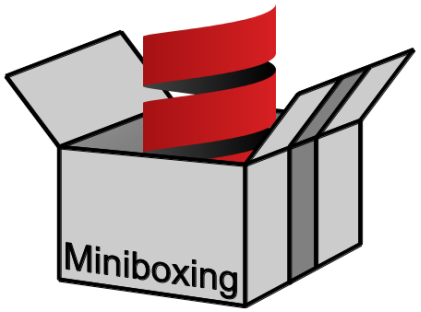


Late Data Layout

or how to boil a frog without it noticing

LAMP, 11th of February 2014

Vlad Ureche
Cristian Talau
Martin Odersky

Frog clipart credits:
www.frog-life-cycle.com

Late Data Layout

or how to boil a frog without it noticing

LAMP, 11th of February 2014

Vlad Ureche
Cristian Talau
Martin Odersky

EPFL

We all like generics

We all like generics

a trivial example

We all like generics

a trivial example

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

We all like generics

a trivial example

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

- will take any type and

We all like generics

a trivial example

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

- will take any type and
- will return **that same type**

We all like generics

a trivial example

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

but under **erasure:**

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

We all like generics

a trivial example

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

but under **erasure:**

Any indicates a by-reference parameter

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

Specialization

generates too much code

Specialization

generates too much code

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

Specialization

generates too much code

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

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

Specialization

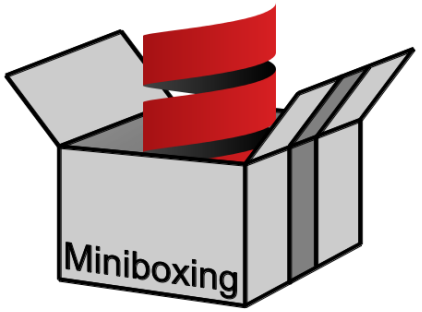
generates too much code

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

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

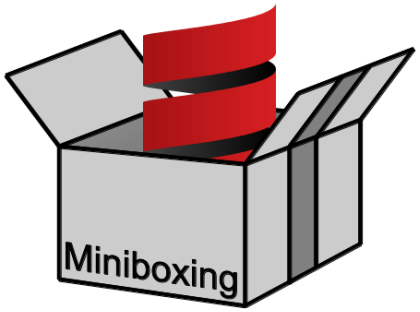
Generates 10 times the original code

Miniboxing

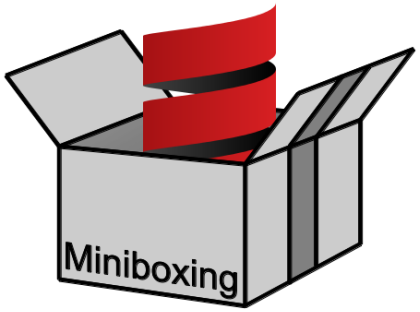


Miniboxing

reduces the variants



Miniboxing



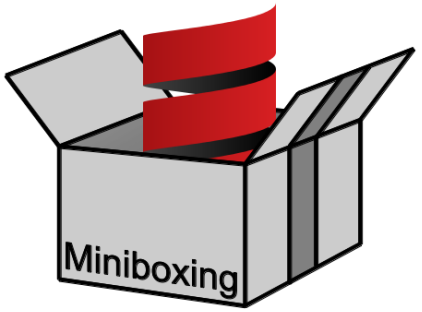
reduces the variants

by using something like a **tagged union**

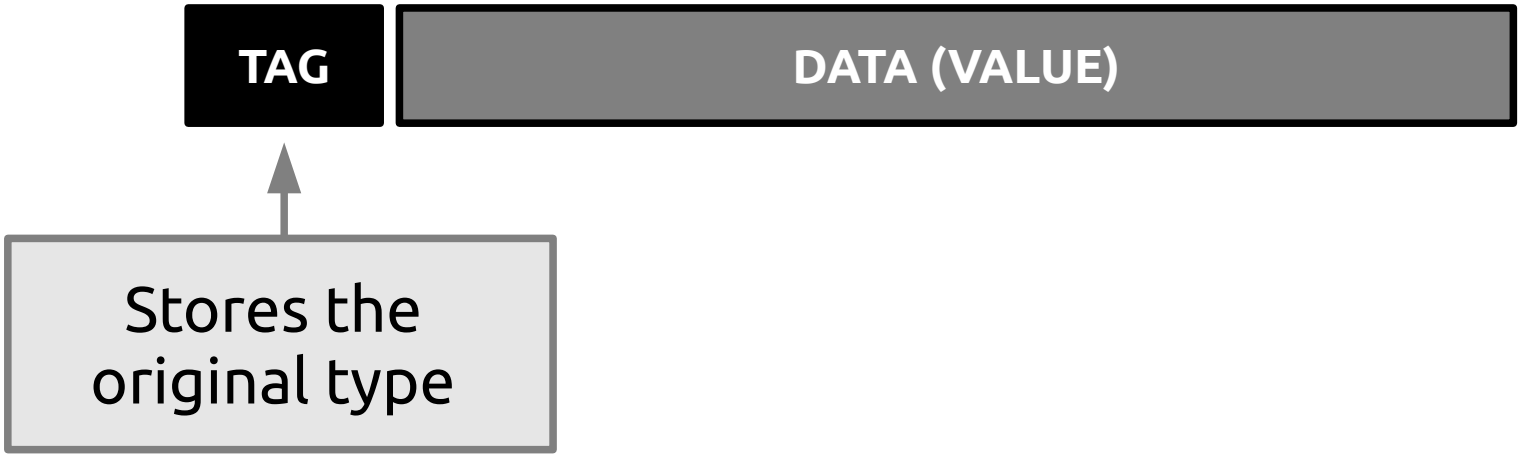
TAG	DATA (VALUE)

Miniboxing

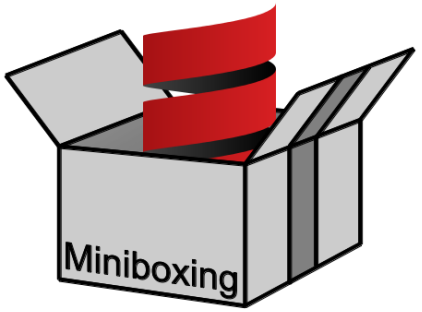
reduces the variants



by using something like a **tagged union**

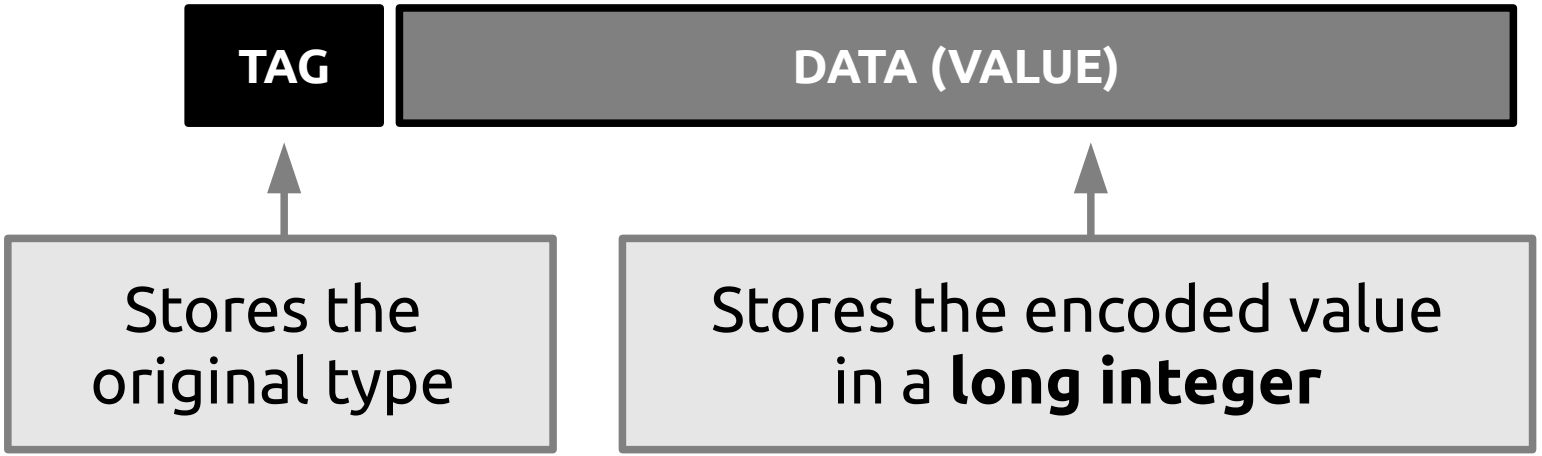


Miniboxing

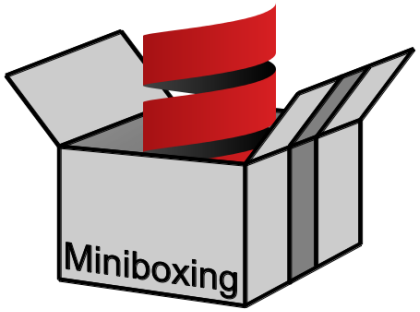


reduces the variants

by using something like a **tagged union**



Miniboxing



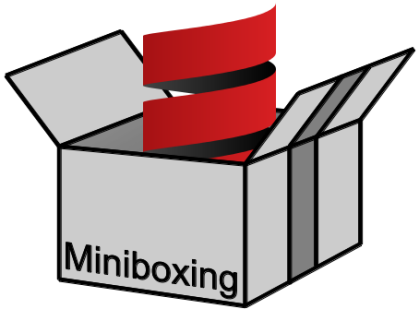
reduces the variants

by using something like a **tagged union**

TAG	DATA (VALUE)

Miniboxing

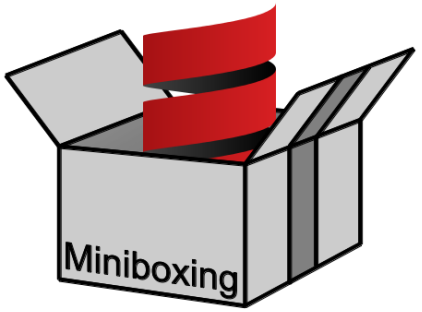
reduces the variants



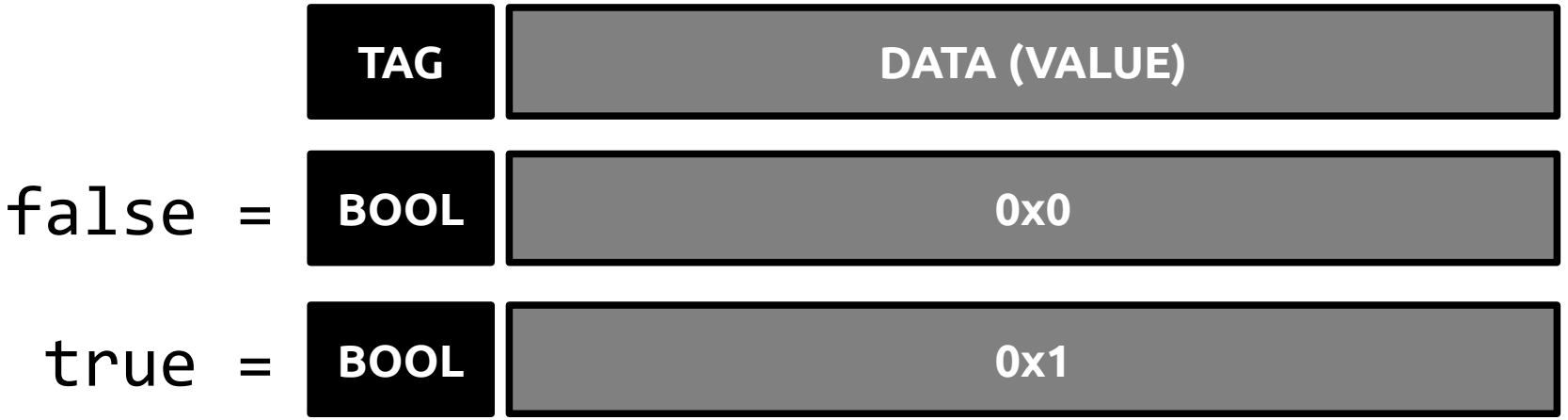
by using something like a **tagged union**

Miniboxing

reduces the variants

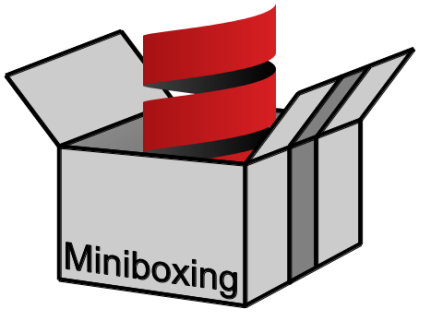


by using something like a **tagged union**



Miniboxing

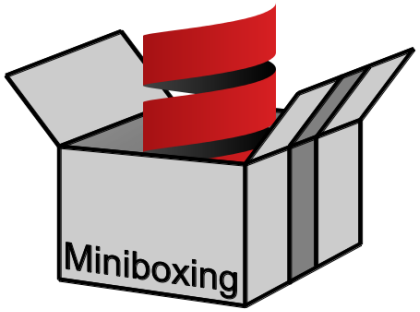
reduces the variants



by using something like a **tagged union**

	TAG	DATA (VALUE)
`false` =	BOOL	0x0
`true` =	BOOL	0x1
`42` =	INT	0x2A

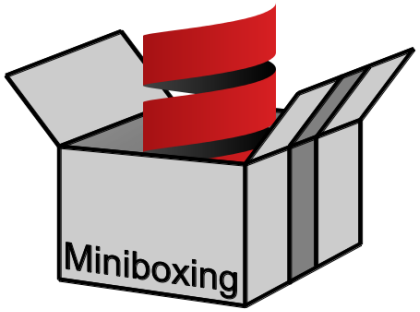
Miniboxing



reduces the variants

by using something like a **tagged union**

Miniboxing



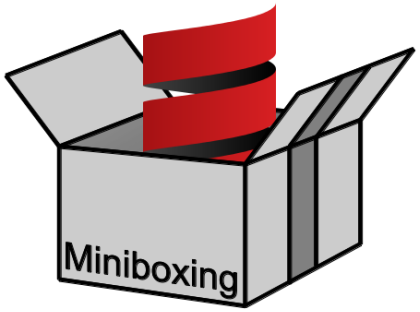
reduces the variants

by using something like a **tagged union**

and using the **static type information**

- tags are attached to **code**, not to values

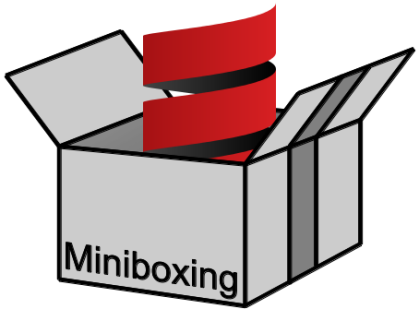
Miniboxing



let's revisit `def identity`

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

Miniboxing

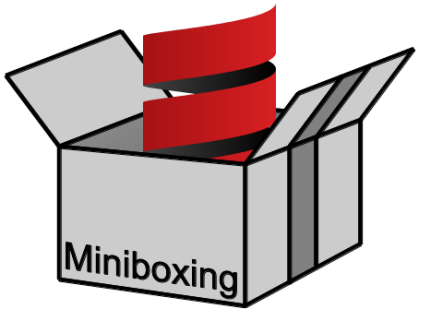


let's revisit `def identity`

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

```
def identity_J(T_tag: Byte, t: Long): Long
```

Miniboxing

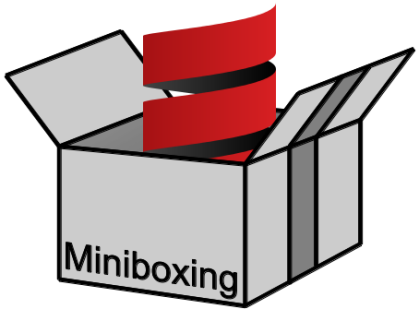


let's revisit `def identity`

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

```
def identity_J(T_tag: Byte, t: Long): Long
```

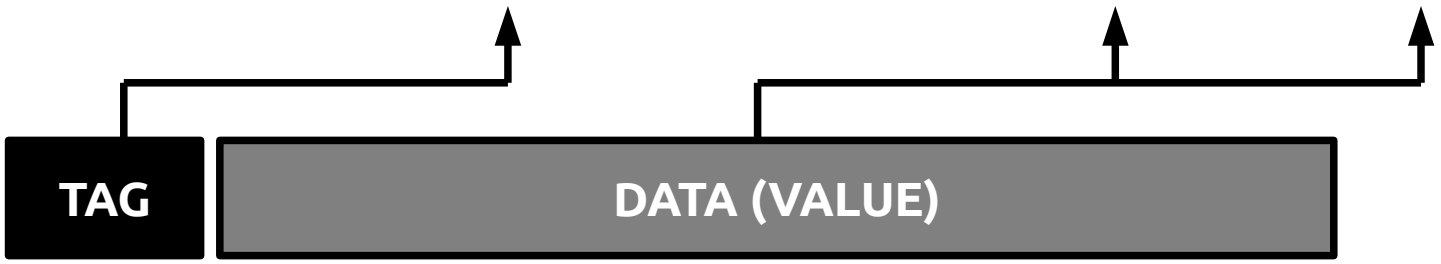
Miniboxing



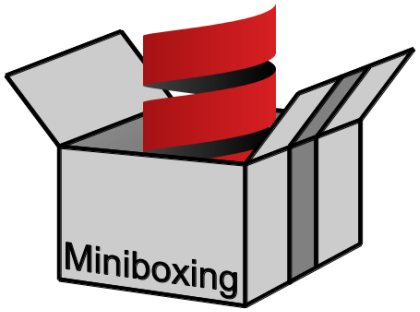
let's revisit `def identity`

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

```
def identity_J(T_tag: Byte, t: Long): Long
```



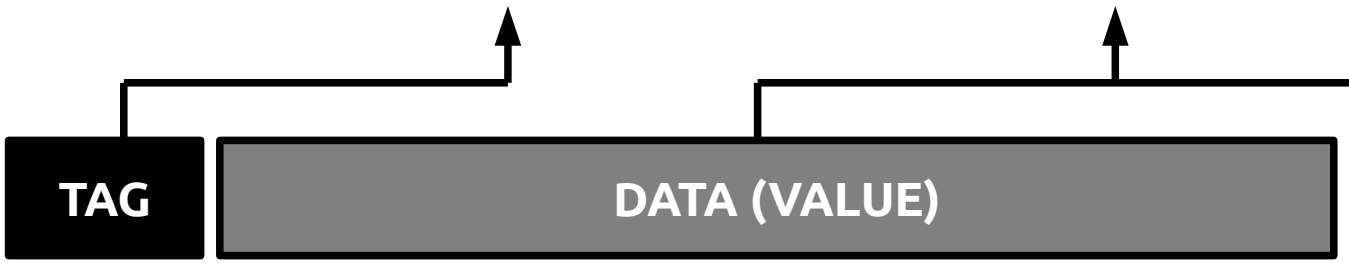
Miniboxing



let's revisit `def identity`

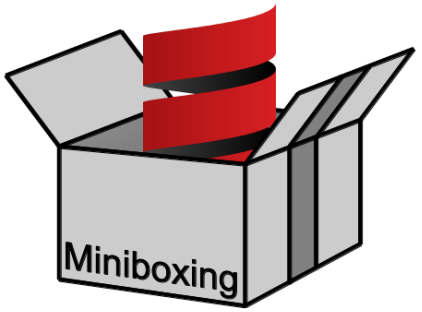
```
def identity[T](t: T): T = t
```

```
def identity_J(T_tag: Byte, t: Long): Long
```



T_tag corresponds to the **type parameter**, instead of the values being passed around.

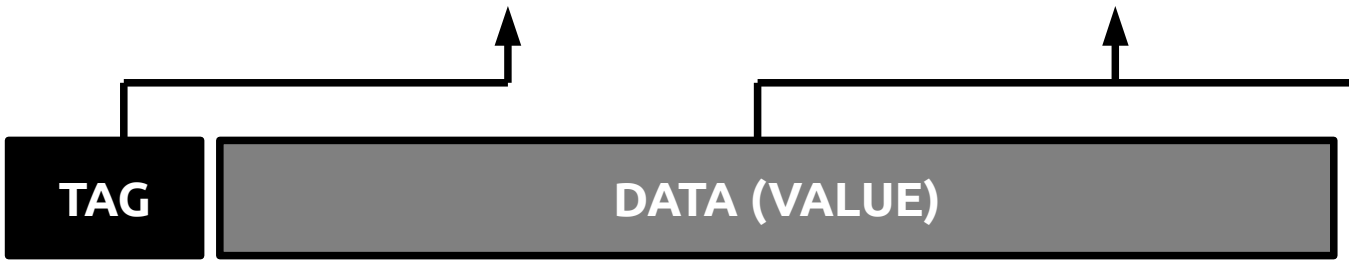
Miniboxing



let's revisit `def identity`

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

```
def identity_J(T_tag: Byte, t: Long): Long
```



T_tag corresponds to the **type parameter**, instead of the value[illegible]und.

Tag hoisting

Transform

WE ARE HERE

Boil the frog

Advantages

Other uses

Conclusion

Transforming

Transforming

could certainly be easier

Transforming

could certainly be easier

```
class C[@miniboxed T] {
  def foo(t: T): T =
    if (...)
      t
    else
      ???
}
```

Transforming

could certainly be easier

```
class C[@miniboxed T] {
  def foo(t: T): T =
    if (...)
      t
    else
      ???
}
```

How to generate `foo_J`, the miniboxed version of `foo`?

Transforming

could certainly be easier

```
def foo_J(t: T): T =
  if (...)
    t
  else
    ???
```

Transforming

could certainly be easier

```
def foo_J(t: T): T =
  if (...)
    t
  else
    ???
```

- change T to Long, one variable at a time

Transforming

could certainly be easier

```
def foo_J(t: T): T =
  if (...)
    t
  else
    ???
```

- change T to Long, one variable at a time
- patch up the tree

Transforming

could certainly be easier

```
def foo_J(t: Long): T =
  if (...)
    t // error: found: Long req'd: T
  else
    ???
```

- change T to Long, one variable at a time
- patch up the tree

Transforming

could certainly be easier

```
def foo_J(t: Long): T =
  if (...)
    minibox2box(T_Tag, t)
  else
    ???
```

- change T to Long, one variable at a time
- patch up the tree

Transforming

could certainly be easier

```
def foo_J(t: Long): T =
  if (...)
    minibox2box(T_Tag, t)
  else
    ???
```

- change T to Long, one variable at a time
- patch up the tree

Transforming

could certainly be easier

```
def foo_J(t: Long): Long =
  if (...)
    minibox2box(T_Tag, t)
  else
    ??? // error: found: T req'd: Long
```

- change T to Long, one variable at a time
- patch up the tree

Transforming

could certainly be easier

```
def foo_J(t: Long): Long =
  box2minibox(T_Tag,
    if (...)
      minibox2box(T_Tag, t)
    else
      ???)
```

- change T to Long, one variable at a time
- patch up the tree

Transforming

could certainly be easier

```
def foo_J(t: Long): Long =
  box2minibox(T_Tag,
    if (...)
      minibox2box(T_Tag, t)
    else
      ???)
```

Transforming

could certainly be easier

```
def foo_J(t: Long): Long =
  box2minibox(T_Tag,
    if (...)
      minibox2box(T_Tag, t)
    else
      ???)
```

- performance?
  - we'd be better off boxing everywhere

Peephole optimization

to save the day

```
def foo_J(t: Long): Long =
  box2minibox(T_Tag,
    if (...)
      minibox2box(T_Tag, t)
    else
      ???)
```

Peephole optimization

to save the day

```
def foo_J(t: Long): Long =
  box2minibox(T_Tag,
    if (...)
      minibox2box(T_Tag, t)
    else
      ???)
```

- special case for if
  - and blocks, and array operations and and and

Peephole optimization

to save the day

```
def foo_J(t: Long): Long =
    if (...)
      box2minibox(minibox2box(t))
    else
      box2minibox(???)
```

Peephole optimization

to save the day

```
def foo_J(t: Long): Long =
    if (...)
      box2minibox(minibox2box(t))
    else
      box2minibox(???)
```

- special case for nested corecions

Peephole optimization

to save the day

```
def foo_J(t: Long): Long =
    if (...)
      t
    else
      box2minibox(???)
```

Peephole optimization

to save the day

```
def foo_J(t: Long): Long =
    if (...)
      t
    else
      box2minibox(???)
```

- contains too many special cases

Peephole optimization

to save the day

```
def foo_J(t: Long): Long =
    if (...)
      t
    else
      box2minibox(???)
```

- contains too many special cases
- tedious and error-prone to write

Transform

Boil the frog

Advantages

Other uses

Conclusion

WE ARE HERE

Transforming

Transforming

is like boiling a frog

Transforming

is like boiling a frog

- if you put it in hot water
  - it jumps out right away

Transforming

is like boiling a frog

- if you put it in hot water
  - it jumps out right away

```
error:
   found: T
   req'd: Long
```

Transforming

is like boiling a frog

- if you put it in hot water
  - it jumps out right away

```
error:
   found: T
   req'd: Long
```

  - so you need special precautions
    - patching up the tree + peephole optimization

Transforming

is like boiling a frog

```
error:
   found: T
   req'd: Long
```

- if you put it in hot water
  - it jumps out right away
  - so you need special precautions
    - patching up the tree + peephole optimization
- if you put it in cold water
  - it will like it there :)

Transforming

is like boiling a frog

```
error:
   found: T
   req'd: Long
```

- if you put it in hot water
  - it jumps out right away
  - so you need special precautions
    - patching up the tree + peephole optimization
- if you put it in cold water
  - it will like it there :)
  - and then you slowly heat up the water :D

Transforming

is like boiling a frog

```
def foo_J(t: T): T =
  if (...)
    t
  else
    ???
```

Transforming

is like boiling a frog

```
def foo_J(t: T): T =
  if (...)
    t
  else
    ???
```

- you put it in cold water

Transforming

is like boiling a frog

```
def foo_J(t: @storage T): @storage T =
  if (...)
    t
  else
    ???
```

- you put it in cold water

Transforming

is like boiling a frog

```
def foo_J(t: @storage T): @storage T =
  if (...)
    t
  else
    ???
```

- you put it in cold water

T =:= @storage T

Transforming

is like boiling a frog

```
def foo_J(t: @storage T): @storage T =
  if (...)
    t
  else
    ???
```

- you can easily do rewiring
  - `foo(t)` → `foo_J(t)`, no coercions needed

Transforming

is like boiling a frog

```
def foo_J(t: @storage T): @storage T =
  if (...)
    t
  else
    ???
```

Transforming

is like boiling a frog

```
def foo_J(t: @storage T): @storage T =
  if (...)
    t
  else
    ???
```

- then you heat up the water

Transforming

is like boiling a frog

```
def foo_J(t: @storage T): @storage T =
  if (...)
    t
  else
    ???
```

- then you heat up the water

```
T =/= @storage T
```

Transforming

is like boiling a frog

```
def foo_J(t: @storage T): @storage T =
  if (...)
    t
  else
    ???
```

retypecheck
expected type: @storage T

- then you heat up the water

```
T =/= @storage T
```

Transforming

is like boiling a frog

```
def foo_J(t: @storage T): @storage T =
  if (...)
    t
  else
    ???
```

retypecheck
expected type: @storage T

- then you heat up the water

```
T =/= @storage T
```

Transforming

is like boiling a frog

```
def foo_J(t: @storage T): @storage T =
  if (...)
    t
  else
    ???
```

retypecheck
expected type: @storage T **OK**

- then you heat up the water

```
T =/= @storage T
```

Transforming

is like boiling a frog

```
def foo_J(t: @storage T): @storage T =
  if (...)
    t
  else
    ???
```

retypecheck
expected type: @storage T

- then you heat up the water

```
T =/= @storage T
```

Transforming

is like boiling a frog

```
def foo_J(t: @storage T): @storage T =
  if (...)
    t
  else
    ???
```

retypecheck
expected type: @storage T **NO**

- then you heat up the water

```
T =/= @storage T
```

Transforming

is like boiling a frog

```
def foo_J(t: @storage T): @storage T =
  if (...)
    t
  else
    box2minibox(???)
```

retypecheck
expected type: @storage T

- then you heat up the water

T =/= @storage T

Transforming

is like boiling a frog

```
def foo_J(t: @storage T): @storage T =
  if (...)
    t
  else
    box2minibox(???)
```

retypecheck
expected type: @storage T

OK

- then you heat up the water

```
T =/= @storage T
```

Transforming

is like boiling a frog

```
def foo_J(t: @storage T): @storage T =
  if (...)
    t
  else
    box2minibox(???)
```

- then you boil the frog

Transforming

is like boiling a frog

```
def foo_J(t: @storage T): @storage T =
  if (...)
    t
  else
    box2minibox(???)
```

- then you boil the frog

@storage T → Long

Transforming

is like boiling a frog

```
def foo_J(t: Long): Long =
  if (...)
    t
  else
    box2minibox(T_Tag, ???)
```

- then you boil the frog

@storage T → Long

Transforming

is like boiling a frog

- the three steps are:

Transforming

is like boiling a frog

- the three steps are:

```
T =:= @storage T
```

Transforming

is like boiling a frog

- the three steps are:

```
T =:= @storage T
```

```
T =/= @storage T
```

Transforming

is like boiling a frog

- the three steps are:

```
T =:= @storage T
```

```
T =/= @storage T
```

```
@storage T → Long
```

Transform

Boil the frog

Advantages

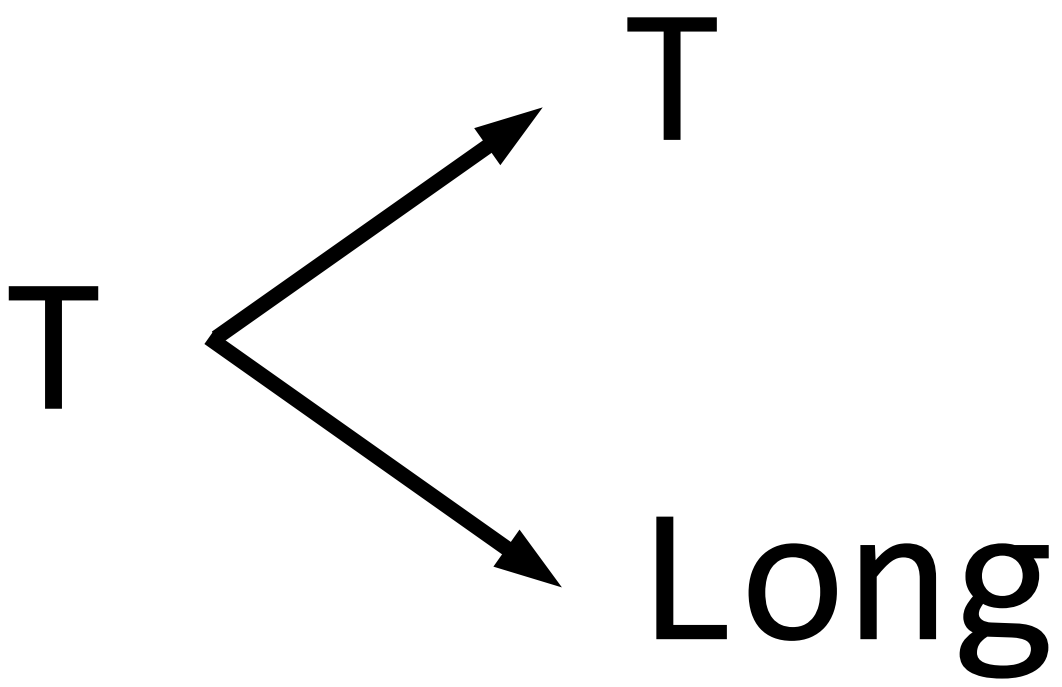
WE ARE HERE

Other uses

Conclusion

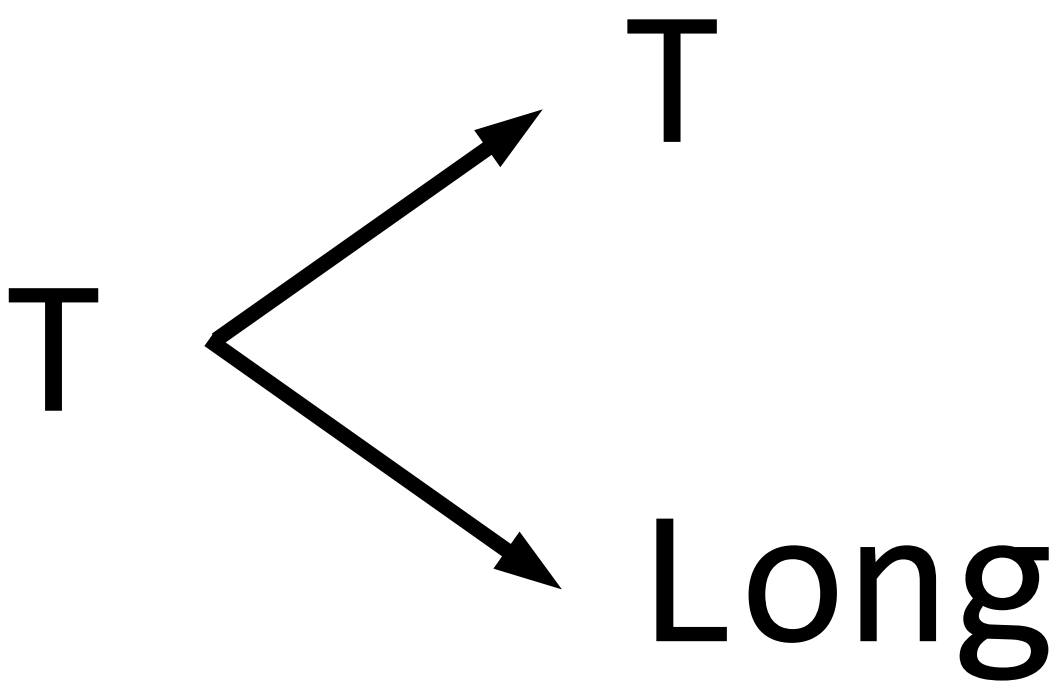
Boiling the frog

Boiling the frog



Boiling the frog

is Late Data Layout



Boiling the frog

is Late Data Layout

Boiling the frog

is Late Data Layout

- choice made by **injecting annotations**
- easy to **rewrite the tree**
- coercions inserted **late** and **on-demand**

Boiling the frog

is Late Data Layout

- choice made by **injecting annotations**
- easy to **rewrite the tree**
- coercions inserted **late** and **on-demand**

```
def bar: Unit = {
  this.foo(...)
  ...
}
```

Boiling the frog

is Late Data Layout

- choice made by **injecting annotations**
- easy to **rewrite the tree**
- coercions inserted **late** and **on-demand**

```
def bar: Unit = {
  this.foo(...)
  ...
}
```

```
this.foo_J(...)
```

Boiling the frog

is Late Data Layout

- choice made by **injecting annotations**
- easy to **rewrite the tree**
- coercions inserted **late** and **on-demand**

```
def bar: Unit = {
  this.foo(...)
  ...
}
```

this.foo_J(...)

No coercion necessary
@storage T <: WildcardType

Transform

Boil the frog

Advantages

Other uses

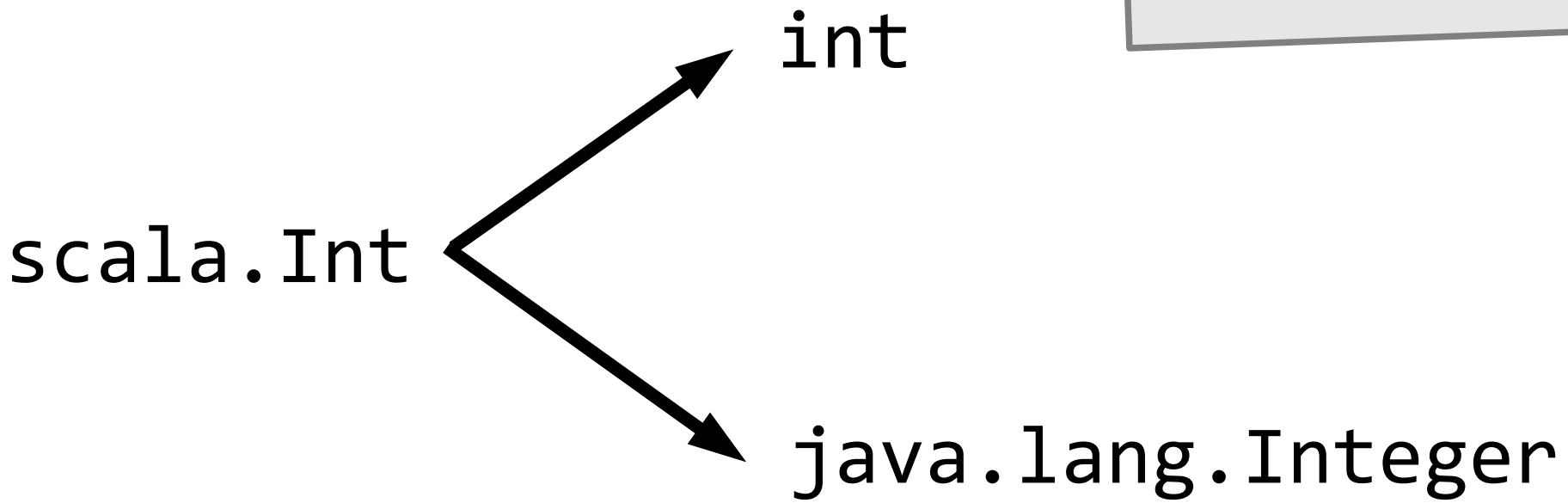
WE ARE HERE

Conclusion

Late Data Layout

is general

Autoboxing

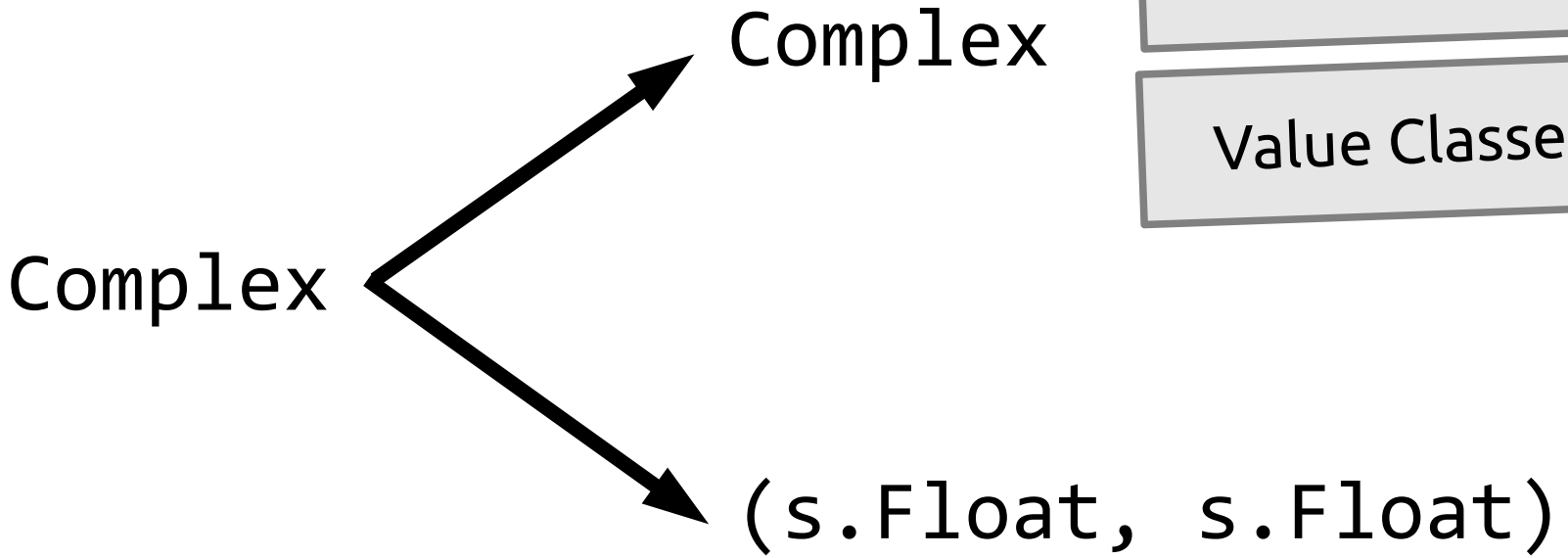


Late Data Layout

is general

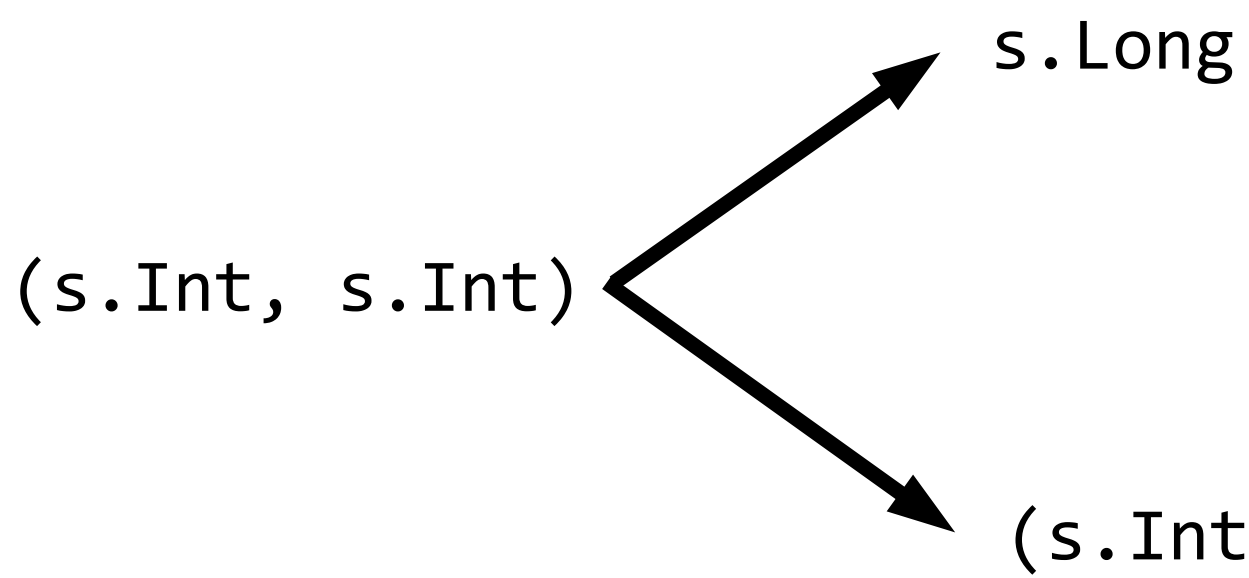
Autoboxing

Value Classes



Late Data Layout

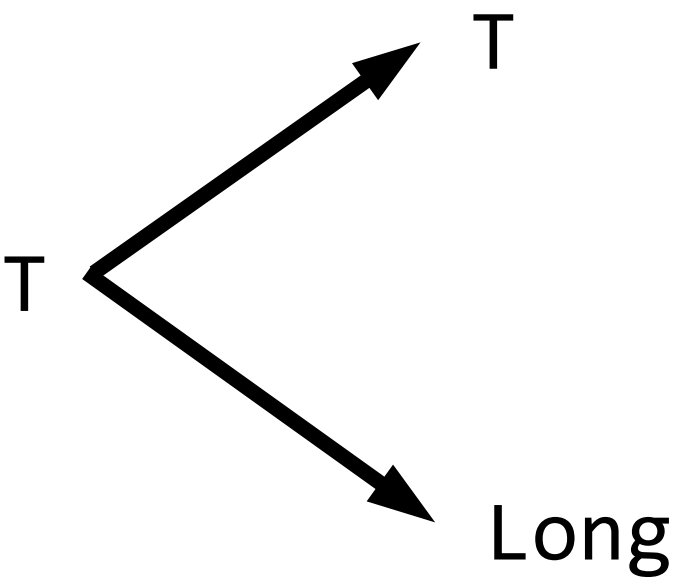
is general



- Autoboxing
- Value Classes
- Encoding

Late Data Layout

is general



- Autoboxing
- Value Classes
- Encoding
- Miniboxing

Transform

Boil the frog

Advantages

Other uses

Conclusion

Conclusion

Late Data Layout

- type-driven transformation
  - heavy-lifting in the type system
- generalization
  - of existing transformations
  - not tied to erasure
- formalization
  - thinking about it now

Autoboxing

Value Classes

Encoding

Miniboxing