

# HBase 簡介與實作

David Chiu  
2016/07/07

# NoSQL

# 關聯式資料庫

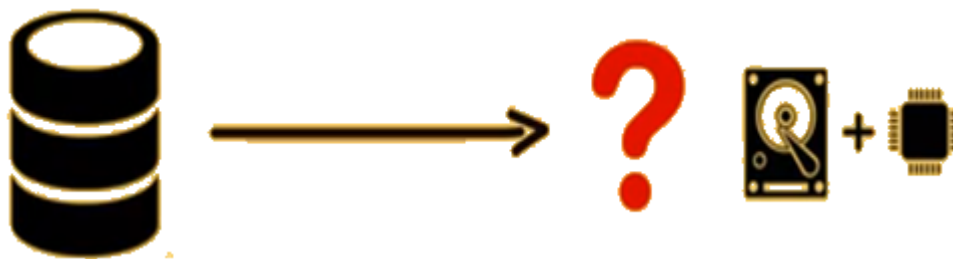
- 安全存儲、管理資料
  - 有效管理磁碟上的資料
- 保持資料的一致性
  - ACID 四原則
- 可以透過標準模型整合資料
  - 使用SQL 操作資料





# 面對大量資料

- 垂直擴展 (Vertical Scaling – Scale Up)
  - 增加單一台的 CPU, Storage, Memory
- 水平擴展 (Horizontal Scaling – Scale Out)
  - 使用多台機器分散工作



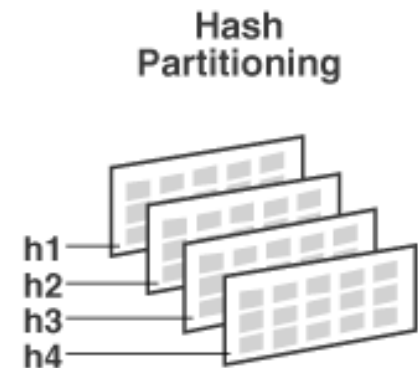
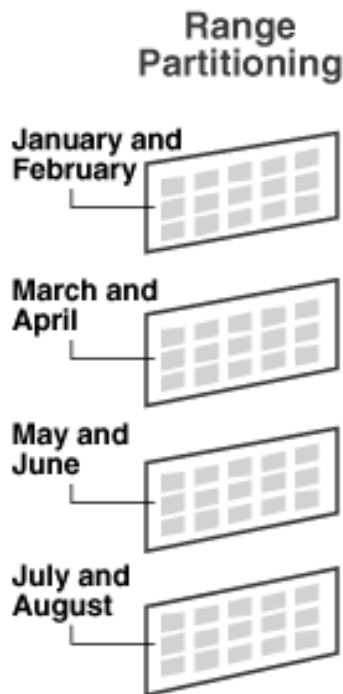
# 資料表分割

e.g. 用時間切割

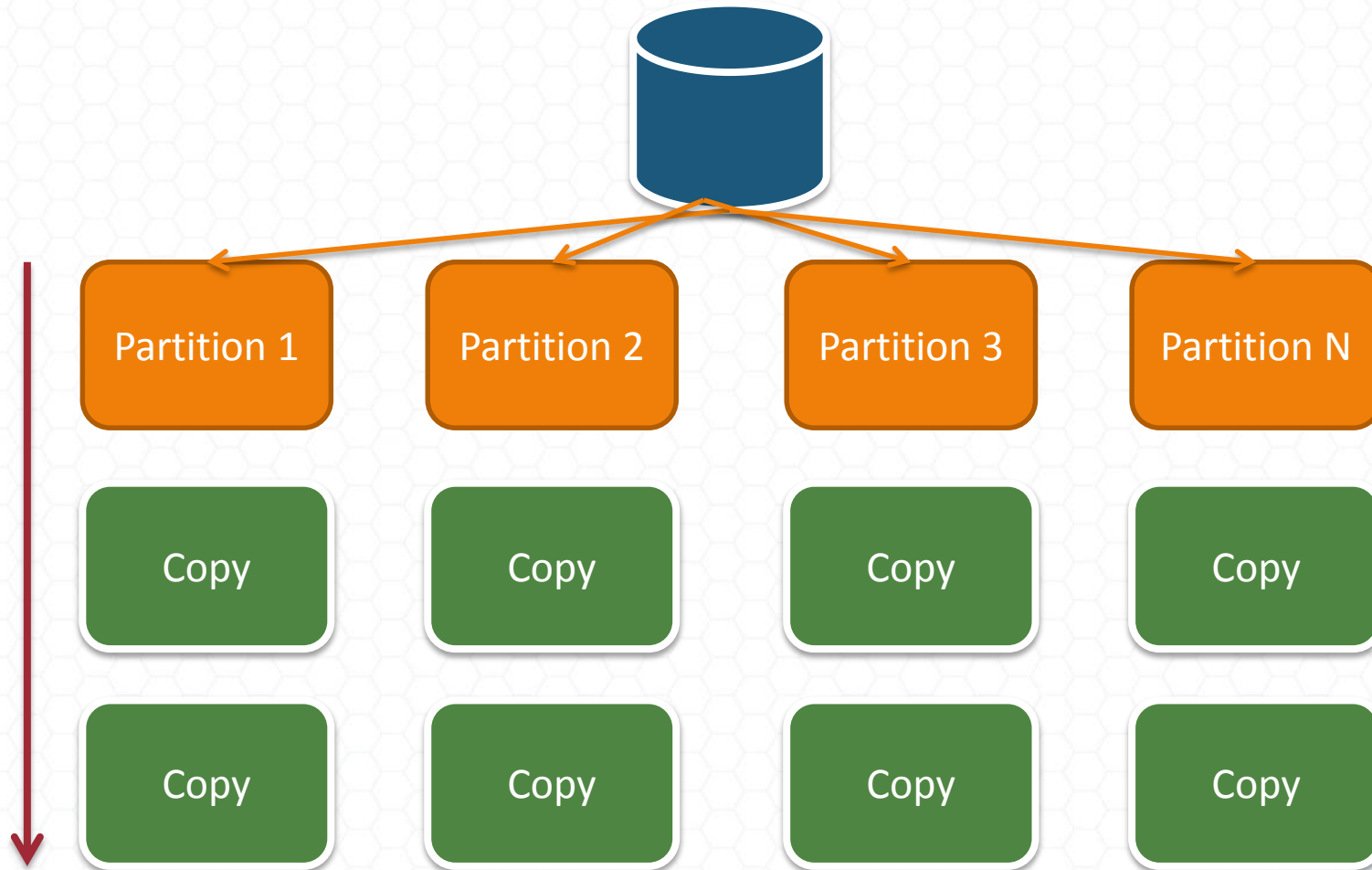
查詢作業就可以只針對這個部分去處理，而不用掃描整張資料表，才能找到指定資料



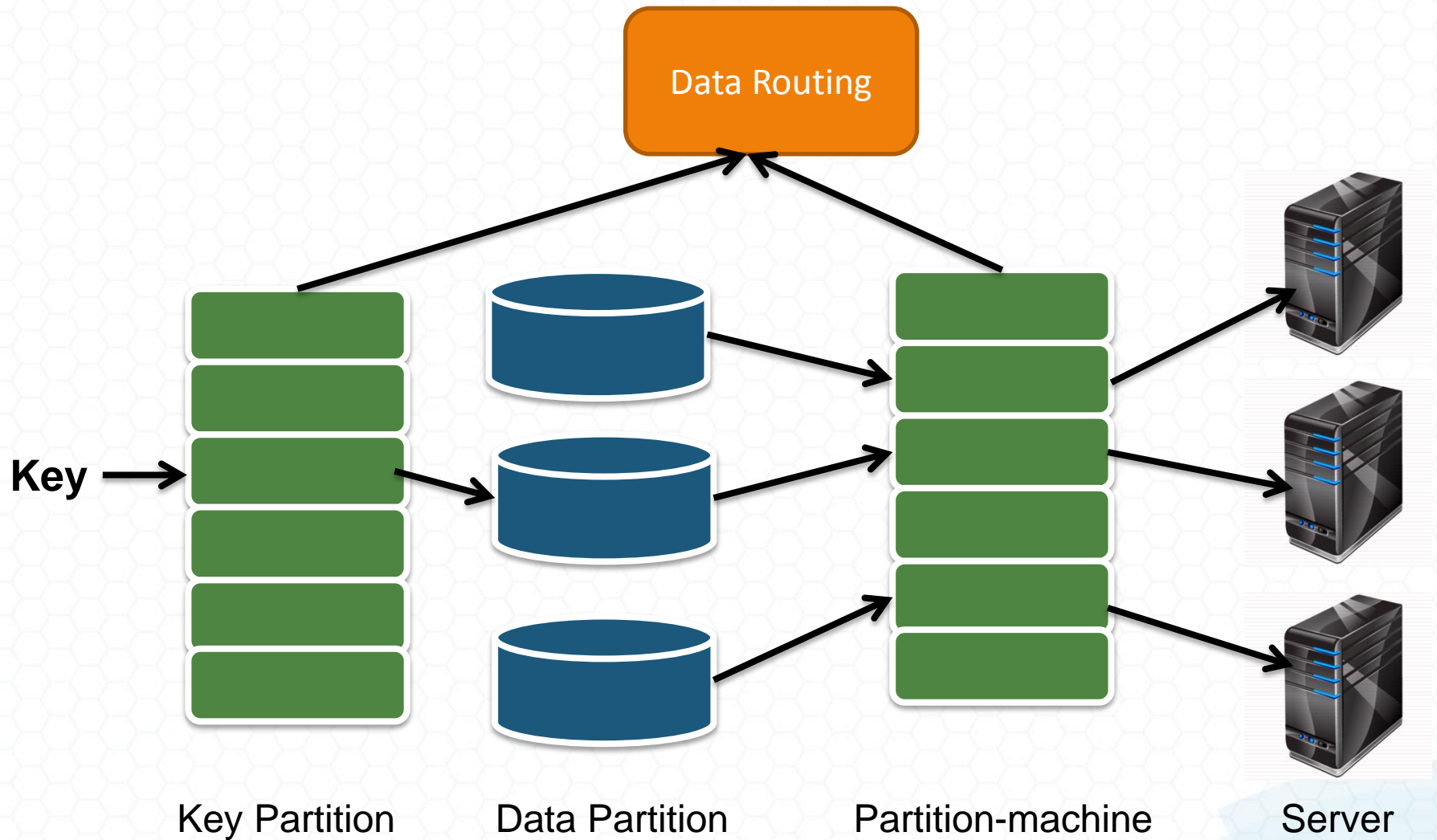
iThome



# 水平分割 (Horizontal Partition)



# 數據路由 – 水平分割





# 為什麼要用 NoSQL



# NoSQL的出現

## ■ 網路公司的崛起

- BigTable

- DynamoDB



## ■ Strozzi NoSQL

- 用Tab 分割資料

- 使用shell 操作資料



## ■ Johan Oskarsson舉辦 DynamoDB 與BigTable技 術討論聚會時所用的名詞

# NoSQL

- NoSQL (Not Only SQL)
- 或是指相對於關聯式資料庫而言(MySQL 或 PostgreSQL) 的非關聯式資料庫
- 特性
  - 可以將資料平行分散於叢集中
  - 不使用關聯性模型 (Schema Less)
  - 極具成本效益 (Cost Effective)
  - 開源 (Open Source)

# Visual Guide to NoSQL Systems

**Availability:**  
Each client can  
always read  
and write.

**A**

## Data Models

Relational (comparison)  
Key-Value  
Column-Oriented/Tabular  
Document-Oriented

**CA**

RDBMSs  
(MySQL,  
Postgres,  
etc)

Aster Data  
Greenplum  
Vertica

**AP**

Dynamo  
Voldemort  
Tokyo Cabinet  
KAI

Cassandra  
SimpleDB  
CouchDB  
Riak

Pick Two

**C**

**Consistency:**  
All clients always  
have the same view  
of the data.

**CP**

BigTable  
Hypertable  
Hbase

MongoDB  
Terrastore  
Scalaris

Berkeley DB  
MemcacheDB  
Redis

**P**

**Partition Tolerance:**  
The system works  
well despite physical  
network partitions.

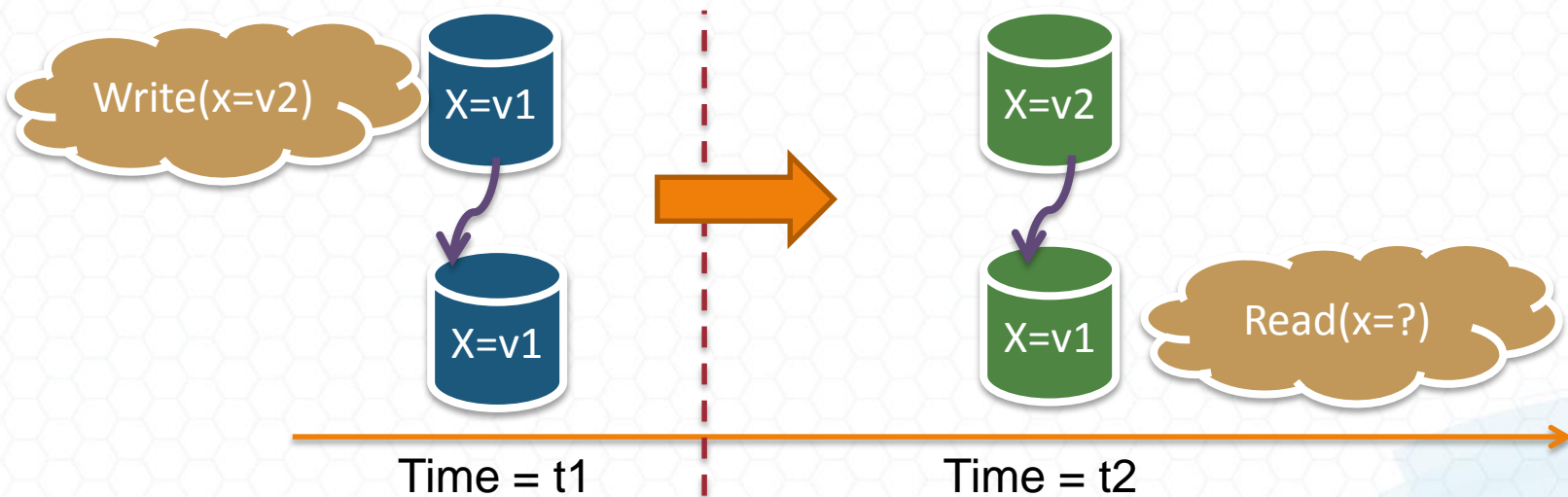
By Eric Brewer  
@ 1991

# CAP 滿足的情境

## ■ 情境一

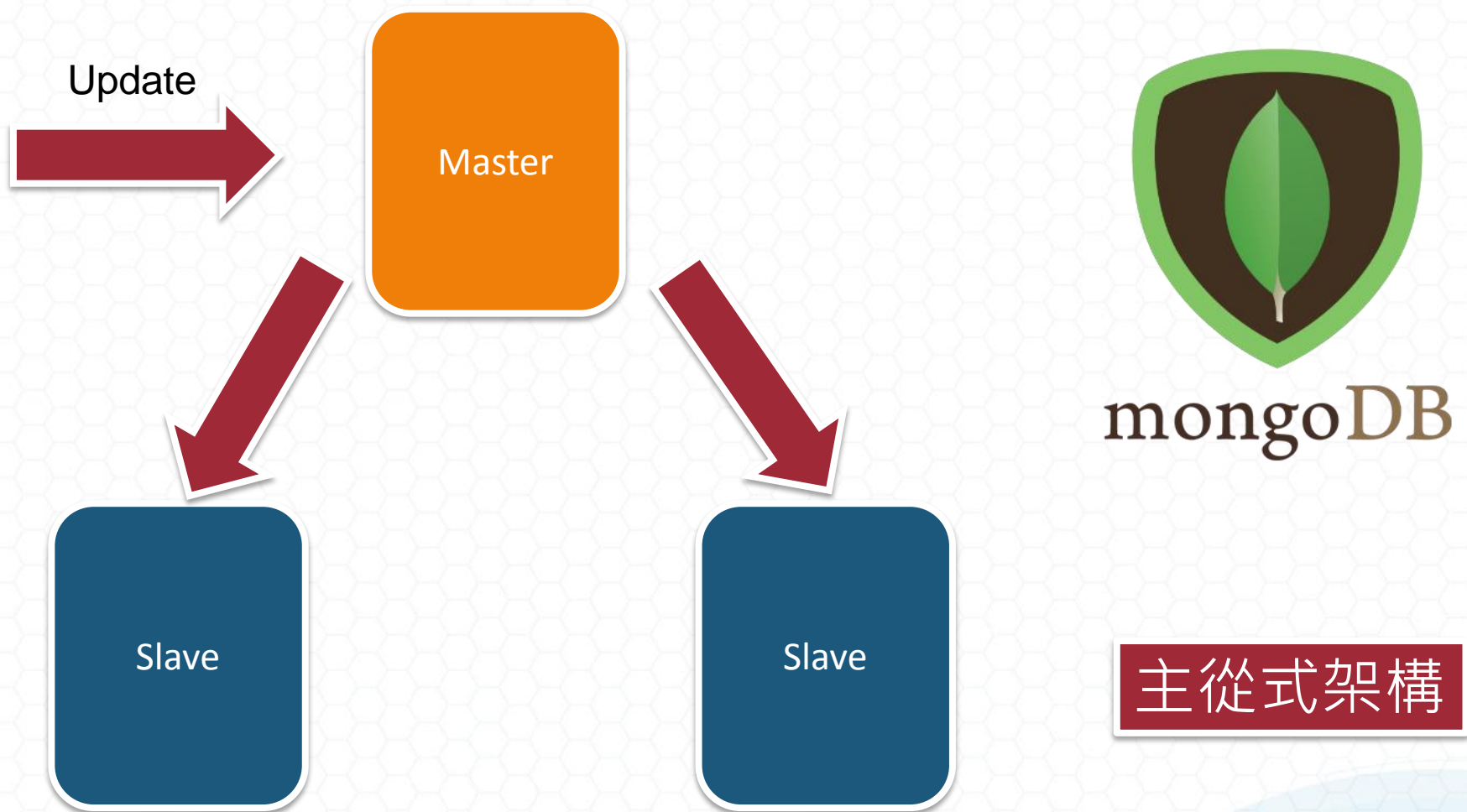
- 資料無副本 -> 必然滿足 C & P
- 網路無法連通或有機器當機時 -> 部分數據不可被訪問

## ■ 情境二

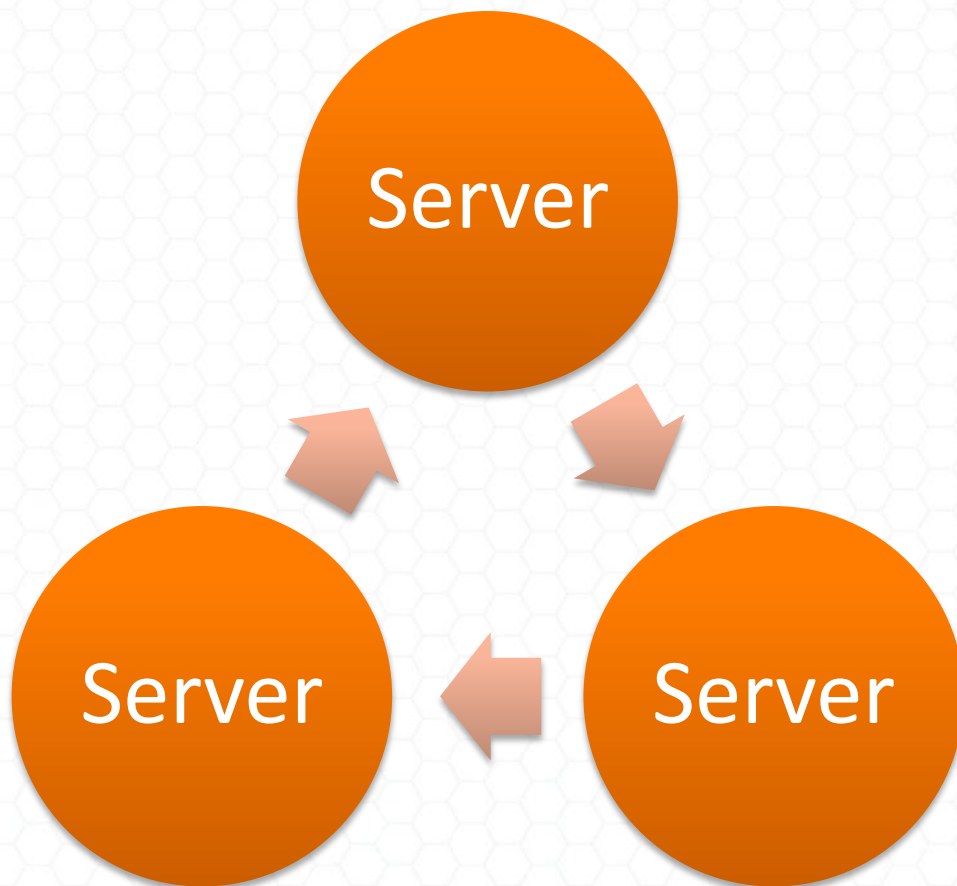




# 滿足CP的非關聯式案例



# 滿足AP的非關聯式案例



P2P架構

# 關聯式資料庫的設計模式 - ACID

## ■ ACID 原則

- ▣ **A**tomicity: 全部執行或全部不執行 (如銀行轉帳)
- ▣ **C**onsistency: 當設定了約束條件，交易始終必須滿足約束條件 (如餘額大於零等約束條件)
- ▣ **I**solation: 不同交易並行執行時，不影響互相狀態
- ▣ **D**urability: 交易完成或認可(Commit)後，狀態的變更是永久的

# BASE 原則

## ■ Basically Available

- 系統大部分屬於可用狀態，允許偶爾的失敗

## ■ Soft State

- 不要求數據在任意時刻都保持同步

## ■ Eventually Consistency

- 資料在一定時間內最終會一致



# NoSQL 的優點

- 自動水平分散資料 (Sharding)
  - 無綱要模型 (Schema Less)
  - 極具成本效益(Cost-Effective)
- 
- 適合用在資料一致性與完整性要求較低但量大且快速累積的資料



# 擴展方便性

## ■ 垂直擴展(Scale Up)

- 需要幫機器增添CPU, RAM 磁碟空間
- 擴展時需要啟用備援機器以避免服務中斷
- 必須使用SAN 等裝置才能分散儲存資料

## ■ 水平擴展 (Scale Out)

- 只需要加機器到叢集中
- 可使用標準化量產硬體(Commodity Hardware)
- 自動負載平衡

# 無綱要模型 (Schema-Less)

- 使用JSON 等Key-Value 模型表示資料
- 不用顧慮到資料綱要變動時，是否要去修改資料表綱要
- 可以表示概念不同的異質資料 (e.g. 電視與電腦)

```
{
  "data": [
    {
      "id": "10153600155794890",
      "created_time": "2015-09-16T13:12:06+0000",
      "place": {
        "id": "164657790411032",
        "location": {
          "city": "Taipei",
          "country": "Taiwan",
          "latitude": 25.047789030864,
          "longitude": 121.5171581701,
          "street": "中正區北平西路三號二樓"
        },
        "name": "長榮桂冠葡萄酒坊台北車站店"
      }
    },
    {
      "id": "10153472862224890",
      "created_time": "2015-07-26T14:50:22+0000",
      "place": {
        "id": "112196945506136",
        "location": {
          "city": "Sungshan",
          "country": "Taiwan",
          "latitude": 25.050769444444,
          "located_in": "110850655605422",
          "longitude": 121.54993888889,
          "street": "臺北市松山區南京東路4段2號",
          "zip": "10553"
        }
      }
    }
  ]
}
```

# 極具成本效益 (Cost-Effective)

## ■ 擴展成本

- 只需要加機器便可完成水平擴展

## ■ 管理成本

- 管理上較不複雜 (缺乏索引、交易等資料庫複雜設計)

## ■ 硬體成本

- 可搭載在量產化硬體機器上



A history of databases in No-tation

1970: NoSQL = We have no SQL

1980: NoSQL = Know SQL

2000: NoSQL = No SQL!

2005: NoSQL = Not only SQL

2013: NoSQL = No, SQL!

(R)DB(MS)



SAMSUNG

# NoSQL 應與關聯式資料庫機制並行

- Linear Scalability
- Schema flexibility
- High Performance



NoSQL

- Multi-document transactions
- Complex security needs
- Complex joins
- Extreme compression needs



RDBMS

- Both / depends on the data

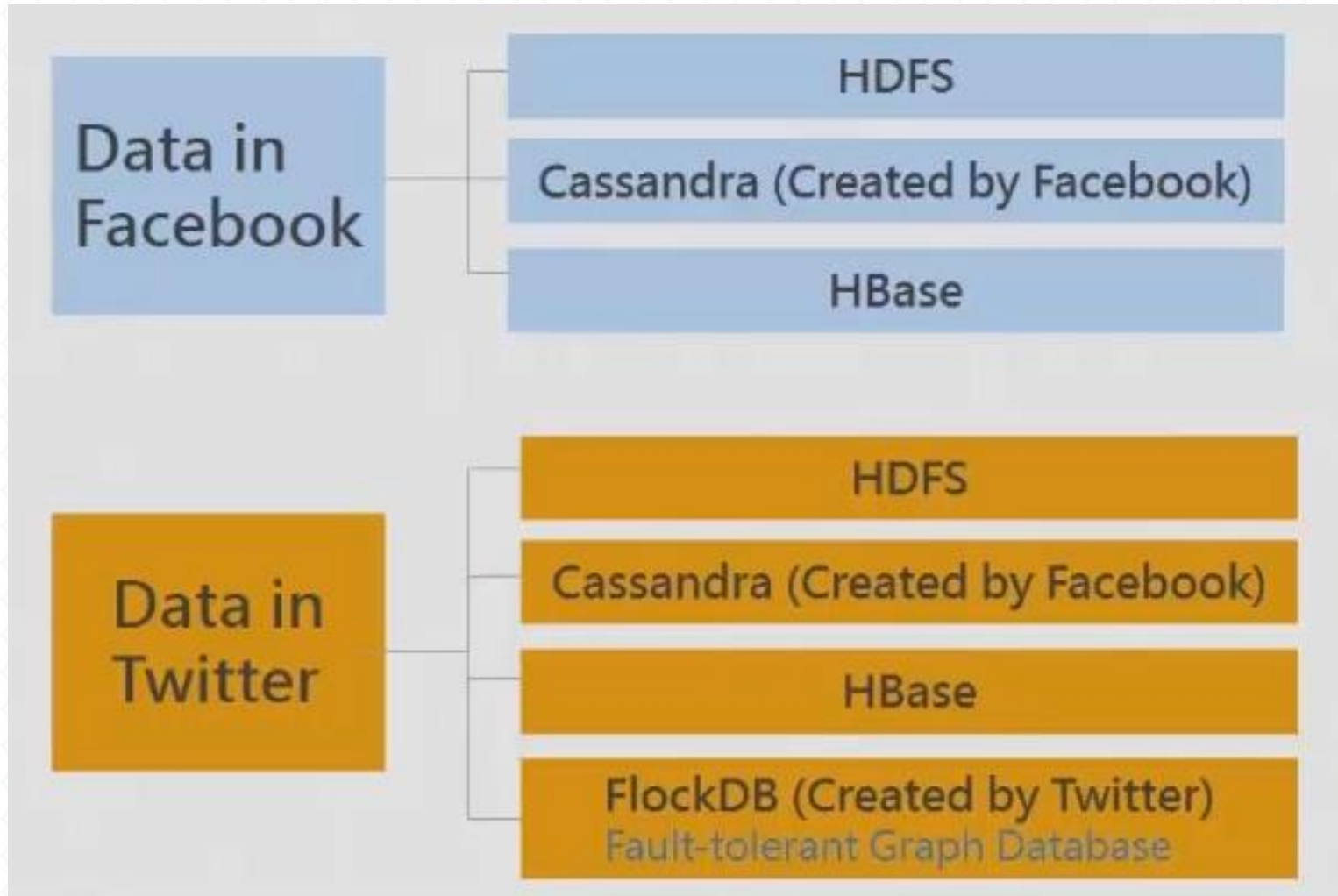


RDBMS



NoSQL

# NoSQL在不同公司的應用情境

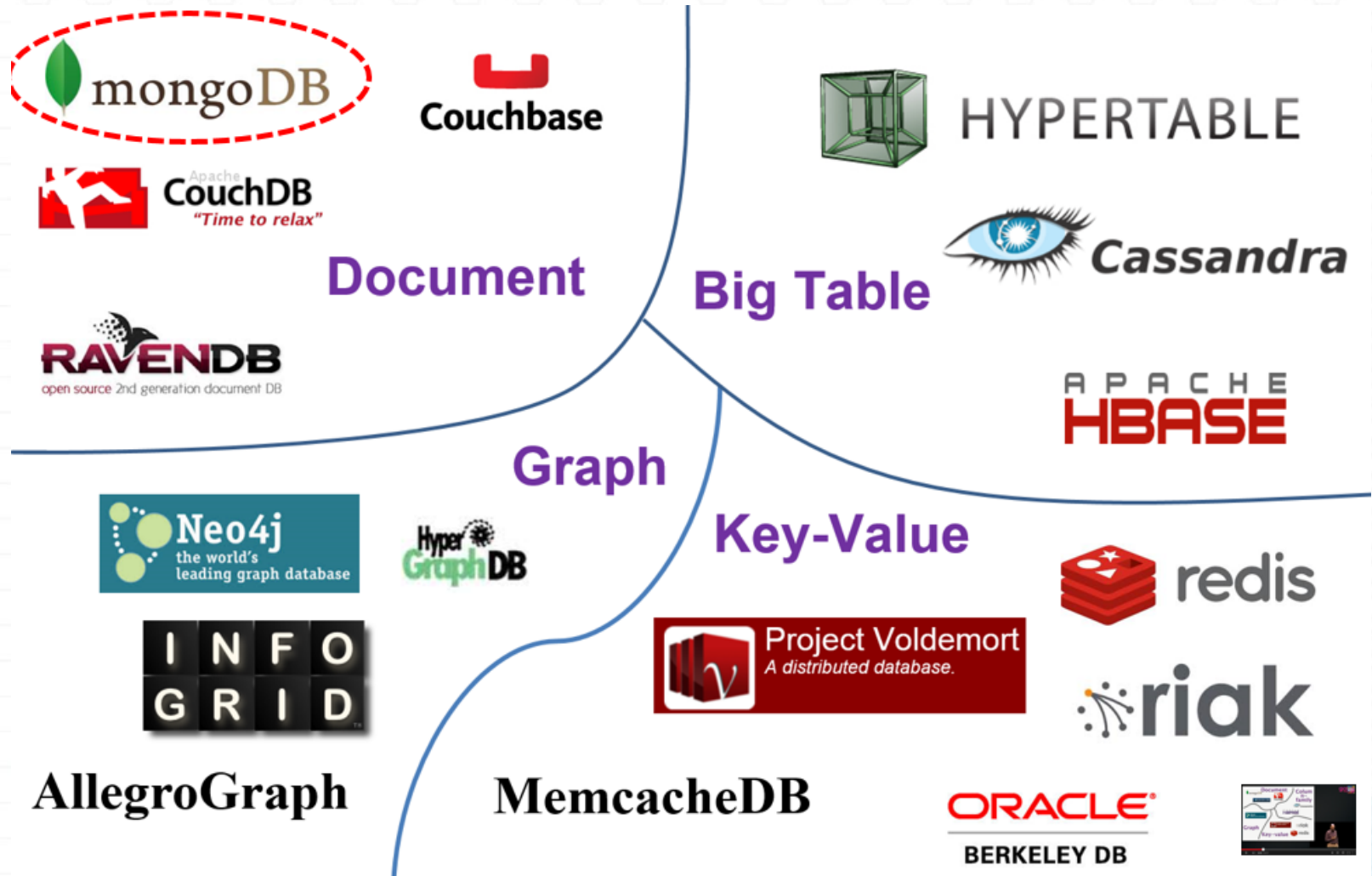




# NoSQL的種類



# NoSQL 種類



# Facebook 的儲存技術

- Facebook 用關聯式資料庫與NoSQL儲存資料
  - ▣ MySQL：儲存會員資料
  - ▣ Memcached: 資料庫快取
  - ▣ Cassandra: 儲存站內信箱資訊
  - ▣ HBase：訊息的儲存，建立訊息的反向索引



# NoSQL 種類

- Key/Value型 – 以Key/Value存放資料
  - Memcached
  - Amazon DynamoDB
- Column-based型 – 列式資料庫
  - BigTable
  - Cassandra
- Document-based – 以XML、JSON等文件形式保存資料
  - MongoDB ,
  - CouchDB

# Key-value

Key	Value
"India"	{"B-25, Sector-58, Noida, India – 201301"}
"Romania"	{"IMPS Moara Business Center, Buftea No. 1, Cluj-Napoca, 400606", City Business Center, Coriolan Brediceanu No. 10, Building B, Timisoara, 300011"}
"US"	{"3975 Fair Ridge Drive. Suite 200 South, Fairfax, VA 22033"}



# key-value 資料庫

## ■ 適合

- 儲存session
- 使用者檔案與喜好設定
- 購物車資料

## ■ 不適合

- 資料間關係
- 多動作交易
- 由資料做查詢
- 一組的動作



電商的購物車資料

# Redis

## ■ **RE**remote **DI**ctionary **S**erver(Redis)

### ■ 將資料存放在記憶體中

- 如同Memcached



### ■ 以Key-Value作為儲存形態

- 一個Key值所儲存的Value可以是字串(strings), 雜湊(hashes), 清單(lists), 資料集(sets)與可排序資料集(sorted sets)。

# Redis v.s. Memcached

## ■ 資料類型

- Redis支援更豐富的資料類型

## ■ 物件大小

- Redis支援的物件大小最大支援1GB，而Memcached僅為1MB

## ■ 分片(Sharding)

- 可以將資料離散地存儲在不同的物理機器上

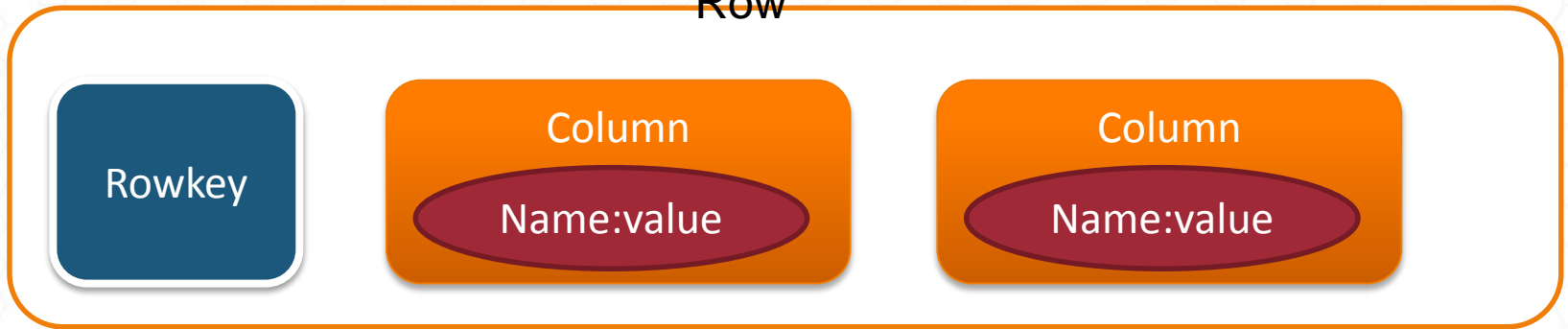
## ■ 持久化

- Redis能夠將記憶體中的資料持久化到磁片，而Memcached則只能用做功能有限的緩存中介

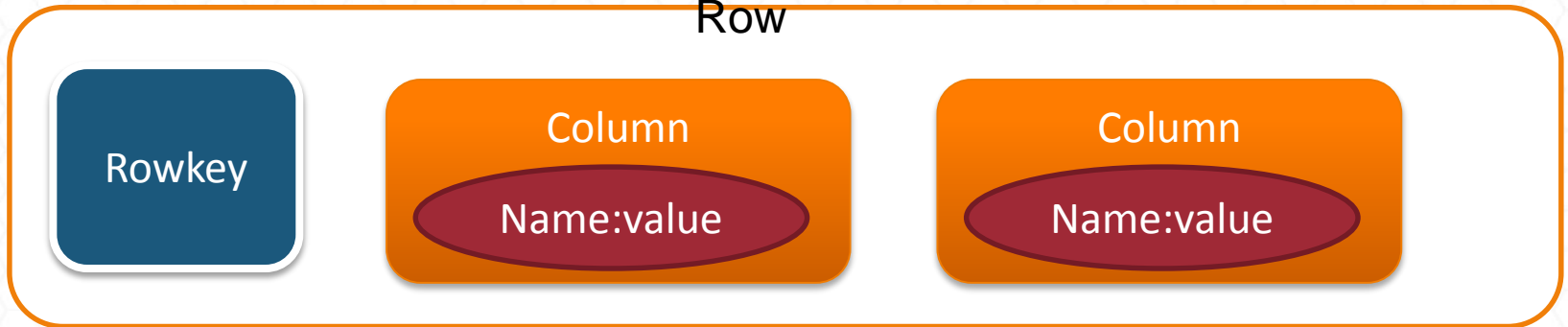
# Column-based NoSQL 資料庫

Column Family

Row

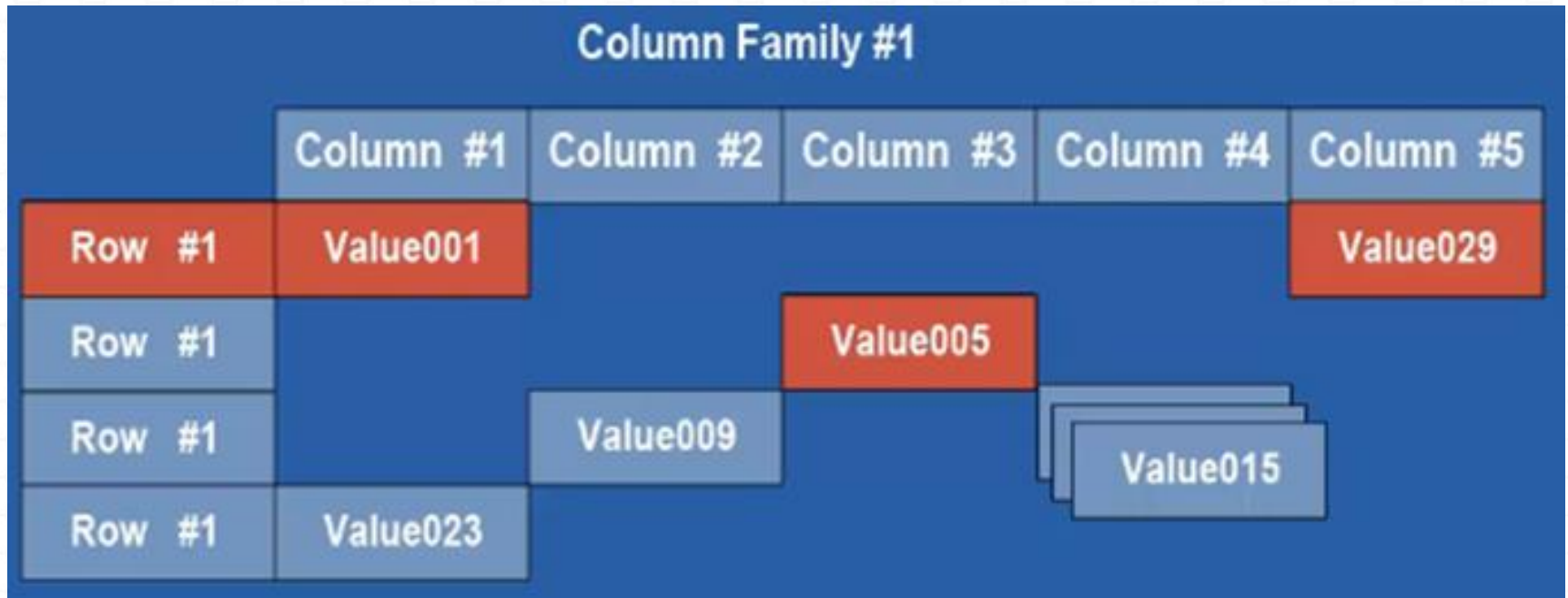


Row





# Schema-Less 資料儲存概念



# Column-based NoSQL 資料庫

```
{
  3PillarNoida: {
    address: { city: Noida, pincode: 201301},
    details: { strength: 250, projects: 20}
  },
  3PillarCluj: {
    address: { city: Cluj, pincode: 400606},
    details: { strength: 200, projects: 15}
  },
  3PillarTimisoara: {
    address: { city: Timisoara, pincode: 300011},
    details: { strength: 150, projects: 10}
  },
  3PillarFairfax: {
    address: { city: Fairfax, pincode: VA 22033},
    details: { strength: 100, projects: 5}
  }
}
```

City	Pincode	Strength	Project
Noida	201301	250	20
Cluj	400606	200	15
Timisoara	300011	150	10
Fairfax	VA 22033	100	5

• 3PillarNoida, 3PillarCluj, 3PillarTimisoara, 3PillarFairfax 為資料鍵 (Key) 相當於表格列

- address, details 為 column families
- address 包含欄位 city, pincode
- details 包含欄位 strength, projects

# Column-based 資料庫

## ■ 適合

- 事件歷史紀錄
- 內容平台
- 計數器
- 限時間使用的資料



## ■ 不適合

- 需要ACID 的系統

Facebook 訊息資料

# Cassandra



- 由Facebook開發
- 集Google BigTable的資料模型與Amazon Dynamo的完全分散式的架構於一身
- 列被組織成為列族（ Column Family ），在資料庫中增加一列非常方便
- 基於P2P架構，與傳統的基於分片的資料庫集群相比，Cassandra可以幾乎無縫地加入或刪除節點



# Cassandra 特點

## ■ 模式靈活

- 使用Cassandra，像文檔存儲，你不必提前解決記錄中的欄位。你可以在系統運行時隨意的添加或移除欄位

## ■ 真正的可擴展性

- 當需要替集群添加更多容量，可以指向另一台電腦。不必重啟任何機器，改變應用查詢，或遷移任何資料

## ■ 多資料中心識別

- 可以調整你的節點佈局來避免某一個資料中心發生意外

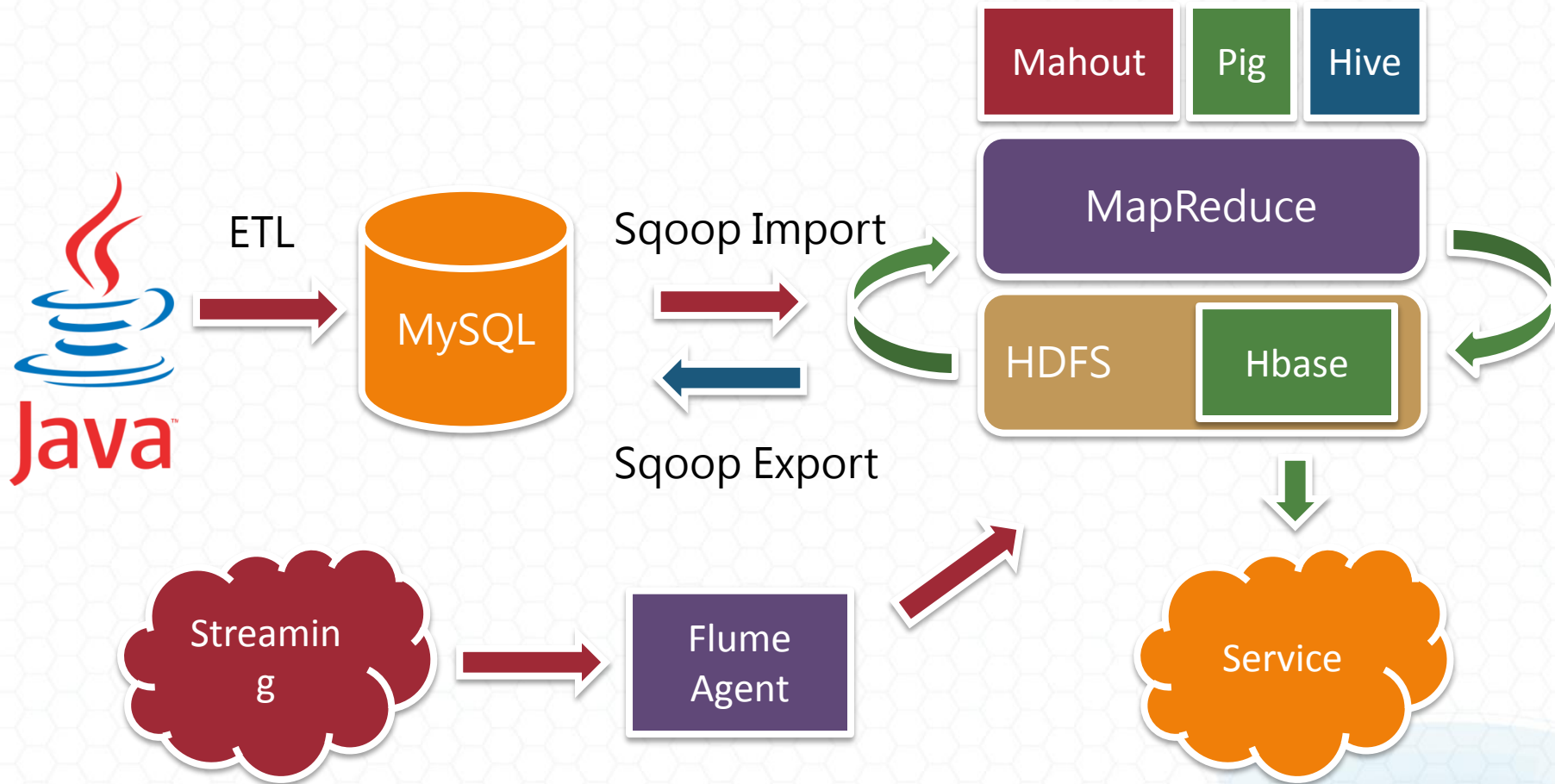
# Hbase

- 分散式
- 多維度
- 高效能
- 存儲系統
- High- Availability
- Column- Oriented

存儲大量的行、列及多版本資料，並能分散到數以萬計的機器中



# 建立推薦系統



# Document Database

- 將資料以“檔案”(XML, JSON)形式儲存
- 可以透過半結構化語法存取內容

```
1 {officeName:"3Pillar Noida",  
2 {Street: "B-25, City:"Noida", State:"UP", Pincode:"201301"}  
3 }  
4 {officeName:"3Pillar Timisoara",  
5 {Boulevard:"Coriolan Brediceanu No. 10", Block:"B, Ist Floor", City: "Timisoara", Pinc  
6 }  
7 {officeName:"3Pillar Cluj",  
8 {Latitude:"40.748328", Longitude:"-73.985560"}  
9 }
```



# Document Database

■ 適合

- 事件歷史紀錄
- 內容平台
- 電子商務應用程式
- 網路點擊資料

■ 不適合

- 不同動作的複雜交易
- 不同結構的查詢



## 使用者點擊資料蒐集

# MongoDB

## ■ 使用BSON的格式來存取資料-

- Bson = binary json
- bson也支援較Json 更多的data type

## ■ GridFS

- 存儲資料的chunks與metadata
- 當存取資料時就可以快速的到達資料存放的位置

## ■ 支援分散式資料庫

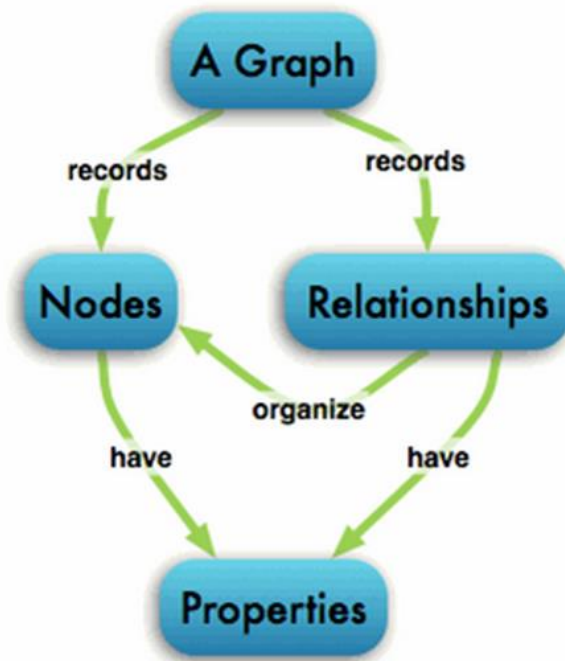
- 可以把資料庫放在不同主機
- 自動分散資料負載 (Sharding, Rebalancing)



mongoDB

# GraphDB

- 儲存、搜尋物品與人際關係
- 每個人可視為一個點，每條線視為之間的關係





# GraphDB

## ■ 適合

- 連結的資料
- LBS 服務
- 推薦引擎

## ■ 不適合

- 大量log 資料



儲存人與人之間的關係



# Neo4j

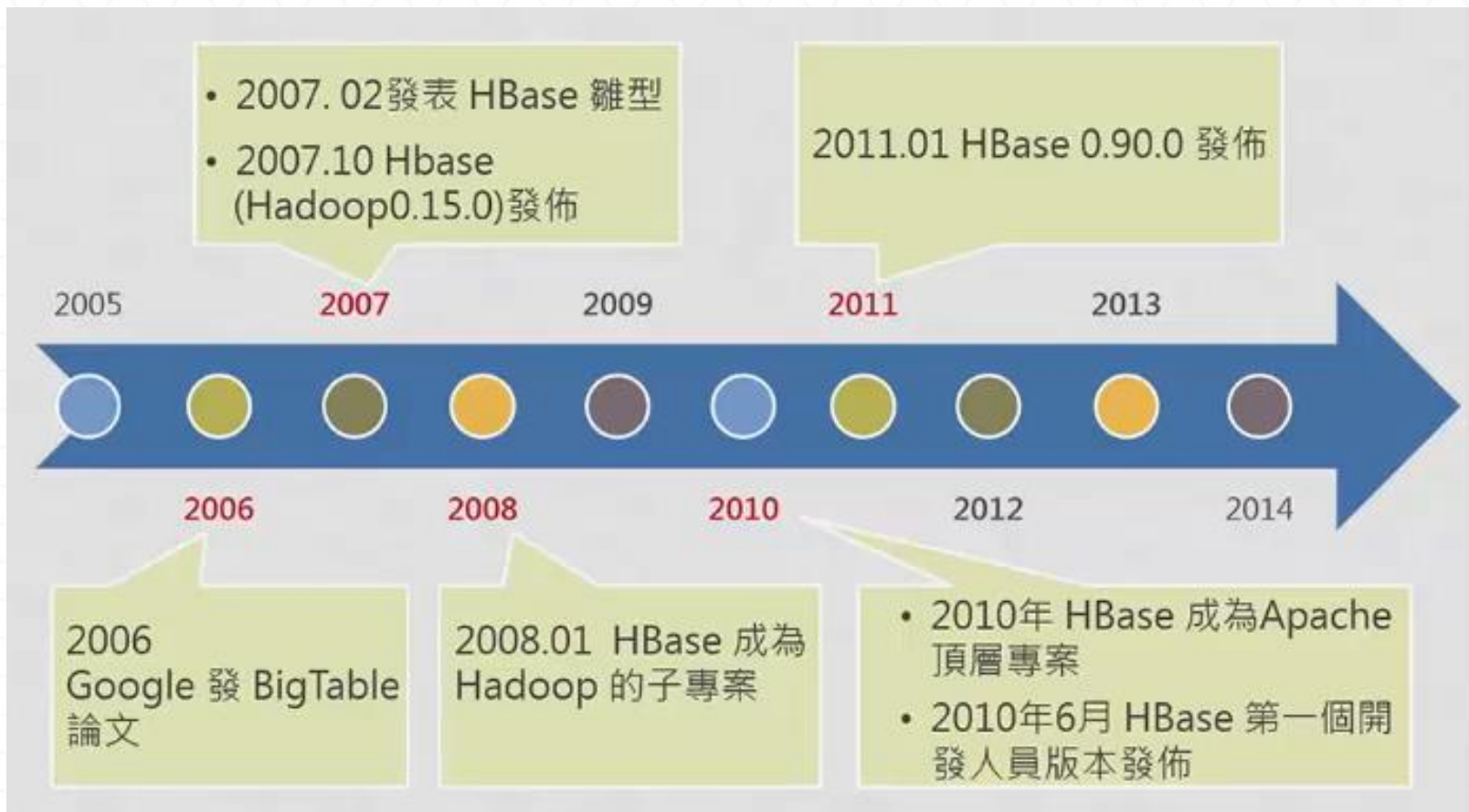
- 圖形資料庫系統 (Graph Database)
- 主要使用 Java 程式語言撰寫
- 可以針對複雜的「關係」(Relationship) 進行設計與分析
- 解決傳統關聯式資料庫難以處理的關係搜尋



# Hbase 簡介

# HBase 歷史

## ■ Hbase 1.0 於 2015 年發表



# Hbase

- 根據Google 2006年 BigTable 論文實作
- 分散式、多維度、高效能的存儲系統
- 存數十億列、數百萬欄
- High- Availability
- Column- Oriented

存儲大量的行、列及多版本資料，並能分散到數以萬計的機器中

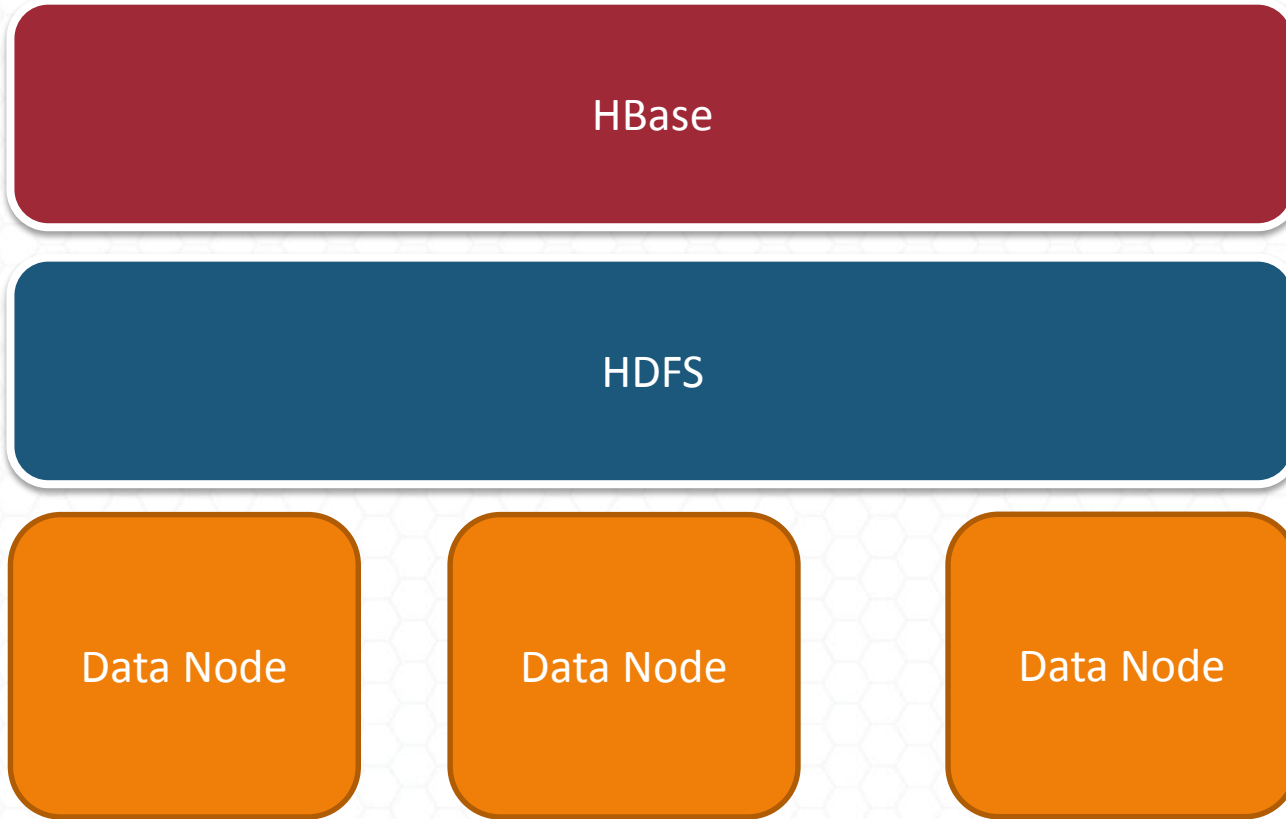




# Hbase 與RDBMS 的差異

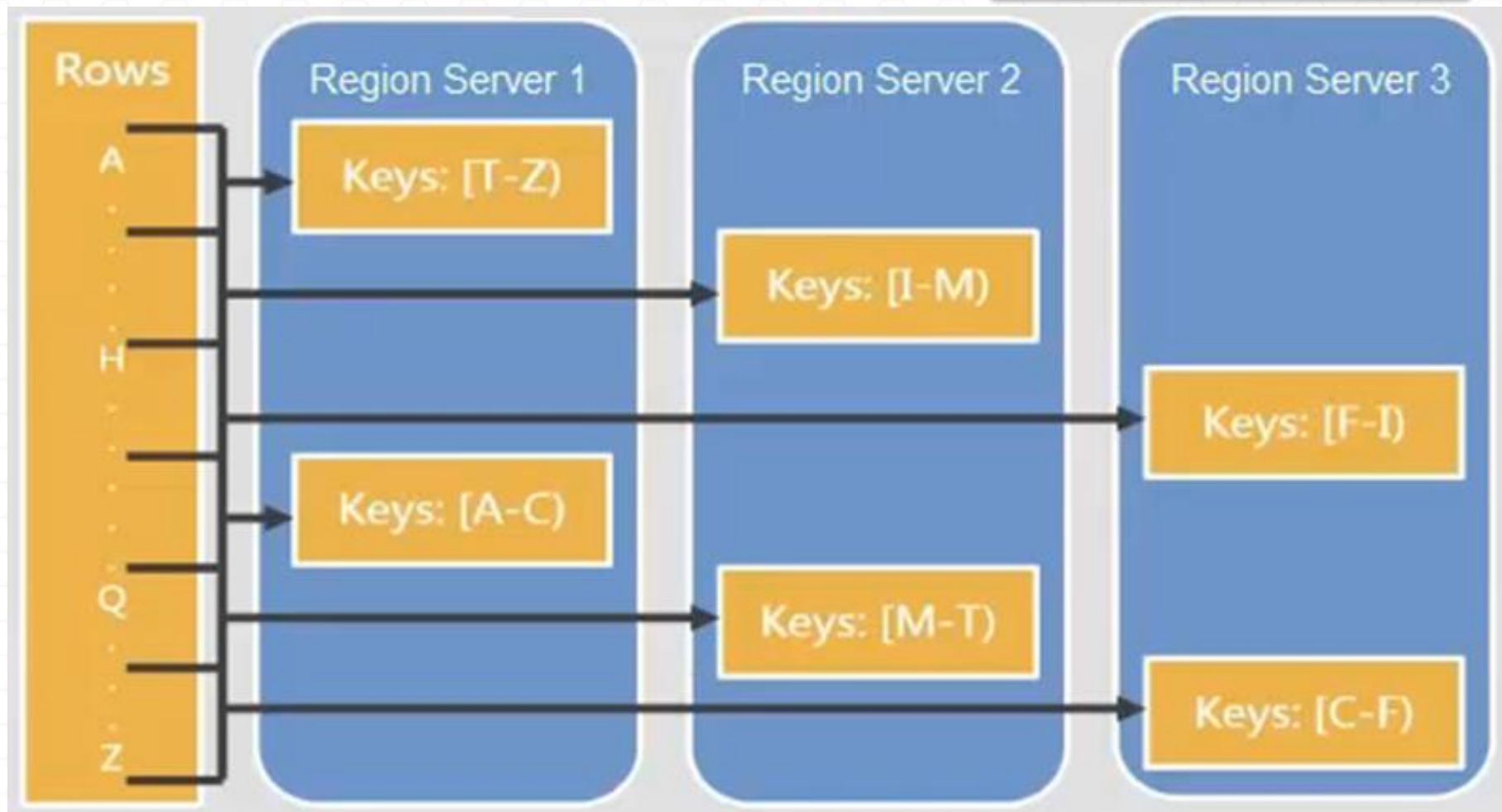
- 並非關聯式資料庫
  - 沒有Join, Query engine, 類型, SQL
  - 有支援Transaction 跟 Secondary Index – 發展中
- 並非傳統資料庫的替代品
- 沒有Schema (Anti-Schema)的概念
  - 資料是非正規化的
  - 超大型Excel

# HBase 架構



# 資料存放模式

資料是根據字典順序擺放

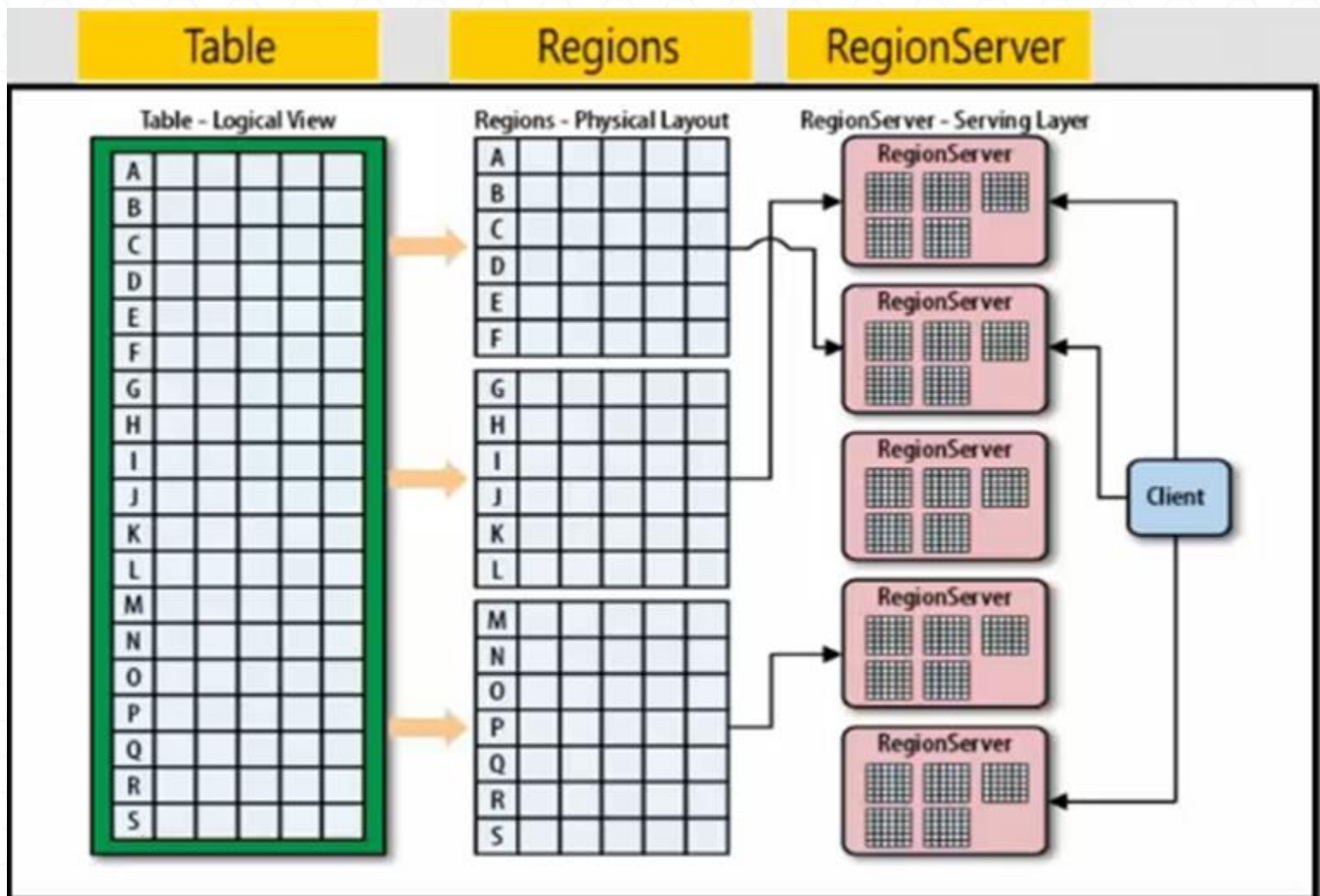


# Hbase 架構

- 表格是由Regions 所組成
- Region 是由startKey 與endKey 所定義的
  - 空白表格:
    - Table, NULL, NULL
  - Two-region table
    - (Table, NULL, “streamy”) and (Table, “Streamy”, NULL)
- 每個Region是由多個HDFS 檔案與Block所組成，同時存在於多台node 上



# Table, Region, Region Server



一個Region Server 所存放的Region 可能來自不同的Table  
由Master 決定哪台Region Server 擺放哪些Regions

# HBase 硬體與元件

# 硬體配置



Region Server 存放資料

# HBase - HMaster

- 分配 Region Server 管理的 Region
- 做為 Region Server 的 Load Balancer
- 偵測故障的Region Server, 並將其Region 重新分配到其他Region Server
  - 重新分配時系統會暫時Hang 住
- 更新表格的Table Schema



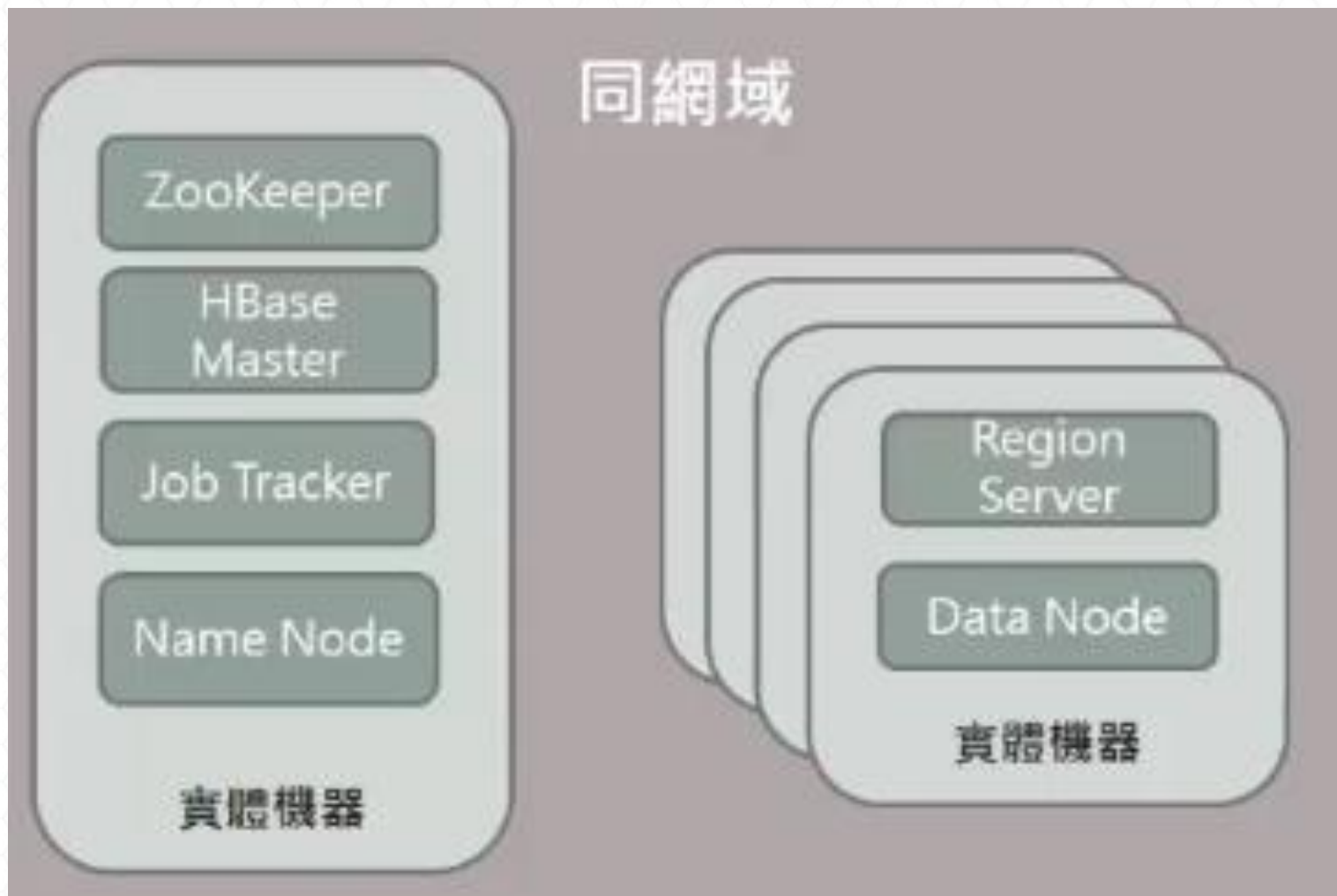
# Region Server

- 運行在 Data Node 上
- 維護Master 分配的 Region，處理對Region 的 IO 需求
- 切割在運行過程中儲存空間超過門檻值的Region
  - 會警示Master 是否存放Region 過大，請示Master 裁切資料

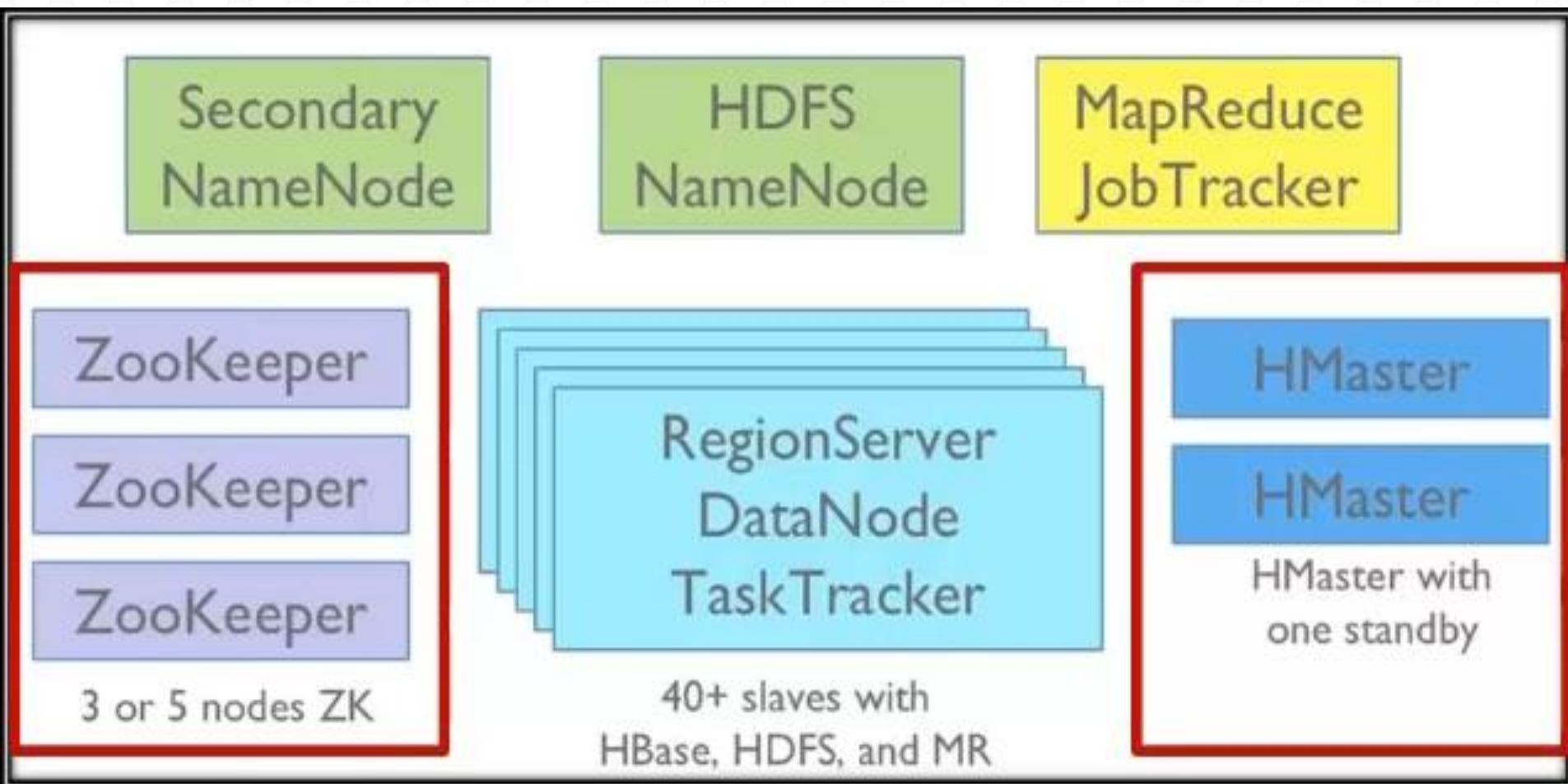
# Zookeeper

- Google 的Message Queue (Chubby)
  - 分散式系統協調機制
- 確保在任何時候都只有一個Master
  - 使用投票機制
- 儲存Region 的 Mapping 機制
  - 可跳過Master 存取到Region Server 的內容
- 儲存Hbase 的 Schema, 包含Table 以及Table 有哪些Column Family
- 監控Region Server 的狀況，告知Master 關於Region Server 的啟動與斷線
- 所有的服務及HeartBeat 都交由Zookeeper 管理與協調

# 小型叢集佈署

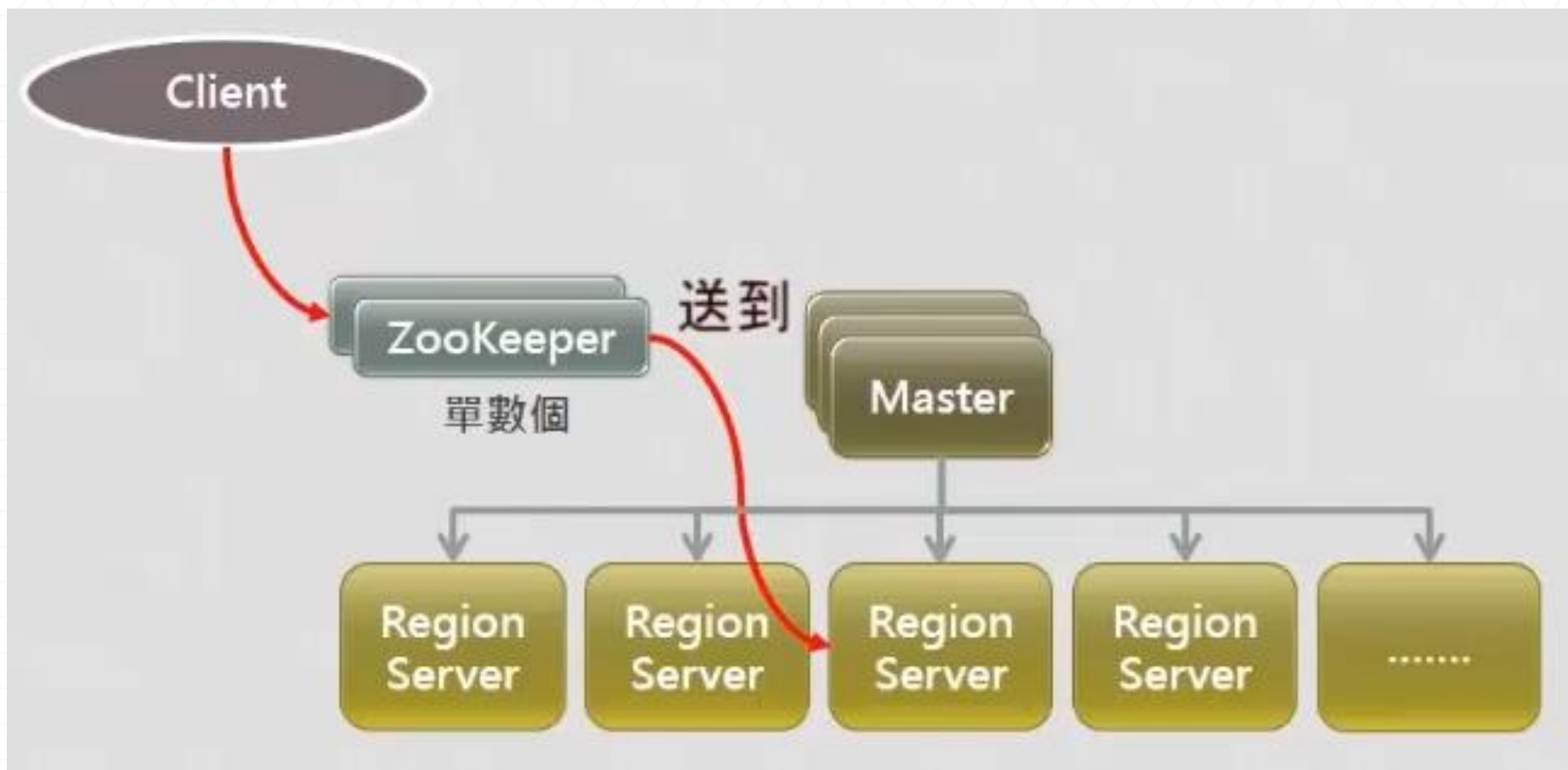


# 叢集配置



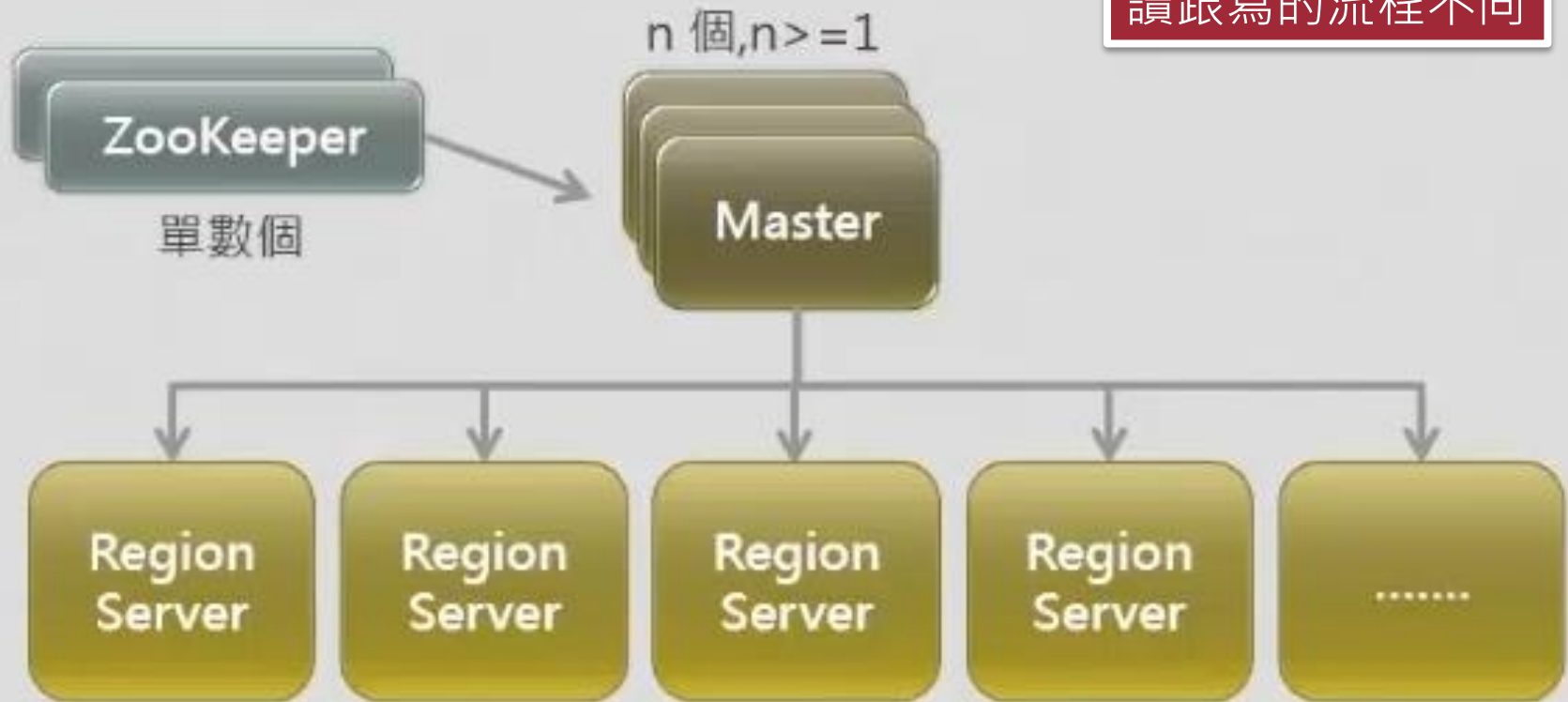


# 資料讀取



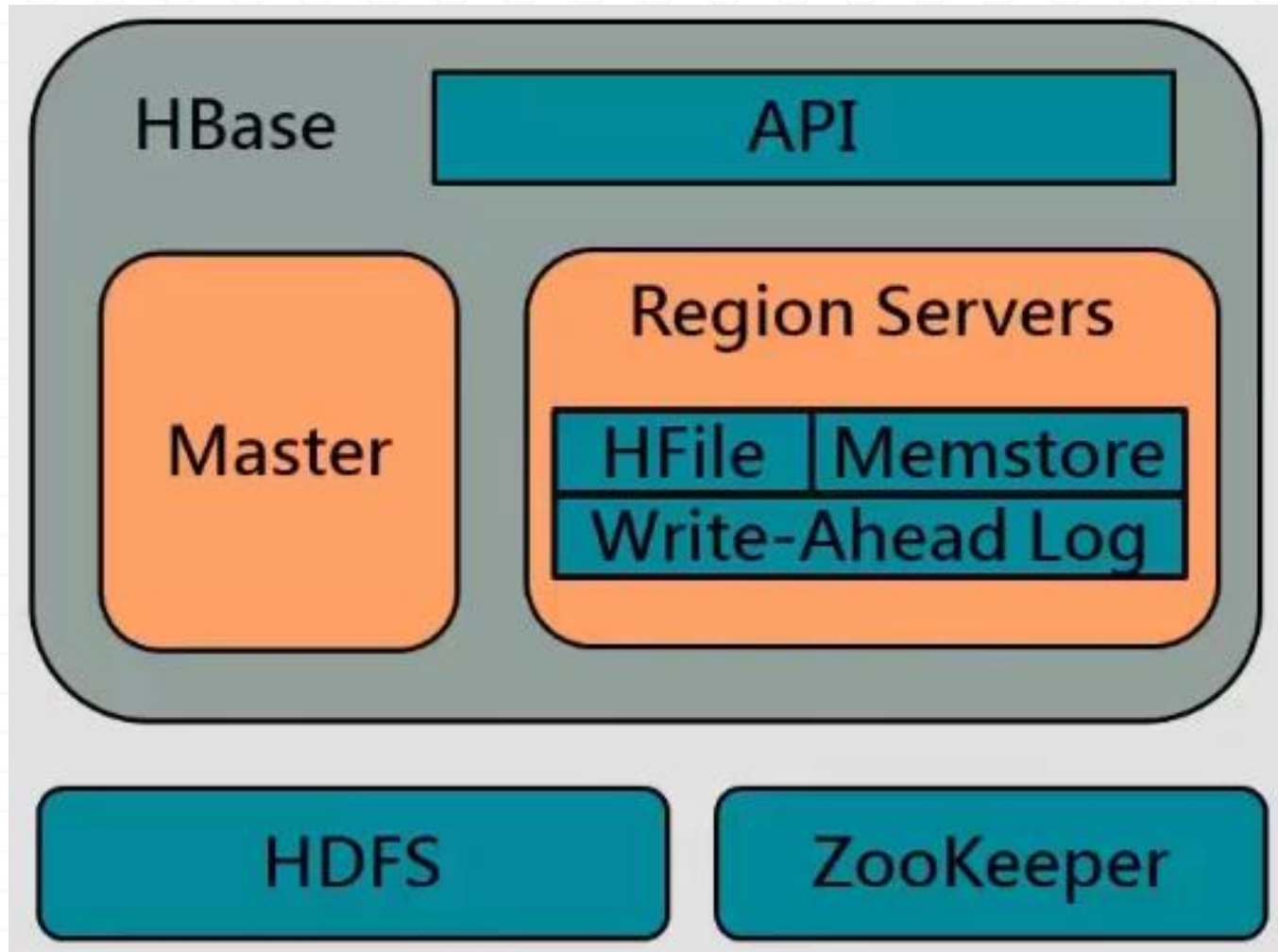
# 資料寫入

讀跟寫的流程不同



# HBase 資料存儲

# Hbase 架構





# Hbase 架構

- HMaster**
- Responsible for Admin Operations
  - Manages and Monitors the Cluster
  - Assigns Regions to Region Servers
  - Controls Load Balancing and Failover

## HRegionServer

### HRegion

HLog

Store  
MemStore  
StoreFile

## HRegionServer

### HRegion

HLog

Store  
MemStore  
StoreFile

## HRegionServer

### HRegion

HLog

Store  
MemStore  
StoreFile

## HRegionServer

### HRegion

HLog

Store  
MemStore  
StoreFile

# Flushing & Compaction, Splitting

## ■ Flushing

- 大量資料寫入時將記憶體的资料寫到新的檔案
- WAL 會 Archive

## ■ Compaction

- 將檔案合併，將Flushing 產生的檔案合併

## ■ Splitting

- 檔案分割(對切)，將資料寫到另一個Region Server (會將Region Server 下線)



# Compaction

- 合併多個Hfile 成一個Hfile
- 兩種類型
  - Major Compaction (完整文件合併)
  - 刪除過期與已刪除的Data
  - 一個Store 只會有一 Store File
  - Minor Compaction(部分文件合併)

# Compaction 優點

- 減少Hfile 個數 -> 減少IO
- 提升效能
- 刪除過期與已刪除的資料 -> 增加空間



# Splitting

## ■ 優點

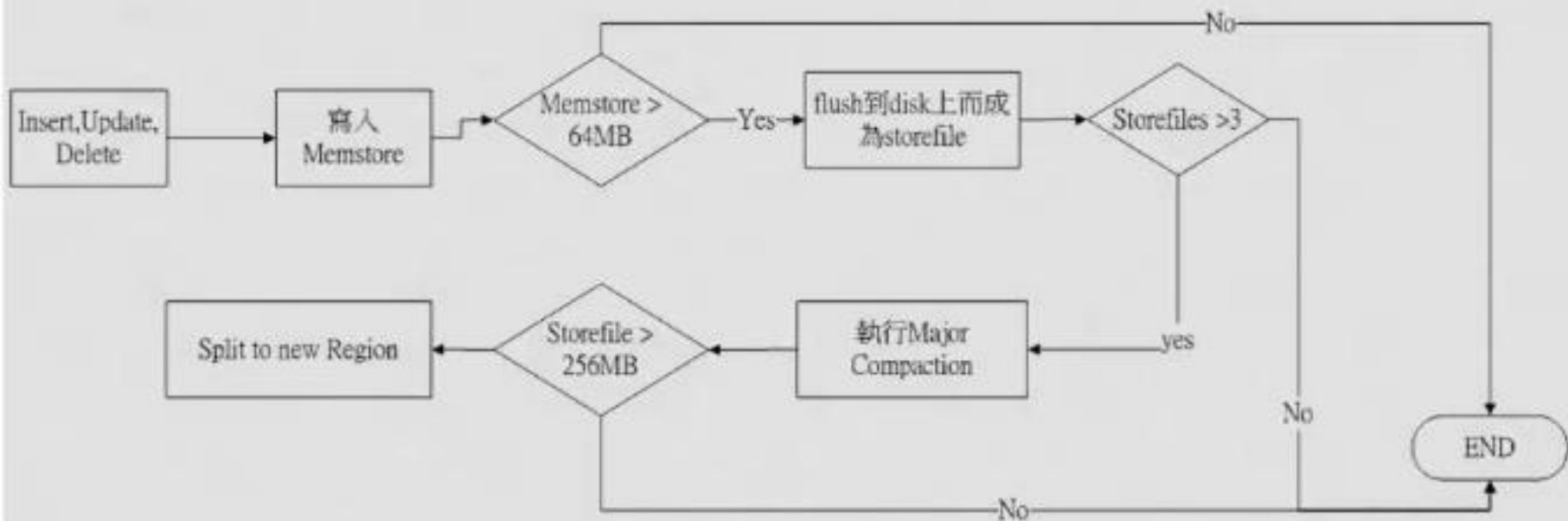
- 減少 HFile 檔案大小
- 提高分散使用率

## ■ 缺點:

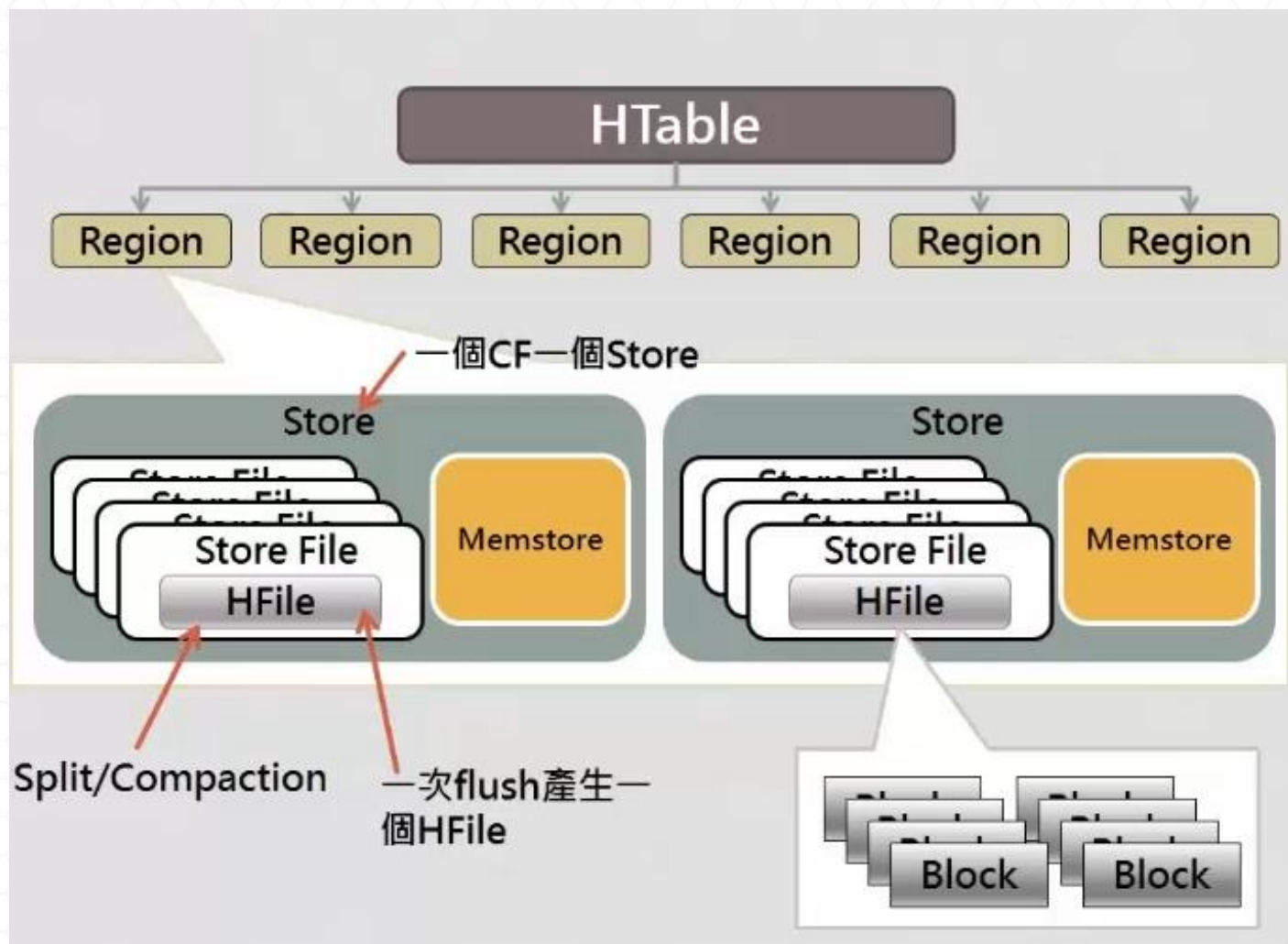
- Region Server Downtime (兩個Region Server Down)
- 檔案數變多 => IO 變多

# 資料寫入流程

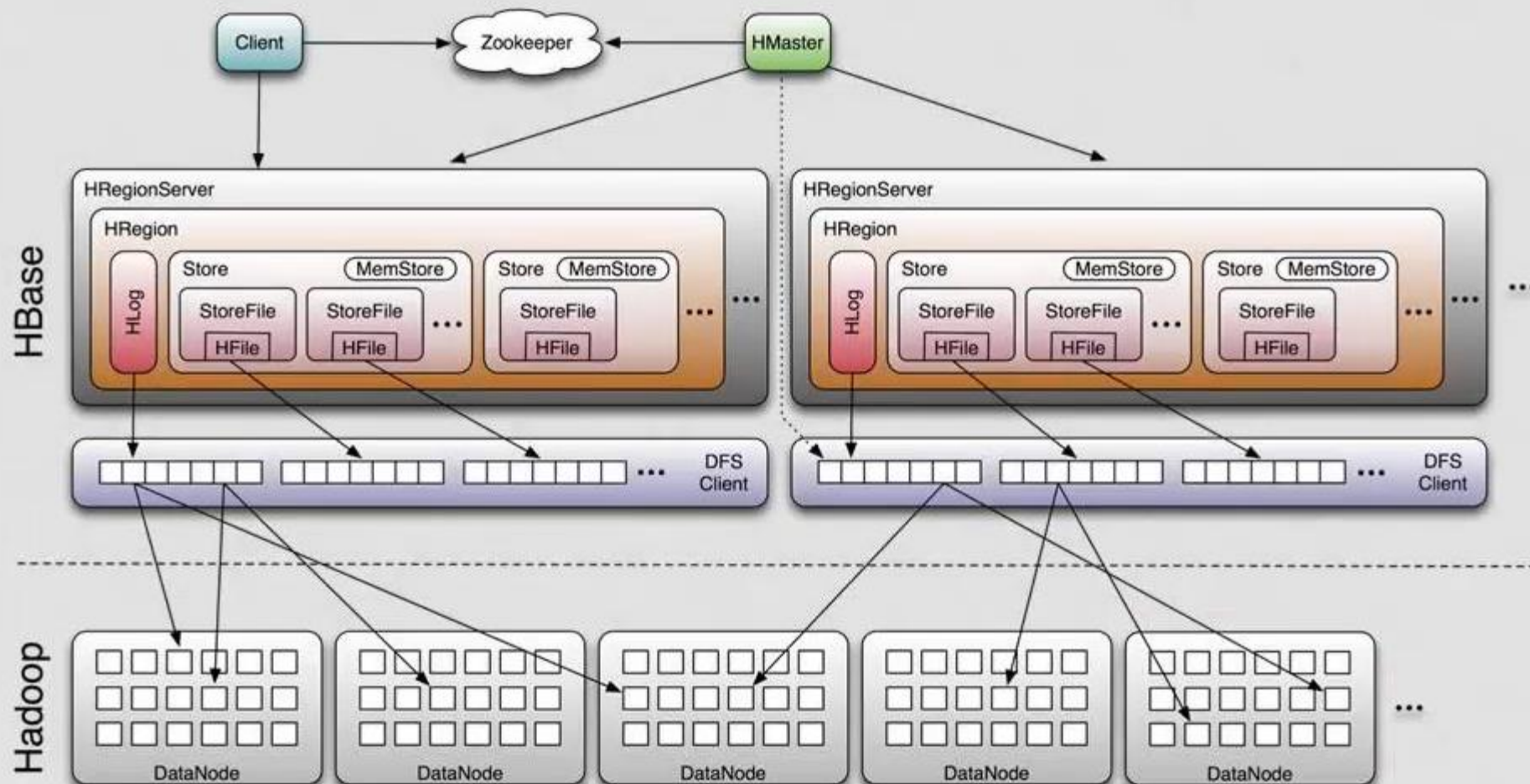
- HBase中Hfile的默認最大值(hbase.hregion.max.filesize)是256MB



# 存儲模式運作



# 完整HBase 架構圖





# HBase 存儲模式

# 資料存放架構

## ■ Table (HBase Table)

### □ Region (Regions for the Table)

- Store = Column Family
- MemStore
- StoreFile
  - ✓ Block

# 資料結構

## • Table Terms

- Table
- Column Family
- Row
- Qualifier(Column)
- Cell

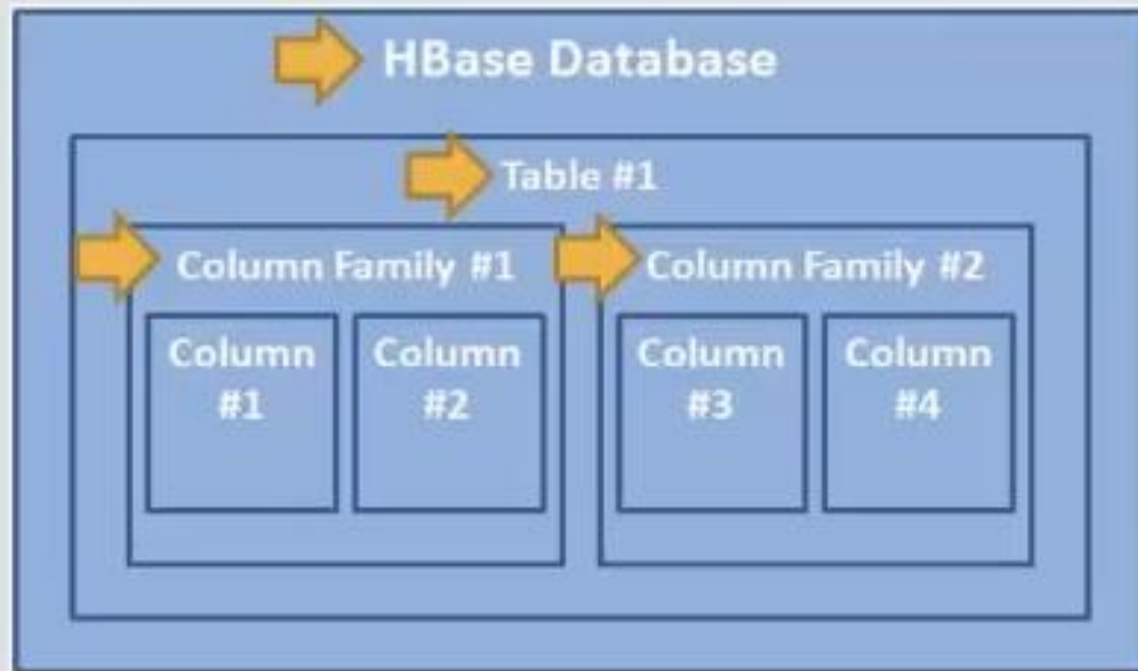


Figure 1 - HBase Data Organization

# 資料結構

Column Family

HR					
Person		Contact Info			
rowkey	Name	Gender	TEL	CELL	Address
A000001	Hubert	男	03-5630345	0933123456	
A000002	Mike	男	03-5630345	0932789456	
A000003	Rebecca	女	03-5630345	0920456789	

Qualifier

Row Key



# 資料結構

## HBase Database

### Table #N

### Table #2

### Table #1

#### Column Family #2

#### Column Family #1

	Column #1	Column #2	Column #3	Column #4	Column #5
Row #1	Value001				Value029
Row #1			Value005		
Row #1		Value009			
Row #1	Value023				

# 資料儲存範例

Row Key	Customer		Sales	
Customer Id	Name	City	Product	Amount
101	John White	Los Angeles, CA	Chairs	\$400.00
102	Jane Brown	Atlanta, GA	Lamps	\$200.00
103	Bill Green	Pittsburgh, PA	Desk	\$500.00
104	Jack Black	St. Louis, MO	Bed	\$1600.00

Column Families

# Hbase 表格

- 所以表格是依據Row 做排序
- 表格的Schema 會定義Column Family
  - 每個Family 都會有任意數量的欄位
  - 每個欄位都會有不同版本
  - 欄位只有在塞資料的時候才存在
  - 同個Family 的欄位會同時被排序以及存儲
  - 所有的欄位與定義都以byte[]存儲
- (Table, Row, Family:Column, Timestamp) -> value

# Hbase 表格資料結構

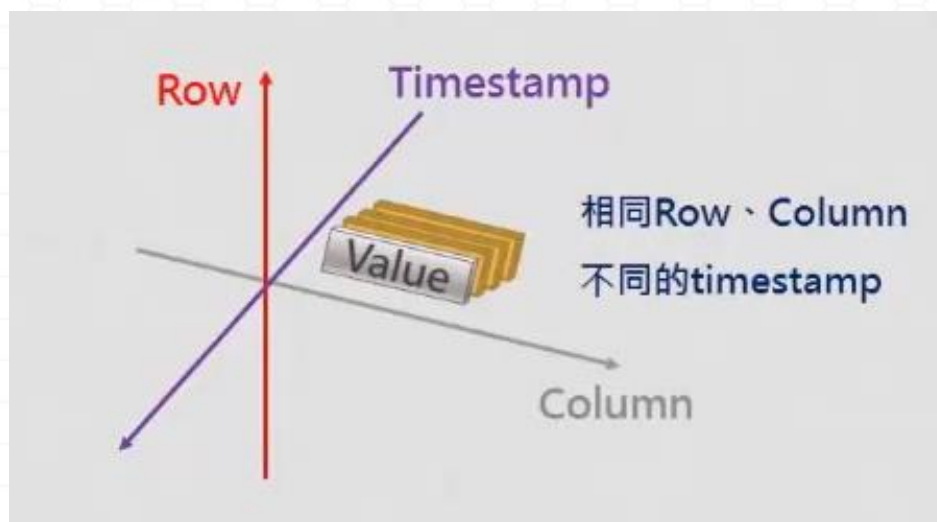
```
SortedMap(  
  RowKey, List(  
    SortedMap(  
      Column, List(  
        Value, Timestamp  
      )  
    )  
  )  
)
```



# 資料存儲

Key{row, column, timestamp} => Value

```
A00001.00 column=cf:DATE, timestamp=1317950820261, value=\x00\x00\x012\xDB\xFD\x8F\x9E
A00001.00 column=cf:FACILITY, timestamp=1317950820261, value=BSET
A00001.00 column=cf:LOT, timestamp=1317950820261, value=A00001.00
A00001.00 column=cf:OPERATOR, timestamp=1317950820261, value=andy
```



# 實際儲存結構

- Table Name
- Column Family Name
- Rowkey
- Qualifier

Hbase Table 只定義 Table Name 與 Column Family Name · 不定義 Qualifier。

```
hbase(main):019:0> scan 'SaleAmount'
```

ROW	COLUMN+CELL
A001	column=Product:Drive_Side, timestamp=1385472194283, value=Left
A001	column=Product:Product_ID, timestamp=1385472194178, value=C200
A001	column=Sale_Amount:China, timestamp=1385472194220, value=8000
A001	column=Sale_Amount:Taiwan, timestamp=1385472194330, value=1000
A002	column=Product:Drive_Side, timestamp=1385472194378, value=Left
A002	column=Product:Product_ID, timestamp=1385472194429, value=C300
A002	column=Sale_Amount:China, timestamp=1385472194468, value=6000
A003	column=Product:Drive_Side, timestamp=1385472869841, value=Right
A003	column=Product:Product_ID, timestamp=1385472869913, value=E300
A003	column=Sale_Amount:China, timestamp=1385472869959, value=200
A003	column=Sale_Amount:Japan, timestamp=1385472870007, value=3000

```
3 row(s) in 0.0280 seconds
```

# Schema 設計原則





# DDI 原則

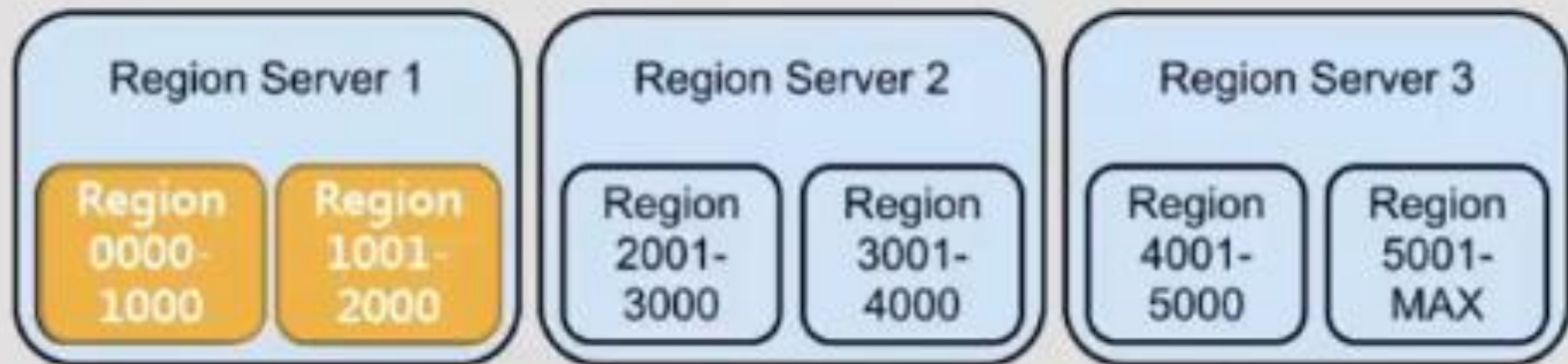
---

- De-normalization
- Duplication
- Intelligent key



# 避免HotSpot

## Hot-spot (熱點)



### □ Hot-Spot產生原因：

字典排序 這些特殊符號也是有排序的

字典排序的盲點

# 避免HotSpot 的做法

## Time Series Data

① Salting :

資料前面加特殊演算法產生的字串

② Field swap/promotion :

例如Google搜尋引擎的作法

③ Randomization :

隨機，加上一個ND5的編碼

# HBase 實做

# Hbase Shell

檢查狀態

- `hbase> status`

檢查版本

- `hbase> version`

檢查跟表格相關說明

- `hbase> table_help`

使用者資訊

- `hbase> whoami`



# Hbase DDL 指令

## ■ 啟用表格

- Syntax :

- enable 'table-name'

- Example :

- enable 'easylearning'

## ■ 確認表格啟用

- Syntax :

- is\_enabled 'table-name'

- Example :

- is\_enabled 'easylearning'

# Hbase DDL 指令

## ■ 停止使用表格

- Syntax :

- disable 'table-name'

- Example :

- disable 'easylearning'

## ■ 確認表格停止

- Syntax :

- is\_disabled 'table-name'

- Example :

- is\_disabled 'easylearning'

# Hbase DDL 指令

## ■ 建立表格

- Syntax :

- create 'table-name' , { NAME=>'column-family-name' } , { NAME=>'column-family-name' }

- Example :

- create 'easylearning' , {NAME=> 'Student'} , {NAME => 'Teacher'}

## ■ 檢視表格

- Syntax :

- describe 'table-name'

- Example :

- describe 'easylearning'

## ■ 表列所有表格

- Syntax :

- list

# Hbase DDL 指令

## ■ 確認表格是否存在

- Syntax :
- exists 'table\_name'
- Example:
- exists 'easylearning'

## ■ 刪除表格

- Syntax :
- drop 'table\_name'
- Example :
- disable 'easylearning'
- drop 'easylearning'



# Hbase DDL 指令

## ■ 删除Column Family

- Syntax:

- alter ' table name ', 'delete' ⇒ ' column family '

- Example:

- alter 'easylearning', 'delete' ⇒ 'student'

# Hbase DML 指令

## ■ 新增資料

- Syntax :

- put 'table-name' , 'Row-Key' , 'Colum-family-name : column-name' , 'value to be inserted'

- Example :

- put 'easylearning' , 'stud-101' , 'Student : stud\_name' , 'Alice'

## ■ 取得資料內容

- Syntax :

- get '<table name>' , 'row1'

- Example :

- get 'easylearning' , 'stud-101'

# Hbase DML 指令

## ■ 刪除Cell 資料

- Syntax :

- delete '<table name>', '<row>', '< Colum-family-name : column name >', '<timestamp>'

- Example :

- delete 'easylearning', 'stud-101', 'Student : stud\_name', 1417521848375

## ■ 刪除全部資料

- Syntax :

- deleteall '<table name>', '<row>'

- Example :

- deleteall 'easylearning', 'stud-101'

# Hbase DML 指令

## ■ 取得特定表格資料

- Syntax :

- scan 'table-name'

- Example :

- scan 'easylearning'

## ■ 計算列數

- Syntax :

- count 'table-name'

- Example :

- count 'easylearning'



# Hbase DML 指令

## ■ 重建表格

- Syntax :

- truncate 'table-name'

- Example :

- truncate 'easylearning'

# HBase API

# 使用pig 將資料放置到hbase 中

```
1 A = load 'small_recommend/part-*' Using PigStorage() as (userid:int, recommend:chararray);  
2 STORE A INTO 'mydata' USING org.apache.pig.backend.hadoop.hbase.HBaseStorage('mycf:recommend_list');
```

# (選擇性)將推薦結果轉移到HBase

## ■ 使用ImportTSV

### □ hbase

org.apache.hadoop.hbase.mapreduce.ImportTsv -  
Dimporttsv.columns=UserID,MovieID,Vote,datetime  
Recommendation /user/cloudera/mlresult





# 使用HappyBase 建立API

```
def recommend(request, userid):  
    connection = happybase.Connection('localhost')  
    connection.open()  
    table = connection.table('mydata')  
    recommend_str = table.row(userid)['mycf:recommend_list']  
    recommend_list = [i.split(':')[0] for i in recommend_str.strip('[').strip(']').split(',')]  
    data = json.dumps(recommend_list)  
    return HttpResponse(data, mimetype='application/json' )
```

(選擇性) 使用jetty, jersey, mvn建立API

***jetty://***

***ma*ven**



**Jersey**

RESTful Web Services in Java.

# 透過Python 操作HBase

- 安裝HappyBase

- `pip install happybase`

- HappyBase 可以讓使用者透過Python 操作 Apache HBase.

- HappyBase 使用 Thrift連結HBase,

- HBase 0.9x 被正式納入

# 透過Python 操作Hbase (範例)

```
import happybase
connection = happybase.Connection('localhost')
connection.open()
print connection.tables()

connection.create_table(
    'recommendation',
    {'cf1': dict() }
)
table = connection.table(' recommendation')
table.put('userid', {'cf1:rec1': 'value1'})
row = table.row('userid')
print row['cf1:rec1']

for key, data in table.scan():
    print key, data

connection.close()
```



The background features a light blue hexagonal grid pattern. Overlaid on this is a large, faint, circular graphic composed of concentric rings and radial lines, resembling a stylized sun or a target. The text "THANK YOU" is centered in a bold, dark blue, sans-serif font.

**THANK YOU**