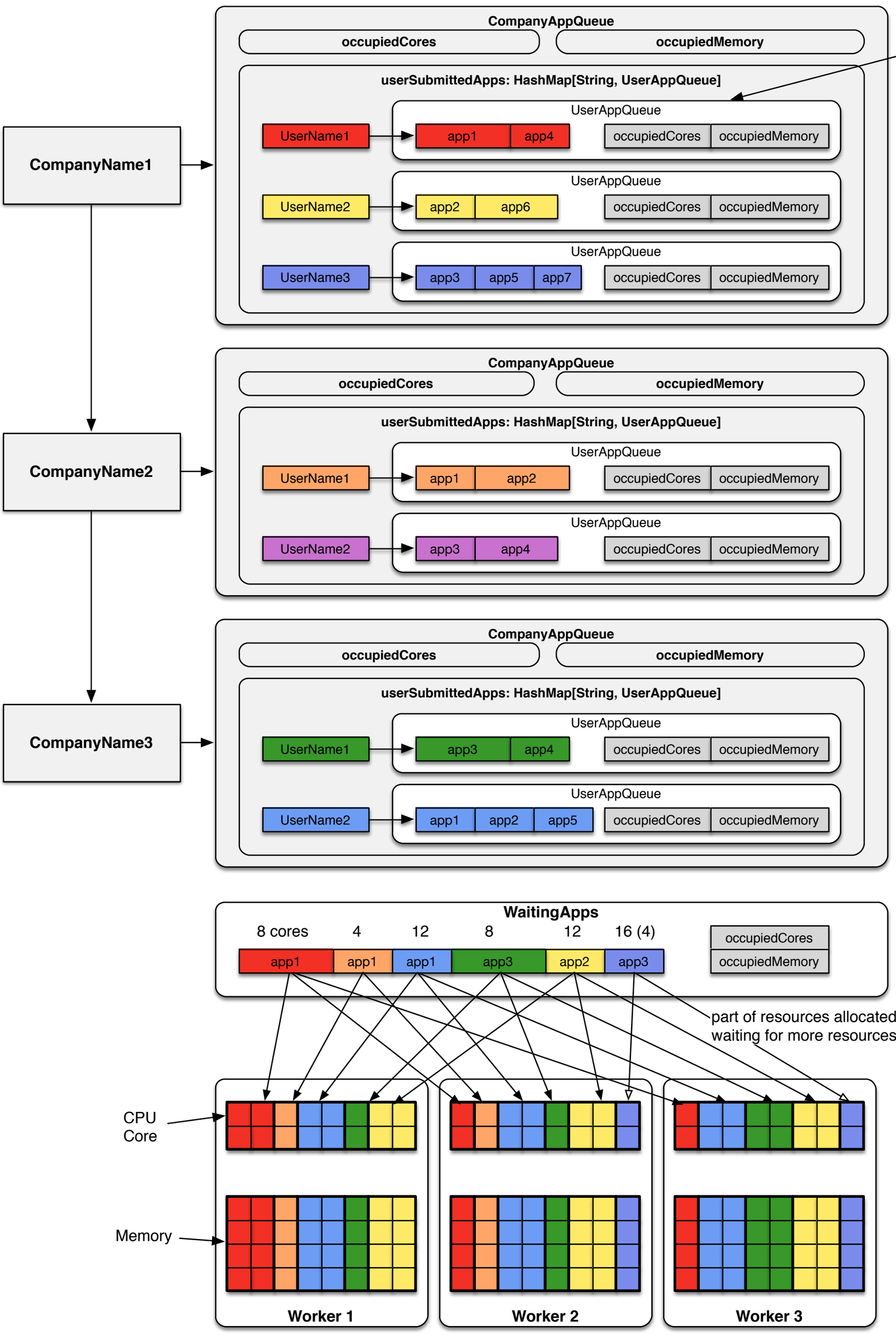
Spark 多用户资源分配方案

Master 上的 AppQueue
Map < CompanyName, Map<UserName, UserAppQueue> >



触发 Master 对 app 进行资源分配的时刻:

1. 有新 app 被提交到 Master，app = (requestCores, memoryPerTask)
2. app 执行完（可以回收资源）
3. 新 Worker 加入或退出
4. Master 出错恢复

新加入一个 app 的处理逻辑 RegisterApplication(app):

1. 加入对应的 UserAppQueue 中
2. 如果 waitingAppQueue 中的 occupiedCores 和 occupiedMemory 未占满整个集群，那么挑选合适的 app 进入 waitingAppQueue

挑选 app 进入等待队列 selectAppToWaitingAppQueue() 的逻辑:

1. 根据各公司当前资源占用情况公平挑选出下一个可运行 app 的公司
2. 在该公司内根据各个 user 资源占用情况挑选下一个可运行 app 的 user
3. 在该 user 对应的 UserAppQueue 中挑选一个 app *p* 进入 waitingApps
4. 将 app *p* 对应的 requestedCores 和 requestedMemory 累加到 company/user/waitingApps 中的 occupiedCores 和 occupiedMemory
5. 如果 waitingApps 资源仍未占满集群，继续进行 selectAppToWaitingAppQueue()
6. 调用 schedule() 方法为 waitingApps 中的 apps 分配资源，如果某些 apps 此时只被分配到一半资源，那么下次 schedule() 时继续为该 app 分配资源

删除一个 app (app 执行完成) 时的处理逻辑 RemoveApplication(app):

1. 将该 app 对应的 company/user/waitingApps 中的 occupiedCores 和 occupiedMemory 资源相应减除
2. 从 waitingApps 中删除该 app

挑选公司 company 的方法 selectCompany():

前提条件：系统管理员配置每个公司可以使用的资源数目或比例，比如 Comp1 最多使用 512 cores，1024 GB 的资源，Comp2 最多使用 128 cores，256 GB 的资源

理想挑选策略：pay-as-you-go，需要考虑每个公司购买的资源量，及目前已经占用的资源量，及占用资源的时间

目前简单方案：选取目前占用资源量百分比最小的公司。比如三个公司购买的资源量为 Comp1: Comp2: Comp3 = 512: 128: 1024，目前三个公司 occupied 资源量为 Comp1: Comp2: Comp3 = 128: 64: 128 = 25%: 50%: 12.5%，Comp3 占用资源比例最低，选取 Comp3 中的 app 进入 waitingApps 中，如果 Comp3 中没有 app，那么选取 Comp1 中的 app，依次类推

资源占用率度量：资源占用率既有 CPU cores 占用率也有 Memory 占用率，目前取两者之间的最大值

挑选公司 user 的方法 selectUser():

基本原则：保证用户之间的公平性，各个 user 可以公平地使用资源

理想挑选策略：考虑每个用户当前占用的资源量、等待时间、要运行的 app 的资源需求量等

目前简单方案：选择等待时间（距上一次被选中间隔时间）最长的 user

挑选 app 的方法:

理想挑选策略：优先保证 SQL app 的执行，app 总体是 FIFO

目前方案：先来先服务

动态分配的 occupiedCores 和 occupiedMemory 统计量一致性问题:

1. waitingApps 会统计当前队列中 apps 占用的 cores/memory 总量
2. 每个 user 占用的 cores/memory 总量
3. 每个 company 占用的 cores/memory 总量

当使用动态分配策略（比如可以动态地为 waitingApps 中的 app 增加或减少资源）时，注意统计量的一致性

另外 waitingApps 中的 occupiedCores/occupiedMemory 是否要使用真实运行时 app 占用的资源（目前只是 requestedCores/memory 的加和）

用户提交 app 时需要设置的参数:

1. requestCores: app 需要的 core 数目，也就是可以并行执行的 task 数目，一般为 max(FileSize/blocksize，partition) 个数，通过 spark.app.cores.max 设置，默认为 8
2. memoryPerTask: 每个 task 需要的 memory 大小，通过 spark.app.task.memory.mb 设置

我们避免让用户去设置一些与系统实现相关的参数：如 executor 数目，每个 executor 需要的 memory 大小等，这些参数根据既定原则及集群当前资源使用情况计算得到

系统管理员需要配置的参数

1. 每个公司可以使用的最大资源量（cores 和 memory 总数，默认为集群的 1/N(comp) ）
2. 每个 node 上可运行的 worker 数目
3. 每个 worker 可用的 cores 和 memory
4. 每个 executor 最少要包含的 cores 和 memory（默认为 cores = 4, memory = 2GB）
   设置该参数的目的是我们想让 executor 的大小是 4 的倍数，同时不想让一个 executor 过大（比如 cores = 32, memory=16GB），因为 JVM 管理大内存时 GC 效率很低。Executor 的

集群资源配置一般准则:

1. 每个 Worker 上配置的可用 CPU cores 是 2 的整数倍（比如 4, 8, 16）
2. 每个 Worker 上配置的 Memory 是 512MB 的整数倍 （如 1GB, 1.5GB）
3. 分配给每个 Executor 的 cores 个数为 2 的整数倍，memory 大小为 256/512MB 的整数倍

Executor 数目确定及分配策略 startExecutorsOnWorkers():

基本原则:
1. 一个 app 可以包含多个 executor，每个 executor 不超过 16 cores
2. 每个 worker 上可以运行多个 executor

当前实现方法:
1. 首先在集群中挑选出至少可以运行一个最小 executor 的 usableWorkers，也就是说被选中的 worker 至少剩余 4 cores 和 2GB memory
2. 计算每个 executor 应该包含多少 cores
   实现方法在 computeCoresPerExecutor()
   如果 usableWorkers 中大部分有 8 个 free cores，而且当前 app 比较大（比如 requestedCores > = usableWorkers * 8），那么每个 executor 分配 8 个 cores
   如果 usableWorkers 中大部分有 16 个 free cores，而且当前 app 非常大，那么每个 executor 分配 16 个 cores

3. 计算每个 executor 应该包含多少 memory
   实现方法在 computeMemoryPerExecutorMB()
   如果用户设置了 memoryPerTask，那么每个 executor memory 大小为 memoryPerTaskMB * coresPerExecutor
   如果用户没有设置 memoryPerTask，先选出 cores > coresPerExecutor 的 workers，然后计算这些 workers 剩余的最小内存 minFreeMemory
   如果 minFreeMemory / app.coresPerExecutor <= 256，那么 executor 内存大小为 256MB * coresPerExecutor
   否则，executor 内存大小为 512MB * coresPerExecutor