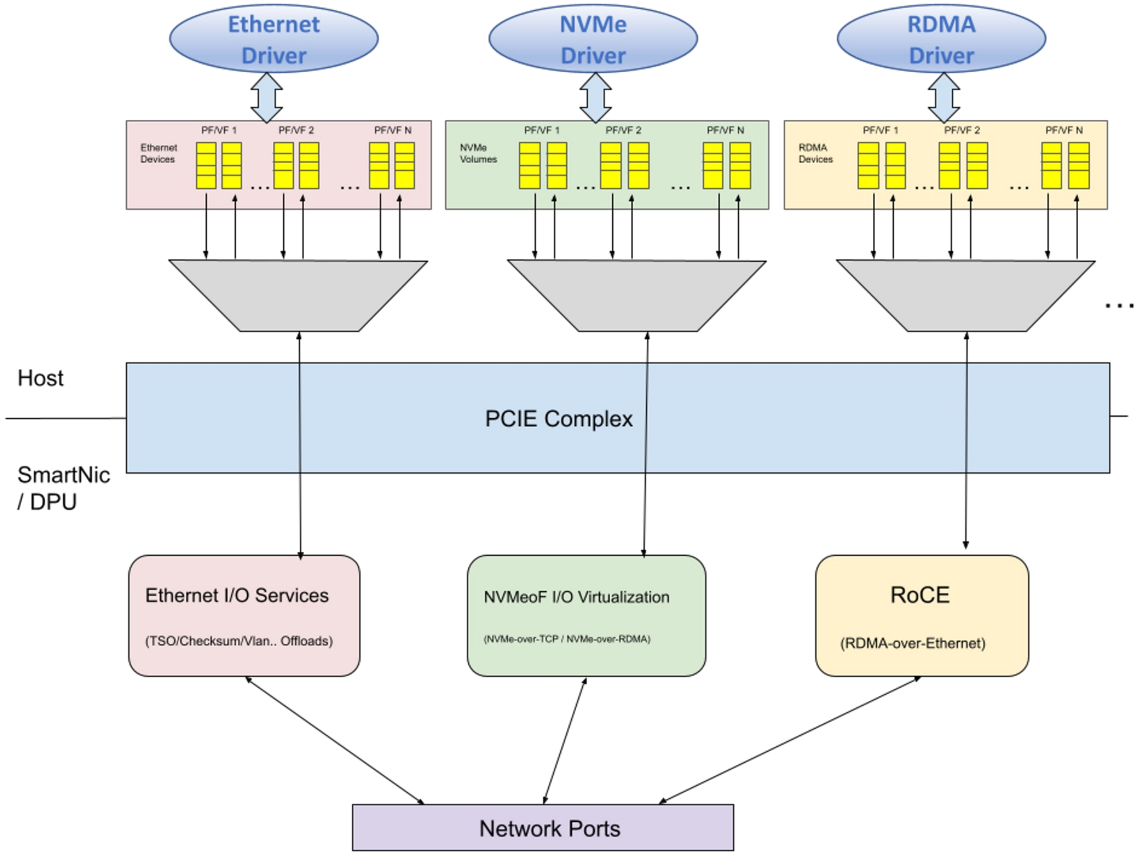
P4 at the Interface between NIC and Host

Raghava Sivaramu
Fellow, Pensando

Agenda

- Host interface functions
- Programmability at the host interface
- Fundamental design considerations
- Language / Compiler extensions
- Example Host I/O call-flow

Host Interface Functions



- Communication between the SmartNIC and the host
  - Networking: Typical Ethernet I/O NIC functions
    - Includes classic offloads - Checksum / Vlan / TSO / GRO / RSS
  - Storage: NVMe I/O functions for Block storage virtualization
    - NVMe over fabrics virtualization (NVMe over TCP / RDMA)
  - Remote DMA function for applications
    - HPC / Storage
  - Crypto-Dev (DPDK)
- Smart services for host applications
  - Switching/Routing
  - IPsec encryption
  - L4-L7 services (e.g., TLS offload)

Programmability at Host Interface - Why?

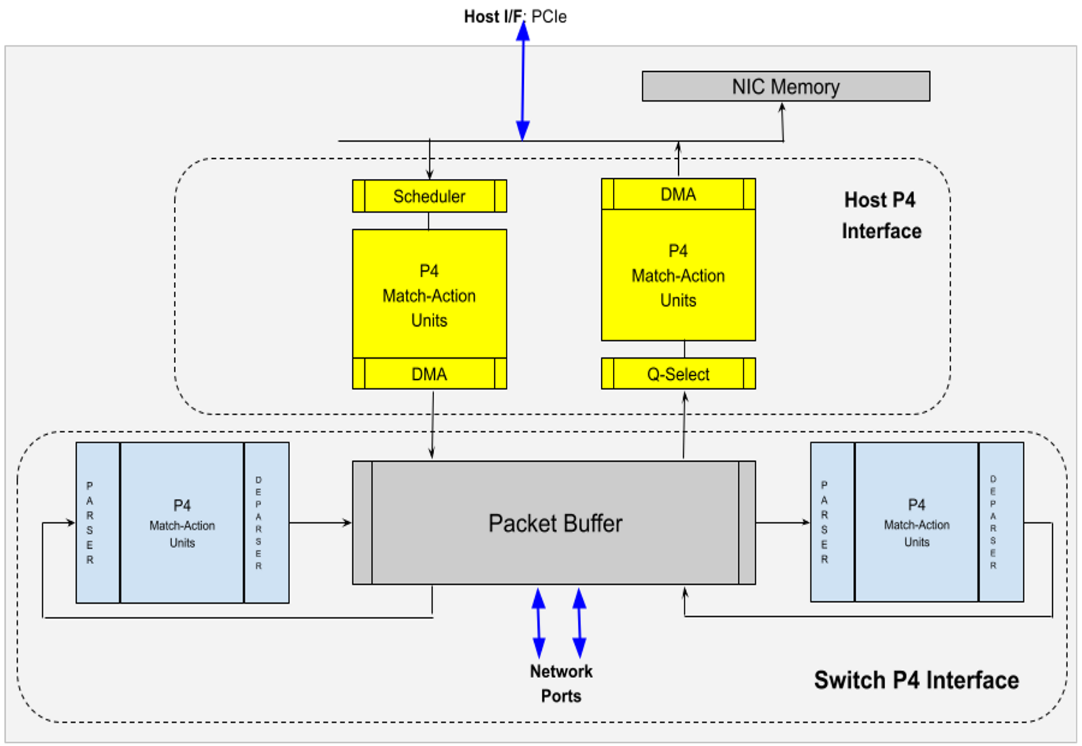
- Fixed function logic: Lack of flexibility/future proofing with changing interface requirements
  - Different I/O descriptor formats - Classic Ethernet/Virtio/VDPA/VMXNet3/UPT ..
  - Offload advanced services – NVME-over-Fabrics/Encryption/Compression ..
- General purpose CPU: Performance issues for
  - High Packets/Connections per-second needs
  - Large stateful DB processing/policy evaluations

↓

Need for specialized instruction-set processors

Programmability at the Host interface - How?

- Leverage P4-programmable switch HW architecture
    - Pipelined and multi processing paradigm
- Extend execution model beyond packet-oriented processing
    - Match-action units act on data in memory
    - DMA engines to transfer data from/to Host/NIC memory
    - Message processing / On demand scheduling of work
- Minimal P4 language extensions beneficial
    - Constructs needed for advanced programming: loops, switch-case
- P4 compiler backend extensions
    - Externs/Annotations/Table-properties support in P4 used significantly for additional capabilities
    - Support of architecture specific HW constructs - DMA engines, schedulers, timers, semaphores



Enables high-performance/line-rate services to host traffic

Fundamental design considerations

- *Event-based* triggers in addition to packet-based triggers
  - Message/interrupt events for host driver interactions
  - Classification and on-demand/timer based processing of messages
- *Stateful processing*
  - Associate and maintain context across packets/messages at various granularity
    - Host devices / Interfaces / Queues
    - Flows / Connection states
    - Ex: NVMe volume <-> NVMeoF q-pair <-> TCP-Connection context-block
  - Protocol state-machines - like TCP flow-control / congestion-control

Fundamental design considerations

- *Complex data processing,* not just packet-header manipulation
  - Units of data dealt with are not just packets, read/write data from memory
  - Memory can be in host/NIC
  - Manipulate in-memory data-structures, with atomic read-modify-write capabilities
  - Need DMA capability for memory<->packet/memory<->memory transfer
  - One or more events/packets as result of an event/packet processing
- *Code maintainability,* extend language as needed for advanced P4 programming
  - Conducive to implementations of
    - stateful TCP / RDMA protocols
    - higher-level applications like NVMe, TLS

Language / Compiler Extensions

- The *extern* construct is used extensively to define architecture specific functions
  - Invoke low-level instructions for specialized / hardwired functions like
    - Raw-Table: Setup table match to raw memory address
    - Raw-action: Action reference does not come from table entry (is setup by previous actions)
    - Scheduler events
    - Timer events
    - DMA commands / memory read/write
    - Counters / Rate-limiters
    - Data swizzle / encryption
- Many architecture specific *annotations*, like
  - Table write-back (parameter by reference)

```
action nvme_req_tx_sqcb_process(@__ref sqcb_t d) {
  …
        if (__likely(d.busy == d.wb_busy)) {
            d.ring_empty_sched_eval_done = 0;
            ...
}
```

Language / Compiler Extensions

- *Annotations..*
    - Structure field alignment

```
struct metadata_t {
    control_metadata_t        cntrl;
    csum_metadata_t           csum;
    @align(8)
    l3_metadata_t             l3;
    l4_metadata_t             l4;
    ..
}
```

- Symbolic reference to run-time config values

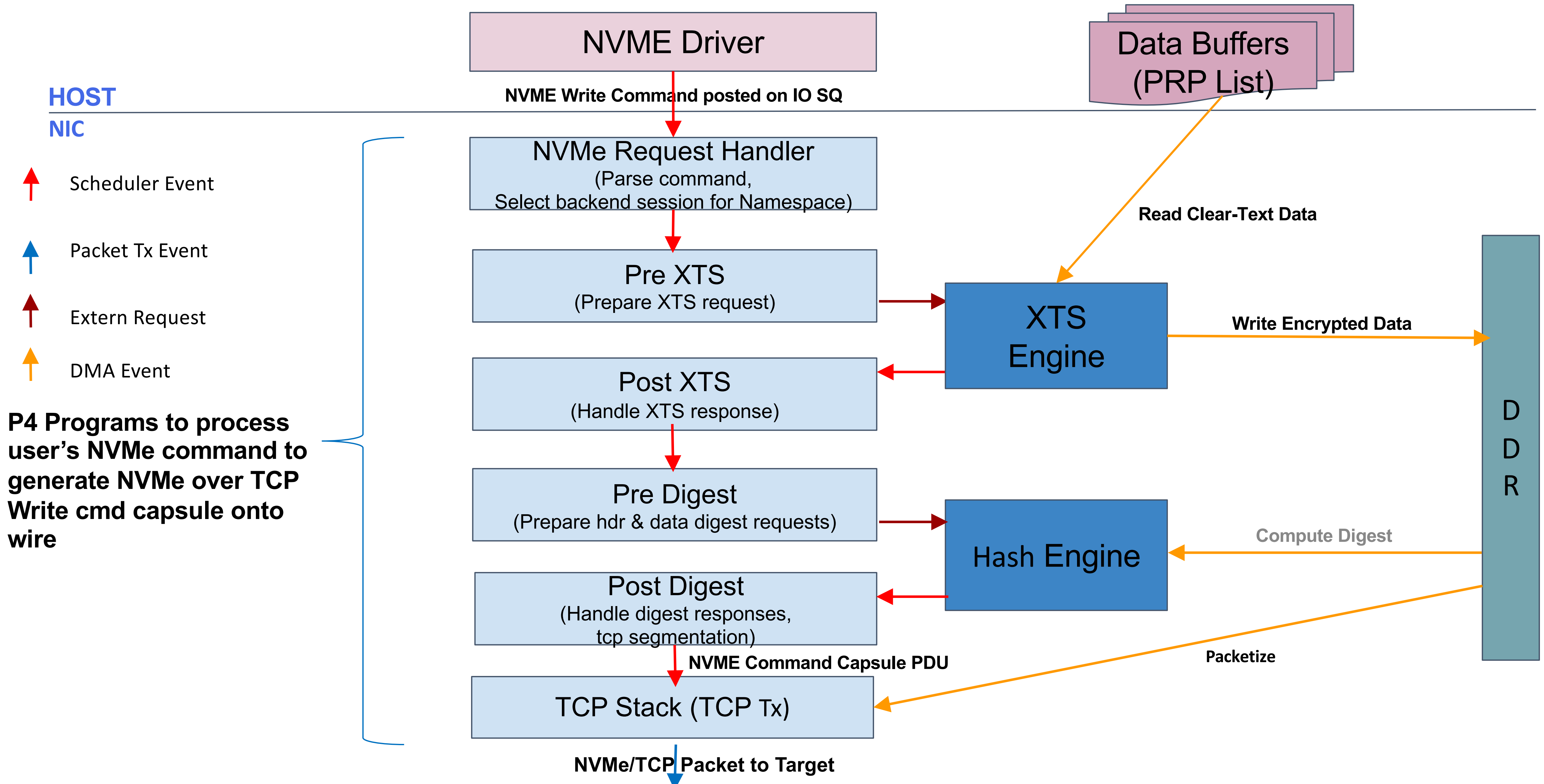
```
action prexts_tx_sess_wqe_process(@__ref sess_wqe_t d) {
    ...
    @param("nvme_tx_pdu_context_base") bit<64> nvme_tx_pdu_context_base;
    bit<64> pdu_ctxt_addr = (bit <64>) (nvme_tx_pdu_context_base + (bit <64>) (d.pduid << 
LOG_PDU_CTXT_SIZE));
    ...
```

Language / Compiler Extensions

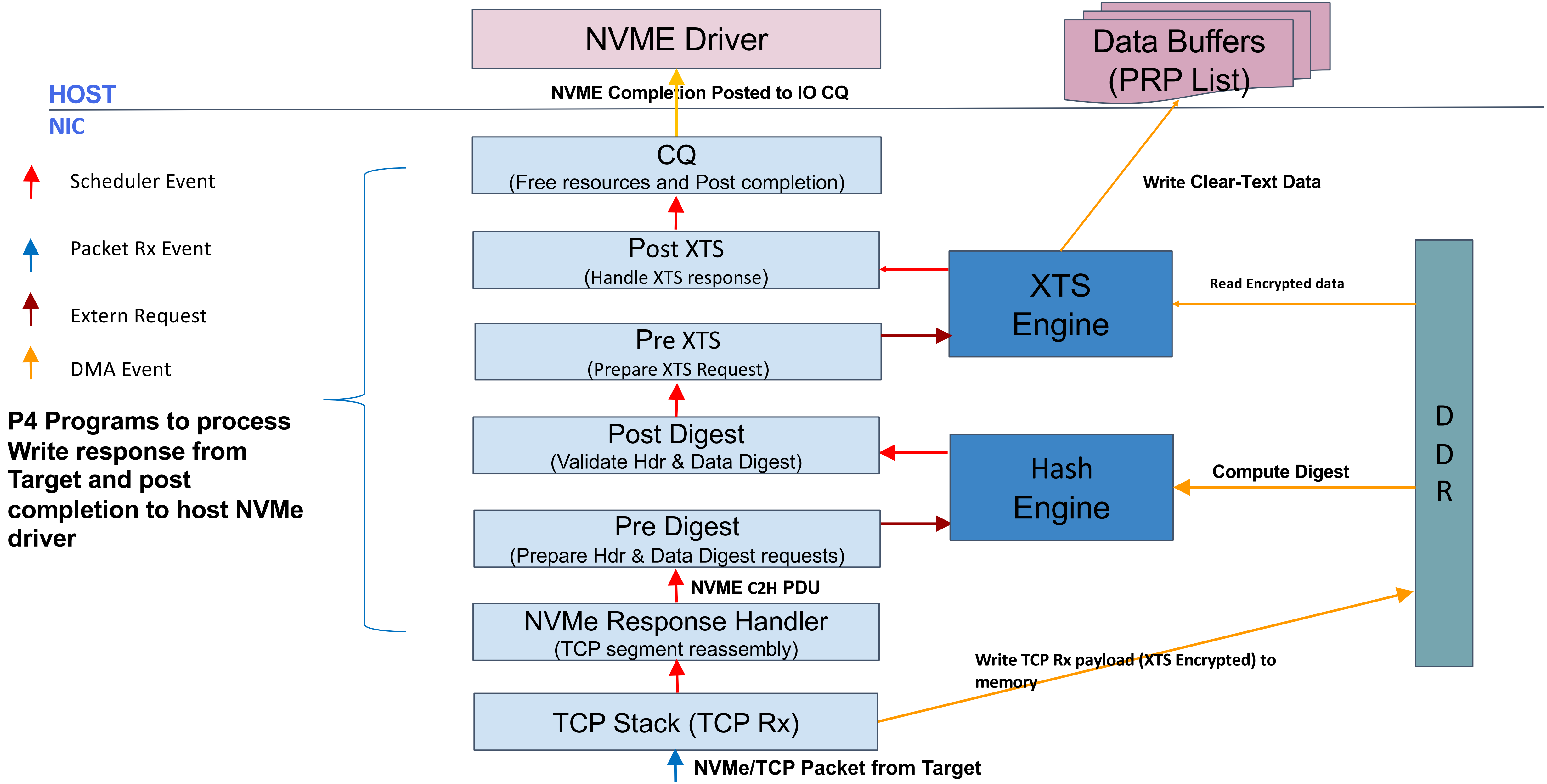
- *New constructs*
  - Loops

```
foreach (bit<2> i in virtio_tx_global.pref_q_index[1:0] .. 2w3) {
    if (buffers_left == 0) {
        break;
    }
    form_one_mem2pkt(buffers_left, i[1:0], desc_flit, 0);
}
```

An example Host-to-Network flow - NVMe Initiator IO: Write Command Request



An example Network-to-host flow -
NVMe Initiator IO: Read Command Response



Thank You

<additional resources>