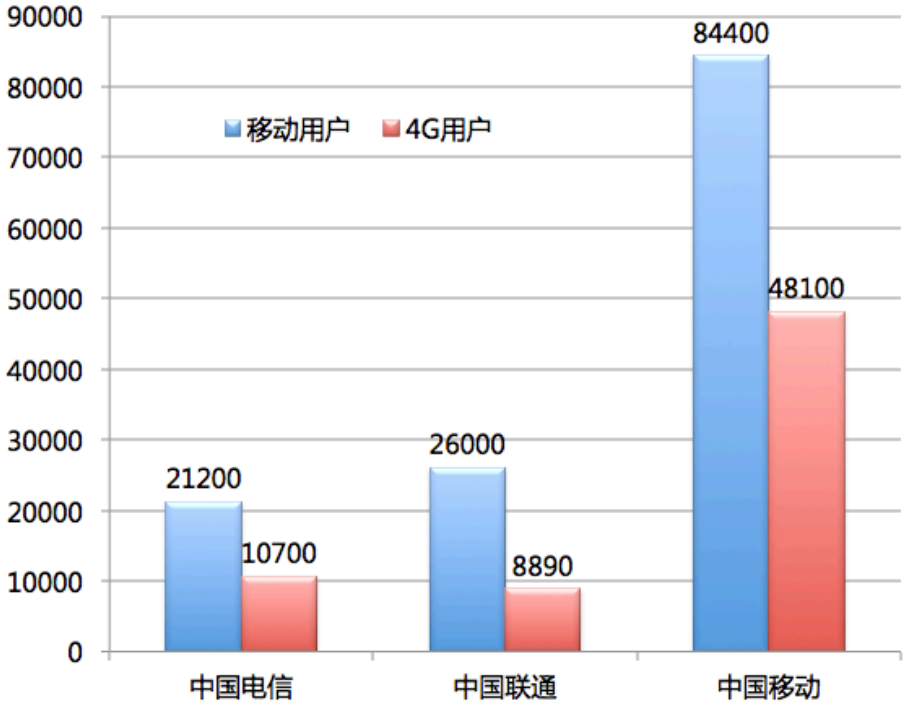
C6. 深入MapReduce

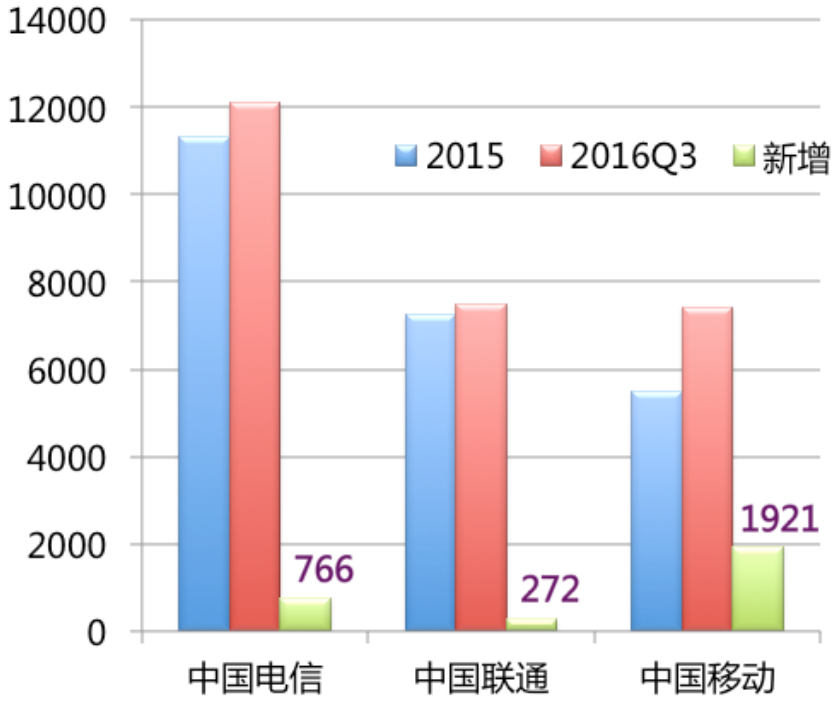
刘军（liujun@bupt.edu.cn）
北京邮电大学 数据科学中心

一周新鲜事

三大运营商2016Q3财报

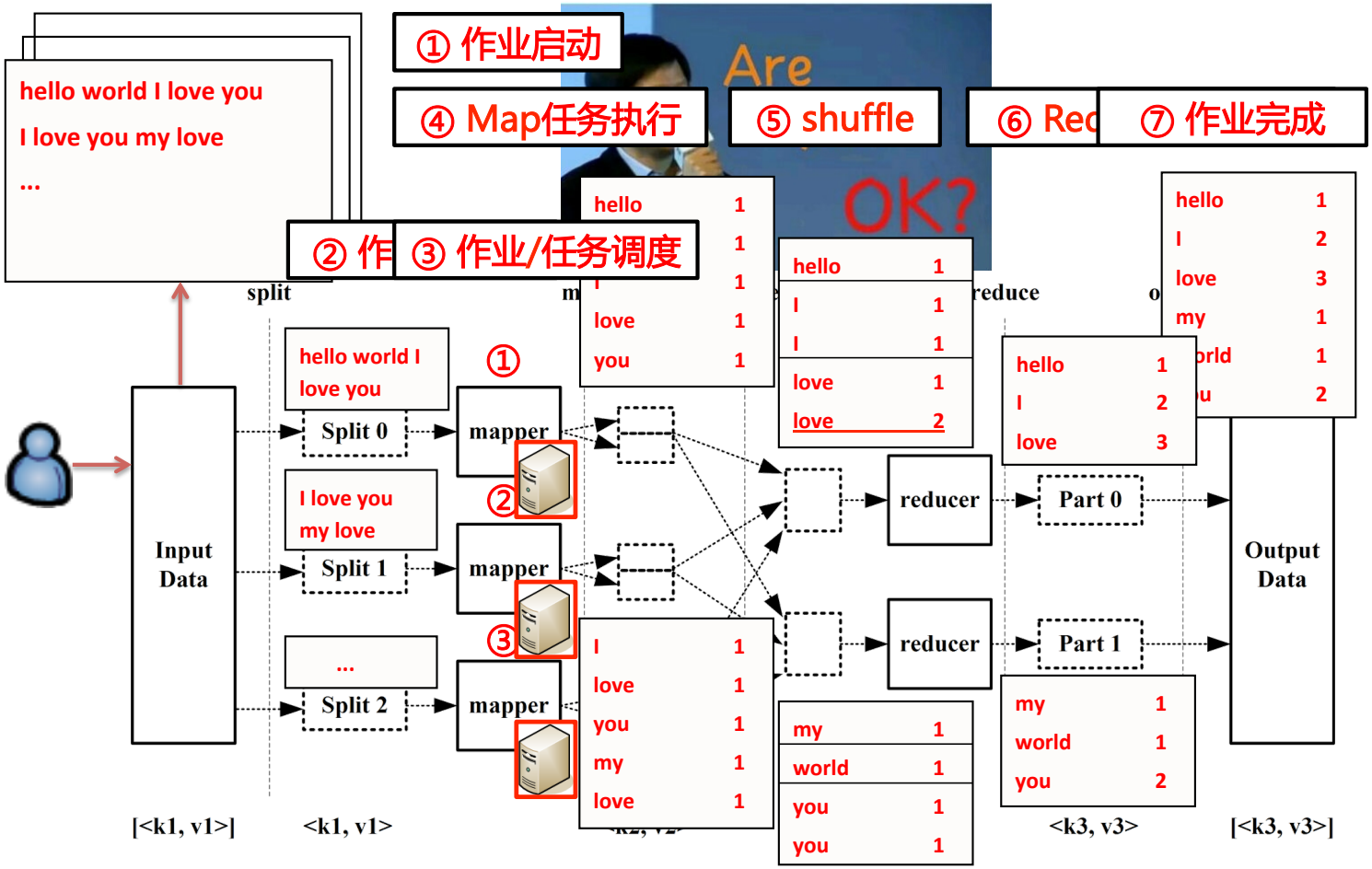


移动用户数

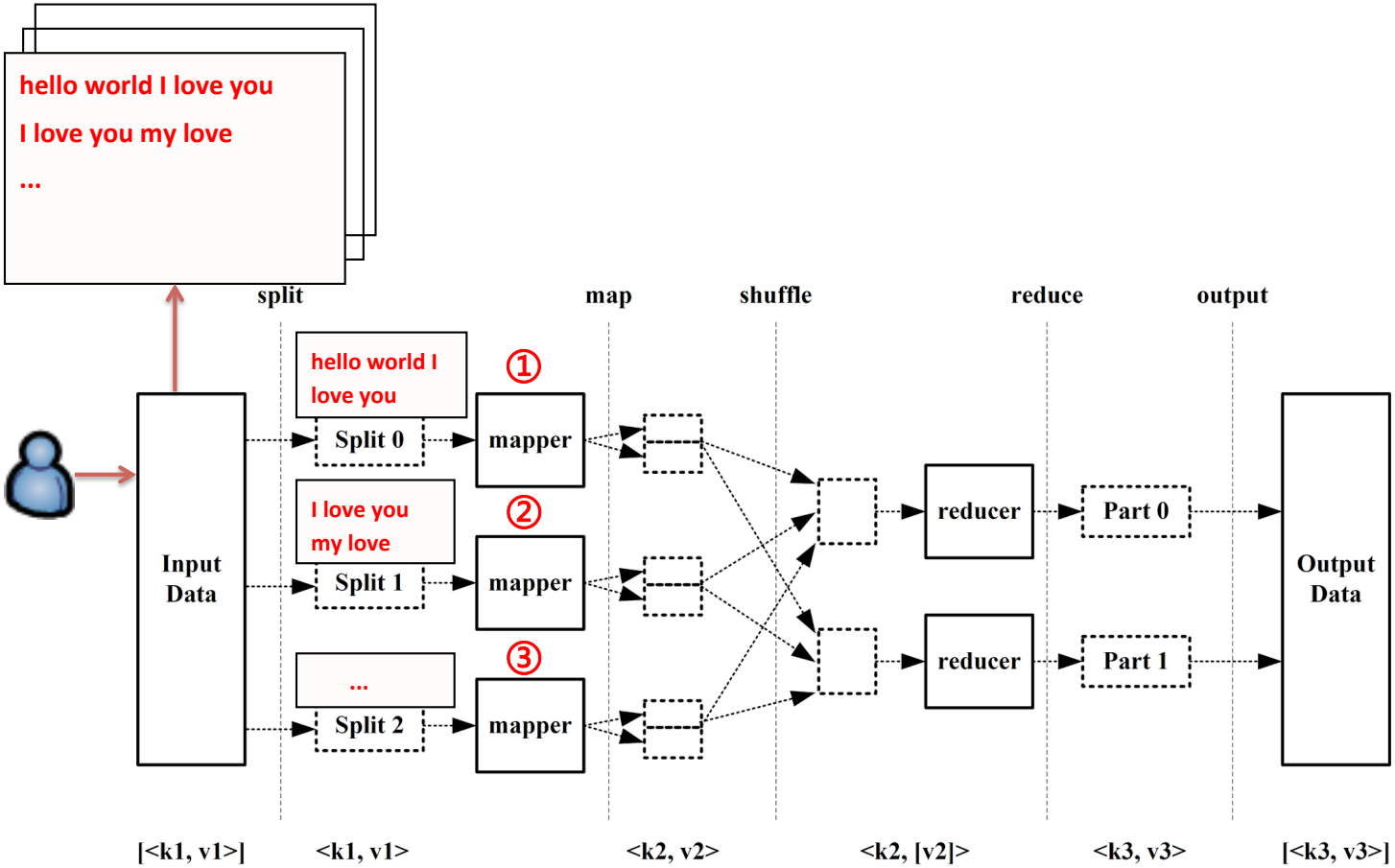


有线宽带用户数

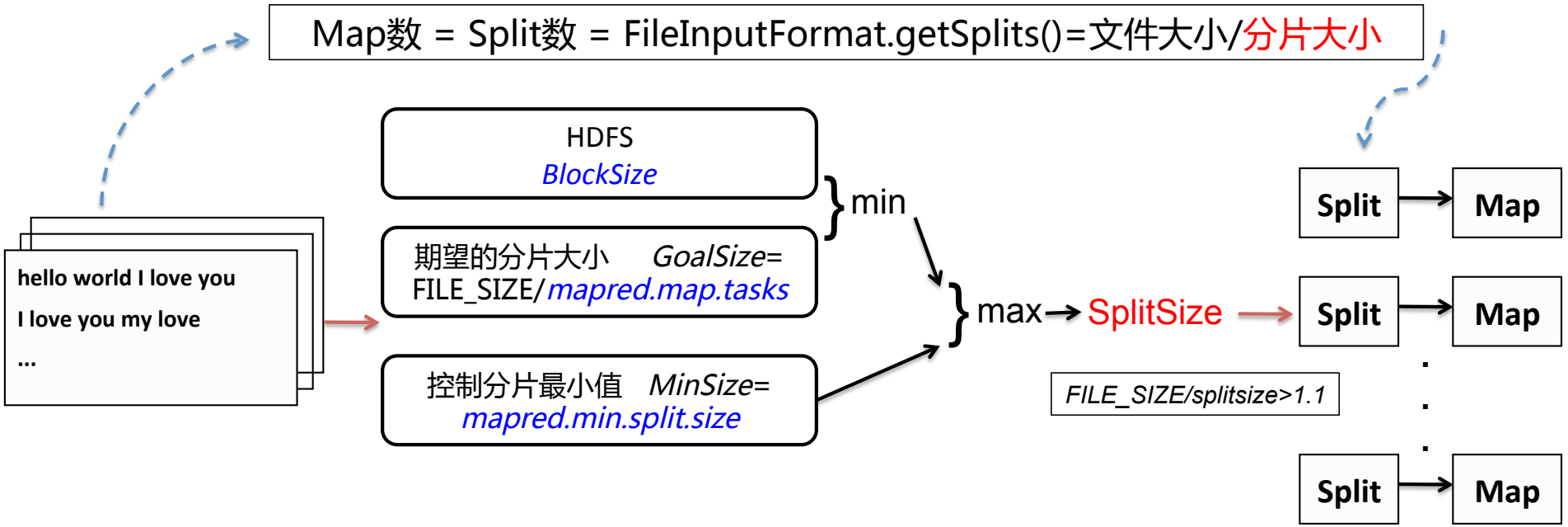
重温MapReduce完整过程



2. 作业初始化 - 数据怎么分片？会有多少个Map？

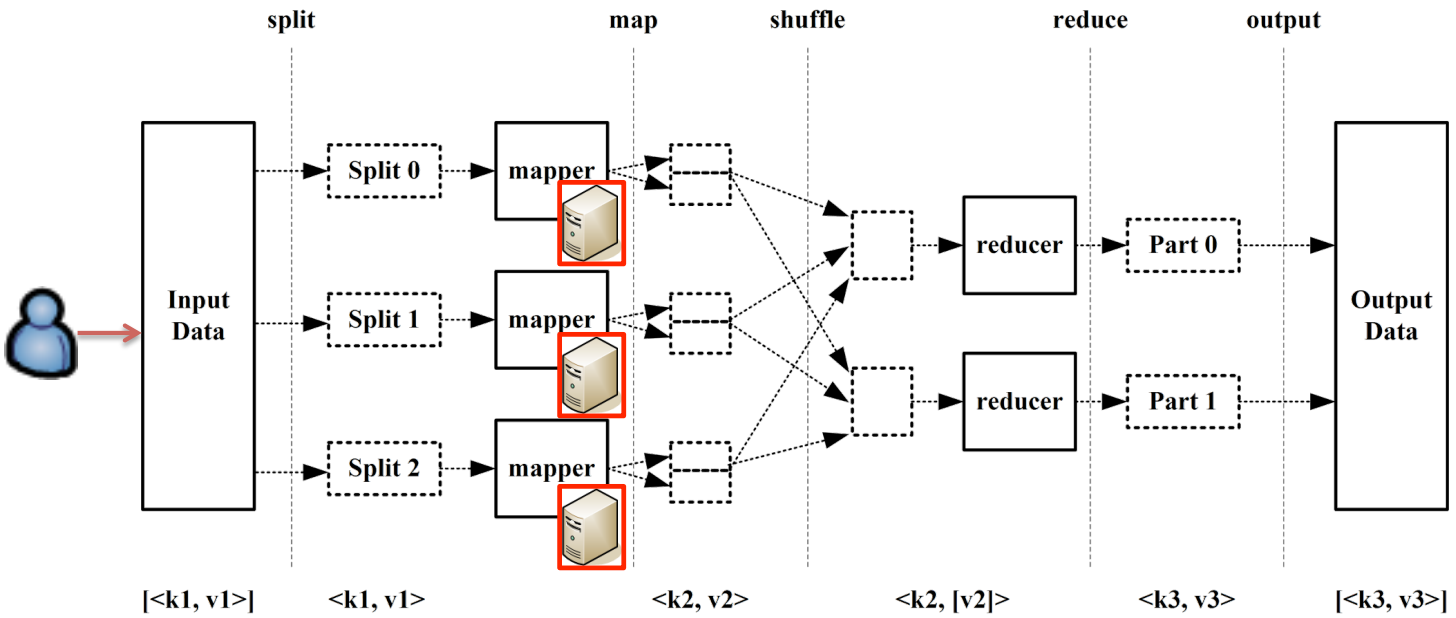


通过控制Split数改变Map并行度

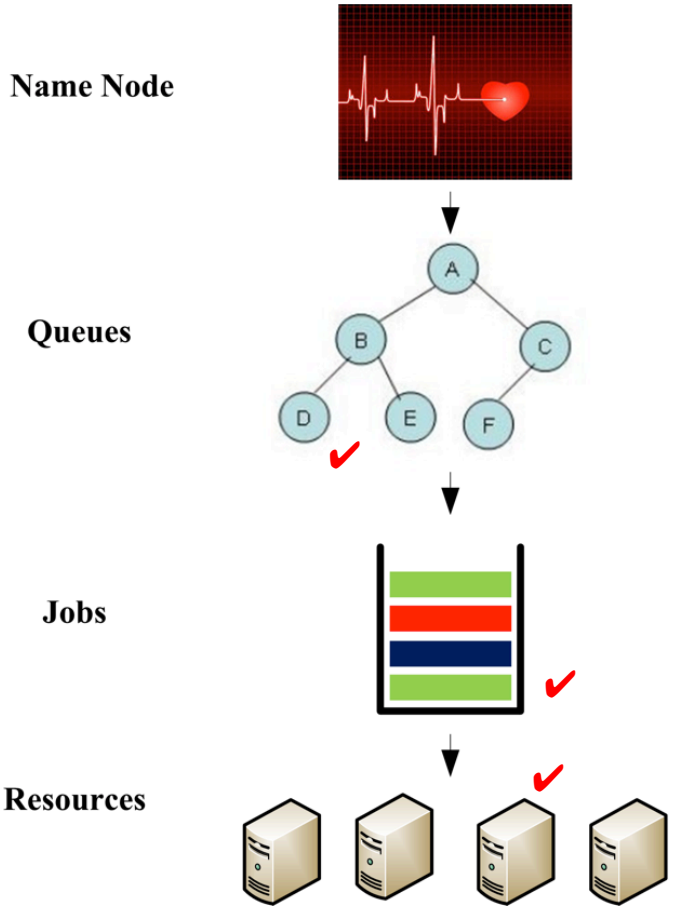


- 示例（默认配置64MB）：
  - 一个100M文件，Split数几个？每个Split多大？
  - 一个129M文件，Split数几个？每个Split多大？
  - 10个10M文件，Map数是多少？
  - **1个100M文件，如何将Map数增大到10？**

3. 作业调度 - 如何调度？


split
map
shuffle
reduce
output
Input Data
Split 0
Split 1
Split 2
mapper
mapper
mapper
reducer
reducer
Part 0
Part 1
Output Data
[<k1, v1>]
<k1, v1>
<k2, v2>
<k2, [v2]>
<k3, v3>
[<k3, v3>]

调度原理及调度机制

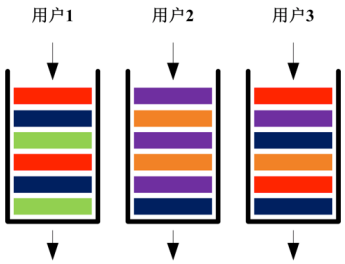


FIFO

简单

区分优先级

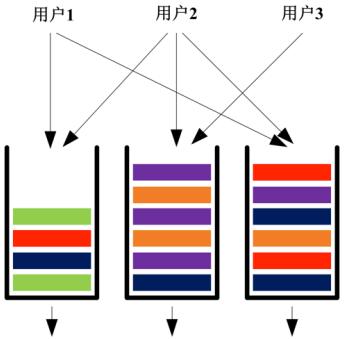
Fair



用户公平

按用户分配资源

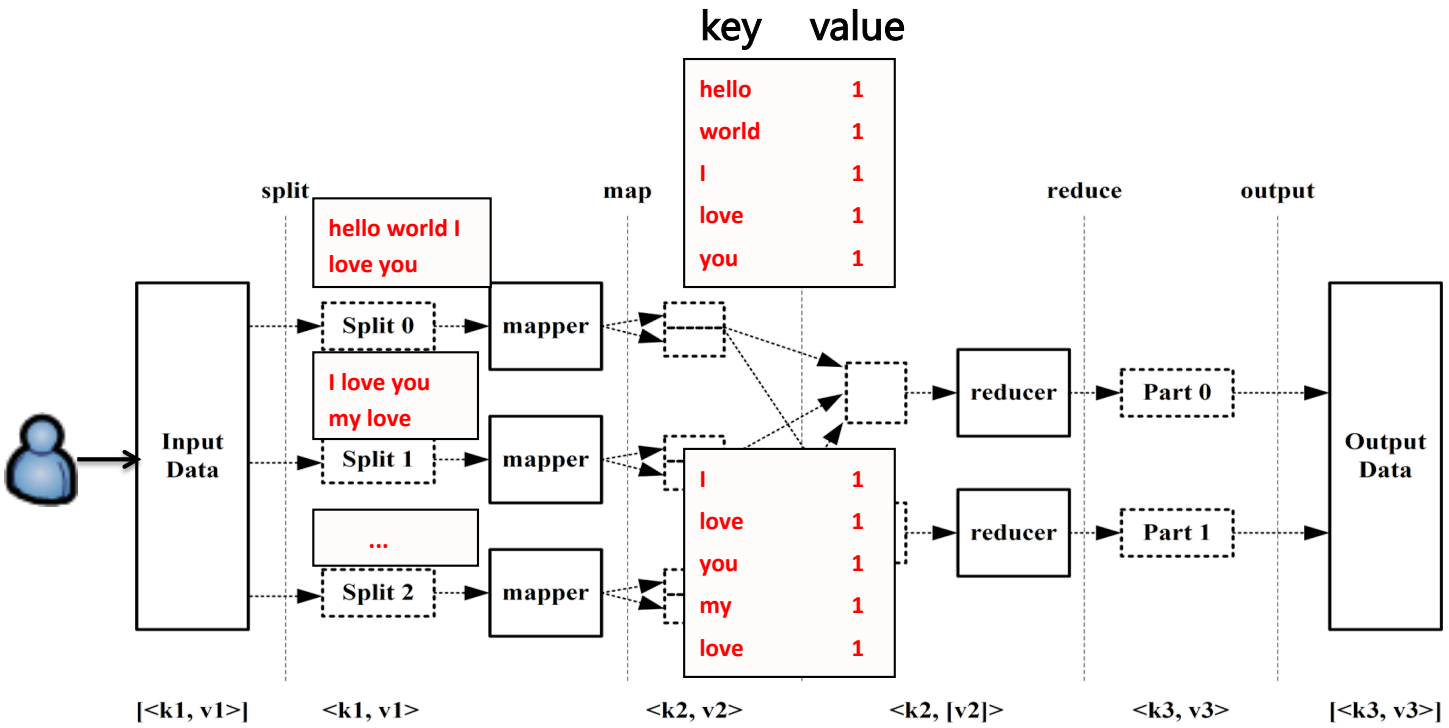
Capacity



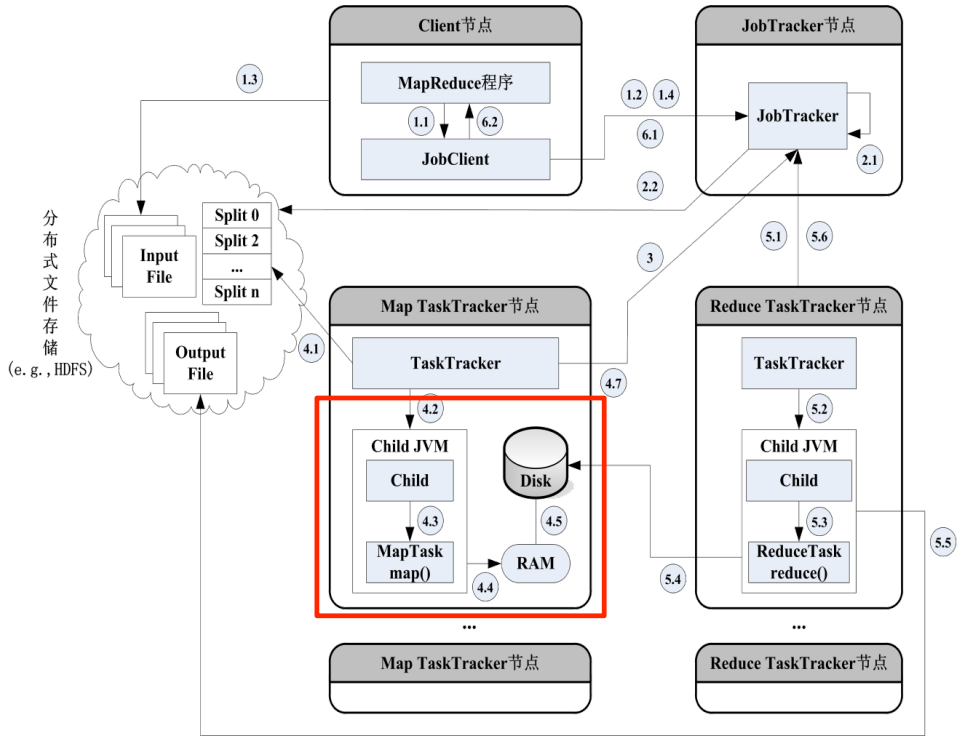
集群效率

先执行负载轻的队列

4. Map执行 - 为什么比原来的分布式并行计算快？

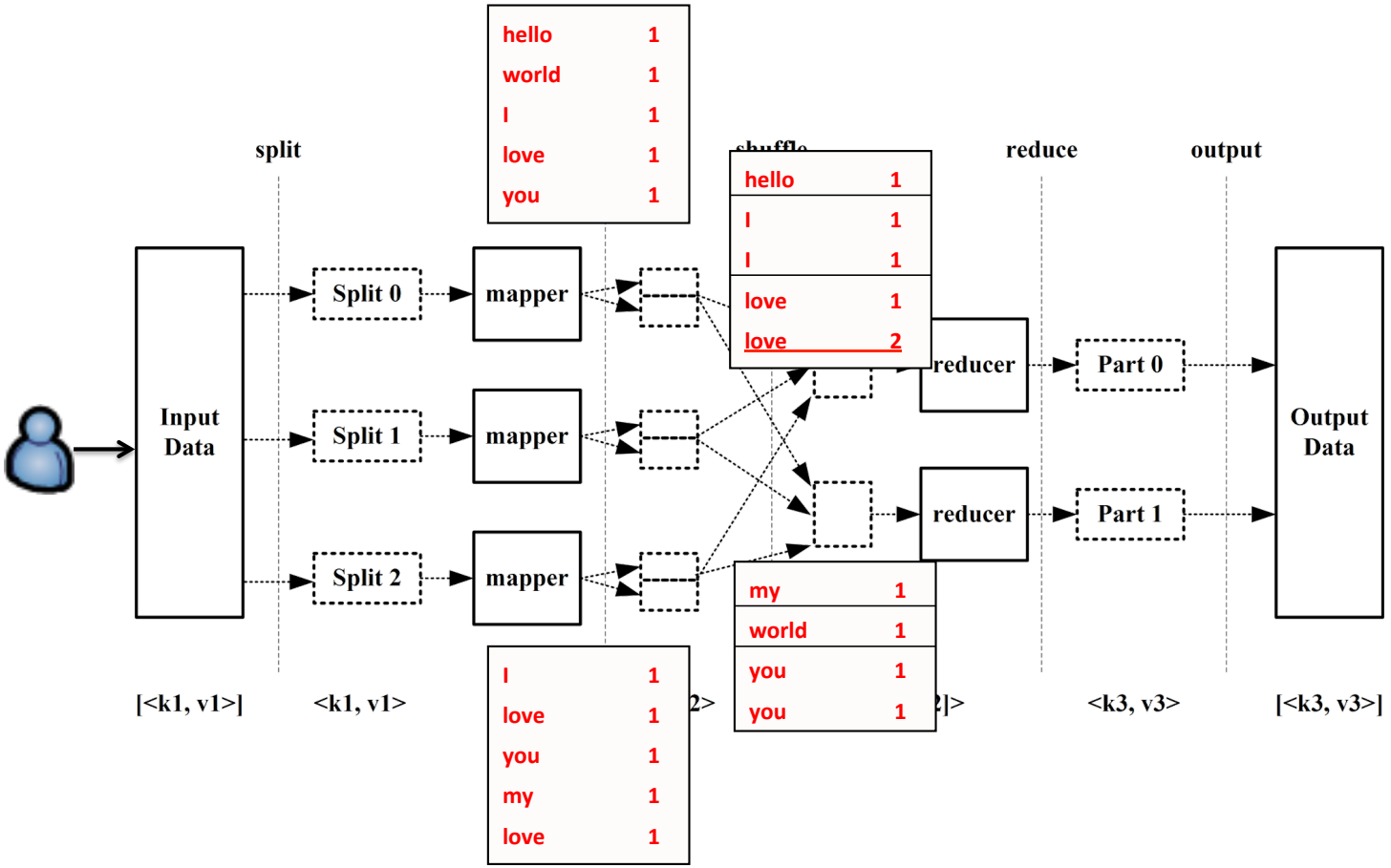


计算靠近数据

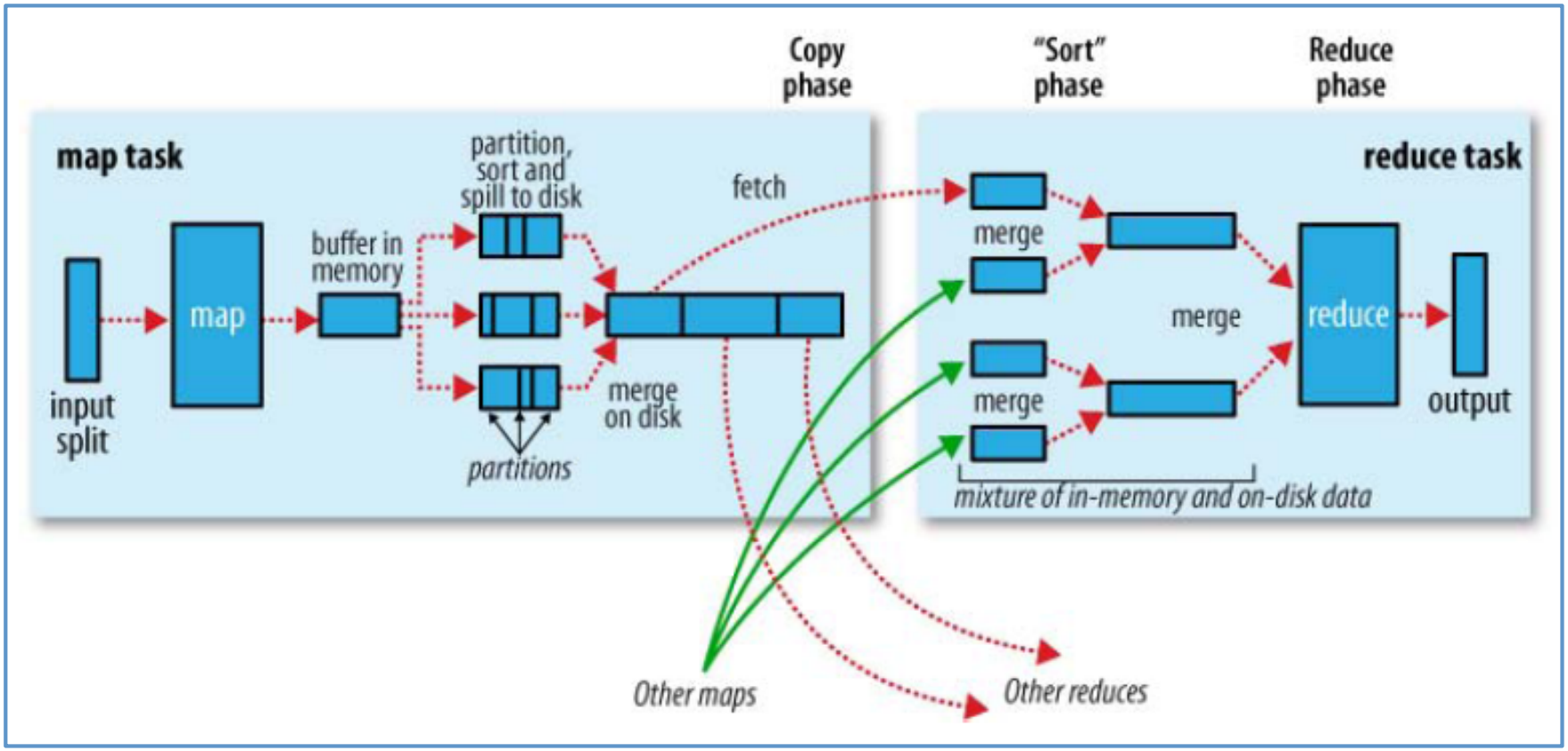


- **大规模数据处理时，外存文件数据I/O访问会成为一个制约系统性能的瓶颈**
- **代码靠近数据**
  - 原则：本地化数据处理（locality），即一个计算节点尽可能处理其本地磁盘上所存储的数据
  - 尽量选择数据所在DN启动Map任务
  - 减少数据通信，提高计算效率
- **数据靠近代码**
  - 当本地没有数据处理时，尽可能从同一机架或最近的其他节点传输数据进行处理

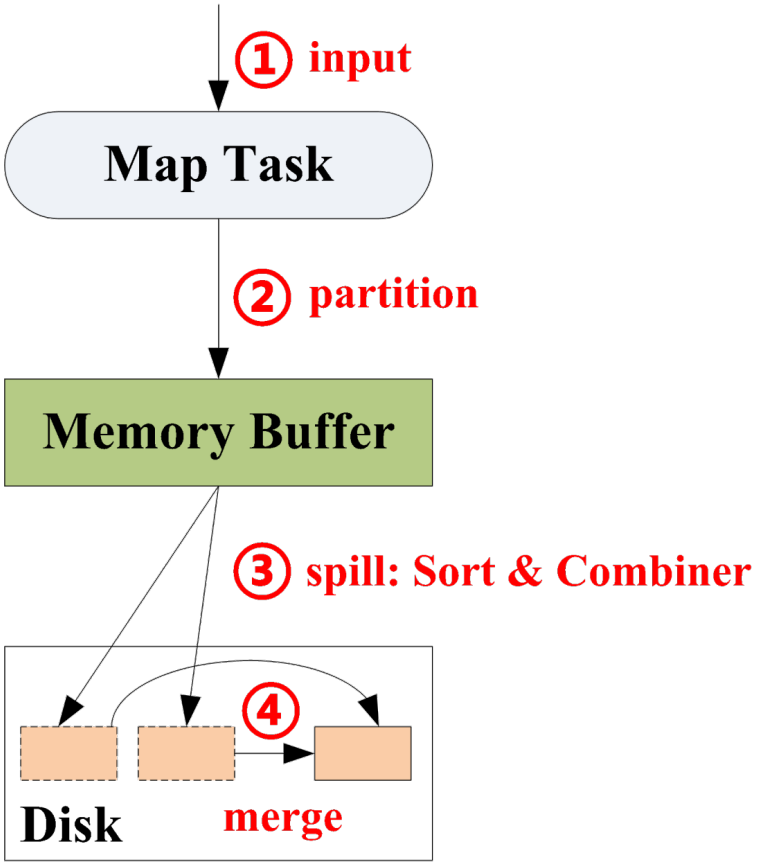
5. Shuffle - 这中间发生了什么？



Shuffle概览


Copy phase
"Sort" phase
Reduce phase
map task
reduce task
partition, sort and spill to disk
fetch
buffer in memory
map
input split
merge on disk
partitions
merge
merge
merge
reduce
output
mixture of in-memory and on-disk data
Other maps
Other reduces

Map端处理过程



① 处理输入数据

② 为对应的Reduce生成结果（Partition）

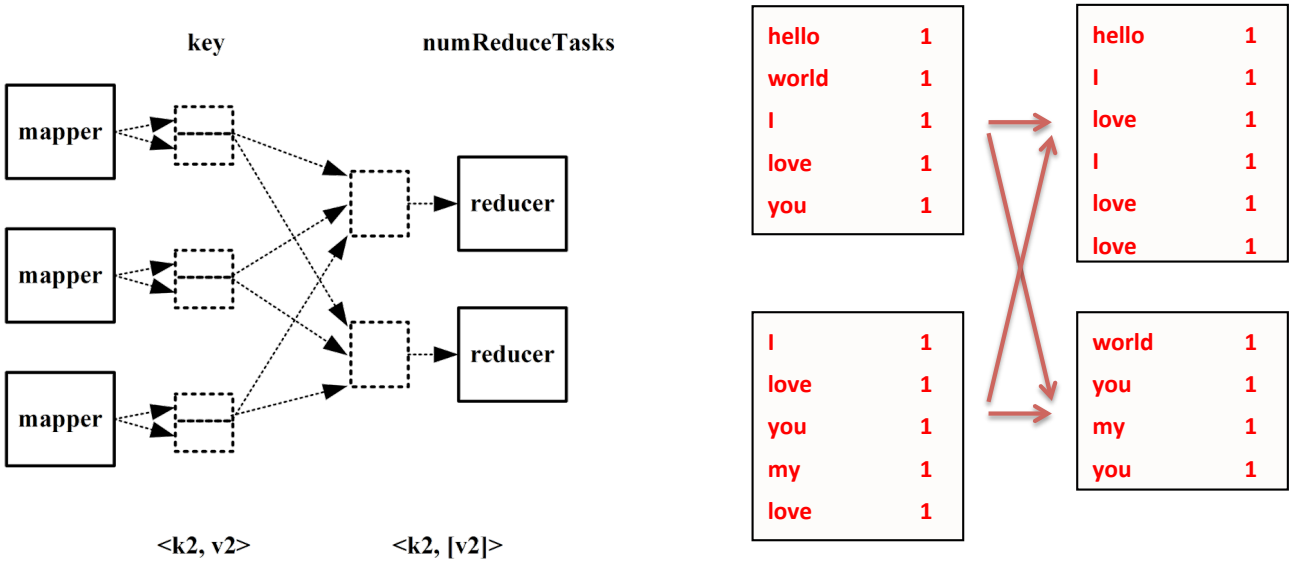
③ 内存数据溢出到磁盘（Spill）
- Sort
- Combiner

④ 合并中间结果文件（Merge）

Partition

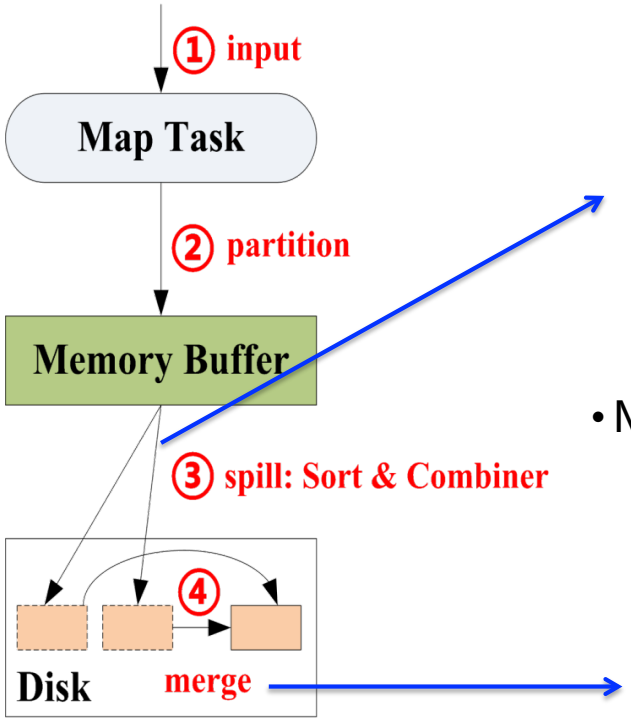
- 作用：将map的结果发送到相应的reduce
- 要求：负载均衡、效率
- 默认HashPartitioner

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

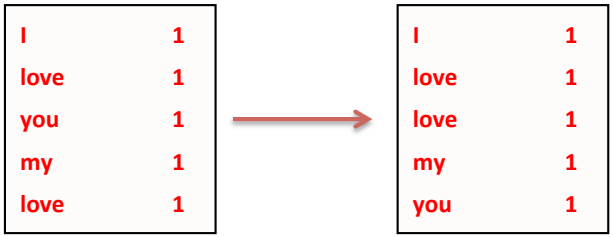


- 可自定义：job.setPartitionerClass(MyPartitioner.class);

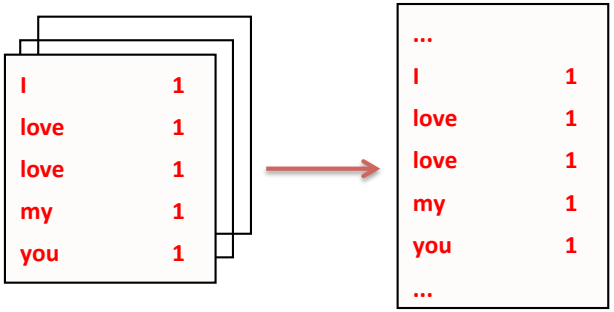
Sort



- Map后的第1次排序：文件内部快速排序（Sort）
  - 每次spill时，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

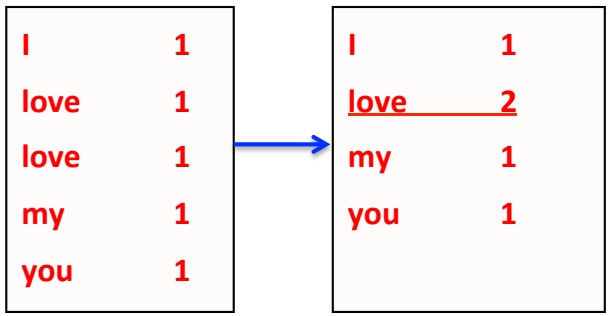


- Map后的第2次排序：多个文件归并排序（Merge）
  - Map任务执行完成后会对这些内部排好序的文件做一次归并排序，并将排好序的结果输出到一个大的文件中



Combine

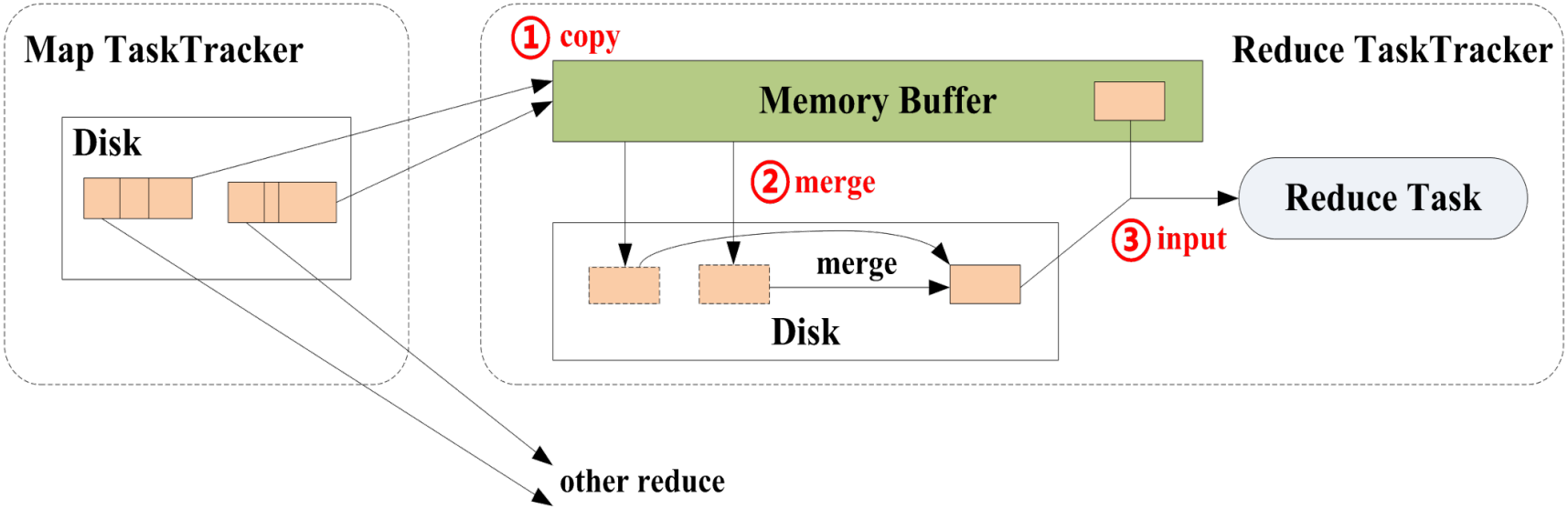
- 作用：合并Map输出的中间数据，减少数据传输、提高处理效率




```
public static void main(String[] args) throws Exception{
   ...
   job.setMapperClass(TokenizerMapper.class);
   job.setCombinerClass(IntSumReducer.class);
   job.setReducerClass(IntSumReducer.class)
   ...
}
```

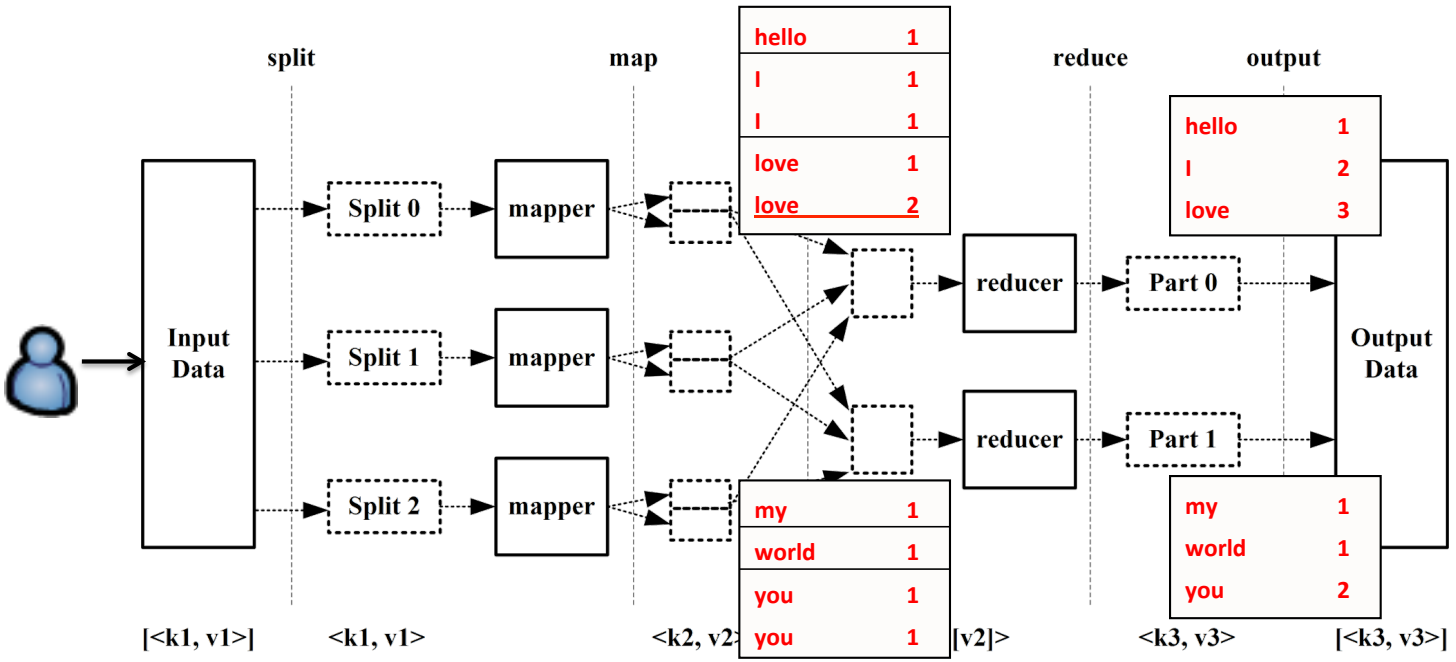
```
public static class IntSumReducer extends Reducer
   <Text, IntWritable, Text, IntWritable>{
   private IntWritable result = new IntWritable();
   public void reduce(Text key, Iterable<IntWritable>
      values,  Context context) throws IOException{
      int sum = 0;
      for (IntWritable val : values) {
         sum += val.get();}
      result.set(sum);
      context.write(key, result);
} }
```

Reduce端拉取数据



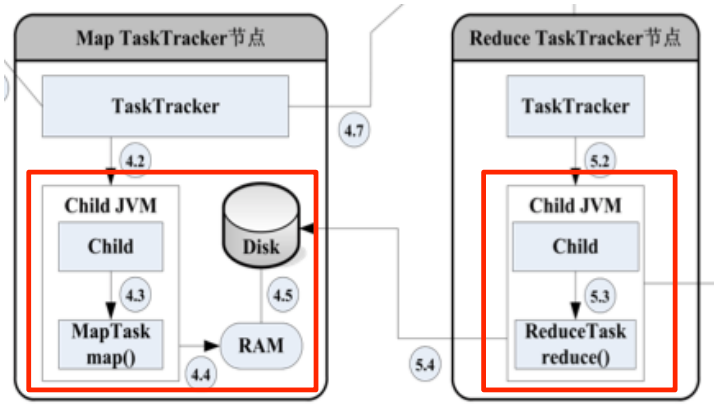
① 拉取数据（Copy）
② 合并中间结果文件（Merge）
③ 处理数据

6. 计算过程中如果出错了怎么办？



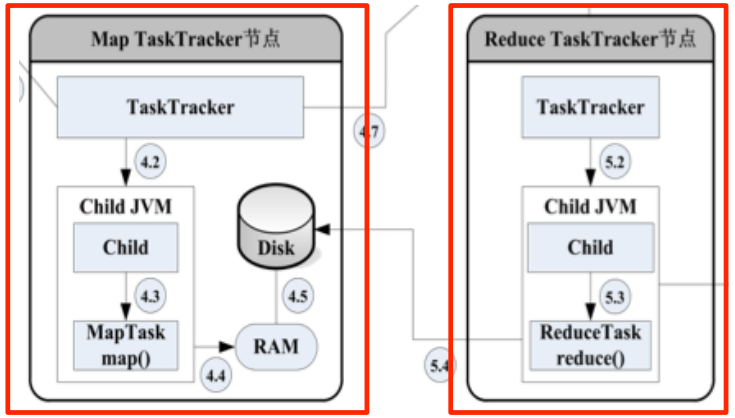
计算容错：Task出错

- Task异常：失败（failed）和终止（killed）
- 失败（failed）
  - 失败原因：
    - ✓ map或reduce函数代码不正确
    - ✓ 任务所在的JVM出现运行异常
    - ✓ 任务进度更新超时
  - 失败处理：
    - ✓ TaskTracker将此任务的失败信息报告给JobTracker
    - ✓ JobTracker分配新的节点执行此任务
    - ✓ 如果同一个任务出现多次失败，且失败次数超过由参数指定的最大次数时，作业会在未完成的情况下被终止
      - mapred.max.map.attempts
      - mapred.reduce.max.attempts
- 终止（killed）
  - Speculative execution，避免慢Task拖慢整个job
  - TaskTracker挂了，该节点上的任务被标记为killed



计算容错：TaskTracker出错

- TaskTracker与JobTracker间的心跳监测
  - mapred.tasktracker.expiry.interval，默认10分钟
  - 已完成的任务会正常返回，未完成的任务则重新分配TaskTracker节点执行



- 黑名单机制
  - Job黑名单：每个Job会维护一个TaskTracker黑名单
  - 集群黑名单：整个集群的TaskTracker黑名单
    - ✓ 当一个Job成功结束时，对该Job黑名单中的tasktracker做如下三个判断：
      - 该TaskTracker被4个Job加入了黑名单（mapred.max.tracker.blacklists）
      - 该TaskTracker被加入Job黑名单的次数，超过了集群中所有tasktracker被加入Job黑名单平均次数的50%（mapred.cluster.average.blacklist.threshold）
      - 已经加入黑名单的TaskTracker个数不超过集群总TaskTracker的50%
    - ✓ 以上三条如果均满足，则将Job黑名单中的该TaskTracker加入集群黑名单，以后将不在该TaskTracker上调度任何task
  - 恢复：重启TaskTracker

下周课前请准备（理论课）

- 阅读《Hadoop大数据处理》以下章节：
  - 3.4 MapReduce设计模式
  - 3.5 MapReduce算法实践

- 在课程平台观看第7章《MapReduce设计模式》

刘军
北京邮电大学 数据科学中心
北邮本部 明光楼七层
邮件地址：liujun@bupt.edu.cn
新浪微博：北邮刘军
电　　话：010-62283742