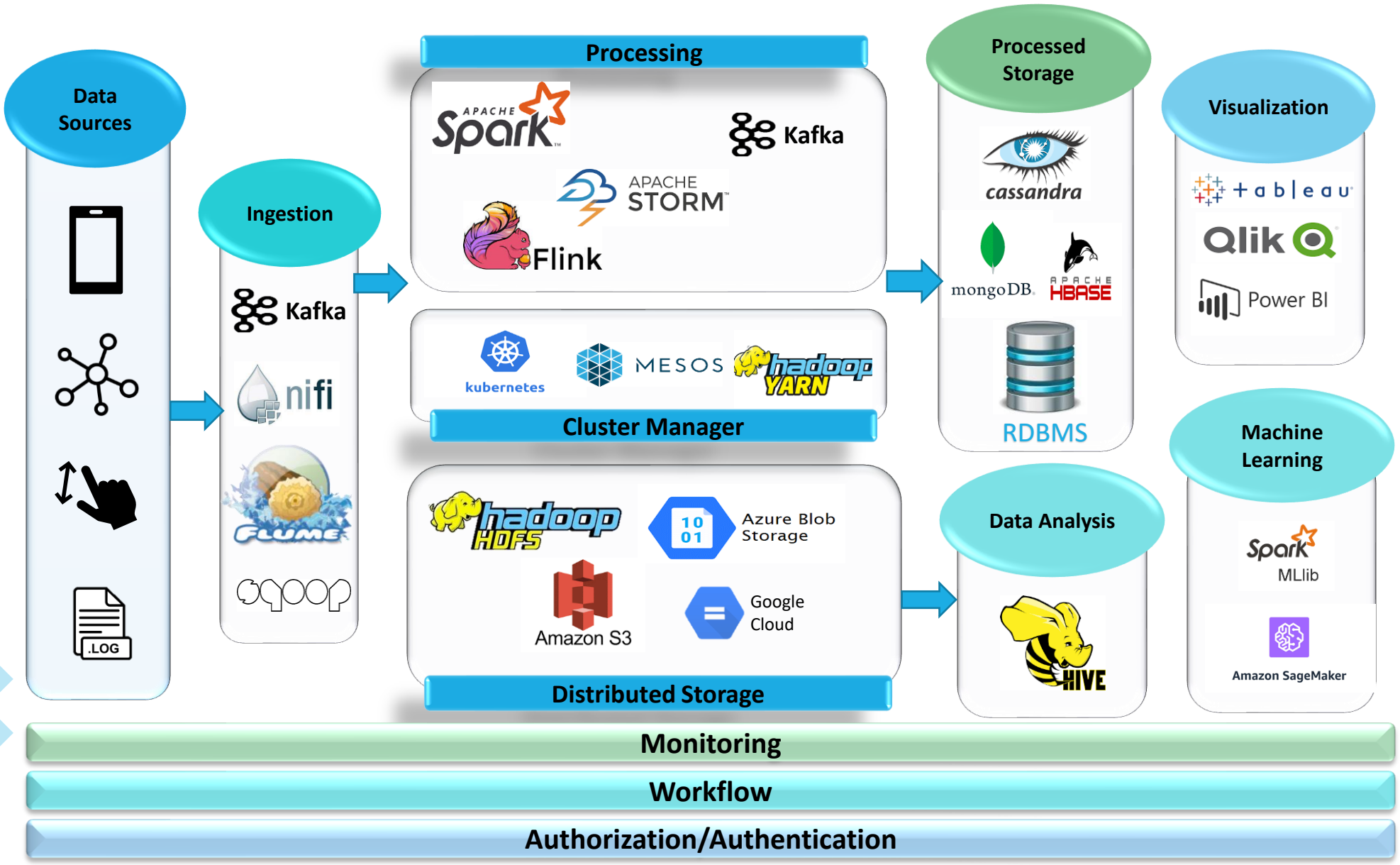
Big Data Ecosystem



Apache Spark

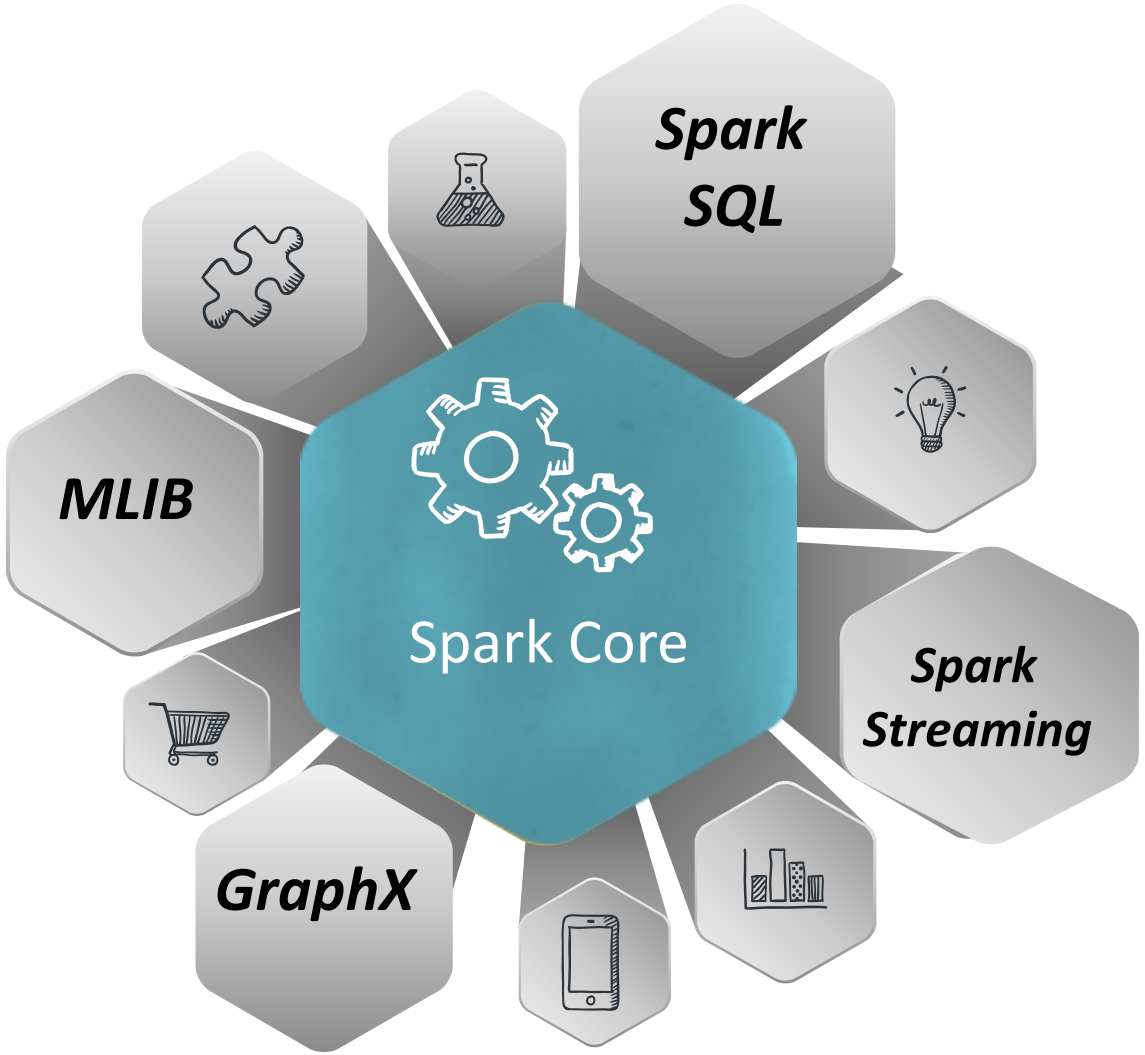
Apache Spark

Spark is a framework for processing distributed data in efficient and fault tolerant manner.

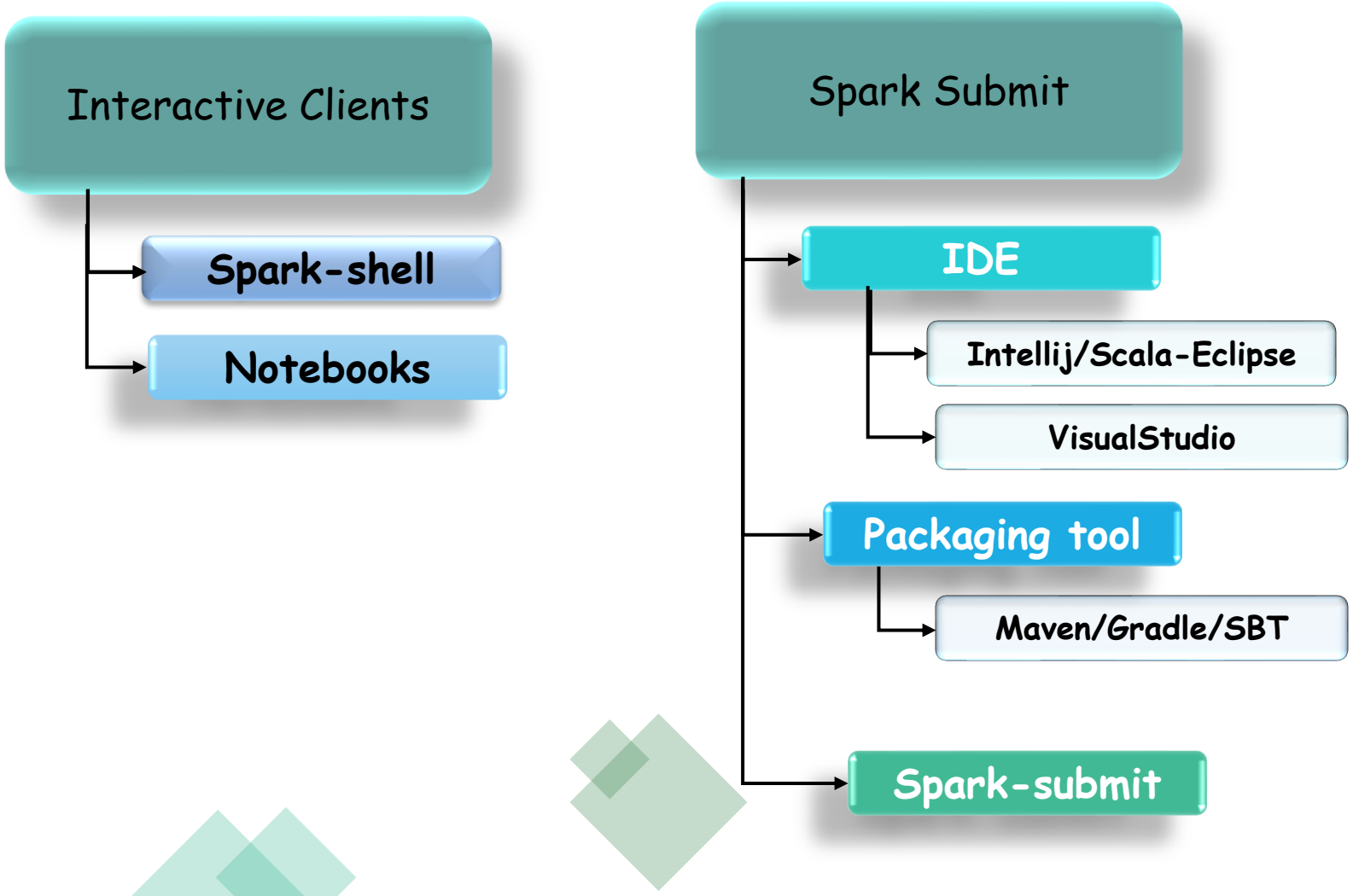
Spark Vs Hadoop MapReduce

- In memory computation
- 100X times faster
- Intermediate results are stored in memory
- Lazy Evaluation
- Support for multiple language- Python, Scala, Java

Spark Components



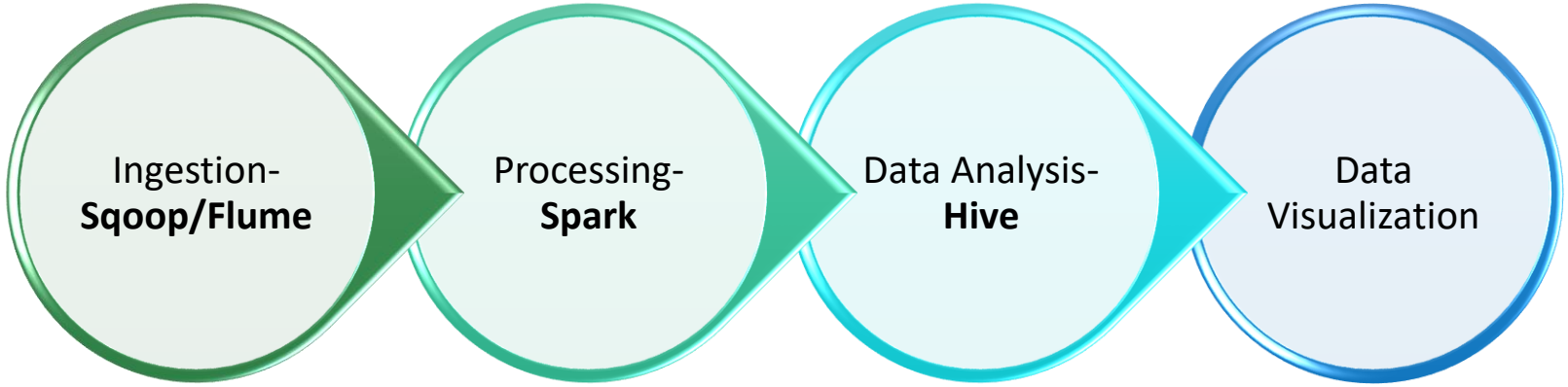
Execution Methods



What makes it better than MapReduce?

- In memory computation
- 100X times faster
- Intermediate results are stored in memory
- Lazy Evaluation
- Support for multiple language- Python, Scala, Java

Big Data Pipelines



Spark Example

01 Read Json document

02 Filter movies with adventure genre

03 Convert movie name to upper case

04 Group by released year, and get movies name list

05 Finally save it back in json format

How Apache Spark run?

How Spark runs?

01 Understanding Cluster Manager (yarn)

02 Understanding Distributed Storage (HDFS)

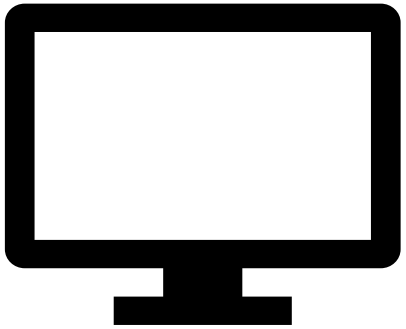
03 Spark Application with Yarn & HDFS

What is Cluster?

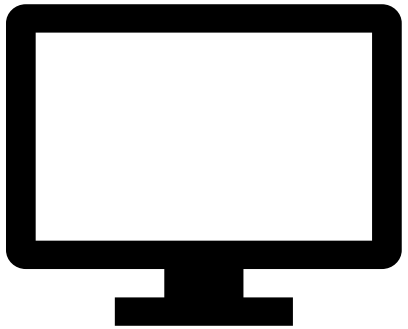
Multiple machines connected over the network to perform as Single system is called cluster.

4 Nodes Cluster

Master
4 cores/16 GIB



Worker
4 cores/16 GIB



Worker
4 cores/16 GIB

Worker
4 cores/16 GIB

What is Cluster Manager

Cluster Manager is responsible allocating resources, keeping track of state of cluster and scheduling tasks.

Spark can run on which cluster manager?

01 Local Mode

02 Apache Yarn

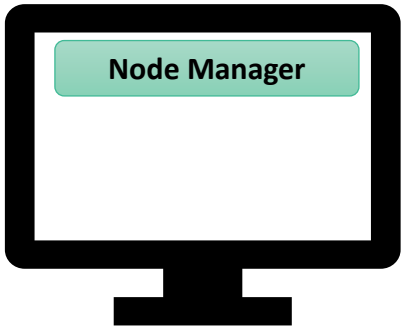
03 Apache Mesos

04 Standalone cluster

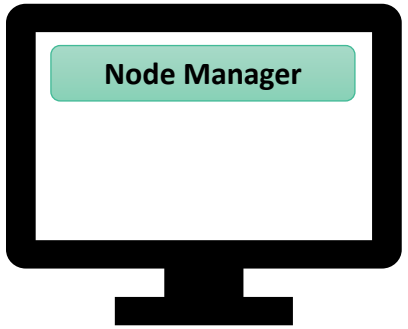
05 Kubernetes

Yarn Cluster Manager

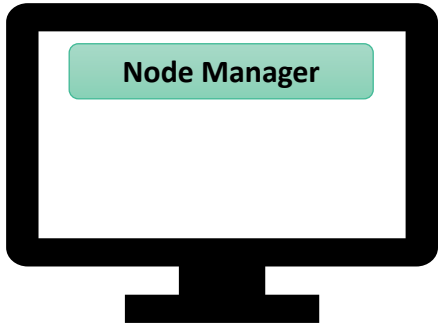
Master
4 cores/16 GIB



Worker
4 cores/16 GIB



Worker
4 cores/16 GIB



Worker
4 cores/16 GIB

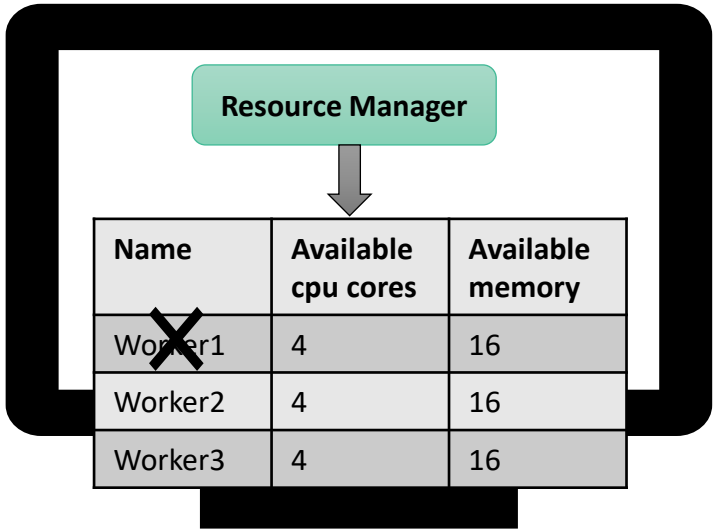
Yarn Cluster Manager

Keeps tracks of cluster resources

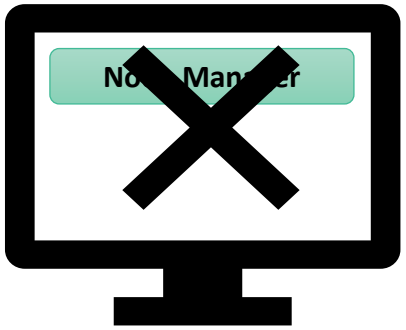
Responsible for resource allocations

Maintains a list of healthy and dead nodes in cluster

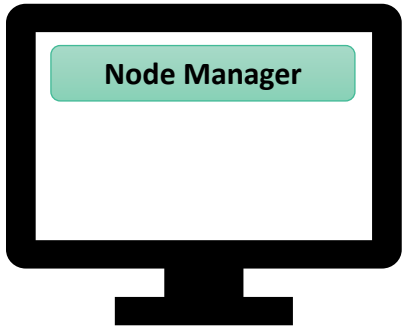
Maintains a list of live/dead Application Master



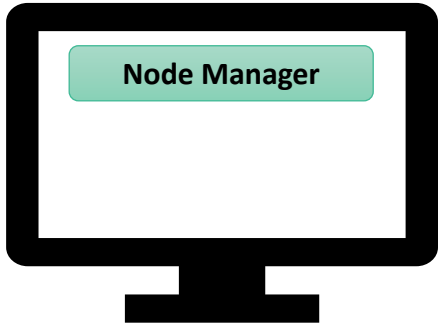
Master
4 cores/16 GIB



Worker
4 cores/16 GIB



Worker
4 cores/16 GIB

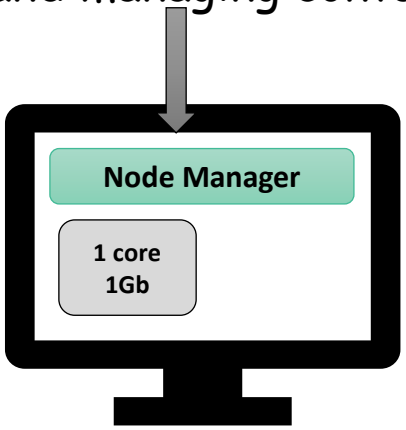


Worker
4 cores/16 GIB

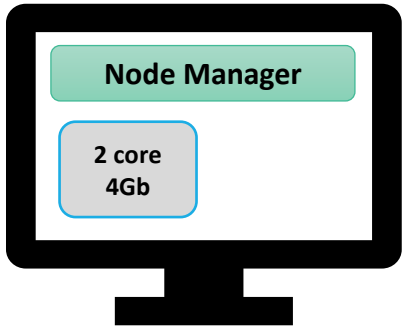
Yarn Cluster Manager

Master
4 cores/16 GIB

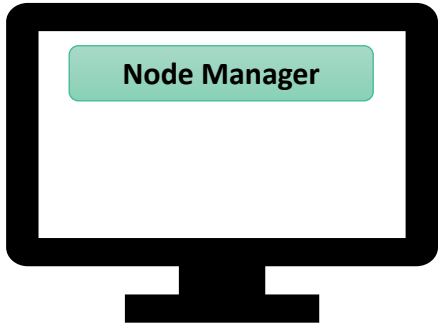
Launching and managing containers on a node



Worker
4 cores/16 GIB



Worker
4 cores/16 GIB



Worker
4 cores/16 GIB

What is Distributed File System?

Distributed file system which means data is distributed across multiple machines that these machines might be located in separate physical locations.

Spark can run on which distributed storage?

01 Local File System

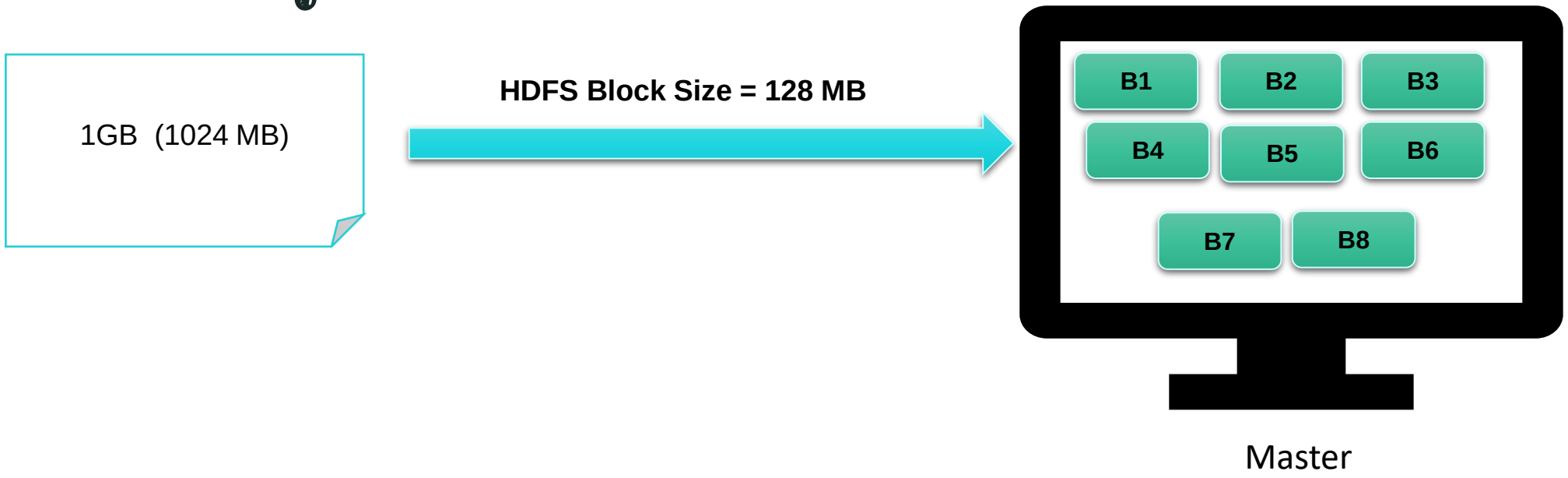
02 Hadoop Distributed File System (HDFS)

03 Amazon S3

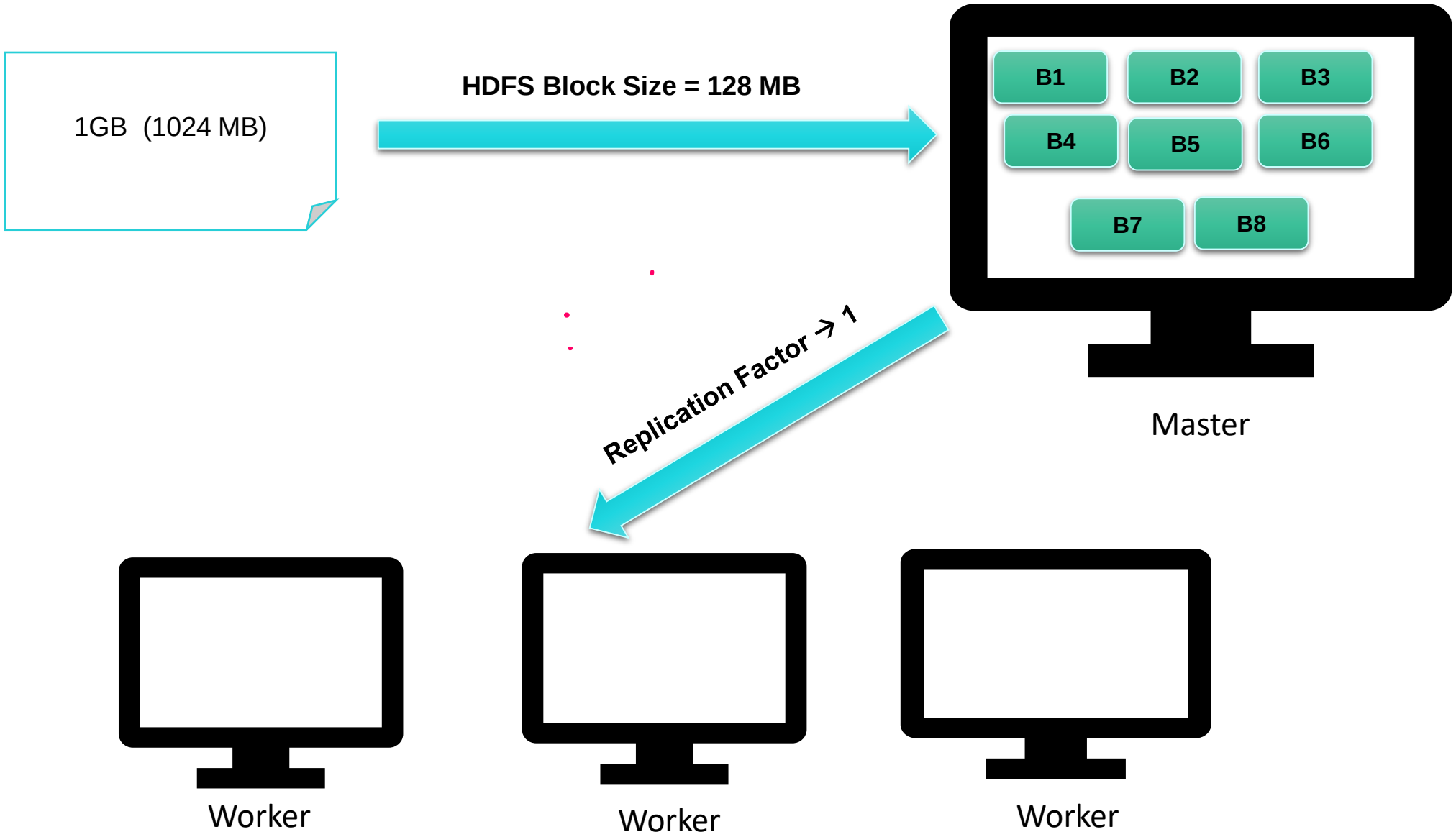
04 Google Cloud Storage

05 Microsoft Azure Storage

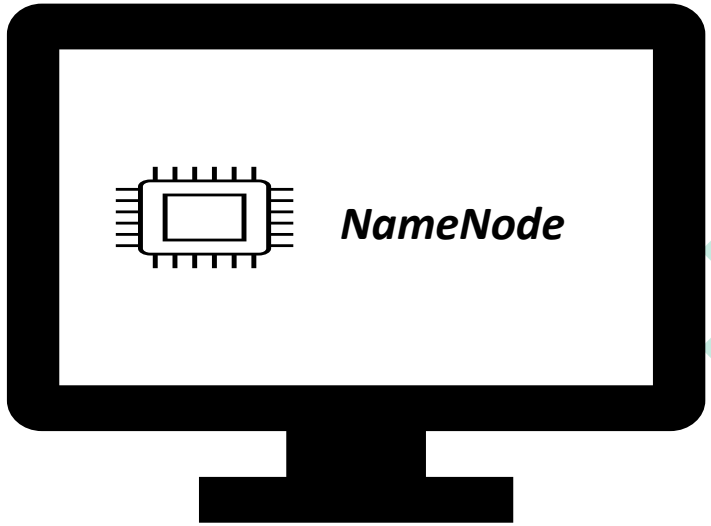
How files are stored in HDFS



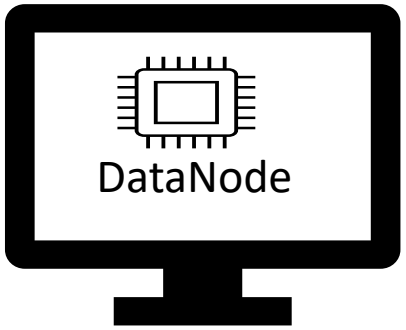
How files are stored in HDFS



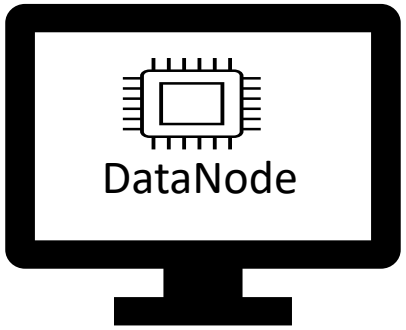
Master / Workers



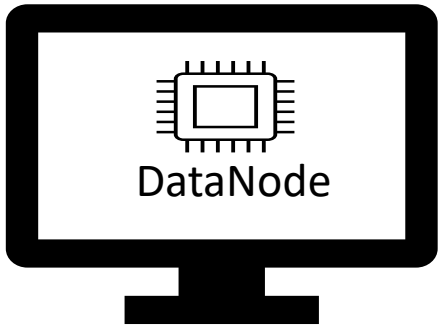
Master



Worker



Worker



Worker

NameNode

Entry point & accepts file read/write request

Keeps meta information about files

File name, permissions, block locations

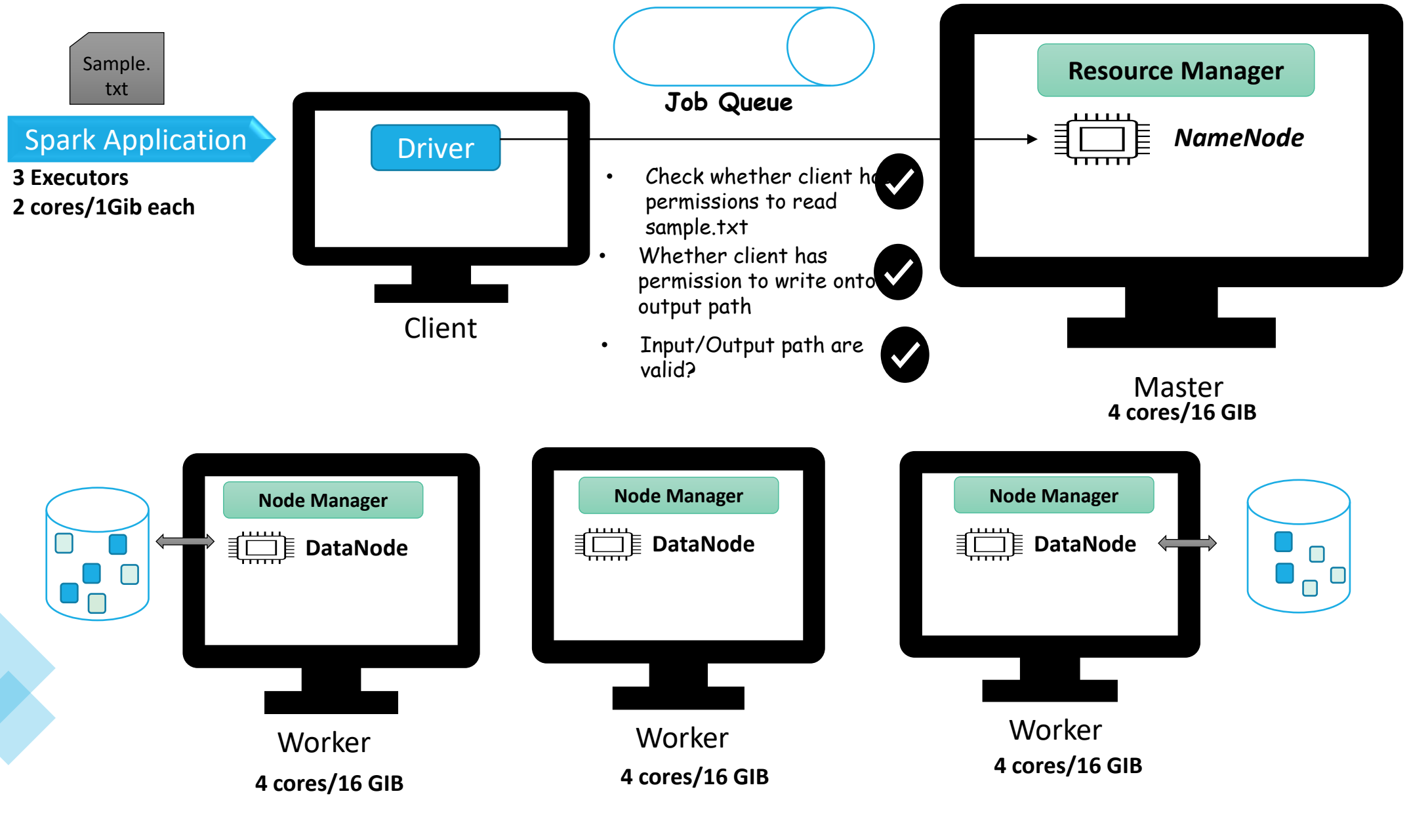
Data Nodes

Stores the actual data in forms of blocks

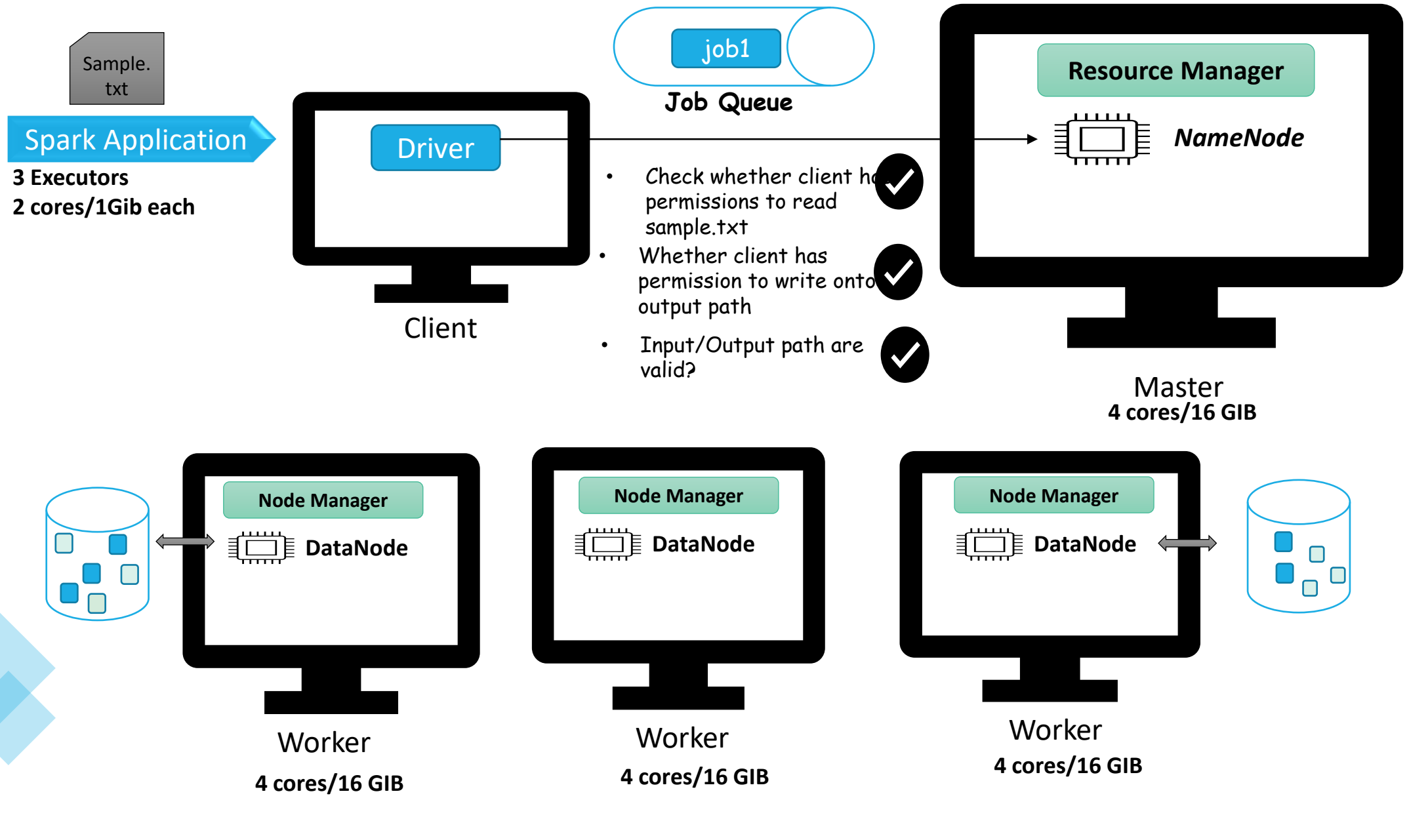
Multiple Data nodes per cluster

Perform Block Creation, deletion & replication

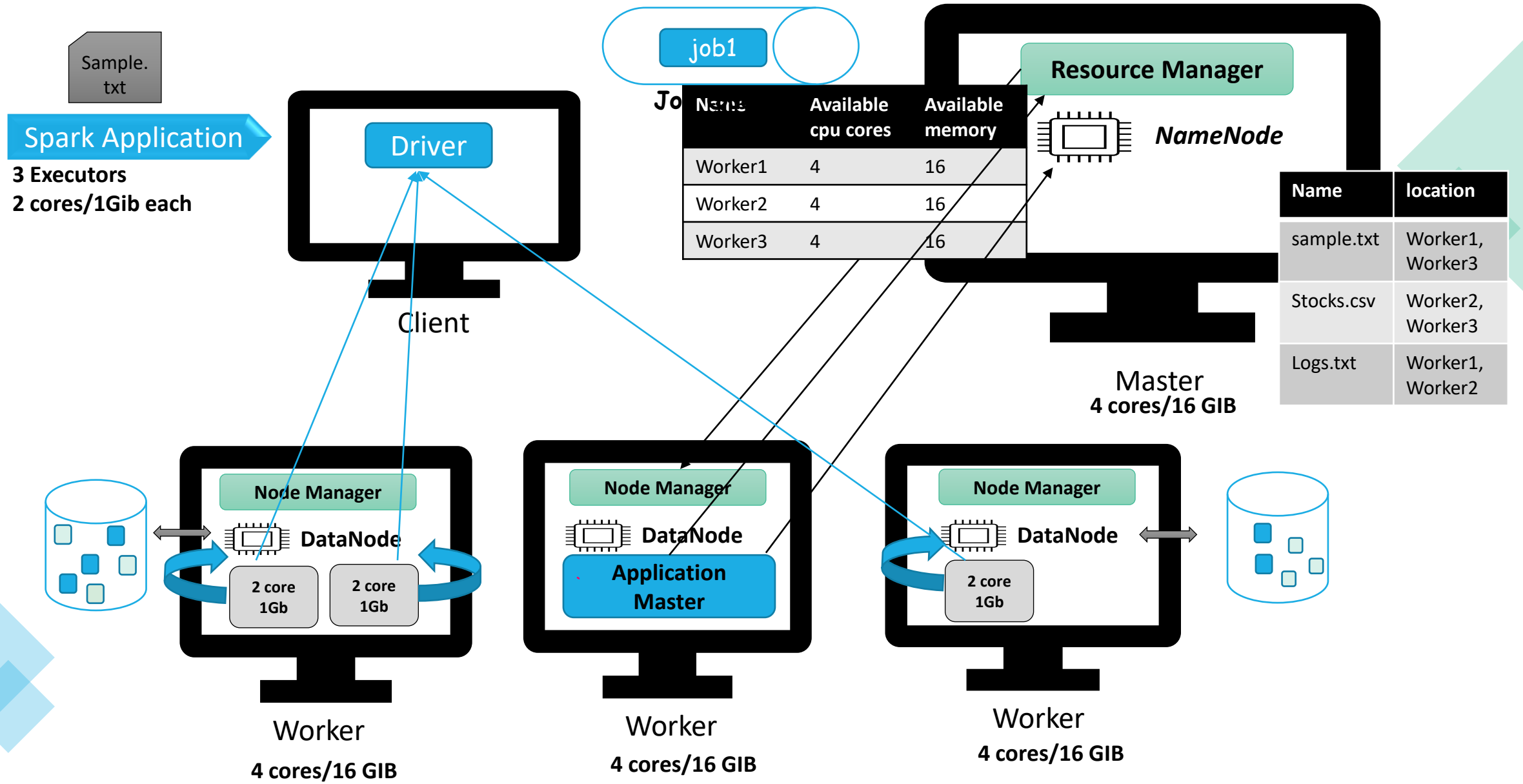
Running Spark on Cluster



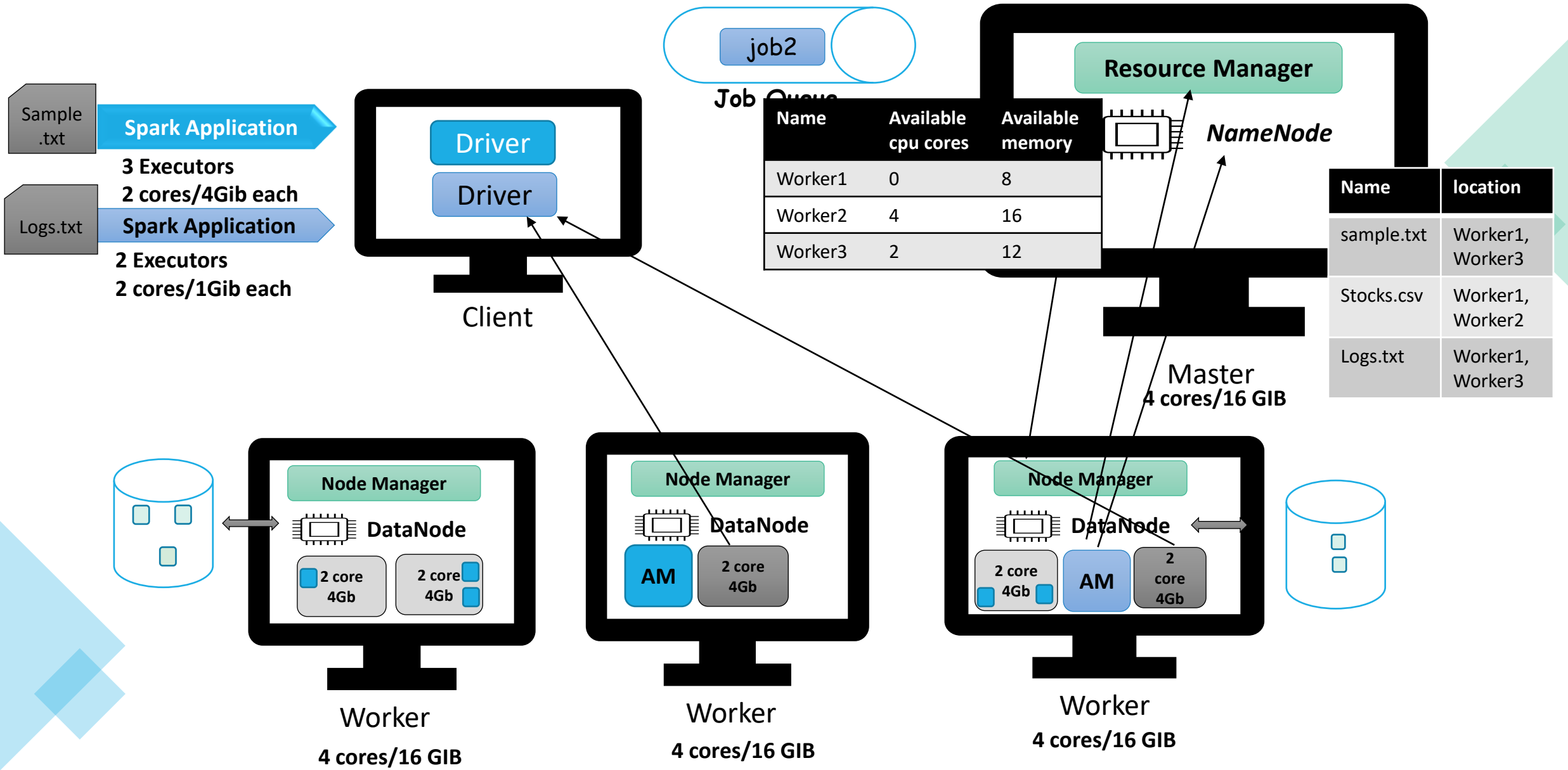
Running Spark on Cluster



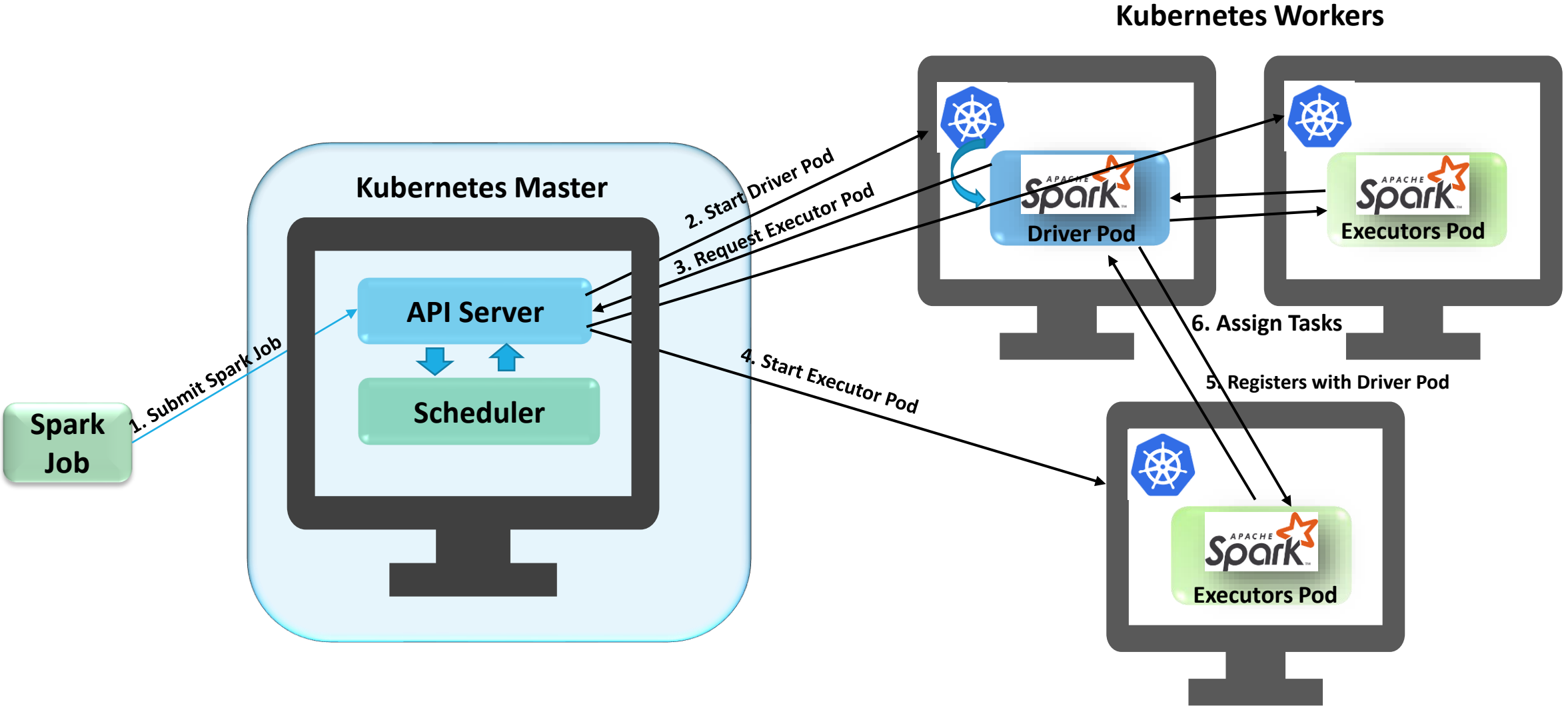
Running Spark on Cluster



Running Spark on Cluster



Spark on Kubernetes



Spark on Kubernetes?

Pros

- Easy Dependency Management
- Run Different Spark Versions Jobs
- Run Spark & non-Spark Application
- Cloud Agnostic

Cons

- Lot to setup & maintain
- Learning curve of K8s is high

Spark Submit on Kubernetes

```
./bin/spark-submit \
    --master k8s://https://<k8s-apiserver-host>:<k8s-apiserver-port> \
    --deploy-mode cluster \
    --name spark-pi \
    --class org.apache.spark.examples.SparkPi \
    --conf spark.executor.instances=5 \
    --conf spark.kubernetes.container.image=<spark-image> \
    local:///path/to/examples.jar
```

Spark Context Vs Spark Session

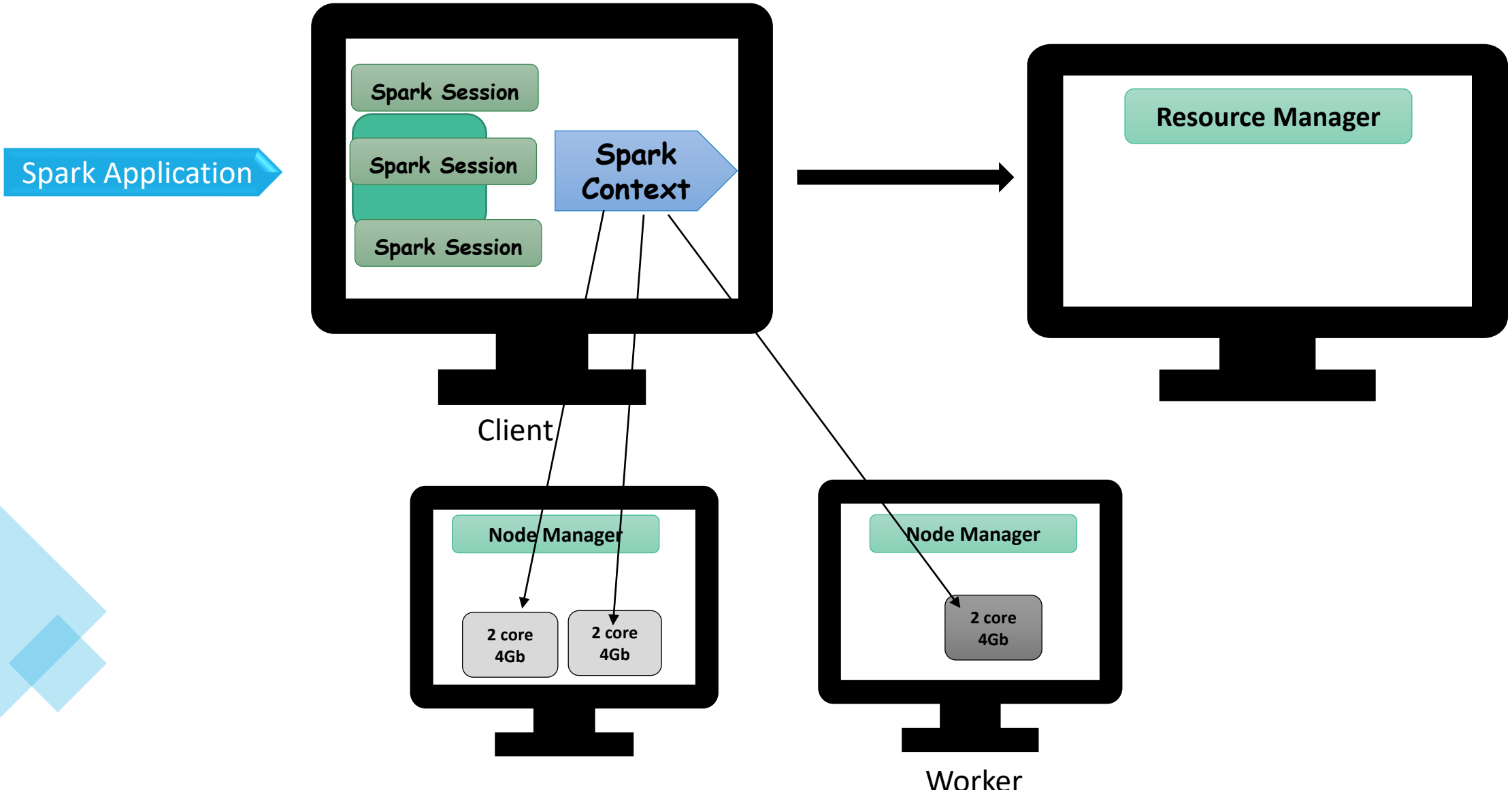
Spark Session is an entry point and is a wrapper over.

- Spark Context
- SQL Context
- Streaming Context
- Hive Context

Single Spark Session per application spark.driver.allowMultipleContexts

Every Spark session can set its own properties

Every Spark session can have its own tables unless created as global view



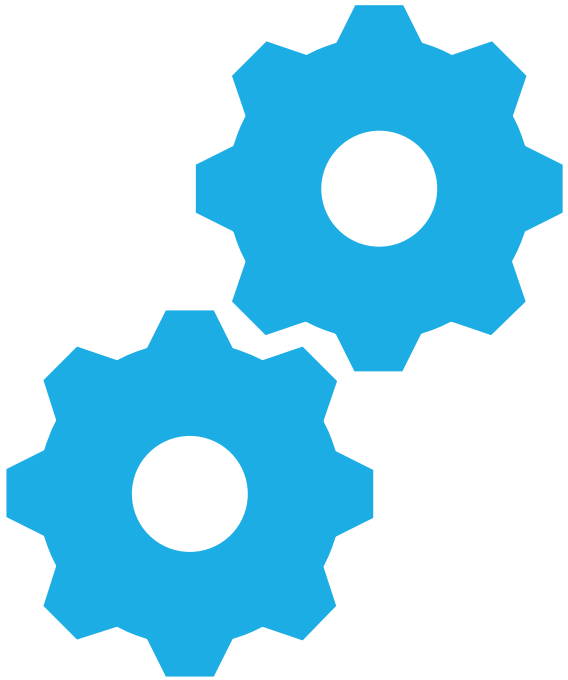
Deploy-Mode

Client Mode

- In client mode, the driver runs in the client process, and the application master is only used for requesting resources from YARN

Cluster Mode

- In cluster mode, the Spark driver runs inside an application master process which is managed by YARN on the cluster, and the client can go away after initiating the application.



Local Mode

Local[N] – Spark runs locally as a multi-threaded application with no cluster manager.

Local[1] – means only driver program with no executor. No parallelism.

Local[n] – means one driver & n-1 executors locally running on multiple threads.

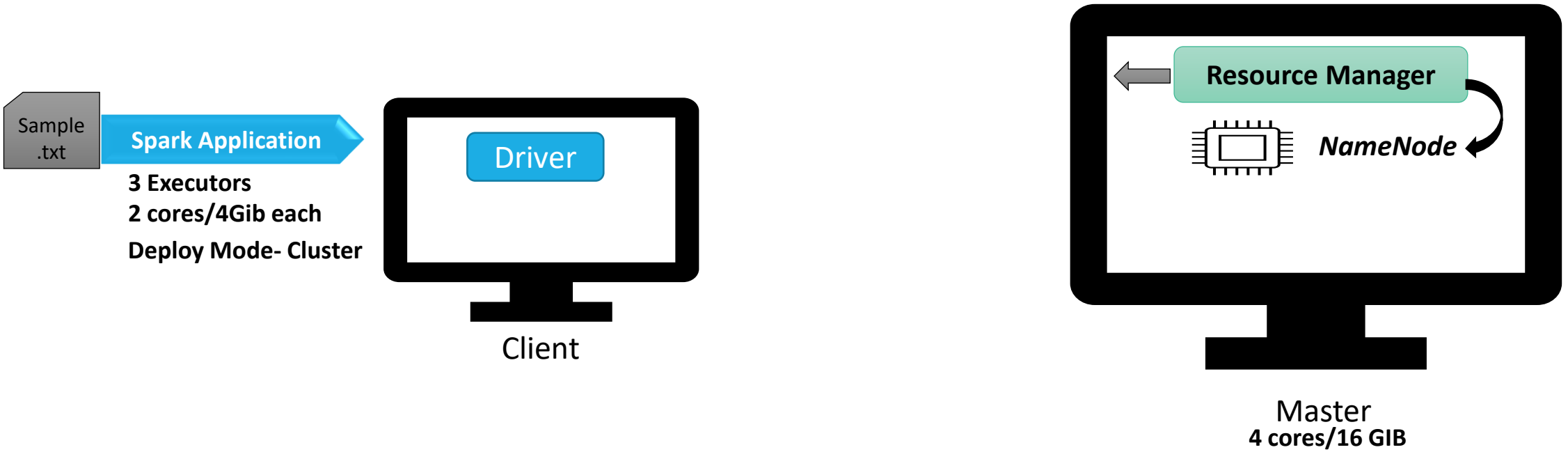
Client Mode

- Designed for interactive programming such as spark-shell & notebook.
- Driver runs on client machine from where you execute your command interactively.
- As soon as you close shell or notebook, all the processes dies.
- Suitable for local development & testing.

Cluster Mode

- Everything runs on cluster – Driver/Executor
- Suitable for long running jobs & production environments.

Running Spark on Cluster



Spark Application

Driver

```
object SparkDriver extends Serializable{

  def main(args: Array[String]): Unit = {

    val spark = SparkSession.builder()
      .master( master = "local[3]")
      .config(readSparkConfig)
      .getOrCreate()

    val moviesDF = spark.read.option("multiLine",true).

    import spark.implicits._

    moviesDF
      .filter(!array_contains(moviesDF("genre"),
      .withColumn( colName = "name", upper(col( colName =
      .groupBy( col1 = "year").agg(collect_list( columnName
      .write.json( path = "data/output")

    scala.io.StdIn.readLine()

  }
```

Logical Plan

Physical Plan

ResourceManager

NameNode

Master

4 Cores Executor

DataNode

Worker

4 Cores Executor

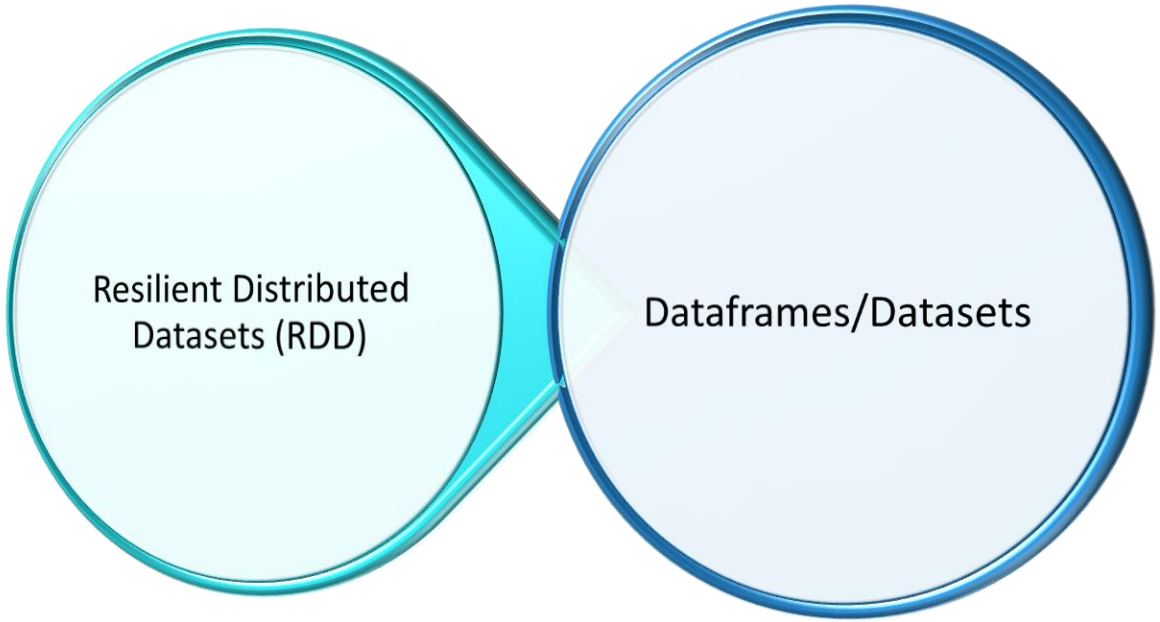
DataNode

Worker

DataNode

Worker

Data Structures



Operations

Transformations

Apply some operations and creates new dataset

Lazy Operation & computed only when action encountered

Actions

Triggers the execution

Returns the result to driver

Wide / Narrow Transformations

Narrow Transformations

Does not require data shuffling

Wide Transformations

Requires data shuffling

Creates new stage

Job/Stages/Tasks

```
spark
.read
.json("input_path")
.filter("rating>3.0")
.groupBy("year")
.count()
.write
.json("output_path")
```

Job1

Stage1

Map partitions
Create dataframe

Tasks

=number of partitions
of input_file

Job2

Stage1

Tasks

=number of partitions
of input_file

Stage2

Tasks

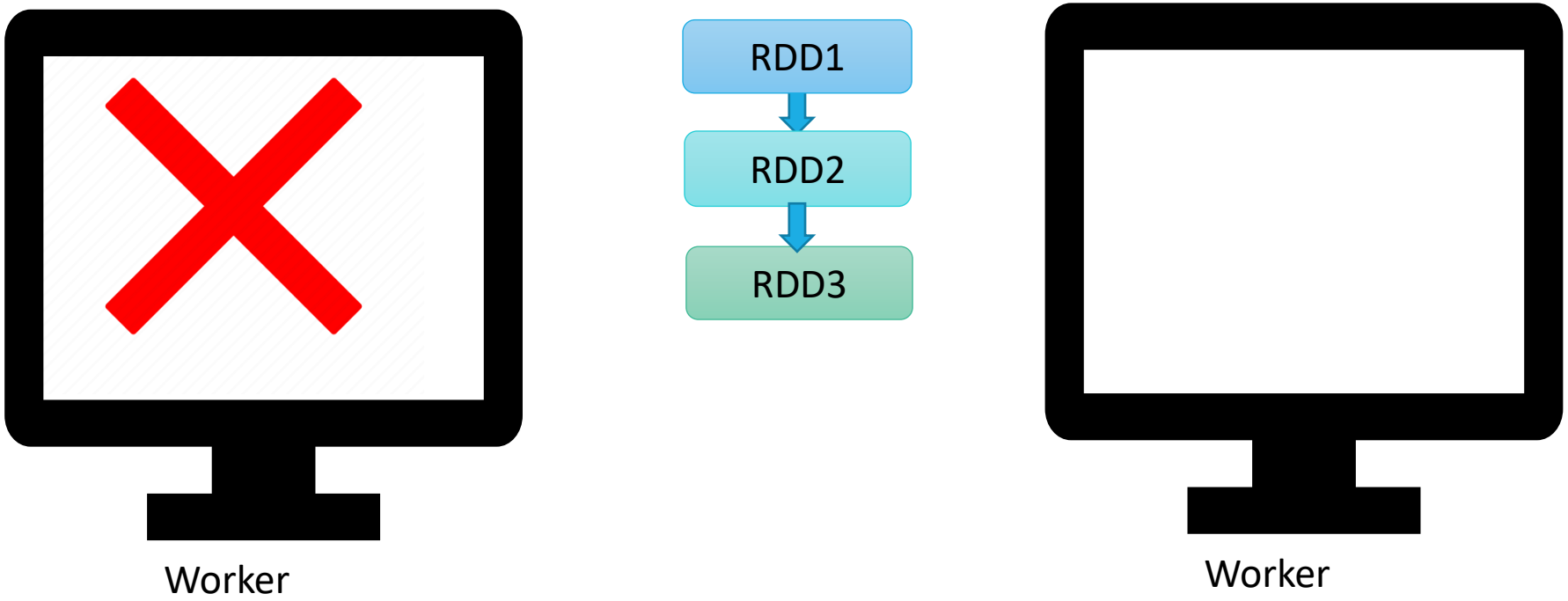
=spark.sql.shuffle.partitions
Default value = 200

Lineage & Fault Tolerance

Lineage are steps involved in the creation of RDD/DF.

Logical plan created by Catalyst Optimizer

It is used to achieve fault tolerance.

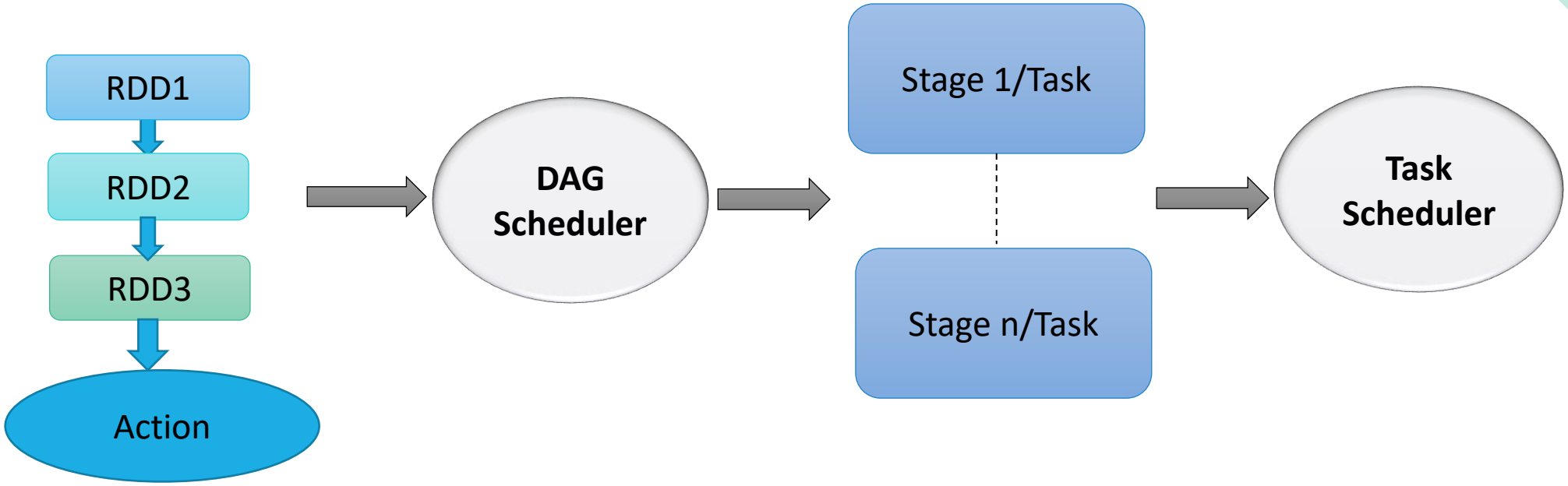


DAG

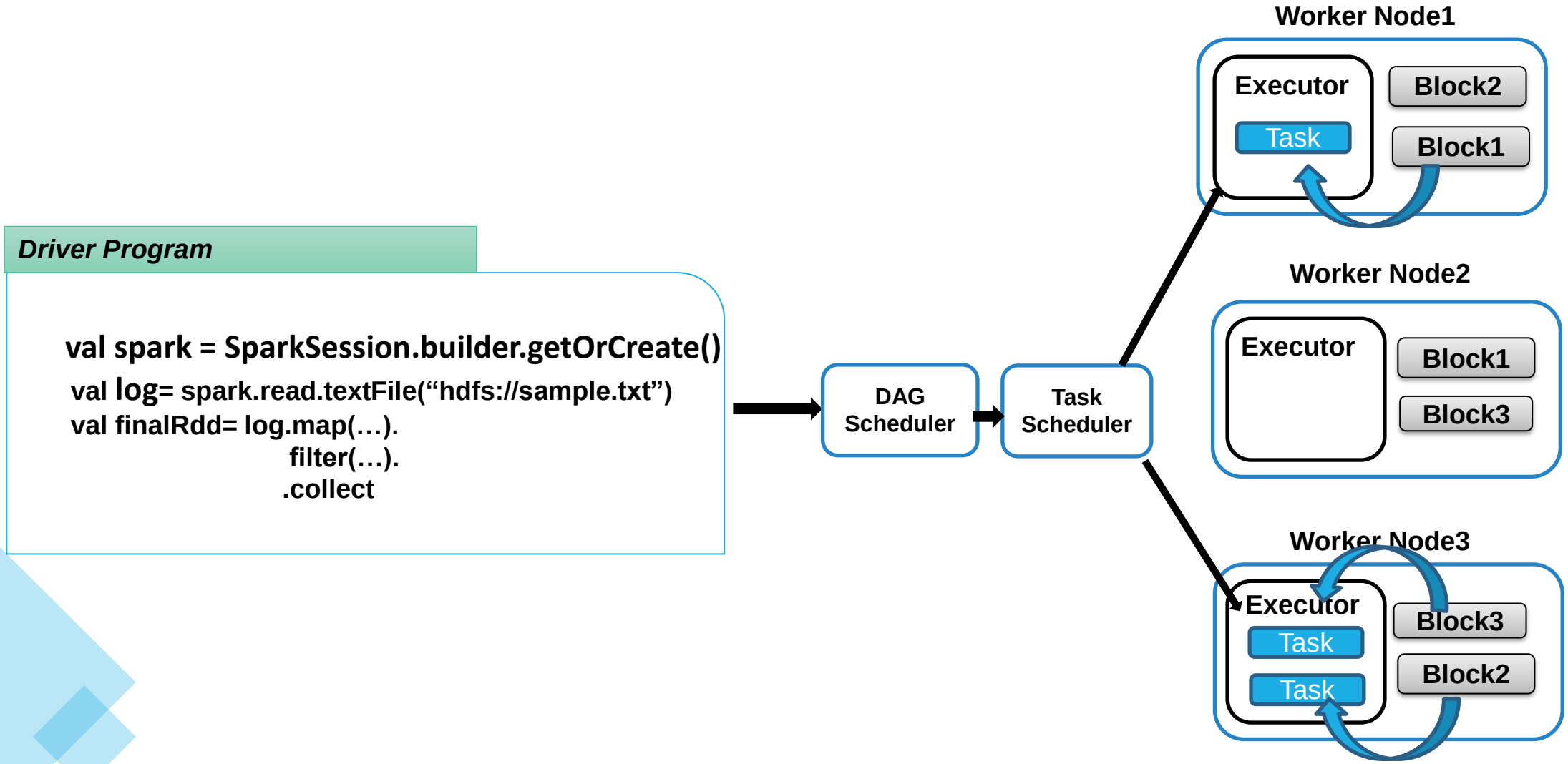
DAG is a physical plan.

It depicts the stages, tasks involved.

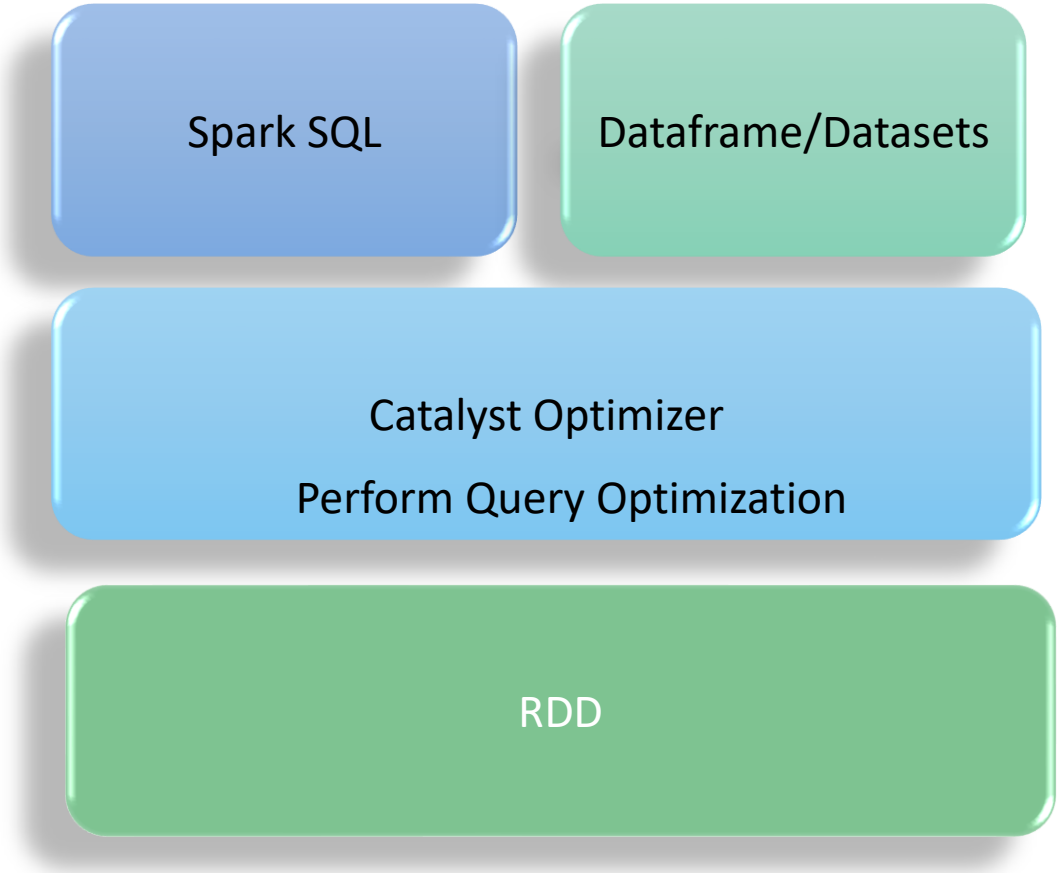
More info than logical plan.



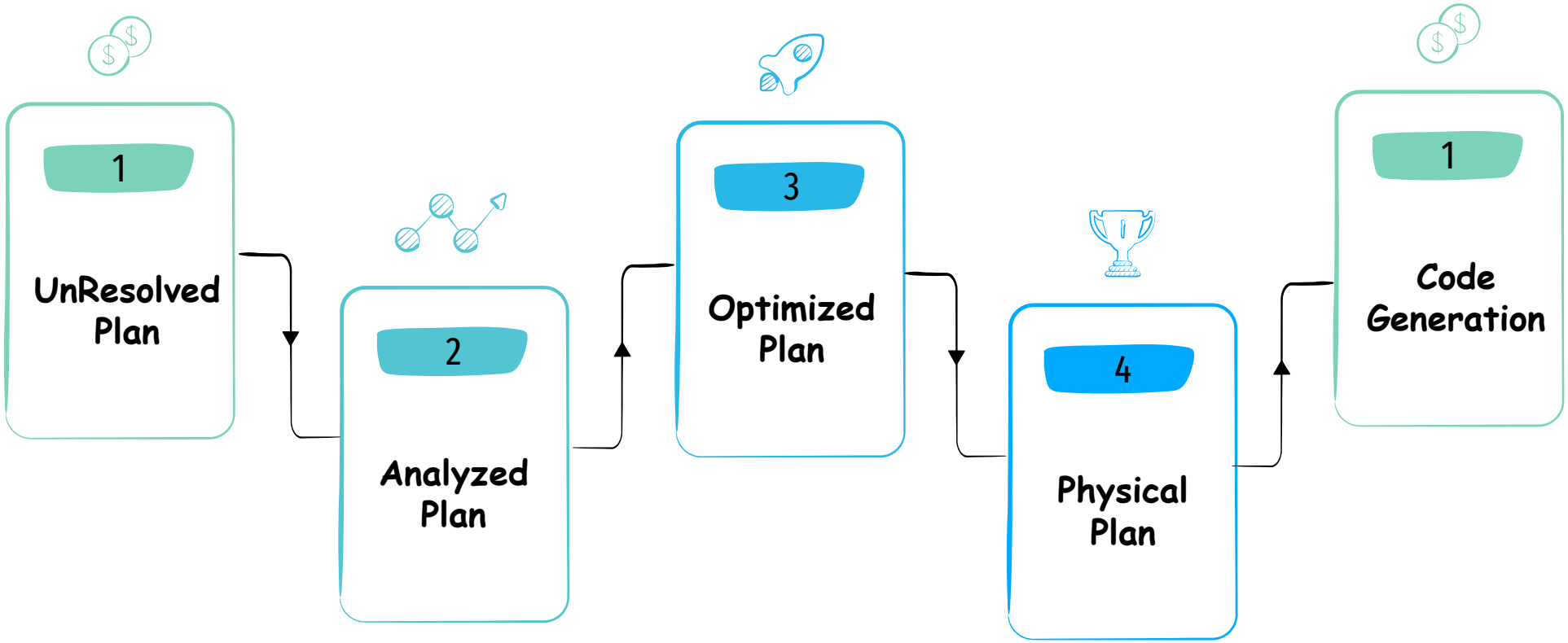
INDEPTH Cluster Overview



Catalyst Optimizer



Catalyst Optimizer

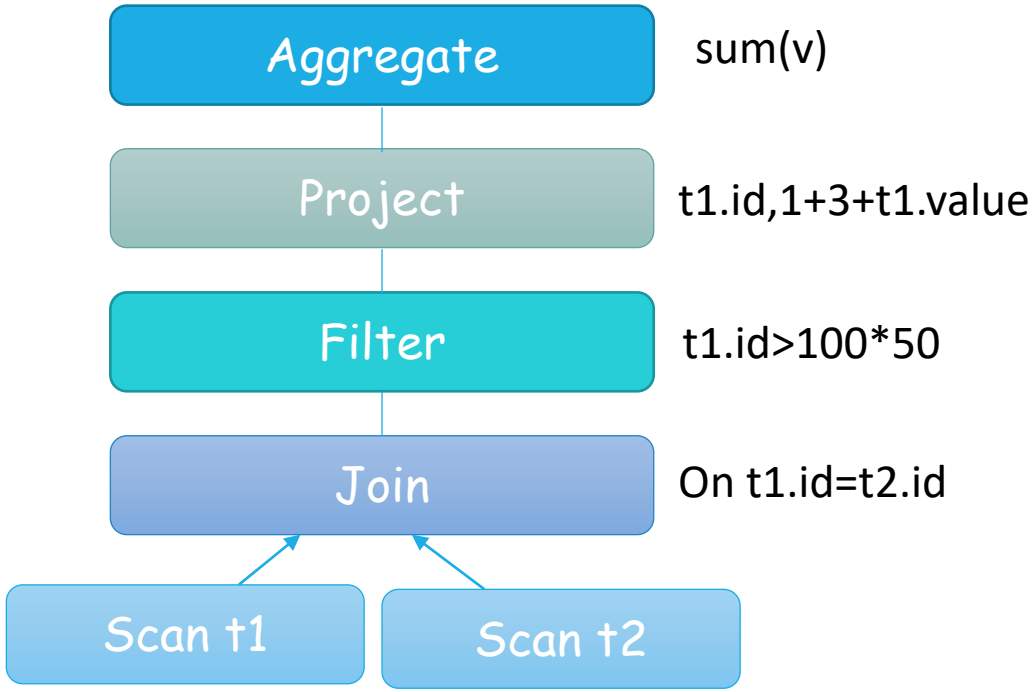


Catalyst Optimizer

Converts your code into a Tree

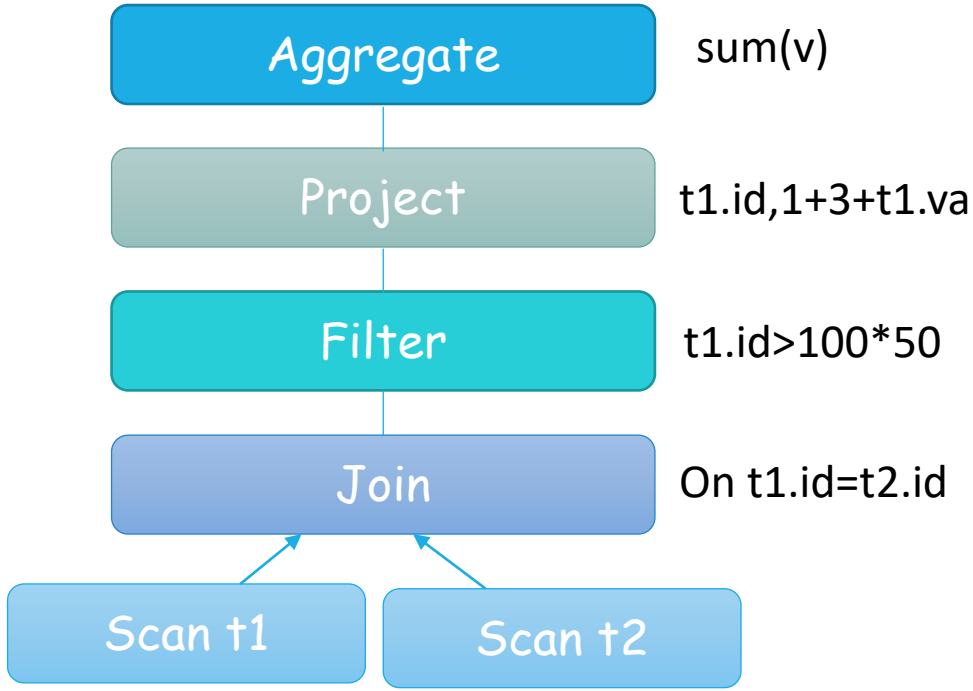
Step1

```
SELECT sum(v)
FROM (
   SELECT
    t1.id,
    1+3+t1.value as v
    FROM t1 JOIN t2
    ON t1.id=t2.id
    WHERE t1.id>100*50) tmp
```

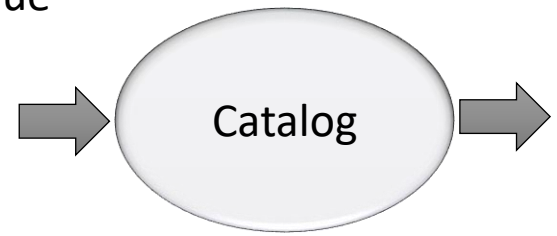


Parsed/ Unresolved Logical Plan

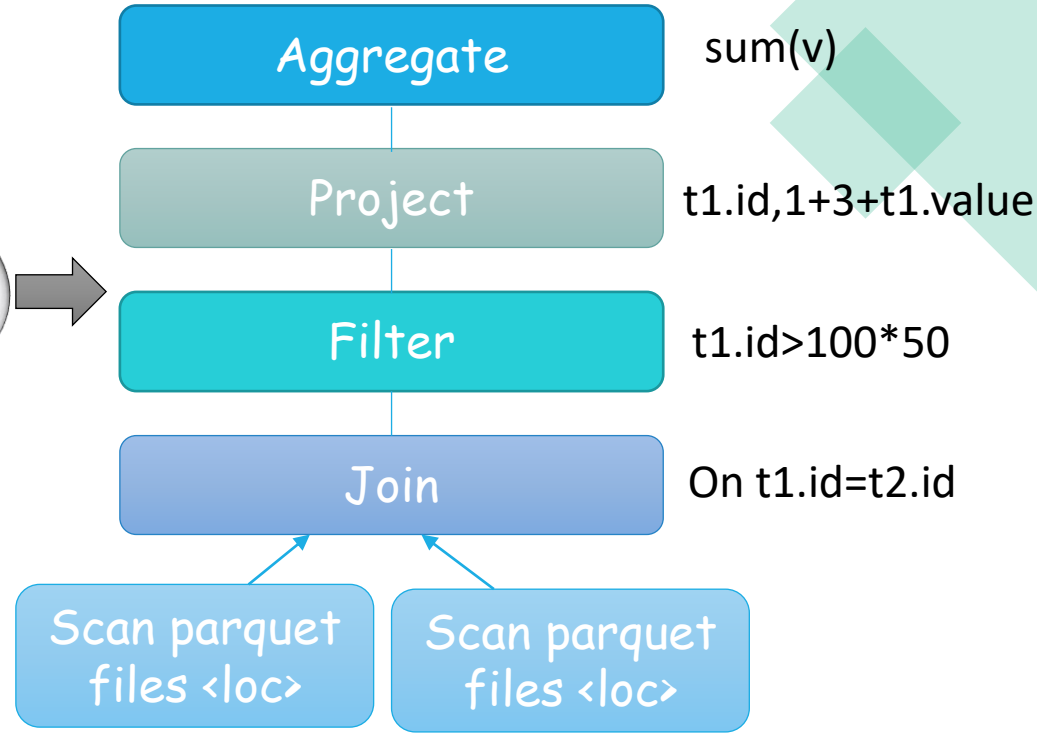
Step2



Unresolved Tree/ Unresolved Logical Plan

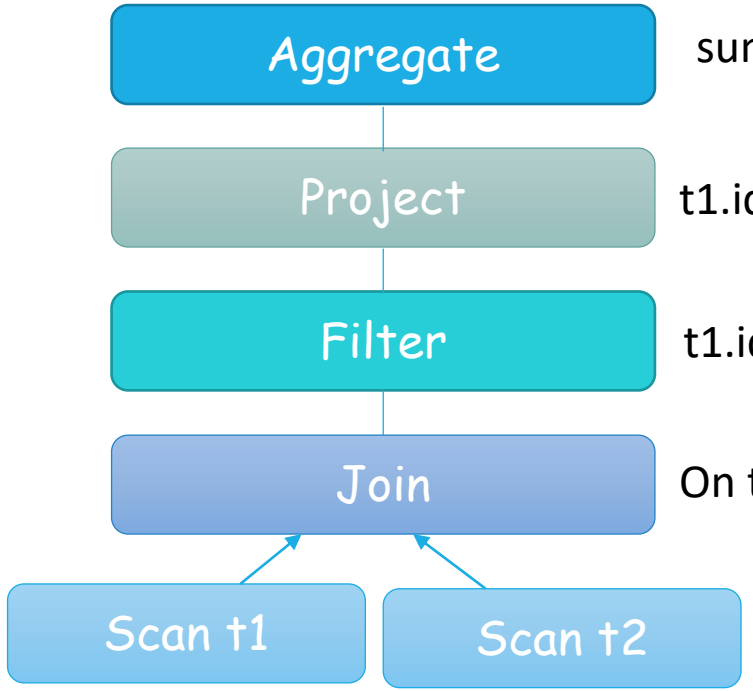


Check for data types
Check for validity



Analysed Logical Plan

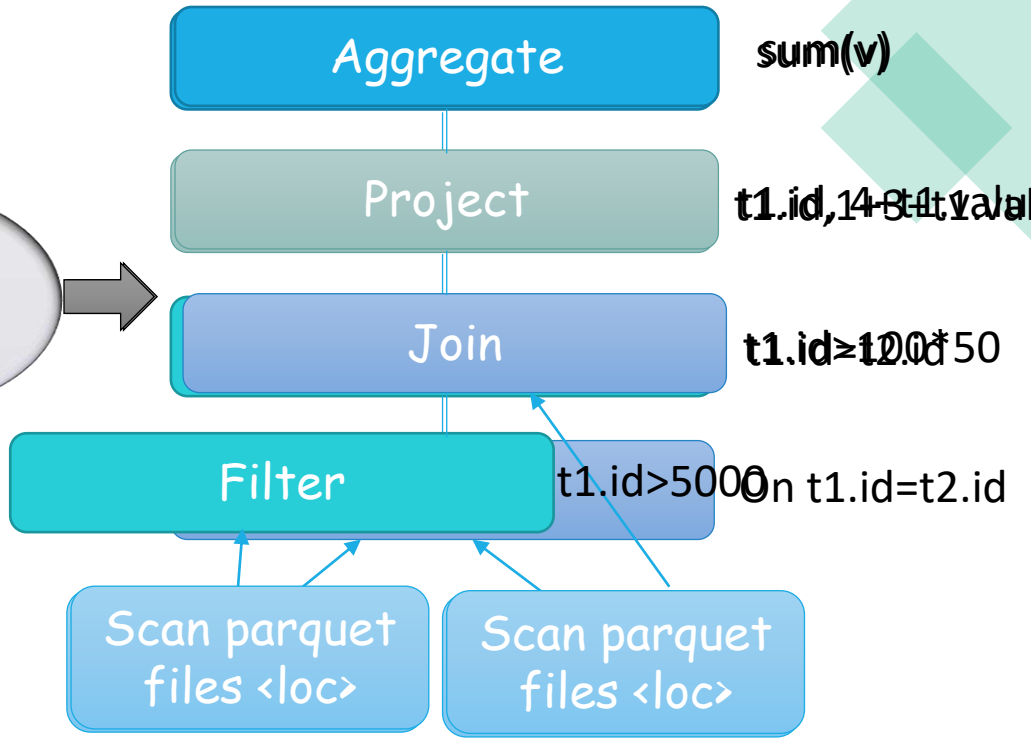
Step3



Unresolved Tree/ Unresolved Logical Plan

Catalog

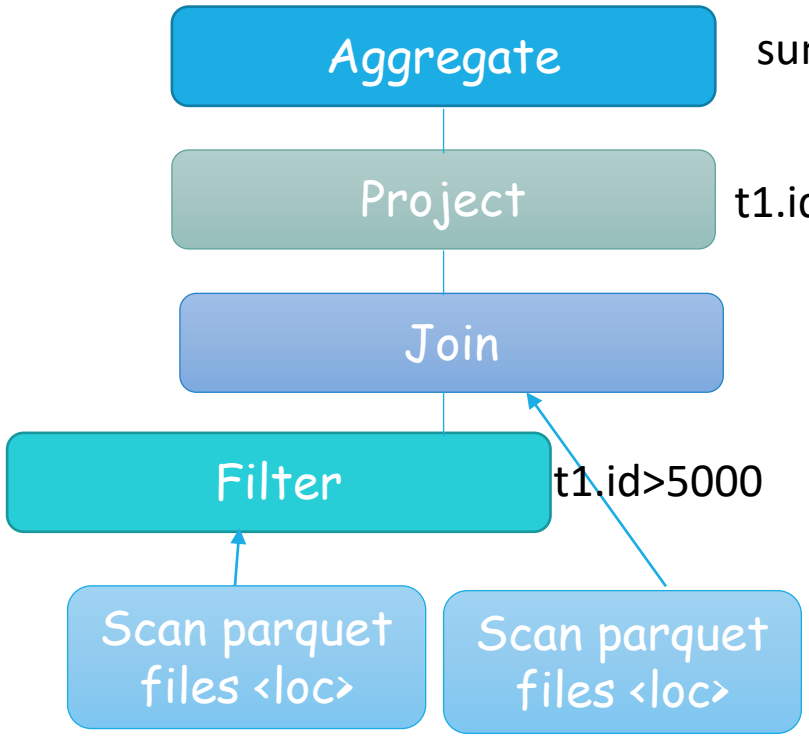
Check for data types
Check for validity



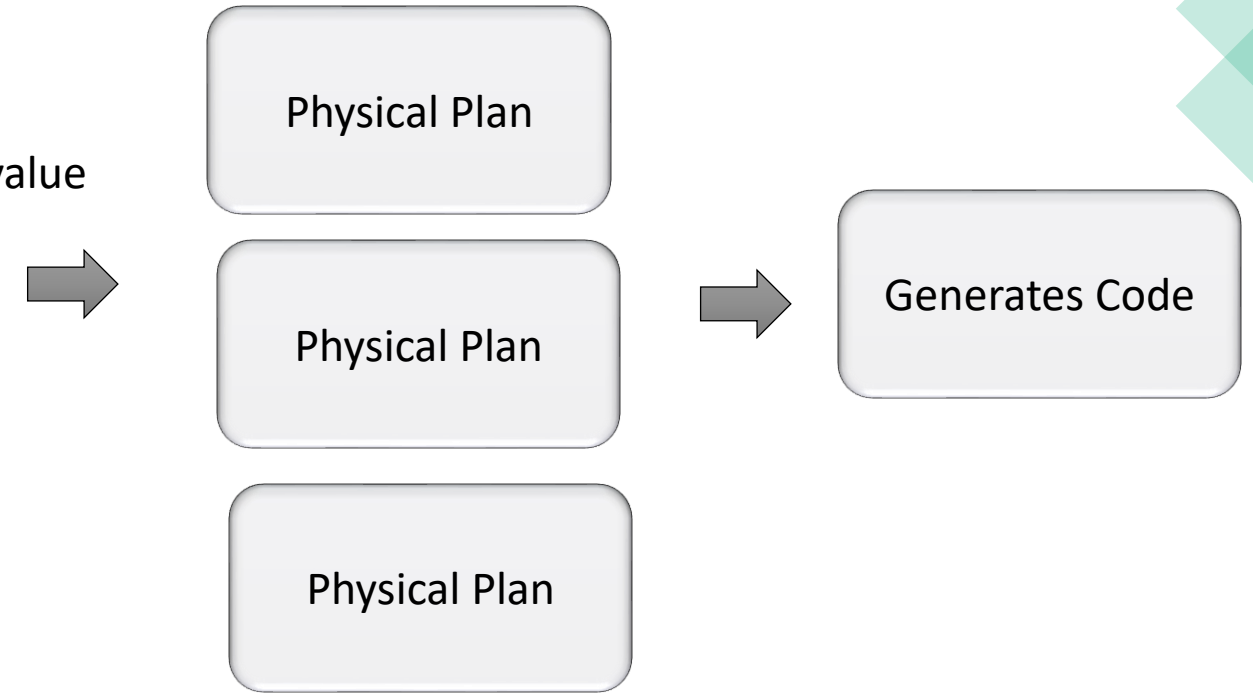
Analyzed Logical Plan

Optimized Logical Plan

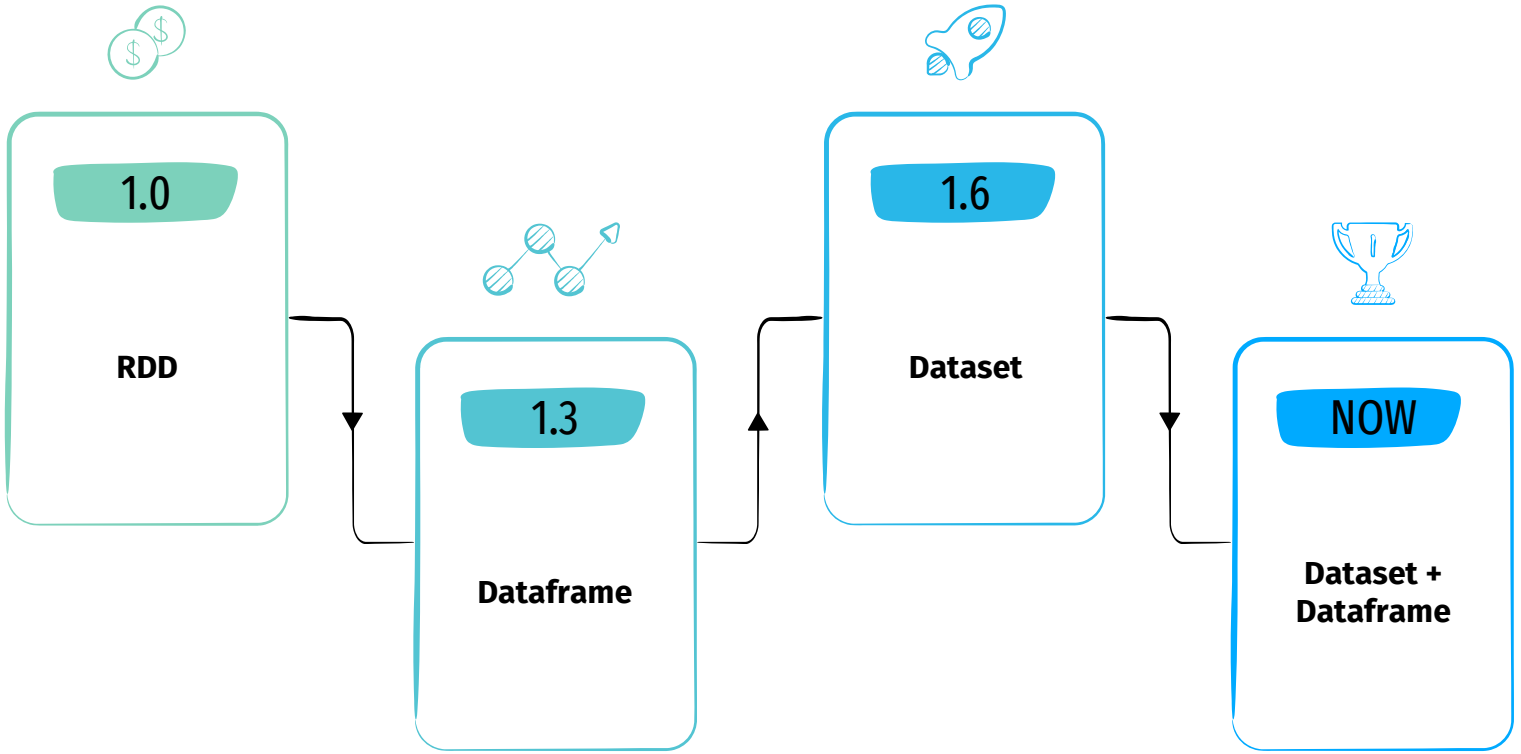
Step4



Optimized Logical Plan



Spark API Evolution



Spark Partitions

Dynamic Partitioning

Repartition	Coalesce
It cause complete shuffling of data.	It does not cause complete shuffling
It creates a new stage in execution plan	It does not create a new stage
It can increase or decrease partitions	It can only be used to decrease the number of partition

Optimal Partitions

Too less partitions

Worker nodes might be sitting idle resulting in less concurrency

Too many partitions

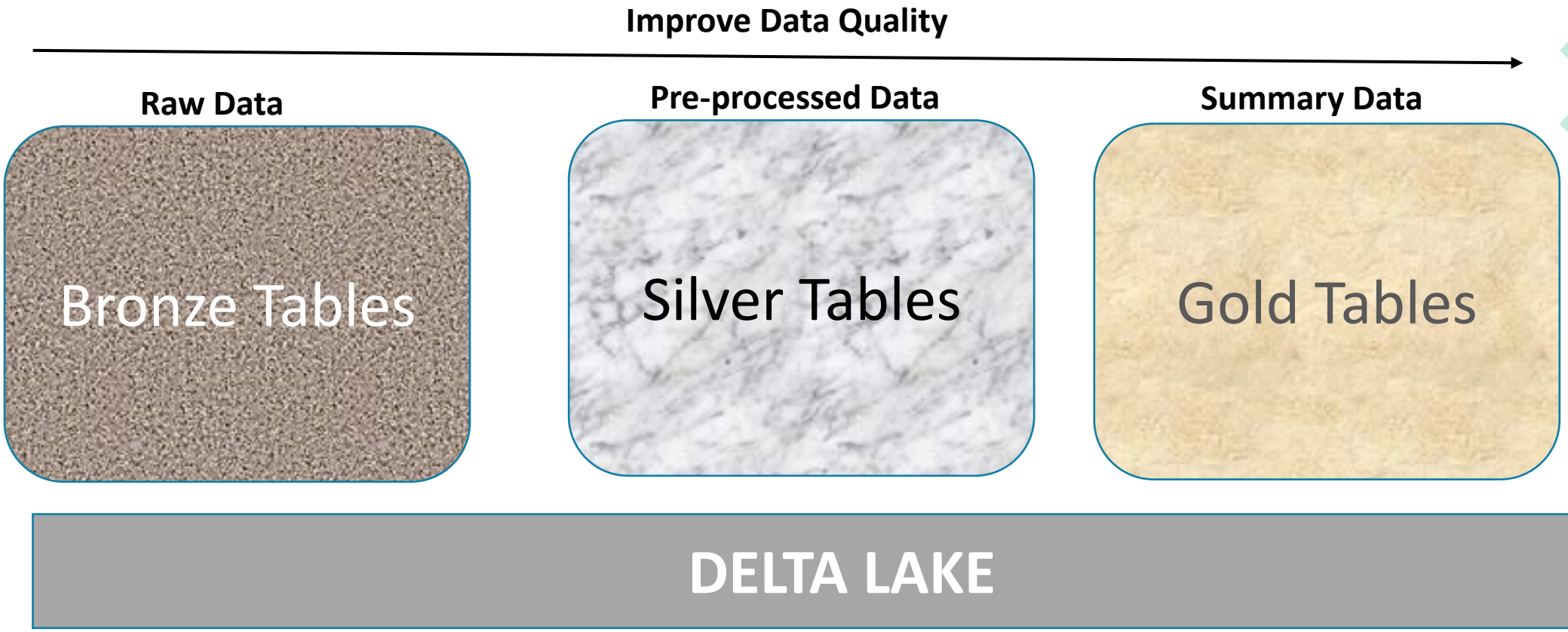
Task scheduling may take more time than actual execution time

Delta Architecture

Challenges with Data Lakes?

- Insert/Updates are challenging
- No Support for ACID
- Small File Problems
- Schema Validation/Evolution is not easy
- Time Travelling/Rollback is missing

Delta Lake


Improve Data Quality
Raw Data
Pre-processed Data
Summary Data
Bronze Tables
Silver Tables
Gold Tables
DELTA LAKE