Get Start from SparkPi

图解Spark core

@时金魁
花名玄畅
淘宝技术部-数据挖掘与计算-高性能计算

2015-3

Why

现在spark源码解读的博客和书已经很多了

图片来自500px

focus

- 全局观
- 一个Spark任务完整的执行流程

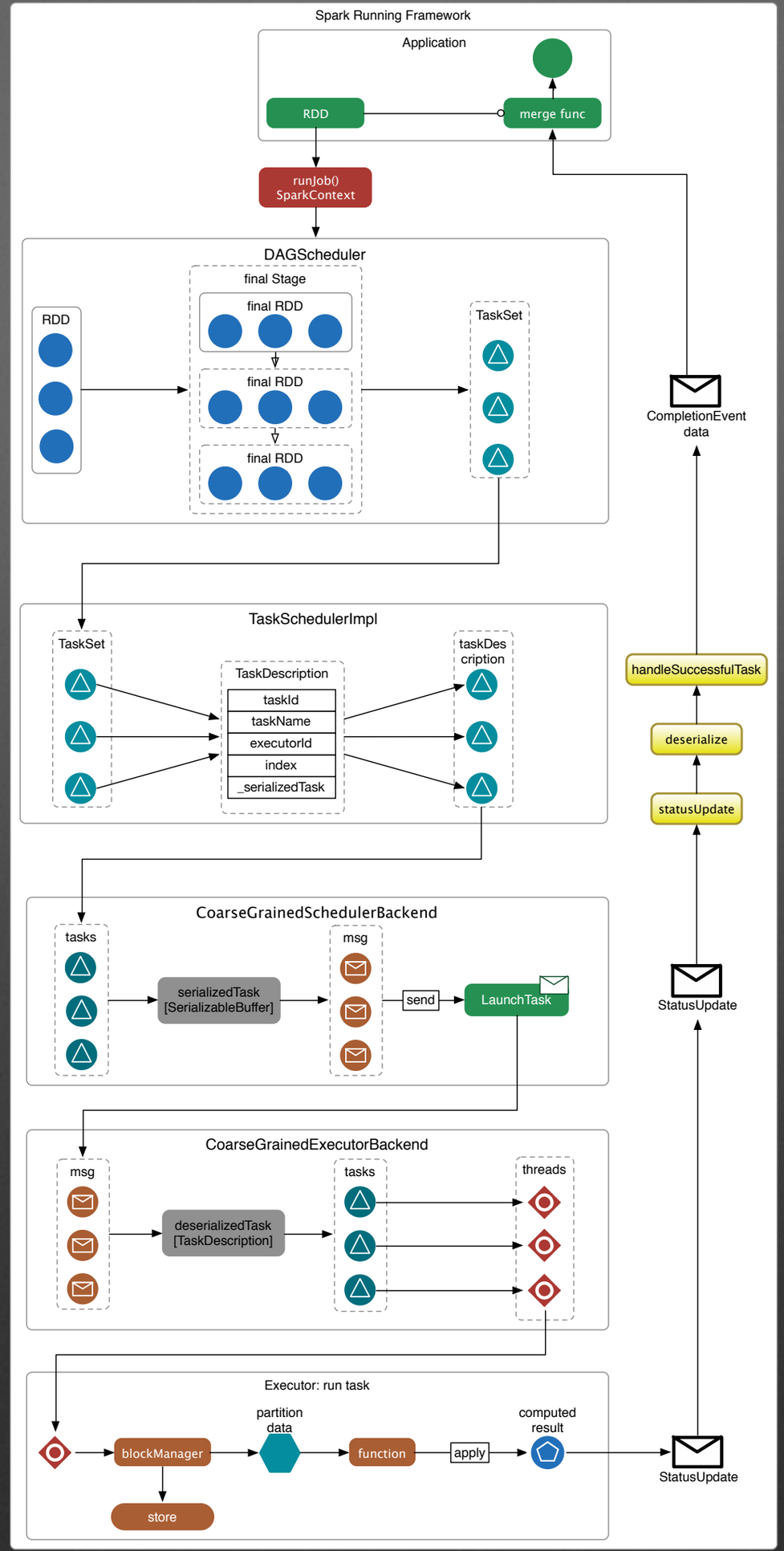
说明:

1. 以图说为主
2. 配合graffle图
3. shuffleTask略(这块太复杂)

get start from

```
def main(args: Array[String]) {
  val conf = new SparkConf().setAppName("Spark Pi")
  val spark = new SparkContext(conf)
  val slices = if (args.length > 0) args(0).toInt else 2
  val n = 100000 * slices
  val count = spark.parallelize(1 to n, slices).map { i =>
    val x = random * 2 - 1
    val y = random * 2 - 1
    if (x*x + y*y < 1) 1 else 0
  }.reduce(_ + _)
  println("Pi is roughly " + 4.0 * count / n)
  spark.stop()
}
```

调用顺序流程图


Spark Running Framework
Application
RDD
merge func
runJob()
SparkContext
DAGScheduler
final Stage
final RDD
final RDD
final RDD
RDD
TaskSet
CompletionEvent
data
TaskSchedulerImpl
TaskSet
TaskDescription
taskId
taskName
executorId
index
_serializedTask
taskDes
cription
handleSuccessfulTask
deserialize
statusUpdate
CoarseGrainedSchedulerBackend
tasks
serializedTask
[SerializableBuffer]
msg
send
LaunchTask
StatusUpdate
CoarseGrainedExecutorBackend
msg
deserializedTask
[TaskDescription]
tasks
threads
Executor: run task
blockManager
partition
data
function
apply
computed
result
store
StatusUpdate

数据流向图

Driver

```
def main(args: Array[String]) {
  val conf = new SparkConf().setAppName("Spark Pi")
  val spark = new SparkContext(conf)
  val slices = if (args.length > 0) args(0).toInt else 2
  val n = 100000 * slices
  val count = spark.parallelize(1 to n, slices).map { i =>
    val x = random * 2 - 1
    val y = random * 2 - 1
    if (x*x + y*y < 1) 1 else 0
  }.reduce(_ + _) // reduce fn
  println("Pi is roughly " + 4.0 * count / n)
  spark.stop()
}
```

MapPartitionsRDD

partition fn:
call by name fun

pars

ParallelCollectionRDD

partition data
1-100000
100001-200000
200001-300000

dependency

dependency

dependency

reduce()

reduce fn:
call by name fun

runJob()

DAGScheduler

TaskScheduler

CoarseGrainedSc
hedulerBackend

Task1

1-100000

Task2

100001-200000

Task3

200001-300000

Cluster: Worker1

Task1

1-100000

apply

partition fn:
call by name fun

Cluster: Worker2

Task1

1-100000

apply

partition fn:
call by name fun

Cluster: Worker3

Task1

1-100000

apply

partition fn:
call by name fun

count变量

reduce fn:
call by name fun

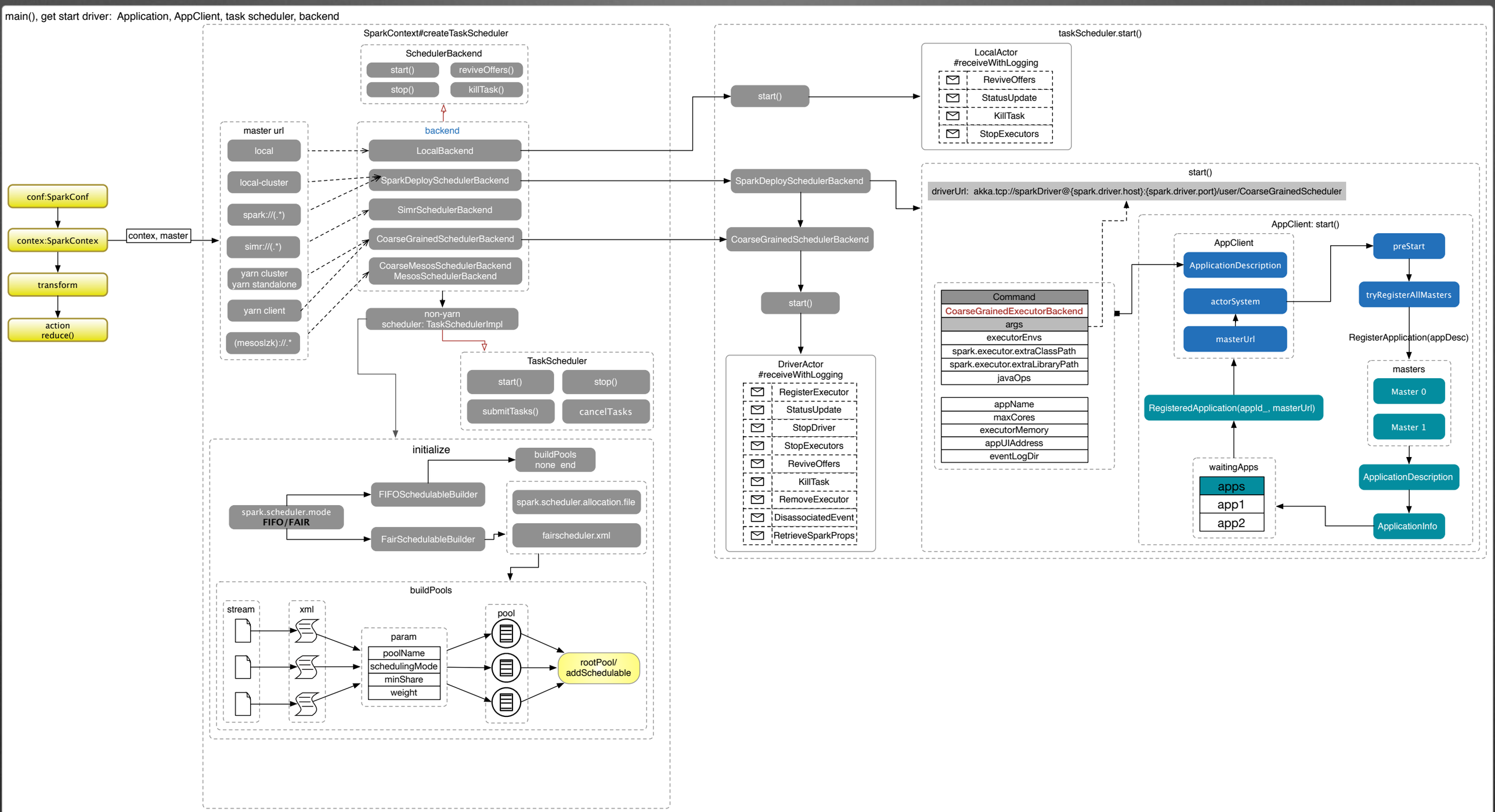
DAGScheduler

Driver
Local

Cluster

第1步： start driver

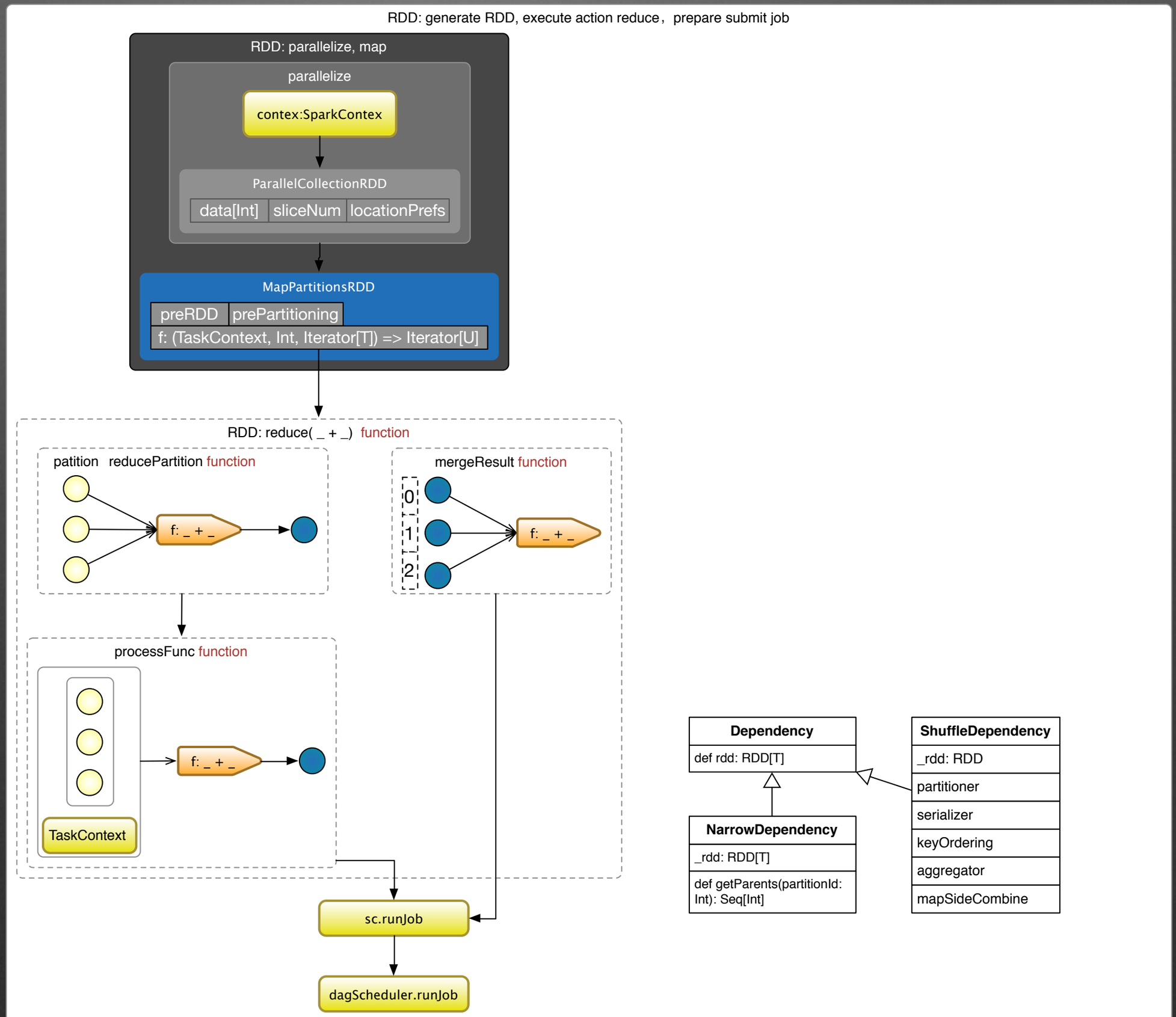
初始化: Application, AppClient, task scheduler, backend



第2步: Action->提交job任务

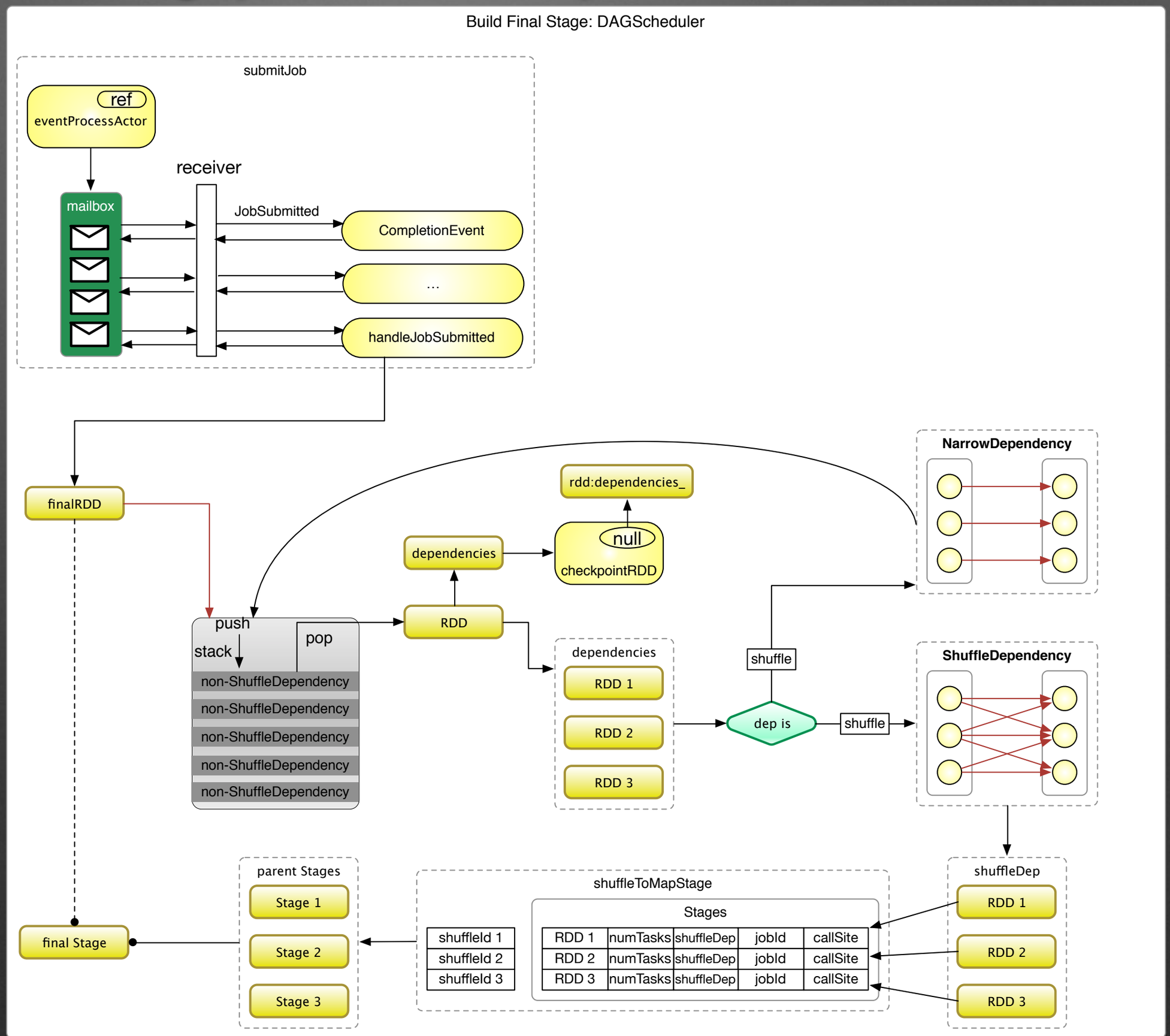
两个函数:

1. processFunc -> Partition
2. mergeResult -> Partition Result



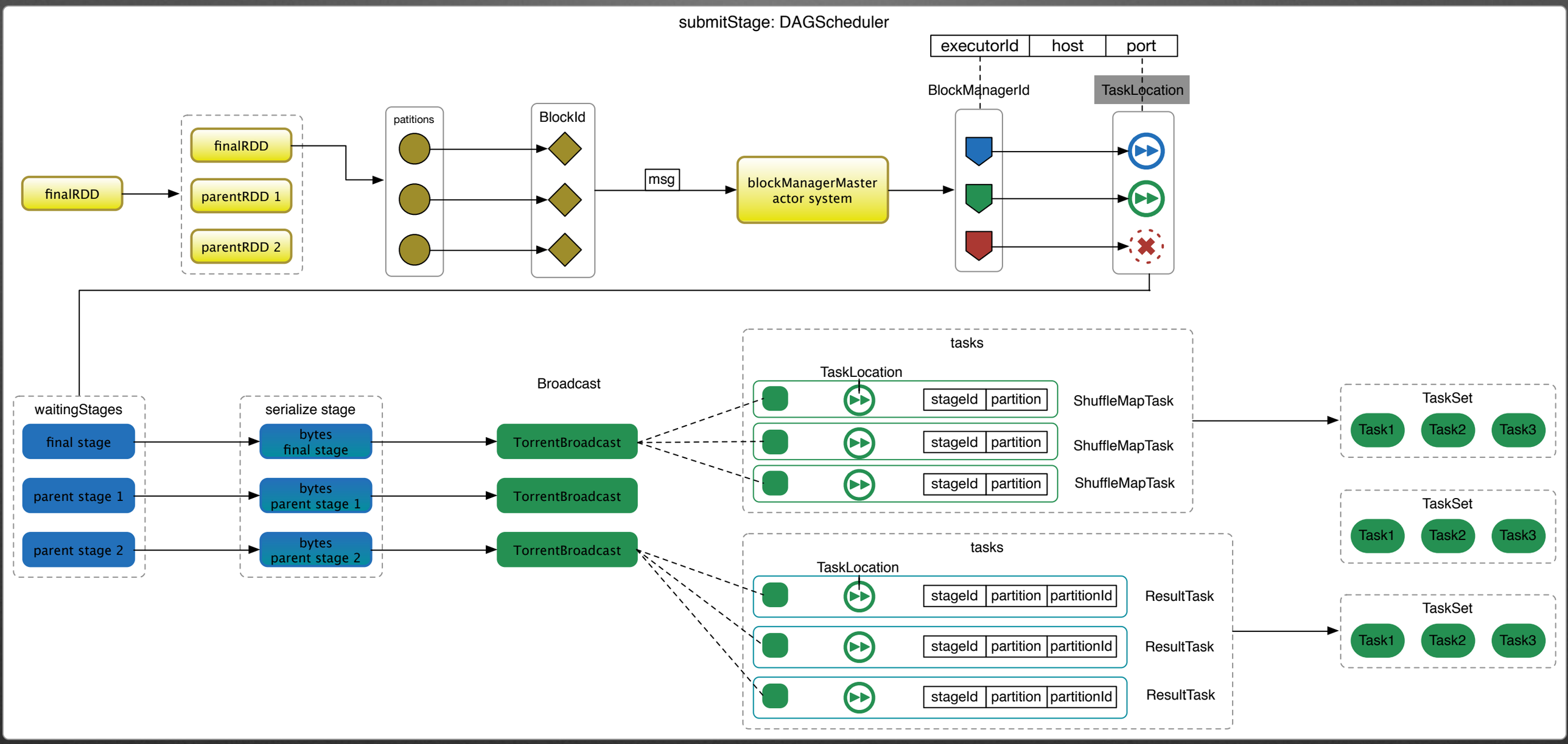
第3步: build final stage(DAGScheduler)

根据依赖关系, 生成
有序的父RDD列表



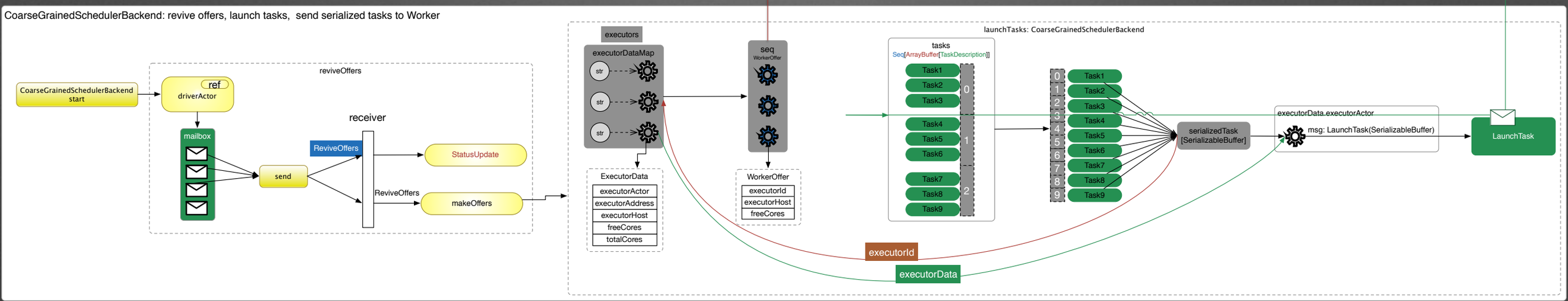
第4步: submitStage(DAGScheduler)

1. 以partition为最小单元, 封装成最终执行的Task对象
2. 转换过程: partition -> TaskLocation -> ShuffleMapTask/ResultTask
   最终转化成Task对象, Task对象即将被调度到分散的worker上，给分配线程执行



第5步: lauch tasks(CoarseGrainedSchedulerBackend)

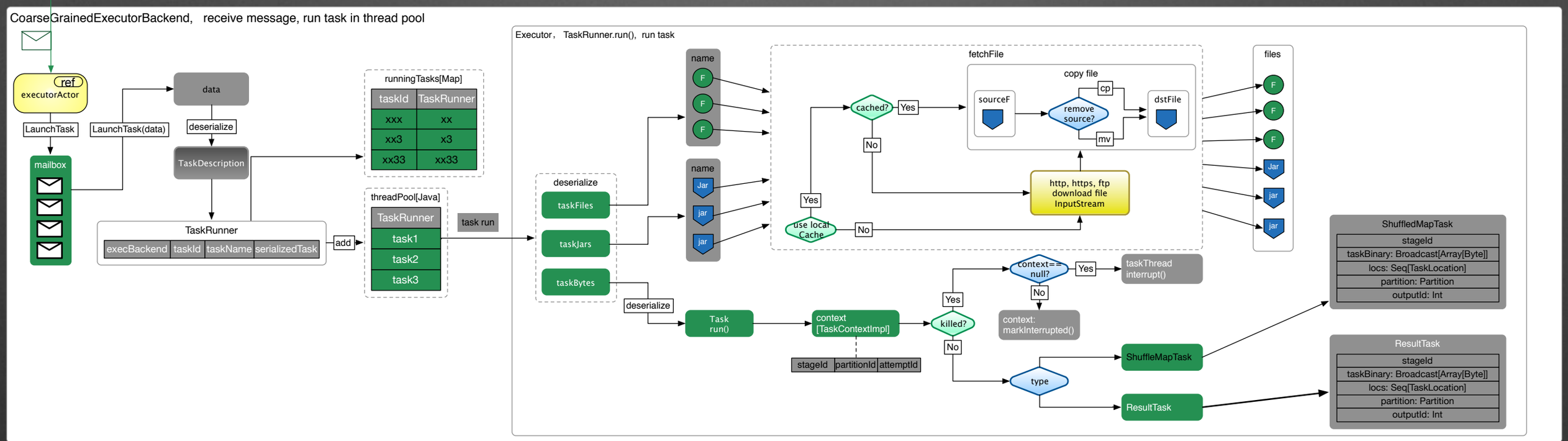
粗粒度后台调度器, 向正在running的饥饿的Worker发消息: Task来了



第6步: submitTasks(TaskSchedulerImpl)

分散在各台机器上的Worker是独立启动的main，
它收到新Task消息后：

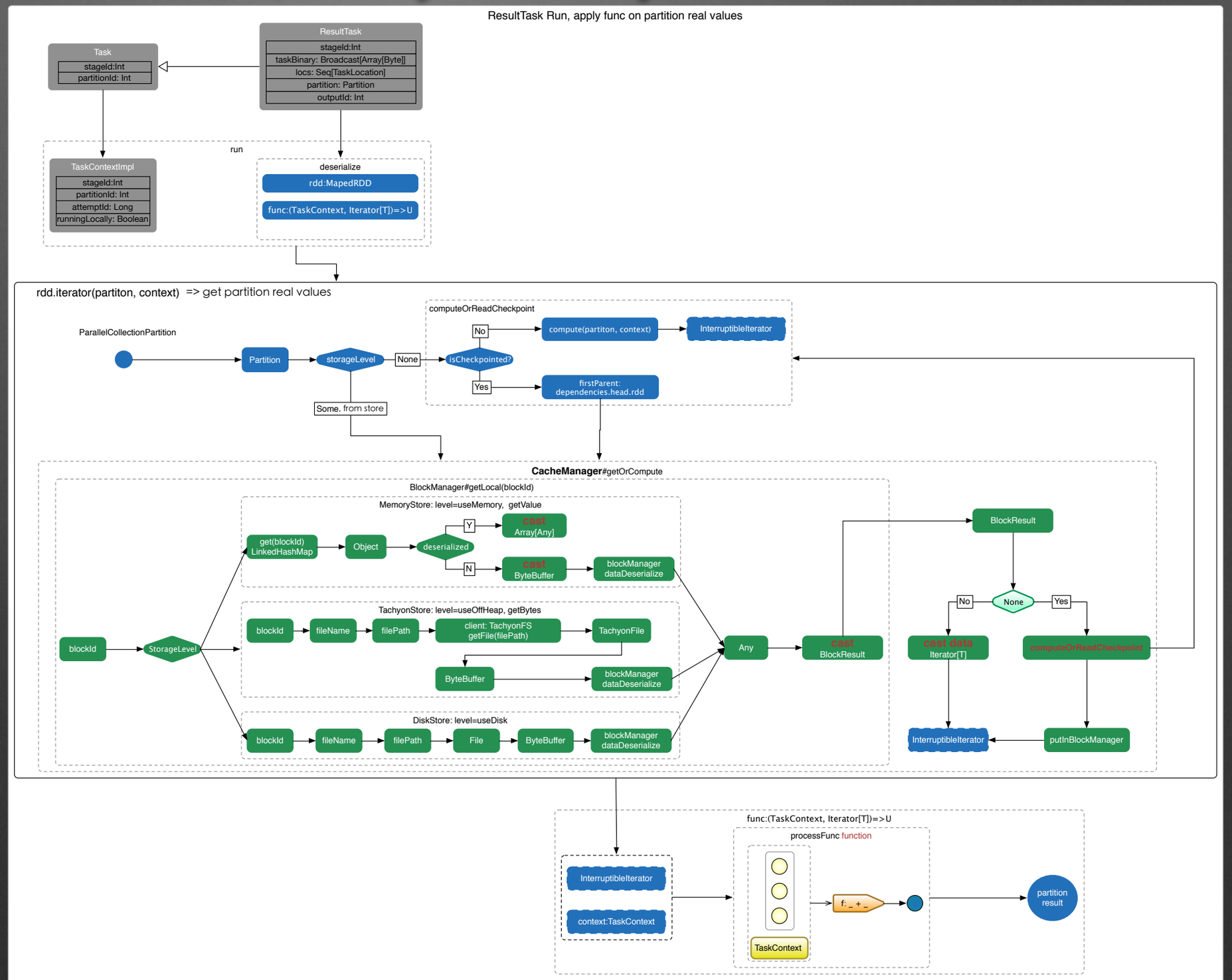
1. 从线程池中分配一个线程，让Executor去执行Task
2. Executor会把Task运行时所需的资源准备好: jar, file…



第7步: submitTasks(TaskSchedulerImpl)

这里才是硬菜:

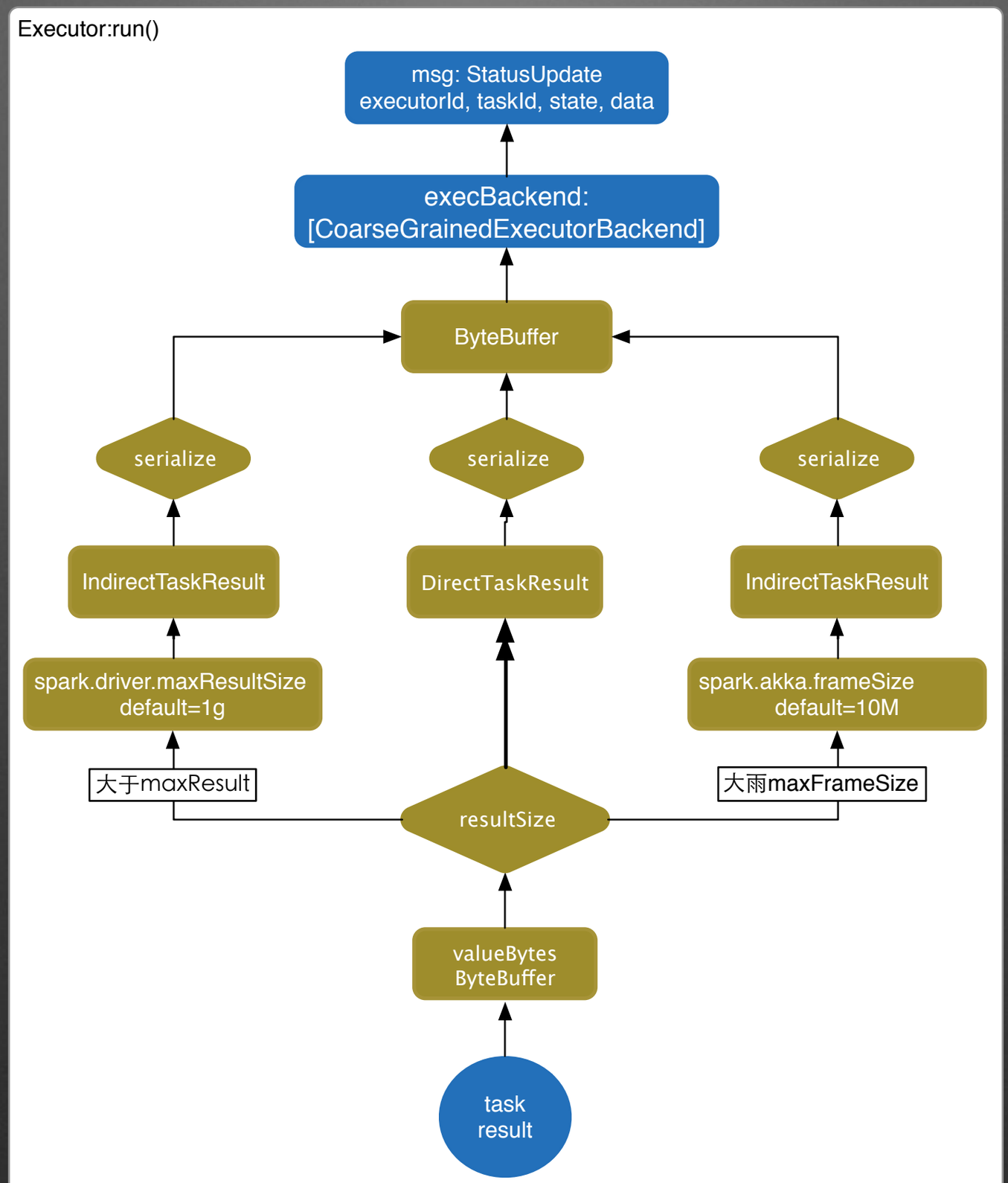
1. 反序列化出RDD及应用在partition上的计算函数
2. BlockManager从local/remote 拿到MemoryStore/TachyStore/DiskStore上的具体数据
3. 应用计算函数



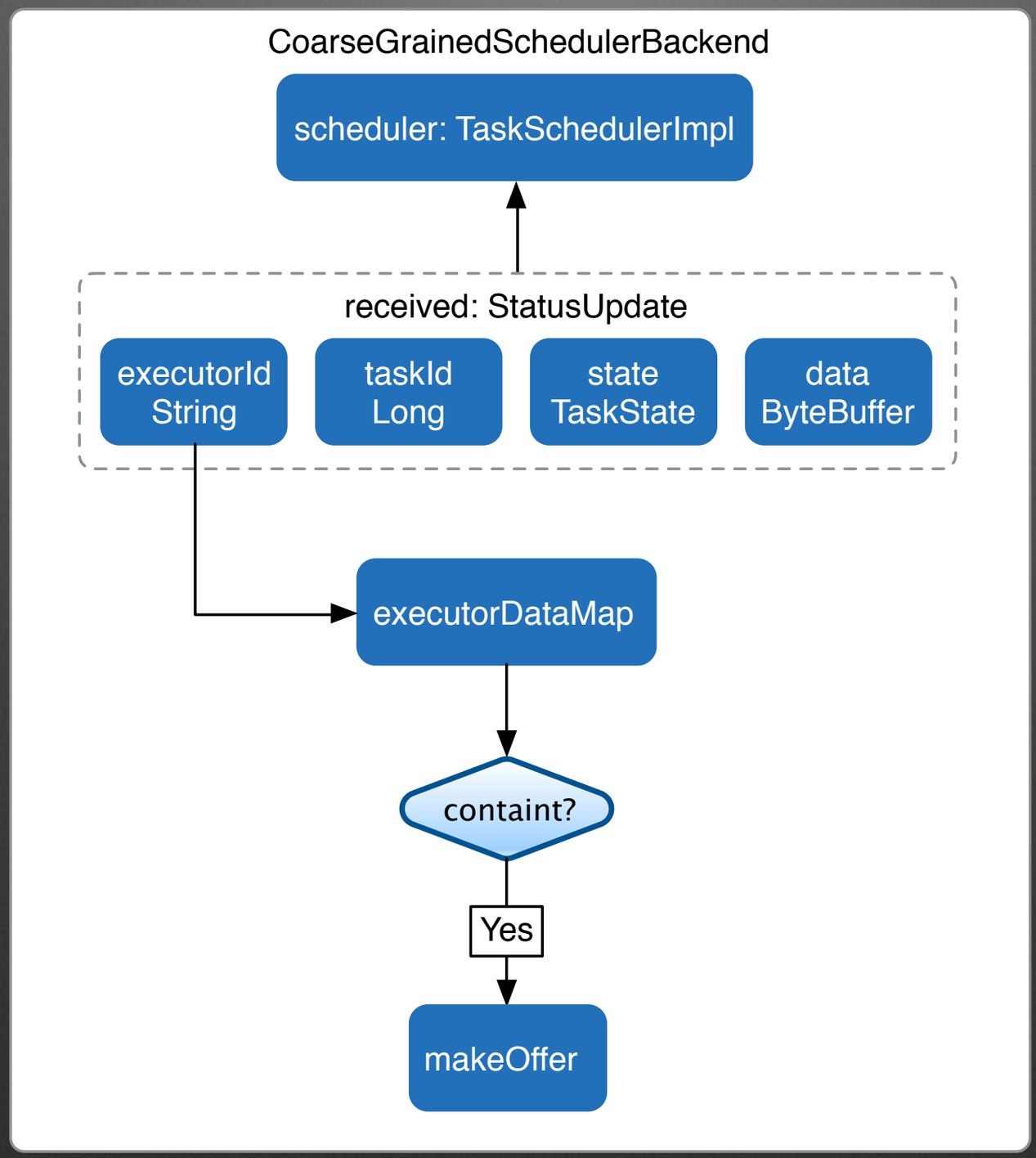
图片来自500px

原路返程

第1步：序列化partion执行结果，向上发消息给CoarseGrainedSchedulerBackend



第2步：CoarseGrainedSchedulerBackend封装StatusUpdate对象，向TaskSchedulerImpl发消息



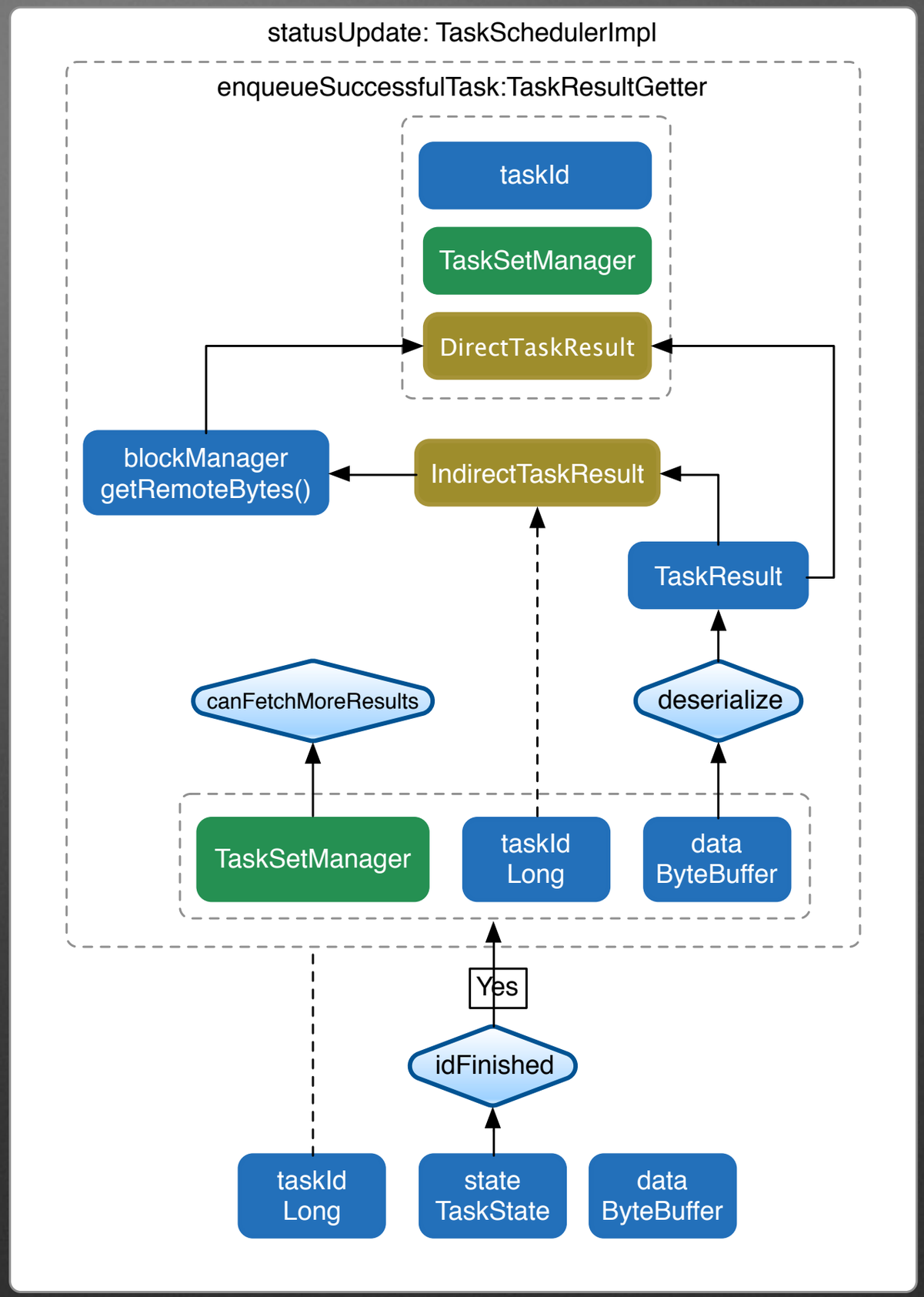
第3步：序列化partion执行结果，向上发消息

Executor中Task执行结果有两种：

1. DirectTaskResult 完整的结果
2. IndirectTaskResult 结果信息

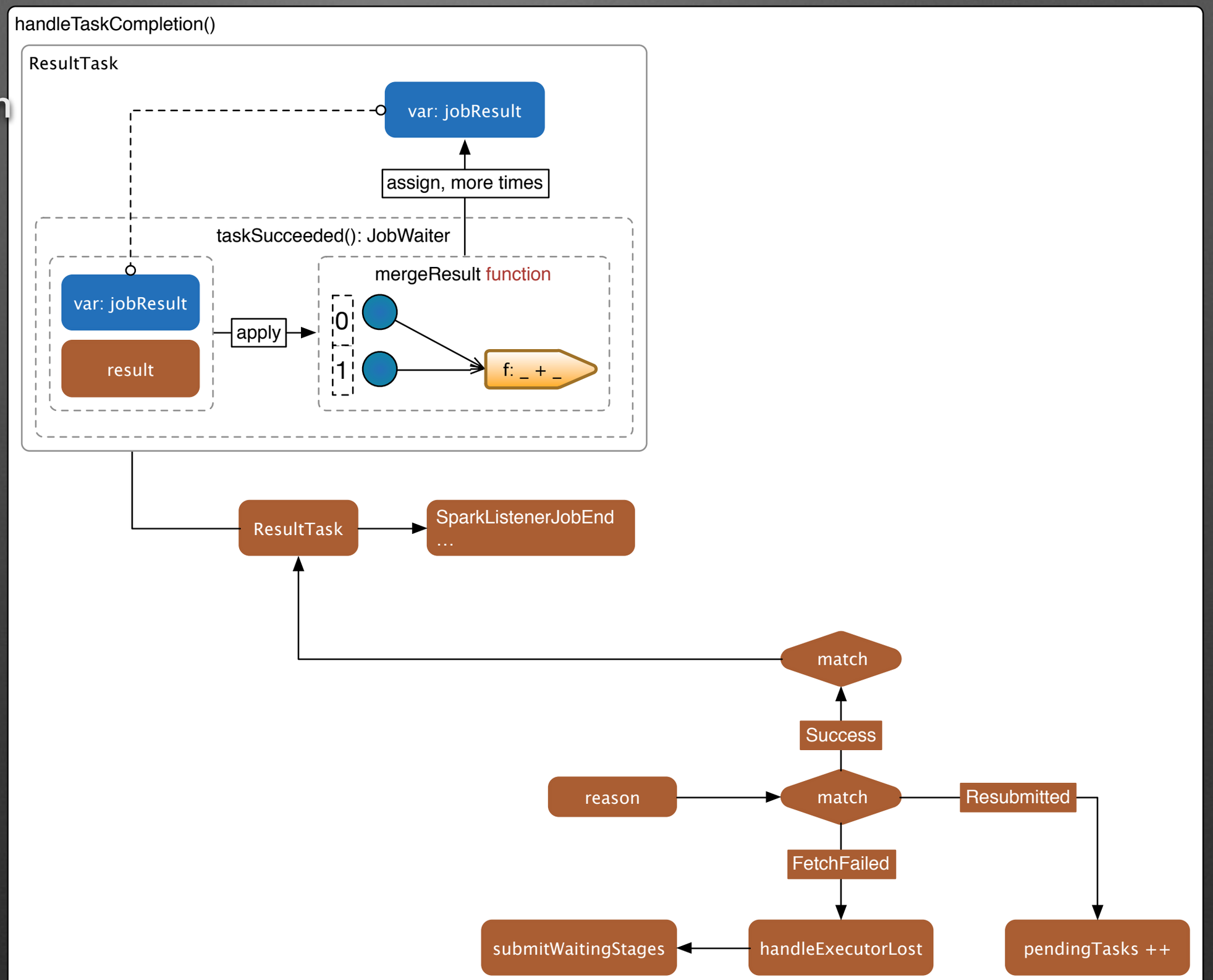
一次Akka消息发布出去，就只封装结果信息：TaskResultBlockId和resultSize

TaskSchedulerImpl如果收到IndirectTaskResult, 就要根据TaskResultBlockId和大小去Worker上拉结果数据。这是就需要NettyBlockTransferService干重活了.



第4步：DAGScheduer终结者

1. 拿到所有的partition执行结果
2. 合并结果
3. 返回给app

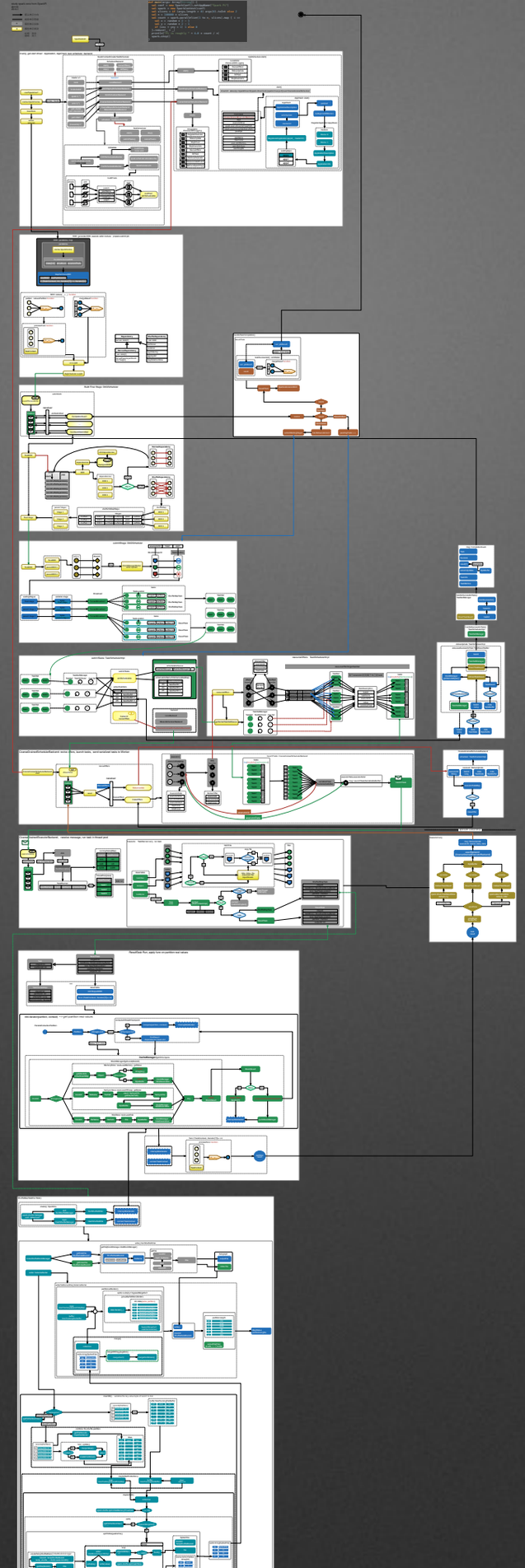


“一来一去，任务结束”

summary

- Task开箱即跑: Task在Driver上封装好的
- Partition数据: 根据partition id可以确定数据的物理位置
- 中间调度过程: 失败了可以重新提交；数据丢失了, 可以根据RDD依赖关系重新计算出来

全图:



图片来自500px

招贤纳士

1. 推荐算法工程师
2. 推荐系统专家
3. 开发工程师(Scala/Java)

mailto: mingfeng@taobao.com

Thanks