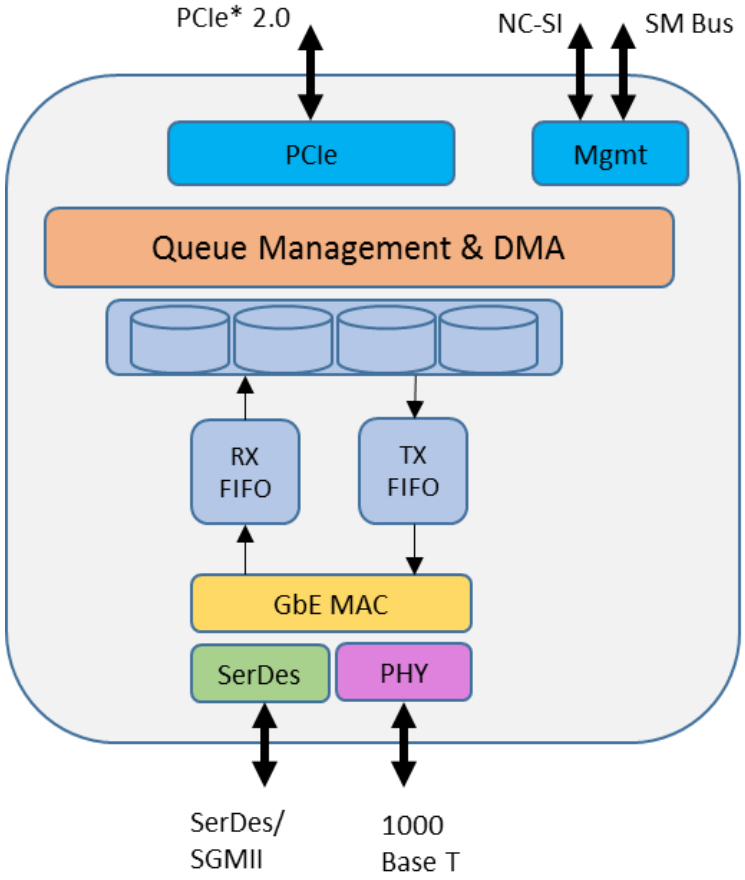
DPDK ETHDEV FOUNDATION

Jingjing Wu, Beilei Xing

Agenda

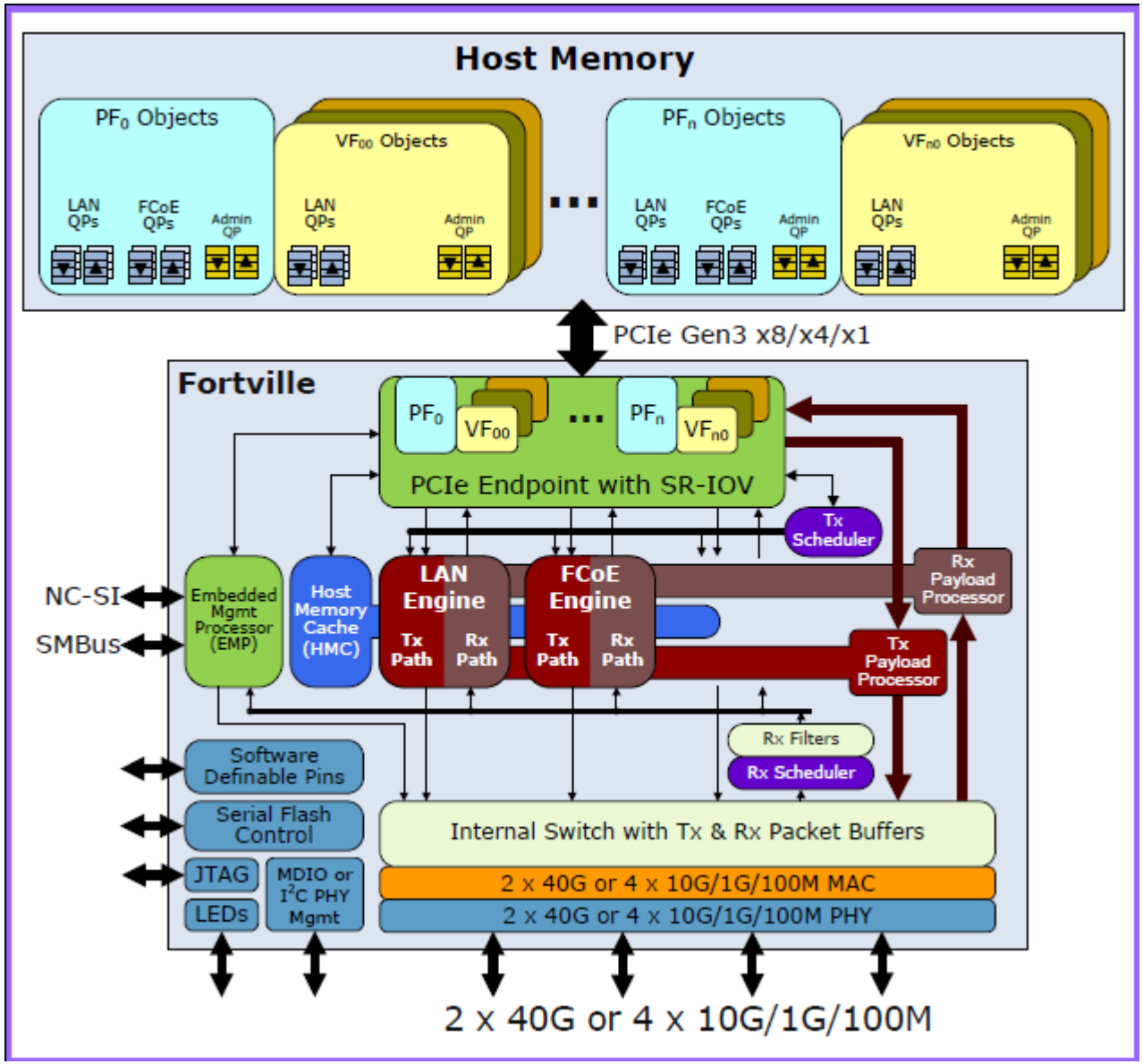
- **Ethernet controller overview**
- **DPDK PMD driver**
- **NIC Features**

Ethernet controller overview



Intel® 82580

- PHY
- MAC
- FIFO
- DMA
- Queue
- PCIe



接收的时候有Rx Filter/Rx Scheduler.

发送和接收的时候都有Payload Processor.猜测是添加package header。

Intel® X710/XL710

Rx Descriptors

Read Format

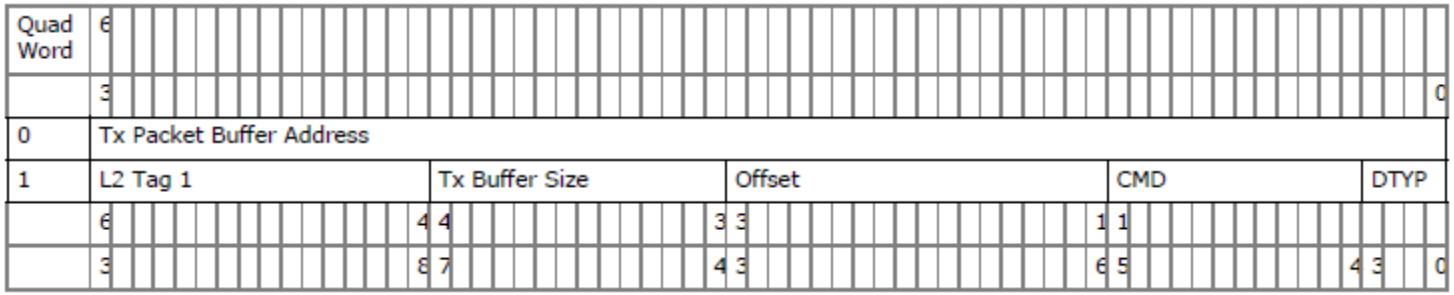
Quad Word	63 ... 0
0	Packet Buffer Address
1	Header Buffer Address
2	Reserved (0x0)
3	Reserved (0x0)

Write-Back Format

Quad Word	63 ... 44	43 ... 33	32 ... 21	20 ... 9	8 ... 7	6 ... 5	4 ... 3	2 ... 0
0	Filter Status		L2TAG1			rsv	MIRR / FCoE Context	
1	Length			PTYPE		rsv	Error	Status
2	L2TAG2 (2nd)		L2TAG2 (1st)			rsv	rsvrsv	Ext_Status
3	FD Filter ID / Flexible Bytes High					Flexible Bytes Low		

Tx Descriptors

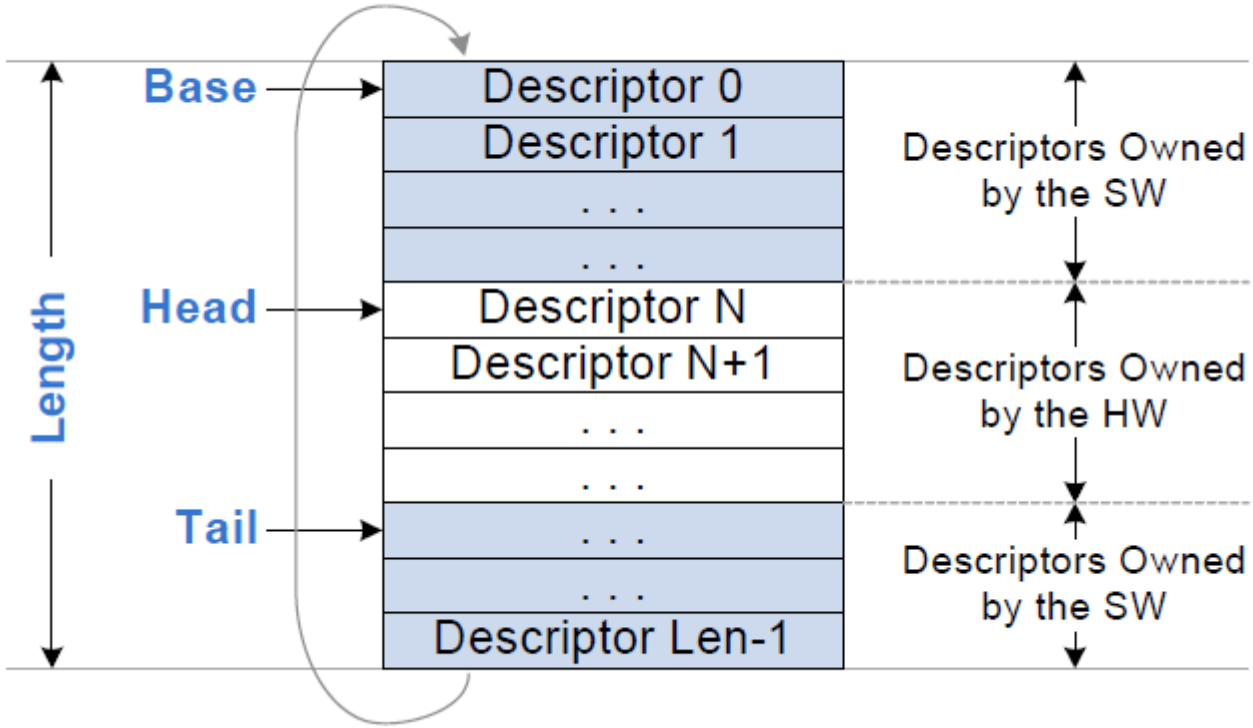
Read Format



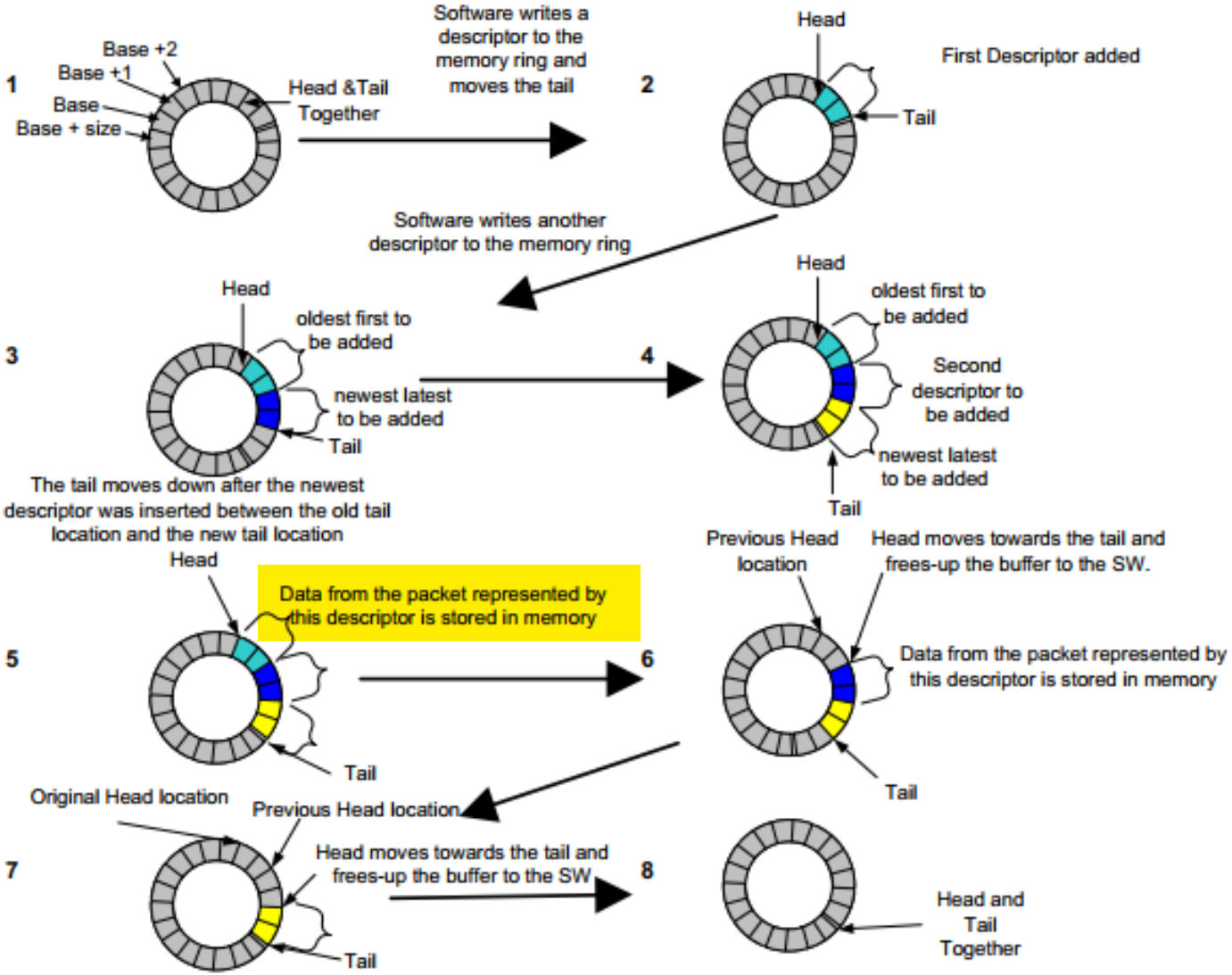
Write-Back: RS bit with DTYP = 0xF

Descriptor Ring Structure

HAS fig.10-6



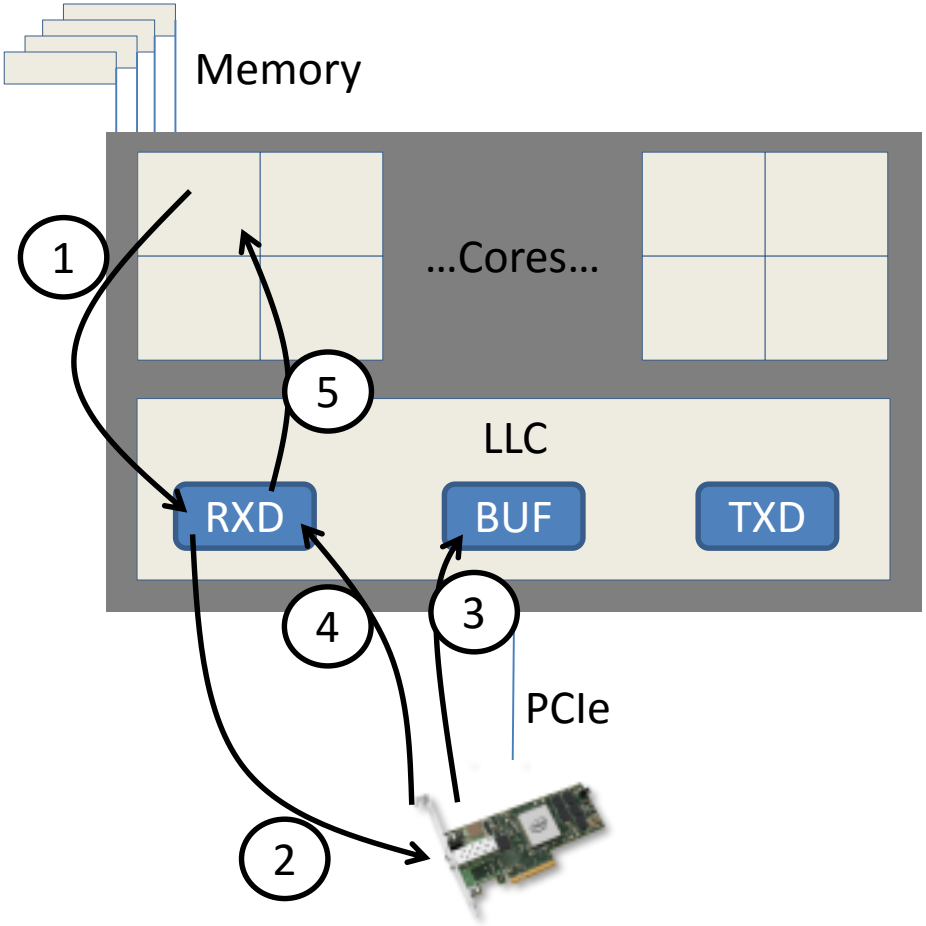
Rx flow



Rx flow

- **PMD checks completion**
- **Application frees mbuf**
- **PMD allocates mbuf**
- **PMD fills descriptor**

Rx Overview

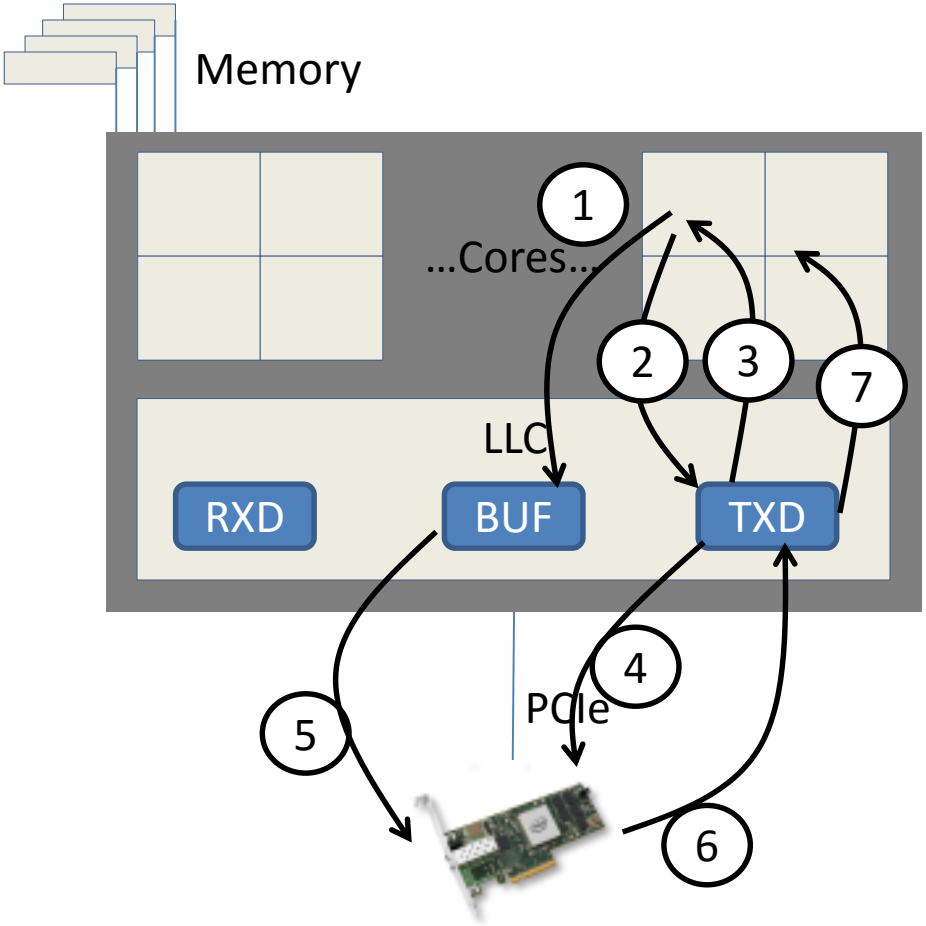


1. CPU Write Rx descriptor `update rx tail`
2. NIC Read Rx descriptor to get buffer address
3. NIC Write Rx packet to buffer address
4. NIC Write Rx descriptor `update rx head`
5. CPU Read Rx descriptor (polling)

Tx flow

- PMD Checks completion
- PMD frees mbuf
- Application prepares mbuf
- PMD fills descriptor

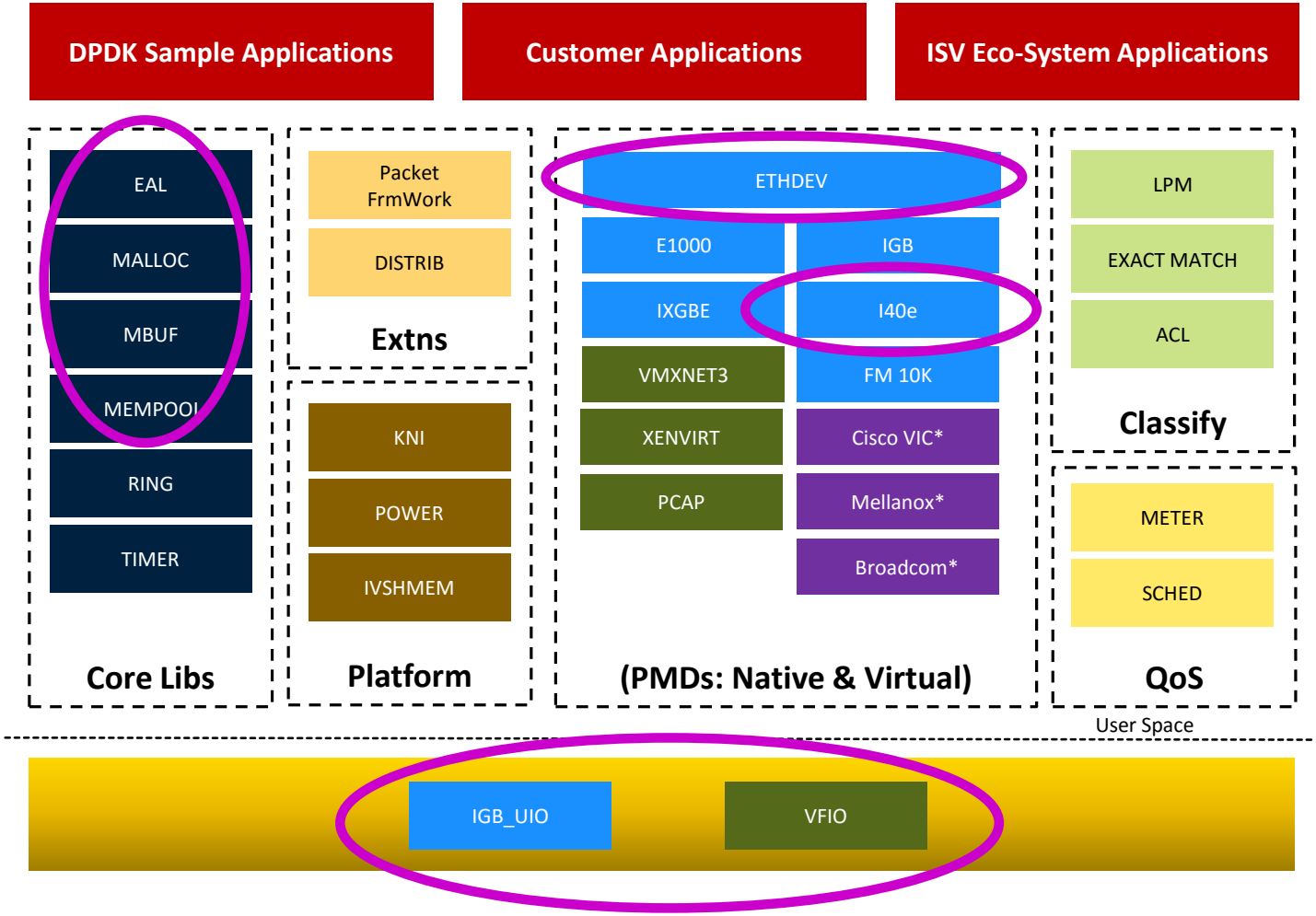
Tx Overview



1. CPU Prepare packet
2. CPU Read Tx descriptor
3. CPU Write Tx descriptor
4. NIC Read Tx descriptor to get buffer address
5. NIC Read Tx packet from buffer address
6. NIC Write Tx descriptor
7. CPU Read Tx descriptor

Poll-Mode Drivers

Data Plane Development Kit Architecture

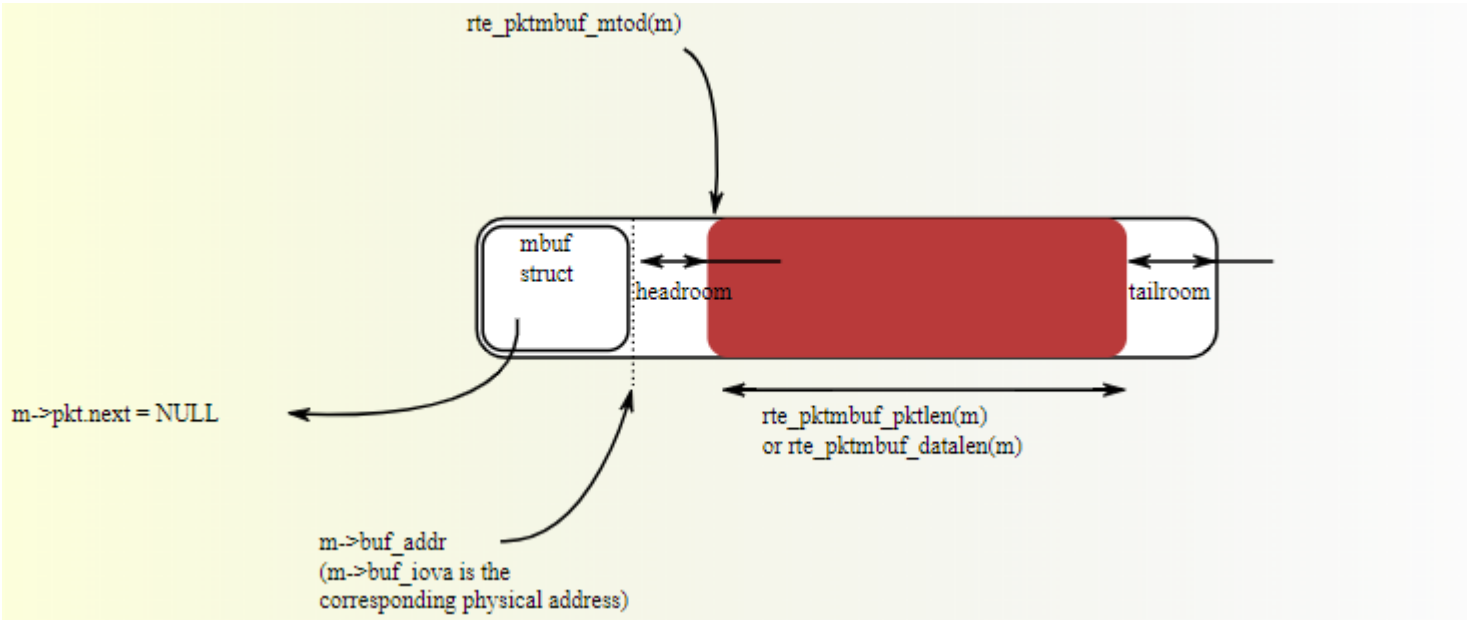


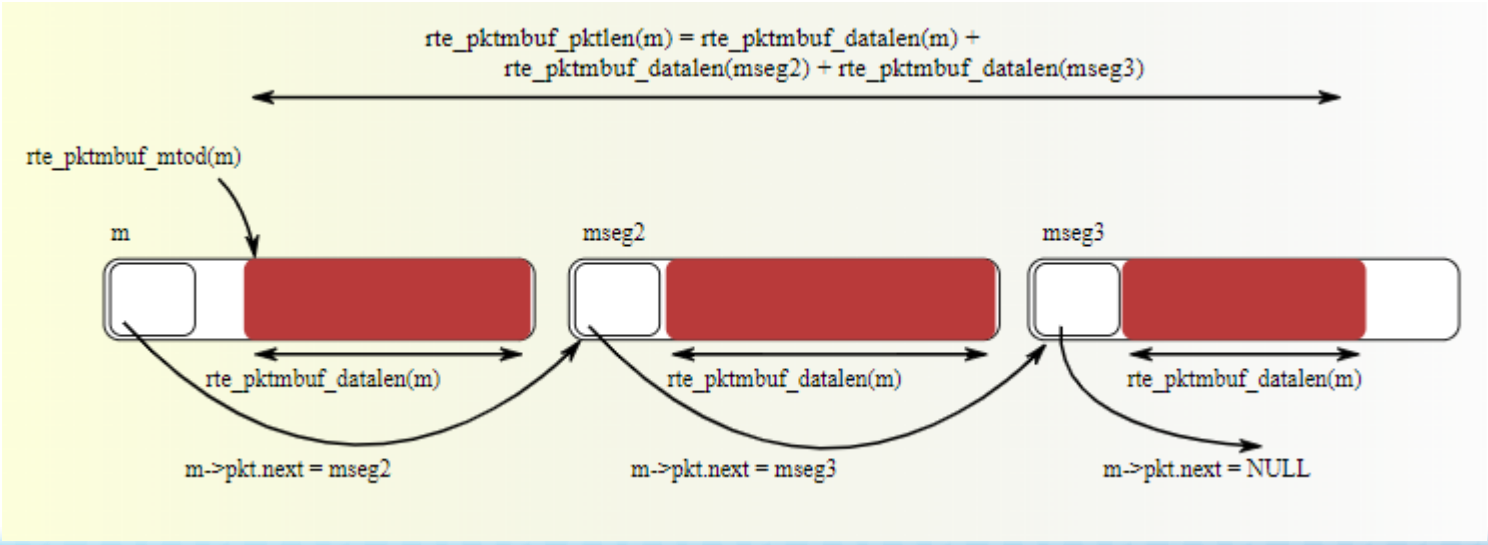
把外设资源映射到用户空间（地址，中断）。
vfio更安全，有保护机制，防止写到内核空间。

Mbuf

```
struct rte_mbuf {
        MARKER cacheline0;
        void *buf_addr;           /**< Virtual address of segment buffer. */
        rte_iova_t buf_iova;      /**< Physical address of segment buffer. */
        /* next 8 bytes are initialized on RX descriptor rearm */
        MARKER8 rearm_data;
        uint16_t data_off;         /**< Offset of data. */
        …… packet type, pkt_len, data_len, rss, fd_id, vlan_tci ……
        /* remaining bytes are set on RX when pulling packet from descriptor */
        MARKER rx_descriptor_fields1;
        …… port, ol_flags, nb_segs ……
        /* second cache line - fields only used in slow path or on TX */
        MARKER cacheline1 __rte_cache_min_aligned;
        union {
                void *userdata;   /**< Can be used for external metadata */
                uint64_t udata64; /**< Allow 8-byte userdata on 32-bit */
        };
        struct rte_mempool *pool; /**< Pool from which mbuf was allocated. */
        struct rte_mbuf *next;    /**< Next segment of scattered packet. */
         …… tx_offload, ……
} __rte_cache_aligned;
```

mbuf


rte_pktmbuf_mtod(m)
mbuf
struct
headroom
tailroom
m->pkt.next = NULL
rte_pktmbuf_pktlen(m)
or rte_pktmbuf_datalen(m)
m->buf_addr
(m->buf_iova is the
corresponding physical address)


rte_pktmbuf_pktlen(m) = rte_pktmbuf_datalen(m) +
rte_pktmbuf_datalen(mseg2) + rte_pktmbuf_datalen(mseg3)
rte_pktmbuf_mtod(m)
m
mseg2
mseg3
rte_pktmbuf_datalen(m)
rte_pktmbuf_datalen(m)
rte_pktmbuf_datalen(m)
m->pkt.next = mseg2
m->pkt.next = mseg3
m->pkt.next = NULL

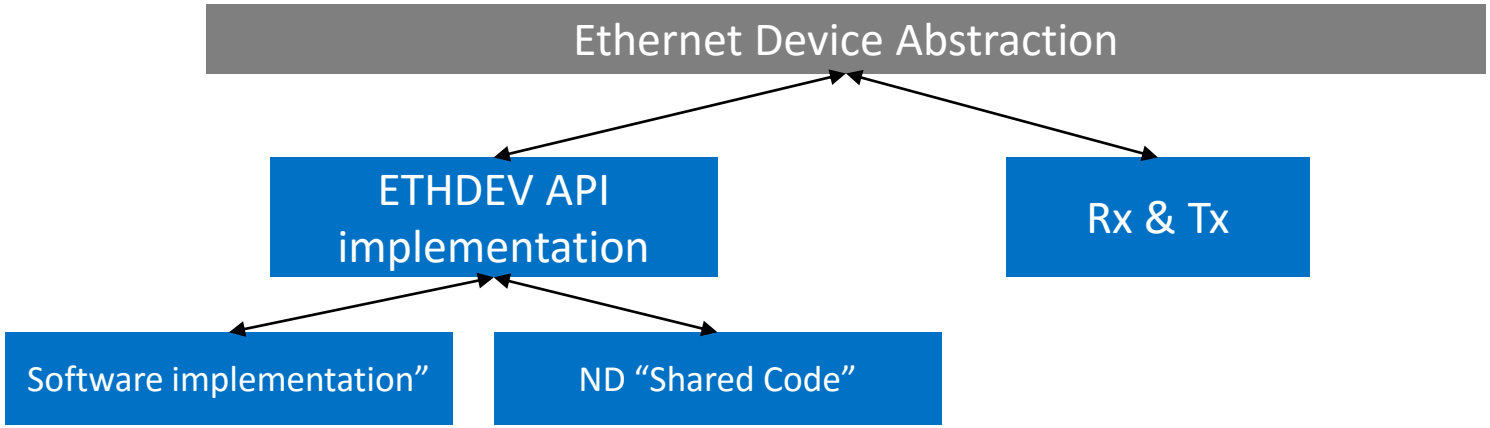
Rx offload

- PKT_RX_VLAN, PKT_RX_VLAN_STRIPPED, PKT_RX_QINQ PKT_RX_QINQ_STRIPPED
- PKT_RX_RSS_HASH, PKT_RX_FDIR, PKT_RX_FDIR_ID, PKT_RX_FDIR_FLX
- PKT_RX_IP_CKSUM_UNKNOWN, PKT_RX_IP_CKSUM_GOOD, PKT_RX_IP_CKSUM_BAD, PKT_RX_IP_CKSUM_NONE
- PKT_RX_L4_CKSUM_UNKNOWN, PKT_RX_L4_CKSUM_BAD, PKT_RX_L4_CKSUM_GOOD, PKT_RX_L4_CKSUM_NONE
- …

Tx offload

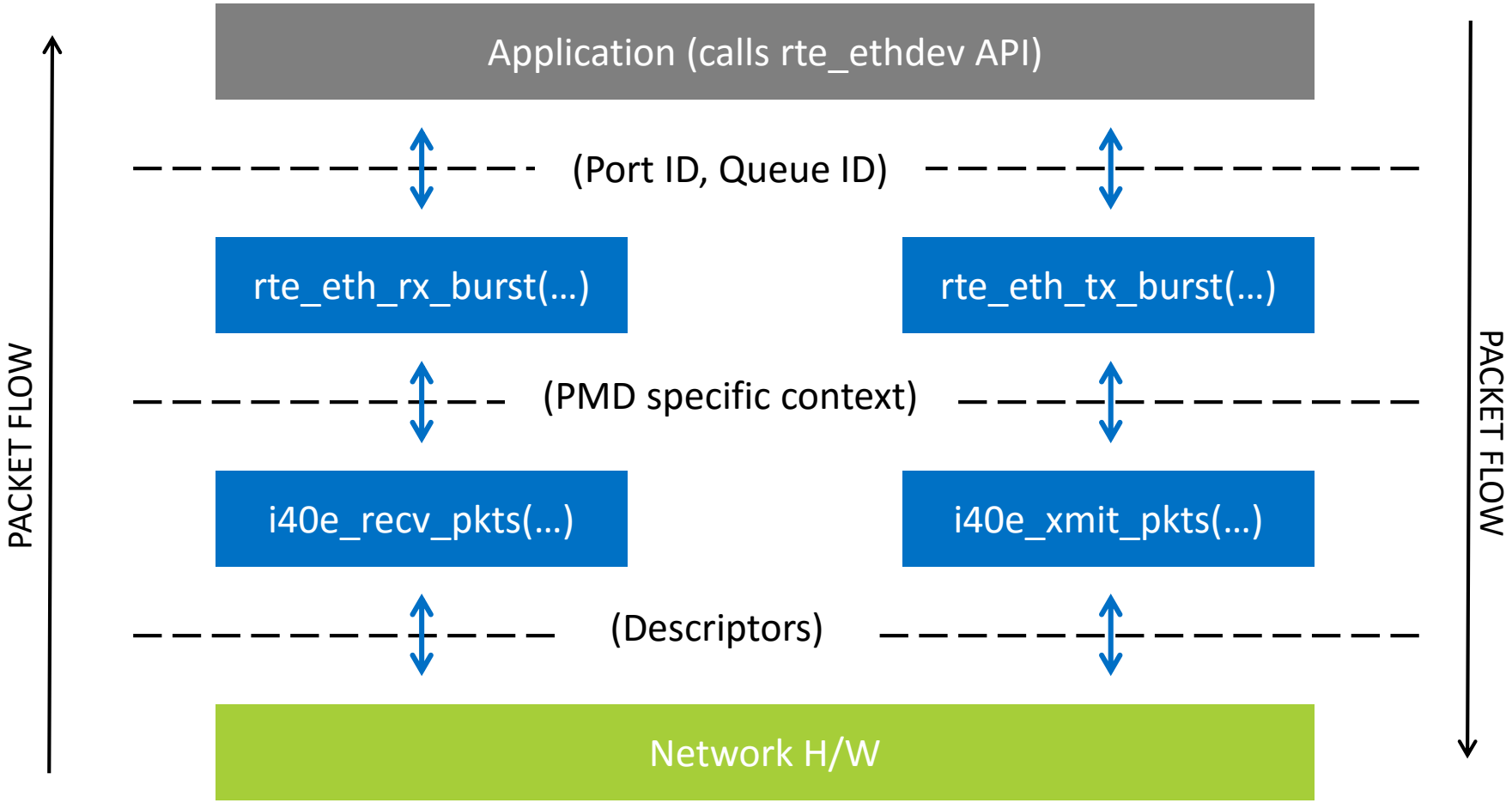
```
#define PKT_TX_OFFLOAD_MASK (    \
            PKT_TX_OUTER_IPV6 |      \
            PKT_TX_OUTER_IPV4 |      \
            PKT_TX_OUTER_IP_CKSUM |  \
            PKT_TX_VLAN_PKT |        \
            PKT_TX_IPV6 |            \
            PKT_TX_IPV4 |            \
            PKT_TX_IP_CKSUM |        \
            PKT_TX_L4_MASK |         \
            PKT_TX_IEEE1588_TMST |   \
            PKT_TX_TCP_SEG |         \
            PKT_TX_QINQ_PKT |        \
            PKT_TX_TUNNEL_MASK |     \
            PKT_TX_MACSEC |          \
            PKT_TX_SEC_OFFLOAD |     \
            PKT_TX_UDP_SEG |         \
            PKT_TX_OUTER_UDP_CKSUM)
```

PMD Logical View

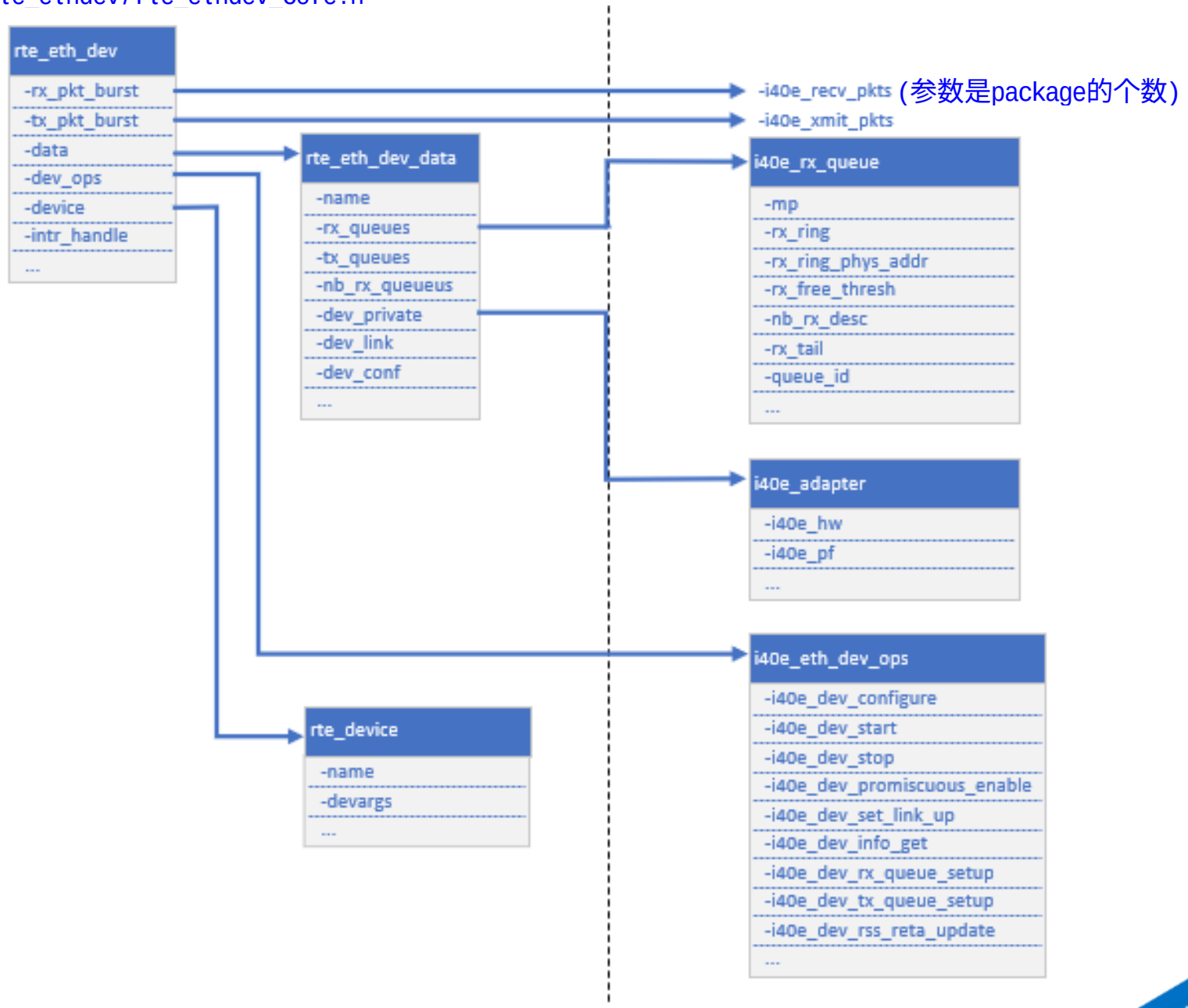


- Hardware PMDs implement the DPDK Ethernet device abstraction by programming NIC’s registers or wrapping around ND Shared code.
- Receive and Transmit functions written from scratch, optimized for the hardware

Ethernet Device Framework



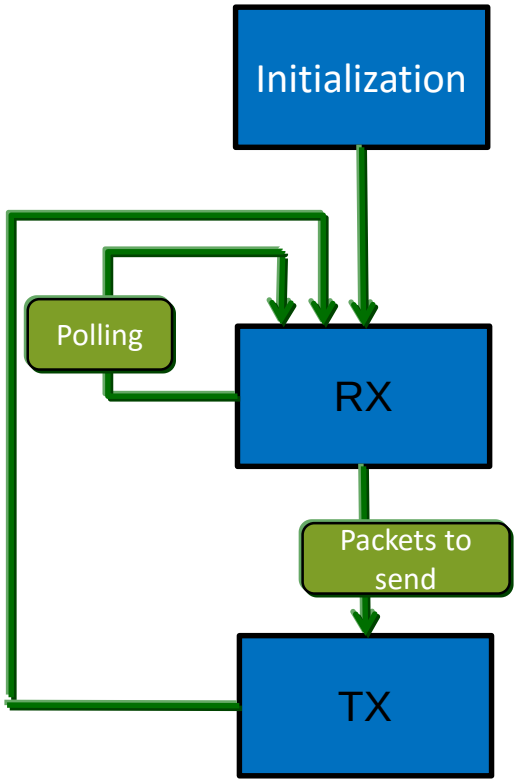
lib/librte_ethdev/rte_ethdev_core.h



Example DPDK Network Application

```
int main(int argc, char **argv)
{
    /* Initialize environment */
    rte_eal_init(argc, argv);
    rte_eal_pci_probe();

    /* Allocate memory for packet buffers */
    buf_pool = rte_mempool_create("pool", num_mbufs,..., socket_id, FLAGS);

    /* Configure device */
    rte_eth_dev_configure(port_id, num_rxqs, num_txqs, &port_conf);

    /* Configure device Rx/Tx queues */
    rte_eth_rx_queue_setup(port_id, queue_id, num_rxds, socket_id, &rx_conf, buf_pool);
    rte_eth_tx_queue_setup(port_id, queue_id, num_txds, socket_id, &tx_conf);

    /* Start the device */
    rte_eth_dev_start(port_id);

    /* Echo back any received packets */
    while (!done) {
        /* Receive packets */
        num_rx = rte_eth_rx_burst(port_id, queue_id, pkt_list, MAX_BURST);
        if (!num_rx) continue;

        /* Transmit packets */
        num_tx = rte_eth_tx_burst(port_id, queue_id, pkt_list, num_rx);
        while (num_tx != num_rx)
            /* uh oh... drop some pkts */
            rte_pktmbuf_free(pkt_list[num_tx++]);
    }

    /* Stop the device */
    rte_eth_dev_stop(port_id);
}
```

Simple forward example

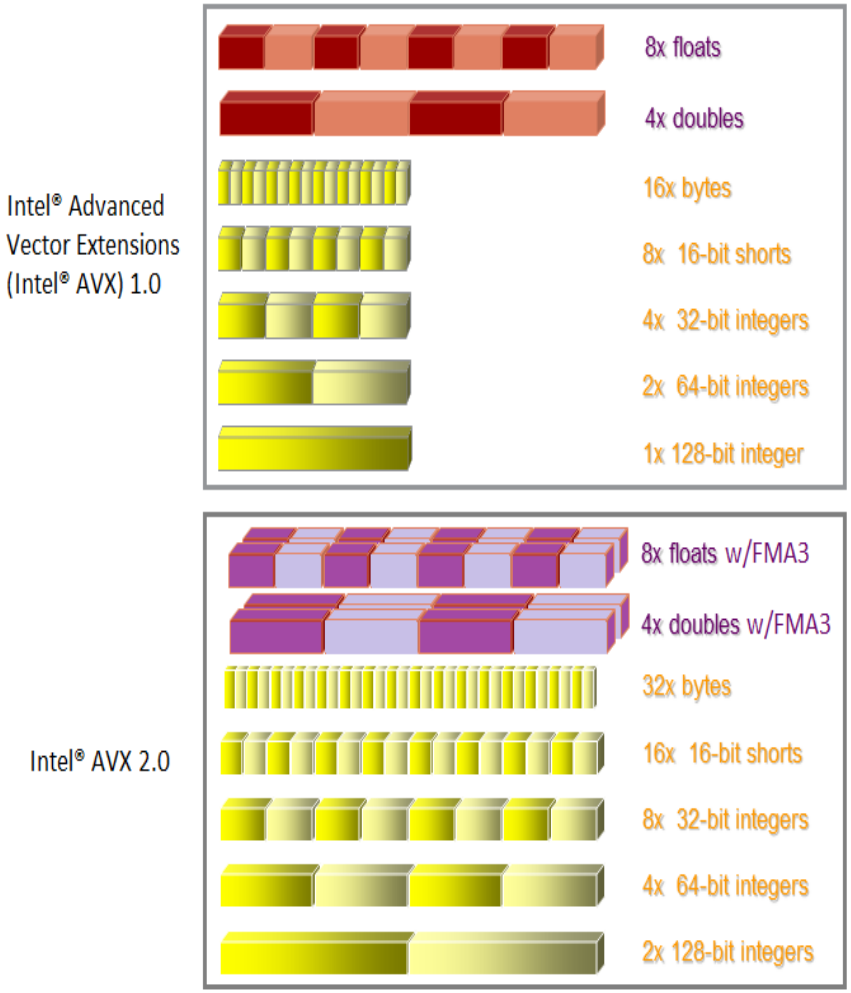


1. Initialization
   - Init Memory Zones and Pools
   - Init Devices and Device Queues
   - Start Packet Forwarding Application
2. Packet Reception (RX)
   - Poll Devices' RX queues and receive packets in bursts
   - Allocate new RX buffers from per queue memory pools to stuff into descriptors
3. Packet Transmission (TX)
   - Transmit the received packets from RX
   - Free the buffers that we used to store the packets

- **Bursting** 批量分配释放buffer。
  - Multiple buffers can be allocated, and sometimes freed, at once, removing per-packet overhead
- **Scattered**
  - Multi-segmented rte_mbuf 第一个mbuf
  - EOP flag in descriptor indicates if the descriptor is the last one of the packet. 最后一个mbuf
- **Threshold**
  - rx_free_thresh: Drives the freeing of RX descriptors 32 ice_rxtx.c
  - tx_rs_thresh : Drives the setting of RS bit on TXDs
  - tx_free_thresh : Start freeing TX buffers if there are less free descriptors than this value
- **Number of descriptors on queue**

Vector PMD

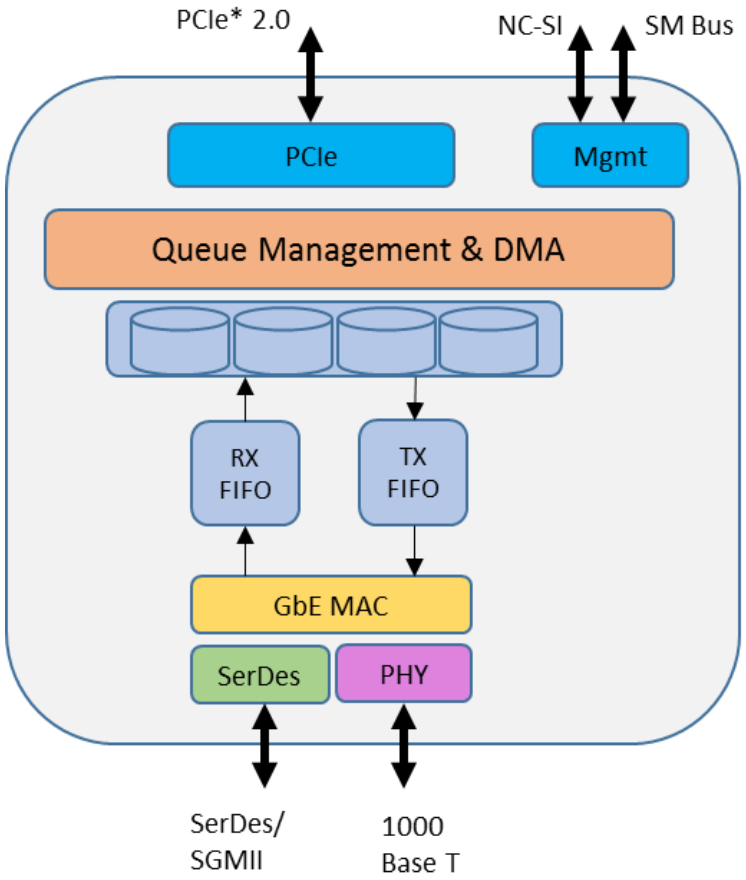
- **Intel® SSE, AVX**
- **Bulk Processing**



NIC features Overview:

-- Intel® Ethernet Controller XL710 (Fortville)

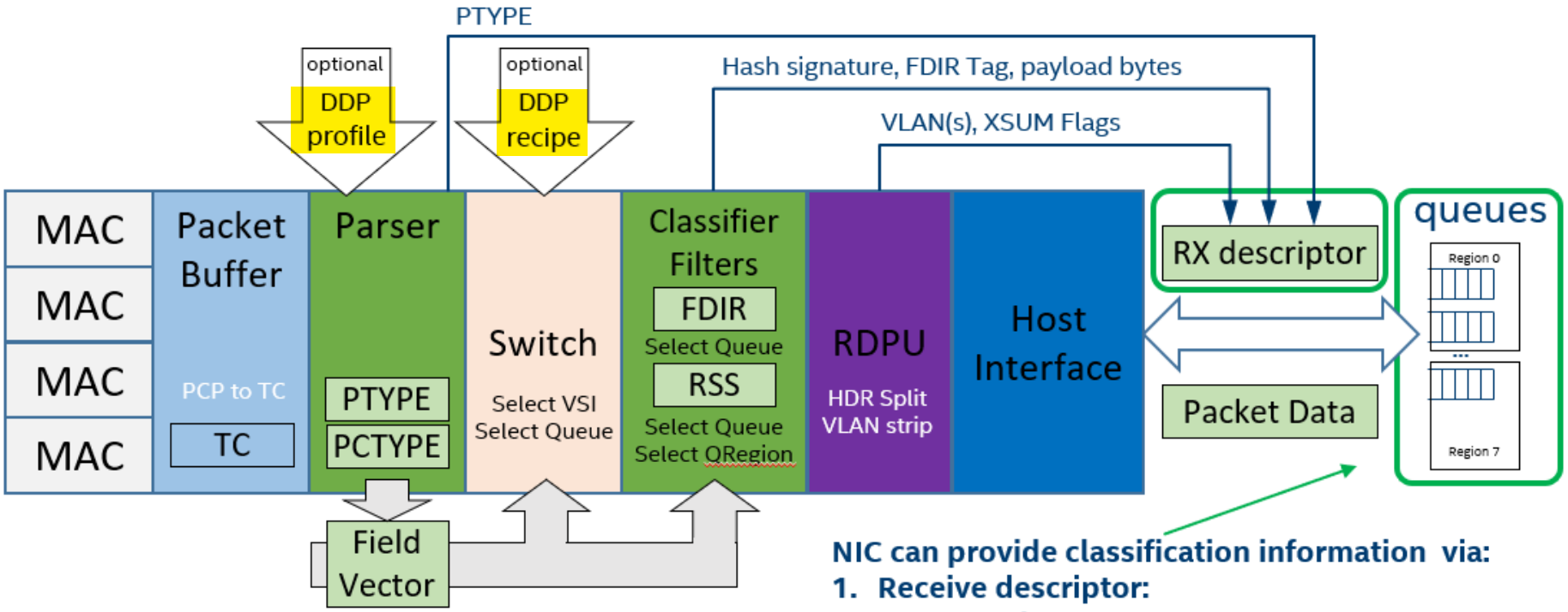
Multi-queue



多核处理包，使用multi queue可以更好地利用多核性能。queue和core绑定

NIC	Rx Queue
NNT	128
FVL	1536
CVL	2048

Rx Processing pipeline

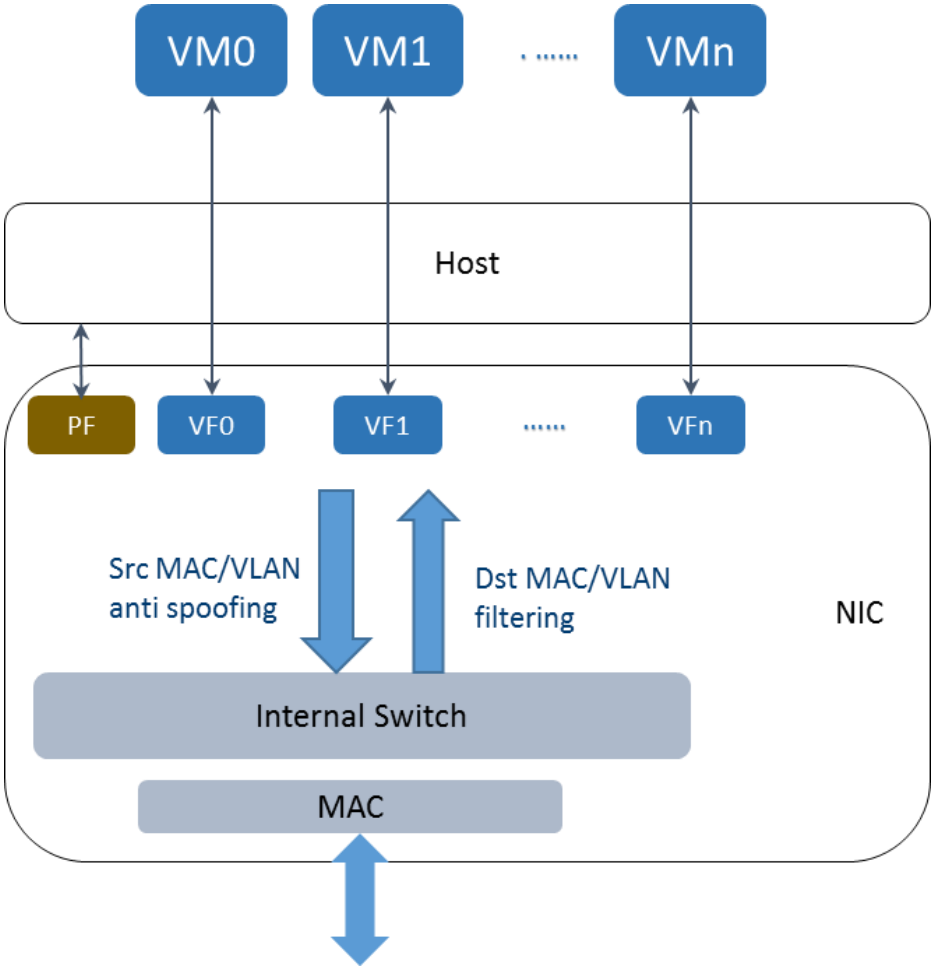


NIC can provide classification information via:

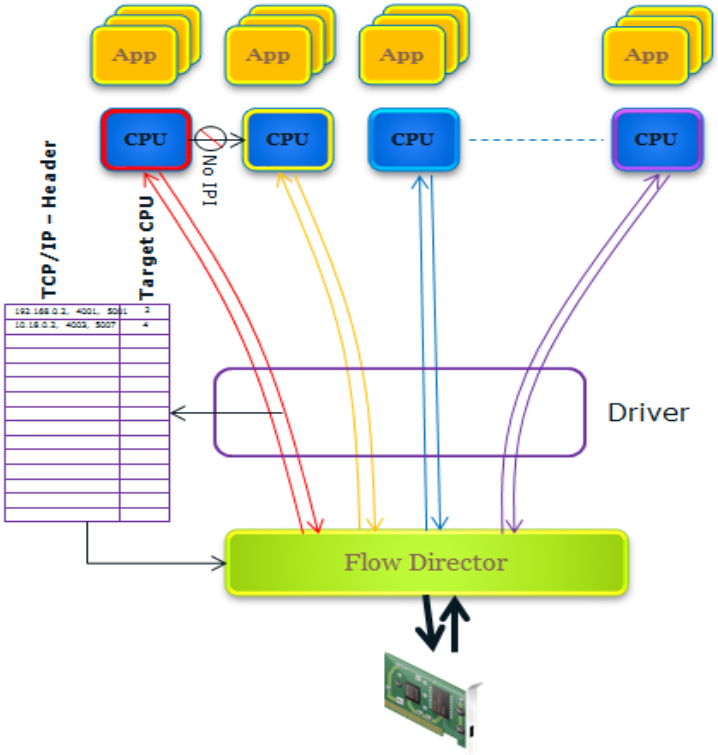
1. **Receive descriptor:**
   - **Packet Type (PTYPE)**
   - **Packet validation flags**
   - **Metadata (VLAN, payload)**
2. **Receive queue index**

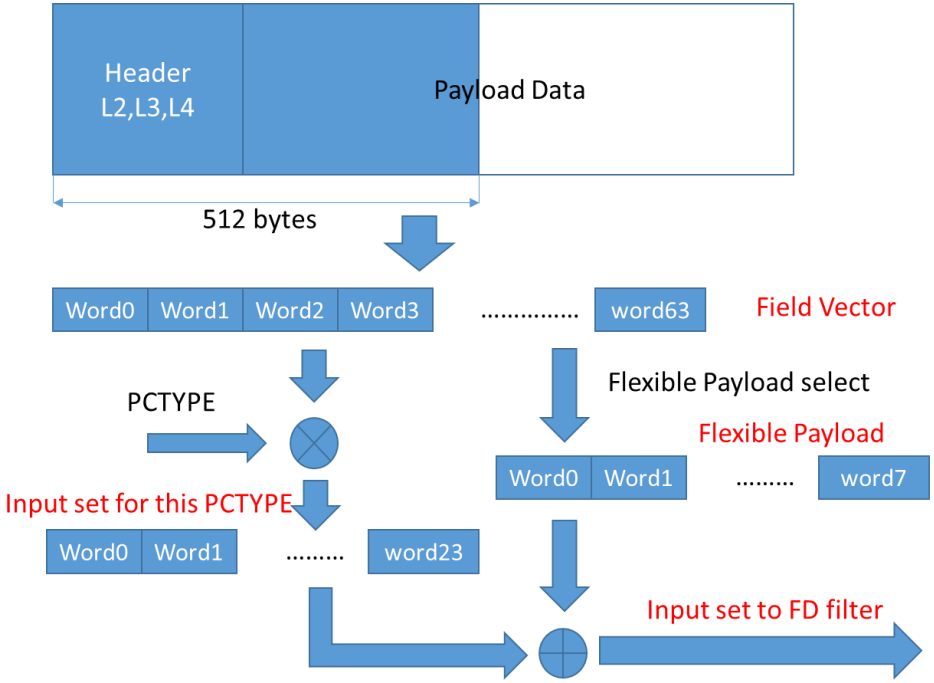
看一下这个图。

MAC VLAN filtering/Internal Switch/cloud filter



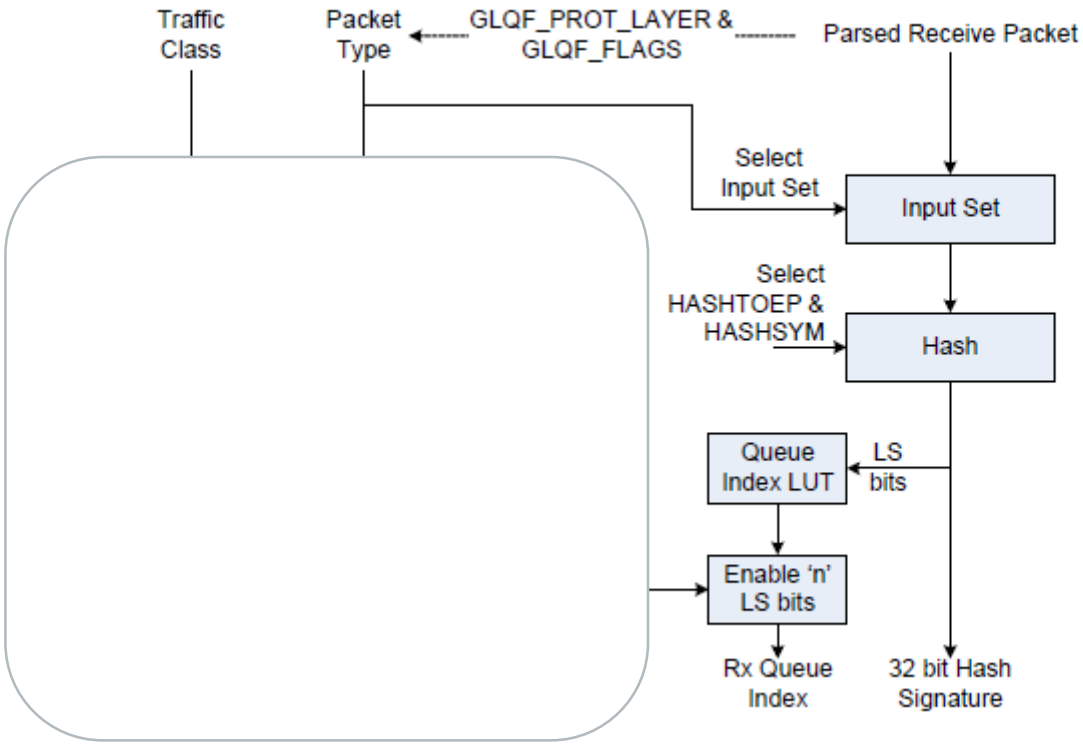
Flow director





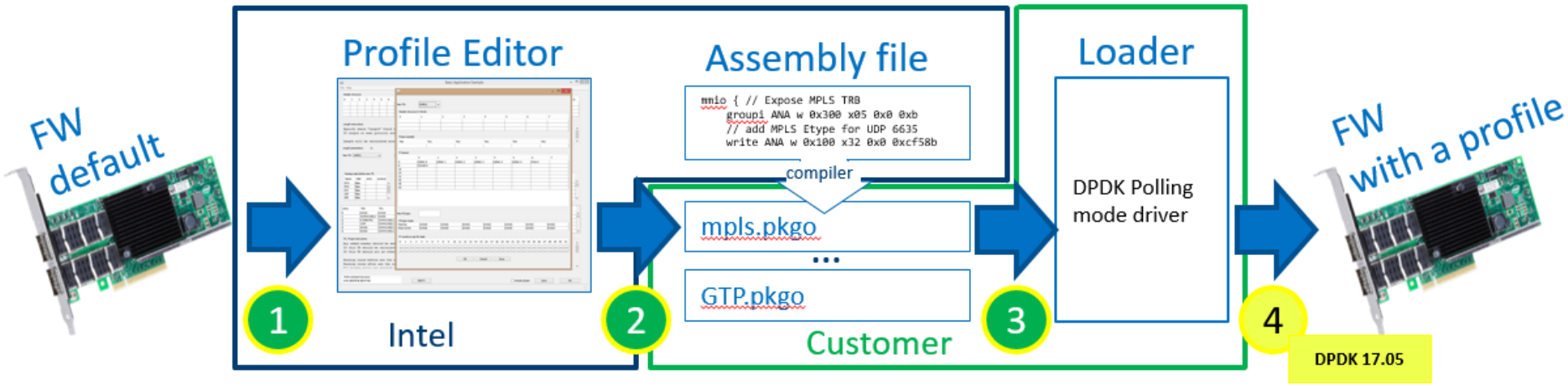
- **rte_mbuf->ol_flags, rte_mbuf->hash. fdir,**
- **Matching Key**

RSS – Receive Side Scaling



- **rte_mbuf->ol_flags, rte_mbuf->hash.rss**
- **LUT**
- **Queue region**

DDP


FW default
Profile Editor
Assembly file
mmio { // Expose MPLS TRB
groupi ANA w 0x300 x05 0x0 0xb
// add MPLS Etype for UDP 6635
write ANA w 0x100 x32 0x0 0xcf58b
compiler
mpls.pkgo
...
GTP.pkgo
Loader
DPDK Polling mode driver
FW with a profile
1
Intel
2
Customer
3
4
DPDK 17.05

HW offload

```
dev_info->rx_offload_capa =
        DEV_RX_OFFLOAD_VLAN_STRIP |
        DEV_RX_OFFLOAD_QINQ_STRIP |
        DEV_RX_OFFLOAD_IPV4_CKSUM |
        DEV_RX_OFFLOAD_UDP_CKSUM |
        DEV_RX_OFFLOAD_TCP_CKSUM |
        DEV_RX_OFFLOAD_OUTER_IPV4_CKSUM |
        DEV_RX_OFFLOAD_KEEP_CRC |
        DEV_RX_OFFLOAD_SCATTER |
        DEV_RX_OFFLOAD_VLAN_EXTEND |
        DEV_RX_OFFLOAD_VLAN_FILTER |
        DEV_RX_OFFLOAD_JUMBO_FRAME |
        DEV_RX_OFFLOAD_RSS_HASH;
```

```
dev_info->tx_offload_capa =
        DEV_TX_OFFLOAD_VLAN_INSERT |
        DEV_TX_OFFLOAD_QINQ_INSERT |
        DEV_TX_OFFLOAD_IPV4_CKSUM |
        DEV_TX_OFFLOAD_UDP_CKSUM |
        DEV_TX_OFFLOAD_TCP_CKSUM |
        DEV_TX_OFFLOAD_SCTP_CKSUM |
        DEV_TX_OFFLOAD_OUTER_IPV4_CKSUM |
        DEV_TX_OFFLOAD_TCP_TSO |
        DEV_TX_OFFLOAD_VXLAN_TNL_TSO |
        DEV_TX_OFFLOAD_GRE_TNL_TSO |
        DEV_TX_OFFLOAD_IPIP_TNL_TSO |
        DEV_TX_OFFLOAD_GENEVE_TNL_TSO |
        DEV_TX_OFFLOAD_MULTI_SEGS |
        dev_info->tx_queue_offload_capa;
```

RX HW offload

- **rte_mbuf->ol_flags indicate the Rx offload status**
- **Classification filter**
- **Checksum verification**
- **VLAN stripping**
- **CRC stripping**

PKT_RX_VLAN_PKT	VLAN is stripping to descriptor
PKT_RX_RSS_HASH	RSS hash value is stored in mbuf
PKT_RX_FDIR	Flow director ID is stored in mbuf
PKT_RX_L4_CKSUM_BAD PKT_RX_IP_CKSUM_BAD	Checksum verification
PKT_RX_IEEE1588_PTP PKT_RX_IEEE1588_TMST	IEEE1588

Pack type

- **rte_mbuf->ptype indicates the packet type according to NIC's parsing.**

```
union {
        uint32_t packet_type; /**< L2/L3/L4 and tunnel information. */
        struct {
                uint32_t l2_type:4; /**< (Outer) L2 type. */
                uint32_t l3_type:4; /**< (Outer) L3 type. */
                uint32_t l4_type:4; /**< (Outer) L4 type. */
                uint32_t tun_type:4; /**< Tunnel type. */
                uint32_t inner_l2_type:4; /**< Inner L2 type. */
                uint32_t inner_l3_type:4; /**< Inner L3 type. */
                uint32_t inner_l4_type:4; /**< Inner L4 type. */
        };
};
```

Example：

RTE_PTYPE_L2_ETHER |
RTE_PTYPE_L3_IPV4_EXT_UNKNOWN |
RTE_PTYPE_L4_TCP

Tx HW offload

- **VLAN Insert**
- **Checksum offload**
- **TSO**

```
uint64_t ol_flags;        /**< Offload features. */
…….
/* fields to support TX offloads */
union {
        uint64_t tx_offload;       /**< combined for easy fetch */
        struct {
                uint64_t l2_len:7; /**< L2 (MAC) Header Length. */
                uint64_t l3_len:9; /**< L3 (IP) Header Length. */
                uint64_t l4_len:8; /**< L4 (TCP/UDP) Header Length. */
                uint64_t tso_segsz:16; /**< TCP TSO segment size */

                /* fields for TX offloading of tunnels */
                uint64_t outer_l3_len:9; /**< Outer L3 (IP) Hdr Length. */
                uint64_t outer_l2_len:7; /**< Outer L2 (MAC) Hdr Length. */
        };
};
```

Thanks