# General notes

1. Since switching to pointer-based data it seems that all unattended race conditions become “uninitialized memory access” errors. This effectively means that we cannot find “dequeue before enqueue” linearizability issues only “out of order dequeue” issues.
2. As a result, this makes “--disable-race-detection” ineffective. It also became somewhat impossible to test “relaxed” versions for linearizability issues.
3. HW is still the only queue with a discoverable linearizability issue.
4. Tests were run with a timeout of one hour per test.
5. When limiting potential loops, limiting to a single iteration was usually the only way to get a queue to run through our entire test suite. In some cases (MS, LCRQ), it is not a valid option since these queues have conditional statements that break the loop. These cases were not resolved so some tests timed out.
6. LCRQ – no good solution was found to working with pointers in this case.

# Queues

## Chase-Lev

<https://github.com/ori-saporta83/linearizability-testing/blob/master/wrappers/chase-lev-wrapper.h>

Requires 2 atomic integer fields (top/bottom) as well as the data buffer to be atomic and read/written using release/acquire. This is caused by a race between two enqueue operations trying to write to the same location since enqueue does not change any state before writing.

Results log:



## Fast-MPMC

<https://github.com/ori-saporta83/linearizability-testing/blob/master/wrappers/fastmpmc-wrapper.h>

Requires only 3 atomic unsigned integer fields. Moreover, all but a single release/acquire operation on the *version* field can be relaxed.

Results log:



## HW

<https://github.com/ori-saporta83/linearizability-testing/blob/master/wrappers/hwqueue-wrapper.h>

Requires both *back* and the *AR* fields to be atomic. Access to the former must be done using release/acquire operations or acq\_rel/acquire to avoid the linearizability issue. Access to the buffer field can be relaxed.

Results log:

## LCRQ

https://github.com/ori-saporta83/linearizability-testing/blob/master/wrappers/lcrq.c

1. Requires a single CAS operation to work on two fields. C11 atomics only support CAS operations on pointer size words so could not find a way to mesh two fields into a single pointer size variable. As a result, could not test with pointer-based data.
2. Contains complex loops that could not be limited to a single iteration without blocking valid paths. As a result, essentially all tests timed out.

Results log: N/A

## MPMC

<https://github.com/ori-saporta83/linearizability-testing/blob/master/wrappers/mpmc-queue-wrapper.h>

Requires 3 atomic fields (*rdwr, read, written*), the first of which can be handled with relaxed operations. The other two with release/acquire.

It probably cannot be considered wait/lock free since it essentially has a spin lock in both enqueue and dequeue operations to sync between multiple threads performing the same operation:

\_\_VERIFIER\_assume((atomic\_load\_explicit(&q->m\_written, memory\_order\_acquire) & 0xFFFF) == wr);

\_\_VERIFIER\_assume((atomic\_load\_explicit(&q->m\_read, memory\_order\_acquire) & 0xFFFF) == rd);

In these cases, if other threads perform this operation terminate without completing it, threads in these spin locks will hang.

Results log:



## MS

<https://github.com/ori-saporta83/linearizability-testing/blob/master/wrappers/ms-queue-wrapper.h>

Requires 2 global atomics and another one per queue entry. The data buffer itself is not atomic. Most accesses are release/acquire. Some can be relaxed. This queue is another example for a case where a conditional behavior in the loop requires us to allow for at least two iteration which causes some tests to time out.

Results log:



## QU

<https://github.com/ori-saporta83/linearizability-testing/blob/master/wrappers/qu-wrapper.h>

Similarly to previous structures, this queue requires 2 global atomics (head, tail) and another one (next) per node. It is implemented as a linked list so there is no data buffer. Almost all atomic operations can be relaxed. A single pair of operations on the *next* field is required to used release/acquire.

Results log:



## UNIQ

<https://github.com/ori-saporta83/linearizability-testing/blob/master/wrappers/uniq-wrapper.h>

https://github.com/bittnkr/uniq/blob/d4a5b9c413aef2a242a7f9654ded198fba78b9bf/cpp/uniq.h

Requires 2 atomic integer fields (in/out) and an atomic integer array (is\_used). All atomic operations can be relaxed with the addition of a single release/acquire fence pair.

Results log:

