DAG

DAG Stands for **D**irected **A**cyclic **G**raph.

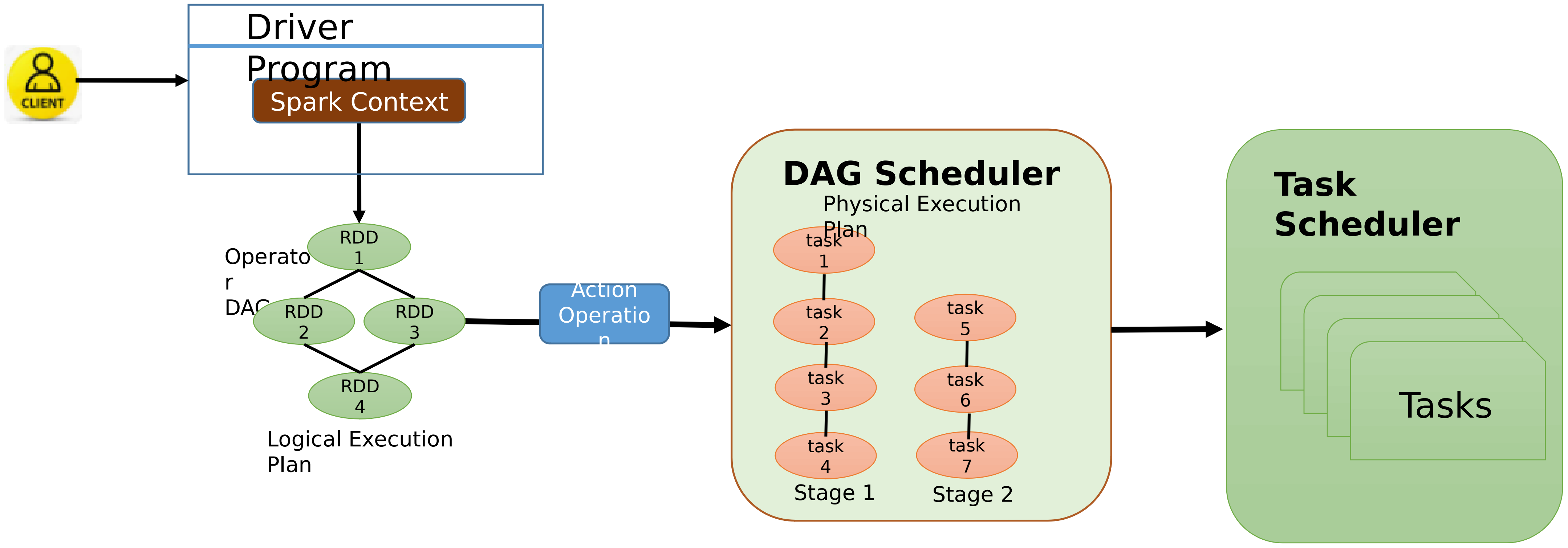
Directed ✉ Directly connected to one node to another.

Acyclic ✉ There is no cycle or loop. So it is in line and we can not go back to its original position.

Graph ✉ It has Vertices and Edges. Vertices indicates RDDs and edges refers to the operations on the RDD.

These all represented as a graph.

DAG Scheduler



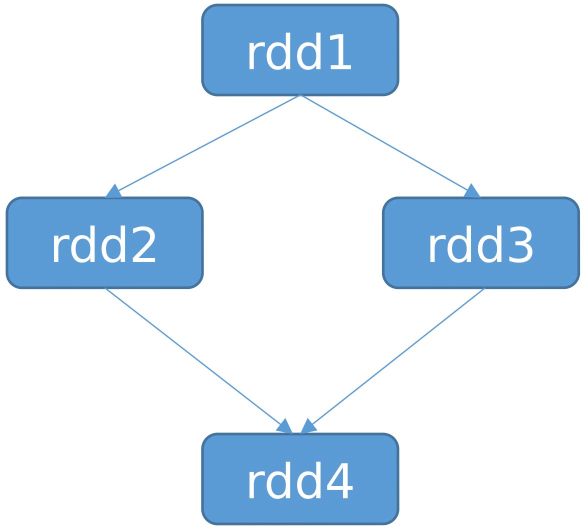
Steps to build a DAG

1. User submits a application job to spark.
2. Drivers takes the application and create a Spark Context to process the application.
3. Spark Context identifies all the T and A operations present in the application.
4. All the operations are arranged in a logical flow of operations called DAG (Logical Execution Plan).
5. It stops here if SC doesn’t find any A Operations.
6. If it identifies an A operations, spark submit the Operator DAG to DAG scheduler.
7. DAG Scheduler converts the Logical Execution plan into Physical Execution plan and creates stages and tasks. Here Narrow T are fused together into one stage. Wide T involving shuffle process creates new stages.
8. DAG scheduler bundles all the tasks and send it Task Scheduler which then submit the job to cluster manager for execution.

RDD Lineage

RDD Lineage:

- Each RDD maintains a pointer to one or more parent along with metadata about what type of relationship it has with the parent.
- Ex - if we call rdd2=rdd1.map(), the rdd2 keeps a reference to its parent rdd1 and this is called RDD lineage.
- Print the RDD lineage information using toDebugString() API.



RDD Lineage
Graph

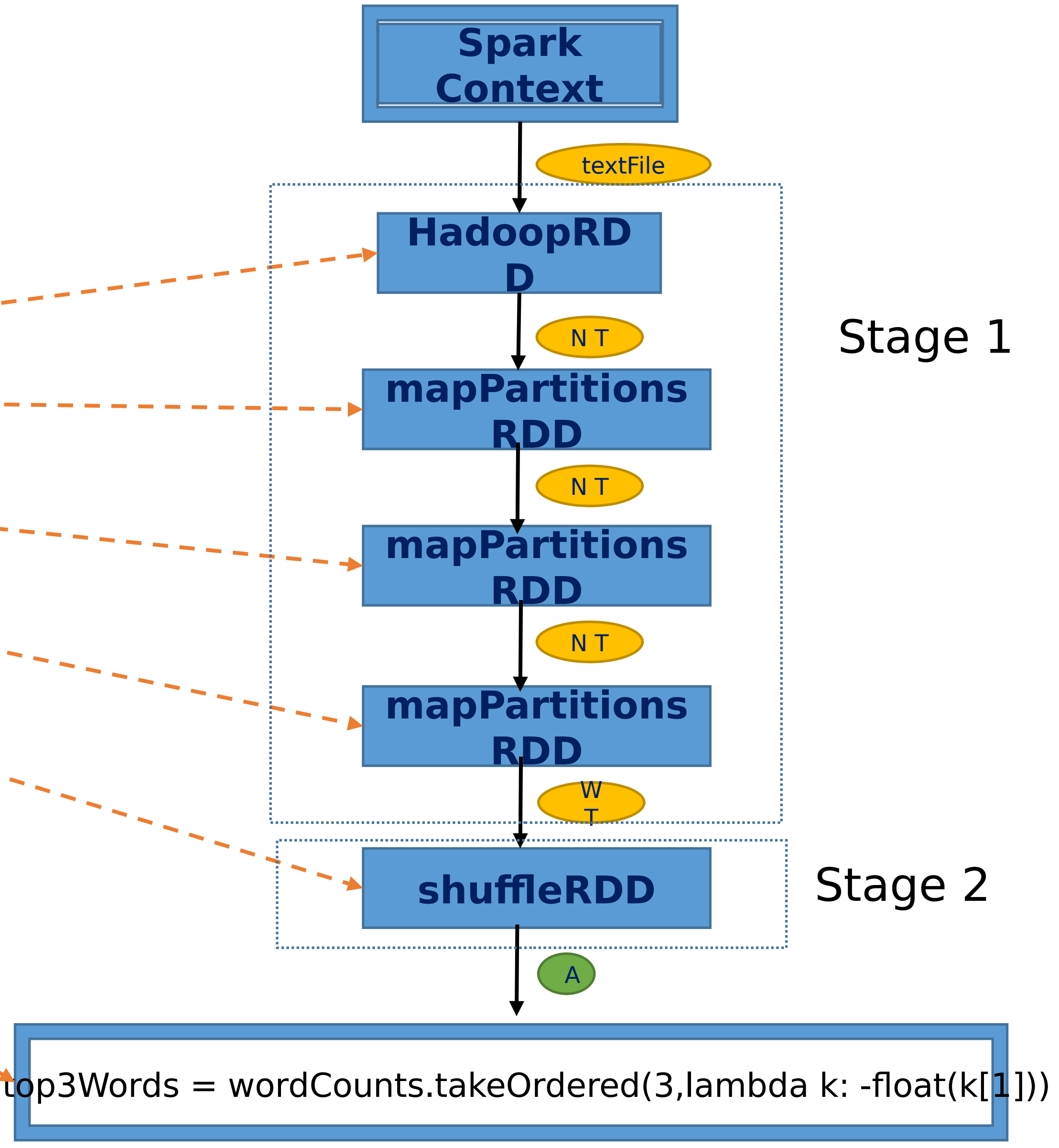
toDebugString(self) :

- Displays Logical Execution Plan.
- We can learn about a RDD Lineage Graph using API toDebugString.
- Displays the description of this RDD and its recursive dependencies for debugging.

Word Count Program:

```
text_file = sc.textFile('practice/retail_db/word')
wordCounts = text_file.flatMap(lambda line:
line.split(",")) \
.filter(lambda x : x.isdigit() == False) \
.map(lambda word: (word, 1)) \
.reduceByKey(lambda a, b: a + b)
top3Words = wordCounts.takeOrdered(3,lambda k: -
float(k[1]))

for i in rdd.toDebugString().split("\n") : print(i)
```



```
top3Words = wordCounts.takeOrdered(3,lambda k: -float(k[1]))
```