Miniboxing

the functional side of it :)

BJUG, 17th of July 2014

Vlad Ureche
vlad.ureche@epfl.ch

Motivation

WE ARE HERE

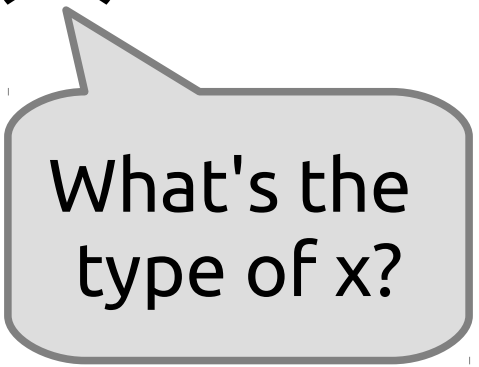
Specialization

Miniboxing

Functions

Evaluation

```
def identity(x) = x
```



```
def identity(x: Int): Int = x

identity(1)                    // okay
identity("three")              // error
```

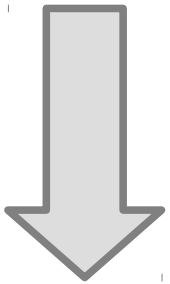
```
def identity(x: Any): Any = x

identity(1)                    // okay
identity("three")              // okay
identity(1) + identity(3)      // error
```

No + operation
on values of type Any

Generics to our rescue!

```
def identity[T](x: T): T = x

identity(1)                    // okay
identity("three")              // okay
identity(1) + identity(3)      // okay
```

```
def identity[T](x: T): T = x
```



scalac / javac
(erasure)

```
def identity(x: Any): Any = x
```

Passed by reference
(java.lang.Object)

scala.Int

int

- fast access
- no garbage collection
- locality

java.lang.Integer

- indirect access
- garbage collection
  - and object allocation
- no locality guarantees
- **compatible with generics**

No-go for any performance-oriented application

Motivation

Specialization

WE ARE HERE

Miniboxing

Functions

Evaluation

```
def identity[T](x: T): T = x
```

For reference types (e.g. String)

```
def identity_V(x: Unit): Unit = x
def identity_Z(x: Boolean): Boolean = x
def identity_B(x: Byte): Byte = x
def identity_C(x: Char): Char = x
def identity_S(x: Short): Short = x
def identity_I(x: Int): Int = x
def identity_L(x: Long): Long = x
def identity_F(x: Float): Float = x
def identity_D(x: Double): Double = x
```

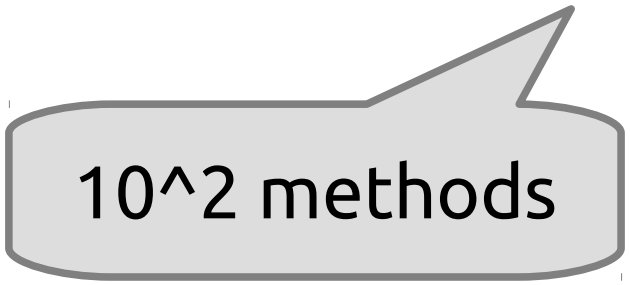
For primitive types

```
def identity[T](x: T): T = x
def identity_V(x: Unit): Unit = x
def identity_Z(x: Boolean): Boolean = x
def identity_B(x: Byte): Byte = x
def identity_C(x: Char): Char = x
def identity_S(x: Short): Short = x
def identity_I(x: Int): Int = x
def identity_L(x: Long): Long = x
def identity_F(x: Float): Float = x
def identity_D(x: Double): Double = x
```

```
identity(1)                 => identity_I(1)
identity("three")           => identity[String]("three")
identity(1) + identity(3)   => identity_I(1) +
                               identity_I(3)
```

This is specialization
(Iulian Dragos, 2009)

```
def tupled[T1, T2](t1: T1, t2: T2) = ...
```



Motivation

Specialization

Miniboxing

WE ARE HERE

Functions

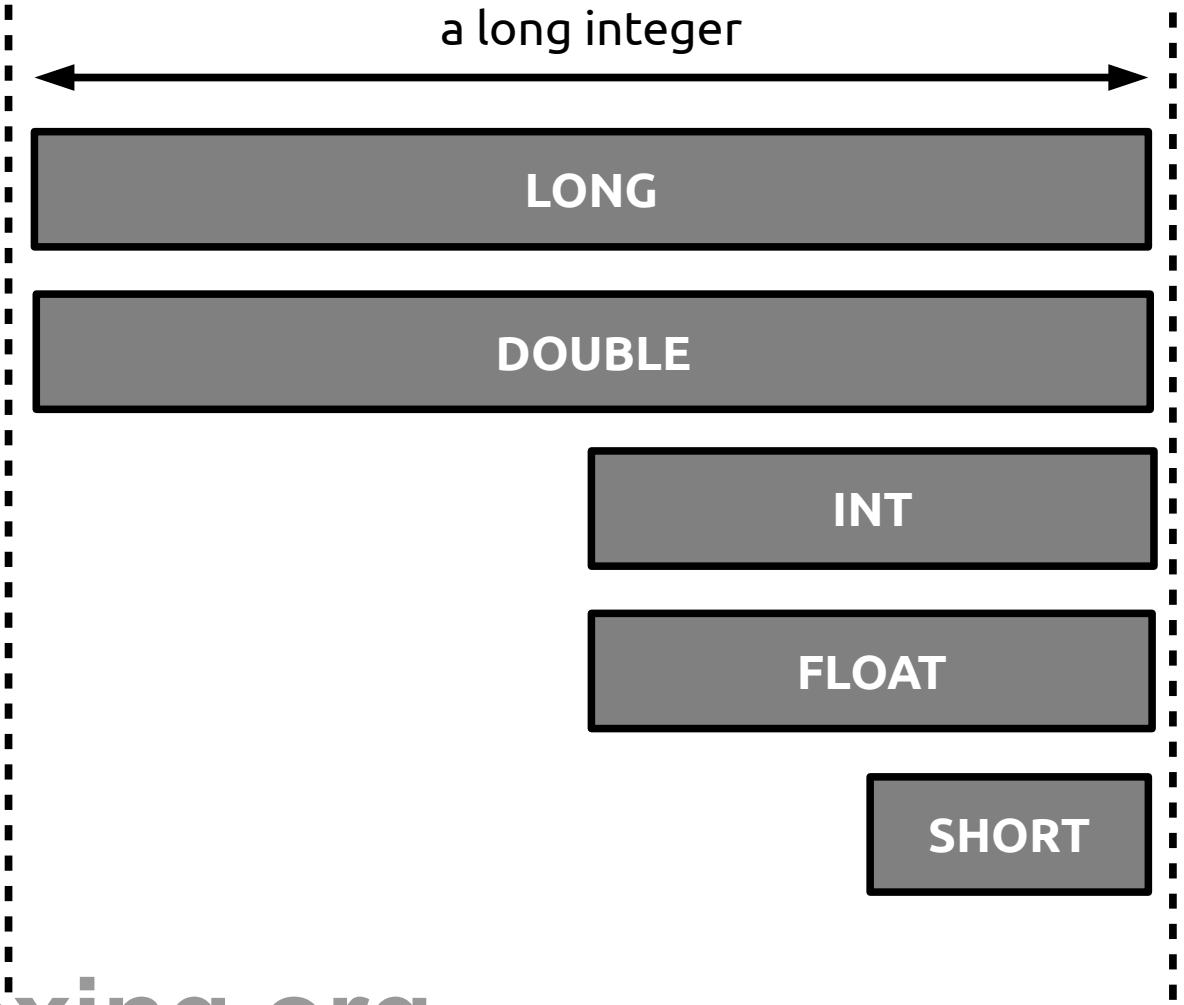
Evaluation

```
def identity[T](x: T): T = x

def identity_V(x: Unit): Unit = x
def identity_Z(x: Boolean): Boolean = x
def identity_B(x: Byte): Byte = x
def identity_C(x: Char): Char = x
def identity_S(x: Short): Short = x
def identity_I(x: Int): Int = x
def identity_L(x: Long): Long = x
def identity_F(x: Float): Float = x
def identity_D(x: Double): Double = x
```

But...

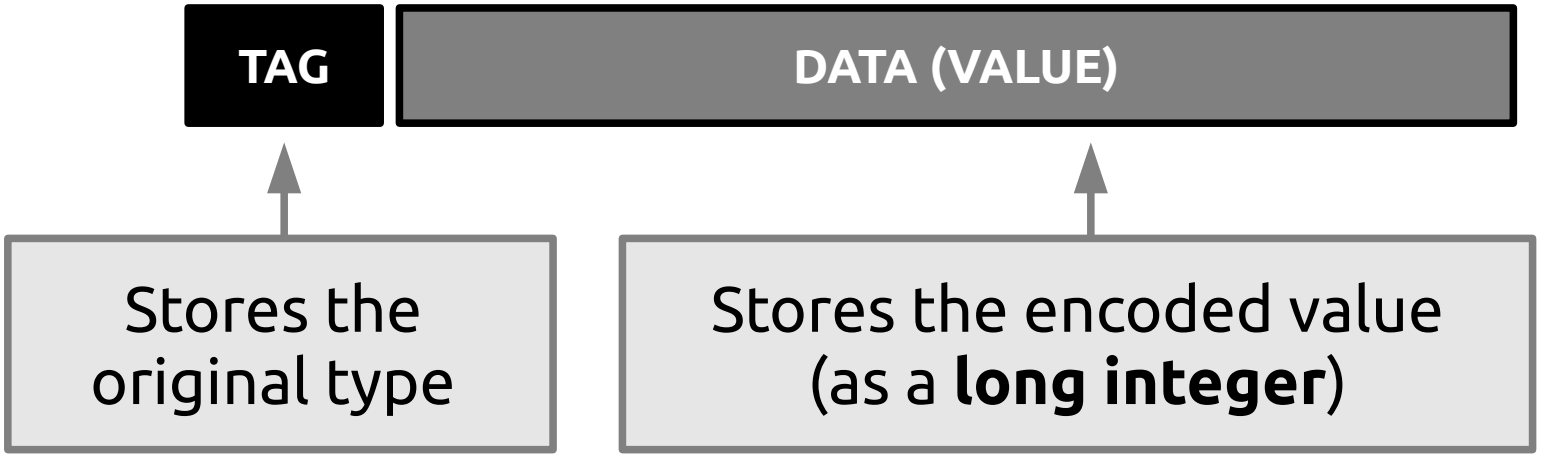
we can do better

One day in 2012 Miguel Garcia walked into my office and said: *"From a low-level perspective, there are only values and pointers. Maybe you can use that!"*



And then the idea was born

it started from the **tagged union**



And then the idea was born

it started from the **tagged union**

	TAG	DATA (VALUE)
`true` =	BOOL	0x1
`42` =	INT	0x2A
`5.0f` =	FLOAT	bit representation

And then the idea was born

it started from the **tagged union**

- somewhat similar to a boxed object
- but not in the heap memory
- direct access to the value

Same benefits as for unboxed values

Let's take

an example


```
def choice[@miniboxed T](t1: T,
                         t2: T): T =
  if (util.Random.nextBoolean())
    t1
  else
    t2
```

We'll have a version for primitive types

Let's take

an example


```
def choice_J[T](t1: ...,
                t2: ...): ... =
  if (util.Random.nextBoolean())
    t1
  else
    t2
```

But what's the signature?

Let's take an example

That's wasteful: we carry the tag for T twice

```
def choice_J[T](t1: (Tag,Value),
                t2: (Tag,Value)):(Tag,Value)=
  if (util.Random.nextBoolean())
    t1
  else
    t2
```

And we even return it, despite the caller having passed it

Insight: we're in a statically typed language, **use that**!

Let's take an example

Sort of a class tag

T_Tag corresponds to the type parameter

```
def choice_J[T](T_Tag: Tag, t1: Value,
                                t2: Value):Value=
  if (util.Random.nextBoolean())
    t1
  else
    t2
```

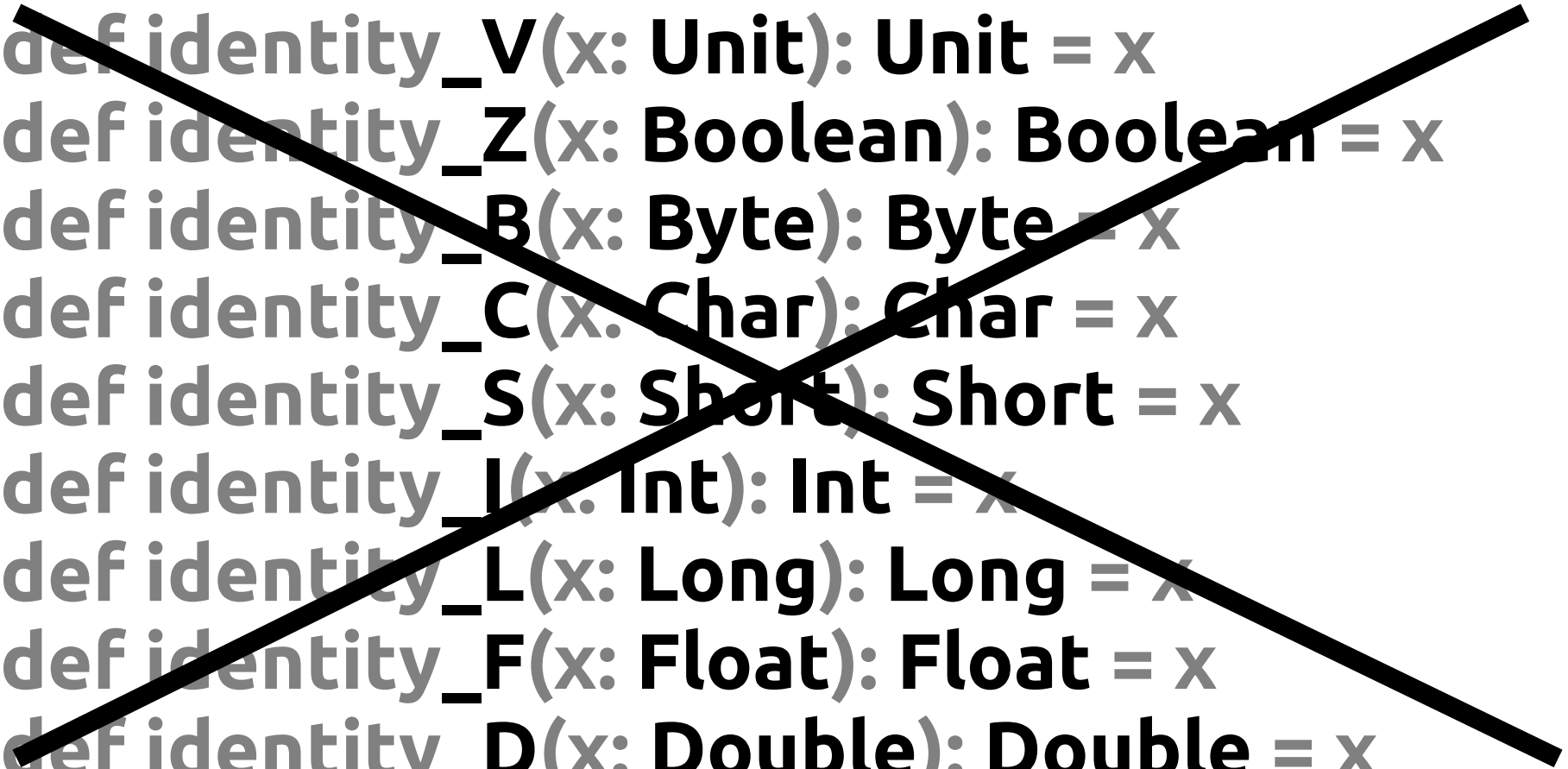
Encoded as **Long**

Let's take

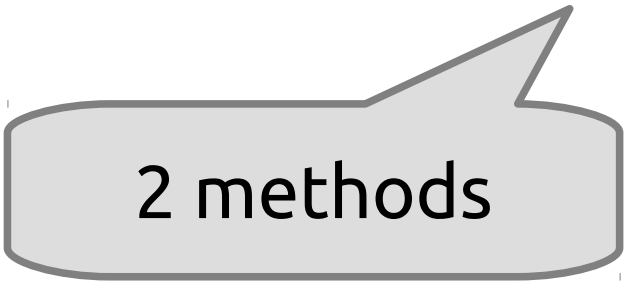
an example


```
def choice_J[T](T_Tag: Byte, t1: Long,
                                t2: Long): Long=
  if (util.Random.nextBoolean())
    t1
  else
    t2
```

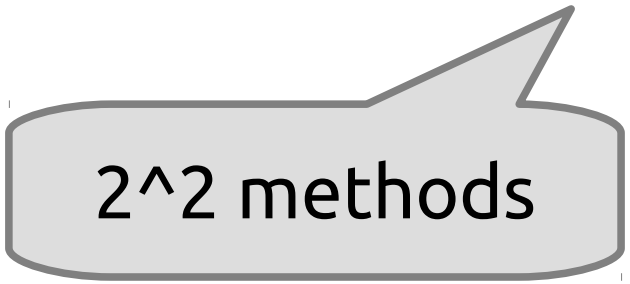
```
def identity[T](x: T): T = x

def identity_V(x: Unit): Unit = x
def identity_Z(x: Boolean): Boolean = x
def identity_B(x: Byte): Byte = x
def identity_C(x: Char): Char = x
def identity_S(x: Short): Short = x
def identity_I(x: Int): Int = x
def identity_L(x: Long): Long = x
def identity_F(x: Float): Float = x
def identity_D(x: Double): Double = x
```



```
def identity[T](x: T): T = x
def identity[T](T_Tag: Byte, x: Long): Long = x
```



```
def tupled[T1, T2](t1: T1, t2: T2) = ...
```



Motivation

Specialization

Miniboxing

Functions

WE ARE HERE

Evaluation

Scala

is object-oriented and functional

… but first of all, object-oriented!

Functions

are also objects

```
trait Function1[-T, +R] {
  def apply(t: T): R
  ...
}
```

Functions

are also objects

```
val f = (x: Int) => x + 1
```

```
val f = {
  class $anon extends Function1[Int, Int] {
    def apply(x: Int): Int = x + 1
  }
  new $anon()
}
```

Functions

are also objects

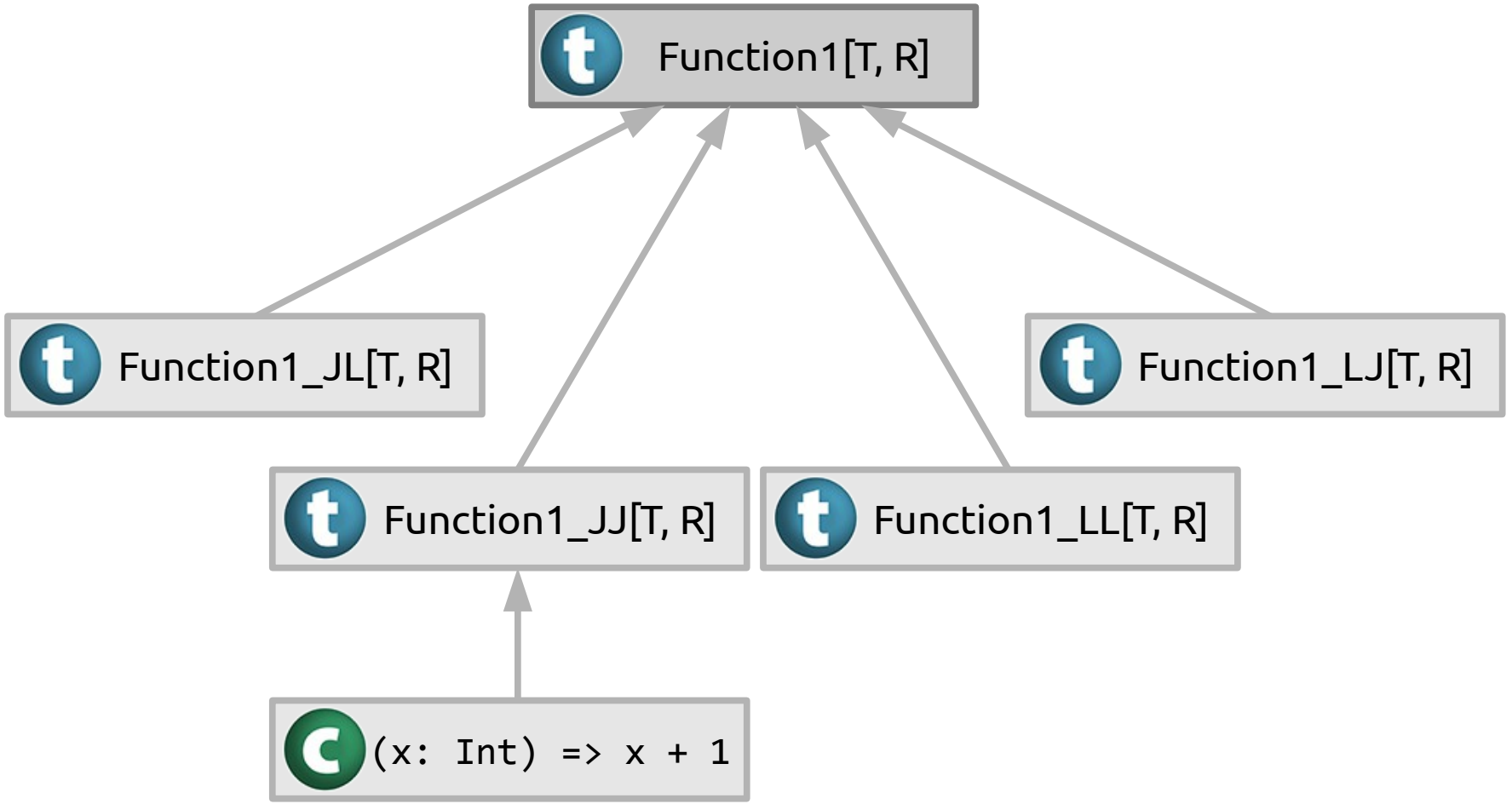
```
f(4)
```

```
f.apply(4)
```

with miniboxing...

```
trait Function1[-T, +R]
```

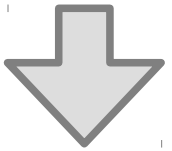
```
(x: Int) => x + 1
```



Functions

are also objects


```
val f = (x: Int) => x + 1
```



Specialized

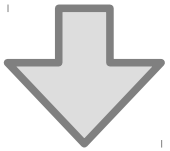
```
val f = {
  class $anon extends Function1_JJ... {
    def apply_JJ(..., x: Long): Long = ...
  }
  new $anon()
}
```

Specialized

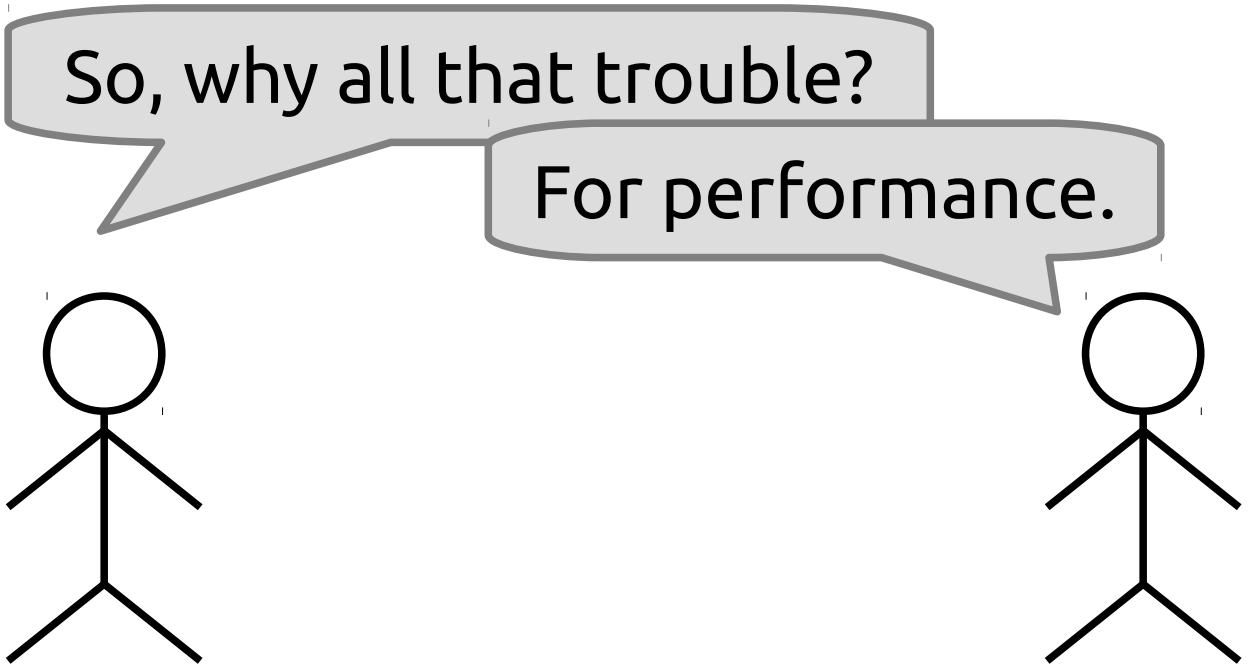
Functions

are also objects


```
f(4)
```



```
f.apply_JJ(INT, INT, int2minibox(4))
```


So, why all that trouble?
For performance.

Motivation

Specialization

Miniboxing

Functions

Evaluation

WE ARE HERE

Evaluation

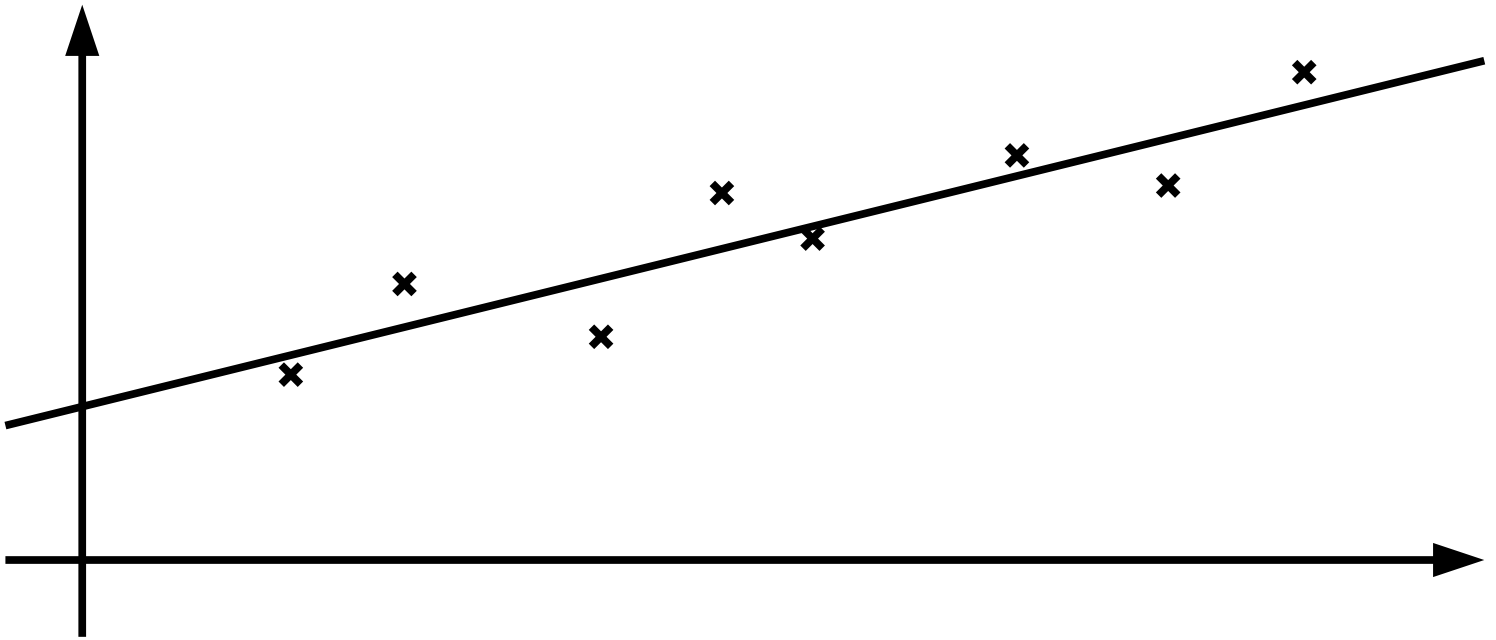
on the Scala linked list

- work with Aymeric Genet (github: @MelodyLucid)
- mock-up of Scala linked list
  - Function1 / Function2 / Tuple2
  - Traversable / TraversableLike
  - Iterator / Iterable / IterableLike
  - LinearSeqOptimized
  - Builder / CanBuildFrom

Benchmarks

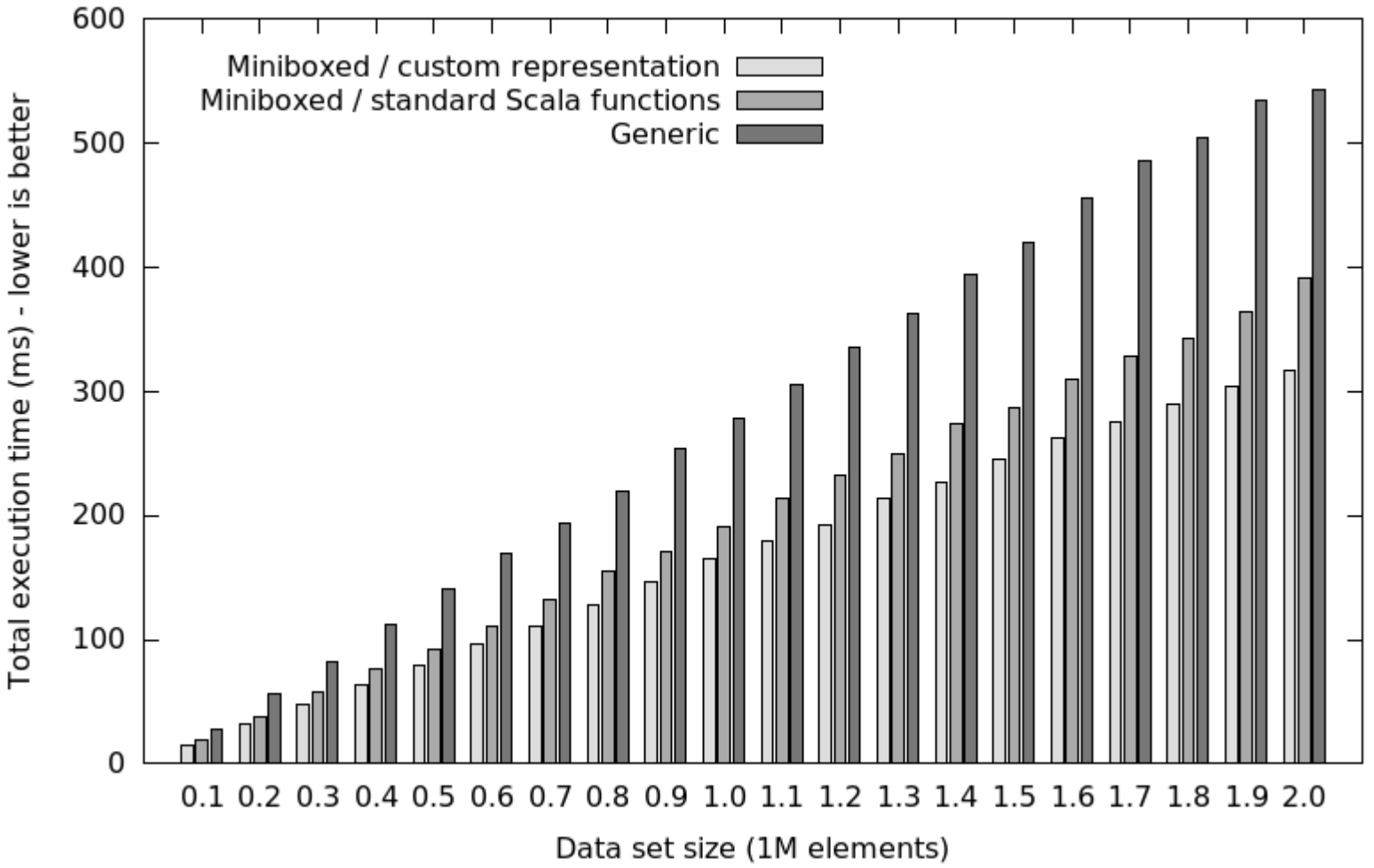
on the Scala library

- benchmark: Least Squares Method



Benchmarks

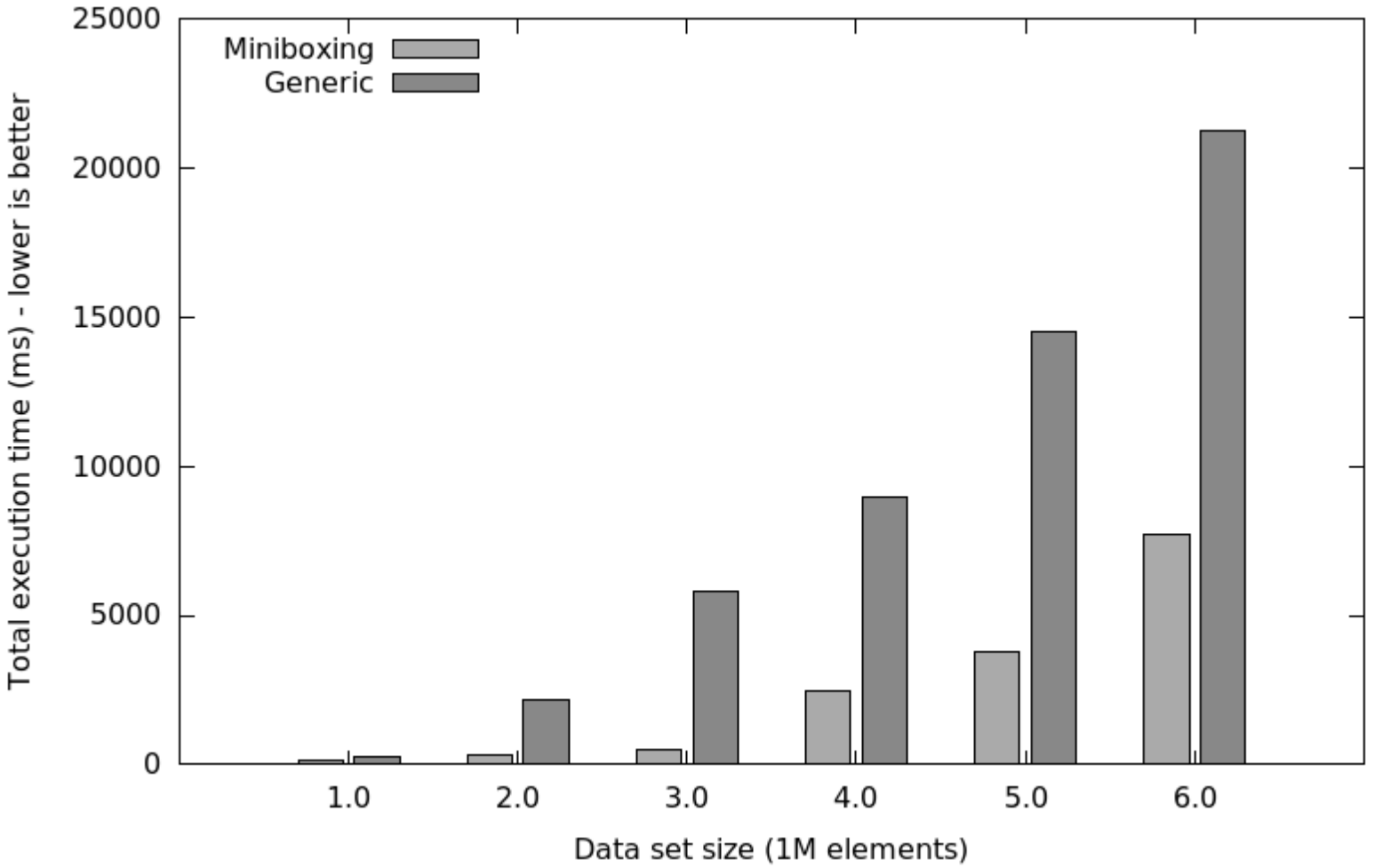
on the Scala library



30%-45% faster

Benchmarks

on the Scala library



30%-45% faster

Credits

- Cristian Talau - developed the initial prototype, as a semester project
- Eugene Burmako - the value class plugin based on the LDL transformation
- Aymeric Genet - developing collection-like benchmarks for the miniboxing plugin
- Martin Odersky, for his patient guidance
- Eugene Burmako, for trusting the idea enough to develop the value-plugin based on the LDL transformation
- Iulian Dragos, for his work on specialization and many explanations
- Miguel Garcia, for his original insights that spawned the miniboxing idea
- Michel Schinz, for his wonderful comments and enlightening ACC course
- Andrew Myers and Roland Ducournau for the discussions we had and the feedback provided
- Heather Miller for the eye-opening discussions we had
- Vojin Jovanovic, Sandro Stucki, Manohar Jonalagedda and the whole LAMP laboratory in EPFL for the extraordinary atmosphere
- Adriaan Moors, for the miniboxing name which stuck :))
- Thierry Coppey, Vera Salvisberg and George Nithin, who patiently listened to many presentations and provided valuable feedback
- Grzegorz Kossakowski, for the many brainstorming sessions on specialization
- Erik Osheim, Tom Switzer and Rex Kerr for their guidance on the Scala community side
- OOPSLA paper and artifact reviewers, who reshaped the paper with their feedback
- Sandro, Vojin, Nada, Heather, Manohar - reviews and discussions on the LDL paper
- Hubert Plociniczak for the type notation in the LDL paper
- Denys Shabalin, Dmitry Petrashko for their patient reviews of the LDL paper

functional or not,
do try it!

visit scala-miniboxing.org!