Effective Scala Futures

Future[Presentation]

Problem - Global Objects and Untestability

Library (say Slick) forces you to use Futures

Call method, end up with:

```
Cannot find an implicit ExecutionContext. You might pass
    an (implicit ec: ExecutionContext) parameter to your method or import
    scala.concurrent.ExecutionContext.Implicits.global.
```

Most end up just using import (it works!)

Congratulations, you just used a global variable

Suddenly testing becomes hard

Await.result wastes one thread, ideally we never use it

If you are in Play! There is now a threadpool used by play and one used by anything importing the Scala implicit

The Solution - Dependency Injection

Add an implicit ExecutionContext to all of your methods:

```
def addAsync(a: Int, b: Int)(implicit ec: ExecutionContext): Future[Int] = {
  client.callAddService(a, b).map { result => result.value }
}
```

Add an implicit ExecutionContext to your class:

```
Class MyAsyncClass(service: MyService)(implicit ec: ExecutionContext)
```

Now everything can be injected, we can reuse the ExecutionContext and even change it

```
"can add two numbers" in { implicit ee: ExecutionEnv =>
  service.addAsync(4, 8) mustEqual 12.await
}
```

Problem - Long running DB requests stop site

Several long-running DB operations use up pool

Bulkhead separate thread pools for handling different tasks

Pass them into services with DI

```
class MathController(mathService: MathService)(implicit ex:
ExecutionContext) {
  def add(a: Int, b: Int) = Action.async {
    mathService.adAsync(a, b).map { result =>
      Ok(s"result is $result")
    } recover {
      case _: RejectedExecutionException =>
        ServiceUnavailable("Try later")
      case _: Throwable =>
        InternalServerError("Whoops")
    }
  }
}
```

```
// When using play, otherwise use Executors and
ExecutionContext.fromExecutor
object Contexts {
  private val dispatch = Akka.system.dispatchers
  val simpleDbLookups = dispatch .lookup("simple-db-lookups")
  val expensiveDbLookups = dispatch .lookup("expensive-db-lookups")
  val remoteCalls = dispatch .lookup("db-write-operations")
}
```

```
// Guice module
class DbRepoModule() extends AbstractModule {
  override def configure(): Unit = {
    bind(classOf[ExecutionContext])
      .annotatedWith(Names.named("db-ec"))
      .toInstance(Context.simpleDbLookups)
  }
}
```