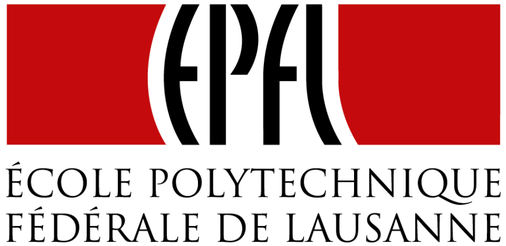


Monads and Effects (1/2)

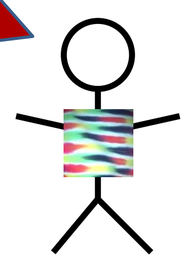
Principles of Reactive Programming

Erik Meijer

Warning

There is no type-checker for PowerPoint yet, hence these slides might contain typos and bugs. Hence, do not take these slides as the gospel or ultimate source of truth.

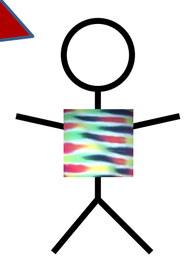
The only artifact you can trust is actual source code.



Warning

When we show code fragments in these lectures we really mean code *fragments*.

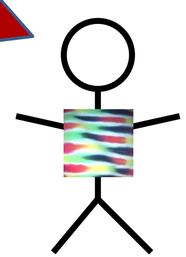
In particular, do not expect to be able to cut & past working code from the slides. You can find running & up-to-date on the GitHub site for this course.



Warning

When we use RxScala in these lectures, we assume version 0.23. Different versions of RxScala might not be compatible.

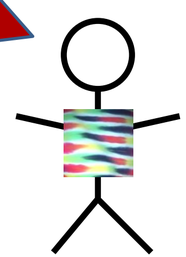
The RxScala method names do not necessarily correspond 1:1 with the underlying RxJava method names.



Warning

When we say "*monad*" in these lectures we mean a generic type with a constructor and a flatMap operator.

In particular, we'll be fast and loose about the monad laws (that is, we completely ignore them).



The Four Essential Effects In Programming

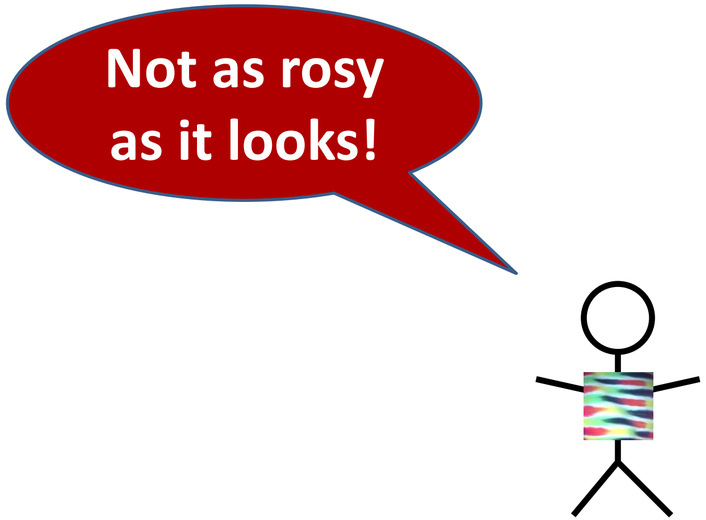
	One	Many
Synchronous	`T/Try[T]`	`Iterable[T]`
Asynchronous	`Future[T]`	`Observable[T]`

The Four Essential Effects In Programming

	One	Many
Synchronous	`T/Try[T]`	`Iterable[T]`
Asynchronous	`Future[T]`	`Observable[T]`

A simple adventure game

```
trait Adventure {
  def collectCoins(): List[Coin]
  def buyTreasure(coins: List[Coin]):
Treasure
}
```




```
val adventure = Adventure()
val coins = adventure.collectCoins()
val treasure = adventure.buyTreasure(coins)
```

Actions may fail

```
def collectCoins(): List[Coin] = {
  if (eatenByMonster(this))
    throw new GameOverException(
"Ooops")
  List(Gold, Gold, Silver)
}

val adventure = Adventure()
val coins = adventure.collectCoins()
val treasure = adventure.buyTreasure(coins)
```

Actions may fail

```
def buyTreasure(coins: List[Coin]):
Treasure = {
  if (coins.sumBy(_.value) < treasureCost)
    throw new GameOverException("Nice try!")
  Diamond
}

val adventure = Adventure()
val coins = adventure.collectCoins()
val treasure = adventure.buyTreasure(coins)
```

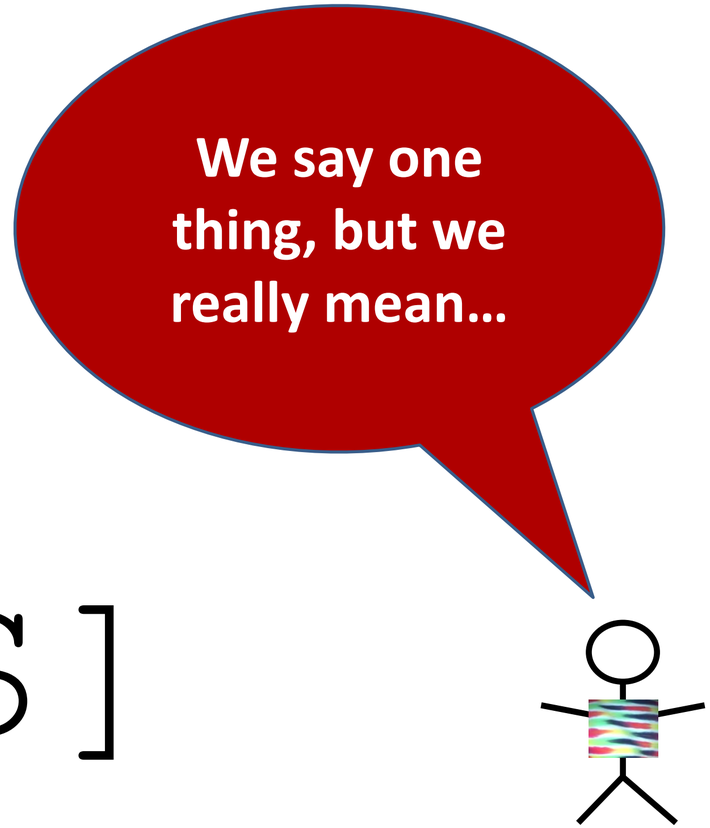
Sequential composition of actions that may fail

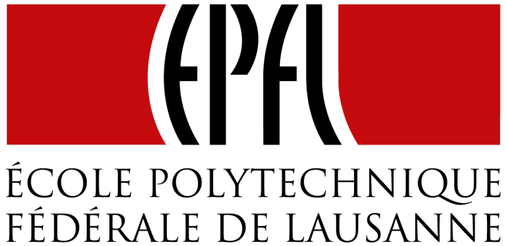
```
val adventure = Adventure()

val coins = adventure.collectCoi
// block until coins are collected
// only continue if there is no exception
val treasure = adventure.buyTreasure(coins)
// block until treasure is bought
// only continue if there is no exception
```

Expose possibility of failure in the types, honestly

```
T => S
```

```
T => Try[S]
```





End of Monads and Effects (1/2)

Principles of Reactive Programming

Erik Meijer