Transactional Memory - An Overview

This lesson gives an outline of transactional memory, predicted to be introduced in C++20.

WE'LL COVER THE FOLLOWING

Transactional memory is based on the idea of a transaction from database theory. Transactional memory makes working with threads a lot easier for two reasons: first, data races and deadlocks disappear; second, transactions are composable.

A transaction is an action that has the following properties: **A**tomicity, **C**onsistency, **I**solation, and **D**urability (ACID). Except for the durability or storing the result of an action, all properties hold for transactional memory in C++. Now three short questions are left.

ACI(D)

What do atomicity, consistency, and isolation mean for an atomic block consisting of some statements?

```
atomic{
  statement1;
  statement2;
  statement3;
}
```

Atomicity: Either all or none of the statements in the block are performed.

Consistency: The system is always in a consistent state. All transactions establish a total order.

Isolation: Each transaction runs in total isolation from other transactions.

How do these properties apply? A transaction remembers its initial state and will be performed without synchronization. If a conflict happens during its execution, the transaction will be interrupted and restored to its initial state. This rollback causes the transaction to be executed again. If the initial state of the transaction is maintained at the end of the transaction, the transaction will be committed.

A transaction is a kind of speculative activity that is only committed if the initial state holds. In contrast to a mutex, it is an optimistic approach. A transaction is performed without synchronization; it will only be published if no conflict occurs. That being said, a mutex is a pessimistic approach. First, the mutex ensures that no other thread can enter the critical region. Next, the thread will enter the critical region if it is the exclusive owner of the mutex, hence all other threads are blocked.