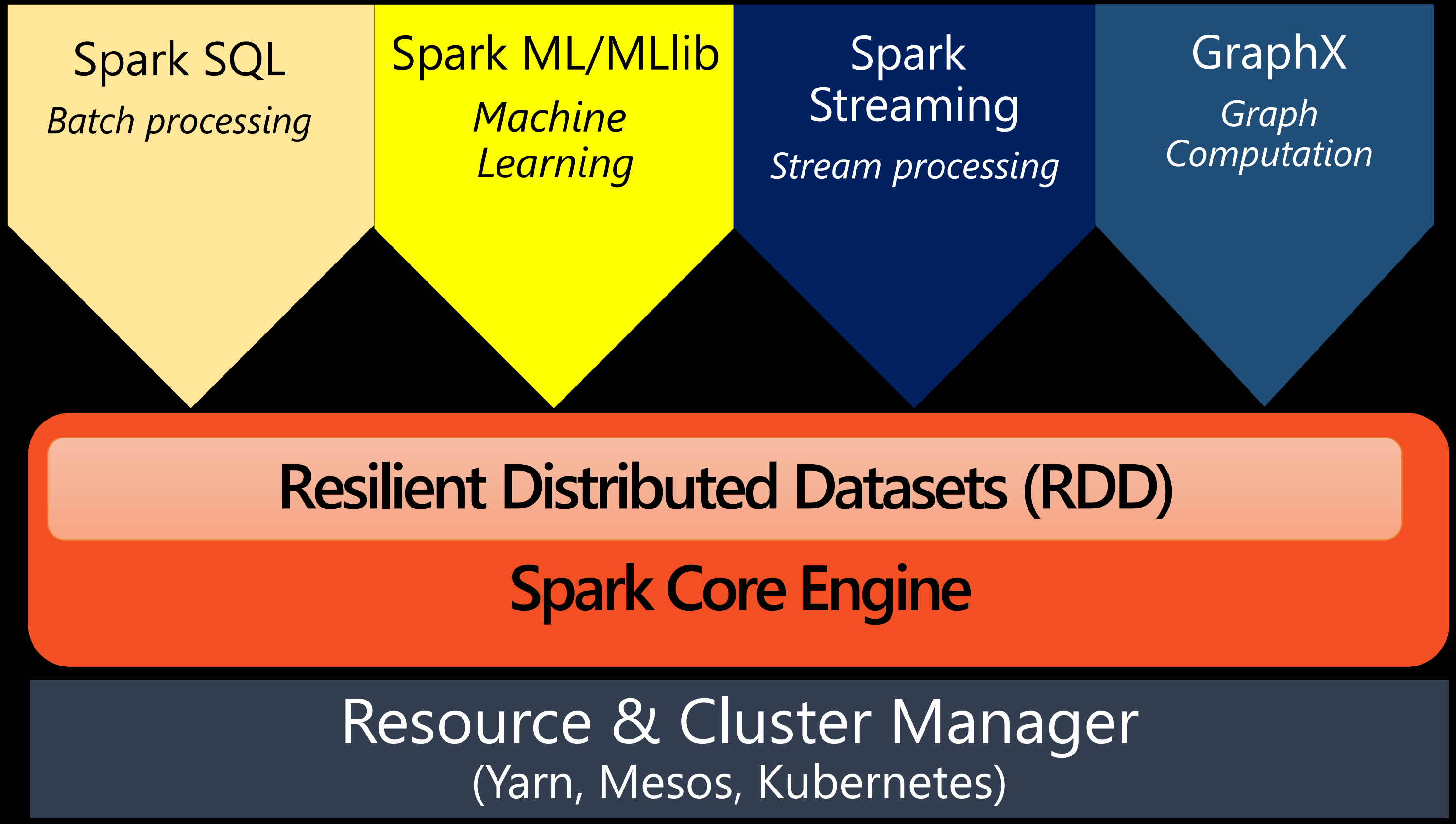
Aas Trailblazers

Unleashing insights

Apache Spark – initial version

A unified, open source, parallel, data processing framework for Big Data Analytics



Spark SQL

Batch processing

Spark ML/MLlib

Machine Learning

Spark Streaming

Stream processing

GraphX

Graph Computation

Resilient Distributed Datasets (RDD)

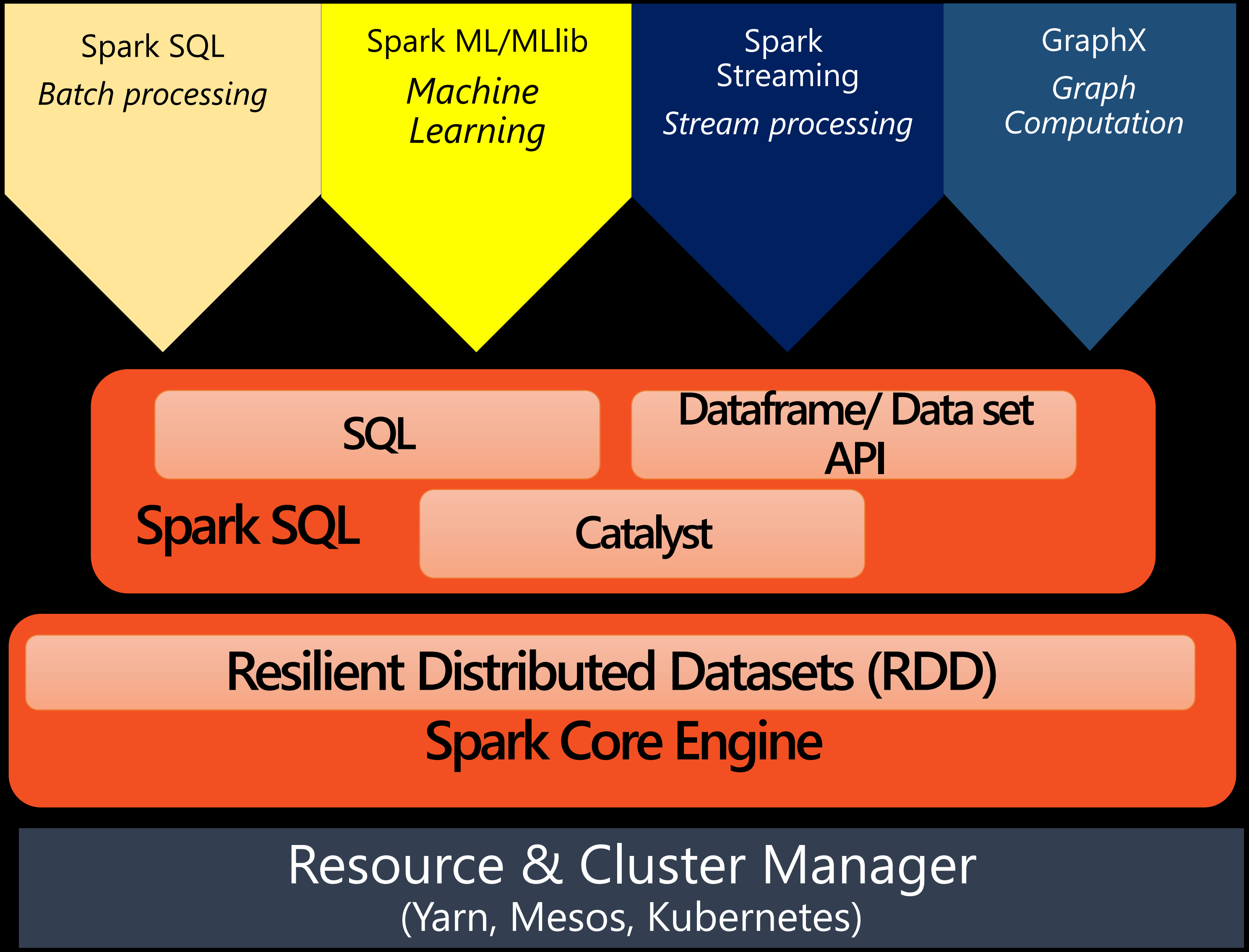
Spark Core Engine

Resource & Cluster Manager

(Yarn, Mesos, Kubernetes)

http://spark.apache.org

Apache Spark – with catalyst optimizer



Spark SQL

Batch processing

Spark ML/MLlib

Machine Learning

Spark Streaming

Stream processing

GraphX

Graph Computation

SQL

Dataframe/ Data set API

Spark SQL

Catalyst

Resilient Distributed Datasets (RDD)

Spark Core Engine

Resource & Cluster Manager

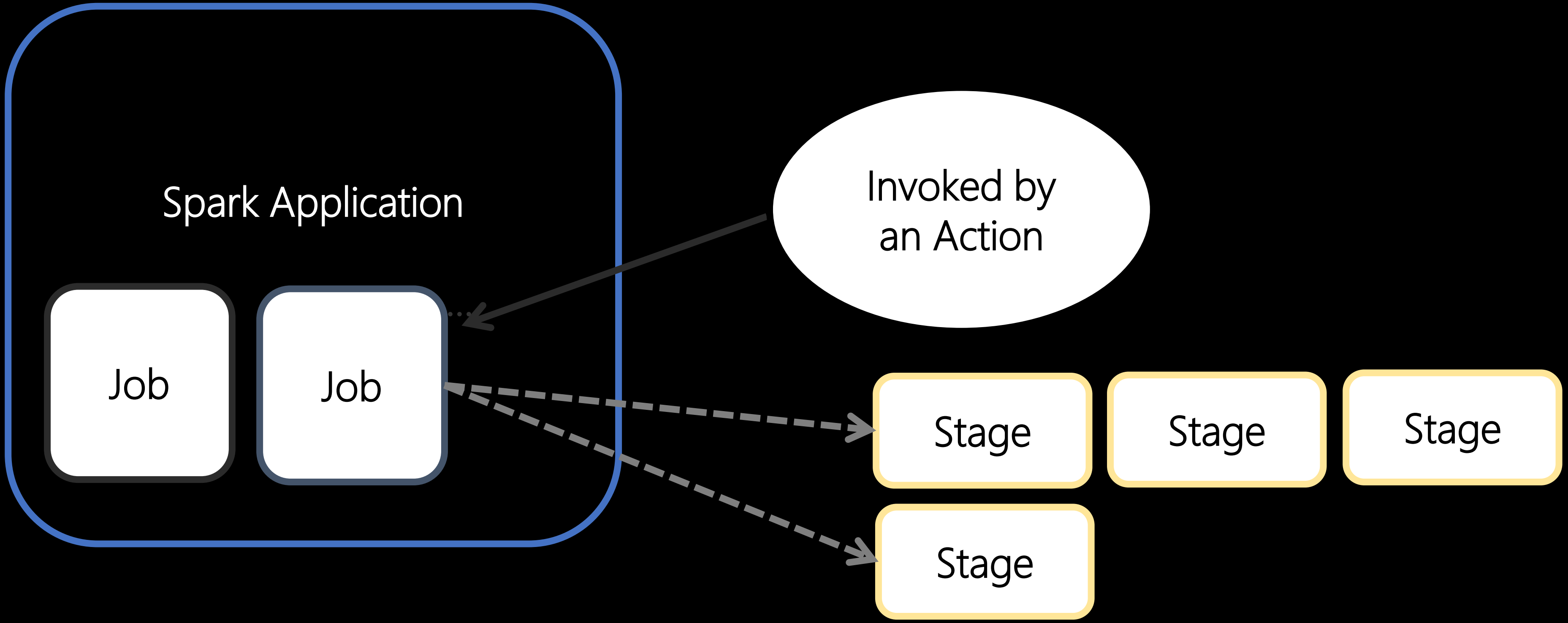
(Yarn, Mesos, Kubernetes)

http://spark.apache.org

Physical Execution of Job

- When you submit a Spark Job, the program gets executed in stages

Application -> Jobs -> Stages -> Tasks



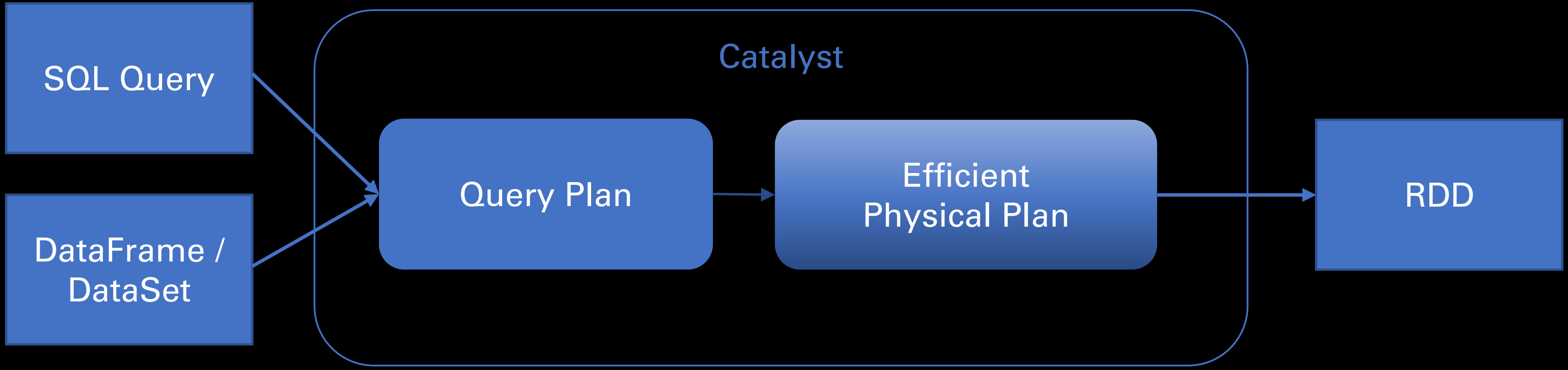
Apache Spark SQL: Dataframe/Dataset API

```
data
    .groupBy ($“a”)
    .agg ( avg ($”b”))
    .filter ($”a” > 10)
```

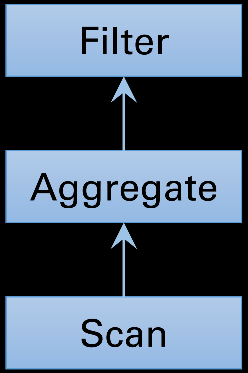
```
dataRDD.map {
    case (a, b) => a -> (b, 1)
}.reduceByKey {
    case ((b1, c1), (b2, c2)) => (b1+ b2, c1+c2)
}.map {
    case (a, b, c) => (a, b / c)
}.filter {
    case (a, b) => a > 10
}
```

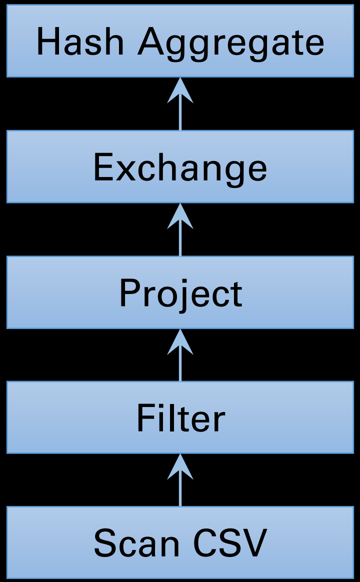
- Catalyst Optimizer – finds the efficient execution plan for a given dataset/sql query.

Catalyst Optimizer

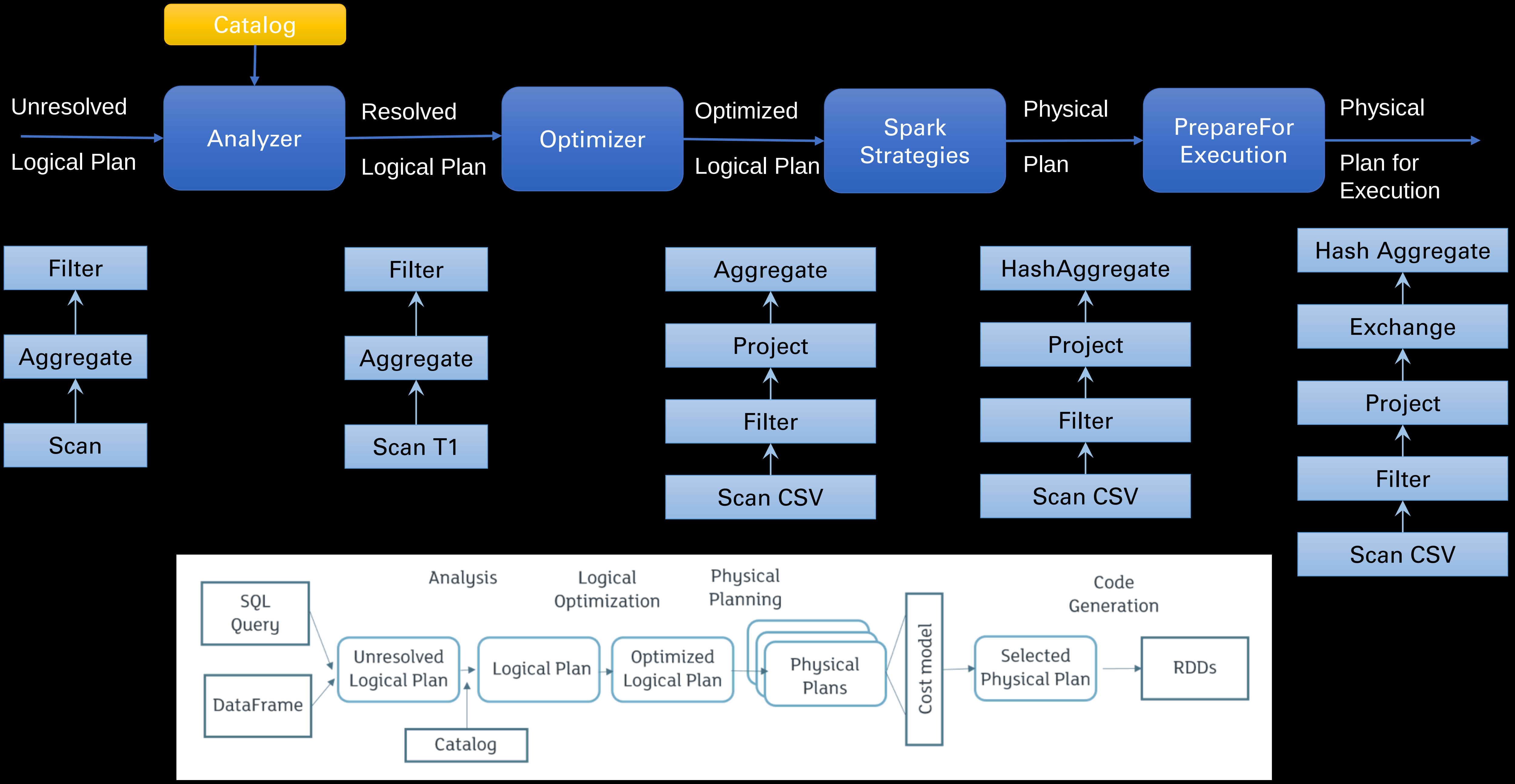


```
data
    .groupBy ($“a”)
    .agg ( avg ($”b”))
    .filter ($”a” > 10)
```





Catalyst Optimizer Stages



Catalog

Unresolved Logical Plan

Analyzer

Resolved Logical Plan

Optimizer

Optimized Logical Plan

Spark Strategies

Physical Plan

PrepareFor Execution

Physical Plan for Execution

Filter
Aggregate
Scan

Filter
Aggregate
Scan T1

Aggregate
Project
Filter
Scan CSV

HashAggregate
Project
Filter
Scan CSV

Hash Aggregate
Exchange
Project
Filter
Scan CSV

SQL Query

DataFrame

Analysis

Unresolved Logical Plan

Catalog

Logical Optimization

Logical Plan

Optimized Logical Plan

Physical Planning

Physical Plans

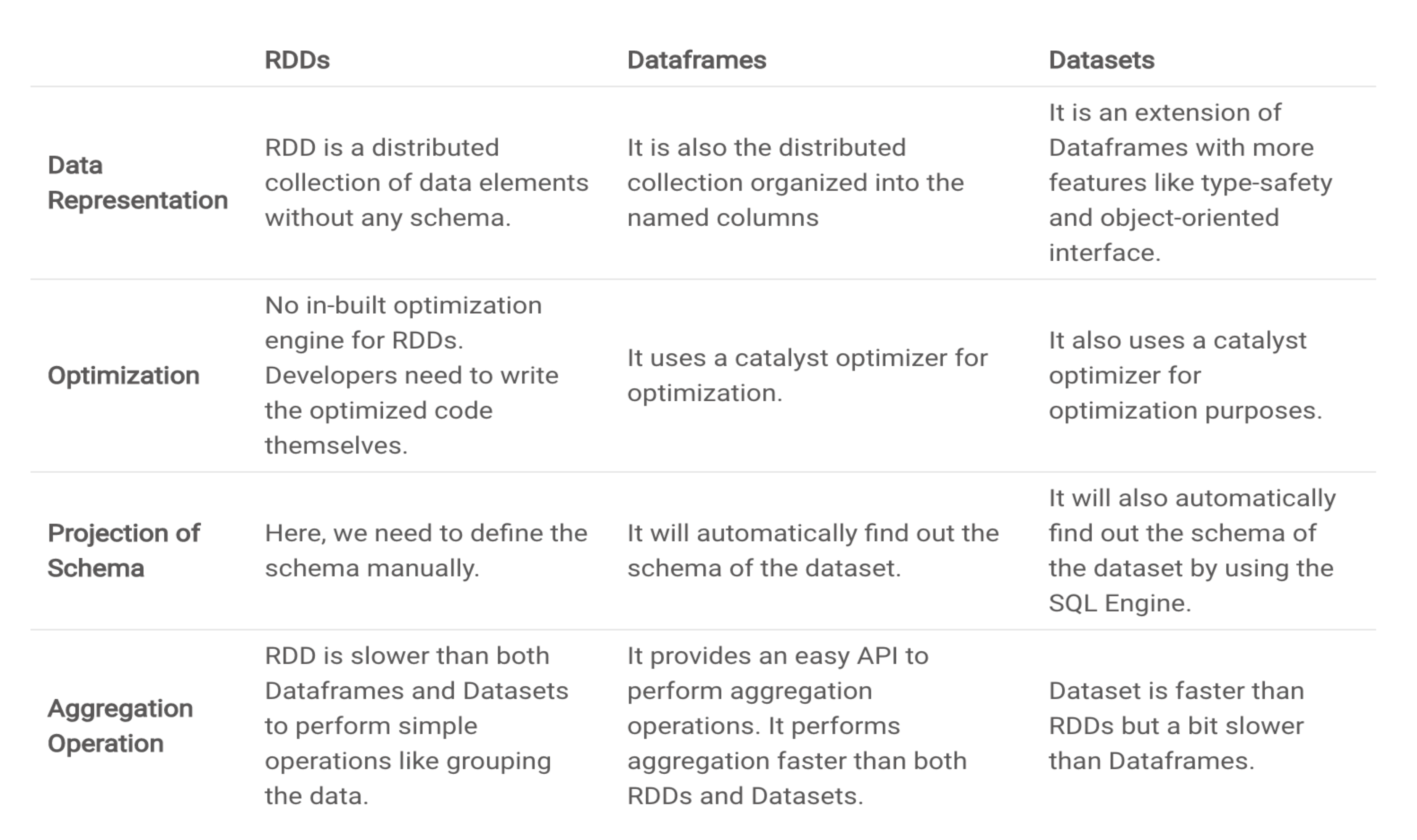
Cost model

Selected Physical Plan

Code Generation

RDDs

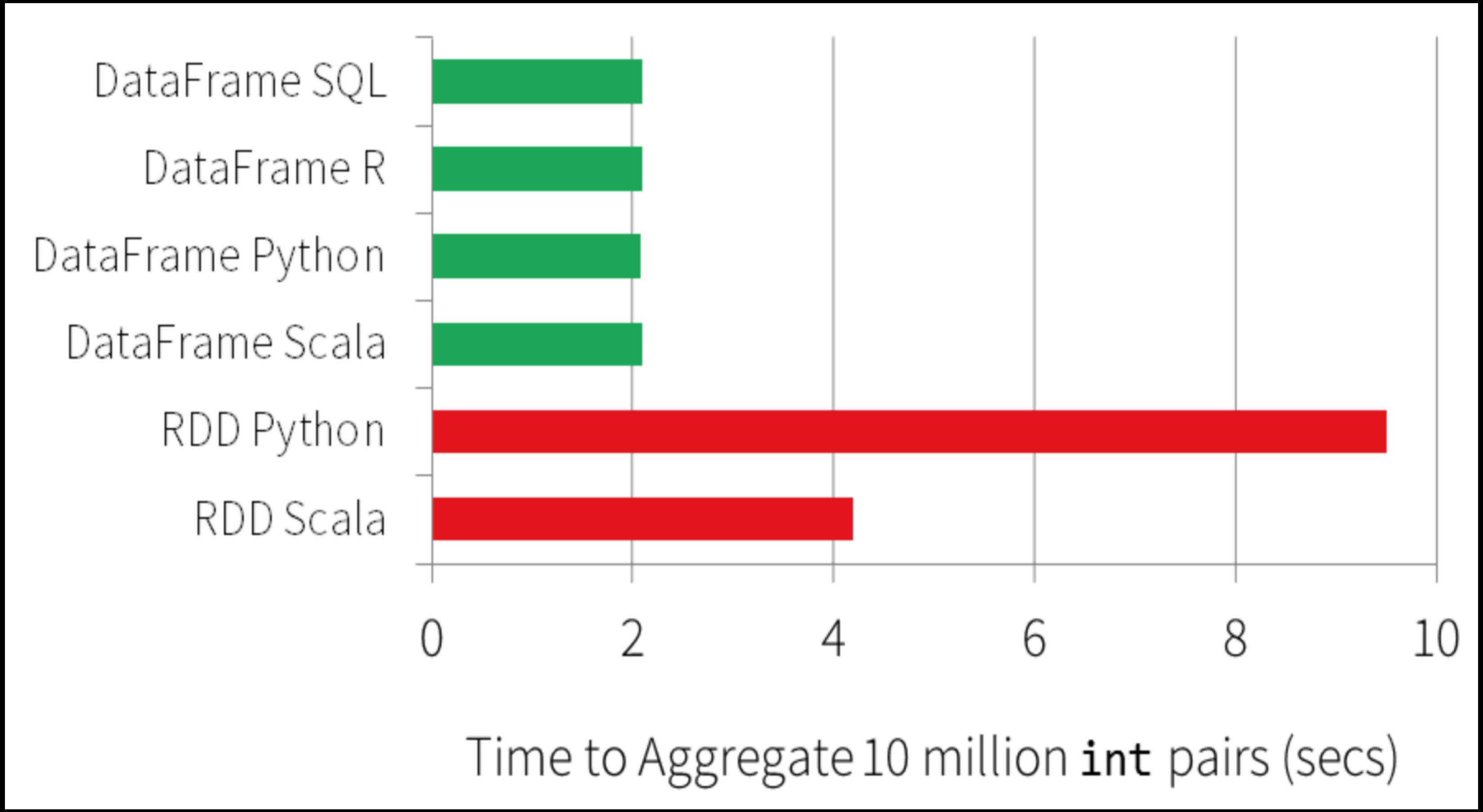
RDDs vs Dataframe vs Dataset API



	RDDs	Dataframes	Datasets
Data Representation	RDD is a distributed collection of data elements without any schema.	It is also the distributed collection organized into the named columns	It is an extension of Dataframes with more features like type-safety and object-oriented interface.
Optimization	No in-built optimization engine for RDDs. Developers need to write the optimized code themselves.	It uses a catalyst optimizer for optimization.	It also uses a catalyst optimizer for optimization purposes.
Projection of Schema	Here, we need to define the schema manually.	It will automatically find out the schema of the dataset.	It will also automatically find out the schema of the dataset by using the SQL Engine.
Aggregation Operation	RDD is slower than both Dataframes and Datasets to perform simple operations like grouping the data.	It provides an easy API to perform aggregation operations. It performs aggregation faster than both RDDs and Datasets.	Dataset is faster than RDDs but a bit slower than Dataframes.

https://docs.microsoft.com/en-us/azure/hdinsight/spark/optimize-data-storage#choose-data-abstraction

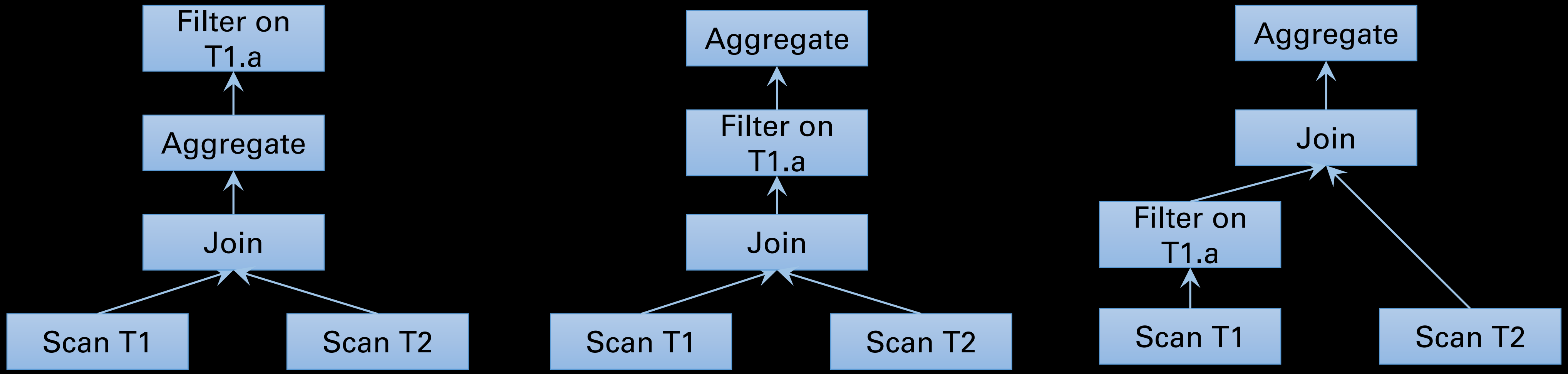
RDD vs Dataframe – Performance



Apache Spark DataFrames for Large Scale Data Science (databricks.com)

Rule executor

- Rules are defined as Sequence of Batches of rules.
  - Batches in sequence are applied one after another.
  - Rules within a Batch are applied continuously till FixedPoint



Filter on T1.a
Aggregate
Join
Scan T1
Scan T2

Aggregate
Filter on T1.a
Join
Scan T1
Scan T2

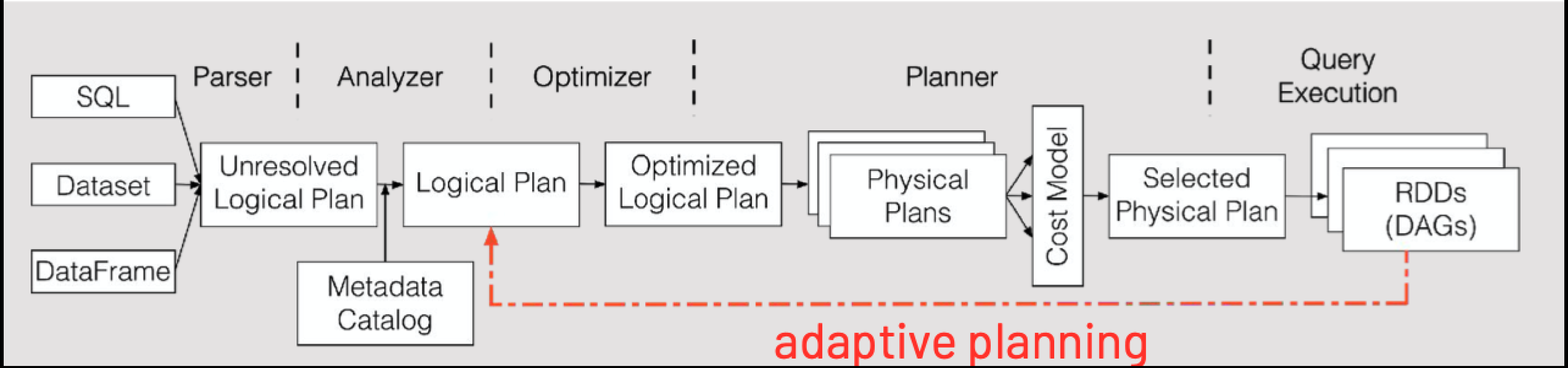
Aggregate
Join
Filter on T1.a
Scan T1
Scan T2

Adaptive Query Execution (AQE)

With each major release of Spark, it's been introducing a new optimization features in order to better execute the query to achieve the greater performance.

- **Spark 1.x** – Introduced Catalyst Optimizer and Tungsten Execution Engine
- **Spark 2.x** – Added Cost-Based Optimizer
  - **Single-pass optimization** by creating an execution plan (set of rules), once execution starts it sticks with the plan and starts executing the rules it created in the plan and doesn't do any further optimization
- **Spark 3.0** – Now added Adaptive Query Execution
  - Reoptimizes and adjusts query plans based on runtime metrics collected during the execution of the query, this re-optimization of the execution plan happens after each stage of the query as stage gives the right place to do re-optimization.

After enabling Adaptive Query Execution, Spark performs Logical Optimization, Physical Planning, and Cost model to pick the best physical.
By doing the re-plan with each Stage, Spark 3.0 performs 2x improvement on TPC-DS over Spark 2.4.

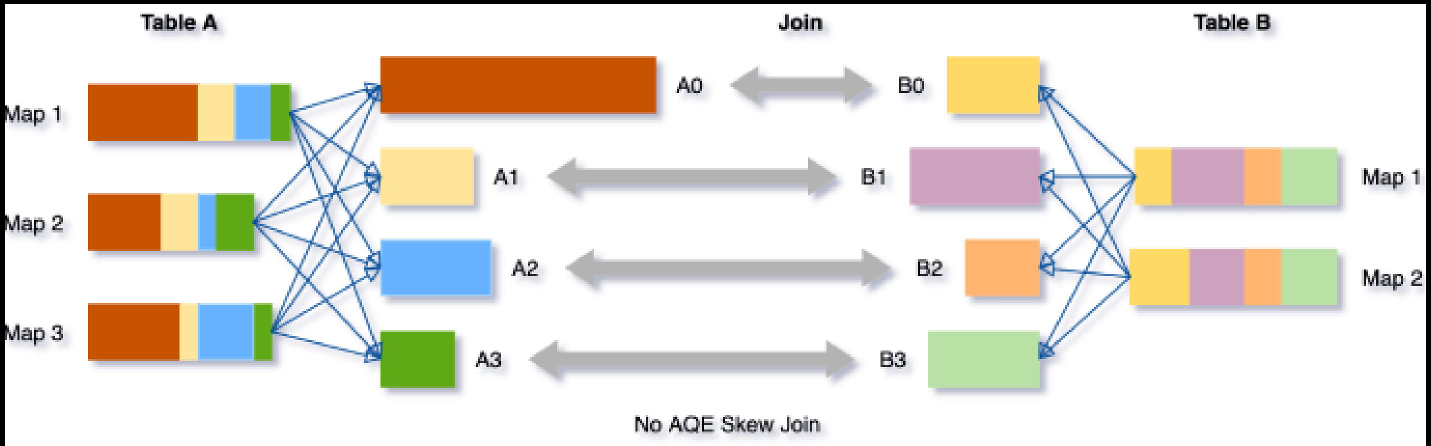


Adaptive Query Execution

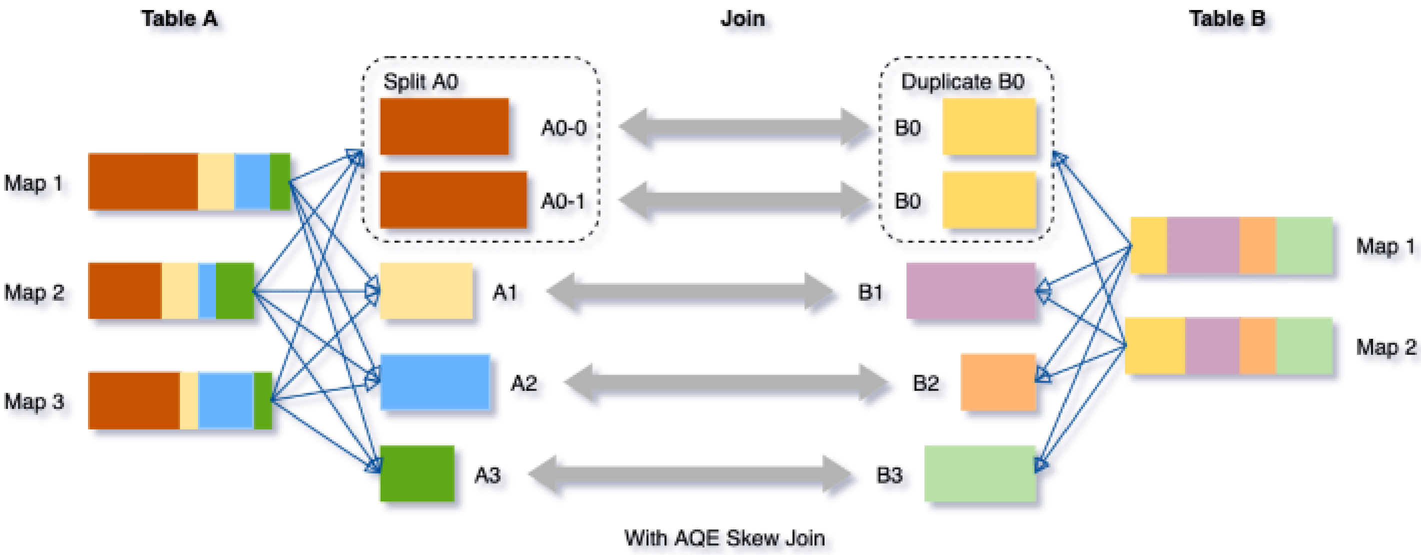
- Compute statistics at runtime and update plan if needed.
    - At the end of each stage, get the actual estimates.
    - Try to re-optimize remaining plan based on new stats.
- Few useful optimizations:
    - Changing join type
    - Handling partition skew
    - Coalescing shuffle partition

https://spark.apache.org/docs/latest/sql-performance-tuning.html#adaptive-query-execution

Adaptive Query Execution



No AQE Skew Join



With AQE Skew Join

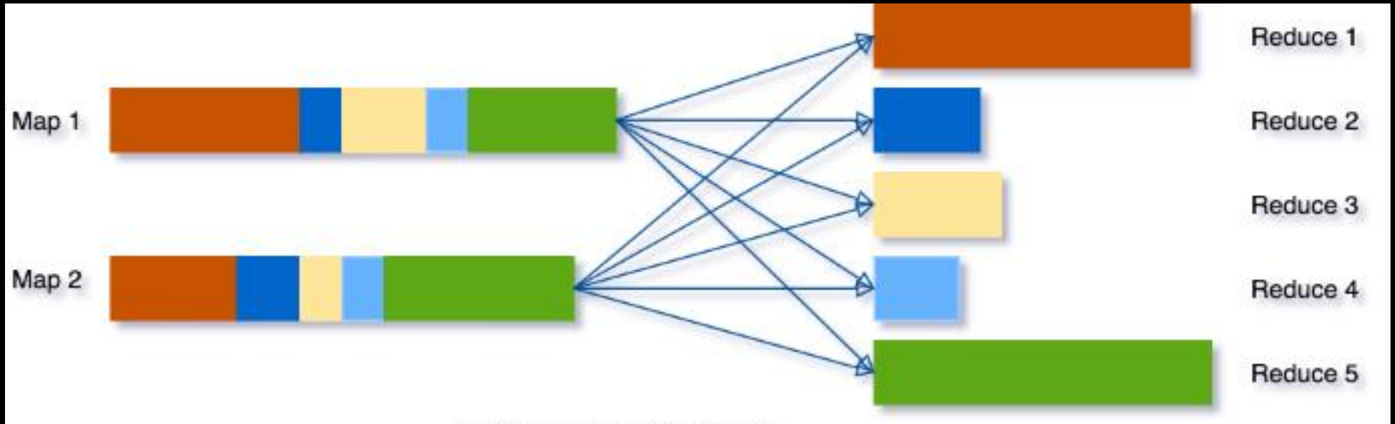
How to Speed up SQL Queries with Adaptive Query Execution (databricks.com)

Adaptive Query Execution

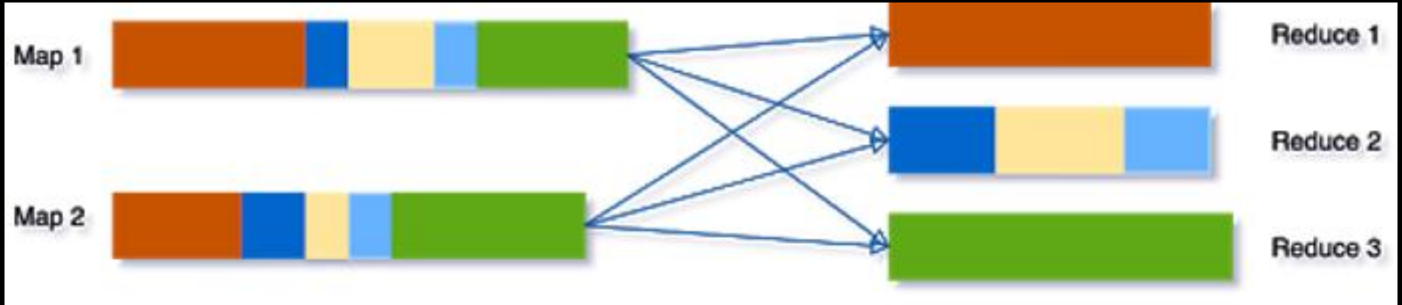
- Compute statistics at runtime and update plan if needed.
    - At the end of each stage, get the actual estimates.
    - Try to re-optimize remaining plan based on new stats.
- Few useful optimizations:
    - Changing join type
    - Handling partition skew
    - Coalescing shuffle partition

https://spark.apache.org/docs/latest/sql-performance-tuning.html#adaptive-query-execution

Adaptive Query Execution



No AQE Partition Coalescing



With AQE Partition Coalescing

[How to Speed up SQL Queries with Adaptive Query Execution (databricks.com)]

Demo