MapReduce编程

培训目标

1. 理解MapReduce编程
2. 掌握Mapreduce编程技巧

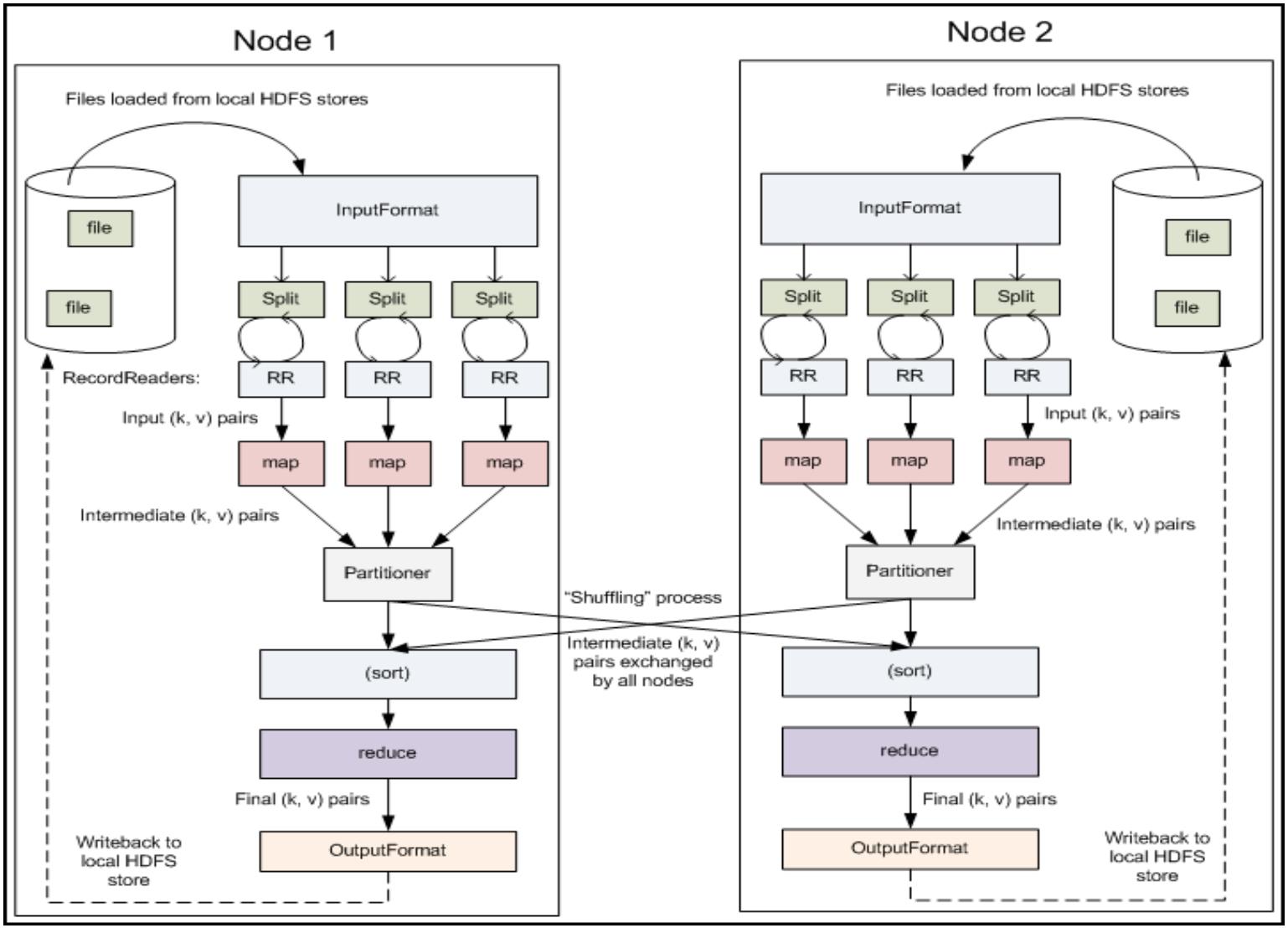
培训目录

Hello World

API介绍和编程详解

案例与高级编程

MapReduce剖析图



Hello World

WordCount v2.0

- Hadoop原生实现“单词统计”实现类

org.apache.hadoop.examples.WordCount

- 执行命令

hadoop jar hadoop-examples-1.1.2.jar
wordcount <in> <out>

- 构造数据

文件 file1：

Hello World Bye World

文件 file2：

Hello Hadoop Goodbye Hadoop

WordCount v2.0

- 加载数据

hadoop fs -put file1 /input/wordcount/
hadoop fs -put file2 /input/wordcount/

- 执行

hadoop jar hadoop-examples-1.1.2.jar
wordcount /input/wordcount/ /output

- 结果查看

hadoop fs -ls /output
hadoop fs -cat /output/part-r-00000

WordCount v2.0

- 内部如何实现？
- 如何编写MR程序？
- 代码什么流程？

- 下面的wordcount v2代码讲解内容只做参考，详细参见课堂实际代码示例讲解

WordCount v2.0

```
public class WordCount  {
      public static void main(String[] args) throws Exception{
          Configuration conf = new Configuration();
          String[] otherArgs = new GenericOptionsParser(conf,
        args).getRemainingArgs();

          if (otherArgs.length != 2) {
            System.err.println("Usage: wordcount <in> <out>");
            System.exit(2);
          }
          Job job = new Job(conf, "word count");
          job.setJarByClass(WordCount.class);
          job.setMapperClass(TokenizerMapper.class);
          job.setCombinerClass(IntSumReducer.class);
          job.setReducerClass(IntSumReducer.class);
          job.setOutputKeyClass(Text.class);
          job.setOutputValueClass(IntWritable.class);
          FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
          FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
          System.exit(job.waitForCompletion(true) ? 0 : 1);
      }
}
```

Mapper函数类

```
class TokenizerMapper
        extends Mapper<Object, Text, Text, IntWritable>
{
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context)
        throws IOException, InterruptedException
    {
        StringTokenizer itr = new StringTokenizer(value.toString());

        while (itr.hasMoreTokens())
        {
            word.set(itr.nextToken());
            context.write(word, one);
        }
    }
}
```

Reducer函数类

```
class IntSumReducer
    extends Reducer<Text,IntWritable,Text,IntWritable>
{
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
    Context context)
            throws IOException, InterruptedException
    {
            int sum = 0;
            for (IntWritable val : values) {
                    sum += val.get();
            }

            result.set(sum);
            context.write(key, result);
    }
}
```

Java代码简要分析

- Job job = new Job(conf, "word count");
  - 原型：Job(Configuration conf, String jobName) ；

- job.setJarByClass(WordCount.class);
  - Set the Jar by finding where a given class came from

- job.setMapperClass(TokenizerMapper.class);
  - Set the Mapper for the job

- job.setCombinerClass(IntSumReducer.class);
  - Set the combiner class for the job

Java代码简要分析

- job.setReducerClass(IntSumReducer.class);
  - Set the Reducer for the job.

- job.setOutputKeyClass(Text.class);
  - Set the key class for the job output data.

- job.setOutputValueClass(IntWritable.class);
  - Set the value class for job outputs.

- FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
  - 原型：static void addInputPath(JobConf conf, Path path);
  - Add a Path to the list of inputs for the map-reduce job.

Java代码简要分析

- FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
  - 原型：static void setOutputPath (JobConf conf, Path outputDir)
  - Set the Path of the output directory for the map-reduce job.

- System.exit(job.waitForCompletion(true) ? 0 : 1);
  - Submit the job to the cluster and wait for it to finish.

Java代码简要分析

1. TokenizerMapper类的定义

```
class TokenizerMapper
        extends Mapper<Object, Text, Text,
IntWritable> { … }
```

- 所有的mapper类都必须实现接口Mapper，该接口有4个类型参数，分别是：
- Object：Input Key Type
- Text: Input Value Type
- Text: Output Key Type
- IntWritable: Output Value Type

Java代码简要分析

- 因为所有的mapper类都必须实现接口Mapper，即必须实现Mapper中的map函数

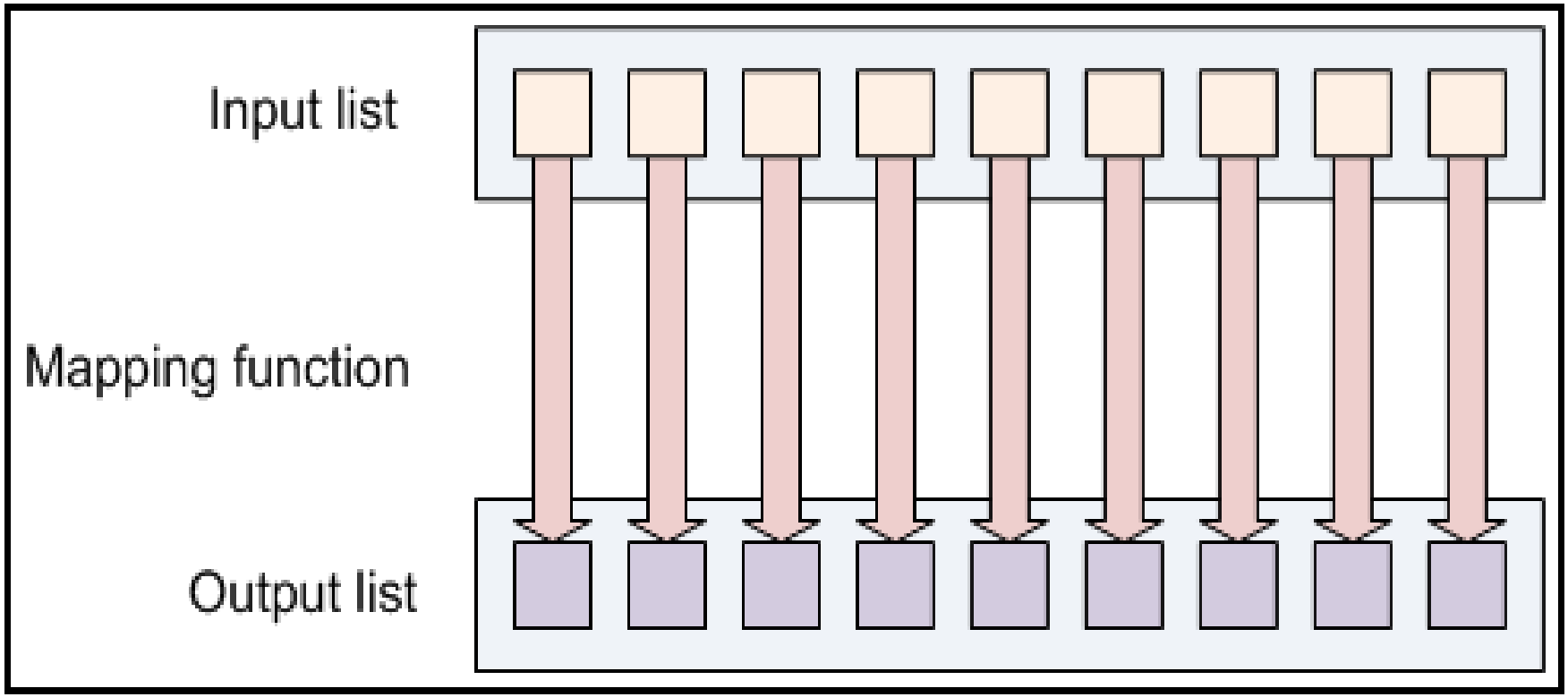
```
public void map(Object key, Text value, Context context)
        throws IOException, InterruptedException
    {
        StringTokenizer itr = new
    StringTokenizer(value.toString());

        while (itr.hasMoreTokens()) {
                word.set(itr.nextToken());
                context.write(word, one);
        }
    }
```

API介绍

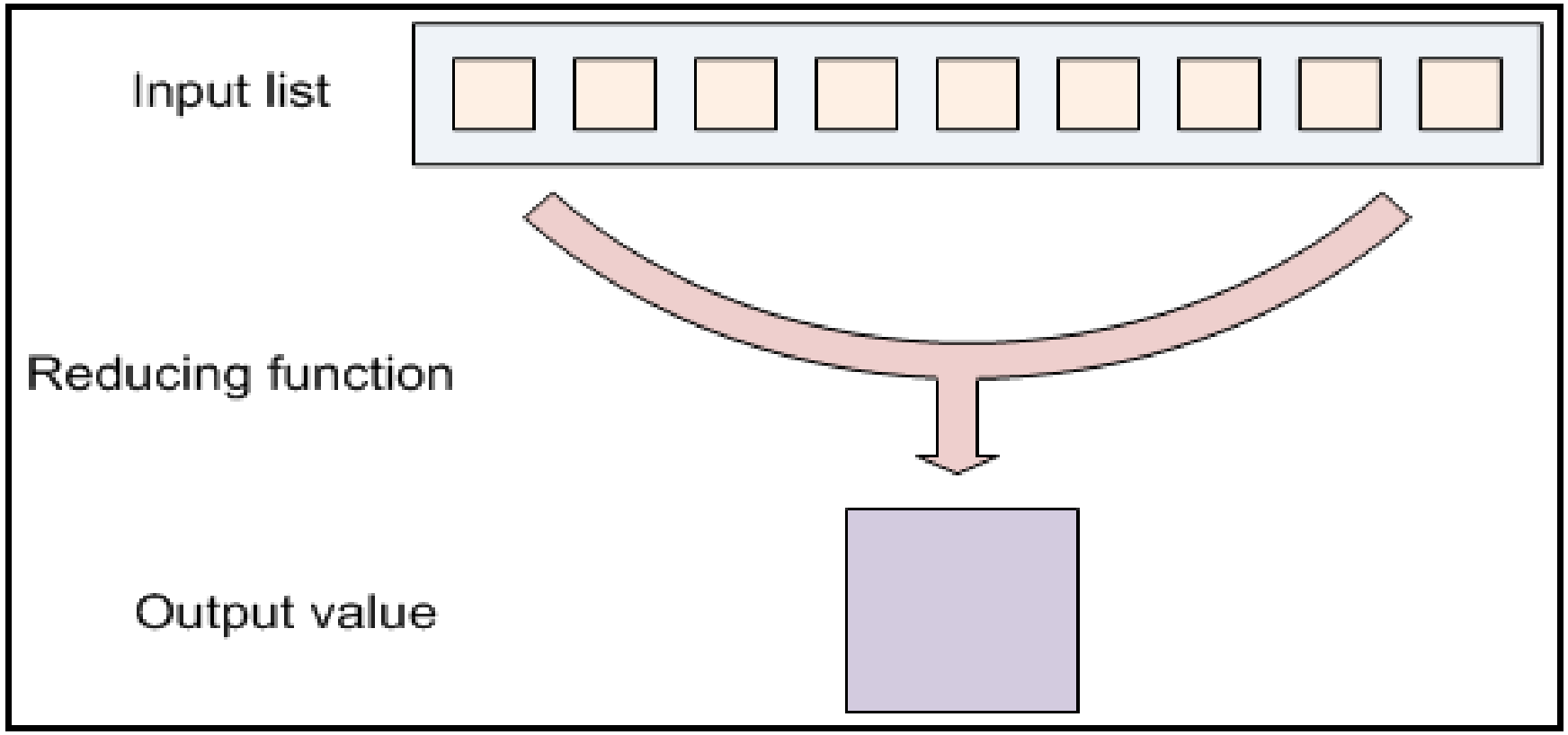
Map过程

- Map过程通过在输入列表中的每一项执行函数，生成一系列的输出列表。



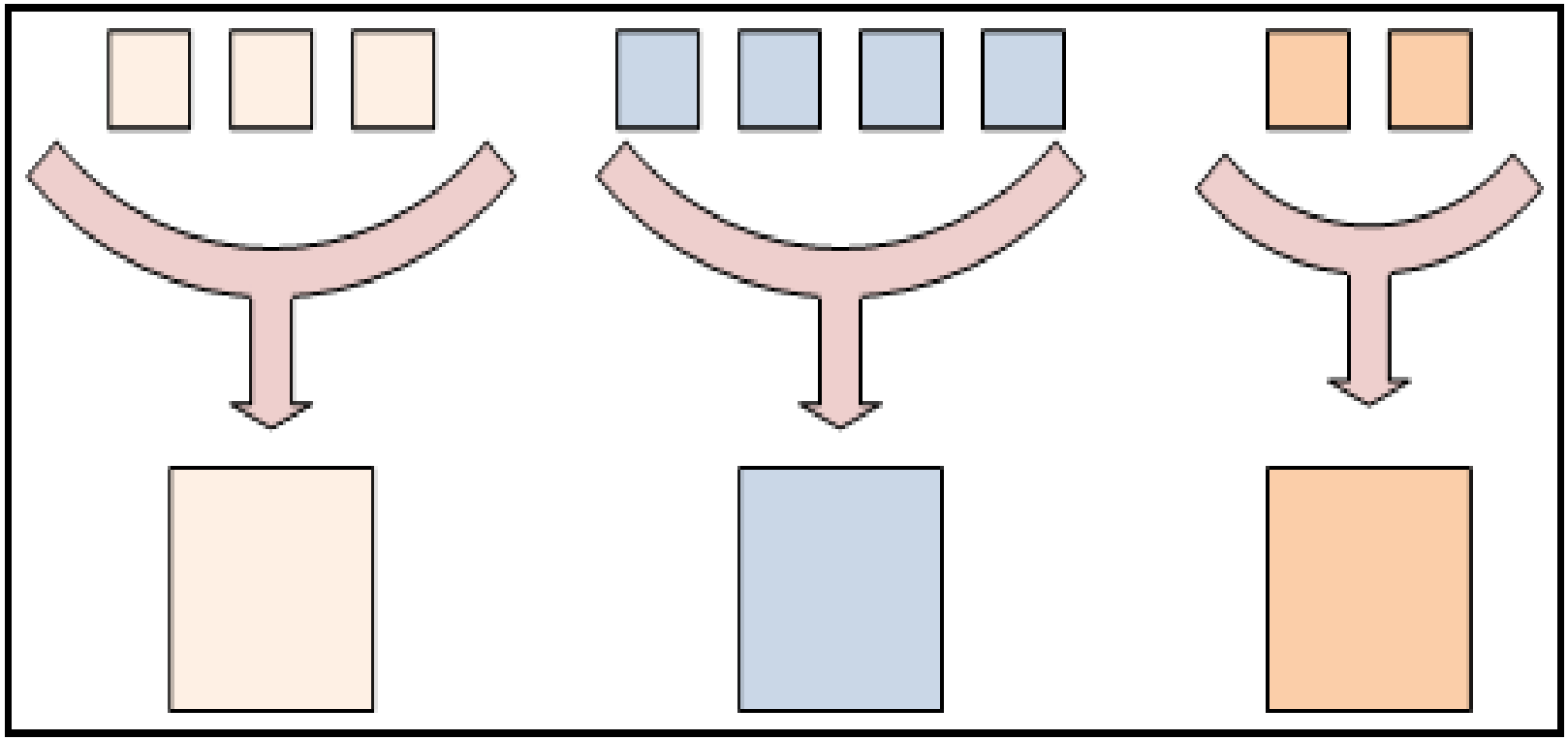
Reduce过程

- Reduce过程在一个输入的列表进行扫描工作，随后生成一个聚集值，作为最后的输出



MapReduce的Reduce过程

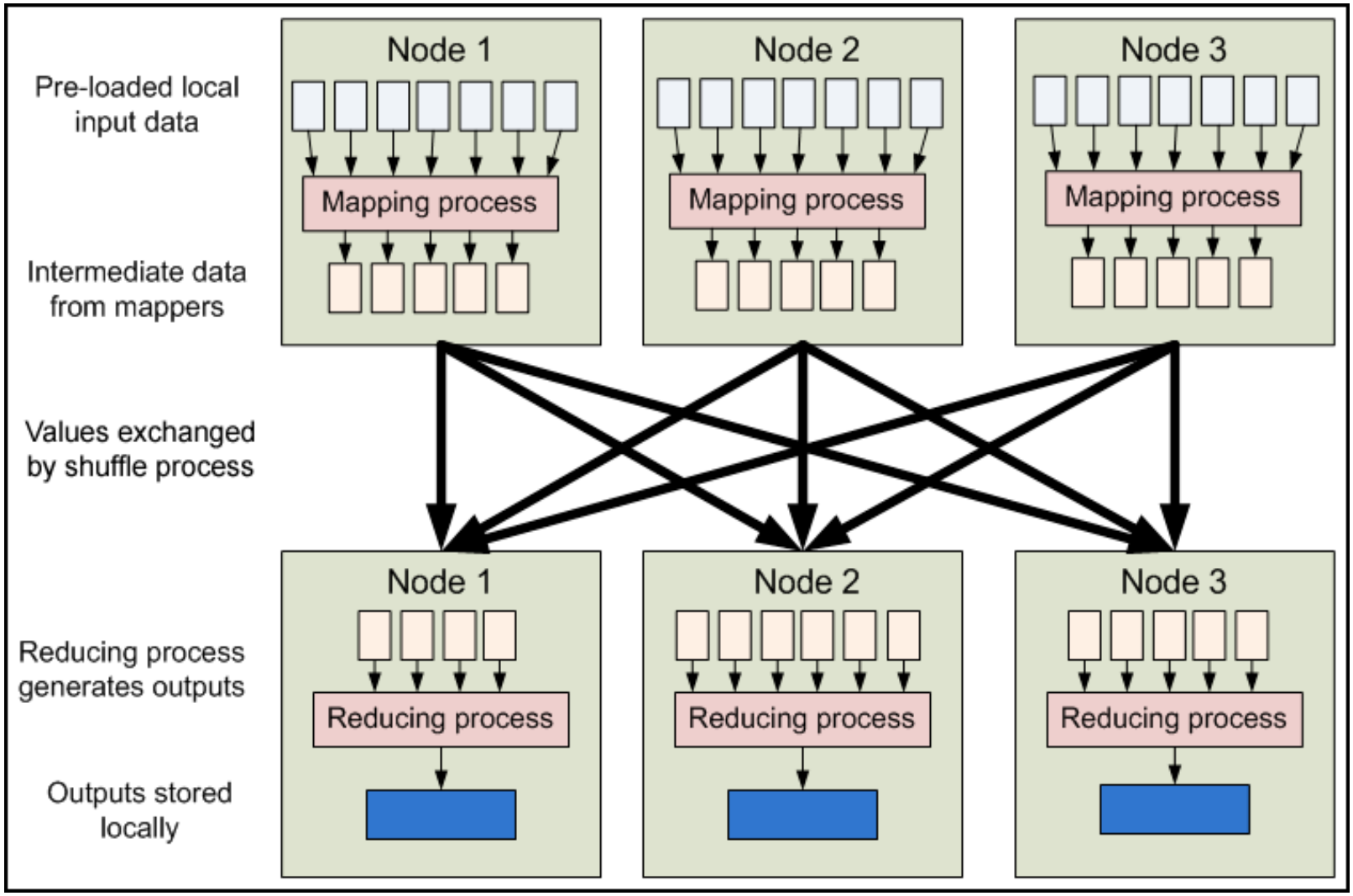
- 所有不同的颜色代表不同的键值（keys）。所有相同键值的列表被输入到同一个Reduce任务中



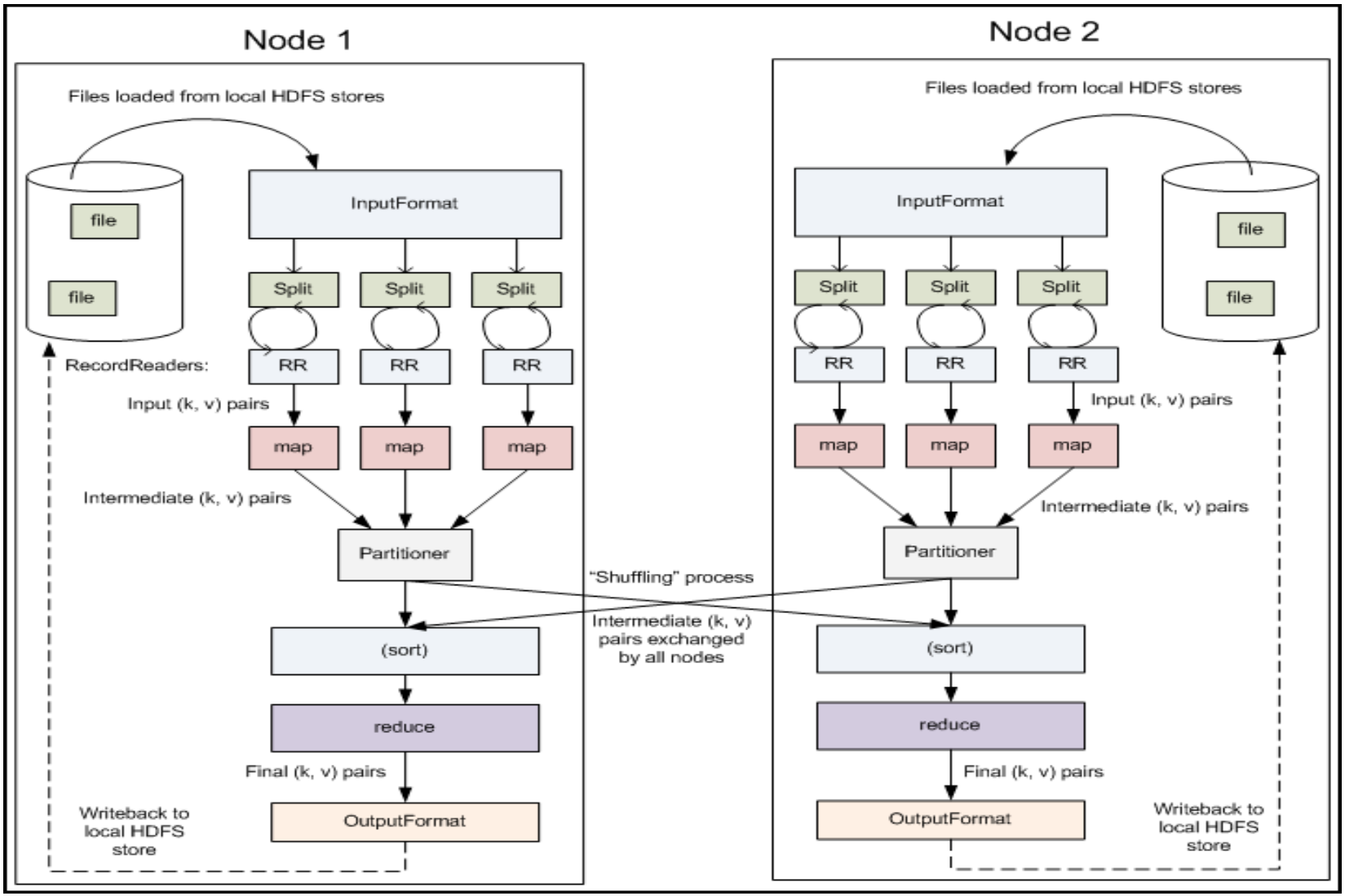
Components

- Mapper
  - public static class TokenizerMapper
  - extends Mapper<Object, Text, Text, IntWritable>
- Reducer
  - public static class IntSumReducer
  - extends Reducer<Text,IntWritable,Text,IntWritable>
- Driver
  - import org.apache.hadoop.mapreduce.Job;

MapReduce的数据流程



MapReduce的执行过程



Input Files

- 输入文件一般 保存在HDFS中
- 文件的类型不固定，可能是文本的，也有可能是其它形式的文件
- 文件经常很大，甚至有几十个GB

InputFormat

- 定义了这些文件如何分割，读取
- InputFile提供了以下一些功能
  - 选择文件或者其它对象，用来作为输入
  - 定义InputSplits，将一个文件分开成为任务
  - 为RecordReader提供一个工厂，用来读取这个文件

InputFormat

- 有一个抽象的类FileInputFormat，所有的输入格式类都从这个类继承这个类的功能以及特性。当启动一个Hadoop任务的时候，一个输入文件所在的目录被输入到FileInputFormat对象中。FileInputFormat从这个目录中读取所有文件。然后FileInputFormat将这些文件分割为一个或者多个InputSplits。
- 通过在Job对象上设置JobConf.setInputFormatClass设置文件输入的格式

预定义的文件输入格式

InputFormat:	Description:	Key:	Value:
TextInputFormat	Default format; reads lines of text files	The byte offset of the line	The line contents
KeyValueTextInputFormat	Parses lines into key, val pairs	Everything up to the first tab character	The remainder of the line
SequenceFileInputFormat	A Hadoop-specific high-performance binary format	user-defined	user-defined

各种InputFormat

- TextInputFormat，默认的格式，每一行是一个单独的记录，并且作为value，文件的偏移值作为key
- KeyValueTextInputFormat，这个格式每一行也是一个单独的记录，但是Key和Value用Tab隔开，是默认的OutputFormat，可以作为中间结果，作为下一步MapReduce的输入。
- SequenceFileInputFormat
  - 基于块进行压缩的格式
  - 对于几种类型数据的序列化和反序列化操作
  - 用来将数据快速读取到Mapper类中

InputSplits

- InputSplit定义了输入到单个Map任务的输入数据
- 一个MapReduce程序被统称为一个Job，可能有上百个任务构成
- InputSplit将文件分为64MB的大小
  - hadoop-site.xml中的mapred.min.split.size参数控制这个大小
- mapred.tasktracker.map.taks.maximum用来控制某一个节点上所有map任务的最大数目

RecordReader

- InputSplit定义了一项工作的大小，但是没有定义如何读取数据
- RecordReader实际上定义了如何从数据上转化为一个(key,value)对，从而输出到Mapper类中
- TextInputFormat提供了LineRecordReader

Mapper

- 每一个Mapper类的实例生成了一个Java进程（在某一个InputSplit上执行）
- 有两个额外的参数OutputCollector以及Reporter，前者用来收集中间结果，后者用来获得环境参数以及设置当前执行的状态
- 现在用Mapper.Context提供给每一个Mapper函数，用来提供上面两个对象的功能

Partition&Shuffle

- 在Map工作完成之后，每一个 Map函数会将结果传到对应的Reducer所在的节点，此时，用户可以提供一个Partitioner类，用来决定一个给定的(key,value)对传输的具体位置

Sort

- 传输到每一个节点上的所有的Reduce函数接收到得Key,value对会被Hadoop自动排序（即Map生成的结果传送到某一个节点的时候，会被自动排序）

Reduce

- 做用户定义的Reduce操作
- 接收到一个OutputCollector的类作为输出
- 最新的编程接口是Reducer.Context

OutputFormat

- 写入到HDFS的所有OutputFormat都继承自FileOutputFormat
- 每一个Reducer都写一个文件到一个共同的输出目录，文件名是part-nnnnn，其中nnnnn是与每一个reducer相关的一个号（partition id）
- FileOutputFormat.setOutputPath()
- JobConf.setOutputFormat()

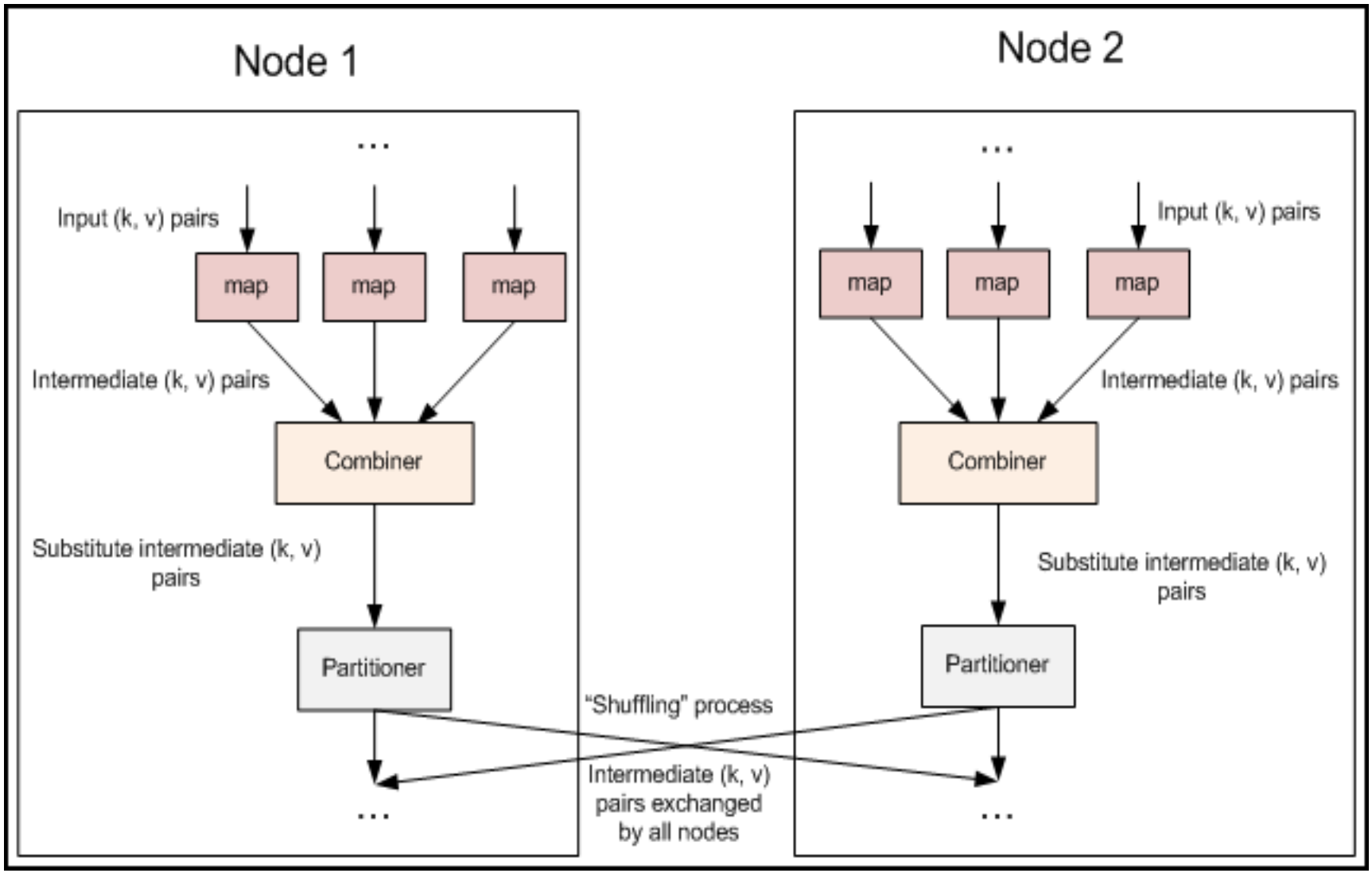
Output Format

OutputFormat:	Description
TextOutputFormat	Default; writes lines in "key \t value" form
SequenceFileOutputFormat	Writes binary files suitable for reading into subsequent MapReduce jobs
NullOutputFormat	Disregards its inputs

RecordWriter

- 用来指导如何输出一个记录到文件中

Combiner


Node 1
Node 2
Input (k, v) pairs
map
map
map
Intermediate (k, v) pairs
Combiner
Substitute intermediate (k, v) pairs
Partitioner
"Shuffling" process
Intermediate (k, v) pairs exchanged by all nodes

Combiner

- conf.setCombinerClass(Reduce.class);
- 是在本地执行的一个Reducer，满足一定的条件才能够执行。

容错

- 由Hadoop系统自己解决
- 主要方法是将失败的任务进行再次执行
- TaskTracker会把状态信息汇报给JobTracker，最终由JobTracker决定重新执行哪一个任务
- 为了加快执行的速度，Hadoop也会自动重复执行同一个任务，以最先执行成功的为准（投机执行）
- mapred.map.tasks.speculative.execution
- mapred.reduce.tasks.speculative.execution

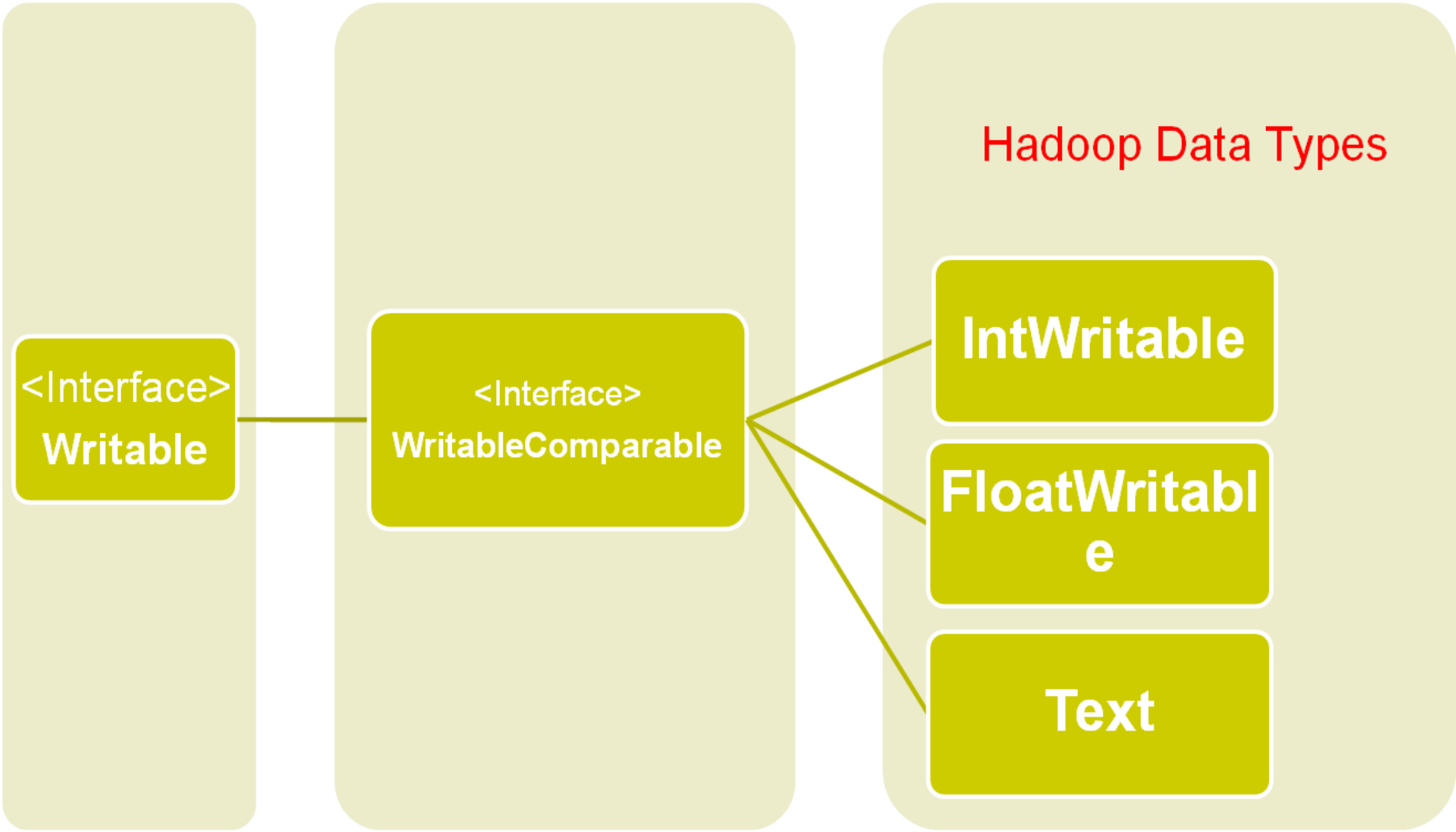
Chaining Jobs

- Map1 -> Reduce1 -> Map2 -> Reduce2 -> Map3...
- 生成多个JobClient.runJob()，等待上一个任务结束，并且开始下一个任务
- org.apache.hadoop.mapred.jobcontrol.Job 可以做一些任务控制
  - x.addDependingJob(y)，x只有在y执行完了之后才能够开始执行

程序的调试

- **日志文件**
  - 分门别类存放在hadoop-version/logs目录下面，hadoop-*username*-*service*-*hostname*.log
  - 对于用户程序来说，TaskTracker的log可能是最重要的
  - 每一个计算节点上有日志logs/userlogs日志，也有重要的日志信息
- 在单机上首先执行，看看是否能够正确执行，而后再在多机的集群系统上执行

Data Types hierachy



案例与高级编程

Python编写MapReduce程序

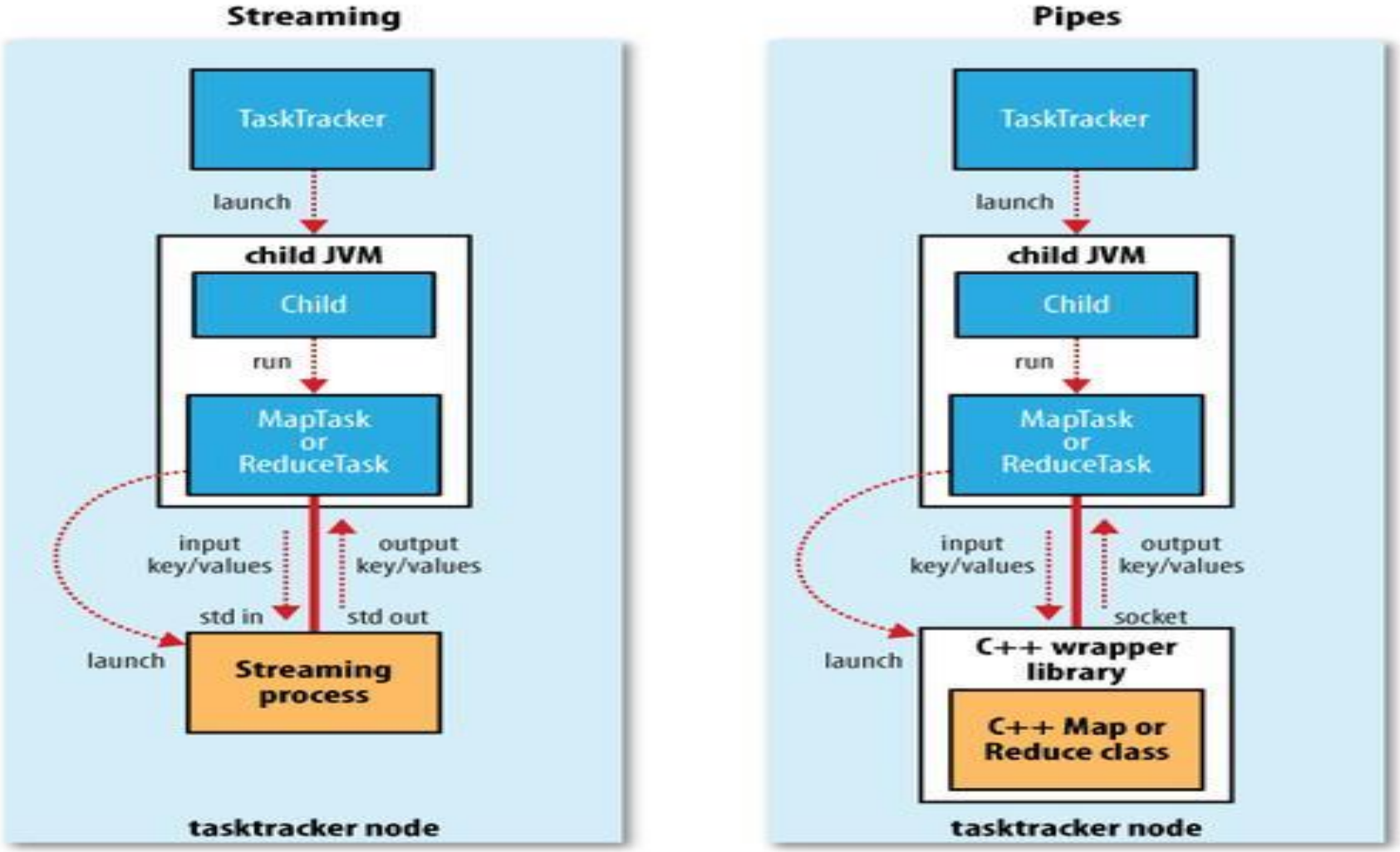
- Hadoop Streaming简介

➢ Hadoop streaming是同Hadoop一起发行的实用工具套件包，利用Hadoop streaming， 可以将任意的可执行文件或者脚本文件作为mapper和reducer来创建并运行MapReduce任务。

- Hadoop Streaming工作原理

➢ Streaming会创建一个map/reduce作业，然后将该作业提交到合适的cluster，并会监视该作业的执行流程直到作业完成。

Hadoop Streaming 原理解析



MapReduce框架与streaming mapper/reducer之间的通信协议概述

- 从mapper视角分析

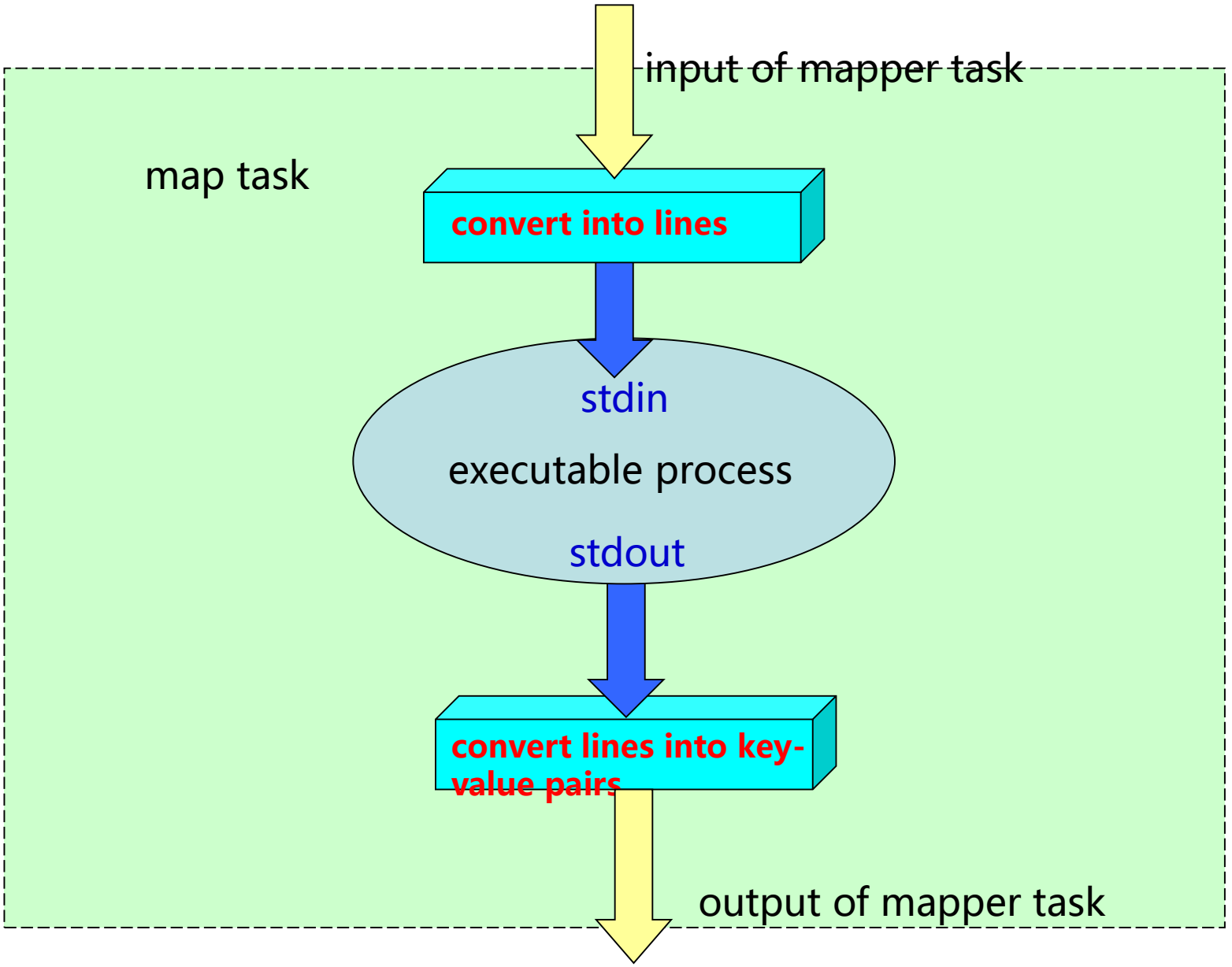
- 1. 当为mappre指定了一个可执行文件(executable)时，每一个map task会将该executable作为一个单独的进程启动。
- 2.　当map task运行时，会将其接收到的输入转换为行，并将其送至进程的standard input，也就是送到了指定的可执行文件的标准输入。

MapReduce框架与streaming mapper/reducer之间的通信协议概述

- 从mapper视角分析

➢ 3. 同时，mapper按行从该进程的standard output收集数据，并将每一行转化为一个key-value pair，作为mapper的输出数据。

➢ 4. 默认情况下，对于步骤（3），每一行中第一个tab字符之前的数据作为key，之后的数据全部作为value。


input of mapper task
map task
convert into lines
stdin
executable process
stdout
convert lines into key-value pairs
output of mapper task

MapReduce框架与streaming mapper/reducer之间的通信协议概述

- 从reducer视角分析

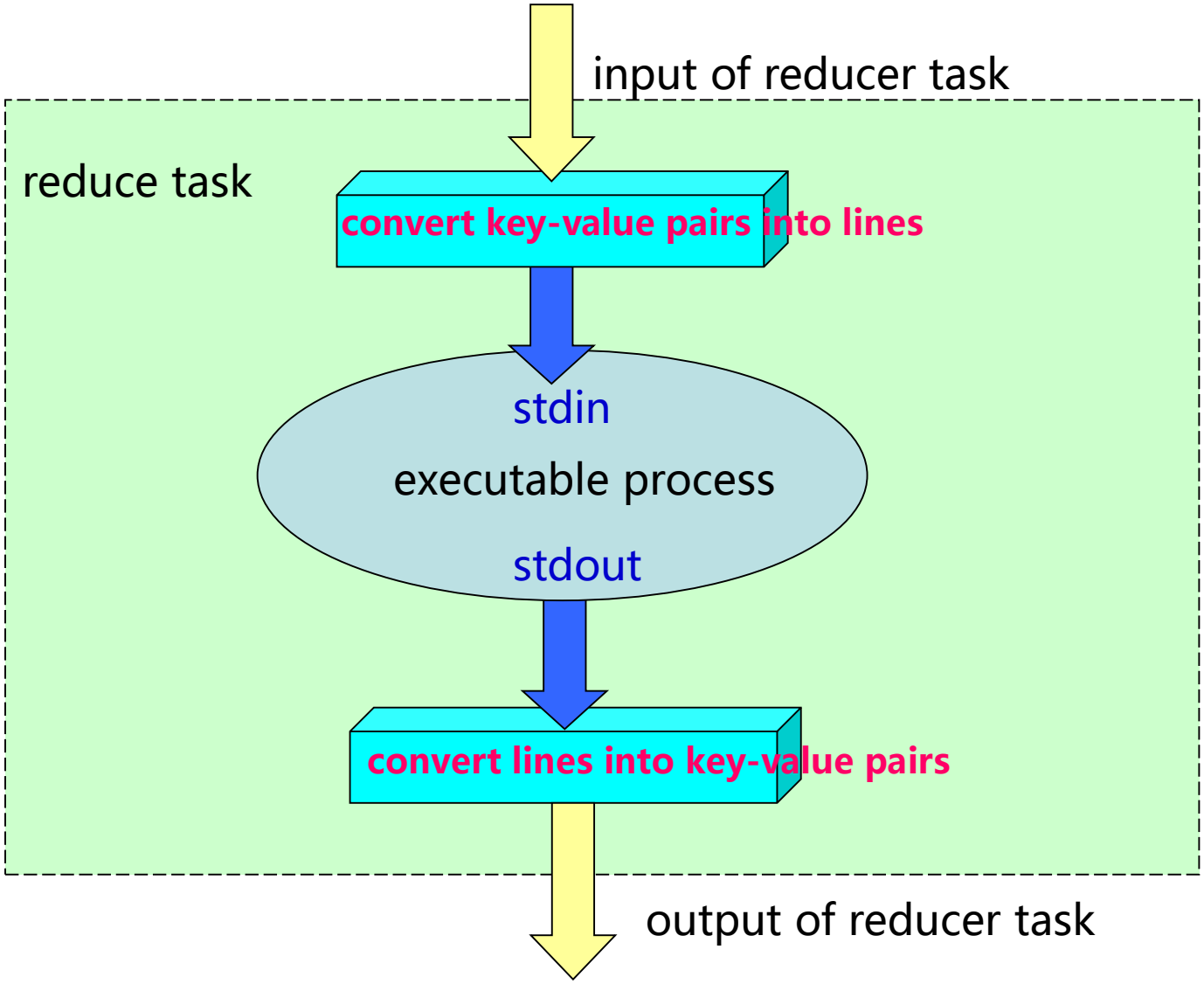
- 1. 当为reducer指定了一个可执行文件(executable)时，每一个reduce task会将该executable作为一个单独的进程启动。
- 2.　当reduce task运行时，会将其接收到的key-value pairs转换为行,并将其送至进程的standard input，也就是送到了指定的可执行文件的标准输入。

MapReduce框架与streaming mapper/reducer之间的通信协议概述

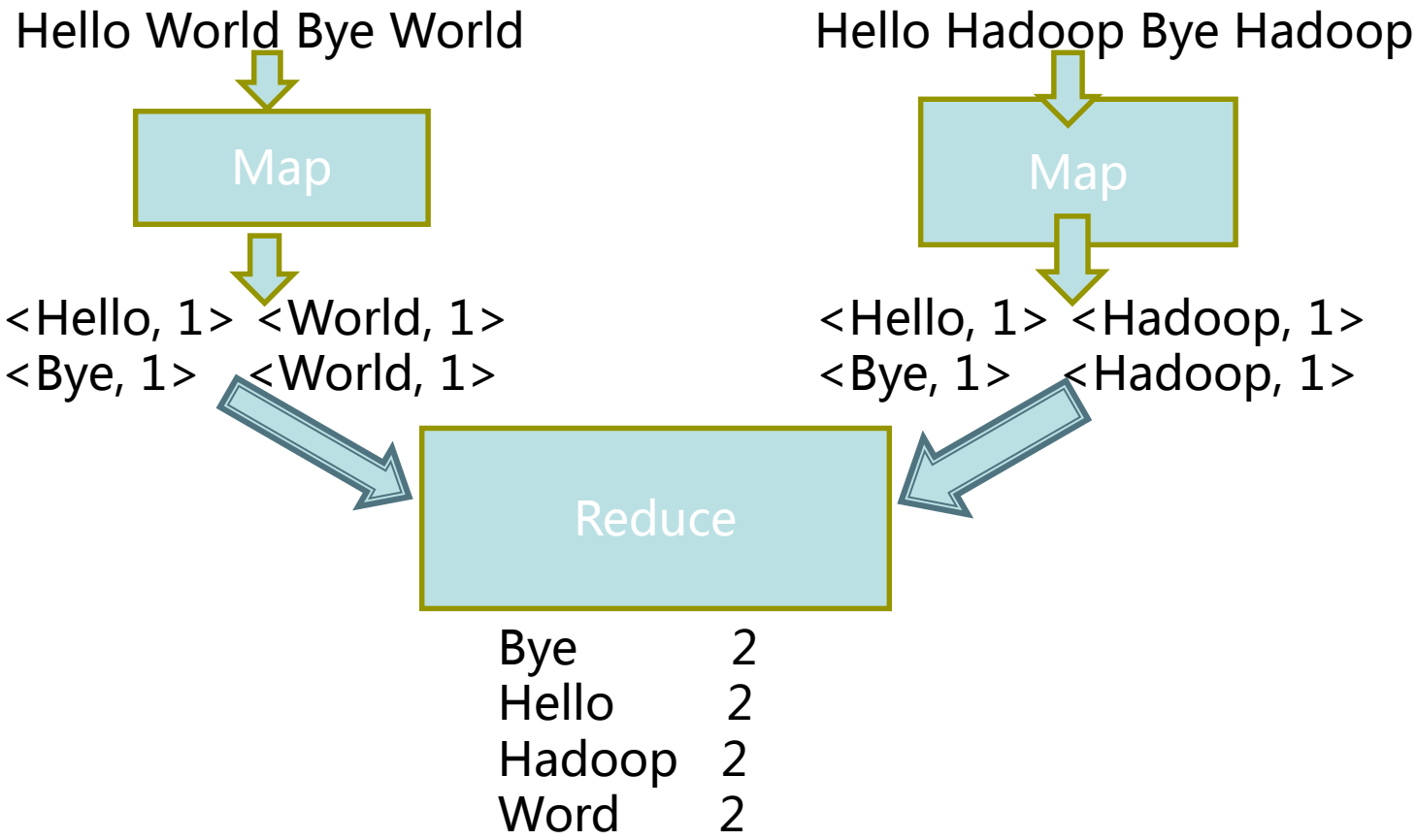
- 从reducer视角分析

➢ 3. 同时，reducer按行从该进程的standard output收集数据，并将每一行转化为一个key-value pair，作为reducer的输出数据。

➢ 4. 默认情况下，对于步骤（3），每一行中第一个tab字符之前的数据作为key，之后的数据全部作为value。


input of reducer task
reduce task
convert key-value pairs into lines
stdin
executable process
stdout
convert lines into key-value pairs
output of reducer task

WordCount 数据流概览



怎样提交并运行一个MapReduce任务

```
bin/hadoop jar contrib/streaming/hadoop-0.20.2-
    streaming.jar \
-input  myInputDirs          \
-output  myOutputDirs     \
-mapper  myMapperExecutable \
-reducer  myReducerExceutable \
-file  myMapperExecutable  myReducerExceutable
```

注意：这里的myInputDirs与myOutputDir均为HDFS文件系统中的目录路径，而不是本机的Linux目录路径，所以要首先将输入文件复制到HDFS中去。注意不要先创建输出目录。

Python的WordCount例子

- Python介绍

- 一种解释型语言，语法简洁，功能强大，有很多成熟的库，且移植性非常好。适合于快速的程序开发。
- 在计算机内部，Python解释器借鉴了Java虚拟机的优点，把代码转换成称为字节码的中间形式，然后再把它翻译成计算机使用的机器语言并运行。

Python的WordCount例子

- Python介绍
  - Goggle, Twitter, Facebook等很多知名的网络服务商均大量使用Python来编写应用程序。
  - 在Google， Python是仅次于Java、C++的使用率排名第三的编程语言。

mapper.py文件

```
1. #!/usr/bin/python
2. import sys

 # 处理标准输入中的每一行
3. for line in sys.stdin:
4. for word in line.split():      # 从一行中提取出每一个单词并输出
5.             sys.stdout.write("%s\t%d\n" % (word, 1))
```

reducer.py文件

```
1. #!/usr/bin/python
2. import sys
3. dict = {}  # 创建一个字典数据结构
   # 每一行输入的格式为<word, count>
4. for line in sys.stdin:
5. (word, count) = line.split()   #将一行输入中的word与count提取出来
6. count = int(count)
7. dict[word] = dict.get(word, 0) + count  #为每一个单词计数
8. for key in dict.keys():
9. sys.stdout.write( "%s\t%d\n"  % (key, dict[key])) #输出每一个
   (word, count)
```

测试该Mapper与Reducer是否正确

- 编写测试数据文件 text.dat
  Hello Hadoop
  Hello MapReduce
  Hello Java and Python
- 键入命令
- cat text.dat | ./mapper.py | ./reducer.py
- 输出结果
  and 1
  Java 1
  Python 1
  MapReduce 1
  Hadoop 1
  Hello 3

验证Map/Reduce

- **提交集群：** sudo -u mapred hadoop jar /usr/share/hadoop/contrib/streaming/hadoop-streaming-1.1.2.jar \

-file mapper.py \

-file reducer.py \

-mapper mapper.py \

-reducer reducer.py \

-input /user/mapred/test.log \

-output /user/mapred/test \

-inputformat org.apache.hadoop.mapred.TextInputFormat \

-outputformat org.apache.hadoop.mapred.TextOutputFormat

Question