Asynchronous Web Apps with Play 2.0

Oscar Renalias

What we'll do today

Take a look at Play's asynchronous capabilities

Real world usage scenarios

About me

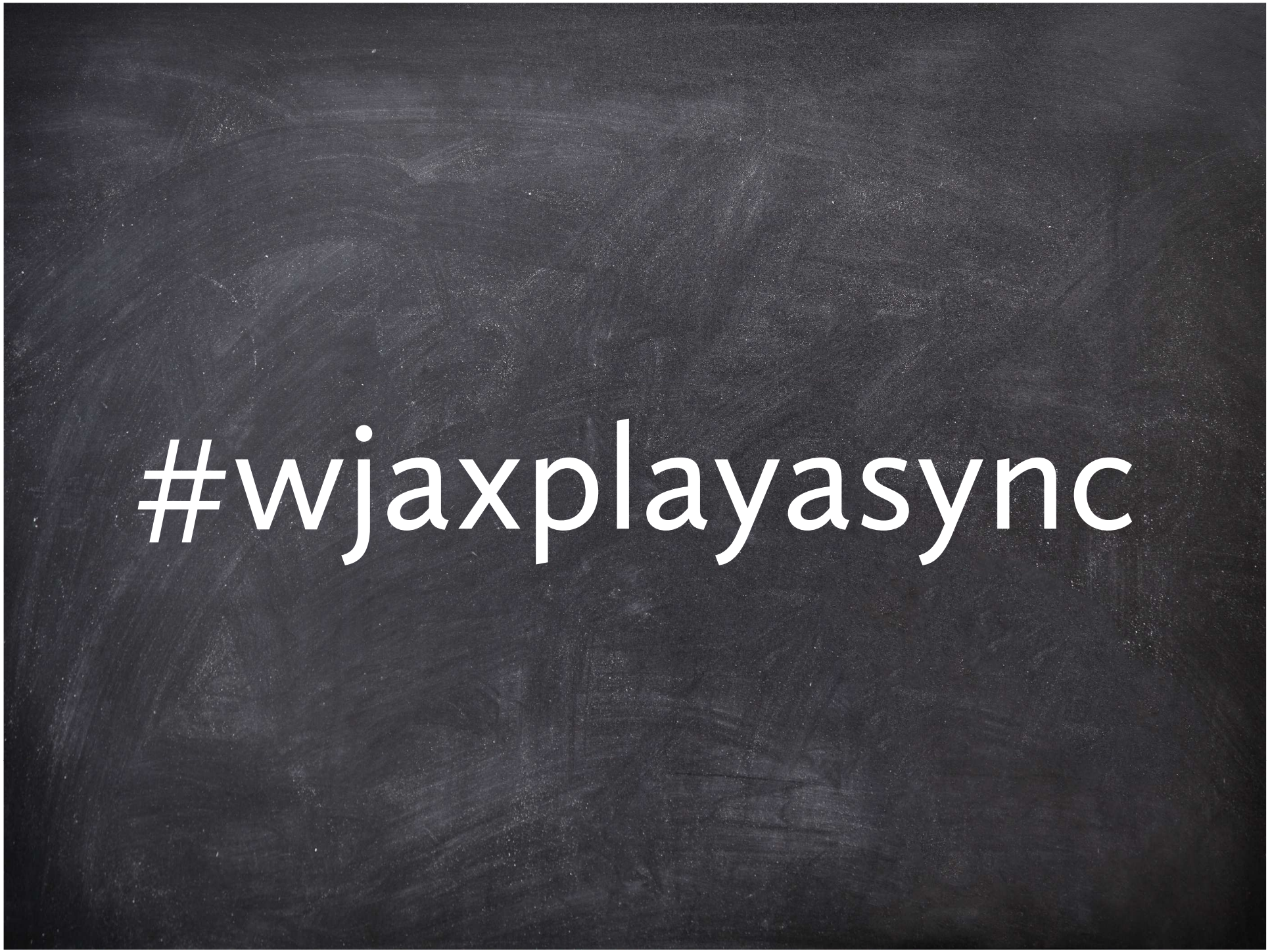
Oscar Renalias

oscarrenalias

github.com/oscarrenalias

> oscar.renalias@accenture.com

✉ oscar@renalias.net


#wjaxplayasync

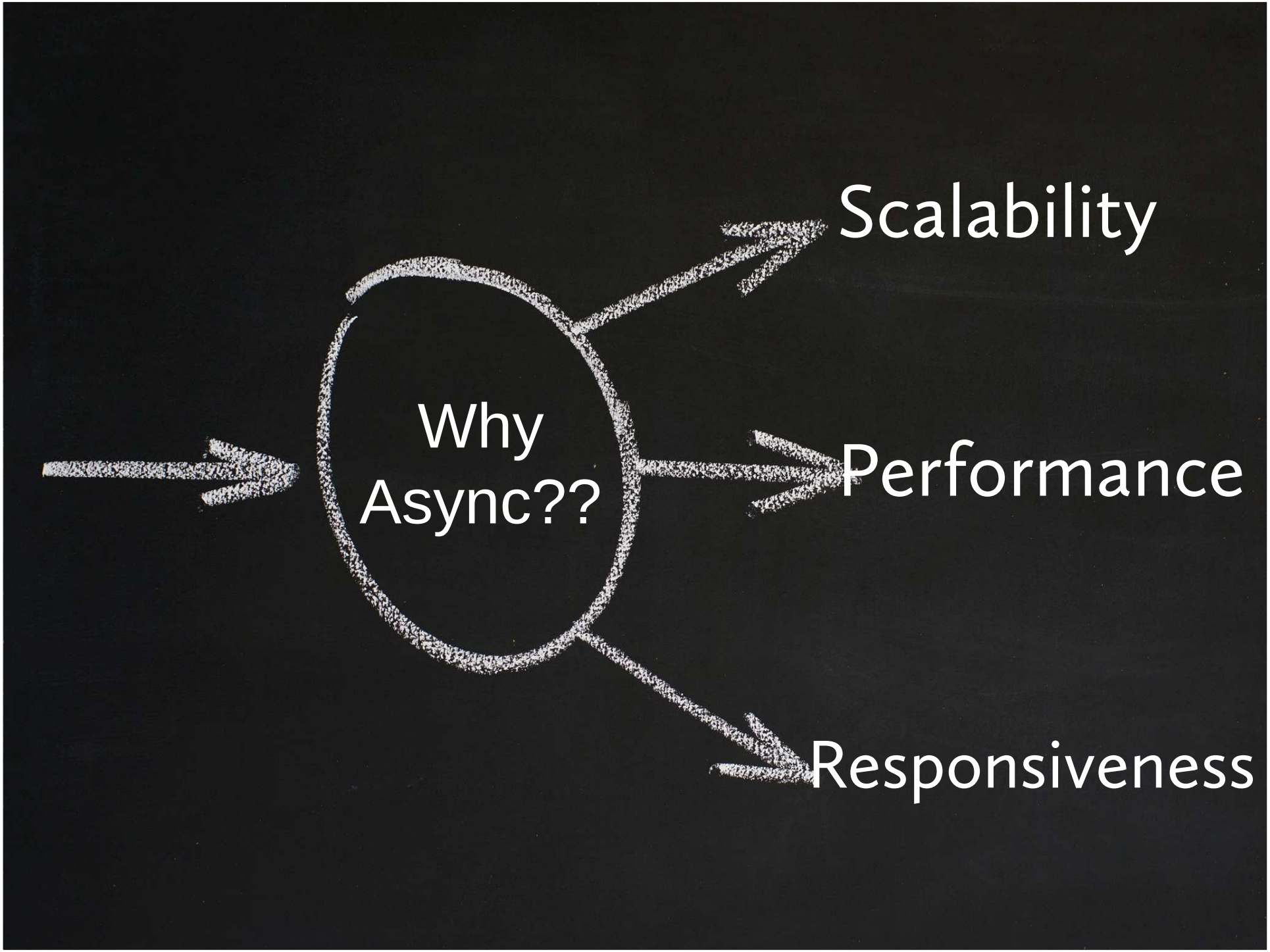
?

Stateless

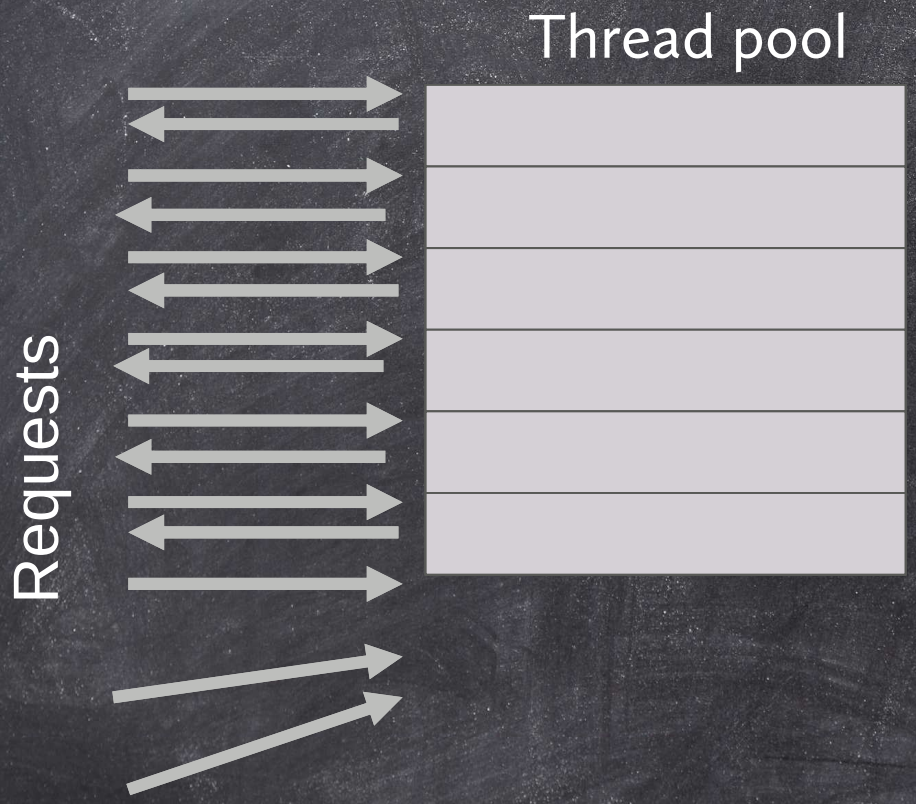
Asynchronous

Reactive

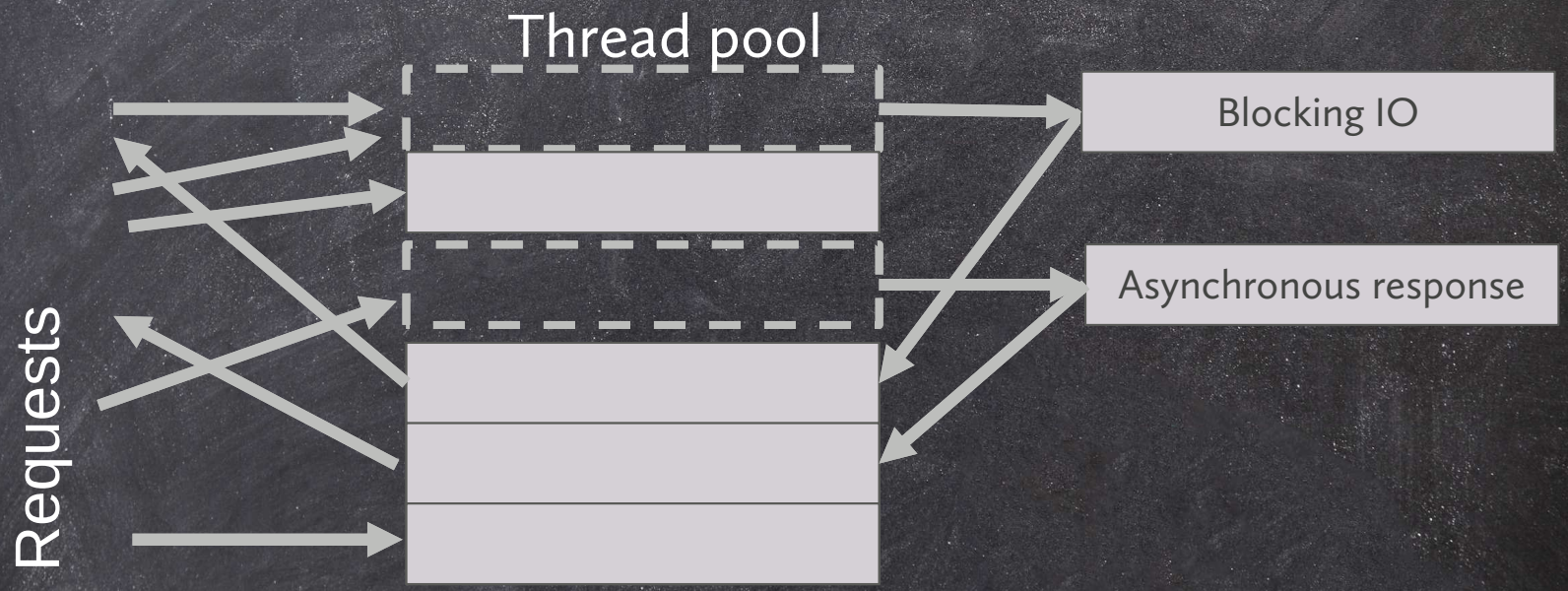
RESTful

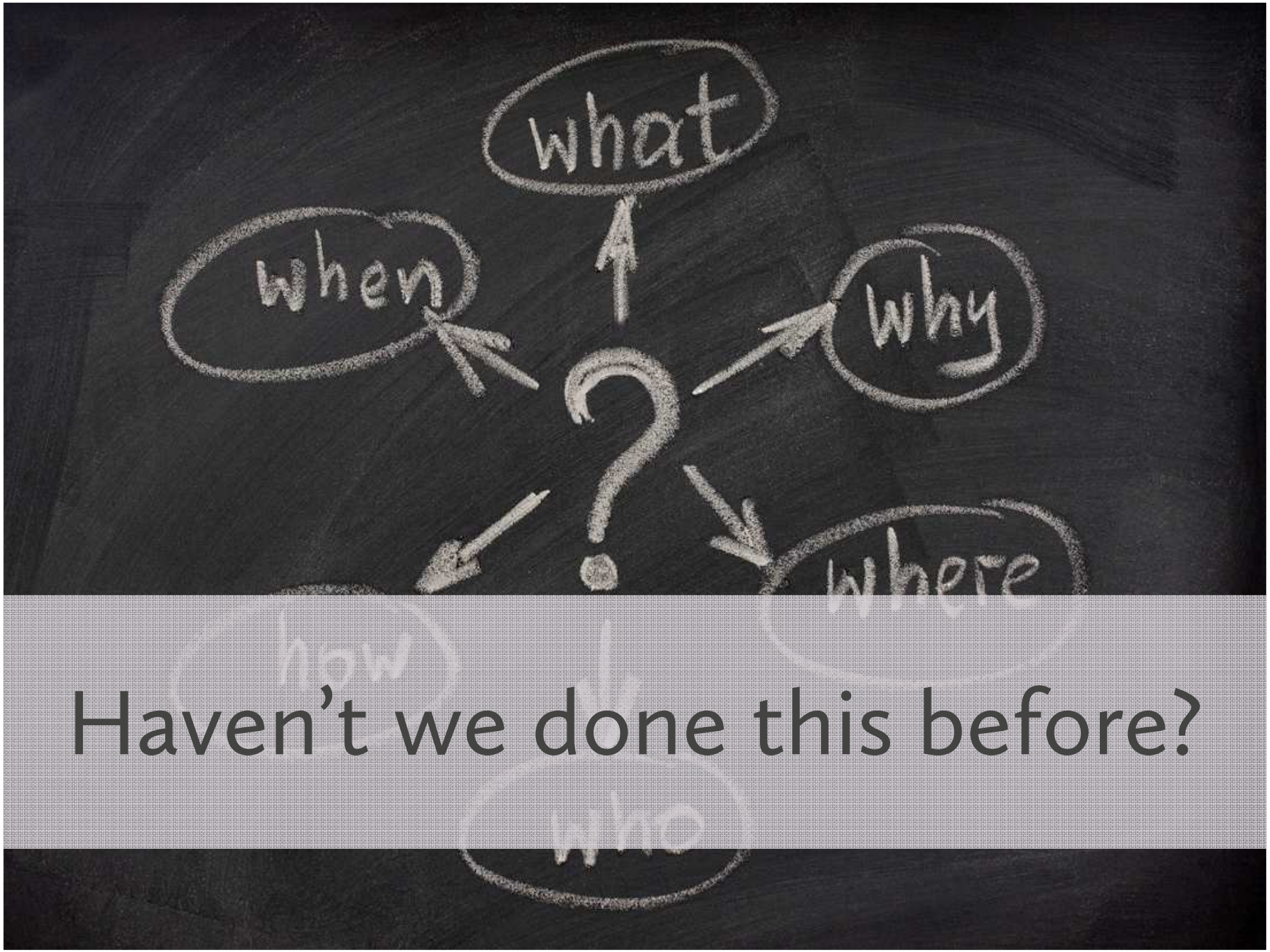

Why
Async??
Scalability
Performance
Responsiveness

Traditional Request Processing Model



Play's Model





Haven’t we done this before?

Other attempts

App-server specific, e.g. Jetty continuations

NIO/Netty

Servlet 3.0

Vert.x

Node.js

BlueEyes

Play's asynchronous capabilities

Asynchronous requests

Non-blocking reactive IO

Asynchronous Requests

Futures

Promises

Planning for the Future

A Future is a read-only placeholder for a value that may be available at a later stage

Futures and Promises

```
val f:Promise[Something] = Akka.future {
  longOperation()
}

val f2:Promise[SomethingElse] =
  f.map(s=>handleContents(s))

f2.value.fold(
  ex=> handleError(ex),
  value=> handleOk(value)
)
```

Asynchronous Results

```
val promiseOfPIValue: Promise[Double] =
    computePIAsynchronously()

val promiseOfResult: Promise[Result] =
    promiseOfPIValue.map { pi =>
        Ok("PI value computed: " + pi)
}
```

Promise[Something] → Promise[Result] → Browser

Making our actions asynchronous with asynchronous responses

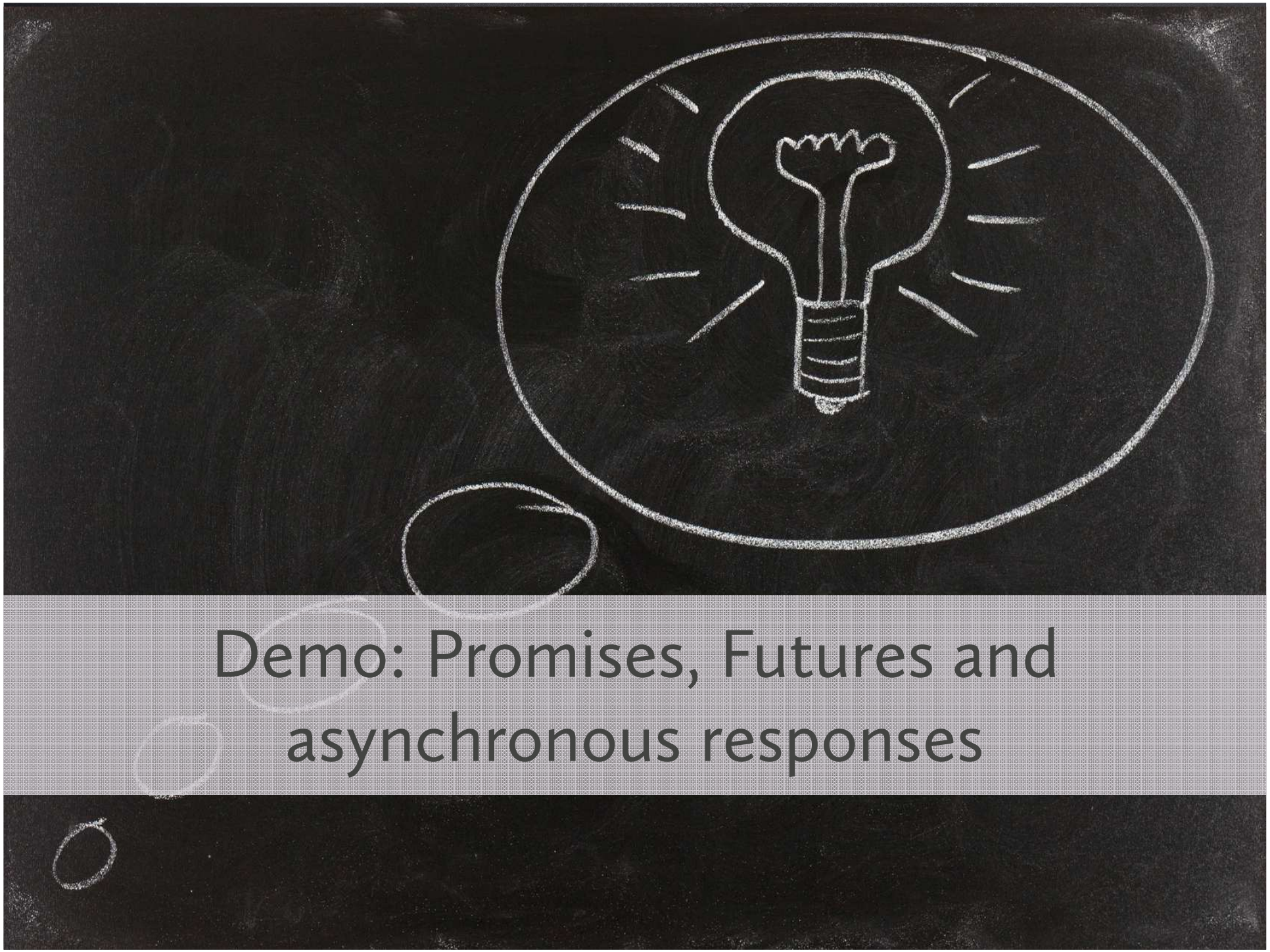
```
object MyController extends Controller {
 def simpleAsyncAction = Action {
  val p:Promise[Result] = Akka.future {
   val someResult = longOperation()
    Ok(someResult)
   }
  AsyncResult(p)
 }
}
```

```
def orders = Action {
    Async {
      Akka.future {
        SalesOrder.findAll
      } orTimeout(Ok(Json.toJson(JsonError("Timeout"))), 5,
SECONDS) map { orders =>
        orders.fold(
          orders => Ok(Json.toJson(orders)),
          error => Ok(Json.toJson(JsonError("Error")))
        )
      }
    }
  }
```

AsyncResult gets very tedious very soon. Can we do better?

Asynchronous responses in the real world

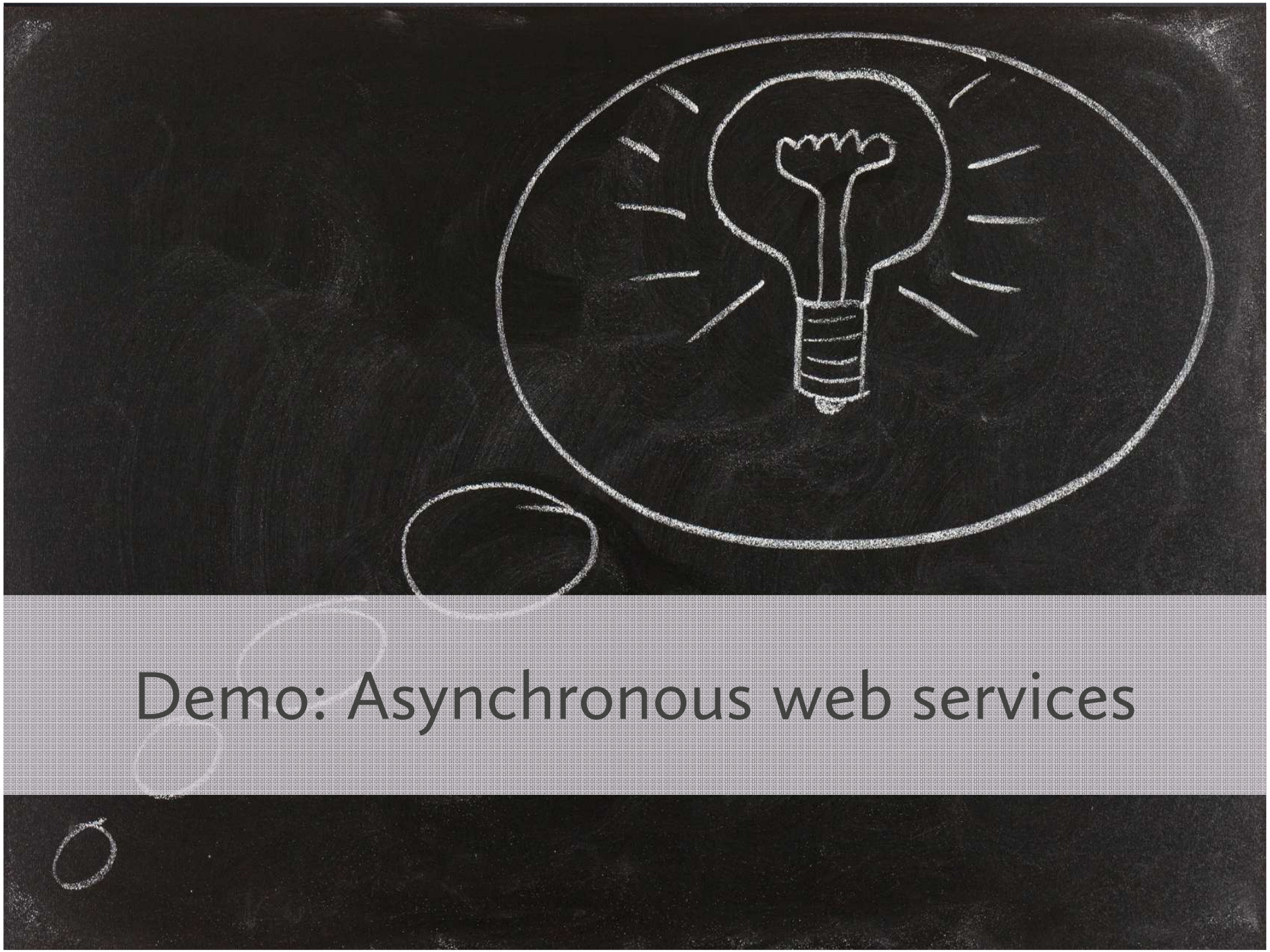
```
def WithFuture[T](seconds:Int)(f: => T)(implicit jsonHelper:Writes[T]) = {
   Async {
     Akka.future {
       f
     } orTimeout(Ok(Json.toJson(JsonError("...")")), seconds, SECONDS) map
{ result =>
       result.fold(
         data => Ok(Json.toJson(data)),
         error => Ok(Json.toJson(JsonError("Error")))
       )
     }
   }
 }

 def prettyOrders = Action {
   WithFuture(1) {
     SalesOrder.findAll
   }
 }
```



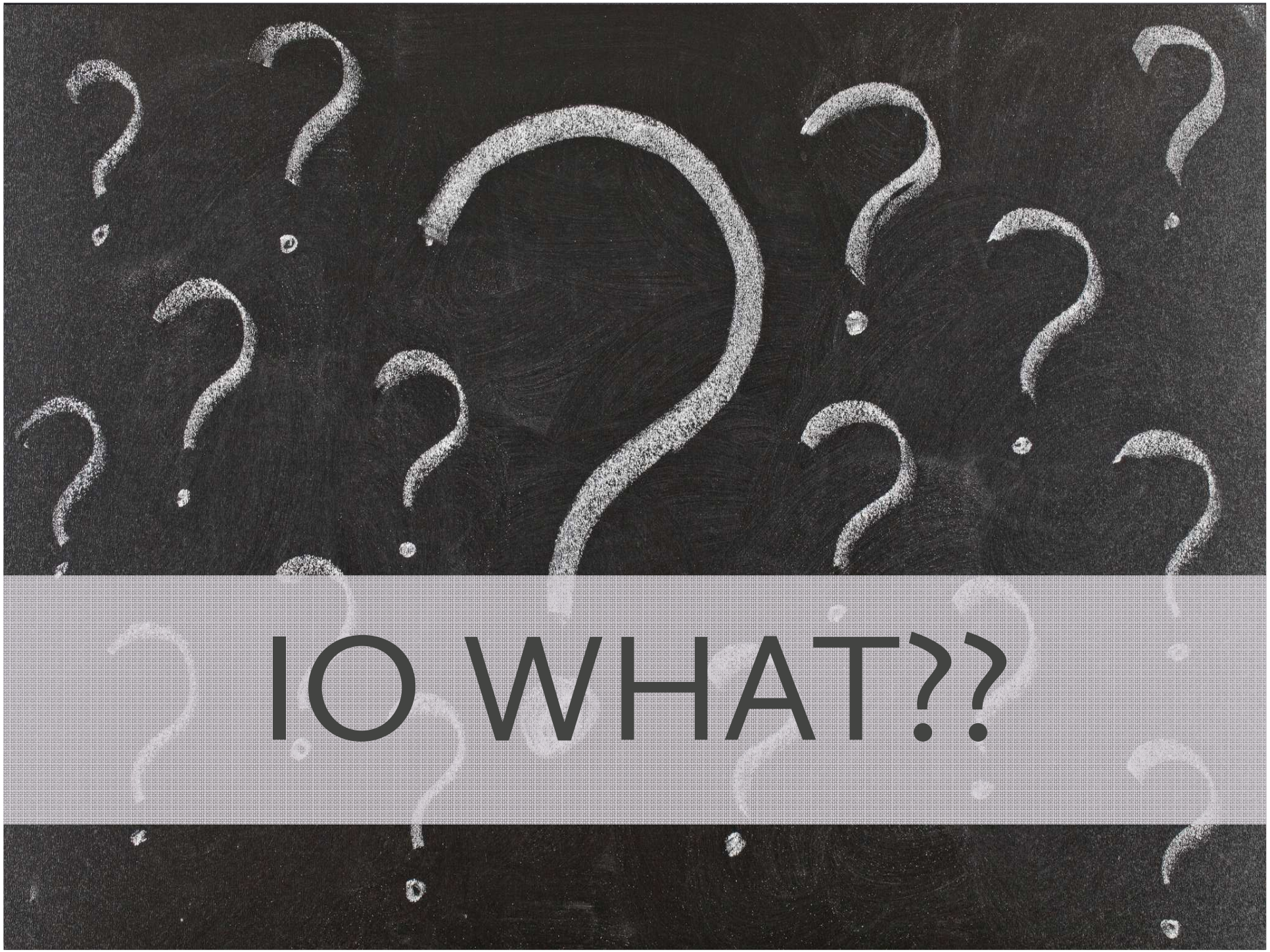
Demo: Promises, Futures and asynchronous responses

Asynchronous web services

```
Async {
   WS.url("http://.../").get().map { resp =>
      Ok(someJsonContent(response))
   }
}
```



Demo: Asynchronous web services


IO WHAT??

Reactive IO

"Don't call us, we'll call you"

Reactive IO

Enumerator = producer

Iteratee = consumer

Enumerators: the theory

```
trait Enumerator[E] {
   def apply[A](i: Iteratee[E, A]):
Promise[Iteratee[E, A]]
}
```

Enumerator(Iteratee) → Promise[AnotherIteratee]

Enumerators produce data

```
val stringEnumerator = Enumerator("one", "two",
"three", "four")

 val updateGenerator = Enumerator.fromCallback
{ () => Promise.timeout(Some(Update.random),
5000 milliseconds)
}

val e = Enumerator.imperative(...)
e.push(“data”)
e.push(“more data”)
```

Behind the scenes with Iteratees

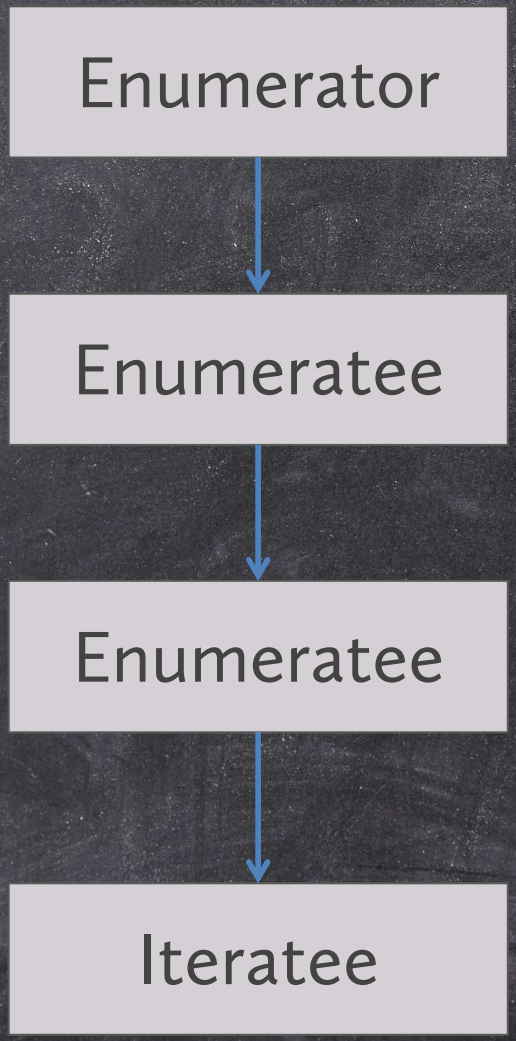
```
def fold[B](
 done: (A, Input[E]) => Promise[B],
 cont: (Input[E] => Iteratee[E,A]) => Promise[B],
 error: (String, Input[E]) => Promise[B]
): Promise[B]
```

- Done: there is no more input
- Cont: more input incoming
- Error: there was an error with the input

Simplified Iteratees

```
Iteratee.foreach
Iteratee.fold
Iteratee.consume
```

Enumeratees



Useful Enumeratees

`Enumeratee.map`
`Enumeratee.filter`
`Enumeratee.drop`
`Enumeratee.take`

Composability

```
val dataGenerator = Enumerator.fromCallback { () =>
   Promise.timeout(Some(new
   java.util.Random().nextInt(100)),
   5000 milliseconds)
}

val toStr = Enumeratee.map[Int] { x => x.toString }

val toConsole = Iteratee.foreach[String](println(_))

dataGenerator &> toStr |>> toConsole
```

Reactive IO and HTTP responses

```
Ok.feed(iteratee)
Ok.stream(iteratee)
Ok.stream(enumerator)
```

Reactive IO in the real world

Streaming APIs
File streaming
Server-generated events
Reactive data
WebSockets

Streaming Files

```
Enumerator.fromFile
```

Or

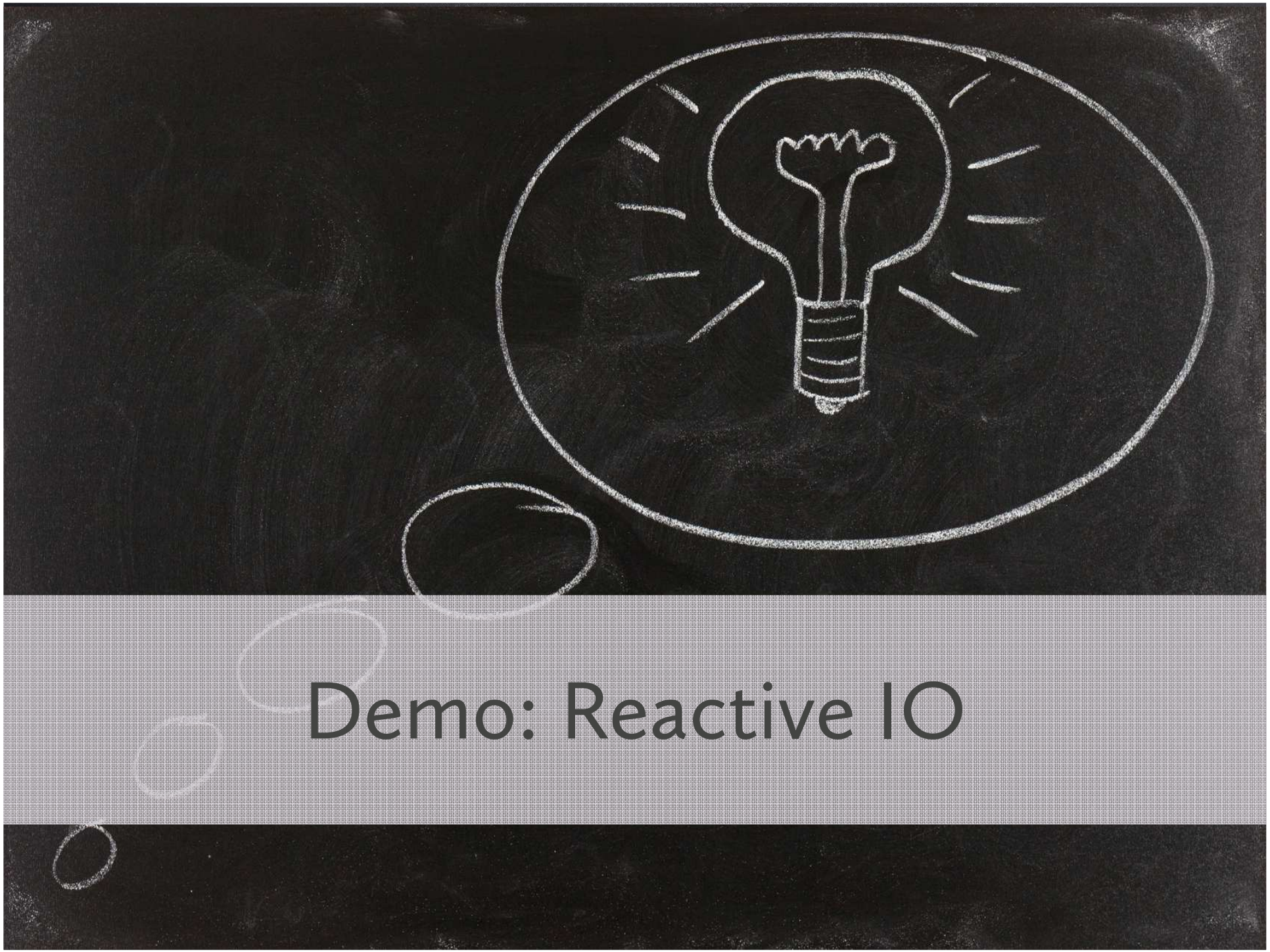
```
Ok.sendFile(new File(…))
```

Streaming APIs

Data source (Enumerator) → Enumeratee → Client

WebSockets

```
 def websocketTime = WebSocket.async[String] { request =>
    Akka.future {
      val timeEnumerator = Enumerator.fromCallback { () =>
        Promise.timeout(Some((new Date).toString()), 5000
milliseconds)
      }

      val in = Iteratee.foreach[String] { message =>
          println(message)
       }

      (in, timeEnumerator)
    }
  }
```



Demo: Reactive IO

A glimpse of the future: Reactive Mongo

```
val cursor = collection.find(query)
val futureListOfArticles: Future[List[Article]] =
cursor.toList
futureListOfArticles.onSuccess { articles =>
  for(article <- articles)
    println("found article: " + article)
}
```

All work and no Play! makes Jack a dull boy (but there's still hope)

Grails 2.0 + Servlet 3.0

```
def index() {
    def ctx = startAsync()
    ctx.start {
        new Book(title:"The Stand").save()
        render template:"books", model:[books:Book.list()]
        ctx.complete()
    }
}
```

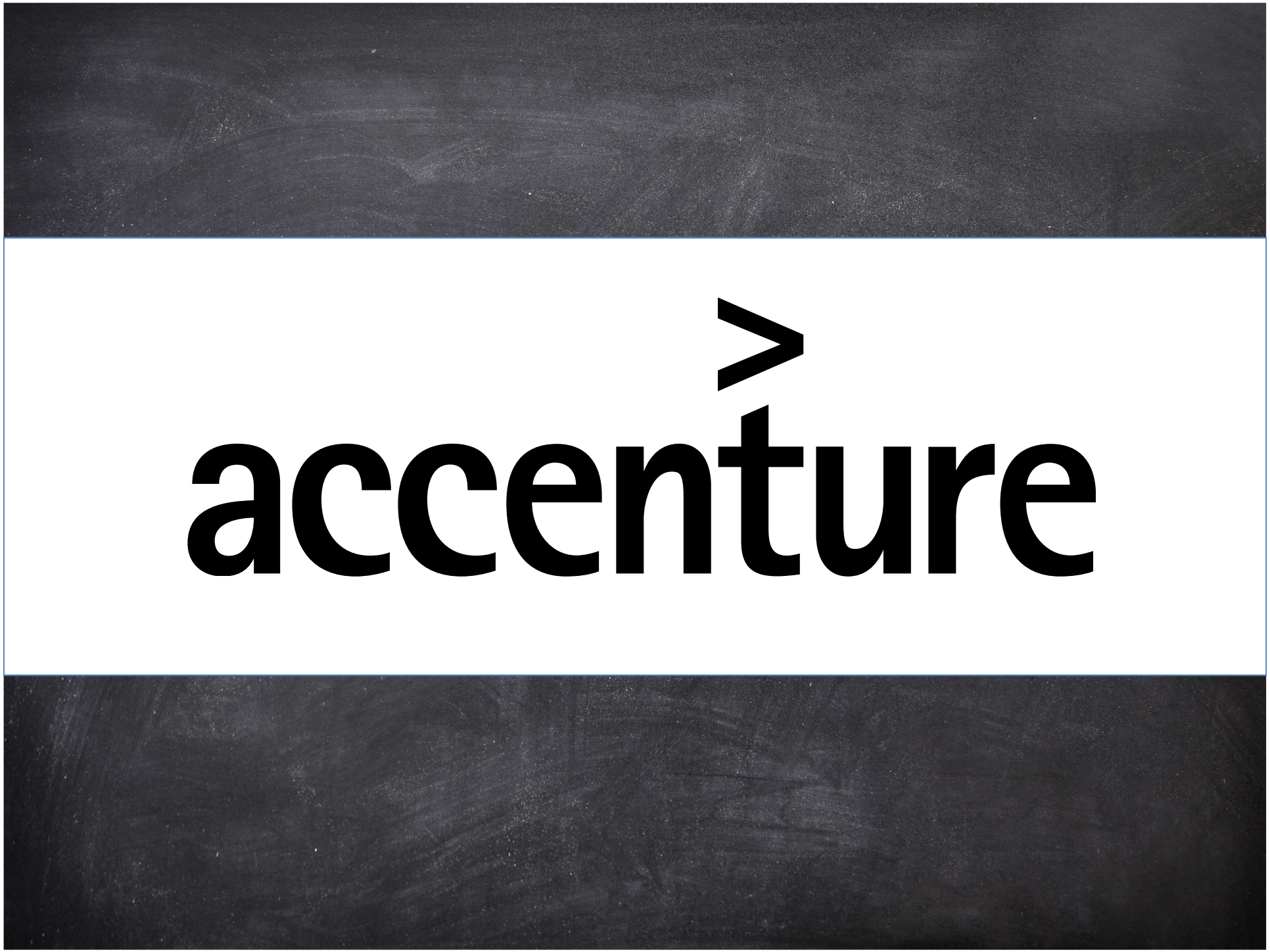
All work and no Play! makes Jack a dull boy (but there's still hope)

Vert.x

```
public class ServerExample extends Verticle {

  public void start() {
    vertx.createHttpServer().requestHandler(new Handler<HttpServerRequest>() {
      public void handle(HttpServerRequest req) {
        System.out.println("Got request: " + req.uri);
        System.out.println("Headers are: ");
        for (String key : req.headers().keySet()) {
          System.out.println(key + ":" + req.headers().get(key));
        }
        req.response.headers().put("Content-Type", "text/html; charset=UTF-8");
        req.response.end("<html><body><h1>Hello from
vert.x!</h1></body></html>");
      }
    }).listen(8080);
  }
}
```

Github repo or it didn't happen

https://github.com/oscarrenalias/wjax-2012-play-async-apps


accenture


Thank you