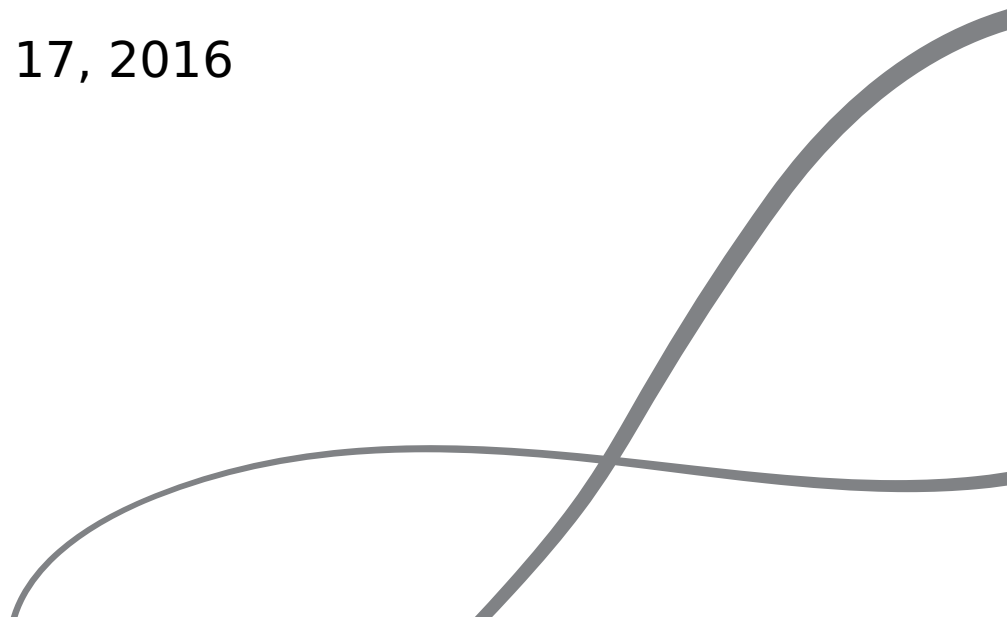
A journey to AutoLifts

Daniel Jentsch

leanovate

February 17, 2016



Targets

- Introduction to AutoLift
- Better understanding of the power of implicit parameter

Outline

Monad-Transformers

AutoLifts

Conclusion

Warning

- Missing imports
  This slides do not contain any of the necessary imports!
- This is advanced stuff
  If you don’t understand everything it’s ok

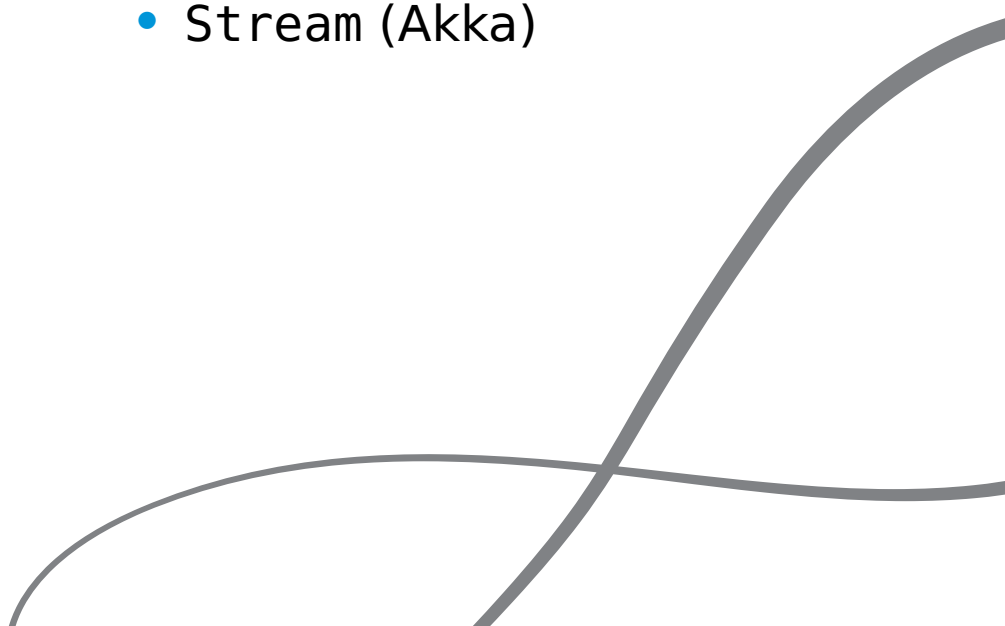
Recap: Monads n’ Stuff

Things with those methods:

- `map`
- `flatMap`
- `fold`
- `filter`

Famous examples

- `List`
- `Option`
- `Future`
- `DBIOAction` (Slick)
- `Stream` (Akka)



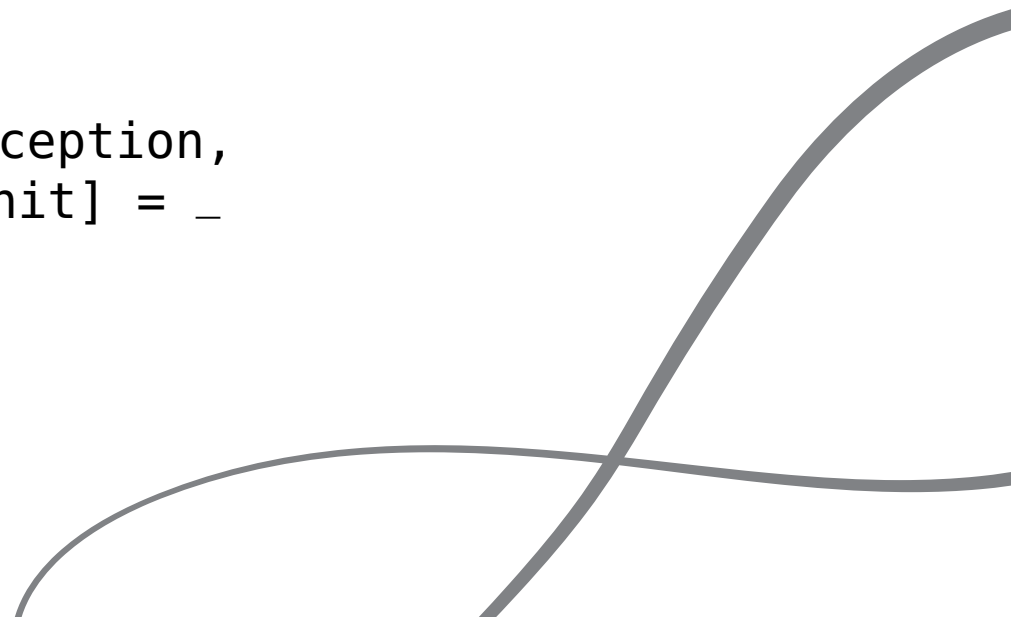
Motivation

```
val future: Future[List[Int]] = _

future.map(_.map(_.toString))

future.map{ list =>
  list.map{ int =>
    int.toString
  }
}


val events: Source[Either[Exception,
    AWSQueueMessage[Int]], Unit] = _
```



Monad-Transformers

- A separate place for the `map(_.map(_.map(...)))`
- `type Example[A] = TList[Future, A]`

Drawbacks

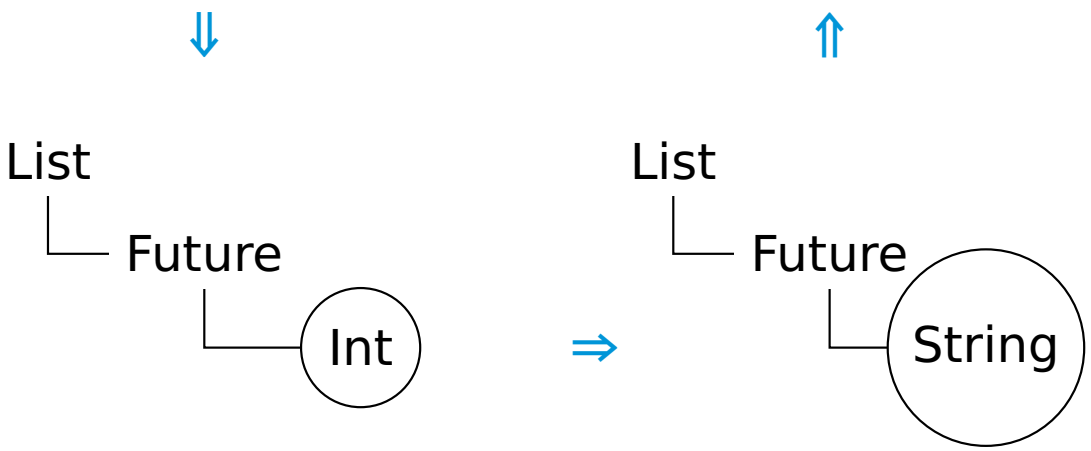
- `TSomething`
- Explicitly constructed

⇒ inflexible

We know what to do

We want to map all integers to strings.

```
List[Future[Int]] ⇒ List[Future[String]]
```



liftMap

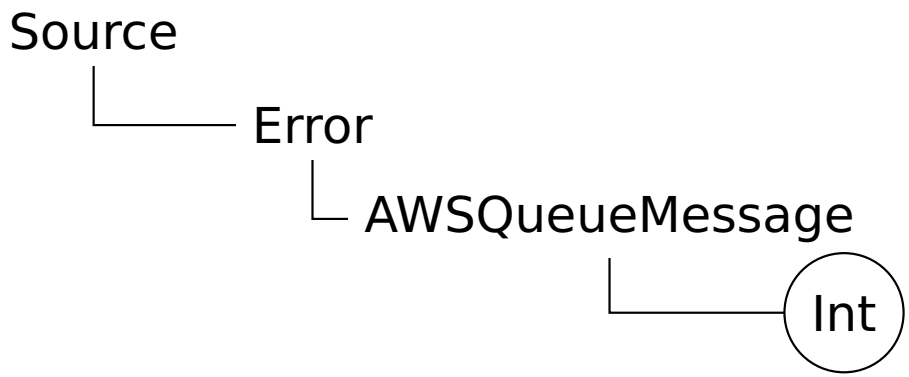
```
val future: Future[List[Int]] = _

future.liftMap{ i: Int => i.toString }
: Future[List[String]]
```

Concrete example

We still want to map all integers to strings.

```
val events: Source[Error[AWSQueueMessage[Int]]] = _
```



Concrete example

```
val events = Source[Error[AWSQueueMessage[Int]]] = _

events.liftMap{ i: Int => i.toString }


def dequeue(message: AWSQueueMessage[Int]): Int

events.liftMap{ dequeue }
: Source[Error[Int]]
```

liftExsits

```
val events = List[Option[Int]] = _

events.foldExists{ i: Int => i > 2 }
```

Conclusion
(personal view)

Cons[AutoLift]

- Mechanism is hard to grasp - even for IDEs
- Still experimental
- Compile time
- May impact on run time
- Bad error message (Computer says no)

Pros[AutoLift]

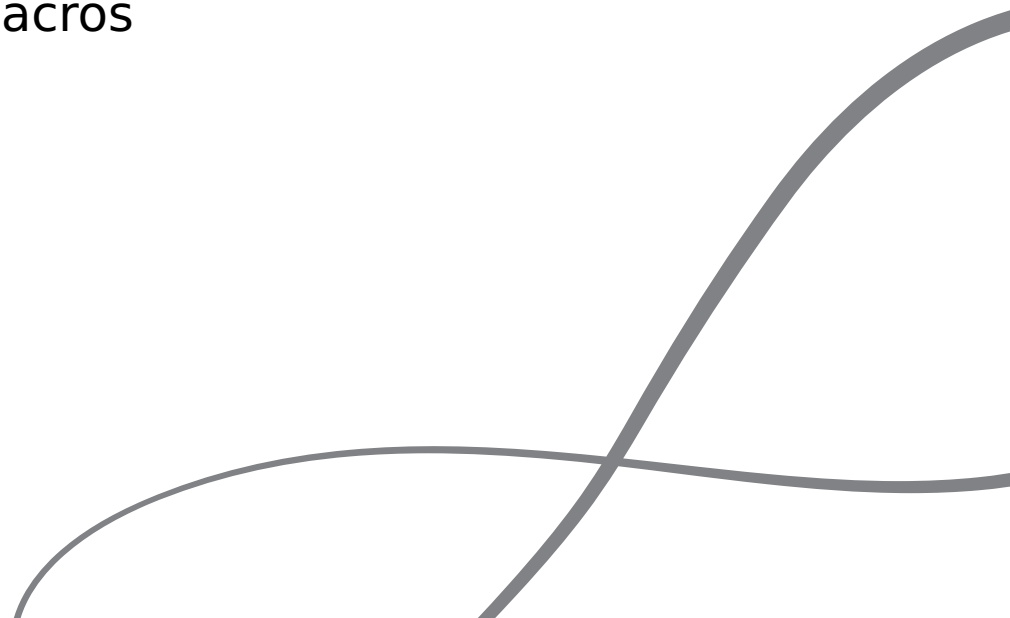
- helps to handle nested generic data structures
  - flexible
  - generic
- readable
- none invasive
- much more methods `flatMap`, `fold`, `filter` ...
- Macro free
- supports Scalaz, Cats, and Algebird
- Version 0.4 just shipped

Future[AutoLift]

- Unapply support
- work on tuples and eithers
- sequence and traverse support
- accepting generic functions

} prototype shows is possible

- better documentation
- better error messages with macros



Questions?

Thanks

Owein Reese & Yosef Fertel
leanovate & ImmobilienScout24

More to read

- AutoLifts:
  https://github.com/wheaties/AutoLifts
- Learning Scalaz (Eugene Yokota)
  http://eed3si9n.com/learning-scalaz/Monad+transformers.html
- Shapless (Miles Sabin)
  https://github.com/milessabin/shapeless
- MonadStuff
  https://github.com/Jentsch/monadstuff

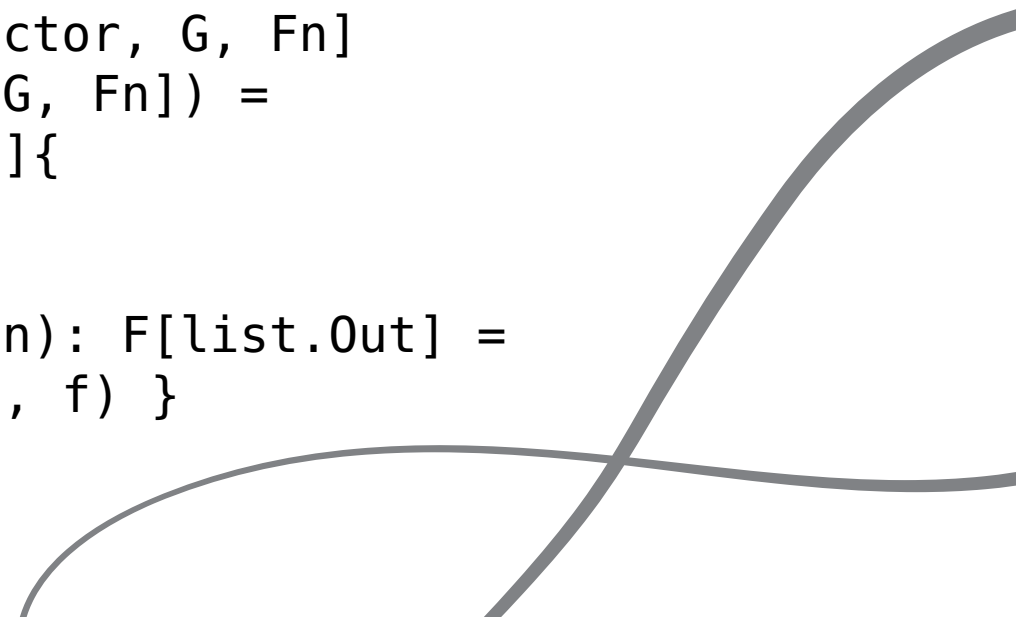
```scala
def liftMap[F[_], A, B, C](fa: F[A])(f: B => C)
    (implicit lift: LiftMap[F[A], B => C]): lift.Out =
  lift(fa, f)


implicit def base[F[_]: Functor, A, B] =
  new LiftMap[F[A], A => B]{
    type Out = F[B]

    def apply(fa: F[A], f: A => B): F[B] =
      fa.map(f)
  }


implicit def recur[F[_]: Functor, G, Fn]
    (implicit lift: LiftMap[G, Fn]) =
  new ScalazLiftMap[F[G], Fn]{
    type Out = F[lift.Out]

    def apply(fg: F[G], f: Fn): F[list.Out] =
      fg.map{ g: G => lift(g, f) }
  }
```



Natural transformations #44

```
def dequeue[A](l: AWSQueueMessage[A]): A =
  l.headOption

val dequeue = new AWSQueueMessage ~> Identity {
  def apply[A](l: List[A]): Option[A] =
    ...
}

val in: Stream[Either[Exception,
    AWSQueueMessage[String]]]
val out = in.map(dequeue)
```

traverse support #56 - traverse included

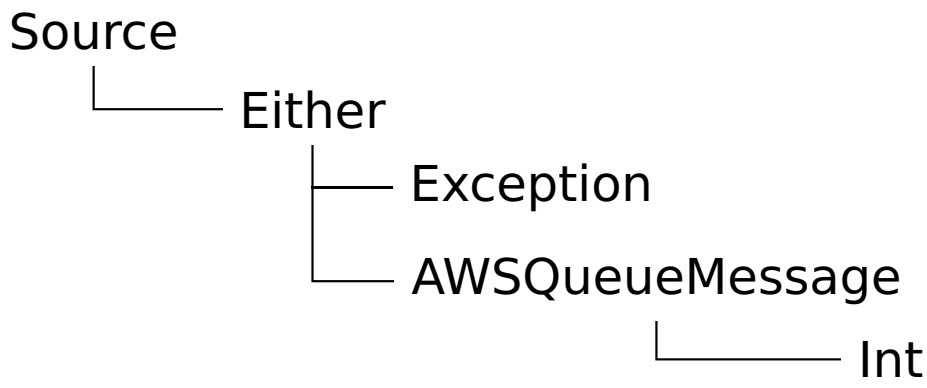
```
import autoLift.Cats._

val result: Future[List[Int]]

result
  .flatTraverse { i: Int => Future{i.toString}}
  : Future[List[Int]]
```

Undecided map

```
val events = Source[Either[Exception,
    AWSQueueMessage[Int]]] = _
```




```
events.liftMap{ e: Exception => e.getMessage }

def dequeue[Int](message: AWSQueueMessage[Int]): Int
events.liftMap(dequeue)
```

Unapply #42

```
import autoLift.Cats._

val result: Flow[String, Option[Double], Unit]

result
  .liftMap { s: Double => sqrt(s) }
: Flow[String, Option[Double], Unit]
```