

## A well defined memory model

1. What are atomic operations?
2. Which order of operations is ensured?
3. When are memory effects of operations visible?

**To 1:**Atomic operations are operations that follow the first three letters of the famous ACID Idioms from the database theory. Atomic operations are atomic (A), going from one consistent (C) state to the next and are executed in isolation (I). This means in particular, no other thread can observe an intermediate state of an atomic operation. The incrementation atomVar++ shows the consistency and isolation of an atomic operation very nice. If atomVar is an atomic variable, atomVar can have only the old or the new value. The consistency of the variable atomVar is, that it changes only from one state to the other and the isolation, that another thread can not observe any intermediate value.

**To 2:** Both the compiler that translate the program into assembler instructions, and the processor that executes the assembler instructions, can rearrange the operations. Most often this is for performance reasons. In addtion the various tiers of storage (cache) posse the possibility to provide the result of the operations in a delayed way.

**To 3:** Since it is quite possible that one thread sees an operation on a variable later than another, the threads have to obey certain rules.

## The standardized threading interface

The standardized threading interface in C++11 is composed of the following components.

1. Threads
2. Tasks
3. Thread local data
4. Condition variables

**To 1:** Threads are the basic building blocks of multithreaded programming. They do their work autonomously, are parameterized by arguments and interact with other threads via shared variables.

**To 2**:Tasks are a relatively modern concept. Tasks consist of two components, which are connected by a communication channel. One component as endpoint of the channel produces the result, while the other endpoint consumes it. The producer is called Promise, the consumer Future.

**To 3**: Thread local data is data - such as it is easy to guess from the name- that explicitly belongs to one thread.

**To 4**:Condition variables enables it to implement producer/consumer workflows. The consumer waits for the notification of the producer so that he can continue his work.