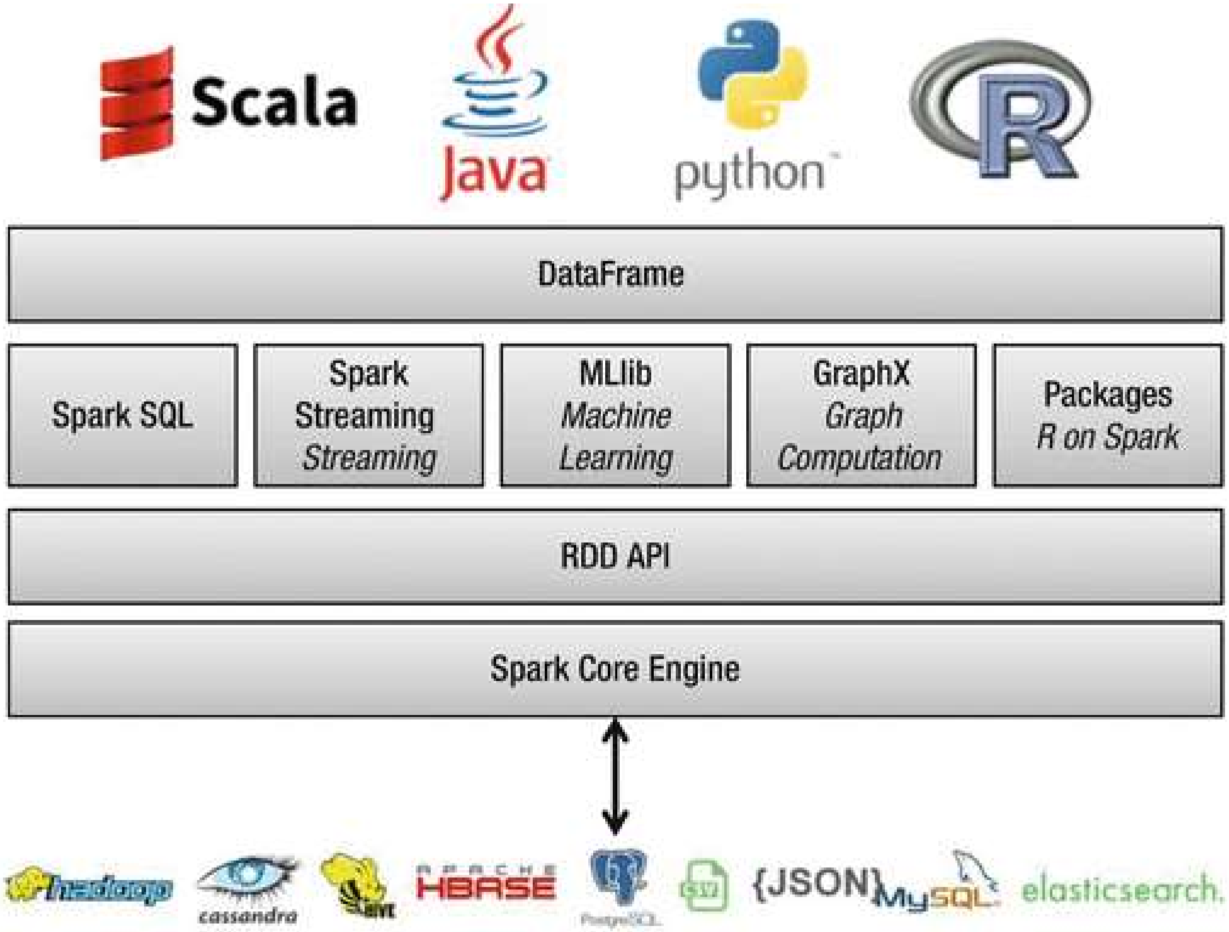
Apache Sparks Diagrams



DataFrame

Spark SQL

Spark Streaming *Streaming*

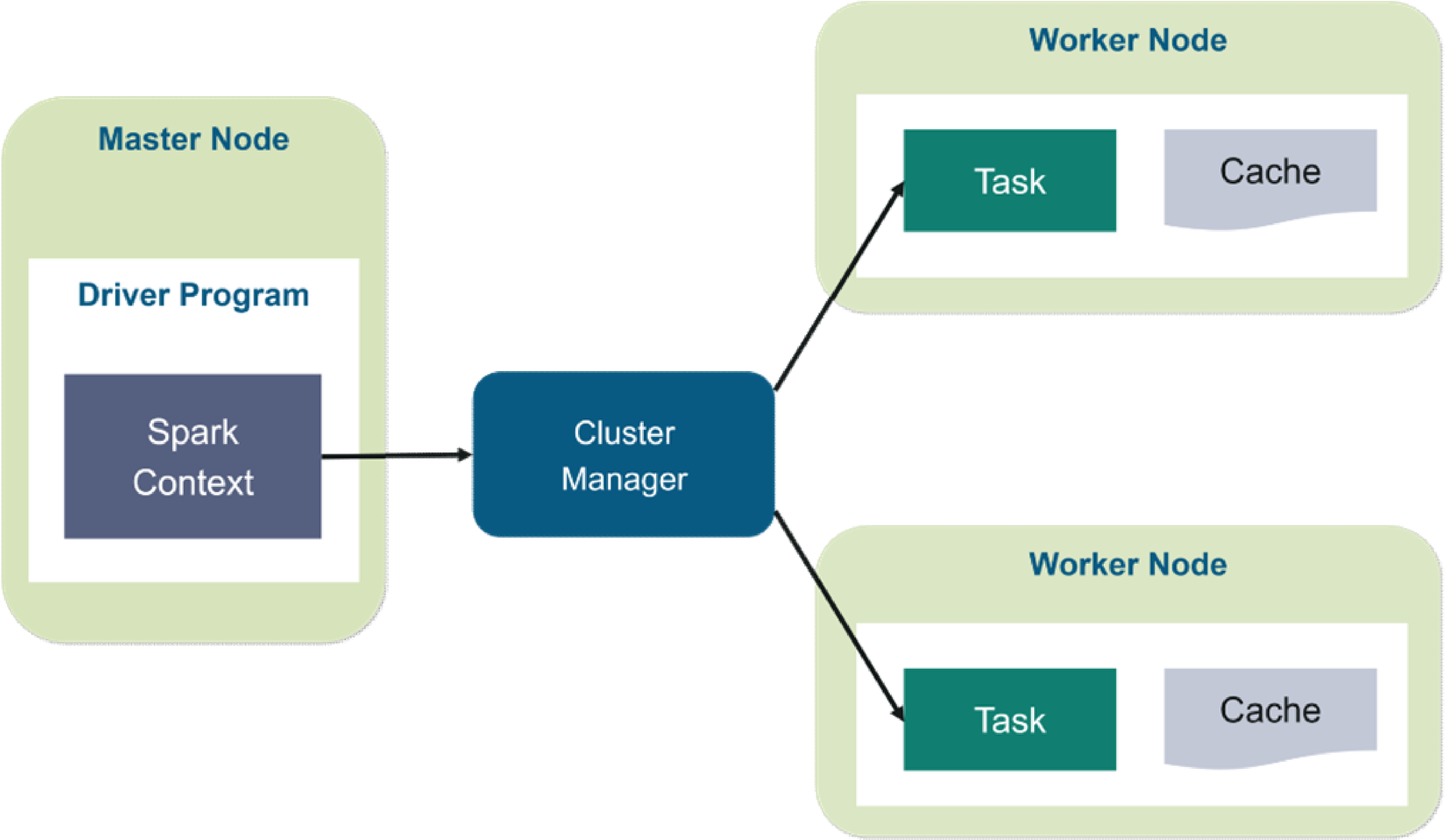
MLlib *Machine Learning*

GraphX *Graph Computation*

Packages *R on Spark*

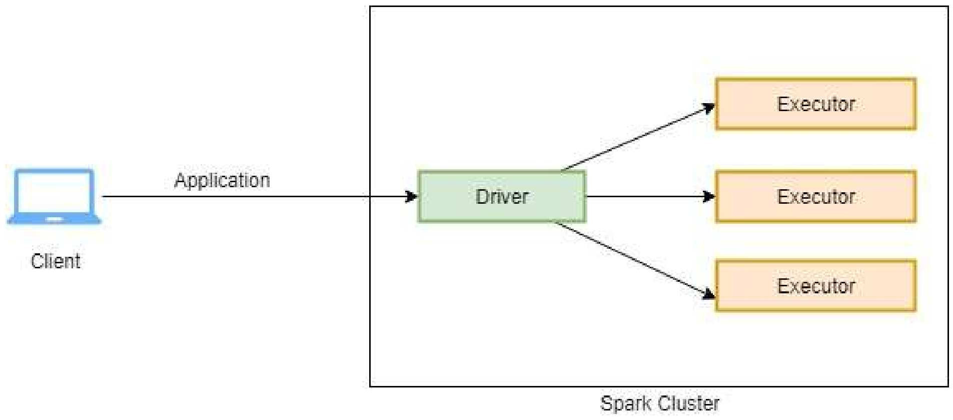
RDD API

Spark Core Engine

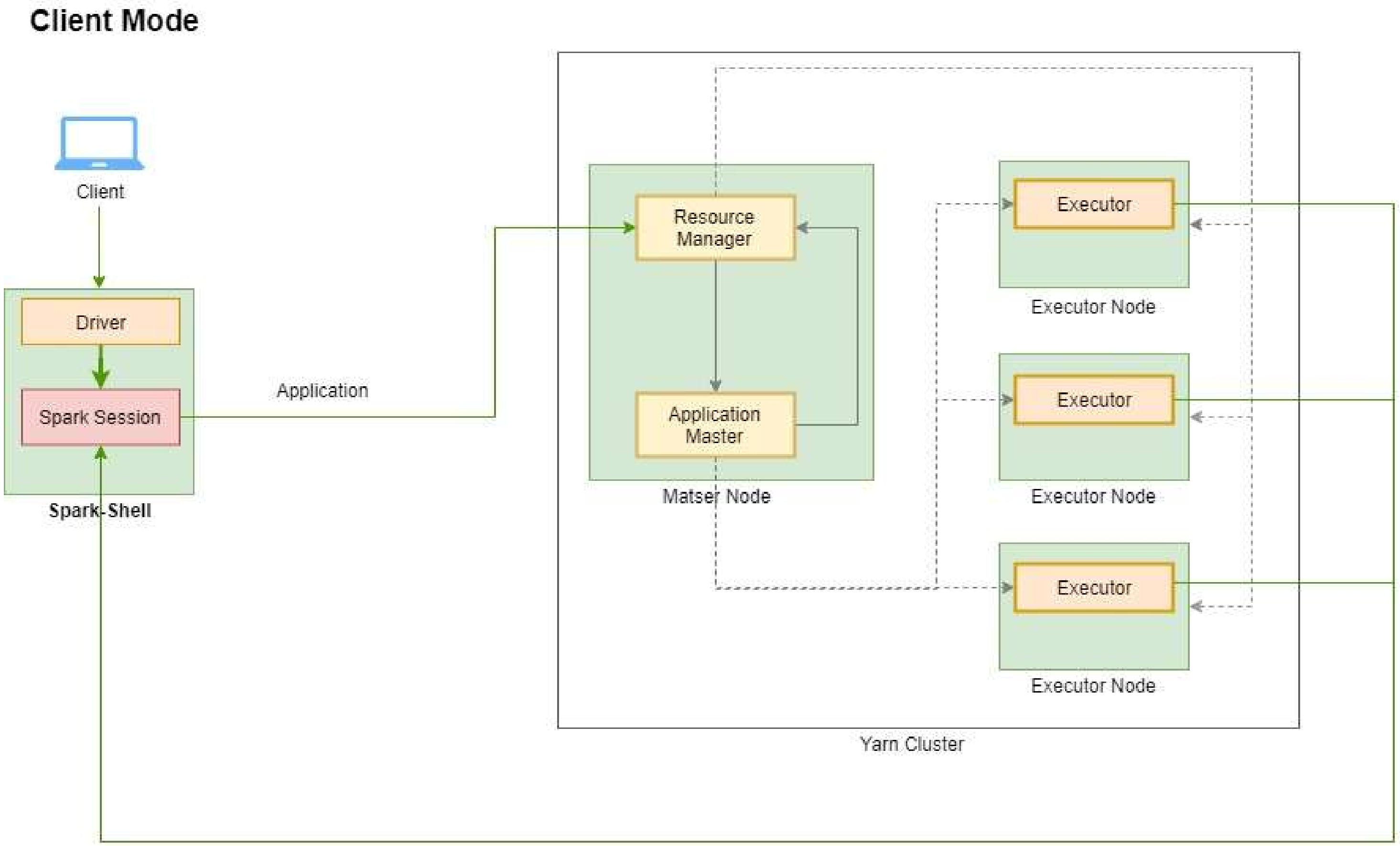

Master Node
Driver Program
Spark
Context
Cluster
Manager
Worker Node
Task
Cache
Worker Node
Task
Cache

Apache Spark Execution

- For every application submitted on spark cluster spark creates a dedicated Driver process and bunch of Executor processes.
- Driver process is responsible for analyzing, distributing, scheduling and monitoring of executor processes.
- Whereas the executor process is only responsible for running the task they were assigned by drivers and reporting the status back to the driver.

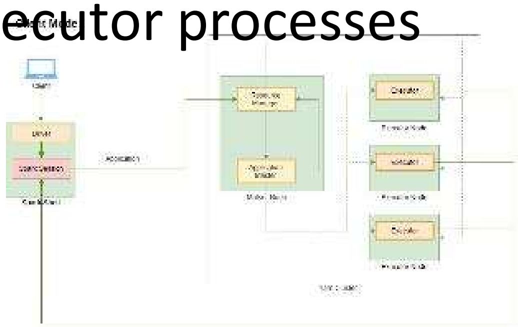


Apache Spark Execution – Client Mode

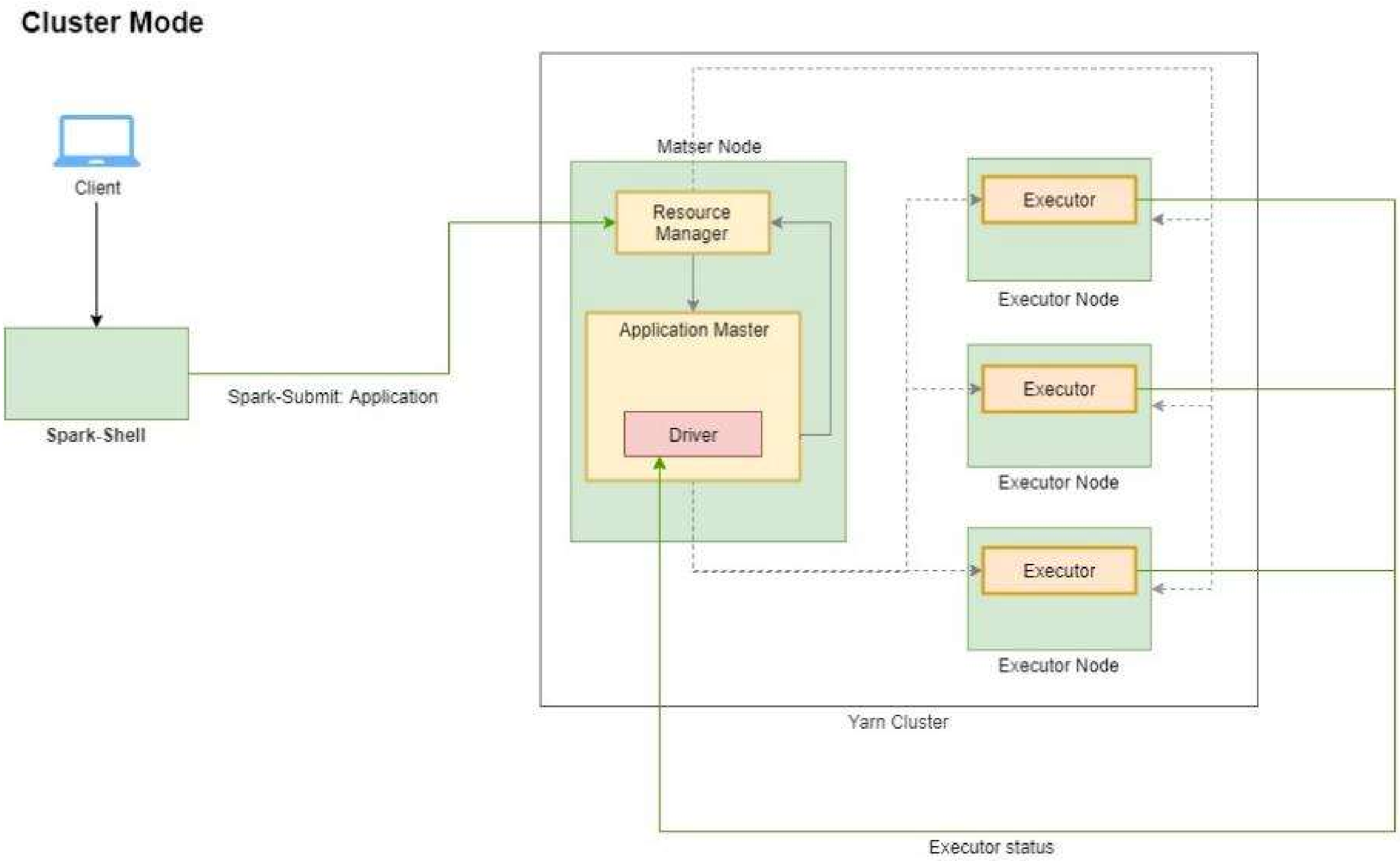


Apache Spark Execution – Client Mode

1. The spark driver is created within the spark-client (i.e. spark-shell)
2. Driver then connects with the Yarn Resource Manager to initiate the application.
3. The yarn resource manager then Spawns a Application Manager (AM container).
4. Application container then contacts to Resource manager for Executor containers.
5. Yarn Resource manager then creates and allocates the executor container to AM container.
6. AM container then distributes the spark application tasks to these executors
7. Executors then begins the tasks and creates the Spark-Executor processes and these reports back the status to Spark-Driver.

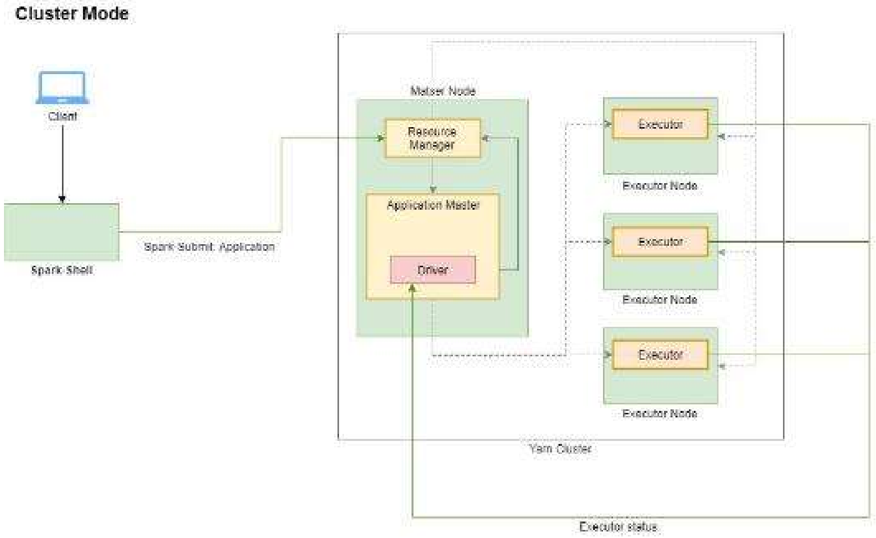


Apache Spark Execution – Cluster Mode



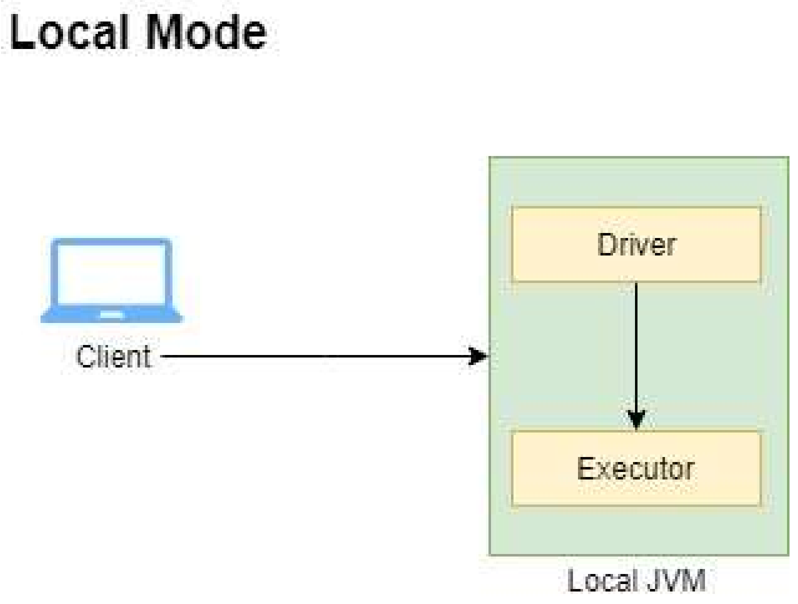
Apache Spark Execution – Cluster Mode

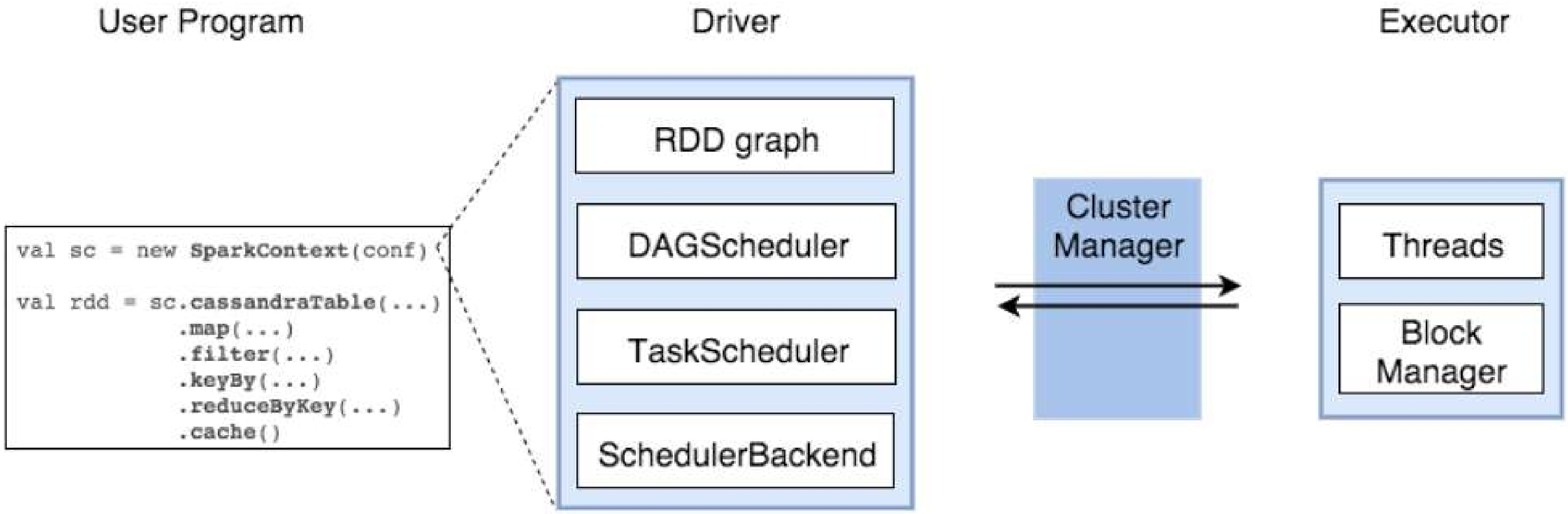
- Spark client submits the packed application to yarn request manager.
- Yarn request manager then creates the AM container. The Spark-Driver is also created in AM container.
- Rest of the flow is same as mentioned in the client mode.



Apache Spark Execution – Local Mode

- There is another mode in which spark can be run locally without any cluster requirement.
- This mode is suitable for scenarios when we do not have enough resources to create cluster.
- But in this mode you get only one executor and both the Driver and Excuter runs in the same JVM.





User Program

```
val sc = new SparkContext(conf)

val rdd = sc.cassandraTable(...)
            .map(...)
            .filter(...)
            .keyBy(...)
            .reduceByKey(...)
            .cache()
```

Driver

RDD graph

DAGScheduler

TaskScheduler

SchedulerBackend

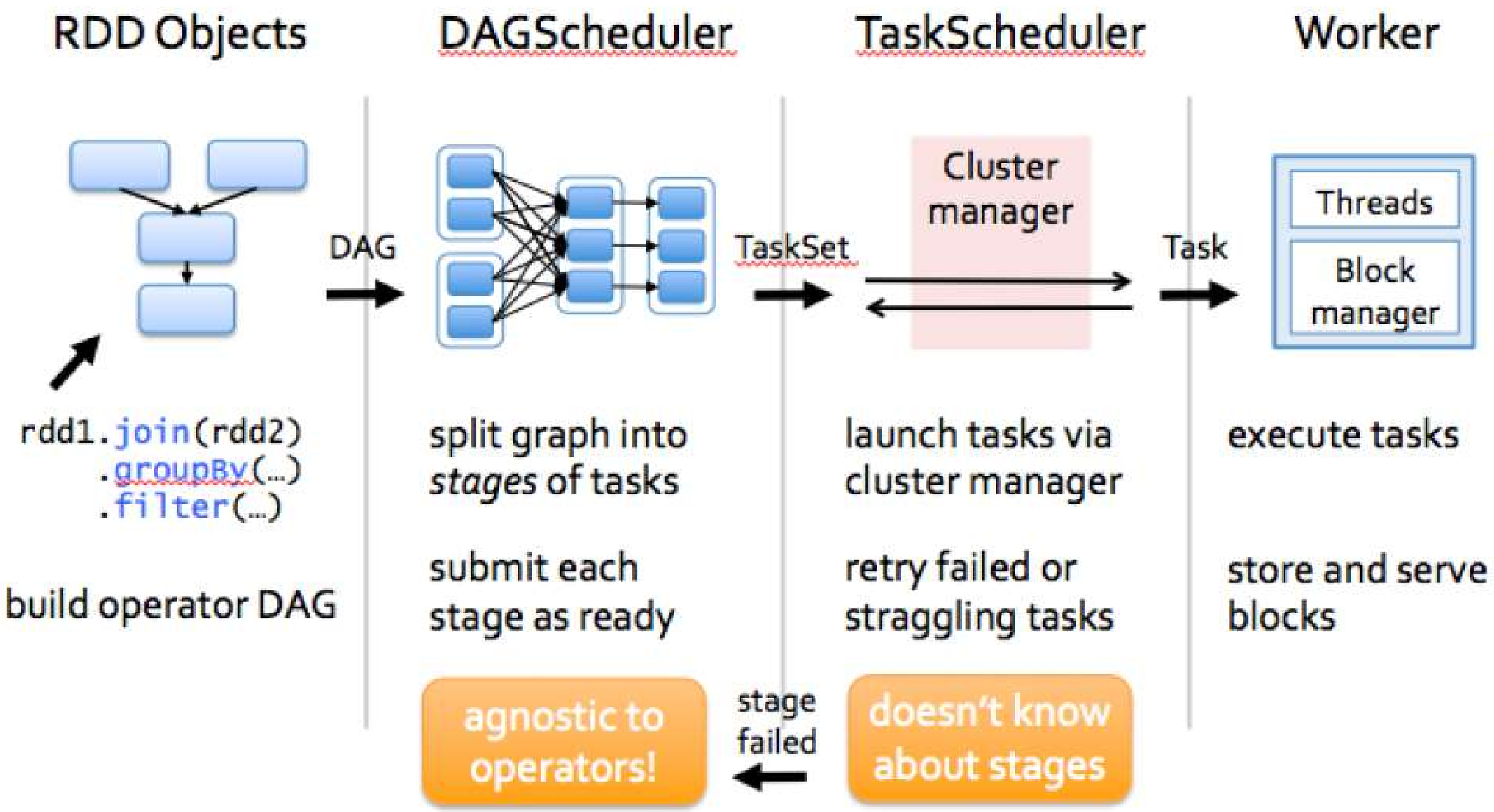
Cluster Manager

Executor

Threads

Block Manager

How Sparks work?



```
rdd1.join(rdd2)
    .groupBy(…)
    .filter(…)
```

build operator DAG

split graph into *stages* of tasks

submit each stage as ready

agnostic to operators!

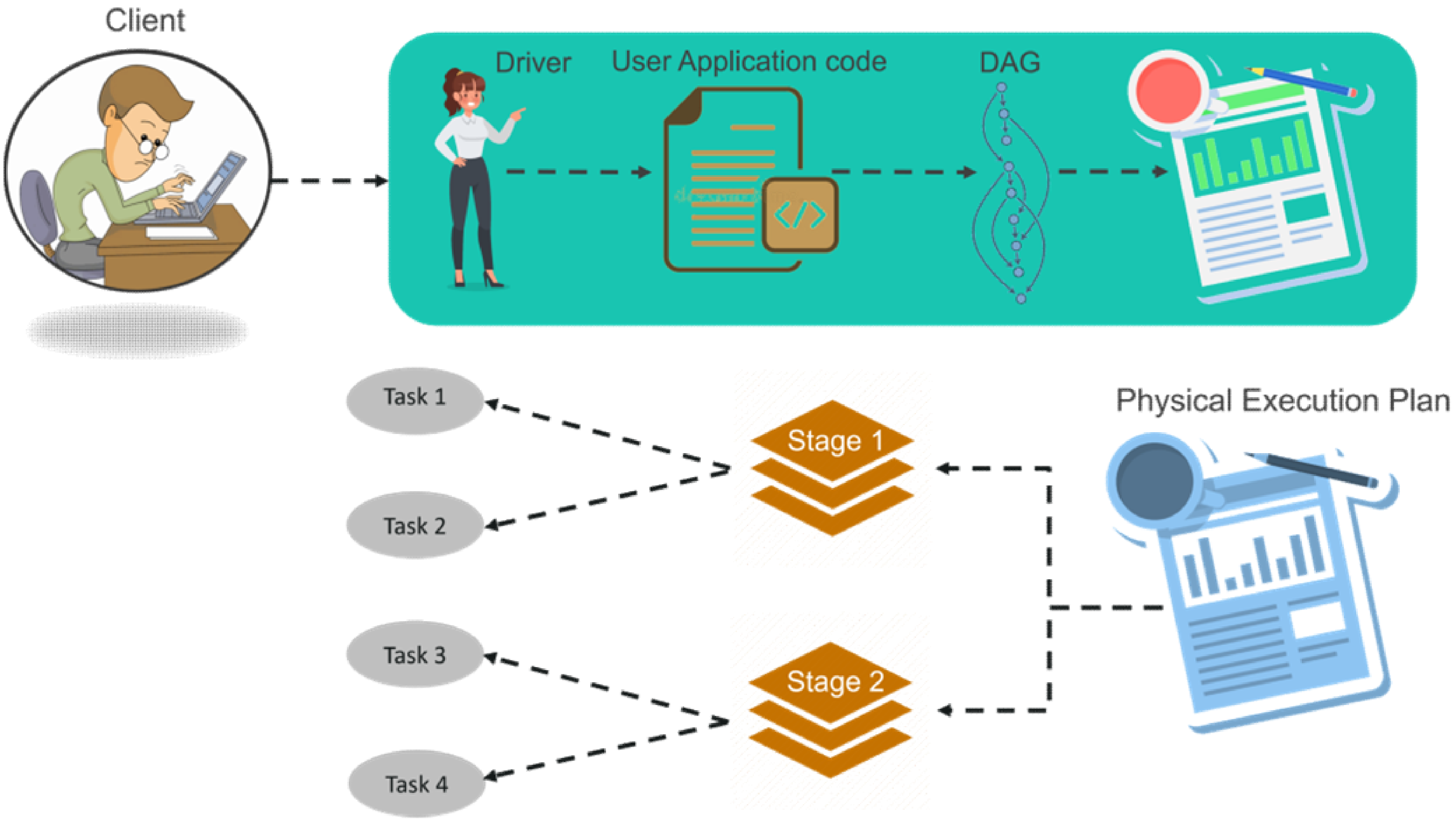
launch tasks via cluster manager

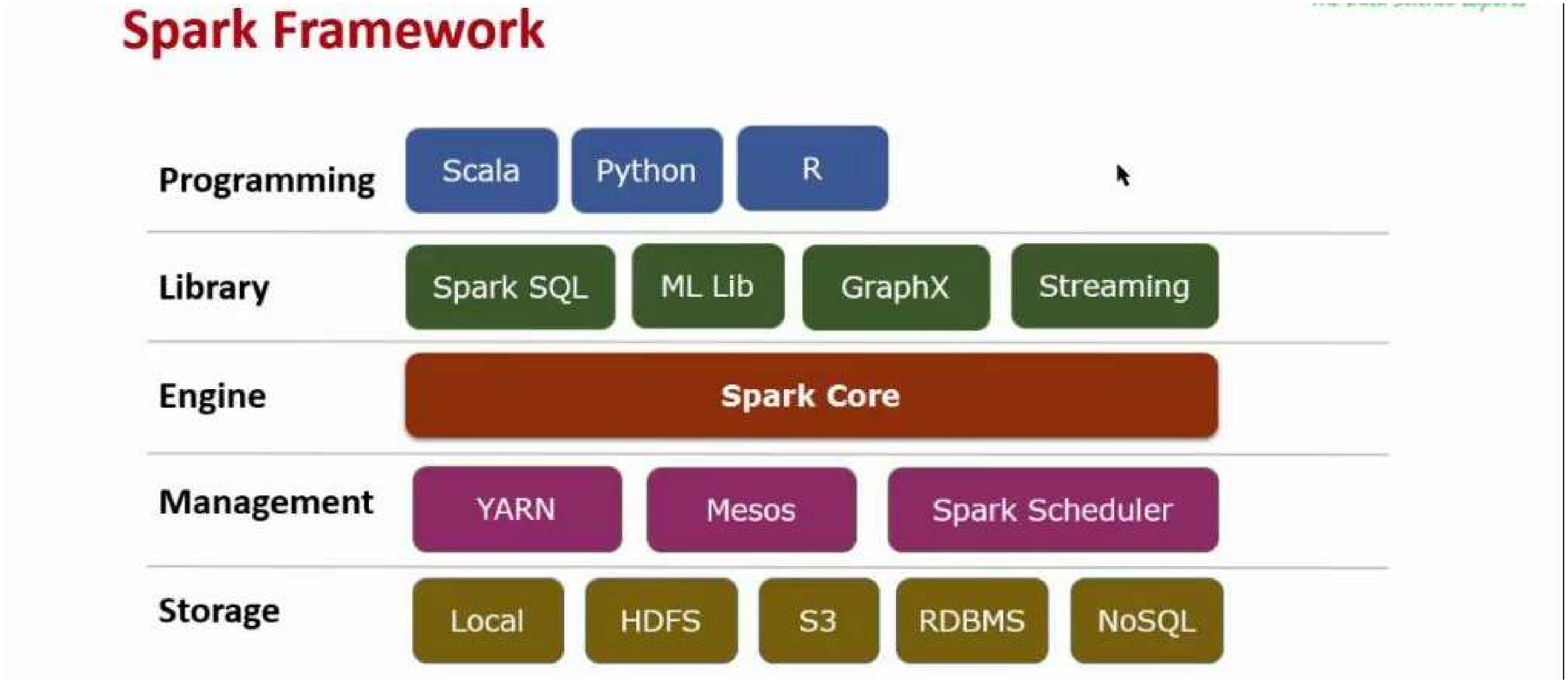
retry failed or straggling tasks

doesn't know about stages

execute tasks

store and serve blocks


Client
Driver
User Application code
DAG
Physical Execution Plan
Stage 1
Stage 2
Task 1
Task 2
Task 3
Task 4



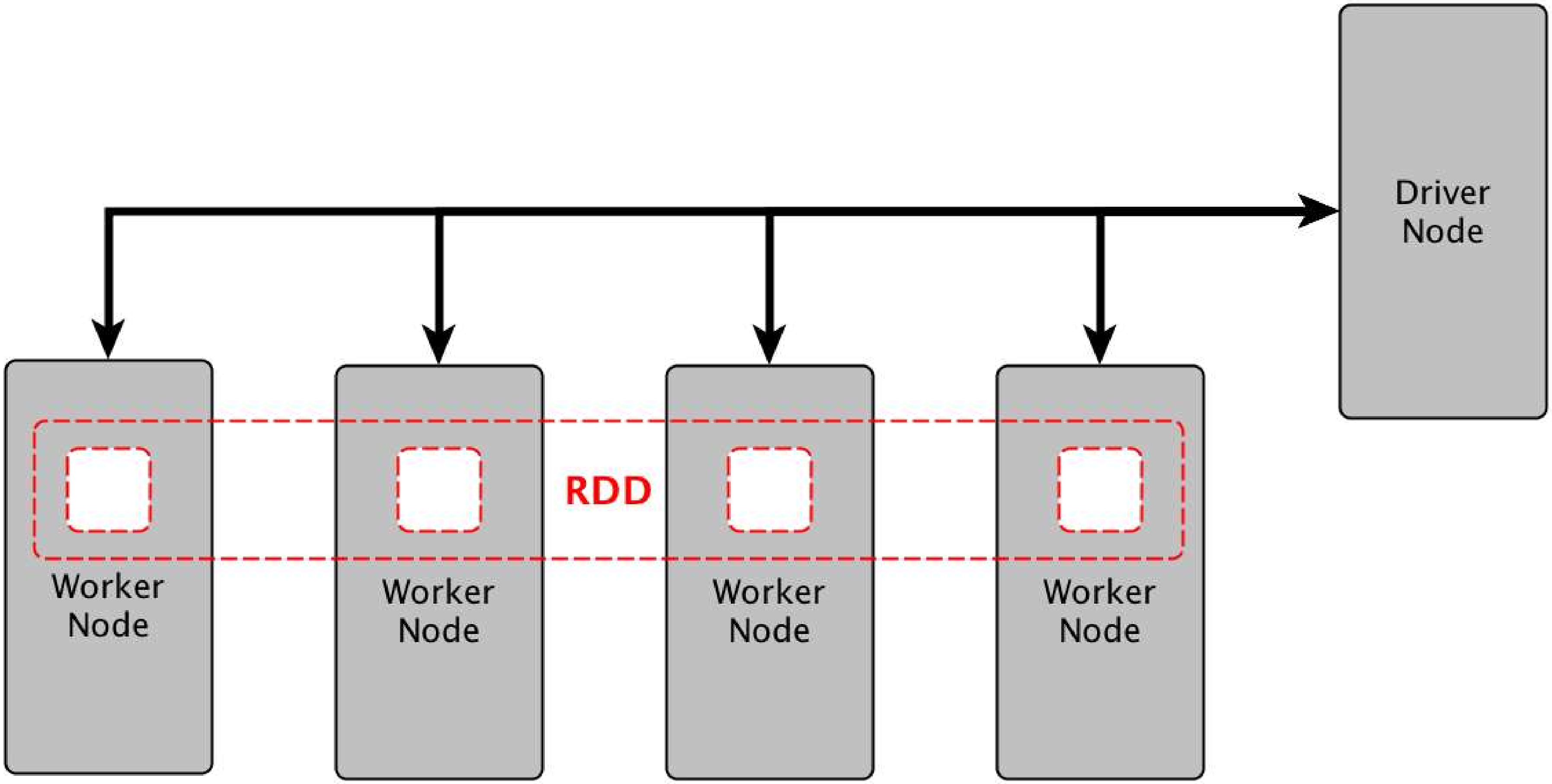
Spark Framework

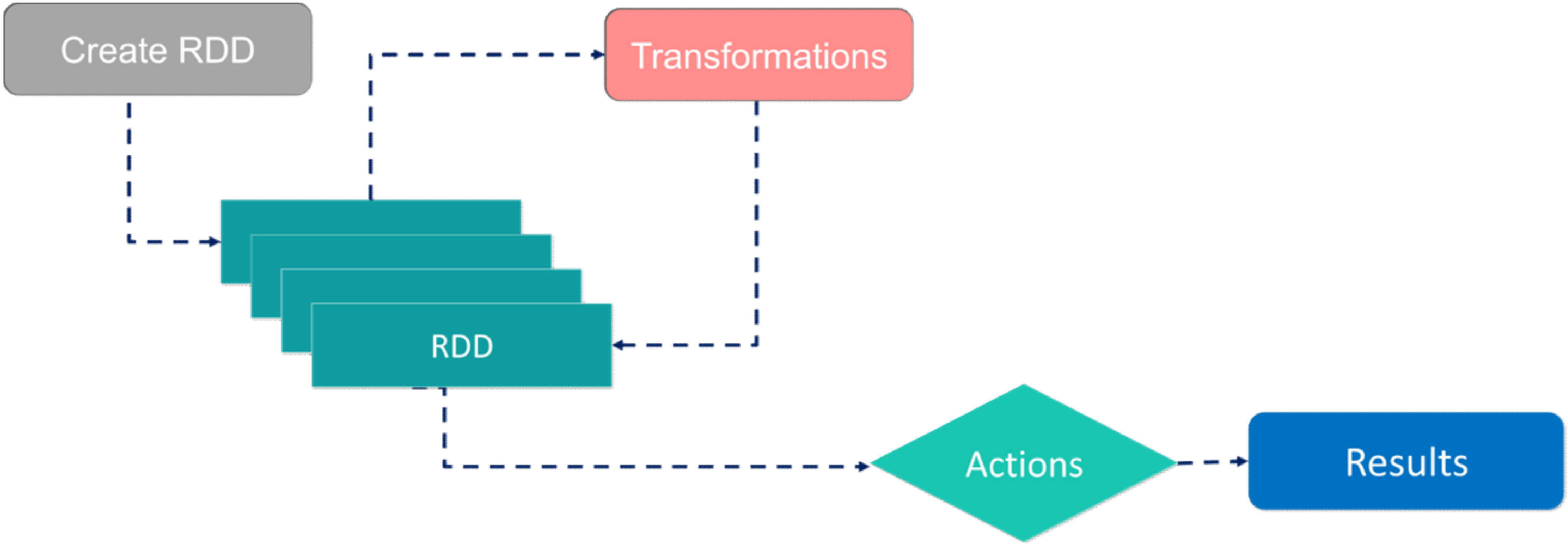
Programming	Scala	Python	R		
Library	Spark SQL	ML Lib	GraphX	Streaming	
Engine	Spark Core				
Management	YARN	Mesos	Spark Scheduler		
Storage	Local	HDFS	S3	RDBMS	NoSQL

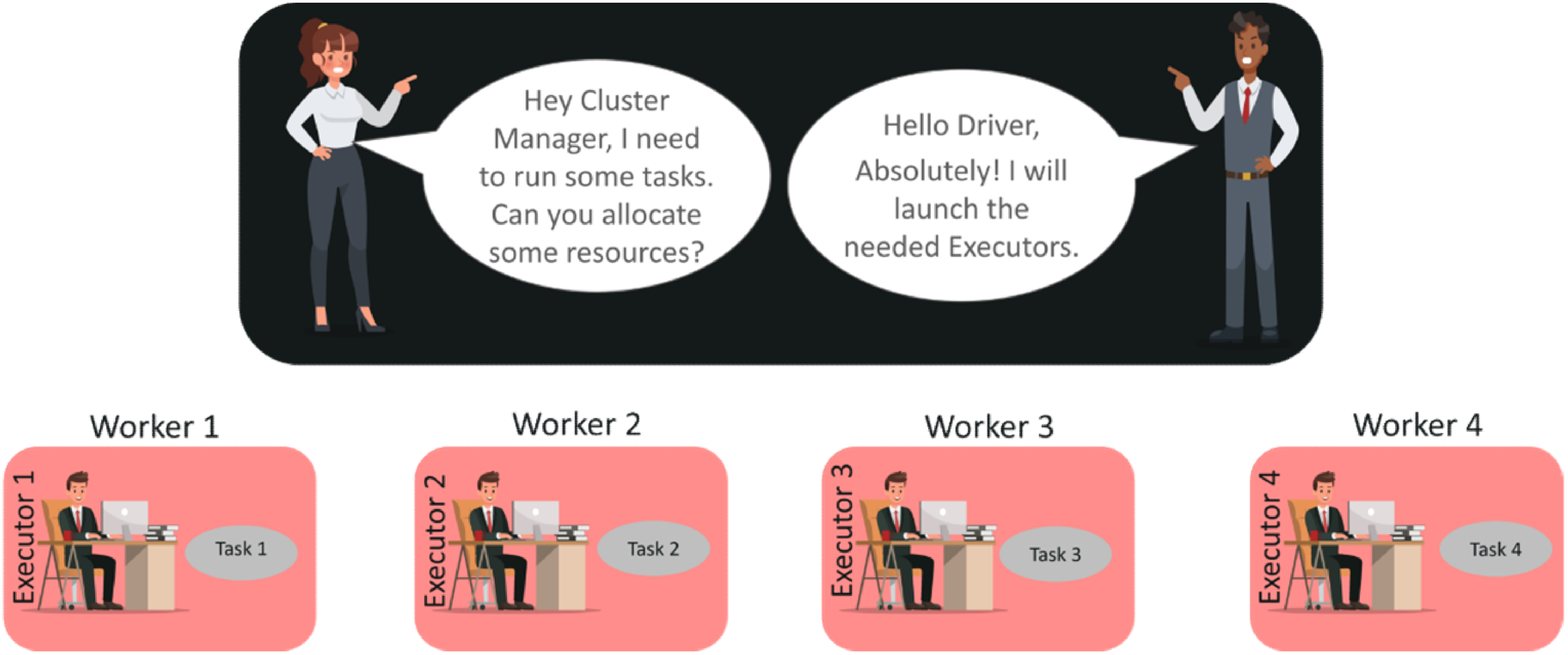
RDDs

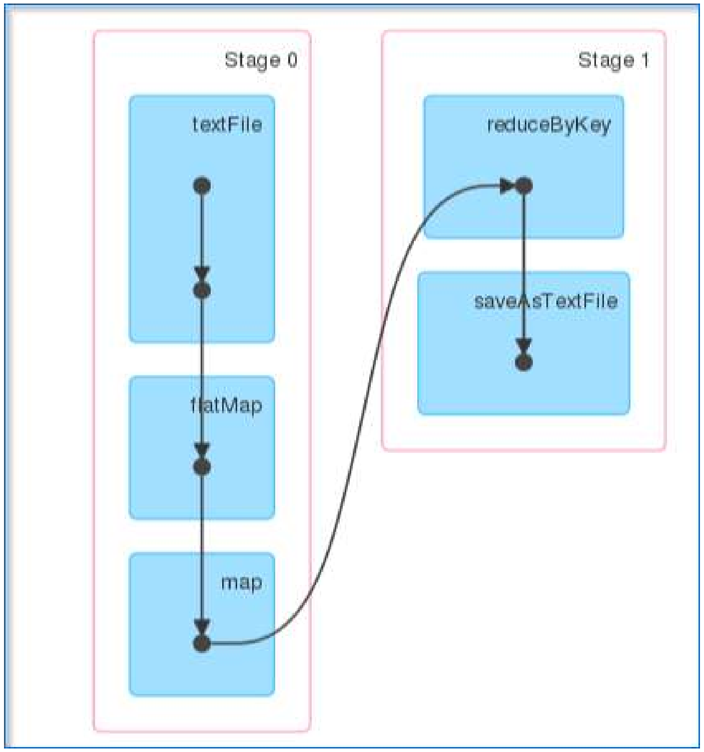
- immutable distributed collection of objects
- logical reference of a dataset which is partitioned across many server machines in the cluster

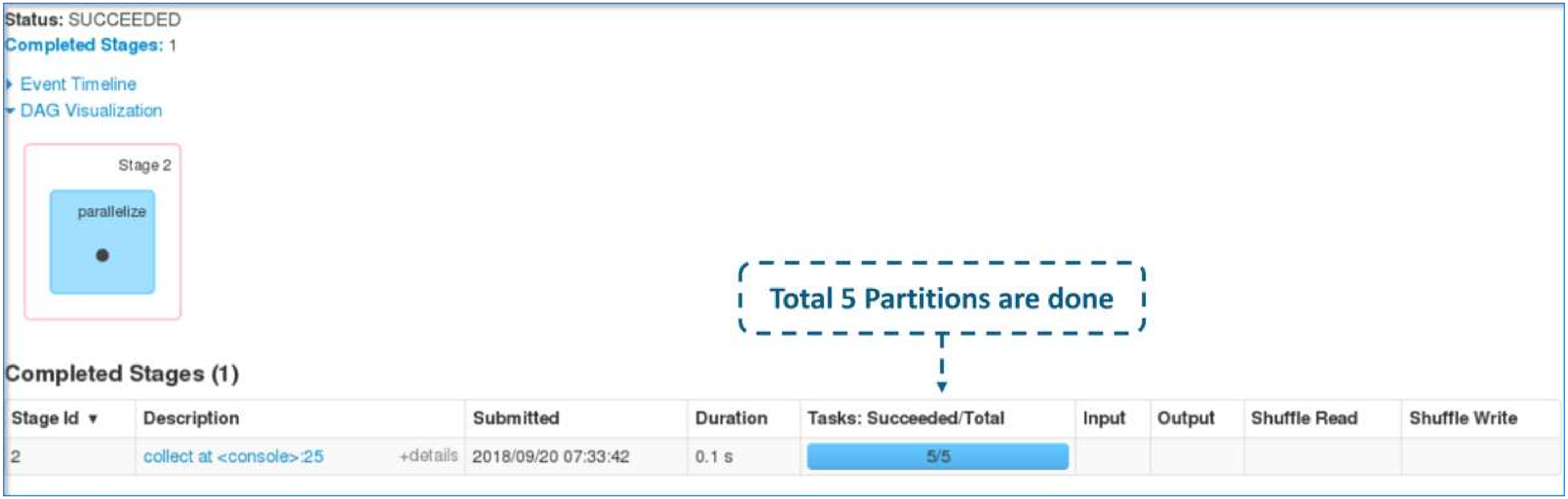
- Partitions
  - If it cannot fit into a single node it should be partitioned across various nodes.
  - More the number of partitions, the more the parallelism.
  - These partitions of an RDD is distributed across all the nodes in the network.
- By applying transformations you incrementally build a RDD lineage with all the parent RDDs of the final RDD(s).
- RDDs can also be thought of as a set of instructions that has to be executed, first instruction being the load instruction.


Driver Node
RDD
Worker Node
Worker Node
Worker Node
Worker Node


Create RDD
Transformations
RDD
Actions
Results


Hey Cluster Manager, I need to run some tasks. Can you allocate some resources?
Hello Driver, Absolutely! I will launch the needed Executors.
Worker 1
Executor 1
Task 1
Worker 2
Executor 2
Task 2
Worker 3
Executor 3
Task 3
Worker 4
Executor 4
Task 4


Stage 0
textFile
flatMap
map
Stage 1
reduceByKey
saveAsTextFile



Status: SUCCEEDED

Completed Stages: 1

▸ Event Timeline

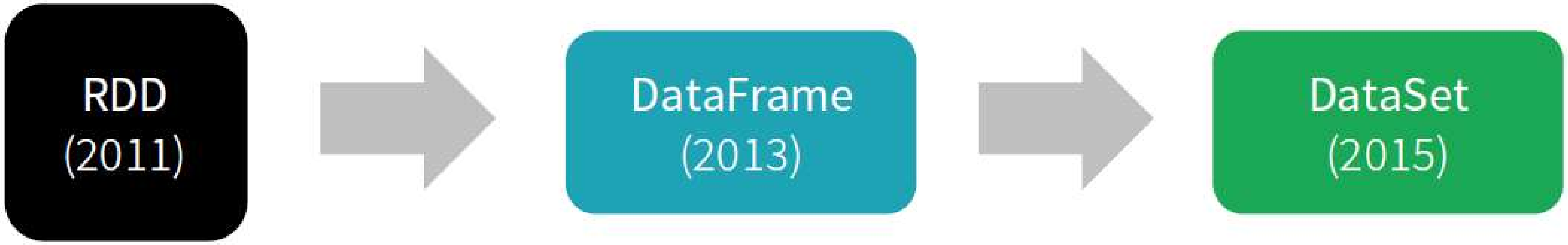
▾ DAG Visualization

Total 5 Partitions are done

Completed Stages (1)

Stage Id ▾	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
2	collect at <console>:25 +details	2018/09/20 07:33:42	0.1 s	5/5				

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
- "Best of both worlds"
  type safe + fast

Shared Variables

- Broadcast variables
  - Allows user to keep a read-only variable cached on each machine vs shipping it with tasks.
  - e.g. lookup table
- Accumulators
  - workers can “add” to using associative operations
  - only driver can read
  - used for
    - counters
    - sums

RDD Operations – Expressive

- Transformations
  - Creation of a new RDD dataset from an existing
    - map, filter, distinct, union, sample, groupByKey, join, reduce, etc…
- Actions
  - Return a value after running a computation
    - collect, count, first, takeSample, foreach, etc…
- Easy: Expressive API

- map
- filter
- groupBy
- sort
- union
- join
- leftOuterJoin
- rightOuterJoin
- reduce
- count
- fold
- reduceByKey
- groupByKey
- cogroup
- cross
- zip
- sample
- take
- first
- partitionBy
- mapWith
- pipe
- save

Table 3-2. Basic RDD transformations on an RDD containing {1, 2, 3, 3}

Function name	Purpose	Example	Result
`map()`	Apply a function to each element in the RDD and return an RDD of the result.	`rdd.map(x => x + 1)`	`{2, 3, 4, 4}`
`flatMap()`	Apply a function to each element in the RDD and return an RDD of the contents of the iterators returned. Often used to extract words.	`rdd.flatMap(x => x.to(3))`	`{1, 2, 3, 2, 3, 3, 3}`
`filter()`	Return an RDD consisting of only elements that pass the condition passed to `filter()`.	`rdd.filter(x => x != 1)`	`{2, 3, 3}`
`distinct()`	Remove duplicates.	`rdd.distinct()`	`{1, 2, 3}`
`sample(withReplacement, fraction, [seed])`	Sample an RDD, with or without replacement.	`rdd.sample(false, 0.5)`	Nondeterministic

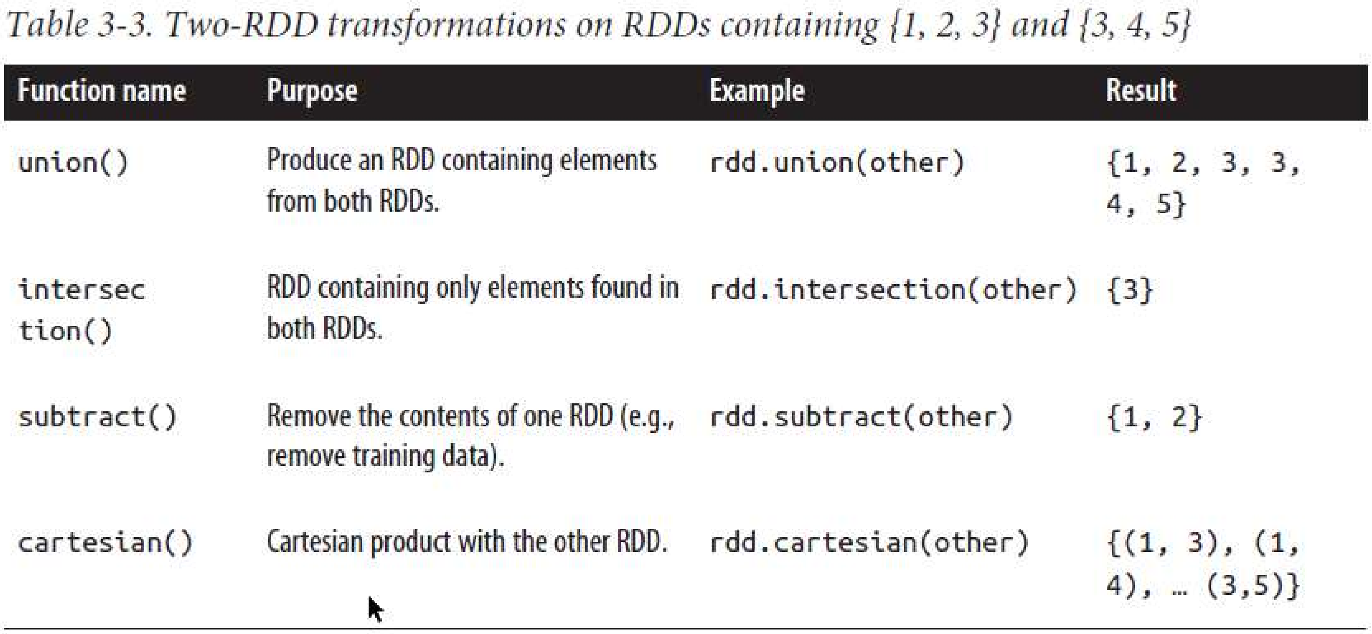


Table 3-3. Two-RDD transformations on RDDs containing {1, 2, 3} and {3, 4, 5}

Function name	Purpose	Example	Result
`union()`	Produce an RDD containing elements from both RDDs.	`rdd.union(other)`	`{1, 2, 3, 3, 4, 5}`
`intersection()`	RDD containing only elements found in both RDDs.	`rdd.intersection(other)`	`{3}`
`subtract()`	Remove the contents of one RDD (e.g., remove training data).	`rdd.subtract(other)`	`{1, 2}`
`cartesian()`	Cartesian product with the other RDD.	`rdd.cartesian(other)`	`{(1, 3), (1, 4), … (3,5)}`

Table 3-4. Basic actions on an RDD containing {1, 2, 3, 3}

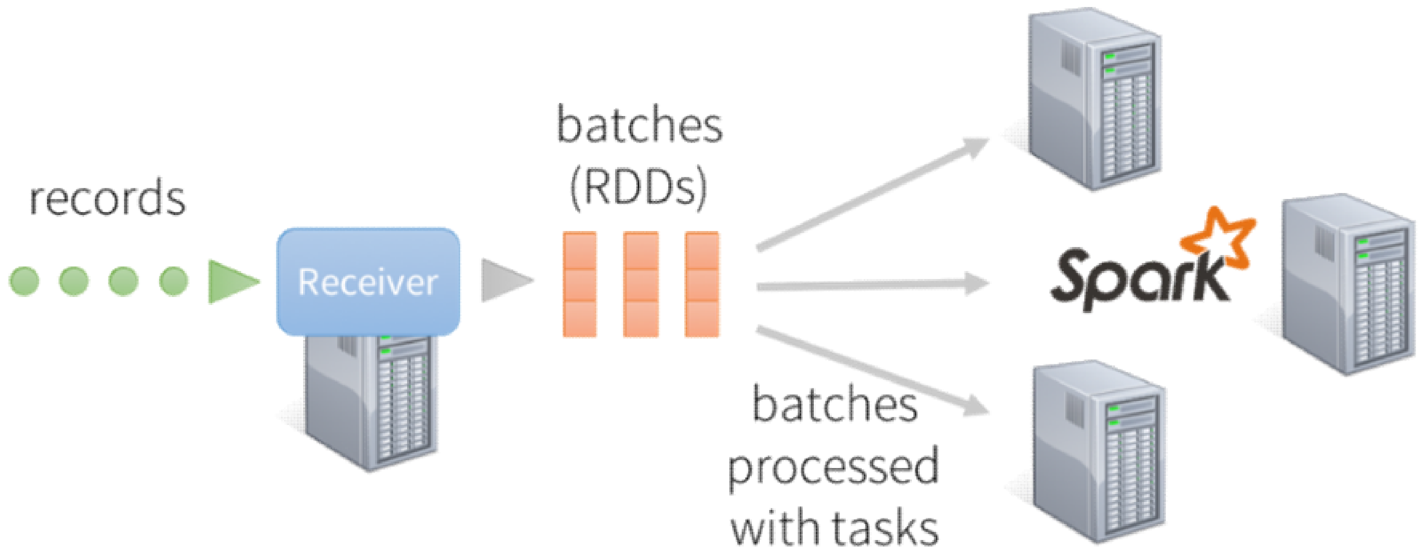
Function name	Purpose	Example	Result
`collect()`	Return all elements from the RDD.	`rdd.collect()`	`{1, 2, 3, 3}`
`count()`	Number of elements in the RDD.	`rdd.count()`	`4`
`countByValue()`	Number of times each element occurs in the RDD.	`rdd.countByValue()`	`{(1, 1), (2, 1), (3, 2)}`

Table 4-3. Actions on pair RDDs (example ({(1, 2), (3, 4), (3, 6)}))

Function	Description	Example	Result
`countByKey()`	Count the number of elements for each key.	`rdd.countByKey()`	`{(1, 1), (3, 2)}`
`collectAsMap()`	Collect the result as a map to provide easy lookup.	`rdd.collectAsMap()`	`Map{(1, 2), (3, 4), (3, 6)}`
`lookup(key)`	Return all values associated with the provided key.	`rdd.lookup(3)`	`[4, 6]`

Spark Streaming

discretized stream processing



records processed in batches with short tasks
each batch is a RDD (partitioned dataset)

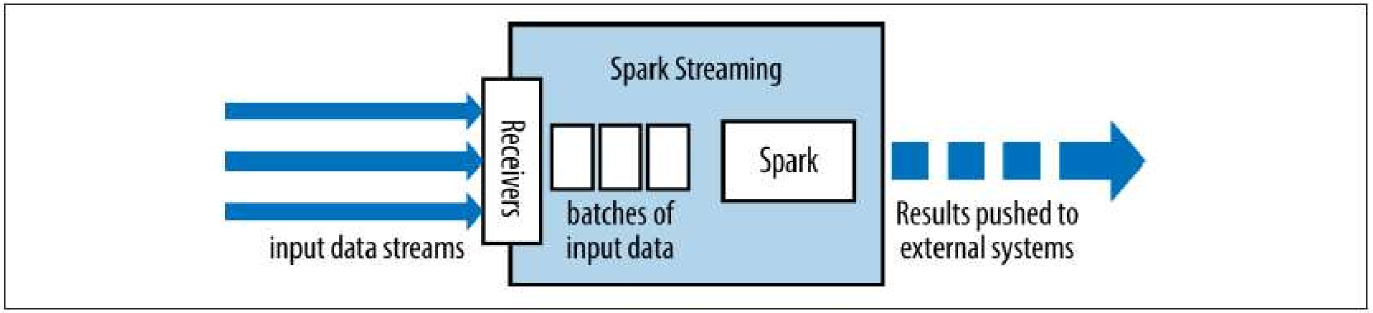


Figure 10-1. High-level architecture of Spark Streaming

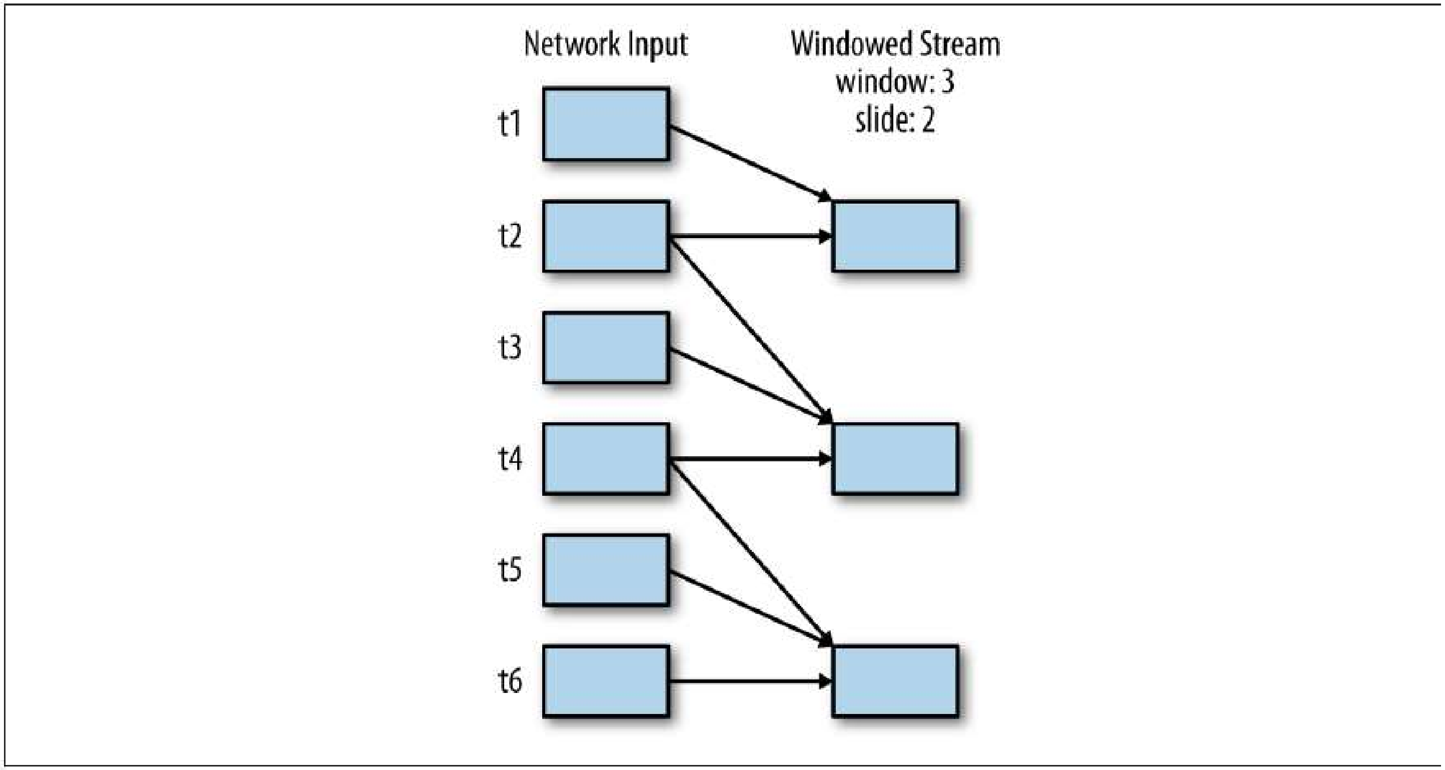


Figure 10-6. A windowed stream with a window duration of 3 batches and a slide duration of 2 batches; every two time steps, we compute a result over the previous 3 time steps

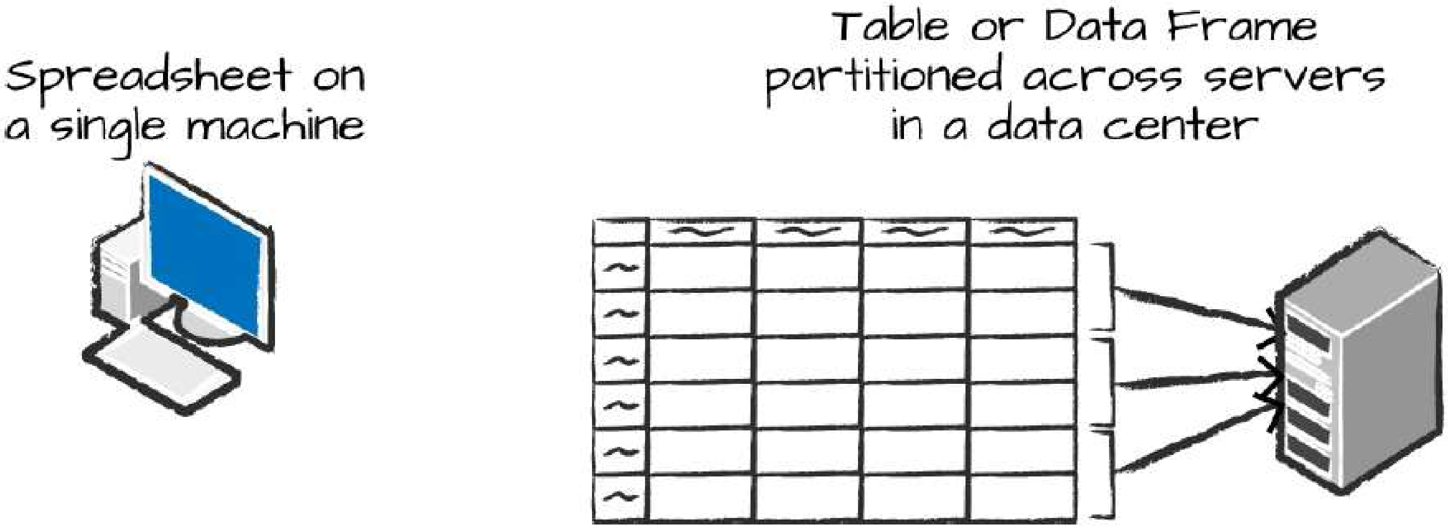


Figure 2-3. Distributed versus single-machine analysis

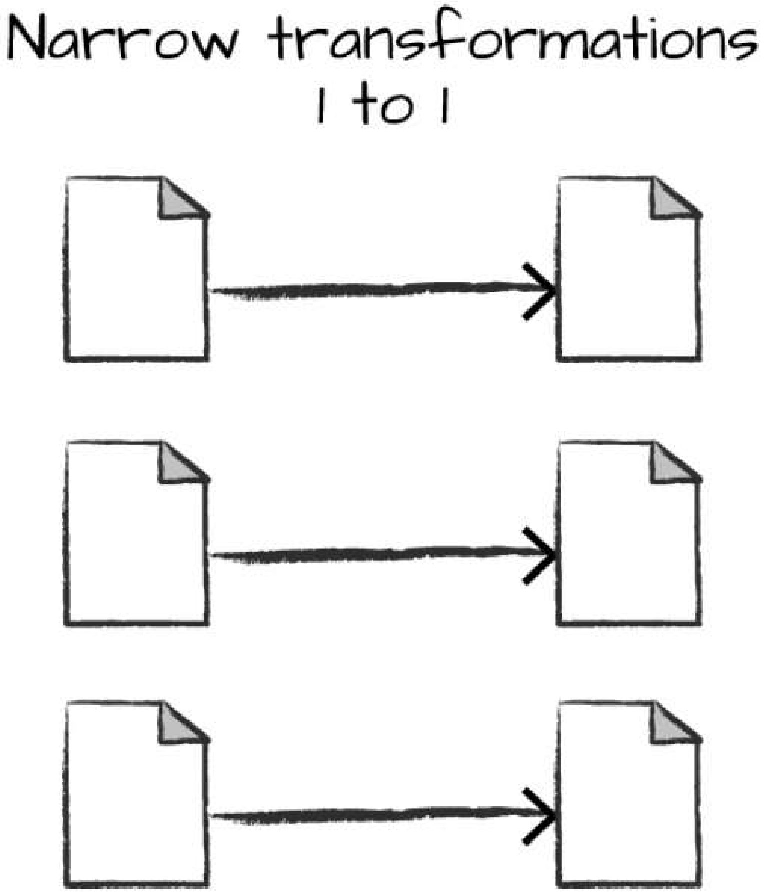
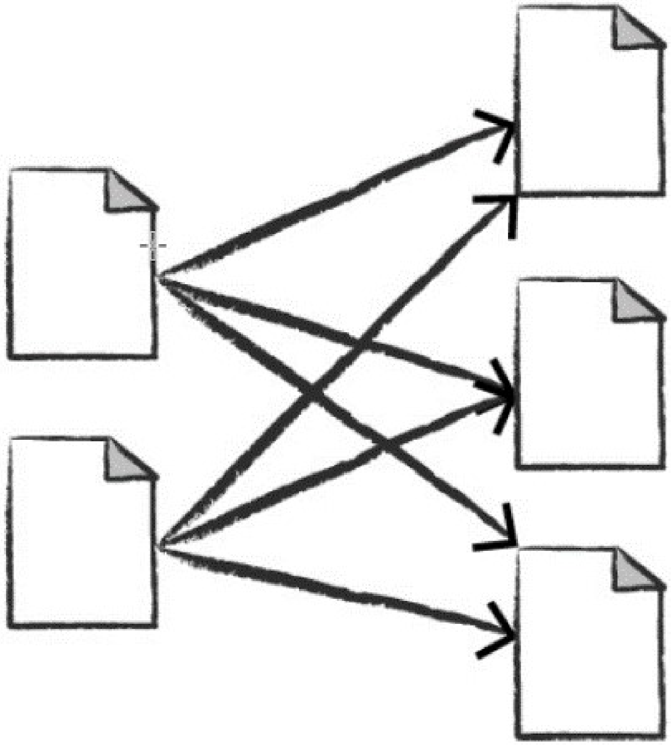


Figure 2-4. A narrow dependency

Wide transformations (shuffles) 1 to N



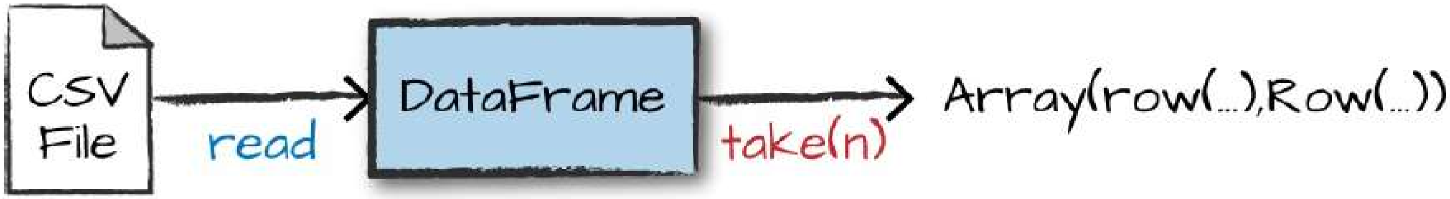


Figure 2-7. Reading a CSV file into a DataFrame and converting it to a local array or list of rows

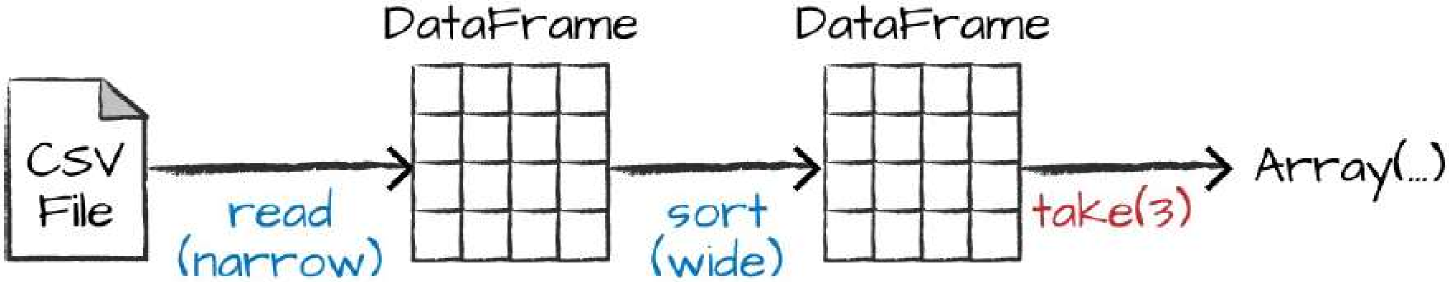


Figure 2-8. Reading, sorting, and collecting a DataFrame

Catalyst Optimizer

- Spark SQL uses an optimizer called catalyst to optimize all the queries
- This optimizer makes queries run much faster
- An optimizer automatically finds out the most efficient plan to execute data operations specified in the user’s program.
- logical plan — series of algebraic or language constructs, as for example: SELECT, GROUP BY or UNION keywords in SQL. It’s usually represented as a tree.
- physical plan — Concerns low level operations.

Catalyst Optimizer

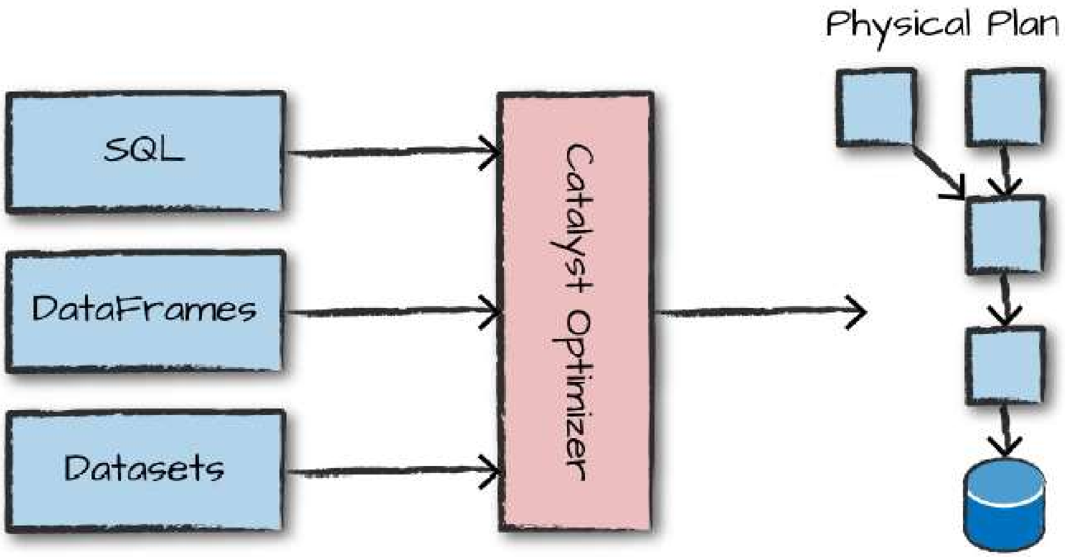


Figure 4-1. The Catalyst Optimizer

Catalyst Optimizer

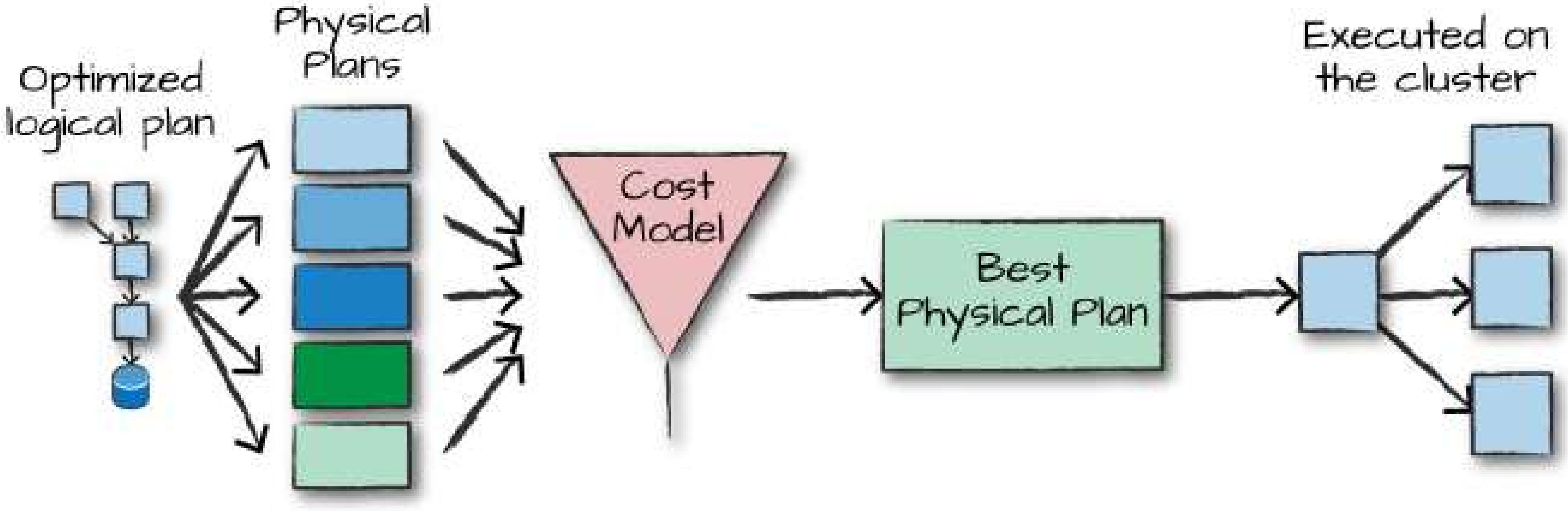


Figure 4-3. The physical planning process

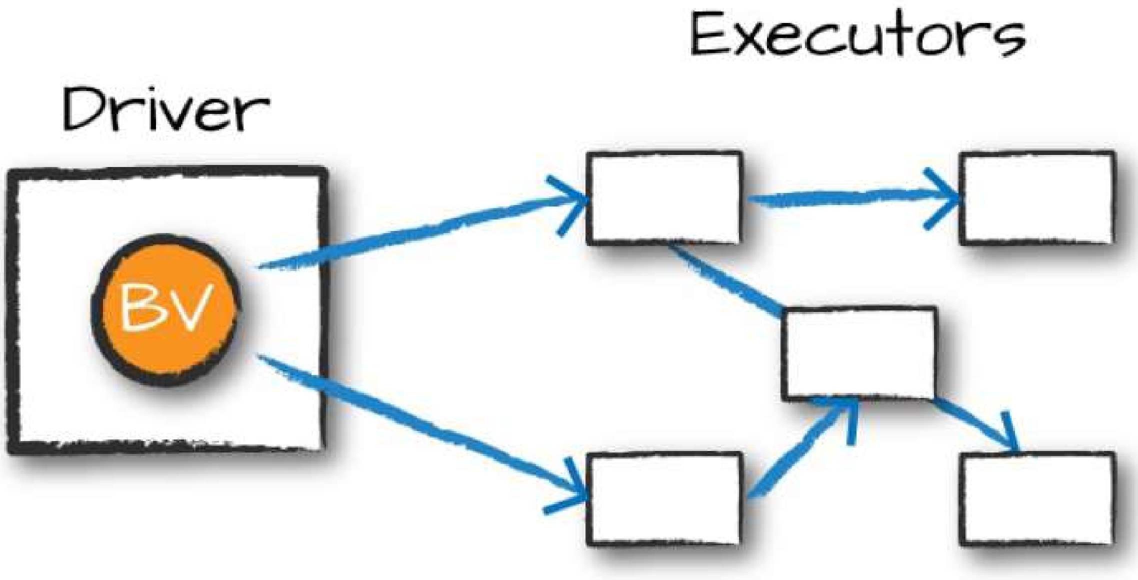


Figure 14-1. Broadcast variables

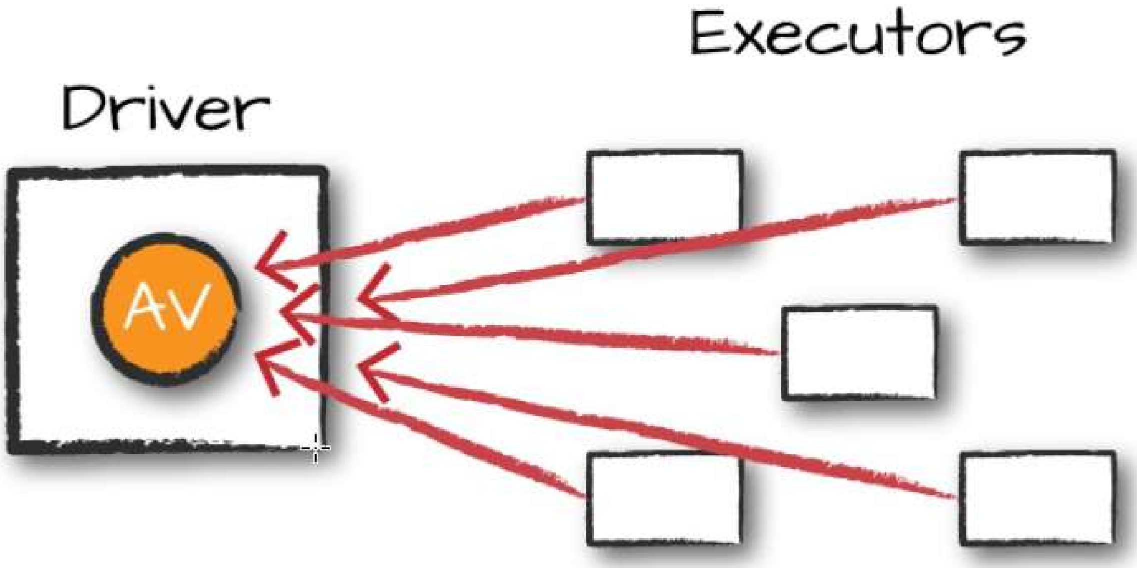


Figure 14-2. Accumulator variable

Heartbeat Receiver

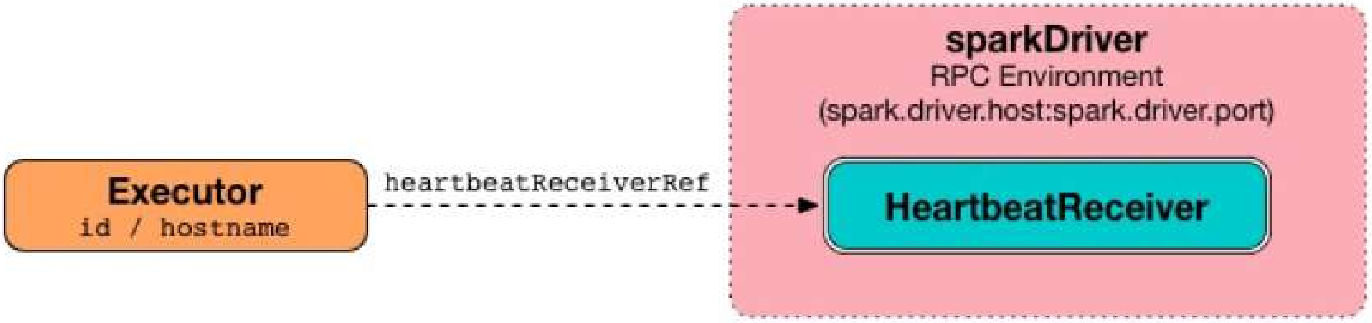
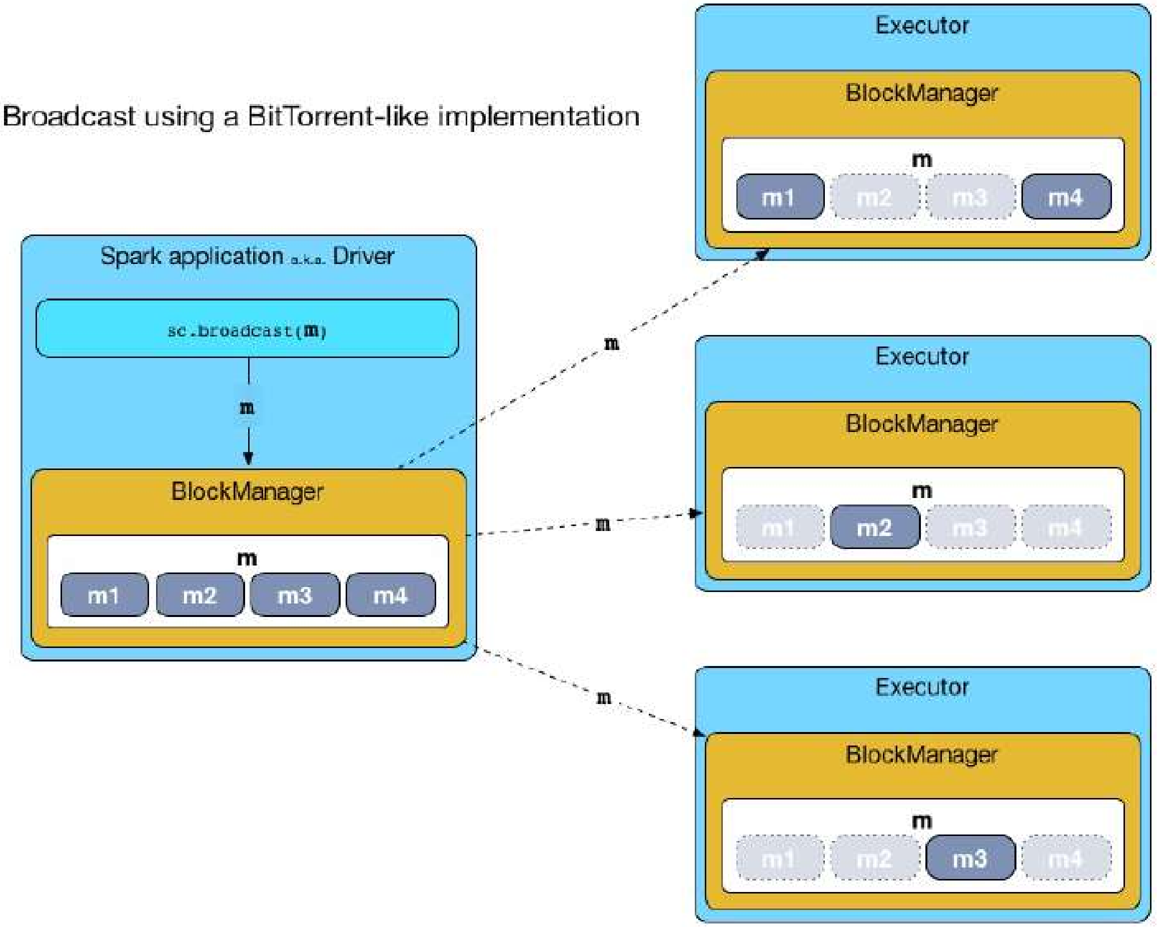


Figure 2. Executors use HeartbeatReceiver endpoint to report task metrics

Broadcast


Broadcast using a BitTorrent-like implementation
Spark application a.k.a. Driver
sc.broadcast(m)
m
BlockManager
m
m1
m2
m3
m4
m
m
m
Executor
BlockManager
m
m1
m2
m3
m4
Executor
BlockManager
m
m1
m2
m3
m4
Executor
BlockManager
m
m1
m2
m3
m4

SparkContext

Located in the Spark Driver

Entry point for RDDs

The Spark application

- One SparkContext per application

Created for you in the REPL

You need to create for spark2-submit

SparkSession

Entry point to Spark SQL

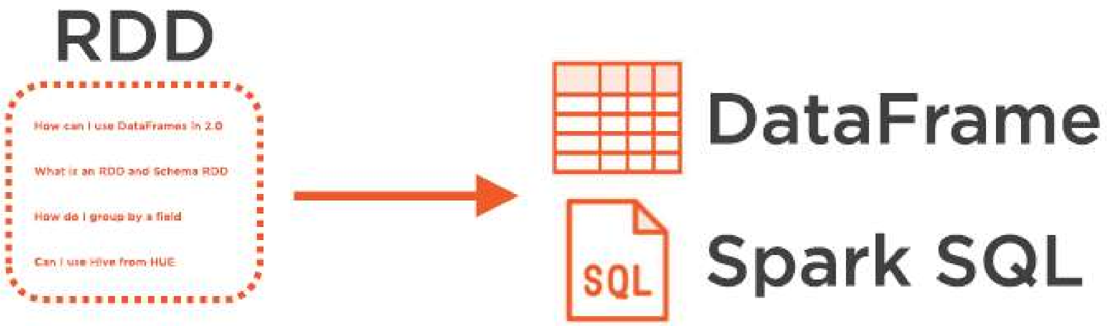
Merges SQLContext and HiveContext

Access SparkContext

Can have multiple SparkSession objects

Created for you in REPL

You need to create for spark2-submit



RDD

How can I use DataFrames in 2.0

What is an RDD and Schema RDD

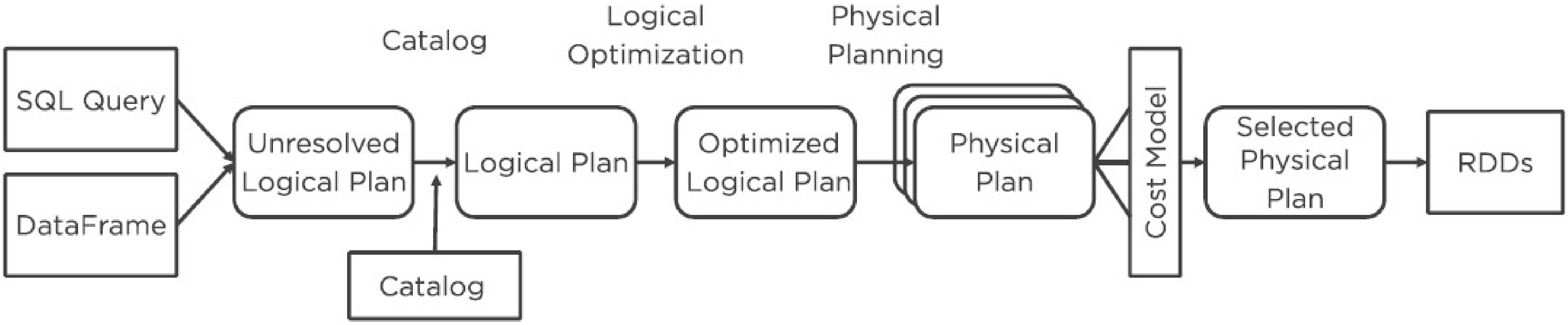
How do I group by a field

Can I use Hive from HUE

DataFrame

Spark SQL

Execution


Catalog
Logical Optimization
Physical Planning
SQL Query
DataFrame
Unresolved Logical Plan
Logical Plan
Optimized Logical Plan
Physical Plan
Cost Model
Selected Physical Plan
RDDs
Catalog

Thanks