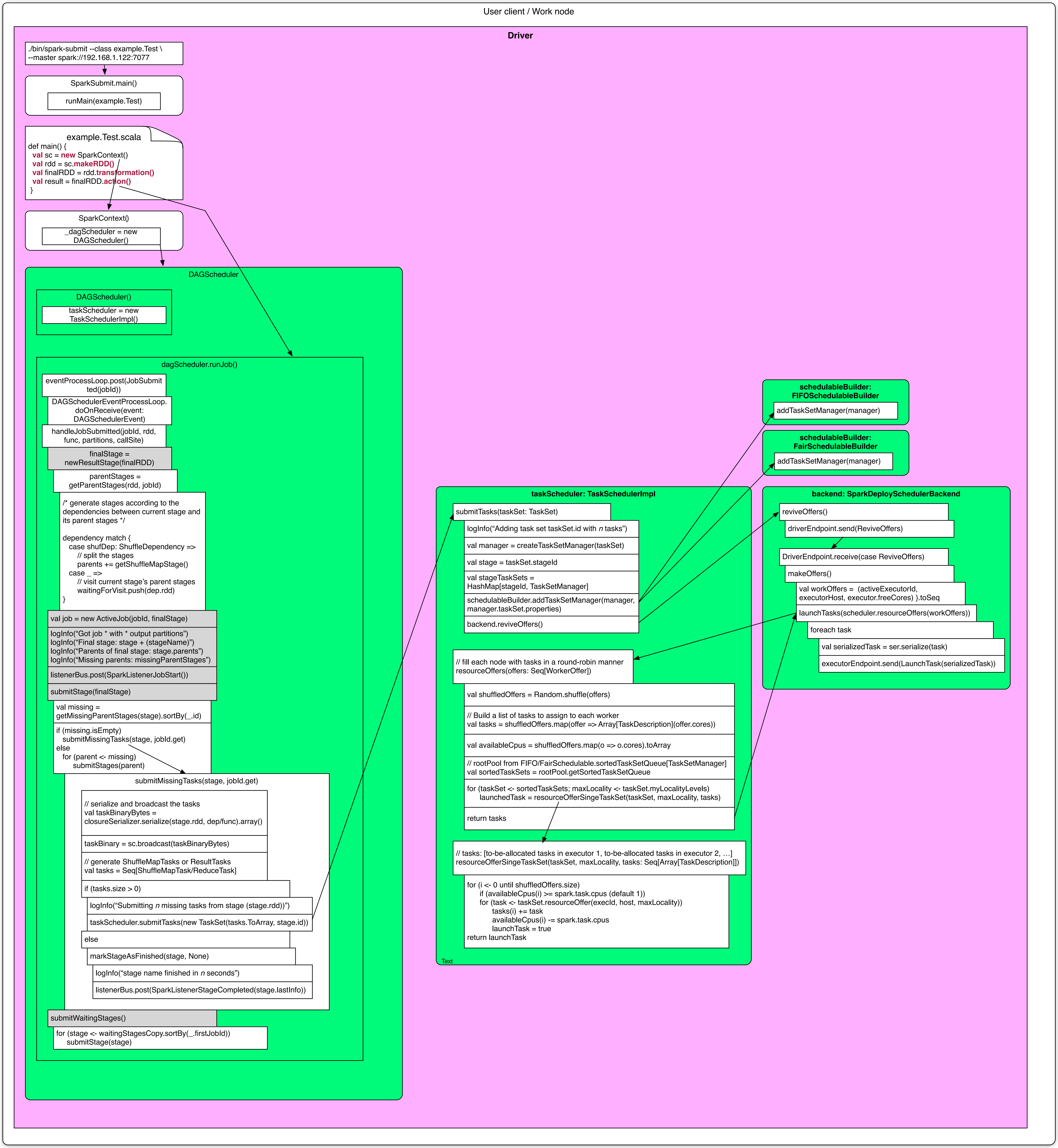
User client / Work node
Driver
DAGScheduler
dagScheduler.runJob()
taskScheduler: TaskSchedulerImpl
schedulableBuilder: FIFOSchedulableBuilder
schedulableBuilder: FairSchedulableBuilder
backend: SparkDeploySchedulerBackend
Text

```
./bin/spark-submit --class example.Test \
--master spark://192.168.1.122:7077
```

```
SparkSubmit.main()
    runMain(example.Test)
```

example.Test.scala

```
def main() {
  val sc = new SparkContext()
  val rdd = sc.makeRDD()
  val finalRDD = rdd.transformation()
  val result = finalRDD.action()
 }
```

```
SparkContext()
    _dagScheduler = new
    DAGScheduler()
```

```
DAGScheduler()
    taskScheduler = new
    TaskSchedulerImpl()
```

```
dagScheduler.runJob()
eventProcessLoop.post(JobSubmitted(jobId))
 DAGSchedulerEventProcessLoop.doOnReceive(event: DAGSchedulerEvent)
handleJobSubmitted(jobId, rdd, func, partitions, callSite)
 finalStage = newResultStage(finalRDD)
  parentStages = getParentStages(rdd, jobId)
   /* generate stages according to the
   dependencies between current stage and
   its parent stages */

   dependency match {
     case shufDep: ShuffleDependency =>
         // split the stages
         parents += getShuffleMapStage()
     case _ =>
         // visit current stage's parent stages
         waitingForVisit.push(dep.rdd)
   }
 val job = new ActiveJob(jobId, finalStage)
 logInfo("Got job * with * output partitions")
 logInfo("Final stage: stage + (stageName)")
 logInfo("Parents of final stage: stage.parents")
 logInfo("Missing parents: missingParentStages")
 listenerBus.post(SparkListenerJobStart())
 submitStage(finalStage)
  val missing =
  getMissingParentStages(stage).sortBy(_.id)
  if (missing.isEmpty)
     submitMissingTasks(stage, jobId.get)
  else
     for (parent <- missing)
         submitStages(parent)
   submitMissingTasks(stage, jobId.get)
     // serialize and broadcast the tasks
     val taskBinaryBytes =
     closureSerializer.serialize(stage.rdd, dep/func).array()
     taskBinary = sc.broadcast(taskBinaryBytes)
     // generate ShuffleMapTasks or ResultTasks
     val tasks = Seq[ShuffleMapTask/ReduceTask]
     if (tasks.size > 0)
       logInfo("Submitting n missing tasks from stage (stage.rdd))")
       taskScheduler.submitTasks(new TaskSet(tasks.ToArray, stage.id))
     else
       markStageAsFinished(stage, None)
         logInfo("stage name finished in n seconds")
         listenerBus.post(SparkListenerStageCompleted(stage.lastInfo))
 submitWaitingStages()
  for (stage <- waitingStagesCopy.sortBy(_.firstJobId))
      submitStage(stage)
```

```
taskScheduler: TaskSchedulerImpl
submitTasks(taskSet: TaskSet)
  logInfo("Adding task set taskSet.id with n tasks")
  val manager = createTaskSetManager(taskSet)
  val stage = taskSet.stageId
  val stageTaskSets =
  HashMap[stageId, TaskSetManager]
  schedulableBuilder.addTaskSetManager(manager,
  manager.taskSet.properties)
  backend.reviveOffers()

// fill each node with tasks in a round-robin manner
resourceOffers(offers: Seq[WorkerOffer])
  val shuffledOffers = Random.shuffle(offers)
  // Build a list of tasks to assign to each worker
  val tasks = shuffledOffers.map(offer => Array[TaskDescription](offer.cores))
  val availableCpus = shuffledOffers.map(o => o.cores).toArray
  // rootPool from FIFO/FairSchedulable.sortedTaskSetQueue[TaskSetManager]
  val sortedTaskSets = rootPool.getSortedTaskSetQueue
  for (taskSet <- sortedTaskSets; maxLocality <- taskSet.myLocalityLevels)
        launchedTask = resourceOfferSingeTaskSet(taskSet, maxLocality, tasks)
  return tasks

// tasks: [to-be-allocated tasks in executor 1, to-be-allocated tasks in executor 2, …]
resourceOfferSingeTaskSet(taskSet, maxLocality, tasks: Seq[Array[TaskDescription]])
  for (i <- 0 until shuffledOffers.size)
        if (availableCpus(i) >= spark.task.cpus (default 1))
        for (task <- taskSet.resourceOffer(execId, host, maxLocality))
              tasks(i) += task
              availableCpus(i) -= spark.task.cpus
              launchTask = true
  return launchTask
```

```
schedulableBuilder: FIFOSchedulableBuilder
  addTaskSetManager(manager)

schedulableBuilder: FairSchedulableBuilder
  addTaskSetManager(manager)
```

```
backend: SparkDeploySchedulerBackend
reviveOffers()
  driverEndpoint.send(ReviveOffers)
DriverEndpoint.receive(case ReviveOffers)
  makeOffers()
    val workOffers = (activeExecutorId,
    executorHost, executor.freeCores) }.toSeq
    launchTasks(scheduler.resourceOffers(workOffers))
      foreach task
        val serializedTask = ser.serialize(task)
        executorEndpoint.send(LaunchTask(serializedTask))
```