What’s new in Spark 2.0?

Rerngvit Yanggratoke @ Combient AB
Örjan Lundberg @ Combient AB

Machine Learning Stockholm Meetup
27 October, 2016
Schibsted Media Group

Combient - Who We Are

- A joint venture owned by several Swedish global enterprises in the Wallenberg sphere

- One of our key missions is the Analytics Centre of Excellence (ACE)
  - Providing *data science and data engineer* resources to our owners and members
  - Sharing best practises, advanced methods, and state-of-the-art technology to generate *business value* from data

Spark 2.0 “Easier, Faster, and Smarter”

- Spark Session - a new entry point
- Unification of APIs => Dataset & Dataframe
- Spark SQL enhancements: subqueries, native Window functions, better error messages
- Built-in CSV file type support
- Machine learning pipeline persistence across languages
- Approximate DataFrame Statistics Functions
- Performance enhancement (Whole-stage-code generation)
- Structured streaming (Alpha)

Spark Session

Spark 1.6 => SparkContext, SQLContext, HiveContext

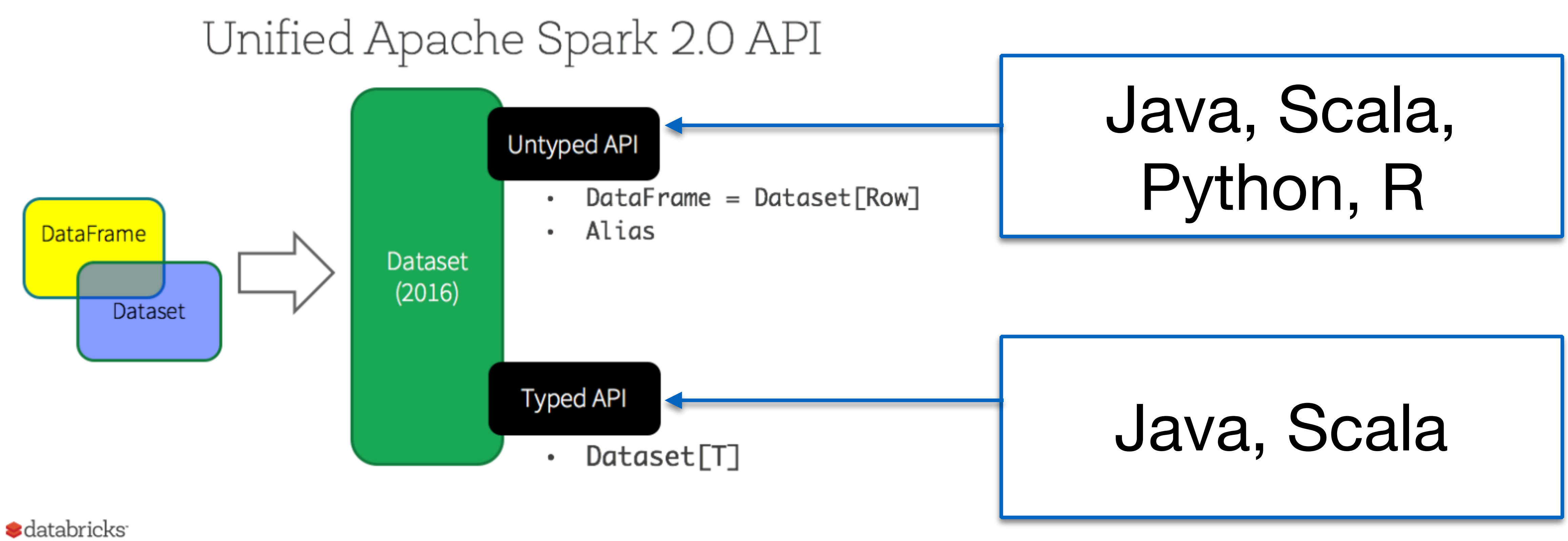
```
val conf = new SparkConf().setMaster(master).setAppName(appName)
val sc   = new SparkContext(conf)
val sqlContext  = new org.apache.spark.sql.SQLContext(sc)
val hiveContext = new org.apache.spark.sql.hive.HiveContext(sc)
```

Spark 2.0 => SparkSession

```
val ss = SparkSession.builder().master(master)
         .appName(appName).getOrCreate()
```

Unified API: DataFrame & Dataset

- DataFrame - introduced in Spark 1.3 (March 2015)
- Dataset - experimental in Spark 1.6 (March 2016)



- Observation: very rapid changes of APIs

Dataset API example

THE WORLD BANK
IBRD • IDA

name	population_2014	population_2015
Afghanistan	*XXXX*	*XXXX*
Albania	*XXXX*	*XXXX*
....	....	....

Table: country_pop

What countries has a shrinking population from 2014 to 2015?

```
> case class CountryData(name: String, population_2014: Long, population_2015: Long)
  val countryDS = ss.sql(" select * from country_pop ").as[CountryData]
  countryDS.filter(c => {c.population_2015 < c.population_2014}).show(10, false)
```

```
+--------+---------------+---------------+
|name    |population_2014|population_2015|
+--------+---------------+---------------+
|Japan   |127131800      |126958472      |
|Spain   |46480882       |46418269       |
|Ukraine |45362900       |45198200       |
|Poland  |38011735       |37999494       |
|Romania |19908979       |19832389       |
+--------+---------------+---------------+
```

Compile time error detection

- Key benefit over DataFrame API: static typing!

Subqueries

Ex. Which countries have more than 100M people in 2016?

Spark 1.6 => Subqueries inside FROM clause

```
> ss.sql(""" SELECT *
            FROM (SELECT * FROM country_pop WHERE population_2014 > 100e6) t1
            ORDER BY population_2014 DESC LIMIT 5""").show(10, false)
```

Spark 2.0 => Subqueries inside FROM and WHERE clause

```
> ss.sql(""" SELECT *
            FROM country_pop
            WHERE population_2014 IN (select population_2014 from country_pop WHERE population_2014 > 100e6)
            ORDER BY NAME ASC LIMIT 5 """).show(10, false)
```

```
+----------+---------------+---------------+
|name      |population_2014|population_2015|
+----------+---------------+---------------+
|Bangladesh|159077513      |160995642      |
|Brazil    |206077898      |207847528      |
|China     |1364270000     |1371220000     |
|India     |1295291543     |1311050527     |
|Indonesia |254454778      |257563815      |
+----------+---------------+---------------+
```

Window Functions

Spark 1.X => required HiveContext

Spark 2.0 => native window function

```
> // PARTITION BY id  ORDER BY cykle ROWS BETWEEN 2 PRECEDING AND 2 FOLLOWING (5)
  import org.apache.spark.sql.expressions.Window
  import org.apache.spark.sql.functions._
  val w = Window.partitionBy("id").orderBy("cykle").rowsBetween(-2, 2)

  val x = dft.select($"id",$"cykle",avg($"value").over(w))
  x.show
```

```
dft.createOrReplaceTempView("dft")
val sqlq = """
            SELECT id,cykle,avg(value) OVER (PARTITION BY id ORDER BY cykle ASC ROWS BETWEEN 2 PRECEDING AND 2 FOLLOWING)
            FROM dft
"""

val dfSQL = sqlContext.sql(sqlq)
```

SQL Error messages

Finding typos/errors in larger SQL?

```
> sqlloggik4_df = """
  SELECT *
    , CAST(id as BIGINT) *10000 + SUM(new_session)
      OVER (PARTITION BY id ORDER BY starttid)
       AS session_id
  FROM(
  SELECT *,
      unix_timestamp(l.starttid) - LAG(unix_timestamp(l.starttid)) OVER (PARTITION BY l.id ORDER BY  l.starttid) timesincelast,
      CASE
      WHEN unix_timestamp(l.starttid) - LAG(unix_timestamp(l.starttid)) OVER (PARTITION BY l.id ORDER BY  l.starttid)  >= 30 * 60
      THEN 1
      ELSE 0
      END as new_session
      from loggik_df l
  ) s1
      """

  sql4df = sqlHContext.sql(sqlloggik4_df)
```

SQL Error messages

Spark 1.6 => an exception happen at this location X

```
> val sqlq = """
             SELECT id,cykle,avg(value) OVERT (PARTITION BY id ORDER BY cykle ASC ROWS BETWEEN 2 PRECEDING AND 2 FOLLOWING)
             FROM dft
  """

  val dfSQL = sqlContext.sql(sqlq)
```

```
org.apache.spark.sql.AnalysisException: missing EOF at '(' near 'OVERT'; line 1 pos 33
        at org.apache.spark.sql.hive.HiveQl$.createPlan(HiveQl.scala:318)
        at org.apache.spark.sql.hive.ExtendedHiveQlParser$$anonfun$hiveQl$1.apply(ExtendedHiveQlParser.scala:41)
        at org.apache.spark.sql.hive.ExtendedHiveQlParser$$anonfun$hiveQl$1.apply(ExtendedHiveQlParser.scala:40)
        at scala.util.parsing.combinator.Parsers$Success.map(Parsers.scala:136)
        at scala.util.parsing.combinator.Parsers$Success.map(Parsers.scala:135)
```

Spark 2.0 => illustrate possible options

```
> val sqlq = """
             SELECT id,cykle,avg(value) OVERT (PARTITION BY id ORDER BY cykle ASC ROWS BETWEEN 2 PRECEDING AND 2 FOLLOWING)
             FROM dft
  """

  val dfSQL = sqlContext.sql(sqlq)
```

```
org.apache.spark.sql.catalyst.parser.ParseException:
extraneous input '(' expecting {<EOF>, ',', 'FROM', 'WHERE', 'GROUP', 'ORDER', 'HAVING', 'LIMIT', 'LATERAL', 'WINDOW', 'UNION', 'EXCEPT', 'INTERSEC
T', 'SORT', 'CLUSTER', 'DISTRIBUTE'}(line 2, pos 45)

== SQL ==

             SELECT id,cykle,avg(value) OVERT (PARTITION BY id ORDER BY cykle ASC ROWS BETWEEN 2 PRECEDING AND 2 FOLLOWING)
---------------------------------------------^^^
             FROM dft
```

Built-in CSV file support

Spark 1.6 => require package databricks/spark-csv

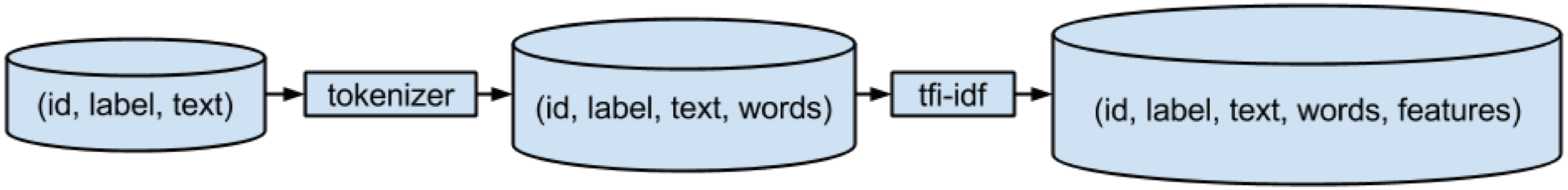
```
> val df = sqlContext.read.format("com.databricks.spark.csv").option("header", "true")
  .load("s3n://com-combient-test/ml-meetup/The_world_population_data.csv")
```

Spark 2.0 => built-in

```
> val df = spark.read.option("header", "true")
              .csv("s3n://com-combient-test/ml-meetup/The_world_population_data.csv")
```

Spark 2.0 Machine learning (ML) APIs Status

- spark.mllib (RDD-based APIs)
  - Entering bug-fixing modes (no new features)
- spark.ml (Dataframe-based): models + transformations
  - Introduced in January 2015
  - Become a primary API



An example Pipeline model

ML Pipeline Persistence Across languages

Why?

Data scientist

Propose →

ML Pipelines

Prefer Python / R / ..

Source: classroomclipart.com

- Prototype and experiments with candidate ML pipelines
- Focus on rapid prototype and fast exploration of data

Data engineer

← Implement / deploy

Prefer Java / Scala

Source: worldartsme.com/

- Implement and deploy pipelines in production environment
- Focus on maintainable and reliable systems

Pipeline persistence allow for **import/export** pipeline across languages

- Introduced only for Scala in Spark 1.6
- Status of Spark 2.0
  - Scala/Java (Full), Python (Near-full), R (with hacks)

Create ML pipeline and persist it in Python

```
%py
# Building a pipeline
from pyspark.ml import Pipeline
from pyspark.ml.feature import StandardScaler
from pyspark.ml.classification import RandomForestClassifier
scaler = StandardScaler(inputCol="features", outputCol="scaledFeatures",
                        withStd=True, withMean=False)
scaler_model = scaler.fit(training_df)
dataset_scaled = scaler_model.transform(training_df)
rf_classifier = RandomForestClassifier(featuresCol="scaledFeatures",labelCol="label",
                                       predictionCol="prediction",probabilityCol="probability",
                                       maxDepth=5,numTrees=100)
ml_pipeline =  Pipeline(stages=[scaler_model, rf_classifier])

# Fitting the pipeline model to the training set
pipeline_model = ml_pipeline.fit(training_df)

# Writing the pipeline to a file
base_path = "/tmp/mlpipe-persistence-example"
pipeline_model.save(base_path + "/fitted_pipeline")
```

Loading the pipeline and test it in Scala

```scala
import org.apache.spark.ml._
val base_path = "/tmp/mlpipe-persistence-example"
// Loading the pipeline
val pipeline = PipelineModel.read.load(base_path + "/fitted_pipeline")

// Testing the pipeline
val predictions = pipeline.transform(test_df)
predictions.show
```

```
+-----+--------------------+--------------------+--------------------+--------------------+----------+
|label|            features|      scaledFeatures|       rawPrediction|         probability|prediction|
+-----+--------------------+--------------------+--------------------+--------------------+----------+
|  7.0|(784,[202,203,204...|(784,[202,203,204...|[2.42187287483487...|[0.02421872874834...|       7.0|
|  2.0|(784,[94,95,96,97...|(784,[94,95,96,97...|[9.73366907185490...|[0.09733669071854...|       2.0|
|  1.0|(784,[128,129,130...|(784,[128,129,130...|[0.28122374573791...|[0.0028122374573 7...|       1.0|
|  0.0|(784,[124,125,126...|(784,[124,125,126...|[77.5100579522886...|[0.77510057952288...|       0.0|
|  4.0|(784,[150,151,159...|(784,[150,151,159...|[3.55187312690512...|[0.03551873126905...|       4.0|
|  1.0|(784,[156,157,158...|(784,[156,157,158...|[0.18799849144322...|[0.00187998491443...|       1.0|
|  4.0|(784,[149,150,151...|(784,[149,150,151...|[1.78116677229671...|[0.01781166772296...|       4.0|
|  9.0|(784,[179,180,181...|(784,[179,180,181...|[0.95908360963345...|[0.00959083609633...|       9.0|
|  5.0|(784,[129,130,131...|(784,[129,130,131...|[6.89697832373067...|[0.06896978323730...|       6.0|
|  9.0|(784,[209,210,211...|(784,[209,210,211...|[1.72278921501600...|[0.01722789215016...|       9.0|
|  0.0|(784,[123,124,125...|(784,[123,124,125...|[88.0199490767903...|[0.88019949076790...|       0.0|
|  6.0|(784,[94,95,96,97...|(784,[94,95,96,97...|[15.5591258414737...|[0.15559125841473...|       6.0|
|  9.0|(784,[208,209,210...|(784,[208,209,210...|[1.58963036387379...|[0.01589630363873...|       9.0|
|  0.0|(784,[152,153,154...|(784,[152,153,154...|[76.4044263510522...|[0.76404426351052...|       0.0|
|  1.0|(784,[125,126,127...|(784,[125,126,127...|[0.21724594335647...|[0.00217245943356...|       1.0|
|  5.0|(784,[124,125,126...|(784,[124,125,126...|[10.5215301036557...|[0.10521530103655...|       3.0|
|  9.0|(784,[179,180,181...|(784,[179,180,181...|[2.80678452853574...|[0.02806784528535...|       9.0|
|  7.0|(784,[200,201,202...|(784,[200,201,202...|[3.93447536107229...|[0.03934475361072...|       7.0|
|  3.0|(784,[118,119,120...|(784,[118,119,120...|[3.49591652293212...|[0.03495916522932...|       6.0|
|  4.0|(784,[158,159,185...|(784,[158,159,185...|[1.98901146497452...|[0.01989011464974...|       4.0|
+-----+--------------------+--------------------+--------------------+--------------------+----------+
```

Approximate DataFrame Statistics Functions

Motivation: rapid explorative data analysis

Bloom filter - Approximate set membership data structure

Burton H. Bloom. 1970. Space/time trade-offs in hash coding with allowable errors. Commun. ACM 13, 7 (July 1970), 422-426. DOI=http://dx.doi.org/10.1145/362686.362692

CountMinSketch - Approximate frequency estimation data structure

Cormode, Graham (2009). "Count-min sketch" (PDF). Encyclopedia of Database Systems. Springer. pp. 511–516.

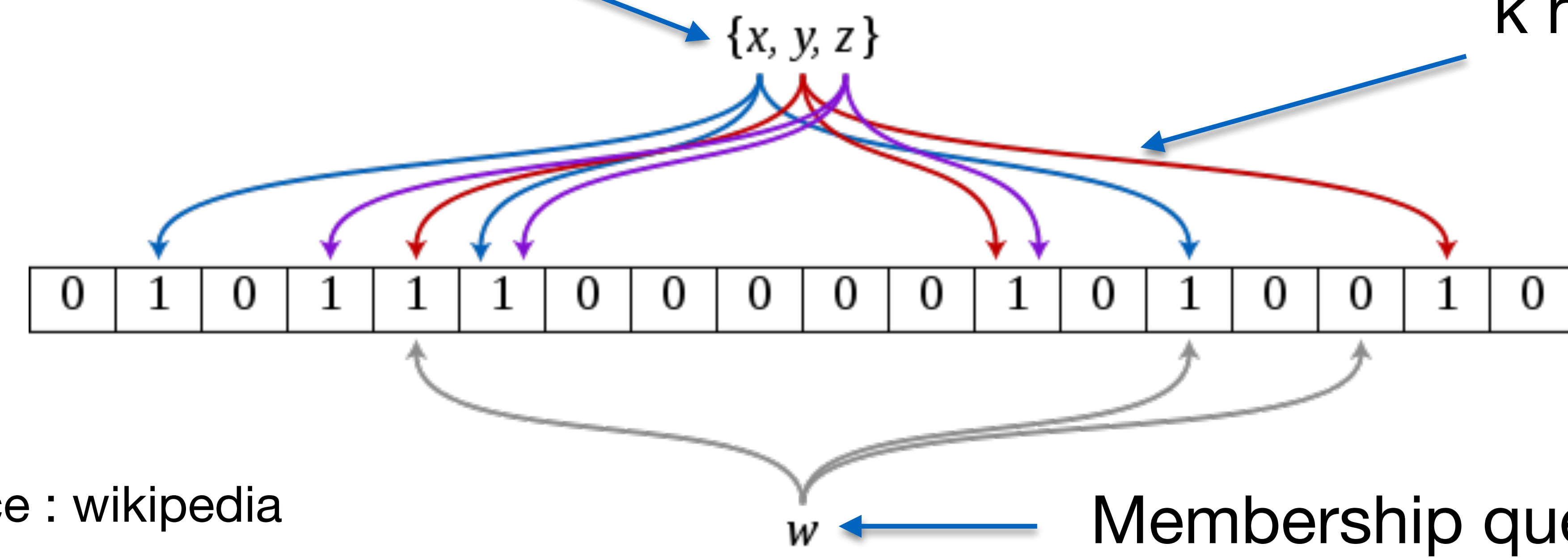
Approximate Quantile

Michael Greenwald and Sanjeev Khanna. 2001. Space-efficient online computation of quantile summaries. In Proceedings of the 2001 ACM SIGMOD international conference on Management of data (SIGMOD '01), Timos Sellis and Sharad Mehrotra (Eds.). ACM, New York, NY, USA, 58-66.

Bloom filter- Approximate set membership data structure

Inserting items O(k)

k hash functions



Source : wikipedia

Membership query => O(k)

Trade space efficiency with false positives

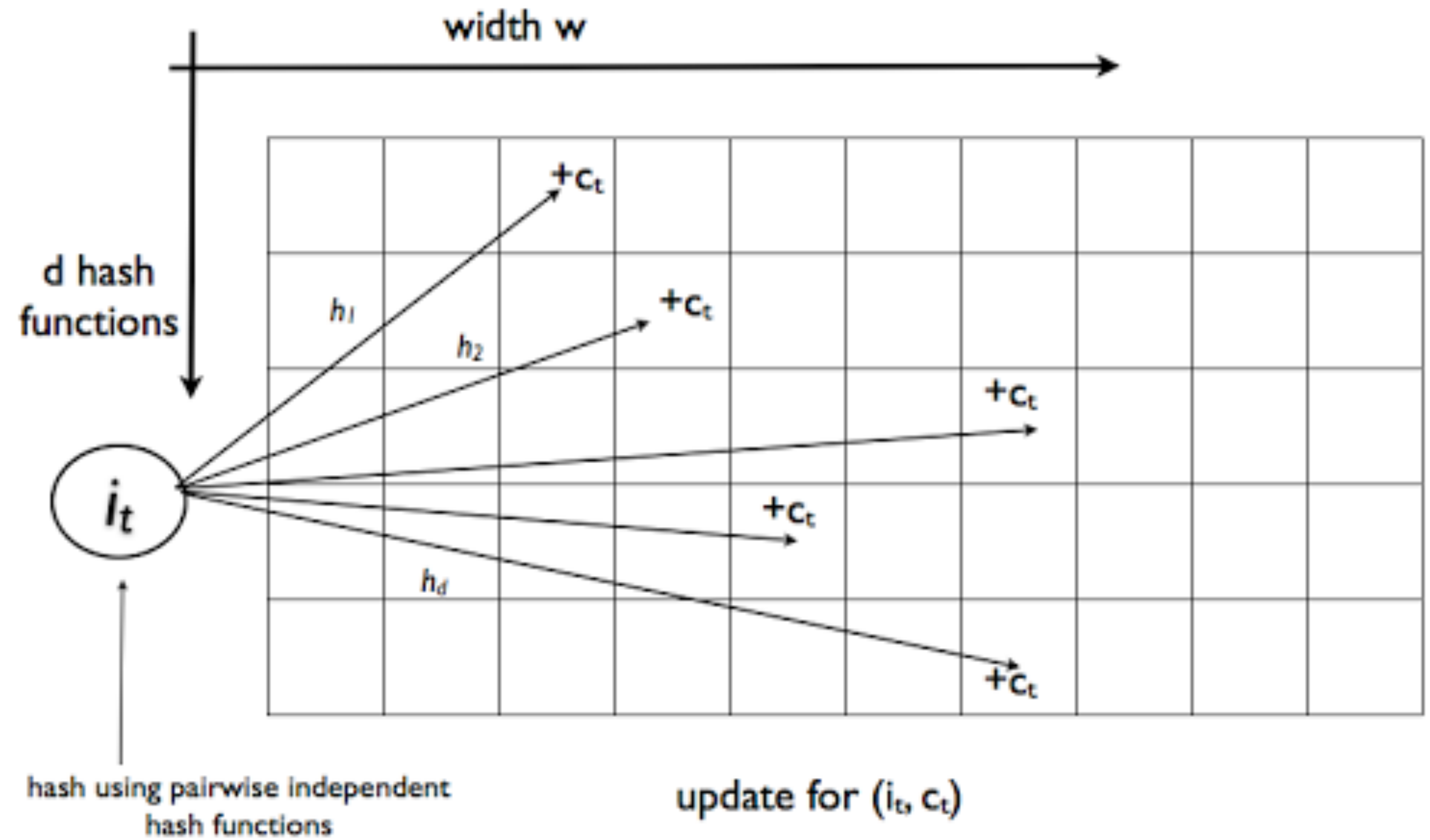
Spark 2: example code

```
> val bfCountryName = df.stat.bloomFilter(colName="name", expectedNumItems=300, fpp =0.01)
  println(bfCountryName.mightContain("Sweden"))
  println(bfCountryName.mightContain("Unknow Country"))
```

```
true
false
```

Count-Min Sketch

Operations
- Add item - O(wd)
- Query item frequency - O(wd)



Source : http://debasishg.blogspot.se/2014/01/count-min-sketch-data-structure-for.html

Spark 2: example code

```
> val nameSketch = df.stat.countMinSketch(colName="name", eps = 0.001,
                                          confidence = 0.99, seed = 42)
  println(nameSketch.estimateCount("Sweden"))
  println(nameSketch.estimateCount("Unknown Country"))
```

```
1
0
```

Approximate Quantile

Details are much more complicated than the above two. :)

Spark 2: example code

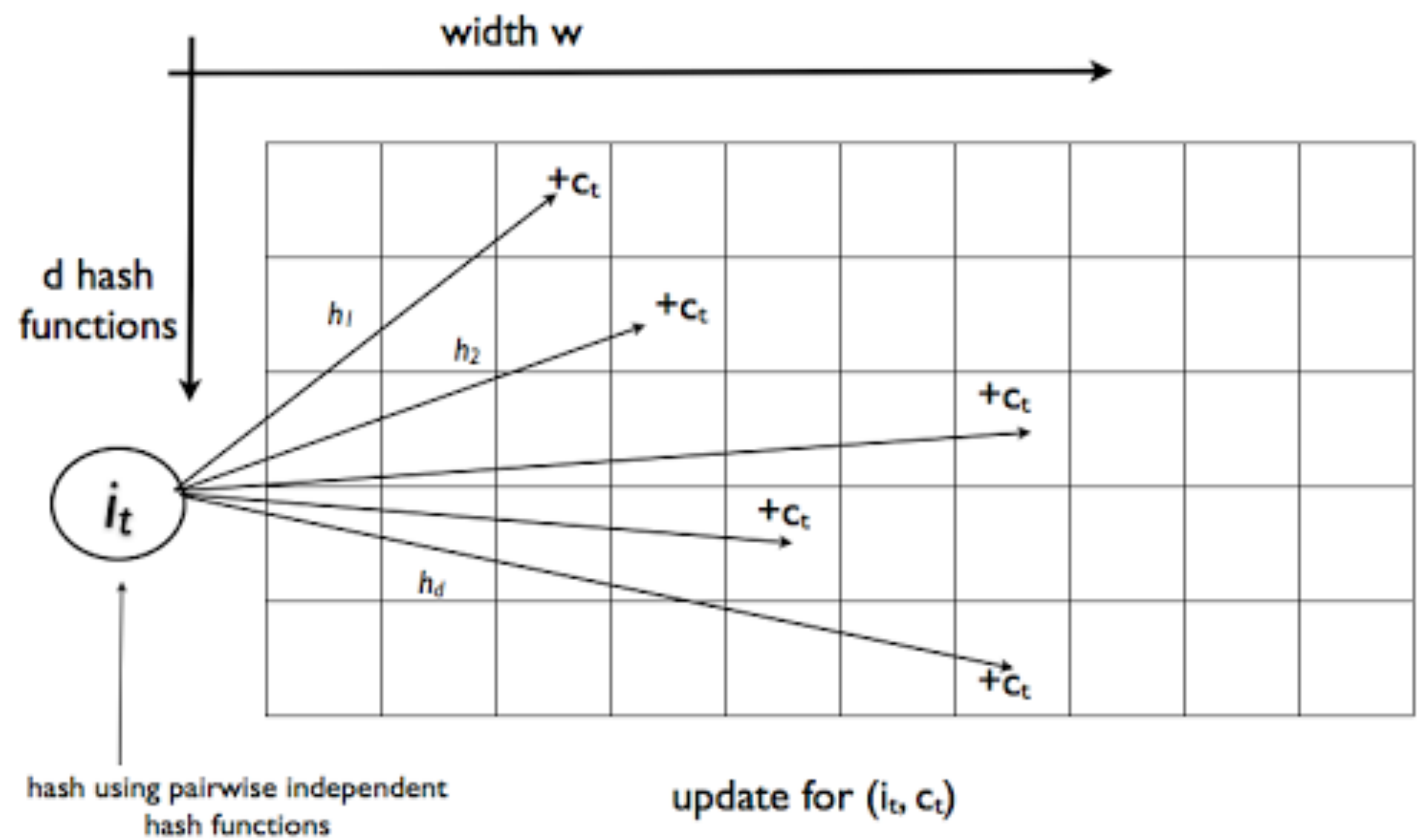
```
> df.stat.approxQuantile(
          col="population_2014",
          probabilities=Array(0.25, 0.5, 0.75, 0.9),
          relativeError=0.05)
```

```
res189: Array[Double] = Array(2219937.0, 1.0598482E7, 1.9113728E7, 2.6786598E7)
```

Count-Min Sketch

Operations

- Add item - O(wd)
- Query item frequency - O(wd)



Source : http://debasishg.blogspot.se/2014/01/count-min-sketch-data-structure-for.html

Spark 2: example code

```
> val nameSketch = df.stat.countMinSketch(colName="name", eps = 0.001,
                                          confidence = 0.99, seed = 42)
  println(nameSketch.estimateCount("Sweden"))
  println(nameSketch.estimateCount("Unknown Country"))
```

```
1
0
```

Approximate Quantile

Details are much more complicated than the above two. :)

Spark 2: example code

```
> df.stat.approxQuantile(
          col="population_2014",
          probabilities=Array(0.25, 0.5, 0.75, 0.9),
          relativeError=0.05)
```

res8: Array[Double] = Array(167543.0, 1260934.0, 3753121.0, 5643475.0)

Databrick example notebook:
https://docs.cloud.databricks.com/docs/latest/sample_applications/04%20Apache%20Spark%202.0%20Examples/05%20Approximate%20Quantile.html

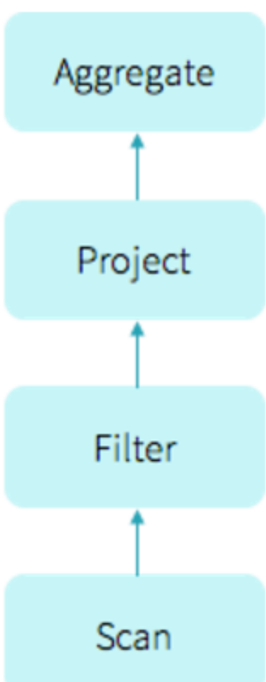
Whole-stage code generation

- Collapsing multiple function calls into one
  - Minimising overheads (e.g., virtual function calls) and using CPU registers

Source: databricks

Example

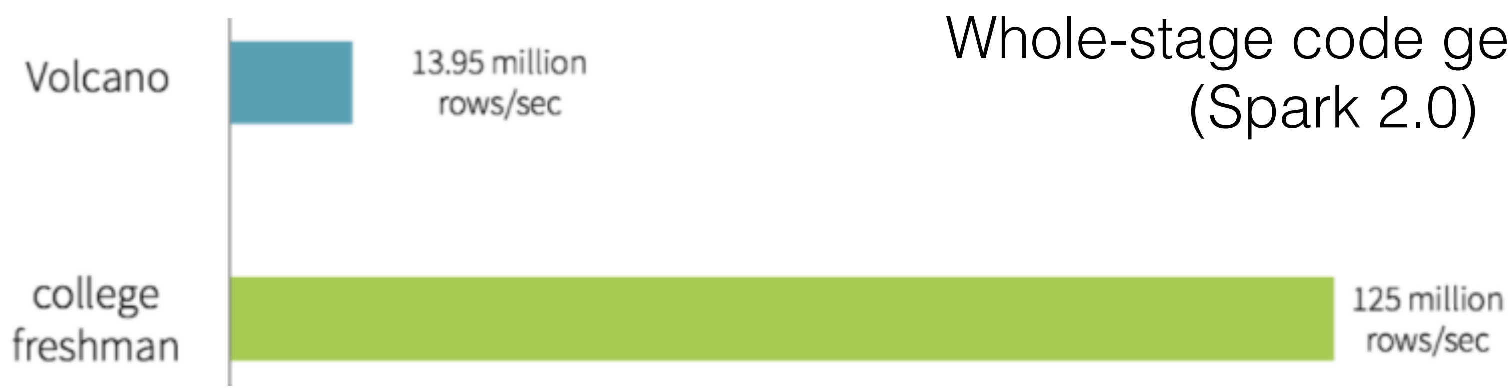
```
select count(*) from store_sales
where ss_item_sk = 1000
```



Volcano (Spark 1.X)

```
var count = 0
for (ss_item_sk in store_sales) {
  if (ss_item_sk == 1000) {
    count += 1
  }
}
```

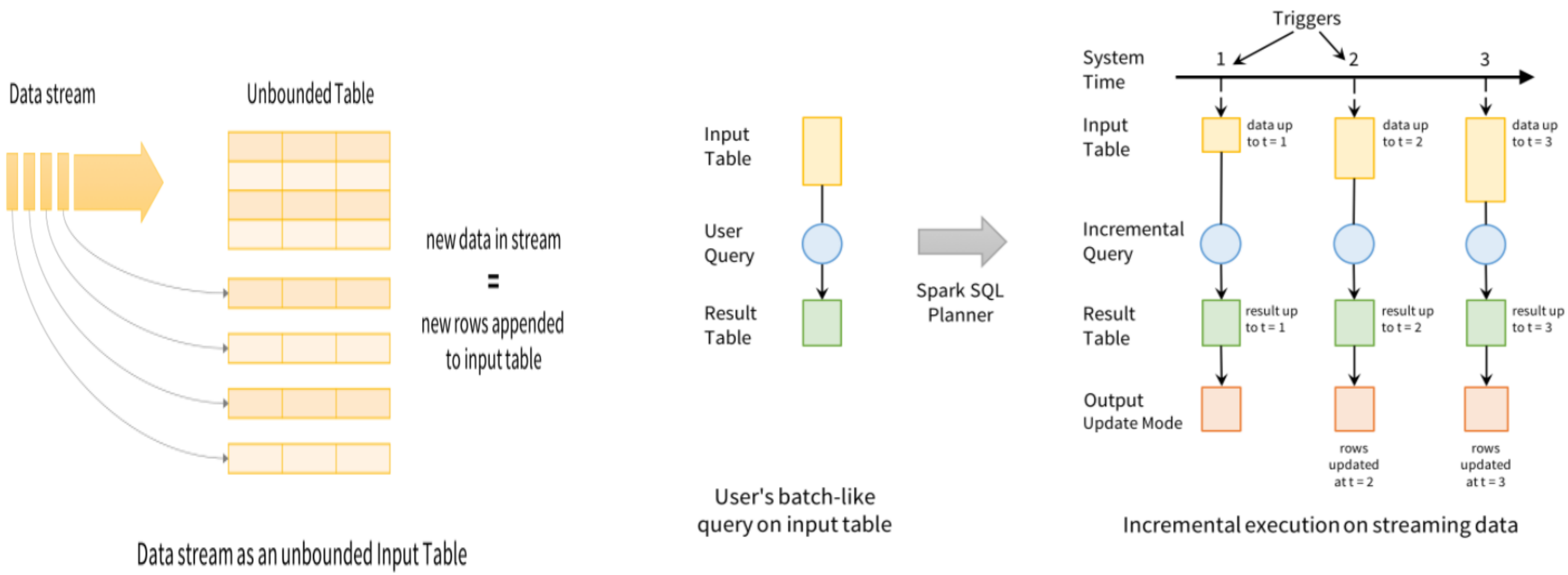
Whole-stage code generation
(Spark 2.0)



Databrick example notebook:
https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/6122906529858466/293651311471490/5382278320999420/latest.html

Structure Streaming [Alpha]

- SQL on streaming data



Source : https://databricks.com/blog/2016/07/28/structured-streaming-in-apache-spark.html

Databrick example notebook:
https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/4012078893478893/295447656425301/5985939988045659/latest.html

Structure Streaming [Alpha]

Batch Version

```
// Read JSON once from S3
logsDF = spark.read.json("s3://logs")

// Transform with DataFrame API and save
logsDF.select("user", "url", "date")
      .write.parquet("s3://out")
```

Streaming Version

Not ready for production

```
// Read JSON continuously from S3
logsDF = spark.readStream.json("s3://logs")

// Transform with DataFrame API and save
logsDF.select("user", "url", "date")
      .writeStream.parquet("s3://out")
      .start()
```

Source: https://databricks.com/blog/2016/07/28/continuous-applications-evolving-streaming-in-apache-spark-2-0.html

COMBIENT

We are currently recruiting 🙂

combient.com/#careers