Cross-Thread Stack Traces

The problem

```
func() {
 Future {}
}
```

```
// processing
throw
```

```
def func(): Future[String] = {
  Future {
    // ... some processing
    throw new Exception("Error!")
  }
}
```

The problem

```
func() {
 Future {}
}
```

```
// processing
throw
```

```
Exception in thread "main" java.lang.Exception: Error!
  at dummy$.$anonfun$func$1(dummy.scala:10)
  at scala.concurrent.Future$.$anonfun$apply$1(Future.scala:655)
  at scala.util.Success.$anonfun$map$1(Try.scala:251)
  at scala.util.Success.map(Try.scala:209)
  at scala.concurrent.Future.$anonfun$map$1(Future.scala:289)
  at scala.concurrent.impl.Promise.liftedTree1$1(Promise.scala:29)
  at scala.concurrent.impl.Promise.$anonfun$transform$1(Promise.scala:29)
  at scala.concurrent.impl.CallbackRunnable.run(Promise.scala:60)
  at scala.concurrent.impl.ExecutionContextImpl$AdaptedForkJoinTask.exec(ExecutionContextImpl.scala:140
  at java.util.concurrent.ForkJoinTask.doExec(ForkJoinTask.java:289)
  at java.util.concurrent.ForkJoinPool$WorkQueue.runTask(ForkJoinPool.java:1056)
  at java.util.concurrent.ForkJoinPool.runWorker(ForkJoinPool.java:1692)
  at java.util.concurrent.ForkJoinWorkerThread.run(ForkJoinWorkerThread.java:157)
```

```scala
def func(): Future[String] = {
  Future {
    // ... some processing
    throw new Exception("Error!")
  }
}
```

We want to get

```
func() {
 Future {}
}
```

```
// processing
throw
```

```scala
def func(): Future[String] = {
  Future {
    // ... some processing
    throw new Exception("Error!")
  }
}
```

```
Exception in thread "main" java.lang.Exception: Error!
  at dummy$.func
  at dummy$.main
  at dummy.main
```

+

```
Exception in thread "main" java.lang.Exception: Error!
  at dummy$.$anonfun$func$1(dummy.scala:10)
  at scala.concurrent.Future$.$anonfun$apply$1(Future.scala:655)
  at scala.util.Success.$anonfun$map$1(Try.scala:251)
  at scala.util.Success.map(Try.scala:209)
  at scala.concurrent.Future.$anonfun$map$1(Future.scala:289)
  at scala.concurrent.impl.Promise.liftedTree1$1(Promise.scala:29)
  at scala.concurrent.impl.Promise.$anonfun$transform$1(Promise.scala:29)
  at scala.concurrent.impl.CallbackRunnable.run(Promise.scala:60)
  at scala.concurrent.impl.ExecutionContextImpl$AdaptedForkJoinTask.exec(ExecutionContextImpl.scala:140
  at java.util.concurrent.ForkJoinTask.doExec(ForkJoinTask.java:289)
  at java.util.concurrent.ForkJoinPool$WorkQueue.runTask(ForkJoinPool.java:1056)
  at java.util.concurrent.ForkJoinPool.runWorker(ForkJoinPool.java:1692)
  at java.util.concurrent.ForkJoinWorkerThread.run(ForkJoinWorkerThread.java:157)
```

Origin of a stack trace

```
def func(): Future[String] = {
  Future {
    // ... some processing
    throw new Exception("Error!")
  }
}
```

```
public class Throwable {
  public Throwable() {
    fillInStackTrace();
  }
}
```

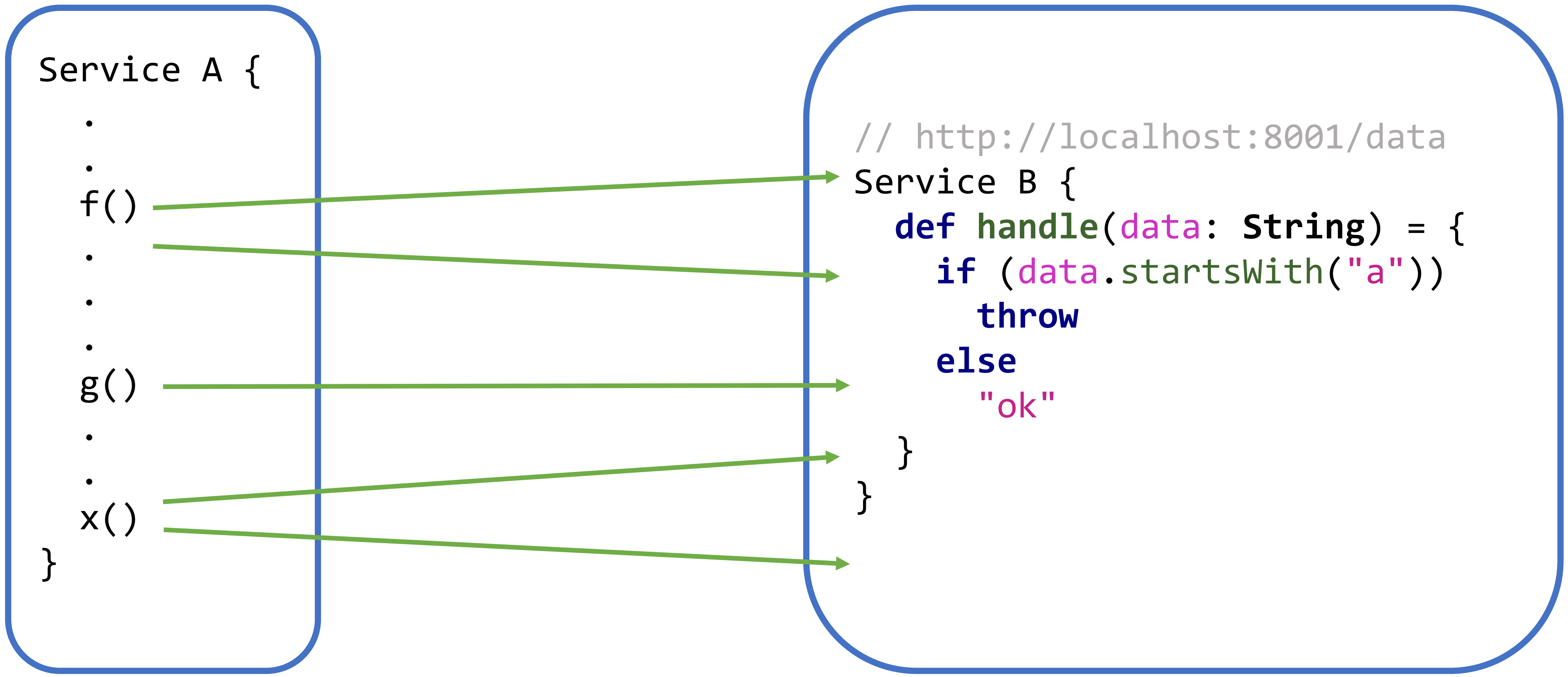
Origin of a stack trace

```
def func(): Future[String] = {
  Future {
    // ... some processing
    throw new Exception("Error!")
  }
}
```

```
public class Throwable {
  public Throwable() {
    fillInStackTrace();
  }
}
```

Get the
stack trace
here

Use case



```
Service A {
    .
    .
    f()
    .
    .
    .
    g()
    .
    .
    x()
}
```

```
// http://localhost:8001/data
Service B {
  def handle(data: String) = {
    if (data.startsWith("a"))
      throw
    else
      "ok"
  }
}
```

Conclusion

- Cross-thread stack traces can give you a lot of added information
- Add the `withTrace` call at the end of your futures and for comprehensions
- Useful with Akka's ask pattern
  - `val f = (actor ? "message").withTrace`
- Performance?
  - This implementation gets the stack trace even if no exceptions occur
- https://gist.github.com/spore1/38bd79cac7204540a03c2dd422273c6d
- Or search gist.github.com for "cross thread trace"