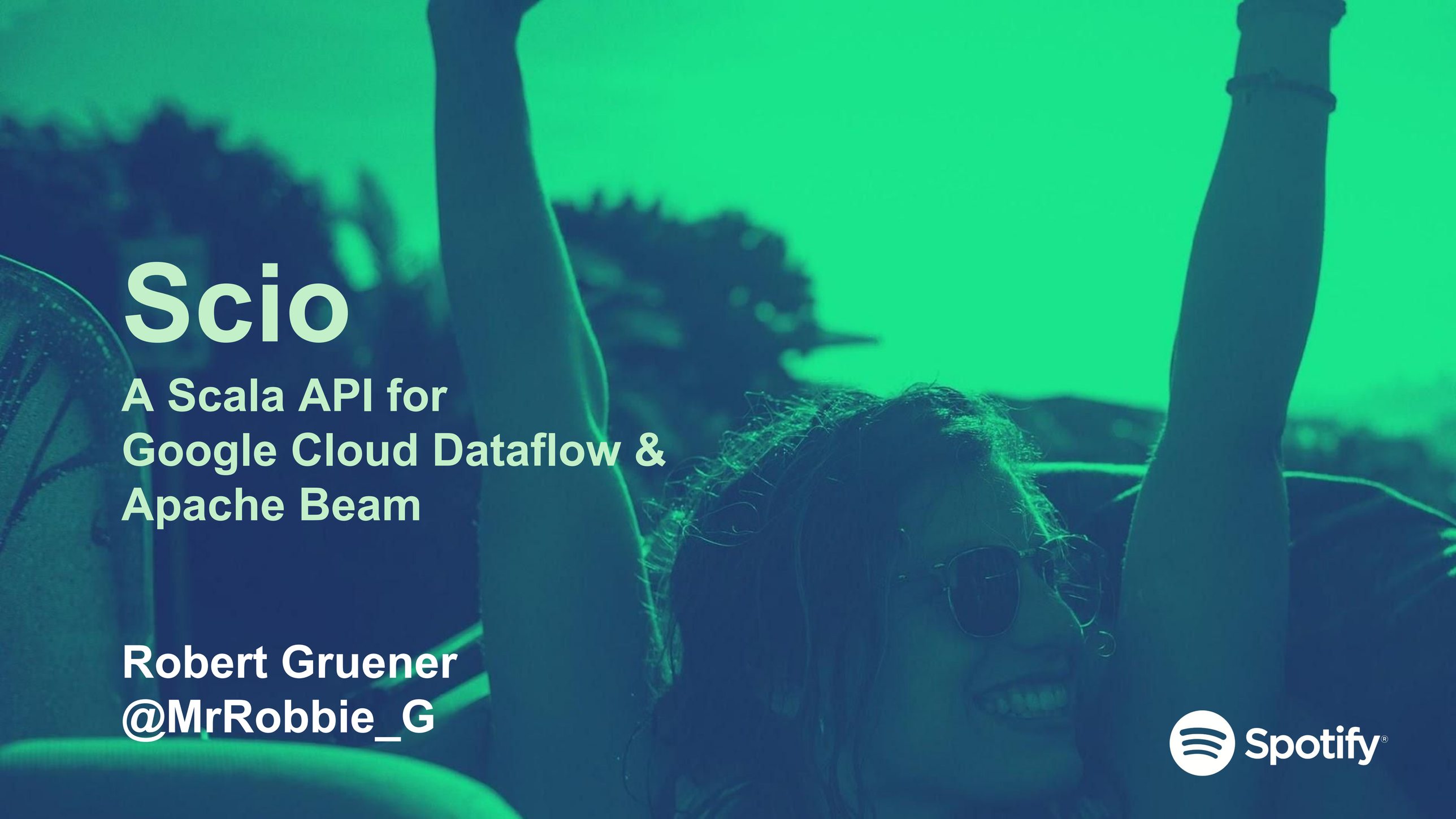


Scio

A Scala API for Google Cloud Dataflow & Apache Beam

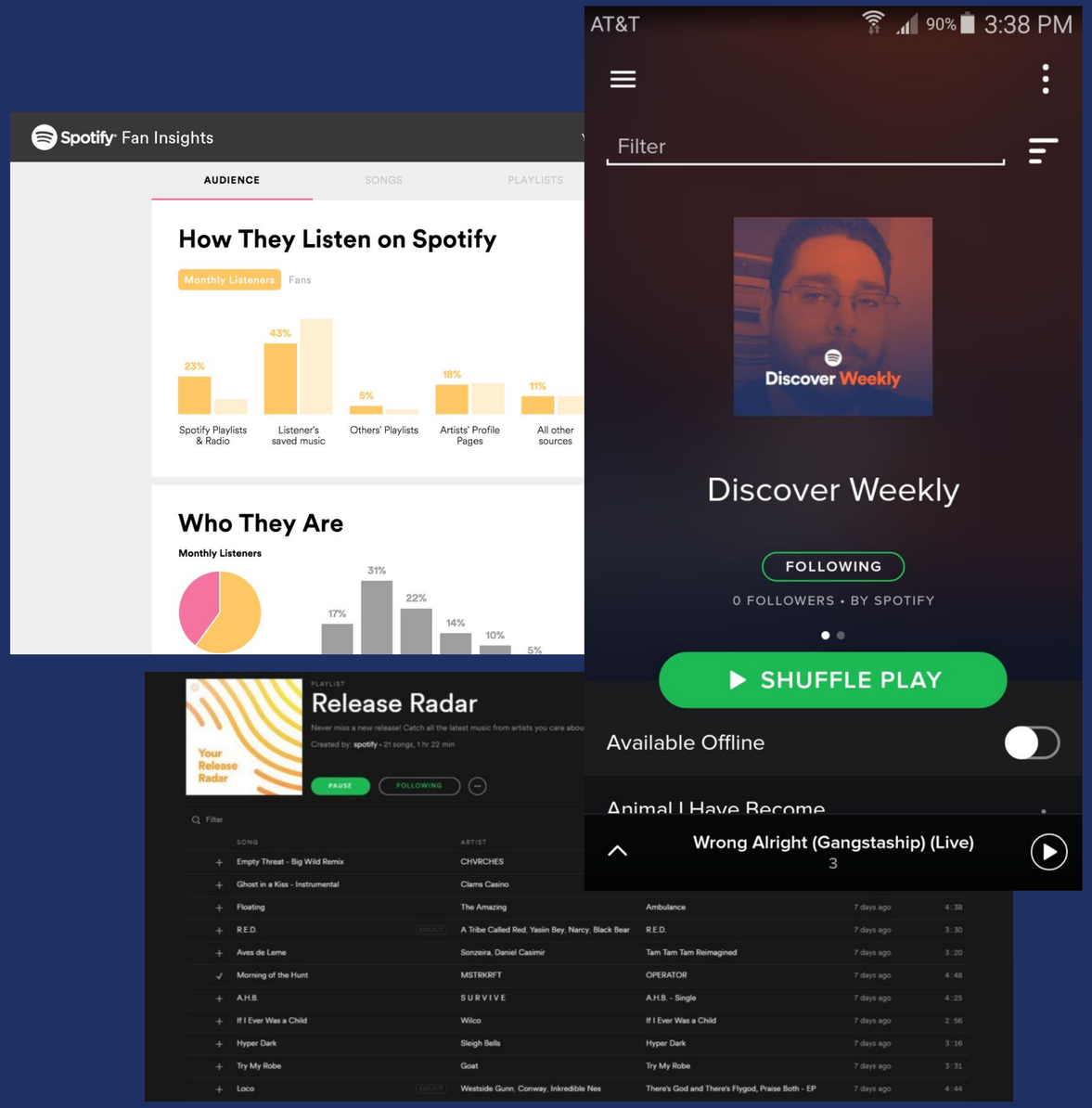
Robert Gruener
@MrRobbie_G

Spotify®

About Us

- 100M+ active users, 40M+ paying
- 30M+ songs, 20K new per day
- 2B+ playlists
- 60+ markets

- 2500+ node Hadoop cluster
- 50TB logs per day
- 10K+ jobs per day



Who am I?

- Spotify NYC since 2013
- Music recommendations - Discover Weekly, Release Radar
- Data infrastructure

Origin Story

- Python Luigi, circa 2011
- Scalding, Spark and Storm, circa 2013
- ML, recommendation, analytics
- 100+ Scala users, 500+ unique jobs

Moving to Google Cloud

Early 2015 - Dataflow Scala hack project

Spotify®

What is Dataflow/Beam?

The Evolution of Apache Beam

What is Apache Beam?

1. The Beam Programming Model
2. SDKs for writing Beam pipelines -- starting with Java
3. Runners for existing distributed processing backends
    - Apache Flink (thanks to data Artisans)
    - Apache Spark (thanks to Cloudera and PayPal)
    - Google Cloud Dataflow (fully managed service)
    - Local runner for testing

The Beam Model: Asking the Right Questions

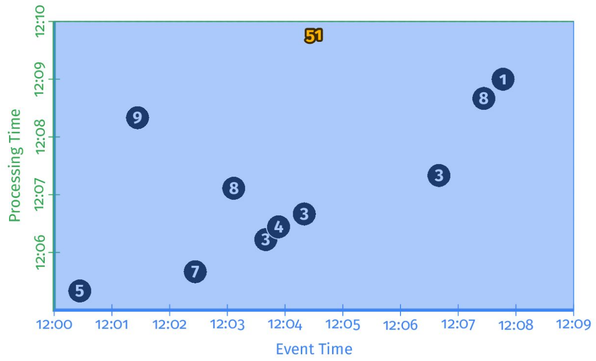
What results are calculated?

Where in event time are results calculated?

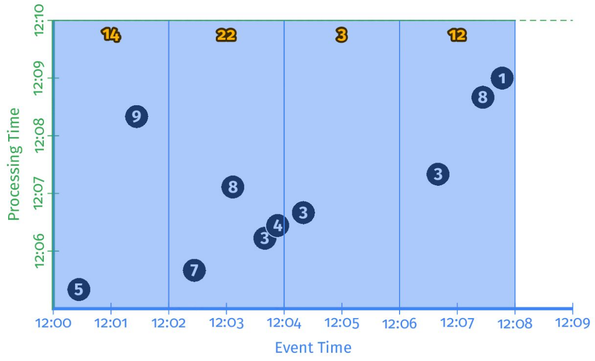
When in processing time are results materialized?

How do refinements of results relate?

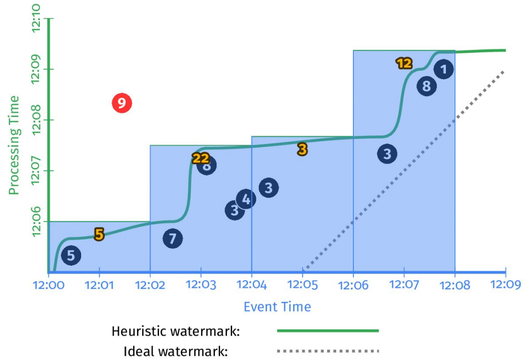
Customizing What Where When How



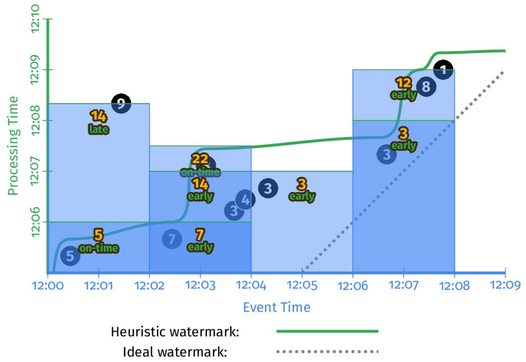
1
Classic Batch



2
Windowed Batch



3
Streaming



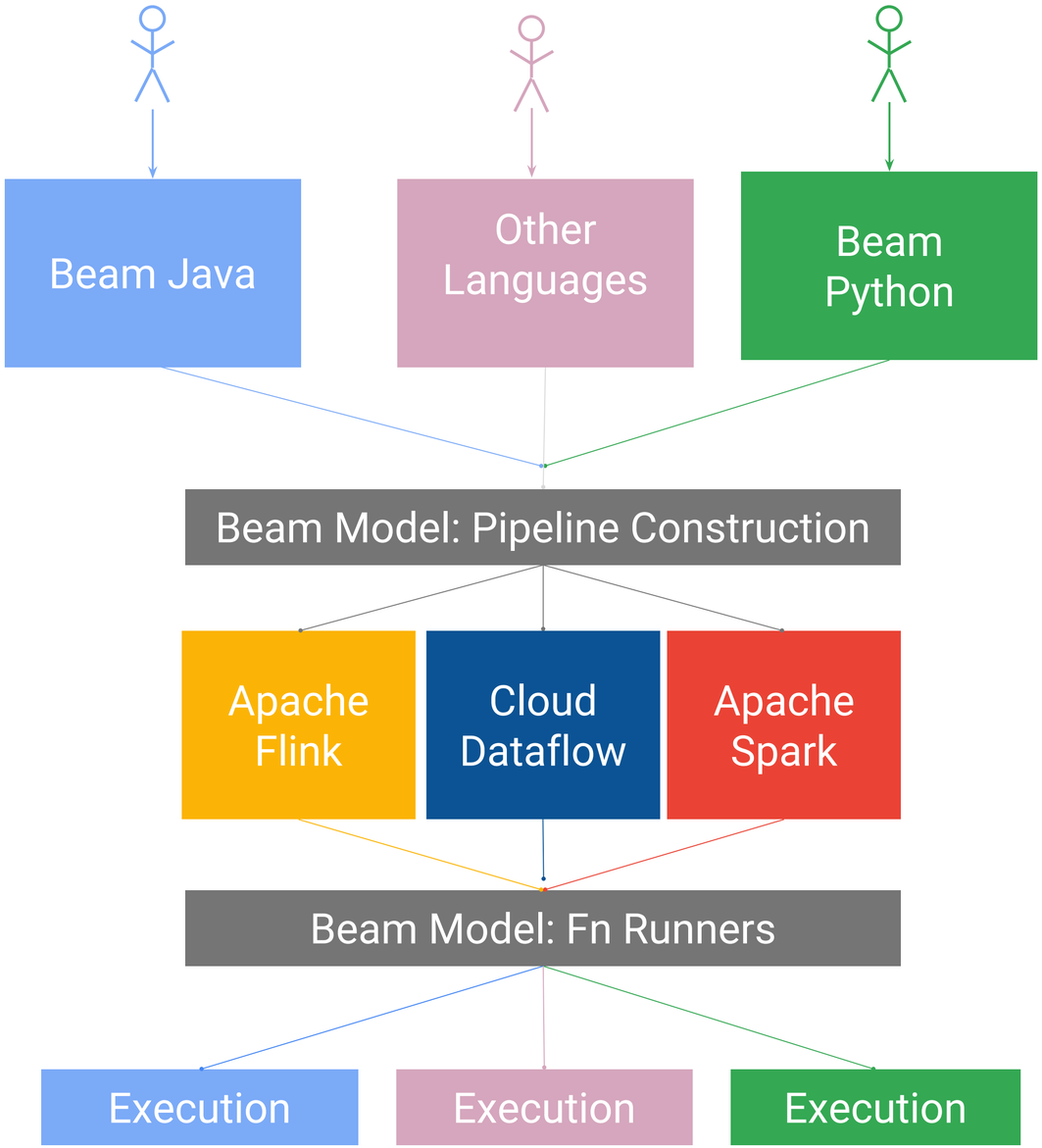
4
Streaming + Accumulation

The Apache Beam Vision

1. **End users:** who want to write pipelines in a language that's familiar.

2. **SDK writers:** who want to make Beam concepts available in new languages.

3. **Runner writers:** who have a distributed processing environment and want to support Beam pipelines



Data model

Spark

- RDD for batch, DStream for streaming
- Explicit caching semantics
- Two sets of APIs

Dataflow / Beam

- PCollection for batch and streaming
- Windowed and timestamped values
- One unified API

Execution

Spark

- One driver, n executors
- Dynamic execution from driver
- Transforms and actions

```
def countByValue()(implicit ord: Ordering[T] = null): Map[T, Long]
```

Dataflow / Beam

- No master
- Static execution planning
- Transforms only, no actions

```
def countByValue: SCollection[(T, Long)]
```

Why Dataflow/Beam?

Scalding on Google Cloud

Pros

- Community - Twitter, Stripe, Etsy, eBay
- Hadoop stable and proven

Cons

- Cluster ops
- Multi-tenancy - resource contention and utilization
- No streaming (Summingbird?)
- Integration with GCP - BigQuery, Bigtable, Datastore, Pubsub

Spark on Google Cloud

Pros

- Batch, streaming, interactive, SQL and MLLib
- Scala, Java, Python and R
- Zeppelin, spark-notebook

Cons

- Cluster lifecycle management
- Hard to tune and scale
- Integration with GCP - BigQuery, Bigtable, Datastore, Pubsub

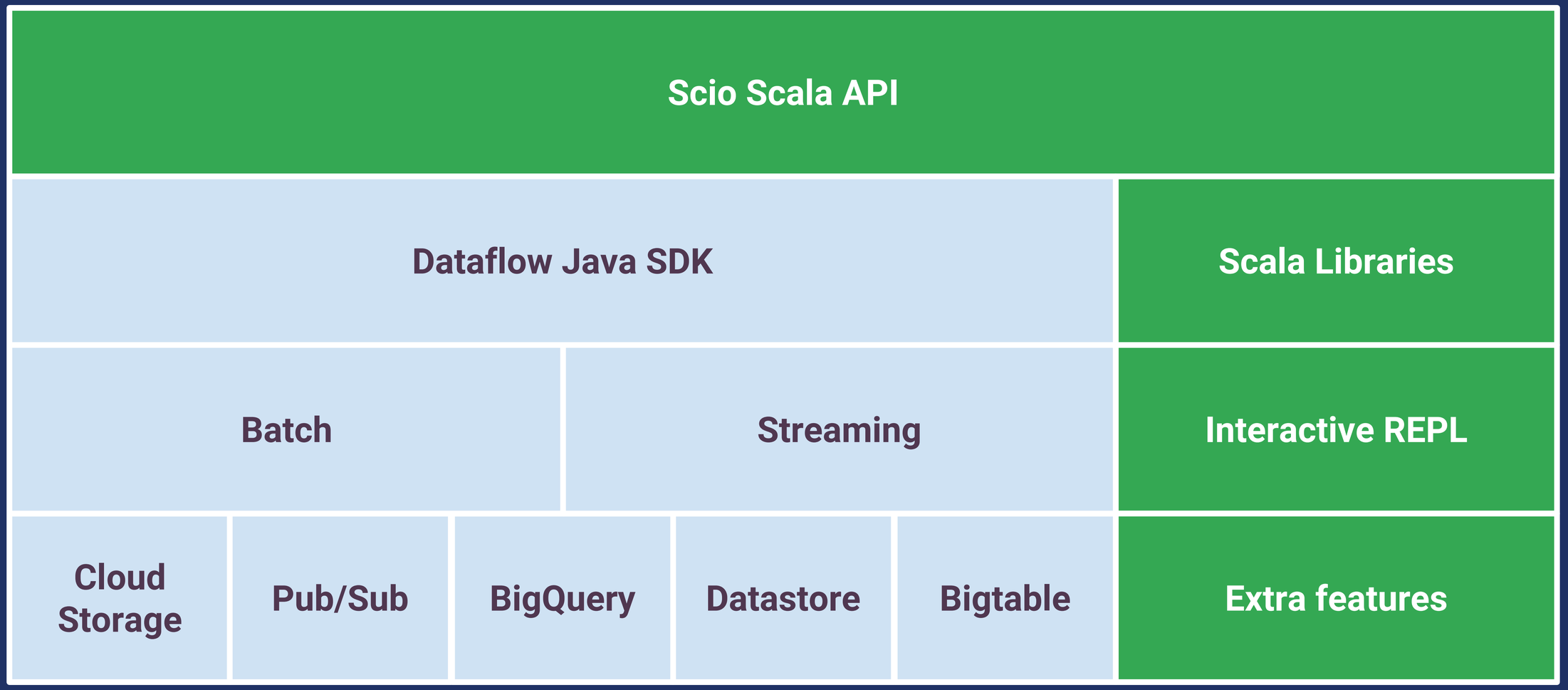
Why Dataflow with Scala

Dataflow

- Hosted, fully managed, no ops
- GCP ecosystem - BigQuery, Bigtable, Datastore, Pubsub
- Unified batch and streaming model

Scala

- High level DSL
- Functional programming natural fit for data
- Numerical libraries - Breeze, Algebird


Scio Scala API
Dataflow Java SDK
Scala Libraries
Batch
Streaming
Interactive REPL
Cloud Storage
Pub/Sub
BigQuery
Datastore
Bigtable
Extra features

Scio

Ecclesiastical Latin IPA: /ˈʃi.o/, [ˈʃiː.o], [ˈʃi.i̯o]

Verb: I can, know, understand, have knowledge.

github.com/spotify/scio
Apache Licence 2.0

WordCount

```
val sc = ScioContext()
sc.textFile("shakespeare.txt")
  .flatMap { _
    .split("[^a-zA-Z']+")
    .filter(_.nonEmpty)
  }
  .countByValue
  .saveAsTextFile("wordcount.txt")
sc.close()
```

PageRank

```
def pageRank(in: SCollection[(String, String)]) = {
  val links = in.groupByKey()
  var ranks = links.mapValues(_ => 1.0)
  for (i <- 1 to 10) {
    val contribs = links.join(ranks).values
      .flatMap { case (urls, rank) =>
        val size = urls.size
        urls.map((_, rank / size))
    }
    ranks = contribs.sumByKey.mapValues((1 - 0.85) + 0.85 * _)
  }
  ranks
}
```

Why Scio?

Type safe BigQuery

Macro generated case classes, schemas and converters

```
@BigQuery.fromQuery("SELECT id, name FROM [users] WHERE ...")
class User  // look mom no code!
sc.typedBigQuery[User]().map(u => (u.id, u.name))

@BigQuery.toTable
case class Score(id: String, score: Double)
data.map(kv => Score(kv._1, kv._2)).saveAsTypedBigQuery("table")
```

REPL

```
$ scio-repl
Welcome to
                    _____
    ________________(_)_____
    __  ___/  ___/_  /_  __ \
    _(__  )/ /__ _  / / /_/ /
    /____/ \___/ /_/  \____/   version 0.2.5

Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_11)

Type in expressions to have them evaluated.
Type :help for more information.

Using 'scio-test' as your BigQuery project.
BigQuery client available as 'bq'
Scio context available as 'sc'

scio> _
```

Available in github.com/spotify/homebrew-public

Future based orchestration

```
// Job 1
val f: Future[Tap[String]] = data1.saveAsTextFile("output")
sc1.close()  // submit job

val t: Tap[String] = Await.result(f)
t.value.foreach(println)  // Iterator[String]

// Job 2
val sc2 = ScioContext(options)
val data2: SCollection[String] = t.open(sc2)
```

DistCache

```
val sw = sc.distCache("gs://bucket/stopwords.txt") { f =>
  Source.fromFile(f).getLines().toSet
}
sc.textFile("gs://bucket/shakespeare.txt")
  .flatMap { _
    .split("[^a-zA-Z']+")
    .filter(w => w.nonEmpty && !sw().contains(w))
  }
  .countByValue
  .saveAsTextFile("wordcount.txt")
```

Other goodies

- DAG visualization & source code mapping
- BigQuery caching, legacy & SQL 2011 support
- HDFS Source/Sink, Protobuf & object file I/O
- Job metrics, e.g. accumulators
  - Programmatic access
  - Persist to file
- Bigtable
  - Multi-table write
  - Cluster scaling for bulk I/O

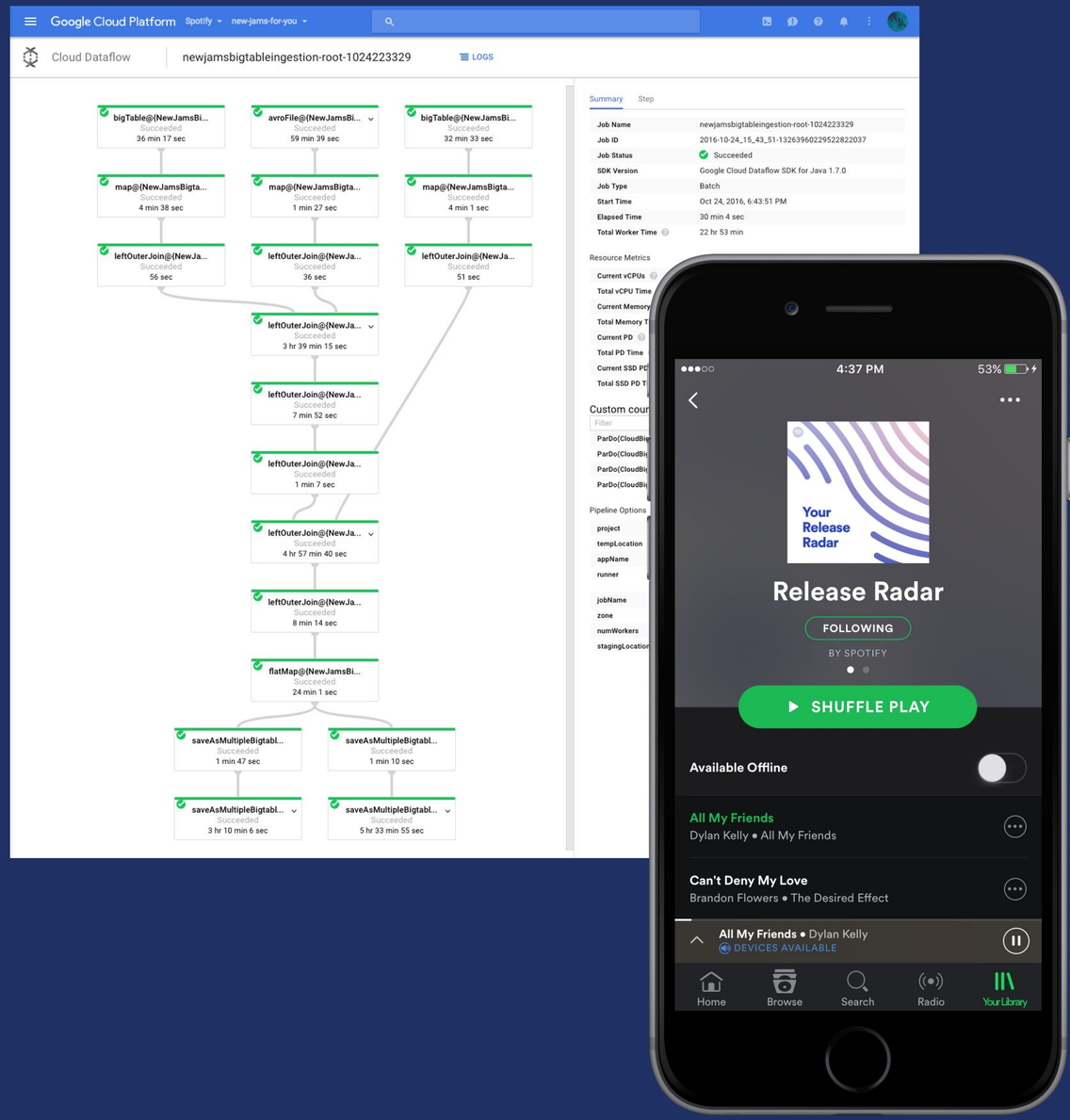
Demo Time!

Adoption

- At Spotify
  - 20+ teams, 80+ users, 70+ production pipelines
  - Most of them new to Scala and Scio
- Open source model
  - Discussion on Slack, mailing list
  - Issue tracking on public Github
  - Community driven - type safe BigQuery, Bigtable, Datastore, Protobuf

Release Radar

- 50 n1-standard-1 workers
- 1 core 3.75GB RAM
- 130GB in - Avro & Bigtable
- 130GB out x 2 - Bigtable in US+EU
- 110M Bigtable mutations
- 120 LOC



Fan Insights

- Listener stats

  [artist|track] ×

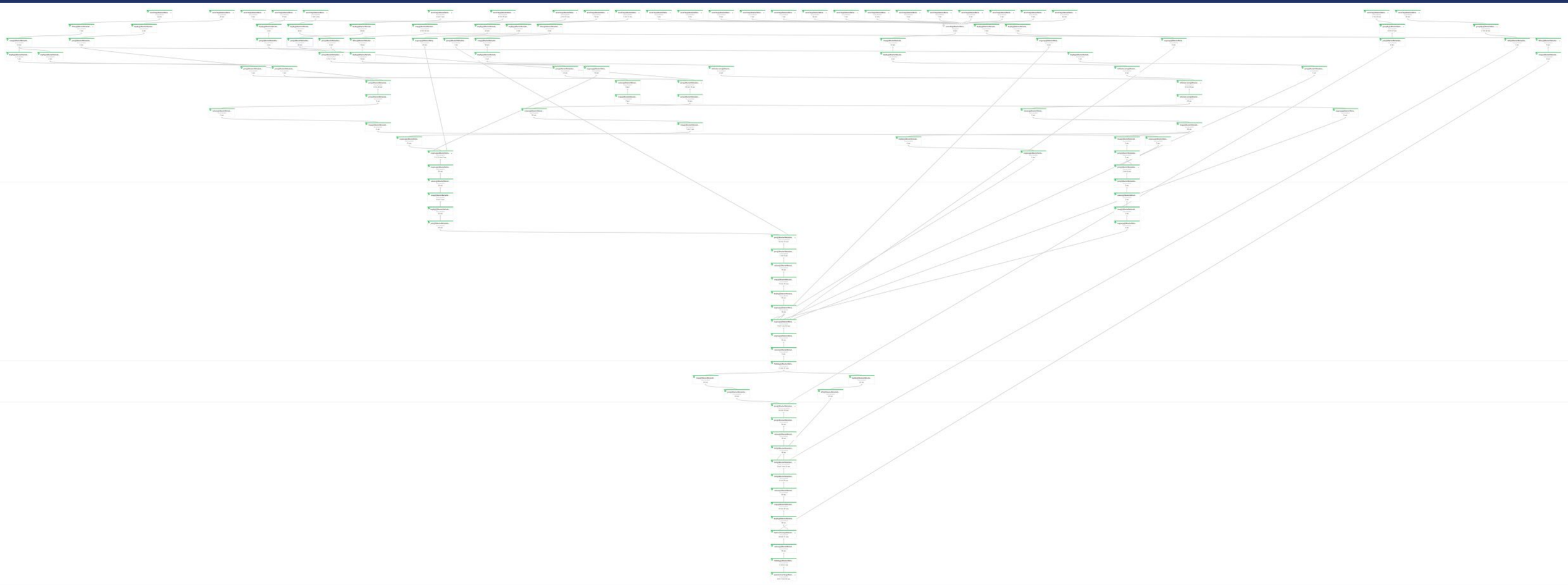
  [context|geography|demography] ×

  [day|week|month]
- BigQuery, GCS, Datastore
- TBs daily

Master Metadata

- n1-standard-1 workers
- 1 core 3.75GB RAM
- Autoscaling 2-35 workers
- 26 Avro sources - artist, album, track, disc, cover art, ...
- 120GB out, 70M records
- 200 LOC vs original Java 600 LOC

And we broke Google



BigDiffy

- **Pairwise field-level statistical diff**
- **Diff 2** `SCollection[T]` **given** `keyFn: T => String`
- **T: Avro, BigQuery, Protobuf**
- **Field level Δ - numeric, string, vector**
- **Δ statistics -** $\min, \max, \mu, \sigma$, etc.
- **Non-deterministic fields**
  - **ignore field**
  - **treat "repeated" field as unordered list**

Part of github.com/spotify/ratatool

Dataset Diff

- **Diff stats**
  - **Global:** # of SAME, DIFF, MISSING LHS/RHS
  - **Key:** key → SAME, DIFF, MISSING LHS/RHS
  - **Field:** field → min, max, μ, σ, etc.
- **Use cases**
  - **Validating pipeline migration**
  - **Sanity checking ML models**

Pairwise field-level deltas

```
val lKeyed = lhs.map(t => (keyFn(t) -> ("l", t)))
val rKeyed = rhs.map(t => (keyFn(t) -> ("r", t)))
val deltas = (lKeyed ++ rKeyed).groupByKey.map { case (k, vs) =>
  val m = vs.toMap
  if (m.size == 2) {
    val ds = diffy(m("l"), m("r")) // Seq[Delta]
    val dt = if (ds.isEmpty) SAME else DIFFERENT
    (k, (ds, dt))
  } else {
    val dt = if (m("l")) MISSING_RHS else MISSING_LHS
    (k, (Nil, dt))
  }
}
```

Summing deltas

```
import com.twitter.algebird._

// convert deltas to map of (field → summable stats)
def deltasToMap(ds: Seq[Delta], dt: DeltaType)
: Map[String, (Long, Option[(DeltaType, Min[Double], Max[Double], Moments)])] = {
  // ...
}

deltas
  .map { case (_, (ds, dt)) => deltasToMap(ds, dt) }
  .sum  // Semigroup!
```

Other uses

- AB testing
  - Statistical analysis with bootstrap and DimSum
  - BigQuery, Datastore, TBs in/out
- Monetization
  - Ads targeting
  - User conversion analysis
  - BigQuery, TBs in/out
- User understanding
  - Diversity
  - Session analysis
  - Behavior analysis
- Home page ranking
- Audio fingerprint analysis

Implementation

Serialization

- **Data ser/de**
  - **Scalding, Spark and Storm uses Kryo and Chill**
  - **Dataflow/Beam requires explicit** `Coder[T]`

    Sometimes **inferable via Guava** `TypeToken`
  - `ClassTag` **to the rescue, fallback to Kryo/Chill**
- **Lambda ser/de**
  - `ClosureCleaner`
  - `Serializable` **and** `@transient lazy val`

REPL

- Spark REPL transports lambda via HTTP
- Dataflow requires job jar for execution (no master)
- Custom class loader and ILoop
- Interpreted classes → job jar → job submission
- `SCollection[T]#closeAndCollect(): Iterator[T]` to mimic Spark actions

Macros and IntelliJ IDEA

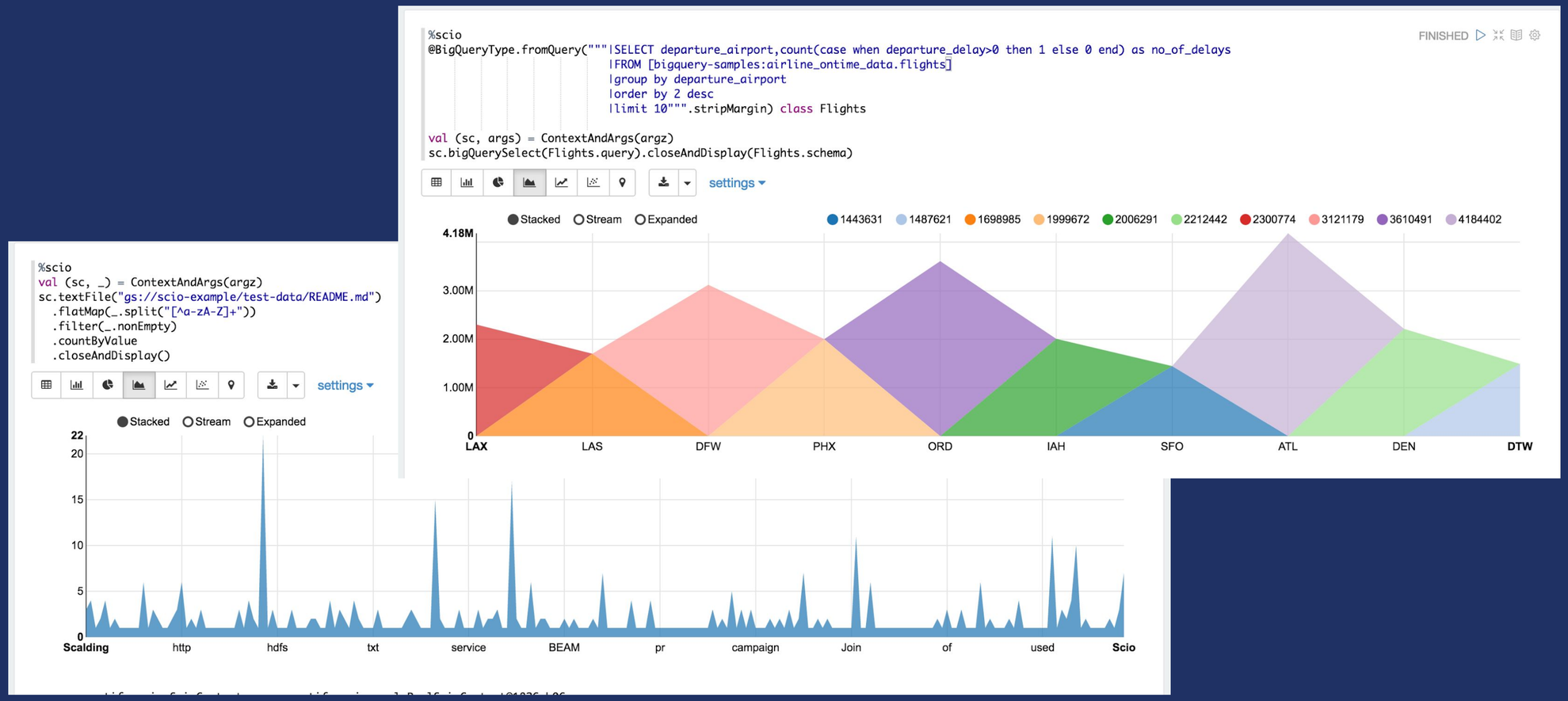
- **IntelliJ IDEA does not see macro expanded classes**

  https://youtrack.jetbrains.com/issue/SCL-8834
- `@BigQueryType.{fromTable, fromQuery}`

  `class MyRecord`
- **Scio IDEA plugin**

  https://github.com/spotify/scio-idea-plugin

Scio in Apache Zeppelin



Local Zeppelin server, remote managed Dataflow cluster, NO OPS

What's Next?

- Better streaming support [#163]
- Working branch on Beam 0.2.0-incubating
- Support other runners
- Donate to Beam as Scala DSL [BEAM-302]

The End
Thank You

Robert Gruener
@MrRobbie_G