

# Promise



# Promise

A purely functional concurrency primitive with two methods

- `get`: blocks the fiber (semantically) until a value is present
- `complete`: inserts a value that can be read by the blocked fibers

```
val aPromise = Promise.make[Throwable, Int]
```

## Why

- allows inter-fiber communication
- avoids busy waiting
- maintains thread safety

## Use-cases

- producer-consumer-like problems
- sending data between fibers
- notification mechanisms

```
def consumer(signal: Promise[Throwable, Int]) = for {  
  _ <- ZIO.succeed("[consumer] waiting for result...").debug  
  meaningOfLife <- signal.get // blocks  
  _ <- ZIO.succeed(s"[consumer] got: $meaningOfLife").debug  
} yield ()  
  
def producer(signal: Promise[Throwable, Int]) = for {  
  _ <- ZIO.succeed("[producer] crunching numbers...").debug  
  _ <- ZIO.succeed.sleep(1.second)  
  _ <- ZIO.succeed("[producer] complete: 42").debug  
  meaningOfLife <- ZIO.succeed(42)  
  _ <- signal.complete(meaningOfLife) // unblocks consumer  
} yield ()
```

**ZIO rocks**

