```cpp
template<typename T>
class drow_queue_v6
{
public:
    drow_queue_v6 (size_t capacity_)
        : capacity (std::bit_ceil (capacity_))
    {}

    bool try_push (const T&);
    bool try_pop (T&);

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) 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 };
};
```

```
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_acquire);

        if (current_head == cached_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;
}
```

```
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 - 1)) // full
    {
        cached_head = head.load (std::memory_order_acquire);
        size = current_tail - cached_head;

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

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

    return true;
}
```

```cpp
template<typename T>
class drow_queue_v6
{
public:
    drow_queue_v6 (size_t capacity_)
        : capacity (std::bit_ceil (capacity_))
    {}

    bool try_push (const T&);
    bool try_pop (T&);

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) 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 };
};
```

```
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) 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 };
};
```

```
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 - 1)) // full
    {
        cached_head = head.load (std::memory_order_acquire);
        size = current_tail - cached_head;

        if (size >= (capacity - 1))
            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);

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

        if (current_head == cached_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;
}
```

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_v5
drow_spsc_v6
choc_spsc
rigtorp_spscqueue
farbot_spscqueue
mc_readerwriterqueue
boost_lockfree_spsc_queue