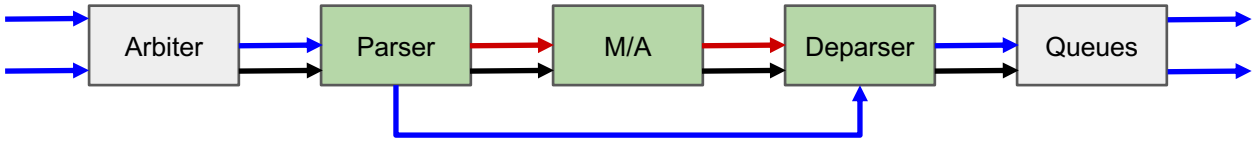
P4 Architectures and Data-Plane Events

Steve Ibanez
6/22/2020

Current P4 Architecture Descriptions



- Declaration of:
  - standard metadata
  - externs
  - P4 element interfaces
- Functionality mostly specified using English and diagrams

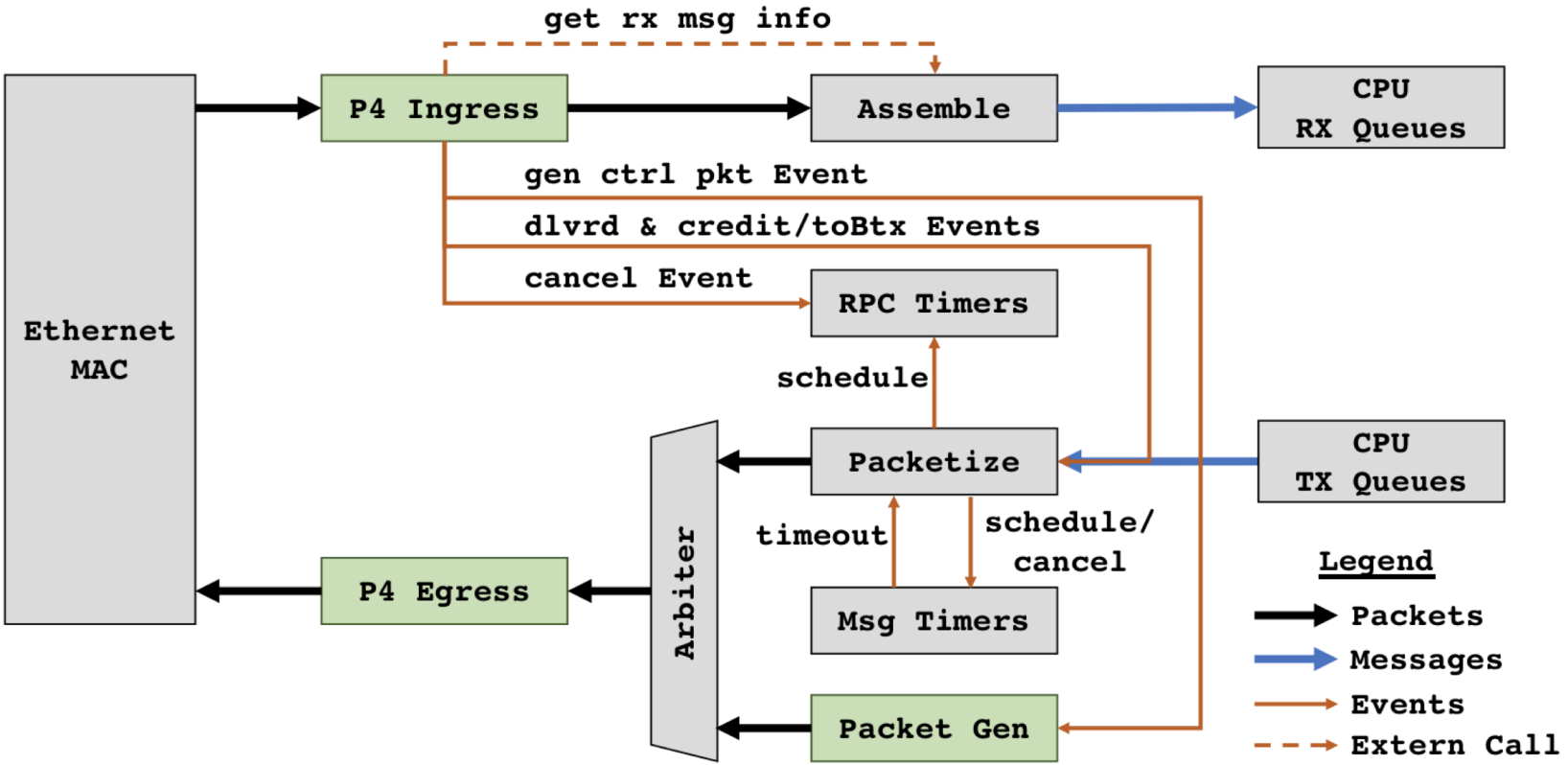
```
struct std_meta {...}

extern Checksum16 {...}

parser Parser<H>(packet_in          p,
                 inout std_meta_t   std_meta,
                 out H              headers);

control Pipe<H>(inout H             headers,
                inout std_meta_t    std_meta);

control Deparser<H>(in H                headers,
                    inout std_meta_t    std_meta,
                    packet_out          p);

package BasicSwitch<H> (Parser<H> p,
                        Pipe<H> map,
                        Deparser<H> d);
```

Data-Plane Events

- Architectures process more than just packets - they can process various data-plane events
- Data-plane events can trigger processing in either P4 or non-prog. blocks

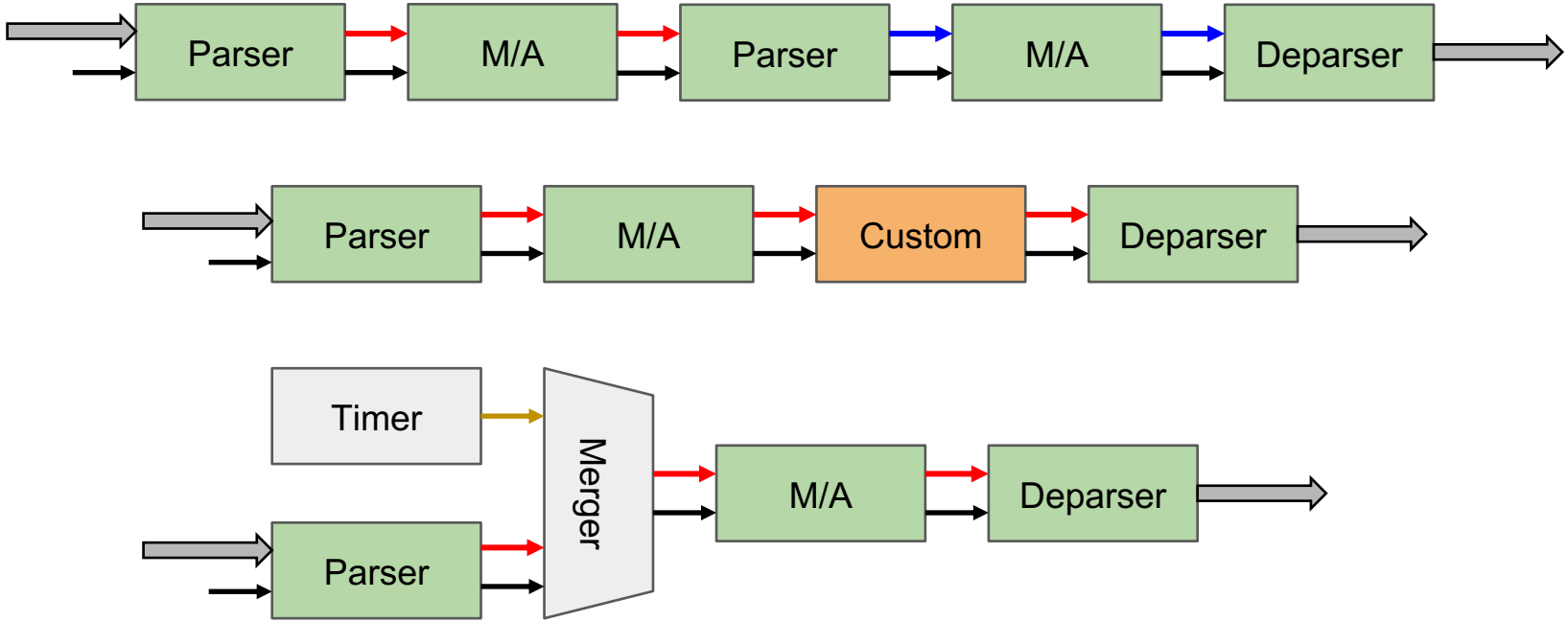
Data-Plane Events	
Ingress Packet	Buffer Overflow
Egress Packet	Buffer Underflow
Recirculated Packet	Timer Expiration
Generated Packet	Control-Plane triggered
Buffer Enqueue	Link status change
Buffer Dequeue	User Event

An Example NIC Architecture



The Case for Architecture Specification

- Enable custom generation of new architectures for flexible targets (e.g. FPGA software)

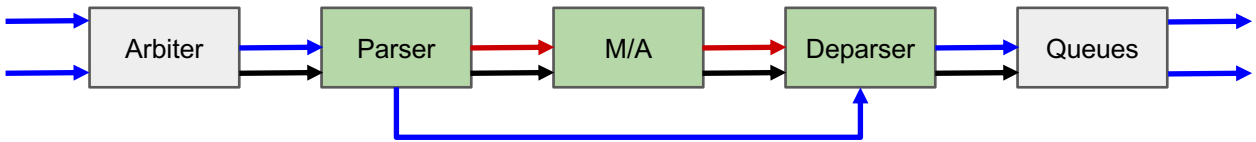


The Case for Architecture Specification

- Enable formal verification of data-plane programs
- P4++: a single program that fully specifies data-plane functionality
- End Goal:
  - Data-plane verification integrates with control-plane verification to enable end-to-end network verification

One Possibility - Verilog Inspiration

- Assume we have precise, well-understood models of non-programmable elements
- Precisely specify connectivity of architecture element interfaces
- Sufficient to automate process of building architectures



Verilog Inspiration

```
block Arbiter(packet_in   p1,
              packet_in   p2,
              packet_out  p_out,
              out std_meta_t meta);

parser Parser<H>(packet_in         p,
                 inout std_meta_t  meta,
                 out H             headers);

control Pipe<H>(inout H            headers,
                inout std_meta_t   meta);

control Deparser<H>(in H              headers,
                    inout std_meta_t  meta,
                    packet_out        p);

block Queues(in std_meta_t  meta,
             packet_in  p_in,
             packet_out p1,
             packet_out p2);
```

```
package BasicSwitch<H> (Parser<H> p,
                        Pipe<H> map,
                        Deparser<H> d) {

  packet_in p1_in;
  packet_in p2_in;
  packet_out p1_out;
  packet_out p2_out;

  Arbiter a;
  Queues q;

  main {
    a.p1 = p1_in;
    a.p2 = p2_in;
    p.p = a.p_out;
    p.meta = a.meta;
    map.headers = p.headers;
    map.meta = p.meta;
    d.headers = map.headers;
    d.meta = map.meta;
    q.meta = d.meta;
    q.p_in = d.p;
    p1_out = q.p1;
    p2_out = q.p2;
  }
}
```

Another Possibility - Event-Driven Description

- Specify architecture in terms of data-plane events that are supported and how they are processed by elements
- Make events a 1st class citizen of the language

Event-Driven Architecture Description

```
event PacketIn { packet_in p; }

event ArbiterDone {
  packet_in p;
  std_meta_t meta;
}

event ParserDone<H> {
  std_meta_t meta;
  H headers;
}

block Arbiter(packet_in   p1,
              packet_in   p2,
              packet_out  p_out,
              out std_meta_t meta) {
  ArbiterDone apply(PacketIn);
}

parser Parser<H>(packet_in         p,
                 inout std_meta_t  meta,
                 out H             headers) {
  ParserDone apply(ArbiterDone);
}
```

```
package BasicSwitch<H> (Parser<H> p,
                        Pipe<H> map,
                        Deparser<H> d) {

  Arbiter a;
  Queues q;

  main {
    @PacketIn     : a.apply();
    @ArbiterDone  : p.apply();
    @ParserDone   : map.apply();
    @MapDone      : d.apply();
    @DeparserDone : q.apply();
  }
}
```

Takeaways

- It is useful to have precise architecture specification in the language
- Keep in mind: architectures process not just packets, but data-plane events more generally
- Open question: What is the best way to specify architectures in the language?