

# DPark

轻量友好的MapReduce框架

刘洪清 (Davies) [davies@douban.com](mailto:davies@douban.com)  
2011/12/07 Velocity China 2011

豆瓣 

- 背景
- 计算模型
- 示例
- 总结

# Douban 的规模

# Douban 的规模

- 5500万用户

# Douban 的规模

- 5500万用户
- 1000G 日志数据, 每月

# Douban 的规模

- 5500万用户
- 1000G 日志数据, 每月
- 60+ 台物理服务器

# Douban 的规模

- 5500万用户
- 1000G 日志数据, 每月
- 60+ 台物理服务器
- 200+ 员工

# Douban 的规模

- 5500万用户
- 1000G 日志数据, 每月
- 60+ 台物理服务器
- 200+ 员工
- 开发效率 > 执行效率

# 基础架构

# 基础架构

- MooseFS
  - 260 T, 44 节点

# 基础架构

- MooseFS
  - 260 T, 44 节点
- InfoBright ICE
  - weblog, 60+ B 记录

# 基础架构

- MooseFS
  - 260 T, 44 节点
- InfoBright ICE
  - weblog, 60+ B 记录
- Python, R, MPI (C++)

# Hadoop

# Hadoop

- 08 年，开始尝试

# Hadoop

- 08 年，开始尝试
  - 矩阵运算效果不理想

# Hadoop

- 08 年，开始尝试
  - 矩阵运算效果不理想
- 09 年，尝试使用

# Hadoop

- 08 年，开始尝试
  - 矩阵运算效果不理想
- 09 年，尝试使用
  - HDFS, MapR, ZooKeeper, Hive

# Hadoop

- 08 年，开始尝试
  - 矩阵运算效果不理想
- 09 年，尝试使用
  - HDFS, MapR, ZooKeeper, Hive
  - Dumbo

# 计算方面的问题

# 计算方面的问题

- 计算资源管理和调度

# 计算方面的问题

- 计算资源管理和调度
- Hadoop, MPI 等等

# 计算方面的问题

- 计算资源管理和调度
  - Hadoop, MPI 等等
- 更好用的计算框架

# 计算方面的问题

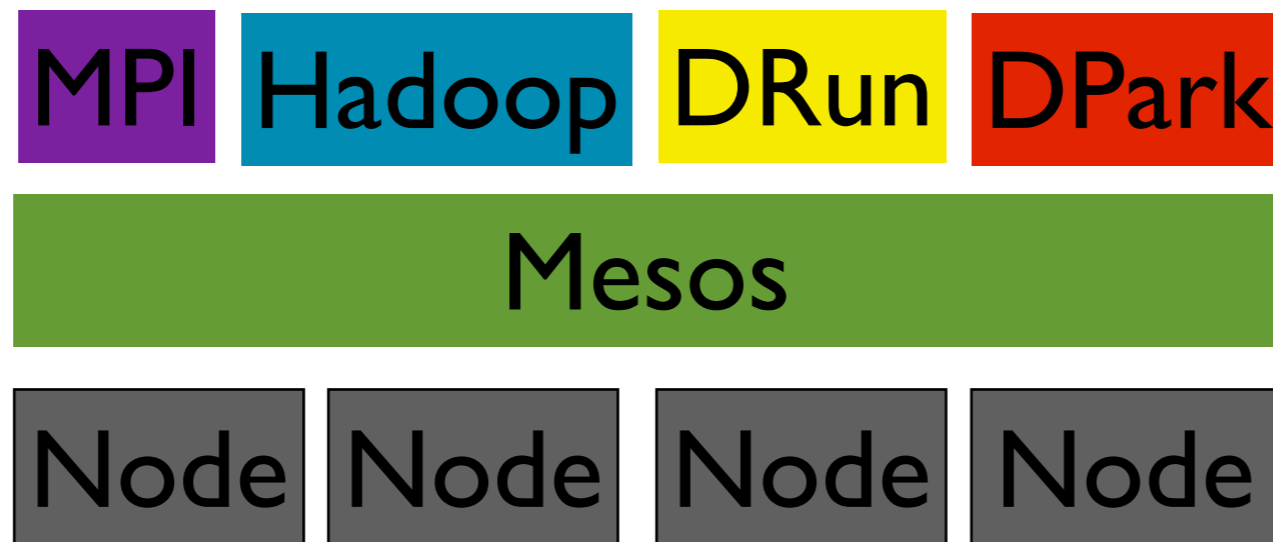
- 计算资源管理和调度
  - Hadoop, MPI 等等
- 更好用的计算框架
  - 比 Hadoop 轻量，灵活

# 计算方面的问题

- 计算资源管理和调度
  - Hadoop, MPI 等等
- 更好用的计算框架
  - 比 Hadoop 轻量，灵活
  - 比 MPI 简单，鲁棒

# Apache Mesos\*

- 资源管理和调度框架
- 开放资源调度接口
- 隔离，资源限制，Linux Container



<http://www.mesosproject.org/>

豆瓣 [douban](http://www.douban.com/)

# DRun

# DRun

- 基于 Mesos 的作业提交

# DRun

- 基于 Mesos 的作业提交
- 支持 MPICH2

# DRun

- 基于 Mesos 的作业提交
- 支持 MPICH2
- 支持 ad-hoc 并行计算, RANK, SIZE

# DRun

- 基于 Mesos 的作业提交
  - 支持 MPICH2
  - 支持 ad-hoc 并行计算, RANK, SIZE

```
$ drun -n 10 -c 1 -m 10m -r 2 hostname
```

```
[3@alg221] alg221
```

```
[2@alg224] alg224
```

```
.....
```

# DRun 案例

# DRun 案例

- 音频压缩
  - 70万，AAC 格式

# DRun 案例

- 音频压缩
  - 70万，AAC 格式
- 抓取北京车牌号抽签结果
  - 几十万页面

# DRun + Mesos

- 解决了计算资源管理和调度问题

# Spark\*

- Lightning-Fast Cluster Computing

```
val file = spark.textFile("hdfs://...")  
file.flatMap(line => line.split(" "))  
  .map(word => (word, 1))  
  .reduceByKey(_ + _)
```

## Word Count in Spark

\* <http://www.spark-project.org/>

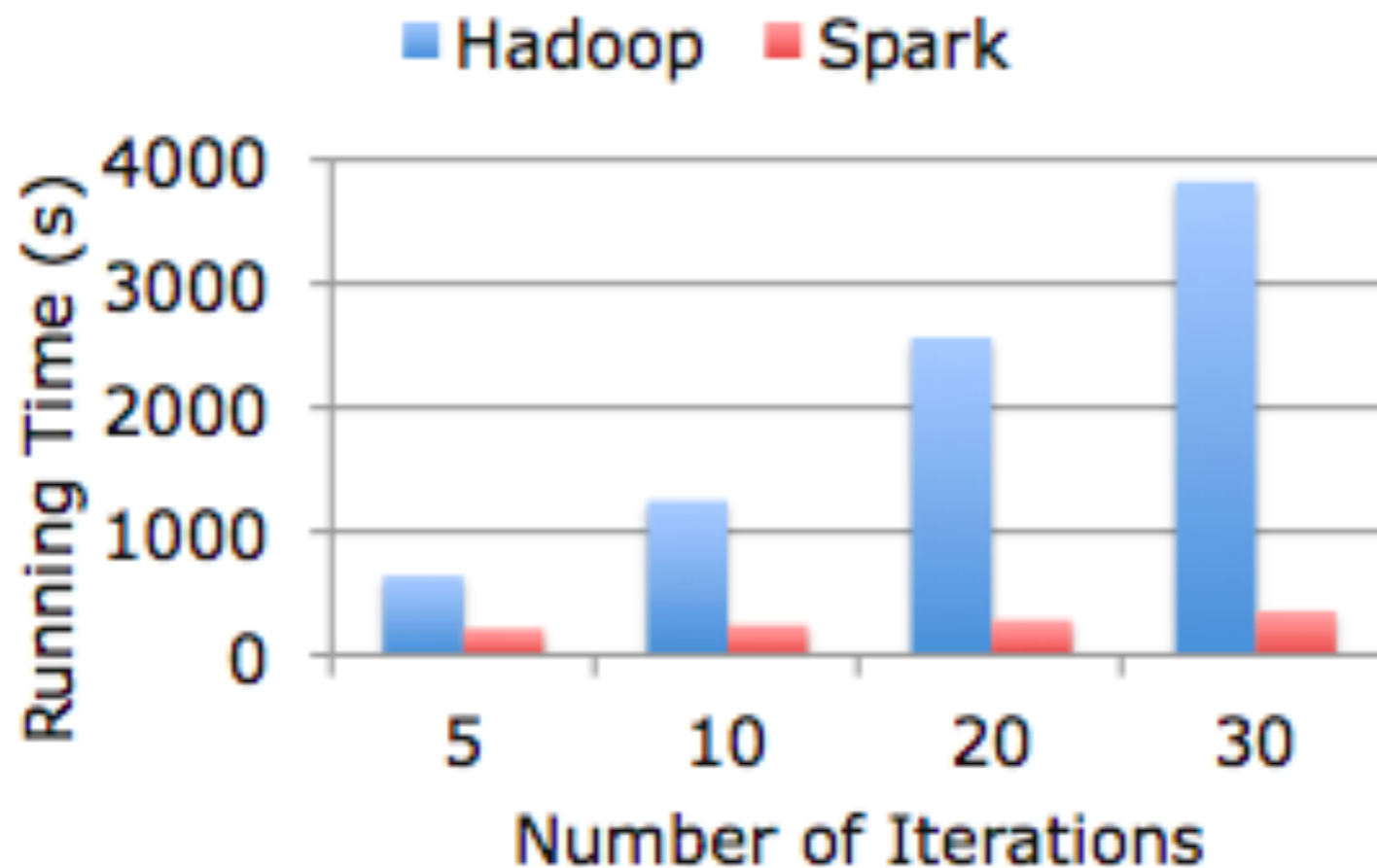
豆瓣 [douban](http://www.douban.com/)

# Spark

- 更好地支持迭代计算

# Spark

- 更好地支持迭代计算



Logistic regression in Spark vs Hadoop

# Spark

# Spark

[Spark: Cluster Computing with Working Sets](#). Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, Ion Stoica. *USENIX HotCloud 2010*. June 2010.

- 1 [Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing](#). Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, Ion Stoica. *Technical Report UCB/EECS-2011-82*. July 2011.

# Spark

# Spark

- 优点

# Spark

- 优点
  - 简洁, Scala, 10000 locs

# Spark

- 优点
  - 简洁， Scala, 10000 locs
  - 迭代计算更友好

# Spark

- 优点
  - 简洁， Scala, 10000 locs
  - 迭代计算更友好
- 缺点

# Spark

- 优点
  - 简洁， Scala, 10000 locs
  - 迭代计算更友好
- 缺点
  - 学习 Scala

# Spark

- 优点
  - 简洁， Scala, 10000 locs
  - 迭代计算更友好
- 缺点
  - 学习 Scala
  - 不能复用积累的 Python / C代码

# DPark 诞生

# DPark 诞生

- Spark 的 Python 克隆

# DPark 诞生

- Spark 的 Python 克隆
  - 拥抱Python

# DPark 诞生

- Spark 的 Python 克隆
  - 拥抱Python
- 第 7 天，基本可用

# DPark 诞生

- Spark 的 Python 克隆
  - 拥抱Python
- 第 7 天，基本可用
  - 比Hive 快 15 倍 :-)

# DPark 诞生

- Spark 的 Python 克隆
  - 拥抱Python
- 第 7 天，基本可用
  - 比Hive 快 15 倍 :-)
- 持续改进，性能优化

# DPark 诞生

- Spark 的 Python 克隆
  - 拥抱Python
- 第 7 天，基本可用
  - 比Hive 快 15 倍 :-)
- 持续改进，性能优化
  - 取代Hadoop

# DPark 计算模型

# DPark 计算模型

- RDD
  - 弹性分布式数据集(Resilient Distributed Dataset)
  - 只读，可拆分
  - 惰性，需要时才计算，可完全重建

# DPark 计算模型

- RDD
  - 弹性分布式数据集(Resilient Distributed Dataset)
  - 只读，可拆分
  - 惰性，需要时才计算，可完全重建

RDD

Split Split Split Split····· Split

# 构造RDD

# 构造RDD

- 数组

# 构造RDD

- 数组
  - `dpark.parallelize(range(100), 10)`

# 构造RDD

- 数组
  - `dpark.parallelize(range(100), 10)`
- 分布式文件系统中的文件

# 构造RDD

- 数组
  - `dpark.parallelize(range(100), 10)`
- 分布式文件系统中的文件
  - `dpark.textFile('/mfs/weblog/')`

# 构造 RDD

# 构造 RDD

- RDD 经过变换得到新的RDD

# 构造 RDD

- RDD 经过变换得到新的RDD
  - 变换是惰性的

# 构造 RDD

- RDD 经过变换得到新的RDD
- 变换是惰性的

FileRDD

Split Split Split····· Split

# 构造 RDD

- RDD 经过变换得到新的RDD
- 变换是惰性的

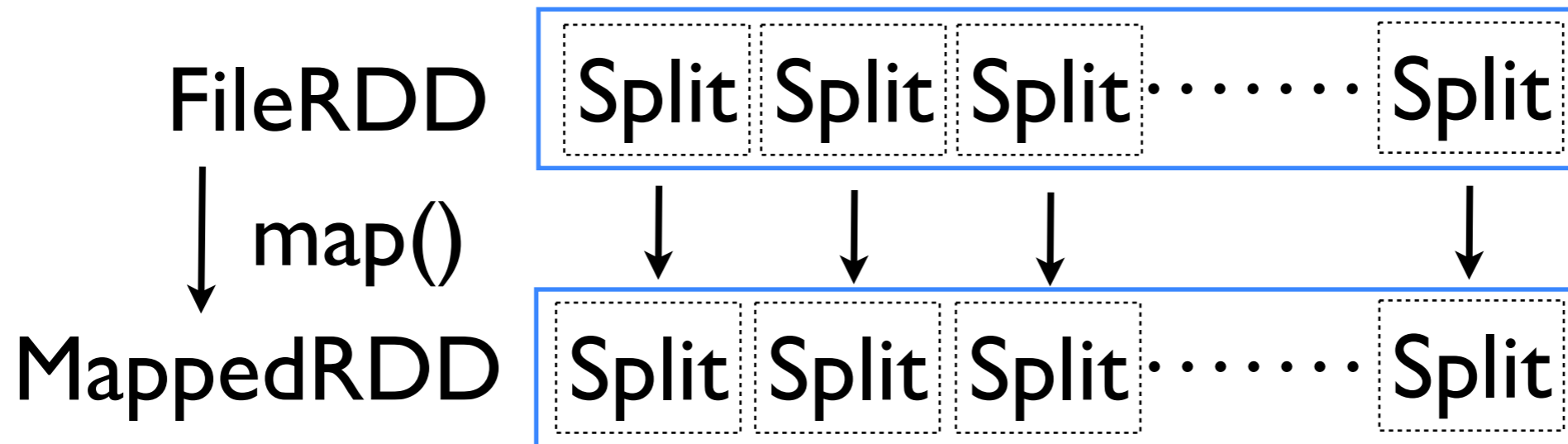
FileRDD

Split Split Split ····· Split

↓ map()

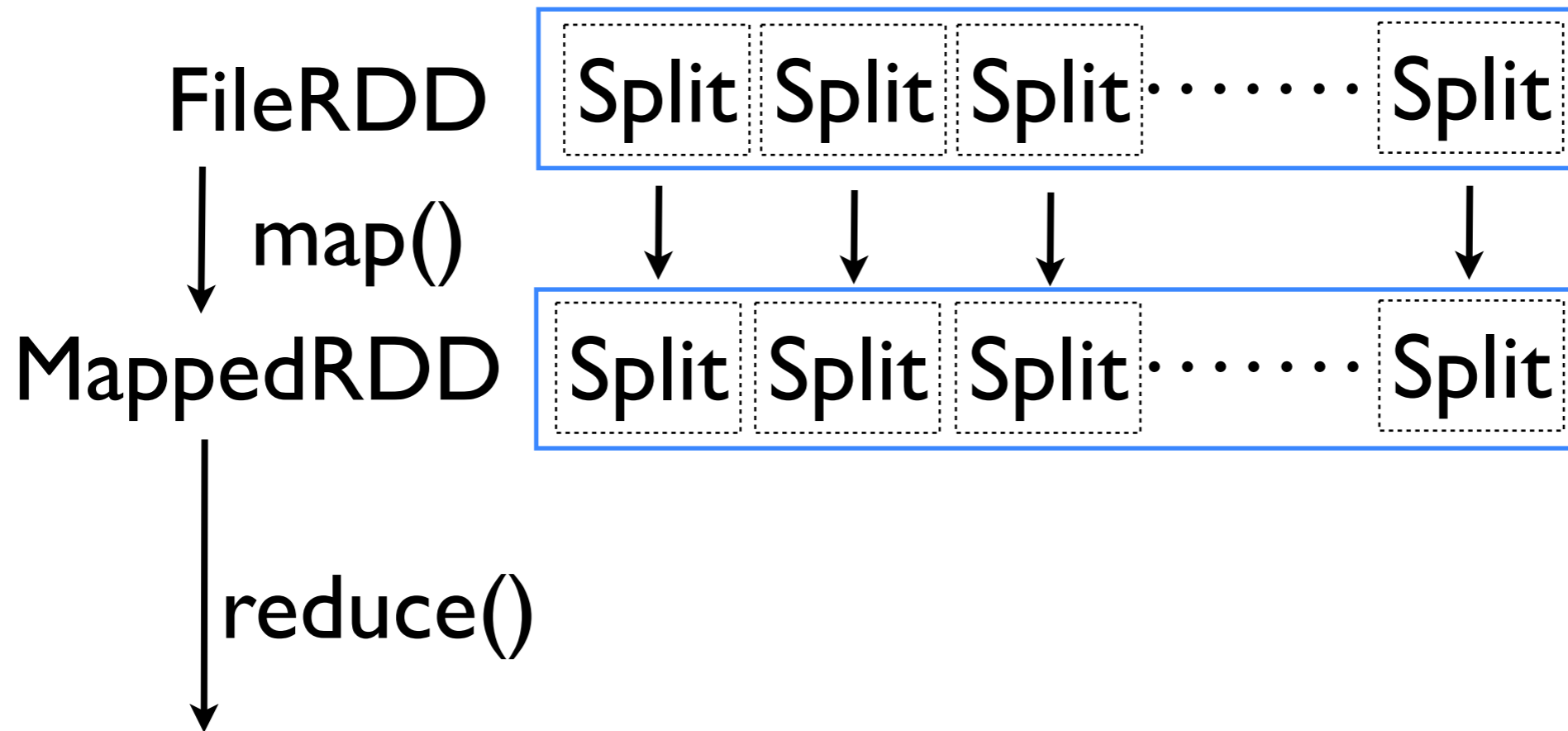
# 构造 RDD

- RDD 经过变换得到新的RDD
- 变换是惰性的



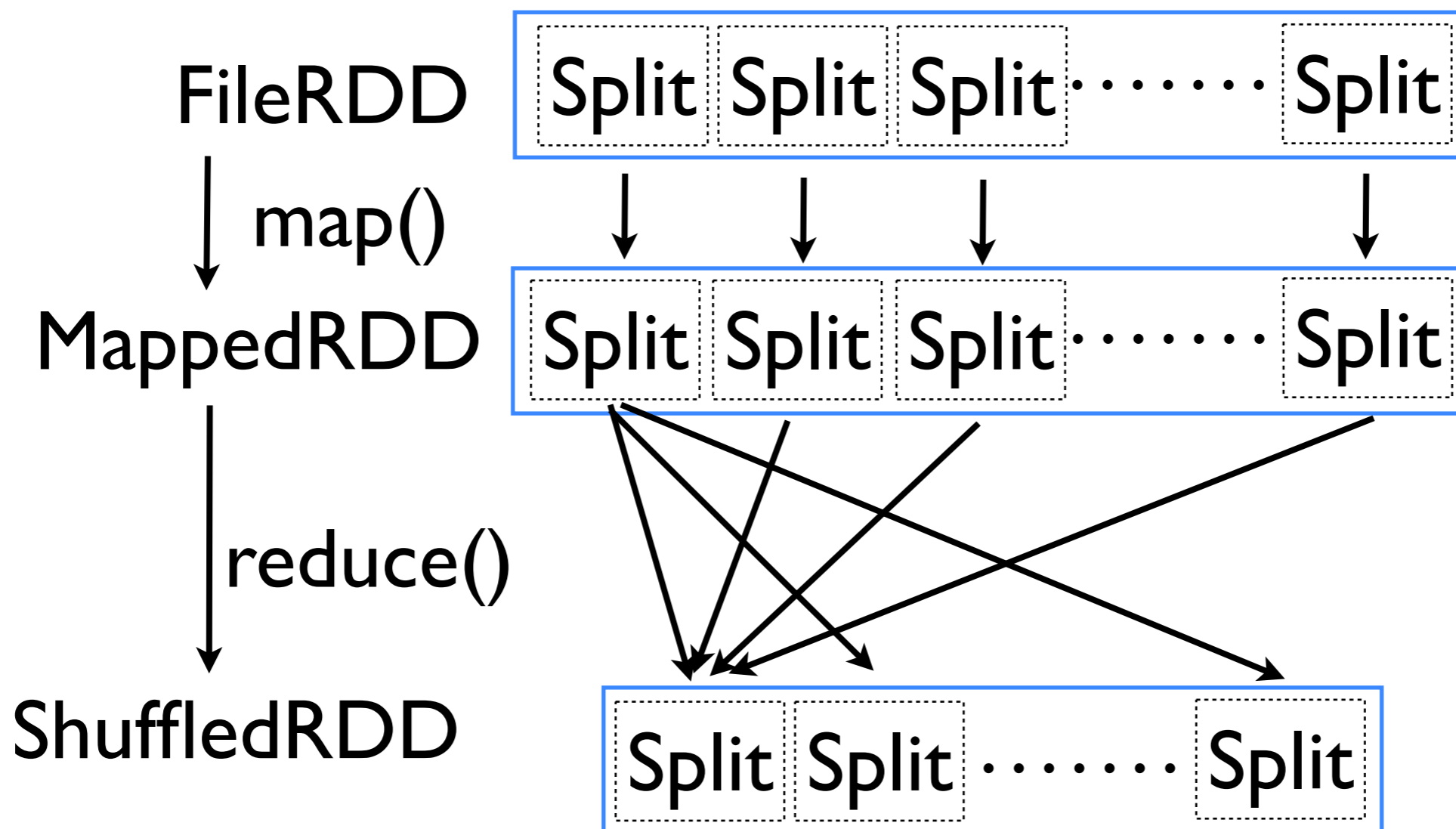
# 构造 RDD

- RDD 经过变换得到新的RDD
- 变换是惰性的



# 构造 RDD

- RDD 经过变换得到新的RDD
- 变换是惰性的

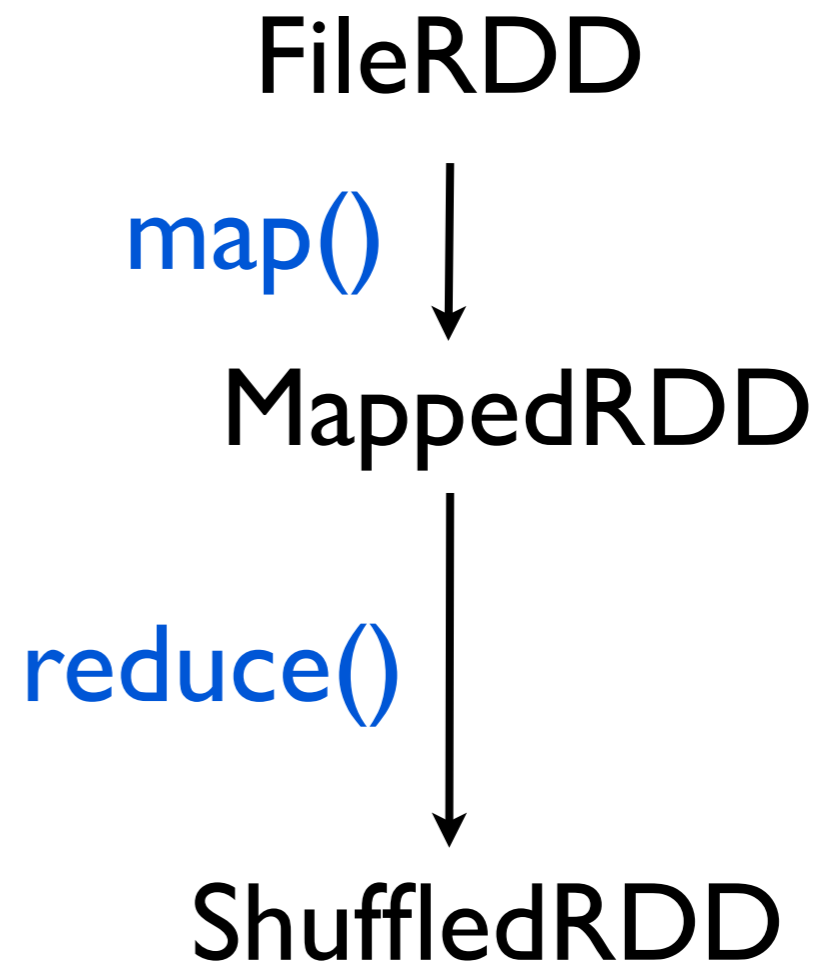


# RDD计算过程

- 由操作发起计算
  - `count()`, `collect()`, `save()`
- 根据依赖关系生成 DAG
- 划分Stage
  - Shuffle, Result

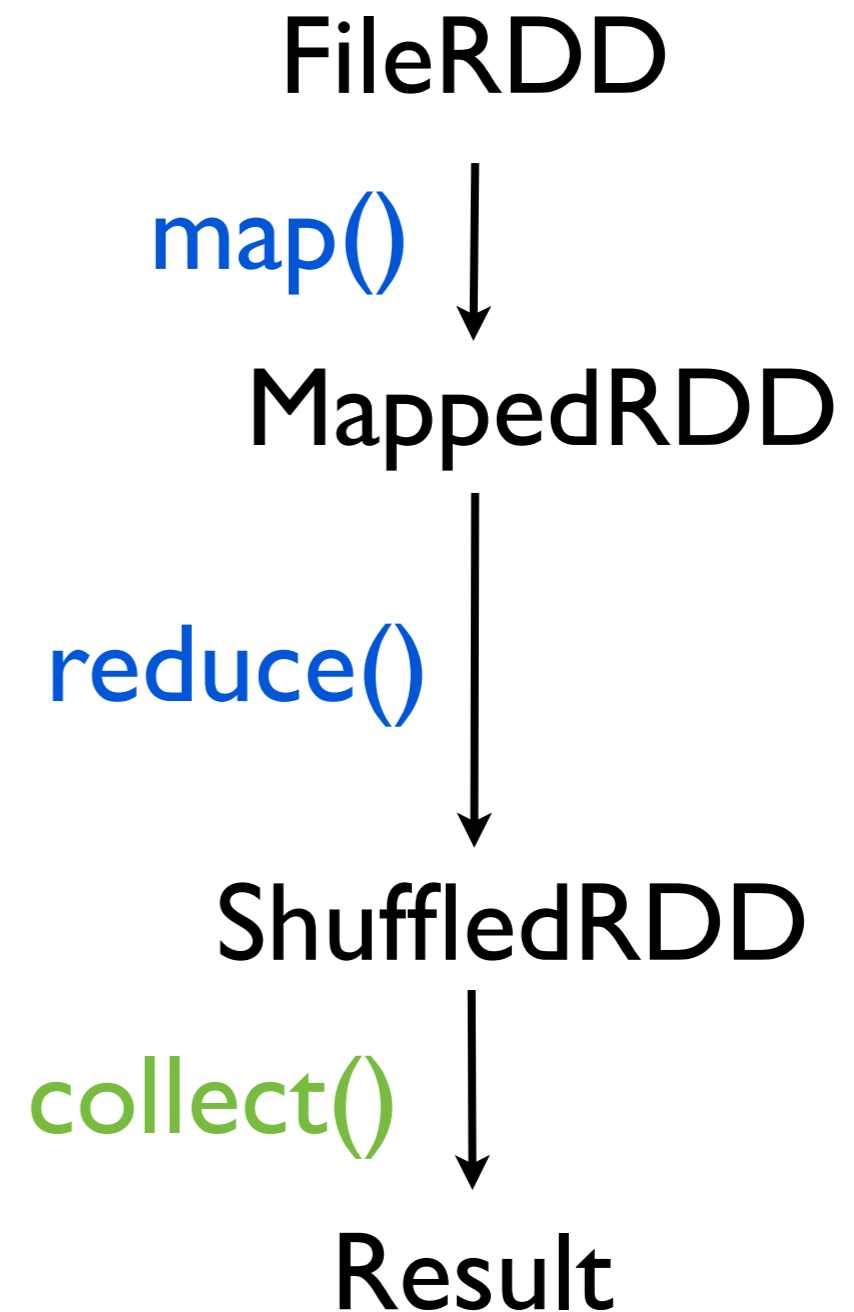
# RDD计算过程

- 由操作发起计算
- `count()`, `collect()`, `save()`
- 根据依赖关系生成 DAG
- 划分Stage
- Shuffle, Result



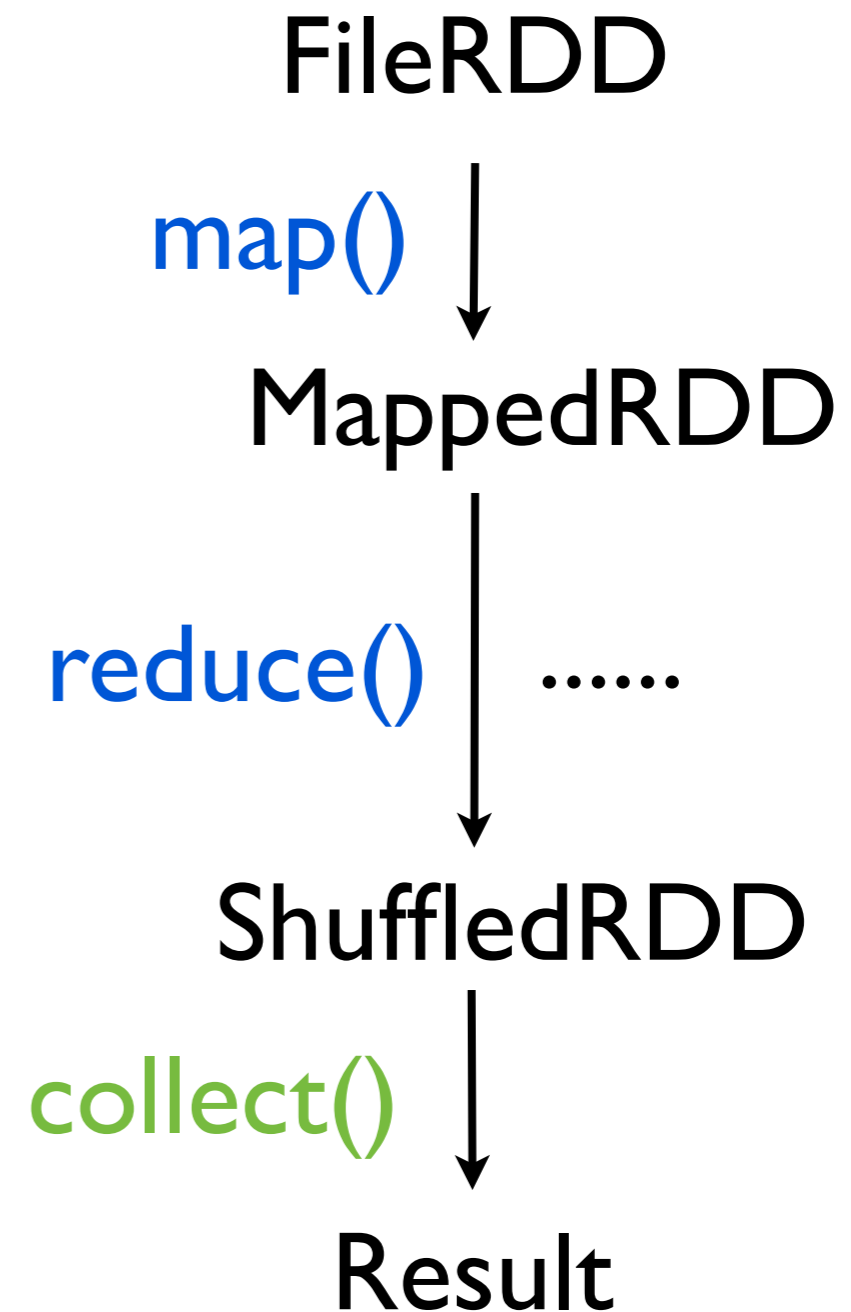
# RDD计算过程

- 由操作发起计算
- `count()`, `collect()`, `save()`
- 根据依赖关系生成 DAG
- 划分Stage
- Shuffle, Result



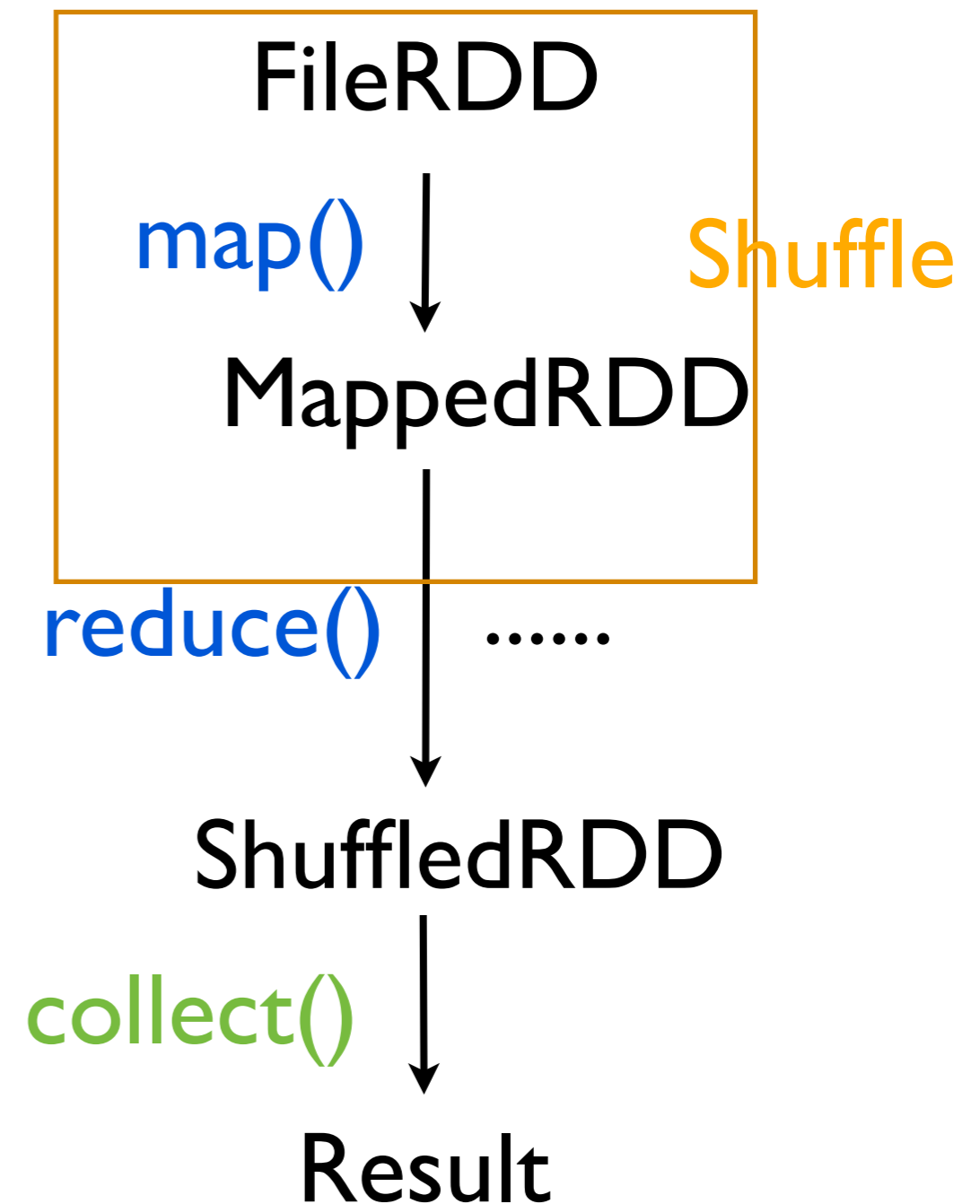
# RDD计算过程

- 由操作发起计算
- `count()`, `collect()`, `save()`
- 根据依赖关系生成 DAG
- 划分Stage
- Shuffle, Result



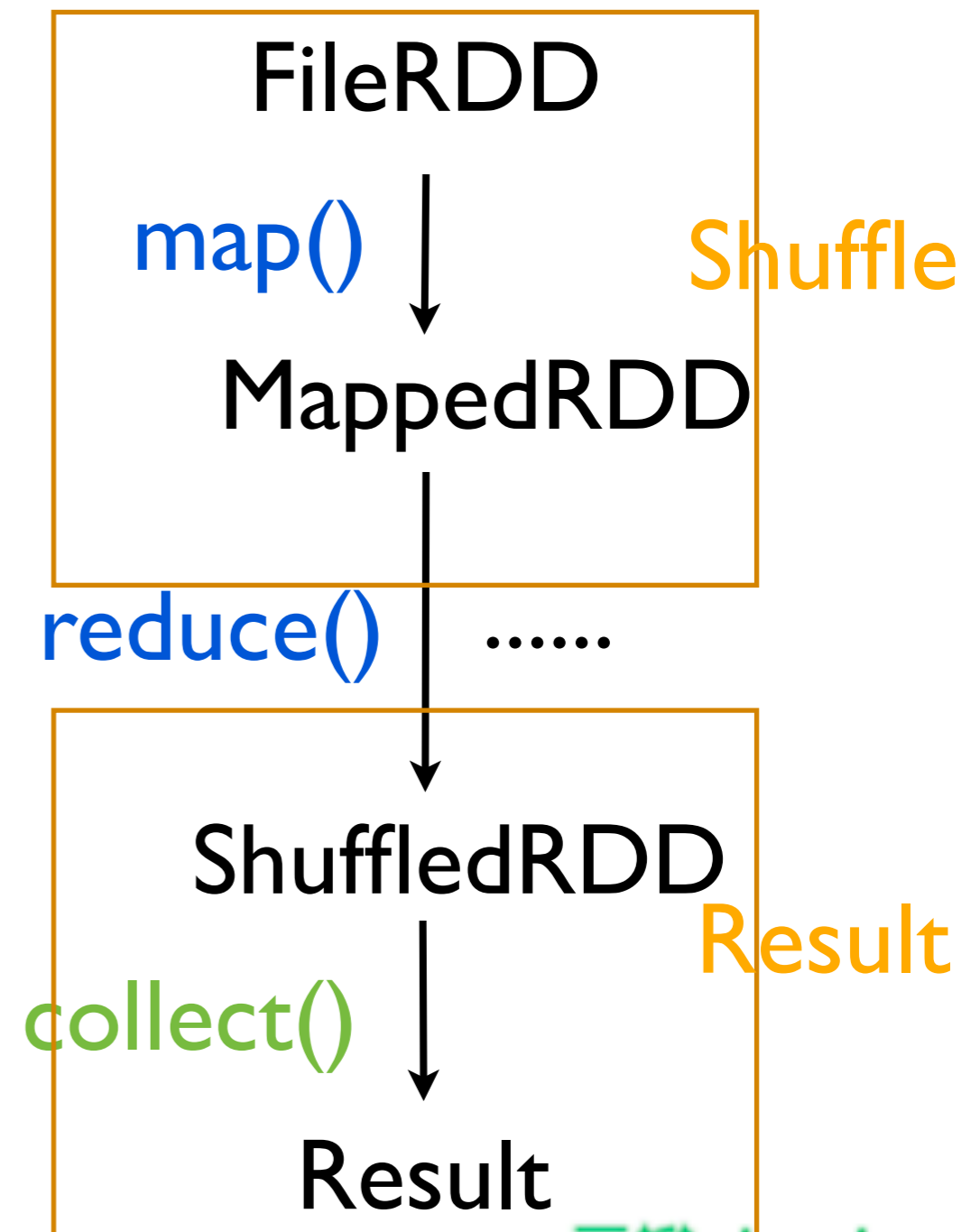
# RDD计算过程

- 由操作发起计算
- `count()`, `collect()`, `save()`
- 根据依赖关系生成 DAG
- 划分Stage
- Shuffle, Result

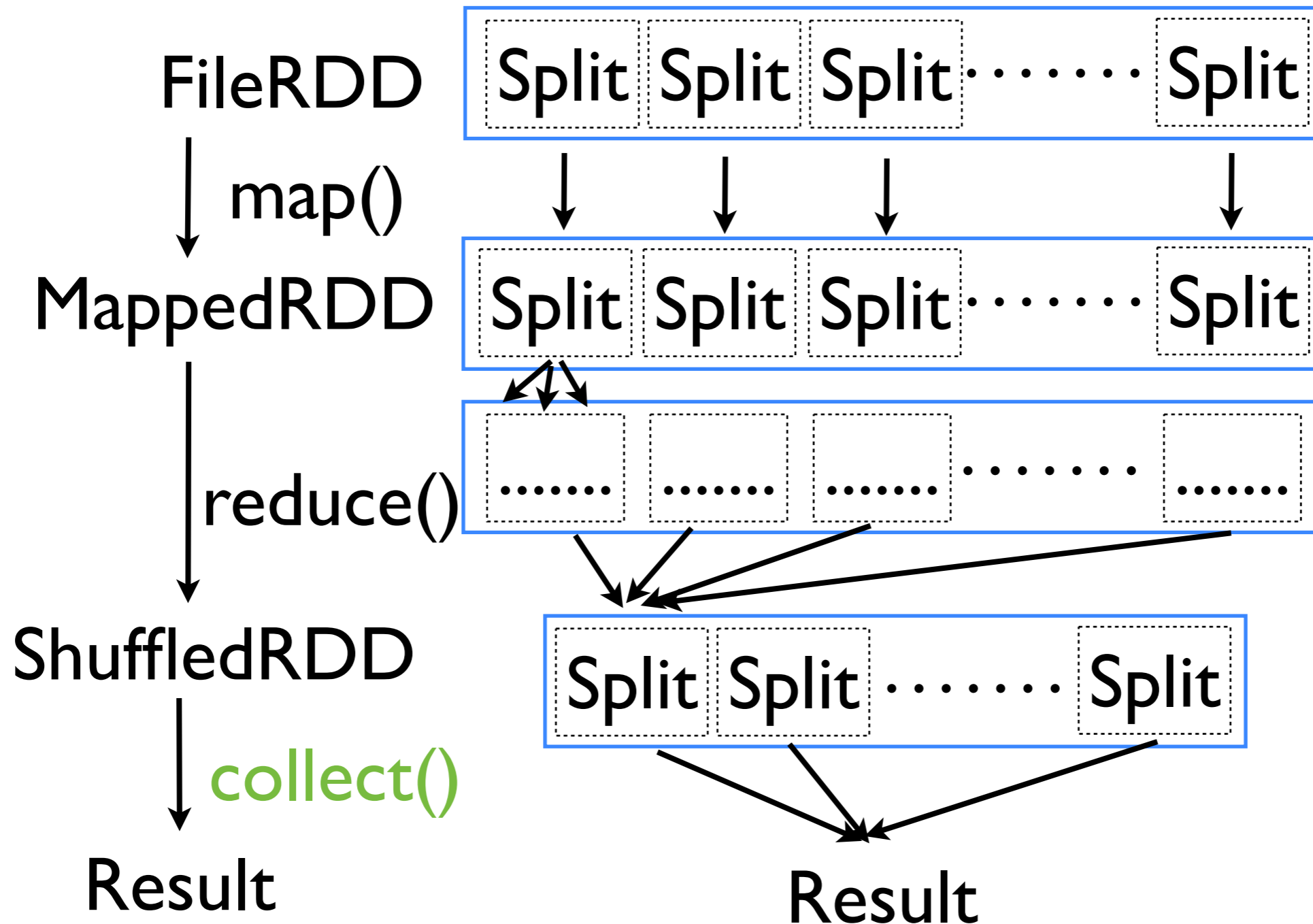


# RDD计算过程

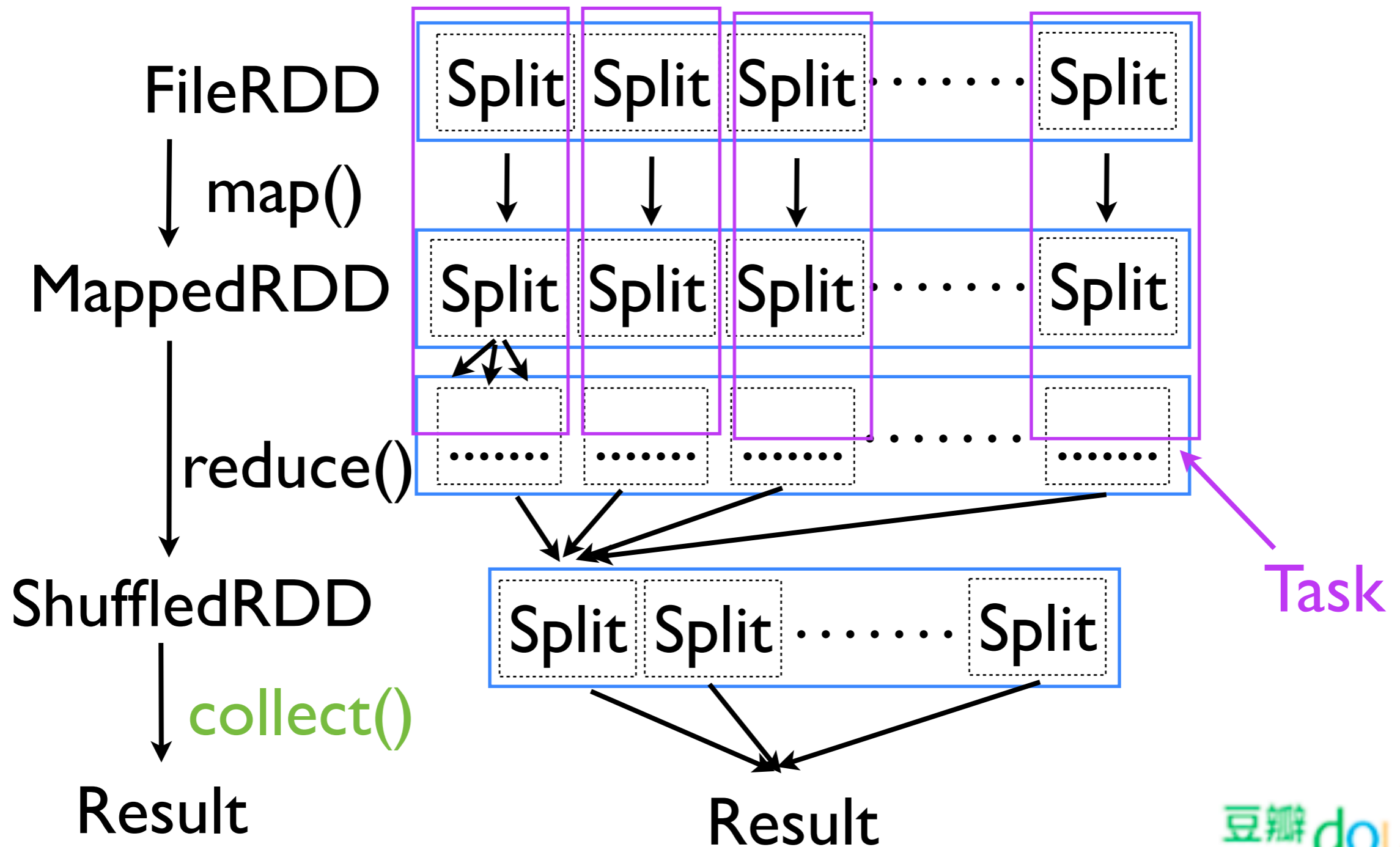
- 由操作发起计算
  - `count()`, `collect()`, `save()`
- 根据依赖关系生成 DAG
- 划分Stage
  - Shuffle, Result



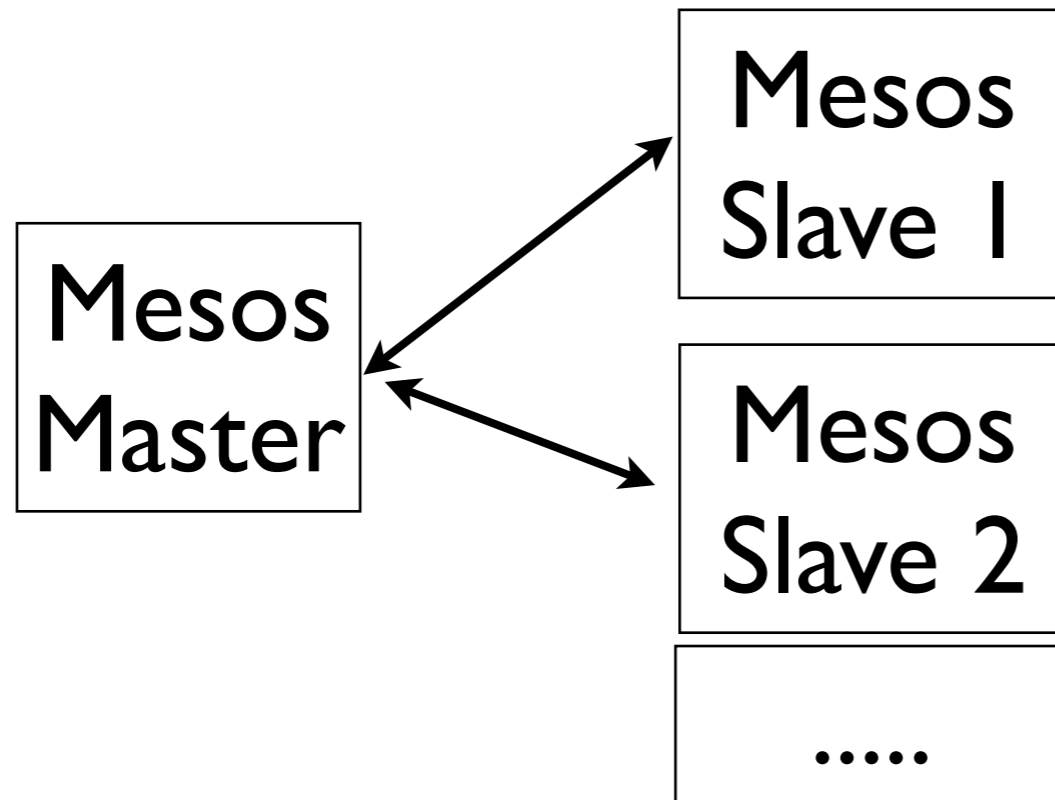
# RDD计算过程



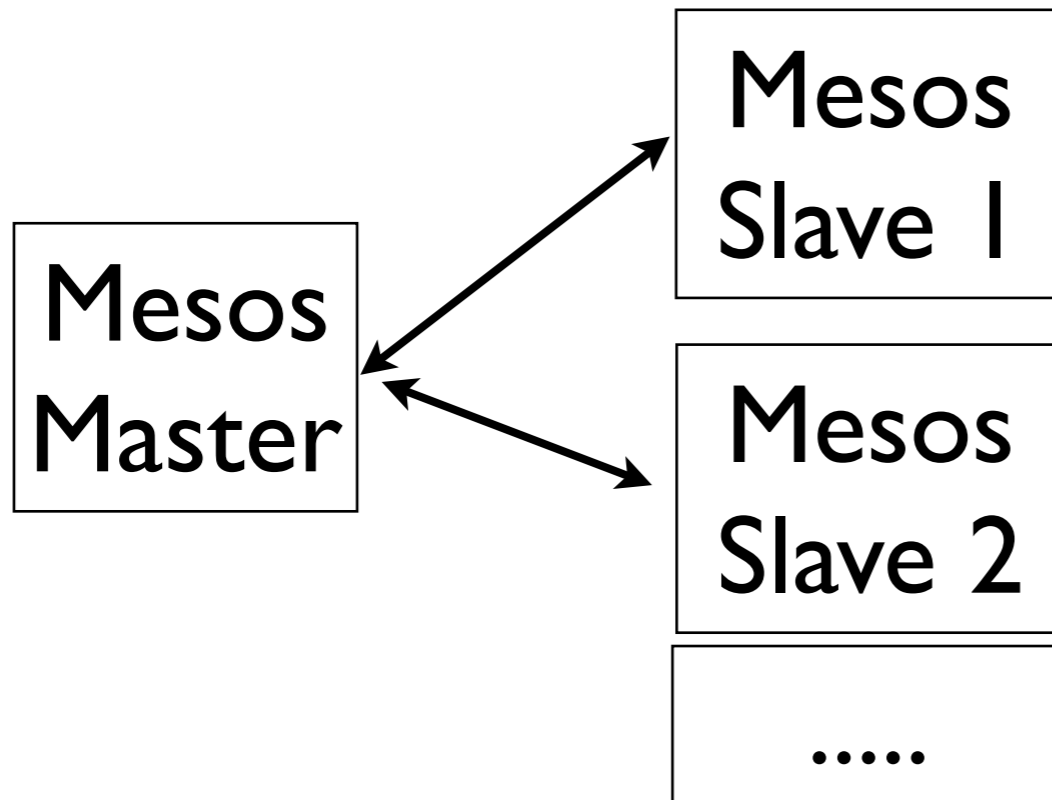
# RDD计算过程



# DPark on Mesos

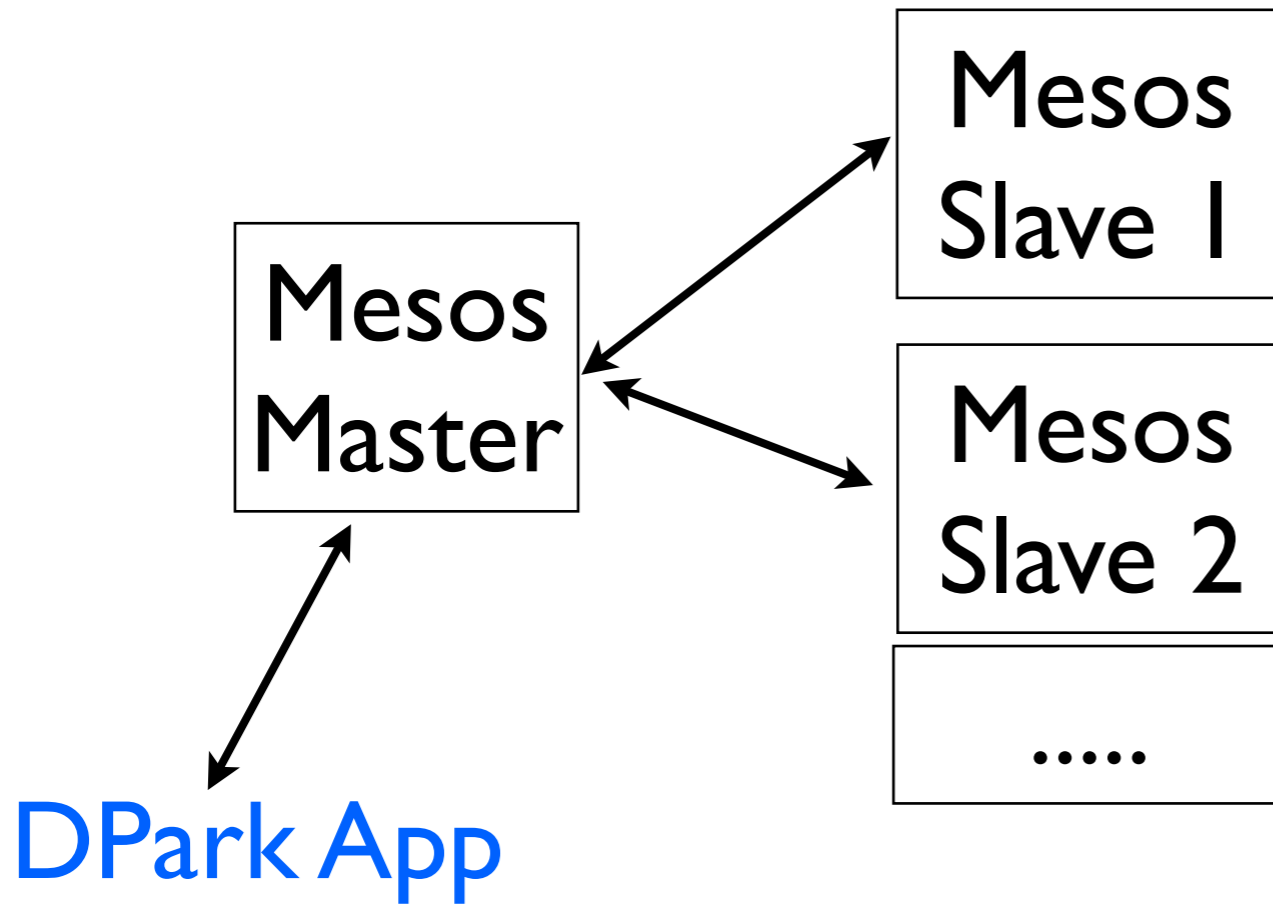


# DPark on Mesos

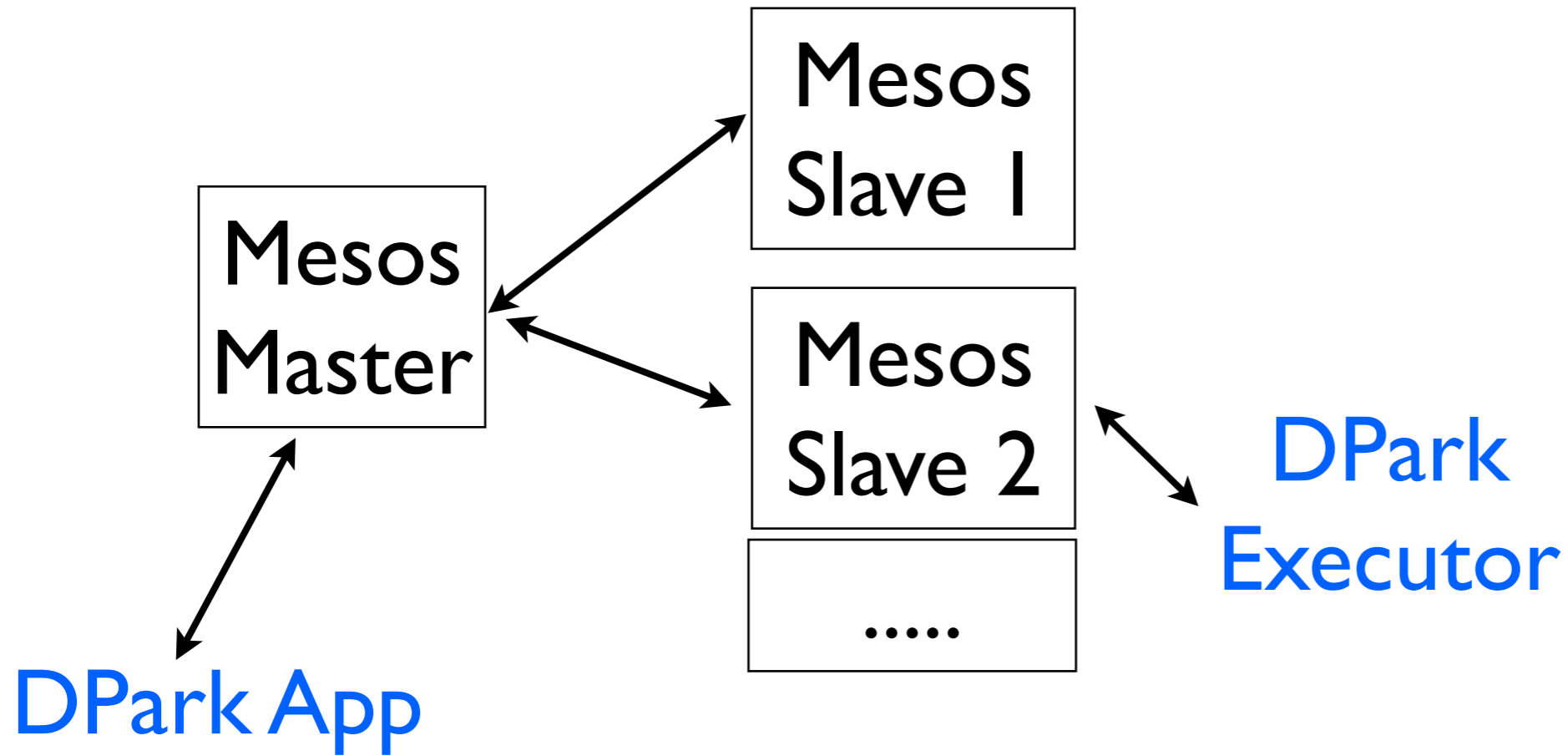


DPark App

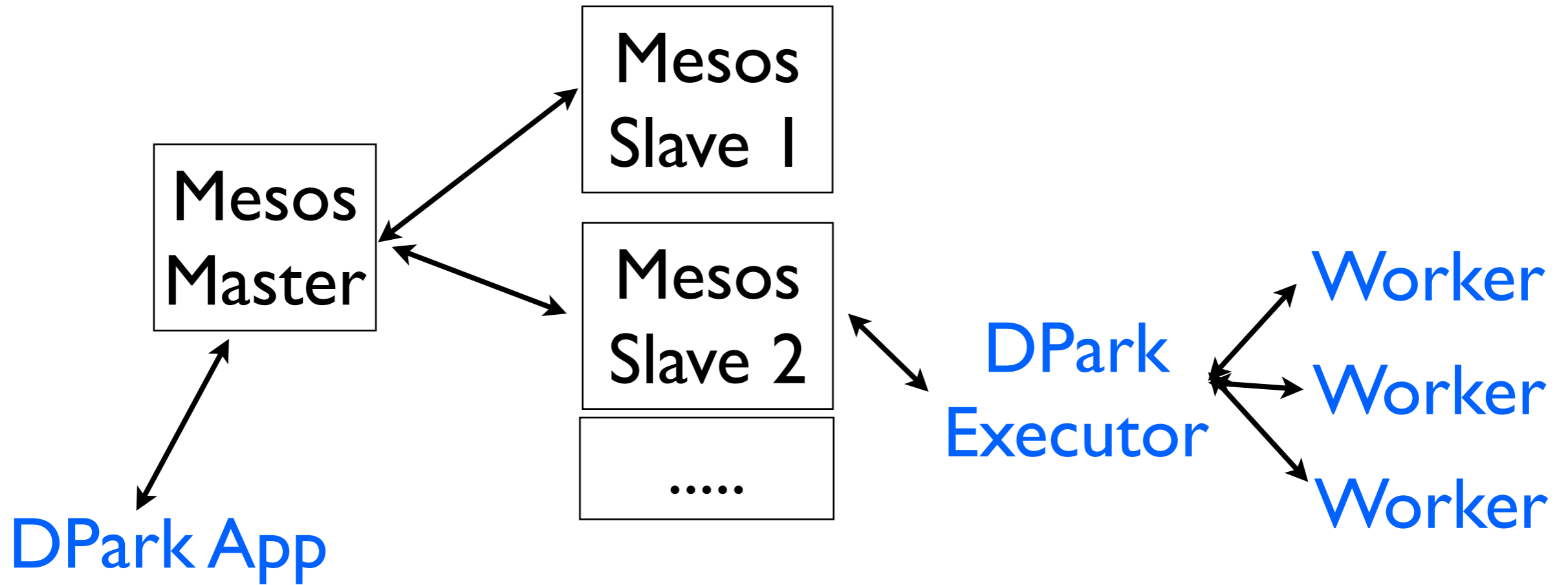
# DPark on Mesos



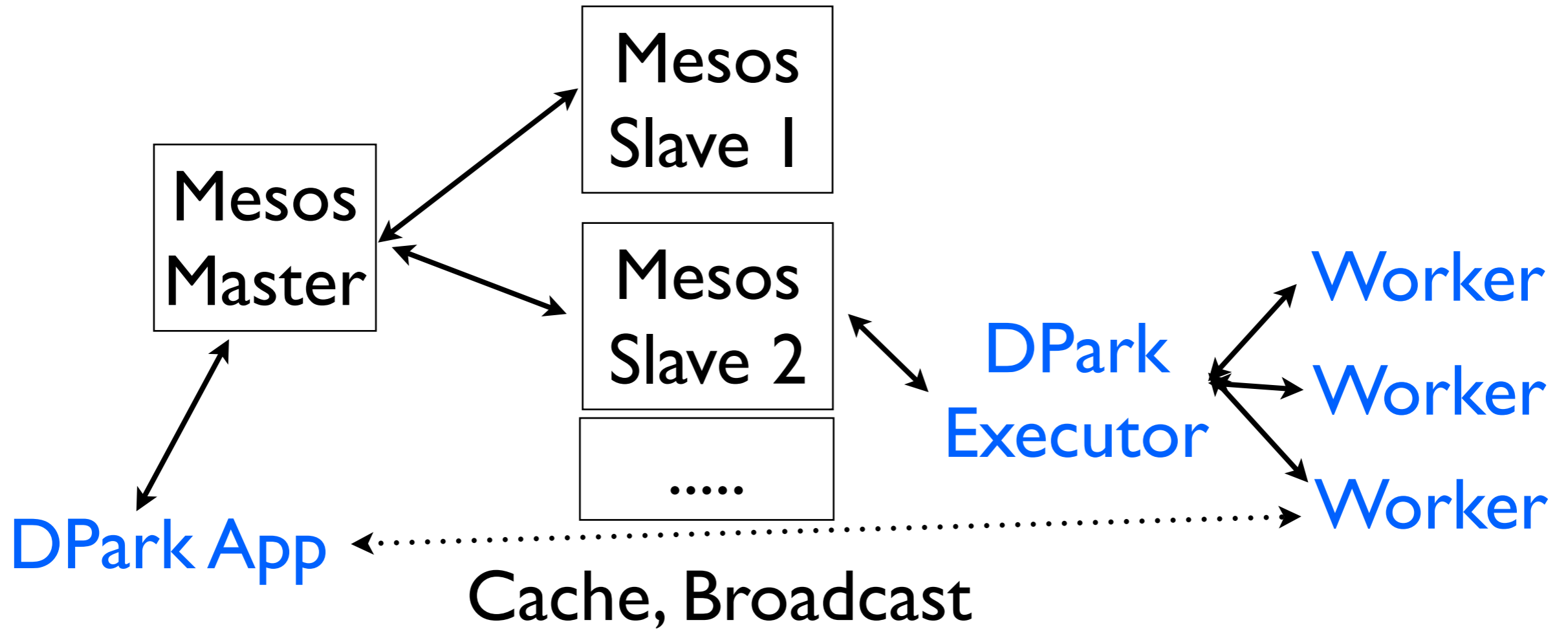
# DPark on Mesos



# DPark on Mesos



# DPark on Mesos



# Word Count

# Word Count

```
files = dpark.textFile('word.txt')
```

# Word Count

```
files = dpark.textFile('word.txt')
```

```
words = files.flatMap(str.split).map(lambda x:(x,1))
```

# Word Count

```
files = dpark.textFile('word.txt')
```

```
words = files.flatMap(str.split).map(lambda x:(x,1))
```

```
cnt = words.reduceByKey(add).collectAsMap()
```

# Word Count

```
files = dpark.textFile('word.txt')
```

```
words = files.flatMap(str.split).map(lambda x:(x,1))
```

```
cnt = words.reduceByKey(add).collectAsMap()
```

```
$ python wordcount.py
```

# Word Count

```
files = dpark.textFile('word.txt')
```

```
words = files.flatMap(str.split).map(lambda x:(x,1))
```

```
cnt = words.reduceByKey(add).collectAsMap()
```

```
$ python wordcount.py
```

```
$ python wordcount.py -m mesos
```

# 交互使用

```
>>> dpark = DparkContext('mesos')
>>> d = dpark.parallelize(range(1000), 10)
>>> print d.reduce(add)

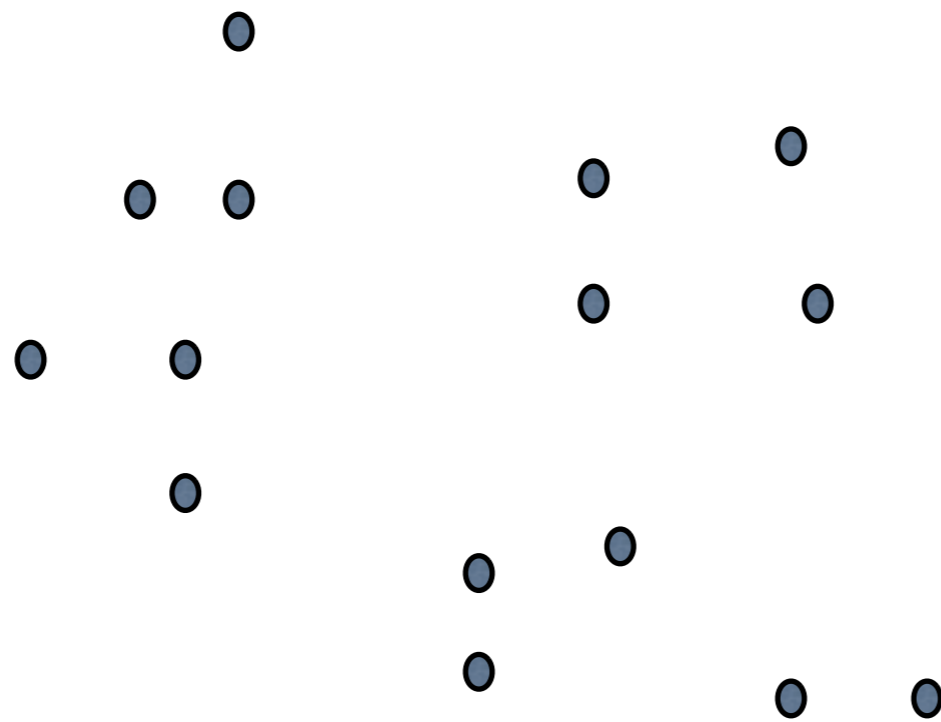
[dpark] 2011-12-02 17:10:47,670 Got a job with 10 tasks
[dpark] 2011-12-02 17:11:04,926 Finished 5 (progress: 1/10)
in 6.64s

....

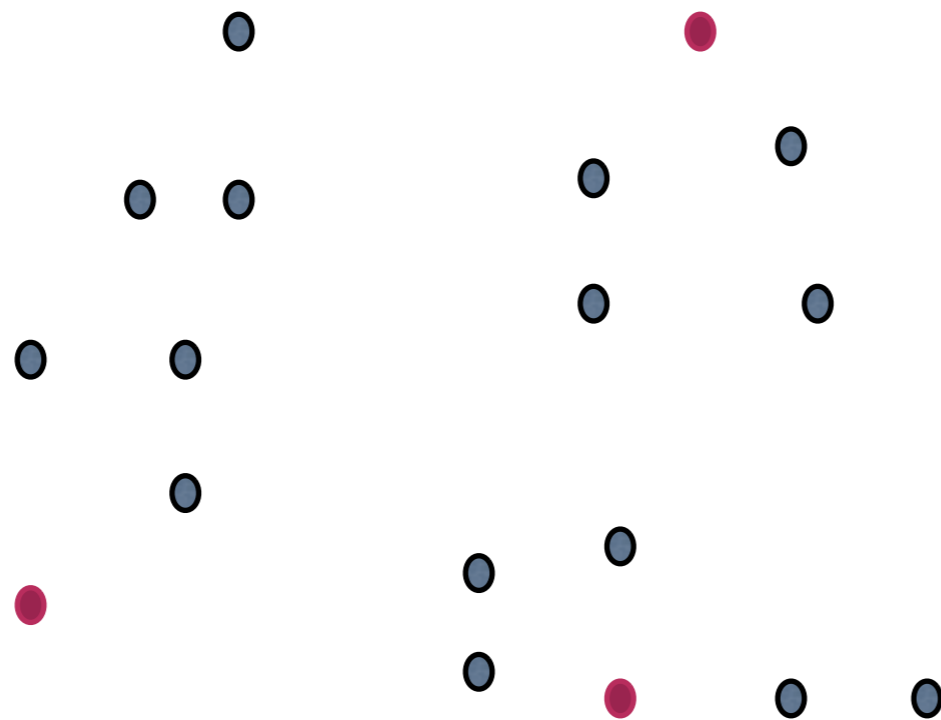
499500
```

# K-Means 算法

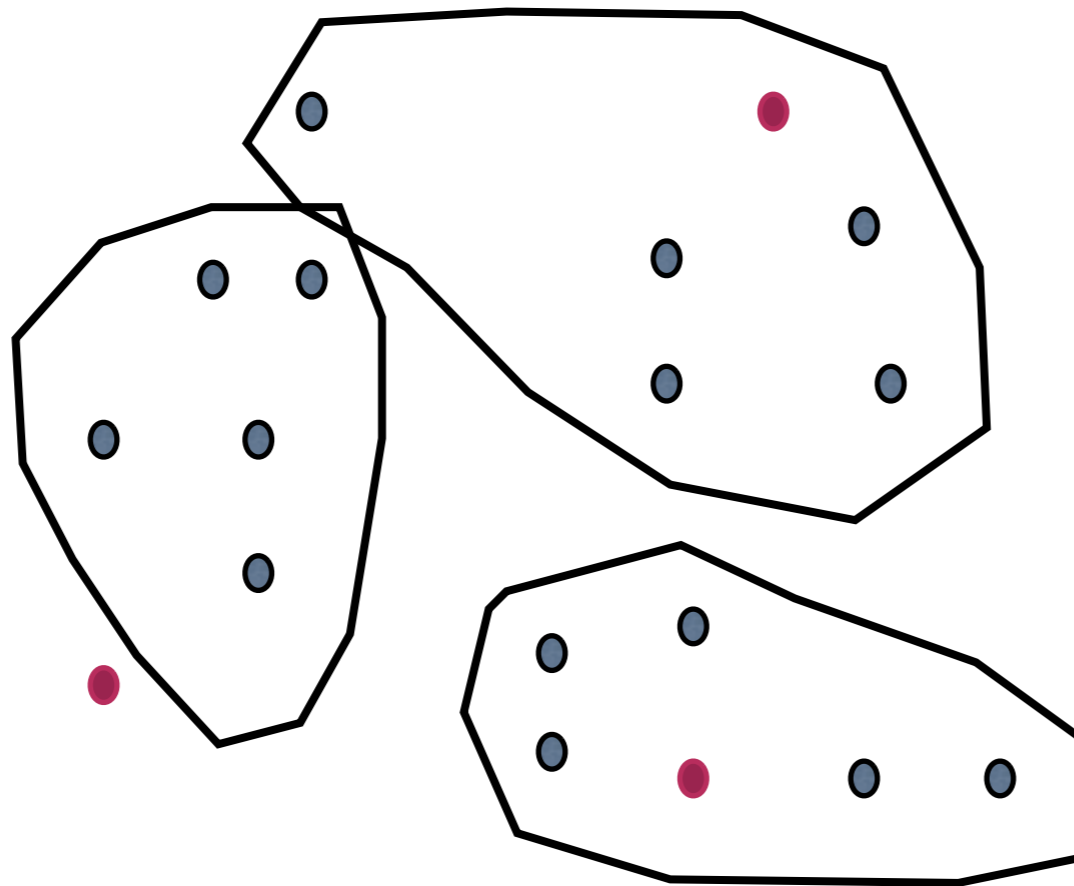
# K-Means 算法



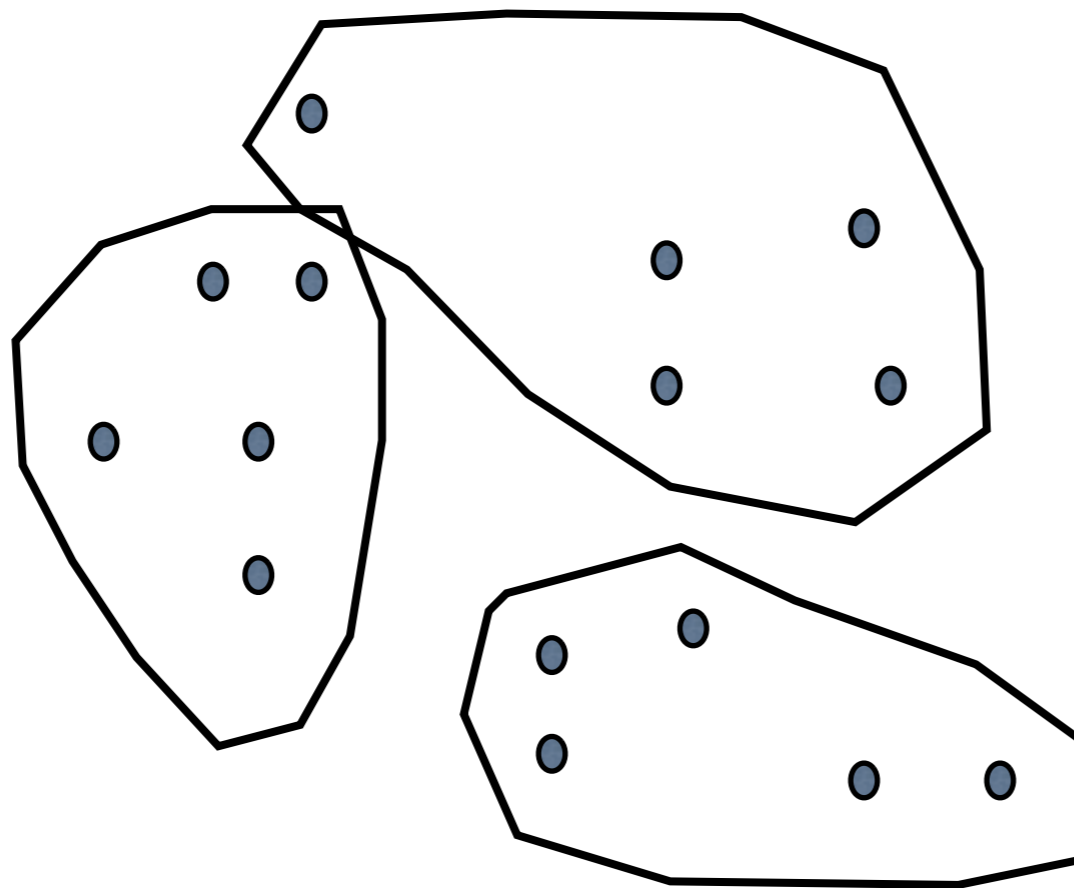
# K-Means 算法



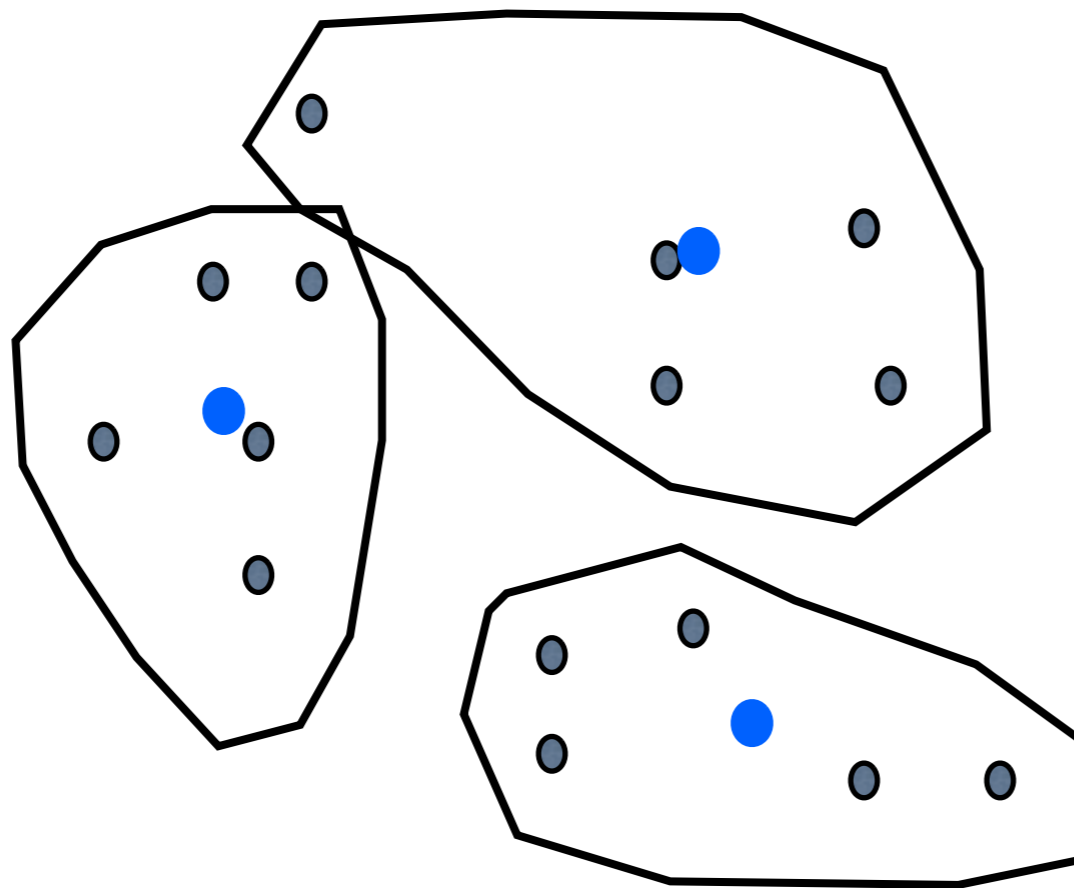
# K-Means 算法



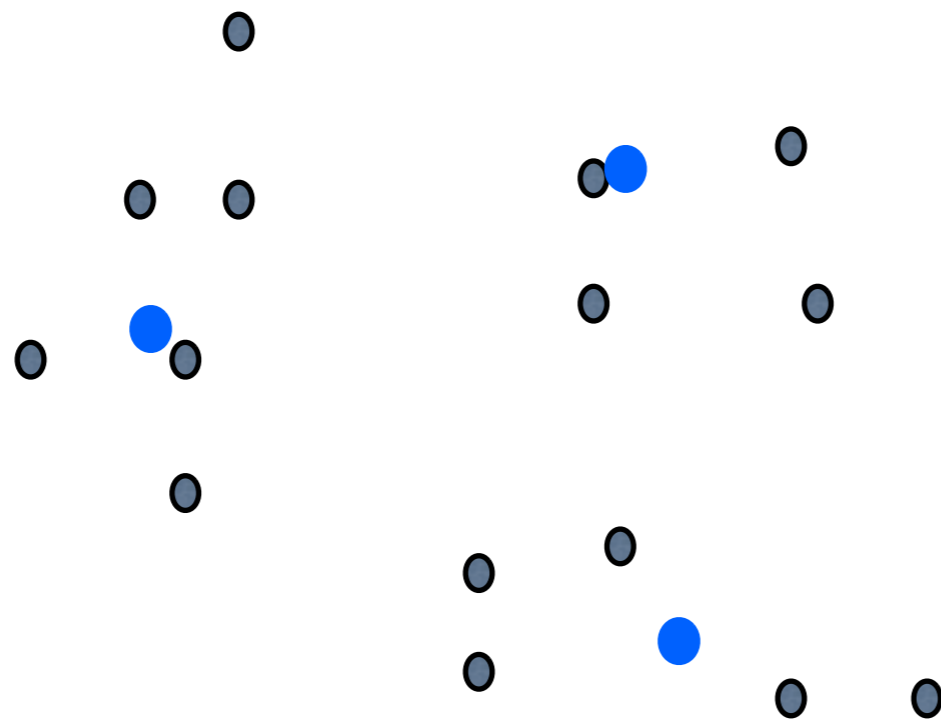
# K-Means 算法



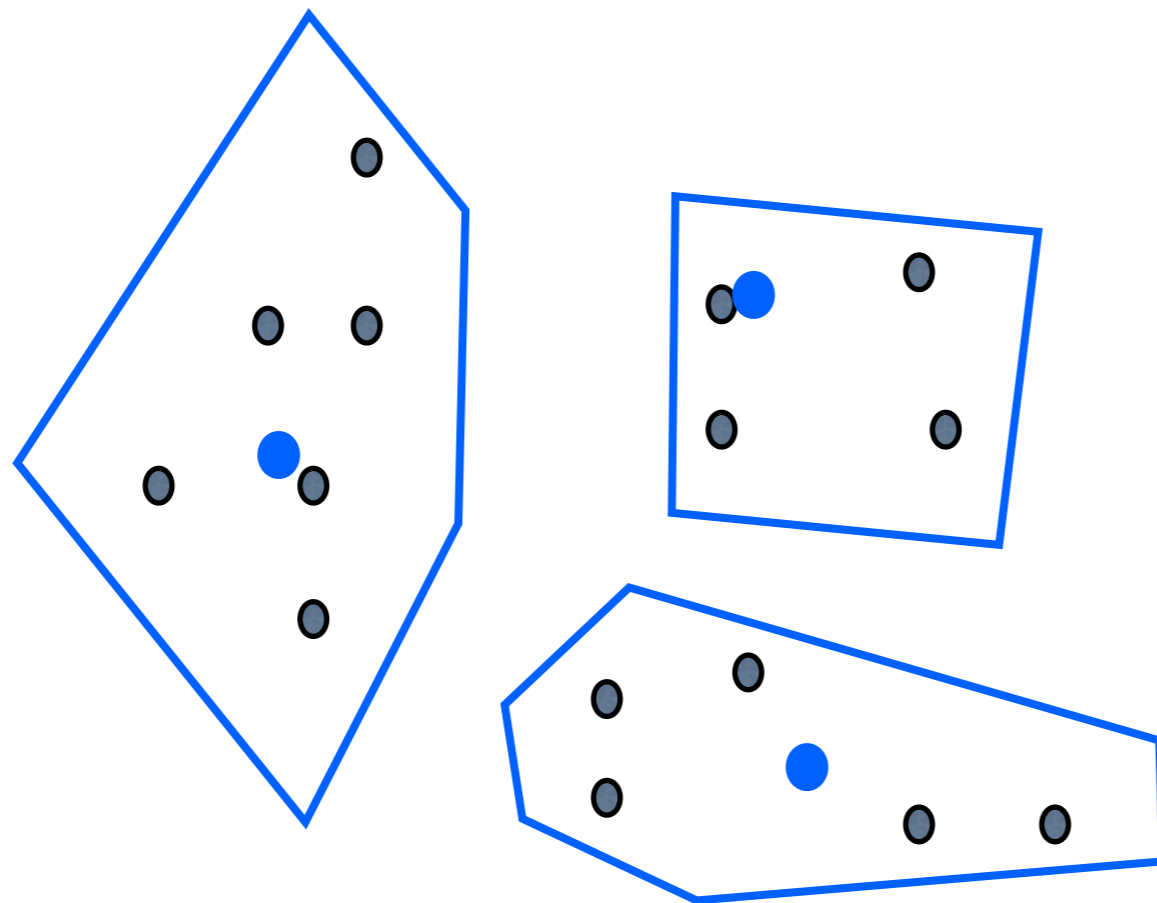
# K-Means 算法



# K-Means 算法



# K-Means 算法



# K-Means

# K-Means

```
points = dpark.textFile('kmeans.txt').map(parseVector).cache()
```

# K-Means

```
points = dpark.textFile('kmeans.txt').map(parseVector).cache()  
centers = [Vector() for i in range(K)]
```

# K-Means

```
points = dpark.textFile('kmeans.txt').map(parseVector).cache()
```

```
centers = [Vector() for i in range(K)]
```

```
for it in range(N):
```

# K-Means

```
points = dpark.textFile('kmeans.txt').map(parseVector).cache()
```

```
centers = [Vector() for i in range(K)]
```

```
for it in range(N):
```

```
    ps = points.map(lambda p:(closestCenter(p, centers), (p, 1)))
```

# K-Means

```
points = dpark.textFile('kmeans.txt').map(parseVector).cache()
```

```
centers = [Vector() for i in range(K)]
```

```
for it in range(N):
```

```
    ps = points.map(lambda p:(closestCenter(p, centers), (p, 1)))
```

```
    ncenters = ps.reduceByKey(mergePoints).collectAsMap()
```

# K-Means

```
points = dpark.textFile('kmeans.txt').map(parseVector).cache()
```

```
centers = [Vector() for i in range(K)]
```

```
for it in range(N):
```

```
    ps = points.map(lambda p:(closestCenter(p, centers), (p, 1)))
```

```
    ncenters = ps.reduceByKey(mergePoints).collectAsMap()
```

```
    centers = ncenters.values()
```

# 相似度计算

```
def cos((a, b)):
```

```
    return cos_sim(a, b)
```

```
ratings = spark.csvFile('r.csv').map(parse).groupByKey()
```

```
blocks = ratings.glom().cache()
```

```
sims = blocks.cartesion(blocks).flatMap(cos)
```

```
final = sims.reduceByKey(lambda x,y:x+y)
```

# 相似度计算

```
def cos((a, b)):
```

```
    return cos_sim(a, b)
```

```
ratings = spark.csvFile('r.csv').map(parse).groupByKey()
```

```
blocks = ratings.glom().cache()
```

```
sims = blocks.cartesion(blocks).flatMap(cos)
```

```
final = sims.reduceByKey(lambda x,y:x+y)
```

可以通过C实现核心算法来提升性能

# DPark 的性能

# DPark 的性能

- 比Hadoop慢

# DPark 的性能

- 比Hadoop慢
- 基本够用

# DPark 的性能

- 比Hadoop慢
- 基本够用
- 优化方向

# DPark 的性能

- 比Hadoop慢
- 基本够用
- 优化方向
  - C 扩展

# DPark 的性能

- 比Hadoop慢
- 基本够用
- 优化方向
  - C 扩展
  - PyPy

# 总结

# 总结

- 简单，灵活，友好
- 更好地支持迭代计算
- 在中小规模下可以取代Hadoop

# DPark 的未来

# DPark 的未来

- 开源

# DPark 的未来

- 开源
- 完善文档

# DPark 的未来

- 开源
- 完善文档
- 进一步提升性能和可扩展性

**Thank you!**

**Question ???**