Real-Life Use-Case Examples for Apache Spark Libraries

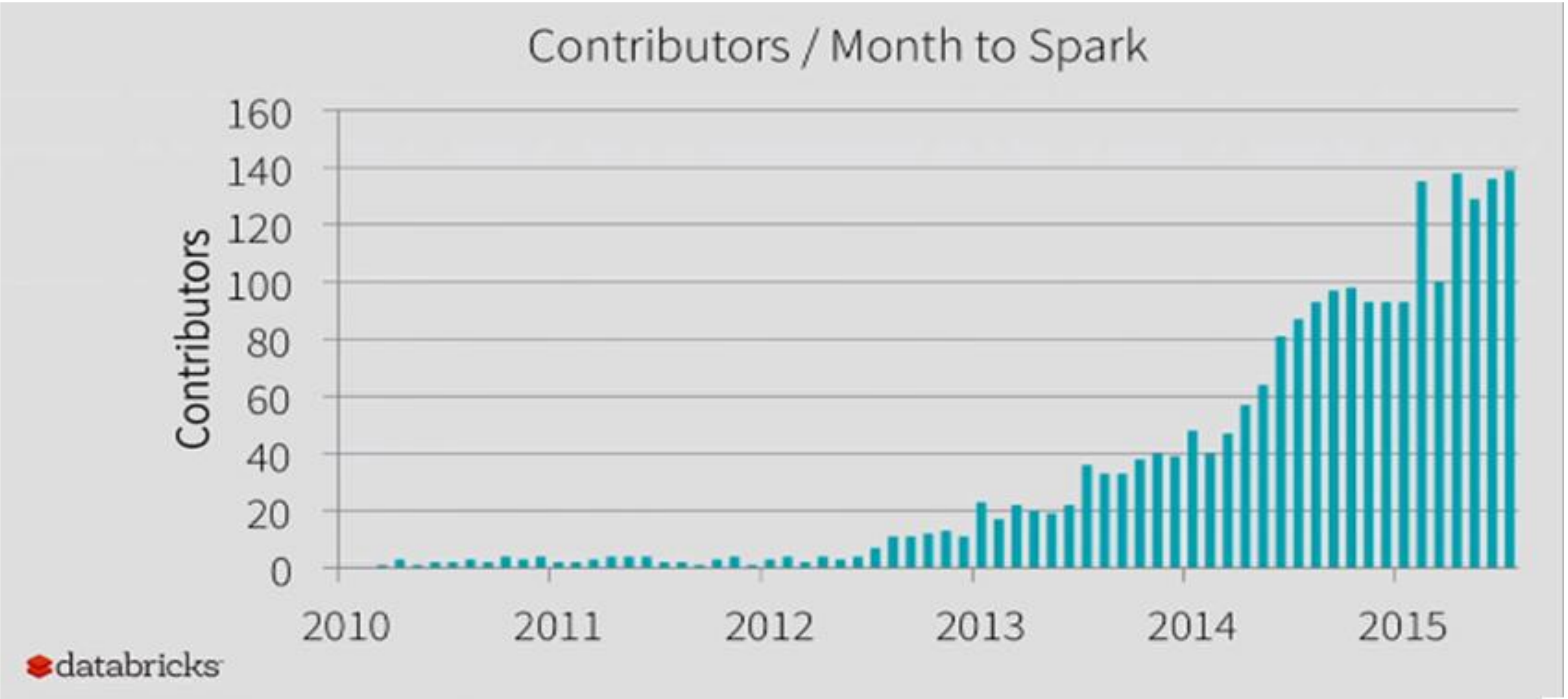
Sepideh Seifzadeh
Frantisek Mantlik
Gordon Gibson
Tri Nguyen

Jan 27th, 2016

From MapReduce to Apache Spark

- Hadoop: Distributed data infrastructure
  - Storage component: Hadoop Distributed File System (HDFS)
  - Processing component: MapReduce
- MapReduce for batch processing
- Most ML algorithms run iteratively
- Apache Spark: General framework for processing and analysis of big data

Community Growth

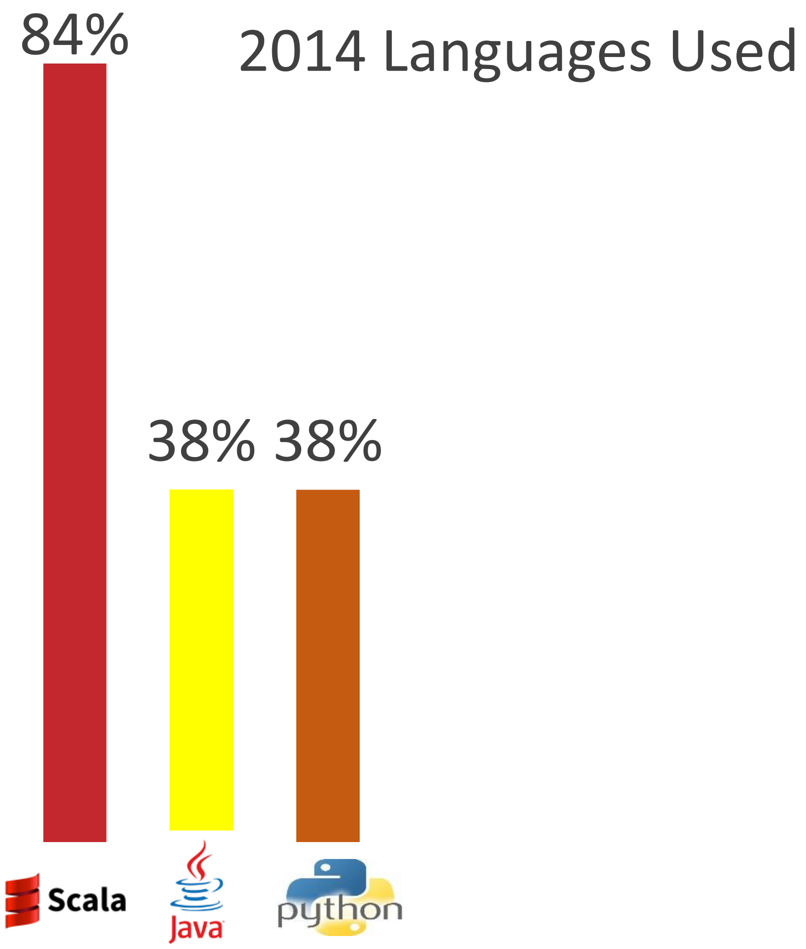


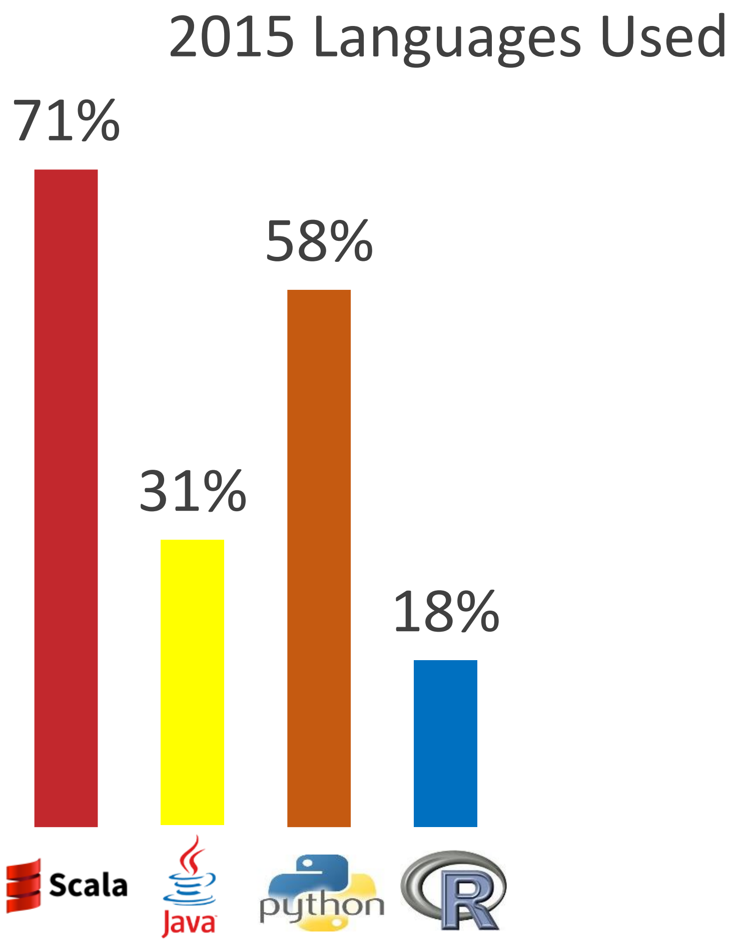
Spark Users

YAHOO!
amazon web services
Adobe
IBM
intel
verizon
redhat
ORACLE
SAP
ebay
Goldman Sachs
NBCUniversal
阿里巴巴 Alibaba.com
NETFLIX
NTT DaTa
TOYOTA
CISCO
Telefónica
DATASTAX
databricks
Hortonworks
cloudera
MAPR

databricks

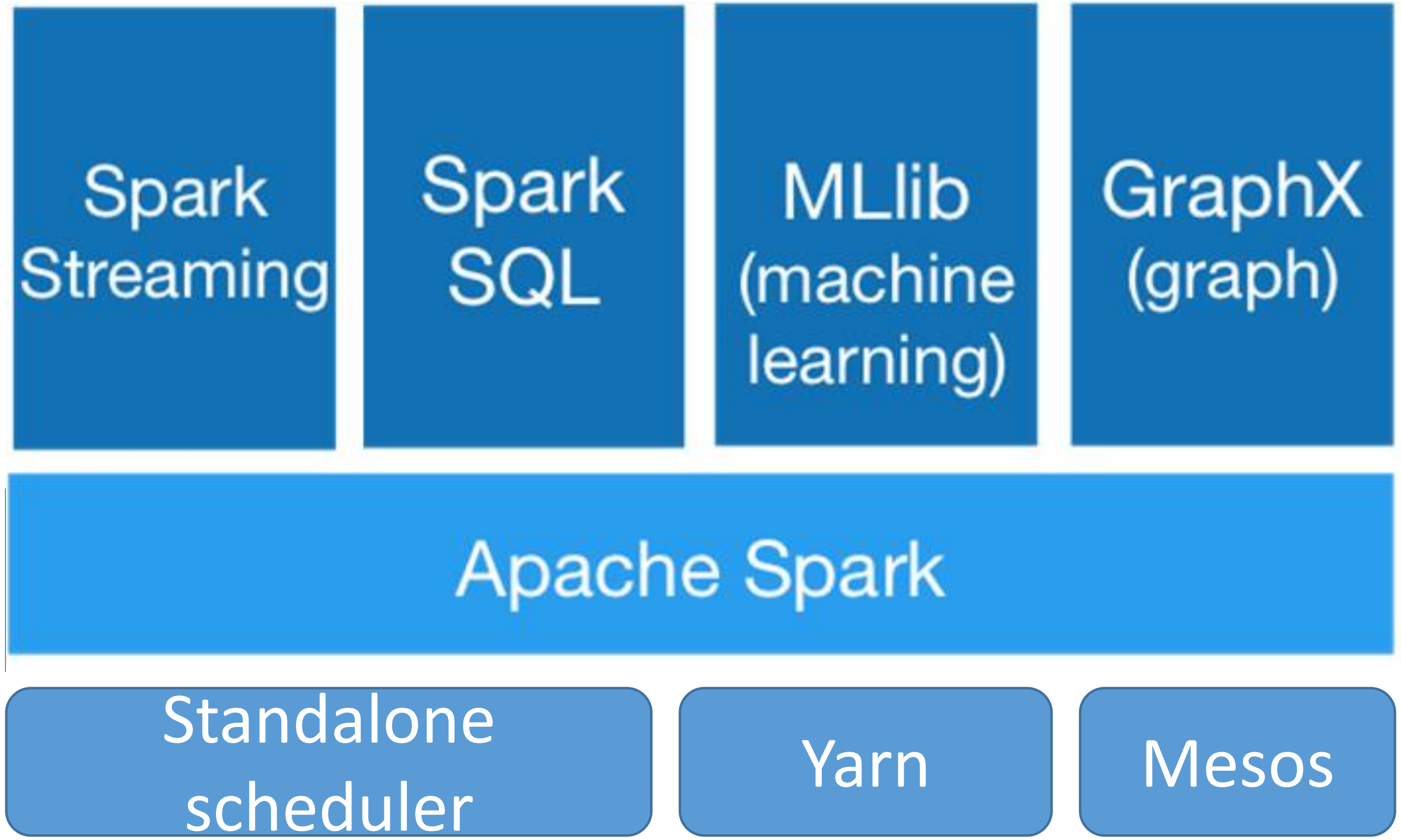
Programming Languages



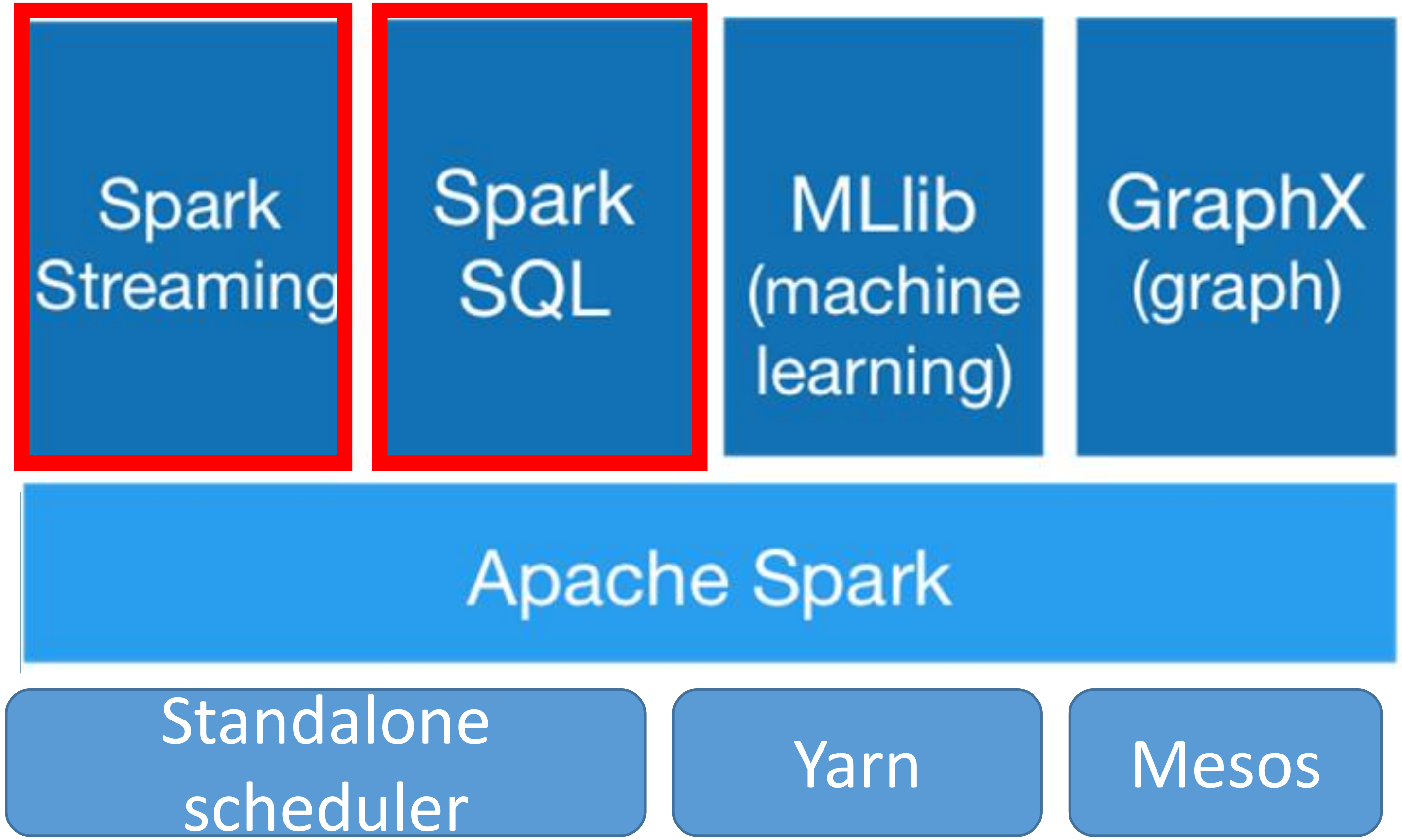


databricks

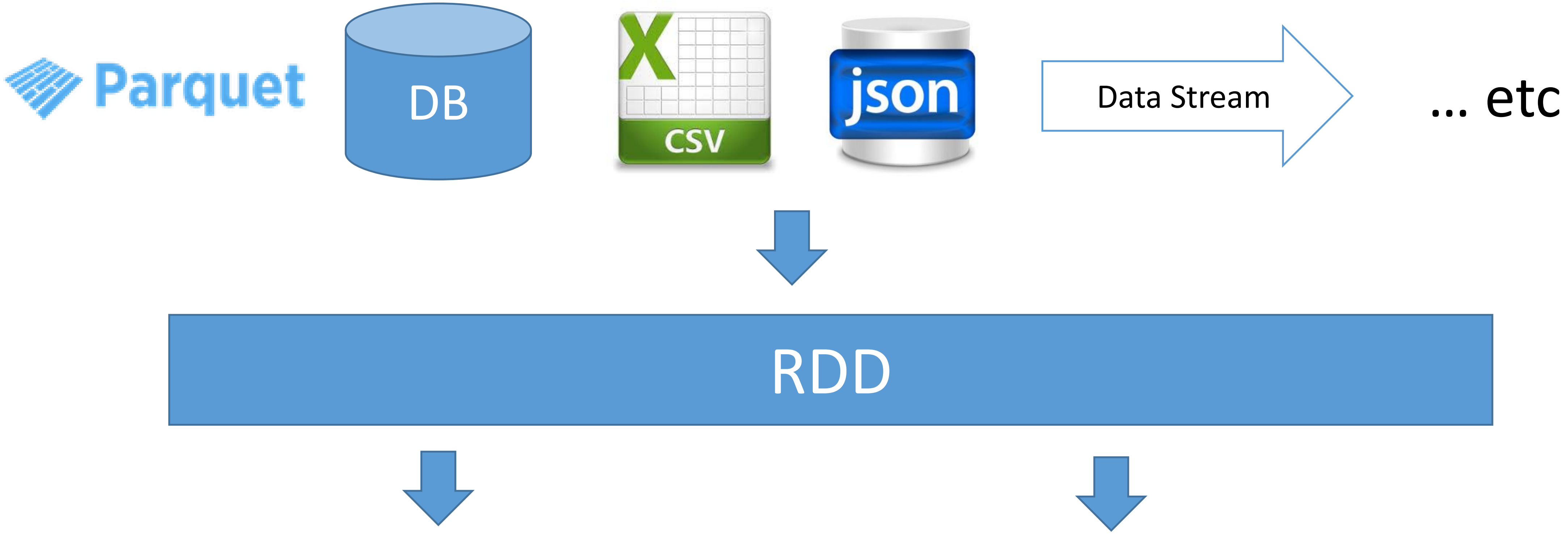
Spark Architecture



Spark Architecture



RDD: Resilient Distributed Dataset

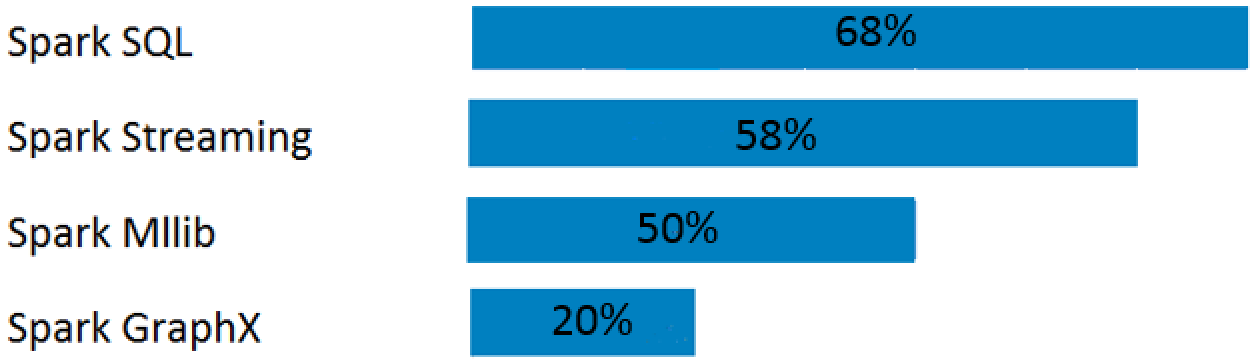


Transformation: Uppercase, Split words

Action: count, saveAsTextFile

Apache Spark Libraries

Which libraries do people use?



75% of users use more than one component

Spark Streaming

Ingest and manipulate real-time data streams

Process data in near real-time

Spark SQL

Use high level SQL language to query data

- Structured data
- Semi-structured data

Compatible with various file formats and data stores

Compatible with Hive

- Can read and write to Hive tables including ORC, Parquet, and SequenceFiles

DataFrames

Distributed collections of data organized into named columns

Equivalent to tables in relational databases

Spark MLlib

MLlib is Spark’s Machine Learning library

It has the implementation of scalable Machine Learning algorithms

Good for integration with Spark SQL, Spark Streaming and Spark GraphX

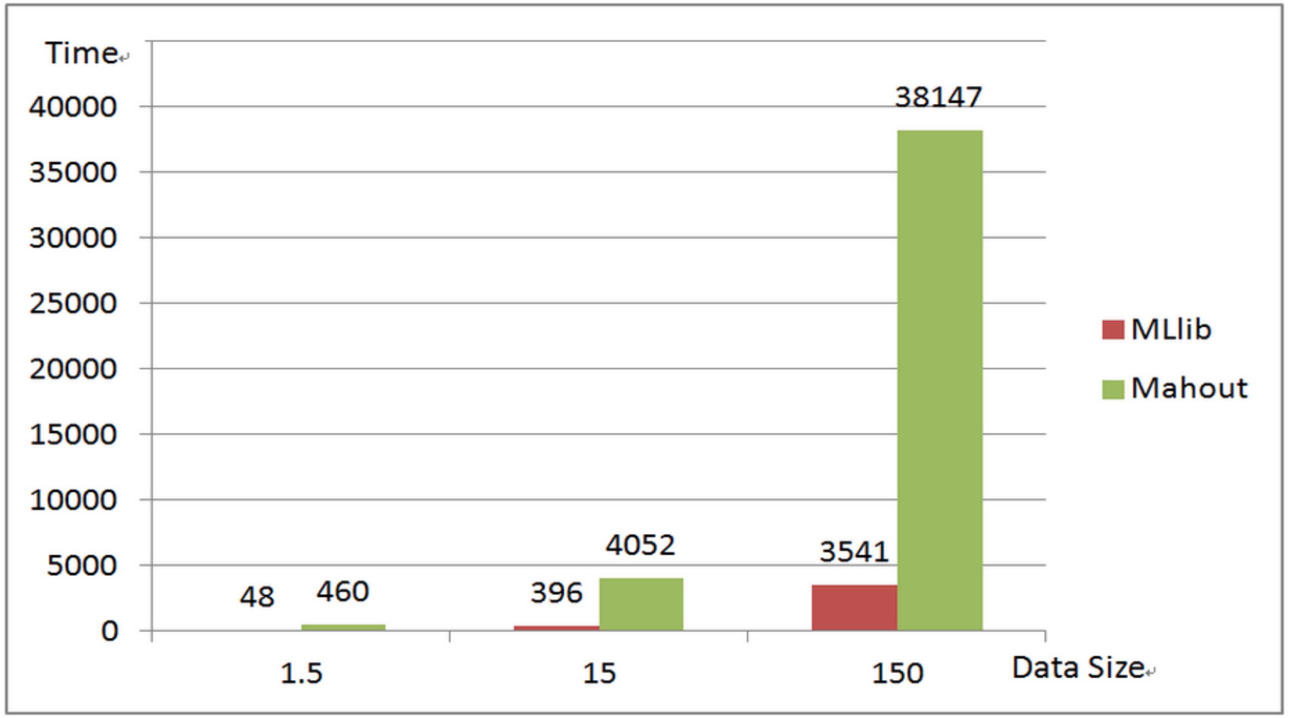
- Spark SQL + MLlib → used to easily extract and analyze data from existing Apache Hive
- Spark Streaming + MLlib → used for analyzing streaming data in near-real-time

MLlib

Spark MLlib

Spark MLlib is the replacement for Mahout

- Machine Learning algorithms implemented based on Map-Reduce concept



databricks

Running times for different data sizes (GB)

Spark MLlib

- Available algorithms on MLlib
  - Classification: logistic regression, linear SVM, naïve Bayes, least squares, classification tree
  - Regression: generalized linear models (GLMs), regression tree
  - Collaborative filtering: alternating least squares (ALS), non-negative matrix factorization (NMF)
  - Clustering: k-means
  - Decomposition: SVD, PCA
  - Optimization: stochastic gradient descent, L-BFGS

Spark MLlib

Applications:

- Classification of products, prediction of users' behaviour or interest
- Grouping similar entities, e.g. Twitter dataset for topic detection
- Recommender systems: make the appropriate personalized recommendation for different customers

Spark GraphX

A new component for Graph-parallel computation

Advantages of using Spark GraphX:

- Growing set of algorithms and builders
- Parallel processing of graph operations
- Full integration with Spark RDD and DataFrame API

Application:

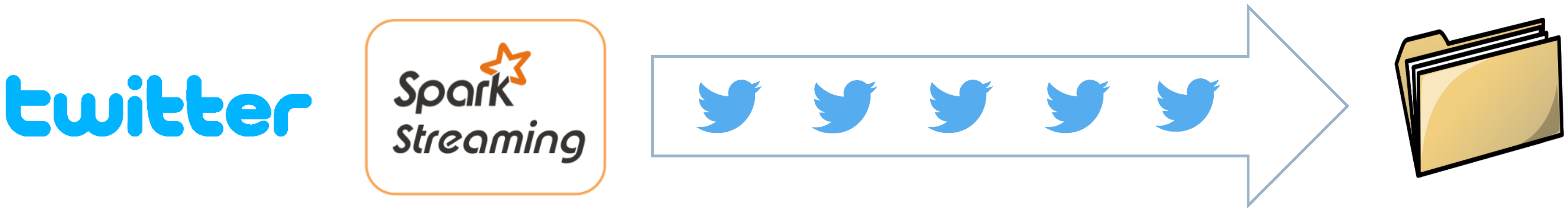
- Page Rank (importance of a node in a Graph e.g. Google Search)
- Shortest Distance

Spark Streaming

Section Topics

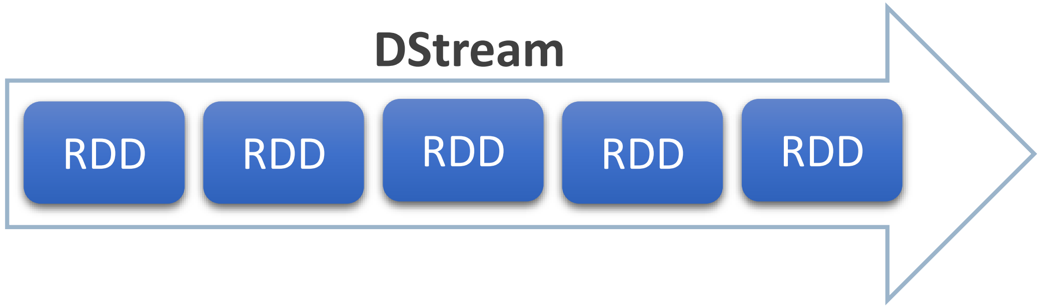
- DStreams
- Transformations
- Window Operations
- Output Operations

Twitter Streaming Example

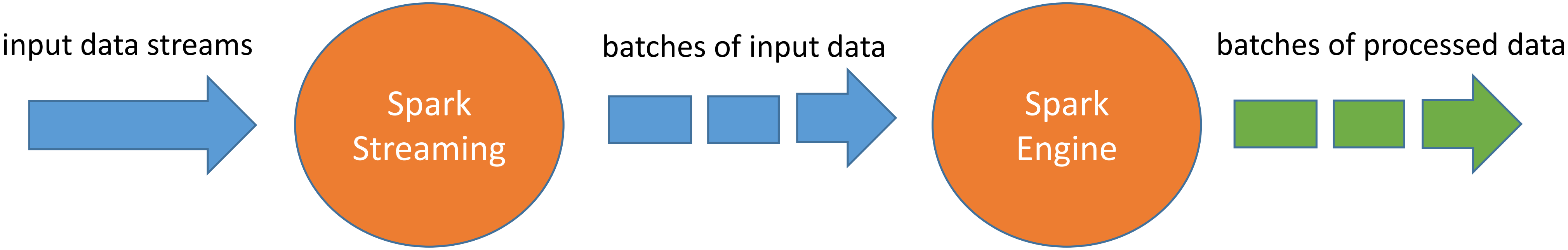


1. Count tweets
2. Track most frequent words
3. Save tweets to file

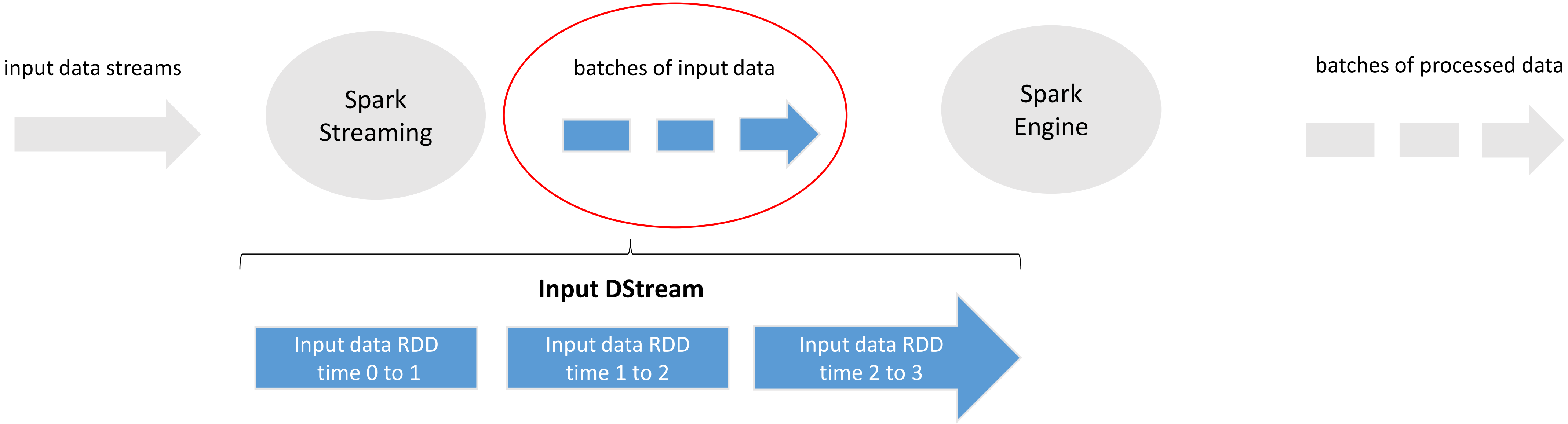
DStream – Discretized Stream



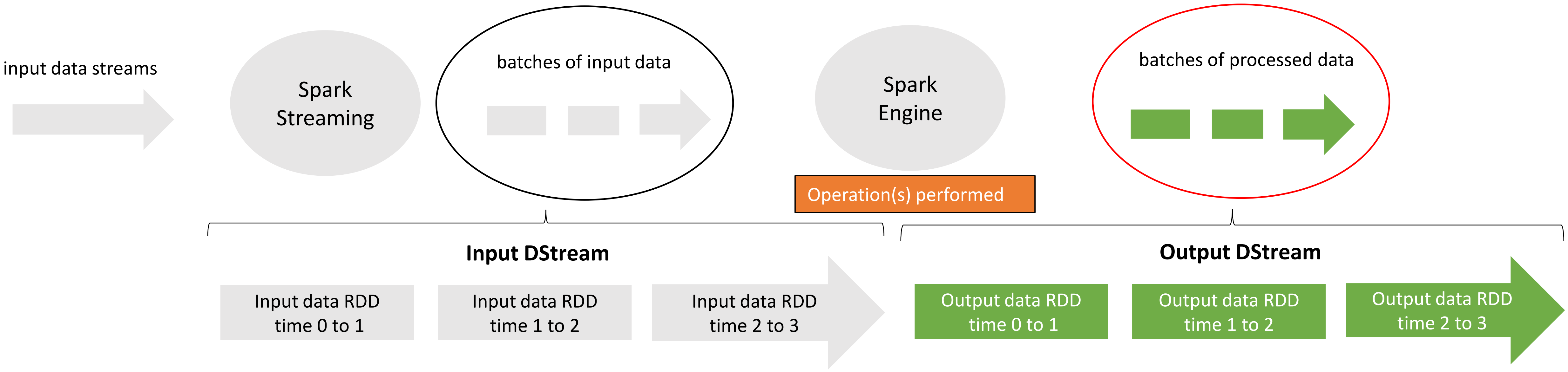
DStreams


input data streams
Spark Streaming
batches of input data
Spark Engine
batches of processed data

DStreams


input data streams
Spark Streaming
batches of input data
Spark Engine
batches of processed data
Input DStream
Input data RDD time 0 to 1
Input data RDD time 1 to 2
Input data RDD time 2 to 3

DStreams


input data streams
Spark Streaming
batches of input data
Spark Engine
batches of processed data
Operation(s) performed
Input DStream
Output DStream
Input data RDD time 0 to 1
Input data RDD time 1 to 2
Input data RDD time 2 to 3
Output data RDD time 0 to 1
Output data RDD time 1 to 2
Output data RDD time 2 to 3

Input DStreams

FLUME

kafka

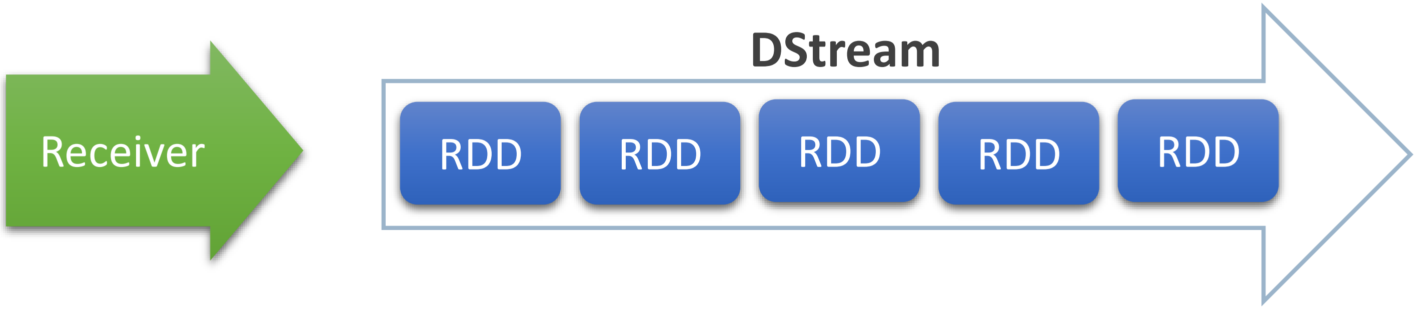
Amazon Kinesis

twitter

ØMQ

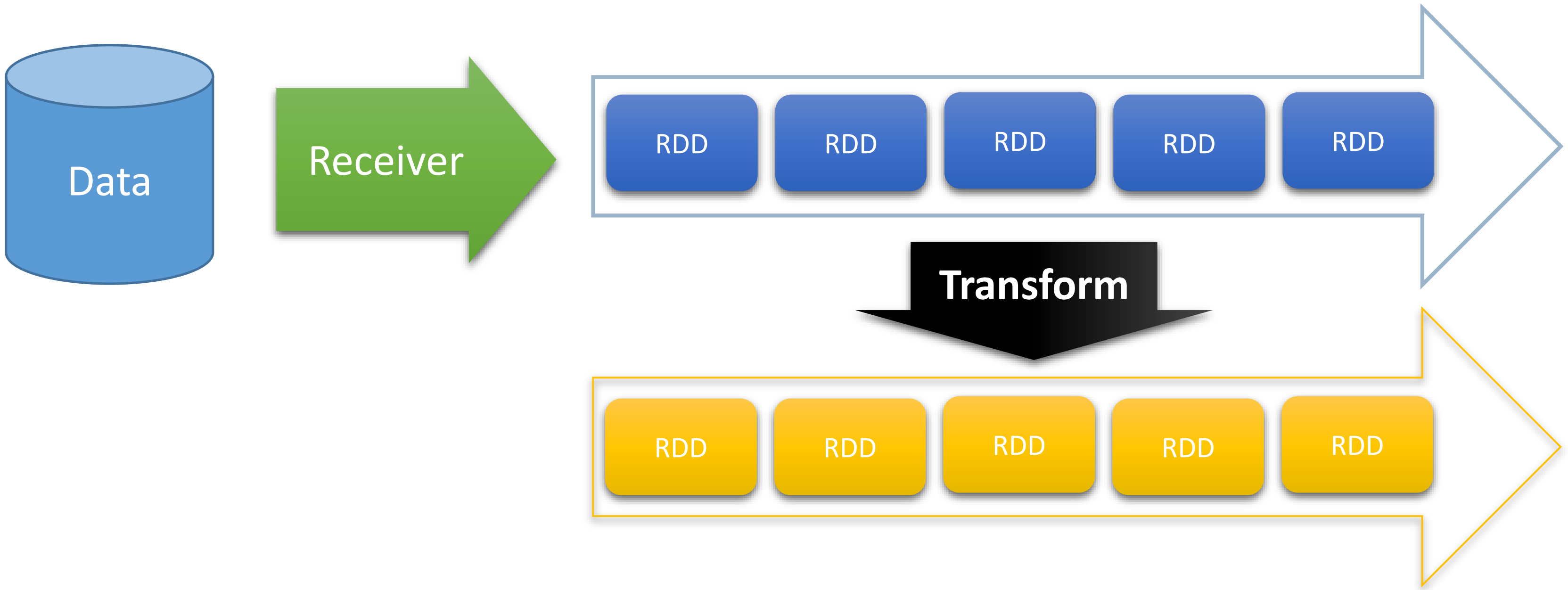
MQTT.ORG

etc

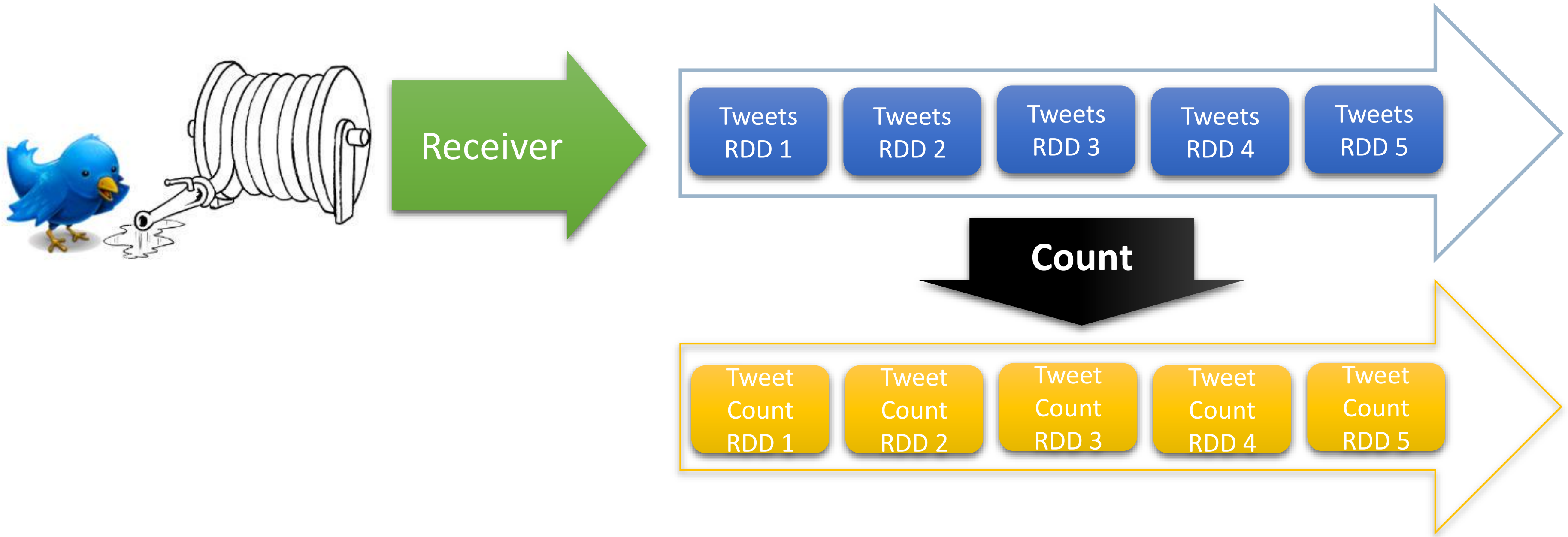


Twitter Firehose

Transformations



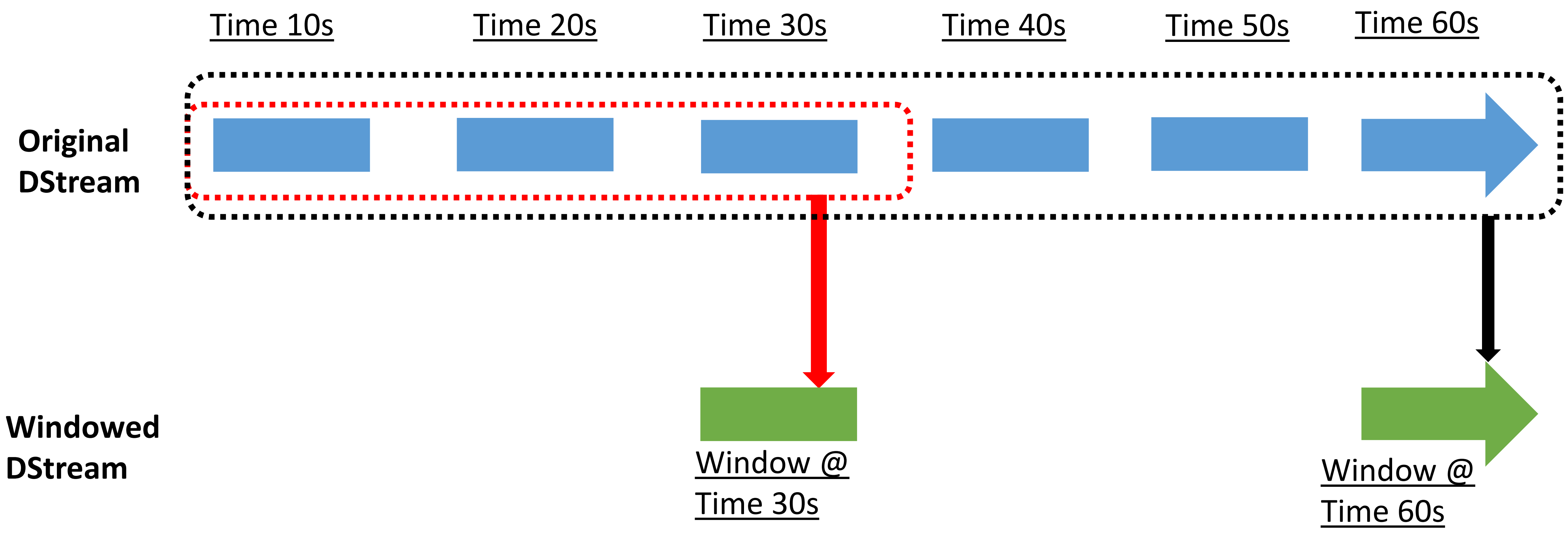
Transformations



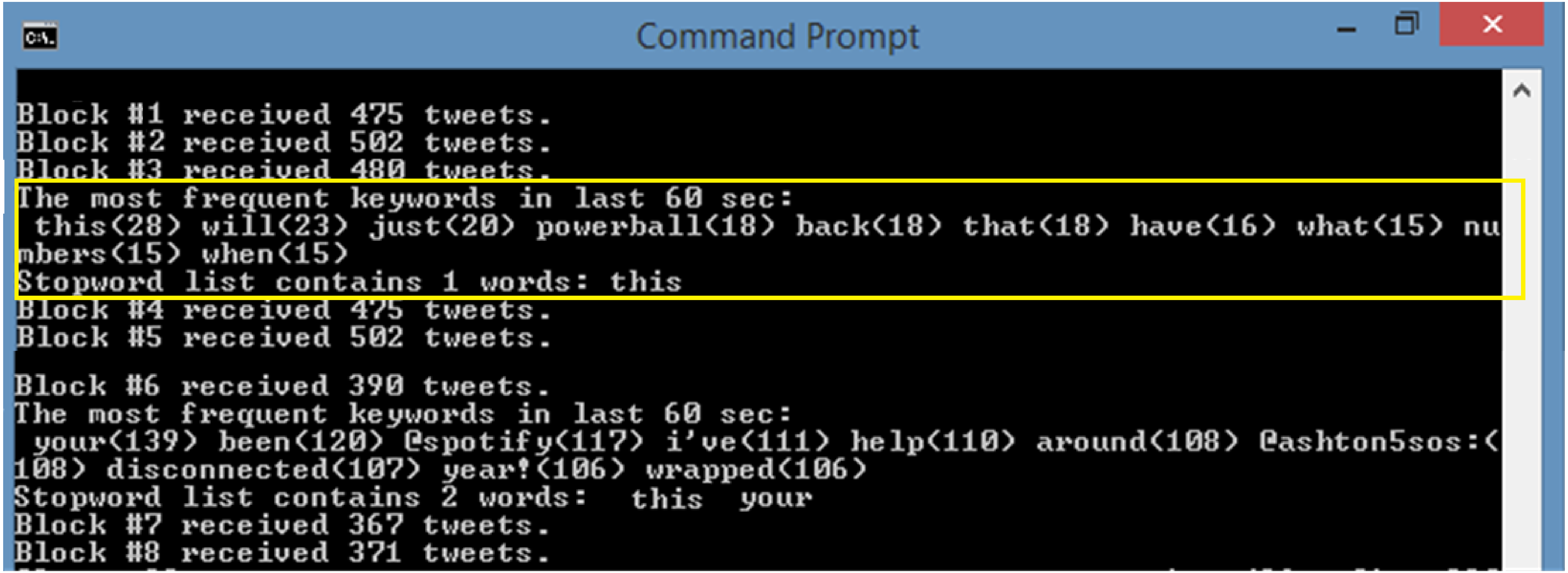
Tweet Count Results

```
spark-submit  --class academy.adastra.streamingexample.TwitterStream Document...
.TwitterStream Documents\NetBeansProjects\StreamingExample\target\StreamingExamp
le-1.0-SNAPSHOT.jar
15/08/11 16:18:04 WARN NativeCodeLoader: Unable to load native-hadoop library fo
r your platform... using builtin-java classes where applicable
Block #1 received 0 tweets.
Block #2 received 373 tweets.
Block #3 received 427 tweets.
Block #4 received 450 tweets.
Block #5 received 429 tweets.
Block #6 received 405 tweets.
Block #7 received 404 tweets.
Block #8 received 427 tweets.
Block #9 received 471 tweets.
Block #10 received 439 tweets.
Block #11 received 420 tweets.
Block #12 received 396 tweets.
Block #13 received 425 tweets.
Block #14 received 430 tweets.
Block #15 received 442 tweets.
Block #16 received 498 tweets.
Block #17 received 426 tweets.
Block #18 received 471 tweets.
Block #19 received 437 tweets.
Block #20 received 447 tweets.
```

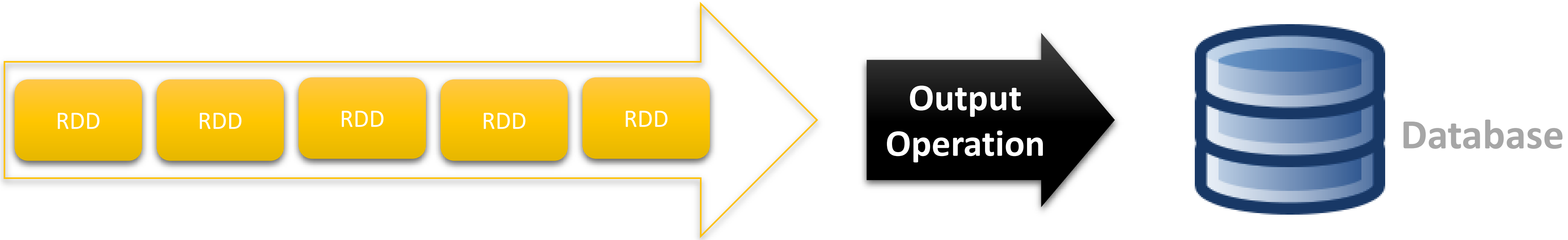
Window Operations



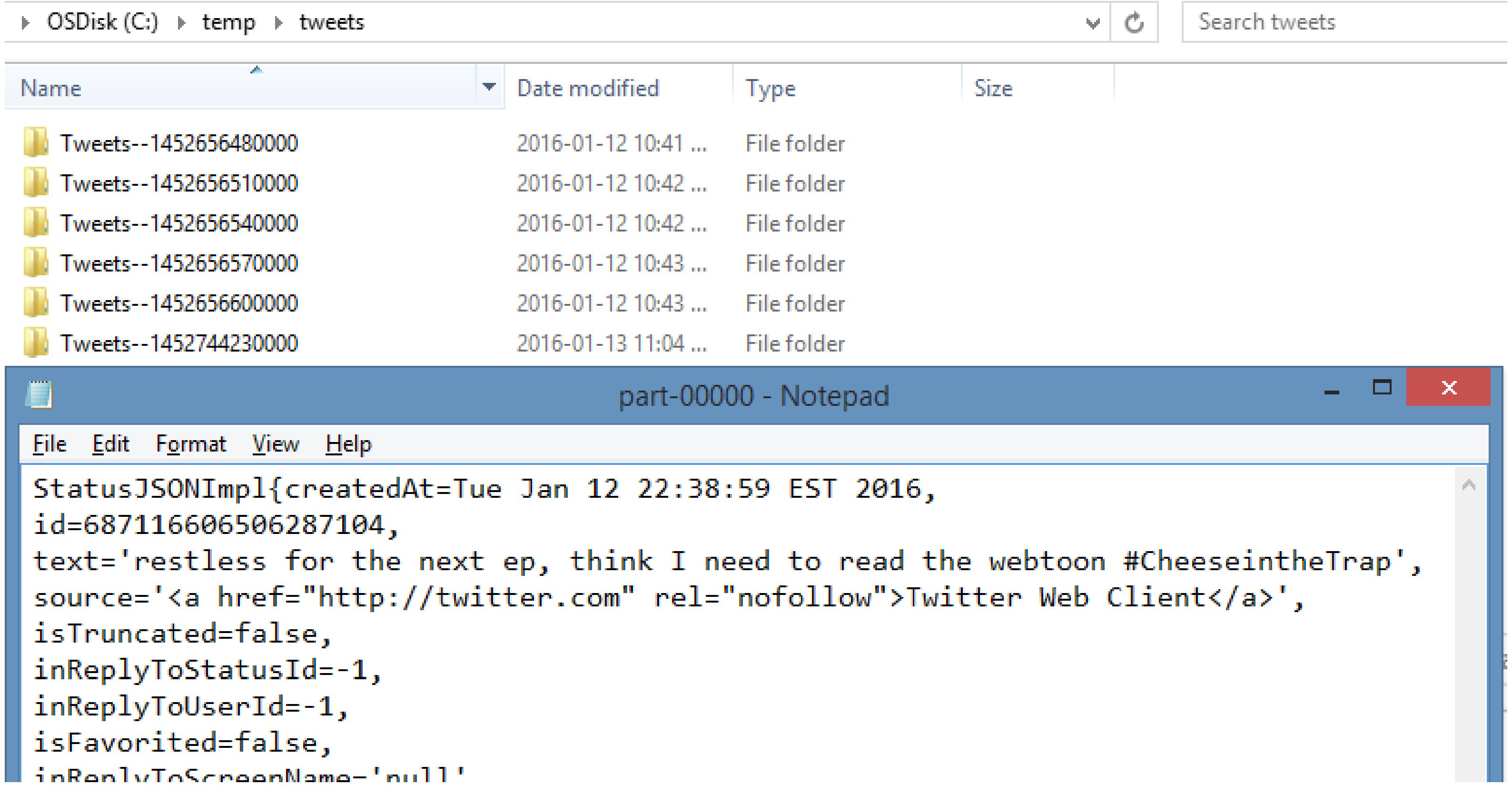
Tweet and Word Count



Output Operations



Output Data


OSDisk (C:) temp tweets
Search tweets
Name
Date modified
Type
Size
Tweets--1452656480000 2016-01-12 10:41 ... File folder
Tweets--1452656510000 2016-01-12 10:42 ... File folder
Tweets--1452656540000 2016-01-12 10:42 ... File folder
Tweets--1452656570000 2016-01-12 10:43 ... File folder
Tweets--1452656600000 2016-01-12 10:43 ... File folder
Tweets--1452744230000 2016-01-13 11:04 ... File folder
part-00000 - Notepad
File Edit Format View Help
StatusJSONImpl{createdAt=Tue Jan 12 22:38:59 EST 2016,
id=687116606506287104,
text='restless for the next ep, think I need to read the webtoon #CheeseintheTrap',
source='Twitter Web Client',
isTruncated=false,
inReplyToStatusId=-1,
inReplyToUserId=-1,
isFavorited=false,

Output Operation Results

C:\WINDOWS\system32\cmd.exe - spark-submit --class academy.adastra.streamin...

```
[Stage 49:==============================================>         (35 + 3) / 42]
[Stage 49:======================================================> (41 + 1) / 42]

Block #10 received 582 tweets.
Block #11 received 610 tweets.
Block #12 received 588 tweets.
The most frequent keywords in last 60 sec:
 #mtvhottest(45) that(43) when(32) just(31) more(28) love(27) para(26) #????????
???(23) follow(23) will(23)
Stopword list contains 4 words: this your with #mtvhottest
[Stage 69:=========>                                               (15 + 3) / 86]
[Stage 69:=============>                                           (21 + 3) / 86]
[Stage 69:================>                                        (25 + 3) / 86]
[Stage 69:====================>                                    (31 + 3) / 86]
[Stage 69:======================>                                  (34 + 3) / 86]
[Stage 69:==========================>                              (40 + 3) / 86]
[Stage 69:=============================>                           (46 + 3) / 86]
[Stage 69:=================================>                       (52 + 3) / 86]
[Stage 69:=====================================>                   (58 + 3) / 86]
[Stage 69:=========================================>               (64 + 3) / 86]
[Stage 69:=============================================>           (70 + 3) / 86]
[Stage 69:=================================================>       (76 + 3) / 86]
[Stage 69:=====================================================>   (82 + 3) / 86]
```

Spark SQL

What is SparkSQL?

- **SparkSQL is a component of Spark**
- Distributed analysis framework on **structured** and **semi-structured** data
  - Data organized in rows, columns, columns can contain embedded data structures, arrays, or maps
- Uses high level SQL language to query data
- Read/write compatible with various file formats and data stores

SQLContext

- **SQLContext** - object which supports SQL functionality in Spark
  - Created automatically when Spark Shell starts
    - Spark SQL is available out-of-the-box
    - No special configuration actions are necessary
  - Hive functionality with Spark speed and scalability
    - common tasks were reported performing ~10x faster than in native Hive running on Hadoop/YARN cluster

Where are my data?

API

hiveContext.tableNames

table.printSchema

SQL

SHOW TABLES

DESCRIBE table

Start data exploration with SQLContext

- Show tables available in Hive metastore:

Scala API:

sqlContext.**tableNames**.foreach(println)

```
C:\WINDOWS\system32\cmd.exe - spar...
scala> sqlContext.tableNames.foreach(println)
cr_card_dim
merch_cat_dim
merch_dim
transaction
tweets

scala>
```

SQL:

sqlContext.sql(**"SHOW TABLES"**).show

```
C:\WINDOWS\system32\cmd.exe - spar...
scala> sqlContext.sql("SHOW TABLES").show
+-------------+-----------+
|    tableName|isTemporary|
+-------------+-----------+
|  cr_card_dim|      false|
|merch_cat_dim|      false|
|    merch_dim|      false|
|  transaction|      false|
|       tweets|      false|
+-------------+-----------+
```

Show table structure

Scala API:

sqlContext.table("cr_card_dim").printSchema

```
scala> sqlContext.table("cr_card_dim").printSchema
root
 |-- CARD_KEY: integer (nullable = true)
 |-- TSFR_FROM_CARD_KEY: integer (nullable = true)
 |-- TSFR_TO_CARD_KEY: integer (nullable = true)
 |-- ACCT_NUM: long (nullable = true)
 |-- ACCT_SEQ_NUM: integer (nullable = true)
 |-- CR_CARD_TYP: string (nullable = true)
 |-- EXPR_DT: long (nullable = true)
 |-- CR_CARD_NM: string (nullable = true)
 |-- CARD_ISS_DT: string (nullable = true)
 |-- CUST_KEY: integer (nullable = true)
 |-- EMBOSSED_NM: string (nullable = true)
 |-- SPND_LMT_AMT: integer (nullable = true)
 |-- CSH_SPND_LMT_AMT: integer (nullable = true)


scala>
```

SQL:

sqlContext.sql("**DESCRIBE** cr_card_dim").show

```
scala> sqlContext.sql("DESCRIBE cr_card_dim").show
+------------------+---------+-------+
|          col_name|data_type|comment|
+------------------+---------+-------+
|          CARD_KEY|      int|       |
|TSFR_FROM_CARD_KEY|      int|       |
|  TSFR_TO_CARD_KEY|      int|       |
|          ACCT_NUM|   bigint|       |
|      ACCT_SEQ_NUM|      int|       |
|       CR_CARD_TYP|   string|       |
|           EXPR_DT|   bigint|       |
|        CR_CARD_NM|   string|       |
|       CARD_ISS_DT|   string|       |
|          CUST_KEY|      int|       |
|       EMBOSSED_NM|   string|       |
|      SPND_LMT_AMT|      int|       |
|  CSH_SPND_LMT_AMT|      int|       |
+------------------+---------+-------+
```

How to query data?

API	SQL
table.show	SELECT * FROM table
table.select("column"...)	SELECT column... FROM table

Query data

Scala API: sqlContext.table("merch_dim").show

SQL: sqlContext.sql("SELECT * FROM merch_dim").show

C:\WINDOWS\system32\cmd.exe - spark-shell

```
+---------+--------------------+----------------+------------+
|MERCH_KEY|            MERCH_NM|  MERCH_ACCT_NUM|MERCH_CAT_KEY|
+---------+--------------------+----------------+------------+
|        1|Francis Garza Cas...|5871773318452673|           1|
|        2|BitVision        ...|9551426714760194|           4|
|        3|LPL              ...|3832410296436075|           1|
|        4|James Williams   ...|8005714589058633|           1|
|        5|Vincent Buchanan ...|7179991882545131|           1|
|        6|Albos            ...|9258191614317116|           1|
|        7|Avartis          ...|8851720442551984|           4|
|        8|Novional         ...|9584819689622677|           1|
|        9|Artia            ...|4132201451908995|           1|
|       10|ObjectWorks      ...|8607743711864275|           1|
|       11|GJJ              ...|4212697982249185|           1|
|       12|Illumos          ...|9376864827219431|           1|
|       13|XQU Health Care  ...|8372669659144557|           1|
|       14|Escentis         ...|1378449196210869|           1|
|       15|TNF Stock Exchang...|3342709748470650|           1|
|       16|Albexis          ...|3557611478088378|           4|
|       17|Pentagon         ...|2572703956292519|           5|
|       18|Ronald Barr      ...|4253910480804024|           4|
|       19|ClickStation     ...|1851150018123793|           1|
|       20|Cepheon          ...|6965703255980256|           1|
+---------+--------------------+----------------+------------+
only showing top 20 rows
```

Query specific columns

Scala API: sqlContext.table("cr_card_dim").select("EMBOSSED_NM","SPND_LMT_AMT").show

SQL: sqlContext.sql("SELECT EMBOSSED_NM, SPND_LMT_AMT FROM cr_card_dim").show

C:\WINDOWS\system32\cmd.exe - spark-shell

```
+--------------+------------+
|   EMBOSSED_NM|SPND_LMT_AMT|
+--------------+------------+
|Chris Harmon LP|       13418|
|   Robin Yates|        8332|
| NanoForge Inc.|        3141|
|        Procum|       22699|
|        Procum|       22699|
| Alberta Cannon|        3716|
|  Daniel Hicks|       23957|
| Jennifer Long|        1492|
| Dorothy Holmes|        7482|
|    CZN Motors|       21460|
|    CZN Motors|       21460|
|Claudia Bennett|       21835|
|  Teresa Lyons|       11320|
| Clarissa Mills|       23555|
| Clarissa Mills|       23555|
|Robert Espinoza|       14630|
|         Polon|       15967|
|   Jason Dixon|        5876|
|  Manuel Allen|        1701|
|  Manuel Allen|        1701|
+--------------+------------+
only showing top 20 rows
```

How to filter specific data rows?

API

table.select(…).where(condition)

Condition example:
$"YEAR" > 2016

SQL

SELECT * FROM table WHERE condition

Condition example:
YEAR > 2016

Data filtering

Scala API: sqlContext.table("cr_card_dim").select($"EMBOSSED_NM",$"SPND_LMT_AMT" as "LIMIT").
where($"LIMIT" > 24900).show

SQL: sqlContext.sql("SELECT EMBOSSED_NM, SPND_LMT_AMT AS LIMIT FROM cr_card_dim
WHERE SPND_LMT_AMT > 24900").show

C:\WINDOWS\system32\cmd.exe - spark-shell

```
scala> sqlContext.table("cr_card_dim").select($"EMBOSSED_NM",$"SPND_LMT_AMT" as
"LIMIT").where($"LIMIT" > 24900).show
+--------------+-----+
|   EMBOSSED_NM|LIMIT|
+--------------+-----+
| Kenneth Davis|24956|
|Dorothy Morgan|24980|
|    Jorge Hall|24947|
| Theresa Scott|24907|
|   Nancy Craft|24972|
|    June Scott|24984|
|    June Scott|24984|
|   Lena Campos|24932|
|  Brandon Cole|24929|
|     Ross Knox|24998|
|   Ellen Klein|24972|
| Gregory Smith|24974|
+--------------+-----+
```

How to aggregate data?

API

table.groupBy(cols).agg(exps)

Expression examples:
count($"ID")
sum($"PRICE")

SQL

SELECT expr FROM table GROUP BY cols

Expression examples:
COUNT(*)
SUM(PRICE)

Grouping and aggregations

Scala API: `sqlContext.table("transaction").groupBy($"MERCH_CAT_KEY" as "CATEGORY").`
`agg(count($"TXN_KEY") as "COUNT", sum($"TXN_AMT") as "TOTAL").show`

SQL: `sqlContext.sql("SELECT MERCH_CAT_KEY AS CATEGORY, COUNT(TXN_KEY) AS COUNT,`
`SUM(TXN_AMT) AS TOTAL FROM transaction GROUP BY MERCH_CAT_KEY").show`

C:\WINDOWS\system32\cmd.exe - spark-shell

```
+--------+------+---------+
|CATEGORY| COUNT|    TOTAL|
+--------+------+---------+
|       1|626792|305993626|
|       2| 37557| 18219684|
|       3| 53839| 26163033|
|       4| 93667| 45206157|
|       5| 92375| 44676341|
+--------+------+---------+


scala>
```

How to join and sort data?

API	SQL
table.join(other, expr)	SELECT * FROM table JOIN other ON expr
table.orderBy(columns)	SELECT * FROM table ORDER BY columns
Columns examples: $"YEAR" $"PRICE".desc	Columns examples: YEAR PRICE DESC

Join and sort result

Scala API:

```
sqlContext.table("transaction").groupBy($"MERCH_CAT_KEY").
    agg(count($"TXN_KEY") as "COUNT", sum($"TXN_AMT") as "TOTAL").
    join(sqlContext.table("merch_cat_dim"),  "MERCH_CAT_KEY").
    select($"MERCH_CAT_NM" as "CATEGORY",$"COUNT",$"TOTAL").orderBy($"TOTAL".desc).show
```

SQL:

```
sqlContext.sql("SELECT merch_cat_dim.MERCH_CAT_NM AS CATEGORY, COUNT(TXN_KEY) AS COUNT,
    SUM(TXN_AMT) AS TOTAL FROM transaction JOIN merch_cat_dim
    ON transaction.MERCH_CAT_KEY = merch_cat_dim.MERCH_CAT_KEY
    GROUP BY merch_cat_dim.MERCH_CAT_NM ORDER BY TOTAL DESC").show
```

C:\WINDOWS\syste...

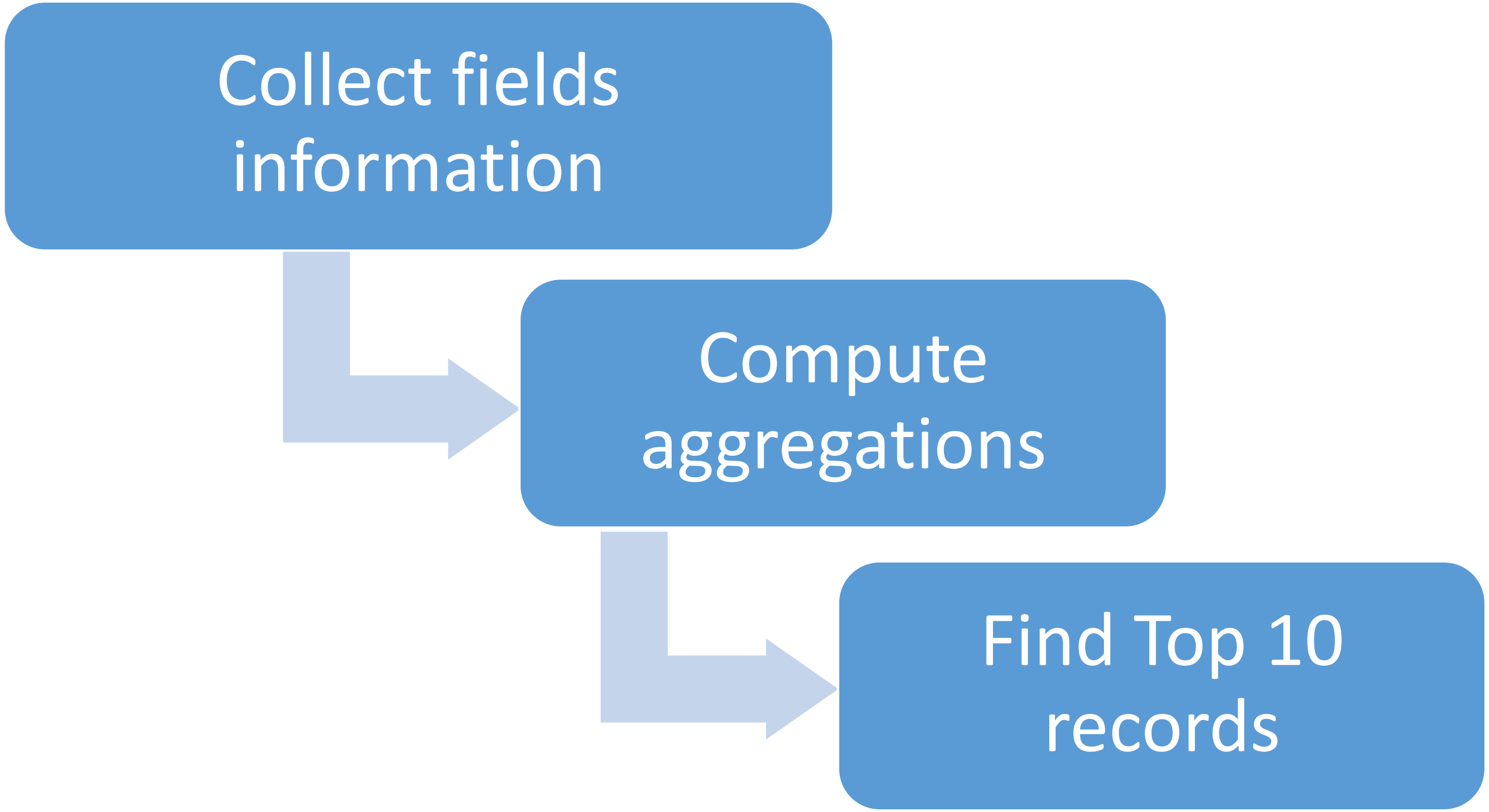
```
+--------+------+---------+
|CATEGORY| COUNT|    TOTAL|
+--------+------+---------+
|  Retail|626792|305993626|
|   Telco| 93667| 45206157|
|Services| 92375| 44676341|
|  Health| 53839| 26163033|
|     Car| 37557| 18219684|
+--------+------+---------+

scala>
```

Example: Data profiling application

- Extract interesting information from large sets of fuzzy data
- Look for various issues in data
- **Task**:
  - Compute and show field-by-field statistics
    - count, count distinct, count null, count not null – all supported types
    - average, minimum, maximum - numeric types only
    - first, last – numeric and string types
    - top 10 values

Application flow



Example dataset profile highlights

10 min of the Twitter firehose sample data collected by the Twitter Streaming example application

25,217 total statuses

149 columns

24,425 distinct users

106 locations

49 languages

228 possibly sensitive statuses

Weakness – sparse data

TWEETS.profile.txt - Notepad

File Edit Format View Help

column	dataType	count	count_distinct	count_null	count_no
type	string	25217	1	0	25217
contributorsIDs	array	25217	1	0	25217
createdAt	long	25217	591	0	25217
currentUserRetweetId	long	25217	1	0	25217
geoLocation	struct	25111	1	25111	0
geoLocation.latitude	double	106	106	0	106
geoLocation.longitude	double	106	106	0	106
hashtagEntities	array	25217	1	0	25217
id	long	25217	25217	0	25217
inReplyToScreenName	string	25217	4087	20907	4310
inReplyToStatusId	long	25217	3533	0	25217
inReplyToUserId	long	25217	4087	0	25217
isFavorited	boolean	25217	1	0	25217
isPossiblySensitive	boolean	25217	2	0	25217
isTruncated	boolean	25217	1	0	25217
mediaEntities	array	25217	1	0	25217
place	struct	24546	1	24546	0

Ln 1, Col 1

Extracted information from the data

TWEETS.profile.txt - Notepad

File Edit Format View Help

column	value	frequency
user.location	Null	10544
user.location	Brazil	103
user.location	Brasil	71
user.location	USA	64
user.location	United States	59
user.location	London	58
user.location	?stanbul, Türkiye	53
user.location	Argentina	52
user.location	UK	44
user.location	Rio de Janeiro, B...	44

Related Courses

- Adastra Academy courses on Apache Spark include:
  - Introduction to Apache Spark for Developers and Engineers
  - Scala in Practice
  - Advanced Apache Spark for Data Scientists and Developers

- Go to our website: www.adastra.academy

- Use coupon code: TASM127

- Email us: info@adastra.academy

Thank you!