大数据Hadoop高薪直通车课程

Spark 核心RDD

讲师：轩宇（北风网版权所有)

课程大纲

1. Spark RDD 特性
2. Spark RDD 操作
3. Spark RDD 依赖
4. Spark RDD Shuffle
5. Spark 内核分析

课程大纲

1. Spark RDD 特性
2. Spark RDD 操作
3. Spark RDD 依赖
4. Spark RDD Shuffle
5. Spark 内核分析

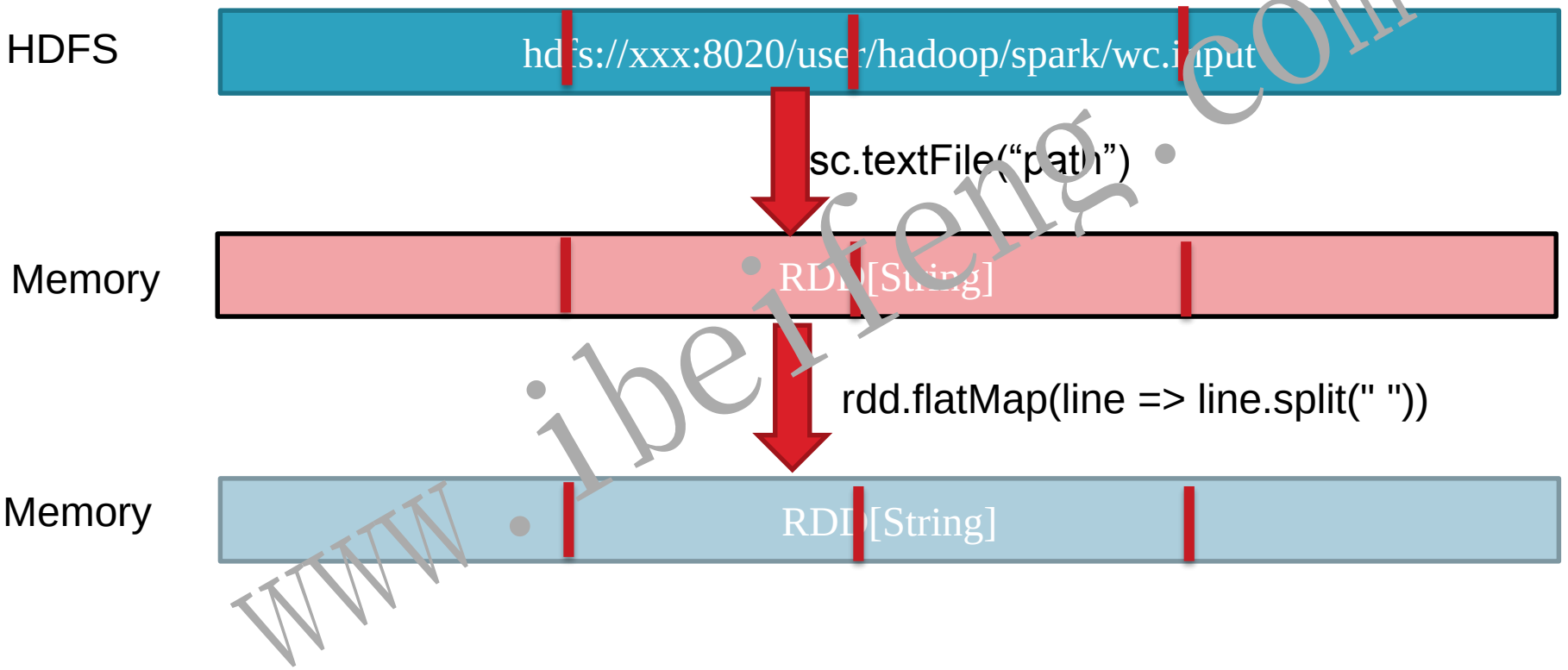
Spark WordCount

```
val rdd=sc.textFile("hdfs://bigdata-cdh01.ibeifeng.com :8020/user/beifeng/spark/wc.input")
```

```
val wordcount=rdd.flatMap(_.split(" "))
                 .map((_,1))
                 .reduceByKey(_ + _)
wordcount.collect()
```

```
val wordsort=wordcount.map(x=>(x._2,x._1))
                      .sortByKey(false)
                      .map(x=>(x._2,x._1))
wordsort.collect()
```

RDD



RDD

A Resilient Distributed Dataset (RDD), the **basic abstraction** in Spark.

Represents an **immutable**, **partitioned** collection of elements

that can be operated on **in parallel**.

```
Internally, each RDD is characterized by five main properties:

 - A list of partitions
 - A function for computing each split
 - A list of dependencies on other RDDs
 - Optionally, a Partitioner for key-value RDDs (e.g. to say that the RDD is hash-partitioned)
 - Optionally, a list of preferred locations to compute each split on (e.g. block locations for
   an HDFS file)
```

RDD

RDD: Resilient Distributed Dataset

RDD的特点：

1、A list of **partitions**

一系列的分片：比如说64M一片；类似于Hadoop中的split；

2、A **function** for computing each split

在每个分片上都有一个函数去迭代/执行/计算它

3、A list of **dependencies** on other RDDs

一系列的依赖：RDDa转换为RDDb，RDDb转换为RDDc，那么RDDc就依赖于RDDb，RDDb就依赖于RDDa

4、Optionally, a **Partitioner** for key-value RDDs (e.g. to say that the RDD is hash-partitioned)

对于key-value的RDD可指定一个partitioner，告诉它如何分片；常用的有hash，range

5、Optionally, a list of **preferred location(s)** to compute each split on (e.g. block locations for an HDFS file)

要运行的计算/执行最好在哪(几)个机器上运行。数据本地性。

为什么会有哪几个呢？

比如：hadoop默认有三个位置，或者spark cache到内存是可能通过StorageLevel设置了多个副本，所以一个partition可能返回多个最佳位置。

课程大纲

1. Spark RDD 特性
2. Spark RDD 操作
3. Spark RDD 依赖
4. Spark RDD Shuffle
5. Spark 内核分析

Create RDDs

- Parallelized Collections
- External Datasets

Parallelized Collections

Parallelized collections are created by calling `SparkContext`'s `parallelize` method on an existing collection in your driver program (a Scala `Seq`). The elements of the collection are copied to form a distributed dataset that can be operated on in parallel. For example, here is how to create a parallelized collection holding the numbers 1 to 5:

```
val data = Array(1, 2, 3, 4, 5)
val distData = sc.parallelize(data)
```

Once created, the distributed dataset (`distData`) can be operated on in parallel. For example, we might call `distData.reduce((a, b) => a + b)` to add up the elements of the array. We describe operations on distributed datasets later on.

External Datasets

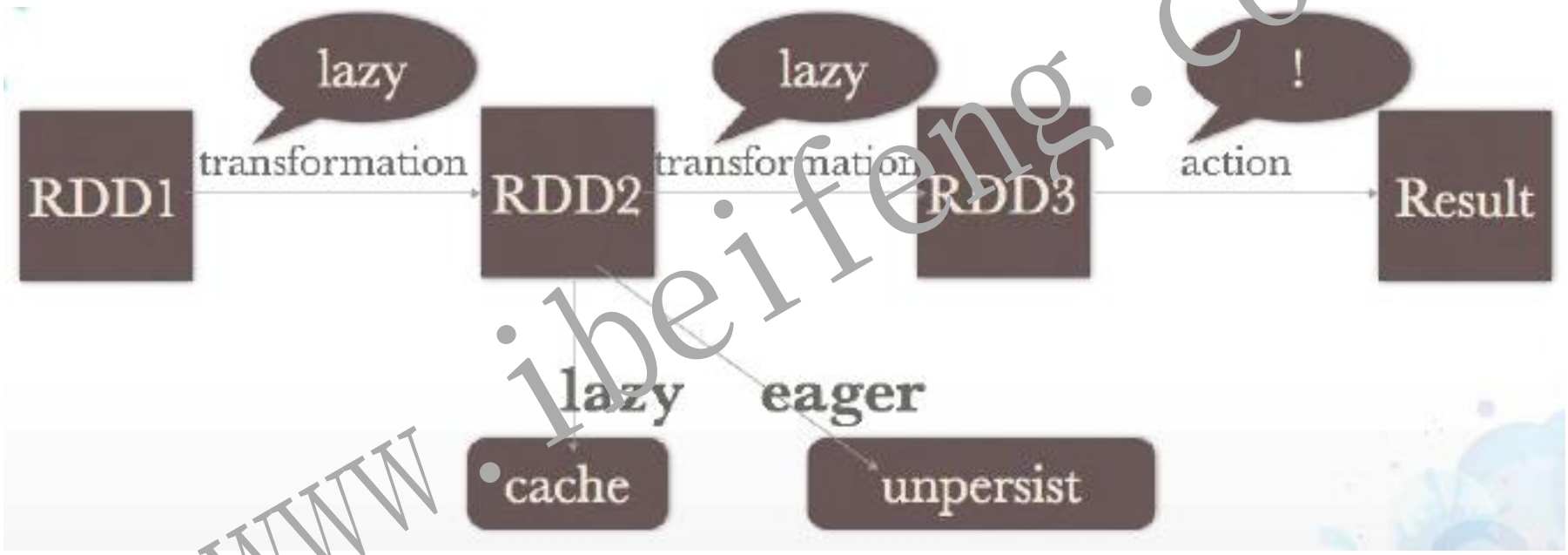
Spark can create distributed datasets from any storage source supported by Hadoop, including your local file system, HDFS, Cassandra, HBase, Amazon S3, etc. Spark supports text files, SequenceFiles, and any other Hadoop InputFormat.

Text file RDDs can be created using `SparkContext`'s `textFile` method. This method takes an URI for the file (either a local path on the machine, or a `hdfs://`, `s3n://`, etc URI) and reads it as a collection of lines. Here is an example invocation:

```
scala> val distFile = sc.textFile("data.txt")
distFile: RDD[String] = MappedRDD@1d4cee08
```

Once created, `distFile` can be acted on by dataset operations. For example, we can add up the sizes of all the lines using the `map` and `reduce` operations as follows: `distFile.map(s =>`

RDD



Resilient Distributed Datasets

- Resilient Distributed Datasets (RDDs)
  - Parallelized Collections
  - External Datasets
  - RDD Operations
    - Basics
    - Passing Functions to Spark
    - Understanding closures
      - Example
      - Local vs. cluster modes
      - Printing elements of an RDD
    - Working with Key-Value Pairs
    - Transformations
    - Actions
    - Shuffle operations
      - Background
      - Performance Impact
  - RDD Persistence
    - Which Storage Level to Choose?
    - Removing Data

http://spark.apache.org/docs/1.3.0/programming-guide.html

RDD

Transformations

•Create a new dataset from and existing one.

•**Lazy** in nature. They are executed only when some action is performed.

•Example :
- map(func)
- filter(func)
- distinct() …

Actions

•Returns to the driver program a value or exports data to a storage system after performing a computation.

•Example:
- count()
- reduce(func)
- collect
- take()…

Persistence

•For caching datasets in memory for future operations.

•Option to store on disk or RAM or mixed (Storage Level).

•Example:
- persist()
- cache()

Transformation

Create new datasets from existing ones

map()	intersection()	cartesion()
flatMap()	distinct()	pipe()
filter()	**groupByKey()**	coalesce()
mapPartitions()	**reduceByKey()**	repartition()
mapPartitionsWithIndex()	**sortByKey()**	partitionBy()
sample()	**join()**	...
union()	cogroup	...

✓**Transformations aren't applied to an RDD until an action is executed;**
✓**Spark remembers set of transformations applied to base dataset;**

Action

reduce()

collect()

count()

first()

take()

takeSample()

saveToCassandra()

takeOrdered()

saveAsTextFile()

saveAsSequenceFile()

saveAsObjectFile()

countByKey()

foreach()

...

✓**Cause Spark to execute recipe to transform source;**
✓**Cause data to be returned to driver or saved to output;**

RDD Persistence

One of the most important capabilities in Spark is *persisting* (or *caching*) a dataset in memory across operations. When you persist an RDD, each node stores any partitions of it that it computes in memory and reuses them in other actions on that dataset (or datasets derived from it). This allows future actions to be much faster (often by more than 10x). Caching is a key tool for iterative algorithms and fast interactive use.

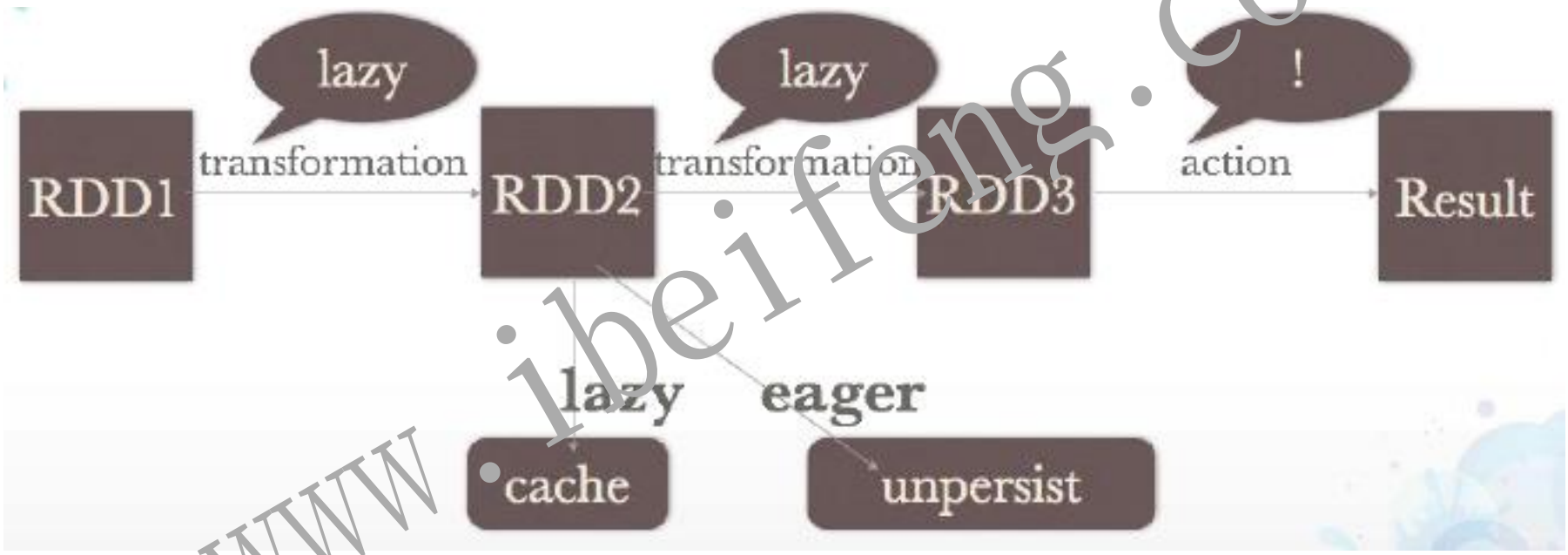
You can mark an RDD to be persisted using the `persist()` or `cache()` methods on it. The first time it is computed in an action, it will be kept in memory on the nodes. Spark's cache is fault-tolerant – if any partition of an RDD is lost, it will automatically be recomputed using the transformations that originally created it.

In addition, each persisted RDD can be stored using a different *storage level*, allowing you, for example, to persist the dataset on disk, persist it in memory but as serialized Java objects (to save space), replicate it across nodes, or store it off-heap in Tachyon. These levels are set by passing a `StorageLevel` object (Scala, Java, Python) to `persist()`. The `cache()` method is a shorthand for using the default storage level, which is `StorageLevel.MEMORY_ONLY` (store deserialized objects in memory). The full set of storage levels is:

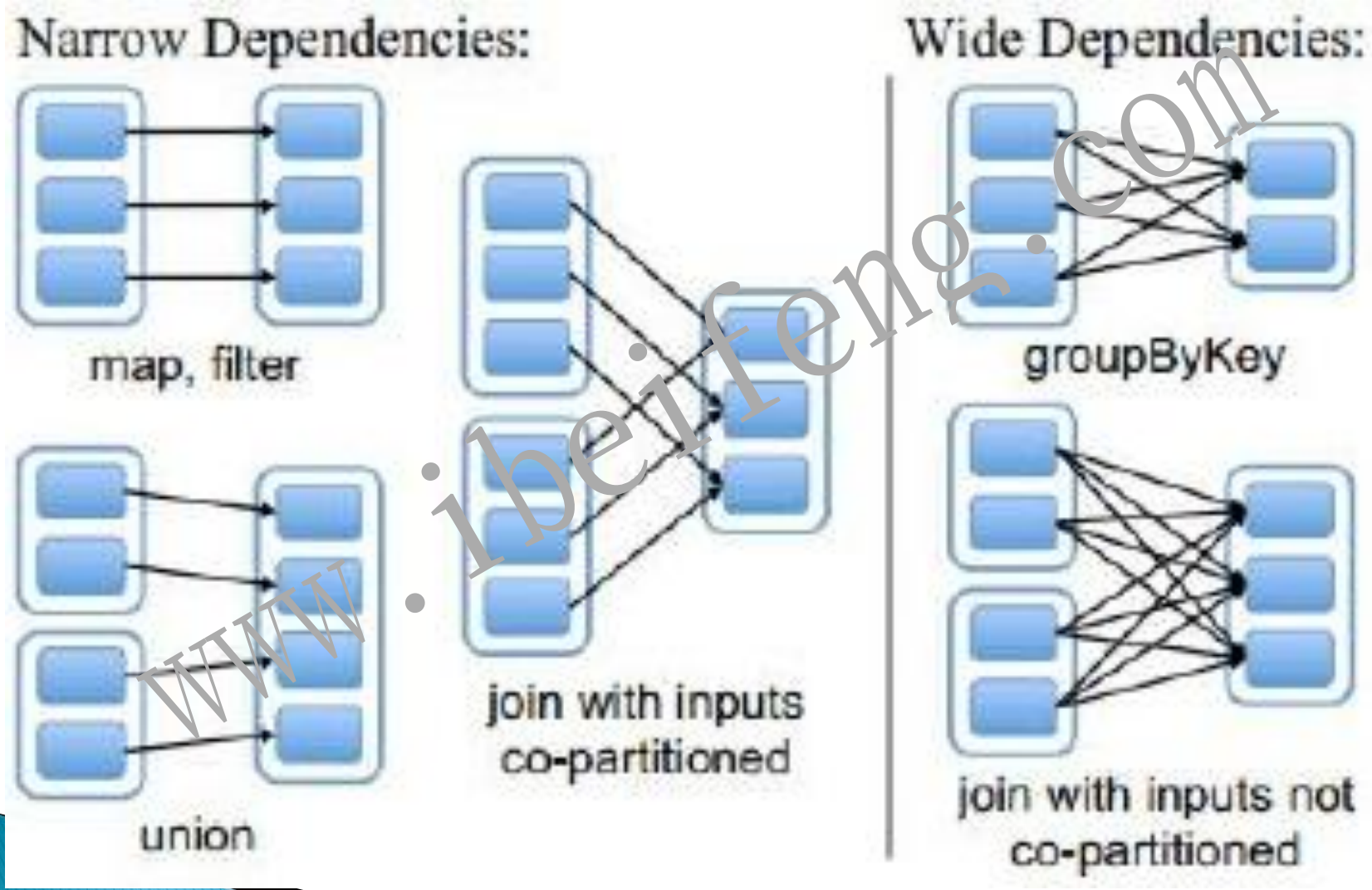
课程大纲

1. Spark RDD 特性
2. Spark RDD 操作
3. Spark RDD 依赖
4. Spark RDD Shuffle
5. Spark 内核分析

RDD



RDD Dependencies



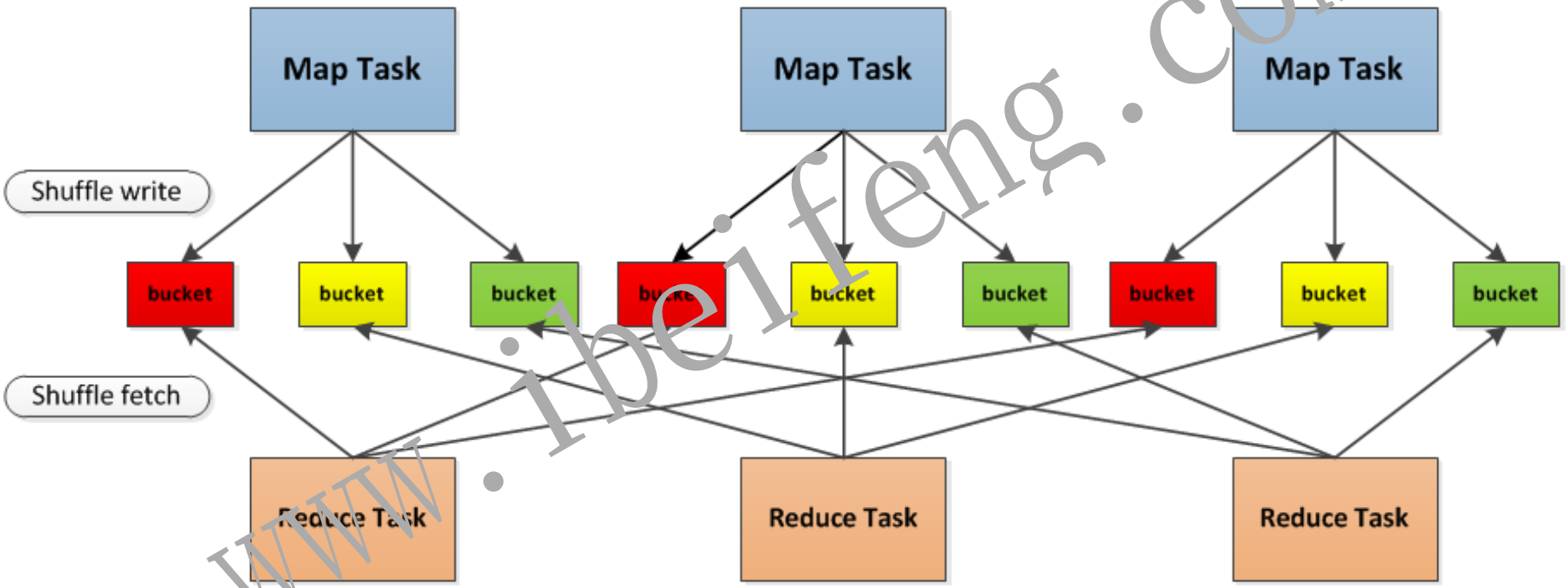
RDD Dependencies

- 窄依赖（ narrow dependencies ）
  - 子 RDD 的每个分区依赖于常数个父分区（即与数据规模无关）
  - 输入输出一对一的算子，且结果 RDD 的分区结构不变，主要是 map 、 flatMap
  - 输入输出一对一，但结果 RDD 的分区结构发生了变化，如 union 、 coalesce
  - 从输入中选择部分元素的算子，如 filter 、 distinct 、 subtract 、 sample
- 宽依赖（ wide dependencies ）
  - 子 RDD 的每个分区依赖于所有父 RDD 分区
  - 对单个 RDD 基于 key 进行重组和 reduce ，如 groupByKey 、 reduceByKey ；
  - 对两个 RDD 基于 key 进行 join 和重组，如 join

课程大纲

1. Spark RDD 特性
2. Spark RDD 操作
3. Spark RDD 依赖
4. **Spark RDD Shuffle**
5. Spark 内核分析

Spark Shuffle



RDD Shuffle

- 什么是spark Shuffle
  - ✓ The shuffle is Spark's mechanism for re-distributing data
- 那些操作会引起Shuffle？
  - ✓ 具有重新调整分区操作， eg：repartition,coalesce
  - ✓ *BeyKey eg: groupByKey ,reduceByKey
  - ✓ 关联操作 eg： join ， cogroup

课程大纲

1. Spark RDD 特性
2. Spark RDD 操作
3. Spark RDD 依赖
4. Spark RDD Shuffle
5. Spark 内核分析

Initializing Spark

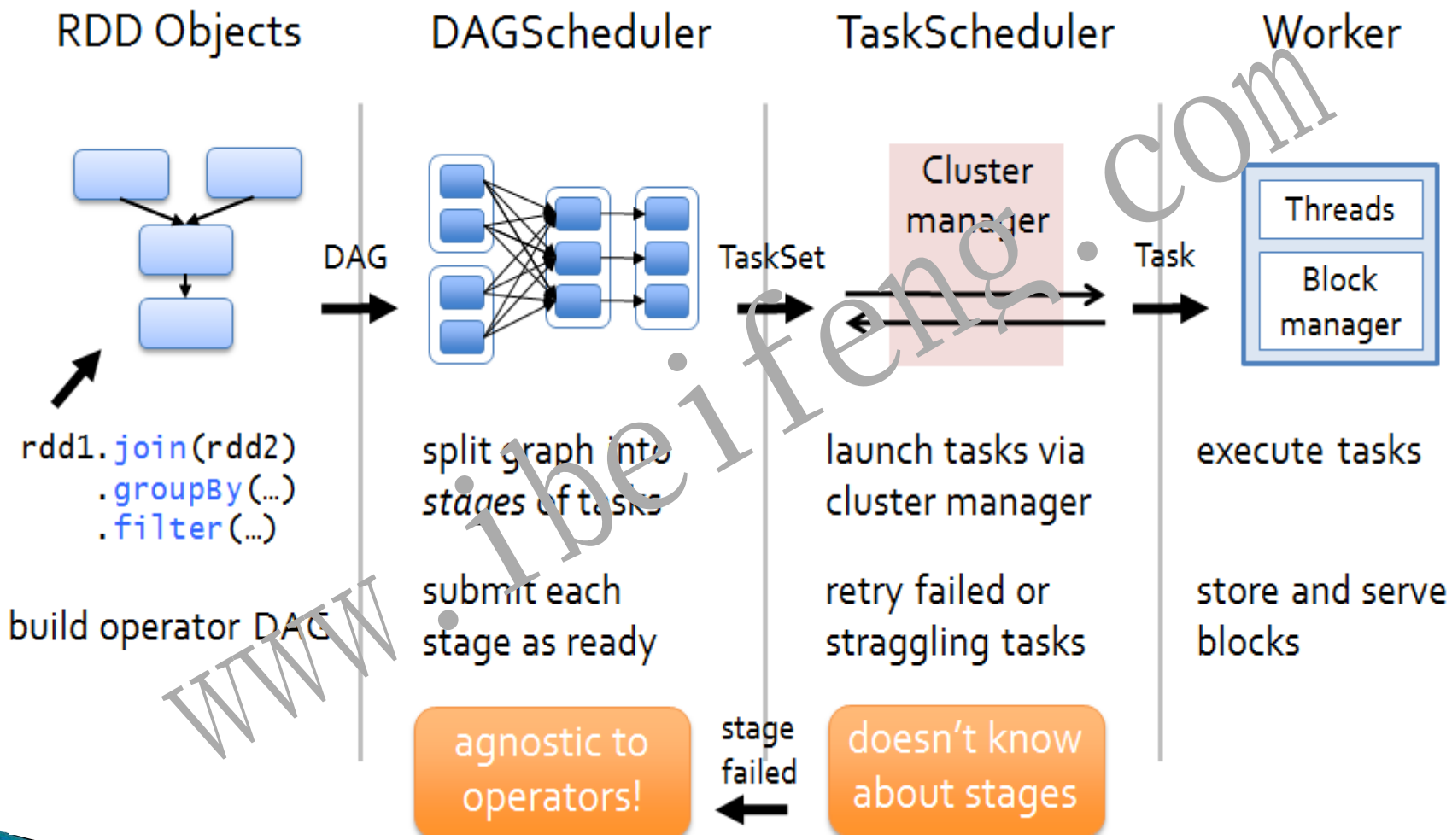
The first thing a Spark program must do is to create a SparkContext object, which tells Spark how to access a cluster. To create a `SparkContext` you first need to build a SparkConf object that contains information about your application.

Only one SparkContext may be active per JVM. You must `stop()` the active SparkContext before creating a new one.

```
val conf = new SparkConf().setAppName(appName).setMaster(master)
new SparkContext(conf)
```

The `appName` parameter is a name for your application to show on the cluster UI. `master` is a Spark, Mesos or YARN cluster URL, or a special "local" string to run in local mode. In practice, when running on a cluster, you will not want to hardcode `master` in the program, but rather launch the application with `spark-submit` and receive it there. However, for local testing and unit tests, you can pass "local" to run Spark in-process.

Spark Scheduler



DAG Scheduler

DAGScheduler

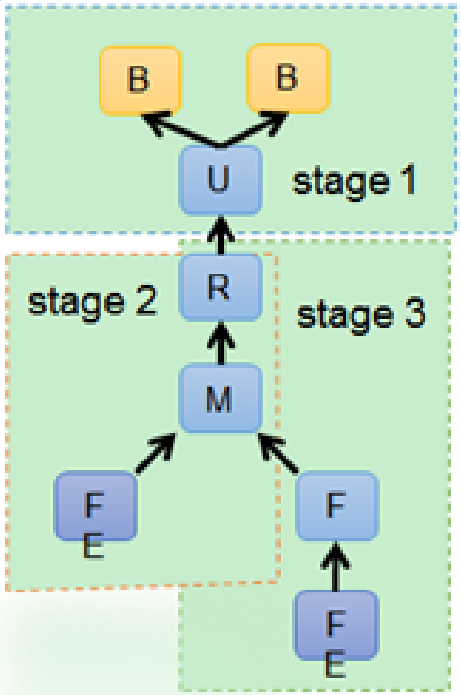
Spark program

```
val lines1 = sc.textFile(inputPath1)
val lines2 = sc.textFile(inputPath2)

t = t1.union(t2).map(…).reduce(…)

t.saveAsHadoopFiles(…)
t.filter(…).foreach(…)
```

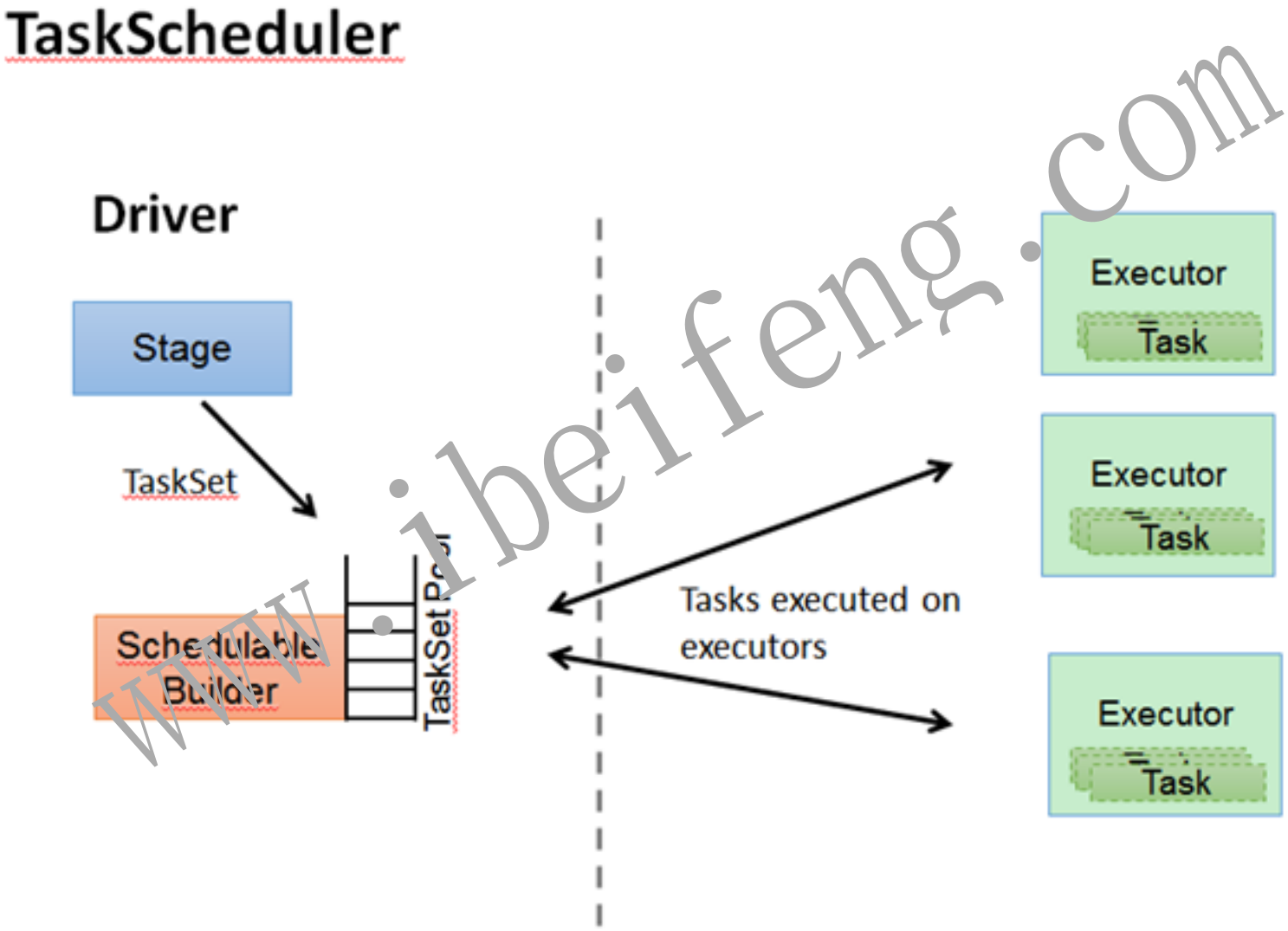
RDD Graph



DAG Scheduler

- 接收用户提交的job
- 构建 Stage，记录哪个 RDD 或者 Stage 输出被物化
- 重新提交 shuffle 输出丢失的 stage
- 将 Taskset 传给底层调度器

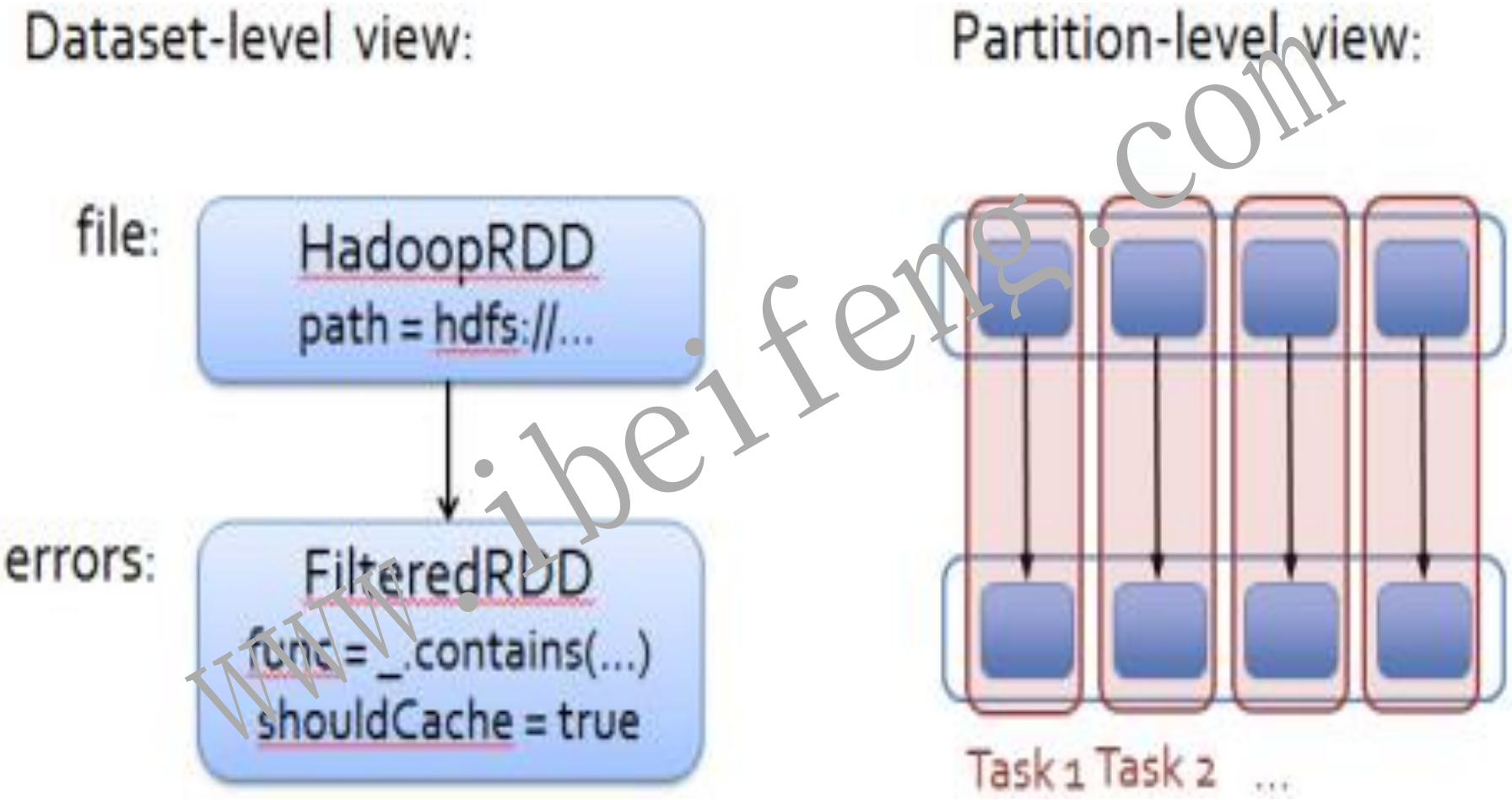
Task Scheduler



Task Scheduler

- 提交 taskset(一组 task) 到集群运行并监控
- 为每一个 TaskSet 构建一个 TaskSetManager 实例管理这个 TaskSet 的生命周期
- 数据本地性决定每个 Task 最佳位置 (process local, node-local, rack-local and then any)
- 推测执行，碰到 straggle 任务需要放到别的节点上重试出现 shuffle 输出 lost 要报告 fetch failed 错误

Partition & Task



Task

- Task是Executor中的执行单元
- Task处理数据常见的两个来源：外部存储以及shuffle数据
- Task可以运行在集群中的任意一个节点上
- 为了容错，会将shuffle输出写到磁盘或者内存中

Spark 案例分析

- 排序

  WordCount 程序，依据词频降序

- TOP KEY

  WordCount 程序，前KEY值

- 二次排序（作业）

  MapReduce 中的两次排序思路

本课程版权归北风网所有

欢迎访问我们的官方网站

www.ibeifeng.com