Futures and Promises

Back to the Future[T]

Future[T] is a computation which will finish *at some point*

```scala
import ExecutionContext.Implicits.global

val recipesFuture: Future[List[Recipe]] = Future {
  // some code that takes a long time to run
  jamieOliverDb.getAll("chicken")
}
```

a default ExecutionContext already implemented

ec is passed implicitly*

non-blocking processing

```scala
future.onComplete { case Success(recipes) => ... }
```

map, *flatMap*, *filter*, for-comprehensions

falling back

```scala
future.recover { case NotFoundException => ... }
```

blocking if need be

```scala
val txStatus = Await.result(transaction, 1 seconds)
```

Making Promises

Futures are immutable, "read-only" objects.

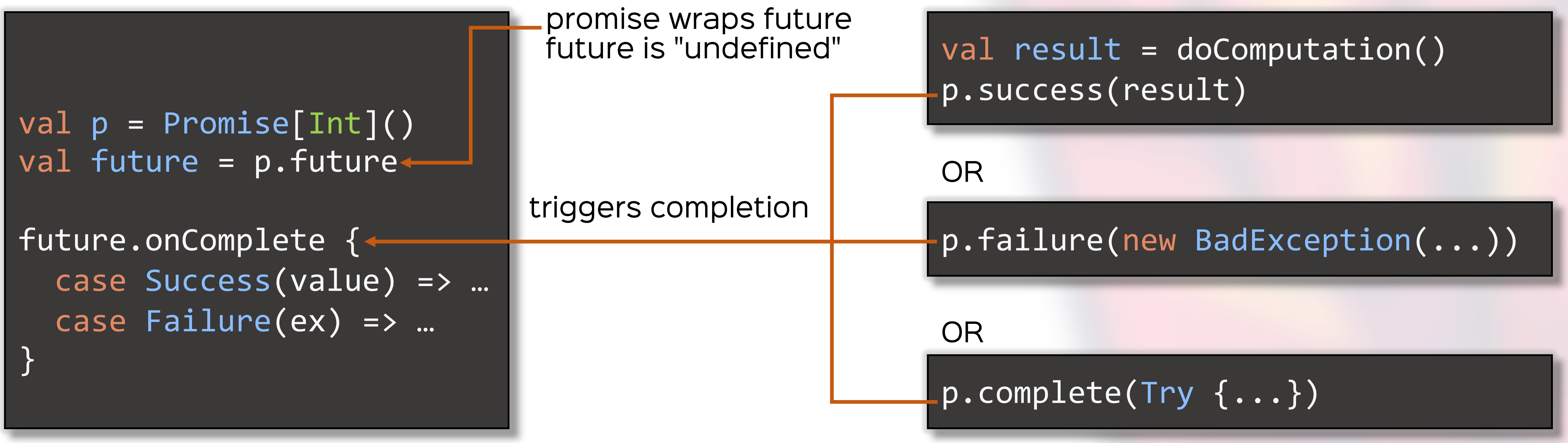
Promises are "writable-once" containers over a future.

thread 1:

- creates an empty promise
- knows how to handle the result

thread 2:

- holds the promise
- fulfills or fails the promise



```scala
val p = Promise[Int]()
val future = p.future

future.onComplete {
  case Success(value) => …
  case Failure(ex) => …
}
```

```scala
val result = doComputation()
p.success(result)
```

OR

```scala
p.failure(new BadException(...))
```

OR

```scala
p.complete(Try {...})
```

Scala rocks