3.1 Lecture Summary

3.1 Actor Model

Lecture Summary: In this lecture, we introduced the *Actor Model* as an even higher level of concurrency control than locks or isolated sections. One limitation of locks, and even isolated sections, is that, while many threads might correctly control the access to a shared object (e.g., by using object-based isolation) it only takes one thread that accesses the object directly to create subtle and hard-to-discover concurrency errors. The Actor model avoids this problem by forcing all accesses to an object to be isolated *by default*. The object is part of the *local state* of an actor, and cannot be accessed directly by any other actor.

An Actor consists of a *Mailbox*, a set of *Methods*, and *Local State*. The Actor model is *reactive*, in that actors can only execute methods in response to messages; these methods can read/write local state and/or send messages to other actors. Thus, the only way to modify an object in a pure actor model is to send messages to the actor that owns that object as part of its local state. In general, messages sent to actors from different actors can be arbitrarily reordered in the system. However, in many actor models, messages sent between the same pair of actors preserve the order in which they are sent

Optional Reading:

1. Wikipedia article on the Actor Model

2. Documentation on the Akka Actor Library (though Akka is not used in this course, it is a useful library to be aware of if you are interested in using the actor model with Java and Scala applications)

Mark as completed