

# 巨量資料實戰課程(I) – Hive Batch 與Spark Streaming 簡介

David Chiu  
2016/12/01

# 關於我



- 大數軟體有限公司創辦人
- 前趨勢科技工程師
- [ywchiu.com](http://ywchiu.com)
- 大數學堂  
<http://course.largitdata.com/>
- 粉絲頁  
<https://www.facebook.com/largitdata>
- R for Data Science Cookbook  
<https://www.packtpub.com/big-data-and-business-intelligence/r-data-science-cookbook>
- Machine Learning With R Cookbook  
<https://www.packtpub.com/big-data-and-business-intelligence/machine-learning-r-cookbook>



# 課程資料

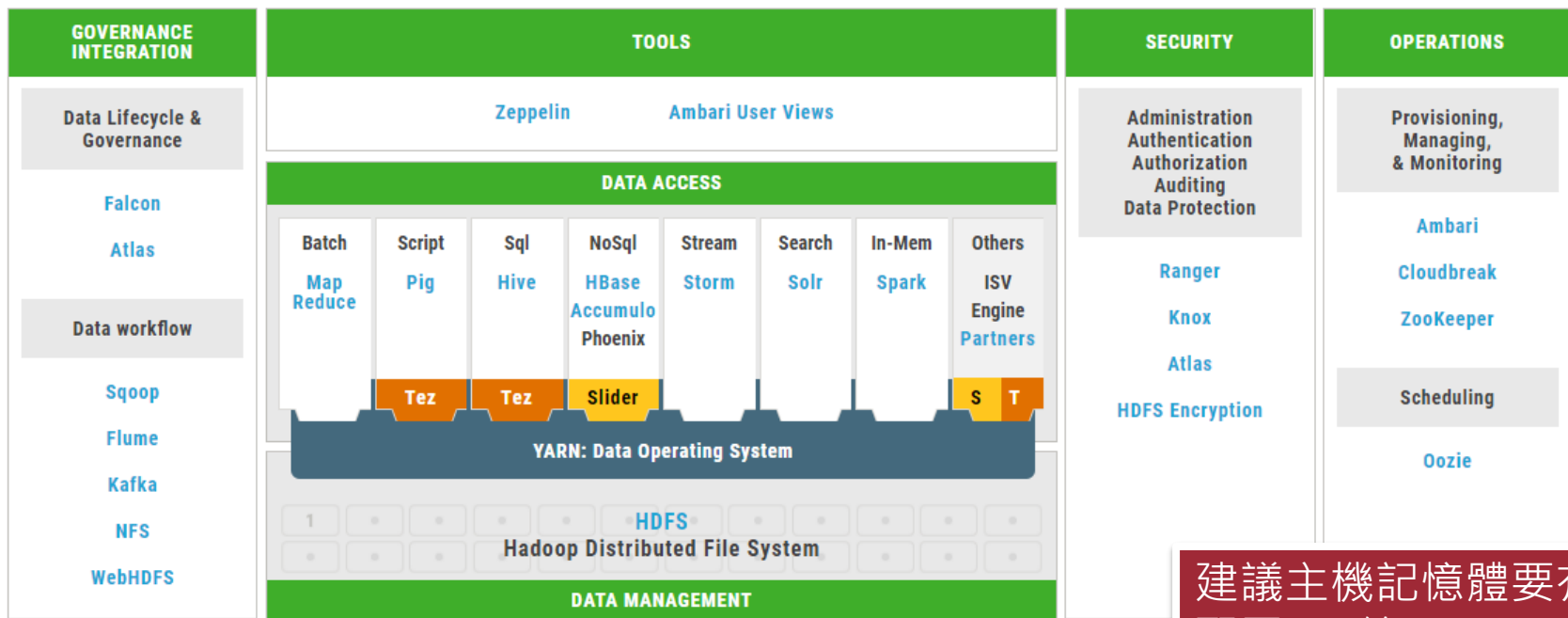
- 所有課程補充資料、投影片皆位於
  - <https://github.com/ywchiu/micronbigdata>

# 操作環境簡介

# (最新版)HDP 2.4

## ■ Hortonworks Sandbox

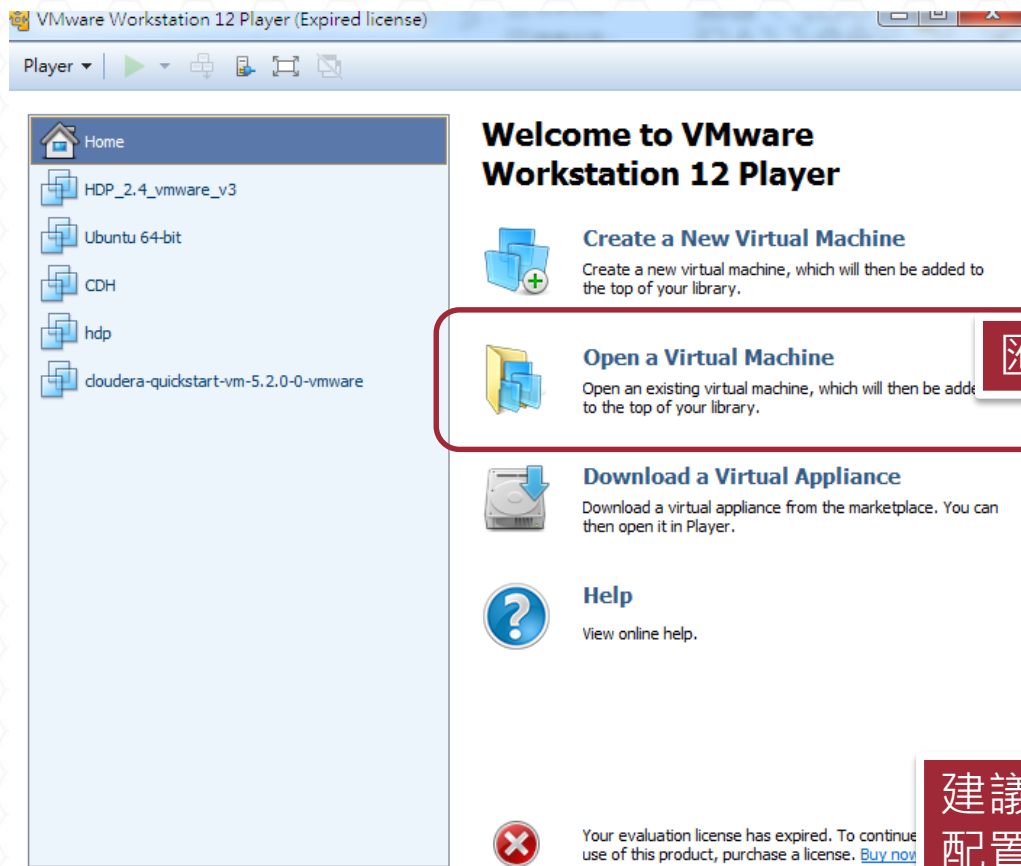
<http://hortonworks.com/downloads/>



建議主機記憶體要有16G  
配置8G 給VM

# 使用vmware player 打開 sandbox (ova)

- 可前往 <https://www.vmware.com/tw/products/player> 下載 vmware player



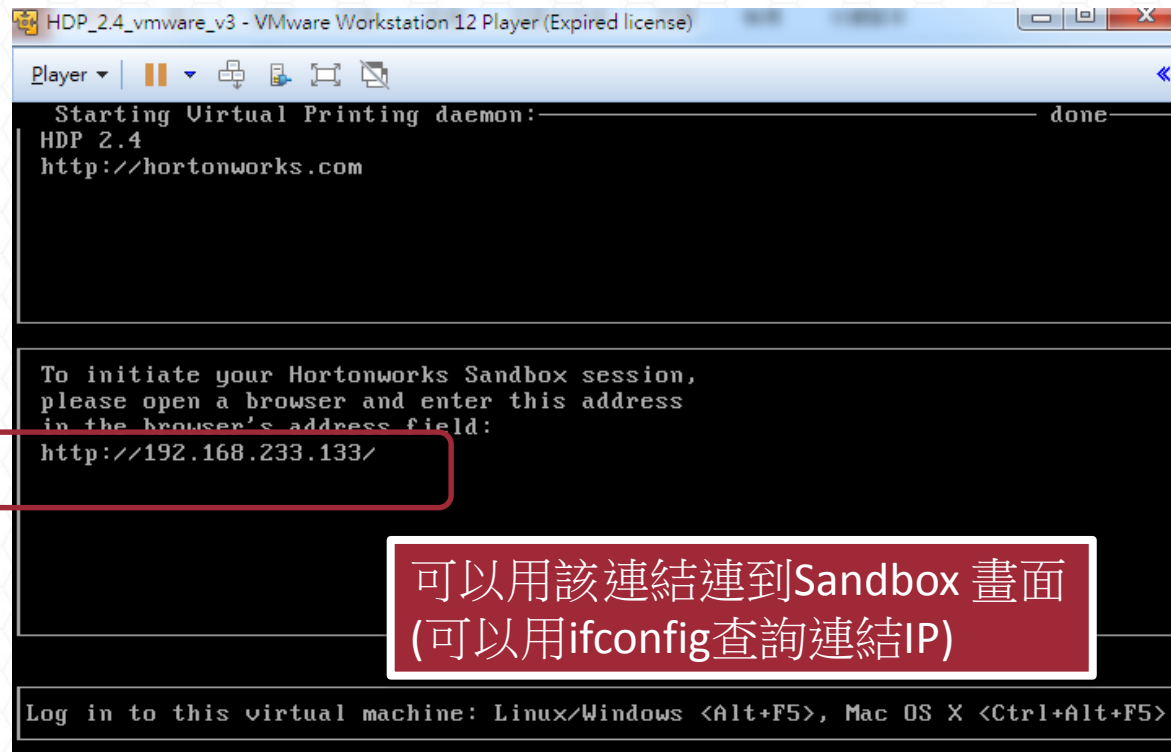
匯入下載的ova 檔

建議主機記憶體要有16G  
配置8G 給VM



# 進到sandbox 畫面

- 第一次進來root需要修改密碼，可修改成 **Hado0p0610**
- 之後可以用 `passwd` 修改回原密碼



```
HDP_2.4_vmware_v3 - VMware Workstation 12 Player (Expired license)
Player | [Icons]
Starting Virtual Printing daemon: done
HDP 2.4
http://hortonworks.com

To initiate your Hortonworks Sandbox session,
please open a browser and enter this address
in the browser's address field:
http://192.168.233.133/

Log in to this virtual machine: Linux/Windows <Alt+F5>, Mac OS X <Ctrl+Alt+F5>
```

# 連結到Sandbox 畫面

## ■ 打開瀏覽器連結到Sandbox

← → ↻ 192.168.233.133

# Hortonworks Sandbox

With **HDP 2.4** in a few simple steps

1 get started 2 try 3 what's new

Develop queries for data & manage your HDP cluster

Apache Ambari is the best way to get started with your HDP journey. It provides a user driven wizard interface to interact with HDP. Try this simple tutorial to get started with HDP. [Hello HDP!](#)

url: <http://192.168.233.133:8080>  
username: maria\_dev  
password: maria\_dev

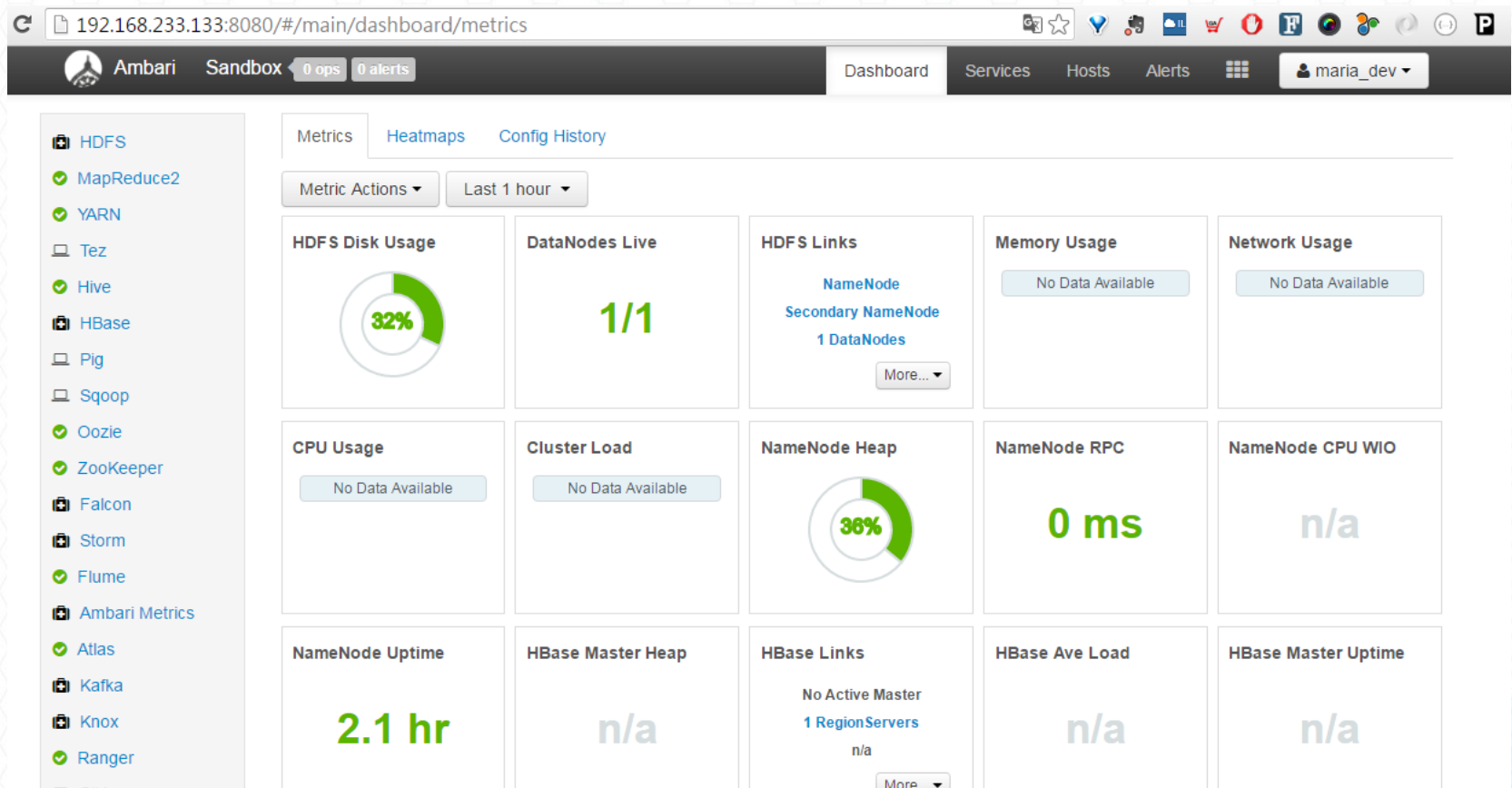
可以用該連結連到Ambari

Close Advanced Options



# 進到Ambari

■ 連結192.168.233.133:8080





# **Batch Processing v.s. Streaming**

# Batch v.s. Straming

	Batch Processing	Stream Processing
Data scope	Queries or processing over all or most of the data	Queries or processing over data on rolling window or most recent data record
Data size	Large batches of data	Individual records or micro batches of few records
Performance	Latencies in minutes to hours.	<b>Requires latency in the order of seconds or milliseconds.</b>
Analytics	Complex analytics.	Simple response functions, aggregates, and rolling metrics.

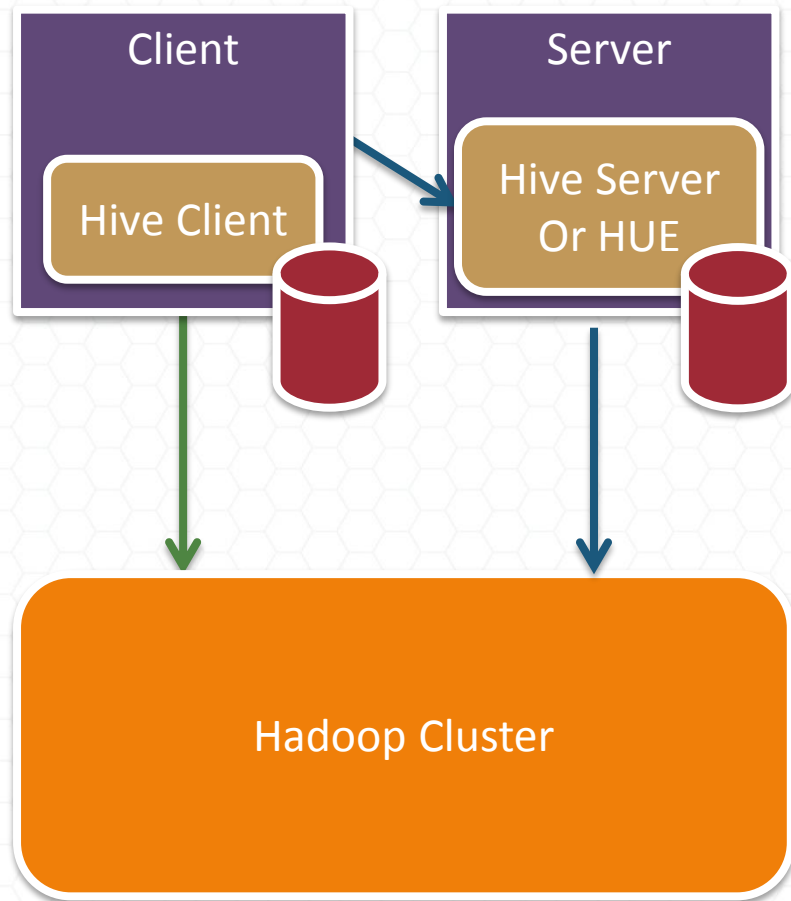




# **Batch Data Processing With Hive**

# Hive 架構

- Facebook 所開發
- SQL Like 語言
- Hive 可以以Client 的形式安裝
- 或是可以使用網路服務存取
  - Hue (beeswax, web interface)
  - HiveServer2 (ODBC, JDBC)
- 需要使用metastore
  - 提供Hive 對HDFS 資料的映對



# Hive 的優點

- 簡潔方便 (相較於Java 而言)
- Ad-Hoc 分析 (比Impala 慢)
- 可以依時間分割資料 (Partition)
- DBA 可以重複使用部分Query (同MySQL 語法)
- 使用共用的View節省表格建置的時間成本
  - Hive 專注在資料表的建立與寫入
  - 共用的View 可以加速分析



# Hive 的缺點

- 無法即時性分析(使用Impala)
- 非高效能 (使用Impala)
- 如需要使用資料ETL (使用PIG)
- 如需要精細控制流程(IF ...ELSE) (使用 PIG)
- 難以處理非結構化(沒有明確的Schema)資料 (使用 PIG)

# HiveQL

- Hive 是建立於Hadoop 之上的資料倉儲套件
- 提供SQL Like 的語法 Hive Query Language (HiveQL)，使用者可以使用HiveQL 以MapReduce存取HDFS內的資料
- 語法接近MySQL

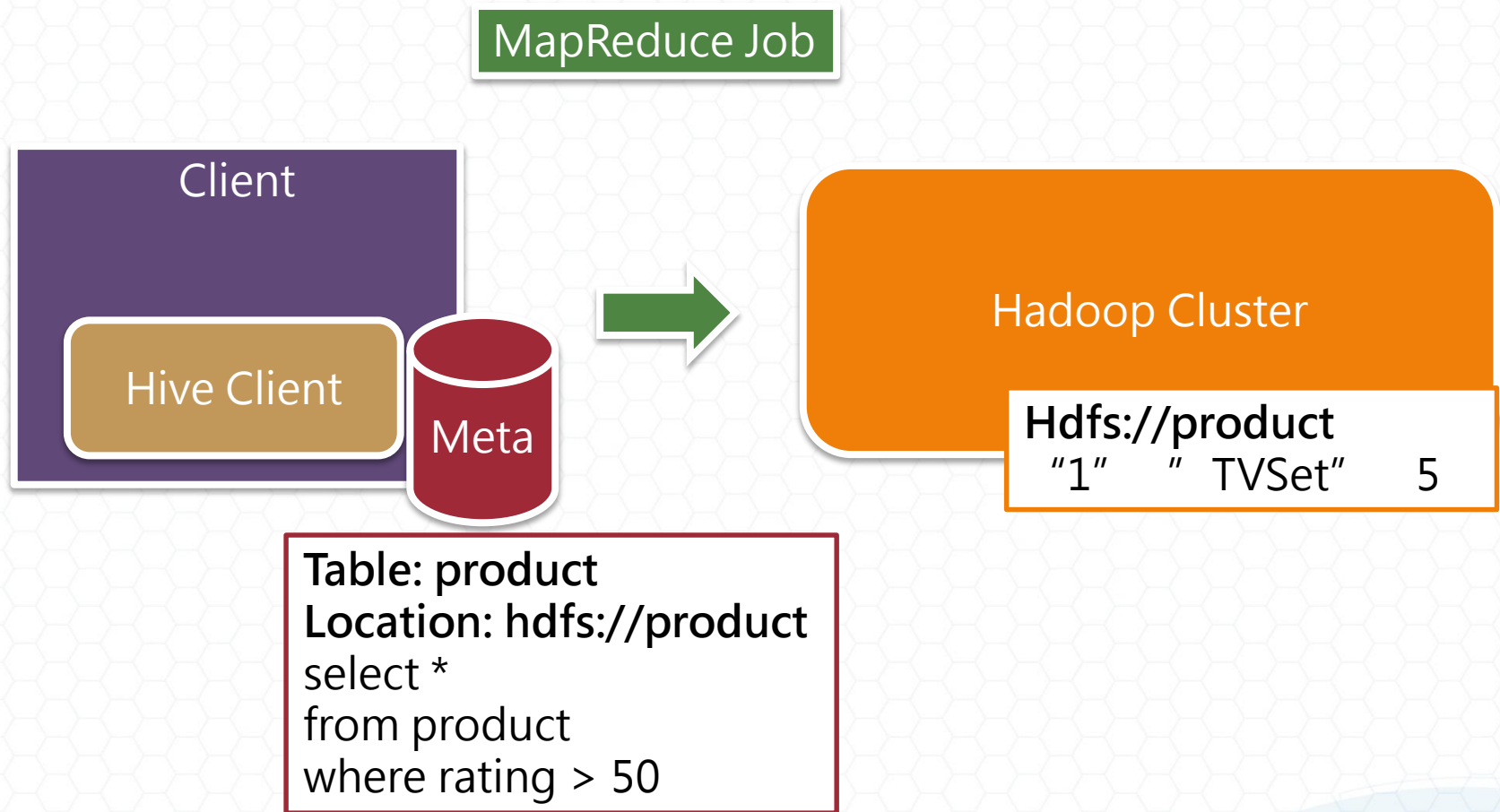
# 與傳統資料庫比較

- Hive 不提供
  - row-level insert
  - updates
  - deletes
  - Transactions
- 並非資料庫
- Hive 並非用作線上交易
- Hive 並不提供即時查詢 (可使用Impala 替代)



# Hive 與 MapReduce

# Metastore



# Select 與 Where 只使用到Mapper

```
SELECT SYMBOL, OPEN, CLOSE, TIMESTAMP  
WHERE SYMBOL = '3MINDIA'  
AND SERIES= "EQ"
```





# GROUP BY & HAVING 會在Reduce 階段完成

```
SELECT SYMBOL, MAX(OPEN)
WHERE SERIES = 'EQ'
GROUP BY SYMBOL
HAVING MAX(OPEN) > 20
```



# JOIN 會牽涉到 MAP 及 REDUCE

Item	Category
I1	C1
I2	C2

Item	Rating
I1	5
I2	4



{Key:I1,  
Value: {Tag:s1~ ,  
Category:C1}}

{Key:I1)  
{Value: {Tag:s2~ ,  
,Rating:5}}



(Key:I1)  
{Value: [{Tag:s1~ ,  
,Category:C1},{Tag:s2~ ,  
Rating: 5}]}



Item	Category	Rating
I1	C1	5
I2	C2	4

# Partition



# Partition

- Hive 可以使用分區(partitions) 加速查詢
  - Partition 是以目錄的方式存在於主表格中
  - 例如：使用日期做分割(table/2014-10, table/2014-11)
  - 每次只會存取符合查詢條件的分割, 可加速資料查詢

# 可根據任意欄位分割資料

LET'S PARTITION IT ON THE PRODUCT COLUMN

StoreLocation	Product	Date	Revenue
Bellandur	Bananas	January 18,2016	8,236.33
Bellandur	Nutella	January 18,2016	7,455.67
Bellandur	Peanut Butter	January 18,2016	5,316.89
Bellandur	Milk	January 18,2016	2,433.76
Koramangala	Bananas	January 18,2016	9,456.01
Koramangala	Nutella	January 18,2016	3,644.33
Koramangala	Peanut Butter	January 18,2016	8,988.64
Koramangala	Milk	January 18,2016	1,621.58

4 PARTITIONS OF THIS TABLE

**PRODUCT = BANANAS**

StoreLocation	Date	Revenue
Bellandur	January 18,2016	8,236.33
Koramangala	January 18,2016	9,456.01

**PRODUCT = PEANUT BUTTER**

StoreLocation	Date	Revenue
Bellandur	January 18,2016	5,316.89
Koramangala	January 18,2016	8,988.64

**PRODUCT = NUTELLA**

StoreLocation	Date	Revenue
Bellandur	January 18,2016	7455.67
Koramangala	January 18,2016	3644.33

**PRODUCT = MILK**

StoreLocation	Date	Revenue
Bellandur	January 18,2016	2,433.76
Koramangala	January 18,2016	1,621.58

# 根據商品做資料分割

partitioned by(column name column\_data\_type)

```
CREATE TABLE Sales_Data_Product_Partition  
(  
  StoreLocation VARCHAR(30),  
  OrderDate DATE,  
  Revenue DECIMAL(10,2)  
)  
partitioned by(product varchar(30));
```

指定切割欄位

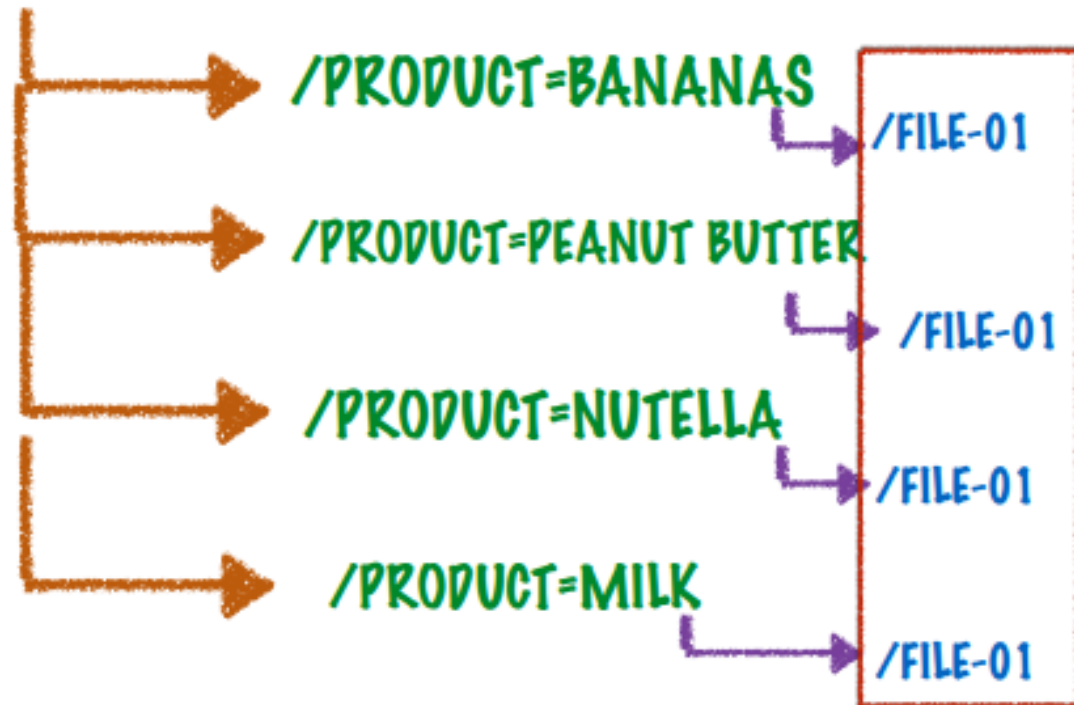


# 資料會被分別擺放在不同目錄下

**/USER/HIVE/WAREHOUSE**

└─ **/SALES-TABLE**

**DATA IS STORED IN  
THESE FILES**



# 根據時間做切割

partitioned by(column name column\_data\_type)

```
CREATE TABLE Sales_Data_Product_Partition  
(  
  StoreLocation VARCHAR(30),  
  OrderDate DATE,  
  Revenue DECIMAL(10,2)  
)  
partitioned by(OrderDate DATE);
```

指定切割欄位

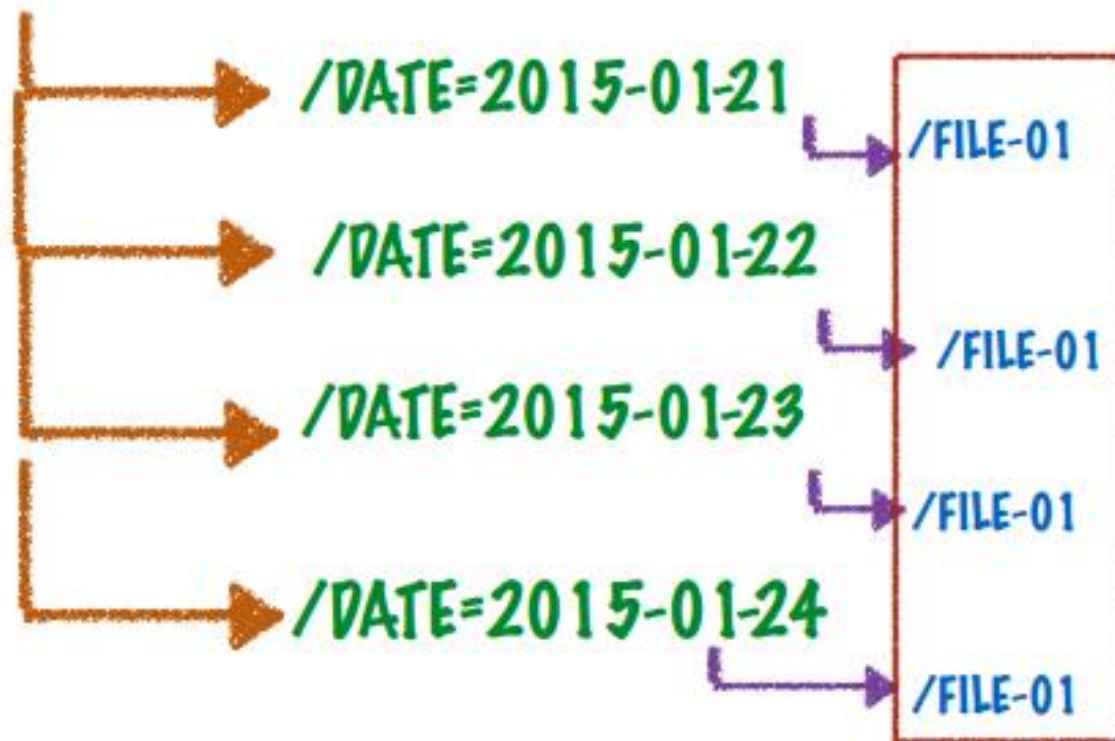
資料會根據時間被分別擺放在不同目錄下

**/USER/HIVE/WAREHOUSE**



**/SALES-DATA-DATE-PARTITION**

**DATA IS STORED IN  
THESE FILES**





# 同時根據兩個欄位切割

```
CREATE TABLE Sales_Data_Date_Product_Partition  
(  
  StoreLocation VARCHAR(30),  
  Revenue DECIMAL(10,2)  
)  
partitioned by  
(  
  OrderDate DATE,  
  product VarChar(30)  
);
```

指定切割欄位

# 同時根據時間跟產品做切割

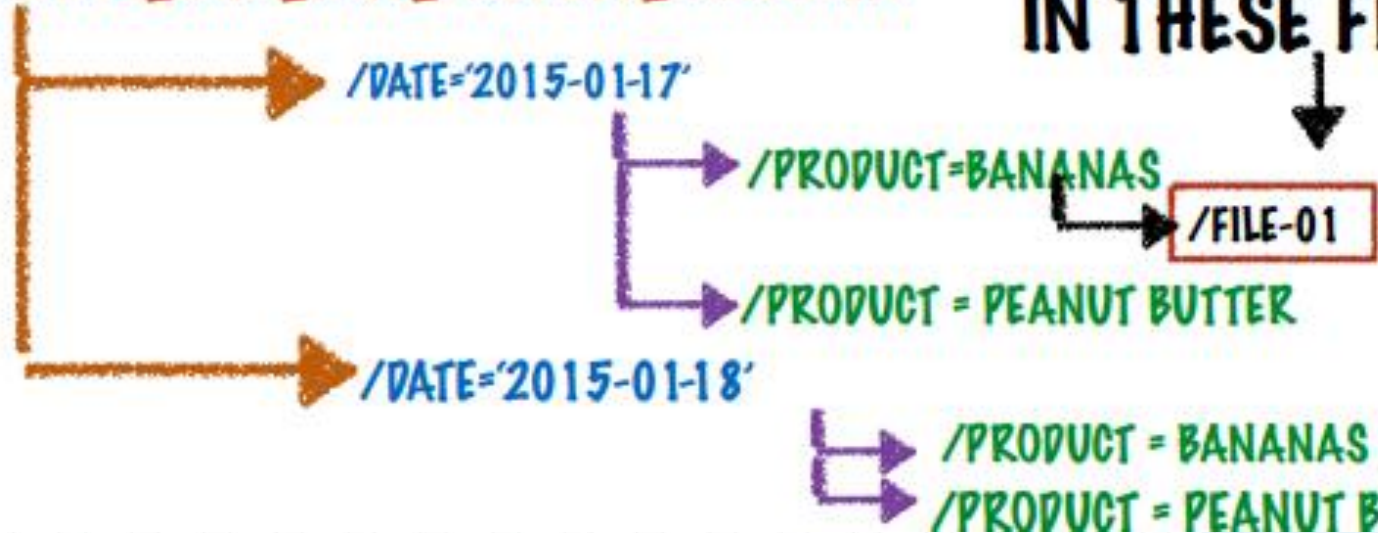
**WE CAN PARTITION TABLES BY TWO COLUMNS**

**LET'S LOOK AT THE DIRECTORY STRUCTURE FOR SUCH A TABLE**

**/USER/HIVE/WAREHOUSE**

**SALES\_DATA\_DATE\_PRODUCT\_PARTITION**

**DATA IS STORED  
IN THESE FILES**



# 如何存取Partition Table 資料

```
CREATE TABLE Sales_Data_Date_Product_Partition  
(  
  StoreLocation VARCHAR(30),  
  Revenue DECIMAL(10,2)  
)  
partitioned by  
(  
  OrderDate DATE,  
  product VarChar(30)  
);
```

在使用SELECT 語句時可以當  
有四個欄位存在

如果存取到設定為Partition 的  
欄位，存取速度會加快



# 如何將資料插入分割表格

## ■ 表格SCHEMA

```
CREATE TABLE Sales_Data_Date_Partition  
(  
  StoreLocation VARCHAR(30),  
  product VarChar(30),  
  Revenue DECIMAL(10,2)  
)  
partitioned by(OrderDate DATE);
```

## ■ 資料插入SQL

```
Insert into Sales_Data_Date_Partition  
partition (OrderDate ='2016-01-16')  
Values  
(  
  ('Bellandur','Nutella',7455.67),  
  ('Bellandur','Peanut Butter',5316.89),  
  ('Bellandur','Milk',2433.76),  
  ('Koramangala','Bananas',9456.01);
```

# Dynamic Partition

## ■ 如何要將資料插入有多個Partition 的表格

- 使用Dynamic Partition

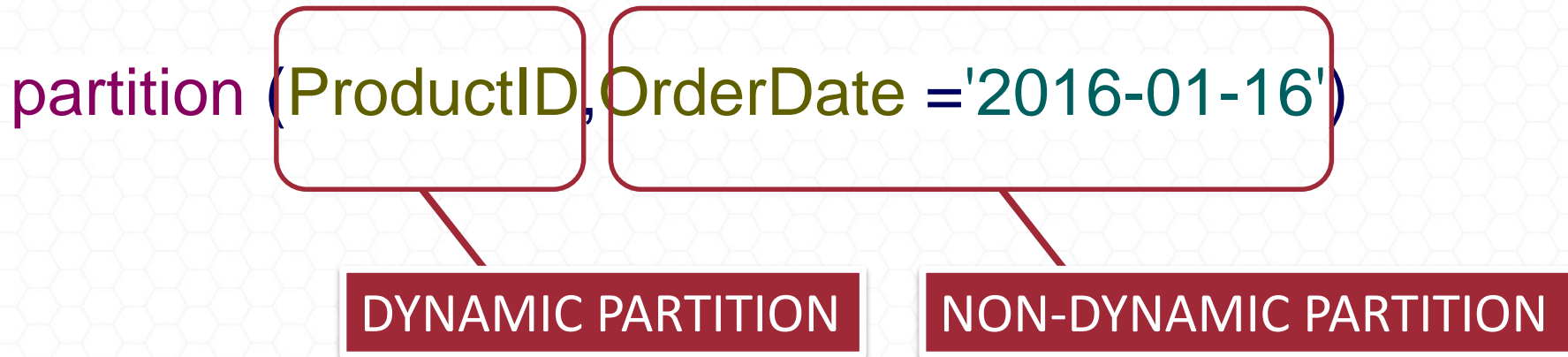
## ■ 啟動Dynamic Partition

SET hive.exec.dynamic.partition = true;

SET hive.exec.dynamic.partition.mode = nonstrict;

可以在 HIVE-SITE.XML 編輯

# Strict v.s. Non-Strict Mode



Strict Mode	Non-Strict Mode
至少有一個非動態 (Non-Dynamic) 切割的欄位	所有的切割都必須是動態 (Dynamic)



# 如何動態產生Partition Table

```
Insert into Sales_Data_Date_Product_Partition  
partition(Product,OrderDate)  
select StoreLocation,Revenue,Product,OrderDate  
from Sales_Data_Without_Partition;
```

要分割的欄位

SALES\_DATA\_WITHOUT\_PARTITION



Sales\_Data\_Date\_Product\_Partition

# SALES\_DATA\_WITHOUT\_PARTITION

StoreLocation	Product	Date	Revenue
Bellandur	Bananas	January 18,2016	8,236.33
Bellandur	Nutella	January 18,2016	7,455.67
Bellandur	Peanut Butter	January 18,2016	5,316.89
Bellandur	Milk	January 18,2016	2,433.76
Koramangala	Bananas	January 18,2016	9,456.01
Koramangala	Nutella	January 18,2016	3,644.33
Koramangala	Peanut Butter	January 18,2016	8,988.64
Koramangala	Milk	January 18,2016	1,621.58
Bellandur	Bananas	January 17,2016	2342.33
Bellandur	Nutella	January 17,2016	6345.10
Bellandur	Peanut Butter	January 17,2016	5673.01
Bellandur	Milk	January 17,2016	4543.98
Koramangala	Bananas	January 17,2016	8902.65
Koramangala	Nutella	January 17,2016	9114.67
Koramangala	Peanut Butter	January 17,2016	5102.05
Koramangala	Milk	January 17,2016	1299.45

# 將資料建立到Partition Table 中

**/USER/HIVE/WAREHOUSE**



**SALES\_DATA\_DATE\_PRODUCT\_PARTITION**



**/DATE='2015-01-17'**



**/PRODUCT=BANANAS**



**/FILE-01**

**/PRODUCT = PEANUT BUTTER**

**/DATE='2015-01-18'**



**/PRODUCT = NUTELLA**

**/PRODUCT = MILK**

**/DATE='2015-01-19'**



**/PRODUCT = MILK**

**/PRODUCT=BANANAS**

**/DATE='2015-01-20'**



**/PRODUCT = PEANUT BUTTER**

**DATA IS STORED  
IN THESE FILES**





# Bucket

# Bucket

## ■ Bucket 可以做資料取樣

- 建立 Bucket 會將資料做依鍵值切割
- 每個 Bucket 包含隨機取樣資料
- 可以使用 Bucket 去取樣部分資料

## ■ Bucket 是一種資料分割的方法

- 可以套用到分割(Partition)表格與非分割(Non-Partition)表格

# Bucket v.s. Partition

- 通常Partition 切割的數量有限制
  - 來自HDFS 的限制，無法切太多Bucket
- 將資料切成不同Bucket
  - 會知道資料會被分到哪個Bucket 中
  - 找尋資料時，也只要找一個Bucket 即可



# 建立Bucket

## ■ 如何建立Bucket

`clustered by (COLUMN_NAME) INTO N Buckets`

**N** 代表Bucket 的數量

## ■ 如何找到哪筆資料屬於哪個Bucket?

- 使用欄位的Hash 值
- 使用餘數(MODULO)找到Bucket
- 同樣值的資料會被放置於同個Bucket

# 將資料切成3 個Bucket

- 資料會被分為 0, 1, 2 三個Bucket

StudentID	FirstName	LastName	Gender	Email
1	Janani	Ravi	F	janani@loonycorn.com
2	Swetha	Kolalapudi	F	swetha@loonycorn.com
3	Navdeep	Singh	M	navdeep@loonycorn.com
4	Anu	Radha	F	anuradha@gmail.com
5	Vitthal	Srinivasan	M	vitthal@loonycorn.com
6	Jitendra	Kedia	M	jitendra@loonycorn.com

- 第一筆資料會被放到Bucket 1

□  $1 \text{ module } 3 = 1$

StudentID	FirstName	LastName	Gender	Email
1	Janani	Ravi	F	janani@loonycorn.com

# 根據餘數將資料切成三個桶

StudentID	FirstName	LastName	Gender	Email
1	Janani	Ravi	F	janani@loonycorn.com
2	Swetha	Kolalapudi	F	swetha@loonycorn.com
3	Navdeep	Singh	M	navdeep@loonycorn.com
4	Anu	Radha	F	anuradha@gmail.com
5	Vitthal	Srinivasan	M	vitthal@loonycorn.com
6	Jitendra	Kedia	M	jitendra@loonycorn.com

BUCKET 0

StudentID	FirstName	LastName	Gender	Email
3	Swetha	Kolalapudi	F	swetha@loonycorn.com
6	Jitendra	Kedia	M	jitendra@loonycorn.com

BUCKET 1

StudentID	FirstName	LastName	Gender	Email
1	Janani	Ravi	F	janani@loonycorn.com
4	Anu	Radha	F	anuradha@gmail.com

BUCKET 2

StudentID	FirstName	LastName	Gender	Email
2	Swetha	Kolalapudi	F	swetha@loonycorn.com
5	Vitthal	Srinivasan	M	vitthal@loonycorn.com

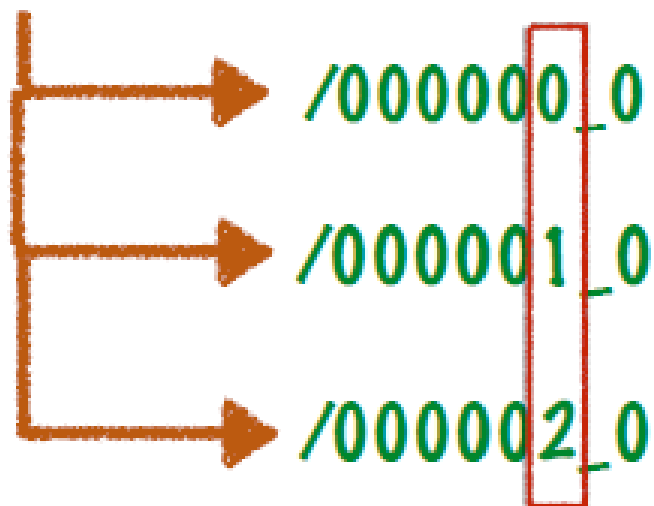


# 資料會放置在對應的檔案

/USER/HIVE/WAREHOUSE

└─▶ /STUDENTS-TABLE

DATA IS STORED IN  
THESE 3 BUCKETS



# Bucket 的優點

## ■ 有效取樣(Sampling)資料

- 取樣可以有效挑選出部份測試用資料
- 當資料集相當巨大時，Bucket 會相當有用

## ■ 加速Map-Side Join

- 透過Bucket 做 Join，則左表會清楚知道要如何迅速從右表格找到對應列，因此只需要讀取包含該資料的Bucket，不用讀取整個右表
- 會對每個左方的Bucket 表跟右方的Bucket 表做Merge Sort

# 為什麼做Bucket 會比 Partition 好

- Bucket 做的分割數是固定的(取決於N)，而 Partition 做出來的分割數是不固定的(取決於該欄位)
- 當資料能做的分割有限時，Bucket 是比較好的選擇
- Partition 如果選擇的欄位不對，有可能切割出數以萬計的檔案



# 可以根據欄位排序各Bucket 的資料

```
CREATE TABLE Sales_Data_NEW  
(  
  PRODUCTID INT,  
  ORDERDATE DATE,  
  REVENUE DECIMAL(10,2)  
)partitioned by (STOREID INT)  
Clustered by (PRODUCTID)  
SORTED BY (ORDERDATE) into 4 buckets;
```

使用Sorted By

# 建立欄位有經排序的分桶表格

BUCKETS ARE SORTED BY ORDERDATE

ProductID	OrderDate	Revenue
2	January 17,2016	8,236.93
2	January 18,2016	7,455.67
1	January 18,2016	5,316.89
1	January 17,2016	2,433.76

THE PARTITION  
WITH STOREID =1

ProductID	OrderDate	Revenue
2	January 17,2016	8,236.93
2	January 18,2016	7,455.67

THIS IS BUCKET =2

ProductID	OrderDate	Revenue
1	January 17,2016	2,433.76
1	January 18,2016	5,316.89

THIS IS BUCKET =1

# 如何將資料塞入Buckets 表格

## ■ 設定動態Bucketing (Dynamic Bucketing)

```
SET hive.enforce.bucketing = true;
```

## ■ 建立一個Buckets 表格

```
create table Reviews  
(  
  ReviewID BIGINT,  
  MovieID INT,  
  Review VARCHAR(100),  
  Rating TINYINT  
) clustered by (ReviewID) INTO 4 Buckets;
```

```
insert into Reviews  
values  
(1,1,'Amazing',5),  
(2,1,'Genre-Defining',5);
```

資料插入SQL



# 從Buckets 表格取樣

## ■ 取樣SQL

```
select * from table_name tablesample(bucket x out of y on column);
```

BUCKET

BUCKET總數

BUCKET用的欄位

```
select * from reviews tablesample(bucket 4 out of 4 on ReviewId);
```

產生Bucket 4 所有列的資料

# 取樣範例

```
select * from reviews tablesample(bucket 4 out of 4 on ReviewId);
```

```
hive> select * from reviews;
OK
4      16      Cinema At Its Best      5
2      12      Not Best As Good as Original      3
3      8       Love It 4
1      4       Overrated      1
3      9       Nailbiting!      5
1      5       OK, Not Great      3
1      1       Amazing 5
2      13      Overrated      2
1      10      Cinematic Masterpiece      5
1      2       Genre-Defining      5
1      6       Two Thumbs Up      5
2      14      Too Morbid      3
1      3       Classic 5
5      11      So Bad Its Good 0
3      7       Crossover Hit      5
4      15      Sad, Thought-Provoking      4
```

ORIGINAL TABLE

```
hive> select * from reviews tablesample(bucket 4 out of 4 on ReviewId);
OK
1      3       Classic 5
5      11      So Bad Its Good 0
3      7       Crossover Hit      5
4      15      Sad, Thought-Provoking      4
```

BUCKET 4

# 如果更改Bucket 的數量呢？

```
select * from reviews tablesample(bucket 1 out of 2 on ReviewId);
```

- 會產生新的Bucket，每個Bucket 會是由原本兩個Bucket 所組成
- 會是由Bucket 1 與 Bucket 3 所組成

```
select * from reviews tablesample(bucket 1 out of 8 on ReviewId);
```

- 會產生新的Bucket，每個Bucket 會是由Bucket 一半所組成
- 會只包含原Bucket 1 一半的列



# 如果更改Sample 使用的欄位呢？

```
select * from reviews tablesample(bucket 1 out of 4 on MovieId);
```

- 會根據MovieID 建立Buckets
- Tablesample 會從所有表格中建立新的Buckets
- 會是從新產生的 4個Bucket 中篩選一個當做輸出

# 從非Bucketed 表格做Sampling

- **TABLESAMPLE** 指令依然可以從資料中做取樣
- 但該指令會從**所有**的表格資料中取樣部份資料列
- 在Bucketed Tables 中，TABLESAMPLE 只會從Bucket 中掃合乎Hash 比例的資料
  - 效率相當低落

# BLOCK SAMPLING

- 從所有資料中取樣4 %的資料

```
select * from sales_data_new tablesample(4 PERCENT);
```

```
hive> SELECT * FROM SALES_DATA_NEW;
OK
4      2016-01-17      5673.01 1
4      2016-01-18      5316.89 1
1      2016-01-17      2342.33 1
1      2016-01-18      8236.33 1
2      2016-01-17      4543.98 1
2      2016-01-18      2433.76 1
3      2016-01-17      6345.10 1
3      2016-01-18      7455.67 1
4      2016-01-17      5102.05 2
4      2016-01-18      8988.64 2
1      2016-01-17      8902.65 2
1      2016-01-18      9456.01 2
2      2016-01-17      1299.45 2
2      2016-01-18      1621.58 2
3      2016-01-17      9114.67 2
3      2016-01-18      3644.33 2
Time taken: 0.113 seconds, Fetched: 16 rows(1)
hive> SELECT * FROM SALES_DATA_NEW TABLESAMPLE(4 PERCENT);
OK
4      2016-01-17      5673.01 1
4      2016-01-17      5102.05 2
Time taken: 0.142 seconds, Fetched: 2 rows(1)
hive> SELECT * FROM SALES_DATA_NEW TABLESAMPLE(20 PERCENT);
OK
4      2016-01-17      5673.01 1
4      2016-01-18      5316.89 1
4      2016-01-17      5102.05 2
4      2016-01-18      8988.64 2
```

# ROW COUNT SAMPLING

## ■ 從資料集中取樣兩列資料

```
select * from sales_data_new tablesample(2 ROWS);
```

```
hive> SELECT * FROM SALES_DATA_NEW;
OK
4      2016-01-17      5673.01 1
4      2016-01-18      5316.89 1
1      2016-01-17      2342.33 1
1      2016-01-18      8236.33 1
2      2016-01-17      4543.98 1
2      2016-01-18      2433.76 1
3      2016-01-17      6345.10 1
3      2016-01-18      7455.67 1
4      2016-01-17      5102.05 2
4      2016-01-18      8988.64 2
1      2016-01-17      8902.65 2
1      2016-01-18      9456.01 2
2      2016-01-17      1299.45 2
2      2016-01-18      1621.58 2
3      2016-01-17      9114.67 2
3      2016-01-18      3644.33 2
Time taken: 0.099 seconds, Fetched: 16 row(s)
hive> SELECT * FROM SALES_DATA_NEW TABLESAMPLE(2 ROWS);
OK
4      2016-01-17      5673.01 1
4      2016-01-18      5316.89 1
```



# LIMIT v.s. Tablesample

- 掃描整個表格以後再回傳四筆
  - 回傳首四筆資料

```
select * from SALES_DATA_NEW limit 4;
```

- 從Bucket 挑出資料 (速度較快)

```
select * from SALES_DATA_NEW  
tablesample(bucket 1 out of 4 on ProductID);
```

# Windowing

# Windowing

- 如何計算到當前為止的累計加總資料？
  - 例如: 將每一列的結果消費金額相加？
  - 使用WINDOW Function 可以累加到目前為止的資料

OrderID	StoreID	ProductID	OrderDate	Revenue	Running Total
1	2	2	2016-01-17	1299.45	1299.45
2	1	1	2016-01-17	2342.33	3641.78
3	1	2	2016-01-17	4543.98	8185.76
4	2	4	2016-01-17	5102.05	13287.81
5	1	4	2016-01-17	5673.01	18960.82
6	1	3	2016-01-17	6345.10	25305.92
7	2	1	2016-01-17	8902.65	34208.57
8	2	3	2016-01-17	9114.67	43323.24
9	2	2	2016-01-18	1621.58	44944.82
10	1	2	2016-01-18	2433.76	47378.58
11	2	3	2016-01-18	3644.33	51022.91
12	1	4	2016-01-18	5316.89	56339.8
13	1	3	2016-01-18	7455.67	63795.47
14	1	1	2016-01-18	8236.33	72031.8
15	2	4	2016-01-18	8988.64	81020.44
16	2	1	2016-01-18	9456.01	90476.45

# Windowing 範例

- 可以使用AVG, MAX, MIN, COUNT, SUM 做各種計算

```
from Sales_Data  
select OrderID,storeID,ProductID,OrderDate,Revenue,  
sum(Revenue)over(order by OrderID ROWS BETWEEN  
UNBOUNDED PRECEDING and CURRENT ROW) as Running_Total;
```



# 如果希望根據日期各字計算Revenue 百分比

OrderID	StoreID	ProductID	OrderDate	Revenue	Percentage_Revenue_Day_Level
1	2	2	1/17/2016	1299.45	3.0
8	2	3	1/17/2016	9114.67	21.0
7	2	1	1/17/2016	8902.65	20.5
THEIR SUM IS 100.				6345.1	14.6
				5673.01	13.1
				5102.05	11.8
3	1	2	1/17/2016	4543.98	10.5
2	1	1	1/17/2016	2342.33	5.4
16	2	1	1/18/2016	9456.01	20.1
15	2	4	1/18/2016	8988.64	19.1
14	1	1	1/18/2016	8236.33	17.5
THEIR SUM IS 100.				7455.67	15.8
				5316.89	11.3
				3644.33	7.7
11	2	3	1/18/2016	3644.33	7.7
10	1	2	1/18/2016	2433.76	5.2
9	2	2	1/18/2016	1621.58	3.4

# 計算每日的營收

REVENUE X 100

---

TOTAL REVENUE FOR THE DAY

```
from Sales_Data
select OrderID,storeID,ProductID,OrderDate,Revenue,
(Revenue)*100/
(SUM(REVENUE)over(partition by OrderDate)) as
Percentage_Revenue_Day_Level;
```

# Hive Tuning

# 如何合併兩個表格 (Names 與 Trades)

## Trades

SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE	LAST
--------	--------	------	------	-----	-------	------

## Names

SYMBOL	NAME
RIL	Reliance
NESTLEIND	Nestle

## Revenue

TOTAL REVENUE	NAME	SYMBOL
\$ 4 Billion	Reliance	RIL
\$ 0.5 Billion	Nestle	NESTLEIND

```
SELECT N.SYMBOL, N.NAME, T.HIGH , T.TIMESTAMP  
FROM NAMES N  
OUTER JOIN TRADES T  
ON N.SYMBOL = T.SYMBOL
```



# Map Reduce 工作會將資料轉變為key/value

$\langle \text{Rownum}, \langle \text{Row}, \text{Tablename} \rangle \rangle$



$\langle \text{Symbol}, \langle \text{Other Columns}, \text{Table} \rangle \rangle$

N.SYMBOL, N.NAME

RIL	Reliance
-----	----------

RIL	"Reliance", N
-----	---------------

T.SYMBOL, T.HIGH, T.TIMESTAMP

RIL	100	02DEC2014
-----	-----	-----------

RIL	"100, 02DEC2014", T
-----	---------------------

Hive 會將表格合併的工作  
放在一個 Map/Reduce 工作

# 如何合併三個表格

## Trades

SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE	LAST
--------	--------	------	------	-----	-------	------

## Names

SYMBOL	NAME
RIL	Reliance
NESTLEIND	Nestle

## Revenue

TOTAL REVENUE	NAME	SYMBOL
\$ 4 Billion	Reliance	RIL
\$ 0.5 Billion	Nestle	NESTLEIND

```
SELECT NAMES.SYMBOL, TRADES.SERIES,  
REVENUE.TOTAL_REVENUE,  
FROM TRADES JOIN NAMES ON  
(TRADES.SYMBOL = NAMES.SYMBOL)  
JOIN REVENUE ON (REVENUE.SYMBOL = NAMES.SYMBOL)
```

Hive 會將工作  
變成兩個 Map/Reduce 工作

# 假設表格是以下面的大小計算

Trades

SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE	LAST
--------	--------	------	------	-----	-------	------

500GB

Names

SYMBOL	NAME
RIL	Reliance
NESTLEIND	Nestle

50MB

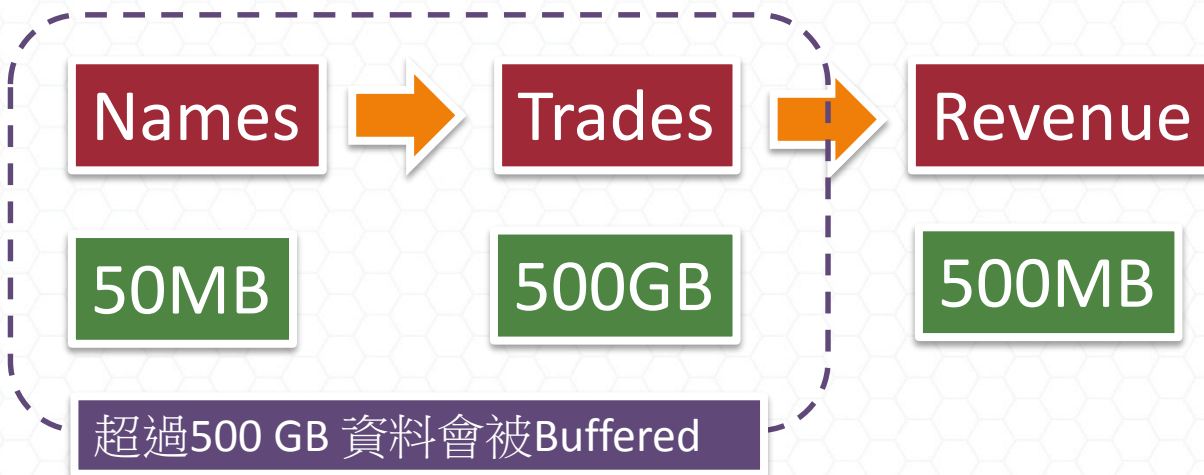
Revenue

TOTAL REVENUE	NAME	SYMBOL
\$ 4 Billion	Reliance	RIL
\$ 0.5 Billion	Nestle	NESTLEIND

500MB

# 最後一個表格會被放在Reduced 階段

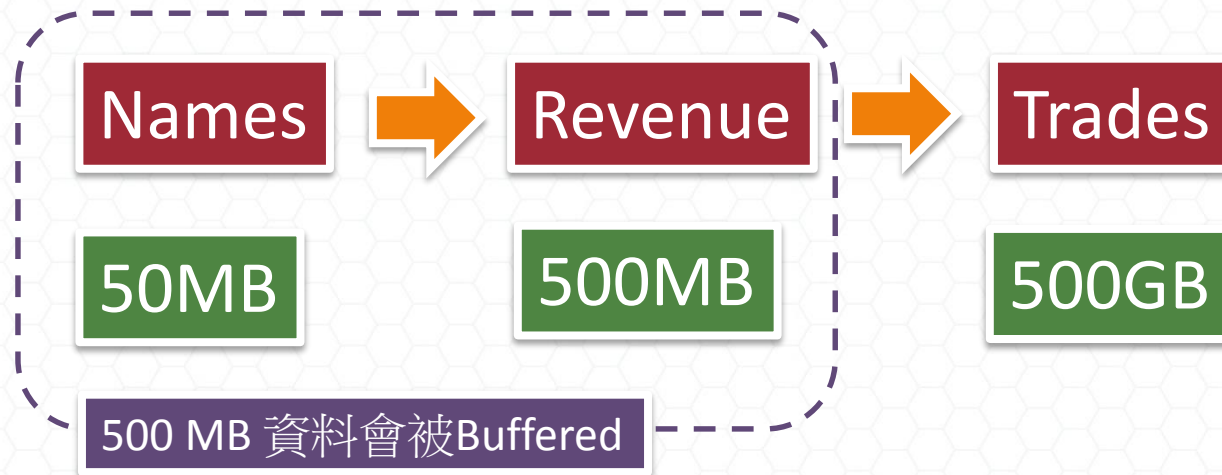
```
SELECT NAMES.SYMBOL, TRADES.SERIES,  
REVENUE.TOTAL_REVENUE,  
FROM TRADES JOIN NAMES ON  
(TRADES.SYMBOL = NAMES.SYMBOL)  
JOIN REVENUE ON (REVENUE.SYMBOL = NAMES.SYMBOL)
```





# 最大的表格應該被放在最後

```
SELECT NAMES.SYMBOL, TRADES.SERIES,  
REVENUE.TOTAL REVENUE,  
FROM NAMES JOIN REVENUE  
ON (REVENUE.SYMBOL = NAMES.SYMBOL) JOIN TRADES  
ON (TRADES.SYMBOL = NAMES.SYMBOL)
```



# STREAMTABLE

```
SELECT NAMES.SYMBOL, TRADES.SERIES,  
REVENUE.TOTAL REVENUE,  
FROM NAMES JOIN TRADES ON  
(TRADES.SYMBOL = NAMES.SYMBOL)  
JOIN REVENUE ON (REVENUE.SYMBOL = NAMES.SYMBOL)
```



```
SELECT /*+ STREAMTABLE(TRADES) */  
NAMES.SYMBOL, TRADES.SERIES, REVENUE.TOTAL  
REVENUE,  
FROM NAMES JOIN TRADES ON  
(TRADES.SYMBOL = NAMES.SYMBOL)  
JOIN REVENUE ON (REVENUE.SYMBOL = NAMES.SYMBOL)
```

# Join 會在Where 之前被Evaluate

SYMBOL	NAME	SECTOR
RIL	Reliance	ENERGY
NESTLEIND	Nestle	FMCG
TATA	TATA	AUTOMOBILE

**NAMES**

TOTAL REVENUE	NAME	SYMBOL
\$ 4 Billion	Reliance	RIL
\$ 0.5 Billion	Nestle	NESTLEIND

**REVENUE**

```
SELECT NAMES.SYMBOL,REVENUE.TOTAL_REVENUE,  
FROM NAMES  
LEFT JOIN REVENUE ON  
(REVENUE.SYMBOL = NAMES.SYMBOL)  
WHERE SECTOR = 'FMCG';
```

Join 會先被執行

# Join 可以套用在根本不存在的Output

SYMBOL	NAME	SECTOR
RIL	Reliance	ENERGY
NESTLEIND	Nestle	FMCG
TATA	TATA	AUTOMOBILE

NAMES

TOTAL REVENUE	NAME	SYMBOL
\$ 4 Billion	Reliance	RIL
\$ 0.5 Billion	Nestle	NESTLEIND

REVENUE

SYMBOL	TOTAL REVENUE
RIL	\$ 4 Billion
NESTLEIND	\$ 0.5 Billion
TATA	-

JOIN OUTPUT

WHERE SECTOR = 'FMCG' ;

REVENUE  
(SYMBOL, NAME, SYMBOL)

SYMBOL	TOTAL REVENUE
NESTLEIND	\$ 0.5 Billion

FINAL OUTPUT



# IN

TOTAL REVENUE	NAME	SYMBOL
\$ 4 Billion	Reliance	RIL
\$ 0.5 Billion	Nestle	NESTLEIND

REVENUE

SYMBOL	NAME	SECTOR
RIL	Reliance	ENERGY
NESTLEIND	Nestle	FMCG
TATA	TATA	AUTOMOBILE

NAMES



SYMBOL
RIL
NESTLEIND

```
SELECT NAMES.SYMBOL  
FROM NAMES  
WHERE NAMES.SYMBOL IN  
(SELECT SYMBOL FROM REVENUE);
```

# NOT IN

TOTAL REVENUE	NAME	SYMBOL
\$ 4 Billion	Reliance	RIL
\$ 0.5 Billion	Nestle	NESTLEIND

REVENUE

SYMBOL	NAME	SECTOR
RIL	Reliance	ENERGY
NESTLEIND	Nestle	FMCG
TATA	TATA	AUTOMOBILE

NAMES



SYMBOL
TATA

```
SELECT NAMES.SYMBOL  
FROM NAMES  
WHERE NAMES.SYMBOL NOT IN  
(SELECT SYMBOL FROM REVENUE);
```

IN 跟 NOT IN 只能針對  
一個欄位篩選

# EXISTS

- Exists 必需要搭配SUBQUERY 一起使用

```
SELECT *  
FROM TABLE_NAME  
WHERE EXISTS  
SUBQUERY;
```

# Exists

TOTAL REVENUE	NAME	SYMBOL
\$ 4 Billion	Reliance	RIL
\$ 0.5 Billion	Nestle	NESTLEIND

**REVENUE**

SYMBOL	NAME	SECTOR
RIL	Reliance	ENERGY
NESTLEIND	Nestle	FMCG
TATA	TATA	AUTOMOBILE

**NAMES**



SYMBOL
RIL
NESTLEIND

```
SELECT NAMES.SYMBOL
FROM NAMES
WHERE EXISTS
(SELECT TOTAL_REVENUE FROM REVENUE
NAMES.name = REVENUE.name);
```



# LEFT SEMI JOIN

SYMBOL	NAME	SECTOR
RIL	Reliance	ENERGY
NESTLEIND	Nestle	FMCG
TATA	TATA	AUTOMOBILE

**NAMES**

TOTAL REVENUE	NAME	SYMBOL
\$ 4 Billion	Reliance	RIL
\$ 0.5 Billion	Nestle	NESTLEIND

**REVENUE**

```
SELECT
NAMES.SYMBOL
FROM NAMES
WHERE
NAMES.SYMBOL IN
(SELECT SYMBOL
FROM REVENUE);
```



```
SELECT NAMES.SYMBOL
FROM NAMES
LEFT SEMI JOIN REVENUE
ON
(NAMES.SYMBOL = REVENUE.SYMBOL);
```

不能使用來自右邊表格的欄位

# LEFT SEMI JOIN v.s. IN & EXISTS

- LEFT SEMI JOIN 會比IN & EXISTS 更有效率，因為只需要查找右邊的表格是否有任意資料比對到才開始合併
- IN & EXISTS 需要掃描右方所有，才開始查找資料

# MAP SIDE JOIN

- 只使用到Mapper，並不需要使用到Reducer
- 會使用Map Side Join 的情形
  - 有一個表格非常的小，這較小的表格會被儲存在記憶體中，該表格會被複製到每個Mapper 本地端的磁碟
  - 假設兩個表格都有在將要Join 的欄位做Bucketing

# 當有一個表格的量特別小時

## NAMES

SYMBOL	NAME
RIL	Reliance
NESTLEIND	Nestle

## TRADES

SYMBOL	SERIES	OPEN	HIGH	LOW
--------	--------	------	------	-----

假設NAMES 是比較小的表格

```
SELECT NAMES.SYMBOL,SERIES,HIGH  
FROM NAMES JOIN TRADES  
ON(NAMES.SYMBOL = TRADES.SYMBOL);
```



# 如果合併一個較小的表格

- TRADES 會被切成Chunk，並分散到各個Mapper去，Names 會被複製到每個Mapper 本地端的磁碟



# 哪些JOIN 符合 MAP SIDE JOIN

## ■ INNER JOIN

```
SELECT NAMES.SYMBOL,SERIES,HIGH  
FROM NAMES JOIN TRADES  
ON(NAMES.SYMBOL = TRADES.SYMBOL);
```

## ■ RIGHT OUTER JOIN

```
SELECT NAMES.SYMBOL,SERIES,HIGH  
FROM NAMES RIGHT OUTER JOIN TRADES  
ON(NAMES.SYMBOL = TRADES.SYMBOL);
```

假設NAMES 是比較小的表格

# 可宣告MAPJOIN

## ■ 使用者也可以明白宣告MAPJOIN

□ /\*+ MAPJOIN(TRADES) \*/

```
SELECT /*+ MAPJOIN(TRADES) */  
NAMES.SYMBOL,SERIES,HIGH  
FROM NAMES JOIN TRADES  
ON(NAMES.SYMBOL = TRADES.SYMBOL);
```

# 對JOIN 的欄位做Bucketing

## ■ 對Symbol 做Bucketing

### NAMES

SYMBOL	NAME
RIL	Reliance
NESTLEIND	Nestle

### TRADES

SYMBOL	SERIES	OPEN	HIGH	LOW
--------	--------	------	------	-----

```
SELECT NAMES.SYMBOL,SERIES,HIGH  
FROM NAMES JOIN TRADES  
ON(NAMES.SYMBOL = TRADES.SYMBOL);
```



# 兩個表格的Symbol 欄位都做Bucketing

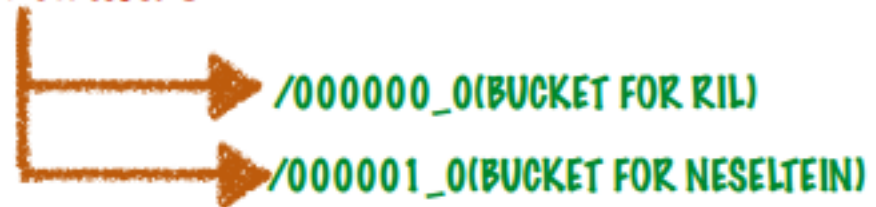
開啟設定 `set hive.optimize.bucketmapjoin = true;`

## NAMES

SYMBOL	NAME
RIL	Reliance
NESTLEIND	Nestle

/USER/HIVE/WAREHOUSE

## /NAMES

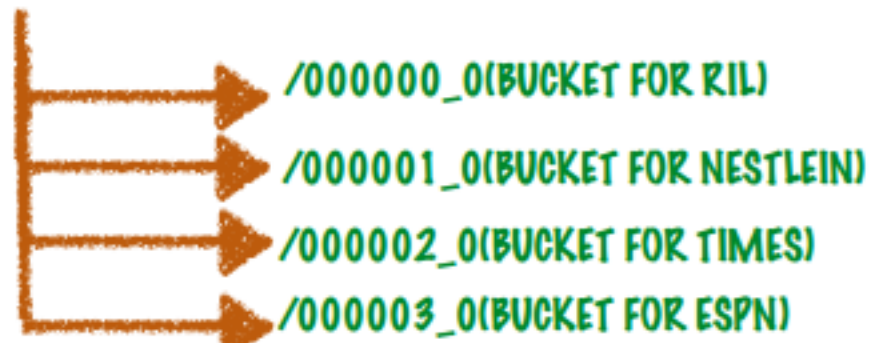


/USER/HIVE/WAREHOUSE

## /TRADES

## TRADES

SYMBOL	SERIES	OPEN	HIGH	LOW
RIL	EQ	23	23.5	22.7
TIMES	EQ	23	23.5	22.7
NESTLIN	EQ	33	33.4	33.8
ESPN	EQ	44	47.8	45.5



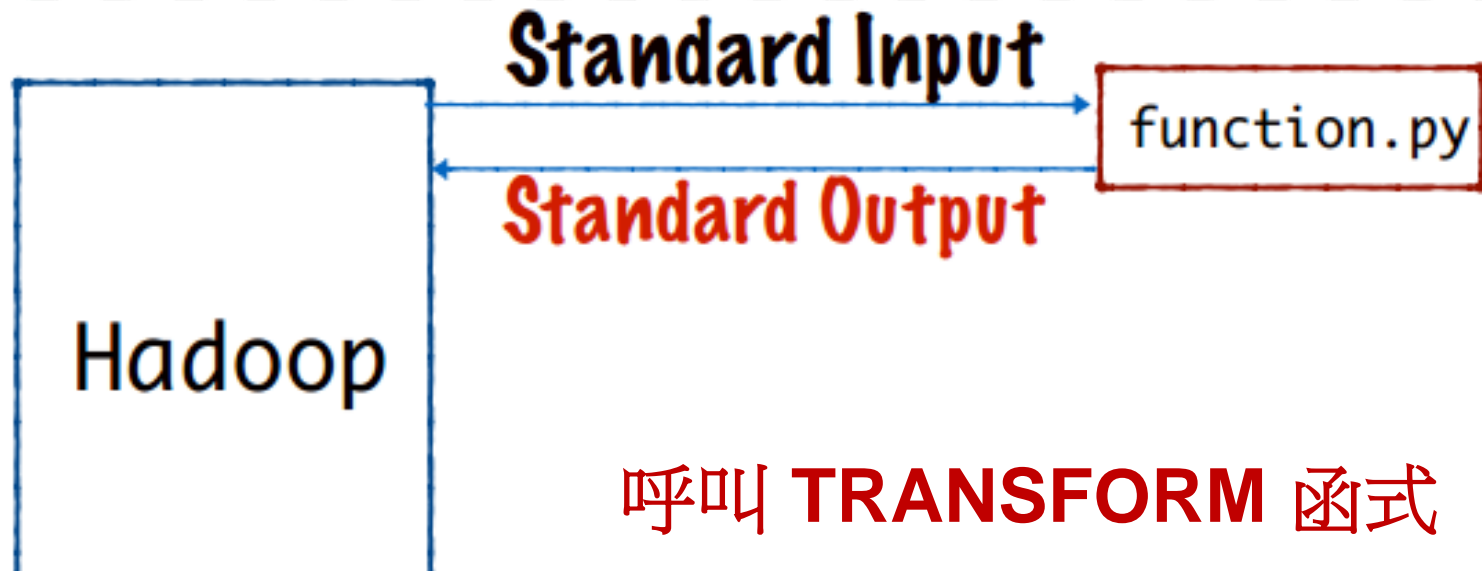
# 使用者自訂函式

# 使用者自定義函式

- **Hive** 可以讓使用者自己定義要用的函式
  - ▣ 假設使用者要建立 **REPLACETEXT()** 函式替代掉字串中的部分內容
  - ▣ 使用者可以用 **JAVA** 或 **Python** 實作函式
- 函式是透過 **Python** 的 **Streaming API** 實作
  - ▣ **Streaming API** 使用的是標準 **IO** 做溝通



# Hive 如何呼叫Python



呼叫 **TRANSFORM** 函式

```
SELECT TRANSFORM(firstname,lastname) USING  
'python function.py' as isLonger from  
employees;
```



# Function.py

- 透過Standard IN 與 OUT 接續Streaming API

```
#!/usr/bin/python
import sys
for line in sys.stdin:
    (firstname,lastname)=line.split('\t')
    if len(firstname)>len(lastname):
        print "TRUE"
```

- 必須將Function 註冊後才能使用

ADD FILE function.py

# Spark Streaming

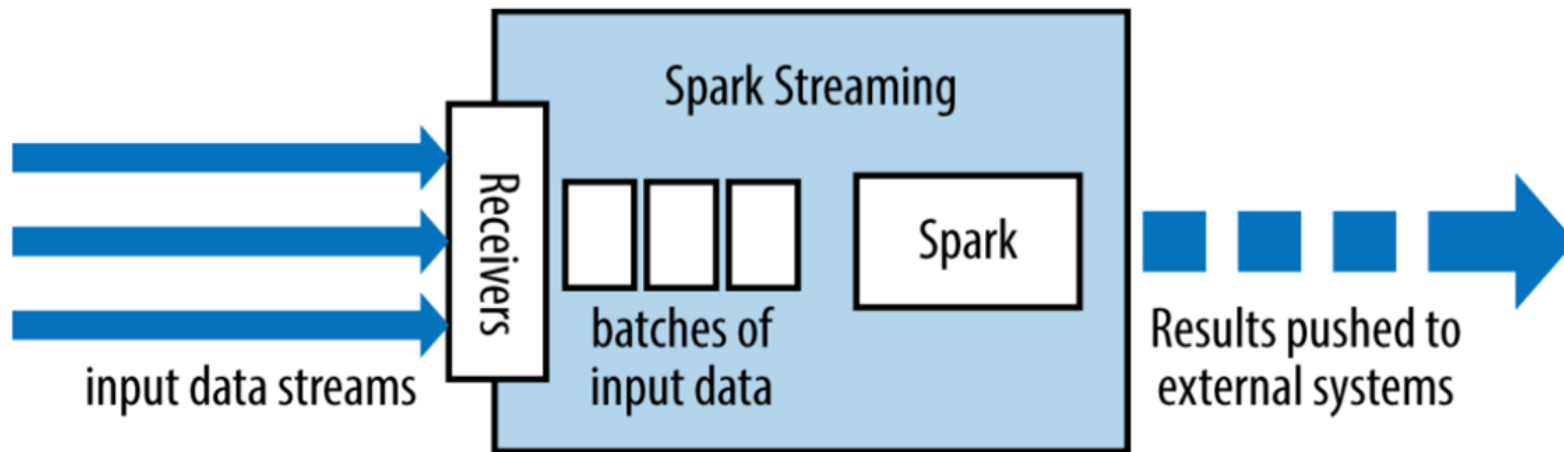
# 為什麼需要Spark Streaming

- 實時分析、更新資料串流，而非批次處理巨量資料
- 根據網站使用者的行為，做出即時反應
  - RTB - 網路廣告
- 根據感測器資料做出即時警示
  - 使用物聯網了解製程流程

# 什麼是Spark Streaming

## ■ Spark Streaming 特性

- ▣ 可擴展 – 使用分散式架構分散處理RDD
- ▣ 高吞吐
- ▣ 容錯





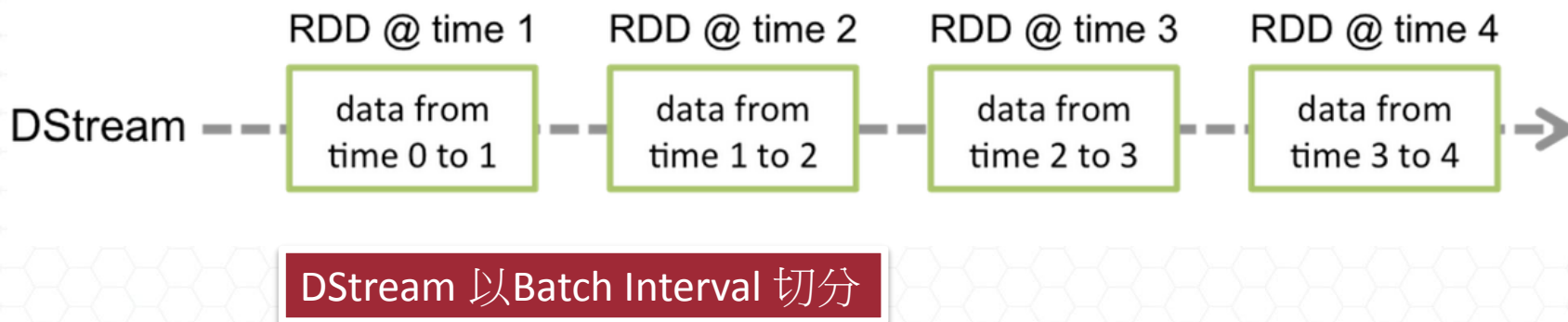
# 什麼是Spark Streaming

- 可以整合來自 file systems, socket connections, & Akka actors. *Advanced sources*: Kafka, Flume, Kinesis, Twitter, 等資料源



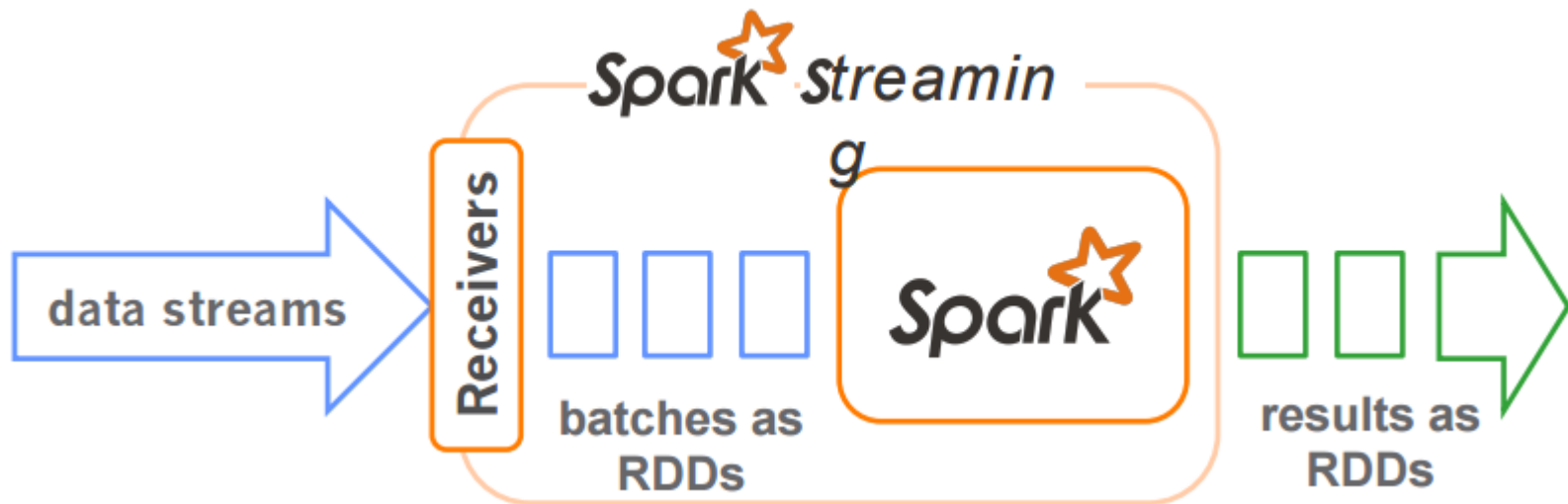
# Discretized Streams (Micro Batch)

- 將工作拆解成連續的資料流 (RDD序列)
  - 每個步驟都會產生新的RDD
  - 可以對該RDD進行Transformation 與 Action
  - 可以直接存取RDD



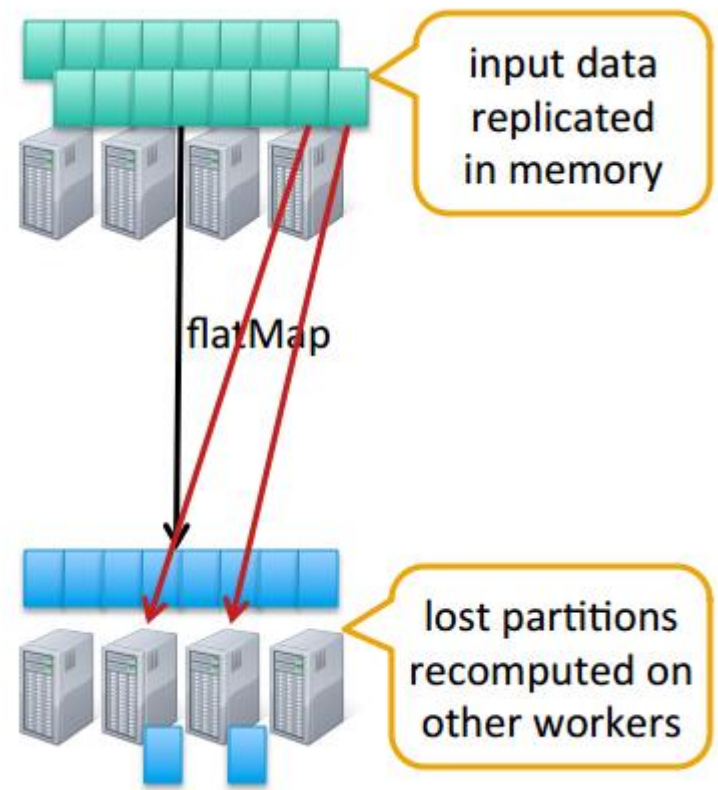
# Dstream to RDD

- 每一段 Dstream 可以被分批轉換成RDD
- RDD 會被複製到叢集中，具有容錯性
- 每個 DStream 動作都對應到 RDD transformation
- 可以透過 API 直接存取 RDD
- 可以與Spark Batch 工作併行



# 高容錯性

- 每一批輸入資料會被複製到記憶體中
- 若因為worker 失敗導致資料消失，可以從複製資料重新計算
- 所有轉換都是高容錯性，只做單次轉換





# 高容錯性

- 每個輸入資料都會被複製到至少兩個Worker Nodes
- 可以設定checkpoint 目錄滿足重啟Stream 的需求
  - ▣ 設定 checkpoint 到StreamingContext
  - ▣ Stateful 資料需要Checkpoints

# 改善Receiver 容錯性

## ■ Push 機制

- 如果Receiver 毀損,資料就消失了

## ■ Pull 機制的Receiver 容錯性比較高

- 可以將資料Queue 在來源端

# 改善Driver Script 容錯性

- 資料會被複製，但Driver Node 會遭遇Single Point Failure 的風險
- 使用StreamingContext.getOrCreate
  - ▣ 在分散式檔案系統建立Checkpoints
  - ▣ StreamingContext.getOrCreate (Checkpoints 目錄)
  - ▣ 可以從checkpoint 重啟 Driver Script
- 必須使用zookeeper 以及 manager 監控driver node 狀態，必要時重啟



# Spark Streaming 範例



# 目標：從文字中找出包含ERROR 字樣的資料

- 讀取文字資料，找到文字中包含有ERROR 的行，並將該筆資料印出
- 參照 `streaming/Streaming.py`

# Streaming.py 完整範例

```
from pyspark import SparkContext
from pyspark.streaming import StreamingContext

sc = SparkContext(appName="StreamingErrorCount")
ssc = StreamingContext(sc, 1)

ssc.checkpoint("hdfs:///user/hdp/streaming")
lines = ssc.socketTextStream(sys.argv[1], int(sys.argv[2]))
counts = lines.flatMap(lambda line: line.split(" "))\
    .filter(lambda word: "ERROR" in word)\
    .map(lambda word: (word, 1))\
    .reduceByKey(lambda a, b: a+b)

counts.pprint()
ssc.start()
ssc.awaitTermination()
```

# 啟用Streaming.py

## ■ 先啟用Netstat Util

- ▣ nc -lk 9999

## ■ 送出Spark Streaming 工作

- ▣ spark-submit Streaming.py localhost 9999

- ▣ 聆聽來自PORT 9999的資料

# 將資料倒入9999 服務

```
-----  
Time: 2016-06-24 01:37:00  
-----
```

```
Time: 2016-06-24 01:37:01  
-----
```

```
(u'ERROR', 1)  
-----
```

```
Time: 2016-06-24 01:37:06  
-----
```

```
('u'ERROR', 1)
```

```
> nc -lk 9999
```

```
This line won't be printed
```

```
This line has ERROR
```

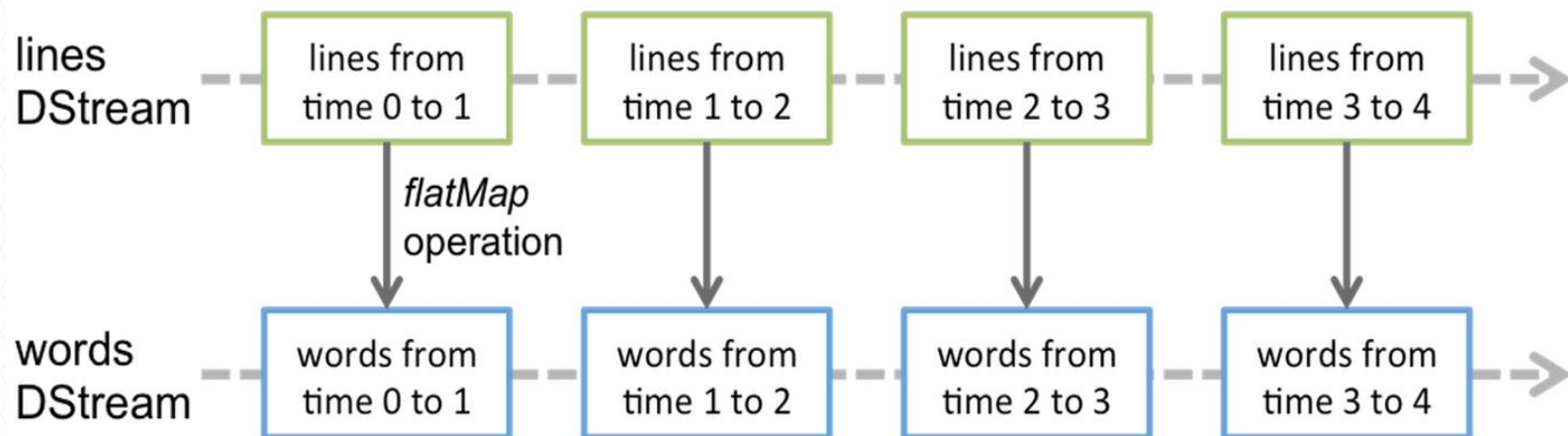
```
This line has ERROR too
```

**We typed each  
sentence after a  
couple of seconds**



# Stateless

## ■ 每個Dstream 之間互不關連



# 常見Stateless 操作

- 可以使用一般RDD 常見的Transformation 函式
  - ▣ Map
  - ▣ FlatMap
  - ▣ Filter
  - ▣ reduceByKey

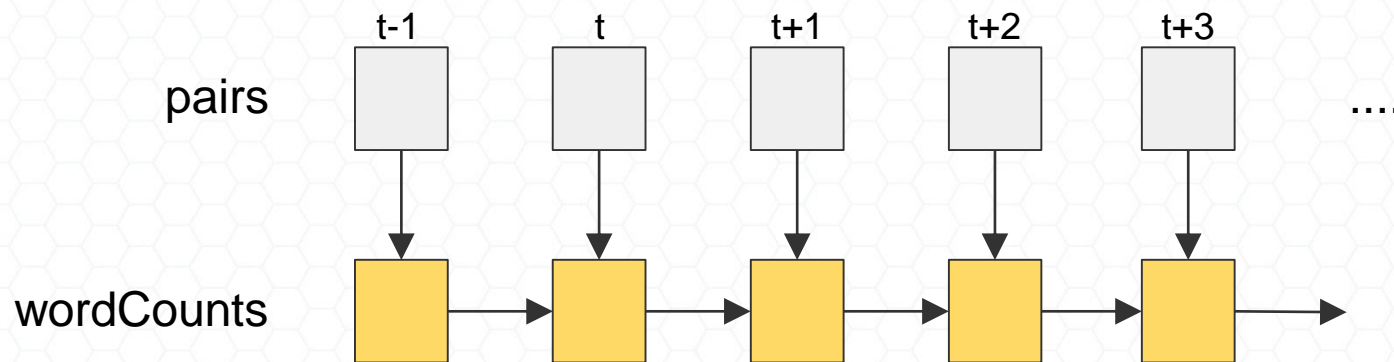
# Stateful- updateByKey

- `updateStateByKey` 操作可讓新訊息不斷更新時，依然維持其狀態：
  - ▣ 定義狀態-狀態可以是任何的資料類型
  - ▣ 定義狀態更新函數-怎樣利用更新前的狀態和從輸入串流裡面獲取的更新值

```
def updateFunction(newValues, runningCount):  
    if runningCount is None:  
        runningCount = 0  
    return sum(newValues, runningCount)  
runningCounts = pairs.updateStateByKey(updateFunction)
```

# Stateful Data

- 可以保持Dstream 的狀態
  - e.g. 計算加總
  - 聚合網路行為中的Session





# Transform操作

## ■ 允許在DStream運行任何RDD轉換函數

- ▣ 例如，如果想藉由連接帶有預先計算的垃圾邮件訊息的輸入資料串流來清理即時資料

```
spamInfoRDD = sc.pickleFile(...) # RDD containing spam information
```

```
# join data stream with spam information to do data cleaning
```

```
cleanedDStream = wordCounts.transform(lambda rdd:  
rdd.join(spamInfoRDD).filter(...))
```

# 各種Interval 計算

## ■ Batch Interval:

- ▣ Dstream 產生的資料

## ■ Slide Interval

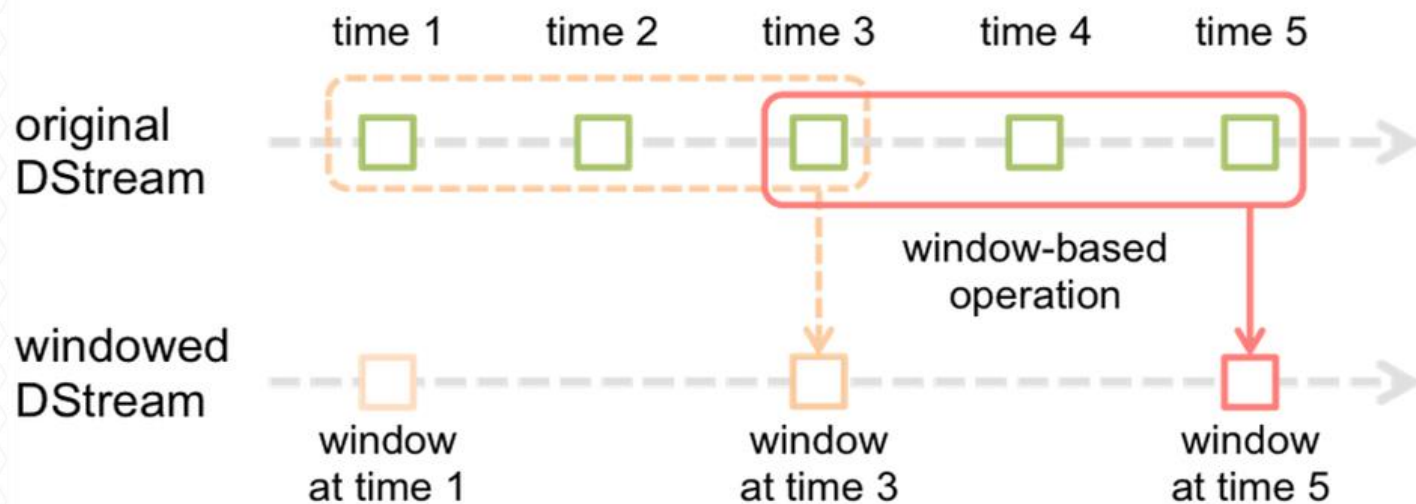
- ▣ 多長Windowed Transformation 被計算

## ■ Window Interval

- ▣ 多長回顧的時間

# Window

- window length = 3
- sliding interval = 2



```
windowedWordCounts = pairs.reduceByKeyAndWindow(lambda x, y: x + y, lambda x, y: x - y, 30, 10)
```

# 轉變成Stateful 操作

```
ssc.checkpoint("hdfs:///user/hdp/streaming")
lines = ssc.socketTextStream(sys.argv[1],
int(sys.argv[2]))
counts = lines.flatMap(lambda line: line.split(" "))\
    .filter(lambda line:"ERROR" in line)\
    .map(lambda word: (word, 1))\
    .reduceByKeyAndWindow(lambda x, y: x
+ y, lambda x, y: x - y, 30, 10)
```

Window = 30 s  
Slide = 10 s  
Batch = 1s



# DStreams + RDDs

## ■ 結合線上Data Stream 與歷史資料

- 可以從歷史資料產生模型
- 用模型預測線上Data Stream 行為

## ■ 結合 Streaming 及MLlib, GraphX

- 線下學習，線上預測
- 線上學習與預測



## ■ 結合SQL 做資料探索

- `select * from streaming_data`

The background features a light blue hexagonal grid pattern. Overlaid on this is a large, faint, light blue circular graphic composed of concentric rings and radial lines, resembling a stylized spiral or a target. A solid dark blue horizontal bar runs across the top of the image, and a similar but slightly textured dark blue bar runs across the bottom.

**THANK YOU**