北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

# 并行编程原理与实践

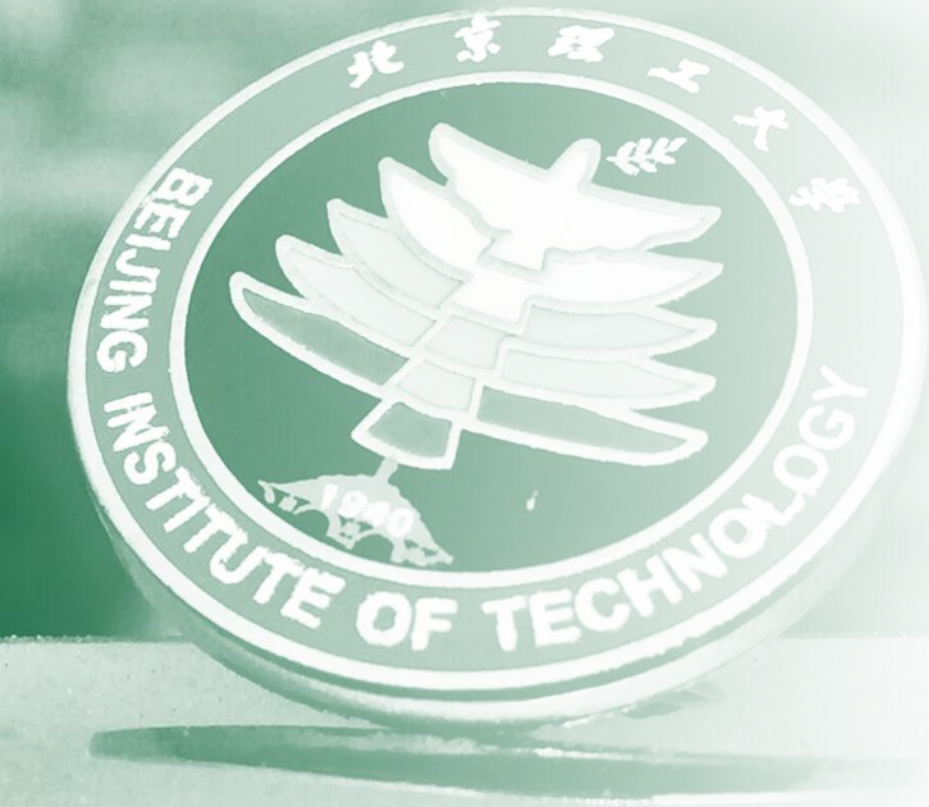
## 8. 大数据相关编程技术

王一拙、计卫星

北京理工大学计算机学院

德以明理 学以精工

# 目录 | CONTENTS



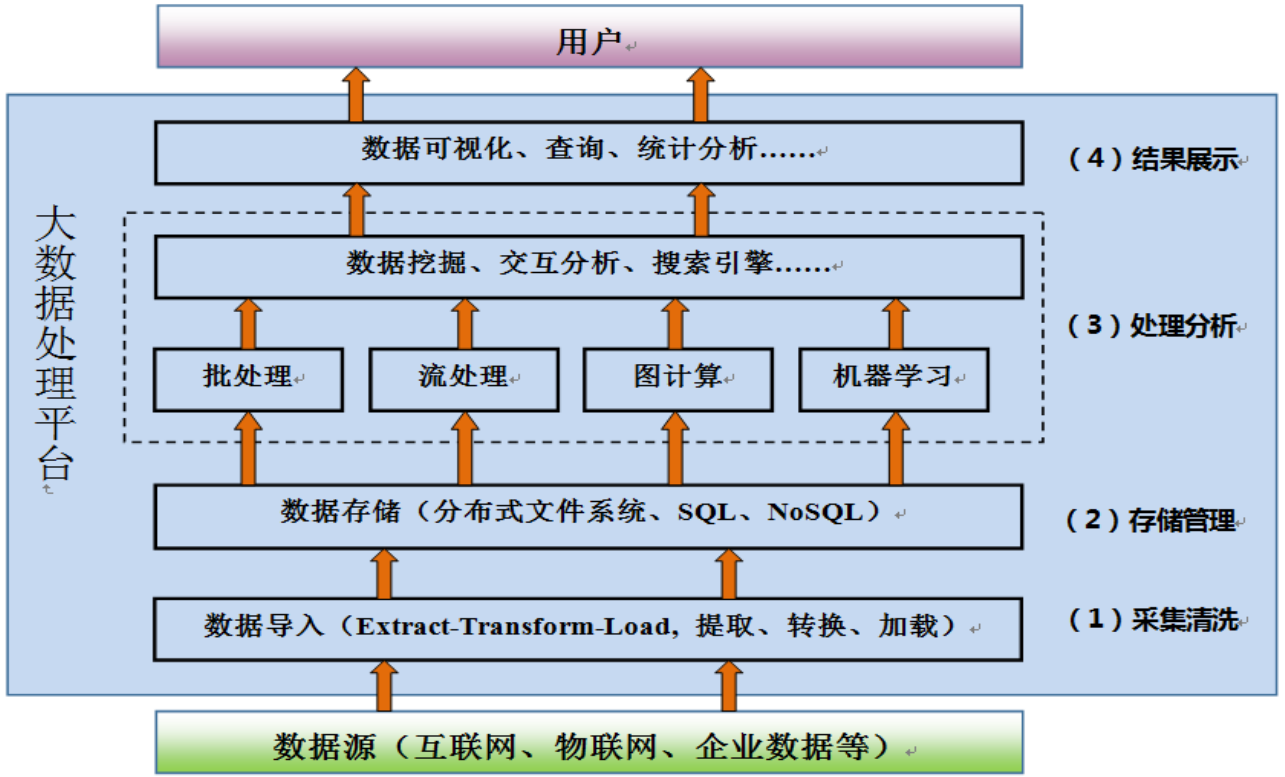

# 1 大数据处理平台概述

## 大数据处理平台技术架构

- 大数据处理过程通常包括4个环节：数据采集与清洗、数据存储与管理、计算处理与分析、结果展示

## 大数据处理平台技术架构

- 大数据处理过程通常包括4个环节：数据采集与清洗、数据存储与管理、计算处理与分析、结果展示



## Hadoop简介

- Hadoop是一个分布式系统基础架构，由Apache基金会开发。
- 2006年2月从Nutch项目中分离出来,正式成为Apache顶级项目之一。
- 作者:Doug Cutting
- 官方网站http://hadoop.apache.org
- 用Java编写
- 运行平台:Linux, Mac OS/X, Solaris, Windows


## Google Origins

SOSP 2003

**The Google File System**

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung
Google*

→ hadoop HDFS

OSDI 2004

**MapReduce: Simplified Data Processing on Large Clusters**

Jeffrey Dean and Sanjay Ghemawat
jeff@google.com, sanjay@google.com
*Google, Inc.*

→ hadoop MapReduce

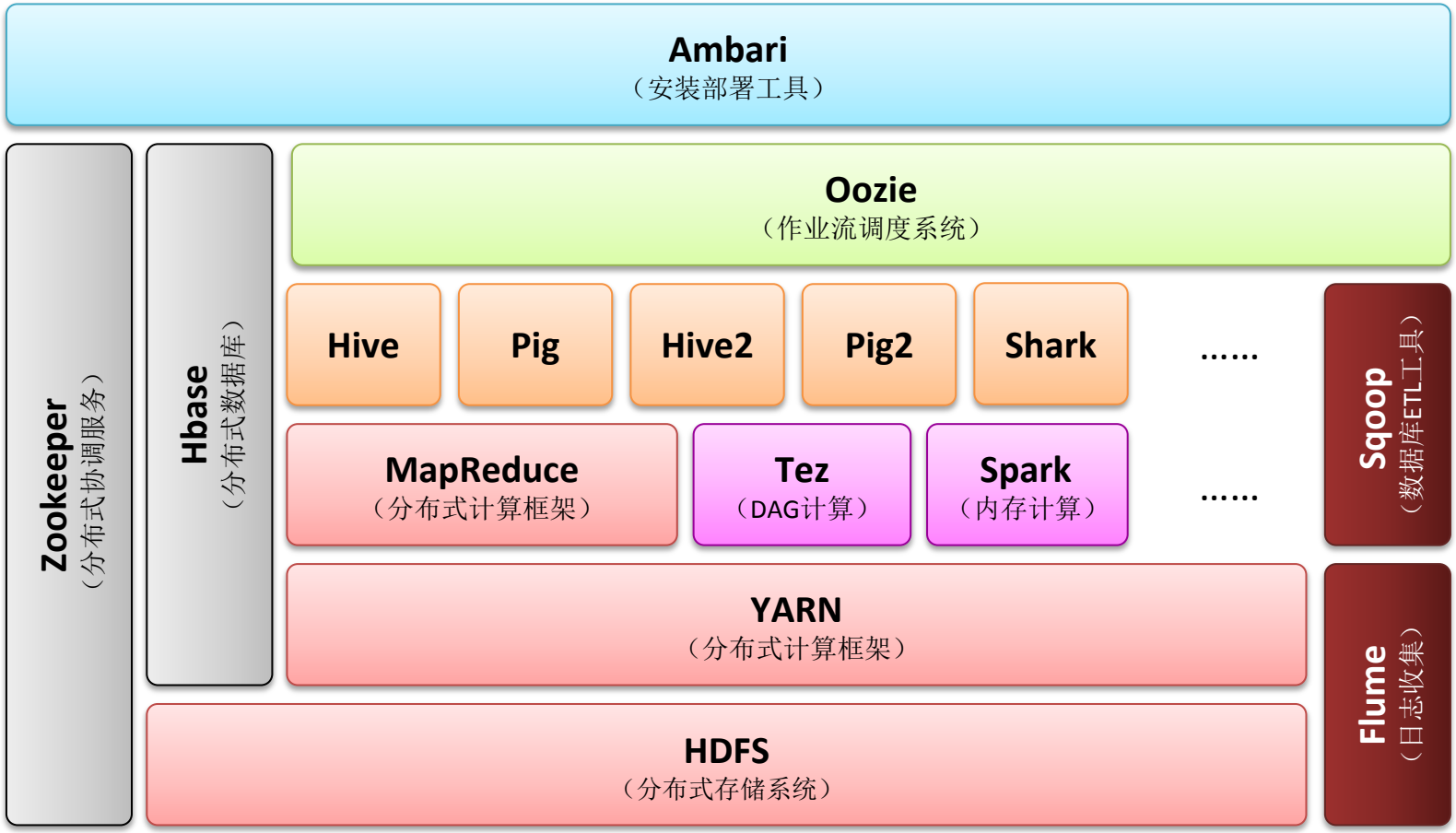
OSDI 2006

**Bigtable: A Distributed Storage System for Structured Data**

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach
Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber
{fay,jeff,sanjay,wilsonh,kerr,m3b,tushar,fikes,gruber}@google.com
*Google, Inc.*

→ APACHE HBASE

## Hadoop 生态系统


Ambari
（安装部署工具）
Zookeeper
（分布式协调服务）
Hbase
（分布式数据库）
Oozie
（作业流调度系统）
Hive
Pig
Hive2
Pig2
Shark
.....
MapReduce
（分布式计算框架）
Tez
（DAG计算）
Spark
（内存计算）
.....
YARN
（分布式计算框架）
HDFS
（分布式存储系统）
Sqoop
（数据库ETL工具）
Flume
（日志收集）

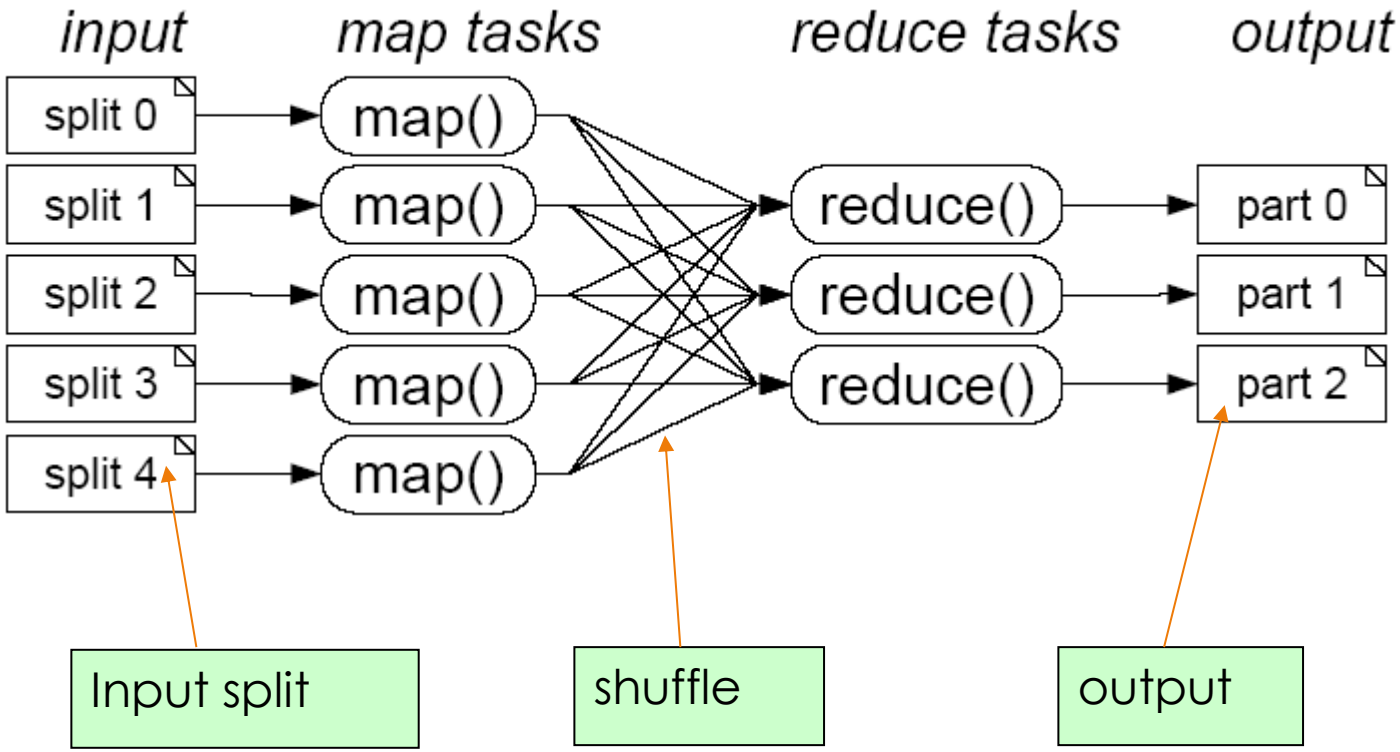
## 大数据处理的编程框架

- 批处理
  - MapReduce
  - Spark
- 流处理
  - Storm
  - Spark Streaming


# 2 MapReduce

## Mapreduce概述

- MapReduce是一个分布式/并行编程模型



## MapReduce解决问题的基本过程

- 读入数据: key/value 对的记录格式数据
- Map: 从每个记录里提取要处理的东西

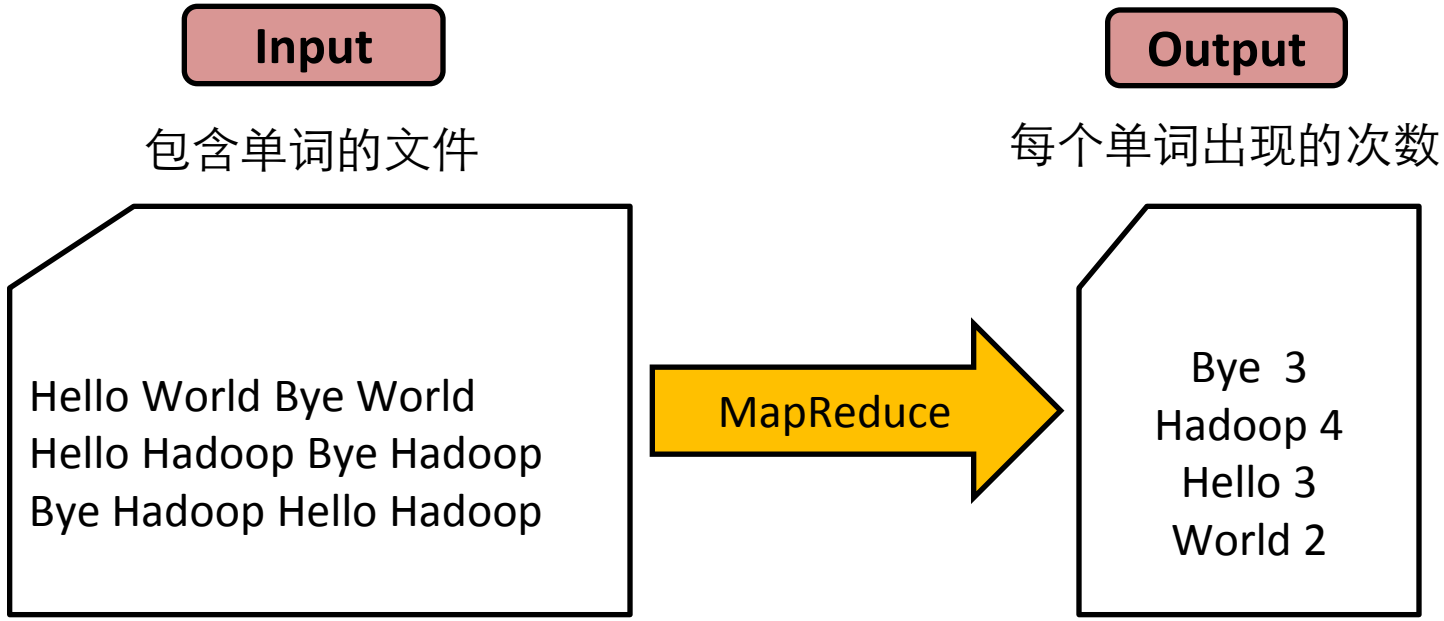
```
map(in_key, in_value) -> list(out_key, intermediate_value)
```

  - 处理input key/value pair
  - 输出中间结果key/value pairs
- Shuffle: 混排交换数据
  - 把相同key的中间结果汇集到相同节点上
- Reduce: aggregate, summarize, filter, etc.

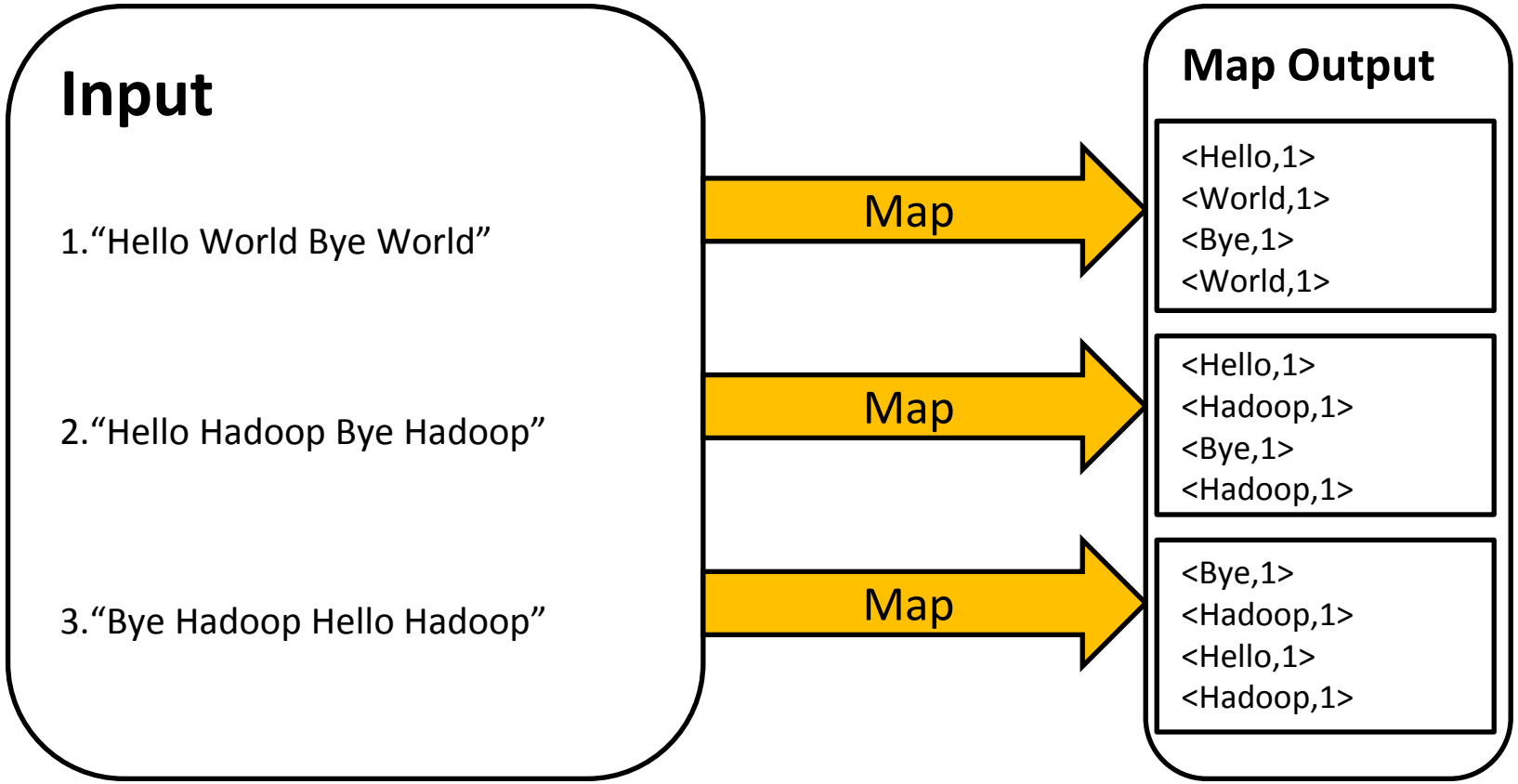
```
reduce(out_key, list(intermediate_value)) -> list(out_value)
```

  - 归并某一个key的所有values，进行计算
  - 输出合并的计算结果 (usually just one)
- 输出结果

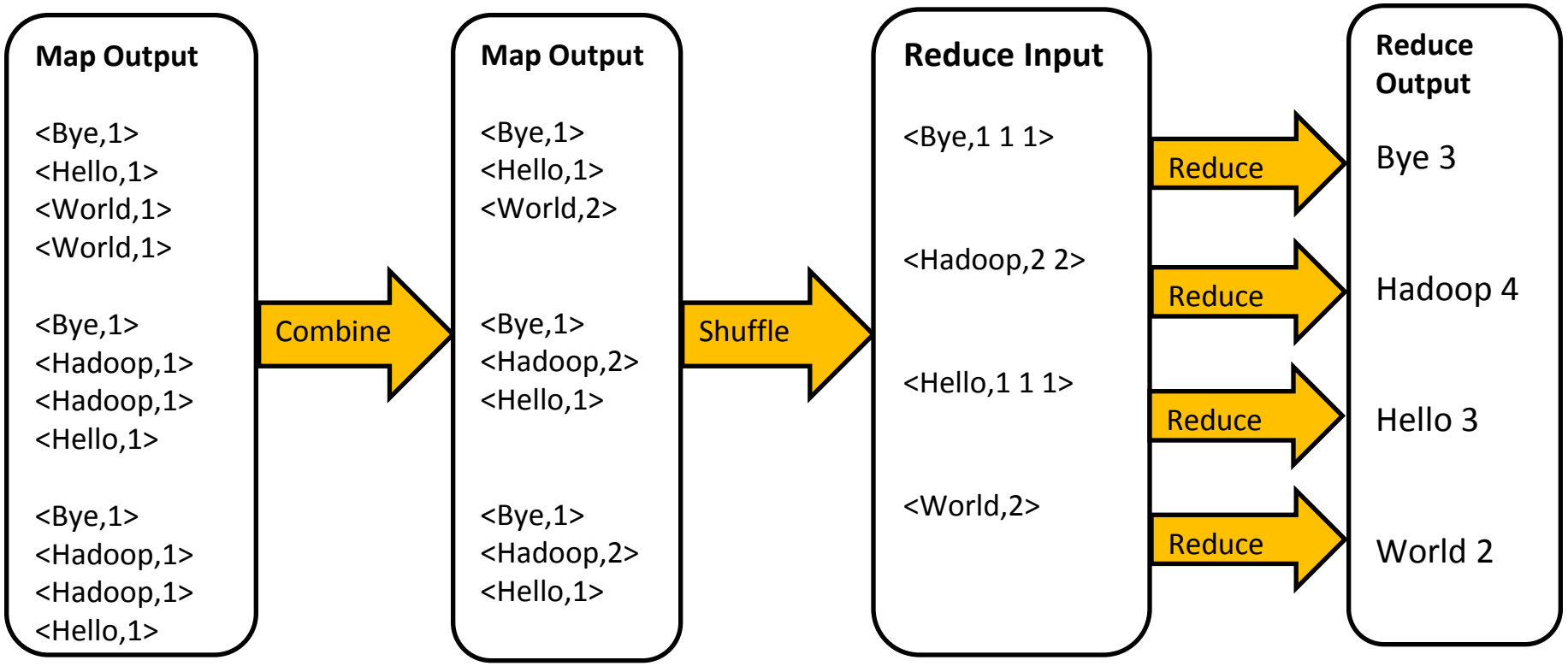
## MapReduce示例：单词计数（WordCount）



## WordCount 的Map过程



## WordCount的Reduce过程



## WordCount程序实现

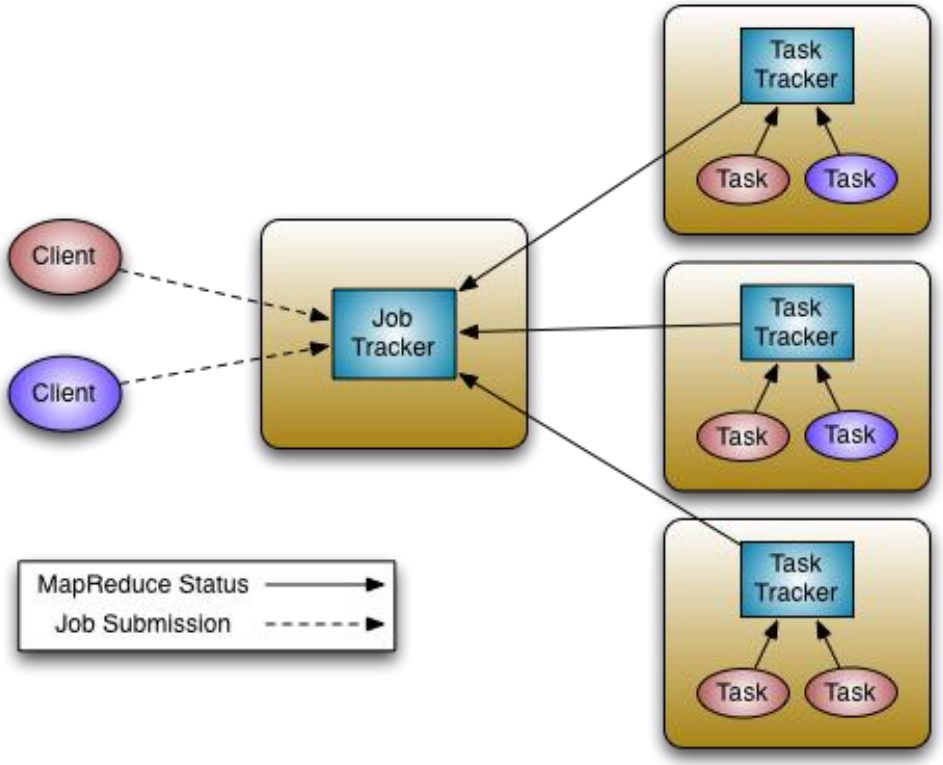
```
public static class doMapper extends Mapper<Object, Text, Text, IntWritable> {
    //第一个Object表示输入key的类型；第二个Text表示输入value的类型；
    //第三个Text表示表示输出键的类型；第四个IntWritable表示输出值的类型
    public static final IntWritable one = new IntWritable(1);
    public static Text word = new Text();
    @Override
    protected void map(Object key, Text value, Context context) throws
                        IOException, InterruptedException {
        StringTokenizer tokenizer = new StringTokenizer(value.toString(),"\t");
        //StringTokenizer是Java工具包中的一个类，用于将字符串进行拆分
        word.set(tokenizer.nextToken());
        //返回当前位置到下一个分隔符之间的字符串
        context.write(word, one);
        //将word存到容器中，记一个数
    }
}
```

## WordCount程序实现

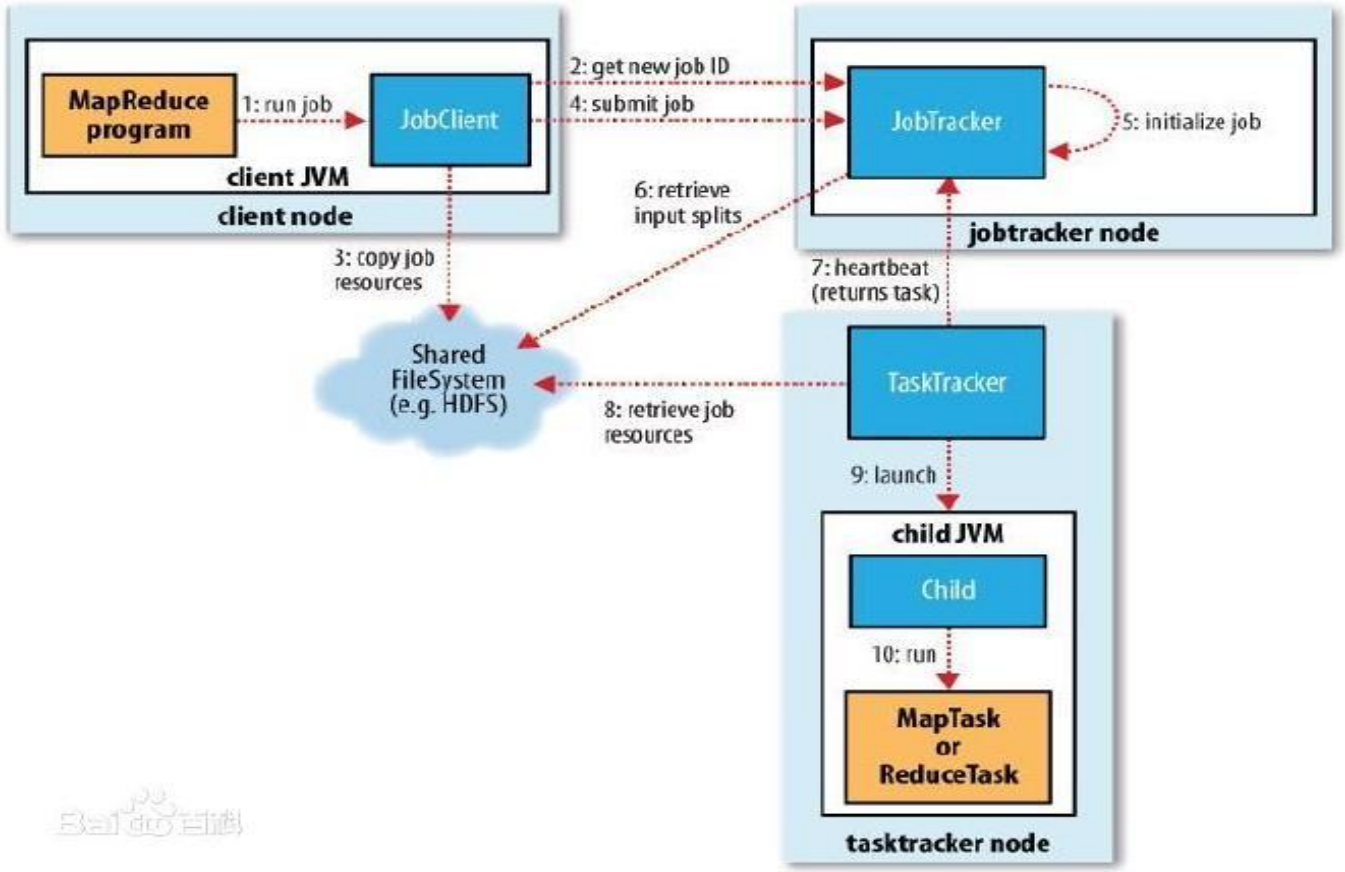
```
public static class doReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
    //参数同Map一样，依次表示是输入键类型，输入值类型，输出键类型，输出值类型
    private IntWritable result = new IntWritable();
    @Override
    protected void reduce(Text key, Iterable<IntWritable> values, Context context)
            throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable value : values) {
            sum += value.get();
        }
        //for循环遍历，将得到的values值累加
        result.set(sum);
        context.write(key, result);
    }
}
```

## Hadoop MapReduce 1.0

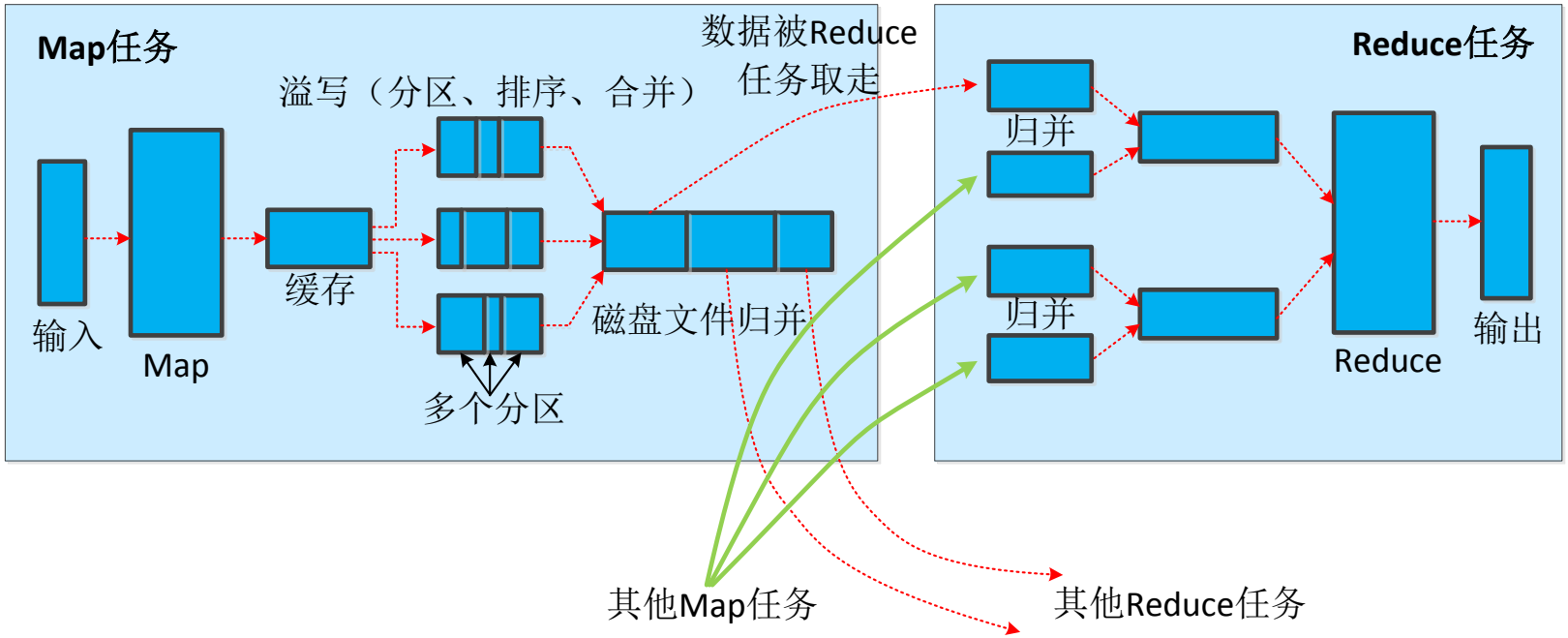
- JobTracker
  - Manages cluster resources and job scheduling
- TaskTracker
  - Per-node agent
  - Manage tasks



## MapReduce作业运行过程



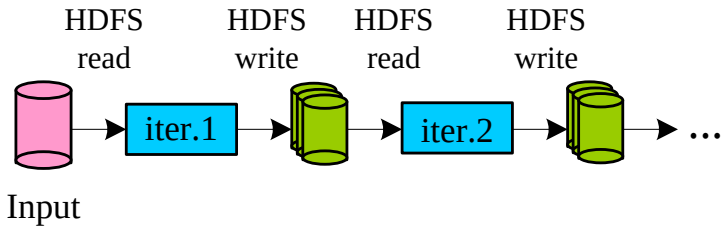
## Shuffle过程

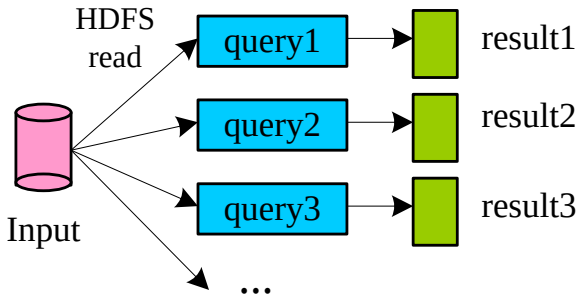



来源：厦门大学-林子雨-《大数据技术原理与应用》教材

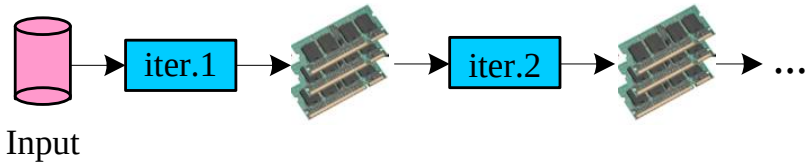

# 3 Spark

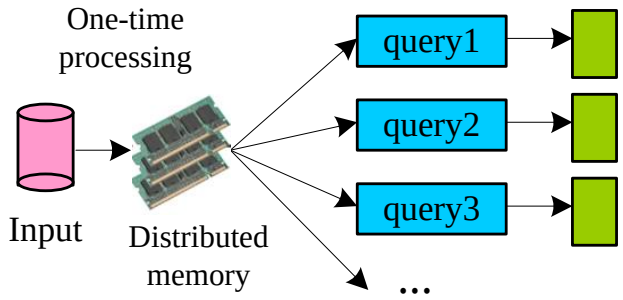
## Spark产生的动机




**Data Sharing in MapReduce**

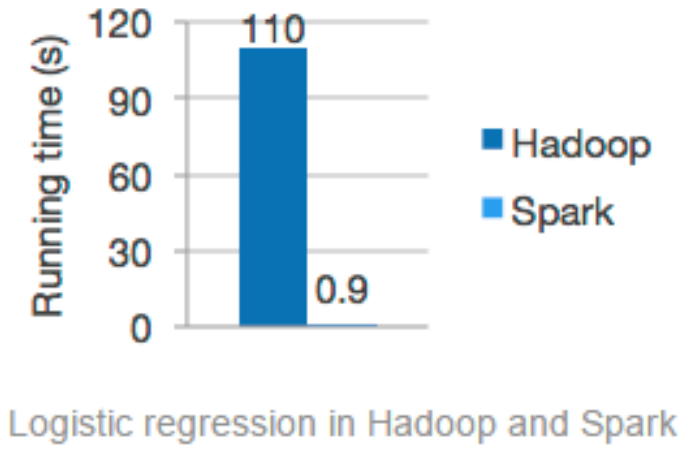


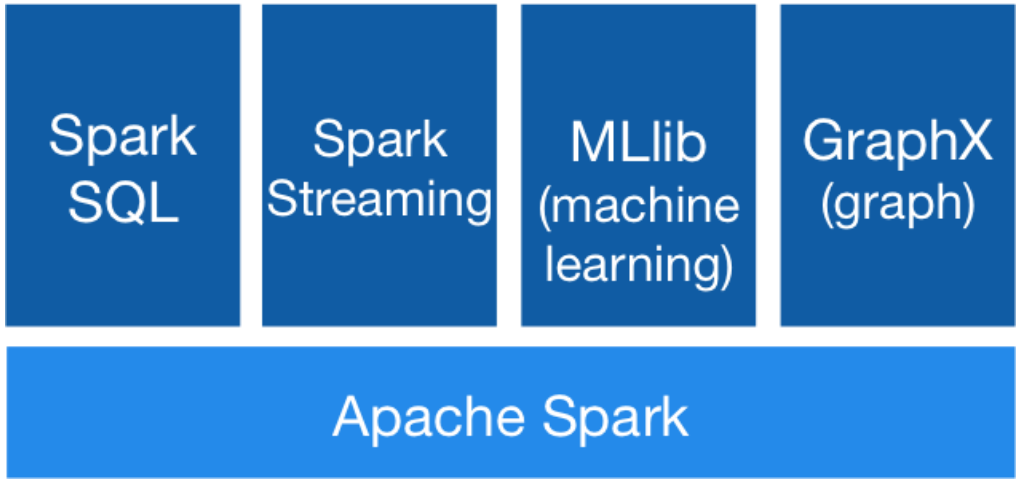

**Data Sharing in Spark**

## Spark概述

- Spark是一个快速通用的大规模数据处理框架。由UC Berkeley AMP Lab开发。可用于批处理、流处理、交互式查询、机器学习和图计算



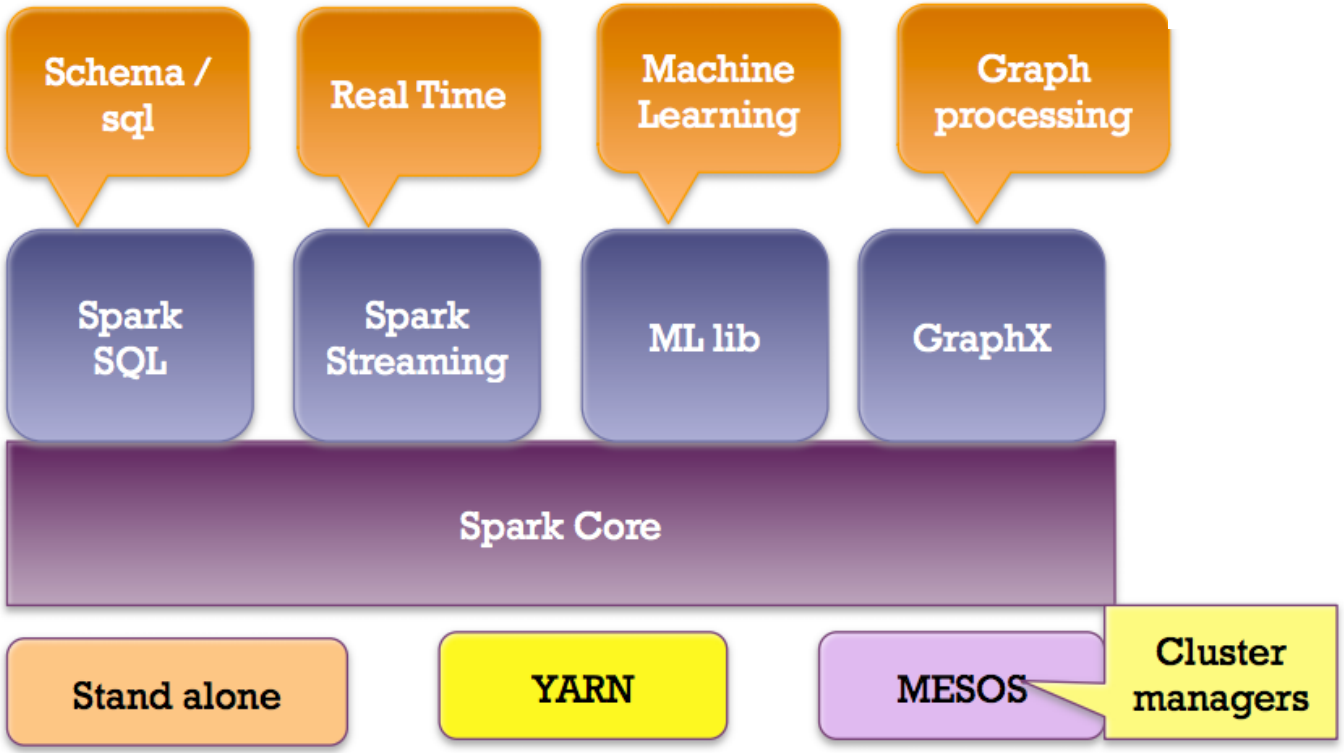

Logistic regression in Hadoop and Spark



## Spark的特点

- 运行速度快：使用DAG执行引擎以支持循环数据流与内存计算
- 容易使用：支持使用Scala、Java、Python和R语言进行编程，可以通过Spark Shell进行交互式编程
- 通用性：Spark提供了完整而强大的技术栈，包括SQL查询、流式计算、机器学习和图算法组件
- 运行模式多样：可运行于独立的集群模式中，可运行于Hadoop中，也可运行于Amazon EC2等云环境中，并且可以访问HDFS、Cassandra、HBase、Hive等多种数据源

## Spark生态系统




lephant Scale, 2014

## Spark in Scala and Java

```
// Scala:

val lines = sc.textFile(...)
lines.filter(x => x.contains("ERROR")).count()



// Java:

JavaRDD<String> lines = sc.textFile(...);
lines.filter(new Function<String, Boolean>() {
  Boolean call(String s) {
    return s.contains("error");
  }
}).count();
```

## Spark 基本概念

- RDD：是Resillient Distributed Dataset（弹性分布式数据集）的简称，是分布式内存的一个抽象概念，提供了一种高度受限的共享内存模型

## Spark 基本概念

- RDD：是Resillient Distributed Dataset（弹性分布式数据集）的简称，是分布式内存的一个抽象概念，提供了一种高度受限的共享内存模型

```
rdd1= sparkContext.textFile(“hdfs://…”)
```

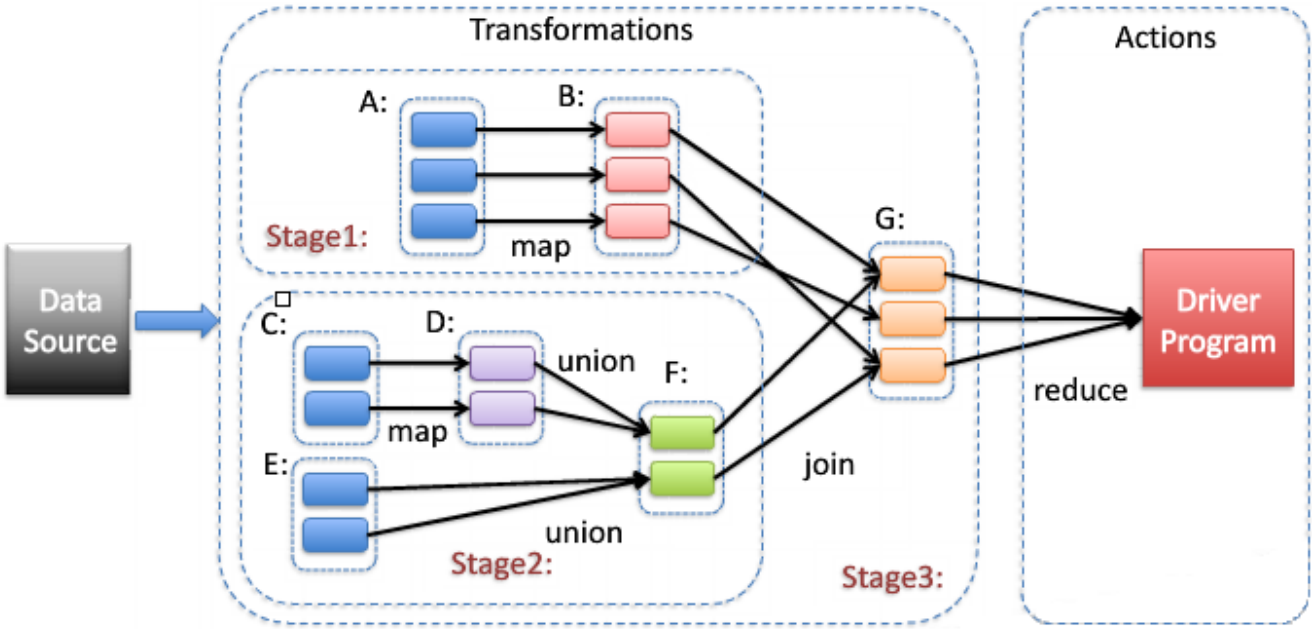
```
rdd2= rdd1.filter(_.startsWith(“ERROR”))
```

## Spark 基本概念

- RDD：是Resillient Distributed Dataset（弹性分布式数据集）的简称，是分布式内存的一个抽象概念，提供了一种高度受限的共享内存模型
- DAG：是Directed Acyclic Graph（有向无环图）的简称，反映RDD之间的依赖关系
- Executor：是运行在工作节点（WorkerNode）的一个进程，负责运行Task
- Application：用户编写的Spark应用程序
- Task：运行在Executor上的工作单元
- Job：一个Job包含多个RDD及作用于相应RDD上的各种操作
- Stage：是Job的基本调度单位，一个Job会分为多组Task，每组Task被称为Stage，或者也被称为TaskSet，代表了一组关联的、相互之间没有Shuffle依赖关系的任务组成的任务集

## Spark Transformation & Action

| | |
|---|---|
| Transformations | 将一个已经存在的RDD中转换成一个新的RDD,所有的转换操作都是lazy执行的。 |
| Actions | 一般用于对RDD中的元素进行实际的计算,然后返回相应的值。 |



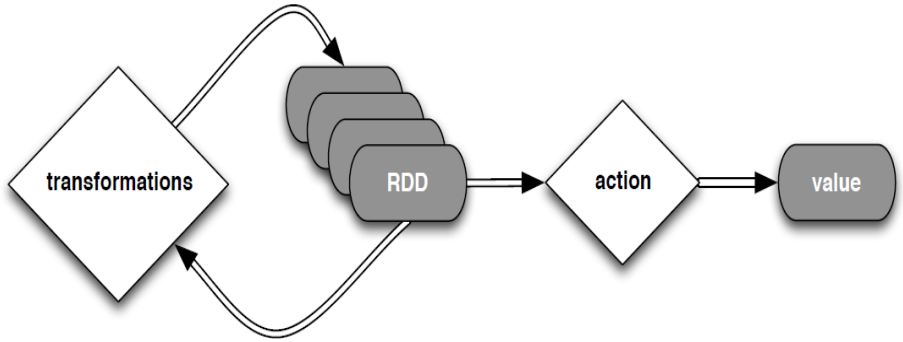
## Spark Transformation & Action

**Transformations**

- **map**(*func*)
- **filter**(*func*)
- **union**(*otherDataset*)
- **reduceByKey**(*func*, [*numTasks*])
- **repartitionAndSortWithinPartitions**(*partitioner*)

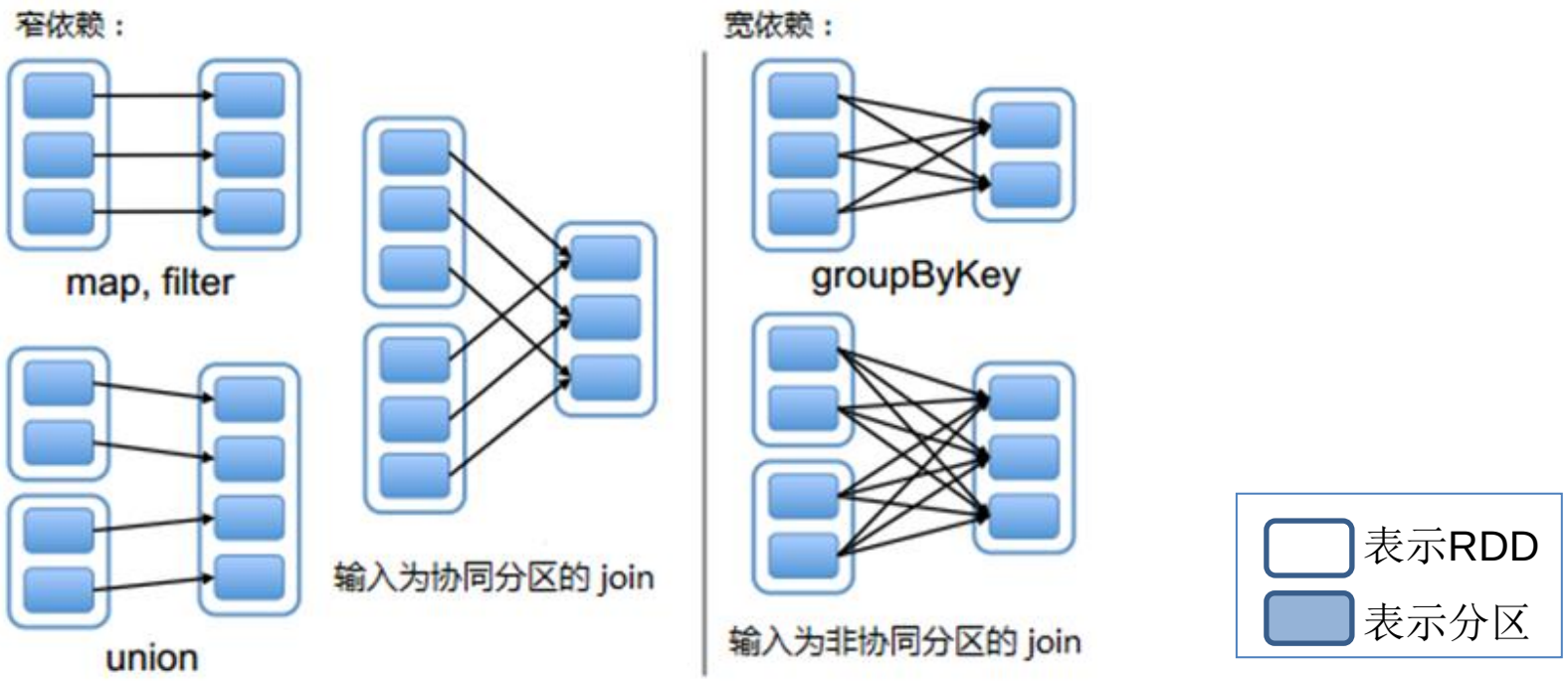
**Action**

- **reduce**(*func*)
- **collect**()
- **count**()
- **foreach**(*func*)

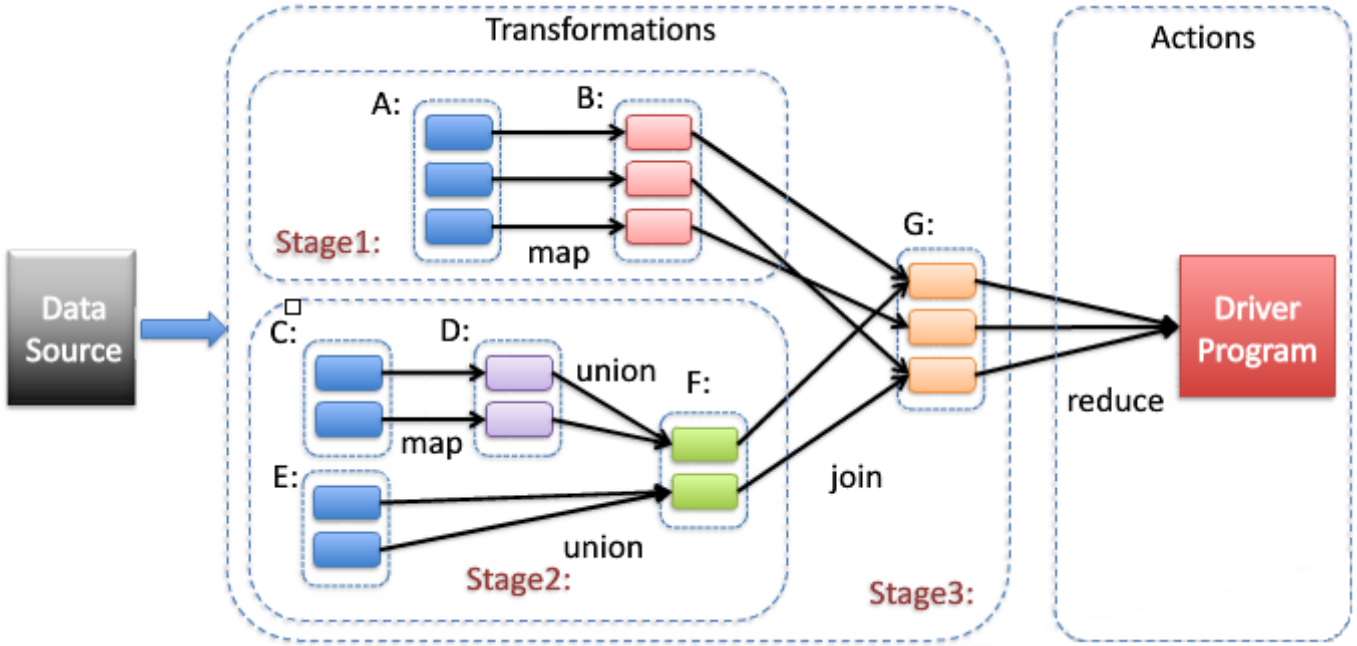


## RDD的依赖关系

- RDD之间的依赖关系可以分为两类，即：
  - 窄依赖：子RDD的每个分区依赖于常数个父分区（即与数据规模无关）；
  - 宽依赖：子RDD的每个分区依赖于所有父RDD分区。

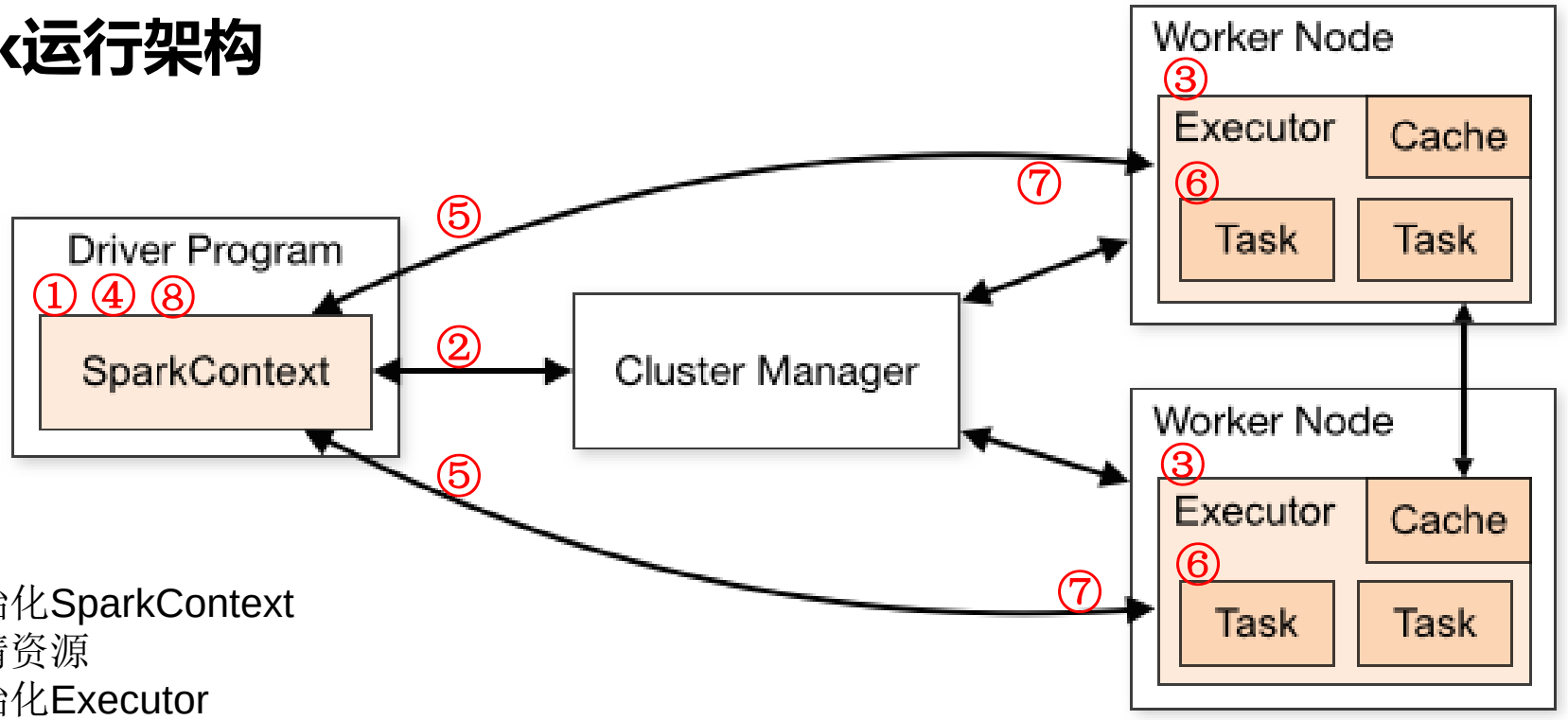


## Spark中的Stage划分




- 从HDFS中读入数据生成3个不同的RDD，通过一系列操作后，再将计算结果保存回HDFS
- 只有join操作是宽依赖，以此为边界将其前后划分成不同的Stage
- Stage2中，从map到union都是窄依赖，可以形成流水线操作

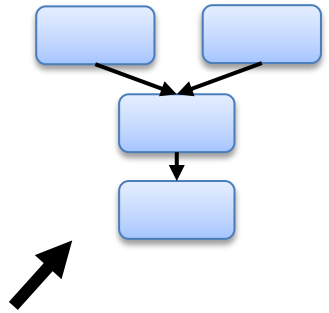
## Spark运行架构




① 初始化SparkContext
② 申请资源
③ 初始化Executor
④ 解析RDD，划分Stage，调度任务
⑤ 发送任务到Executor
⑥ 执行计算任务
⑦ 返回计算结果
⑧ 关闭SparkContex，回收资源

## Spark的工作调度（Job scheduling）

RDD Objects



```
rdd1.join(rdd2)
    .groupBy(…)
    .filter(…)
```

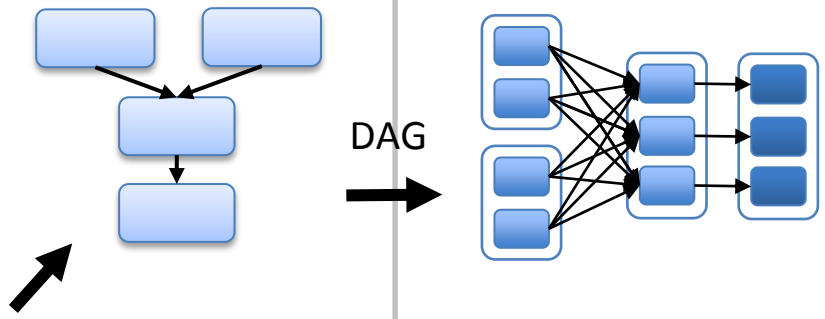
build operator DAG

source: https://cwiki.apache.org/confluence/display/SPARK/Spark+Internals

## Spark的工作调度（Job scheduling）

RDD Objects

DAGScheduler




```
rdd1.join(rdd2)
    .groupBy(…)
    .filter(…)
```

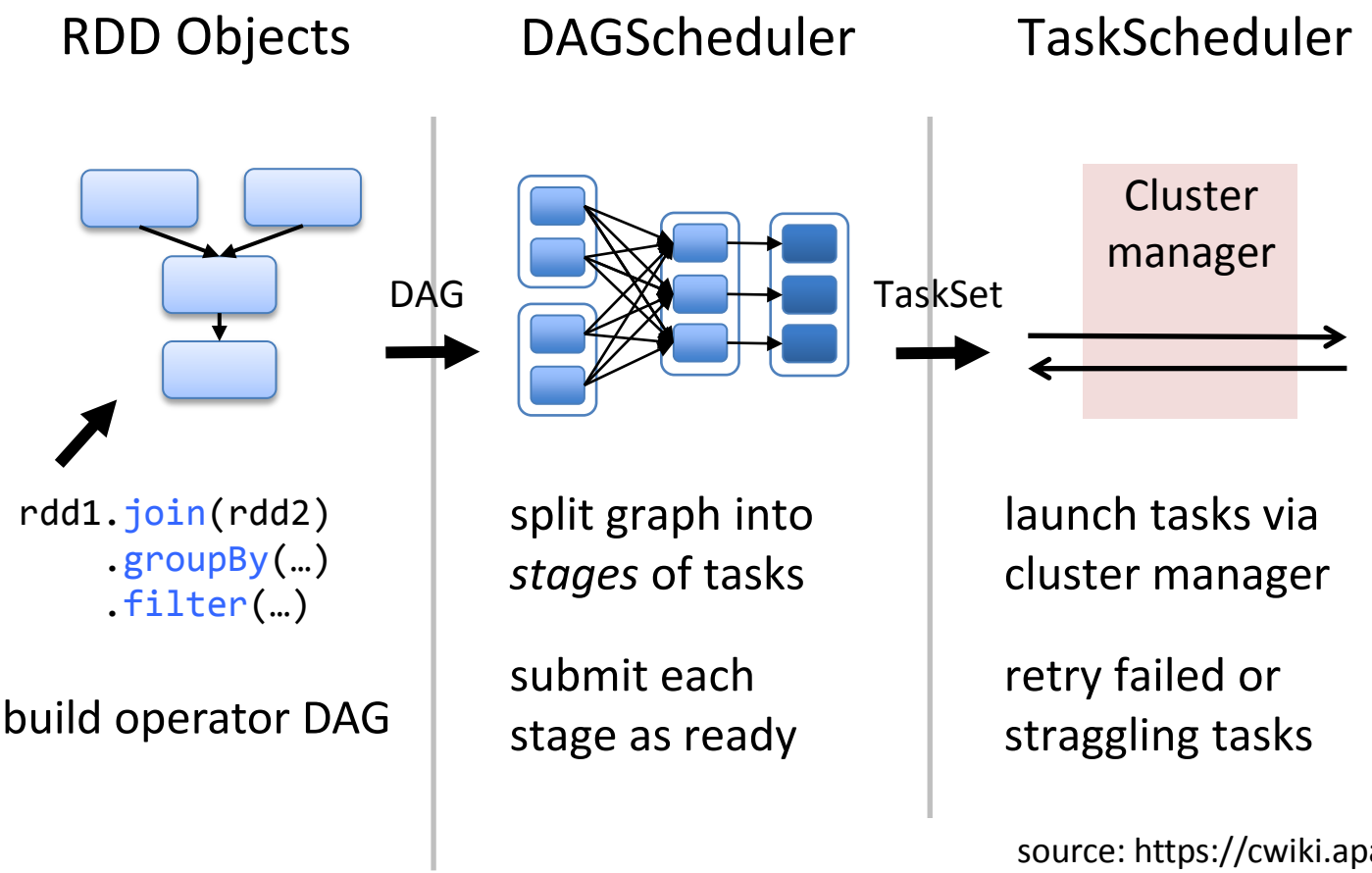
build operator DAG

split graph into *stages* of tasks

submit each stage as ready

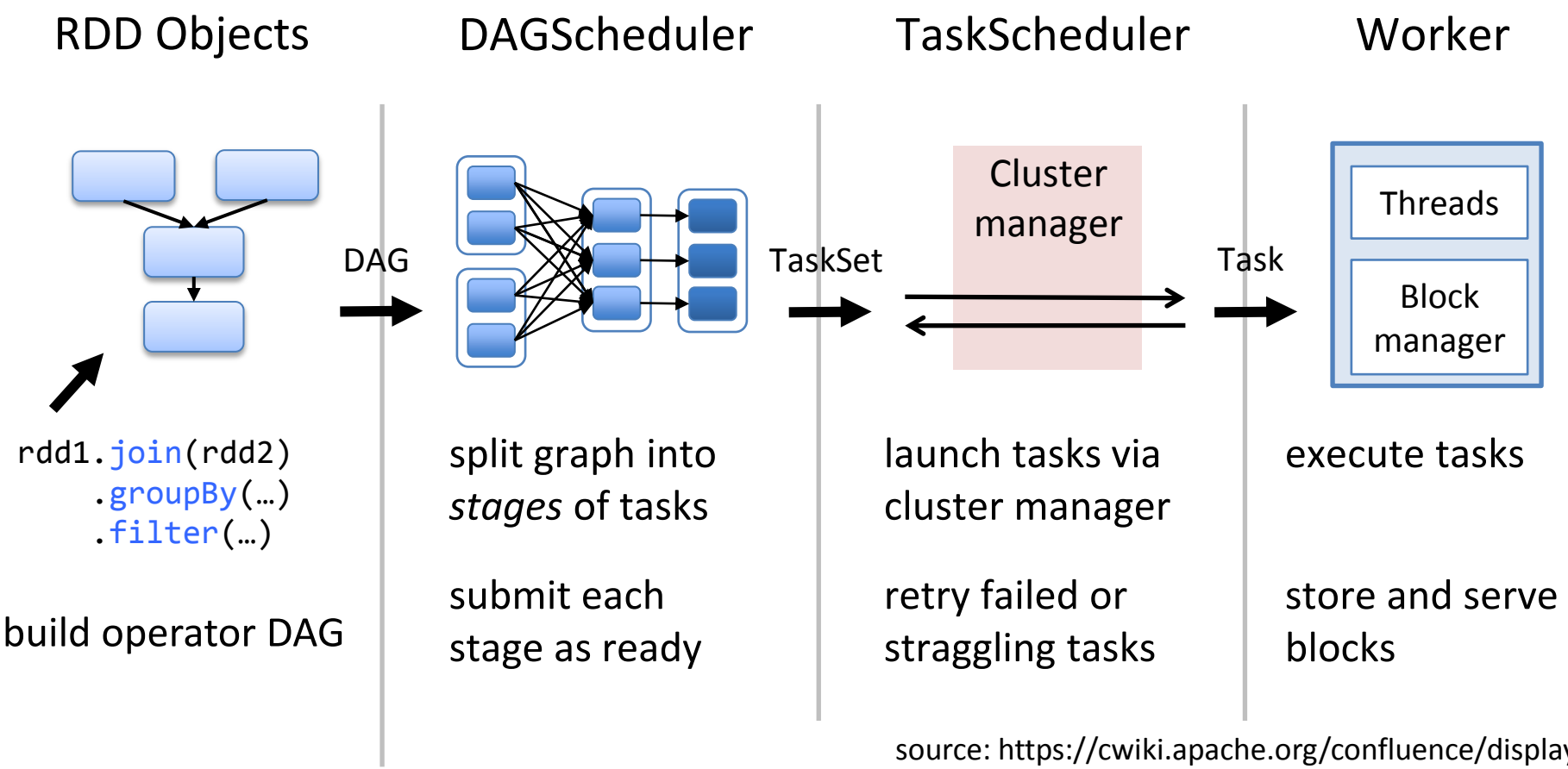
source: https://cwiki.apache.org/confluence/display/SPARK/Spark+Internals

## Spark的工作调度（Job scheduling）




source: https://cwiki.apache.org/confluence/display/SPARK/Spark+Internals

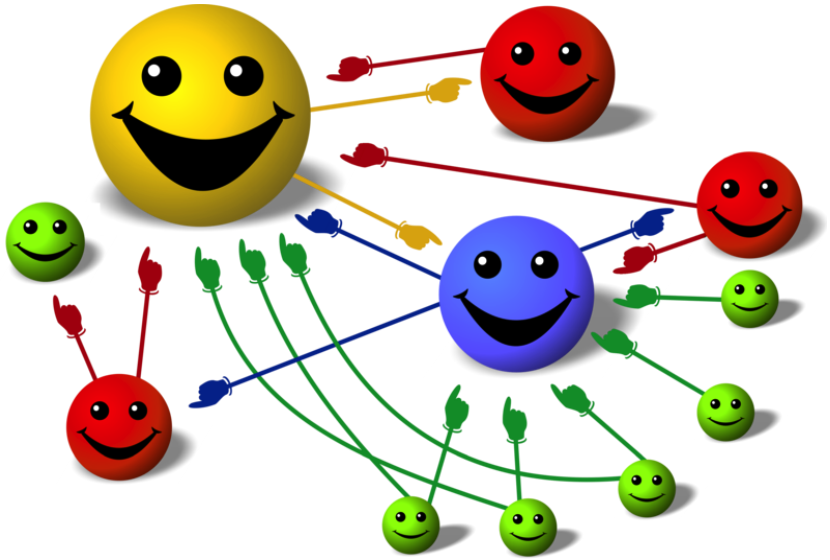
## Spark的工作调度（Job scheduling）




source: https://cwiki.apache.org/confluence/display/SPARK/Spark+Internals

## Spark示例：PageRank

- PageRank是执行多次连接的一个迭代算法，因此它是RDD分区操作的一个很好的用例
- 一个页面的重要程度用PageRank来衡量，PageRank基本原理：
  - Links from many pages ➔ high rank
  - Link from a high-rank page ➔ high rank

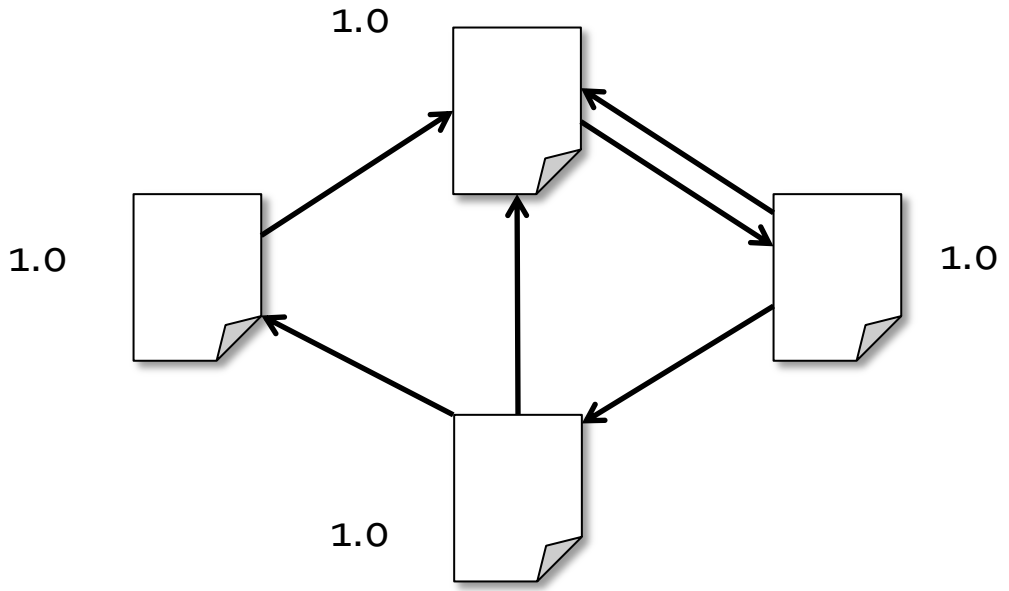


## PageRank算法

① 将每个页面的排序值初始化为1.0

② 在每次迭代中，对页面p，向其每个相邻页面（有直接链接的页面）发送一个值为rank(p)/numNeighbors(p)的贡献值

③ 将每个页面的排序值设为0.15 + 0.85 * contributionsReceived

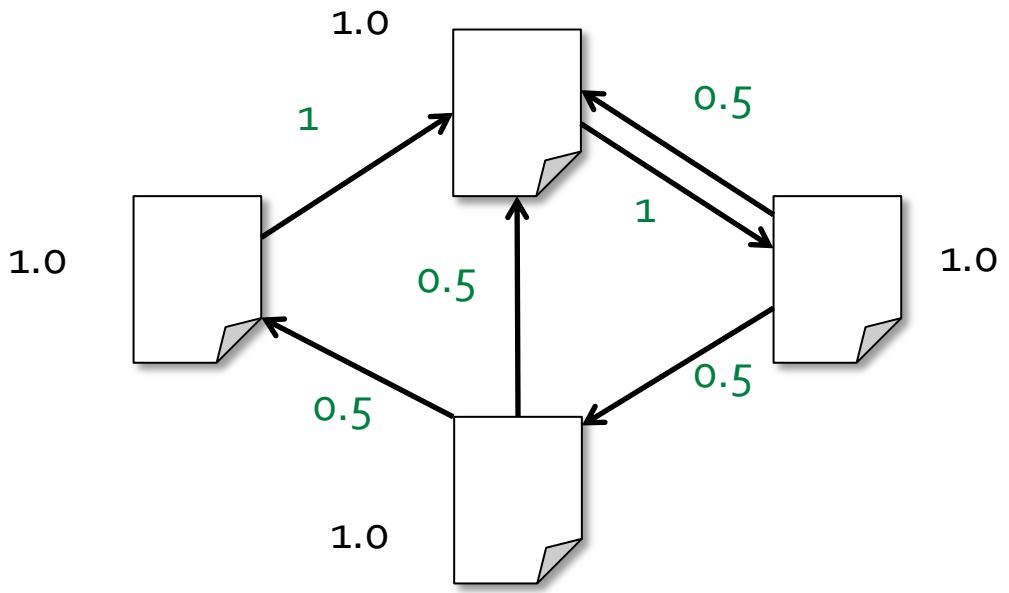


## PageRank算法

① 将每个页面的排序值初始化为1.0

② 在每次迭代中，对页面p，向其每个相邻页面（有直接链接的页面）发送一个值为rank(p)/numNeighbors(p)的贡献值

③ 将每个页面的排序值设为0.15 + 0.85 * contributionsReceived

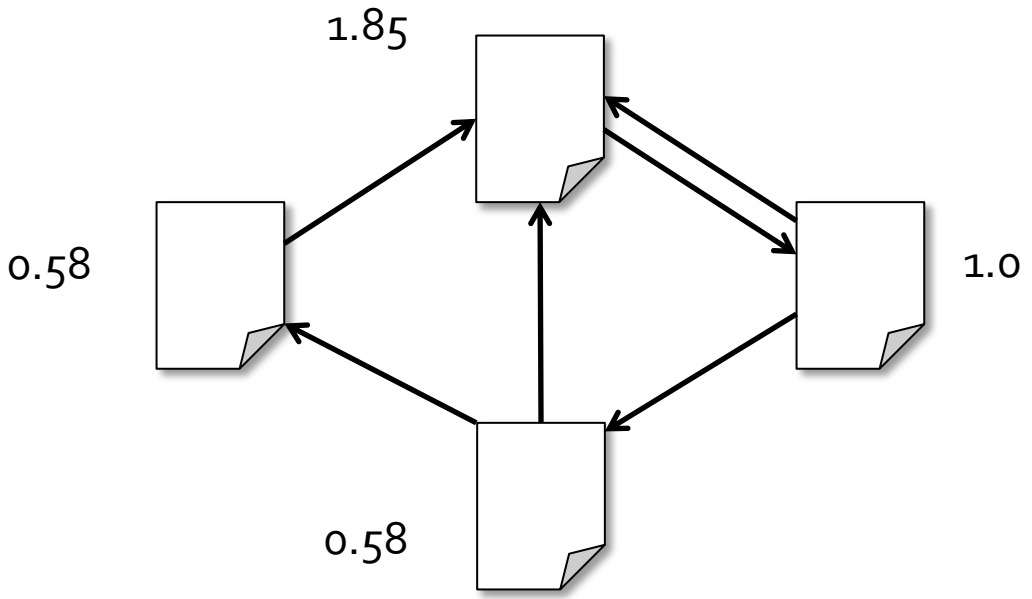


## PageRank算法

① 将每个页面的排序值初始化为1.0

② 在每次迭代中，对页面p，向其每个相邻页面（有直接链接的页面）发送一个值为rank(p)/numNeighbors(p)的贡献值

③ 将每个页面的排序值设为0.15 + 0.85 * contributionsReceived

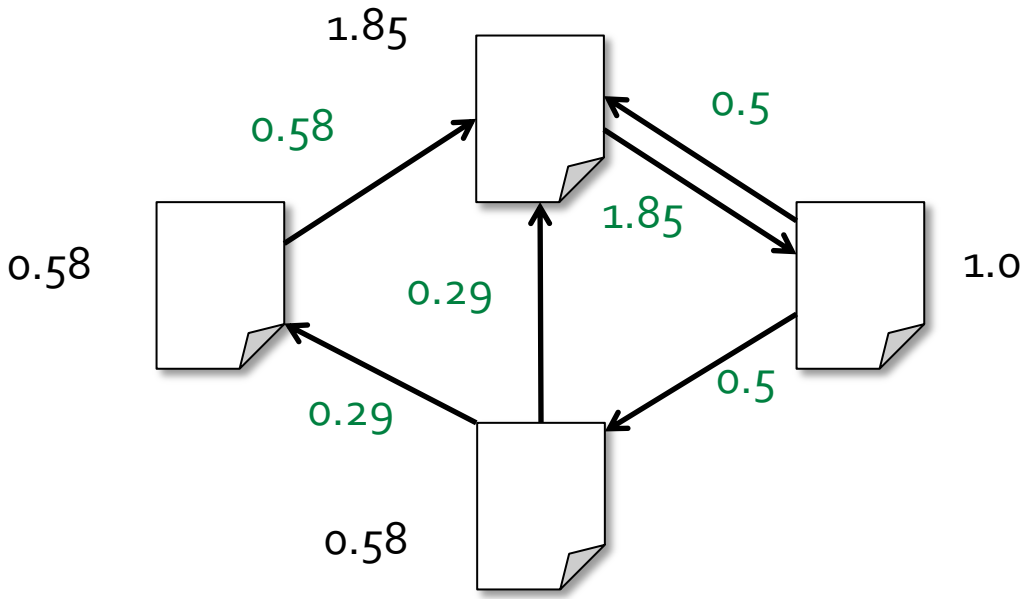


## PageRank算法

① 将每个页面的排序值初始化为1.0

② 在每次迭代中，对页面p，向其每个相邻页面（有直接链接的页面）发送一个值为rank(p)/numNeighbors(p)的贡献值

③ 将每个页面的排序值设为0.15 + 0.85 * contributionsReceived

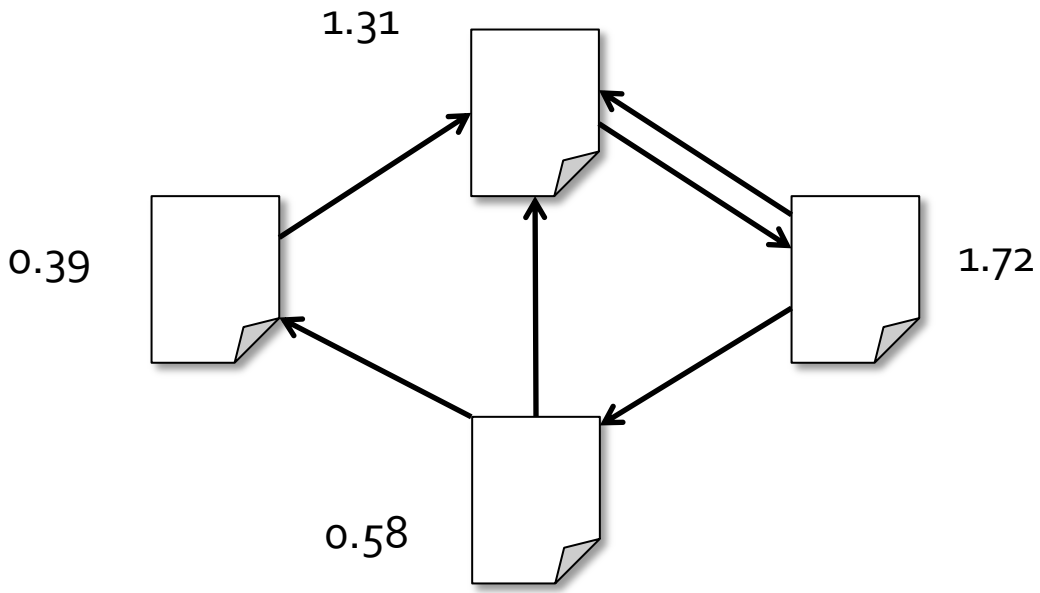


## PageRank算法

① 将每个页面的排序值初始化为1.0

② 在每次迭代中，对页面p，向其每个相邻页面（有直接链接的页面）发送一个值为rank(p)/numNeighbors(p)的贡献值

③ 将每个页面的排序值设为0.15 + 0.85 * contributionsReceived

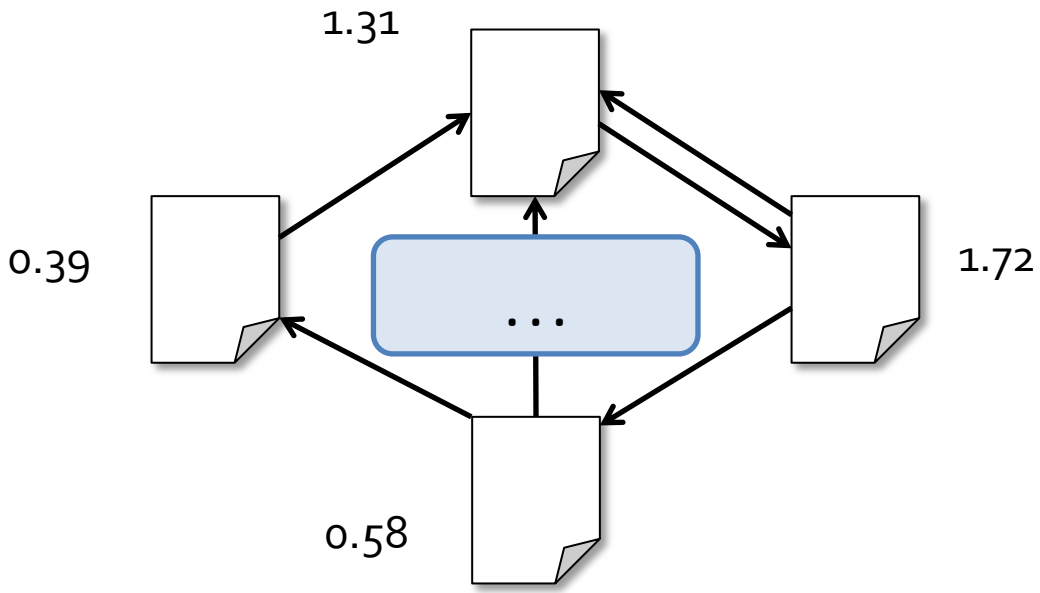


## PageRank算法

① 将每个页面的排序值初始化为1.0

② 在每次迭代中，对页面p，向其每个相邻页面（有直接链接的页面）发送一个值为rank(p)/numNeighbors(p)的贡献值

③ 将每个页面的排序值设为0.15 + 0.85 * contributionsReceived

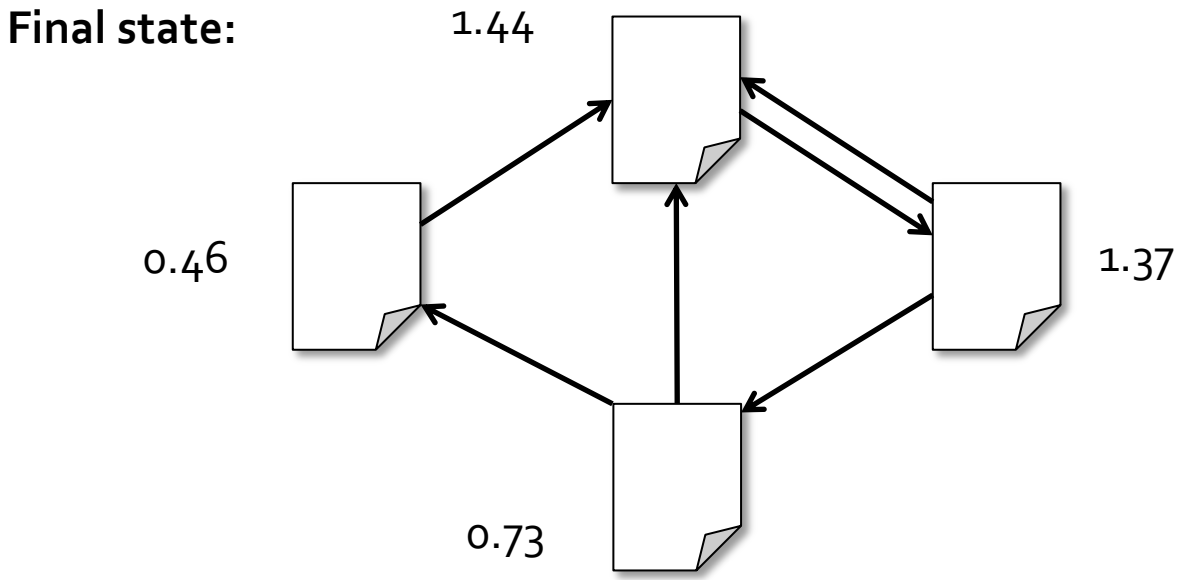


## PageRank算法

① 将每个页面的排序值初始化为1.0

② 在每次迭代中，对页面p，向其每个相邻页面（有直接链接的页面）发送一个值为rank(p)/numNeighbors(p)的贡献值

③ 将每个页面的排序值设为0.15 + 0.85 * contributionsReceived



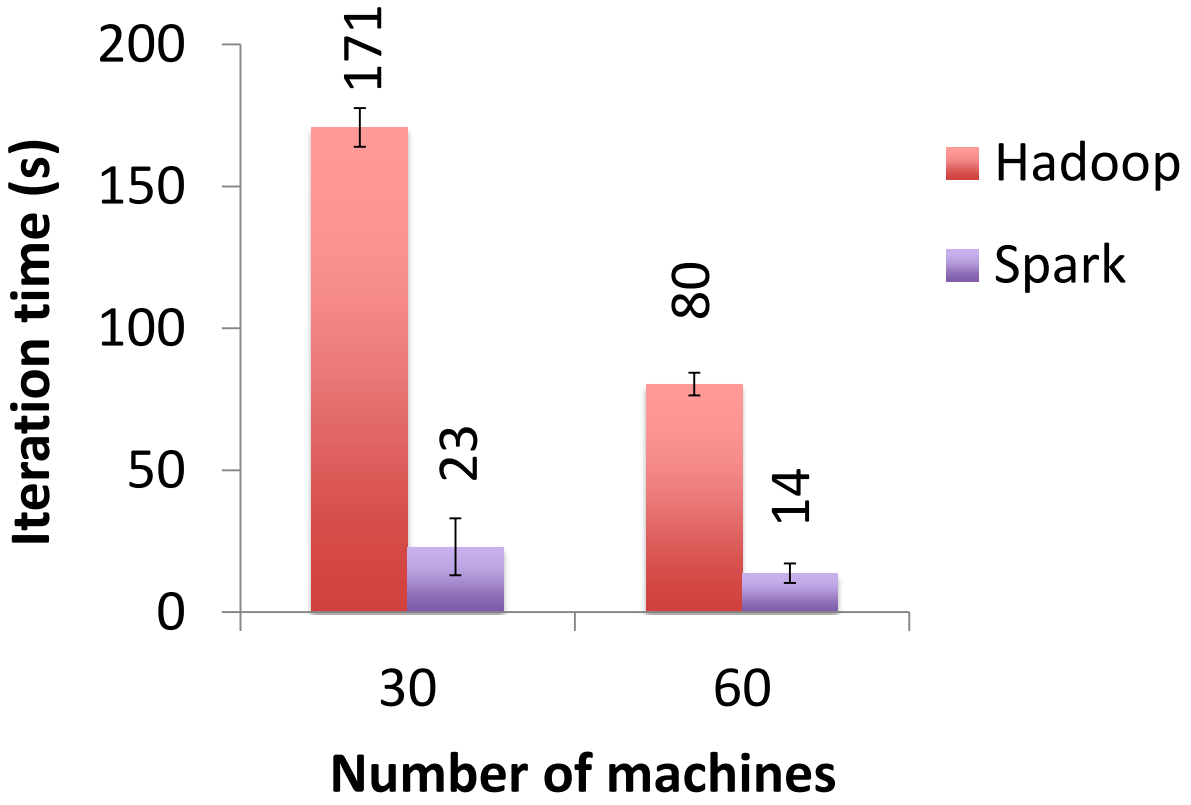
## PageRank的Scala语言实现

```
val sc = new SparkContext("local", "PageRank", sparkHome,
Seq("pagerank.jar"))

val links = // load RDD of (url, neighbors) pairs
var ranks = // load RDD of (url, rank) pairs

for (i <- 1 to ITERATIONS) {
  val contribs = links.join(ranks).flatMap {
    case (url, (links, rank)) =>
      links.map(dest => (dest, rank/links.size))
  }
  ranks = contribs.reduceByKey(_ + _)
                  .mapValues(0.15 + 0.85 * _)
}
ranks.saveAsTextFile(...)
```

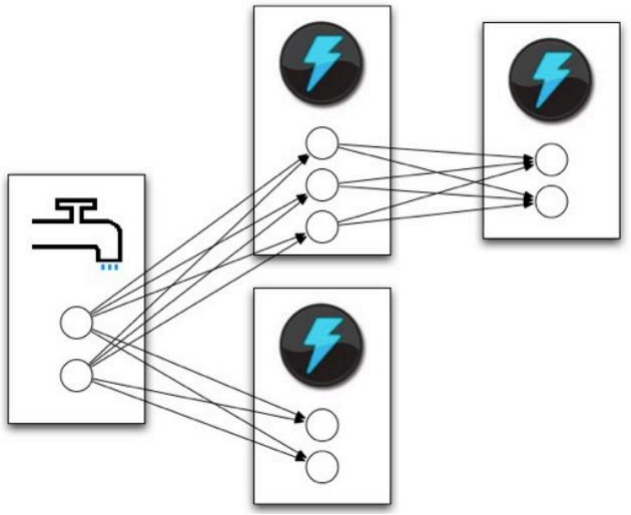
## PageRank性能测试




# 4 Storm

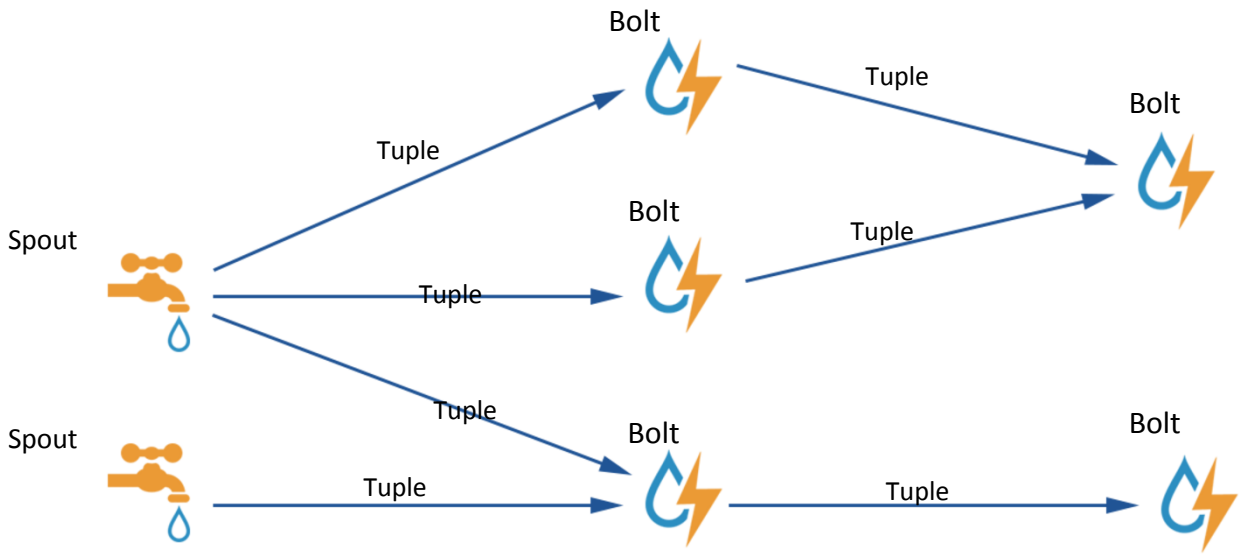
## Storm概述


- Storm是一个分布式实时流式计算平台
- 分布式
  - 水平扩展：通过加机器、提高并发数就提高处理能力
  - 自动容错：自动处理进程、机器、网络异常
- 实时：数据不写磁盘，延迟低（毫秒级）
- 流式：不断有数据流入、处理、流出
- 模型简单：水流模型
- 支持多种编程语言
- 开源：由Twitter开源，社区很活跃



## Storm计算模型

- Storm的现实模型就是一个水流的处理模型



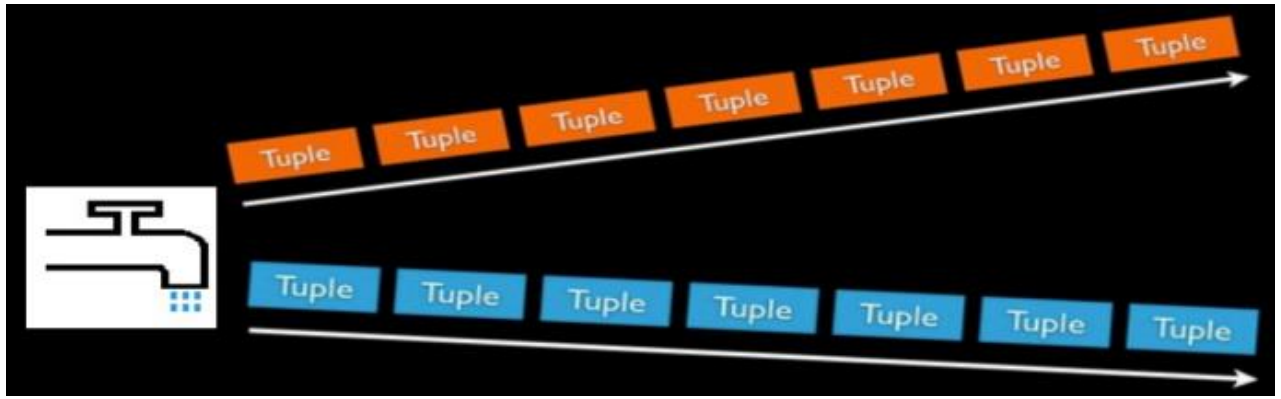
## Streams

- Streams：持续的Tuple流

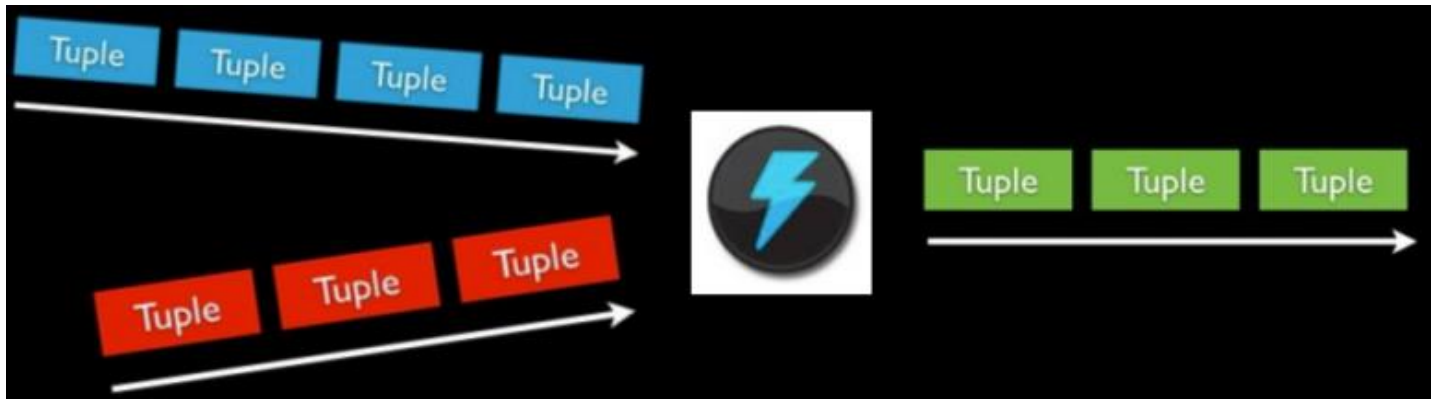

- Tuple：数据处理单元，一个Tuple由多个字段组成

| Field1 | Field2 | Field3 | Field4 |
|---|---|---|---|

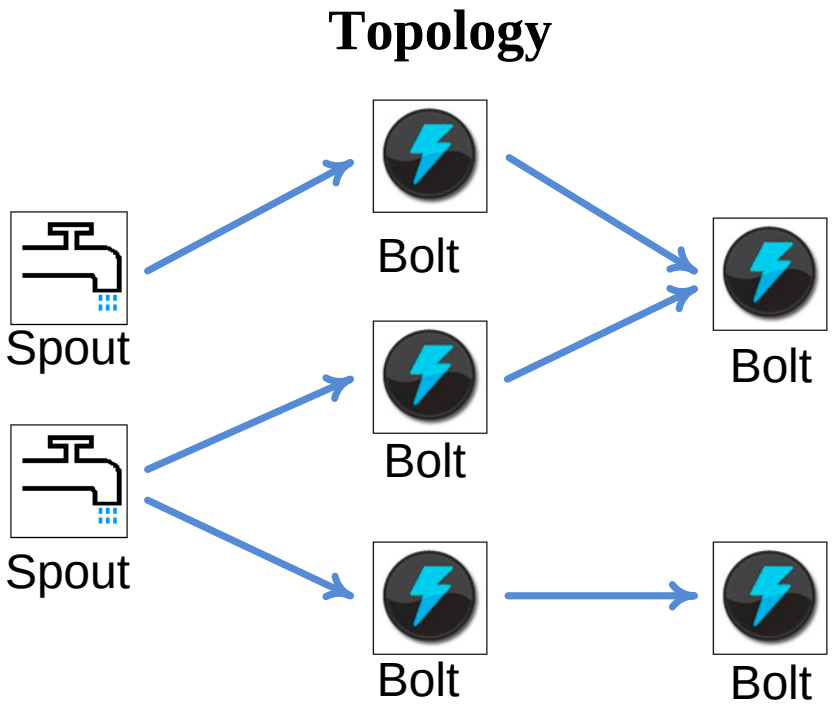
- **Spout**：Storm认为每个Stream都有一个源头，并把这个源头抽象为Spout



- **Bolt**：Storm将Streams的状态转换过程抽象为Bolt，Bolt接收Spout/Bolt输出的Tuple进行处理，处理后的Tuple作为新的Streams发送给其他Bolt

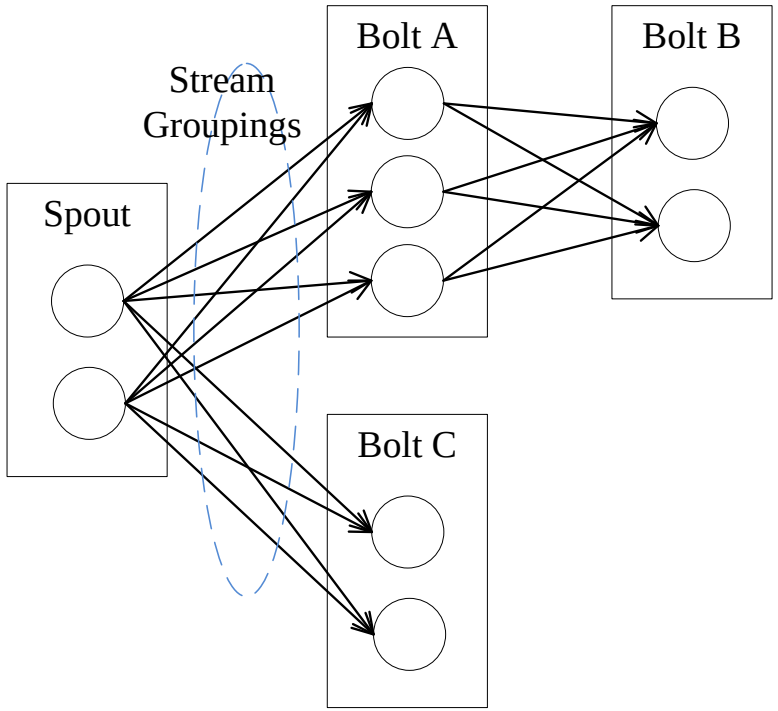


- **Topology**：一个应用的spout, bolt, grouping组合

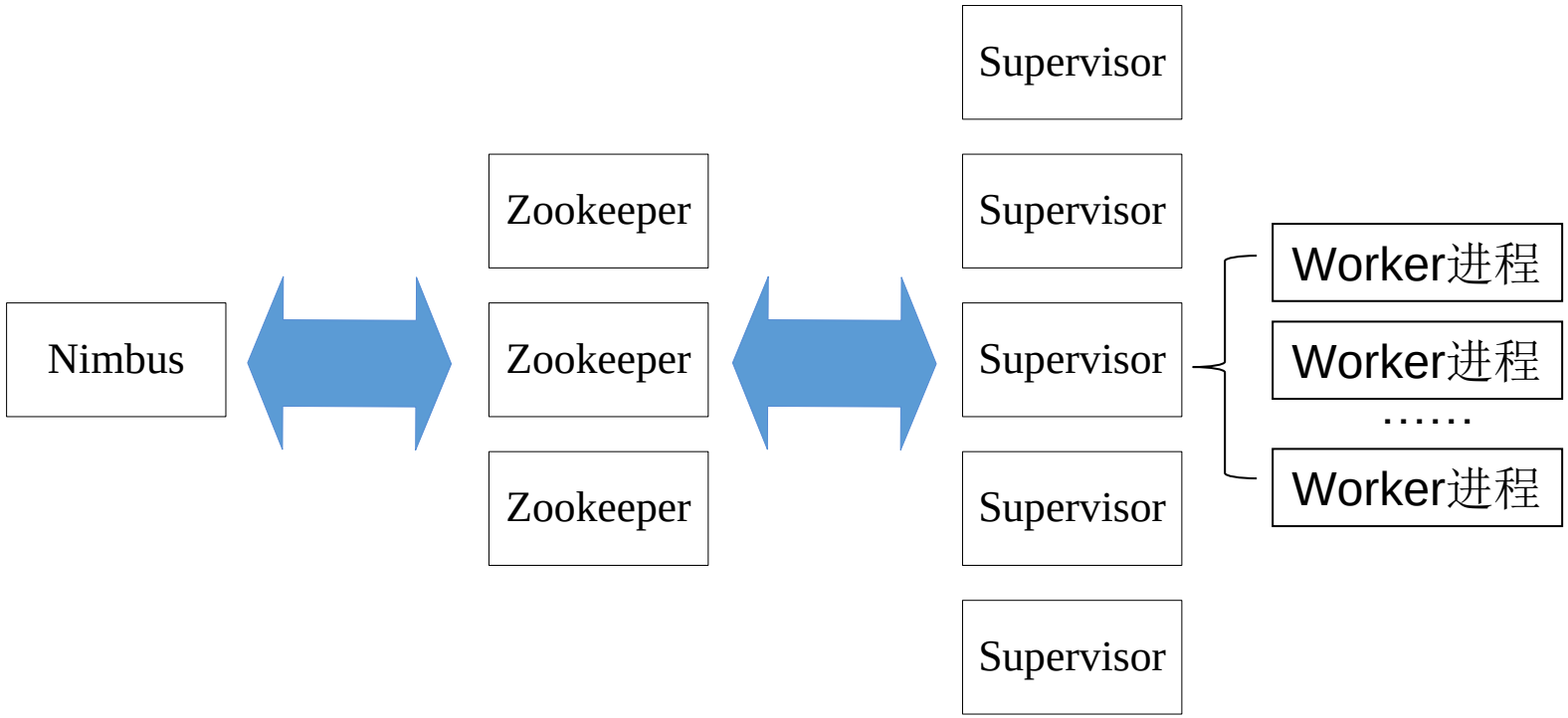


- **Stream Groupings**：用于告知Topology如何在两个组件间（如Spout和Bolt之间，或者不同的Bolt之间）进行Tuple的传送。每一个Spout和Bolt都可以有多个分布式任务，一个任务在什么时候、以什么方式发送Tuple就是由Stream Groupings来决定的

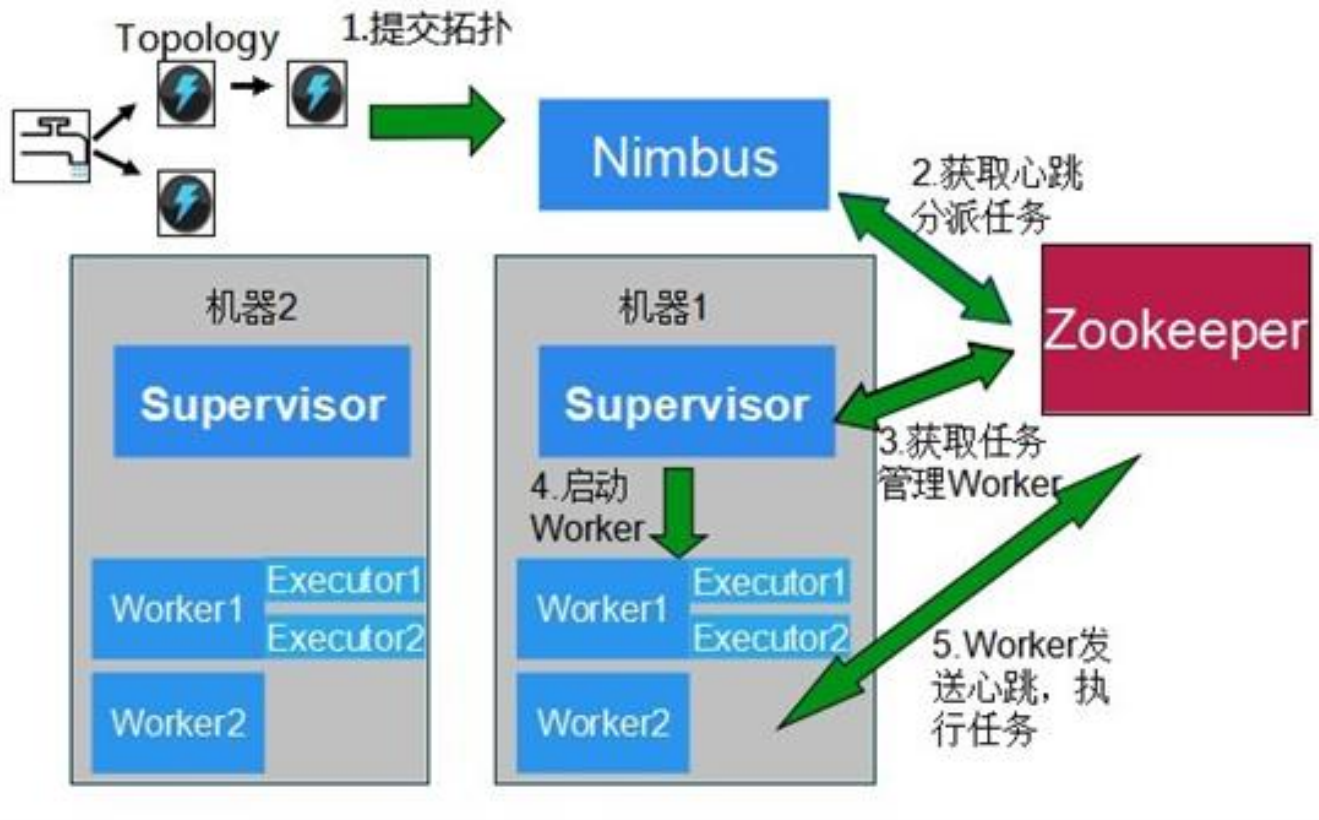
  - Shuffle Grouping
  - Fields Grouping
  - All Grouping
  - Global Grouping
  - Direct Grouping
  - Local Grouping



## Storm系统架构


Nimbus
Zookeeper
Zookeeper
Zookeeper
Supervisor
Supervisor
Supervisor
Supervisor
Supervisor
Worker进程
Worker进程
……
Worker进程

## Storm工作流程


Topology
1.提交拓扑
Nimbus
2.获取心跳
分派任务
Zookeeper
机器2
Supervisor
Worker1
Executor1
Executor2
Worker2
机器1
Supervisor
3.获取任务
管理Worker
4.启动
Worker
Worker1
Executor1
Executor2
Worker2
5.Worker发
送心跳，执
行任务


# 5 Spark Streaming

## Spark Streaming概述

- Spark Streaming是Spark core API的扩展，支持实时数据流的处理，并且具有可扩展，高吞吐量，容错的特点
- Spark Streaming能够和Spark的其他模块无缝集成，形成适用于批处理和流处理的统一框架（编程模型）
- 能够接收多种数据源的数据

## Discretized Stream (DStream) Processing

- 把流式计算转化为一系列微小数据块的批处理计算

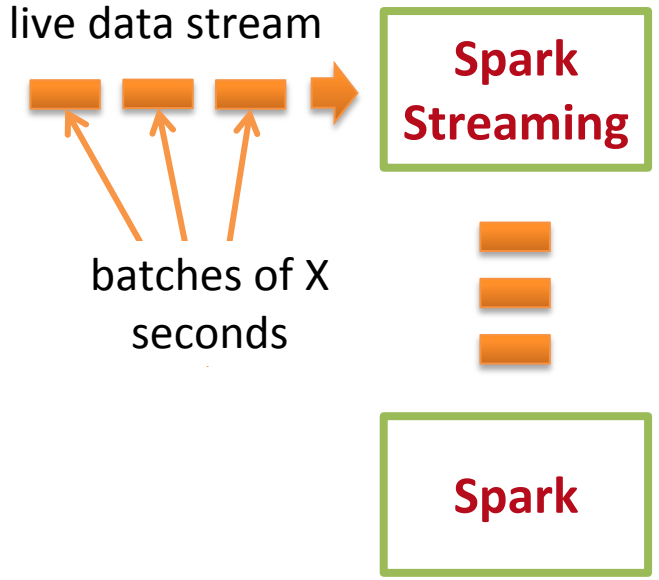
## Discretized Stream (DStream) Processing

- 把流式计算转化为一系列微小数据块的批处理计算

## Discretized Stream (DStream) Processing

- 把流式计算转化为一系列微小数据块的批处理计算

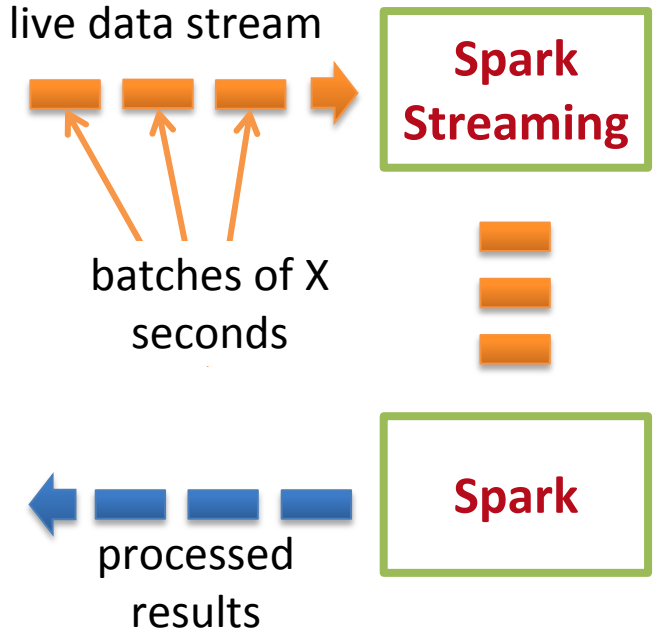
  - 把实时输入数据流以时间片（如X秒）为单位切分成块



## Discretized Stream (DStream) Processing

- 把流式计算转化为一系列微小数据块的批处理计算

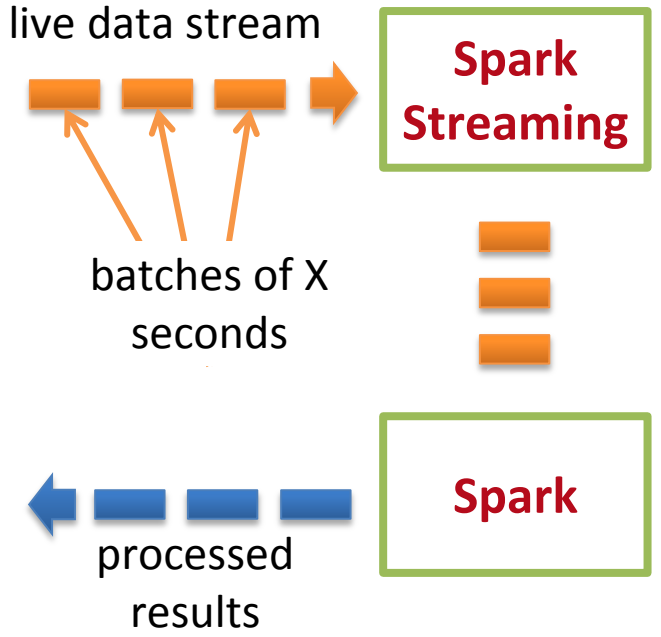
- 把实时输入数据流以时间片（如X秒）为单位切分成块
- 把每块数据作为一个RDD，使用RDD操作处理每一小块数据
- RDD操作的结果也以一小块一小块的形式返回



## Discretized Stream (DStream) Processing

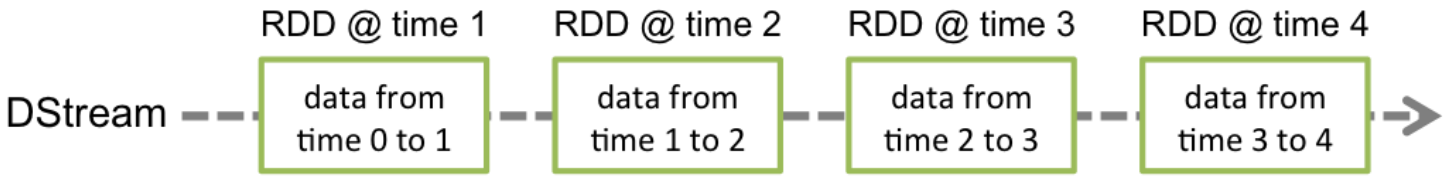
- 把流式计算转化为一系列微小数据块的批处理计算

  - 数据块时间片大小可低至 ½ 秒, 延迟大约为1 秒

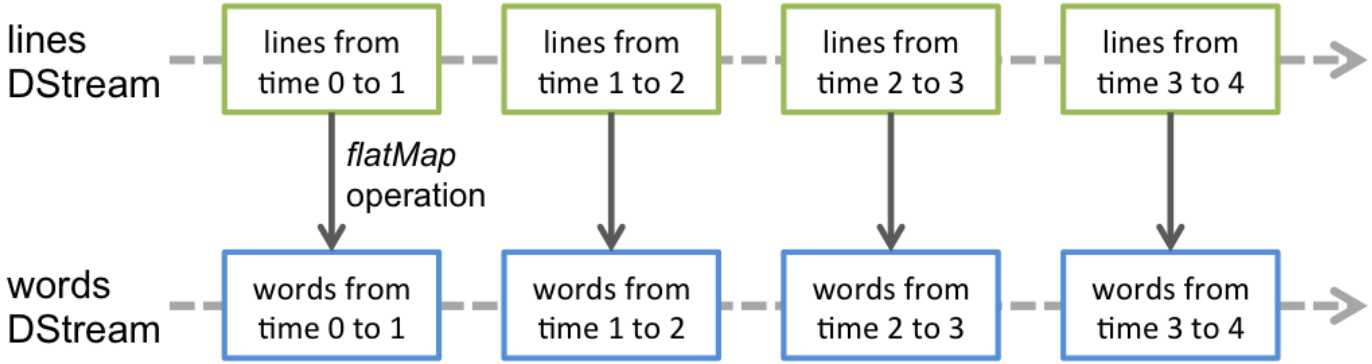


## Discretized Stream (DStream) Processing

- DStream内部是由一系列连续的RDD组成的，每个RDD都包含了特定时间间隔内的一批数据

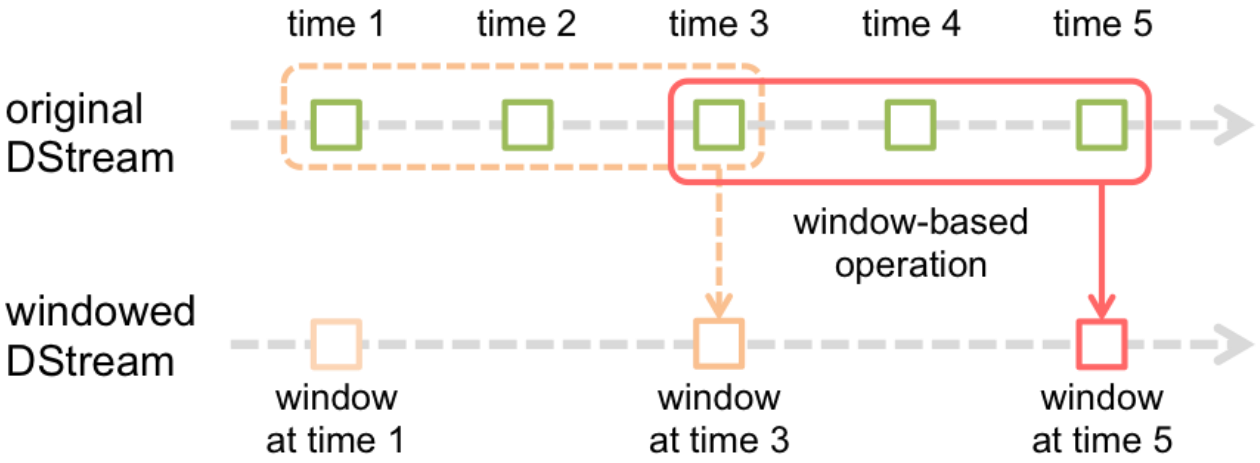



- 任何作用于DStream的算子，其实都会被转化为对其内部RDD的操作



## Discretized Stream (DStream) Processing

- Spark Streaming提供基于时间窗口的计算，也就是说，可以对某一个滑动时间窗内的数据施加特定tranformation




- 每隔10秒统计一下前30秒内的单词计数

```
val windowedWordCounts = pairs.reduceByKeyAndWindow((a:Int,b:Int)
=> (a + b), Seconds(30), Seconds(10))
```


北京理工大學
BEIJING INSTITUTE OF TECHNOLOGY
谢谢!
德以明理 学以精工