P4 Expert Roundtable Series
April 28-29, 2020
Hosted by:
ONF

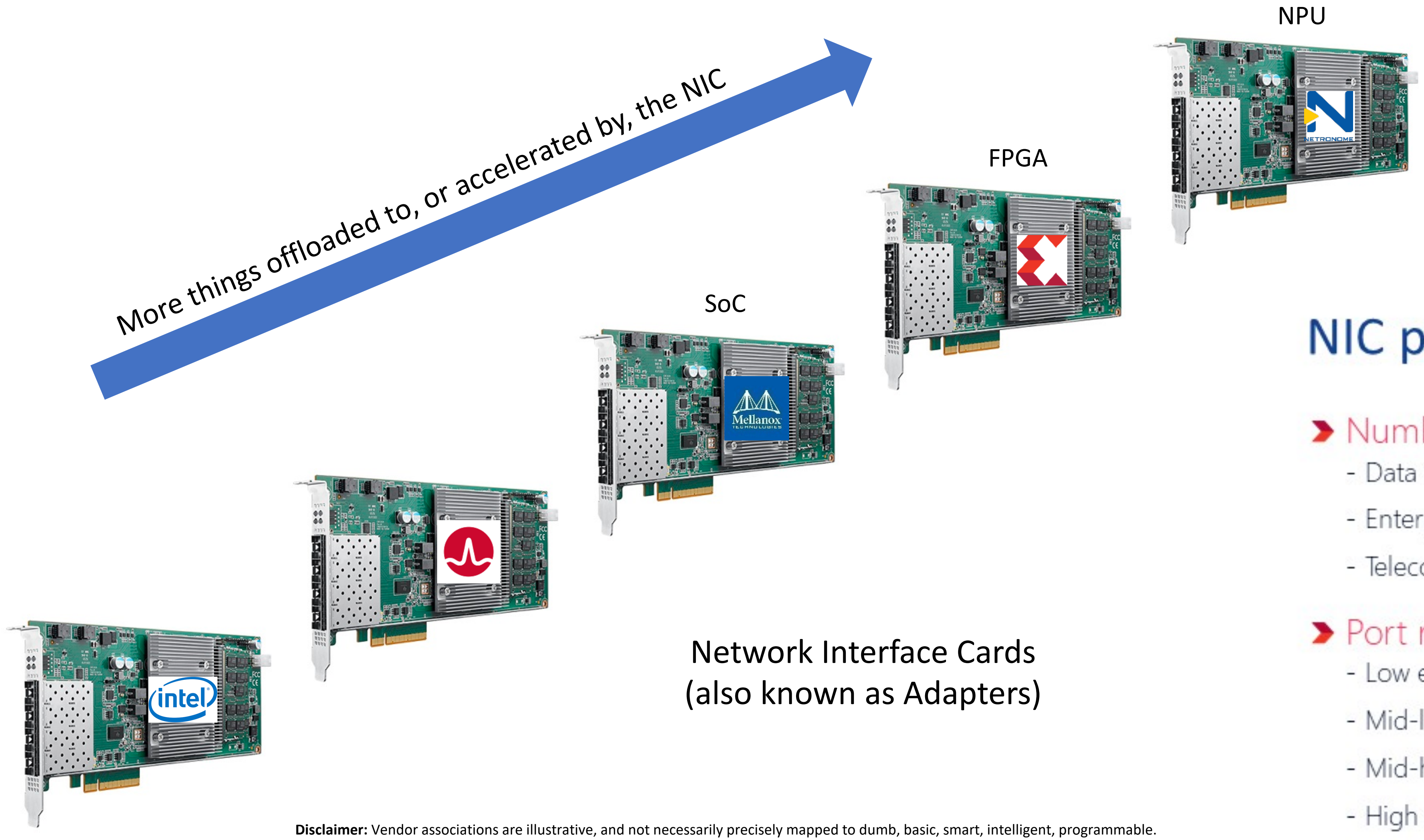
Sponsored By
Google intel STORDIS The Open Networking Expert

P4 Use Cases in Programmable NICs

Gordon Brebner
Fellow
Xilinx Labs

XILINX®

NICs: dumb, basic, smart, intelligent, programmable



Network Interface Cards
(also known as Adapters)

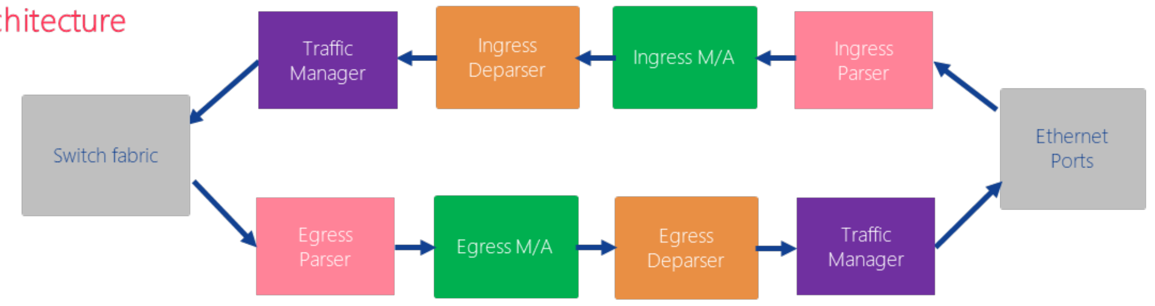
NIC parameters

- Number of ports
  - Data center, typically 1, maybe 2
  - Enterprise, typically 2, maybe 4
  - Telecommunications, typically 2
- Port rates (increasing over time)
  - Low end: 1G, 5G
  - Mid-low end: 10G, 25G
  - Mid-high end: 40G, 50G
  - High end: 100G, 200G

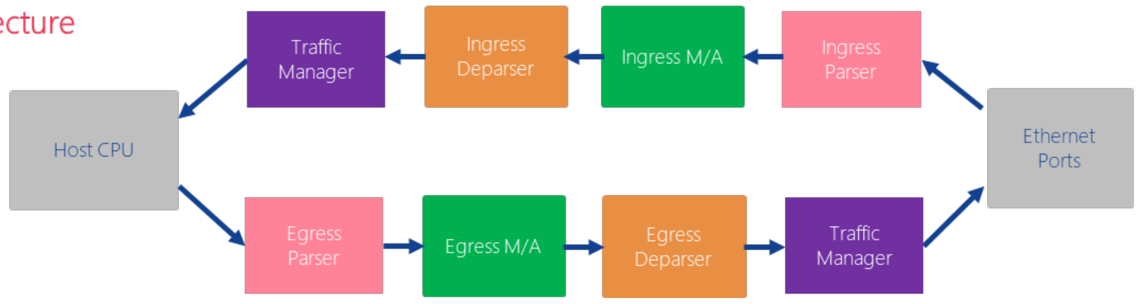
Disclaimer: Vendor associations are illustrative, and not necessarily precisely mapped to dumb, basic, smart, intelligent, programmable.

NICs vs. switches; NICs in endpoints and middleboxes

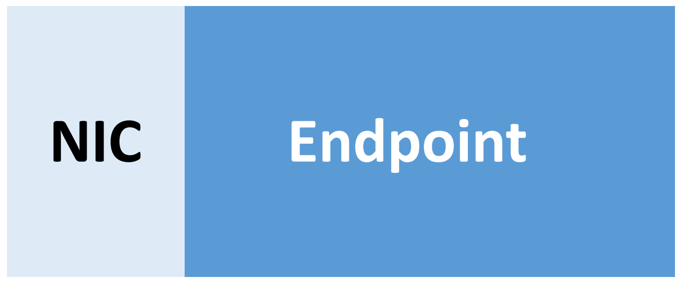
- Switch-style architecture

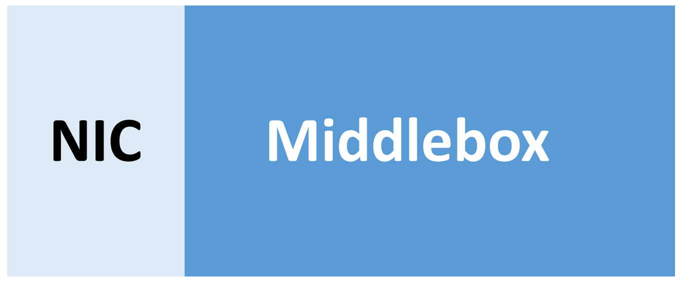


- NIC-style architecture



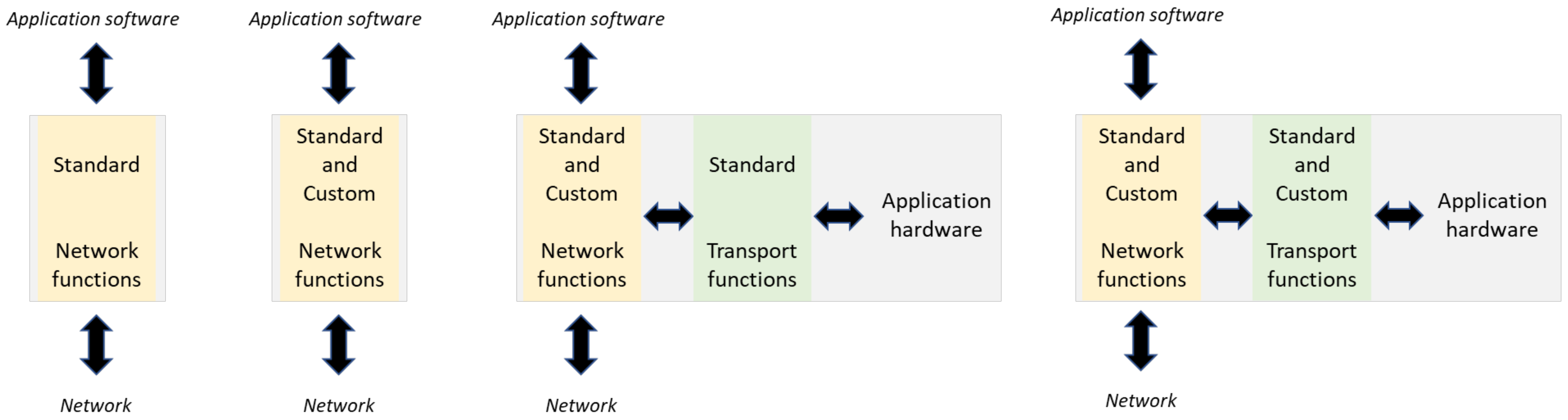
Look pretty similar P4 use cases:
It's the programmable data plane pipeline that matters





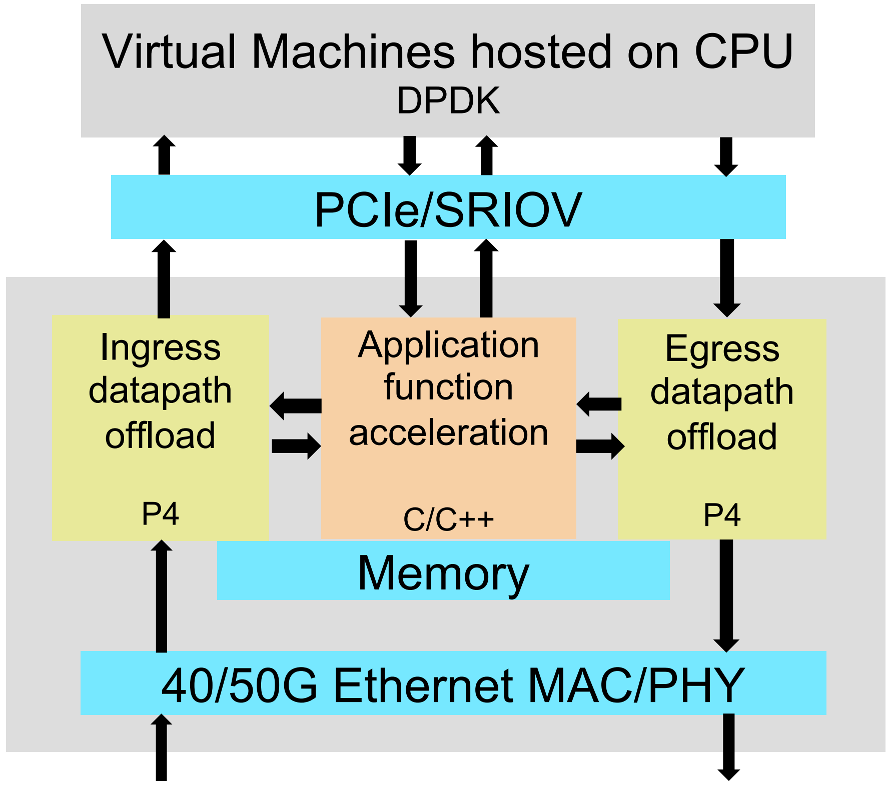
Look pretty similar: it's a NIC on a server

Xilinx Labs ‘Adaptive NIC’ taxonomy



Type 1 Type 2 Type 3 Type 4

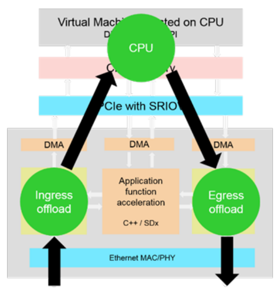
Xilinx Labs Type 2/3 NIC prototypes, 2015-2018

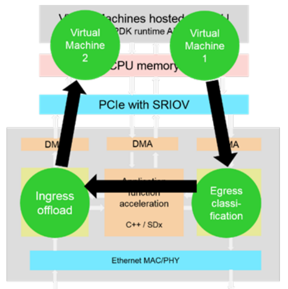


Later upgraded to 100G Ethernet

Smart NIC:

- Ingress/egress offload
- One CPU core instead of six
- Full line rate with 64-byte packets



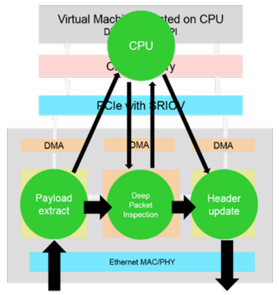


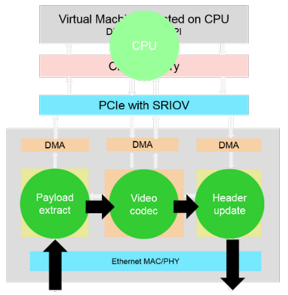
Virtual switching:

- vSwitch/vRouter/SFC acceleration
- 5x reduction in VM-to-VM latency
- Throughput = PCIe bandwidth

Deep packet inspection:

- Lookaside acceleration
- … only descriptors to/from CPU
- 100x more rules for DPI





Video transcoding:

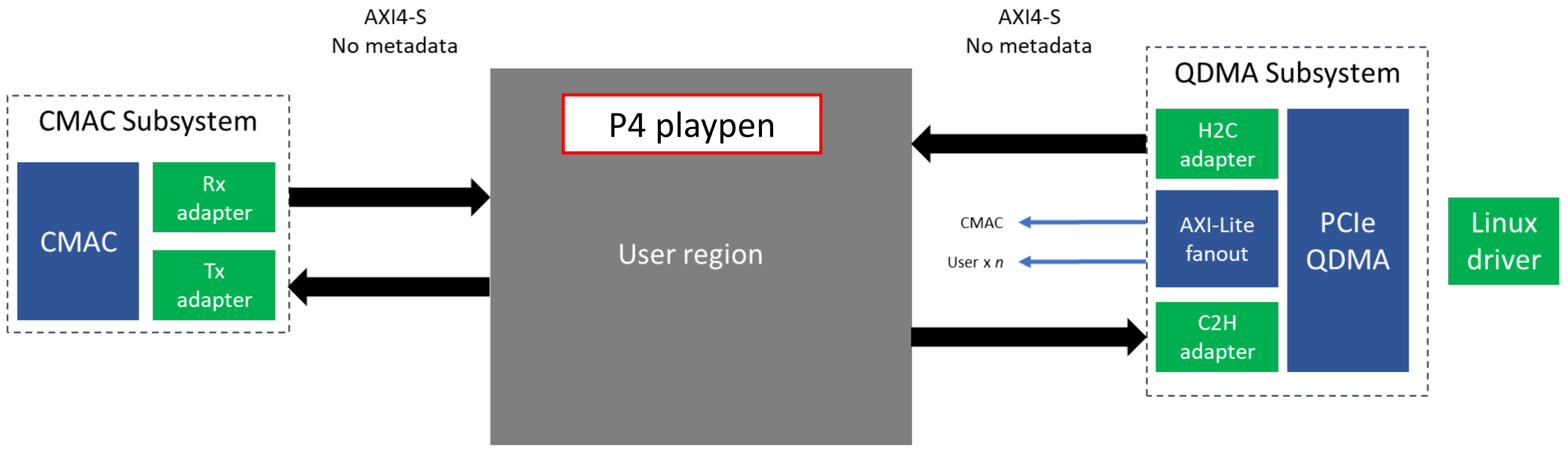
- Bump-in-wire acceleration
- … only exceptions to/from CPU
- 25x better frames/sec/W for video

Experimental use cases

Current playpen: Xilinx Labs supported base shell

🟦 Xilinx Vivado IP

🟩 Xilinx Labs open IP



2 x 100Gb/s

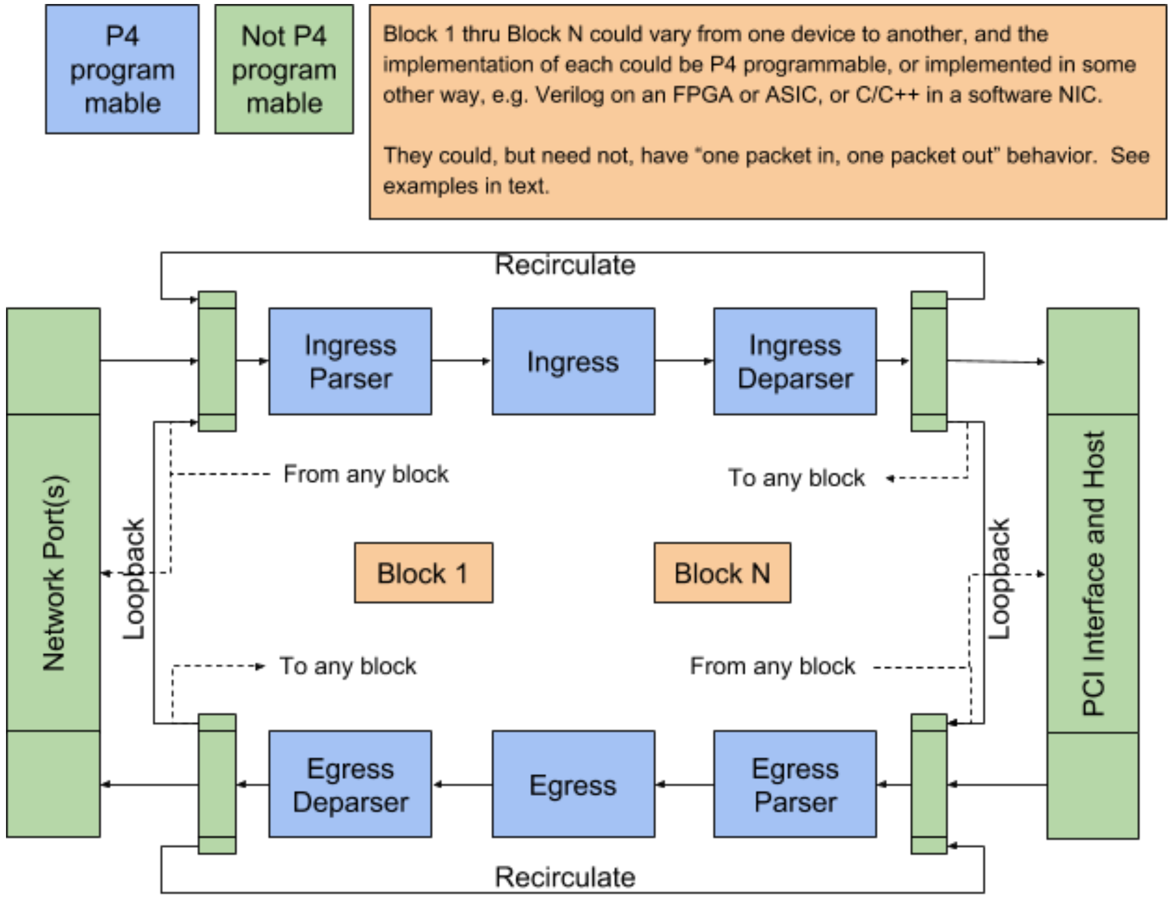
FPGA
SRAM, DRAM
PCIe Gen3 x 16

† See P4 Roundtable 2020 talk: "Towards an Open P4 Programmable Hardware Platform"

P4 Portable NIC Architecture (PNA) project

- Some issues being considered:
  - There are ingress and egress pipelines
    - What are the standard components of each pipeline?
      - Which are P4 programmable?
    - Is more variability needed than in PSA?
      - What about in-line externs, particularly fixed-function NIC blocks?
      - What lookaside externs might be needed?
    - Is direct interaction between ingress and egress allowed?
    - What about isolation/virtualization?
    - Should the collection of network ports be modelled explicitly?
    - Is there a switch within the NIC?
  - How is host CPU interface modelled?
    - Differentiate data plane CPU, and control plane CPU, roles
    - Impact on P4Runtime
  - Beyond packet forwarding:
    - Is protocol (e.g., TCP) termination covered?
    - Is 'Type 3' NIC covered – payload processing as well as forwarding?
  - Aiming for modular spec that has as much in common with PSA as possible
  - Wondering what extensions to P4 might be needed

- Current P4.org architecture sub-group
  - Active helpers wanted …


P4 programmable
Not P4 programmable
Block 1 thru Block N could vary from one device to another, and the implementation of each could be P4 programmable, or implemented in some other way, e.g. Verilog on an FPGA or ASIC, or C/C++ in a software NIC.
They could, but need not, have "one packet in, one packet out" behavior. See examples in text.
Recirculate
Network Port(s)
Loopback
Ingress Parser
Ingress
Ingress Deparser
From any block
To any block
Block 1
Block N
To any block
From any block
Egress Deparser
Egress
Egress Parser
Loopback
PCI Interface and Host
Recirculate

One initial proposal

P4
Expert
Roundtable Series
April 28-29, 2020

Hosted by:
ONF

Sponsored By

Google intel STORDIS The Open Networking Expert

Thank You

Contact info for further questions:
Gordon.Brebner@Xilinx.com

XILINX®

P4 Expert Roundtable Series

April 28-29, 2020

Hosted by: ONF

Sponsored By

Google intel STORDIS The Open Networking Expert

PENSANDO

~~P4~~ Use Cases (in Programmable NICs) for the Evolution of P4 and PSA

Mario Baldi
Distinguished Technologist

John Cruz
Member of Technical Staff

Pensando Systems, Inc.

What Are We Going to Talk About?

An overview of the Distributed Services Card

- Our P4 playground

Two use cases

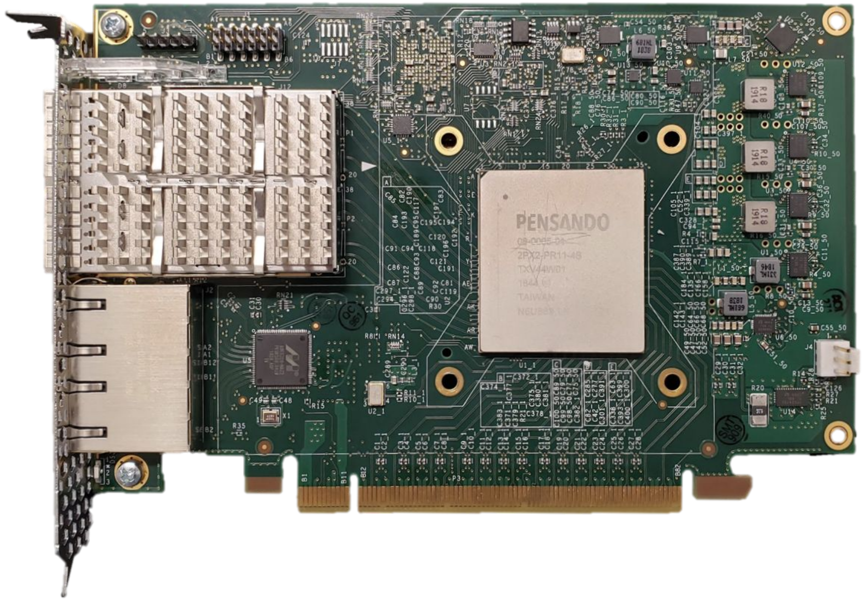
- TCP connection tracking
  - Benefits of P4 writable tables
- Generic Segmentation Offload (GRO) and Large Receive Offload (LRO)
  - Value of a PSA variant

PENSANDO

The Distributed Services Card (DSC)

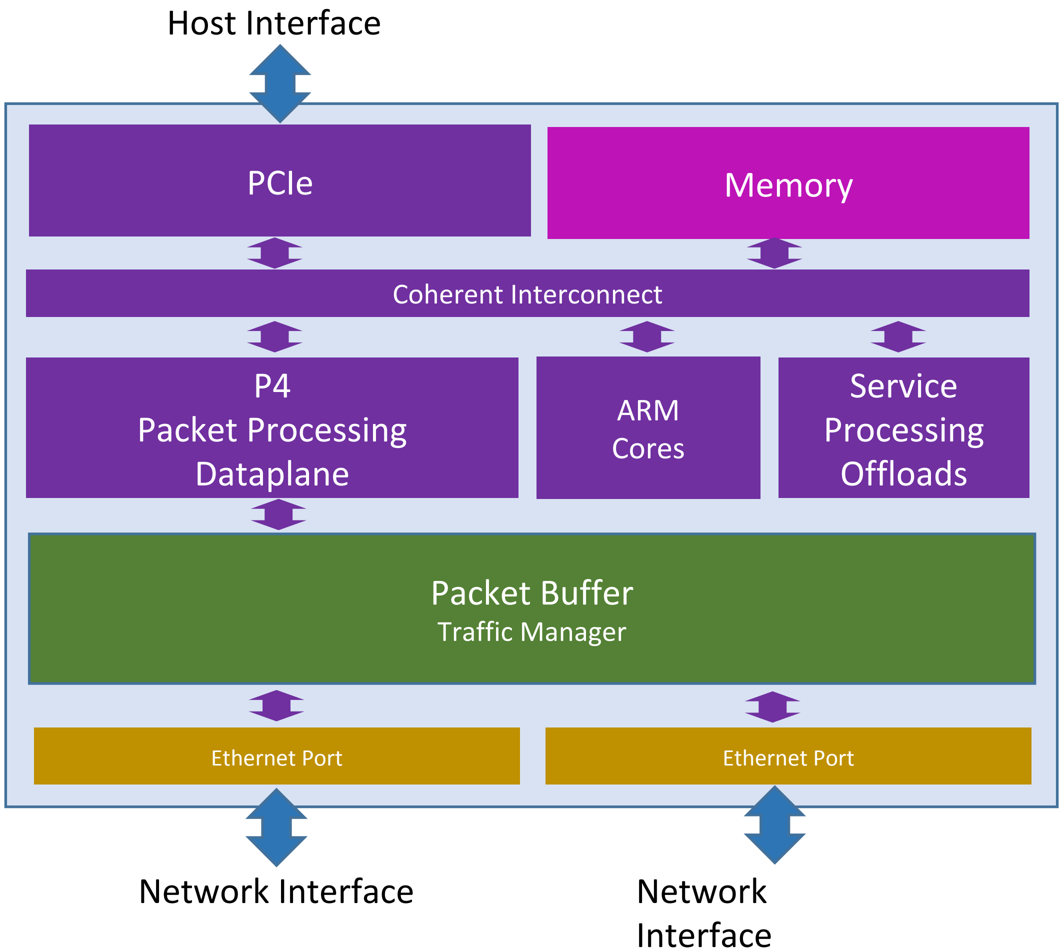
PCIe Programmable Platform

- 2 x 25Gb/s Ethernet ports
- 2 x 100Gb/s Ethernet ports



DSC Programmable Processor Architecture

A more detailed explanation in the P4 Expert Roundtable Series talk "Programmable Data Plane Architecture for the Network Edge"



PENSANDO

Use Case 1

A case for P4 enhancements

PENSANDO

TCP Connection Tracking

Reconstruct the state of a TCP connection by observing packets

- Make sure the end points are in a legitimate state
- Security purposes - protect against injection of packets that are not consistent with the state
  - E.g., wrong sequence numbers
- Stateful operation
  - Connection state needs to be kept
  - Process header fields and update connection state accordingly

TCP Session State Table

The state will be updated dynamically

```
table session_state {
        key = {
            ...
            metadata.flow_lkp.lkp_src    :  exact;
            metadata.flow_lkp.lkp_dst    :  exact;
            metadata.flow_lkp.lkp_proto  :  exact;
            metadata.flow_lkp.lkp_sport  :  exact;
            metadata.flow_lkp.lkp_dport  :  exact;
        }
        actions = {
            nop;
            tcp_session_state_info;
        }
        default_action = nop;
        size = FLOW_STATE_TABLE_SIZE;
       ...
    }
```

PENSANDO

Table Entry Update within the P4 Data Plane

Table fields (action parameters) can be declared as writable

```
action tcp_session_state_info(
        @__ref bit<32>      iflow_tcp_seq_num,
        @__ref bit<32>      iflow_tcp_ack_num,
        @__ref bit<16>      iflow_tcp_win_sz,
        @__ref bit<4>       iflow_tcp_win_scale,
        ...
        bit<32>             syn_cookie_delta,
        bit<14>             rflow_exceptions_seen,
        bit<1>              tcp_sack_perm_option_negotiated) {
    ...
}
```

PENSANDO

TCP Session State Update - a P4 Extension

A table field (action parameter) can be on the left side of an assignment operator

```
action tcp_session_state_info(
        @__ref bit<32>      iflow_tcp_ack_num,
        @__ref bit<16>      iflow_tcp_win_sz,
        bit<32>             syn_cookie_delta,
        ...
        bit<1>              tcp_sack_perm_option_negotiated) {
    ...
    if (CMP_SEQNUM_LE32(iflow_tcp_ack_num, hdr.l4_u.tcp.ackNo) &&
        CMP_SEQNUM_LE32(hdr.l4_u.tcp.ackNo, rflow_tcp_seq_num)) {
        iflow_tcp_ack_num  = hdr.l4_u.tcp.ackNo;
        iflow_tcp_win_sz   = hdr.l4_u.tcp.window;
    }
    ...
}
```

In Standard P4

Registers must be used for state that can be updated from within the data plane

```
typedef bit<32> SessionIndx_t;

Register<bit<32>, SessionIndx_t>(FLOW_STATE_TABLE_SIZE) iflow_tcp_seq_num;
Register<bit<32>, SessionIndx_t>(FLOW_STATE_TABLE_SIZE) iflow_tcp_ack_num;
Register<bit<16>, SessionIndx_t>(FLOW_STATE_TABLE_SIZE) iflow_tcp_win_sz;
Register<bit<4>,  SessionIndx_t>(FLOW_STATE_TABLE_SIZE)
iflow_tcp_win_scale;
Register<bit<32>, SessionIndx_t>(FLOW_STATE_TABLE_SIZE) rflow_tcp_seq_num;

...
```

Session table seems unchanged ...

```
table session_info {
        key = {
            ...
             metadata.flow_lkp.lkp_src    :  exact;
             metadata.flow_lkp.lkp_dst    :  exact;
             metadata.flow_lkp.lkp_proto  :  exact;
             metadata.flow_lkp.lkp_sport  :  exact;
             metadata.flow_lkp.lkp_dport  :  exact;

        }
        actions  = {
            nop;
            tcp_session_state_info;
        }
        default_action  = nop;
        size  = FLOW_STATE_TABLE_SIZE;
        ...
    }
}
```

PENSANDO

But it contains only static data and index mapping

```
action tcp_session_state_info(
        bit<32>   syn_cookie_delta,
        ...
        bit<1>    tcp_sack_perm_option_negotiated),
        bit<16>    session_index {
    ...
    bit<32> t_iflow_ack_num = iflow_tcp_ack_num.read (session_index);
    bit<32> t_rflow_seq_num = rflow_tcp_seq_num.read(session_index);
    if (CMP_SEQNUM_LE32(t_iflow_ack_num, hdr.l4_u.tcp.ackNo)  &&
        CMP_SEQNUM_LE32(hdr.l4_u.tcp.ackNo, t_rflow_seq_num)) {
            iflow_tcp_ack_num.write (session_index,
        hdr.l4_u.tcp.ackNo);
            iflow_tcp_win_sz.write (session_index,
        hdr.l4_u.tcp.window);
    }
    ...
}
```

Register vs. Writable Table Comparison

More cumbersome

Less legible

- Code less more compact
- All relevant and related information not in one place

Less efficient memory allocation

- Might be difficult to optimize allocation with separate registers
- Table is one memory block vs several allocations for registers
- Each register has discrete width

PENSANDO

Writable Table Implementation Challenges

Ensure that reads after writes are not getting stale data

- Performance and complexity
- Specialized hardware to propagate the writes

Coherence across parallel pipelines and multiple stages

T-CAM cannot be used

Registers work around this through constraints

- Specific hardware devices, not general memory
- Limited size
- Limited parallelism
- Limited scope

PENSANDO

Use Case 2

A case for a new architecture

PENSANDO

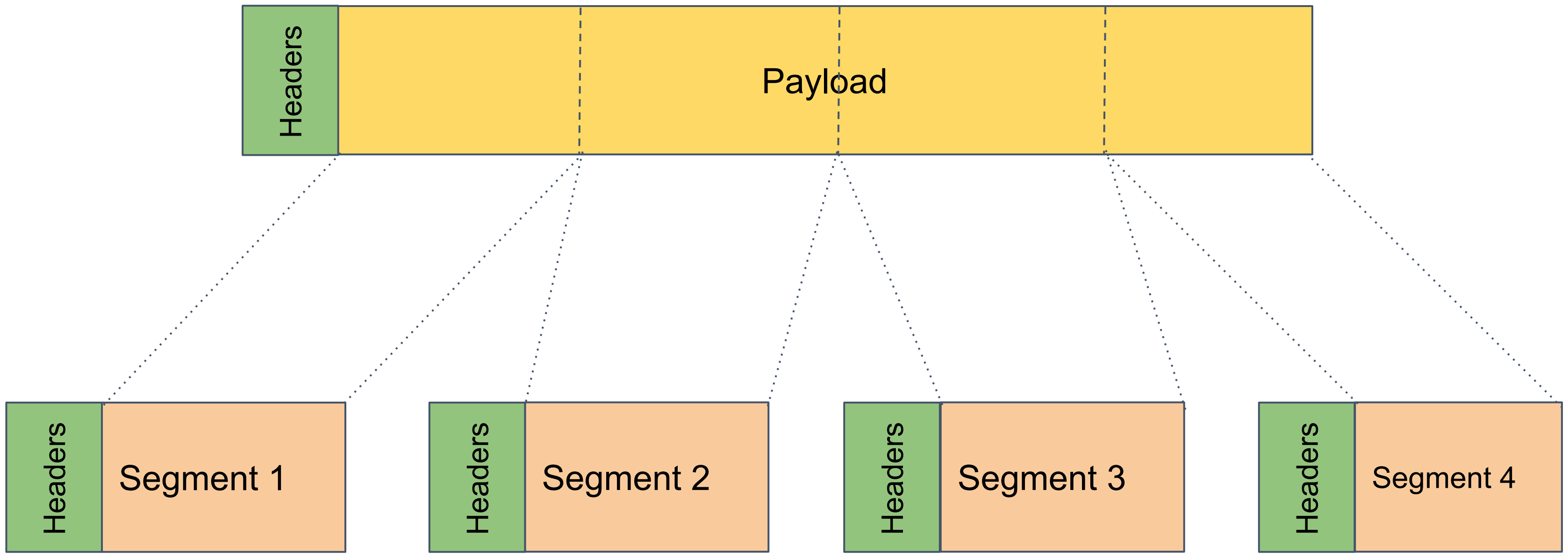
Use Case 2a: Generic Segmentation Offload (GSO)

Host hands over a large buffer or a collection of buffers to the DSC in a single transmit request

DSC performs the offload using the following operations

- Splits the data portion of the packet into smaller segments (say 1460 bytes)
- Adds the header portion of the packet to each segment
- Updates IP and L4 lengths
- Update IP identifier, TCP sequence number, and TCP flags
- Update IP and L4 checksums

Use Case 2a: Generic Segmentation Offload (GSO)



PENSANDO

GSO Pseudocode (Host Functionality)

Input : TX ring with pointers to {hdr_addr, hdr_sz, data_addr, data_sz}

```
● While TX ring is not empty:
    ○ read next buffer
    ○ data_left = buffer.data_sz
    ○ data_ptr = buffer.data_addr
    ○ while (data_left > 0):
        ■ this_segment_size = MAX(data_left, SEGMENT_SIZE)
        ■ DMA [buffer.hdr_addr, buffer.hdr_sz] +  [buffer.data_addr,
          this_segment_size]
        ■ data_left -= this_segment_size
```

Note that there is no parsing or deparsing

PENSANDO

GSO Pseudocode (Network Functionality)

Input : Packet

- Parse the packet to get individual packet headers
- For each layer (tunnel)
    - Update IP identifier, IP length, IP checksum
    - Update sequence number (if TCP), L4 length (if UDP), L4 checksum

GSO (Host Functionality)

```
control tx_pkt {
    tx_ring.apply();
    if (metadata.tx_ring_done == 0) {
        tx_buffer.apply();
    }
}
```

PENSANDO

GSO (Host Functionality)

```
table tx_ring {
  key = {
    metadata.tx_ring_addr : exact;
  }
  actions = {
    process_tx_ring;
  }
}
```

```
action process_tx_ring(bit<32> p_index,
@@__ref bit<32> c_index) {
    if (c_index == pindex) {
        metadata.tx_ring_done = 1;
        ring_doorbell();
    } else {
        c_index = (c_index + 1) % RING_SZ;
    }
}
```

PENSANDO

GSO (Host Functionality)

```
table tx_buffer {

  key = {

    metadata.c_index : exact;

  }

  actions = {

    process_tx_buffer;

  }

}
```

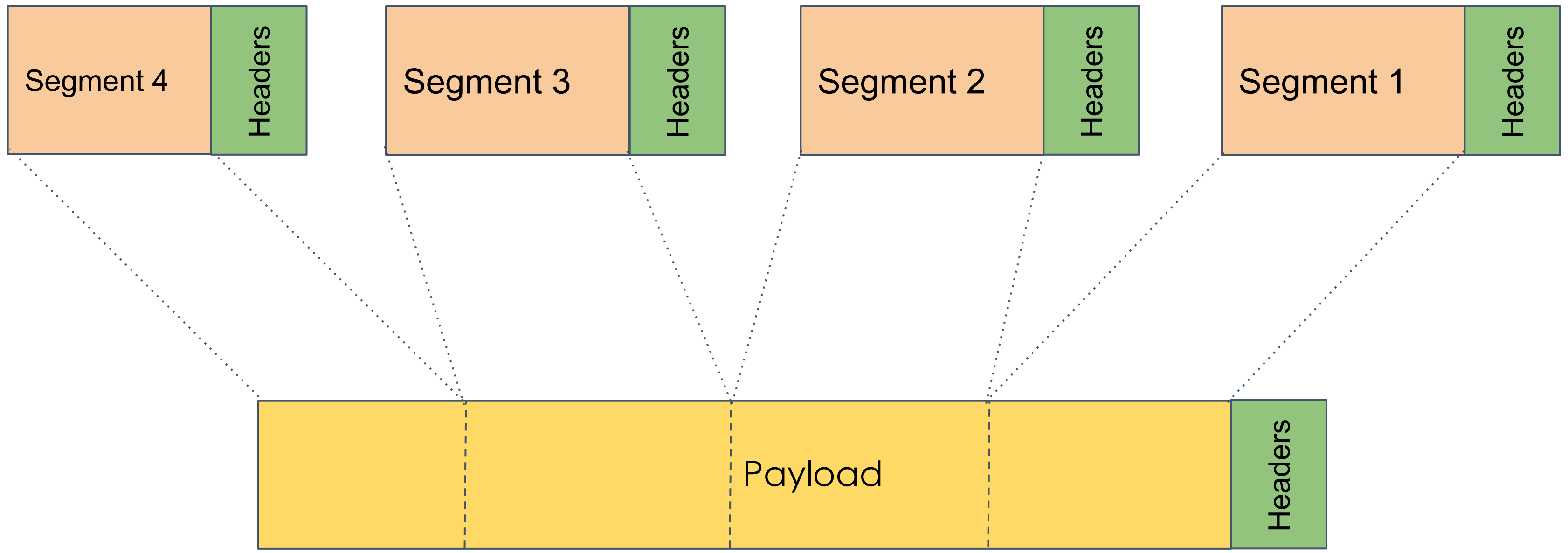
```
action process_tx_buffer(
        bit<64> hdr_addr,bit<16> hdr_sz,
        bit<64> data_addr, bit<32> data_sz) {

     bit<32> data_remain = data_sz;

     bit<32> data_ptr = data_addr;

     bit<32> seg_sz;

     while (data_remain > 0) {

         seg_sz = MAX(data_remain, MAX_SEGMENT_SZ);

         dma(hdr_addr, hdr_sz, data_ptr, seg_sz);

         data_ptr += seg_sz;

         data_remain -= seg_sz;

     }

}
```

Use Case 2b: Large Receive Offload (LRO)

Performed on packet reception (opposite of GSO)

- Identify consecutive packets that belong to the same TCP connection
- Append payload data to LRO buffer
    - for TCP, update sequence number in header
- If buffer is full, notify host of new packet
- On timer expiry
    - If LRO buffer is not empty, notify host of new packet

Use Case 2b: Large Receive Offload (LRO)



New Constructs Required

- Doorbell
  - A register the driver can update to start a P4 program
  - A register a P4 program can update to notify completion
- DMA
  - GSO
    - Copy header and segments to form multiple packets
  - LRO
    - Copy payload from multiple packets to form a larger message
- Timers
- Stateful Memory
  - To keep track of state
  - GSO : last segment offloaded; LRO : last segment received

Doorbell

- P4 programs are traditionally triggered by an incoming packet on an interface
- Need capability to trigger P4 programs by driver after placing data in the TX descriptor ring
- Need capability to notify driver after appending an entry to the completion queue or RX descriptor ring

Required constructs:

- Start P4 program execution on doorbell
- Extern to ring doorbell

DMA

Required constructs:

- Extern function(s) to perform different DMA operations
    - Memory to Packet (for transmission)
        - Support for gather
    - Packet to Memory (on reception)
        - Support for scatter
    - Memory to Memory
        - Offloads like encryption, compression
    - Metadata to Memory
        - Update TX and RX descriptor rings

Timers

Required Constructs:

- Extern to start a timer
- Start P4 program execution on timer expiry

PENSANDO

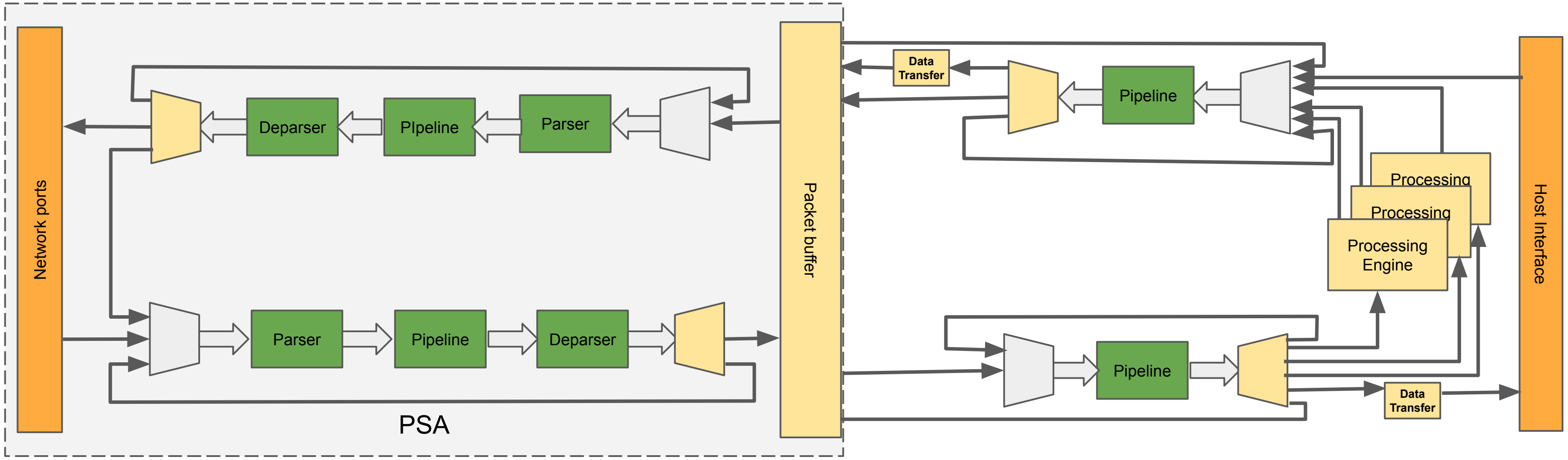
To conclude: P4 for Host-Attached Devices?

Yes, but with an architecture that is an evolution of the PSA

- Parsing is optional
  - Different trigger than packet arrival to start the processing
  - E.g., Timers, doorbells
- Preparation of multi packet data
  - On card memory possibly required
- Access to memory
  - For on-card memory, the table construct might be used
- Memory transfer capability
  - E.g., DMA

PENSANDO

A Possibile (Portable NIC) Architecture



🟩 P4 programmable 🟨 Metadata controllable/configurable

PENSANDO

Thank You

baldi@pensando.io - jcruz@pensando.io

www.pensando.io
blog.pensando.io