Since the introduction of multi-core processors, programmers must explicitly use concurrency to make their programs faster. This is notoriously difficult. To this end, developers use concurrency models: techniques that introduce parallelism in a controlled manner and provide guarantees to prevent common errors, e.g. race conditions and deadlocks. An empirical study has shown that existing programs often combine multiple such models. We study these combinations and show that they can annihilate the guarantees of their constituent parts. Hence, the assumptions of developers are invalidated and errors can resurface.

In this dissertation, we start from three concurrency models: futures, transactions, and actors. First, for the combination of transactions and futures, we create transactional futures: futures created in a transaction with access to the encompassing transactional context. Second, the combination of transactions and actors leads to transactional actors. These make it possible both to create transactions in actors, and vice versa, to send messages to actors in transactions. Finally, we combine all three models into one unified framework: Chocola (for “Composable Concurrency Language”), implemented as an extension of Clojure, formalized using an operational semantics, and evaluated in three applications.

This dissertation thus comprehensively studies the combination of three radically different concurrency models – futures, transactions, and actors – and specifies a semantics for their combinations that aims to introduce parallelism while maintaining their guarantees.