The Three Fences

This lesson gives an overview of the acquire, release, and full fences used in C++ as memory barriers.

WE'LL COVER THE FOLLOWING

Typically, three kinds of fences are used: full fence, acquire fence and release fence. As a reminder, acquire is a load, and release is a store operation. What happens if I place one of the three memory barriers between the four combinations of load and store operations?

- **Full fence**: A full fence `std::atomic_thread_fence()` between two arbitrary operations prevents the reordering of these operations, but guarantees that it won't hold for StoreLoad operations. Also, they can be reordered.
- **Acquire fence**: An acquire fence `std::atomic_thread_fence(std::memory_order_acquire)` prevents a read operation before an acquire fence from being reordered with a read or write operation after the acquire fence.
- **Release fence**: A release fence `std::atomic_thread_fence(std::memory_order_release)` prevents a read or write operation before a release fence from being reordered with a write operation after a release fence.

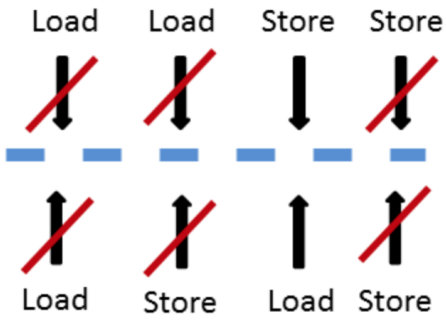
A lot of energy goes into accurately forming the definitions of the acquire and release fence and their consequences for lock-free programming. The subtle

differences between the acquire-release semantic of atomic operations are

especially challenging to understand. Before I get to that point, I will illustrate the definitions with graphics.

Which kind of operations can cross a memory barrier? Have a look at the following three graphics. If the arrow is crossed with a red bar, the fence prevents this type of operation.

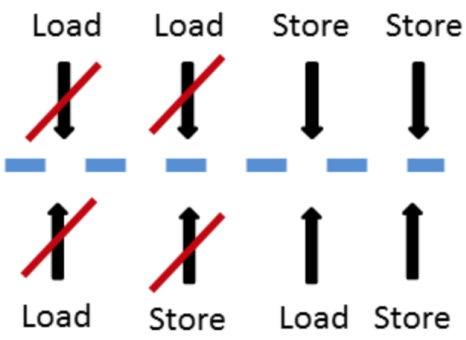
Full fence




```
std::atomic_thread_fence()
std::atomic_thread_fence(std::memory_order_acq_rel)
```

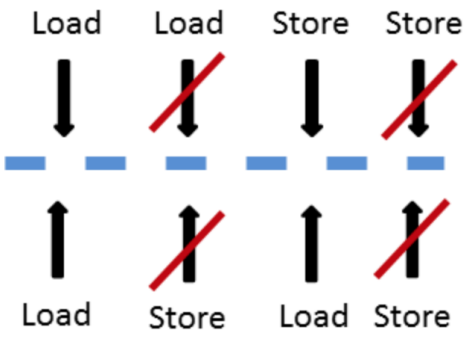
Of course, instead of writing `std::atomic_thread_fence()` you can explicitly write `std::atomic_thread_fence(std::memory_order_seq_cst)`. Sequential consistency is applied to fences by default. If you use sequential consistency for a full fence, the `std::atomic_thread_fence` follows a global order.

Acquire fence



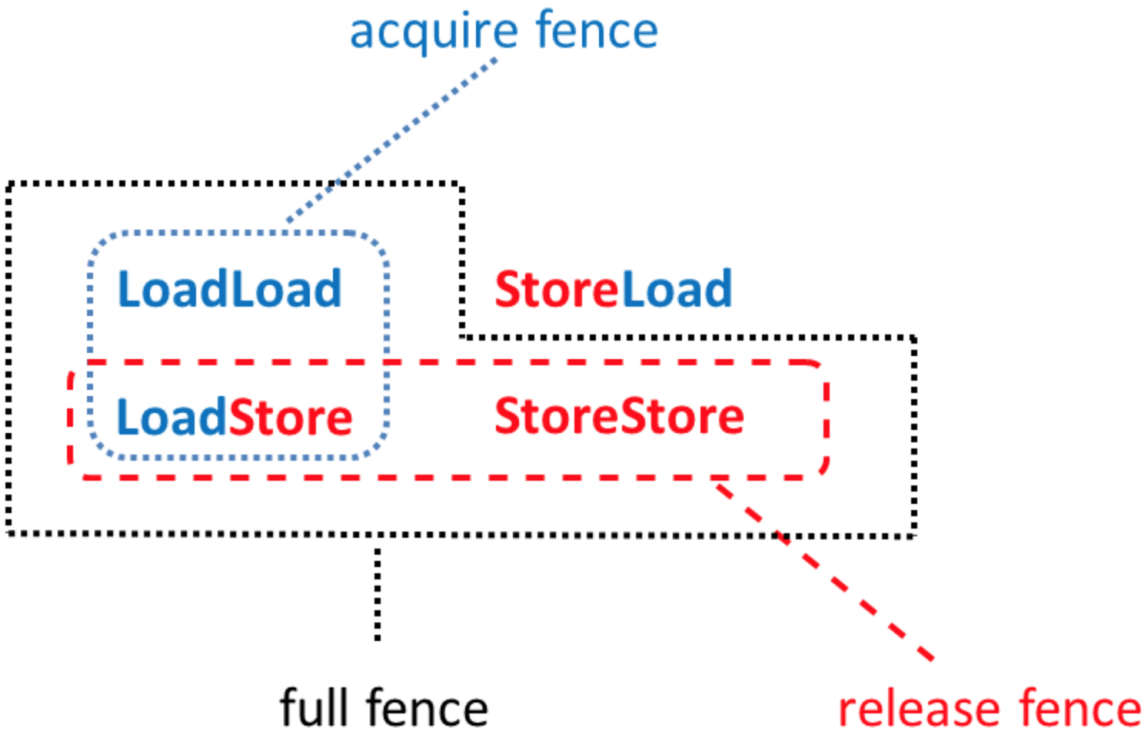

```
std::atomic_thread_fence(std::memory_order_acquire)
```

Release fence




```
std::atomic_thread_fence(std::memory_order_release)
```

The three memory barriers will be depicted even more concisely in the diagram below:



Acquire and release fences guarantee similar synchronization and ordering constraints that are similar to atomics with acquire-release semantic.