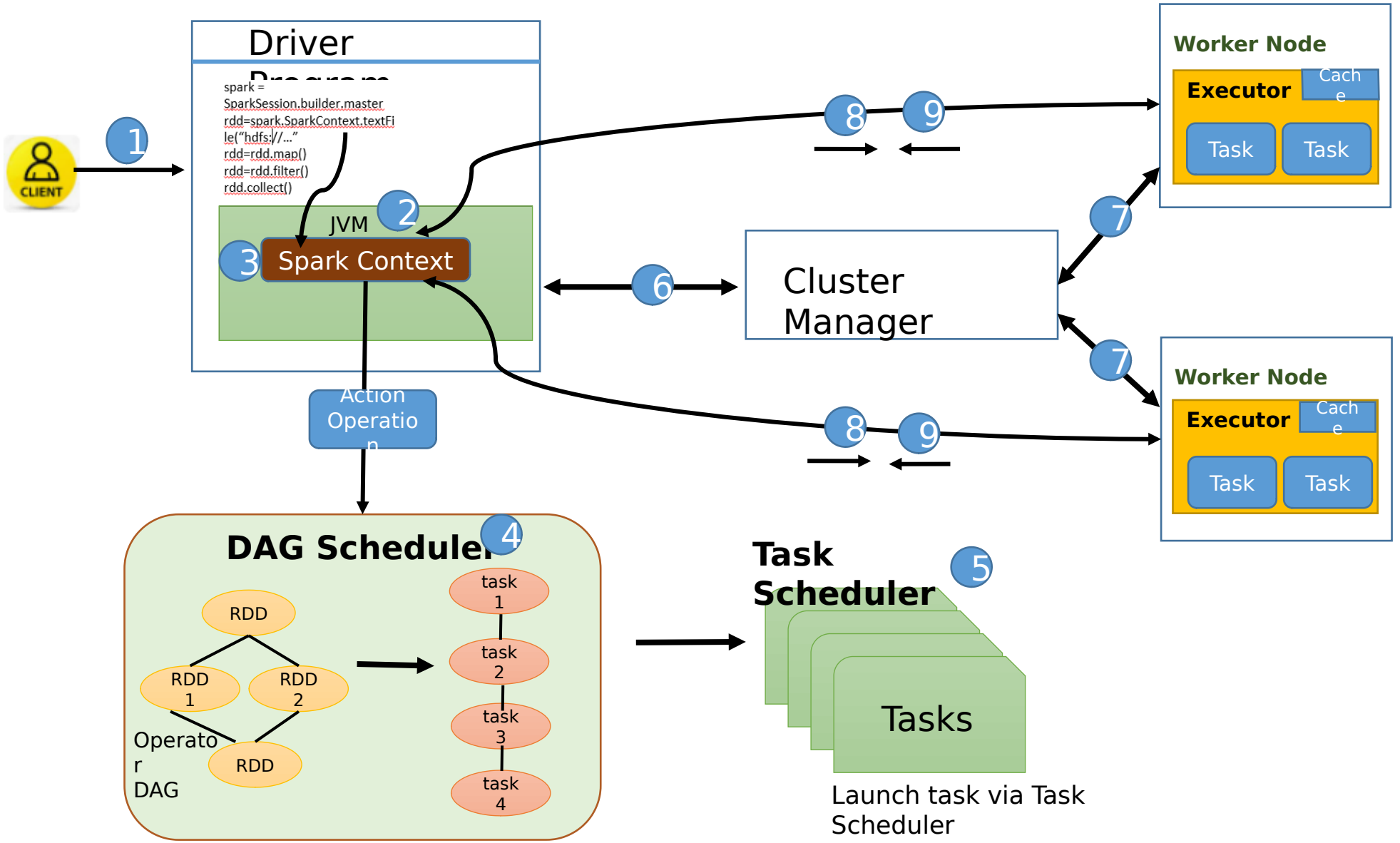
Spark Execution Architecture



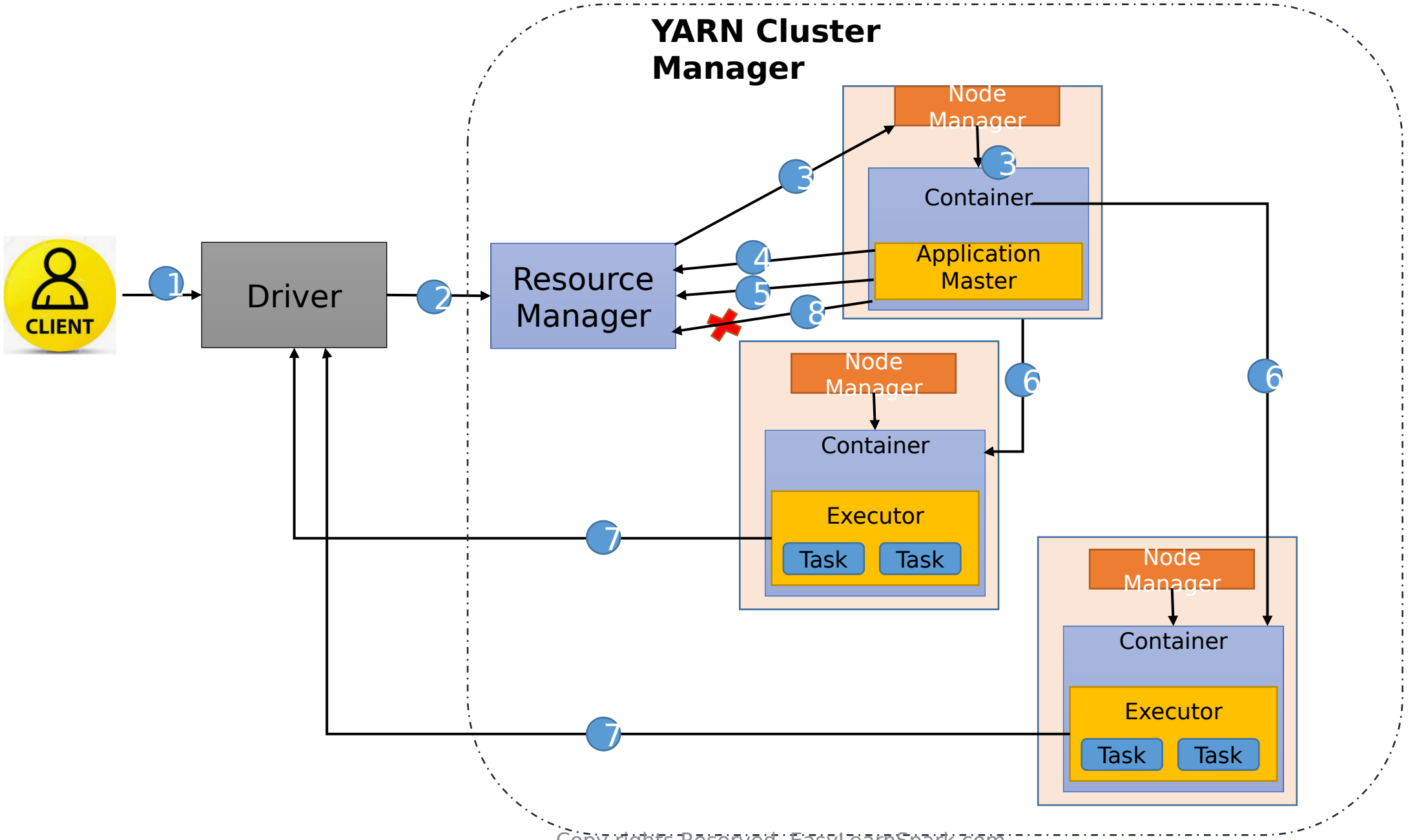
- Follows master-slave architecture.
- Client submits user application code to Driver. (1)
- JVM Is created. (2)
- Spark Context is created in the JVM of driver program. Only one active SC per JVM. (3)
- Driver implicitly converts the user code into logically DAG(Directed Acyclic Graph) using DAG scheduler. (4)
    - DAG Scheduler performs optimizations such as pipelining transformations and then it converts the logical graph DAG into Physical executing plan with many stages.
    - After creating physical executing plans, it creates Physical executing units called tasks under each stage.
- The stages pass on to Task Scheduler. It launces task through cluster manager. (5)
- Now driver via Spark Context talks to the cluster manager and negotiates resources. It request for worker nodes and executors in the cluster. (6) (7)
    - Spark is agnostic to the underlying cluster manager. As long as it can acquire executor processes, and these communicate with each other, it is relatively easy to run it even on a cluster manager that also supports other applications (e.g. Mesos/YARN).
    - Cluster Manager allocate resources and instruct executors to execute the job.
    - Also track the submitted jobs and report back the status of the job to the driver.
- Driver sends application code and dependencies(defined by jar or Python files passed to the SparkContext) to executors. (8)
- Finally driver also send the tasks to the executors to run.
- Job resources which we are trying to pass as part of execution can be cached at Worker Nodes. One of the job resource can be our code itself.
- All the executors start to register themselves with the drivers so that driver will have a complete view of the executors.
- Executors now start executing the tasks that are assigned by the driver program.
- When application is running, the driver program will monitor the set of executors that runs. Driver also schedules the future tasks based on data placement.
- After execution, the result returns back to the Spark Context. (9)

YARN as Spark Cluster Manager:

- Yet Another Resource Negotiator for Hadoop 2.x.
- Cluster Management – Used for resource allocation and Scheduling.
- It has 3 major components –
  - ✓ Resource Manager
  - ✓ Node Manager
  - ✓ Application Master

Flow:

1. Client submit the Spark application. Driver instantiates SparkContext.
2. Driver talks to the cluster manager(YARN) and negotiates resources.
3. The YARN resource manager search for a Node Manger which will, in turn, launch an ApplicationMaster for the specific job in a container.
4. The ApplicationMaster registers itself with the resource Manager.
5. The ApplicationMaster negotiates containers for executors from the ResourceManager. Can request for more resources from RM.
6. The ApplicationMaster notifies the Node Managers to launch the containers and executors. Executor then executes the tasks.
7. Driver communicates with executors to coordinate the processing of tasks of an application.
8. Once the tasks are complete, ApplicationMaster un-registers with the Resource Manager.



Resource Manager:

- Runs on Master Node.
- Manages the resources used across the cluster.
- Two components – Scheduler and Application Manager.

Scheduler - Performs scheduling based on the requirement of resources by the applications.

Application Manager: It manages the running of Application Master and restart it on its failure. Also it is responsible to accept the submission of jobs.

Node Manager:

- Runs on all Worker Nodes.
- Launches and monitor containers which are assigned by RM.
- Responsible for the execution of the task in each data node.

Containers:

- Are set of resources like RAM, CPU, memory etc on a single node and they are scheduled by RM and monitored by NM.

Application Master:

- An individual ApplicationMaster is assigned for each job by RM.
- It’s chief responsibility is to negotiate the resources from the RM. It works with the Node Manager to monitor and execute the tasks.