```cpp
template<typename T>
class drow_queue_v5
{
public:
    drow_queue_v5 (size_t capacity_)
        : capacity (std::bit_ceil (capacity_))
    {}

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

        size_t size = current_tail - current_head;

        if (size >= (capacity - 1)) // full
            return false;

        size_t index = current_tail & (capacity - 1);
        data[index] = v;
        tail.store (current_tail + 1, std::memory_order_release);

        return true;
    }

    bool try_pop (T& v)
    {
        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);
        v = data[index];
        head.store (current_head + 1, std::memory_order_release);

        return true;
    }

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

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


head

tail

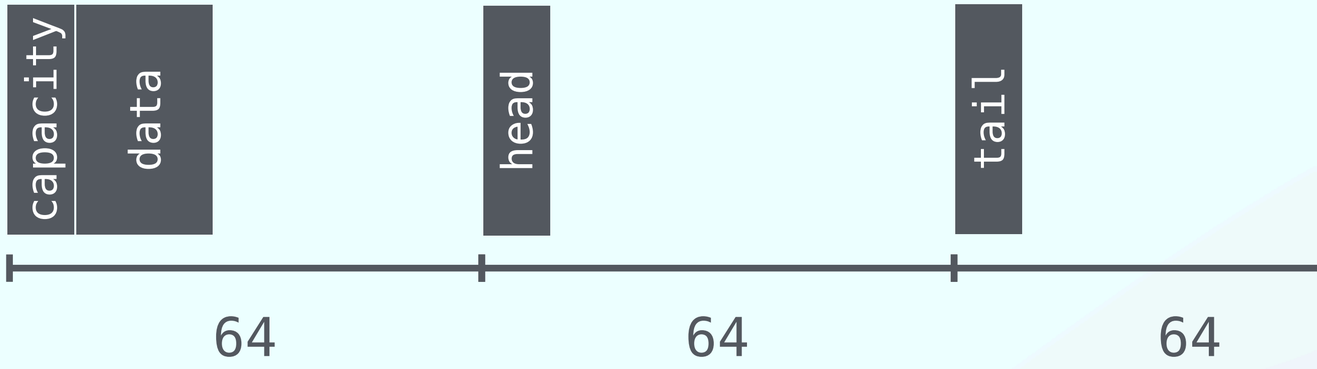
capacity

data


64
64
64

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

```cpp
template<typename T>
class drow_queue_v5
{
public:
    drow_queue_v5 (size_t capacity_)
        : capacity (std::bit_ceil (capacity_))
    {}

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

        size_t size = current_tail - current_head;

        if (size >= (capacity - 1)) // full
            return false;

        size_t index = current_tail & (capacity - 1);
        data[index] = v;
        tail.store (current_tail + 1, std::memory_order_release);

        return true;
    }

    bool try_pop (T& v)
    {
        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);
```




```cpp
private:
    size_t capacity = 0;
    std::vector<T> data { std::vector<T> (capacity) };
    alignas(hardware_destructive_interference_size) std::atomic<size_t> head { 0 };
    alignas(hardware_destructive_interference_size) std::atomic<size_t> tail { 0 };
```

5.00E+08
4.00E+08
3.00E+08
2.00E+08
1.00E+08
0.00E+00
mutex_queue
realtime_mutex_queue
drow_spsc_v1
drow_spsc_v2
drow_spsc_v3
drow_spsc_v4
drow_spsc_v5