Data Processing with **Scio**

Neville Li | SmartData 2020

Spotify

who am I?

- Spotify NYC since 2011
- Music Recommendations
- Data Infrastructure
- Prev. Yahoo! Search

spotify / data

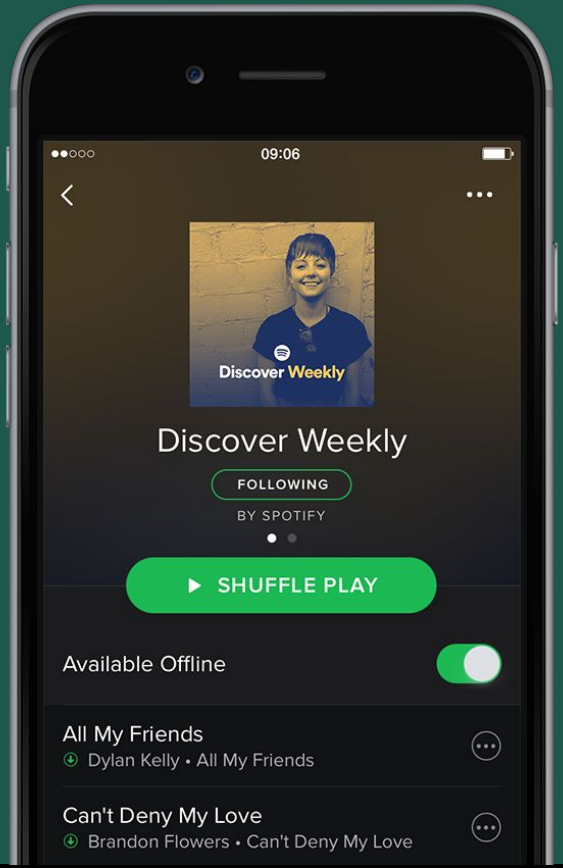
- Audio streaming service
- **144M+** Subscribers
- **320M+** Monthly Active Users
- **60M+** Songs
- **1.9M+** Podcast Titles
- **4B+** Playlists
- **92** Markets

Spotify®

Привет!

spotify / data

- Discover Weekly
- Release Radar
- Daily Mixes
- Yearly Wrapped
- Fan Insights
- ...



Spotify

spotify / scala

- Scala for **Data Engineering**
- **Scala Center** advisory board member
- Open-source :)
  - **https://github.com/spotify/scio**
  - **https://github.com/spotify/featran**
  - **https://github.com/spotify/ratatool**
  - **https://github.com/spotify/magnolify**
  - **...**

Spotify®

scalability

- Volume of data
- Number of datasets
- Number of data engineers / data scientists

challenges

- Orchestration
- Discovery
- Lineage
- Access Control
- Privacy
- Observability
- Quality
- Storage
- Productivity
- 💰💰💰

Data at Spotify

spotify / hadoop

- On-premise ➜ Amazon EMR ➜ On-premise
- ~2,500 nodes
- 100PB+ disk, 100TB+ RAM
- 60TB+ log ingestion / day
- 20K+ jobs / day

Spotify®

data processing / **past**

- Luigi, Python Map/Reduce, ~2011

- Scalding, Spark, ~2013

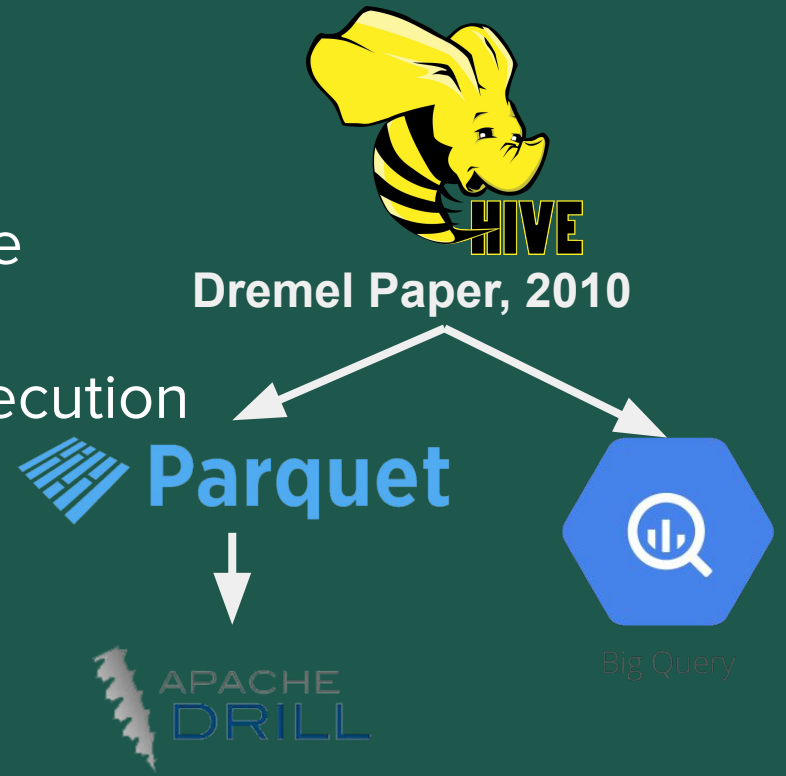
- Storm - real time

- Hive - ad hoc analysis

Moving to Google Cloud

Hive ➜ **BigQuery**

- Full row scans ➜ columnar storage
- Map/Reduce jobs ➜ optimized execution
- Batch ➜ interactive
- Apache Beam integration

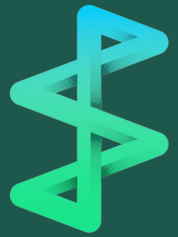


more **serverless**

- Kafka ➜ Pubsub
- Cassandra ➜ Bigtable
- Hadoop, Scalding, Storm ➜ Scio + Beam + Dataflow
- Bare metal ➜ GKE

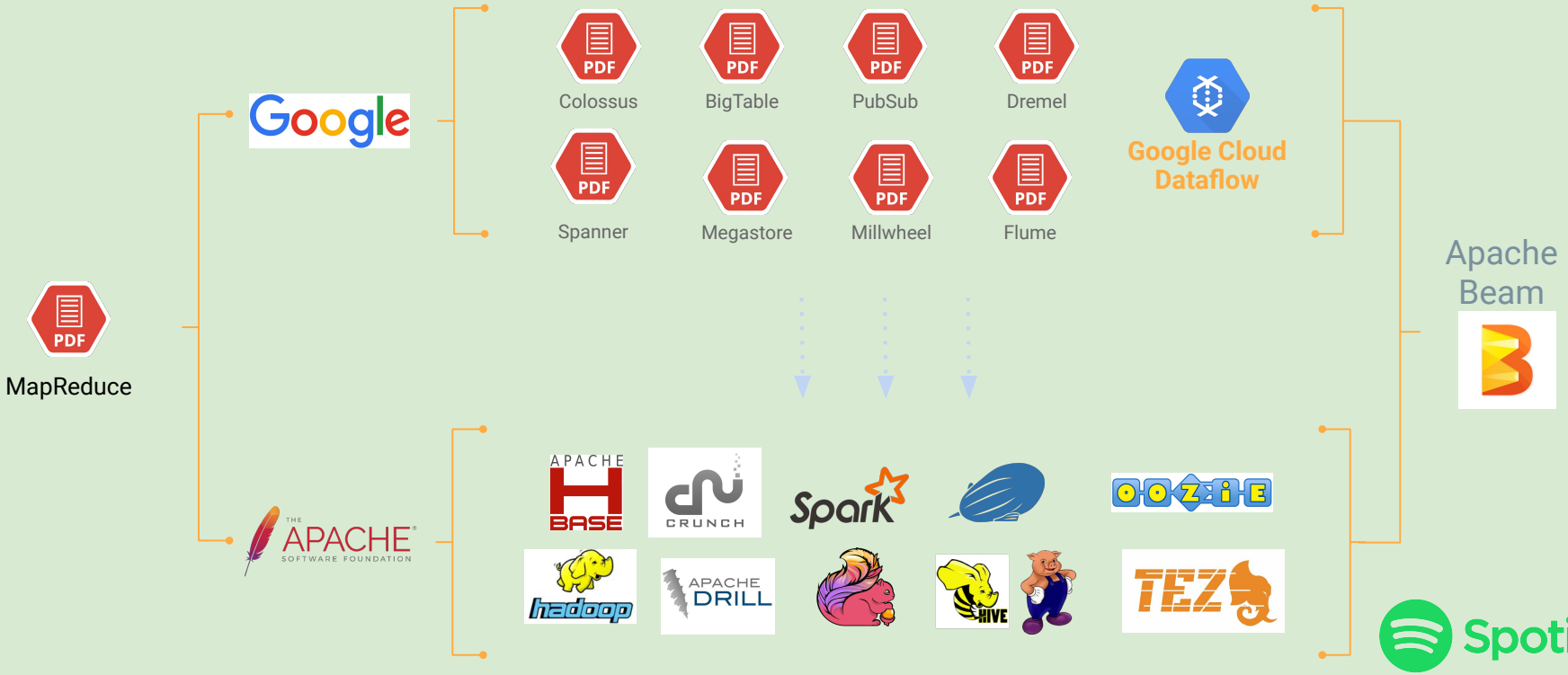
data processing / **present**

- Scio
- Apache Beam
- Cloud Dataflow
- BigQuery
- Bigtable, TensorFlow, Spanner, ...



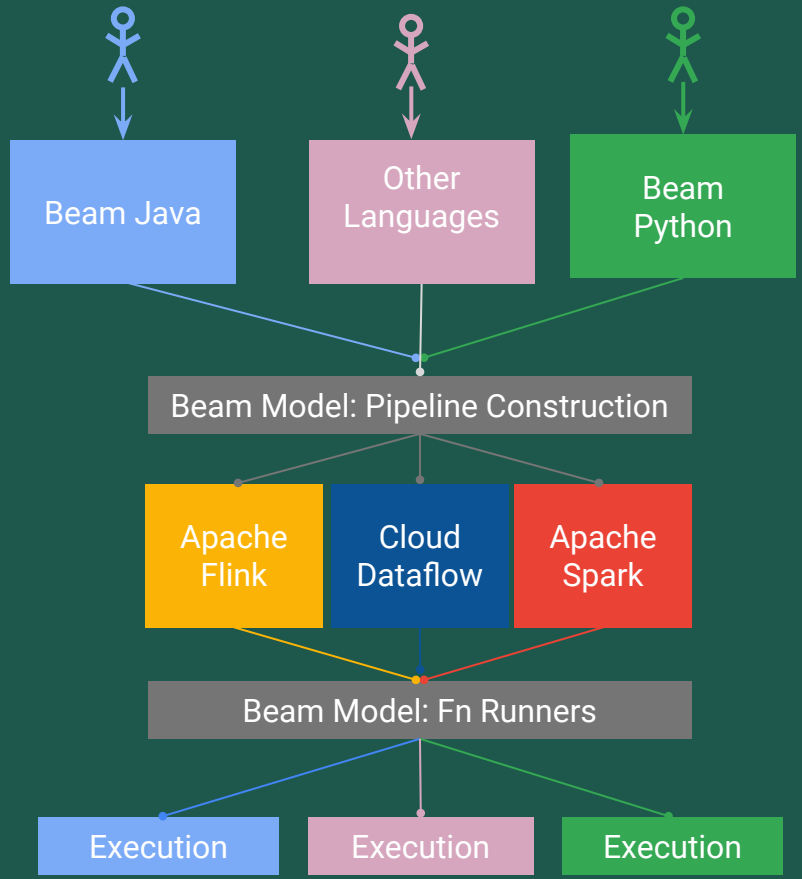

data processing / **beam**

slides by Frances Perry & Tyler Akidau, April 2016



Apache Beam

- Unified Batch and Streaming
- SDKs - Java, Python, Go
- Runners
  - Google Cloud Dataflow
  - Apache Flink
  - Apache Spark

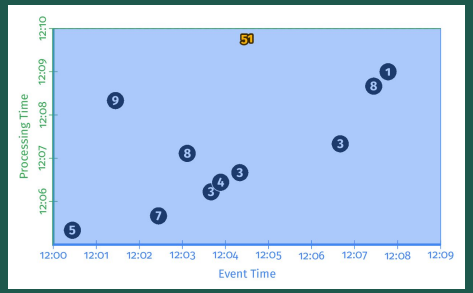


Spotify

the **beam** model

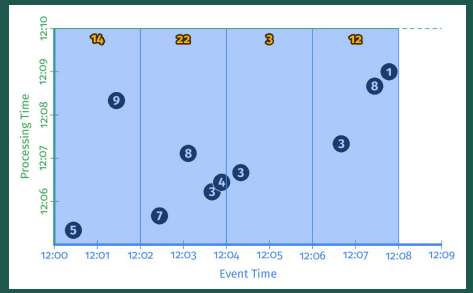
- *What* results are calculated?
- *Where* in event time are results calculated?
- *When* in processing time are results materialized?
- *How* do refinements of results relate?

Customizing What Where When How



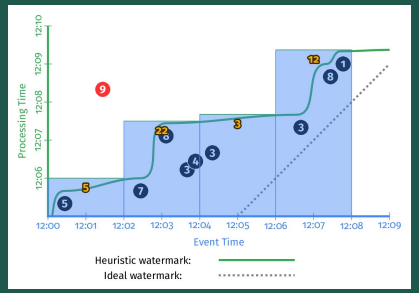
1

Classic Batch



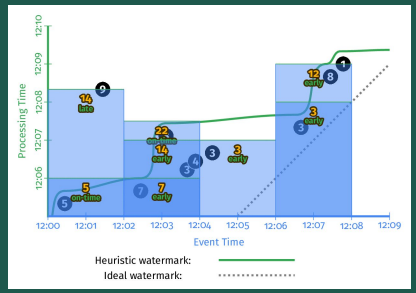
2

Windowed Batch



3

Streaming



4

Streaming + Accumulation

Spotify®

beam + dataflow

- Hosted, fully managed, no ops
- Autoscale, dynamic work re-balance
- Shuffle service
- GCP ecosystem integration
  - BigQuery, Bigtable, Datastore, Pubsub, Spanner

beam + scala

- High level DSL
- Familiarity with Scalding, Spark or Flink
- Functional programming natural fit for data
- Numerical libraries - Breeze, Algebird
- Macros for code generation

Scio

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

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

data processing / **scio**

- A **Scala** API for data processing
- For Apache Beam (Java SDK)
- Unified **batch** and **streaming**
- Runs on Dataflow, Spark, Flink, ...
- **Open Source** (Apache v2.0)

word count / **scio**

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

Spotify

page rank / **scio**

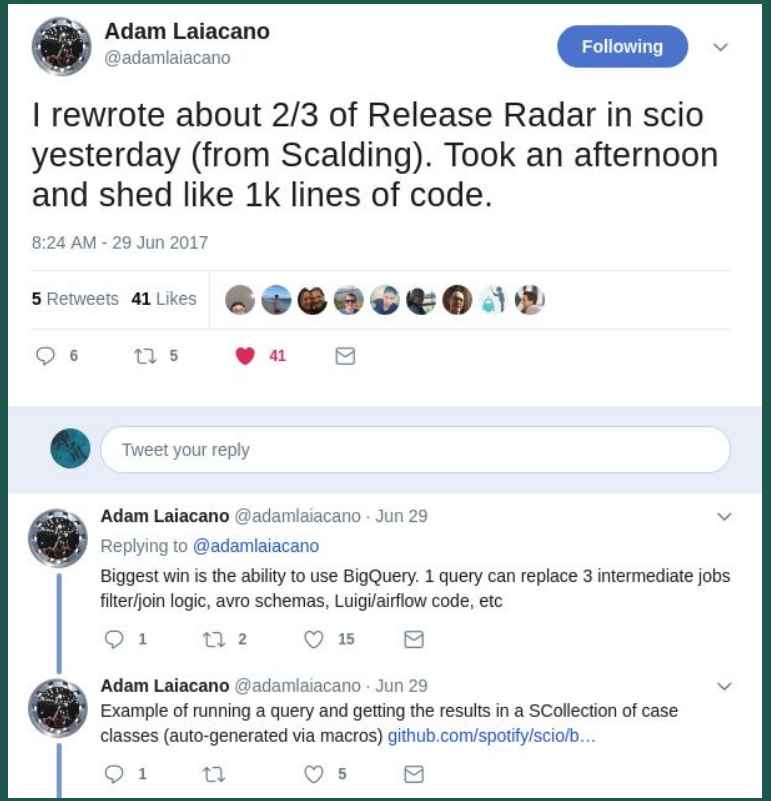
```
def pageRank(in: SCollection[(String, String)]) = {
  val links = in.groupByKey()
  var ranks = links.mapValues(_ => 1.0)
  for (i <- 1 to 10) {
    val contribs = links.join(ranks).values
      .flatMap { case (urls, rank) =>
        urls.map((_, rank / urls.size))
    }
    ranks = contribs.sumByKey.mapValues((1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

Spotify

bigquery + **scio**

Macro generated types, schemas & converters

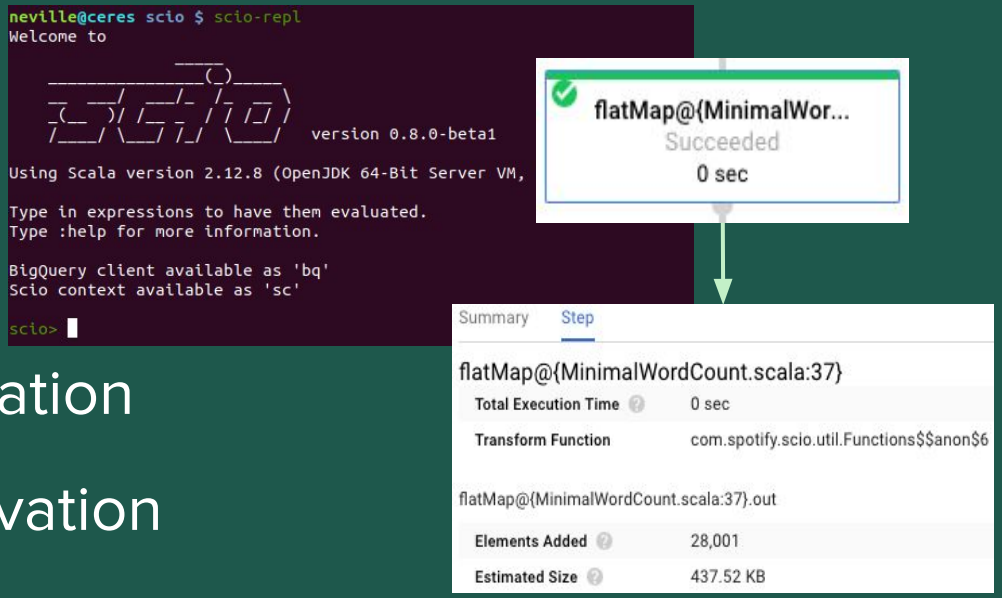
```
@BigQuery.fromQuery(
  "SELECT id, name FROM [users] WHERE ...")
class User  // look mom no code!
sc.typedBigQuery[User]().map(u => (u.id, u.name))

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



Spotify

scio + extras

- Interactive REPL
- DAG and source visualization
- Compile time coder derivation
- Join optimizations - hash, skewed, sparse
- TensorFlow, Protobuf, Elasticsearch, Parquet, etc.



Spotify

scio + bigdiffy

- Pairwise field-level statistical diff for datasets
- Diff 2 SCollection[T] given keyFn: T => String
- T: Avro, BigQuery TableRow, Protobuf, case classes
- Leaf field Δ - numeric, string (Levenshtein), vector (cosine)
- Δ statistics - min, max, μ, σ, etc.
- Non-deterministic fields - ignore or treat as unordered (list => set)

github.com/spotify/ratatool/tree/master/ratatool-diffy

scio + bigdiffy

- 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
  - Validate pipeline migration, e.g. Python Luigi ➜Scio
  - ML data quality check

scio + featran

- Type safe feature transformer for machine learning
- Column-wise aggregation and transformation
- Single shuffle for arbitrary # of features
- In memory, Scio, Scalding, Flink & Spark backends
- Scala collection, array, Breeze, TensorFlow & NumPy output format

github.com/spotify/featran

Spotify

scio + featran

```
case class Account(age: Int, income: Double, balance: Double, profession: Option[String])

val spec = FeatureSpec.of[Account]
  .required(_.age.toDouble)(Bucketizer("age", Array(0.0, 21.0, 40.0, 65.0)))
  .required(_.income)(StandardScaling("income"))
  .required(_.balance)(QuantileDiscretizer("balance", 10))
  .optional(_.profession)(OneHotEncoder("profession"))

val f = spec.extract(accounts)
f.featureNames                          // human readable names
f.featureValues[DenseVector[Double]]    // output format
f.featureSettings                       // settings can be reloaded, i.e. in service
```

spotify / **scio**

- **450+** users

  (Data engineers, data scientists, backend engineers)
- **4000+** unique production jobs

  (Dec 2020)
- Storage: Avro, Protobuf, BigQuery, Bigtable, ...
- Batch and streaming

spotify / **scio**

- 1000+ pipeline repos
- Internal "monorepo" of libraries
  - Encryption
  - Monitoring
  - Data quality
  - Capacity planning
- sbt plugin for common settings & tasks
- scala-steward for auto-bumping

deep dive / **serialization**

- One of the most expensive parts of a data pipeline
- Elements in memory ➜ disk ➜ network during groups, joins, etc.
- Every **PCollection<T>** requires a **Coder<T>**:

```
public abstract void encode(T value, OutputStream outStream);

public abstract T decode(InputStream inStream);
```

Spotify

serialization / **scio < 0.7.0**

- Serialization was done using Java Kryo library
  - +Chill (Scala extension)
- Reflection based, class name overhead
- Used in Spark, Storm, Scalding, etc.
- Lack of compile time types
- Can't leverage runner optimization/check based on coder properties
  - Determinism (GBK)
  - Consistent with equals (performance)
  - Size estimation

serialization / **scio >= 0.7.0**

- **Kryo** to **typesafe** coders
- Kryo - runtime reflection, speed & size penalty
- New coder - **compile time** derivation with Magnolia
- Runner type hinting - Is **T** encoding deterministic? consistent with equals?
- **Fallback** to Kryo

serialization / **scio 0.6 → 0.7**

0.6

(mostly)
automated migration
using scalafix

0.7

```
def map[U: ClassTag]
  (f: T => U): SCollection[U]
```

Unsafe, Kryo based, **runtime** reflection

```
def map[U: Coder]
  (f: T => U): SCollection[U]
```

Safer, simpler, **compile time**, deterministic

Spotify®

deep dive / **smb**

- **S**ort **M**erge **B**ucket join
- Bucket data by key bytes
- Sort bucket elements by key bytes
- Store 1 file per bucket
- Downstream join = merge sort of matching bucket files
- No shuffle

smb / **bucketing**

	id	
	8 7 6 0 3 1 7 2 6 1 4 3 3 4 5	

$$hash(\texttt{id}) \mod B$$

B (number of buckets) = 3

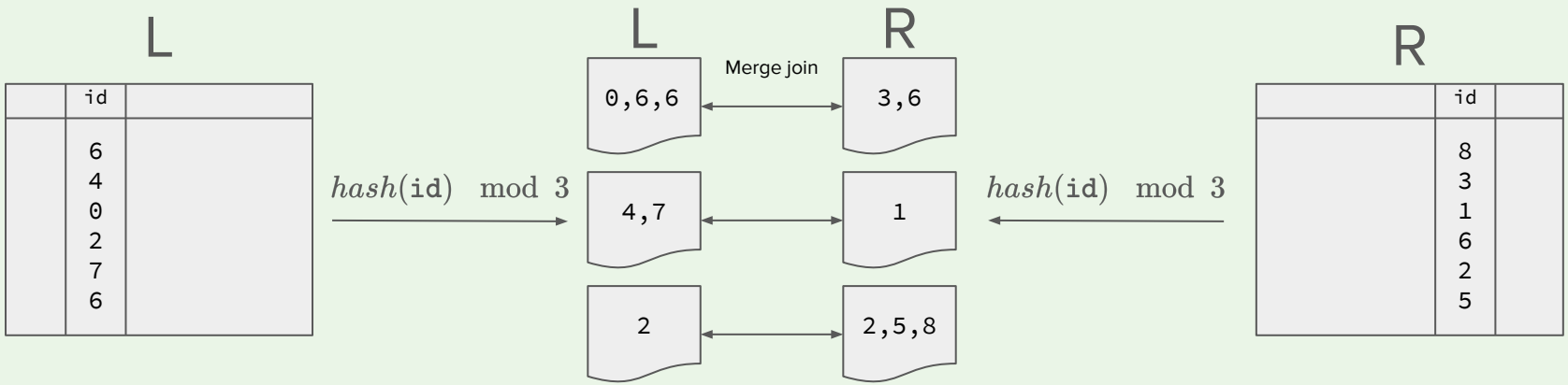
	id	
	0 3 3 3 6 6	

	id	
	1 1 4 4 7 7	

	id	
	2 5 8	

Spotify

smb / **joining**



Spotify

deep dive / **smb**

- Shuffle once, join everywhere ➜ amortized cost
- scio-smb since Scio >= 0.8.0
- Better compression, storage saving
- Some planning required
  - Core datasets
  - Key semantic & encoding
  - Bucket settings

deep dive / **wrapped 2019**

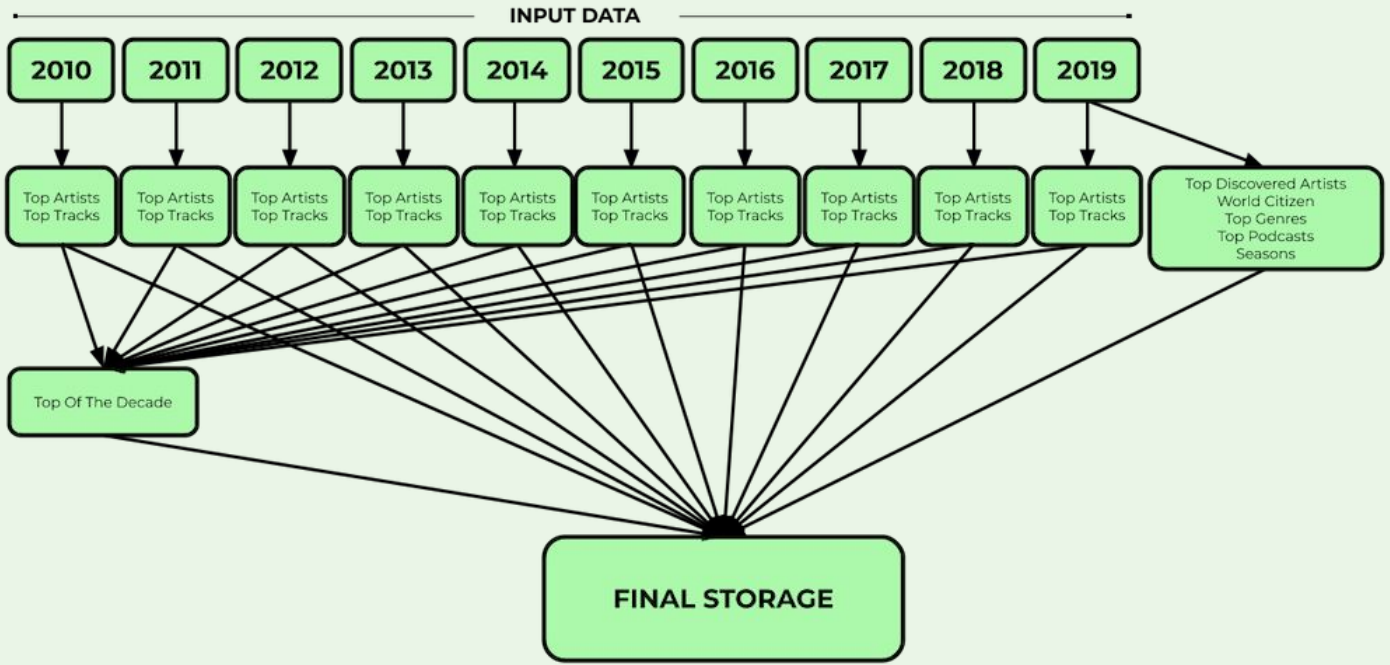
deep dive / **wrapped 2019**

- **2018 Wrapped** - one massive job crunching yearly data
  a. Largest shuffle in GCP Dataflow history!
- **2019 Wrapped** - many small jobs sharing intermediate data in database
  a. Bigtable range scans with Scio AsyncDoFn & Bigtable client directly
  b. Parallelize computation as much possible
    - i.e. computations for each year can run independently of each other
    - 5x as much data processed, -25% total cost
- **2020 Wrapped** - even more speed up with SMB, stay tuned :)

Spotify

deep dive / **wrapped 2019**

source: https://labs.spotify.com/2020/02/18/wrapping-up-the-decade-a-data-story



deep dive / **wrapped 2019**

- How Spotify ran the largest Google Dataflow job ever for Wrapped 2019
- Spotify Unwrapped: How we brought you a decade of data

https://techcrunch.com/2020/02/18/how-spotify-ran-the-largest-google-dataflow-job-ever-for-wrapped-2019/
https://labs.spotify.com/2020/02/18/wrapping-up-the-decade-a-data-story/

next steps / **scio**

- Optimizations
- Better support for different runners
- Streaming
  - Stateful DoFn
  - Refreshing side inputs
- Scala 3

next steps / **scio**

- Optimizations
- Better support for different runners
- Streaming
  - Stateful DoFn
  - Refreshing side inputs
- Scala 3

Questions?

Neville Li
@sinisa_lyh

github.com/spotify/scio
spotify.github.io/scio

slackin.spotify.com

spotifyjobs.com

Spotify®