Scala в картинках

Цель презентации: понимание того, как можно решать простые задачи в функциональном стиле на Scala.

Задача: получить сок из нескольких яблок.

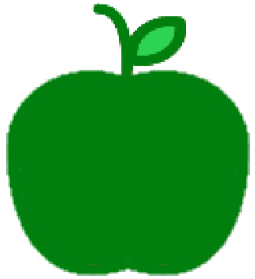
- Как мы представим сок?
- Как мы представим яблоко?
- Как мы получим из яблок сок?

Яблоко

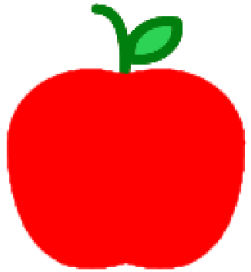
```
case class Apple(color: String, size: Int)
```

```
val apple1 = Apple("green", 2)
```

```
val apple2 = Apple("green", 3)
```



```
val apple3 = Apple("red", 3)
```



case class Apple - это чертеж яблока, по этому чертежу можно сделать яблоко!

Стакан сока и соковыжималка

```scala
case class GlassOfJuice(volume: Int)
```

```scala
case class Juicer() {
  def squeezeTheJuice(apple: Apple): GlassOfJuice = GlassOfJuice(apple.size * 100)
}
```

Как получить из яблока сок ?

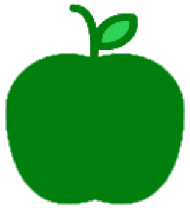
Тип (необязательно указывать)

```
val juicer: Juicer = Juicer()
```

Это типа соковыжималка
(не блендер)

Тип

```
val oneApple: Apple = Apple("green", 2)
```

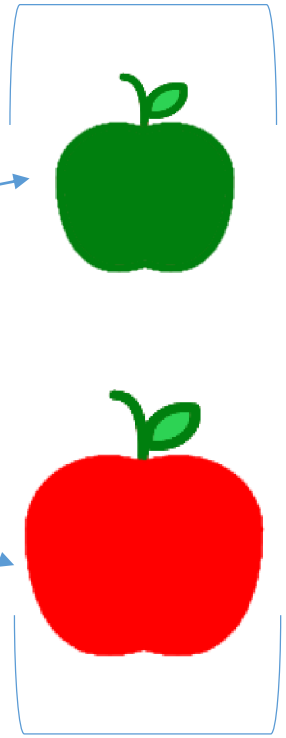


```
val onePortionJuice: GlassOfJuice = juicer.squeezeTheJuice(oneApple)
```

Не поверите! Это тоже тип

Несколько яблок (список яблок)

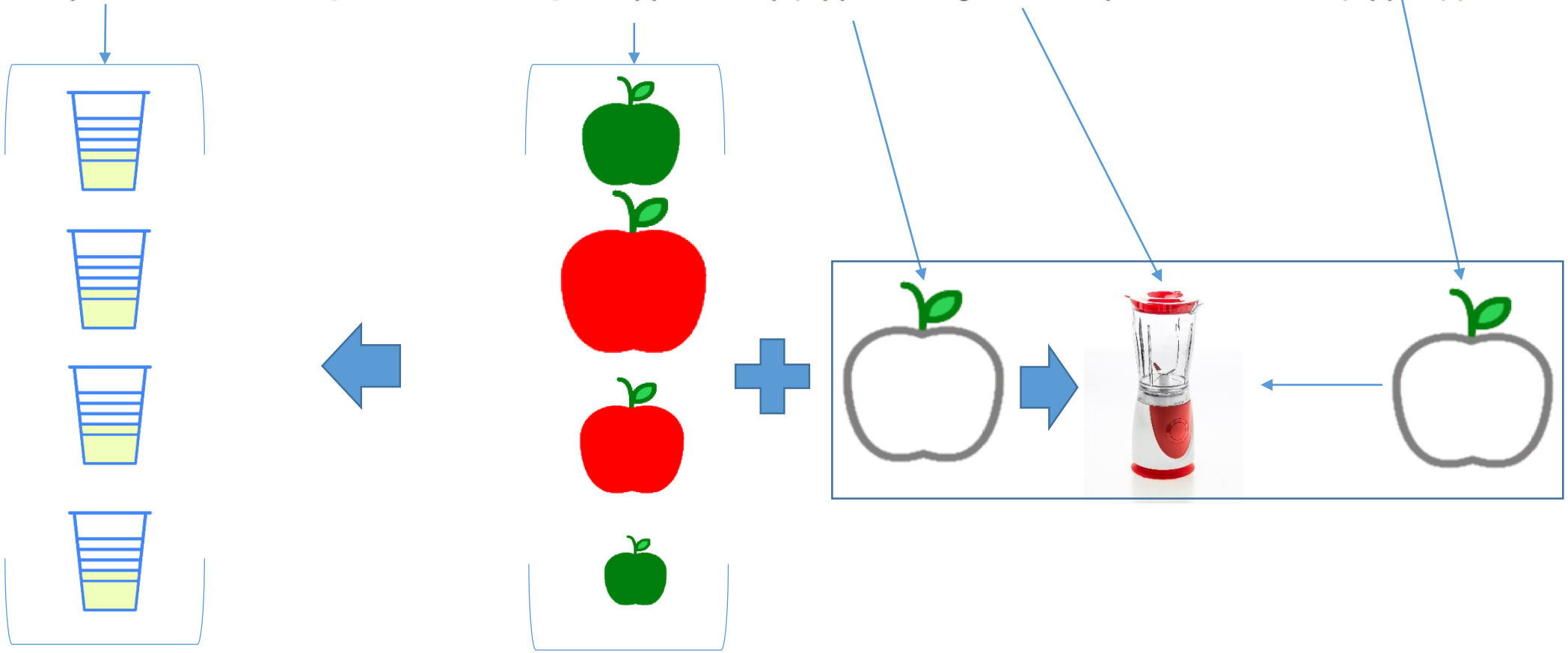
```
val listEmpty = List()

val listInt = List(1, 2, 10)

val listString = List("Hello", "world")

val listApple  = List(Apple("green", 2), Apple("red", 3))
```



Как получить из яблок сок ?

```
val apples: List[Apple] = List(Apple("green", 2), Apple("red", 3), Apple("red", 2), Apple("green", 1))
```

```
val cupsOfJuice: List[GlassOfJuice] = apples.map(apple => juicer.squeezeTheJuice(apple))
```

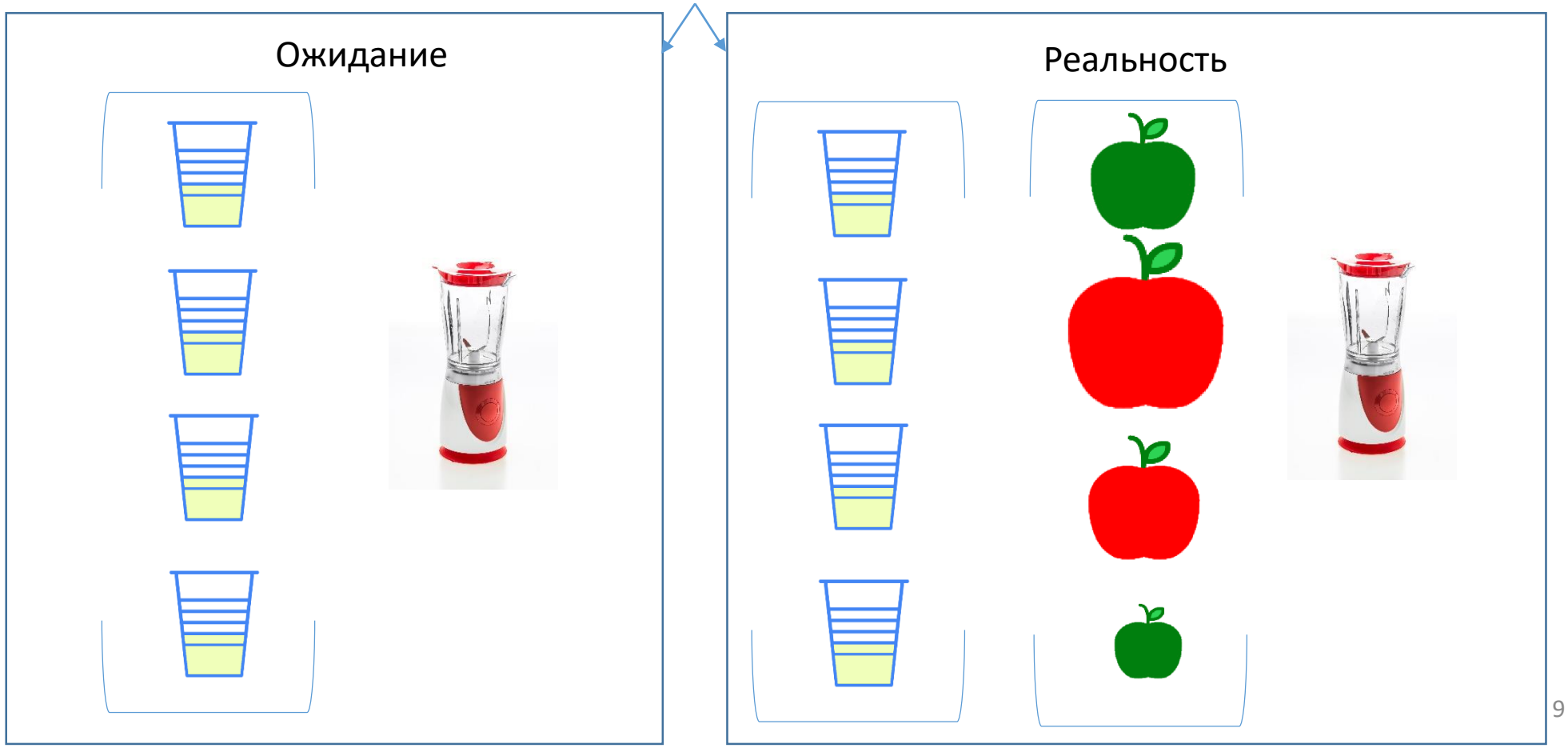


Функциональный мир – неизменяемый мир

МЫ НЕ ИЗМЕНЯЕМ СТАРЫЕ ДАННЫЕ (коллекции, экземпляры класса и т.д.)**, МЫ СОЗДАЕМ НОВЫЕ**

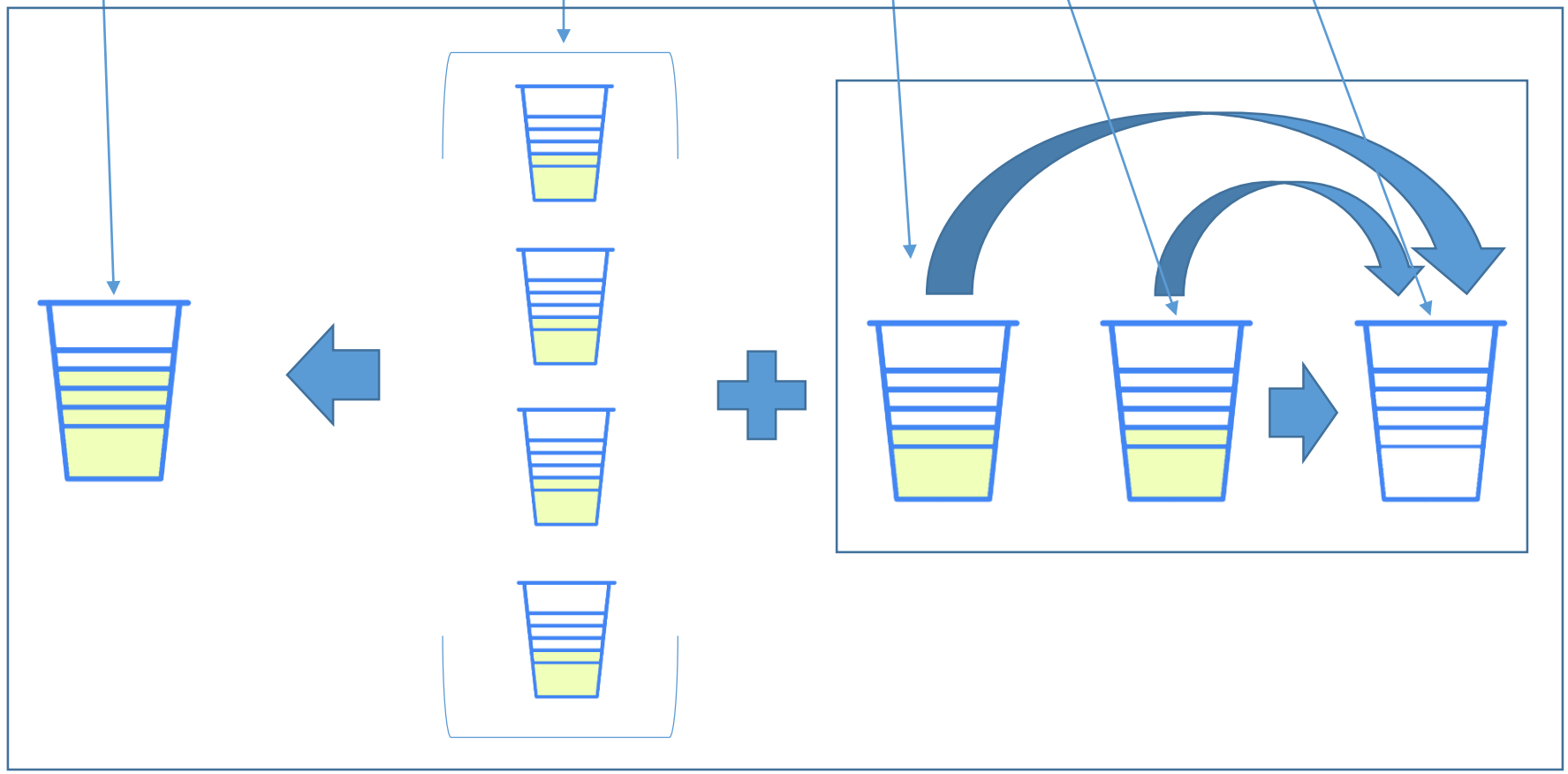
```
val cupsOfJuice: List[GlassOfJuice] = apples.map(apple => juicer.squeezeTheJuice(apple))
```

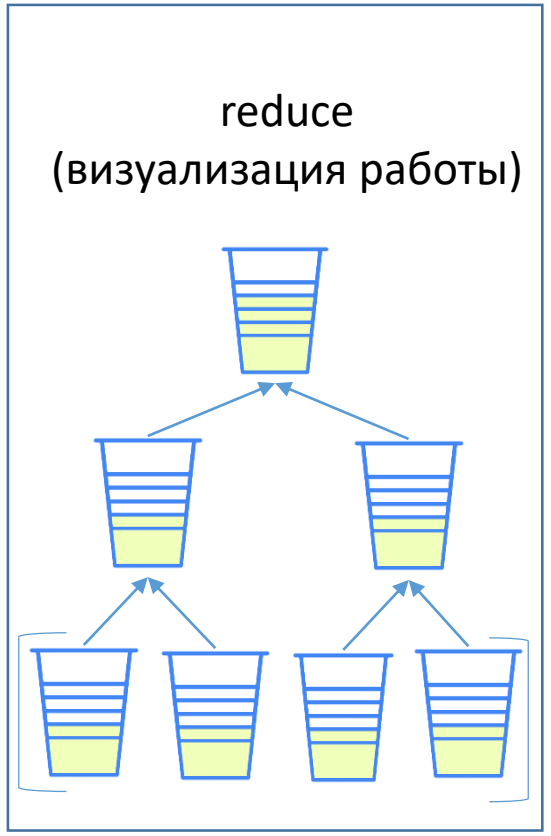
Что у нас хранится в пямяти ?



Как получить из нескольких стаканов сока один?

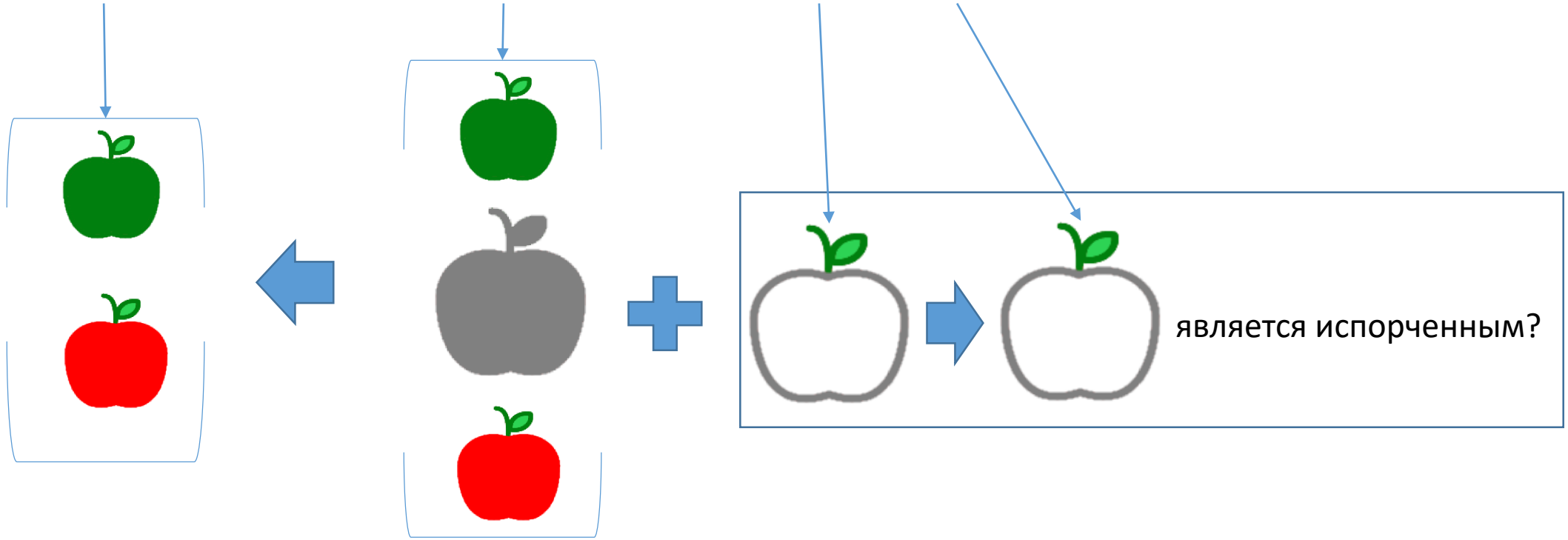
```
val allJuice: GlassOfJuice = cupsOfJuice.reduce((juiceL, juiceR) => GlassOfJuice(juiceL.volume + juiceR.volume))
```





Что делать с испорченными яблоками?

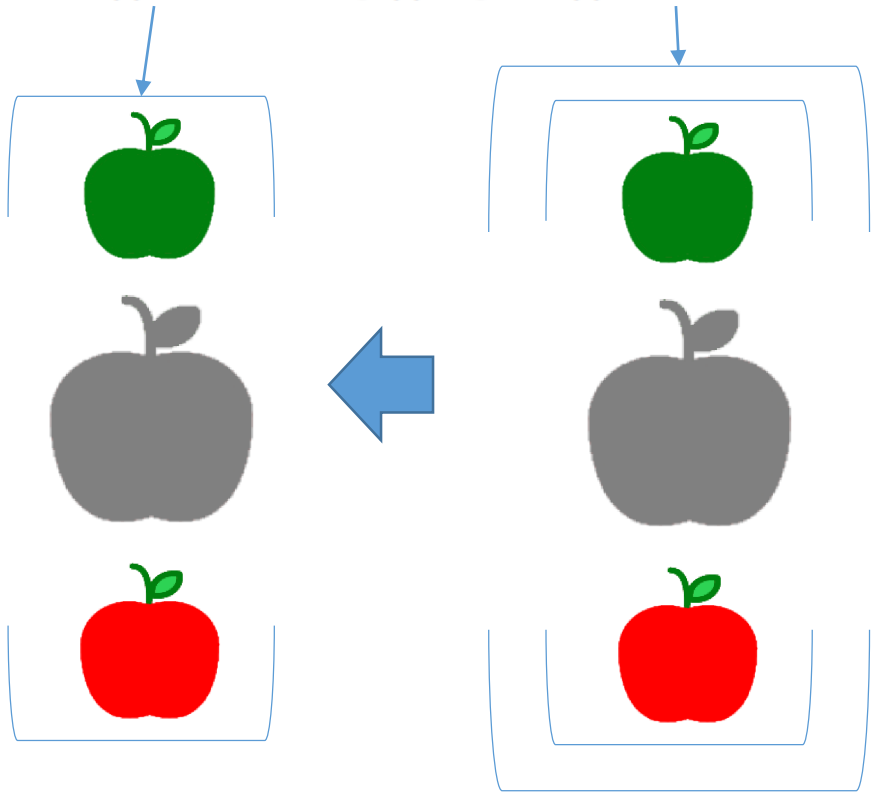
```
val applesWithRotten: List[Apple] = List(Apple("green", 2), Apple("grey", 3), Apple("red", 2))

val applesGood: List[Apple] = applesWithRotten.filter(apple => apple.color != "grey")
```



Что делать с списком в списке?

```
val applesNested1: List[List[Apple]] = List(List(Apple("green", 2), Apple("grey", 3), Apple("red", 2)))
```

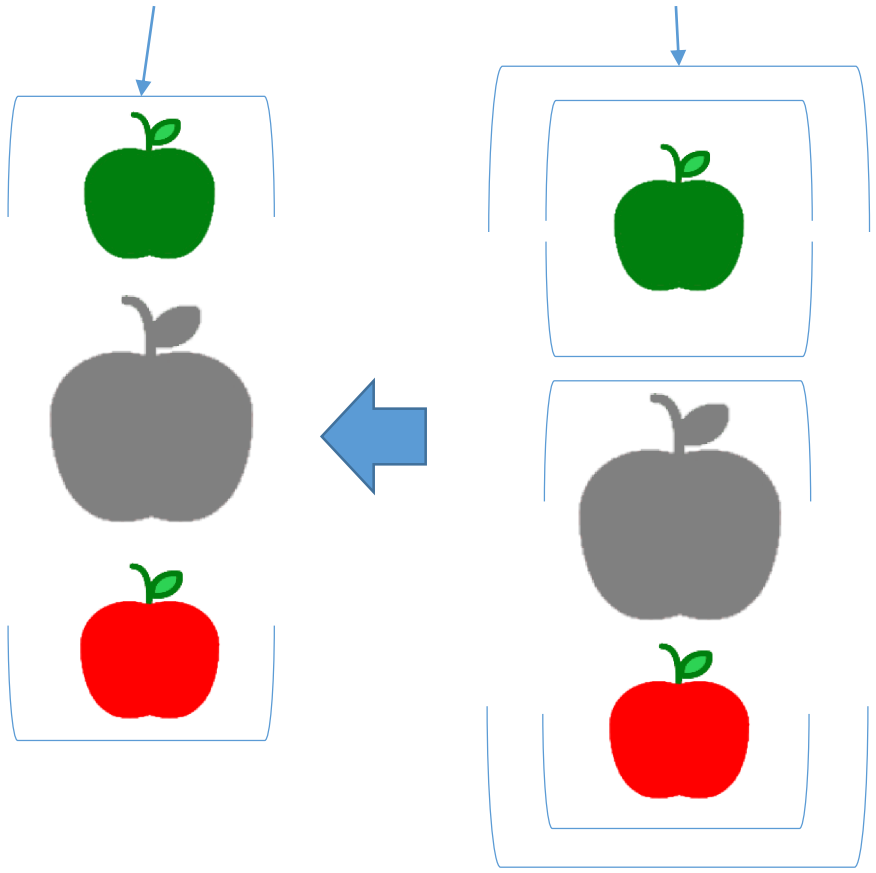
```
val apples1: List[Apple] = applesNested1.flatten
```



Что делать с списком в списке?

```
val applesNested2: List[List[Apple]] = List(List(Apple("green", 2)), List(Apple("grey", 3), Apple("red", 2)))
```

```
val apples2: List[Apple] = applesNested2.flatten
```



Лайфхак : `Flatten + Map = FlatMap`

В циклах (while, for) есть изменяемые части а, что нам может предложить функциональный мир на замену циклов?

Вместо циклов – хвостовая рекурсия

РЕКУРСИЯ

Хвостовая и нехвостовая рекурсия

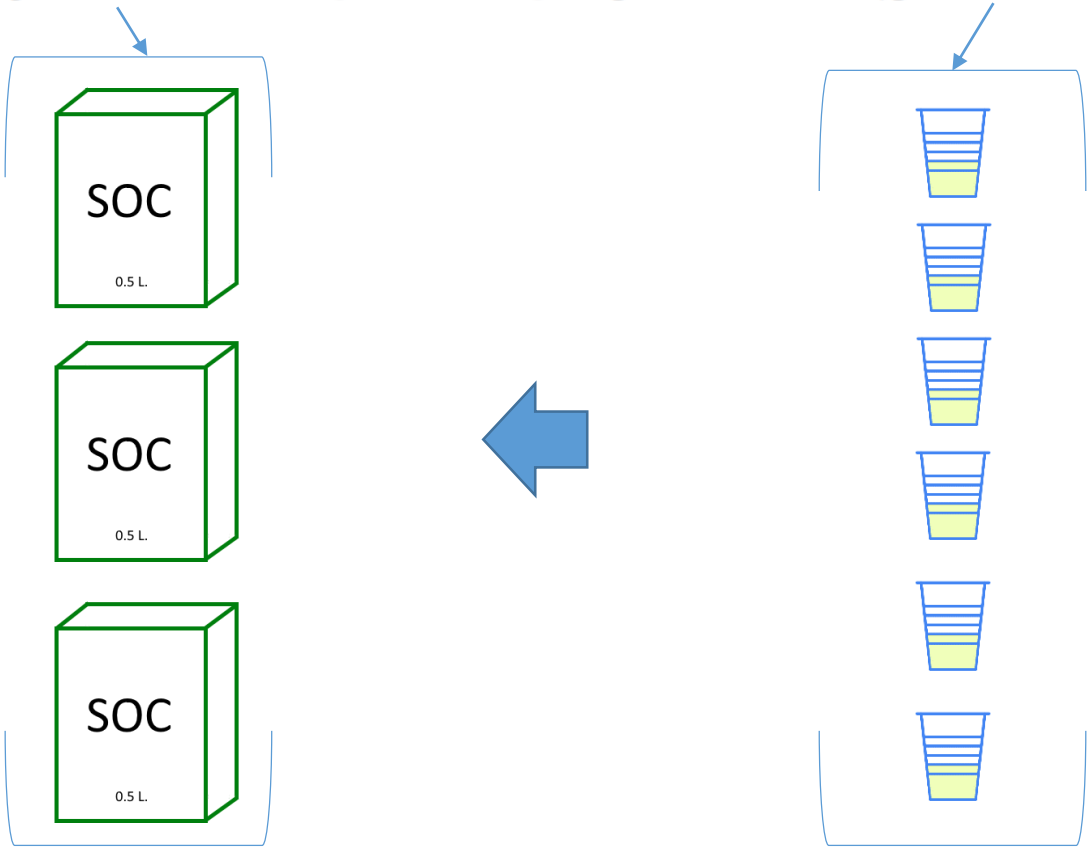
```
def sumNotTailRecursion(list: List[Int]): Int = {
  if (list.isEmpty) 0
  else if (list.size == 1) list.head
  else
  // последняя операция - сумма, сначала нужно
  // найти list.head и sumNotTailRecursion(list.tail), затем их сложить
    list.head + sumNotTailRecursion(list.tail)
}

@tailrec // аннотация - явно говорит коспилятору, что нужно использовать хвостовую рекурсию
def sumTailRecursion(list: List[Int], sumAccumulator: Int): Int = {
  if (list.isEmpty) {
    sumAccumulator
  } else {
    //Список = голова списка (один элемент) + хвост списка
    val sumNow: Int = list.head + sumAccumulator //голова списка + общая сумма
    // последняя операция - вызов функции sumRecursion(tail, sumNow),
    // из за этого это хвостовая рекурсия под "капотом" разворачивается в цикл
    sumTailRecursion(list.tail, sumNow) //list.tail - хвост списка
  }
}

val listNumber: List[Int] = List(1, 2, 4, 5)

val sum1: Int = listNumber.reduce((x, y) => x + y) // 12

val sum2: Int = sumNotTailRecursion(listNumber) // 12

val sum3: Int = sumTailRecursion(listNumber, sumAccumulator = 0) // 12
```

Упаковки сока

```scala
case class JuicePack(volume: Int)

val volumeJuicePacking: Int = 500

@tailrec
def getJuicePacks(cupsOfJuice: List[GlassOfJuice],
                  restVolumeJuice: Int,
                  juicePackAccumulator: List[JuicePack]): List[JuicePack] = {
  // алгоритм максимально упрощенный, поэтому не на всех примерах он будет правильно работать
  if (cupsOfJuice.isEmpty) {
    juicePackAccumulator
  } else {
    val sumRestVolumeJuice: Int = restVolumeJuice + cupsOfJuice.head.volume
    if (sumRestVolumeJuice >= volumeJuicePacking) {
      val restVolumeJuiceNow: Int = sumRestVolumeJuice - volumeJuicePacking
      val juicePackAccumulatorNow: List[JuicePack] = JuicePack(volumeJuicePacking) :: juicePackAccumulator
      getJuicePacks(cupsOfJuice.tail, restVolumeJuiceNow, juicePackAccumulatorNow)
    } else {
      getJuicePacks(cupsOfJuice.tail, sumRestVolumeJuice, juicePackAccumulator)
    }
  }
  // алгоритм максимально упрощенный, поэтому не на всех примерах он будет правильно работать
}

val glassesOfJuice: List[GlassOfJuice] = List(100, 300, 200, 400, 450, 50).map(volume => GlassOfJuice(volume))

val juicePacks: List[JuicePack] = getJuicePacks(glassesOfJuice, restVolumeJuice = 0, List.empty)
```

Упаковки сока

```
val glassesOfJuice: List[GlassOfJuice] = List(100, 300, 200, 400, 450, 50).map(volume => GlassOfJuice(volume))

val juicePacks: List[JuicePack] = getJuicePacks(glassesOfJuice, restVolumeJuice = 0, List.empty)
```



А что, если соковыжималка выжимает сок не мгновенно, а некоторое время

```
// соковыжималка
case class Juicer() {

  def squeezeTheJuice(apple: Apple): GlassOfJuice = {
    Thread.sleep( millis = 4000) // sleep 4 sec
    GlassOfJuice(apple.size * 100)
  }
}
```

Измерение времени выполнения кода

```scala
/**
  * copy from http://biercoff.com/easily-measuring-code-execution-time-in-scala/
  */
def time[R](block: => R): R = {
  val t0 = System.nanoTime()
  val result = block // call-by-name
  val t1 = System.nanoTime()
  println("Elapsed time: " + (t1 - t0) / 1e9 + "sec")
  result
}
```

Класс Future и асинхронный код

```scala
val apples = List(Apple("green", 2), Apple("red", 3), Apple("red", 2), Apple("green", 1))

// В нашем случае Juicer().squeezeTheJuice(apple) работает долго
//Время работы - Elapsed time: 16.006446792sec
time {
  val cupsOfJuice: List[GlassOfJuice] = apples.map(apple => Juicer().squeezeTheJuice(apple))

  val allJuice: GlassOfJuice = cupsOfJuice.reduce((juiceL, juiceR) => GlassOfJuice(juiceL.volume + juiceR.volume))

  println(allJuice)
}

import scala.concurrent.ExecutionContext.Implicits.global // необходим для работы Future

//Время работы - Elapsed time: 4.072226708sec
time {
  val cupsOfJuice: List[Future[GlassOfJuice]] = apples.map(apple => Future(Juicer().squeezeTheJuice(apple)))
  //  каррирование:  reduceLeft(...)(...)
  val allJuiceInFuture: Future[GlassOfJuice] = Future.reduceLeft(cupsOfJuice)((juiceL, juiceR) =>
    GlassOfJuice(juiceL.volume + juiceR.volume))

  // Так делать не нужно, особенно в проде !!! это для примера!!!
  // Получем результат блокируемым образом. Из ассинхронного кода - получаем синхронный.
  val allJuice: GlassOfJuice = Await.result(allJuiceInFuture, Duration.Inf) // Так делать не нужно,
  // особенно в проде !!! это для примера!!!
  println(allJuice)
}
```

В чем сила scala, брат?

Взгляд автора, не обязательно верный

1. Сложная логика простым кодом.
2. Конкурентные, высоконагруженные приложения.
3. Распределенные приложения.
4. ООП + ФП.
5. Перспектива в machine learning.

С чего начать?

1. https://habr.com/ru/company/piter/blog/423317/
2. https://docs.scala-lang.org/ru/tour/tour-of-scala.html
3. SCALA для нетерпеливых, Кей Хорстманн
4. http://groz.github.io/
5. https://stepik.org/course/16243/syllabus
6. https://github.com/anton-k/ru-neophyte-guide-to-scala

Цель презентации: понимание того, как можно решать простые задачи в функциональном стиле на Scala.

Цель достигнута?

А какую **скалу** выберешь ты?

Контакты

Презентация : https://github.com/salamandraa/ScalaForLittle/blob/master/scalaInPictures1.0.pdf

Код : https://github.com/salamandraa/ScalaForLittle

Телеграмм: @sa1amandraa

Почта: salamandraaa@list.ru

Спасибо за внимание!

Использованные ресурсы

- Мемасы
- https://www.autodraw.com/
- https://ru.freepik.com/free-photos-vectors/kid
- https://ru.freepik.com/free-photos-vectors/rock
- https://ru.freepik.com/free-photos-vectors/background

Использованные программы

- Intellij idea
- Powerpoint
- PAINT.NET