scala-miniboxing.org

14th of November 2014
PNWScala
Portland, OR

Vlad URECHE

PhD student in the Scala Team @ EPFL

Miniboxing guy. Also worked on Scala specialization, the backend and scaladoc.


@VladUreche


@VladUreche


vlad.ureche@epfl.ch

scala-miniboxing.org

scala-miniboxing.org

What is miniboxing?

Why use it?

How to use it?

Benchmarks

Conclusion

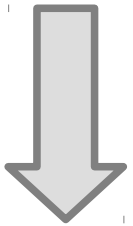
scala-miniboxing.org

Erased Generics

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

Erased Generics

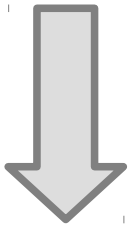
```
def identity[T](t: T): T = t
```



scalac / javac

Erased Generics

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



scalac / javac

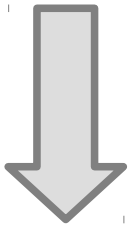
```
def identity(t: Object): Object = t
```

Erased Generics

identity(5)

Erased Generics

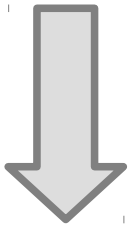
identity(5)



scalac / javac

Erased Generics

identity(5)

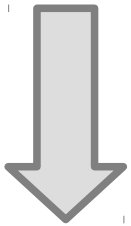


scalac / javac

identity(java.lang.Integer.valueOf(5))

Erased Generics

identity(5)



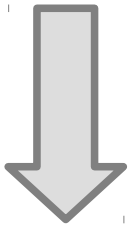
scalac / javac

identity(java.lang.Integer.valueOf(5))

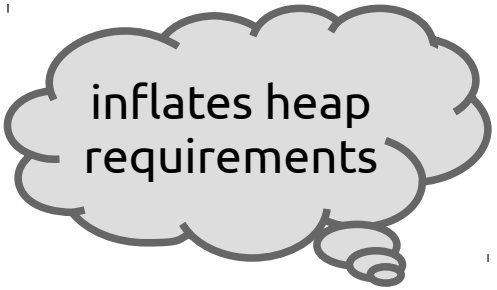
Object representation

Erased Generics

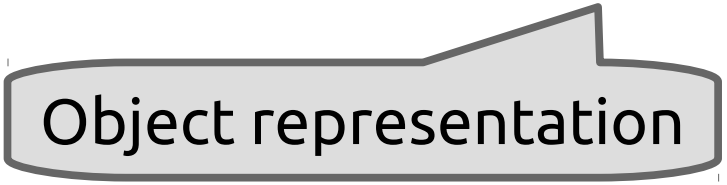
identity(5)



scalac / javac

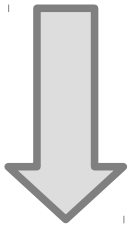


identity(java.lang.Integer.valueOf(5))

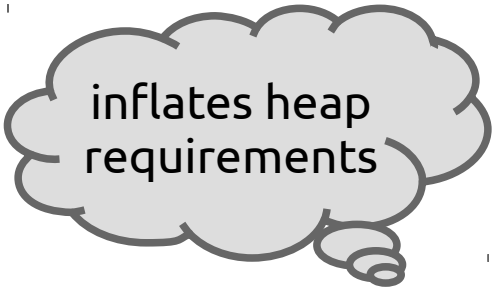


Erased Generics

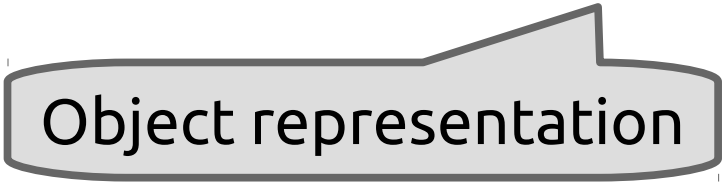
identity(5)



scalac / javac

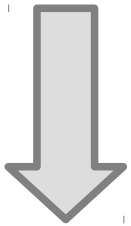


identity(java.lang.Integer.valueOf(5))

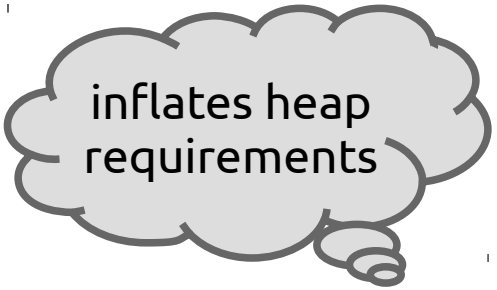


Erased Generics

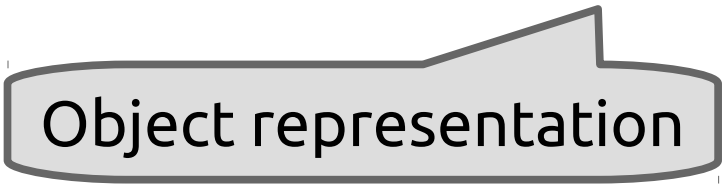
identity(5)

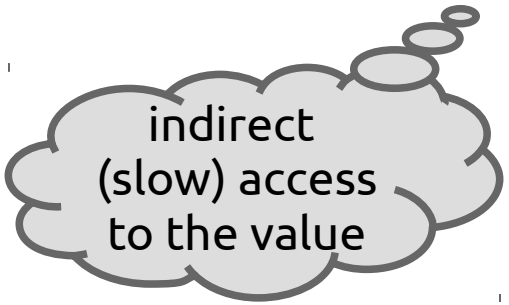


scalac / javac



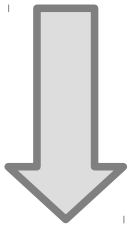
identity(java.lang.Integer.valueOf(5))



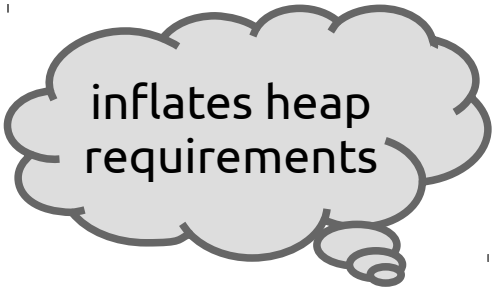


Erased Generics

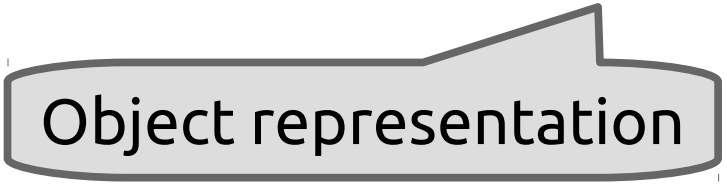
identity(5)

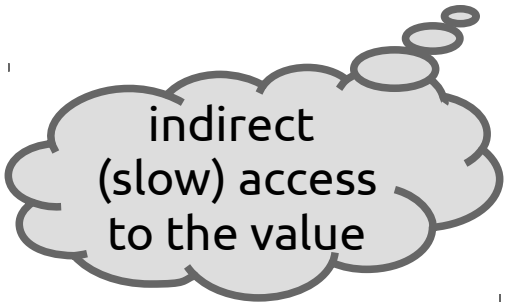


scalac / javac



identity(java.lang.Integer.valueOf(5))



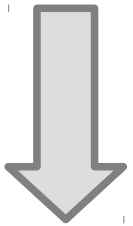


Specialization

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

Specialization

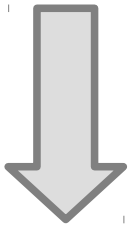
```
def identity[T](t: T): T = t
```



specialization

Specialization

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

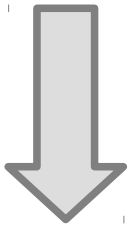


specialization

```
def identity(t: Object): Object = t
```

Specialization

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

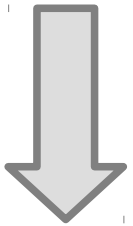


specialization

```
def identity(t: Object): Object = t
def identity_Z(t: bool): bool = t
```

Specialization

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

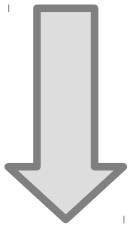


specialization

```
def identity(t: Object): Object = t
def identity_Z(t: bool): bool = t
def identity_C(t: char): char = t
```

Specialization

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



specialization

```
def identity(t: Object): Object = t
def identity_Z(t: bool): bool = t
def identity_C(t: char): char = t
```

... (7 other variants)

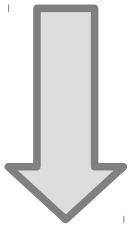
scala-miniboxing.org

Specialization

identity(5)

Specialization

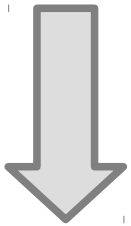
identity(5)



specialization

Specialization

identity(5)

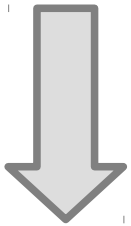


specialization

identity_I(5)

Specialization

identity(5)

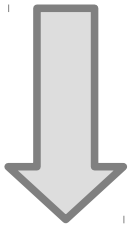


specialization

identity_I(5) **// no boxing!**

Specialization

identity(5)



specialization

identity_I(5) **// no boxing!**

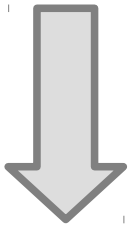
The variant of identity specialized for **int**

Specialization

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

Specialization

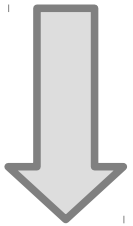
```
def tupled[T1, T2](t1: T1, t2: T2) ...
```



specialization

Specialization

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

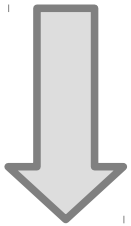


specialization

```
// 100 methods (10²)
```

Specialization

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

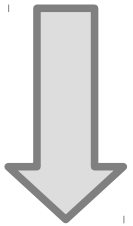


specialization

// 100 methods (10^2)

Specialization

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



specialization

```
// 100 methods (10²)
```

Specialization

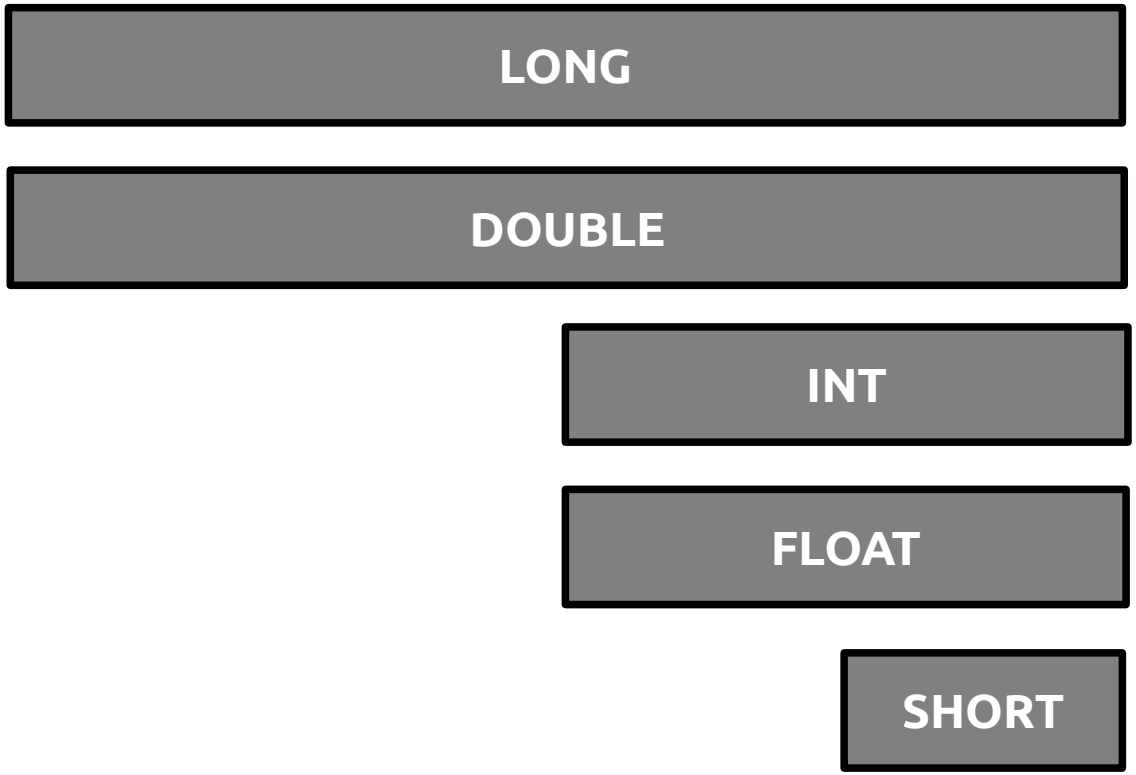
the insight

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!"*

Specialization

the insight

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!"*



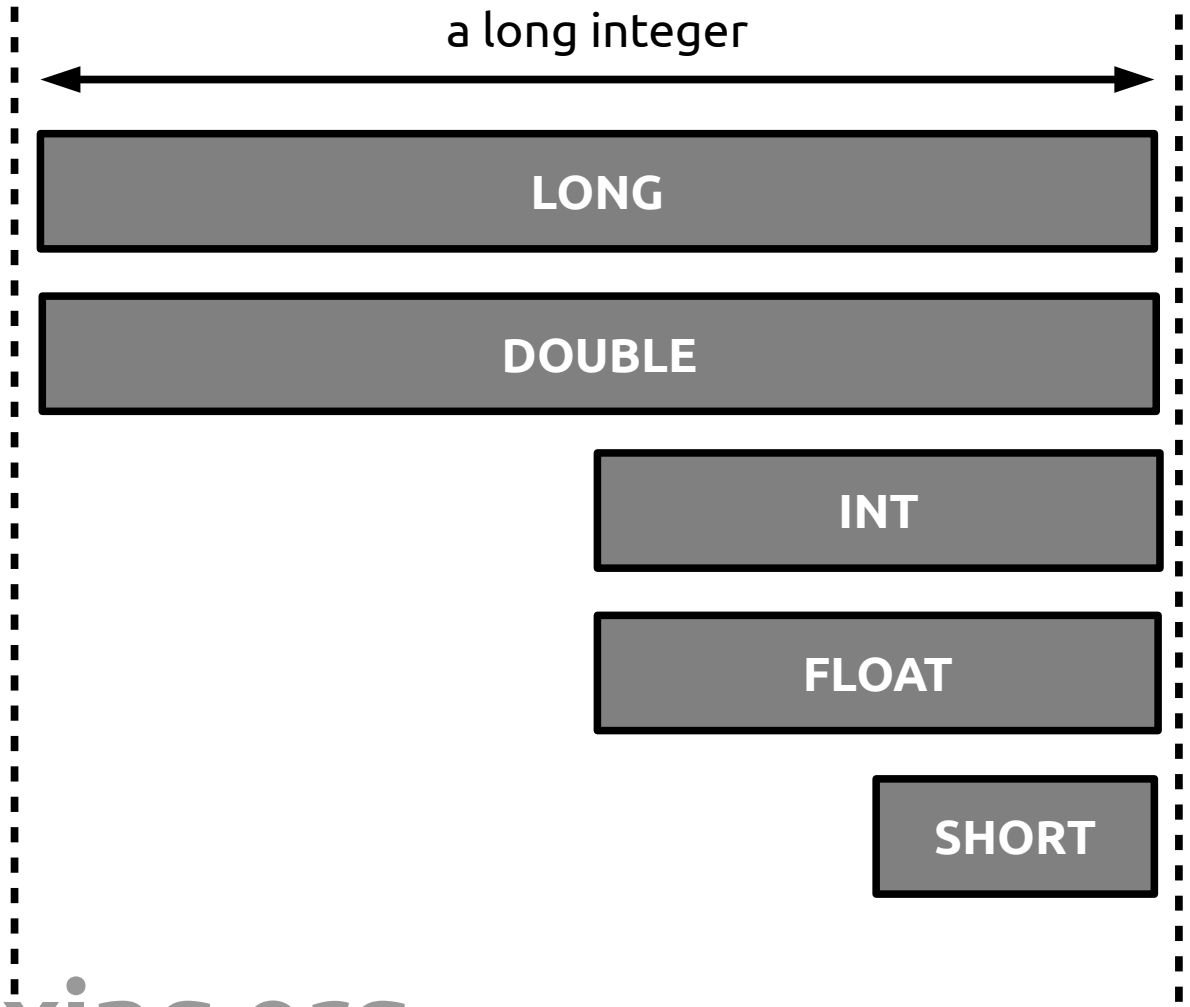
scala-miniboxing.org

...

Specialization

the insight

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!"*



scala-miniboxing.org

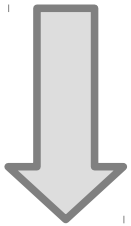
Miniboxing


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

scala-miniboxing.org

Miniboxing


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

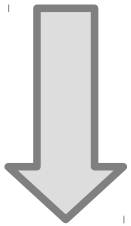


miniboxing

scala-miniboxing.org

Miniboxing


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



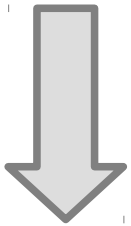
miniboxing

```
def identity(t: Object): Object = t
```

scala-miniboxing.org

Miniboxing


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



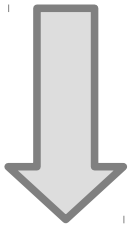
miniboxing

```
def identity(t: Object): Object = t
def identity_M(..., t: long): long = t
```

scala-miniboxing.org

Miniboxing


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



miniboxing

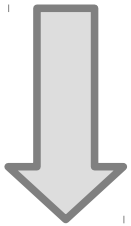
```
def identity(t: Object): Object = t
def identity_M(..., t: long): long = t
```

long **encodes** all primitive types

scala-miniboxing.org

Miniboxing

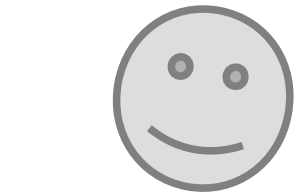

```
def identity[T](t: T): T = t
```



miniboxing

```
def identity(t: Object): Object = t
def identity_M(..., t: long): long = t
```

long **encodes** all primitive types



Only 2^n variants

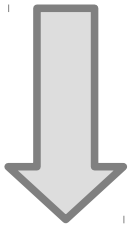
scala-miniboxing.org

Miniboxing

identity(3)

Miniboxing

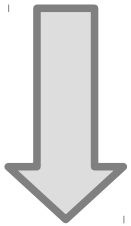
identity(3)



miniboxing

Miniboxing

identity(3)

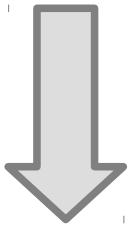


miniboxing

identity_M(..., int2minibox(3))

Miniboxing

identity(3)



miniboxing

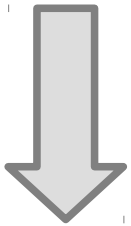
identity_M(..., int2minibox(3))

The miniboxed
variant of identity

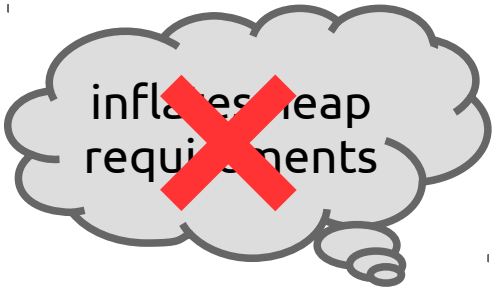
scala-miniboxing.org

Miniboxing

identity(3)



miniboxing



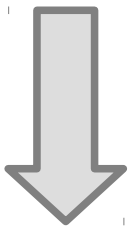
identity_M(..., int2minibox(3))

The miniboxed
variant of identity

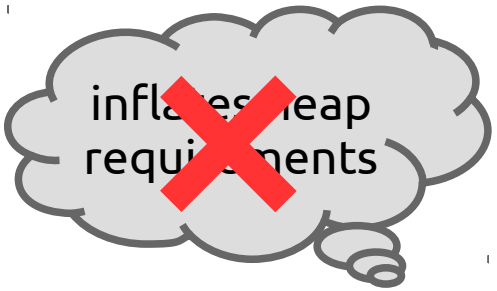
scala-miniboxing.org

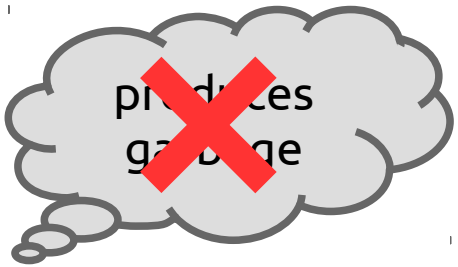
Miniboxing

identity(3)



miniboxing





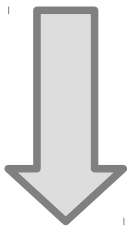
identity_M(..., int2minibox(3))

The miniboxed
variant of identity

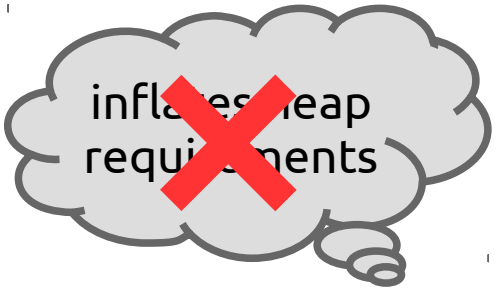
scala-miniboxing.org

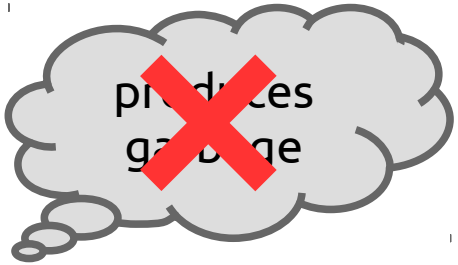
Miniboxing

identity(3)



miniboxing





identity_M(..., int2minibox(3))

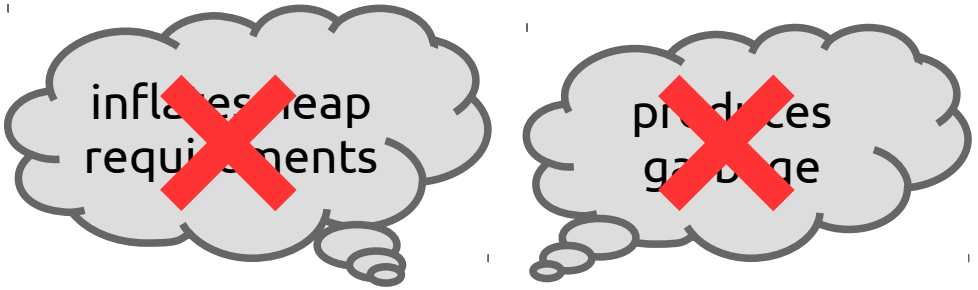
The miniboxed variant of identity

scala-miniboxing.org

Miniboxing

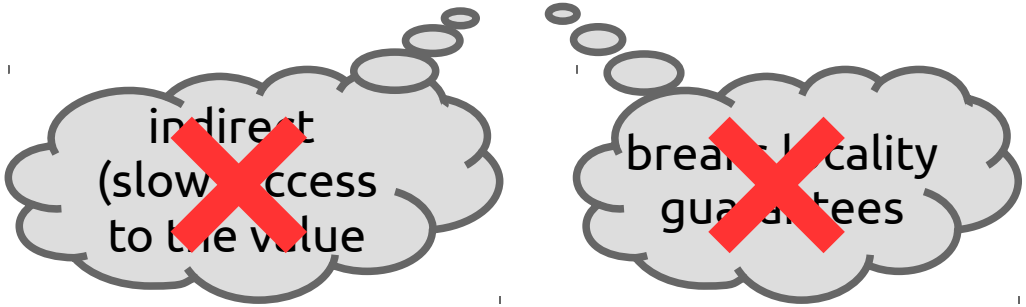
identity(3)

miniboxing



identity_M(..., int2minibox(3))

The miniboxed variant of identity



scala-miniboxing.org

What is miniboxing?

Why use it?

How to use it?

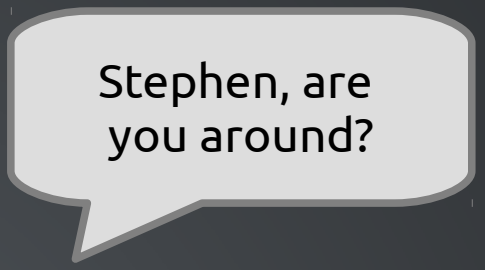
Benchmarks

Conclusion

scala-miniboxing.org

PUREIMAGE: RASTER IMAGE PROCESSING FOR SCALA

http://stephenjudkins.github.io/pureimage-presentation/#/



PUREIMAGE: RASTER IMAGE PROCESSING FOR SCALA

http://stephenjudkins.github.io/pureimage-presentation/#/

IN PRODUCTION, NO SINGLE OPTIMIZATION PROVED AS FRUITFUL AS AVOIDING BOXING

http://stephenjudkins.github.io/pureimage-presentation/#/32

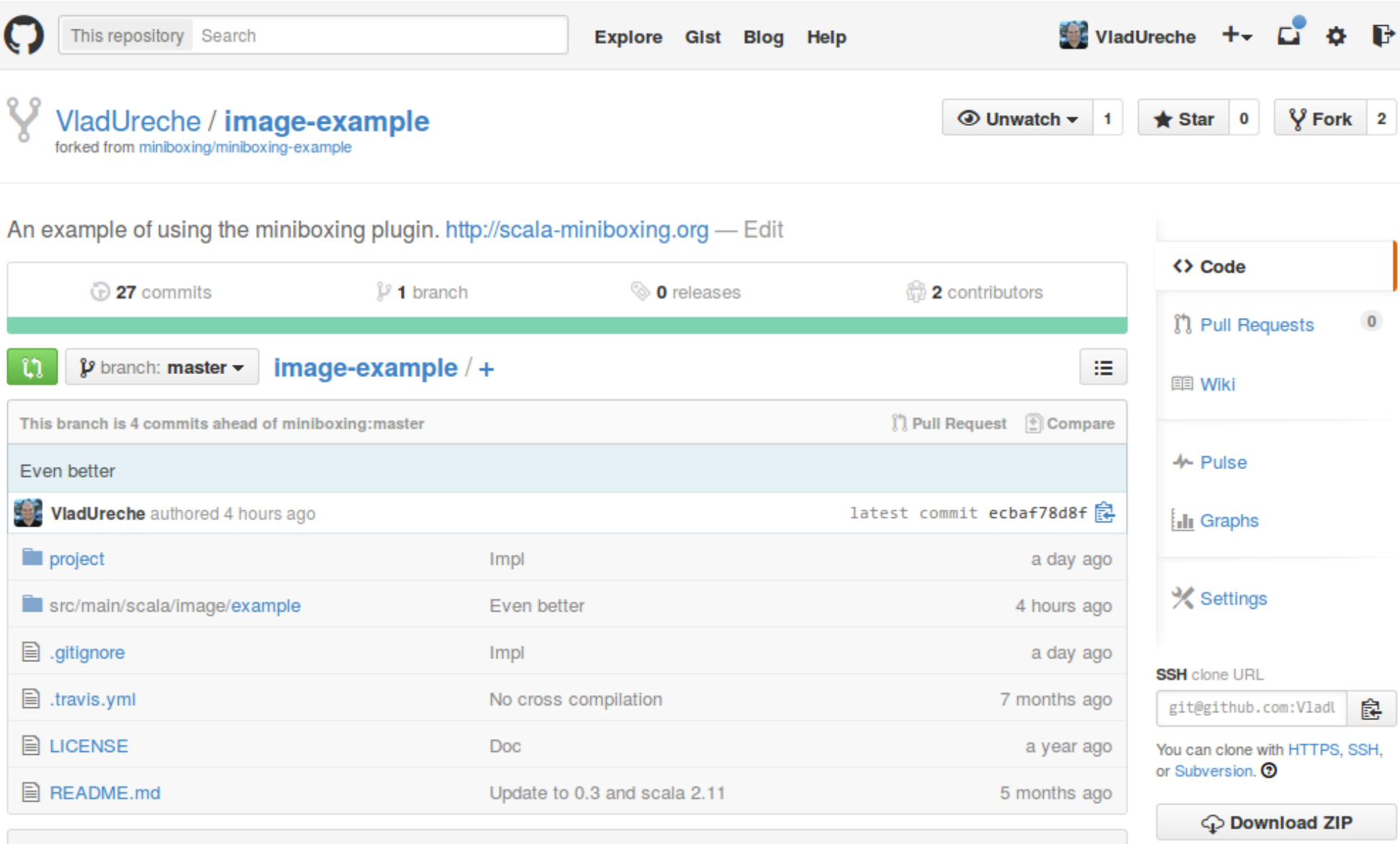
PureImage

- has very nice abstractions
  - generalizes over input, output format
  - generalizes over pixel format
  - provides collection-like mapping over images
- liked it a lot

PureImage

- took the usual path to using miniboxing
  - code up a mock-up
  - become familiar with the problem
  - try out miniboxing on a small scale
  - then extend to the whole program [not yet]

PureImage Mock-up

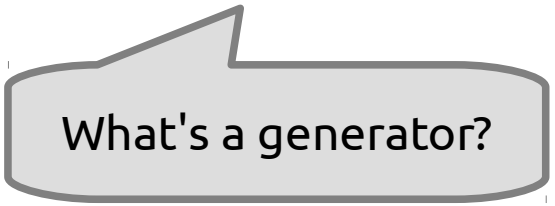


PureImage Mock-up

```
abstract class Image[Repr: Pixel] {
  def width: Int
  def height: Int
  def apply(x: Int, y: Int): Repr
  def map[Repr2: Pixel](f: Image[Repr] =>
        Generator[Repr2]): Image[Repr2]
}
```

PureImage Mock-up

```
abstract class Image[Repr: Pixel] {
  def width: Int
  def height: Int
  def apply(x: Int, y: Int): Repr
  def map[Repr2: Pixel](f: Image[Repr] =>
        Generator[Repr2]): Image[Repr2]
}
```



PureImage Mock-up

```
abstract class Generator[Repr: Pixel] {
  def width: Int
  def height: Int
  def generate(x: Int, y: Int): Repr
}
```

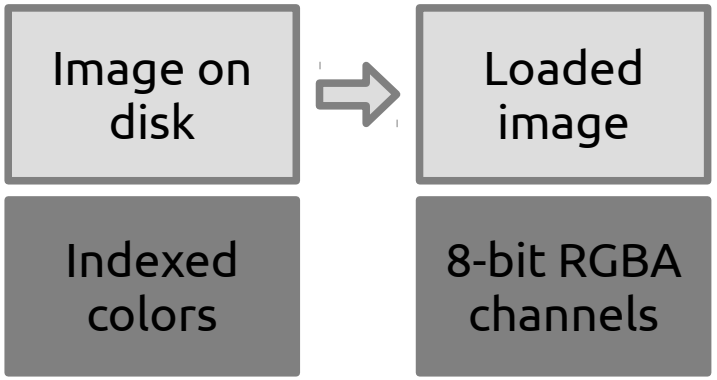
PureImage Mock-up

Why generic? What is pixel?

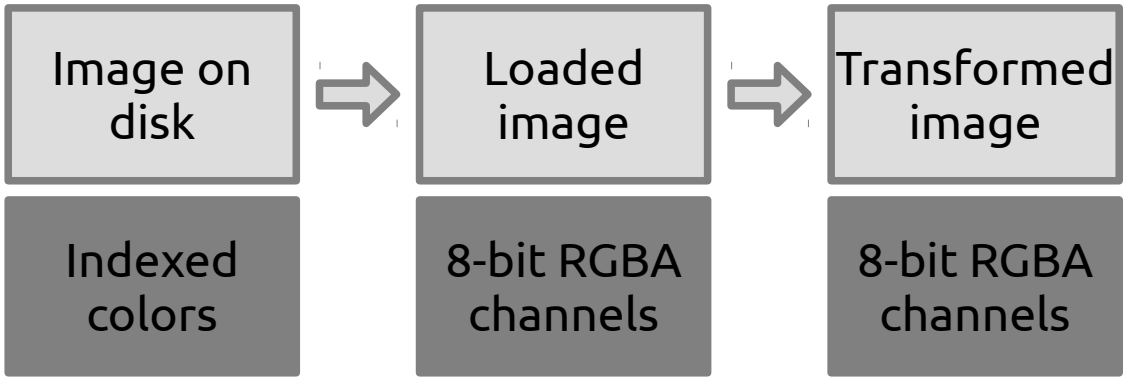
```
abstract class Generator[Repr: Pixel] {
  def width: Int
  def height: Int
  def generate(x: Int, y: Int): Repr
}
```

PureImage Mock-up

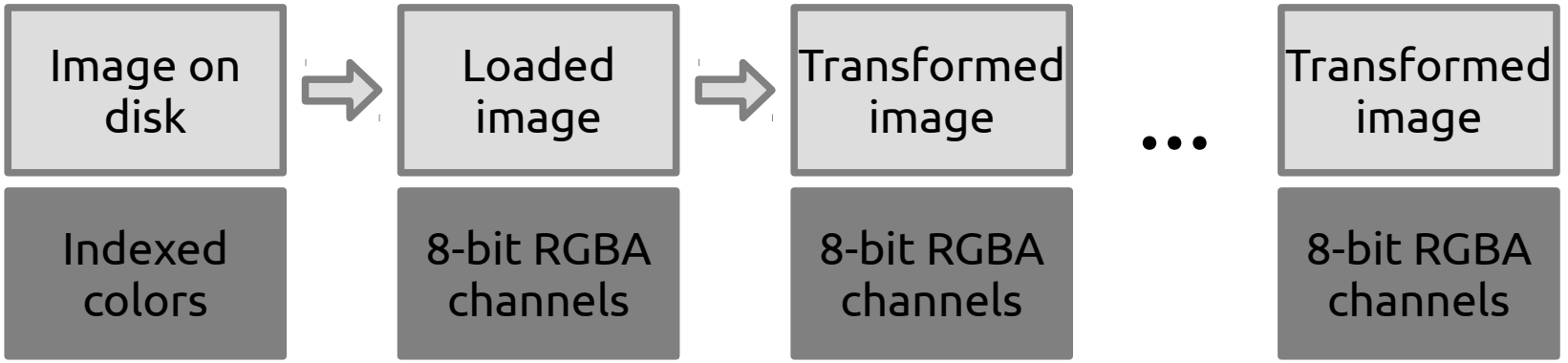
PureImage Mock-up



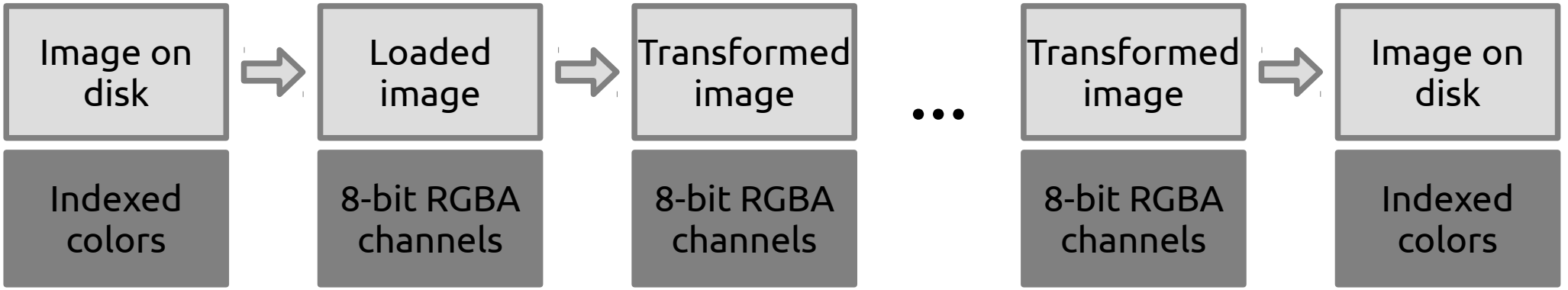
PureImage Mock-up



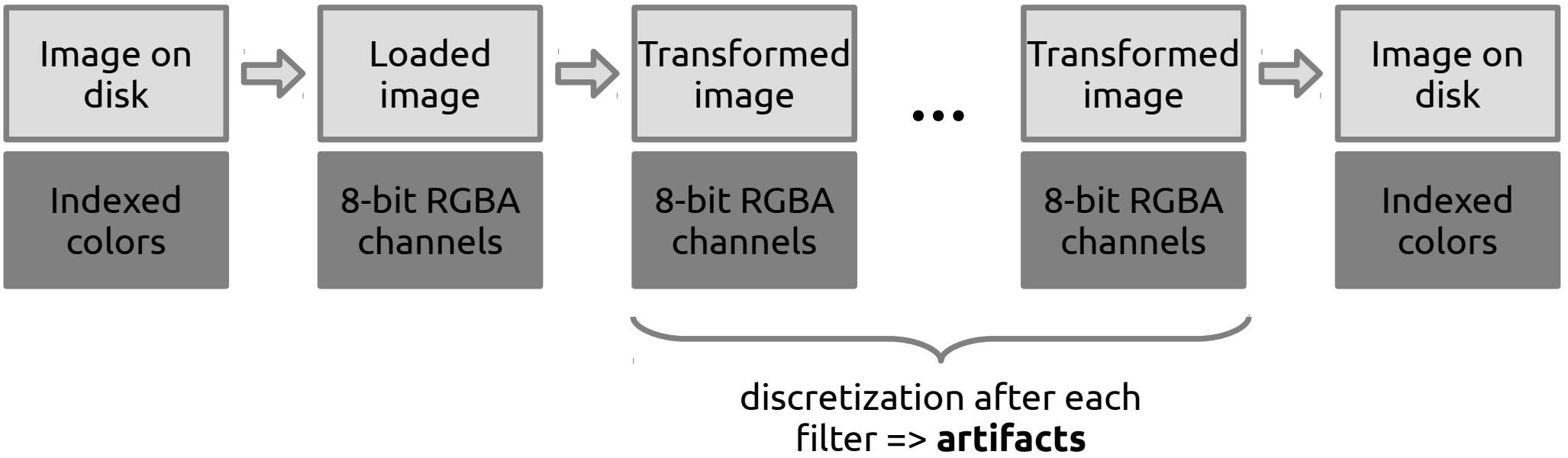
PureImage Mock-up



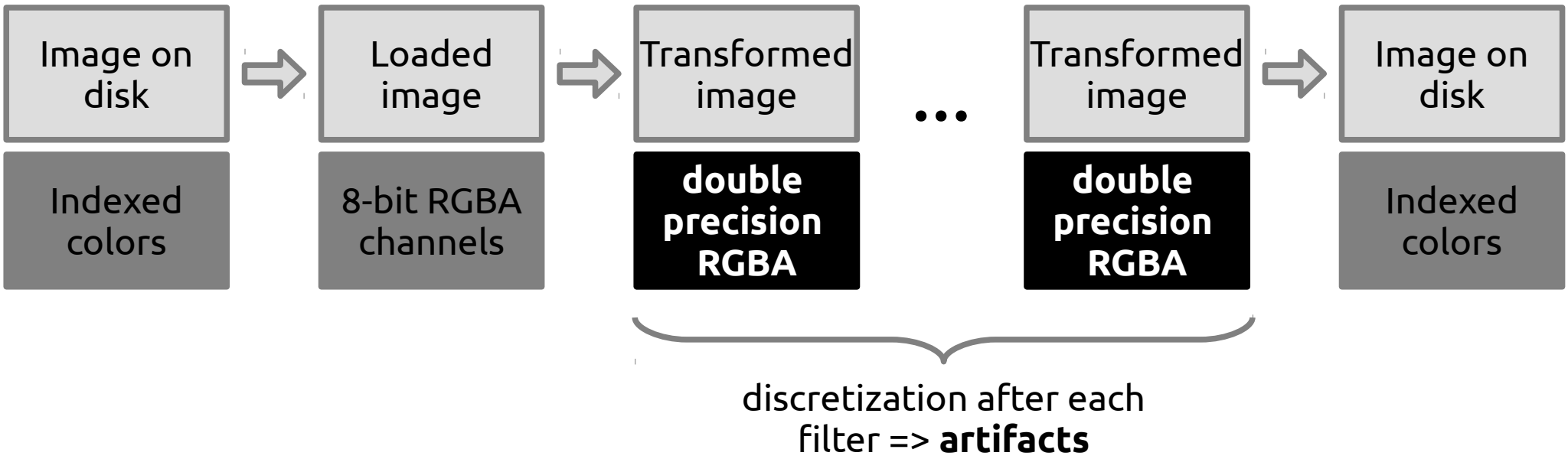
PureImage Mock-up



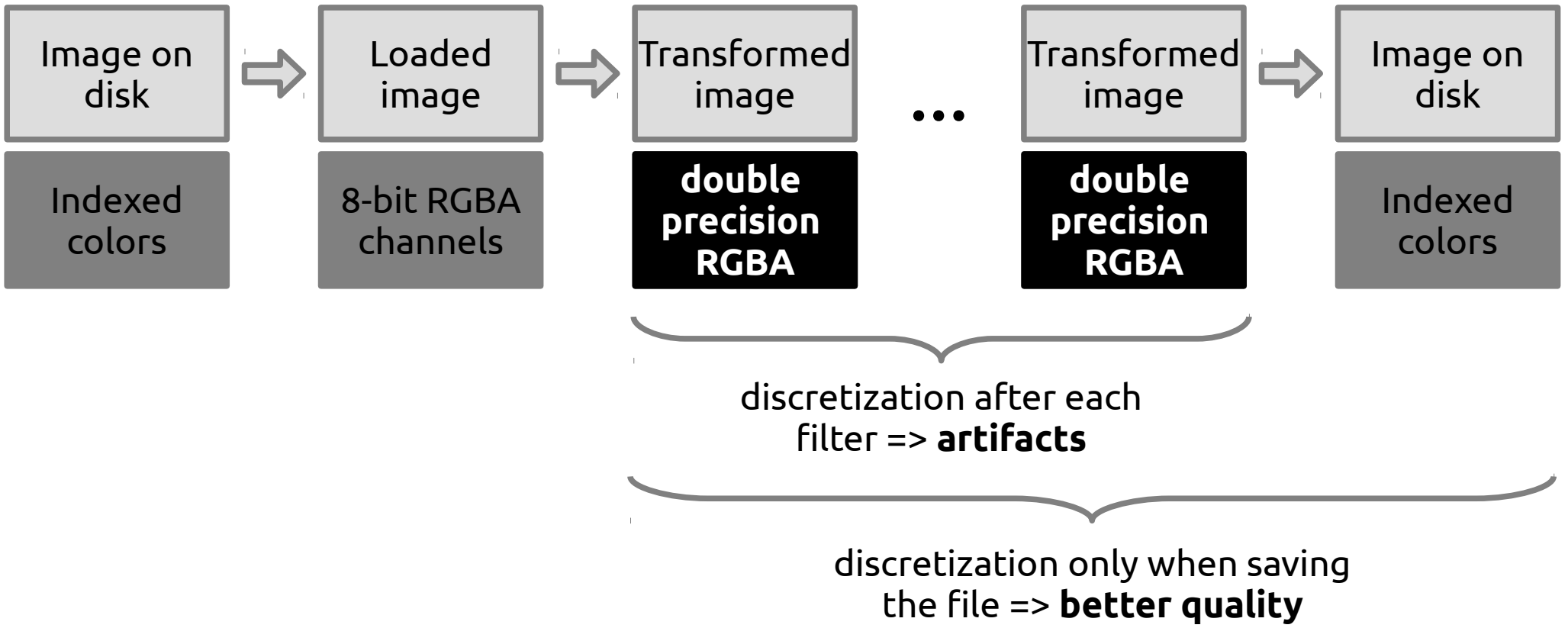
PureImage Mock-up



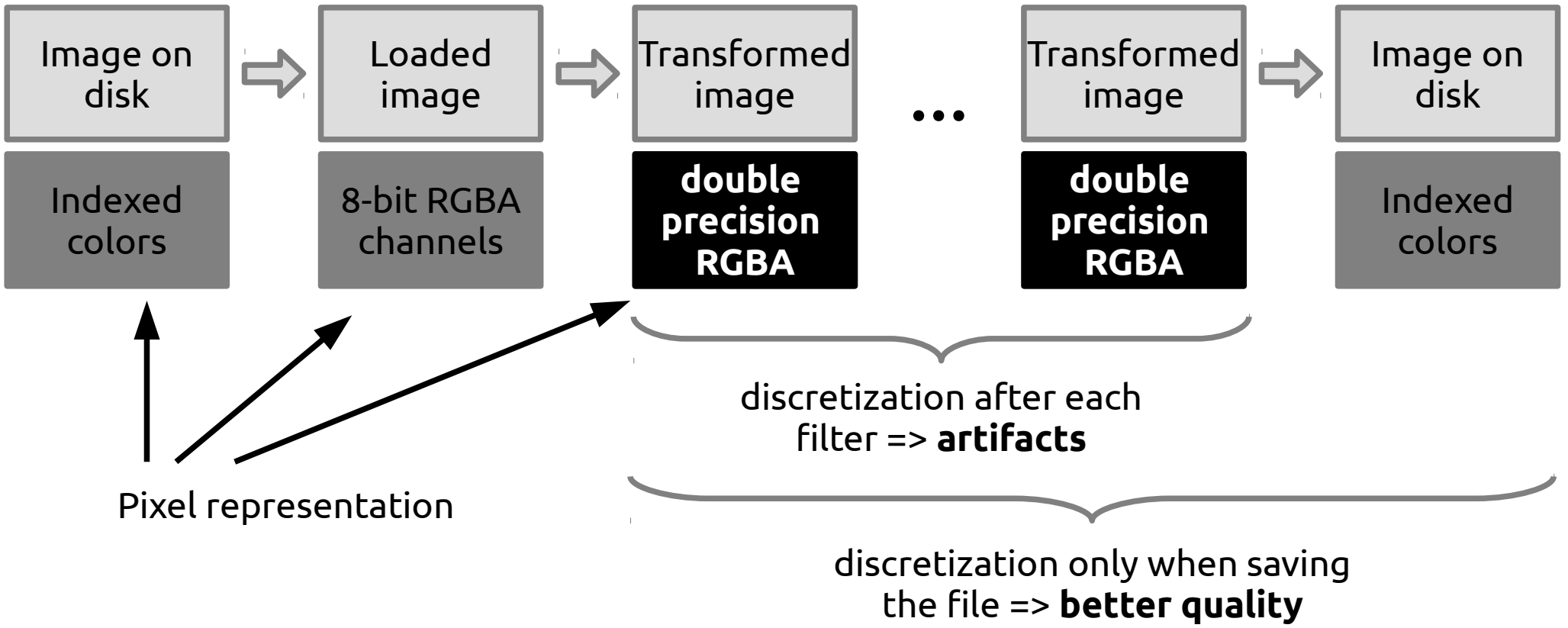
PureImage Mock-up



PureImage Mock-up



PureImage Mock-up



PureImage Mock-up

```
abstract class Pixel[Repr: Manifest] {
  def r(t: Repr): Double // red
  def g(t: Repr): Double // green
  def b(t: Repr): Double // blue
  def a(t: Repr): Double // alpha
  def pack(r: Double,
           g: Double,
           b: Double,
           a: Double): Repr

  ...
}
```

PureImage Mock-up

```
abstract class Pixel[Repr: Manifest] {
  def r(t: Repr): Double // red
  def g(t: Repr): Double // green
  def b(t: Repr): Double // blue
  def a(t: Repr): Double // alpha
  def pack(r: Double,
           g: Double,
           b: Double,
           a: Double): Repr

  ...
}
```

All transformations work on double precision FP numbers

PureImage Mock-up

```
abstract class Pixel[Repr: Manifest] {
  def r(t: Repr): Double // red
  def g(t: Repr): Double // green
  def b(t: Repr): Double // blue
  def a(t: Repr): Double // alpha
  def pack(r: Double,
           g: Double,
           b: Double,
           a: Double): Repr
  ...
}
```

All transformations work on double precision FP numbers

But the data can be encoded differently

PureImage Mock-up

```
object RGBA extends Pixel[Int] { ... }
object RGBAExtended extends Pixel[Long] { ... }
```

PureImage Mock-up


```
object RGBA extends Pixel[Int] { ... }
object RGBAExtended extends Pixel[Long] { ... }
```

PureImage Mock-up

Encoding all channels

```
object RGBA extends Pixel[Int] { ... }
object RGBAExtended extends Pixel[Long] { ... }
```

```
case class DiscreteChannels[T](r: T, g: T, b: T, a: T)
object FullPixel extends FourChannelPixel[Double]
object HalfPixel extends FourChannelPixel[Float]
```

PureImage Mock-up

Encoding all channels

```
object RGBA extends Pixel[Int] { ... }
object RGBAExtended extends Pixel[Long] { ... }
```

Storing channels individually

```
case class DiscreteChannels[T](r: T, g: T, b: T, a: T)
object FullPixel extends FourChannelPixel[Double]
object HalfPixel extends FourChannelPixel[Float]
```

PureImage Mock-up

DEMO

What is miniboxing?

Why use it?

How to use it?

Benchmarks

Conclusion

scala-miniboxing.org

How to use it?

- Sbt Configuration
- Guidance
- Website

scala-miniboxing.org

How to use it?

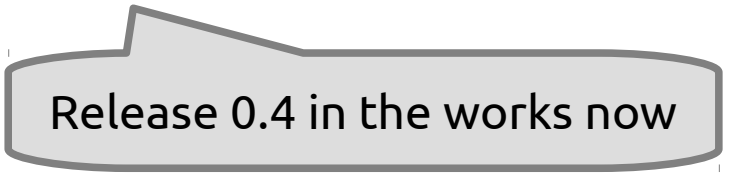
- Sbt Configuration
- Guidance
- Website

build.sbt

```
resolvers += Resolver.sonatypeRepo("snapshots")

libraryDependencies += "org.scala-miniboxing.plugins" %%
                       "miniboxing-runtime" % "0.4-SNAPSHOT"

addCompilerPlugin("org.scala-miniboxing.plugins" %%
                  "miniboxing-plugin" % "0.4-SNAPSHOT")
```

build.sbt

```
resolvers += Resolver.sonatypeRepo("snapshots")

libraryDependencies += "org.scala-miniboxing.plugins" %%
            "miniboxing-runtime" % "0.4-SNAPSHOT"

addCompilerPlugin("org.scala-miniboxing.plugins" %%
            "miniboxing-plugin" % "0.4-SNAPSHOT")
```



build.sbt

```
scalacOptions ++=
  "-P:minibox:hijack" ::      // transform @specialized into @miniboxed
  "-P:minibox:mark-all" :: // mark all type parameters as @miniboxed
  "-P:minibox:log" ::         // explain how classes are transformed
  "-P:minibox:warn" ::        // warn for suboptimal code
  "-P:minibox:warn-all" :: // warn for suboptimal code across projects
  Nil
```

build.sbt

```
scalacOptions ++=
  "-P:minibox:hijack" ::      // transform @specialized into @miniboxed
  "-P:minibox:mark-all" :: // mark all type parameters as @miniboxed
  "-P:minibox:log" ::         // explain how classes are transformed
  "-P:minibox:warn" ::        // warn for suboptimal code
  "-P:minibox:warn-all" :: // warn for suboptimal code across projects
  Nil
```

warn vs warn-all

```
scala> 3 :: Nil   // under -P:minibox:warn
res0: List[Int] = List(3)
```

warn vs warn-all

```
scala> 3 :: Nil   // under -P:minibox:warn
res0: List[Int] = List(3)

scala> 3 :: Nil  // under -P:minibox:warn-all
<console>:8: warning: The method List.:: would benefit from miniboxing
type parameter B, since it is instantiated by a primitive type.
              3 :: Nil
                ^
res0: List[Int] = List(3)
```

warn vs warn-all

```
scala> 3 :: Nil   // under -P:minibox:warn
res0: List[Int] = List(3)

scala> 3 :: Nil  // under -P:minibox:warn-all
<console>:8: warning: The method List.:: would benefit from miniboxing type parameter B, since it is instantiated by a primitive type.
              3 :: Nil
                ^
res0: List[Int] = List(3)
```

warn vs warn-all

```
scala> 3 :: Nil   // under -P:minibox:warn
res0: List[Int] = List(3)

scala> 3 :: Nil  // under -P:minibox:warn-all
<console>:8: warning: The method List.:: would benefit from miniboxing type parameter B, since it is instantiated by a primitive type.
              3 :: Nil
                ^
res0: List[Int] = List(3)
```

Warning for the scala library :)

warn vs warn-all

```
scala> 3 :: Nil   // under -P:minibox:warn
res0: List[Int] = List(3)

scala> 3 :: Nil  // under -P:minibox:warn-all
<console>:8: warning: The method List.:: would benefit from miniboxing type parameter B, since it is instantiated by a primitive type.
              3 :: Nil
                ^
res0: List[Int] = List(3)
```

Warning for the scala library :)

Across projects, not limited to the program being compiled.

How to use it?

- Sbt Configuration
- Guidance
- Website

Quirks of Scala Specialization

posted by Alex, 03.11.2013.

Having used specialization a lot and having fixed some of its issues, I came across a couple of useful tricks – I want to document them both for myself and others. Specialization is the feature that allows you to generate separate versions of generic classes for primitive types, thus avoiding boxing in most cases. First introduced in Scala 2.8 by Iulian Dragos, by Scala 2.11 specialization has become a pretty robust language feature, and a lot of its issues have been fixed, but there are places where it might stab you in the back if you don't watch out. Problem is, specialization interacts with a some edge-cases in the language and obscure language features in ways that are not expected. Sometimes, these are just unresolved bugs. Here are some tips and tricks that might help you.

Note: when used correctly, this is a powerful and extremely useful feature few JVM languages (if any) can parallel these days. Don't get scared by these tips.

Know the conditions for method specialization

Perhaps you're not aware of this, but even if a method is a part of a specialized class and contains specializable code, it will not really be specialized unless the specialized type appears in its argument list or its return type. For example:

```
def getValue[@specialized T]: T = ???

class Foo[@specialized T] {
  var value: T = _
  def reset() {
    value = getValue
  }
}
```

Quirks of Scala Specialization

posted by Alex, 03.11.2013.

Alex Prokopec, axel22.github.io

Having used specialization a lot and having fixed some of its issues, I came across a couple of useful tricks – I want to document them both for myself and others. Specialization is the feature that allows you to generate separate versions of generic classes for primitive types, thus avoiding boxing in most cases. First introduced in Scala 2.8 by Iulian Dragos, by Scala 2.11 specialization has become a pretty robust language feature, and a lot of its issues have been fixed, but there are places where it might stab you in the back if you don't watch out. Problem is, specialization interacts with a some edge-cases in the language and obscure language features in ways that are not expected. Sometimes, these are just unresolved bugs. Here are some tips and tricks that might help you.

Note: when used correctly, this is a powerful and extremely useful feature few JVM languages (if any) can parallel these days. Don't get scared by these tips.

Know the conditions for method specialization

Perhaps you're not aware of this, but even if a method is a part of a specialized class and contains specializable code, it will not really be specialized unless the specialized type appears in its argument list or its return type. For example:

```
def getValue[@specialized T]: T = ???

class Foo[@specialized T] {
  var value: T = _
  def reset() {
    value = getValue
  }
}
```

Quirks of Scala Specialization

posted by Alex, 03.11.2013.

Alex Prokopec, axel22.github.io

designer of the parallel collections

Having used specialization a lot and havin[illegible]ks – I want to document them both for myself and [illegible]generate separate versions of generic classes for primitive types, thus avoiding boxing in most cases. First introduced in Scala 2.8 by Iulian Dragos, by Scala 2.11 specialization has become a pretty robust language feature, and a lot of its issues have been fixed, but there are places where it might stab you in the back if you don't watch out. Problem is, specialization interacts with a some edge-cases in the language and obscure language features in ways that are not expected. Sometimes, these are just unresolved bugs. Here are some tips and tricks that might help you.

Note: when used correctly, this is a powerful and extremely useful feature few JVM languages (if any) can parallel these days. Don't get scared by these tips.

Know the conditions for method specialization

Perhaps you're not aware of this, but even if a method is a part of a specialized class and contains specializable code, it will not really be specialized unless the specialized type appears in its argument list or its return type. For example:

```
def getValue[@specialized T]: T = ???

class Foo[@specialized T] {
  var value: T = _
  def reset() {
    value = getValue
  }
}
```

Quirks of Scala Specialization

posted by Alex, 03.11.2013.

Alex Prokopec, axel22.github.io

designer of the parallel collections

- performance (on the JVM) is **magic**

Quirks of Scala Specialization

posted by Alex, 03.11.2013.

Alex Prokopec, axel22.github.io

designer of the parallel collections

- performance (on the JVM) is **magic**
- specialization (in scalac) is **black magic**

Quirks of Scala Specialization

posted by Alex, 03.11.2013.

Alex Prokopec, axel22.github.io

designer of the parallel collections

- performance (on the JVM) is **magic**
- specialization (in scalac) is **black magic**
  - “know the conditions of specialization”
  - point n: “Use Traits”
  - point n+1: “Don't use Traits”
  - … and other 10 pieces of advice ...
  - is your code running at max performance?
    - who knows?

Quirks of Scala Specialization

posted by Alex, 03.11.2013.

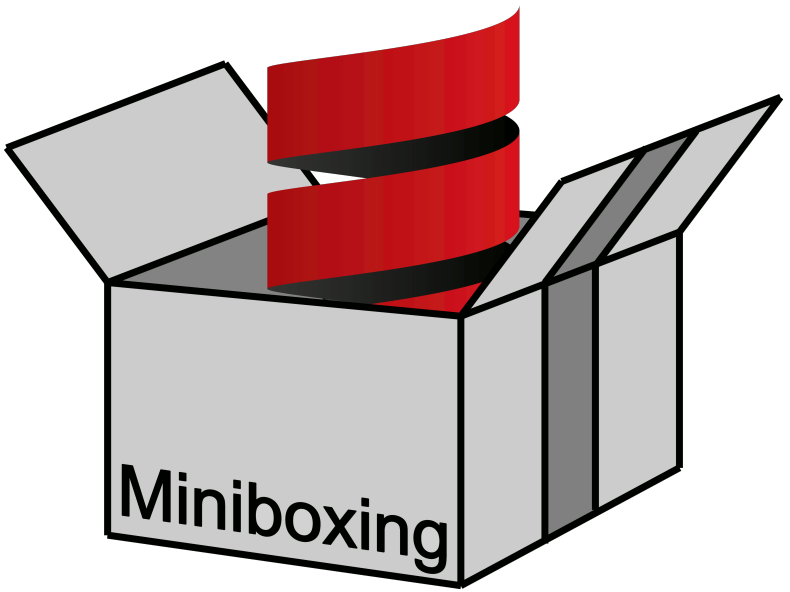
Alex Prokopec, axel22.github.io

designer of the parallel collections

- performance (on the JVM) is **magic**
- specialization (in scalac) is **black magic**
  - “know the conditions of specialization”
  - point n: “Use Traits”
  - point n+1: “Don't use Traits”
  - … and other 10 pieces of advice ...
  - is your code running at max performance?
    - who knows?

Can we do something about this?

Guidance



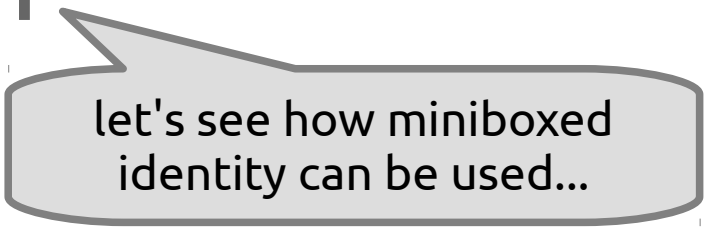
-P:minibox:warn

Guidance

```
scala> def identity[@miniboxed T](t: T) = t
foo: [T](t: T)T
```

Guidance

```
scala> def identity[@miniboxed T](t: T) = t
foo: [T](t: T)T
```



Guidance

Guidance

```
scala> identity(3)
res1: Int = 3
```

Guidance

```
scala> identity(3)
res1: Int = 3

scala> identity("3")
res2: String = 3
```

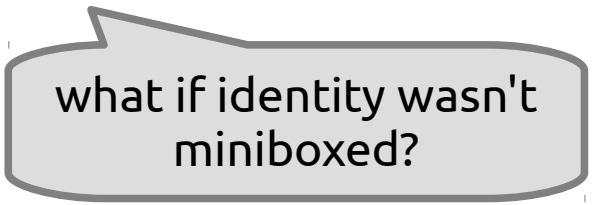
Guidance

```
scala> identity(3)
res1: Int = 3

scala> identity("3")
res2: String = 3

scala> identity[Any](3)
<console>:9: warning: Using the type argument "Any" for the miniboxed type
parameter T of method identity is not specific enough, as it could mean
either a primitive or a reference type. Although method foo is miniboxed, it
won't benefit from specialization:
              identity[Any](3)
              ^
res3: Any = 3
```

scala-miniboxing.org

Guidance

```
scala> identity(3)
res1: Int = 3

scala> identity("3")
res2: String = 3

scala> identity[Any](3)
<console>:9: warning: Using the type argument "Any" for the miniboxed type
parameter T of method identity is not specific enough, as it could mean
either a primitive or a reference type. Although method foo is miniboxed, it
won't benefit from specialization:
              identity[Any](3)
              ^
res3: Any = 3
```



Guidance

Guidance

```
scala> def foo[T](t: T) = t
foo: [T](t: T)T
```

Guidance

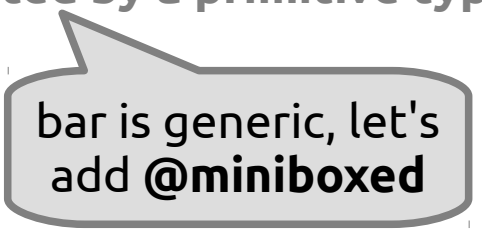
```
scala> def foo[T](t: T) = t
foo: [T](t: T)T

scala> def bar[T](t: T) = foo(t)
bar: [T](t: T)T
```

Guidance

```
scala> def foo[T](t: T) = t
foo: [T](t: T)T

scala> def bar[T](t: T) = foo(t)
bar: [T](t: T)T

scala> bar(3)
<console>:10: warning: The method bar would benefit from miniboxing type
parameter T, since it is instantiated by a primitive type.
              bar(3)
              ^
res1: Int = 3
```

Guidance

```
scala> def foo[T](t: T) = t
foo: [T](t: T)T

scala> def bar[T](t: T) = foo(t)
bar: [T](t: T)T

scala> bar(3)
<console>:10: warning: The method bar would benefit from miniboxing type
parameter T, since it is instantiated by a primitive type.
              bar(3)
              ^
res1: Int = 3
```



Guidance

```
scala> def foo[T](t: T) = t
foo: [T](t: T)T
```

Guidance

```
scala> def foo[T](t: T) = t
foo: [T](t: T)T

scala> def bar[@miniboxed T](t: T) = foo(t)
<console>:8: warning: The method foo would benefit from miniboxing type parameter T, since it is instantiated by miniboxed type parameter T of method bar.
       def bar[@miniboxed T](t: T) = foo(t)
                                     ^
bar: [T](t: T)T
```

Guidance

```
scala> def foo[T](t: T) = t
foo: [T](t: T)T

scala> def bar[@miniboxed T](t: T) = foo(t)
<console>:8: warning: The method foo would benefit from miniboxing type
parameter T, since it is instantiated by miniboxed type parameter T of method
bar.
     def bar[@miniboxed T](t: T) = foo(t)
                                   ^
bar: [T](t: T)T

scala> bar(3)
res1: Int = 3
```

Guidance

```
scala> def foo[T](t: T) = t
foo: [T](t: T)T

scala> def bar[@miniboxed T](t: T) = foo(t)
<console>:8: warning: The method foo would benefit from miniboxing type
parameter T, since it is instantiated by miniboxed type parameter T of method
bar.
     def bar[@miniboxed T](t: T) = foo(t)
                                   ^
bar: [T](t: T)T

scala> bar(3)
res1: Int = 3
```

Why? Because the miniboxed bar should call miniboxed foo, but foo is not miniboxed...

Guidance

```
scala> def foo[@miniboxed T](t: T) = t
foo: [T](t: T)T
```

Guidance

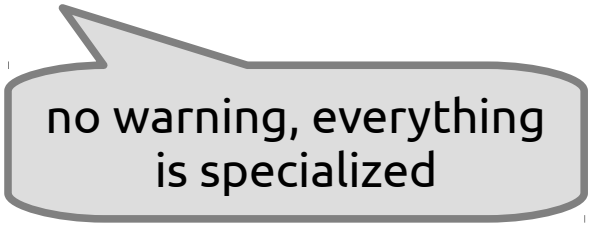
```
scala> def foo[@miniboxed T](t: T) = t
foo: [T](t: T)T

scala> def bar[@miniboxed T](t: T) = foo(t)
bar: [T](t: T)T
```

Guidance

```
scala> def foo[@miniboxed T](t: T) = t
foo: [T](t: T)T

scala> def bar[@miniboxed T](t: T) = foo(t)
bar: [T](t: T)T

scala> bar(3)
res1: Int = 3
```

Guidance

```
scala> def foo[@miniboxed T](t: T) = t
foo: [T](t: T)T

scala> def bar[@miniboxed T](t: T) = foo(t)
bar: [T](t: T)T

scala> bar(3)
res1: Int = 3
```



Guidance

- “optimized trace”
  - one miniboxed method calling another
    - data uses miniboxing encoding
  - three patterns
    - initiator – starts an optimized trace
    - propagator – propagates it
    - inhibitor – goes back to boxed :(

Guidance

- “optimized trace”
  - one miniboxed method calling another
    - data uses miniboxing encoding
  - three patterns
    - initiator – starts an optimized trace
    - propagator – propagates it
    - inhibitor – goes back to boxed :(

Full tutorial on the website

How to use it?

- Sbt Configuration
- Guidance
- Website

Introduction

Miniboxing is a compilation scheme that improves the performance of generics in the Scala programming language. The miniboxing transformation is generic enough to be potentially useful for any statically typed language running on one of the Java Virtual Machines, such as Managed X10, Kotlin or Ceylon.

We'll start by following what happens to a generic class, as it gets compiled. Let's take `class C` as an example:

```
class C[T](t: T)
```

After compiling this class to Java Virtual Machine bytecode, under the erasure transformation one would get bytecode which roughly corresponds to:

```
class C {
   var t: Object = _          // field
   def C(t: Object): C = {    // constructor
     this.t = t
   }
 }
```

As you can see, erasure transformed `t` from a generic value into a pointer to a heap object. While this is perfectly suited for storing a string or another class inside `class C`, it becomes suboptimal when dealing with primitive value types, such as booleans, bytes, integers and floating point numbers.

The reason it's suboptimal is because primitive value types are not heap objects but values passed on the stack, so under the Java Virtual Machine it is common to encode them as heap objects, a process known as boxing.

scala-miniboxing.org

Introduction

Miniboxing is a compilation scheme that improves the performance of generics in the Scala programming language. The miniboxing transformation is generic enough to be potentially useful for any statically typed language running on one of the Java Virtual Machines, such as Managed X10, Kotlin or Ceylon.

We'll start by following what happens to a generic class, as it gets compiled. Let's take `class C` as an example:

```
class C[T](t: T)
```

After compiling this class to Java Virtual Machine bytecode, under the erasure transformation one would get bytecode which roughly corresponds to:

```
class C {
   var t: Object = _          // field
   def C(t: Object): C = {    // constructor
     this.t = t
   }
 }
```

As you can see, erasure transformed `t` from a generic value into a pointer to a heap object. While this is perfectly suited for storing a string or another class inside `class C`, it becomes suboptimal when dealing with primitive value types, such as booleans, bytes, integers and floating point numbers.

The reason it's suboptimal is because primitive value types are not heap objects but values passed on the stack, so under the Java Virtual Machine it is common to encode them as heap objects, a process known as boxing.

scala-miniboxing.org

Introduction

Miniboxing is a compilation scheme that improves the performance of generics in the Scala programming language. The miniboxing transformation is generic enough to be potentially useful for any statically typed language running on one of the Java Virtual Machines, such as Managed X10, Kotlin or Ceylon.

We'll start by following what happens to a generic class, as it gets compiled. Let's take `class C` as an example:

```
class C[T](t: T)
```

After compiling this class to Java Virtual Machine bytecode, under the erasure transformation one would get bytecode which roughly corresponds to:

```
class C {
  var t: Object = _          // field
  def C(t: Object): C = {    // constructor
    this.t = t
  }
}
```

As you can see, erasure transformed `t` from a generic value into a pointer to a heap object. While this is perfectly suited for storing a string or another class inside `class C`, it becomes suboptimal when dealing with primitive value types, such as booleans, bytes, integers and floating point numbers.

The reason it's suboptimal is because primitive value types are not heap objects but values passed on the stack, so under the Java Virtual Machine it is common to encode them as heap objects, a process known as boxing.

scala-miniboxing.org

tutorials

Introduction

n scheme that improves the performance of generics in the Scala programming
transformation is generic enough to be potentially useful for any statically typed
f the Java Virtual Machines, such as Managed X10, Kotlin or Ceylon.

We'll start by following what happens to a generic class, as it gets compiled. Let's take `class C` as an example:

```
class C[T](t: T)
```

After compiling this class to Java Virtual Machine bytecode, under the erasure transformation one would get bytecode which roughly corresponds to:

```
class C {
  var t: Object = _          // field
  def C(t: Object): C = {    // constructor
    this.t = t
  }
}
```

As you can see, erasure transformed `t` from a generic value into a pointer to a heap object. While this is perfectly suited for storing a string or another class inside `class C`, it becomes suboptimal when dealing with primitive value types, such as booleans, bytes, integers and floating point numbers.

The reason it's suboptimal is because primitive value types are not heap objects but values passed on the stack, so under the Java Virtual Machine it is common to encode them as heap objects, a process known as boxing.

scala-miniboxing.org

tutorials

sbt config

Start here
- Start Page
- Introduction
- Full Tutorial
- Benchmarks

Use it
- Sbt config
- Command line
- Source code
- Issue tracker
- License

Experiment
- Sbt project
- Linked List
- Functions
- Reverse

Discuss
- Mailing List
- Twitter
- News

Introduction

n scheme that improves the performance of generics in the Scala programming
transformation is generic enough to be potentially useful for any statically typed
f the Java Virtual Machines, such as Managed X10, Kotlin or Ceylon.

We'll start by following what happens to a generic class, as it gets compiled. Let's take `class C` as an

After compiling this class to Java Virtual Machine bytecode, under the erasure transformation one would get bytecode which roughly corresponds to:

```
class C {
  var t: Object = _          // field
  def C(t: Object): C = {    // constructor
    this.t = t
  }
}
```

As you can see, erasure transformed `t` from a generic value into a pointer to a heap object. While this is perfectly suited for storing a string or another class inside `class C`, it becomes suboptimal when dealing with primitive value types, such as booleans, bytes, integers and floating point numbers.

The reason it's suboptimal is because primitive value types are not heap objects but values passed on the stack, so under the Java Virtual Machine it is common to encode them as heap objects, a process known as boxing.

scala-miniboxing.org

tutorials

sbt config

examples

scala-miniboxing.org

tutorials

sbt config

examples

support

What is miniboxing?

Why use it?

How to use it?

Benchmarks

Conclusion

scala-miniboxing.org

Linked List

Benchmarks

on the 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 linked list

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

All the things you hate about the library are there!

Benchmarks

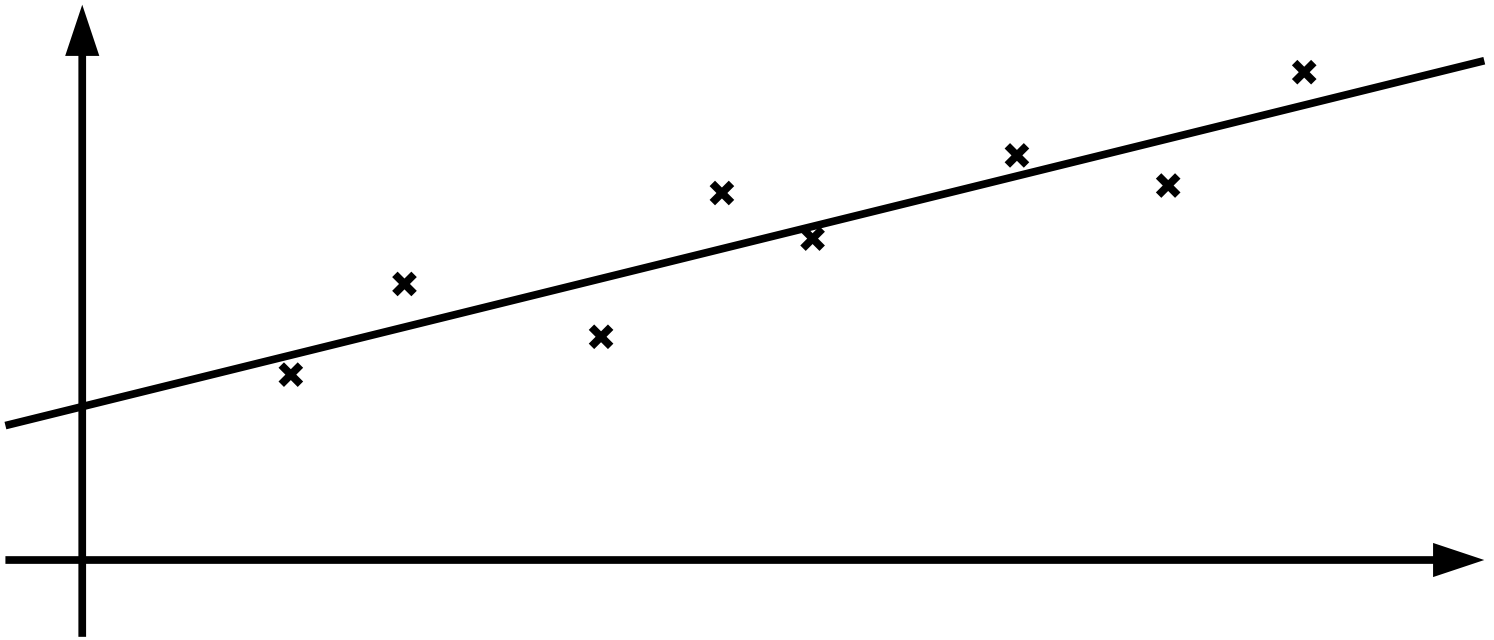
on the linked list

- benchmark: Least Squares Method

Benchmarks

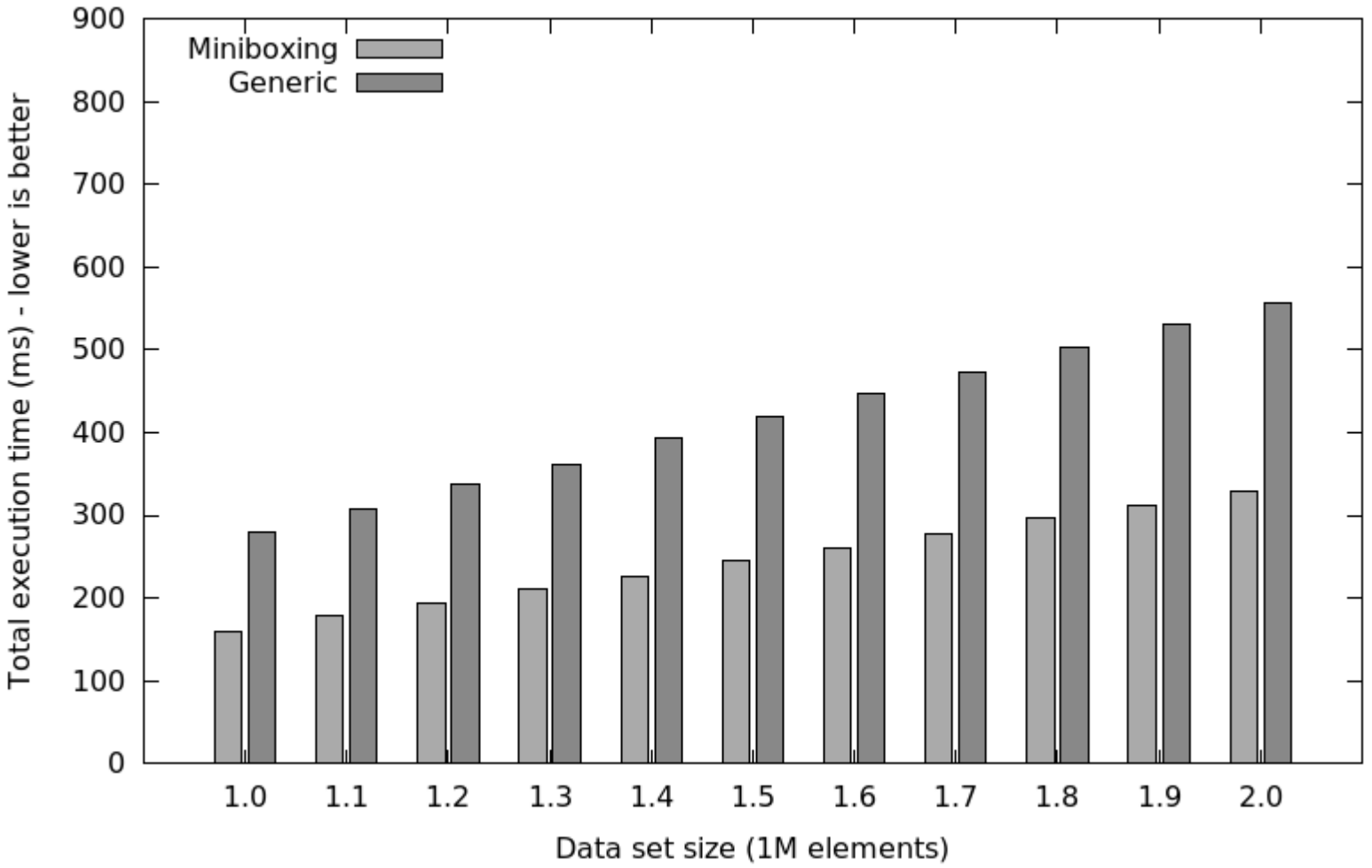
on the linked list

- benchmark: Least Squares Method



Benchmarks

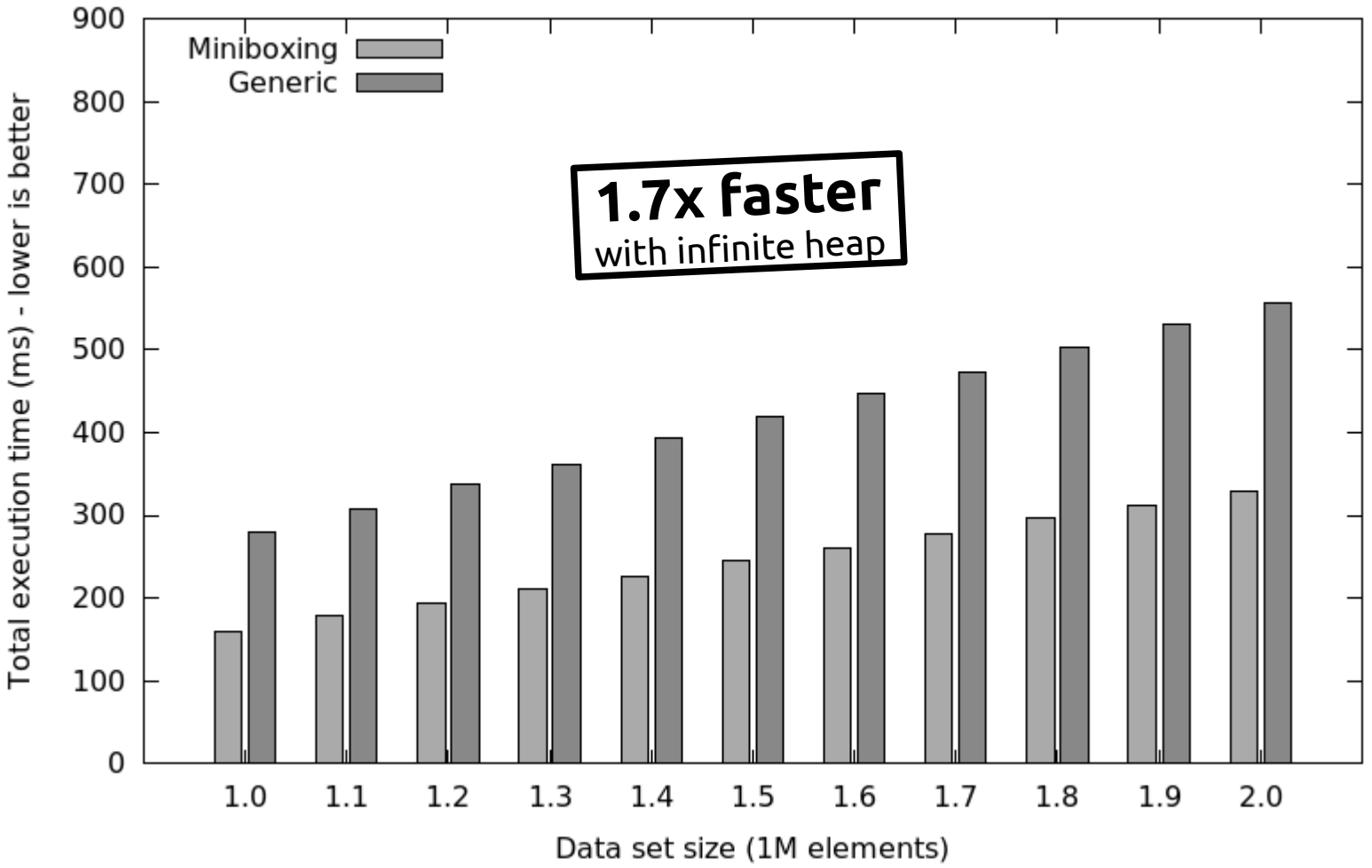
on the linked list (infinite heap)



scala-miniboxing.org

Benchmarks

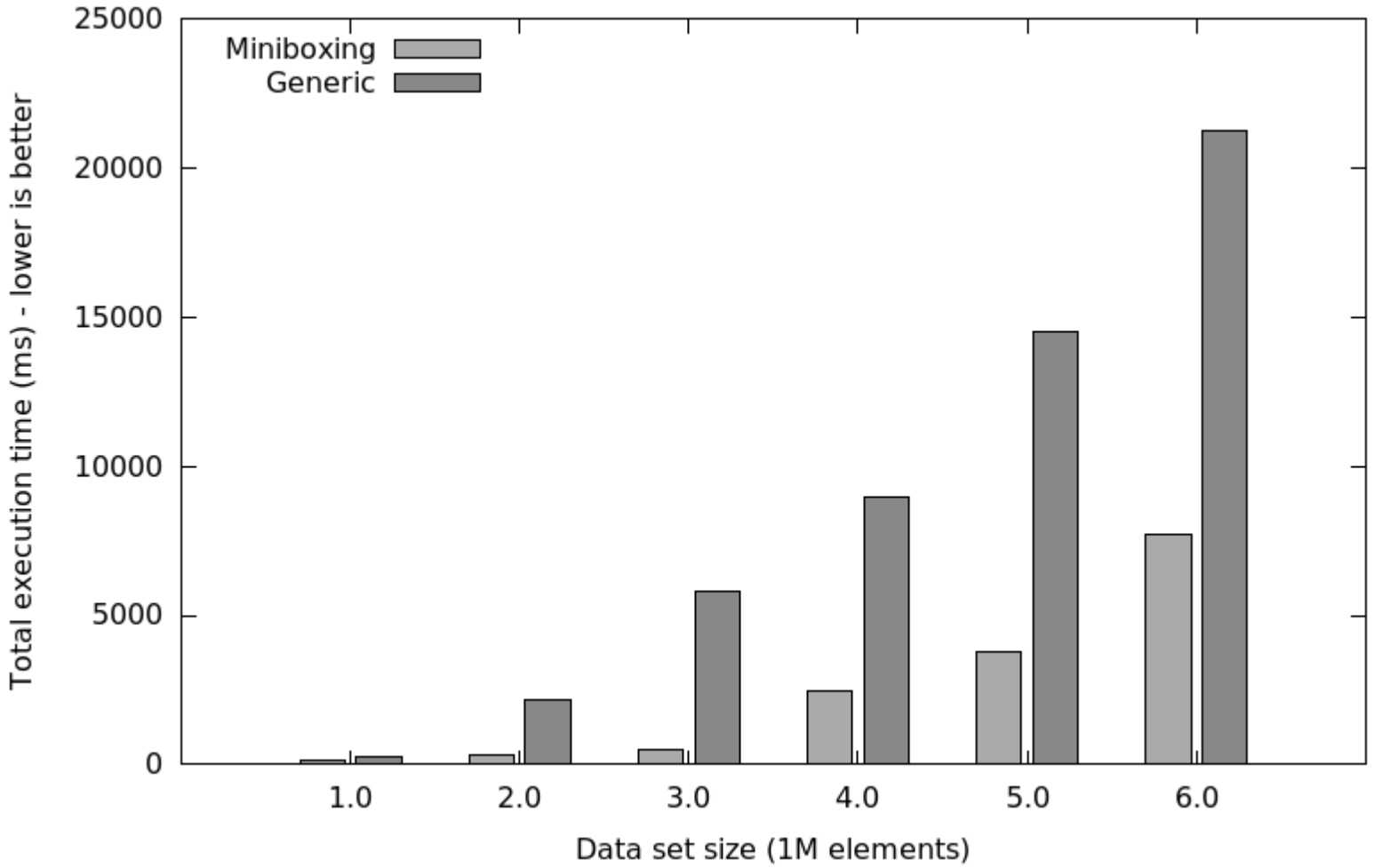
on the linked list (infinite heap)



scala-miniboxing.org

Benchmarks

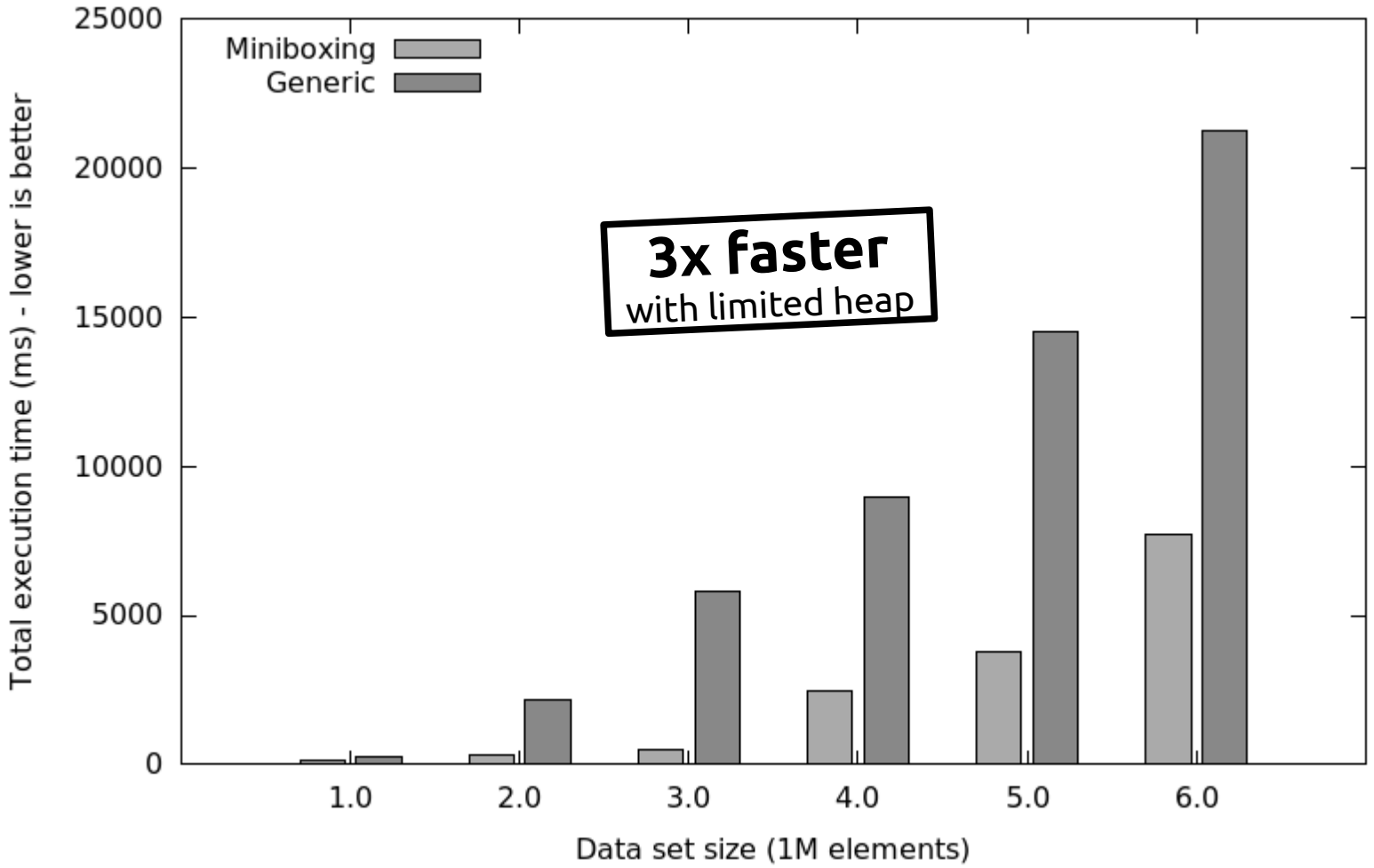
on the linked list (limited heap)



scala-miniboxing.org

Benchmarks

on the linked list (limited heap)

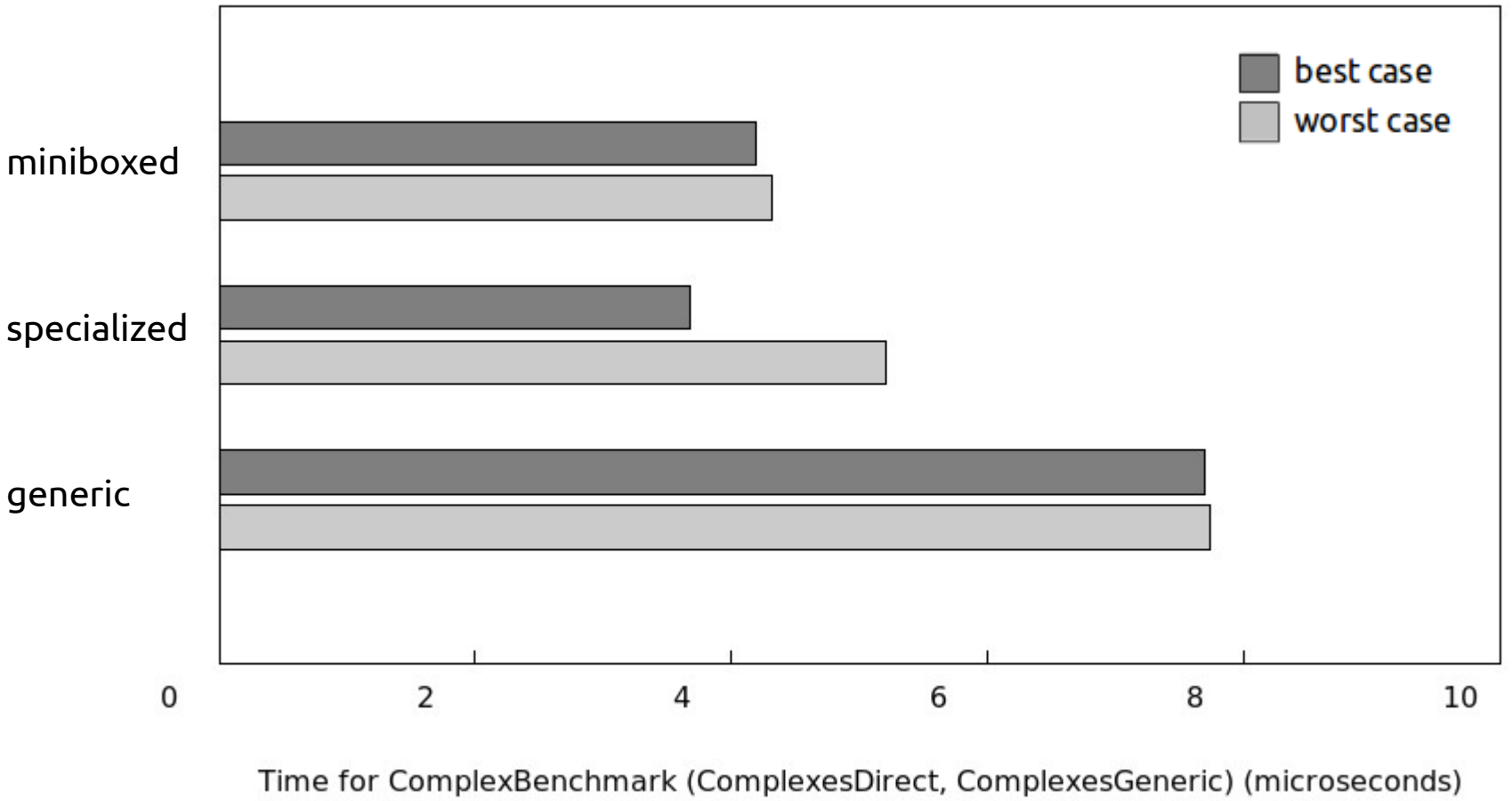


scala-miniboxing.org

non/spire

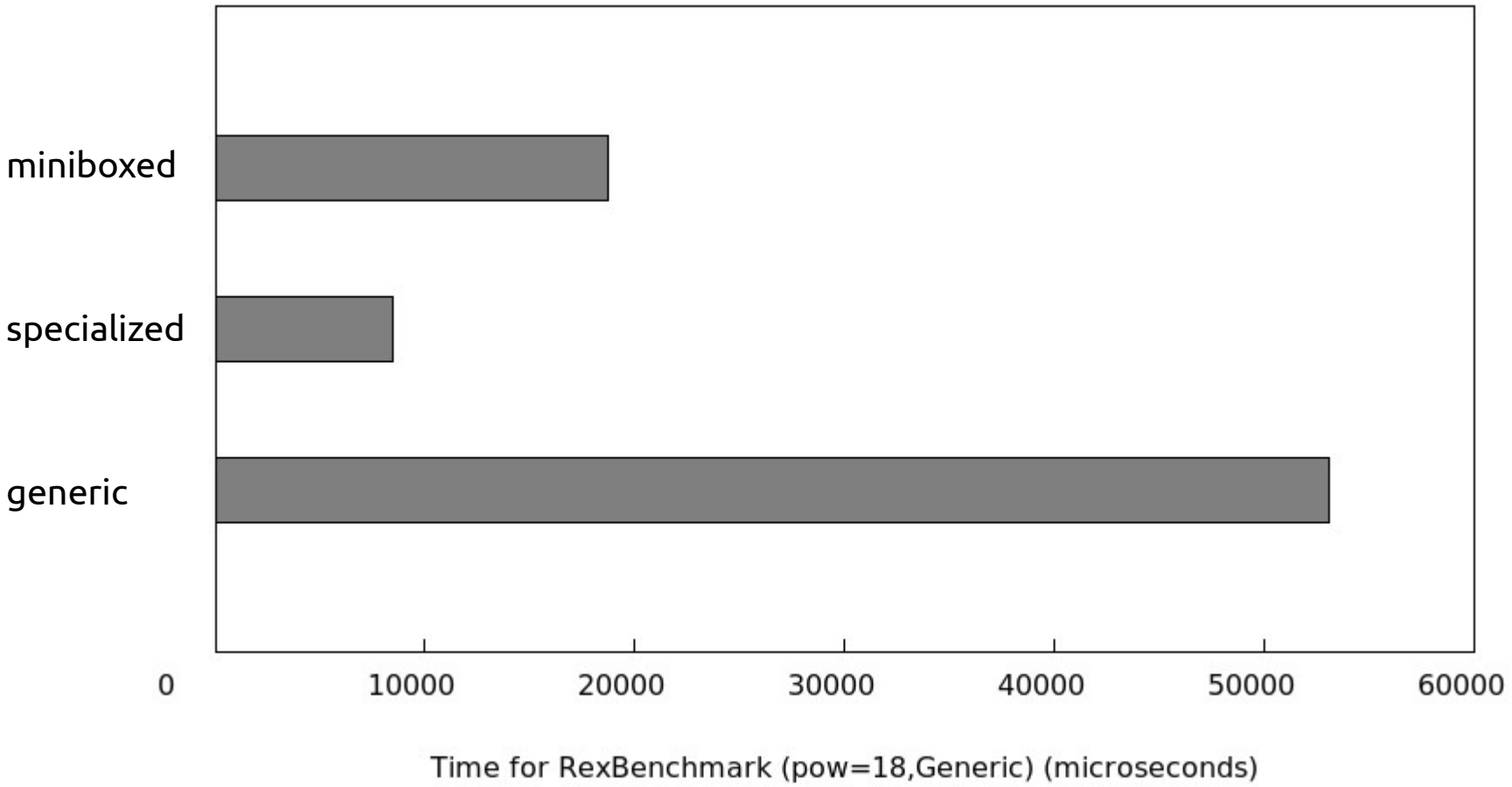
Benchmarks

on the Spire library (Complex)



Benchmarks

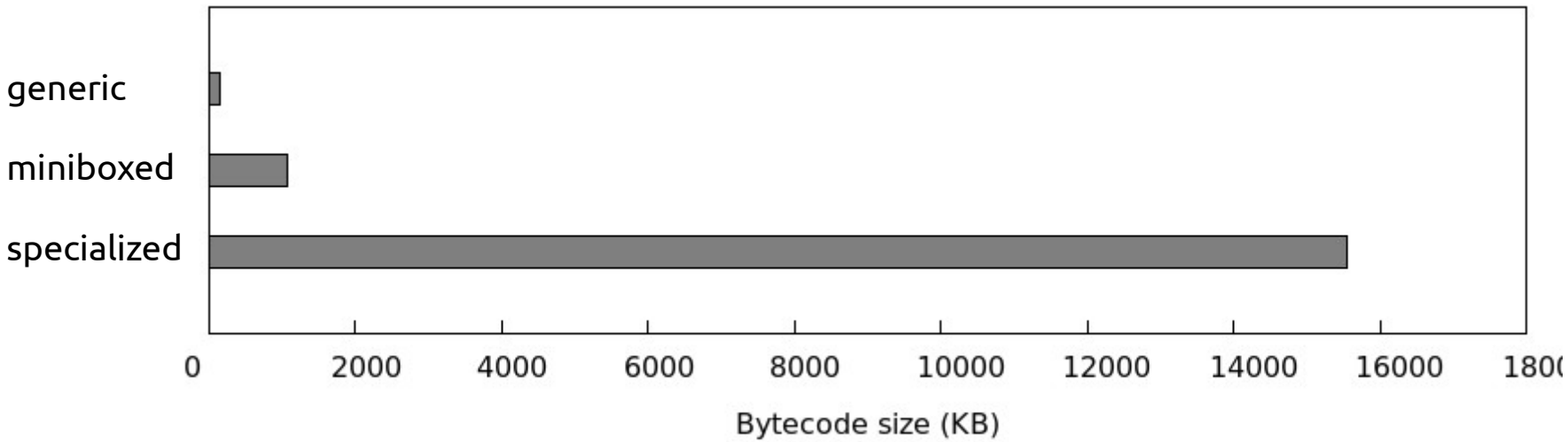
on the Spire library (RexBench)



bytecode

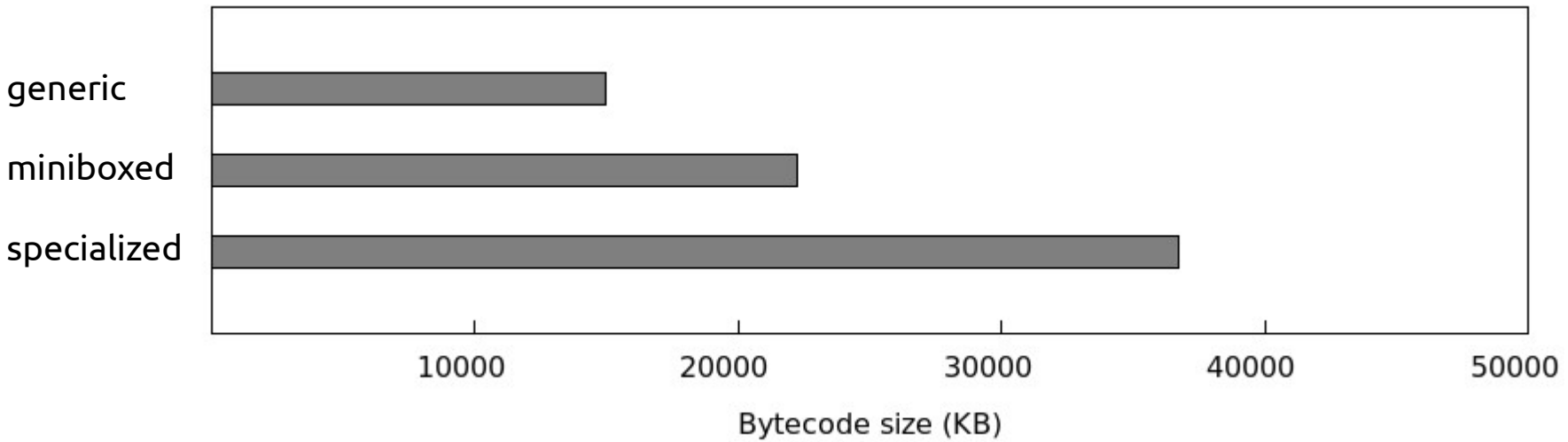
Benchmarks

on the linked list (bytecode)



Benchmarks

on the Spire library (bytecode)



What is miniboxing?

Why use it?

How to use it?

Benchmarks

Conclusion

scala-miniboxing.org

Credits and Thank you-s

- 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
- Dmitry Petrashko, for the many cool discussions we had
- Ilya Klyuchnikov and Pablo Guerrero - fixes and commits
- 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

Special thanks to the Scala Community for their support!
(@StuHood, @vpatryshev and everyone else!)

scala-miniboxing.org

Clipart from www.clipartpanda.com

Miniboxed Encoding

- ScalaDays 2014 Talk
  https://www.parleys.com/play/53a7d2d0e4b0543940d9e567
- OOPSLA '13 Paper
  http://infoscience.epfl.ch/record/188060

Clipart from www.clipartpanda.com

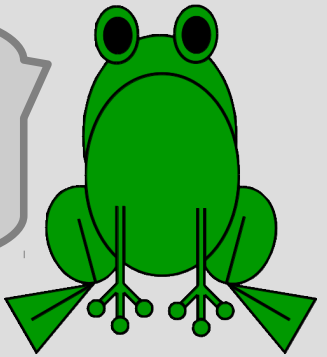
Miniboxed Encoding

- ScalaDays 2014 Talk
  https://www.parleys.com/play/53a7d2d0e4b0543940d9e567
- OOPSLA '13 Paper
  http://infoscience.epfl.ch/record/188060

Code Transformation

- OOPSLA Talk
  https://speakerdeck.com/vladureche/late-data-layout-ooplsa-talk
  (or the Scala Bay talk)
- OOPSLA '14 Paper
  http://infoscience.epfl.ch/record/200963

Related to value classes and multi-stage programming



scala-ldl.org

scala-miniboxing.org

Clipart from www.clipartpanda.com

Miniboxed Encoding

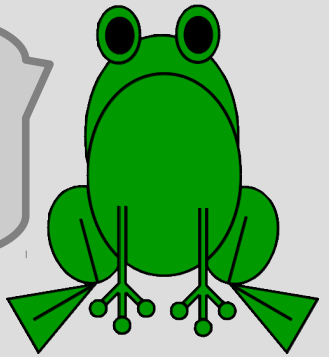
- ScalaDays 2014 Talk
  https://www.parleys.com/play/53a7d2d0e4b0543940d9e567
- OOPSLA '13 Paper
  http://infoscience.epfl.ch/record/188060

Other Considerations

- Function Encoding (Bucharest FP)
  https://github.com/miniboxing/miniboxing-plugin/blob/wip/docs/2014-08-miniboxing-bjug.pdf
- The Quirks of Miniboxing (PDXScala)
  https://www.youtube.com/watch?v=g5yFlQySQ5k

Code Transformation

- OOPSLA Talk
  https://speakerdeck.com/vladureche/late-data-layout-ooplsa-talk
  (or the Scala Bay talk)
- OOPSLA '14 Paper
  http://infoscience.epfl.ch/record/200963



scala-ldl.org

Clipart from www.clipartpanda.com

Miniboxed Encoding

- ScalaDays 2014 Talk
  https://www.parleys.com/play/53a7d2d0e4b0543940d9e567
- OOPSLA '13 Paper
  http://infoscience.epfl.ch/record/188060

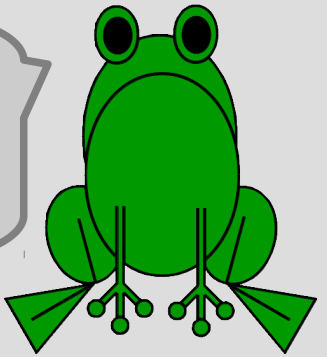
Other Considerations

- Function Encoding (Bucharest FP)
  https://github.com/miniboxing/miniboxing-plugin/blob/wip/docs/2014-08-miniboxing-bjug.pdf
- The Quirks of Miniboxing (PDXScala)
  https://www.youtube.com/watch?v=g5yFlQySQ5k

Code Transformation

- OOPSLA Talk
  https://speakerdeck.com/vladureche/late-data-layout-ooplsa-talk
  (or the Scala Bay talk)
- OOPSLA '14 Paper
  http://infoscience.epfl.ch/record/200963

Related to value classes and multi-stage programming



scala-ldl.org

scala-miniboxing.org

Clipart from www.clipartpanda.com

scala-miniboxing.org

ScalaTeam @ EPFL

lamp.epfl.ch

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

ScalaTeam @ EPFL

- **YinYang** frontend multi-stage execution
  - based on macro transformations
  - Vojin Jovanovic, Sandro Stucki

https://github.com/vjovanov/yin-yang

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

ScalaTeam @ EPFL

- **Scala.js** backend
  - compiles Scala to JavaScript
  - Sébastien Doeraene, Tobias Schlatter

http://www.scala-js.org/

ScalaTeam @ EPFL

- **Lightweight Modular Staging**
  - program optimization
  - Tiark Rompf + pretty much everyone

http://scala-lms.github.io/

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

ScalaTeam @ EPFL

- **Dependent Object Types** calculus
  - core type system of the **dotty** compiler
  - Nada Amin, Tiark Rompf


https://github.com/TiarkRompf/minidot


https://github.com/lampepfl/dotty

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

ScalaTeam @ EPFL

- **Pickling** framework and **Spores**
  - support for **distributed programming**
  - Heather Miller, Philipp Haller + others


https://github.com/scala/pickling


https://github.com/heathermiller/spores

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

ScalaTeam @ EPFL

- **Staged Parser-combinators**
  - fast parser combinators through staging
  - Manohar Jonnalagedda + others

https://github.com/manojo/experiments

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

ScalaTeam @ EPFL

- **dotty** compiler
  - compiler for Scala but with the DOT type system
  - Martin Odersky, Dmitry Petrashko + many others

https://github.com/lampepfl/dotty

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

ScalaTeam @ EPFL

- **scala.meta** metaprogramming support
  - Improved reflection, macros, and many more
  - Eugene Burmako, Denys Shabalin + others

http://scalameta.org/

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

ScalaTeam @ EPFL

- **scaladyno** plugin
  - giving Scala a dynamic language look and feel
  - Cédric Bastin, Vlad Ureche

https://github.com/scaladyno/scaladyno-plugin

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

ScalaTeam @ EPFL

- **miniboxing** specialization
  - LDL transformation
  - Vlad Ureche, Aymeric Genêt + others

www http://scala-miniboxing.org/

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

ScalaTeam @ EPFL

- **ScalaBlitz** optimization framework
  - macro-based collection optimization
  - Dmitry Petrashko, Aleksandar Prokopec

www http://scala-blitz.github.io/

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

ScalaTeam @ EPFL

- **LMS-Kappa** protein simulator
  - using multi-stage programming for performance
  - Sandro Stucki

https://github.com/sstucki/lms-kappa

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

ScalaTeam @ EPFL

- **Odds** probabilistic programming framework
  - using scala-virtualized and macros
  - Sandro Stucki

https://github.com/sstucki/odds

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

ScalaTeam @ EPFL

- **Type debugger** for Scala
  - debugging aid for Scala type errors
  - Hubert Plociniczak

http://lampwww.epfl.ch/~plocinic/type-debugger-tutorial/

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

ScalaTeam @ EPFL

- **ScalaMeter** benchmarking framework
  - google caliper for scala
  - Aleksandar Prokopec

www http://scalameter.github.io/

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

ScalaTeam @ EPFL

- **Vector** implementation using RRB trees
  - improved performance for Scala collections
  - Nicolas Stucki

https://github.com/nicolasstucki/scala-rrb-vector

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

ScalaTeam @ EPFL

lamp.epfl.ch

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE