DPDK Slab Allocator and zero-copy

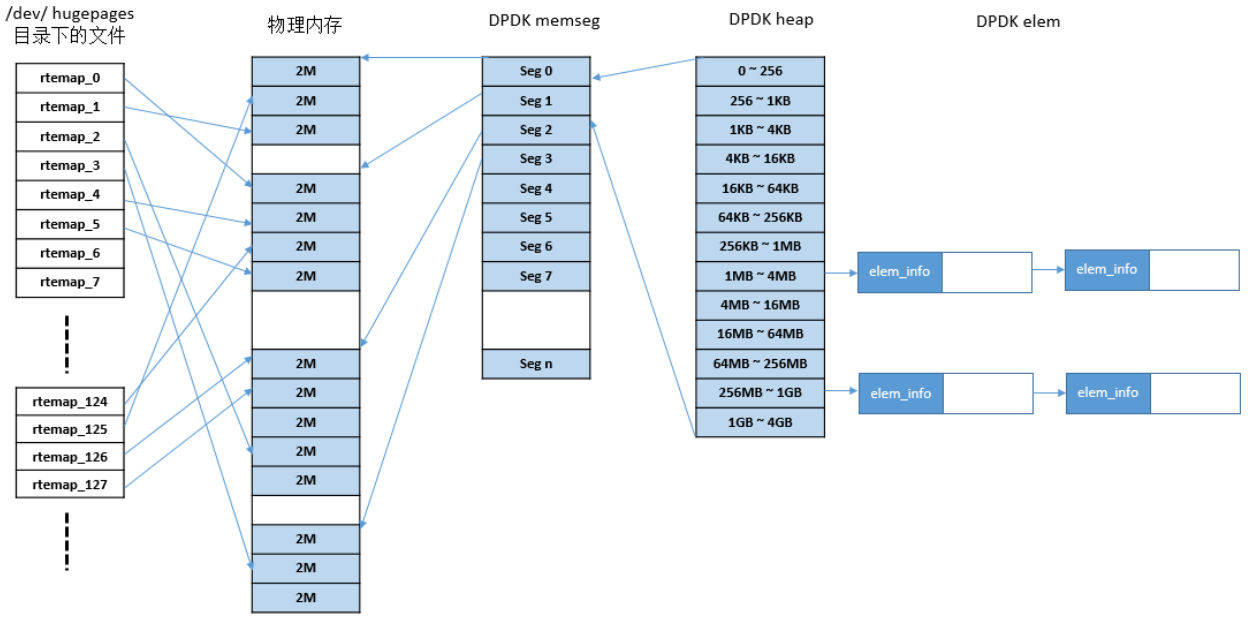
LINGJUN.ZHU
ALIBABA

Topics

- Why we need DPDK Slab?
- How does DPDK Slab work?
- Performance
- Applied it to zero-copy

Why we need DPDK Slab?

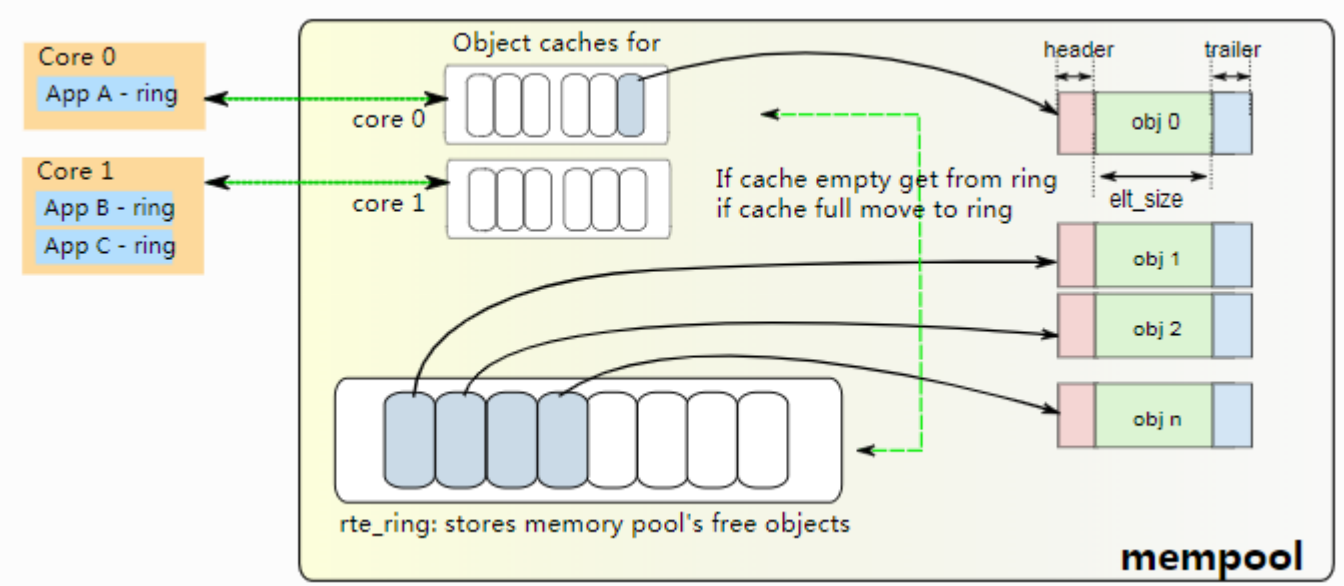
- rte_malloc / rte_free
  - ✓ unfixed-size buffer
  - ✓ spinlock for multi-thread
  - ✓ lowest performance




```
/*
 * Main function to allocate a block of memory from the heap.
 * It locks the free list, scans it, and adds a new memseg if the
 * scan fails. Once the new memseg is added, it re-scans and should return
 * the new element after releasing the lock.
 */
void *
malloc_heap_alloc(struct malloc_heap *heap,
        const char *type __attribute__((unused)), size_t size, unsigned flags,
        size_t align, size_t bound)
{
    struct malloc_elem *elem;

    size = RTE_CACHE_LINE_ROUNDUP(size);
    align = RTE_CACHE_LINE_ROUNDUP(align);

    rte_spinlock_lock(&heap->lock);

    elem = find_suitable_element(heap, size, flags, align, bound);
    if (elem != NULL) {
        elem = malloc_elem_alloc(elem, size, align, bound);
        /* increase heap's count of allocated elements */
        heap->alloc_count++;
    }
    rte_spinlock_unlock(&heap->lock);

    return elem == NULL ? NULL : (void *)(&elem[1]);
} ? end malloc_heap_alloc ?
```

Why we need DPDK Slab?

- Mempool
  - ✓ fixed-size buffer
  - ✓ No spinlock for multi-thread
  - ✓ Best performance




```
/**
 * Get several objects from the mempool.
 *
 * This function calls the multi-consumers or the single-consumer
 * version, depending on the default behaviour that was specified at
 * mempool creation time (see flags).
 *
 * If cache is enabled, objects will be retrieved first from cache,
 * subsequently from the common pool. Note that it can return -ENOENT when
 * the local cache and common pool are empty, even if cache from other
 * lcores are full.
 *
 * @param mp
 *   A pointer to the mempool structure.
 * @param obj_table
 *   A pointer to a table of void * pointers (objects) that will be filled.
 * @param n
 *   The number of objects to get from the mempool to obj_table.
 * @return
 *   - 0: Success; objects taken
 *   - -ENOENT: Not enough entries in the mempool; no object is retrieved.
 */
static __rte_always_inline int
rte_mempool_get_bulk(struct rte_mempool *mp, void **obj_table, unsigned n)
{
    struct rte_mempool_cache *cache;
    cache = rte_mempool_default_cache(mp, rte_lcore_id());
    return rte_mempool_generic_get(mp, obj_table, n, cache, mp->flags);
}


static __rte_always_inline void
rte_mempool_put_bulk(struct rte_mempool *mp, void * const *obj_table,
            unsigned n)
{
    struct rte_mempool_cache *cache;
    cache = rte_mempool_default_cache(mp, rte_lcore_id());
    rte_mempool_generic_put(mp, obj_table, n, cache, mp->flags);
}
```

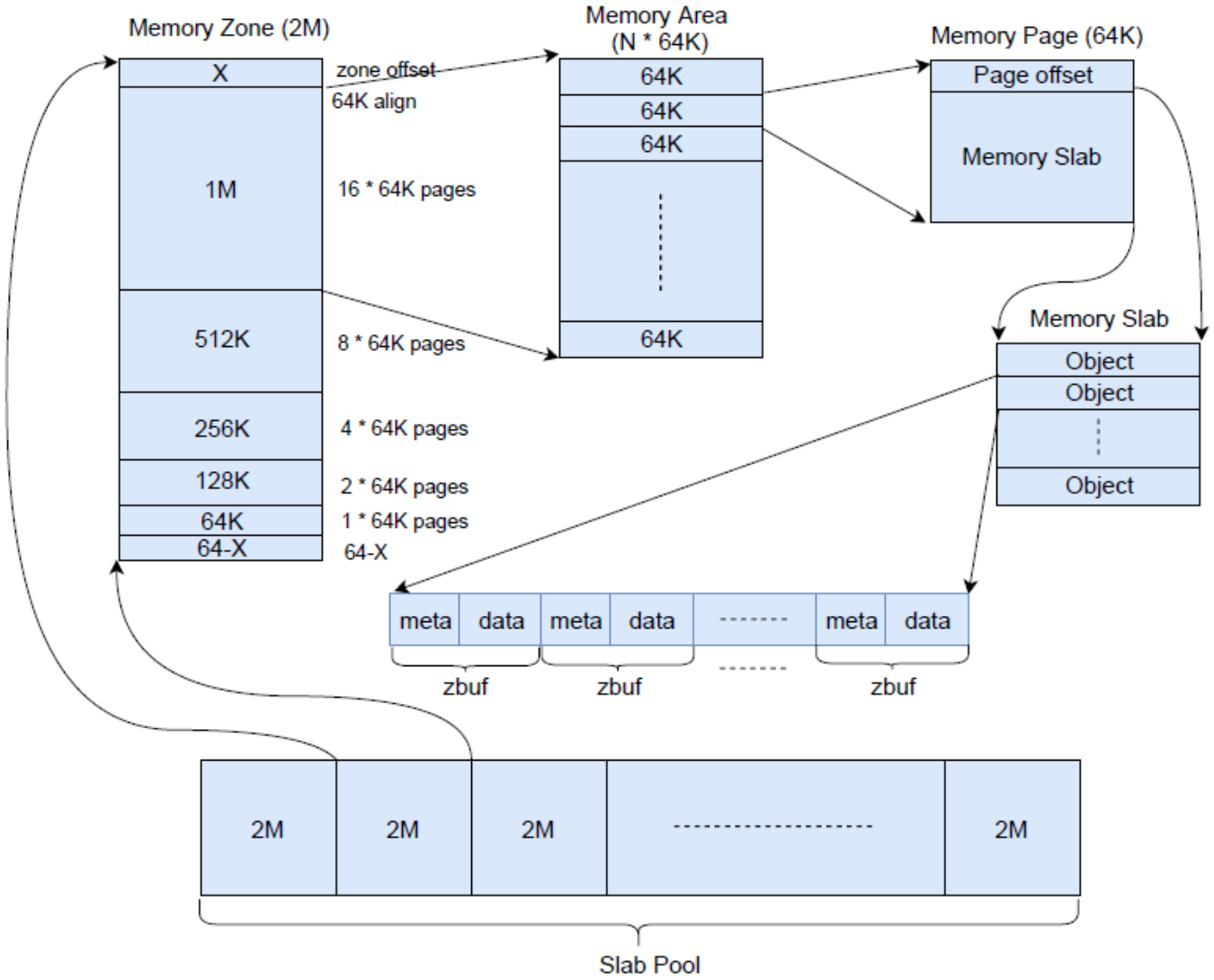
Why we need DPDK Slab?

- DPDK application’s requirements:
  - ✓ unfixed-size buffer
  - ✓ No spinlock for multi-thread
  - ✓ Best performance

How does DPDK Slab work?

- DPDK Slab supports the below features:
  - ✓ unfixed-size buffer
  - ✓ No spinlock for multi-thread
  - ✓ Meta information
  - ✓ Get meta info from any address

How does DPDK Slab work?



How does DPDK Slab work?

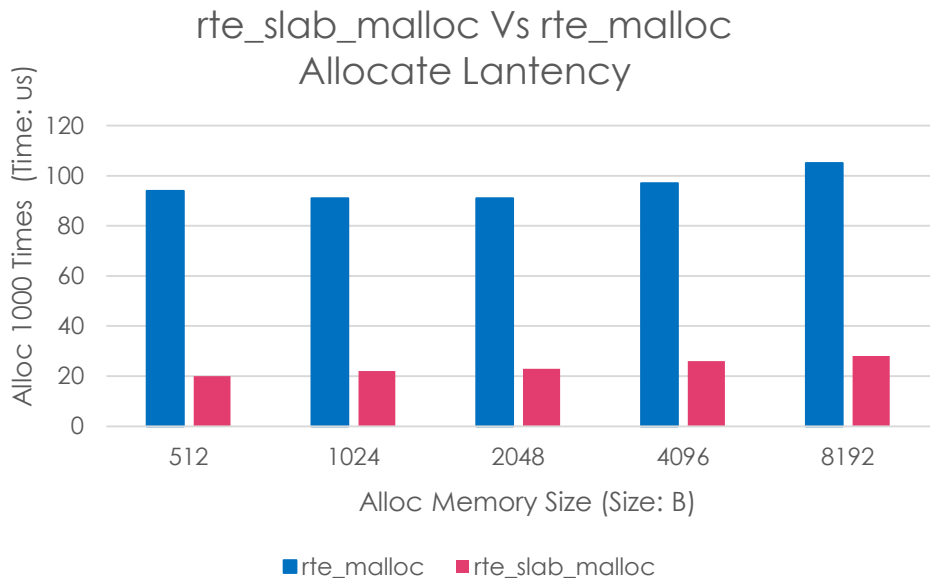
- DPDK Slab API interfaces:

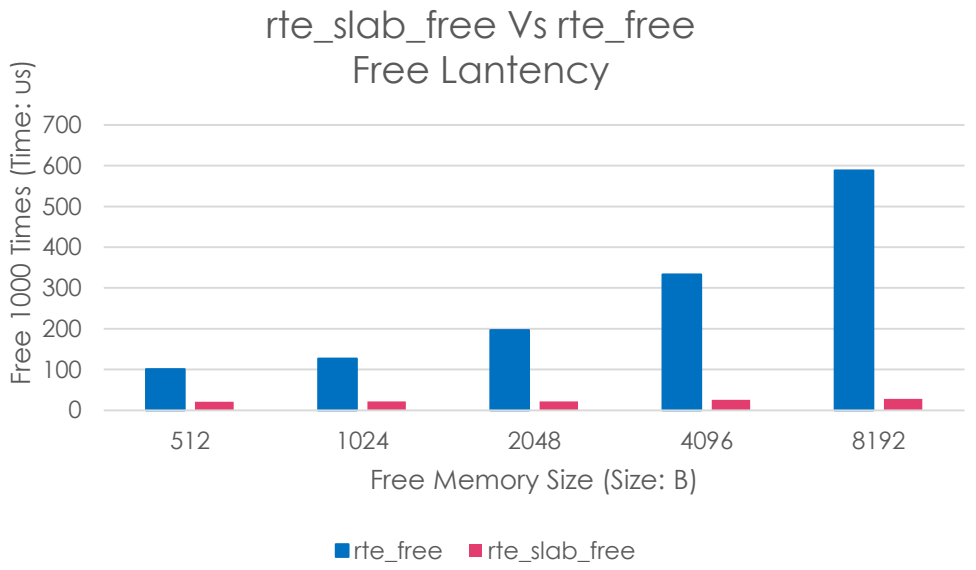
```
rte_slab_mem_pool_t *rte_slab_mem_pool_create(char *pool_name, size_t size, size_t align);

void rte_slab_mem_pool_destroy(rte_slab_mem_pool_t *pool);

void* rte_slab_mem_pool_alloc(rte_slab_mem_pool_t *pool, size_t size, size_t align);

void rte_slab_mem_pool_free(rte_slab_mem_pool_t *pool, void *p);
```

- DPDK zbuf API interfaces:

```
void *rte_zbuf_malloc(uint32_t size);

void rte_zbuf_free(void *ptr);

uint64_t rte_zbuf_virt2phy(void *ptr);

rte_zbuf_t *rte_zbuf_get(void *ptr);

rte_zbuf_t *rte_zbuf_get_by_header(void *ptr);

void rte_zbuf_inc(rte_zbuf_t *zbuf);

void rte_zbuf_dec(rte_zbuf_t *zbuf);

void rte_zbuf_ref(void *ptr, uint32_t size);

void rte_zbuf_unref(void *ptr);

int rte_zbuf_is(const void *ptr);

int rte_zbuf_is_reusable(void *ptr);
```

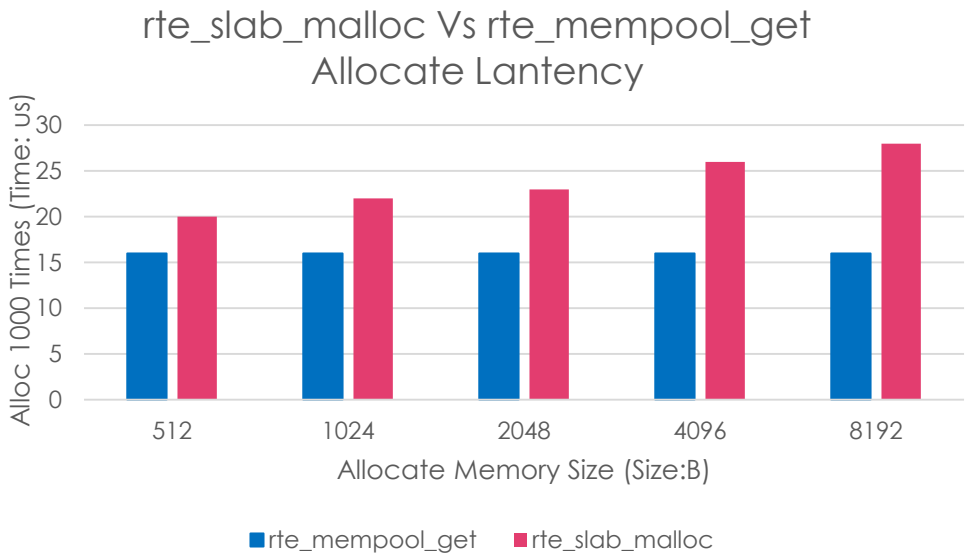
DPDK Slab Vs Rte_malloc

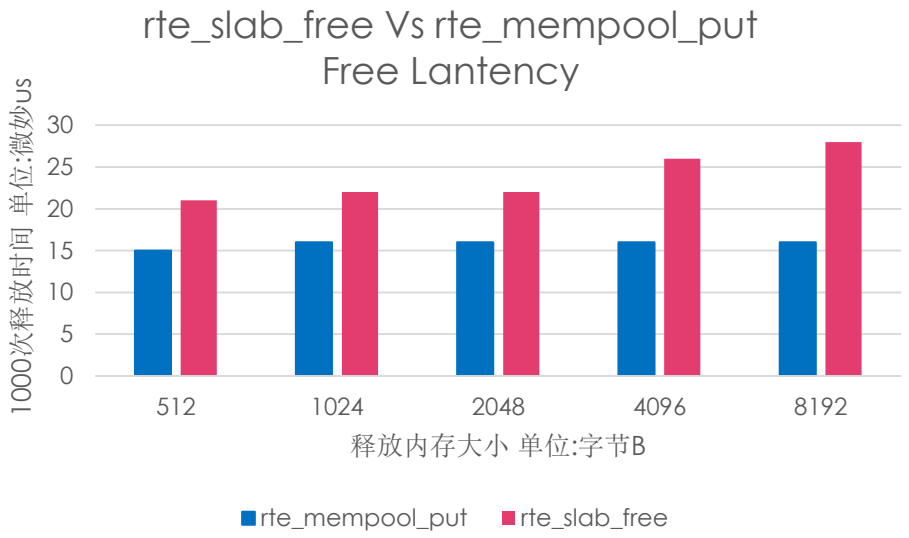




rte_slab_malloc better than rte_malloc: 75%
rte_slab_free better than rte_free: 90%

DPDK Slab Vs Mempool





Note: mempool no-local cache

rte_slab_malloc worse than rte_mempool_get: 27%
rte_slab_free worse than rte_mempool_put: 28%

Applied DPDK Slab to zero-copy

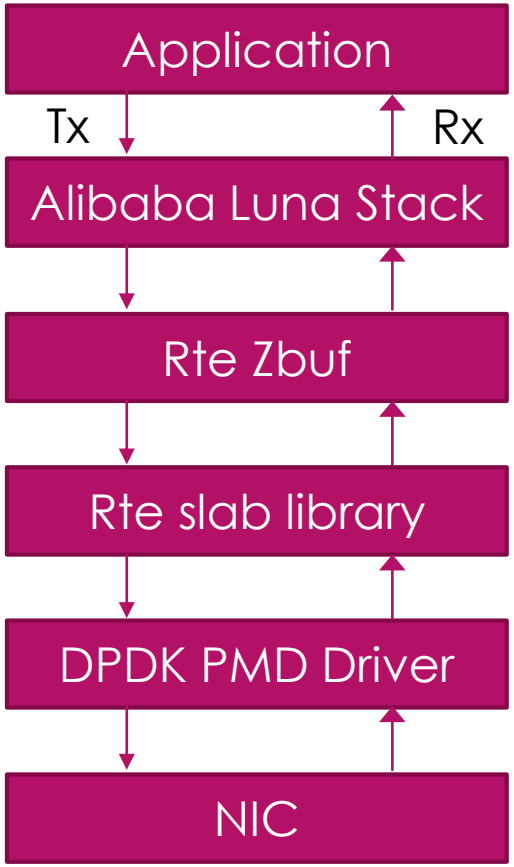
Zero Copy

- zero-copy from application, Luna Stack, DPDK to NIC.

It's not DPDK to NIC.

Latency

- 1/3 kernel TCP
- nearly as fast as RDMA



Q&A

DPDK
DATA PLANE DEVELOPMENT KIT

Thank You!