# 23/09/15 CppCon

rolling queues/ring: LEWG. Sg14: D0059R0,https://github.com/WG21-SG14/SG14/blob/master/Docs/Proposals/RingProposal.pdf

**GD:** std::queue causes fragmentation problems. Fixed-size <= 1 allocation. Why 'ring'?

`circular_buffer` already in boost, quite different. `rolling_queue` looks strange. `cyclic_buffer` and `ring_buffer` have `_buffer` and this is an adaptor. Looks like std::queue.

Exceptions: push and emplace may fail because ring is full. You might not care and be happy to overwrite older entries.

Open suggestion: aggressive_push and aggressive_emplace.

vector allows decide ring size at run-time and array at compile-time.

e.g. with vector, items only created when actually used in ring.

std::array -> static_ring and std::vector -> dynamic_ring.

In many places already use ring. When first suggested in discussion group, 6 implementations were submitted by other members.

On to the paper. Synopsis not complete for dynamic_ring.

**Scott:** This is a wrapper right? Should array be a parameter? Why different classes? Why not array and vector be type parameter of a single class template?

**GD:** One reason instantiation an issue. Reason two: not knowing size of vector

**LC:** Another proposal submitted. Far more concurrency-related version. Paper: n3353

**MW:** Has to deal with concurrent erasure which is tough problem. Advises GD to look through other paper.

**LC:** Doesn’t expect them to be unified.

**GD:** Purpose of this proposal is to avoid allocations.

**LC:** But there might be cross-fertilization and they should be distinct.

**Q:** Why not use perfect forwarding to pass array/vector to a unified class template c’tor? And what kind of iterators are they?

**GD:** FIFO iterators. No plan to run iterator over queue. (Ones in synopsis are boilerplate / internal.) Aims to make interface the same as std::queue.

**BR:** std::array can be expensive to move. So maybe not such a great idea.

**Scott:** Is this connected to discussion of fixed-sized containers?

**GD:** Both of these meant to be fixed sized. The names, dynamic_ and static_ are no concrete design decision currently.

**Q:** Buffer overrun: map has try_emplace(?) perhaps try_emplace here

**GD:** <not recorded>

**Bob:** fixed_ring instead of static_ring.

**GD:** Yes.

**Bob:** Lifetimes using c’tor / d’tor (?)

**Q:** Way to construct dynamic_ring from fixed_ring? E.g. when run out of space with static_ring?

**GD:** Not implemented. Good idea.

**Q:** Recommend don’t bikeshed names. The library working group will do all of this. They know this stuff! You can actually leave things unnamed and they will just take care of it.

**Authur:** Retracting uninitialized storage suggestion. How do we do this with vector and array. How does construction happen.

**GD:** static_ size starts ready-constructed and dynamic_‘s elements get

constructed as they are pushed - just like the two rings' underlying storage types.

**Authur:** How to tell empty / capacity?

**DG:** Can look at iterators. No capacity.

**Authur:** so for static_ is emplace a misnomer because array elements are around all the time.

**GD:** Popping doesn't destroy. Was tried. Was a mess. The details of conversation with Jon Wakely will be added to paper.

**SM:** Ranged-based creation or assignment iterating through all the objects would make it easier to convert from static_ to dynamic_.

**Charles:** More concerns. Uninitialized buffer instead of buffer so c'tor / d'tor called - same as for dynamic_ suggested. By extension maybe this isn't an adaptor.

**Q:** If you have a container that manages this, you can still keep it as an adaptor.

**GD:** Maybe deal with this with template parameter on static_ring.

**Ville:** Non-assignable elements are the reason for need for emplace in interface.

**Q:** If I just have allocated memory, I feel I should be able to use that with this adaptor. Could you use array_view and make that the adaptor? Would keep ability to allocate everything at once.

**Ville:** Moving the container into the ring is another option. Then no double-allocation.

**GD:** Both have container move constructor.

**Ville:** So already possible to avoid double-allocator.

**Q:** Reiterates idea of passing a block of memory.

**Scott:** begin and end might be tricky? The situation when begin and

end are the same.

**GD:** ring is FIFO. No intention to iterate through the buffer.

**Q:** Could ring be more like a ring?

**GD:** Not thought about it.

**MW:** Suggests Arthur work with GD on this.

**Ville:** non owning view is very different trade-off. Passing by values then poses different set of problems. And when you return them by value - very different set of problem.

**Q:** Could static_ring be composed of container and view?

**MW:** Now potentially looking at new proposal.

**GD:** Happy to develop this paper in response to feedback. Rather not turn this into a swiss army knife that gets turned down.

**JM:** Suggests static_ring and dynamic_ring don’t deal with object lifetime within interface.

MW & GD discuss how to proceed given many suggestions.

**MW:** Revise before Friday but vote in the mean time about whether we ‘think’ we’ll like.

**Ville:** Yes, can cast general ‘up’/‘down’ vote that suggests whether if warrants continued development - as opposed to voting on passing it right now.

**MW:** We could even work on it today and vote late today.

‘Up’ / ‘Down’ vote shows support by show of hands. Guy to continue to work on it until Friday

---

# 19/10/15 Kona

Circular-buffer queues

https://issues.isocpp.org/show_bug.cgi?id=129

Latest paper

https://issues.isocpp.org/attachment.cgi?id=10

Previous discussion

Author

Presenter

Guy Davidson

Latest update

http://wiki.edg.com/twiki/pub/Wg21kona2015/SG14/D0059R2.pdf

Pre-meeting comments

Discussing: https://issues.isocpp.org/attachment.cgi?id=10

Meeting discussion

Small group discussion (Thursday AM)

Attendees

Michael Wong (MW)

Patrice Roy (PR)

Billy Baker (BB) [Scribe]

Arthur O'Dwyer (AO)

Bill Seymour (BS)

Howard Hinnant (HH)

Juan Alday (JA)

Alan Talbot (AT)

Presentation and discussion

Review of D0059R2

MW presents

**PR:** the topic is something that is written quite a bit

**AO:** there are additional ongoing explorations in this same area

**MW:** the circular_buffer in boost is a bit heavier than what is proposed

**MW:** ring interface tries to match std::queue while having a fixed sized

**MW:** author would like feedback on whether two classes should be created

**AT:** why not just tweak queue

**AT:** similar fixed/dynamic issues with a string implementation, used template parameters

**PR:** the queue invariants don’t really apply here

**AO:** the ongoing work is available in a gethub repository

**AO:** D0059 owns memory (allocator aware) and not iterable

**PR:** with the allocator, can’t use external storage

**AO:** the ability to use external storage would be possible for both the fixed and dynamic cases

**AO:** other work would be a complete rework of this paper

**AO:** there is room for both D0059 as well as other work

**AO:** example from Sony would not work with the D0059 interface

**HH:** a synopsis is provided but not a specification, would like the specification before making a decision

back to author

Full group (Thursday PM)

Attendees

Jeffrey Yasskin (JY)

Bill Seymour (BS)

Patrice Roy (PR)

Axel Naumann (AN)

Lars Bjonnes (LB)

Pete Becker (PB)

Howard Hinnant (HH)

Juan Alday (JA)

Billy Baker (BB)

Bryce Lollander (BL)

Mike Spencer (MS)

Titus Winters (TW)

Alan Talbot (AT)

Thomas Koeppe (TK)

Arthur O'Dwyer (AO)

Nico Josuttis (NJ)

Jonathan Wakely (JW)

Presentation and discussion

Discussion of the design.

The buffer is a queue and never constructs/destroys, only assigns.

It has fixed capacity and bounded size.

**AN:** I don't like that pop() doesn't destroy. Rename the operation, or require that the element type be trivially destructible.

**TK:** Why isn't this a deque?

**JW:** Because you can't destroy from the middle of a vector.

**TK:** Then don't use a vector underneath.

**JY:** For the use case of trivially destructible types, many of the design decisions become trivial.

**HH:** 1) the “pop” is more like “remove” than “erase”, in that it doesn’t destroy anything. 2) Please send a specification rather than just a synopsis. It feels like we’re just guessing the semantics.

**TW:** Without destructors, I can’t put refcounted things in.

**HH:** I feel that this should not be in the standard. It’s fine for them to write code like that, but it doesn’t fit for the standard. I might feel different if it were marketed differently. For example, a “circular range” object would be interesting.

Polls:

circular range/span (non-owning, points into an array of existing objects): 9

owning buffer of once-constructed objects, assignment-only operations: 1

buffer that constructs and destroys elements in-place as it goes along: 9

compile-time sized: 12

dynamic, fixed-size: 13

dynamic, resizable: 5

Should it support iteration?

```
SF F N A SA
```

```
2 6 5 3 0
```

---

# 14/3/16 GDC

Adapter type

*Questions about iterators

*Questions about concurrency

*Questions about owning vs. non-owning

---

# 11/16 Issaquah

Michael Wong presenting on behalf of Guy Davidson

Ring span is a circular buffer as SG14 like contiguous structures

Changed from Ring buffer to span to reflect normal langauge as it is a view across non owned memory

Referred back from LEWG in Oulu

**Nat:** why is a requirement for single producer/ single consumer

**Hans:** what is in mind is the avoidance of contention because producer and consumer are at opposite ends

**Michael:** multi prod/cons adds overhead

**Will:** notes that concurrent ring buffers are in wide use between hardware and software

**Detlef:** How do I wait on space?

Will and David discussed memory order constraints

**Will:** this is acquire release in most operations

**Lawrence:** Nothing else really makes sense

**David:** we cannot allow default SC if this is really performance based

**Will and Lawrence:** Comment that this is a sequential case with bolt ons and that is not working well

**David:** does Olivier and Geoff discuss the intention of ContiguousIterator

**Geoff:** The point of ContiguousIterator is to allow type erasure

**Nat:** respond to Will's observation. Notes that the author state that this is not a concurrent ring but rather is adapted from a concurrent queue

Michael seeks view from finance

Thomas confirms that this is a use case in HFT

Olivier concerned about single/single versus multi/multi which apparently involves a lot of razor blades and a close shave!

**Detlef:** I would like to see a more compelling use case that this is different to Lawrence's queue

**Lawrence:** Has a sequential interface that requires a test pattern to enable access. Needs and extension to the interface

**Geoff:** Why is there an iterator in this as it does not otherwise meet requirements?

**Lawrence:** This is for debugging purposes as it allows you to work our why it crashed. But in concurrent use this has issues

**Neil:** I don't think that that can be the motivation, it does not feel appropriate to SG14 type use cases if Lawrences motivation is correct

**Nat:** We may need

**Thomas:** This is a view type and thus has an underlying container.

**Paul:** Can we use a trait to support the iterator.

**Hans:** We need to performance justification.

**Lawrence:** history buffer and communication buffer are different things

**Nat:** Are there concurrent ring span use cases that cannot be addressed by the concurrent queue

**Michael:** get the authors to work with Lawrence

**Nat:** Use of an adaptor over another container, is mostly an avoidance of memory allocation, there is at least one impl of Lawrence's queue that can do this. Can Lawrence's queue be adapted to work over an underlying memory?

**Detlef:** Push_back is different in serial versus concurrent, this is not acceptable in a single data structure.

**Hans:** We really need to understand why the two different use cases are represented in a single impl, as this complicates matters

**Nat:** Can we focus on the two cases of single/single multi/multi, not worry about the single/multi variants. Should direct the authors to come up with a use case for concurrent communication that is not captured in Lawrence's queue.

**Maged:** Notes that the single/multi variants can have special performance optimizations

**Olivier:** To counter Nat, we need a non-concurrent sequential and a multi/multi/concurrent. Thus two interfaces, sequential and concurrent are the two that we need. sequential is the single/single while multi/multi should be defined in terms of concurrent queue. Divorce the two interfaces as they are no compatible with one another. Our job is to provide the correct vocabulary for interfaces. Implementations are free to innovate but the user gets standardised API. Advice to split this in to two papers. Thread unsafe goes straight to LEWG. SG1 is concerned with the concurrent and there are severe reservations around iterators In order to allow the association between the underlying and the span, we need to be able to access the head and tail positions directly.

**Hans:** Java APIs would typically snapshot, is this something we can

consider.

**Michael summary:**

If concurrent interface is to remain we need a strong justification for divergence from Lawrence's interface.

A performance argument would probably need to be more precise about memory ordering.

Divorce the `concurrent_ring_span` and `ring_span` (`ring_span` can go to LEWG).

`concurrent_ring_span` deferred to `concurrent_queue`

---

# 03/17 Kona

From the LEWG Wiki :

Meeting discussion

Please invite the following people when the paper is discussed:

Arthur O'Dwyer

Michael Wong

Walter E Brown

Day AM/PM (In case there are multiple discussions)

Discussing: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/p0059r3.pdf

Attendees (If the notetaker can gather this)

Bryce Adlestein-Lelbach

Hal Finkel

Patrice Roy

Tim Shen

Jeffrey Yasskin

Mark Zeren

Presentation and discussion

**Hal:** like standardizing something called *_popper

**Patrice:** I think they did the job that was asked of them (fixed size, no iterators, policy-based poppers)

**Jeffrey:** I'm glad the name ring's gone. Too much mathematics-related meaning

**Jeffrey:** why not circular_buffer?

**Bryce:** if we keep ring_span, I'd like the relation to span to be clarified

**Jeffrey:** it's because it does not own memory

**Bryce:** users might be surprised

**Mark:** in a sense, it does not let you get chunks from it

**Patrice:** *_buffer would be weird too, as it's non-owning

**Jeffrey:** *_buffer in networking is non-owning

**Jeffrey:** I think a signed size_type would be useful here, like in the case of span, as it's less error-prone

**Patrice:** so ring_adapter would be better? (suggested by Hal)

**Hal:** we want to separate this from allocator issues

**Bryce:** it would be cool to have a default for users that don't want to create a vector and adapt it. Not something to block the proposal, just something to think about

**Jeffrey:** it was pretty much thought for POD types at first

**Patrice:** it was made by / for games people

**Patrice:** they allow moveable T iff T is nothrow-move-assignable

**Bryce:** can it be constructed from an empty range? If so, they have a precondition on front() / back() and cannot be noexcept

**Hal:** how does it signal error?

**Patrice:** they assert. Technically, I think they want UB

**Bryce:** they might get noexcept on front() / back(), but they'll need faith

**Hal:** it would require constraining the policy class

**Hal:** this class should be so simple that it will all be inlined anyway

**Mark:** we'll have to use «throws : nothing» instead of noexcept

**Tim:** I see begin() and end() in the member function pseudocode even though there are no iterators

**To summarize:**

good in theory, wording needs work

span might not be the right name (it doesn't really match the other *_span classes)

explore a companion container for the future

it's sometimes needed to build a ring on uninitialized memory buffers. The current design does not seem to support that

this one's fine; another that manages raw memory and destroy objects to is needed

spec needed («this is implemented as…» is not a spec). Please look at std::queue. Bryce offers to help

the rationale for «no iterator» needs work

the rationale for having a separate buffer too

clarify the rationale for the *_popper policies, particularly the non-default ones including alternatives

noexcept should be «throws : nothing» as in the case of an empty range, operations have preconditions

in general, the behavior of the class on an empty range needs to be described

remove «synchronous» from push_back() and others (editorial)

in the design decisions, we seem to be removing objects and releasing memory. It’s probably not true

Full LEWG Discussion:

**Patrice:** (summarizing) Worries about name. Maybe ring_adaptor? Had debates about whether being non-owning is the right thing there. But it has been back and forth in previous meetings. Might be useful to have an adaptor class to make its usage simpler. Clear use case might be managing uninitialized memory. it might be a useful improvement to add that. Specs need some work, so does rationale. So overall a positive view but a number of things need some work.

**Jeffrey:** Any questions for the full group?

**Patrice:** No. Send it back with encouragement and things to work on.