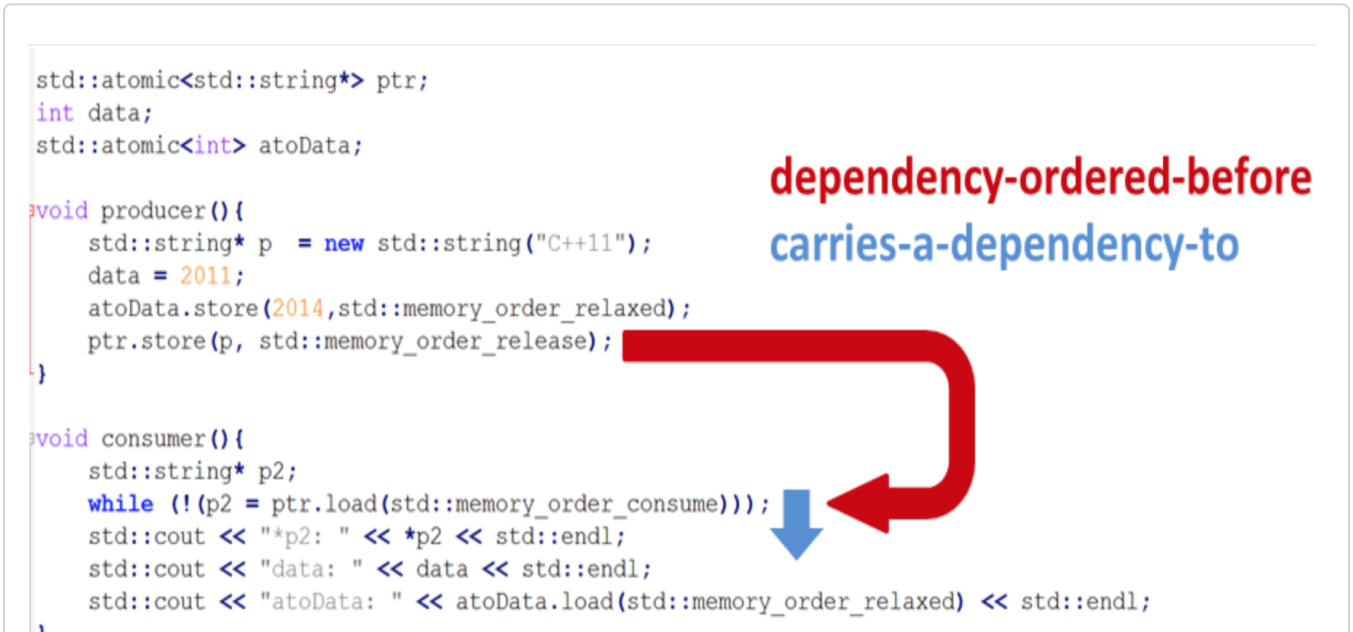
Data dependencies with std::memory_order_consume

This lesson explains data dependencies with std::mem_order_consume in C++.

`std::memory_order_consume` deals with data dependencies on atomics; these data dependencies exist in two ways. First, let us look at *carries-a-dependency-to* in a thread and *dependency-ordered before* between two threads. Both dependencies introduce a *happens-before* relation. These are the kind of relations we are looking for. What does *carries-a-dependency-to* and *dependency-order-before* mean?

- **carries-a-dependency-to**: If the result of operation A is used as an operand in operation B, then A *carries-a-dependency-to* B.
- **dependency-ordered-before**: A store operation (with *std::memory_order_release*, *std::memory_order_acq_rel* or *std::memory_order_seq_cst*) is *dependency-ordered-before* load operation B (with *std::memory_order_consume*) if the result of load operation B is used in a further operation C in the same thread. It is important to note that operations B and C have to be in the same thread.

I know from personal experience that both definitions might not be easy to digest. Here is a graphic to visualize them.



The expression `ptr.store(p, std::memory_order_release)` is *dependency-ordered-before* the expression `while (!(p2 = ptr.load(std::memory_order_consume)))`, because the following line `std::cout << "*p2: " << *p2 << std::endl` will be read as the result of the load operation. Furthermore it holds that `while (!(p2 = ptr.load(std::memory_order_consume))` *carries-a-dependency-to* `std::cout << "*p2: " << *p2 << std::endl`, because the output of `*p2` uses the result of the `ptr.load` operation.

We have no guarantee regarding the output of `data` and `atoData`. That's because neither has a *carries-a-dependency* relation to the `ptr.load` operation. That being said, it gets even worse. Since `data` is a non-atomic variable, there is a race condition on the variable `data`; this is because both threads can access data at the same time, and thread t1 wants to modify `data`. Therefore, the program has undefined behavior.

Finally, we'll cover our relaxed semantic topic in the next lesson!