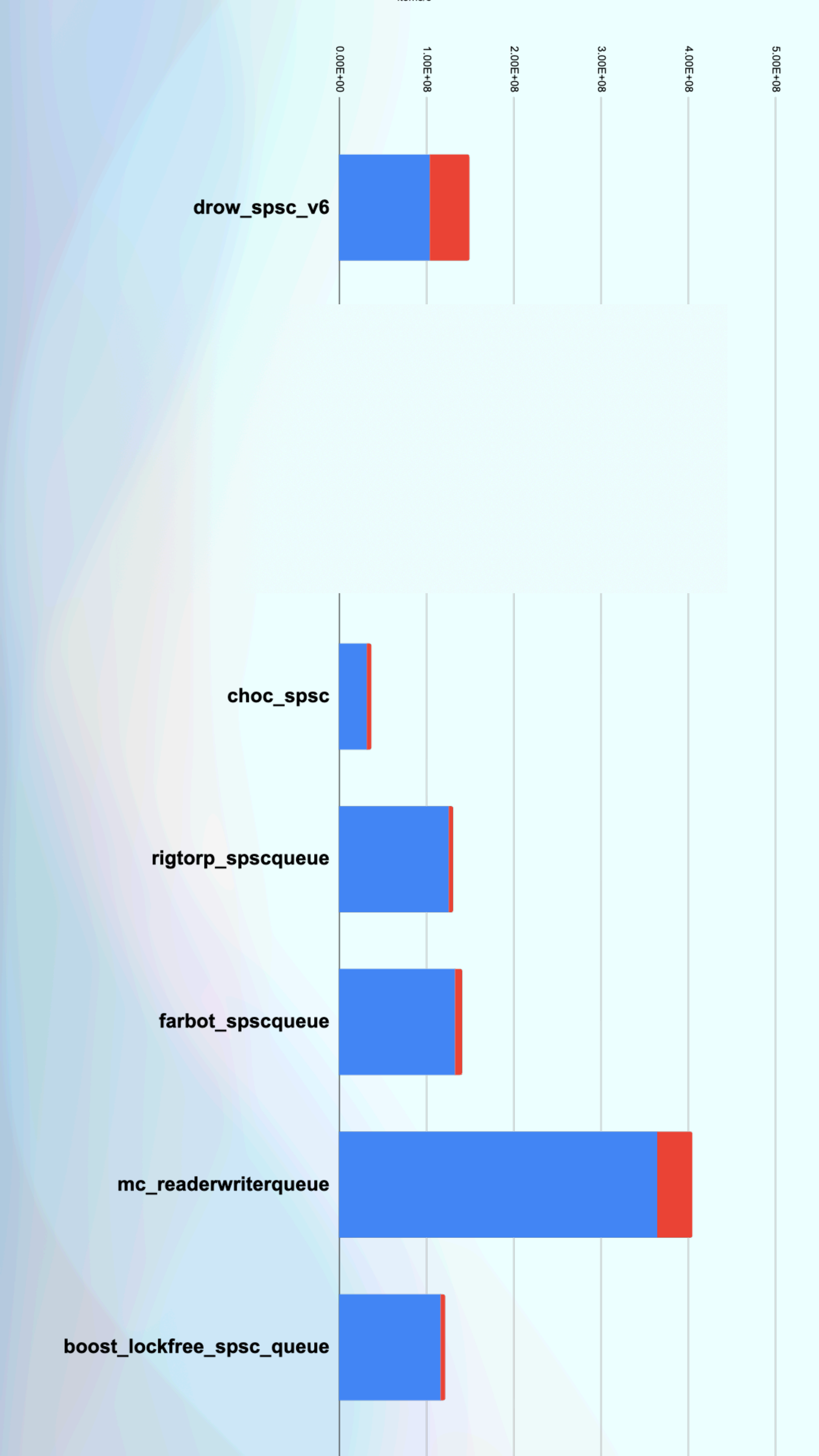
0.00E+00
1.00E+08
2.00E+08
3.00E+08
4.00E+08
5.00E+08
drow_spsc_v6
choc_spsc
rigtorp_spscqueue
farbot_spscqueue
mc_readerwriterqueue
boost_lockfree_spsc_queue

```cpp
template<typename T>
class drow_queue_v8
{
public:
    drow_queue_v8 (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 size = current_tail - cached_head;

        if (size >= (capacity_minus_one)) // full
        {
            cached_head = head.load (std::memory_order_relaxed);

            size = current_tail - cached_head;

            if (size >= (capacity_minus_one))
                return false;
        }

        size_t index = current_tail & (capacity_minus_one);
        data[index] = v;

        std::atomic_thread_fence (std::memory_order_acq_rel);

        tail.store (current_tail + 1, std::memory_order_relaxed);

        return true;
    }

private:
    size_t capacity = 0;
    size_t capacity_minus_one = capacity - 1;
    alignas(hardware_destructive_interference_size) std::atomic<size_t> head { 0 };
    alignas(hardware_destructive_interference_size) size_t cached_tail { 0 };
    alignas(hardware_destructive_interference_size) std::atomic<size_t> tail { 0 };
    alignas(hardware_destructive_interference_size) size_t cached_head { 0 };
    std::vector<T> data { std::vector<T> (capacity) };
};
```

```cpp
bool try_pop (T& v)
{
    size_t current_head = head.load (std::memory_order_relaxed);

    if (current_head == cached_tail) // empty
    {
        cached_tail = tail.load (std::memory_order_relaxed);
        std::atomic_thread_fence (std::memory_order_acquire);

        if (current_head == cached_tail) // empty
            return false;
    }

    size_t index = current_head & (capacity_minus_one);
    v = data[index];

    head.store (current_head + 1, std::memory_order_relaxed);

    return true;
}
```