大数据基础与应用
BIG DATA FOUNDATION AND APPLICATION

大数据开源平台与工具

北京理工大学计算机学院　王一拙

2019年1月

内容提要

- 数据采集与清洗
- 数据存储与管理
- 数据处理与分析
- 资源管理与调度

3. 数据处理与分析

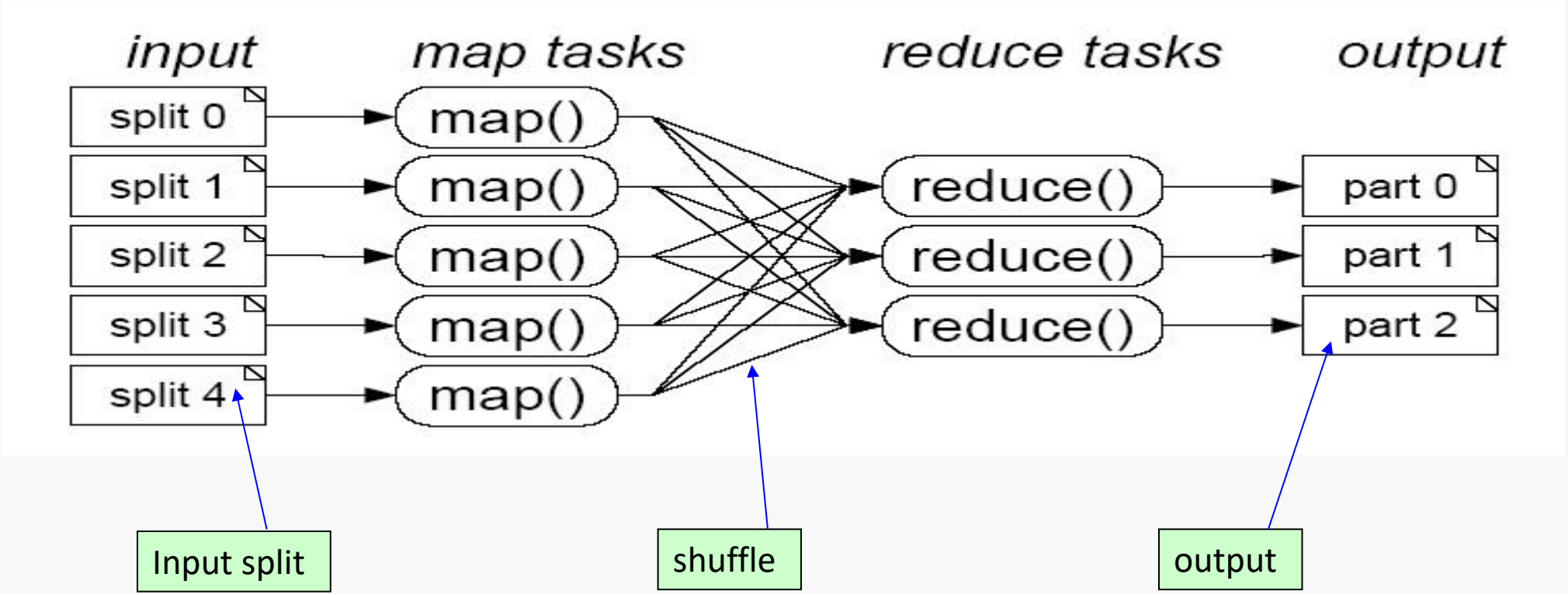
- 批处理
  - MapReduce
  - Spark
- 流处理
  - Storm
  - Spark Streaming

MapReduce

- 概述和编程实例
- 架构和基本运行原理

Mapreduce概述

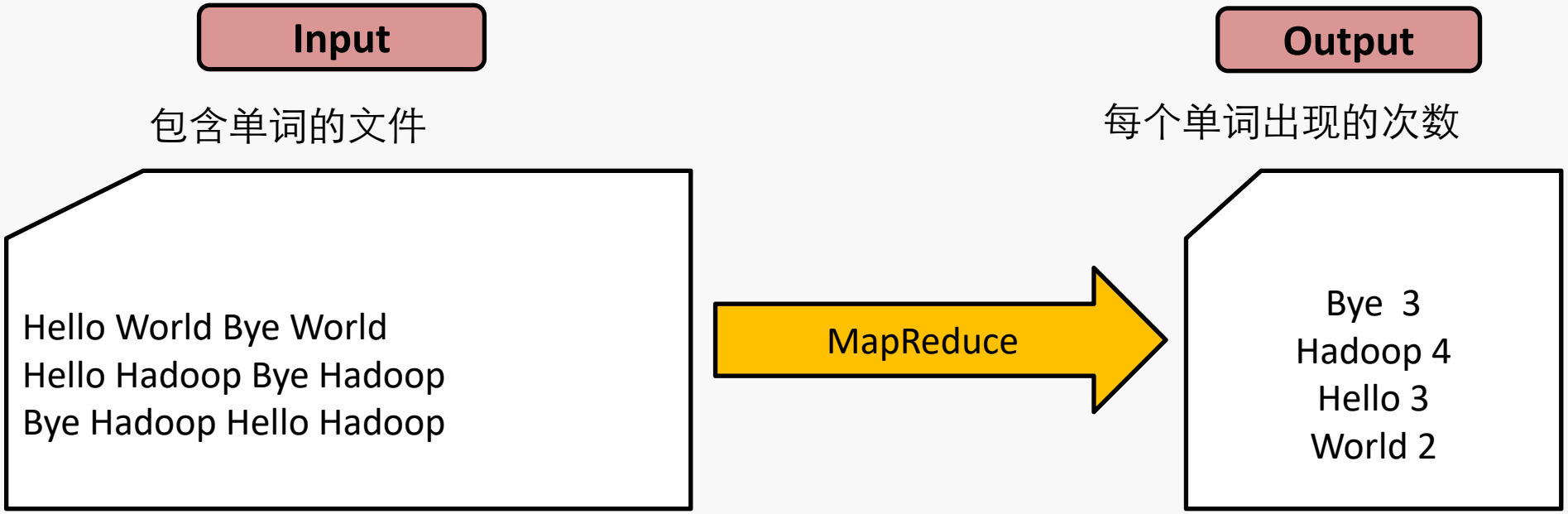
- Parallel/Distributed Computing Programming Model



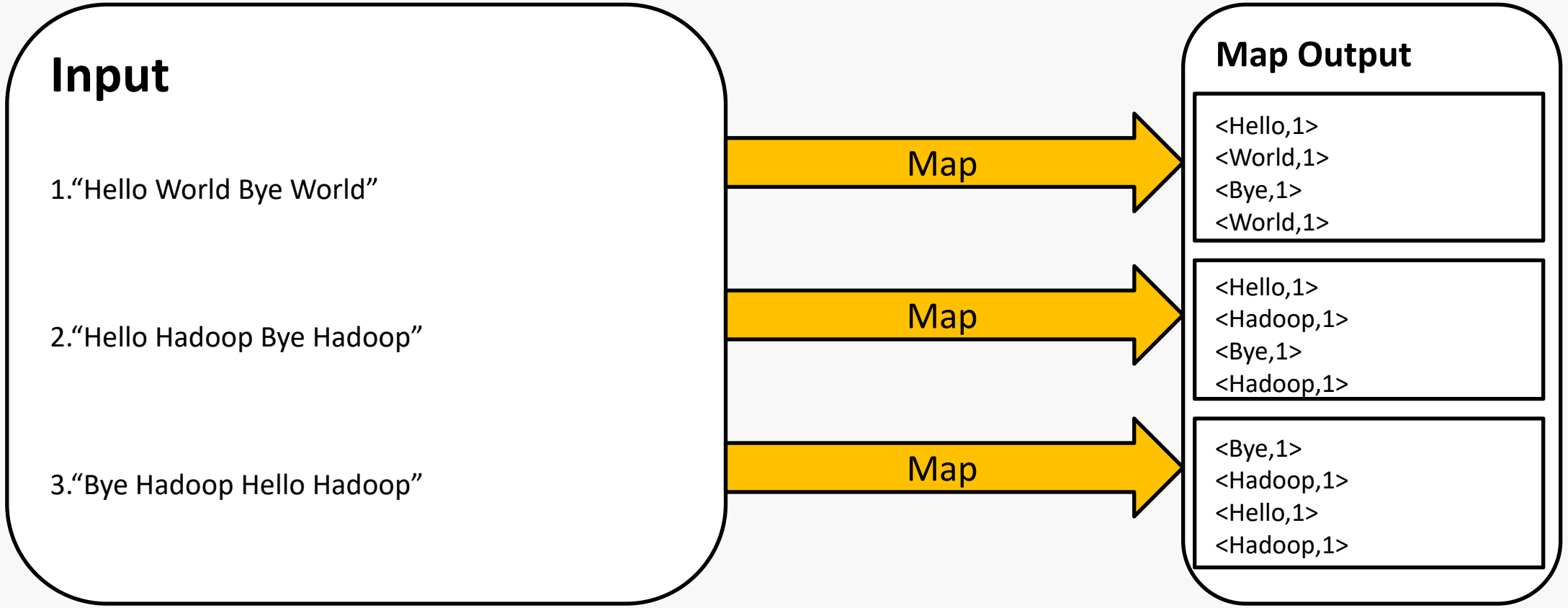
MapReduce解决问题的基本过程

- 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

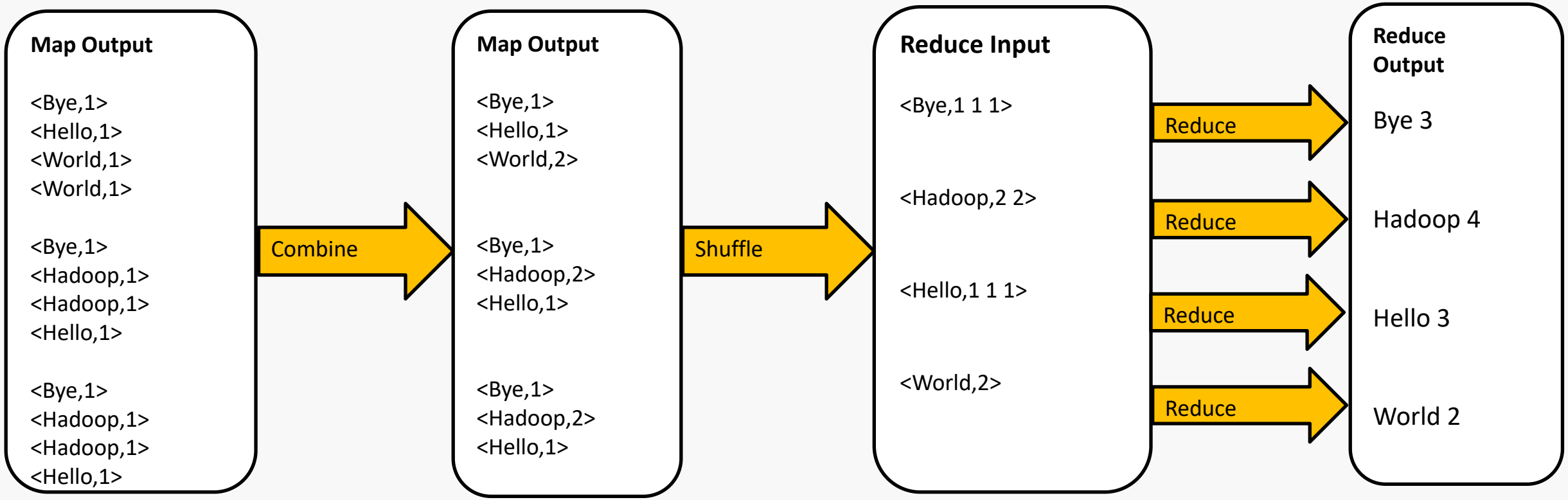
WordCount程序功能



WordCount 的Map过程



WordCount的Reduce过程



MapReduce示例：单词计数

- 使用MapReduce求解该问题
  - 定义Map和Reduce函数

```
Map(K,V){
    For each word w in V
        Collect(w , 1);
}
Reduce(K,V[]){
    int count = 0;
    For each v in V
        count += v;
    Collect(K , count);
}
```

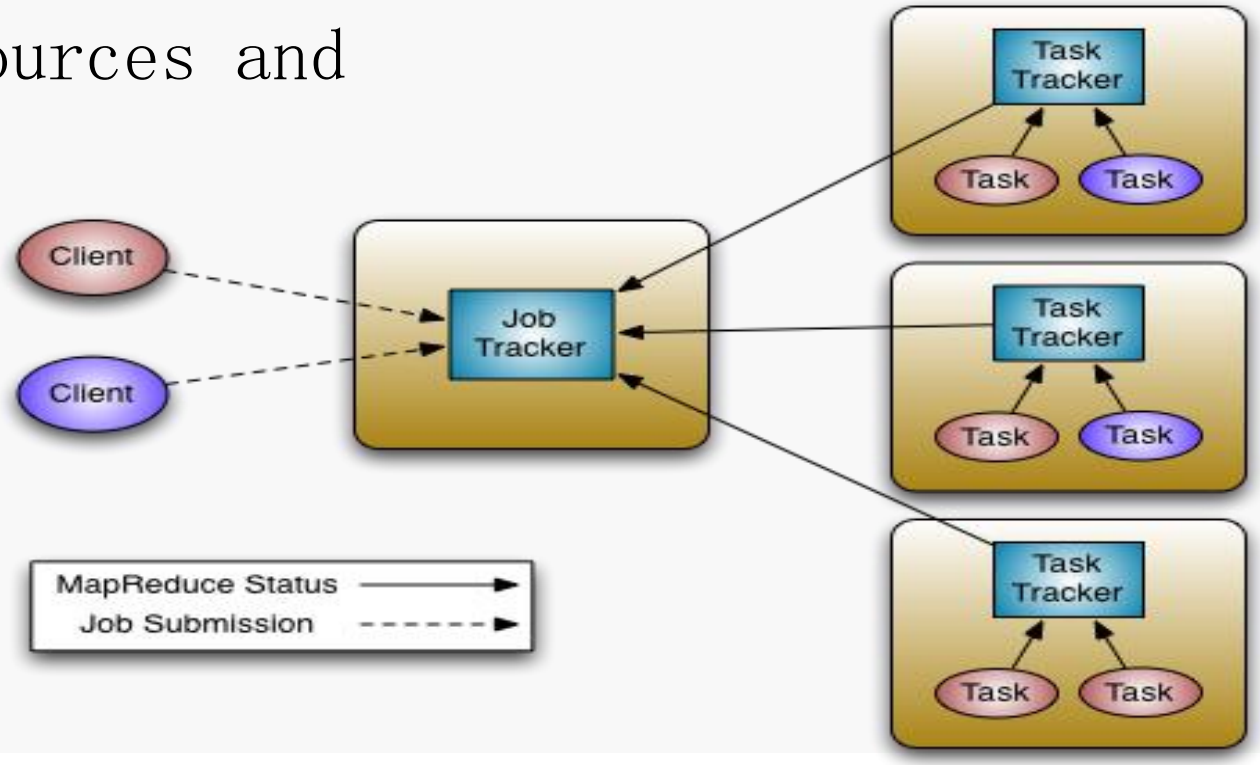
编程实例演示

MapReduce

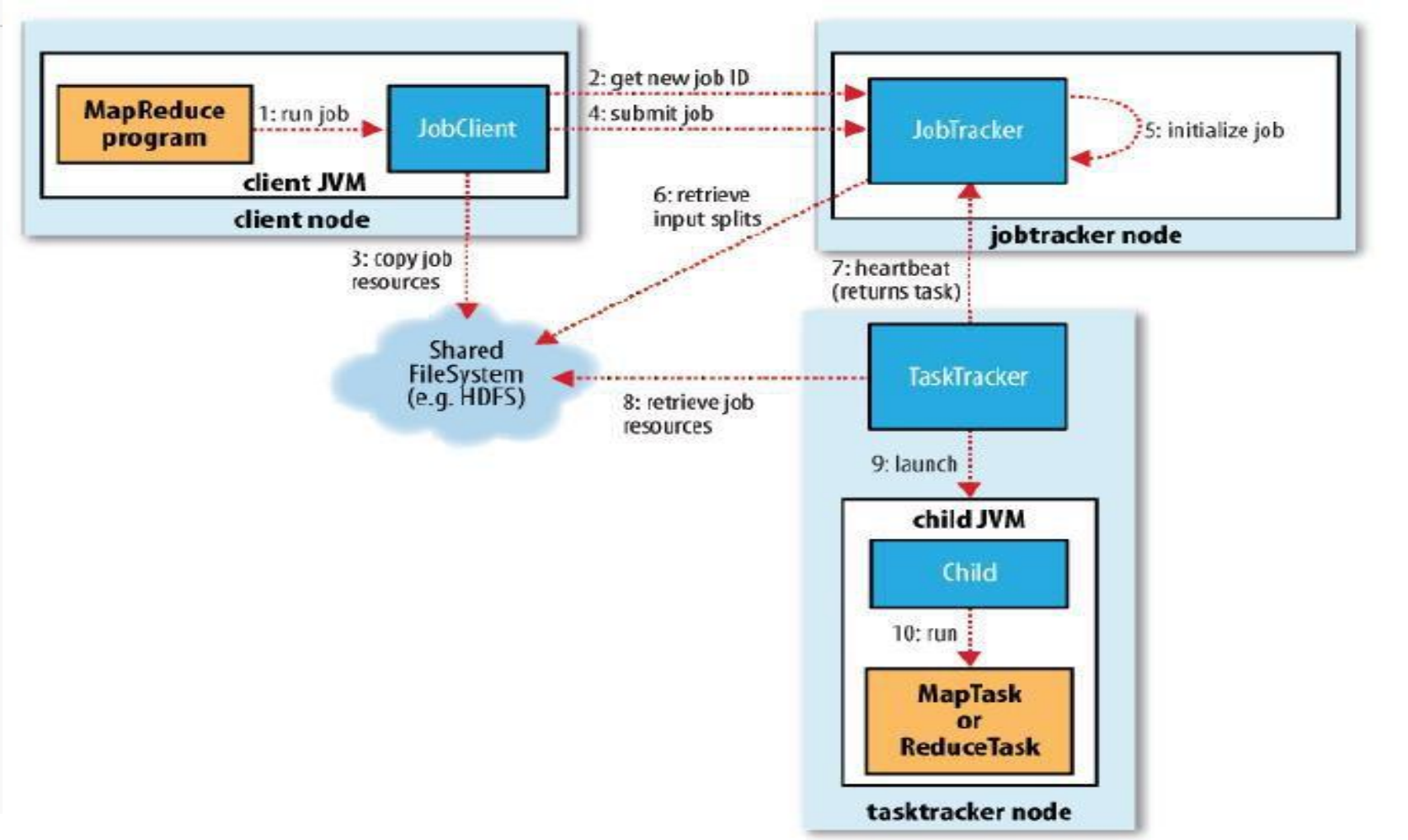
- 概述和编程实例
- 架构和基本运行原理

Hadoop MapReduce 1.0

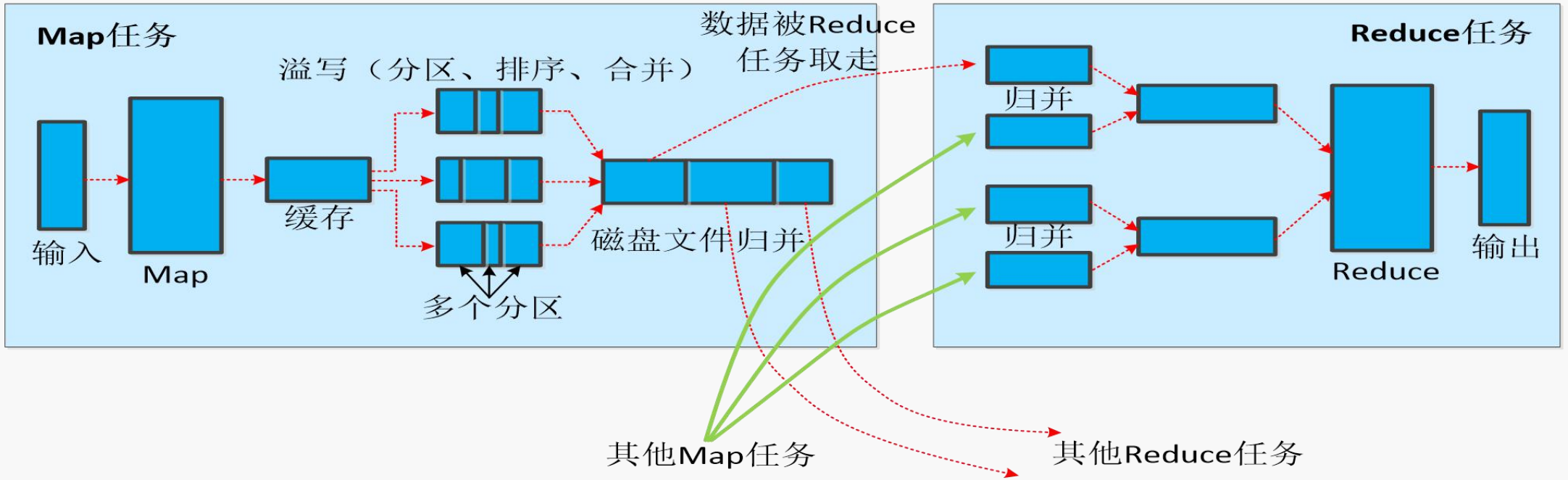
- JobTracker
  - Manages cluster resources and job scheduling
- TaskTracker
  - Per-node agent
  - Manage tasks



MapReduce作业运行过程



Shuffle过程



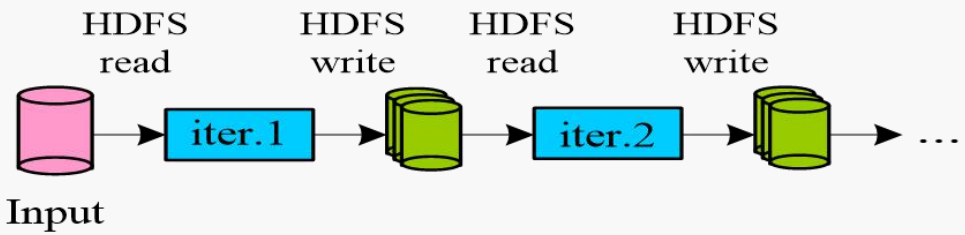
来源：厦门大学-林子雨-《大数据技术原理与应用》教材

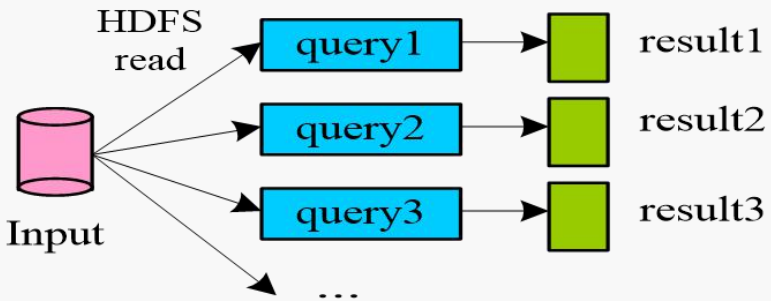
Spark

- 概述
- 架构和原理
- 编程实例

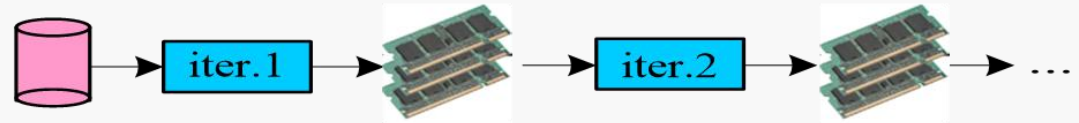
Spark产生的动机

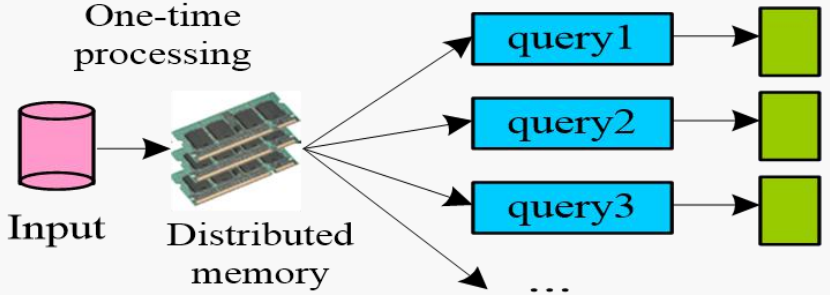
适用场景





Data Sharing in MapReduce

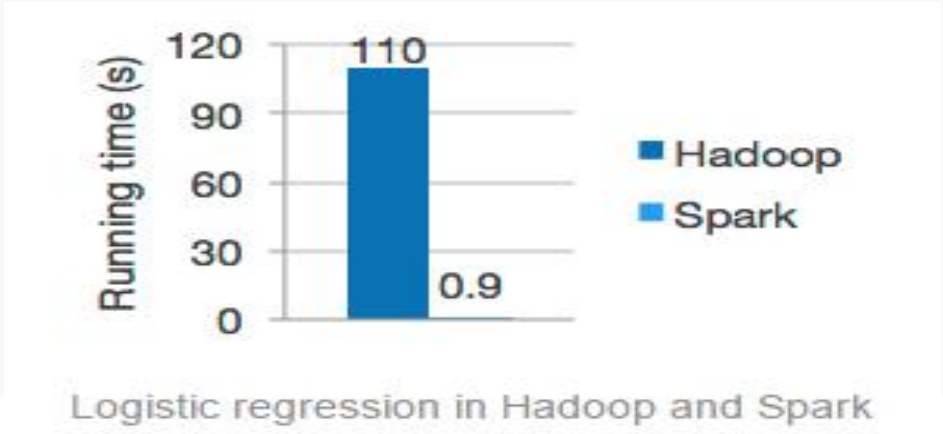




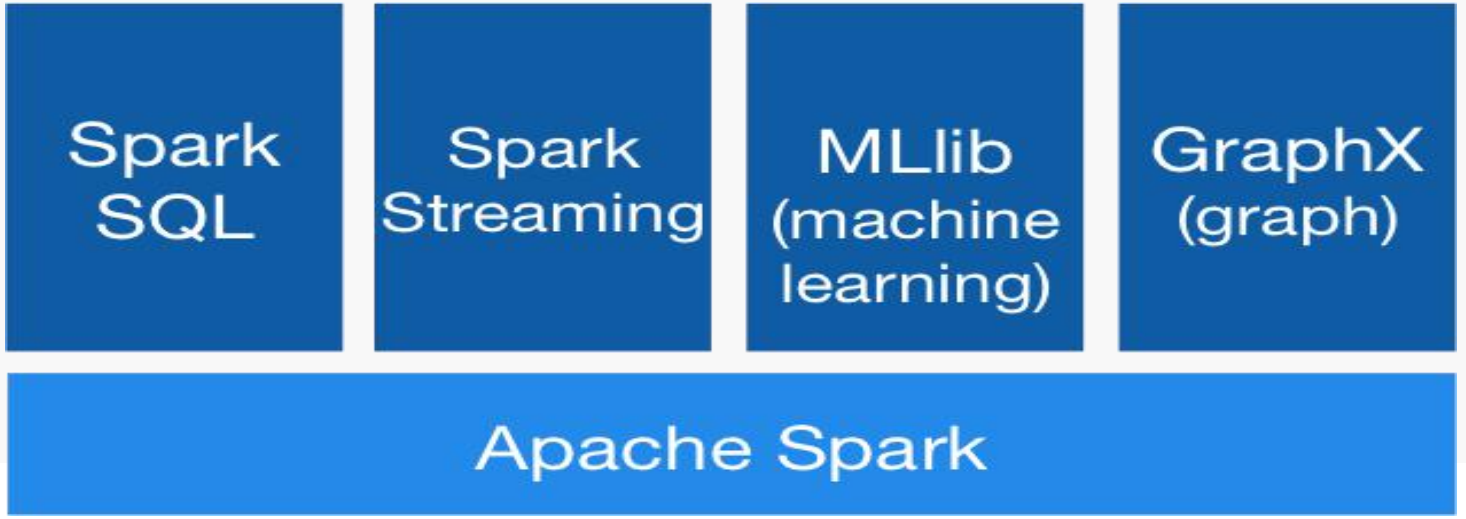
Data Sharing in Spark

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。



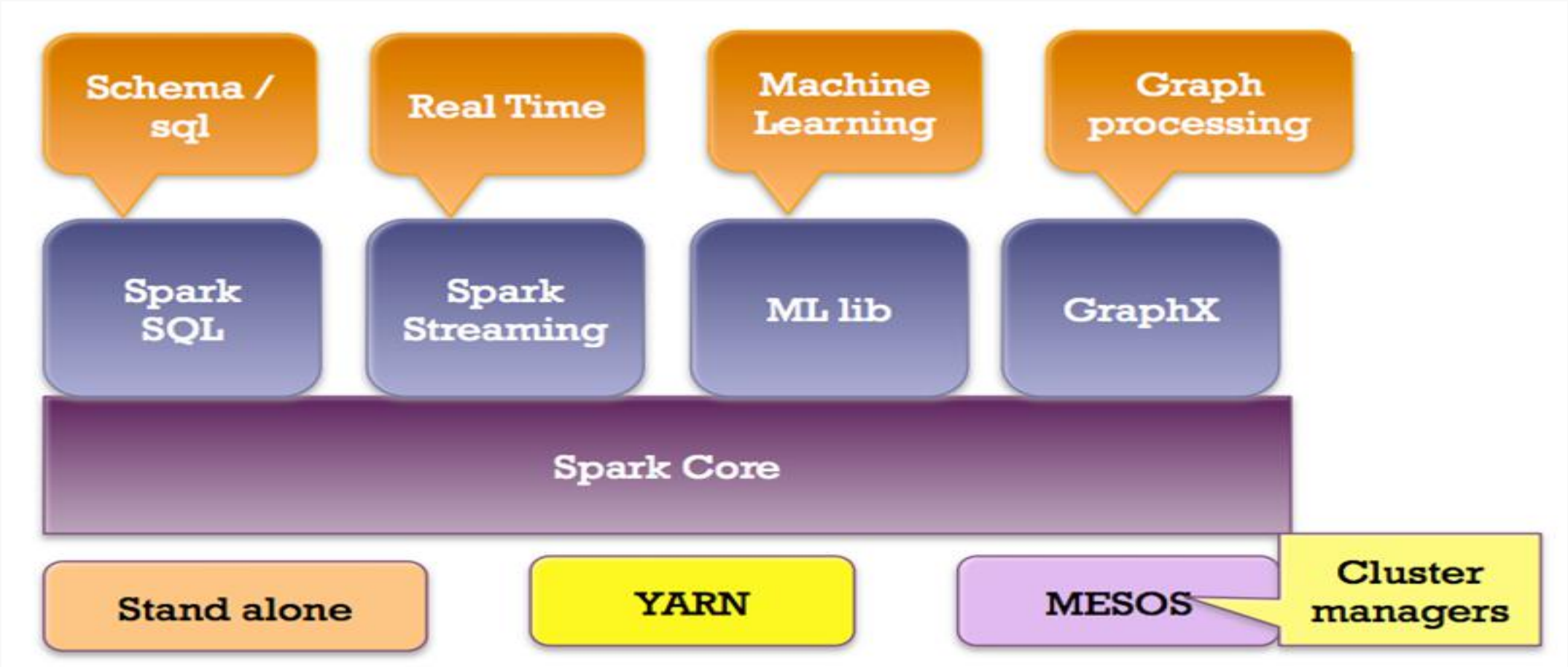
Logistic regression in Hadoop and Spark



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


Schema / sql
Real Time
Machine Learning
Graph processing
Spark SQL
Spark Streaming
ML lib
GraphX
Spark Core
Stand alone
YARN
MESOS
Cluster managers

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

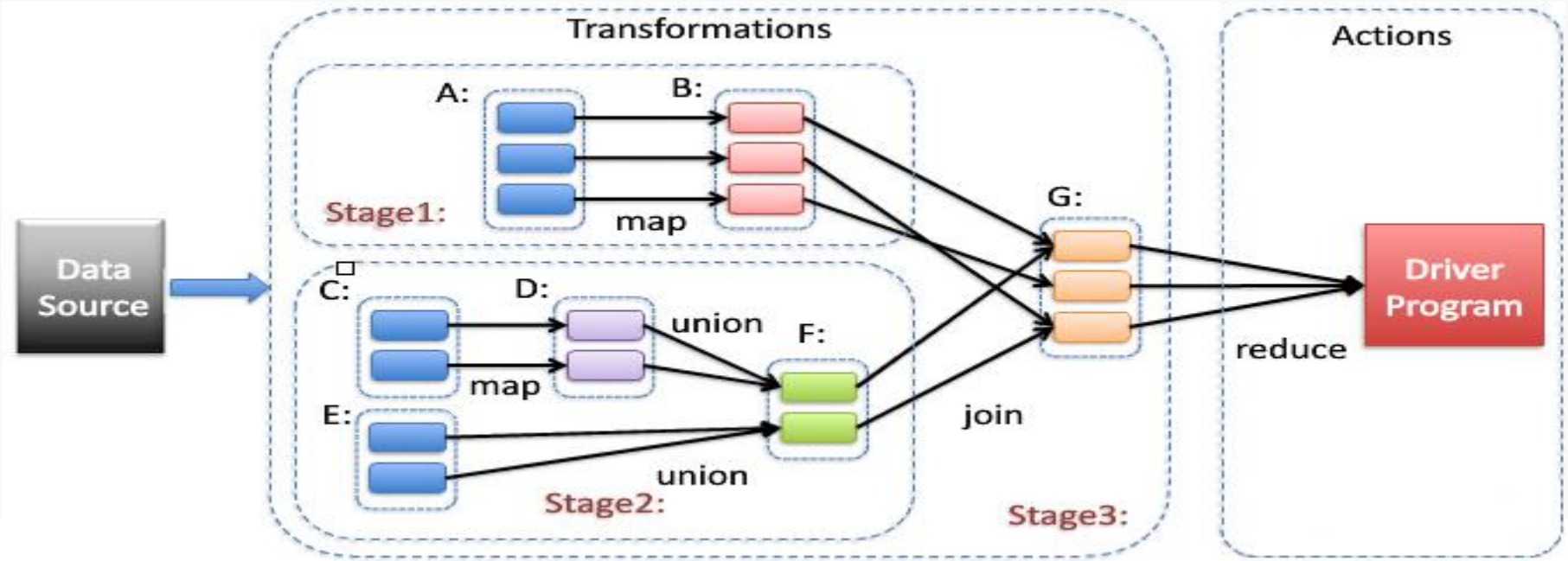
- RDD：是Resillient Distributed Dataset（弹性分布式数据集）的简称，是分布式内存的一个抽象概念，提供了一种高度受限的共享内存模型
- DAG：是Directed Acyclic Graph（有向无环图）的简称，反映RDD之间的依赖关系
- Executor：是运行在工作节点（Worker Node）上的一个进程，负责运行Task
- Application：用户编写的Spark应用程序
- Task：运行在Executor上的工作单元
- Job：一个Job包含多个RDD及作用于相应RDD上的各种操作
- Stage：是Job的基本调度单位，一个Job会分为多组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

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

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

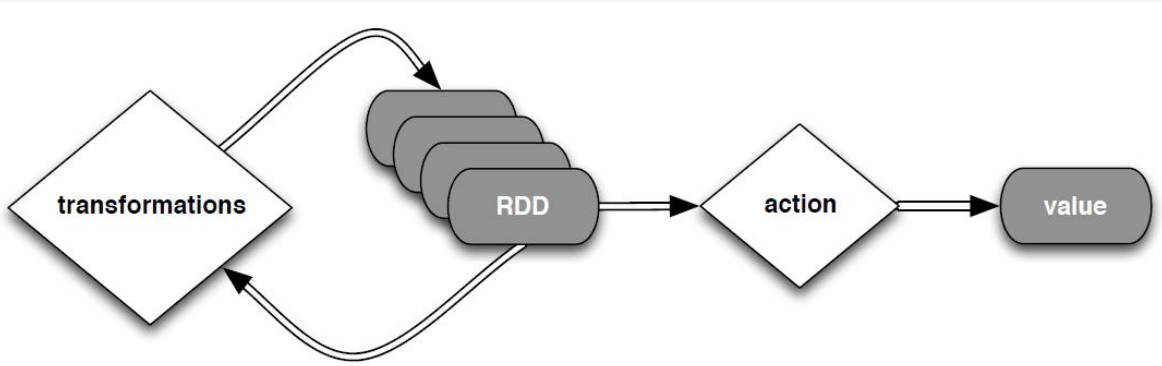
Spark Transformation & Action

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*)

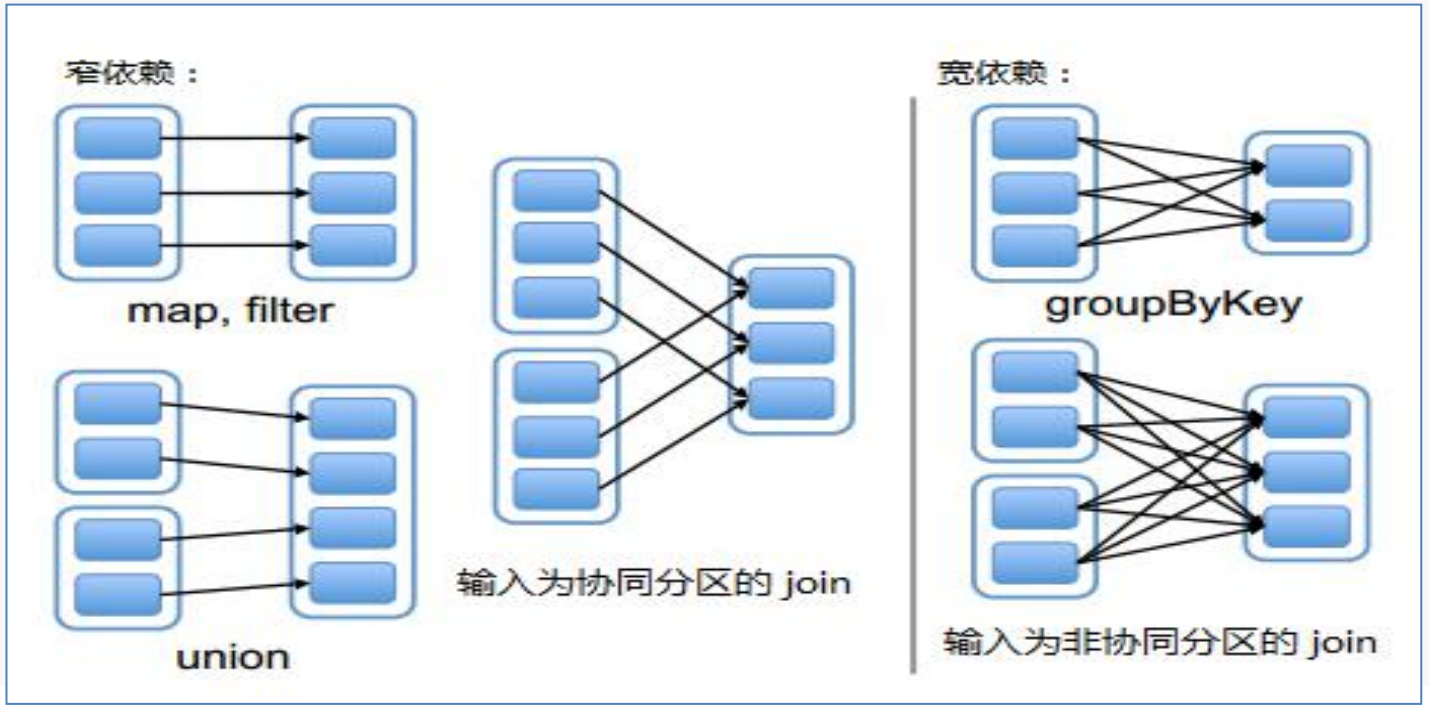


RDD的依赖关系

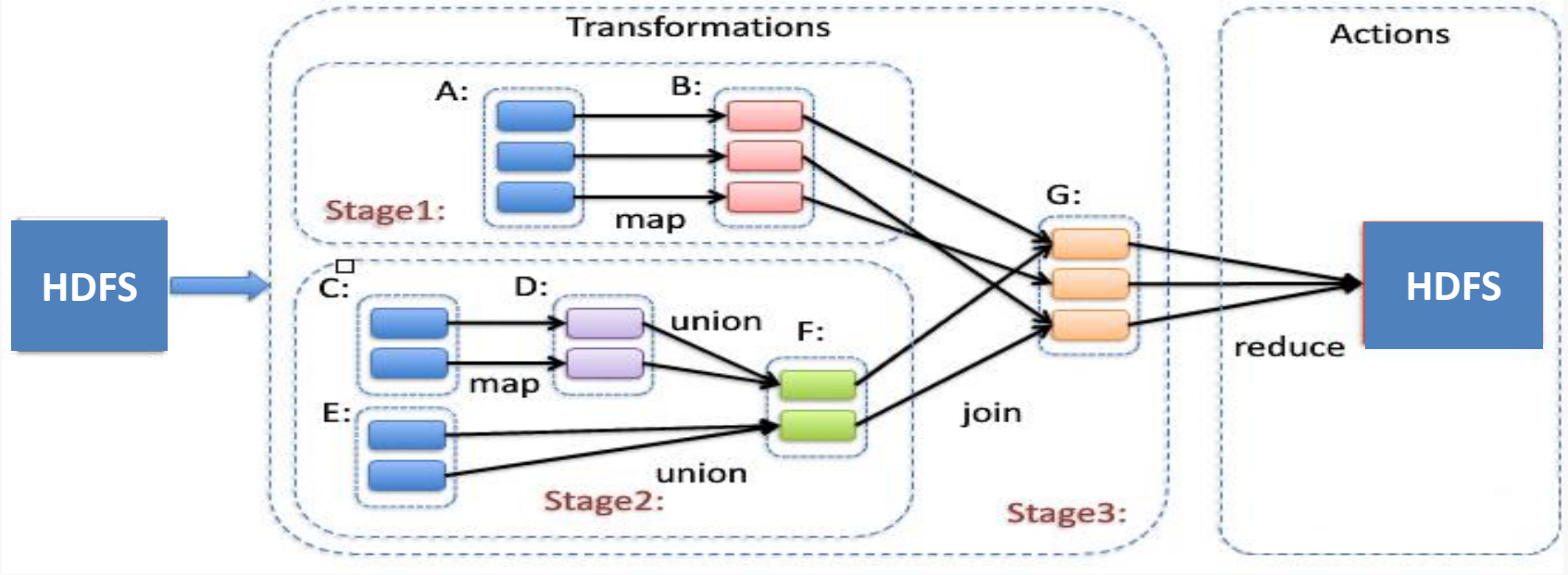
RDD之间的依赖关系可以分为两类，即：

（1）**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）；

（2）**宽依赖**：子RDD的每个分区依赖于所有父RDD分区。

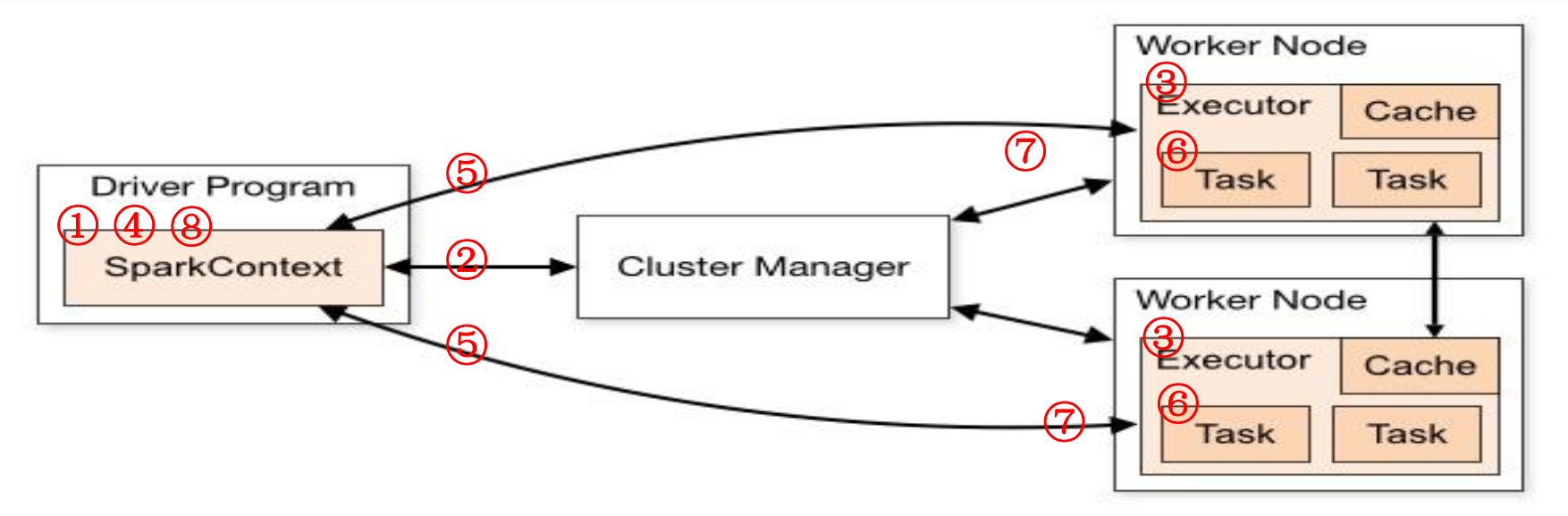


Spark中的Stage划分



- 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

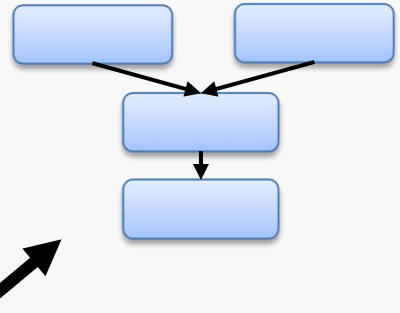
Spark运行架构



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

Job scheduling

RDD Objects

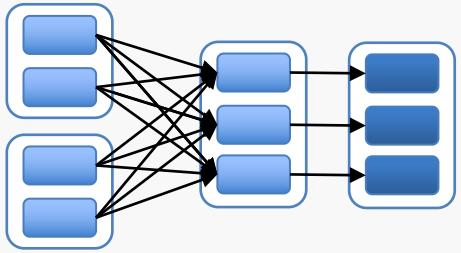


```
rdd1.join(rdd2)
    .groupBy(…)
    .filter(…)
```

build operator DAG

DAG

DAGScheduler

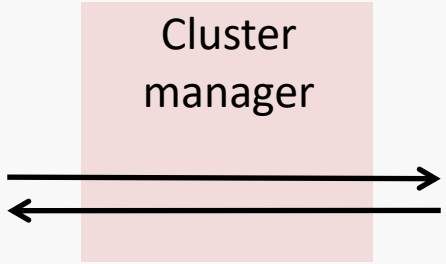


split graph into *stages* of tasks

submit each stage as ready

TaskSet

TaskScheduler

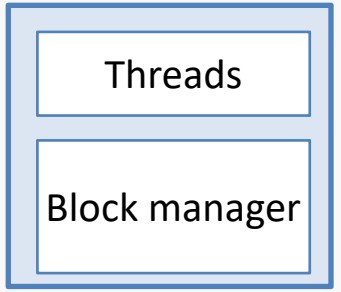


launch tasks via cluster manager

retry failed or straggling tasks

Task

Worker



execute tasks

store and serve blocks

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

Example: PageRank

- Good example of a more complex algorithm
  - Multiple stages of map & reduce
- Benefits from Spark's in-memory caching
  - Multiple iterations over the same data

http://ampcamp.berkeley.edu/wp-content/uploads/2014/02/Strata-Intro-to-Spark.pptx

Basic Idea

- Give pages ranks (scores) based on links to them
  - Links from many pages ➔ high rank
  - Link from a high-rank page ➔ high rank

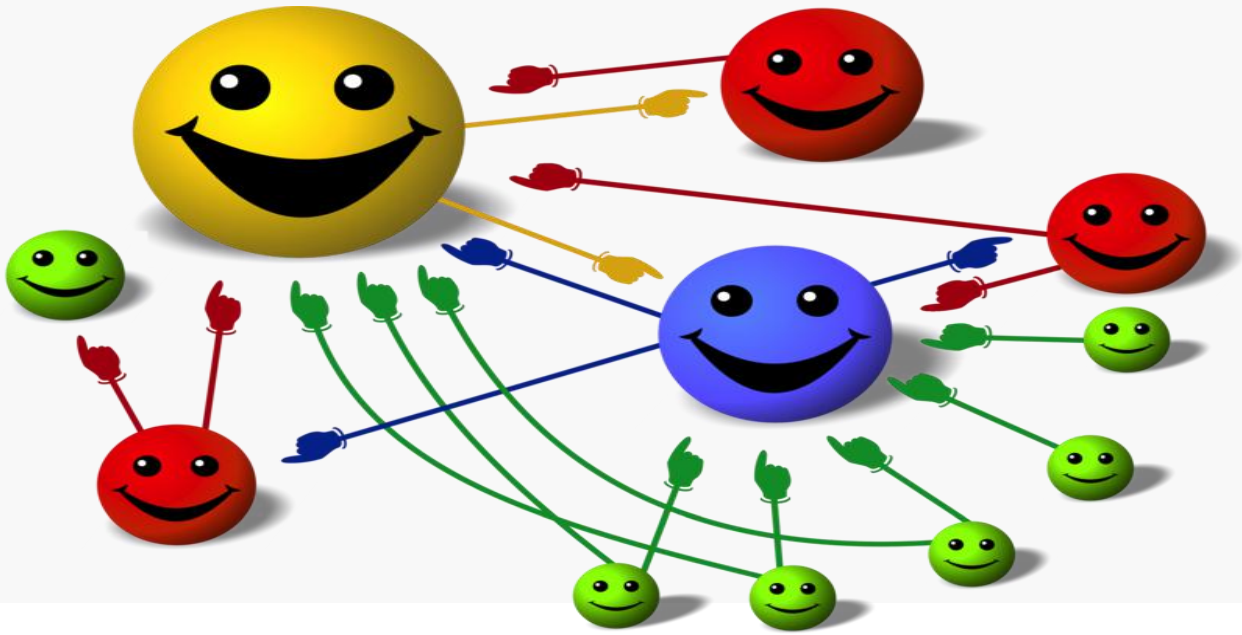
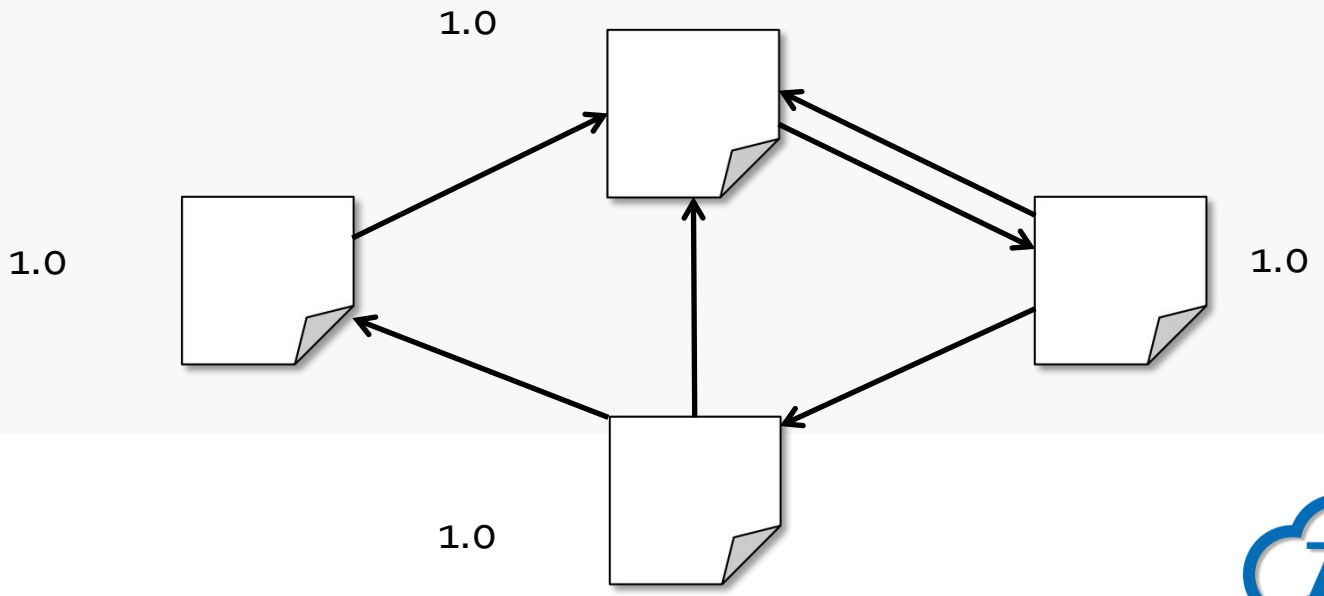


Image: en.wikipedia.org/wiki/File:PageRank-hi-res-2.png

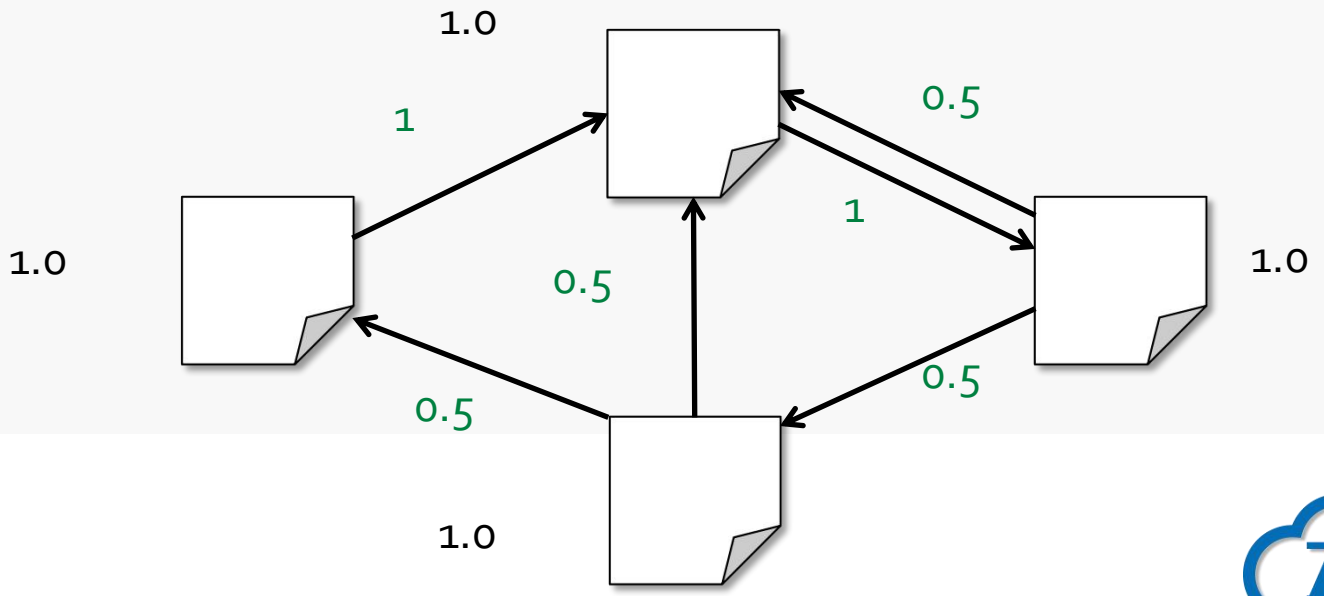
Algorithm

1. Start each page at a rank of 1
2. On each iteration, have page p contribute $rank_p$ / $|neighbors_p|$ to its neighbors
3. Set each page's rank to 0.15 + 0.85 × contribs



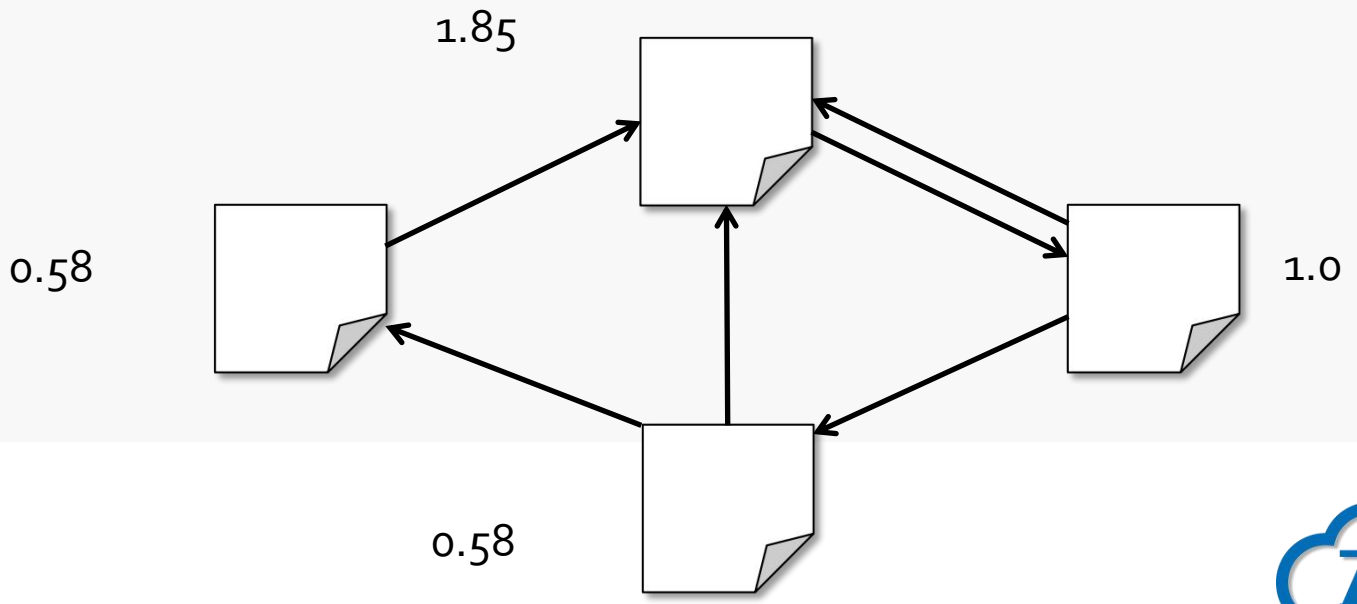
Algorithm

1. Start each page at a rank of 1
2. On each iteration, have page p contribute $rank_p$ / $|neighbors_p|$ to its neighbors
3. Set each page's rank to 0.15 + 0.85 × contribs



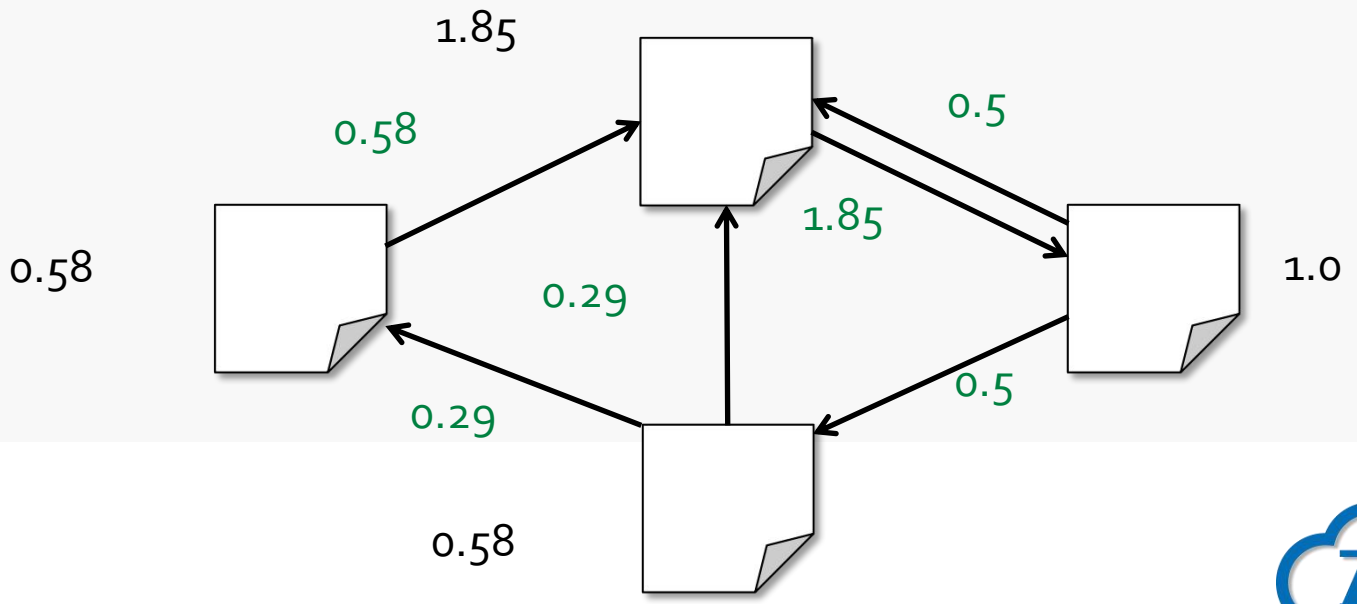
Algorithm

1. Start each page at a rank of 1
2. On each iteration, have page p contribute $rank_p$ / $|neighbors_p|$ to its neighbors
3. Set each page's rank to 0.15 + 0.85 × contribs



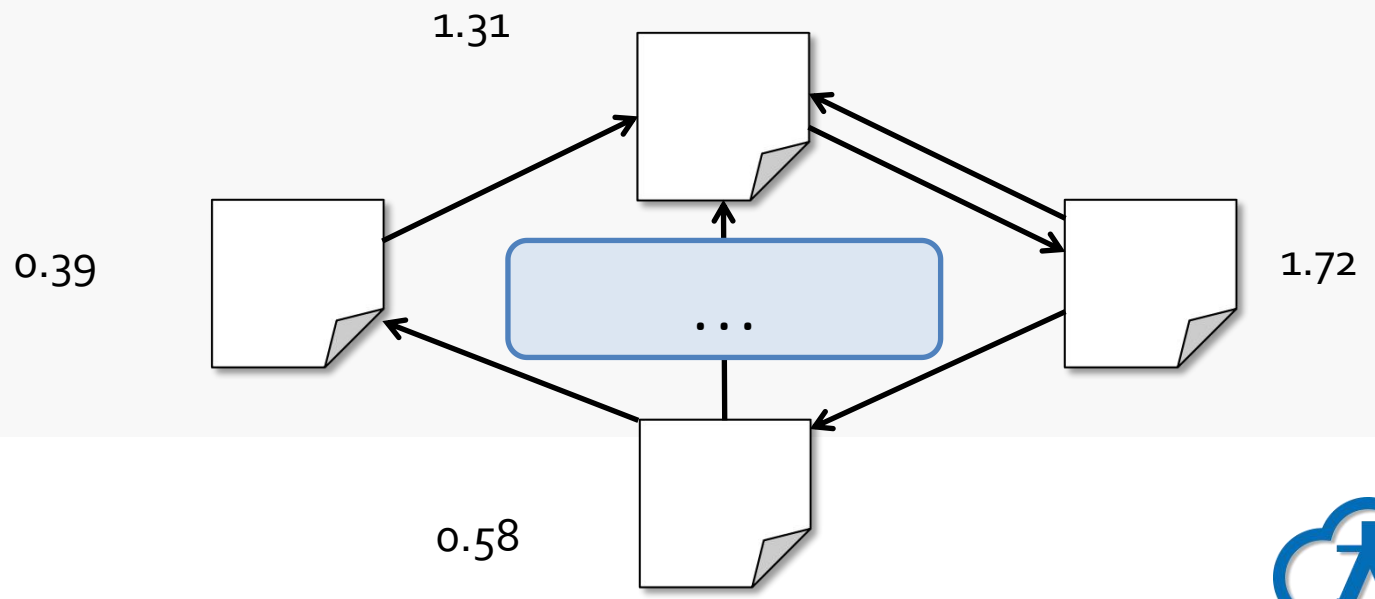
Algorithm

1. Start each page at a rank of 1
2. On each iteration, have page p contribute $rank_p$ / $|neighbors_p|$ to its neighbors
3. Set each page's rank to 0.15 + 0.85 × contribs



Algorithm

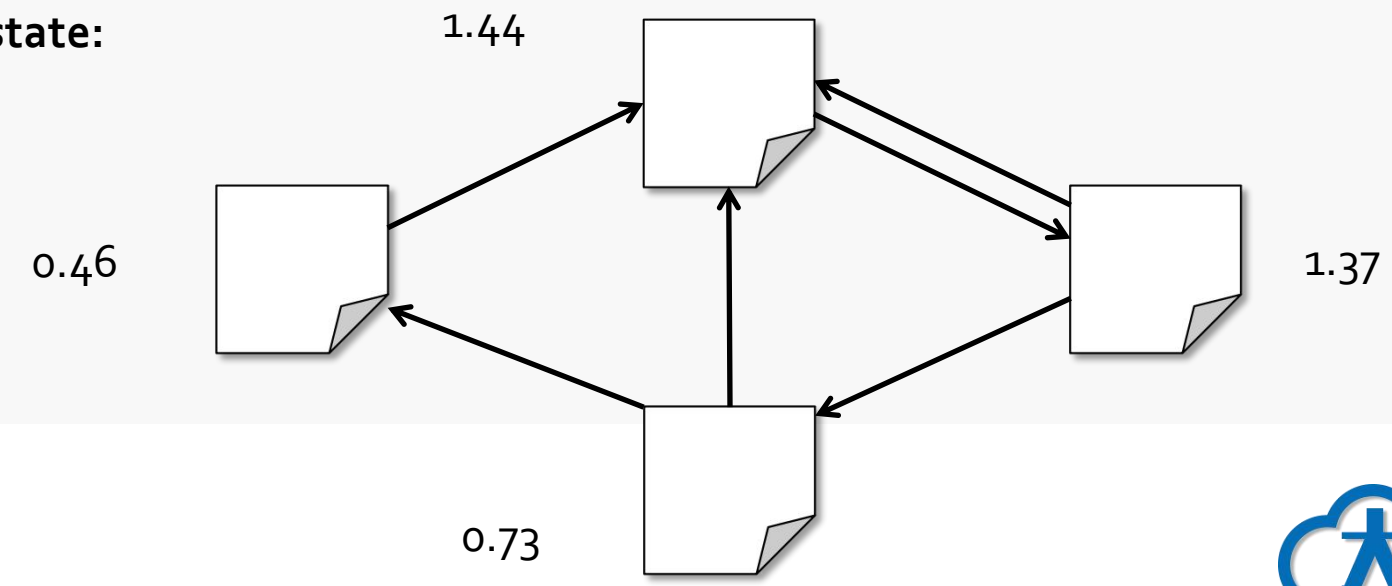
1. Start each page at a rank of 1
2. On each iteration, have page p contribute $rank_p$ / $|neighbors_p|$ to its neighbors
3. Set each page's rank to 0.15 + 0.85 × contribs



Algorithm

1. Start each page at a rank of 1
2. On each iteration, have page p contribute $rank_p$ / $|neighbors_p|$ to its neighbors
3. Set each page's rank to 0.15 + 0.85 × contribs

Final state:



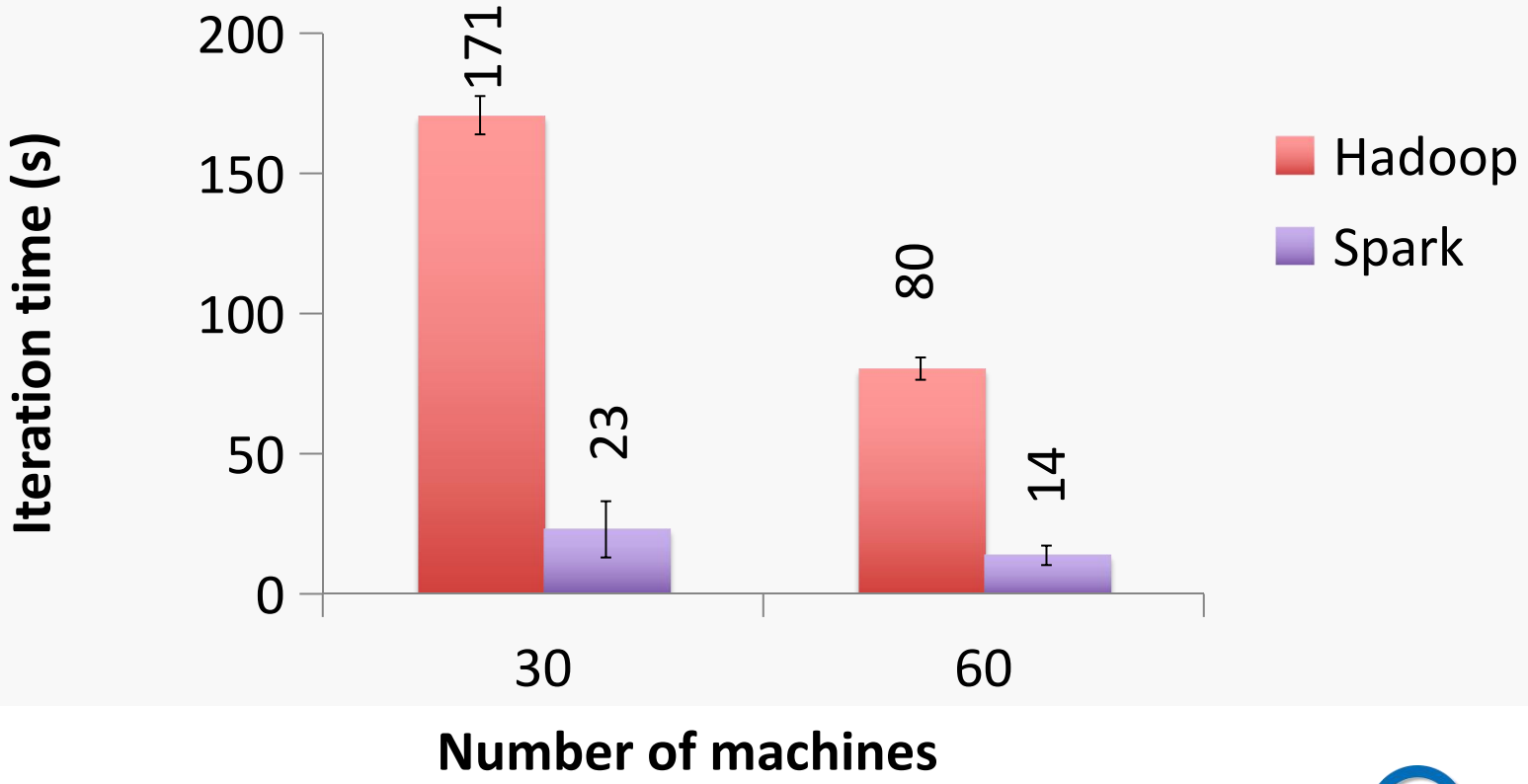
Scala Implementation

```
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 Performance



3. 数据处理与分析

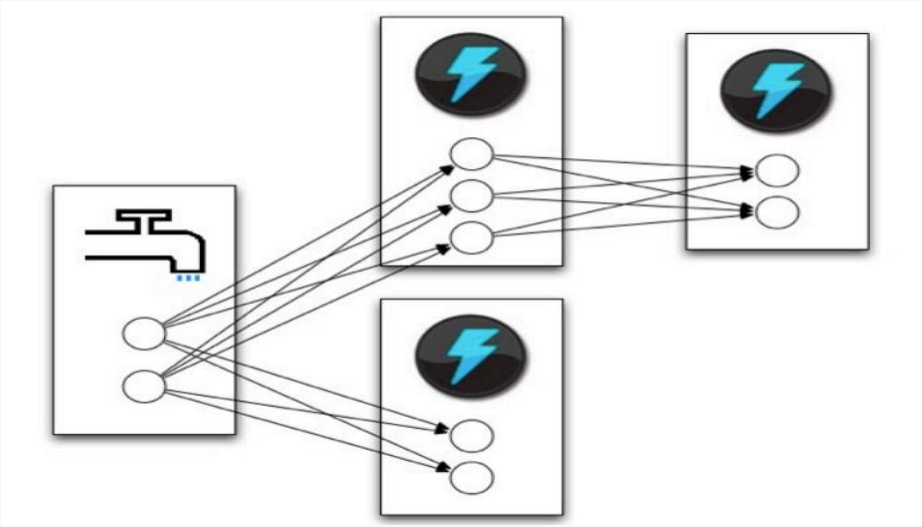
- 批处理
  - MapReduce
  - Spark
- 流处理
  - Storm
  - Spark Streaming

大数据基础与应用
BIG DATA FOUNDATION AND APPLICATION

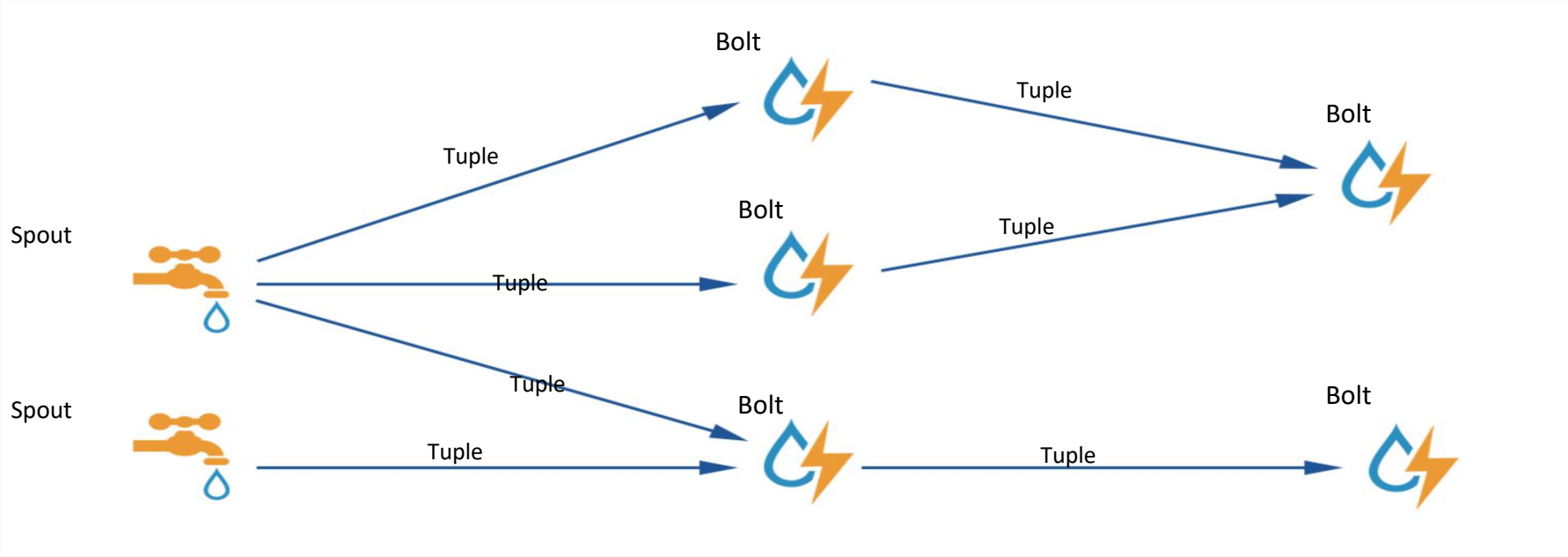
Storm

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



Streams

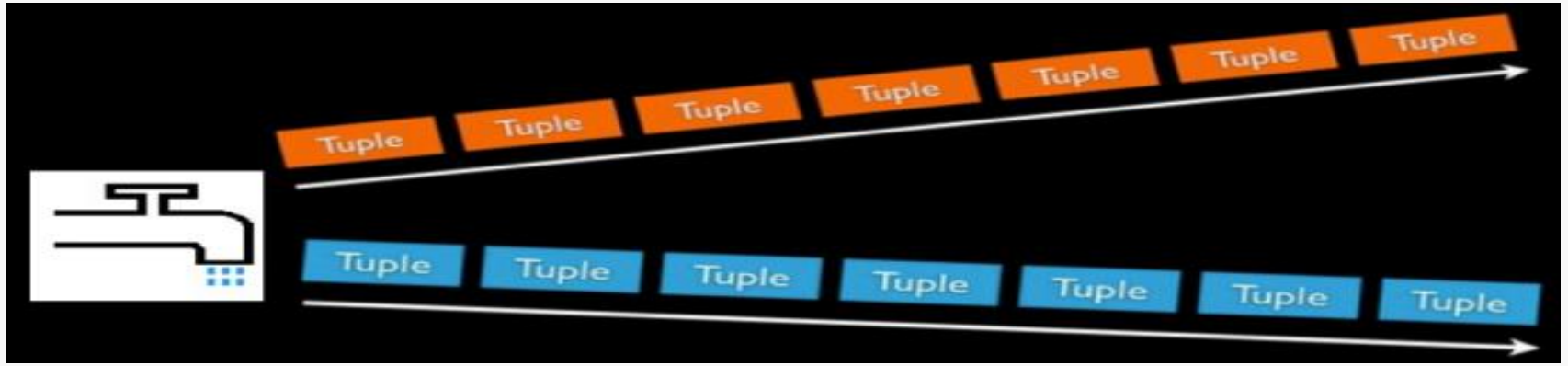
- Streams：持续的Tuple流

•Tuple：数据处理单元，一个Tuple由多个字段组成

Field1	Field2	Field3	Field4

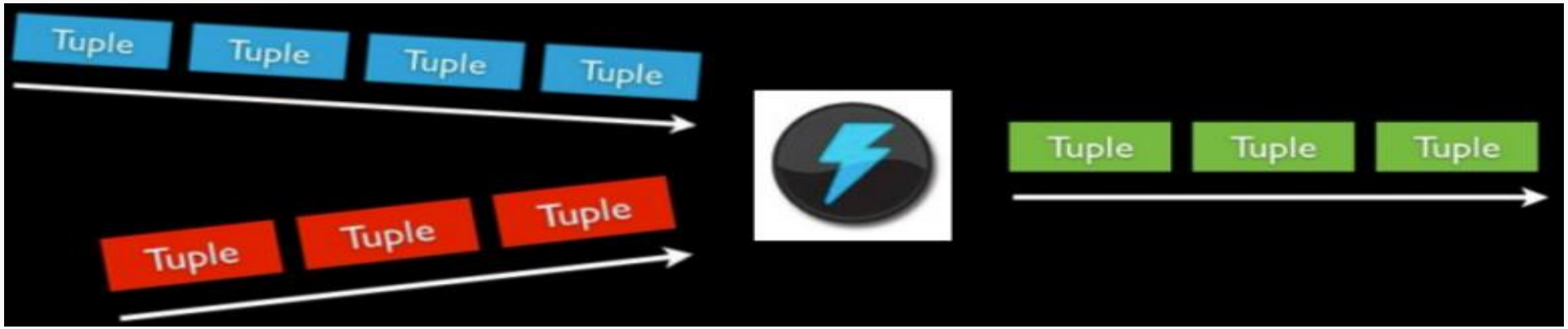
Spout

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



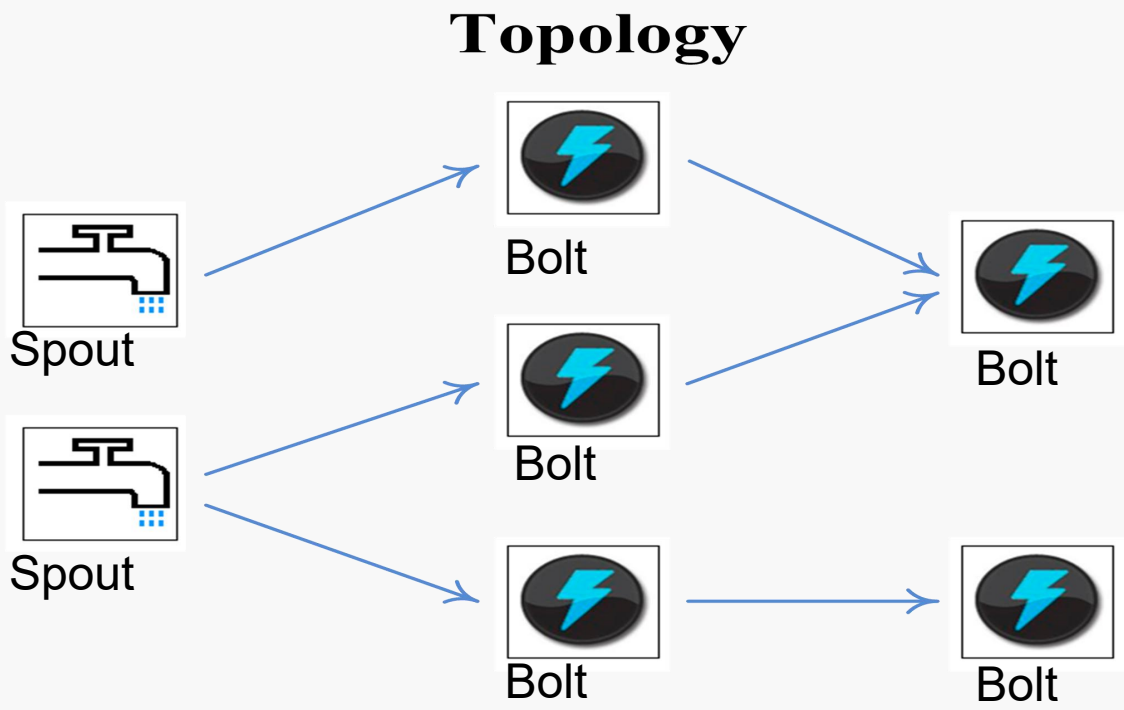
Bolt

- **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。

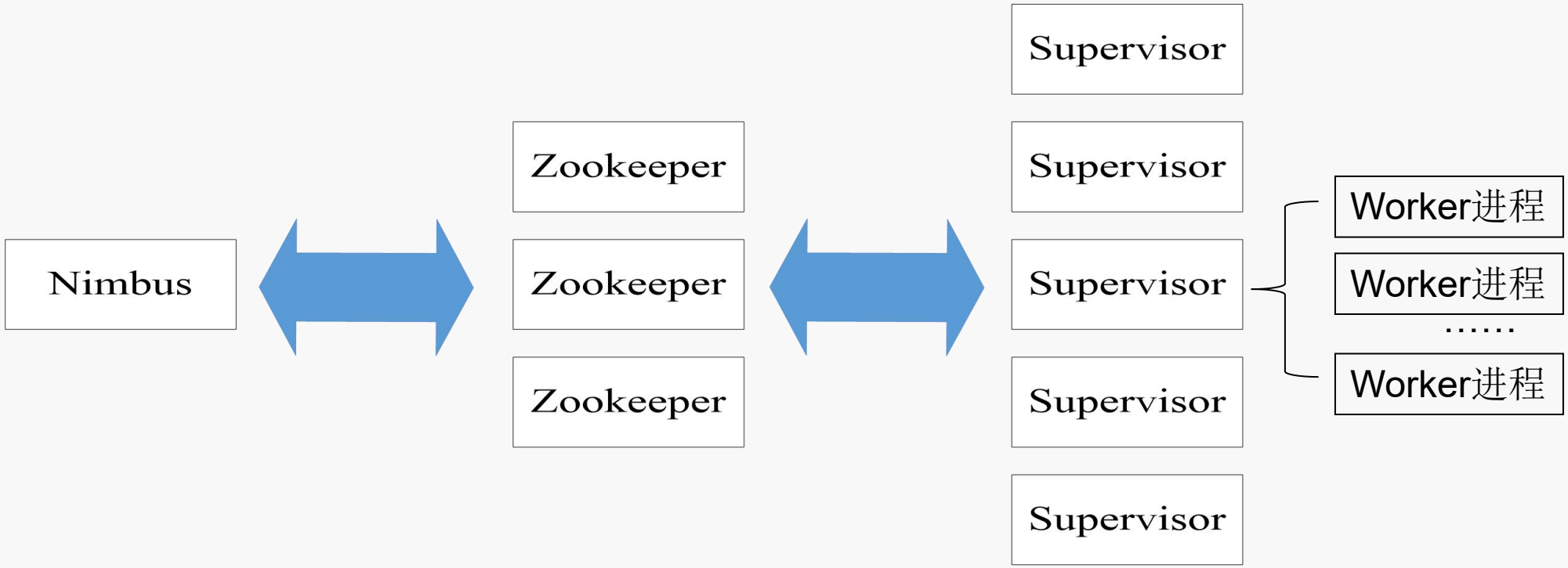


Topology

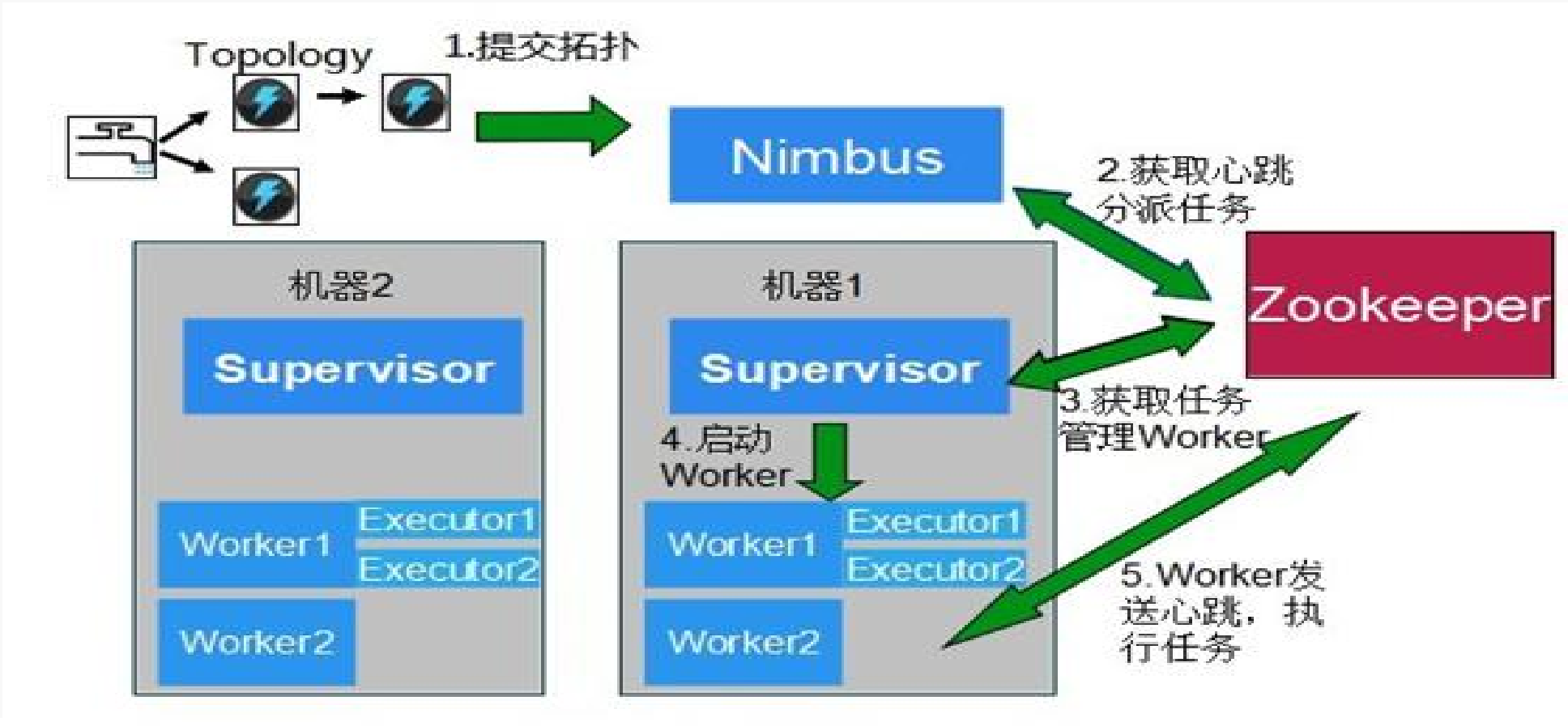
- Topology：一个应用的spout，bolt，grouping组合



Storm系统架构



Storm工作流程



Spark Streaming

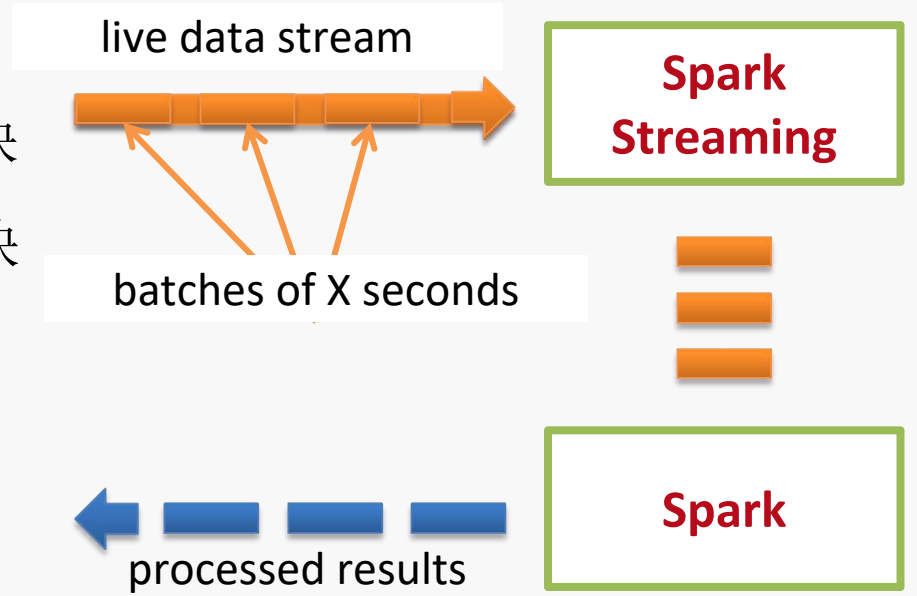
Spark Streaming概述

- 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

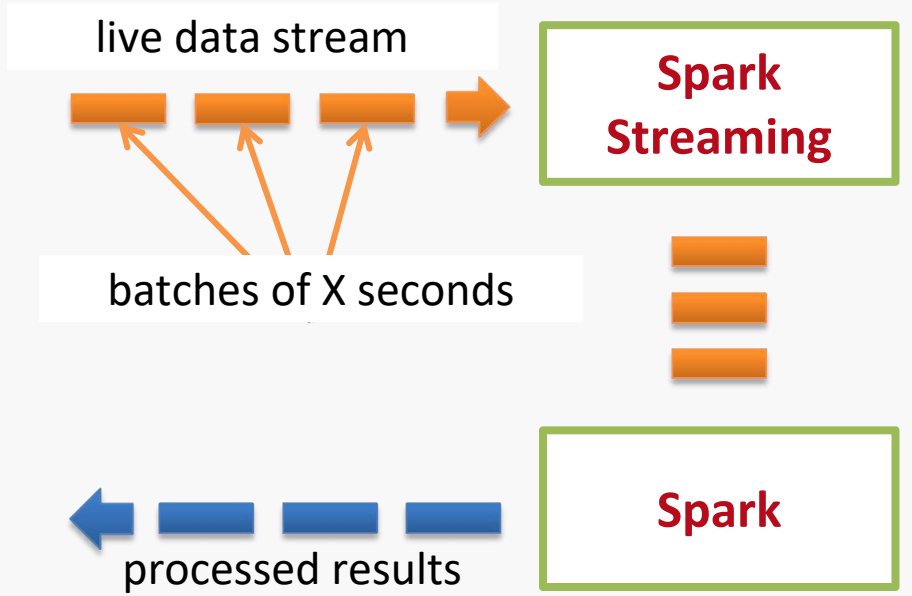
- 把实时输入数据流以时间片（如X秒）为单位切分成块
- 把每块数据作为一个RDD，使用RDD操作处理每一小块数据
- RDD操作的结果也以一小块一小块的形式返回



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

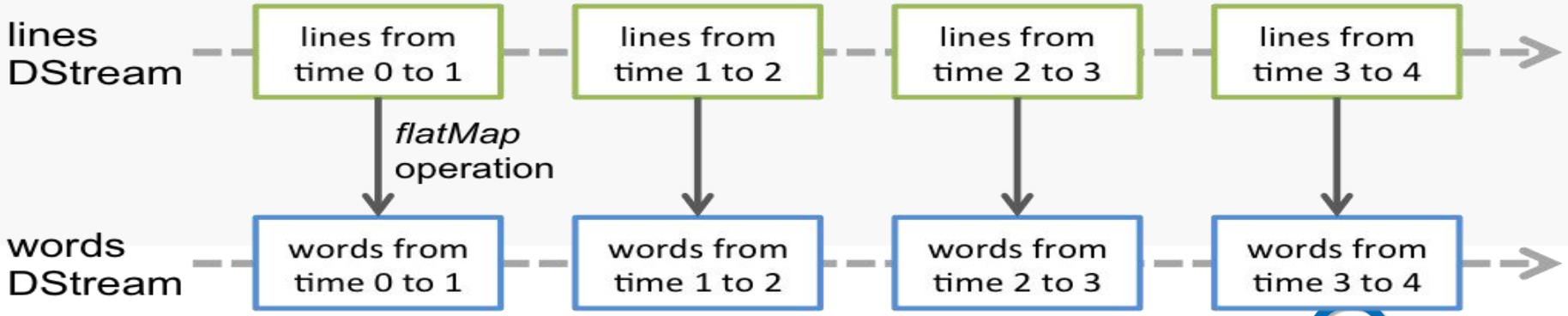
- 数据块时间片大小可低至 ½ 秒, 延迟大约为1 秒



Discretized Stream (DStream) Processing

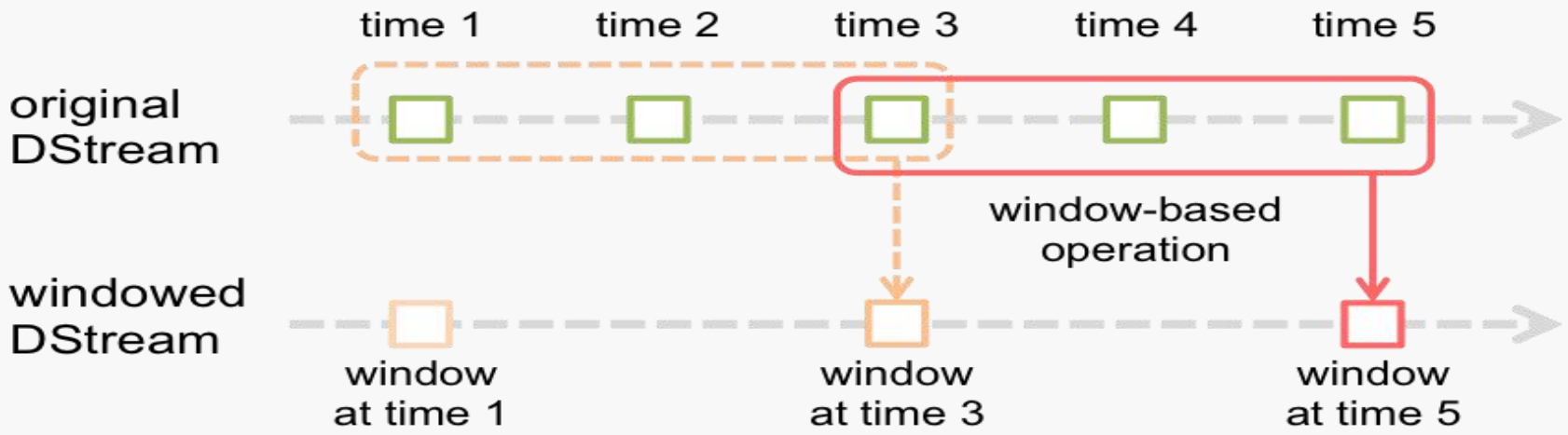
- DStream内部是由一系列连续的RDD组成的，每个RDD都包含了特定时间间隔内的一批数据

- 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))
```