Concepts for Concurrent Queues

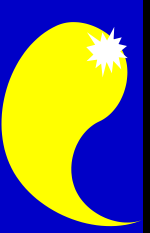
P0260R21

LEWG Telecon

August, 2026

Detlef Vollmann

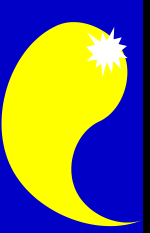
vollmann engineering gmbh



Design Space

- The design space for concurrent queues is pretty big
  - partly in interface design
  - more in semantics
- Single or multiple connections on producer and/or consumer side
- Lock-free vs. locking
  - separate for both ends
- Memory allocation
  - up-front, per push/pop, external
- Ordering guarantee
  - FIFO vs. priorities
- Non-blocking only vs. synchronous interface
- Single push/pop vs. two-phase
- Strongly typed vs. (dynamically sized) byte chunks

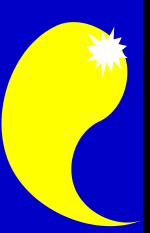
 Copyright ©1995-2026, Detlef Vollmann



Design Space

- More interface
  - timed waits
  - asynchronous
  - debugging
  - single ended interfaces
- Efficiency vs. robust/portable interface
- Error handling (exceptions)
- Concurrency vs. parallelism vs. asynchronicity

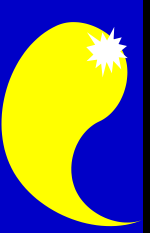
 Copyright ©1995-2026, Detlef Vollmann



History

- N3353, mailing 2012-01
  - had already (several) concepts
    - as specification (named requirements)
  - had a separate `buffer_queue` that implemented them
- SG1 discussed heavily what's in the concepts, and what's in the concrete queue (later named `bounded_queue`)
  - sequential consistent memory ordering in `bounded_queue`, but not in concept
  - FIFO ordering in `bounded_queue`, not in concepts
  - memory management specified for `bounded_queue`, not in concepts

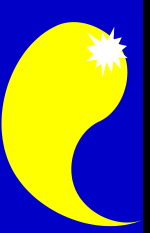
 Copyright ©1995-2026, Detlef Vollmann



History

- P0260R9, mailing 2024-05
  - discussed by SG1 in St.Louis 2024
  - sent to LEWG for IS: SF F N A SA: 4 5 1 0 0
  - with three named `concepts`
- P0260R10, mailing 2024-07
  - discussed by LEWG in St.Louis 2024
  - Poll: Remove the user-facing concepts from P0260R10 (the author is free to use exposition-only concepts or named type requirements to achieve the same effect)
  - SF F N A SA: 2 8 5 0 0
  - with named `concepts`
- P0260R11, mailing 2024-11 had exposition-only concepts

 Copyright ©1995-2026, Detlef Vollmann

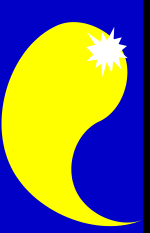


History

- P0260R19, mailing 2025-07
  - discussed by LWG in Brno
  - Poll: We think the exposition-only concepts should be removed from the paper and `bounded_queue` should be specified directly?
  - F N A: 4 3 1
  - JW: that isn't consensus
  - JW: we should ask them how strongly they feel

  - discussed by SG1 in Brno
  - Poll: SG1 requires P0260 concurrent queue concepts to be public concepts (as they were in prior revisions of P0260) instead of exposition-only concepts because they are essential to interoperability with external concurrent queues.
  - SF F N A SA: 2 5 3 0 0

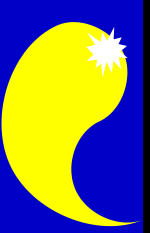
 Copyright ©1995-2026, Detlef Vollmann



P4295: Single Ends for Queues

- Concurrent queues are a communication mechanism
- In most systems this communication is uni-directional
  - one part of the syatem only pushes
  - another part only pops
- For better code structure it's useful to have interfaces that only provide one end of the queue
  - this was already in N3353
  - removed by SG1 to make it a separate class
  - P0260R12 explicitly proposes single-ended interface as wrapper as separate proposal.

 Copyright ©1995-2026, Detlef Vollmann



P4295: Single Ends for Queues

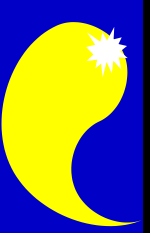
- P4295 uses the queue concepts at several places:

```
template <basic_concurrent_queue QType> class queue_front {
  // ...

  std::expected<typename queue_type::value_type, conqueue_status> try_pop()
    requires concurrent_queue<QType>;

  execution::sender auto async_pop()
    requires async_concurrent_queue<QType>;
}
```

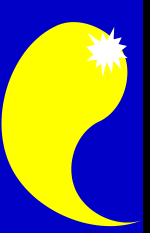
 Copyright ©1995-2026, Detlef Vollmann



Concepts Use in User Code

- Logging example from P0260 as template using single-ended wrapper
- Custom concepts for the ends
  - based on the standard concepts

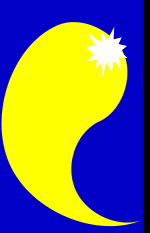
 Copyright ©1995-2026, Detlef Vollmann



Concepts Use in User Code

```
template <class QE>
concept MyConcurrentQueueFrontConcept =
  concurrent_queue<typename QE::queue_type> &&
  requires (QE q) {
    { q.try_pop() } -> same_as<expected<typename QE::queue_type::value_type,
                                        conqueue_status>>;
  };
template <class QE>
concept MyConcurrentQueueBackConcept =
  concurrent_queue<typename QE::queue_type> &&
  requires (QE q, typename QE::queue_type::value_type &&t) {
    { q.try_push(forward<typename QE::queue_type::value_type>(t)) }
      -> same_as<conqueue_status>;
  };
template <MyConcurrentQueueBackConcept QE>
void enqueueLogMessage(QE &qTail, std::string &&msg);
template <MyConcurrentQueueFrontConcept QE>
void outputLogMessage(QE &qHead);
```

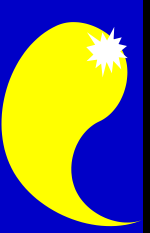
 Copyright ©1995-2026, Detlef Vollmann



Concepts Use in Tests

- There are lots of queue implementations
- Many tests are the same
- At some places modifications need to be done due to specific concept

 Copyright ©1995-2026, Detlef Vollmann



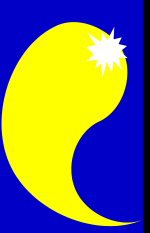
Concepts Use in Tests

```
template <template <typename> class QT, typename T>
requires stdqueue::basic_concurrent_queue<QT<T>>
const lest::test generalTests[] = //...

  if constexpr (concurrent_queue<QT<T>>) {
    auto res = q.try_pop();
    EXPECT(bool(res));
  } else {
    auto val = q.pop();
    EXPECT(bool(val));
  }

  if constexpr (async_concurrent_queue<QT<T>>) {
    auto aRes = ex::sync_wait(q.async_pop());
    EXPECT(bool(aRes));
  } else {
    auto val = q.pop();
    EXPECT(bool(val));
  }
```

 Copyright ©1995-2026, Detlef Vollmann



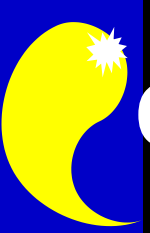
Concepts Use in Tests

```
typedef QTypeList<
  bounded_queue,
  simple_bounded_queue,
  concurrent_bounded_queue,
  async_bounded_queue,
  TestPrioQueue,
  TestStaticQueue> QTypes;

std::tuple<int, RealMoveInt, MoveOnlyInt> vTypes;

return runAllTests(QTypes(), vTypes, argc, argv);
```

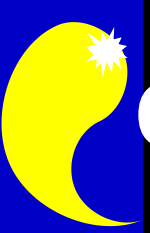
 Copyright ©1995-2026, Detlef Vollmann



Concepts Use in Queue Implementation

- `bounded_queue` implements `concurrent_queue` and `async_concurrent_queue`
- This cause some problems
  - `busy_async` in current version
  - Ill-formed with some schedulers after P3669
- Sometimes `concurrent_queue` *and* `async_concurrent_queue` are not needed
- `basic_bounded_queue` can solve this
- At some places the implementation now needs to check for the specific concept

 Copyright ©1995-2026, Detlef Vollmann



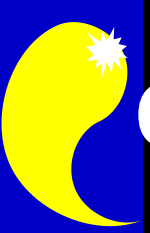
Concepts Use in Queue Implementation

```
enum class concurrent_queue_type {
  basic_queue = 0,
  concurrent_queue = 1,
  async_queue = 2
};

template <typename T,
          concurrent_queue_type QType,
          class Allocator = std::allocator<T>>
class basic_bounded_queue;

template <typename T, class Allocator = std::allocator<T>>
using bounded_queue =
  basic_bounded_queue<T,
                      concurrent_queue_type::concurrent_queue
                      | concurrent_queue_type::async_queue,
                      Allocator>;
```

 Copyright ©1995-2026, Detlef Vollmann

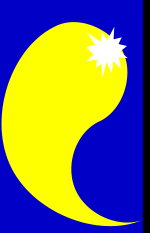


Concepts Use in Queue Implementation

- In `basic_bounded_queue::Popper::~Popper()`:

```
if constexpr (async_concurrent_queue<basic_bounded_queue>) {
  if (asyncPusher) {
    q->doPush(false, std::move(asyncPusher->val));
    asyncPusher->scheduleContinuation(true);
  }
}
```

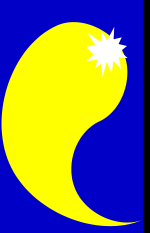
 Copyright ©1995-2026, Detlef Vollmann



Proposal

- Confirm SG1 decision to make concepts non-exposition-only again
- Modify `concept basic_concurrent_queue` to require explicit opt-in via tag type
- Put `bounded_queue` into separate header `<bounded_queue>`
- Not proposed (yet): `basic_bounded_queue`

 Copyright ©1995-2026, Detlef Vollmann



Questions

- ??

Code: branch base-bounded-queue in
https://gitlab.com/cppzs/bounded-queue/
or directly:
https://gitlab.com/cppzs/bounded-queue/-/tree/base-bounded-queue

Detlef Vollmann

dv@vollmann.ch
http://www.vollmann.ch/

 Copyright ©1995-2026, Detlef Vollmann