Lab4 - Spark Programming

NTHU 2015 Spring
LSA Lab Tony Wei

Outline

- Introduction to Scala & functional programming
- Tour of Spark operations
- Example - PageRank
- Lab Assignment
- Reference

Outline

- Introduction to Scala & functional programming
- Tour of Spark operations
- Example - PageRank
- Lab Assignment
- Reference

About Scala

- High-level language for the java VM
  - Object-oriented + functional programming
- Statically typed
  - Comparable in speed to Java
  - But often no need to write types due to type inference
- Interoperates with Java
  - Can use any Java class, inherit from it, etc; can also call Scala code from Java

Quick Tour

- Declaring variables:

```
var x: Int = 7
var x = 7 //type inferred
val y = "hi" //read-only
```

- Java equivalent:

```
int x = 7;

final String y = "hi";
```

- Functions:

```
def square(x: Int): Int = x*x
def square(x: Int) :Int = {
    x*x
}
def announce(text: String) {
    println(text)
}
```

- Java equivalent:

```
int square(int x) {
    return x*x;
}

void announce(String text) {
    System.out.println(text);
}
```

Quick Tour

- Generic types:

```
var arr = new Array[Int](8)

var lst = List(1, 2, 3)
```

- Java equivalent:

```
int[] arr = new int[8];

List<Integer> lst =
        new ArrayList<Integer>();
list.add(…);
```

- Indexing:

```
arr(5) = 7
println(lst(5))
```

- Java equivalent:

```
arr[5] = 7;
System.out.println(lst.get(5));
```

Quick Tour

- Scala pair syntax:

```
val pair = (a,b)
pair._1
pair._2
```

- Using hierarchical pair

```
val x = 10
val y = 5
val key = pair(x,y)
val value = “something”
val key_value = pair(key,value)
```

- Java equivalent:

```
Pair pair = new Pair(a,b);
pair.getFirst();
pair.getSecond();
```

Quick Tour

- Case Classes: (use it to define your own type)

```
abstract class Type
case class MyInt(ele: Int) extends Type
case class MyDouble(ele: Double) extends Type{
        override def equals(o: Any) = o match {
                case that: MyDouble => /*your equal function*/
                case _ : false
        }
}
def printType(t: Type) {
        t match {
                case MyInt(x) => println("Int " + x)
                case MyDouble(x) => println("Double " + x)
        }
}
```

Quick Tour

- Processing collections with functional programming:

```
val list = List(1, 2, 3)
list.foreach( x => println(x) )          // prints 1, 2, 3
list.foreach(println)

list.map(x => x + 2)                     // => List(3, 4, 5)
list.map(_ + 2)

list.filter(x => x%2 == 1)               // => List(1, 3)
list.filter(_ %2 == 1)

list.reduce((x,y) => x+y)                // => 6
list.reduce(_ + _)
```

Scala Closure Syntax

```
(x: Int) => x + 2          // full version
x => x + 2                 // type inferred
_ + 2                      // when each argument is used exactly one
x => {                     // when body is a block of code
    val numberToAdd = 2
    x + numberToAdd
}

//If closure is too, long, can always pass a function
def addTwo(x: Int): Int = x+ 2
list.map(addTwo)
```

Other Collection Methods

Method on Seq[T]	Explanation
`map(f: T => U): Seq[U]`	Pass each element through f
`flatMap(f: T => Seq[U]): Seq[U]`	One-to-many map
`filter(f: T => Boolean): Seq[T]`	Keep elements passing f
`exists(f: T => Boolean): Boolean`	True if one element passes
`forall(f: T => Boolean): Boolean`	True if all elements pass
`reduce(f: (T, T) => T): T`	Merge elements using f
`groupBy(f: T => K): Map[K,List[T]]`	Group elements by f(element)
`sortBy(f: T => K): Seq[T]`	Sort elements by f(element)
. . .	

Outline

- Introduction to Scala & functional programming
- Tour of Spark operations
- Example - PageRank
- Lab Assignment
- Reference

Spark Overview

- **Goal: work with distributed collections as you would with local ones**
- Concept: resilient distributed datasets (RDDs)
  - Immutable collections of objects spread across a cluster
  - Built through parallel transformations (map, filter, etc)
  - Automatically rebuilt on failure
  - Controllable persistence (e.g. caching in RAM) for reuse

Spark Overview

- Resilient distributed datasets (RDDs)
  - Immutable, partitioned collections of objects
- Transformations (e.g. map, filter, groupBy, join)
  - Lazy operations to build RDDs from other RDDs
- Actions (e.g. count, collect, save)
  - Return a result or write it to storage

SparkContext

- Main entry point to Spark functionality
- val conf = new SparkConf().setAppsName(“Test”)
- val sc = new SparkContext(conf)

Creating RDDs

- Turn a Scala collection into an RDD
  - sc.parallelize(List(1, 2, 3))
- Load text file from local FS, HDFS or S3
  - sc.textFile("file.txt")
  - sc.textFile("directory/*.txt")
  - sc.textFile("hdfs://<namenode>:<port>/*$path*")
- Use any existing Hadoop InputFormat
  - sc.hadoopFile(keyClass, valClass, inputFmt, conf)

Basic Transformations

- val nums = sc.parallelize(List(1, 2, 3))
- val squares = nums.map(x => x*x) // {1, 4, 9}
- val even = squares.filter(_ % 2 == 0) // {4}
- nums.flatMap(x => 1 to x) // {1, 1, 2, 1, 2, 3}

Basic Actions

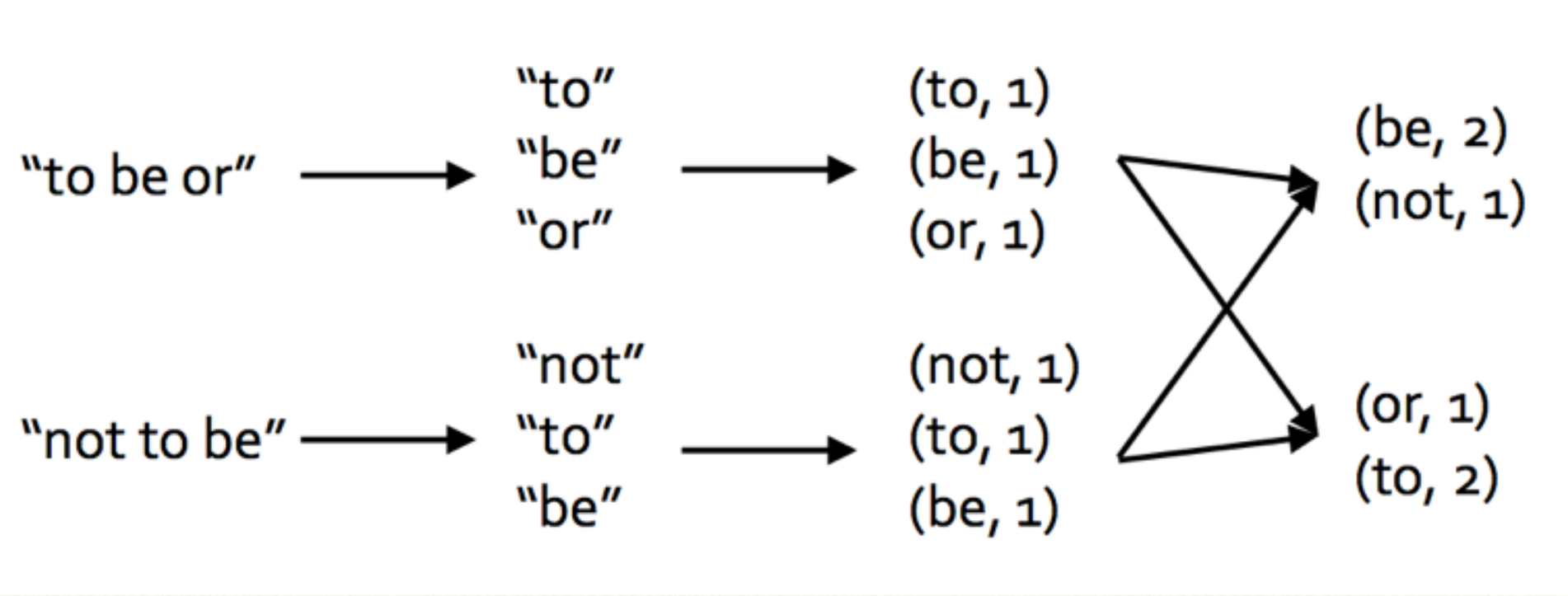
- val nums = sc.parallelize(List(1, 2, 3))
- nums.collect() // => Array(1, 2, 3)
- nums.take(2) // => Array(1, 2)
- nums.count() // => 3
- nums.reduce(_ + _) // => 6
- nums.saveAsTextFile("hdfs://<namenode>:<port>/file.txt")

Working with Key-Value Pairs

- val pets = sc.parallelize(List(("cat", 1), ("dog", 1), ("cat", 2)))
- pets.reduceByKey(_ +_) // => {(cat, 3), (dog, 1)}
- pets.groupByKey() // => {(cat, Seq(1, 2)), (dog, Seq(1))}
- pets.sortByKey() // => {(cat, 1), (cat, 2), (dog, 1)}
-

Example: Word Count

- val lines = sc.textFile("hamlet.txt")
- val counts = lines.flatMap(line => line.split(" "))
  .map(word => (word, 1))
  .reduceByKey(_ + _)



Other Key-Value Operations (Join by Key)

- val visits = sc.parallelize(List(
  (“index.html”, “1.2.3.4”),
  (“about.html”, “3.4.5.6”),
  (“index.html”, “1.3.3.1”)))
- val pageNames = sc.parallelize(List(
  (“index.html”, “Home”), (“about.html”, “About”)))
- visit.join(pageNames)

// (“index.html”, (“1.2.3.4”, “Home”))
// (“index.html”, (“1.3.3.1”, “Home”))
// (“about.html”, (3.4.5.6”, “About”))

Controlling The Number of Reduce Tasks

- All the pair RDD operations take an optional second parameter for number of tasks
- words.reduceByKey(_ + _ , 5)
- words. groupByKey(5)
- visits.join(pageViews. 5)
- Can also set spark.default.parallelism property

Other RDD Operations

- sample(): deterministically sample a subset
- union(): merge two RDDs
- cache(): persist with default storage level
- distinct(): return a new RDD containing the distinct elements
- mapValues(f: (V) => U)
- See Programming Guide for more:

  www.spark-project.org/documentation.html

Outline

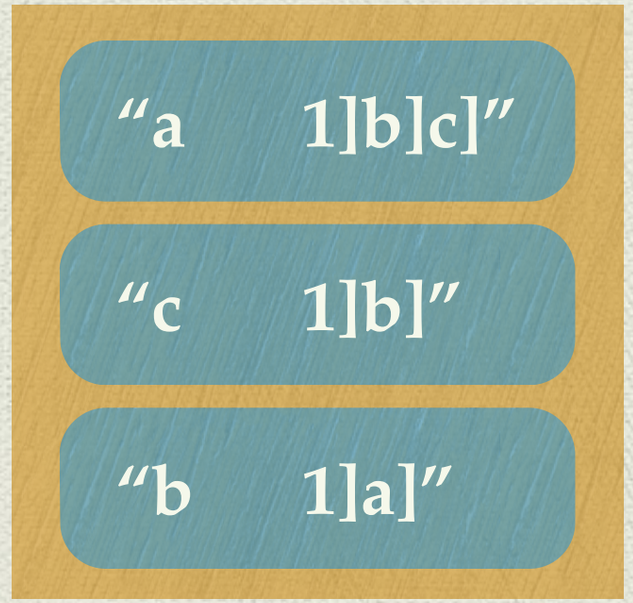
- Introduction to Scala & functional programming
- Tour of Spark operations
- Example - PageRank
- Lab Assignment
- Reference

Example - PageRank

```
val lines = sc.textFile("link.txt")
val links = lines.map{ s =>
    val parts = s.split("\\t")
    (parts(0), parts(1))
}.flatMapValues(_.split("]")).filter{ x =>
    !x._2.equals("") && !x._2.equals("1")
}.groupByKey().cache()
```

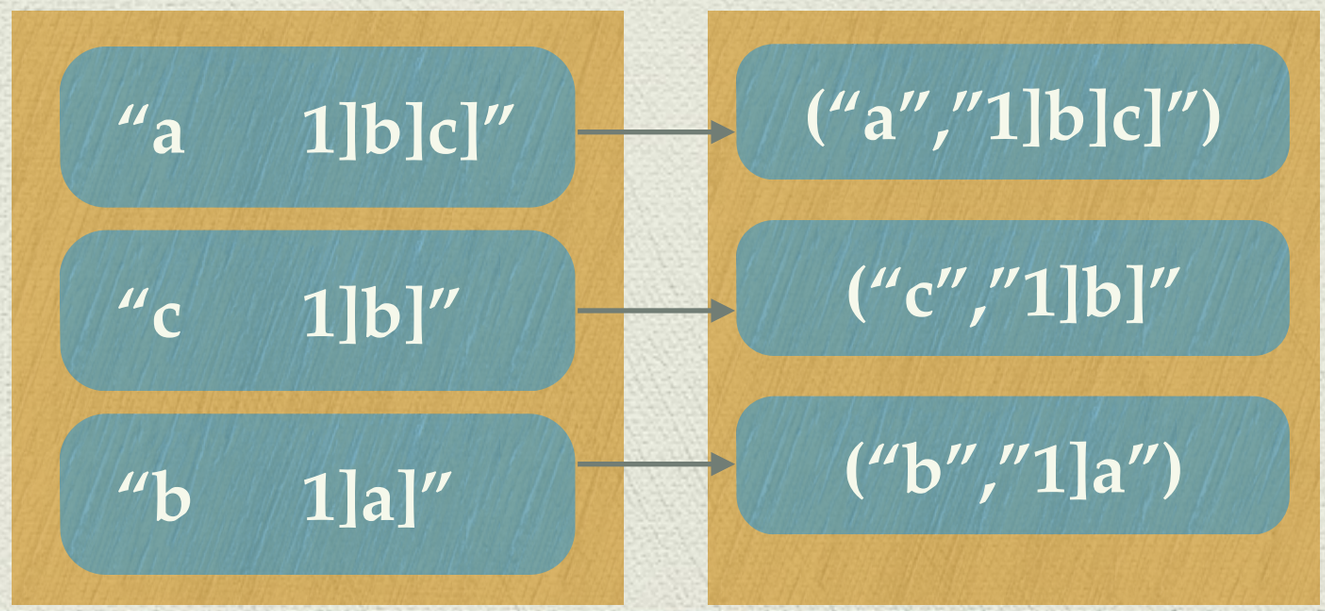
Example - PageRank

```
val lines = sc.textFile("link.txt")
val links = lines.map{ s =>
    val parts = s.split("\\t")
    (parts(0), parts(1))
}.flatMapValues(_.split("]")).filter{ x =>
    !x._2.equals("") && !x._2.equals("1")
}.groupByKey().cache()
```



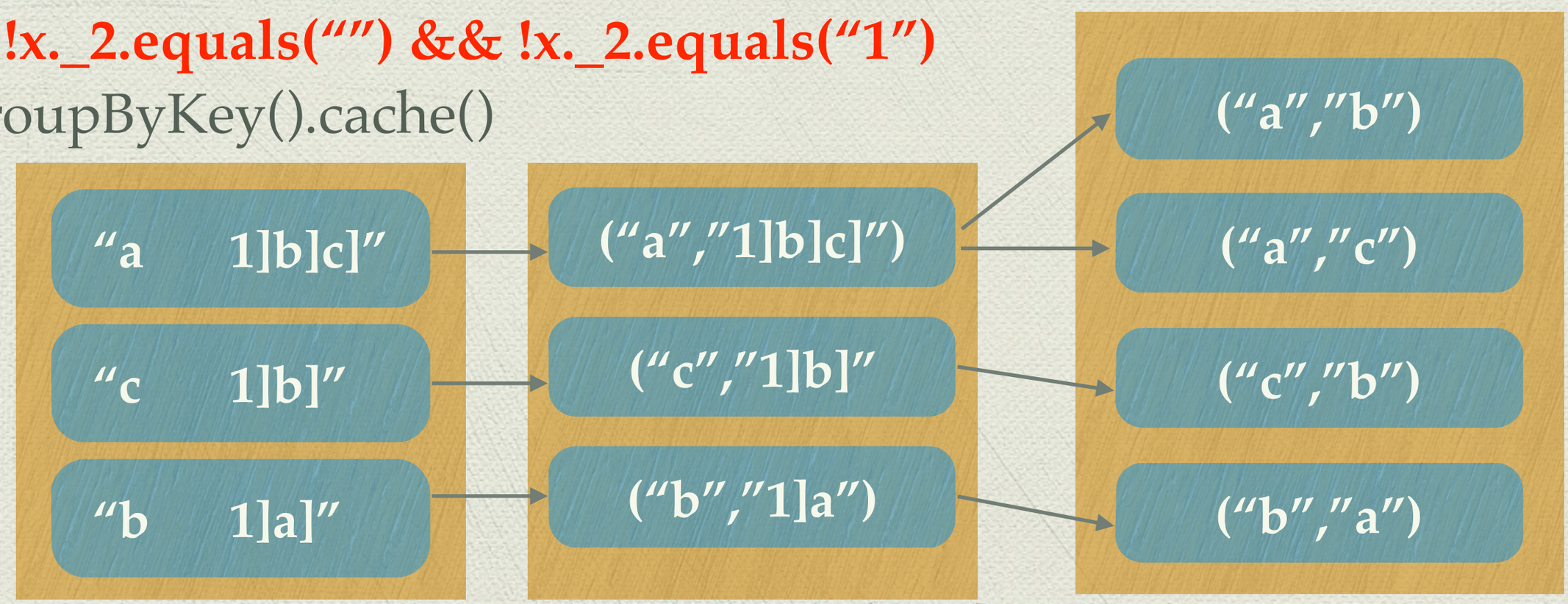
Example - PageRank

```
val lines = sc.textFile("link.txt")
val links = lines.map{ s =>
    val parts = s.split("\\t")
    (parts(0), parts(1))
}.flatMapValues(_.split("]")).filter{ x =>
    !x._2.equals("") && !x._2.equals("1")
}.groupByKey().cache()
```



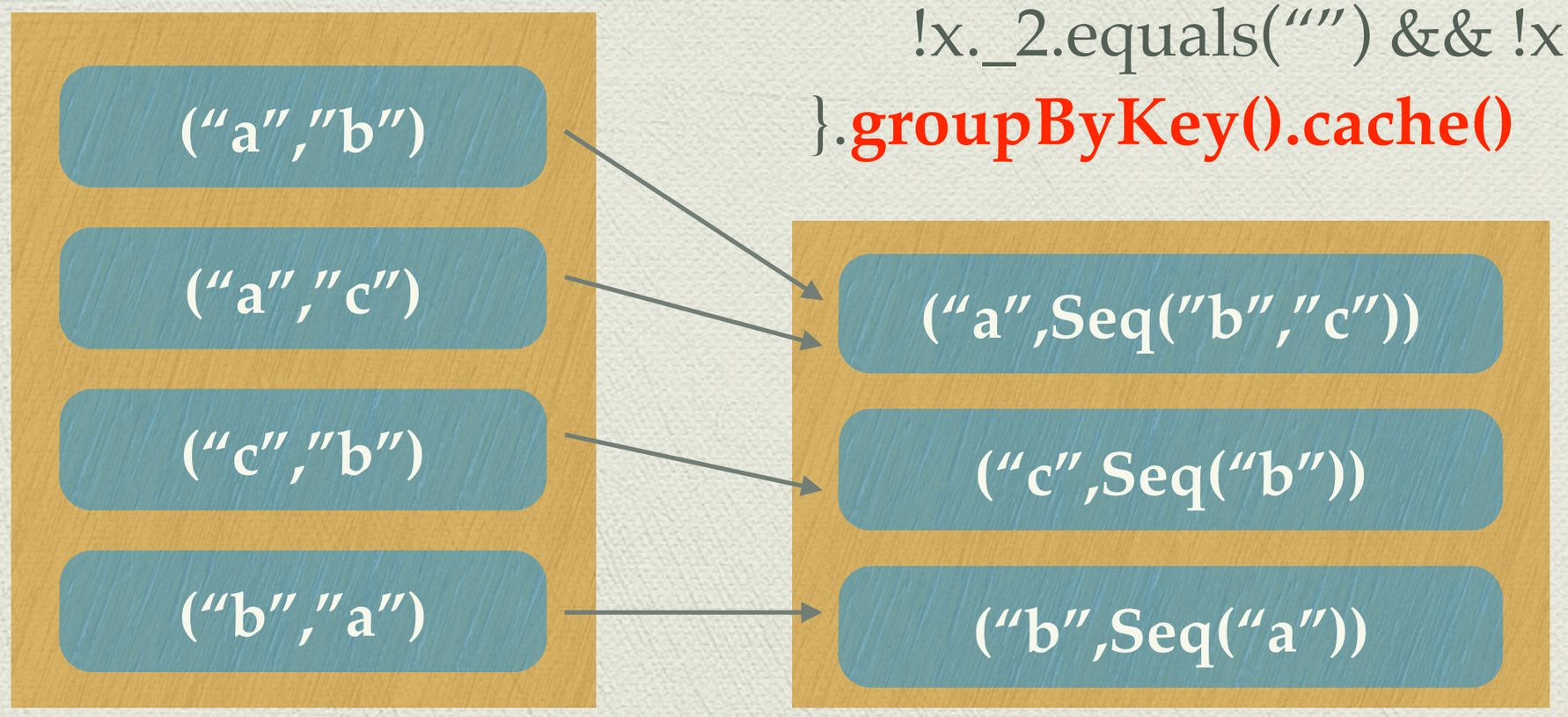
Example - PageRank

```
val lines = sc.textFile("link.txt")
val links = lines.map{ s =>
    val parts = s.split("\\t")
    (parts(0), parts(1))
}.flatMapValues(_.split("]")).filter{ x =>
    !x._2.equals("") && !x._2.equals("1")
}.groupByKey().cache()
```



Example - PageRank

```
val lines = sc.textFile("link.txt")
val links = lines.map{ s =>
    val parts = s.split("\ \t")
    (parts(0), parts(1))
}.flatMapValues(_.split("]")).filter{ x =>
    !x._2.equals("") && !x._2.equals("1")
}.groupByKey().cache()
```



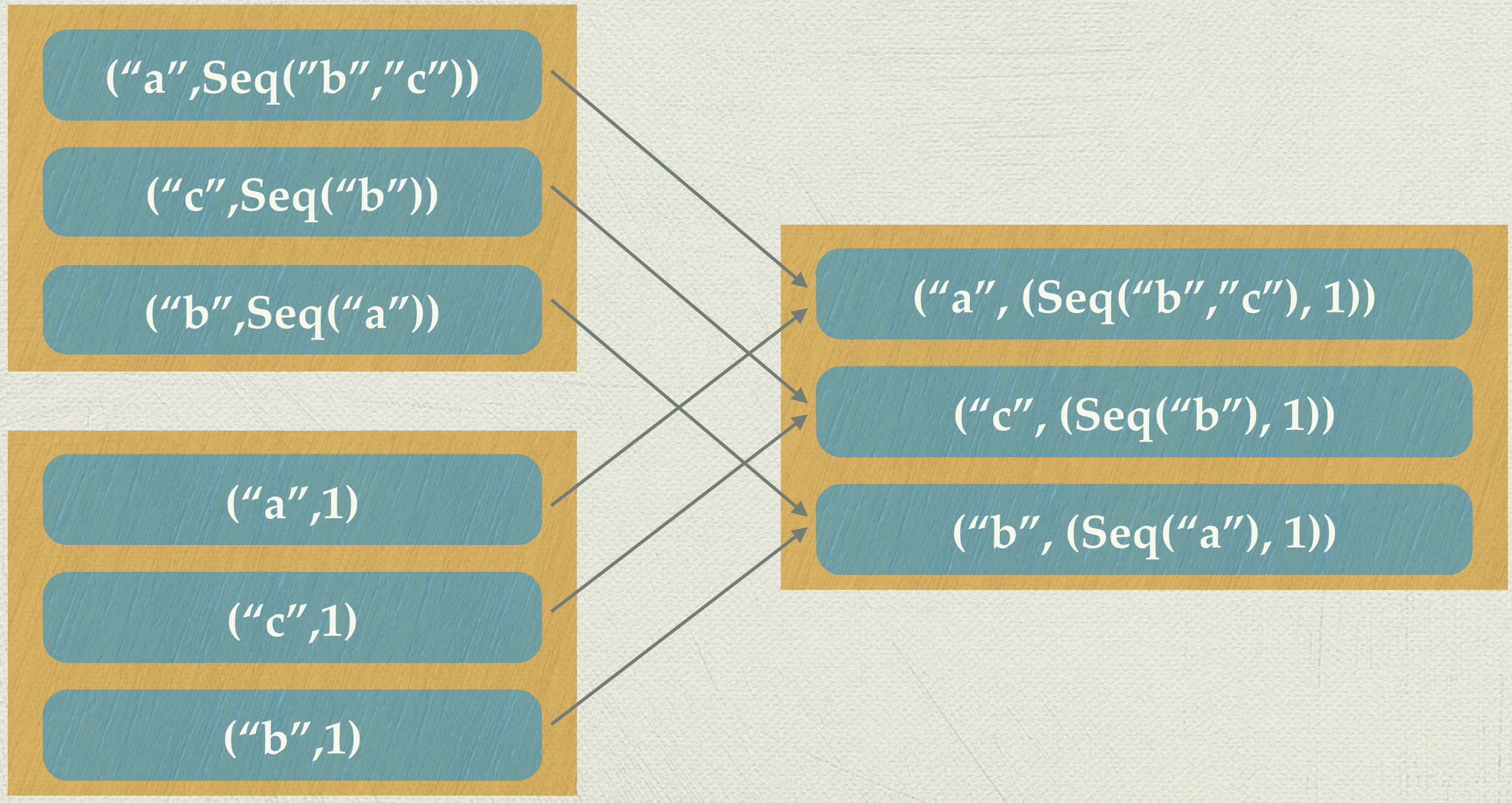
Example - PageRank

```
val n = lines.count()
var ranks = links.mapValues( v => 1.0)
val iden = links.mapValues( v => 0.0)
for( i <- 1 to iters ){
    val con = links.join(ranks).values.flatMap{ case(urls, rank) =>
        val size = urls.size
        urls.map(url => (url, rank/size))
    }
    ranks = con.union(iden).reduceByKey(_+_)
                .mapValues(0.15/n + 0.85*_)
}
val output = ranks.sortByKey().collect();
```

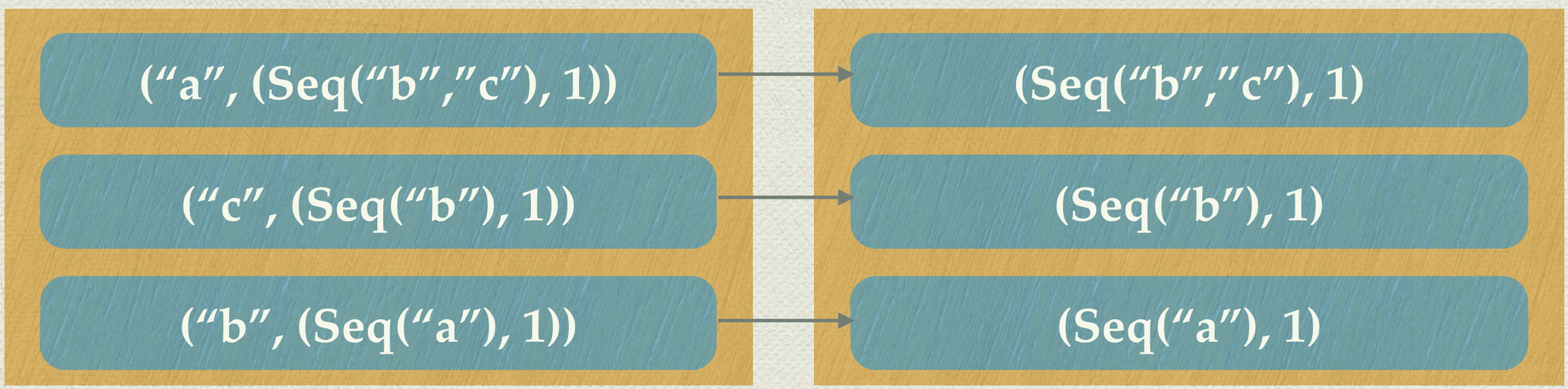
Example - PageRank

```
val n = lines.count()
var ranks = links.mapValues( v => 1.0)
val iden = links.mapValues( v => 0.0)
for( i <- 1 to iters ){
    val con = links.join(ranks).values.flatMap{ case(urls, rank) =>
        val size = urls.size
        urls.map(url => (url, rank/size))
    }
    ranks = con.union(iden).reduceByKey(_+_)
                .mapValues(0.15/n + 0.85*_)
}
val output = ranks.sortByKey().collect();
```

Example - PageRank



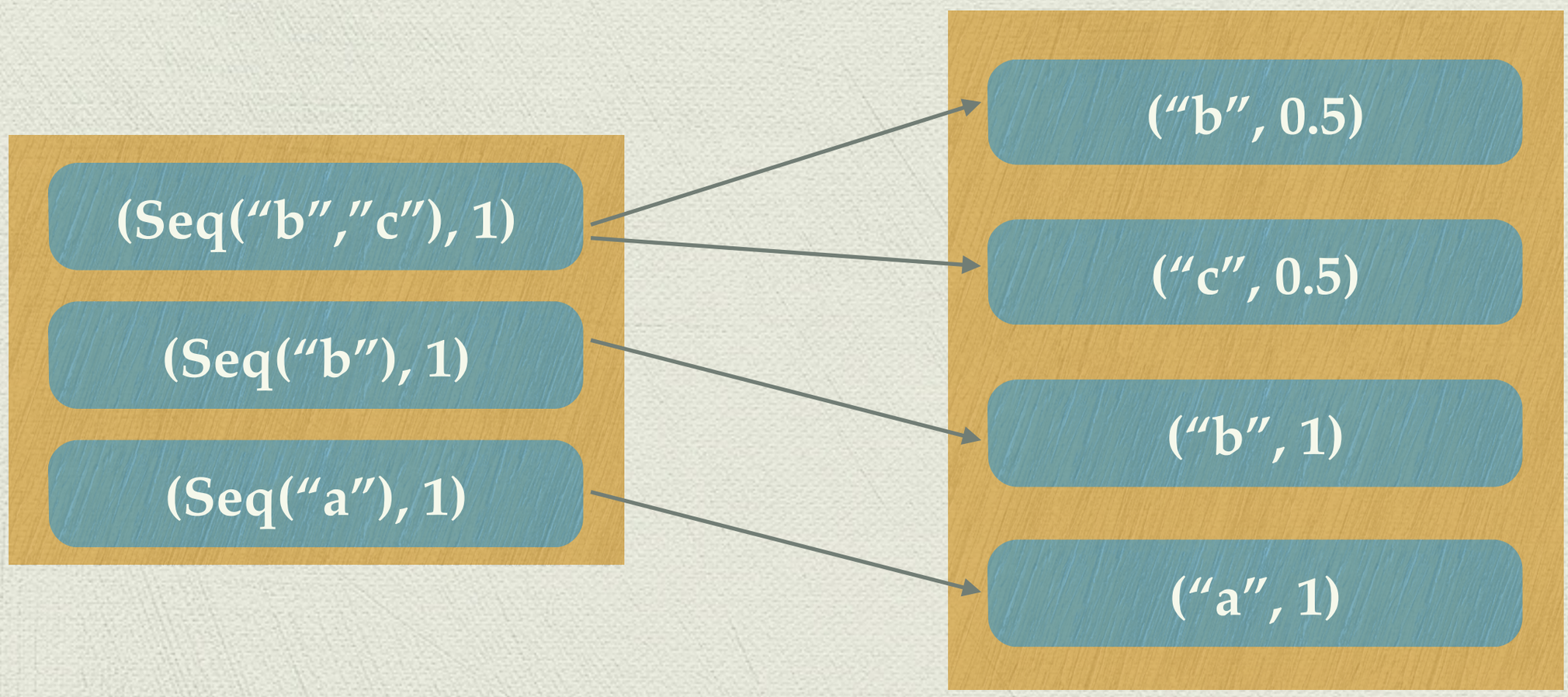
Example - PageRank



Example - PageRank

```
val n = lines.count()
var ranks = links.mapValues( v => 1.0)
val iden = links.mapValues( v => 0.0)
for( i <- 1 to iters ){
    val con = links.join(ranks).values.flatMap{ case(urls, rank) =>
        val size = urls.size
        urls.map(url => (url, rank/size))
    }
    ranks = con.union(iden).reduceByKey(_+_)
                .mapValues(0.15/n + 0.85*_)
}
val output = ranks.sortByKey().collect();
```

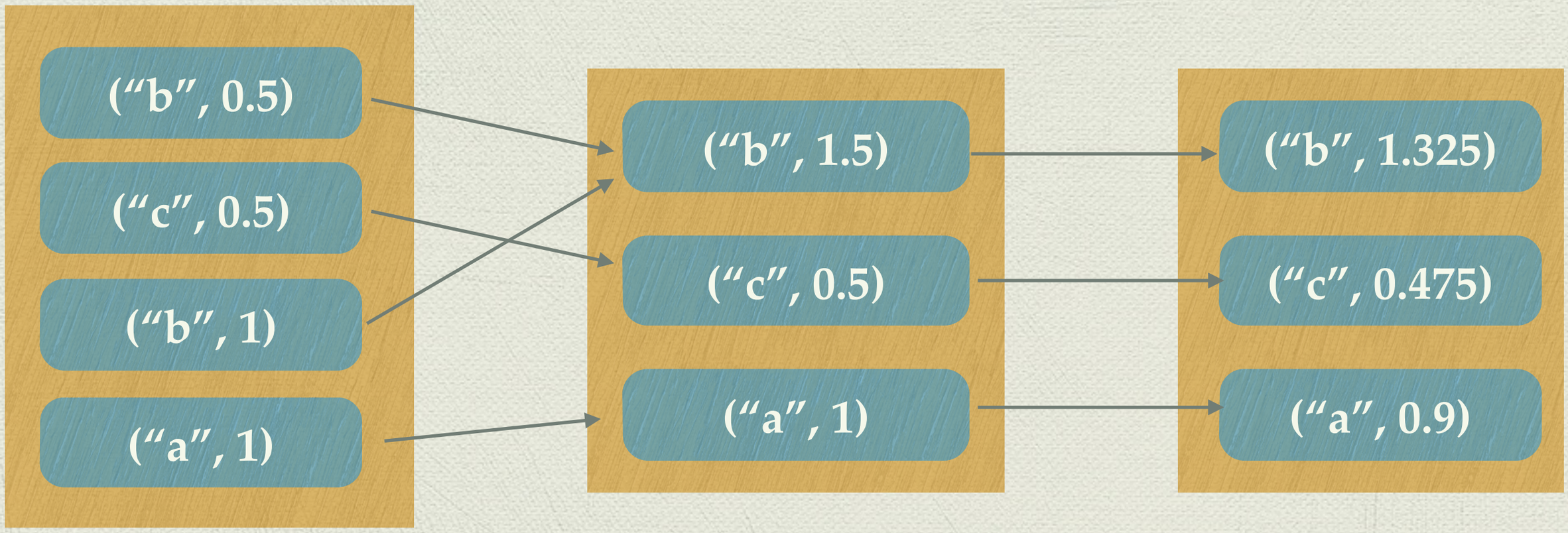
Example - PageRank



Example - PageRank

```
val n = lines.count()
var ranks = links.mapValues( v => 1.0)
val iden = links.mapValues( v => 0.0)
for( i <- 1 to iters ){
    val con = links.join(ranks).values.flatMap{ case(urls, rank) =>
        val size = urls.size
        urls.map(url => (url, rank/size))
    }
    ranks = con.union(iden).reduceByKey(_+_)
                .mapValues(0.15/n + 0.85*_)
}
val output = ranks.sortByKey().collect();
```

Example - PageRank



Example - PageRank

```
val n = lines.count()
var ranks = links.mapValues( v => 1.0)
val iden = links.mapValues( v => 0.0)
for( i <- 1 to iters ){
    val con = links.join(ranks).values.flatMap{ case(urls, rank) =>
        val size = urls.size
        urls.map(url => (url, rank/size))
    }
    ranks = con.union(iden).reduceByKey(_+_)
                .mapValues(0.15/n + 0.85*_)
}
val output = ranks.sortByKey().collect();
```

Outline

- Introduction to Scala & functional programming
- Tour of Spark operations
- Example - PageRank
- Lab Assignment
- Reference

Assignment Topic

- Application name: Descendant query
- use **Standalone** deploy model
- read input file form HDFS
- Input File: friend list (name1,name2)
- Input argument: *$InputPath,* Name and H
- Output: return all friends within H hops for that person

Assignment Topic

- Sample Input :

  Tom,Bob
  Tom,Alice
  Elisa,Tom
  Eric,Elisa
  Elisa,Harry

- Input argument: friend.txt Eric 2
- Output: should contain “Eric”, “Elisa”, “Tom”, “Harry”

Instructions and Resources

- cd *$PATH OF YOUR PROJECT*
- compile : sbt package
- run :
  spark-submit \
  - - class <main-class> \
  - - master <master-url>\
  <application-jar> \
  [application-arguments]

Master URL

- Spark Host: 140.114.91.199 Port: 7077
- Spark Web UI: http://140.114.91.199:8081
- HDFS Host: Quanata006 Port: 8100

Master URL	Meaning
local	Run Spark locally with one worker thread (i.e. no parallelism at all).
local[K]	Run Spark locally with K worker threads (ideally, set this to the number of cores on your machine).
spark://HOST:PORT	Connect to the given Spark standalone cluster master. The port must be whichever one your master is configured to use, which is 7077 by default.
mesos://HOST:PORT	Connect to the given Mesos cluster. The host parameter is the hostname of the Mesos master. The port must be whichever one the master is configured to use, which is 5050 by default.

Spark Web UI

Spark Master at spark://14

140.114.91.199:8081

常用 ACM Lab 學校相關 其他 Google Yahoo!奇摩

Spark 1.3.0 **Spark Master at spark://140.114.91.199:7077**

URL: spark://140.114.91.199:7077
REST URL: spark://140.114.91.199:6066 *(cluster mode)*
Workers: 8
Cores: 192 Total, 0 Used
Memory: 747.7 GB Total, 0.0 B Used
Applications: 0 Running, 37 Completed
Drivers: 0 Running, 1 Completed
Status: ALIVE

Workers

Worker Id	Address	State	Cores	Memory
worker-20150406075846-Quanta015-37185	Quanta015:37185	ALIVE	24 (0 Used)	93.5 GB (0.0 B Used)
worker-20150406075912-Quanta010-51011	Quanta010:51011	ALIVE	24 (0 Used)	93.5 GB (0.0 B Used)
worker-20150406075916-Quanta012-53281	Quanta012:53281	ALIVE	24 (0 Used)	93.5 GB (0.0 B Used)
worker-20150406080222-Quanta017-52131	Quanta017:52131	ALIVE	24 (0 Used)	93.5 GB (0.0 B Used)
worker-20150406080512-Quanta013-59661	Quanta013:59661	ALIVE	24 (0 Used)	93.5 GB (0.0 B Used)
worker-20150406080930-Quanta016-36038	Quanta016:36038	ALIVE	24 (0 Used)	93.5 GB (0.0 B Used)
worker-20150406080945-Quanta011-60454	Quanta011:60454	ALIVE	24 (0 Used)	93.5 GB (0.0 B Used)
worker-20150406160252-Quanta006-42605	Quanta006:42605	ALIVE	24 (0 Used)	93.4 GB (0.0 B Used)

Running Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration

Running Drivers

Submission ID	Submitted Time	Worker	State	Cores	Memory	Main Class

Completed Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration

HDFS Web UI

Spark Master at spark://14 | Hadoop NameNode Quant

140.114.91.199:52070/dfshealth.jsp

常用 ACM Lab 學校相關 其他 Google Yahoo!奇摩

NameNode 'Quanta006:8100'

Started:	Mon Mar 30 11:16:07 CST 2015
Version:	1.0.3, r1335192
Compiled:	Tue May 8 20:31:25 UTC 2012 by hortonfo
Upgrades:	There are no upgrades in progress.

Browse the filesystem
Namenode Logs

Cluster Summary

333 files and directories, 365 blocks = 698 total. Heap Size is 1.19 GB / 1.74 GB (68%)

WARNING : There are about 15 missing blocks. Please check the log or run fsck.

Configured Capacity	:	641.86 GB
DFS Used	:	23.53 GB
Non DFS Used	:	199.61 GB
DFS Remaining	:	418.72 GB
DFS Used%	:	3.67 %
DFS Remaining%	:	65.23 %
Live Nodes	:	7
Dead Nodes	:	0
Decommissioning Nodes	:	0
Number of Under-Replicated Blocks	:	43

NameNode Storage:

Storage Directory	Type	State
/mnt/ssd-1/jhadoop-namestore/hadoop-jhadoop	IMAGE_AND_EDITS	Active

This is Apache Hadoop release 1.0.3

Instructions and Resources

- $ find .
  .
  ./simple.sbt
  ./src
  ./src/main
  ./src/main/scala
  ./scr/main/scala/SimpleApp.scala
  ./project
  ...
  ./target
  ...

Instructions and Resources

- $ cat simple.sbt
  name := “Simple Project”

  version := “1.0”

  scalaVersion := “2.10.3”

  libraryDependencies += “org.apache.spark” %% “spark-core” % “1.3.0”

  libraryDependencies += “org.apache.hadoop” % “hadoop-client” % “1.0.3”

Outline

- Introduction to Scala & functional programming
- Tour of Spark operations
- Example - PageRank
- Lab Assignment
- Reference

Reference

- Spark doucmentation
- Parallel Programming with Spark
- Spark Java/Scala API