

# Combinators on Futures (1/2)

Principles of Reactive Programming

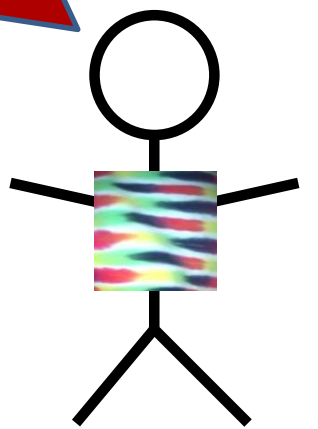
Erik Meijer

# Futures recap

```
trait Awaitable[T] extends AnyRef {  
  abstract def ready(atMost: Duration):  
  abstract def result(atMost: Duration)  
}
```

**All these methods  
take an implicit  
execution context**

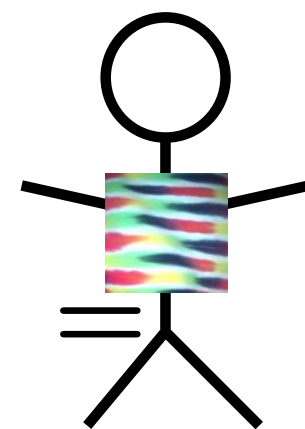
```
trait Future[T] extends Awaitable[T] {  
  def filter(p: T=>Boolean): Future[T]  
  def flatMap[S] (f: T=>Future[S]): Future[U]  
  def map[S] (f: T=>S): Future[S]  
  def recoverWith(f: PartialFunction[Throwable,  
Future[T]]): Future[T]  
}  
object Future {  
  def apply[T] (body : =>T): Future[T]  
}
```



# Sending packets using futures

```
val socket = Socket()
val packet: Future[Array[Byte]] =
  socket.readFromMemory()
packet onComplete {
  case Success(p) => {
    val confirmation: Future[Array[Byte]] =
      socket.sendToEurope(p)
  }
  case Failure(t) => ...
}
```

**Remember  
this mess?**



# Flatmap to the rescue

```
val socket = Socket()
val packet: Future[Array[Byte]] =
    socket.readFromMemory()

val confirmation: Future[Array[Byte]] =
    packet.flatMap(p => socket.sendToEurope(p))
```

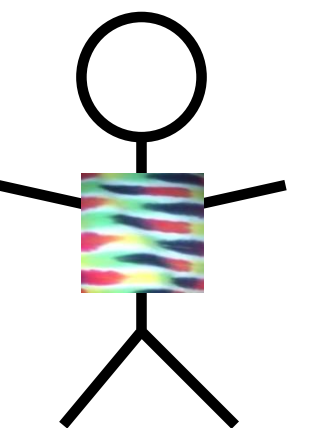
# Sending packets using futures under the covers

```
import scala.concurrent.ExecutionContext.Implicits.global
import scala.imaginary.Http._

object Http {
  def apply(url: URL, req: Request): Future[Response] =
    {... runs the http request asynchronously ...}
}

def sendToEurope(packet: Array[Byte]): Future[Array[Byte]] =
  Http(URL("mail.server.eu"), Request(packet))
    .filter(response => response.isOK)
    .map(response => response.toByteArray)
```

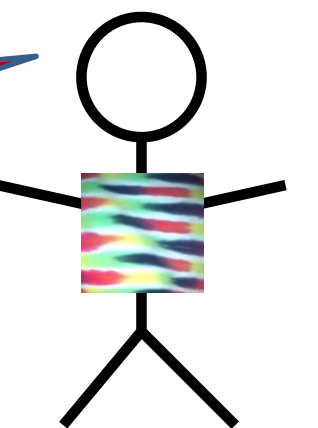
**But, this can  
still fail!**



# Sending packets using futures robustly (?)

```
def sendTo(url: URL, packet: Array[Byte]): Future[Array[Byte]]  
  Http(url, Request(packet))  
    .filter(response => response.isOK)  
    .map(response => response.toByteArray)  
  
def sendToAndBackup(packet: Array[Byte]):  
  Future[(Array[Byte], Array[Byte])] = {  
  
    val europeConfirm = sendTo(mailServer.europe, packet)  
    val usaConfirm = sendTo(mailServer.usa, packet)  
    europeConfirm.zip(usaConfirm)  
  }
```

Cute, but no  
cigar



# Send packets using futures robustly

```
def recover(f: PartialFunction[Throwable, T]): Future[T]
```



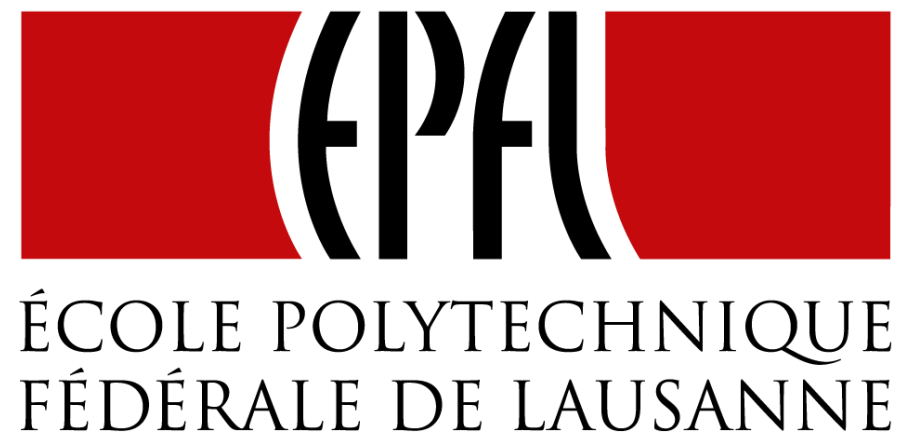
```
def recoverWith(f: PartialFunction[Throwable, Future[T]])  
: Future[T]
```

# Send packets using futures robustly

```
def sendTo(url: URL, packet: Array[Byte]):  
Future[Array[Byte]] =  
  Http(url, Request(packet))  
    .filter(response => response.isSuccess)  
    .map(response => response.toByteArray)
```

```
def sendToSafe(packet: Array[Byte]):  
Future[Array[Byte]] =  
  sendTo(mailServer.europe, packet) recoverWith {  
    case europeError =>  
      sendTo(mailServer.usa, packet) recover {  
        case usaError => usaError.getMessage.toByteArray  
      }  
  }
```





# End of Combinators on Futures (1/2)

Principles of Reactive Programming

Erik Meijer