大數據分析工具工作坊
- Spark建置與開發

國網中心核心技術組

莊家雋

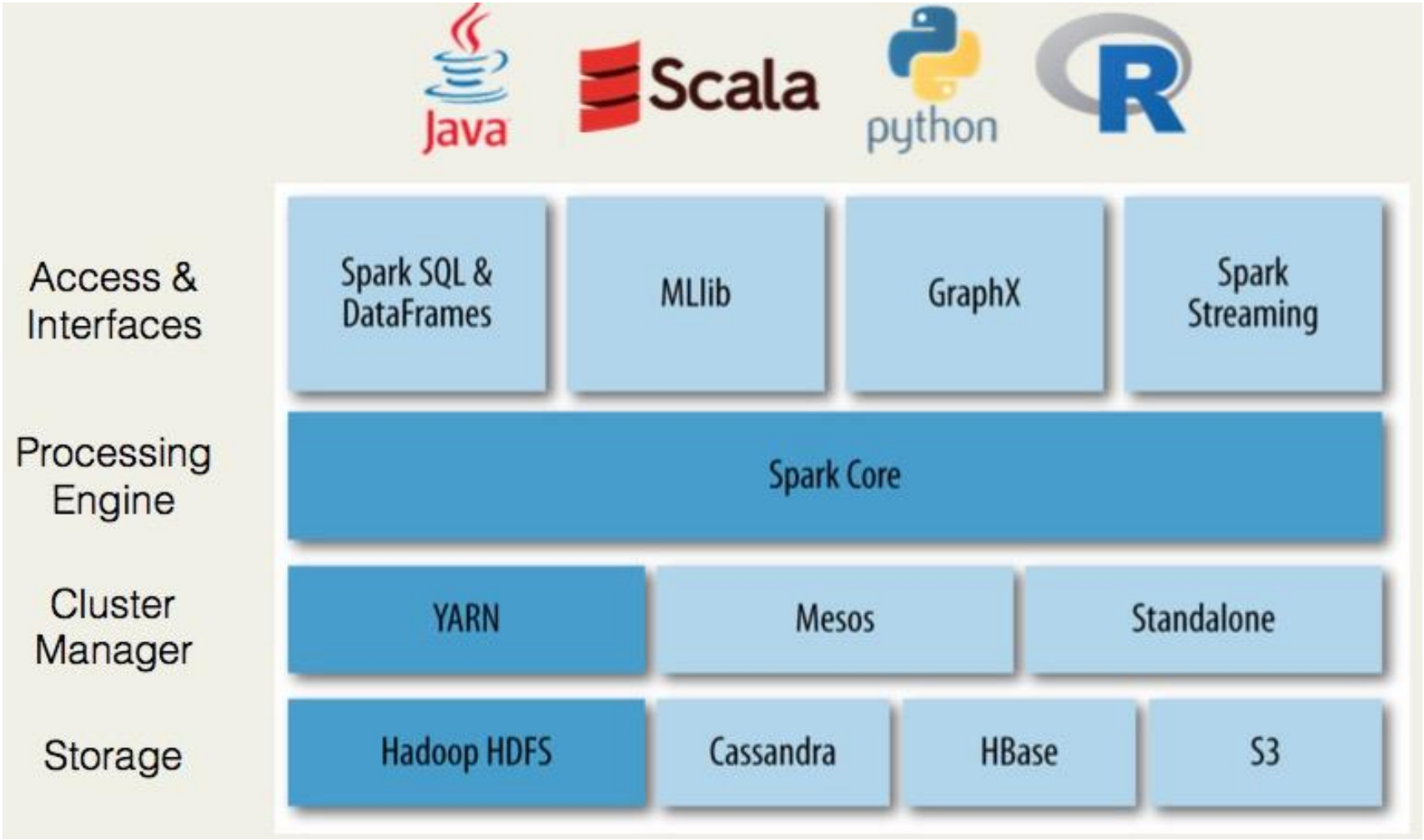
Outline

- Part I : Quick tour of Spark
  - Spark Recap
  - Prepare Lab VM
  - Run first Spark example
- Part II: Setup Cluster Lab
  - Prepare VirtualBox network
  - Prepare Cluster VMs

Outline

- Part III: Cluster Manager
  - Setup HDFS
  - Setup Spark Standalone Cluster
- Part IV: Run Spark on Cluster
  - Spark application
  - Components of Execution
    - Application, Job, stage, task, RDD, partition, DAG

Spark Stack


Java
Scala
python
R
Access & Interfaces
Spark SQL & DataFrames
MLlib
GraphX
Spark Streaming
Processing Engine
Spark Core
Cluster Manager
YARN
Mesos
Standalone
Storage
Hadoop HDFS
Cassandra
HBase
S3

Exercise:
Setup Lab Environment

- Lab VM has
  - Ubuntu / Java8 / maven3 / spark 2.1.0

- 安裝
  - $cp /opt/spark-2.1.0-bin-hadoop2.6.tgz ~
  - $tar zxvf spark-2.1.0-bin-hadoop2.6.tgz
  - $mv spark-2.1.0-bin-hadoop2.6 spark
  - Add PATH
    - PATH=$PATH:/home/spark/bin

non-Java Lambda v.s. Java 8 Lambda

```
button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent event) {
        System.out.println("button clicked");
    }
});
```

```
button.addActionListener(event ->
    System.out.println("button clicked"));
```

```
Thread thread1 = new Thread(new Runnable() {
    @Override
    public void run(){
        System.out.println("Task #1 is running");
    }
});
```

```
Runnable task2 = () -> {
    System.out.println("Task #2 is running"); };
```

```
//Sorting using Anonymous Inner class.
Collections.sort(personList, new Comparator<Person>(){
  public int compare(Person p1, Person p2){
    return p1.firstName.compareTo(p2.firstName);
  }
});
```

```
//Anonymous Inner class replaced with Lambda
expression.
Collections.sort(personList, (Person p1, Person p2) ->
p1.firstName.compareTo(p2.firstName));
```

Passing function **class** to Spark

- We will use function class in our tutorial
  - Override different call() in different Function class

```
Thread thread1 = new Thread(new Runnable() {
    @Override
    public void run(){
        System.out.println("Task #1 is running");
    }
});
```

Function name	Method to implement	Usage
`Function<T, R>`	`R call(T)`	Take in one input and return one output, for use with operations like `map()` and `filter()`.
`Function2<T1, T2, R>`	`R call(T1, T2)`	Take in two inputs and return one output, for use with operations like `aggregate()` or `fold()`.

Passing function **class** to Spark

- Using anonymous function class

```
JavaRDD<String> lines = sc.textFile("hdfs://log.txt").filter(
  new Function<String, Boolean>() {
    public Boolean call(String s) {
      return s.contains("error");
    }
});
long numErrors = lines.count();
```

- Using named class

```
class Contains implements Function<String, Boolean>() {
  private String query;
  public Contains(String query) { this.query = query; }
  public Boolean call(String x) { return x.contains(query); }
}

RDD<String> errors = lines.filter(new Contains("error"));
```

Exercise:
Build and launch standalone project

- Java project
  - With-Lambda & non-Lambda
  - Built by maven, launch by spark-submit
-

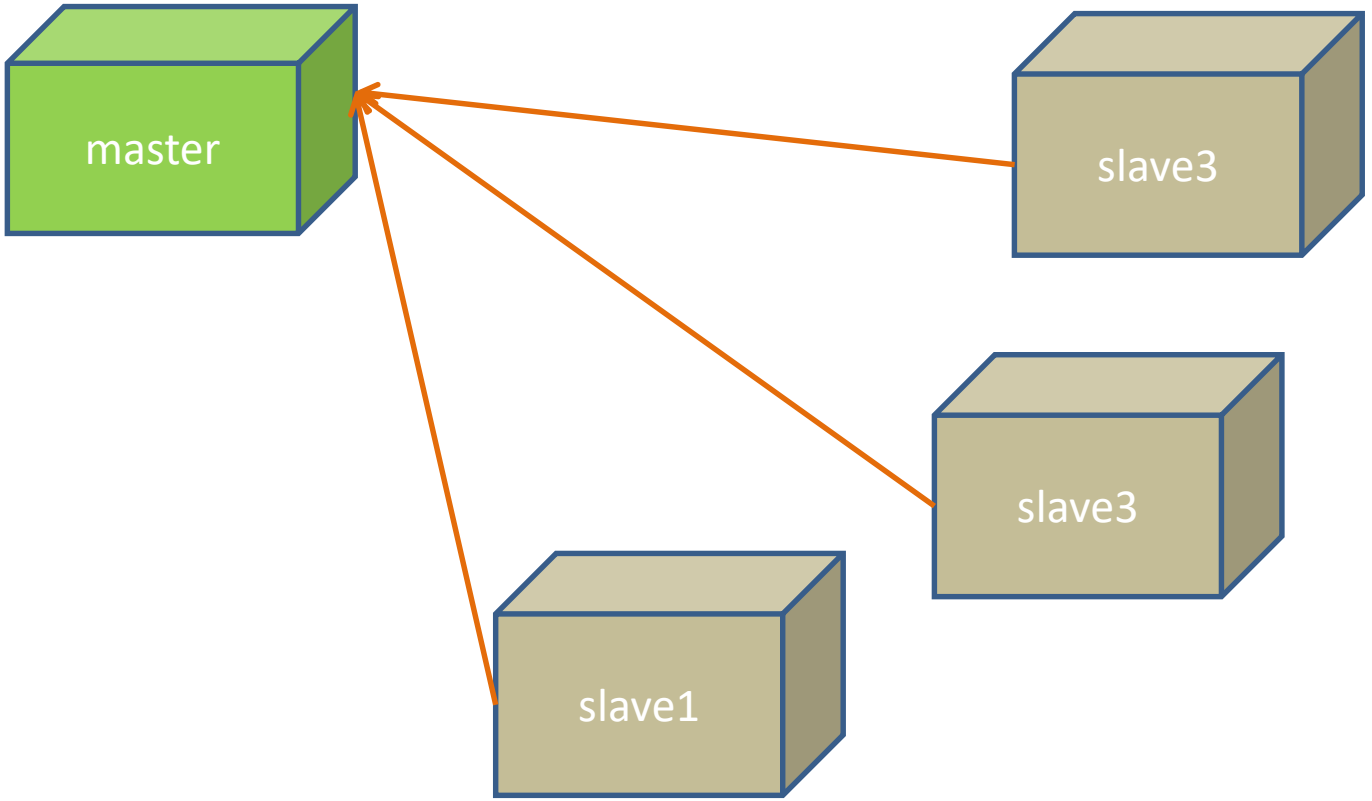
```
$ cd ~/spark-workshop
$ mvn clean package
$ cd ~/spark
$ ./bin/spark-submit \
  --class spark.HelloWorld \
  ~/spark-workshop/target/spark-sample-0.0.1.jar \
  ./README.md
```

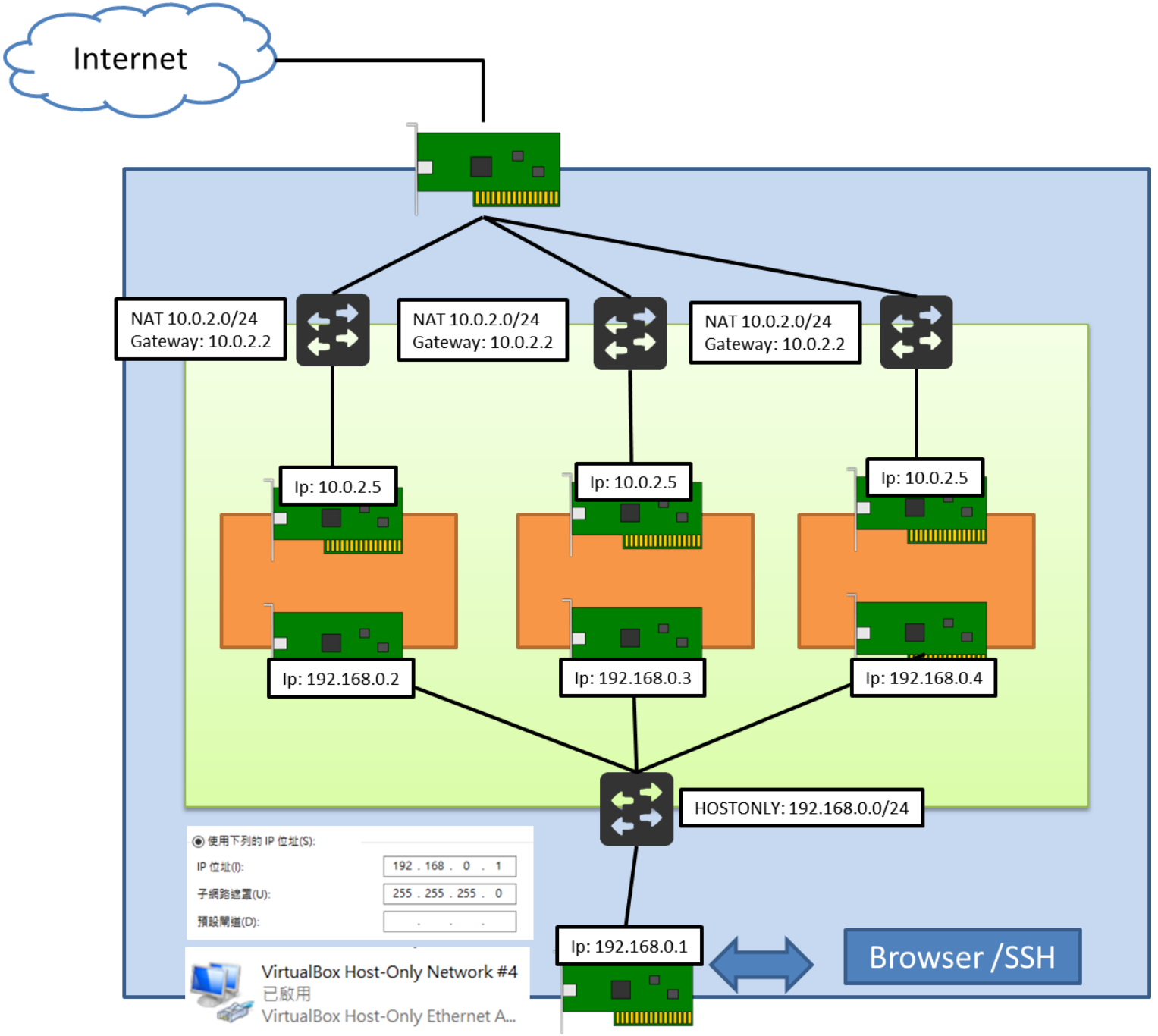
Outline

- Part I : Quick tour of Spark
  - Spark Recap
  - Prepare Lab VM
  - Run first Spark example
- Part II: Setup Cluster Lab
  - Prepare VirtualBox network
  - Prepare Cluster VMs

Distributed System

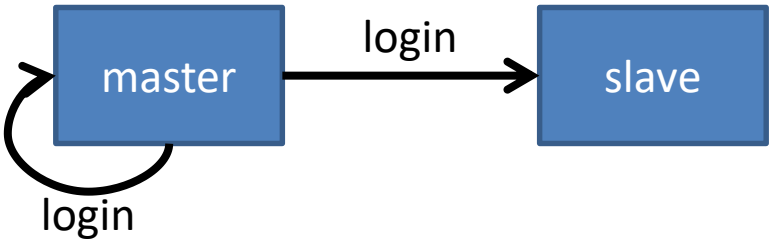
- Master /slave architecture




Internet
NAT 10.0.2.0/24
Gateway: 10.0.2.2
NAT 10.0.2.0/24
Gateway: 10.0.2.2
NAT 10.0.2.0/24
Gateway: 10.0.2.2
Ip: 10.0.2.5
Ip: 10.0.2.5
Ip: 10.0.2.5
Ip: 192.168.0.2
Ip: 192.168.0.3
Ip: 192.168.0.4
HOSTONLY: 192.168.0.0/24
使用下列的 IP 位址(S):
IP 位址(I): 192 . 168 . 0 . 1
子網路遮罩(U): 255 . 255 . 255 . 0
預設閘道(D):
VirtualBox Host-Only Network #4
已啟用
VirtualBox Host-Only Ethernet A...
Ip: 192.168.0.1
Browser /SSH

OS base installation

- 使用 hostname設定主機名稱
  - 修改 /etc/hosts



1. 在master產生 公鑰(id_rsa.pub) 與 私鑰(id_rsa)
   $ ssh-keygen –t rsa

2-法1. 將公鑰由master送到**master**與**所有slave**上並公開
   spark@spark-master $ scp id_rsa.pub spark@spark-slave:/home/spark/
   spark@spark-slave $ cat ~/id_rsa.pub >> ~/.ssh/authorized_keys
   spark@spark-slave$ chmod 600 ~/.ssh/authorized_keys

2-法2. spark@spark-master $ ssh-copy-id spark@spark-master
   spark@spark-master $ ssh-copy-id spark@spark-slave

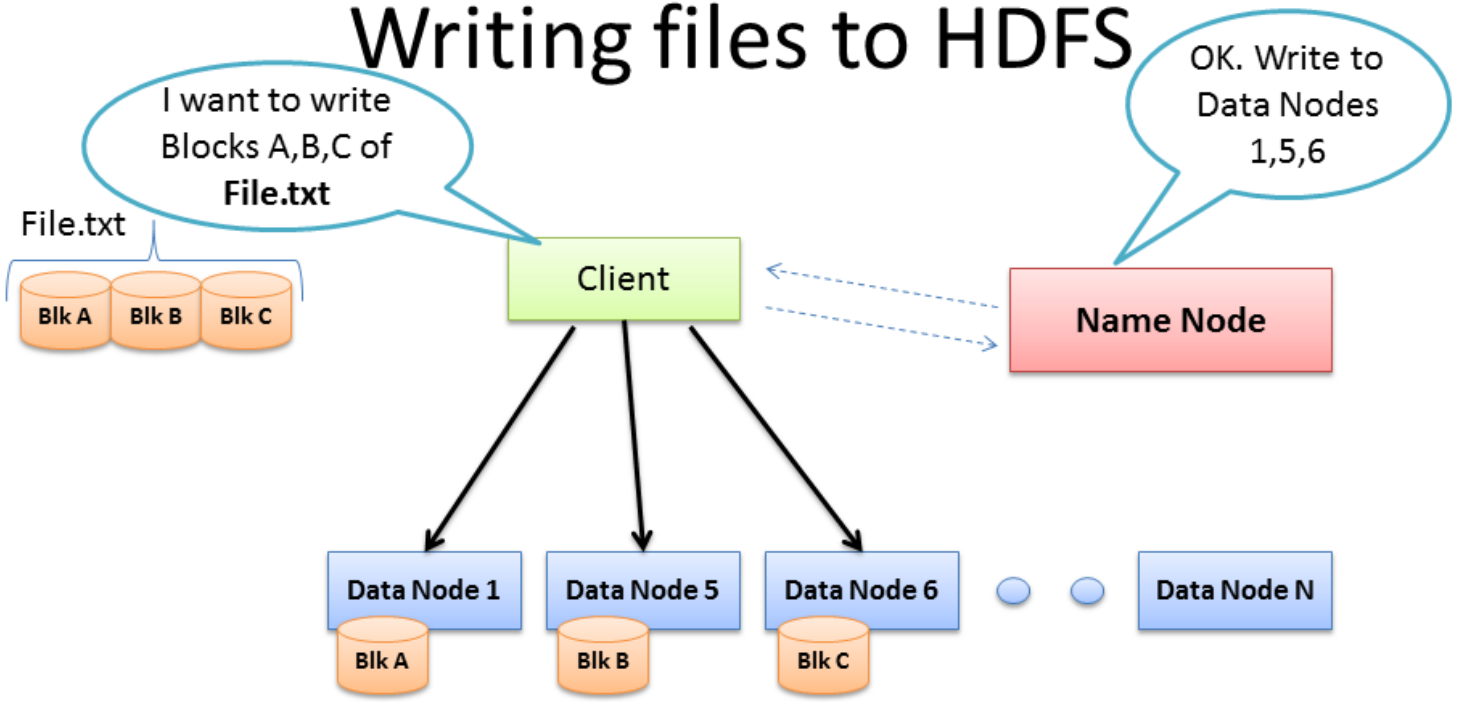
Outline

- Part III: Cluster Manager
  - Setup HDFS
  - Setup Spark Standalone Cluster
- Part IV: Run Spark on Cluster
  - Spark application
  - Components of Execution
    - Application, Job, stage, task, RDD, partition, DAG

HDFS

- Master-slave architecture
  - 1 master and MANY slaves
- Master host 上運行NameNode
  - Single point failure of NameNode
- Slave host 上運行 DataNode

Writing files to HDFS

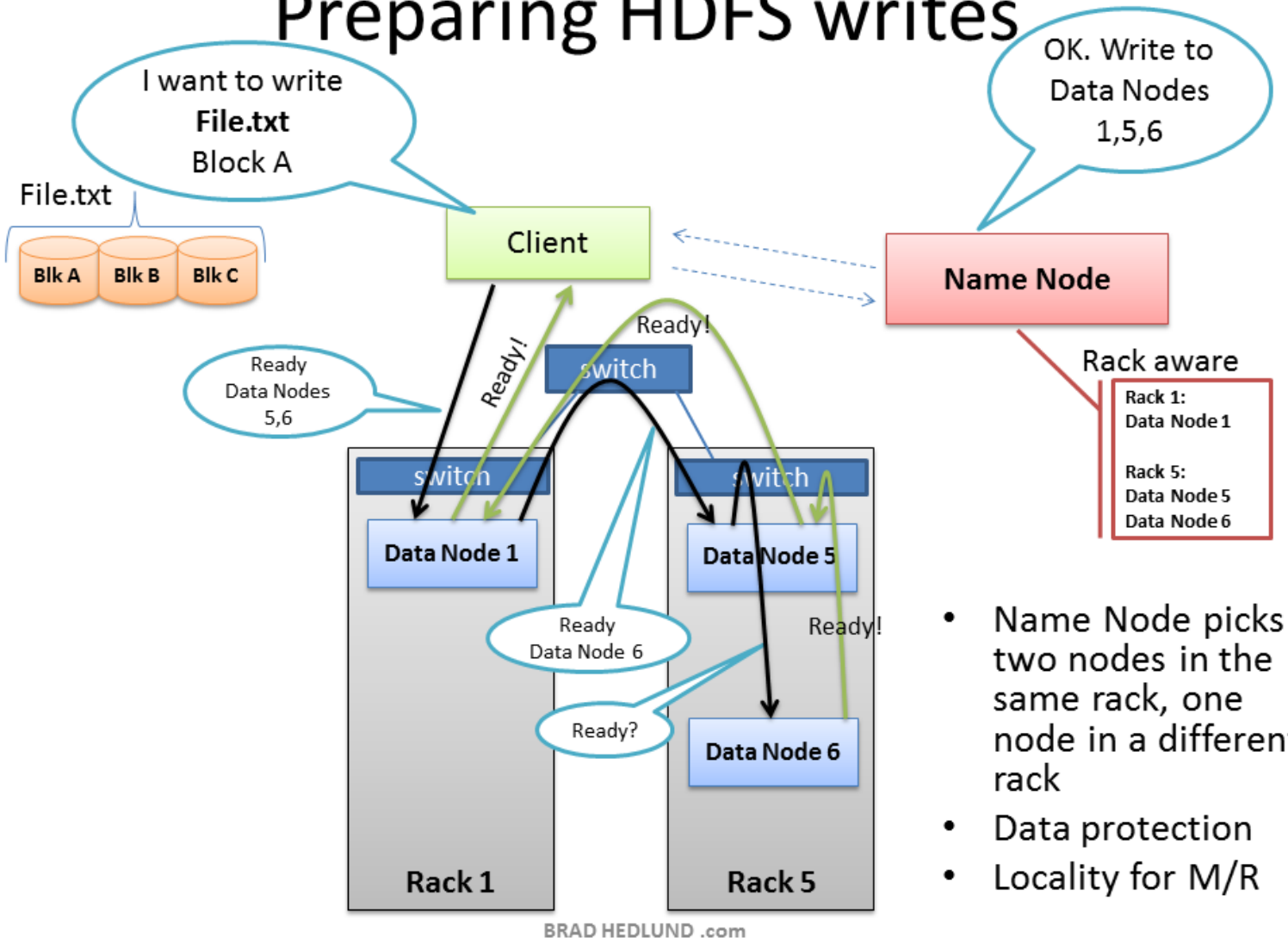


- Client consults Name Node
- Client writes block directly to one Data Node
- Data Nodes replicates block
- Cycle repeats for next block

BRAD HEDLUND .com

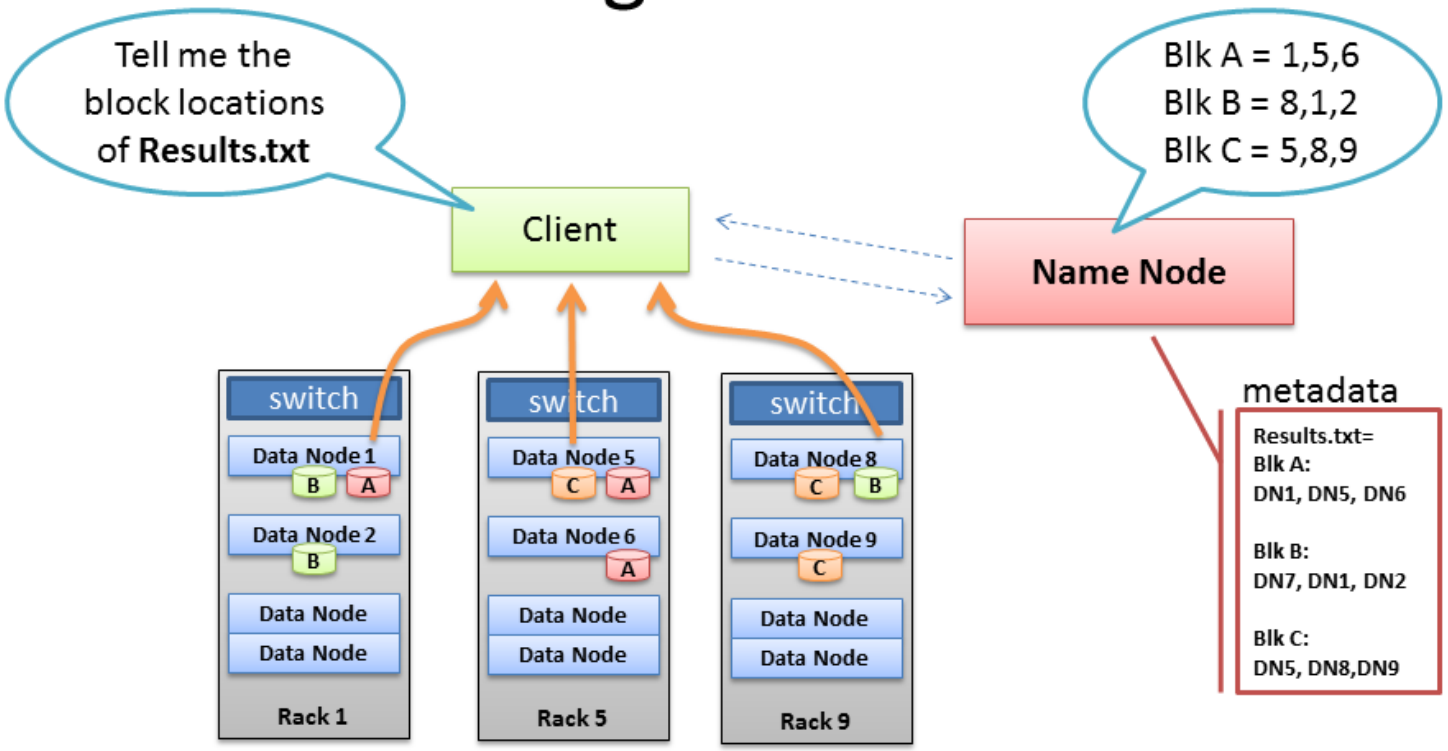
http://www.ewdna.com/2013/04/Hadoop-HDFS-Comics.html
http://bradhedlund.com/2011/09/10/understanding-hadoop-clusters-and-the-network/

Preparing HDFS writes



- Name Node picks two nodes in the same rack, one node in a different rack
- Data protection
- Locality for M/R

Client reading files from HDFS



- Client receives Data Node list for each block
- Client picks first Data Node for each block
- Client reads blocks sequentially

BRAD HEDLUND .com

http://www.ewdna.com/2013/04/Hadoop-HDFS-Comics.html
http://bradhedlund.com/2011/09/10/understanding-hadoop-clusters-and-the-network/

- 解壓hadoop-2.6.0.tar.gz
  - cp /opt/hadoop-2.6.0.tar.gz /home/spark
  - tar zxvf hadoop-2.6.0.tar.gz
  - mv hadoop-2.6.0 hadoop

- Setup HDFS
  - Edit $HADOOP_HOME/libexec/hadoop-config.sh
    - export JAVA_HOME=/usr/lib/jvm/java-8-oracle
  - Edit $HADOOP_HOME/etc/hadoop/hadoop-env.sh
    - export JAVA_HOME=/usr/lib/jvm/java-8-oracle
  - Edit $HADOOP_HOME/etc/hadoop/slaves
  - Edit $HADOOP_HOME/etc/hadoop/core-site.xml
  - Edit $HADOOP_HOME/etc/hadoop/hdfs-site.xml
  - Edit ~/.bashrc
  - 同步所有hadoop設定檔
  - Format HDFS
    - $hadoop namenode -format
  - Start / stop HDFS:
    - start-dfs.sh / stop-dfs.sh
  - Read file from HDFS:
    - sc.textFile(“hdfs://spark-master:9000/path/to/file”)

core-site.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
 <property>
  <name>fs.defaultFS</name>
  <value>hdfs://spark-master:9000</value>
 </property>
 <property>
  <name>hadoop.tmp.dir</name>
  <value>/home/spark/hadoop_dir</value>
 </property>
</configuration>
```

hdfs-site.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
 <property>
  <name>dfs.replication</name>
  <value>1</value>
 </property>
 <property>
  <name>dfs.permissions</name>
  <value>false</value>
 </property>
</configuration>
```

.bashrc

```
export HADOOP_HOME=/home/spark/hadoop
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop


export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin
```

HDFS CLI

- 格式化NameNode,
  - $hadoop namenode –format
  - 破壞性指令，只需執行一次
- 啟動與關閉HDFS
  - $start-dfs.sh
  - $stop-dfs.sh
- 確認namenode與datanode皆啟動
  - jps
  - http://spark-master:50070/
  - /home/spark/hadoop/logs

HDFS CLI

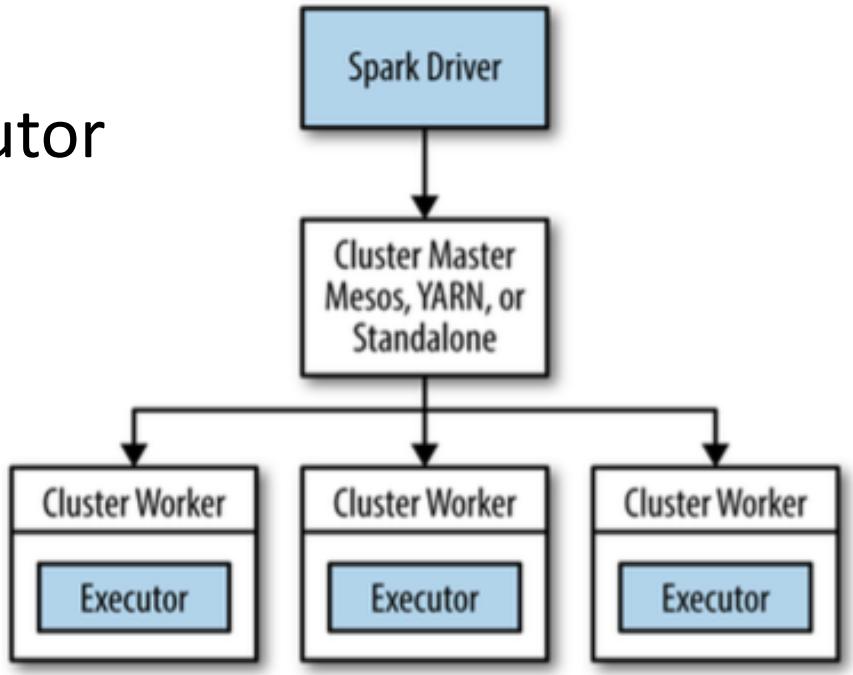
- 基本指令
  - hadoop fs –ls <file_in_hdfs>
  - hadoop fs –lsr <dir_in_hdfs>
  - hadoop fs –get <file_in_hdfs> <file_in_local>
  - hadoop fs –put <file_in_local> <file_in_hdfs>
  - hadoop fs –rm <file_in_hdfs>
  - hadoop fs –rmr <dir_in_hdfs>
  - hadoop fs -mkdir <dir_in_hdfs>
  - hadoop fs –chmod XXX <file_in_hdfs>
  - hadoop fs –chown XXX <file_in_hdfs>
  - hadoop fs –chgrp XXX <file_in_hdfs>

HDFS namespace

- HDFS default absolute URI
  - hadoop fs –ls /abc.txt
  - 等同 hadoop fs –ls **hdfs://spark-master:9000** /abc.txt
- HDFS default relative URI
  - Hadoop fs –ls abc.txt
  - 等同於hadoop fs –ls hdfs://master:9000/**user/hadoop**/abc.txt
  - hadoop為目前在Linux的使用者帳號
- Quiz1: 如何存取其他HDFS cluster ??
- Quiz2: hadoop如何知道default URI??

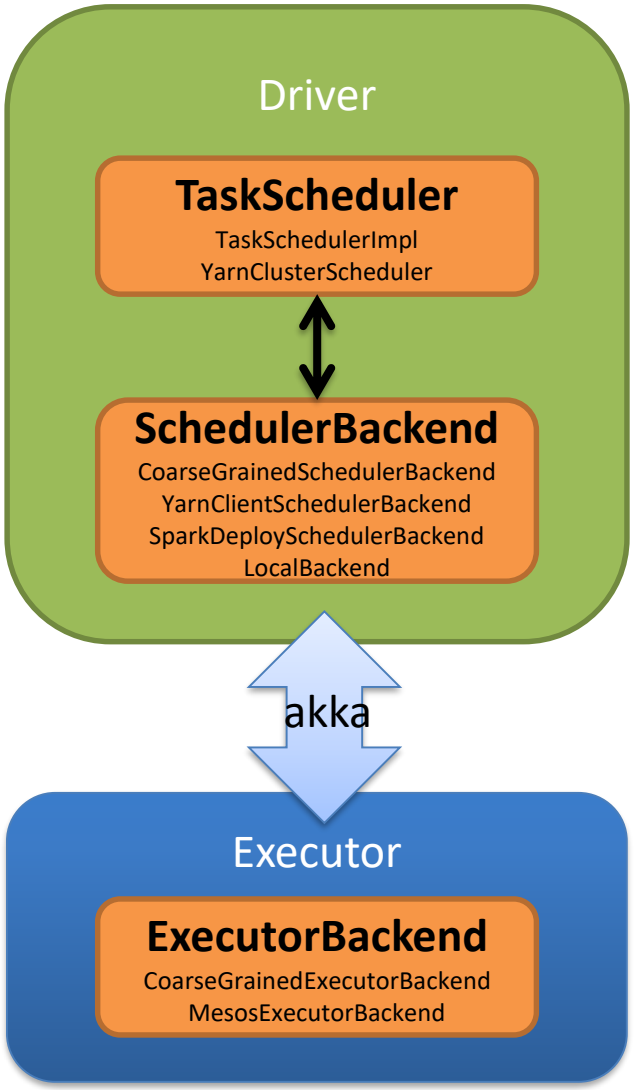
Spark Runtime Architecture

- Driver
  - Process which has the main() method
  - Convert application to tasks
    - DAGScheduler
  - Scheduling tasks on executor
    - TaskScheduler
    - SchedulerBackend
- Executor
  - Running individual task



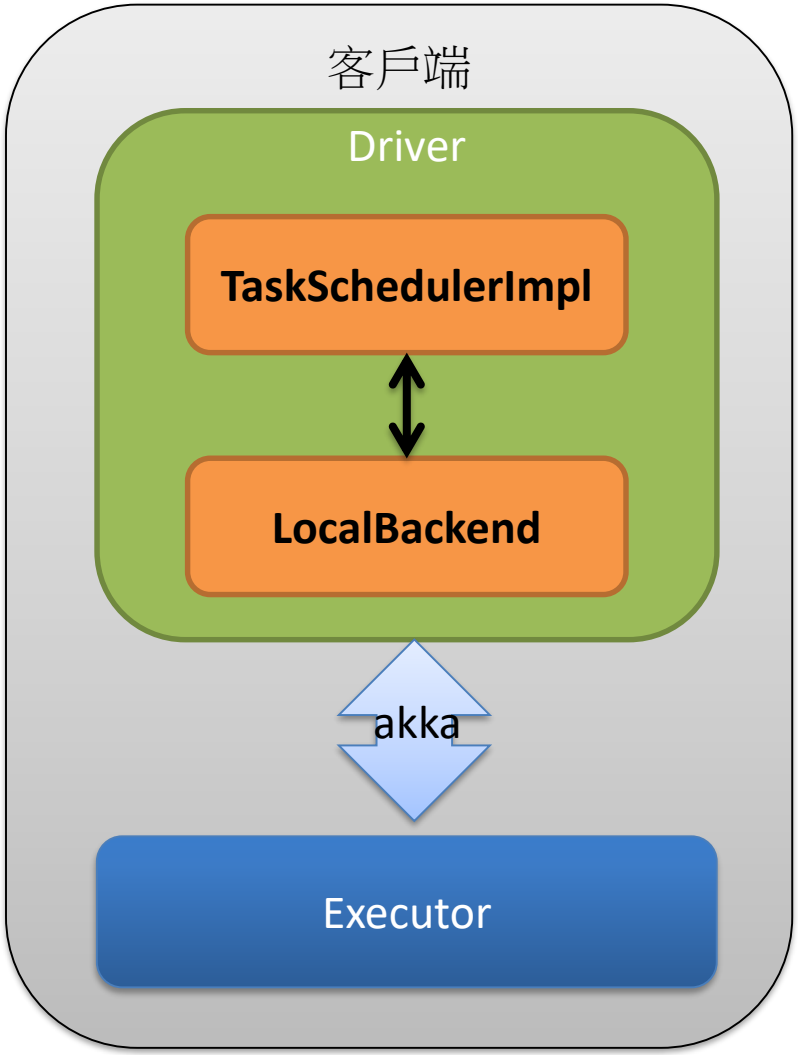
Spark Deployment

- Spark on Local
- Spark on Cluster



Local mode

- local
  - local mode with 1 core
- local[N]
  - local mode with N core
- local[*]
  - Local mode with as many cores the node has

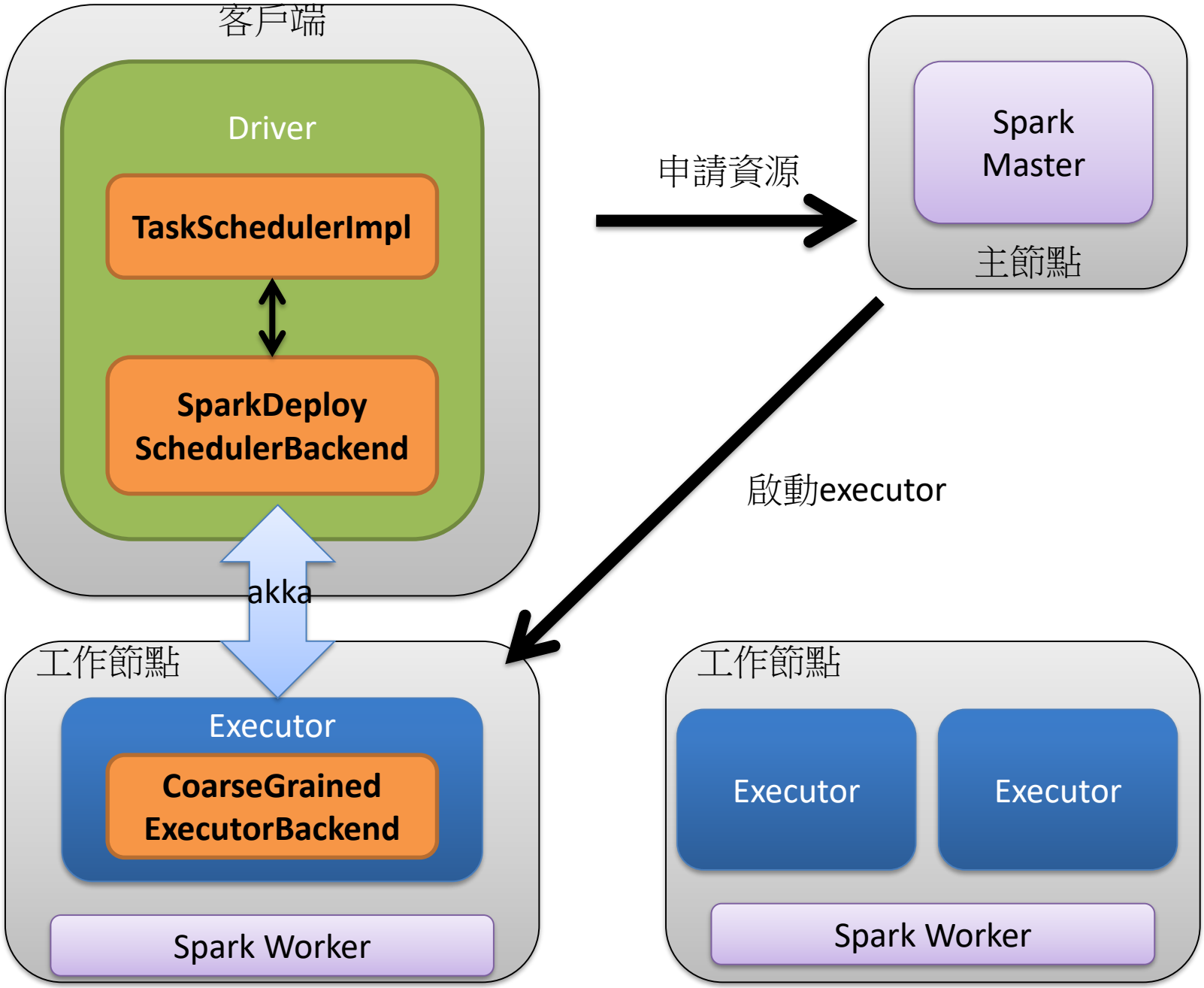


Exercise:
use different mode

- $ ./spark-shell --master local
  - **scala>** sc.master
  - **scala>** sc.isLocal

- $./spark-submit --master local[2] --class xxx.yyy.zzz xyz.jar
  - **DON'T** hard code setMaster() in source

Standalone mode



Exercise:
Setup Standalone cluster

- ~~Step 1: edit /etc/hosts~~
- ~~Step 2: Password less login~~
  - ~~spark@master $ ssh-keygen –t rsa~~
  - ~~spark@master $ ssh-copy-id spark@master~~
  - ~~spark@master $ ssh-copy-id spark@slave~~
- Step 3:
  - Edit $SPARK_HOME/conf/slaves
  - Edit $SPARK_HOME/conf/spark-env.sh
    - SPARK_WORKER_MEMORY=512m
- Step 4: distribute $SPARK_HOME to all nodes
  - $scp –r sprak spark@slave:/home/spark

- Step 5: Start / Stop Spark Cluster
  - $SPARK_HOME/sbin/start-all.sh
  - $SPARK_HOME/sbin/stop-all.sh
- Step 6: connect to Spark master using **hostname**
  - $ spark-shell --master spark://**spark-master**:7077

  - $ spark-submit --master spark://**spark-master**:7077 \
    --executor-memory 512m --driver-memory 512m \
    --class xx.yy.zz xyz.jar

  - export MASTER=spark://spark-master:7077

Outline

- Part III: Cluster Manager
  - Setup Spark Standalone Cluster
- Part IV: Run Spark on Cluster
  - Spark application
  - Components of Execution
    - Application, Job, stage, task, RDD, partition, DAG

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

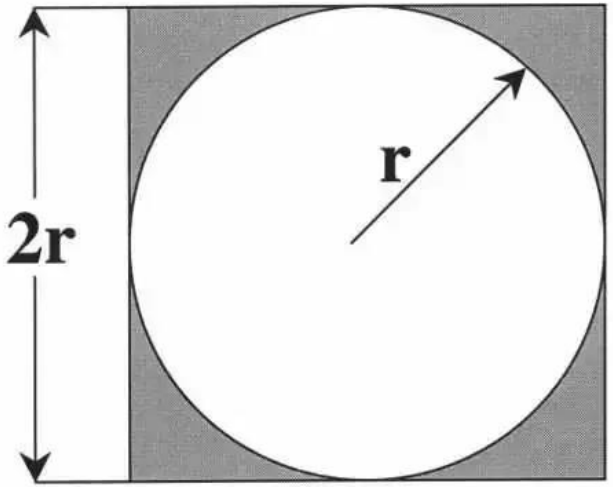
- For caching datasets in-memory for future operations.
- Option to store on disk or RAM or mixed (Storage Level).
- Example:
  - Persist()
  - Cache()

Spark application

- Set Master Mode
- Pi Estimation

Pi Estimation

- 在單位方型內隨機生成資料點。
  - mapToPair(…)
- 判斷那些點在單位圓內，那些點在單位圓外面。
  - filter(…)
- 統計在單位圓內點的個數
  - count()
- Pi = 4*圓內點總數/全部點數



$$\frac{Area\ of\ Circle}{Area\ of\ Square} = \frac{\pi r^2}{(2r)^2} = \frac{\pi}{4}$$

Exercise:
use different mode

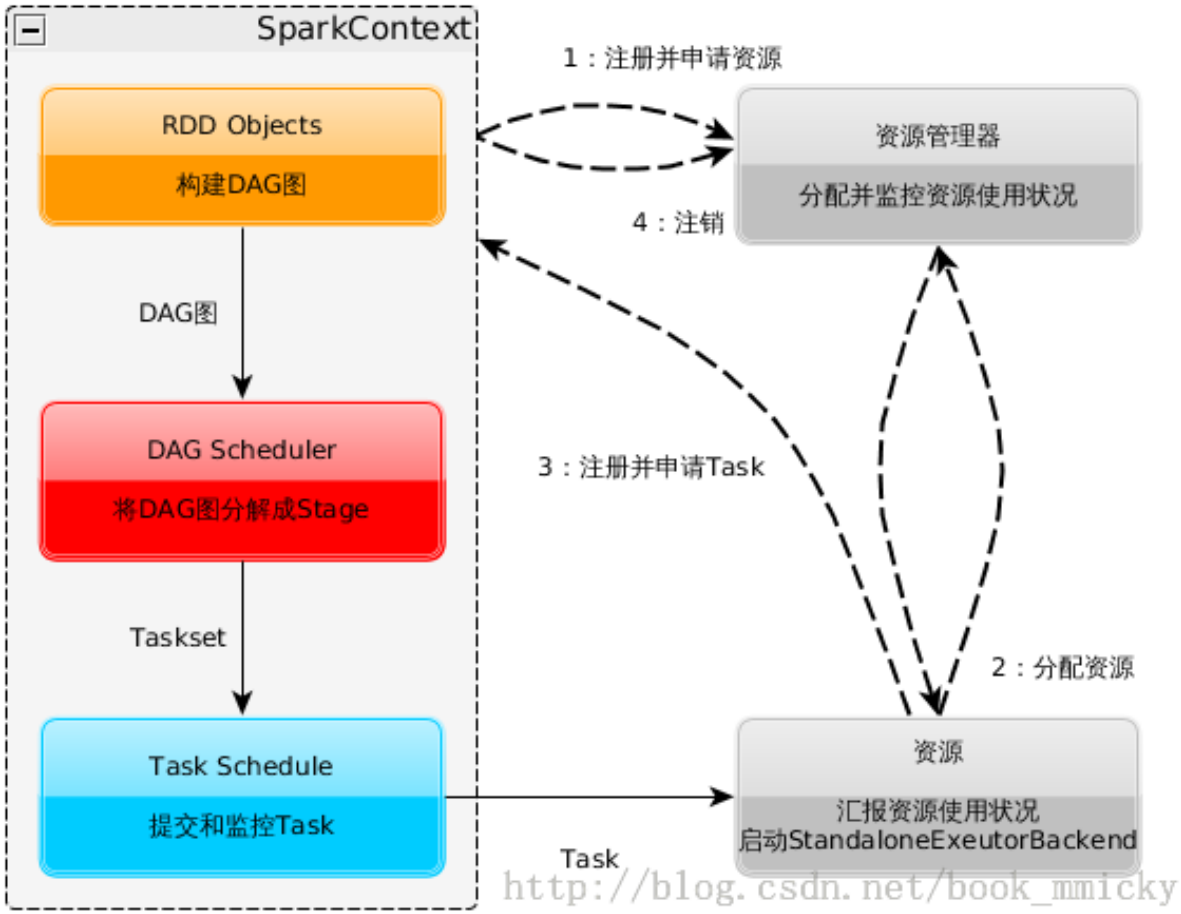
- $./spark-submit \
  --master spark://spark-master:7077 \
  --class spark.EstimatePi
  spark-sample-0.0.1.jar.jar

- **DON'T** hard code setMaster() in source

Outline

- Part III: Cluster Manager
  - Setup Spark Standalone Cluster
- Part IV: Run Spark on Cluster
  - Spark application
  - Components of Execution
    - Application, Job, stage, task, RDD, partition, DAG

Spark Runtime



http://blog.csdn.net/book_mmicky/article/details/25714419

- 建構Spark Application執行環境(啟動sc)

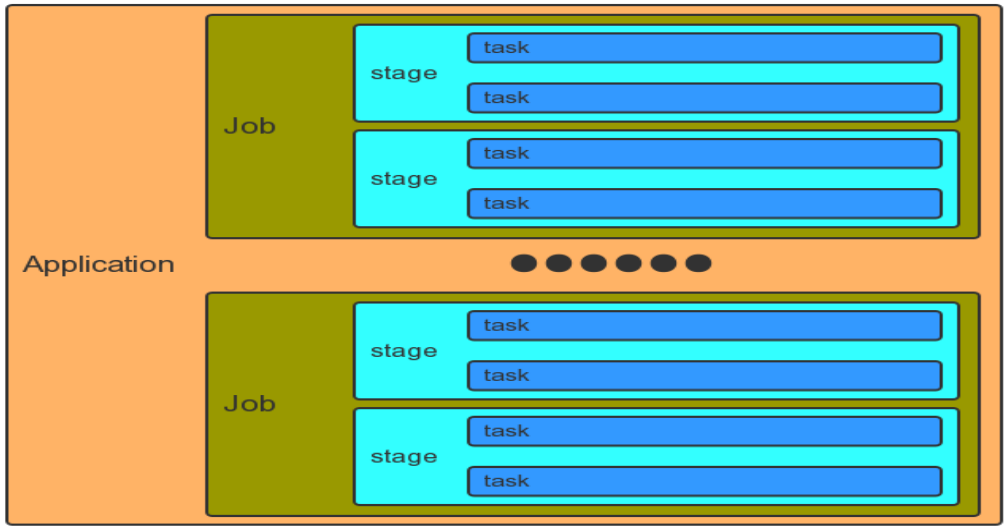
- SC透過cluster manager (Standalone、Mesos、Yarn)申請Executor資源後啟動ExecutorBackend，executor會向sc註冊。

- **SC利用DAGScheduler生成DAG圖(logic plan)，將DAG圖分解成Stage(physical plan)**，將Taskset發送給Task Scheduler，最後由最Task Scheduler將Task分發給Executor執行。

- Task在Executor上執行完成後，釋放資源。

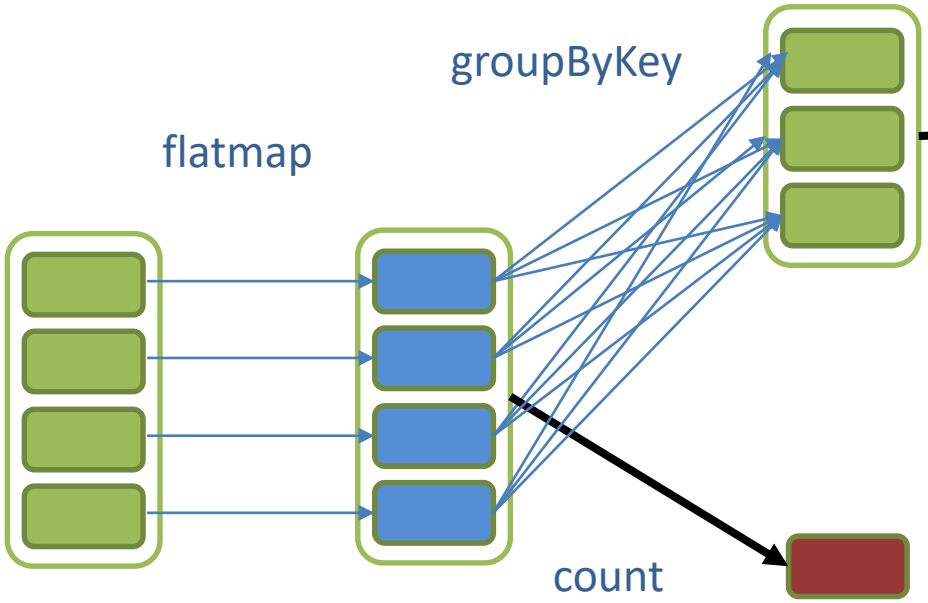
Spark Application

- 一個sparkcontext對應一個Application
- 每個applicatio可有多個job，一個action產生一個job，job可平行或依序執行
- Job由多個stage組成，job裡的stage是由shuffle畫分
- Stage裡有多個task，task數由最後一個RDD的partition決定



How to determine Job

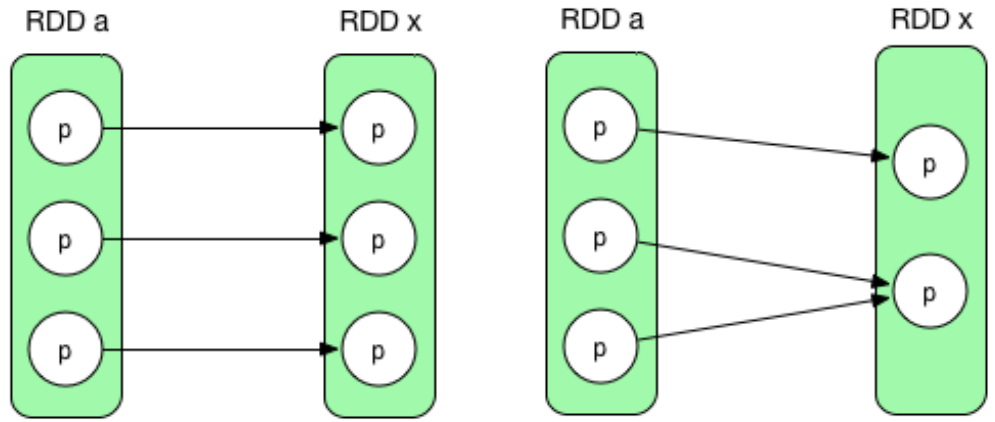
- application 可以包含多個job
- Driver 中每執行一個action，就會產生一個job

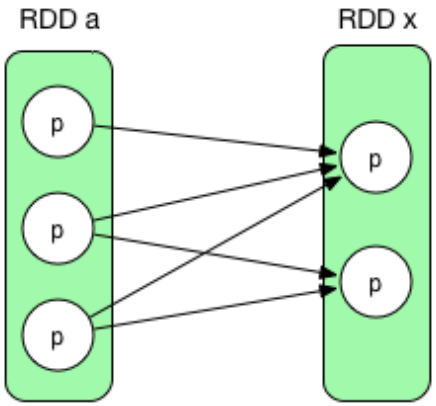



```
val pairs1 = sc.parallelize().flatMap {
            p =>  .....
}.cache

pairs1.count

pairs1.groupByKey(numReducers).count
```

Narrow Dependency

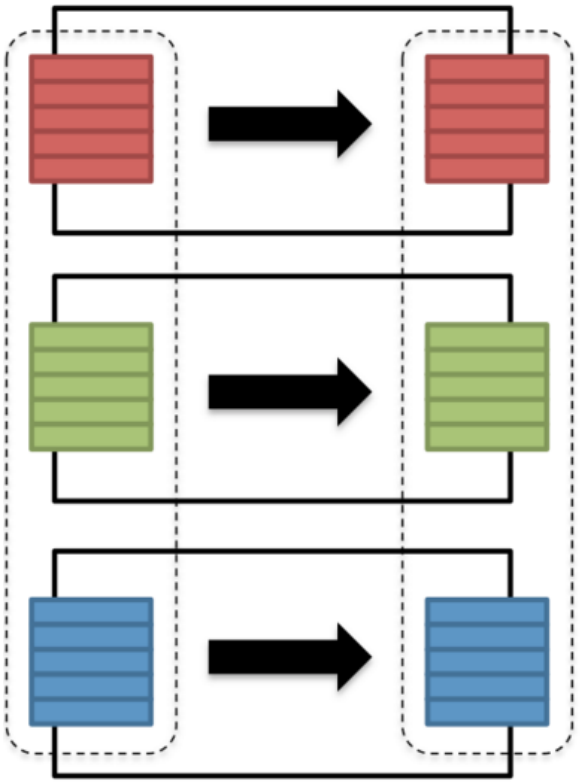
- depends only on data that is already residing in the partition and data shuffling is unnecessary
- Filter()、map()、…





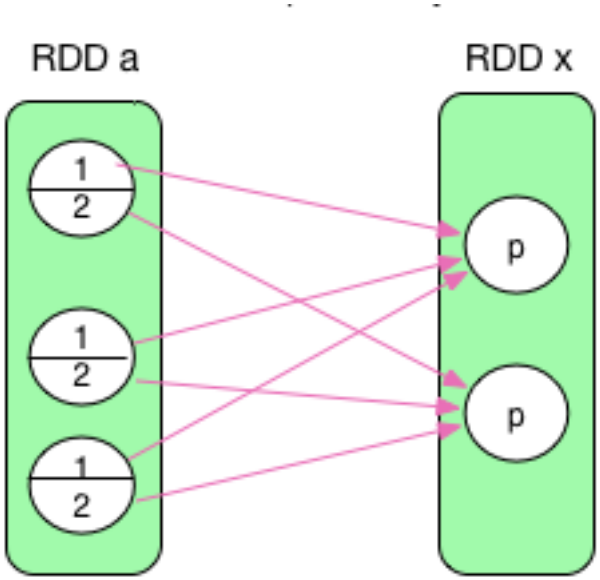
Narrow transformation

- Input and output stays in same partition
- No data movement is needed



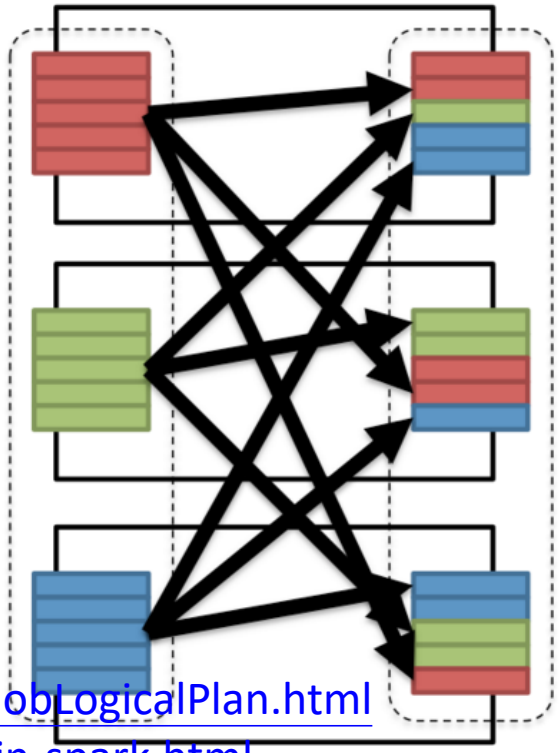
Wide Dependency

- depends on data residing in multiple partitions and therefore data shuffling is needed to bring them together in one place
- groupByKey()、reduceByKey()



Wide transformation

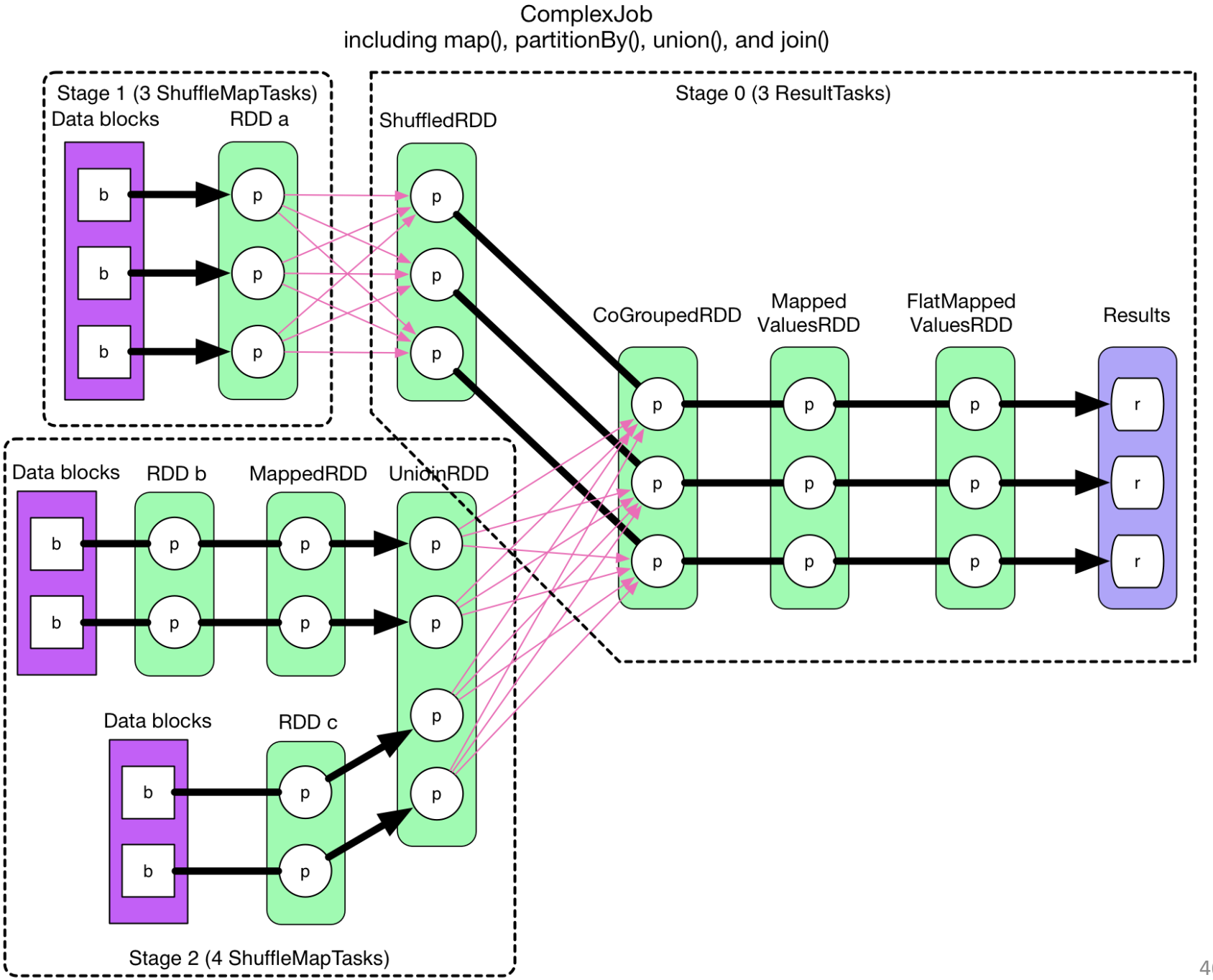
- Input from other partitions are required
- Data shuffling is needed before processing



http://spark-internals.books.yourtion.com/markdown/2-JobLogicalPlan.html
http://horicky.blogspot.tw/2015/02/big-data-processing-in-spark.html

How to determine stage and task

- Stage
  - Stages are sequences of RDDs, that don't have a **Shuffle** in between
  - 由後往前推，遇到wideDependency (or Shuffle) 就斷開，遇到NarrowDependency 就加入stage。
- Task
  - 每個Stage裡的 task數目由該 stage最後一個 RDD 中的 partition個數決定。


ComplexJob
including map(), partitionBy(), union(), and join()
Stage 1 (3 ShuffleMapTasks)
Data blocks
RDD a
ShuffledRDD
Stage 0 (3 ResultTasks)
CoGroupedRDD
Mapped ValuesRDD
FlatMapped ValuesRDD
Results
Data blocks
RDD b
MappedRDD
UnionRDD
Data blocks
RDD c
Stage 2 (4 ShuffleMapTasks)

Exercise: Setup history server

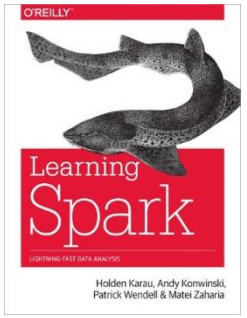
- Create local log directory
- Edit $SPARK_HOME/conf/spark-defaults.conf
  - 讓Executor 輸出event log
    - spark.eventLog.enabled true
    - spark.eventLog.dir hdfs://spark-master:9000/sparkLogs
  - 設定history server讀取目錄
    - spark.history.fs.logDirectory hdfs://spark-master:9000/sparkLogs
- Start history server
  - $SPARK_HOME/sbin/start-history-server.sh
  - http://spark-master:18080

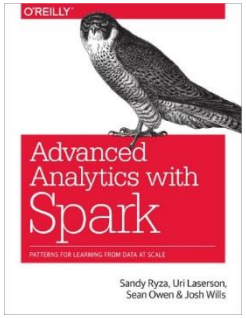
Exercise:
觀察Job、Stage、Task in UI

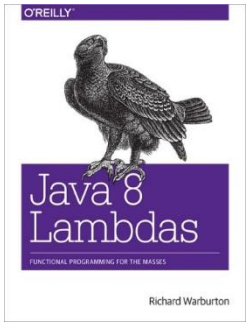
- 分別執行simpleJob與complexJob
  - --class spark.simpleJob
  - --class complexJob

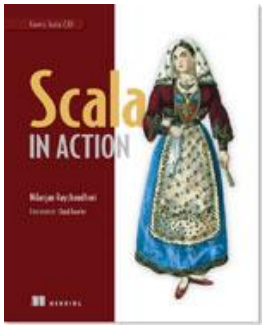
- 在job-history-server上驗證job數、stage數、task數

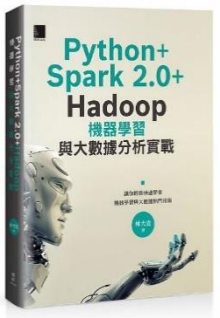
Reference











- Spark
  - Learning Spark
  - Advanced Analytics with Spark
- Java 8
  - Java 8 Lambdas
- Scala
  - Scala in action
  - 为Java程序员编写的Scala的入门教程
    - http://www.iteblog.com/archives/1325
- PySpark
  - Python+Spark 2.0+Hadoop機器學習與大數據分析實戰