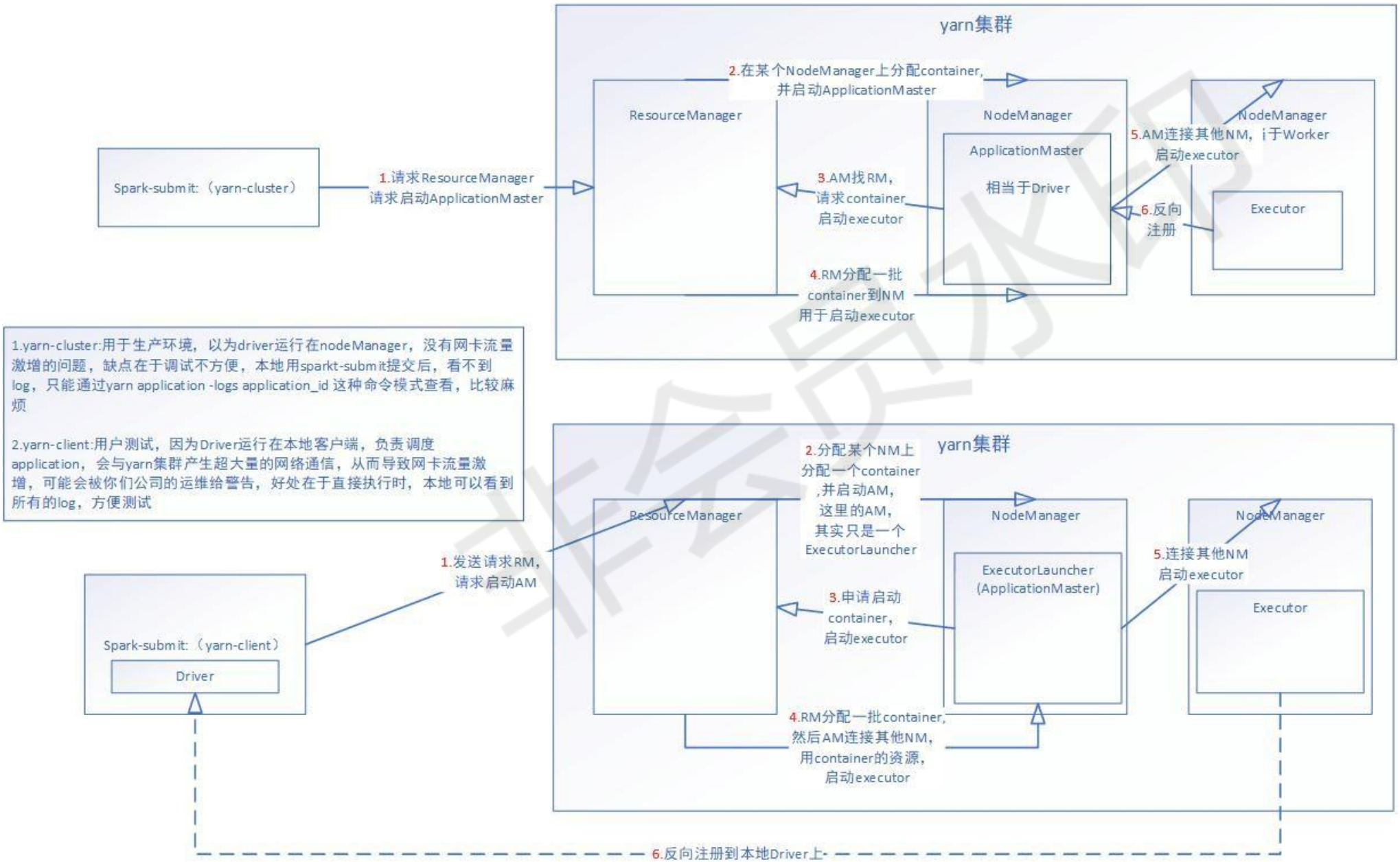
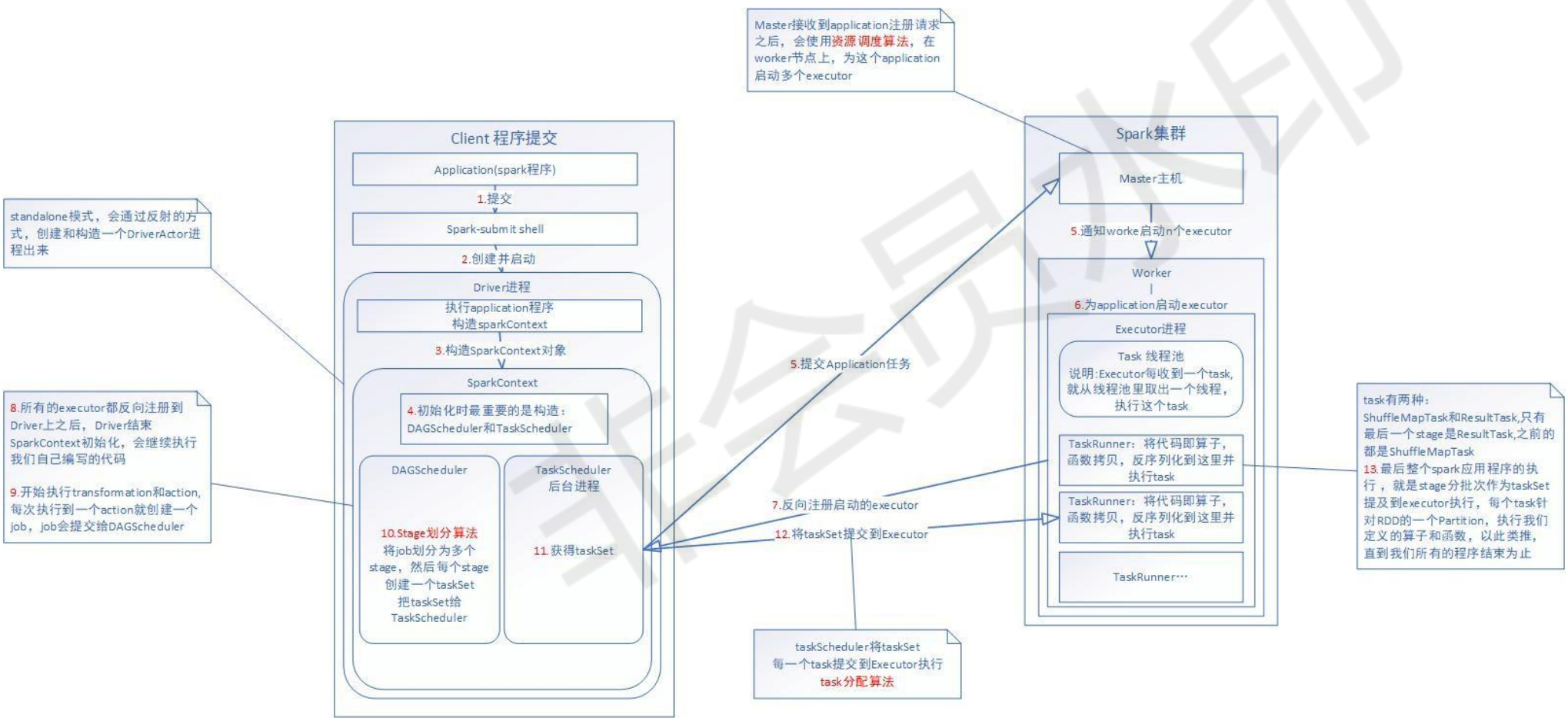
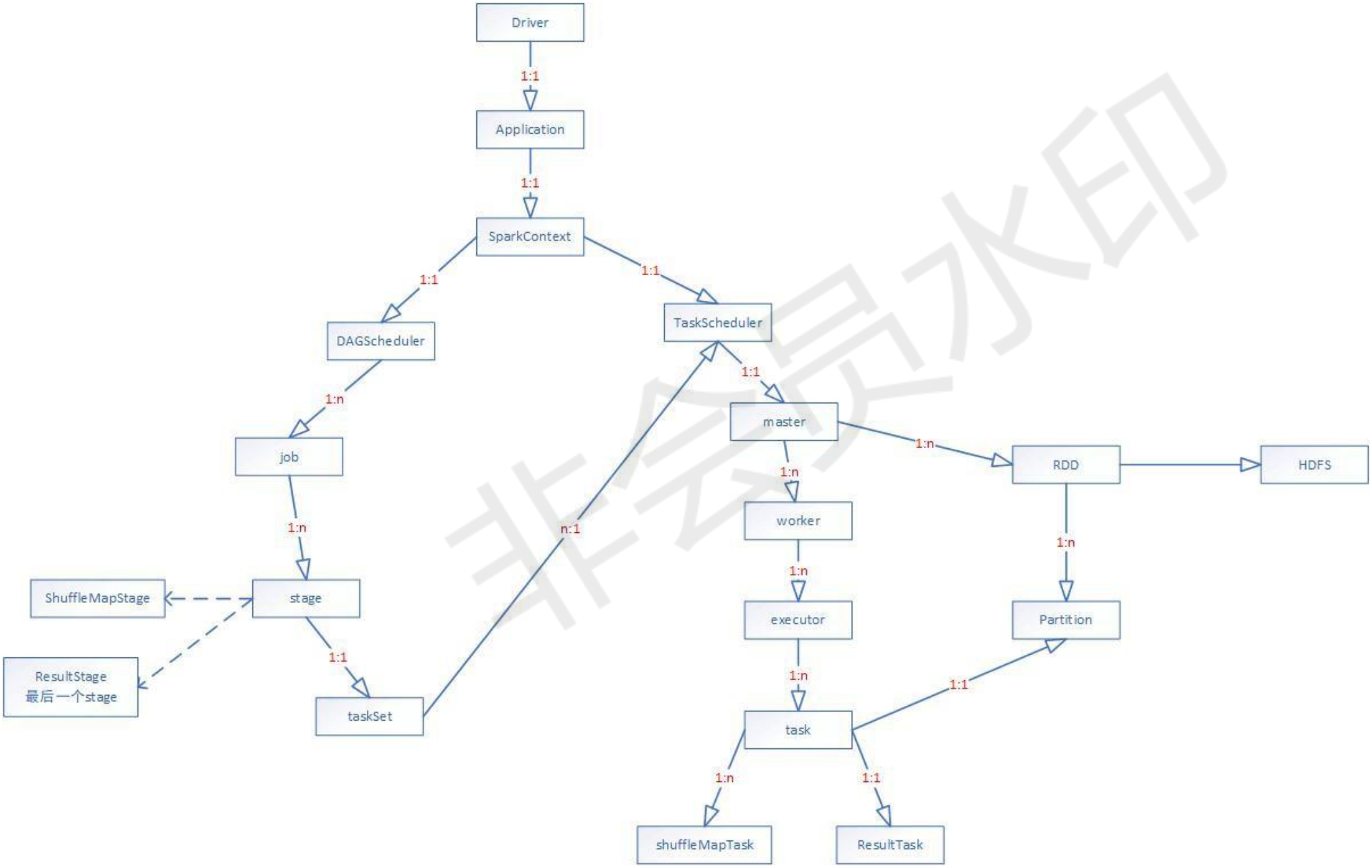
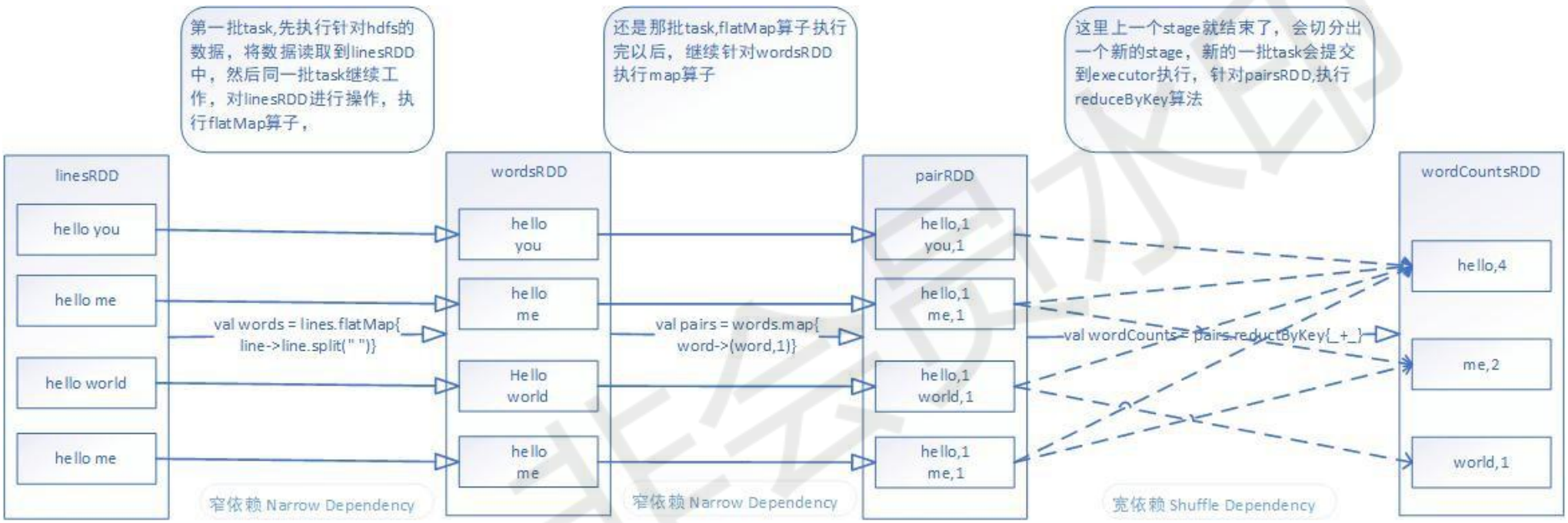

yarn集群
2.在某个NodeManager上分配container，并启动ApplicationMaster
ResourceManager
NodeManager
ApplicationMaster
相当于Driver
NodeManager
5.AM连接其他NM，i于Worker启动executor
Executor
Spark-submit:（yarn-cluster）
1.请求ResourceManager请求启动ApplicationMaster
3.AM找RM，请求container启动executor
6.反向注册
4.RM分配一批container到NM用于启动executor
1.yarn-cluster:用于生产环境，以为driver运行在nodeManager，没有网卡流量激增的问题，缺点在于调试不方便，本地用sparkt-submit提交后，看不到log，只能通过yarn application -logs application_id 这种命令模式查看，比较麻烦
2.yarn-client:用户测试，因为Driver运行在本地客户端，负责调度application，会与yarn集群产生超大量的网络通信，从而导致网卡流量激增，可能会被你们公司的运维给警告，好处在于直接执行时，本地可以看到所有的log，方便测试
yarn集群
2.分配某个NM上分配一个container,并启动AM,这里的AM,其实只是一个ExecutorLauncher
ResourceManager
NodeManager
NodeManager
ExecutorLauncher (ApplicationMaster)
5.连接其他NM启动executor
Executor
1.发送请求RM,请求启动AM
Spark-submit:（yarn-client）
Driver
3.申请启动container,启动executor
4.RM分配一批container,然后AM连接其他NM,用container的资源,启动executor
6.反向注册到本地Driver上

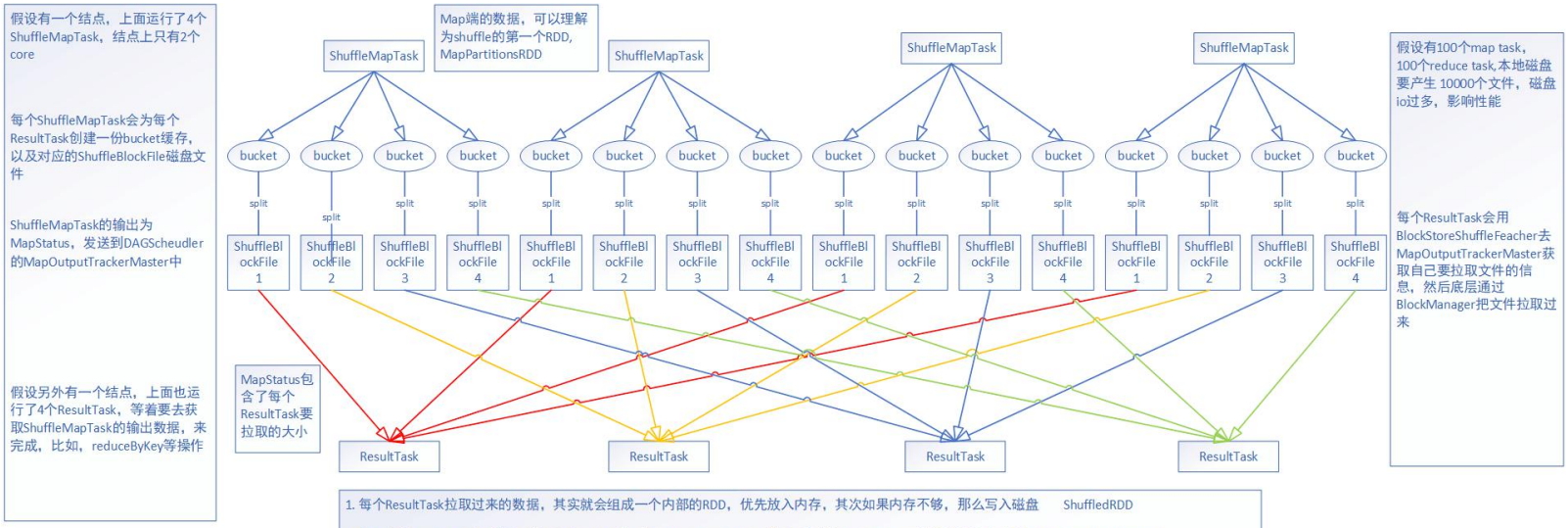

Client 程序提交
Application(spark程序)
1.提交
Spark-submit shell
2.创建并启动
Driver进程
执行application程序
构造sparkContext
3.构造SparkContext对象
SparkContext
4.初始化时最重要的是构造：
DAGScheduler和TaskScheduler
DAGScheduler
10.Stage划分算法
将job划分为多个stage，然后每个stage创建一个taskSet
把taskSet给TaskScheduler
TaskScheduler
后台进程
11.获得taskSet
standalone模式，会通过反射的方式，创建和构造一个DriverActor进程出来
8.所有的executor都反向注册到Driver上之后，Driver结束SparkContext初始化，会继续执行我们自己编写的代码
9.开始执行transformation和action，每次执行到一个action就创建一个job，job会提交给DAGScheduler
Master接收到application注册请求之后，会使用资源调度算法，在worker节点上，为这个application启动多个executor
Spark集群
Master主机
5.通知worke启动n个executor
Worker
6.为application启动executor
Executor进程
Task 线程池
说明:Executor每收到一个task,就从线程池里取出一个线程，执行这个task
TaskRunner：将代码即算子，函数拷贝，反序列化到这里并执行task
TaskRunner：将代码即算子，函数拷贝，反序列化到这里并执行task
TaskRunner…
5.提交Application任务
7.反向注册启动的executor
12.将taskSet提交到Executor
taskScheduler将taskSet
每一个task提交到Executor执行
task分配算法
task有两种：
ShuffleMapTask和ResultTask,只有最后一个stage是ResultTask,之前的都是ShuffleMapTask
13.最后整个spark应用程序的执行，就是stage分批次作为taskSet提及到executor执行，每个task针对RDD的一个Partition，执行我们定义的算子和函数，以此类推，直到我们所有的程序结束为止


Driver
1:1
Application
1:1
SparkContext
1:1
1:1
DAGScheduler
TaskScheduler
1:n
1:1
job
master
1:n
1:n
RDD
HDFS
1:n
worker
1:n
1:n
ShuffleMapStage
stage
n:1
executor
Partition
1:1
1:n
ResultStage
最后一个stage
taskSet
task
1:1
1:n
1:1
shuffleMapTask
ResultTask



窄依赖：Narrow Dependency, 一个RDD，对它的父RDD，只有简单的一对一的依赖关系，也就是说，RDD的每个partition，仅仅依赖于父RDD的一个Partition，父RDD和子RDD的partition之间的对应关系，是一对一的

宽依赖：Shuffle Dependency，就是shuffle，每一个RDD的patitioin中的数据，都可能会传输一部分给下一个RDD的每个partition，此时就会出现，父RDD和子RDD的partition之间，具有交互操作复杂的关系，这种情况下，就叫做两个RDD之间的宽依赖，同事，他们之间发生的操作，就是shuffle。



1. 每个ResultTask拉取过来的数据，其实就会组成一个内部的RDD，优先放入内存，其次如果内存不够，那么写入磁盘　　ShuffledRDD

2. 然后每个ResultTask针对数据进行聚合，最后生成MapPartitionsRDD，就是我们执行reduceByKey等操作希望获得的RDD　MapPartitionsRDD

Spark 的 Shuffle 特点说明：
1. 早期版本中，bucket缓存是非常重要，因为需要将一个ShuffleMapTask所有的数据都写入内存缓存之后，才会刷新到磁盘，但有一个问题，如果map side数据过多，那么很容易造成内存溢出，所以在spark新版本中优化了，默认那个内存缓存是100Kb，然后写入一点数据达到了，刷新到磁盘的阈值后，就会想数据一点一点的刷新到磁盘。
　　优点：不容易发出内存溢出
　　缺点：如果内存缓存过小的话，那么可能发生过多测磁盘写IO操作，所以，这里的内存缓存要根据实际情况进行优化
2. 与MapReduce完全不一样的是，MapReduce必须将所有的数据都写入磁盘文件之后，才能启动reduce操作来拉取数据，为什么？因为MapReduce要实现默认的根据key的排序，所以要排序肯定先要写完所有数据，才能排序，然后reduce来拉取。
　　但是spark不同，spark默认情况下，是不会对数据进行排序的，因此ShuffleMapTask每写入一点数据，ResultTask就可以拉取一点数据，然后在本地执行我们定义的聚合函数和算子，进行计算。
　　spark这种机制的好处在于，速度比MapReduce快多了，但也有一个问题，MapReduce提供的reduce，是可以处理每个key对应的value的，很方便，但是spark中，由于这种拉取的机制，因此提供不了，直接处理key对应的values的算子，只能通过groupbykey，先shuffle，有一个MapPartitiionsRDD，然后用map算子，来处理每个key对应的values，就没有mapReduce那么方便。