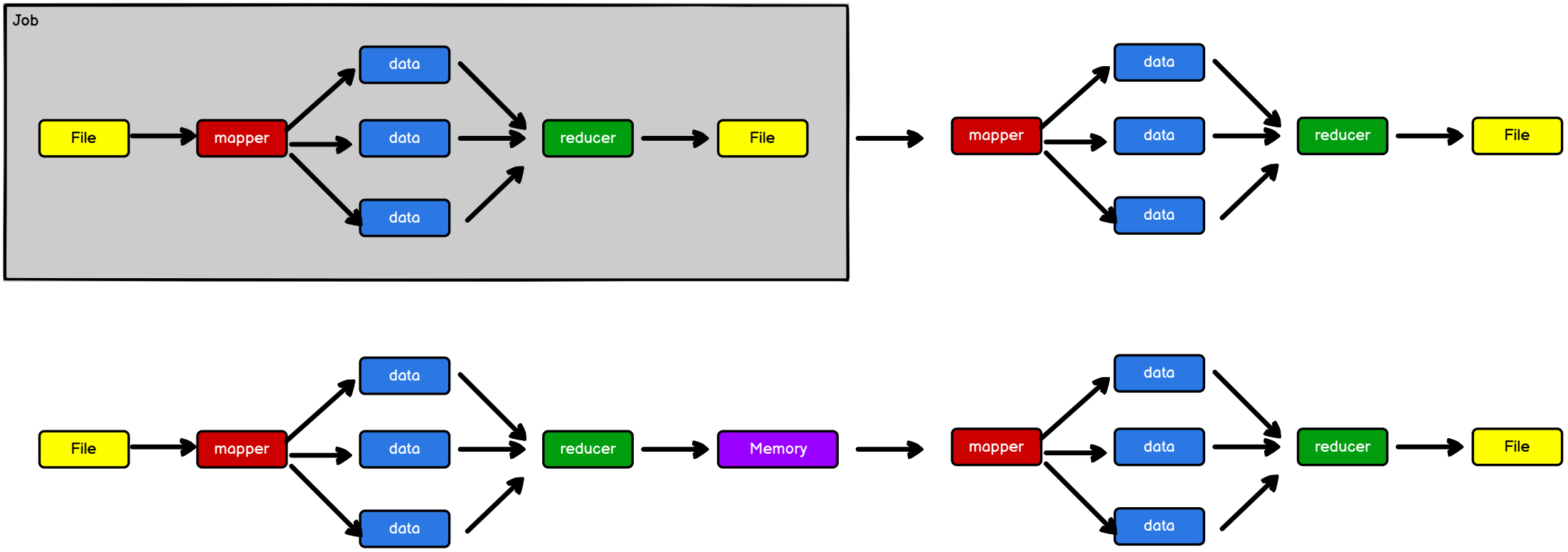
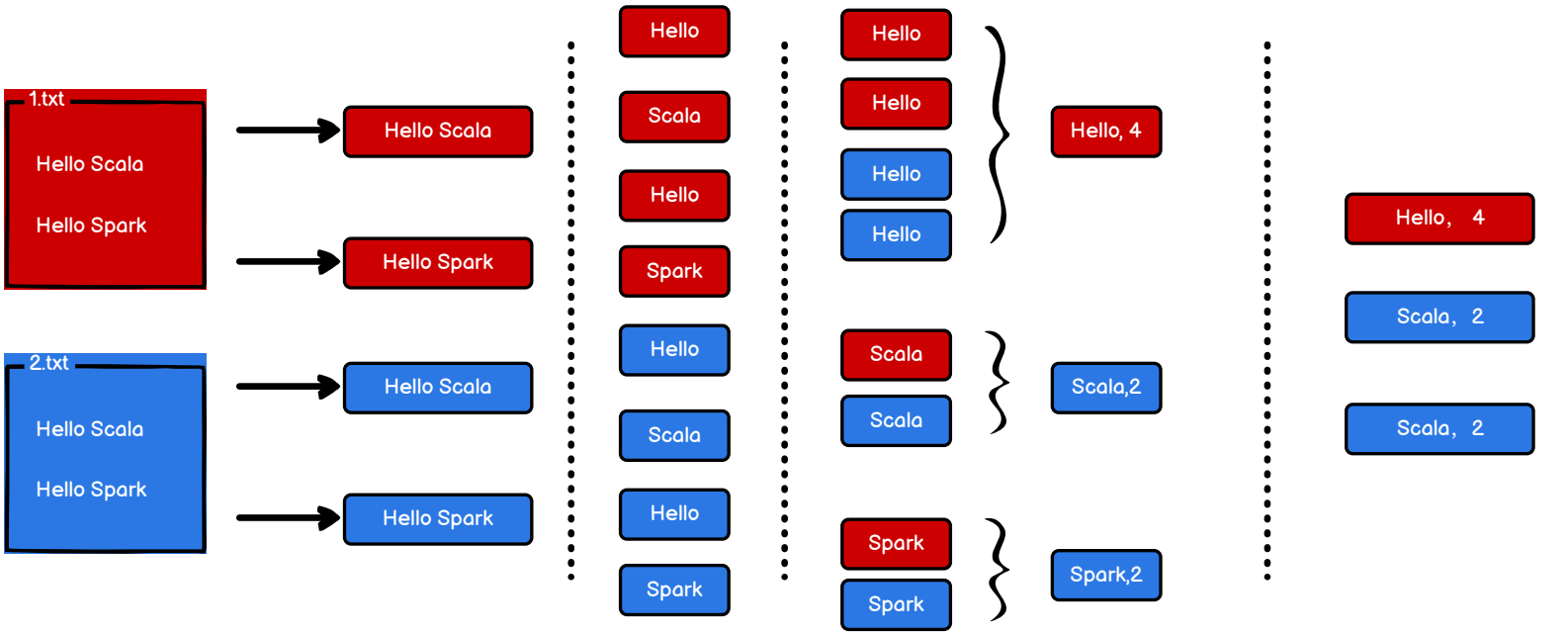
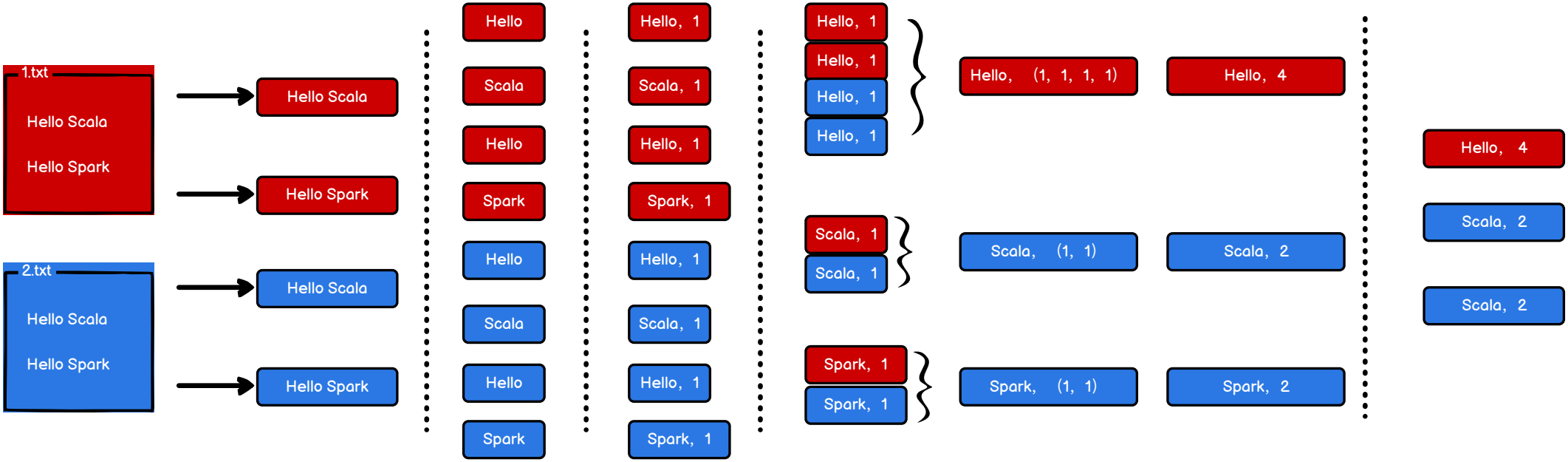
一次性数据计算
框架在处理数据的时候，会从存储设备中读取数据，进行逻辑操作，然后将处理的结果重新存储到介质中，

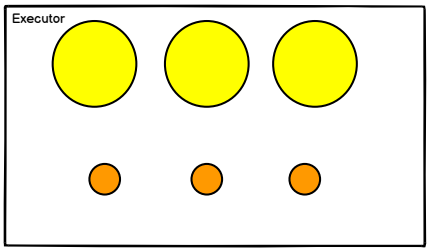


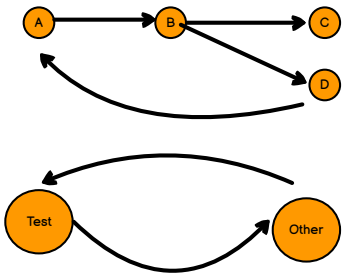
缺什么，补什么

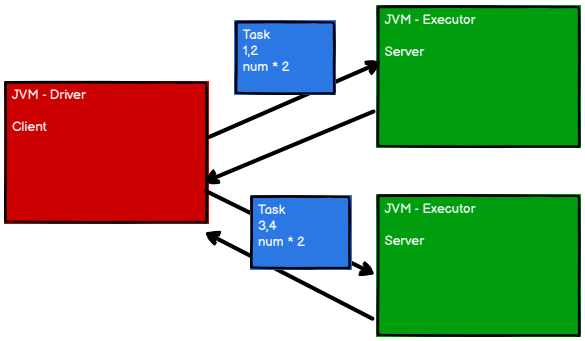


reduceByKey









RDD方法

转换： 功能的补充和封装，将旧的RDD包装成新的RDD

flatMap, map

行动： 触发任务的调度和作业的执行

collect

RDD方法 => RDD算子

认知心理学认为解决问题其实将问题的状态进行改变：

问题（初始） => 操作（算子） => 问题（审核中） => 操作（算子） =>

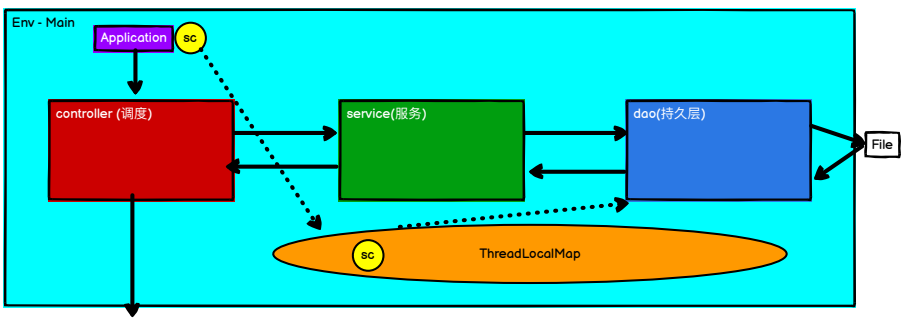
MVC : Model View Controller

Java => Net

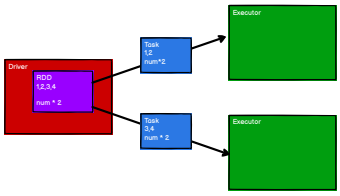
Java Web : Servlet (Java + HTML + CSS + JS)

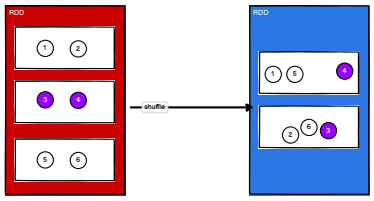
Java Web : JSP

三层架构： controller (控制层)， service（服务层），dao(持久层)



ThreadLocal可以对线程的内存进行控制，存储数据，共享数据

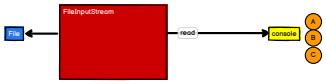


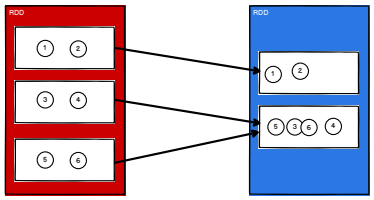


IO : Input & Output

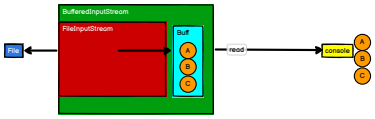
字节流 & 字符流

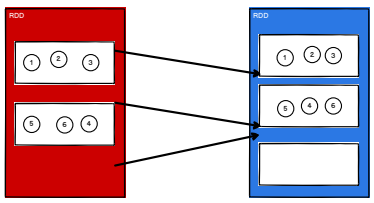
```
InputStream in = new FileInputStream("path")
int i = -1
while ( (i = in.read()) != -1 ) {
   println(i);
}
```





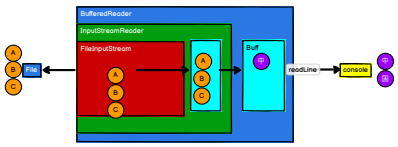

```
InputStream in = new BufferedInputStream(new FileInputStream("path"))
int i = -1
while ( (i = in.read()) != -1 ) {
   println(i);
}
```

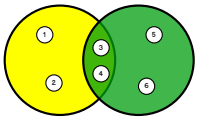


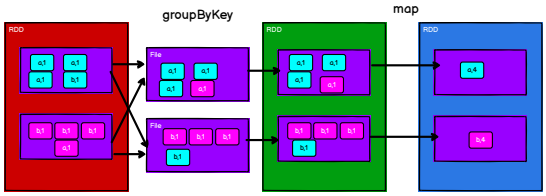



```
Reader in = new BufferedReader(
   new InputStreamReader(
         new FileInputStream("path"),
         "UTF-8"
   )
)
String s = null
while ( (s = in.readLine()) != null ) {
   println(i);
}
```

IO操作体现了装饰者设计模式

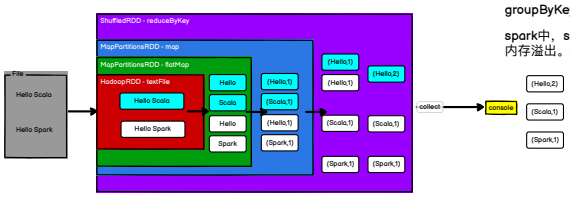






groupByKey会导致数据打乱重组，存在shuffle操作

spark中，shuffle操作必须落盘处理，不能在内存中数据等待，会导致内存溢出。shuffle操作的性能非常低

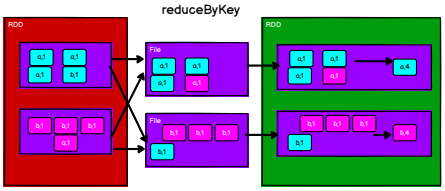


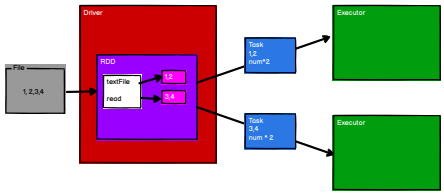
RDD的数据处理方式类似于IO流，也有装饰者设计模式

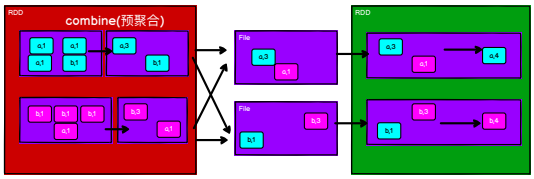
RDD的数据只有在调用collect方法时，才会真正执行业务逻辑操作。之前的封装全部都是功能的扩展

RDD是不保存数据的，但是IO可以临时保存一部分数据

reduceByKey







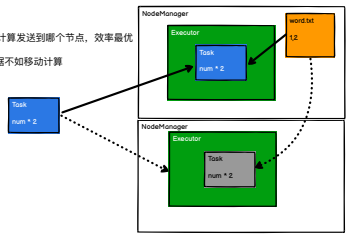
reduceByKey支持分区内预聚合功能，可以有效减少shuffle时落盘的数据量，提升shuffle的性能。

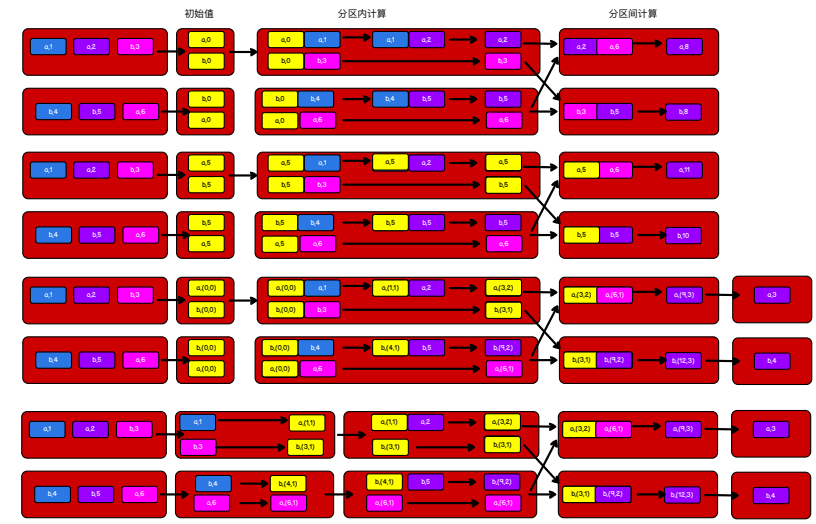
分区内 分区间

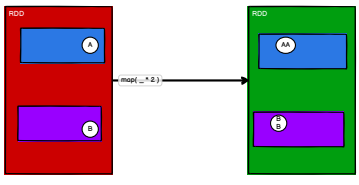
reduceByKey分区内和分区间计算规则是相同的。

首选位置：判断计算发送到哪个节点，效率最优

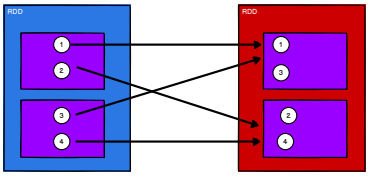
移动数据不如移动计算

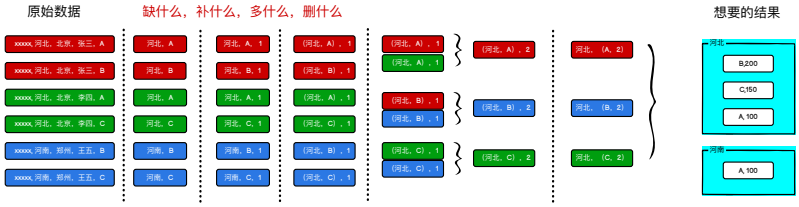


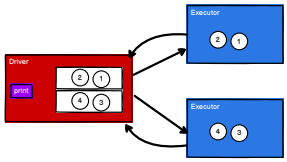


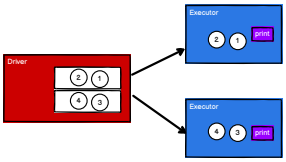


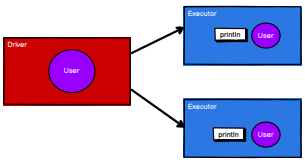
groupBy会将数据打乱（打散），重新组合，这个操作我们称之为shuffle

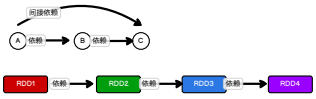












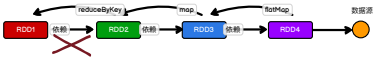
相邻的两个RDD的关系称之为依赖关系

val rdd1 = rdd.map(_* 2)

新的RDD依赖于旧的RDD

多个连续的RDD的依赖关系，称之为血缘关系

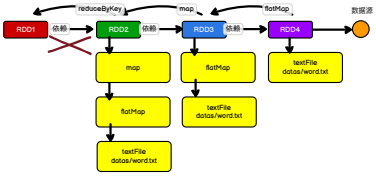
每个RDD会保存血缘关系

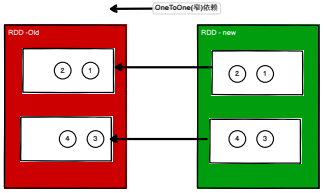


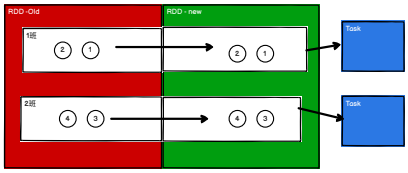
RDD不会保存数据的

RDD为了提供容错性，需要将RDD间的关系保存下来

一旦出现错误，可以根据血缘关系将数据源重新读取进行计算

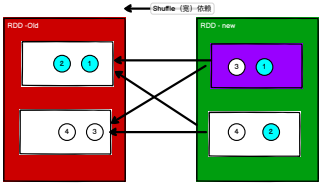


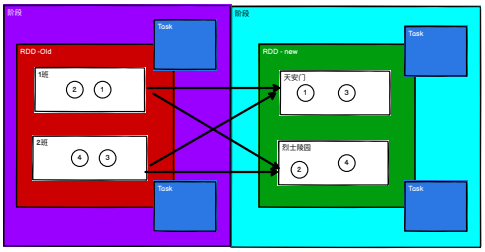




新的RDD的一个分区的数据依赖于旧的RDD一个分区的数据。

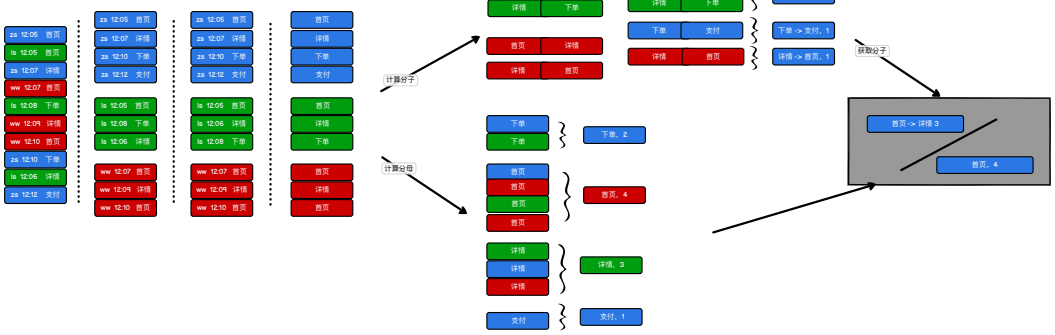
这个依赖称之为OneToOne依赖



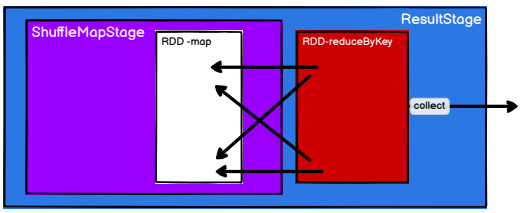


新的RDD的一个分区的数据依赖于旧的RDD多个分区的数据。

这个依赖称之为Shuffle依赖



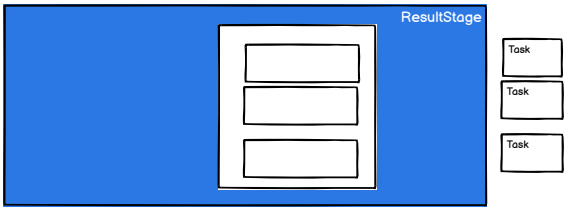
阶段划分



当RDD中存在shuffle依赖时，阶段会自动增加一个

阶段的数量 = shuffle依赖的数量 + 1

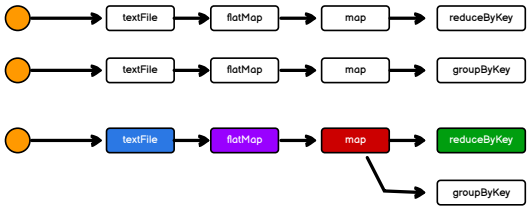
ResultStage只有一个，最后需要执行的阶段



任务的数量 = 当前阶段中最后一个RDD的分区数量

ShuffleMapStage => ShuffleMapTask

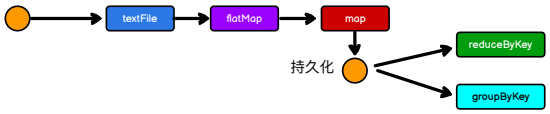
ResultStage => ResultTask



RDD中不存储数据

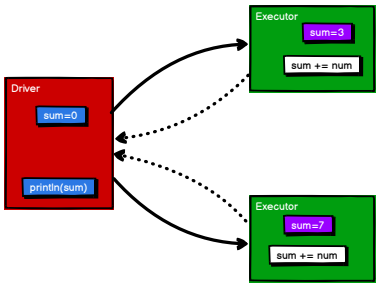
如果一个RDD需要重复使用，那么需要从头再次执行来获取数据

RDD对象可以重用的，但是数据无法重用

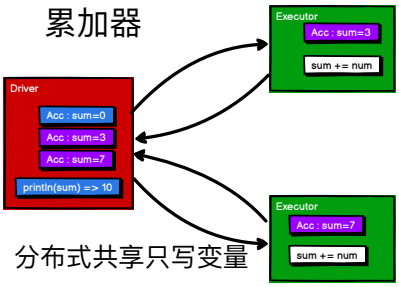


RDD对象的持久化操作不一定是为了重用

在数据执行较长，或数据比较重要的场合也可以采用持久化操作

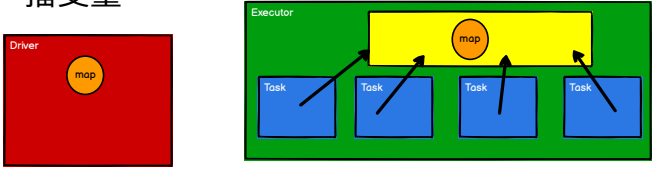


累加器



分布式共享只写变量

广播变量



闭包数据，都是以Task为单位发送的，每个任务中包含闭包数据

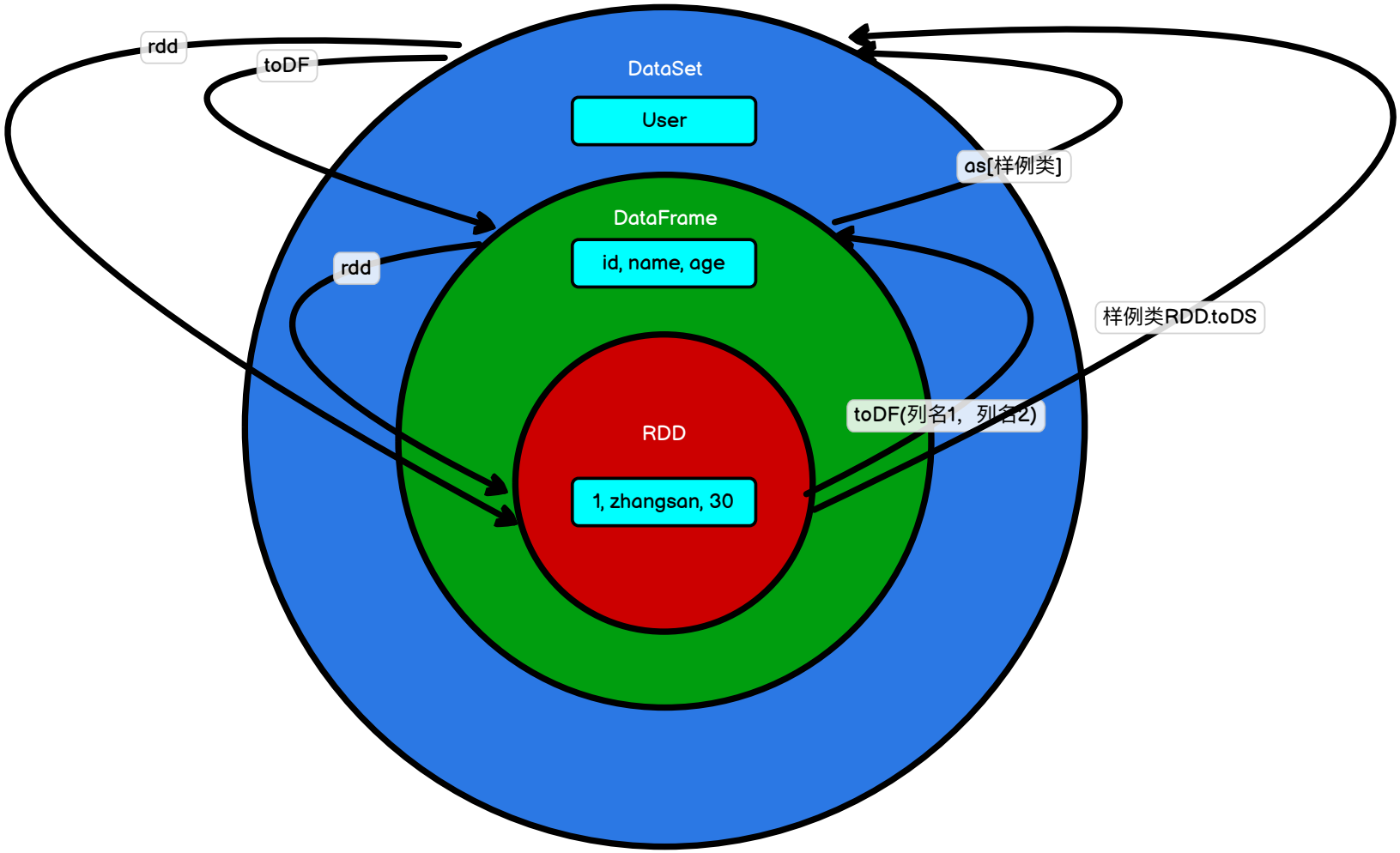
这样可能会导致，一个Executor中含有大量重复的数据，并且占用大量的内存

Executor其实就一个JVM，所以在启动时，会自动分配内存

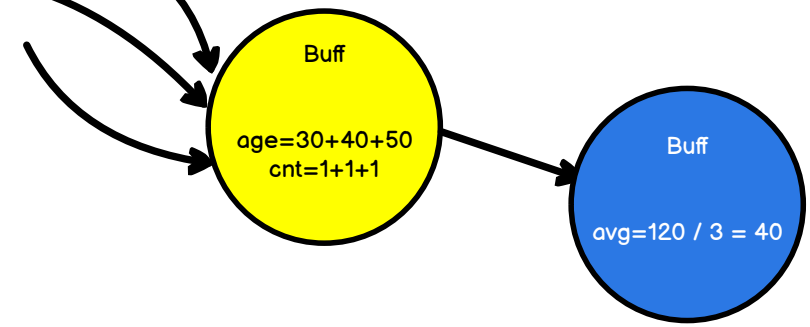
完全可以将任务中的闭包数据放置在Executor的内存中，达到共享的目的

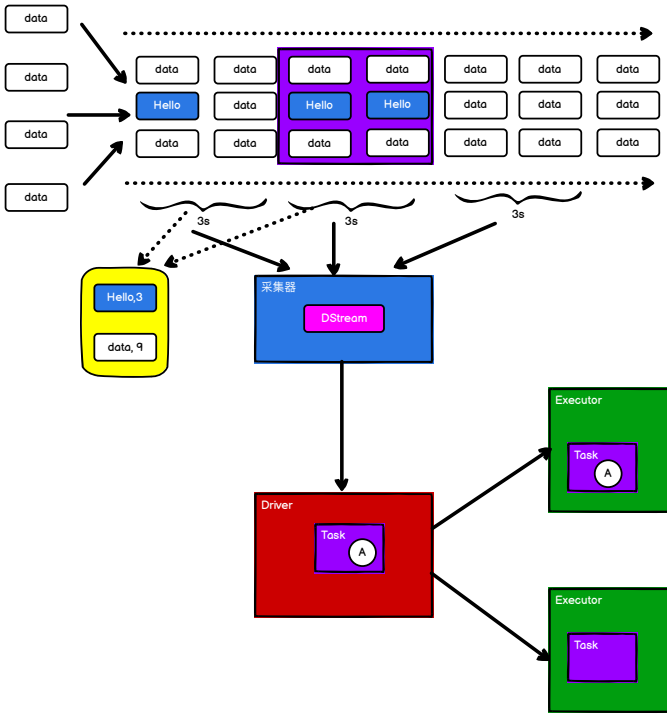
Spark中的广播变量就可以将闭包的数据保存到Executor的内存中

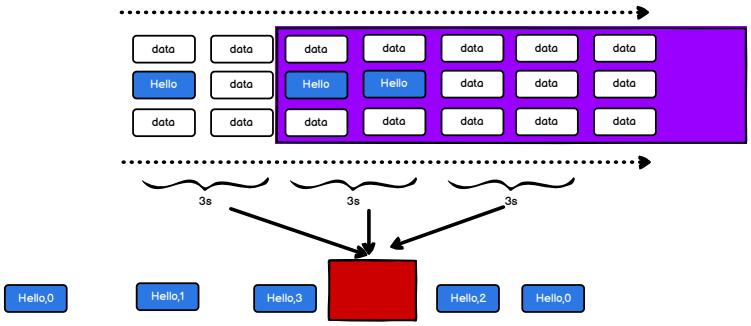
Spark中的广播变量不能够更改： 分布式共享只读变量

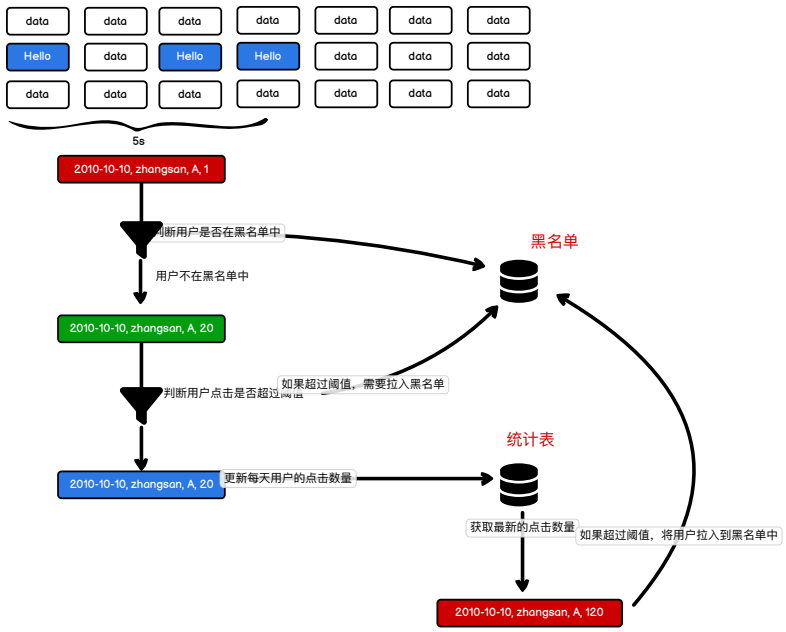


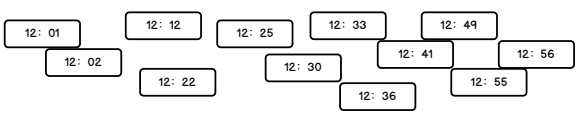
id	name	age
1	zhangsan	30
2	lisi	40
3	wangwu	50

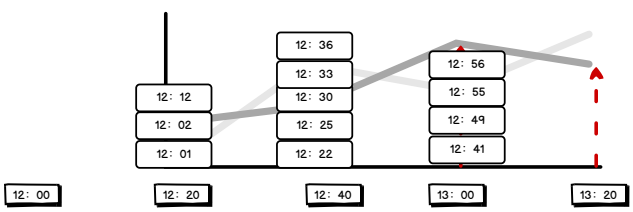



data
Hello
3s
采集器
DStream
Hello,3
data, 9
Executor
Task
A
Driver


data
Hello
3s
Hello,0
Hello,1
Hello,3
Hello,2
Hello,0


data
Hello
5s
2010-10-10, zhangsan, A, 1
判断用户是否在黑名单中
黑名单
用户不在黑名单中
2010-10-10, zhangsan, A, 20
如果超过阈值，需要拉入黑名单
统计表
2010-10-10, zhangsan, A, 20
更新每天用户的点击数量
获取最新的点击数量
如果超过阈值，将用户拉入到黑名单中
2010-10-10, zhangsan, A, 120


12：01
12：02
12：12
12：22
12：25
12：30
12：33
12：36
12：41
12：49
12：55
12：56


12：12
12：02
12：01
12：36
12：33
12：30
12：25
12：22
12：56
12：55
12：49
12：41
12：00
12：20
12：40
13：00
13：20