REACTIVE STREAMS

THE GLUE OF THE REAL-TIME ORGANIZATION

WHAT TO EXPECT

» Introduction

» A Journey into Reactive Streams

» Diving into Akka Streams

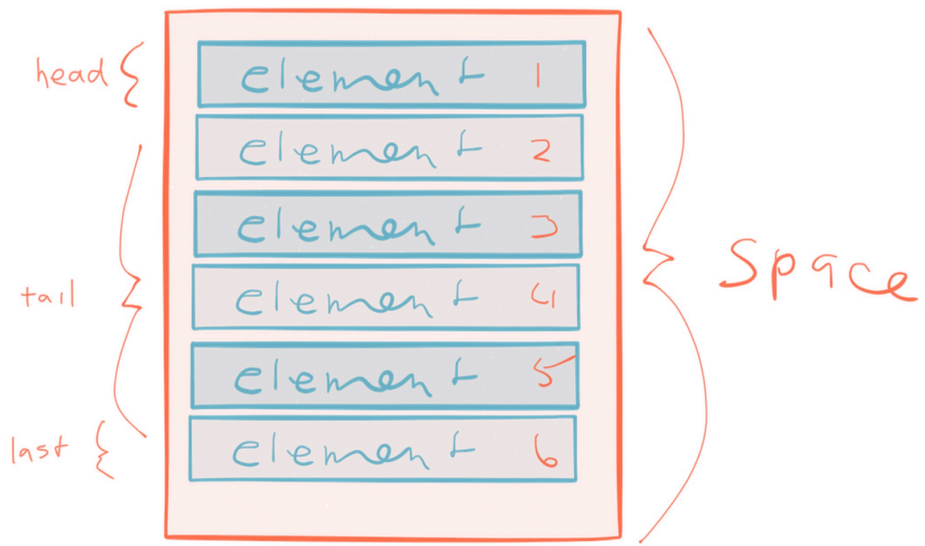
» Code walkthrough and demo

» Q&A

“IF ANALYTICS ARE THE BRAIN OF AN ORGANIZATION, STREAMS ARE THE HEART.”

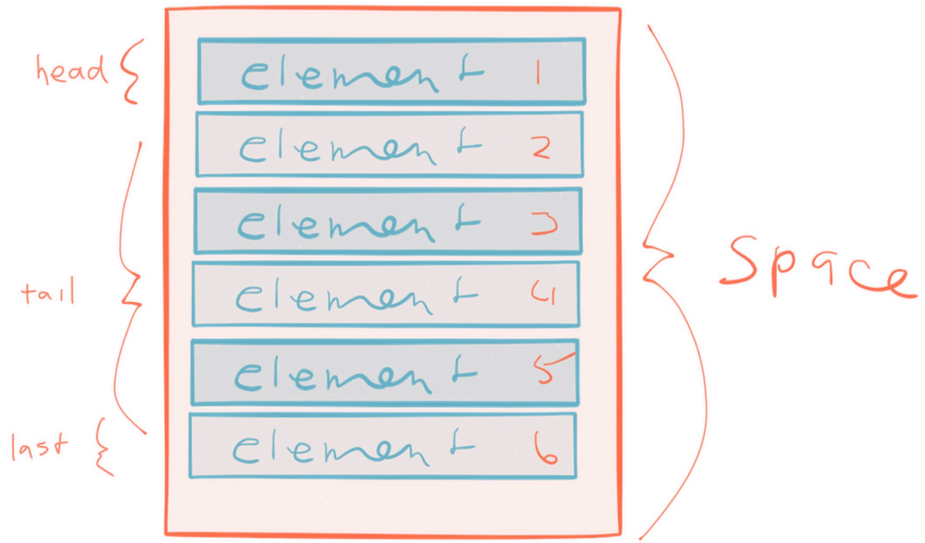
AN INTRODUCTION

AN **ARRAY** IS A COLLECTION OF ELEMENTS...



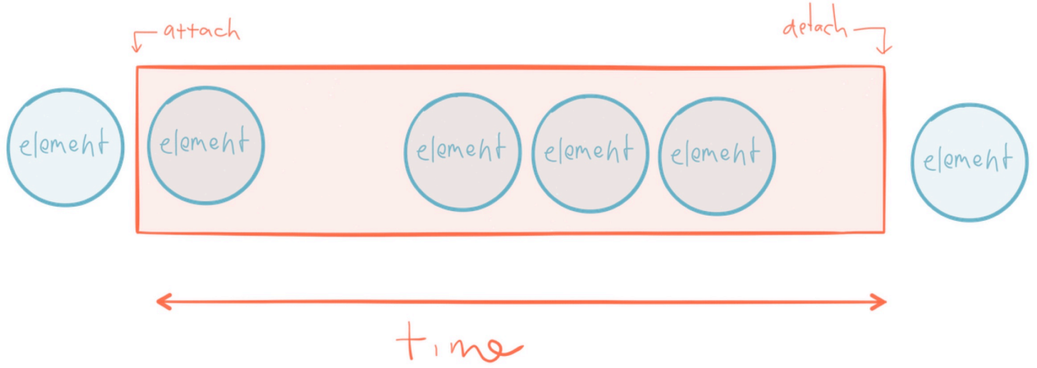
... ARRANGED IN **MEMORY**.

AN **ARRAY** IS A COLLECTION OF ELEMENTS...



... ARRANGED IN **MEMORY**.

A **STREAM** IS A SERIES OF ELEMENTS...



... EMITTED OVER **TIME**.

APPEAL OF STREAMS?

» Per-event processing (≤ 1 second latencies)

» Mini-batch processing (≤ 10 second latencies)

» Batch processing (≤ 1 hour latencies)

» Monitoring, analytics, complex event processing

CHALLENGES?

» Ephemeral

» Unbounded in size

» Potential flooding

» Unfamiliar

REACTIVE STREAMS

REACTIVE STREAMS?

» Reactive Streams is a specification and low-level API for library developers

REACTIVE STREAMS?

» Reactive Streams is a specification and low-level API for library developers

» Started as an initiative in late 2013 between engineers at Netflix, Pivotal, and Lightbend

REACTIVE STREAMS?

- Reactive Streams is a specification and low-level API for library developers
- Started as an initiative in late 2013 between engineers at Netflix, Pivotal, and Lightbend
- Aimed to address two critical challenges
  - Interoperability between streaming libraries
  - Flow control over an asynchronous boundary

WHAT IS REACTIVE STREAMS?

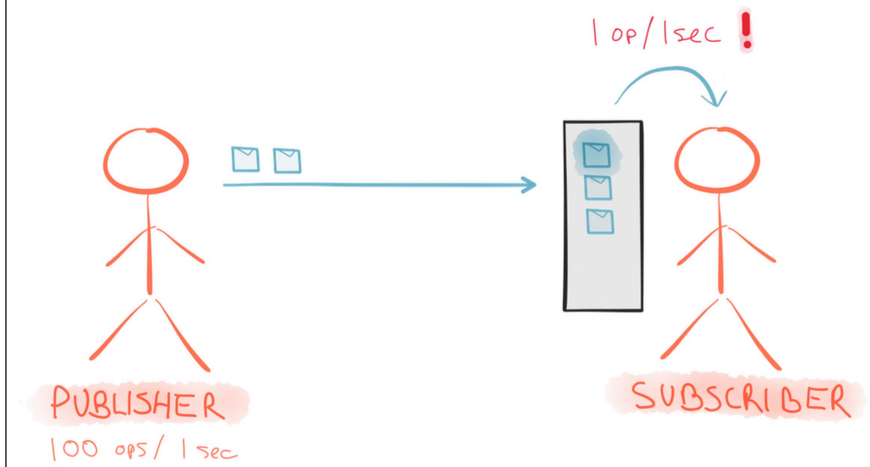
» TCK (Technology Compatibility Kit)

» API (JVM, JavaScript)

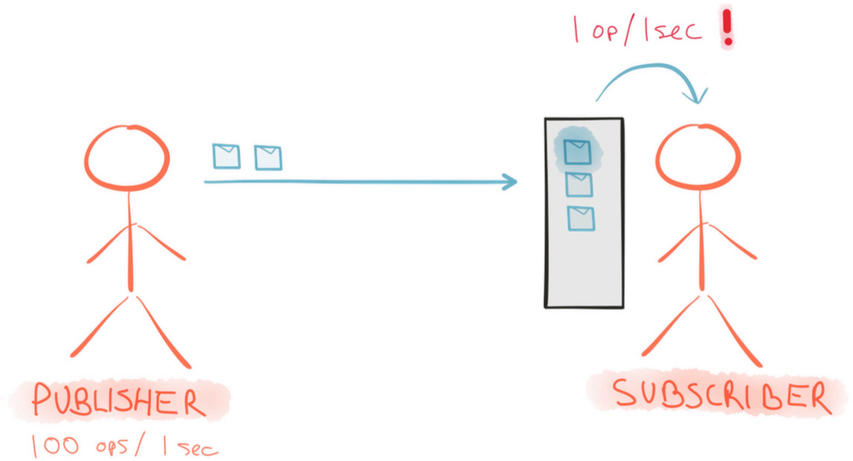
» Specifications for library developers

FLOW CONTROL

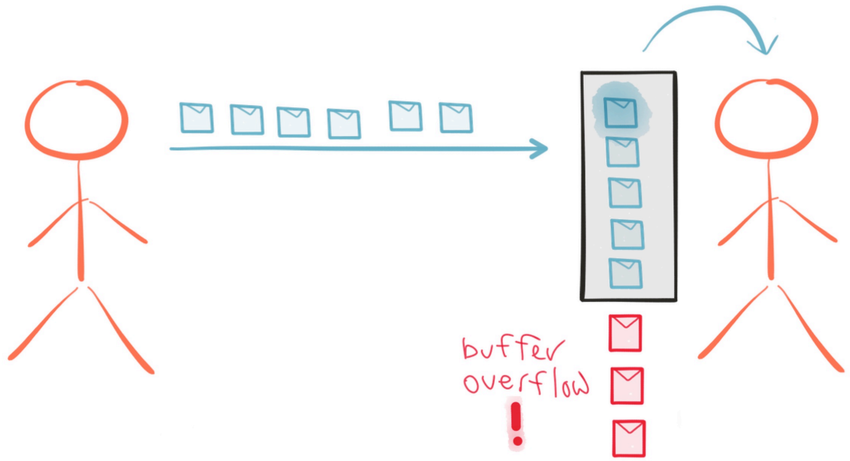
A SUBSCRIBER USUALLY HAS SOME TYPE OF **BUFFER**.



A SUBSCRIBER USUALLY HAS SOME TYPE OF **BUFFER**.

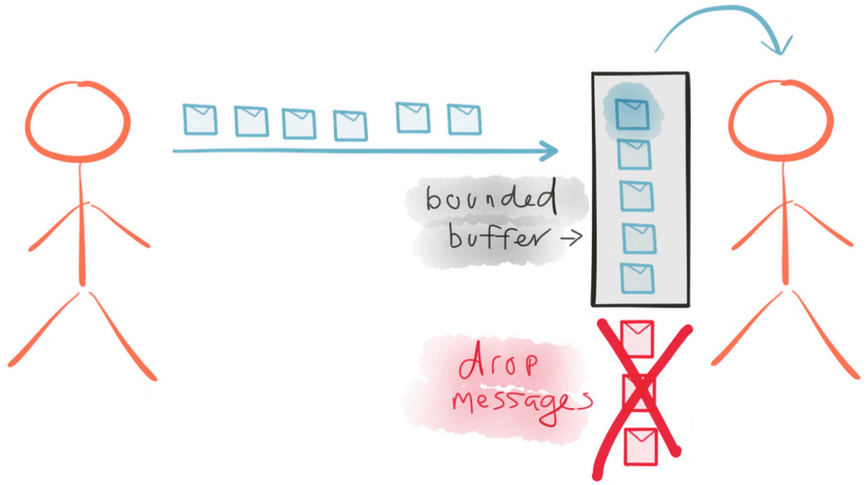


A **FASTER** PUBLISHER CAN EASILY OVERWHELM THE BUFFER OF A SLOW SUBSCRIBER...



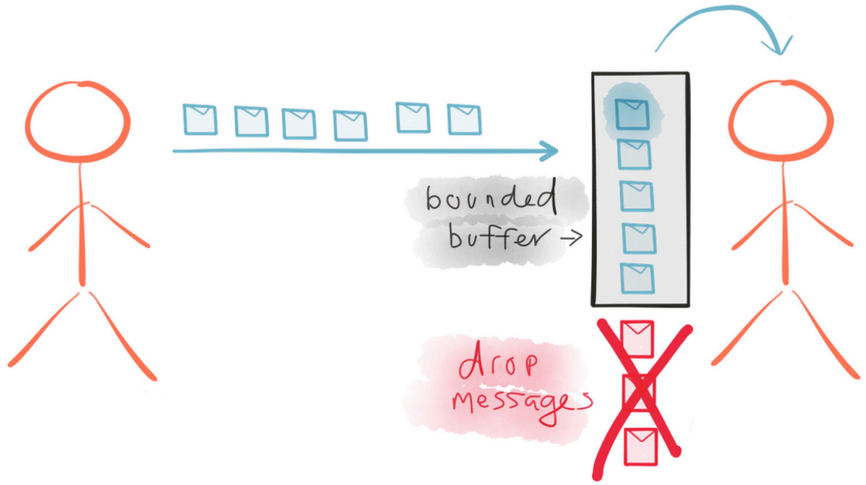
... WHICH CAN LEAD TO A **CASCADING FAILURE** THROUGH THE ENTIRE SYSTEM.

OPTION 1: DROP MESSAGES.



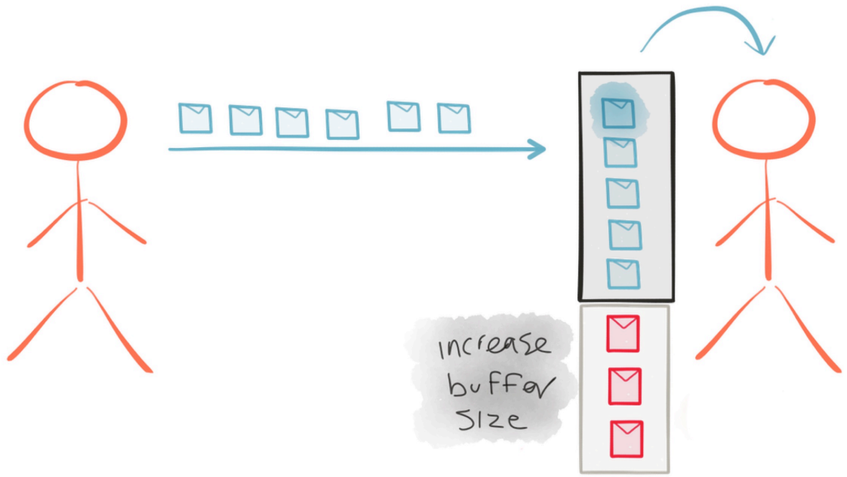
WORKS WELL FOR MONITORING AND OTHER EPHEMERAL TASKS.

OPTION 1: DROP MESSAGES.



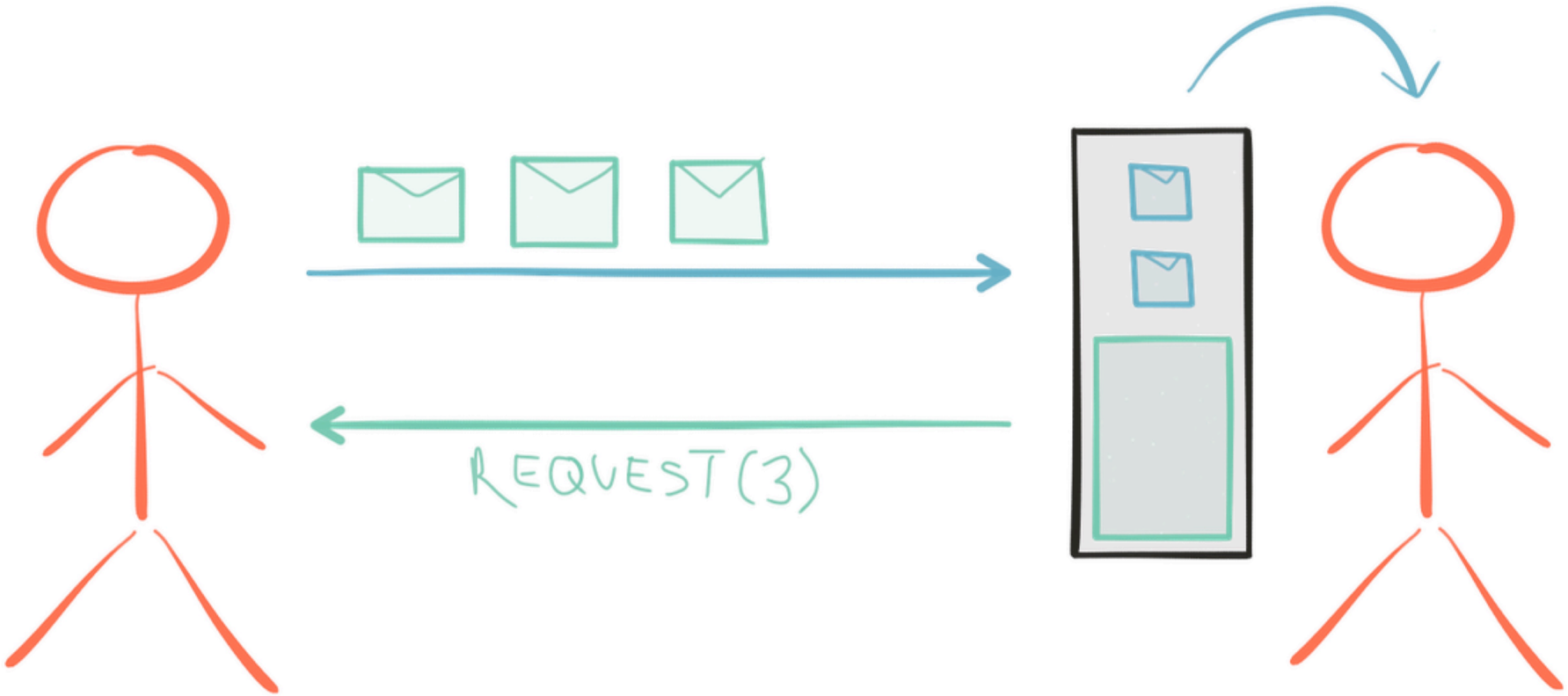
WORKS WELL FOR MONITORING AND OTHER EPHEMERAL TASKS.

OPTION 2: INCREASE BUFFER SIZE.



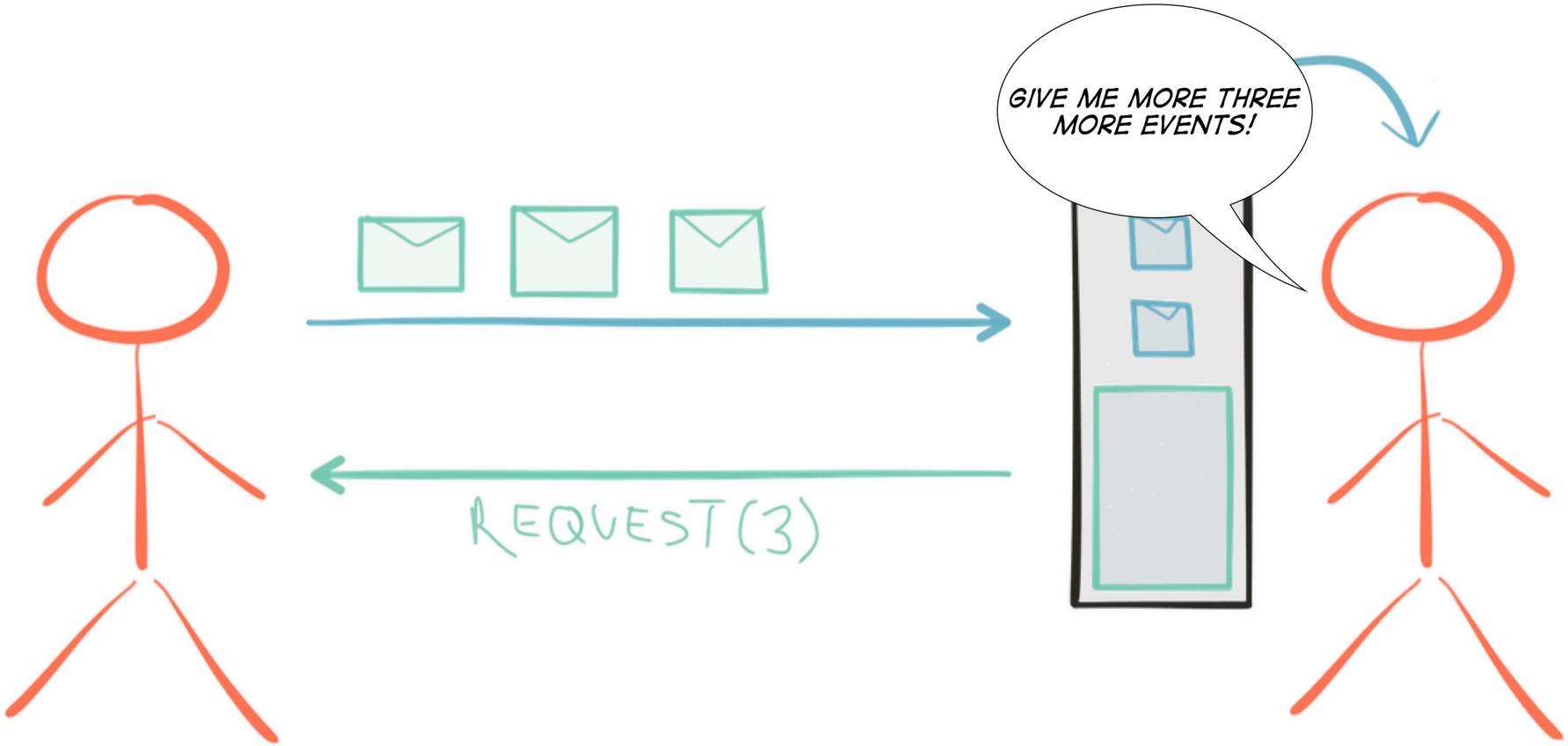
WORKS WELL UNTIL YOU RUN OUT OF MEMORY!

IDEAL SOLUTION: **PULL-BASED BACKPRESSURE.**



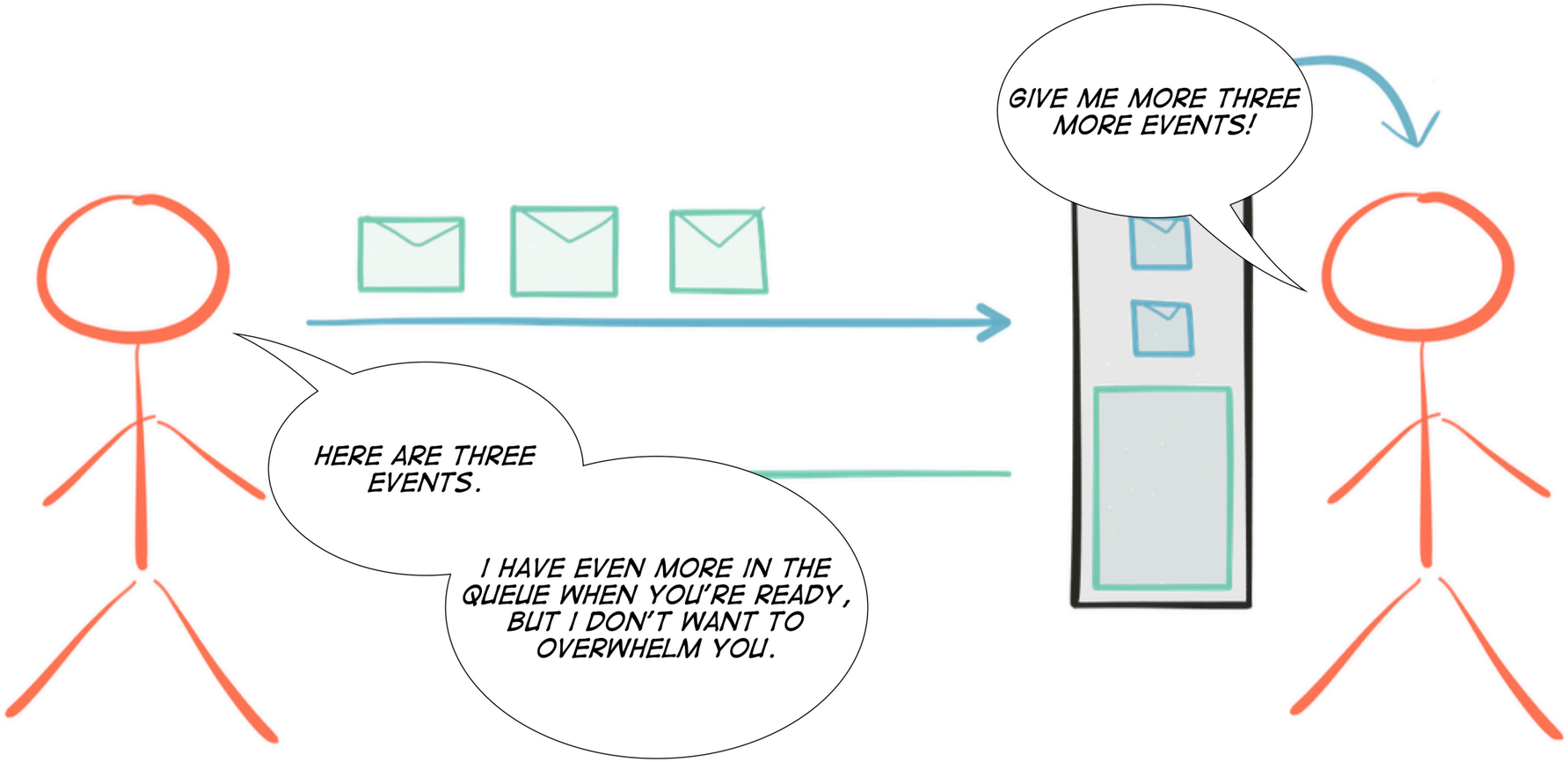
DEMAND FLOWS UPSTREAM, EVENTS FLOW DOWNSTREAM.

IDEAL SOLUTION: **PULL-BASED BACKPRESSURE.**



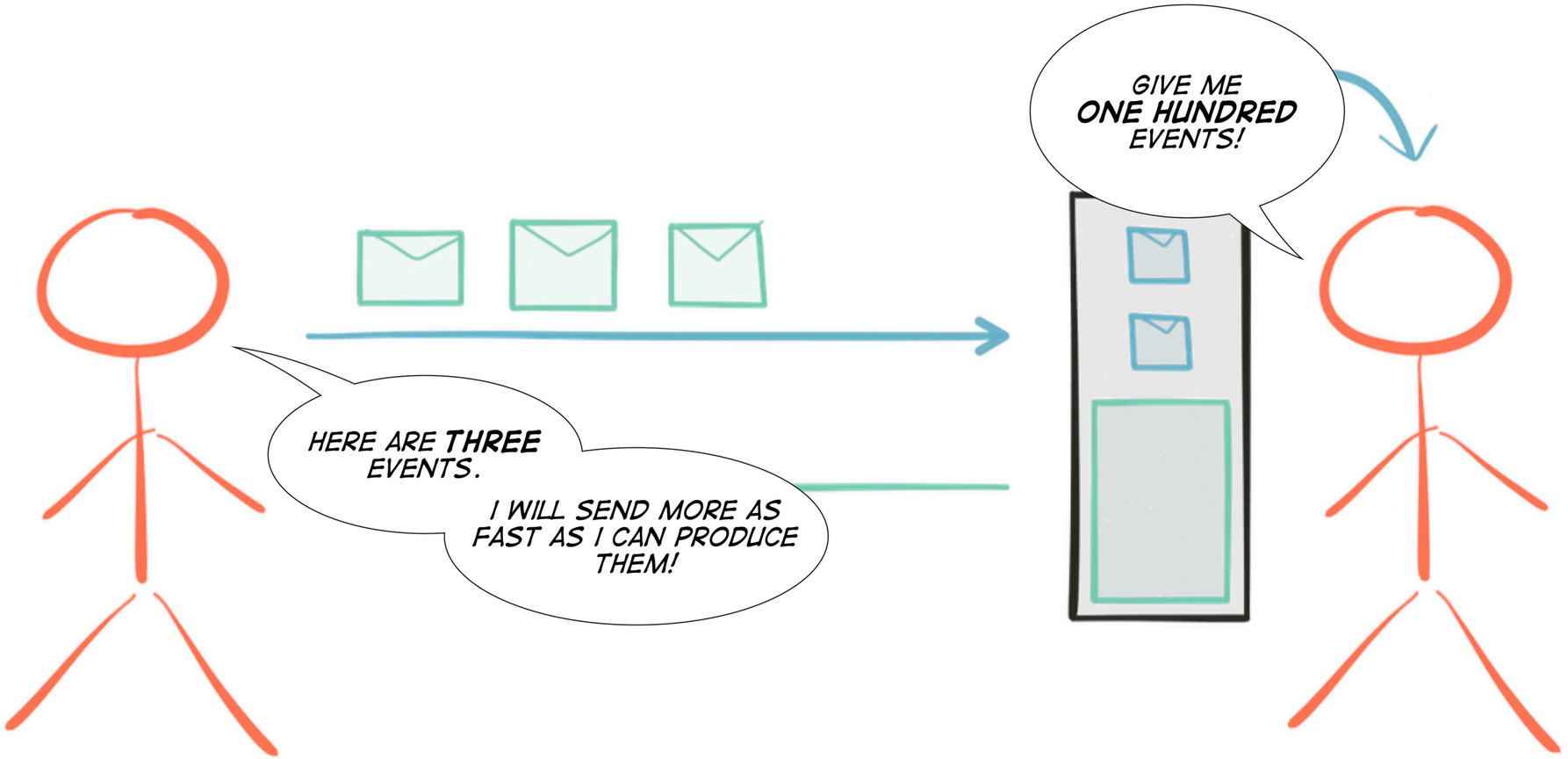
DEMAND FLOWS UPSTREAM, EVENTS FLOW DOWNSTREAM.

IDEAL SOLUTION: **PULL-BASED BACKPRESSURE.**



DEMAND FLOWS UPSTREAM, EVENTS FLOW DOWNSTREAM.

PULL-BASED BACKPRESSURE **DYNAMICALLY CHANGES** TO **PUSH** WHEN **MORE DEMAND** THAN SUPPLY.



INTEROP

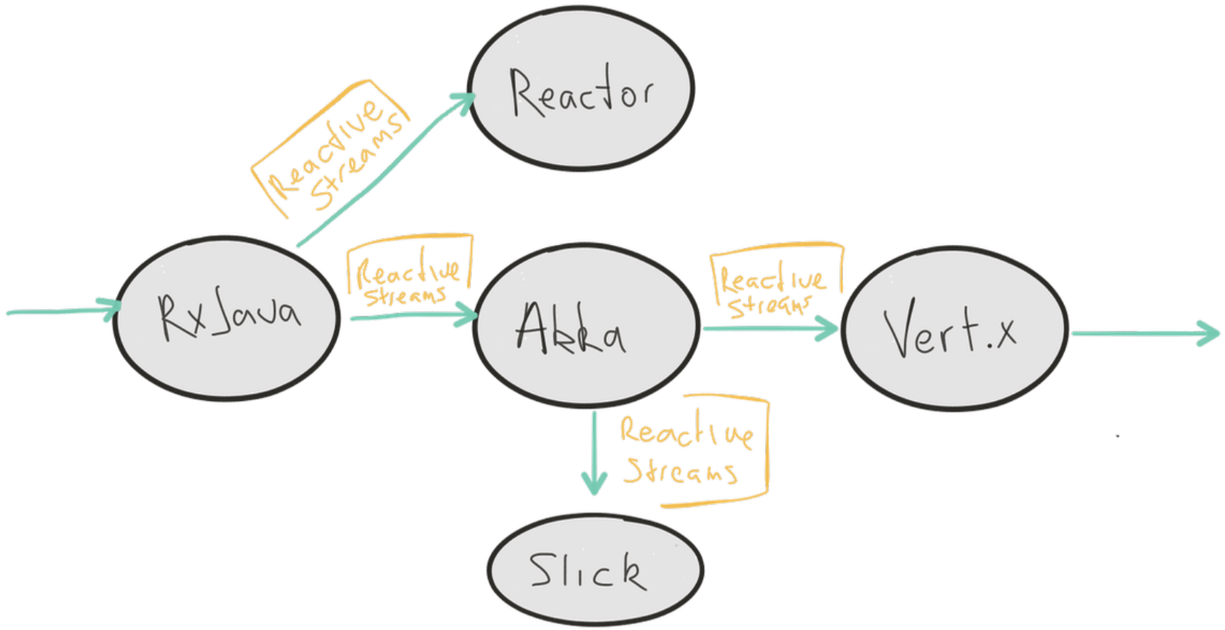
» RxJava (Netflix)

» Reactor (Pivotal)

» Vert.x (RedHat)

» Akka Streams (Lightbend)

» Slick (Lightbend)



REACTIVE STREAMS

Visit the Reactive Streams website for more information.

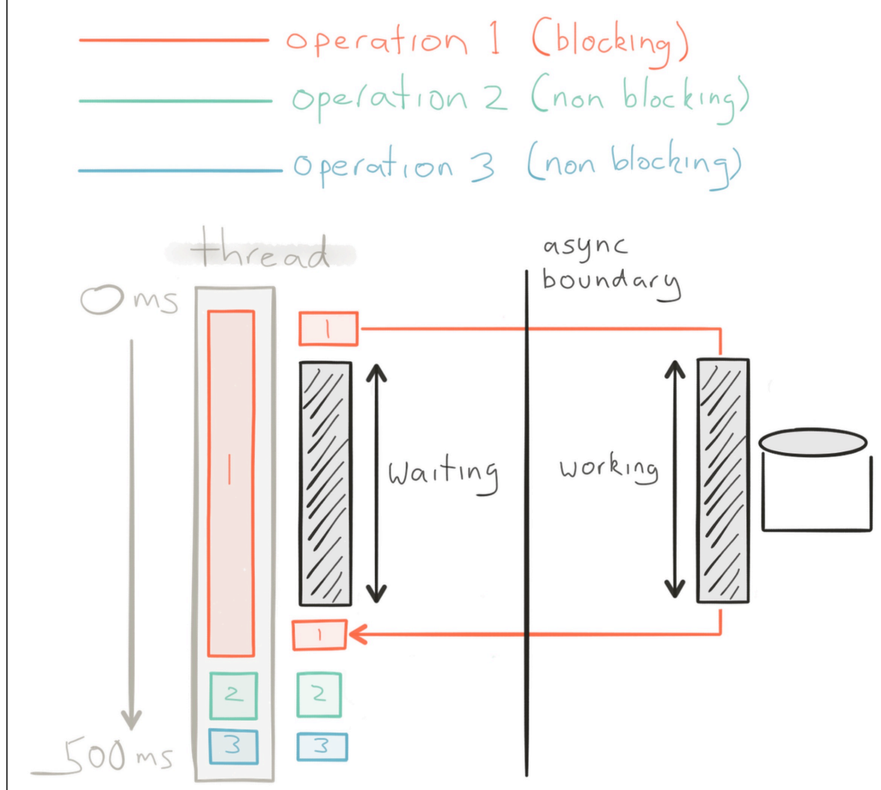
http://www.reactive-streams.org/

AKKA STREAMS

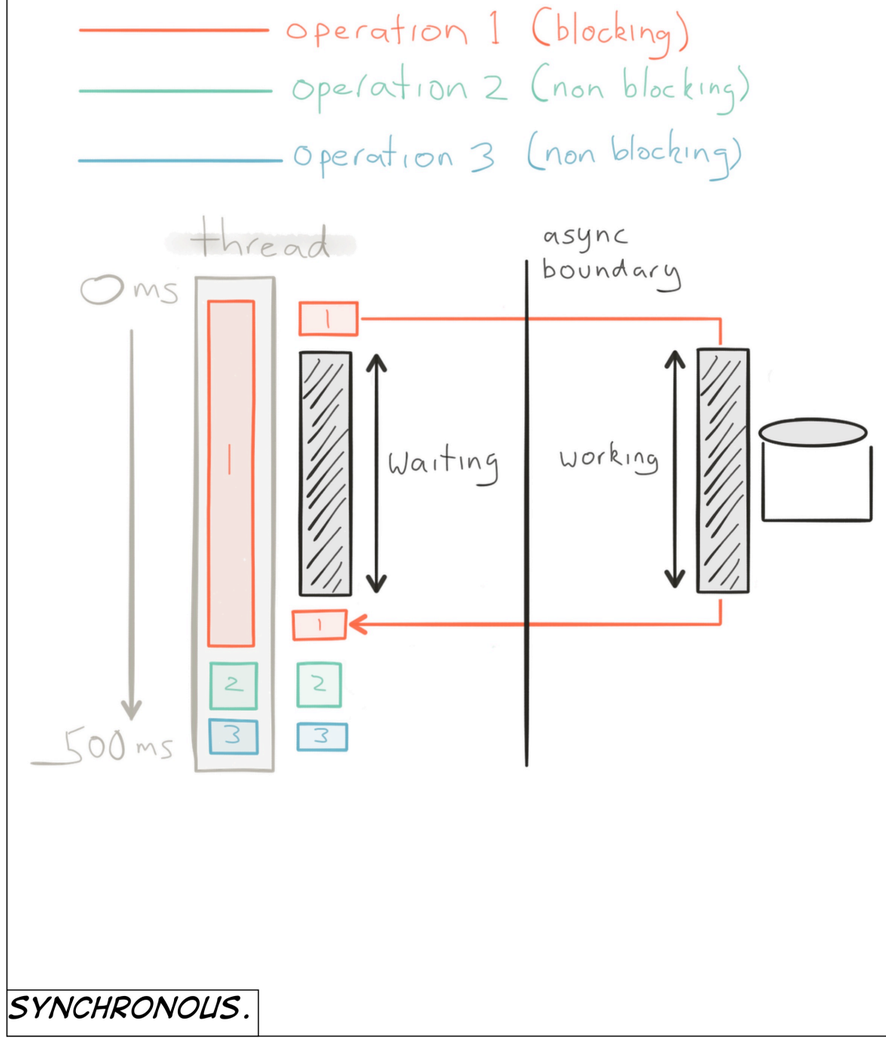
AKKA STREAMS

- A library to express and run a chain of asynchronous processing steps acting on a sequence of elements
  - DSL for async/non-blocking stream processing
  - Backpressure enabled by default
  - Implements the Reactive Streams spec for interoperability
  - Scala and Java APIs

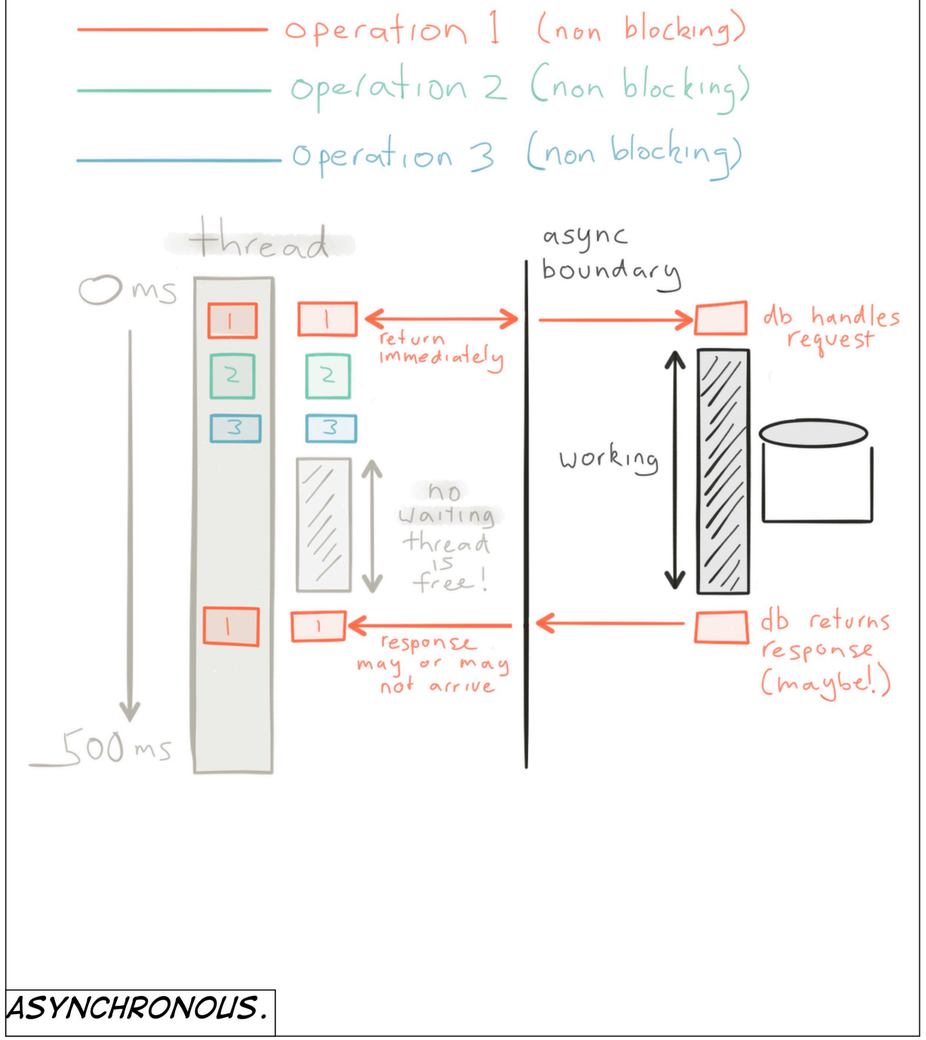
ASYNCHRONOUS BOUNDARY



SYNCHRONOUS.

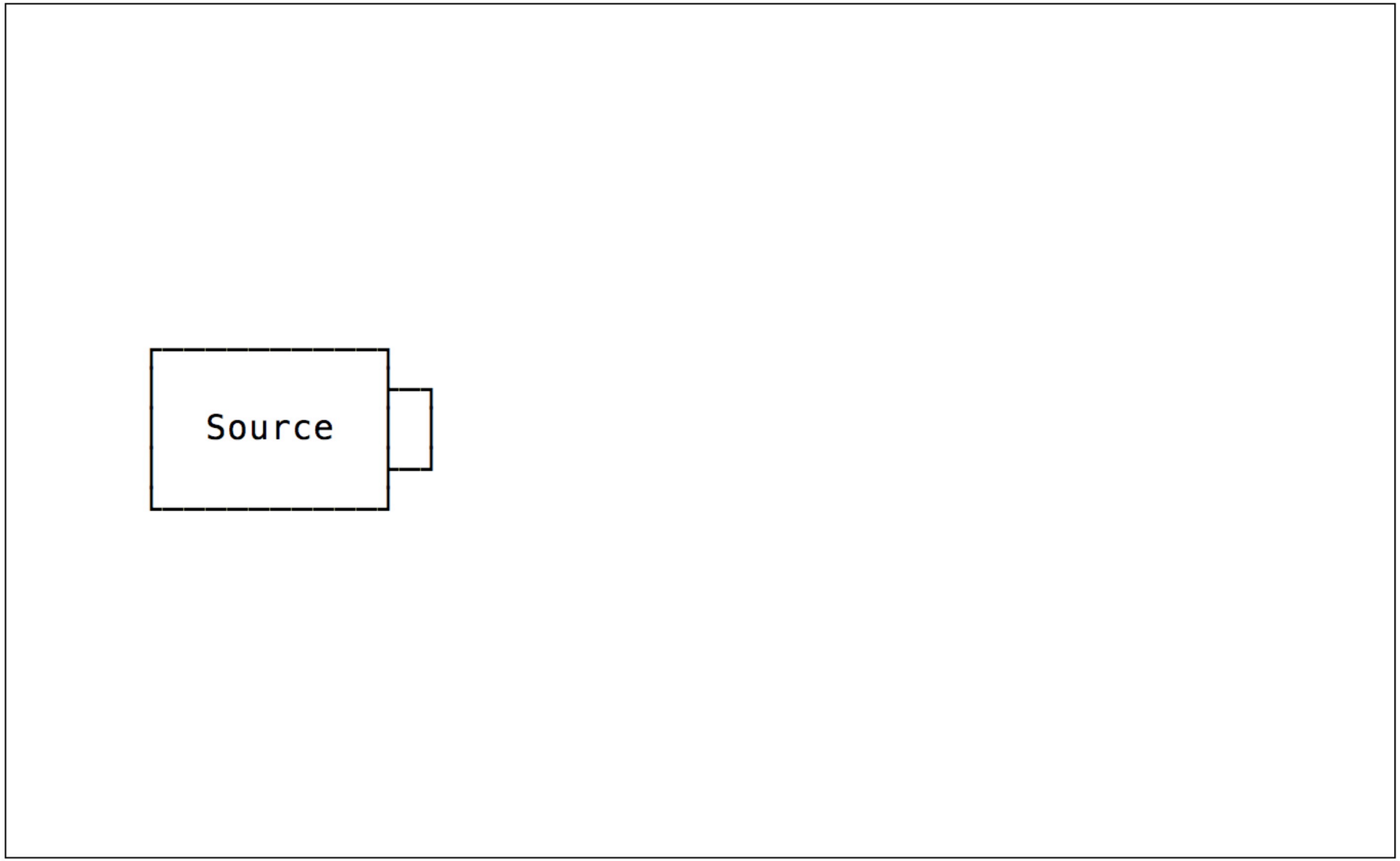


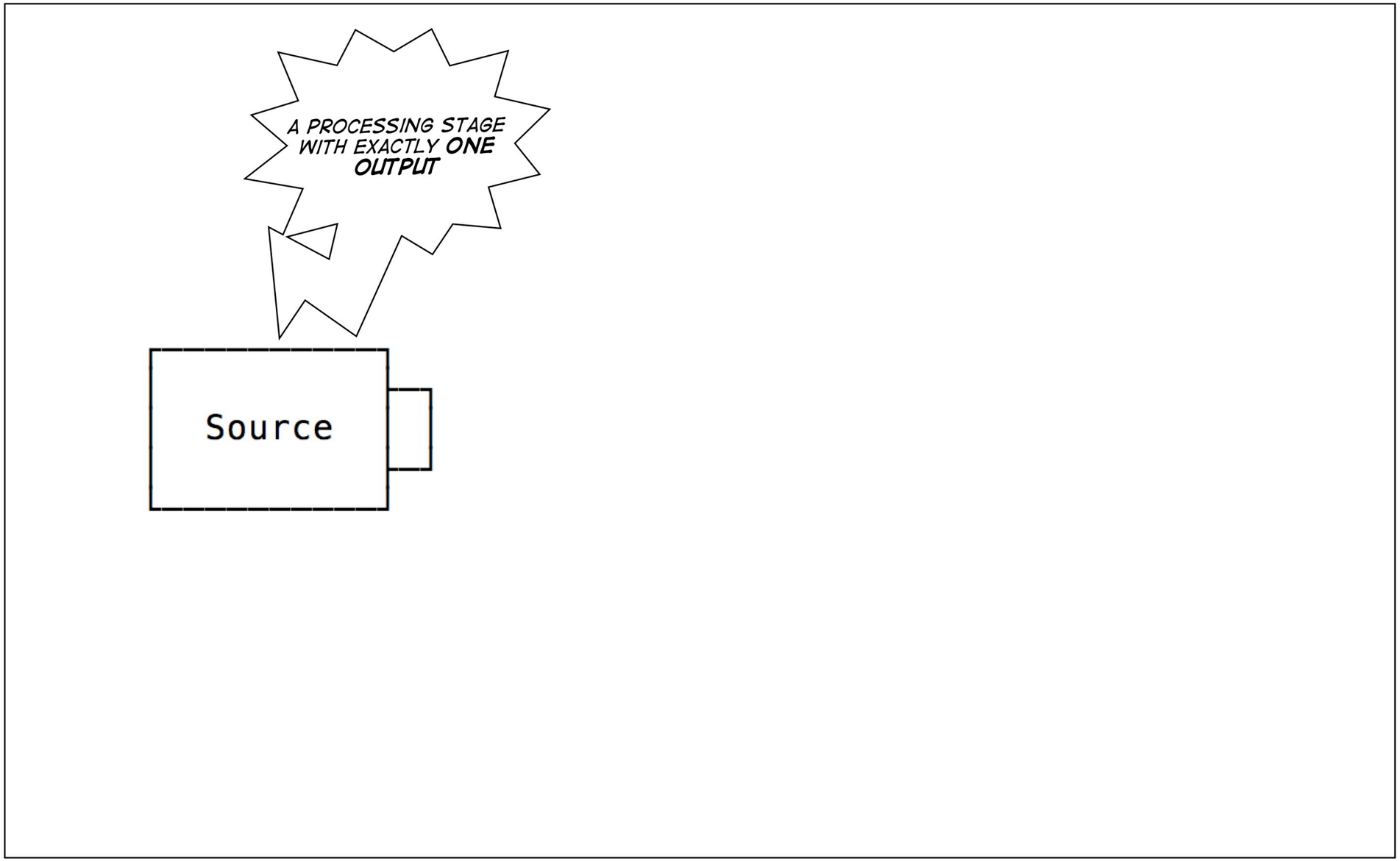
SYNCHRONOUS.

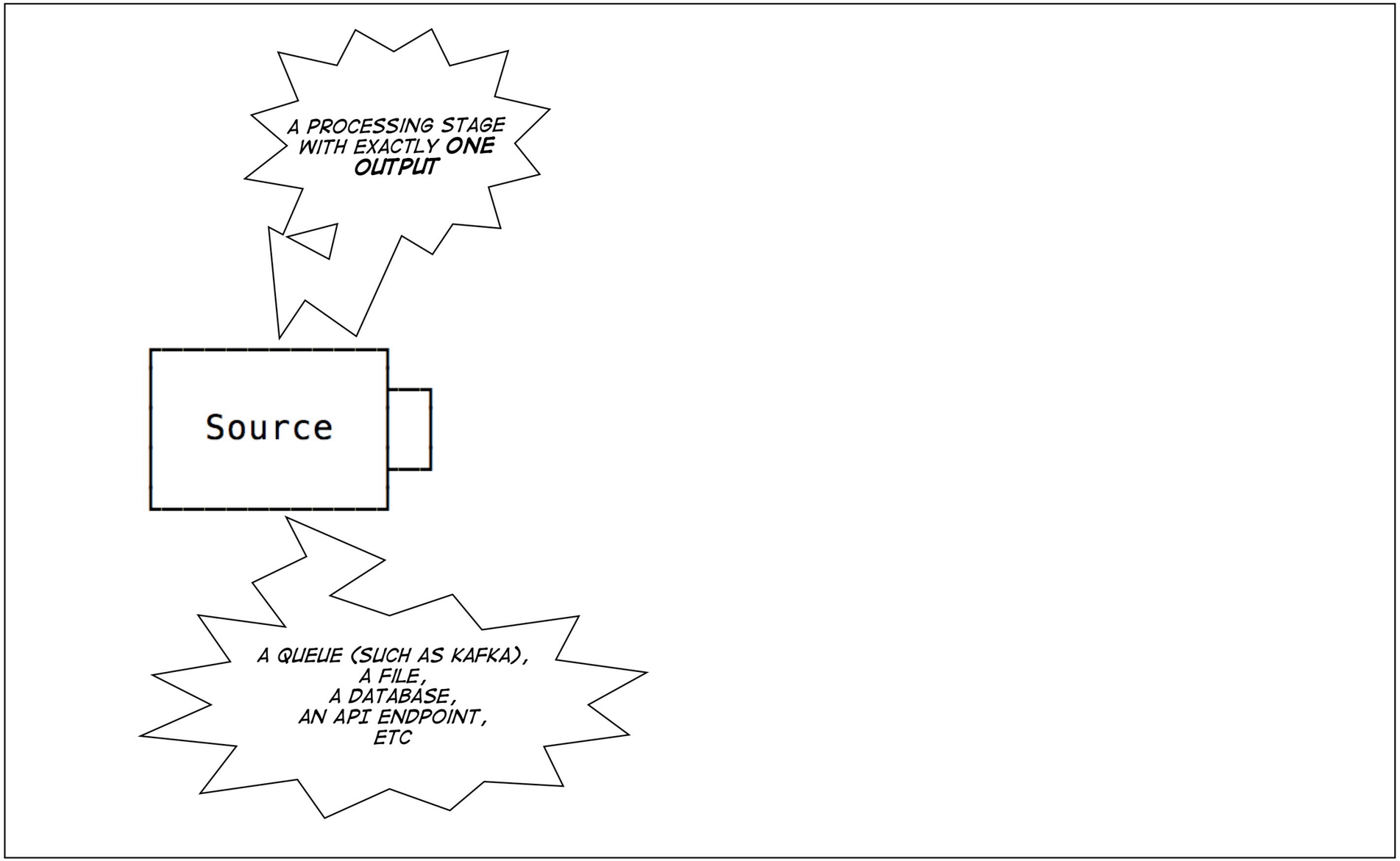


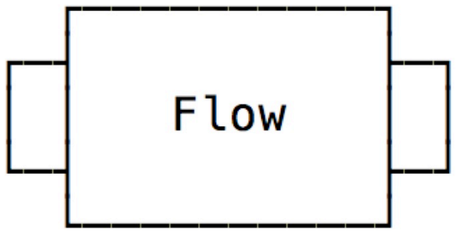
ASYNCHRONOUS.

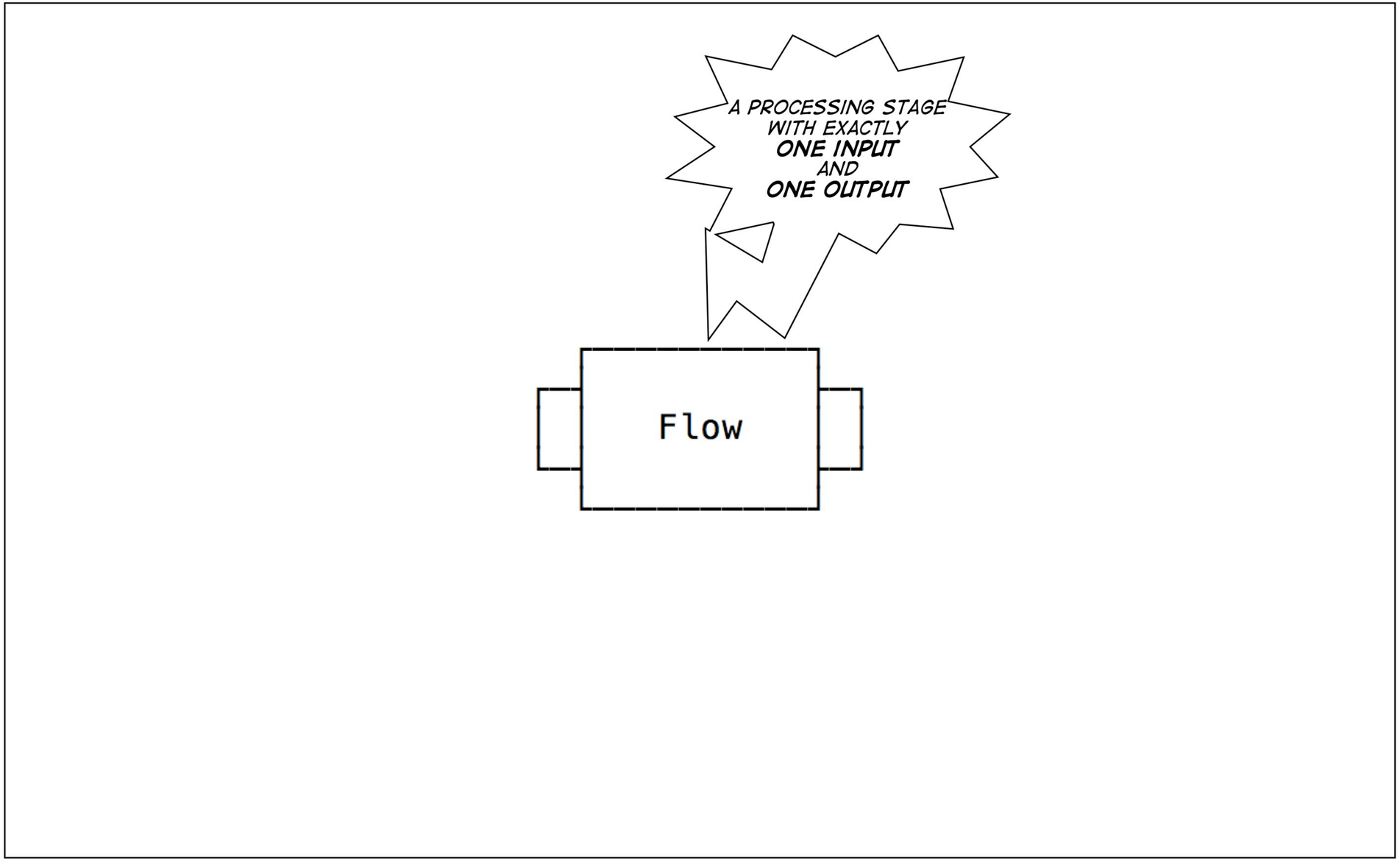
BASICS

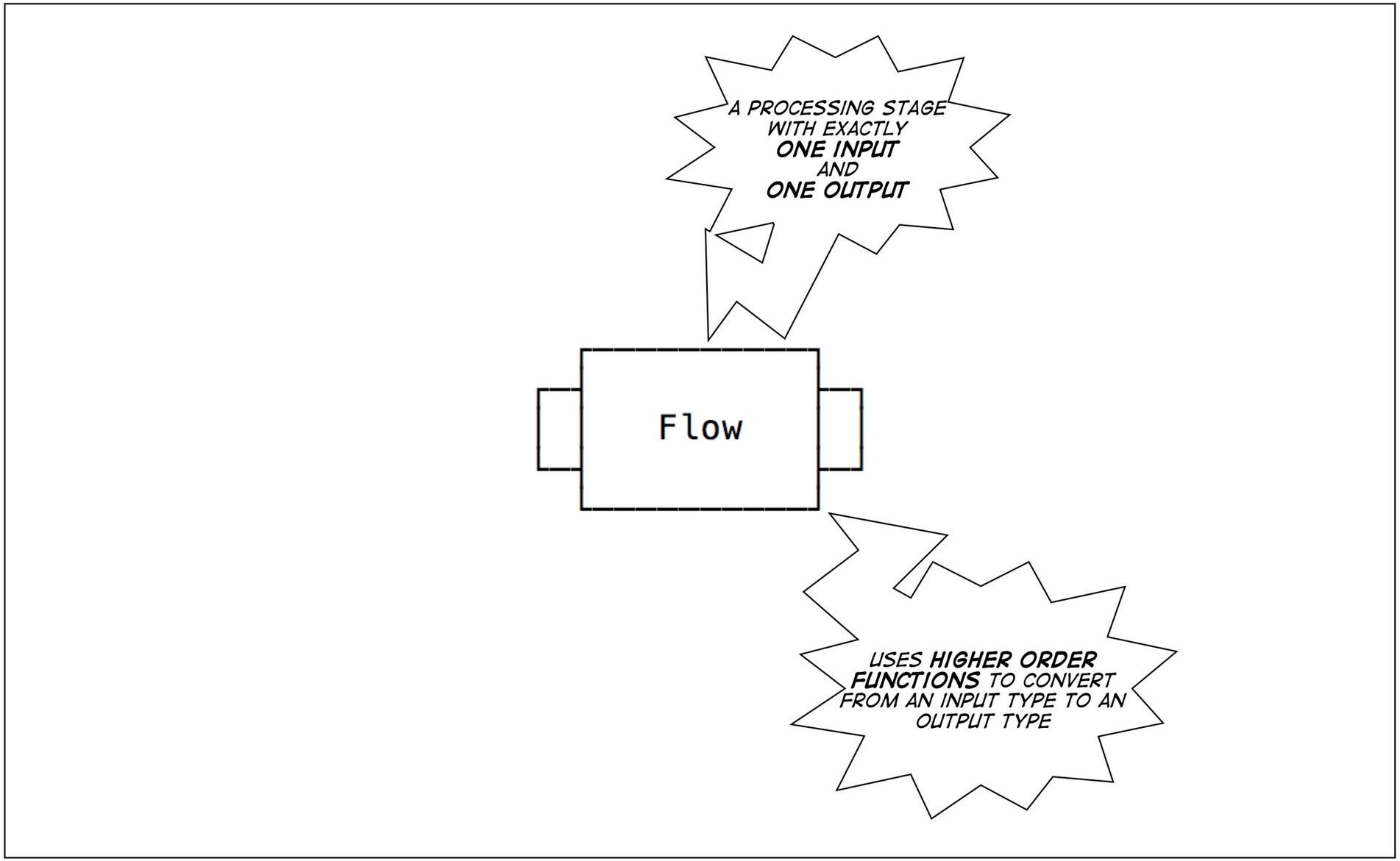

Source

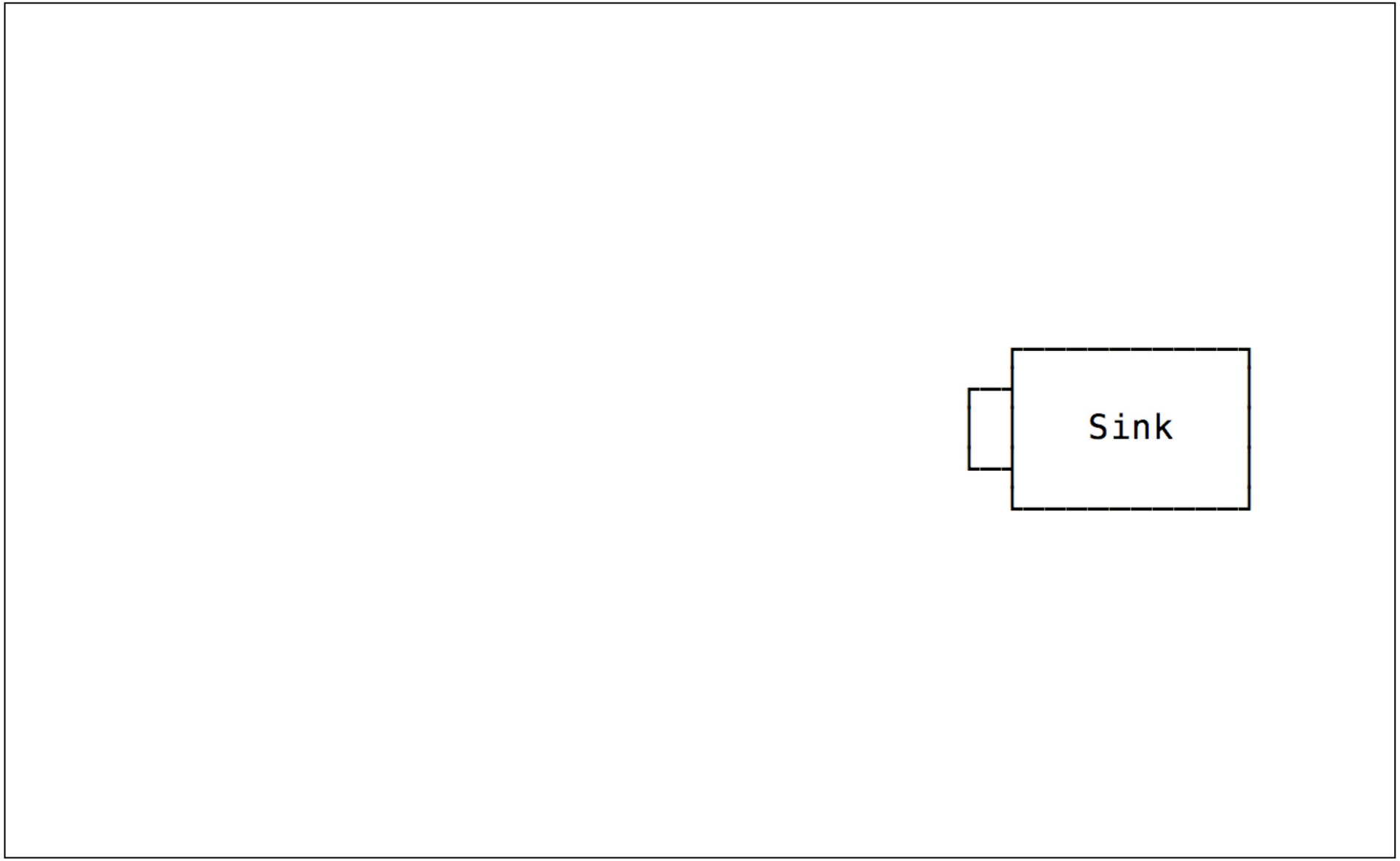

A PROCESSING STAGE WITH EXACTLY ONE OUTPUT
Source

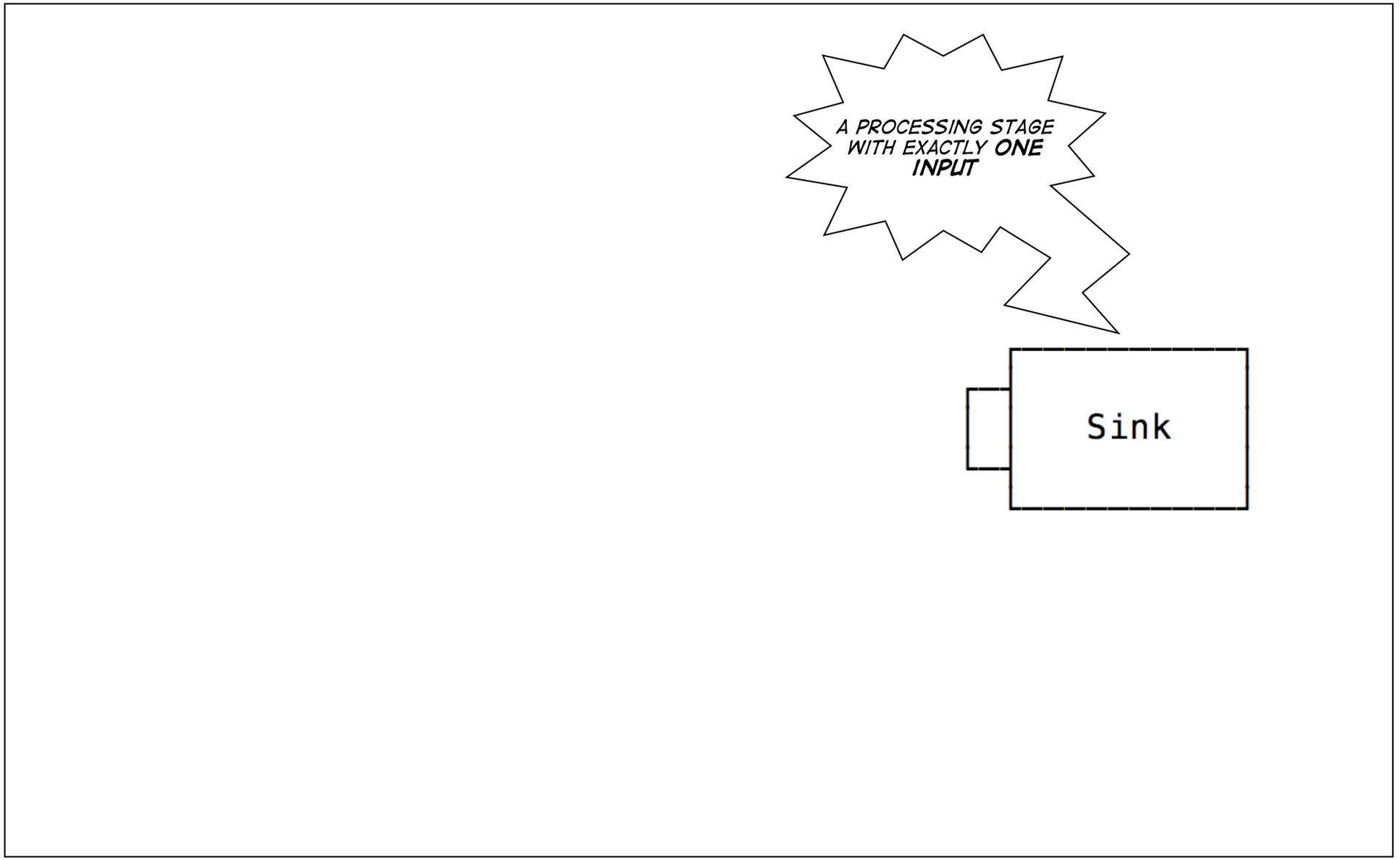

A PROCESSING STAGE WITH EXACTLY ONE OUTPUT
Source
A QUEUE (SUCH AS KAFKA),
A FILE,
A DATABASE,
AN API ENDPOINT,
ETC

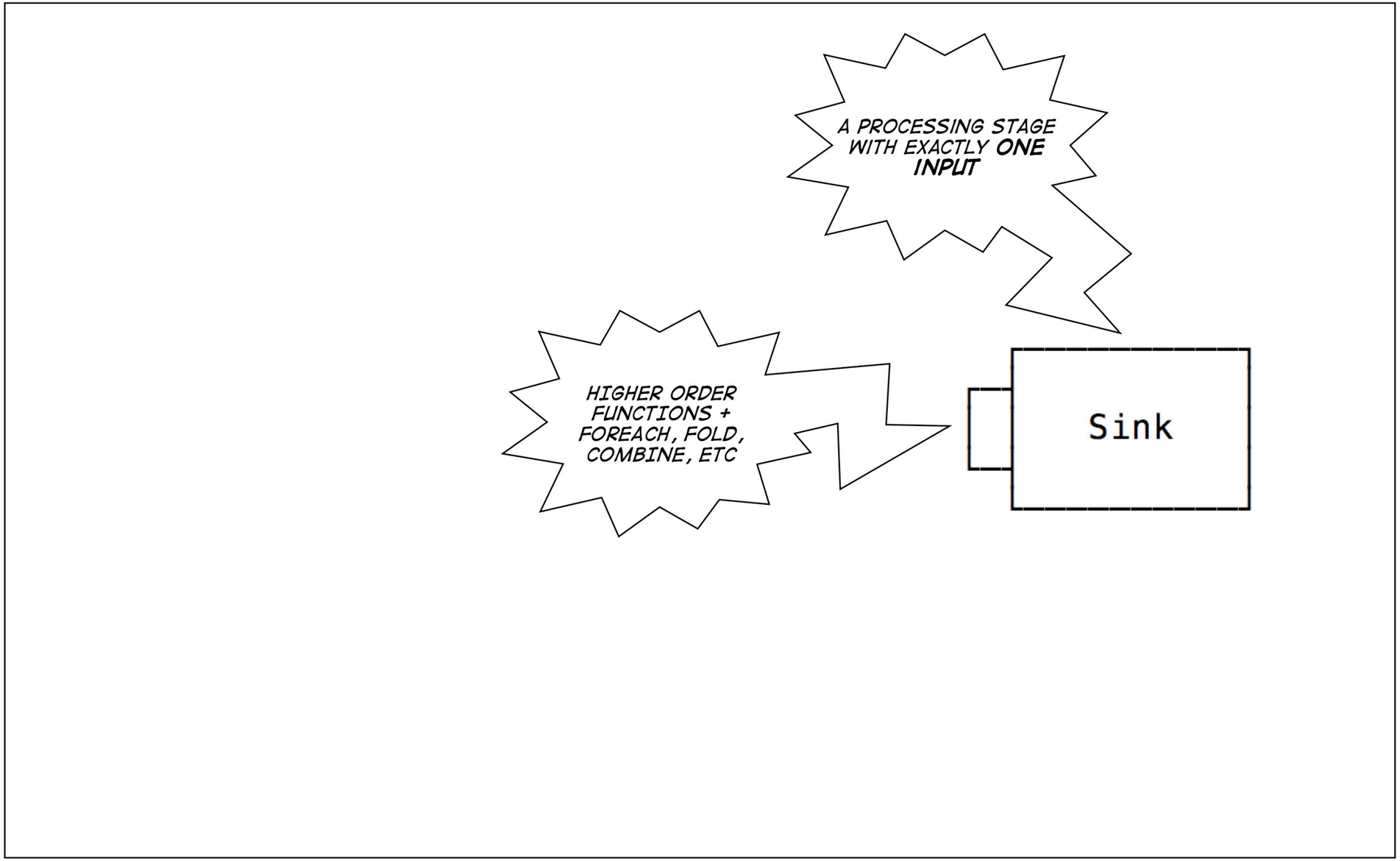

Flow

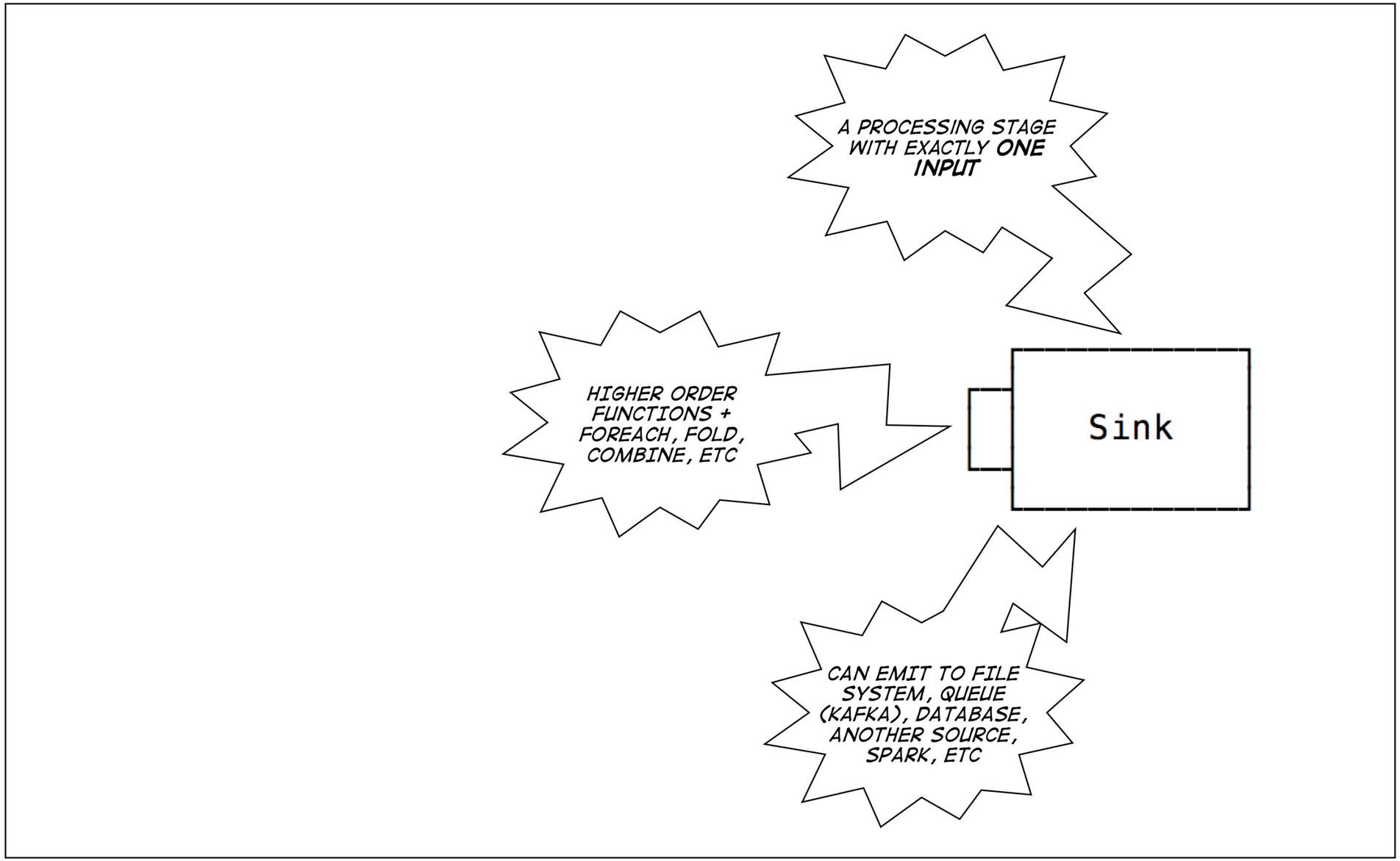

A PROCESSING STAGE
WITH EXACTLY
ONE INPUT
AND
ONE OUTPUT
Flow


A PROCESSING STAGE WITH EXACTLY ONE INPUT AND ONE OUTPUT
Flow
USES HIGHER ORDER FUNCTIONS TO CONVERT FROM AN INPUT TYPE TO AN OUTPUT TYPE

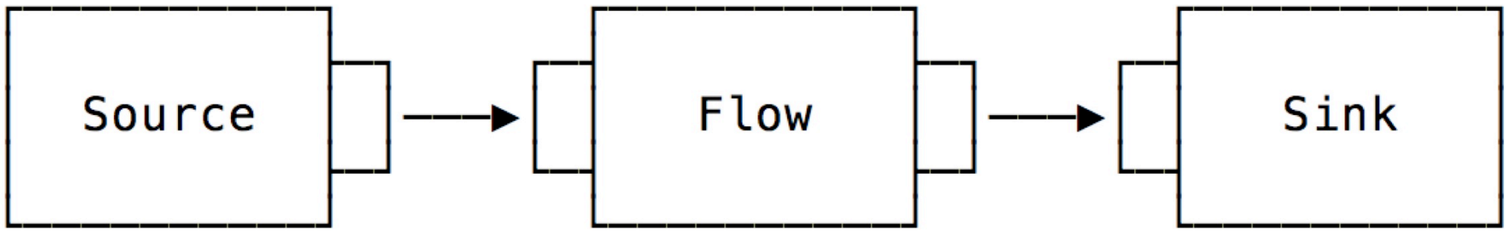

Sink


A PROCESSING STAGE WITH EXACTLY ONE INPUT
Sink


A PROCESSING STAGE WITH EXACTLY ONE INPUT
HIGHER ORDER FUNCTIONS + FOREACH, FOLD, COMBINE, ETC
Sink

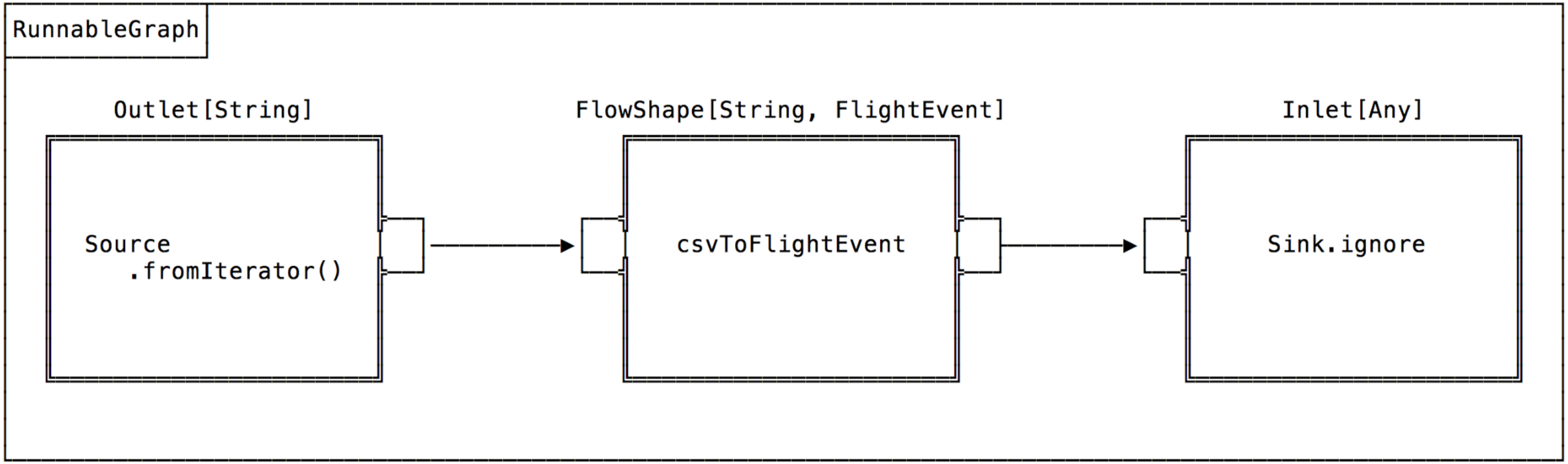

A PROCESSING STAGE WITH EXACTLY ONE INPUT
HIGHER ORDER FUNCTIONS + FOREACH, FOLD, COMBINE, ETC
Sink
CAN EMIT TO FILE SYSTEM, QUEUE (KAFKA), DATABASE, ANOTHER SOURCE, SPARK, ETC

RUNNABLEGRAPH – ALL STAGES ARE CONNECTED TO A SOURCE AND A SINK.



LET'S ANALYZE FLIGHT DATA

1. Read in all flight data from 2008
2. Transform CSV rows to data structures
3. Compute the average delay per airline
4. Emit the results to console


RunnableGraph
Outlet[String]
FlowShape[String, FlightEvent]
Inlet[Any]
Source
.fromIterator()
csvToFlightEvent
Sink.ignore

```scala
val g: RunnableGraph[_] = RunnableGraph.fromGraph(GraphDSL.create() {
    implicit builder =>

      // Source
      val A: Outlet[String] = builder.add(Source.fromIterator(() => flightDelayLines)).out
      val B: FlowShape[String, FlightEvent] = builder.add(csvToFlightEvent)
      val C: Inlet[Any] = builder.add(Sink.ignore).in

      import GraphDSL.Implicits._ // allows us to build our graph using ~> combinators

      // Graph
      A ~> B ~> C

      ClosedShape // defines this as a "closed" graph, not exposing any inlets or outlets
  })

  g.run() // materializes and executes the blueprint
```

API DESIGN

Considerations

» Immutable, composable stream blueprints

» Explicit materialization step

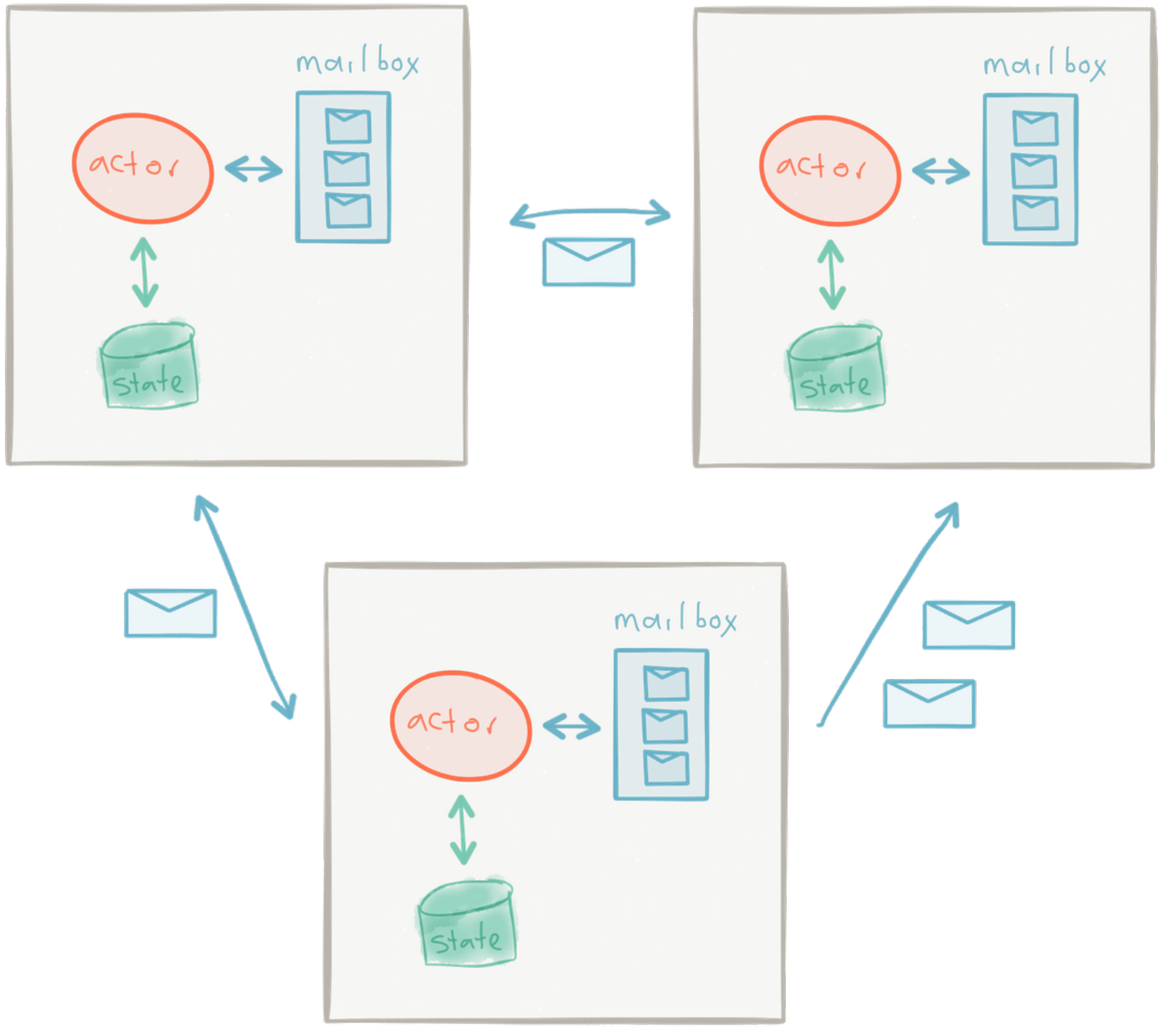
» No magic at the expense of some extra code

MATERIALIZATION

Separates the what from the how.

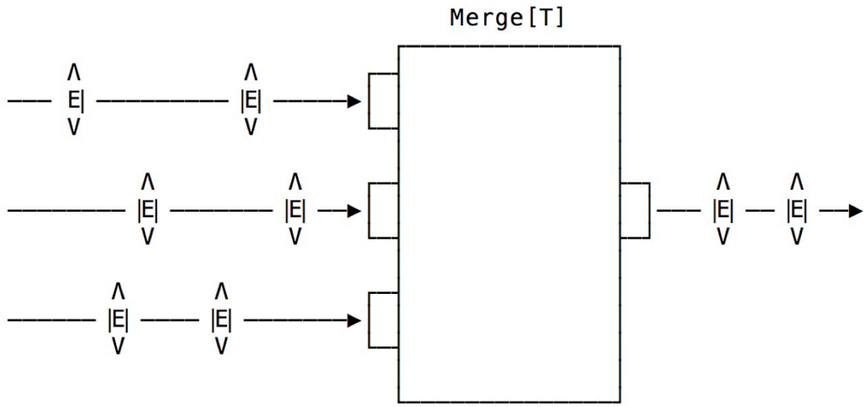
» Use Source/Flow/Sink to create a blueprint

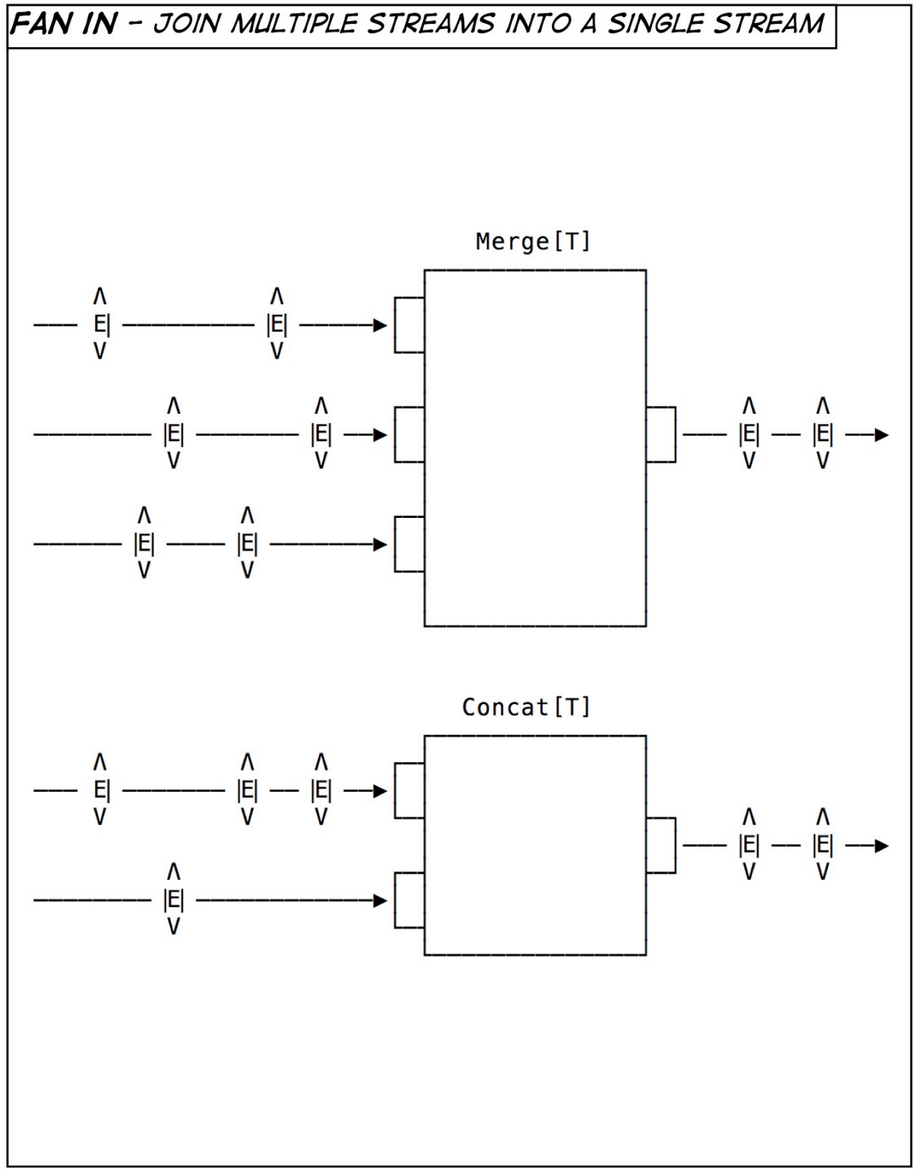
» FlowMaterializer turns a blueprint into Akka Actors



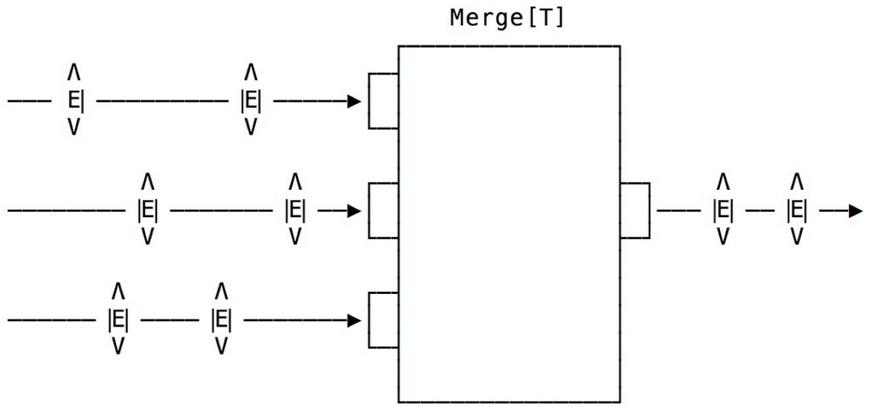
GRAPHS

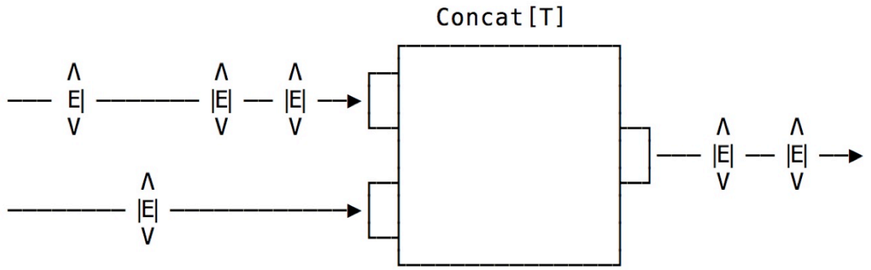
FAN IN – JOIN MULTIPLE STREAMS INTO A SINGLE STREAM



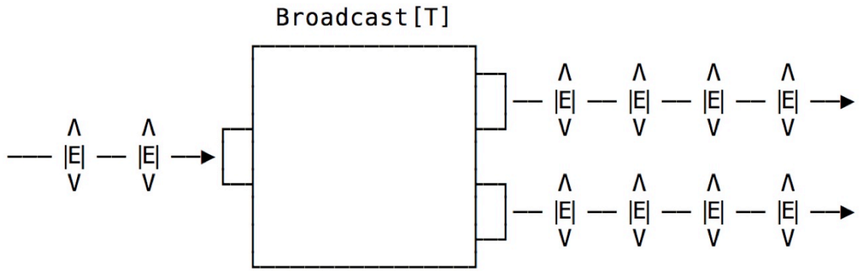

FAN IN - JOIN MULTIPLE STREAMS INTO A SINGLE STREAM
Merge[T]
Concat[T]

FAN IN - JOIN MULTIPLE STREAMS INTO A SINGLE STREAM

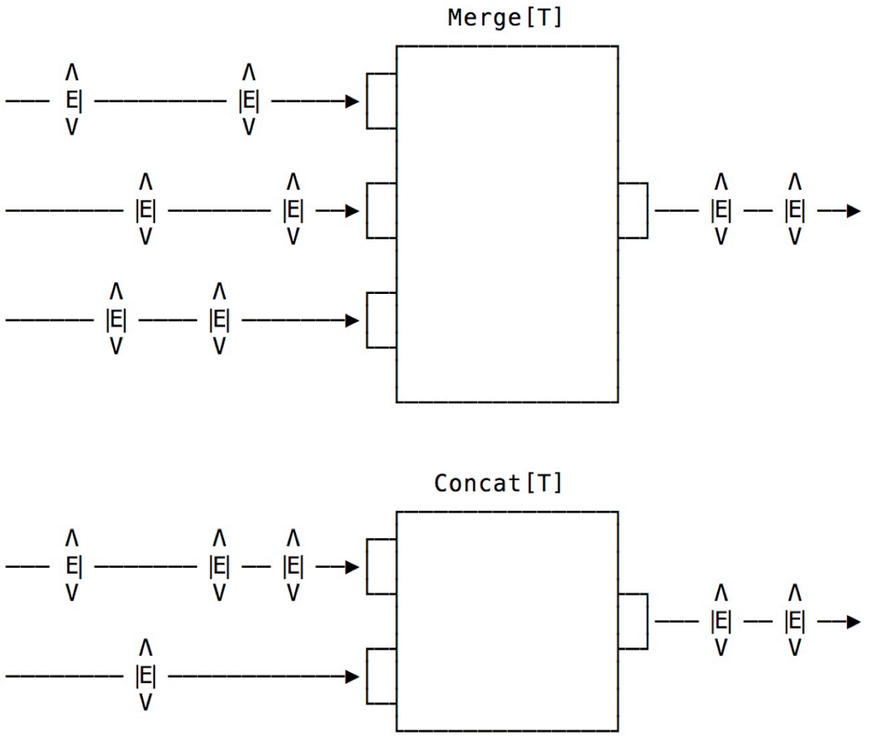




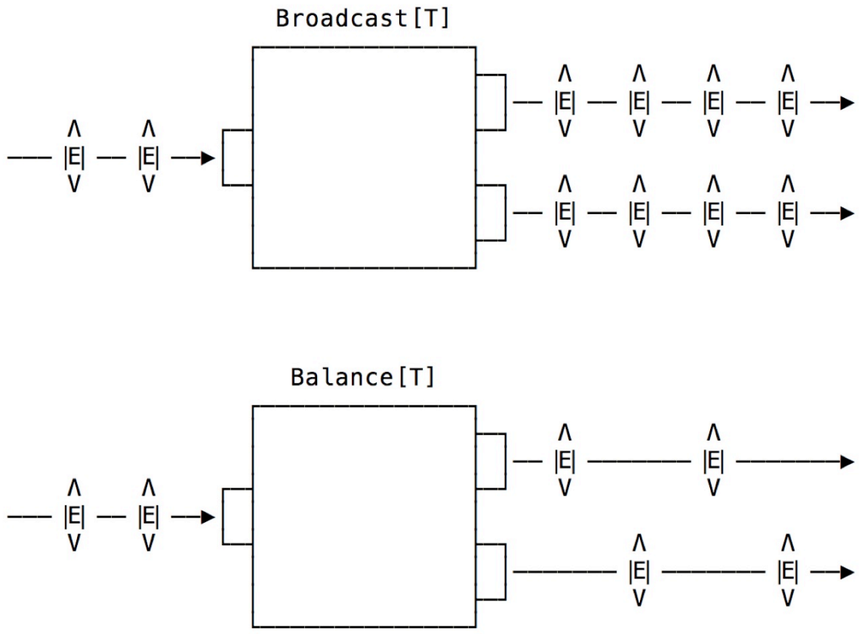
FAN OUT - SPLIT A STREAM INTO SUBSTREAMS



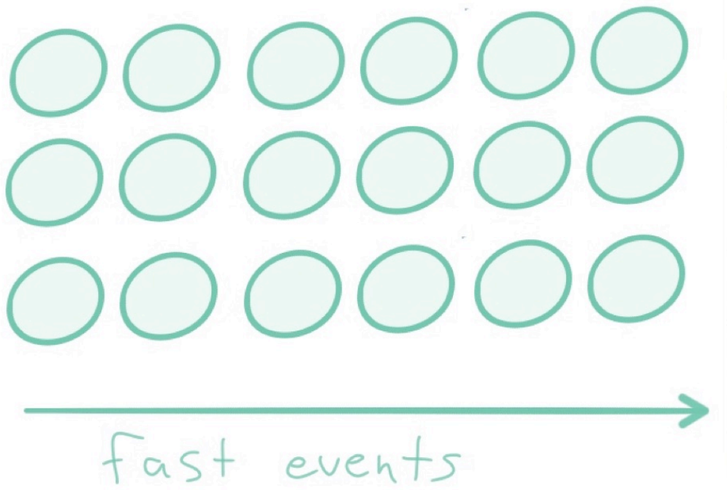
FAN IN – JOIN MULTIPLE STREAMS INTO A SINGLE STREAM



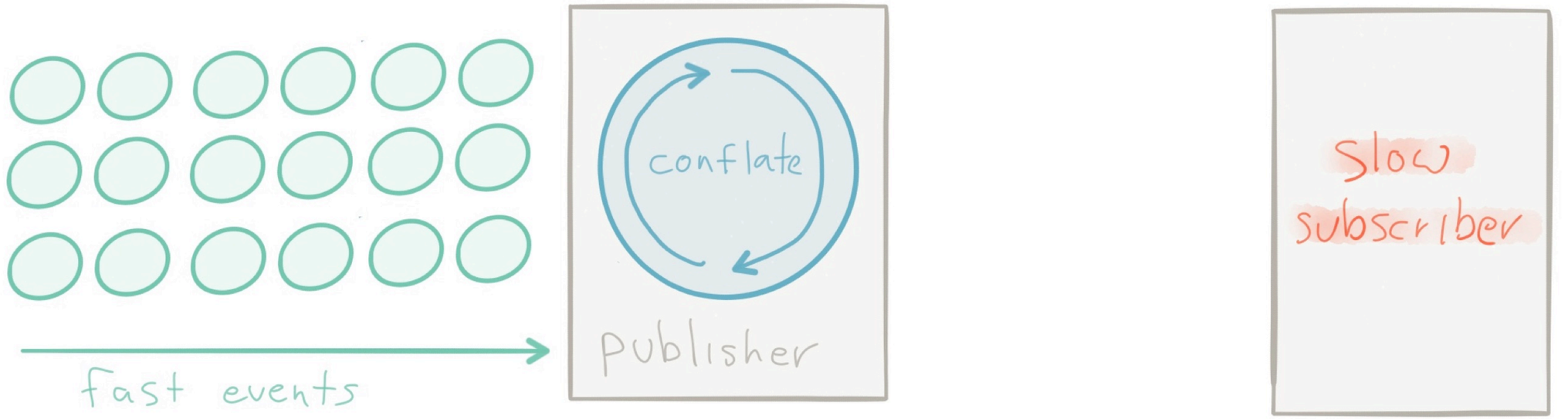
FAN OUT – SPLIT A STREAM INTO SUBSTREAMS

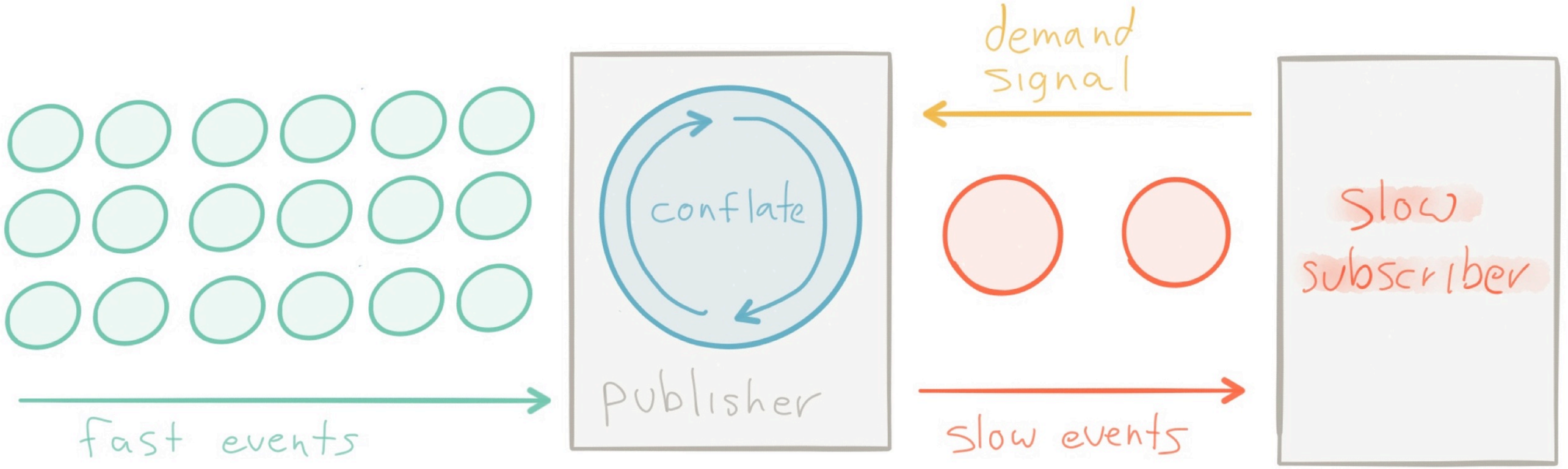


ADVANCED FLOW CONTROL

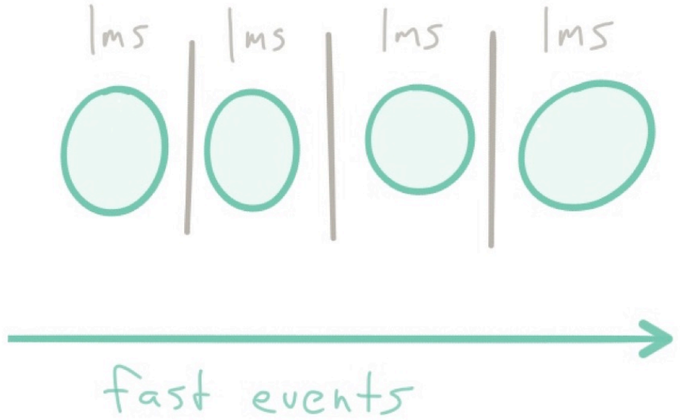

fast events

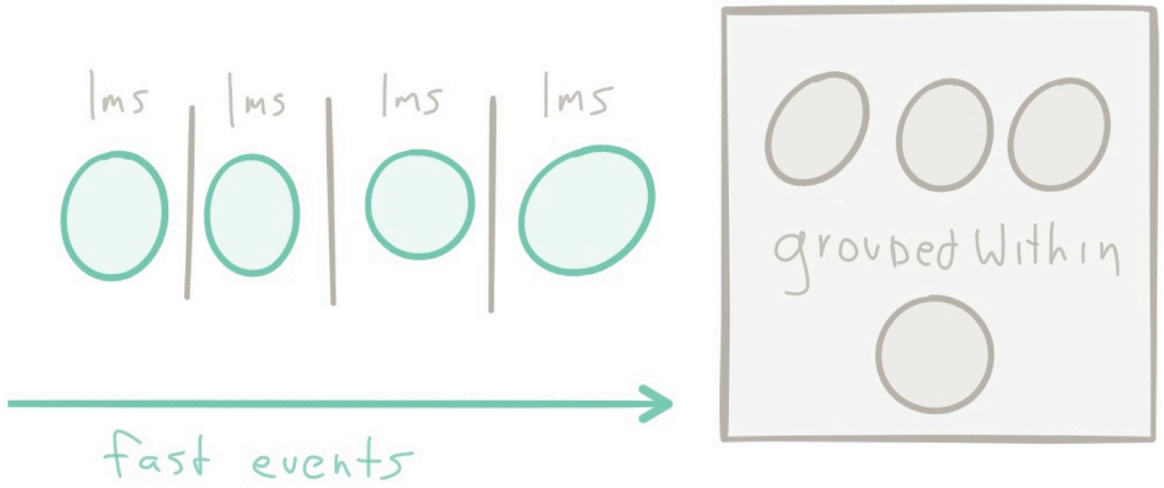

slow
subscriber

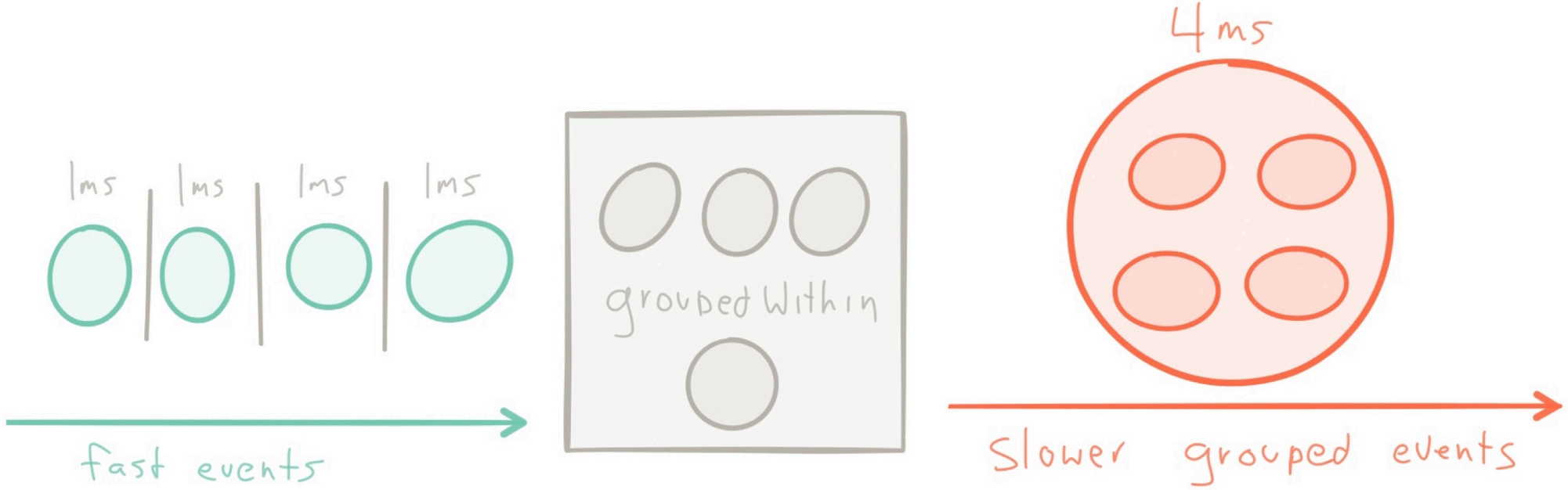

conflate
publisher
fast events
slow subscriber



CONFLATE - EMIT AN AGGREGATE EVENT BASED ON ANY CRITERIA (AVERAGE, HIGH, LOW, ETC)

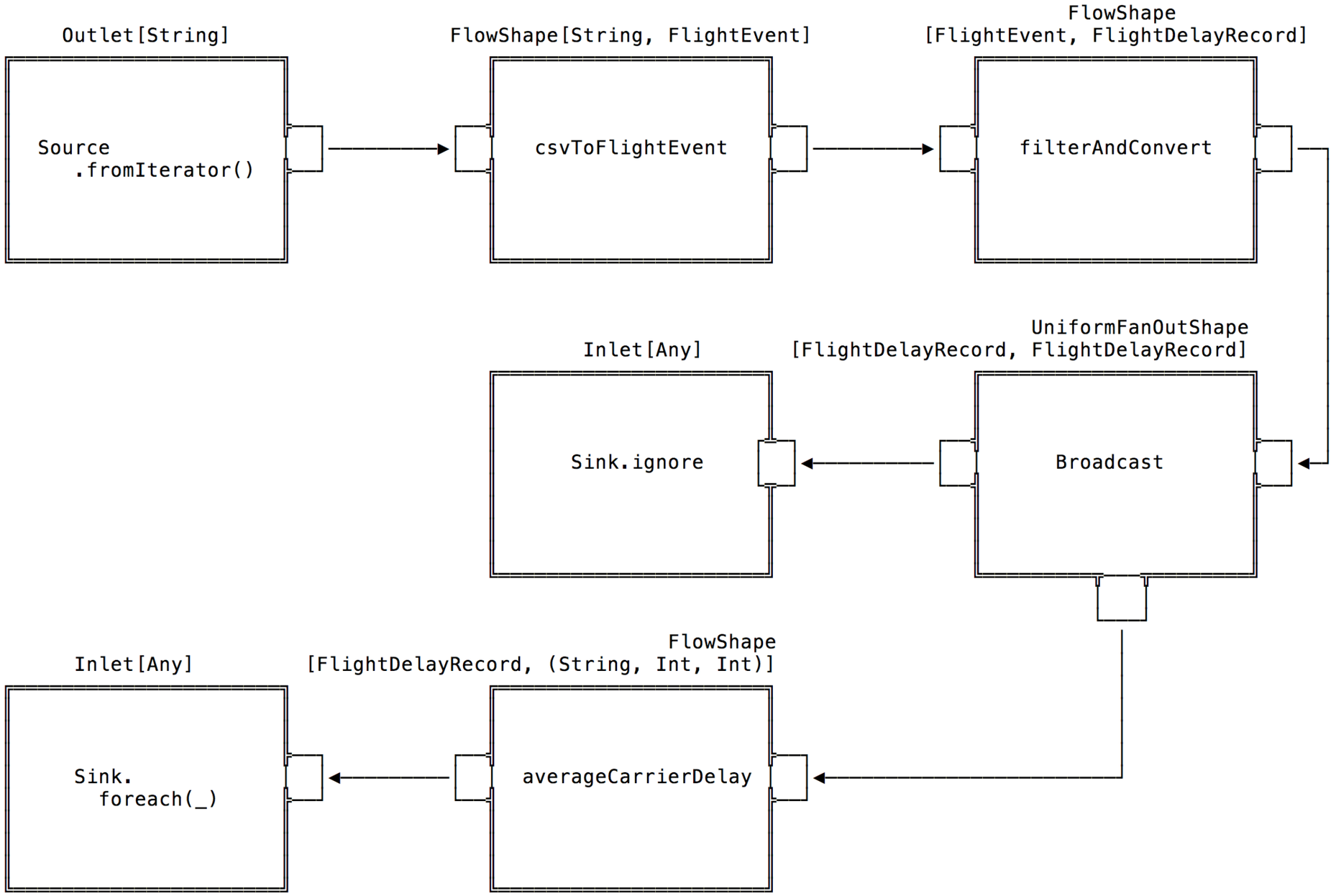

1ms
1ms
1ms
1ms
fast events


1ms
1ms
1ms
1ms
fast events
grouped within

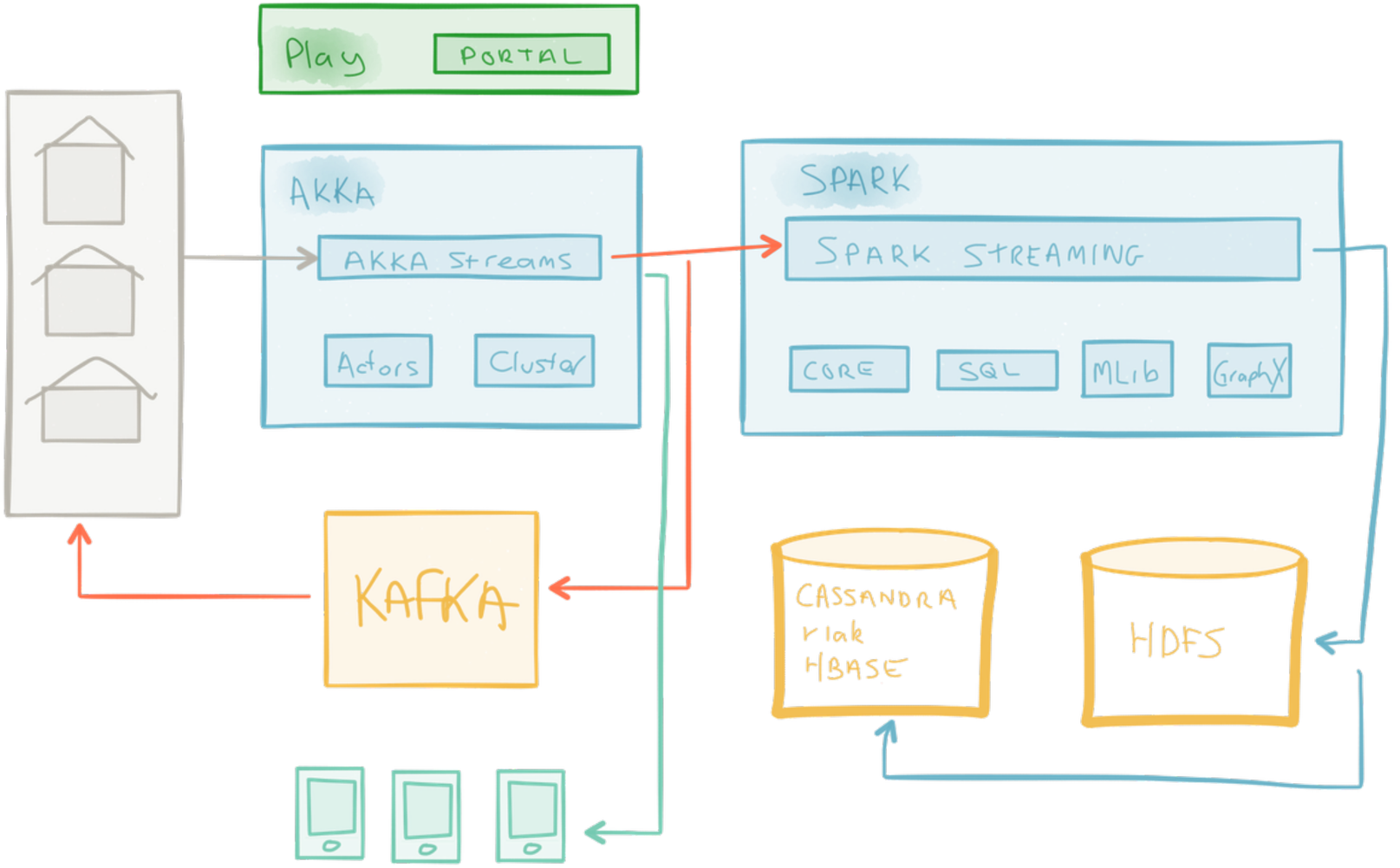


GROUPEDWITHIN - EMIT MINI-BATCH OF EVENTS BY TIME WINDOW OR # OF EVENTS

CODE AND REVIEW


Outlet[String]
Source
.fromIterator()
FlowShape[String, FlightEvent]
csvToFlightEvent
FlowShape
[FlightEvent, FlightDelayRecord]
filterAndConvert
Inlet[Any]
Sink.ignore
UniformFanOutShape
[FlightDelayRecord, FlightDelayRecord]
Broadcast
Inlet[Any]
Sink.
foreach(_)
FlowShape
[FlightDelayRecord, (String, Int, Int)]
averageCarrierDelay

```scala
val g = RunnableGraph.fromGraph(GraphDSL.create() {
  implicit builder =>
    import GraphDSL.Implicits._

    // Source
    val A: Outlet[String] = builder.add(Source.fromIterator(() => flightDelayLines)).out

    // Flows
    val B: FlowShape[String, FlightEvent] = builder.add(csvToFlightEvent)
    val C: FlowShape[FlightEvent, FlightDelayRecord] = builder.add(filterAndConvert)
    val D: UniformFanOutShape[FlightDelayRecord, FlightDelayRecord] = builder.add(Broadcast[FlightDelayRecord](2))
    val F: FlowShape[FlightDelayRecord, (String, Int, Int)] = builder.add(averageCarrierDelay)

    // Sinks
    val E: Inlet[Any] = builder.add(Sink.ignore).in
    val G: Inlet[Any] = builder.add(Sink.foreach(averageSink)).in

    // Graph
    A ~> B ~> C ~> D
              E <~ D
         G <~ F <~ D

    ClosedShape
})

g.run()
```


Play
PORTAL
AKKA
AKKA Streams
Actors
Cluster
SPARK
SPARK STREAMING
CORE
SQL
MLib
GraphX
KAFKA
CASSANDRA
riak
HBASE
HDFS

THANK YOU!

Visit https://www.lightbend.com/products/lightbend-reactive-platform to get started

CONTACT INFO

» Lightbend: https://www.lightbend.com/contact

» Twitter: @kvnwbbr

» Email: kevin.webber@lightbend.com