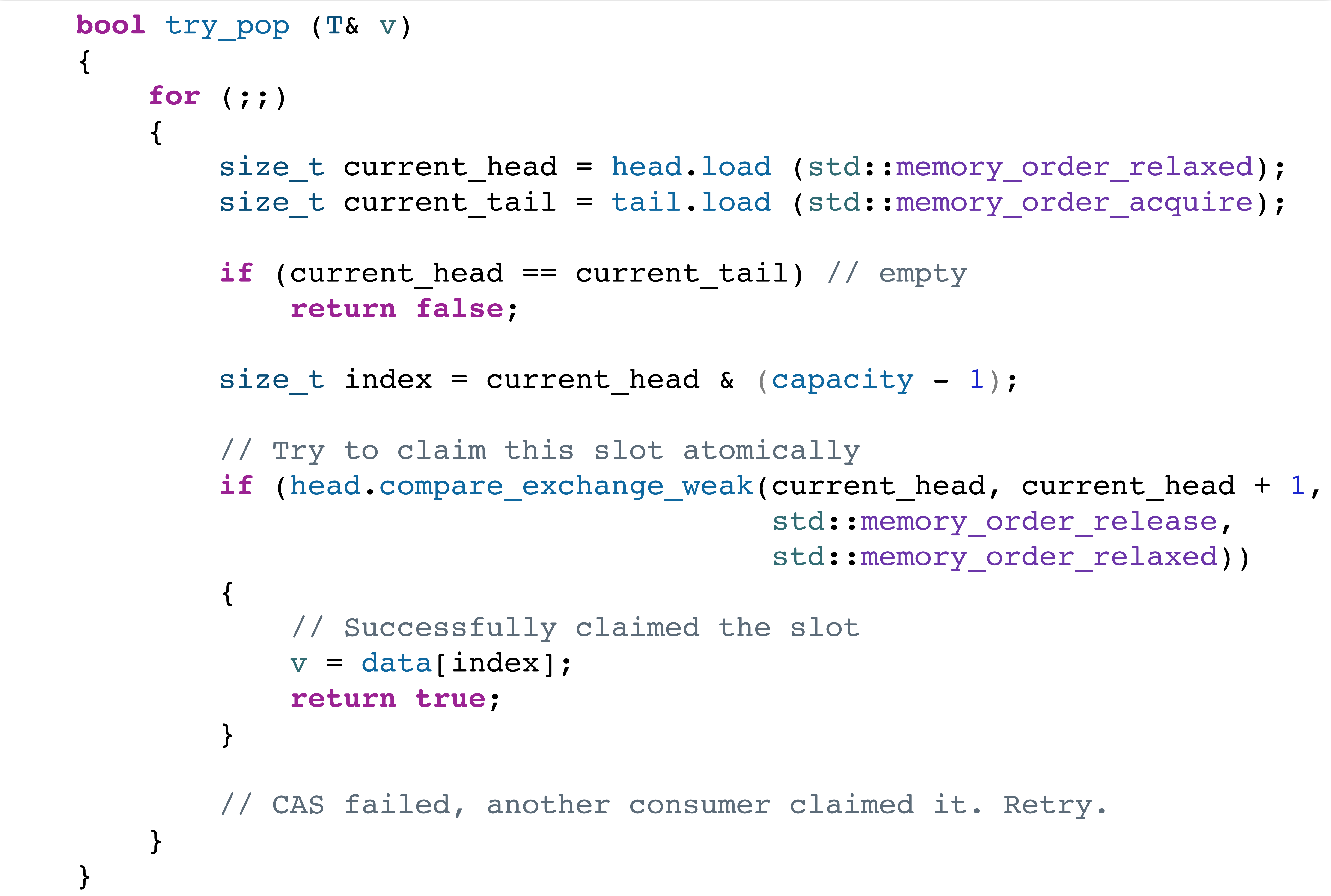
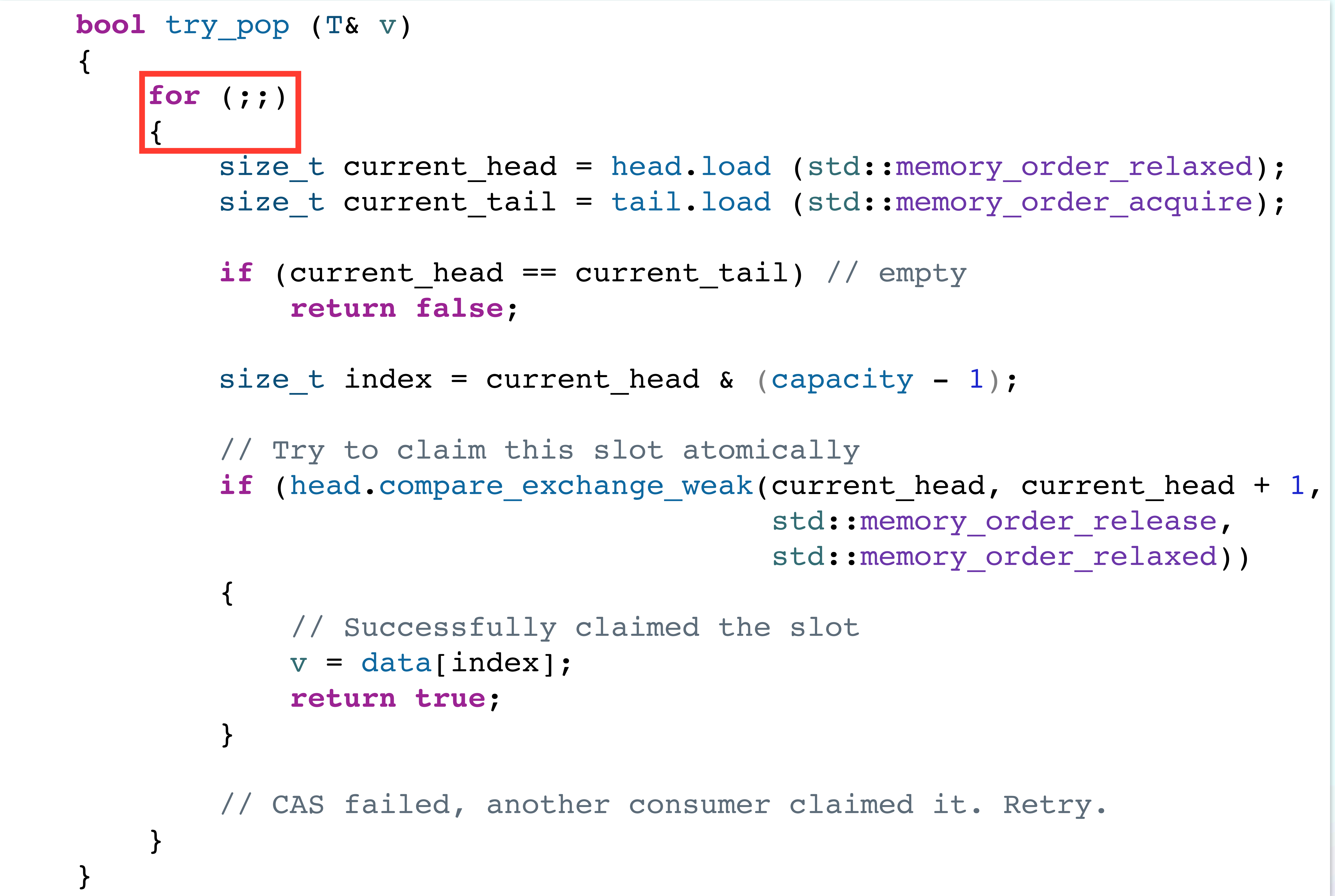
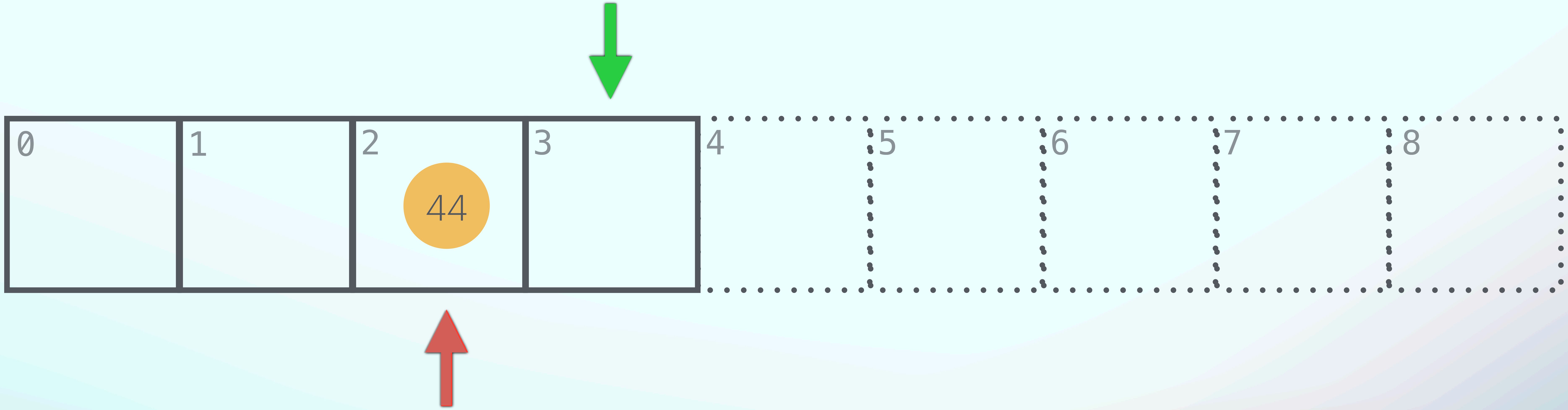


```
    bool try_pop (T& v)
    {
        for (;;)
        {
            size_t current_head = head.load (std::memory_order_relaxed);
            size_t current_tail = tail.load (std::memory_order_acquire);

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

            size_t index = current_head & (capacity - 1);

            // Try to claim this slot atomically
            if (head.compare_exchange_weak(current_head, current_head + 1,
                                           std::memory_order_release,
                                           std::memory_order_relaxed))
            {
                // Successfully claimed the slot
                v = data[index];
                return true;
            }

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



```
bool try_pop (T& v)
{
    for (;;)
    {
        size_t current_head = head.load (std::memory_order_relaxed);
        size_t current_tail = tail.load (std::memory_order_acquire);

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

        size_t index = current_head & (capacity - 1);

        // Try to claim this slot atomically
        if (head.compare_exchange_weak(current_head, current_head + 1,
                                       std::memory_order_release,
                                       std::memory_order_relaxed))
        {
            // Successfully claimed the slot
            v = data[index];
            return true;
        }

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

tail/
write



head/
read