EPL451: Data Mining on the Web – Lab 3

University of Cyprus
Department of
Computer Science

Παύλος Αντωνίου

Γραφείο: Β109, ΘΕΕ01

Classes in Hadoop: InputFormat

- Fundamental class in Hadoop Map-Reduce
- Defines the list of tasks that make up the mapping phase
- Responsible for:
  1. Data splits:
     a) defines the size of data to be processed from individual Map tasks and splits data into chunks (see ***InputSplit*** class)
     b) determines execution node
        – tasks are assigned to nodes based on where the input file chunks are physically resident
  2. Record reader:
     a) Each split is divided into records
     b) ***RecordReader*** is responsible for actual reading records from the InputSplit object and submitting them (as key/value pairs) to the mapper

Java Classes in Hadoop

- By default the **FileInputFormat** and its descendants break a file up into 64 MB chunks (the same size as blocks in HDFS)
  - control this value by setting the mapred.min.split.size parameter in hadoop-site.xml, or by overriding the parameter in the JobConf object used to submit a particular MapReduce job.

```
FileInputFormat.addInputPath(job, new Path("hdfs://localhost:54310/user/csdeptucy/input"));
```

- Based on chunk size, one ***InputSplit*** object for each map task is created
  - FileSplit is the default InputSplit

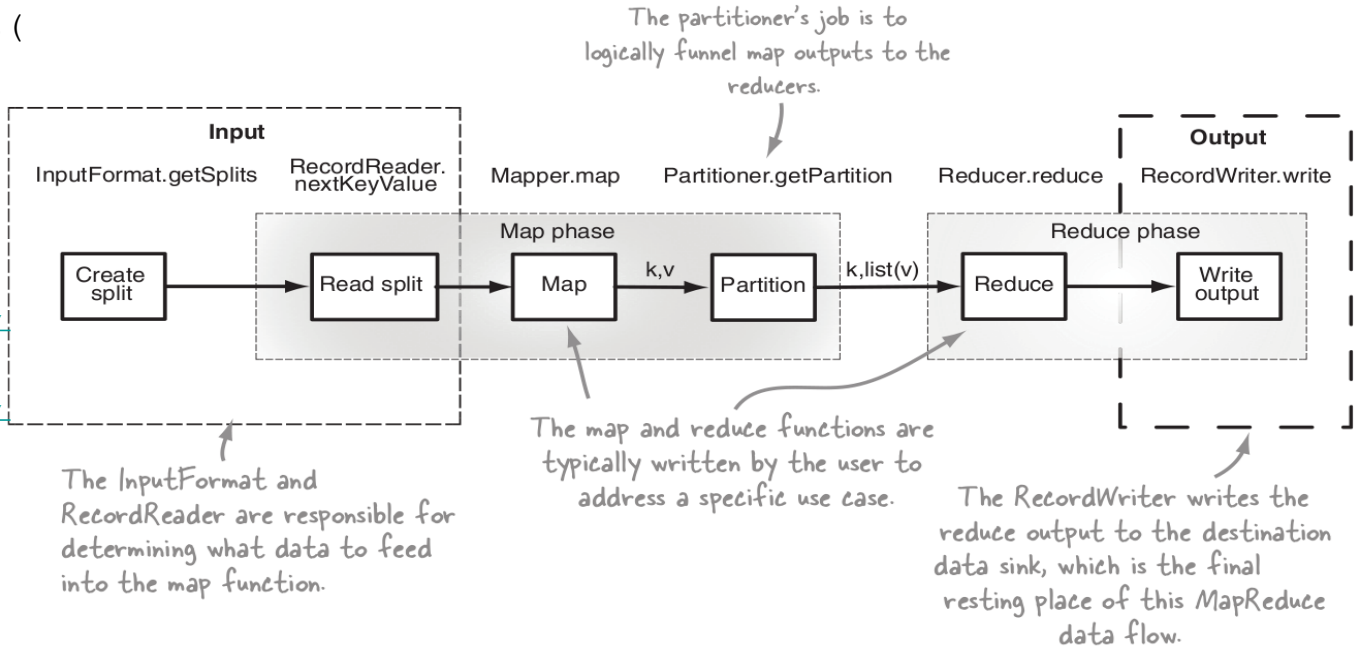
- ***InputSplit*** describes a unit of work, that comprises a single map task in MapReduce program.

Java Classes in Hadoop

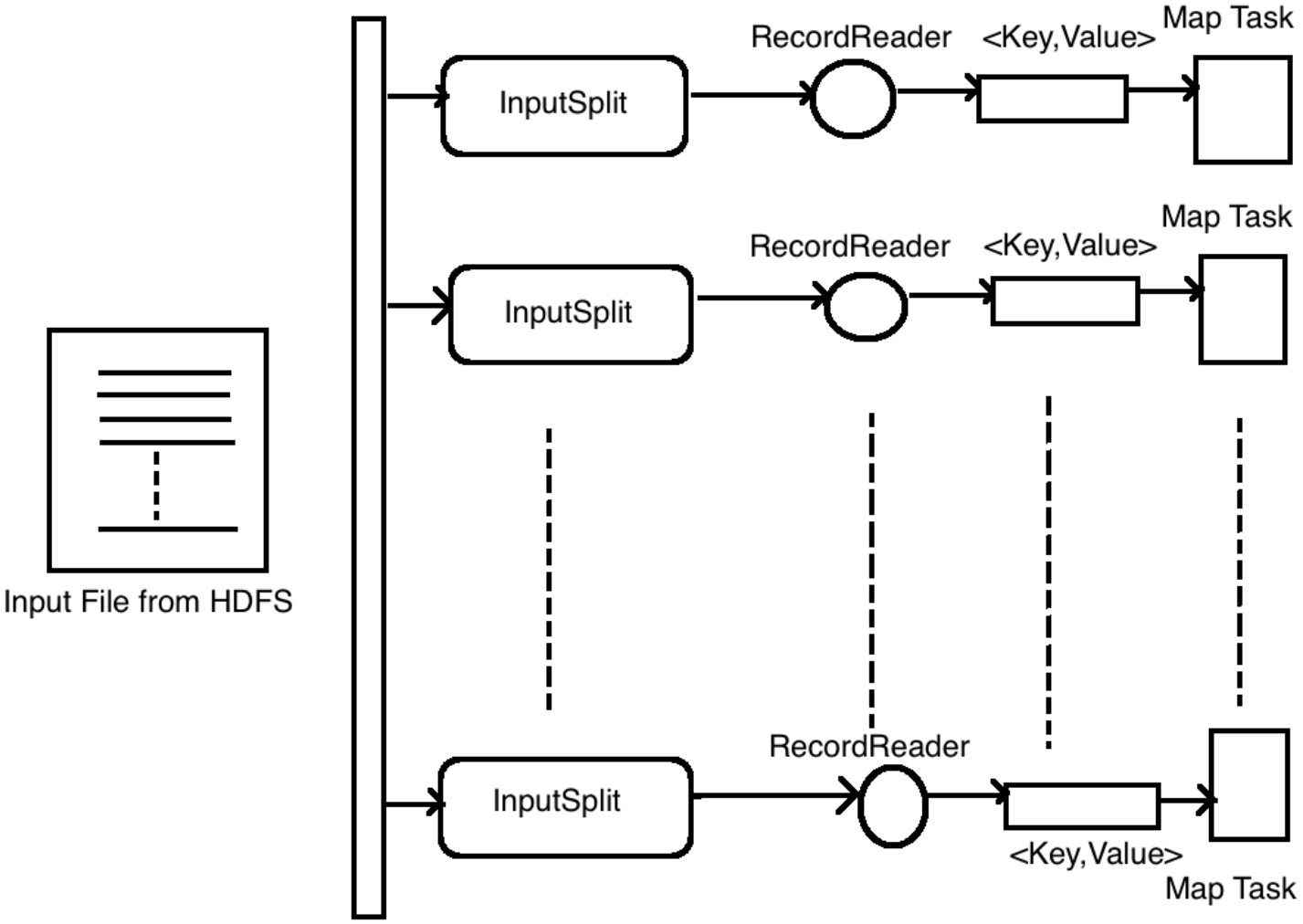
- ***RecordReader*** reads <k,v> pairs from InputSplit
  - instance is defined by the InputFormat
  - default InputFormat, ***TextInputFormat***, provides a *LineRecordReader*, which treats each line of the input file as a new value.

```
job.setInputFormatClass(
TextInputFormat.class);
```

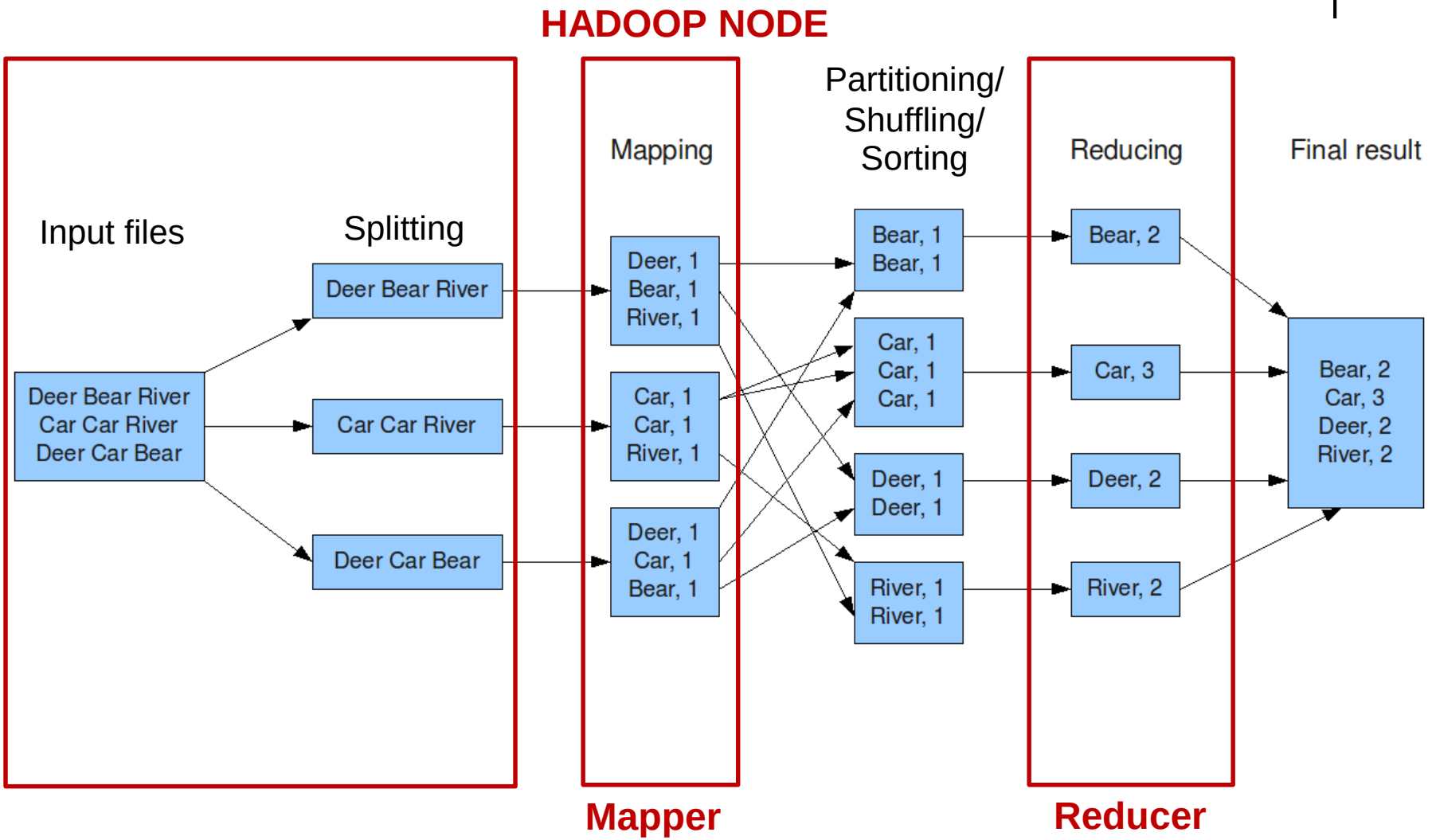
CombineFileInputFormat,
FixedLengthInputFormat,
KeyValueTextInputFormat,
NLineInputFormat,
SequenceFileInputFormat,
TextInputFormat



File splitting into map tasks



Διαδικασία Map Reduce word count



Map Reduce process

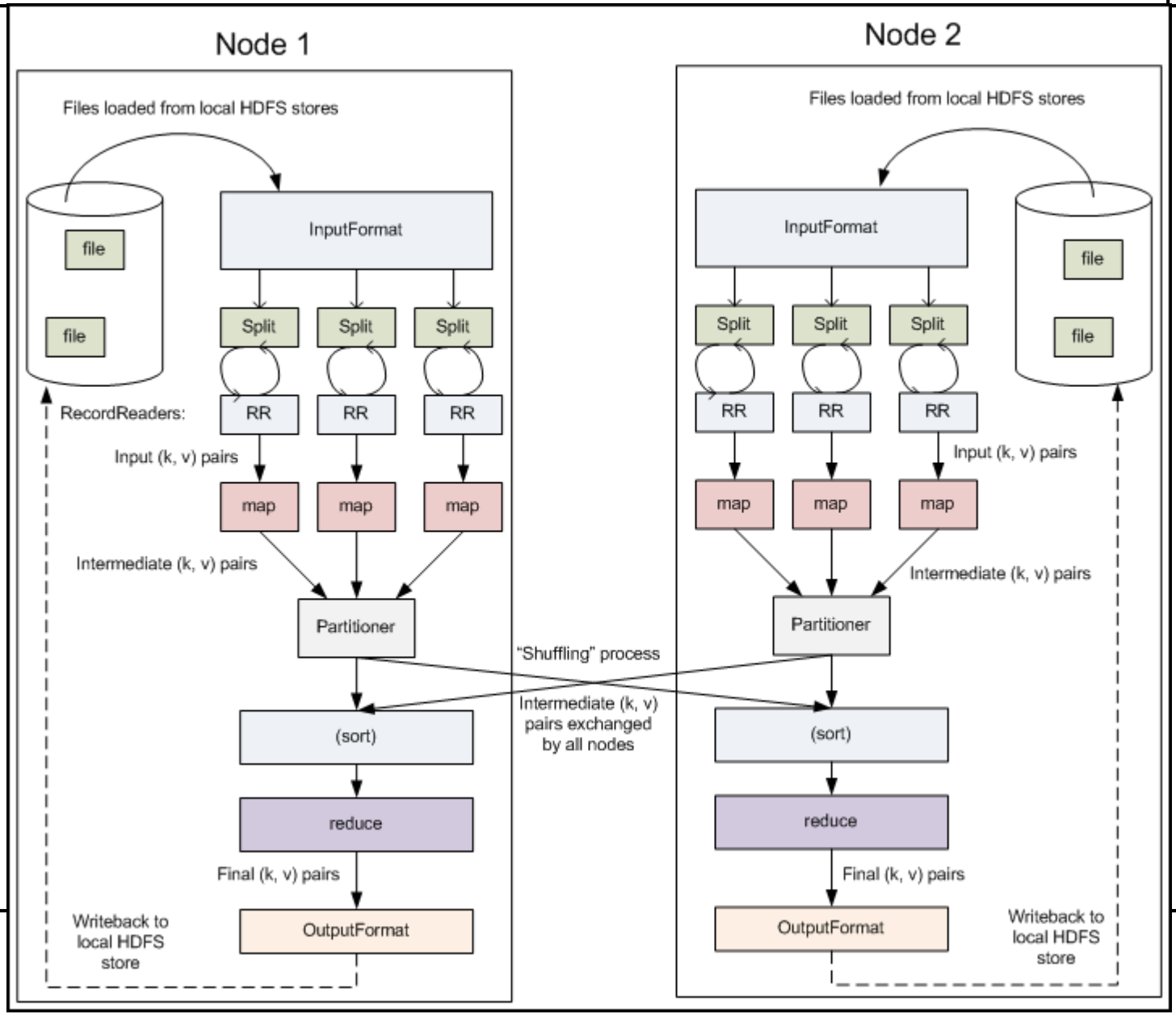
- **Mapping**: Όταν η διαδικασία του mapping ολοκληρωθεί, τα ζεύγη (κλειδί, ενδιάμεση τιμή) πρέπει να ανταλλαχθούν μεταξύ των κόμβων (αν υπάρχουν πολλοί) για να σταλούν όλες οι τιμές (values) με το ίδιο κλειδί σε ένα μοναδικό reducer

- **Partitioning**: Ο κάθε partitioner καθορίζει ποιος reducer θα λάβει τα ενδιάμεσα ζεύγη key-value που προκύπτουν από κάθε mapper
  - ζεύγη με το ίδιο key καταλήγουν στον ίδιο reducer

Map Reduce process

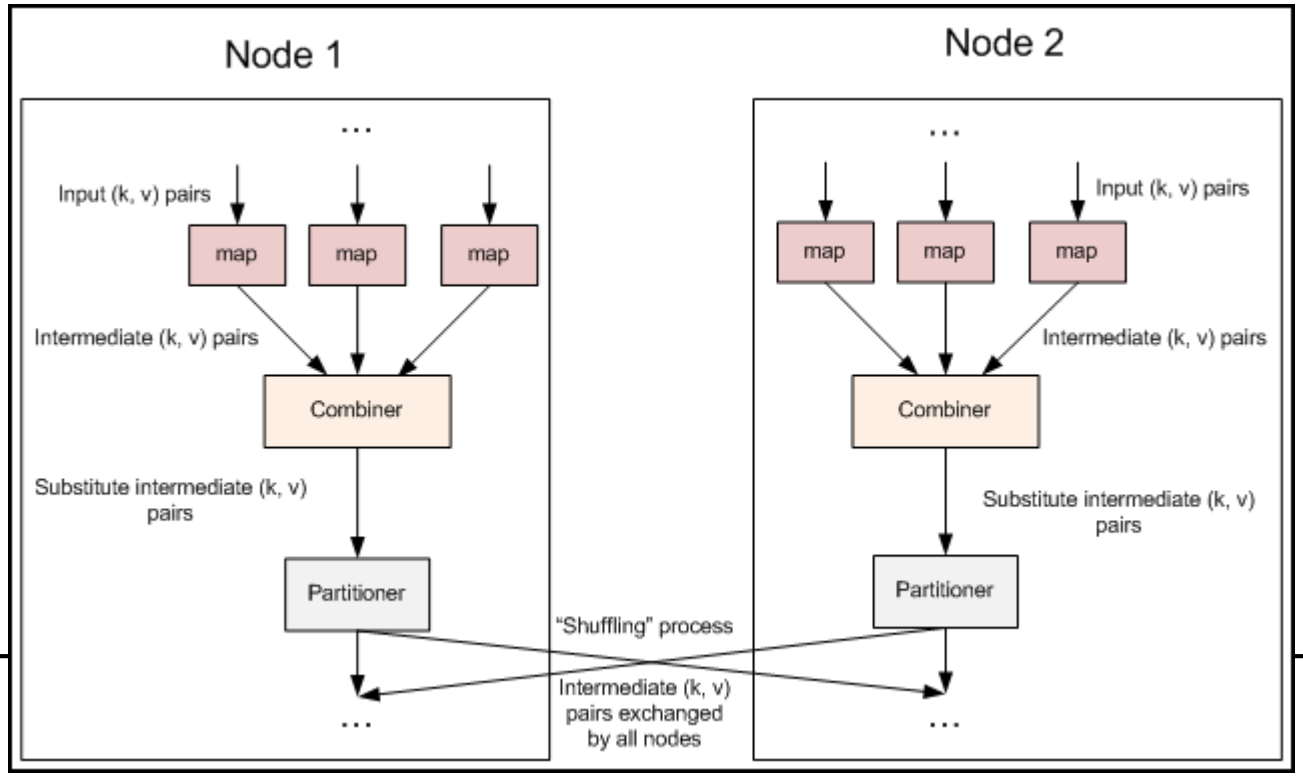
- **Shuffling:** διαδικασία μεταφοράς των ζευγών <κλειδί, ενδιάμεση τιμή> από κάθε mapper στον reducer → το μοναδικό σημείο επικοινωνίας μεταξύ κόμβων (γίνεται πάνω από HTTP)
- **Sorting:** διαδικασία συνένωσης και ταξινόμησης ως προς το κλειδί των ζευγών <κλειδί, ενδιάμεση τιμή>. Στο τέλος του sorting, έχουμε μια ταξινομημένη λίστα με <key, (collection of values)>
- **Reduce** είναι η φάση που σε κάθε record της πιο πάνω ταξινομημένης λίστας γίνεται η συνάθροιση των values για το ίδιο κλειδί.
- Ένα πολύ ωραίο άρθρο για το Map Reduce: https://developer.yahoo.com/hadoop/tutorial/module4.html

Map Reduce process



Map Reduce process

- **Combiner (optional)**: Μπορεί να τρέξει μετά τον Mapper (για να κάνει ένα mini-reduce) πάνω στα τοπικά δεδομένα (ενός κόμβου)
  - Μειώνει τα ενδιάμεσα δεδομένα που ανταλλάζονται (κατά το shuffling) πάνω από το δίκτυο



Υλοποίηση Map

- Μέσα στην κλάση που θα δημιουργήσει ο χρήστης, πρέπει να οριστεί μια static class που κάνει extend
  class Mapper <KEYIN,VALUEIN,KEYOUT,VALUEOUT>
- Η πιο εξέχουσα μέθοδος της κλάσης Mapper είναι η μέθοδος map που ορίζεται:
  void map (KEYIN key, VALUEIN value, Context context)
- Οι τύποι ***KEYIN, VALUEIN, KEYOUT, VALUEOUT*** πρέπει να «υπακούουν» σε μια συγκεκριμένη διεπαφή (interface), που ονομάζεται *Writable*
- Ο τύπος Context χρησιμεύει για την αποθήκευση των δεδομένων που θα επιστρέψει η μέθοδος

Υλοποίηση Reduce

- Μέσα στην κλάση που θα δημιουργήσει ο χρήστης, πρέπει να οριστεί μια static class που κάνει extend
  class Reducer <KEYIN,VALUEIN,KEYOUT,VALUEOUT>

- Η πιο εξέχουσα μέθοδος της κλάσης Reducer είναι η reduce. Η σύνταξη της είναι:
  void reduce(KEYIN key, Iterable<VALUEIN> value, Context context)
  - Η είσοδος της μεθόδου reduce είναι ένα ζεύγος της μορφής <key, (collection of values)>

Writable interface

- Όλα τα δεδομένα στο Hadoop (από/προς αρχεία, από/προς Mappers/Reducers) αποθηκεύονται σε αντικείμενα
- Αντικείμενα τα οποία περιέχουν πληροφορίες που διαβάζονται από αρχεία ή θα γραφούν σε αρχεία ή θα σταλούν πάνω από δίκτυο (π.χ. κατά το shuffling) πρέπει να υπακούουν σε μια συγκεκριμένη διεπαφή (interface), που ονομάζεται Writable, η οποία επιτρέπει στο Hadoop να διαβάσει και να γράψει τα δεδομένα σε διατεταγμένη (serialized) μορφή (ροή από bytes) αναγκαία για μετάδοση (transmission).

Writable interface

- Το Hadoop παρέχει κάποιες κλάσεις που υπακούουν στη διεπαφή *Writable*:
  - Text (που αποθηκεύει δεδομένα τύπου String), IntWritable (ακέραιοι), LongWritable, FloatWritable, BooleanWritable, καθώς και άλλες που ορίζονται στο org.apache.hadoop.io package
- Το Hadoop επιτρέπει στον προγραμματιστή να δημιουργήσει δικούς του τύπους κλειδιών/τιμών. Αυτό επιτρέπει τη χρήση άλλων τύπων πέραν των κλασσικών Text, IntWritable κτλ.
- Αυτοί οι τύποι πρέπει να υλοποιούν (implement) την διεπαφή ***Writable***

Writable interface: Παράδειγμα

- Δίνονται log files (text files) από κάποιο server
  - έστω ότι κάθε γραμμή έχει πολλές πληροφορίες σχετικά με ένα γεγονός που συμβαίνει στο server
- Θέλουμε να κρατήσουμε μόνο κάποιες πληροφορίες σχετικά με αιτήσεις (requests) που γίνονται στον server
- Για κάθε αίτηση θα φυλάγουμε τις πληροφορίες σε στιγμιότυπο (αντικείμενο) της κλάσης RequestInfo
- Χαρακτηριστικά κλάσης:
  - requestId,
  - requestType,
  - timestamp.
- Οι 3 αυτές τιμές μπορούν να εξαχθούν (σαν ένα αντικείμενο) από Mapper /Reducer σαν κλειδί (key) ή σαν τιμή (value).
- Δείτε αρχείο εδώ.

Writable interface

- Αν οι πληροφορίες θα εξαχθούν σαν:
  - τιμή (value) → κλάση ΝΑ υλοποιεί την διεπαφή Writable

```
public interface Writable
{
    void readFields(DataInput in);
    void write(DataOutput out);
}
```

  - κλειδί (key) → κλάση ΝΑ υλοποιεί την διεπαφή WritableComparable

```
public interface WritableComparable extends Writable,
Comparable
{
    void readFields(DataInput in);
    void write(DataOutput out);
    int compareTo(WritableComparable o);
}
```

Writable interface

- Η κλάση που θα υλοποιεί τη διεπαφή Writable θα πρέπει να ορίσει τουλάχιστον τις 2 μεθόδους:
  - `write(DataOutput out)` – Χρησιμοποιείται για να «διευθετήσει» σε σειρά (serialize) τα πεδία του αντικειμένου μέσα στο ‘out’ (που είναι μια ροή από bytes, byte stream)
  - `readFields(DataInput in)` – Χρησιμοποιείται για να κάνει deserialize τα πεδία του αντικειμένου από το byte stream ‘in’.
- Η κλάση που θα υλοποιεί τη διεπαφή WritableComparable θα πρέπει να ορίσει τουλάχιστον τις 2 μεθόδους + τη compareTo()
  - Η μέθοδος αυτή επιτρέπει στο Hadoop να ταξινομεί τα κλειδιά στη φάση του sort.

Task1: Inverted index

- Σας δίνεται ο κώδικας που κτίζει το Inverted Index (αντίστροφο ευρετήριο) για κάποια αρχεία.
- Ο κώδικας δημιουργεί ένα αρχείο που περιέχει τη κάθε λέξη και σε ποια αρχεία/γραμμές βρίσκεται:
  - word filename1@lineoffset, filename2@lineoffset, …
- Μελετήστε τον κώδικα για να καταλάβετε πώς λειτουργεί.
- Αλλάξτε των κώδικα ούτως ώστε να δημιουργήσετε ένα δικό σας τύπο δεδομένων για να εξάγετε από τον **mapper** τα records [filename@offset] και να εισάγετε στον **reducer** με τη μορφή **IndexMapRecordWritable** (σας δίδεται μισό-συμπληρωμένη η κλάση)

Task1: Inverted index

- Αλλάξτε των κώδικα ούτως ώστε να δημιουργήσετε ένα δικό σας τύπο δεδομένων για να εξάγετε από τον **reducer** το **τελικό** αποτέλεσμα με τη μορφή **IndexRecordWritable** (σας δίδεται μισό-συμπληρωμένη η κλάση)
- *ΣΗΜΕΙΩΣΗ:* Πρέπει να κάνετε override τις ακόλουθες μεθόδους:

```
/**
 * Concat all the index map records
 */
@Override
public String toString() {}
```

```
/**
 * Serialize the fields
 */
@Override
public void
write(DataOutput out)
throws IOException {}
```

```
/**
 * Deserialize the fields
 */
@Override
public void
readFields(DataInput in)
throws IOException {}
```