



.....

Gökhan Atıl



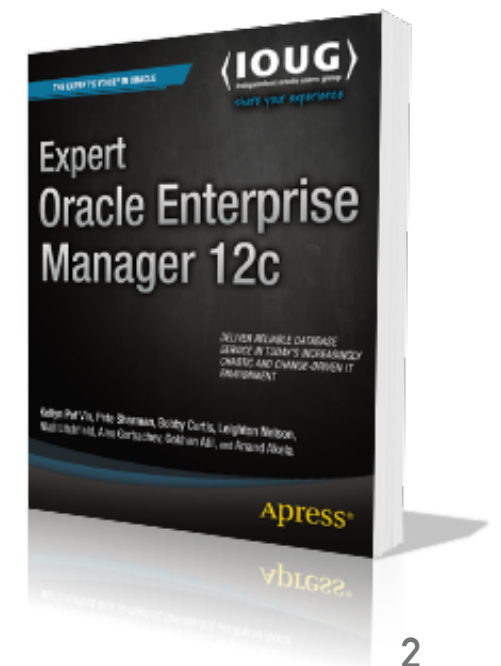
ORACLE
ACE Director



GÖKHAN ATIL



- Database Administrator
- Oracle ACE Director (2016)
ACE (2011)
- 10g/11g and R12 Oracle Certified Professional (OCP)
- Co-author of Expert Oracle Enterprise Manager 12c
- Founding Member and Vice President of **TROUG**
- Blogger (since 2008) gokhanatil.com
- Twitter: [@gokhanatil](https://twitter.com/gokhanatil)



APACHE SPARK WITH PYTHON



- Introduction to Apache Spark
- Why Python (*PySpark*) instead of Scala?
- Spark RDD
- SQL and DataFrames
- Spark Streaming
- Spark Graphx
- Spark MLlib (Machine Learning)

INTRODUCTION TO APACHE SPARK

- A fast and general engine for large-scale data processing
- Top-Level Apache Project since 2014.
- Response to limitations in the MapReduce
- Run programs up to **100x** faster than Hadoop *MapReduce* in memory, or **10x** faster on disk
- Implemented in Scala programming language, supports Java, Scala, Python, R
- Runs on Hadoop, Mesos, Kubernetes, standalone, cloud

DOWNLOAD AND RUN ON YOUR PC

- <https://spark.apache.org/downloads.html>

1. Choose a Spark release: 2.3.0 (Feb 28 2018) 
2. Choose a package type: Pre-built for Apache Hadoop 2.7 and later 
3. Download Spark: [spark-2.3.0-bin-hadoop2.7.tgz](#)

- Extract and Spark is ready:

```
tar -xzf spark-2.3.0-bin-hadoop2.7.tgz
```

```
spark-2.3.0-bin-hadoop2.7/bin/spark
```

- You can also use PIP:

```
pip install pyspark
```

PYSPARK AND SPARK-SUBMIT

- PySpark is the interface that gives access to Spark using the Python programming language

Welcome to

[illegible]

```
Using Python version 2.7.14 (default, Mar 27 2018 12:28:59)
SparkSession available as 'spark'.
>>>
```

- The spark-submit script in Spark's bin directory is used to launch applications on a cluster

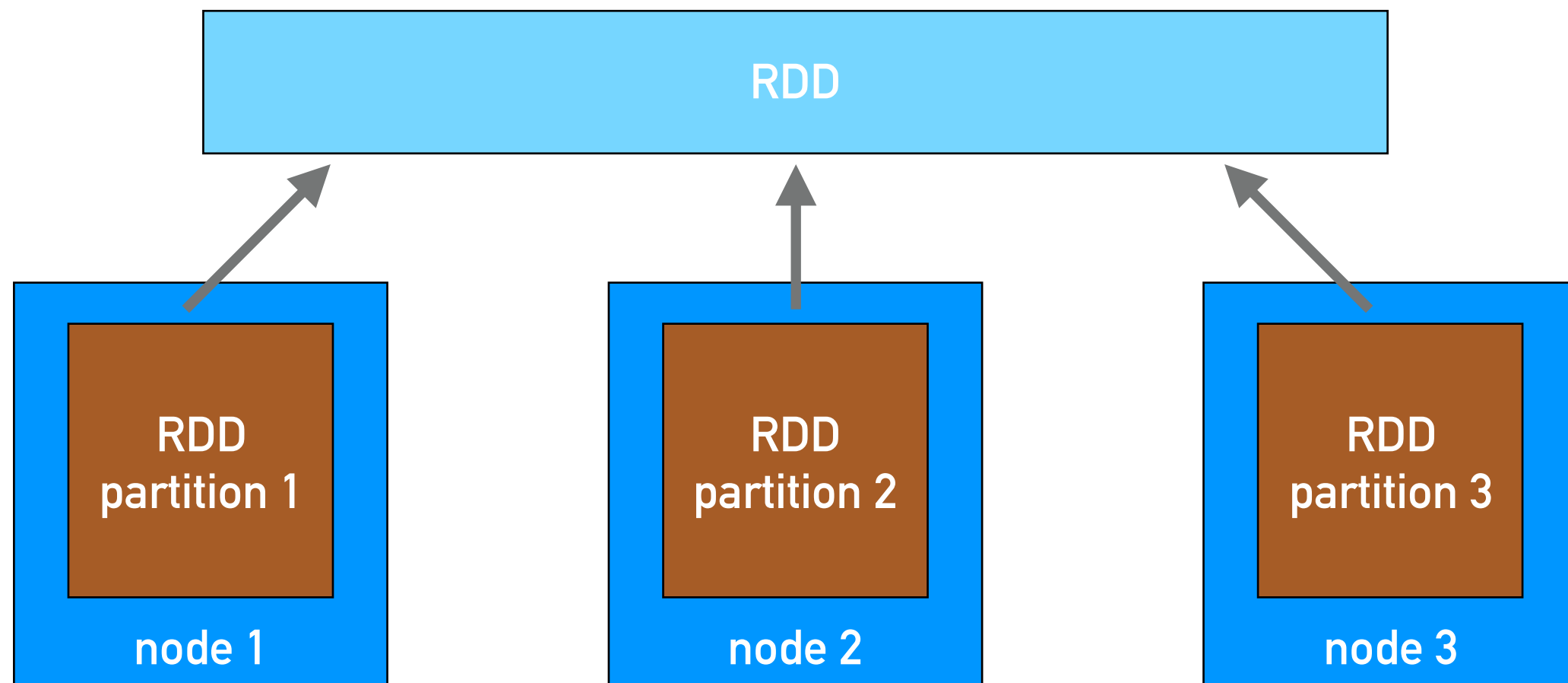
spark-submit *example1.py*

WHY PYTHON INSTEAD OF SCALA?

- If you know Scala, then use Scala!
- Learning curve: Python is comparatively easier to learn
- Easy to use: Code readability, maintainability and familiarity is far better with Python
- Libraries: Python comes with great libraries for data analysis, statistics and visualization (numpy, pandas, matplotlib etc...)
- Performance: Scala is faster than Python but if your Python code just calls Spark libraries, the differences in performance is minimal (*)
- 🔊 **Reminder: Any new feature added in Spark API will be available in Scala first**

RESILIENT DISTRIBUTED DATASET (RDD)

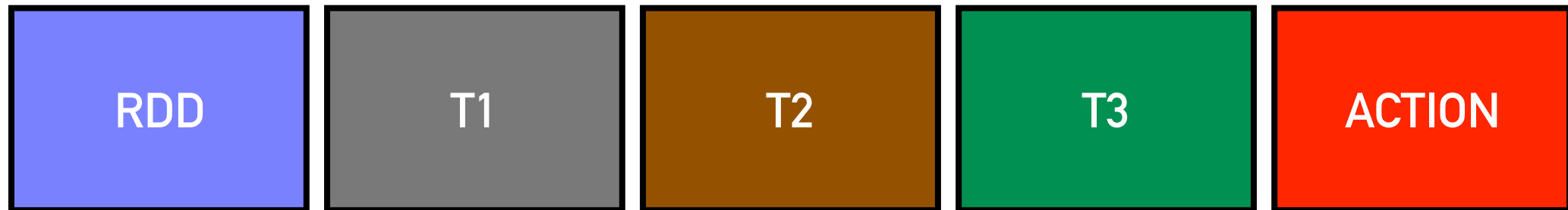
- RDDs are the core data structure in Spark
- Distributed, resilient, immutable, can store unstructured and structured data, lazy evaluated



RDD TRANSFORMATIONS AND ACTIONS

SPARK CONTEXT

```
sc.textFile(*).map(*).filter(*).reduceByKey(*).collect()
```



LAZY EVALUATION

TRANSFORMATIONS

- map
- filter
- flatMap
- mapPartitions
- reduceByKey
- union
- intersection
- join

ACTIONS

- collect
- count
- first
- take
- takeSample
- takeOrdered
- saveAsTextFile
- foreach

HOW TO CREATE RDD IN PYSPARK

- Referencing a dataset in an external storage system:

```
rdd = sc.textFile( ... )
```

- Parallelizing already existing collection:

```
rdd = sc.parallelize( ... )
```

- Creating RDD from already existing RDDs:

```
rdd2 = rdd1.map( ... )
```

USERS.CSV (MOVIELENS DATABASE)

id | age | gender | occupation | zip

1 | 24 | M | technician | 85711

2 | 53 | F | other | 94043

3 | 23 | M | writer | 32067

4 | 24 | M | technician | 43537

5 | 33 | F | other | 15213

6 | 42 | M | executive | 98101

7 | 57 | M | administrator | 91344

8 | 36 | M | administrator | 05201

M = 670
F = 273

EXAMPLE #1: USE RDD TO GROUP DATA FROM CSV

```
from pyspark import SparkContext
```

```
sc = SparkContext.getOrCreate()
```

```
print sc.textFile( "users.csv" ) \
```

```
.map( lambda x: (x.split("|")[2], 1) ) \
```



M, 1
M, 1
F, 1
M, 1

```
.reduceByKey(lambda x,y:x+y).collect()
```

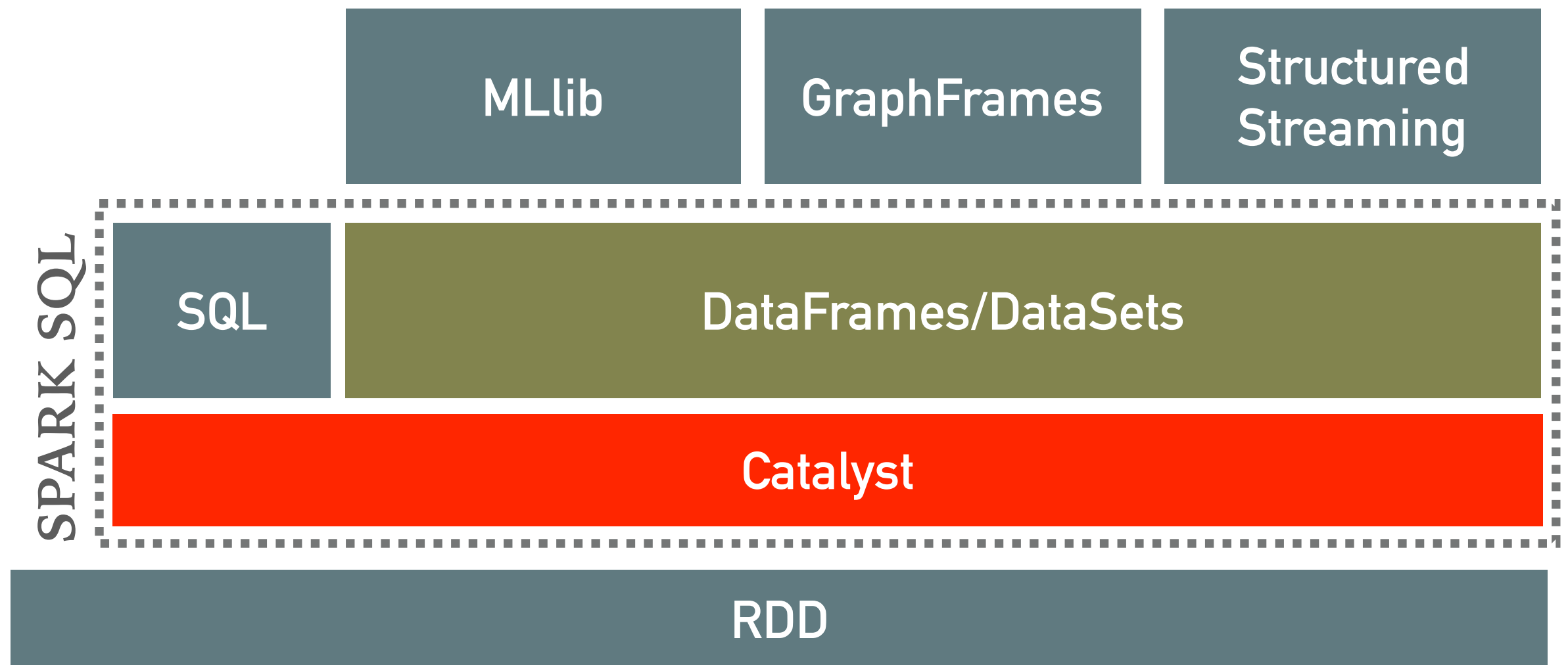
```
sc.stop()
```



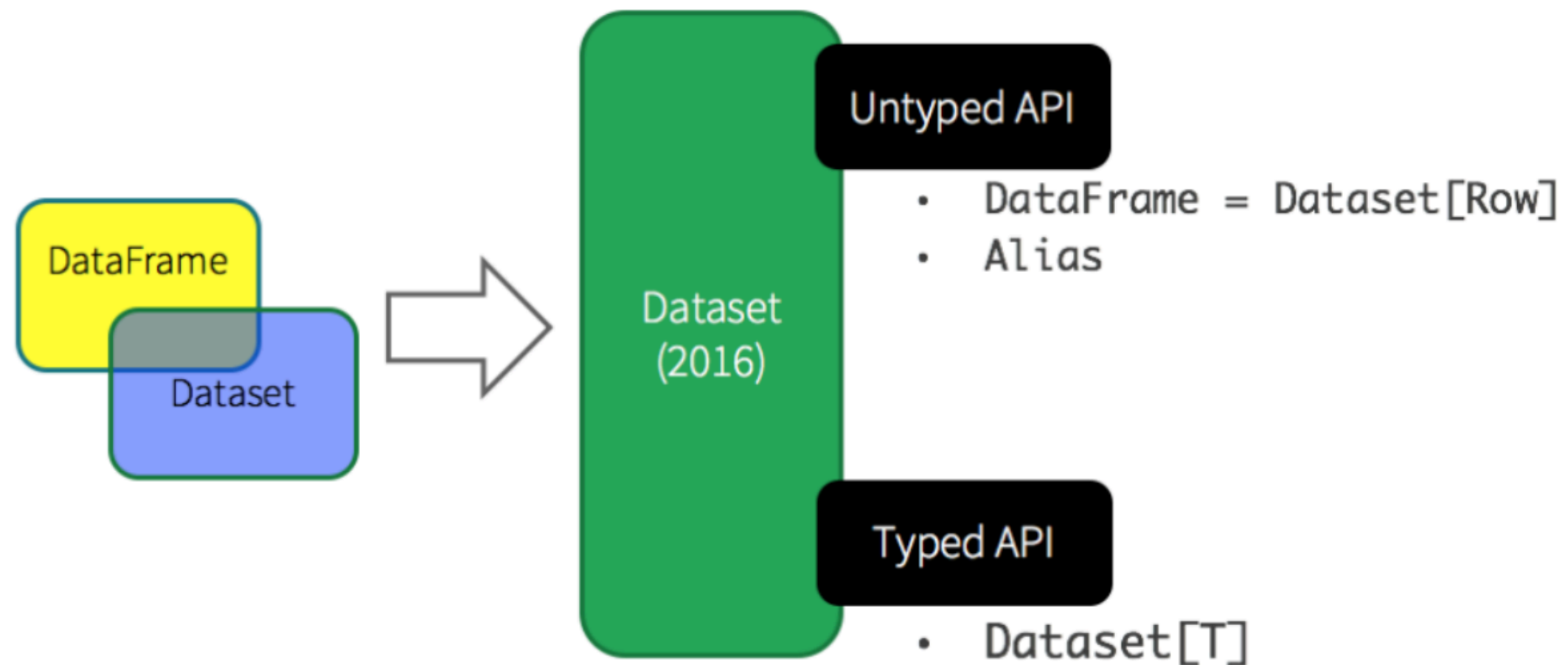
[(u'M', 670), (u'F', 273)]

SPARK SQL AND DATAFRAMES

- Spark SQL is Apache Spark's module for working with structured data



DATAFRAMES AND DATASETS



- DataFrame is a distributed collection of "structured" data, organized into named columns.
- Spark DataSets are statically typed, while Python is a dynamically typed programming language so Python supports only DataFrames.

EXAMPLE #2: USE DATAFRAME TO GROUP DATA FROM CSV

```
from pyspark import SparkContext
```

```
from pyspark.sql import SparkSession
```

```
sc = SparkContext.getOrCreate()
```

```
spark = SparkSession(sc)
```

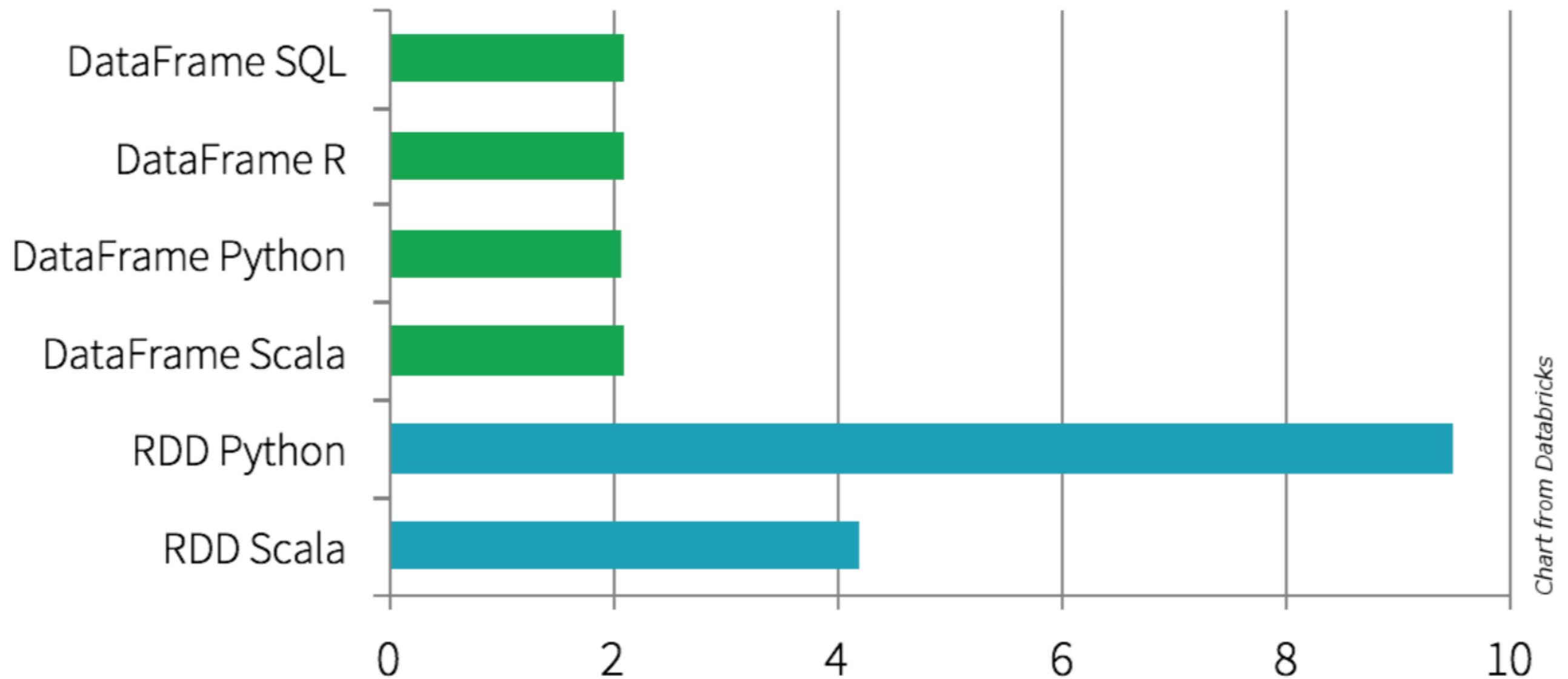
```
spark.read.load( "users.csv", format="csv", sep="|" ) \
```

```
.toDF( "id","age","gender","occupation","zip" ) \
```

```
.groupby( "gender" ).count().show()
```

```
sc.stop()
```

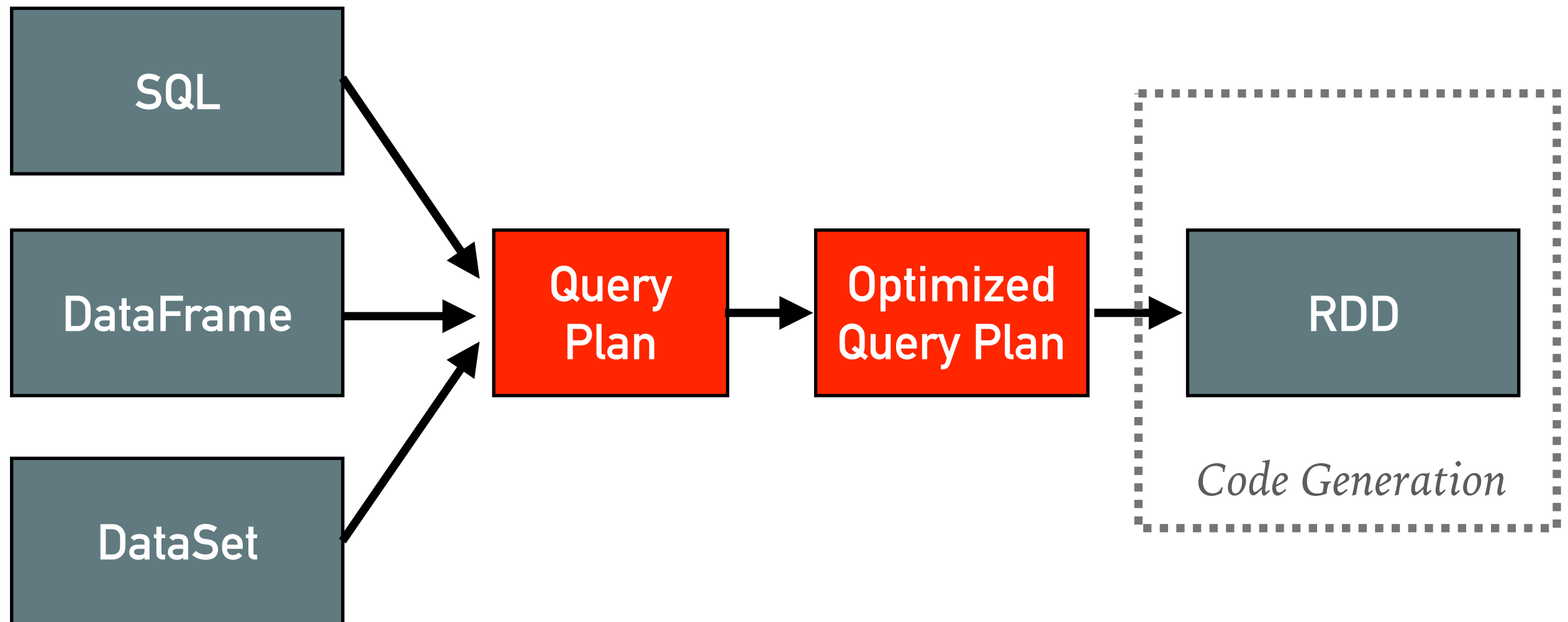
DATAFRAME VERSUS RDD



Time to aggregate 10 million integer pairs (in seconds)

CATALYST OPTIMIZER

- Spark SQL uses Catalyst optimizer to optimize query plans.
- Supports cost-based optimization since Spark 2.2



CONVERSION BETWEEN RDD AND DATAFRAME

- An RDD can be converted to DataFrame using `createDataFrame` or `toDF` method:

```
rdd = sc.parallelize([("osman",21),("ahmet",25)])
```

```
df = rdd.toDF( "name STRING, age INT" )
```

```
df.show()
```

- You can access underlying RDD of a DataFrame using `rdd` property:

```
df.rdd.collect()
```

```
[Row(name=u'osman',age=21),Row(name=u'ahmet',age=25)]
```

EXAMPLE #3: CREATE TEMPORARY VIEWS FROM DATAFRAMES

```
spark.read.load( "users.csv", format="csv", sep="|" ) \  
    .toDF( "id","age","gender","occupation","zip" ) \  
    .createOrReplaceTempView( "users" )
```

```
spark.sql( "select count(*) from users" ).show()
```

```
spark.sql( "select case when age < 25 then '-25' \  
    when age between 25 and 39 then '25-40' \  
    when age >= 40 then '40+' end age_group, \  
    count(*) from users group by age_group order by 1" ).show()
```

EXAMPLE #4: READ AND WRITE DATA

```
df = spark.read.load( "users.csv", format="csv", sep="|" ) \  
    .toDF( "id","age","gender","occupation","zip" )
```

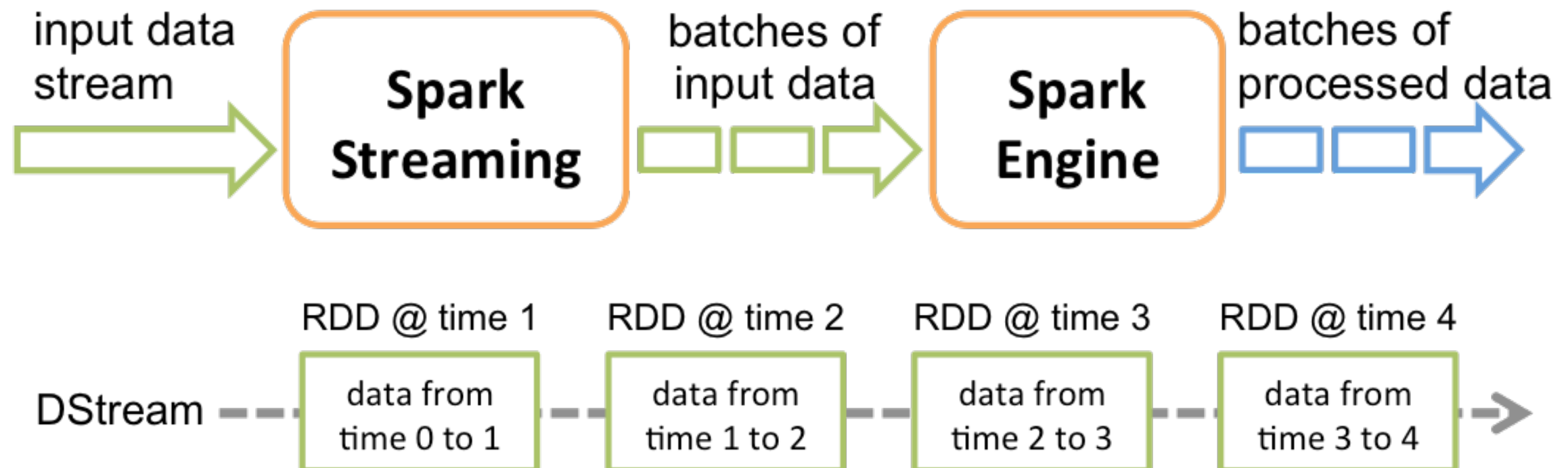
```
df.write.saveAsTable("users")  HIVE
```

```
df.write.save("users.json", format="json", mode="overwrite")
```

```
spark.sql("SELECT gender, count(*) FROM \  
    json.`users.json` GROUP BY gender").show()
```

SPARK STREAMING (DSTREAMS)

- Scalable, high-throughput, fault-tolerant stream processing of live data streams
- Supports: File, Socket, Kafka, Flume, Kinesis
- Spark Streaming receives live input data streams and divides the data into batches



EXAMPLE #5: DISCRETIZED STREAMS (DSTREAMS)

```
ssc = StreamingContext(sc, 1)
```

```
stream_data = ssc.textFileStream("file:///tmp/stream") \  
    .map( lambda x: x.split(","))
```

```
stream_data.pprint()
```

```
ssc.start()
```

```
ssc.awaitTermination()
```


EXAMPLE #5: OUTPUT

.....

Fatih,5
Cenk,4
Ahmet,3
Arda,1



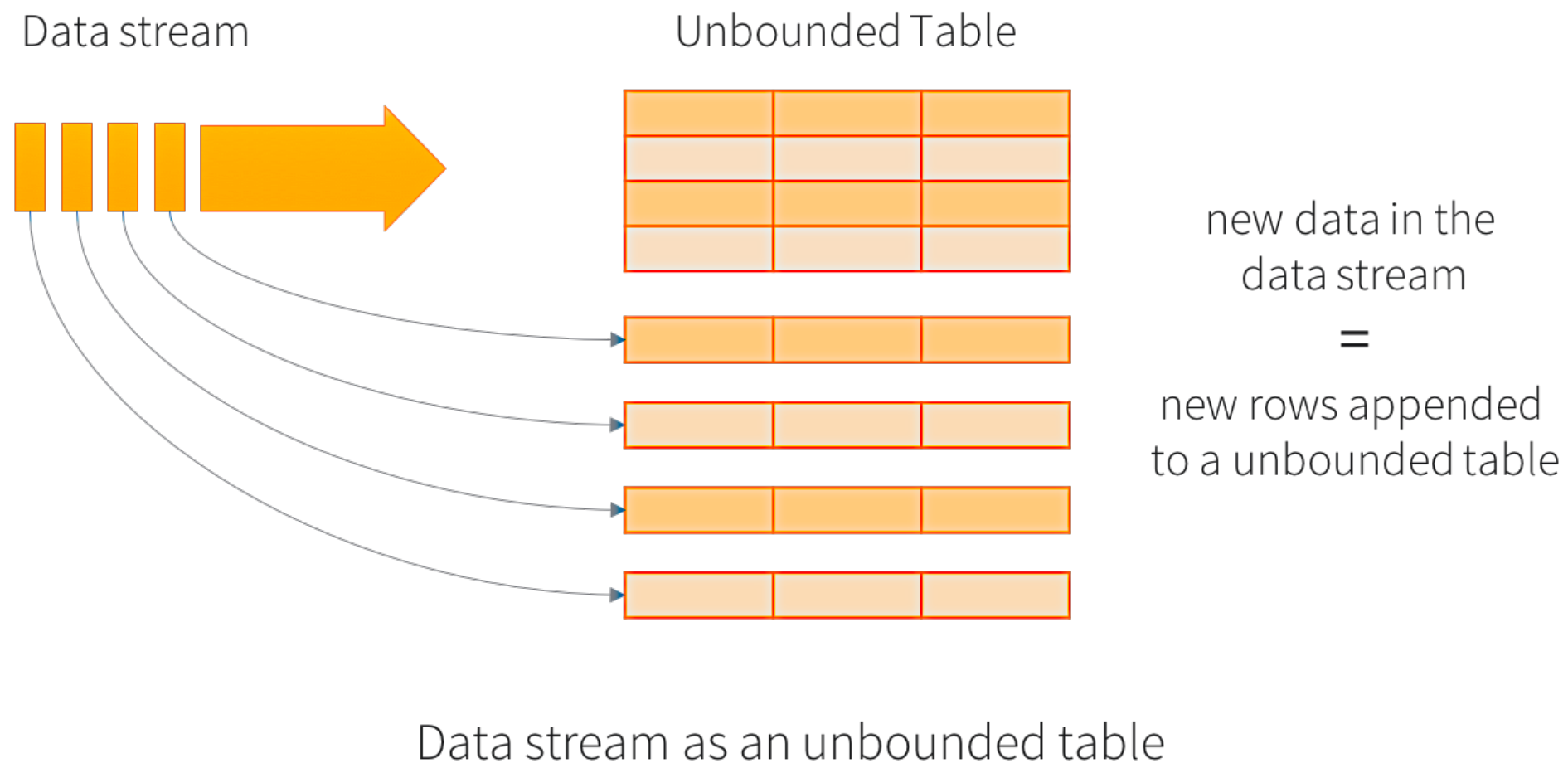
Time: 2018-04-09 12:09:06

[u'Fatih', u'5']
[u'Cenk', u'4']
[u'Ahmet', u'3']
[u'Arda', u'1']

Time: 2018-04-09 12:09:07

STRUCTURED STREAMING

- Stream processing engine built on the Spark SQL engine
- Supports File and Kafka sources for production; Socket and Rate sources for testing



EXAMPLE #6: STRUCTURED STREAMING

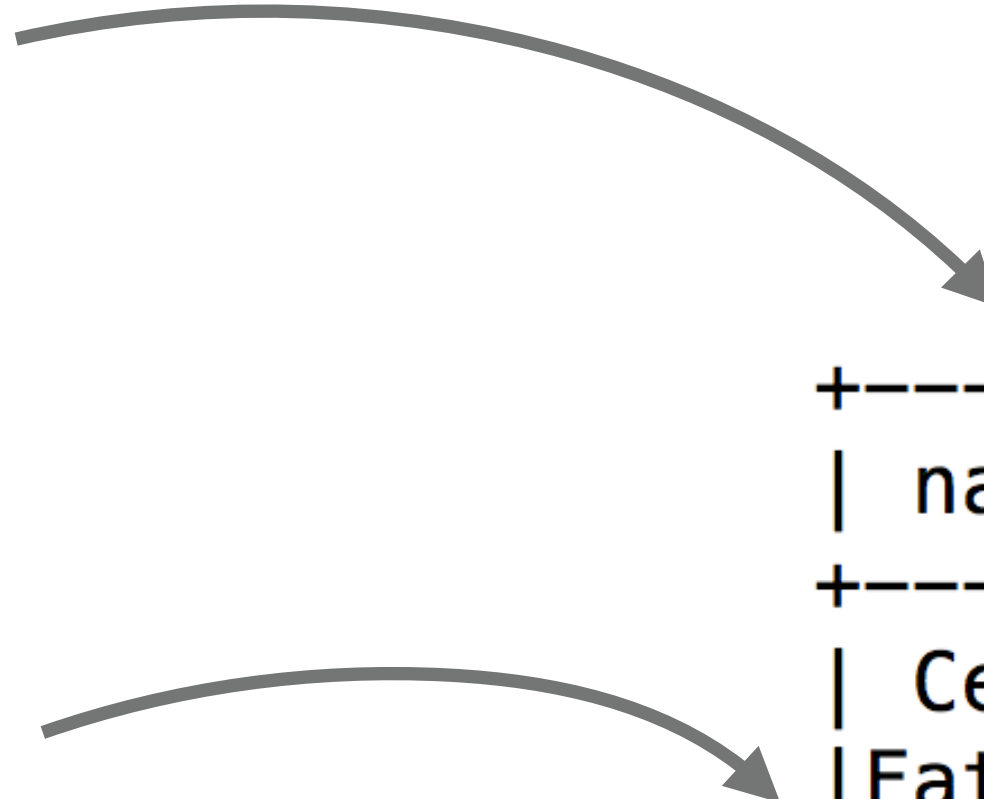
```
stream_data = spark.readStream \  
    .load( format="csv",path="/tmp/stream/*.csv",  
    schema="name string, points int" ) \  
    .groupBy("name").sum("points").orderBy( "sum(points)",  
    ascending=0 )
```

```
stream_data.writeStream.start( format="console",  
    outputMode="complete" ).awaitTermination()
```

EXAMPLE #6: OUTPUT

Fatih,5
Cenk,4
Ahmet,3
Arda,1

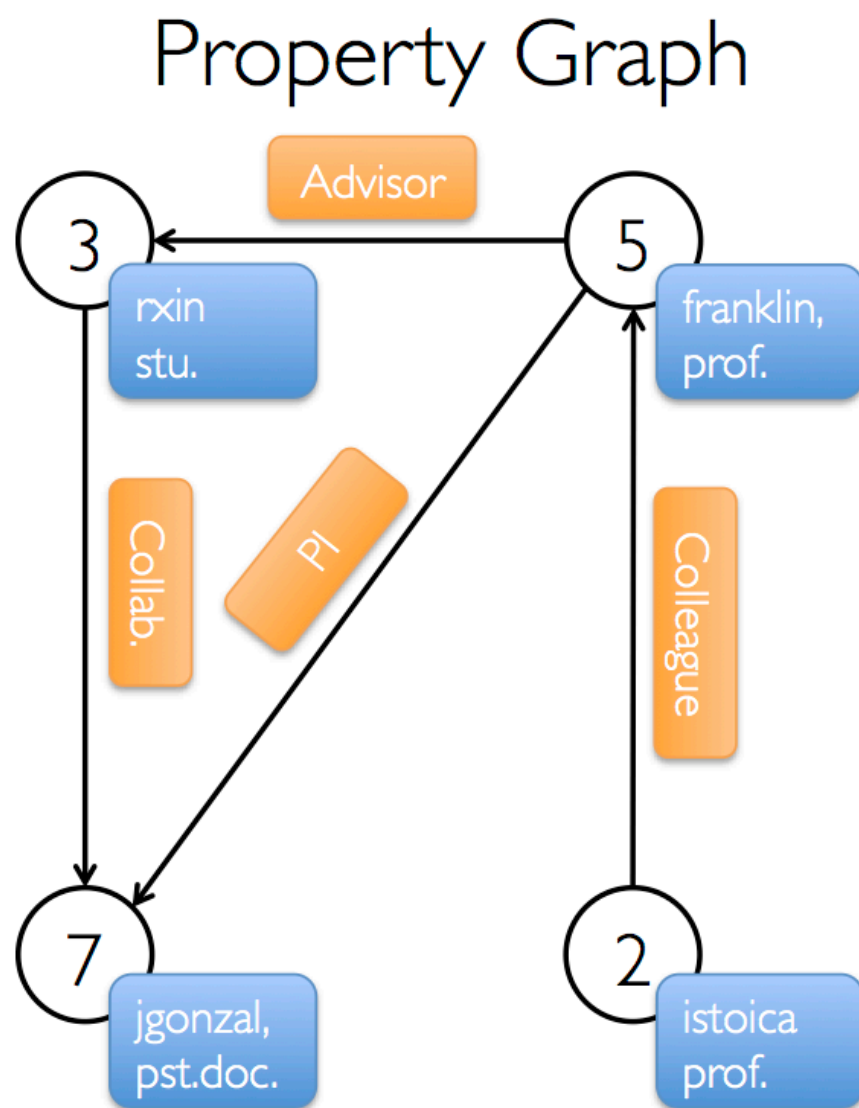
Cenk,4
Fatih,3
Osman,2
Ahmet,1
Arda,1



| +-----+-----+ | |
|---------------|-------------|
| name | sum(points) |
| +-----+-----+ | |
| Cenk | 8 |
| Fatih | 8 |
| Ahmet | 4 |
| Arda | 2 |
| Osman | 2 |
| +-----+-----+ | |

~~GRAPHX~~ (GRAPHFRAMES)

- GraphX is a *new* component in Spark for graphs and graph-parallel computation.



Vertex Table

| Id | Property (V) |
|----|-----------------------|
| 3 | (rxin, student) |
| 7 | (jgonzal, postdoc) |
| 5 | (franklin, professor) |
| 2 | (istoica, professor) |

Edge Table

| SrcId | DstId | Property (E) |
|-------|-------|--------------|
| 3 | 7 | Collaborator |
| 5 | 3 | Advisor |
| 2 | 5 | Colleague |
| 5 | 7 | PI |

EXAMPLE #7: GRAPHFRAMES

vertex =

```
spark.createDataFrame([
```

```
    (1, "Ahmet"),
```

```
    (2, "Mehmet"),
```

```
    (3, "Cengiz"),
```

```
    (4, "Osman")],
```

```
    ["id", "name"])
```

edges =

```
spark.createDataFrame([
```

```
    ( 1, 2, "friend" ),
```

```
    ( 2, 1, "friend" ),
```

```
    ( 2, 3, "friend" ),
```

```
    ( 3, 2, "friend" ),
```

```
    ( 2, 4, "friend" ),
```

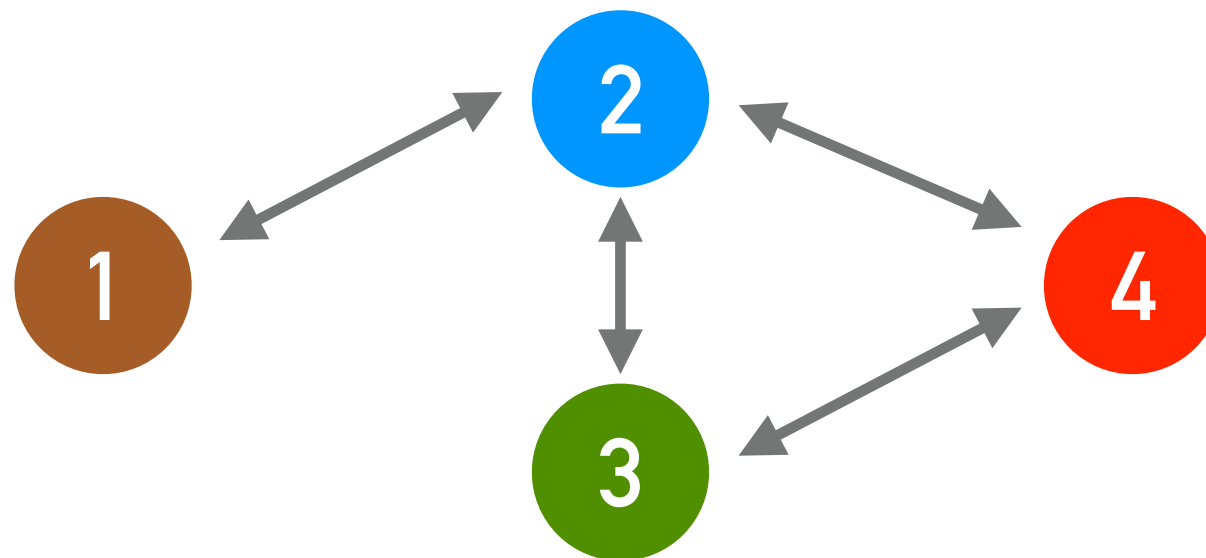
```
    ( 4, 2, "friend" ),
```

```
    ( 3, 4, "friend" ),
```

```
    ( 4, 3, "friend" )],
```

```
    ["src", "dst", "relation"])
```

EXAMPLE #7: GRAPHFRAMES



```
pyspark --packages graphframes:graphframes:0.5.0-spark2.1-s_2.11
```

```
import graphframes as gf
```

```
g = gf.GraphFrame(vertex, edges)
```

```
g.shortestPaths([4]).show()
```

```
+---+-----+-----+
| id|  name|distances|
+---+-----+-----+
|  1| Ahmet|[4 -> 2]|
|  3| Cengiz|[4 -> 1]|
|  2| Mehmet|[4 -> 1]|
|  4|  Osman|[4 -> 0]|
+---+-----+-----+
```

MLLIB (MACHINE LEARNING)

- Supports common ML Algorithms such as classification, regression, clustering, and collaborative filtering
- Featurization:
 - Feature extraction (TF-IDF, Word2Vec, CountVectorizer ...)
 - Transformation (Tokenizer, StopWordsRemover ...)
 - Selection (VectorSlicer, RFormula ...)
- Pipelines: combine multiple algorithms into a single pipeline, or workflow
- DataFrame-based API is primary API

EXAMPLE #8: ALTERNATING LEAST SQUARES (ALS)

```
def parseratings( x ):
```

```
    v = x.split("::")
```

```
    return (int(v[0]), int(v[1]), float(v[2]))
```

```
ratings = sc.textFile("ratings.dat").map(parseratings) \
```

```
    .toDF( ["user", "id", "rating"] )
```

```
als = ALS(userCol="user", itemCol="id", ratingCol="rating")
```

```
model = als.fit(ratings)
```

```
model.recommendForAllUsers(10).show()
```

EXAMPLE #8 OUTPUT

| | |
|--|--|
| | |
| +-----+ | |
| title | |
| +-----+ | |
| Fight Club (1999) | |
| More (1998) | |
| Amelie (Fabuleux destin d'Amélie Poulain, Le) (2001) | |
| The Lord of the Rings: The Fellowship of the Ring (2001) | |
| The Lord of the Rings: The Two Towers (2002) | |
| The Lord of the Rings: The Return of the King (2003) | |
| The Life of Oharu (Saikaku ichidai onna) (1952) | |
| The Man Who Planted Trees (Homme qui plantait des arbres, L') (1987) | |
| Shadows of Forgotten Ancestors (1964) | |
| Sun Alley (Sonnenallee) (1999) | |
| +-----+ | |

