# A dive into Akka Streams

From the basics to a real-world scenario

Gioia Ballin, Simone Fabiano - SW developers

measurence

# Who are these guys?

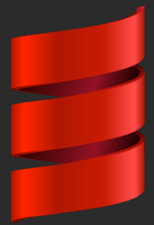
## Gioia Ballin

Twitter / @GioiaBallin

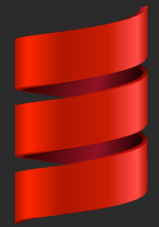
## Simone Fabiano

Twitter / @mone_tpatpc









R&D | Data Analysis | Blog | Microservices | Infrastructure | Web

measurence

# Agenda

The Problem

Akka Streams Basics

The Collector Service

Recap and Learnings

measurence

# Data processing pipeline

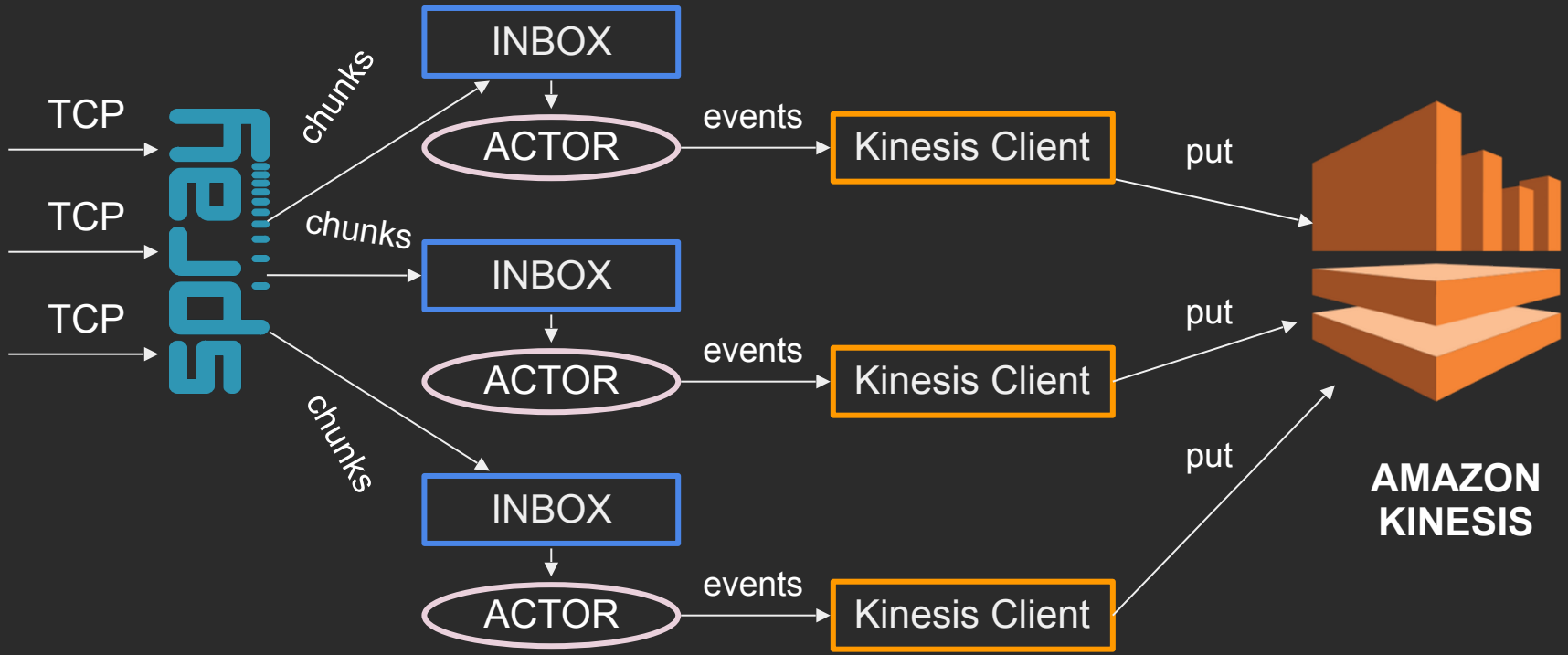
WI-FI/3G SENSORS

COLLECTOR SVC

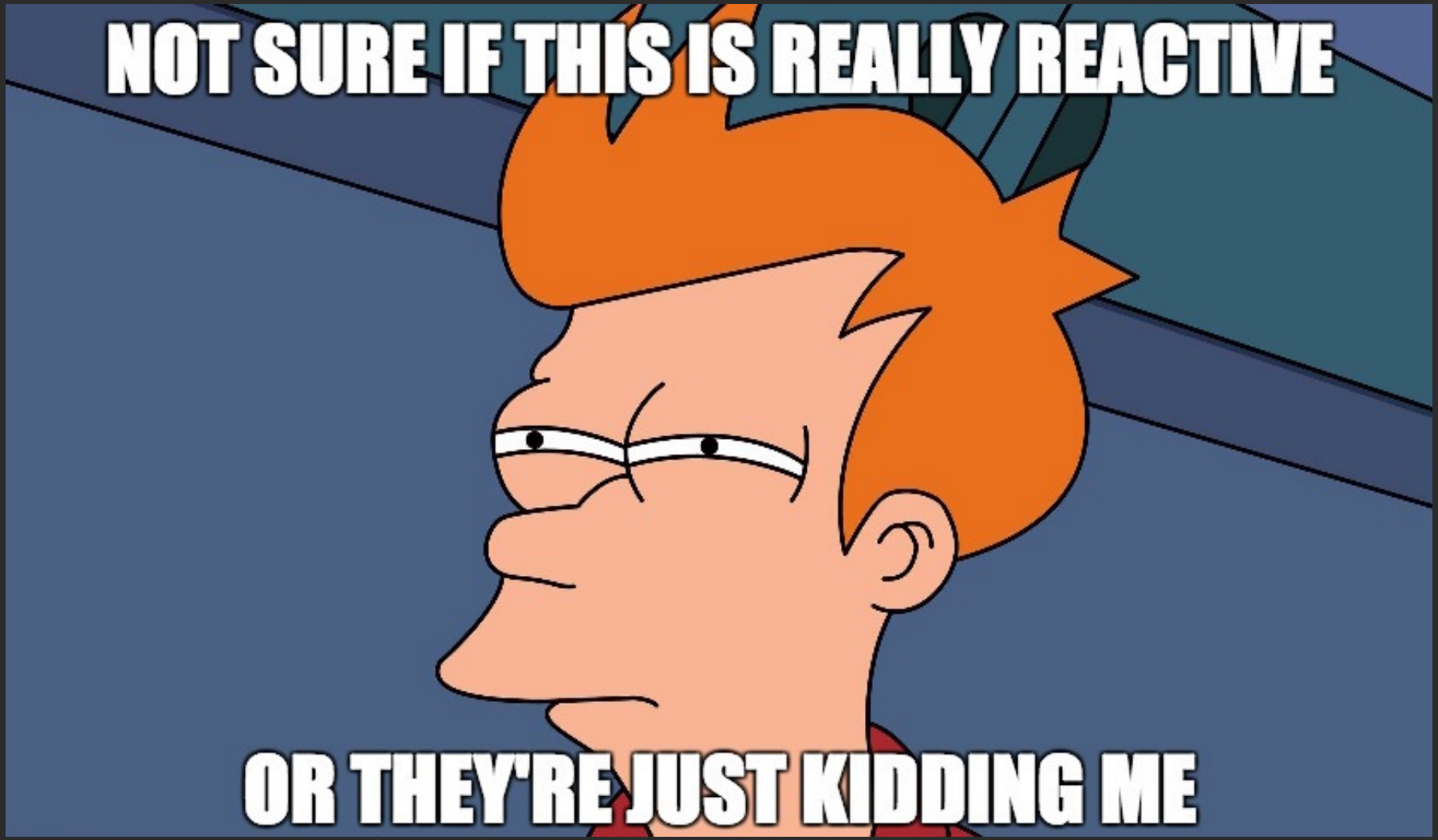
AMAZON KINESIS

measurence

# The Collector Service: two months ago...




measurence


NOT SURE IF THIS IS REALLY REACTIVE
OR THEY'RE JUST KIDDING ME

# Data may be lost

WI-FI/3G SENSORS

MESSAGE OVERLOAD

COLLECTOR SVC

CONNECTION ISSUES

AMAZON KINESIS

measurence

# Not REALLY reactive: why?

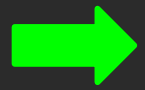
Lack of...

**RECOVERY MECHANISMS**

**BACKPRESSURE**

measurence

# Recovery mechanisms

Kinesis unavailable?  Buffer

Upload failure?  Retry

# Ingestion speed under control

Take advantage of sensors buffer

Sensors switch from busier to less busy services

measurence

# Welcome Akka Streams!

**Backpressure on the network**

**+**

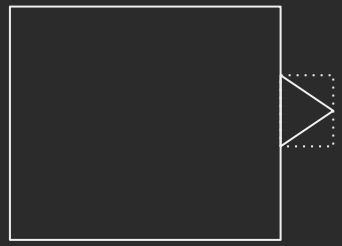
Easy fit for stream paradigm

Typed flow & Increased Readability

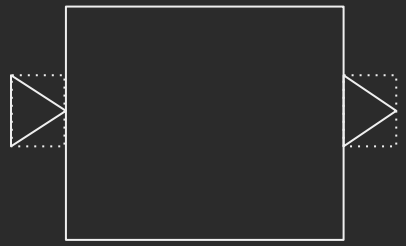
measurence

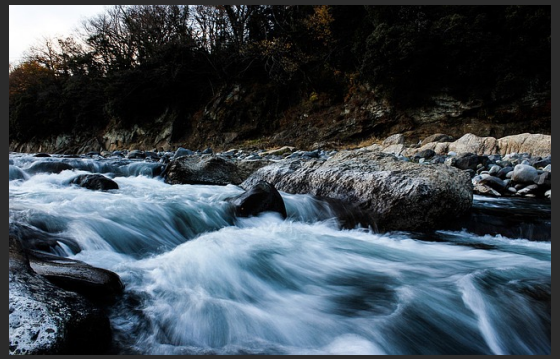
# Akka Streams Basics - 3 2 1 Go!

## SOURCE

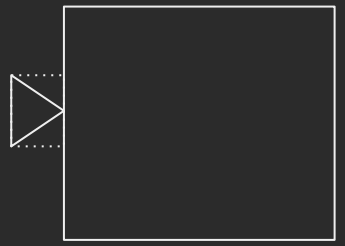


## FLOW





## SINK



measurence

```scala
val source = Source(1 to 42)
val flow = Flow[Int].map(_ + 1)
val sink = Sink.foreach(println)

val graph = source.via(flow).to(sink)

graph.run()
```

measurence

# Flows are collections (almost)

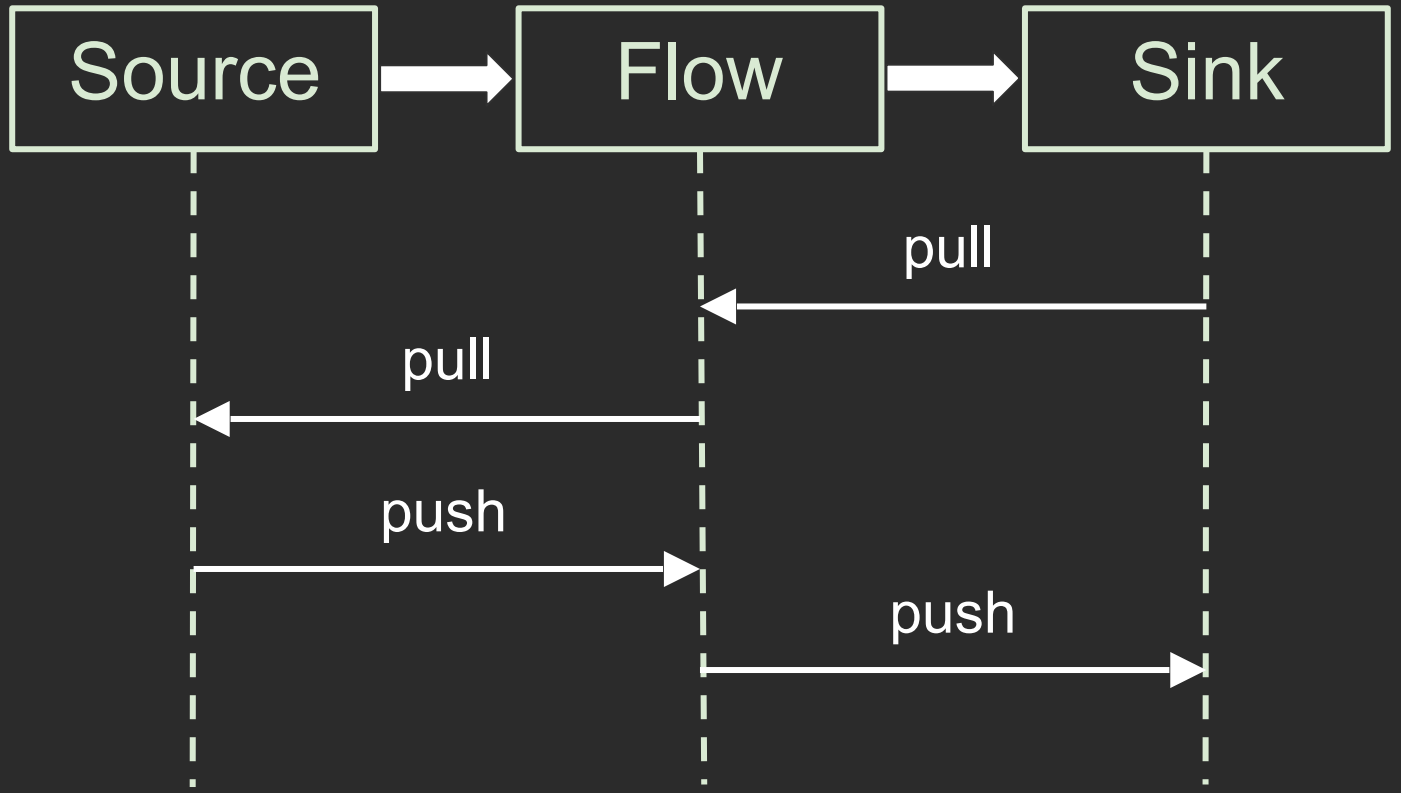
**OK** *map, filter, fold, …*

**No *flatMap*!** (there's *mapConcat* though)

**Extras** mapAsync, mapAsyncUnordered, buffer..

measurence

# How backpressure works




measurence

# Collector with Akka Streams

measurence

```
path("data") {
  post {
   extractRequest { request =>

     val source = request.entity.dataBytes
     val flow = processing()
     val sink = Sink.ignore

     source.via(flow).runWith(sink)

… } }
```

measurence

```
Flow[ByteString]
.via(chunkBuffer)
```

measurence

# Transformation tools: Framing.delimiter

```
val chunkBuffer = Framing.delimiter(
   ByteString("\n"),
   maxPresenceEventBytes,
   false
)
  .map(_.dropRight(1))
  .map(_.utf8String)
```

measurence

```
Flow[ByteString]
.via(chunkBuffer)
.via(presenceEventFactory.asStream)
```

measurence

```
Flow[ByteString]
.via(chunkBuffer)
.via(presenceEventFactory.asStream)
.via(persistentBuffer)
```

measurence

When the available stages are not enough, write your own

```
class PersistentBuffer[A] (...)
 extends GraphStage[FlowShape[A, A]] {

    val in =
      Inlet[A]("PersistentBuffer.in")

    val out =
      Outlet[A]("PersistentBuffer.out")

    override val shape = FlowShape.of(in,out)
```

# Custom stages: State

```
override def createLogic(
  attr: Attributes
): GraphStageLogic =

 new GraphStageLogic(shape) {

   var state: StageState[A] = initialState[A]
```

measurence

```
val cb = getAsyncCallback[Try[A]] {
    case Success(elements:A) =>
      state = state.copy(...)
      ...
}

queue.getCurrentQueue
   .onComplete(cb.invoke)
```

measurence

```
setHandler(in, new InHandler {
  override def onPush() = {
    val element = grab(in)
    pull(in)
    ...
  }
...
})
setHandler(out, new OutHandler {
  override def onPull(): Unit = {
    push(out, something)
  }
...
})
```

# Custom Stages: ports

measurence

# Custom Stages: Start and stop

```
override def postStop(): Unit = {
  ...
}

override def preStart(): Unit = {
  ...
}
```

measurence

```
Flow[ByteString]
.via(chunkBuffer)
.via(presenceEventFactory.asStream)
.via(persistentBuffer)
.via(kinesisPublisher)
```

measurence

# Delegating work to actors

```
Flow[KinesisEvent]
  .mapAsync(1) { event =>
    (publisher ? Publish(event))
      .mapTo[PublishCompleted]
  }
```

measurence

```
Flow[KinesisEvent]
  .mapAsync(1) { event =>
    (publisher ? Publish(event))
      .mapTo[Future[PublishCompleted]]
      .flatten
 }
```

measurence

```
Flow[ByteString]
.via(chunkBuffer)
.via(presenceEventFactory.asStream)
.via(persistentBuffer)
.via(kinesisPublisher)
.alsoTo(countEvents)
```

measurence

# Extra side-effects

```
val countEvents = Sink.foreach[Event] { _ =>
 metricEventsPerTimeUnitOfSensor.mark()
 metricEventsPerTimeUnit.mark()
}
```

measurence

# Automatic Fusion and Async Stages

A stream is handled by 1 thread

NO CROSS-THREAD COMMUNICATIONS ⇨ FASTER CODE!

NO PARALLELISM ⇨ SLOWER CODE!

A boundary will split your stream on different threads

```
source
  .via(doSomething).async
  .via(doSomething).async
  .runWith(Sink.foreach(println))
```

measurence

# Materialized Values

```
val sink: Sink[Any, Future[Done]] = Sink.ignore
```

**STREAM RUN** → Obtain a materialized value per each stage

`.runWith(sink)` is a shortcut for `.toMat(sink)(Keep.Right).run`

measurence

# Materialized Values Example: TestKit

```
val myFlow = Flow[String].map { v => v.take(1) }

val (pub, sub) = TestSource.probe[String]
  .via(myFlow)
  .toMat(TestSink.probe[Any])(Keep.both)
  .run()

sub.request(1)
pub.sendNext("Gathering")
sub.expectNext("G")
```

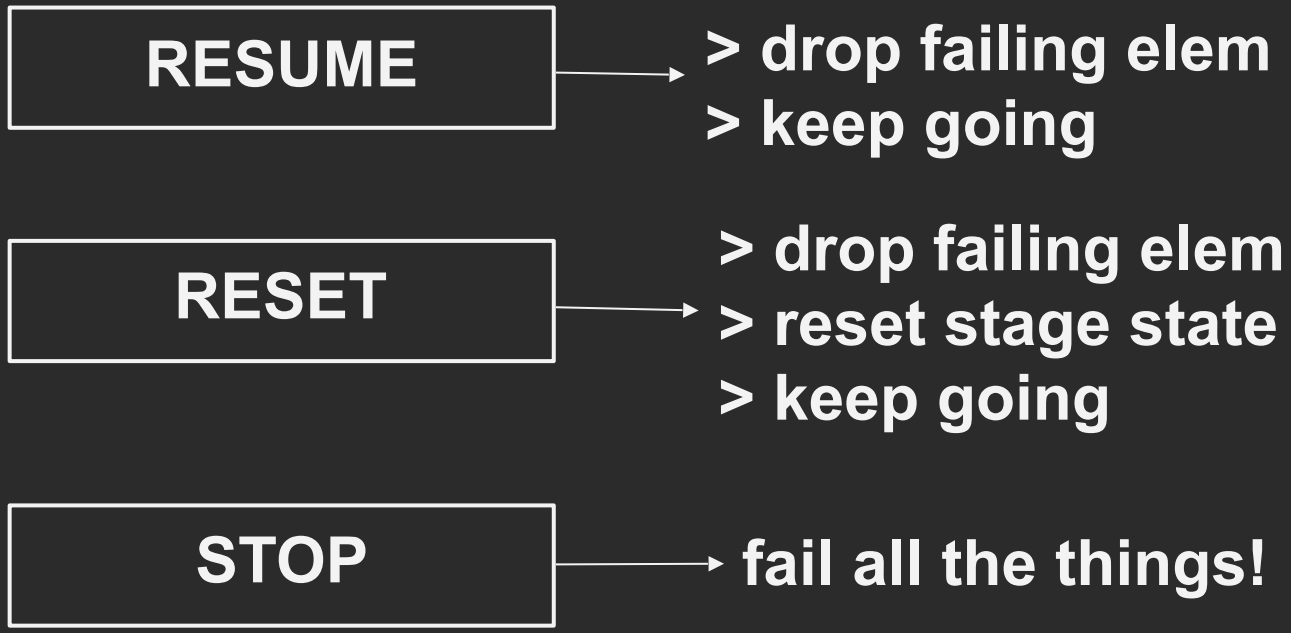
measurence

# Stream Ending: Supervisioning

```
val flow = otherFlow
  .withAttributes
    ActorAttributes.supervisionStrategy {
      case _ => Supervision.resume
    }
  )
```

measurence

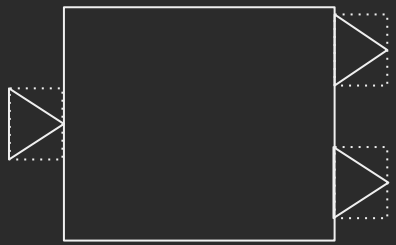
# Stream Ending: Supervisioning



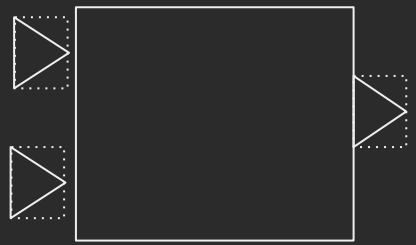

measurence

# Other shapes

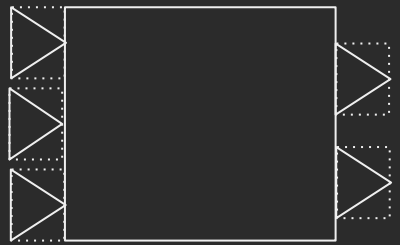
**FAN OUT**



**FAN IN**



**WHATEVER**



measurence

# Finally Got Reactive!

# In two months to production...

Memory Leaks
Data Losses

Backpressure on the network
Recovery mechanisms

measurence

# Key learnings

Actors aren’t sufficient to be reactive

Akka Streams API is rich

Backpressure as a building block

measurence

# THANK YOU!

Sleep more worry less

measurence