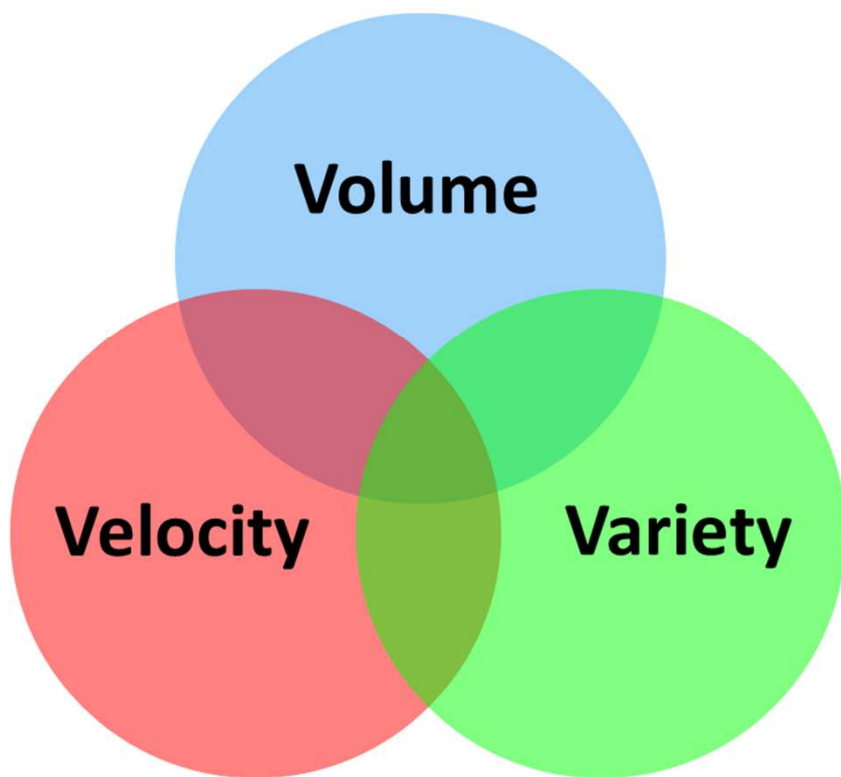


大数据系统与大规模数据分析

# 大数据运算系统 (4)



陈世敏

中科院计算所  
计算机体系结构  
国家重点实验室

©2015-2017 陈世敏

# Outline

- 内存计算

- 内存数据库

- 内存键值系统

- 内存MapReduce

- Spark

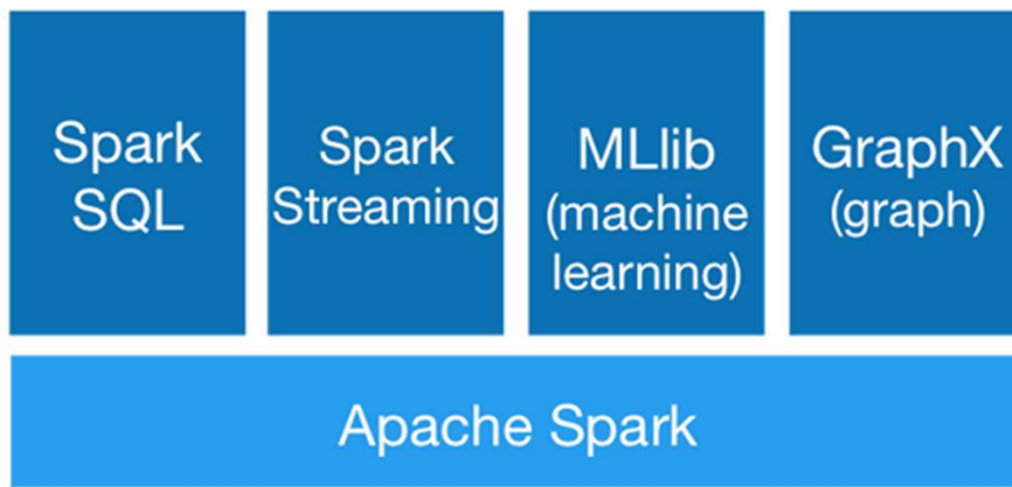
- Cloudera Impala

- 数据流系统

# Spark

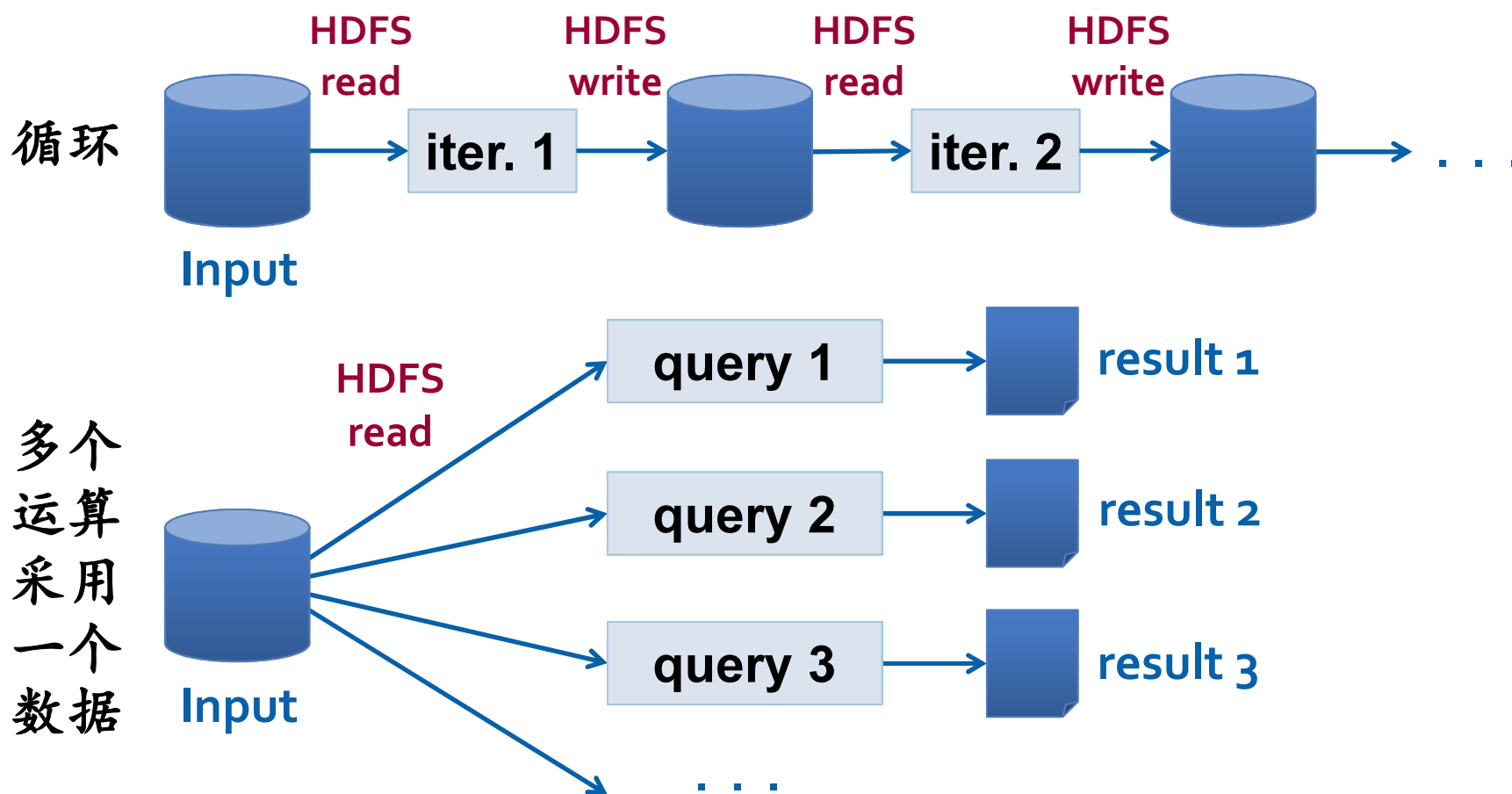
- 原理
- 编程
- 实现

# Spark: 面向大数据分析的内存系统



- Berkeley AMP Lab 研发
- 可以从HDFS读数据，但是运算中数据放在内存中，不使用Hadoop，而是新实现了分布式的处理
- 目标是低延迟的分析操作
- “Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing”, NSDI’12

# MapReduce的问题



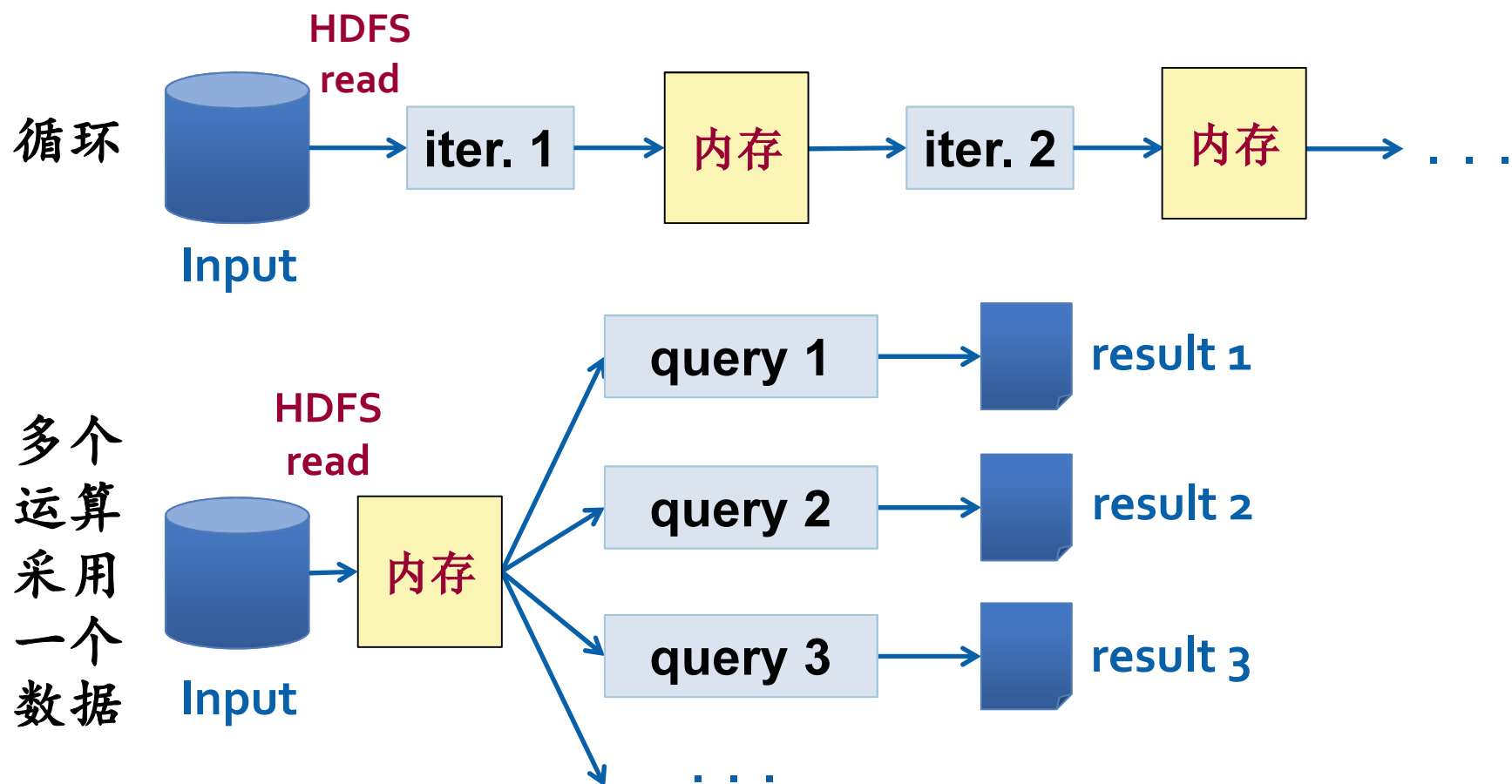
- 通过HDFS进行作业间数据共享，代价太高

图来源：NSDI'12 slides

# Spark的思路

- 内存容量越来越大
- 把数据放入多台机器的内存
- 避免HDFS开销

# Spark的思路



# 运算和数据如何表达？

- 目标

- 高效

- 容错



# RDD

- Resilient Distributed Data sets

- 一个数据集
- 只读，整个数据集创建后不能修改
- 通常进行整个数据集的运算

- 优点

- 并发控制被简化了
- 可以记录lineage（数据集上的运算序列），可以重新计算
  - 并不需要把RDD存储在stable storage上

# RDD vs. Distributed Shared Memory

Aspect	RDDs	Distr. Shared Mem.
Reads	Coarse- or fine-grained	Fine-grained
Writes	Coarse-grained	Fine-grained
Consistency	Trivial (immutable)	Up to app / runtime
Fault recovery	Fine-grained and low-overhead using lineage	Requires checkpoints and program rollback
Straggler mitigation	Possible using backup tasks	Difficult
Work placement	Automatic based on data locality	Up to app (runtimes aim for transparency)
Behavior if not enough RAM	Similar to existing data flow systems	Poor performance (swapping?)

图来源：NSDI'12 paper

# RDD运算

- Transformation

- 输入是RDD(数据集)
- 输出也是RDD(数据集)
- $RDD \rightarrow RDD$

- Action

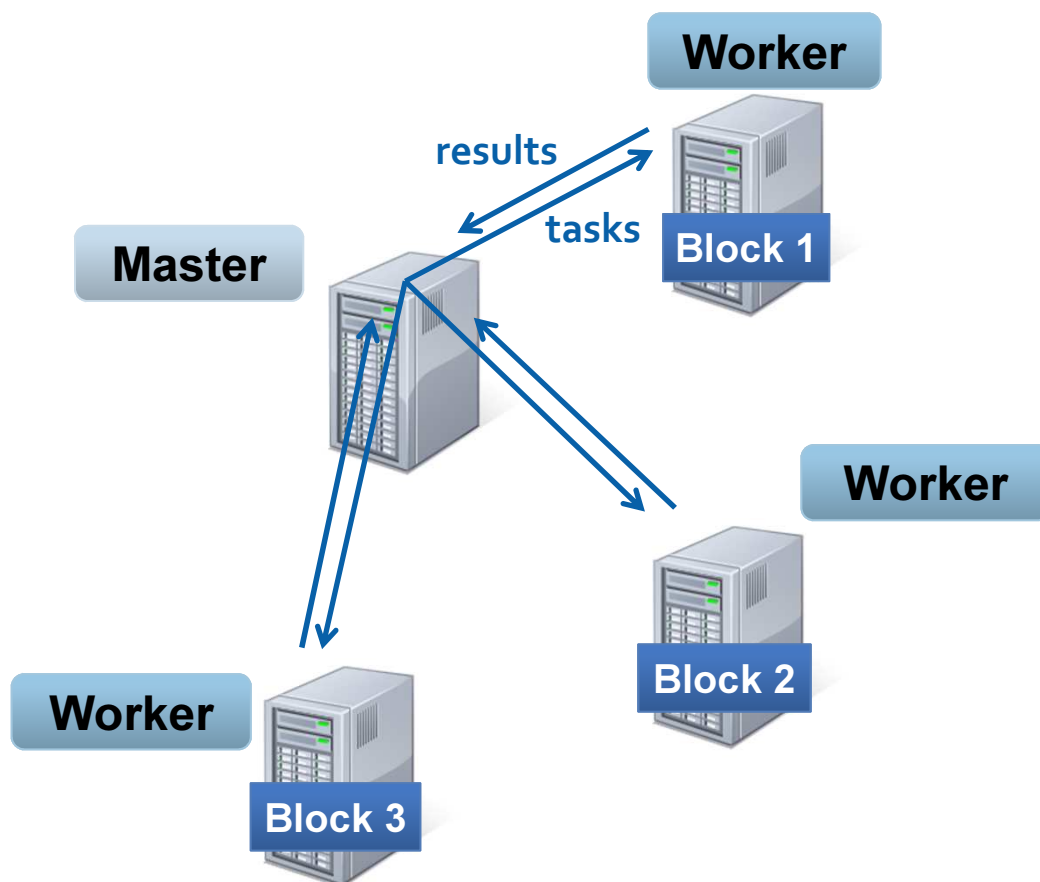
- 输入是RDD(数据集)
- 输出是某种计算结果（例如，一个数值或者一系列数值）
  - 注意：RDD可能非常大，但是计算结果总是比较小的
- $RDD \rightarrow \text{计算结果}$

图来源：NSDI'12 paper

# 运算过程

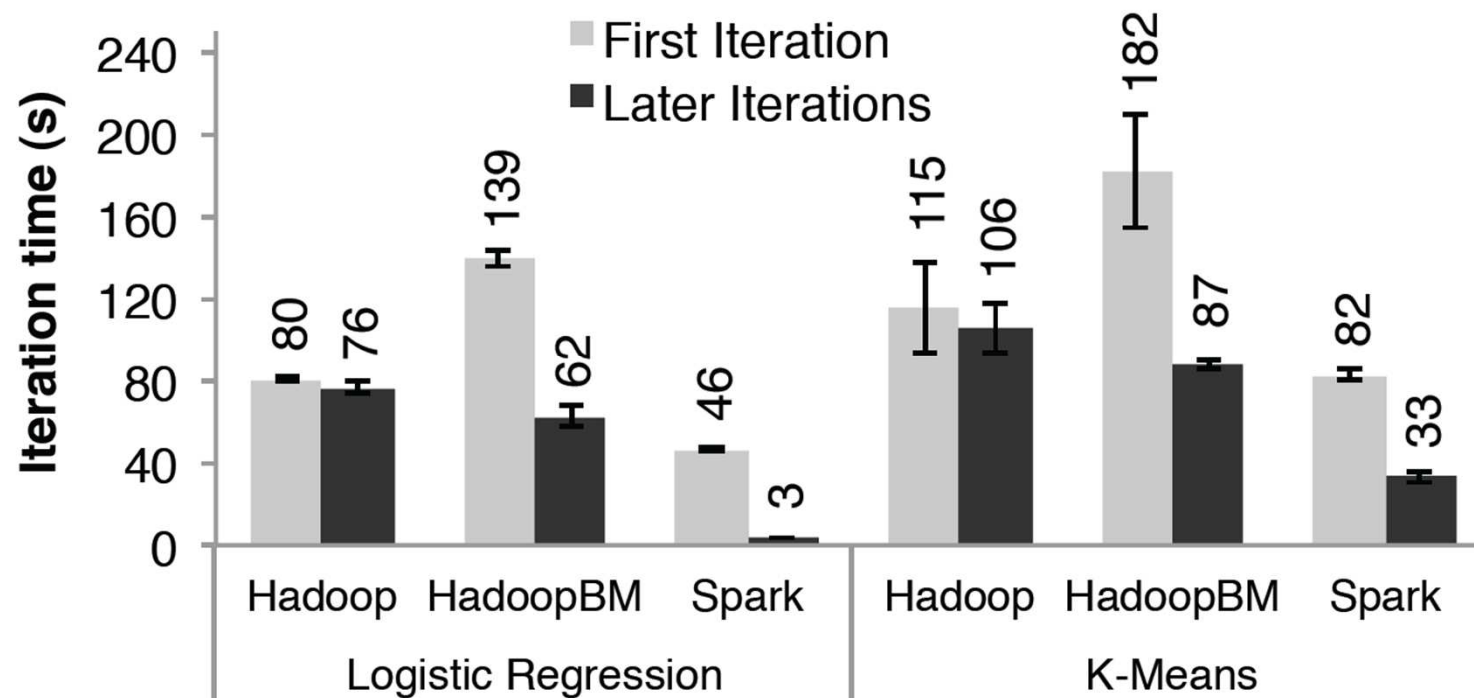
读入内存一次

在内存中可以多次处理



图来源：NSDI'12 slides

# 性能比较



- HadoopBM: 第一个MapReduce把输入转换为二进制
- 一旦装入了内存，性能有明显改善

图来源：NSDI'12 paper

# Spark

- 原理
- 编程
- 实现

# Scala

- Spark支持的主要语言（之一）
  - 其它语言：Java, Python
- Scala是一种新的程序设计语言
  - 面向目标的（Object Oriented）
  - 函数型（Functional）
- Scala程序是在JVM上执行的
- Scala语言资料 <http://www.scala-lang.org/>
  - 本课程不进行深入讲解

# 简单的例子（Scala）

```
/* SimpleApp.scala */  
  
import org.apache.spark.SparkContext  
import org.apache.spark.SparkContext._  
import org.apache.spark.SparkConf  
  
object SimpleApp {  
  def main(args: Array[String]) {  
    val logFile = "YOUR_SPARK_HOME/README.md" // Should be some file on your system  
    val conf = new SparkConf().setAppName("Simple Application")  
    val sc = new SparkContext(conf)  
    val logData = sc.textFile(logFile, 2).cache()  
    val numAs = logData.filter(line => line.contains("a")).count()  
    val numBs = logData.filter(line => line.contains("b")).count()  
    println("Lines with a: %s, Lines with b: %s".format(numAs, numBs))  
  }  
}
```

例子来源：Spark Manual



# 简单的例子（Scala）

```
/* SimpleApp.scala */
```

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf
```

有些像Java import,  
或者C/C++ include,  
所需要的库说明

```
object SimpleApp {
  def main(args: Array[String]) {
    val logFile = "YOUR_SPARK_HOME/README.md" // Should be some file on your system
    val conf = new SparkConf().setAppName("Simple Application")
    val sc = new SparkContext(conf)
    val logData = sc.textFile(logFile, 2).cache()
    val numAs = logData.filter(line => line.contains("a")).count()
    val numBs = logData.filter(line => line.contains("b")).count()
    println("Lines with a: %s, Lines with b: %s".format(numAs, numBs))
  }
}
```

# 简单的例子（Scala）

```
/* SimpleApp.scala */  
import org.apache.spark.SparkContext  
import org.apache.spark.SparkContext._  
import org.apache.spark.SparkConf  
  
object SimpleApp {  
  def main(args: Array[String]) {  
    val logFile = "YOUR_SPARK_HOME/README.md" // Should be some file on your system  
    val conf = new SparkConf().setAppName("Simple Application")  
    val sc = new SparkContext(conf)  
    val logData = sc.textFile(logFile, 2).cache()  
    val numAs = logData.filter(line => line.contains("a")).count()  
    val numBs = logData.filter(line => line.contains("b")).count()  
    println("Lines with a: %s, Lines with b: %s".format(numAs, numBs))  
  }  
}
```

主程序，  
实际上是driver，  
发出Spark操作请求

# 简单的例子 (Scala)

```
/* SimpleApp.scala */  
import org.apache.spark.SparkContext  
import org.apache.spark.SparkContext._  
import org.apache.spark.SparkConf  
  
object SimpleApp {  
  def main(args: Array[String]) {  
    val logFile = "YOUR_SPARK_HOME/README.md" // Should be some file on your system  
    val conf = new SparkConf().setAppName("Simple Application")  
    val sc = new SparkContext(conf)  
    val logData = sc.textFile(logFile, 2).cache()  
    val numAs = logData.filter(line => line.contains("a")).count()  
    val numBs = logData.filter(line => line.contains("b")).count()  
    println("Lines with a: %s, Lines with b: %s".format(numAs, numBs))  
  }  
}
```

建立  
SparkContext

# 简单的例子 (Scala)

```
/* SimpleApp.scala */  
import org.apache.spark.SparkContext  
import org.apache.spark.SparkContext._  
import org.apache.spark.SparkConf  
  
object SimpleApp {  
  def main(args: Array[String]) {  
    val logFile = "YOUR_SPARK_HOME/README.md" // Should be some file on your system  
    val conf = new SparkConf().setAppName("Simple Application")  
    val sc = new SparkContext(conf)  
    val logData = sc.textFile(logFile, 2).cache()  
    val numAs = logData.filter(line => line.contains("a")).count()  
    val numBs = logData.filter(line => line.contains("b")).count()  
    println("Lines with a: %s, Lines with b: %s".format(numAs, numBs))  
  }  
}
```

读文本文件生  
成一个RDD



# 简单的例子 (Scala)

```
/* SimpleApp.scala */  
import org.apache.spark.SparkContext  
import org.apache.spark.SparkContext._  
import org.apache.spark.SparkConf  
  
object SimpleApp {  
  def main(args: Array[String]) {  
    val logFile = "YOUR_SPARK_HOME/README.md" // Should be some file on your system  
    val conf = new SparkConf().setAppName("Simple Application")  
    val sc = new SparkContext(conf)  
    val logData = sc.textFile(logFile, 2).cache()  
    val numAs = logData.filter(line => line.contains("a")).count()  
    val numBs = logData.filter(line => line.contains("b")).count()  
    println("Lines with a: %s, Lines with b: %s".format(numAs, numBs))  
  }  
}
```

**RDD操作**

# 简单的例子 (Scala)

```
/* SimpleApp.scala */  
import org.apache.spark.SparkContext  
import org.apache.spark.SparkContext._  
import org.apache.spark.SparkConf  
  
object SimpleApp {  
  def main(args: Array[String]) {  
    val logFile = "YOUR_SPARK_HOME/README.md" // Should be some file on your system  
    val conf = new SparkConf().setAppName("Simple Application")  
    val sc = new SparkContext(conf)  
    val logData = sc.textFile(logFile, 2).cache()  
    val numAs = logData.filter(line => line.contains("a")).count()  
    val numBs = logData.filter(line => line.contains("b")).count()  
    println("Lines with a: %s, Lines with b: %s".format(numAs, numBs))  
  }  
}
```

**RDD filter**  
对每个元素，调用给定函数，如果True保留，False丢弃

# 简单的例子 (Scala)

```
/* SimpleApp.scala */  
import org.apache.spark.SparkContext  
import org.apache.spark.SparkContext._  
import org.apache.spark.SparkConf  
  
object SimpleApp {  
  def main(args: Array[String]) {  
    val logFile = "YOUR_SPARK_HOME/README.md" // Should be some file on your system  
    val conf = new SparkConf().setAppName("Simple Application")  
    val sc = new SparkContext(conf)  
    val logData = sc.textFile(logFile, 2).cache()  
    val numAs = logData.filter(line => line.contains("a")).count()  
    val numBs = logData.filter(line => line.contains("b")).count()  
    println("Lines with a: %s, Lines with b: %s".format(numAs, numBs))  
  }  
}
```

**RDD count**  
**有多少元素**

# 简单的例子 (Scala)

```
/* SimpleApp.scala */  
import org.apache.spark.SparkContext  
import org.apache.spark.SparkContext._  
import org.apache.spark.SparkConf  
  
object SimpleApp {  
  def main(args: Array[String]) {  
    val logFile = "YOUR_SPARK_HOME/README.md" // Should be some file on your system  
    val conf = new SparkConf().setAppName("Simple Application")  
    val sc = new SparkContext(conf)  
    val logData = sc.textFile(logFile, 2).cache()  
    val numAs = logData.filter(line => line.contains("a")).count()  
    val numBs = logData.filter(line => line.contains("b")).count()  
    println("Lines with a: %s, Lines with b: %s".format(numAs, numBs))  
  }  
}
```

**RDD操作**



# 同一个的例子 (Java)

```
import org.apache.spark.api.java.*;
import org.apache.spark.SparkConf;
import org.apache.spark.api.java.function.Function;
```

## Java import

```
public class SimpleApp {
    public static void main(String[] args) {
        String logFile = "YOUR_SPARK_HOME/README.md"; // Should be some file on your system
        SparkConf conf = new SparkConf().setAppName("Simple Application");
        JavaSparkContext sc = new JavaSparkContext(conf);
        JavaRDD<String> logData = sc.textFile(logFile).cache();

        long numAs = logData.filter(new Function<String, Boolean>() {
            public Boolean call(String s) { return s.contains("a"); }
        }).count();

        long numBs = logData.filter(new Function<String, Boolean>() {
            public Boolean call(String s) { return s.contains("b"); }
        }).count();

        System.out.println("Lines with a: " + numAs + ", lines with b: " + numBs);
    }
}
```

例子来源: Spark Manual

# 同一个的例子 (Java)

```
public class SimpleApp {  
    public static void main(String[] args) {  
        String logFile = "YOUR_SPARK_HOME/README.md"; // Should be some file on your system  
        SparkConf conf = new SparkConf().setAppName("Simple Application");  
        JavaSparkContext sc = new JavaSparkContext(conf);  
        JavaRDD<String> logData = sc.textFile(logFile).cache();  
  
        long numAs = logData.filter(new Function<String, Boolean>() {  
            public Boolean call(String s) { return s.contains("a"); }  
        }).count();  
  
        long numBs = logData.filter(new Function<String, Boolean>() {  
            public Boolean call(String s) { return s.contains("b"); }  
        }).count();  
  
        System.out.println("Lines with a: " + numAs + ", lines with b: " + numBs);  
    }  
}
```

主程序，  
实际上是driver，  
发出Spark操作请求

# 同一个的例子 (Java)

```
public class SimpleApp {  
    public static void main(String[] args) {  
        String logFile = "YOUR_SPARK_HOME/README.md"; // Should be some file on your system  
        SparkConf conf = new SparkConf().setAppName("Simple Application");  
        JavaSparkContext sc = new JavaSparkContext(conf);  
        JavaRDD<String> logData = sc.textFile(logFile).cache();  
  
        long numAs = logData.filter(new Function<String, Boolean>() {  
            public Boolean call(String s) { return s.contains("a"); }  
        }).count();  
  
        long numBs = logData.filter(new Function<String, Boolean>() {  
            public Boolean call(String s) { return s.contains("b"); }  
        }).count();  
  
        System.out.println("Lines with a: " + numAs + ", lines with b: " + numBs);  
    }  
}
```

建立  
SparkContext

# 同一个的例子 (Java)

```
public class SimpleApp {  
    public static void main(String[] args) {  
        String logFile = "YOUR_SPARK_HOME/README.md"; // Should be some file on your system  
        SparkConf conf = new SparkConf().setAppName("Simple Application");  
        JavaSparkContext sc = new JavaSparkContext(conf);  
        JavaRDD<String> logData = sc.textFile(logFile).cache();  
  
        long numAs = logData.filter(new Function<String, Boolean>() {  
            public Boolean call(String s) { return s.contains("a"); }  
        }).count();  
  
        long numBs = logData.filter(new Function<String, Boolean>() {  
            public Boolean call(String s) { return s.contains("b"); }  
        }).count();  
  
        System.out.println("Lines with a: " + numAs + ", lines with b: " + numBs);  
    }  
}
```

读文本文件  
生成一个  
JavaRDD



# 同一个的例子 (Java)

```
public class SimpleApp {  
    public static void main(String[] args) {  
        String logFile = "YOUR_SPARK_HOME/README.md"; // Should be some file on your system  
        SparkConf conf = new SparkConf().setAppName("Simple Application");  
        JavaSparkContext sc = new JavaSparkContext(conf);  
        JavaRDD<String> logData = sc.textFile(logFile).cache();  
  
        long numAs = logData.filter(new Function<String, Boolean>() {  
            public Boolean call(String s) { return s.contains("a"); }  
        }).count();  
    }  
}
```

## RDD操作

### 看一下filter的参数:

- 一个类的对象: 这里是匿名的类, 实现了spark.api.java.function
- 包含call函数
- call的参数是logData这个JavaRDD的元素类型, 即String
- call的返回值是boolean
- 将对logData的每个元素调用call一次, true 保留, false 丢弃

# 同一个的例子 (Java)

```
public class SimpleApp {  
    public static void main(String[] args) {  
        String logFile = "YOUR_SPARK_HOME/README.md"; // Should be some file on your system  
        SparkConf conf = new SparkConf().setAppName("Simple Application");  
        JavaSparkContext sc = new JavaSparkContext(conf);  
        JavaRDD<String> logData = sc.textFile(logFile).cache();  
  
        long numAs = logData.filter(new Function<String, Boolean>() {  
            public Boolean call(String s) { return s.contains("a"); }  
        }).count();  
  
        long numBs = logData.filter(new Function<String, Boolean>() {  
            public Boolean call(String s) { return s.contains("b"); }  
        }).count();  
  
        System.out.println("Lines with a: " + numAs + ", lines with b: " + numBs);  
    }  
}
```

另一个filter,  
count

# 以Java为例讲一下

- RDD输入
- RDD transformation
- RDD action

# Java RDD的类型

- Class `JavaRDD<T>`

- 元素类型为T的RDD

- Class `JavaPairRDD<K,V>`

- 元素包含一个K和一个V

- 转换

- `JavaRDD → JavaPairRDD`

- `JavaPairRDD.fromJavaRDD(rdd)`
    - 要求rdd的每个元素是`Tuple2<K,V>`类型

- `JavaPairRDD → JavaRDD`

- `JavaPairRDD.keys()`, `JavaPairRDD.values()`
    - 可以把K和V部分分别形成一个JavaRDD



# RDD输入

- Class `JavaSparkContext` 中函数
- `JavaSparkContext.textFile(path)`
  - 从文本文件读入，每一行是一个元素，元素类型为String
  - 返回`JavaRDD<String>`
- `JavaSparkContext.wholeTextFiles(path)`
  - 读一个目录，每个文件成为一个元素
  - key是文件路径，value是文件内容
  - 返回`JavaPairRDD<String,String>`
- 其它
  - `sequenceFile`, `hadoopRDD`, `binaryFiles`, 等

# 程序产生RDD

```
List<Integer> data = Arrays.asList(1, 2, 3, 4, 5);
```

```
JavaRDD<Integer> distData = sc.parallelize(data);
```

Class JavaSparkContext 中 parallelize 函数

可以根据Java程序中的List产生JavaRDD

# RDD运算

<i>map</i> ( $f : T \Rightarrow U$ )	: RDD[T] $\Rightarrow$ RDD[U]	<b>Transformation</b>
<i>filter</i> ( $f : T \Rightarrow \text{Bool}$ )	: RDD[T] $\Rightarrow$ RDD[T]	
<i>flatMap</i> ( $f : T \Rightarrow \text{Seq}[U]$ )	: RDD[T] $\Rightarrow$ RDD[U]	
<i>sample</i> ( <i>fraction</i> : Float)	: RDD[T] $\Rightarrow$ RDD[T] (Deterministic sampling)	
<i>groupByKey</i> ()	: RDD[(K, V)] $\Rightarrow$ RDD[(K, Seq[V])]	
<i>reduceByKey</i> ( $f : (V, V) \Rightarrow V$ )	: RDD[(K, V)] $\Rightarrow$ RDD[(K, V)]	
<i>union</i> ()	: (RDD[T], RDD[T]) $\Rightarrow$ RDD[T]	
<i>join</i> ()	: (RDD[(K, V)], RDD[(K, W)]) $\Rightarrow$ RDD[(K, (V, W))]	
<i>cogroup</i> ()	: (RDD[(K, V)], RDD[(K, W)]) $\Rightarrow$ RDD[(K, (Seq[V], Seq[W]))]	
<i>crossProduct</i> ()	: (RDD[T], RDD[U]) $\Rightarrow$ RDD[(T, U)]	
<i>mapValues</i> ( $f : V \Rightarrow W$ )	: RDD[(K, V)] $\Rightarrow$ RDD[(K, W)] (Preserves partitioning)	
<i>sort</i> ( $c : \text{Comparator}[K]$ )	: RDD[(K, V)] $\Rightarrow$ RDD[(K, V)]	
<i>partitionBy</i> ( $p : \text{Partitioner}[K]$ )	: RDD[(K, V)] $\Rightarrow$ RDD[(K, V)]	
<i>count</i> ()	: RDD[T] $\Rightarrow$ Long	<b>Action</b>
<i>collect</i> ()	: RDD[T] $\Rightarrow$ Seq[T]	
<i>reduce</i> ( $f : (T, T) \Rightarrow T$ )	: RDD[T] $\Rightarrow$ T	
<i>lookup</i> ( $k : K$ )	: RDD[(K, V)] $\Rightarrow$ Seq[V] (On hash/range partitioned RDDs)	
<i>save</i> ( <i>path</i> : String)	: Outputs RDD to a storage system, <i>e.g.</i> , HDFS	

图来源：NSDI'12 paper

# RDD Transformation

- 运算的结果是新的RDD
  - 有2类
  - 通用的：在JavaRDD和JavaPairRDD上都可以运算
  - 仅在JavaPairRDD上的运算：需要Key

# Transformation

- `JavaRDD<R> A;`
- **Map: 一对一映射**
  - `B=A.map(f)`
    - 对A的每个元素调用f.call, 其返回值成为B的一个元素
    - 这样形成的RDD就是B
  - `JavaRDD<R> map(Function<T,R> f)`
  - Function interface中仅定义了一个函数
    - `R call(T v)`
- 下面用词说明: 输出RDD = 输入RDD.transform(...)
  - 用输入RDD——A
  - 用输出RDD——B
- **MapToPair**
  - `JavaPairRDD<K2,V2>mapToPair(PairFunction<T,K2,V2> f)`

# Transformation

- Filter: 过滤

- `JavaRDD<T> filter(Function<T, Boolean> f)`
- 对输入RDD的每个元素调用f.call
  - 如果返回true, 那么此元素成为输出RDD的一个元素
  - 如果返回false, 那么就把这个元素丢弃

- FlatMap: 一对多映射

- `JavaRDD<R> flatMap(FlatMapFunction<T, R> f)`
- f.call有这样的定义: `Iterable<R> call(T v)`
- 对输入RDD的每个元素调用f.call, 其返回值的每个R都成为输出RDD的一个元素

# Transformation

- Sample: 采样

- `JavaRDD<T> sample(boolean withReplacement, double fraction)`
  - 对输入RDD进行采样，采样的结果放入输出RDD

- Union: 并集

- Intersection: 交集

- Distinct: 去掉重复元素

# JavaPairRDD Transformation

- groupByKey

- `JavaPairRDD<K,Iterable<V>> groupByKey()`
- 把输入JavaPairRDD中，相同的Key的元素group by，所有的同组的Value放入Iterable，形成输出JavaPairRDD

- Join

- `JavaPairRDD<K, Tuple2<V,W>> join(JavaPairRDD<K,W> C)`
- `B=A.join(C)`
- 把Key相同的A和C的元素Join在一起，把A的value和C的value形成一个Tuple2结构，成为输出JavaPairRDD的value



# JavaPairRDD Transformation

- reduceByKey

- `JavaPairRDD<K,V> reduceByKey(Function2<V,V,V> f)`
- `f.call`有这样的定义: `T3 call(T1 v1, T2 v2)`
- 把输入JavaPairRDD中, 相同的Key的元素, group by, 所有的同组的Value放入调用`f.call`, 生成一个元素, 形成输出JavaPairRDD
- 这个与MapReduce中的Reduce相似

- 其它

- cartesian
- aggregateByKey
- sortByKey

# Action

- 在RDD上的运算结果，而不是RDD
- Reduce
  - `T reduce(Function2<T,T,T> f)`
  - `f.call`有这样的定义：`T3 call(T1 v1, T2 v2)`
  - 输入RDD的所有元素都调用`f.call`，生成一个值返回
- Collect: 返回列表
  - `List<T> collect()`
  - 与`parallelize`功能相反，把RDD转换为List返回
- Count: 元素数
- 其它
  - `saveAsTextFile`, `saveAsSequenceFile`等
  - `countByKey`, `lookup`等

# 让我们来看些例子

- Spark自带的例子

- `examples/src/main/java/org/apache/spark/examples`

- Word Count

- PageRank

```

public final class JavaWordCount {
    private static final Pattern SPACE = Pattern.compile(" ");

    public static void main(String[] args) throws Exception {

        if (args.length < 1) {
            System.err.println("Usage: JavaWordCount <file>");
            System.exit(1);
        }

```

## 主要部分

```

        SparkConf sparkConf = new SparkConf().setAppName("JavaWordCount");
        JavaSparkContext ctx = new JavaSparkContext(sparkConf);

```

```

        JavaRDD<String> lines = ctx.textFile(args[0], 1);

```

```

        JavaRDD<String> words = lines.flatMap(new FlatMapFunction<String, String>() {
            @Override
            public Iterable<String> call(String s) {
                return Arrays.asList(SPACE.split(s));
            }
        });

```

```

        JavaPairRDD<String, Integer> ones = words.mapToPair(new PairFunction<String, String, Integer>() {
            @Override
            public Tuple2<String, Integer> call(String s) {
                return new Tuple2<String, Integer>(s, 1);
            }
        });

```

```

        JavaPairRDD<String, Integer> counts = ones.reduceByKey(new Function2<Integer, Integer, Integer>() {
            @Override
            public Integer call(Integer i1, Integer i2) {
                return i1 + i2;
            }
        });

```

```

        List<Tuple2<String, Integer>> output = counts.collect();
        for (Tuple2<?, ?> tuple : output) {
            System.out.println(tuple._1() + ": " + tuple._2());
        }
        ctx.stop();
    }
}

```

例子来源：Spark Code

# Word Count

```
JavaRDD<String> lines = ctx.textFile(args[0], 1);

JavaRDD<String> words = lines.flatMap(
    new FlatMapFunction<String, String>() {
        @Override
        public Iterable<String> call(String s) {
            return Arrays.asList(SPACe.split(s));
        }
    });

JavaPairRDD<String, Integer> ones = words.mapToPair(
    new PairFunction<String, String, Integer>() {
        @Override
        public Tuple2<String, Integer> call(String s) {
            return new Tuple2<String, Integer>(s, 1);
        }
    });

JavaPairRDD<String, Integer> counts = ones.reduceByKey(
    new Function2<Integer, Integer, Integer>() {
        @Override
        public Integer call(Integer i1, Integer i2) {
            return i1 + i2;
        }
    });
```

读文本文件  
生成一个  
JavaRDD,  
1是partition个数

# Word Count

```
JavaRDD<String> lines = ctx.textFile(args[0], 1);
```

```
JavaRDD<String> words = lines.flatMap(  
    new FlatMapFunction<String, String>() {  
        @Override  
        public Iterable<String> call(String s) {  
            return Arrays.asList(SPACE.split(s));  
        }  
    });
```

```
JavaPairRDD<String, Integer> ones = words.mapToPair(  
    new PairFunction<String, String, Integer>() {  
        @Override  
        public Tuple2<String, Integer> call(String s) {  
            return new Tuple2<String, Integer>(s, 1);  
        }  
    });
```

```
JavaPairRDD<String, Integer> counts = ones.reduceByKey(  
    new Function2<Integer, Integer, Integer>() {  
        @Override  
        public Integer call(Integer i1, Integer i2) {  
            return i1 + i2;  
        }  
    });
```

**FlatMap**  
words是所有单词组成的RDD

# Word Count

```
JavaRDD<String> lines = ctx.textFile(args[0], 1);

JavaRDD<String> words = lines.flatMap(
    new FlatMapFunction<String, String>() {
        @Override
        public Iterable<String> call(String s) {
            return Arrays.asList(SPACE.split(s));
        }
    });
```

```
JavaPairRDD<String, Integer> ones = words.mapToPair(
    new PairFunction<String, String, Integer>() {
        @Override
        public Tuple2<String, Integer> call(String s) {
            return new Tuple2<String, Integer>(s, 1);
        }
    });
```

```
JavaPairRDD<String, Integer> counts = ones.reduceByKey(
    new Function2<Integer, Integer, Integer>() {
        @Override
        public Integer call(Integer i1, Integer i2) {
            return i1 + i2;
        }
    });
```

**mapToPair**  
类似map, 生成K,V



# Word Count

```
JavaRDD<String> lines = ctx.textFile(args[0], 1);

JavaRDD<String> words = lines.flatMap(
    new FlatMapFunction<String, String>() {
        @Override
        public Iterable<String> call(String s) {
            return Arrays.asList(SPACE.split(s));
        }
    });

JavaPairRDD<String, Integer> ones = words.mapToPair(
    new PairFunction<String, String, Integer>() {
        @Override
        public Tuple2<String, Integer> call(String s) {
            return new Tuple2<String, Integer>(s, 1);
        }
    });

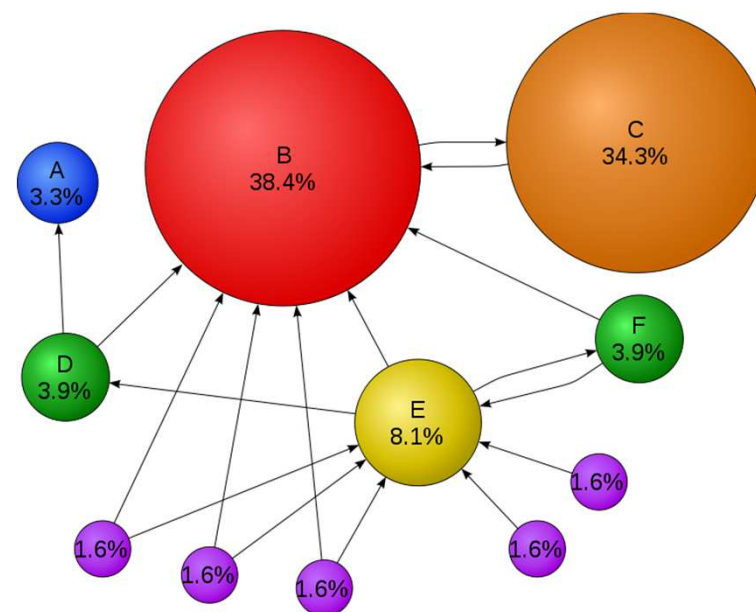
JavaPairRDD<String, Integer> counts = ones.reduceByKey(
    new Function2<Integer, Integer, Integer>() {
        @Override
        public Integer call(Integer i1, Integer i2) {
            return i1 + i2;
        }
    });
```

reduceByKey



# PageRank

- $R_u = 1 - d + d \sum_{v \in B(u)} \frac{R_v}{L_v}$ 
  - $R_v$ : 顶点v的PageRank\*N
  - $L_v$ : 顶点v的出度（出边的条数）
  - $B(u)$ : 顶点u的入邻居集合
  - d: damping factor
  - N: 总顶点个数



图来源: Wikipedia

## • 计算方法

- 初始化: 所有的顶点的PageRank为**1**
- 迭代: 用上述公式迭代直至收敛

# Page Rank

```
JavaRDD<String> lines = ctx.textFile(args[0], 1);

// Loads all URLs from input file and initialize their neighbors.
JavaPairRDD<String, Iterable<String>> links = lines.mapToPair(
    new PairFunction<String, String, String>() {
        @Override
        public Tuple2<String, String> call(String s) {
            String[] parts = SPACES.split(s);
            return new Tuple2<String, String>(parts[0], parts[1]);
        }
    }).distinct().groupByKey().cache();

// Loads all URLs with other URL(s) link to from input file
// and initialize ranks of them to one.
JavaPairRDD<String, Double> ranks = links.mapValues(
    new Function<Iterable<String>, Double>() {
        @Override
        public Double call(Iterable<String> rs) {
            return 1.0;
        }
    });

// Calculates and updates URL ranks continuously using PageRank algorithm.
for (int current = 0; current < Integer.parseInt(args[1]); current++) {
    // Calculates URL contributions to the rank of other URLs.
    JavaPairRDD<String, Double> contribs = links.join(ranks).values()
        .flatMapToPair(new PairFlatMapFunction<Tuple2<Iterable<String>, Double>, String, Double>() {
            @Override
            public Iterable<Tuple2<String, Double>> call(Tuple2<Iterable<String>, Double> s) {
                int urlCount = Iterables.size(s._1);
                List<Tuple2<String, Double>> results = new ArrayList<Tuple2<String, Double>>();
                for (String d : s._1) {
                    results.add(new Tuple2<String, Double>(d, s._2() / urlCount));
                }
                return results;
            }
        });

    // Re-calculates URL ranks based on neighbor contributions.
    ranks = contribs.reduceByKey(new Sum()).mapValues(new Function<Double, Double>() {
        @Override
        public Double call(Double sum) {
            return 0.15 + sum * 0.85;
        }
    });
}
```

初始化

循环

例子来源：Spark Code

陈世敏(chensm@ict.ac.cn)

# Page Rank初始化

```
// Loads in input file. It should be in format of:
//      URL      neighbor URL
//      URL      neighbor URL
//      URL      neighbor URL
//      ...
JavaRDD<String> lines = ctx.textFile(args[0], 1);
```

```
// Loads all URLs from input file and initialize their neighbors.
JavaPairRDD<String, Iterable<String>> links = lines.mapToPair(
    new PairFunction<String, String, String>() {
        @Override
        public Tuple2<String, String> call(String s) {
            String[] parts = SPACES.split(s);
            return new Tuple2<String, String>(parts[0], parts[1]);
        }
    })
    .distinct().groupByKey().cache();

// Loads all URLs with other URL(s) link to from input file
// and initialize ranks of them to one.
JavaPairRDD<String, Double> ranks = links.mapValues(
    new Function<Iterable<String>, Double>() {
        @Override
        public Double call(Iterable<String> rs) {
            return 1.0;
        }
    })
    .collect();
```

读文本文件  
生成一个  
JavaRDD,  
每行一个元素

# Page Rank初始化

```
// Loads in input file. It should be in format of:
//      URL      neighbor URL
//      URL      neighbor URL
//      URL      neighbor URL
//      ...
JavaRDD<String> lines = ctx.textFile(args[0], 1);
```

**mapToPair提取  
(URL, neighbor URL)  
为一个JavaPairRDD**



```
// Loads all URLs from input file and initialize their neighbors.
JavaPairRDD<String, Iterable<String>> links = lines.mapToPair(
    new PairFunction<String, String, String>() {
        @Override
        public Tuple2<String, String> call(String s) {
            String[] parts = SPACES.split(s);
            return new Tuple2<String, String>(parts[0], parts[1]);
        }
    })
    .distinct().groupByKey().cache();
```

```
// Loads all URLs with other URL(s) link to from input file
// and initialize ranks of them to one.
JavaPairRDD<String, Double> ranks = links.mapValues(
    new Function<Iterable<String>, Double>() {
        @Override
        public Double call(Iterable<String> rs) {
            return 1.0;
        }
    })
    .groupByKey().mapValues(ranks => 1.0 / ranks.size());
```



# Page Rank初始化

```
// Loads in input file. It should be in format of:
//      URL      neighbor URL
//      URL      neighbor URL
//      URL      neighbor URL
//      ...
JavaRDD<String> lines = ctx.textFile(args[0], 1);

// Loads all URLs from input file and initialize their neighbors.
JavaPairRDD<String, Iterable<String>> links = lines.mapToPair(
    new PairFunction<String, String, String>() {
        @Override
        public Tuple2<String, String> call(String s) {
            String[] parts = SPACES.split(s);
            return new Tuple2<String, String>(parts[0], parts[1]);
        }
    }).distinct().groupByKey().cache();

// Loads all URLs with other URL(s) link to from input file
// and initialize ranks of them to one.
JavaPairRDD<String, Double> ranks = links.mapValues(
    new Function<Iterable<String>, Double>() {
        @Override
        public Double call(Iterable<String> rs) {
            return 1.0;
        }
    });
```

去除相同的边

# Page Rank初始化

```
// Loads in input file. It should be in format of:
//      URL      neighbor URL
//      URL      neighbor URL
//      URL      neighbor URL
//      ...
JavaRDD<String> lines = ctx.textFile(args[0], 1);
```

```
// Loads all URLs from input file and initialize their neighbors.
JavaPairRDD<String, Iterable<String>> links = lines.mapToPair(
    new PairFunction<String, String, String>() {
        @Override
        public Tuple2<String, String> call(String s) {
            String[] parts = SPACES.split(s);
            return new Tuple2<String, String>(parts[0], parts[1]);
        }
    }).distinct().groupByKey().cache();
```

把相同的起点的边都放在一起

links <src, Iterable<dest>>

```
// Loads all URLs with other URL(s) link to from i
// and initialize ranks of them to one.
JavaPairRDD<String, Double> ranks = links.mapValues(
    new Function<Iterable<String>, Double>() {
        @Override
        public Double call(Iterable<String> rs) {
            return 1.0;
        }
    });
```

# Page Rank初始化

```
// Loads in input file. It should be in format of:
//      URL      neighbor URL
//      URL      neighbor URL
//      URL      neighbor URL
//      ...
JavaRDD<String> lines = ctx.textFile(args[0], 1);
```

把相同的起点的边都放在一起

links <src, Iterable<dest>>

```
// Loads all URLs from input file and initialize their neighbors.
JavaPairRDD<String, Iterable<String>> links = lines.mapToPair(
    new PairFunction<String, String, String>() {
        @Override
        public Tuple2<String, String> call(String s) {
            String[] parts = SPACES.split(s);
            return new Tuple2<String, String>(parts[0], parts[1]);
        }
    })
    .distinct().groupByKey().cache();
```

把links的value部分map成为1.0,

ranks<src, rank=1.0>

```
// Loads all URLs with other URL(s) link to from input file
// and initialize ranks of them to one.
JavaPairRDD<String, Double> ranks = links.mapValues(
    new Function<Iterable<String>, Double>() {
        @Override
        public Double call(Iterable<String> rs) {
            return 1.0;
        }
    })
    .cache();
```

# Page Rank循环

循环次数是输入确定的

```
// Calculates and updates URL ranks continuously using PageRank algorithm.
for (int current = 0; current < Integer.parseInt(args[1]); current++) {
    // Calculates URL contributions to the rank of other URLs.
    JavaPairRDD<String, Double> contribs = links.join(ranks).values()
        .flatMapToPair(
            new PairFlatMapFunction<Tuple2<Iterable<String>, Double>, String, Double>() {
                @Override
                public Iterable<Tuple2<String, Double>> call(Tuple2<Iterable<String>, Double> s) {
                    int urlCount = Iterables.size(s._1);
                    List<Tuple2<String, Double>> results = new ArrayList<Tuple2<String, Double>>();
                    for (String d : s._1) {
                        results.add(new Tuple2<String, Double>(d, s._2() / urlCount));
                    }
                    return results;
                }
            });

    // Re-calculates URL ranks based on neighbor contributions.
    ranks = contribs.reduceByKey(new Sum()).mapValues(new Function<Double, Double>() {
        @Override
        public Double call(Double sum) {
            return 0.15 + sum * 0.85;
        }
    });
}
```



# Page Rank循环

links <src, Iterable<dest>>,  
ranks<src, rank>  
join的结果的value部分是  
<Iterable<dest>, rank>

```
// Calculates and updates URL ranks continuously using P
for (int current = 0; current < Integer.parseInt(args[1])
// Calculates URL contributions to the rank of other URLs.
JavaPairRDD<String, Double> contribs = links.join(ranks).values()
    .flatMapToPair(
        new PairFlatMapFunction<Tuple2<Iterable<String>, Double>, String, Double>() {
            @Override
            public Iterable<Tuple2<String, Double>> call(Tuple2<Iterable<String>, Double> s) {
                int urlCount = Iterables.size(s._1);
                List<Tuple2<String, Double>> results = new ArrayList<Tuple2<String, Double>>();
                for (String d : s._1) {
                    results.add(new Tuple2<String, Double>(d, s._2() / urlCount));
                }
                return results;
            }
        });

// Re-calculates URL ranks based on neighbor contributions.
ranks = contribs.reduceByKey(new Sum()).mapValues(new Function<Double, Double>() {
    @Override
    public Double call(Double sum) {
        return 0.15 + sum * 0.85;
    }
});
}
```

# Page Rank循环

links <src, Iterable<dest>>,  
ranks<src, rank>;  
输入<Iterable<dest>, rank>

输出 contribs=<dest, rank/n>

```
// Calculates and updates URL ranks continuously using PageRank
for (int current = 0; current < Integer.parseInt(args[1]); current++) {
    // Calculates URL contributions to the rank of other URLs.
    JavaPairRDD<String, Double> contribs = links.join(ranks).values()
        .flatMapToPair(
            new PairFlatMapFunction<Tuple2<Iterable<String>, Double>, String, Double>() {
                @Override
                public Iterable<Tuple2<String, Double>> call(Tuple2<Iterable<String>, Double> s) {
                    int urlCount = Iterables.size(s._1);
                    List<Tuple2<String, Double>> results = new ArrayList<Tuple2<String, Double>>();
                    for (String d : s._1) {
                        results.add(new Tuple2<String, Double>(d, s._2() / urlCount));
                    }
                    return results;
                }
            }
        );

    // Re-calculates URL ranks based on neighbor contributions.
    ranks = contribs.reduceByKey(new Sum()).mapValues(new Function<Double, Double>() {
        @Override
        public Double call(Double sum) {
            return 0.15 + sum * 0.85;
        }
    });
}
```

# Page Rank循环

```
// Calculates and updates URL ranks continuously using PageRank algorithm.
for (int current = 0; current < Integer.parseInt(args[1]); current++) {
    // Calculates URL contributions to the rank of other URLs.
    JavaPairRDD<String, Double> contribs = links.join(ranks).values()
        .flatMapToPair(
            new PairFlatMapFunction<Tuple2<Iterable<String>, Double>, String, Double>() {
                @Override
                public Iterable<Tuple2<String, Double>> call(Tuple2<Iterable<String>, Double> s) {
                    int urlCount = Iterables.size(s._1);
                    List<Tuple2<String, Double>> results = new ArrayList<Tuple2<String, Double>>();
                    for (String d : s._1) {
                        results.add(new Tuple2<String, Double>(d, s._2() / urlCount));
                    }
                    return results;
                }
            });
    // Re-calculates URL ranks based on neighbor contributions.
    ranks = contribs.reduceByKey(new Sum()).mapValues(new Function<Double, Double>() {
        @Override
        public Double call(Double sum) {
            return 0.15 + sum * 0.85;
        }
    });
}
```

Contribs <dest, rank<sub>src</sub>/n>  
reduceByKey, 得到<dest,  $\sum \frac{rank_{src}}{n_{src}}$ >

# Page Rank循环

```
// Calculates and updates URL ranks continuously using PageRank algorithm.
for (int current = 0; current < Integer.parseInt(args[1]); current++) {
    // Calculates URL contributions to the rank of other URLs.
    JavaPairRDD<String, Double> contribs = links.join(ranks).values()
        .flatMapToPair(
            new PairFlatMapFunction<Tuple2<Iterable<String>, Double>, String, Double>() {
                @Override
                public Iterable<Tuple2<String, Double>> call(Tuple2<Iterable<String>, Double> s) {
                    int urlCount = Iterables.size(s._1);
                    List<Tuple2<String, Double>> results = new ArrayList<Tuple2<String, Double>>();
                    for (String d : s._1) {
                        results.add(new Tuple2<String, Double>(d, s._2() / urlCount));
                    }
                    return results;
                }
            }
        );
    // Re-calculates URL ranks based on neighbor contributions.
    ranks = contribs.reduceByKey(new Sum()).mapValues(new Function<Double, Double>() {
        @Override
        public Double call(Double sum) {
            return 0.15 + sum * 0.85;
        }
    });
}
```

得到 $\langle \text{dest}, 0.15 + \text{sum} * 0.85 \rangle$ 为新的ranks

```
// Re-calculates URL ranks based on neighbor contributions.
ranks = contribs.reduceByKey(new Sum()).mapValues(new Function<Double, Double>() {
    @Override
    public Double call(Double sum) {
        return 0.15 + sum * 0.85;
    }
});
}
```



# Page Rank循环

每次循环都生成新的  
**contribs**和**ranks**

```
// Calculates and updates URL ranks continuously using PageRank algorithm.
for (int current = 0; current < Integer.parseInt(args[1]); current++) {
    // Calculates URL contributions to the rank of other URLs.
    JavaPairRDD<String, Double> contribs = links.join(ranks).values()
        .flatMapToPair(
            new PairFlatMapFunction<Tuple2<Iterable<String>, Double>, String, Double>() {
                @Override
                public Iterable<Tuple2<String, Double>> call(Tuple2<Iterable<String>, Double> s) {
                    int urlCount = Iterables.size(s._1);
                    List<Tuple2<String, Double>> results = new ArrayList<Tuple2<String, Double>>();
                    for (String d : s._1) {
                        results.add(new Tuple2<String, Double>(d, s._2() / urlCount));
                    }
                    return results;
                }
            }
        );

    // Re-calculates URL ranks based on neighbor contributions.
    ranks = contribs.reduceByKey(new Sum()).mapValues(new Function<Double, Double>() {
        @Override
        public Double call(Double sum) {
            return 0.15 + sum * 0.85;
        }
    });
}
```

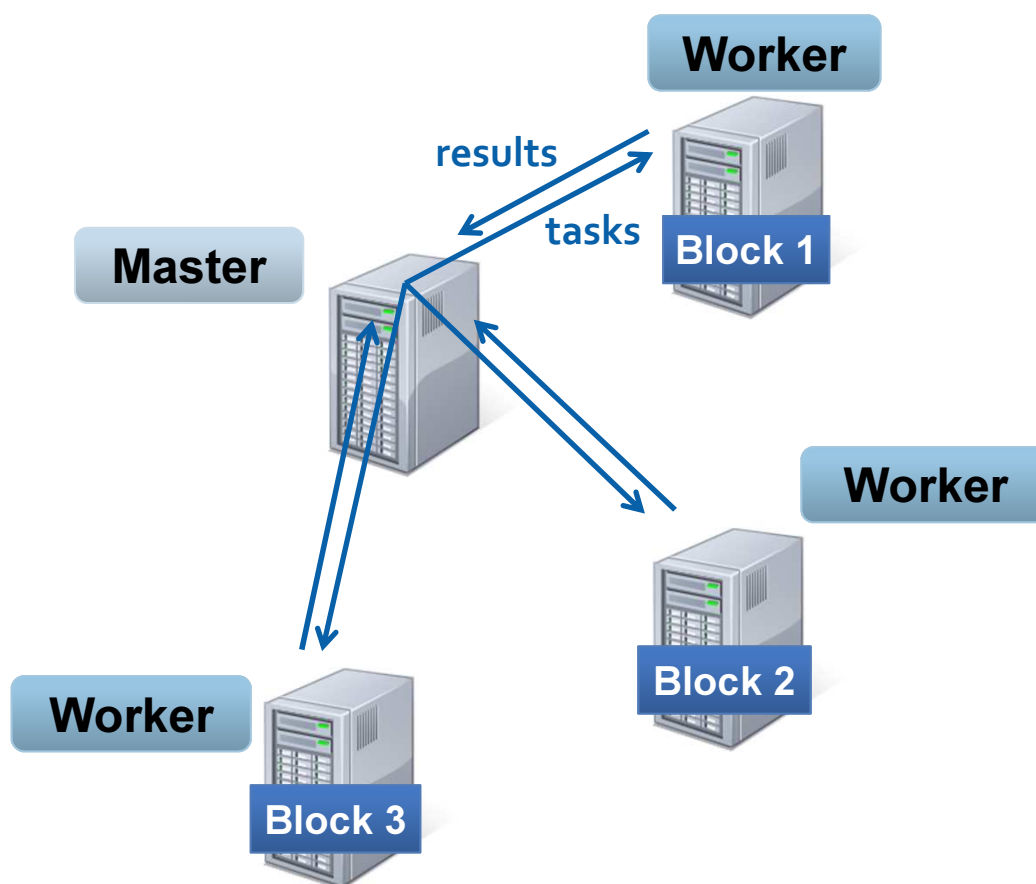
# Spark

- 原理
- 编程
- 实现

# 运算过程

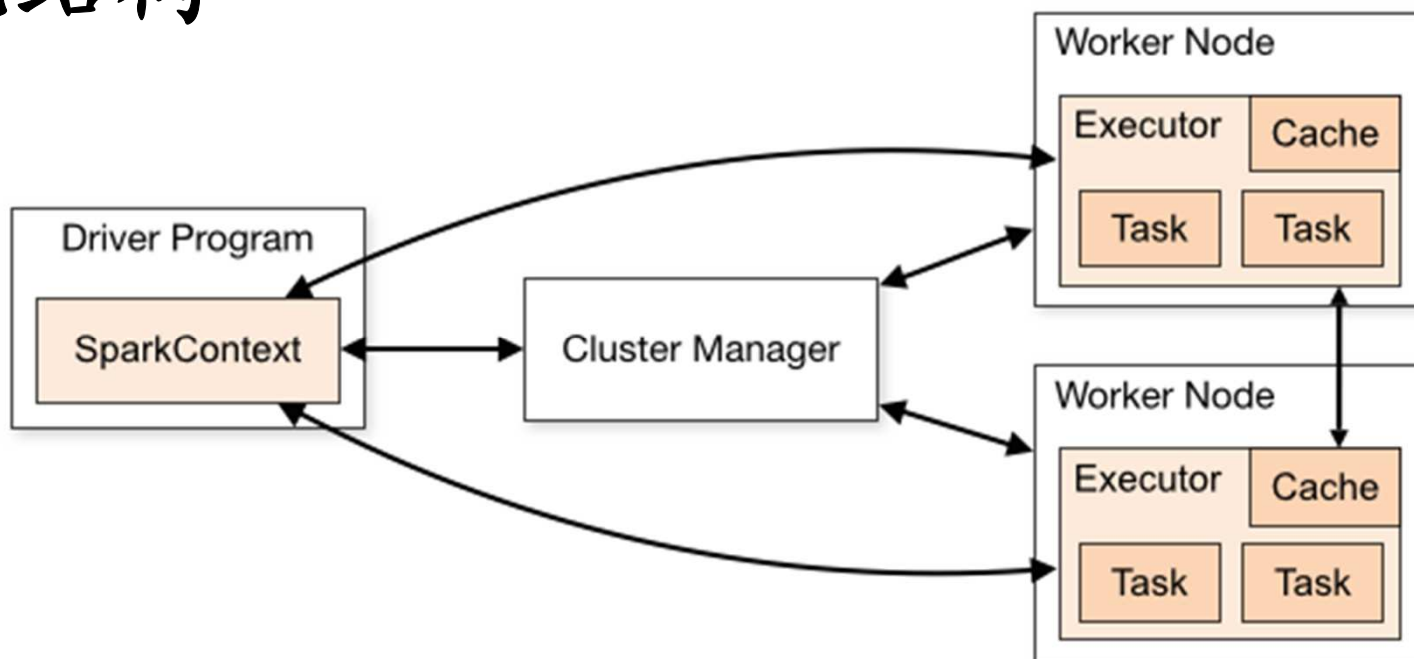
读入内存一次

在内存中可以多次处理



图来源：NSDI'12 slides

# 系统结构



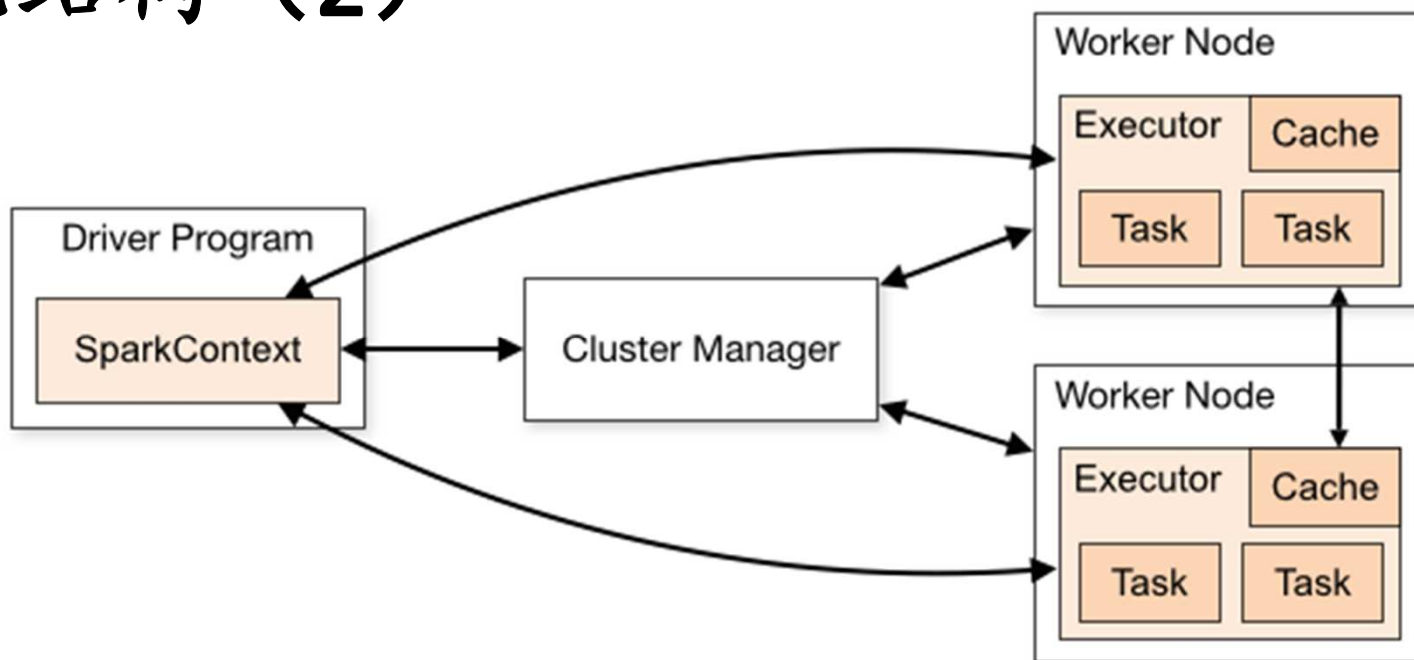
- 每个应用程序

- 一个自己的SparkContext，多个Executor
- SparkContext从外部的某种资源管理系统获取资源
  - 例如：standalone, hadoop YARN, apache Mesos
- 每个executor运行在一个不同的worker node上
- SparkContext协调多个worker运行

图来源：Spark Manual



## 系统结构（2）



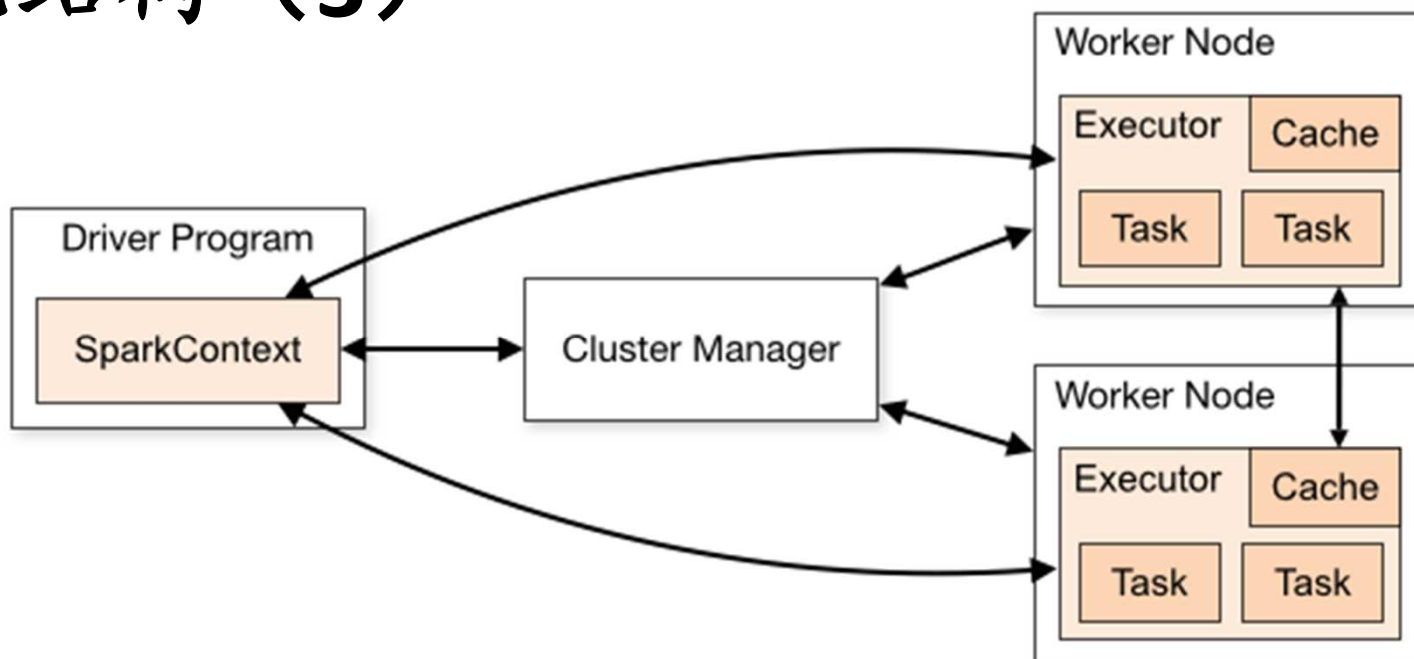
- 应用程序

- 有一个driver主程序，创建SparkContext，发出各种RDD操作要求

- Executor：执行并行的运算，存储数据

图来源：Spark Manual

## 系统结构（3）



- 多个应用程序

- 各自有自己的SparkContext
  - 互相隔离，但是也无法共享数据
- 必须通过外部的文件系统进行数据共享

图来源：Spark Manual

# Spark 中 RDD 存储方式

- 有多种方式，可以通过 `persist()` 函数指定每个 RDD 的存储方式
  - 内存缓冲
  - 内存 `serialized`(序列化)
  - 多个内存副本
  - 硬盘保存
  - 多种组合

# Spark运算的运行

- Transformation

- 仅记录，不运算
- Lazy execution

- Action

- 当遇到Action时，需要返回结果，才真正执行已经记录的前面的运算

- 容错/内存缓冲替换：当内存缓冲的RDD丢失时

- 可以重新执行记录的运算，重新计算这个RDD

# Outline

- 内存计算

  - 内存数据库

  - 内存键值系统

  - 内存MapReduce

    - Spark

    - Cloudera Impala

- 数据流系统



# Cloudera Impala

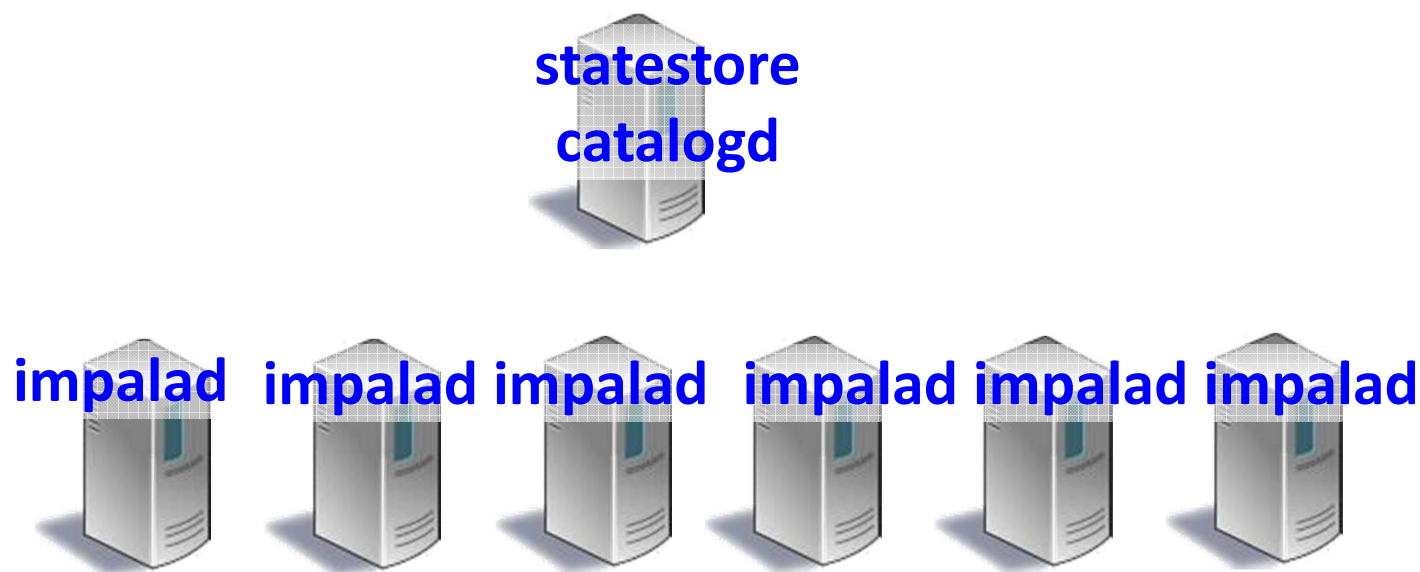
- Cloudera公司

- 2009成立
- 致力于开源大数据平台和服务
- CDH (Cloudera Distribution Including Apache Hadoop)
  - Cloudera发布的软件，包括Hadoop, Hbase, Hive, Spark, Impala, etc.

- Cloudera Impala

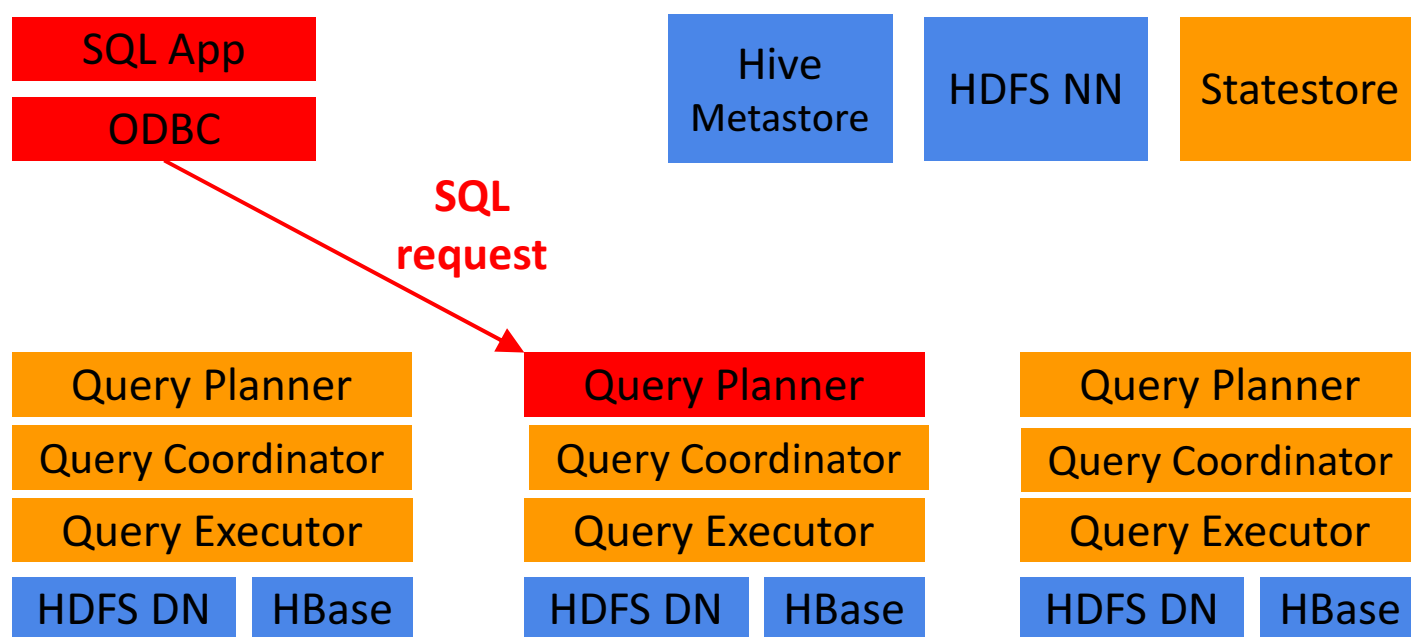
- 2012/10第一次发布
- 开源的SQL query engine，支持HDFS, HBase等

# Cloudera Impala 系统结构



- Impalad: 并行运算
- Statestore: 监控impalad的状态
- Catalogd: SQL的表格的schema的增删改操作

# Impala概念理解



图来源：Cloudera Impala: A Modern SQL Engine for Apache Hadoop



# Impala 内部

- 前端 (Java) : 支持HiveQL
  - SQL parsing
  - Query planning
- 后端 (C++)
  - Impalad, statestore, catalogd
  - 其中impalad
    - Query coordinator
    - Query execution engine
    - Code generation using LLVM

# Outline

- 内存计算

- 内存数据库
- 内存键值系统
- 内存MapReduce
  - Spark
  - Cloudera Impala

- 数据流系统

# 数据流

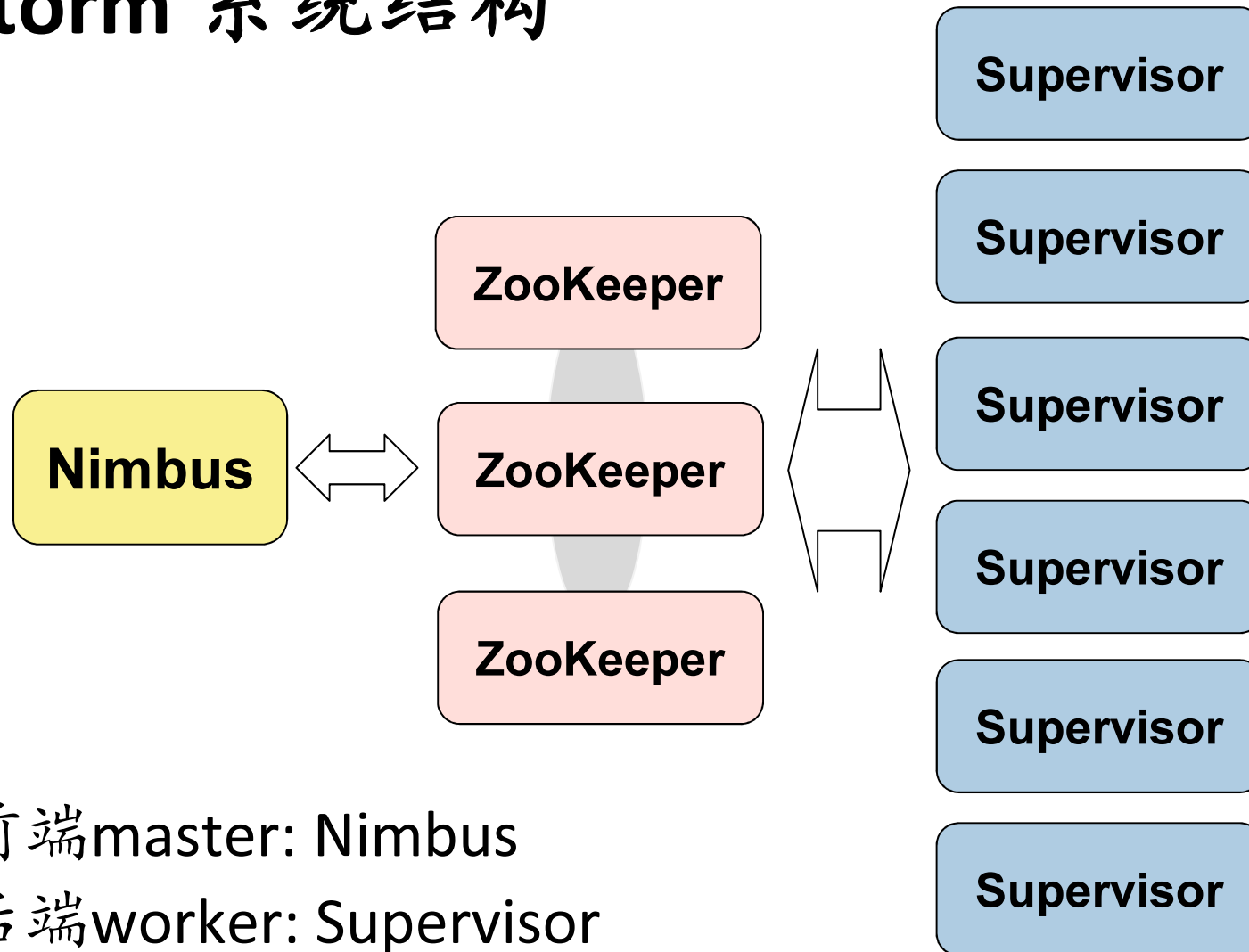
- 概念

- 数据流过系统
- 在流动的数据上完成处理

# Apache Storm

- Twitter于2011年9月发布
  - 目前是Apache的开源项目
- 数据流处理
- 内部实现：Java 与 Clojure混合实现
  - 大部分功能代码是Clojure写的
    - Clojure一种Lisp
    - 编译成为JVM bytecode
  - 提供的编程接口主要为Java
    - 函数说明等基本是Java
    - 通过thrift支持多种语言

# Storm 系统结构



- 前端master: Nimbus
- 后端worker: Supervisor
- 通过ZooKeeper通信

# Storm程序概念

- Topology

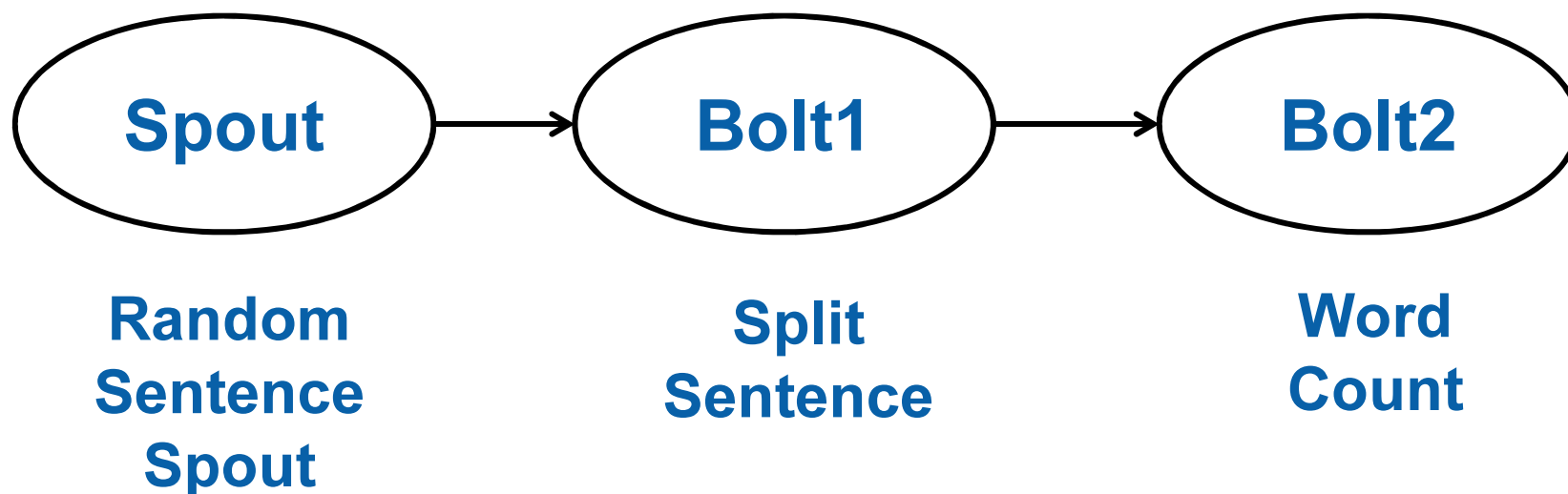
- 相当于一个job，以Topology为单位运行
  - 对应于数据流处理运算关系的一张图

- Topology中的每个顶点代表一个运算

- Spout: 产生数据流
    - 没有输入，有输出
  - Bout: 对数据流进行某种运算
    - 有输入，有输出

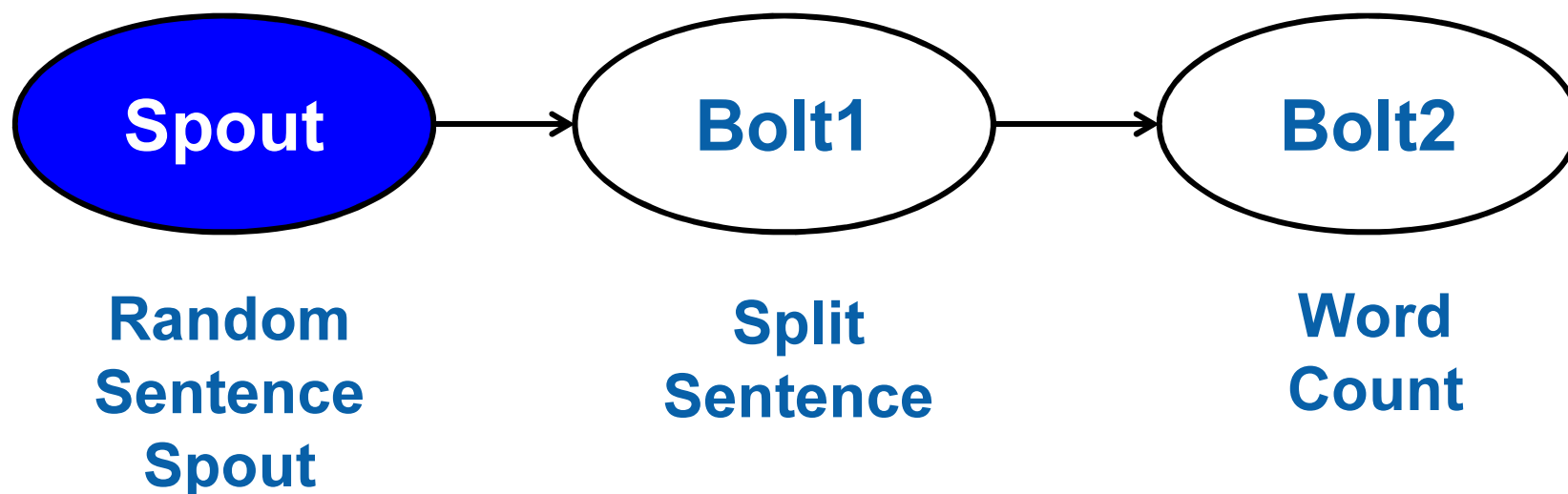
- Topology中两个顶点之间的边代表数据流动的关系

# 举例:一个简单的Topology



- Topology表示数据流上定义的一组运算

# 举例:一个简单的Topology



- Random Sentence Spout:  
每调用一次返回一个随机的句子



# 举例： RandomSentenceSpout

```
public class RandomSentenceSpout extends BaseRichSpout {  
    SpoutOutputCollector _collector;  
    Random _rand;  
  
    @Override  
    public void open(Map conf, TopologyContext context,  
                     SpoutOutputCollector collector) {  
        _collector = collector;  
        _rand = new Random();  
    }  
    @Override  
    public void nextTuple() {  
        Utils.sleep(100);  
        String[] sentences = new String[]{ "the cow jumped over the moon",  
                                           "four score and seven years ago", "snow white and the seven dwarfs"};  
        String sentence = sentences[_rand.nextInt(sentences.length)];  
        _collector.emit(new Values(sentence));  
    }  
    @Override  
    public void declareOutputFields(OutputFieldsDeclarer declarer) {  
        declarer.declare(new Fields("word"));  
    }  
}
```

实现一个  
BaseRichSpout的子类

例子来源： Storm Example Code

# 举例： RandomSentenceSpout

```
public class RandomSentenceSpout extends BaseRichSpout {  
    SpoutOutputCollector _collector;  
    Random _rand;
```

```
    @Override  
    public void open(Map conf, TopologyContext context,  
                     SpoutOutputCollector collector) {  
        _collector = collector;  
        _rand = new Random();  
    }
```

主要是实现nextTuple函数

```
    @Override  
    public void nextTuple() {  
        Utils.sleep(100);  
        String[] sentences = new String[]{ "the cow jumped over the moon",  
                                             "four score and seven years ago", "snow white and the seven dwarfs"};  
        String sentence = sentences[_rand.nextInt(sentences.length)];  
        _collector.emit(new Values(sentence));  
    }
```

```
    @Override  
    public void declareOutputFields(OutputFieldsDeclarer declarer) {  
        declarer.declare(new Fields("word"));  
    }
```

```
}
```

# 举例： RandomSentenceSpout

```
public class RandomSentenceSpout extends BaseRichSpout {  
    SpoutOutputCollector _collector;  
    Random _rand;
```

```
    @Override  
    public void open(Map conf, TopologyContext context,  
                     SpoutOutputCollector collector) {  
        _collector = collector;  
        _rand = new Random();  
    }
```

使用emit向Storm发出一个数据tuple

```
    @Override  
    public void nextTuple() {  
        Utils.sleep(100);  
        String[] sentences = new String[]{ "the cow jumped over the moon",  
                                             "four score and seven years ago", "snow white and the seven dwarfs"};  
        String sentence = sentences[_rand.nextInt(sentences.length)];  
        _collector.emit(new Values(sentence));  
    }
```

```
    @Override  
    public void declareOutputFields(OutputFieldsDeclarer declarer) {  
        declarer.declare(new Fields("word"));  
    }
```

```
}
```

# 举例： RandomSentenceSpout

```
public class RandomSentenceSpout extends BaseRichSpout {  
    SpoutOutputCollector _collector;  
    Random _rand;
```

```
    @Override  
    public void open(Map conf, TopologyContext context,  
                     SpoutOutputCollector collector) {  
        _collector = collector;  
        _rand = new Random();  
    }
```

Open时初始化

```
    @Override  
    public void nextTuple() {  
        Utils.sleep(100);  
        String[] sentences = new String[]{ "the cow jumped over the moon",  
                                           "four score and seven years ago", "snow white and the seven dwarfs"};  
        String sentence = sentences[_rand.nextInt(sentences.length)];  
        _collector.emit(new Values(sentence));  
    }  
    @Override  
    public void declareOutputFields(OutputFieldsDeclarer declarer) {  
        declarer.declare(new Fields("word"));  
    }  
}
```



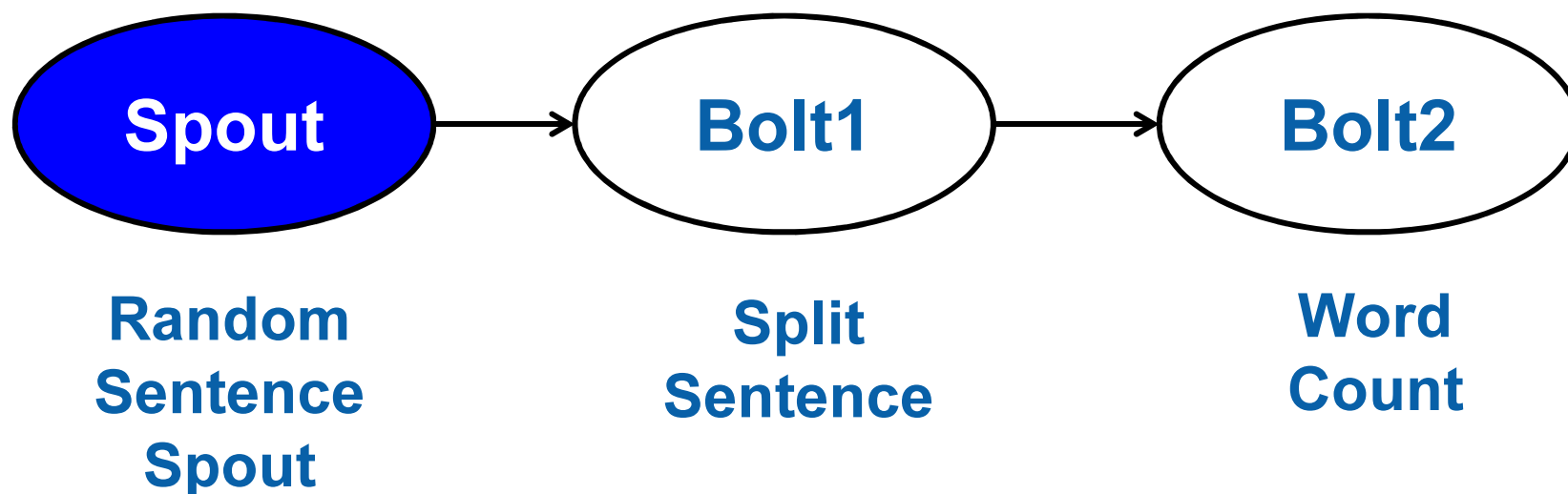
# 举例： RandomSentenceSpout

```
public class RandomSentenceSpout extends BaseRichSpout {
    SpoutOutputCollector _collector;
    Random _rand;

    @Override
    public void open(Map conf, TopologyContext context,
                    SpoutOutputCollector collector) {
        _collector = collector;
        _rand = new Random();
    }
    @Override
    public void nextTuple() {
        Utils.sleep(100);
        String[] sentences = new String[]{ "the cow jumped over the moon",
            "four score and seven years ago", "snow white and the seven dwarfs"};
        String sentence = sentences[_rand.nextInt(sentences.length)];
        _collector.emit(new Values(sentence));
    }
    @Override
    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        declarer.declare(new Fields("word"));
    }
}
```

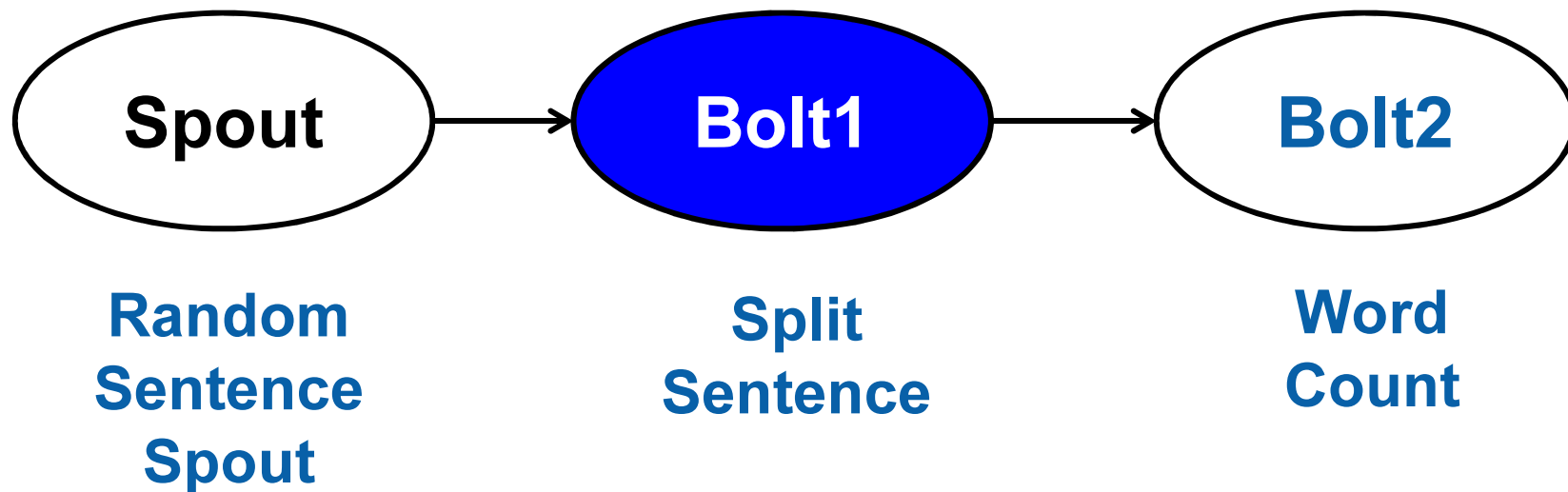
定义输出tuple各属性的名字

# 举例:一个简单的Topology



- Random Sentence Spout:  
每调用一次返回一个随机的句子
- 实际使用时, Spout通常是获得外部数据, emit
  - 例如: 微博的推送

# 举例:一个简单的Topology



- Split Sentence: 把句子分成单词

# 举例： SplitSentence

```
public static class SplitSentence extends BaseBasicBolt {  
    @Override  
    public void execute(Tuple tuple, BasicOutputCollector collector) {  
        String sentence = tuple.getString(0);  
        StringTokenizer itr = new StringTokenizer(sentence);  
        while (itr.hasMoreTokens()) {  
            collector.emit(new Values(itr.nextToken()));  
        }  
    }  
    @Override  
    public void declareOutputFields(OutputFieldsDeclarer declarer) {  
        declarer.declare(new Fields("word"));  
    }  
}
```

实现一个  
BaseBasicBolt的子类



# 举例：SplitSentence

主要实现execute函数，每一个输入tuple被调用一次，用emit发出输出的tuple

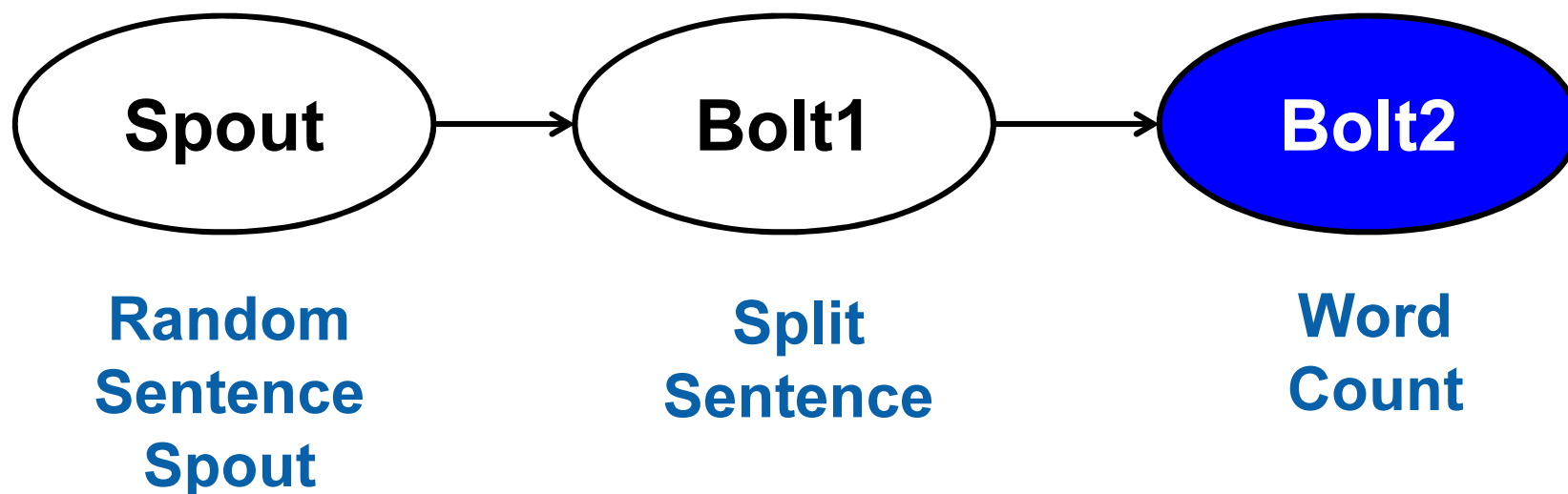
```
public static class SplitSentence extends BasicOutputCollector {
    @Override
    public void execute(Tuple tuple, BasicOutputCollector collector) {
        String sentence = tuple.getString(0);
        StringTokenizer itr = new StringTokenizer(sentence);
        while (itr.hasMoreTokens()) {
            collector.emit(new Values(itr.nextToken()));
        }
    }
    @Override
    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        declarer.declare(new Fields("word"));
    }
}
```

# 举例： SplitSentence

```
public static class SplitSentence extends BaseBasicBolt {
    @Override
    public void execute(Tuple tuple, BasicOutputCollector collector) {
        String sentence= tuple.getString(0);
        StringTokenizer itr = new StringTokenizer(sentence);
        while (itr.hasMoreTokens()) {
            collector.emit(new Values(itr.nextToken()));
        }
    }
    @Override
    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        declarer.declare(new Fields("word"));
    }
}
```

定义输出tuple的属性的名字

# 举例:一个简单的Topology



- Word Count: 单词计数

# 举例：WordCount

## BaseBasicBolt的子类

```
public static class WordCount extends BaseBasicBolt {
    Map<String, Integer> counts = new HashMap<String, Integer>();

    @Override
    public void execute(Tuple tuple, BasicOutputCollector collector) {
        String word = tuple.getString(0);
        Integer count = counts.get(word);
        if (count == null) count = 0;
        count++;
        counts.put(word, count);
        collector.emit(new Values(word, count));
    }

    @Override
    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        declarer.declare(new Fields("word", "count"));
    }
}
```

# 举例： WordCount

每个输入Tuple是一个word，在HashMap中计数，输出当前计数

```
public static class WordCount extends BaseB  
    Map<String, Integer> counts = new HashMap<
```

```
@Override  
public void execute(Tuple tuple, BasicOutputCollector collector) {  
    String word = tuple.getString(0);  
    Integer count = counts.get(word);  
    if (count == null) count = 0;  
    count++;  
    counts.put(word, count);  
    collector.emit(new Values(word, count));  
}
```

```
@Override  
public void declareOutputFields(OutputFieldsDeclarer declarer) {  
    declarer.declare(new Fields("word", "count"));  
}  
}
```

# 举例： WordCount

```
public static class WordCount extends BaseBasicBolt {
    Map<String, Integer> counts = new HashMap<String, Integer>();

    @Override
    public void execute(Tuple tuple, BasicOutputCollector collector) {
        String word = tuple.getString(0);
        Integer count = counts.get(word);
        if (count == null) count = 0;
        count++;
        counts.put(word, count);
        collector.emit(new Values(word, count));
    }
    @Override
    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        declarer.declare(new Fields("word", "count"));
    }
}
```

定义输出tuple的属性的名字



# 举例：主程序

```
public static void main(String[] args) throws Exception {  
  
    TopologyBuilder builder = new TopologyBuilder();  
  
    builder.setSpout("spout", new RandomSentenceSpout(), 5);  
    builder.setBolt("split", new SplitSentence(), 8)  
        .shuffleGrouping("spout");  
    builder.setBolt("count", new WordCount(), 12)  
        .fieldsGrouping("split", new Fields("word"));  
  
    Config conf = new Config();  
    conf.setMaxTaskParallelism(3);  
  
    LocalCluster cluster = new LocalCluster();  
    cluster.submitTopology("word-count", conf, builder.createTopology());  
    Thread.sleep(10000);  
    cluster.shutdown();  
}
```

建立  
Topology

运行



# 举例：主程序

```
public static void main(String[] args) throws Exception {
```

```
    TopologyBuilder builder = new TopologyBuilder
```

创建Spout, 并行度为5

```
    builder.setSpout("spout", new RandomSentenceSpout(), 5);
```

```
    builder.setBolt("split", new SplitSentence(), 8)
```

```
        .shuffleGrouping("spout");
```

```
    builder.setBolt("count", new WordCount(), 12)
```

```
        .fieldsGrouping("split", new Fields("word"));
```

```
    Config conf = new Config();
```

```
    conf.setMaxTaskParallelism(3);
```

```
    LocalCluster cluster = new LocalCluster();
```

```
    cluster.submitTopology("word-count", conf, builder.createTopology());
```

```
    Thread.sleep(10000);
```

```
    cluster.shutdown();
```

```
}
```

# 举例：主程序

```
public static void main(String[] args) throws Exception {  
    TopologyBuilder builder = new TopologyBuilder;  
    builder.setSpout("spout", new RandomSentenceGenerator(), 1);  
    builder.setBolt("split", new SplitSentence(), 8)  
        .shuffleGrouping("spout");  
    builder.setBolt("count", new WordCount(), 12)  
        .fieldsGrouping("split", new Fields("word"));  
  
    Config conf = new Config();  
    conf.setMaxTaskParallelism(3);  
  
    LocalCluster cluster = new LocalCluster();  
    cluster.submitTopology("word-count", conf, builder.createTopology());  
    Thread.sleep(10000);  
    cluster.shutdown();  
}
```

创建Bolt1, 并行度为8,  
连接spout输出

# 举例：主程序

```
public static void main(String[] args) throws Exception {
```

```
    TopologyBuilder builder = new TopologyBuilder();
```

```
    builder.setSpout("spout", new RandomSentenceSpout());
    builder.setBolt("split", new SplitSentence(
        .shuffleGrouping("spout");
```

创建Bolt2, 并行度为12,  
连接Bolt1输出

```
    builder.setBolt("count", new WordCount(), 12)
        .fieldsGrouping("split", new Fields("word"));
```

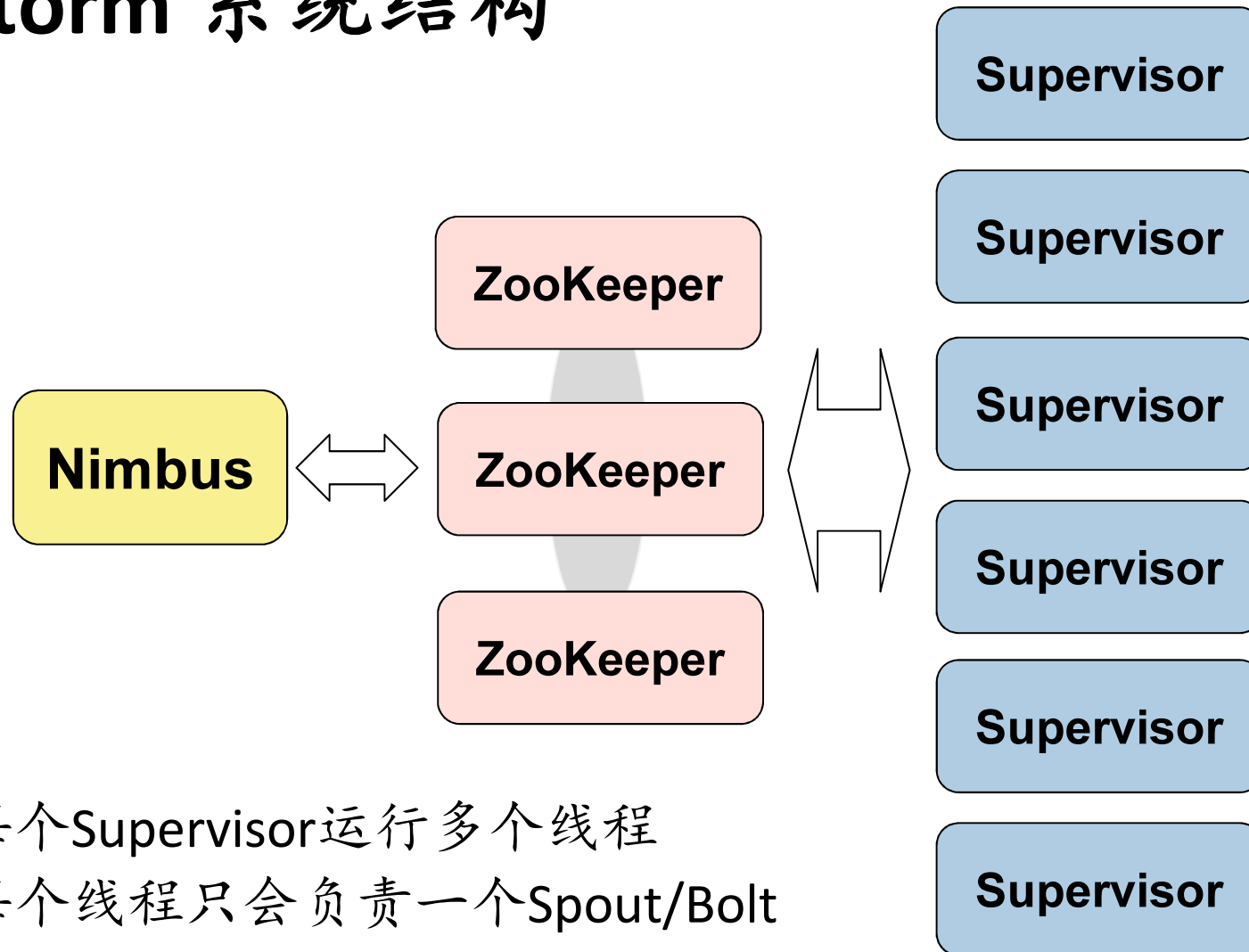
```
    Config conf = new Config();
    conf.setMaxTaskParallelism(3);
```

```
    LocalCluster cluster = new LocalCluster();
    cluster.submitTopology("word-count", conf, builder.createTopology());
    Thread.sleep(10000);
    cluster.shutdown();
}
```

# Stream Grouping

- Shuffle grouping: 随机
- Fields grouping: group-by shuffle
- 其它种类

# Storm 系统结构



- 每个Supervisor运行多个线程
- 每个线程只会负责一个Spout/Bolt
- 一个Spout/Bolt可以对应多个Supervisor和多个线程