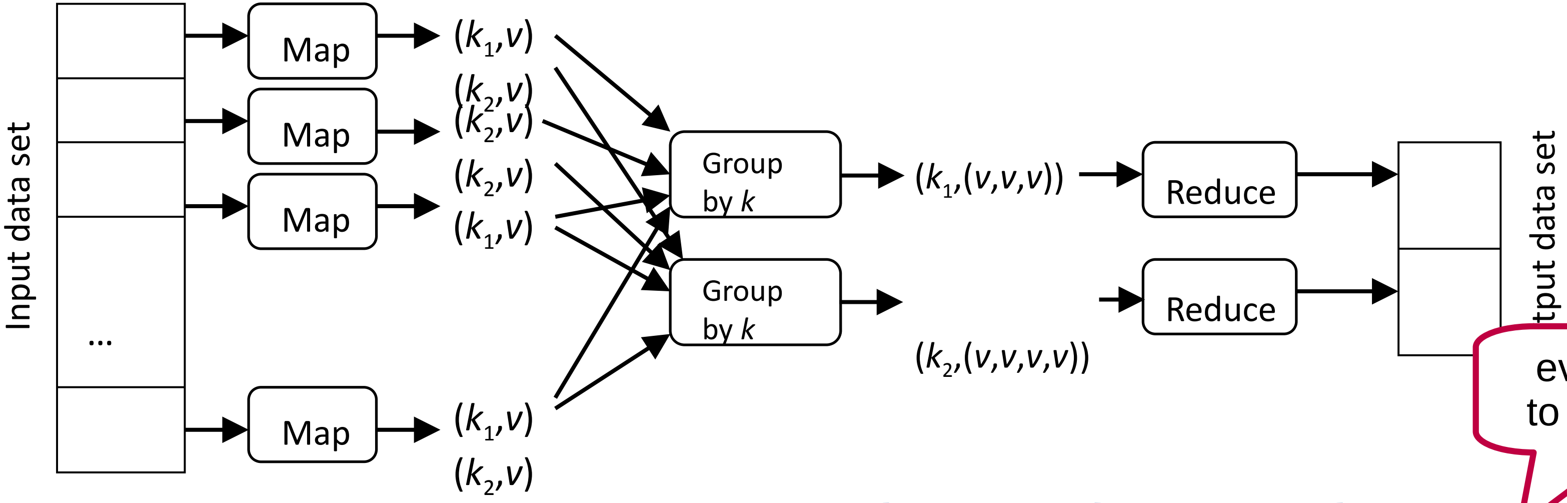
Data flow

MR: Efficiency limitations



read from file

map to pair

persist

sort and group by

persist (again...)

reduce

write to file

even if it results from another job

even if it fits in memory

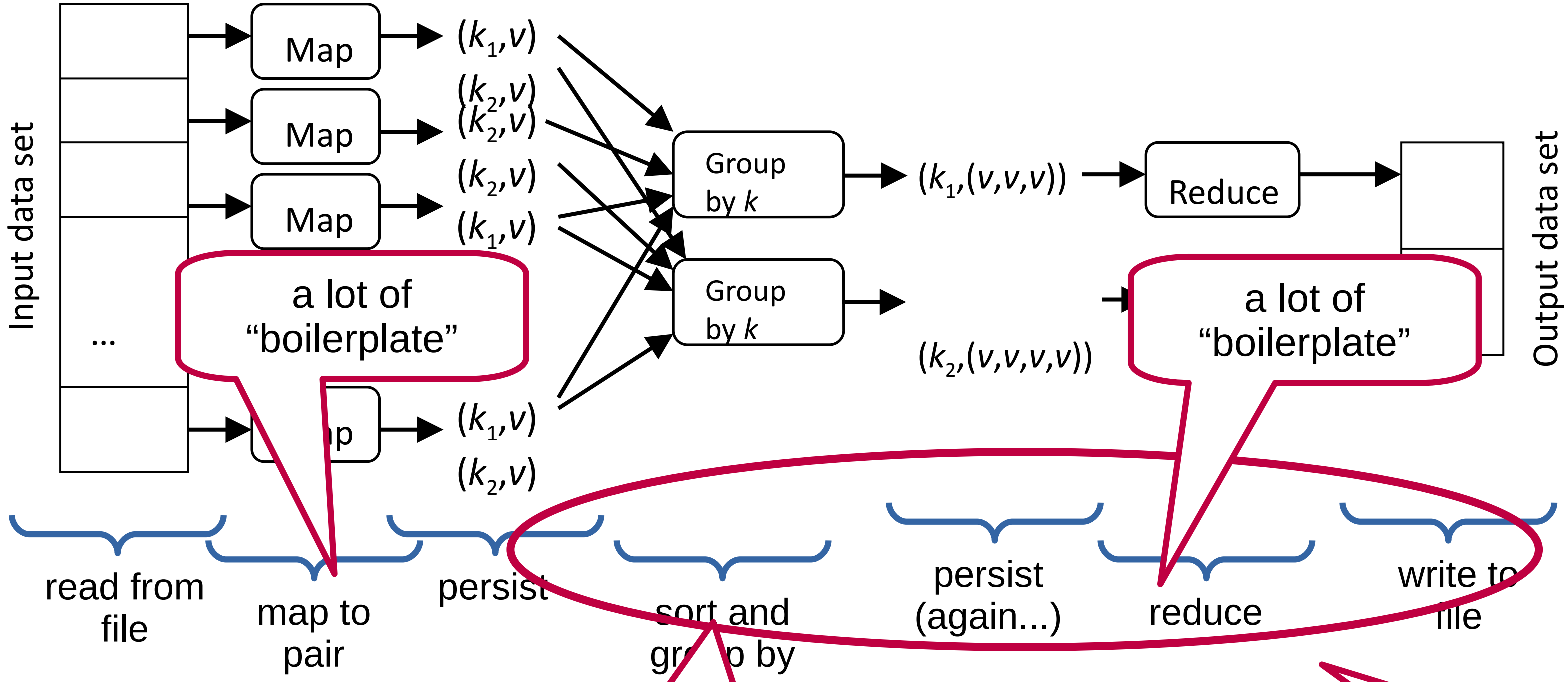
even if it fits in memory

even if going to another job

Need to avoid writing to disk so much!

Image adapted from: M. Tamer Özsu, Patrick Valduriez. Principles of Distributed Database Systems (3rd Edition)

MR: Usability limitations



a lot of "boilerplate"

a lot of "boilerplate"

re-write from scratch

sort without group?

re-write from scratch

optional but... all or nothing

Need to freely compose stages!

Image adapted from: M. Tamer Özsu, Patrick Valduriez. Principles of Distributed Database Systems (3rd Edition)

Data flow

- Data flows through a sequence (or DAG) of transformations
- Examples:
  - The Unix shell:

    ```
    $ du -sk * | sort -rn | head > top10.txt
    ```

  - Map-Reduce
  - Java Streams:

    -
      ```
      list.stream().filter(x→x>10)
          .sort().forEach(System.out::println);
      ```

APACHE
Spark
TM

Spark Core

- Based on the abstraction of a collection of objects:
  - **RDD: Resilient Distributed Dataset**

will regenerate
if parts are lost

computed and
stored by many
nodes

- Described by functional composition of transformations
- Lazily executed when observed with actions

Example

```
// Initialize and connect to a local Spark cluster
SparkConf conf = new SparkConf().setMaster("local").setAppName("simple");
JavaSparkContext sc = new JavaSparkContext(conf);

// A first RDD: the recipe for scanning a List<Integer> in parallel
JavaRDD<Integer> rdd = sc.parallelize(l);

// A second RDD: the recipe for filtering the result of
// mapping the result of... the previous RDD
rdd = rdd.map(x->x+1).filter(x->x>5);

// The collect action on the RDD returns a new List<Integer>
l = rdd.c
```

no computation has occurred só far…

this action triggers the execution of all transformations described by the RDD

RDD classes

- JavaRDD: collection of objects
  - Generic transformations:
    - filter(), map(), ….
- JavaPairRDD: collection of pairs
  - Created with .mapToPair(…) transformation
  - Allows transformations based on Keys:
    - groupByKey(), sortByKey(), reduceByKey(), ...
- JavaDoubleRDD: collection of real numbers
  - Created with .mapToDouble() transformation
  - Allows computation of statistics:
    - mean(), histogram(), stddev(), ….

Transformations and actions

- Methods in *RDD classes that return some *RDD instance are transformations
- Other methods, that return or output the data, are actions
- Check Javadoc for RDD classes:

  https://spark.apache.org/docs/latest/api/java/index.html?org/apache/spark/api/java/JavaRDD.html

MapReduce translation

MapReduce translation

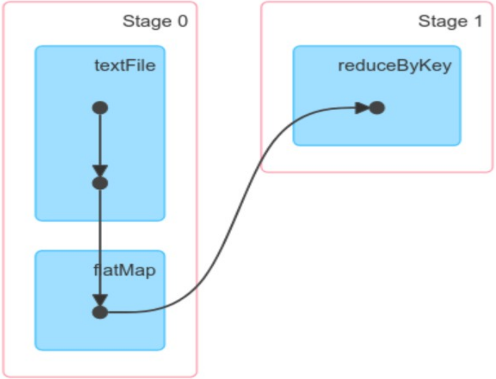
job configuration

```
SparkConf conf = new SparkConf().setMaster("local").setAppName("g0spark")
JavaSparkContext sc = new JavaSparkContext(conf);

JavaPairRDD<String, Integer> mr = sc.textFile("file:///pathto/title.basics.tsv.bz2")
        .flatMapToPair(l -> {
            String[] f = l.split("\t");
            if (!f[0].equals("tconst") && !f[8].equals("\\N"))
                return Arrays.stream(f[8].split(","))
                          .map(g->new Tuple2<>(g,1)).iterator();
            else
                return Collections.<Tuple2<String,Integer>>emptyList().iterator();
        })
        .reduceByKey((i, j) -> i + j);

List<Tuple2<String, Integer>> g                ect();
```

running the job

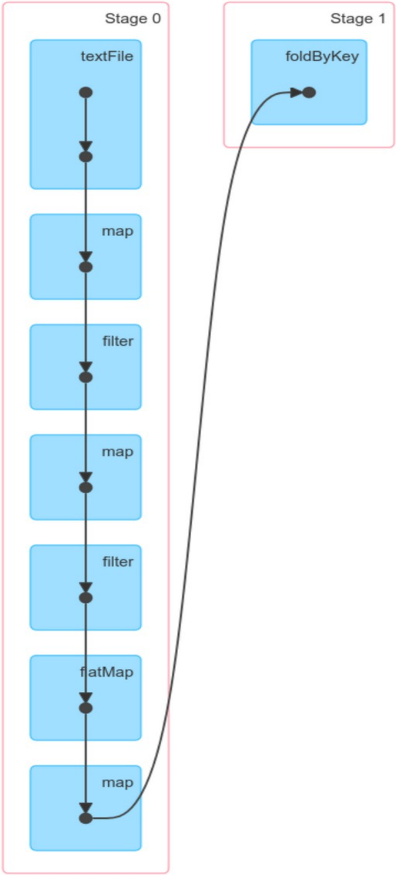


MapReduce translation

```
SparkConf conf = new SparkConf().setMaster("local").setAppName("g0spark");
JavaSparkContext sc = new JavaSparkContext(conf);

JavaPairRDD<String, Integer> mr = sc.textFile("file:///pathto/title.basics.tsv.bz2")
        .map(l -> l.split("\t"))
        .filter(l -> !l[0].equals("tconst"))
        .map(l -> l[8])
        .filter(l -> !l.equals("\\N"))
        .flatMap(l -> Arrays.asList(l.split(",")).iterator())
        .mapToPair(l -> new Tuple2<>(l, 1))
        .foldByKey(0, (v1, v2) -> v1 + v2);

List<Tuple2<String, Integer>> genres = mr.collect();
```



many transformations
in a single stage

Configuration

- A local cluster needs only one Maven dependency:

```
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-core_2.12</artifactId>
    <version>3.1.1</version>
</dependency>
```