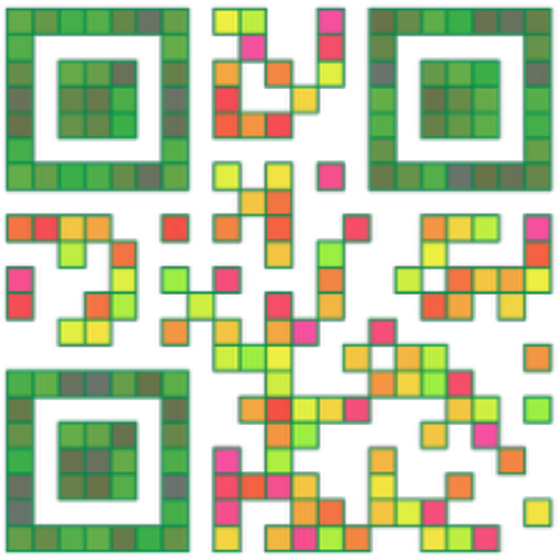
When code reacts to data

CodeMesh 4 Dec 2013



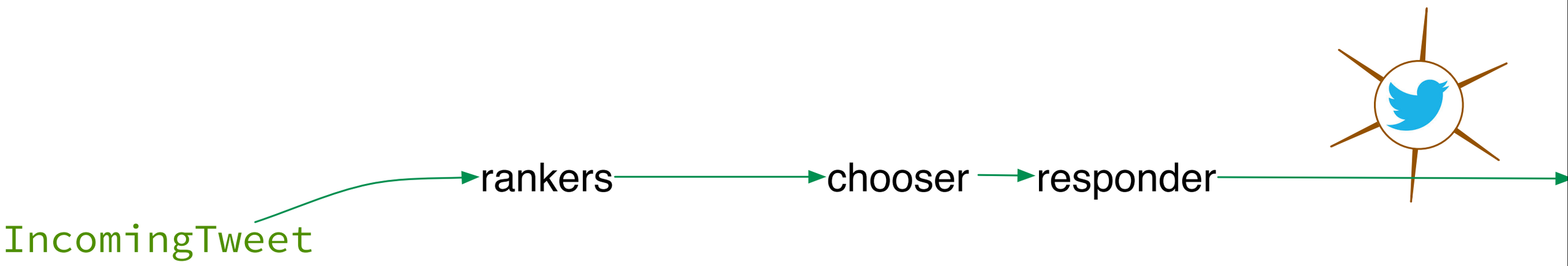
@jessitron

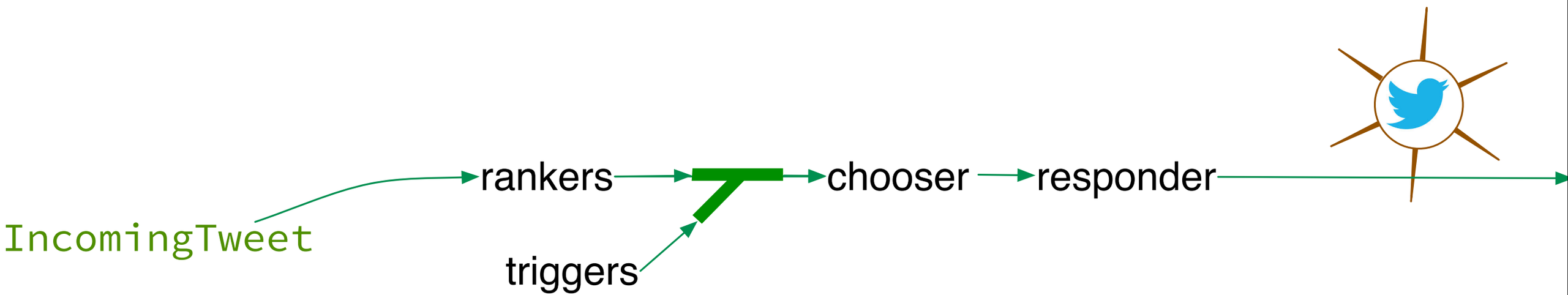

HYSTER

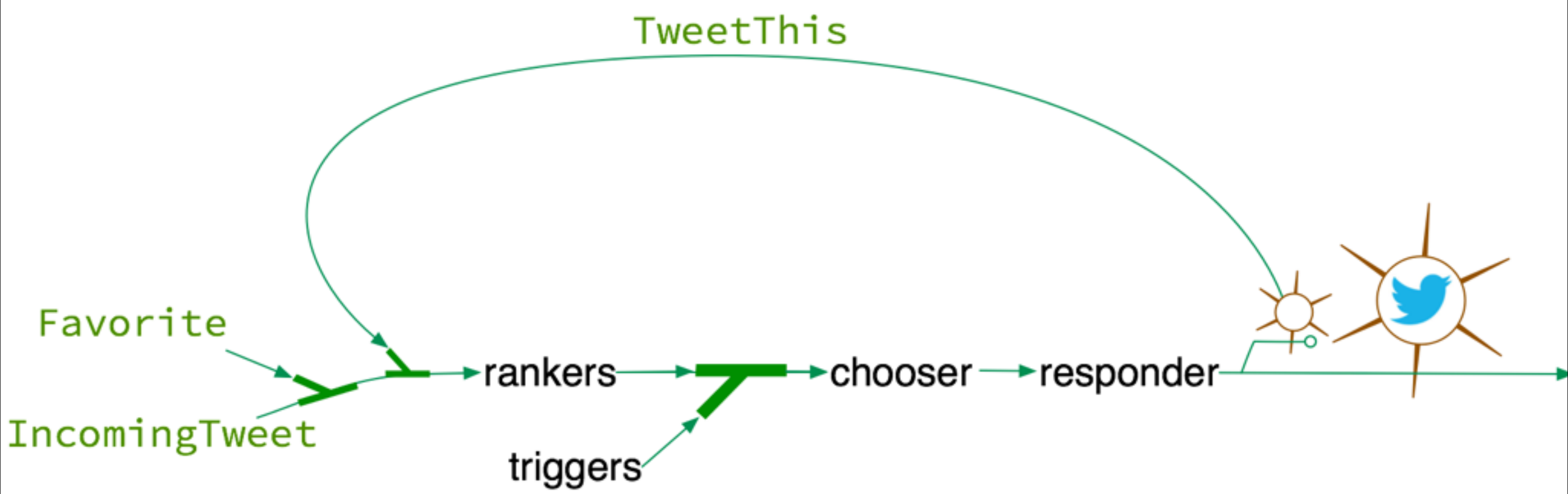
scalaz-stream

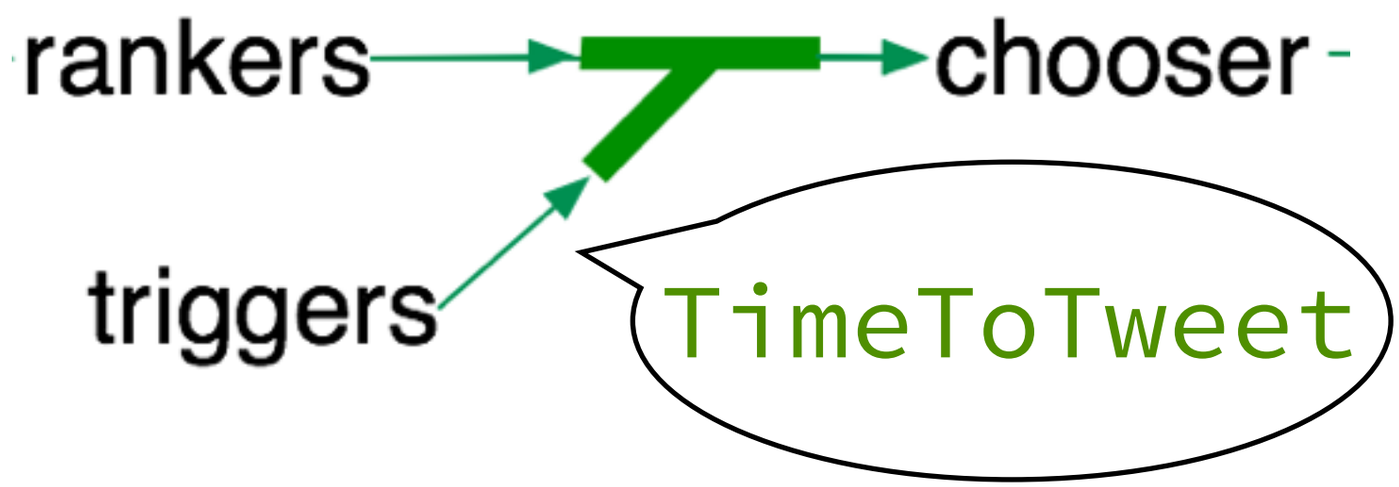
1. compositional
2. expressive
3. resource safe
4. fast

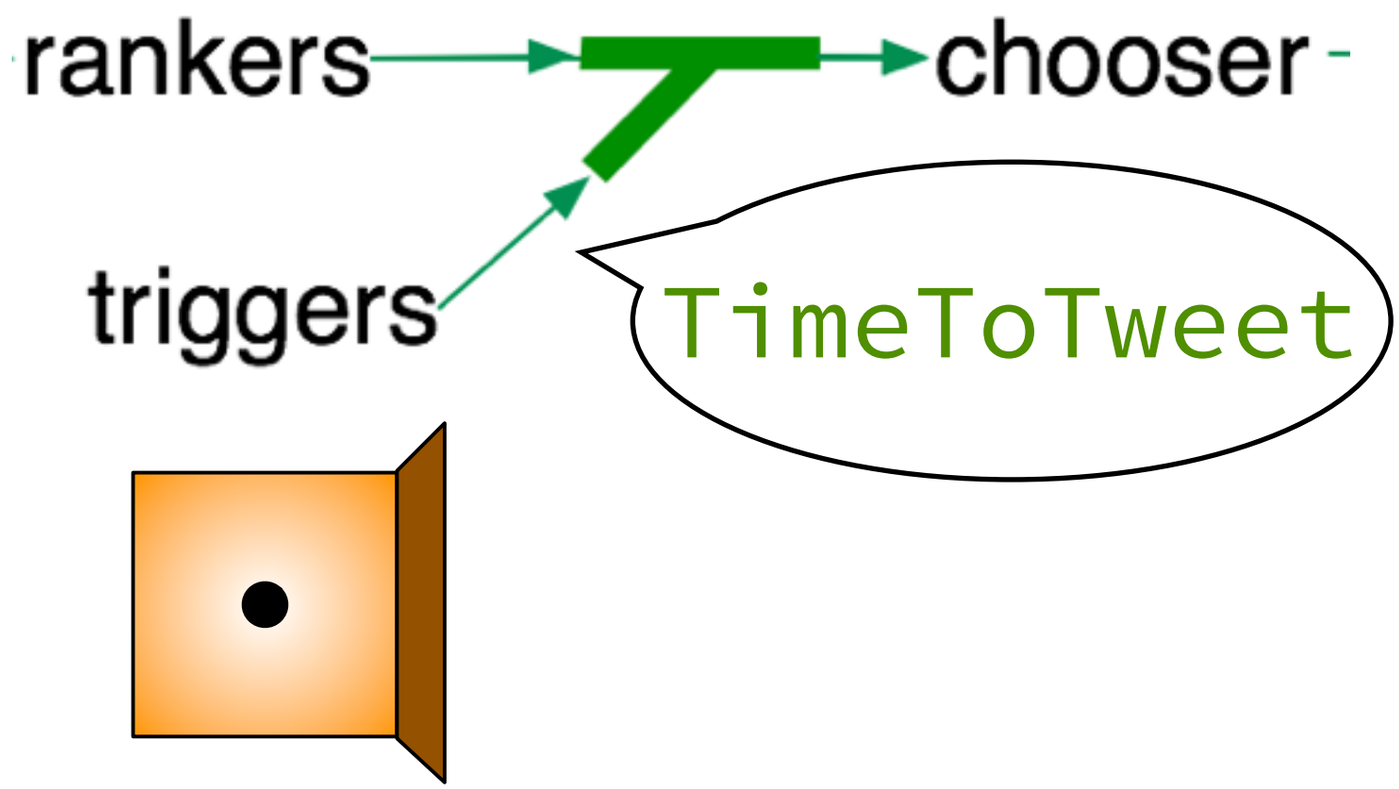

I agree
@Con4mationBison

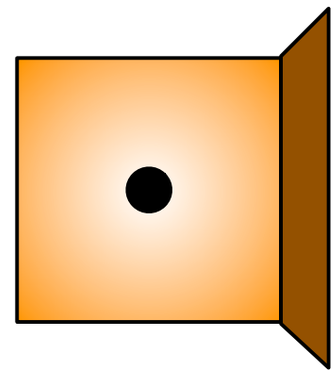

IncomingTweet
rankers
chooser
responder


rankers
chooser
responder
IncomingTweet
triggers

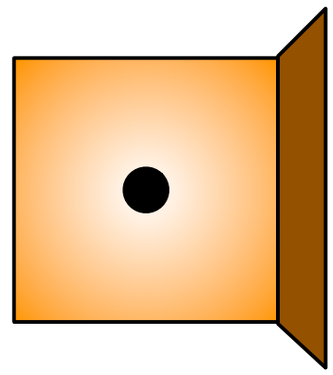

TweetThis
Favorite
IncomingTweet
rankers
triggers
chooser
responder


rankers
chooser
triggers
TimeToTweet

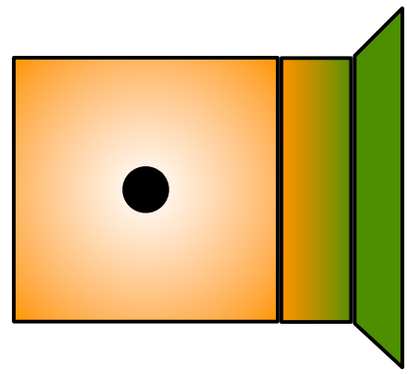

rankers
chooser
triggers
TimeToTweet



Process[+F[_],+0]



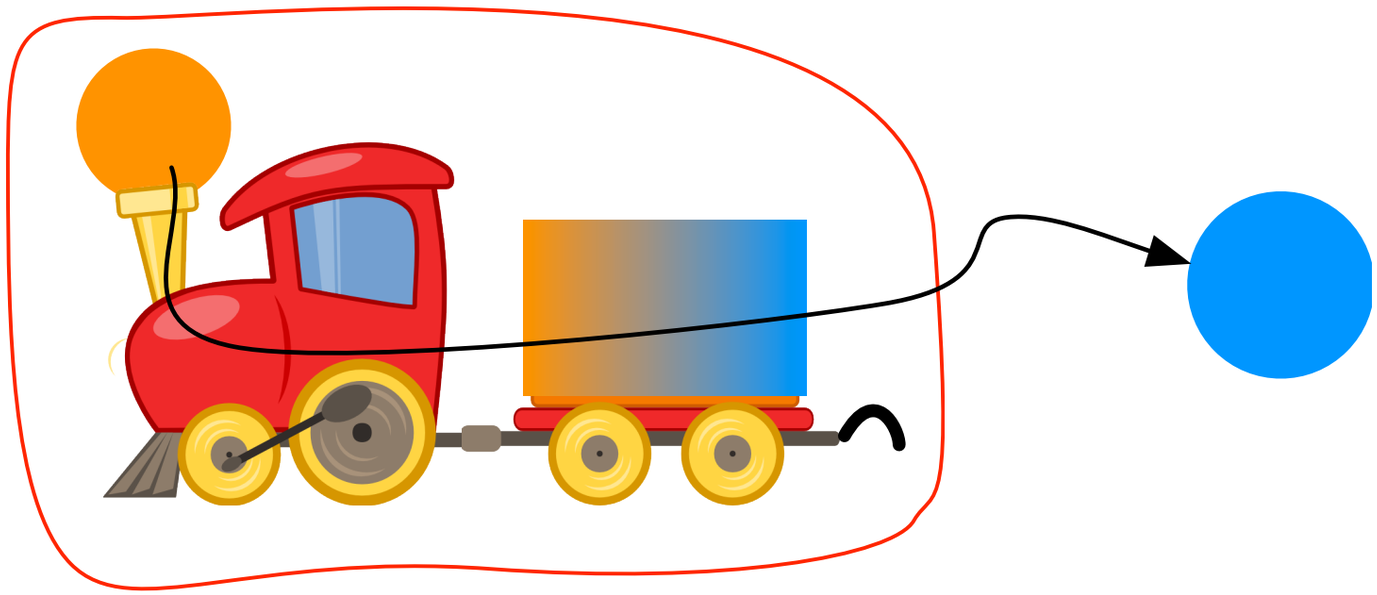
```
Process.awakeEvery(30 seconds):
  Process[Task,Duration]
```



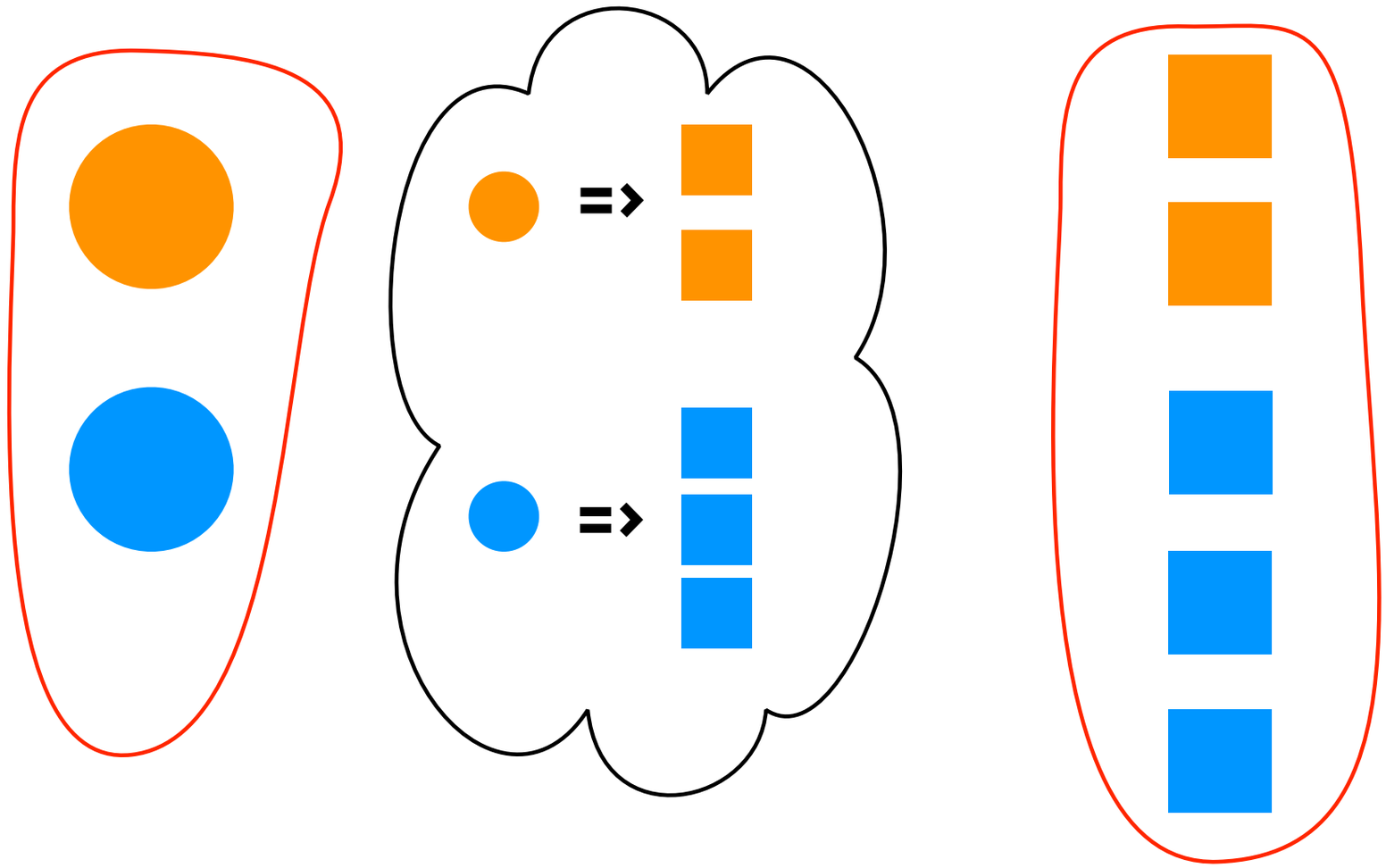
```
Process.awakeEvery(30 seconds).
  map { _:Any => TimeToTweet}
  : Process[Task,TimeToTweet]
```

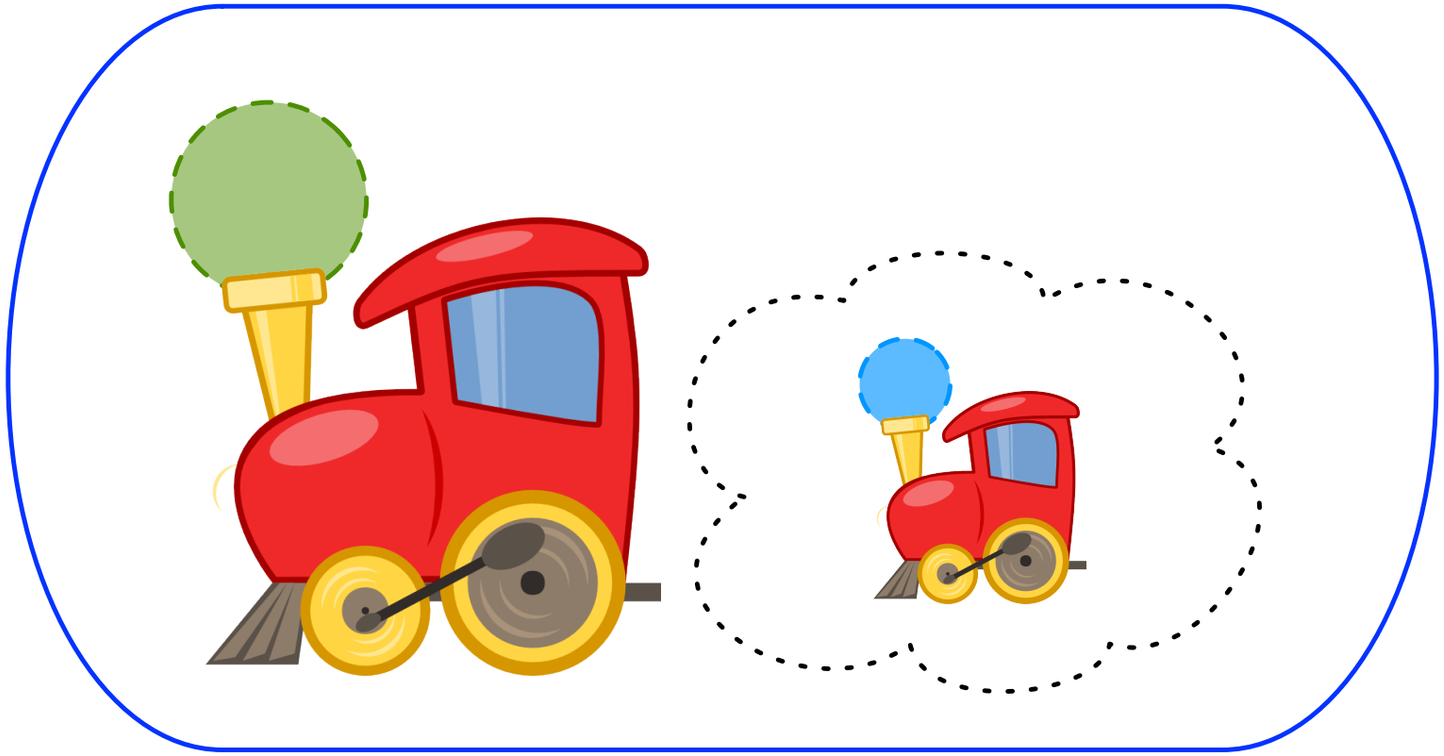
List[A]

List[B]



List[A] **flatmap** => List[B]

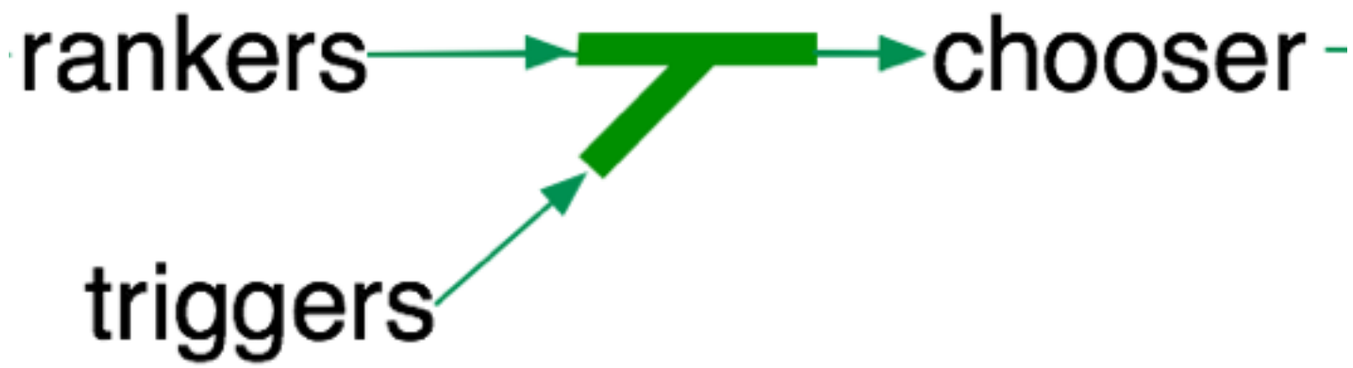





I don't care what your special power is.
Here's your mission, now get it done

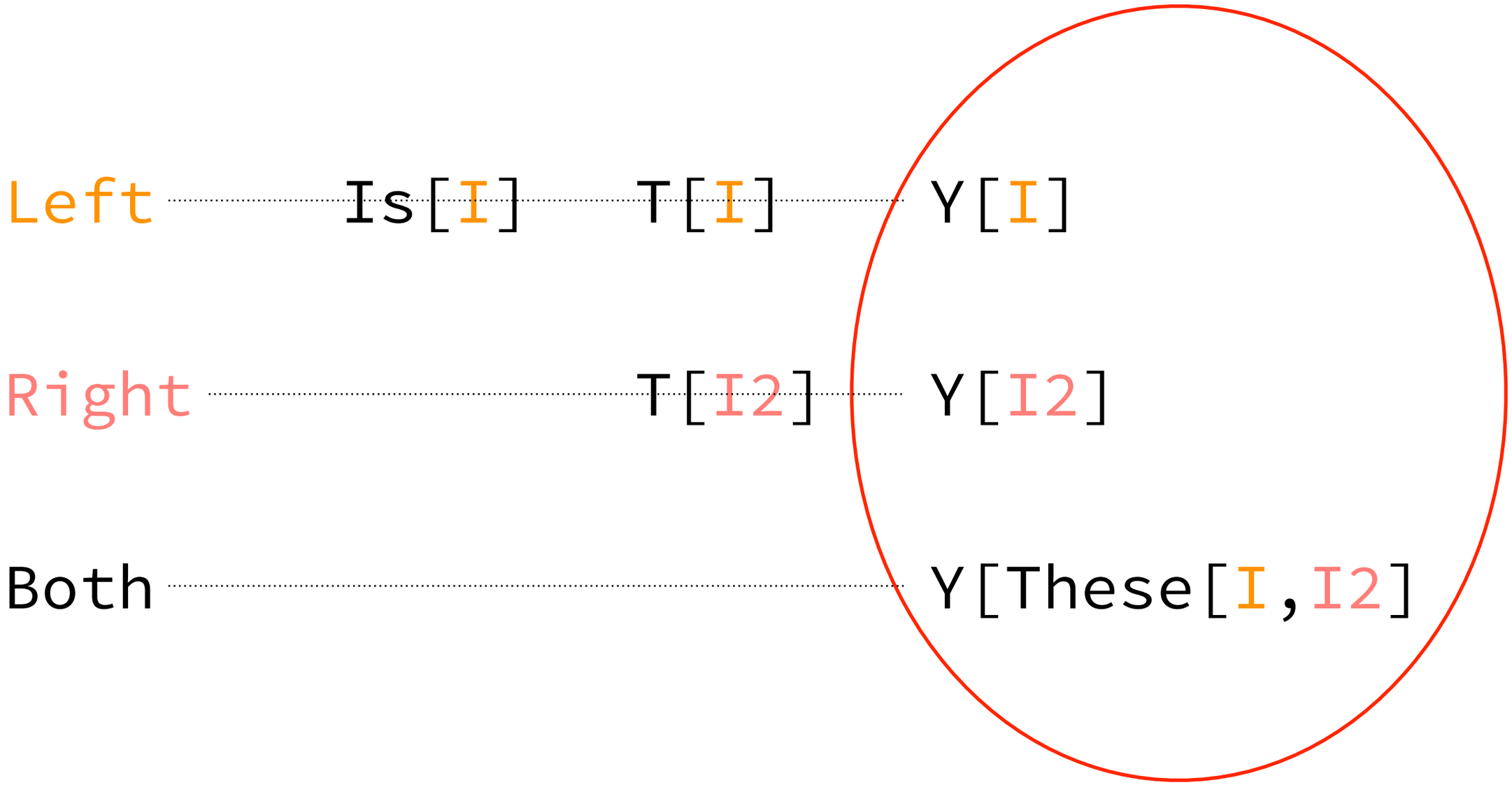
```
 /**
   * Collect the outputs of this Process,
   * given a Monad[F] in which
   * we can catch exceptions…
   */
def runLog(implicit F: Monad[F],
                    C: Catchable[F]): F[Seq[O]]
```

```
(implicit F: Monad[F],
          C: Catchable[F])
```

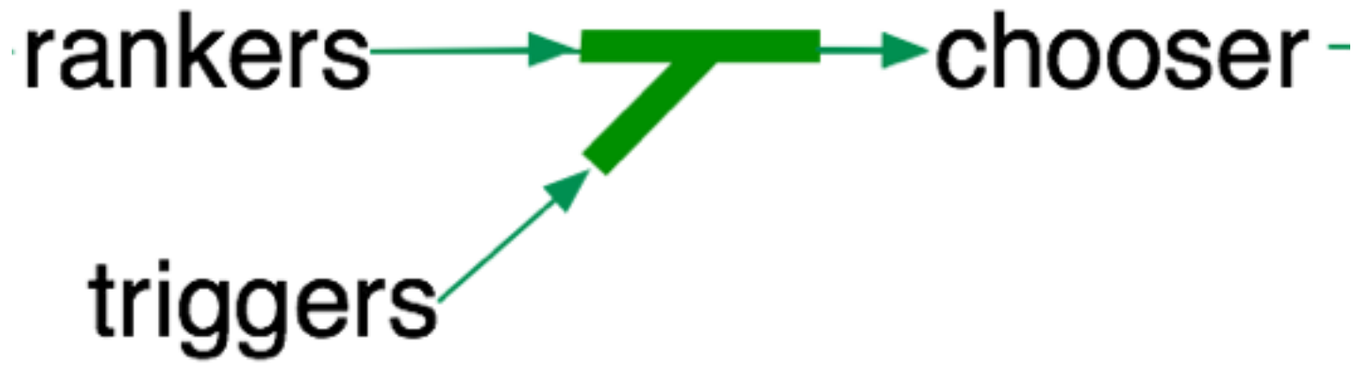

rankers
chooser
triggers

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

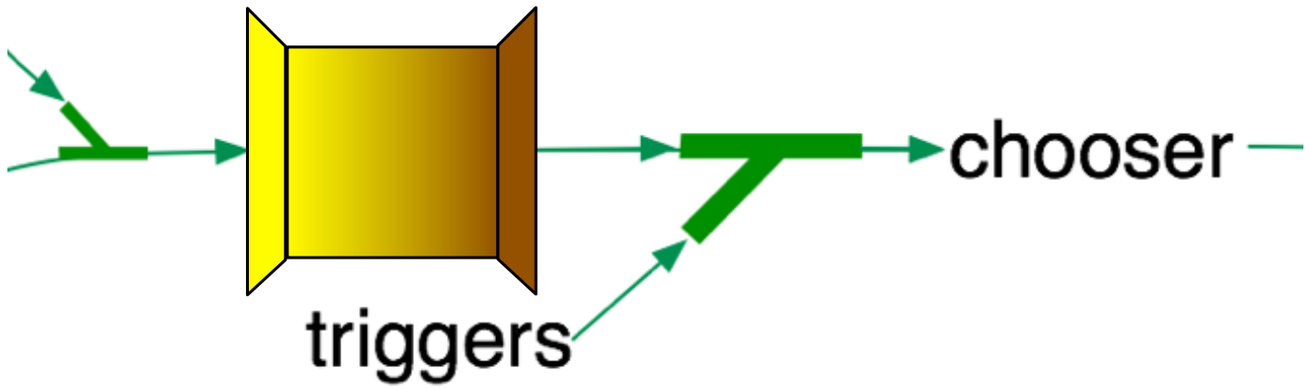
```
type
Wye[-I,-I2,+O] = Process[Env[I,I2]#Y,O]


case class Env[-I,-I2]() {
  sealed trait Y[-X]
  sealed trait T[-X] extends Y[X]
  sealed trait Is[-X] extends T[X]

  case object Left extends Is[I]
  case object Right extends T[I2]
  case object Both extends Y[These[I,I2]]
}
```


Left
Is[I]
T[I]
Y[I]
Right
T[I2]
Y[I2]
Both
Y[These[I,I2]

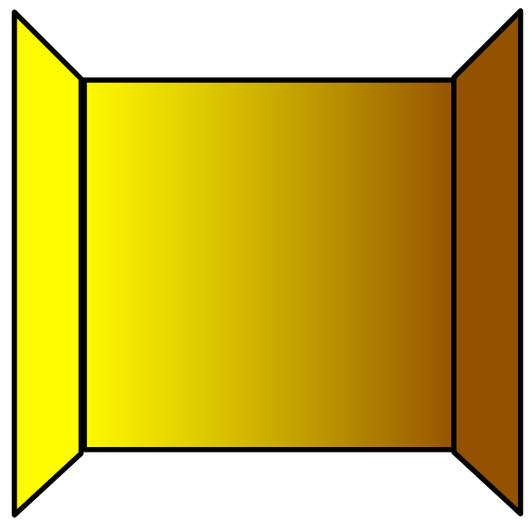
```
Wye[Message, Message, Message]
```



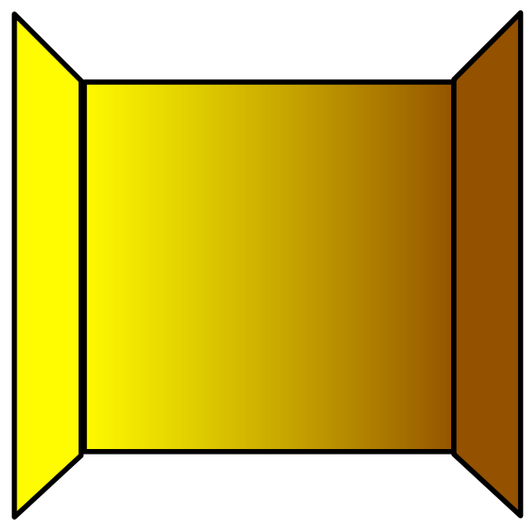

```
ranker.wye(triggers)(wye.merge)
```

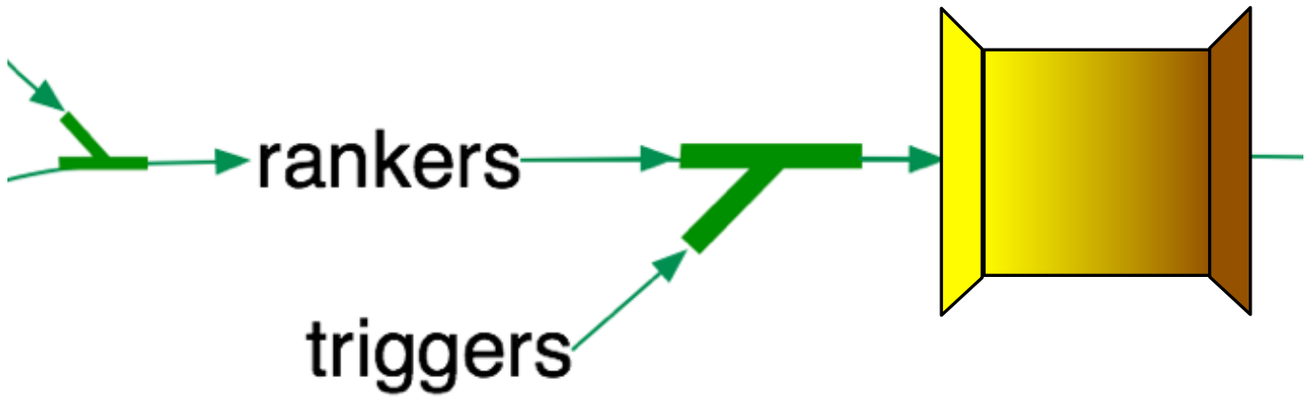

chooser
triggers

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



```
val ranker: Process1[Message,Message] =
  process1.lift {
    case i: IncomingTweet =>
     i.addOpinion(Random.nextDouble,"I agree!")
    case m => m
}
```




rankers
triggers

give me your data!

```
iterator.next()
```

Do this with your data!

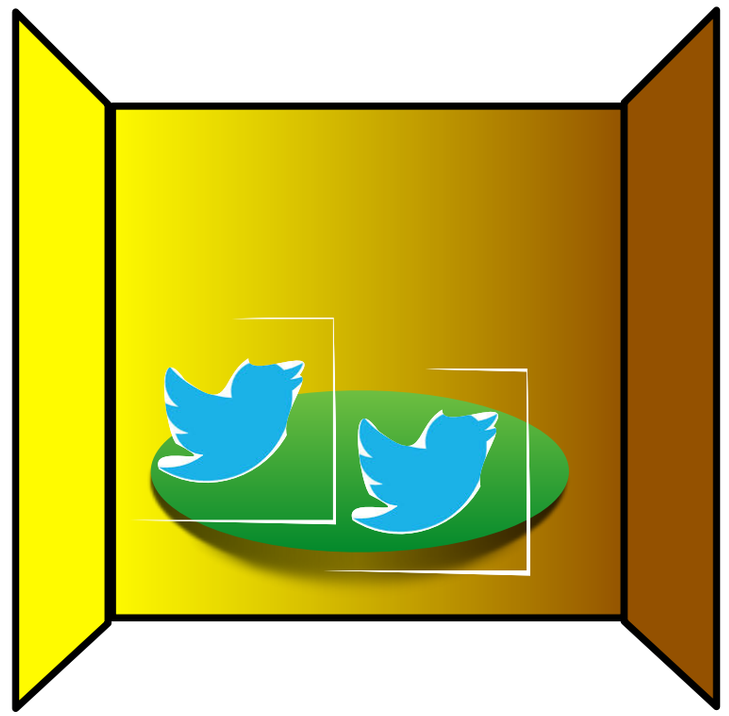
```
list.map(a => b)
```

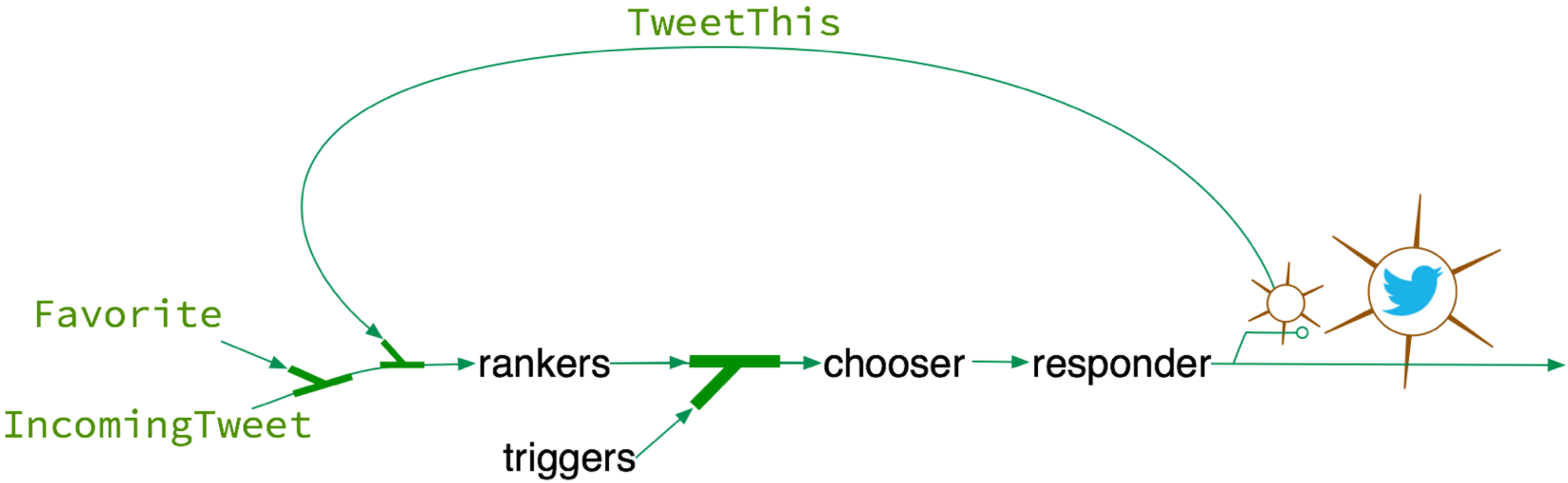
Let's talk about your data.

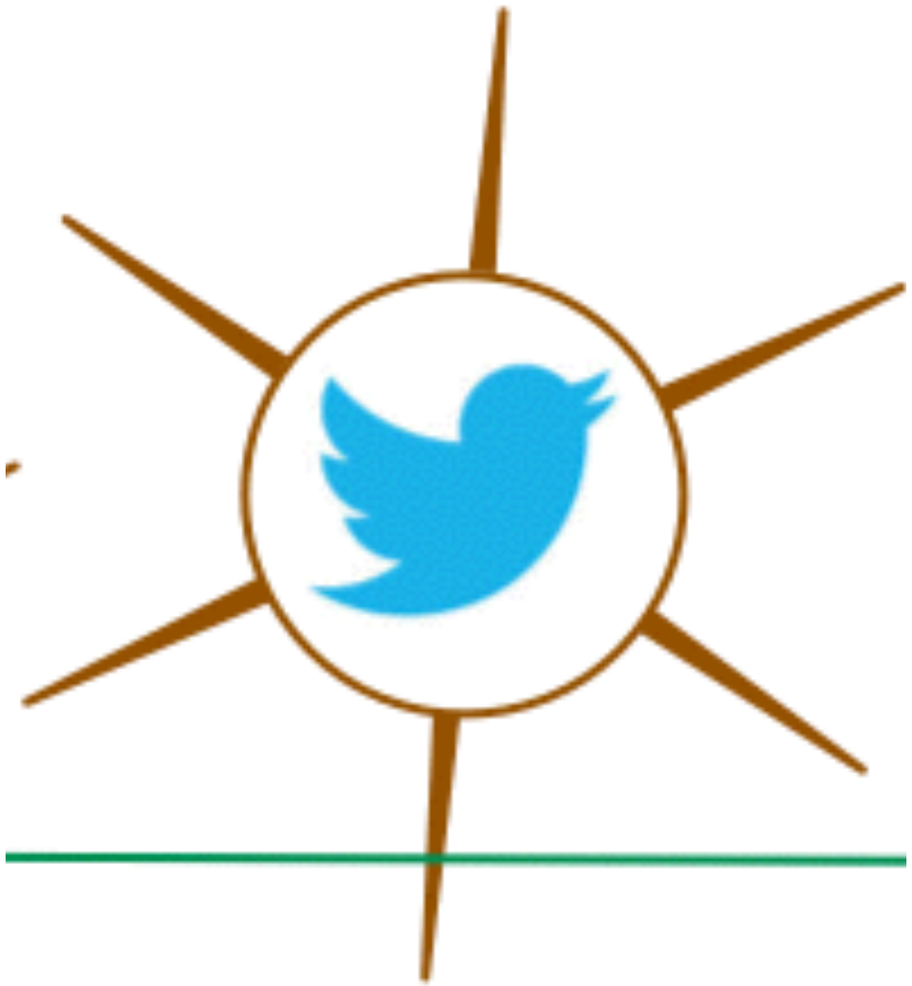
```
f: elem => Instruction
```

```
Halt(cause: Throwable)


Emit(
     head: Seq[O],
     tail: Process[F,O])


Await[F[_],A,+O](
     req: F[A],
     recv:  A => Process[F,O],
     fallback1:  Process[F,O] = halt,
     cleanup1:   Process[F,O] = halt)
```

```
def tweetPicker = {
  def go(pool: TweetPool): Process1[Message, Message] =
    await1[Message] flatMap {
      case i: IncomingTweet => go(pool.absorb(i))
      case TimeToTweet =>
        val (newMessage, newPool) = pool.findBest
        emit(newMessage) ++ go(newPool)
      case m => emit(m) ++ go(pool)
    }
  go(new TweetPool())
}
```

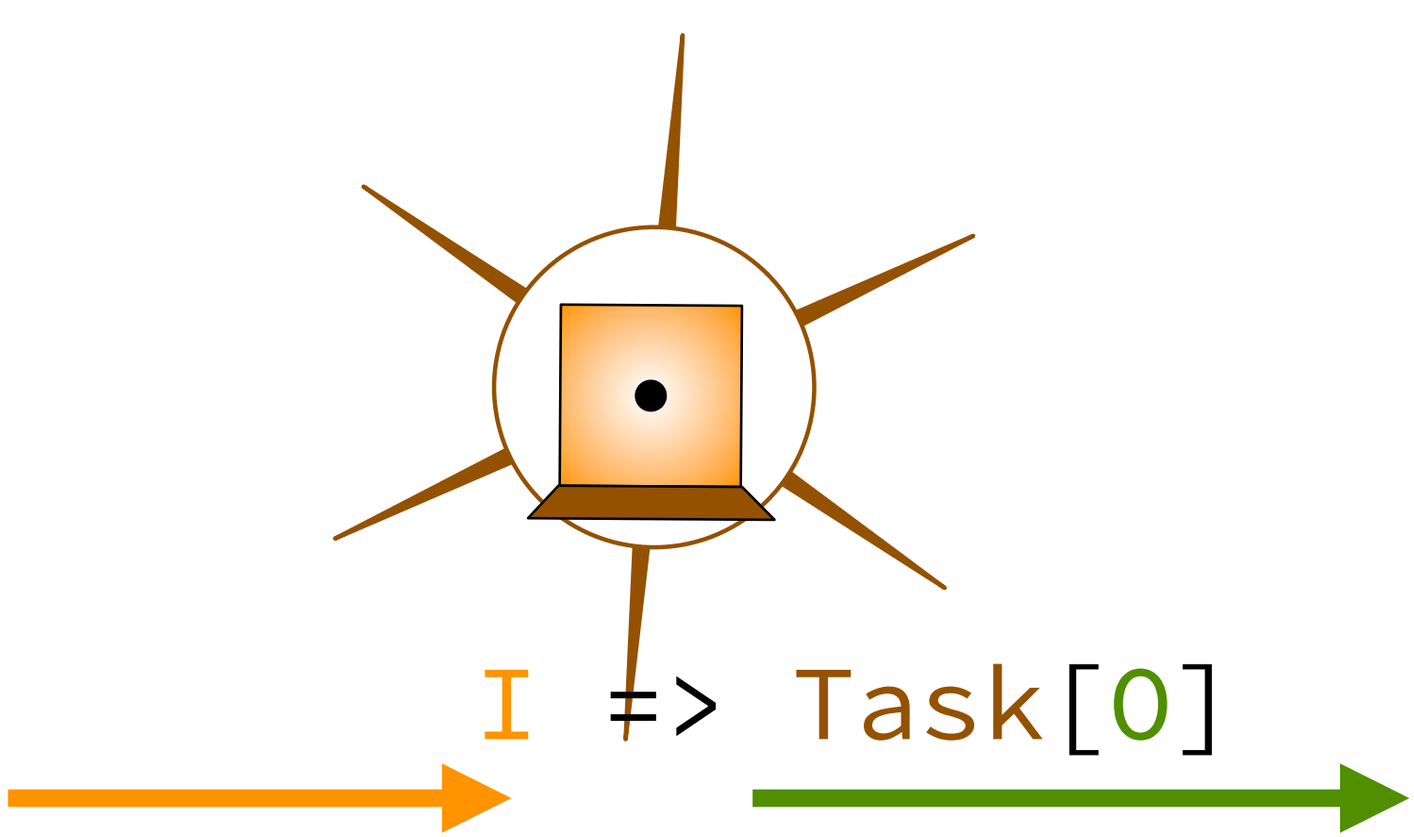


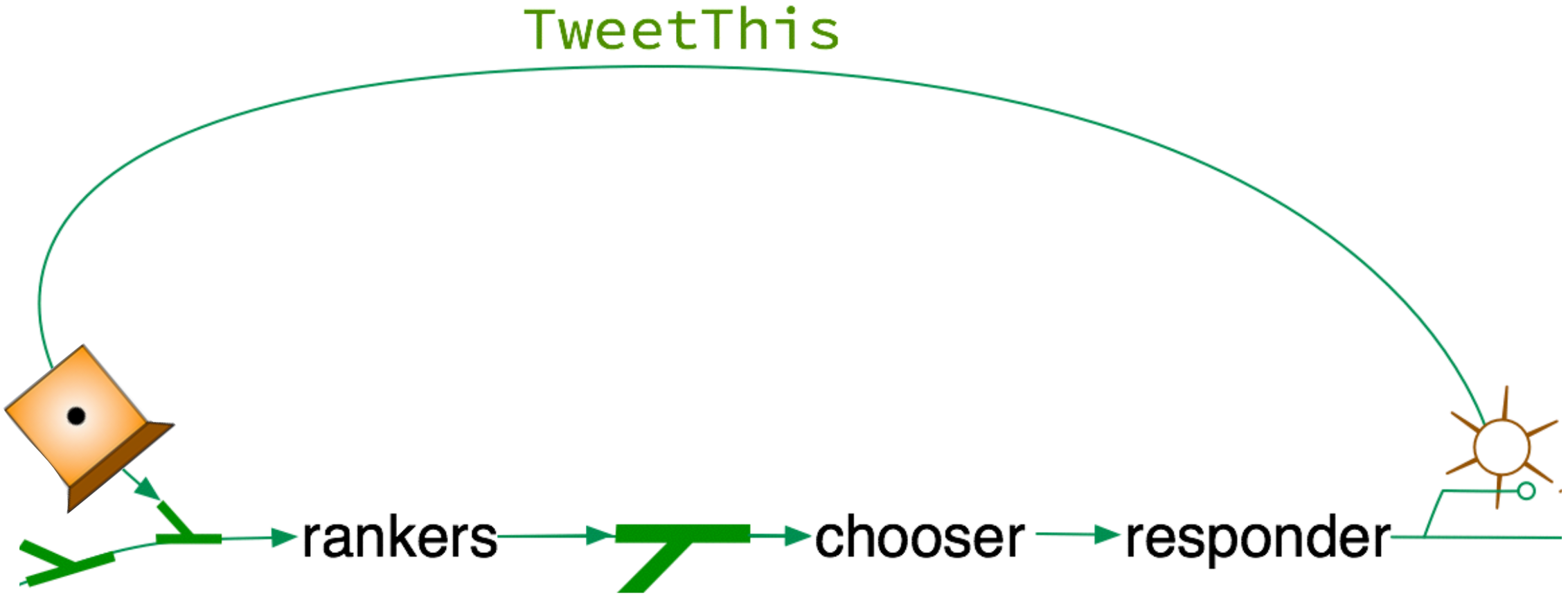

TweetThis
Favorite
IncomingTweet
rankers
triggers
chooser
responder

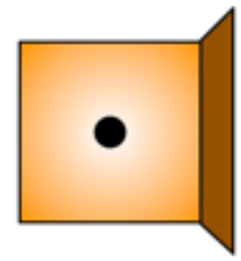


```
type Channel[+F[_],-I,O] = Process[F,I=>F[O]]
```

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

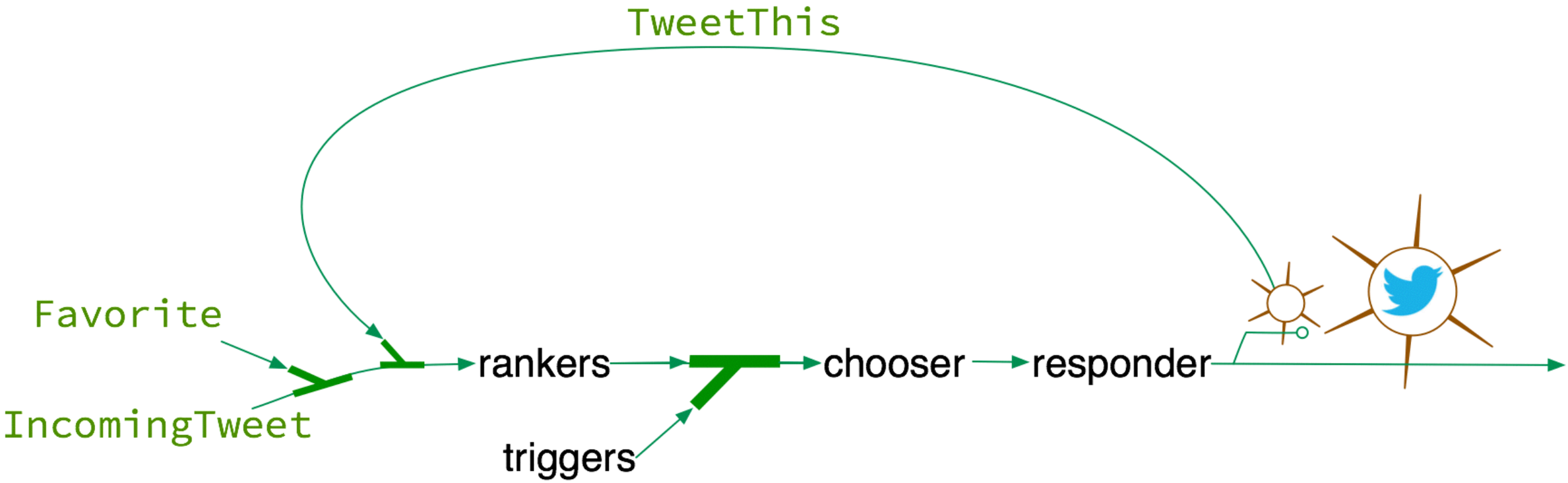


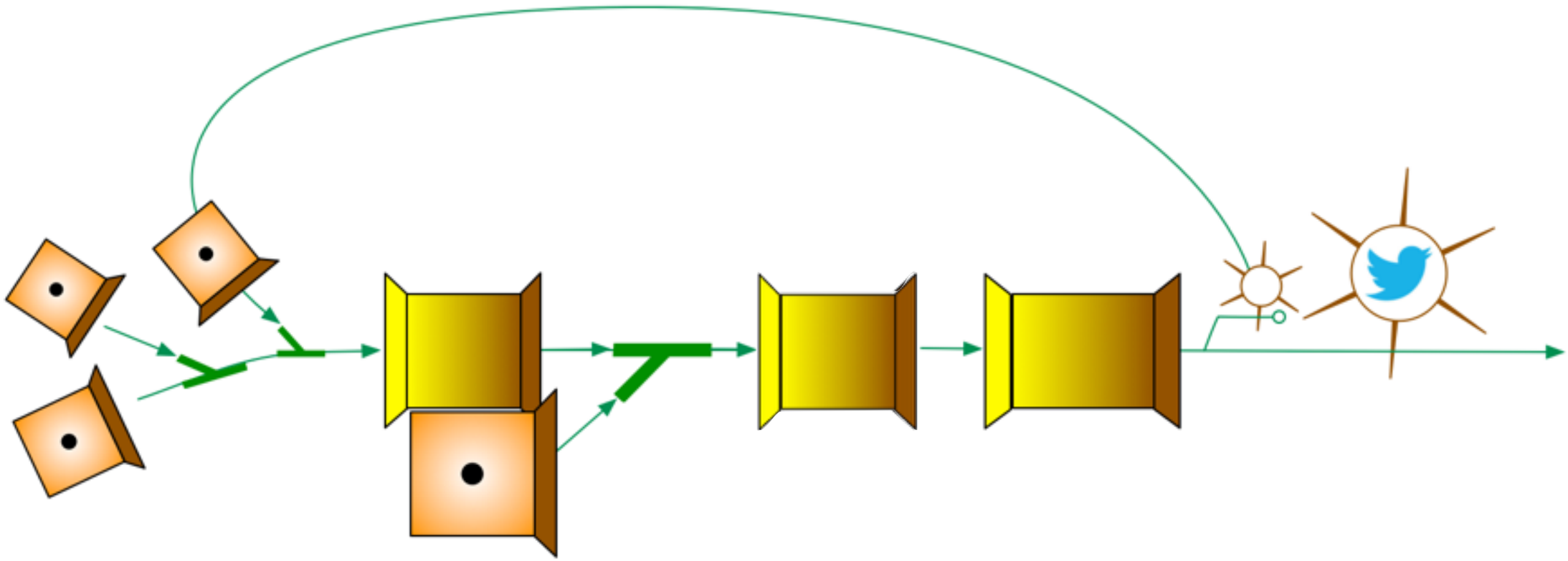

TweetThis
rankers
chooser
responder

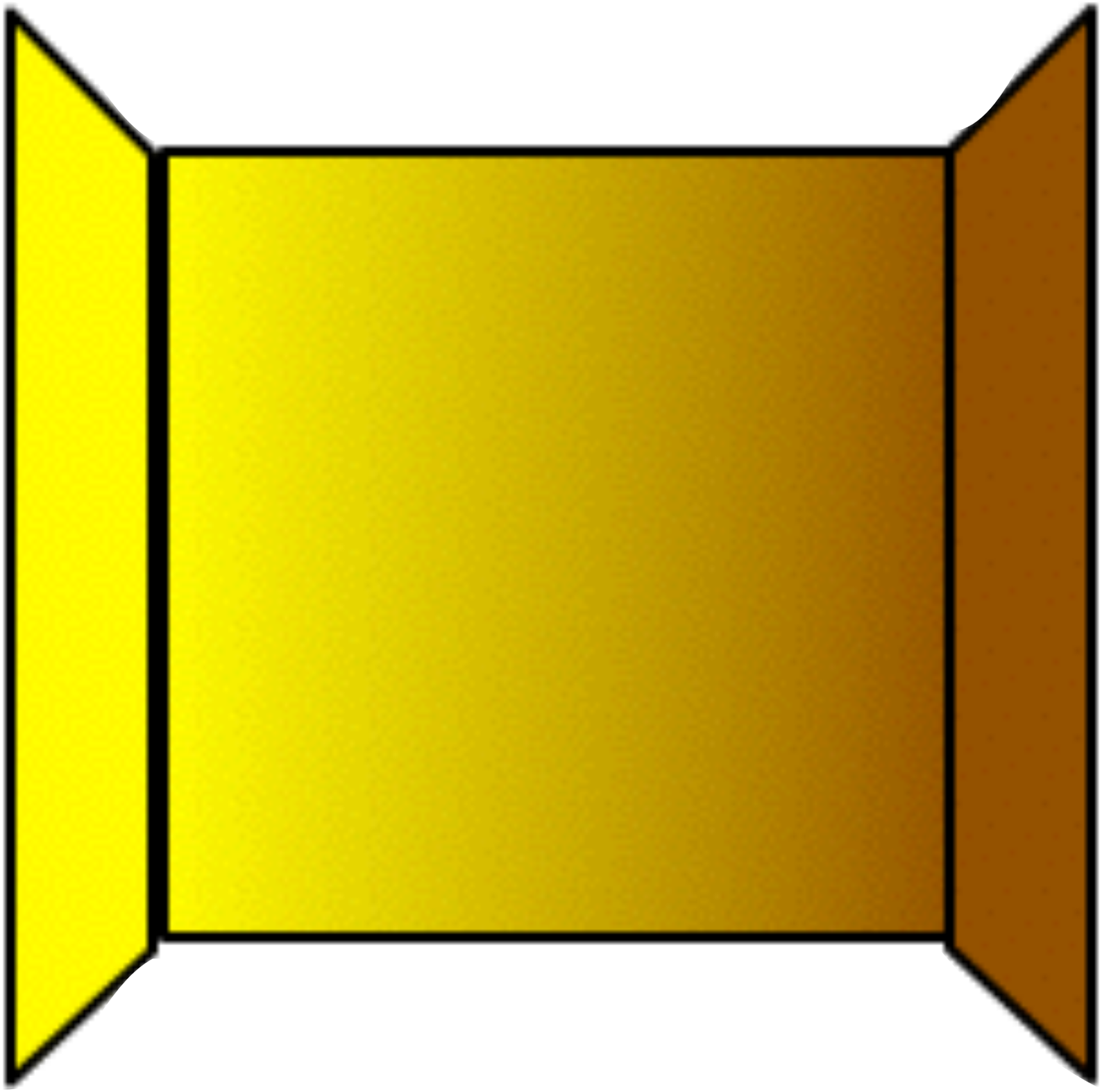


```
async.queue:(Queue[A], Process[Task,A])
```

```
async.toSink(Queue[A]): Sink[Task,A]
```

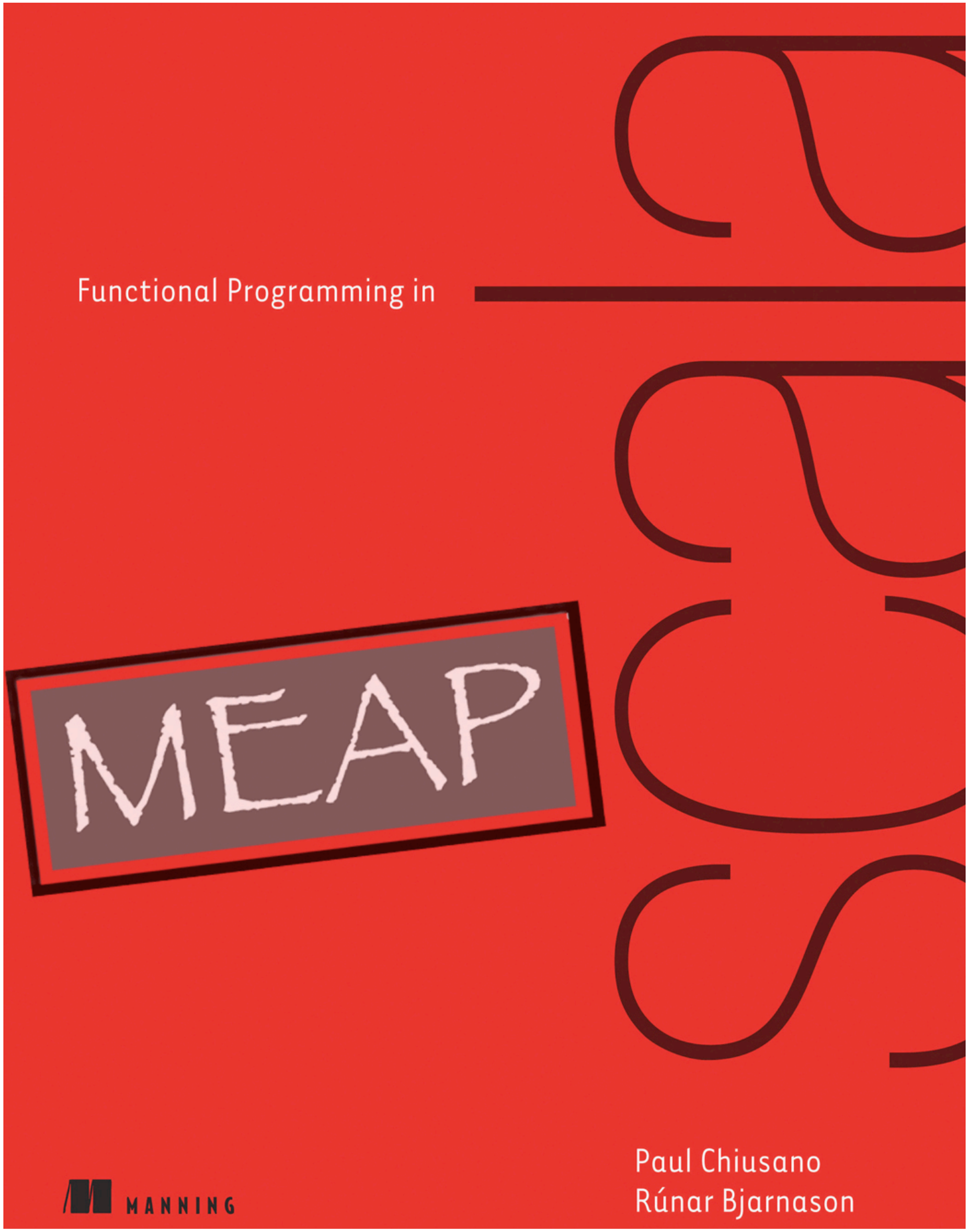

TweetThis
Favorite
IncomingTweet
rankers
triggers
chooser
responder

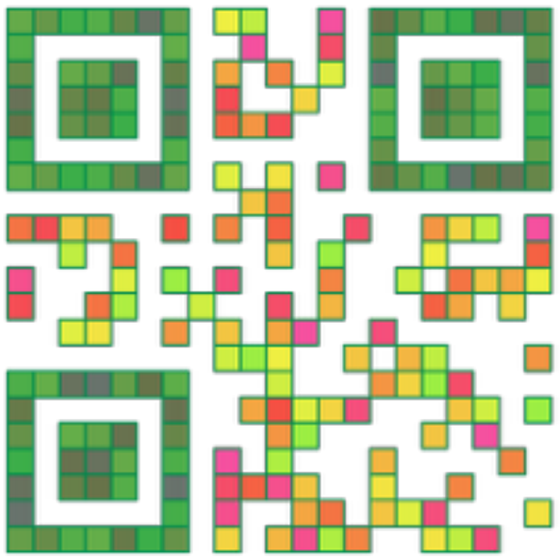




scalaz-stream

1. compositional
2. expressive
3. resource safe
4. fast


Functional Programming in
MEAP
MANNING
Paul Chiusano
Rúnar Bjarnason



Jessica Kerr

blog.jessitron.com

@jessitron

github.com/jessitron/bison

github.com/scalaz/scalaz-stream