Scodec, Netty, and Funny looking Functors

LambdaConf 2017

Aka
Real world Functional Applications

Aka use/build good libraries
Your applications are throw away code!

Who am I?

Vincent Marquez

github.com/vmarquez

@runT1ME

working at

verizon labs

currently hiring

come chat after

Outline:

1. Introduce Encoders/Decoders
2. Our sample use case for codecs
3. Quick look at Netty
4. Let’s put it together on the fly!

scodec.org

Scala library for working with binary data

(authored by:by Michael Pilquist)

Encoder: Takes an A, spits out binary data (BitVector)

Decode takes binary data, consumes some to construct an A, and also returns the remainder

Scodec: Encoders and Decoders

```
trait Encoder[A] {
  def encode(a: A): Attempt[BitVector]
}

trait Decoder[A] {
  def decode(bits: BitVector): Attempt[DecodeResult[A]]
}

DecodeResult[A](a: A, remainder: BitVector)
```

Scodec: Encoders and Decoders

```
trait Encoder[A] {
  def encode(a: A): Attempt[BitVector]
}

trait Decoder[A] {
  def decode(bits: BitVector): Attempt[DecodeResult[A]]
}

DecodeResult[A](a: A, remainder: BitVector)
```

Scodec: Encoders and Decoders

```
trait Encoder[A] {
  def encode(a: A): Attempt[BitVector]
}

trait Decoder[A] {
  def decode(bits: BitVector): Attempt[DecodeResult[A]]
}

DecodeResult[A](a: A, remainder: BitVector)
```

Scodec: Encoders and Decoders

```scala
trait Encoder[A] {
  def encode(a: A): Attempt[BitVector]
}

trait Decoder[A] {
  def decode(bits: BitVector): Attempt[DecodeResult[A]]
}

DecodeResult[A](a: A, remainder: BitVector)
```

Let’s encode some data

```
utf8.encode("hi")
Attempt[scodec.bits.BitVector] =
Successful(BitVector(16 bits, 0x6869))
```

```
utf8.encode("hi")
Attempt[scodec.bits.BitVector] =
Successful(BitVector(16 bits, 0x6869))
```

```
utf8.encode("hi")
Attempt[scodec.bits.BitVector] =
Successful(BitVector(16 bits, 0x6869))
```

```
      uint8.encode(1)
Attempt[scodec.bits.BitVector] =
Successful(BitVector(8 bits, 0x01
      uint8.encode(1)
Attempt[scodec.bits.BitVector] =
Successful(BitVector(8 bits, 0x01
def varEncoder[A](sizeEnc: Encoder[Long], valEnc: Encoder[A]) =

 new Encoder[A] {
  def encode(a: A) = for {
    encA <- valEnc.encode(a)
    size <- sizeEnc.encode(encA.size)
  } yield size ++ encA

def sizeBound = sizeEnc.sizeBound.atLeast
}
```

```scala
def varEncoder[A](sizeEnc: Encoder[Long], valEnc: Encoder[A]) =
new Encoder[A] {
  def encode(a: A) = for {
    encA <- valEnc.encode(a)
    size <- sizeEnc.encode(encA.size)
  } yield size ++ encA
  def sizeBound = sizeEnc.sizeBound.atLeast
}
```

```scala
def varEncoder[A](sizeEnc: Encoder[Long], valEnc: Encoder[A]) =
new Encoder[A] {
  def encode(a: A) = for {
    encA <- valEnc.encode(a)
    size <- sizeEnc.encode(encA.size)
  } yield size ++ encA
  def sizeBound = sizeEnc.sizeBound.atLeast
}
```

```scala
def varEncoder[A](sizeEnc: Encoder[Long], valEnc: Encoder[A]) =

new Encoder[A] {
  def encode(a: A) = for {
    encA <- valEnc.encode(a)
    size <- sizeEnc.encode(encA.size)
  } yield size ++ encA
  def sizeBound = sizeEnc.sizeBound.atLeast
}
```

```
def varEncoder[A](sizeEnc: Encoder[Long], valEnc: Encoder[A]) =

 new Encoder[A] {
  def encode(a: A) = for {
    encA <- valEnc.encode(a)
    size <- sizeEnc.encode(encA.size)
  } yield size ++ encA

 def sizeBound = sizeEnc.sizeBound.atLeast
}
```

```scala
val newCodec = varEncoder(uint8, utf8)
val h = newCodec.encode("h")
val w = newCodec.encode("w")

val hw = h.flatMap(bv => h.map(bv2 =>
    bv ++ bv2))

hw: Attempt[scodec.bits.BitVector] =
Successful(BitVector(32 bits, 0x01680177))
```

```
val newCodec = varEncoder(uint8, utf8)
val h = newCodec.encode("h")
val w = newCodec.encode("w")

val hw = h.flatMap(bv => h.map(bv2 =>
    bv ++ bv2))

hw: Attempt[scodec.bits.BitVector] =
Successful(BitVector(32 bits, 0x01680177))
```

```
val newCodec = varEncoder(uint8, utf8)
val h = newCodec.encode("h")
val w = newCodec.encode("w")

val hw = h.flatMap(bv => h.map(bv2 =>
    bv ++ bv2))

hw: Attempt[scodec.bits.BitVector] =
Successful(BitVector(32 bits, 0x01680177))
```

```
val newCodec = varEncoder(uint8, utf8)
val h = newCodec.encode("h")
val w = newCodec.encode("w")

val hw = h.flatMap(bv => h.map(bv2 =>
    bv ++ bv2))

hw: Attempt[scodec.bits.BitVector] =
Successful(BitVector(32 bits, 0x01680177))
```

Decoding

```
utf8.decode(ByteVector.
fromHex("0x6869").get.toBitVector)

Attempt[scodec.DecodeResult[String]] =
Successful(DecodeResult(hi,BitVector(em
pty)))
```

```
utf8.decode(ByteVector.
fromHex("0x6869").get.toBitVector)
```

```
Attempt[scodec.DecodeResult[String]] =
Successful(DecodeResult(hi,BitVector(em
pty)))
```

```
utf8.decode(ByteVector.
fromHex("0x6869").get.toBitVector)

Attempt[scodec.DecodeResult[String]] =
Successful(DecodeResult(hi,BitVector(em
pty)))
```

```scala
def varDecoder[A](sizeDec: Decoder[Long], valDec: Decoder[A]) =
 new Decoder[A] {
  def decode(bv: BitVector) = sizeDec.decode(bv).flatMap {
   case DecodeResult(size, rem) =>
      valDec.decode(rem.take(size*8)) map { res =>
          DecodeResult(res.value, rem.drop(size*8))
      }
    }
}
```

```
def varDecoder[A](sizeDec: Decoder[Long], valDec: Decoder[A]) =
 new Decoder[A] {
  def decode(bv: BitVector) = sizeDec.decode(bv).flatMap {
   case DecodeResult(size, rem) =>
      valDec.decode(rem.take(size*8)) map { res =>
          DecodeResult(res.value, rem.drop(size*8))
      }
    }
}
```

```
def varDecoder[A](sizeDec: Decoder[Long], valDec: Decoder[A]) =
 new Decoder[A] {
  def decode(bv: BitVector) = sizeDec.decode(bv).flatMap {
   case DecodeResult(size, rem) =>
      valDec.decode(rem.take(size*8)) map { res =>
          DecodeResult(res.value, rem.drop(size*8))
      }
    }
}
```

```scala
def varDecoder[A](sizeDec: Decoder[Long], valDec: Decoder[A]) =
 new Decoder[A] {
  def decode(bv: BitVector) = sizeDec.decode(bv).flatMap {
   case DecodeResult(size, rem) =>
      valDec.decode(rem.take(size*8)) map { res =>
          DecodeResult(res.value, rem.drop(size*8))
      }
    }
}
```

```
varDecoder(uint8,
utf8).decode(ByteVector.fromHex("0x01680177").get
.toBitVector)

Attempt[scodec.DecodeResult[String]] =
Successful(DecodeResult(h,BitVector(16 bits,
0x0177)))

}
```

```
varDecoder(uint8,
utf8).decode(ByteVector.fromHex("0x01680177").get
.toBitVector)
Attempt[scodec.DecodeResult[String]] =
Successful(DecodeResult(h,BitVector(16 bits,
0x0177)))

}
```

```
varDecoder(uint8,
utf8).decode(ByteVector.fromHex("0x01680177").get
.toBitVector)
Attempt[scodec.DecodeResult[String]] =
Successful(DecodeResult(h,BitVector(16 bits,
0x0177)))

}
```

I lied to you. you can’t pass a uint8 to a size decoder or encoder…

```
varDecoder(uint8, utf8)
<console>:21: error: type mismatch;
 found   : scodec.Codec[Int]
 required: scodec.Decoder[Long]
              varDecoder(uint8, utf8)
```

We can map the attempt, and map on the decode result
So we can map on the output of the ‘decoding’

```
trait Decoder[A] {
  def decode(bits: BitVector): Attempt[DecodeResult[A]]

  def map[B](f: A => B): Decoder[B]
}
```

```
uint8.asDecoder.map(i => i.toLong): Decoder[Long]
```

What about the encoder? The A is ‘input’?

```
trait Encoder[A] {
  def encode(a: A): Attempt[BitVector]
}
```

What about the encoder? The A is ‘input’?

```
trait Encoder[A] {
  def encode(a: A): Attempt[BitVector]

 def contramap[B](f: B => A): Encoder[B]

}
```

Contravariant Functors

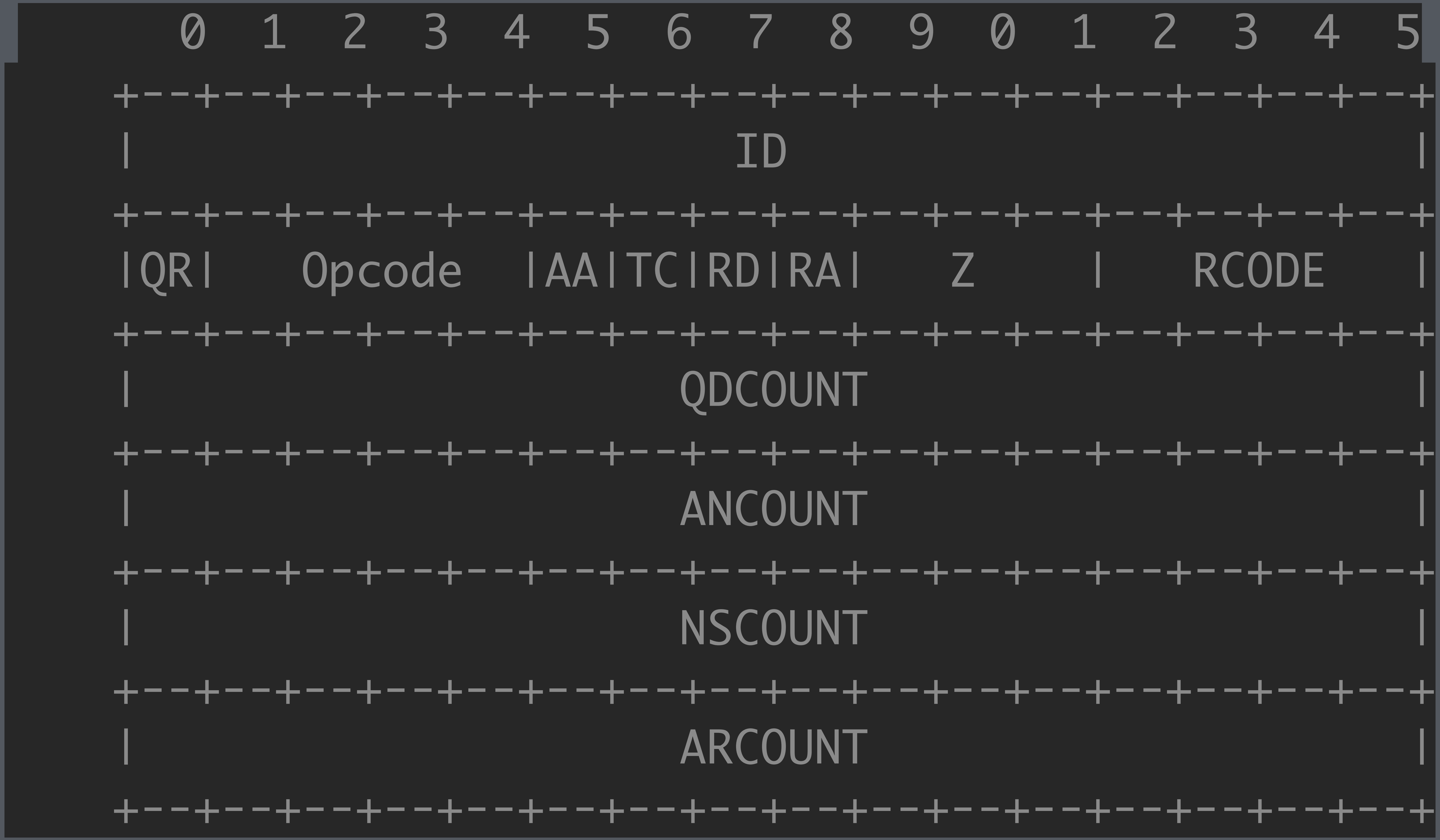
Arrows are reversed.

Not magic. Just function composition!
(Comes in handy later)

Real world example at Verizon: STUN udp protocol

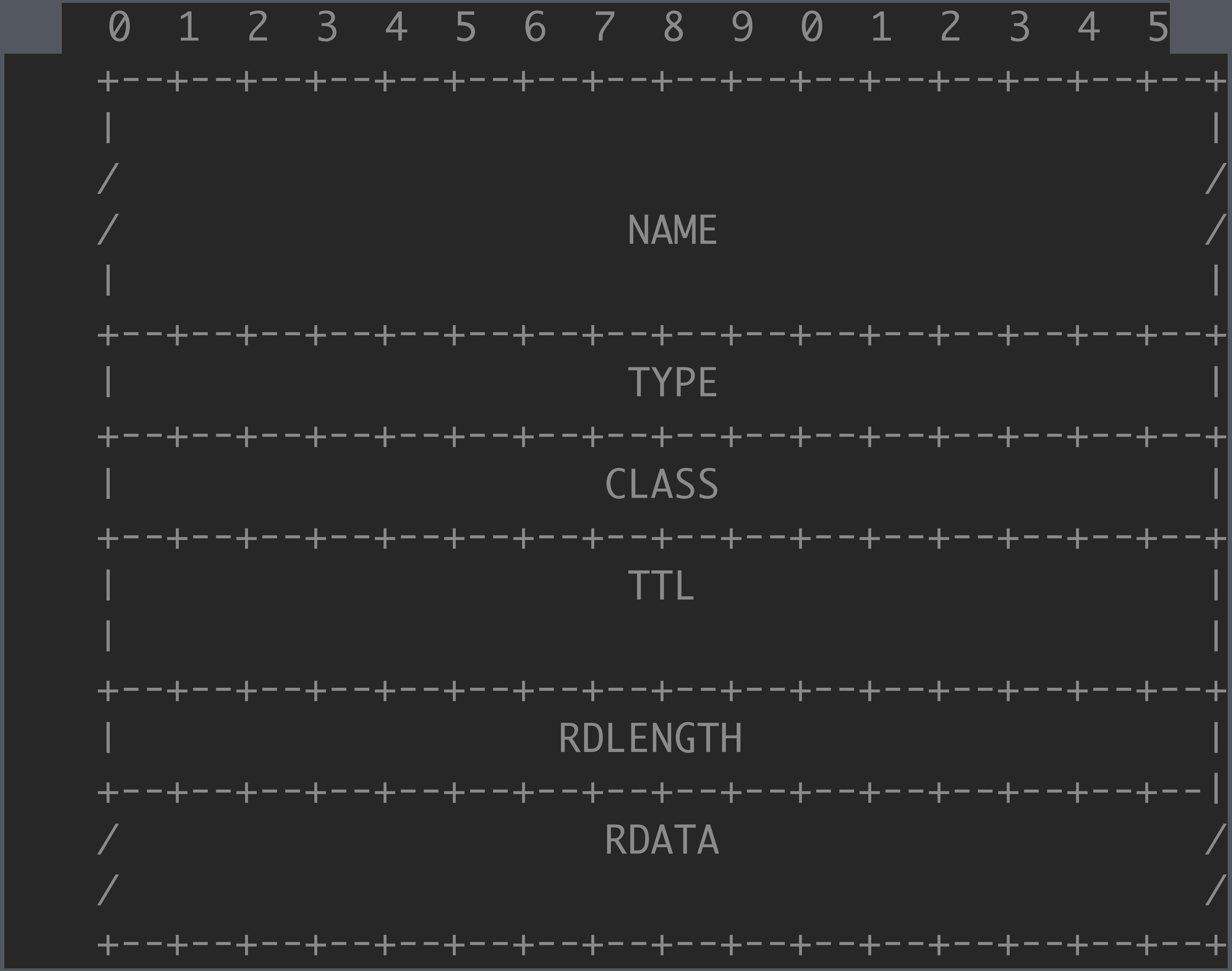
But, not open sourced yet so we will look at another example: DNS

DNS Header:



```
   0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                      ID                       |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|QR|   Opcode  |AA|TC|RD|RA|   Z    |   RCODE   |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    QDCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    ANCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    NSCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    ARCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

DNS Body:



```
  0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                                               |
/                                               /
/                      NAME                     /
|                                               |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                      TYPE                     |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                     CLASS                     |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                      TTL                      |
|                                               |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                   RDLENGTH                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--|
/                     RDATA                     /
/                                               /
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

```scala
case class DnsString(nel: NonEmptyList[String])
case class IPV4(a: Int, b: Int, c: Int, d: Int)
case class ResourceRecord(ttl: Long, ip: IPV4)

case class DnsPacket(
   transactionId: Int,
  msgtype: MessageType,
  question: DnsString,
  addresses: Vector[ResourceRecord])

  trait MessageType
object DnsRequest extends MessageType
object DnsResponse extends MessageType
```

```
  def resourceRecordCodec: Codec[ResourceRecord] = (ignore(64) ::
uint32 :: ipv4).as[ResourceRecord]

  def ipv4: Codec[IPV4] = (uint8 :: uint8 :: uint8 :: uint8).as[IPV4]
```

Etc. until we have a Codec[DnsPacket]

Now to netty

Netty Client …Handler…blah blah Pipeline…blah blah

```
def makeNettyClient(host: String, port: Int)(incoming: DatagramPacket => Task[Unit]): Task[DatagramPacket => Task[Unit]] = {
  for {
   bootstrap <-Task.delay(new Bootstrap())
   group     = new NioEventLoopGroup()
   handler   = simpleHandler(incoming)
   _         = bootstrap.group(group).channel(classOf[NioDatagramChannel]).handler(handler)
   ch       <- Task.delay(bootstrap.bind(0).sync.channel())
  } yield {
    (d: DatagramPacket) =>
     Task.delay {
     ch.writeAndFlush(d).sync()
     println("done sending datagram!")
    }
   }
 }

 def simpleHandler(incoming: DatagramPacket => Task[Unit]) = new SimpleChannelInboundHandler[DatagramPacket] {
  override def channelRead0(ctx: ChannelHandlerContext, packet: DatagramPacket): Unit ={
    println(incoming(packet).attemptRunFor(10.seconds)) //TODO: can I flush here?
   }
 }
```

Netty is just a really good NIO framework. Callbacks etc

```
def makeNettyClient(host: String, port: Int)
 (incoming: DatagramPacket => Task[Unit]): Task[DatagramPacket
=> Task[Unit]]
```

The parameter passed in: a function that handles the response packet

```
(incoming: DatagramPacket => Task[Unit]) => Task[DatagramPacket => Task[Unit]]
```

Output of the function is a Task that has the function for sending out a packet

```
(incoming: DatagramPacket => Task[Unit]) =>
   Task[DatagramPacket => Task[Unit]]
```

To turn any ((A => Unit) => A => Unit) we need to ‘transform’ or map both contravariantly and covariantly

```
(incoming: DatagramPacket => Task[Unit]): Task[DatagramPacket => Task[Unit]]
```

Putting it together:

Codec is a pair of functions

A => Attempt[B], B => Attempt[A]

What else is a pair of functions?

```scala
trait Iso[S, A] {

  def get(s: S): A

  def rget(a: A): S

  def compose[B](iso: Iso[A, B]): Iso[S, B]

  def reverse: Iso[A, S] = Iso(rget, get)
}
```

```
trait Iso[S, A] {

  def get(s: S): A

  def rget(a: A): S

  def compose[B](iso: Iso[A, B]): Iso[S, B]

  def reverse: Iso[A, S] = Iso(rget, get)
}
```

```scala
trait Iso[S, A] {
  def get(s: S): A
  def rget(a: A): S
  def compose[B](iso: Iso[A, B]): Iso[S, B]
  def reverse: Iso[A, S] = Iso(rget, get)
}
```

```
 def datagramIso:
Iso[(InetSocketAddress, Array[Byte]), DatagramPacket]
```

```
def codecIso[A](implicit codec: Codec[A]):
  Iso[Err \/ A, Err \/ BitVector]
```

```
def bvToBa: Iso[BitVector, Array[Byte]]
def addressAbv[A](implicit c: Codec[A]):
Iso[Err \/ (InetSocketAddress, A), Err \/
(InetSocketAddress, BitVector)]
```

```
def addressAbv[A](implicit c: Codec[A]):
Iso[Err \/ (InetSocketAddress, A), Err \/
(InetSocketAddress, BitVector)]
```

Ok, how do we connect a z
Err \/ BitVector
with a BitVector to Array[Byte]?

Strength

Any A => B can become

(C, A) => (C, B)

Choice

Any A => B can become

$(C \vee A) => (C \vee B)$

```
trait Iso[S, A] {

  def get(s: S): A

  def rget(a: A): S

  def compose[B](iso: Iso[A, B]): Iso[S, B]

  def reverse: Iso[A, S]

  def first[C]: Iso[(C, S), (C, A)]

  def choiceRight[C]: Iso[C \/ S, C \/ A]
}
```

```scala
trait Iso[S, A] {

  def get(s: S): A

  def rget(a: A): S

  def compose[B](iso: Iso[A, B]): Iso[S, B]

  def reverse: Iso[A, S]

  def first[C]: Iso[(C, S), (C, A)]

  def choiceRight[C]: Iso[C \/ S, C \/ A]

}
```

```
def codecToBytes[A](implicit codec: Codec[A]):
Iso[Err \/ (InetSocketAddress, A),
Err \/ DatagramPacket] =
    addrBVIso compose
(bvToBa.first[InetSocketAddress].choiceRight[Err] compose
datagramIso.choiceRight[Err])
  }
```

```scala
def codecToBytes[A](implicit codec: Codec[A]):
Iso[Err \/ (InetSocketAddress, A),
Err \/ DatagramPacket] =
    addrBVIso compose
(bvToBa.first[InetSocketAddress].choiceRight[Err] compose
datagramIso.choiceRight[Err])
  }
```

```
def codecToBytes[A](implicit codec: Codec[A]):
Iso[Err \/ (InetSocketAddress, A),
Err \/ DatagramPacket] =
    addrBVIso compose
(bvToBa.first[InetSocketAddress].choiceRight[Err] compose
datagramIso.choiceRight[Err])
  }
```

Our ISOs and Codec now lets us turn our netty client into an entire application with a few LOC. Le’ts try it…