Introduction to BigData Computing Framework - Apache Hadoop & Spark

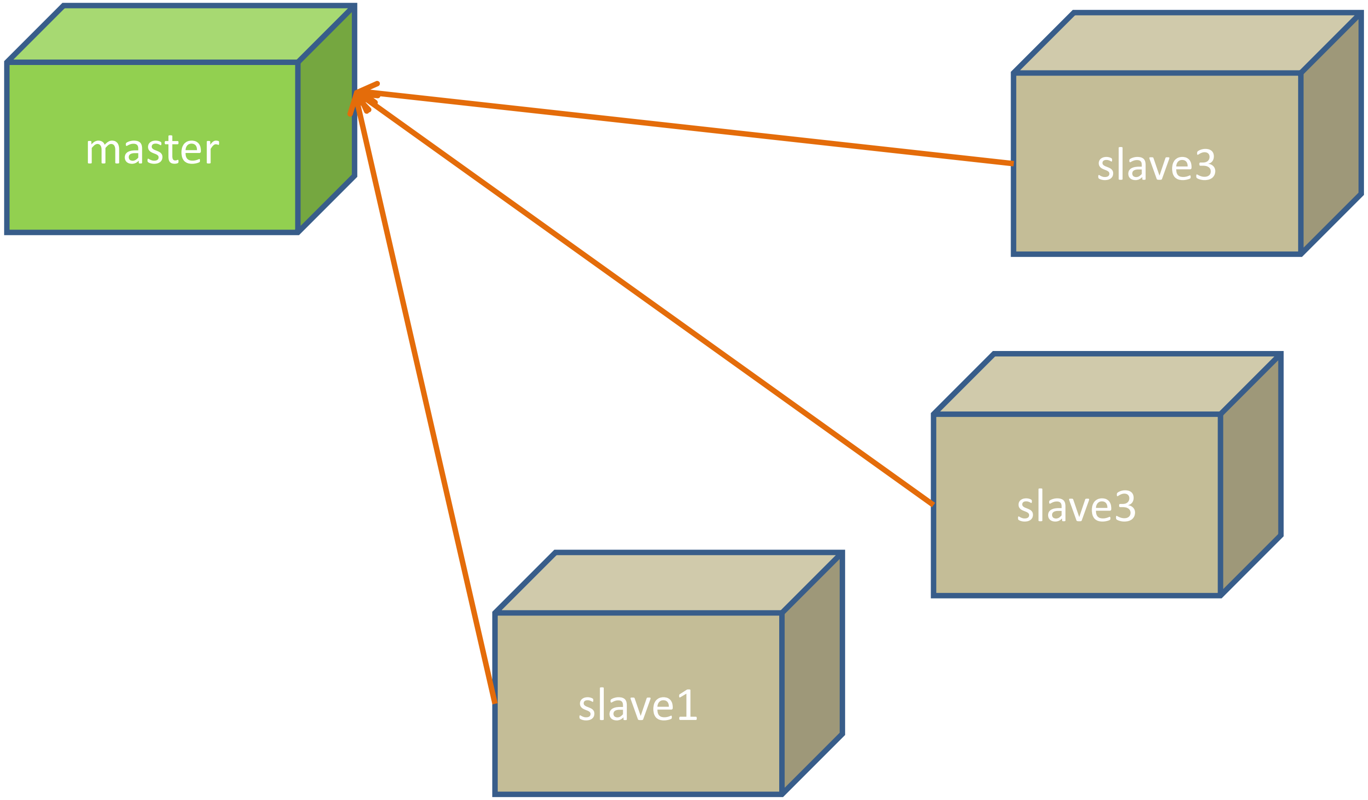
國網中心
莊家雋

What will you learn today ?

- Distributed Computing environment
- Apache Hadoop
  - Concept of MapReduce
  - MapReduce Programing
  - MapReduce Examples
- Apache Spark
  - Concept of RDD
  - RDD Programing
  - Spark Examples

Distributed System

- Master /slave architecture

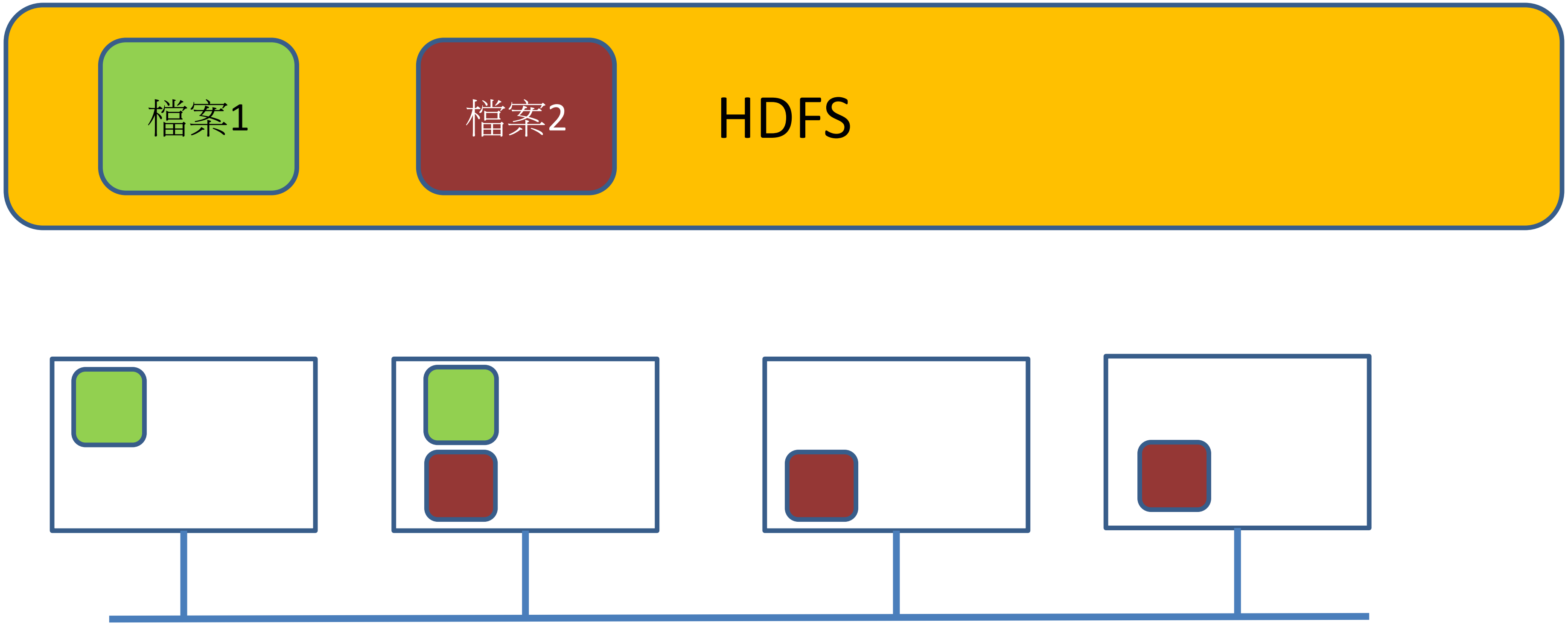


- 開發一個分散式系統很難
  - 主機間如何溝通
  - 系統的可靠性設計
  - …
- Hadoop
  - HDFS、MapReduce
- Spark
  - RDD



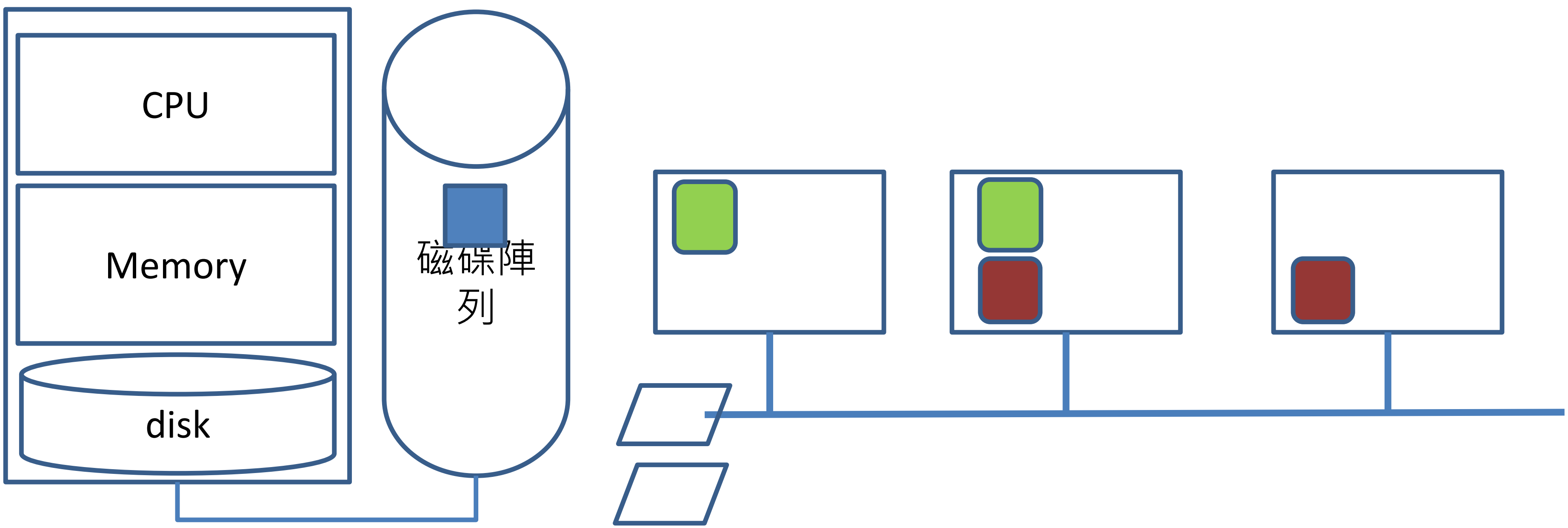
分散式檔案系統：HDFS

- 在分散式的儲存環境裏，提供邏輯上的單一目錄系統
- 每個檔案被分割成許多區塊並進行異地備份

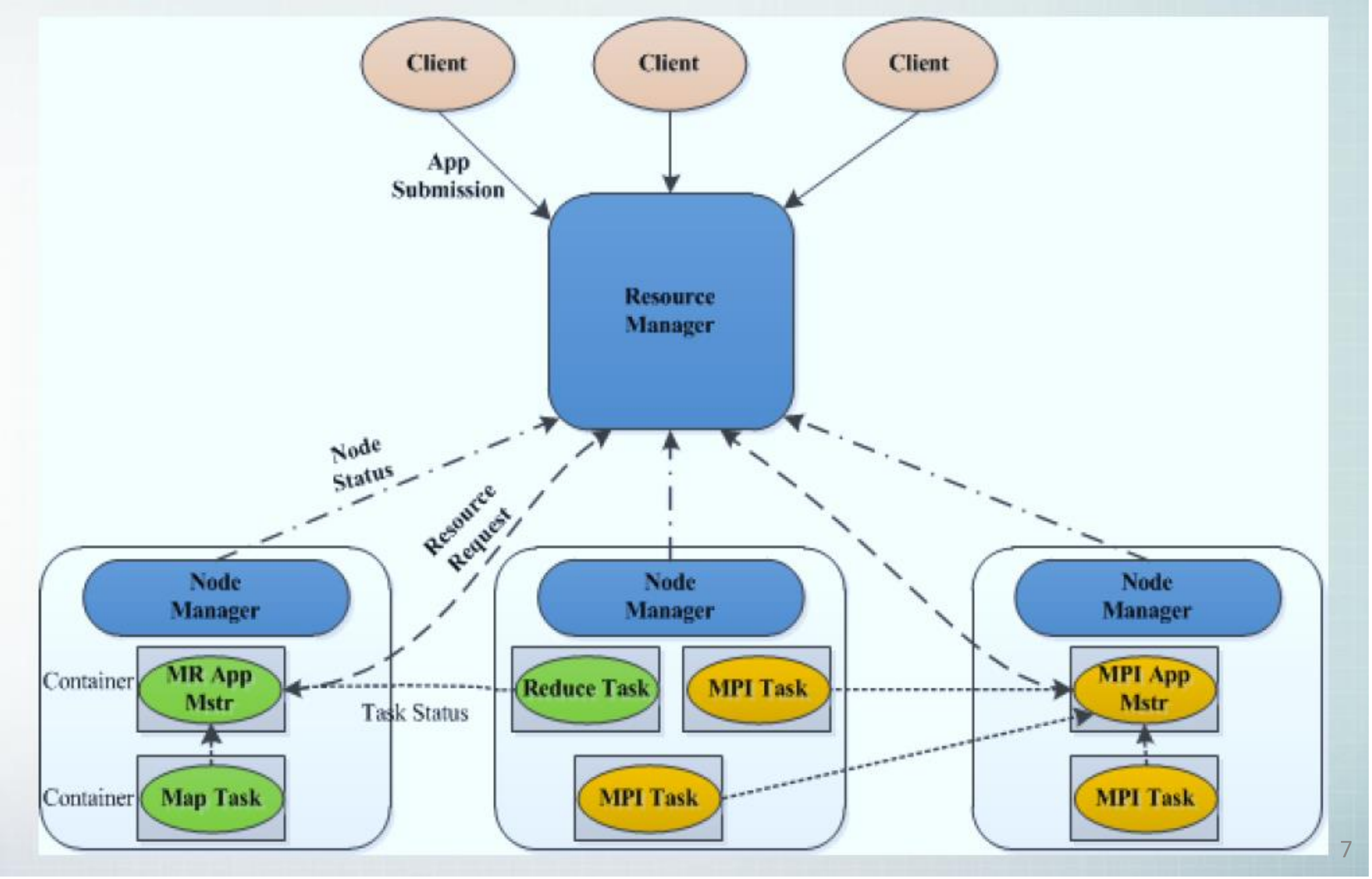


Data Locality

- 移動運算到資料端比移動資料到運算端來的成本低
  - 減少資料搬運，實現在地運算



Concept of Distributed computing



MapReduce Concept & Real life Example

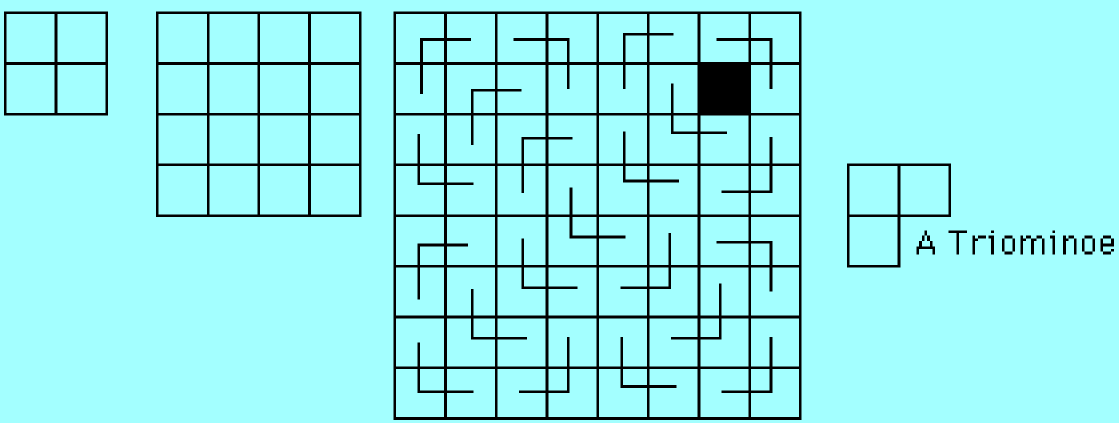
數學歸納法證明

步驟 1、證明$n=1$時，敘述成立。

步驟 2、假設$n=k$時，敘述成立；證明$n=k+1$，敘述也成立。

由數學歸納法得證， n 為任意自然數時都成立

3. Show that any 2^n x 2^n board with one square deleted can be covered by Triominoes.



試證明：當自然數 $n \geq 3$ 時，不等式：$5^n > 3^n + 4^n$ 恆成立.

證明：(1) $n=3$ 時，左式 $=5^3=125$，右式 $=3^3+4^3=91$，因 $125>91$，

故 $5^3 > 3^3 + 4^3$ 成立.

(2) 設 $n=k$ (k 是一個整數且 $k \geq 3$)時，$5^k > 3^k + 4^k$ 成立.

上式兩端同乘 5 得：

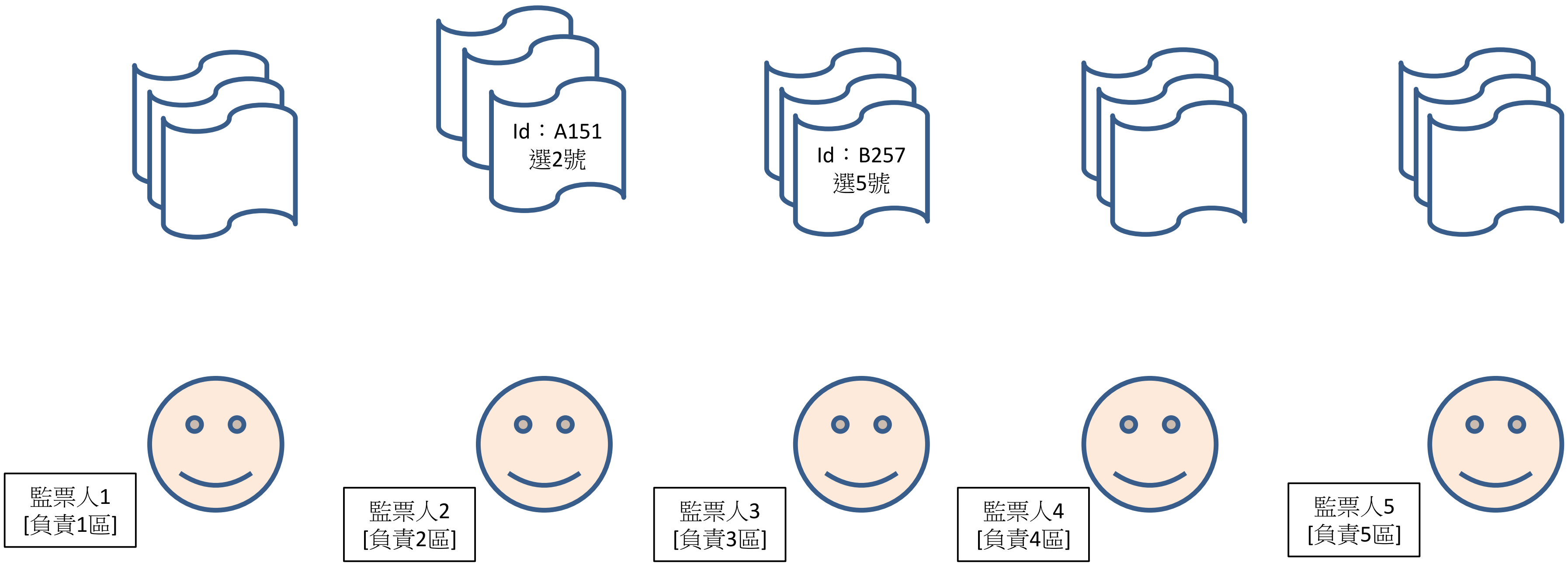
$$5^{k+1} > 5 \cdot 3^k + 5 \cdot 4^k > 3 \cdot 3^k + 4 \cdot 4^k = 3^{k+1} + 4^{k+1},$$

故 $5^{k+1} > 3^{k+1} + 4^{k+1}$ 亦成立.

由數學歸納法知 $5^n > 3^n + 4^n$ ($n \geq 3$)成立.

選舉到了…

- 台北市10個選區，共100萬票，要算出每個候選人的得票數



號次	票數
2	1
1	1
…	…

號次	票數
1	1
1	1
3	1
…	…

號次	票數
3	1
2	1
1	1

號次	票數
1	1
3	1
3	…

號次	票數
3	1
2	1
3	1

號次	票數
2	1
1	1
...	...

號次	票數
5	1
1	1
7	1
...	...

號次	票數
5	1
2	1
1	1

號次	票數
1	1
5	1
3	...

號次	票數
4	1
2	1
6	1

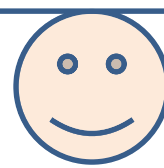
Shuffle & Sort
由各投開票所送到中選會

號次	票數
1	1
1	1
1	1
1	1
1	...

號次	票數
2	1
2	1
2	1
2	1
2	...

號次	票數
3	1
3	1
3	1
3	1
3	...

中選會
[負責 全部的候選人]



號次	總票數
1	187532

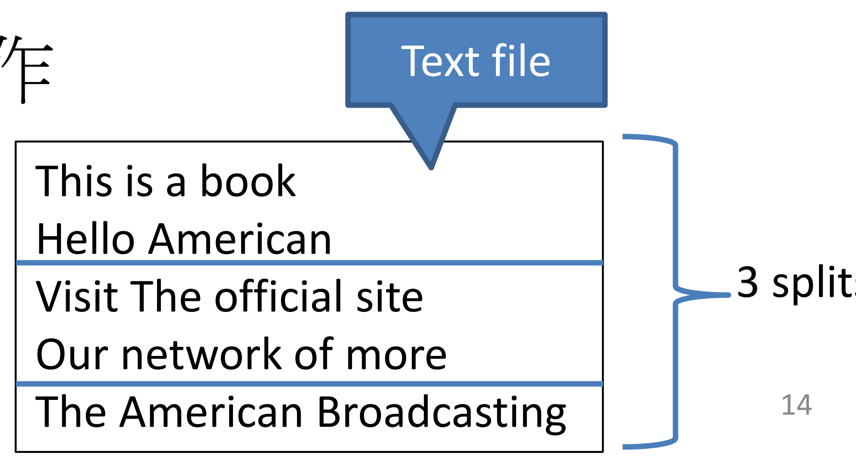
號次	總票數
2	574821

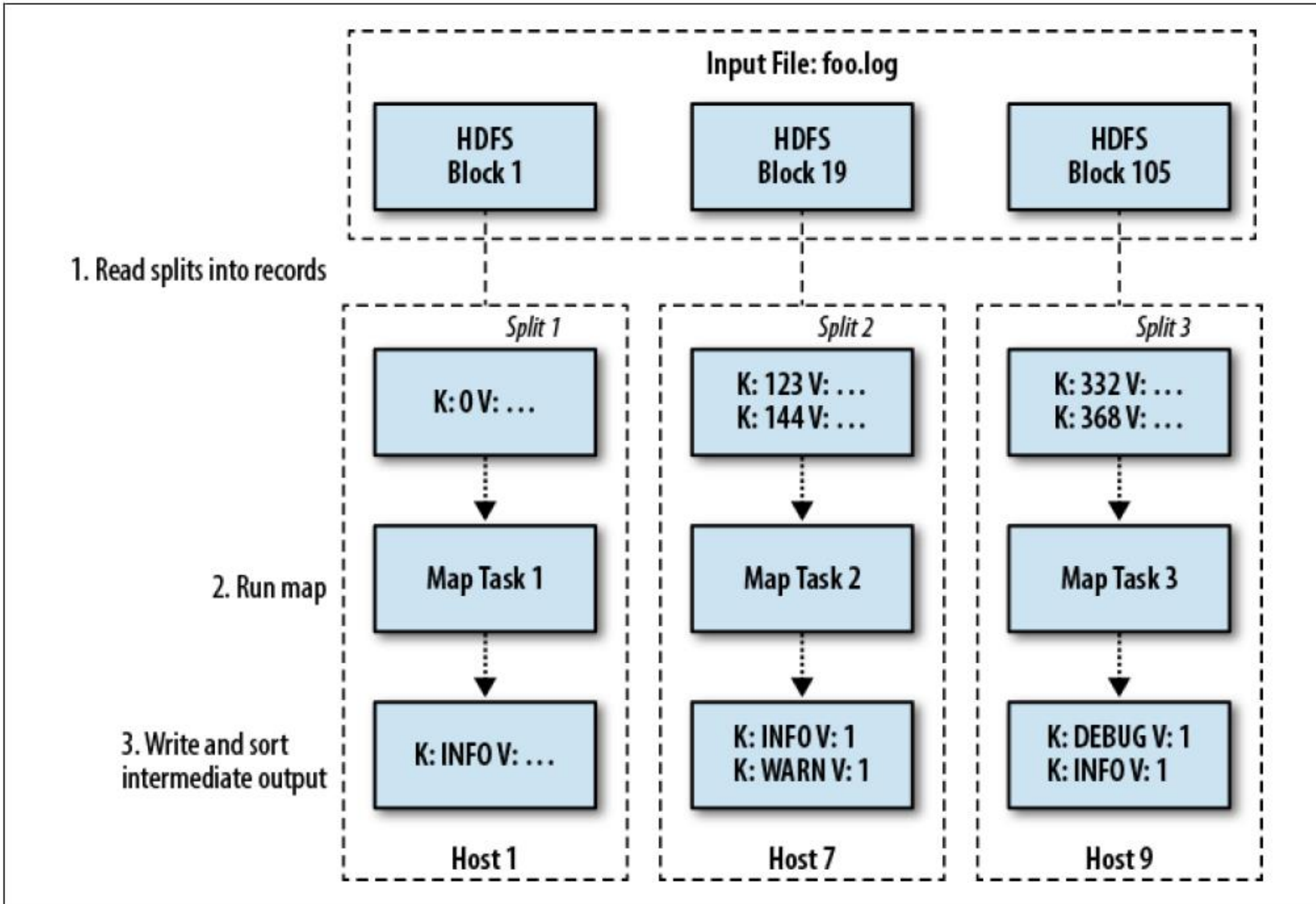
號次	總票數
3	237647

MapReduce Example
- Word Count

Word Count - Mapper

- 將輸入的文字檔案切成split
  - 每個mapper負責一個split
  - 由InputFomrat決定有多少個split
- Mapper處理split中的每一筆record
  - 由RecordReader定義一筆key/value record
- 將每一筆record內的字輸出 (字, 1)
  - 真正map()所執行的工作



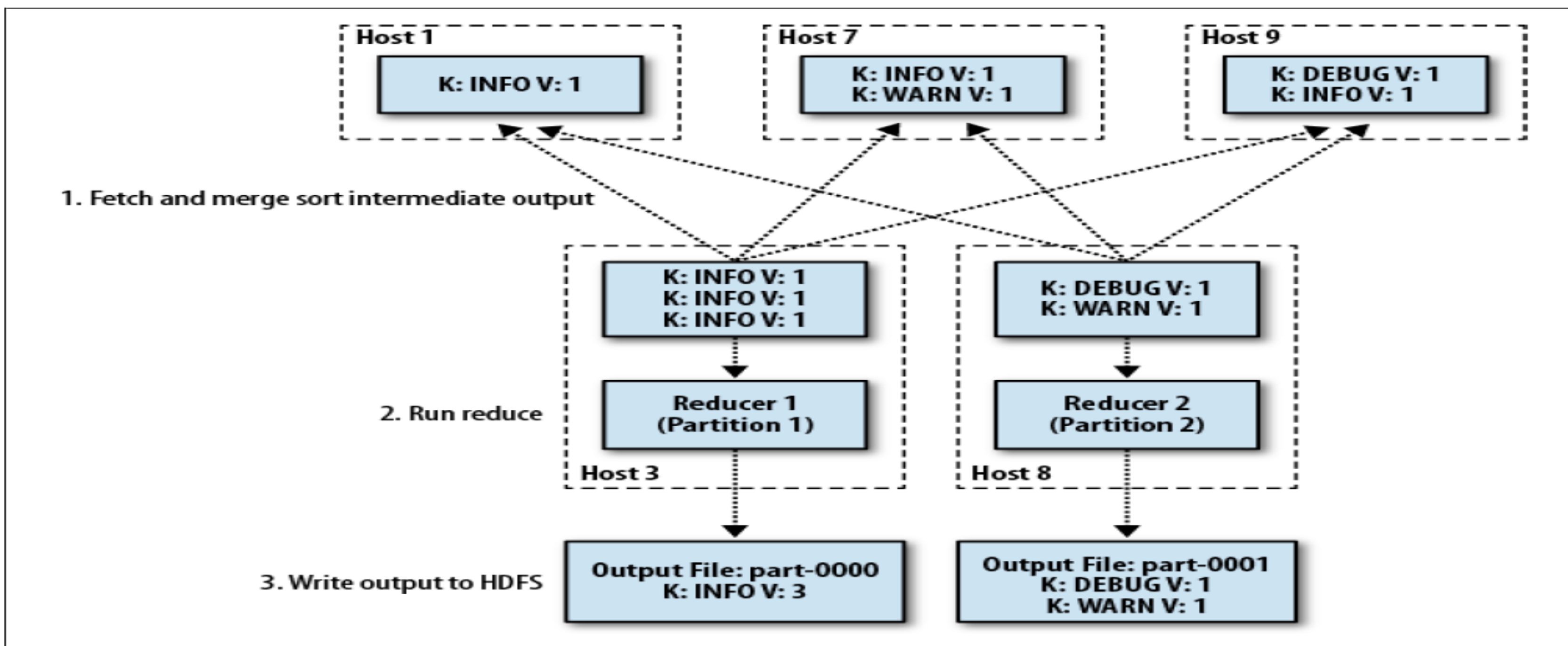

Input File: foo.log
HDFS Block 1
HDFS Block 19
HDFS Block 105
1. Read splits into records
Split 1
Split 2
Split 3
K: 0 V: ...
K: 123 V: ...
K: 144 V: ...
K: 332 V: ...
K: 368 V: ...
2. Run map
Map Task 1
Map Task 2
Map Task 3
3. Write and sort intermediate output
K: INFO V: ...
K: INFO V: 1
K: WARN V: 1
K: DEBUG V: 1
K: INFO V: 1
Host 1
Host 7
Host 9

Word Count – Shuffle & Sort

- Black box
  - 開發人員不用煩腦，framework會自行處理
- 在給Reducer之前完成
- 保證Reducer得到的資訊有下列三個特性
  - 可有多個Reducer
  - 同一個Reducer有可能處理多個Key值
  - 若Reducer看到某個Key1，會看到相對應的所有value
    - 給定Key1，所有Key1的值都會被同一個Reducer處理
    - Reducer 1收到 This這個字，會收到很多 1

Word Count - Reducer

- Reducer收到許多 key與相對應的value list
  - Reducer1 收到 (INFO, [1,1,1])
  - Reducer 2 收到 (DEBUG, [1]), (WARN, [1])
  - Reducer 對每個字的出現次數做加總



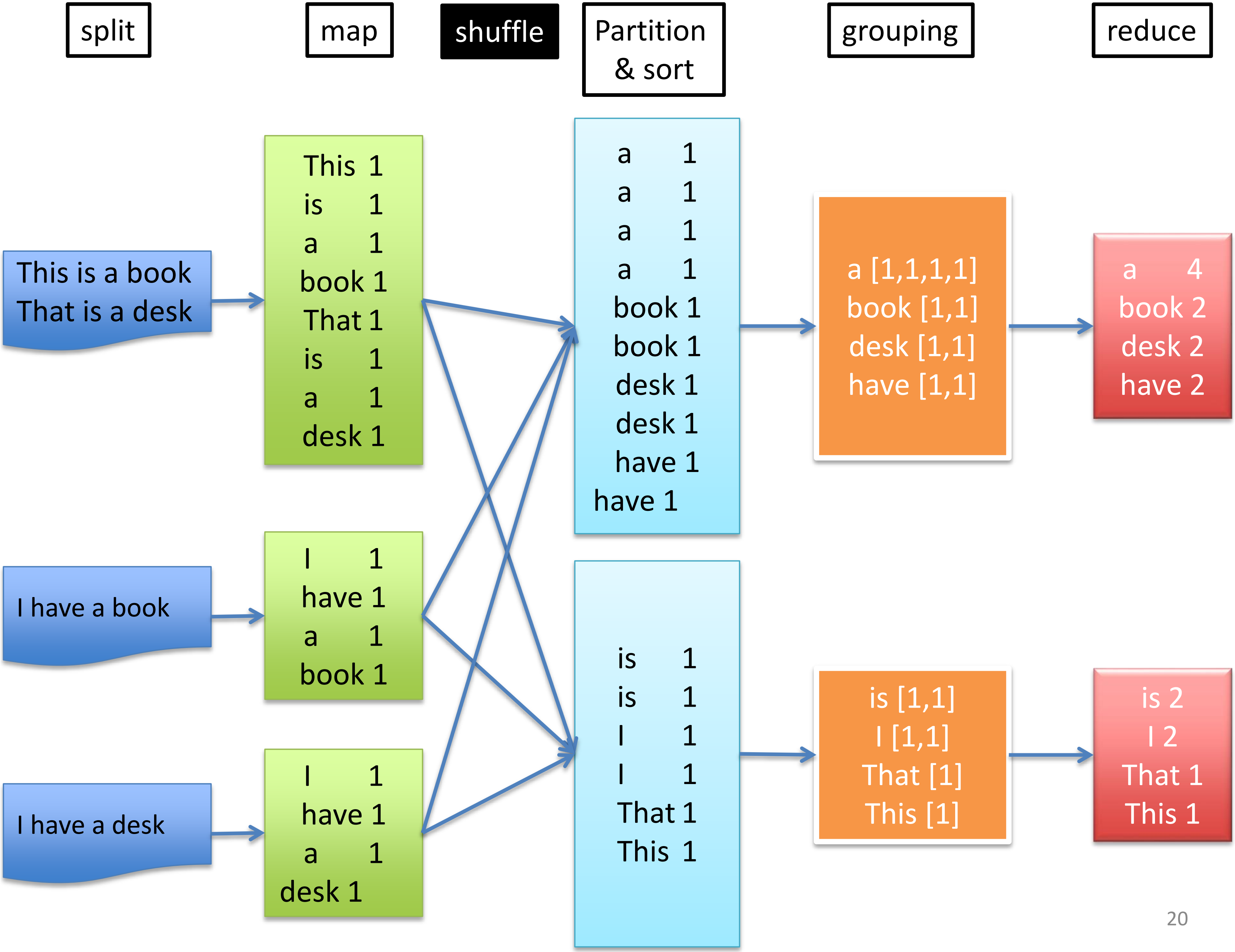
正規描述

- Mapper
  - (k1, v1) → list(k2, v2)
  - (0 , “This is a book book”) →
    (“This”, 1), (“is”,1), (“a”,1), (“book”,1),(“book”,1)
- Reducer
  - (k2, list(v2)) → (k3,v3)
  - (“This”, [1]) → (“This”, 1)
  - (“is”,[1]) → (“is”,1)
  - (“a”,[1]) → (“a”,1)
  - (“book”,[1, 1]) → (“book”,2)

Word Count – Pseudo code

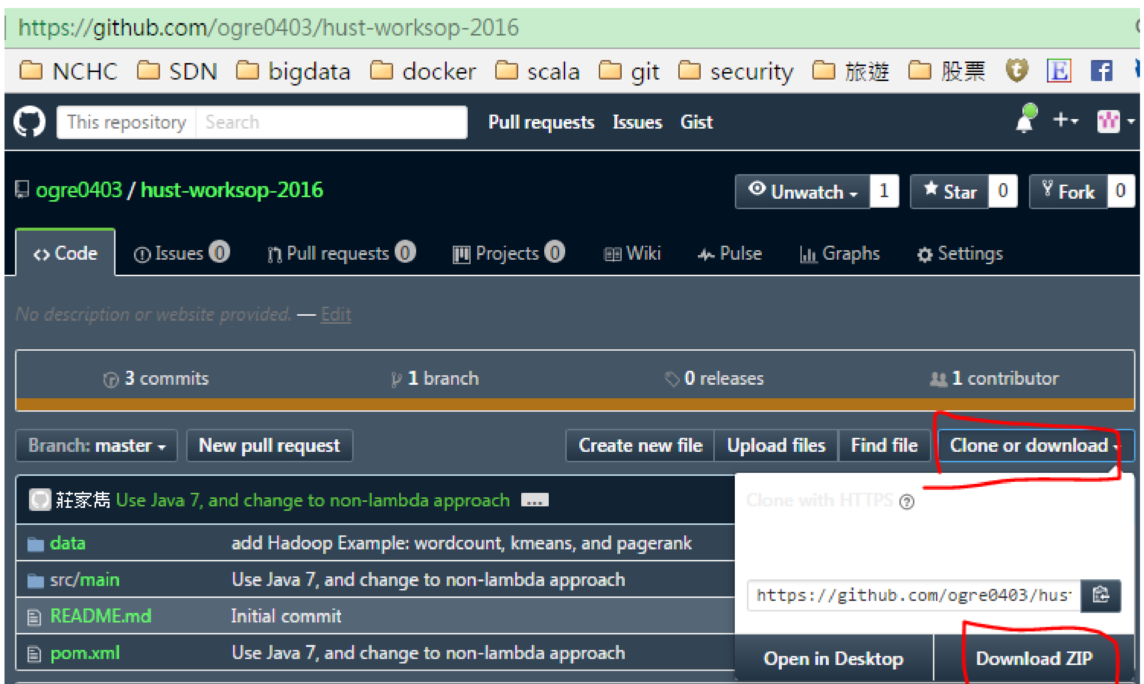
```
void Map (key, value){
        for each word x in value:
                output.collect(x, 1);
}
```

```
void Reduce (keyword, <list of value>){
        for each x in <list of value>:
                sum+=x;
        final_output.collect(keyword, sum);
}
```


split
map
shuffle
Partition & sort
grouping
reduce
This is a book
That is a desk
I have a book
I have a desk
This 1
is 1
a 1
book 1
That 1
is 1
a 1
desk 1
I 1
have 1
a 1
book 1
I 1
have 1
a 1
desk 1
a 1
a 1
a 1
a 1
book 1
book 1
desk 1
desk 1
have 1
have 1
is 1
is 1
I 1
I 1
That 1
This 1
a [1,1,1,1]
book [1,1]
desk [1,1]
have [1,1]
is [1,1]
I [1,1]
That [1]
This [1]
a 4
book 2
desk 2
have 2
is 2
I 2
That 1
This 1

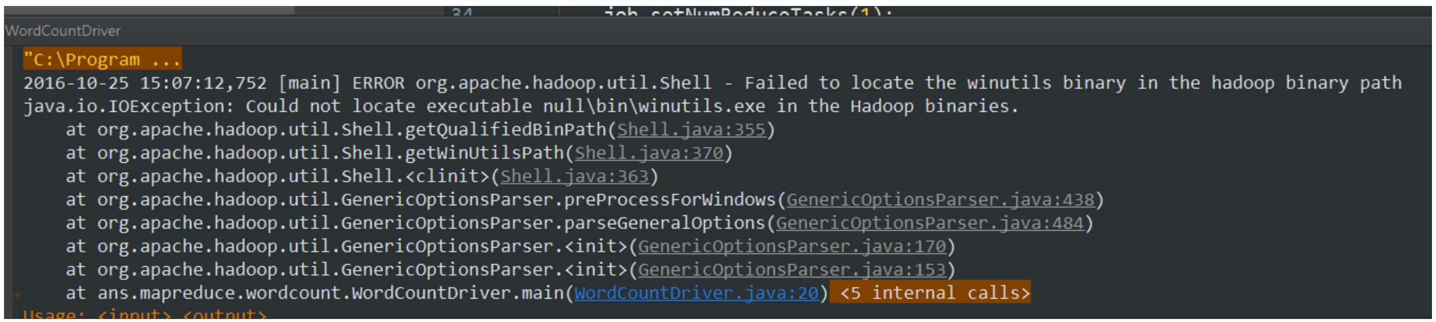
Labs 0: IntelliJ IDEA Setup

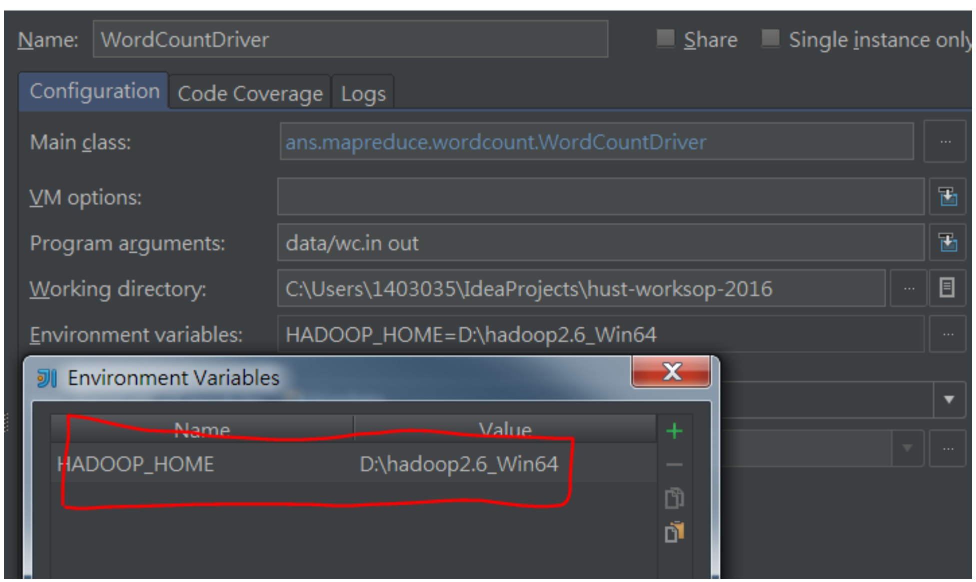
- Install IntelliJ IDEA Community Version
- Download labs code
  - https://github.com/ogre0403/hust-worksop-2016
- Import labs project into Intellij IDEA

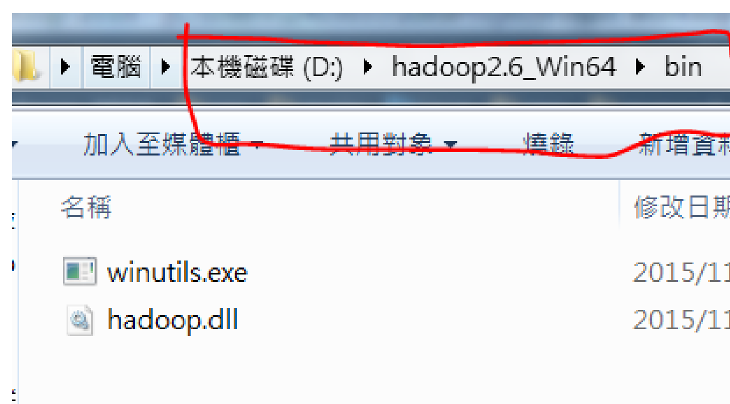


Labs 0: IntelliJ IDEA Setup

- Windows 64
  - Setup winutil.exe/hadoop.dll







Labs 1: Word Count(MapReduce)

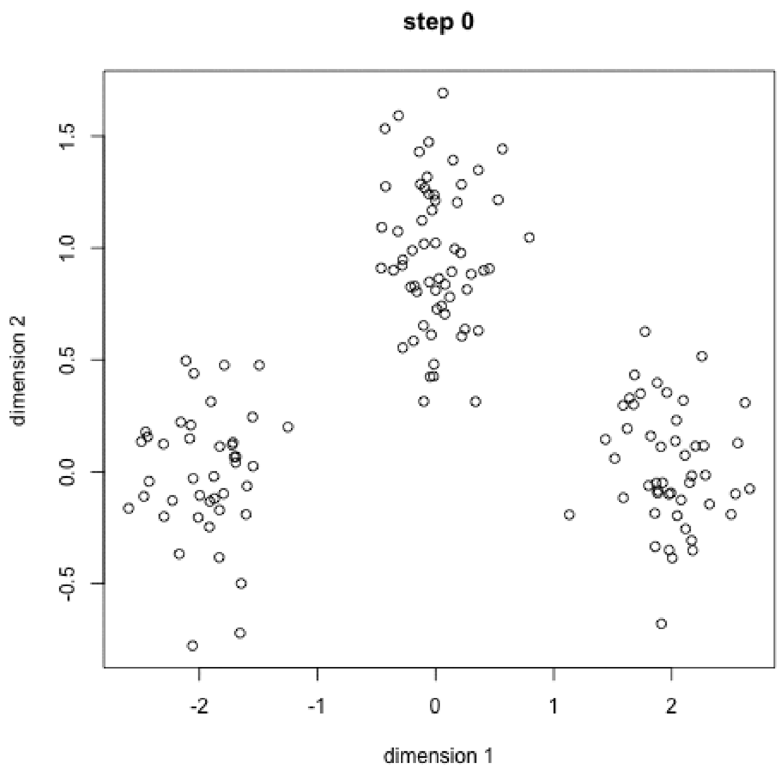
- WordCountMapper()
  - Use StringTokenizer.nextToken() to get word, then set the value by Text.set()

- WordCountReduceer
  - Iterate all elements in values, and sum up all IntWritable objects

MapReduce Example
- K-means

K-means clustering

- 隨機選取資料組中的k筆資料當作初始群中心u_1~u_k
- 計算每個資料xi 對應到最短距離的群中心 (固定 ui 求解所屬群 Si)
- 利用目前得到的分類重新計算群中心 (固定 Si 求解群中心 ui)
- 重複step 2,3直到收斂 (達到最大疊代次數 or 群心中移動距離很小)



集中式版本程式

```
// Add in new data, one at a time, recalculating centroids with each new one.
while(!finish) {
    //Clear cluster state
    clearClusters();

    List lastCentroids = getCentroids();

    //Assign points to the closer cluster
    assignCluster();

    //Calculate new centroids.
    calculateCentroids();

    iteration++;

    List currentCentroids = getCentroids();

    //Calculates total distance between new and old Centroids
    double distance = 0;
    for(int i = 0; i &lt; lastCentroids.size(); i++) {
        distance += Point.distance(lastCentroids.get(i),currentCentroids.get(i));
    }
    System.out.println("#################");
    System.out.println("Iteration: " + iteration);
    System.out.println("Centroid distances: " + distance);
    plotClusters();

    if(distance == 0) {
        finish = true;
    }
```

Map

輸入為<目前的中心，point>
求point到每個中心的距離
輸出為<所屬的中心，point>

Read Distributed cache
C1 : (x1,y1)
C2 : (x2,y2)
C3 : (x3,y3)

Key	value
C0	V1(1,2)
C0	V2(7,4)
C0	V3(16,3)
C0	V4(-1,-23)

mapper

Key	value
C2	V1(1,2)

Key	value
C2	V2(7,4)

Key	value
C1	V3(16,3)

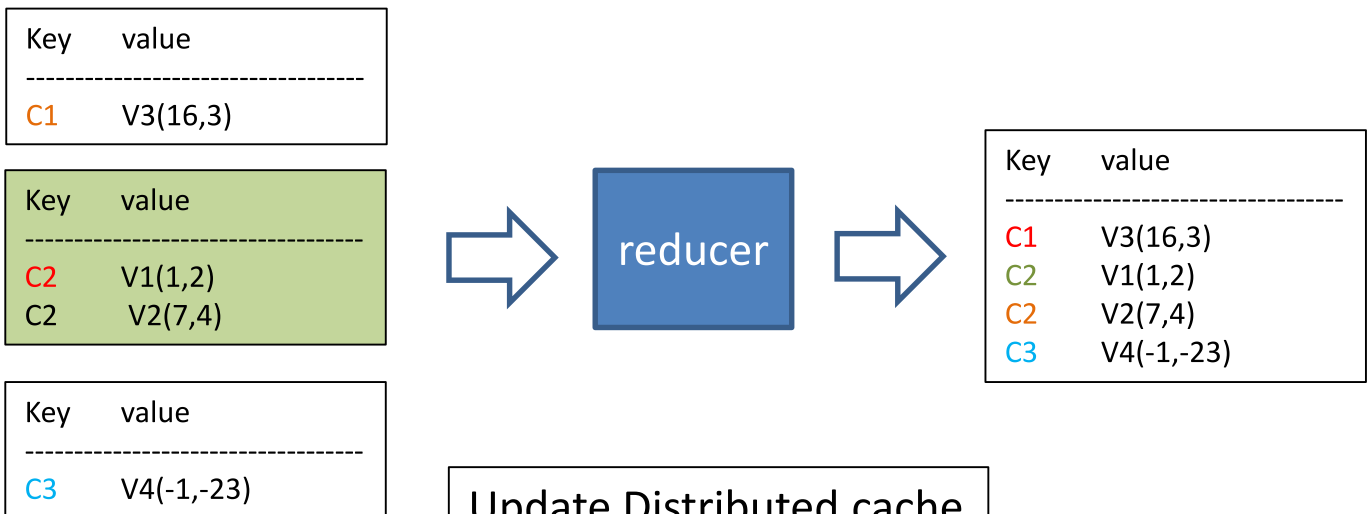
Key	value
C3	V4(-1,-23)

Reducer

輸入為<中心，屬於該中心的所有point>

對所有的point計算出新的中心

輸出<新的中心，point>做為下一次疊代



Update Distributed cache

C1 : (x'1,y'1)

C2 : (x'2,y'2)

C3 : (x'3,y'3)

Labs 2: K-means (MapReduce)

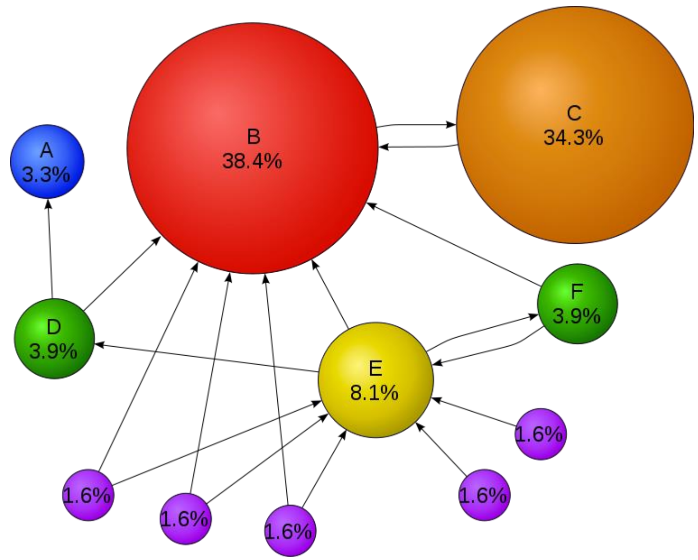
- KMeansMapper
  - Use DistanceMeasurer.measureDistance() to calculate distance between vector and ClusterCenter
  - Find the nearest ClusterCenter and the shortest distance
- KMeansReducer
  - Sum up all Vector value. (Each digital is stored in source[])

    Save result in resultVector[].
  - Calculate mean of each digital in resultVector[]

MapReduce Example
- Page Rank

PageRank

- 評估網頁重要程度的指標



$$PR(A) = \frac{PR(B)}{L(B)} + \frac{PR(C)}{L(C)} + \frac{PR(D)}{L(D)}$$

$$= \sum_{p_j} \frac{\mathrm{PageRank}(p_j)}{L(p_j)}$$

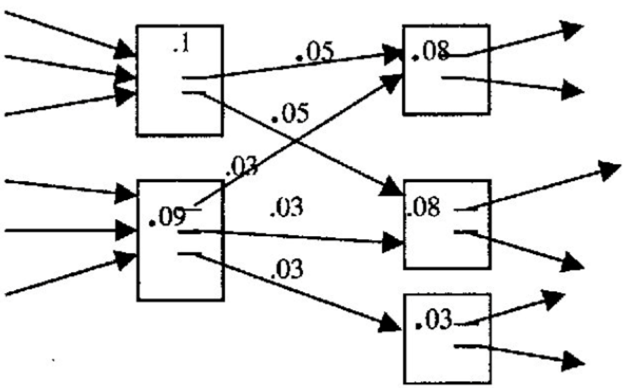
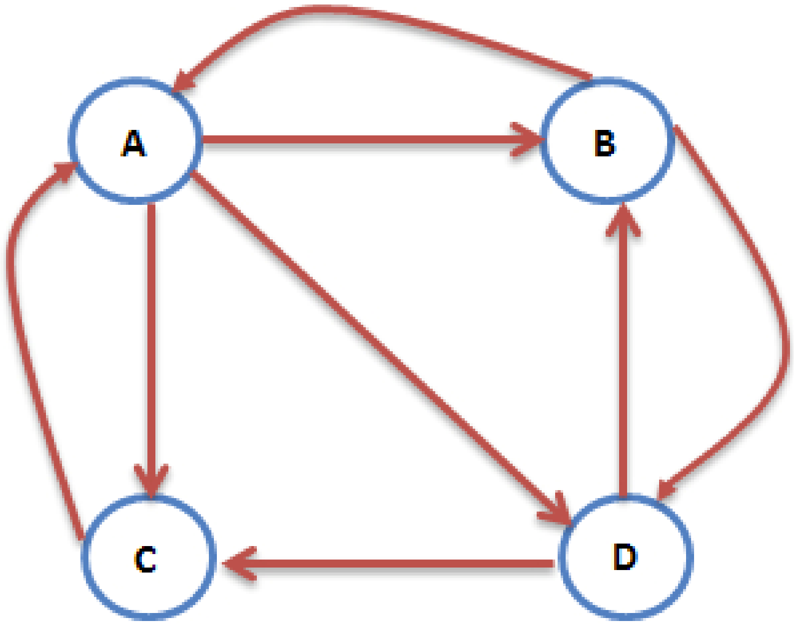


图 1 链接结构中的部分网页及其 PageRank 值



Page link matrix
= adjacency matrix
M[i][j] = 1 表示由 i 到 j 有一個邊

	A	B	C	D
A	0	1	1	1
B	1	0	0	1
C	1	0	0	0
D	0	1	1	0

Page link probability matrix
M[i][j] = p 表示由 i 到 j 的機率為 p

	A	B	C	D
A	0	1/3	1/3	1/3
B	1/2	0	0	1/2
C	1	0	0	
D	0	1/2	1/2	0

Transport Page link probability matrix
M[i][j] = p 表示由 j 到 i 的機率為 p
= 前一頁中的1 / Lj

	A	B	C	D
A	0	1/2	1	0
B	1/3	0	0	1/2
C	1/3	0	0	1/2
D	1/3	1/2	0	0

PR(A) = P(B→A) * PR(B)
+ P(C→A) * PR(C)
+ P(D→A) * PR(D)

PR(B) = P(A→B) * PR(A)
+ P(C→B) * PR(C)
+ P(D→B) * PR(D)

PR(C) = P(A→C) * PR(A)
+ P(B→C) * PR(B)
+ P(D→C) * PR(D)

PR(D) = P(A→D) * PR(A)
+ P(B→D) * PR(B)
+ P(C→D) * PR(C)

$$\sum_{p_j} \frac{\text{PageRank}(p_j)}{L(p_j)}$$

Iteration 1

P	A	B	C	D
A	0	1/2	1	0
B	1/3	0	0	1/2
C	1/3	0	0	1/2
D	1/3	1/2	0	0

X

	PR
A	1/4
B	1/4
C	1/4
D	1/4

=

	PR
A	9/24
B	5/24
C	5/24
D	5/24

Iteration 2

P	A	B	C	D
A	0	1/2	1	0
B	1/3	0	0	1/2
C	1/3	0	0	1/2
D	1/3	1/2	0	0

X

	PR
A	9/24
B	5/24
C	5/24
D	5/24

=

	PR
A	15/48
B	11/48
C	11/48
D	11/48

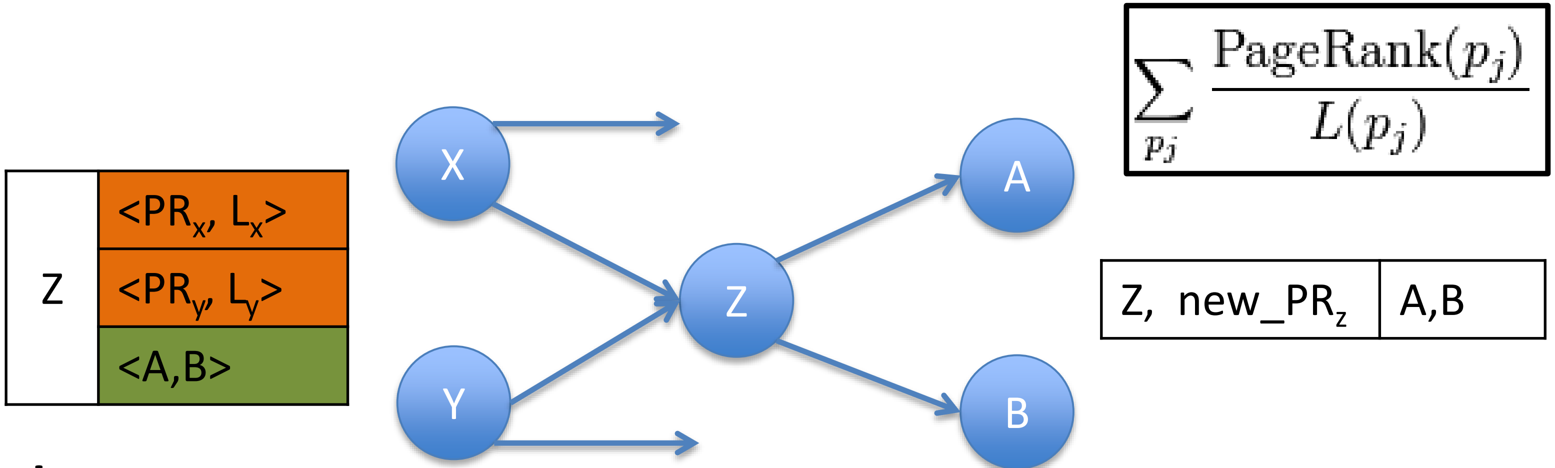
Iteration N

P	A	B	C	D
A	0	1/2	1	0
B	1/3	0	0	1/2
C	1/3	0	0	1/2
D	1/3	1/2	0	0

...

	PR
A	3/9
B	2/9
C	2/9
D	2/9

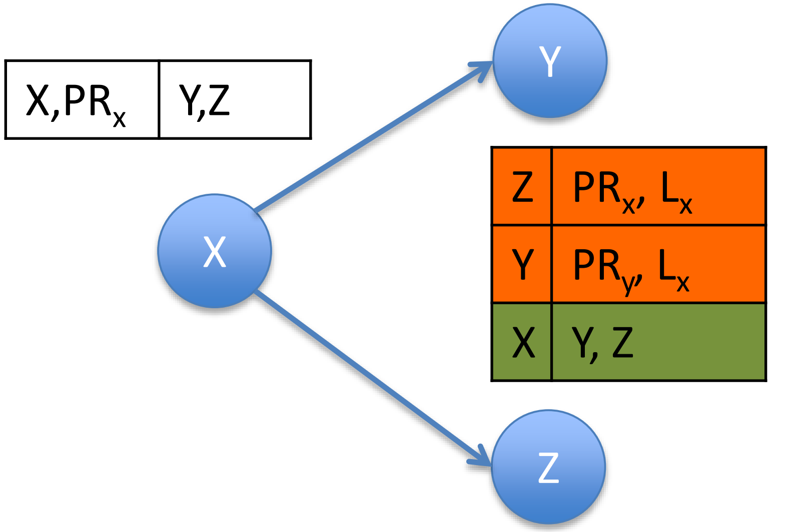
Reduce Pseudo Code



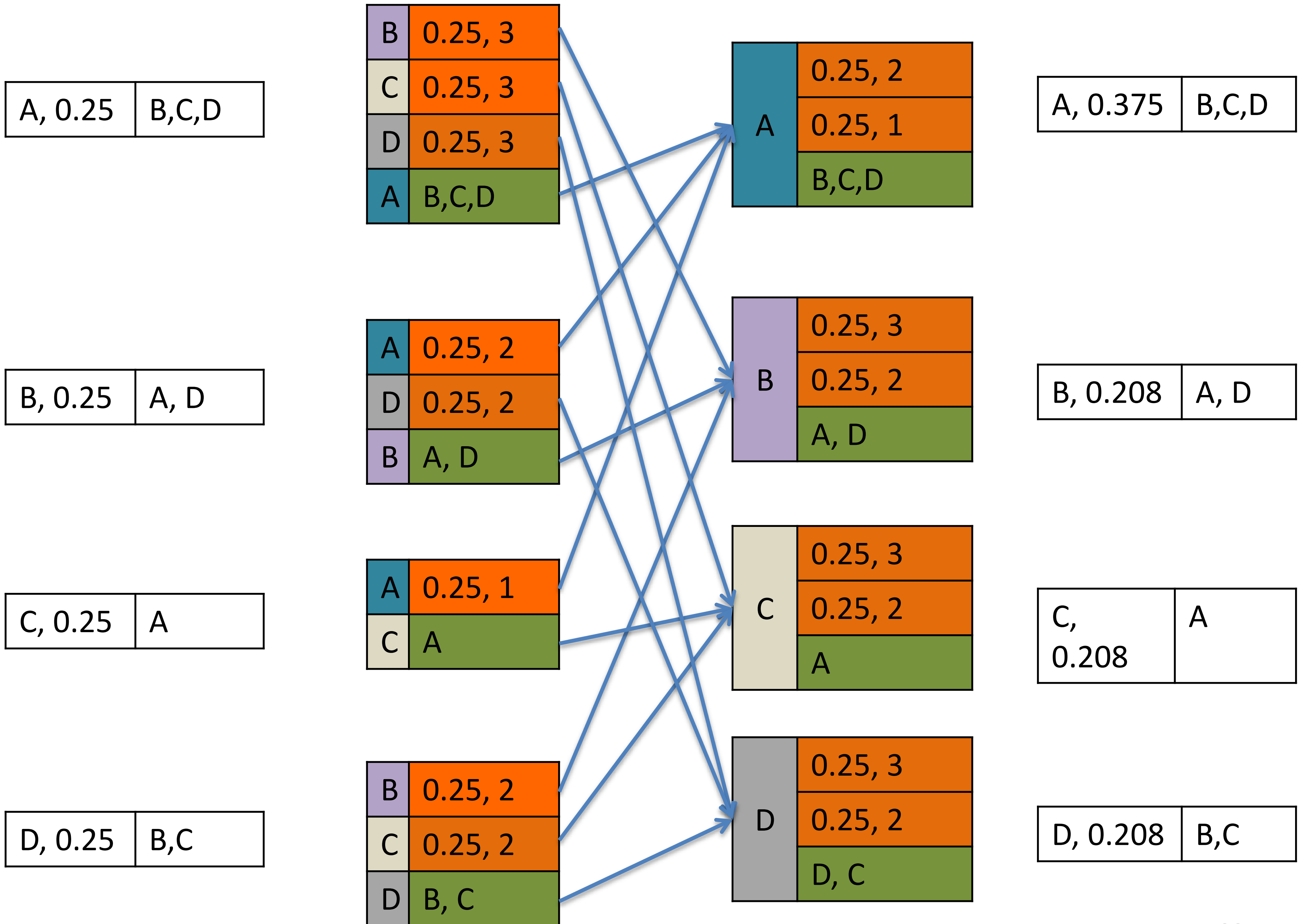
- Input:
  - Key: <p0>
  - Value: [<p1, p2, ..., pn>, <PR, L >, <PR, L> ...]
- Output
  - Key: <p0 new_PR>
  - Value: <p1, p2, ..., pn>

Map Pseudo Code

- Input:
  - Key: <p0, PR>
  - Value: <p1, p2, …, pn>



- Output:
  - Type 1
    - Key: <p1> (<p2>, <p3>, …)
    - Value: <PR L>
  - Type 2
    - Key: <p0>
    - Value: <p1, p2, …, pn>



A, 0.25
B,C,D
B, 0.25
A, D
C, 0.25
A
D, 0.25
B,C
B
0.25, 3
C
0.25, 3
D
0.25, 3
A
B,C,D
A
0.25, 2
D
0.25, 2
B
A, D
A
0.25, 1
C
A
B
0.25, 2
C
0.25, 2
D
B, C
A
0.25, 2
0.25, 1
B,C,D
B
0.25, 3
0.25, 2
A, D
C
0.25, 3
0.25, 2
A
D
0.25, 3
0.25, 2
D, C
A, 0.375
B,C,D
B, 0.208
A, D
C,
0.208
A
D, 0.208
B,C

Labs 3: Page Rank (MapReduce)

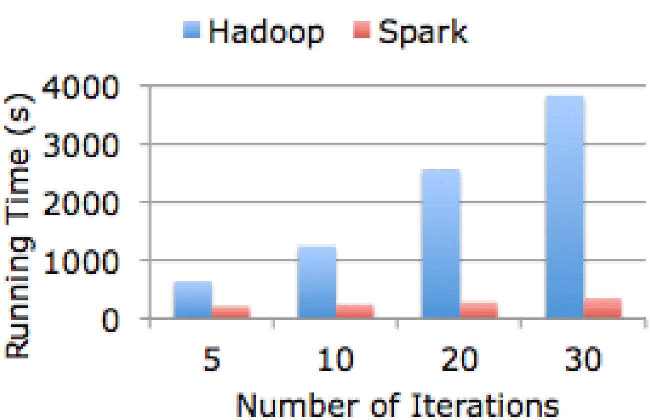
- RankCalculateMapper
  - For each linked to page, store (page, thisPagesRank + TotalLinksNumber)

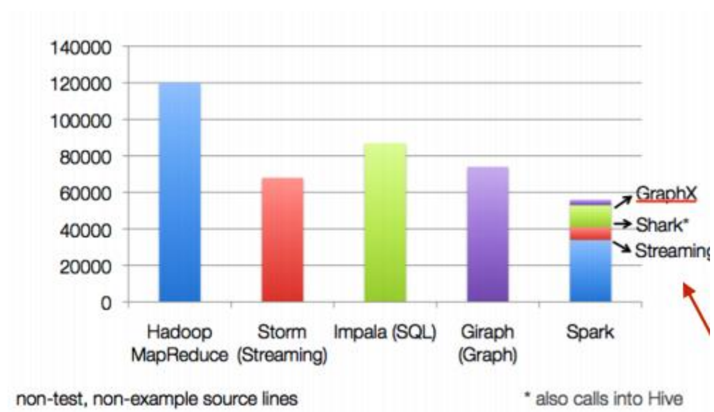
- RankCalculateReducer
  - Calculate fraction pagerank contributed from linked page.
  - Sum up all contributed pagerank.

Hadoop v.s. Spark

Why Spark

- Compare with Hadoop ecosystem
  - More efficient execution
  - More unified program abstraction
  - More flexible program operation



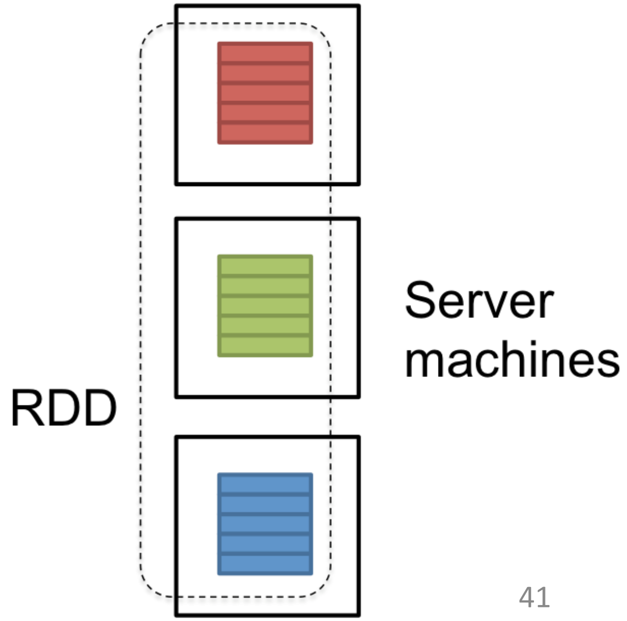


transformation
map(*func*)
filter(*func*)
flatMap(*func*)
sample(*withReplacement*, *fraction*, *seed*)
union(*otherDataset*)
distinct([*numTasks*]))
groupByKey([*numTasks*])
reduceByKey(*func*, [*numTasks*])
sortByKey([*ascending*], [*numTasks*])
join(*otherDataset*, [*numTasks*])
cogroup(*otherDataset*, [*numTasks*])
cartesian(*otherDataset*)

RDD Concept

RDD Essentials

- Resilient distributed dataset
- Each RDD is split into multiple **partitions**
  - Partitions may exist on different machines
- immutable distributed collection of objects
  - Transform creates new RDD
  - Coarse-grained transformation
- Spark keeps track lineage graph
  - Fast recovery from failure
- Lazy Evaluation
  - Until action is called



RDD operations

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()

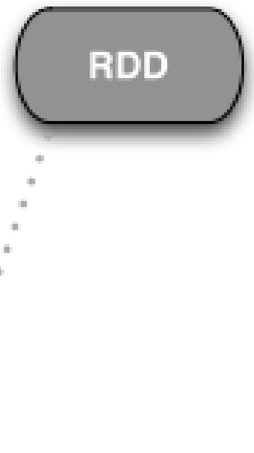
Actions

- Returns to the driver program a value or exports data to a storage system after performing a computation.
- Example:
  - Count()
  - Reduce(funct)
  - 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()

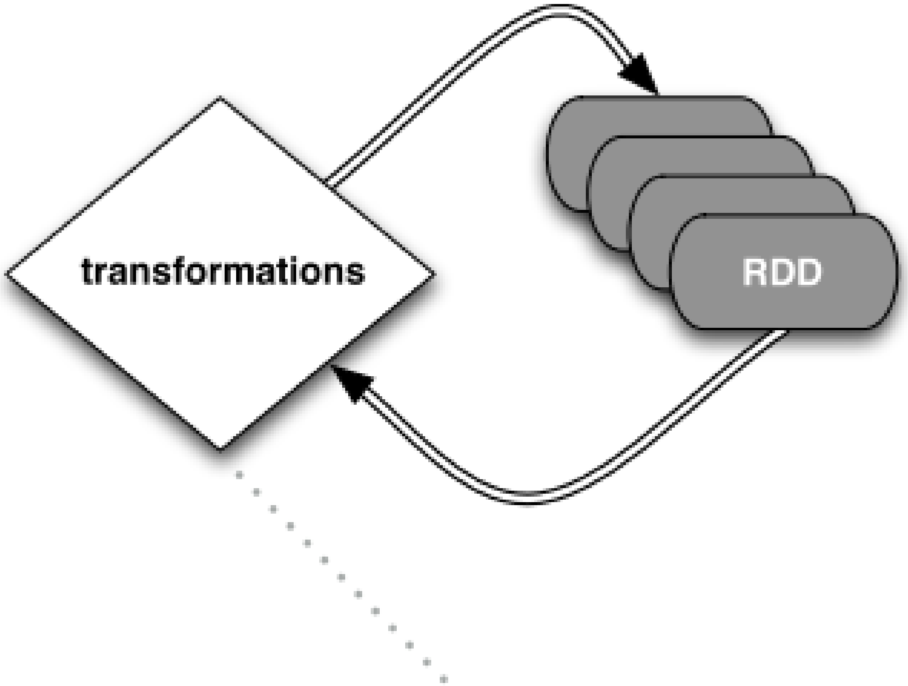
Create RDD




```
// base RDD
val lines = sc.textFile("hdfs://...")
```

```
JavaRDD<String> lines = sc.textFile("/path/to/README.md");
```

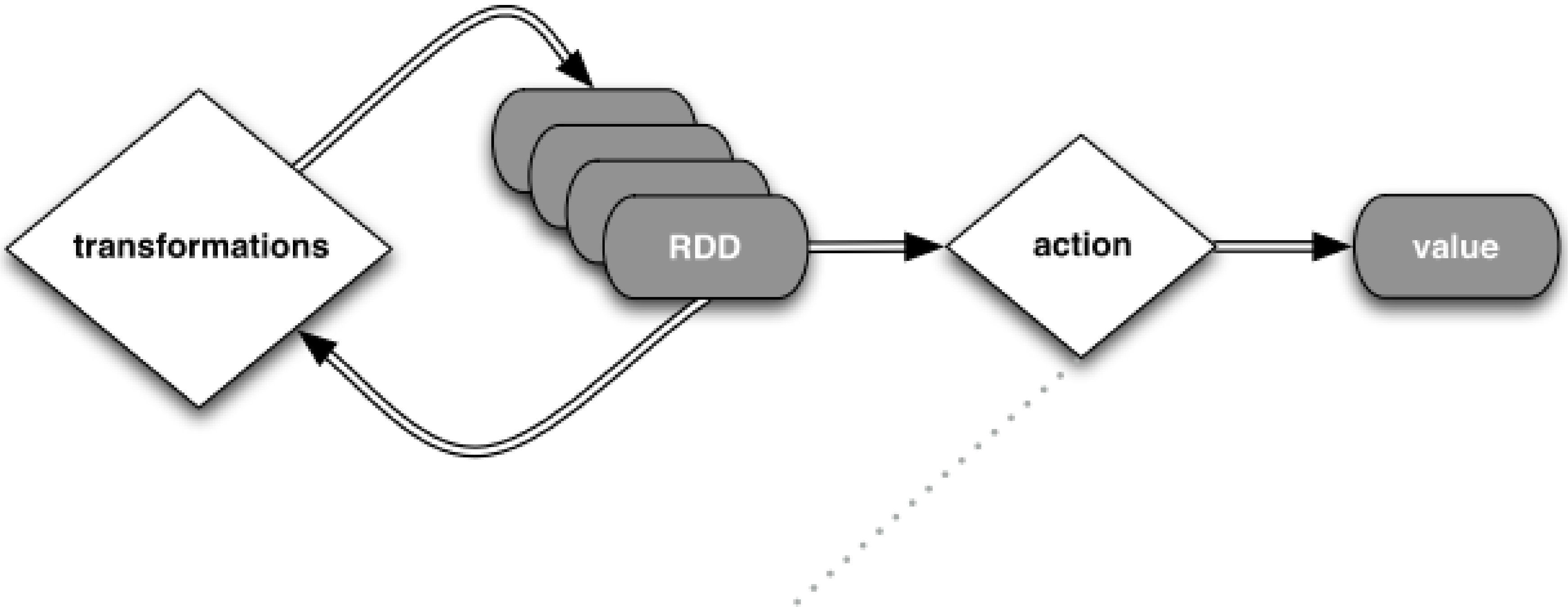
RDD Transformation




```
// transformed RDDs
val errors = lines.filter(_.startsWith("ERROR"))
val messages = errors.map(_.split("\t")).map(r => r(1))
messages.cache()
```

```
JavaRDD<String> errorsRDD = inputRDD.filter(
  new Function<String, Boolean>() {
    public Boolean call(String x) { return x.contains("error"); }
  }
});
```

RDD Action



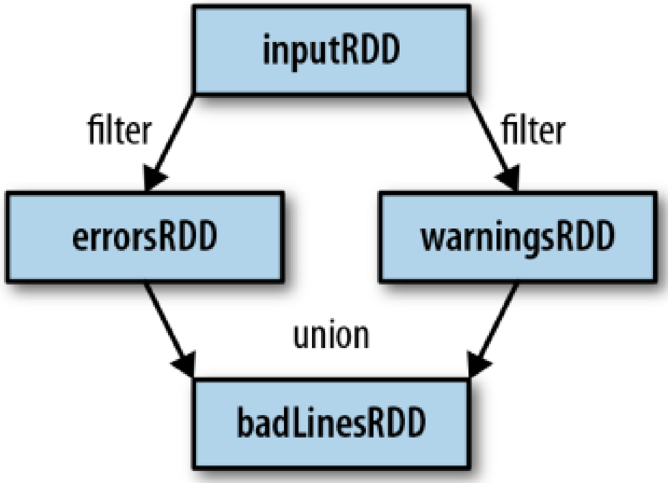

```
// action 1
messages.filter(_.contains("mysql")).count()
```

```
System.out.println("Input had " + badLinesRDD.count() + " concerning lines")
System.out.println("Here are 10 examples:")
for (String line: badLinesRDD.take(10)) {
  System.out.println(line);
}
```

Lineage Graph

- Think of each RDD as consisting of instructions on how to compute the data through transformations.

```
errorsRDD = inputRDD.filter(lambda x: "error" in x)
warningsRDD = inputRDD.filter(lambda x: "warning" in x)
badLinesRDD = errorsRDD.union(warningsRDD)
```



RDD Operations

RDD Type

- Basic RDD[T]
  - considers each data item as a single value
- Convert to other RDD Type
- PairRDD[K,V]
  - each data item containing key/value pairs.

http://blog.csdn.net/pelick/article/details/44922619
http://homepage.cs.latrobe.edu.au/zhe/ZhenHeSparkRDDAPIExamples.html

Basic RDD Transformation

- RDD.map(func)

Table 3-2. Basic RDD transformations on an RDD containing {1, 2, 3, 3}

Function name	Purpose	Example	Result
`map()`	Apply a function to each element in the RDD and return an RDD of the result.	`rdd.map(x => x + 1)`	`{2, 3, 4, 4}`

```
JavaRDD<Integer> rdd = sc.parallelize(Arrays.asList(1, 2, 3, 4));
JavaRDD<Integer> result = rdd.map(new Function<Integer, Integer>() {
  public Integer call(Integer x) { return x*x; }
});
System.out.println(StringUtils.join(result.collect(), ","));
```

Basic RDD Transformation

- RDD.filter(func)

Table 3-2. Basic RDD transformations on an RDD containing {1, 2, 3, 3}

Function name	Purpose	Example	Result
`filter()`	Return an RDD consisting of only elements that pass the condition passed to `filter()`.	`rdd.filter(x => x != 1)`	`{2, 3, 3}`

```
RDD<String> errors = lines.filter(new Function<String, Boolean>() {
  public Boolean call(String x) { return x.contains("error"); }
});
```

Basic RDD Transformation

- RDD.flatMap(func)

Table 3-2. Basic RDD transformations on an RDD containing {1, 2, 3, 3}

Function name	Purpose	Example	Result
`flatMap()`	Apply a function to each element in the RDD and return an RDD of the contents of the iterators returned. Often used to extract words.	`rdd.flatMap(x => x.to(3))`	`{1, 2, 3, 2, 3, 3, 3}`

```
JavaRDD<String> lines = sc.parallelize(Arrays.asList("hello world", "hi"));
JavaRDD<String> words = lines.flatMap(new FlatMapFunction<String, String>() {
  public Iterable<String> call(String line) {
    return Arrays.asList(line.split(" "));
  }
});
words.first();  // returns "hello"
```

Basic RDD Action

- RDD.reduce(func)

Table 3-4. Basic actions on an RDD containing {1, 2, 3, 3}

Function name	Purpose	Example	Result
`reduce(func)`	Combine the elements of the RDD together in parallel (e.g., `sum`).	`rdd.reduce((x, y) => x + y)`	9

```java
Integer sum = rdd.reduce(new Function2<Integer, Integer, Integer>() {
  public Integer call(Integer x, Integer y) { return x + y; }
});
```

- Java Basic RDD convert to
  - PairRDD : **explicitly** implement PairFunction()

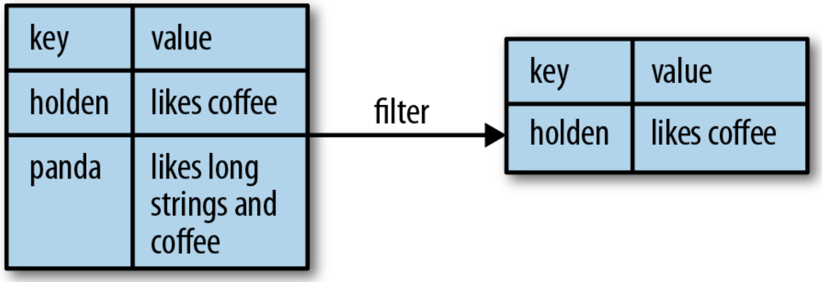
Function name	Equivalent function*<A, B,…>	Usage
PairFunction<T, K, V>	Function<T, Tuple2<K, V>>	PairRDD<K, V> from a mapToPair
DoubleFunction<T>	Function<T, double>	DoubleRDD from map ToDouble

```
PairFunction<String, String, String> keyData =
  new PairFunction<String, String, String>() {
  public Tuple2<String, String> call(String x) {
    return new Tuple2(x.split(" ")[0], x);
  }
};
JavaPairRDD<String, String> pairs = lines.mapToPair(keyData);
```

```
JavaDoubleRDD result = rdd.mapToDouble(
  new DoubleFunction<Integer>() {
    public double call(Integer x) {
      return (double) x * x;
    }
});
System.out.println(result.mean());
```

PairRDD Transformation

- PairRDD is also a RDD
  - RDD.filter(func)
  - RDD.map(func)
  - RDD.flatMap(func)
  - …




```
Function<Tuple2<String, String>, Boolean> longWordFilter =
  new Function<Tuple2<String, String>, Boolean>() {
    public Boolean call(Tuple2<String, String> keyValue) {
      return (keyValue._2().length() < 20);
    }
  };
JavaPairRDD<String, String> result = pairs.filter(longWordFilter);
```

PairRDD Transformation

- RDD.mapValues(func)

Table 4-1. Transformations on one pair RDD (example: {(1, 2), (3, 4), (3, 6)})

Function name	Purpose	Example	Result
`mapValues(func)`	Apply a function to each value of a pair RDD without changing the key.	`rdd.mapValues(x => x+1)`	`{(1, 3), (3, 5), (3, 7)}`

```java
JavaPairRDD<Integer,Integer> result =
        prdd.mapValues(new Function<Integer, Integer>() {
            @Override
            public Integer call(Integer v1) throws Exception {
            return v1 +1;
        }
});
```

PairRDD Transformation

- RDD.reduceByKey(func)

Table 4-1. Transformations on one pair RDD (example: {(1, 2), (3, 4), (3, 6)})

Function name	Purpose	Example	Result
`reduceByKey(func)`	Combine values with the same key.	`rdd.reduceByKey((x, y) => x + y)`	`{(1, 2), (3, 10)}`

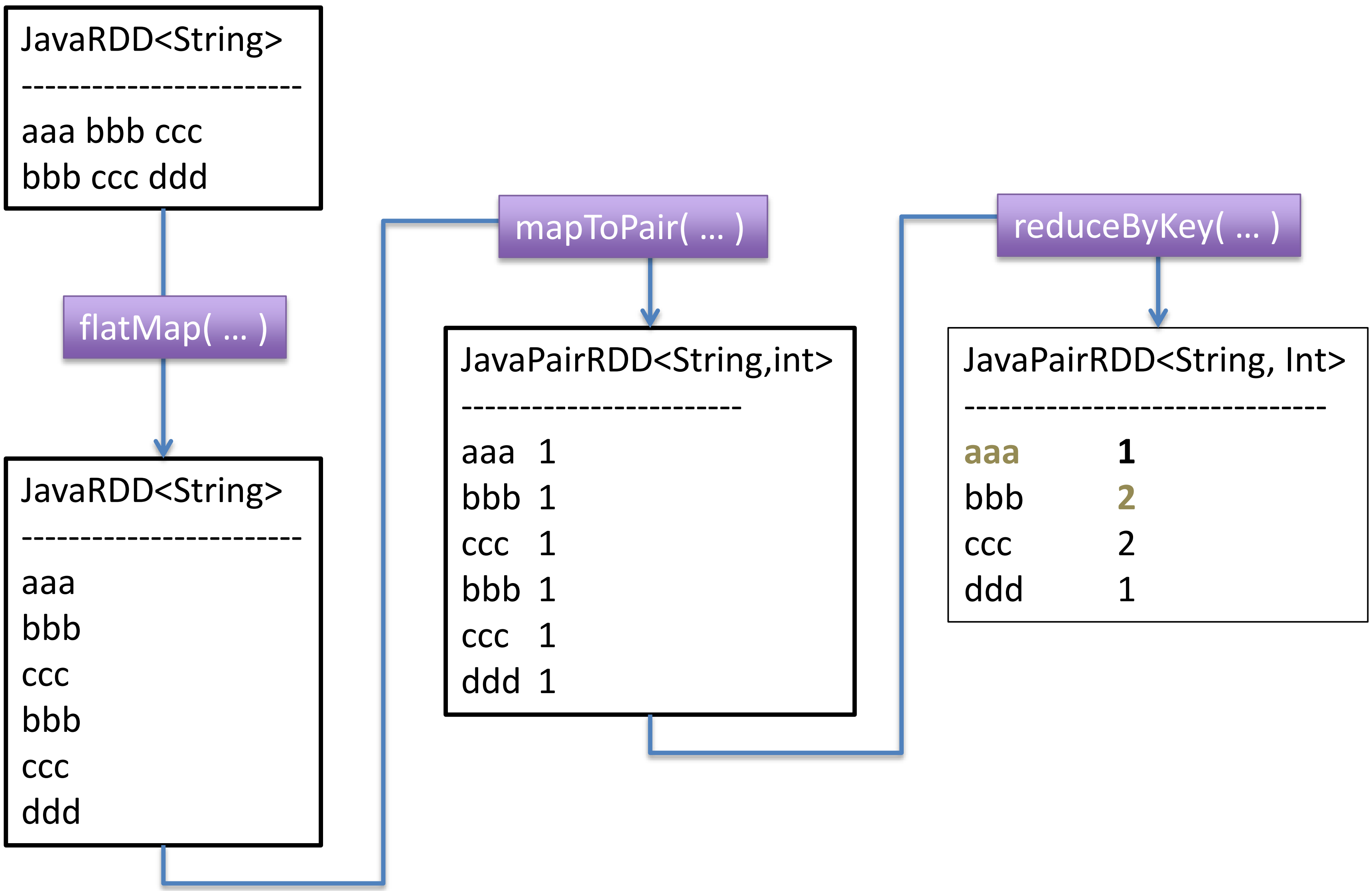
```java
JavaPairRDD<Integer,Integer> result =
        prdd.reduceByKey(new Function2<Integer, Integer, Integer>() {
            @Override
            public Integer call(Integer v1, Integer v2) throws Exception {
                return v1 + v2;
            }
        });
```

Convert MR Job into Spark Job

- Map() in MR
  - flatmap() + map()/mapToPair()

- Reduce() in MR
  - reduceByKey()
  - groupByKey() + mapValue()

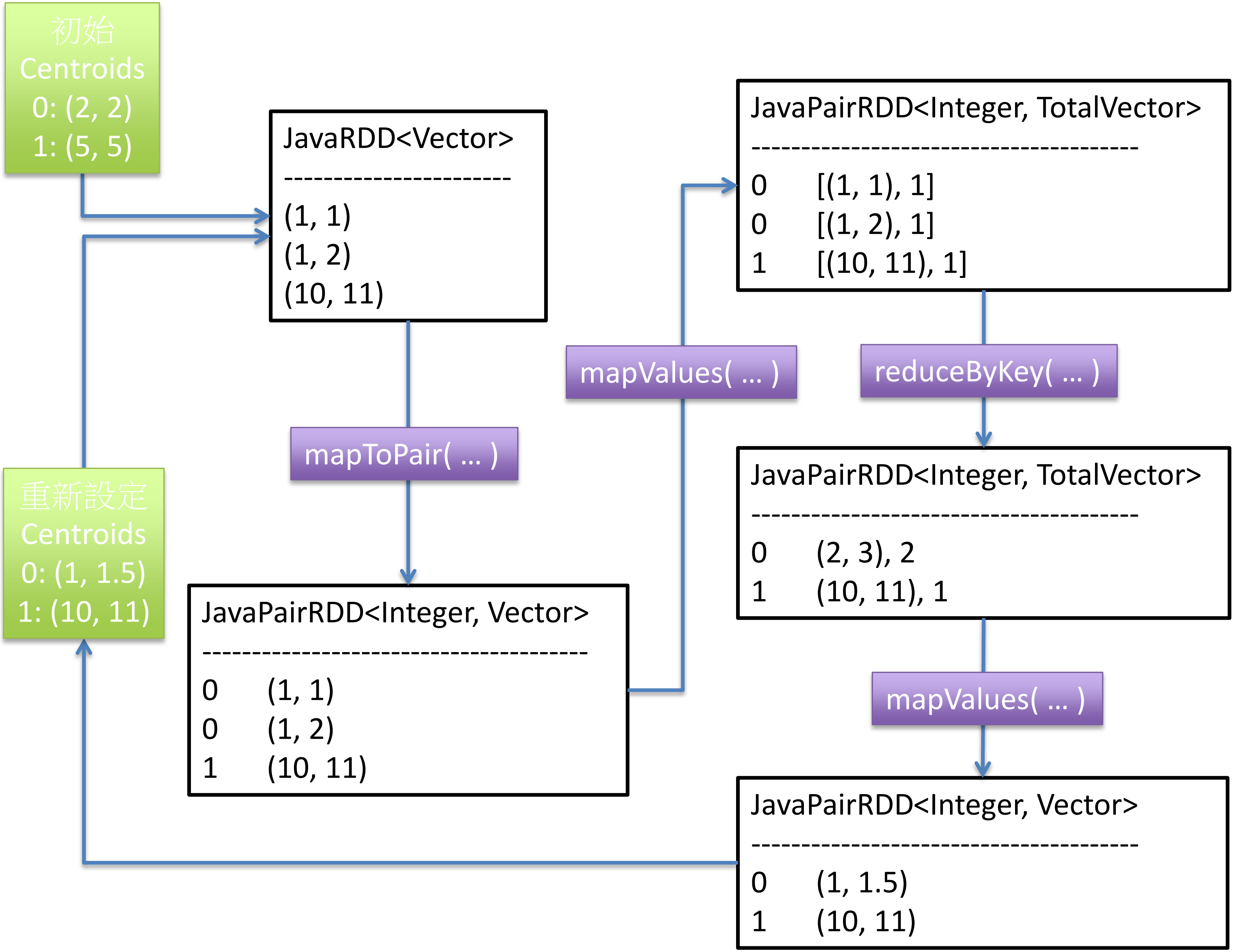
Spark RDD example
- Word Count


JavaRDD<String>
----
aaa bbb ccc
bbb ccc ddd
flatMap(…)
JavaRDD<String>
----
aaa
bbb
ccc
bbb
ccc
ddd
mapToPair(…)
JavaPairRDD<String,int>
----
aaa 1
bbb 1
ccc 1
bbb 1
ccc 1
ddd 1
reduceByKey(…)
JavaPairRDD<String, Int>
----
aaa 1
bbb 2
ccc 2
ddd 1

Labs 4: Word Count(Spark)

- Use flatMap() to split each line into multiple words
  - Use String.split() to generate String array, then use Arrays.asList() to create String iterable
- Use mapToPair() to transform word into (word, one) pair
  - return (word, 1) tuple
- Use reduceByKey() to generate (word, count) pair
  - Create Function2 class
  - Implement Integer call(Integer v1, Integer v2)

Spark RDD example
- K-means

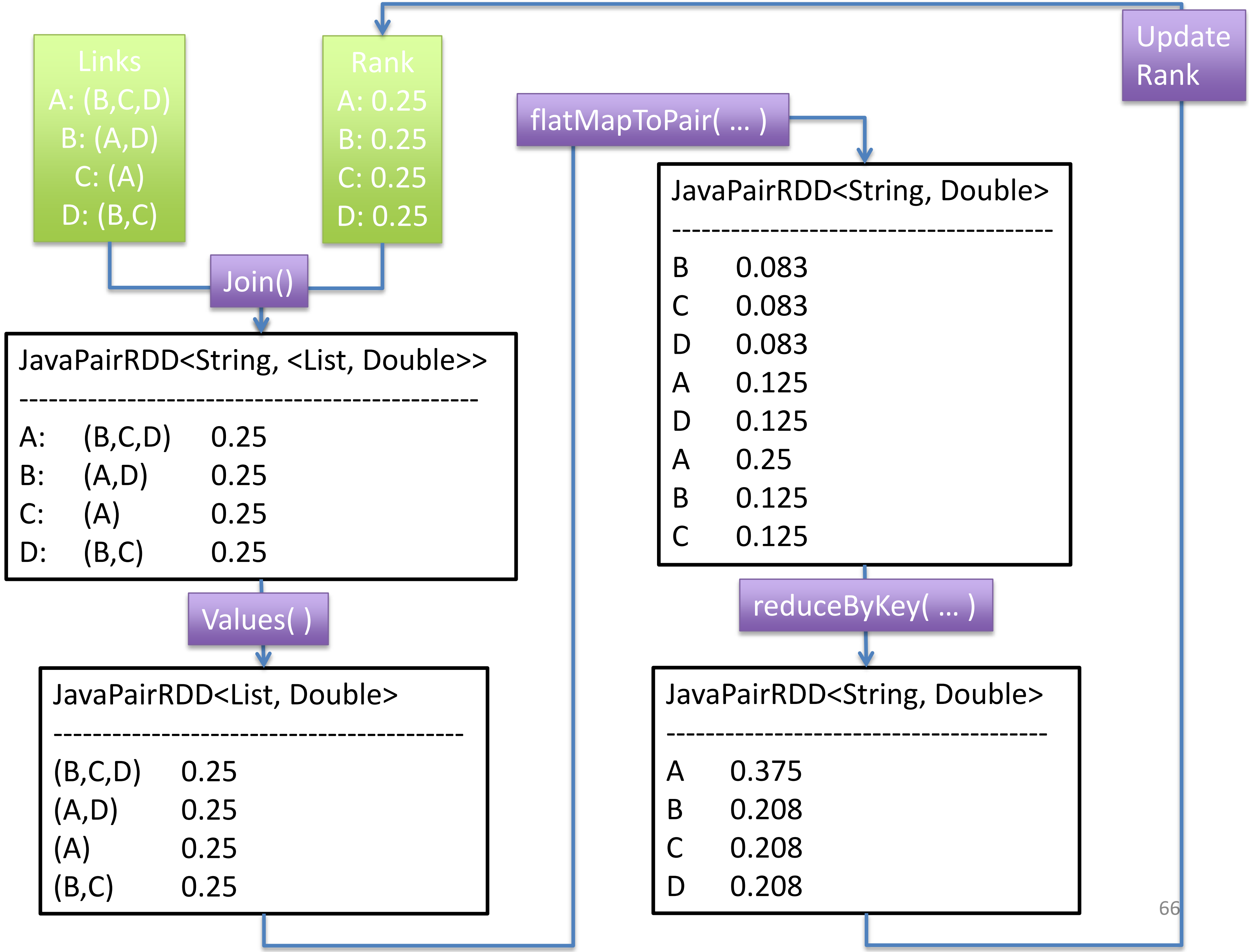


初始
Centroids
0: (2, 2)
1: (5, 5)
JavaRDD<Vector>
(1, 1)
(1, 2)
(10, 11)
mapToPair(…)
JavaPairRDD<Integer, Vector>
0 (1, 1)
0 (1, 2)
1 (10, 11)
mapValues(…)
JavaPairRDD<Integer, TotalVector>
0 [(1, 1), 1]
0 [(1, 2), 1]
1 [(10, 11), 1]
reduceByKey(…)
JavaPairRDD<Integer, TotalVector>
0 (2, 3), 2
1 (10, 11), 1
mapValues(…)
JavaPairRDD<Integer, Vector>
0 (1, 1.5)
1 (10, 11)
重新設定
Centroids
0: (1, 1.5)
1: (10, 11)

Labs 5: K-means (Spark)

- 分群, 計算每個Vector離那一組中心最近
  - Use closestPoint(v, centroids) to find the ID of nearest center, then return (ID, Vector) tuple

Spark RDD example
- Page Rank


Links
A: (B,C,D)
B: (A,D)
C: (A)
D: (B,C)
Rank
A: 0.25
B: 0.25
C: 0.25
D: 0.25
Join()
JavaPairRDD<String, <List, Double>>
A: (B,C,D) 0.25
B: (A,D) 0.25
C: (A) 0.25
D: (B,C) 0.25
Values( )
JavaPairRDD<List, Double>
(B,C,D) 0.25
(A,D) 0.25
(A) 0.25
(B,C) 0.25
flatMapToPair(…)
JavaPairRDD<String, Double>
B 0.083
C 0.083
D 0.083
A 0.125
D 0.125
A 0.25
B 0.125
C 0.125
reduceByKey(…)
JavaPairRDD<String, Double>
A 0.375
B 0.208
C 0.208
D 0.208
Update Rank

Labs 6: Page Rank (Spark)

- For each URL, calculate the PR contributed from its neighbor, then add (url, PR) tuple into results list