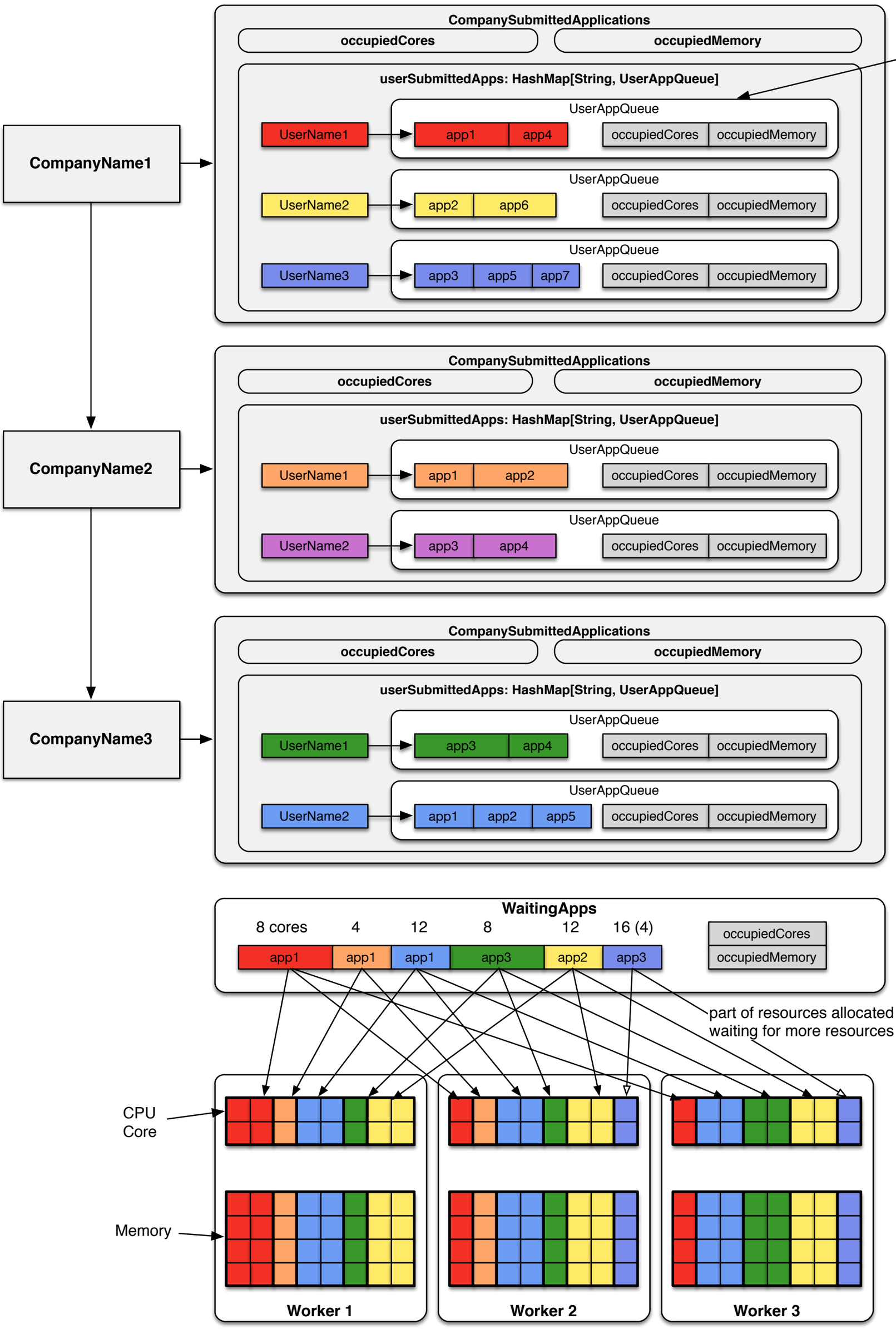
Spark 多用户资源分配方案

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 (requestCores, memoryPerTask)
2. app 执行完成要回收资源
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 waitingApps 中的 occupiedCores 和 occupiedMemory 未占满整个集群，那么触发资源分配方法 allocateResources() 进行资源分配

资源分配方法 allocateResources() 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 requestCores 和 requestedMemory 累加到 company/user/waitingApps 中的 occupiedCores 和 occupiedMemory
5. 如果 waitingApps 资源仍未占满，那么继续进行 allocateResources()
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) 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:

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 Company1 最多使用 512 cores，1024 GB 的资源，company2 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:

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:
1. 如果某个 user 等待时间（距离上一次被选中的间隔时间）超过一个阈值（如半小时），那么选择该 user
2. 如果等待时间都在阈值内，那么挑选目前资源占用比例最低（也就是 occupiedCores 和 occupiedMemory 最少）的用户

集群资源配置方法:

1. 每个 Worker 上的 CPU cores 是 2 的整数倍（比如 4, 8, 16）
2. 每个 Worker 上的 Memory 是 512MB 的整数倍 （如 1GB, 1.5GB）
3. 每个 Executor 需要的 cores 个数为 2 的整数倍，memory 大小为 512MB 的整数倍

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

1. waitingApps 中有各个 app 占用的 cores/memory 总量
2. 每个用户有占用的 cores/memory 总量
3. 每个公司有占用的 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，这些统计量会出现不一致

另外 waitingApps 中的 occupiedCores/occupiedMemory 是否要使用真实运行时 app 占用的资源

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 reducer 个数
2. **memoryPerTask:** 每个 task 需要的 memory 大小

比如，
Spark.core.max (requestedCores) = 8
taskPerMemory = 256MB

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

Executor 数目确定及分配原则:

1. 一个 worker 尽量为一个 app 只分配一个 executor
2. 每个 executor 不超过 16 个 cores

假设 cpuCoresPerWorker = *c* = 8

```
if (requestedCores >= workerNum * c)
    coresPerExecutor = 16
    (每个 worker 会有多个 executor)
else if (requestedCores >= workerNum * c / 2)
    coresPerExecutor = 8
else
    coresPerExecutor = 4
```