

8

44

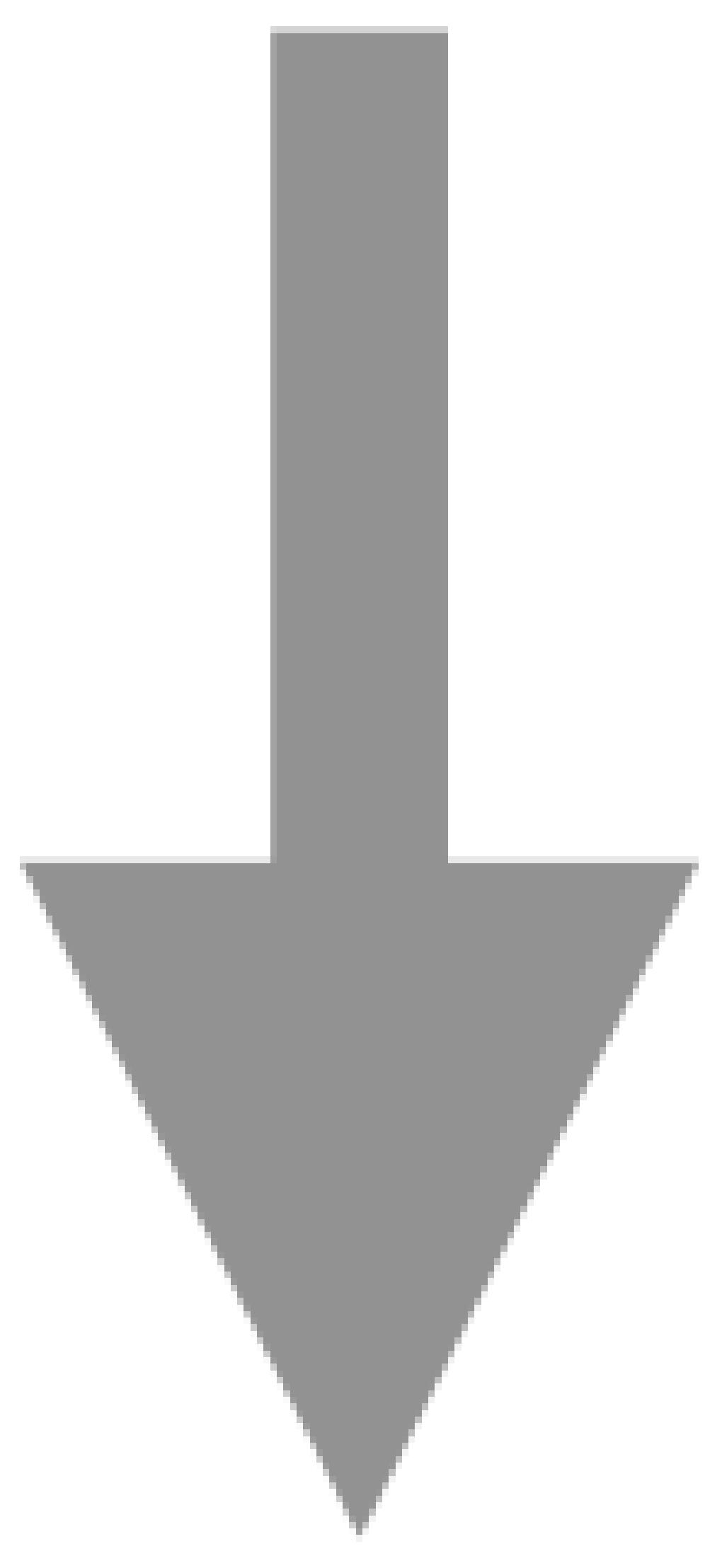
45

tail/

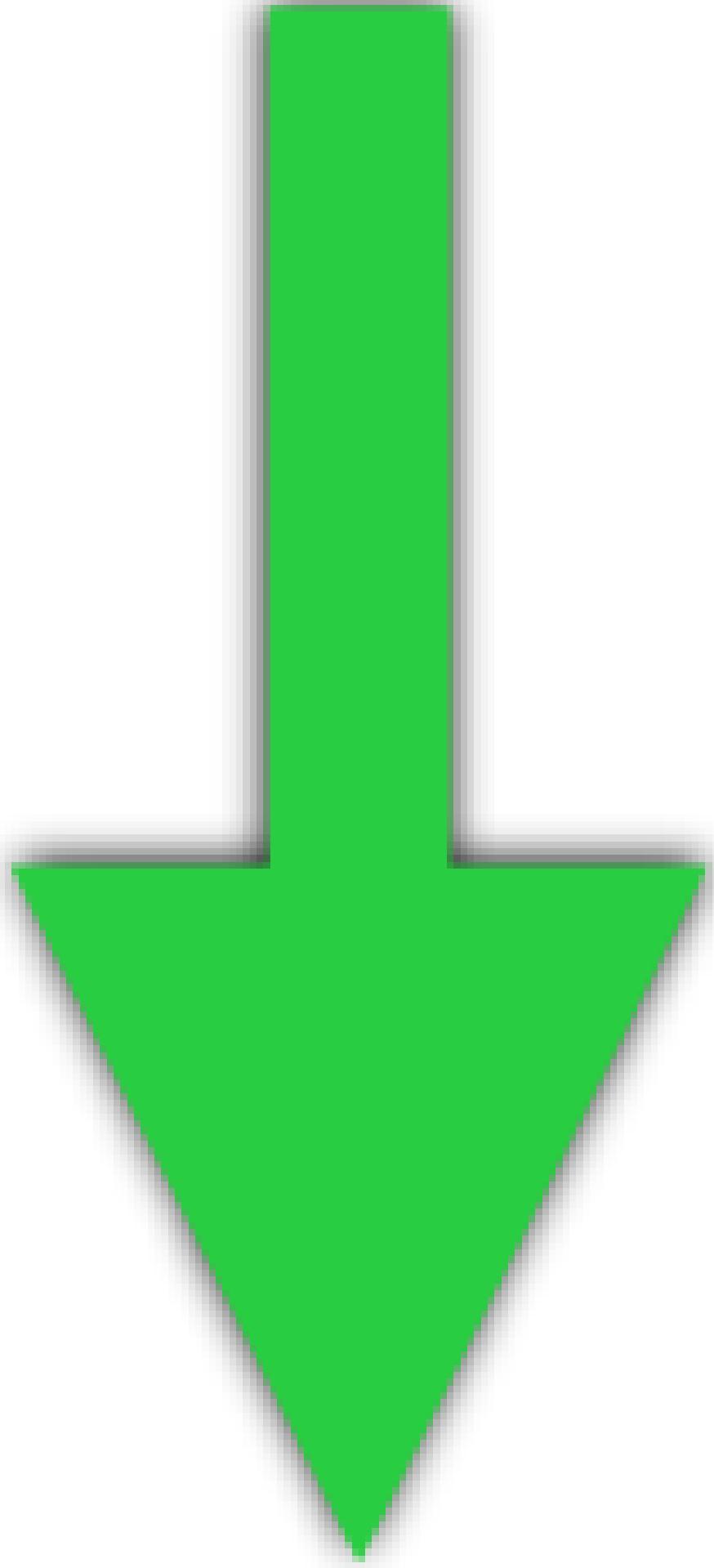
write

head/

read



46



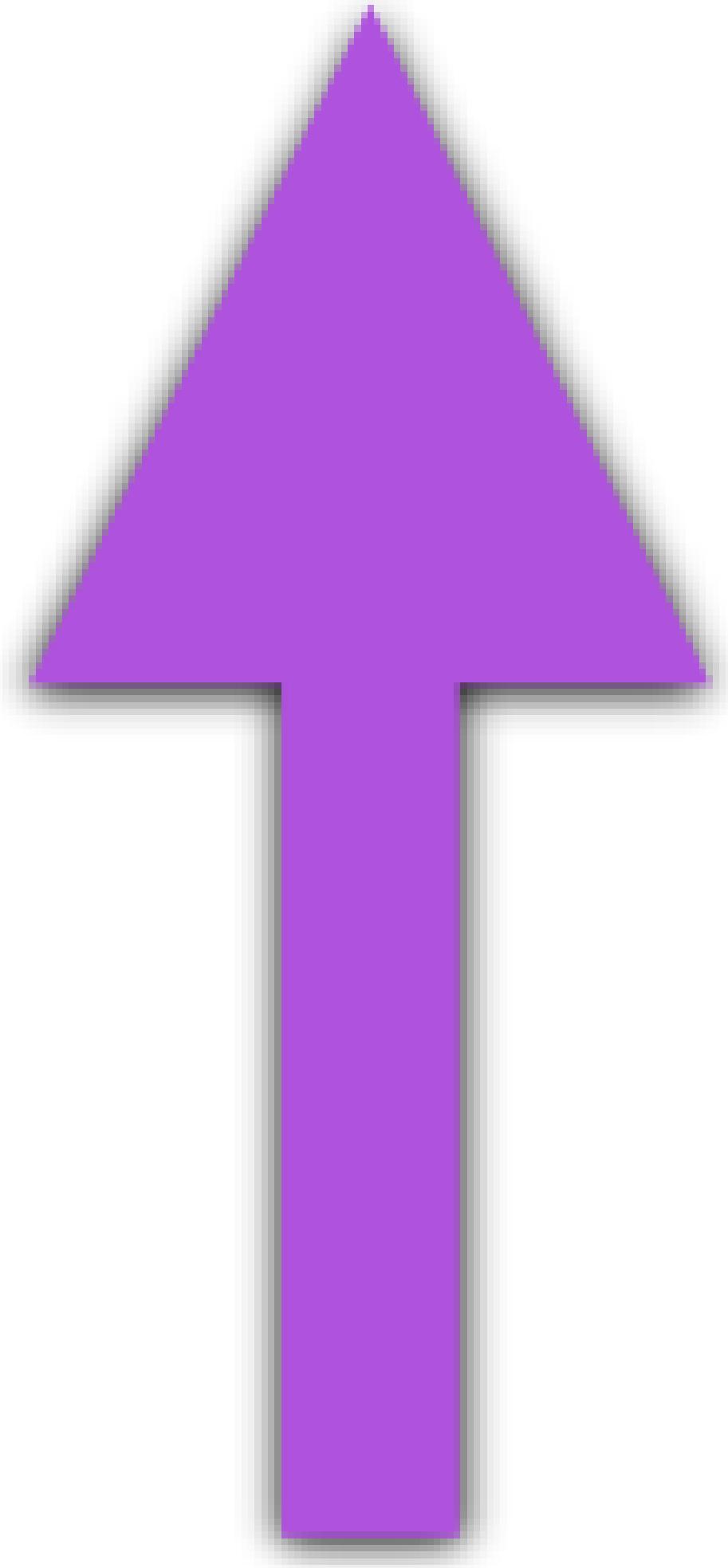


47

44

48





free:  $(\text{cap} - 1) - (\text{tail} - \text{head}) =$

$$(4 - 1) - (6 - 4) =$$

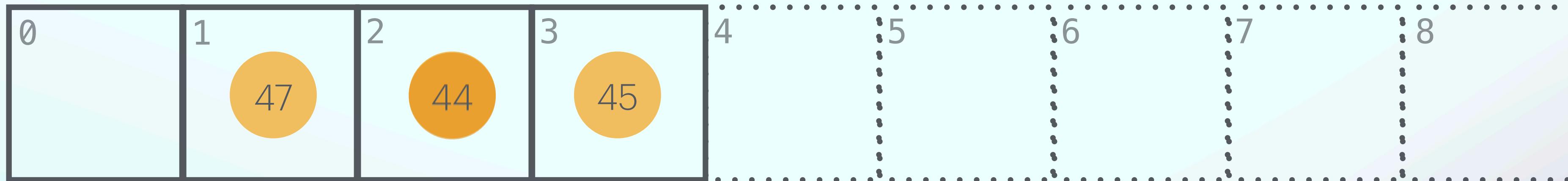
$$3 - 2 =$$

**1**



$$\begin{aligned}\text{free: } (\text{cap} - 1) - (\text{tail} - \text{head}) &= \\ (4 - 1) - (6 - 4) &= \\ 3 - 2 &= \\ &\mathbf{1}\end{aligned}$$

tail/  
write



head/  
read

```

bool try_pop (T& v)
{
    for (;;)
    {
        size_t current_pending = pending_head.load (std::memory_order_acquire);
        size_t current_tail = tail.load (std::memory_order_acquire);

        if (current_pending == current_tail) // empty
            return false;

        size_t index = current_pending & (capacity - 1);

        // Try to claim this slot atomically
        if (pending_head.compare_exchange_weak (current_pending, current_pending + 1,
                                                std::memory_order_acquire,
                                                std::memory_order_relaxed))
        {
            // Successfully claimed the slot, now read the data
            v = data[index];

            // Now we need to commit this read
            // Wait until it's our turn to commit (all previous reads have committed)
            for (;;)
            {
                size_t expected_committed = current_pending;
                if (committed_head.compare_exchange_weak(expected_committed,
                                                          current_pending + 1,
                                                          std::memory_order_release,
                                                          std::memory_order_relaxed))
                {
                    // Successfully committed
                    return true;
                }

                // If CAS failed, it means we're not next in line yet
                // Keep spinning until committed_head catches up to our position
            }
        }

        // CAS failed, another consumer claimed it. Retry.
    }
}

```

**private:**

```

size_t capacity = 0;
alignas(hardware_destructive_interference_size) std::atomic<size_t> pending_head { 0 };
alignas(hardware_destructive_interference_size) std::atomic<size_t> committed_head { 0 };
alignas(hardware_destructive_interference_size) std::atomic<size_t> tail { 0 };
std::vector<T> data { std::vector<T> (capacity) };

```