Advanced Behaviour

MPSC/MPMC

push	Block on full*	Overwrite when full	Fail/try_
pop	Block on empty*	Return default on empty	Fail/try_
capacity	Static (compile-time)	Fixed (runtime)	Dynamic (grows)
Bulk push/pop	No - single item	Yes - multiple items	
Message size	Fixed	Dynamic	
Gurantees	None - blocking	Lock-free	Wait-free
Message size limit	Limited (8 bytes?)	Unlimted	
Triviality	Trival	Non-trivial	
Num processes	Single	Inter-process	
Threads	Multiple-producers	Multiple-consumers	*Max-num threads*
Serialisation	Strict global order	Relaxed	

```cpp
template<typename T>
class drow_spmc_v5
{
public:
    drow_spmc_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;
    }

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

```cpp
    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.
        }
    }
```