ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Composing Futures (1/2)

Principles of Reactive Programming

Erik Meijer

Flatmap …

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

Or comprehensions?

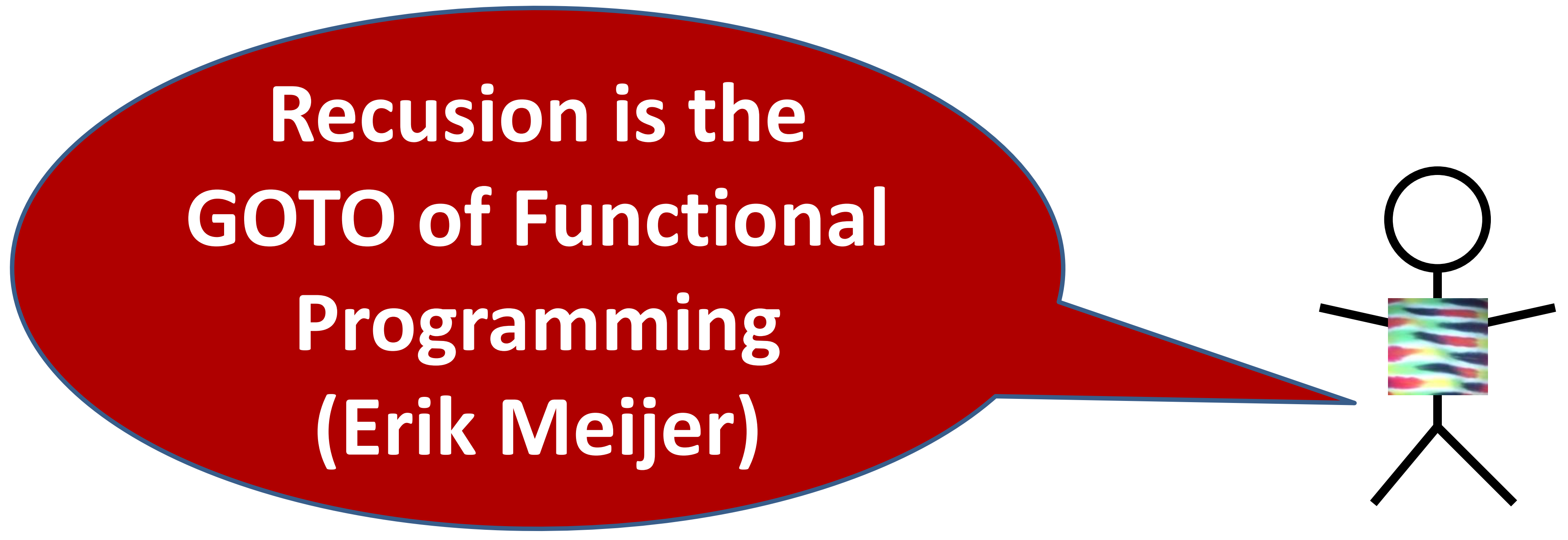
```
val socket = Socket()
val confirmation: Future[Array[Byte]] = for{
  packet       <- socket.readFromMemory()
  confirmation <- socket.sendToSafe(packet)
} yield confirmation
```

Retrying to send

```
def retry(noTimes: Int)(block: =>Future[T]):
Future[T] = {
    … retry successfully completing block
      at most noTimes
    … and give up after that
}
```

Retrying to send

```
def retry(noTimes: Int)(block: ⇒Future[T]):
Future[T] = {
  if (noTimes == 0) {
      Future.failed(new Exception(“Sorry”))
  } else {
      block fallbackTo {
          retry(noTimes-1){ block }
      }
  }
}
```


Recusion is the GOTO of Functional Programming (Erik Meijer)

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

End of Composing Futures (1/2)

Principles of Reactive Programming

Erik Meijer