HBase 簡介

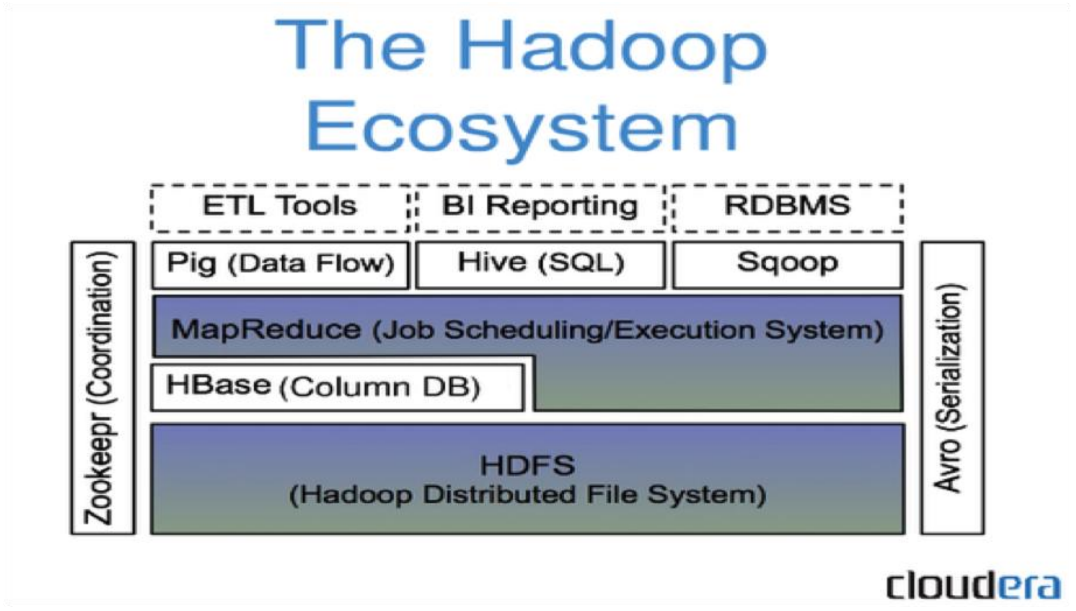
軟體發展組
莊家雋

Outline

- What is Hbase
- Hbase Architecture
  - Master, region server, zookeeper
- Hbase Data model
- Hbase basic operations
  - Put, Get, Scan, Delete
- Hbase limit
- Performance issue
  - Rowkey, filter
- Hbase use case

What is HBase

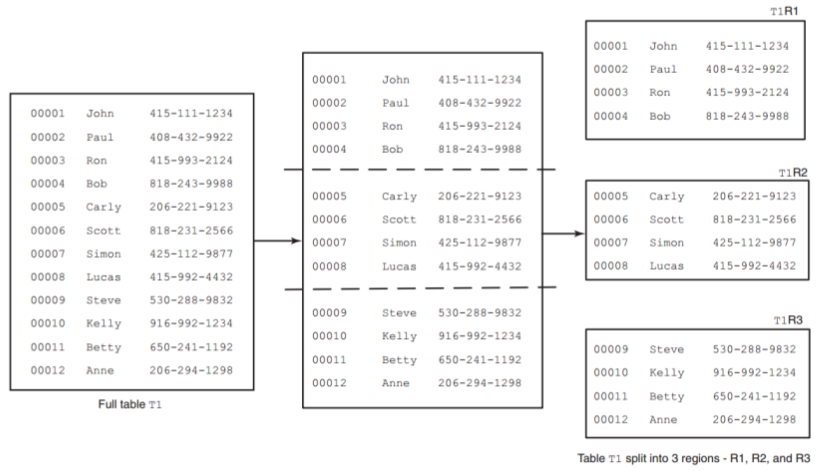
- Hbase是一個高可靠性、高性能、column-orient、 scalability 的分散式儲存系統



Google	OpenSource
GFS	HDFS
MapReduce	Hadoop MapReduce
BigTable	HBase
Chubby	Zookeeper

Hbase architecture

- RegionServer
  - Table can be split into many region
  - Each RegionServer contains many regions
  - Add RS to horizontal scale up



00001	John	415-111-1234
00002	Paul	408-432-9922
00003	Ron	415-993-2124
00004	Bob	818-243-9988
00005	Carly	206-221-9123
00006	Scott	818-231-2566
00007	Simon	425-112-9877
00008	Lucas	415-992-4432
00009	Steve	530-288-9832
00010	Kelly	916-992-1234
00011	Betty	650-241-1192
00012	Anne	206-294-1298

Full table T1

T1R1

00001	John	415-111-1234
00002	Paul	408-432-9922
00003	Ron	415-993-2124
00004	Bob	818-243-9988

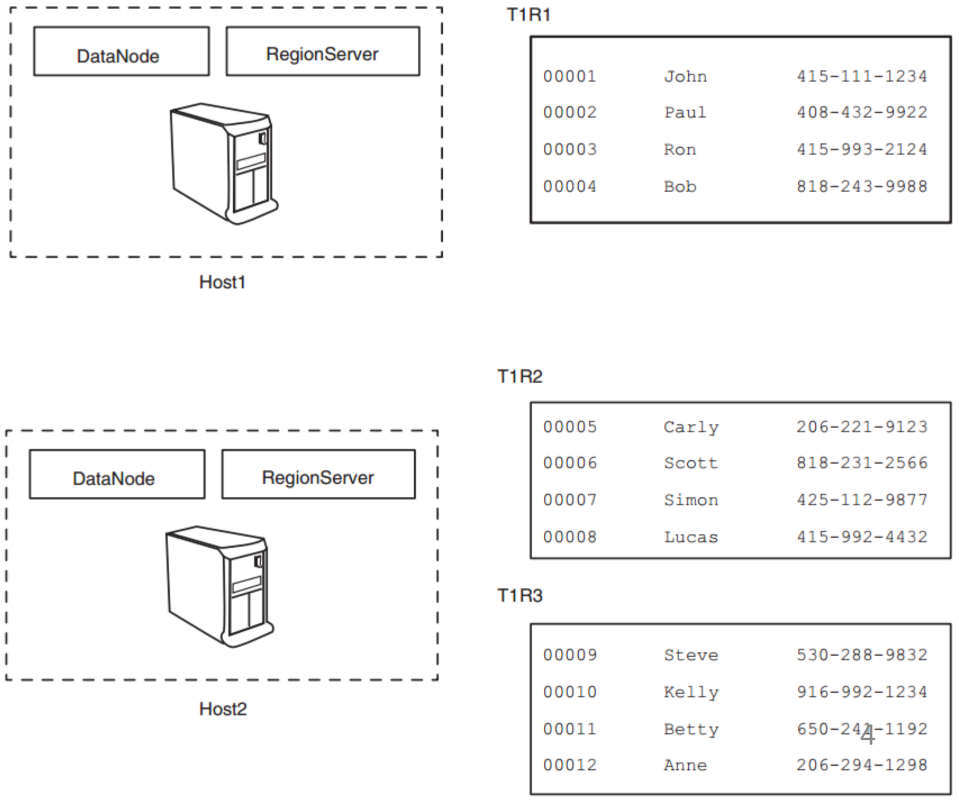
T1R2

00005	Carly	206-221-9123
00006	Scott	818-231-2566
00007	Simon	425-112-9877
00008	Lucas	415-992-4432

T1R3

00009	Steve	530-288-9832
00010	Kelly	916-992-1234
00011	Betty	650-241-1192
00012	Anne	206-294-1298

Table T1 split into 3 regions - R1, R2, and R3



T1R1

00001	John	415-111-1234
00002	Paul	408-432-9922
00003	Ron	415-993-2124
00004	Bob	818-243-9988

T1R2

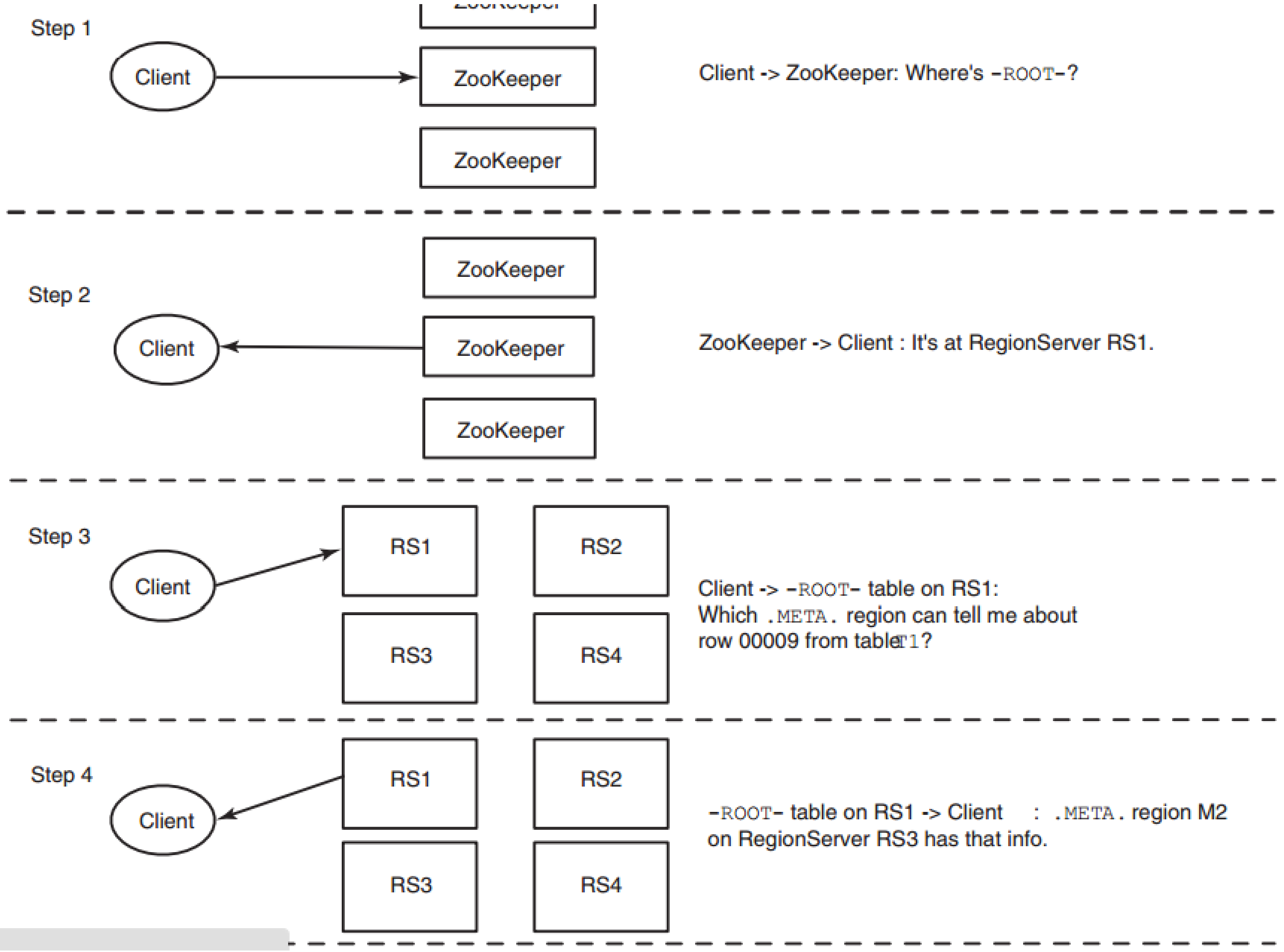
00005	Carly	206-221-9123
00006	Scott	818-231-2566
00007	Simon	425-112-9877
00008	Lucas	415-992-4432

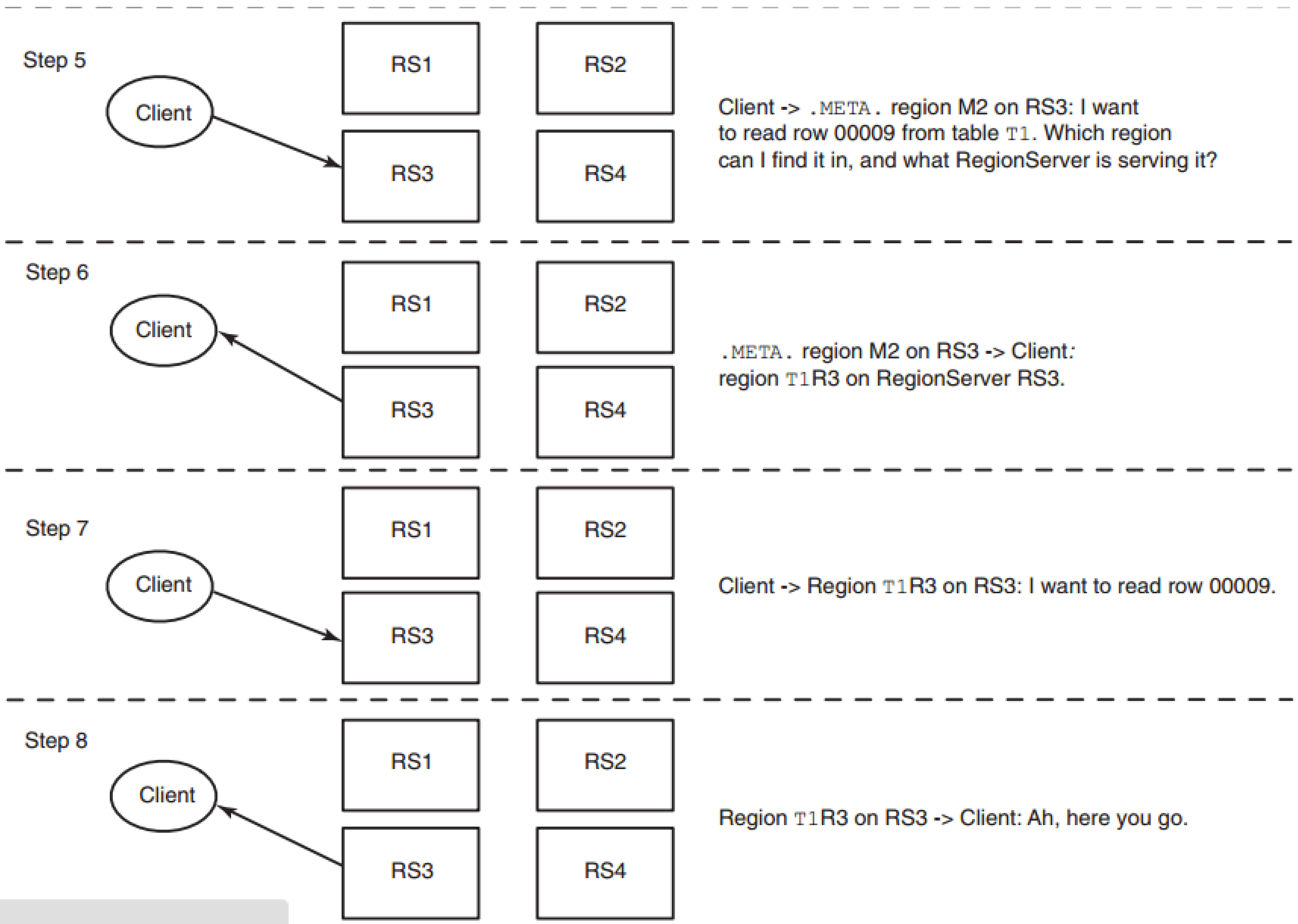
T1R3

00009	Steve	530-288-9832
00010	Kelly	916-992-1234
00011	Betty	650-241-1192
00012	Anne	206-294-1298

Hbase architecture

- HMaster
  - Responsible for assigning regions to RegionServer
- Zookeeper
  - Use to find two special table –ROOT- and .META.


Step 1
Client
ZooKeeper
ZooKeeper
ZooKeeper
Client -> ZooKeeper: Where's -ROOT-?
Step 2
ZooKeeper
Client
ZooKeeper
ZooKeeper
ZooKeeper -> Client : It's at RegionServer RS1.
Step 3
Client
RS1
RS2
RS3
RS4
Client -> -ROOT- table on RS1:
Which .META. region can tell me about
row 00009 from table T1?
Step 4
Client
RS1
RS2
RS3
RS4
-ROOT- table on RS1 -> Client : .META. region M2
on RegionServer RS3 has that info.



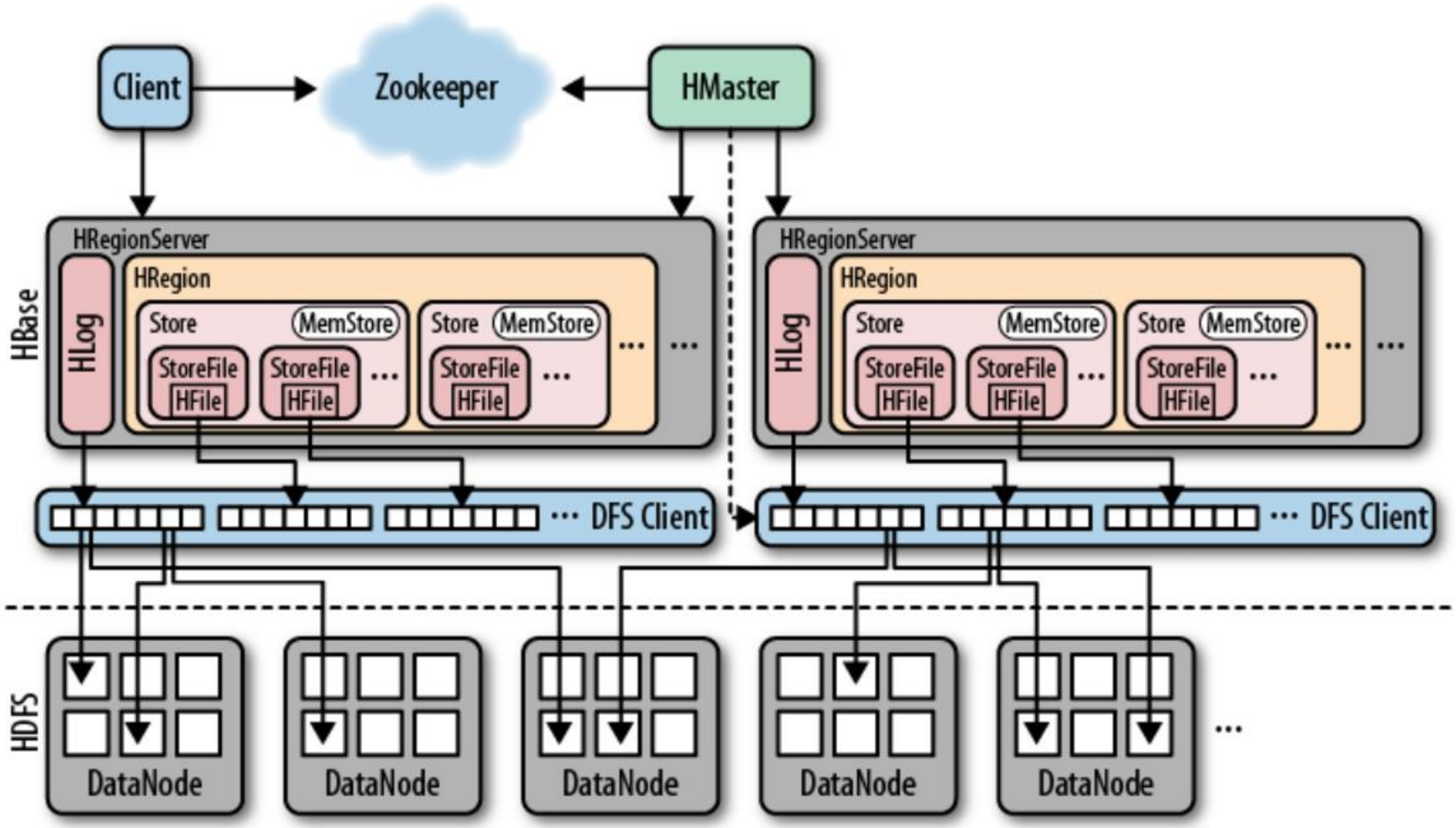
Client -> `.META.` region M2 on RS3: I want to read row 00009 from table `T1`. Which region can I find it in, and what RegionServer is serving it?

`.META.` region M2 on RS3 -> Client: region `T1`R3 on RegionServer RS3.

Client -> Region `T1`R3 on RS3: I want to read row 00009.

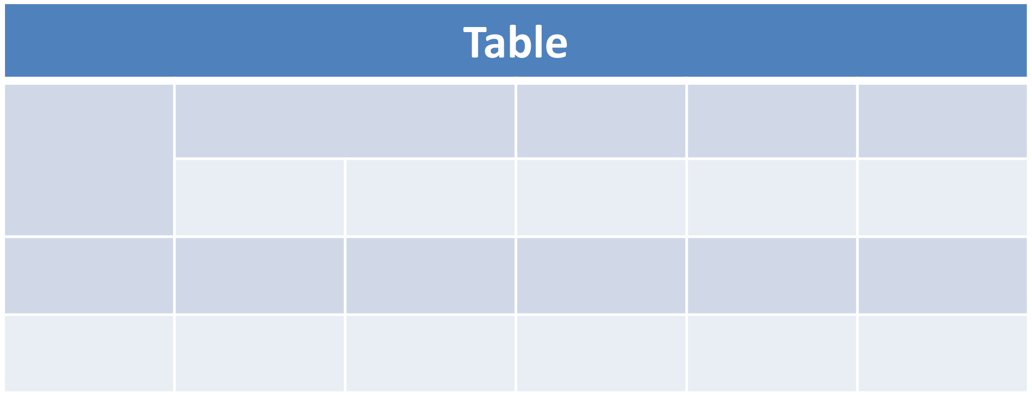
Region `T1`R3 on RS3 -> Client: Ah, here you go.

Hbase architecture



Hbase data model

- Table
  - Hbase organize data into tables



Hbase data model

- Row and rowkey
  - Data is stored to its row
  - Rows are ineditfied uniquely by their rowkey
  - Rowkeys are stored lexicographically
  - Rowkeys are always treated as byte[]

Table					
Rowkey					
R1					
R2					

Hbase data model

- Column family
  - Data within a row is grouped by column family (CF)
  - CF must be declared with table creation
  - CF cannot be add or delete
  - CF names are treated as String

Table					
Rowkey	CF1		CF2	CF3	
R1					
R2					

Hbase data model

- Column qualifier
  - Qualifier is used to address data
  - Qualifier need NOT be specified in advanced
  - Qualifier name is treated as byte[]
  - CF + qualifier can be seen as column in RDBMS
    - Represent by CF:qualifier

Table					
Rowkey	CF1		CF2	CF3	
	q1	q2	q1	q3	q4
R1					
R2					

Hbase data model

- Cell
  - Combination of rowkey, CF, qualifier uniquely identifies a cell
  - Data is stored in a cell, call value
  - Value is treated as byte[]

Table					
Rowkey	CF1		CF2	CF3	
	q1	q2	q1	q3	q4
R1	V1	v2			
R2	v1	v2	v3	v4	v5

Hbase data model

- Version
  - Values within a cell are versioned.
  - Versions are identified by timestamp, treated as long

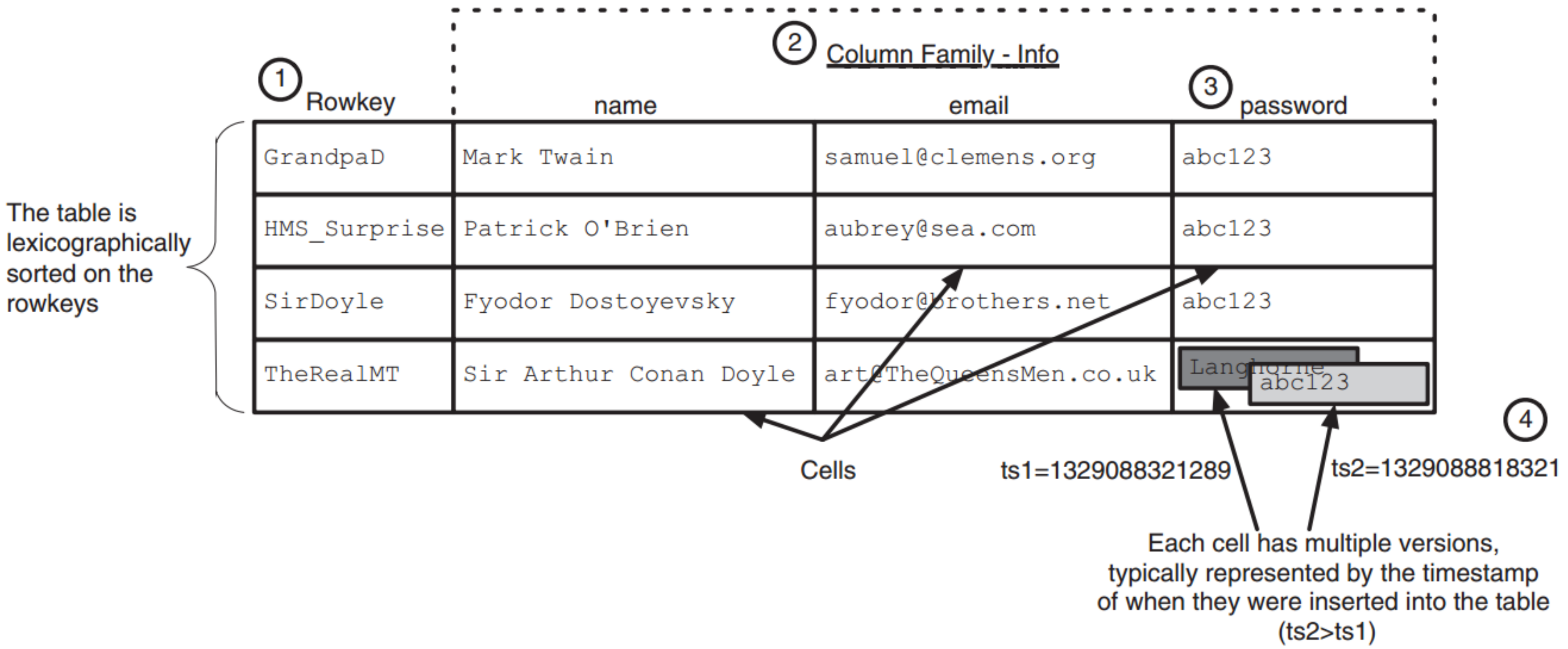
Table					
Rowkey	CF1		CF2	CF3	
	q1	q2	q1	q3	q4
R1	V1	v2			
R2	v1	v2	v3	v4	V5-1 V5-2

Ver:1329088321289

Ver: 132908818321

Hbase data model

- A example put all together



Rowkey	name	email	password
GrandpaD	Mark Twain	samuel@clemens.org	abc123
HMS_Surprise	Patrick O'Brien	aubrey@sea.com	abc123
SirDoyle	Fyodor Dostoyevsky	fyodor@brothers.net	abc123
TheRealMT	Sir Arthur Conan Doyle	art@TheQueensMen.co.uk	Langhorne / abc123

1
2
Column Family - Info
3
4
The table is lexicographically sorted on the rowkeys
Cells
ts1=1329088321289
ts2=1329088818321
Each cell has multiple versions, typically represented by the timestamp of when they were inserted into the table (ts2>ts1)

Hbase basic operations

- Data manipulate
  - Put, Get, Scan
- Table administrate
  - Table Create、Disable、Drop

- Use Hbase shell and Java language

Hbase Shell

- $ hbase shell # start hbase shell
- In Hbase(main):
- create 't1','info'
- put 't1','GrandpaD','info:name', 'mark Twain'
- put 't1','GrandpaD','info:email','samuel@clemens.org‘
- put 't1','GrandpaD','info:password', 'ABC456‘
- put 't1','GrandpaD','info:password', ‘abc123‘
- put 't1','GrandpaD',‘**INFO**:password', ‘abc123‘ **FAIL**

- get 't1','GrandpaD‘
- get 't1', 'GrandpaD', {COLUMN => 'info:password'}
- get 't1', 'GrandpaD', {COLUMN => 'info:password', VERSIONS => 3}

- scan 't1‘

- delete 't1', 'GrandpaD', 'info:password‘
- deleteall 't1','GrandpaD‘

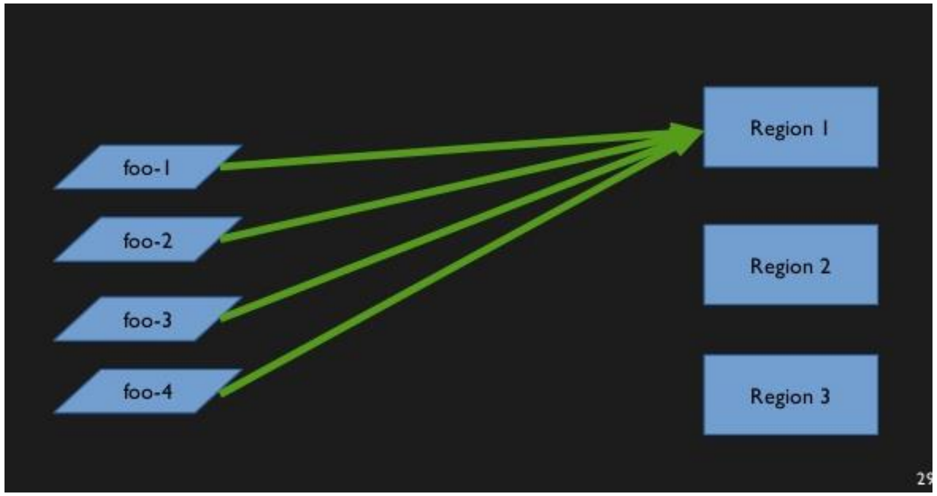
- disable ‘t1’
- drop ‘t1’

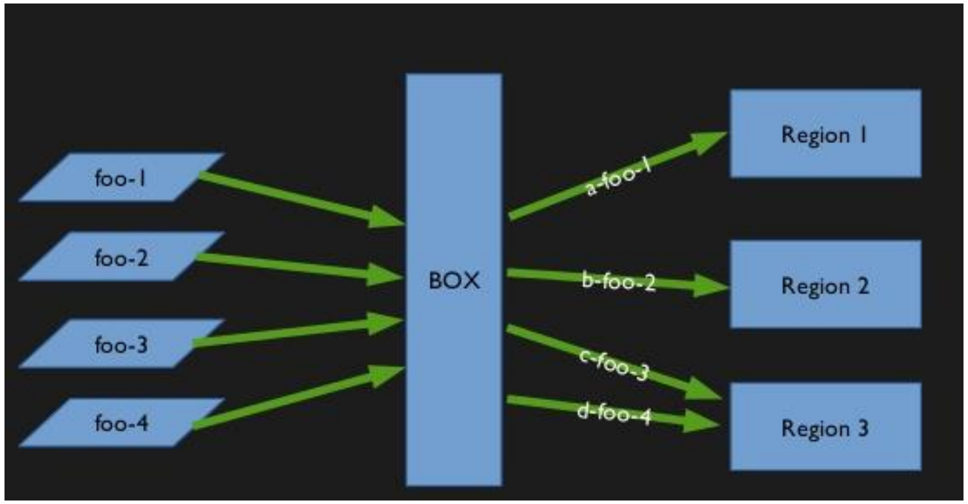
Hbase Limit

- NO Table join
  - A single BIG table, Denormalize Design
- NO ACID
- NO secondary index
  - NO WHERE clause in SQL
- Row level transaction
  - Multiple rows modification is not thread safe
- C & P in CAP theorem

Performance issue

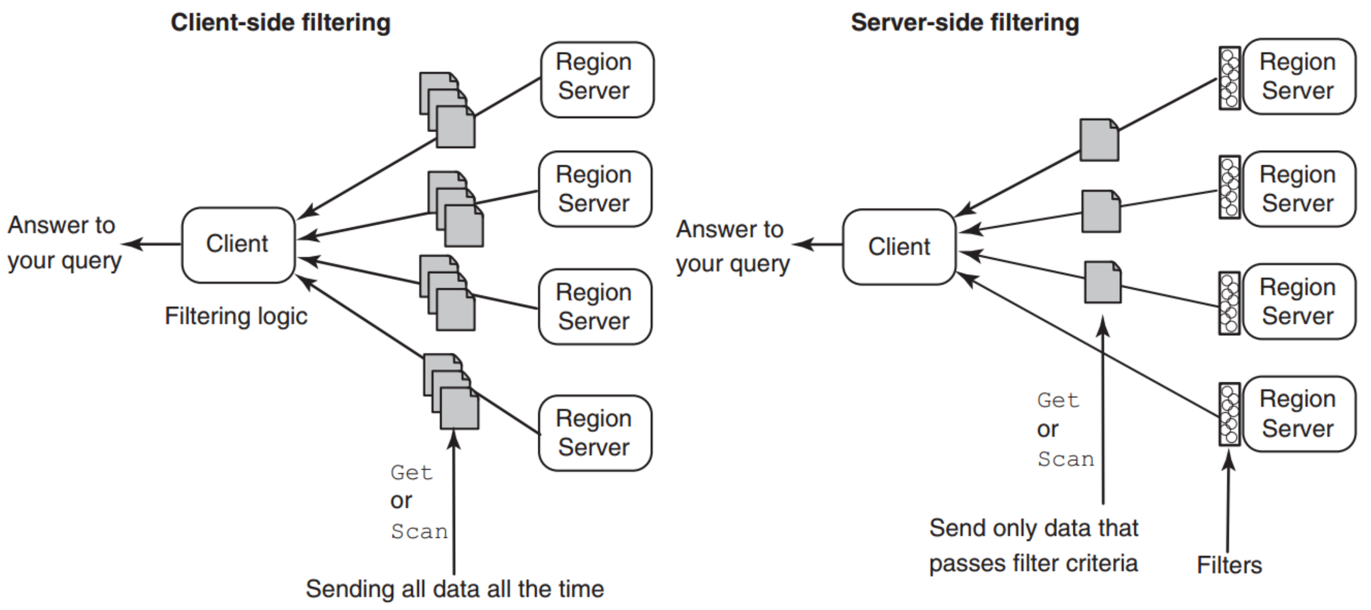
- Bad Rowkey design result in hotspot
  - Rowkey stored lexicographically
  - Avoid by Salting or hashing





Performance issue

- Use Filter to reduce network/IO overhead



Hbase use case

- Count Statistics
- Store Graph

Statistics example

- By user
- By time unit
  - Per Day, per month , per year…
- By action type
  - Like, comment, …

Initial Design

Table													
RK (userID)	TIME												
	20140901	20140902	…	20140931	20140900	20140901	…	20140923	201401	201402	…	201412	…
123	3			4									
987					15			9					

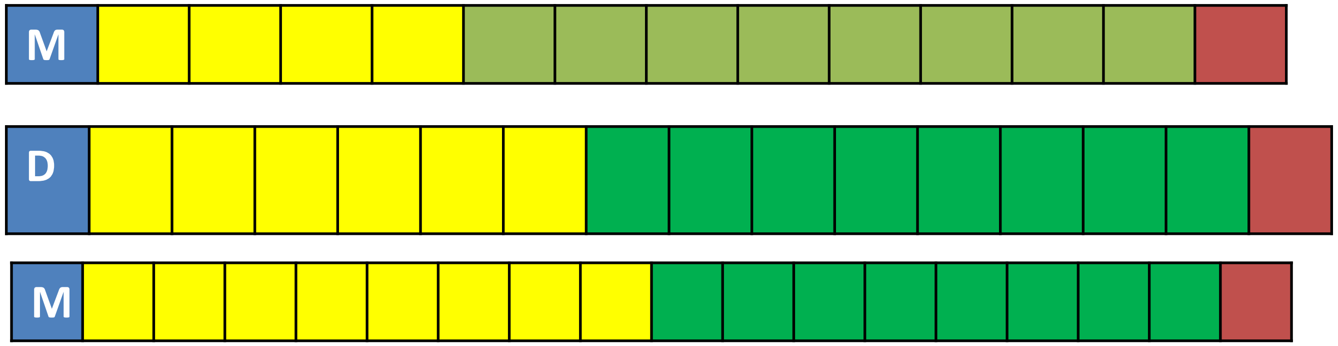
- Bad Design 1
  - Billion column per row is fine
  - Use column filter for query is BAD for performance
  - How about query by action ?

Initial Design

Table												
Rowkey (userID)	Like				Dislike				Comment			
	2014	201401	…	20140914	2014	201401	…	20140914	2014	201401	…	20140914
123	3			4								
987					15			9				

- Bad Design 2
  - CF should be defined first
    - How about add new action type ??
  - Number of CF should NOT be too much

Composition Rowkey



- Unit+TimeBase+UserID+Type
  - Unit : Char, (1 byte, H, D, M)
  - TimeBase: String
    - Length: 4 (Unit = M) or 6 (Unit = D) or 8 (Unit = H)
  - UserID: Long (8 bytes)
  - Type: Short (1 bytes)
    - 1 = Like, 2 = Dislike, 3 = comment

Better Design

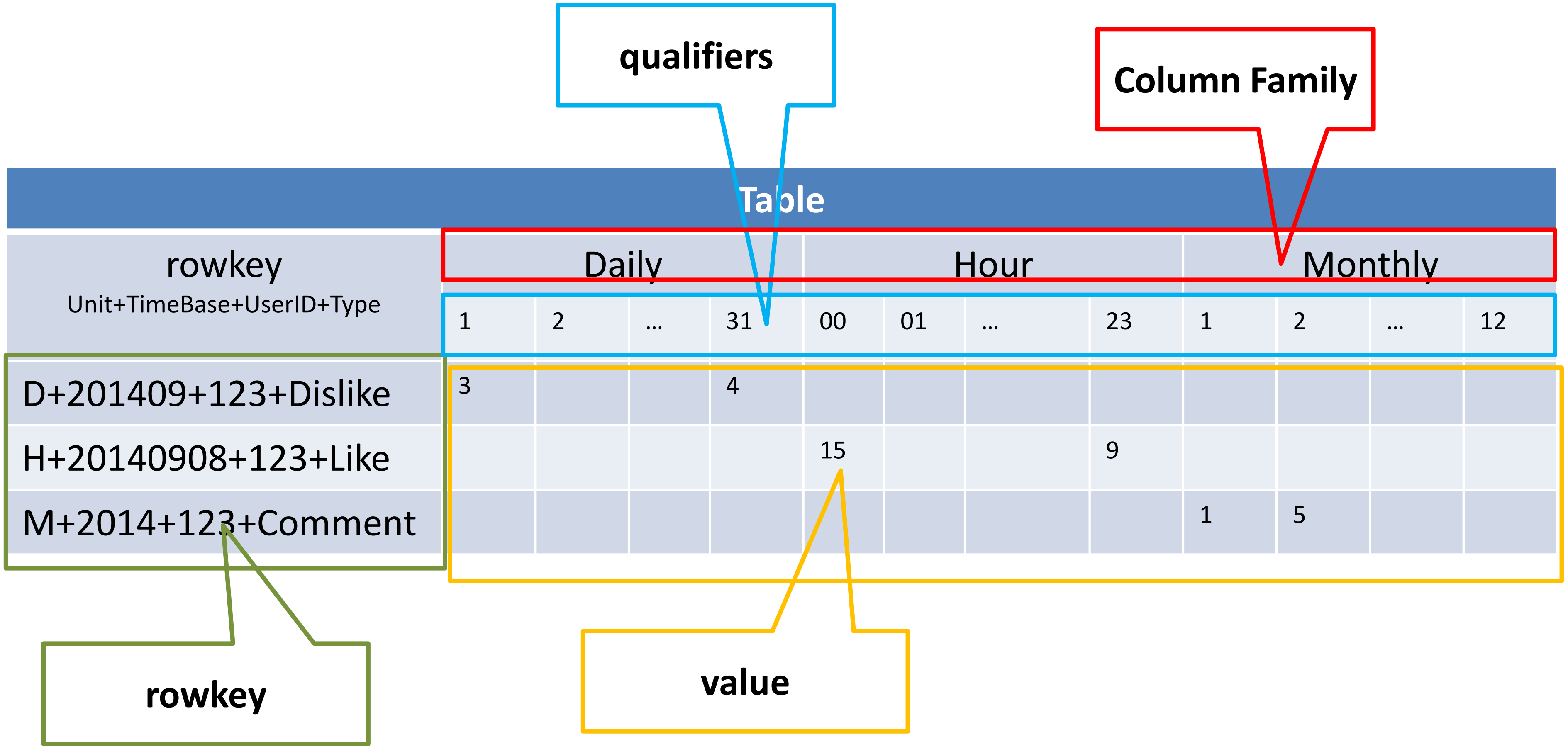


Table												
rowkey Unit+TimeBase+UserID+Type	Daily				Hour				Monthly			
	1	2	…	31	00	01	…	23	1	2	…	12
D+201409+123+Dislike	3			4								
H+20140908+123+Like					15			9				
M+2014+123+Comment									1	5		

主鍵查詢
rowkey query

- Get 789's Like action counts from from 2014/9/7 to 2014/9/20
  - Full RK: D + 201409 + DEF + Like

Table										
Rowkey (Unit+TimeBase+UserID+Type)	…	Daily								…
		…	7	8	9	…	19	20	…	
D+201409+123+Dislike			21	14	56	…	21	47		
D+201409+123+Like			25	12	78	…	98	112		
D+201409+123+comment			27	21	57	…	31	34		
D+201409+789+Like			**26**	**41**	**29**	**…**	**7**	**35**		
H+20140908+123+Like										
M+2014+123+Comment										

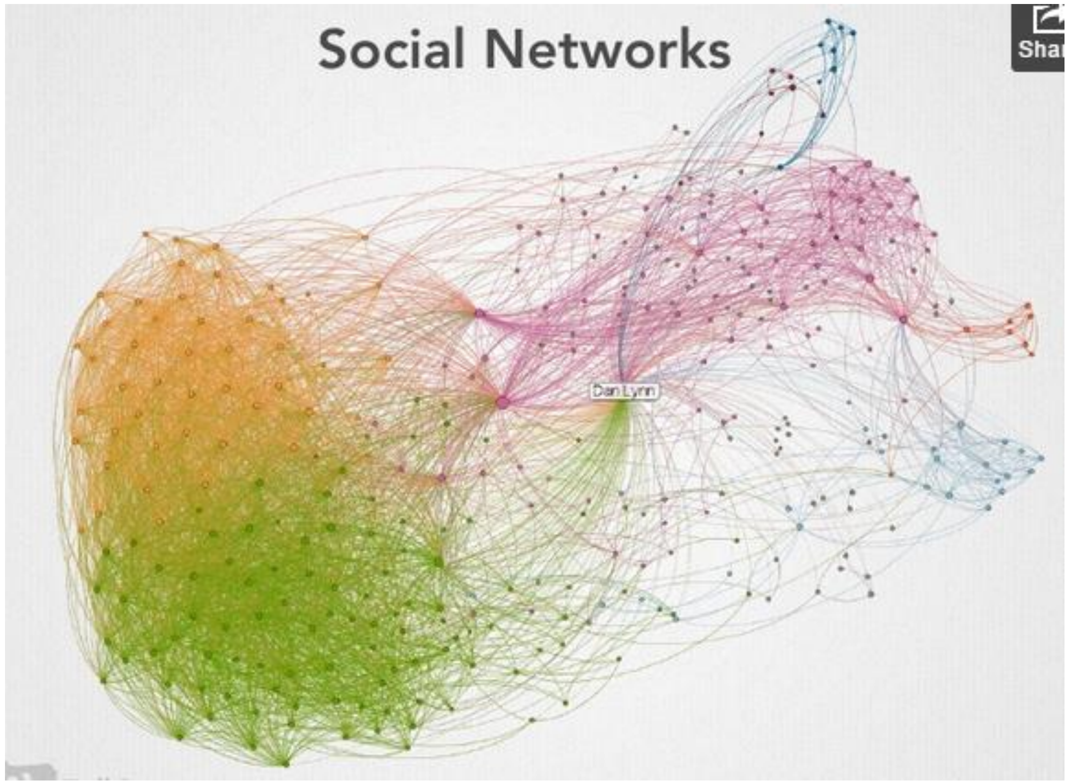
部分主鍵查詢
Partial rowkey query

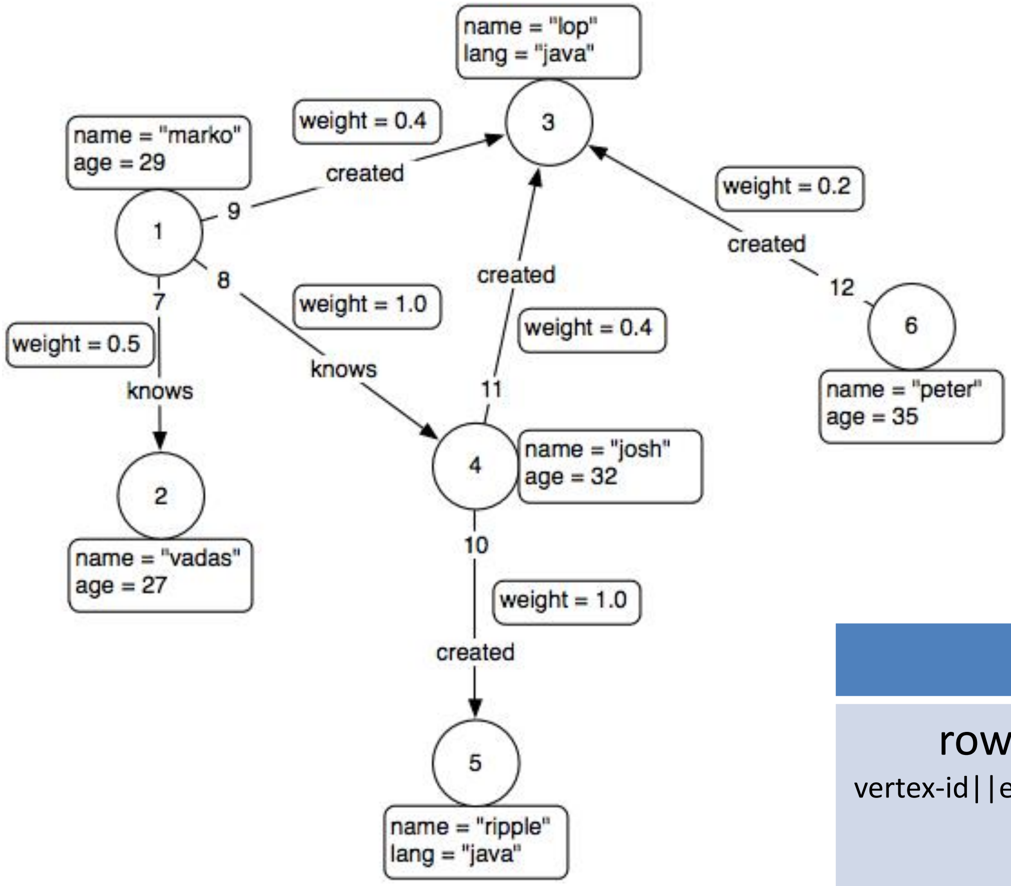
- Get 123's each action counts from from 2014/9/7 to 2014/9/20
  - Start RK: D+ 201409 + 123
  - END RK : D+ 201409 + 124

Table										
Rowkey (Unit+TimeBase+UserID+Type)	…	Daily								…
		…	7	8	9	…	19	20	…	
D+201409+123+Dislike			**21**	**14**	**56**	**…**	**21**	**47**		
D+201409+123+Like			**25**	**12**	**78**	**…**	**98**	**112**		
D+201409+123+comment			**27**	**21**	**57**	**…**	**31**	**34**		
D+201409+789+Like			**26**	**41**	**29**	**…**	**7**	**35**		
H+20140908+123+Like										
M+2014+123+Comment										

Store Graph in HBase

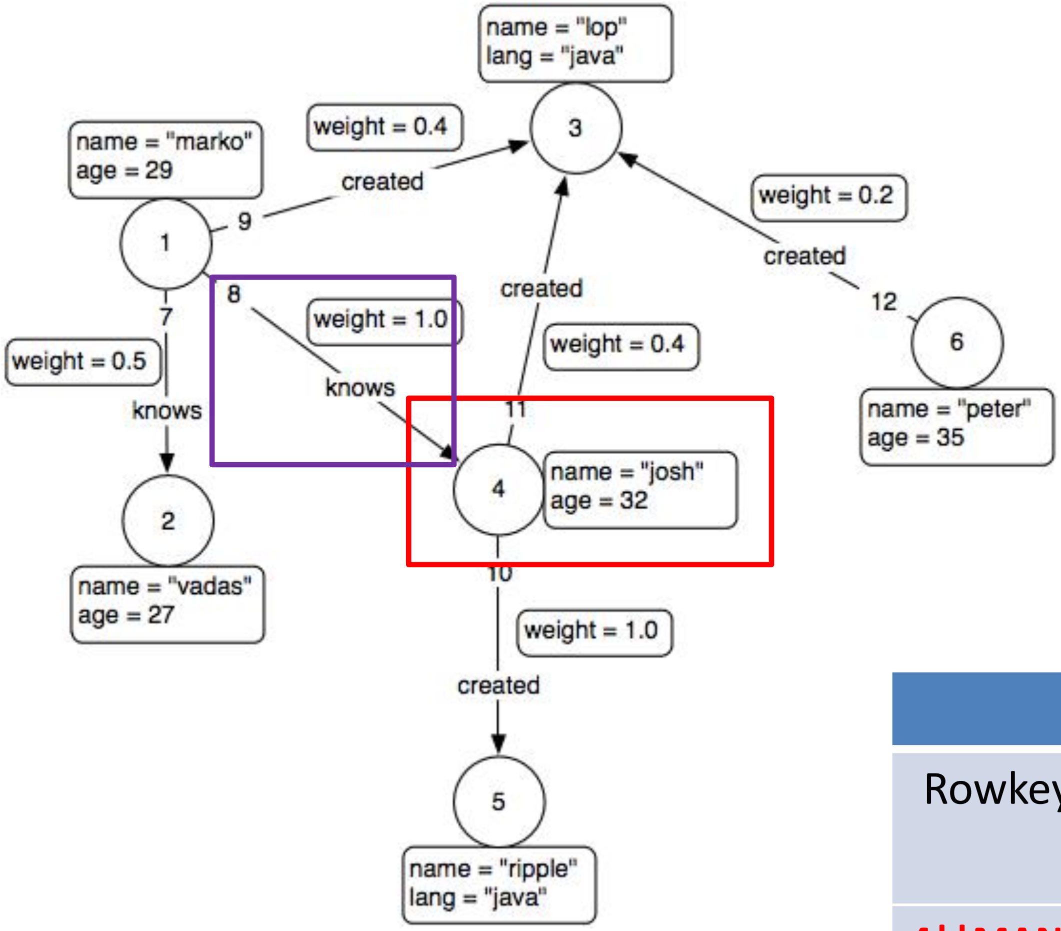
- Use property graph model to present a graph
- Use Hbase to store property graph model





Vertex Table		
rowkey vertex-id||entity-type	Property (CF)	
	property-key1@property-value-type (qualifier)	…
	Property-value1	…

Edge Table		
rowkey vertex1-row-key --> label --> vertex2-row-key	Property	
	property-key1@property-value-type	…
	Property-value1	…



Vertex Table		
Rowkey	Property	
	name@String	age@int
4||MAN	josh	32

Edge Table	
Rowkey	Property
	weight@float
1||WOMAN--> knows-- > 4||MAN	1.0