Strong Memory Model

This lesson gives a brief overview of the strong memory model regarding concurrency in C++.

WE'LL COVER THE FOLLOWING

Atomics are the base of the C++ memory model. By default, the strong version of the memory model is applied to the atomics; therefore, it makes a lot of sense to understand the features of the strong memory model. As you may already know from the subsection on Contract: The Challenges, with the strong memory model I refer to sequential consistency, and with the weak memory model I refer to relaxed semantic.

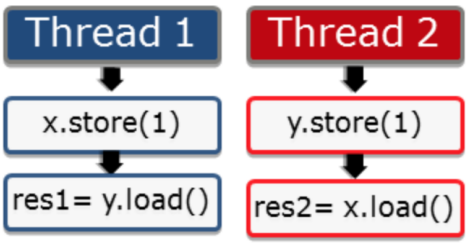
Strong Memory Model

Java 5.0 got its current memory model in 2004, and C++ got its model in 2011. Before that, Java had an erroneous memory model and C++ had no memory model. Those who think this is the endpoint of a long process are completely wrong. The foundations of multithreaded programming are 40 to 50 years old; Leslie Lamport defined the concept of sequential consistency in 1979.

Sequential consistency provides two guarantees:

- The instructions of a program are executed in source code order.
- There is a global order of all operations on all threads.

Before I dive deeper into these two guarantees, I want to explicitly emphasize that these statements only hold for atomics, but still influence non-atomics. This graphic shows two threads: each thread stores it's variable *x* or *y* respectively, loads the other variable *y* or *x*, and stores them in the variable *res1* or *res2*.

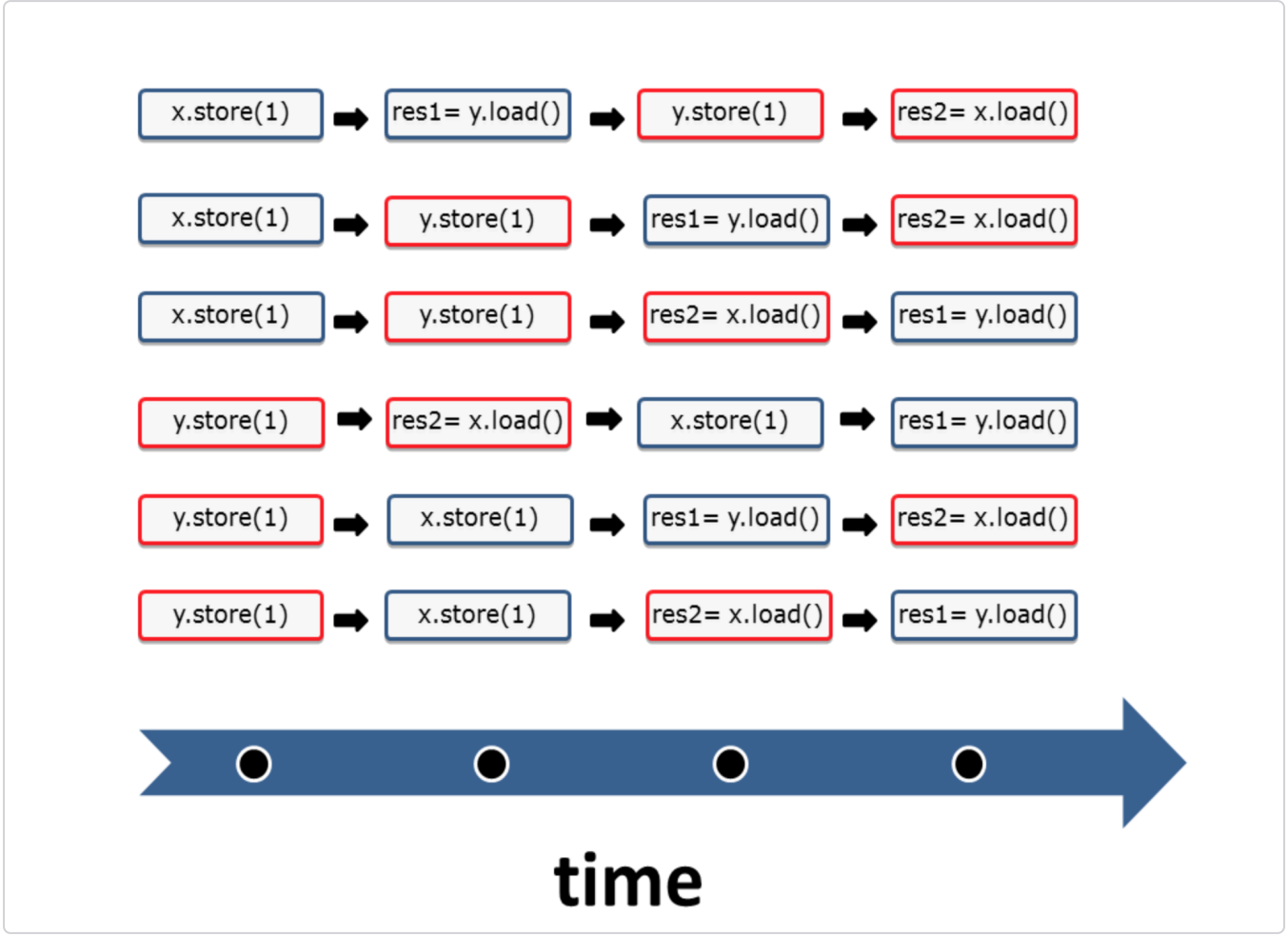


Because the variables are atomic, the operations are executed atomically; by default, sequential consistency applies. The question is, in which order can the statements be executed?

The first guarantee of the sequential consistency is that the instructions will be executed in the order defined in the source code. This is easy; no store operation can overtake a load operation.

The second guarantee of the sequential consistency is that all instructions of all threads have to follow a global order. In the case listed above, it means that thread 2 sees the operations of thread 1 in the same order in which thread 1 executes them. This is the key observation: thread 2 sees all operations of thread 1 in the source code order of thread 1. The same holds from the perspective of thread 1. You can think about characteristic number 2 as a global clock which all threads have to obey. The global clock is the global order. Each time the clock makes a tick, one atomic operation takes place, but you never know which one.

We are not yet done with our riddle! We still need to look at the different interleaving executions of the two threads. So, the following six interleavings of the two threads are possible.



That was easy, right? That was sequential consistency, also known as the Strong Memory Model. Now, we will cover the Weak Memory Model in the next lesson.