MapReduce（一）

@八斗数据

建立几个默契

- 空杯的心态
- 用100%的热情去参与，为自己的学习负责
- 有用就是有用，没用的就是没用
- 坚持不懈，终有所成
- 没问题打“1”示意，否则打“0”

OutLine

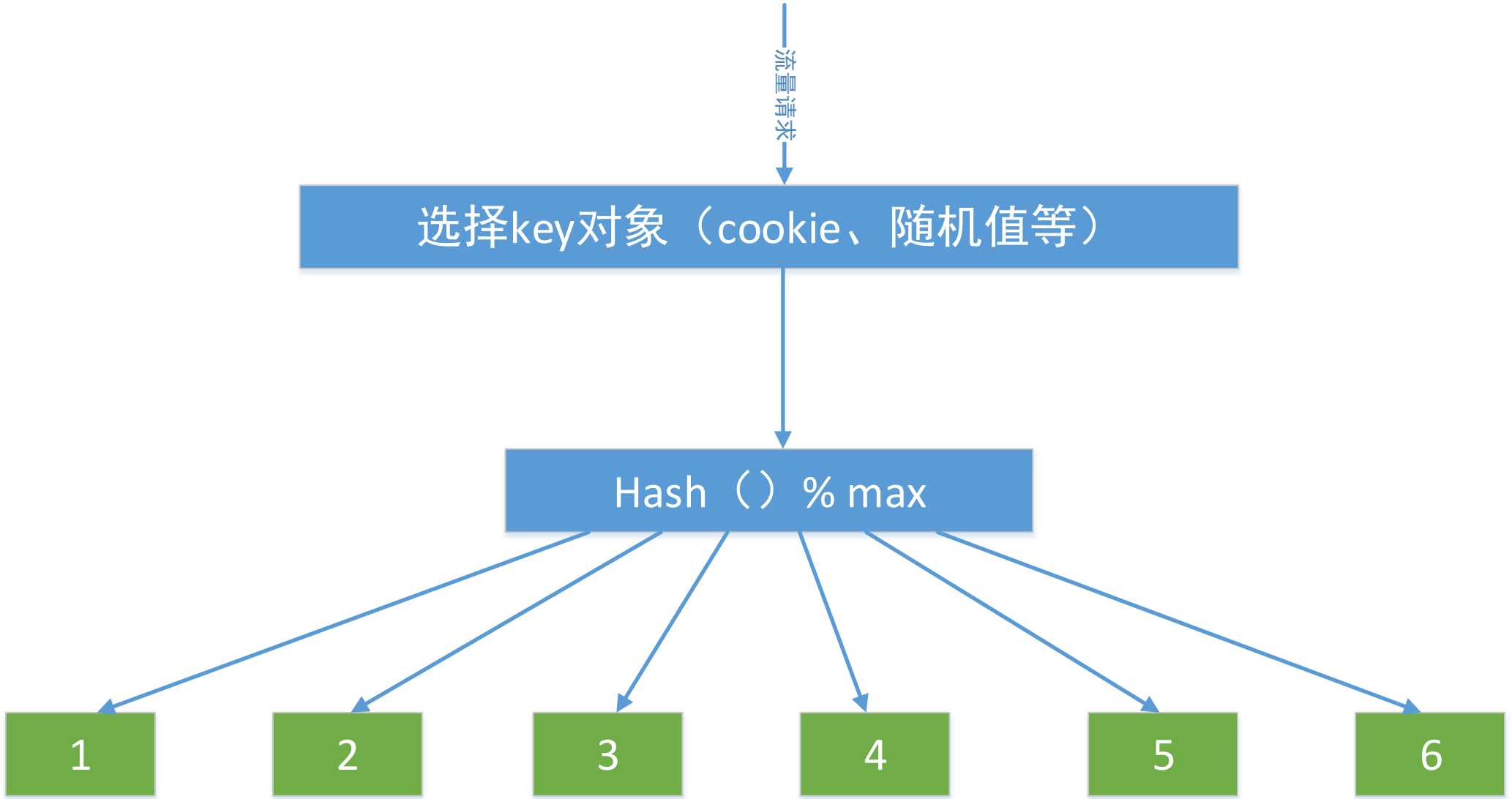
海量数据分流处理技术

MapReduce基础

划分方法——最基本的海量技术思想

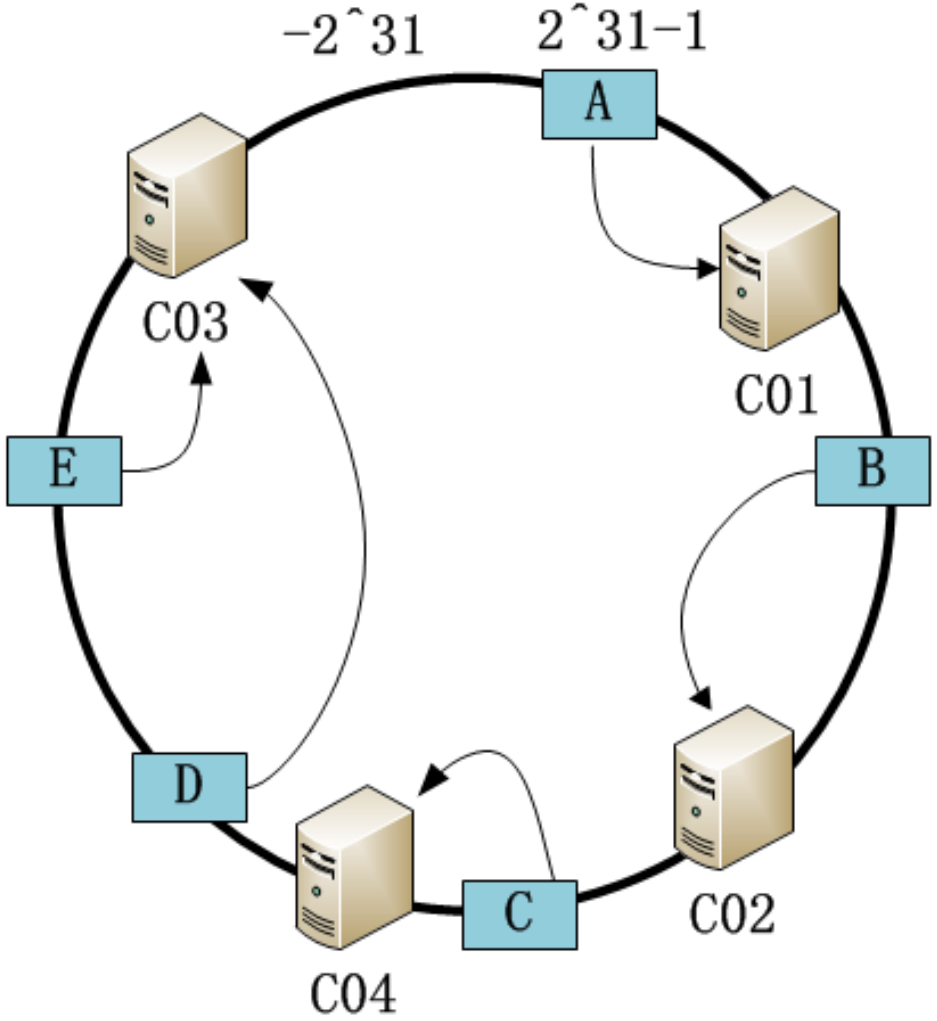
- 传统Hash，最基本的划分方法
  - 如何将大数据、流量均分到N台服务器
  - 找到合理的key，hash（key）尽量分布均匀
  - hash(key)mod N == 0 分到 第0台，
  - hash(key)mod N == i 分到 第i台
  - hash(key)mod N == N-1 分到 第N-1台
- 随机划分
- 一致性Hash：支持动态增长，更高级的划分方法

传统Hash应用——流量分发



一致性Hash算法（consistent hashing）

- 一致性哈希算法在1997年由麻省理工学院的Karger等人在解决分布式Cache中提出的，设计目标是为了解决因特网中的热点(Hot spot)问题



分而治之——最基本的海量技术思想

- 大数据量——按数据划分
  - 早期搜索引擎中的网页存储系统，单机存储数千万网页，几十亿的网页需要通过几百台单机服务器存储，url为key
  - 分布式文件系统，按block(64-256M)来划分组织文件
- 大流量——按流量划分
  - 覆盖的大流量互联网服务
  - 南方流量分到电信机房，北方流量分到联通机房
  - 搜索引擎将query作为key来分流
- 大计算——按输入数据，划分计算任务
  - Map Reduce 按输入数据来划分

云计算技术难点

- 单机系统变为分布式集群系统
- 稳定性和容错能力
- 数据一致性
- 难点:
  - 任何消息存在丢失的可能
  - 任何单机存在故障的风险

OutLine

海量数据分流处理技术

MapReduce基础

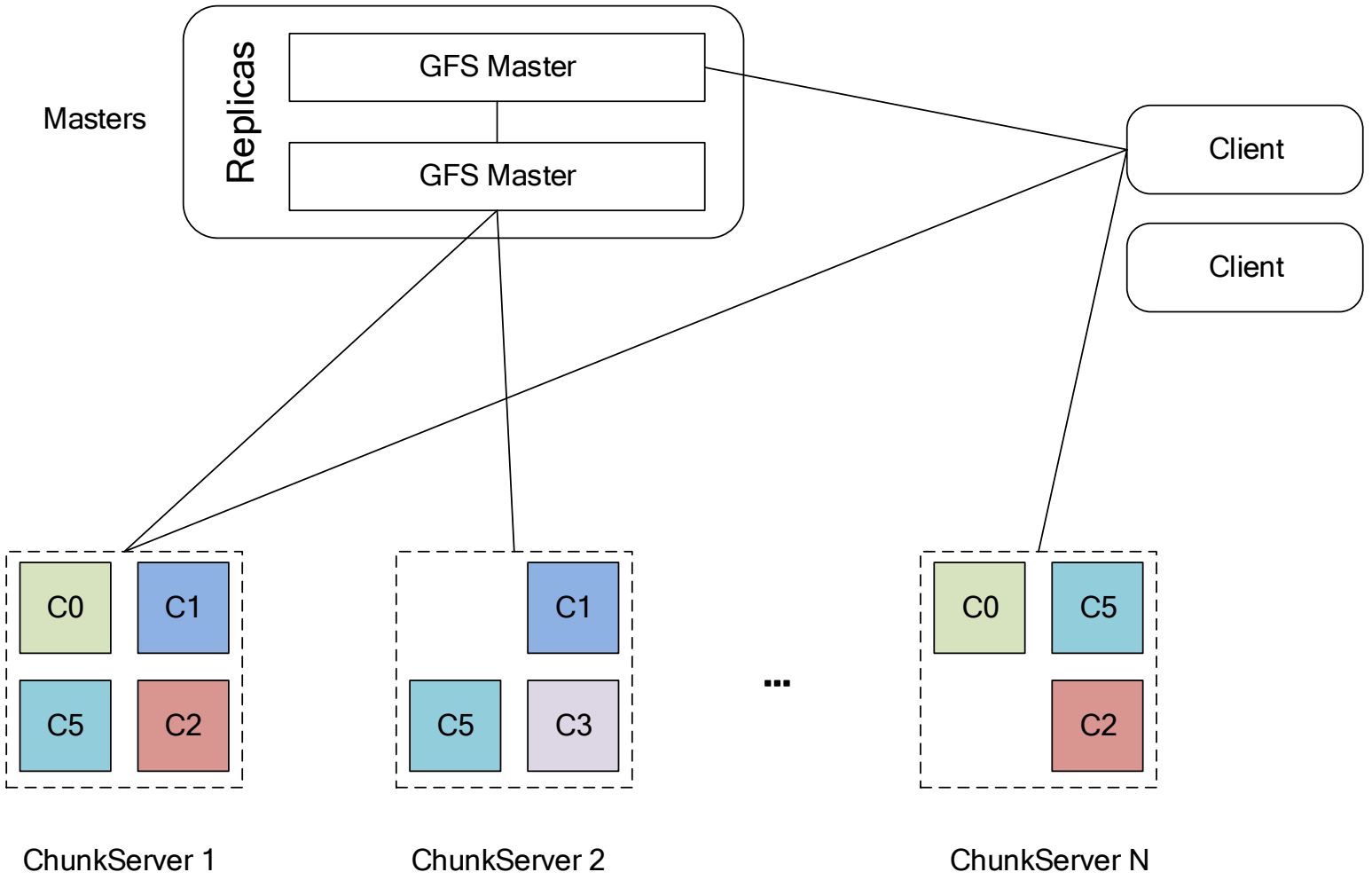
MapReduce简介

- MapReduce是一个用于处理海量数据的分布式计算框架。
- 这个框架解决了
  - 数据分布式存储
  - 作业调度、
  - 容错、
  - 机器间通信等复杂问题

MapReduce存储

- 为什么HDFS
  - 系统可靠性
  - 可扩展性
  - 并发处理

MapReduce存储



MapReduce基础

- 一个简单的WordCount如何编程？
- 1000个词的如何做
  - So easy
- 1000G大小的时候如何做
  - 还行吧
- 1000G*1000G大小的时候如何做
  - 晕了
- 1000G*1000G*1000...如何办
  - 怒了

MapReduce分而治之思想

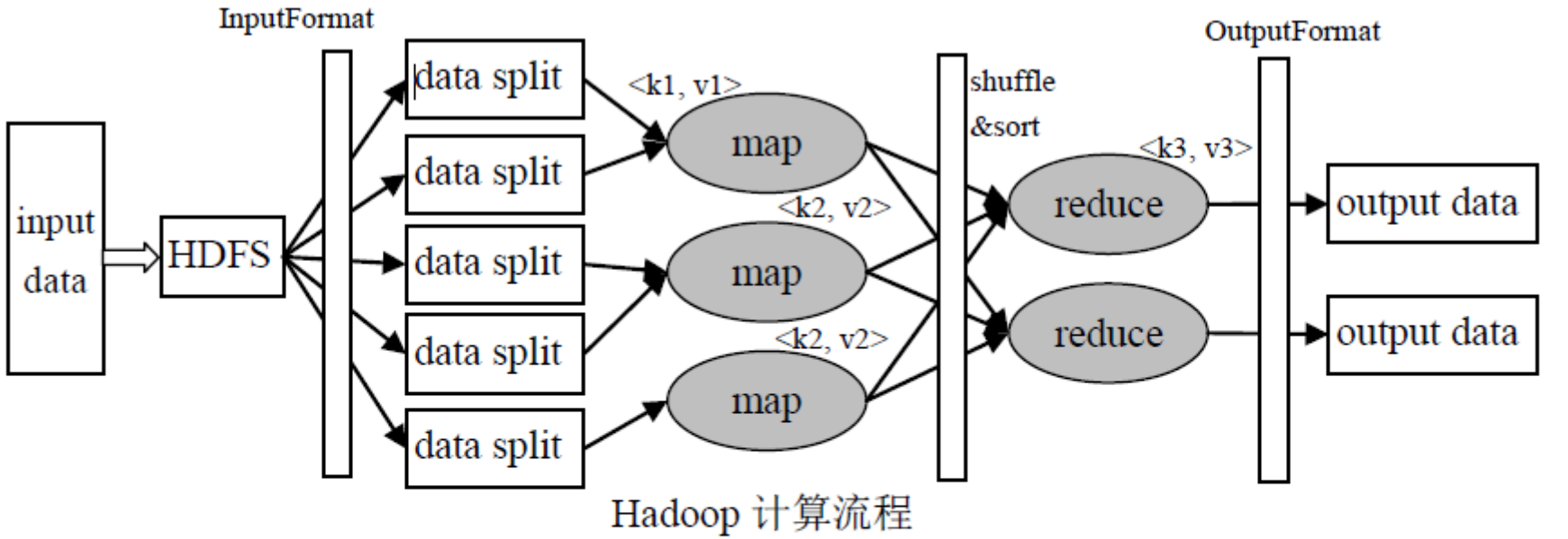
- 数钱实例：一堆钞票，各种面值分别是多少
  - 单点策略
    - 一个人数所有的钞票，数出各种面值有多少张
  - 分治策略
    - 每个人分得一堆钞票，数出各种面值有多少张
    - 汇总，每个人负责统计一种面值

- 解决数据可以切割进行计算的应用

MapReduce分而治之思想

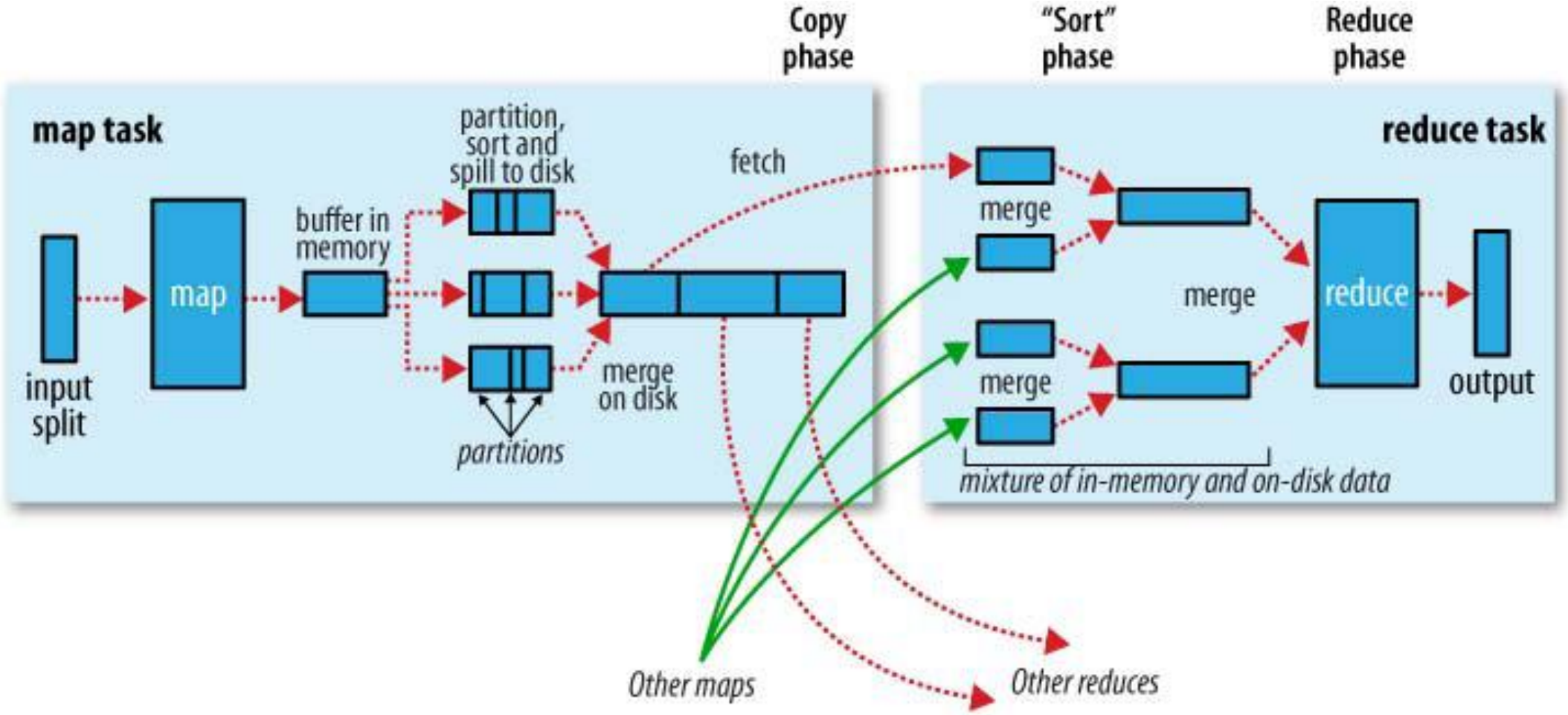
- 分治思想
  - 分解
  - 求解
  - 合并
- MapReduce映射
  - 分：map
    - 把复杂的问题分解为若干“简单的任务”
  - 合：reduce

MapReduce计算框架-执行流程

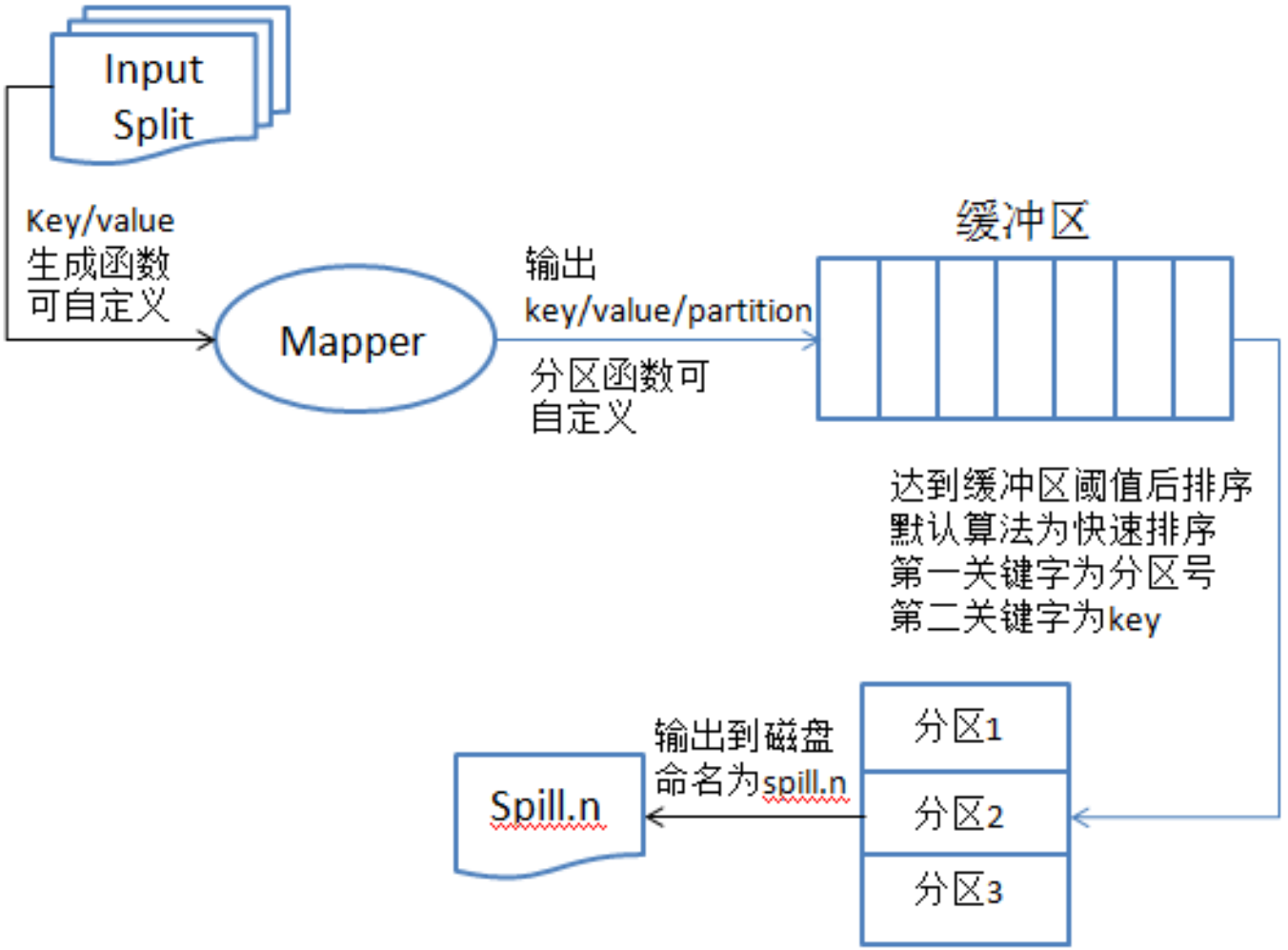


- 应用程序开发人员一般情况下需要关心的是图中灰色的部分!

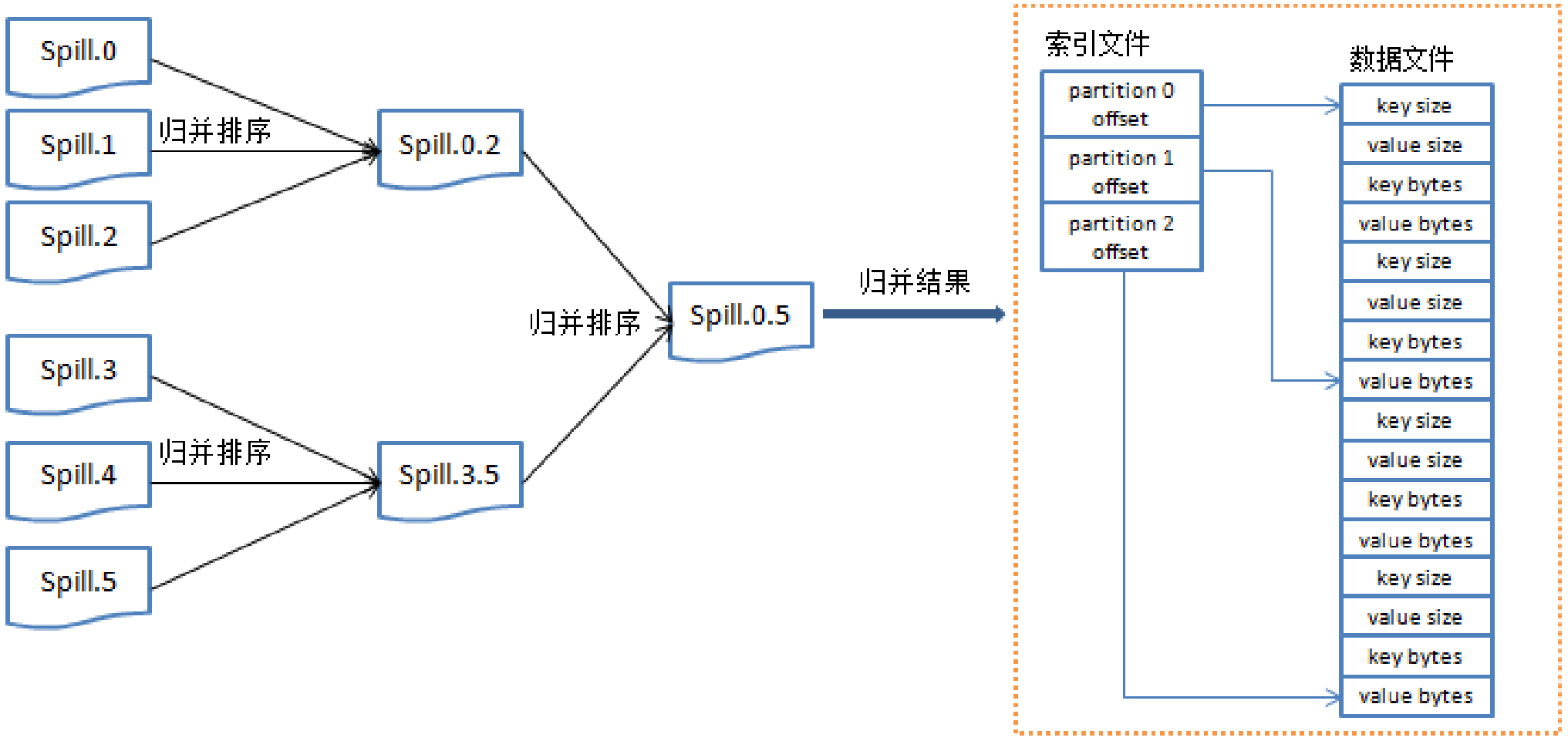
MapReduce计算框架-执行流程



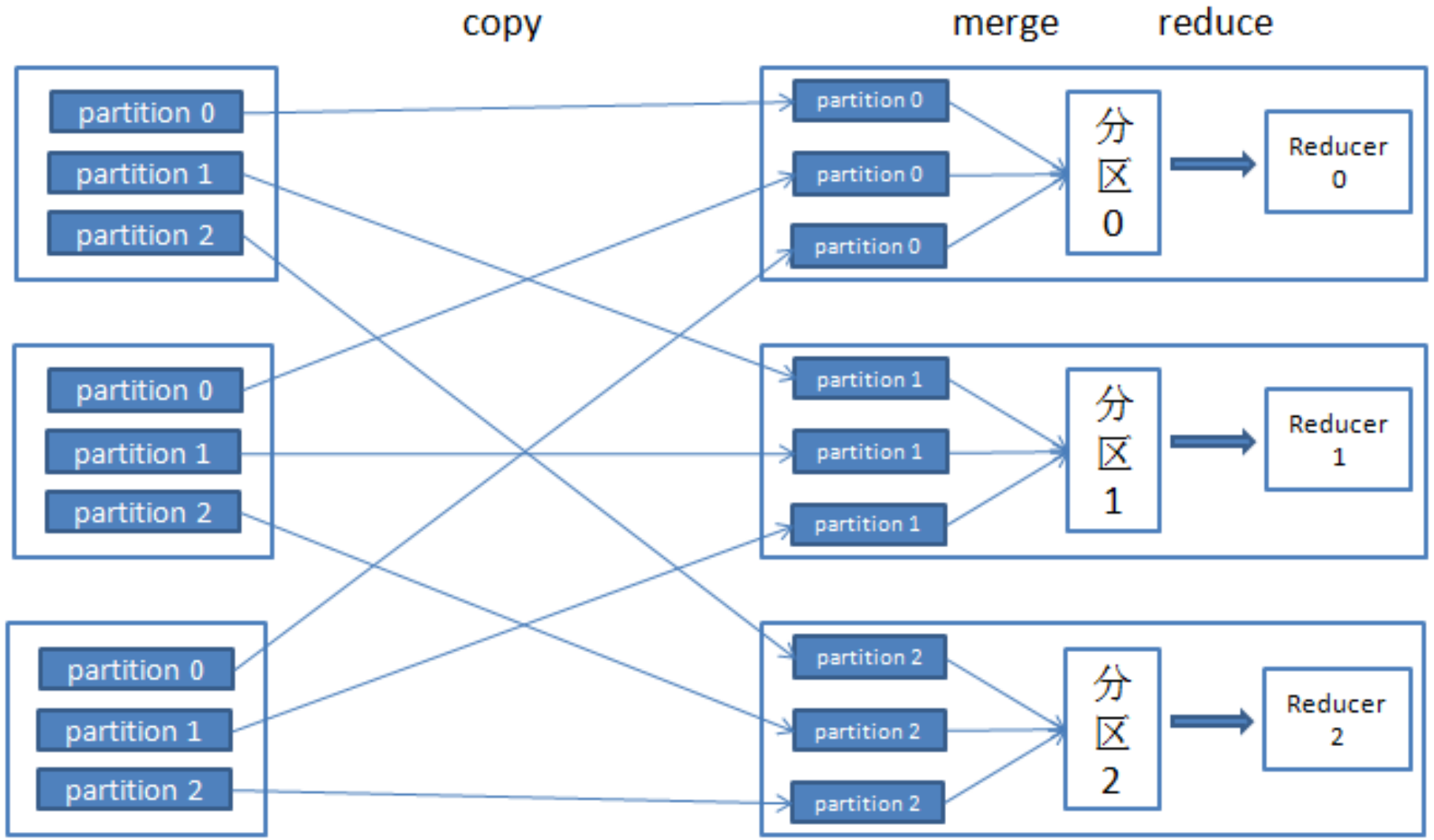
MapReduce计算框架-执行流程



MapReduce计算框架-执行流程



MapReduce计算框架-执行流程



编 程 模 型

- 借鉴函数式的编程方式
- 用户只需要实现两个函数接口：
  - Map(in_key, in_value)

    -> (out_key, intermediate_value) list
  - Reduce (out_key, intermediate_value list)

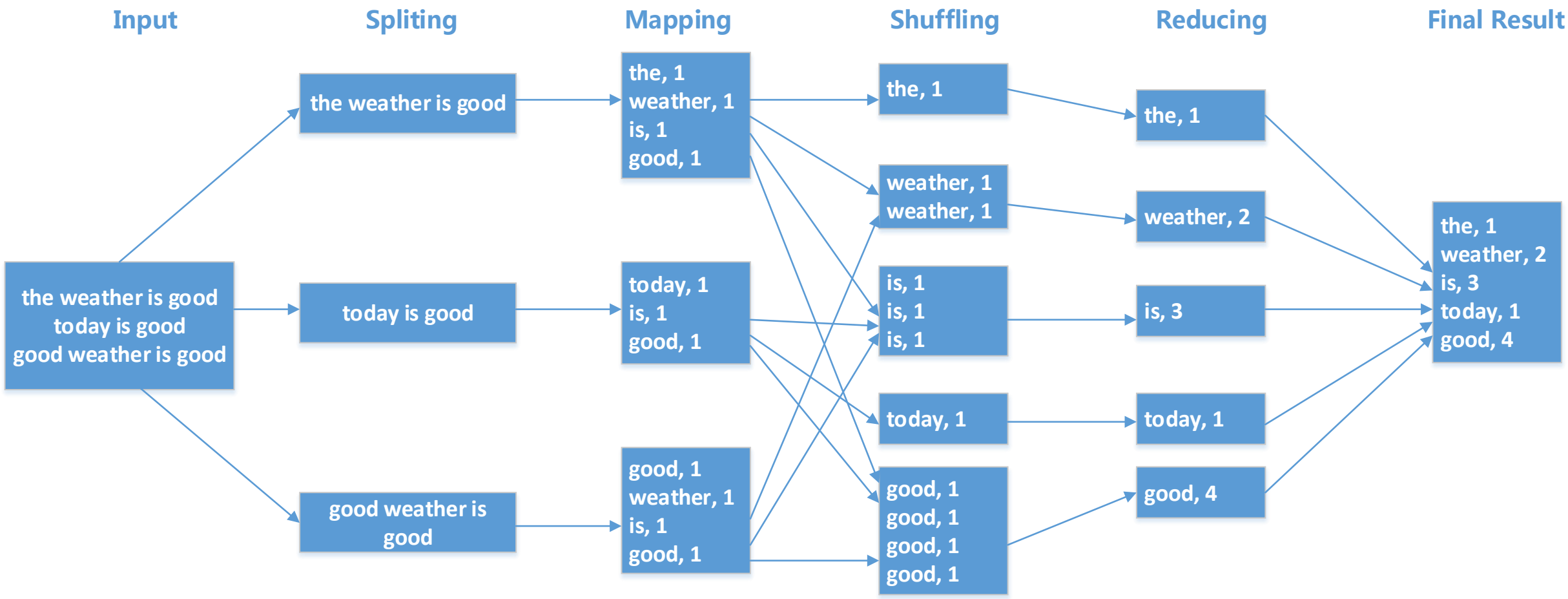
    ->out_value list

编程示例

- WordCount实现过程
- 源数据
  - Document 1
    - the weather is good
  - Document 2
    - today is good
  - Document 3
    - good weather is good

上机实践-WordCount

- 我的第一个MapReduce任务



编 程 示 例

- Map 输出
  - Worker 1
    - (the 1), (weather 1), (is 1), (good 1)
  - Worker 2
    - (today 1), (is 1), (good 1)
  - Worker 3
    - (good 1), (weather 1), (is 1), (good 1)

编 程 示 例

- Reduce 输入
  - Worker 1
    - (the 1)
  - Worker 2
    - (is 1), (is 1), (is 1)
  - Worker 3
    - (weather 1), (weather 1)
  - Worker 4
    - (today 1)
  - Worker 5
    - (good 1), (good 1), (good 1), (good 1)

编程示例

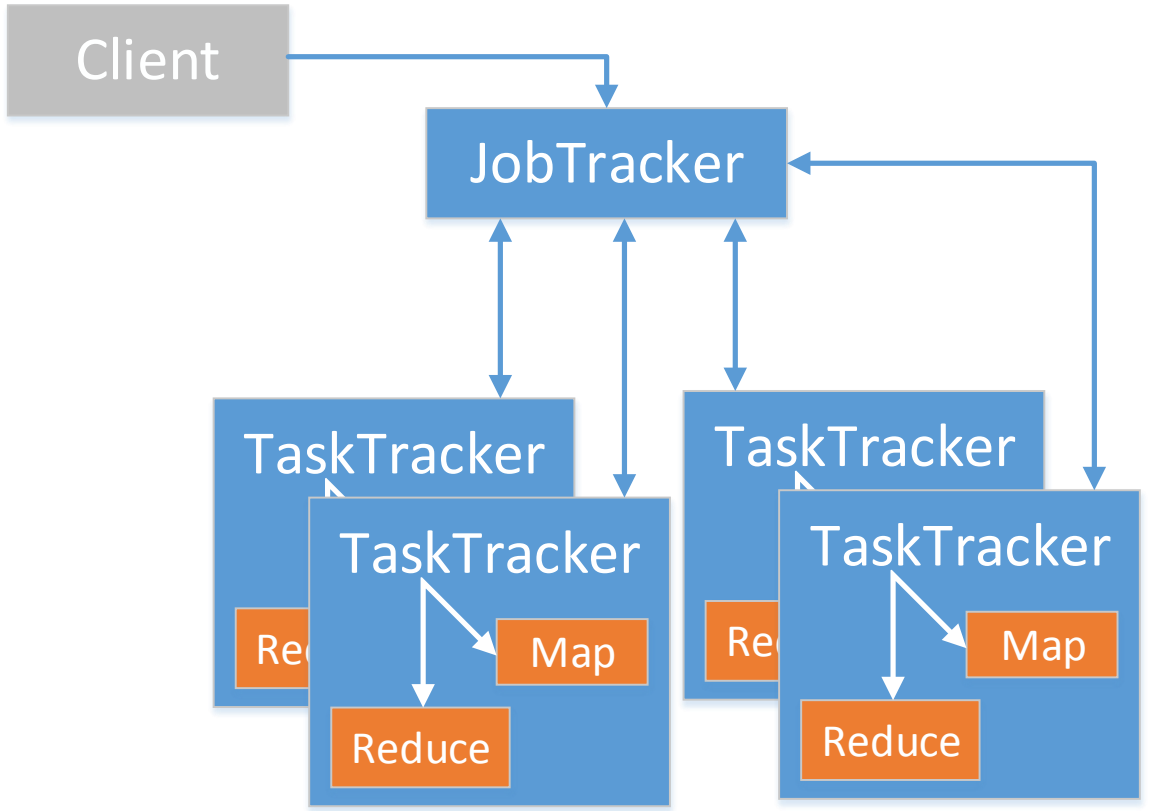
- Reduce 输出
  - Worker 1
    - (the 1)
  - Worker 2
    - (is 3)
  - Worker 3
    - (weather 2)
  - Worker 4
    - (today 1)
  - Worker 5
    - (good 4)

MapReduce实现架构

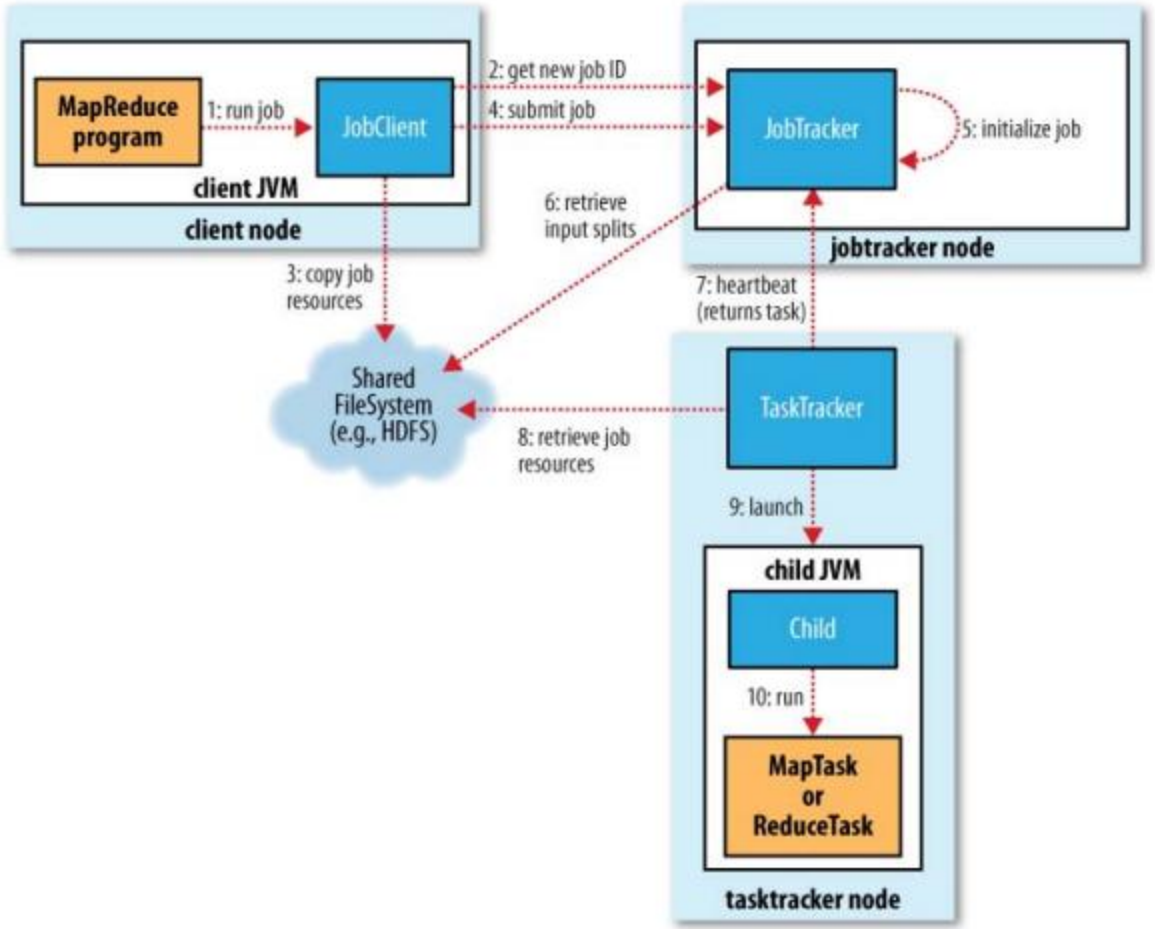
- 两个重要的进程
  - JobTracker
    - 主线程，负责接收客户作业提交，调度任务到作节点上运行，并提供诸如监控工作节点状态及任务进度等管理功能，一个MapReduce集群有一个jobtracker，一般运行在可靠的硬件上。
    - tasktracker是通过周期性的心跳来通知jobtracker其当前的健康状态，每一次心跳包含了可用的map和reduce任务数目、占用的数目以及运行中的任务详细信息。Jobtracker利用一个线程池来同时处理心跳和客户请求。
  - TaskTracker
    - 由jobtracker指派任务，实例化用户程序，在本地执行任务并周期性地向jobtracker汇报状态。在每一个工作节点上永远只会有一个tasktracker

MapReduce工作原理

- JobTracker一直在等待JobClient提交作业
- TashTracker每隔3秒向JobTracker发送心跳询问有没有任务可做，如果有，让其派发任务给它执行
- Slave主动向master拉生意



MapReduce执行流程



MapReduce作业调度

- 默认先进先出队列调度模式（FIFO）
  - 优先级（very_high、high、normal，low，very low）

```
static final Comparator<JobSchedulingInfo> FIFO_JOB_QUEUE_COMPARATOR = new
Comparator<JobSchedulingInfo>() {
  public int compare(JobSchedulingInfo o1, JobSchedulingInfo o2) {
    int res = o1.getPriority().compareTo(o2.getPriority());
    if (res == 0) {
      if (o1.getStartTime() < o2.getStartTime()) {
        res = -1;
      } else {
        res = (o1.getStartTime() == o2.getStartTime() ? 0 : 1);
      }
    }
    if (res == 0) {
      res = o1.getJobID().compareTo(o2.getJobID());
    }
    return res;
  }
};
```

Q & A

@八斗数据