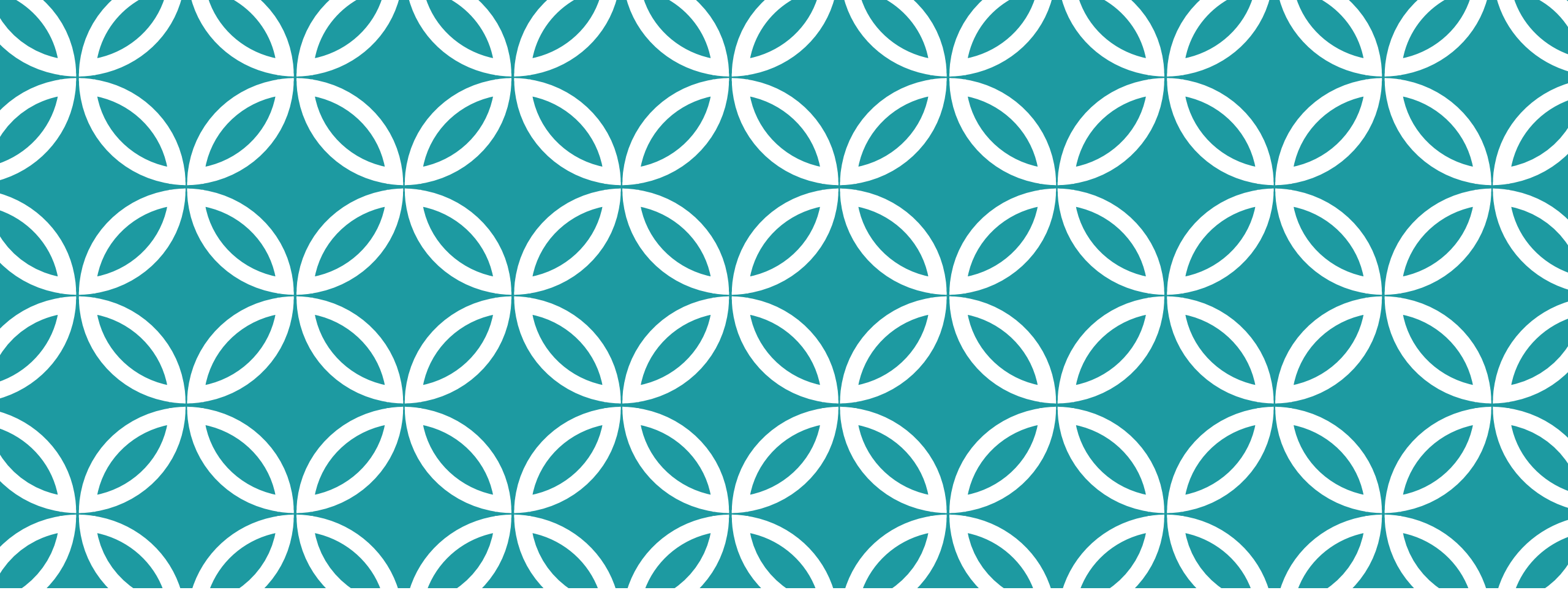


Map/Reduce

Fonctionnement
Outils
Utilisation
Programmation

Idée

HDFS permet de distribuer les données sur de nombreuses machines. De nombreux clusters de plus de 500 machines existent (voir site hadoop systems). Cependant la problématique du BigData n'est pas d'archiver les données mais de permettre d'effectuer des calculs sur ces données. Le vrai challenge est donc de permettre une utilisation efficace de ces données.

Une des prouesses de Hadoop est d'apporter une solution tant sur la problématique de stockage que sur la problématique de calcul. L'idée est simple si on ne peut pas transférer les données vers un serveur de calcul on peut transférer les algorithmes sur les serveurs de stockages. Cela permet d'utiliser la puissance des machines de stockage et de créer un cluster de calcul distribué au-dessus du cluster de stockage distribué.

Map Reduce

La solution proposées dans Hadoop est d'utiliser les paradigme Map Reduce pour implémenter des algorithmes distribués de traitement de données.

Le paradigme Map Reduce est théoriquement très simple. Un programme va être composé uniquement de deux fonctions. Une fonction Map et une fonction Reduce. Chaque fonction a le droit d'allouer de la mémoire et de faire ce qu'elle veut mais la seule contrainte est que les entrés et les sorties de ces deux fonctions sont imposées.

L'avantage de l'utilisation de cette approche est qu'elle va permettre d'utiliser le découpage en bloc de HDFS distribuer les lectures/traitement des données.

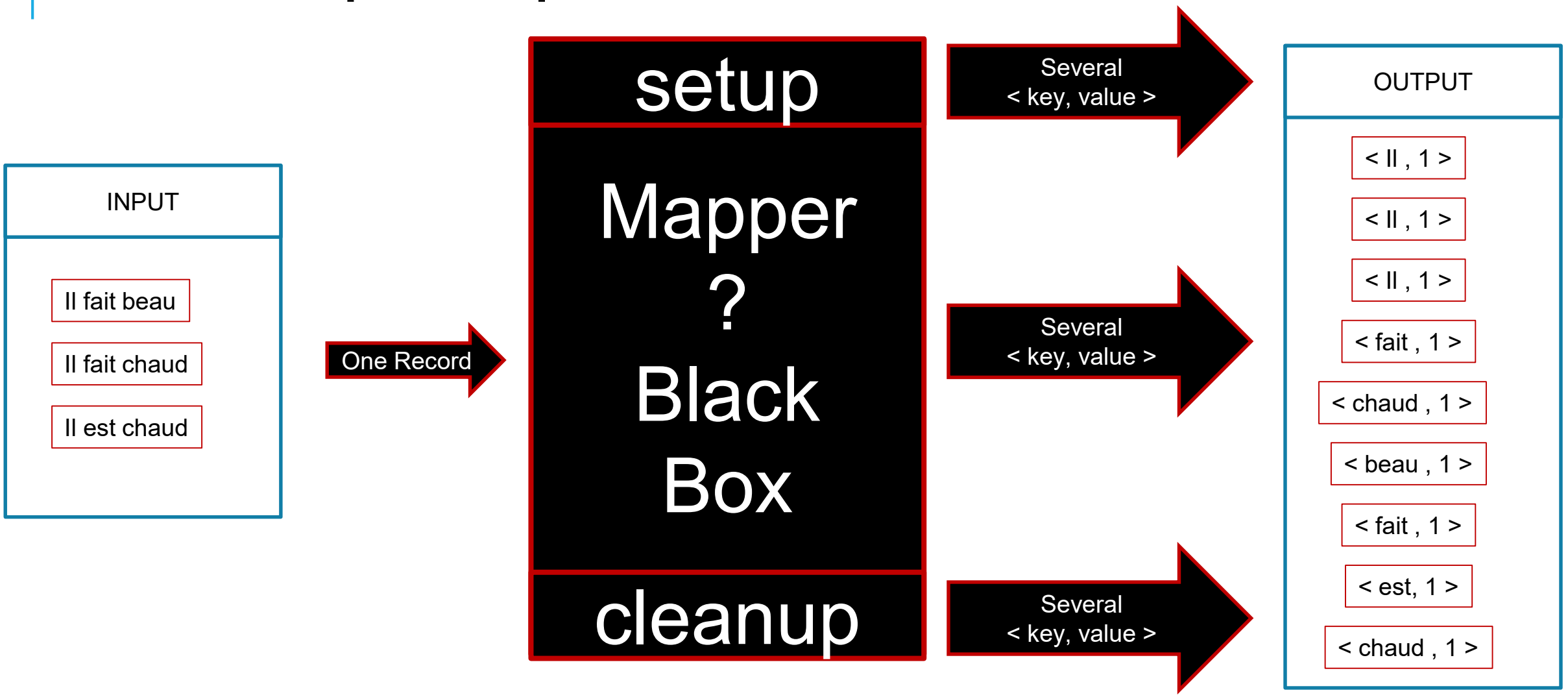
Hadoop Map

Le mapper va recevoir les données une à une à partir des blocs HDFS. En fonction du format d'entrée utilisé, les blocs peuvent être lus de manières différentes. Pour des textes, le mapper reçoit le fichier ligne à ligne, pour des images il peut le recevoir image par image etc…
Hadoop offre la possibilité avec les InputSplit et les RecordReader générer en fonction de ses besoins l'entrée des mappeurs.

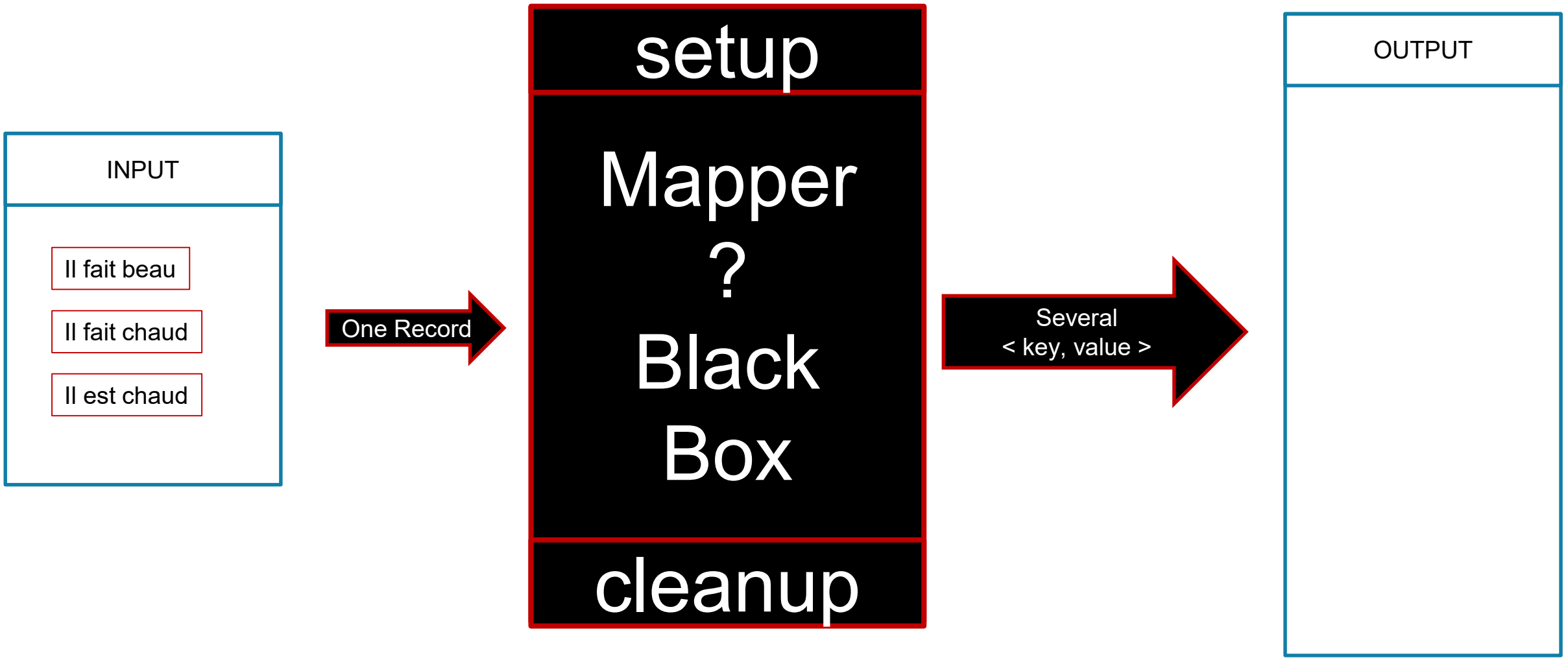
Le mappeur fonctionne en trois temps:
- Initialisation (setup) : L'initialisation permet d'échanger des infomations en mapper.
- Le map : le flux de message est envoyé au mapper.
- Le nettoyage (cleanup): Lorsque le traitement du flux est terminé.

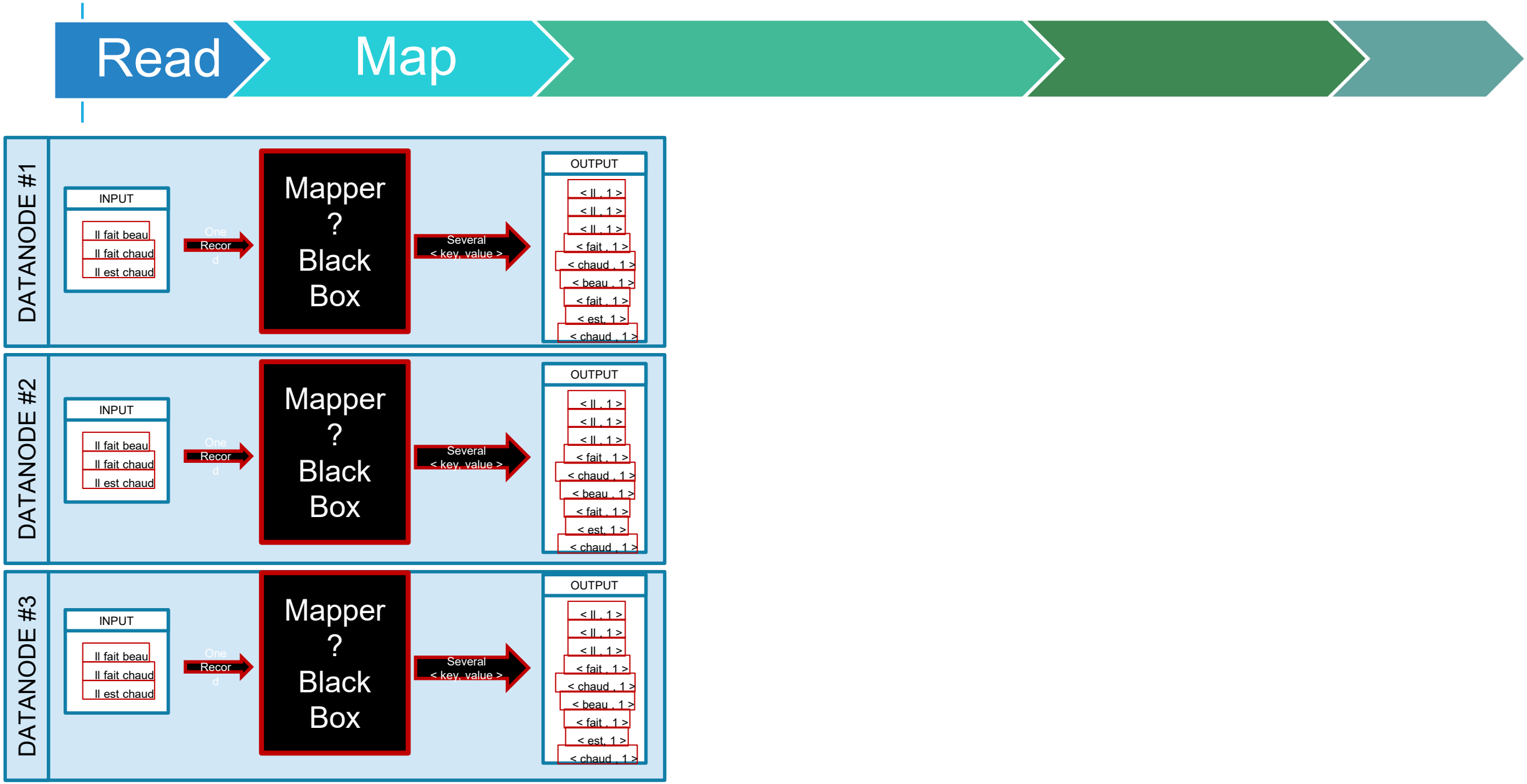
Le mappeur peut envoyer des messages pendant toutes ces opérations. Ce découpage en trois partie permet certaine synchronisation.

Hadoop Map


INPUT
Il fait beau
Il fait chaud
Il est chaud
One Record
setup
Mapper
?
Black
Box
cleanup
Several
< key, value >
Several
< key, value >
Several
< key, value >
OUTPUT
< Il , 1 >
< Il , 1 >
< Il , 1 >
< fait , 1 >
< chaud , 1 >
< beau , 1 >
< fait , 1 >
< est, 1 >
< chaud , 1 >

Hadoop Map


INPUT
Il fait beau
Il fait chaud
Il est chaud
One Record
setup
Mapper
?
Black
Box
cleanup
Several
< key, value >
OUTPUT

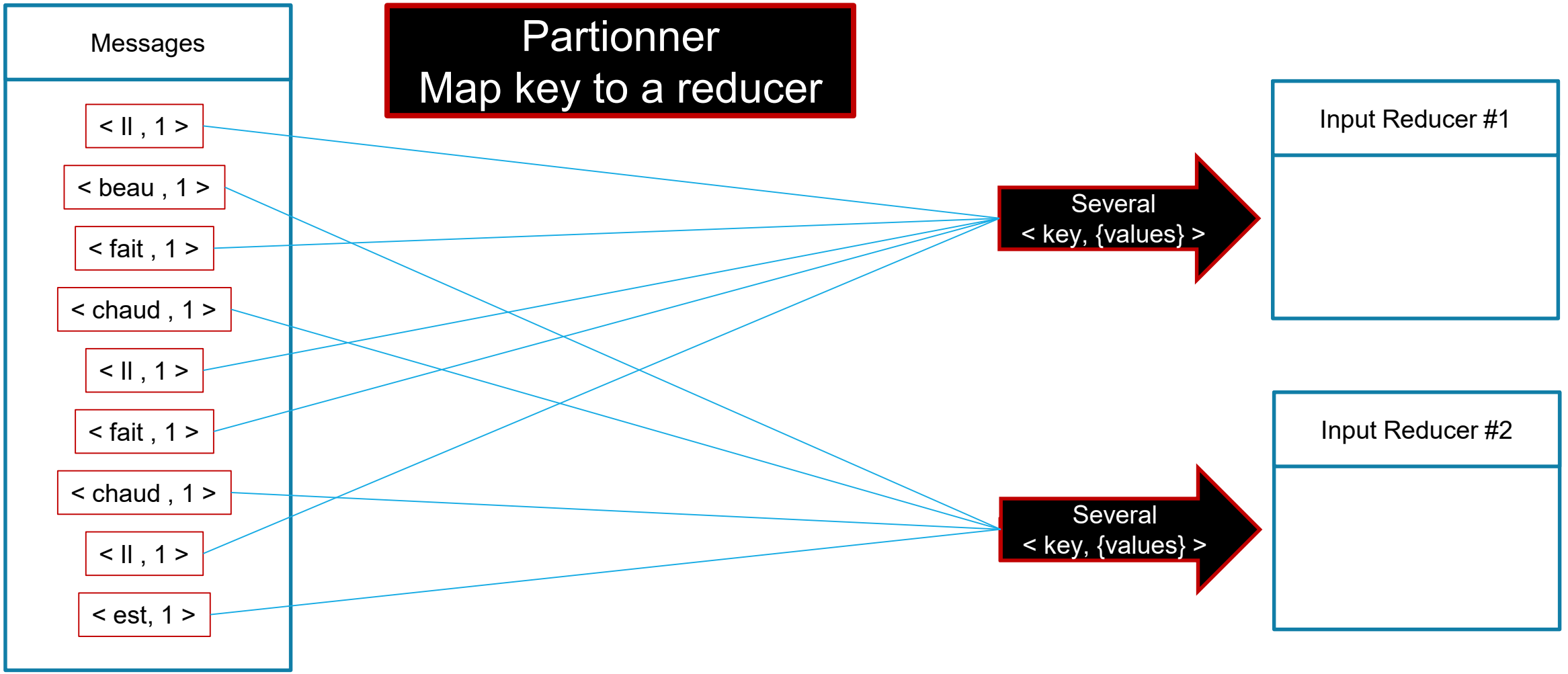

Read
Map
DATANODE #1
INPUT
Il fait beau
Il fait chaud
Il est chaud
One Record
Mapper ? Black Box
Several < key, value >
OUTPUT
< Il , 1 >
< Il , 1 >
< Il , 1 >
< fait , 1 >
< chaud , 1 >
< beau , 1 >
< fait , 1 >
< est, 1 >
< chaud , 1 >
DATANODE #2
INPUT
Il fait beau
Il fait chaud
Il est chaud
One Record
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 >

Hadoop Shuffling

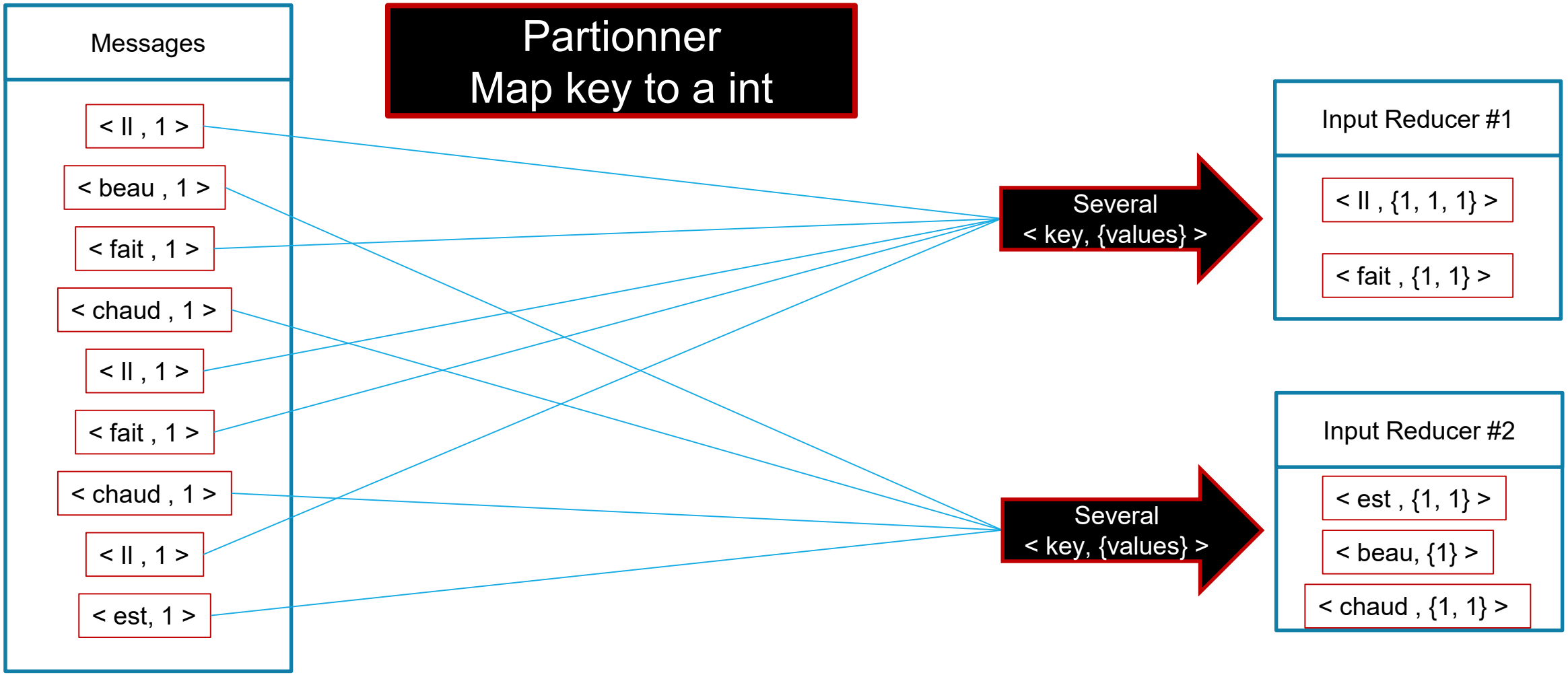
Le « shuffling » correspond à la distribution des messages envoyés par les mappeurs. Les messages vont sont automatiquement groupé ensemble en fonction de leur clé et ils sont envoyé à un reducer (machine, programme) en fonction d'une fonction de partionnement. Cette fonction est de la forme K -> N. K est l'ensemble des clés et N est est entier.

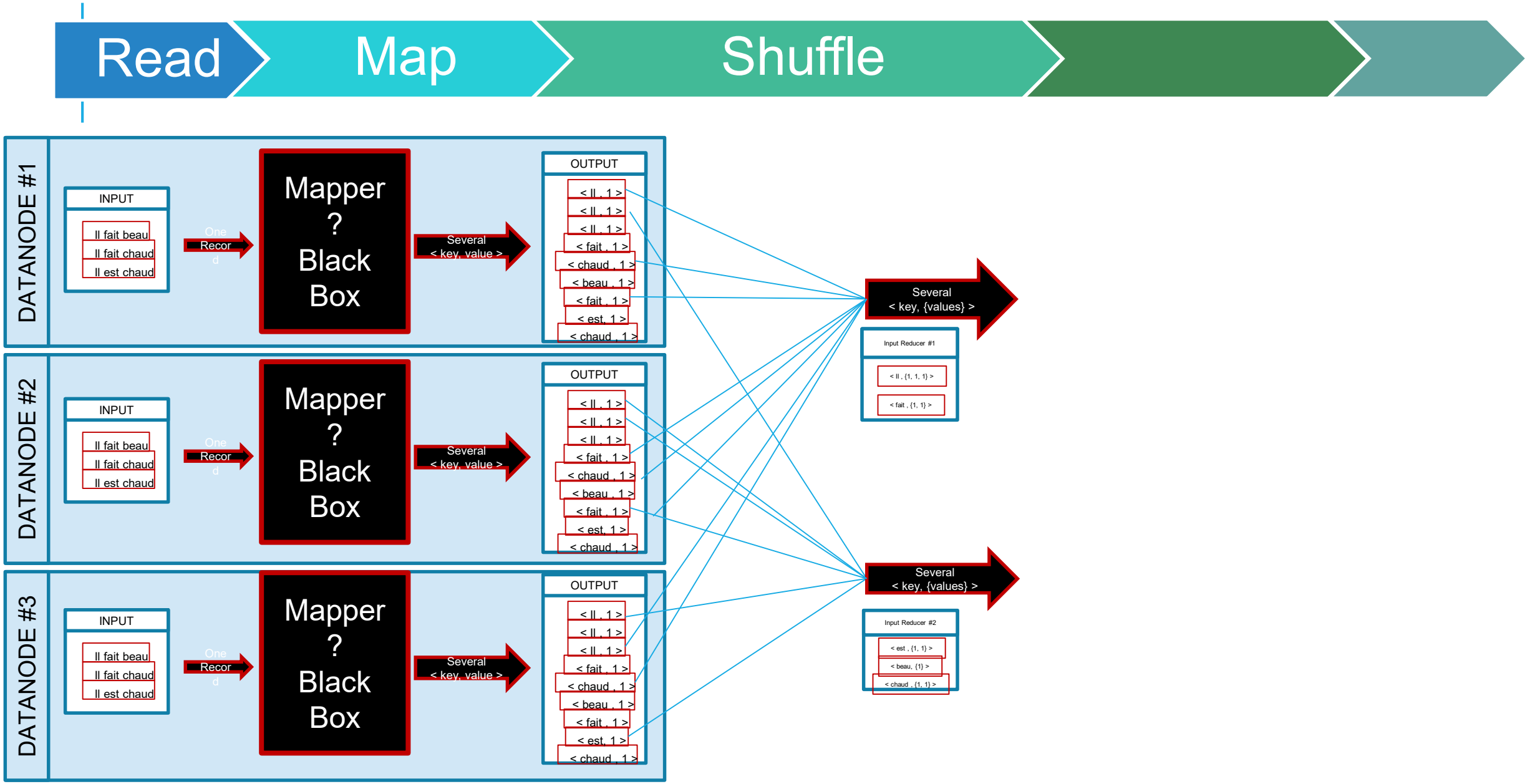
Par défaut hadoop fournit une fonction de distribution qui utilise la fonction de hash de la clé et un modulo sur le nombre de machine disponible pour traiter les sorties. Pour améliorer l'équilibrage de charge des algorithme on peut créer ses propres fonction de partionnement.

Shuffling


Messages
< Il , 1 >
< beau , 1 >
< fait , 1 >
< chaud , 1 >
< Il , 1 >
< fait , 1 >
< chaud , 1 >
< Il , 1 >
< est, 1 >
Partionner
Map key to a reducer
Several
< key, {values} >
Several
< key, {values} >
Input Reducer #1
Input Reducer #2

Shuffling


Messages
< Il , 1 >
< beau , 1 >
< fait , 1 >
< chaud , 1 >
< Il , 1 >
< fait , 1 >
< chaud , 1 >
< Il , 1 >
< est, 1 >
Partionner
Map key to a int
Several
< key, {values} >
Several
< key, {values} >
Input Reducer #1
< Il , {1, 1, 1} >
< fait , {1, 1} >
Input Reducer #2
< est , {1, 1} >
< beau, {1} >
< chaud , {1, 1} >


Read
Map
Shuffle
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 >
< Il , 1 >
< Il , 1 >
< fait , 1 >
< chaud , 1 >
< beau , 1 >
< fait , 1 >
< est, 1 >
< chaud , 1 >
Several < key, {values} >
Input Reducer #1
< Il , {1, 1, 1} >
< fait , {1, 1} >
Several < key, {values} >
Input Reducer #2
< est , {1, 1} >
< beau, {1} >
< chaud , {1, 1} >

Hadoop Reducer

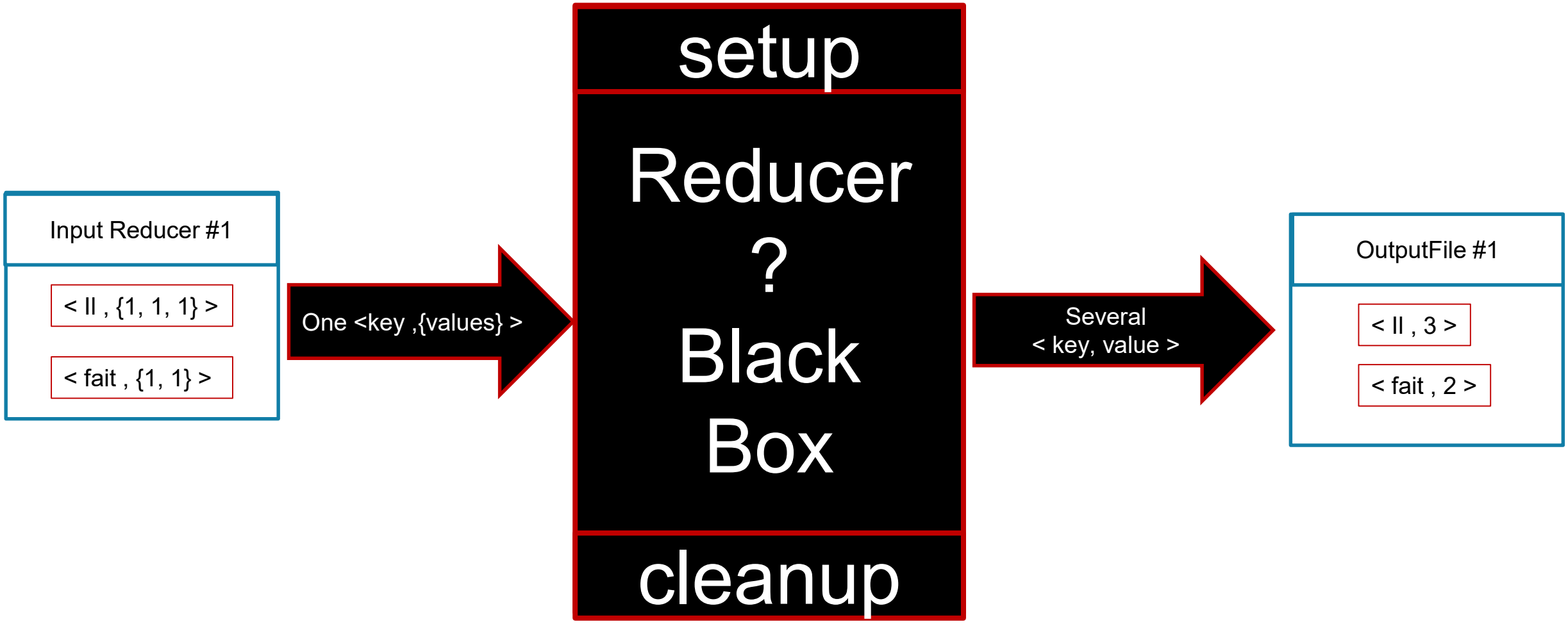
Le « reducer » est la brique qui va s'occuper d'appliquer un traitement à toutes les valeurs ayant la même clé.

Le reducer reçoit en entré des couples <clé, ensemble de valeurs>. À partir de ces couples ils va renvoyer couples <clé, valeur>. Le type des valeurs en entrée d'un reducer peuvent être différent des types des valeurs à la sortie d'un reducer. La sortie est directement enregistré en utilisant un OuputFormat. Le format de sortie peut être du texte, un sequencefile ou encore une base de données. Il est possible de écrire sont propre format de sortie.

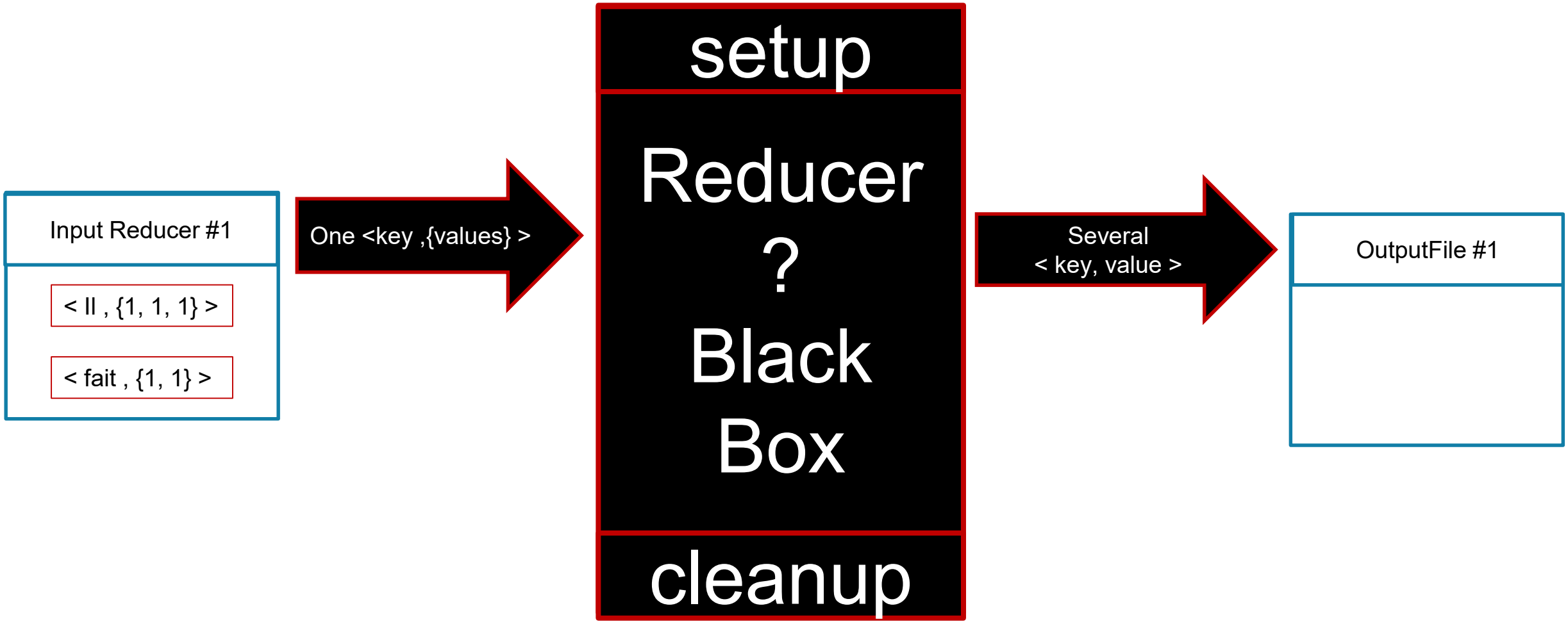
Le reducer fonctionne en trois temps (comme le mappeur) :
- Initialisation (setup) : L'initialisation permet d'échanger des infomations en mapper.
- Le reduce : le flux de couple est envoyé au reducer.
- Le nettoyage (cleanup): Lorsque le traitement du flux est terminé.

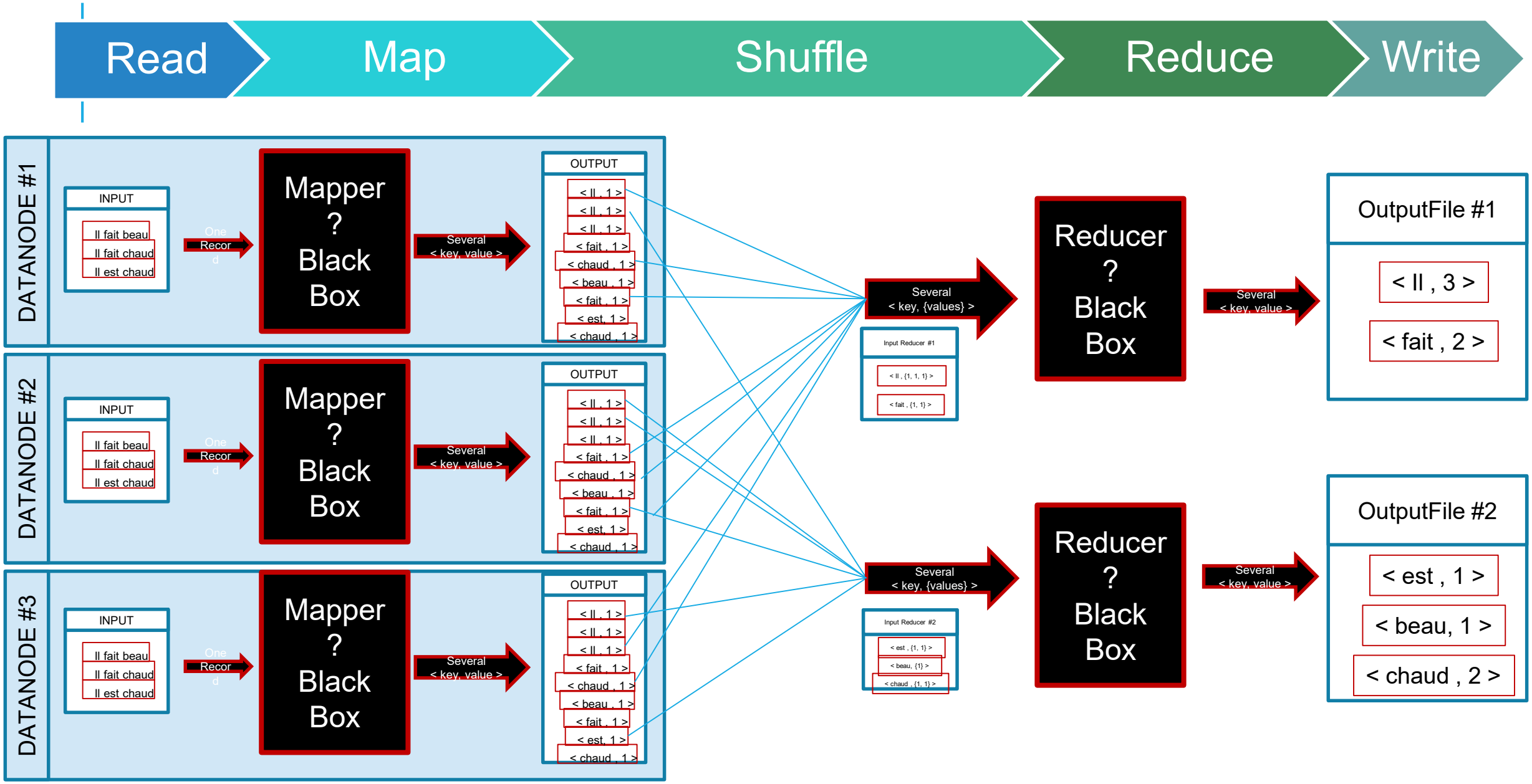
Le reducer peut envoyer des messages pendant toutes ces opérations.

Reducer


Input Reducer #1
< Il , {1, 1, 1} >
< fait , {1, 1} >
One <key ,{values} >
setup
Reducer
?
Black
Box
cleanup
Several
< key, value >
OutputFile #1
< Il , 3 >
< fait , 2 >

Reducer


Input Reducer #1
< Il , {1, 1, 1} >
< fait , {1, 1} >
One <key ,{values} >
setup
Reducer
?
Black
Box
cleanup
Several
< key, value >
OutputFile #1


Read
Map
Shuffle
Reduce
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 >
< Il , 1 >
< Il , 1 >
< fait , 1 >
< chaud , 1 >
< beau , 1 >
< fait , 1 >
< est, 1 >
< chaud , 1 >
Several < key, {values} >
Input Reducer #1
< Il , {1, 1, 1} >
< fait , {1, 1} >
Input Reducer #2
< est , {1, 1} >
< beau, {1} >
< chaud , {1, 1} >
Reducer ? Black Box
Several < key, value >
OutputFile #1
< Il , 3 >
< fait , 2 >
OutputFile #2
< est , 1 >
< beau, 1 >
< chaud , 2 >

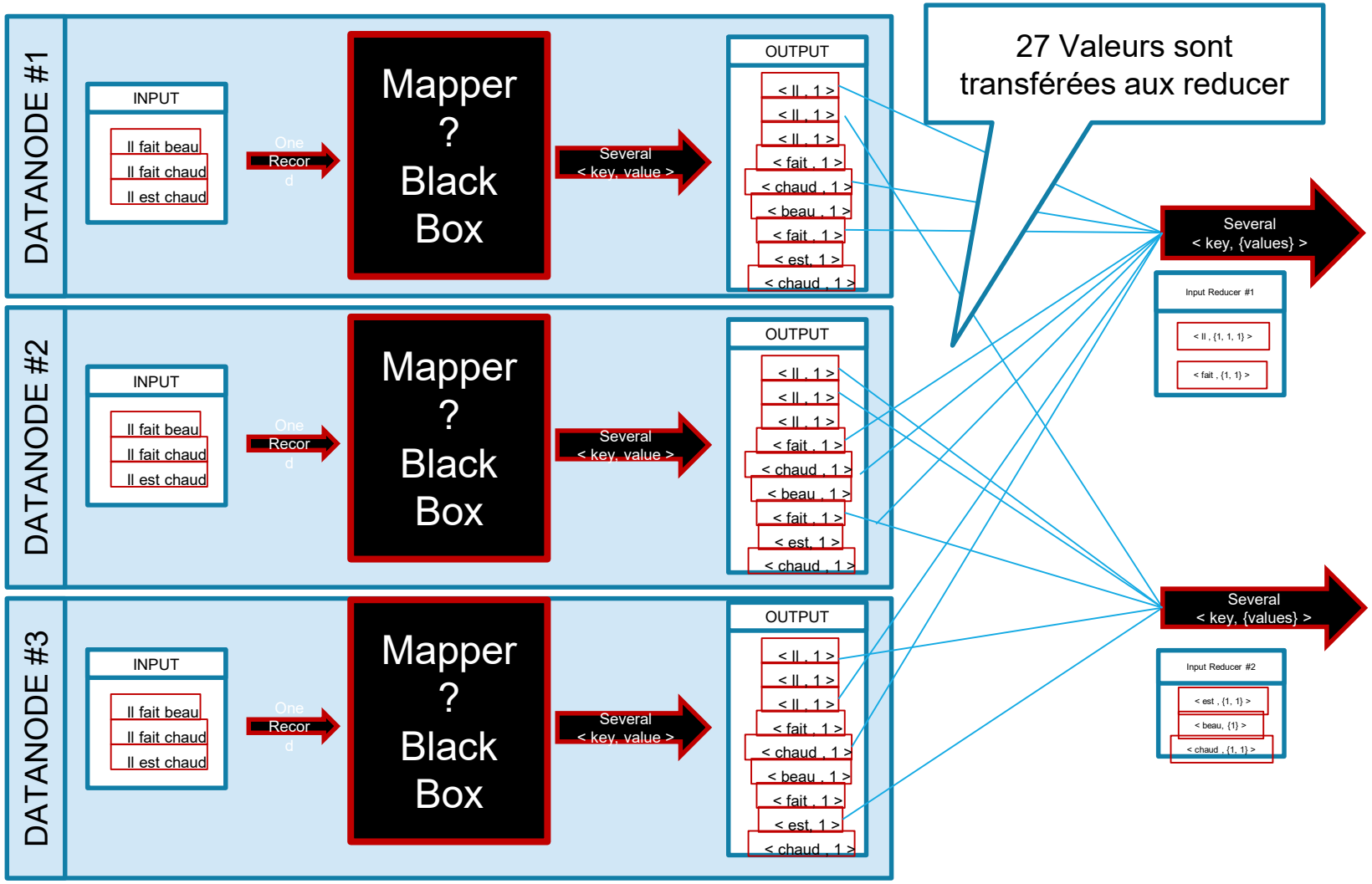
Hadoop Combiner

Pour fonctionner, le « reducer » doit recevoir l'ensemble des valeurs associé à une clé. Il faut donc transférer toutes ces valeurs aux « reducer ». Ce transfert peut engendrer une utilisation importante du réseau. De plus, la taille de l'ensemble de valeur peur être gigantesque (exemple: clé = homme). Dans ce cas seul un reducer traite cet ensemble et il n'y a pas de distribution du calcul.

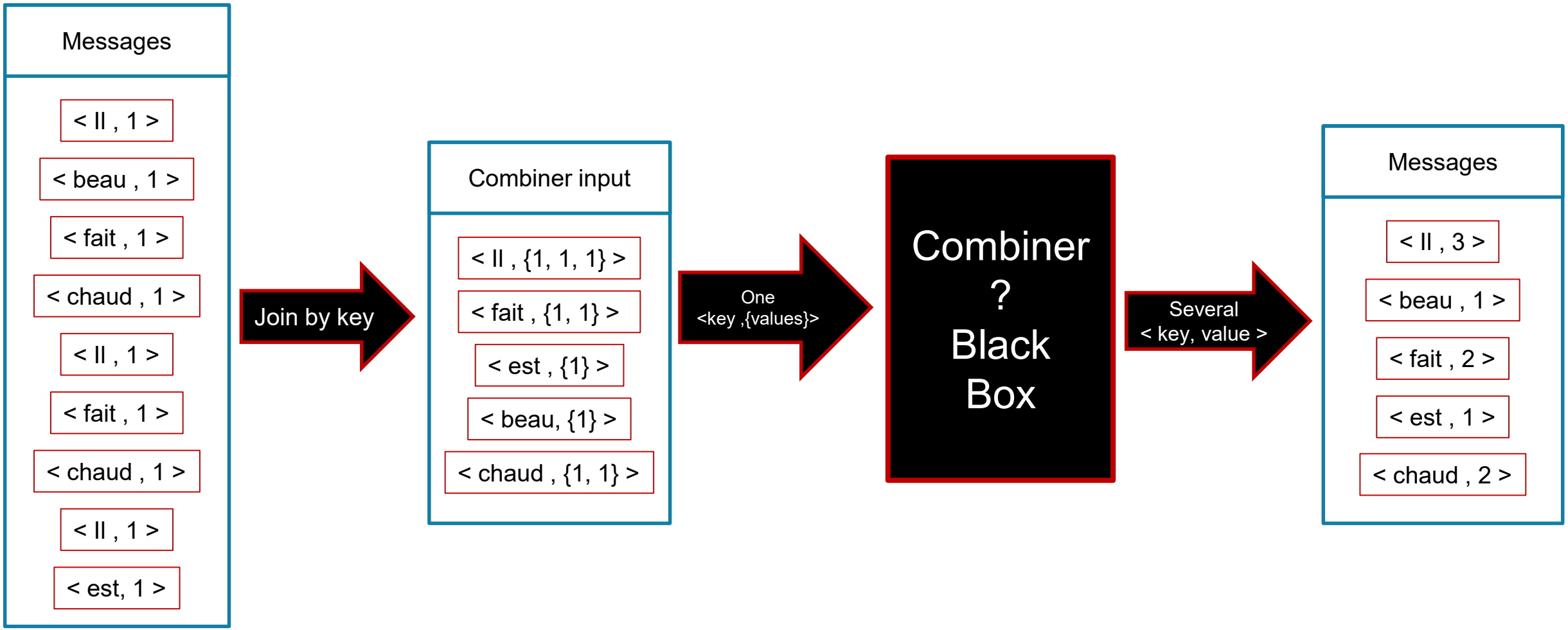
Pour remédier à ce problème, Hadoop permet d'effectuer des « reduce » partiels. Attention, il n'est pas toujours possible de faire ce genre de réduction et une partie de la reflexion dans la conception d'un programme Map/Reduce est d'arriver à créer des fonctions associatives qui peuvent être réduite partiellement.

Le combiner fonctionne exactement comme le « reducer » sauf qu'il est exécuté au même emplacement que le mappeur dont il pré-traite la sortie. Le code et les I/O du combiner peuvent être complètement différent de ceux du reducer.

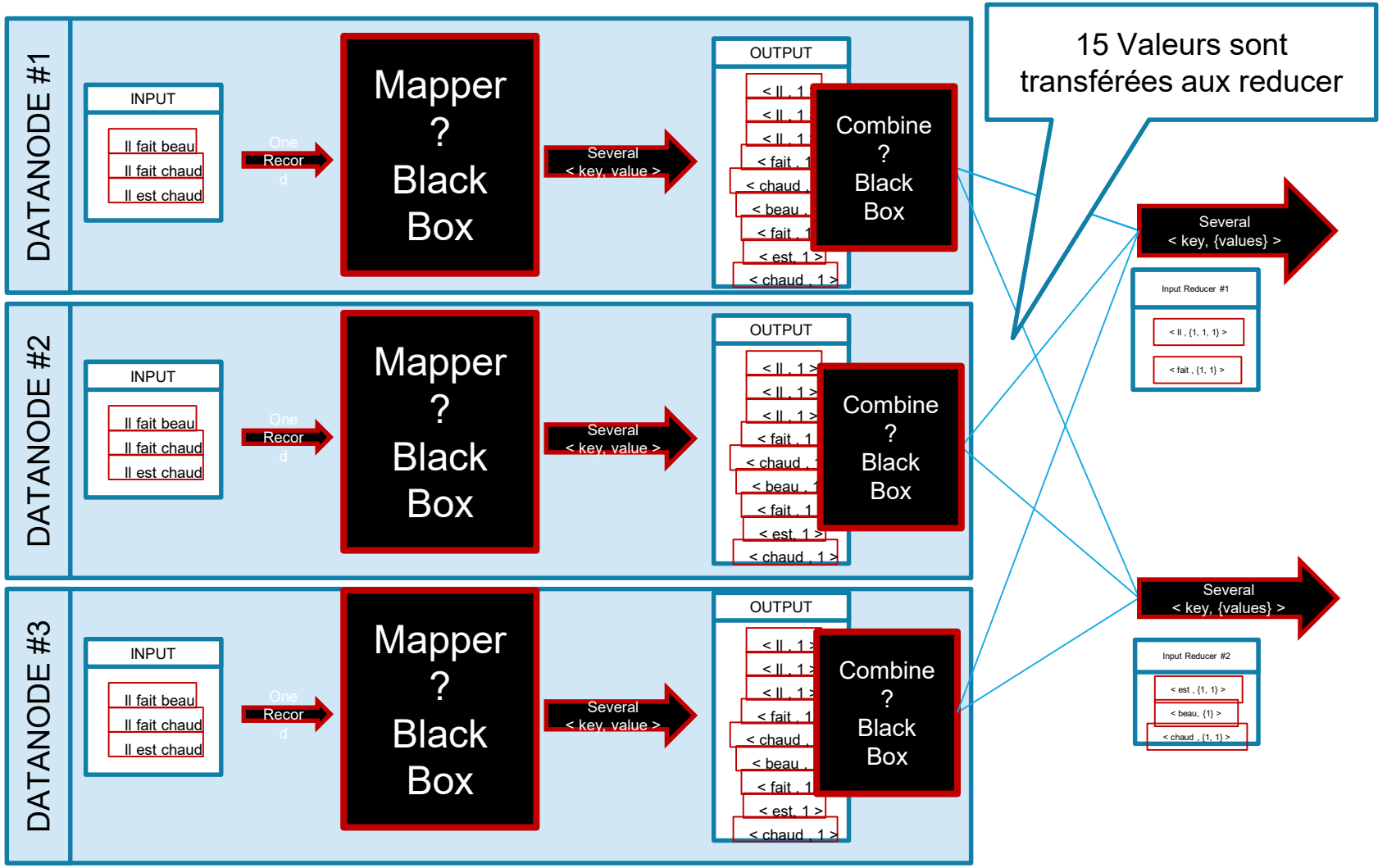
Hadoop Combiner

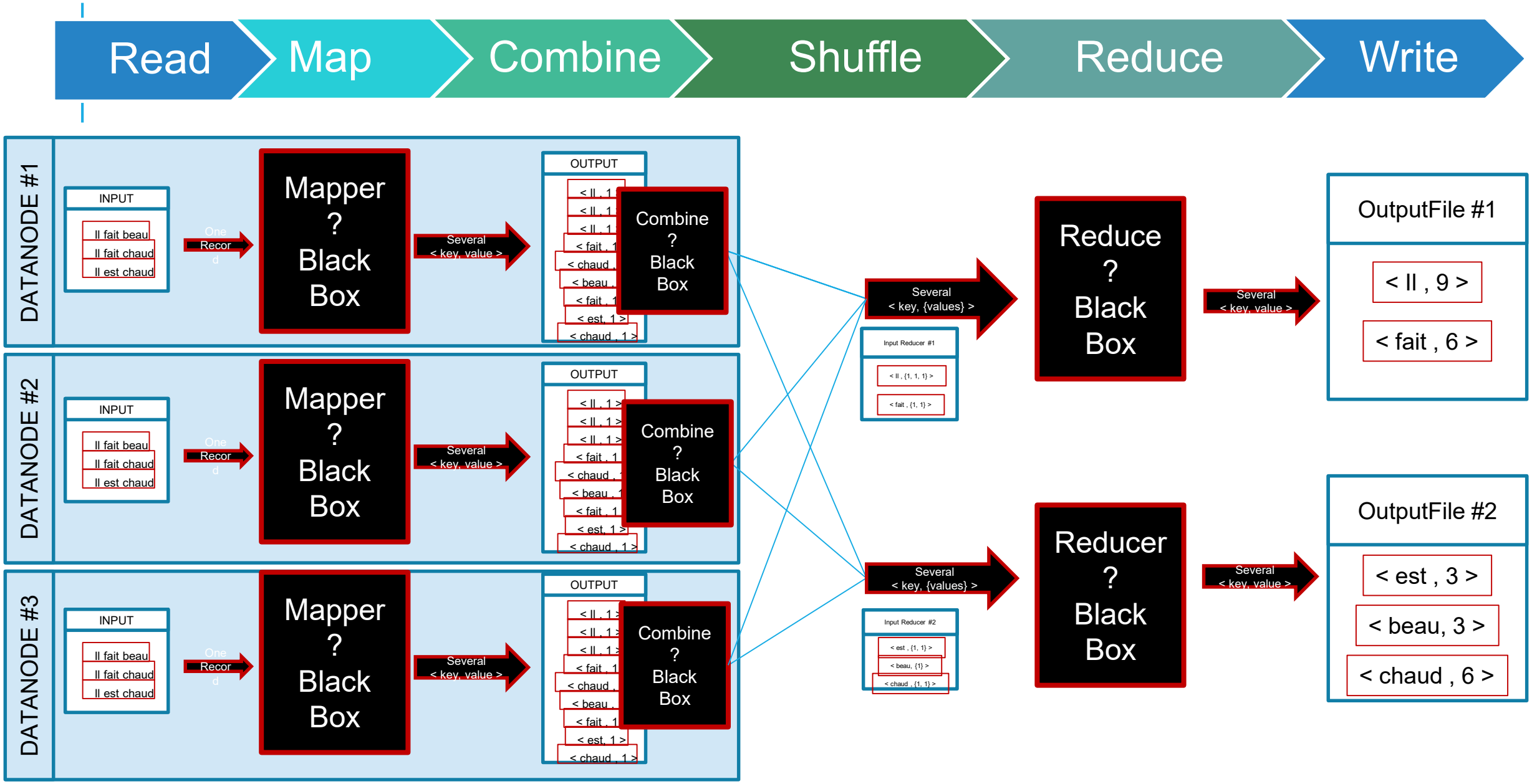


Hadoop Combiner



Combiner


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 >
Combine ? Black Box
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 >
Combine ? Black Box
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 >
Combine ? Black Box
15 Valeurs sont transférées aux reducer
Several < key, {values} >
Input Reducer #1
< Il , {1, 1, 1} >
< fait , {1, 1} >
Several < key, {values} >
Input Reducer #2
< est , {1, 1} >
< beau, {1} >
< chaud , {1, 1} >


Read
Map
Combine
Shuffle
Reduce
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 >
Combine ? Black Box
Several < key, {values} >
Input Reducer #1
< Il , {1, 1, 1} >
< fait , {1, 1} >
Input Reducer #2
< est , {1, 1} >
< beau, {1} >
< chaud , {1, 1} >
Reduce ? Black Box
Reducer ? Black Box
OutputFile #1
< Il , 9 >
< fait , 6 >
OutputFile #2
< est , 3 >
< beau, 3 >
< chaud , 6 >

Job Map Reduce

= Job

Un « Job » map reduce correspond au paramétrage de chaque bloc du la chaîne de traitement. Il faut donc fixer de :

- Le reader : appelé **InputFormat** en hadoop
- Le boite de map : appelé **Mapper** en hadoop
- Le combine : appelé **Combiner** en hadoop
- Le shuffling : appelé **Partionner** en hadoop
- Le reduce : appelé **Reducer** en hadoop
- Le writer : appelé **OutputFormat** en hadoop

Mais aussi la configuration topologique des blocs:

- Fichiers en entrée du job
- Fichiers en sortie du job
- Format de sortie des clés et des valeurs des reducers.
- Nombre de reducers

Une fois configuré le job peut être exécuté par le système Hadoop.

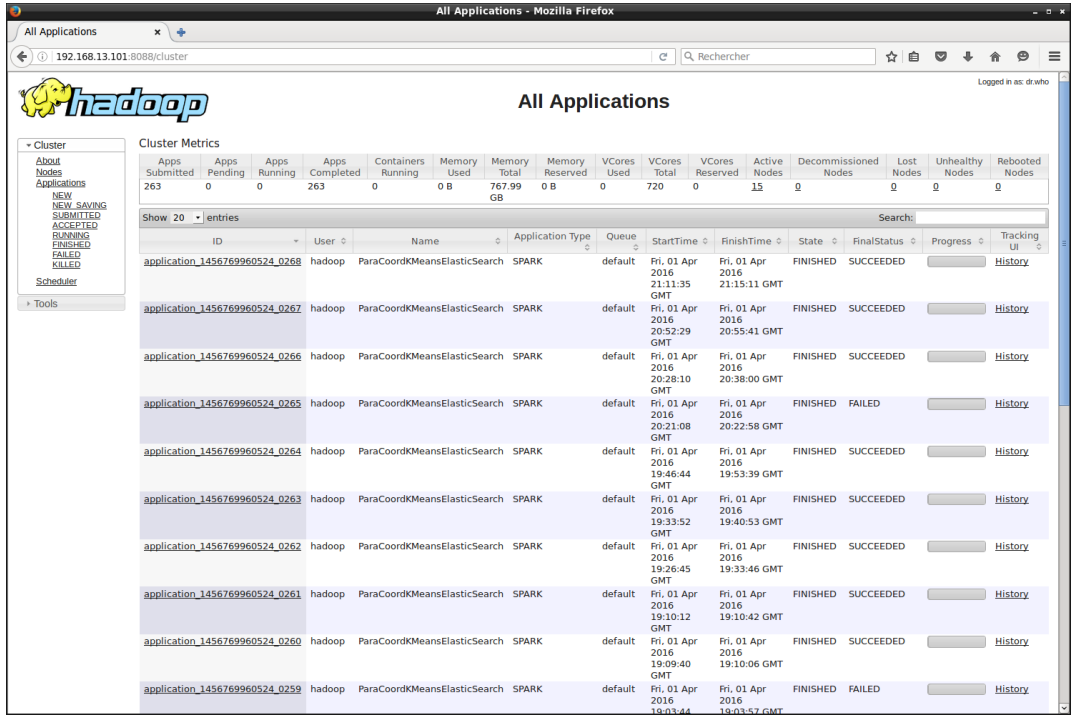
Lancement de Job Map Reduce

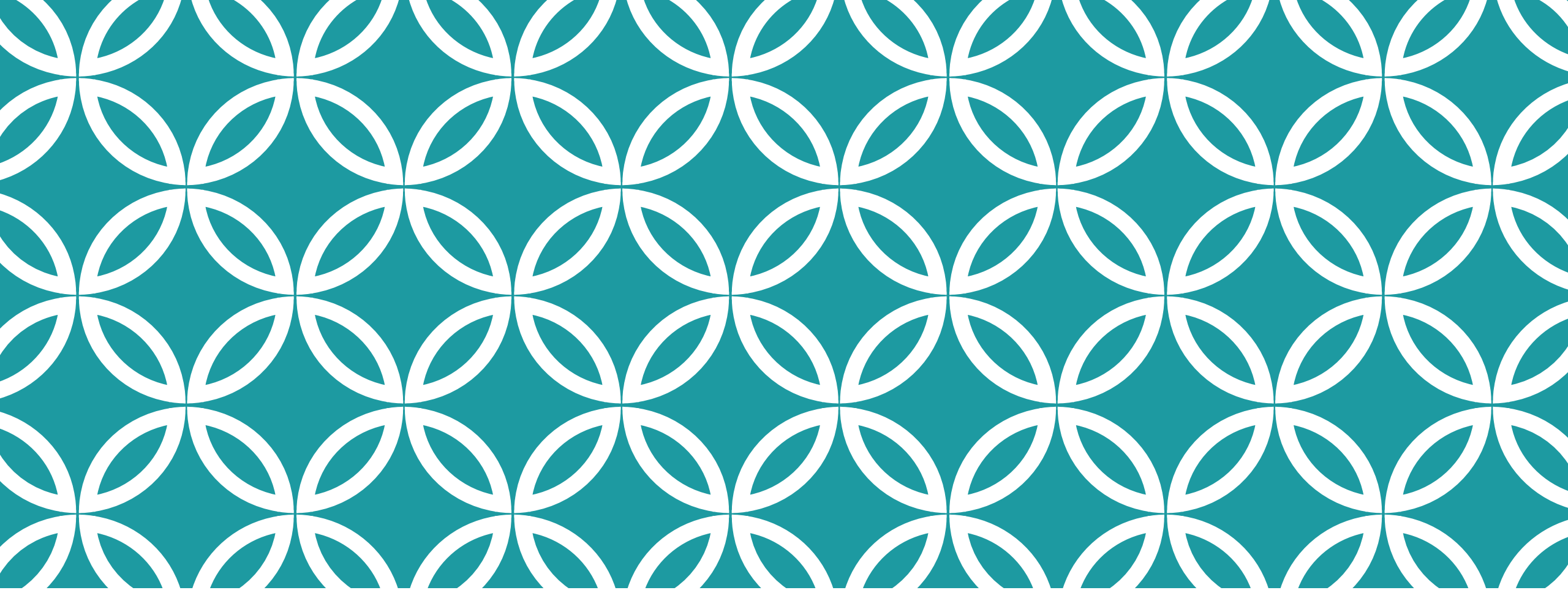
Read → Map → Combine → Shuffle → Reduce → Write = Job

Une fois compilé le job dans un .jar (archive java) on envoie le programme à hadoop via la commande:
yarn jar votre_fichier_jar

On peut suivre la progression de l'exécution des jobs vhosia l'interface web de yarn:
http://RESOURCEMANAGER:8088/

Attention les adresses des fichiers stockés dans hdfs sont de la forme:
hdfs://NAMENODE:9000/





Hadoop Job en pratique I

Fonctionnement
Outils
Utilisation
Programmation

Hadoop Driver : Modèle de Job

```
public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, “JobName");
        job.setJarByClass(MyProg.class);
        job.setInputFormatClass(FileInputFormat.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        job.setMapperClass(MyProgMapper.class);
        job.setCombinerClass(MyProgCombiner.class);
        job.setPartionerClass(HashPartitioner.class);
        job.setReducerClass(MyProgReducer.class);
        job.setNumReduceTasks(NbReducer);
        job.setOutputValueClass(IntWritable.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputFormatClass(FileOuputFomat.class);
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

Hadoop map : Modèle de Mapper

```
public static class AMapper extends Mapper<KeyIn, ValueIn, KeyOut, ValueOut> {
        protected void setup(org.apache.hadoop.mapreduce.Mapper.Context context) {
        }
        public void map(KeyIn key, ValueIn value, Context context ) throws IOException, InterruptedException {
                context.write(new KeyOut(),new KeyValue());
         }
        protected void cleanup(org.apache.hadoop.mapreduce.Mapper.Context context)
                throws IOException, InterruptedException {
        }
}

public static class NGramMapper extends Mapper<Object, Text, Text, IntWritable> {
  public void map(Object key, Text value, Context context ) throws IOException, InterruptedException {
      String tokens[] = value.toString().split("\\W+");
      for (int j = 0; j < tokens.length - 1; ++j) {
        word.set(tokens[j] + " " + tokens[j+1]);
        context.write(new Text(word), new IntWritable(1));
      }
 }
```

Hadoop reduce : Modèle de Reducer/Combiner

```
public class Reducer<KEYIN,VALUEIN,KEYOUT,VALUEOUT> {
  protected void setup(Context context) throws IOException, InterruptedException {}
  protected void reduce(KEYIN key, Iterable<VALUEIN> values, Context context) throws … {
    for(VALUEIN value: values) {
      context.write((KEYOUT) key, (VALUEOUT) value);
    }
  }
  protected void cleanup(Context context) throws IOException, InterruptedException {}
  public void run(Context context) throws IOException, InterruptedException {
    setup(context);
    try {
      while (context.nextKey())
        reduce(context.getCurrentKey(), context.getValues(), context);
    } finally {
      cleanup(context);
    }
  }
}
```

Hadoop reduce : Modèle de Reducer/Combiner

```
public class Reducer<KEYIN,VALUEIN,KEYOUT,VALUEOUT> {
  protected void setup(Context context) throws IOException, InterruptedException …
  protected void reduce(KEYIN key, Iterable<VALUEIN> values, Context context) throws …
  protected void cleanup(Context context) throws IOException, InterruptedException …
  public void run(Context context) throws IOException, InterruptedException ….
}
```

```
public static class NGramReducer extends Reducer<Text,IntWritable,Text,IntWritable> {
  public void reduce(Text key, Iterable<IntWritable> values,  Context context ) throws IOException, InterruptedException {
    int sum = 0;
    for (IntWritable val : values)
       sum += val.get();
     context.write(key, new IntWritbale(sum) );
   }
}
```