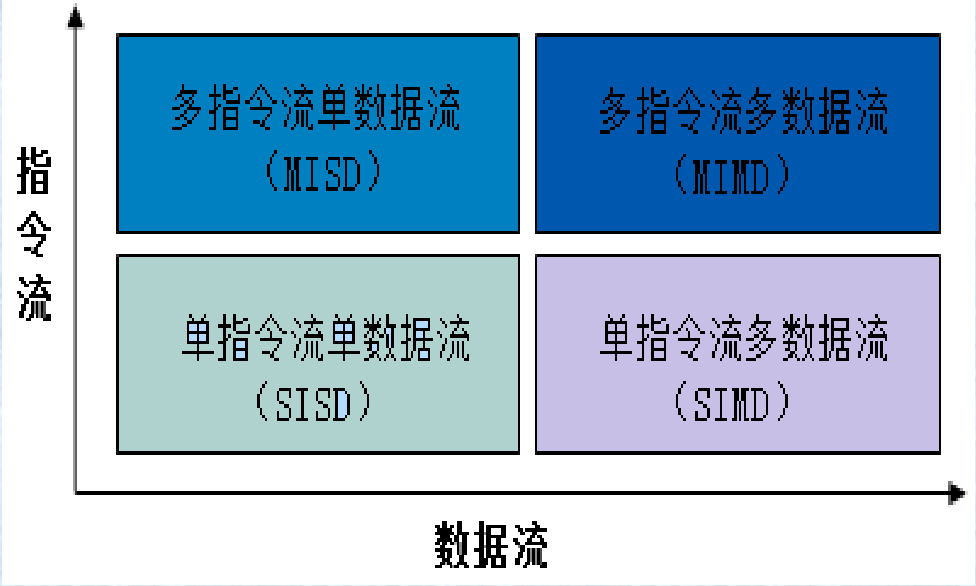
Lecture 14 MapReduce计算模型

- 分布式并行计算系统
- MapReduce计算架构
- MapReduce计算流程
- 实际算例

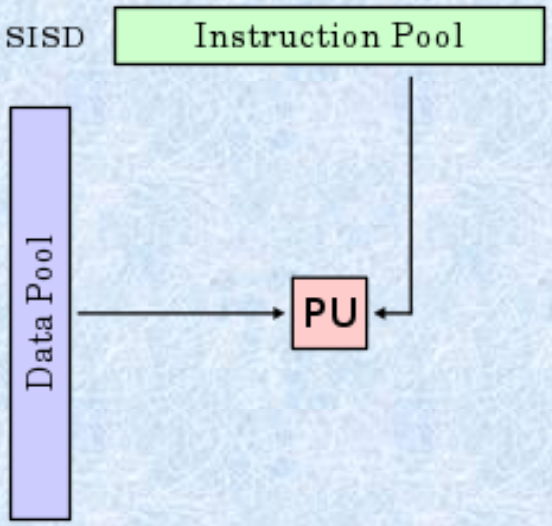
1. 分布式并行计算系统

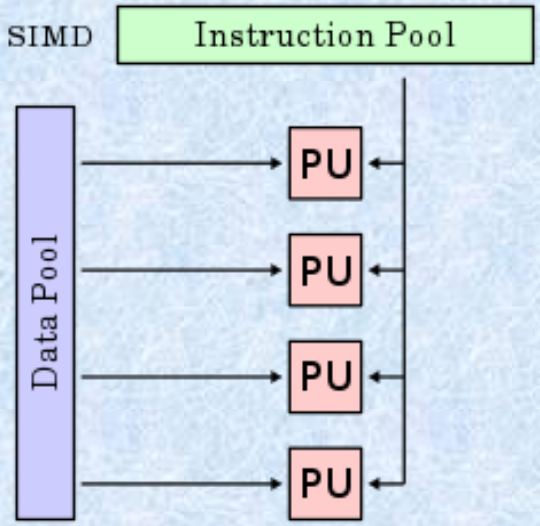
Flynn并行计算模型：按指令流和数据流划分

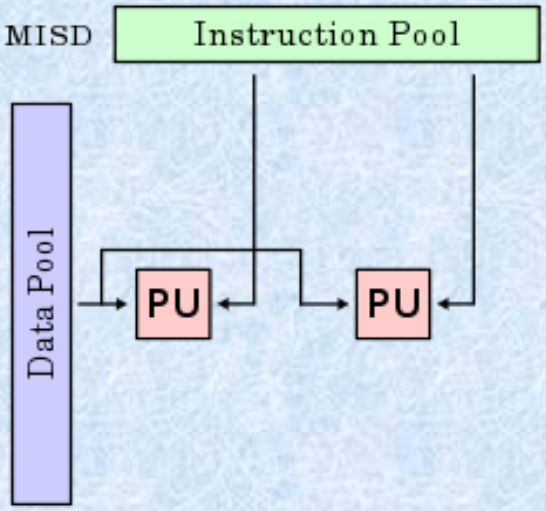
- 单指令流单数据流（SISD）
- 单指令流多数据流（SIMD）
- 多指令流单数据流（MISD）
- 多指令流多数据流（MIMD）

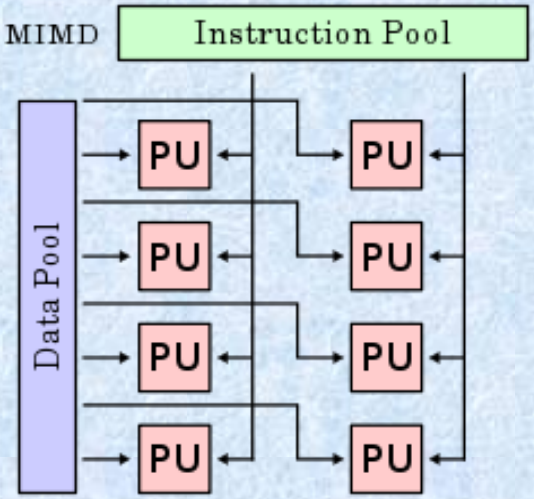


大数据计算技术
Big Data Computing Technology







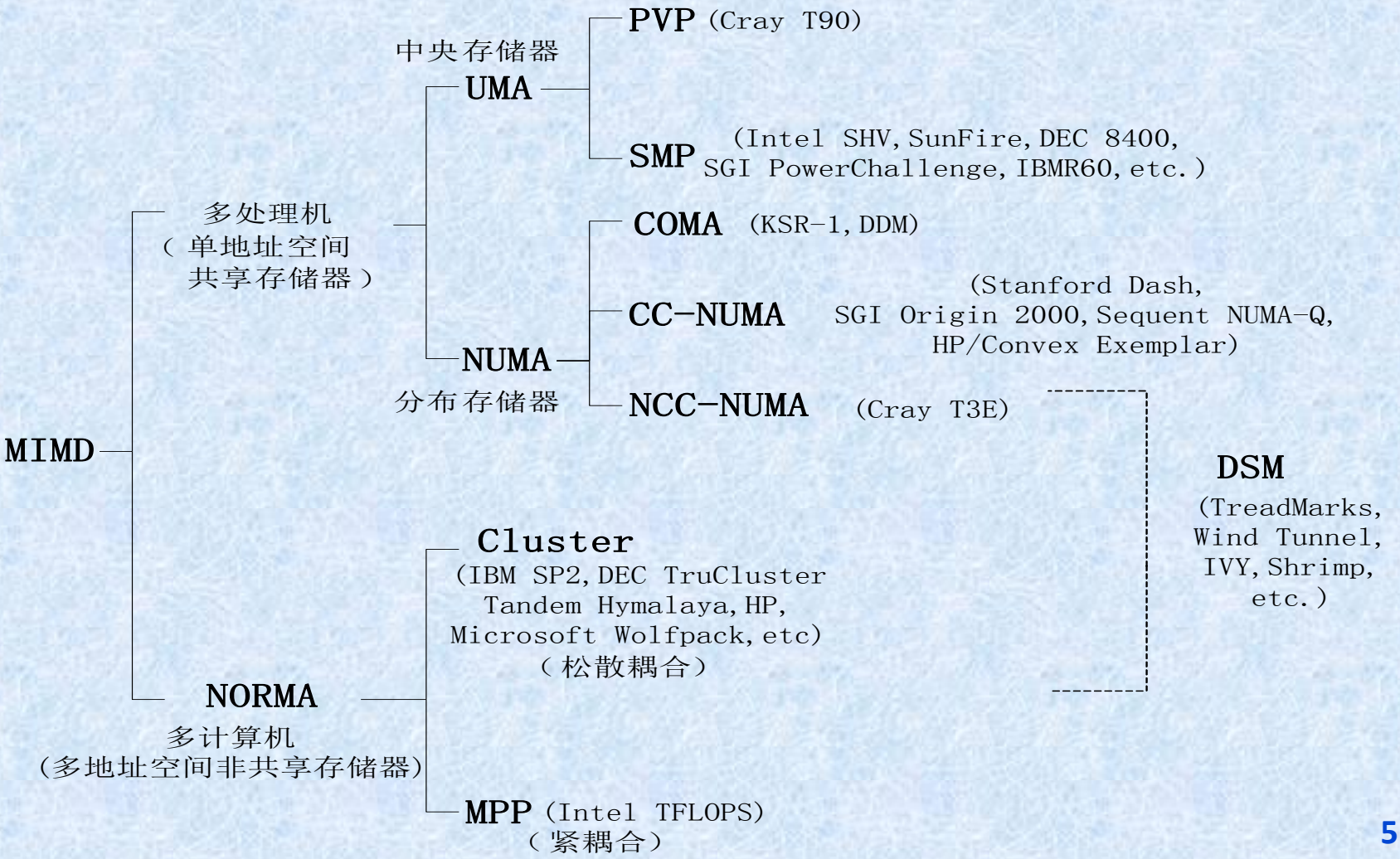


多指令流多数据流（MIMD）模型

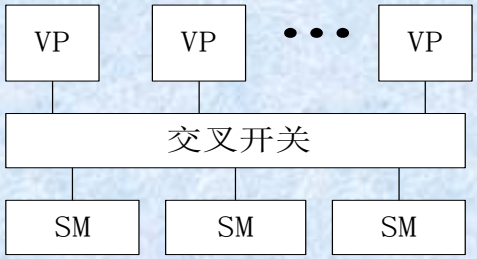
按照处理器是否共享内存划分：

- 多处理器共享内存机器
  - UMA架构
  - NUMA架构
- 多计算机独立内存体系
  - MPP架构
  - 集群架构
- 归属于MIMD体系的计算机架构
  - 并行向量处理机、对称多处理机、大规模并行处理机、工作站机群、分布式共享存储处理机

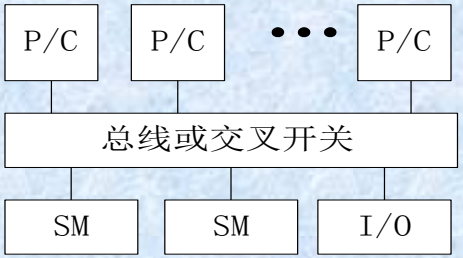
多指令流多数据流（MIMD）模型



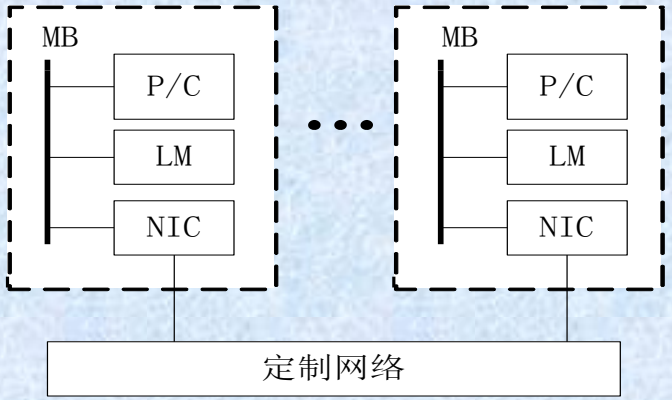
MIMD系统架构



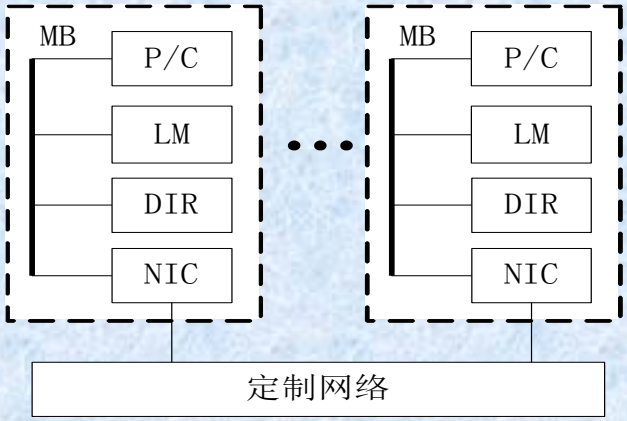
(a) PVP



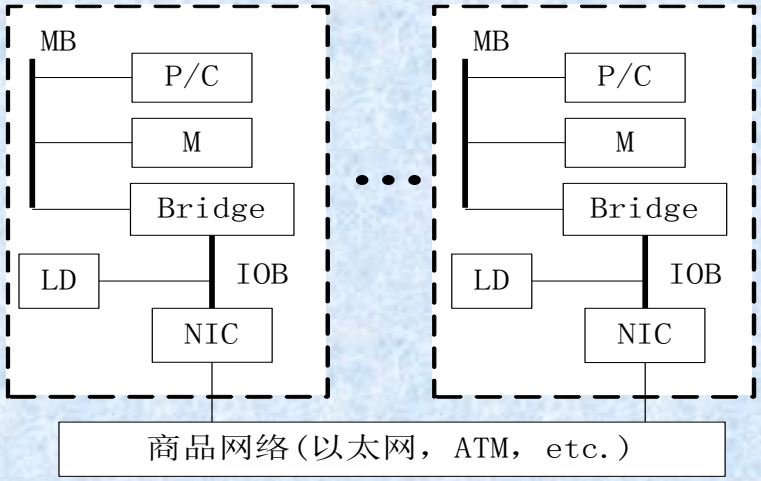
(b) SMP



(c) MPP



(d) DSM

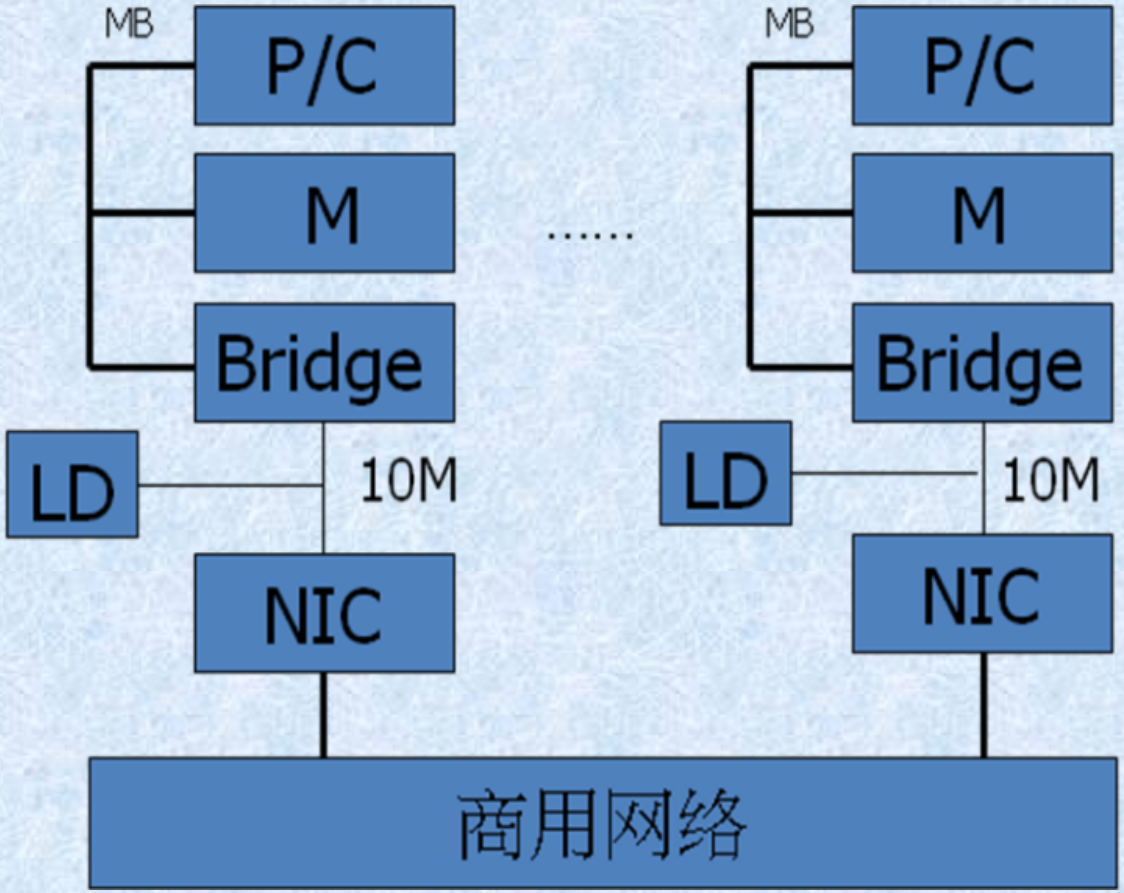


(e) COW

集群（Cluster）计算架构

- 集群由多个独立的计算机（服务器或工作站，称为集群节点）通过高速局域网连接在一起，每个节点拥有独立的内存和磁盘，一个节点的CPU不能直接访问另外一个节点的内存空间；
- 每个节点拥有独立的O/S和文件系统；
- 节点间采用消息传递（message passing）方式进行数据交换，使用如MPI，PVM等中间件；
- 在节点内部（本地机器上）支持共享内存的并行计算模式，可使用OpenMP、pthreads等编程模型；
- 需要一系列集群平台软件来支持整个系统的管理与运行，包括集群系统管理软件（如IBM的CSM、xCat等），消息中间件（如MPI，PVM等），作业管理与调度系统（如LSF、PBS，LoadLeveler）；并行文件系统（如PVFS、GPFS等；
- 系统吞吐量大、可靠性高、可扩展性好、计算性价比好（cost-effective）；

集群（Cluster）计算架构



2. MapReduce计算架构

- 计算原理
- 软件体系
- 输入数据格式

MapReduce介绍

MapReduce是面向大规模数据并行处理的：

- **基于集群的高性能并行计算平台(Cluster Infrastructure)**

  允许用市场上现成的普通PC或性能较高的刀架或机架式服务器，构成一个包含数千个节点的分布式并行计算集群。

- **并行程序开发与运行框架(Software Framework)**

  提供了一个庞大但设计精良的并行计算软件构架，能自动完成计算任务的并行化处理，自动划分计算数据和计算任务，在集群节点上自动分配和执行子任务以及收集计算结果，将数据分布存储、数据通信、容错处理等并行计算中的很多复杂细节交由系统负责处理，大大减少了软件开发人员的负担。

- **并行程序设计模型与方法(Programming Model & Methodology)**

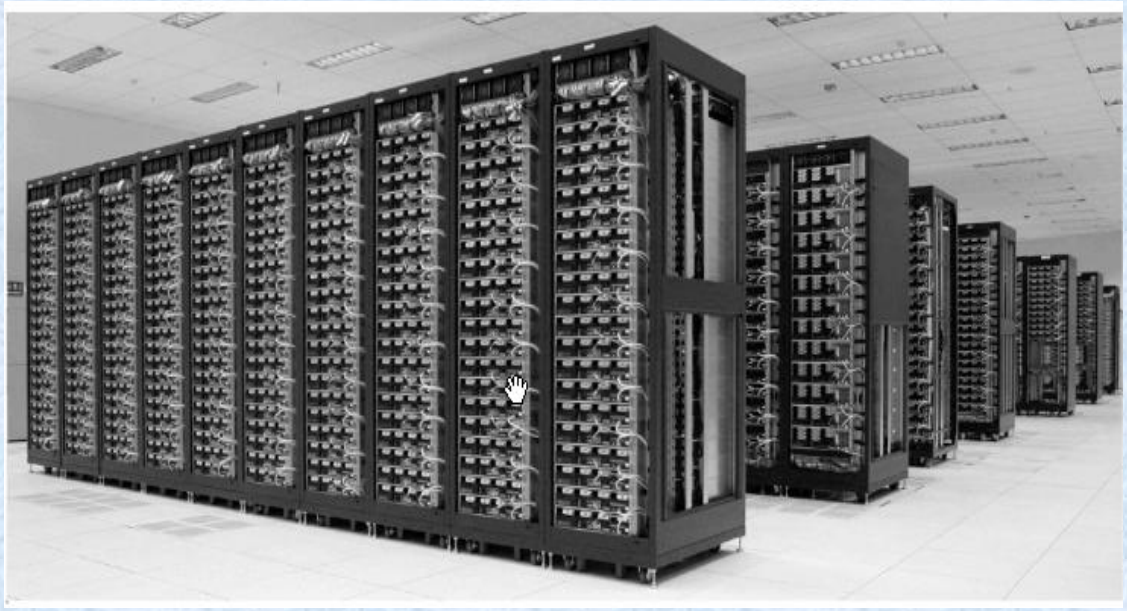
  借助于函数式语言中的设计思想，提供了一种简便的并行程序设计方法，用Map和Reduce两个函数编程实现基本的并行计算任务，提供了完整的并行编程接口，完成大规模数据处理。

Google MapReduce

- 2004年，Google在OSDI国际会议上发表了一篇论文“MapReduce: Simplified Data Processing on Large Clusters”，公布了MapReduce的基本原理和主要设计思想。
- Google公司用MapReduce重新改写了整个搜索引擎中的Web文档索引处理。
- 自MapReduce发明后，Google大量用于各种海量数据处理，目前Google内部有7千以上的程序基于MapReduce实现。
- Google目前在全球的数十个数据中心使用了百万台以上的服务器构成其强大的Web搜索和海量数据并行计算平台,支撑其搜索引擎、Gmail、Google Map、Google Earth、以及其云计算平台AppEngine的大型应用服务需求。
- Google可提供超过80亿网页和10亿张图片的检索索引，每天处理2亿次以上检索请求，平均每个检索耗时0.5秒；每个搜索请求背后有上千个服务器同时进行检索计算和服务。

Hadoop MapReduce

- 在Google发表了文章后，Doug Cutting，2004年，开源项目Lucene(搜索索引程序库)和Nutch(搜索引擎)的创始人，发现MapReduce正是其所需要的解决大规模分布数据处理的重要技术，因而模仿Google MapReduce，基于Java设计出了称为Hadoop的开源MapReduce，该项目成为Apache下最重要项目。



MapReduce的三个基本思想

- **如何对付大数据：分而治之**

对相互间不具有计算依赖关系的大数据，实行并行最自然的方式就是分而治之（divide-and-conquer）。

- **上升到抽象模型：Mapper和Reducer**

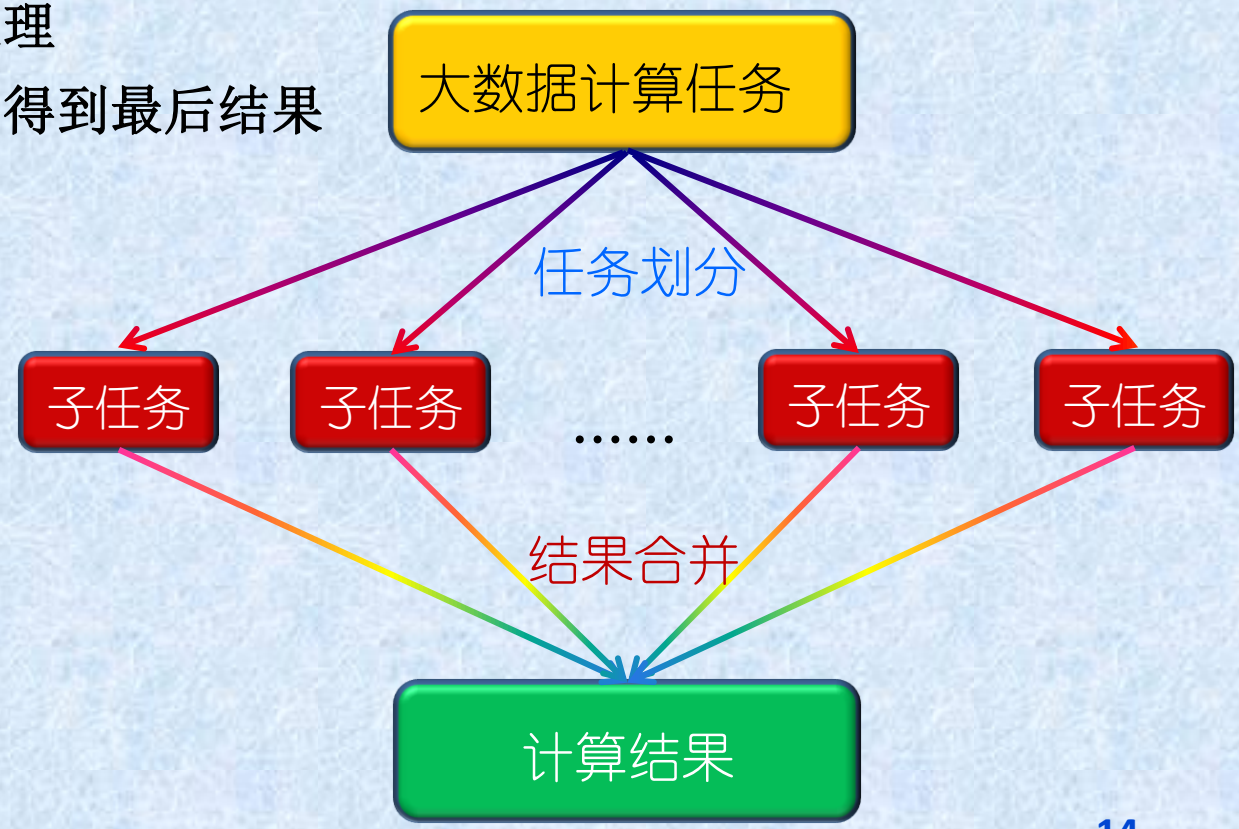
MPI并行计算缺少高层并行编程模型，为了克服这一缺陷，MapReduce借鉴了Lisp函数式语言中的思想，用Map和Reduce两个函数提供了高层的并行编程抽象模型。

- **上升到构架：统一构架，为程序员隐藏系统细节**

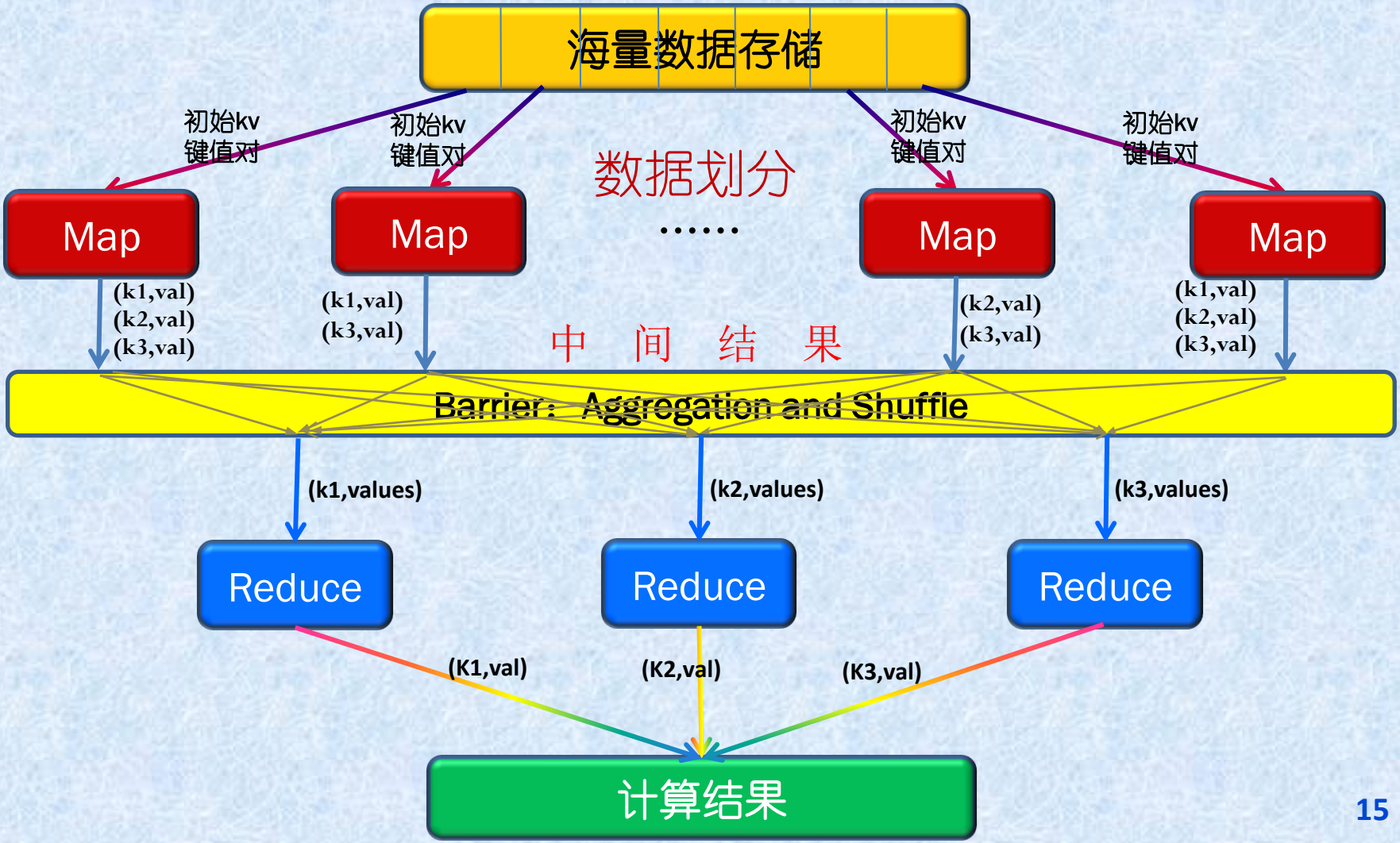
MPI等并行计算缺少统一计算框架支持，程序员需要考虑数据存储、划分、分发、结果收集、错误恢复等诸多细节；为此，MapReduce设计并提供了统一的计算框架，为程序员隐藏了绝大多数系统层面的处理细节。

MapReduce分治法

- 将大数据集划分为小数据集，小数据集划分为更小数据集
- 将最终划分的小数据分布到集群节点上
- 以并行方式完成计算处理
- 将计算结果递归融汇，得到最后结果



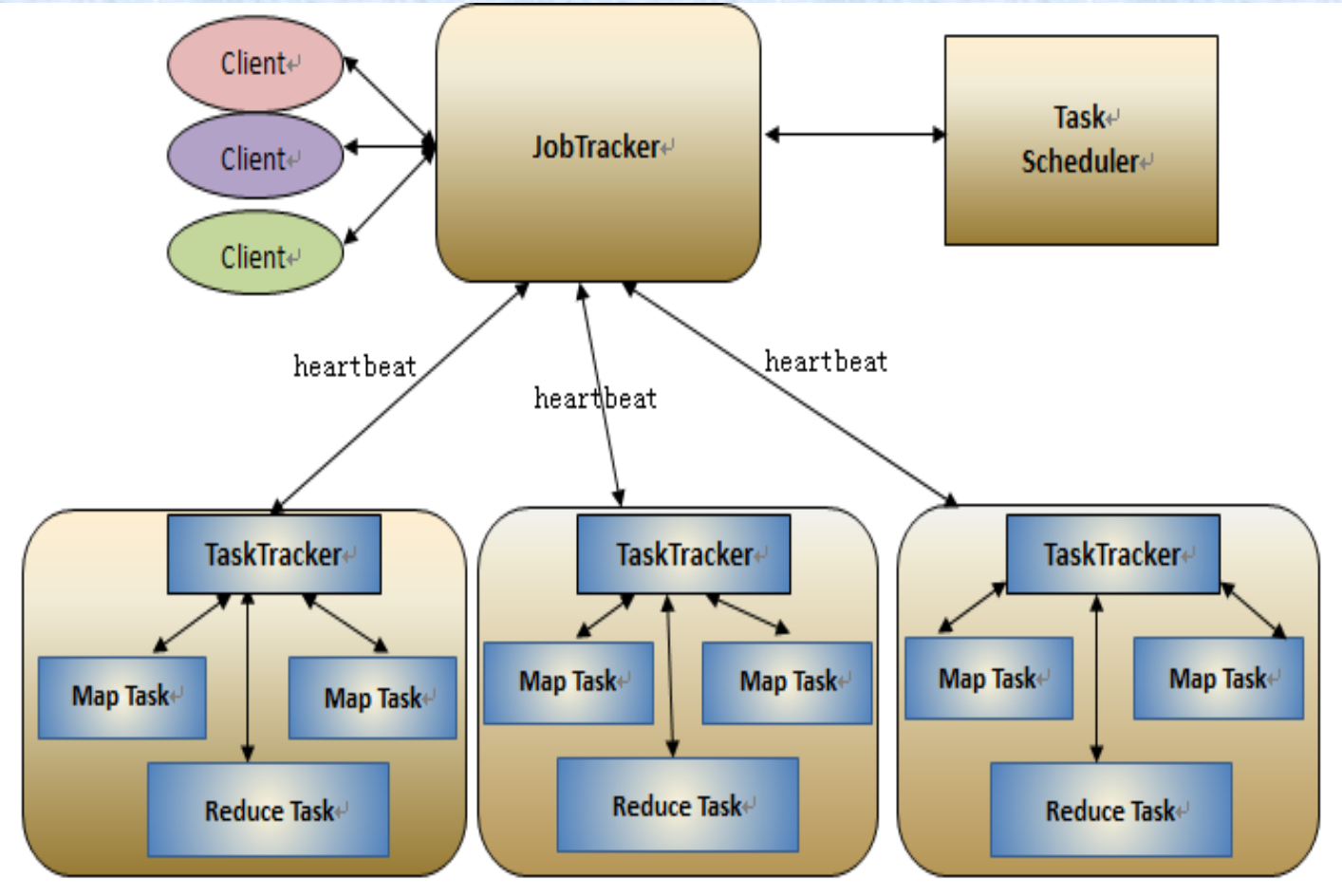
基于Map/Reduce的并行计算模型



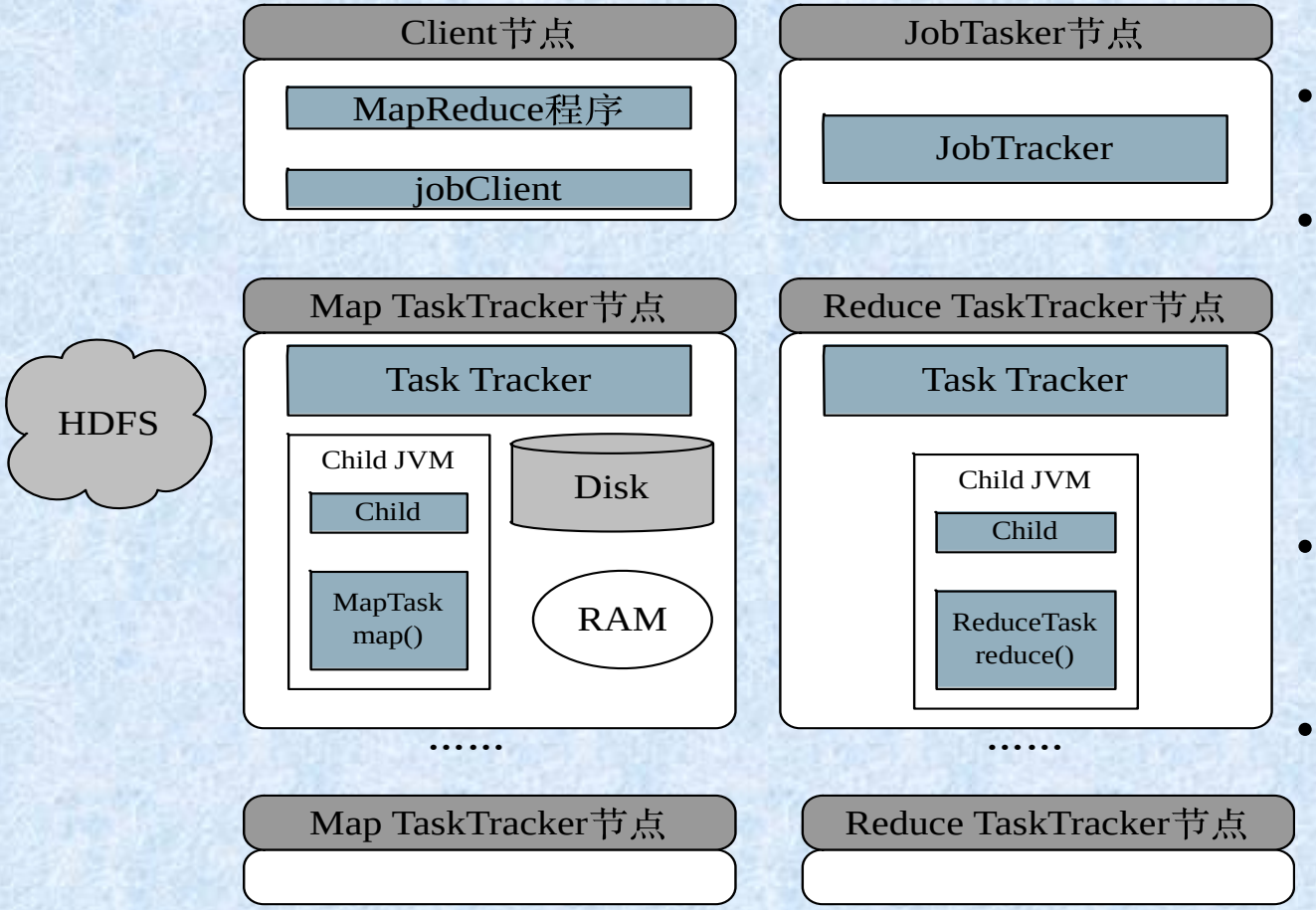
MapReduce计算架构 – JobTracker模式

四大组件：

- Client
- JobTracker
- TaskTracker
- Task



MapReduce 软件框架



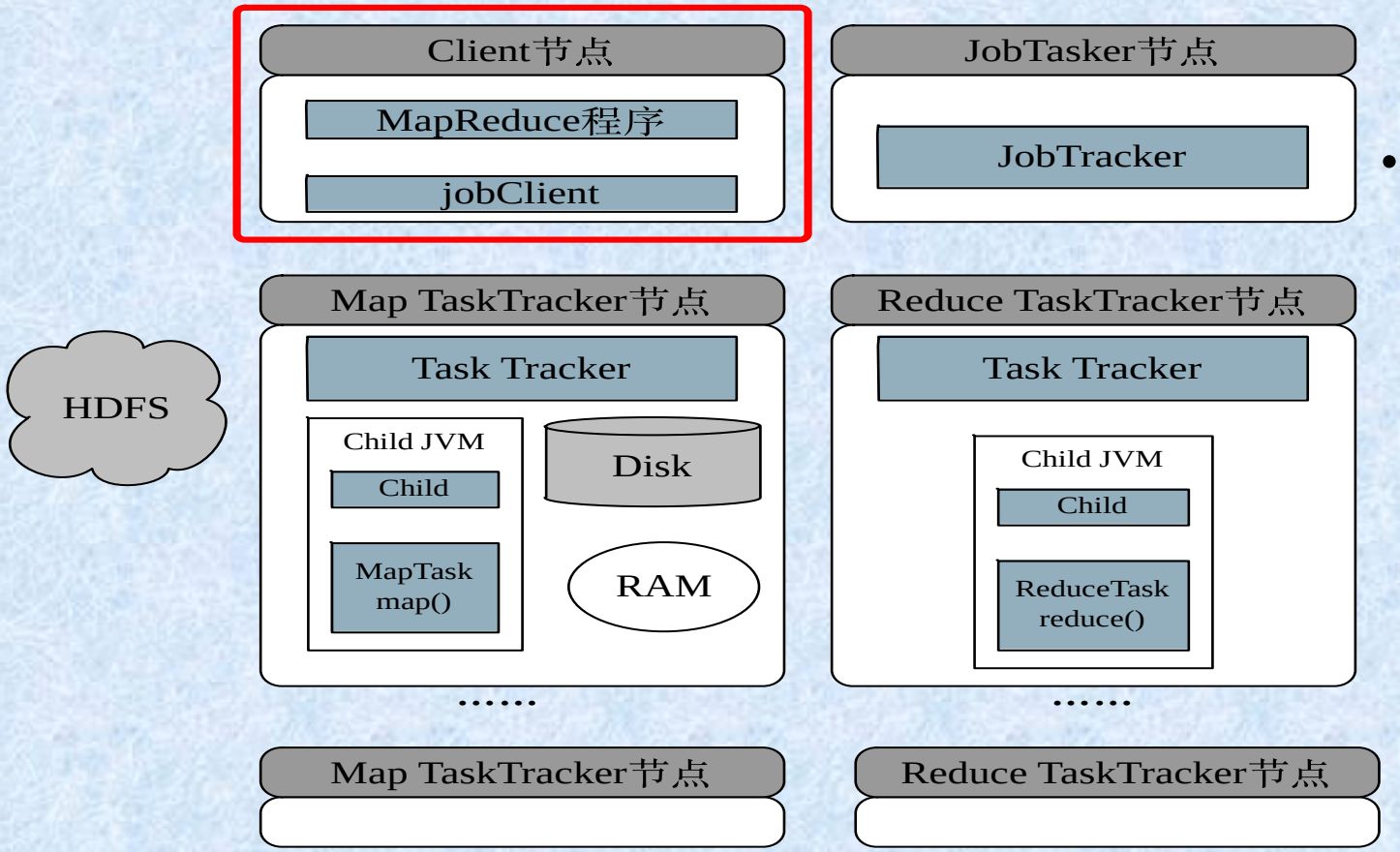
作业（Job）

- MapReduce程序指定的一个完整计算过程；
- 一个作业在执行过程中可以被拆分为若干Map和Reduce任务完成；

任务（Task）

- MapReduce框架中进行并行计算的基本事务单元；
- 分为Map 和Reduce任务，一个作业通常包含多个任务；

MapReduce 组件



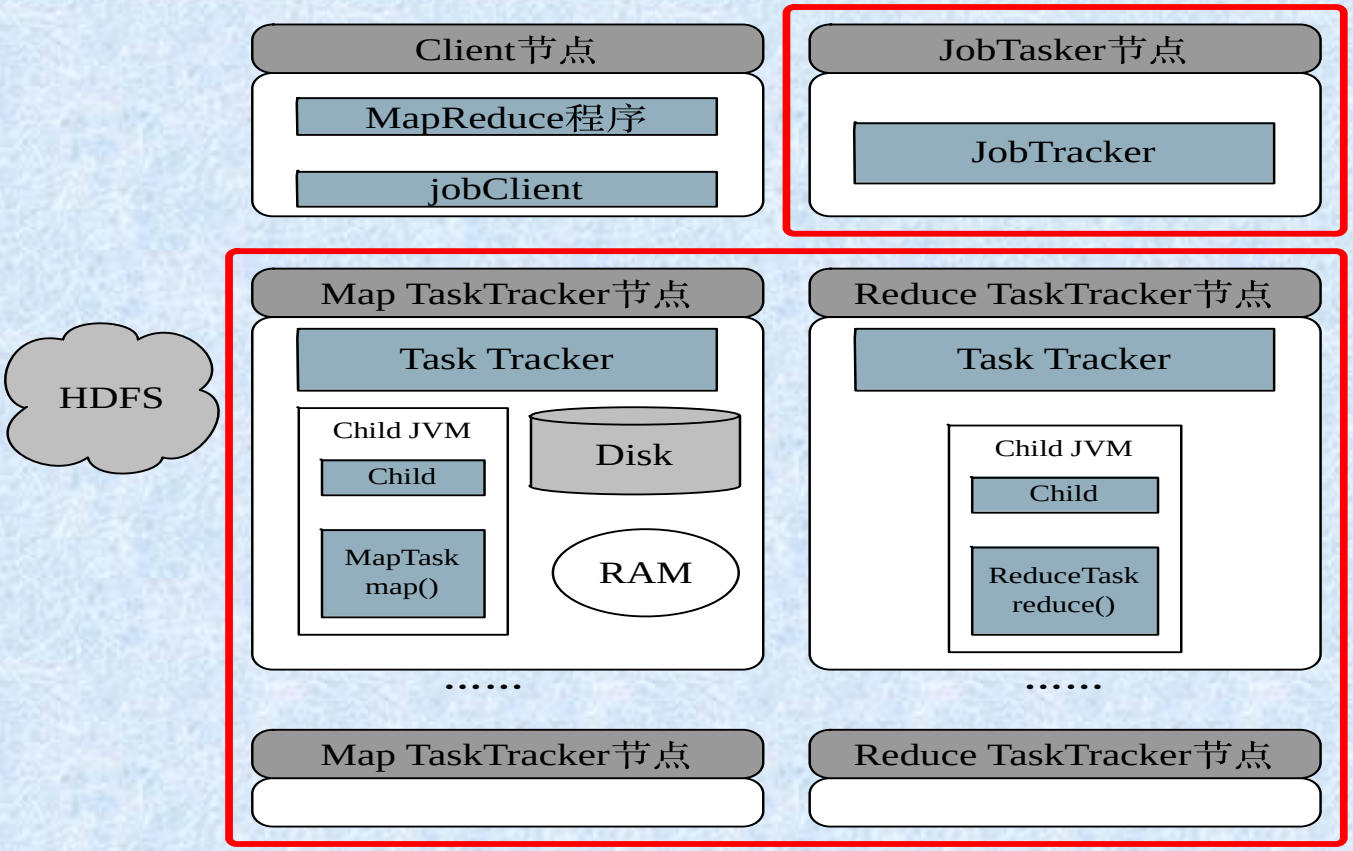
MapReduce程序

- 我们编写的程序

JobClient

- 替程序与MapReduce运行框架交互的对象

MapReduce 组件（续）



JobTracker

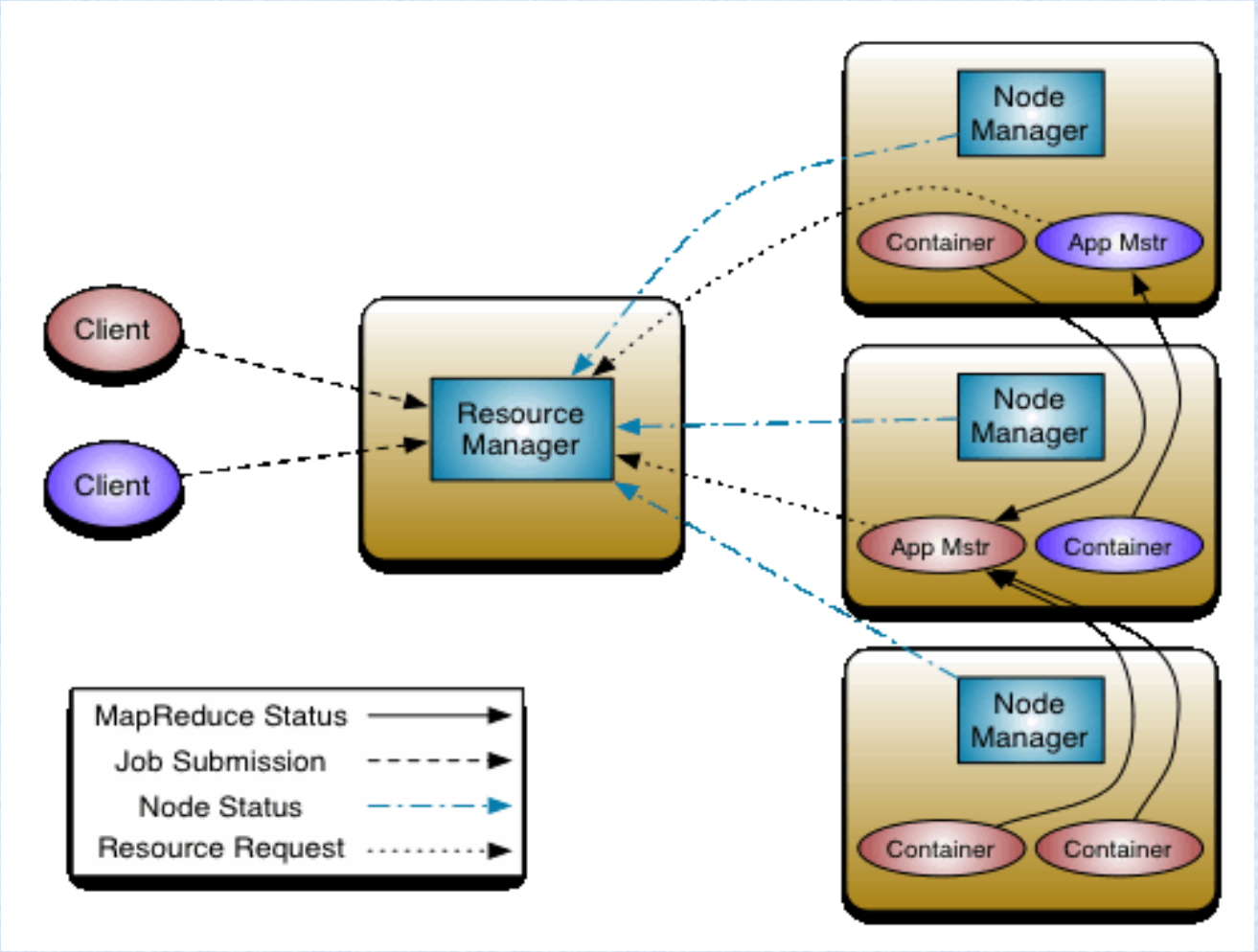
- MapReduce框架的管理者
- 协调MapReduce作业
- 分配任务
- 监控任务

TaskTracker

- 执行JobTracker分配的任务
- 分为Map和Reduce两类

MapReduce计算架构 – YARN模式

- Client
- ResourceManager
- NodeManager
- Container
- Application Master



软件模块/计算单元

MapReduce主要任务—映射与简化

- Map（映射）：负责输入数据的分片、转化、处理，输出中间结果文件；
- Reduce（简化）：以Map的输出文件为输入，对中间结果进行合并处理，得到最终结果并写入HDFS。
- 两类任务都有多个进程运行在DataNode上，相互间通过Shuffle阶段交换数据。

数据模型：键值对

MapReduce是以键值对（key-value pair）来完成数据计算处理。

- 键是行键（RowKey）
  多半用作索引（indexing）
- 值是字符串（character string）或二进制数组（binary string）
  包含存储数据或信息。

键值对（key-value pair）实例：

（123 ， “文件序列编号”）
（“hello”， “1 1 1 1 1 1 1 1 1 1 1 1 1”）
（579.12， “aabbccddeeffgghhiijjkk”）
（“name-0001”， “hsget524##**juyfyf...”）

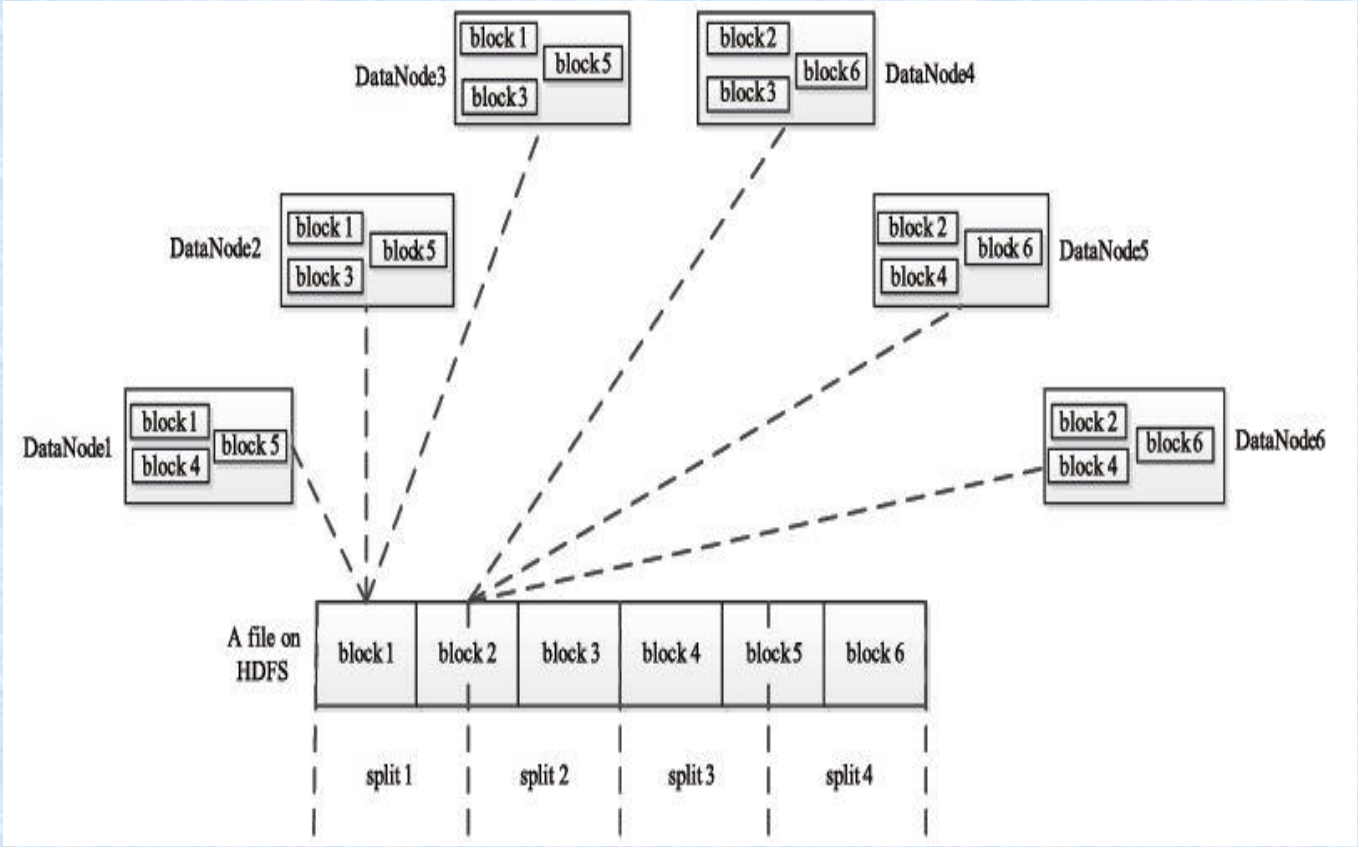
数据输入格式

文件分片—定义

- 把大数据文件进行分片，生成一个个InputSplit（简称为split）
- 一个InputSplit对应一个计算任务（task），分配到计算节点，由map/reduce进程执行计算处理
- split是我们对数据文件出于计算需要的逻辑划分单位，但一个HDFS文件在集群中实际是以块（block）的物理形式存储的

—Split vs block?

Split与Block



Block

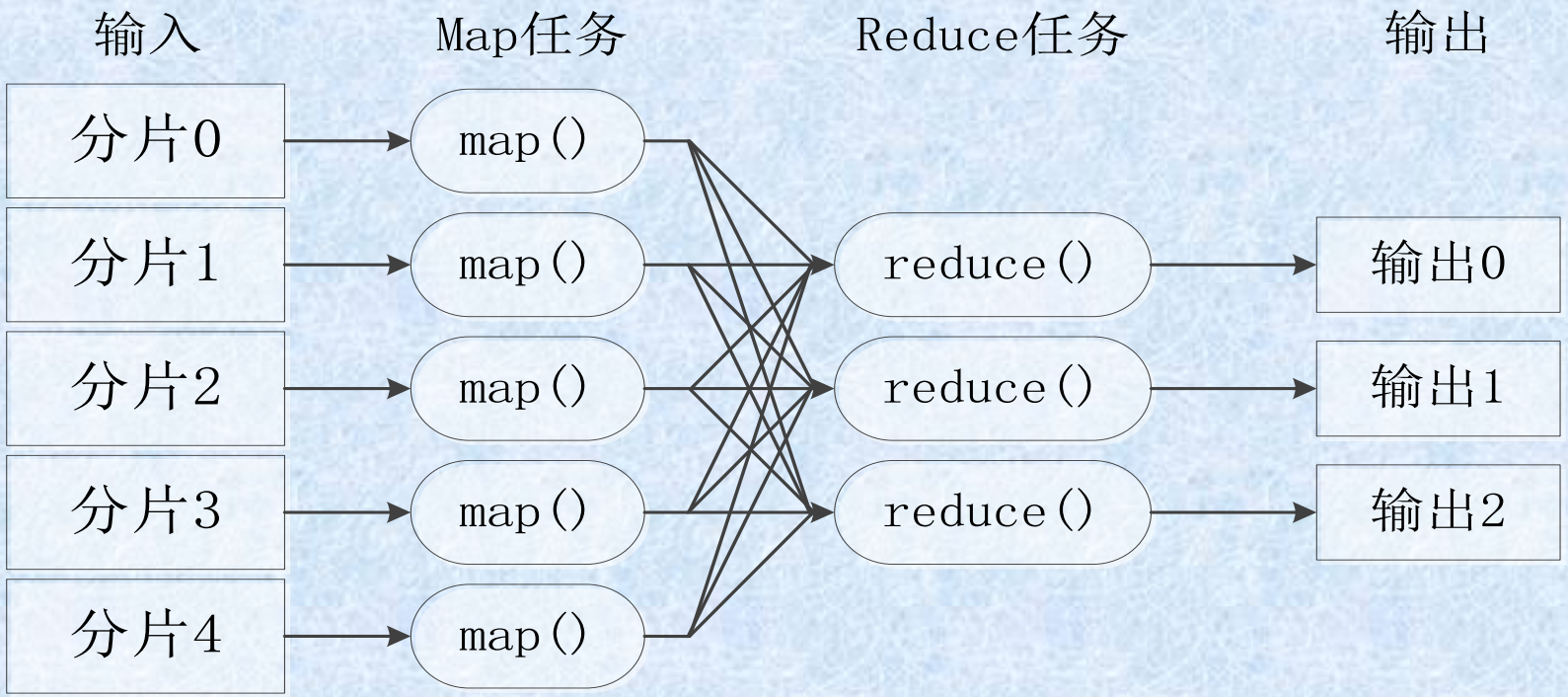
- 一个HDFS文件可以按block形式进行物理存储
- HDFS的物理存储单元

Split

- 一个逻辑上对HDFS文件的划分方式
- MapReduce的计算逻辑单元

Map数目设置

- 相关参数
  - block_size : HDFS文件的block size
  - total_size : 输入文件整体的大小
  - input_file_num : 输入文件个数



Map数目设置

- 计算步骤

1）使用默认map数

如果不进行任何设置，默认的map数由blcok_size决定：

default_num = total_size / block_size;

2）预设map数目

可通过参数mapred.map.tasks来设置期望的map数目，但是这个数只有在大于default_num的时候才会生效：

goal_num = mapred.map.task

3）设置分片大小（split size）

可以通过mapred.min.split.size 设置每个task处理的split的大小，但是这个 大小只有在大于block_size的时候才会生效。

split_size = max(mapred.min.split.size, block_size);

split_num = total_size / split_size;

Map数目设置

4）计算map数目

compute_map_num = min(split_num, max(default_num, goal_num))

5）每一个map处理的分片是不能跨越文件的。

所以，**最终的map个数**应该为：

final_map_num = max(compute_map_num, input_file_num)

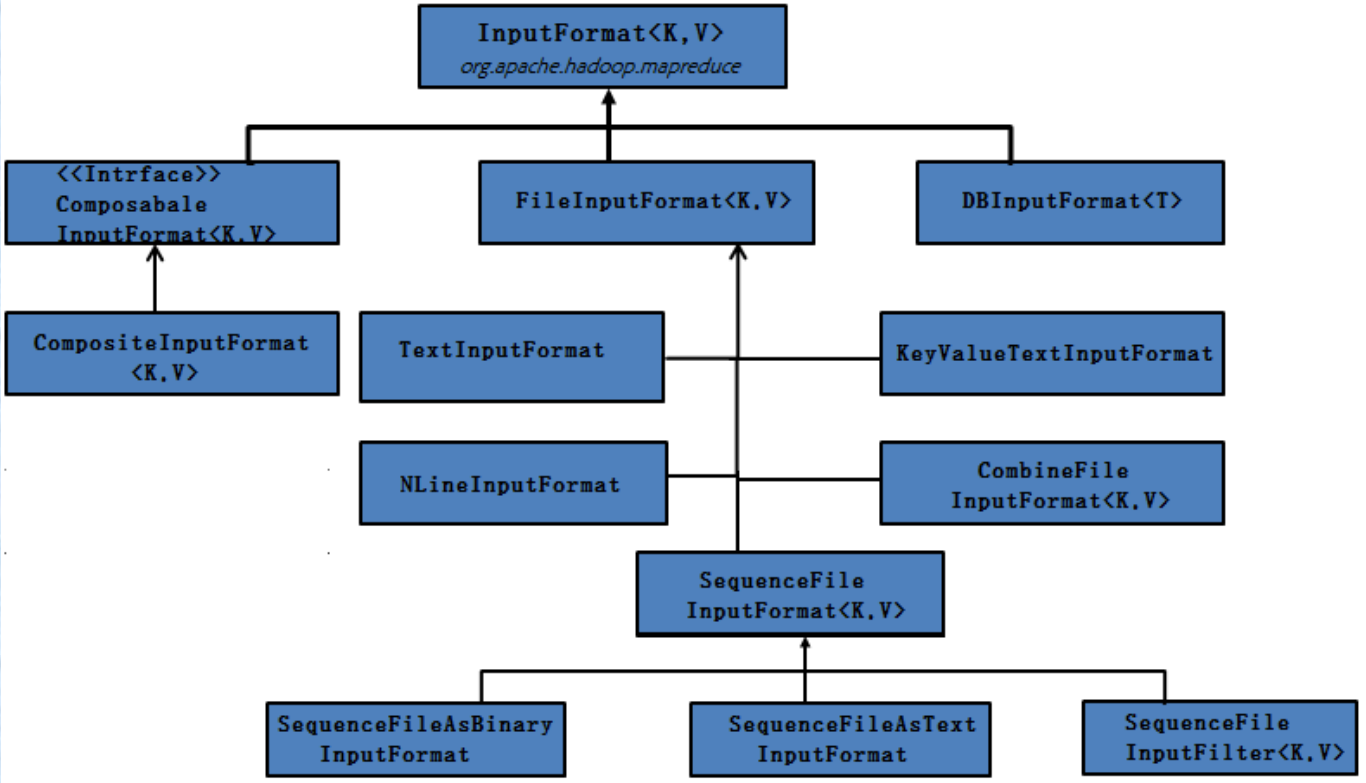
Map数目设置准则

- 增加map个数→设置mapred.map.tasks 为一个较大的值；
- 减小map个数→设置mapred.min.split.size 为一个较大的值；
- 输入中有很多小文件，依然想减少map数目→需将小文件merge为大文件，然后使用第二点准则

输入格式处理

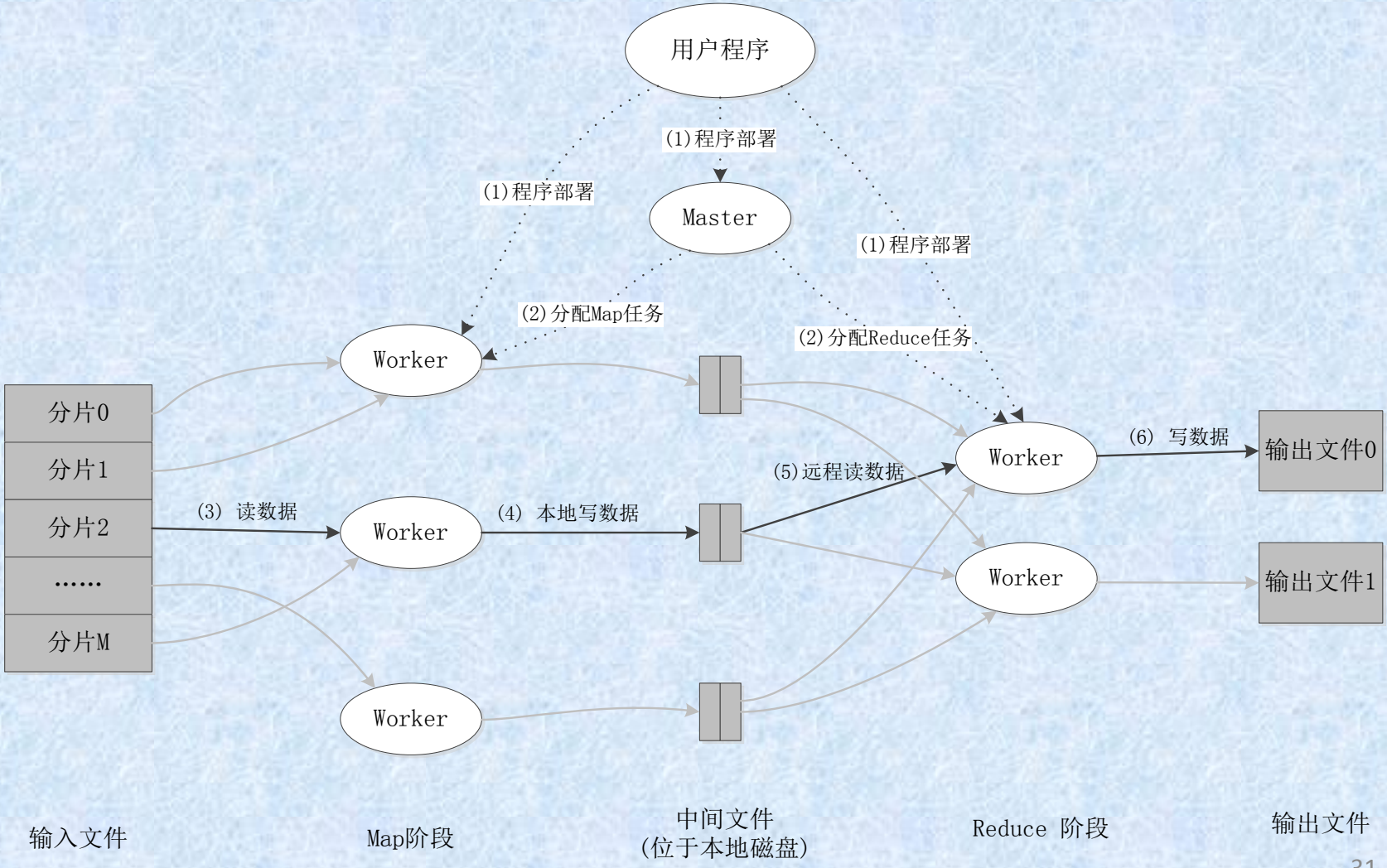
将不同格式数据转换为键值表：基础类InputFormat类

- 选择作为输入的文件或对象
- 提供把文件分片的InputSplits（）方法
- 为RecordReader读取文件提供一个工厂方法



输入格式	描述	键	值
TextInputFormat	默认格式，读取文件的行	行的字节偏移量	行字符串内容
KeyValueInputFormat	把行解析为键值对	第一个tab字符前的所有字符	行剩下的字符串内容
SequenceFileInputFormat	Hadoop定义的二进制格式	用户自定义	用户自定义
SequenceFileAsTextInputFormat	是SequenceFileInputFormat的变体，它将键和值的顺序值转换为text。转换的时候会调用键和值的toString方法。这个格式可以是顺序文件作为流操作的输入。	转换后的键字符串	转换后的值字符串
SequenceFileAsBinaryInputFormat	SequenceFileAsBinaryInputFormat是SequenceFileInputFormat的另一种变体，它将顺序文件的二进制格式键和值封装为BytesWritable对象，应用程序可以任意地将这些字节数组解释为需要的类型。		
DBInputFormat	DBInputForma是一个使用JDBC并且从关系数据库中读取数据的一种输入格式。由于它没有任何碎片技术，所以在访问数据库的时候必须非常小心，太多的mapper可能会事数据库受不了。因此DBInputFormat最好在加载小量数据集的时候用。		

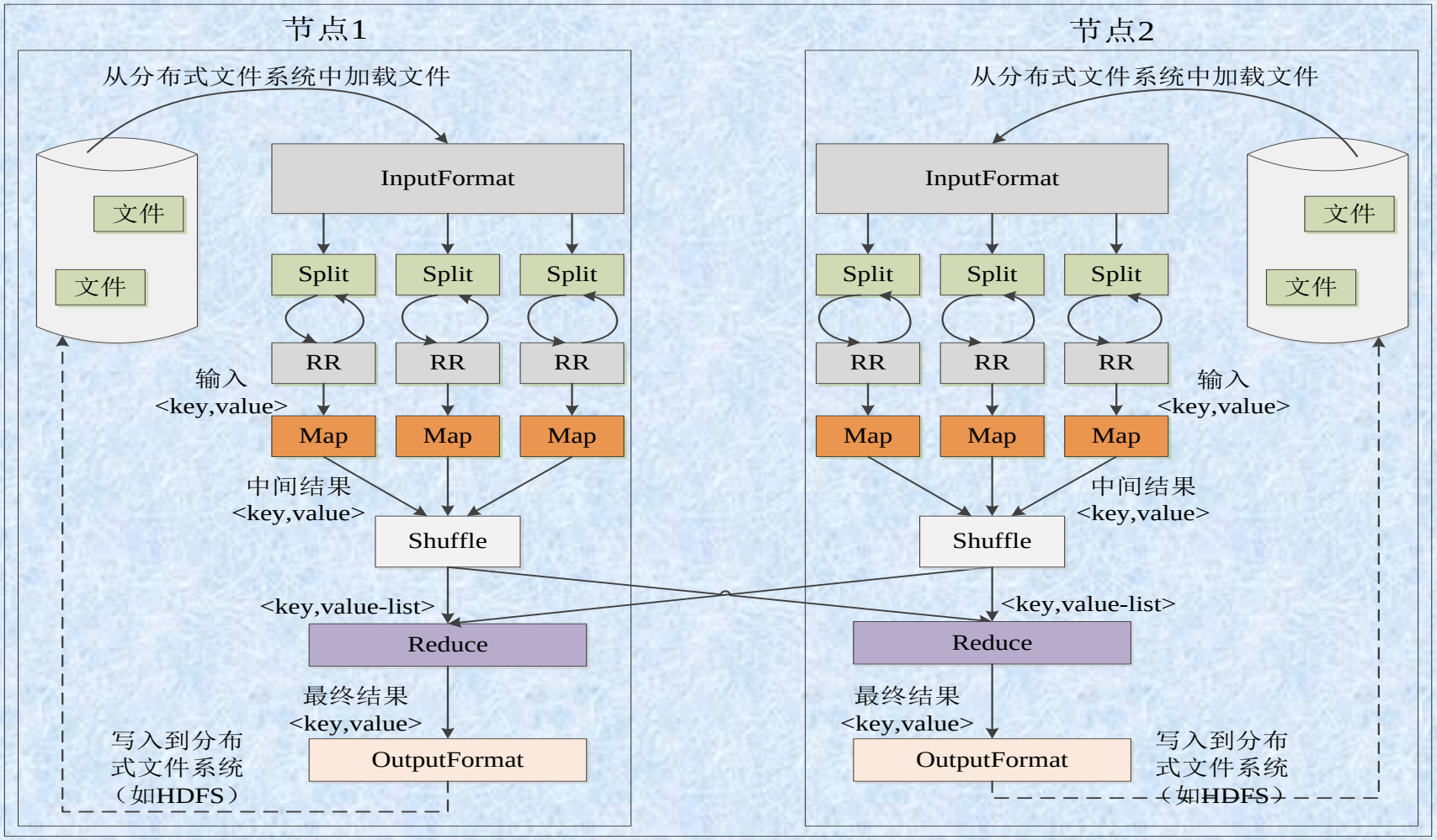
3. MapReduce 计算流程



MapReduce流程三个阶段：Map/Shaffle/Reduce

- Map（映射）
  - Mapper执行map task，将输出结果写入中间文件
- Shuffle（归并）
  - 把Mapper的输出数据归并整理后分发给Reducer处理，包括merge，combine，sort和partition几个步骤；
- Reduce（化简）
  - Reducer执行reduce task，将最后结果写入HDFS。

Map/Shaffle/Reduce任务与实现



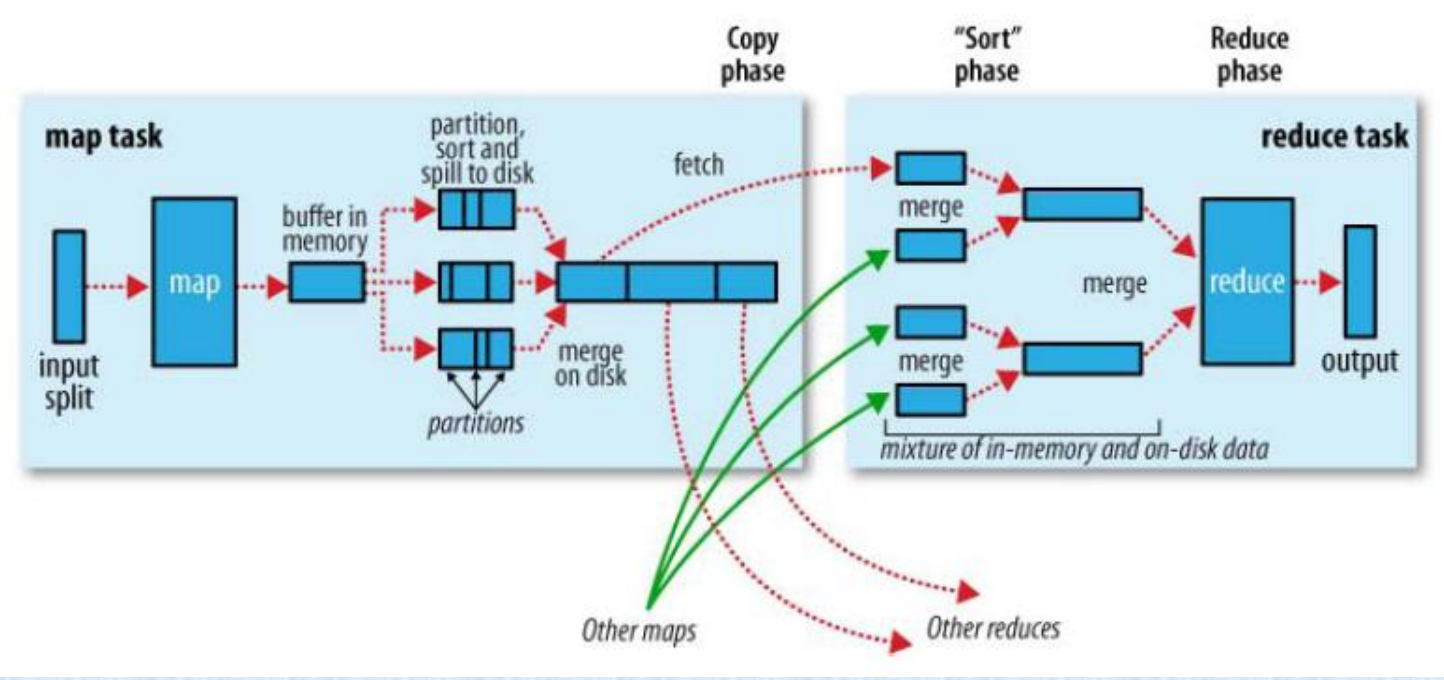
Map（映射）阶段

- Mapper的输入数据来源：MapContext
- MapContext的实现依赖于：MapContextImpl
- MapContextImpl内部组合：InputSplit和RecordReader
  - 提供了读取和封装输入数据键值对（key, value）的方法。

- 对于每一个split，系统都生成一个map task，调用Mapper来执行，将读入数据转换成键值对格式
- 完成计算处理后，将输出结果写入中间文件
- 一个Map任务可以在集群的任何计算节点上运行
- 多个Map任务可以并行地运行在集群上

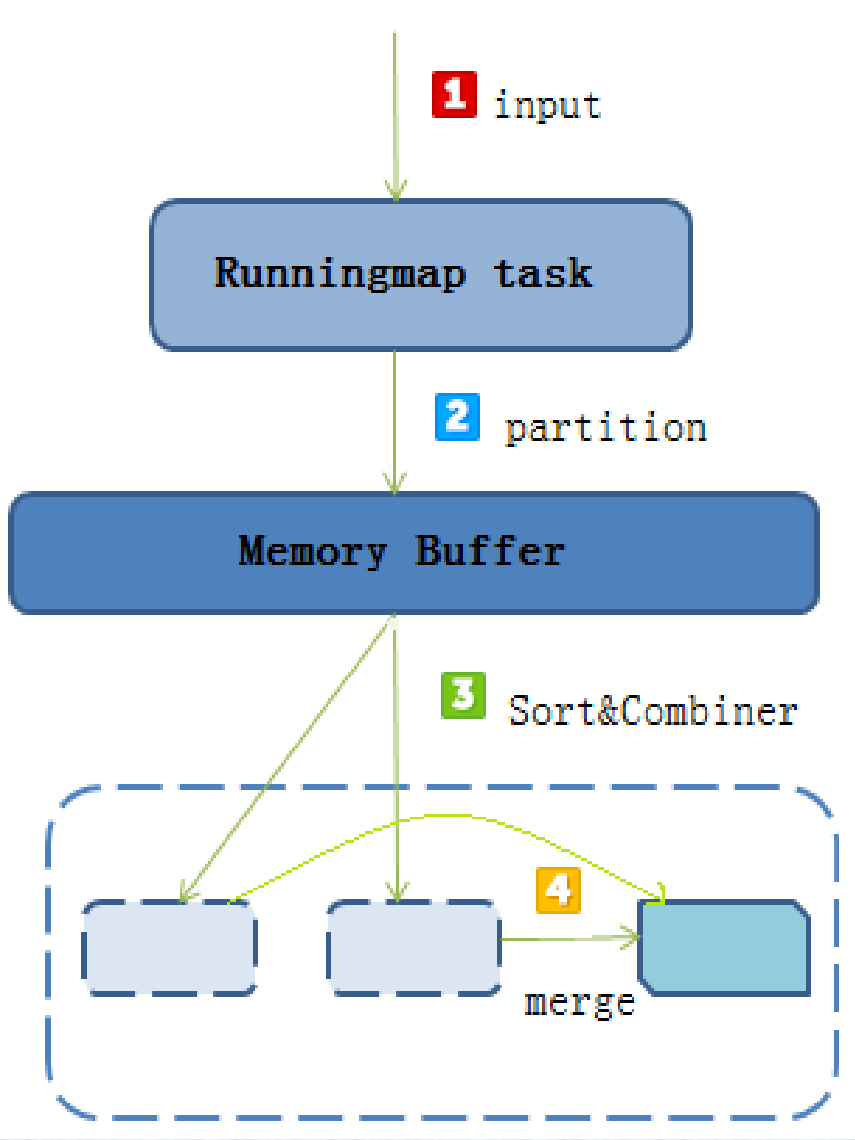
Shuffle（归并）阶段

- 主要任务: 将每个map task的输出结果进行归并、排序、然后按照一定的规则分发给Reducer去执行化简步骤
- Shuffle任务实际上涉及到Map的输出以及Reduce的输入
- Shuffle阶段的两个部分：Map相关部分和Reduce相关部分

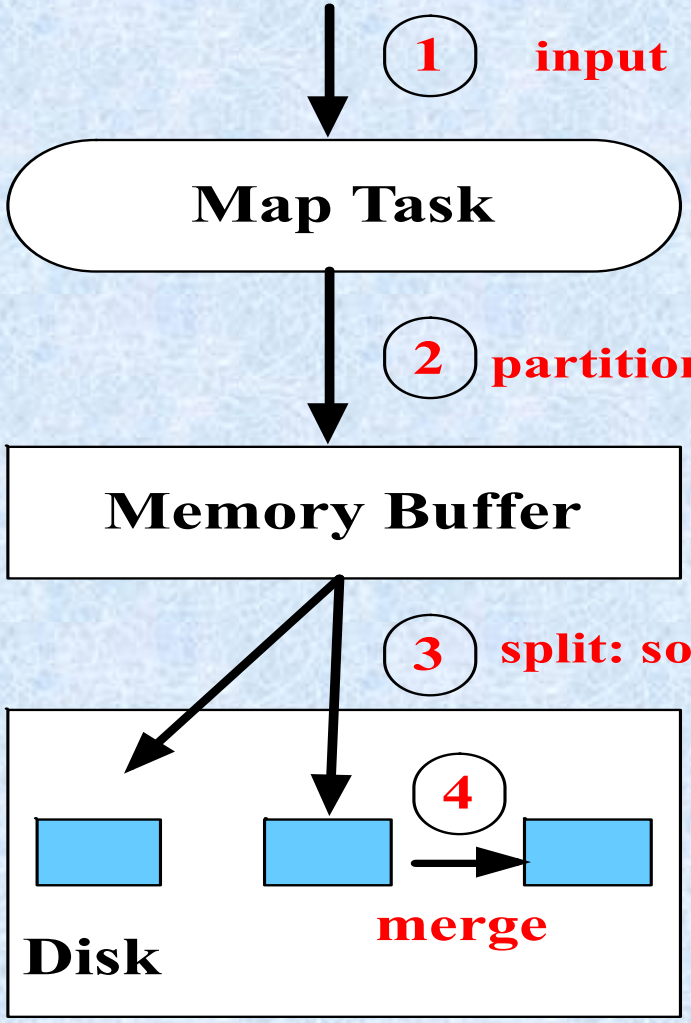


Shuffle阶段—Map端的Shuffle

- Mapper从HDFS读取split，然后执行map ()
- 根据key或value以及Reducer数量划分输出中间表，决定交由哪个reduce task来处理
- 将中间数据写入内存缓冲区
- map task完成时，将全部溢写文件归并到一起合成一个溢写文件（Merge）



Map端Shuffle结果保存



① 处理输入数据

② 寻找对应Reduce

（Partition）

③内存数据溢出到磁盘

（Spill）

– Sort

– Combine

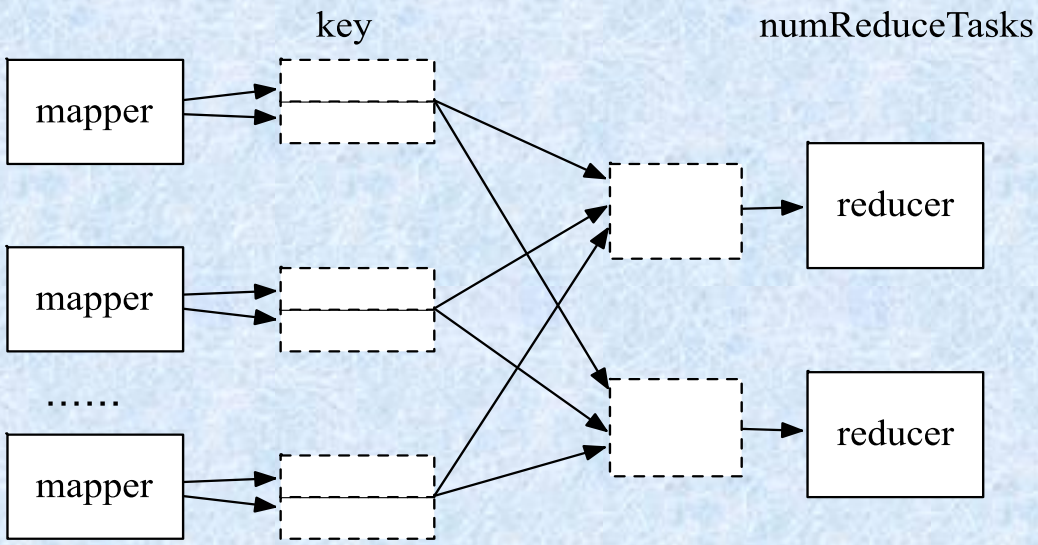
④ 合并中间结果文件

（Merge）

Partition

- 作用：将map的结果发送到reduce
- 要求：负载均衡、效率
- 默认HashPartitioner

(key.hashCode() & Integer.MAX_VALUE)%numReduceTasks



key	value
hello	1
world	1
i	1
love	1
you	1

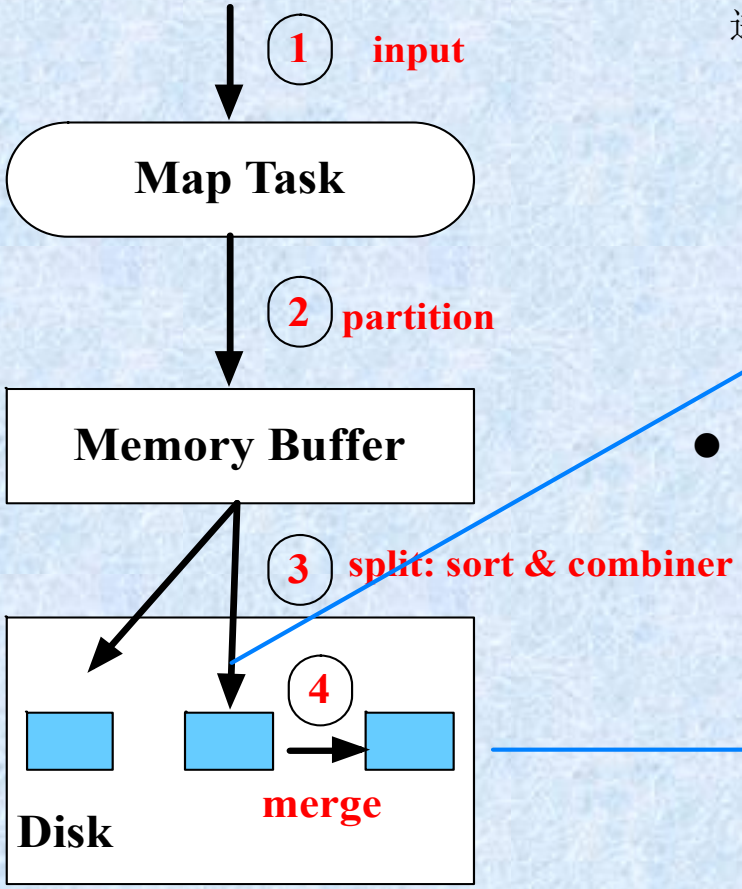
key	value
hello	1
i	1
love	1
i	1
love	1
love	1

key	value
i	1
love	1
you	1
my	1
you	1

key	value
world	1
you	1
my	1
you	1

- 自定义：job.setPartitionerClass(MyPartitioner.class);

Sort-Combine-Merge



- Map后的第1次排序：文件内部快速排序（Sort）
  - map函数处理完输入数据之后，会将中间数据存在本机的一个或者几个文件当中，并且针对这些文件内部的记录进行一次快速排序

key	value
i	1
love	1
you	1
my	1
love	1

key	value
i	1
my	1
love	1
you	1
love	1

- Map后的第2次排序：多个文件归并排序（Merge）
  - Map任务执行完成后会对这些排好序的文件做一次归并排序，并将排好序的结果输出到一个大的文件中

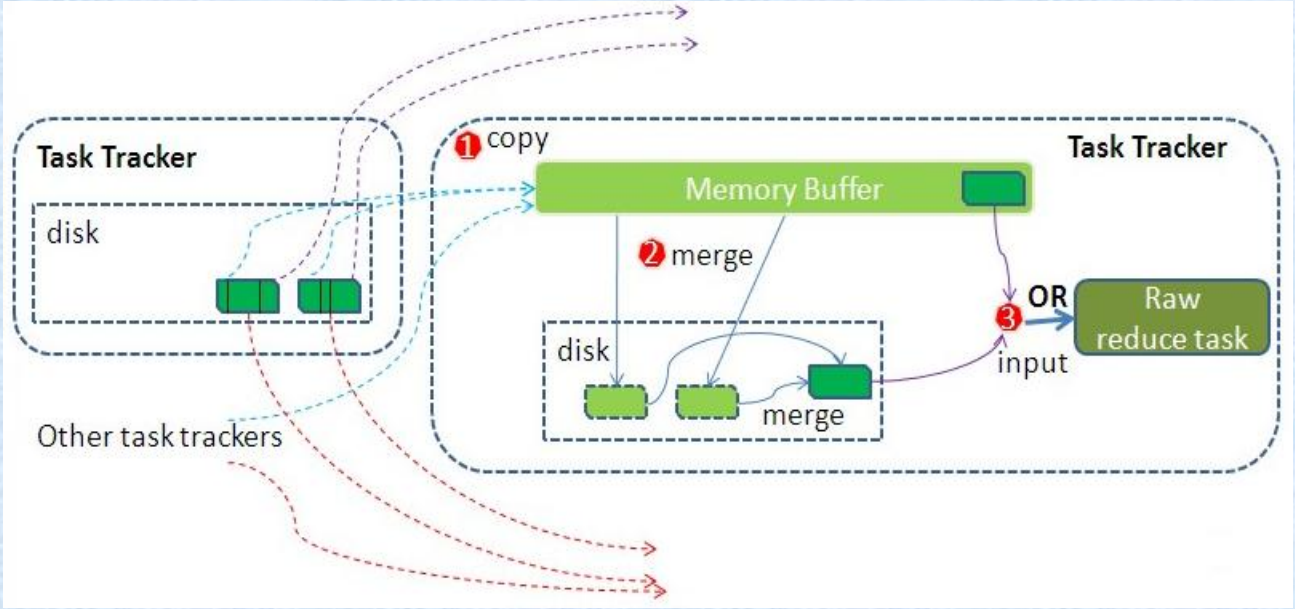
key	value
i	1
love	1
you	1
my	1
love	1

…

key	value
i	1
my	1
love	1
you	1
love	1

Shuffle阶段—Reduce端的Shuffle

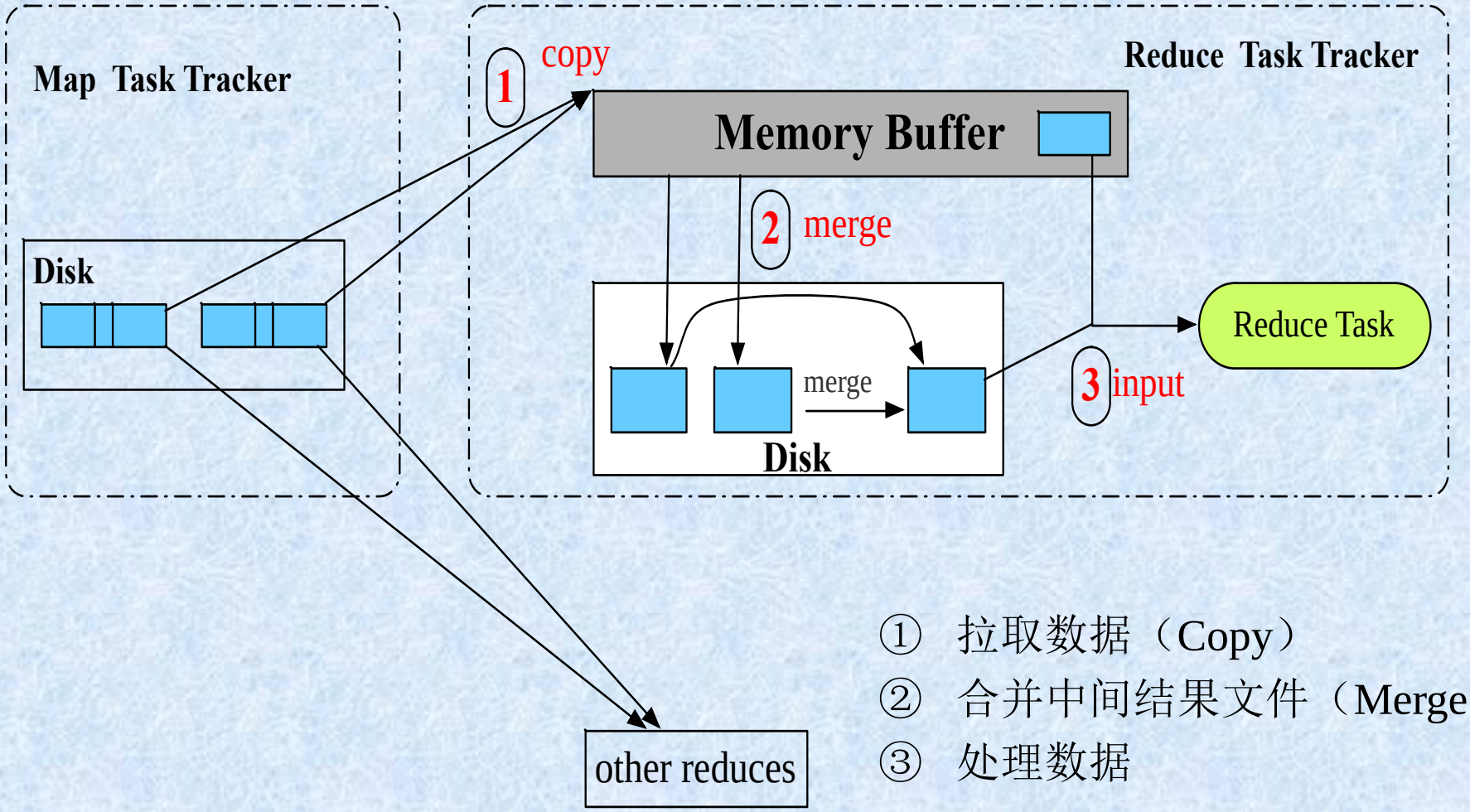
- Copy领取数据
- Merge归并数据
- 最后的归并文件作为Reducer的输入文件发送给Reducer执行
- 最后输出结果存入HDFS。



Shuffle阶段—Reduce端的Shuffle

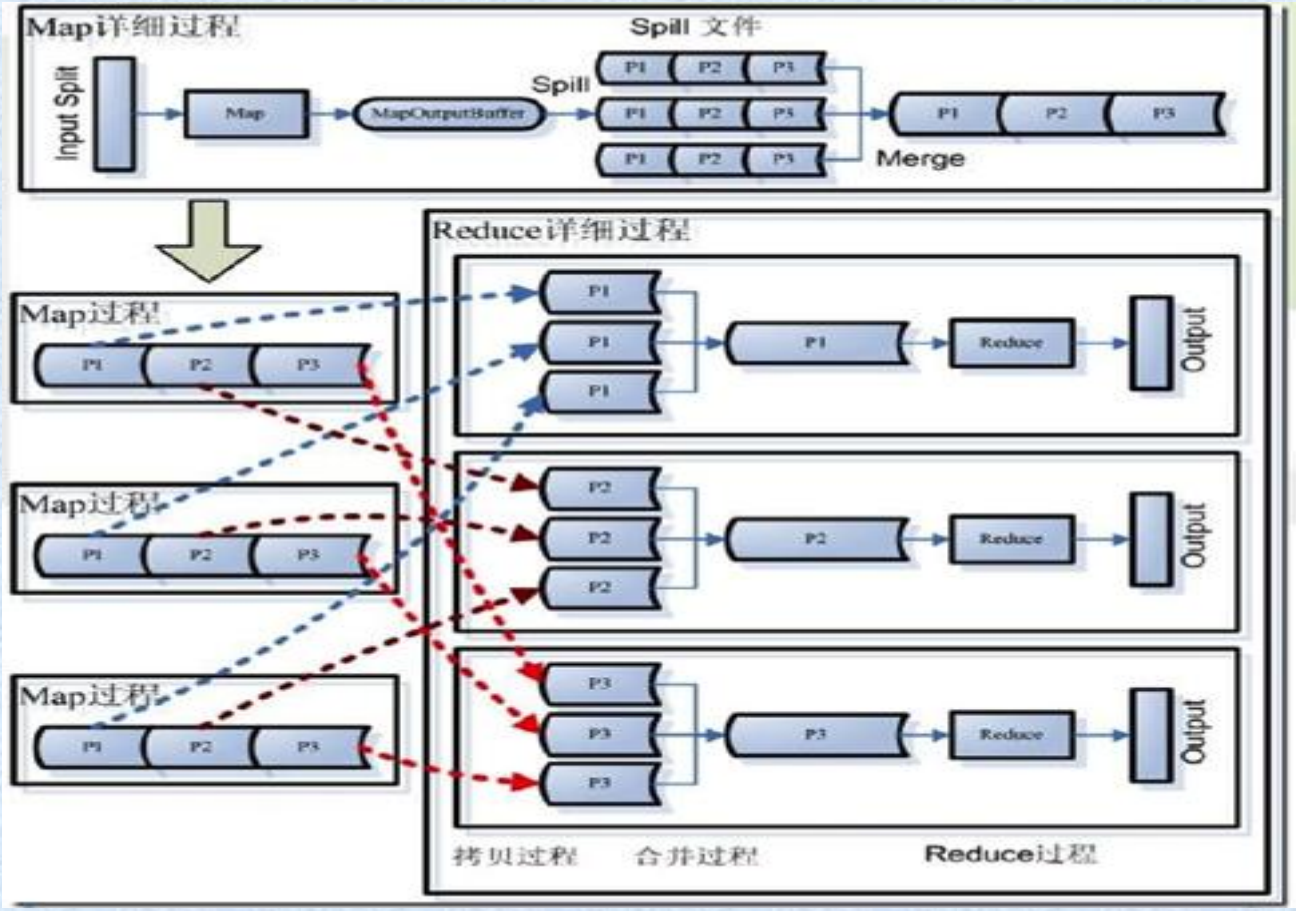
- 将多个Map任务的输出结果按照不同的分区copy到不同的reduce节点执行reduce task
- 每个reduce task都会创建一个Reducer实例
- 通过执行Reducer的reduce ()方法将来自不同Map的具有相同的key值的键值对进行合并处理
- Reducer会通过调用OutputCollector对象将它所完成的化简结果写入HDFS文件系统，输出文件格式由OutputFormat类来控制

Reduce端拉取数据



① 拉取数据（Copy）
② 合并中间结果文件（Merge）
③ 处理数据

Shuffle总结


Map详细过程
Spill 文件
Input Split
Map
MapOutputBuffer
Spill
Merge
Reduce详细过程
Map过程
Map过程
Map过程
Reduce
Output
拷贝过程
合并过程
Reduce过程

4. 实际算例

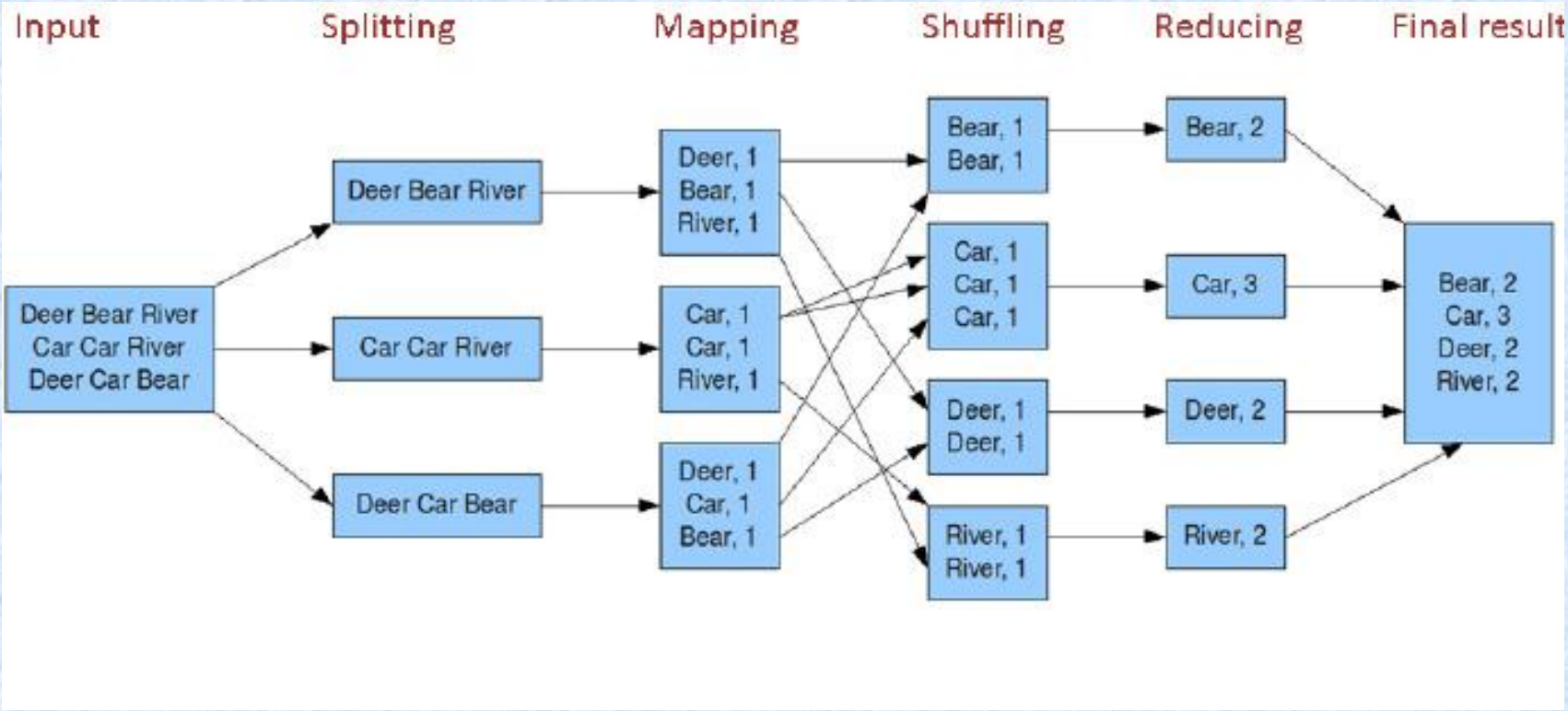
算例描述

- **输入文件**：一个包含3行文字的文本文件（每个单词间用空格隔开，图12-18最左侧Input列所示）；
- **输出结果**：该文件的词频统计，每一行输出一个键值对“单词，出现次数”（图12-18最右侧Final result列所示）；
- **计算模型**：MapReduce

实际案例展示

案例描述

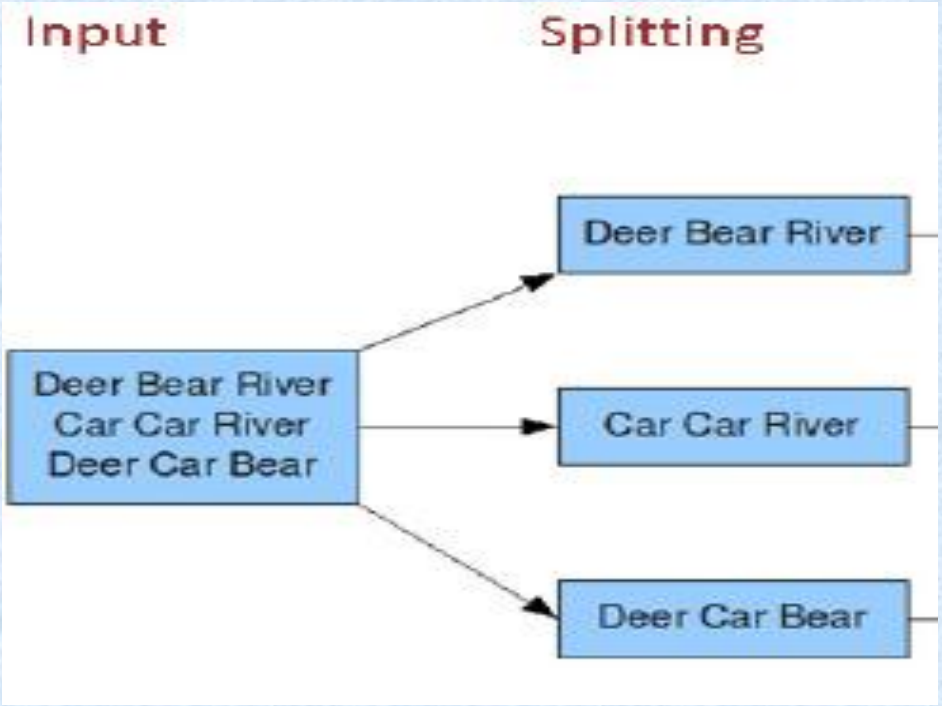
缺点：产生大量中间文件，导致中间件存储在硬盘，导致效率低



实际案例展示

第一步：Split

- 假设将输入数据文件分为3个split，每个split包含一行文字



实际案例展示

第二步：Map

- 先将split的每一行文字转换成如下的键值对（每行第一个字符的字节偏移量作为Key）：

Key	Value
0	Deer Bear River

split 2

Key	Value
16	Car Car River

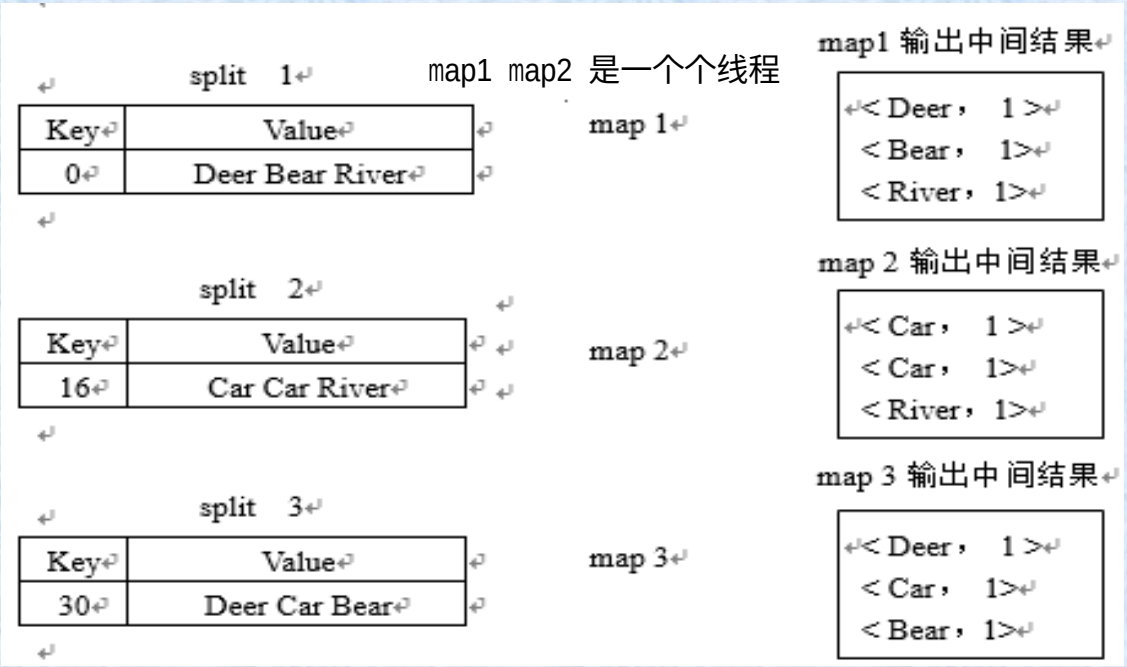
split 3

Key	Value
30	Deer Car Bear

实际案例展示

第二步：Map

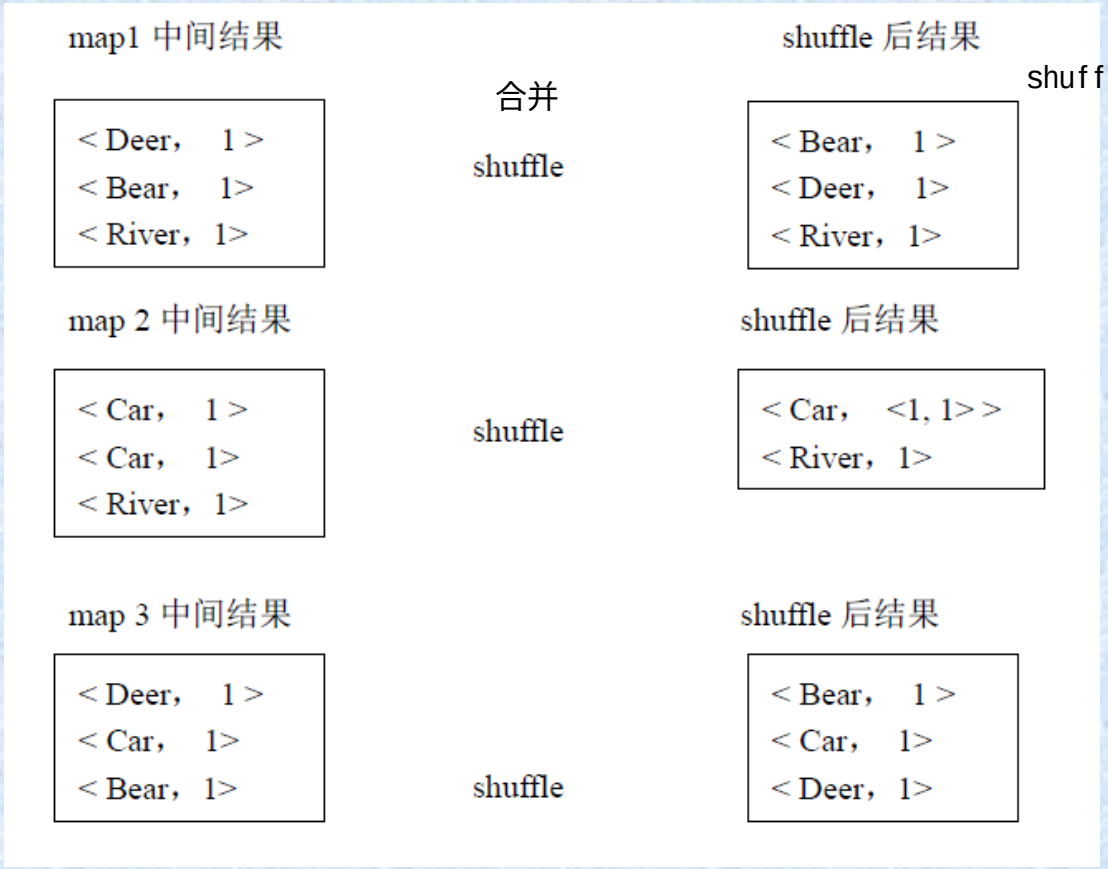
- 然后针对每一个split执行map ( )方法，此处为对上述键值对表的每一行进行词频统计，每一个Map任务（针对一个split）都会生成如下的键值对：



实际案例展示

第三步：Shuffle

- Map端shuffle
- 没有定义Combiner



实际案例展示

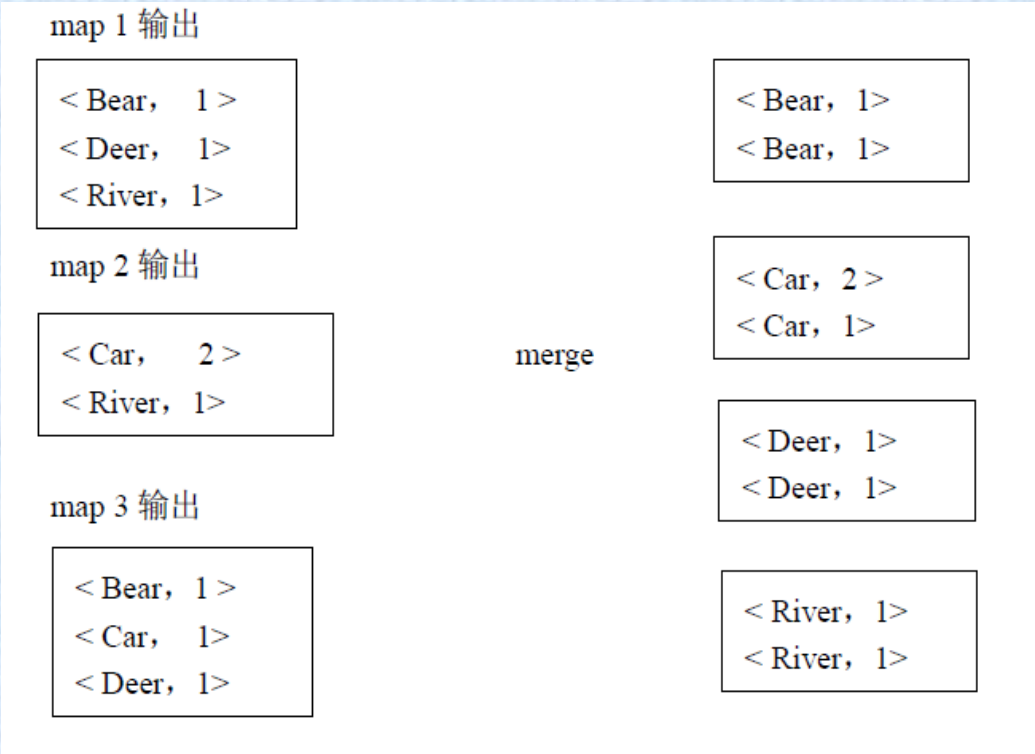
第三步：Shuffle

- Map端shuffle
- 定义了Combiner

实际案例展示

第三步：Shuffle

- Reduce端shuffle



实际案例展示

第四步：Reduce