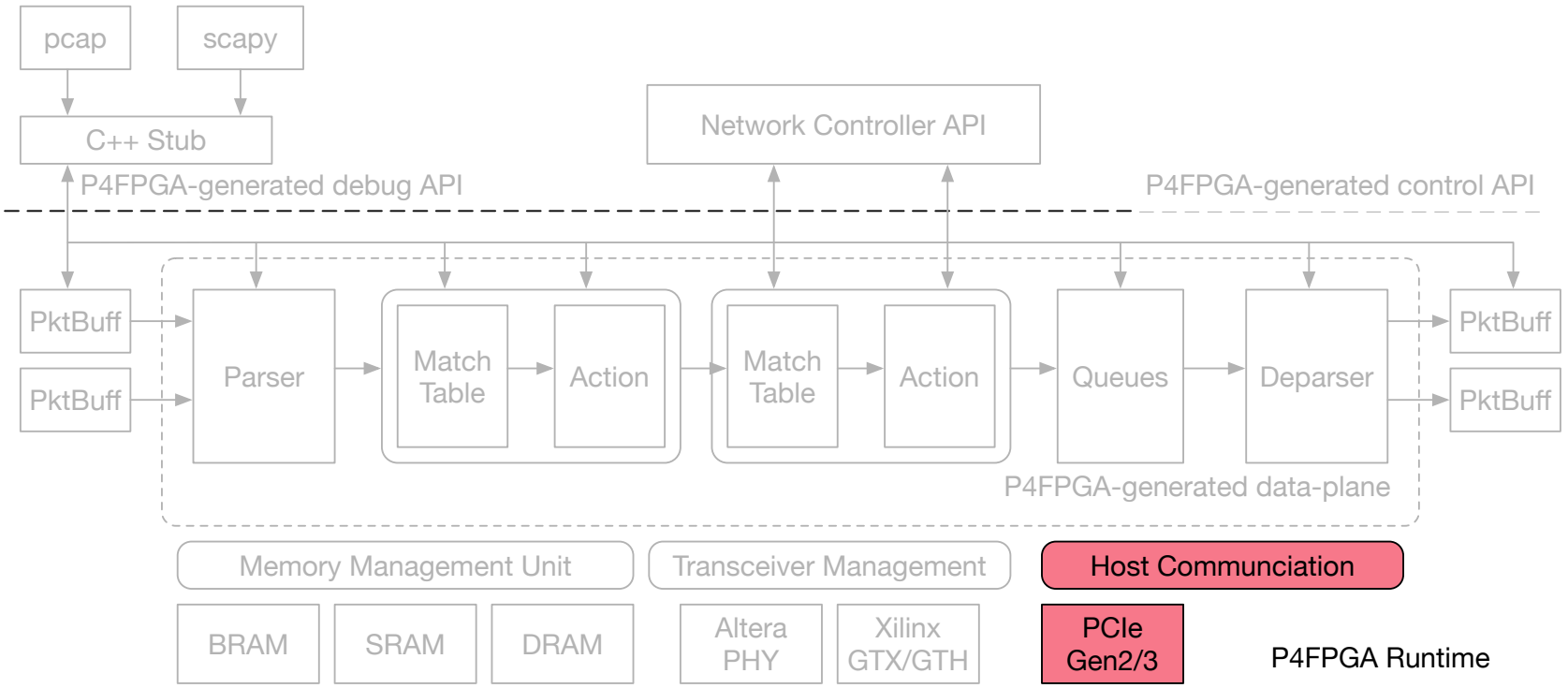
P4FPGA Runtime

- Transceiver/Clock management
  - Hide device difference with common data structure
  - EthData { data, mask, sop, eop }



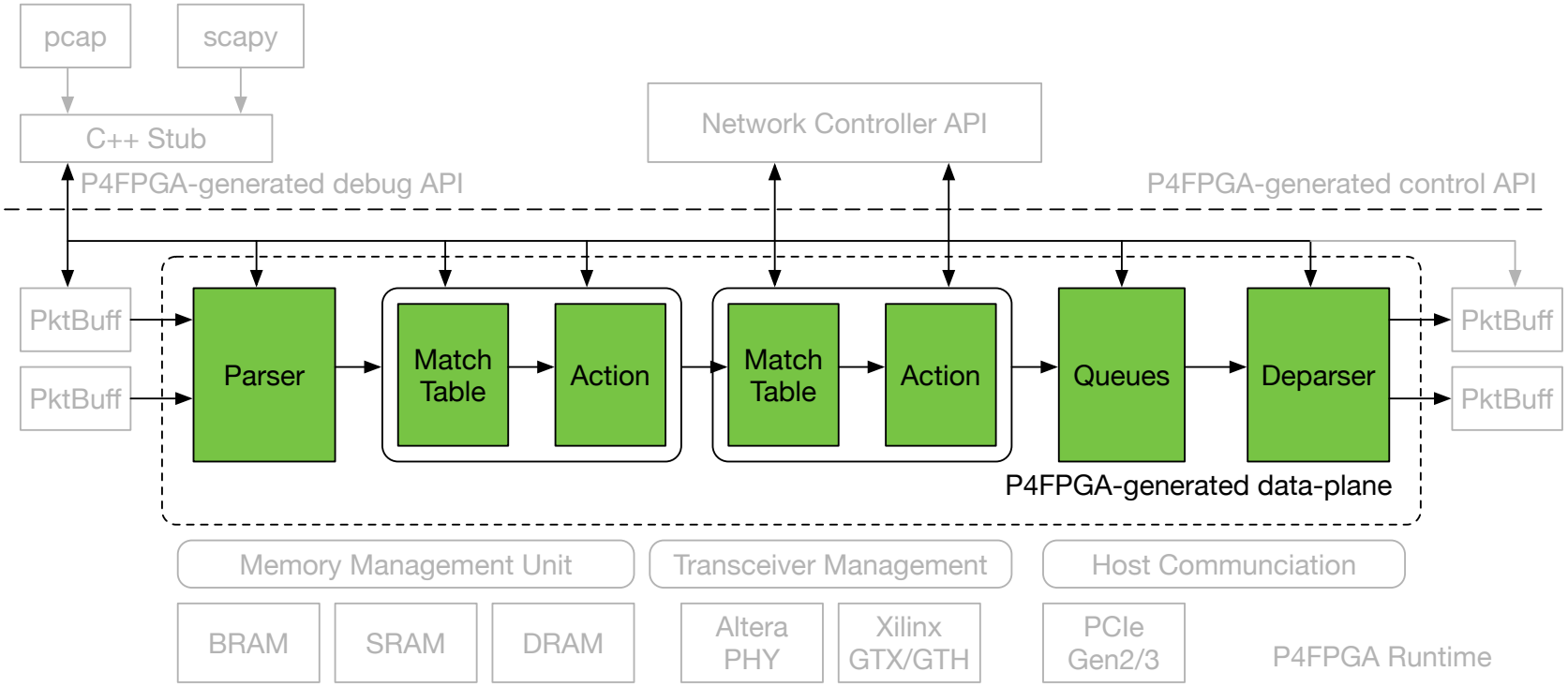
P4FPGA Runtime

- Host communication
  - DMA Engine and MMIO
  - PCIe Gen2 and Gen3 speed



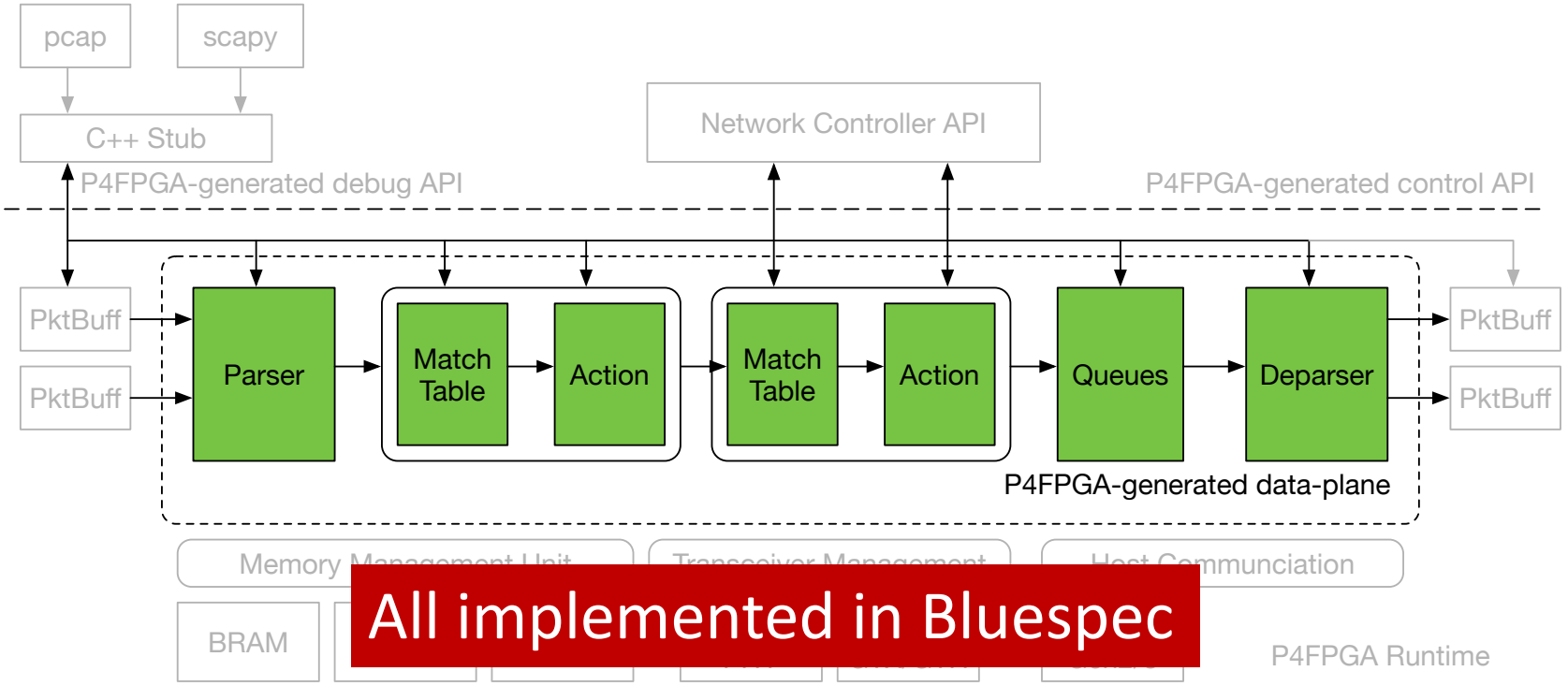
Design Overview

- Runtime
- P4FPGA Templates : Code Generation
- API and User interface



P4FGPA Templates: Code Generation

- Templates to implement P4 in FPGA
  - Parser, Deparser, Dataflow node, Computation node
- Reference Implementation:
  - Parser -> Ingress -> Egress -> Deparser



P4 Source -> P4FPGA Template

P4 Source: parser.p4

```
parser parse_ethernet {
  extract(ethernet);
  return select(latest.etherType) {
    ETHERTYPE_IPV4: parse_ipv4;
    ETHERTYPE_ICMP: parse_icmp;
     ...
  }
}
```

P4 Template

```
rule load_eth if (state==ParseEth &&
                  buffered < 112);
  // concat data to buffer
  rg_tmp <= data_in << shift | rg_tmp;
  // update shift amount
  move_shift_amt(128);
endrule

rule extract_eth if (state==ParseEth &&
                     buffered >= 112);
  EthernetT ether = unpack(truncate(buff));
  select(ether.etherType);
endrule


function select (Bit#(16) type);
  case type matches
  IPv4: state <= ParseIpv4;
  ICMP: state <= ParseIcmp;
  endcase
endfunction
```

P4 Source -> P4FPGA Template

P4 Source: parser.p4

```
parser parse_ethernet {
  extract(ethernet);
  return select(latest.etherType) {
    ETHERTYPE_IPV4: parse_ipv4;
    ETHERTYPE_ICMP: parse_icmp;
    ...
  }
}
```

P4 Template

```
rule load_eth if (state==ParseEth &&
                  buffered < 112);
  // concat data to buffer
  rg_tmp <= data_in << shift | rg_tmp;
  // update shift amount
  move_shift_amt(128);
endrule

rule extract_eth if (state==ParseEth &&
                     buffered >= 112);
  EthernetT ether = unpack(truncate(buff));
  select(ether.etherType);
endrule


function select (Bit#(16) type);
  case type matches
  IPv4: state <= ParseIpv4;
  ICMP: state <= ParseIcmp;
  endcase
endfunction
```

P4 Source -> P4FPGA Template

P4 Source: parser.p4

```
parser parse_ethernet {
  extract(ethernet);
  return select(latest.etherType) {
    ETHERTYPE_IPV4: parse_ipv4;
    ETHERTYPE_ICMP: parse_icmp;
     ...
  }
}
```

P4 Template

```
rule load_eth if (state==ParseEth &&
                  buffered < 112);
  // concat data to buffer
  rg_tmp <= data_in << shift | rg_tmp;
  // update shift amount
  move_shift_amt(128);
endrule

rule extract_eth if (state==ParseEth &&
                     buffered >= 112);
  EthernetT ether = unpack(truncate(buff));
  select(ether.etherType);
endrule


function select (Bit#(16) type);
  case type matches
  IPv4: state <= ParseIpv4;
  ICMP: state <= ParseIcmp;
  endcase
endfunction
```

P4 Source -> P4FPGA Template

P4 Source: parser.p4

```
parser parse_ethernet {
  extract(ethernet);
  return select(latest.etherType) {
    ETHERTYPE_IPV4: parse_ipv4;
    ETHERTYPE_ICMP: parse_icmp;
     ...
  }
}
```

P4 Template

```
rule load_eth if (state==ParseEth &&
                  buffered < 112);
  // concat data to buffer
  rg_tmp <= data_in << shift | rg_tmp;
  // update shift amount
  move_shift_amt(128);
endrule

rule extract_eth if (state==ParseEth &&
                     buffered >= 112);
  EthernetT ether = unpack(truncate(buff));
  select(ether.etherType);
endrule


function select (Bit#(16) type);
  case type matches
  IPv4: state <= ParseIpv4;
  ICMP: state <= ParseIcmp;
  endcase
endfunction
```

P4 Source -> P4FPGA Template

P4 Source: parser.p4

```
parser parse_ethernet {
  extract(ethernet);
  return select(latest.etherType) {
    ETHERTYPE_IPV4: parse_ipv4;
    ETHERTYPE_ICMP: parse_icmp;
    ...
  }
}
```

P4 Template

```
rule load_eth if (state==ParseEth &&
                  buffered < 112);
  // concat data to buffer
  rg_tmp <= data_in << shift | rg_tmp;
  // update shift amount
  move_shift_amt(128);
endrule

rule extract_eth if (state==ParseEth &&
                     buffered >= 112);
  EthernetT ether = unpack(truncate(buff));
  select(ether.etherType);
endrule


function select (Bit#(16) type);
  case type matches
  IPv4: state <= ParseIpv4;
  ICMP: state <= ParseIcmp;
  endcase
endfunction
```

P4 Source -> P4FPGA Template

P4 Source: parser.p4

```
parser parse_ethernet {
  extract(ethernet);
  return select(latest.etherType) {
    ETHERTYPE_IPV4: parse_ipv4;
    ETHERTYPE_ICMP: parse_icmp;
     ...
  }
}
```

P4 Template

```
rule load_eth if (state==ParseEth &&
                  buffered < 112);
  // concat data to buffer
  rg_tmp <= data_in << shift | rg_tmp;
  // update shift amount
  move_shift_amt(128);
endrule

rule extract_eth if (state==ParseEth &&
                     buffered >= 112);
  EthernetT ether = unpack(truncate(buff));
  select(ether.etherType);
endrule


function select (Bit#(16) type);
  case type matches
  IPv4: state <= ParseIpv4;
  ICMP: state <= ParseIcmp;
  endcase
endfunction
```

P4 Source -> P4FPGA Template

P4 Source: parser.p4

```
parser parse_ethernet {
  extract(ethernet);
  return select(latest.etherType) {
    ETHERTYPE_IPV4: parse_ipv4;
    ETHERTYPE_ICMP: parse_icmp;
     ...
  }
}
```

P4 Template

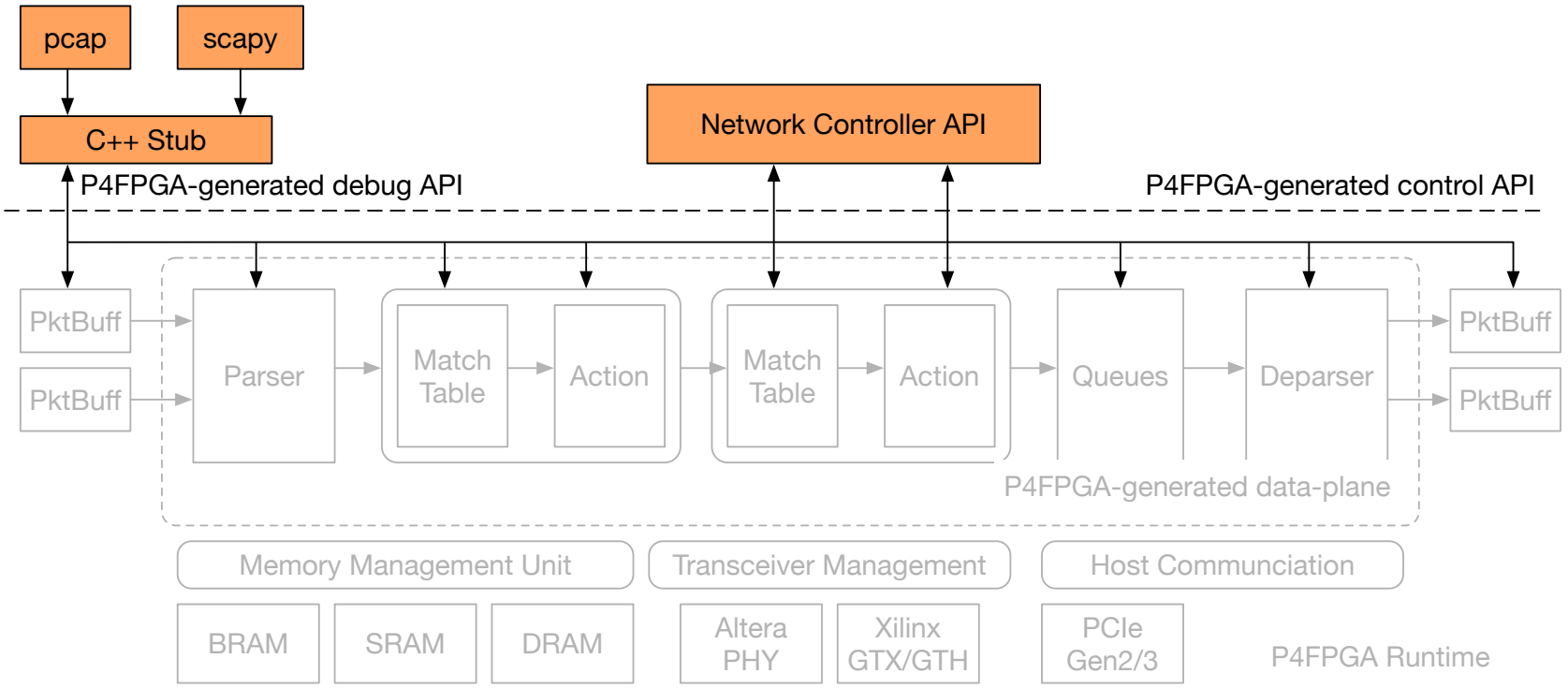
```
rule load_eth if (state==ParseEth &&
                  buffered < 112);
  // concat data to buffer
  rg_tmp <= data_in << shift | rg_tmp;
  // update shift amount
  move_shift_amt(128);
endrule

rule extract_eth if (state==ParseEth &&
                     buffered >= 112);
  EthernetT ether = unpack(truncate(buff));
  select(ether.etherType);
endrule


function select (Bit#(16) type);
  case type matches
  IPv4: state <= ParseIpv4;
  ICMP: state <= ParseIcmp;
  endcase
endfunction
```

P4FPGA API and debug interface

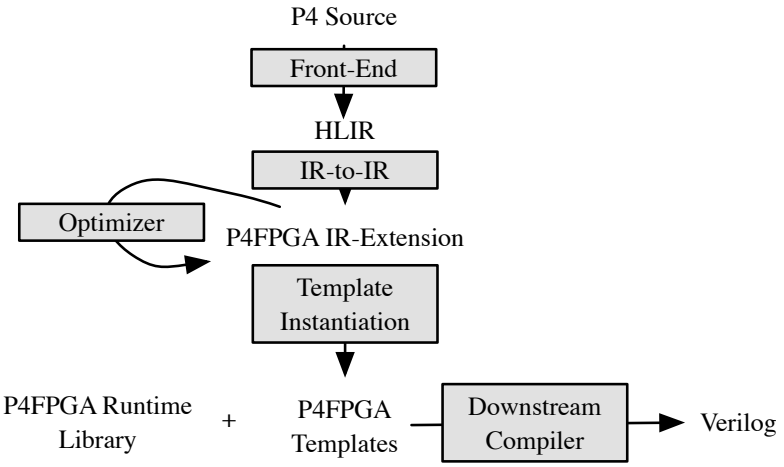
- Auto-generated software and hardware interface
- Packet generation / Packet capture
- Table insertion / deletion / modification



Outline

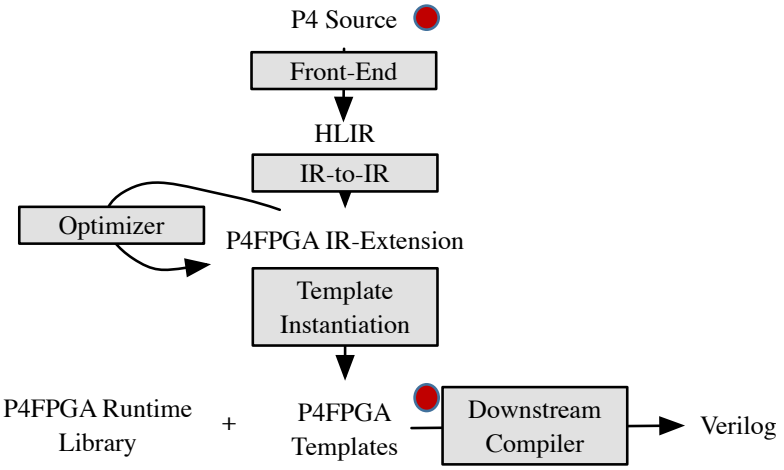
- Introduction
- Design overview
- Implementation
- Evaluation
- Demo
- Related work and Conclusion

Implementation



- P4FPGA Runtime + Templates
  - Implemented in Bluespec
  - 10000 lines of code
- P4FPGA Compiler : IR-to-IR + Template instantiation
  - 5000 Lines of Python
- P4 Standard
  - Compatible with P4-14
  - Porting to C++ frontend and P4-16

Case studies



	LOC in P4	LOC in P4FPGA Template (Bluespec)	
switch.p4	8961	45575	5x
paxos.p4	385	3306	8.5x
phy.p4	946	4954	5.2x
tcp-diagnosis.p4	804	4909	6.1x

Outline

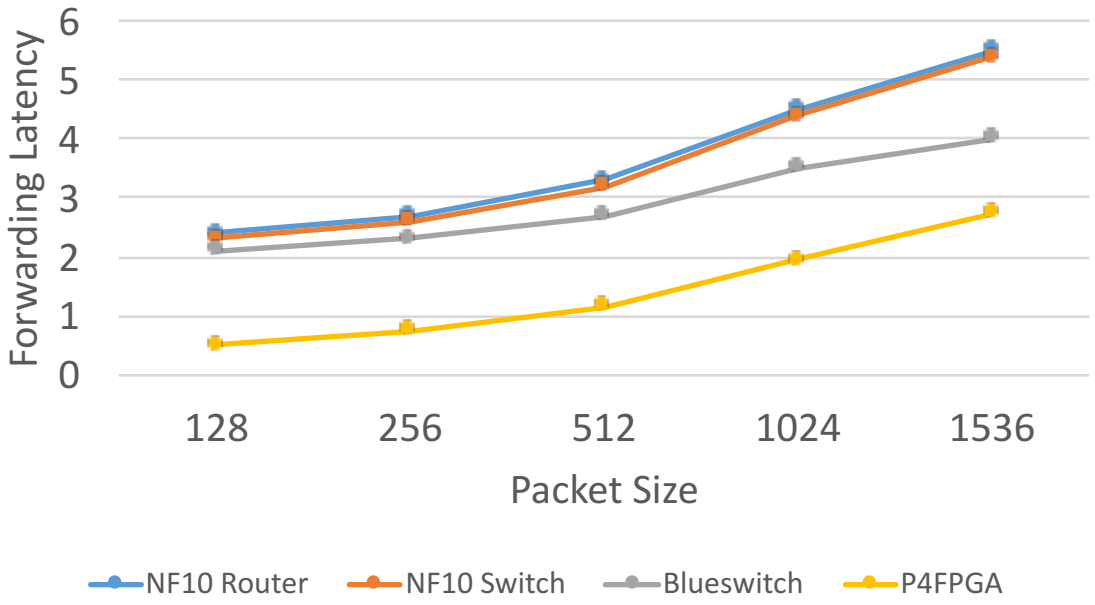
- Introduction
- Design overview
- Implementation
- Evaluation
- Demo
- Related work and Conclusion

Evaluation

- Focused on the performance of generated code:
  - Compared to hand-written Verilog code
  - Compared to commercial compilers
  - FPGA Resource Limits
- Testbed:
  - Bluespec cycle-accurate simulation
  - NetFPGA SUME
  - Altera DE5

P4FPGA performance is good

- Compared to hand-written Verilog code
- router.p4: Implemented 10Gbps, 4 ports



Comparable Performance to existing research prototype in Verilog

NF10, Blueswitch number reproduced from
Blueswitch : Enabling Provably Consistent Configuration of Network Switches (ANCS '15)

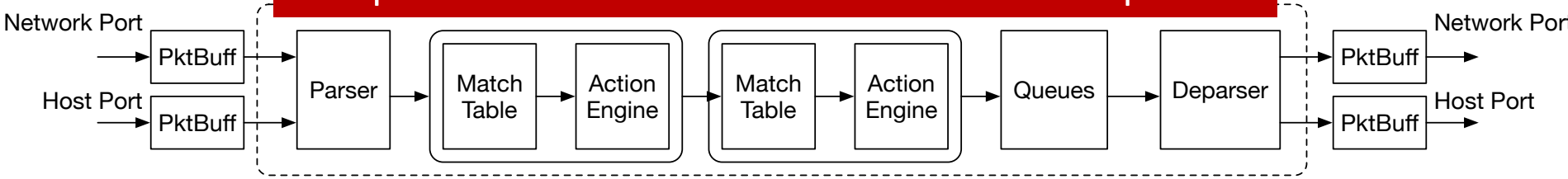
P4FPGA performance is good

- Compared to two other commercial compilers
- “Network Hardware-Accelerated Consensus”

	P4FPGA	Compiler 1	Compiler 2
Forwarding	0.37us	0.73us	-
Acceptor	0.79us	1.44us	0.81us
Coordinator	0.72us	1.21us	0.33us

Throughput: 102 byte @ 9 million pps, close to line rate

Comparable Performance to industrial counterparts



FPGA Resource Limits

- Support BCAM and TCAM
- on-chip / off-chip mode
- On-chip TCAM up to 32k entries for 288 bit key

	Available	1K	2K	4K	8K	16K	32K
ALMs	234,720	20,547(9%)	20,999(9%)	21,504(9%)	22,026(10%)	23,946(11%)	25,120(12%)
Registers	939,000	13,754(1%)	15,277(2%)	15,003(2%)	15,521(2%)	17,368(2%)	18,125(2%)
BlockRAM	2,560	243(9%)	292(11%)	391(15%)	688(27%)	1,184(46%)	2,364(92%)

288-bit TCAM, Altera Stratix V 5SGXMA7H2F35C2
Axonerve, Nagase Inc.

ALM: Adaptive Logic Module

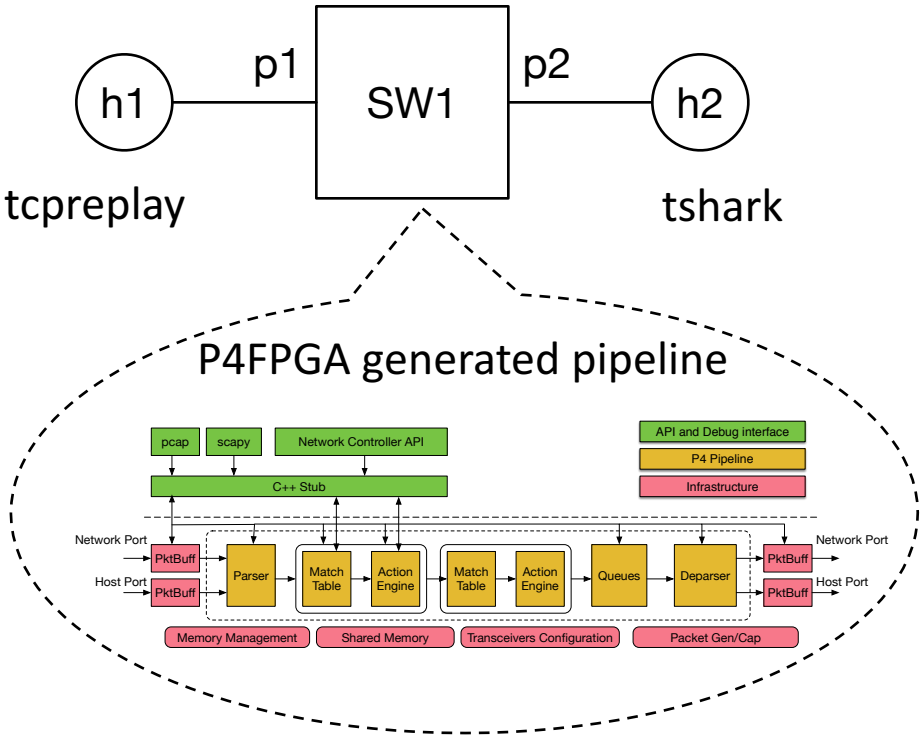
Work in progress

- Limited resource constraint on FPGA
  - How to fit switch.p4 on FPGA?
- Dealing with different architectures
  - Streaming versus shared memory
  - Store-and-forward versus cut-through
- Shared access to registers, external modules
  - How to resolve RAW hazards
- Correct-by-construction versus verification

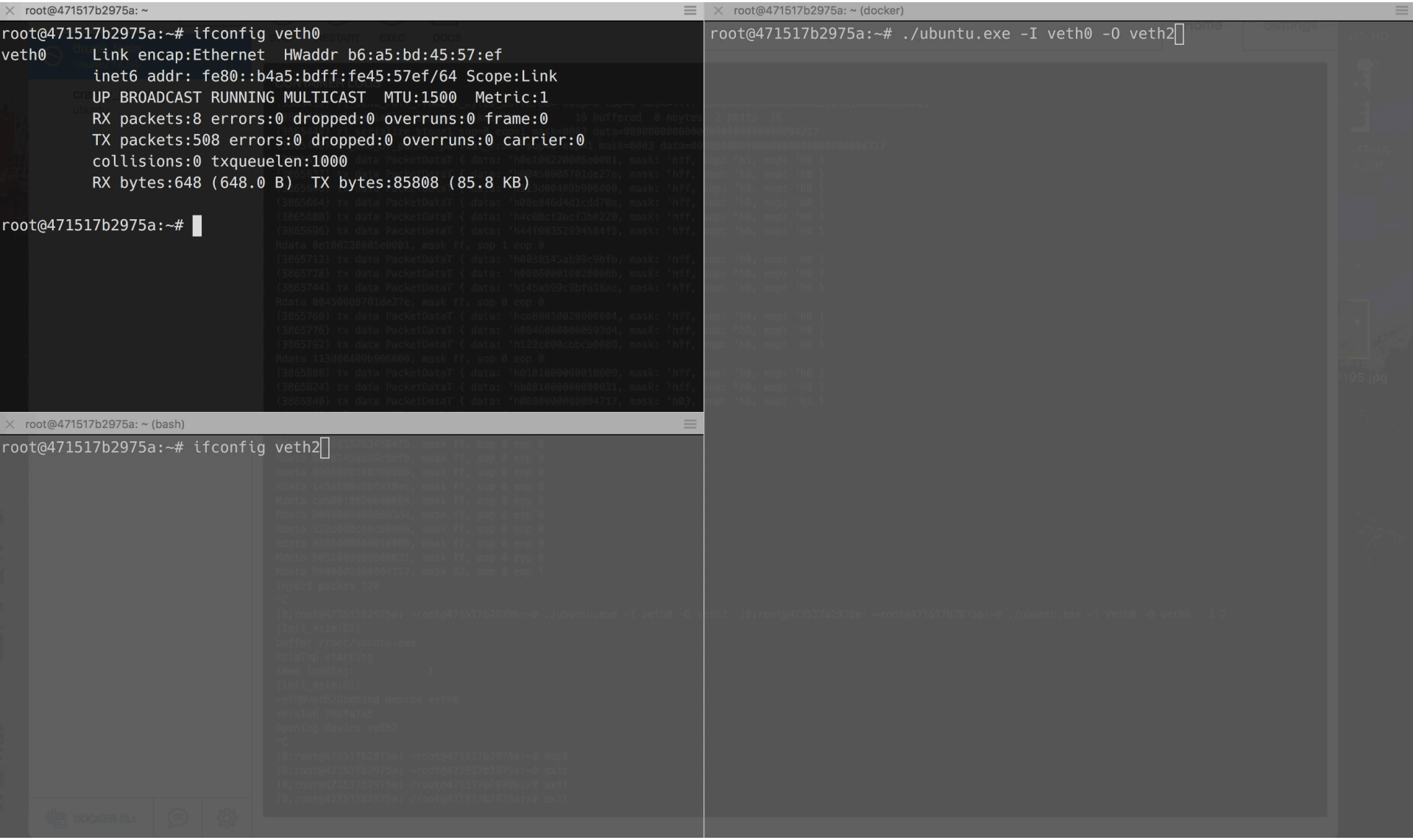
Outline

- Introduction
- Design overview
- Implementation
- Evaluation
- Demo
- Related work and Conclusion

Demo



No.	Time	Source	Destination	Protocol	Length	Frame	Info
1	0.000000000	205.209.212.70	224.0.32.2	UDP	114	Yes	15311 → 15311 Len=68 [
2	0.000000849	205.209.221.70	224.0.31.2	UDP	114	Yes	14311 → 14311 Len=68 [
3	0.012717210	205.209.212.70	224.0.32.2	UDP	178	Yes	15311 → 15311 Len=132
4	0.012717791	205.209.221.70	224.0.31.2	UDP	178	Yes	14311 → 14311 Len=132
5	0.016138196	205.209.221.70	224.0.31.2	UDP	146	Yes	14311 → 14311 Len=100
6	0.016139187	205.209.212.70	224.0.32.2	UDP	146	Yes	15311 → 15311 Len=100


root@471517b2975a: ~
root@471517b2975a:~# ifconfig veth0
veth0 Link encap:Ethernet HWaddr b6:a5:bd:45:57:ef
inet6 addr: fe80::b4a5:bdff:fe45:57ef/64 Scope:Link
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:8 errors:0 dropped:0 overruns:0 frame:0
TX packets:508 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:648 (648.0 B) TX bytes:85808 (85.8 KB)
root@471517b2975a:~#
root@471517b2975a: ~ (bash)
root@471517b2975a:~# ifconfig veth2
root@471517b2975a: ~ (docker)
root@471517b2975a:~# ./ubuntu.exe -I veth0 -O veth2

Related work

- ClickNP (SIGCOMM ‘16)
  - Click to OpenCL
  - OpenCL to FPGA via High Level Synthesis tool
  - Programming model
- Xilinx SDNet
  - P4 to PX
  - PX to FPGA via Xilinx’s PX compiler

Conclusion

- A new backend/compiler to support P4 on FPGA
- Comparable performance to hand-written design
- Comparable performance to industry compilers
- Used to generate many prototypes

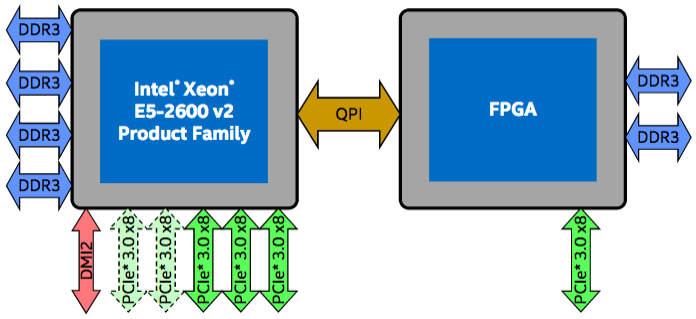
Thank you

http://www.p4fpga.org

Email: hwang@cs.cornell.edu

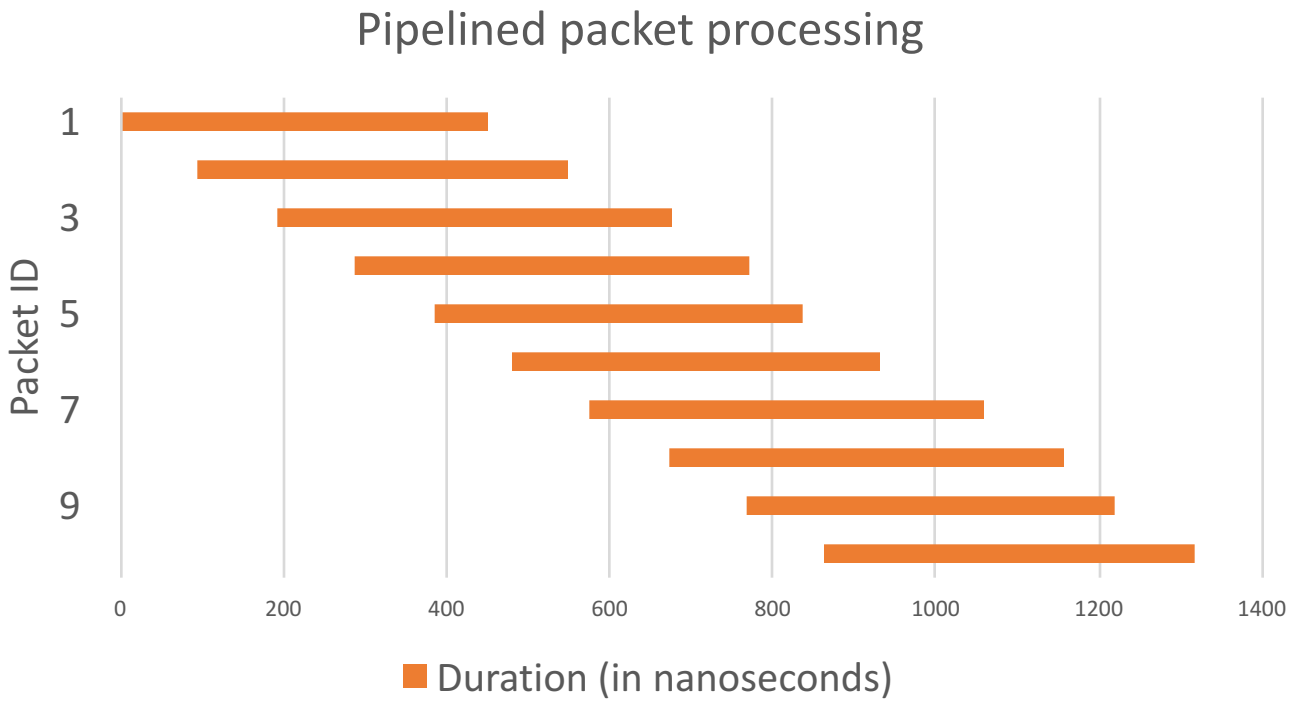
Future Work

- Apply P4 to heterogeneous architecture?
  - Intel Xeon + FPGA over QPI
  - Deploy P4 program over multiple targets
- Find P4's use case beyond networking
  - Data processing acceleration



Backup Slides

Pipeline



Throughput is improved by pipelined packet processing

Why Bluespec over Verilog?

- Guarded Atomic Rules
- Type Checking
  - No wire mismatch
- Polymorphism
  - FIFO#(MetadataT), FIFO#(Bit#(128))
- Library
  - Reusable primitives
- Simulation
  - debugging at higher level of abstraction

Increase programmer productivity by at least 3 to 5 x

Headers and Fields

- Header and metadata maps to Struct
- Tagged Union for control flow metadata

```
header_type ethernet_t {
   fields {
      dstAddr: 48;
      srcAddr: 48;
      etherType: 16;
   }
}
/* instance */
header ethernet_t ether;
```

P4

```
typedef struct {
   Bit#(48) dstAddr;
   Bit#(48) srcAddr;
   Bit#(16) etherType;
} EthernetT deriving (Bits);

/* Register */
Reg#(EthernetT) ethernet <- mkRegU;
```

Bluespec

Headers and Fields

- Header and metadata maps to Struct
- Tagged Union for control flow metadata

```
table acl{
   read {
      ipv4.dstAddr: ternary
   }
   actions {
      no_op;
      drop;
   }
}
```

P4

```
typedef union tagged {
   struct {
      PacketId pkt;
      MetadataT meta;
   } NO_OP;
   struct {
      PacketId pkt;
      MetadataT meta;
   } DROP;
} ACL_RspT deriving (Bits);
```

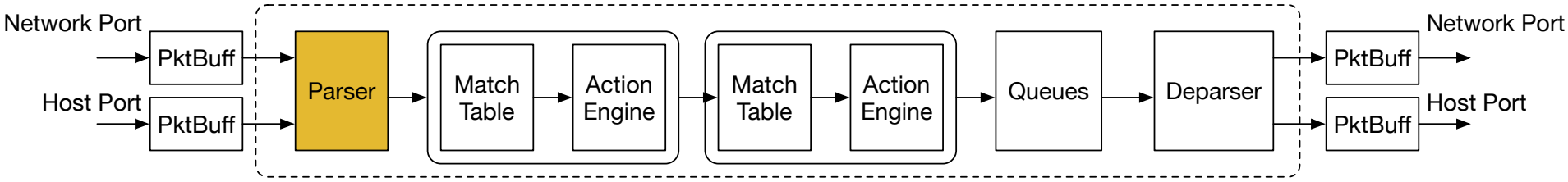
Bluespec

Parser

- Two atomic rules for each states
  - Append incoming bit-stream to buffer
  - Extract header and transit to next state

```
parser parse_ethernet {
  extract(ethernet);
  return select(latest.etherType) {
    ETHERTYPE_IPV4: parse_ipv4;
    ETHERTYPE_ICMP: parse_icmp;
     ...
  }
}
```

```
rule load_eth if (state==ParseEth &&
                  buffered < 112);
  // concat data to buffer
  rg_tmp <= data_in << shift | rg_tmp;
  // update shift amount
  move_shift_amt(128);
}
```

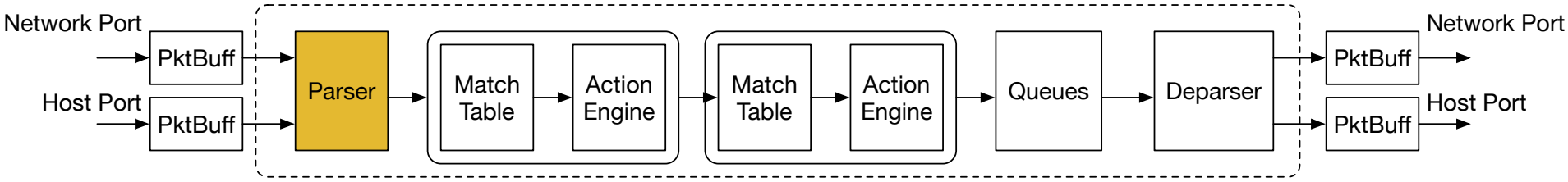


Parser

- Two atomic rules for each states
  - Append incoming bit-stream to buffer
  - Extract header and transit to next state
    - unpack and truncate are built-in bluespec functions

```
parser parse_ethernet {
  extract(ethernet);
  return select(latest.etherType) {
    ETHERTYPE_IPV4: parse_ipv4;
    ETHERTYPE_ICMP: parse_icmp;
     ...
  }
}
```

```
rule extract_eth if (state==ParseEth &&
                     buffered >= 112);
  // concat data to buffer
  EthernetT ether = unpack(truncate(buff));
  // update shift amount
  compute_next_state(ether);
}
```

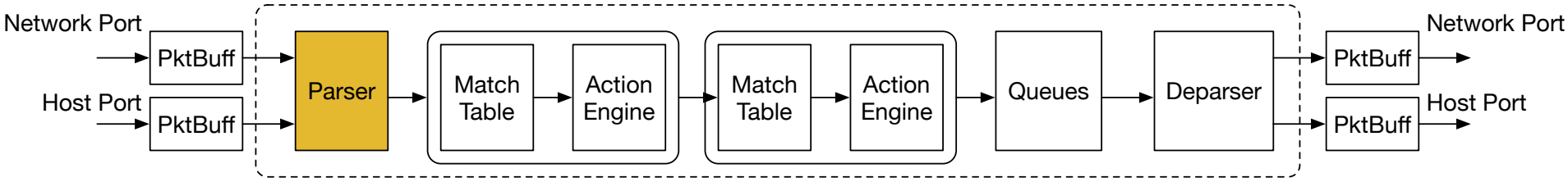


Parser

- Two atomic rules for each states
  - Append incoming bit-stream to buffer
  - Extract header and transit to next state
    - unpack and truncate are built-in bluespec functions

```
parser parse_ethernet {
  extract(ethernet);
  return select(latest.etherType) {
    ETHERTYPE_IPV4: parse_ipv4;
    ETHERTYPE_ICMP: parse_icmp;
     ...
  }
}
```

```
rule extract_eth if (state==ParseEth &&
                     buffered >= 112);
  // concat data to buffer
  EthernetT ether = unpack(truncate(buff));
  // update shift amount
  compute_next_state(ether);
}
```

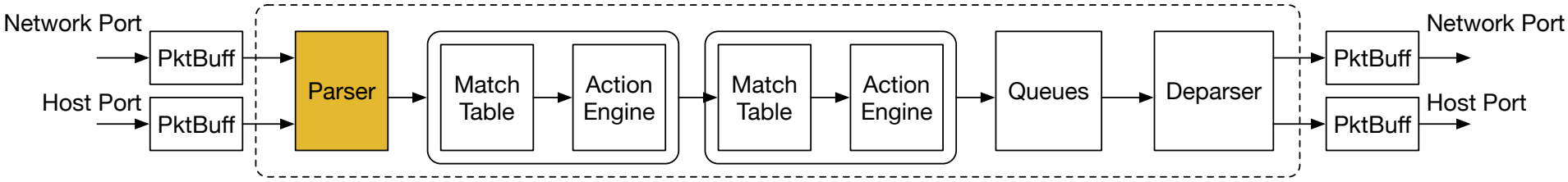


Parser

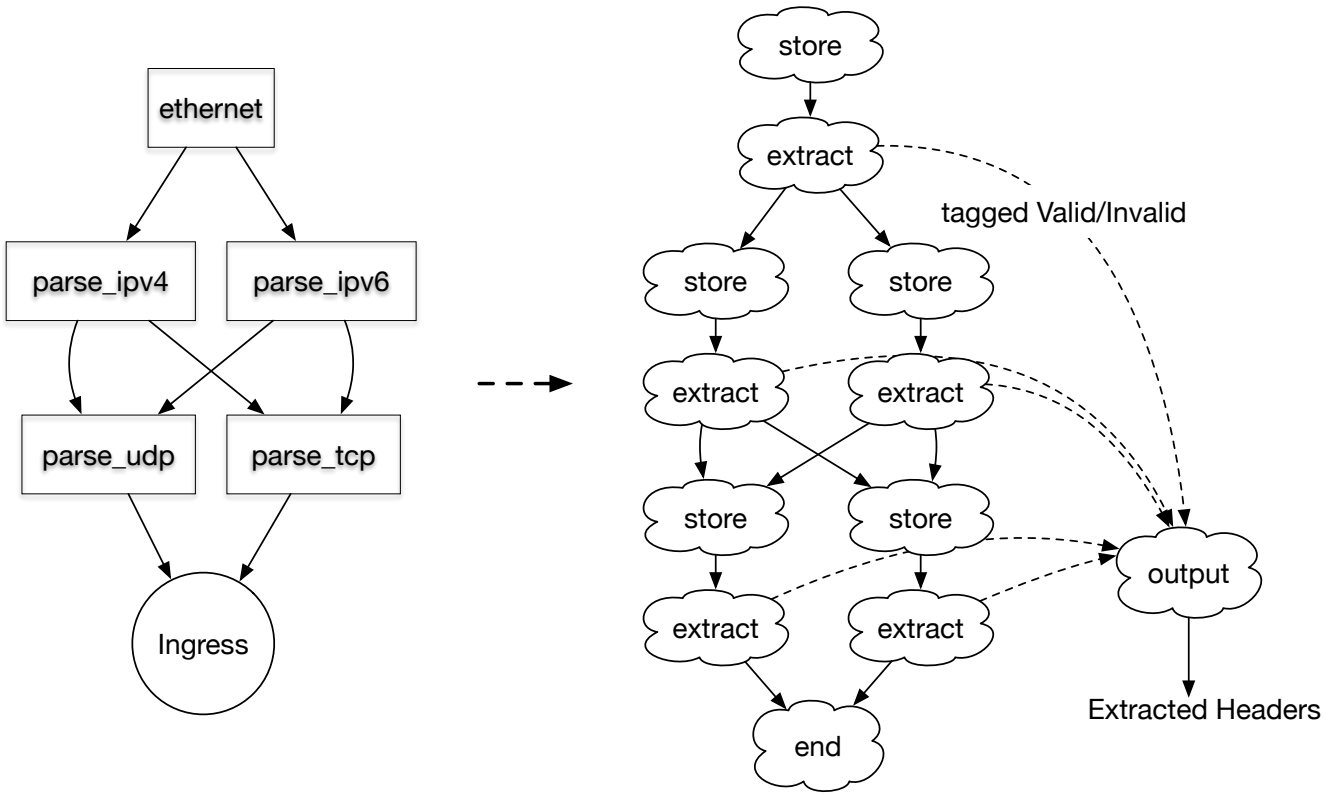
- Two atomic rules for each states
  - Append incoming bit-stream to buffer
  - Extract header and transit to next state
    - w_* will send a signal to another rule

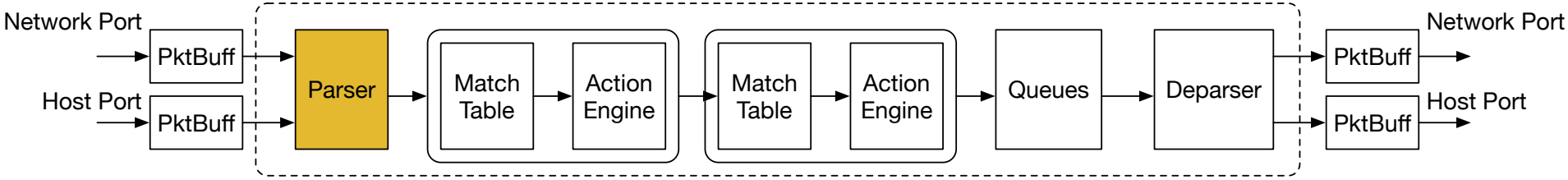
```
parser parse_ethernet {
  extract(ethernet);
  return select(latest.etherType) {
    ETHERTYPE_IPV4: parse_ipv4;
    ETHERTYPE_ICMP: parse_icmp;
     ...
  }
}
```

```
function compute_next_state (EthernetT eth);
  case (eth.etherType) matches
    IPV4: w_parse_ipv4.send();
    ICMP: w_parse_icmp.send();
  endcase
endfunction
```



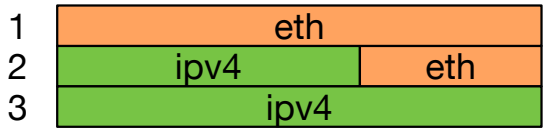
Parser

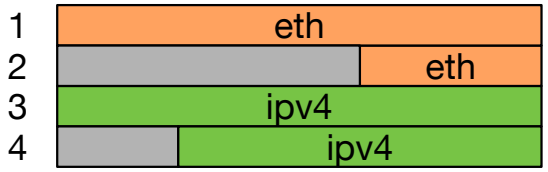

ethernet
parse_ipv4
parse_ipv6
parse_udp
parse_tcp
Ingress
store
extract
tagged Valid/Invalid
store
store
extract
extract
store
store
output
extract
extract
Extracted Headers
end

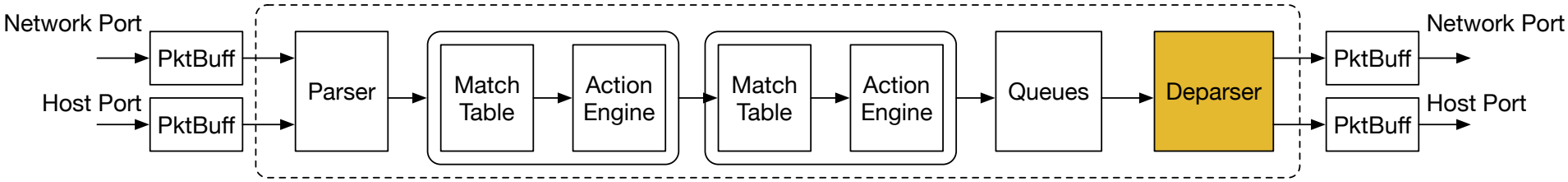

Network Port
PktBuff
Host Port
PktBuff
Parser
Match Table
Action Engine
Match Table
Action Engine
Queues
Deparser
PktBuff
Network Port
PktBuff
Host Port

Deparser

- Goal: Simplify code generation
  - Packet header
- Remove header marked as Invalid

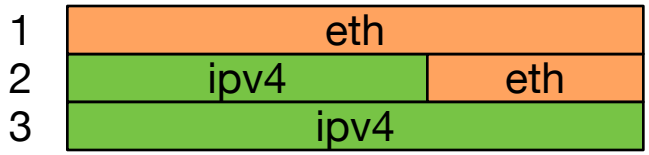


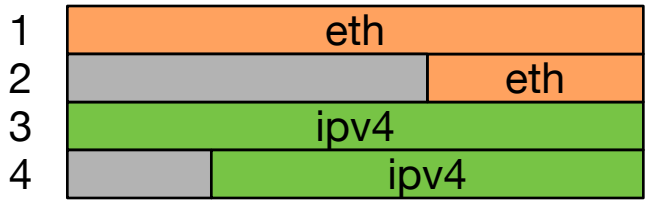




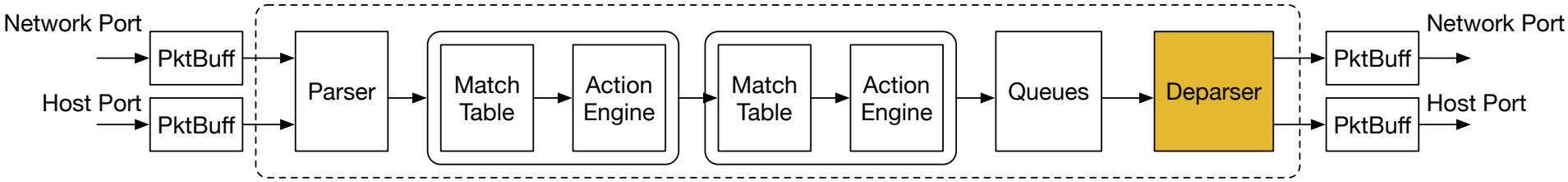
Deparser

- Apply metadata
- Remove invalid header

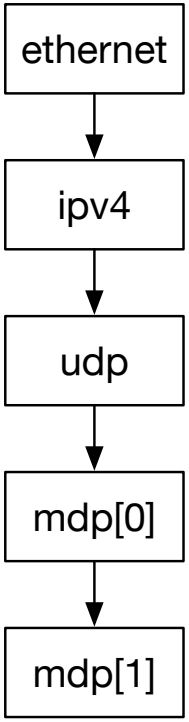





```
typedef tagged union {
    void Forward;
    void Delete;
    void Add;
} HeaderState;
```

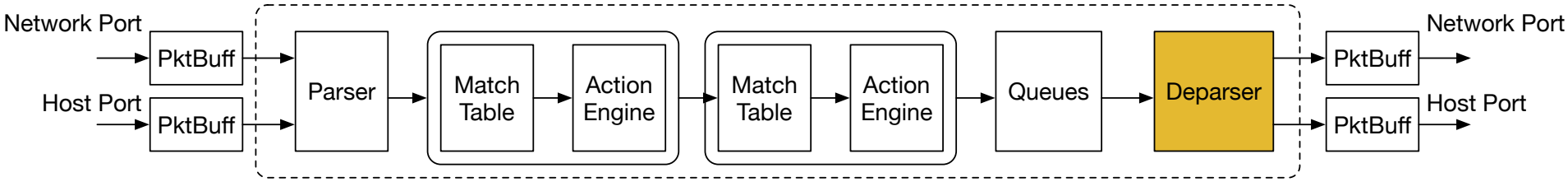


Deparser



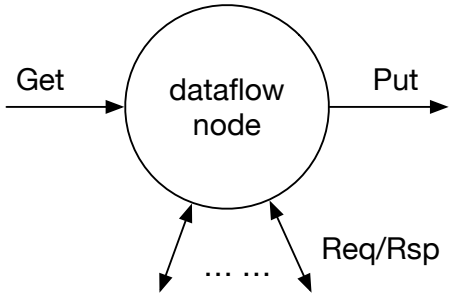

```
MetadataT {
    HeaderState ethernet;
    HeaderState ipv4;
    HeaderState udp;
    Vector(10, HeaderState) mdp;
}
```

ethernet: Forward, ipv4: Forward, udp: Forward,
map <V Forward, Forward, NotPresent, NotPresent …. >

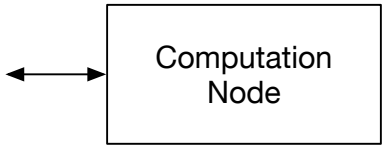


Control flow

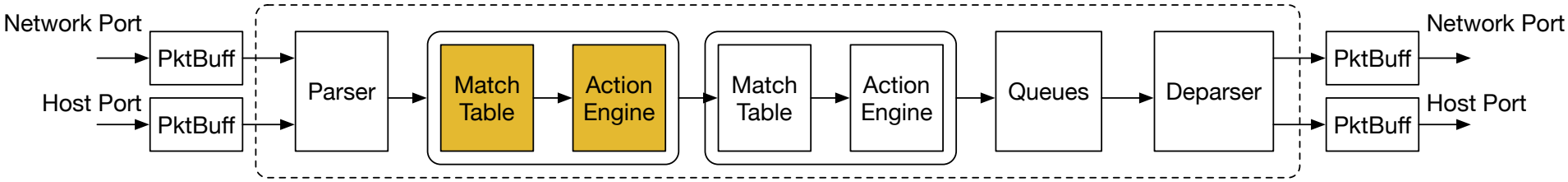
- Two types of nodes



dataflow node (table)

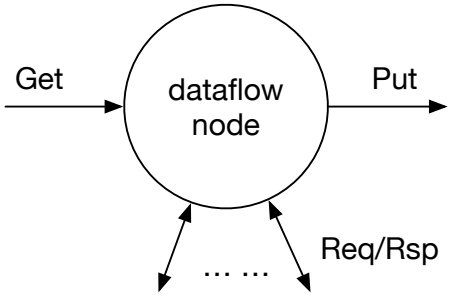


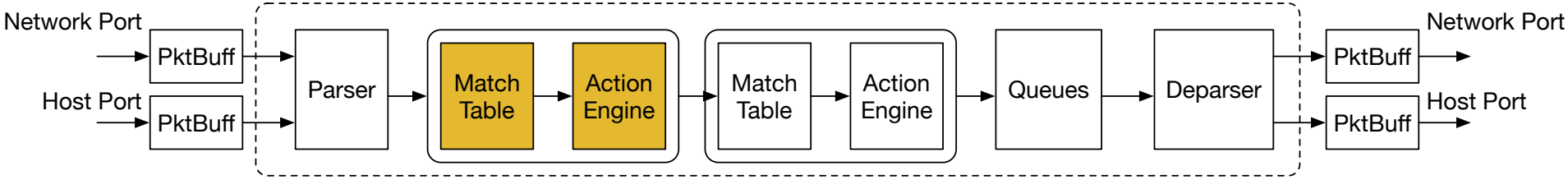
computation node (action/extern)



Control flow: Dataflow node

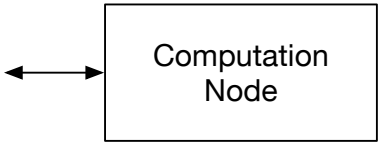
- Get/Put interface
  - One-way Push-only interface
  - MetadataT and PacketID
- Req/Rsp Interface
  - Two-way interface
  - Individual metadata and header field
  - Invariant: No. Request == No. Response
- Provided as a template



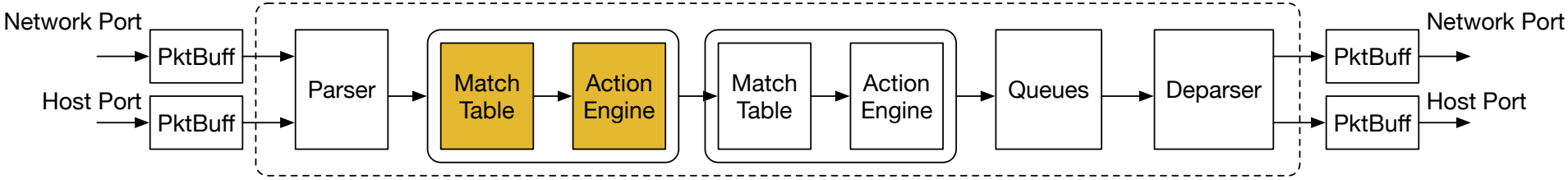


Primitive: Computation Node

- Req / Rsp Interface
- Latency insensitive

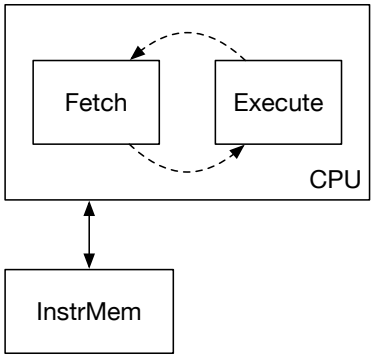


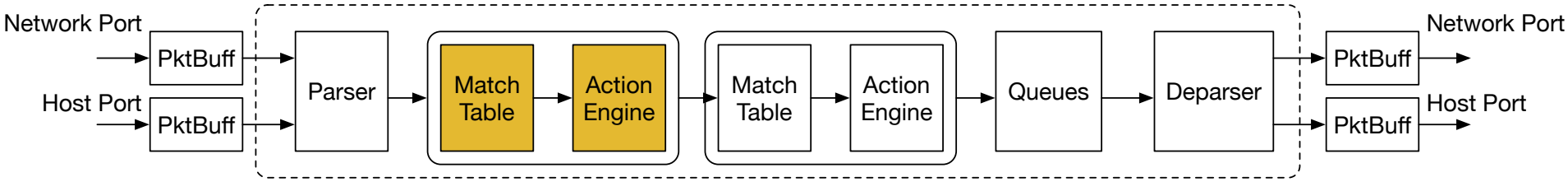
Code example?



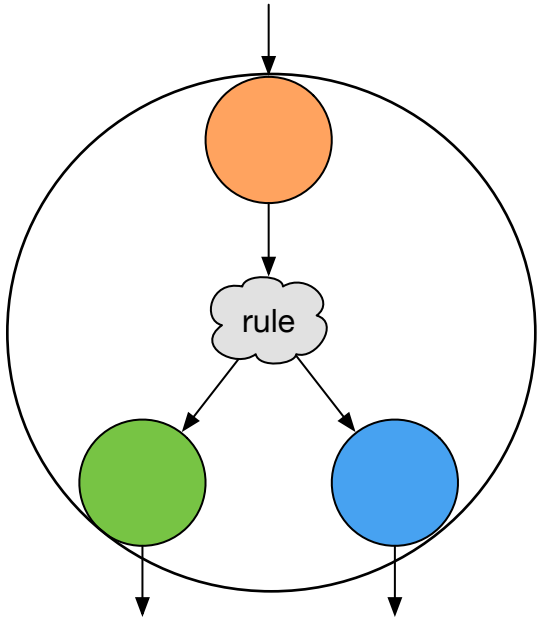
Computation Node Implementation

- Option:
  - Rich instruction set: Packet transaction (SIGCOMM'16)
  - Simple instruction set: P4FPGA
- P4FPGA implementation:
  - ALU to support arithmetic operation
  - RISC-V encoded instructions
- Access to externs are encoded as load/store
  - Register, Counter



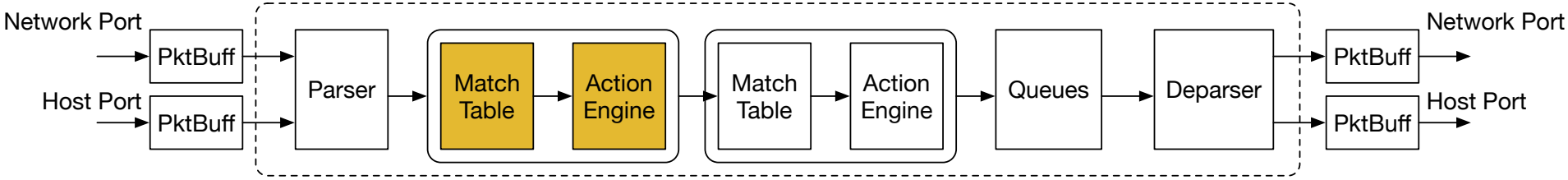


From dataflow node to control flow

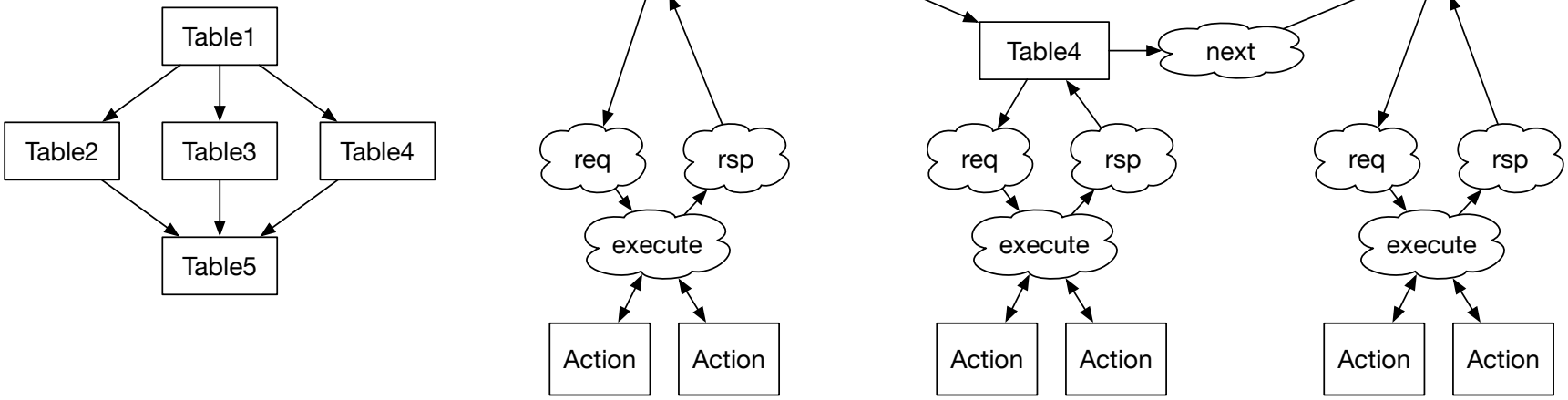


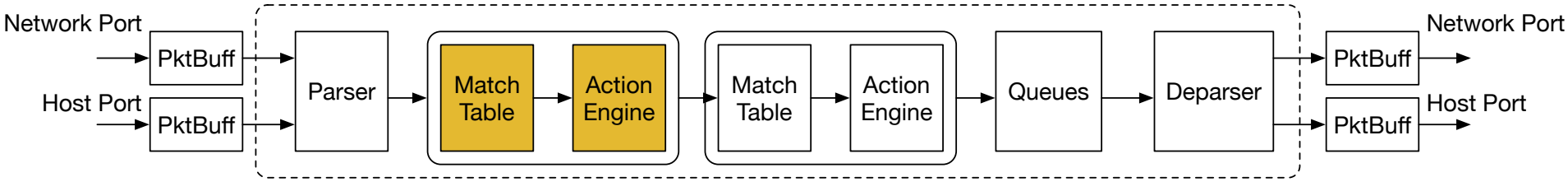

```
rule rl_ctrl_flow if (ORANGE.notEmpty);
   let token = ORANGE.get;
   case (token) matches
      tagged GOTO_GREEN
         GREE.enq(token);
      tagged GOTO_BLUE
         BLUE.enq(token);
    endcase
endrule
```

Rule only fires if ORANGE has a token to forward



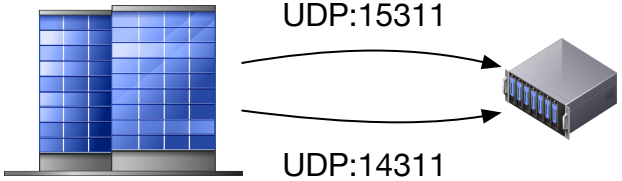
Put both together

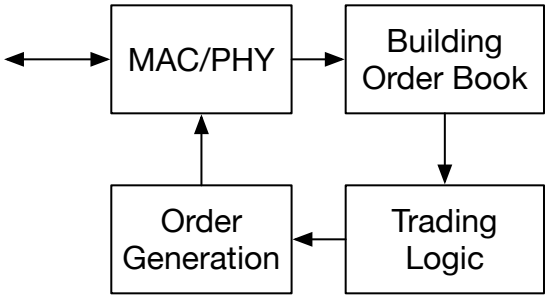




Application

- Financial Data Feed handler
  - Packet deduplication
  - Extern in P4-16
  - Extern assumes all API to be atomic?





But programming FPGA is hard!

- Long compilation time, NP-hard problem
- Verilog, VHDL?
- Timing closure? What is that?!

We build a flexible P4 backend and compiler for FPGA