

Fibers



Concurrency vs Parallelism

Parallelism = multiple computations running *at the same time*

Concurrency = multiple computations *overlap*

Parallel programs may not necessarily be concurrent

- e.g. the tasks are independent

Concurrent programs may not necessarily be parallel

- e.g. multi-tasking on the same CPU

We focus on concurrency

- poses the most problems
- is almost always a requirement for useful programs



Fibers

Fiber = description of an effect being executed on some other thread

```
def createFiber: Fiber[Throwable, String] = ???
```

failure type

result type

Creating a fiber is an *effectful* operation

- the fiber will be wrapped in a ZIO

```
val aFiber: ZIO[Any, Nothing, Fiber[Throwable, Int]] = meaningOfLife.fork
```

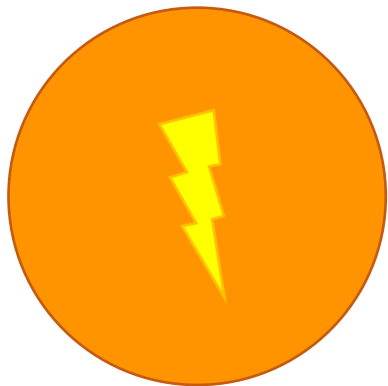
Managing a fiber is an *effectful* operation

- the result of the operation is wrapped in another ZIO

```
def runOnSomeOtherThread[R,E,A](zio: ZIO[R,E,A]): ZIO[R,E,A] = for {  
  fib <- io.fork  
  result <- fib.join  
} yield result
```

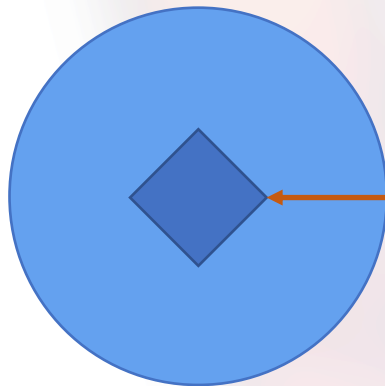
How Fibers work

ZIO has a thread pool that manages the execution of effects



thread

active = can run code



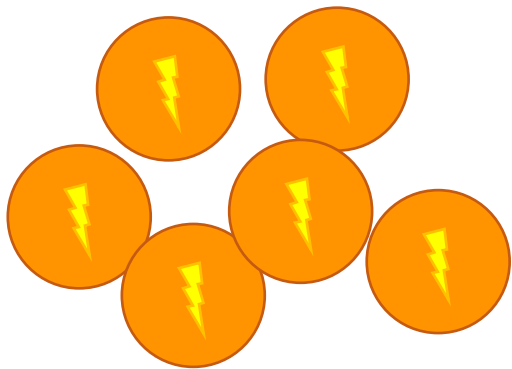
fiber

passive = just a data structure

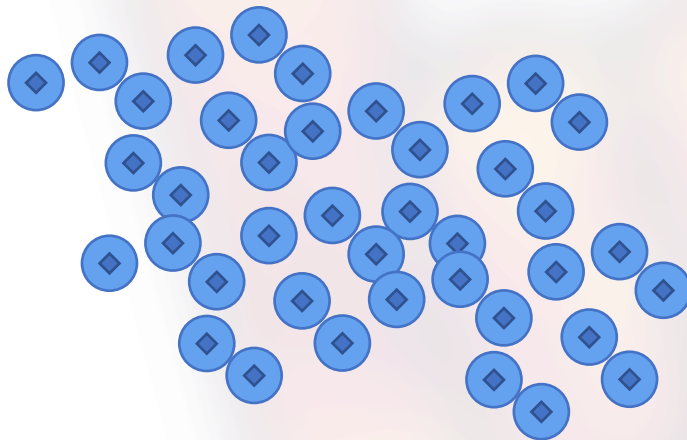
effect

How Fibers work

ZIO has a thread pool that manages the execution of effects



a few threads
(100s)



LOTS of fibers
(100000000s per GB heap)

ZIO schedules fibers for execution.

Motivation for Fibers

Why we need fibers

- no more need for threads and locks
- delegate thread management to ZIO runtime
- avoid asynchronous code with callbacks (callback hell)
- maintain pure functional programming
- keep low-level primitives (e.g. blocking, waiting, joining, interrupting, cancelling)

Fiber scheduling concepts & impl details

- blocking effects in a fiber lead to *descheduling*
- semantic blocking
- cooperative scheduling
- the same fiber can run on *multiple* JVM threads
- work-stealing thread pool

Fibers rock

