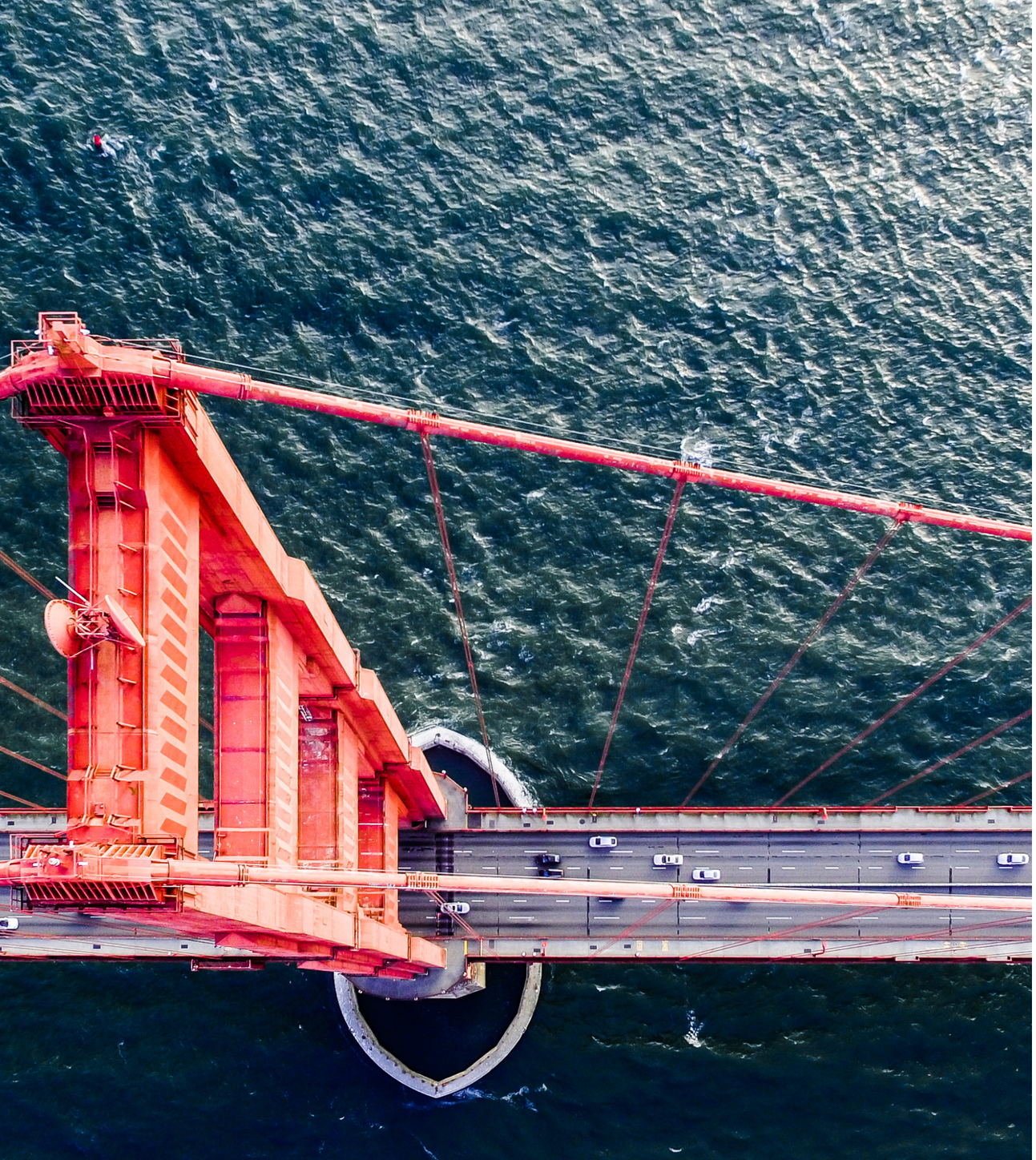


Functional Stream Processing with Scala

@adilakhter

The next 45 minutes

... is about Scalaz-Stream

“You cannot enter
the same river twice"

— Heraclitus

```
@ load.ivy("org.scalaz.stream" %% "scalaz-stream" % "0.8.3")
```

Scalaz-Stream

an extension to the core Scala library for functional programming

SCALAZ-STREAM

01. scalaz.\/

02. scalaz.concurrent.Task

scalaz.

```
\/[A,B]  ≅ scala.Either[A, B]
```

```
type Response = Error \/ String
```

```
def compute(): Response

for {
    response <- compute
  } yield response
```

scalaz.concurrent.

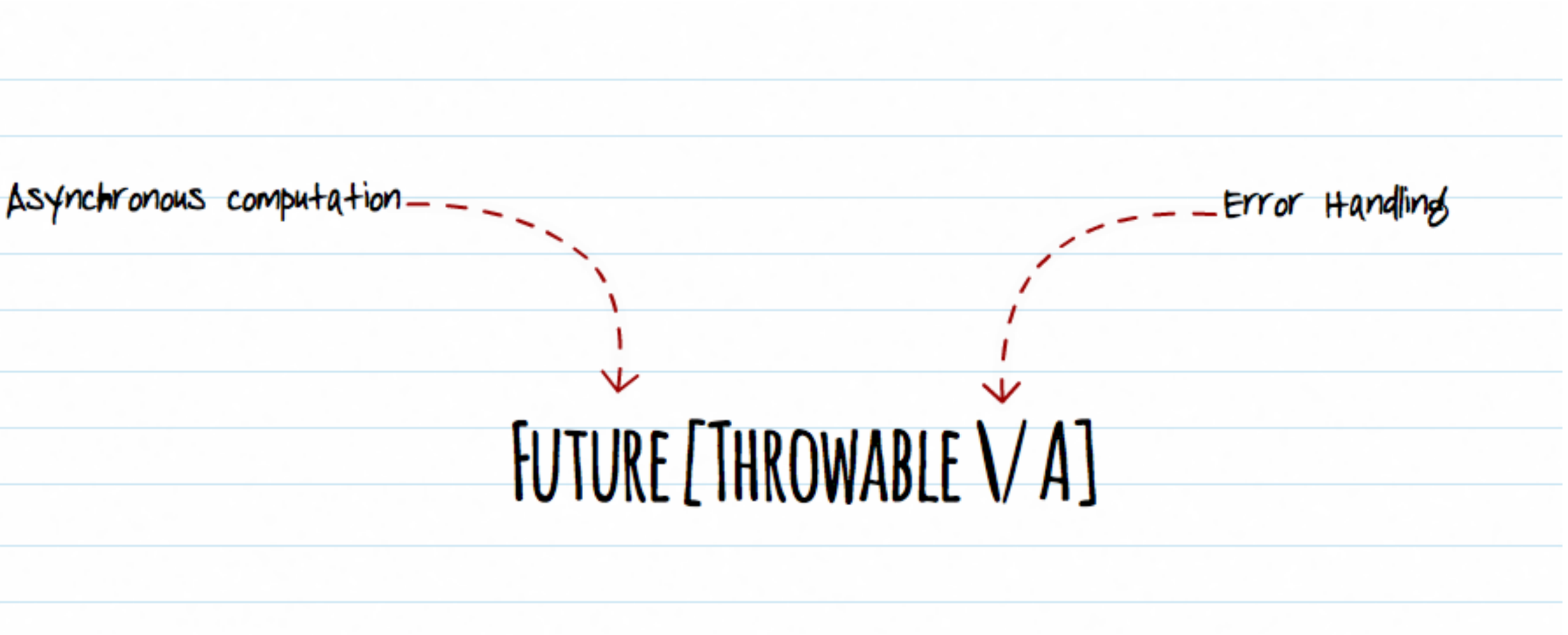
Task

Task

is a

Monadic context for Asynchronous computation

Task[A]


Asynchronous computation
Error Handling
Future[Throwable \/ A]

Task

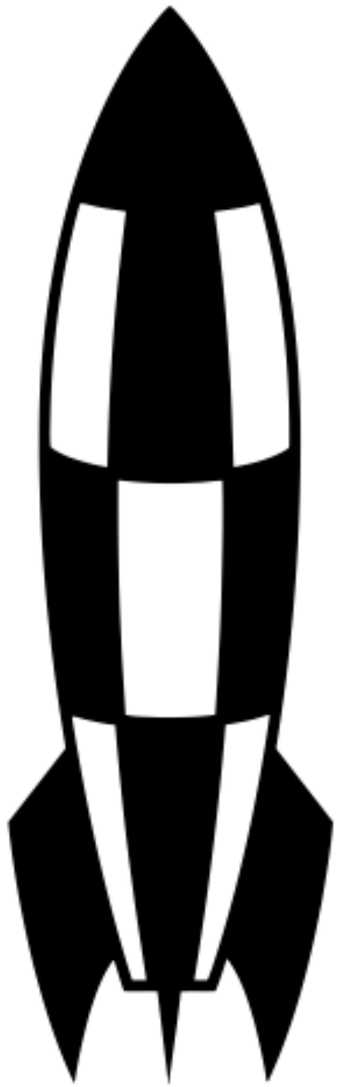
separates

WHAT TO DO FROM WHEN TO DO

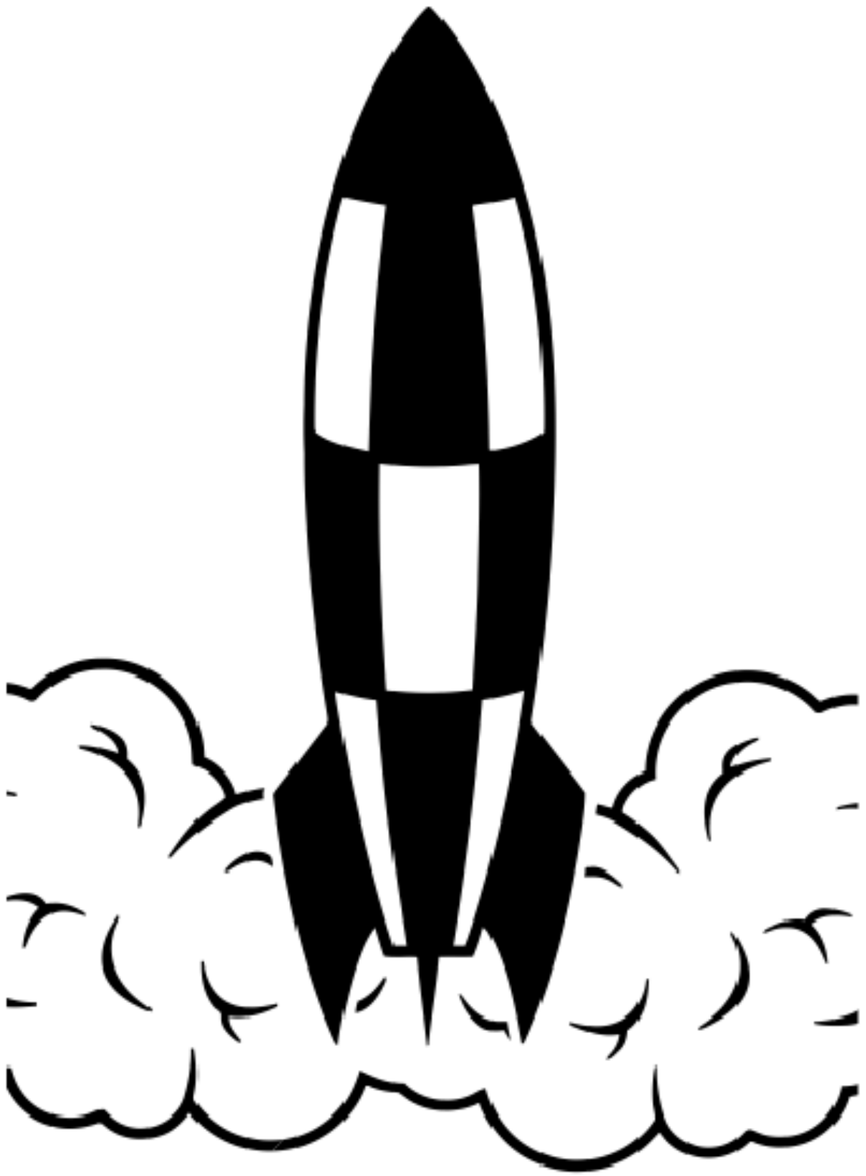
Example

Launching missile

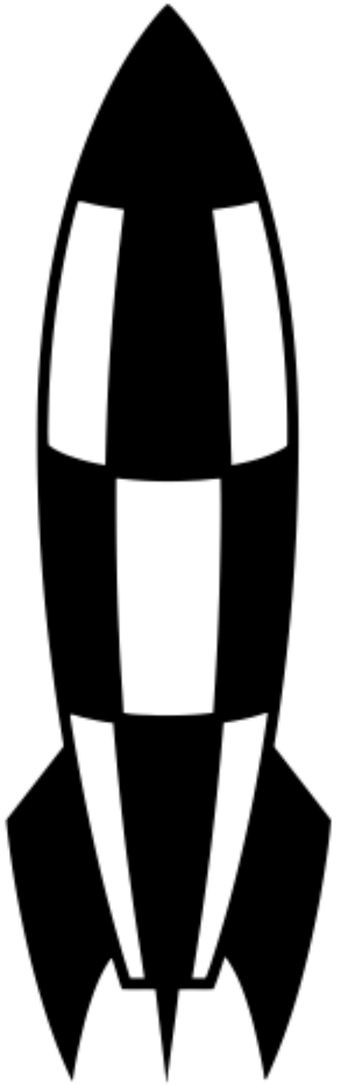
```
@ def launchMissile() : Unit // side-effect
launchMissile: ()Unit
```



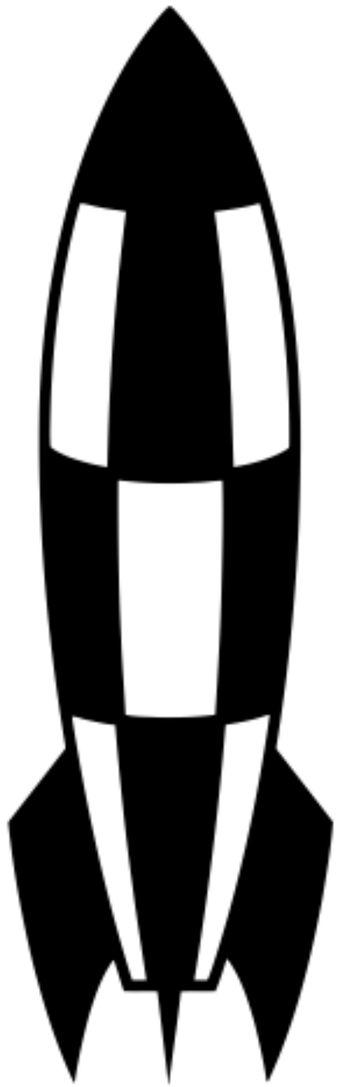
```
@ val launchFuture = Future { launchMissile() }
launchFuture: scala.concurrent.Future[Unit] = ...
// A missile has already launched!
```



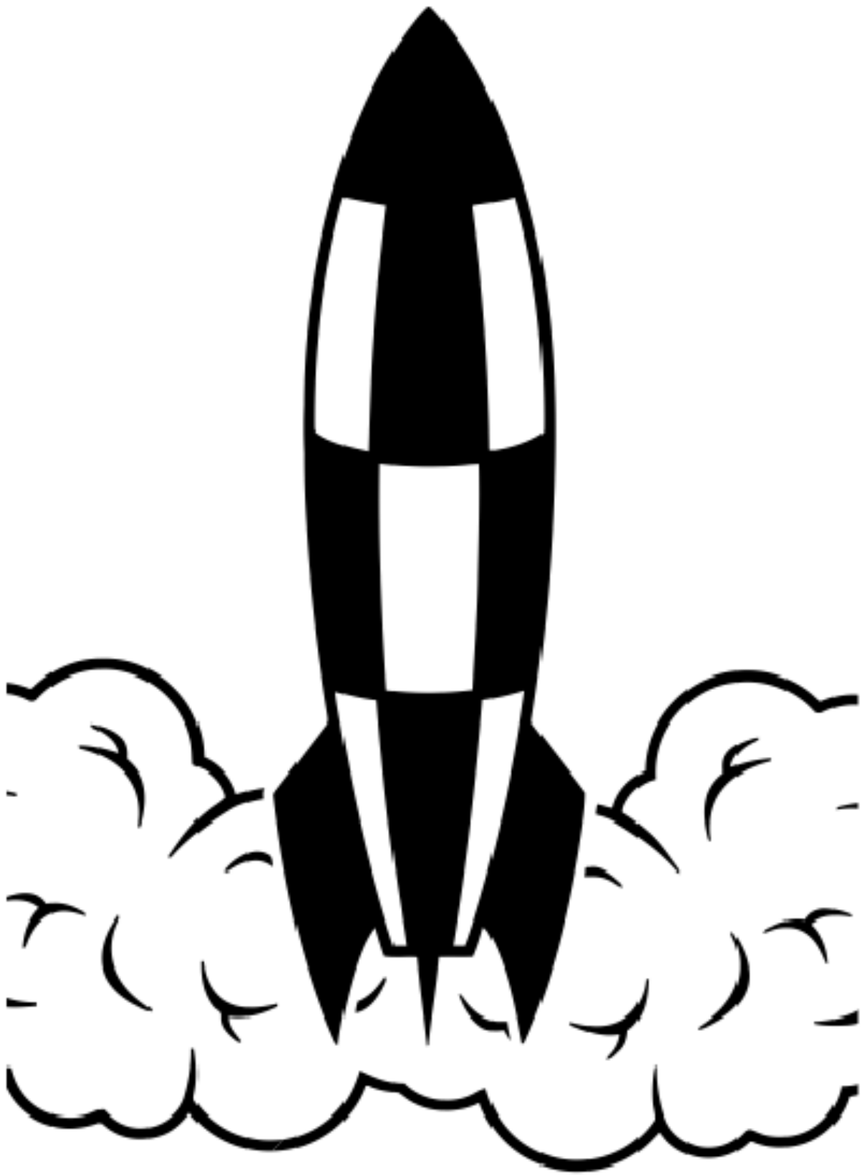
```
@ launchFuture
// nothing happens due to side-effect memoization!
```



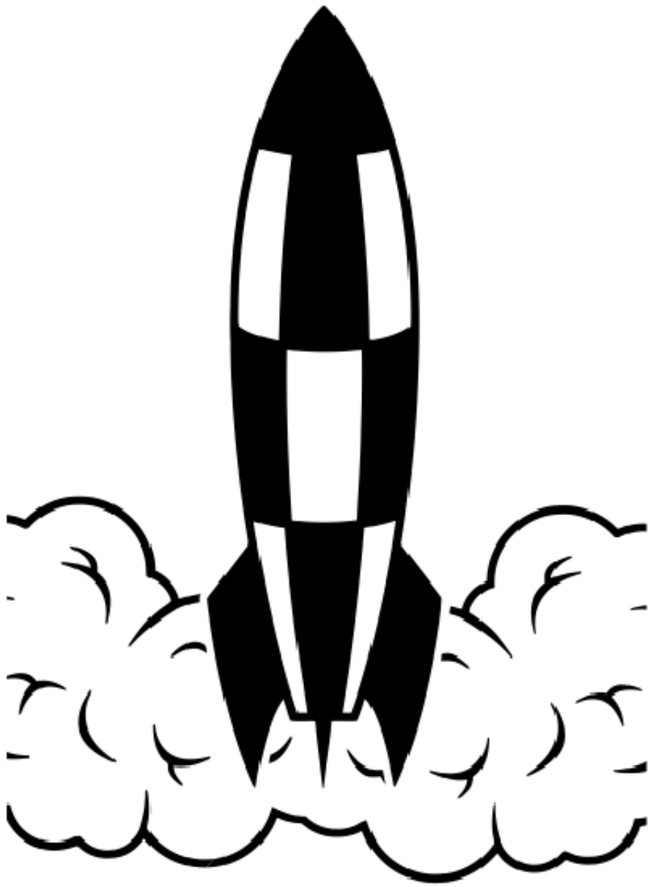
```
@ val launchTask = Task { launchMissile() }
launchTask: scalaz.concurrent.Task[Unit]= ...
```

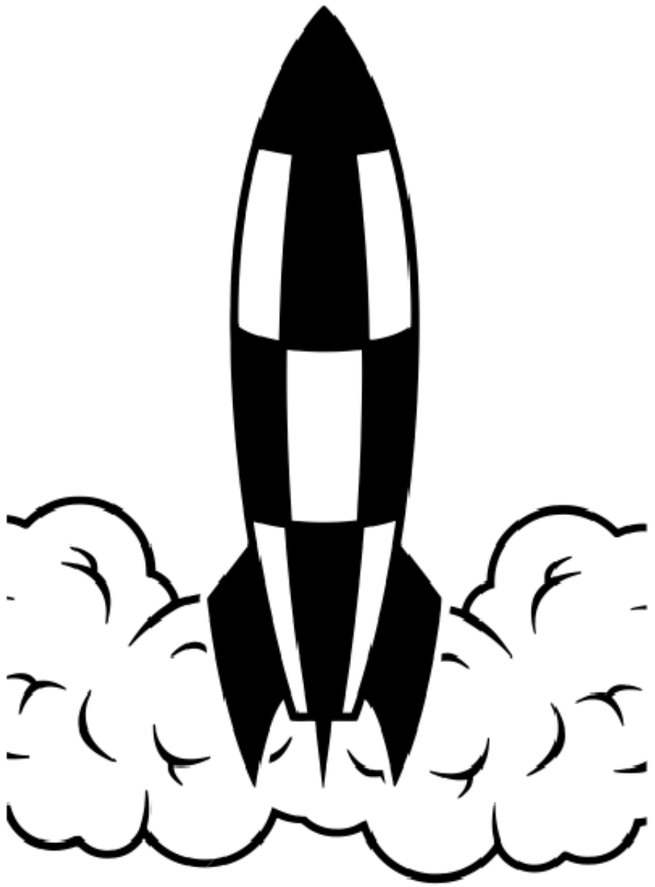


```
@ launchTask.run  // <- a missile now launched
```



```
@ launchTask.run // missile 1 -> launched
@ launchTask.run // missile 2 -> launched
```





Task

is

PURELY FUNCTIONAL & COMPOSITIONAL

Task Constructors

```scala
def now[A](a: A): Task[A]

def delay[A](a: => A): Task[A]

def fail(e: Throwable): Task[Nothing]

def fork[A](a: => Task[A]): Task[A]
```

Example

Querying Twitter

```
@ def statuses: Query => Task[List[Status]] = ???
```

```
@ import twitter4j._

@ val twitterClient = new TwitterFactory(twitterConfig).getInstance()
//      ^
//      |
//      |
//      +
// A Twitter Client
```

```
@ def statuses: Query => Task[List[Status]] = Task {
  twitterClient
    .search(_)
    .getTweets
    .toList
}
```

Scalaz-Stream

Objectives

- Incremental IO & Stream Processing
- Modularity & Compositionality
- Resource-safety
- Performance

- Incremental IO & Stream Processing
- **Modularity & Compositionality**
- Resource-safety
- Performance

```
@ import scala.concurrent.duration._

@ import scalaz._
@ import Scalaz._
@ import scalaz.concurrent.Task

@ import scalaz.stream._
@ import Process._
```

Example

File IO with Scalaz-Stream

```
@ def fahrenheitToCelsius(f: Double): Double
```

```
@ val s: Process[Task, Unit] =
  io.linesR("fahrenheit.txt")
  .filter(s => !s.trim.isEmpty && !s.startsWith("//"))
  .map(line => fahrenheitToCelsius(line.toDouble).toString)
  .intersperse("\n")
  .pipe(text.utf8Encode)
  .to(io.fileChunkW("celsius.txt")) // writes in chunk to files

@ val c: Task[Unit] = s.run // compiles stream to Task
@ c.run // Unit => executes to sideffects to happen
```

Design Philosophy

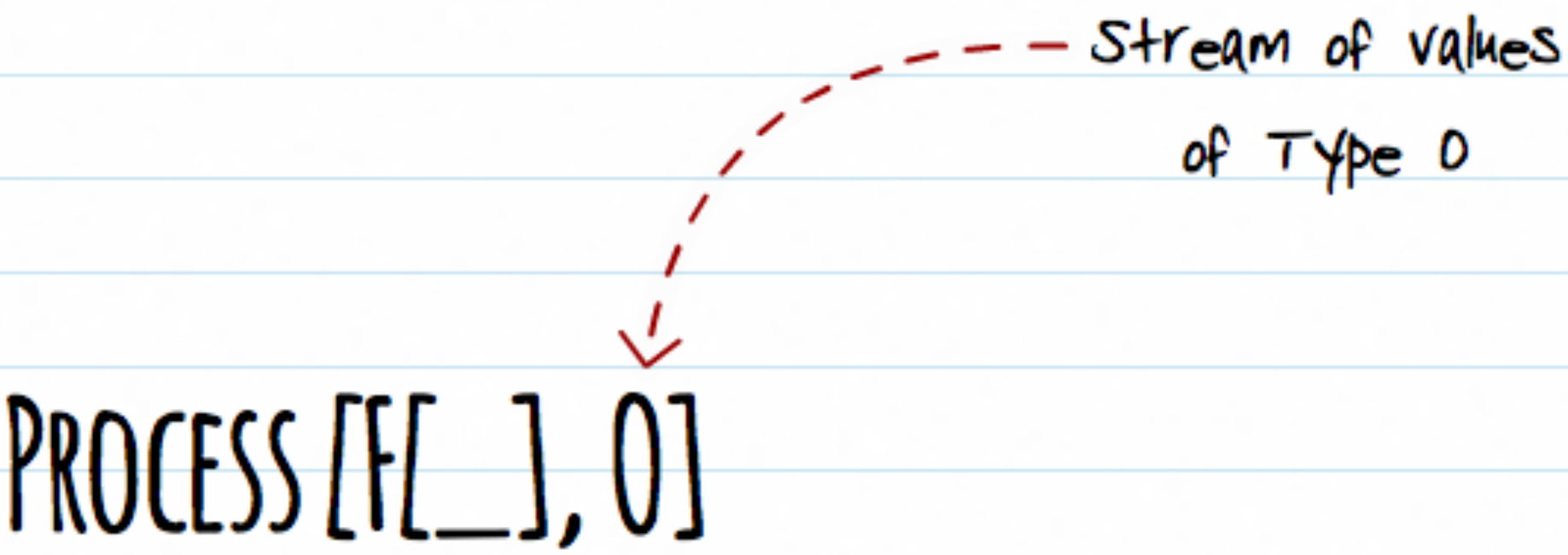
Scalaz-Stream

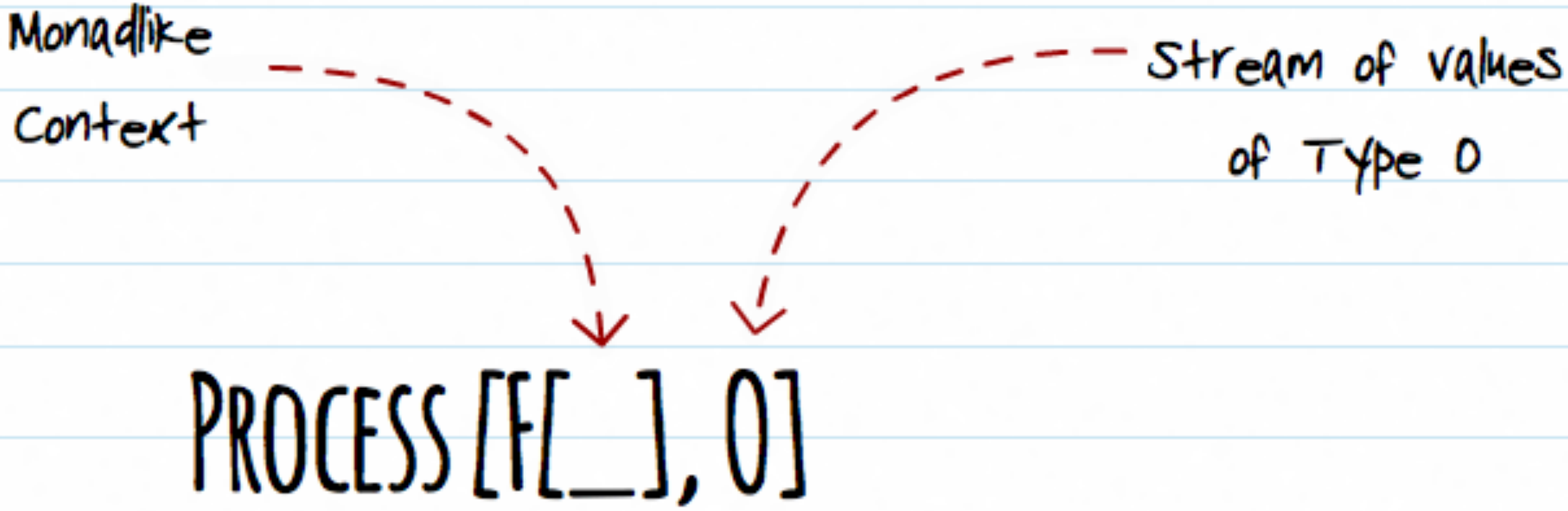
... provides an abstraction
to declaratively specify how to obtain stream of data.

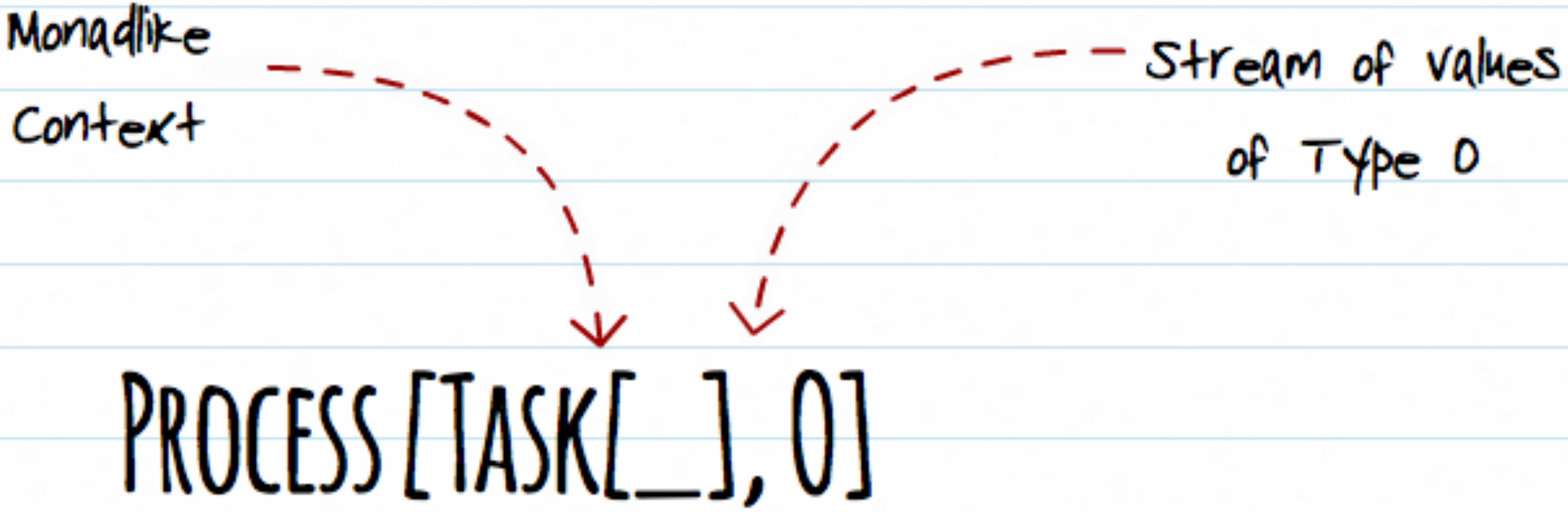
Algebra

Process[F, 0]

```
trait Process[+F[_], +O]
```


Stream of values
of Type O
Process[F[_], O]


Monadlike
Context
Stream of values
of Type O
PROCESS [F[_], O]


Monadlike
Context
Stream of values
of Type O
Process [Task[_], O]

Process = Halt | Emit | Await

Halt

```
case class Halt(cause: Cause)
  extends Process[Nothing, Nothing]
```

```
@ Process.halt
res19: scalaz.stream.Process0[Nothing] = Halt(End)
```

Emit[F[_],O]

```
case class Emit[F[_],O](
  head: Seq[O],
  tail: Process[F, O] = halt) extends Process[F, O]
```

Example

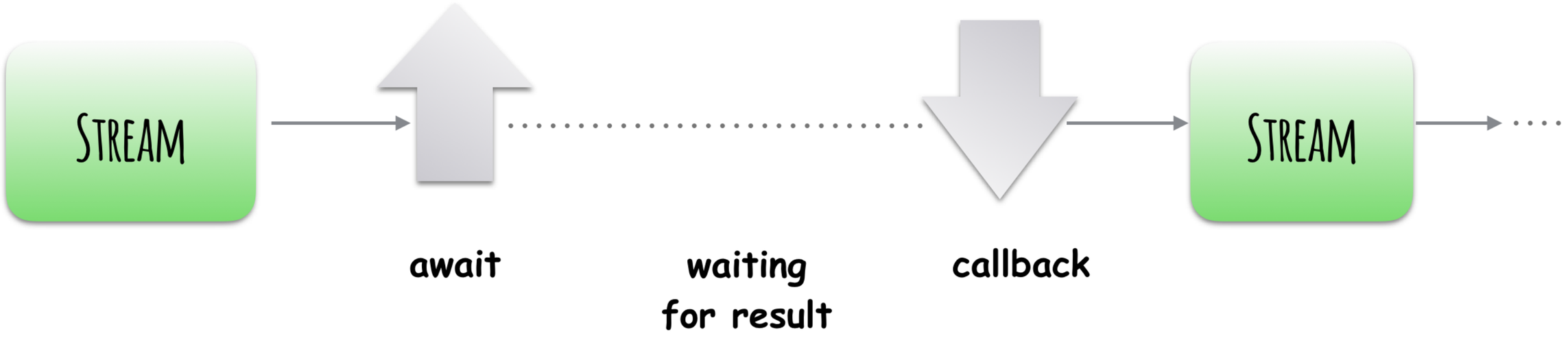
Pure Streams

```
@ import Process._
import scalaz.stream.Process._

@ emit(1)
res1: scalaz.stream.Process0[Int] = Emit(Vector(1))
```

```
@ emitAll(Seq(1,5,10,20))
res2: scalaz.stream.Process0[Int] = Emit(List(1, 5, 10, 20))
```

Await[F[_], I, O]

```
case class Await[F[_], I, O](
  req: F[I], // effect
  recv: I ⇒ Process[F, O], // continuation of the Stream
  ...
  ) extends Process[F, O]
```


STREAM
STREAM
await
waiting
for result
callback

Await[F[_], I, O]

```
case class Await[F[_], I, O](
  req: F[I], // effect
  recv: I ⇒ Process[F, O], // continuation of the Stream
  ...
  ) extends Process[F, O]
```

Example

Effectful Streams

```
// defined earlier
@ val twitterClient: Twitter
@ def statuses(query: Query): Task[List[Status]]
```

```
// Builds a Twitter Search Process for a given Query
@ def buildSearchProcess(query: Query): Process[Task, Status] = {
  val queryTask: Task[List[Status]] = statuses(query)

  await(queryTask){ statusList ⇒ // <-
    Process.emitAll(statusList)
  }
}
//In essence, it builds: Process: Await ⇒ (_ ⇒ Emit ⇒ Halt)
```

```
@ val searchProcess: Process[Task, Status] =
    buildSearchProcess(new Query("#spark"))
```

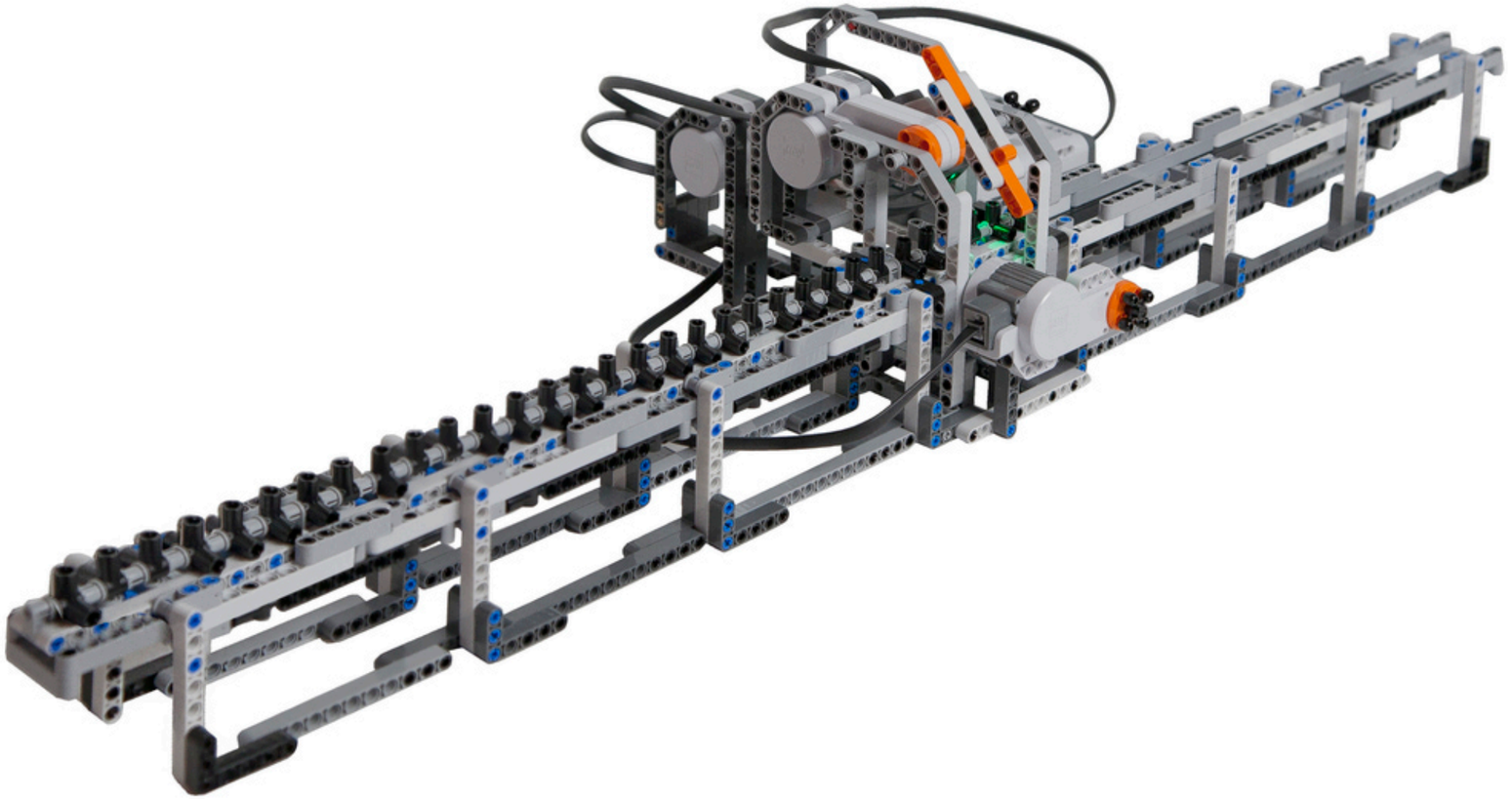
Example

Build a Stream of Positive Integers

```
@ def integerStream: Process[Task, Int] = {
    def next (i: Int): Process[Task,Int] = await(Task(i)){ i ⇒
      emit(i) ++ next(i + 1)
    }
    next(1)
  }
defined function integerStream
```

Process[F, O]

... represents a stream of O values which can interleave with the evaluation of F[_]



Process Constructors

```
def eval[F[_], O](fo: F[O]): Process[F, O]

def repeatEval[F[_], O](fo: F[O]): Process[F, O]

def unfold[S, A](s0: S)(f: S => Option[(A, S)]): Process0[A]

def await[F[_], A, O](req: F[A])(rcv: A => Process[F, O])

...
```

Running a Process

run

```
def run[F[_]]: : F[Unit]
```

```
@ integerStream.take(10)
res3: Process[Task, Int] = ...

@ val task = integerStream.take(10).run
task: Task[Unit] = scalaz.concurrent.Task@44d0008f

@ task.run
// outputs nothing?
```

runLog

```
def runLog[F[_], O]:  F2[Vector[O]]
```

```
@ integerStream.take(10).runLog.run
res4: Vector[Int] = Vector(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

runLog

```
@ def integerStream: Process[Task, Int]

@ integerStream.runLog.run //  = ?
```

Other Approaches

```
def runLast: F[Option[A]]

def runFoldMap(f: A => B): F[B]
```

Transformations

```
// map
@ integerStream.map(_ * 100).take(5).runLog.run
res5: Vector[Int] = Vector(100, 200, 300, 400, 500)
```

```
//flatMap
@ integerStream.flatMap(i ⇒ emit(-i) ++ emit(-i-1)).take(5).runLog.run
res6: Vector[Int] = Vector(-1, -2, -2, -3, -3)
```

```
// Zip
@ val zippedP: Process[Task, (Int, String)]
        = integerStream.take(2) zip emitAll(Seq ("A", "B"))
@ zippedP.runLog.run
res2: Vector[(Int, String)] = Vector((1,A), (2,B))
```

Compositionality

Process1[-I,+O]

```
type Process1[-I,+O] = Process[Env[I,Any]#Is, O]
```

```
> Process[F, A].pipe(Process1[A, B]) => Process[F, B]
```

```
@ import process1._
```

```
@  integerStream |> filter(i => i > 5)
res12: Process[Task, Int] = Append(Halt(End), Vector(<function1>))
```

```
@ integerStream |> filter(i => i > 5) |> exists(_ == 10)
res13: Process[Task, Boolean] = Append(Halt(End), Vector(<function1>))
```


```
Channel[+F[_],-I,O]
```

```
type Channel[+F[_],-I,O] = Process[F, I => F[O]]
```

```
> Process[F, A].through(Channel[Task, A, B]) => Process[Task, B]
```

```
@ val multiply10Channel: Channel[Task, Int, Int] =
    Process.constant { (i: Int) =>
      Task.now(i*10)
    }
multiply10Channel: Channel[Task, Int, Int] = ...
```

```
@ val stream: Process[Task, Int] =
  integerStream.take(5) through multiply10Channel

stream: Process[Task, Int] = ...
```

```
@ stream.runLog.run
res18: Vector[Int] = Vector(10, 20, 30, 40, 50)
```

Sink[+F[_],-O]

```
type Sink[+F[_],-O] = Process[F, O => F[Unit]]
```

```
> Process[F, A].to(Sink[Task, A]) => Process[Task, Unit]
```

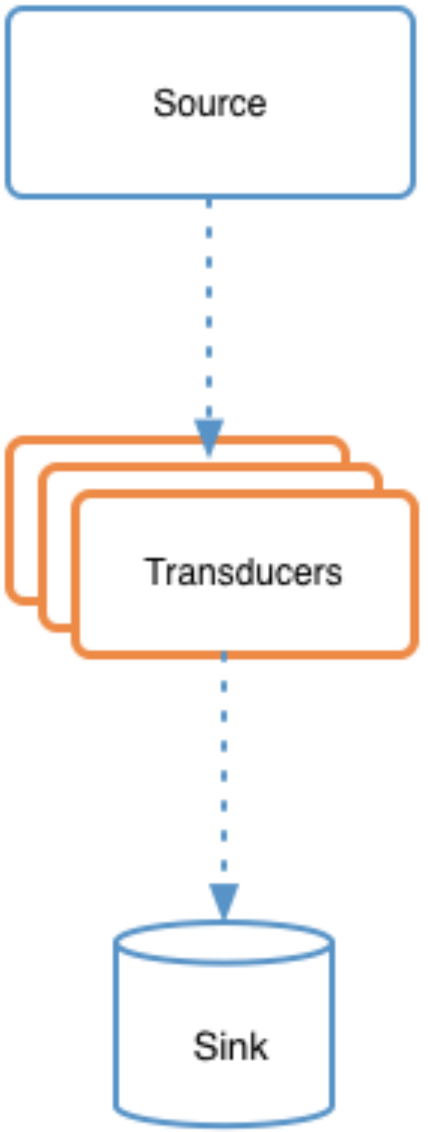
```
@ def log[A] : Sink[Task, A] = Process.constant (
    x => Task.delay{
      println(s"> $x")
    }
  )
```

```
@ val stream = integerStream.take(5) to log
stream: Process[Task, Unit] = ...
```

```
@ val task = stream.run
task: Task[Unit] = scalaz.concurrent.Task@36cf8df4
@ task.run
> 1
> 2
> 3
> 4
> 5
```

Example

Sentiment Analysis of Tweets


Source
Transducers
Sink

Building Source

```
@ val durations: Process[Task, Duration] = awakeEvery(15.seconds)

@ val requests: Process[Task, Query] = Process.constant(query)
```

```
@ val src: Process[Task, Query] = (requests zip durations).map {
      case (request, _) => request
    }
```

Twitter Query Channel

```
// defined earlier
@ def statuses(query: Query): Task[List[Status]] = ...

// Lift the Task to construct a channel
@ val queryChannel: Channel[Task, Query, List[Status]]
  = channel lift statuses
```

Stream Pipeline (so far)

```
@ src
    .through(queryChannel)      // Process[Task, List[Status]]
    .flatMap(emitAll)           // Process[Task,Status]
    .map(s => Tweet(s))         // Process[Task, Tweet]
```

Sentiment Analysis Channel

```
@ import SentimentAnalyzer._

@ def analyze(t: Tweet): Task[EnrichedTweet] = Task {
    EnrichedTweet(
      t.author,
      t.body,
      t.retweetCount,
      sentiment(t.body))
 }
```

```
@ def analysisChannel: Channel[Task, Tweet, EnrichedTweet] =
  channel lift analyze
```

Stream Pipeline (so far)

```
@ src
    .through(queryChannel)      // Process[Task, List[Status]]
    .flatMap(emitAll)           // Process[Task,Status]
    .map(s => Tweet(s))         // Process[Task, Tweet]
```

Stream Pipeline (so far)

```
@ src
    .through(queryChannel)     // Process[Task, List[Status]]
    .flatMap(emitAll)          // Process[Task,Status]
    .map(s => Tweet(s))        // Process[Task, Tweet]
    .through(analysisChannel) // Process[Task, EnrichedTweet]
```

Stream Pipeline

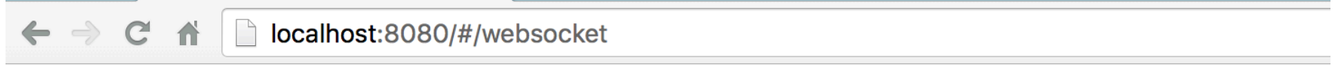
```
@ val tweetStream =
    src
      .through(queryChannel)     // Process[Task, List[Status]]
      .flatMap(emitAll)          // Process[Task,Status]
      .map(s => Tweet(s))        // Process[Task, Tweet]
      .through(analysisChannel) // Process[Task, EnrichedTweet]
```

Evaluating Stream Pipeline

```
@ val consoleTweetStream =
  tweetStream.to(log) // Process[Task, Unit]

@ consoleTweetStream.run.run
```

```
 > 😞🔁0 ➠ @randerzander» millions of 0 byte files.. uhhh #S3 #Spark #problems https://t.co/emfISBLRfS

 > 😒🔁0 ➠ @tgau» Get started developing with IBM Analytics for #Apache #Spark https://t.co/LZ67ONjQhL https://t.co/BFmhrUZsvH

 > 😒🔁2 ➠ @AlineMichel22» RT @olivierrafal: On investit énormément dans l'open source, notamment autour de #Spark
@IBMBigDataFr au #ForumduNumerique https://t.co/7V…

 > 😒🔁2 ➠ @flocalvez» RT @olivierrafal: On investit énormément dans l'open source, notamment autour de #Spark
@IBMBigDataFr au #ForumduNumerique https://t.co/7V…

 > 😒🔁2 ➠ @olivierrafal» On investit énormément dans l'open source, notamment autour de #Spark
@IBMBigDataFr au #ForumduNumerique https://t.co/7VsxRQBEFm

 > 😒🔁1 ➠ @iotsecurity2» RT @esthermeadDev: "#DataScience for #InternetOfThings course-Aug-Sep2016" https://t.co/OFQh9XODBZ (#IoT #ML #python #R #hadoop #spark) htt…

 > 😒🔁0 ➠ @natalieDatio» According to Tech Overflow #Spark tied with #Scala for the top-paying job in technology. https://t.co/5jWwlvNdJw

 > 😒🔁1 ➠ @esthermeadDev» "#DataScience for #InternetOfThings course-Aug-Sep2016" https://t.co/OFQh9XODBZ (#IoT #ML #python #R #hadoop #spark) https://t.co/tmgsMLFubn

 > 😊🔁1 ➠ @TelNews1» RT @mobileworldlive: Asia Briefs: #Singtel to launch #VoWiFi in August, #NewZealand's #Spark claims top spot by revenue https://t.co/iJaog6…

 > 😊🔁1 ➠ @ITCrowdsource» RT @SKBhere: "Why #Spark & #NoSQL?" - great read https://t.co/5Q4AUnUc1R #BigData #DevOps #Couchbase https://t.co/s7OU1aiF79

 > 😊🔁1 ➠ @SKBhere» "Why #Spark & #NoSQL?" - great read https://t.co/5Q4AUnUc1R #BigData #DevOps #Couchbase https://t.co/s7OU1aiF79

 > 😒🔁0 ➠ @TheJol» #functinalprogramming is back! #scala #spark #datascience #bigdata https://t.co/bIEGkEgn7J
```

```
// http4s Websocket Route
case r @ GET -> Root / "websocket" ⇒
  val query = new Query("#spark")
  val stream: Process[Task, Text] =
    twitterSource(query)
      .map(Text(_))

  WS(Exchange(stream, Process.halt)) // <- connected stream to WS
```



Functional Stream Processing with Scala

- Home
- TwitterStream
- TwitterStream-V2

> 😒 🔁 5 ➡ @vikbhan» RT @sreeSF: Our rockstar engineer @chtyim talking about latest hotness @caskdata #brownbag #spark #transactions #startuplife https://t.co/…

> 😒 🔁 0 ➡ @bradhugg» Databricks announces Apache Spark 2.0 technical preview - SD Times https://t.co/G73fhk8d7K #databricks #spark

> 😊 🔁 1 ➡ @AnaMEcheverri» RT @apachespark_tc: Learn how #Scala applications can easily communicate with a #Spark Kernel #apachespark https://t.co/s3Cok0q4xx

> 😞 🔁 1 ➡ @DominicSpitz» RT @timvanbaars: Analytical Searches with Apache #Spark and #MarkLogic working like magic https://t.co/eVFHnJODQX

> 😒 🔁 9 ➡ @BigDataBuzzNet» RT @kdnuggets: #BigData Engineering: Using the Mongo #Hadoop Connector as a Translation Layer to #Spark https://t.co/L2FlWzjMZs

> 😒 🔁 0 ➡ @IBM_Innovation» #SparkBizApps #Spark and R: The deepening open #analytics stack https://t.co/HtkOATBjo7 https://t.co/JqQKFBMCv2

> 😒 🔁 0 ➡ @SocialMktg4U» #SparkBizApps #Spark and R: The deepening open #analytics stack https://t.co/lE6e4R803B https://t.co/UJt84RJlVs

> 😒 🔁 0 ➡ @IBM_zAnalytics» #SparkBizApps #Spark and R: The deepening open #analytics stack https://t.co/bMBmX399XD https://t.co/tooPtV1ggX

> 😊 🔁 0 ➡ @HandiHeather» #CharlieTheRaptor loves his @advocare #Spark #FruitPunch. It's going… https://t.co/a1V0aAUmQX

Example

Sentiment Analysis of Tweets using
Twitter's Streaming API

```
@ val twitterStream: TwitterStream = ???

@ val tweetsQueue: Queue[Status]
  = async.boundedQueue[Status](size)
```

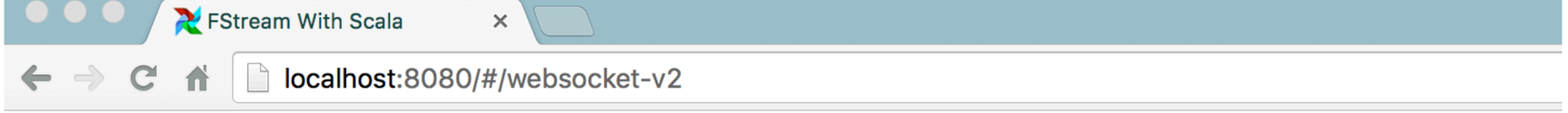
```
@ def stream(sink: Sink[Task, Status]) = Task{
    twitterStream.
      addListener(twitterStatusListener(sink))
    // => status to sink
    ...
  }
```

Producer

```
@ Process.eval_(stream(tweetsQueue.enqueue))
    .run
    .runAsync { _ ⇒ println("input done!") }
```

Consumer

```
@ val tweetStream =
    tweetsQueue
     .dequeue                          // Process[Task, Status]
     .map(s => Tweet(s))               // Process[Task, Tweet]
     .through(analysisChannel) // Process[Task, EnrichedTweet]
```



Functional Stream Processing with Scala

Home

TwitterStream

TwitterStream-V2

> 😞 🔁 0 ➡ @lovelydaisies4» @takeaseatpls I will. When I get home bc then I'm gonna get hype af lol

> 😊 🔁 0 ➡ @tmj_sac_eng» Join the Thermo Fisher Scientific team! See our latest #Engineering #job opening here: https://t.co/8tOlRgep6z #Fremont, CA #Hiring

> 😞 🔁 0 ➡ @83126slhb» Get Weather Updates from The Weather Channel. 17:36:59

> 😒 🔁 0 ➡ @ERG_BoinKy» @Venom_JasonC why do i need to tag the best canadien player in the game?

> 😞 🔁 0 ➡ @Sgperformer» @mile_hi_33 @joeyyeo13 @BocaRatonRC C looney is a giant Ass !!

> 😊 🔁 0 ➡ @DRCarbs» RT @RIDICULOUSNESS: Mood https://t.co/tOtbvcq8WK

> 😞 🔁 0 ➡ @javascriptd» RT @UriSamuels: #javascript windows.open will not work on Chrome console? #Tech #Internet #Question #HowTO https://t.co/IDkZTMcdWU

> 😒 🔁 0 ➡ @lofvesen89» Make-a-wish night planned at tonawanda high school _ march 23, 2016 _ www. kentonbee. com _ ken-ton bee

> 😊 🔁 0 ➡ @ewughno» Women with power and confidence are sexy. Insecurities aren't cute. Stop trying to make them be.

Merge & Join

01. **Tee**

02. **Wye**

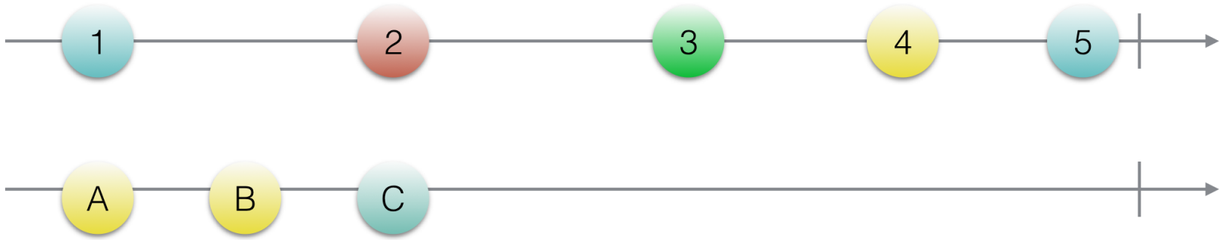
03. **`mergeN`**

```
@ def randomDelays[A](max: FiniteDuration): Channel[Task, A, A] =
    Process.constant {
      val delay = Task.delay(Random.nextInt(max.toMillis.toInt))
      (a: A) => delay.flatMap { d ⇒
        Task.schedule(a, d.millis)
      }
    }
defined function randomDelays
```

```
@ import tee._
@ import wye._

@ val p1: Process[Task, Int] = integerStream through randomDelays(3 seconds)
@ val p2: Process[Task, String] = emitAll(Seq("A", "B", "C")) through randomDelays(1 seconds)
```

Tee

- Deterministic
- Left and then Right



tee.interleave


```
@ val teeExample1 = (p1 tee p2)(interleave) to log
@ teeExample1.run.run
1
A
2
B
3
C
```

```
@ val teeExample2 = (p1 tee p2)(passR) to log

@ teeExample2.run.run
A
B
C
```

```
type Tee[-I,-I2,+O] = Process[Env[I,I2]#T, O]
```

Wye

- Non-deterministic
- Left, right or both

```
@ val wyeExample1 = (p1 wye p2)(wye.merge) to log
wyeExample1: Process[Task, Any] = ...
@ wyeExample1.run
A
B
1
C
2
...
```

```
type Wye[-I,-I2,+O] = Process[Env[I,I2]#Y, O]
```

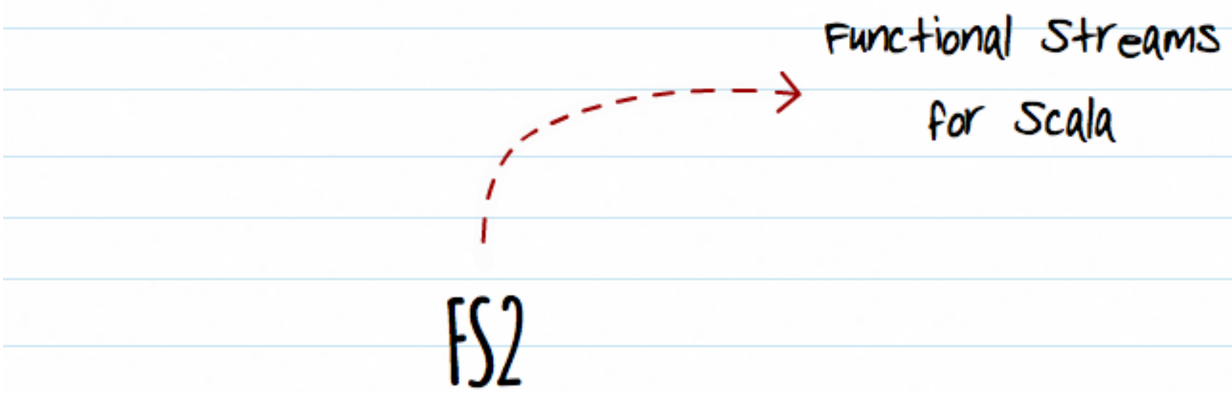
mergeN

```
def mergeN[A]: Process[Task, Process[Task, A]]) => Process[Task, A]
```

```
@ val p3: Process[Task, Int => Task[Int]] =
    Process.repeatEval(Task.delay{x: Int => Task{x * 10}})

@ val processOfNProcesses: Process[Task, Process[Task, Int]] =
    (p1 tee p3)(tee.zipApply).map(Process.eval)

@ val mergeNExample = merge.mergeN(2)(processOfNProcesses)
```

“There are two types of libraries: the ones people hate and the ones nobody use"

— Unknown


Functional Streams
for Scala
FS2

Slides & Code: http://github.com/adilakhter.

References & Further Readings

- FS2: The Official Guide
- Scalaz-Stream Masterclass by Runar at NE Scala 2016
- Scalaz Task - the missing documentation
- scalaz-stream User Notes
- Comparing akka-stream and scalaz-stream with code examples
- Additional Resources
- http://www.cwi.nl/lego-turing-machine