

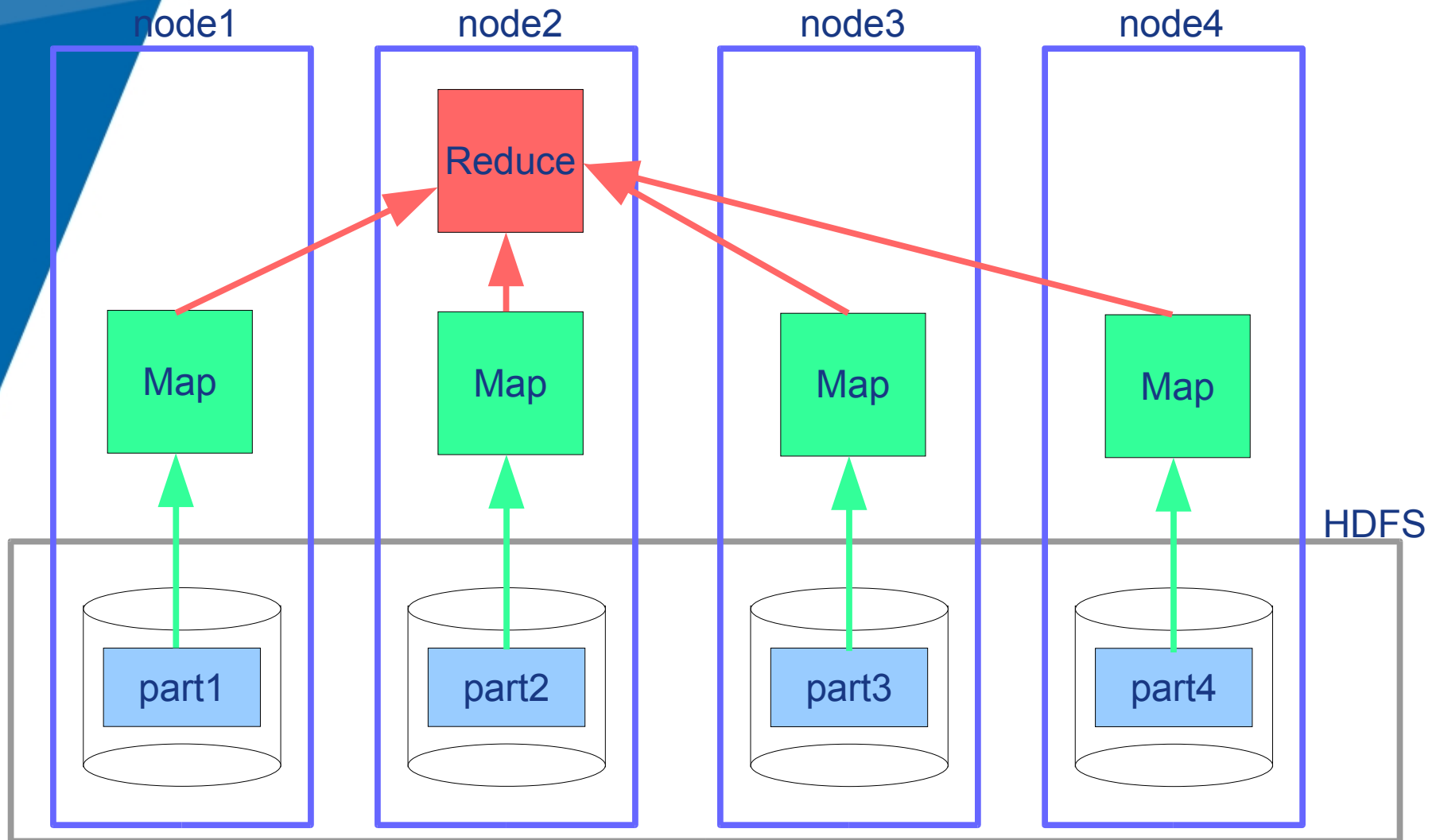
The background features a blue geometric design on the left side, consisting of several overlapping triangles and polygons in different shades of blue. The rest of the background is white.

Projet Hidoop

Implantation d'une infrastructure de calcul map-reduce

Description Hidoop v0

Principe du map-reduce



Exemple d'application comptage de mots

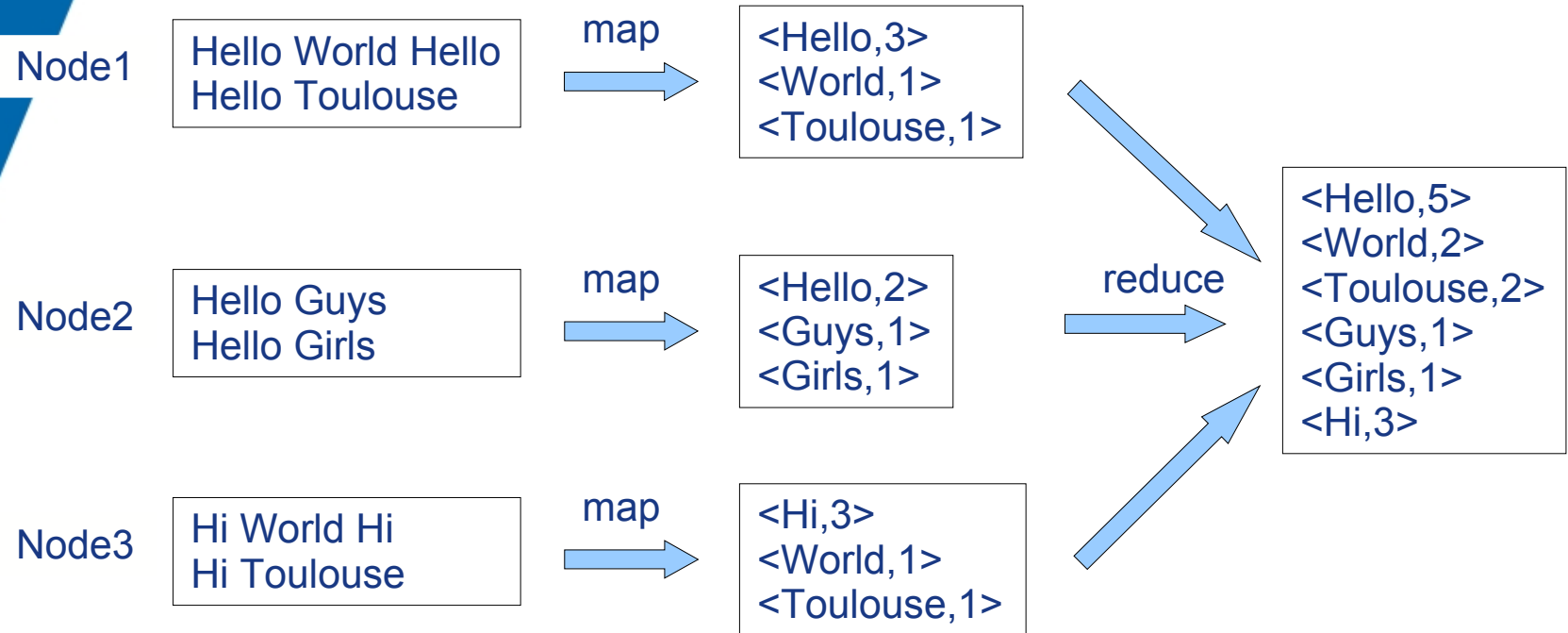
- En itératif

```
HashMap<String,Integer> hm = new HashMap<String,Integer>();

// ouvrir fichier à lire : Inr
while (true) {
    String l = Inr.readLine();
    if (l == null) break;
    String tokens[] = l.split(" ");
    for (String tok : tokens) {
        if (hm.containsKey(tok))
            hm.put(tok, hm.get(tok).intValue()+1);
        else
            hm.put(tok, 1);
    }
}

// recopier la hashmap dans le fichier résultat
```

Exemple d'application comptage de mots



Exemple d'application comptage de mots

- En map-reduce

Read de KV (<xxx,ligne> pour wordcount)

Write de KV (<w,n> pour wordcount)

```
public void map(FormatReader reader, FormatWriter writer) {  
  
    HashMap<String,Integer> hm = new HashMap<String,Integer>();  
    KV kv;  
    while ((kv = reader.read()) != null) {  
        String tokens[] = kv.v.split(" ");  
        for (String tok : tokens) {  
            if (hm.containsKey(tok))  
                hm.put(tok, hm.get(tok).intValue()+1);  
            else  
                hm.put(tok, 1);  
        }  
    }  
    for (String k : hm.keySet())  
        writer.write(new KV(k,hm.get(k).toString()));  
}
```

Exemple d'application comptage de mots

- En map-reduce

Read de KV (<w,n> wordcount)

Write de KV (<w,n> pour wordcount)

```
public void reduce(FormatReader reader, FormatWriter writer) {  
  
    HashMap<String,Integer> hm = new HashMap<String,Integer>();  
    KV kv;  
    while ((kv = reader.read()) != null) {  
        if (hm.containsKey(kv.k))  
            hm.put(kv.k, hm.get(kv.k)+Integer.parseInt(kv.v));  
        else  
            hm.put(kv.k, Integer.parseInt(kv.v));  
    }  
    for (String k : hm.keySet())  
        writer.write(new KV(k,hm.get(k).toString()));  
}
```

Lecture/écriture des données

- Dans un fichier du système de fichiers local
- Dans un fragment dans HDFS
- Il faut faire des lectures/écritures cohérentes, car on suppose qu'on ne fait pas d'accès distants
- On gère des formats de données
 - On lit des données dans un format et on retourne un KV
 - On donne un KV et on écrit dans un format

```
public interface FormatReader {  
    public KV read();  
}  
public interface FormatWriter {  
    public void write(KV record);  
}
```

Lecture/écriture des données

- Un format de fichier implante l'interface Format
- On gère deux formats (qui implantent Format) :
 - TxtFormat : une classe pour les fichiers texte
 - KVFormat : une classe pour des fichiers KV

```
public interface Format extends FormatReader, FormatWriter, Serializable {  
    public static final int FMT_TXT = 0;  
    public static final int FMT_KV = 1;  
    public void open(String mode);  
    public void close();  
    public long getIndex();  
    public String getFname();  
    public void setFname(String fname);  
}
```

getIndex() permet de savoir où on en est dans le parcours d'un fichier

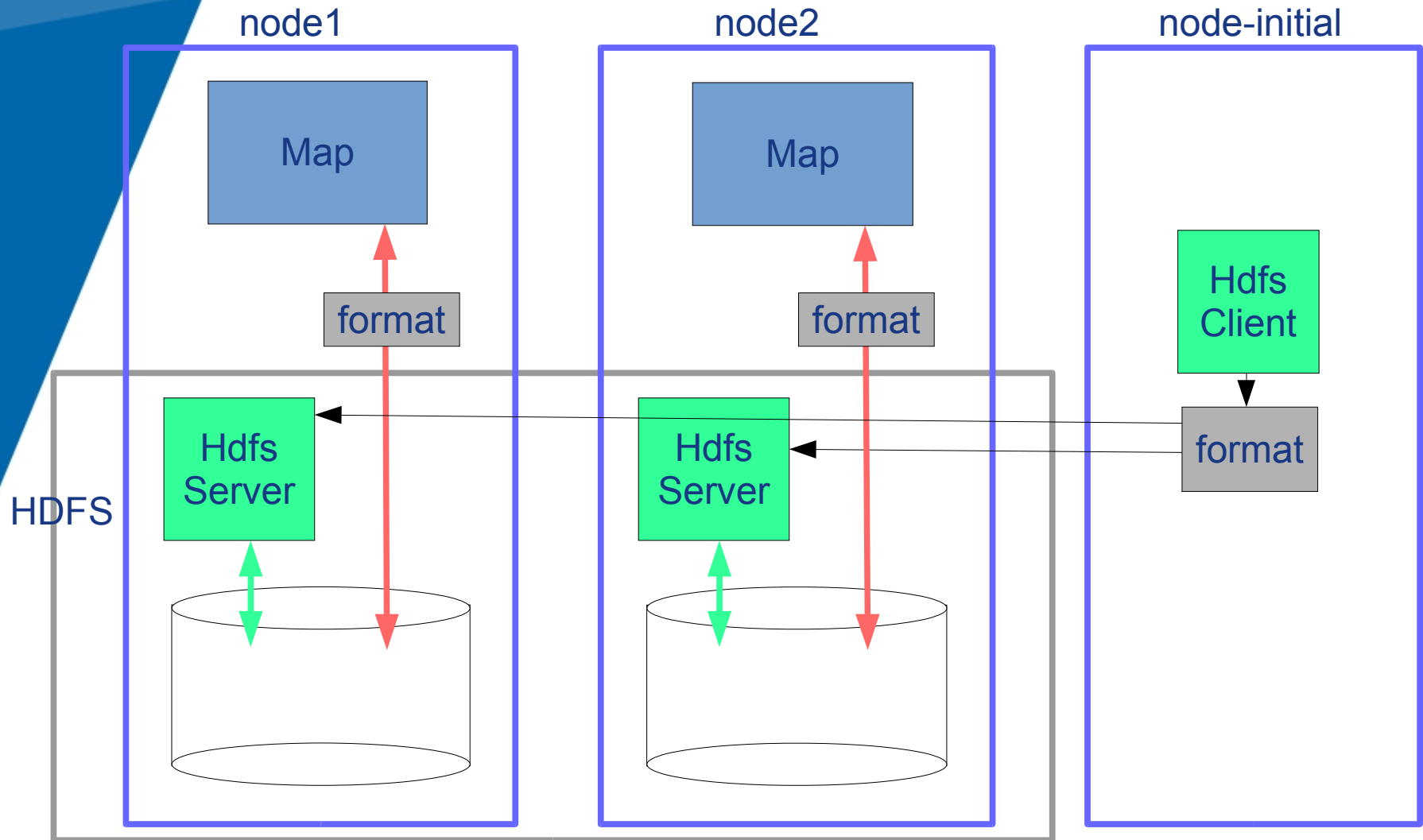
Des key-values

```
public class KV {  
    public static final String SEPARATOR = "<->";  
    public String k;  
    public String v;  
  
    public KV() {}  
  
    public KV(String k, String v) {  
        super();  
        this.k = k;  
        this.v = v;  
    }  
  
    public String toString() {  
        return "KV [k=" + k + ", v=" + v + "];"  
    }  
}
```

HDFS

- Permet de gérer des fichiers fragmentés sur les nœuds
 - Quand on copie un fichier du FS local dans le FS HDFS, le fichier est coupé en fragments qui sont copiés sur les nœuds.
 - Quand on copie un fichier du FS HDFS dans le FS local, les fragments sont rassemblés pour obtenir le fichier complet sur le FS local.
- Les fragments sont copiés sur le FS local du nœud avec un nom particulier
- Les fragments sont de taille variable et non répliqués

Architecture HDFS



HDFS

- Utilisation externe (depuis un shell)

```
public class HdfsClient {  
    public static void HdfsDelete(String fname) {...}  
    public static void HdfsWrite(int fmt, String fname) {...}  
    public static void HdfsRead(String fname) {...}  
    public static void main(String[] args) {  
        // java HdfsClient <read|write> <txt|kv> <file>  
    }  
}
```

- Utilisation interne
 - À distance depuis HdfsClient
 - Utilisation d'un daemon appelé HdfsServer
 - Localement depuis les map
 - Lecture/écriture directe sur le FS local
 - Stockage des meta-données dans un fichier sur le node-initial

Hadoop

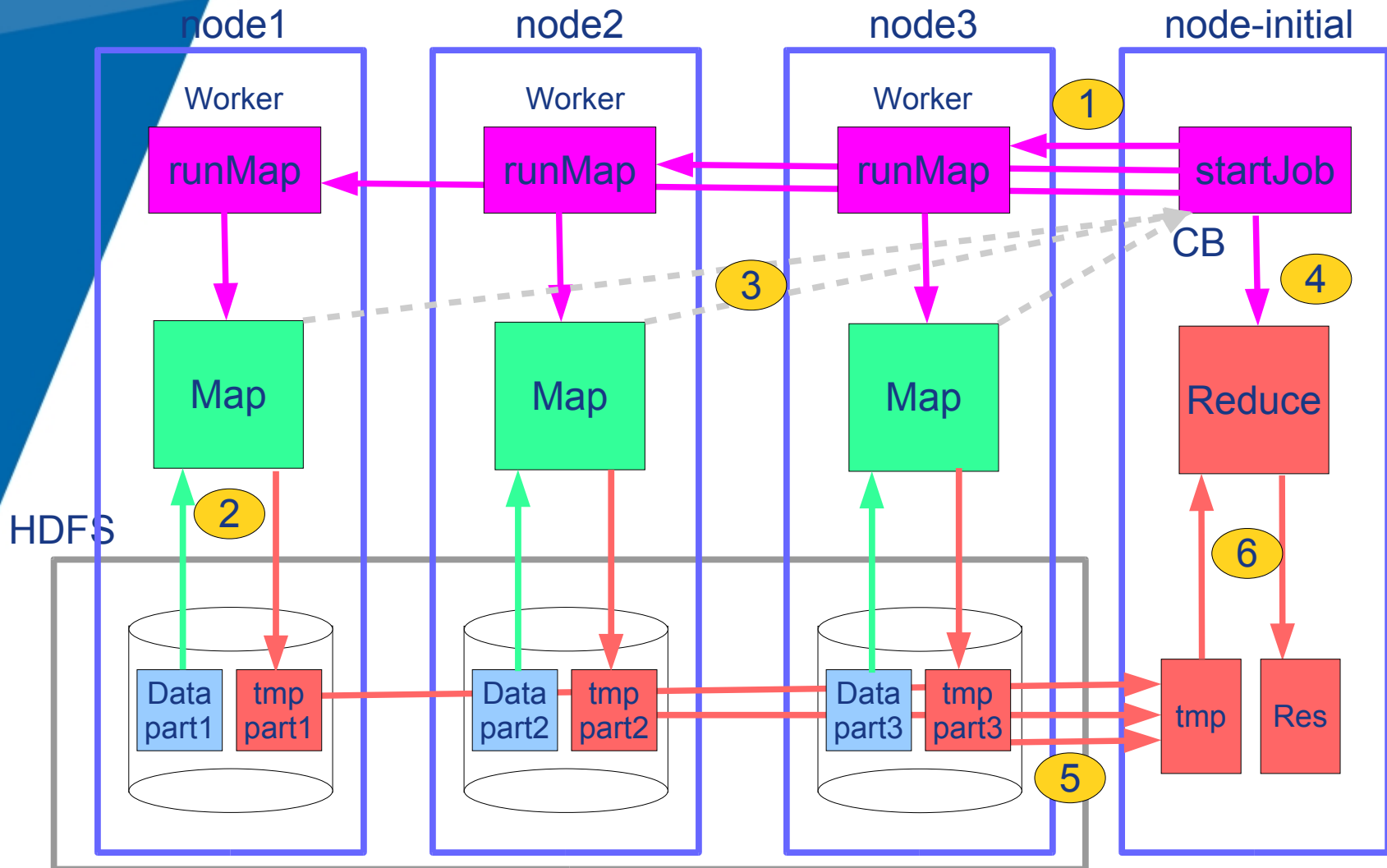
- Lancement d'un Job

```
public class JobLauncher {  
    public static void startJob (MapReduce mr, int format, String fname) {  
        ...  
    }  
}
```

- Interface d'un démon appelé Worker

```
public interface Worker extends Remote {  
    public void runMap (Map m, Format reader, Format writer, Callback cb)  
        throws RemoteException;  
}
```

Hadoop



- 1 • startJob lance les Map en appelant runMap sur les Worker (en leur donnant un reader, writer, callback)
- 2 • Les Map calculent en lisant localement un fragment et génèrent un fragment de résultat (fragment de tmp)
- 3 • Les Map appellent le CallBack pour dire qu'ils ont fini
- 4 • startJob lance le Reduce
- 5 • Le Reduce copie dans son FS local le fichier tmp résultat des Map (composé de fragments), grâce à HdfsClient qui contacte les HdfsServer
- 6 • Le Reduce calcule en lisant localement dans tmp et génère le fichier résultat final

Modèle de programmation

```
public interface Map extends Serializable {  
    public void map(FormatReader reader, FormatWriter writer);  
}  
public interface Reduce extends Serializable {  
    public void reduce(FormatReader reader, FormatWriter writer);  
}  
public interface MapReduce extends Map, Reduce {  
}
```

- Les lectures/écritures se font toujours sur des fichiers ou fragments locaux