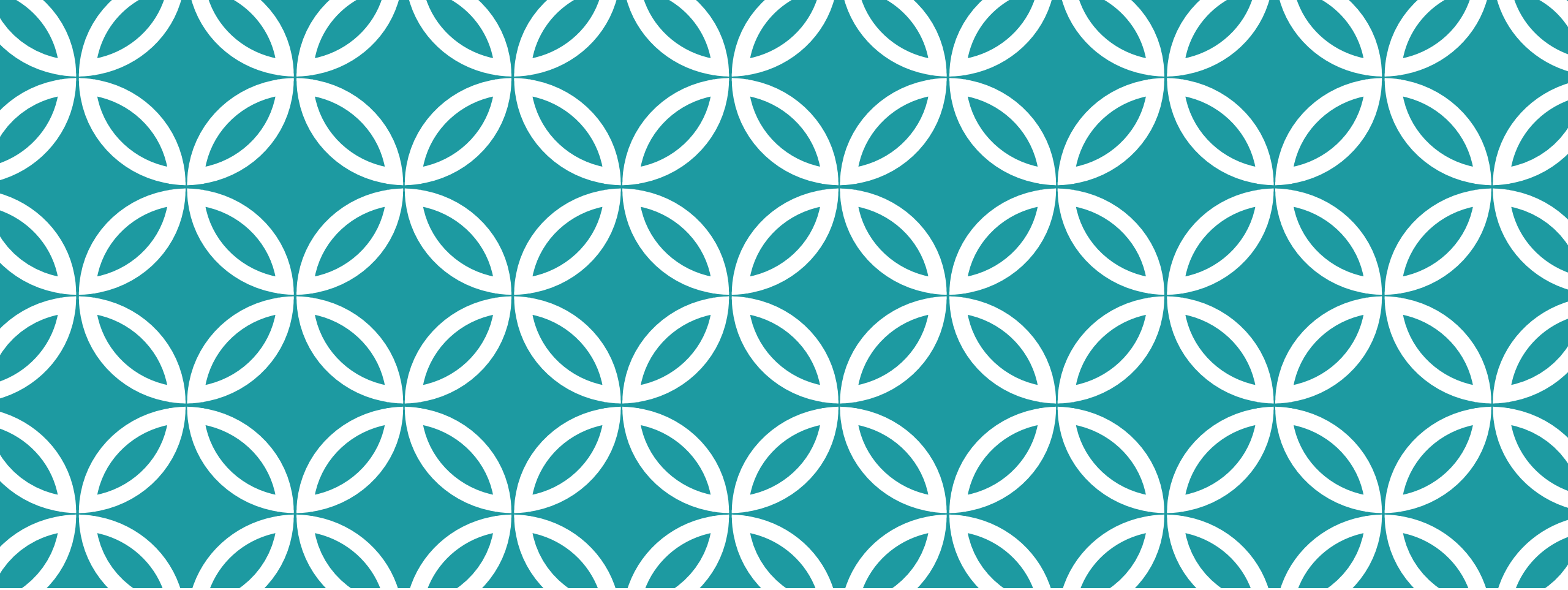


Hadoop Job en pratique II

Fonctionnement
Outils
Utilisation
Programmation

Hadoop Writable : Value

Il est possible de créer ses propres classes pour stocker les valeurs des messages. Les classes représentant des valeurs doivent impérativement implémenter l'interface Writable. Toute classe implémentant Writable peut ainsi être utilisée dans les entrée et sortie des mappeurs, réduceurs et être lu/écrite dans un fichier de sortie.

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

Hadoop Writable : Value, exemple

Les valeurs peuvent donc contenir n'importe quoi, elles doivent juste pouvoir être « sérialisée/désérialisée » ou enregistrée/lue dans DataOuput/DataInput (class Java)

```
public class MyWritable implements Writable {
        private int counter;
        private long timestamp;

        public void write(DataOutput out) throws IOException {
          out.writeInt(counter);
          out.writeLong(timestamp);
        }

        public void readFields(DataInput in) throws IOException {
          counter = in.readInt();
          timestamp = in.readLong();
        }
      }
}
```

Hadoop WritableComparable : Key

Il est possible de créer ses propres classes de clés utilisées pour le « shuffling » des valeurs. Les classes représentant des clés doivent impérativement implémenter l'interface WritableComparable qui étend simulatanement Hadoop.Writable et Java.Comparable. Toutes les classes implémentant WritableComparable peuvent être utilisées comme clé.

```
public interface WritableComparable<T> extends Writable, Comparable<T> {
}

public class MyWritableComparable implements WritableComparable<MyWritableComparable> {
public void write(DataOutput out) throws IOException …
public void readFields(DataInput in) throws IOException ...
public int compareTo(MyWritableComparable o) {
return …
}
public int hashCode() {
  return …
}
}
```

Hadoop WritableComparable : Key

```
public interface WritableComparable<T> extends Writable, Comparable<T> {
}
public class MyWritableComparable implements WritableComparable<MyWritableComparable> {
private int counter;
private long timestamp;
public void write(DataOutput out) throws IOException …
public void readFields(DataInput in) throws IOException ...

public int compareTo(MyWritableComparable o) {
 int thisValue = this.counter;
 int thatValue = o.counter;
 return (thisValue < thatValue ? -1 : (thisValue==thatValue ? 0 : 1));
}
public int hashCode() {
  final int prime = 31;
  int result = 1;
  result = prime * result + counter;
  result = prime * result + (int) (timestamp ^ (timestamp >>> 32));
  return result
}
}
```

Exemple de Writable pour le calcul de la moyenne

Pour calculer une moyenne, il faut sommer toutes les valeurs et diviser par le nombre de valeurs. Cette opération n'est pas associative.

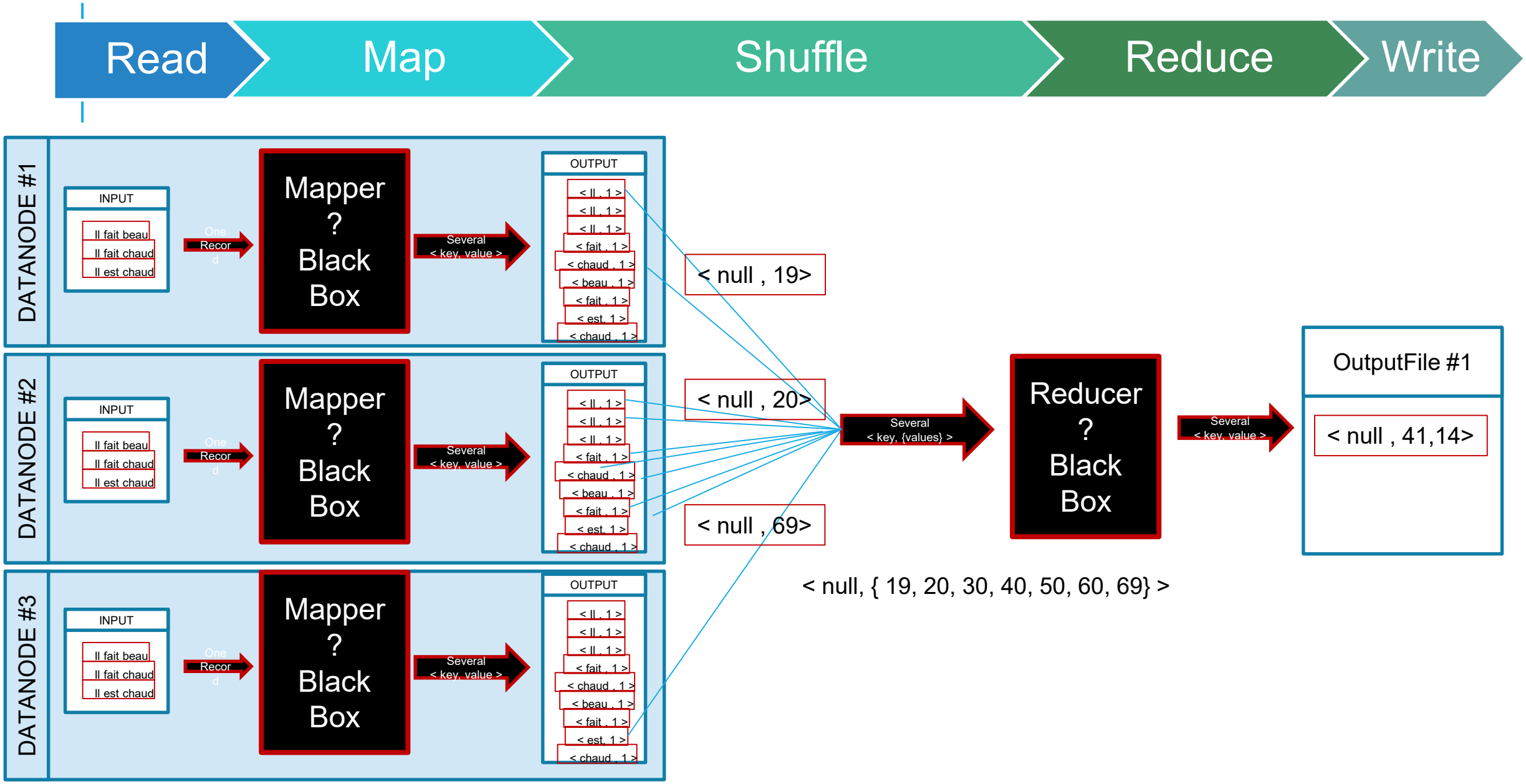
Soit $v_0, v_1, v_2, ..., v_{n-1}$ un ensemble de valeurs et $i \in]0..n-1[$ et $i \neq n/2$.

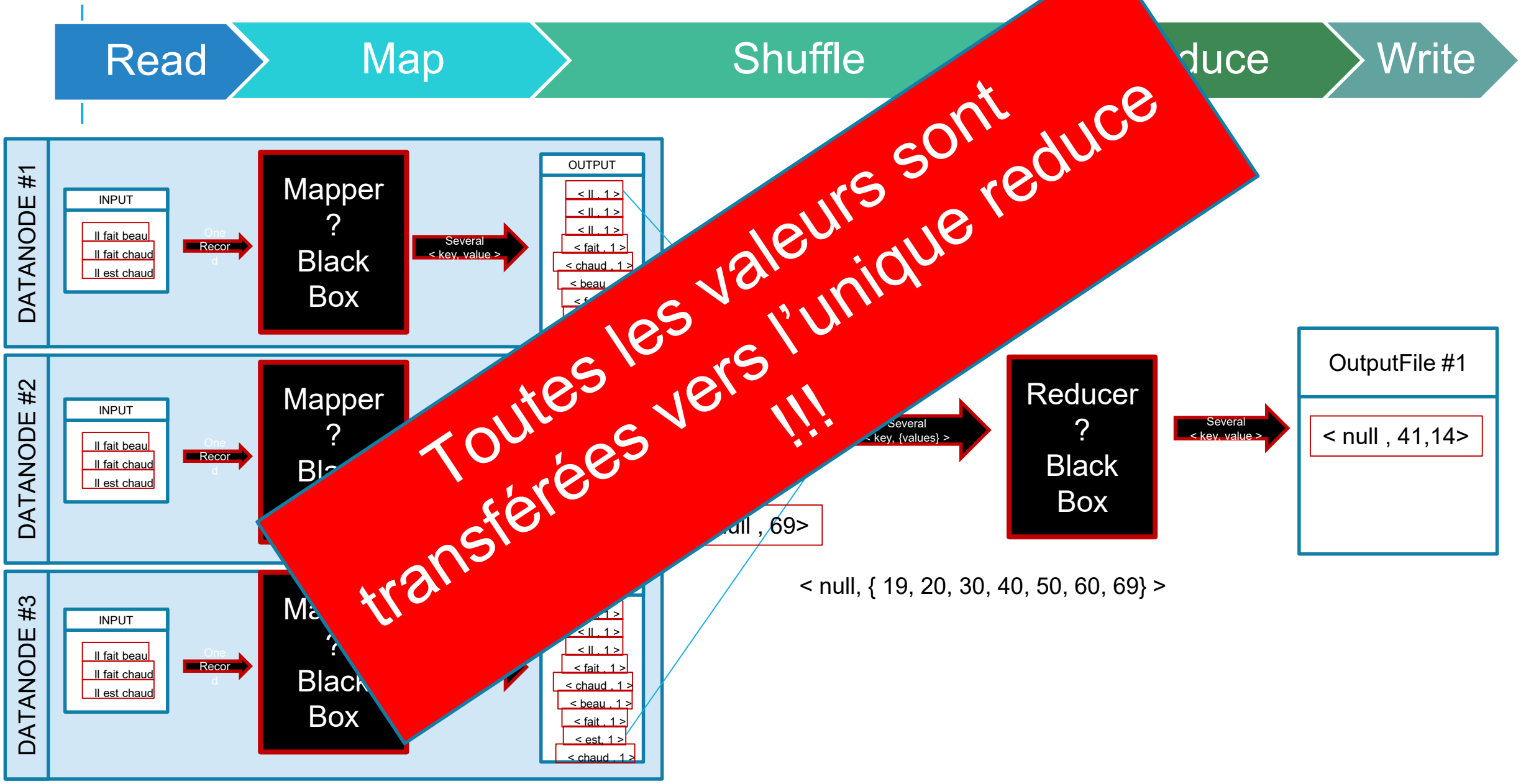
$$\frac{1}{i}\sum_{j=0}^{i-1} v_j + \frac{1}{n-i}\sum_{j=i}^{n-1} v_j \neq \frac{1}{n}\sum_{j=0}^{n-1} v_j$$

La moyenne des moyennes des partitions d'un ensemble de valeurs n'est pas égale à la moyenne des valeurs de l'ensemble. Il n'est donc pas possible d'utiliser le combiner pour faire des moyennes partielles.

Exemple de Writable pour le calcul de la moyenne

```
public static class Mmapper extends Mapper<Object, Text, NullWritable, IntWritable>{
 public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
       String tokens[] = value.toString().split(",");
       try {
       context.write(null, new IntWritable(Integer.parseInt(tokens[4])));
       }catch (Exception e) {}
  }
 }
 public static class MReducer extends Reducer<NullWritable,IntWritable,NullWritable, DoubleWritable> {
   public void reduce(NullWritable key, Iterable<IntWritable> values,Context context
                      ) throws IOException, InterruptedException {
     int tmp = 0;
     int count = 0;
     for (IntWritable i: values) {
     tmp += i.get();
     ++count;
     }
     context.write(null, new DoubleWritable((double)tmp/(double)count));
   }
 }
```


Read
Map
Shuffle
Reduce
Write
DATANODE #1
INPUT
Il fait beau
Il fait chaud
Il est chaud
One Record
Mapper ? Black Box
Several < key, value >
OUTPUT
< Il , 1 >
< Il , 1 >
< Il , 1 >
< fait , 1 >
< chaud , 1 >
< beau , 1 >
< fait , 1 >
< est, 1 >
< chaud , 1 >
DATANODE #2
INPUT
Il fait beau
Il fait chaud
Il est chaud
One Record
Mapper ? Black Box
Several < key, value >
OUTPUT
< Il , 1 >
< Il , 1 >
< Il , 1 >
< fait , 1 >
< chaud , 1 >
< beau , 1 >
< fait , 1 >
< est, 1 >
< chaud , 1 >
DATANODE #3
INPUT
Il fait beau
Il fait chaud
Il est chaud
One Record
Mapper ? Black Box
Several < key, value >
OUTPUT
< Il , 1 >
< Il , 1 >
< Il , 1 >
< fait , 1 >
< chaud , 1 >
< beau , 1 >
< fait , 1 >
< est, 1 >
< chaud , 1 >
< null , 19>
< null , 20>
< null , 69>
Several < key, {values} >
< null, { 19, 20, 30, 40, 50, 60, 69} >
Reducer ? Black Box
Several < key, value >
OutputFile #1
< null , 41,14>


Read
Map
Shuffle
duce
Write
DATANODE #1
DATANODE #2
DATANODE #3
INPUT
Il fait beau
Il fait chaud
Il est chaud
One Record
Mapper ? Black Box
Several < key, value >
OUTPUT
< Il , 1 >
< fait , 1 >
< chaud , 1 >
< beau , 1 >
< est, 1 >
Toutes les valeurs sont transférées vers l'unique reduce !!!
ull , 69>
Several < key, {values} >
Reducer ? Black Box
< null, { 19, 20, 30, 40, 50, 60, 69} >
Several < key, value >
OutputFile #1
< null , 41,14>

Exemple de Writable pour le calcul de la moyenne

On peut rendre le calcul de la moyenne distribuable en séparant le calcul de la somme des valeurs de la division par le nombre de valeurs !

Soit $v_0, v_1, v_2, ..., v_{n-1}$ un ensemble de valeurs et $i \in]0..n-1[$ et $i \neq n/2$.

$$\frac{1}{i}\sum_{j=0}^{i-1} v_j + \frac{1}{n-i}\sum_{j=i}^{n-1} v_j \neq \frac{1}{n}\sum_{j=0}^{n-1} v_j$$

$$\frac{1}{i+(n-i)} \cdot \left(\sum_{j=0}^{i-1} v_j + \sum_{j=i}^{n-1} v_j \right) = \frac{1}{n}\sum_{j=0}^{n-1} v_j$$

Le calcul de la moyenne peut ainsi être fait de manière distribué en calculant deux sommes, la somme des tailles des partitions et la somme des valeurs. L'addition étant associative, on peut faire le calcul en utilisant le combiner.

Exemple de Writable pour le calcul de la moyenne

```
public static class AvgWritable implements Writable {
  public int sum;
  public int count;
  public AvgWritable(int sum, int count) {
    this.sum = sum;
    this.count = count;
  }
  public void readFields(DataInput in) throws IOException {
    sum   = in.readInt();
    count = in.readInt();
  }
  public void write(DataOutput out) throws IOException {
    out.writeInt(sum);
    out.writeInt(count);
  }
}
```

Exemple de Writable pour le calcul de la moyenne

```
public static class M2Mapper
extends Mapper<Object, Text, NullWritable, AvgWritable>{
public void map(LongWritable key, Text value, Context context)
  throws IOException, InterruptedException
  {
   String tokens[] = value.toString().split(",");
   try {
     context.write(null, new AvgWritable(Integer.parseInt(tokens[4]), 1));
   }
   catch (Exception e) {
   }
  }
}
```

Exemple de Writable pour le calcul de la moyenne

```
public static class M2Combiner
extends Reducer<NullWritable,AvgWritable,NullWritable, AvgWritable> {
public void reduce(NullWritable key, Iterable<AvgWritable> values,Context context )
        throws IOException, InterruptedException {
  int partialSum = 0;
  int partialCount = 0;
  for (AvgWritable i: values) {
    partialCount += i.count;
    partialSum += i.sum;
  }
  context.write(null, new AvgWritable(partialSum, partialCount));
 }
}
```

Exemple de Writable pour le calcul de la moyenne

```
public static class M2Reducer
extends Reducer<NullWritable,AvgWritable,NullWritable, DoubleWritable> {
public void reduce(NullWritable key, Iterable<AvgWritable> values, Context context)
        throws IOException, InterruptedException {
  double totalSum = 0;
  double totalCount = 0;
  for (AvgWritable i: values) {
    totalCount += i.count;
    totalSum += i.sum;
  }
  context.write(null, new DoubleWritable(totalSum/totalCount));
 }
}
```

Hadoop Sharing: Distributed Cache

Pour partager des informations entre les maper et les reducer, Hadoop offre la possibilité de partagé des fichiers. Le système s'occupe automatique de copier les fichiers vers les machines ou s'exécutent les map/reduce.

```
public static class AMapper extends Mapper<KeyIn, ValueIn, KeyOut, ValueOut> {
        protected void setup(org.apache.hadoop.mapreduce.Mapper.Context context) {
          URI[] files = context.getCacheFiles(context.getConfiguration());
          DataInputStream strm = new DataInputStream(new FileInputStream(files[0].getPath());
          MyDataWritable data = new MyDataWritable();
          data.readFields(strm);
          strm.close();
        }
}

Main
        job.addCacheFile(new Path(filename).toUri());
```

Hadoop shuffle : Modèle de Partionner

```
public abstract class Partitioner<KEY, VALUE> {
  public abstract int getPartition(KEY key, VALUE value, int numPartitions);
}

public class HashPartitioner<K, V> extends Partitioner<K, V> {
  public int getPartition(K key, V value, int numReduceTasks) {
    return (key.hashCode() & Integer.MAX_VALUE) % numReduceTasks;
  }
}
```