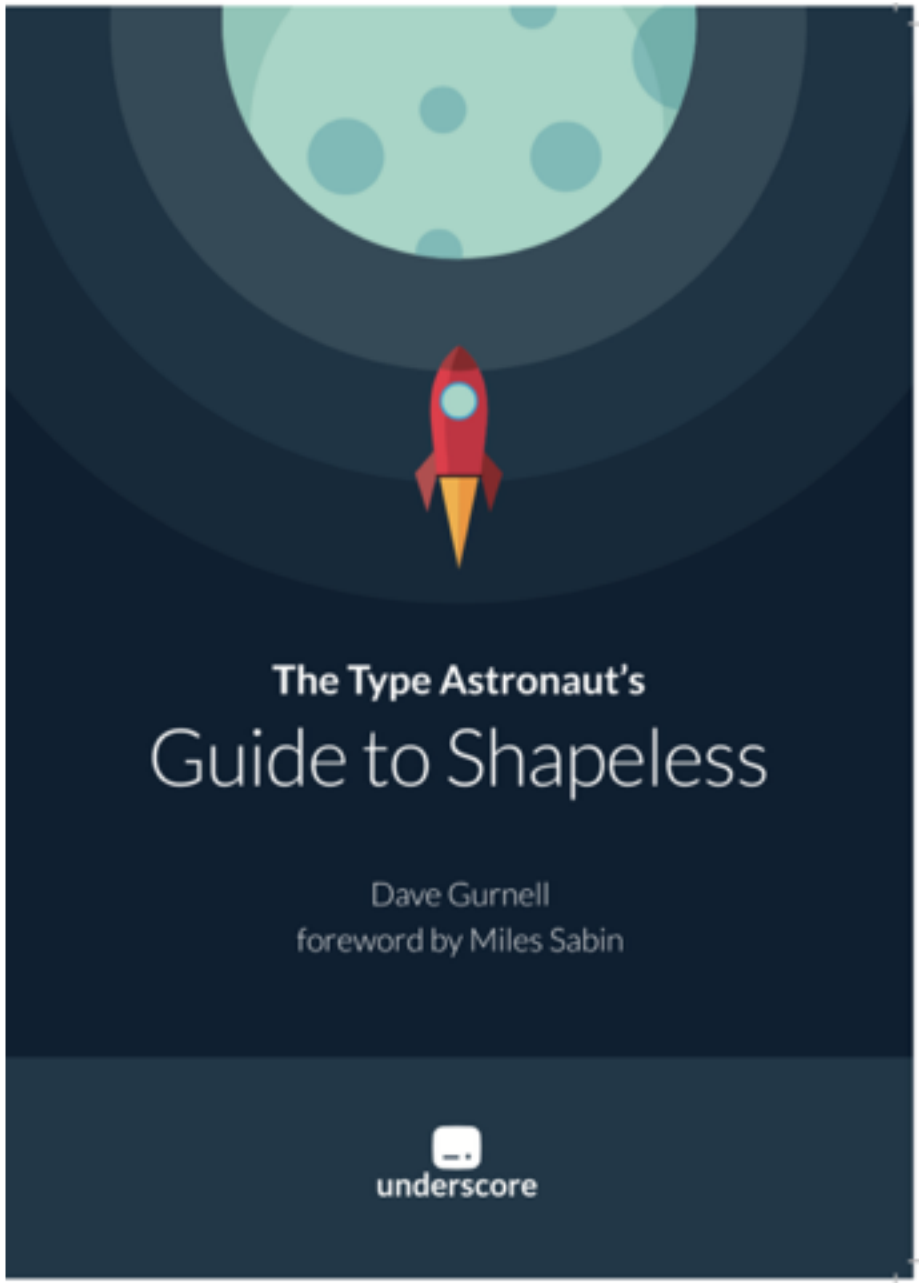
The Type Astronaut's Guide to Shapeless

Dave Gurnell

underscore

Open Source eBook



https://github.com/underscoreio/shapeless-guide

Slides and Examples

eBook Source
https://github.com/underscoreio/shapeless-guide

Slides
https://github.com/davegurnell/shapeless-guide-slides

Example code
https://github.com/underscoreio/shapeless-guide-code

What is Shapeless?

What is Shapeless?

Library for *generic programming*

Created by Miles Sabin in 2011

78 contributors so far

Dependency of >60 libraries

What is
Generic Programming?

Types!

We Like Types!

We Like Types!

They prevent mistakes!

They help us write code!

We Like Types!

They prevent mistakes!

They help us write code!

…because they are specific.

We Like Types!

```
final case class Employee(
  name        : String,
  number      : Int,
  manager     : Boolean
)

final case class IceCream(
  name        : String,
  numCherries : Int,
  inCone      : Boolean
)
```

We Like Types!

```
final case class Employee(
  name        : String,
  number      : Int,
  manager     : Boolean
)

final case class IceCream(
  name        : String,
  numCherries : Int,
  inCone      : Boolean
)
```

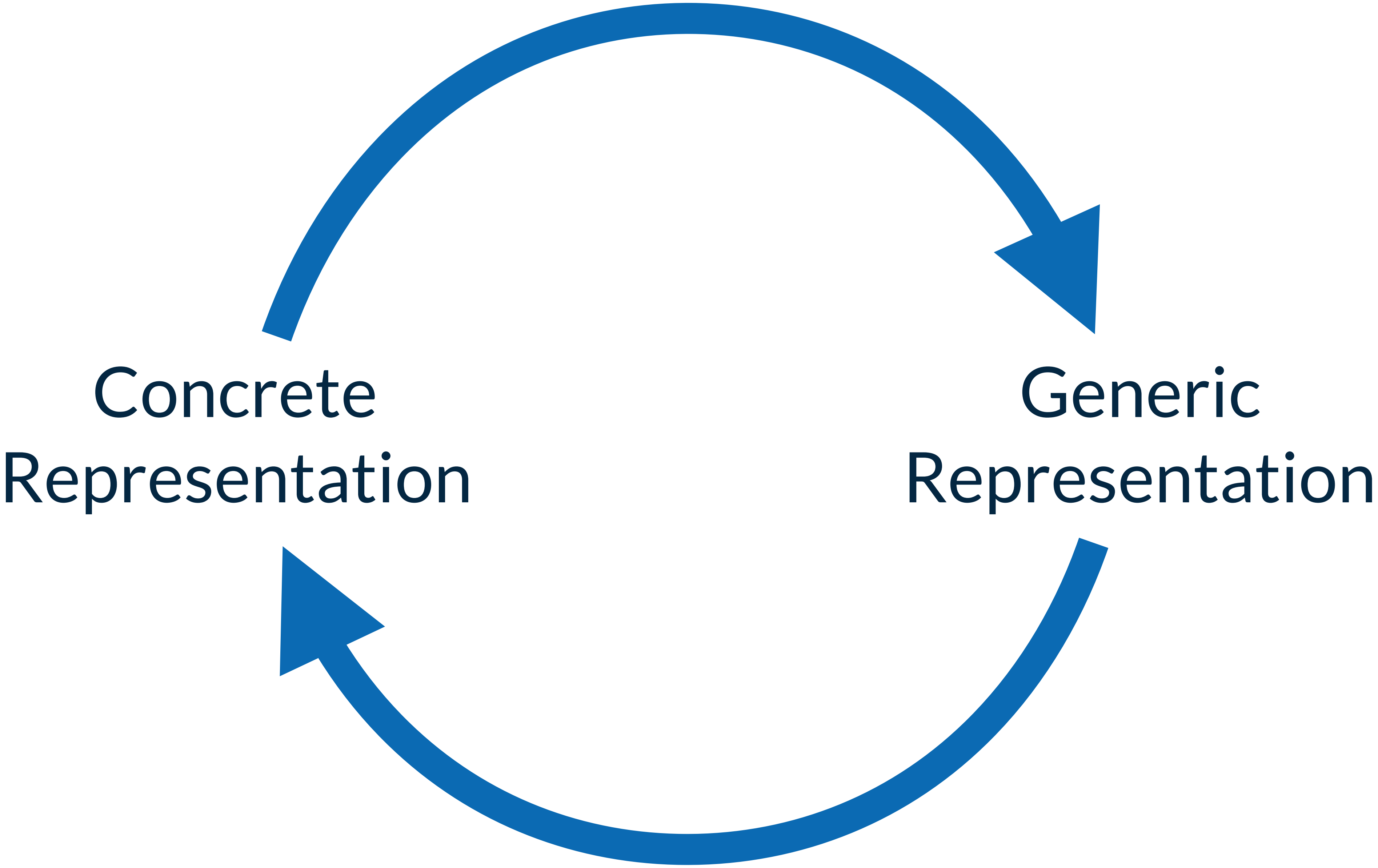
We Like Types!

```
def employeeCsv(e: Employee): List[String] =
  List(
    e.name,
    e.number.toString,
    e.manager.toString
  )

def iceCreamCsv(c: IceCream): List[String] =
  List(
    c.name,
    c.numCherries.toString,
    c.inCone.toString
  )
```

We Like Types???

```scala
def employeeCsv(e: Employee): List[String] =
  List(
    e.name,
    e.number.toString,
    e.manager.toString
  )

def iceCreamCsv(c: IceCream): List[String] =
  List(
    c.name,
    c.numCherries.toString,
    c.inCone.toString
  )
```

The Big Idea



Demo Time!

representations.scala

The Big Idea

Abstract over types…

Abstract over arities…

Eliminate boilerplate…

…write once, run on any type*.

Algebraic Data Types &
Generic Representations

Algebraic Data Types

a shape is a rectangle ***or*** a circle

a rectangle is a width ***and*** a height

a circle is a radius

Algebraic Data Types

Products
case classes / case objects

Coproducts
sealed traits / sealed abstract classes

Algebraic Data Types

```
sealed trait Shape

final case class Rectangle(
  width: Double,
  height: Double
) extends Shape

final case class Circle(
  radius: Double
) extends Shape
```

Algebraic Data Types

```
def area(shape: Shape): Double =
  shape match {
    case Rectangle(w, h) => w * h
    case Circle(r)       => math.Pi * r * r
  }
```

Generic Products

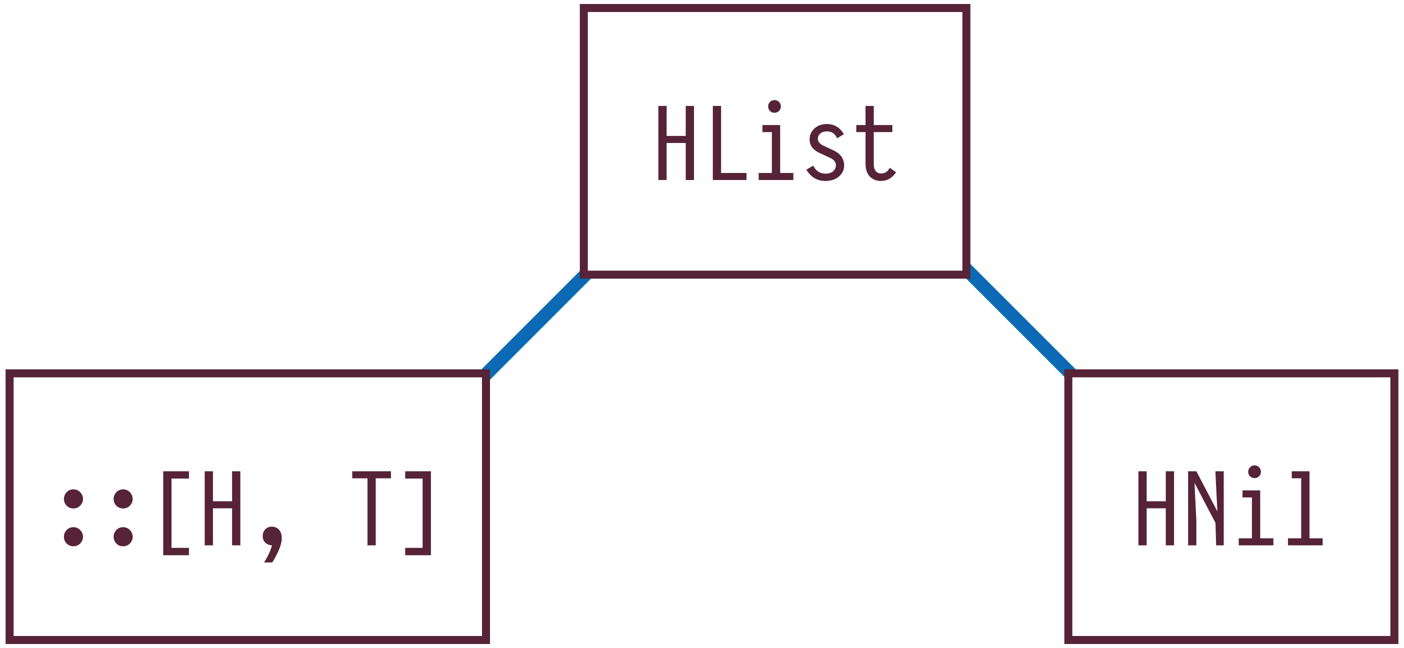
Generic Products

```
case class IceCream(
  name        : String,
  numCherries : Int,
  inCone      : Boolean
)
```

Generic Products?

```
type IceCreamRepr =
  (String, Int, Boolean)
```

Generic Products!



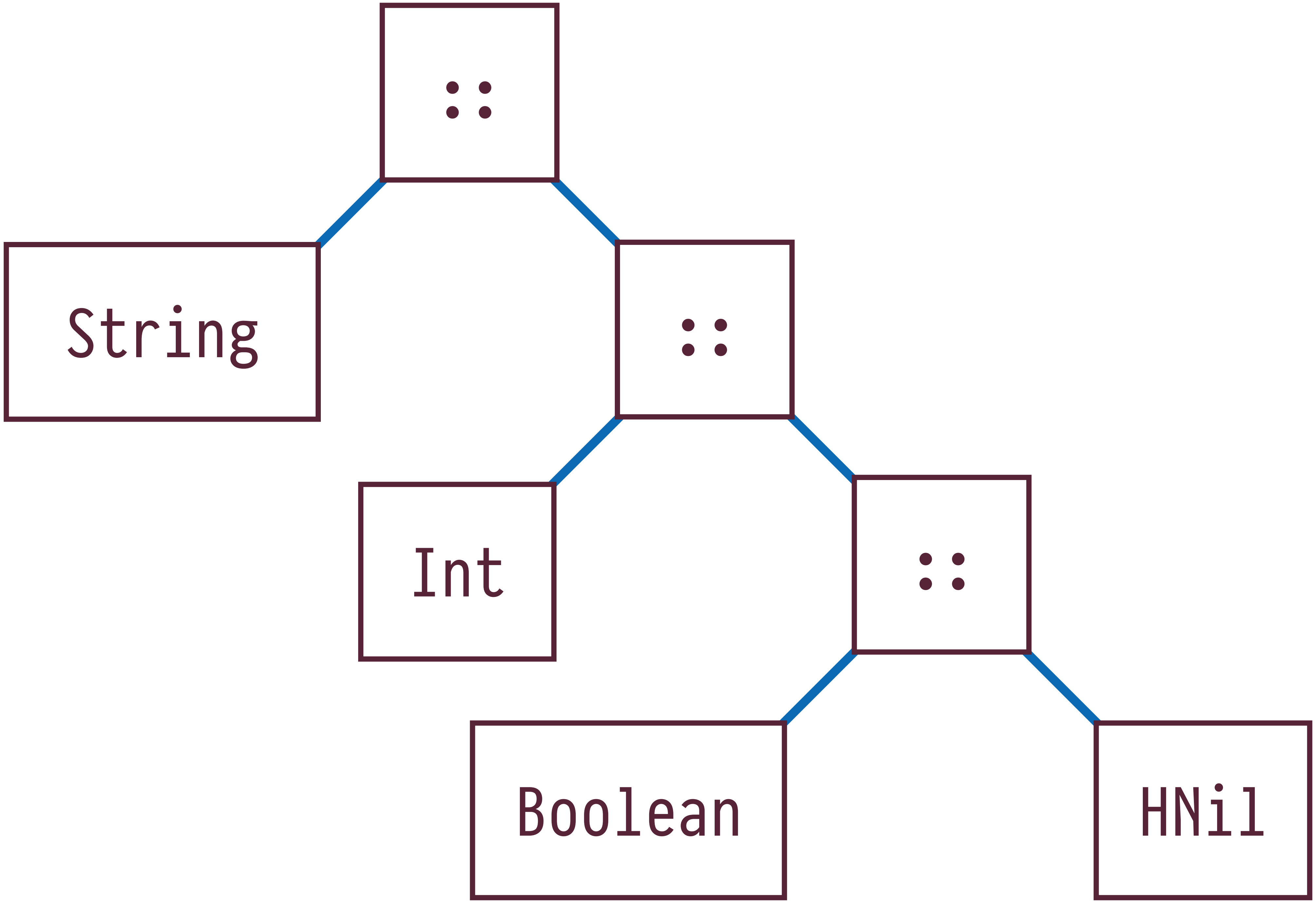
Generic Products

```
import shapeless.{HList, ::, HNil}

type IceCreamRepr =
  String :: Int :: Boolean :: HNil
```

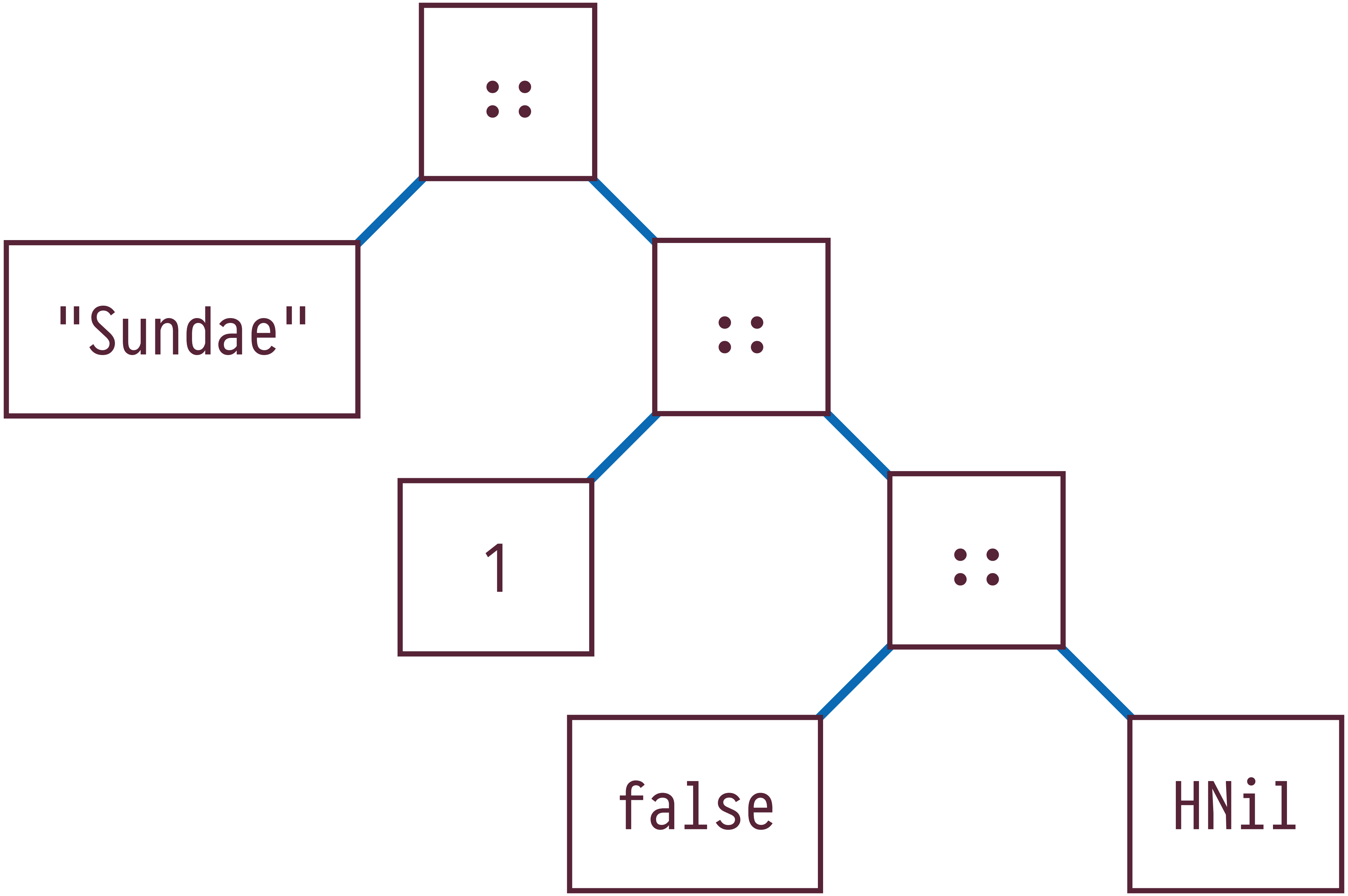
Generic Products

```scala
import shapeless.{HList, ::, HNil}

type IceCreamRepr =
  String :: Int :: Boolean :: HNil

val iceCream: IceCreamRepr =
  "Sundae" :: 1 :: false :: HNil
```

Generic Product Types



Generic Product Values



Demo Time!

representations.scala

Generic Coproducts

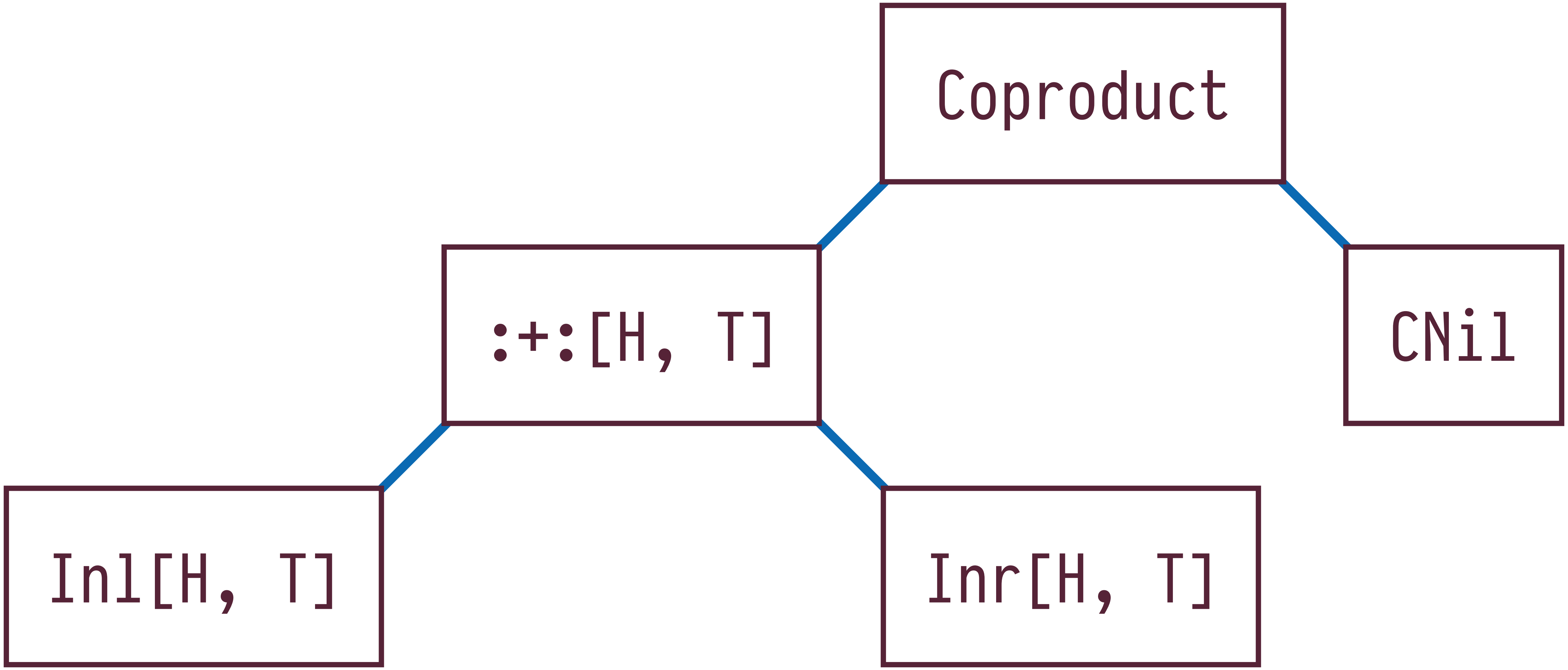
Generic Coproducts

```scala
sealed trait Shape

case class Rectangle(...) extends Shape

case class Circle(...) extends Shape
```

Generic Coproducts?

```
type ShapeRepr =
  Either[Rectangle, Circle]
```

Generic Coproducts!



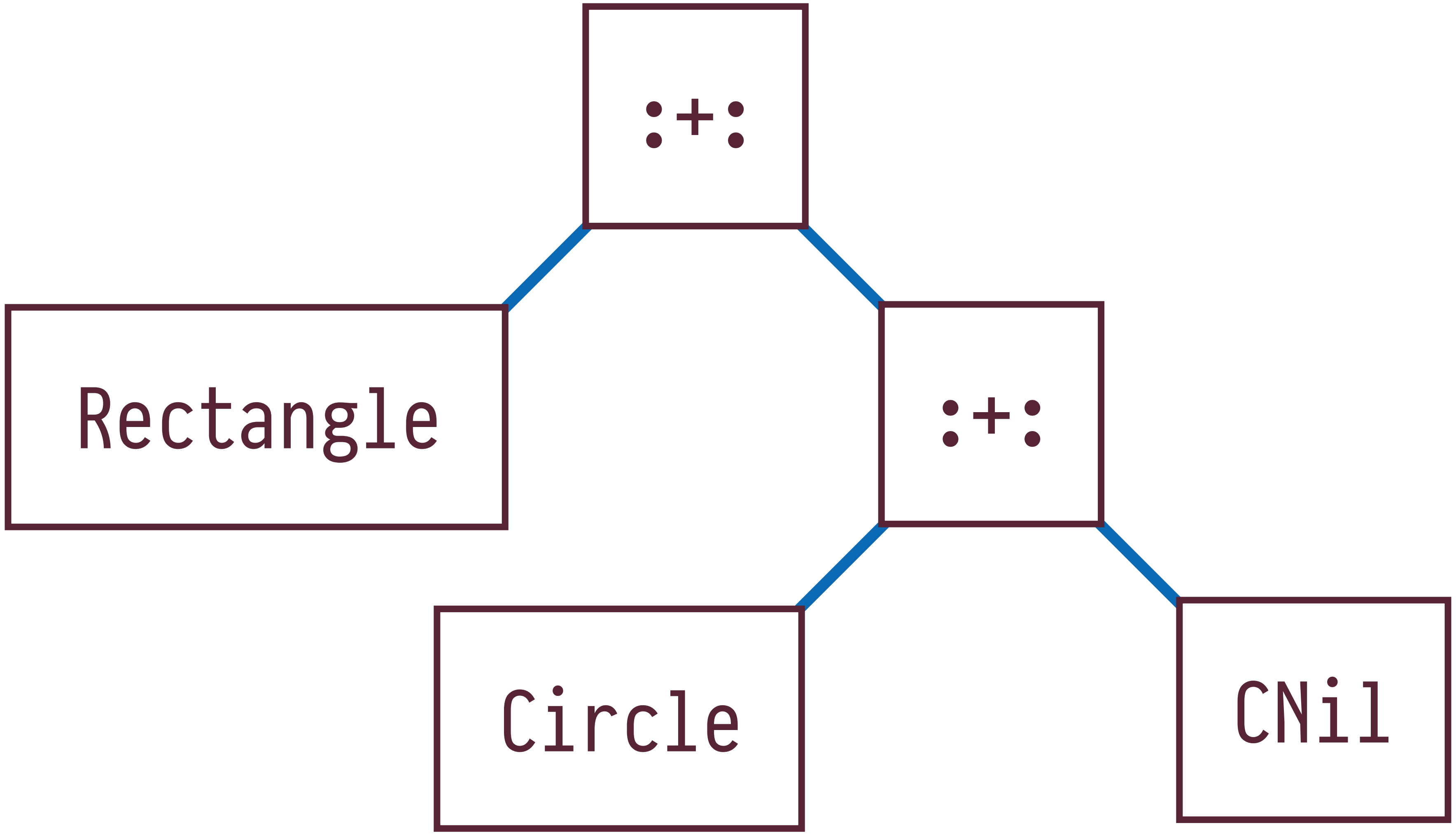
Generic Coproducts

```
import shapeless.{Coproduct, :+:, CNil, Inl, Inr}

type ShapeRepr =
  Rectangle :+: Circle :+: CNil
```

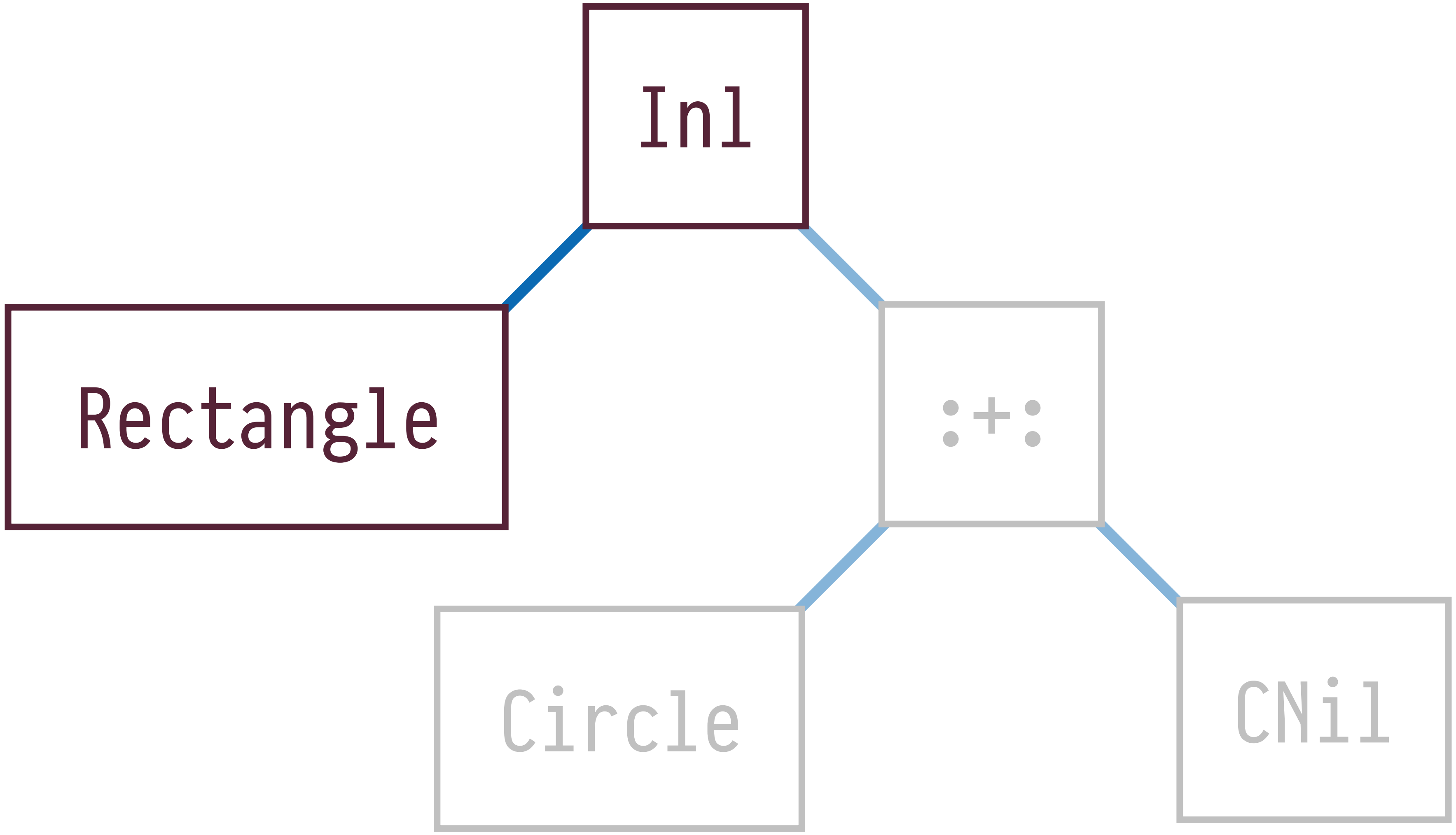
Generic Coproducts

```
import shapeless.{Coproduct, :+:, CNil, Inl, Inr}

type ShapeRepr =
  Rectangle :+: Circle :+: CNil

val shape1: ShapeRepr =
  Inl(Rectangle(1, 2))

val shape2: ShapeRepr =
  Inr(Inl(Circle(1)))
```

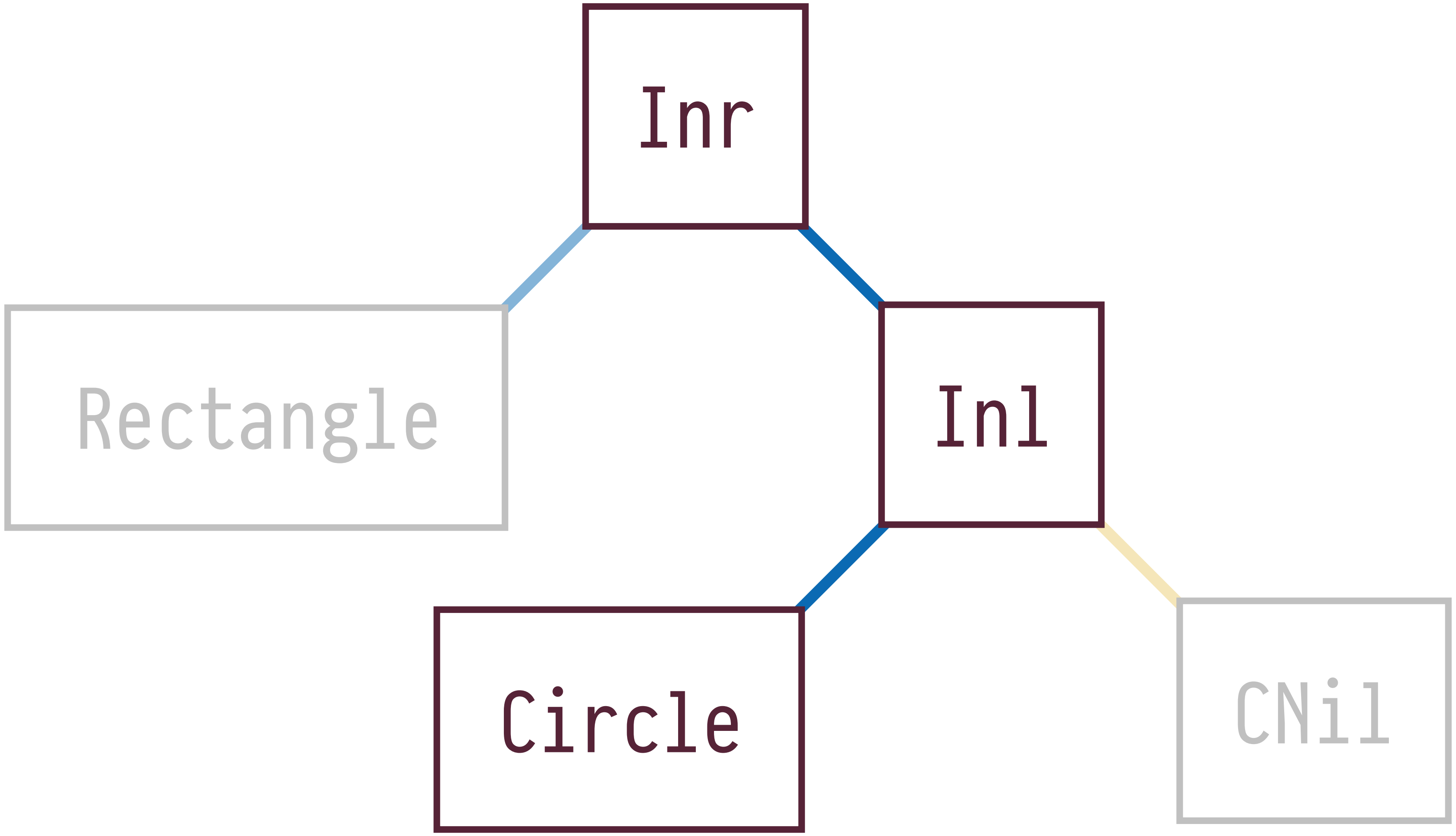
Generic Coproducts



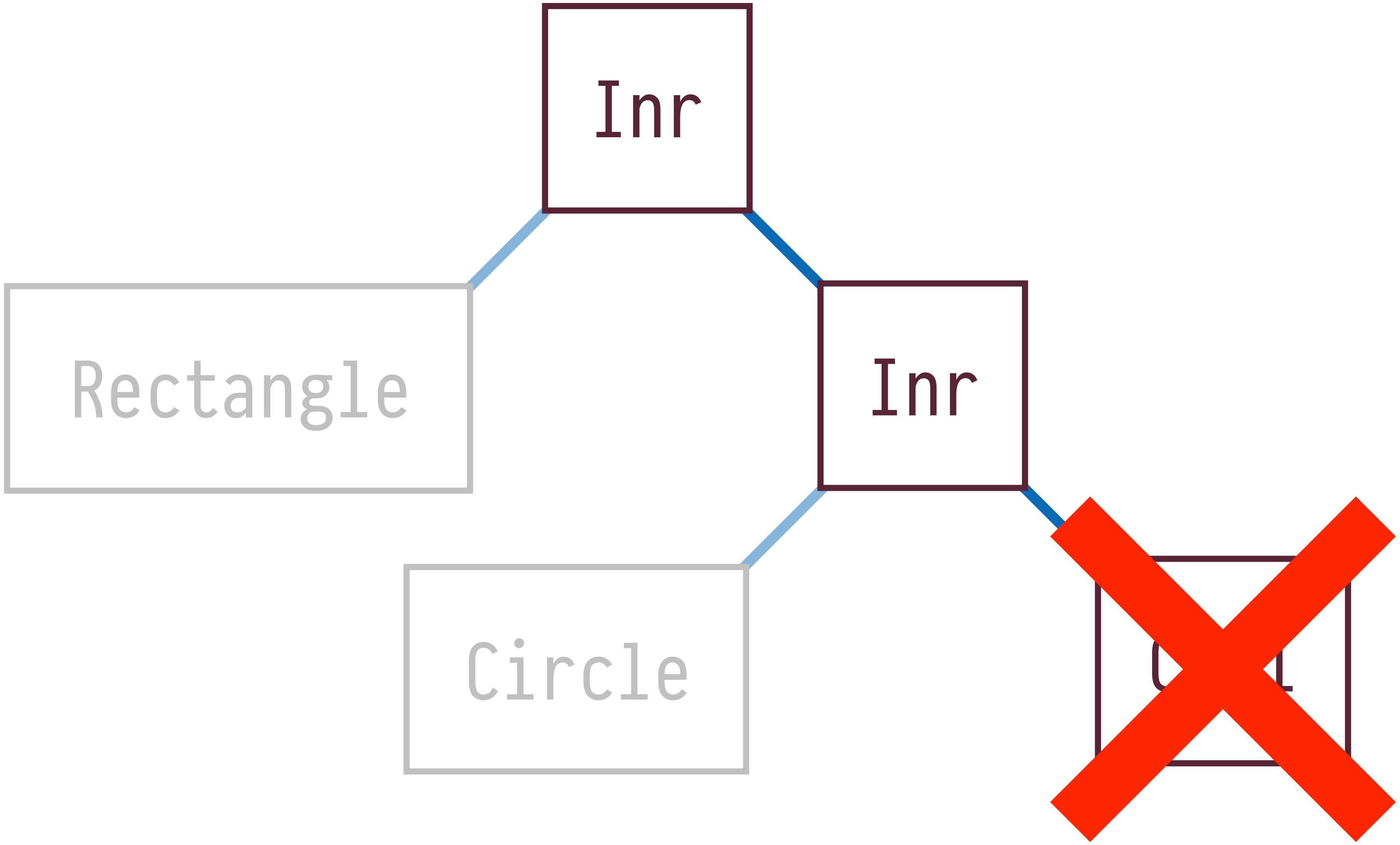
Generic Coproducts



Generic Coproducts



Generic Coproducts



Demo Time!

representations.scala

SI-7046

The *knownDirectSubclasses* bug

Typelevel Scala 2.11.9+
Lightbend Scala 2.12.1+

Any Questions?

Writing Generic Code

Writing Generic Code

```
def encodeCsv[A](value: A): List[String] =
  ???
```

Type Classes

Type Classes

```
trait CsvEncoder[A] {
  def encode(value: A): List[String]
}
```

Type Classes

```
def encodeCsv[A](value: A)(implicit enc: CsvEncoder[A]) =
  ???
```

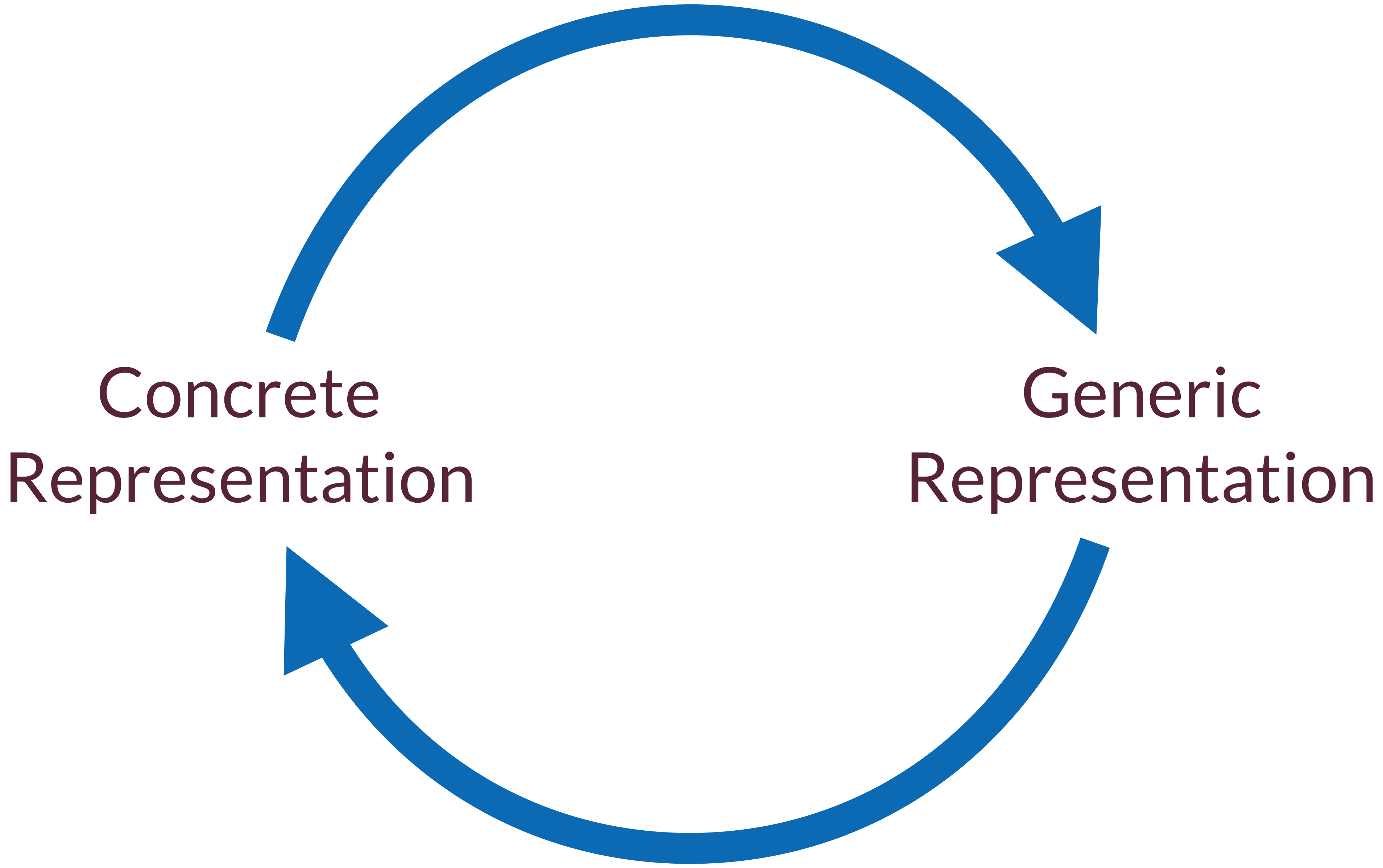
Type Classes

```
implicit val employeeEnc: CsvEncoder[Employee] =
  ???

implicit val iceCreamEnc: CsvEncoder[IceCream] =
  ???
```

Type Classes

```
implicit def pairEnc[A, B](
  implicit
  aEnc: CsvEncoder[A],
  bEnc: CsvEncoder[B]
): CsvEncoder[(A, B)] = ???
```

Automatic
Type Class Derivation



Concrete
Representation

Generic
Representation

Writing Generic Code

```
// Products
implicit val hnilEnc: CsvEncoder[HNil] = ???
implicit def hlistEnc[H, T]: CsvEncoder[H :: T] = ???

// Coproducts
implicit val cnilEnc: CsvEncoder[CNil] = ???
implicit def coprodEnc[H, T]: CsvEncoder[H :+: T] = ???

// Generic
implicit def genericEnc[A: Generic]: CsvEncoder[A] = ???
```

Demo Time!

csv.scala

Any Questions?

Dependent Types

Dependent Types

```
trait Generic[A] {
  type Repr
  def to(a: A): Repr
  def from(repr: Repr): A
}
```

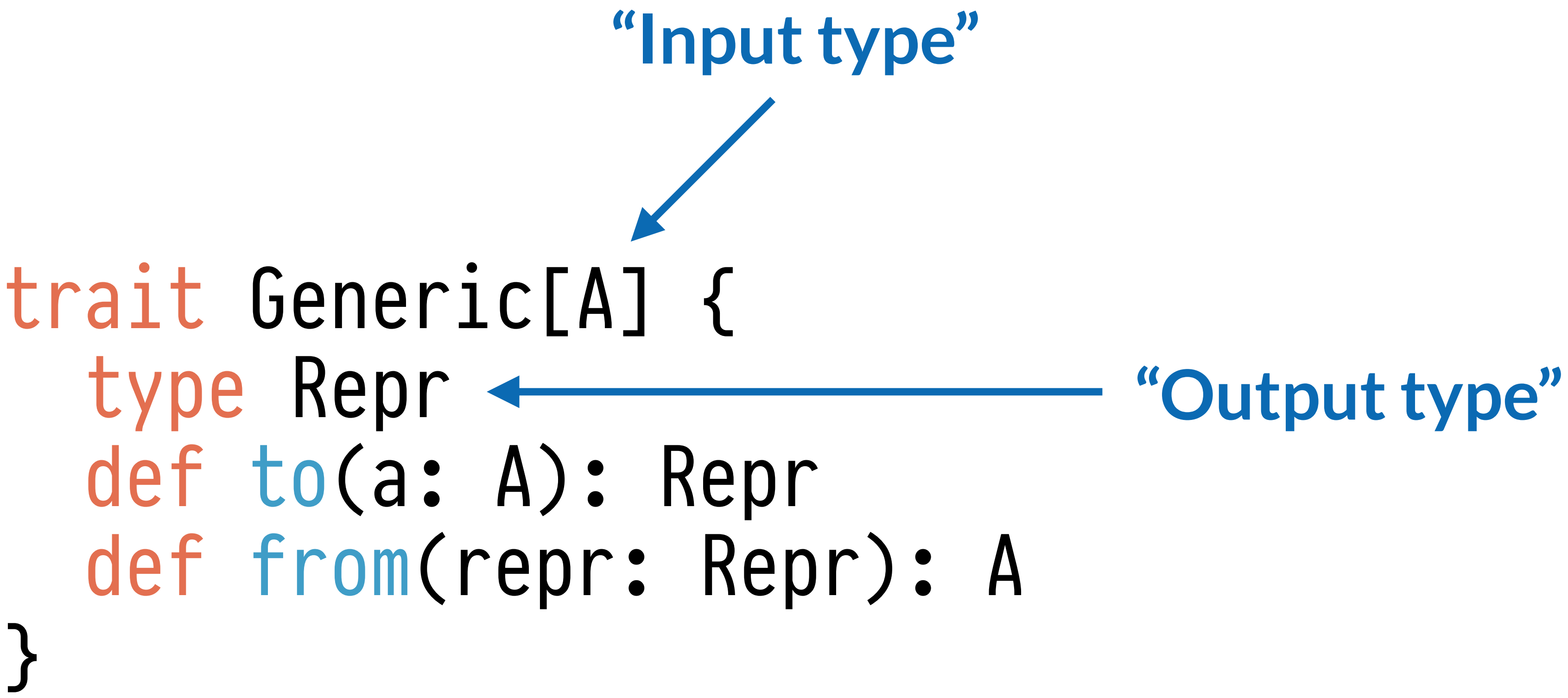
Dependent Types

```
def genericify[A](a: A, gen: Generic[A]) =
  gen.to(a)
```

Dependent Types

```
def genericify[A](a: A, gen: Generic[A]): gen.Repr =
  gen.to(a)
```

Dependent Types



“Input type”

```
trait Generic[A] {
  type Repr
  def to(a: A): Repr
  def from(repr: Repr): A
}
```

“Output type”

Dependent Types

```
def genericEnc[A](
  gen: Generic[A],
  enc: CsvEncoder[gen.Repr]
): CsvEncoder[A] =
  pure(a => enc.encode(gen.to(a)))
```

Dependent Types

```
def genericEnc[A](
  gen: Generic[A],
  enc: CsvEncoder[gen.Repr] ❌
): CsvEncoder[A] =
  pure(a => enc.encode(gen.to(a)))
```

Dependent Types

```
def genericEnc[A, R](
  gen: Generic[A] { type Repr = R },
  enc: CsvEncoder[R]
): CsvEncoder[A] =
  pure(a => enc.encode(gen.to(a)))
```

Dependent Types

```
implicit def genericEnc[A, R](
  implicit
  gen: Generic[A] { type Repr = R },
  enc: CsvEncoder[R]
): CsvEncoder[A] =
  pure(a => enc.encode(gen.to(a)))
```

The “Aux” Pattern

```
trait Generic[A] {
  type Repr
  def to(a: A): Repr
  def from(repr: Repr): A
}

object Generic {
  type Aux[A, R] =
    Generic[A] { type Repr = R }
}
```

The “Aux” Pattern

```scala
implicit def genericEnc[A, R](
  implicit
  gen: Generic[A] { type Repr = R },
  enc: CsvEncoder[R]
): CsvEncoder[A] =
  pure(a => enc.encode(gen.to(a)))
```

The “Aux” Pattern

```
implicit def genericEnc[A, R](
  implicit
  gen: Generic.Aux[A, R],
  enc: CsvEncoder[R]
): CsvEncoder[A] =
  pure(a => enc.encode(gen.to(a)))
```

Demo Time!

csv.scala

Recursive Data Types &
Implicit Divergence

Demo Time!

csv.scala

Implicit Divergence

Is implicit resolution going to converge?

Am I seeing the same type constructor…
…with the same type parameters?
…with more complex type parameters?

Yes?! PANIC!!!

Recursive Data Types

```
sealed trait Tree
case class Branch(left: Tree, right: Tree)
case class Leaf(value: Int)
```

Recursive Data Types

```
sealed trait Tree
case class Branch(left: Tree, right: Tree)
case class Leaf(value: Int)

type TreeRepr   = Branch :+: Leaf :+: CNil
type BranchRepr = Tree :: Tree :: HNil
type LeafRepr   = Int :: HNil
```

Recursive Data Types

```
sealed trait Tree
case class Branch(left: Tree, right: Tree)
case class Leaf(value: Int)

type TreeRepr   = Branch :+: Leaf :+: CNil
type BranchRepr = Tree :: Tree :: HNil
type LeafRepr   = Int :: HNil

genericEnc[Tree, Branch :+: Leaf :+: CNil]
```

Recursive Data Types

```
sealed trait Tree
case class Branch(left: Tree, right: Tree)
case class Leaf(value: Int)

type TreeRepr   = Branch :+: Leaf :+: CNil
type BranchRepr = Tree :: Tree :: HNil
type LeafRepr   = Int :: HNil

genericEnc[Tree, Branch :+: Leaf :+: CNil]
coproductEnc[Branch, Leaf :+: CNil]
```

Recursive Data Types

```
sealed trait Tree
case class Branch(left: Tree, right: Tree)
case class Leaf(value: Int)

type TreeRepr   = Branch :+: Leaf :+: CNil
type BranchRepr = Tree :: Tree :: HNil
type LeafRepr   = Int :: HNil

genericEnc[Tree, Branch :+: Leaf :+: CNil]
coproductEnc[Branch, Leaf :+: CNil]
genericEnc[Branch, Tree :: Tree :: HNil]
```

Recursive Data Types

```
sealed trait Tree
case class Branch(left: Tree, right: Tree)
case class Leaf(value: Int)

type TreeRepr   = Branch :+: Leaf :+: CNil
type BranchRepr = Tree :: Tree :: HNil
type LeafRepr   = Int :: HNil

genericEnc[Tree, Branch :+: Leaf :+: CNil]
coproductEnc[Branch, Leaf :+: CNil]
genericEnc[Branch, Tree :: Tree :: HNil]
hlistEnc[Tree, Tree :: HNil]
```

Recursive Data Types

```
sealed trait Tree
case class Branch(left: Tree, right: Tree)
case class Leaf(value: Int)

type TreeRepr   = Branch :+: Leaf :+: CNil
type BranchRepr = Tree :: Tree :: HNil
type LeafRepr   = Int :: HNil

genericEnc[Tree, Branch :+: Leaf :+: CNil]
coproductEnc[Branch, Leaf :+: CNil]
genericEnc[Branch, Tree :: Tree :: HNil]
hlistEnc[Tree, Tree :: HNil]
genericEnc[Tree, Branch :+: Leaf :+: CNil]
```

Recursive Data Types

```scala
sealed trait Tree
case class Branch(left: Tree, right: Tree)
case class Leaf(value: Int)

type TreeRepr   = Branch :+: Leaf :+: CNil
type BranchRepr = Tree :: Tree :: HNil
type LeafRepr   = Int :: HNil

genericEnc[Tree, Branch :+: Leaf :+: CNil]
coproductEnc[Branch, Leaf :+: CNil]
genericEnc[Branch, Tree :: Tree :: HNil]
hlistEnc[Tree, Tree :: HNil]
genericEnc[Tree, Branch :+: Leaf :+: CNil]
```

❌

It’s Worse Than That…

It’s Worse Than That...

```scala
case class Foo(bar: Bar)
case class Bar(baz: Int, qux: String)
```

It’s Worse Than That...

```
case class Foo(bar: Bar)
case class Bar(baz: Int, qux: String)

type FooRepr = Bar :: HNil
type BarRepr = Int :: String :: HNil
```

It’s Worse Than That...

```
case class Foo(bar: Bar)
case class Bar(baz: Int, qux: String)

type FooRepr = Bar :: HNil
type BarRepr = Int :: String :: HNil

genericEnc[Foo, Bar :: HNil]
```

It’s Worse Than That…

```
case class Foo(bar: Bar)
case class Bar(baz: Int, qux: String)

type FooRepr = Bar :: HNil
type BarRepr = Int :: String :: HNil

genericEnc[Foo, Bar :: HNil]
hlistEnc[Bar, HNil]
genericEnc[Foo, Int :: String :: HNil]
```

It's Worse Than That...

```
case class Foo(bar: Bar)
case class Bar(baz: Int, qux: String)

type FooRepr = Bar :: HNil
type BarRepr = Int :: String :: HNil

genericEnc[Foo, Bar :: HNil]
hlistEnc[Bar, HNil]
genericEnc[Bar, Int :: String :: HNil] ✘
```

Lazy

```
trait Lazy[A]
```

Two jobs:

1. Allow mutually recursive implicits
2. Work around divergence heuristics

Demo Time!

csv.scala

Any Questions?

Accessing
Field and Type Names

JSON Encoding

Field Names in JSON

```
val iceCream: IceCream =
  IceCream("Lolly", 0, false)

{
  "name"        : "Lolly",
  "numCherries" : 0,
  "inCone"      : false
}
```

Type Names in JSON

```
val shape: Shape =
  Rectangle(3, 4)

{
  "Rectangle" : {
    "width"   : 3.0,
    "height"  : 4.0
  }
}
```

LabelledGeneric

Literal & Singleton Types

Singleton Types

```
object Foo

val x: Foo.type = Foo
```

Singleton Types

```
object Foo

val x: Foo.type = Foo

(x : Foo.type)
(x : AnyRef)
(x : Any)
```

Literal Types (SIP-23)

```scala
val x = 42
```

Literal Types (SIP-23)

```
val x = 42

(x : Int)
(x : AnyVal)
(x : Any)
```

Literal Types (SIP-23)

```
val x: Int = 42

(x : Int)
(x : AnyVal)
(x : Any)
```

Literal Types (SIP-23)

Typelevel Scala 2.11.8+
Lightbend Scala 2.12.1+

```
val x: 42 = 42

(x : 42)
(x : Int)
(x : AnyVal)
(x : Any)
```

Literal Types (pre-SIP-23)

```
import shapeless.syntax.singleton._

val x = 42.narrow

// (x : 42)
(x : Int)
(x : AnyVal)
(x : Any)
```

Witness

```
import shapeless.Witness

val witness = Witness.Aux["Dave"]

witness.value // == "Dave"
```

Type Tagging

Type Tagging

```
val data: Int = 12345
```

Type Tagging

```
val data: Int = 12345

trait Tag
```

Type Tagging

```scala
val data: Int = 12345

trait Tag

val tagged = data.asInstanceOf[Int with Tag]
```

Demo Time!

REPL

Tagging with Literal Types

Demo Time!

REPL

Tagging with Literal Types

```
import shapeless.labelled.KeyTag

import shapeless.syntax.singleton._

val tagged = "numCherries" ->> 12345
// tagged: Int with KeyTag["numCherries", Int] = 12345
```

Tagging with Literal Types

```scala
import shapeless.labelled.{KeyTag, FieldType}

import shapeless.syntax.singleton._

val tagged = "numCherries" ->> 12345
// tagged: FieldType["numCherries", Int] = 12345
```

Tagging with Literal Types

```
FieldType[K, V] = V with KeyTag[K, V]
```

Back to LabelledGeneric

LabelledGeneric

```
type IceCreamRepr =
  FieldType['name,        String]  ::
  FieldType['numCherries, Int]     ::
  FieldType['inCone,      Boolean] ::
  HNil
```

LabelledGeneric

```
type ShapeRepr =
  FieldType['Rectangle, Rectangle] :+:
  FieldType['Circle,    Circle]    :+:
  CNil
```

Demo Time!

json.scala

Putting it
All Together

Case Class Migrations

Case Class Migrations

```scala
case class Foo1(a: String, b: Int)
case class Foo2(a: String, b: Int, c: Boolean)

case class Bar1(a: String, b: Int, c: Boolean)
case class Bar2(a: String, c: Boolean)

case class Baz1(a: String, b: Int)
case class Baz2(b: Int, a: String)
```

Case Class Migrations

```
Migration[A, B]
```

1. Convert A to its generic representation
2. Remove fields that are only in A
3. Append fields that are only in B
4. Reorder fields to the order in B
5. Convert generic representation to B

Demo Time!

migrations.scala

Summary

Things We've Seen...

HLists, Coproducts, and Generic

Lazy and Implicit Divergence

Singleton/Literal Types and Type Tagging

LabelledGeneric

Some friends from shapeless.ops

Things We've Not Seen...

Instance prioritisation

Performance
cachedImplicit, Export Hook, etc

Counting with Types
Polymorphic Functions
More friends from shapeless.ops

Further Reading/Watching

Shapeless for Mortals
Sam Halliday, Scala Exchange 2015

Type Parameters versus Type Members
Jon Pretty, NEScala 2016

The source code for
spray-json-shapeless, argonaut-shapeless,
pureconfig, diff, scalacheck-shapeless

We Like Types!

They prevent mistakes!

They help us write code!

We Like Types!

They prevent mistakes!

They help us write code!

They let the compiler write code for us!

Thank You!

Any Questions?

Book

https://github.com/underscoreio/shapeless-guide

Slides

https://github.com/davegurnell/shapeless-guide-slides

Example code

https://github.com/underscoreio/shapeless-guide-code