Mapreduce原理和YARN

培训目标

1. 掌握mapreduce原理和开发
2. 掌握YARN原理
3. 应用举例讲解

Why MapReduce?

一、将计算移动到数据

- 硬盘存储成本不断降低
- 传输速率提升不大
- 磁头定位时间无提升

Why MapReduce?

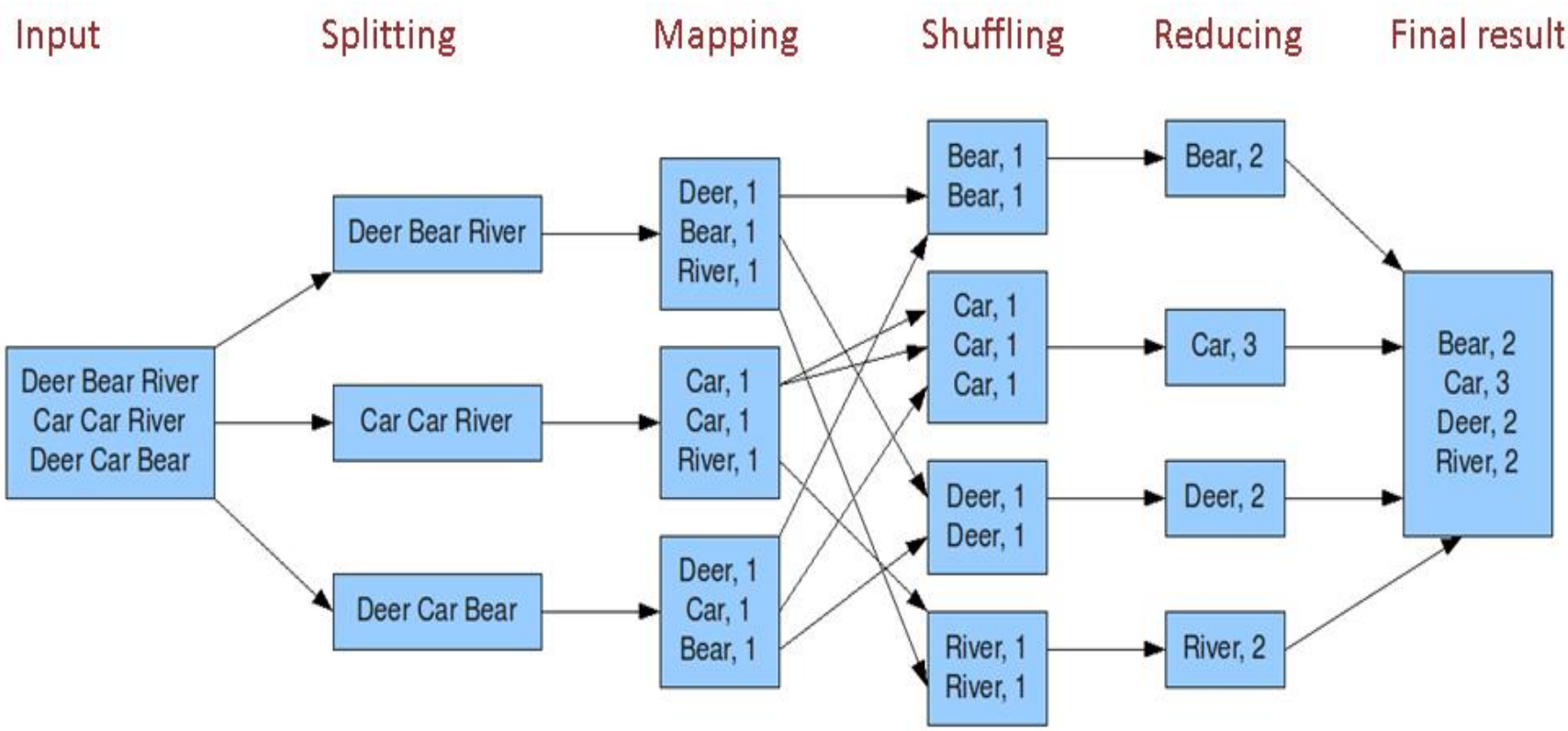
- 并行处理为什么我们不能使用数据库加上更多磁盘来做大规模的批量分析？为什么我们需要MapReduce？
- 这个问题的答案来自于磁盘驱动器的另一个发展趋势：
  - 寻址时间的提高速度远远慢于传输速率的提高速度。
  - 寻址就是将磁头移动到特定位置进行读写操作的工序。
  - 它的特点是磁盘操作有延迟，而传输速率对应于磁盘的带宽。

关系型数据库和MapReduce的比较

	传统关系型数据库	MapReduce
数据大小	GB	PB
访问	交互型和批处理	批处理
更新	多次读写	一次写入多次读取
结构	静态模式	动态模式
集成度	高	低
伸缩性	非线性	线性

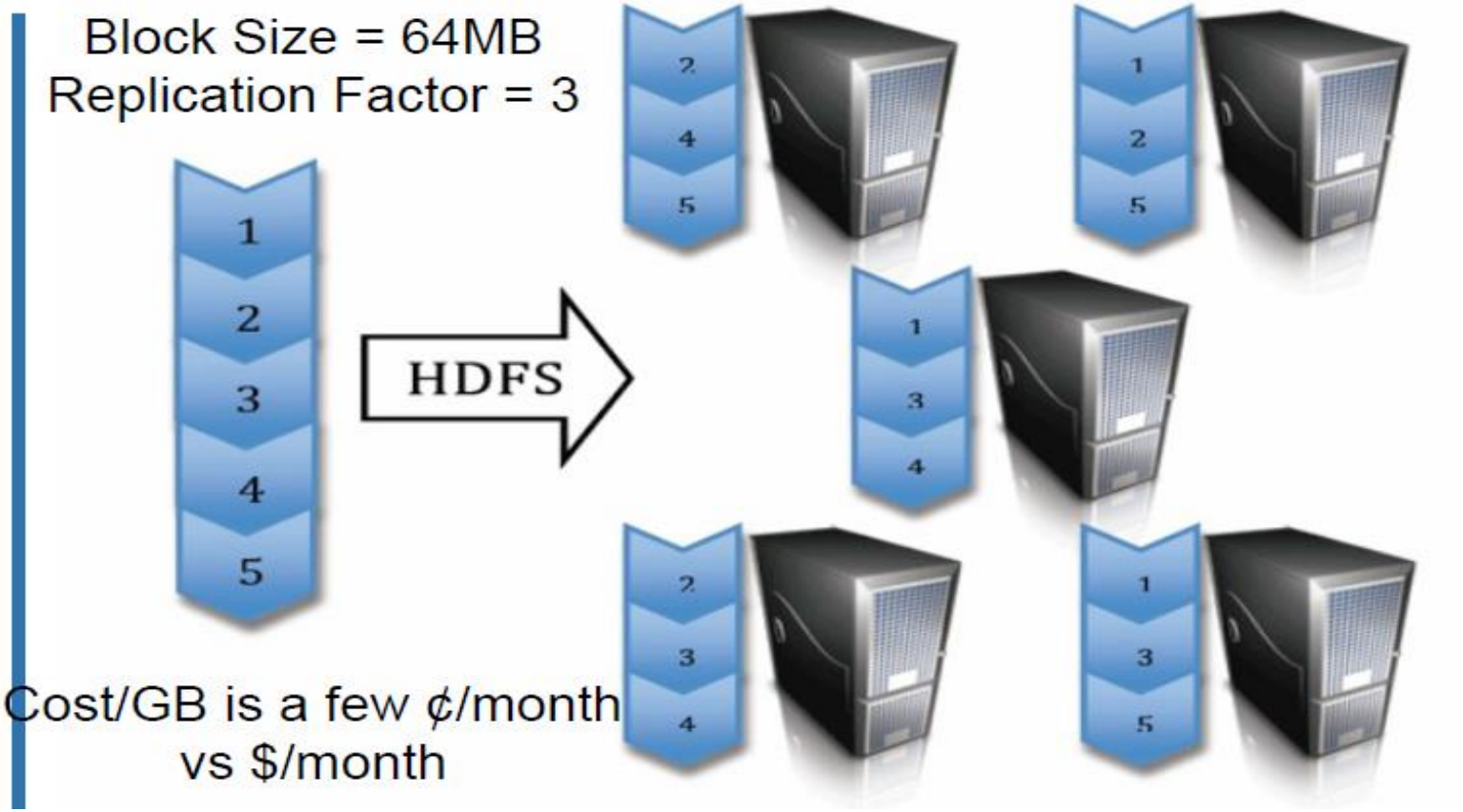
Why MapReduce?

二、Divide and Conquer(分而治之)



分而治之

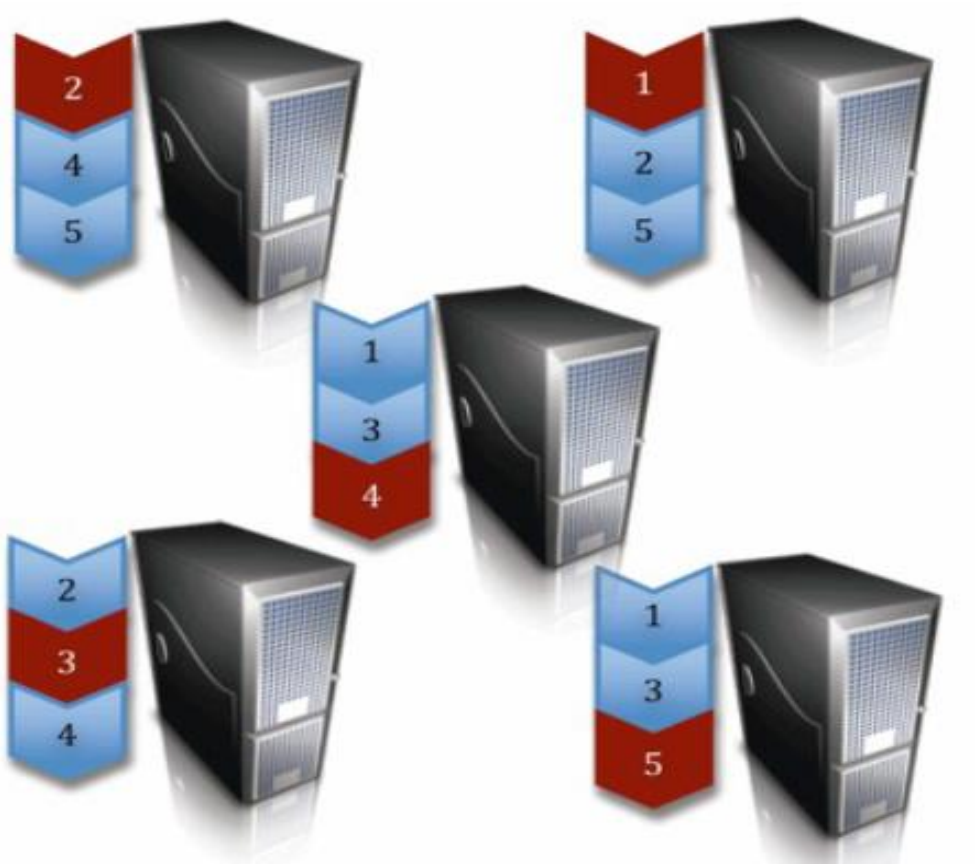
HDFS: Hadoop Distributed File System



分而治之

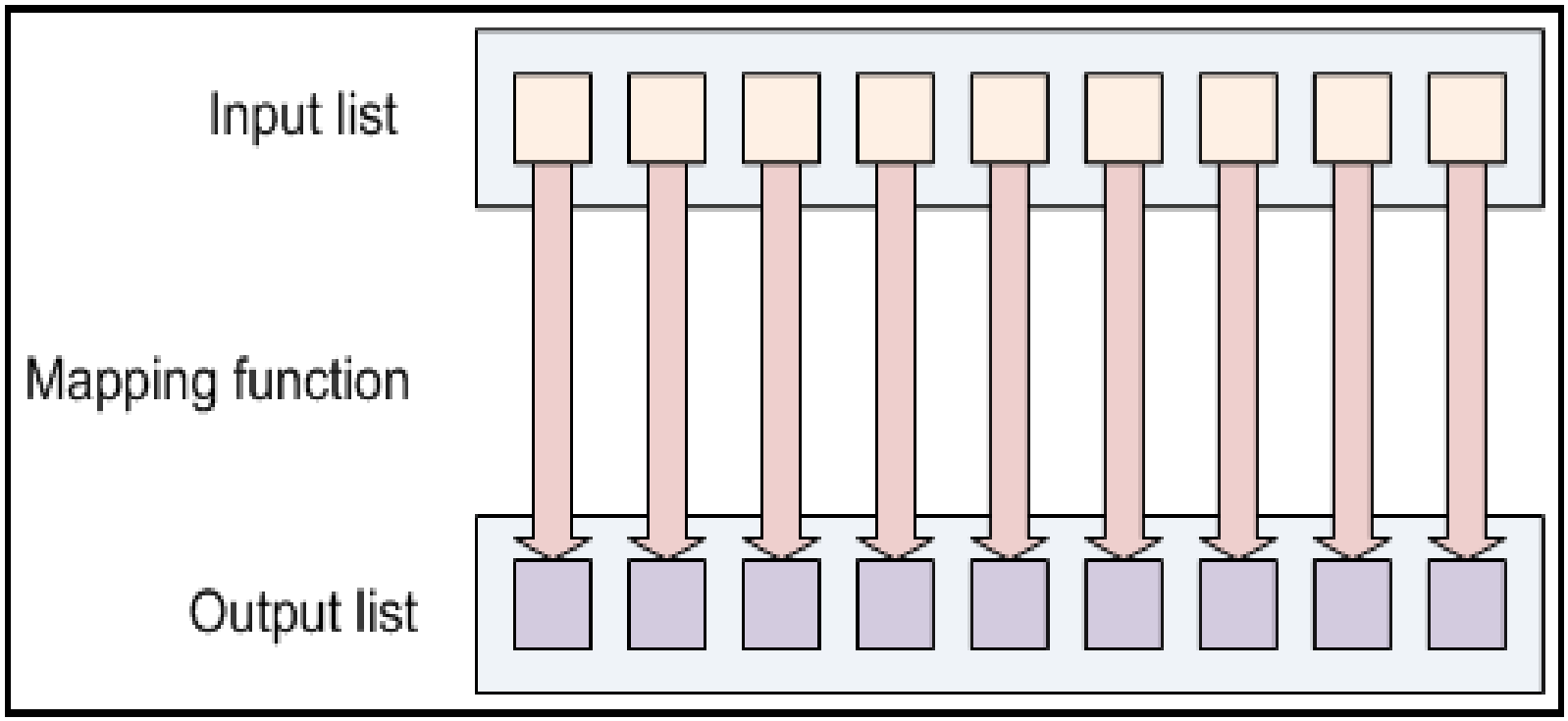
MapReduce:Distributed Processing

Hadoop利用HDFS的数据分布策略把工作分发到多个节点完成。这使得数据分析并行运行，消除了单机存储系统所造成的瓶颈.



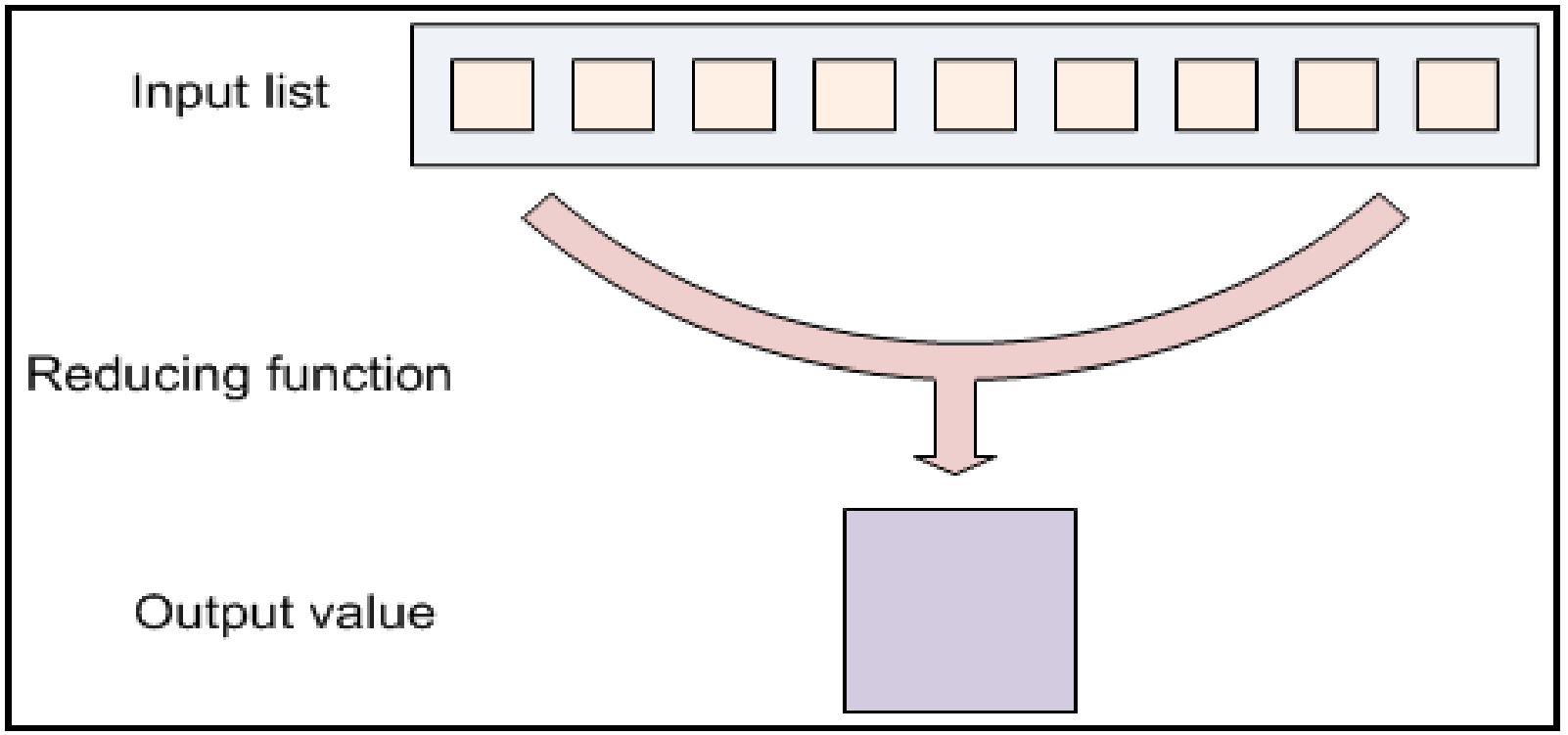
Map过程

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。



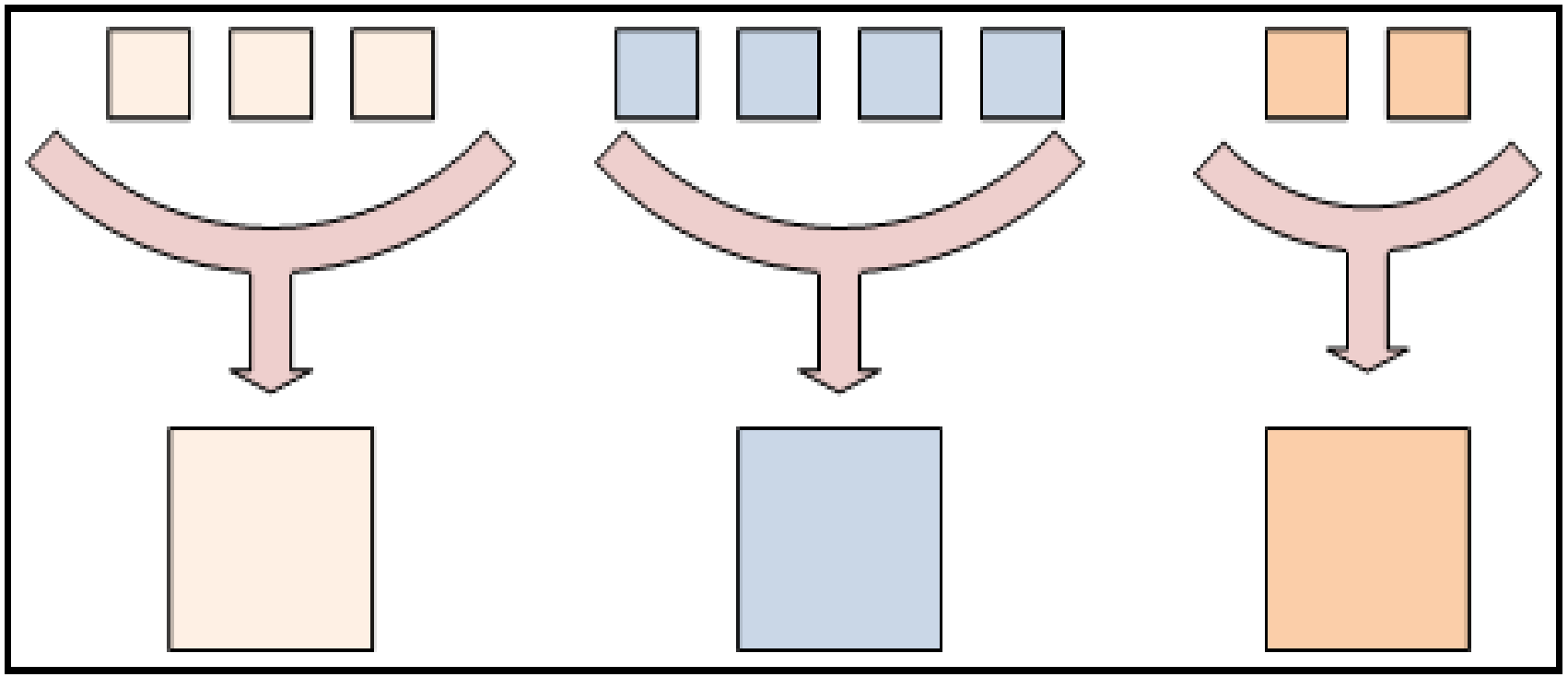
Reduce过程

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。

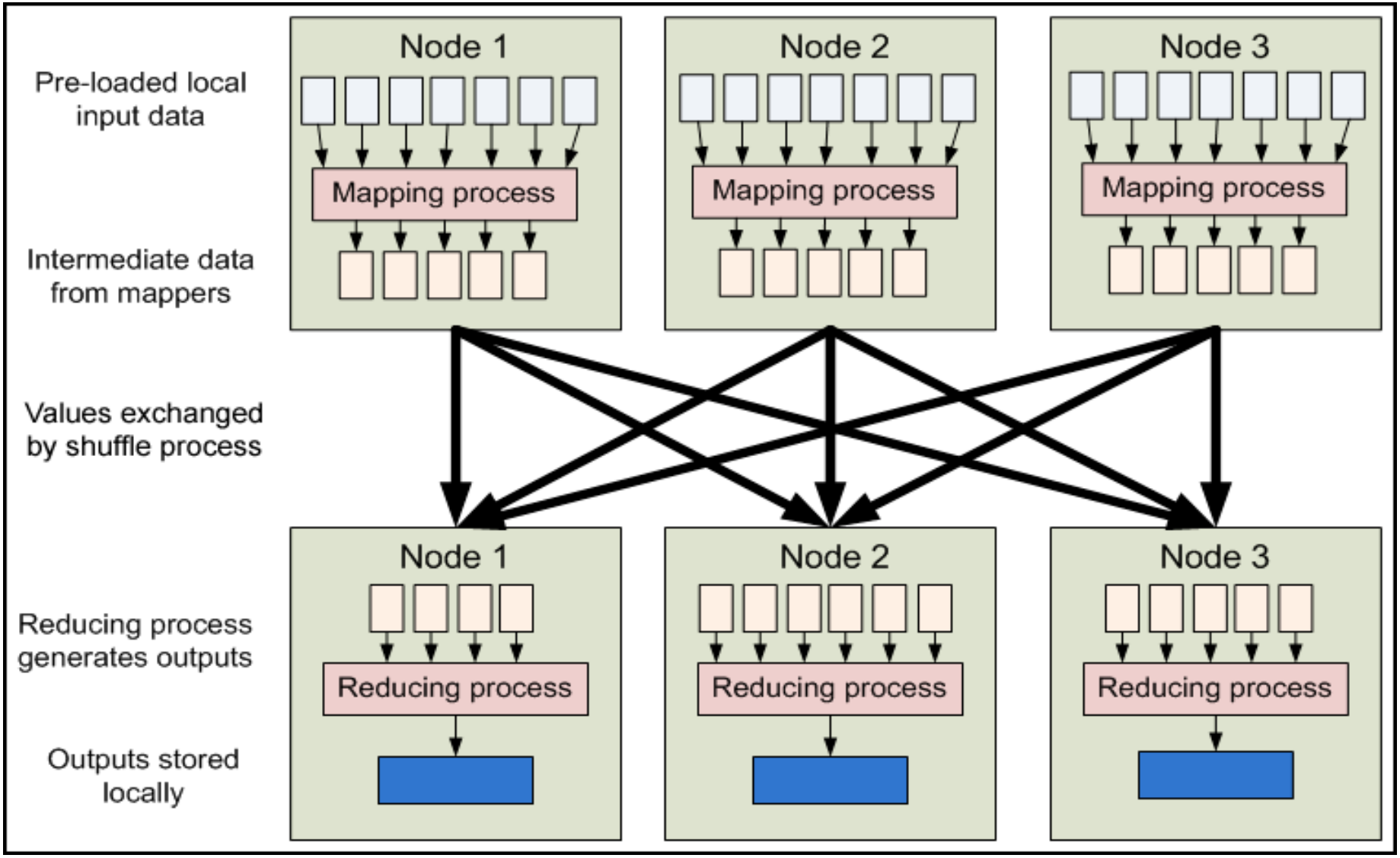


MapReduce的Reduce过程

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。



MapReduce的数据流程模型



测试数据

27 41 39 29 51 45 24 28 56 52 29 51 18 25 19 10
52 37 18 25 23 52 19 33 59 24 39 58 51 12

44 10 42 19 35 28 39 33 58 45 28 35 26 55 32 33
27 40 10 31 42 15 41 56 42 47 40 45 28 52

52 28 50 12 42 28 17 50 31 33 42 14 34 19 23 22
40 21 54 43 52 29 38 53 34 28 11 15 25 44

27 27 43 58 24 12 33 45 39 43 19 57 38 55 54 29
28 58 36 44 59 26 27 21 31 55 29 53 39 38

58 52 46 37 20 49 10 28 15 24 35 38 14 44 59 48
42 18 59 38 43 23 19 28 30 24 36 10 30 15

```
package Sum;
import java.io.IOException;
public class Sum {

    public static class SumMap extends Mapper<Object, Text, Text, IntWritable>{

        private Text SUM=new Text("SUM");
        private IntWritable number=new IntWritable();
        public void map(Object key, Text value, Context context) throws IOException, InterruptedException
        {
            StringTokenizer tokenzer=new StringTokenizer(value.toString().trim());
            while(tokenzer.hasMoreTokens()){
                number.set(Integer.parseInt(tokenzer.nextToken()));
                SUM.set(key.toString());
                context.write(SUM, number);
            }
        }

}
 public static class SumReduce extends Reducer<Text,IntWritable,Text,IntWritable>{

     private IntWritable Sum=new IntWritable();
     public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException
        {
         int Sume=0;
         for(IntWritable aa:values)
         {
             Sume=Sume+aa.get();
         }
         Sum.set(Sume);
         context.write(key, Sum);
        }
 }


 public static void main(String[] args) throws Exception
   {
        Configuration _conf = new Configuration();

        Job job = new Job(_conf, "sum");
        job.setJarByClass(Sum.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        job.setMapperClass(SumMap.class);
        job.setReducerClass(SumReduce.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        System.exit(job.waitForCompletion(true) ? 0 : 1);
      }
}
```

Map

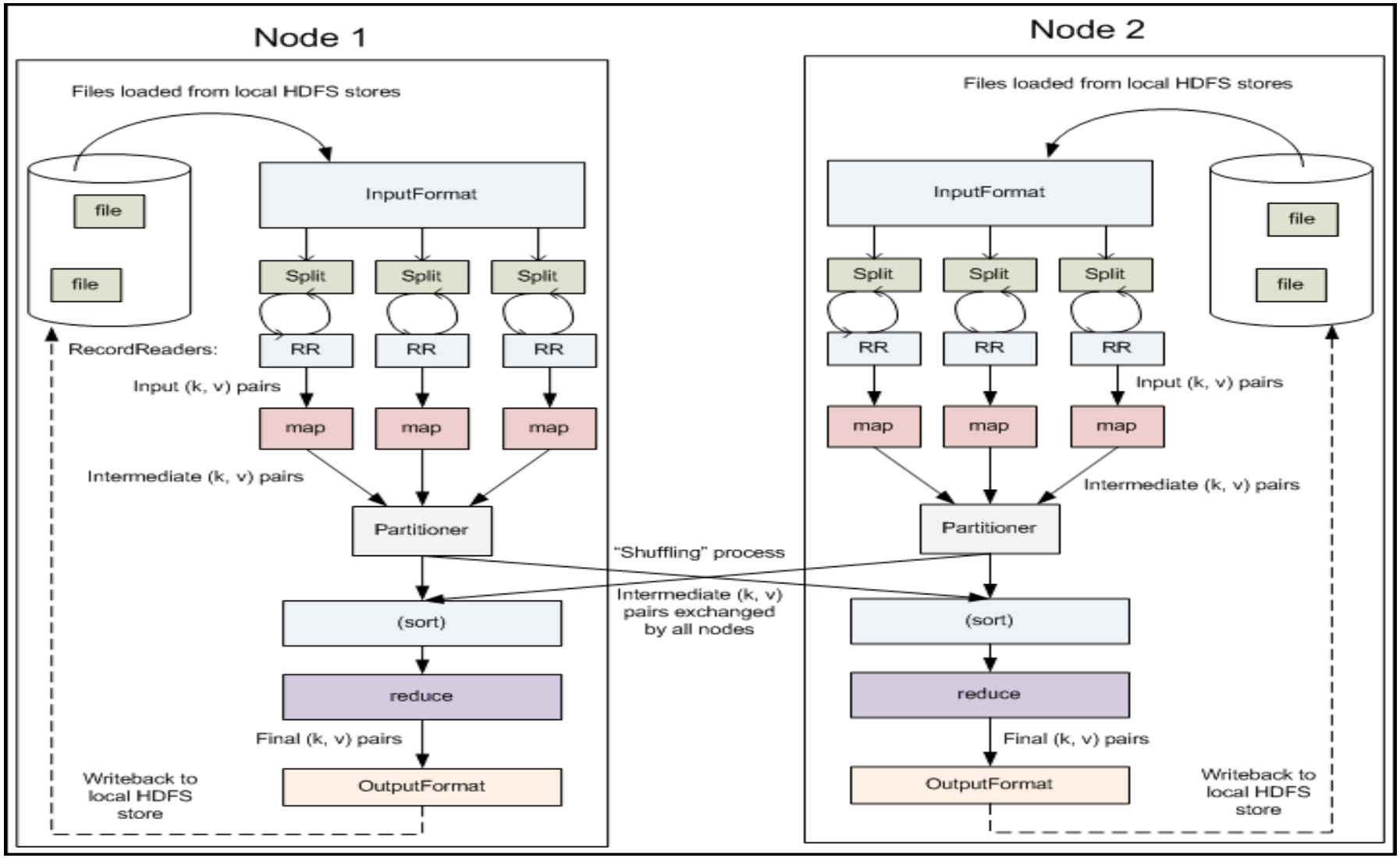
Reduce

Driver

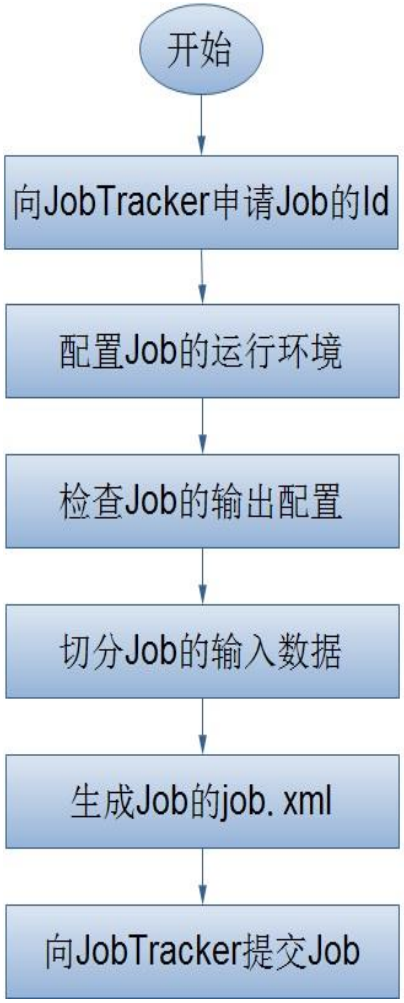
MapReduce特性

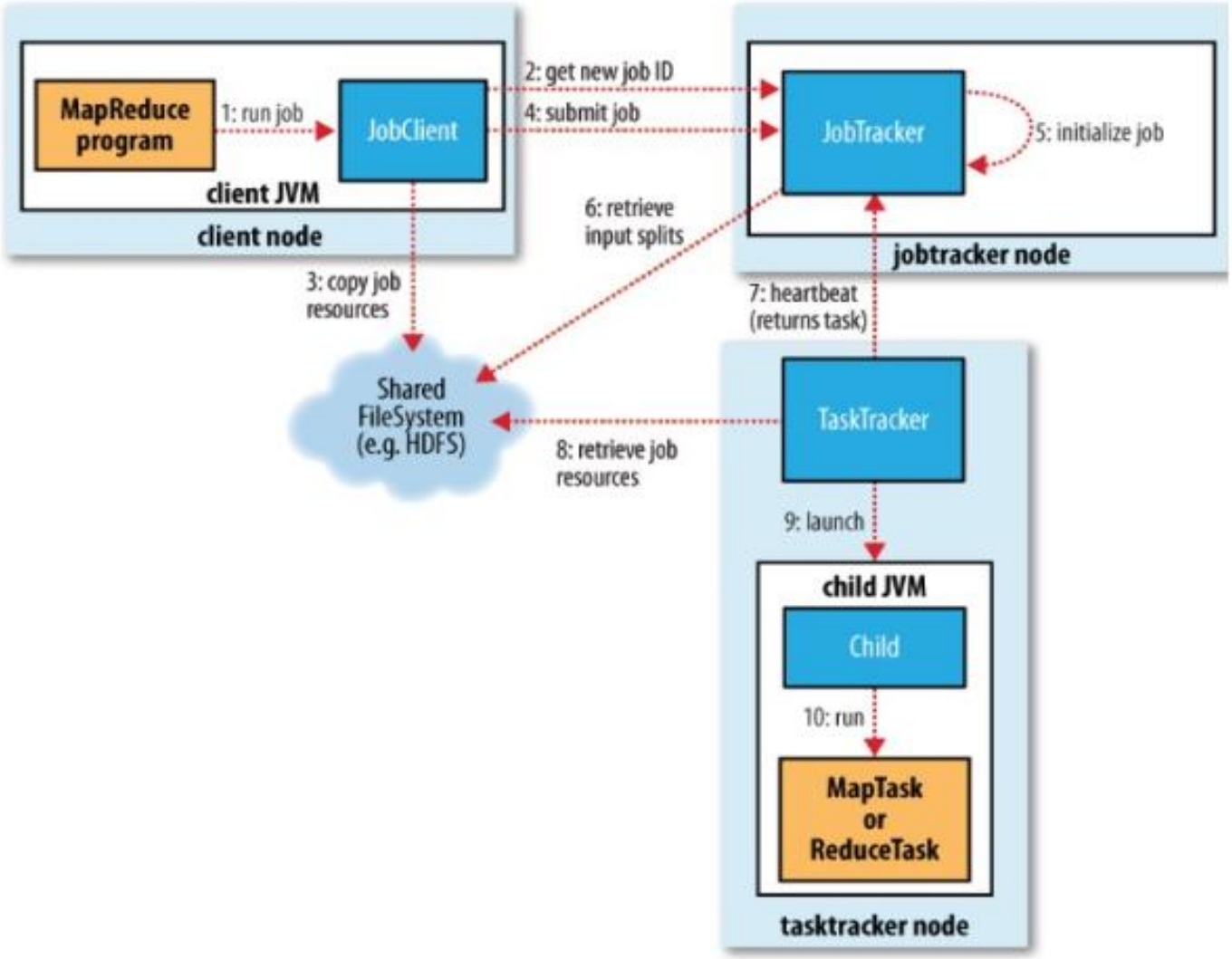
- 自动实现分布式并行计算
- 容错
- 提供状态监控工具
- 模型抽象简洁，程序员易用

MapReduce剖析图

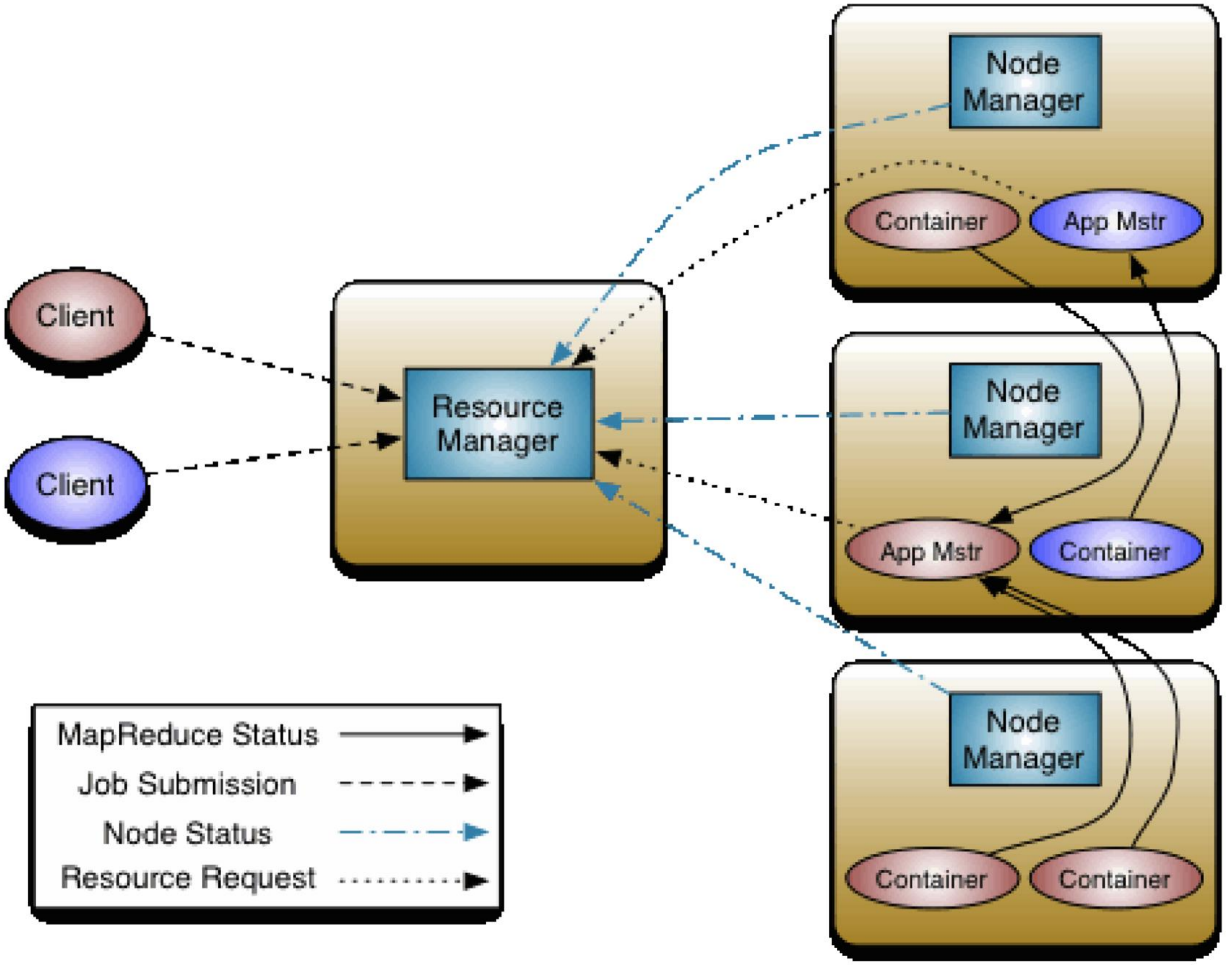


MapReduce执行流程-MRv1

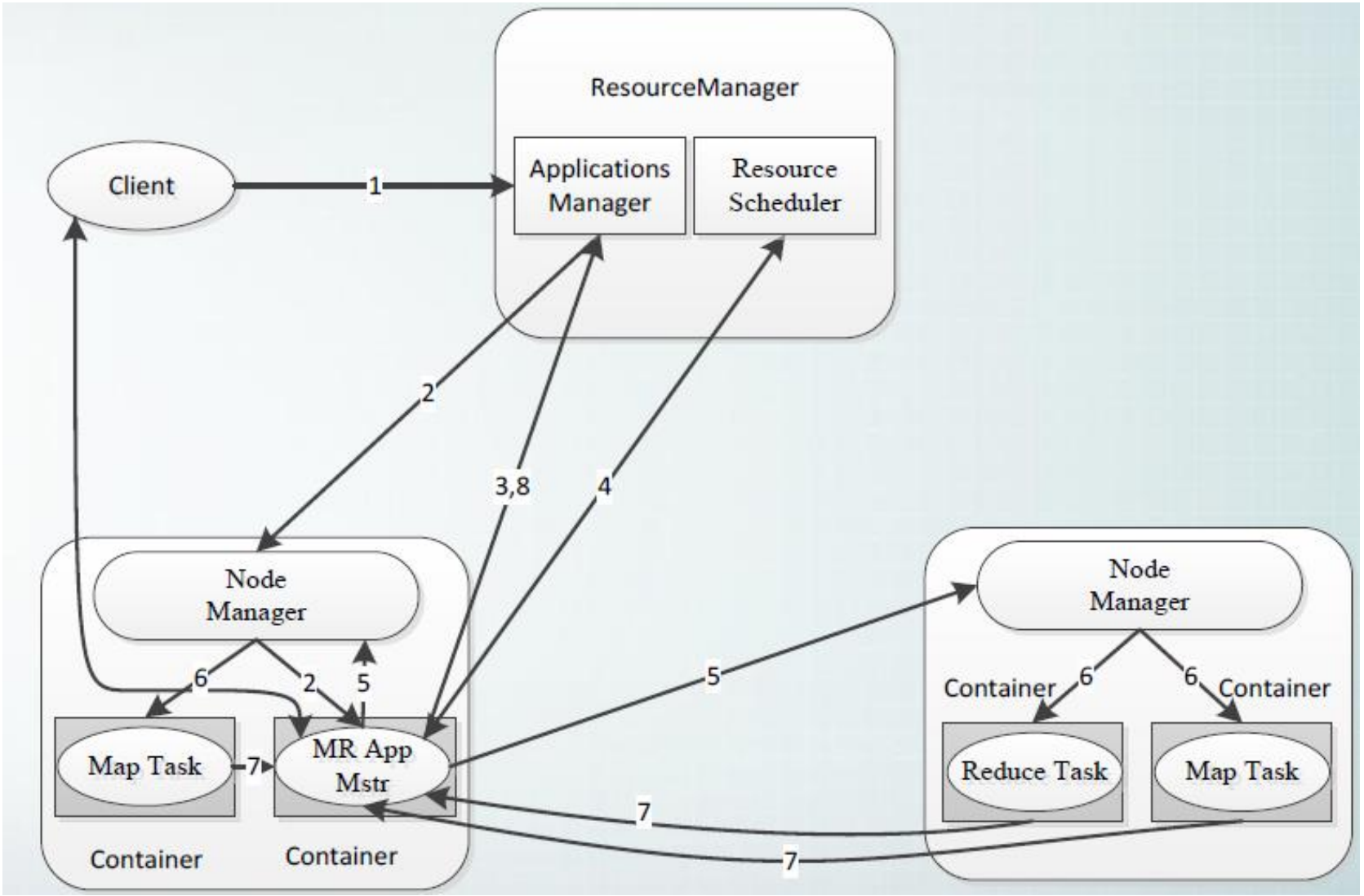




MapReduce执行流程-MRv2



MapReduce On Yarn



MapReduce执行流程-MRv2

- **1** 用户向YARN中提交应用程序，其中包括ApplicationMaster程序、启动ApplicationMaster的命令、用户程序等。
- **2** ResourceManager为该应用程序分配第一个Container，并与对应的NodeManager通信，要求它在这个Container中启动应用程序的ApplicationMaster。

MapReduce执行流程-MRv2

- **3** ApplicationMaster首先向ResourceManager注册，这样，用户可以直接通过ResourceManager查看应用程序的运行状态，然后，它将为各个任务申请资源，并监控它的运行状态，直到运行结束，即重复步骤4~7。
- **4** ApplicationMaster采用轮询的方式通过RPC协议向ResourceManager申请和领取资源。

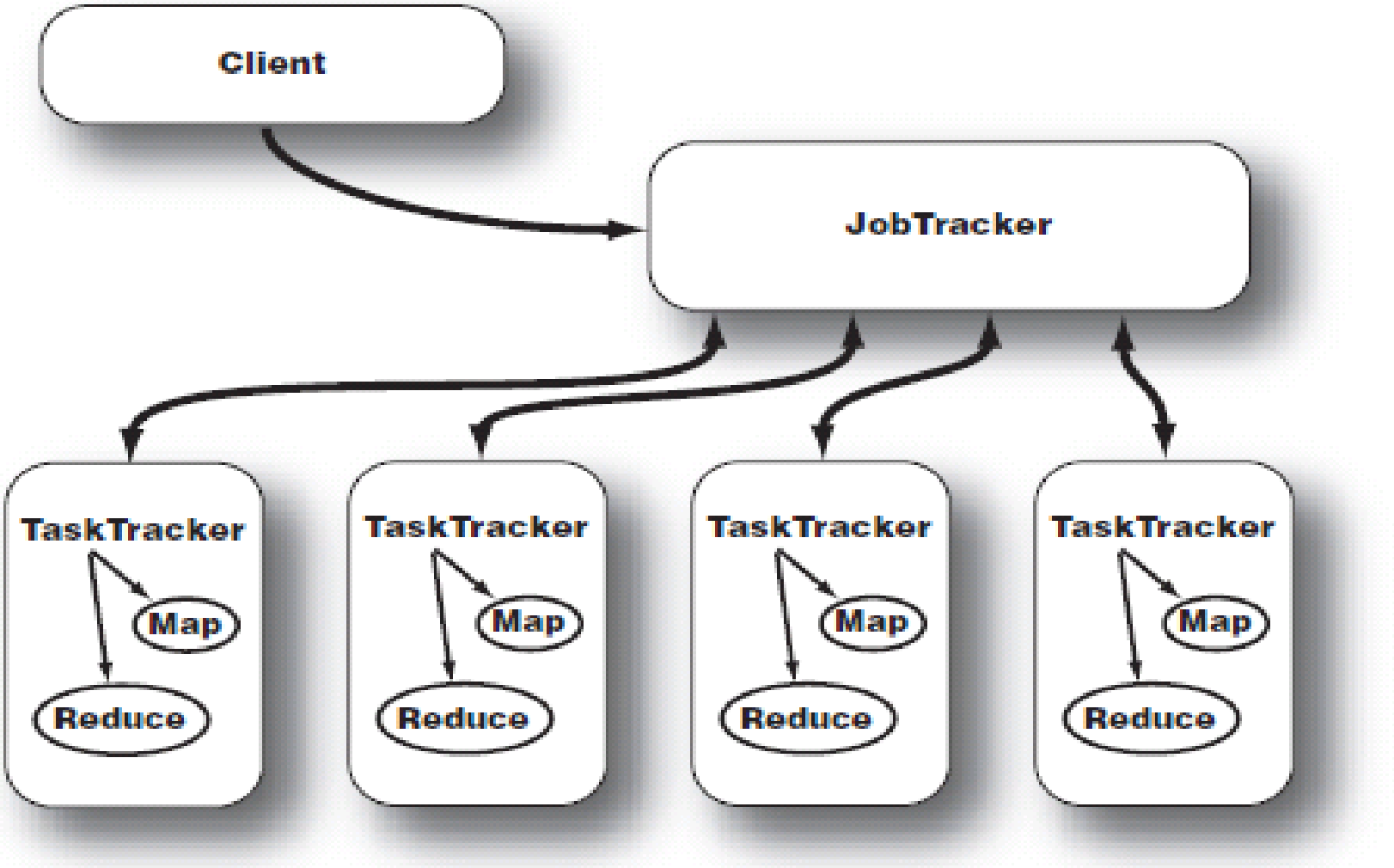
MapReduce执行流程-MRv2

- **5** 一旦ApplicationMaster申请到资源后，则与对应的NodeManager通信，要求其启动任务。
- **6** NodeManager为任务设置好运行环境（包括环境变量、jar包、二进制程序等）后，将任务启动命令写到一个脚本中，并通过运行该脚本启动任务。

MapReduce执行流程-MRv2

- **7** 各个任务通过某个RPC协议向ApplicationMaster汇报自己的状态和进度，以让ApplicationMaster随时掌握各个任务的运行状态，从而可以在任务失败时重新启动任务。在应用程序运行过程中，用户可随时通过RPC向ApplicationMaster查询应用程序的当前运行状态。
- **8** 应用程序运行完成后，ApplicationMaster向ResourceManager注销，并关闭自己。

JobTracker和TaskTracker简介



JobTracker失败

1）JobTracker失败在所有的失败中是最严重的一种

2）hadoop没有处理jobtracker失败的机制。--它是一个单点故障

3）在未来的新版本中可能可以运行多个JobTracker

4）可以使用ZooKeeper来协作JobTracker

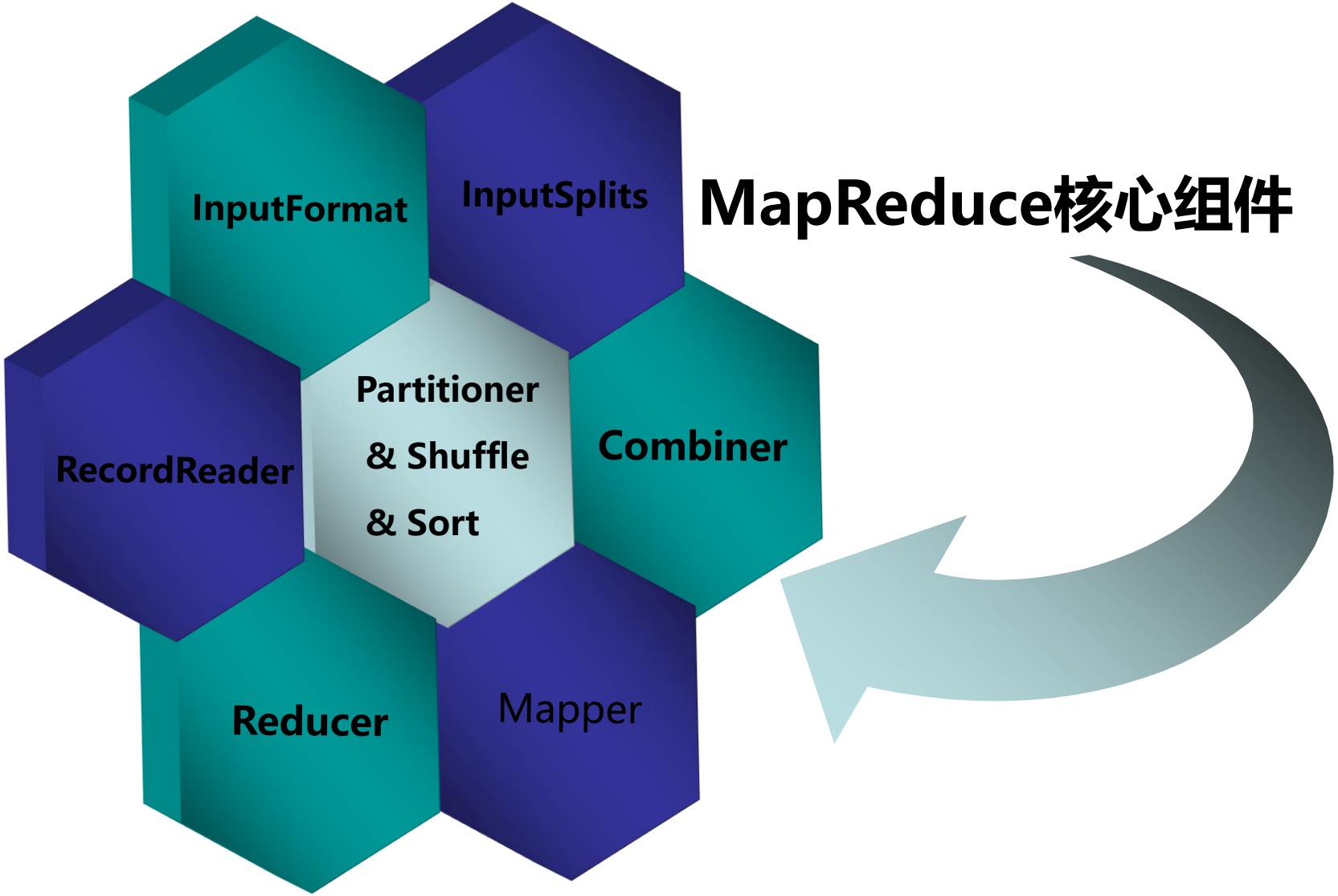
JobTracker失败

1）一个TaskTracker由于崩溃或运行过于缓慢而失败，它会向JobTracker发送“心跳”。

2）如果有未完成的作业，JobTracker会重新把这些任务分配到其他的TaskTracker上面运行。

3）即使TaskTracker没有失败也可以被JobTracker列入黑名单。

MapReduce核心组件

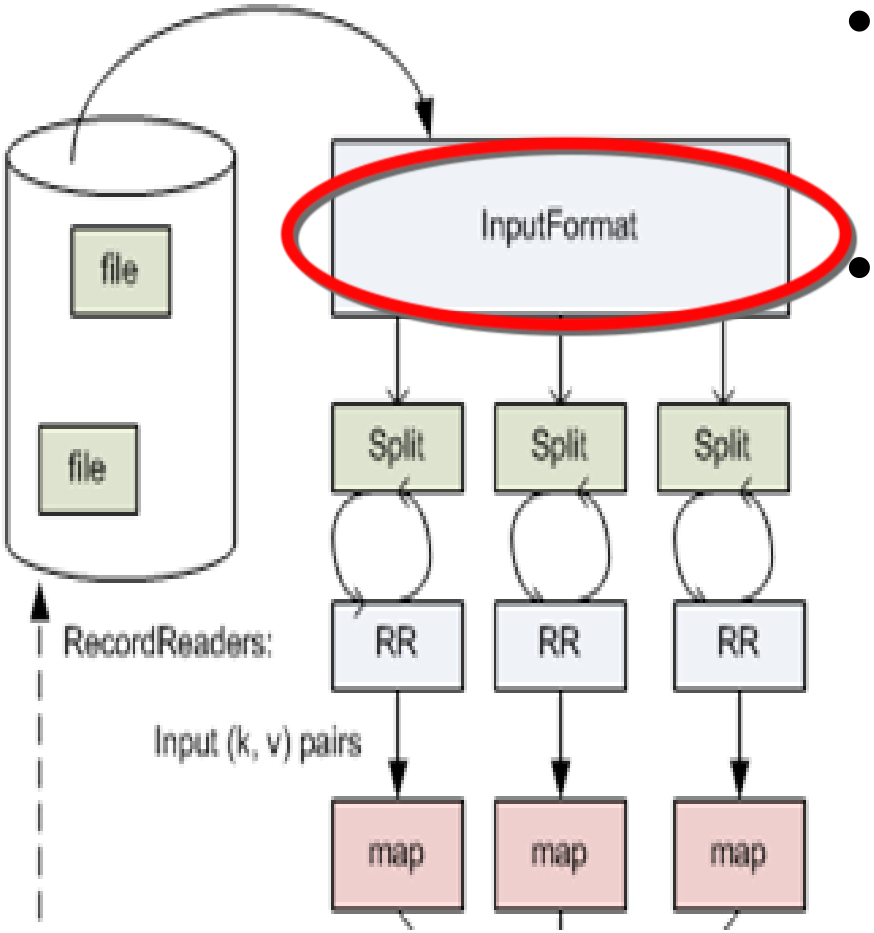


Input Files

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

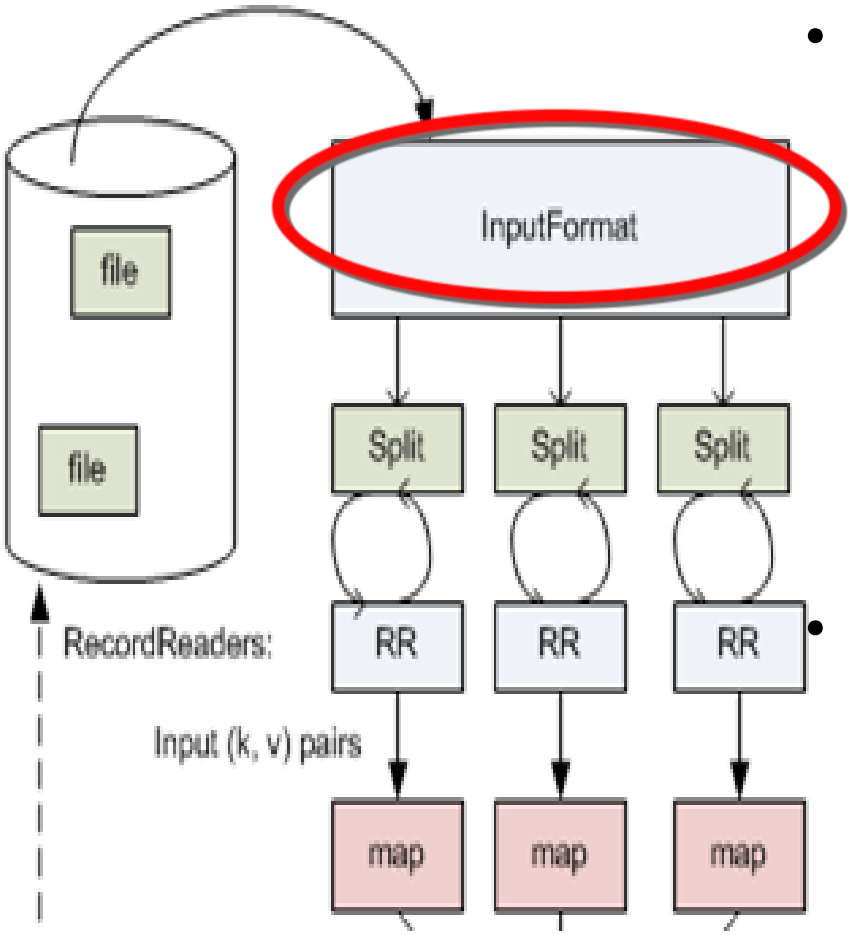
文件经常很大，甚至有几十个GB

文件输入格式InputFormat



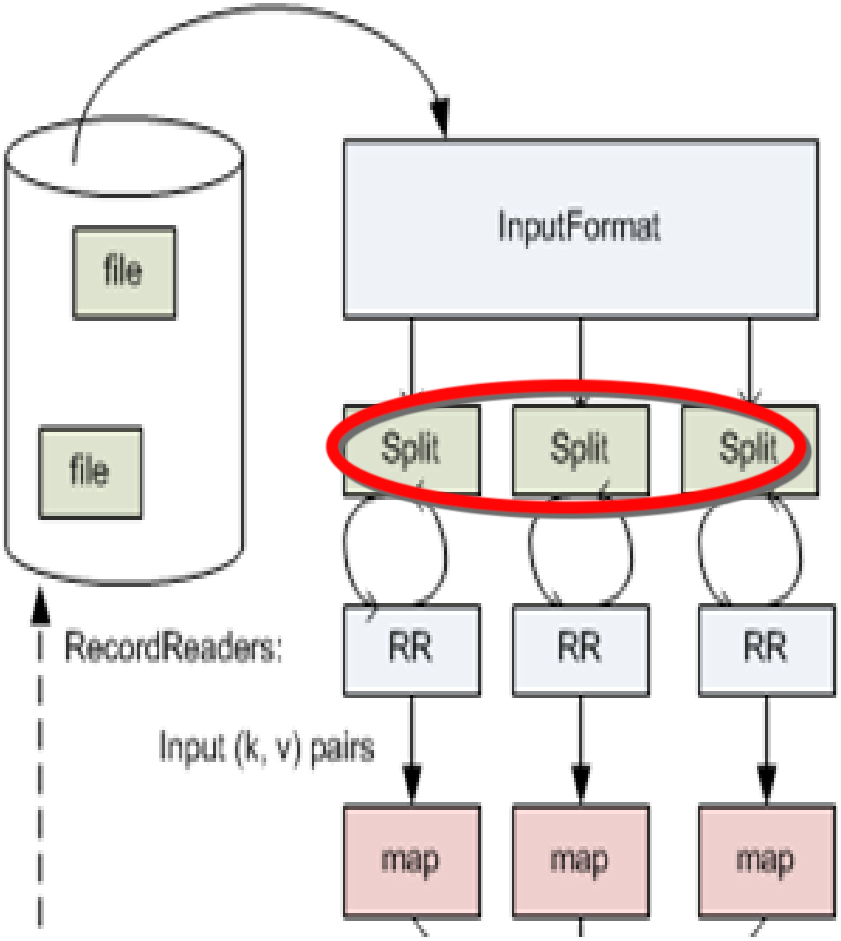
- 定义了数据文件如何分割和读取
- InputFile提供了以下一些功能
  - 选择文件或者其它对象，用来作为输入
  - 定义InputSplits, 将一个文件分为不同任务
  - 为RecordReader提供一个工厂，用来读取这个文件

文件输入格式InputFormat



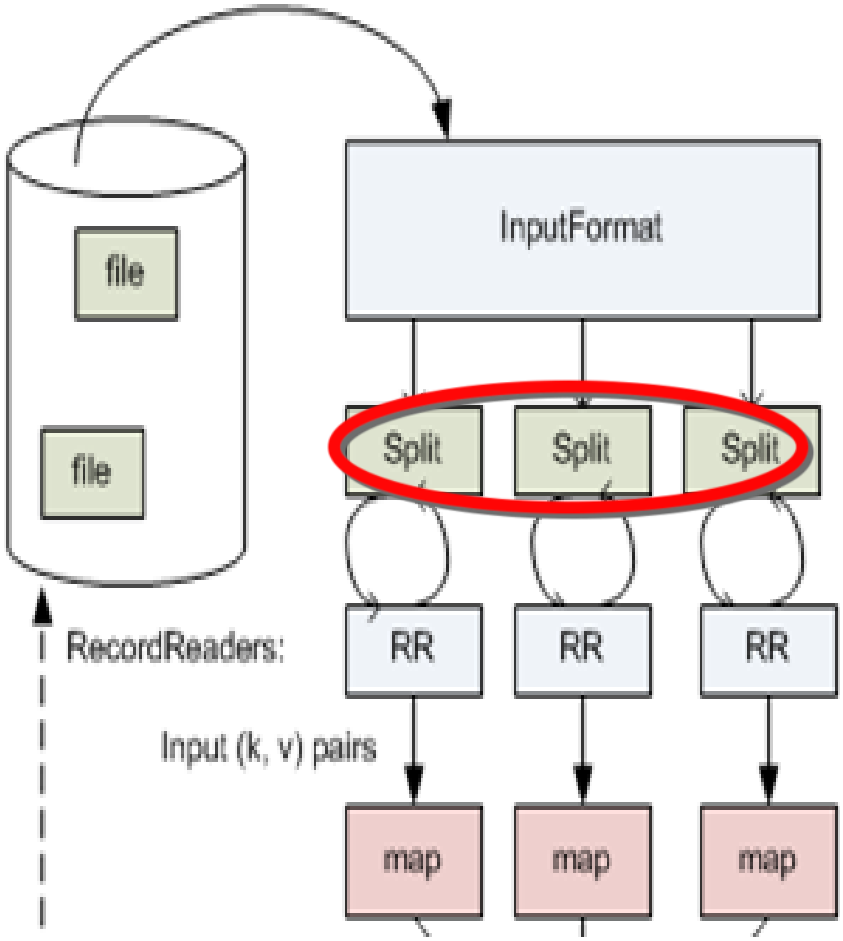
- 有一个抽象的类FileInputFormat，FileInputFormat从这个目录中读取所有文件，然后FileInputFormat将这些文件分割为多个InputSplits。
- 通过在JobConf对象上设置JobConf.setInputFormat设置文件输入的格式

输入数据分块InputSplits



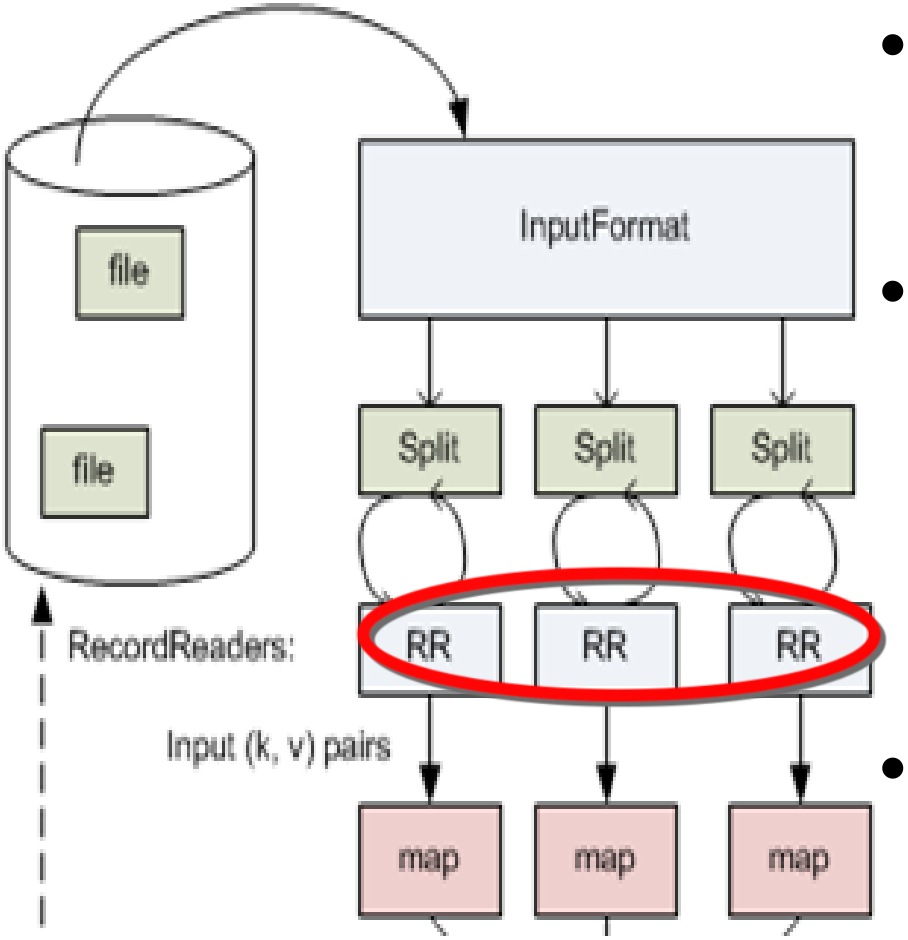
- InputSplit定义了输入到单个Map任务的输入数据
- 一个MapReduce程序被统称为一个Job，可能有上百个任务构成
- InputSplit将文件分为64MB的大

输入数据分块InputSplits



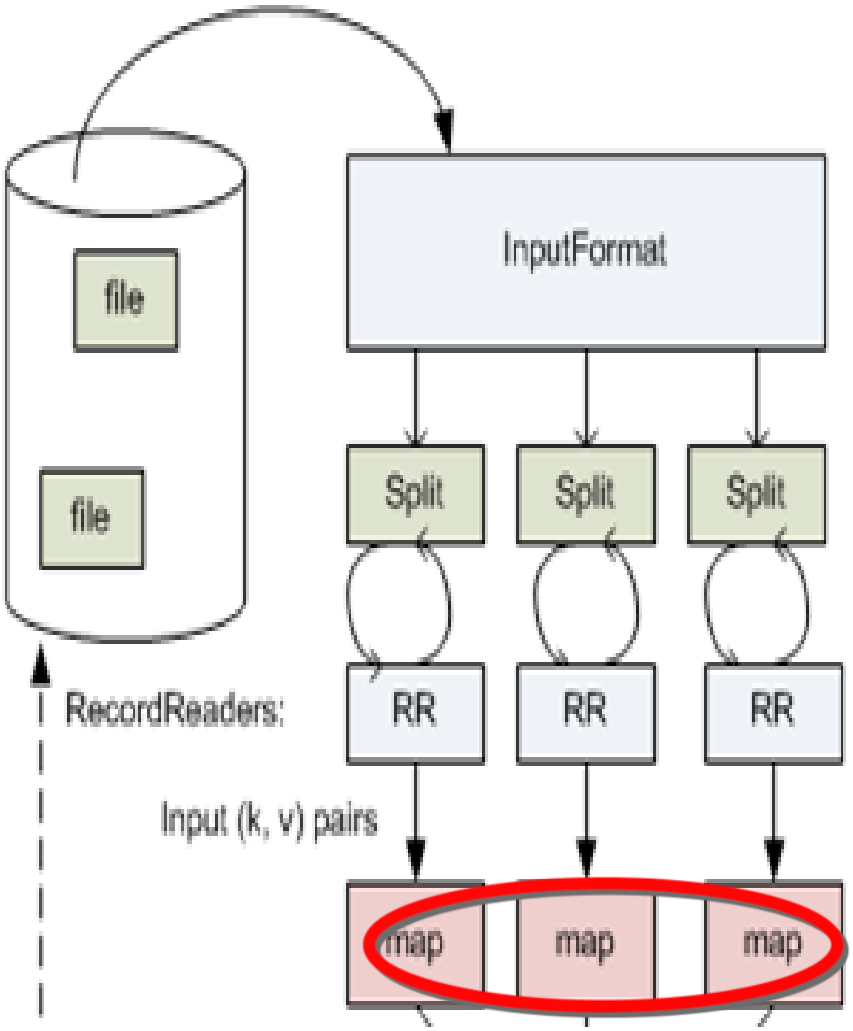
- 配置文件hadoop-site.xml中的mapred.min.split.size参数控制这个大小
- mapred.tasktracker.map.tasks.maximum用来控制某一个节点上所有map任务的最大数目

数据记录读入RecordReader



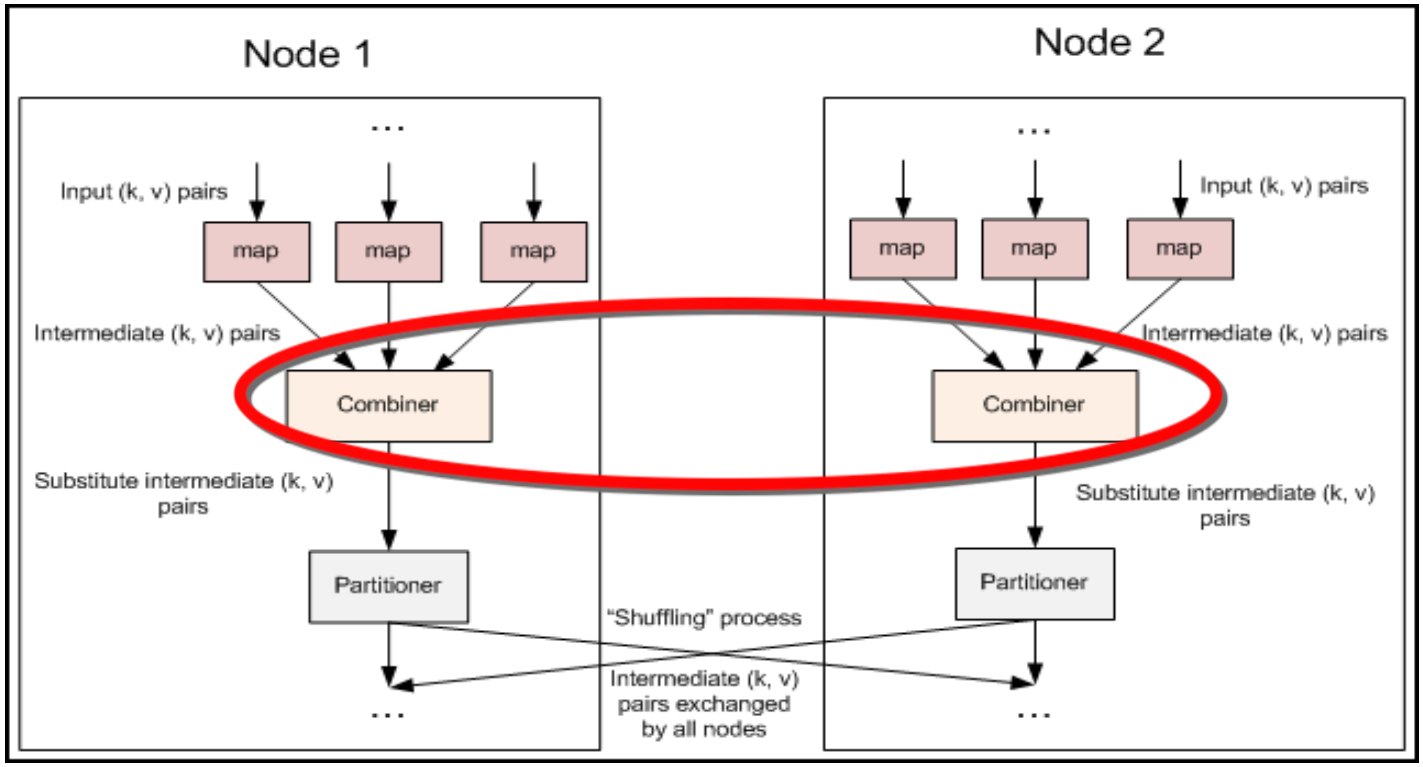
- InputSplit定义了一个数据分块，但是没有定义如何读取数据记录
- RecordReader实际上定义了如何将数据记录转化为一个(key,value)对的详细方法，并将数据记录传给Mapper类
- TextInputFormat提供了LineRecordReader，读入一个文本行数据记录

Mapper



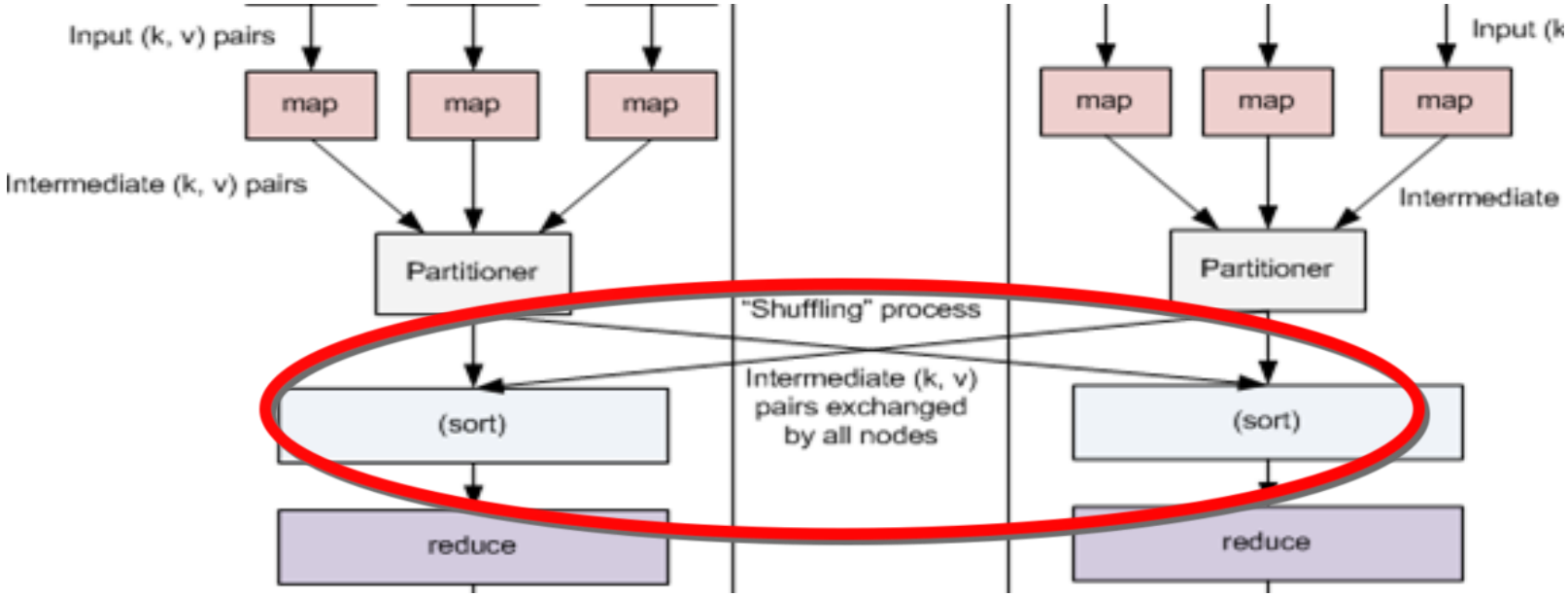
- 每一个Mapper类的实例生成了一个Java进程，负责处理某一个InputSplit上的数据
- 有两个额外的参数OutputCollector以及Reporter，前者用来收集中间结果，后者用来获得环境参数以及设置当前执行的状态。
- 现在的版本用Mapper.Context提供给每一个Mapper函数，用来提供上面两个对象的功能

Combiner



- 合并相同key的键值对，减少partitioner数据通信开销
- conf.setCombinerClass(Reduce.class)
- 本地执行的一个Reducer，满足一定的条件才能够执行

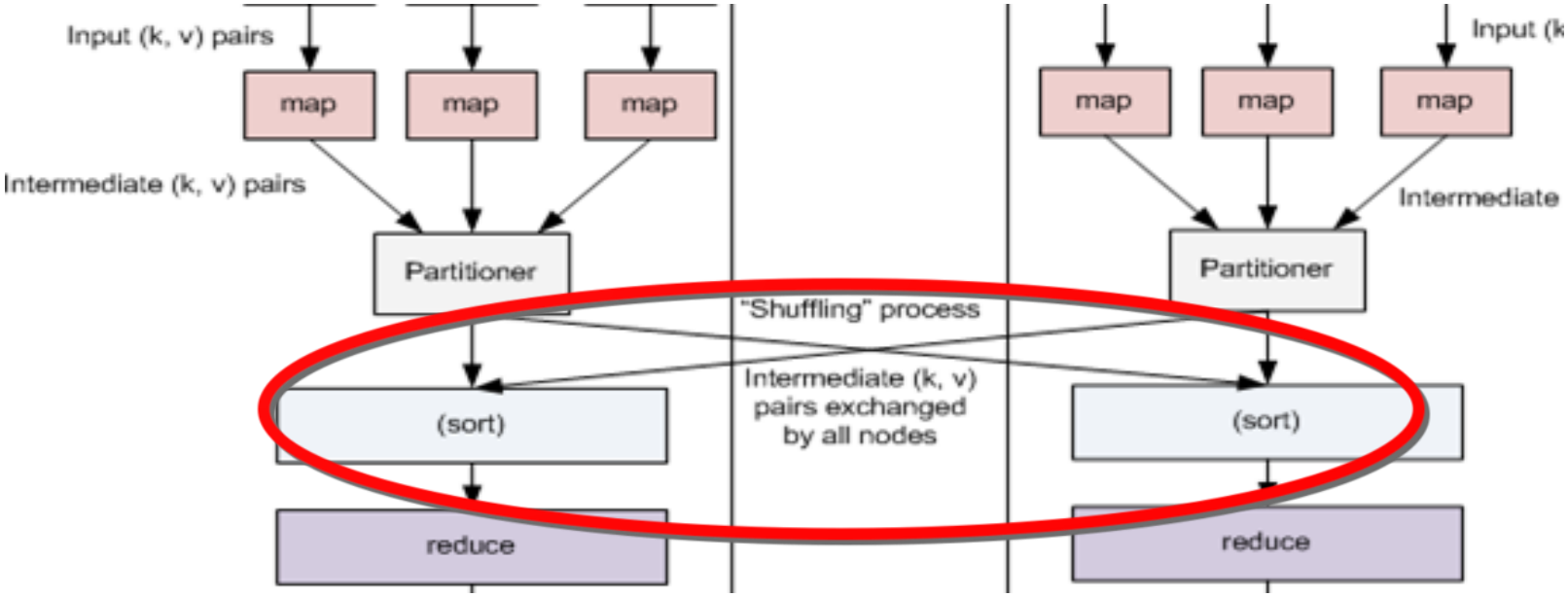
Partitioner & Shuffle&Sort



Partitioner & 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)对传给哪个节点

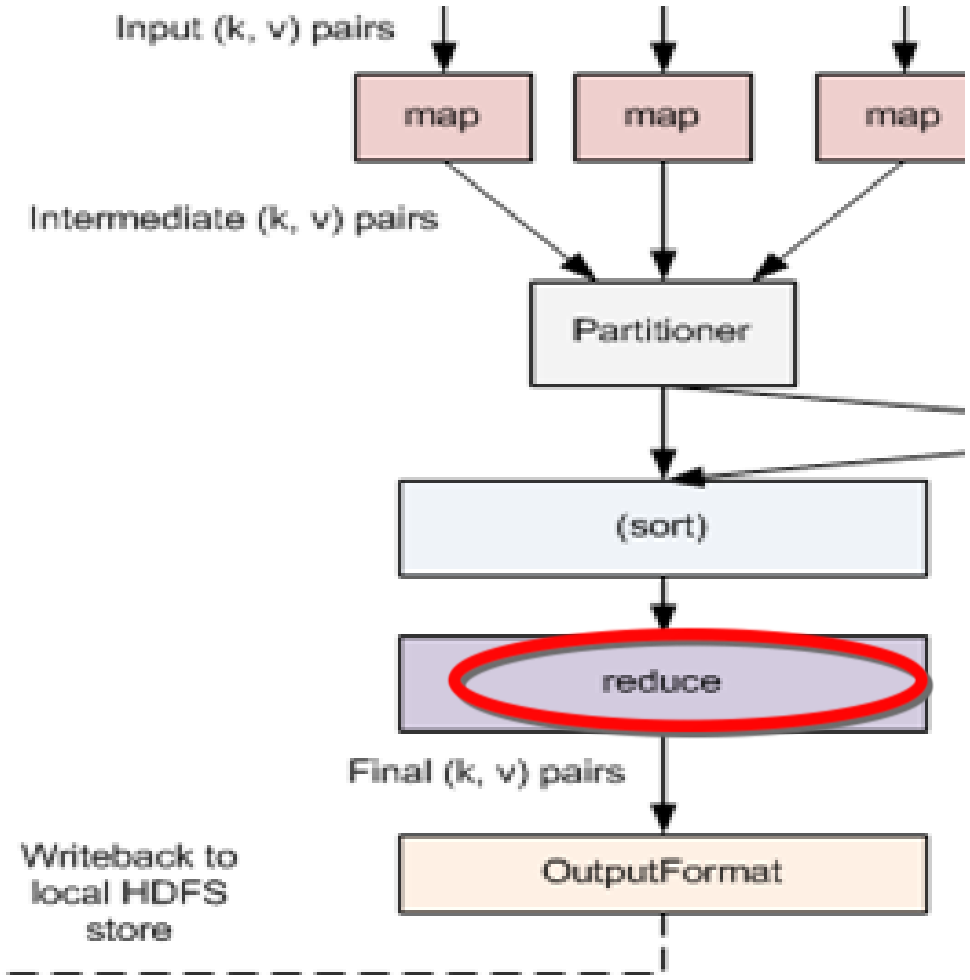
Partitioner & Shuffle&Sort



Sort

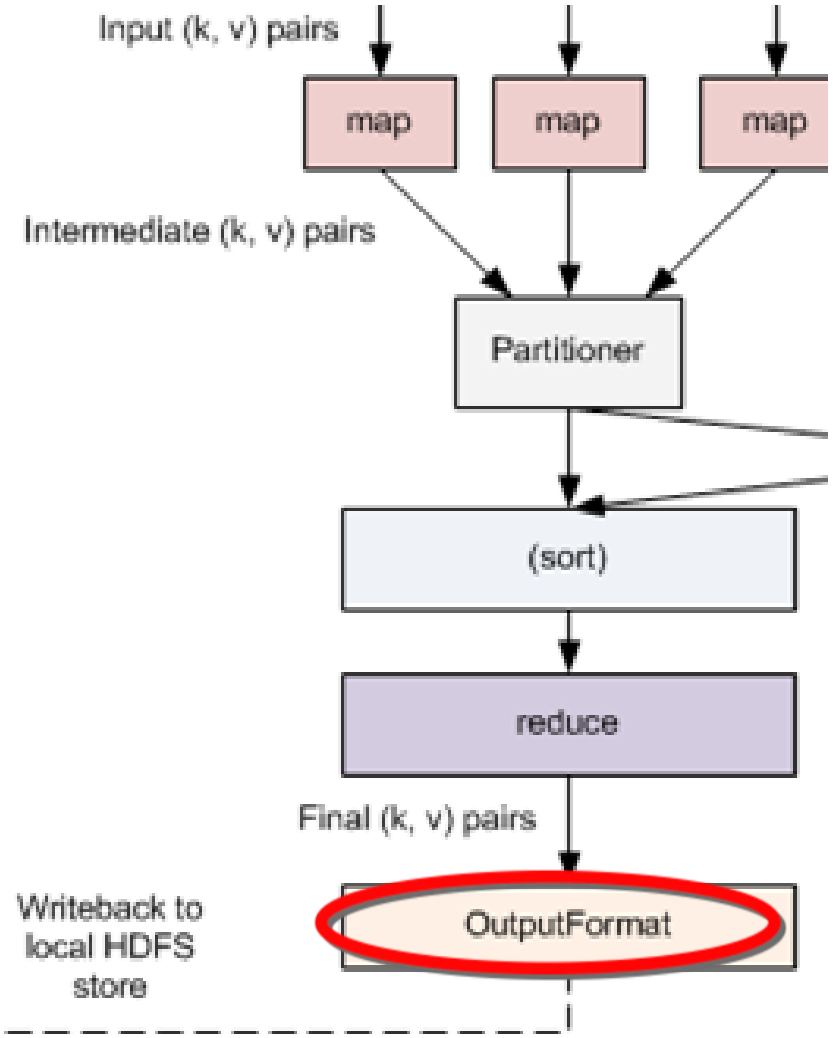
- 传输到每一个Reducer节点上的、将被所有的Reduce函数接收到的Key,value对会被Hadoop自动排序（即Map生成的结果传送到某一个节点的时候，会被自动排序）

Reducer



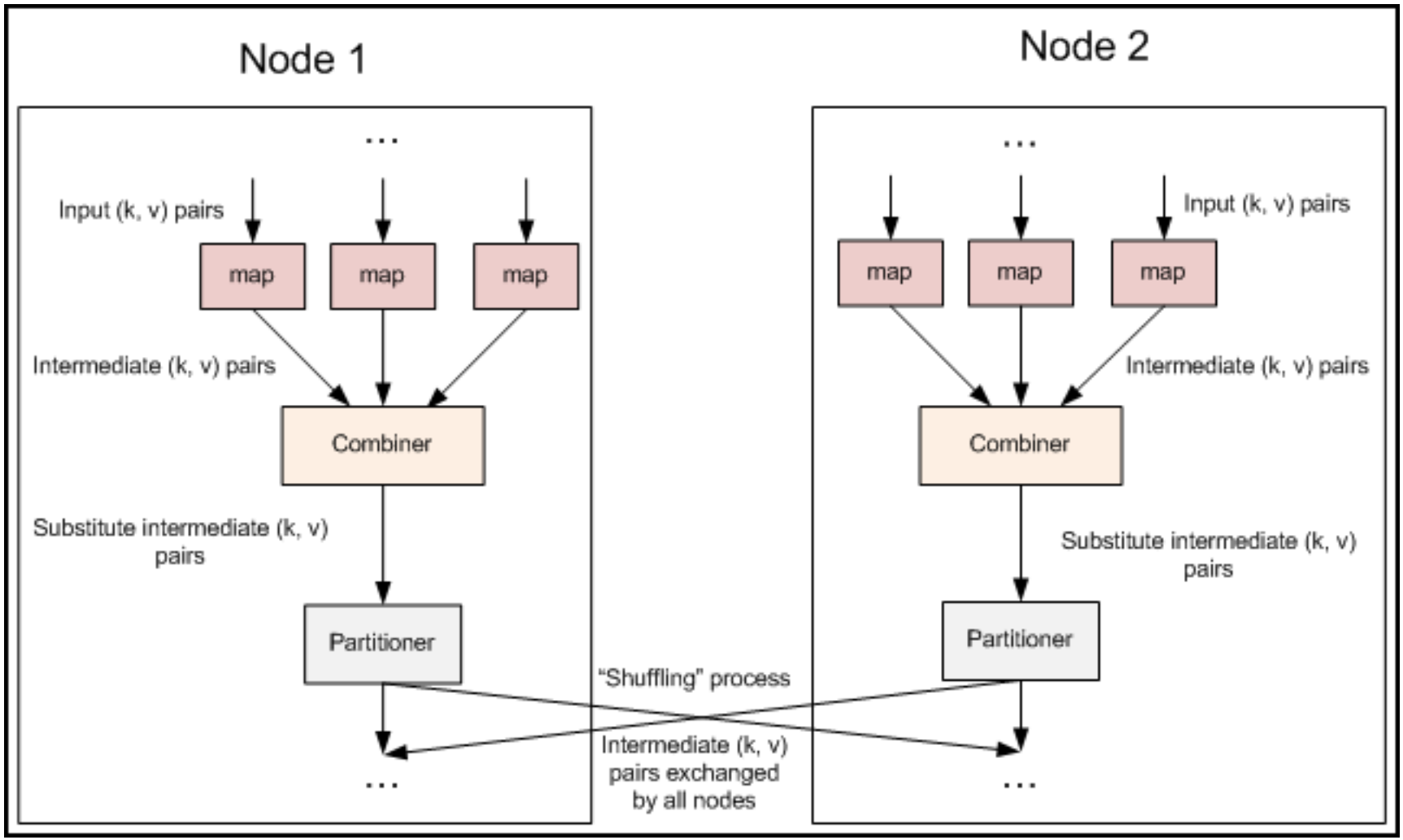
- 做用户定义的Reduce操作
- 接收到一个OutputCollector的类作为输出
- 新版本的编程接口是Reducer.Context

文件输出格式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()

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
Input (k, v) pairs
map
map
map
Intermediate (k, v) pairs
Combiner
Substitute intermediate (k, v) pairs
Partitioner

Combiner

conf.setCombinerClass(Reduce.class);

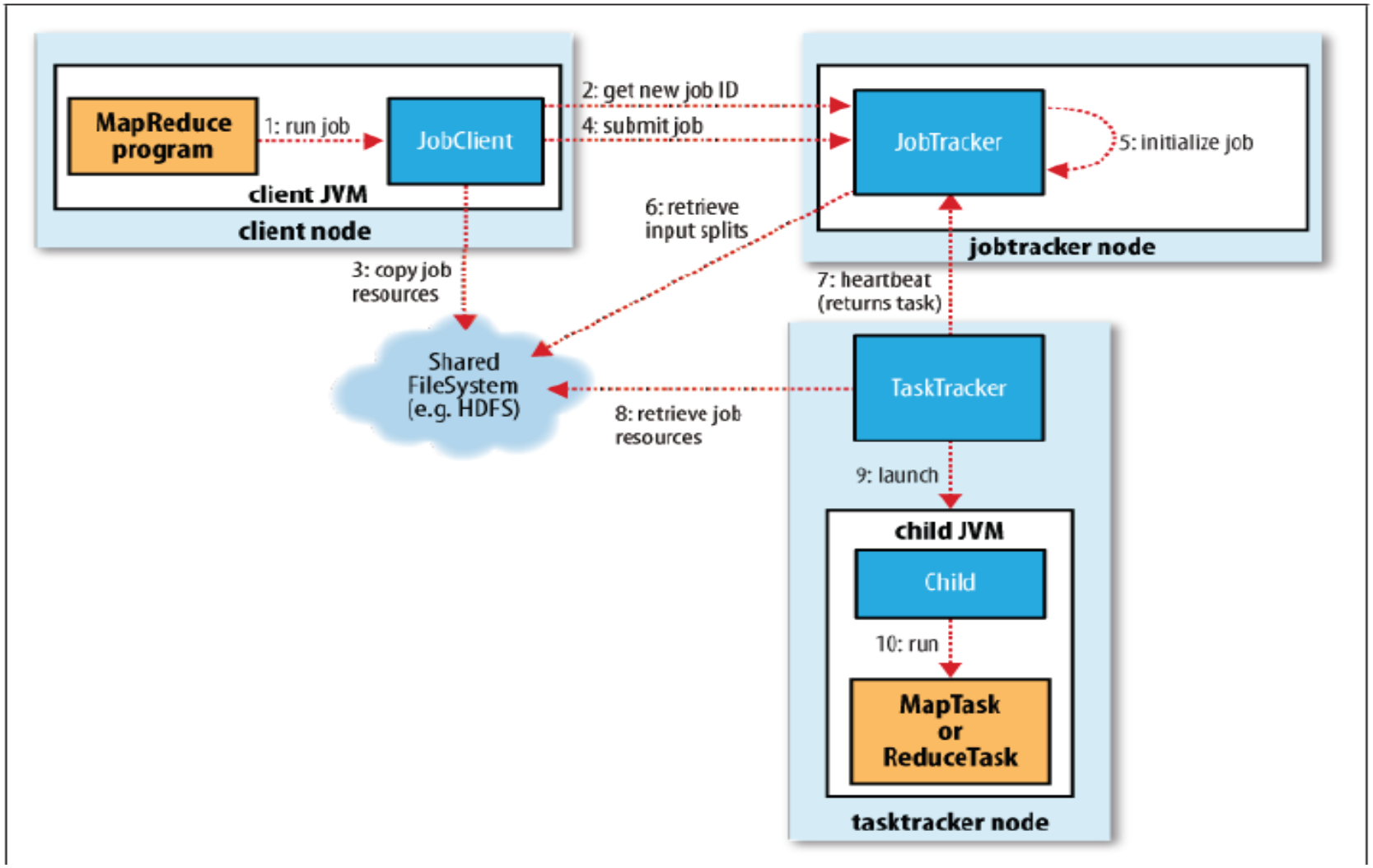
是在本地执行的一个Reducer，满足一定的条件才能够执行。

MapReduce

Hadoop Job

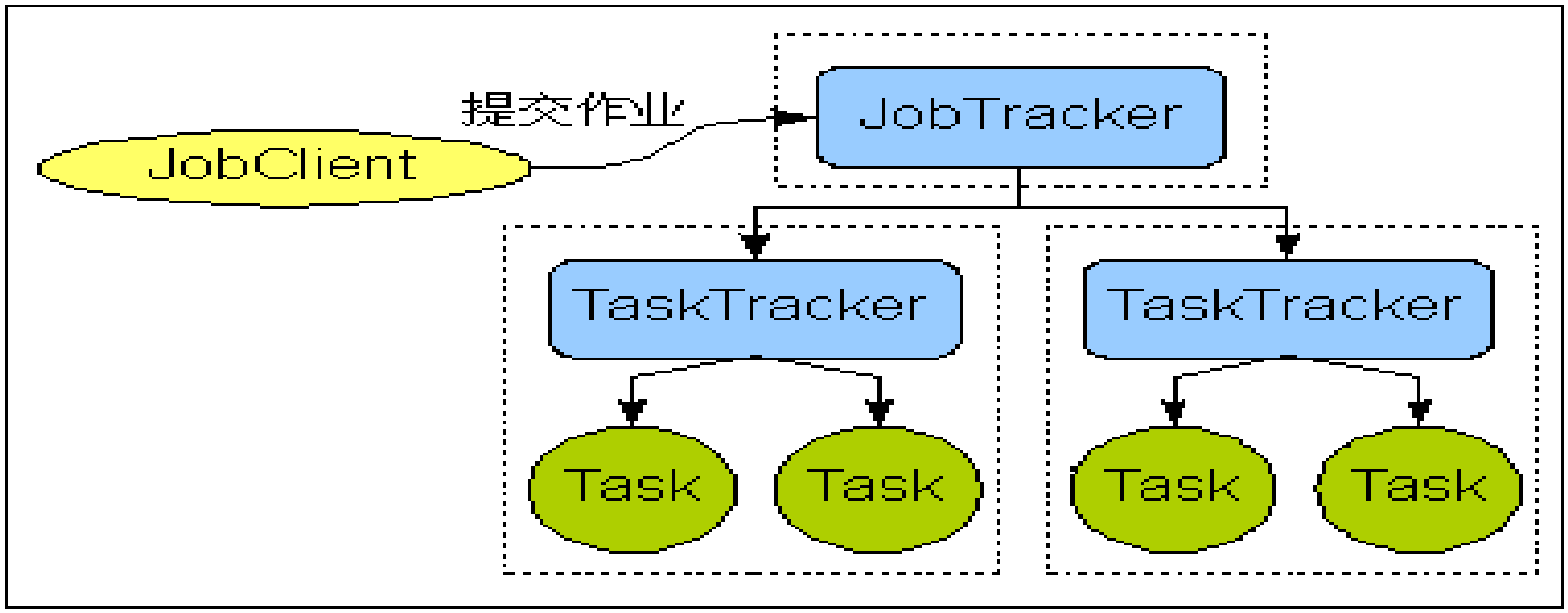
- Job介绍
  - Hadoop中所有MapReduce程序以Job形式提交给集群运行。
  - 一个MapReduce Job被划分为若干个Map Task和Reduce Task并行执行。
  - 一个Job的提交包括数据和程序(Jar文件)的提交。

MapReduce


MapReduce program
1: run job
JobClient
client JVM
client node
2: get new job ID
4: submit job
JobTracker
5: initialize job
jobtracker node
6: retrieve input splits
3: copy job resources
7: heartbeat (returns task)
Shared FileSystem (e.g. HDFS)
8: retrieve job resources
TaskTracker
9: launch
child JVM
Child
10: run
MapTask or ReduceTask
tasktracker node

MapReduce

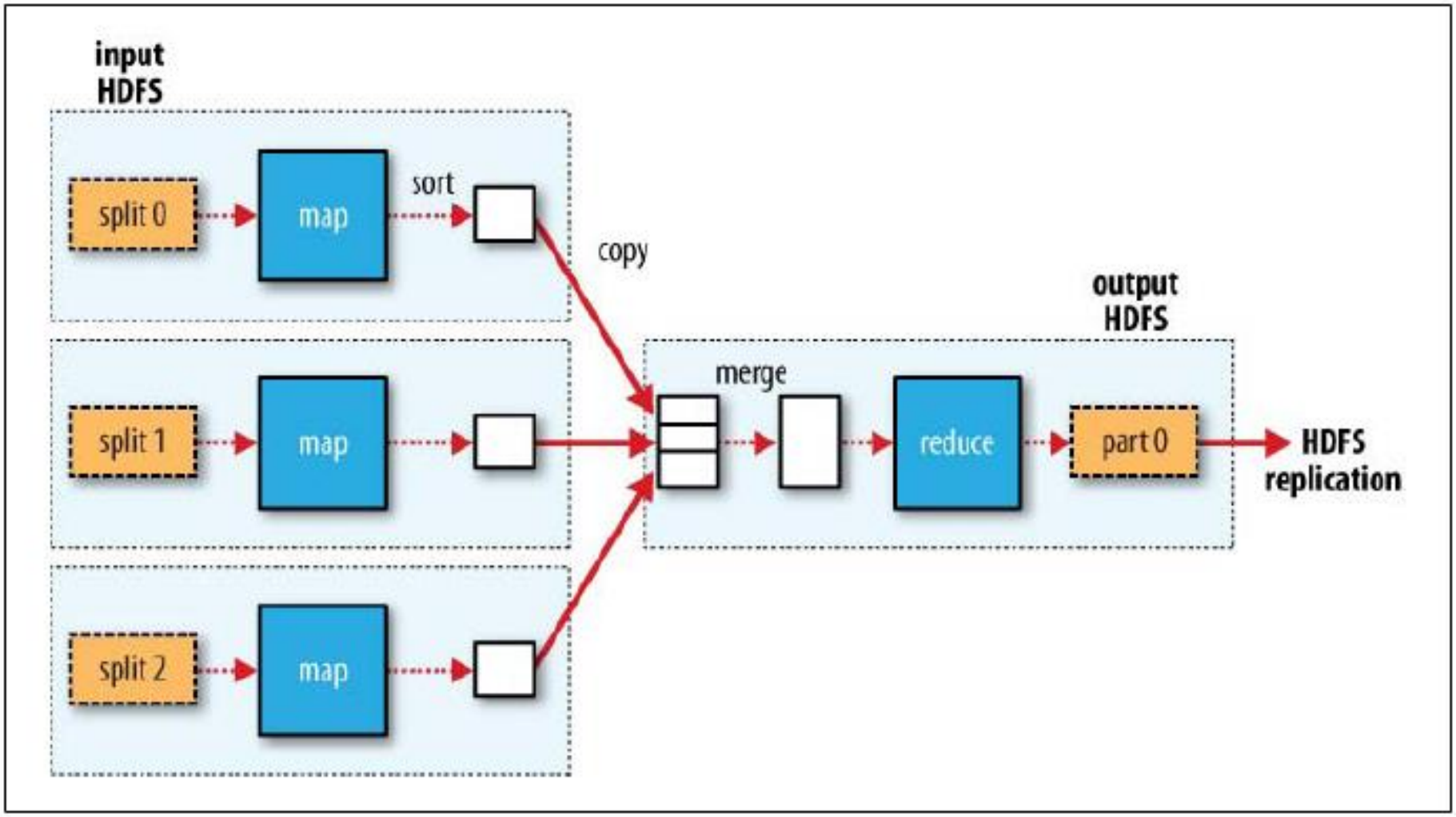
- JobTracker一直在等待JobClient提交作业
- TaskTracker每隔3秒向 JobTracker发送心跳heartbeat询问有没有任务可做，如果有，让其派发任务给它执行
- 这是一道pull过程: slave主动向master拉生意



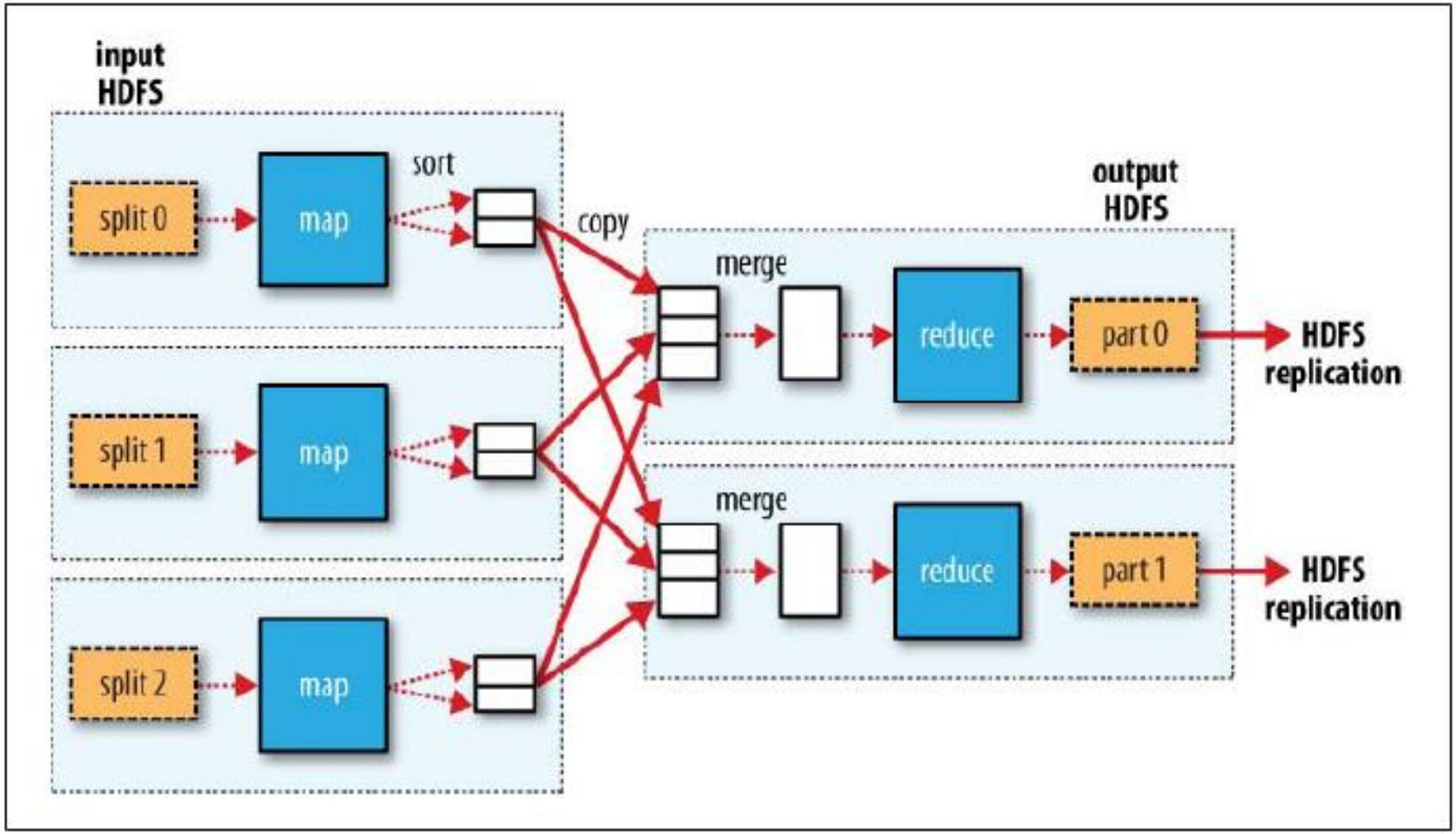
MapReduce

- 特点：Fault-tolerant 容错，很重要！
- M/R Failures
- Task fails
- Try again?
- Try again somewhere else?
- Report failure
- 只有当map处理全部结束后，reduce过程才能够开始
- Map需要考虑数据局部性，Reduce无需考虑数据局部性

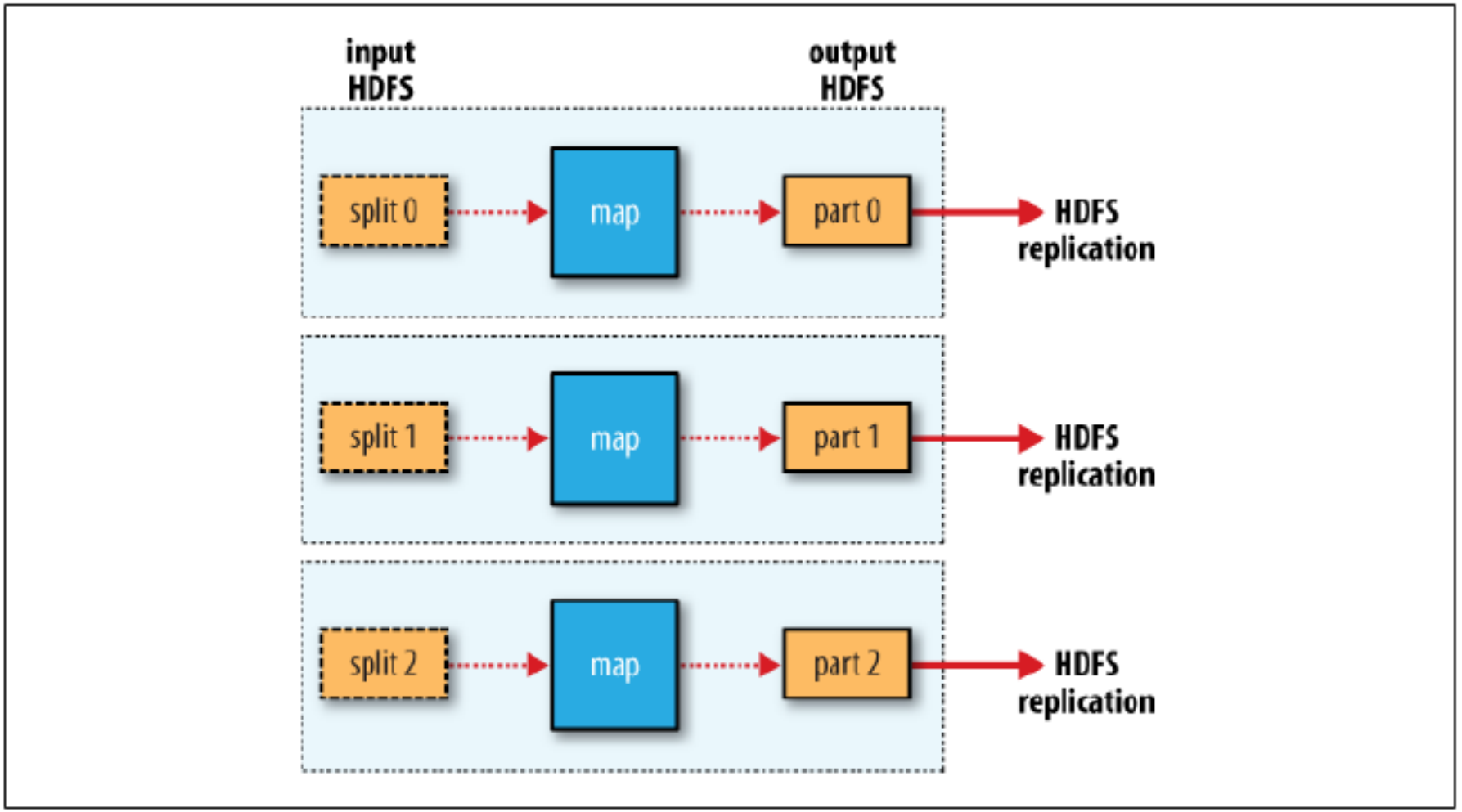
单一Reduce处理


input
HDFS
split 0
map
sort
split 1
map
split 2
map
copy
merge
output
HDFS
reduce
part 0
HDFS
replication

多个Reduce处理


input
HDFS
split 0
map
sort
split 1
map
split 2
map
copy
merge
reduce
part 0
output
HDFS
HDFS
replication
merge
reduce
part 1
HDFS
replication

无Reduce处理


input
HDFS
output
HDFS
split 0
map
part 0
HDFS
replication
split 1
map
part 1
HDFS
replication
split 2
map
part 2
HDFS
replication

Hadoop平台应用特点

- 多种应用共用一个Hadoop平台
  - 生产性应用：数据加载，统计值计算，垃圾数据分析等
  - 批作业：机器学习等
  - 交互式作业：SQL查询、样本采集等
- 不同应用对硬件资源要求不同
  - I/O密集型作业，如：机器学习算法
  - CPU密集型作业：如：过滤，统计值计算，正则匹配等
- 作业之间存在依赖关系
- 如何提高Hadoop平台资源利用效率？ 作业合理调度、监控

MapReduce

- MapReduce的应用：

✓【1】日志分析

✓【2】排序

✓【3】搜索、搜索引擎，创建索引

✓【4】广告计算，广告优化、分析，点击流分析，链接分析

✓【5】搜索关键字进行内容分类

✓【6】计数，统计值计算，统计数据，过滤，分析，查询

MapReduce

- MapReduce的应用：

✓ 【7】垃圾数据分析

✓ 【8】数据分析

✓ 【9】机器学习

✓ 【10】数据挖掘

✓ 【11】大规模图像转换（纽约时报使用Hadoop 和EC2在36个小时内将4TB的TIFF图像—包括405K大TIFF图像，3.3M SGML文章和405K XML文件 — 转换为800K适合在Web上使用的PNG图像）

MapReduce程序：Sum

MapReduce程序：sum

具体分析一个sum程序

- sum.java

首先将原始数据导入到HDFS文件系统中，放置在/user/input目录下

hadoop fs –copyFromLocal inputpath

编写代码，生成jar文件并执行

- mapreduce job一定要编译成jar文件才可以在hadoop环境 中执行，通过eclipse工具，可以自动生成一个jar文件。

- 可以在任意一个能够执行eclipse的环境中编写 MapReduce 程序（包括Windows），通过eclipse工具生成jar文件之后 就可以上载到hadoop执行所需的Linux环境中，通过下面的 命令可以执行所编写的程序

- $hadoop jar sum.jar /user/input /user/output

程序代码与分析(程序头部与Map函数)

```
public static class SumMap extends Mapper<Object, Text, Text, IntWritable>{

    private Text SUM=new Text("SUM");
    private IntWritable number=new IntWritable();
    public void map(Object key, Text value, Context context)
        throws IOException, InterruptedException
    {
        StringTokenizer tokenzer=new StringTokenizer(value.toString().trim());
        while(tokenzer.hasMoreTokens()){
            number.set(Integer.parseInt(tokenzer.nextToken()));
            SUM.set(key.toString());
            context.write(SUM, number);
        }
    }
```

Reduce函数

```
public static class SumReduce extends Reducer<Text,IntWritable,Text,IntWritable>{

    private IntWritable Sum=new IntWritable();
    public void reduce(Text key, Iterable<IntWritable> values, Context context)
        throws IOException, InterruptedException
       {
        int Sume=0;
        for(IntWritable aa:values)
        {
            Sume=Sume+aa.get();
        }
        Sum.set(Sume);
        context.write(key, Sum);
       }
}
```

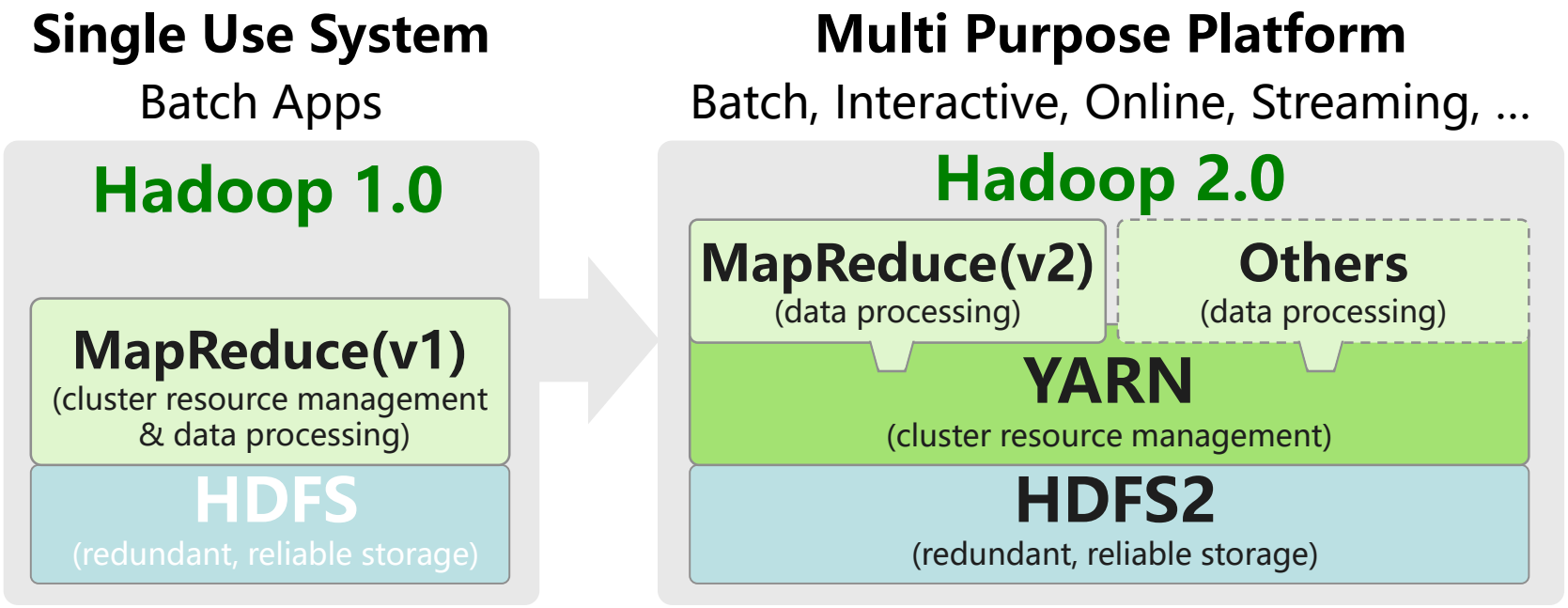
Main函数（驱动过程）

```
public static void main(String[] args) throws Exception
  {
        Configuration _conf = new Configuration();

        Job job = new Job(_conf, "sum");
        job.setJarByClass(Sum.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        job.setMapperClass(SumMap.class);
        job.setReducerClass(SumReduce.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
```

YARN

Hadoop2.0



Hadoop 2.0：

- 由HDFS、MapReduce和YARN三个分支构成
- HDFS：支持NN Federation、HA
- MapReduce：运行在YARN上的MR，编程模型不变
- YARN：资源管理系统
- …

YARN-产生背景

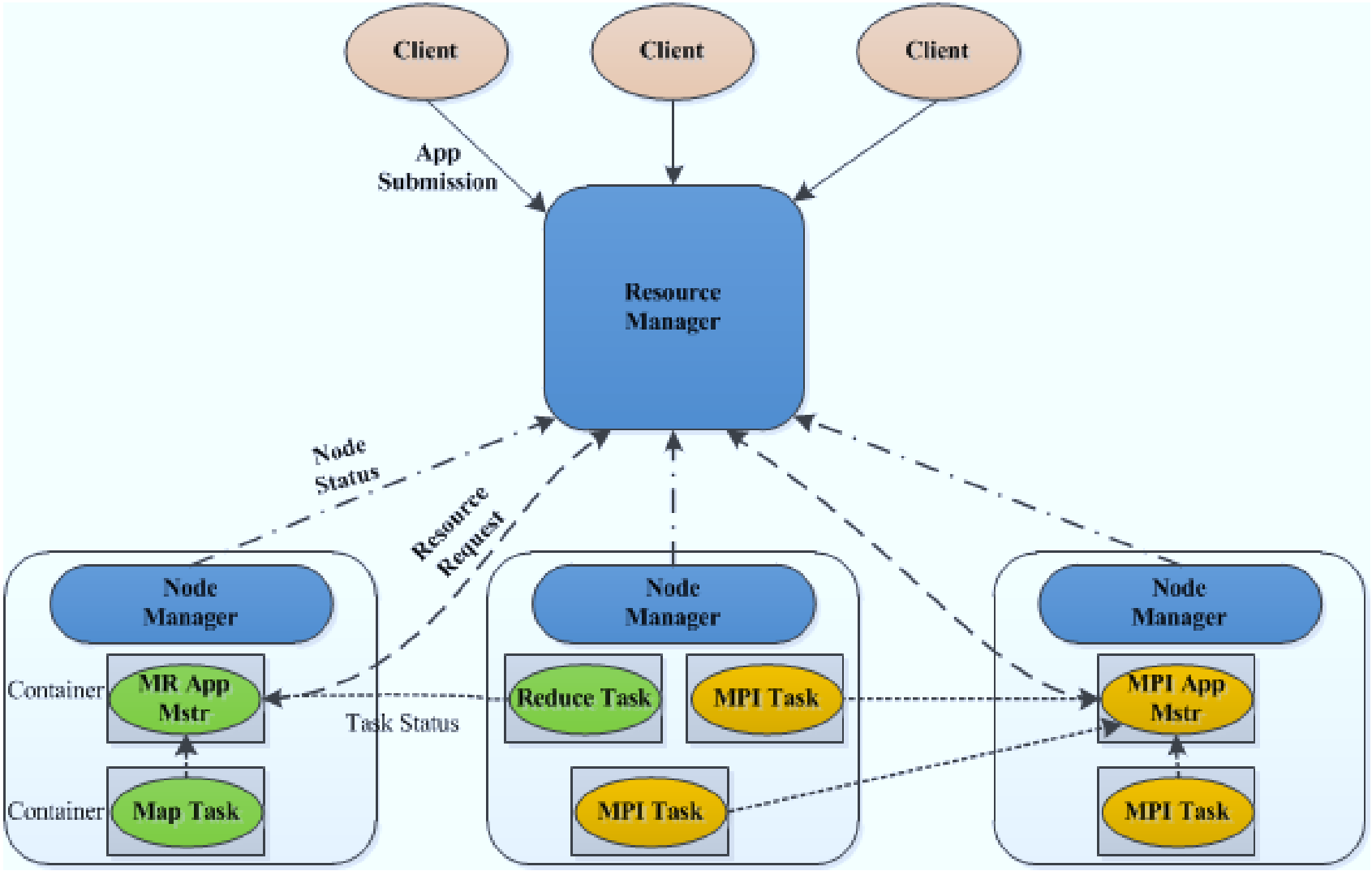
直接源于MRv1在几个方面的无能

- 扩展性差，JobTracker成为瓶颈
- 可靠性差，NameNode单点故障
- 扩展性差，难以支持MR之外的计算
- 资源利用率低

多计算框架各自为战，数据共享困难

- MR：离线计算框架
- Storm：实时计算框架
- Spark：内存计算框架

Yarn 基本框架与组件



YARN-架构及组件

YARN 是Hadoop 2.0 中的资源管理系统，它是一个通用的资源管理模块，可为各类应用程序进行资源管理和调度。

ResourceManager

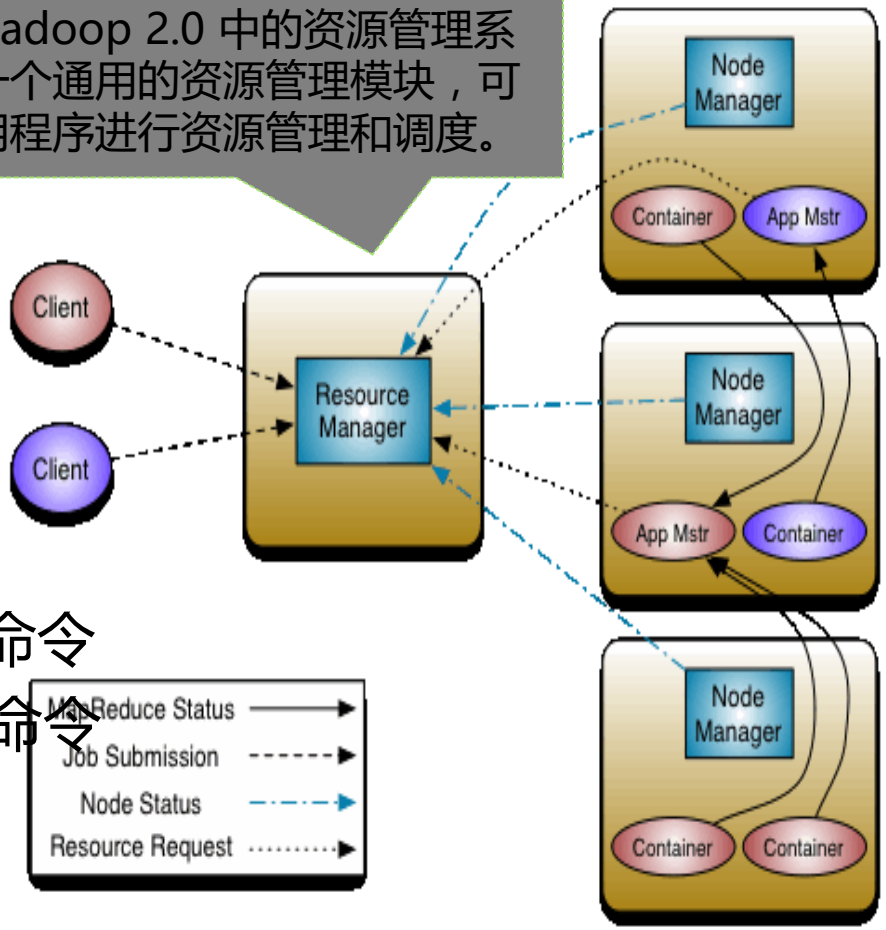
- 处理客户端请求
- 启动/监控ApplicationMaster
- 监控NodeManager
- 资源分配与调度

NodeManager

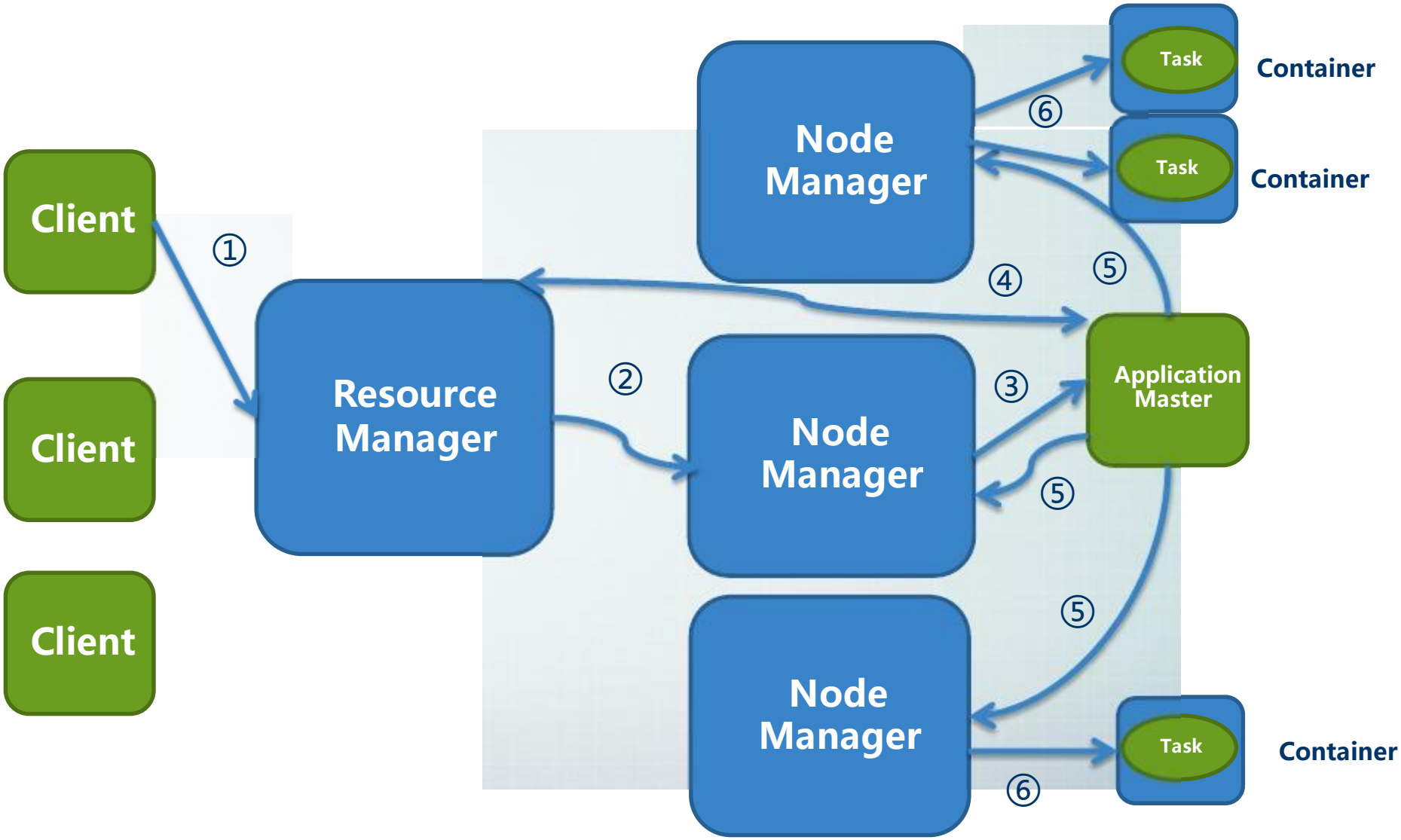
- 单个节点上的资源管理
- 处理来自ResourceManager的命令
- 处理来自ApplicationMaster的命令

ApplicationMaster

- 数据切分
- 为应用程序申请资源，并分配给内部任务
- 任务监控与容错



Yarn作业处理流程


Client
Client
Client
①
Resource Manager
②
Node Manager
Node Manager
Node Manager
③
④
⑤
⑤
⑤
⑥
⑥
Application Master
Task
Container
Task
Container
Task
Container

YARN-MRv2作业处理流程

步骤1　用户向YARN 中提交应用程序， 其中包括ApplicationMaster 程序、启动ApplicationMaster 的命令、用户程序等

步骤2　ResourceManager 为该应用程序分配第一个Container， 并与对应的NodeManager 通信，要求它在这个Container 中启动应用程序的ApplicationMaster

YARN-MRv2作业处理流程

步骤3　ApplicationMaster 首先向ResourceManager 注册， 这样用户可以直接通过ResourceManage 查看应用程序的运行状态，然后它将为各个任务申请资源，并监控它的运行状态，直到运行结束，即重复步骤4~7

步骤4　ApplicationMaster 采用轮询的方式通过RPC 协议向ResourceManager 申请和领取资源

YARN-MRv2作业处理流程

步骤5　一旦ApplicationMaster 申请到资源后，便与对应的NodeManager 通信，要求它启动任务。

步骤6　NodeManager 为任务设置好运行环境（包括环境变量、JAR 包、二进制程序等）后，将任务启动命令写到一个脚本中，并通过运行该脚本启动任务。

YARN-MRv2作业处理流程

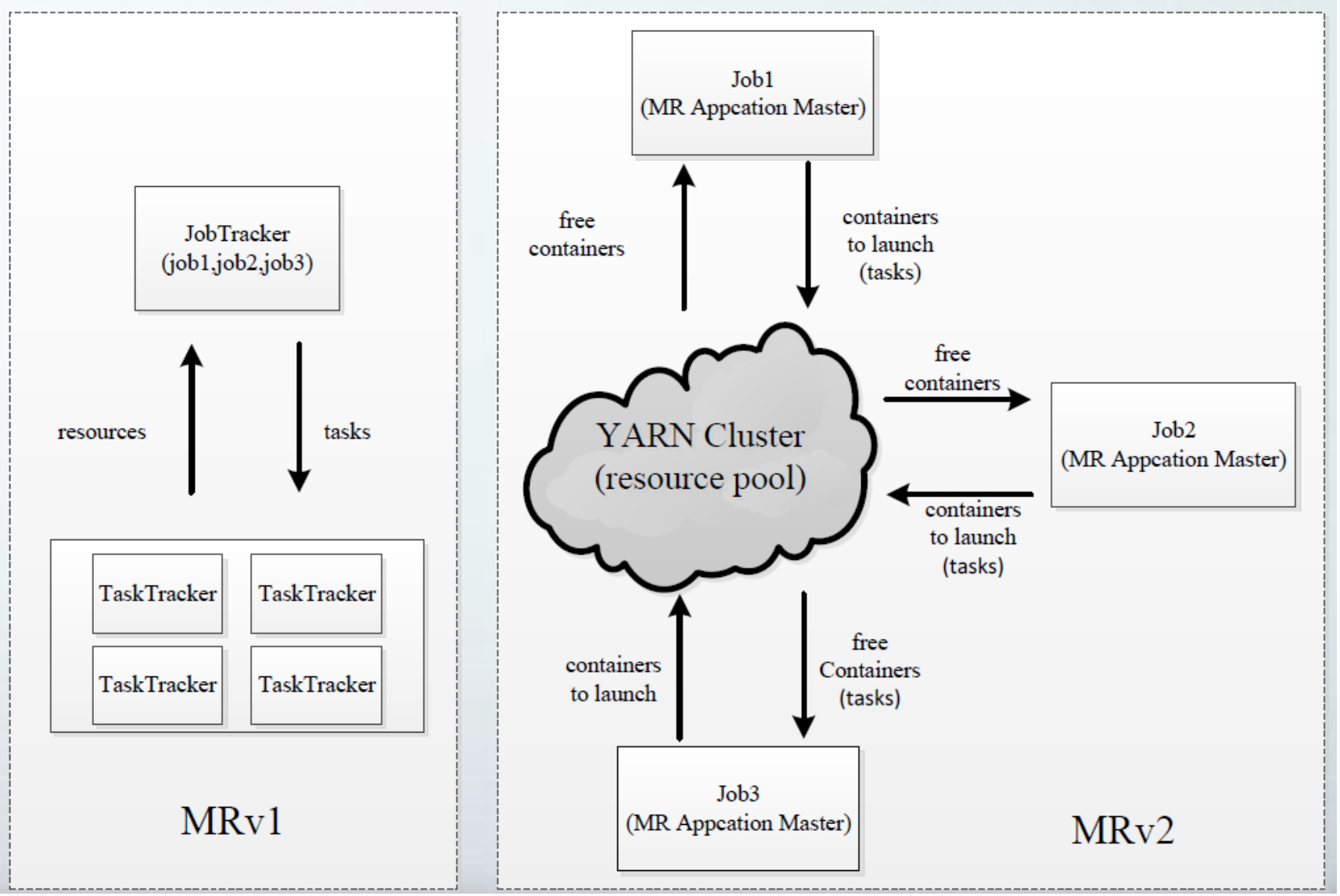
步骤7　各个任务通过某个RPC 协议向ApplicationMaster 汇报自己的状态和进度，以让ApplicationMaster 随时掌握各个任务的运行状态，从而可以在任务失败时重新启动任务。在应用程序运行过程中，用户可随时通过RPC 向ApplicationMaster 查询应用程序的当前运行状态。

步骤8　应用程序运行完成后，ApplicationMaster 向ResourceManager 注销并关闭自己。

Yarn 调度框架

- 双层调度框架
  - ✓ **RM**将资源分配给**AM**
  - ✓ **AM**将资源进一步分配给各个**Task**
- 基于资源预留的调度策略
  - ✓ 资源不够时，会为**Task**预留，直到资源充足
  - ✓ 与“**all or nothing**”策略不同（**Apache Mesos**）

Yarn调度框架


JobTracker
(job1,job2,job3)
resources
tasks
TaskTracker
TaskTracker
TaskTracker
TaskTracker
MRv1
Job1
(MR Appcation Master)
free
containers
containers
to launch
(tasks)
YARN Cluster
(resource pool)
free
containers
Job2
(MR Appcation Master)
containers
to launch
(tasks)
containers
to launch
free
Containers
(tasks)
Job3
(MR Appcation Master)
MRv2

YARN资源调度器

核心概念:Container,实现资源动态管理和分配

- ✓支持CPU和内存两种资源调度；
- ✓Resource-centric Scheduling（NOT Task-centric Scheduling）

调度场景	是否支持	应用举例
任意K个Container（位置没有要求）	√	MR、Tez等
K个独占K个节点的container	√	HBase、MPI等
K个来自同一机架的container	×	Storm等
Container可绑定到固定CPU上	×	MPI等

YARN资源调度器

- 支持CPU和内存两种资源调度方式，允许配置每个节点、每个任务可用的CPU和内存资源总量

```
15.  // Priority for worker containers - priorities are intra-application
16.  Priority priority = Records.newRecord(Priority.class);
17.  priority.setPriority(0);
18.
19.  // Resource requirements for worker containers
20.  Resource capability = Records.newRecord(Resource.class);
21.  capability.setMemory(128);
22.  capability.setVirtualCores(1);
23.
24.  // Make container requests to ResourceManager
25.  for (int i = 0; i < n; ++i) {
26.    ContainerRequest containerAsk = new ContainerRequest(capability, null,
27.    rmClient.addContainerRequest(containerAsk);
28.  }
```

Setup requirements for worker containers

Make resource requests to ResourceManager

- Mesos等资源管理软件

YARN资源调度器

- 提供多种资源调度器
  - ✓ FIFO
  - ✓ Fair Scheduler
  - ✓ Capacity Scheduler
- 多租户资源调度器
  - ✓ 支持资源按比例分配
  - ✓ 支持层级队列划分方式
  - ✓ 支持资源抢占

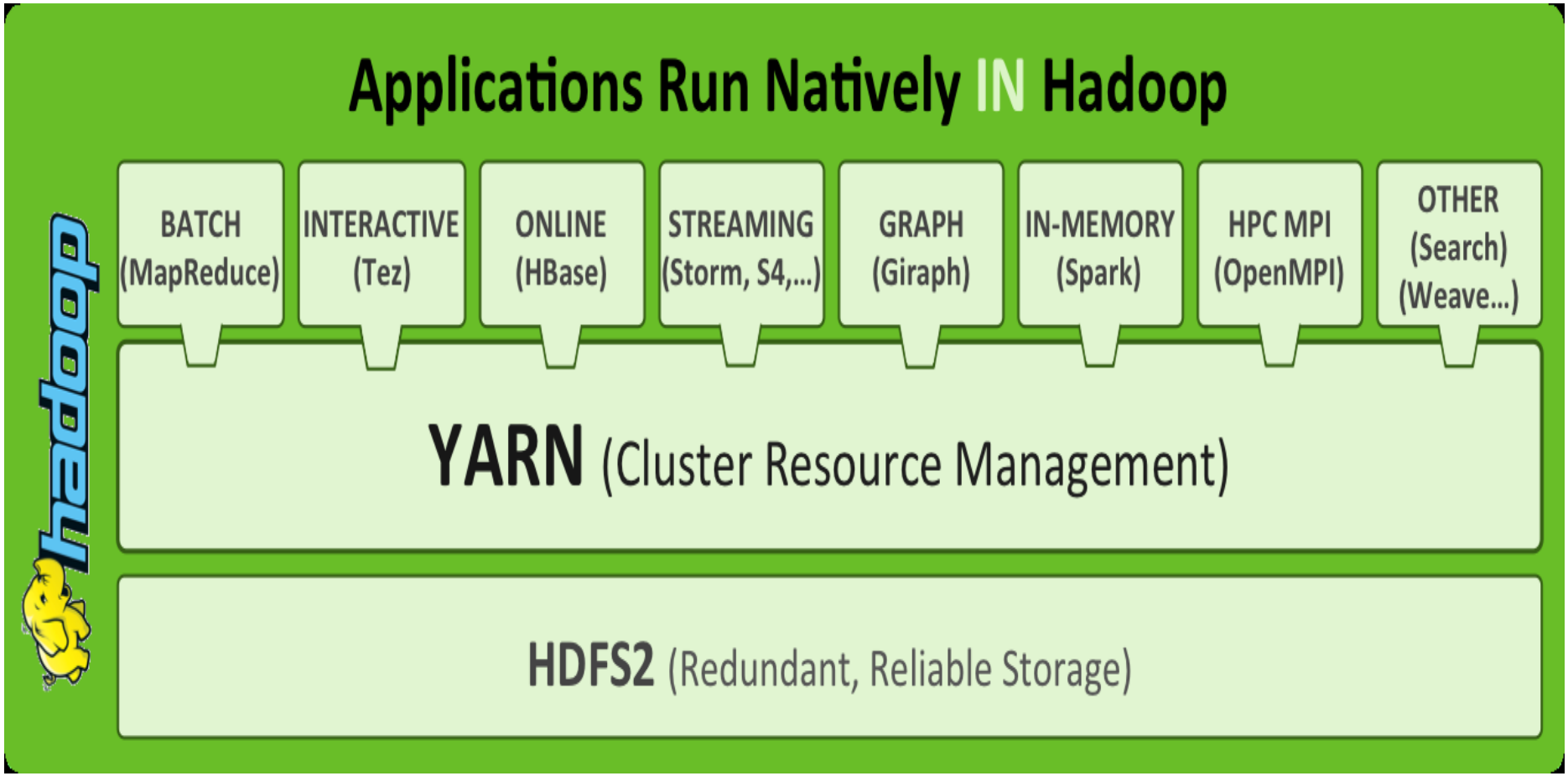
YARN资源隔离方案

- 支持内存和**CPU**两种资源隔离
  - ✓ 内存是一种“决定生死”的资源
  - ✓ **CPU**是一种“影响快慢”的资源
- 内存隔离
  - ✓ 基于线程监控的方案
  - ✓ 基于**Cgroups**的方案
- CPU隔离
  - ✓ 默认不对**CPU**资源进行隔离
  - ✓ 基于**Cgroups**的方案

YARN资源调度语义

- 支持的语义
  - ✓ 请求某个特定节点/机架上的特定资源量
  - ✓ 将某些节点加入（或移除）黑名单，不再为自己分配这些节点上的资源
  - ✓ 请求归还某些资源
- 不支持的语义
  - ✓ 请求任意节点/机架上的特定资源量
  - ✓ 请求一组或几组符合某种特质的资源
  - ✓ 超细粒度资源
  - ✓ 动态调整**Container**资源

以YARN为核心的生态系统



运行在YARN上的计算框架

- 离线计算框架：**MapReduce**
- **DAG**计算框架：**Tez**
- 流式计算框架：**Storm**
- 内存计算框架：**Spark**
- 图计算框架：**Giraph**、**GraphLib**

其他Framework On YARN

- **Hoya：HBase on YARN**

https://github.com/hortonworks/hoya/

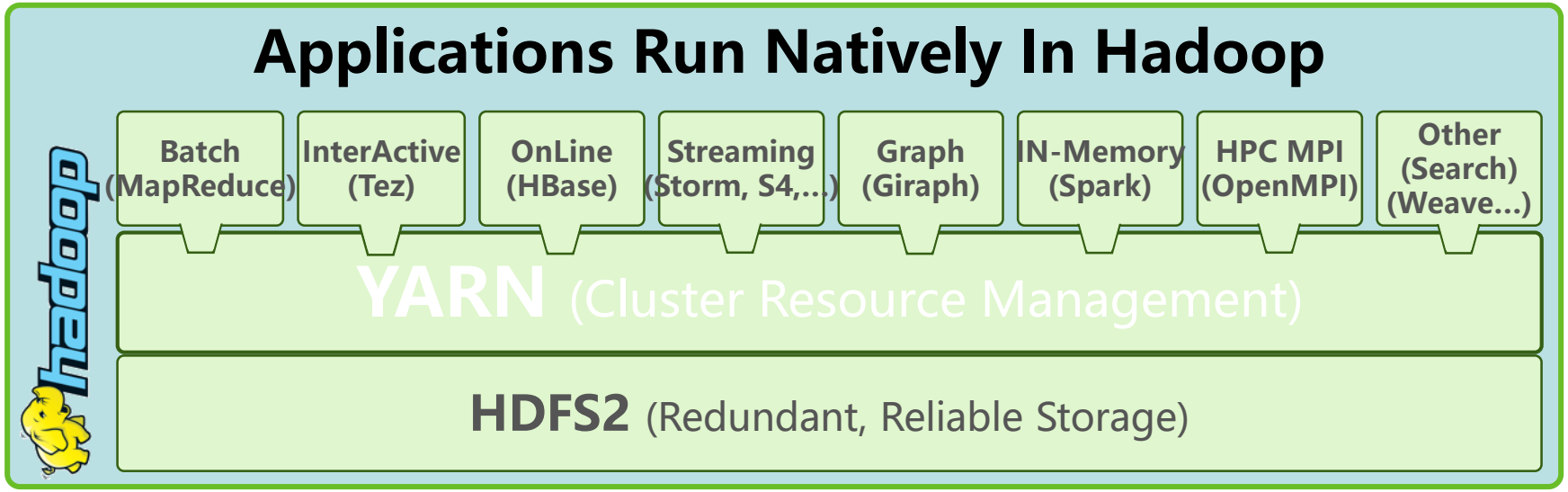
- **LLAMA：Impala On YARN**

http://cloudera.github.io/llama/

- **Kafka On YARN**

https://github.com/kkasravi/kafka-yarn

YARN带来的好处



运行在YARN上带来的好处 ：

- 一个集群部署多个版本
- 计算资源按需伸缩
- 不同负载应用混搭，集群利用率高
- 共享底层存储，避免数据跨集群迁移

YARN自身的完善

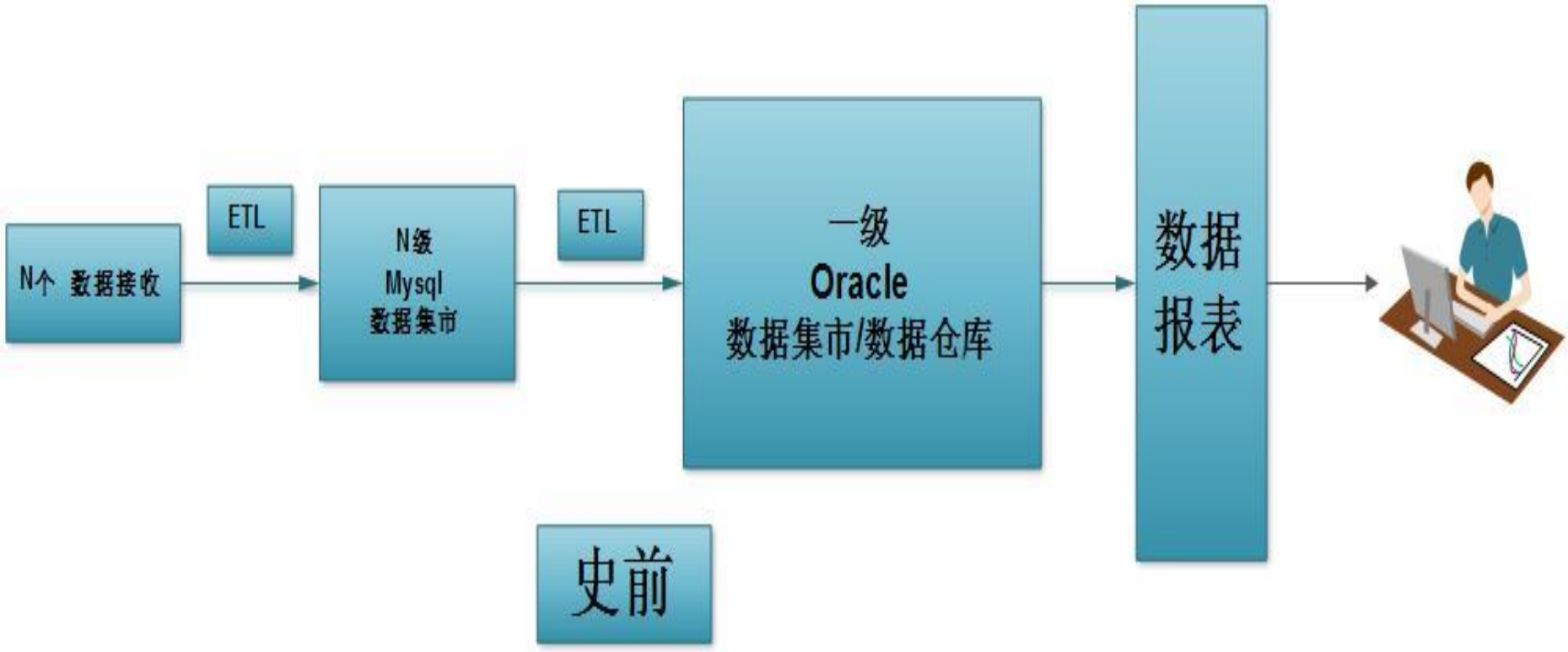
- 调度框架的完善:**YARN-896**
  - ✓ 支持更多的资源类型（网络、磁盘等）
  - ✓ 支持更多的调度语义
  - ✓ Storm在线升级等
- 长作业的在线升级
  - ✓ Container资源动态调整：YARN-1197
- 容错机制
  - ✓ ResourceManager自身容错
  - ✓ NodeManager宕掉，任务不受影响
  - ✓ ApplicationMaster个性化容错

精彩案例

案例 1：数据仓库构建

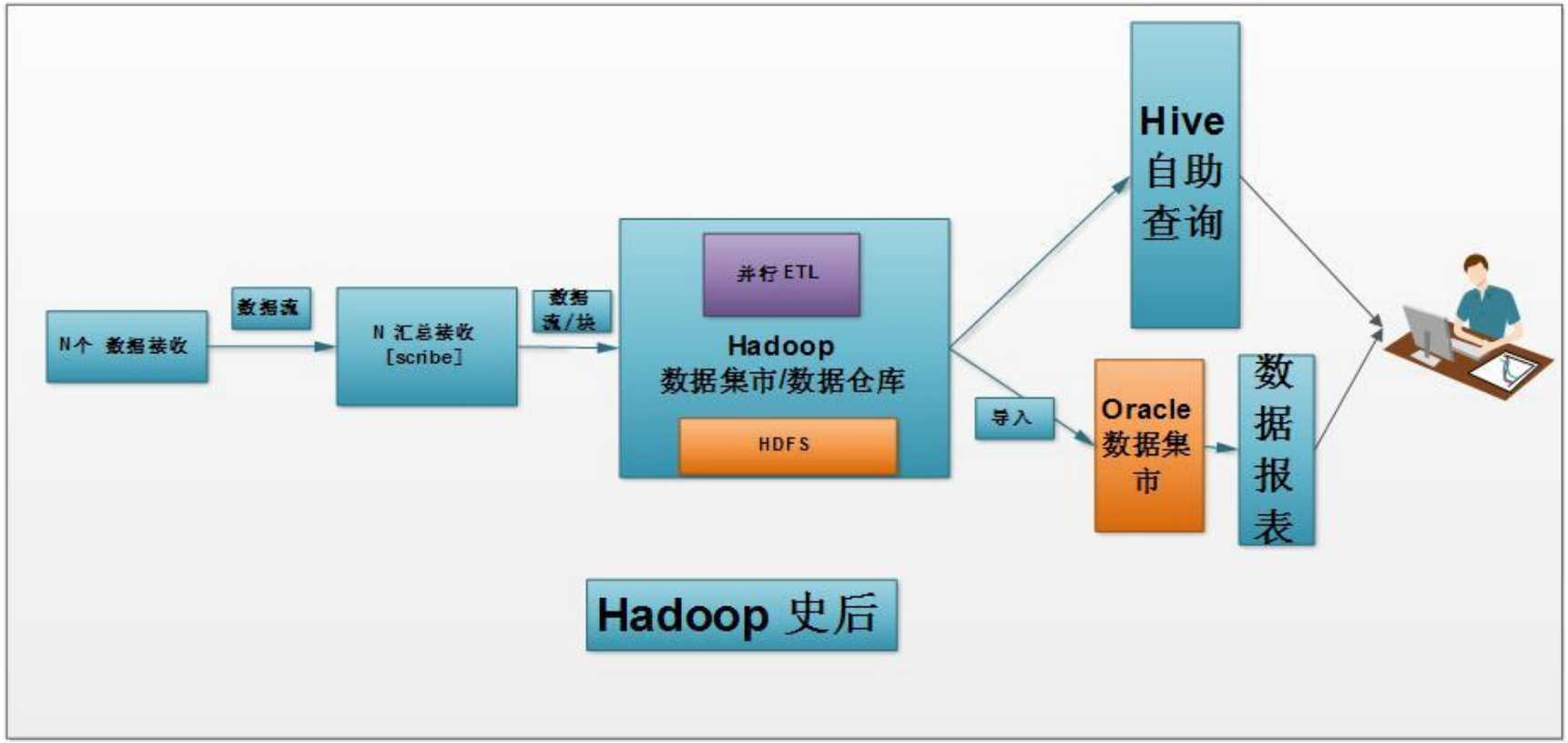
Hadoop 前的数据仓库流程

Perl/Python、Shell、awk



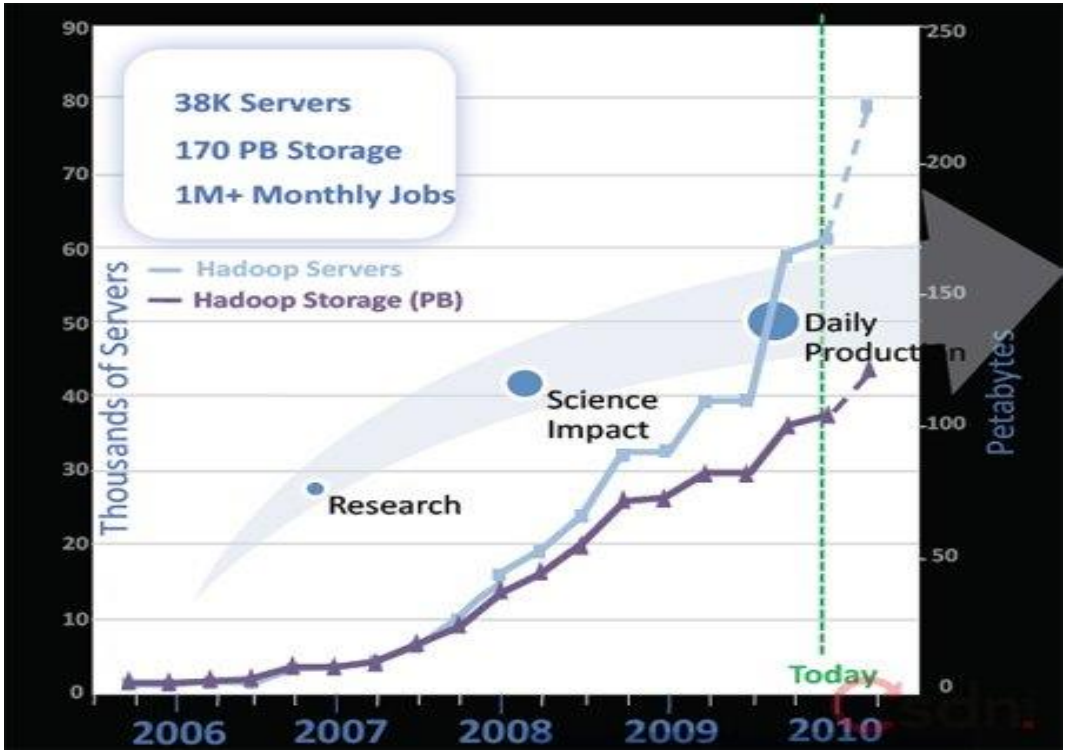
精彩案例

- Hadoop后的数据仓库流程
  - ✓ Hive、Pig、MR、工作流



案例2：Yahoo

- 2010年10月，Yahoo拥有>38000台服务器，有超过4000个以上的服务器集群，数据总量达到了170PB，每日的数据增量在10TB以上



Yahoo应用揭秘

- Yahoo的Hadoop应用包含有搜索、日志处理（Analytics, Reporting, Buzz）、用户建模、内容优化，垃圾邮件过滤器以及广告计算等。
- 网络分析。Yahoo目前有超过100亿个网页，1PB网页数据内容，2万亿条链接，每日面临这300TB的数据输出。在应用Hadoop前，实施这一过程最少1个月的时间，但应用后仅需要2天

Yahoo应用揭秘

- Yahoo搜索。服务器上保留有用户3年来的搜索记录，这个数据是由超过 10TB。如果数据重整，之前需要1个月的时间进行处理，而在有了Hadoop后仅仅需要30分钟
- Hadoop也非万能，它采用Java实现，Java的IO处理虽然没有性能瓶颈，但是对于CPU密集型的任务是一个麻烦， 因此，有些算法效率不会提高很多。

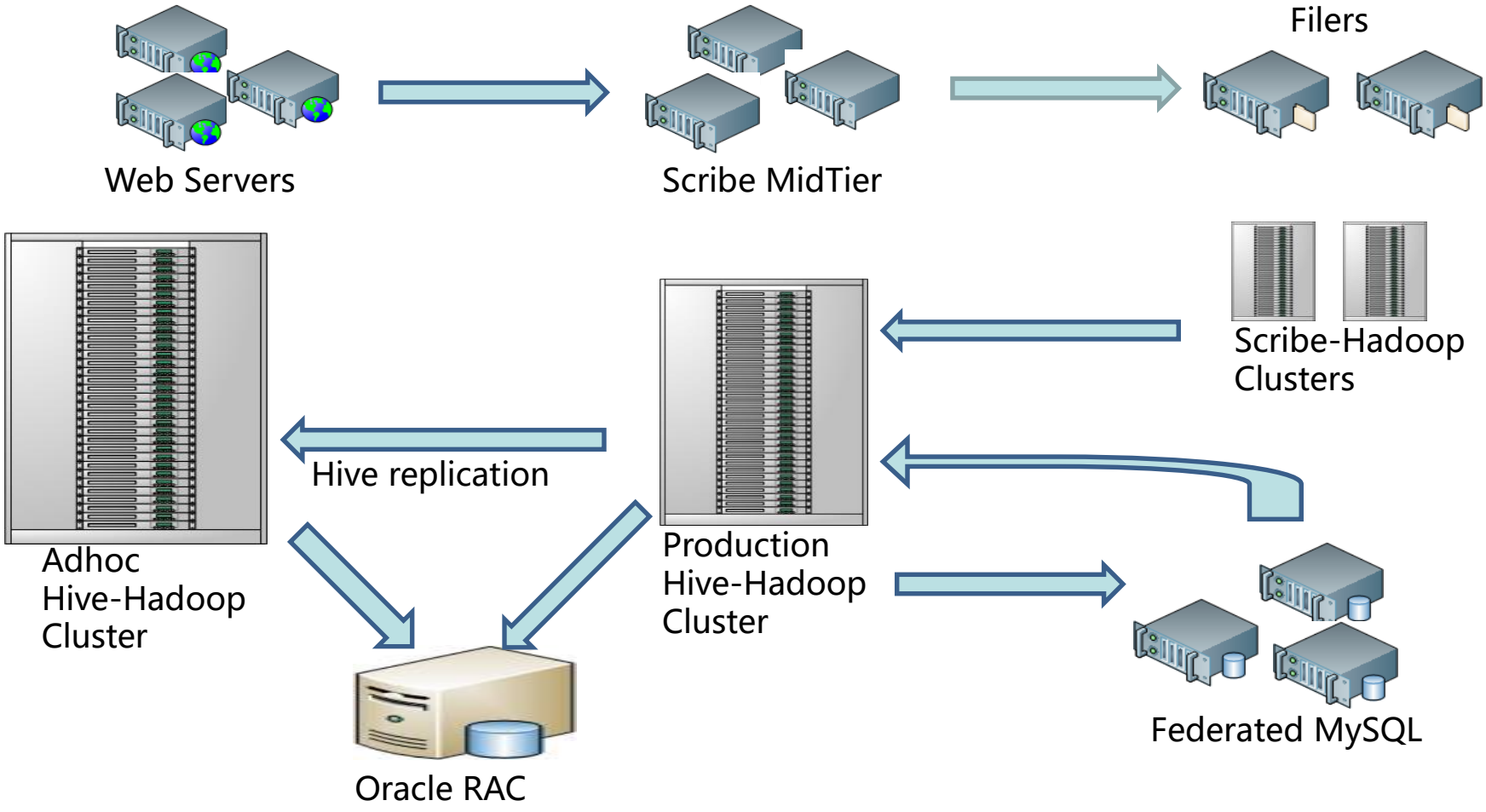
案例3：中国移动

- 中国移动大云
- 中国移动通信研究院院长黄晓庆表示，对中国移动大云来说，Hadoop代表了最核心层面的东西，为了将云计算的服务变成一个更容易编程的使用，中国移动把一些特别重要的功能都设计成完全可以进行规模化使用的一些核心资源。比如，为一些传统数据库（比如SQL)提供了非常方便的接口。

案例3：中国移动

- 在中国移动内部，Hadoop已经成为一个重要的数据挖掘工具，中国移动已经把Hadoop的群用来进行闭环的数据挖掘，和传统算法的数据挖掘性能相比，有很大的提升，而且成本非常低。

案例4：Facebook



Facebook数据仓库

- Hadoop基础平台
- 自动化ETL
- 元数据发现
- 日志查询(HIVE)
- 特殊工作流和扩展
- 报表工具
- 监控和报警

Facebook任务统计

- **每天任务统计**

✓ 10TB 压缩数据新增/天

✓ 135TB 扫描数据总量/天

✓ 7500+ 任务/天

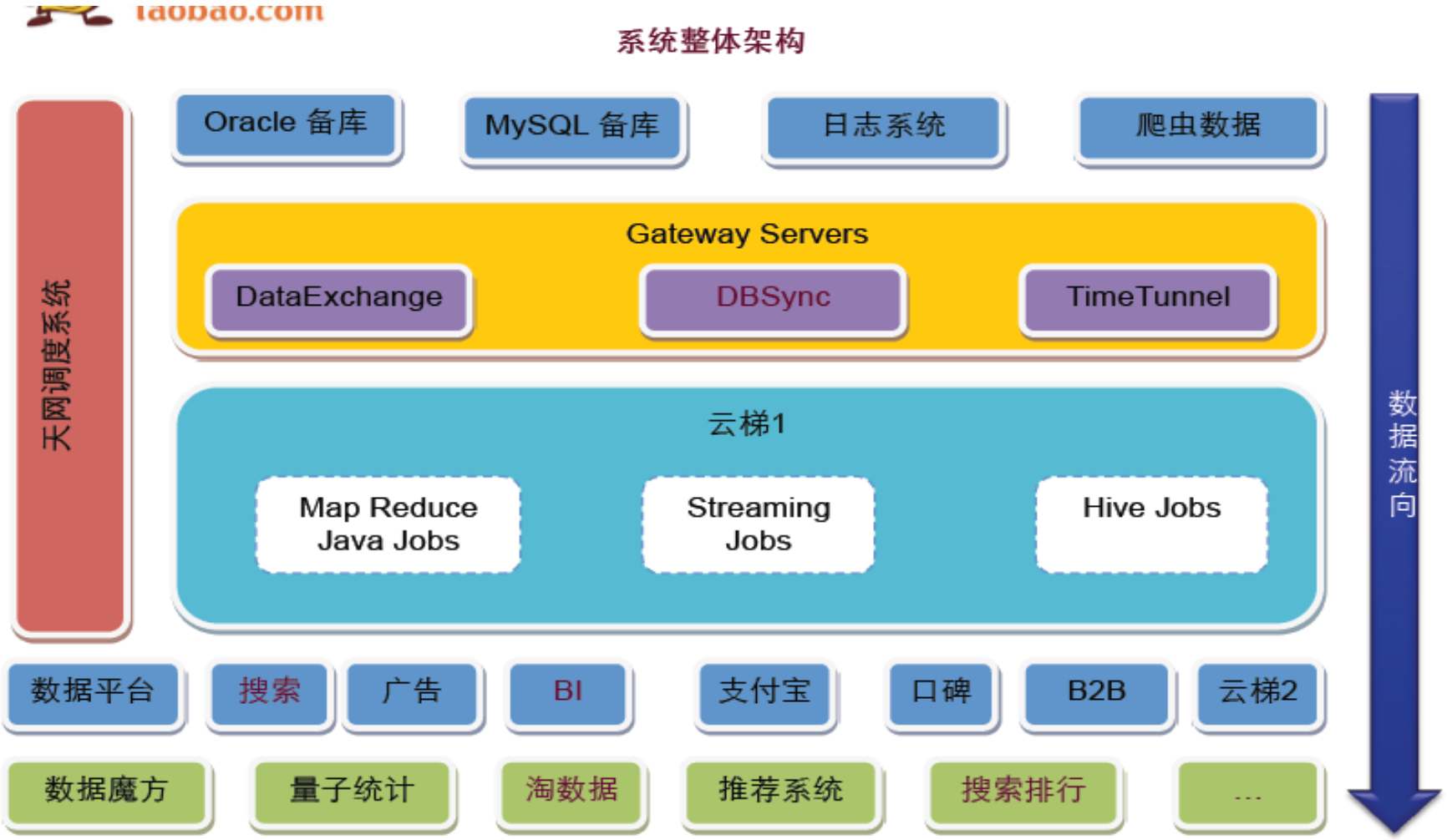
✓ 80K 计算小时/天

✓ 95% Hadoop任务使用Hive

Facebook应用

- 报表
  - ✓ 天/周的展示、点击统计汇总
  - ✓ 用户参与度计算
  - ✓ 战略决策服务
- Ad hoc分析
- 机器学习（ad方向）

案例5：Taobao


taobao.com
系统整体架构
Oracle 备库
MySQL 备库
日志系统
爬虫数据
Gateway Servers
DataExchange
DBSync
TimeTunnel
云梯1
Map Reduce Java Jobs
Streaming Jobs
Hive Jobs
天网调度系统
数据流向
数据平台
搜索
广告
BI
支付宝
口碑
B2B
云梯2
数据魔方
量子统计
淘数据
推荐系统
搜索排行
...

Question