Working with the Data

- In Spark, the core data structures are immutable
  - This means, they cannot be changed once created
- Might look a bit strange at first sight
  - Similar to functional programming languages. . .
- How do we use data if we cannot change it?

Transformations

- Transformations describe how we would like to modify a data structure
  - Applying a transformation returns a new data structure
  - It does not actually modify the existing one
- Assume `myRange` contains a collection of numbers
- We only want to keep the even numbers, so we perform

```
divBy2 = myRange.where("id % 2 = 0")
```

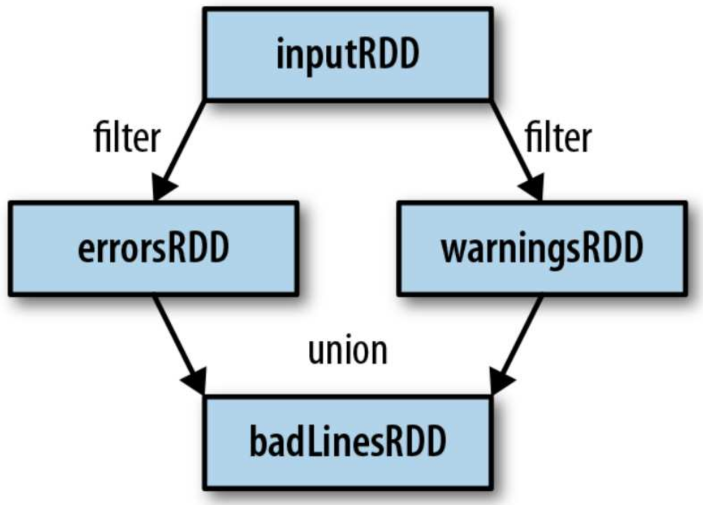
- This will produce no output until we call an action
  - We discuss actions in just a moment

Transformations (2)

- We can apply a whole sequence of transformations
- For example, we can apply another filter:

```
divBy6 = divBy2.where("id % 3 = 0")
```

- This is called a lineage graph in Spark

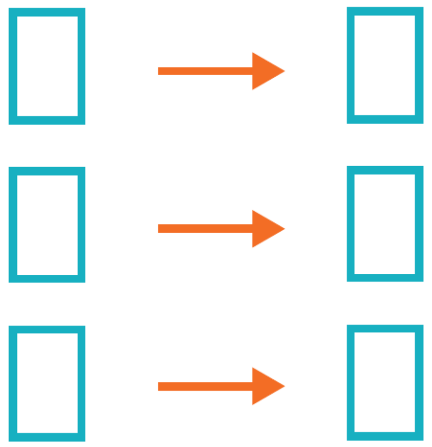


What’s the Advantage?

- Spark employs lazy evaluation:
  - Waits until the last moment to execute a computation
- There are some reasons why this makes sense:
  - This way Spark can build an efficient execution plan
    - It only computes what is strictly needed
    - In our example: only perform action on IDs divisible by 6
  - Also simplifies the restart after losing a data partition
    - The original data is not changed
    - Every partition contains information to recalculate partition

Narrow Transformations

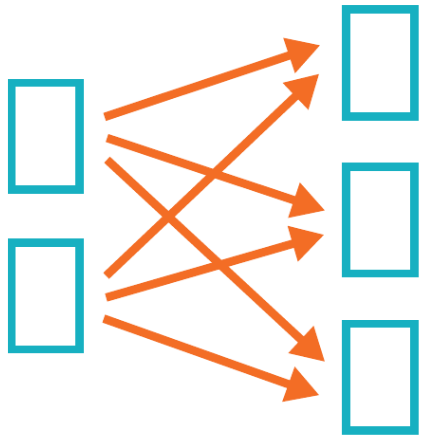
- Applying a narrow transformation means that each input partition contributes to only one output partition



- This can be processed very efficiently
  - Multiple filters can all be performed in-memory (via pipelining)
- The `where` filter we used is a narrow transformation

Wide Transformations

- Applying a wide transformation means that each input partition contributes to many output partitions



- In this case, partitions are exchanged across the cluster
  - This involves writing data to disk, so not as efficient
- Often referred to as a shuffle

Actions

- Actions are operations returning something other than an RDD, DataFrame, or Dataset
- Actions trigger the actual computation (and evaluation of transformations)
- One of the simplest actions is `count`
  - This determines the number of records in a data structure
- For example, we could apply it to `divBy6`:

```
divBy6.count()
```

- Assuming a range from 0 to 999, this would output 167

Actions (2)

- There are three different kinds of actions
  - To bring data back to the driver
    - For example, for viewing data in the console
  - To collect data to native objects in the respective programming language
  - To write to output data sources