MapReduce 2.0应用场景、原理与基本架构

讲师：董西成
博客：dongxicheng.org
微信号：hadoop-123
（二维码见右）

Open Passion Value

目录

MapReduce的定义

- 源自于Google的MapReduce论文
  - ✓ 发表于2004年12月
  - ✓ Hadoop MapReduce是Google MapReduce克隆版
- MapReduce特点
  - ✓ 易于编程
  - ✓ 良好的扩展性
  - ✓ 高容错性
  - ✓ 适合PB级以上海量数据的离线处理

MapReduce的特色—不擅长的方面

- 实时计算
  - ✓像MySQL一样，在毫秒级或者秒级内返回结果
- 流式计算
  - ✓MapReduce的输入数据集是静态的，不能动态变化
  - ✓MapReduce自身的设计特点决定了数据源必须是静态的
- DAG计算
  - ✓多个应用程序存在依赖关系，后一个应用程序的输入为前一个的输出

目录

MapReduce的实例—Wordcount

- 场景：有大量文件，里面存储了单词，且一个单词占一行
- 任务：如何统计每个单词出现的次数？
- 类似应用场景：
  - ✓搜索引擎中，统计最流行的K个搜索词；
  - ✓统计搜索词频率，帮助优化搜索词提示

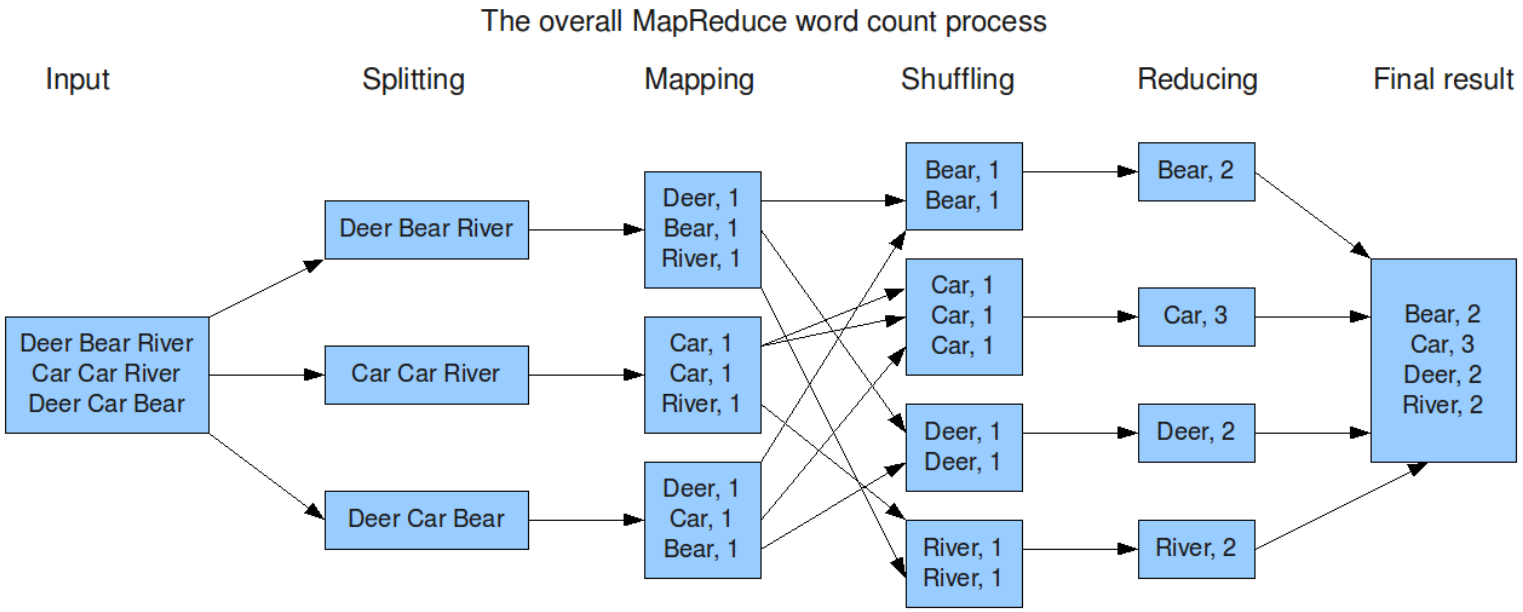
MapReduce的实例—Wordcount

- Case 1：整个文件可以加载到内存中；
  - ✓ sort datafile | uniq -c
- Case 2：文件太大不能加载到内存中，但<word, count>可以存放到内存中；
- Case 3：文件太大无法加载到内存中，且<word, count>也不行

MapReduce的实例—Wordcount

- 将问题范化为：有一批文件（规模为TB级或者 PB级），如何统计这些文件中所有单词出现的次数；
- 方案：首先，分别统计每个文件中单词出现次数，然后累加不同文件中同一个单词出现次数；
- 典型的MapReduce过程。

MapReduce编程模型—WordCount



MapReduce编程模型—WordCount

- Input: 一系列key/value对
- 用户提供两个函数实现:
  - ✓map(k,v) → list(k1,v1)
  - ✓reduce(k1, list(v1)) → v2
- (k1,v1) 是中间key/value结果对
- Output：一系列(k2,v2)对

MapReduce编程模型—WordCount

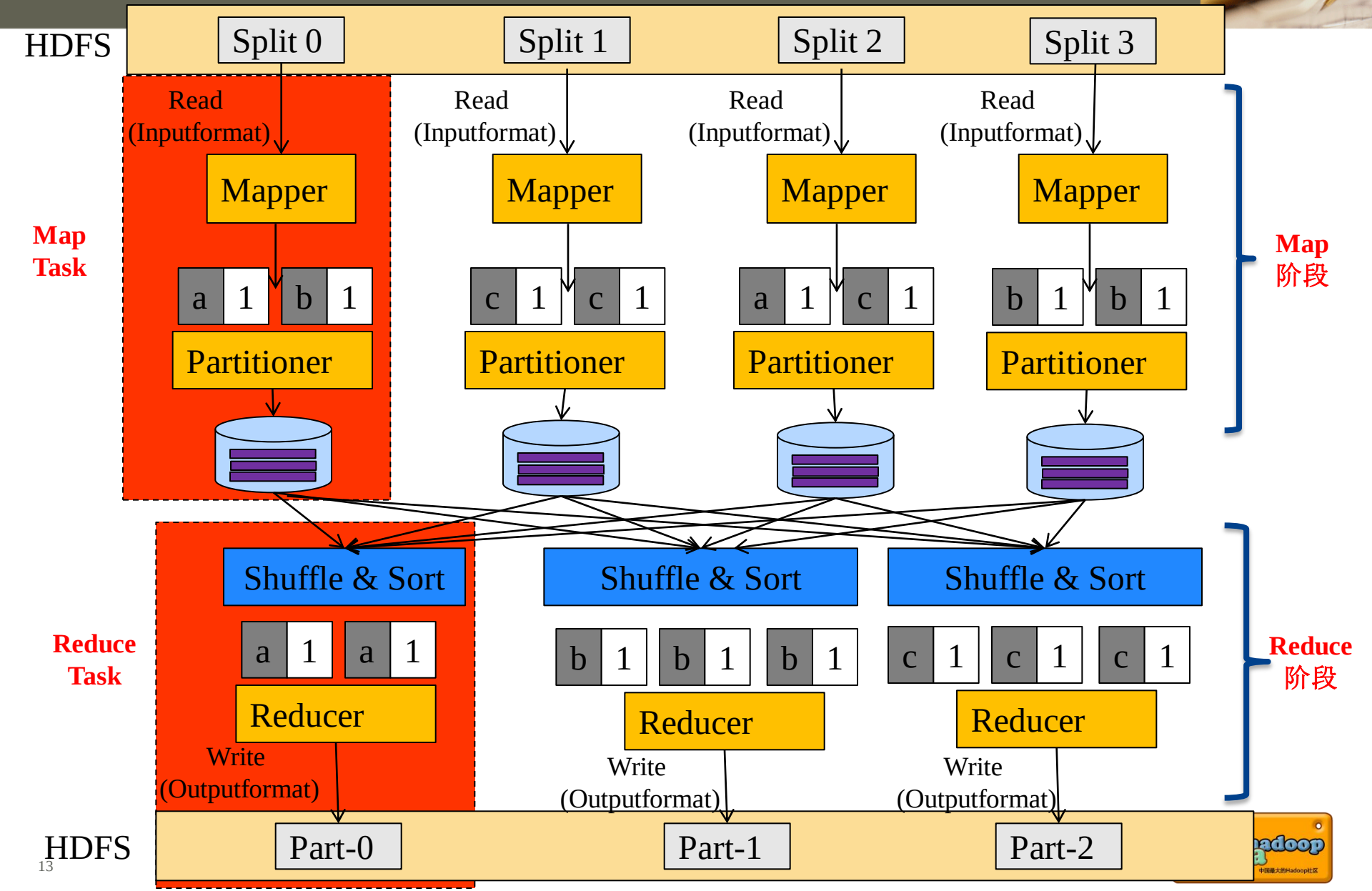
```
map(key, value):
// key: document name; value: text of document
  for each word w in value:
          emit(w, 1)


reduce(key, values):
// key: a word; values: an iterator over counts
          result = 0
          for each count v in values:
                  result += v
          emit(key,result)
```

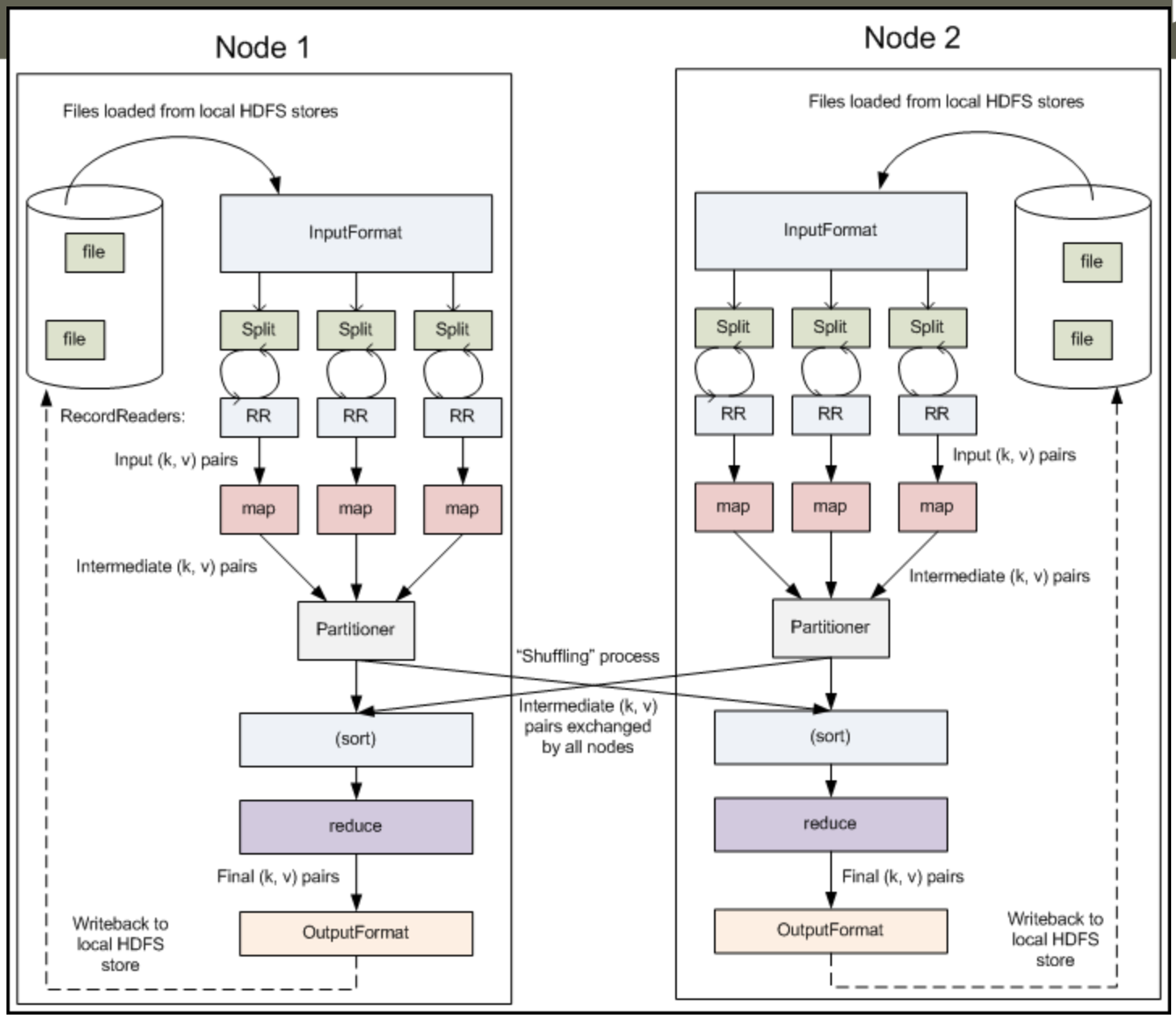
MapReduce编程模型

- MapReduce将作业的整个运行过程分为两个阶段：Map阶段和Reduce阶段
- Map阶段由一定数量的Map Task组成
  - ✓ 输入数据格式解析：InputFormat
  - ✓ 输入数据处理：Mapper
  - ✓ 数据分组：Partitioner
- Reduce阶段由一定数量的Reduce Task组成
  - ✓ 数据远程拷贝
  - ✓ 数据按照key排序
  - ✓ 数据处理：Reducer
  - ✓ 数据输出格式：OutputFormat

MapReduce编程模型—内部逻辑



MapReduce编程模型—外部物理结构



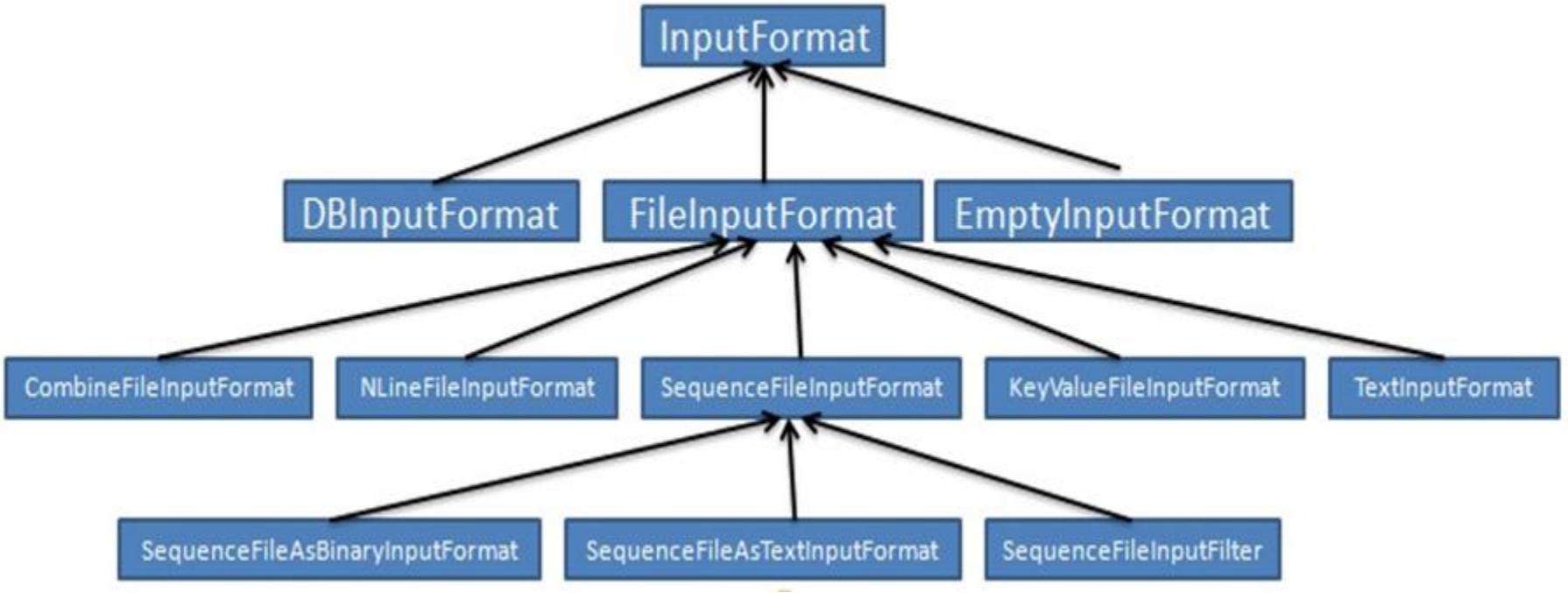
MapReduce编程模型—InputFormat

- 文件分片（InputSplit）方法
  - ✓处理跨行问题
- 将分片数据解析成key/value对
  - ✓默认实现是TextInputFormat
- TextInputFormat
  - ✓Key是行在文件中的偏移量，value是行内容
  - ✓若行被截断，则读取下一个block的前几个字符

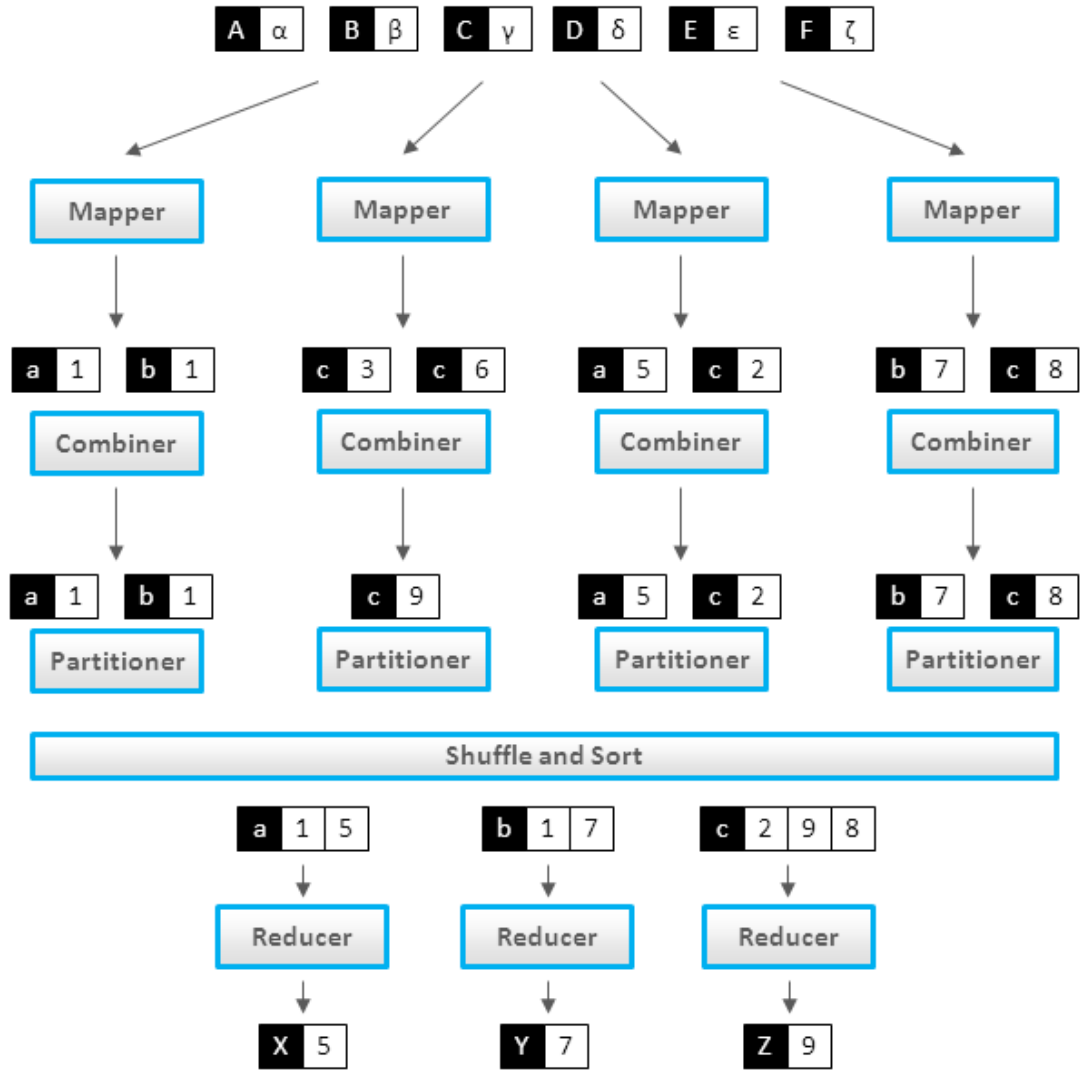
MapReduce编程模型—Split与Block

- **Block**
  - ✓HDFS中最小的数据存储单位
  - ✓默认是64MB
- **Spit**
  - ✓MapReduce中最小的计算单元
  - ✓默认与Block一一对应
- **Block与Split**
  - ✓Split与Block是对应关系是任意的，可由用户控制

MapReduce编程模型—InputFormat



MapReduce编程模型—Combiner



MapReduce编程模型—Combiner

- **Combiner可做看local reducer**
  - ✓合并相同的key对应的value（wordcount例子）
  - ✓通常与Reducer逻辑一样
- **好处**
  - ✓减少Map Task输出数据量（磁盘IO）
  - ✓减少Reduce-Map网络传输数据量(网络IO)
- **如何正确使用**
  - ✓结果可叠加
  - ✓Sum(YES!)，Average（NO!）

MapReduce编程模型—Partitioner

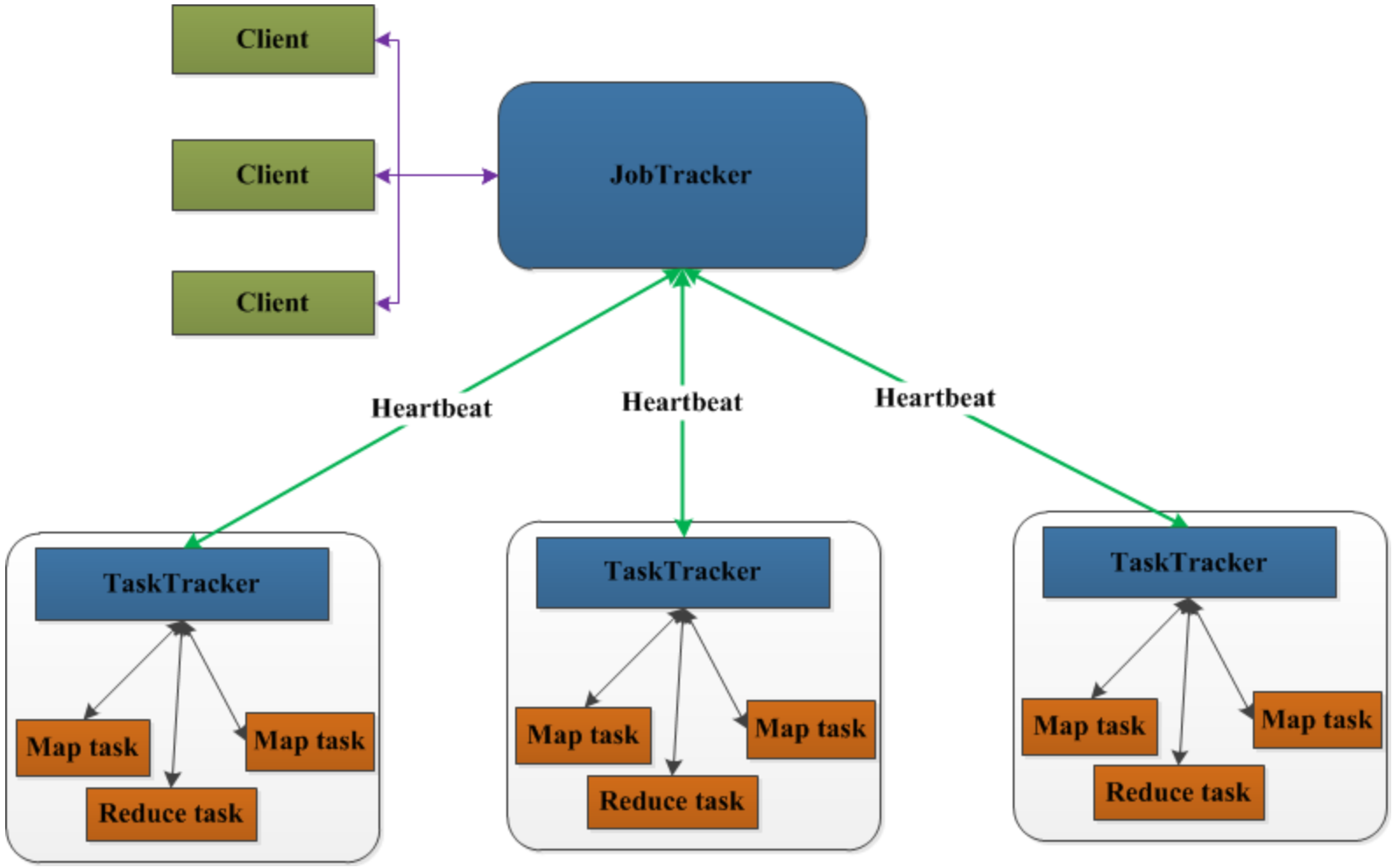
- **Partitioner决定了Map Task输出的每条数据交给哪个Reduce Task处理**
- **默认实现：hash(key) mod R**
  - ✓**R是Reduce Task数目**
  - ✓**允许用户自定义**
- **很多情况需自定义Partitioner**
  - ✓**比如“hash(hostname(URL)) mod R”确保相同域名的网页交给同一个Reduce Task处理**

MapReduce编程模型

- **Map阶段**
  - ✓**InputFormat（默认TextInputFormat）**
  - ✓**Mapper**
  - ✓**Combiner（local reducer）**
  - ✓**Partitioner**
- **Reduce阶段**
  - ✓**Reducer**
  - ✓**OutputFormat（默认TextOutputFormat）**

目录

MapReduce 1.0架构



MapReduce 1.0（分布式计算框架）

JobTracker

- Master
- 管理所有作业
- 将作业分解成一系列任务
- 将任务指派给TaskTracker
- 作业/任务监控、错误处理等

TaskTrackers

- Slave
- 运行Map Task和Reduce Task
- 与JobTracker交互，执行命令，并汇报任务状态

MapReduce 1.0（分布式计算框架）

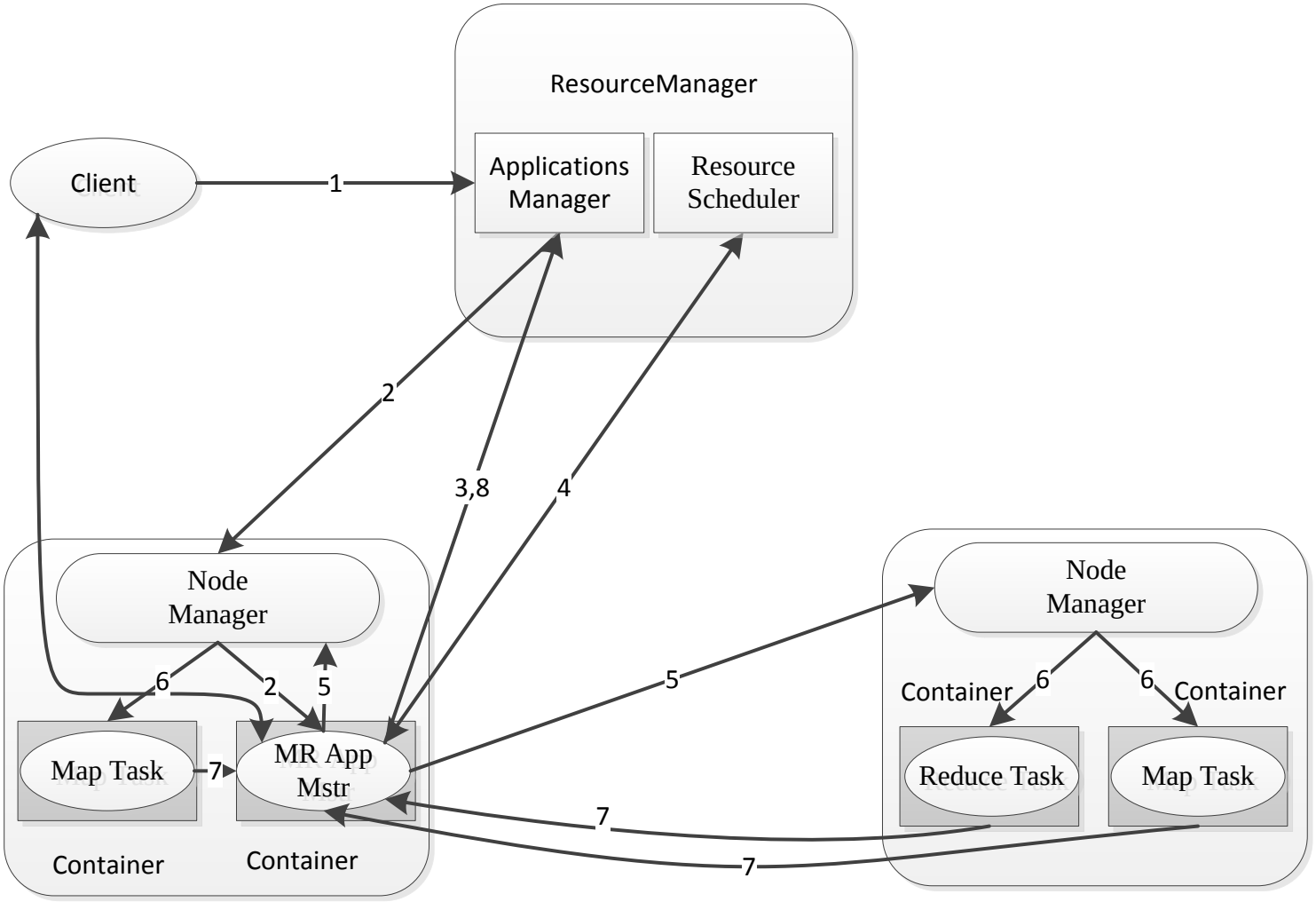
Map Task

- Map引擎
- 解析每条数据记录，传递给用户编写的map()
- 将map()输出数据写入本地磁盘（如果是map-only作业，则直接写入HDFS）

Reduce Task

- Reduce引擎
- 从Map Task上远程读取输入数据
- 对数据排序
- 将数据按照分组传递给用户编写的reduce()

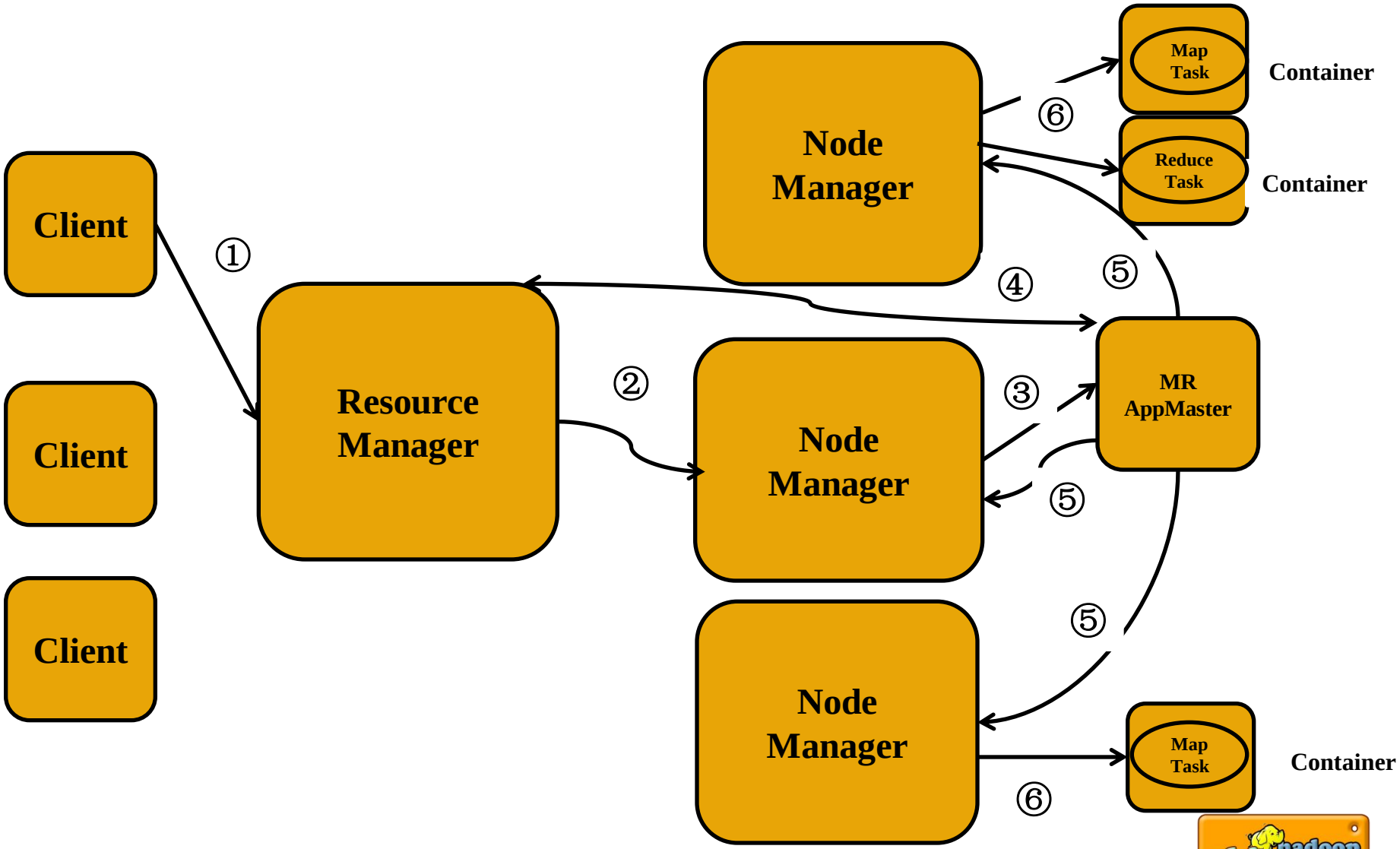
MapReduce 2.0架构



MapReduce 2.0架构

- **Client**
  - ✓ 与MapReduce 1.0的Client类似，用户通过Client与YARN交互，提交MapReduce作业，查询作业运行状态，管理作业等。
- **MRAppMaster**
  - ✓ 功能类似于 1.0中的JobTracker，但不负责资源管理；
  - ✓ 功能包括：任务划分、资源申请并将之二次分配个Map Task和Reduce Task、任务状态监控和容错。

MapReduce 2.0运行流程



MapReduce 2.0容错性

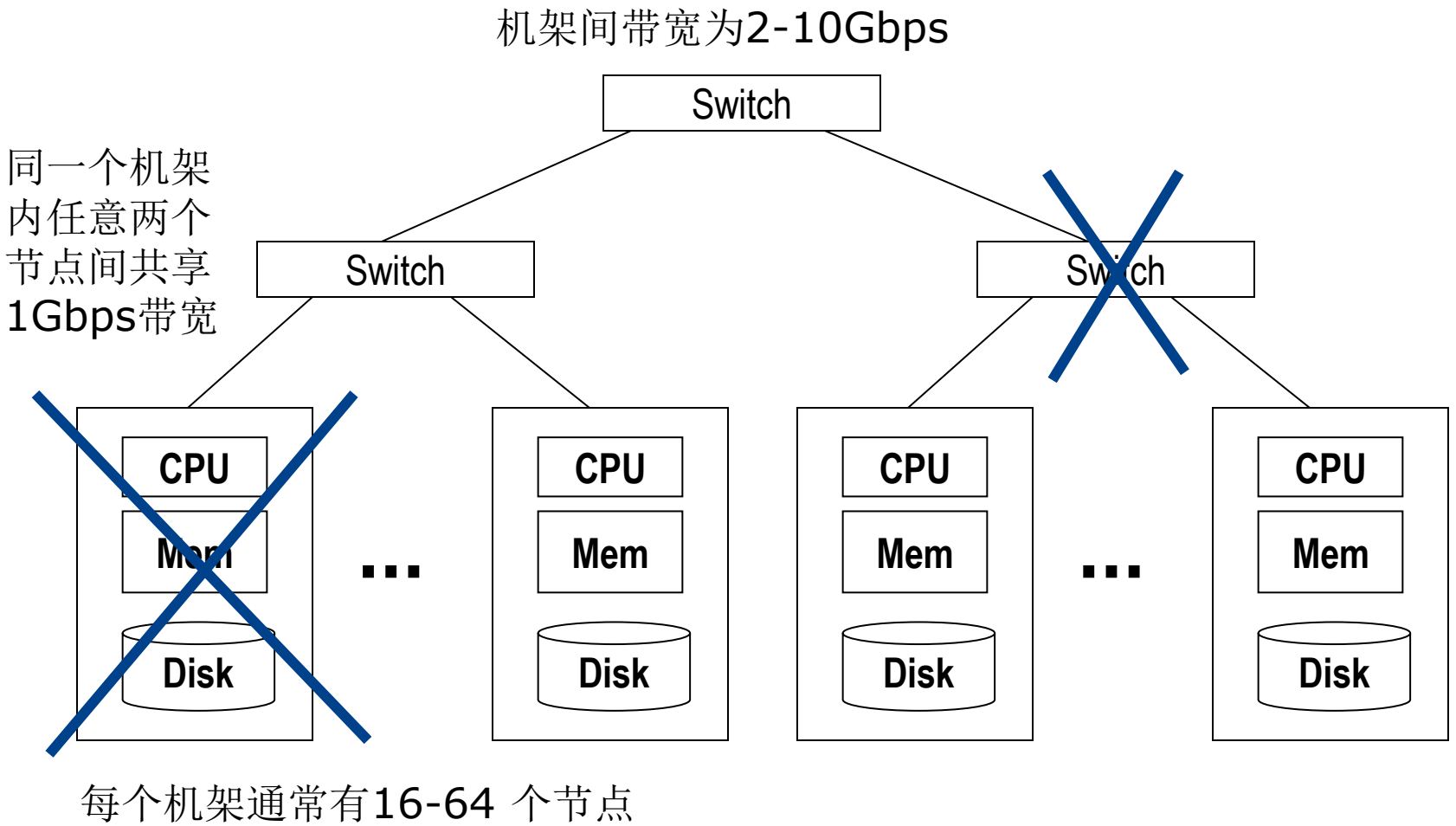
➢ MRAppMaster容错性

- ✓ 一旦运行失败，由YARN的ResourceManager负责重新启动，最多重启次数可由用户设置，默认是2次。一旦超过最高重启次数，则作业运行失败。

➢ Map Task/Reduce Task

- ✓ Task周期性向MRAppMaster汇报心跳；
- ✓ 一旦Task挂掉，则MRAppMaster将为之重新申请资源，并运行之。最多重新运行次数可由用户设置，默认4次。

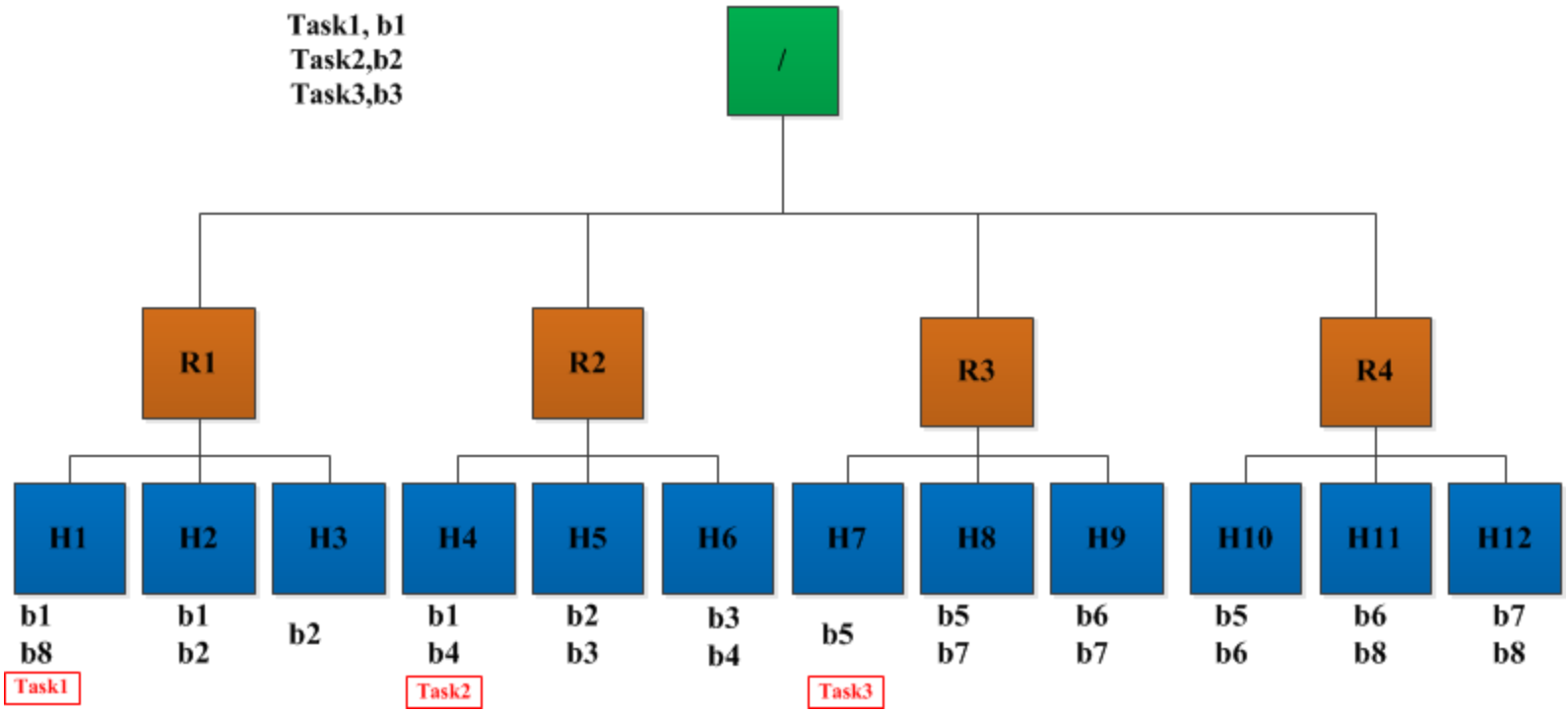
MapReduce计算框架—数据本地性



MapReduce计算框架—数据本地性

- 什么是数据本地性（data locality）
  - ✓如果任务运行在它将处理的数据所在的节点，则称该任务具有“数据本地性”
  - ✓本地性可避免跨节点或机架数据传输，提高运行效率
- 数据本地性分类
  - ✓同节点(node-local)
  - ✓同机架(rack-local)
  - ✓其他（off-switch）

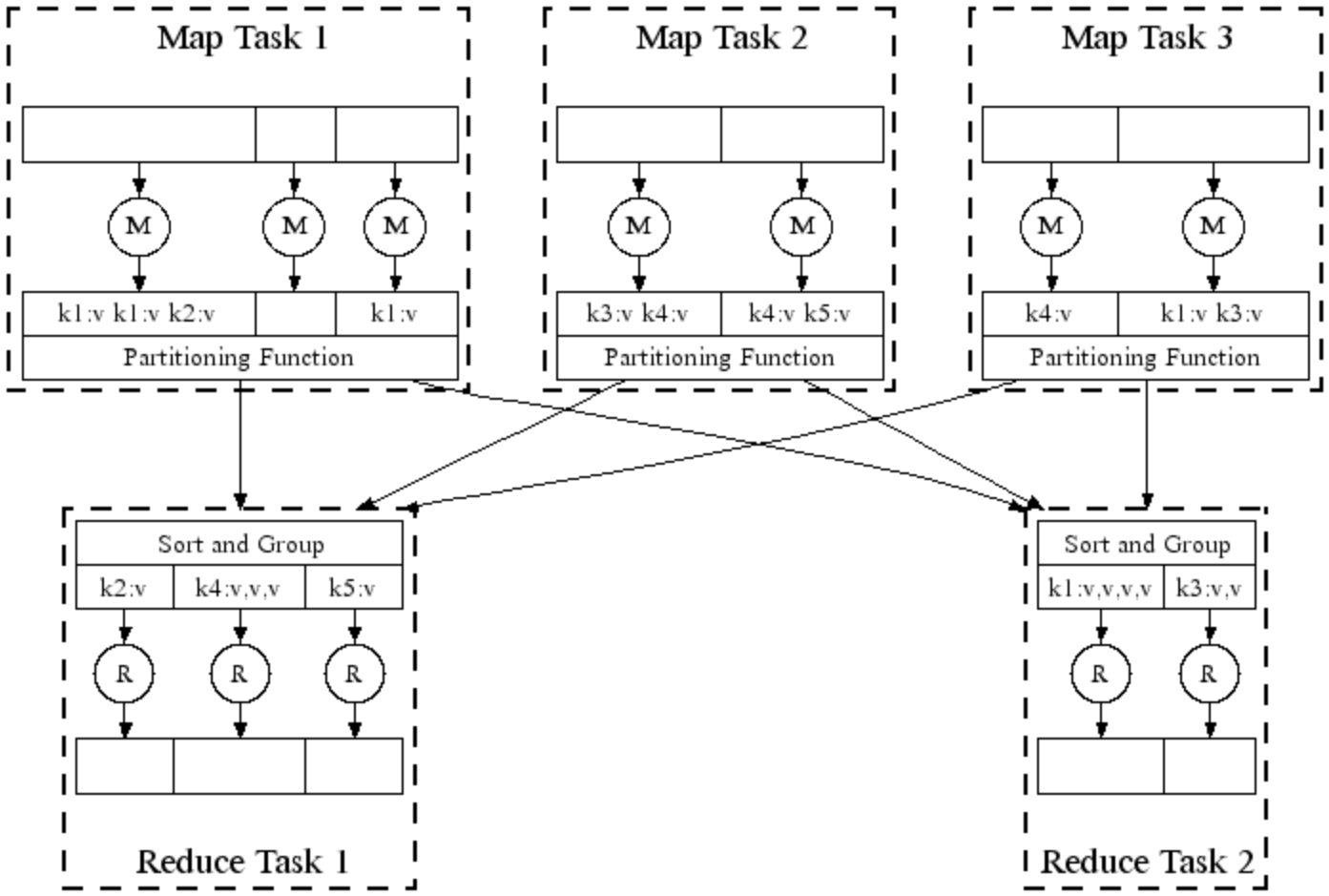
MapReduce计算框架—数据本地性



MapReduce计算框架—推测执行机制

- 作业完成时间取决于最慢的任务完成时间
  - ✓ 一个作业由若干个Map任务和Reduce任务构成
  - ✓ 因硬件老化、软件Bug等，某些任务可能运行非常慢
- 推测执行机制
  - ✓ 发现拖后腿的任务，比如某个任务运行速度远慢于任务平均速度
  - ✓ 为拖后腿任务启动一个备份任务，同时运行
  - ✓ 谁先运行完，则采用谁的结果
- 不能启用推测执行机制
  - ✓ 任务间存在严重的负载倾斜
  - ✓ 特殊任务，比如任务向数据库中写数据

MapReduce计算框架—任务并行执行



目录

常见MapReduce应用场景

- 简单的数据统计，比如网站pv、uv统计
- 搜索引擎建索引
- 海量数据查找
- 复杂数据分析算法实现
  - ✓聚类算法
  - ✓分类算法
  - ✓推荐算法
  - ✓图算法

总结

- MapReduce基本原理
- MapReduce编程模型
- MapReduce架构
- MapReduce任务调度器
- MapReduce应用场景