

STM



Software Transactional Memory

Atomic effects

- have similar APIs/descriptions as ZIOs
- cannot be created from arbitrary Scala code
- cannot be "run"
- evaluate to regular ZIOs

```
val anAttemptSTM: ZSTM[Any, Throwable, Int] = STM.attempt(42 / 0)

// "commit", i.e. evaluate this effect
val anAtomicEffect: ZIO[Any, Throwable, Int] = anAttemptSTM.commit
```

Pros

- atomicity guaranteed
- familiar APIs
- tools for complex CS problems

Cons

- have to replicate common programming structures

STM Data Structures

Tools for general programming with transactional properties

- creation is an STM effect
- all API calls are STM effects

STM Ref: atomic reference (doubles as transactional "variable")

```
// get, update, modify, set  
val aVariable: USTM[TRef[Int]] = TRef.make(42)
```

Array: mutable storage of elements

```
// apply, update, transform, fold  
val specifiedValuesTArray: USTM[TArray[Int]] = TArray.make(1,2,3)
```

Set: mutable collection of unique elements

```
// contains, put, delete, transform, union/intersection/diff, removeIf/retainIf  
val specificValuesTSet: USTM[TSet[Int]] = TSet.make(1,2,3,4,5,1,2,3)
```

STM Data Structures

Map: mutable collection of associations with unique keys

```
// put, get, delete, keys/values, removeIf/retainIf, transform  
val aTMapEffect: USTM[STMap[String, Int]] = STMap.make(("Daniel", 123), ("Alice", 456))
```

Queue: FIFO-ordered mutable collection

```
// offer/offerAll, take/takeAll/takeOption, peek  
val tQueueBounded: USTM[TQueue[Int]] = TQueue.bounded[Int](5)
```

Priority Queue: sorted mutable collection

```
// same API as Queue  
val maxQueue: USTM[TPriorityQueue[Int]] = TPriorityQueue.make(3,4,1,2,5)
```

STM Coordination

TRef: atomic reference for guards against race conditions

```
// cannot be read/written by two fibers at the same time  
val aVariable: USTM[TRef[Int]] = TRef.make(42)
```

TPromise: notification mechanism

```
// same API as regular promise  
val tPromiseEffect: USTM[TPromise[String, Int]] = TPromise.make[String, Int]
```

TSemaphore: controlled access to a critical region

```
// acquire/acquireN, release/releaseN, withPermit  
val tSemaphoreEffect: USTM[TSemaphore] = TSemaphore.make(10)
```

TReentrantLock: combined primitive for readers-writers problem, with

- read lock which can be acquired multiple times
- write lock which can be acquired once

```
// acquireRead/acquireWrite, releaseRead/releaseWrite, readLocked/writeLocked  
val reentrantLockEffect = TReentrantLock.make
```

ZIO rocks

