

COMP9313

Week 3

Lecturer:

Elijah

英语学习 podcasts 推荐



Aussie English

Learn Fair Dinkum Australian English!

<https://aussieenglish.com.au>

G'day! I'm Pete

My aim is to teach you Australian English!



Aussie English

71.2K subscribers

任意 Podcast 软件搜索 Aussie English

免费！

- 
- PySpark 环境搭建
 - Spark 基础入门
 - HDFS 拓展内容



- PySpark 环境搭建

开始之前弄清楚几个概念：

1. 环境变量
2. Python 运行环境
3. Anaconda

环境变量

- 环境变量 PATH 是告诉系统当我们在命令行输入一个命令，例如 python, scala 的时候，如果当前目录找不到这个程序，就去 PATH 里面找
- `echo $PATH` 查看当前系统环境变量
- order matters !
- Mac 的 PATH 的修改来源有三个：
 1. `~/.bash_profile` 它是个 bash 文件(.zshrc)，修改它需要使用 `export PATH=$PATH:/path1/path2/` 的语法，修改完毕之后要 `source $HOME/.bash_profile` 执行这个文件
 2. `/etc/paths` 文件，可以直接加入路径
 3. (推荐) `/etc/paths.d` 文件夹在里面可以，创建小的文件例如 `vim scala` 然后直接加入路径

改了之后，重启 terminal 即可

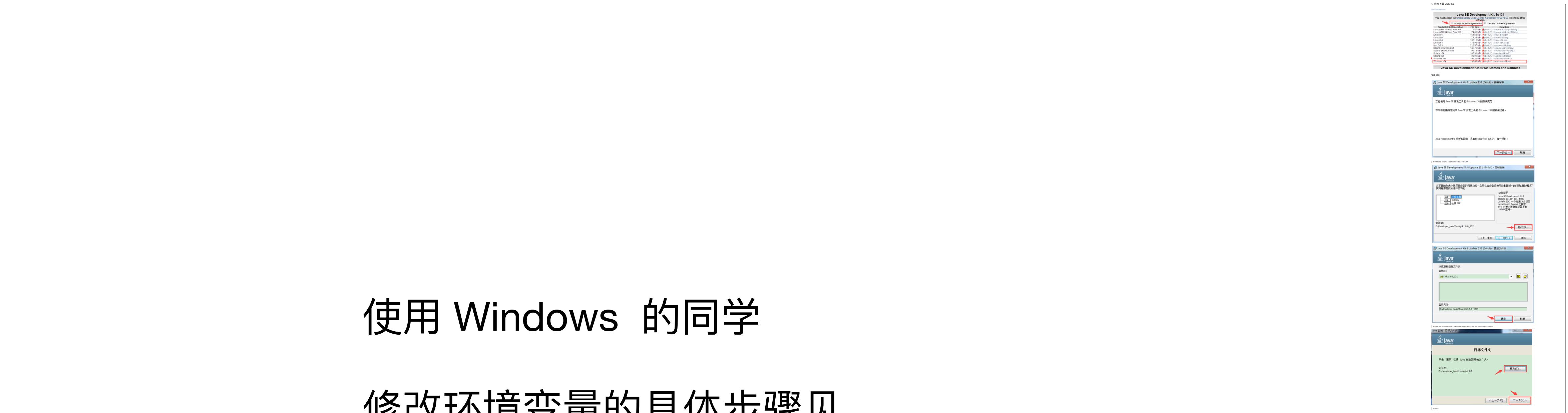
给使用 Mac 的同学的贴士：

注意自己的 Terminal 用的是什么内核

新版本 MacOS 推荐时用 zsh

老版本是 Bash

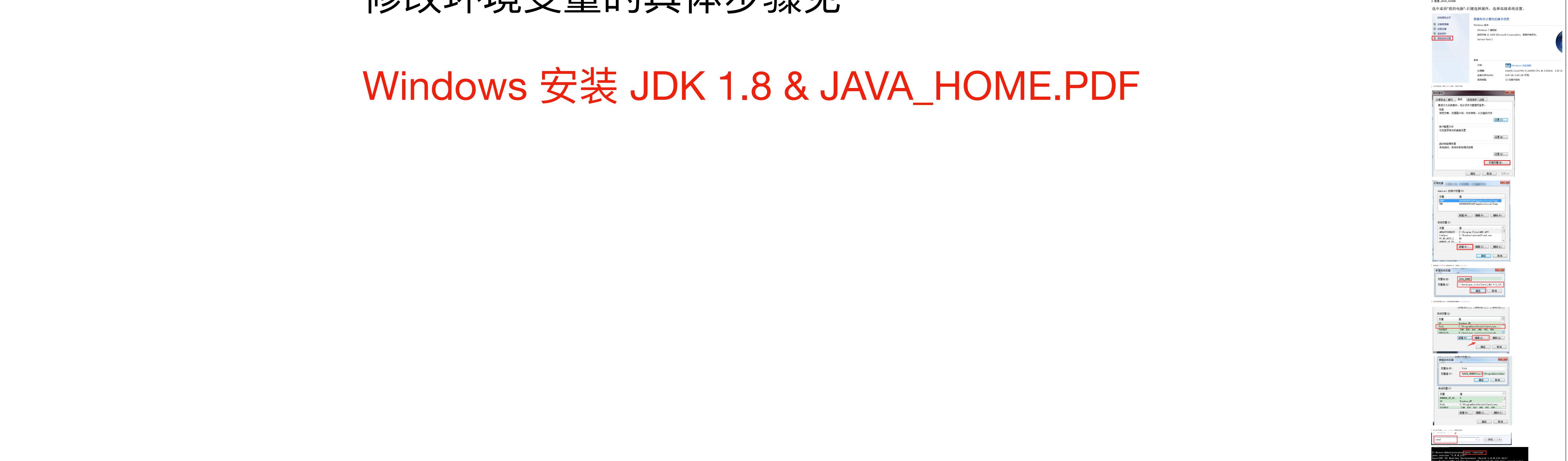
Terminal App 推荐：Hyper



使用 Windows 的同学

修改环境变量的具体步骤见

Windows 安装 JDK 1.8 & JAVA_HOME.PDF



关于 Mac 的 Python 版本和环境

- 由于 Mac **自带** 旧版本 python, 在命令行打 `which python` -> `/usr/bin/python` 其实这个也是个 symbolic link, 使用 `ls -al` 可以看到它 links to `system/Frameworks/` 但是这个版本不适合开发, 于是我们用 Homebrew 来安装
- **HomeBrew** 的 python 安装在 `/usr/local/bin/python3` 其实这个也是个 symbolic link, links to `/usr/local/Cellar/python/3.7.7/bin/python3`
- 在 PyCharm interpreter 里面要使用 `/usr/local/bin/python3` 这个路径下的 python
- 注意: 一旦使用使用 Anacoda 就不用安装 python 了, Anacoda 里面自带 python, 也不需要 brew 了
- 注意如果你在命令行使用 python3 需要注意它的路径是哪个 `which -a python3`

Anaconda

- 安装了 Anaconda 就是安装了 python
- Anaconda 是为 python 的运行提供包的管理的环境的软件，他比 pip3 好在，他可以下载 non-python 的包
- Anaconda 可以手动创建不用同虚拟环境，在 PyCharm 或者 Jupyter 上也可以手动选择使用 Anacoda 的环境

Pyspark 环境搭建 Demo

最后的图文步骤见

配置 Pyspark & Anaconda.PDF

休息

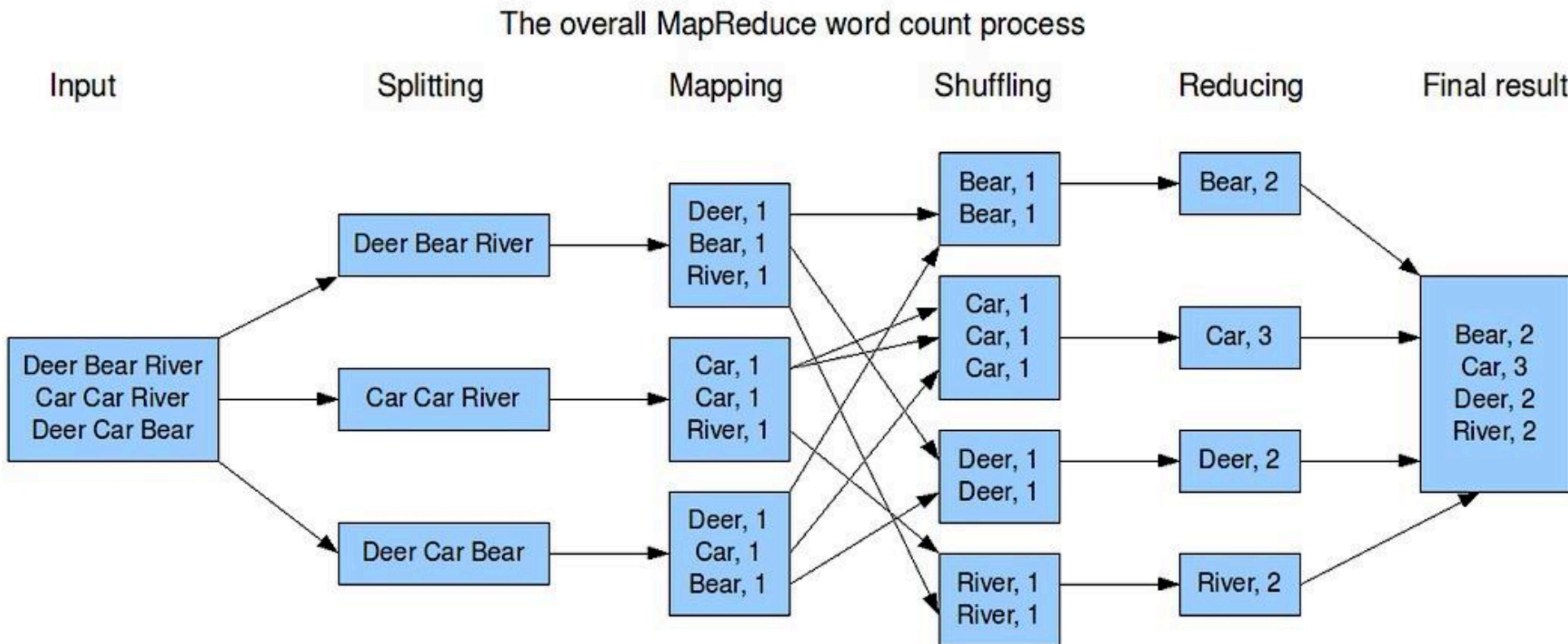


- **Spark 基础入门**

知识点：MapReduce 的概述

1. Hadoop 体系里面用来计算的部分（回忆Hadoop三大组成）
2. 需要程序员手写 MapReduce 程序来实现业务，一般是 Java
3. 分为 Map Reduce 两个阶段一分一合
4. Map 用来做映射，Reduce 用来做合并
5. 中间有 Shuffle 的过程，在框架中已经帮你写好了
6. 看经典案例 WordCount...

A Simple MapReduce Example

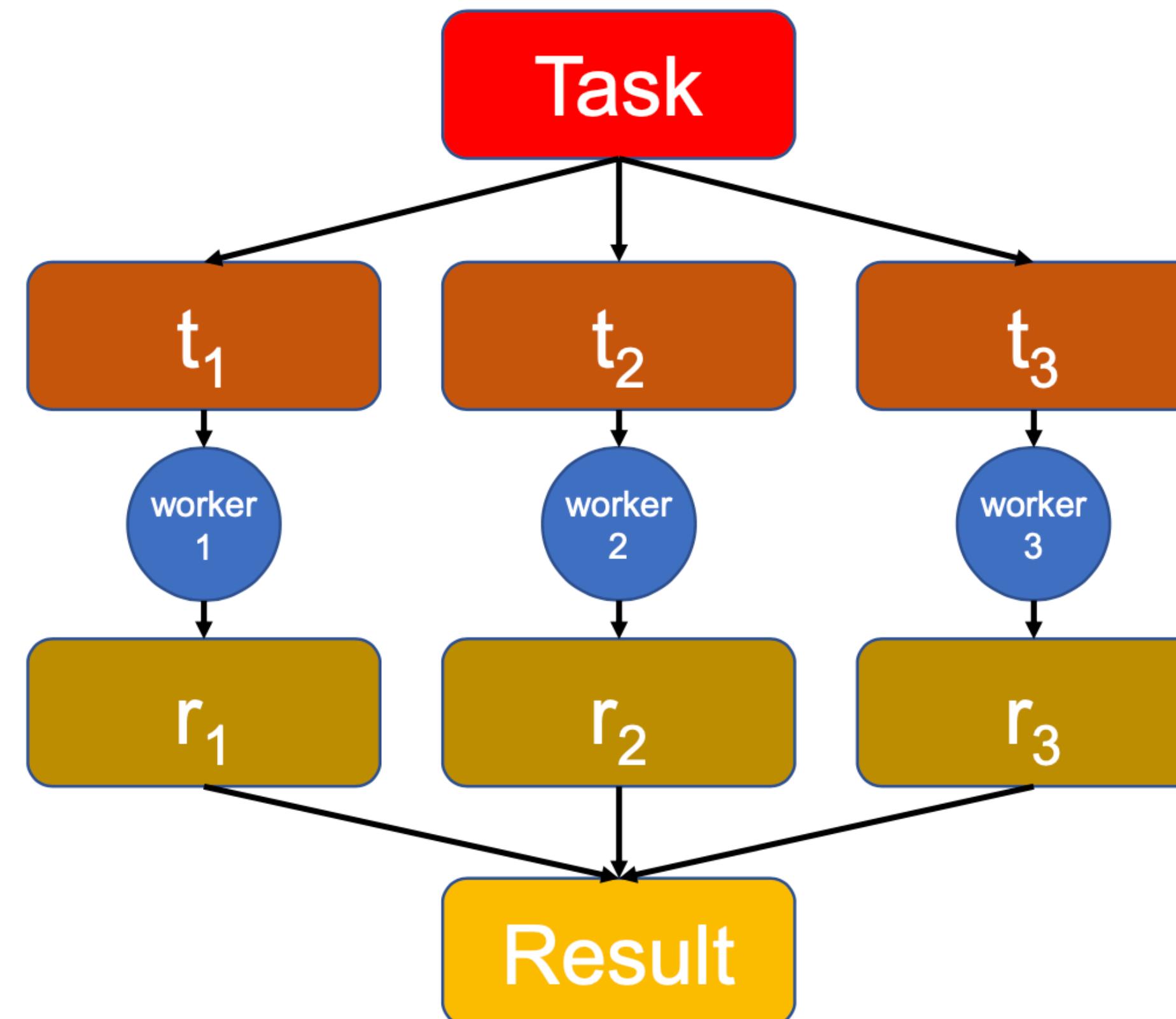


看 Demo

知识点：MapReduce 的特点 (Motivation Of Spark)

Motivation of MapReduce

- Make use of multiple workers



MapReduce

- MapReduce is a programming framework that
 - allows us to perform distributed and parallel processing on large data sets in a distributed environment
 - no need to bother about the issues like reliability, fault tolerance etc
 - offers the flexibility to write code logic without caring about the design issues of the system

Map Reduce

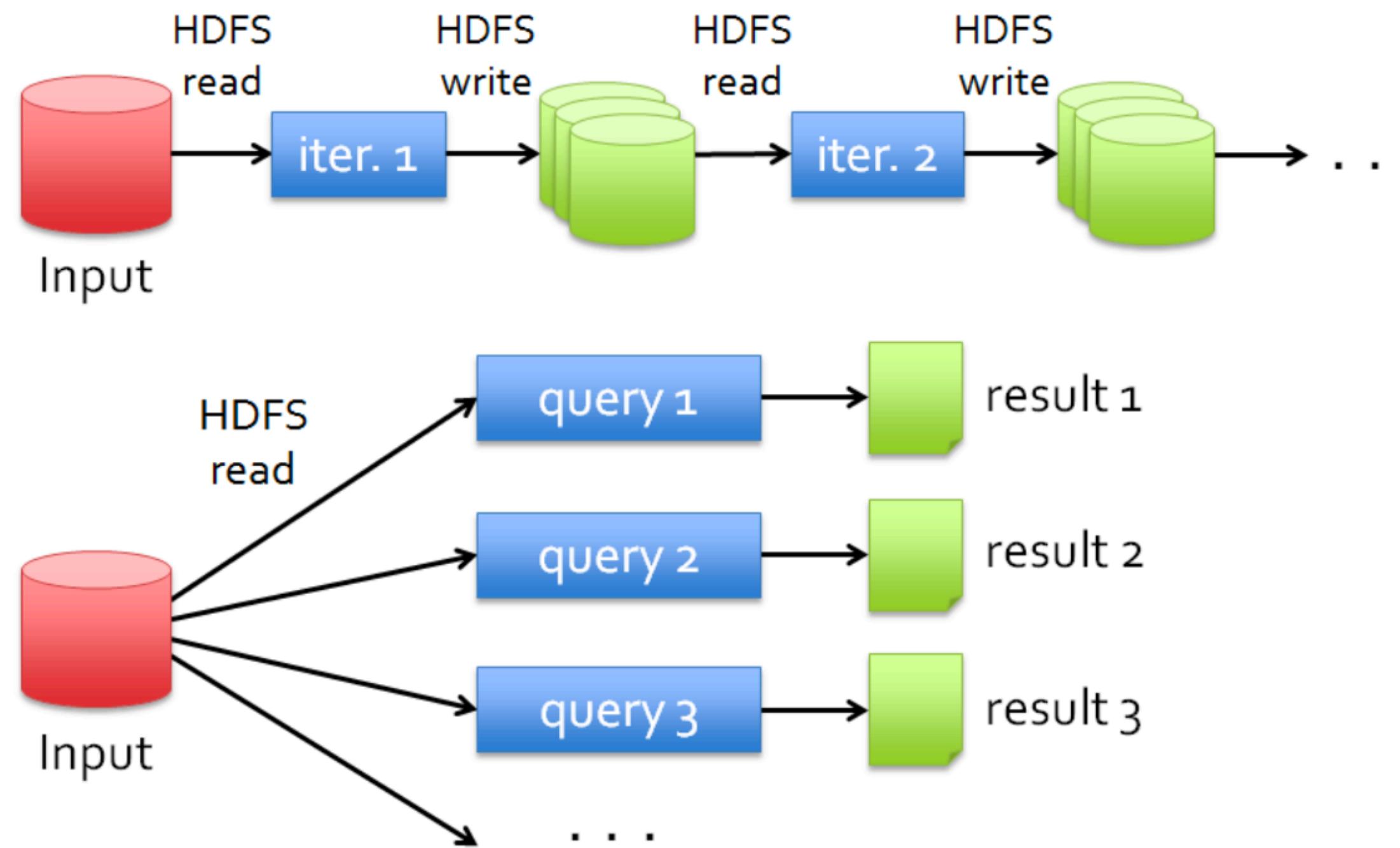
- MapReduce consists of Map and Reduce
- Map
 - Reads a block of data
 - Produces key-value pairs as intermediate outputs
- Reduce
 - Receive key-value pairs from multiple map jobs
 - aggregates the intermediate data tuples to the final output

知识点: MapReduce 的缺点 (Motivation Of Spark)

Limitations of MapReduce

- As a general programming model:
 - more suitable for one-pass computation on a large dataset
 - hard to compose and nest multiple operations
 - no means of expressing iterative operations
- As implemented in Hadoop
 - all datasets are read from disk, then stored back on to disk
 - all data is (usually) triple-replicated for reliability

Data Sharing in Hadoop MapReduce



- Slow due to replication, serialization, and disk IO
- Complex apps, streaming, and interactive queries all need one thing that MapReduce lacks:
 - Efficient primitives for data sharing

关于MapReduce 的缺点一句话：因为总要读写磁盘，所以慢

Spark 怎么解决的？把中间计算过程全存在内存里

知识点：Spark 概述

What is Spark?

- Apache Spark is an open-source cluster computing framework for real-time processing.
- Spark provides an interface for programming entire clusters with
 - implicit data parallelism
 - fault-tolerance
- Built on top of Hadoop MapReduce
 - extends the MapReduce model to efficiently use more types of computations

Spark Eco-System



Spark SQL
结构化数据

Spark Streaming
实时计算

Spark Mlib
机器学习

Spark GraphX
图计算

Spark Core

独立调度器

YARN

Mesos

Spark Core: 实现了 Spark 的基本功能，包含任务调度、内存管理、错误恢复、与存储系统交互等模块。Spark Core 中还包含了对弹性分布式数据集 (Resilient Distributed DataSet, 简称RDD) 的 API 定义。

Spark SQL: 是 Spark 用来操作结构化数据的程序包。通过 Spark SQL，我们可以使用 SQL 或者 Apache Hive 版本的 SQL 方言(HQL) 来查询数据。Spark SQL 支持多种数据源，比如 Hive 表、Parquet 以及 JSON 等。

Spark Streaming: 是 Spark 提供的对实时数据进行流式计算的组件。提供了用来操作数据流的 API，并且与 Spark Core 中的 RDD API 高度对应。

Spark MLlib: 提供常见的机器学习 (ML) 功能的程序库。包括分类、回归、聚类、协同过滤等，还提供了模型评估、数据导入等额外的支持功能。

集群管理器: Spark 设计为可以高效地在一个计算节点到数千个计算节点之间伸缩计算。为了实现这样的要求，同时获得最大灵活性，Spark 支持在各种集群管理器 (Cluster Manager) 上运行，包括 Hadoop YARN、Apache Mesos，以及 Spark 自带的一个简易调度器，叫作独立调度器。

Spark得到了众多大数据公司的支持，这些公司包括Hortonworks、IBM、Intel、Cloudera、MapR、Pivotal、百度、阿里、腾讯、京东、携程、优酷土豆。当前百度的Spark已应用于大搜索、直达号、百度大数据等业务；阿里利用GraphX构建了大规模的图计算和图挖掘系统，实现了很多生产系统的推荐算法；腾讯Spark集群达到8000台的规模，是目前已知的世界上最大的Spark集群。

Spark 和 Hadoop 的关系 (帮助理解) :

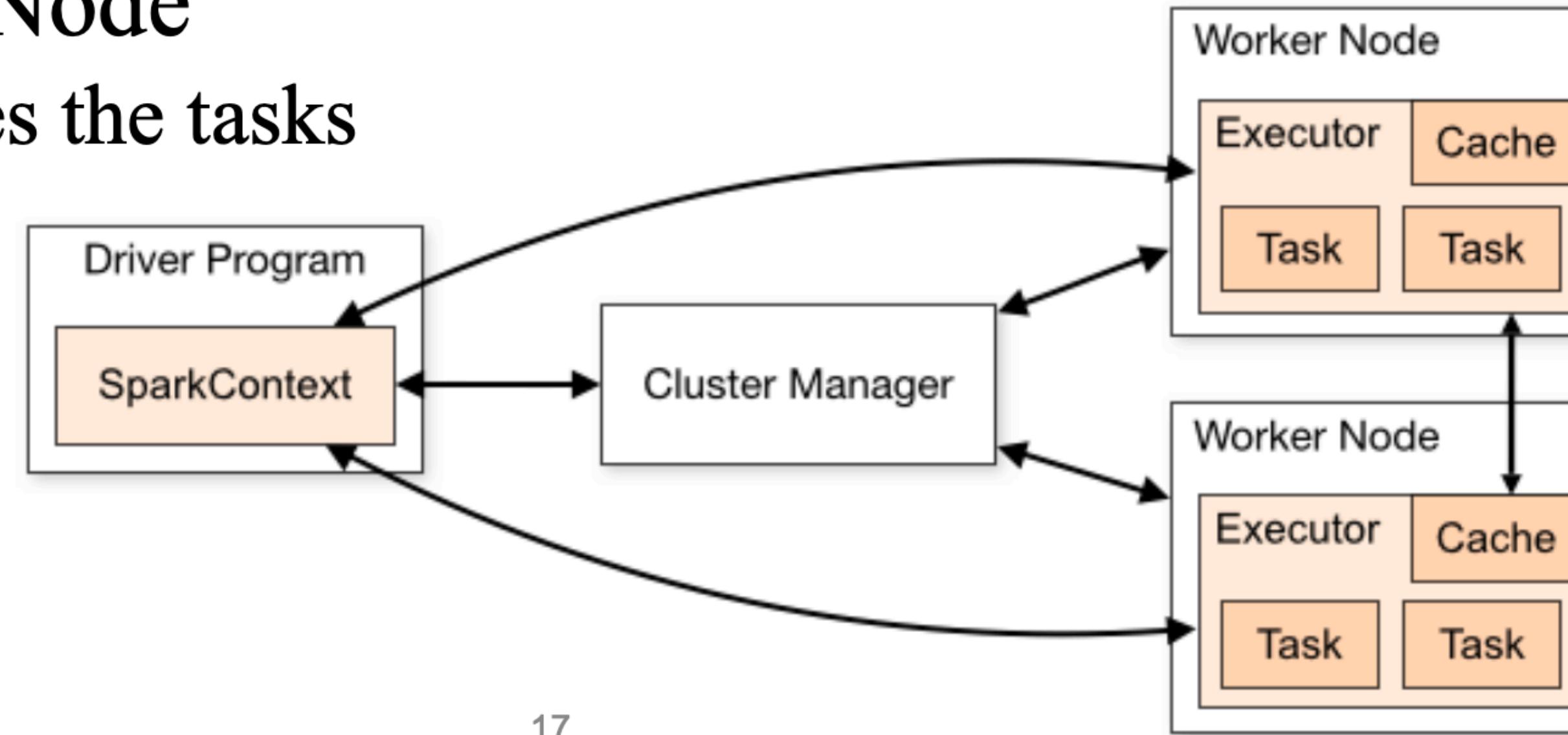
Spark 使用 Scala 写的， Scala 是基于 Java 开发的

Spark 的存储可以用单不局限于 Hadoop 的 HDFS

Spark 的任务调度可以用单不局限于 Hadoop 的 Yarn

Spark Architecture

- Master Node
 - takes care of the job execution within the cluster
- Cluster Manager
 - allocates resources across applications
- Worker Node
 - executes the tasks



1. Driver 里面声明了 SparkContext 是程序的入口 (main)
2. Cluster Manager 是一台不管计算，只负责任务，资源调度的机器
3. Worker 是负责计算的机器
4. Executor 是真正实实在在的计算线程

知识点: Spark RDD 概述

Resilient Distributed Dataset (RDD)

初始阶段可以简单理解为 RDD 就是 python 中的 list

- RDD is where the data stays
- RDD is the fundamental data structure of Apache Spark
 - is a collection of elements
 - Dataset
 - can be operated on in parallel
 - Distributed
 - fault tolerant
 - Resilient

Features of Spark RDD

- In memory computation
- Partitioning
- Fault tolerance Lazy Evaluation
- Immutability
- Persistence
- Coarse-grained operations
- Location-stickiness

Create RDDs 两种创建 RDD 的方式 (一内一外)

- Parallelizing an existing collection in your driver program
 - Normally, Spark tries to set the number of partitions automatically based on your cluster
- Referencing a dataset in an external storage system
 - HDFS, HBase, or any data source offering a Hadoop InputFormat
 - By default, Spark creates one partition for each block of the file

RDD Operations

- **Transformations**

- functions that take an RDD as the input and produce one or many RDDs as the output

- **Narrow Transformation**

- **Wide Transformation**

- **Actions**

- RDD operations that produce non-RDD values.
- returns final result of RDD computations

RDD Operation

RDD Transformations

Narrow

Map

Flatmap

Filter

Sample

Wide

ReduceByKey

GroupByKey

Fold

SortByKey

RDD Actions

Collect

Take

Reduce

ForEach

SaveAsText

RDD Demo !

知识点: Spark Lineage & DAG

为了避免计算过程中，丢失数据分区，将创建RDD的一系列 Lineage（血统）记录下来
一旦出错，可以追溯它的起源，从头再来，来实现 fault-tolerant
Lineage

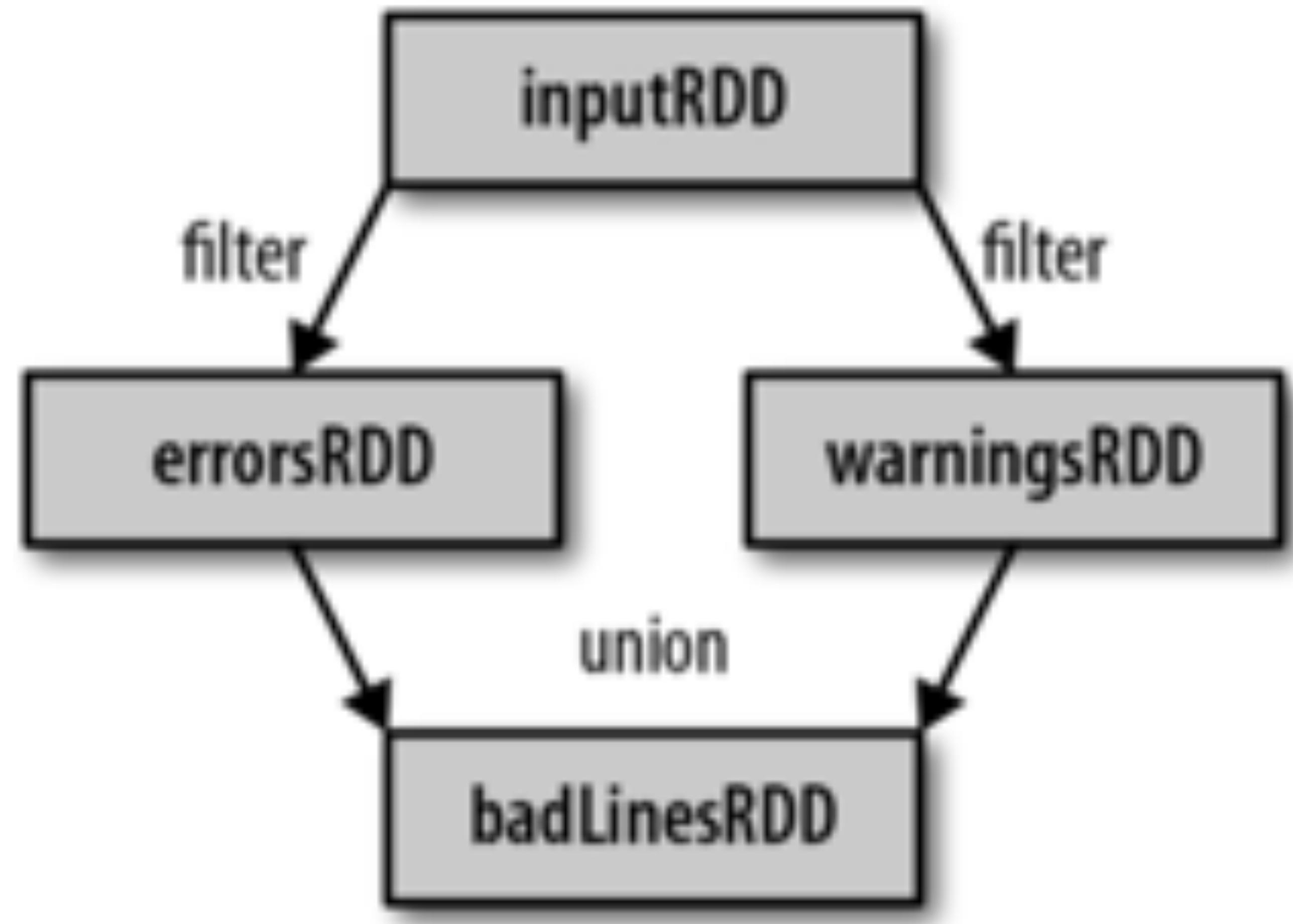
- RDD lineage is the graph of all the ancestor RDDs of an RDD
 - Also called RDD operator graph or RDD dependency graph
- Nodes: RDDs
- Edges: dependencies between RDDs

在底层，RDD 这个类型，有自己的 dependency 属性

```
/** Construct an RDD with just a one-to-one dependency on one parent */
def this(@transient oneParent: RDD[_]) =
  this(oneParent.context, List(new OneToOneDependency(oneParent)))
```

Lineage

它体现了 RDD 和 RDD 之间的一种关系



Spark-Shell Demo !

简单的 wordcount

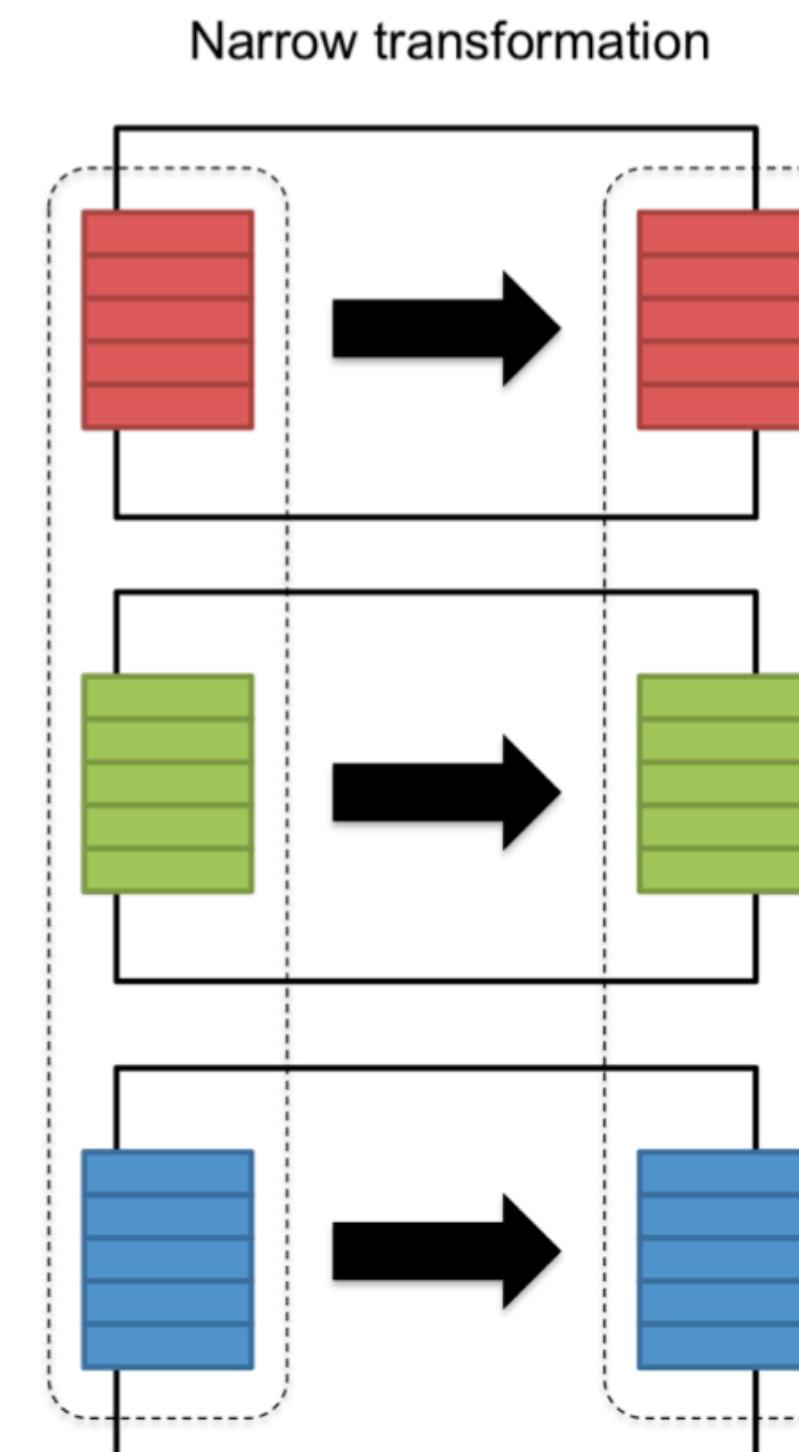
toDebugString 方法来看 lineage

.dependencies 来看某个 RDD 的依赖

Narrow and Wide Transformations

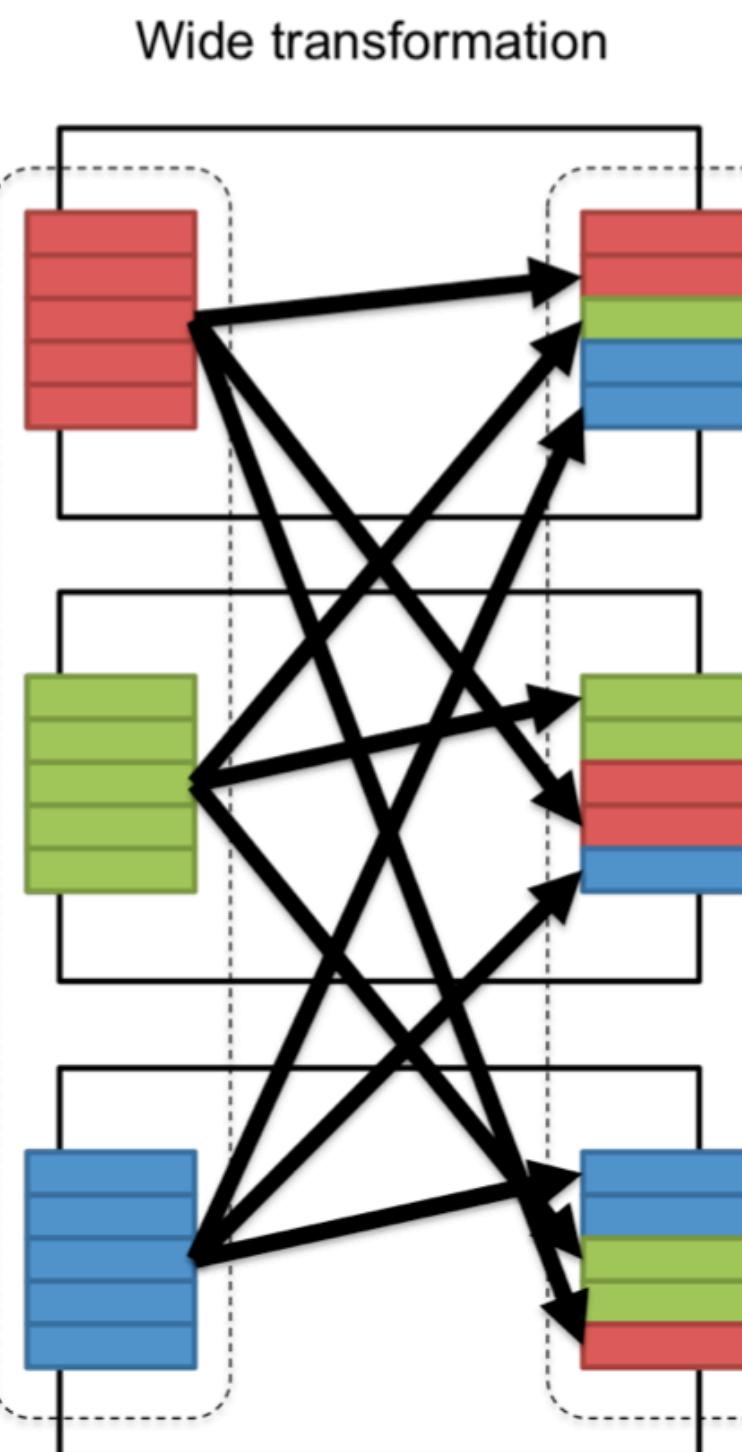
Narrow transformation
involves no data shuffling

- map
- flatMap
- filter
- sample



Wide transformation
involves data shuffling

- sortByKey
- reduceByKey
- groupByKey
- join



22

也可以从Data分区的角度来理解

map 的 Scala 底层代码体现了 Narrow transformation 这一点

```
def map[U: ClassTag](f: T => U): RDD[U] = withScope {  
    val cleanF = sc.clean(f)  
    new MapPartitionsRDD[U, T](this, (context, pid, iter) => iter.map(cleanF))  
}
```

```
/** Construct an RDD with just a one-to-one dependency on one parent */  
def this(@transient oneParent: RDD[_]) =  
    this(oneParent.context, List(new OneToOneDependency(oneParent)))
```

```
@DeveloperApi  
class OneToOneDependency[T](rdd: RDD[T]) extends NarrowDependency[T](rdd) {  
    override def getParents(partitionId: Int): List[Int] = List(partitionId)  
}
```

ReduceByKey 的 Scala 底层代码体现了 Wide transformation 这一点

知识点：DAG 有向无环图 & Stage & 任务划分

1. 为什么不能有环？

联想我们编程的时候倒入的包的依赖关系

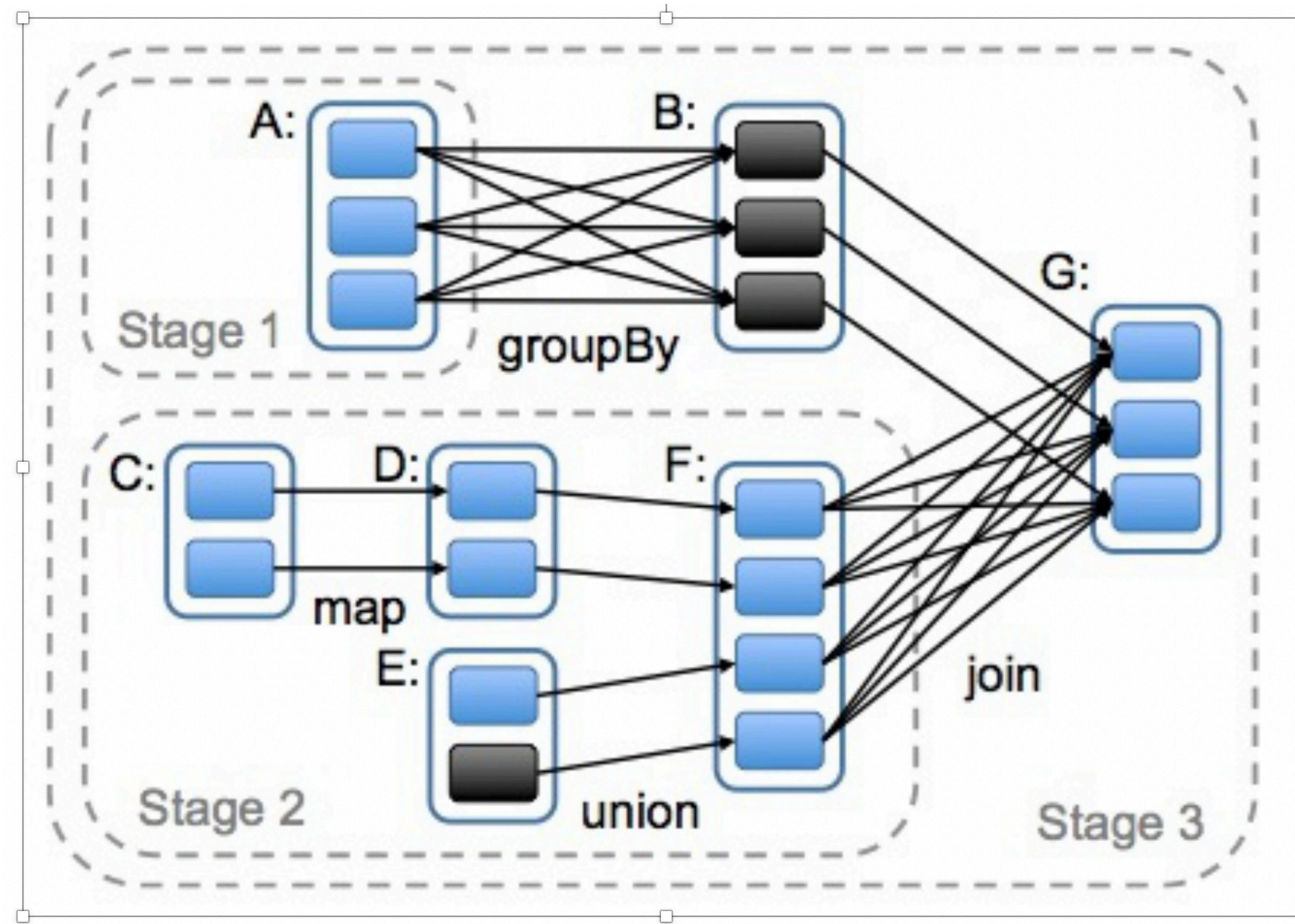
RDD 的依赖也是这样

DAG in Spark

- DAG is a direct graph with no cycle
 - Node: RDDs, results
 - Edge: Operations to be applied on RDD
- On the calling of Action, the created DAG submits to DAG Scheduler which further splits the graph into the stages of the task
- DAG operations can do better global optimization than other systems like MapReduce
先 filter 后 map
先 map 后 filter 的差别 ?

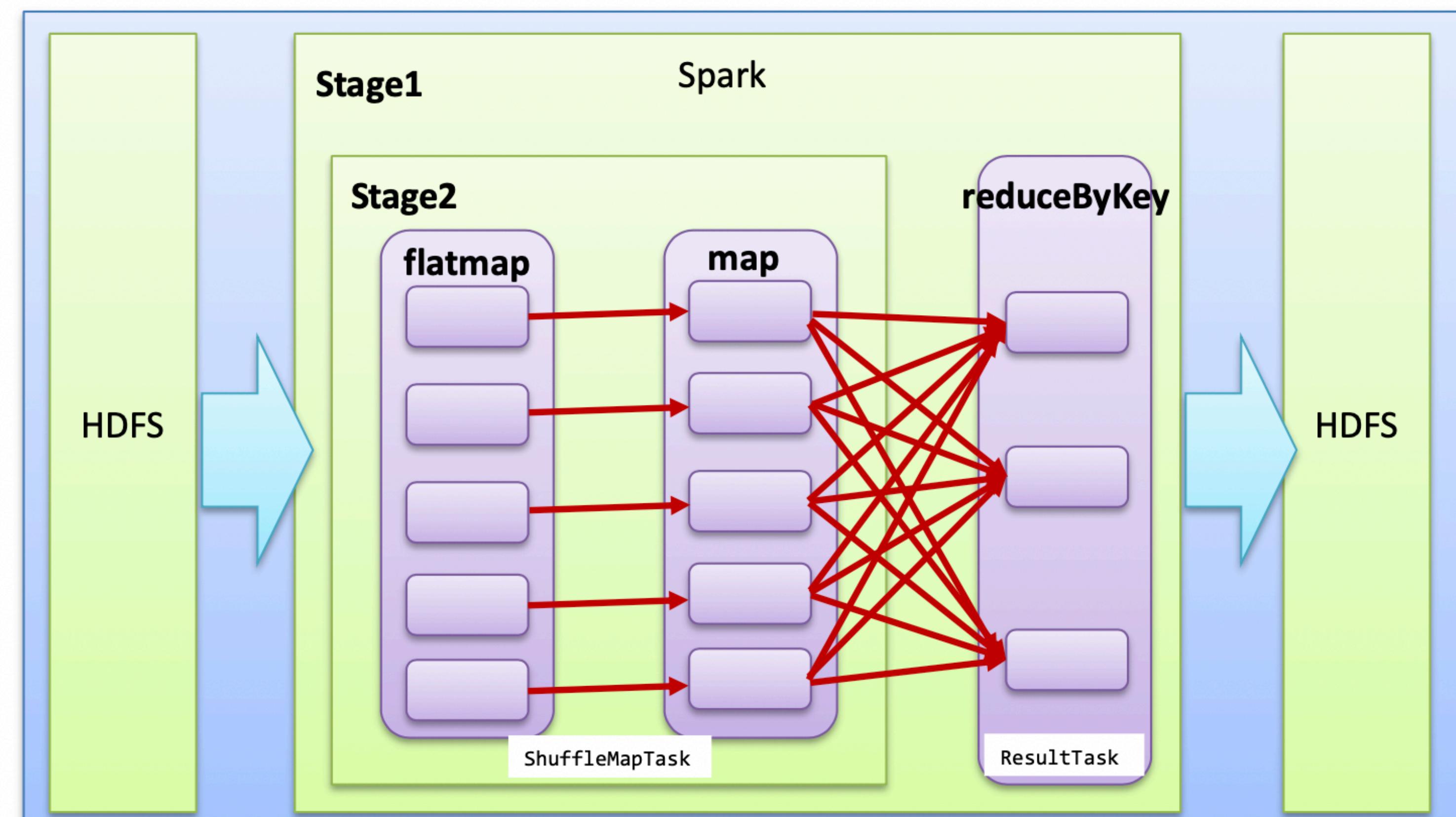
DAG(Directed Acyclic Graph)叫做有向无环图，原始的RDD通过一系列的转换就形成了DAG，根据RDD之间的依赖关系的不同将DAG划分成不同的Stage，对于窄依赖，partition的转换处理在Stage中完成计算。对于宽依赖，由于有Shuffle的存在，只能在parent RDD处理完成后，才能开始接下来的计算，因此**宽依赖是划分Stage的依据**。

判断阶段，要从后往前推



任务划分 (了解)

- Job: 一个 Action 算子就会生成一个 Job
- Stage: 根据 RDD 之间的依赖关系(DAG)的不同将 Job 划分成不同的 Stage, 遇到一个**宽依赖**则划分一个 Stage。 $1 + \text{num(shuffle)}$
- Task: 一个 Stage 里面会有很多个 Task (一个分区就是一个 Task), 同一个阶段的 task 会并行计算
- 一个 Job 会有多个 Stage
- 一个 Stage 里面会有很多个 Task



DAG, Stages and Tasks

- DAG Scheduler splits the graph into multiple stages
- Stages are created based on transformations
 - The narrow transformations will be grouped together into a single stage
 - Wide transformation define the boundary of 2 stages
- DAG scheduler will then submit the stages into the task scheduler
 - Number of tasks depends on the number of partitions
 - The stages that are not interdependent may be submitted to the cluster for execution in parallel



- HDFS 拓展内容

知识点：HDFS Erasure Coding

推荐阅读：

<https://hadoop.apache.org/docs/r3.0.0/hadoop-project-dist/hadoop-hdfs/HDFSErasureCoding.html>

<https://www.youtube.com/watch?v=jgO09opx56o>

