亦思科技
is-land Systems inc.

LET'S PLAY WITH DATA TOGETHER!!
HadoopCon 2016

Apache Spark Installation & RDD

亦思科技 鄭紹志
vito@is-land.com.tw

About me

- 鄭紹志 Vito
- 任職亦思科技
- Hadoop/HBase/Spark 相關的研究開發工作
- 目前投入的領域主要為 Scala & Spark

歡迎加入 Spark@Hsinchu ！

- Meetup - https://www.meetup.com/Apache-Spark-Hsinchu/
- Gitter 聊天室 - https://gitter.im/hubertfc/SparkHsinchu

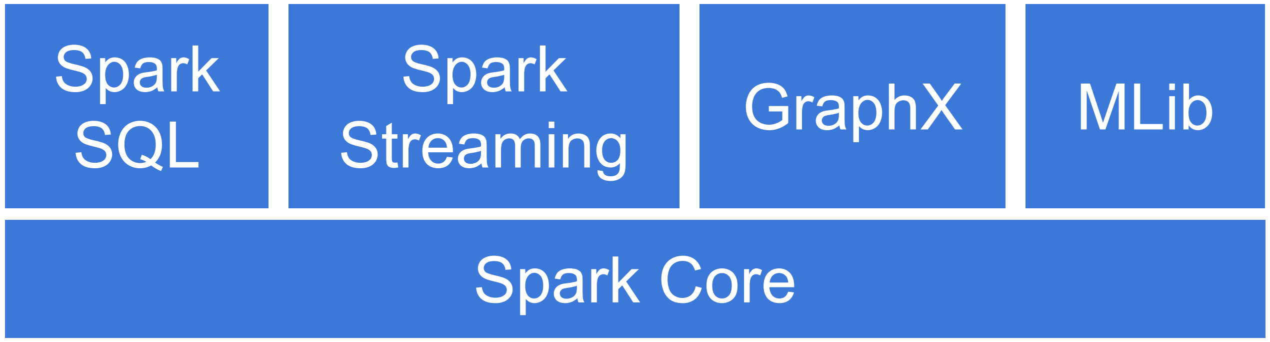
HadoopCon 2016

Agenda

- Spark overview
- Standalone cluster installation
- Spark Glossary
- RDD concept & overview
- RDD operations
- Shuffle & RDD Dependency

Spark Overview

Apache Spark is a fast and general-purpose *cluster computing system*. It provides high-level APIs in Java, Scala, Python and R, and an optimized engine that supports general execution graphs. It also supports a rich set of higher-level tools including Spark SQL for SQL and structured data processing, MLlib for machine learning, *GraphX* for graph processing, and Spark Streaming.



Cluster Mode

- Spark 支援三種叢集模式: standalone, mesos, yarn
- **Standalone** ✓
    - A simpile cluster manager included with Spark
    - 最簡單的 Spark 叢集模式
    - 可以有單機版的 cluster - 方便開發測試
- **Apache Mesos**
    - A cluster manager. Enable resource sharing, improving cluster utilization.
    - http://mesos.apache.org/
- **Hadoop YARN**
    - A resource manager in Hadoop 2
    - https://hadoop.apache.org/docs/r2.7.2/hadoop-yarn/hadoop-yarn-site/YARN.html

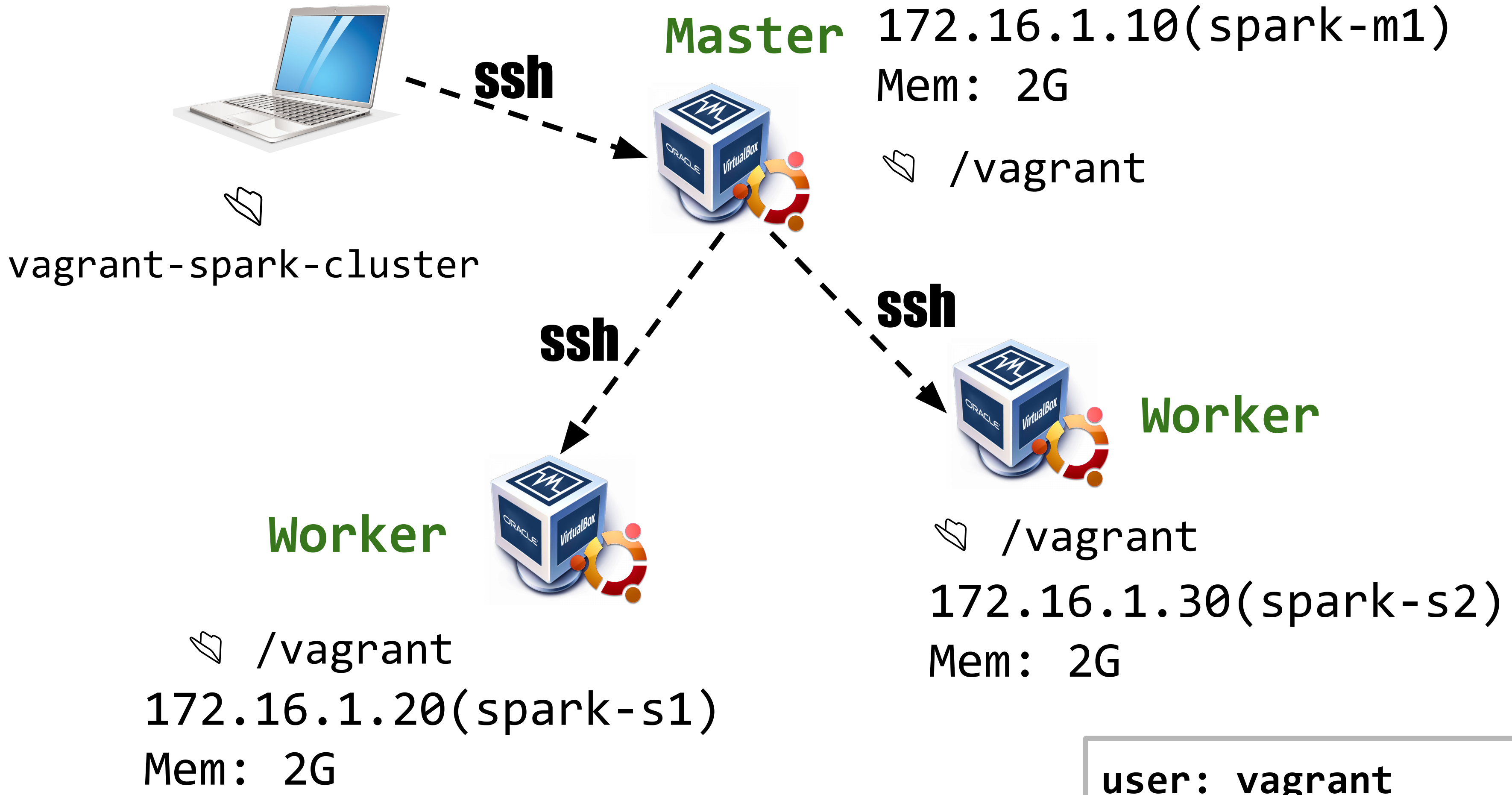
Standalone cluster installation

Spark standalone cluster

- 手工打造
- 自動化佈署 Spark: VirtualBox + Vagrant + Ansible
  - https://github.com/is-land/vagrant-spark-cluster
  - Cluster Information
    - OS: Ubuntu 14.04 LTS
    - JDK 1.8
    - Spark 1.6.2 Pre-Build for Hadoop 2.6
    - 1 Master node
    - 2 Slave node
    - Share folder: /vagrant
    - Master ssh login to Slave without password

http://spark.apache.org/docs/1.6.2/spark-standalone.html

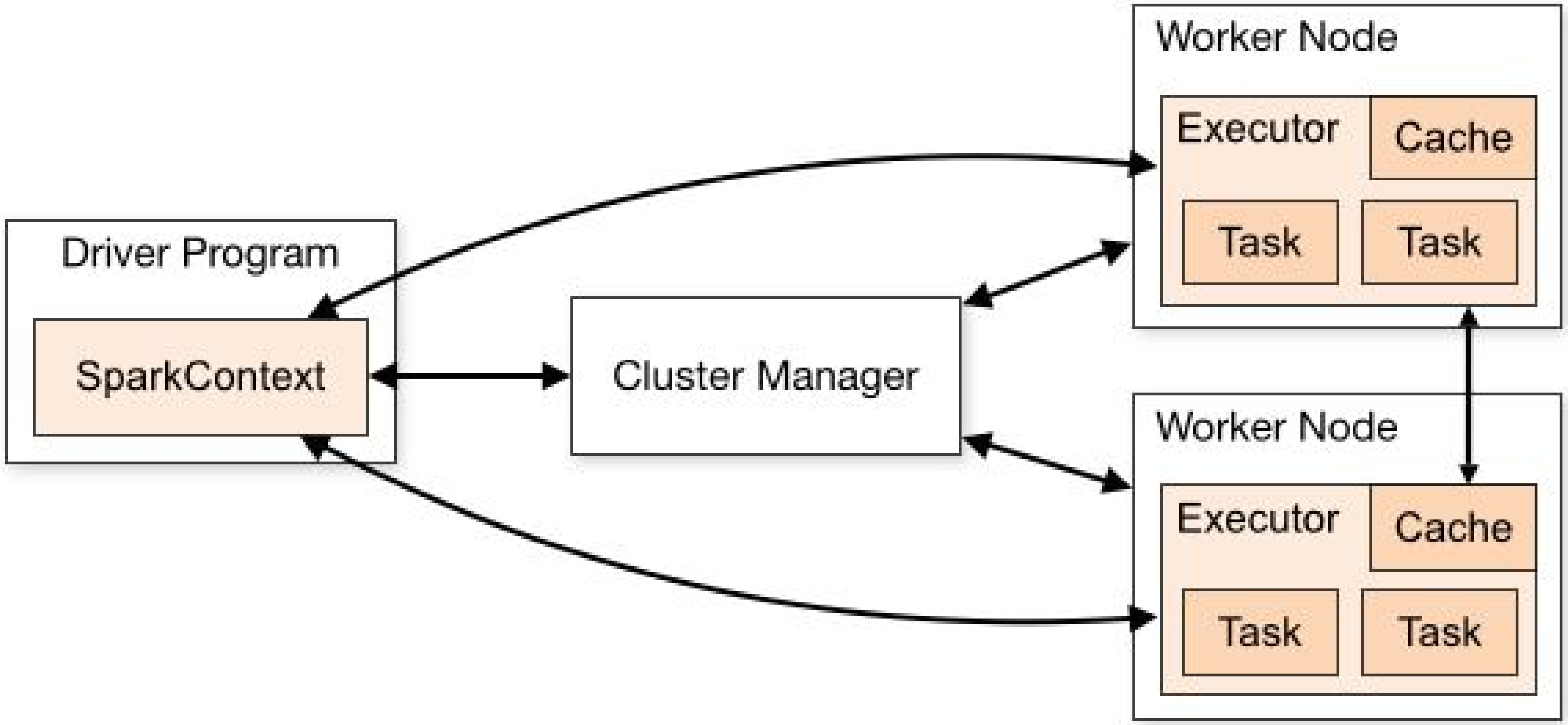
Apache Spark standalone cluster



```
user: vagrant
password: vagrant
```

HadoopCon 2016

Spark cluster architecture



Spark Installation - Master(spark-m1)

- ssh 登入 spark-m1
- 安裝 JDK 1.8.x
  - 設定環境變數 JAVA_HOME
  - 將 $JAVA_HOME/bin 加入 PATH 環境變數
- 修改 /etc/hosts, 加入以下資訊

```
172.16.1.10    spark-m1
172.16.1.20    spark-s1
172.16.1.30    spark-s2
```

- 產生 ssh key, 並複製到共享資料夾(/vagrant)內

```
vagrant@spark-m1:~$ ssh-keygen -t rsa -N "" -f "/home/vagrant/.ssh/id_rsa"
vagrant@spark-m1:~$ cp /home/vagrant/.ssh/id_rsa /vagrant/.ssh/id_rsa.master
vagrant@spark-m1:~$ cp /home/vagrant/.ssh/id_rsa.pub /vagrant/.ssh/id_rsa.master.pub
```

- 設定免密碼登入(spark-m1 --> spark-m1)

```
vagrant@spark-m1:~$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
vagrant@spark-m1:~$ sudo su -
root@spark-m1:~$ cat /vagrant/.ssh/id_rsa.master.pub >> ~/.ssh/authorized_keys
```

Spark Installation - Master(spark-m1)

- 下載 & 解壓縮 Spark 1.6.2(pre-build for Hadoop 2.6)

```
vagrant@spark-m1:~$ wget http://d3kbcqa49mib13.cloudfront.net/spark-1.6.2-bin-hadoop2.6.tgz
vagrant@spark-m1:~$ tar zxvf spark-1.6.2-bin-hadoop2.6.tgz
```

- 修改 ~/.profile 加入環境變數

```
export SPARK_HOME=/home/vagrant/spark-1.6.2-bin-hadoop2.6
```

- 登出 ssh, 再次登入 spark-m1. 並執行 Spark master server

```
vagrant@spark-m1:~$ cd spark-1.6.2-bin-hadoop2.6
vagrant@spark-m1:~/spark-1.6.2-bin-hadoop2.6$ sbin/start-master.sh
```

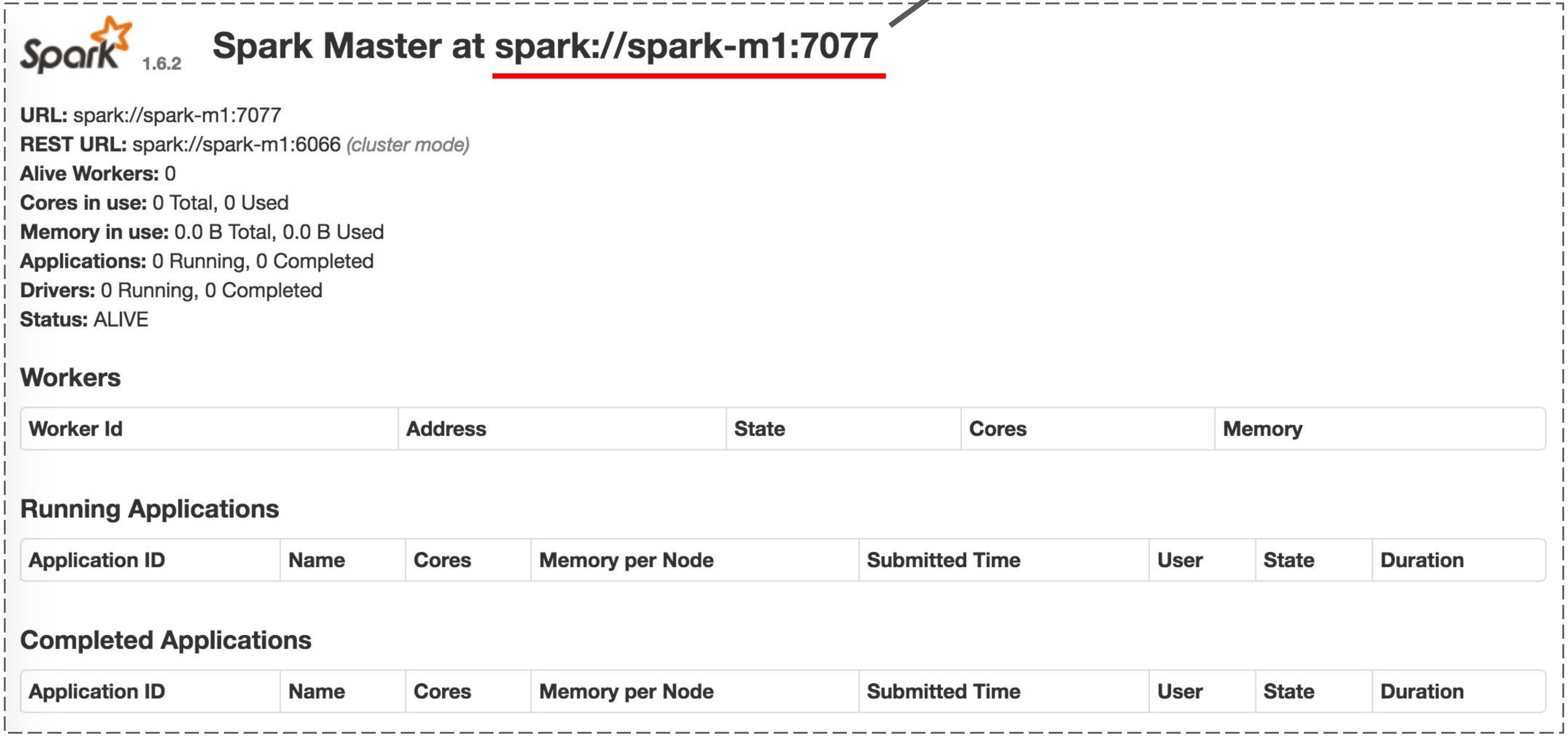
- 檢查是否正常啟動(process & log)

```
vagrant@spark-m1:~/spark-1.6.2-bin-hadoop2.6$ jps -l
10355 sun.tools.jps.Jps
9946 org.apache.spark.deploy.master.Master
vagrant@spark-m1:~/spark-1.6.2-bin-hadoop2.6$ tail -n 100
logs/spark-vagrant-org.apache.spark.deploy.master.Master-1-spark-m1.out
```

Spark WebUI

- `http://spark-m1:8080`

master url



Spark 1.6.2 **Spark Master at spark://spark-m1:7077**

URL: spark://spark-m1:7077
REST URL: spark://spark-m1:6066 *(cluster mode)*
Alive Workers: 0
Cores in use: 0 Total, 0 Used
Memory in use: 0.0 B Total, 0.0 B Used
Applications: 0 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers

Worker Id	Address	State	Cores	Memory

Running Applications

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

Completed Applications

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

Spark Installation - Slave(Worker)

- 共安裝兩台 worker(spark-s1, spark-s2), 安裝方式相同
- ssh 登入 spark-s1
- 安裝 JDK 1.8.x
  - 設定環境變數 export JAVA_HOME
  - 將 $JAVA_HOME/bin 加入 PATH 環境變數
- 修改 **/etc/hosts**, 加入以下資訊

```
172.16.1.10     spark-m1
172.16.1.20     spark-s1
172.16.1.30     spark-s2
```

- 設定 ssh 免密碼登入(spark-m1 --> spark-s1)

```
vagrant@spark-s1:~$ cat /vagrant/.ssh/id_rsa.master.pub >> ~/.ssh/authorized_keys
vagrant@spark-s1:~$ sudo su -
root@spark-s1:~$ cat /vagrant/.ssh/id_rsa.master.pub >> ~/.ssh/authorized_keys
```

- 下載 & 解壓縮 Spark 1.6.2

```
vagrant@spark-s1:~$ wget http://d3kbcqa49mib13.cloudfront.net/spark-1.6.2-bin-hadoop2.6.tgz
vagrant@spark-s1:~$ tar zxvf spark-1.6.2-bin-hadoop2.6.tgz
```

Spark Installation - Slave(Worker)

- 修改 ~/.profile 加入環境變數

```
export SPARK_HOME=/home/vagrant/spark-1.6.2-bin-hadoop2.6
```

- 登出 ssh, 再次登入 spark-s1. 並執行 Spark worker

```
vagrant@spark-s1:~$ cd spark-1.6.2-bin-hadoop2.6
vagrant@spark-s1:~/spark-1.6.2-bin-hadoop2.6$ sbin/start-slave.sh spark://spark-m1:7077
```

- 檢查是否正常啟動(process & log)

```
vagrant@spark-s1:~/spark-1.6.2-bin-hadoop2.6$ jps -l
9111 sun.tools.jps.Jps
9055 org.apache.spark.deploy.worker.Worker
vagrant@spark-s1:~/spark-1.6.2-bin-hadoop2.6$ tail -n 100
logs/spark-vagrant-org.apache.spark.deploy.worker.Worker-1-spark-s1.out
```

- 停止 Spark worker

```
vagrant@spark-s1:~/spark-1.6.2-bin-hadoop2.6$ sbin/stop-slave.sh
```

Spark cluster startup & configuration

- 啟動 cluster

```
vagrant@spark-m1:~/spark-1.6.2-bin-hadoop2.6$ sbin/start-all.sh
```

- 停止 cluster

```
vagrant@spark-m1:~/spark-1.6.2-bin-hadoop2.6$ sbin/stop-all.sh
```

- 重要設定檔:
- $SPARK_HOME/conf/slaves
  - 設定所有 worker 的 hostname 資訊
  - 若找不到 slaves 則 Spark 會在 master 另外起一個 worker, 不須另外建立 VM, 方便測試使用.

```
spark-s1
spark-s2
```

- $SPARK_HOME/conf/spark-defaults.conf
  - cp conf/spark-defaults.conf.template conf/spark-defaults.conf
  - Spark 預設讀取的 conf 檔案

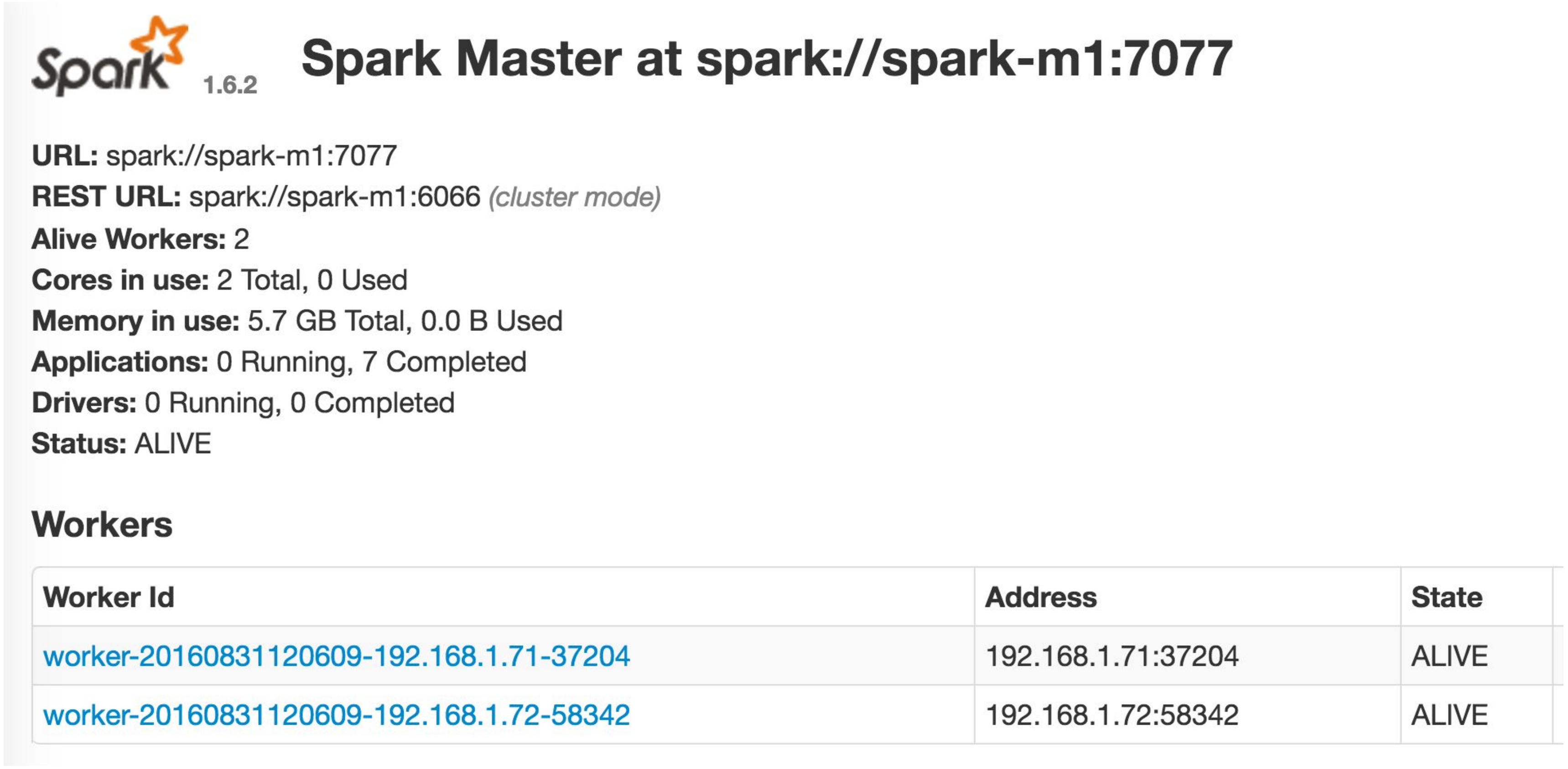
```
spark.master                      spark://spark-m1:7077
spark.eventLog.enabled            true
```

  - spark.eventLog.enabled=true 時, 須要在 master 建立資料夾: /tmp/spark-events

https://spark.apache.org/docs/1.6.2/spark-standalone.html#starting-a-cluster-manually

Spark Web UI

- 預設情形下, spark 各項執行資訊只能在執行時期看到
- 保存 application 執行記錄: spark.eventLog.enabled 設為 true


Spark 1.6.2 Spark Master at spark://spark-m1:7077

URL: spark://spark-m1:7077
REST URL: spark://spark-m1:6066 *(cluster mode)*
Alive Workers: 2
Cores in use: 2 Total, 0 Used
Memory in use: 5.7 GB Total, 0.0 B Used
Applications: 0 Running, 7 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers

Worker Id	Address	State
worker-20160831120609-192.168.1.71-37204	192.168.1.71:37204	ALIVE
worker-20160831120609-192.168.1.72-58342	192.168.1.72:58342	ALIVE

Execute spark shell

- bin/spark-shell 提供 Scala 的交談式執行環境
- bin/pyspark 提供 Python 的交談試執行環境
- spark-shell, pyspark 內部均呼叫 bin/spark-submit

```
$ bin/spark-shell
16/09/06 23:52:27 WARN NativeCodeLoader: Unable to load native-hadoop library
for your platform... using builtin-java classes where applicable
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 1.6.2
      /_/

Using Scala version 2.10.5 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_102)
Type in expressions to have them evaluated.
Type :help for more information.
Spark context available as sc.

SQL context available as sqlContext.

scala>
```

https://spark.apache.org/docs/1.6.2/programming-guide.html#using-the-shell
https://spark.apache.org/docs/1.6.2/#running-the-examples-and-shell

HadoopCon 2016

Execute spark shell - Word count example

```
scala> :paste
// Entering paste mode (ctrl-D to finish)

val words = Seq("copyright", "permission", "software")
val lines = sc.textFile("/vagrant/data/licenses/*.txt")
val counts = lines.flatMap(line=>line.split(" "))
                  .filter( w=>words.contains(w.toLowerCase) )
                  .map( (_, 1) )
                  .reduceByKey( _ + _ )
val result = counts.collect()

// Exiting paste mode, now interpreting.

words: Seq[String] = List(copyright, permission, software)
lines: org.apache.spark.rdd.RDD[String] = /vagrant/data/licenses/*.txt
MapPartitionsRDD[1] at textFile at <console>:29
counts: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[5] at reduceByKey
at <console>:34
result: Array[(String, Int)] = Array((software,42), (Software,32),
(SOFTWARE,49), (copyright,67), (Copyright,40), (Permission,15), (permission,15),
(COPYRIGHT,47))

scala> result.foreach(println)
```

Execute spark example - run-example

- spark 預先打包好的 example jar
  - lib/spark-examples-1.6.2-hadoop2.6.0.jar
- 設定環境變數 *MASTER* 指定 master, 若無則以 local mode 執行
- bin/run-example 內部呼叫 bin/spark-submit

```
$ MASTER=spark://spark-m1:7077 bin/run-example SparkPi 50
16/09/07 00:58:52 WARN NativeCodeLoader: Unable to load native-hadoop library
for your platform... using builtin-java classes where applicable
Pi is roughly 3.142252
$
```

https://spark.apache.org/docs/1.6.2/#running-the-examples-and-shell

Spark Glossary

Spark Glossary

- Application
  - 使用者開發的 Spark 程式統稱, 範圍包含 Driver 與各節點的 Executor
- Driver program
  - 在 Application 裡含有 main() 及建立 SparkContext 的程式
- Cluster manager
  - 在 cluster 上負責取得可用資源的服務
- Worker node
  - 泛指在 cluster 裡可以執行 application 的節點
- Executor
  - 在 worker 上啟動 application 的 process(JVM). 每個 application 都有屬於自己的 Executor

Spark Glossary

- Job
  - RDD action 所觸發的動作即為一個 Job. 每個 Job 均包含多個可平行運算的 task
- Stage
  - 一個 Job 裡的所有 task 會被分解成多組的 task. 每一組 task 即為一個 stage
- Task
  - 在 Executor 裡實際執行的任務
- Partition
  - 將 RDD 中的 data 切分為不同的分區, 通常一個 partition 的資料由一個 task 處理

RDD concept & overview

SparkConf & SparkContext

- *SparkConf* 物件
  - 記錄執行 application(driver) 所需要的資訊
  - 會讀取所有名稱以 ‘**spark.**’ 開頭的 Java 系統資訊
- *SparkContext* 物件
  - 提供 Driver 執行 Spark Job 的進入點
  - 建立 RDDs, accumulators, boradcast varaibles
  - run jobs

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

RDD's trait

- RDD - Resilient Distributed Dataset
  - 顧名思義 - "有彈性的" "分散式" "資料集"
  - 一種物件: RDD[String], RDD[Int]

- RDD 特性
  - 容錯, Fault tolerance
  - 平行運算, parallel
  - 不可變, Immutable(read only)
  - 資料分區, Partitioned data
  - 記憶體儲存, In-Memory
  - 有資料型別, Typed
  - 快取, Cacheable

RDD Creation

- 從既有的 collection 建立 RDD: *parallelize*, *makeRDD*

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

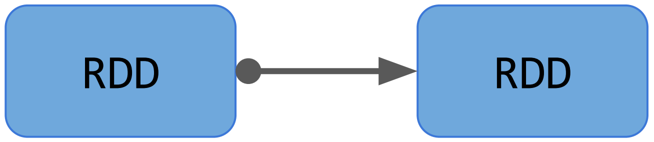
- 由外部讀取: *textFile*
  - Local file system - "file://"

```
scala> val distFile = sc.textFile("data.txt")
```

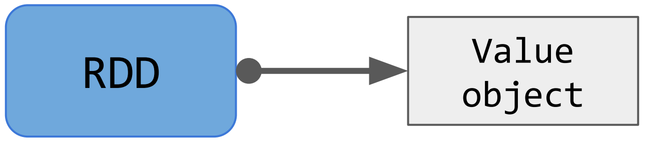
  - HDFS - "hdfs://"
  - Amazon S3 - "s3n://"
  - Cassandra, HBase

RDD Operations

- Transformation
  - 由現有的 RDD 建立一個新的 RDD
  - Lazy, 不立即進行運算
  - *map*, *flatMap*, *filter*...



- Action
  - 在 RDD 上進行運算, 傳回實際值到 Driver
  - *collect*, *count*, *reduce*



String, Int,
Collection...

Word count example(again)

```
val words = Seq("copyright", "permission", "software")

val lines = sc.textFile("/vagrant/data/licenses/*.txt")

val counts = lines.flatMap( line=>line.split(" ") )
                  .filter( w=>words.contains(w.toLowerCase) )
                  .map( (_, 1) )
                  .reduceByKey( _ + _ )

val result = counts.collect()
```

Understand Scala - very, very quickly

```
val words = Seq("copyright", "permission", "software")
```

↳**val** 宣告一個不可改變的變數, **Seq** 是一個 **scala** 的 **collection** 的物件

```
val lines = sc.textFile("/vagrant/data/licenses/*.txt")
```

↳**sc** 是 **SparkContext** 物件

```
val counts = lines.flatMap( line=>line.split(" ") )
```

↳**flatMap** 傳入一個匿名函式, **line** 是傳 入值, **split** 傳出 **Array**

```
                    .filter( w=>words.contains(w.toLowerCase) )
```

↳**filter** 傳入一個匿名函式, **w** 是傳入值, **contains** 傳出 **Boolean**

```
                    .map( (_, 1) )
```

↳_ 表示第一個傳入的參數, **(_,1)** 表示一個 **Tuple2** 物件

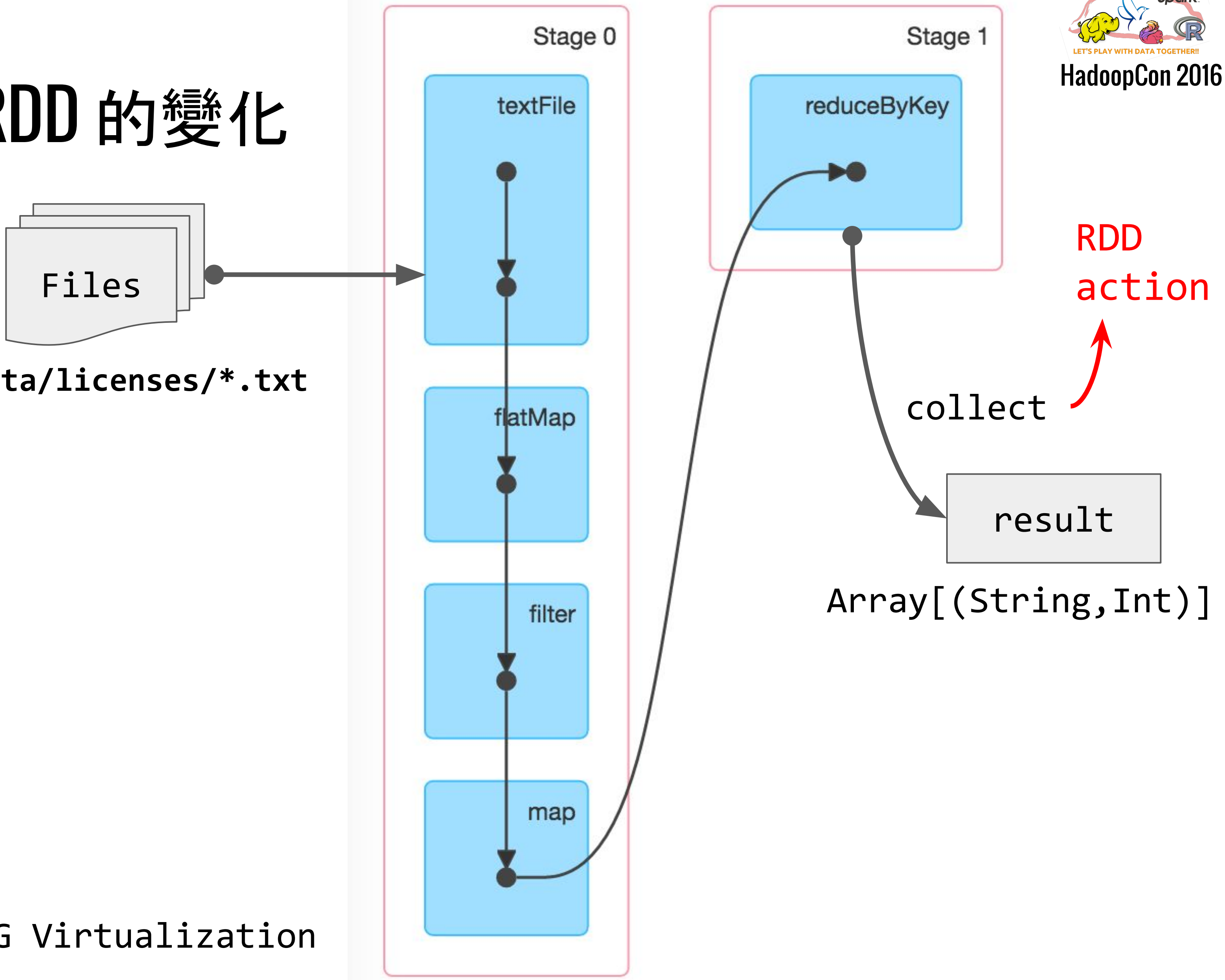
```
                    .reduceByKey( _ + _ )
```

↳第一個 _ 表示第一個參數, 第二個 _ 表示第二個參數

```
val result = counts.collect()
```

↳**RDD action**

觀察 RDD 的變化



/vagrant/data/licenses/*.txt

Array[(String,Int)]

DAG Virtualization

觀察 RDD 的變化 - Spark shell

RDD.*toDebugString*

```
scala> counts.toDebugString
res1: String =
(35) ShuffledRDD[5] at reduceByKey at <console>:34 []
 +-(35) MapPartitionsRDD[4] at map at <console>:33 []
    |   MapPartitionsRDD[3] at filter at <console>:32 []
    |   MapPartitionsRDD[2] at flatMap at <console>:31 []
    |   /vagrant/data/licenses/*.txt MapPartitionsRDD[1] at textFile at <console>:29 []
    |   /vagrant/data/licenses/*.txt HadoopRDD[0] at textFile at <console>:29 []

scala>
```

RDD Persistence

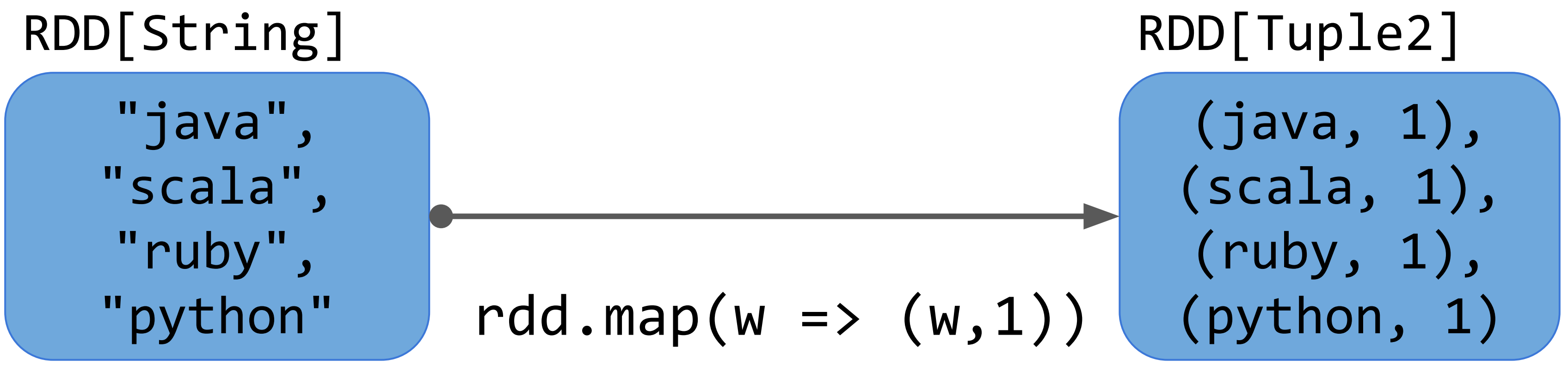
- RDD.*persist()* , RDD.*cache()*
- RDD 的結果保留在記憶體內，下次使用不須重新計算

```
val words = Seq("copyright", "permission", "software")

val lines = sc.textFile("/vagrant/data/licenses/*.txt")
val allWords = lines.flatMap(line=>line.split(" "))
allWords.persist()

val counts = allWords.filter( myfilter1 )
                     .map( (_, 1) ).reduceByKey( _ + _ )

val counts2 = allWords.filter( myfilter2 )
                      .map( (_, 1) ).reduceByKey( _ + _ )
```

RDD operations - transformation

RDD.map()

```
Return a new RDD by applying a function to all
elements of this RDD.
```



```
RDD[String]                                              RDD[Tuple2]
"java",                                                  (java, 1),
"scala",                                                 (scala, 1),
"ruby",                                                  (ruby, 1),
"python"          rdd.map(w => (w,1))                    (python, 1)
```

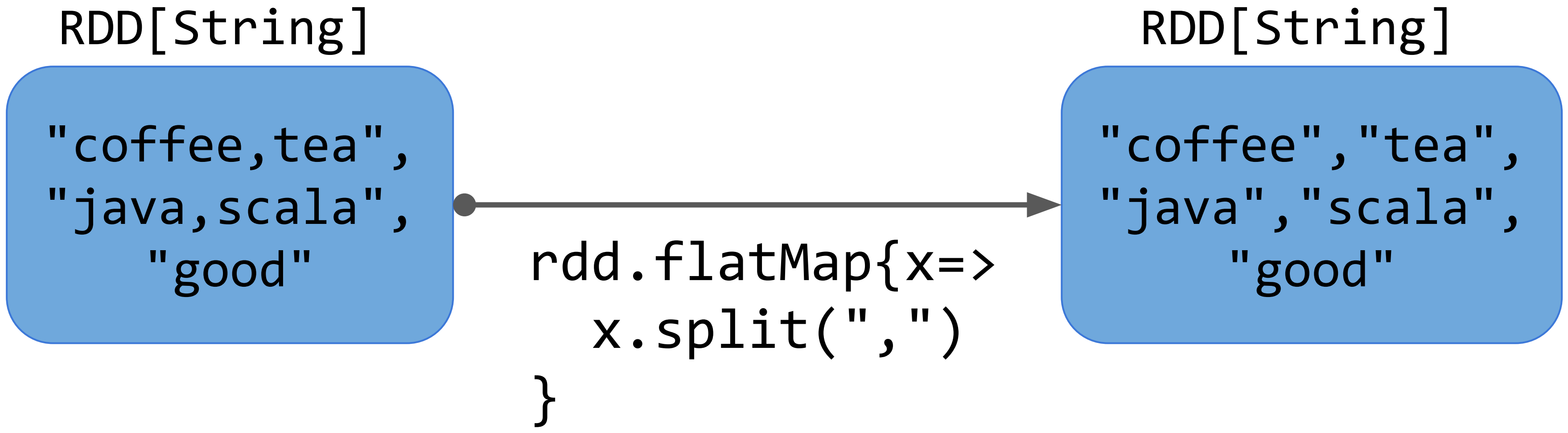
```
rdd.map {w => (w,1)}
```

匿名函數
anonymous function

count 數量不變
資料型態可變

RDD.flatMap()

Return a new RDD by first applying a function to all elements of this RDD, and then flattening the results.


RDD[String]

```
"coffee,tea",
"java,scala",
"good"
```

```
rdd.flatMap{x=>
  x.split(",")
}
```

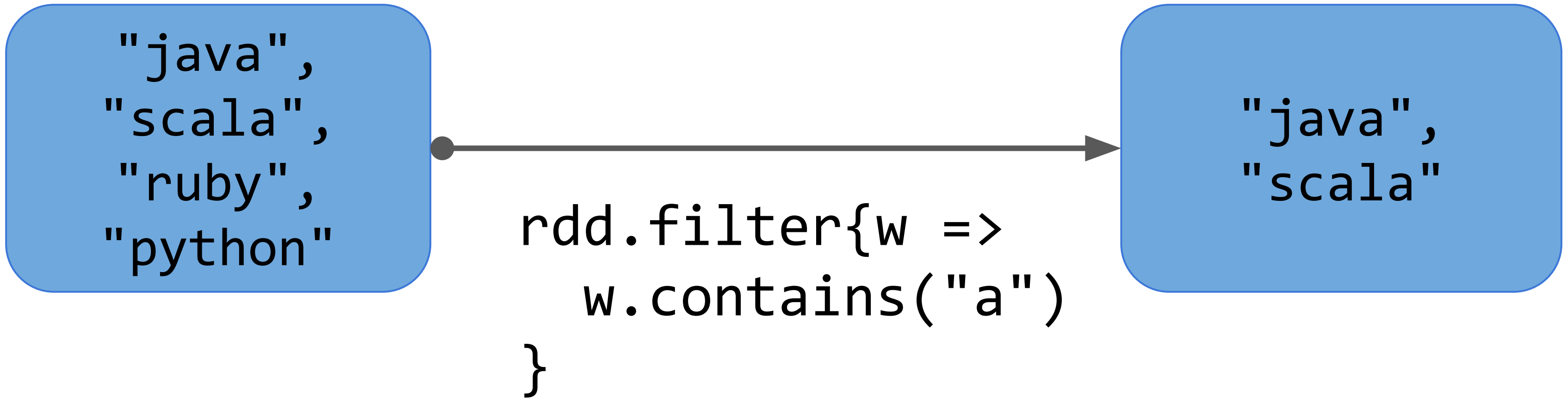
RDD[String]

```
"coffee","tea",
"java","scala",
"good"
```

count 數量可變
資料型態可變

RDD.filter()

Return a new RDD containing only the elements that satisfy a predicate.



count 數量改變
資料型態不變

Key-Value Pairs

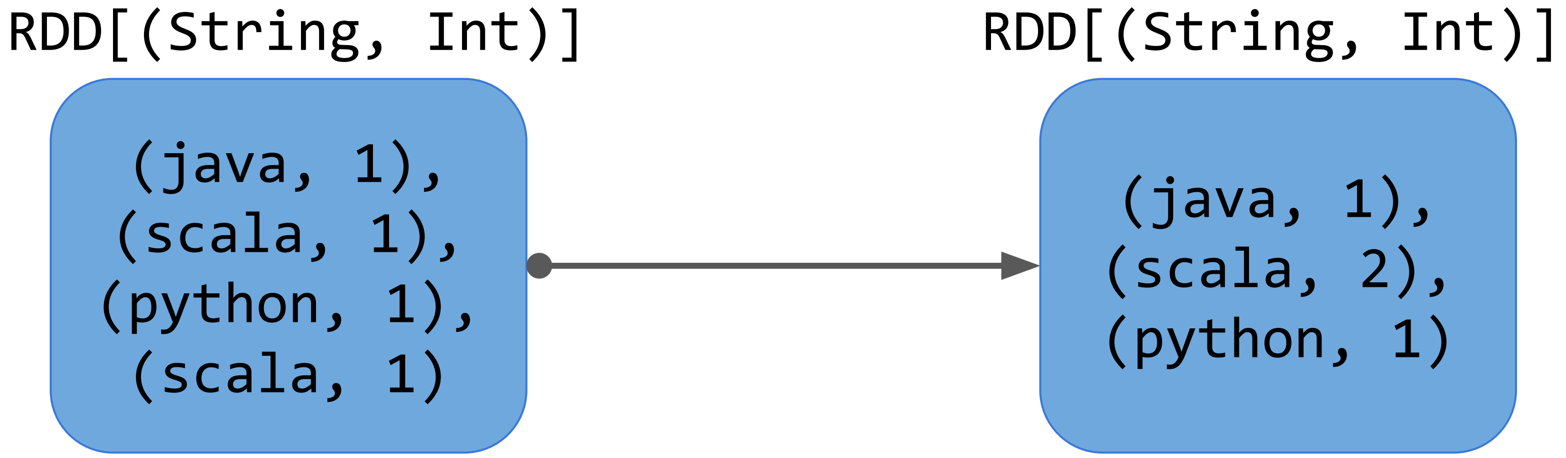
- Scala 物件: *Tuple2*
  - Scala 三種不同寫法
  - 取第一個值: _1
  - 取第二個值: _2

```
scala> val t1 = ("java", 100)
t1: (String, Int) = (java,100)

scala> val t2 = "java" -> 100
t2: (String, Int) = (java,100)

scala> val t3 = new Tuple2("java", 100)
t3: (String, Int) = (java,100)

scala> println(t1._1 + "/" + t1._2)
java/100
```

- *Tuple2* 裡的 *_1* 作為 key, *_2* 作為 value
- Key-Value Pairs 最常見的應用場景是在進行 group, aggregation 等需要 shuffle 的操作, 如 *groupByKey*, *reduceByKey*, *join* ...等
- 參考 API: PairRDDFunctions

RDD.reduceByKey()

Merge the values for each key using an associative reduce function. This will also perform the merging locally on each mapper before sending results to a reducer, similarly to a "combiner" in MapReduce. Output will be hash-partitioned with the existing partitioner/ parallelism level.



```
rdd.reduceByKey{(v1,v2)=>
  v1 + v2
}
```

=

```
rdd.reduceByKey(_ + _)
```

v1 v2

RDD operations - action

HadoopCon 2016

RDD actions

Action	Meaning
reduce(*func*)	Aggregate the elements of the dataset using a function *func* (which takes two arguments and returns one). The function should be commutative and associative so that it can be computed correctly in parallel.
collect()	Return all the elements of the dataset as an array at the driver program. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data.
count()	Return the number of elements in the dataset.
first()	Return the first element of the dataset (similar to take(1)).

HadoopCon 2016

RDD actions

Action	Meaning
saveAsTextFile(*path*)	Write the elements of the dataset as a text file (or set of text files) in a given directory in the local filesystem, HDFS or any other Hadoop-supported file system. Spark will call toString on each element to convert it to a line of text in the file.
countByKey()	Only available on RDDs of type (K, V). Returns a hashmap of (K, Int) pairs with the count of each key.
foreach(*func*)	Run a function *func* on each element of the dataset. This is usually done for side effects such as updating anAccumulator or interacting with external storage systems.

亦思科技
is-land Systems inc.

More transformations and actions

- Spark programming guide - Transformations
- Spark programming guide - Actions
- Unit Test - 最佳學習材料
  - org.apache.spark.rdd

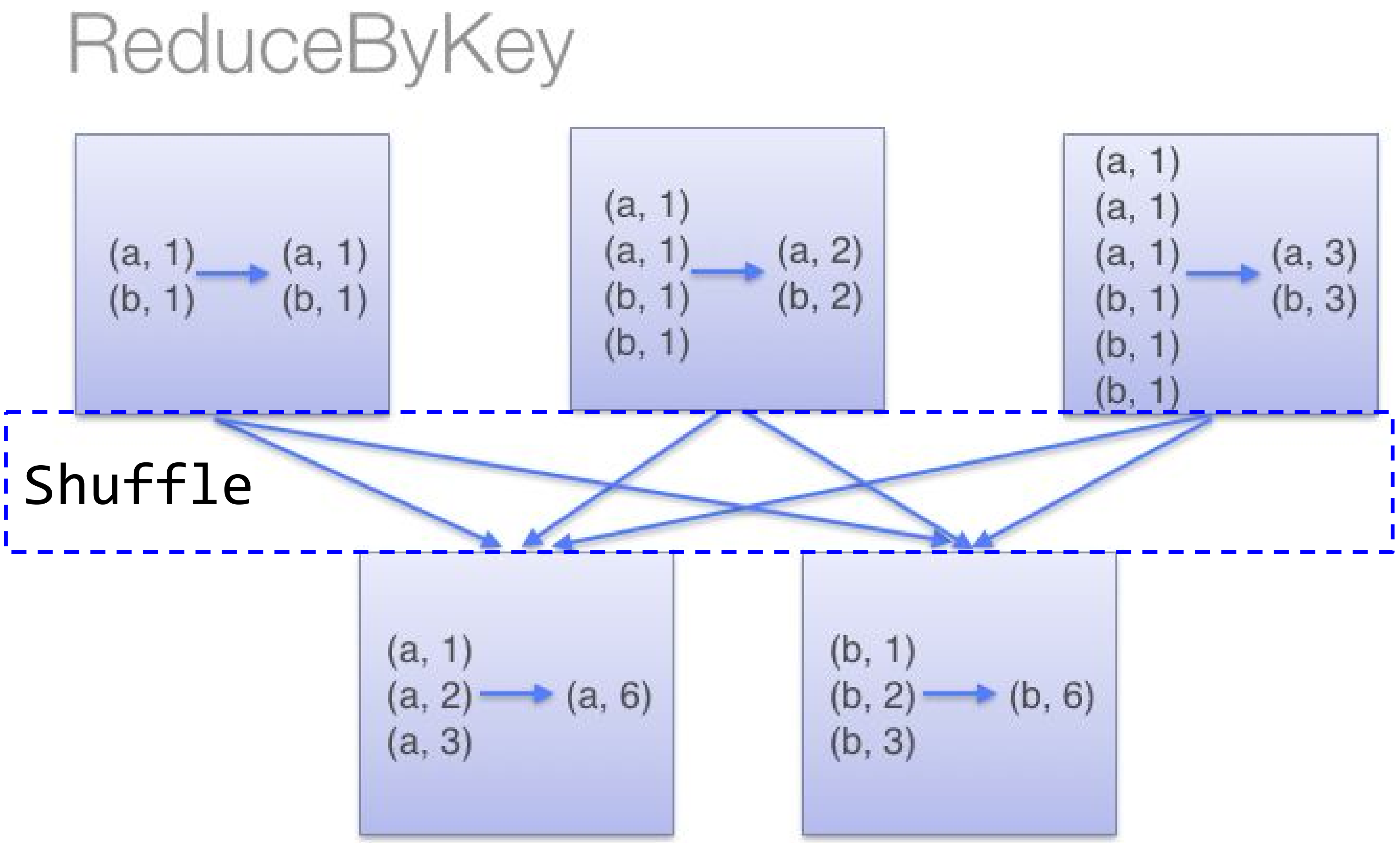
Shuffle & RDD Dependency

Shuffle

- 資料重組 – Shuffle 是一種重新分配 data 的機制，跨越不同的 partition 進行資料重組
- 通常會涉及跨越 Executor 或實體機器的資料複製
- 代價高昂
    - 磁碟 I/O
    - 資料序列化
    - 網路 I/O
- Shuffle 通常是 spark job performance tunning 的重點

HadoopCon 2016

Shuffle

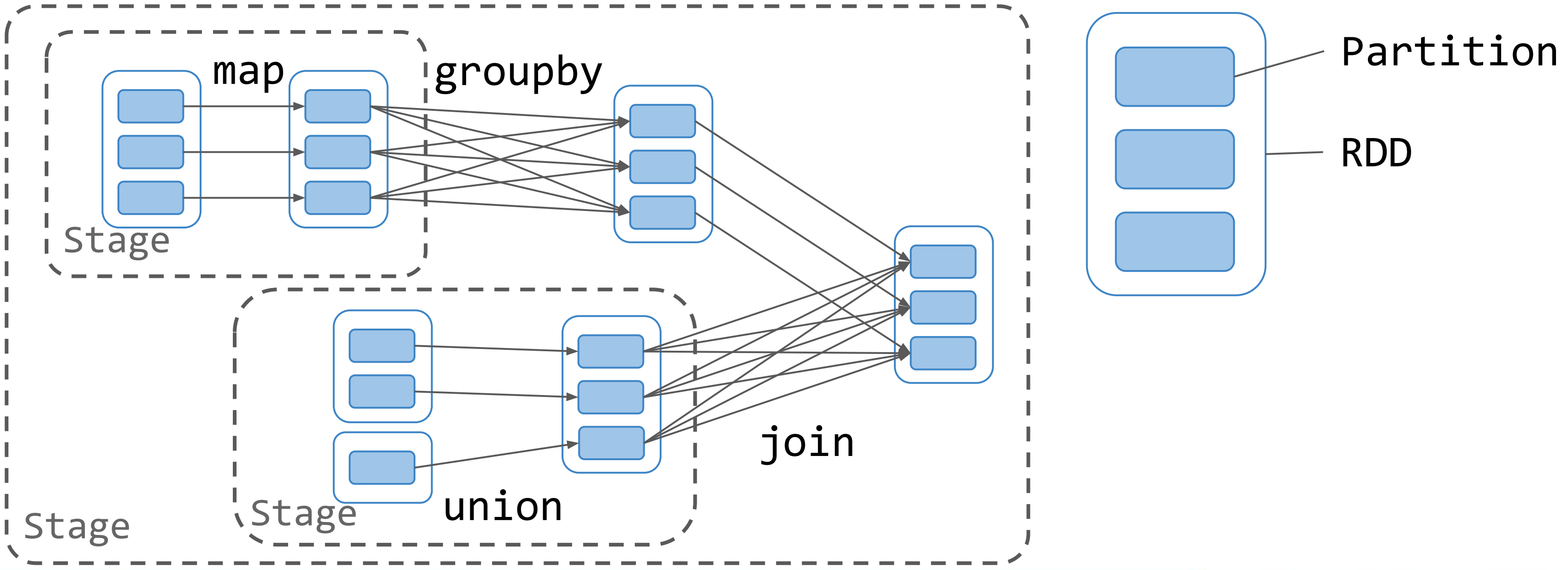


https://databricks.gitbooks.io/databricks-spark-knowledge-base/content/best_practices/prefer_reducebykey_over_groupbykey.html

HadoopCon 2016

RDD Dependency

- Narrow dependency
  - child RDD 只依賴 parent RDD 固定的 partition
- Wide dependency
  - child RDD 每一個 partition 均依賴 parent RDD 所有的 partition



Thank you !