Spark Resilient Distributed Dataset (RDD)

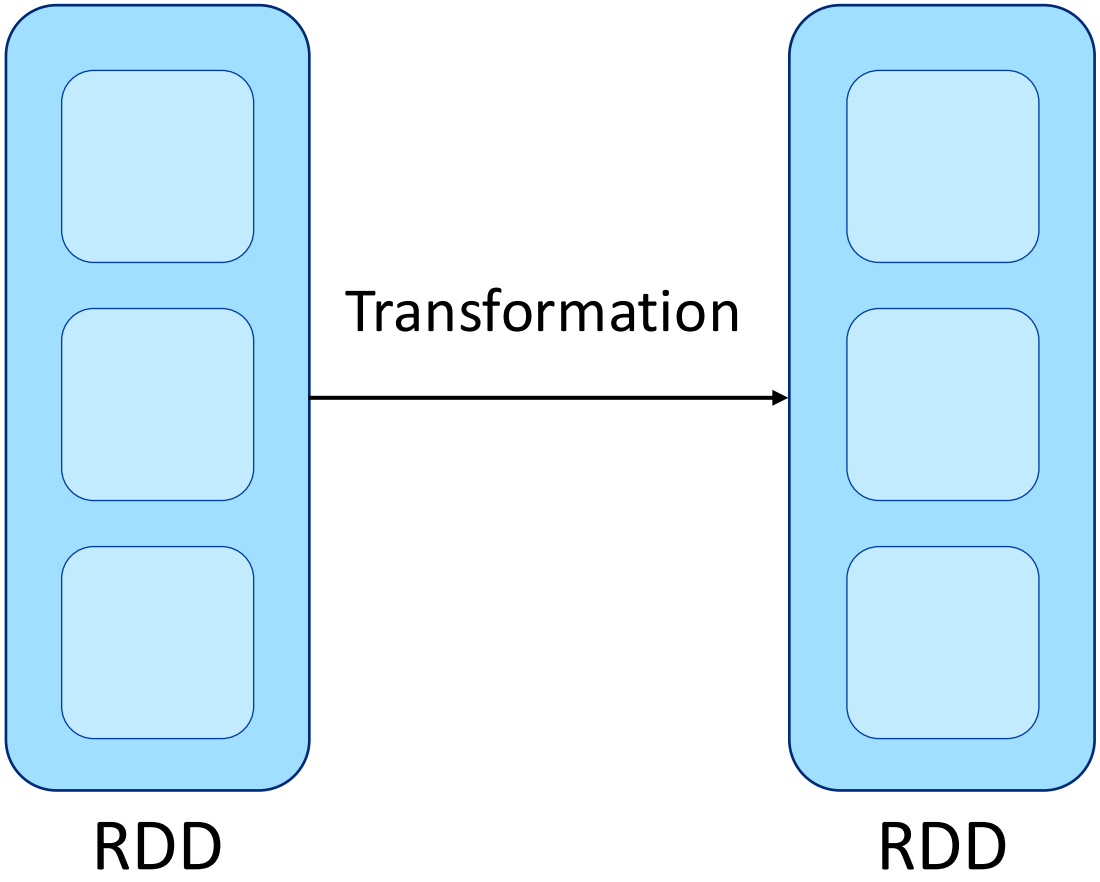
Spark

- A memory-based big-data framework
- Resilient Distributed Dataset (RDD) is an alternative query processing framework for big-data
- Utilizes more memory to speed up query processing

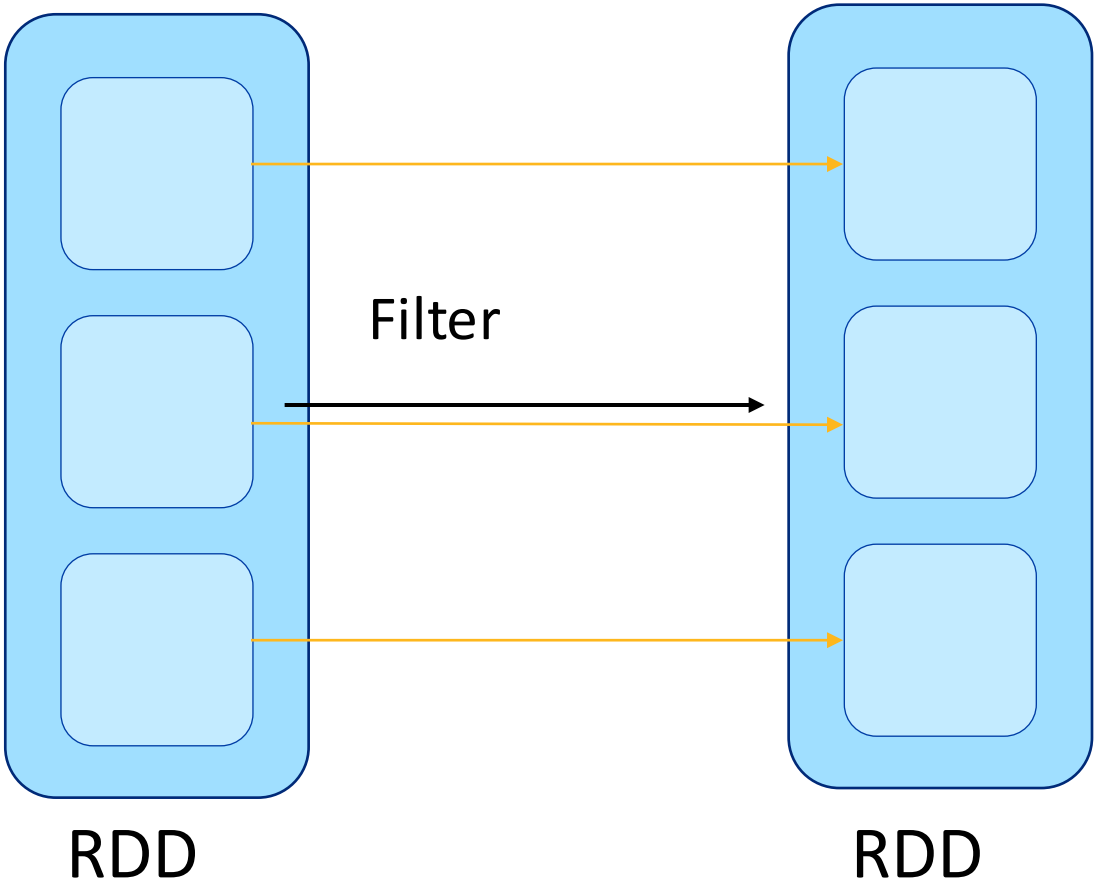
RDD Abstraction

- RDD is a pointer to a distributed dataset
- Stores information about how to compute the data or where the data is
- Transformation: Converts an RDD to another RDD
- Action: Returns an answer of an operation over an RDD

RDD Transformation

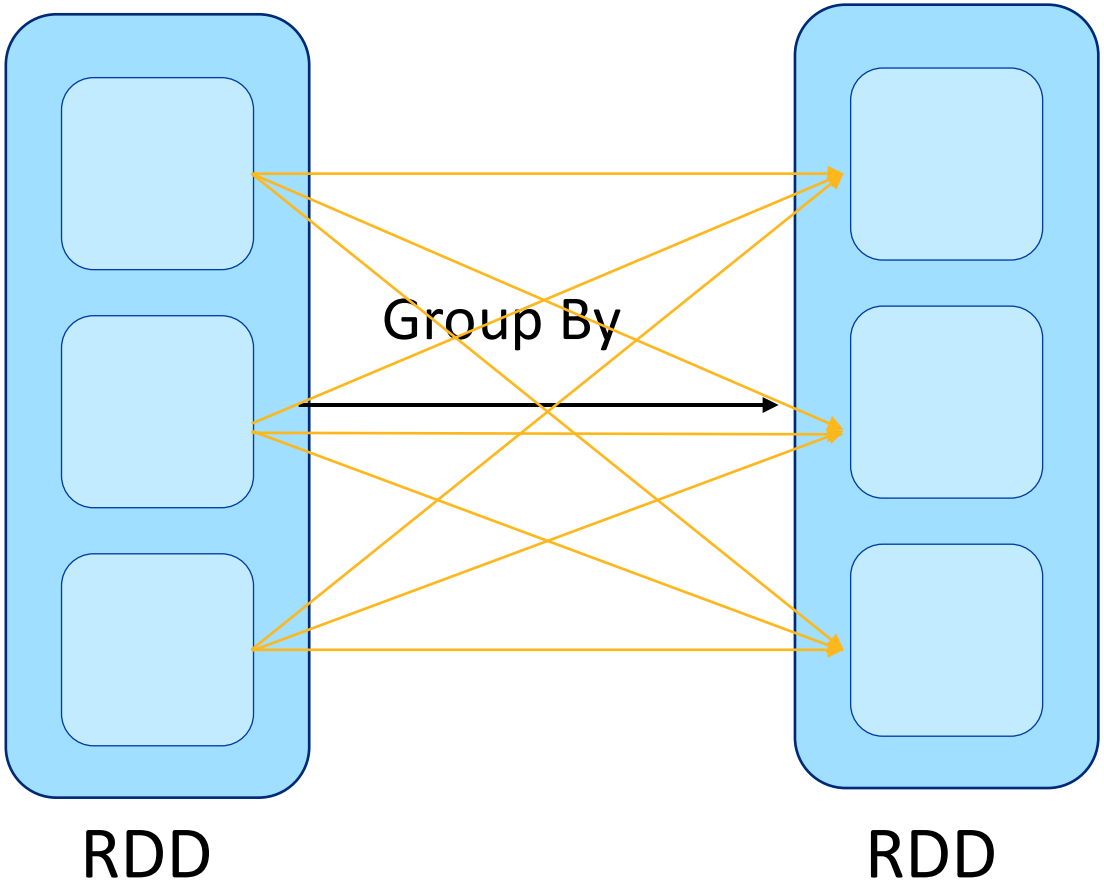


Filter Operation



Similar to the map operation

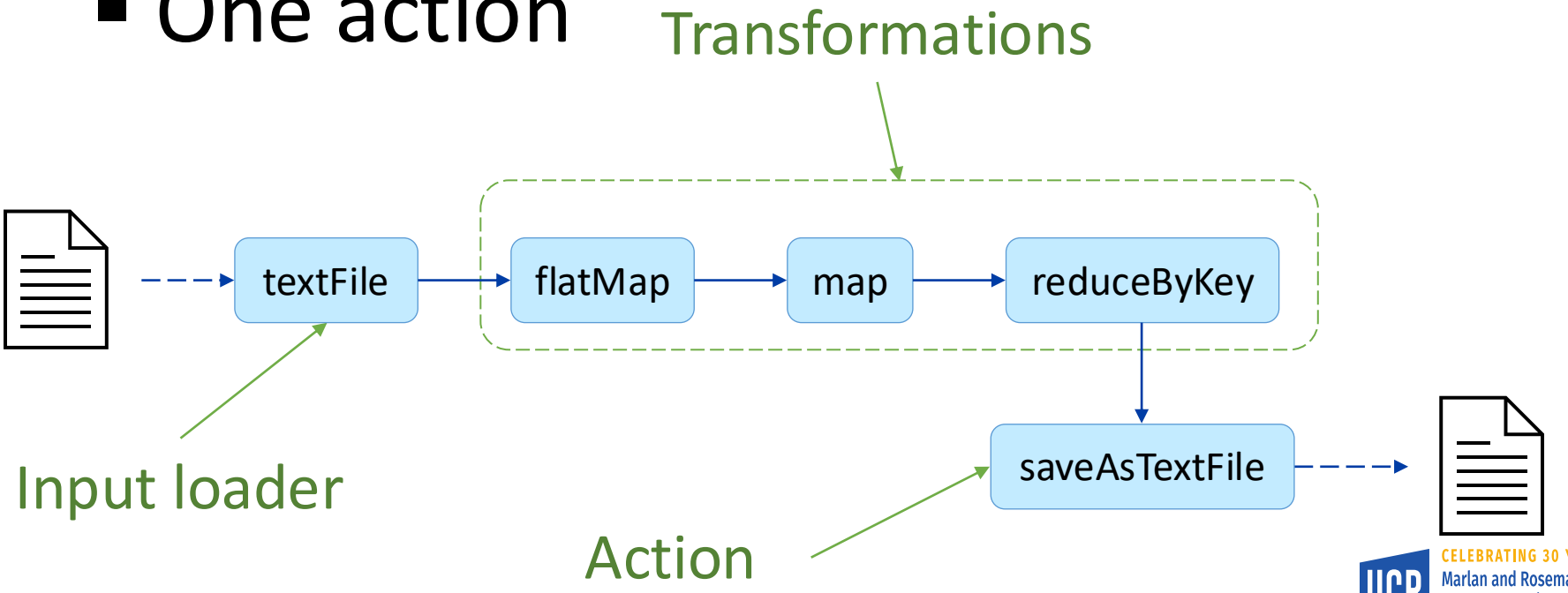
GroupBy (Shuffle) Operation



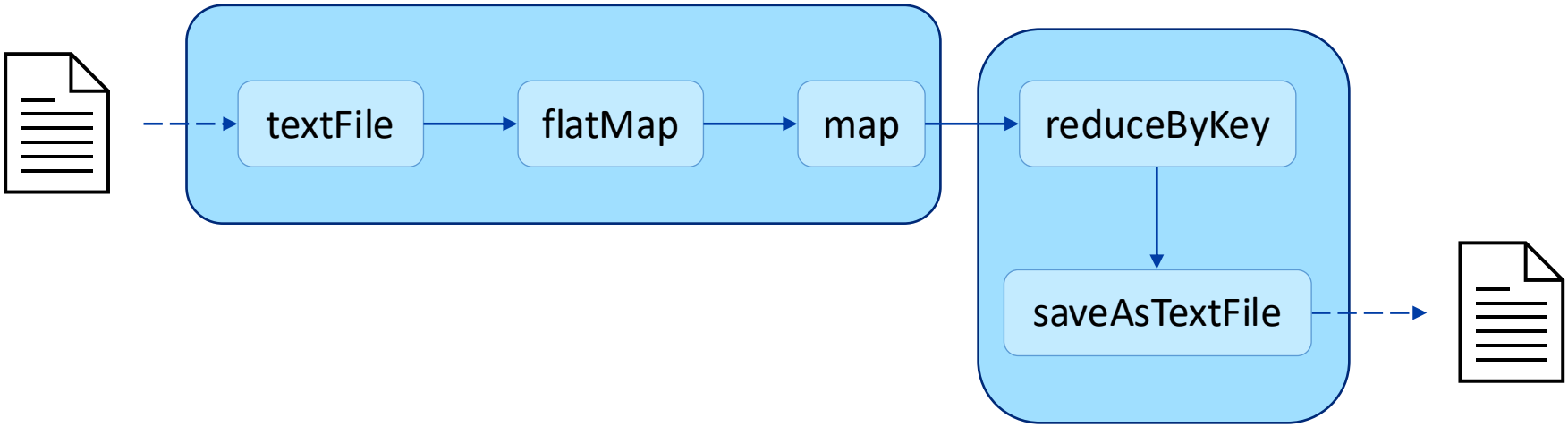
Similar operation **Join**

Application DAG

- A complete DAG consists:
  - One or more input loaders
  - Zero or more transformations
  - One action



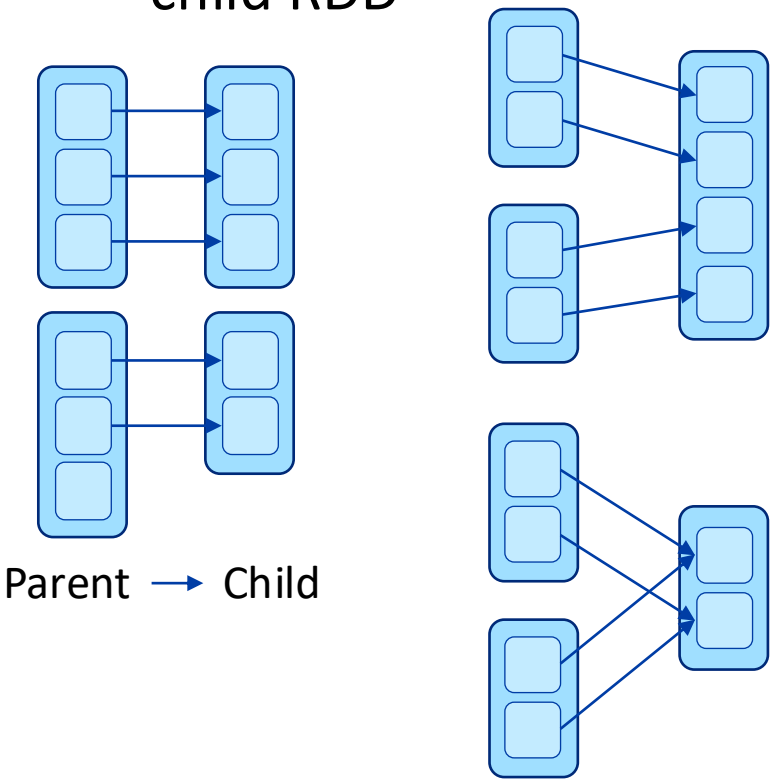
DAG Execution using BSP



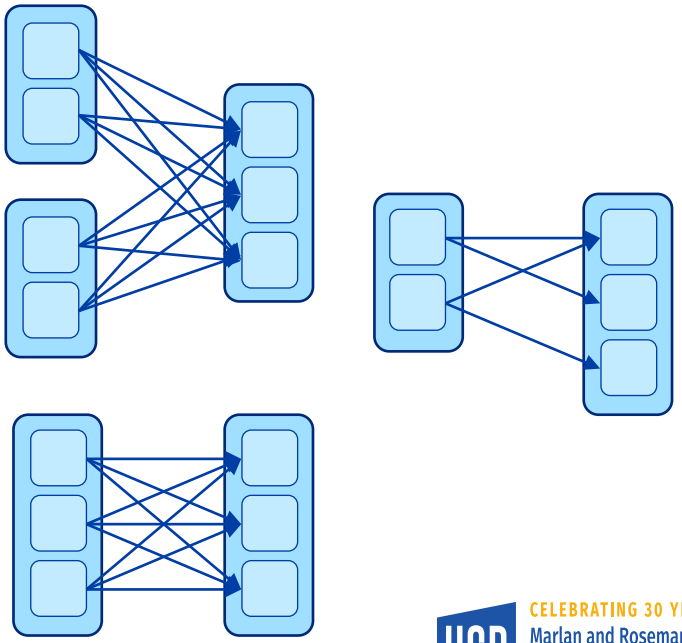
How does Spark split a DAG into stages?

RDD Dependencies

- Narrow Dependency
  - Each partition of the parent RDD is used by at most one partition of the child RDD



- Wide Dependency
  - Each partition of the parent RDD may be depended on by multiple child partitions



Spark RDD Features

- Lazy execution: Collect transformations and execute on actions
- Lineage tracking: Keep track of the lineage of each RDD for fault-tolerance
- Resiliency: When an in-memory partition gets lost, Spark recomputes it

Examples of Transformations

- map
- mapToPair
- flatMap
- reduceByKey
- filter
- sample
- join
- union
- partitionBy

Examples of Actions

- count
- collect
- save(path)
- persist
- reduce

RDD Operations

- Spark is richer than Hadoop in terms of operations
- Sometimes, you can do the same logic with more than one way
- In the following part, we will explain how some RDD operations work
- The goal is to understand the performance implications of these operations and choose the most efficient one

Java Examples

- Apache Spark homepage
  - https://spark.apache.org

```
# Initialize the Spark context
JavaSparkContext spark =
    new JavaSparkContext("local", "BigData-Demo");
```

Examples

```
# Initialize the Spark context
JavaSparkContext spark =
    new JavaSparkContext("local", "BigData-Demo");


# Hello World! Example. Count the number of lines in the file
JavaRDD<String> textFileRDD =
                    spark.textFile("nasa_19950801.tsv");
long count = textFileRDD.count();
System.out.println("Number of lines is "+count);
```

Examples

```
# Count the number of OK lines (response code 200)
JavaRDD<String> okLines = textFileRDD.filter(new
Function<String, Boolean>() {
    @Override
    public Boolean call(String s) throws Exception {
        String code = s.split("\t")[5];
        return code.equals("200");
    }
});
long count = okLines.count();
System.out.println("Number of OK lines is "+count);
```

Examples

```
# Count the number of OK lines (response code 200)
# Shorten the implementation using lambdas (Java 8 and above)
JavaRDD<String> okLines =
 textFileRDD.filter(s -> s.split("\t")[5].equals("200"));

long count = okLines.count();
System.out.println("Number of OK lines is "+count);
```

Examples

```
# Make it parametrized by taking the response code as a
command line argument
String inputFileName = args[0];
String desiredResponseCode = args[1];
...
JavaRDD<String> textFileRDD = spark.textFile(inputFileName);
JavaRDD<String> okLines = textFileRDD.filter(new
Function<String, Boolean>() {
    @Override
    public Boolean call(String s) {
        String code = s.split("\t")[5];
        return code.equals(desiredResponseCode);
    }
});
```

Examples

```
# Count by response code

# Important! Not all transformations and actions are on the
getting started guide

JavaPairRDD<Integer, String> linesByCode =
textFileRDD.mapToPair(new PairFunction<String, Integer,
String>() {
    @Override
    public Tuple2<Integer, String> call(String s) {
        String code = s.split("\t")[5];
        return new Tuple2<Integer,
String>(Integer.valueOf(code), s);
    }
});
Map<Integer, Long> countByCode = linesByCode.countByKey();
System.out.println(countByCode);
```