USING SPARK EFFICIENTLY

DATASETS I DATAFRAMES I SQL

NAME: GARREN STAUBLI

CONSULTANT @ BLUEPRINT CONSULTING SERVICES

SPECIALTY: BIG DATA ENGINEERING

Blueprint™
CONSULTING SERVICES

DISCLAIMERS

I am not a Robot.

Non-OCD friendly formatting may follow.

The views expressed in this presentation probably belong to various different companies to whom I have sold my soul unwittingly by mindlessly accepting Terms of Service and Use.

DISTRIBUTED DATA STRUCTURES

Spark Core:

- RDD (Resilient Distributed Dataset)

Spark SQL:

- Dataset (e.g. Dataset[CountryGDP])
  - Typed templates
  - Matching schema
  - Relational and functional operations
  - Lazily Evaluated
- DataFrame (aka Dataset[Row])
  - Uses generic Row object

DATASETS

org.apache.spark.sql.Dataset[T]

Optimized

Parallelized

Immutable

Encoded

Strongly Typed

O(PIES)

"A DATASET IS A **STRONGLY TYPED COLLECTION** OF **DOMAIN-SPECIFIC OBJECTS** THAT CAN BE TRANSFORMED IN **PARALLEL** USING **FUNCTIONAL OR RELATIONAL OPERATIONS**."

DATASETS

org.apache.spark.sql.Dataset[T]

Optimized

Storage	Tungsten
Execution	Catalyst

O(PIES)

DATASETS

org.apache.spark.sql.Dataset[T]

Optimized [Storage] => Tungsten

"JAVA OBJECTS HAVE A LARGE INHERENT MEMORY OVERHEAD."

O(PIES)

DATASETS

org.apache.spark.sql.Dataset[T]

Optimized [Storage] => Tungsten

Tungsten Objectives:

1. Improve CPU and Memory efficiency

2. Push performance closer to bare metal

O(PIES)

DATASETS

org.apache.spark.sql.Dataset[T]

Optimized [Storage] => Tungsten

Memory Management and Binary Processing

- Manage memory explicitly
- Eliminate JVM object model overhead
- Reduce garbage collection

Cache-aware computation

- Algorithms and data structures to exploit memory hierarchy

Code generation

- Using code generation to exploit modern compilers and CPUs

O(PIES)

DATASETS

org.apache.spark.sql.Dataset[T]

Optimized [Execution] => Catalyst

YOUR REQUEST

exhaust gas

HC (hydrogen)
CO (carbon monoxide)
NOX (nitrogen oxid

EXECUTION PLANNING

major reaction

CO+1/2 O2=CO2
H4C2+3O2=2CO2+2H2O
CO+NOX=CO2+N2

catalytic active mat

alumina oxide - Al2O
cerum oxide - CeO2
rare earth stabilizers
metals - Pt/Pd/Rh

SPARK CATALYST

tail pipe emissions

H2O (water)
CO2 (carbon dioxide)
N2 (nitrogen)

GENERATED CODE

O(PIES)

DATASETS

org.apache.spark.sql.Dataset[T]

Optimized [Execution] => Catalyst

YOUR REQUEST

```
SELECT sum(v)
FROM (
  SELECT
    t1.id,
    1 + 2 + t1.value AS v
  FROM t1 JOIN t2
  WHERE
    t1.id = t2.id AND
    t2.id > 50000) tmp
```

O(PIES)

DATASETS

org.apache.spark.sql.Dataset[T]

Optimized [Execution] => Catalyst

SPARK CATALYST

O(PIES)

DATASETS

org.apache.spark.sql.Dataset[T]

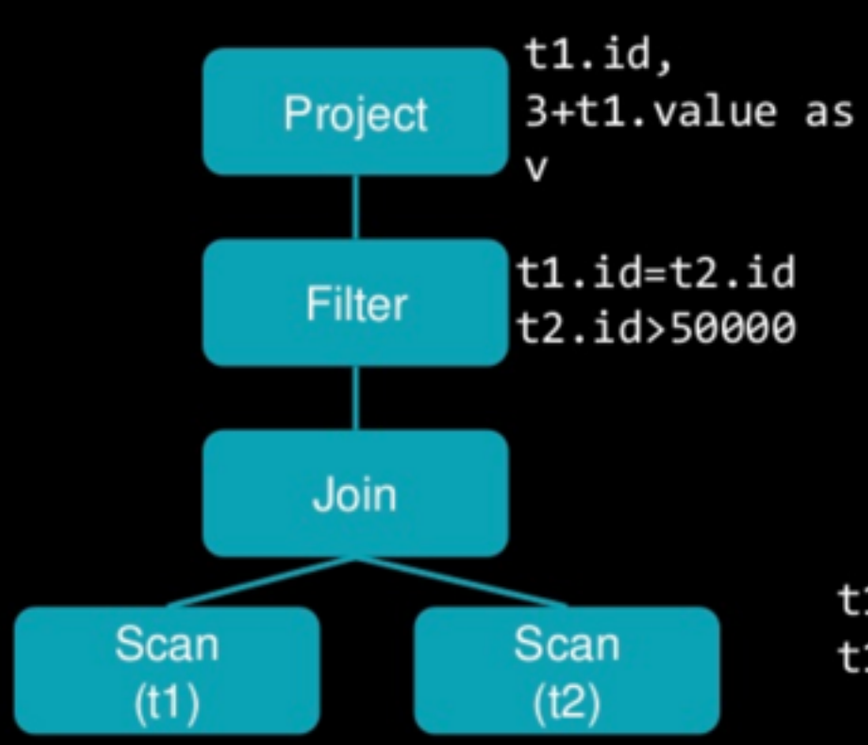
Optimized [Execution] => Catalyst

EXECUTION PLANNING

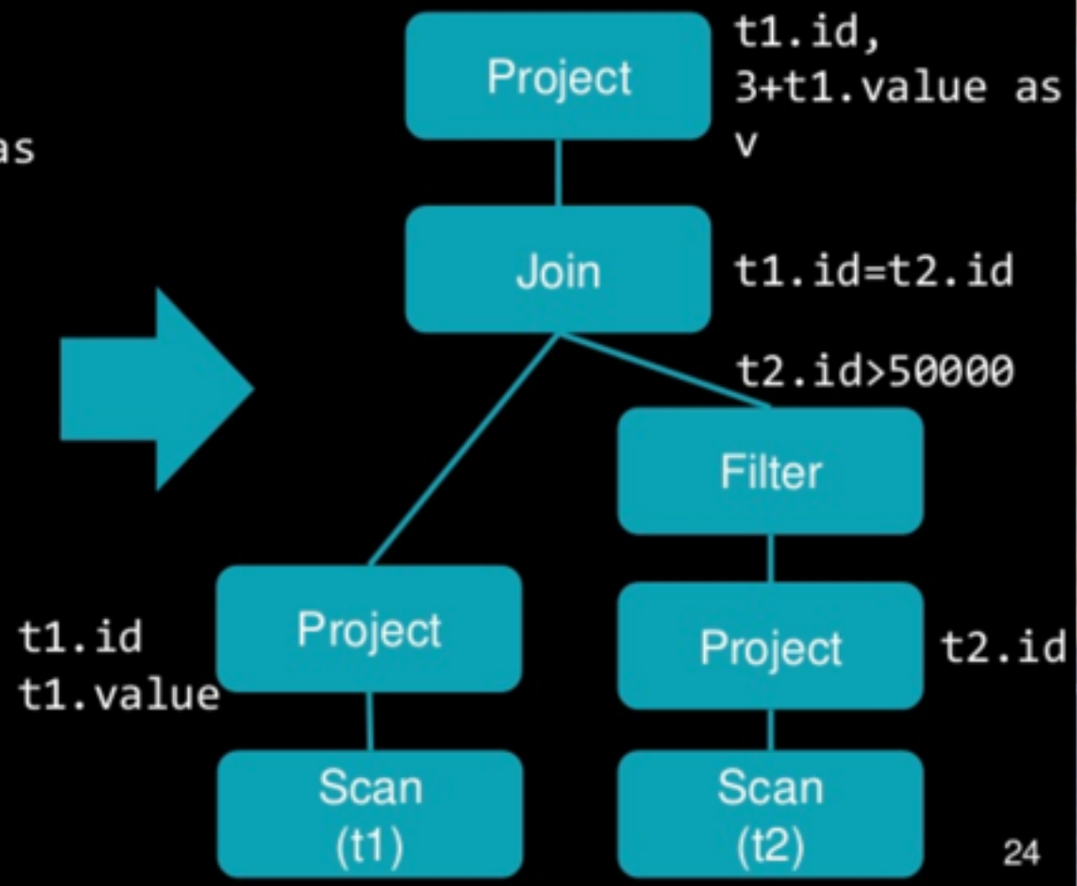
GENERATED CODE

Combining Multiple Rules

Before transformations



After transformations



databricks

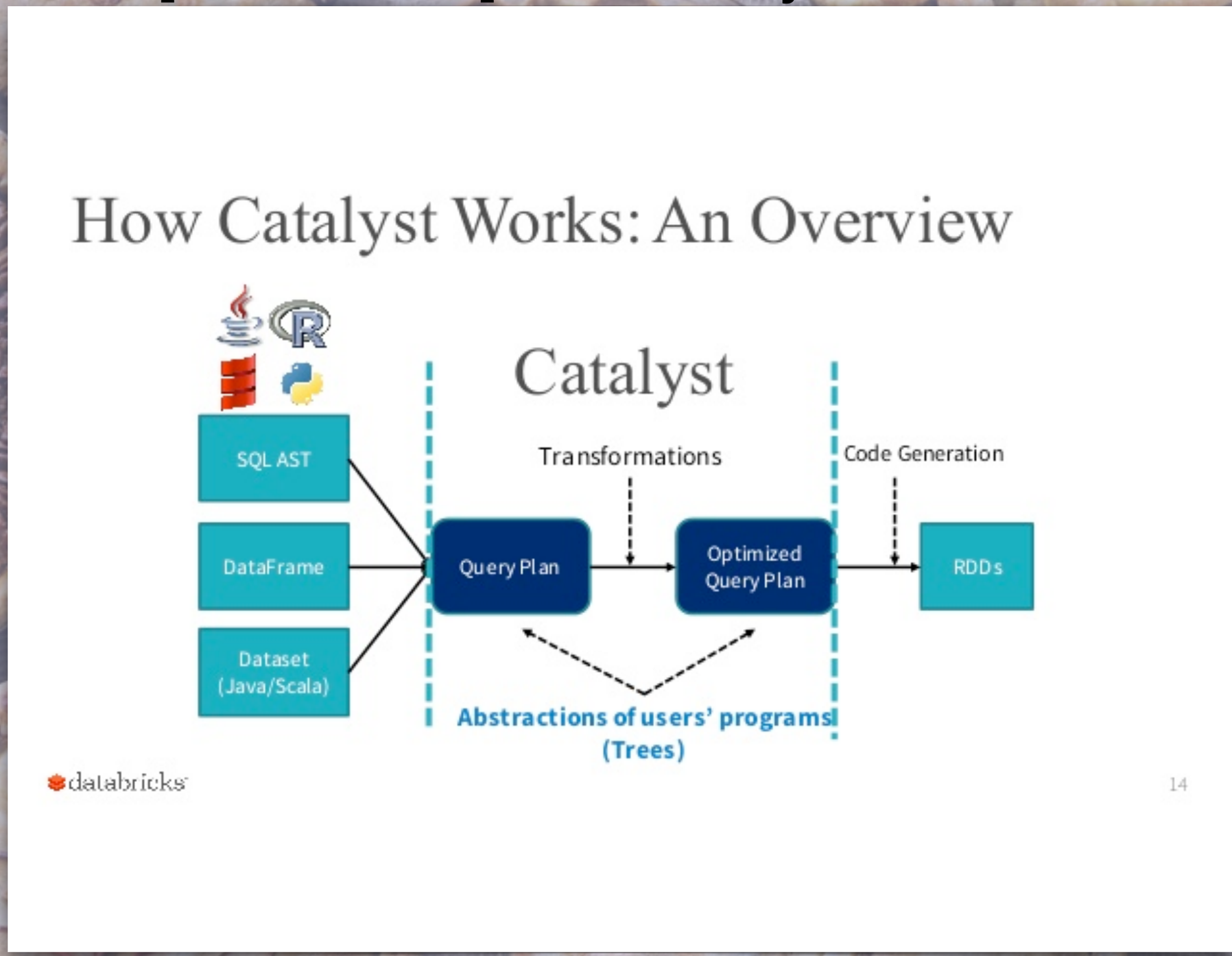
24

O(PIES)

DATASETS

org.apache.spark.sql.Dataset[T]

Optimized [Execution] => Catalyst

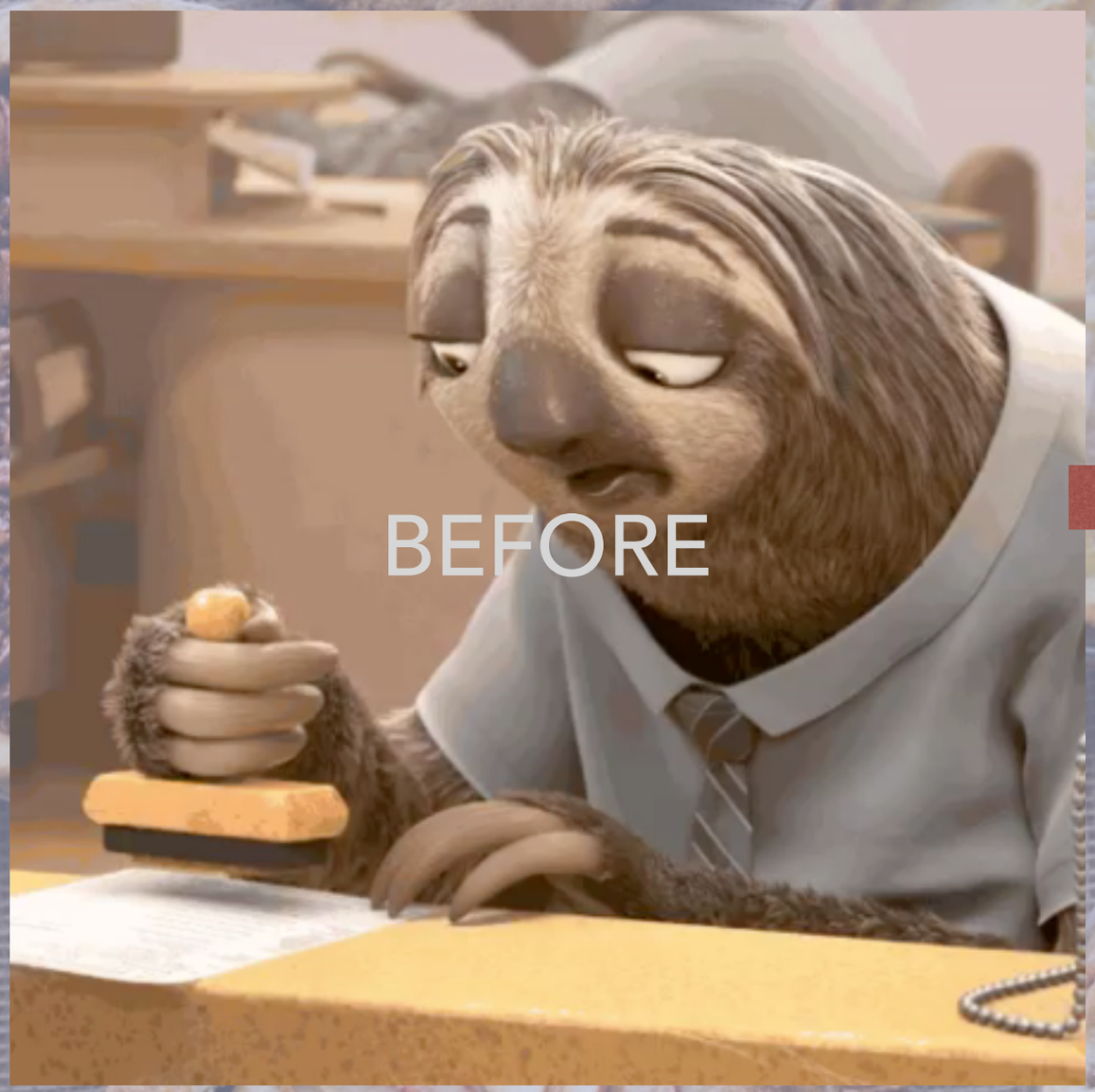


O(PIES)

DATASETS

org.apache.spark.sql.Dataset[T]

Parallelized



O(PIES)

DATASETS

org.apache.spark.sql.Dataset[T]

Parallelized

Partitions

- Logical splits of data
- 64MB like HDFS Blocks

Tasks

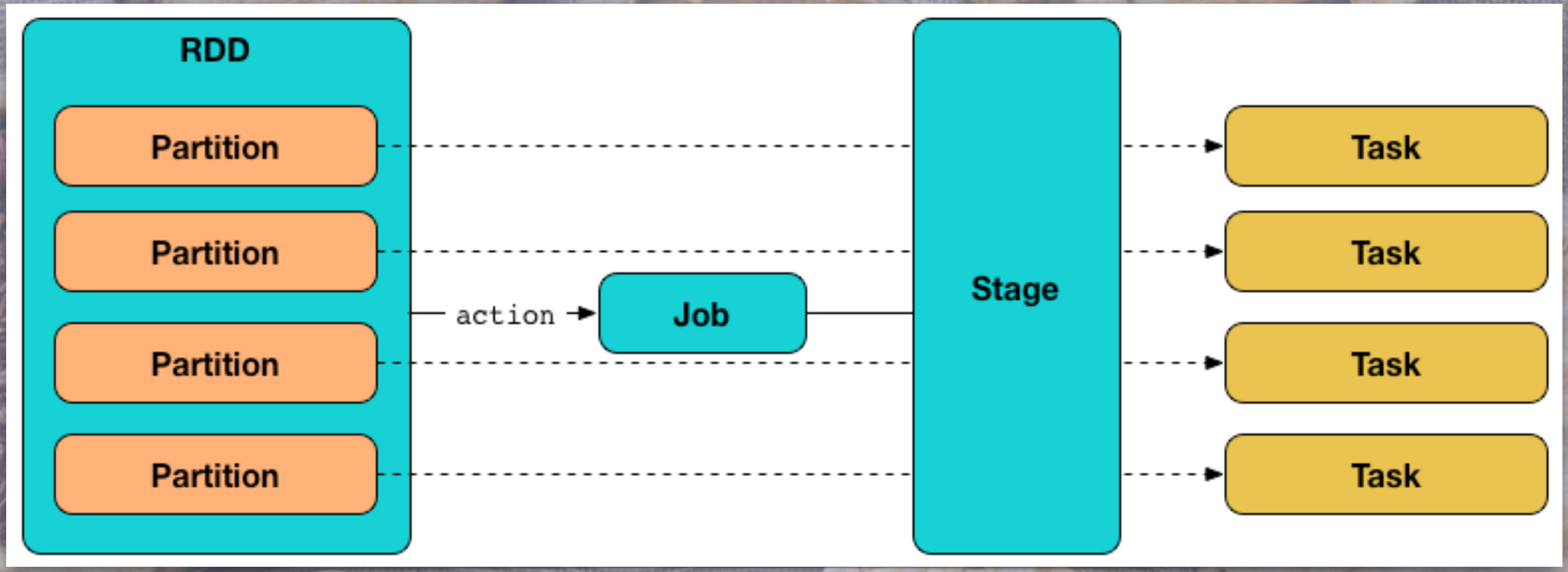
- Individual unit of execution

O(PIES)

DATASETS

org.apache.spark.sql.Dataset[T]

Parallelized



DATASETS

org.apache.spark.sql.Dataset[T]

O(PIES)

Immutable

O(PIES)

DATASETS

org.apache.spark.sql.Dataset[T]

Immutable

```
1. d = {'foo': 'bar'}
2. print(d['foo']) # => 'bar'
3. d['foo'] = 'qux'
4. print(d['foo']) # => 'qux'
```

```
1. df.foreach(print) # Row(foo='qux')
2. df['baz'] = 'qux'
3. # TypeError: 'DataFrame' object does not support item assignment
```

```
5. df.withColumn("baz", lit("qux"))
6. df.foreach(print) # Row(foo='qux')
```

```
8. new_df = df.withColumn("baz", lit("qux"))
9. new_df.foreach(print) # Row(foo='qux', baz='qux')
```

O(PIES)

Datasets

org.apache.spark.sql.Dataset[T]

Encoded

Data validation

```
1. [{"name": "WSU", "yearFounded": 1890,"numStudents": 29686},
2. {"name": "MIT", "yearFounded": 1860,"numStudents": 11318},
3. {"name": "UW", "yearFounded": 1861,"numStudents": 46081}]
```

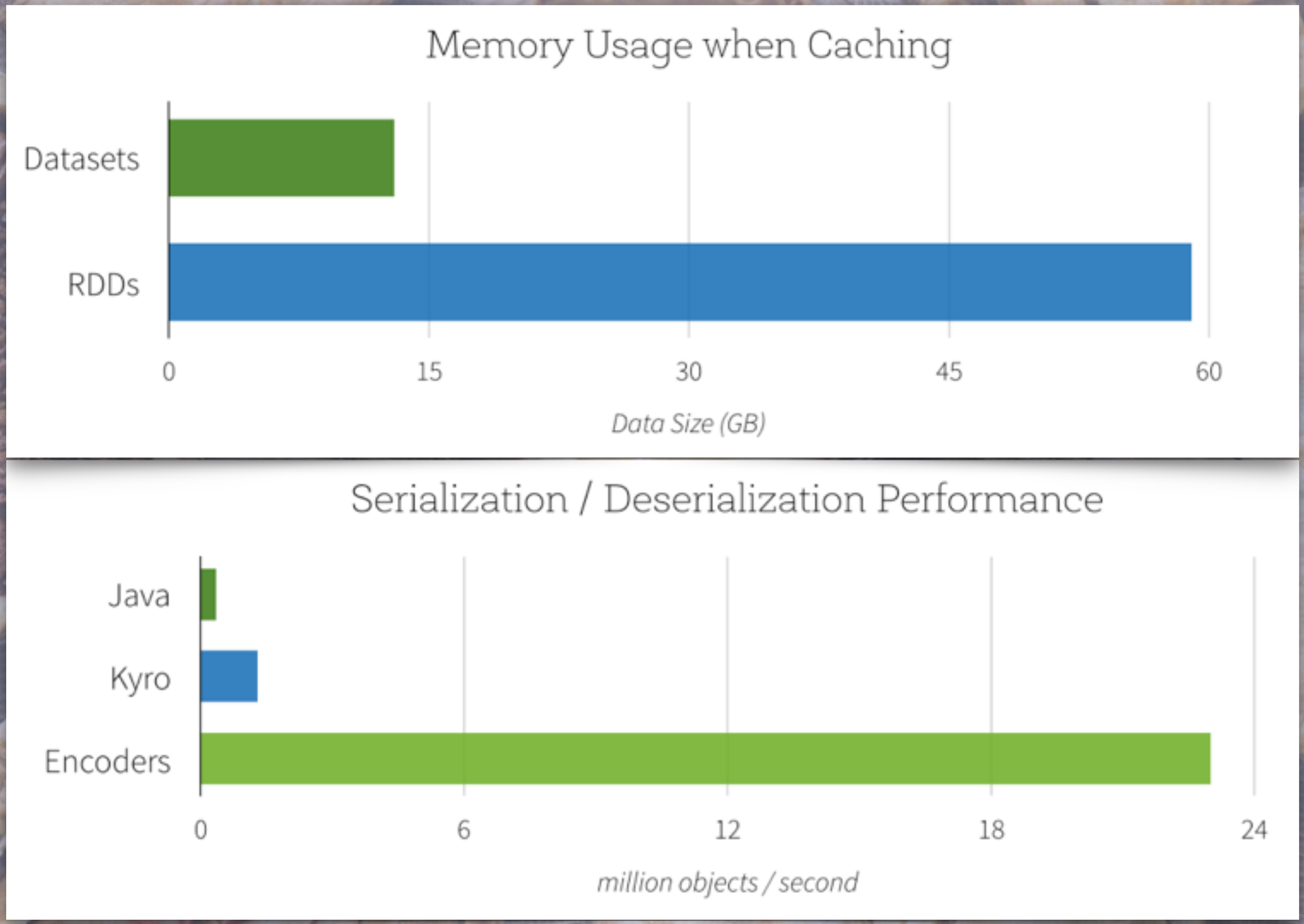
```
case class University(name: String, numStudents: Long, yearFounded: Long)

val schools = sqlContext.read.json("/schools.json").as[University]
```

O(PIES)

DATASETS

org.apache.spark.sql.Dataset[T]

Encoded



O(PIES)

DATASETS

org.apache.spark.sql.Dataset[T]

Strongly Typed

O(PIES)

DATASETS

org.apache.spark.sql.Dataset[T]

Strongly Typed

```
val schools = sqlContext.read.json("/schools.json").as[University]
```

```
schools: org.apache.spark.sql.Dataset[University]
```

```
1.  val bicents = schools.map(school => {
2.      school.yearFounded + 200
3.  }).withColumnRenamed("value", "bicentYear")
```

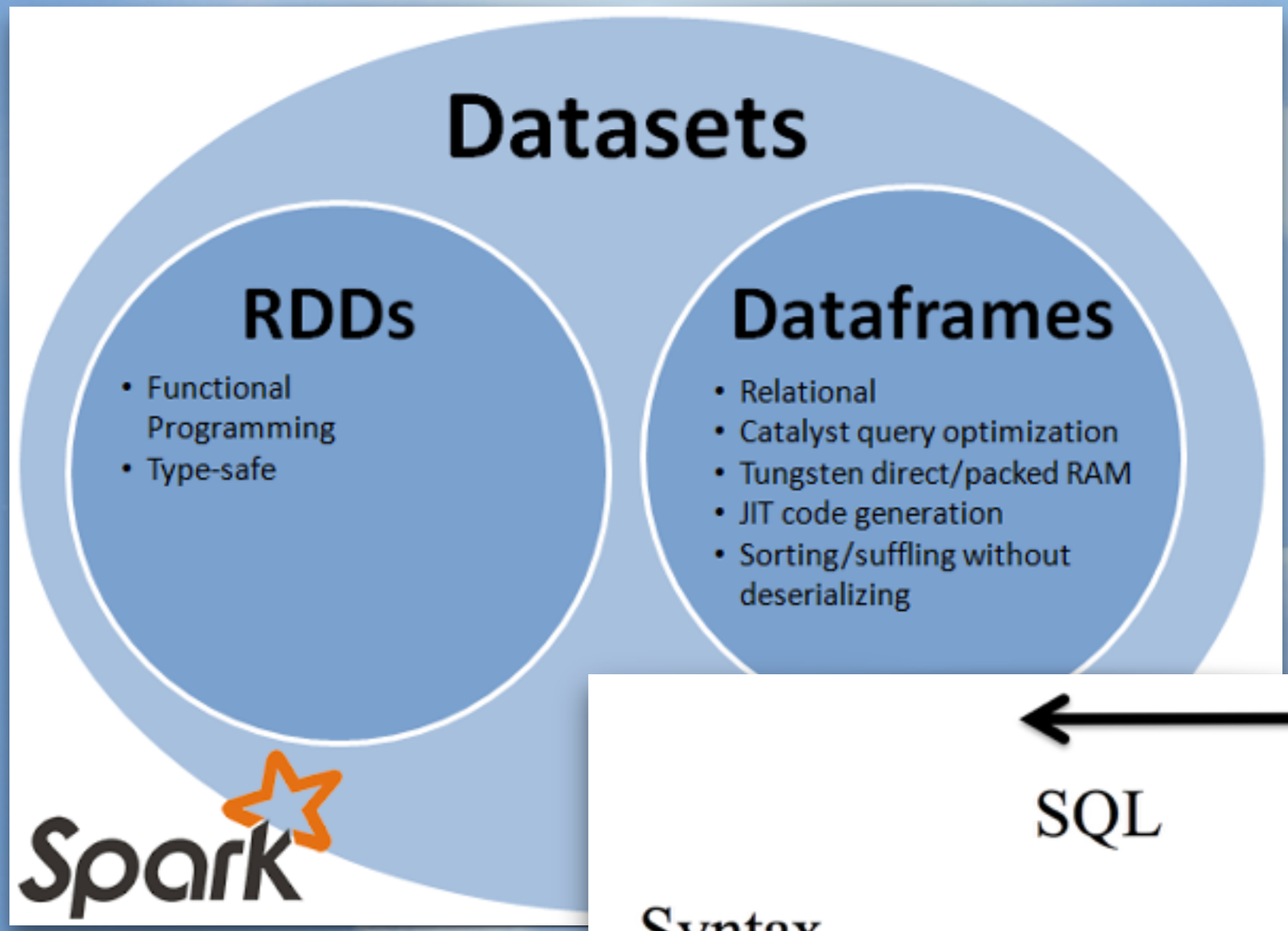
```
val awesomeSchools = schools.map(s => {
  val awesome = s.name match {
    case "WSU" => true
    case "MIT" => true
    case "UW" => false
    case _ => None
  }
  s.copy(isAwesome = awesome)
})
```

DataFrames

org.apache.spark.sql.Dataset[**Row**]

- *Named Columns*
- *Weakly typed Row representation*
- *Similar structures*
  - *Python/R DataFrame*
  - *Relational DB table*

DISTRIBUTED DATA STRUCTURES RECAP



	SQL	DataFrames	Datasets
Syntax Errors	Runtime	Compile Time	Compile Time
Analysis Errors	Runtime	Runtime	Compile Time

USE DATASET OR DATAFRAME?

org.apache.spark.sql.Dataset[**Row**]

DataFrame

- *Exploratory Data Analysis*
- *Transforming Columns*

Dataset

- *Known object requirements*
- *Strong typing in IDE*
- *Needs compilation errors*

CACHING

Caching best practices

- When
  - Fits in memory
  - Computationally expensive
- Broadcast hint
- Eager vs Lazy (default) cache

USING SQL

```
1  schools.createOrReplaceTempView("schools")
2  spark.sql("""SELECT SUM(students)
3  FROM schools
4  WHERE name IN ('WSU','MIT')""")
```

UDF Support

- Spark
- Hive

Create Permanent Hive table

```
schools.saveAsTable("entities.schools")
```

Access Hive/Temporary Table

```
spark.table("entities.schools").sum("students")
```

CREDITS

Databricks:

Datasets: https://databricks.com/blog/2016/01/04/introducing-apache-spark-datasets.html

Catalyst: https://www.slideshare.net/databricks/a-deep-dive-into-spark-sqls-catalyst-optimizer-with-yin-huai

https://de.slideshare.net/SparkSummit/deep-dive-into-catalyst-apache-spark-20s-optimizer-63071120

Tungsten: https://databricks.com/blog/2015/04/28/project-tungsten-bringing-spark-closer-to-bare-metal.html

Matrix: https://databricks.com/blog/2016/07/14/a-tale-of-three-apache-spark-apis-rdds-dataframes-and-datasets.html

General Reference

https://jaceklaskowski.gitbooks.io/mastering-apache-spark/

https://spark.apache.org/docs/latest/sql-programming-guide.html

Images

https://i.ytimg.com/vi/p8MEiZGjETE/maxresdefault.jpg

https://en.wikipedia.org/wiki/Donald_Trump

http://why-not-learn-something.blogspot.com/2016/07/apache-spark-rdd-vs-dataframe-vs-dataset.html

http://sueduff.com/wp-content/uploads/2014/04/bigstock-Magic-Book-9990005.jpg

https://seesouthernstars.files.wordpress.com/2015/06/porkey.jpg