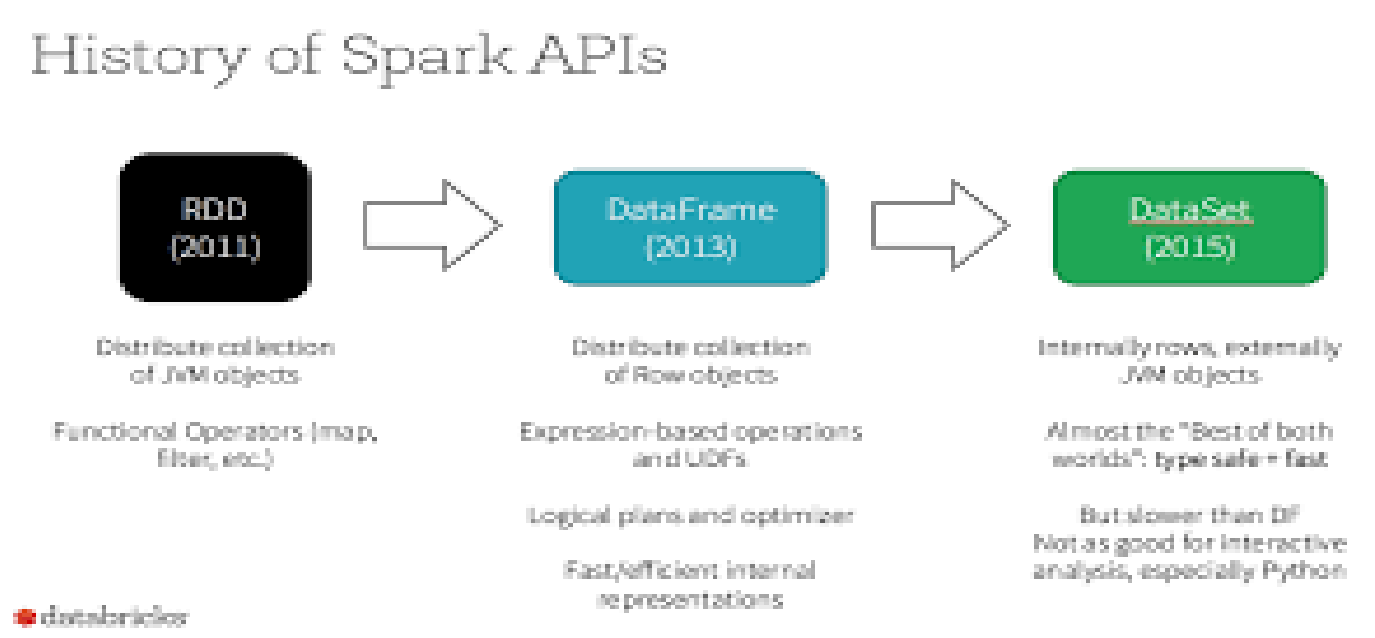
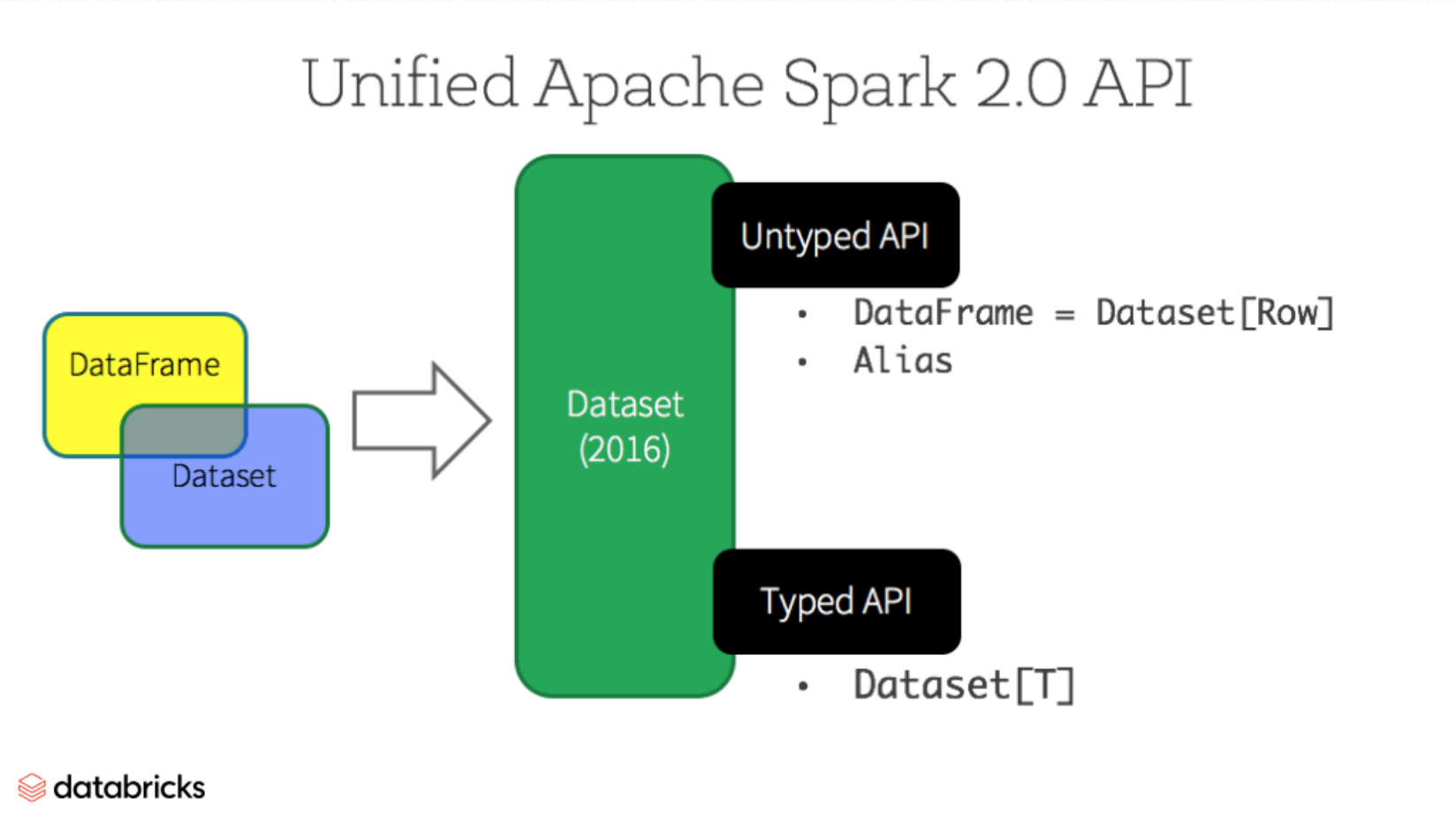
SPARK


History of Spark APIs
RDD (2011)
DataFrame (2013)
DataSet (2015)
Distribute collection of JVM objects
Functional Operators (map, filter, etc.)
Distribute collection of Row objects
Expression-based operations and UDFs
Logical plans and optimizer
Fast/efficient internal representations
Internally rows, externally JVM objects
Almost the "Best of both worlds": type safe + fast
But slower than DF
Not as good for interactive analysis, especially Python
databricks

Typed and Un-typed APIs



Language	Main Abstraction
Scala	Dataset[T] & DataFrame (alias for Dataset[Row])
Java	Dataset[T]
Python*	DataFrame
R*	DataFrame

Three APIs: DataFrames, Datasets, and RDDs

- Resilient Distributed Datasets
- DataFrames
- Datasets
- RDD or Resilient Distributed Datasets, is a collection of records with distributed computing, which are fault tolerant, immutable in nature. They can be operated on in parallel with low-level APIs, while their lazy feature makes the spark operation to work at an improved speed.

- RDDs support two types of operations:
  - Transformations - lazy operations that return another RDD, this RDD doesn't compute unless action is performed on it. Some examples of transformations are map(), flatmap(), filter()

  - Actions - operations that trigger computation and return values. Some examples of actions are count, top(), savetofile()

DataFrames

- DataFrames is a distributed collection of rows under named columns.
- In simple terms, it looks like an Excel sheet with Column headers, or you can think of it as the equivalent to a table in a relational database or a DataFrame in R or Python.
- It has three main common characteristics with RDD:
  - Immutable in nature:
    - You will be able to create a DataFrame but you will not be able to change it. A DataFrame just like an RDD can be transformed
  - Lazy Evaluations:
    - a task is not executed until an action is performed.
  - Distributed:
    - DataFrames just like RDDs are both distributed in nature.

Ways to Create a DataFrame

- In Spark DataFrames can be created in several ways:
  - Using different data formats. Such as loading the data from JSON, CSV, RDBMS, XML or Parquet
  - Loading the data from an already existing RDD.
  - Programmatically specifying schema

- # Constructs a DataFrame from the users table in Hive.
- users = context.table("users")

- # from JSON files in S3
- logs = context.load("s3n://path/to/data.json", "json")

- Convert RDD to DataFrame – Using toDF()
- Spark provides an implicit function toDF() which would be used to convert RDD, Seq[T], List[T] to DataFrame.
- In order to use toDF() function, import implicits first using import spark.implicits._.

- val dfFromRDD1 = rdd.toDF()
- dfFromRDD1.printSchema()

```
import spark.implicits._
val columns = Seq("language","users_count")
val data = Seq(("Java", "20000"), ("Python", "100000"),
("Scala", "3000"))
val rdd = spark.sparkContext.parallelize(data)
val dfFromRDD1 = rdd.toDF()
dfFromRDD1.printSchema()
```

Use DataFrames?

- Once built, DataFrames provide a domain-specific language for distributed data manipulation. Here is an example of using DataFrames to manipulate the demographic data of a large population of users:
- # Create a new DataFrame that contains “young users” only
- young = users.filter(users.age < 21)

- # Alternatively, using Pandas-like syntax
- young = users[users.age < 21]

- # Increment everybody’s age by 1
- young.select(young.name, young.age + 1)

- # Count the number of young users by gender
- young.groupBy("gender").count()

- # Join young users with another DataFrame called logs
- young.join(logs, logs.userId == users.userId, "left_outer")

- You can also incorporate SQL while working with DataFrames, using Spark SQL. This example counts the number of users in the young DataFrame.
- young.registerTempTable("young")
- context.sql("SELECT count(*) FROM young")

Datasets

- Dataset is a data structure in SparkSQL which is strongly typed and is a map to a relational schema.
- It represents structured queries with encoders.
- It is an extension to data frame API. Spark Dataset provides both type safety and object-oriented programming interface.
- A Dataset is a strongly-typed, immutable collection of objects that are mapped to a relational schema.
- A Dataset can be created using JVM objects and manipulated using complex functional transformations.
- Datasets can be created in two ways:
  - Dynamically
  - Reading from a JSON file using SparkSession.