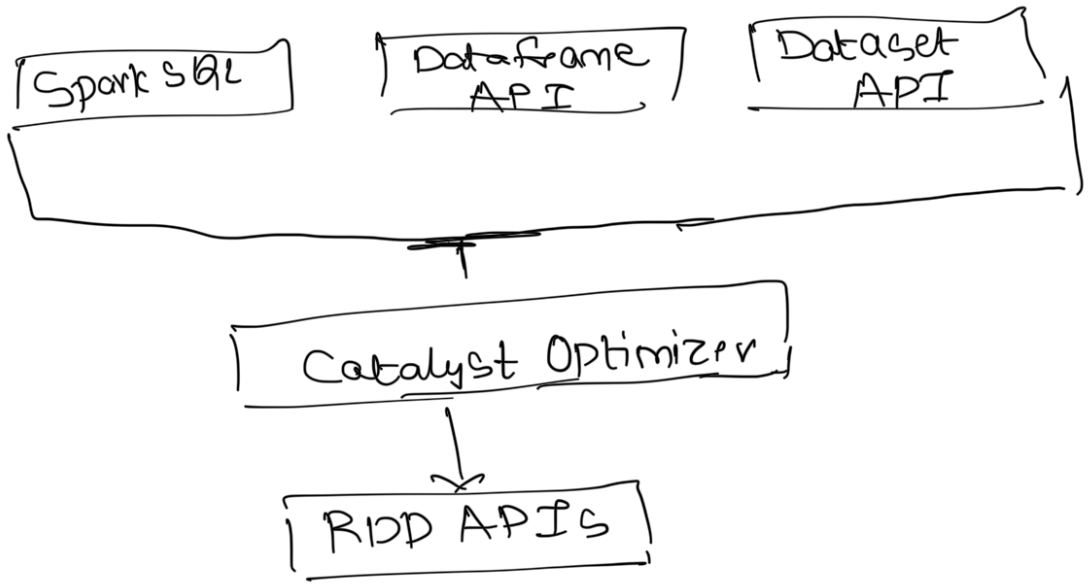
## * Spark API

- Apache Spark started with the goal of simplifying and improving the Hadoop Map/Reduce programming model. To achieve this goal spark came up with the idea of
  RDD - Resilient Distributed Data
- Spark didn't stop at RDD, and came up with the higher level of APIs such as Dataset APIs and Dataframe APIs.




- The RDD is at the core, and use to develop your application

So

- The next one is catalyst optimizer. We write our code using Spark SQL, DataFrame APIs and Dataset APIs. This code is then submitted to Spark for execution.

However the code passes through the Catalyst Optimizer, which decides how it should be executed and lays out an execution plan.

## * Spark RDDs API:

- An RDD is a dataset. That means they are nothing but a datastructure to hold your data records.
- They are similar to Dataframe, but unlike Dataframes, RDD records are language-native objects, and they do not have a row/column structure and a schema.
- You can create an RDD reading your data from a file. However, RDD is internally broken down into partitions to form a distributed collection, same as dataframes.
- They are partitioned and spread across the executor cores so they can be processed in

parallel.

— RDDs are resilient, that means they are Fault-tolerent. RDDs are fault tolerant because they also store information about how they are created.

⇒ what does it mean?

Let's assume an RDD partition is assigned to an executor core for processing it.

In some time, the executor fails or crashes That's a fault, and you could lose your RDD partition, right?

However, the driver will notice the failure and assign the same RDD partition to another executor core. The new executor core will reload the RDD partition and start the processing.

And this can be done easily, because each RDD partition comes with the information about how to create it and how to process it. And that's why we call them resilient.

— That means an RDD partition can be recreated and reprocessed anywhere in the

cluster.

- In general RDD is similar to Dataframes, but they lack a row/column structure and the schema.
- The data reader APIs in the RDD were raw and fundamental. They didn't allow you to work with commonly used files such as CSV, JSON, parquet and Avro. They have methods to read a text file, binary file, sequence file, Hadoop file, and object file.
- The idea of Transformations and Actions are the same for RDDs. However, RDDs offered only basic transformations such as map(), reduce(), filter(), foreach(), etc. Most of the RDD transformations were designed to accept lambda function, and simply apply your code to RDD.
- So, basically RDD API leaves all the responsibility in the developer's hand. You need to take care of giving a structure to your data, implement your operations, create an optimized data structure, slight, etc.

compress your objects

* Spark Engine :-

The Spark SQL Engine is a powerful compiler that optimizes your code and also generates efficient Java Bytecode.

The overall effort of the Spark SQL engine can be broken down into Four phases.

- First phase : Analysis :

In this phase spark SQL engine will read your code and generate an Abstract syntax Tree for your SQL or Dataframe queries

In this phase, your code is analyzed, and the column names, table, or view names, SQL functions are resolved.

You might get an runtime error shown as an analysis error at this stage.

- Second phase : Logical optimization

In this phase, SQL engine will apply rule based optimization and construct a set of multiple execution plans.

In the catalyst optimizer will use

Then the Catalyst ...

a cost based optimization to assign a cost to each plan.

- Third phase :- Physical planning :

In this phase the SQL engine picks the most effective logical plan and generates a physical plan.

The physical plan is nothing but set of RDD operations, which determines how the plan is going to execute on Spark cluster.

- Fourth phase : Code generation

This phase involves generating efficient Java bytecode to run on each machine.