DPDK

Mbuf Changes

Olivier Matz

DPDK Summit Userspace - Dublin- 2016

Plan

- Some recent changes (16.07) in mbuf and mempool
- What’s new in 16.11?
- Ideas for next versions

16.07: mempool memory allocation

- Allow allocation of large mempools in non-contiguous virtual memory
- New API with less arguments (create, populate, obj_init, ...)
- Freeing a mempool is now possible
- Mempool outside hugepage memory

16.07: mempool handlers

- Previously, a mempool stored its objects in a ring
- New API to register a pool handler
- No modification of the per-core cache
- Opens the door for hardware-assisted mempool handler

16.07: user-owned cache

- A mempool object embeds a per-core cache (=per eal thread)
- New API to use a specific cache when enqueing/dequeing objects in a mempool
- Needed to efficiently use a mempool from non-eal threads
- Note: ring still requires that threads are not preemptable

16.07: other mbuf changes

- Raw mbuf allocation becomes public
- New Rx flag for stripped Vlan
- Prefetch helpers

16.11: rx checksum flags

- Previously, there was only one flag “checksum bad”
- Add a new flag, allowing to express:
  - Checksum bad
  - Checksum good
  - Checksum unknown
  - Checksum not present but packet valid (enables offload in virtual drivers)

16.11: software packet type parser

- Parse the network headers in packet data and return a packet type
- Provide a reference implementation to compare with drivers
- Needed for virtio Rx offload

16.11: other mbuf changes

- API to reset the headroom of a packet
- Safe API to read the content of a packet
- New Tx flags for offload in tunnels (TSO or checksum)

Mbuf structure reorganisation

Discuss: adding a new field in mbuf

- The mbuf is a core dpdk structure, used to carry network packets
- Limit/bulk its modification
- How to decide which features should be in the first part (Rx)?
- Can we extend the mbuf ad infinitum?
- Example: timestamp

Discuss: structure reordering

- The mbuf structure is split in 2 part (Rx, Tx) and room in first part is tight
- In PMD Rx functions, it is needed to set m→next to NULL, which is in the Tx part
- m→rearm marker is not aligned, which costs on some architectures
- m→port and m→nb_segs are 8 bits wide
- Is m→port needed?

Discuss: raw mbuf alloc/free + refcnt

- The __rte_mbuf_raw_free() function is not public while the alloc function is
- The raw alloc sets refcnt to 1, free expects refcnt=0
- A solution would be to have m→refcnt to 1 for mbuf in the pool, restoring symmetry and allowing bulk allocation/free
- Same for m→next which could be NULL

Discuss: new mbuf structure proposal

DPDK

```
struct rte_mbuf {
        /* --- cacheline 0 boundary (64 bytes) --- */
        MARKER                    cacheline0;           /*     0     0 */
        void *                    buf_addr;             /*     0     8 */
        phys_addr_t               buf_physaddr;         /*     8     8 */
        uint16_t                  buf_len;              /*    16     2 */
        MARKER8                   rearm_data;           /*    18     0 */
        uint16_t                  data_off;             /*    18     2 */
        uint16_t                  refcnt;               /*    20     2 */
        uint8_t                   nb_segs;              /*    22     1 */
        uint8_t                   port;                 /*    23     1 */
        uint64_t                  ol_flags;             /*    24     8 */
        MARKER                    rx_descriptor_fields1;/*    32     0 */
        uint32_t                  packet_type;          /*    32     4 */
        uint32_t                  pkt_len;              /*    36     4 */
        uint16_t                  data_len;             /*    40     2 */
        uint16_t                  vlan_tci;             /*    42     2 */
        uint64_t                  hash;                 /*    44     8 */
        uint32_t                  seqn;                 /*    52     4 */
        uint16_t                  vlan_tci_outer;       /*    56     2 */
        /* XXX 6 bytes hole, try to pack */
        /* --- cacheline 1 boundary (64 bytes) --- */
        MARKER                    cacheline1;           /*    64     0 */
        void *                    userdata;             /*    64     8 */
        struct rte_mempool *      pool;                 /*    72     8 */
        struct rte_mbuf *         next;                 /*    80     8 */
        uint64_t                  tx_offload;           /*    88     8 */
        uint16_t                  priv_size;            /*    96     2 */
        uint16_t                  timesync;             /*    98     2 */
};
```

```
struct rte_mbuf {
        /* --- cacheline 0 boundary (64 bytes) --- */
        MARKER                    cacheline0;           /*     0     0 */
        void *                    buf_addr;             /*     0     8 */
        phys_addr_t               buf_physaddr;         /*     8     8 */
        MARKER64                  rearm_data;           /*    16     0 */
        uint16_t                  data_off;             /*    16     2 */
        uint16_t                  refcnt;               /*    18     2 */
        uint16_t                  nb_segs;              /*    20     2 */
        uint16_t                  port;                 /*    22     2 */
        uint64_t                  ol_flags;             /*    24     8 */
        MARKER                    rx_descriptor_fields1;/*    32     0 */
        uint32_t                  packet_type;          /*    32     4 */
        uint32_t                  pkt_len;              /*    36     4 */
        uint16_t                  data_len;             /*    40     2 */
        uint16_t                  vlan_tci;             /*    42     2 */
        uint64_t                  hash;                 /*    44     8 */
        uint32_t                  seqn;                 /*    52     4 */
        uint16_t                  vlan_tci_outer;       /*    56     2 */
        uint16_t                  buf_len;              /*    58     2 */
        /* XXX 4 bytes hole, try to pack */
        /* --- cacheline 1 boundary (64 bytes) --- */
        MARKER                    cacheline1;           /*    64     0 */
        void *                    userdata;             /*    64     8 */
        struct rte_mempool *      pool;                 /*    72     8 */
        struct rte_mbuf *         next;                 /*    80     8 */
        uint64_t                  tx_offload;           /*    88     8 */
        uint16_t                  priv_size;            /*    96     2 */
        uint16_t                  timesync;             /*    98     2 */
};
```

Mbuf pool handler

Discuss: default mbuf pool handler

- Currently, the default mbuf pool handler “ring_mp_mc”, set at compilation time
- Hardware-assisted pools are coming
- Hardware have constraints/capabilities
- But application/user decide
- Add params to rte_pktmbuf_pool_create()?
- Global mbuf lib parameter?

Discuss: mempool stack handler

- By default, the mempool uses a ring (FIFO) to store the objects
- Using a LIFO may provide better performance to avoid cache eviction
- There is already a stack handler, but it could be enhanced to be lockless

Mbuf with external data buffer

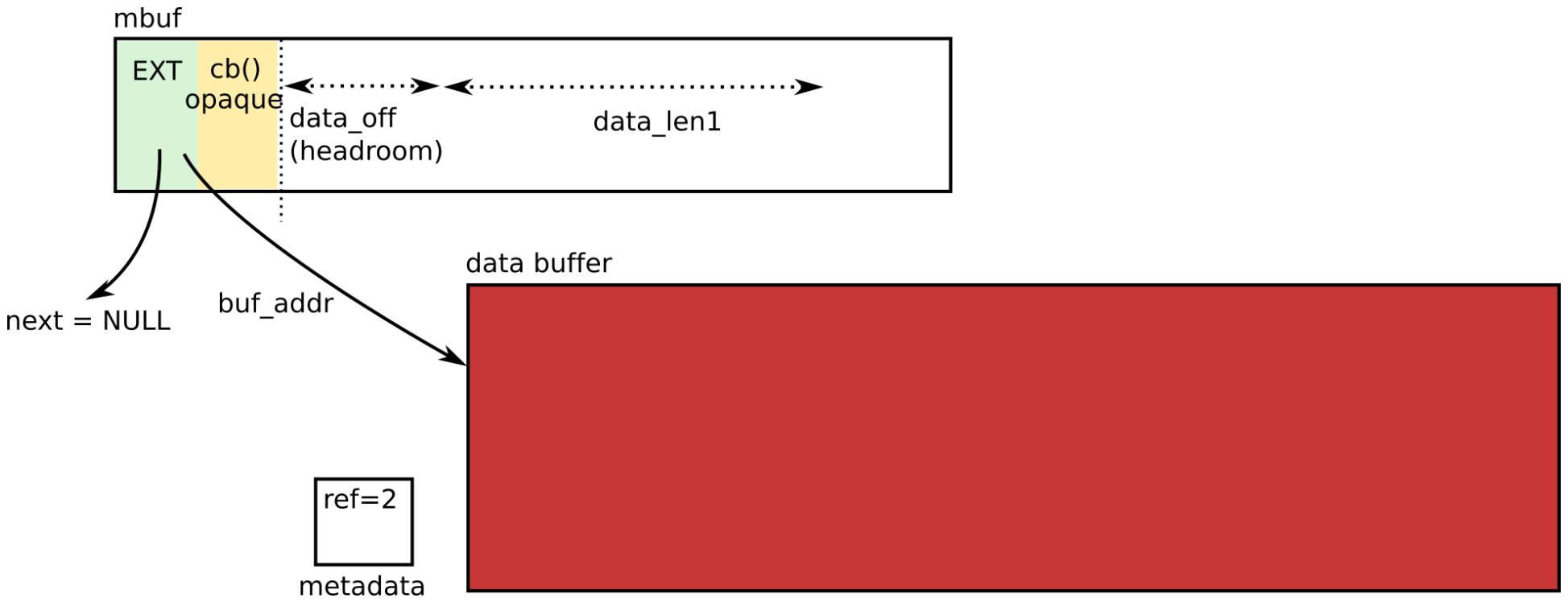
Discuss: mbuf with external buffer (1)

- Currently, a mbuf embeds its data (direct), or references another mbuf (indirect)
- It could make sense to have mbuf referencing external memory
- Use cases: virtual drivers, server applications, storage, traffic generators

Discuss: mbuf with external buffer (2)

- Constraints: known paddr, physically contiguous, non-swappable
- A callback is required when the mbuf is freed
- Reference counting is managed by the application

Discuss: mbuf with external buffer



Offload

Discuss: TSO API and phdr checksum

- Currently in DPDK, to do TSO, one must:
  - Set PKT_TX_TCP_SEG flag
  - Set PKT_TX_IPV4 or PKT_TX_IPV6
  - Set IP checksum to 0 (IPv4)
  - Fill l2_len, l3_len, l4_len, tso_segz
  - Set the pseudo header checksum without taking ip length in account
- Need to fix the packet in case of virtio
- A real phdr checksum makes more sense, but it just moves the problem in other PMDs
- The tx_prep API may help here

Discuss: unify Rx/Tx offload fields

- In Rx, we have packet_type
  - Layer type for: l2, l3, l4, tunnel, inner_l2, inner_l3, inner_l4
  - Flags (checksums, vlan, ...)
- In Tx, we have lengths:
  - Lengths for: l2, l3, tso_segsz, outer_l2, outer_l3
  - Flags (checksums, TSO, vlan, ...)
- Is it possible to unify this information in one struct? (lengths are useful on Rx side)

Misc

Discuss: namespace

- Flags are not prefixed with RTE_
- Example: PKT_RX_VLAN_PKT
- This is something that could be changed, while keeping the compat during some versions

Discuss: constant mbuf headroom

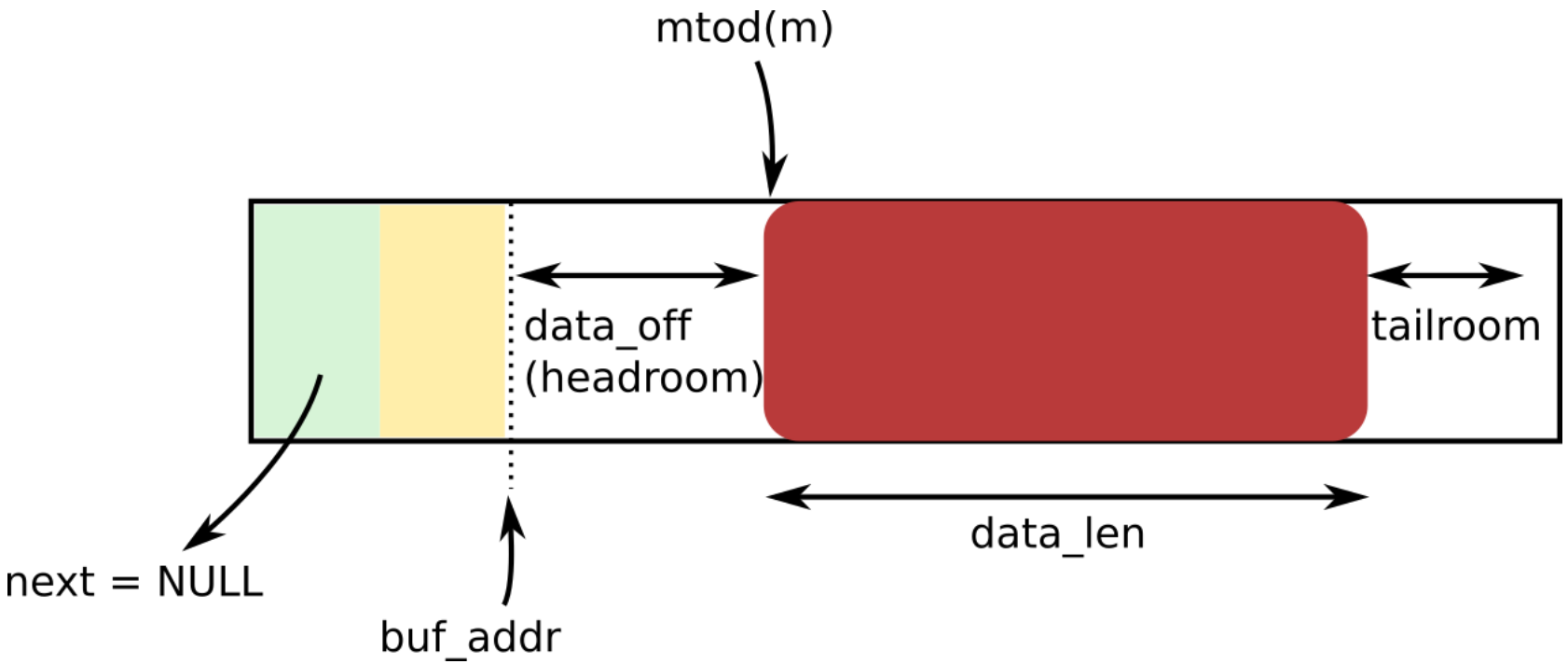
- The amount of headroom in a mbuf is fixed at compilation time: RTE_PKTMBUF_HEADROOM=128
- Depending on use cases, it can be either too large or too small
- Should we make it configurable at run-time?
- Or add rte_mbuf_reserve(headroom)?

Questions?

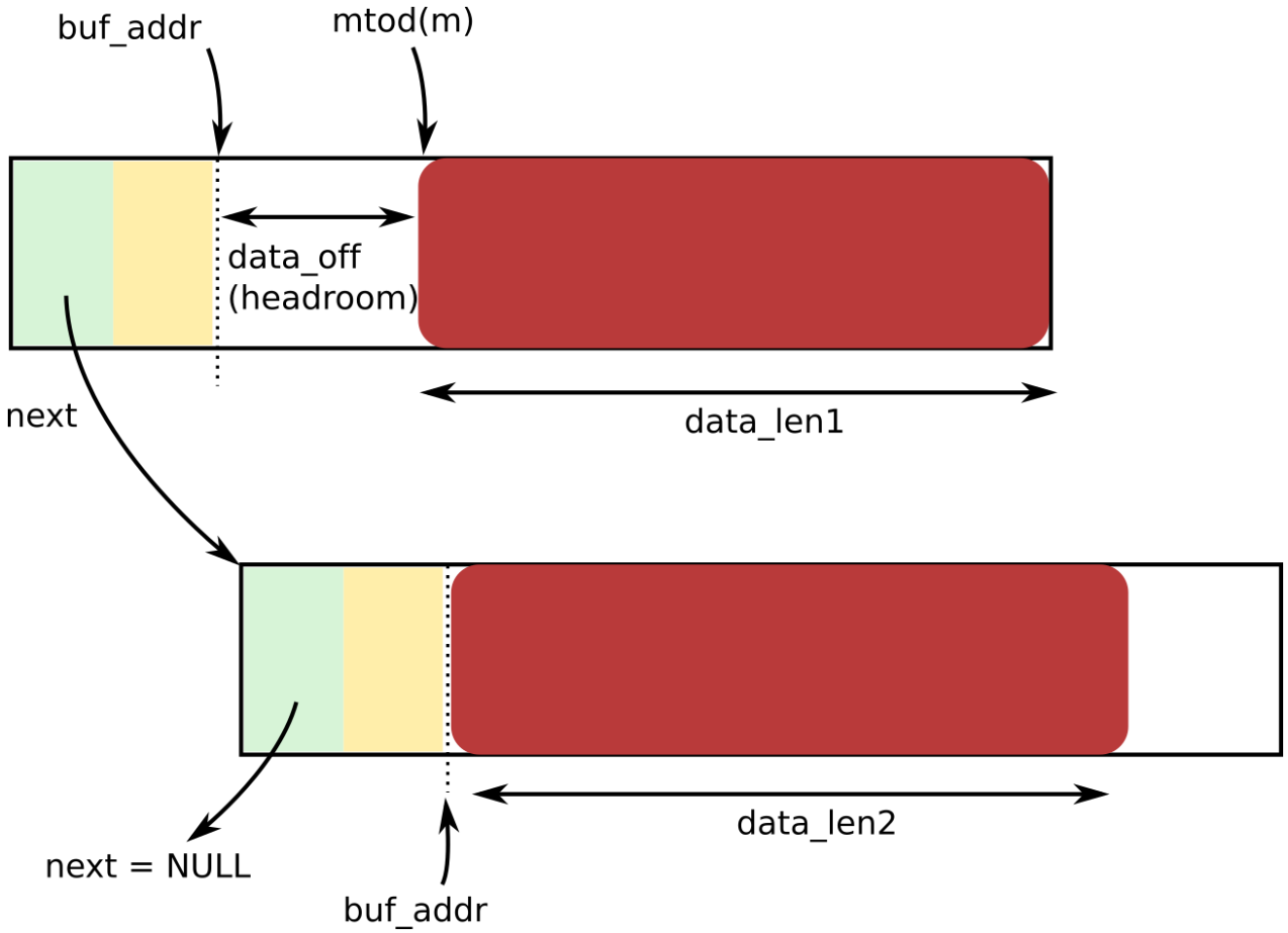
Olivier Matz

olivier.matz@6wind.com

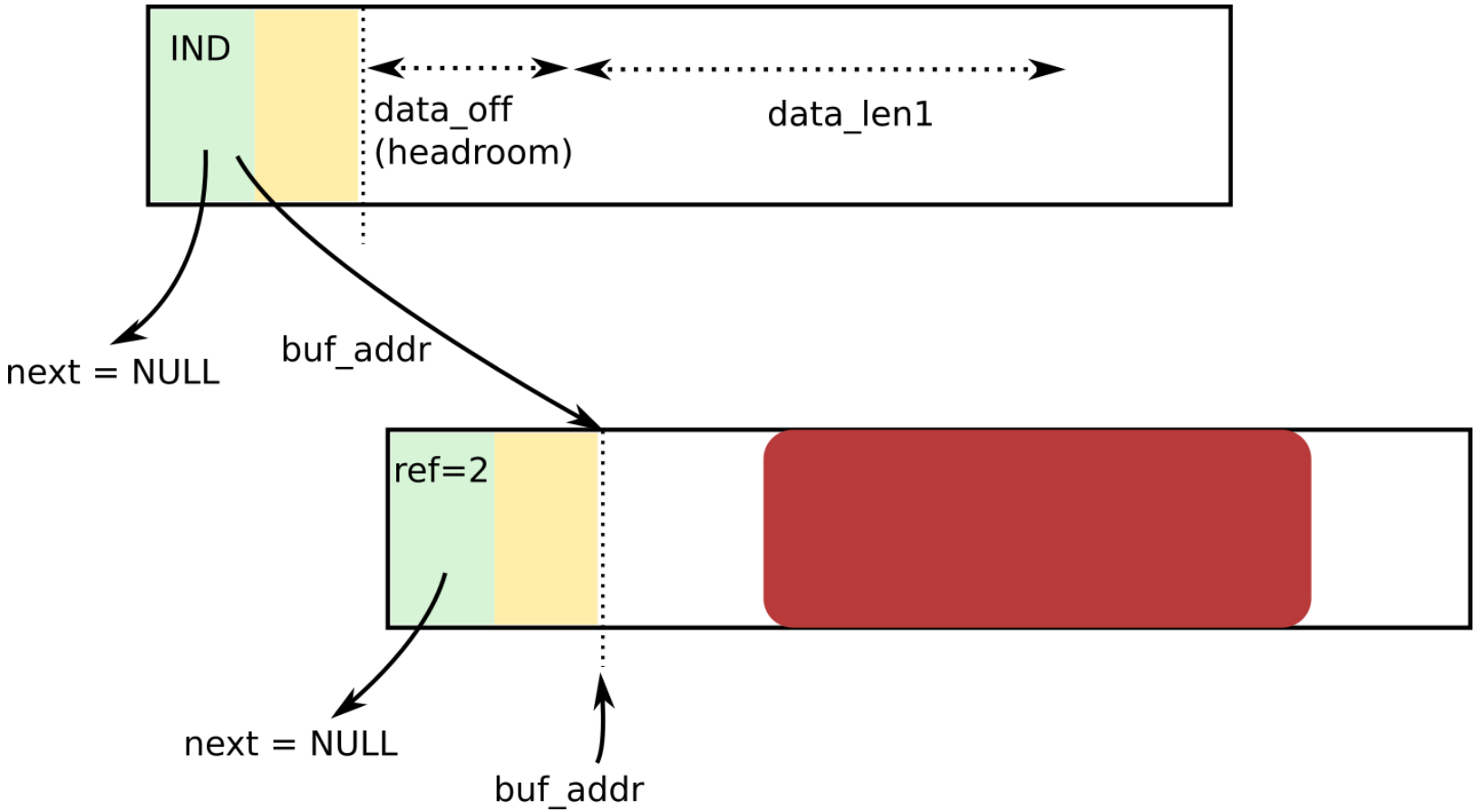
Appendix: mbuf



Appendix: mbuf chain



Appendix: mbuf clone



Appendix: mbuf structure


```
struct rte_mbuf {
        /* --- cacheline 0 boundary (64 bytes) --- */
        MARKER                        cacheline0;           /*     0     0 */
        void *                        buf_addr;             /*     0     8 */
        phys_addr_t                   buf_physaddr;         /*     8     8 */
        uint16_t                      buf_len;              /*    16     2 */
        MARKER8                       rearm_data;           /*    18     0 */
        uint16_t                      data_off;             /*    18     2 */
        uint16_t                      refcnt;               /*    20     2 */
        uint8_t                       nb_segs;              /*    22     1 */
        uint8_t                       port;                 /*    23     1 */
        uint64_t                      ol_flags;             /*    24     8 */
        MARKER                        rx_descriptor_fields1;/*    32     0 */
        uint32_t                      packet_type;          /*    32     4 */
        uint32_t                      pkt_len;              /*    36     4 */
        uint16_t                      data_len;             /*    40     2 */
        uint16_t                      vlan_tci;             /*    42     2 */
        uint64_t                      hash;                 /*    44     8 */
        uint32_t                      seqn;                 /*    52     4 */
        uint16_t                      vlan_tci_outer;       /*    56     2 */
        /* XXX 6 bytes hole, try to pack */
        /* --- cacheline 1 boundary (64 bytes) --- */
        MARKER                        cacheline1;           /*    64     0 */
        void *                        userdata;             /*    64     8 */
        struct rte_mempool *          pool;                 /*    72     8 */
        struct rte_mbuf *             next;                 /*    80     8 */
        uint64_t                      tx_offload;           /*    88     8 */
        uint16_t                      priv_size;            /*    96     2 */
        uint16_t                      timesync;             /*    98     2 */
};
```