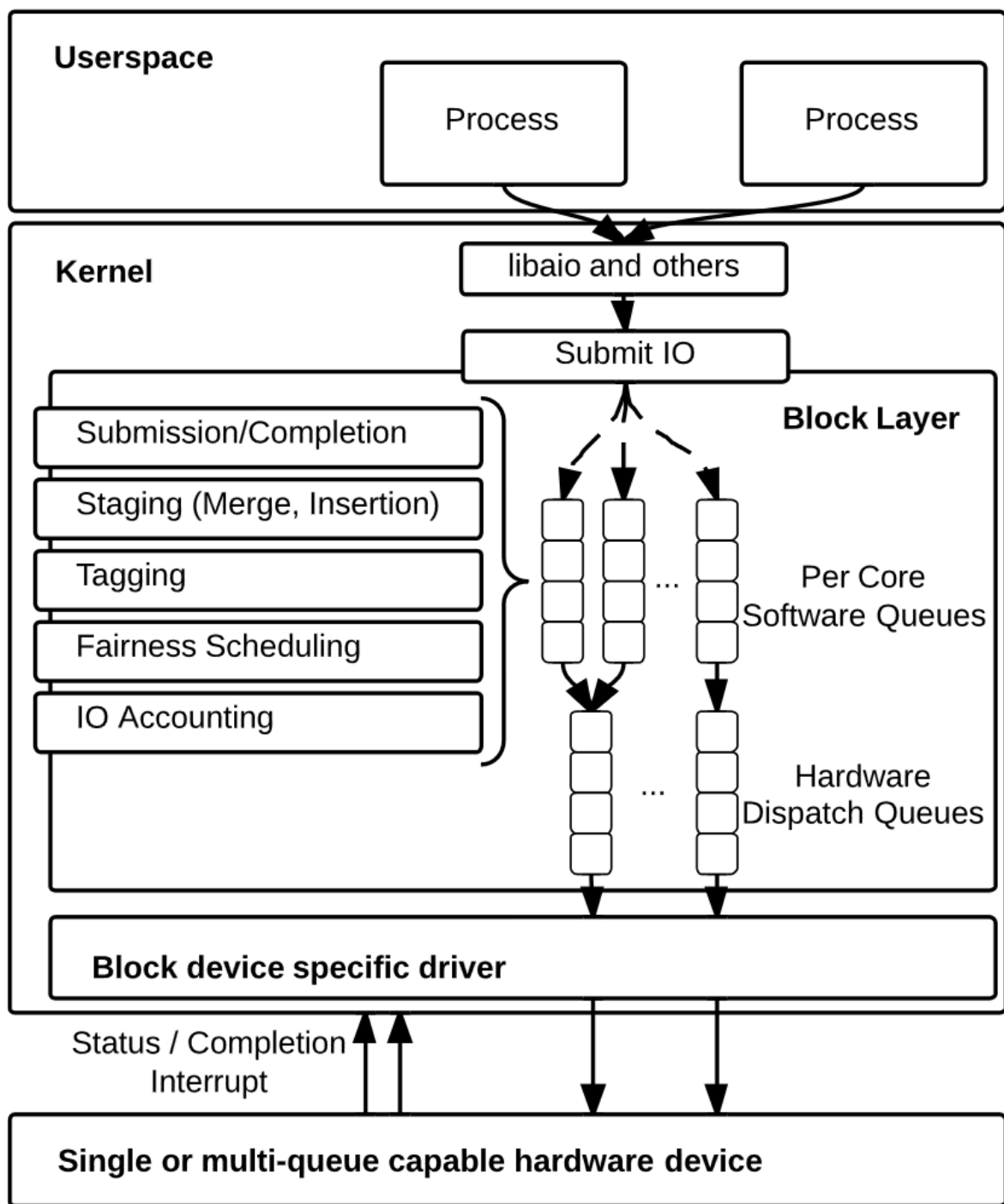


b1ock :: mq :: Request reference
counting

Agenda

- Block layer recap
- Proposed API
- Alternative API



Block layer stack

- m:n topology aiming for $m = n = \text{num_cpu}$.
- n limited by device capabilities.

Figure: "Linux Block IO: Introducing Multi-queue SSD Access on Multi-core Systems"

Block layer: `struct request`

- Request descriptors are allocated and initialized **during device setup**.
- Number of request structures == max outstanding IO requests.
- Requests allocated for and **tied to** each hardware queue.

Issuing IO

- To issue IO, C block layer allocates a **free request** structure and passes it to driver.
- Driver issues and completes commands with device, handing back request when done.
- Driver can touch the request between these two points.
- Request is *no touch* otherwise.

Requirements

- Drivers need to do DMA mapping of request data ➡ Must **keep request alive** for the duration of the mapping.
- Drivers may want to map requests to **multiple** hardware devices at once.
- Drivers may want to keep the `Request` alive for other reasons, such as **intrusive timers** in the private area of the `Request` structure.

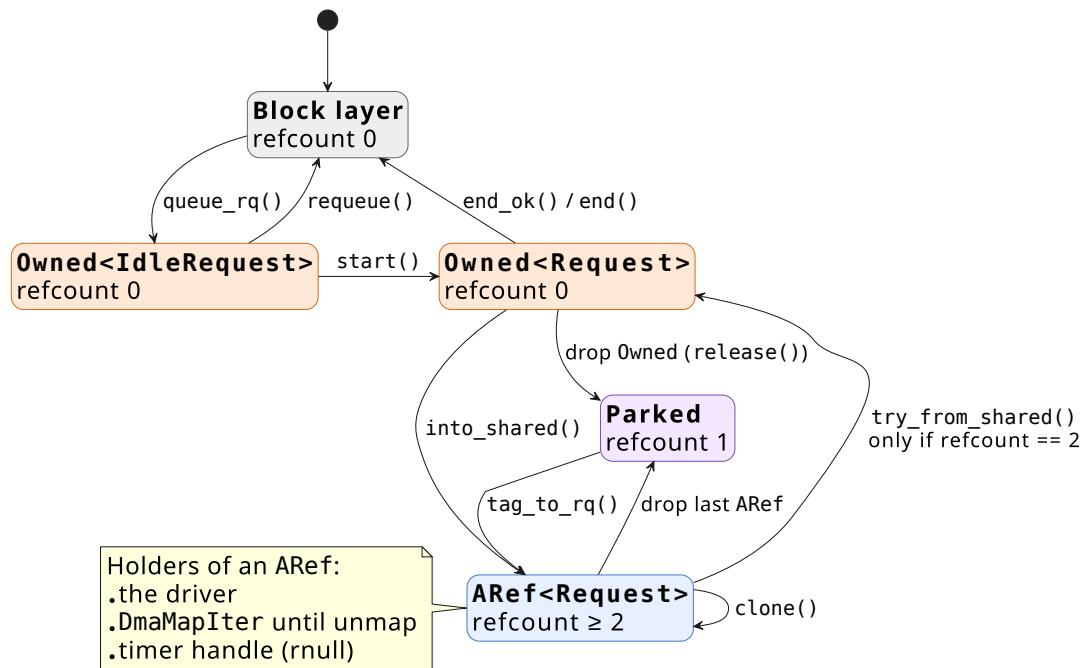
Suggested API

- Suggested Rust API Request states:
 - Owned by block layer (no touch from driver).
 - Uniquely owned by driver (during issue and complete).
 - Shared owned by driver.
 - Parked owned by driver.

Suggested API

- `queue_rq(rq: Owned<IdleRequest<T>>)` : After hand-off, the driver owns the request exclusively.
- Ending request requires ownership: `Owned<Request>::end_ok(self)` consumes the reference.
- Drivers need to do DMA mapping of request data ➡ Must keep request alive for the duration of the mapping.
 - Transition to `ARef<Request>` for the duration of the mapping.
 - Fallible transition back when mapping torn down.
- `Ownable::release()` need not free: `Request::release()` hands the request back to the block layer for reuse (**object cache**).

struct request state transitions



- refcount 0: block layer owns it, or the driver holds it uniquely.
- refcount 1: parked. Rust owns it, nobody references it, `tag_to_rq()` can fetch it.
- refcount ≥ 2 : shared. Number of `ARef` s + 1.
- `rnvme`, `rufs`: `DmaMapIter` holds an `ARef` for the lifetime of the DMA mapping. `try_from_shared()` succeeds only once the mapping is dropped.

Using ownership instead of counting

- No reference counting on the Rust side
 - `Request: !RefCounted`
- Pass `Owned<Request>` to driver.
- Driver must store `Owned<Request>` somewhere.
 - Per `hwctx` array makes sense

Using ownership instead of counting

- As `hwctx` are shared between threads, we must synchronize access.
- Locking the entire array is not efficient.
- So we must use something like `Vec<Lock<Option<Owned<Request>>>>`
- But we need to share the request:
 - `Vec<Lock<Option<Arc<Owned<Request>>>>`
- But we `Arc<Owned<Request>>` is not an exclusive, so we need `into_unique`
 - Full circle.
- Conceptually the same set of states as before.
- Inefficient, and the array already exist in C code.

```
static inline struct request *blk_mq_tag_to_rq(struct blk_mq_tags *tags,
                                                unsigned int tag)
{
    if (tag < tags->nr_tags) {
        prefetch(tags->rqs[tag]);
        return tags->rqs[tag];
    }

    return NULL;
}
```