



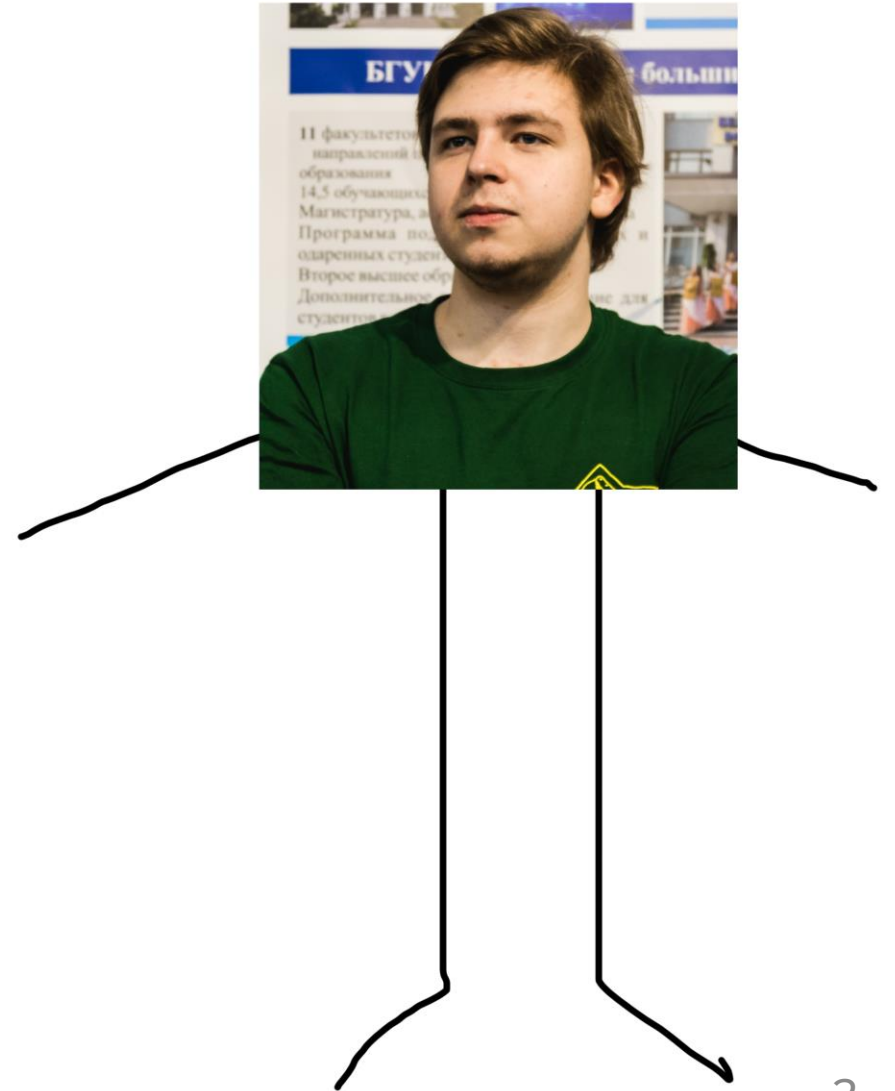
# How to Scala

To Scala or not to Scala? That the  
question...

*W. Shakespeare*

# Вот я был

- Антось Сергеевич
- ВМСиС, 3 курс
- Юзаю Scala 1.5 года с перерывами))0
- Люблю настолочки и пиво
- Я не из BrakhMen





# И вот меня не стало



# Такс-такс, што тут у нас?

Зачем человеку Scala?

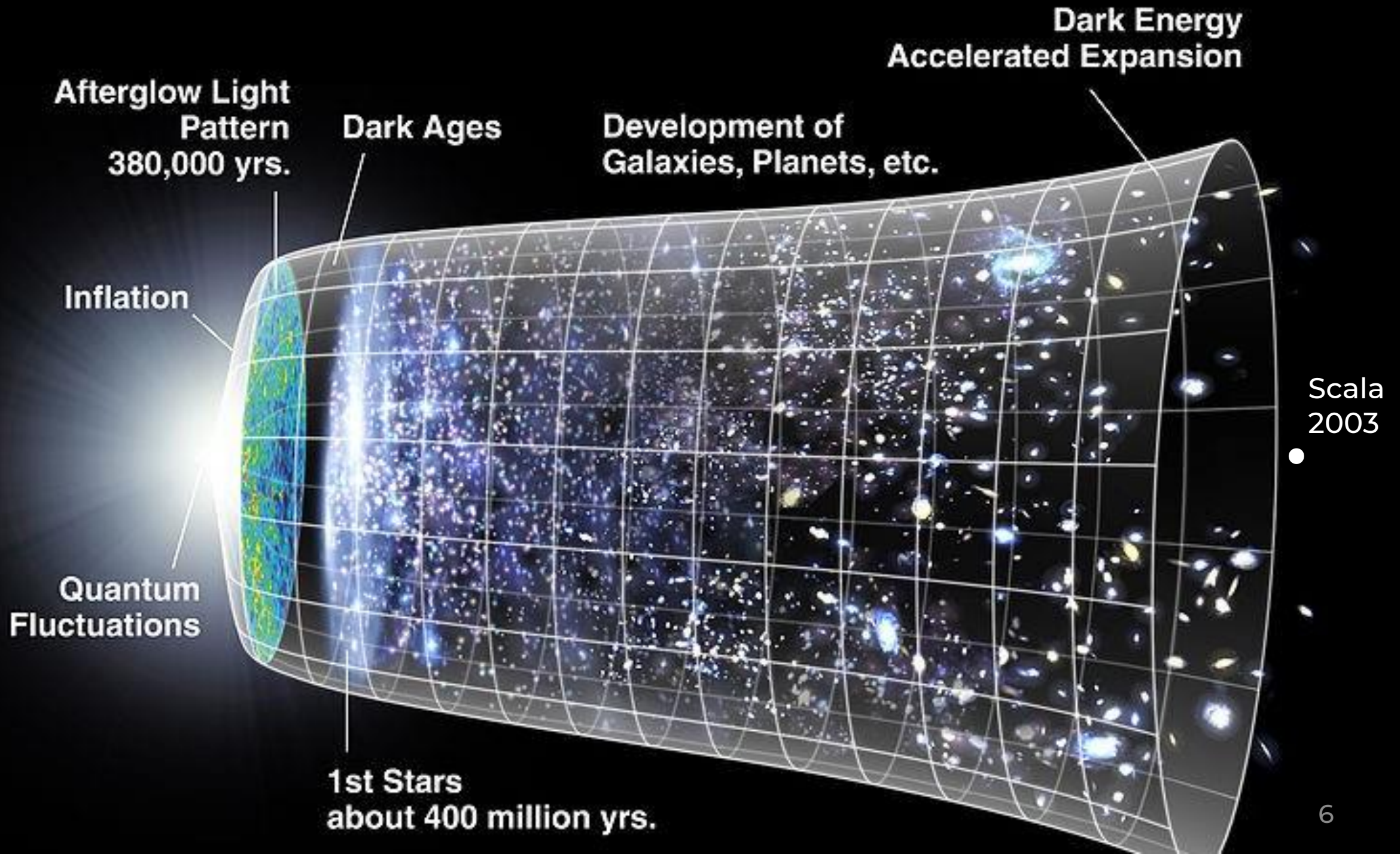
Сводит ли Scala с ума?

Как убедить тим лида использовать  
Scala в реальном проекте?

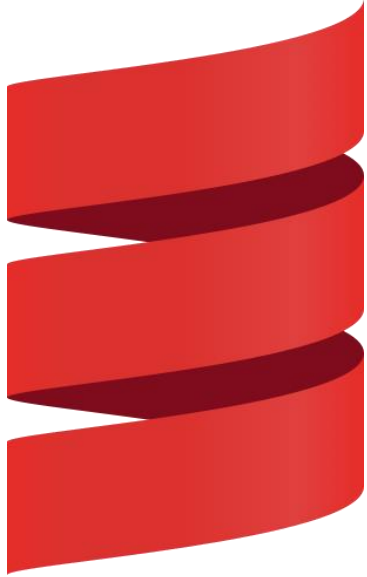


# Планы на ближайший час

- История языка и Scala сегодня
- Философия ФП
- (простите) Синтаксис
- Синтаксический сахарочек
- Scala vs. Kotlin
- Scala среди нас: реальные проекты, переход от Java к Scala



# История Scala



Hi, ich bin Martin  
Odersky und ich habe  
Scala gemacht!





# История Scala

Hi, ich bin Martin  
Odersky und ich habe  
Scala gemacht!





# История Scala




























# Рейтинги: TIOBE

Mar 2019	Mar 2018	Change	Programming Language	Ratings	Change
1	1		Java	14.880%	-0.06%
2	2		C	13.305%	+0.55%
3	4	^	Python	8.262%	+2.39%
4	3	v	C++	8.126%	+1.67%
5	6	^	Visual Basic .NET	6.429%	+2.34%

# Рейтинги: TIOBE

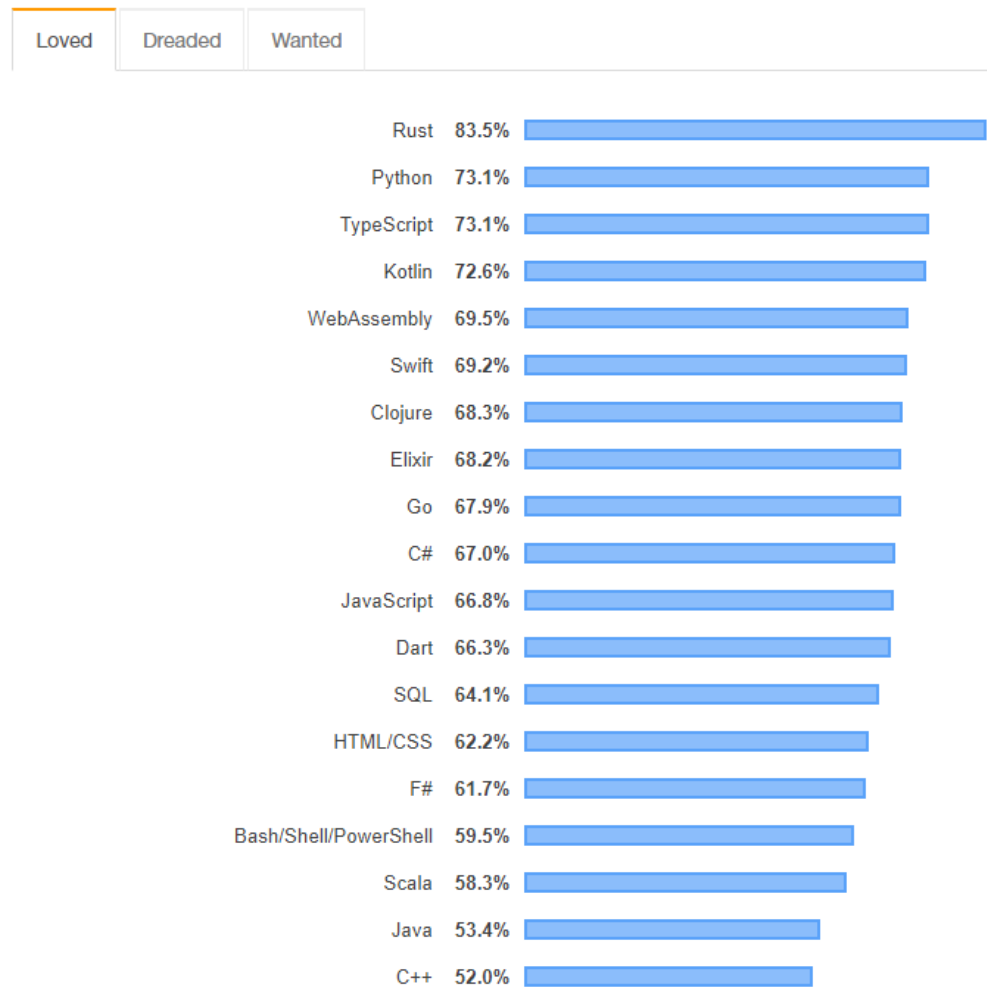
27	Fortran	0.472%
28	Scala	0.467%
29	Lua	0.418%
30	Transact-SQL	0.415%
31	ABAP	0.395%
32	Lisp	0.372%
33	Prolog	0.368%
34	Scheme	0.355%
35	Rust	0.310%
36	Ada	0.310%
37	LabVIEW	0.309%
38	Logo	0.296%
39	Kotlin	0.288%

# Рейтинги: IEEE Spectrum

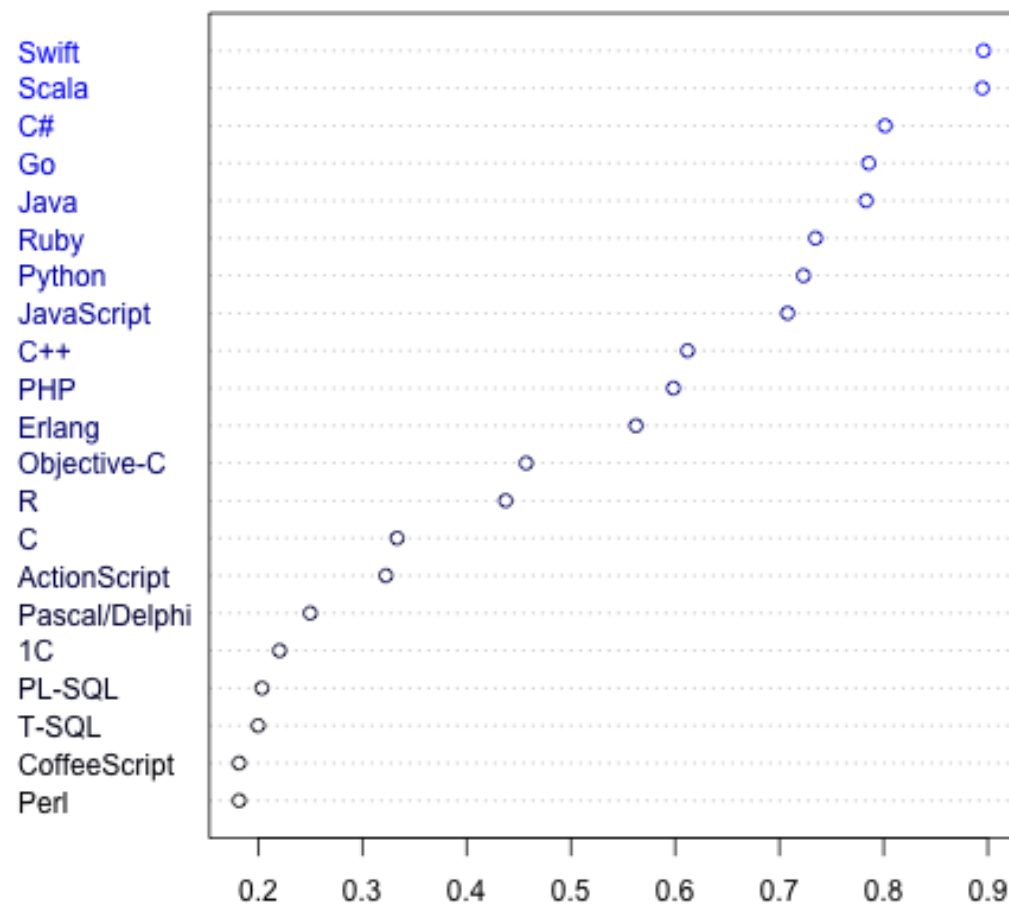
Language Rank	Types	Spectrum Ranking
1. Python	  	100.0
2. C++	  	99.7
3. Java	  	97.5
4. C	  	96.7
5. C#	  	89.4
6. PHP		84.9
7. R		82.9
8. JavaScript	 	82.6
9. Go	 	76.4
10. Assembly		74.1
11. Matlab		72.8
12. Scala	 	72.1

# Рейтинги: Stack Overflow

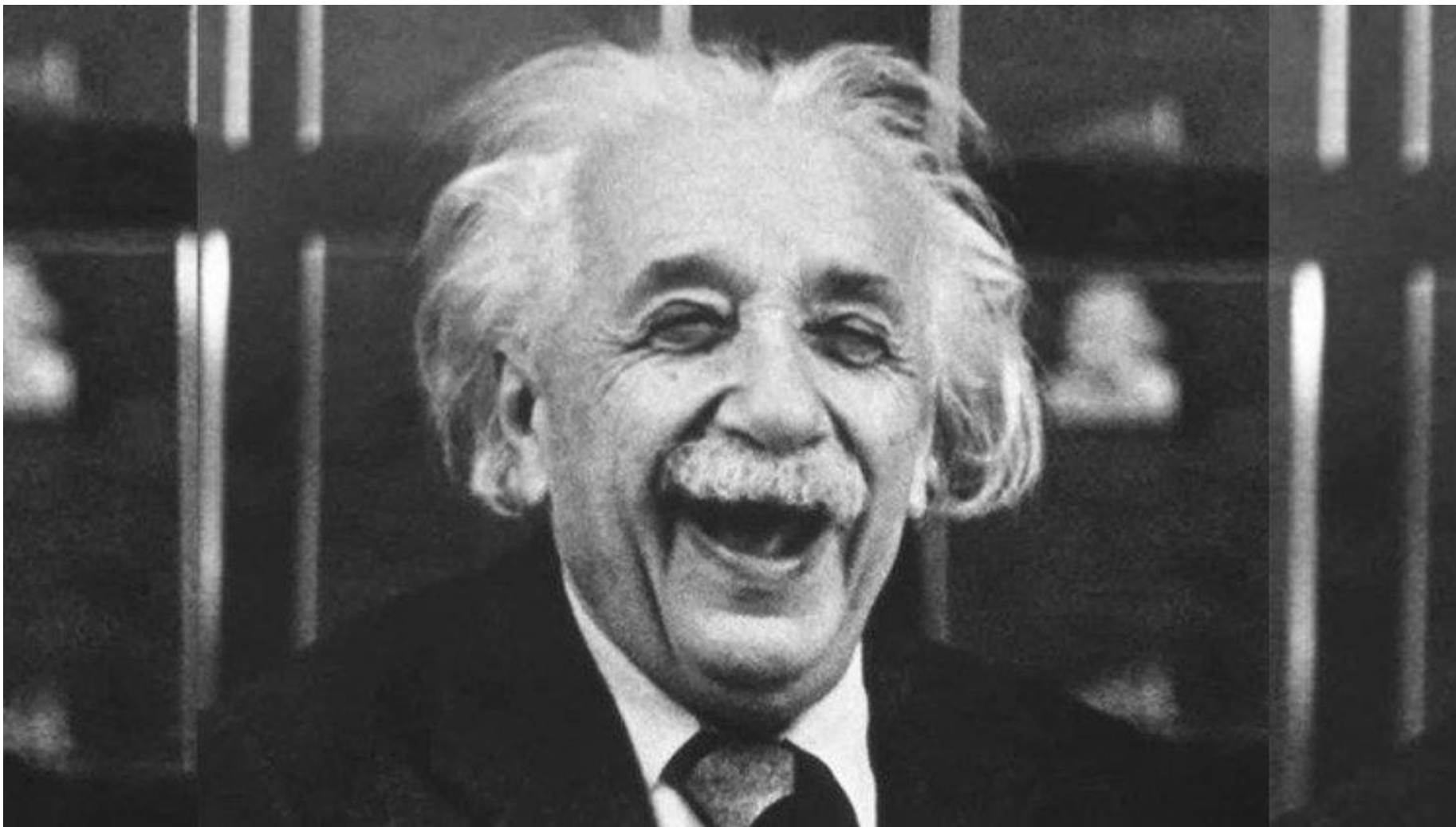
## Most Loved, Dreaded, and Wanted Languages



# Индекс удовлетворённости

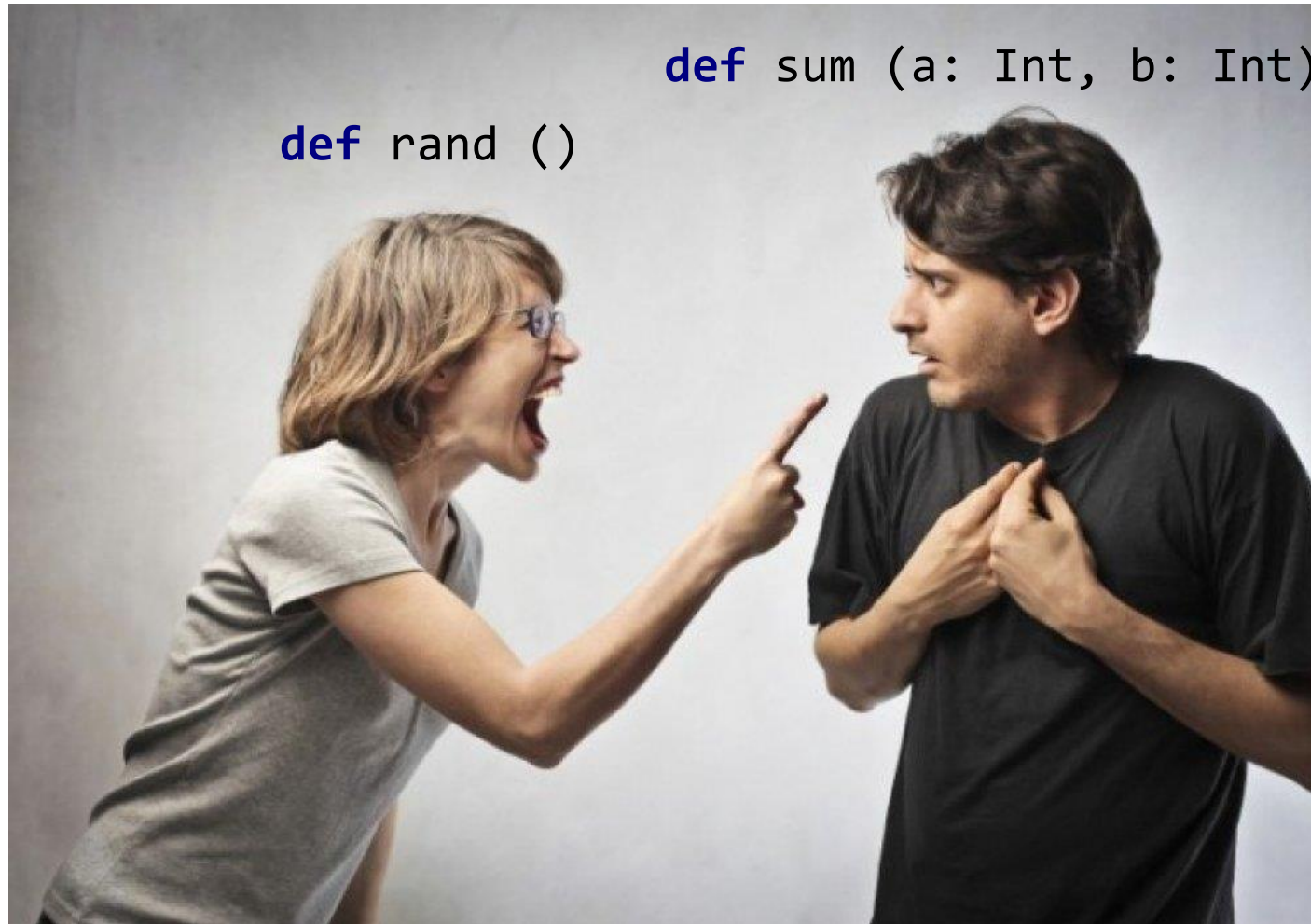


# Немного о ФП





# Все функции - ЧИСТЫЕ





# Все функции – 1 класса

или высшего порядка :)

# Все функции – относительно прозрачны

```
def sumOneAndTwo (): Int = 3
def modifyList(): Unit = {/*Any magic*/}

def main(args: Array[String]): Unit = {
    println(sumOneAndTwo())
    println(3)

    modifyList()
    //nothing?!
}
```

# От variable к value

```
var uni = "BSUIR"
```

```
uni = "BSU" //compiles
```

```
val uni = "BSUIR"
```

```
uni = "BSU" //does not compile
```



# Почему `var` – зло?

- Над изменяемыми объектами труднее думать
- Необходимость синхронизации в многопоточке
- Неочевидные побочные эффекты
- Банально нет необходимости в переменных :)

# Лямбда-исчисление



# Особенности языка





# OOP || FP?



**John @ De Goes**

@jdegoes

Читать



Features Scala has on Haskell:

- first-class modules
- first-class type classes / instances
- OO-style method dispatch
- familiar syntax
- seamless JVM/JS interop
- strict semantics (fits JVM/JS)

Scala could be a nice (if verbose) Haskell alternative.

# OOP || FP?



odersky commented on 21 Mar 2018 • edited ▼

Contributor



This is a proposal to support extension methods and typeclasses in a more direct and convenient way. It is based on [#4114](#).

## Status

This is a first draft proposal (consider it Pre-SIP stage). None of the features that go beyond [#4114](#) are implemented yet.

## Rationale

There are two dominant styles of structuring Scala programs: Standard (object-oriented) class hierarchies or typeclass hierarchies encoded with implicit parameters. Standard class hierarchies lead to simpler code and allow to dispatch on the runtime type, which enables some optimizations that are hard to emulate with implicits. Typeclass hierarchies are more flexible: instances can be given independently of the implementing types and the implemented interfaces, and instances can be made conditional on other typeclass instances.



# OOP && FP!

Scala объектно-ориентированный  
язык

Но, не смотря на это, Scala  
функциональный язык



**Всё – это объект**

Нет, то есть вообще ВСЁ!



# Начиная числами...

$$1 + 2 * 3 / x$$

# Начиная числами...

$$1 + 2 * 3 / x$$

$$(1) + ((2) * (3)) / (x)$$

# ...и заканчивая функциями

```
object Timer {  
  def oncePerSecond(callback: () => Unit): Unit = {  
    while (true) { callback(); Thread sleep 1000 }  
  }  
  def timeFlies(): Unit = {  
    println("Время летит со скоростью 1 секунда в секунду")  
  }  
  
  def main(args: Array[String]): Unit = {  
    oncePerSecond(timeFlies)  
  }  
}
```





# Scala functional basics

1. Изменяемость данных – зло
2. Функции возвращают данные
3. Функции высшего порядка

# Повторение – мать учения

```
var uni = "BSUIR"  
uni = "BSU" //compiles
```

```
val uni = "BSUIR"  
uni = "BSU" //does not compile
```

# Возвращаем данные

```
val result1 = if (true) value1 else value2
val result2 = for (i <- 1 to 10) yield i
def getRandom(): Double = {
    Math.random()
}
```



# Переменная есть определение функции

```
val x = 3
val doubled = (x:Int) => x * 2
println (doubled(x))
```

# Типизация данных

Язык со строгой (сильной) и статической типизацией





# Типизация данных

Горацио говорит: «Но это наша фантазия,  
И не позволит вере завладеть им  
Прикосновение к этому ужасному зрелищу, дважды увиденное нами.  
Поэтому я умолял его,  
С нами, чтобы посмотреть протокол этой ночи,  
Что, если это явление снова придет,  
Он может одобрить наши глаза и поговорить с ним.

Горацио считает это нашей Фантазией, и в жуткое виденье, Представшее нам дважды, он не верит;  
Поэтому его я пригласил  
Посторожить мгновенья этой ночи,  
И, если призрак явится опять,  
Пусть взглянет сам и пусть его окликнет.



# Как Java

Запускается на JVM, компилируется в  
байт-код





# Scala - scalable

```
def factorial(x: BigInt): BigInt =  
  if (x == 0) 1 else x * factorial(x - 1)
```



# «Hello world!»

```
object Main extends App {  
  println("Lorem ipsum")  
}
```



# Toже «Hello world!»

```
object Main {  
    def main(args: Array[String]): Unit = {  
        println("Lorem ipsum")  
    }  
}
```



# Базовое ООП

1. Class
2. Object – реализует паттерн Singleton
3. Case-class
4. Abstract class vs. Trait



# Класс

```
class Group {  
    private val number: Int = 751001  
    val speciality: String = "ПОИТ"  
}
```



# Класс с конструктором

```
class Group (private val number: Int, val speciality: String) {  
    println("Класс сконструирован")  
}
```



# Object

```
object LoneVMSiS {  
  def brazing: Unit = println("Я паяю в одиночестве")  
}
```



# Case-class

Case-class = class + «сахарок» +  
ОПТИМИЗАЦИИ КОМПИЛЯТОРА

```
case class Group(val number: Int)
```





# Case-class

```
val group1 = Group(650503)
println(group1.toString) //Group(650503)
val group2 = Group(650503)
println(group1 == group2) //true
```

# Case-class

```
def groupType(group: Group):String = group match {  
  case Group(650503) => "Лучшая группа ВМСИС"  
  case Group(651005) => "Котики с ПОИТа"  
  case _ => "Какие-то ноунеймы"  
}
```



# Abstract class vs. Trait

Абстрактный класс определяет структуру

Трейт определяет поведение

# Abstract Class

```
abstract class Student(name: String) {  
    def brazing(): Unit = {  
        println("А умею ли я паять?")  
        checkBrazing()  
    }  
    def checkBrazing(): Boolean  
}  
class VMSiS(name: String) extends Student(name) {  
    def checkBrazing(): Boolean = {  
        true  
    }  
}
```

# Trait

```
abstract class Student(name: String) {  
    def brazing(): Unit = {  
        println(«А умею ли я паять?»)   
        checkBrazing()  
    }  
    def checkBrazing()  
}  
  
trait Brazing{  
    def withoutEnthusiasm(msg: String): Unit = println("Ну " + msg)  
    def withEnthusiasm(msg: String): Unit = println("Конечно" + msg + " !!!")  
}  
  
class VMSiS(name: String) extends Student(name) with Brazing {  
    def checkBrazing(): Unit = {  
        withEnthusiasm(msg = "умею")  
    }  
}
```



# Примитивные типы

Примитивных типов нет))0



# Базовые типы

1. Базовые типы == объекты
2. Scala реализует собственные типы, упаковывает и расширяет Java-типы
3. Операторы == методы

Базовый тип	Размер
Byte	8 бит
Short	16 бит
Int	32 бит
Long	64 бит
Char	16 бит, беззнаковый
String	Seq Char
Float	32 бит, single-precision
Double	64 бит, double-precision
Boolean	true или false





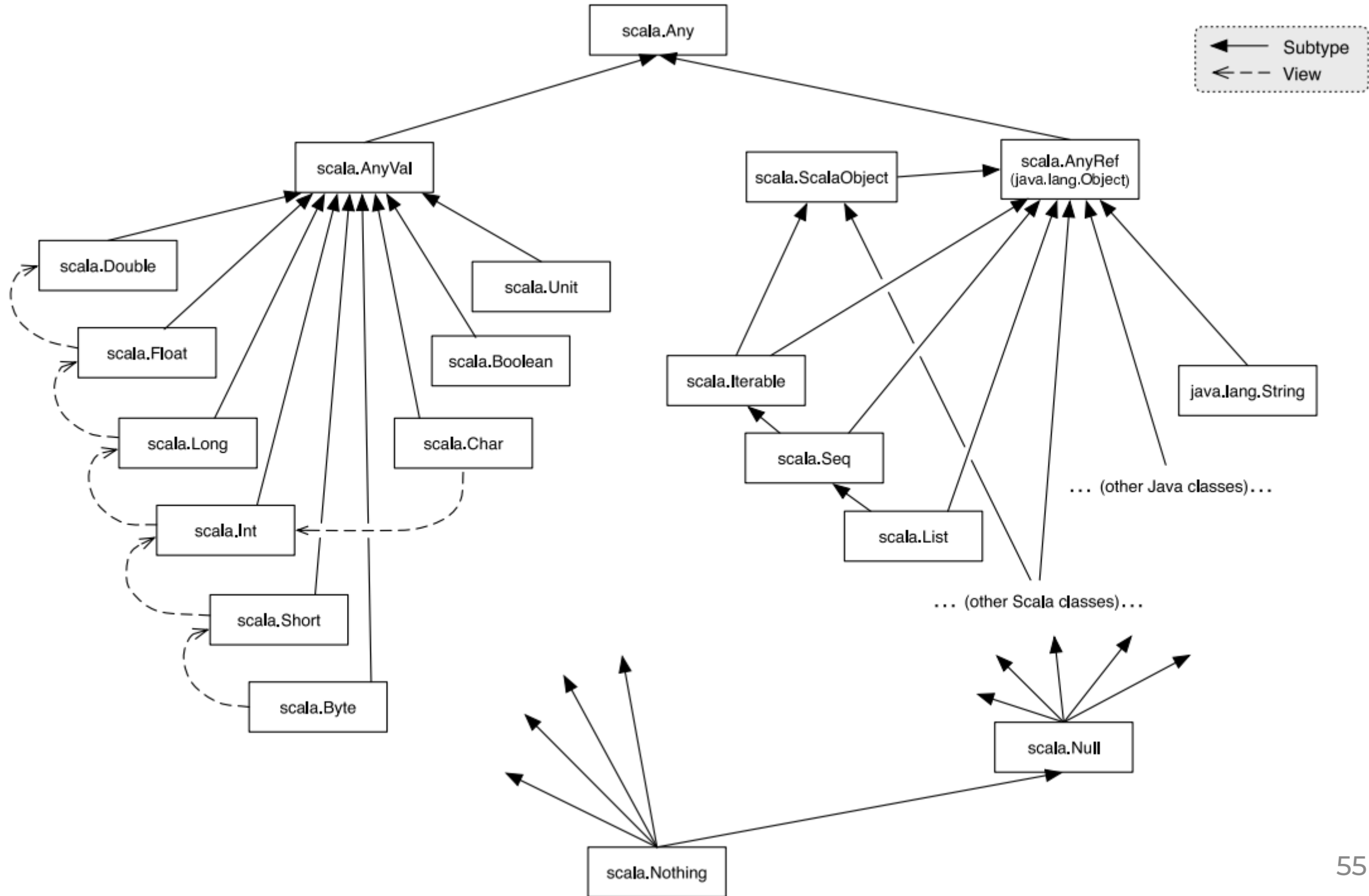
# Операторы == методы

Операторы используют инфиксную нотацию

```
val sum = 1 + 2 // Scala invokes (1).+(2)
```



# Иерархия классов



# Nothing, Null, Nil, None





# Nothing

```
def isTen(number: Int): Boolean =  
    if (10 == number) true  
    else throw new Exception("Number is not ten")
```



# Null

```
def genericIsTen[T](value: T): T =  
    if (10 == value) value  
    else throw new Exception("Generic is not ten")
```



# Nil

```
def listIsTen(number: Int): List[Int] =  
  if (10 == number) List(number)  
  else Nil
```



# None

```
def optionIsTen(number: Int): Option[Int] =  
  if (10 == number) Some(number)  
  else None
```



# Функции и методы

Это не одно и  
тоже



# Автоопределение типа

```
private val age = 20  
val name = "Antosik"
```

```
private val age: Int = 20  
val name: String = "Antosik"
```

# Автоопределение типа

```
def func (a: Int, b: Double) = { a + b } //скомпилится  
def func1 (a, b) = { a + b } //нетушки
```



# Лямбда-выражения

```
val doubler = (n: Int) => n * 2
```

```
val yadobler = { n: Int => n * 2 }
```

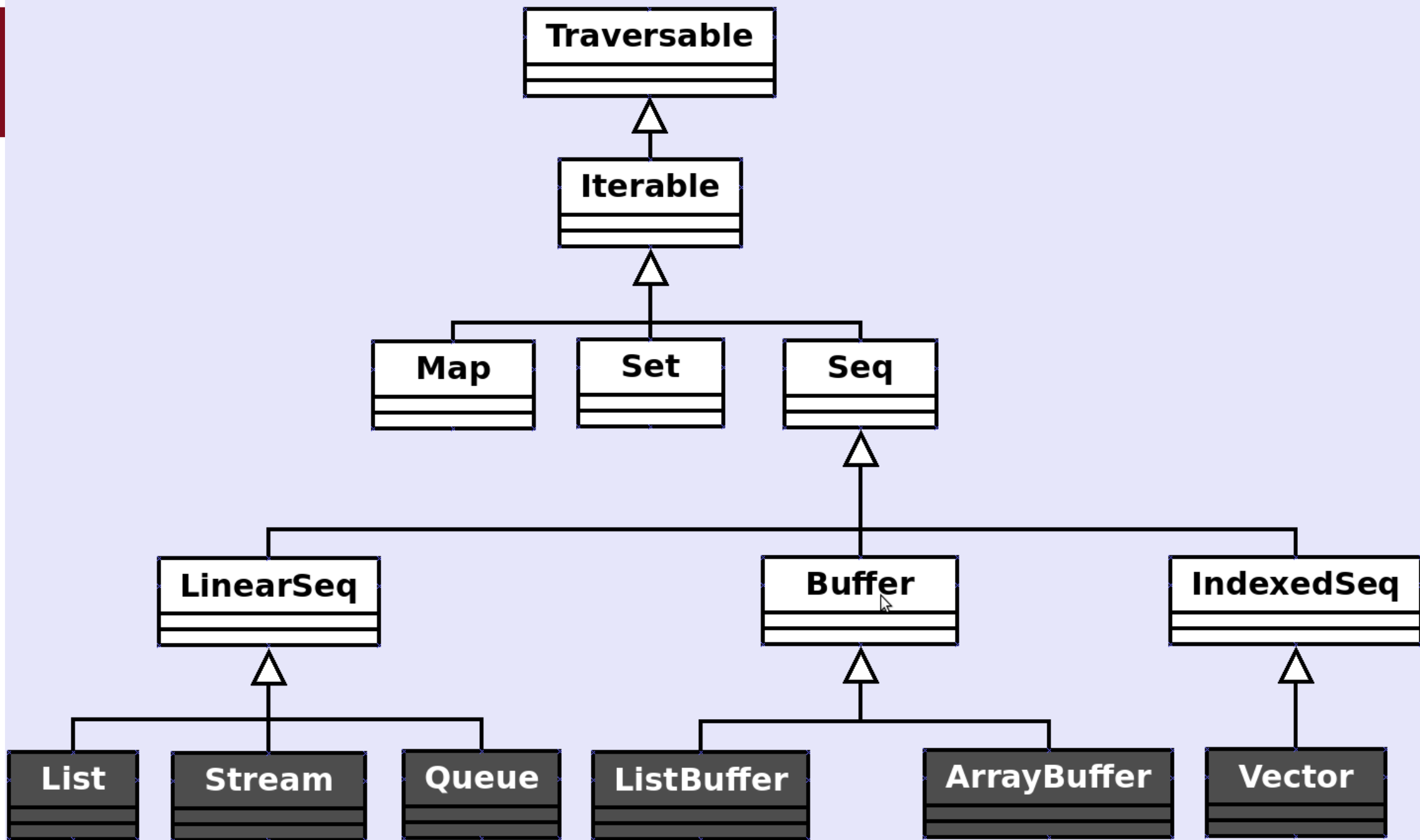
# Коллекции





# Коллекции

```
val map = Map("one" -> 1)  
//results in scala.collection.immutable.Map[String,Int]
```



# Сахар: инициализация Map

Java:

```
Map<String, Integer> mappings = new HashMap<String, Integer>() {{  
    put("One", 1);  
    put("Two", 2);  
    put("Three", 3);  
}};
```

Scala:

```
val mappings = Map(  
    "one" -> 1,  
    "two" -> 2,  
    "three" -> 3  
)
```



# Caxap: Filtering

Java:

```
List<Integer> numbers = new ArrayList<Integer>(){  
    add(1); add(2); add(-55); add(-33); add(122);  
};  
List<Integer> negativeNumbers = new ArrayList<Integer>();  
for (Integer number: numbers) {  
    if (number < 0) {  
        negativeNumbers.add(number);  
    }  
}
```

Scala:

```
val numbers = List(1, 2, -55, -33, 122)  
val negativeNumbers = numbers.filter(_ < 0)
```

# Caxap: Classification

Java:

```
List<Integer> numbers = new ArrayList<Integer>(){  
    add(1); add(2); add(-55); add(-33); add(122);  
};  
List<Integer> negativeNumbers = new ArrayList<Integer>();  
List<Integer> positiveNumbers = new ArrayList<Integer>();  
for (Integer number: numbers) {  
    if (number < 0) {  
        negativeNumbers.add(number);  
    } else {  
        positiveNumbers.add(number);  
    }  
}
```

Scala:

```
val numbers = List(1, 2, -55, -33, 122)  
val (positiveNumbers, negativeNumbers) = numbers.span(_ > 0)
```



# Кортежи

- Простые неизменяемые коллекции
- Могут содержать до 22 элементов различных типов
- Полезны, когда из одной функции необходимо вернуть несколько значений



# Кортежи

```
val pair = (20, "Anton")  
val pair = (20 -> "Anton")  
// the type is Tuple2[Int, String]  
println(pair._1)  
println(pair._2)
```

# Коллекции. Практика

*Map*("one" -> 1) //гуд

HashMap("one" -> 1) //нот соу гуд

**new** HashMap("one" -> 1) //не компилится :)

# Collections API

```
case class User(name: String, password: String)
val users = List("admin:nimda", "user1:asddsa", "root:qwerty")
val mappedUsers = users.map {
  user =>
    val splitted = user.split(":")
    User(splitted(0), splitted(1))
}
// List[User] = List(User(admin,nimda), User(user1,asddsa), User(root,qwerty))
```



# Collections API

```
val names = List(  
    "Alex,Viktor,Eugeny",  
    "Dmitry,Yegor, Sergey",  
    "Michael,Sergey")  
val splitted = names.flatMap(_.split(",").toList).distinct  
// List(Alex, Viktor, Eugeny, Dmitry, Yegor, Michael, Sergey)
```

# Ещё больше сахара!





# Забудьте про equals

Используйте `==` для сравнения всего

```
1 == 1.0 // true
```

```
List(1,2,3) == List(1,2,3) // true
```

```
null == List(1,2,3) // false
```

```
List(1,2) == "string" // false
```

# Забудьте про equals

Оператор `==` делает следующее:

1. Проверяет левую сторону на `null`
2. Если левая сторона `!= null`,  
вызывается `equals` метод

Scala предоставляет возможность для сравнения ссылочного равенства в виде метода `eq`.



# Названия переменных

1. Юзаем UpperCamelCase и lowerCamelCase
2. В названии ТОЛЬКО цифры и буквы, никаких нижних подчёркиваний!
3. Не начинать название с \$ - зарезервировано для переменных компилятора
4. Константы начинаются с большой буквы (math.*Pi*)



# ВЫЗОВ МЕТОДОВ

`Test.method(10)`

`Test.method()`

`Test.method`

# Scala in the Enterprise



*«Она вообще  
используется?»*

# LinkedIn

The screenshot shows a LinkedIn profile for Anton Yurevich. The header includes the LinkedIn logo, a search bar, and navigation links: Главная, Сеть (with 3 notifications), Вакансии, Сообщения, Уведомления, Профиль, and Для работы (with LinkedIn Learning). Language options for English and Русский are available. The profile section features a circular profile picture of a man in a green shirt, a blue banner, and the name Anton Yurevich. His title is 'Student - Belarusian State University of Informatics and Radioelectronics' and his location is 'Беларусь'. There are buttons for 'Добавить раздел' and 'Ещё...'. To the right of the profile, there are links to 'Belarusian State University of Informatics and...', 'См. контактные сведения', and 'См. контакты (193)'. The bio states: 'I study Java, interested in mobile development, IoT and teaching methodology, organize career guidance meetings with high school students, write articles for the newspaper "Impulse" and the portal fksis.bsuir.by'. A GitHub link is shown below the bio. On the right sidebar, there are suggestions for other participants, including Eva Kukar, Kristina Kamendova, Yahor Ptashnik, Anastasiya Sukhotskaya, Наталья Санюк (Лапицкая), and Stsiapan Balashenka.

in Поиск

Главная Сеть Вакансии Сообщения Уведомления Профиль Для работы LinkedIn Learning

English Русский

Редактировать общедоступный профиль и URL-адрес

Другие участники также смотрели

**Eva Kukar** • 1-й  
заместитель декана факультета компьютерных систем и сетей БГУИР

**Kristina Kamendova** • 2-й  
Software Developer - Smart IT

**Yahor Ptashnik** • 1-й  
Student at Belarusian State University of Informatics and Radioelectronics

**Anastasiya Sukhotskaya** • 1-й  
Data Analyst, Учащийся - Belarusian State University of Informatics and Radioelectronics

**Наталья Санюк (Лапицкая)** • ...  
заведующий кафедрой программного обеспечения информационных технологий (БГУИР)

**Stsiapan Balashenka** • 1-й  
Student of Belarusian State University of Informatics and Radioelectronics

**Anton Yurevich**  
Student - Belarusian State University of Informatics and Radioelectronics  
Беларусь

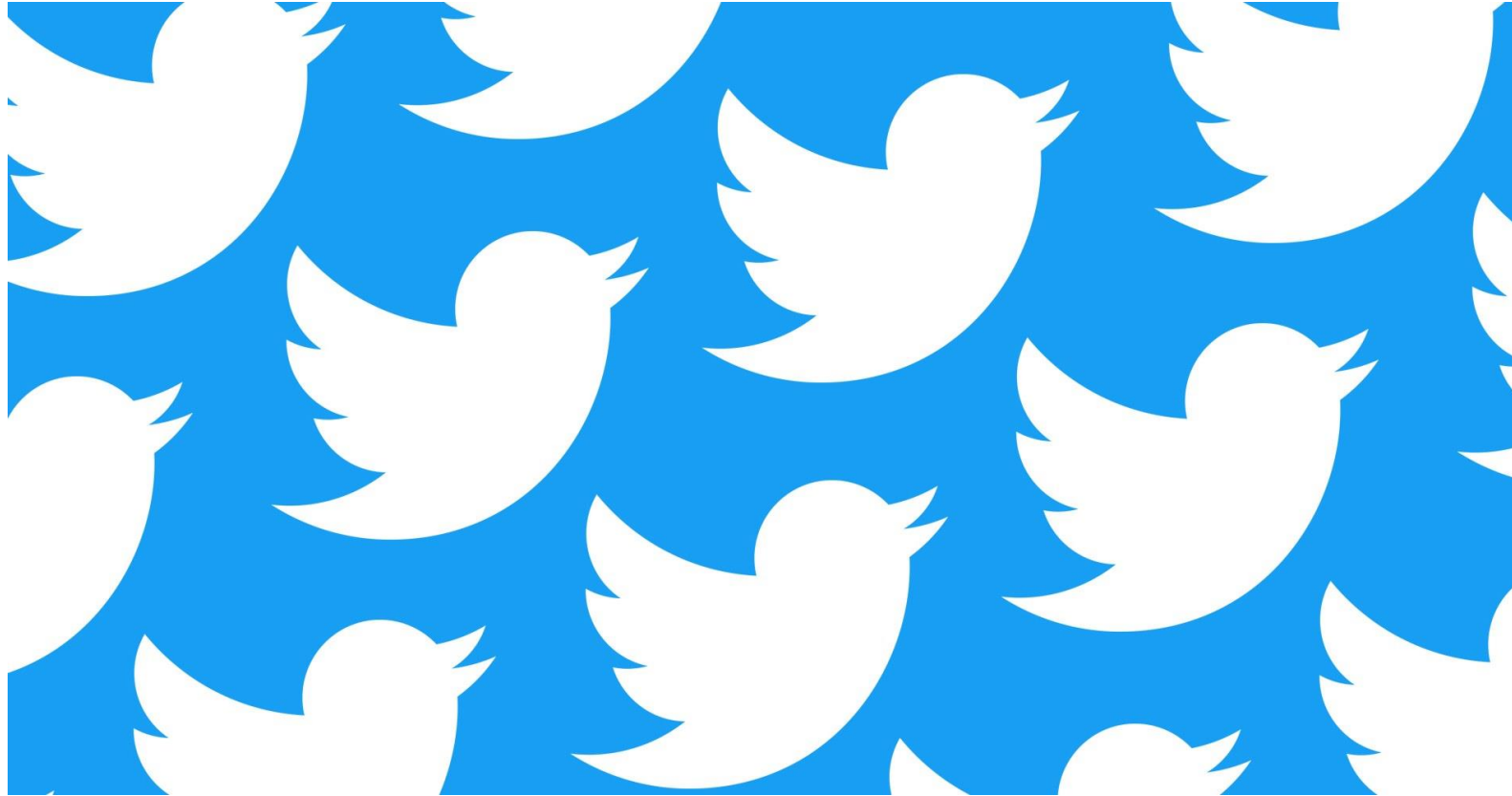
Добавить раздел Ещё...

I study Java, interested in mobile development, IoT and teaching methodology, organize career guidance meetings with high school students, write articles for the newspaper "Impulse" and the portal fksis.bsuir.by

GitHub

<https://www.linkedin.com/in/bulbash3r/>

# Twitter



[https://www.artima.com/scalazine/articles/twitter\\_on\\_scala.html](https://www.artima.com/scalazine/articles/twitter_on_scala.html)



# Coursera



<https://medium.com/coursera-engineering/why-we-love-scala-at-coursera-80fa1fc66d74>





И другие

xerox

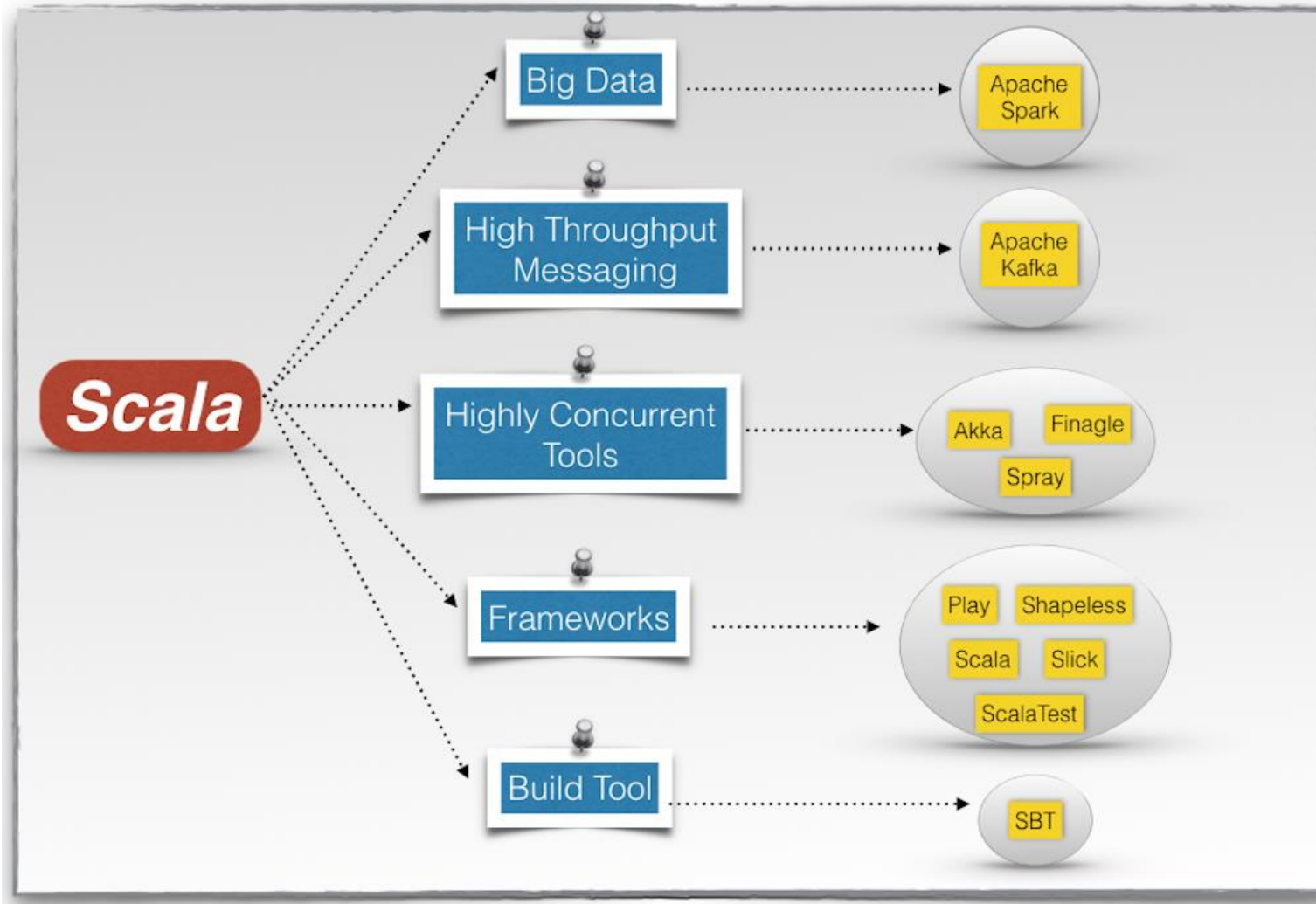


The  
Guardian

SIEMENS

SONY

# Где используется?





# **N причин для изучения**

## **1. Назад дороги нет**



**N причин для изучения**

**2. Scala современнее**



# N причин для изучения

`object1 == object2`

`object1.equals(object2)`



**N причин для изучения**

**3. Меньше кода**



# N причин для изучения

```
String mama = "Таня";  
final String papa = "Саня";
```



# N причин для изучения

```
var мама = "Таня";  
val папа = "Саня";
```





# N причин для изучения

```
public class Child {  
    final String mama;  
    final String papa;  
    public Child(String mama, String papa) {  
        this.mama = mama;  
        this.papa = papa;  
    }  
}
```



# N причин для изучения

```
class Child(val mama: String, val papa: String)
```

# N причин для изучения

4. Это легко  
(ну, по  
крайней мере,  
не трудно)





**N причин для изучения**

**5. Новые горизонты,  
интерес, бла-бла-бла**

# Scala vs. Kotlin





# Неужели всё так хорошо?



# Неужели всё так хорошо?

```
final override def flatMap[B, That](f: A => GenTraversableOnce[B])(implicit bf: CanBuildFrom[List[A], B, That]): That = {
  if (bf eq List.ReusableCBF) {
    if (this eq Nil) Nil.asInstanceOf[That] else {
      var rest = this
      var found = false
      var h: ::[B] = null
      var t: ::[B] = null
      while (rest ne Nil) {
        f(rest.head).seq.foreach{ b =>
          if (!found) {
            h = new ::(b, Nil)
            t = h
            found = true
          }
          else {
            val nx = new ::(b, Nil)
            t.tl = nx
            t = nx
          }
        }
        rest = rest.tail
      }
      (if (!found) Nil else h).asInstanceOf[That]
    }
  }
  else super.flatMap(f)
}
```



# Let's try!

Twitter Scala School [[ТЫК!](#)]

Принципы функционального программирования на Scala  
@Coursera [[ТЫК!](#)]

Scala-lang [[ТЫК!](#)]

Мой Google Drive с книгами [[ТЫК!](#)]