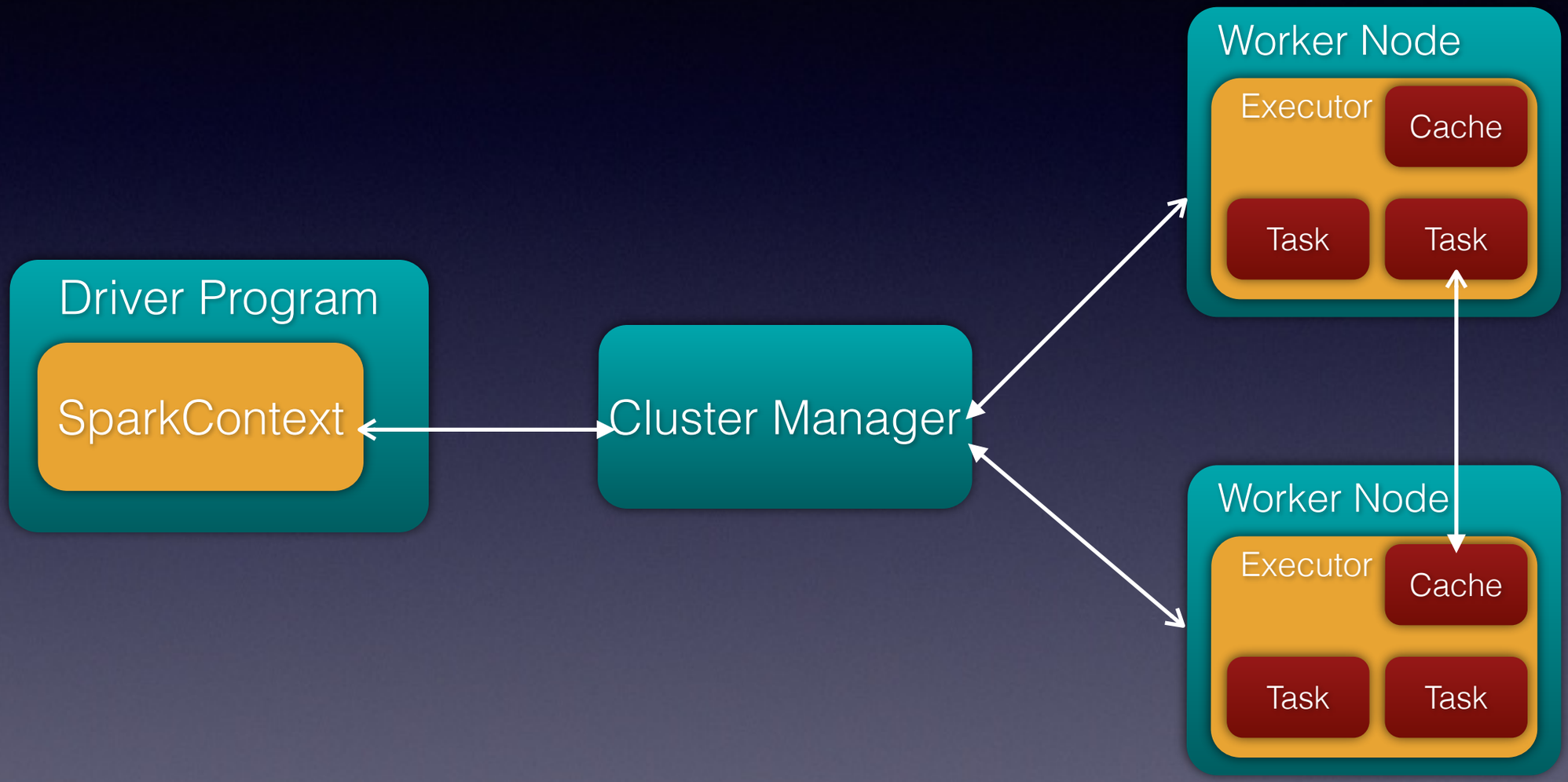
深入Spark内核

陈 超
@CrazyJvm

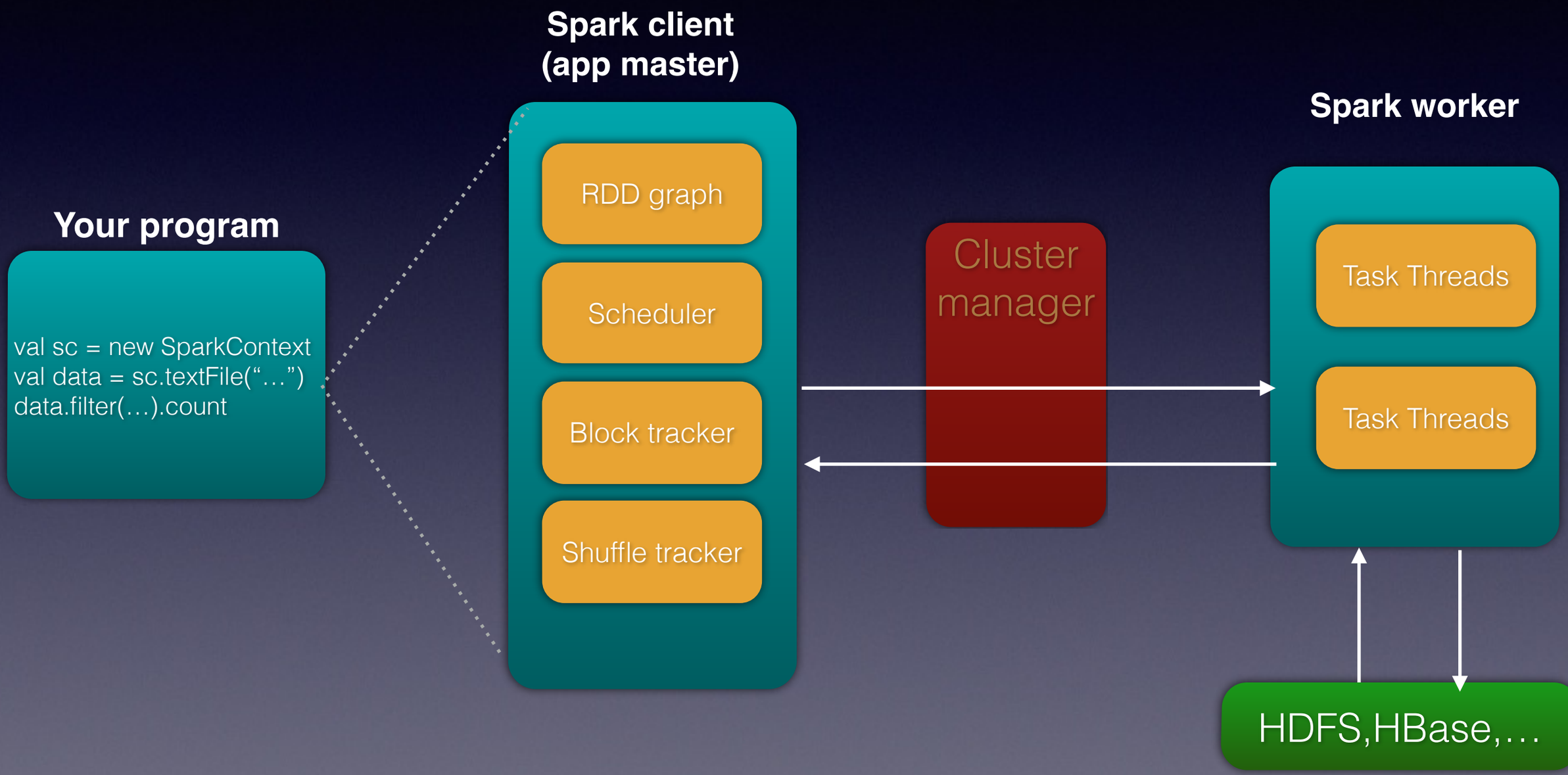
术语解释

术语	解释
Application	基于Spark的用户程序，包含了driver程序和集群上的executor
Driver Program	运行main函数并且新建SparkContext的程序
Cluster Manager	在集群上获取资源的外部服务(例如:standalone,Mesos,Yarn)
Worker Node	集群中任何可以运行应用代码的节点
Executor	是在一个worker node上为某应用启动的一个进程，该进程负责运行任务，并且负责将数据存在内存或者磁盘上。每个应用都有各自独立的executors
Task	被送到某个executor上的工作单元
Job	包含很多任务的并行计算，可以看做和Spark的action对应
Stage	一个Job会被拆分很多组任务，每组任务被称为Stage(就像Mapreduce分map任务和reduce任务一样)

Cluster Overview



核心组件



RDD图

每个partition都会分配一个task

数据层面

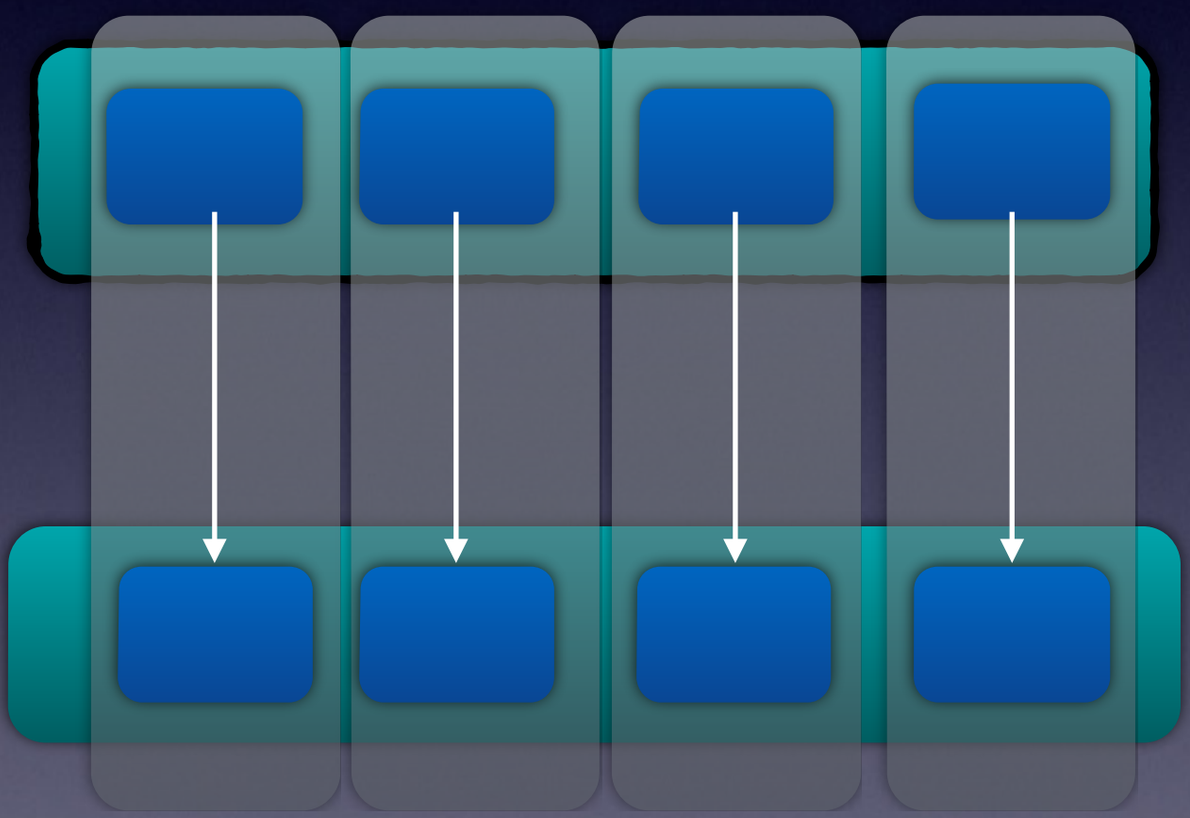
Partition层面

file:

HadoopRDD
path=hdfs://…

errors:

FilteredRDD
func=_.contains(…)
rdd.cache



Task1 Task2 Task3 ……

数据本地性如何?

第一次运行时数据不在内存中，所以从HDFS上取，任务最好运行在数据所在的节点上!

文件系统本地性

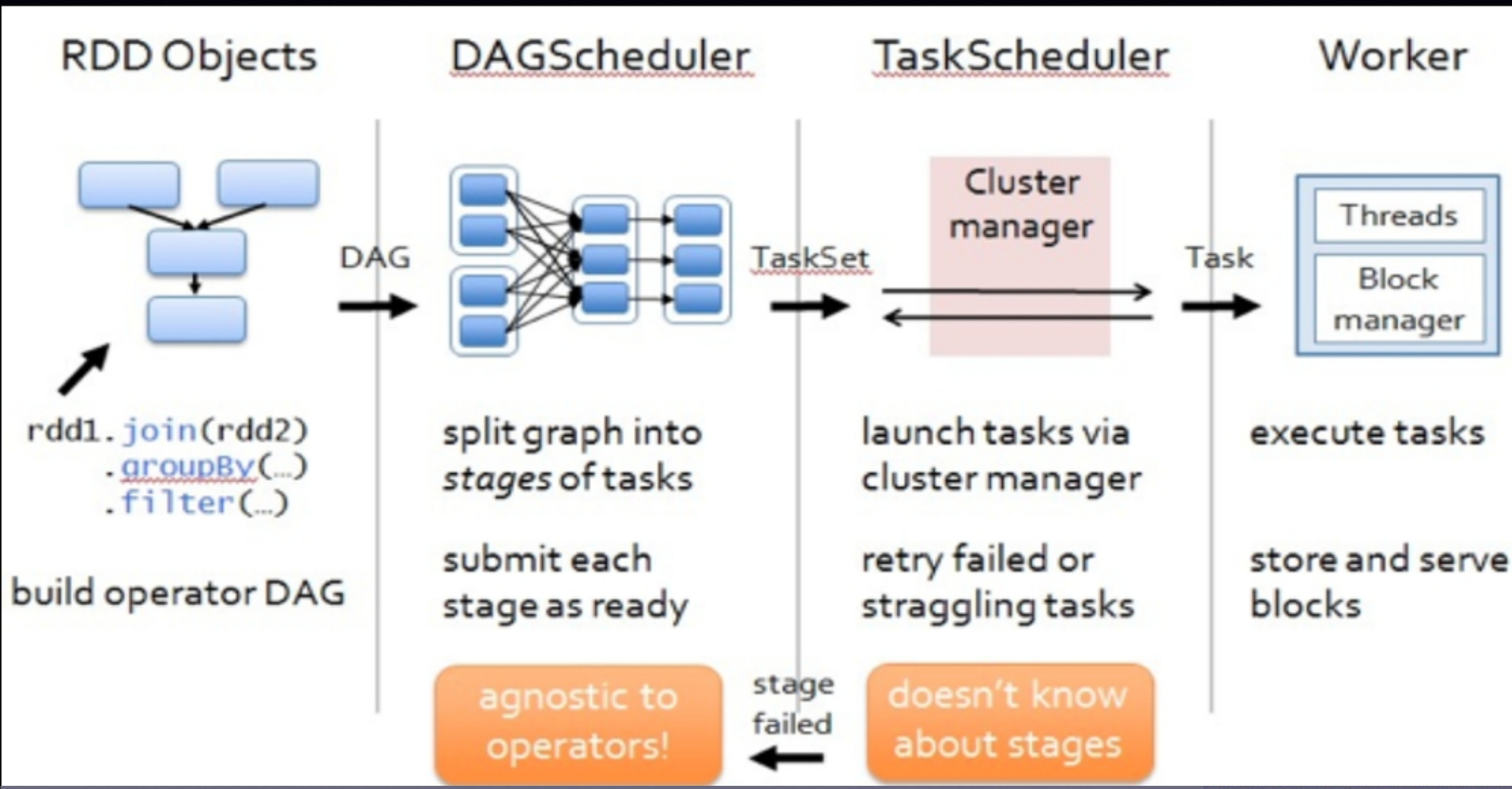
第二次运行，数据已经在内存中，所以任务最好运行在该数据所在内存的节点上。

内存本地性

万一有数据被置换出内存，则仍然从HDFS上取。

LRU置换

任务调度!



再看RDD

- 分区 **protected def getPartitions: Array[Partition]**
- 依赖 **protected def getDependencies: Seq[Dependency[_]] = deps**
- 函数 **def compute(split: Partition, context: TaskContext): Iterator[T]**
- 最佳位置(可选) **protected def getPreferredLocations(split: Partition): Seq[String] = Nil**
- 分区策略(可选) **@transient val partitioner: Option[Partitioner] = None**

最常见的HadoopRDD

- 分区: 每个HDFS block
- 依赖: 无
- 函数: 读取每一个block
- 最佳位置: HDFS block所在位置
- 分区策略: 无

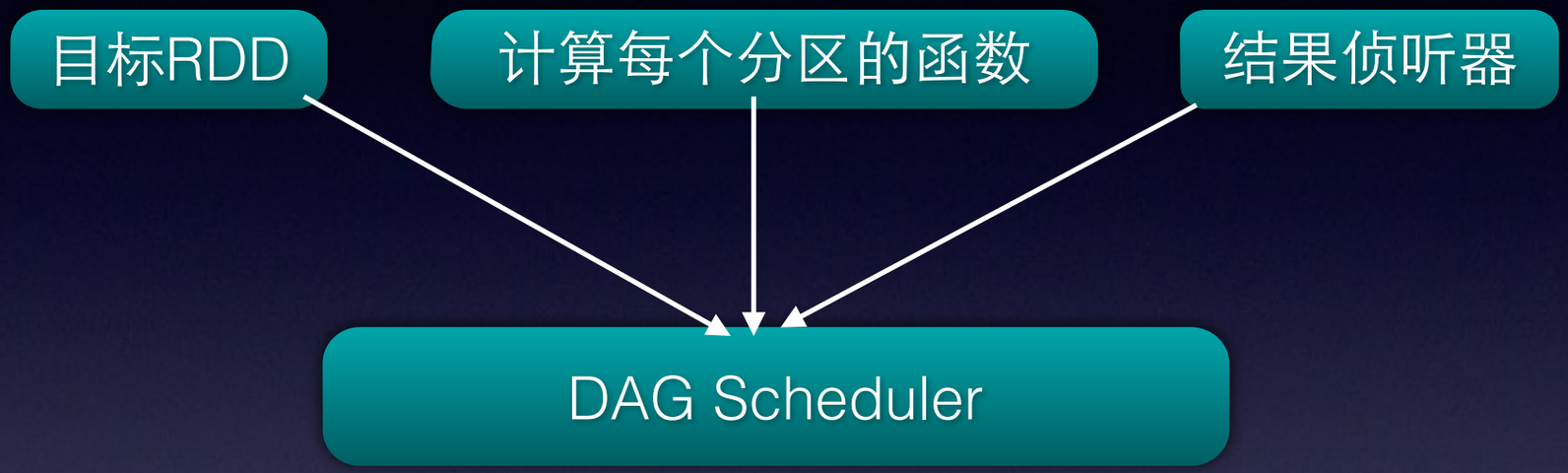
FilteredRDD

- 分区: 与父RDD一致
- 依赖: 与父RDD一对一
- 函数: 计算父RDD的每个分区并过滤
- 最佳位置: 无(与父RDD一致)
- 分区策略: 无

JoinedRDD

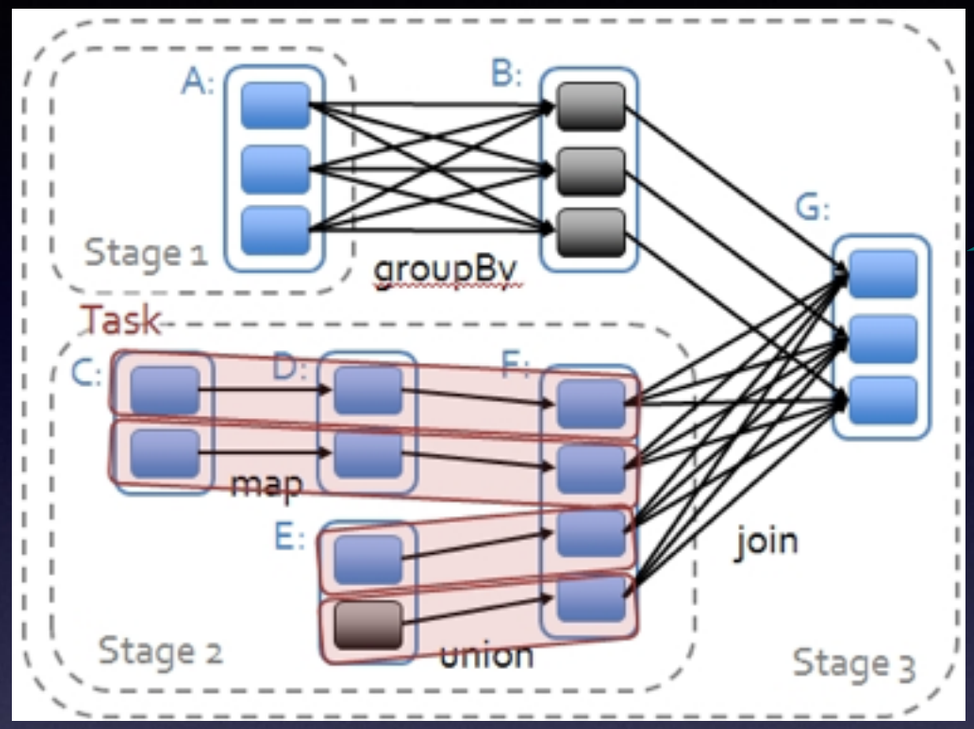
- 分区：每个reduce任务一个分区
- 依赖：依赖所有父RDD
- 函数 :读取suffle数据并计算
- 最佳位置：无
- 分区策略：HashPartitioner(partitions: Int)

细看DAG Scheduler



- 基于Stage构建DAG，决定每个任务的最佳位置
- 记录哪个RDD或者Stage输出被物化
- 将taskset传给底层调度器TaskScheduler
- 重新提交shuffle输出丢失的stage

调度器优化



灰颜色代表之前已经算好的分区

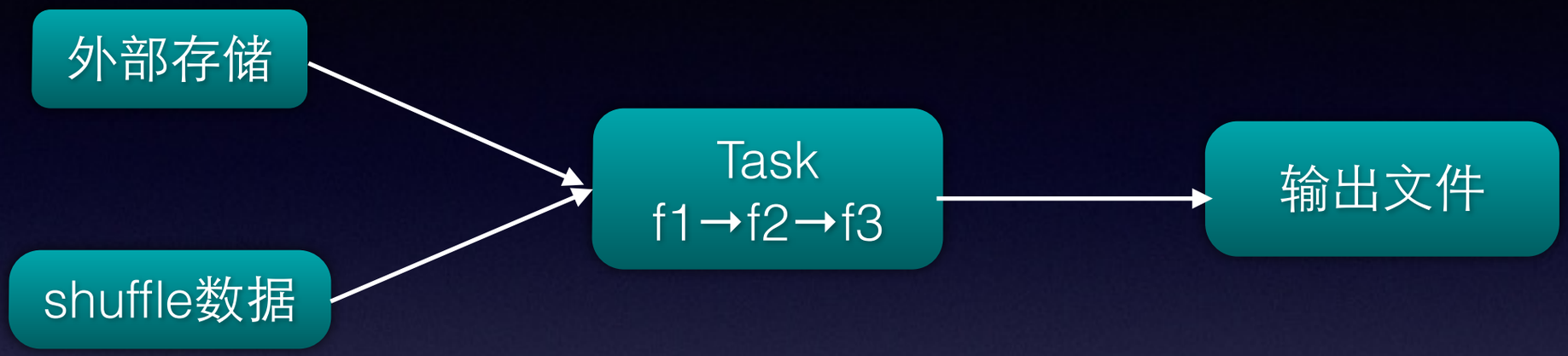
- 一个Stage内的窄依赖进行pipeline操作

  1+1+1+1 = 4 ①　　1+1=2；2+1=3；3+1=4 ②

- 基于partition选择最优的join算法使shuffle的数据最小化

- 重用已经缓存过的数据

Task细节

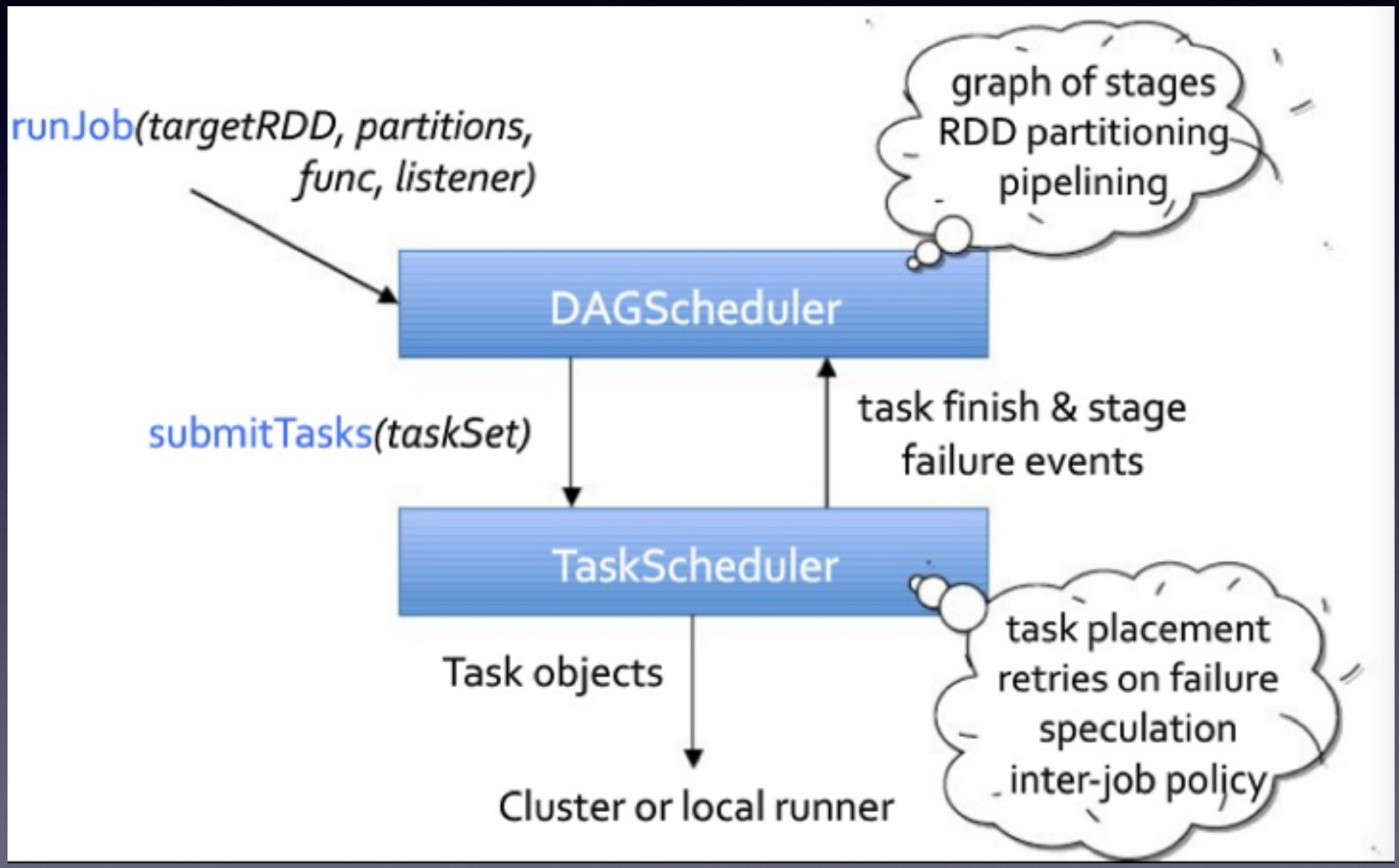


- Stage边界只出现在外部输入及取shuffle数据的时候
- 为了容错，会把suffle输出写在磁盘或者内存
- 任何一个任务可以运行在任何一个节点
- 允许任务使用那些被缓存但是已经被置换出去的数据

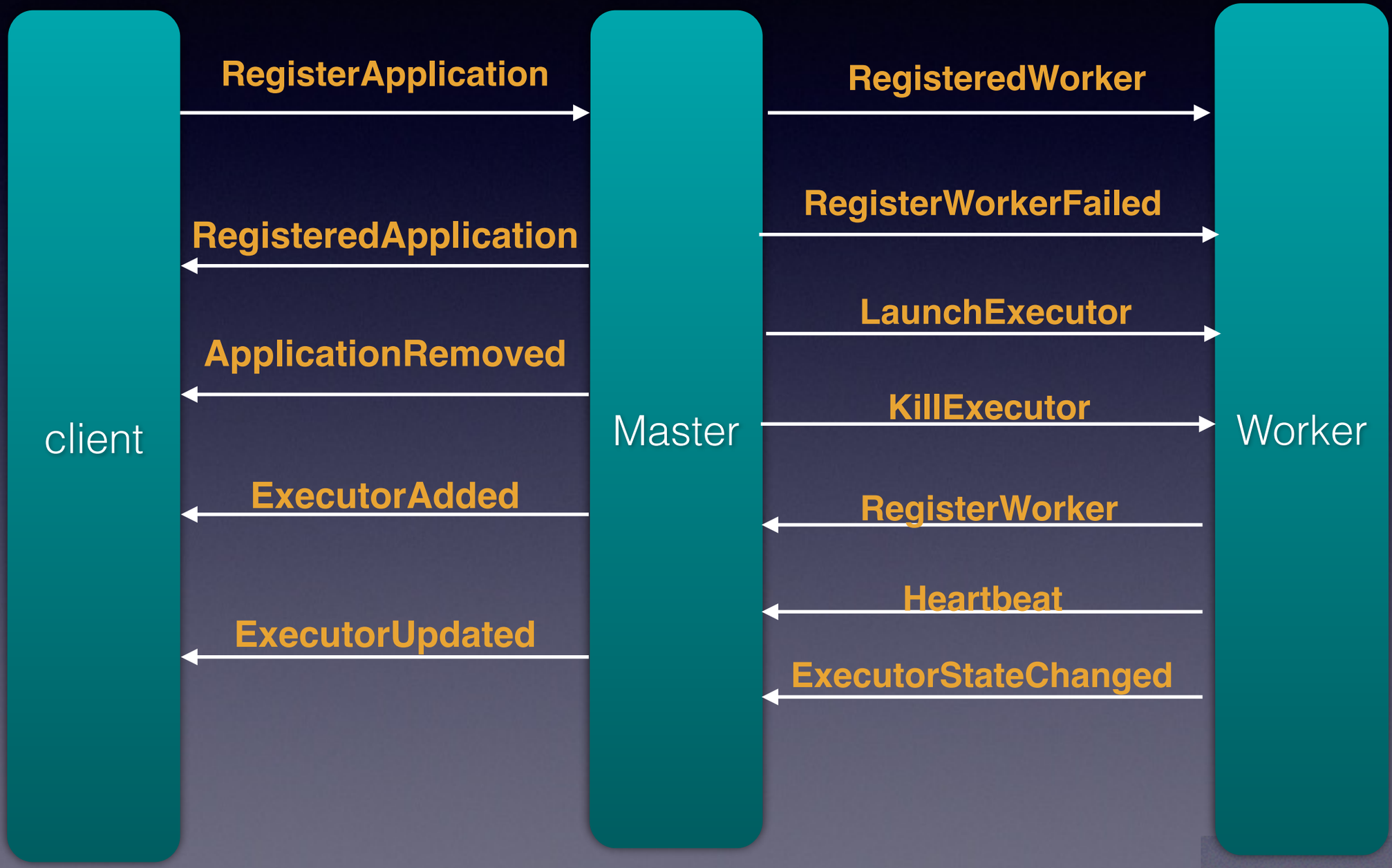
TaskScheduler

- 提交taskset(一组task)到集群运行并汇报结果
- 出现shuffle输出lost要报告fetch failed错误
- 碰到straggle任务需要放到别的节点上重试
- 为每一个TaskSet维护一个TaskSetManager(追踪本地性及错误信息)

Job调度流程



Master & Worker



略去多**Master**选举等相关信息

广播变量 Broadcast variables

- BT形式的广播变量
- 使用场景 : lookup表， mapside join
- 注意点 : 只读，存于每台worker的cache，不随task发送
- 使用方式 : val broadcastVar = sc.broadcast(Array(1, 2, 3))

  broadcastVar.value

累加器 Accumulators

- 只增
- 类似与MapReduce中的counter
- 用法 : val accum = sc.accumulator(0)

```
sc.parallelize(Array(1, 2, 3, 4)).foreach(x => accum += x)

accum.value
```

性能调优

调优是建立在充分理解的基础上！

优化点1

思考？为什么会大?

- 问题: Task序列化后太大
- 解决: 使用广播变量

Spark0.9开始序列化大小超过100K就会警告信息啦!

优化点2

- 问题: val rdd = data.filter(f1).filter(f2).reduceBy…

  经过以上语句会有很多空任务或者小任务

- 解决: 使用coalesce或者repartition去减少RDD中partition数量

coalesce与repartition的关系是什么？有什么异同？请看源码!

优化点3

- 问题: 每个记录的开销太大

  rdd.map{x=>conn=getDBConn;conn.write(x.toString);conn.close}

- 解决: rdd.mapPartitions(records => conn.getDBConn;for(item <- records)) write(item.toString); conn.close)

map是在每个元素上应用此函数,mapPartition是在一个partition上应用此函数

优化点4

- 问题: 任务执行速度倾斜
- 解决: 1、数据倾斜(一般是partition key取的不好)

考虑其它的并行处理方式 中间可以加入一步aggregation

2、Worker倾斜(在某些worker上的executor不给力)

设置spark.speculation=true 把那些持续不给力的node去掉

对比Hadoop MapReduce的speculation

优化点5

- 问题： 不设置spark.local.dir 这是spark写shuffle输出的地方

- 解决: 设置一组磁盘

  spark.local.dir=/mn1/spark, /mnt2/spar, /mnt3/spark

增加IO即加快速度

优化点6

默认reduce数量?

- 问题: reducer数量不合适
- 解决: 需要按照实际情况调整

太多的reducer,造成很多的小任务,以此产生很多启动任务的开销。
太少的reducer,任务执行慢!

reduce的任务数还会影响到内存

优化点7

- 问题：collect输出大量结果慢
- 解决：直接输出到分布式文件系统

查看collect源码？会发现什么？

优化点8

优点:兼容性好
缺点：体积大，速度慢

- 问题:序列化 Spark默认使用JDK自带的ObjectOutputStream

优点:
体积小，速度快

- 解决: 使用Kryo serialization

Note:使用Kryo需要注册自己的类

谢 谢