設計高效能 HBase Schema—了解HBase運作方式與資料特徵

Scott Miao / Trend Micro RD

takeshi.miao@gmail.com

2012/10/2

Agenda

- Big Data
- Architecture Overview
  - HFile
  - KeyValue
- Key Design

大資料時代！

- 過去3年所產生的資料量，比過去四萬年創造的資料量還多！
- WallMart的資料量是美國國會圖書館的167倍！
- eBay分析平台每天處理的資料量高達100PB！(約1,000,000GB)
- 截至2010年，世界電子資料儲存量為1.2ZB！(1,200,000PB)
- 根據IDC預測，2020年世界電子資料儲存量會是2009年的基礎上，再加上44倍，達到35萬億GB！
  - 35,000,000,000,000 Giga Bytes

架构师 10 月刊 – http://www.infoq.com/cn/minibooks/architect-oct-10-2011

APACHE HBASE

- Apache Software Foundation top-level project
- Being Categorized to key-value in noSQL DB
- Originated from Google
  - Bigtable: A Distributed Storage System for Structured Data
  - A distributed storage system for managing structured data that is designed to scale to a very large size: **petabytes** of data across thousands of commodity servers
  - A sparse, distributed, persistent multi-dimensional sorted map

Hbase: The Definitive Guide - http://www.amazon.com/HBase-Definitive-Guide-Lars-George/dp/1449396100/ref=sr_1_1?ie=UTF8&qid=1339060175&sr=8-1

- a sparse, distributed, persistent multi-dimensional sorted map
  - which is indexed by <u>row key, column key (column family + qualifiers)</u>, and a <u>timestamp</u>

```
(Table, RowKey, Family, Column, Timestamp) → Value
```

In a more programming language style, this may be expressed as:

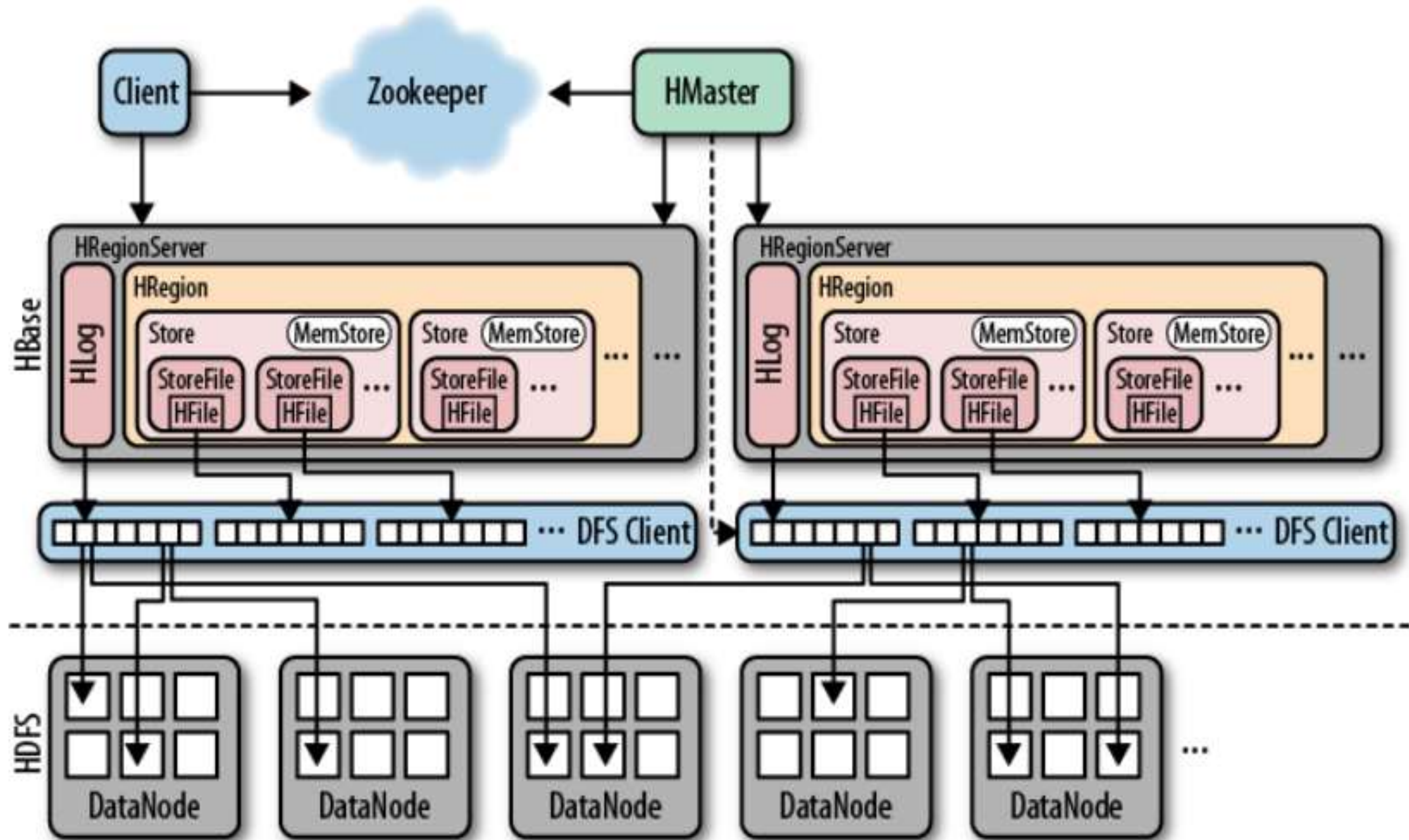
```
SortedMap<
    RowKey, List<
        SortedMap<
            Column, List<
                Value, Timestamp
            >
        >
    >
>
```

Column Families

or all in one line:

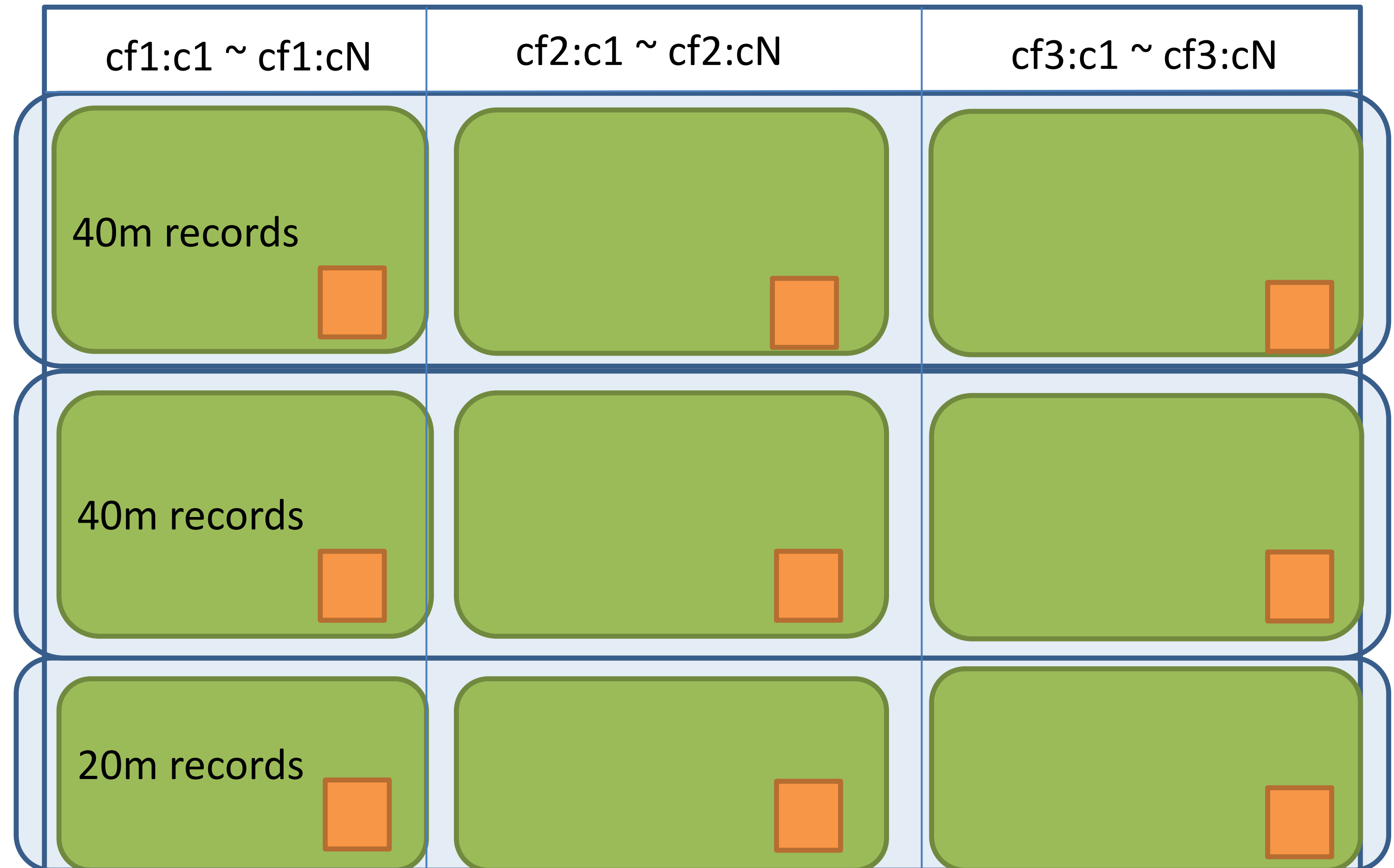
```
SortedMap<RowKey, List<SortedMap<Column, List<Value, Timestamp>>>>
```

Architecture Overview



Born to be sharding

100m records



HFile Format

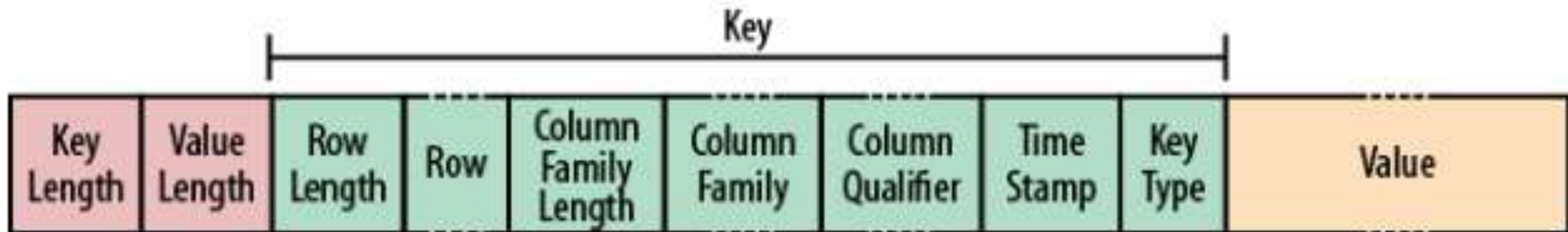
- The actual storage files are implemented by the *HFile* class
- <u>Store HBase’s data efficiently</u>
- Blocks
  - Fixed size
    - Trailer, File Info
  - Others are variable size

Data | Data | Data | Meta (Optional) | Meta (Optional) | File Info | Data Index | Meta Index | Trailer

Magic | Key Value | Key Value | Key Value | Key Value | Key Value | Key Value

KeyValue Format

- Each KeyValue in the HFile is a low-level byte array

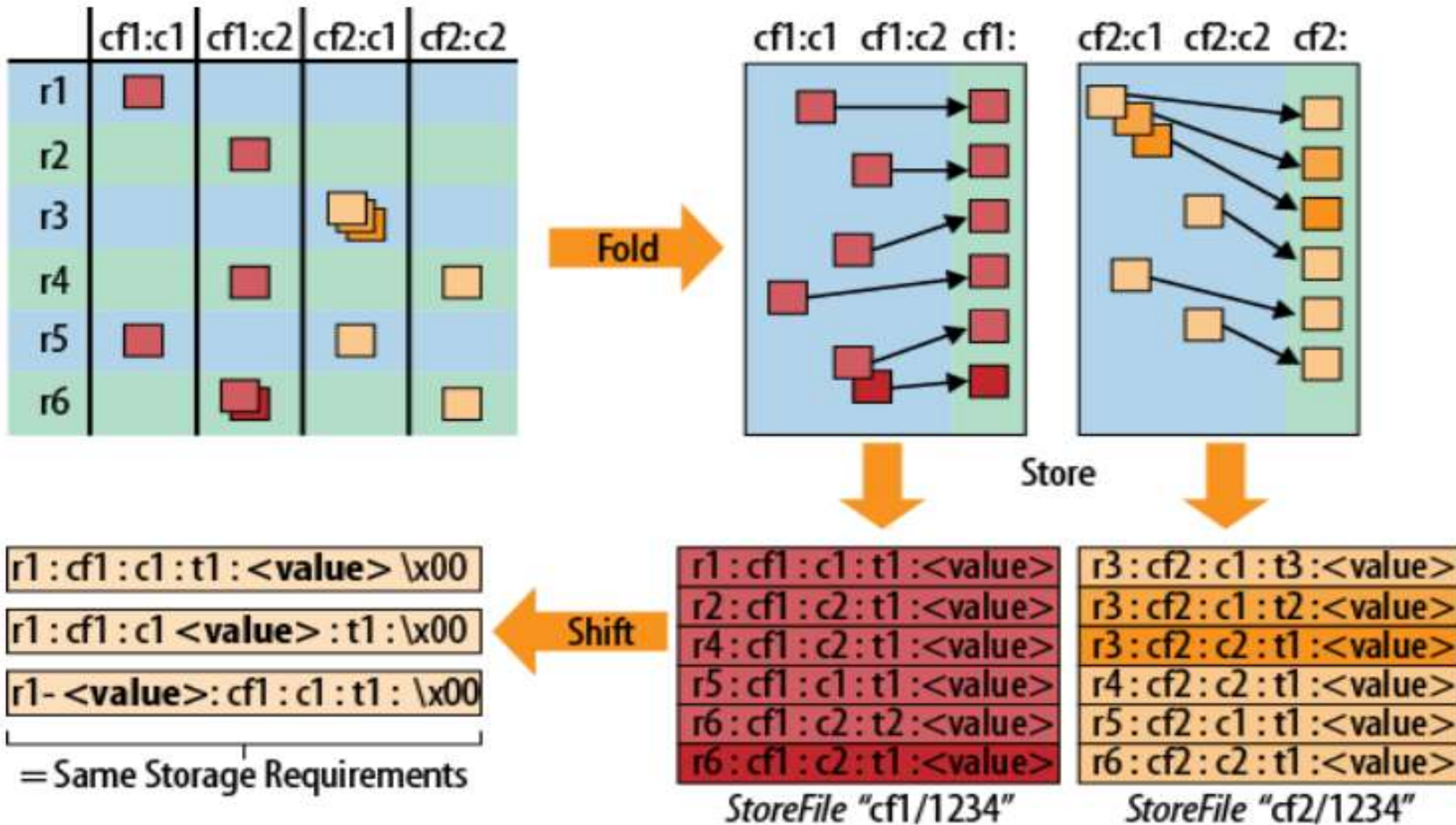


- Fixed-length Numbers
  - Key Length
  - Value Length
- If you deal with small values
  - Try to keep the key small
    - Choose a short row and column key
    - family name with a single byte and the qualifier equally short
  - Compression should help mitigate the overwhelming key size problem
- The sorting of all KeyValues in the store file helps to keep similar keys close together

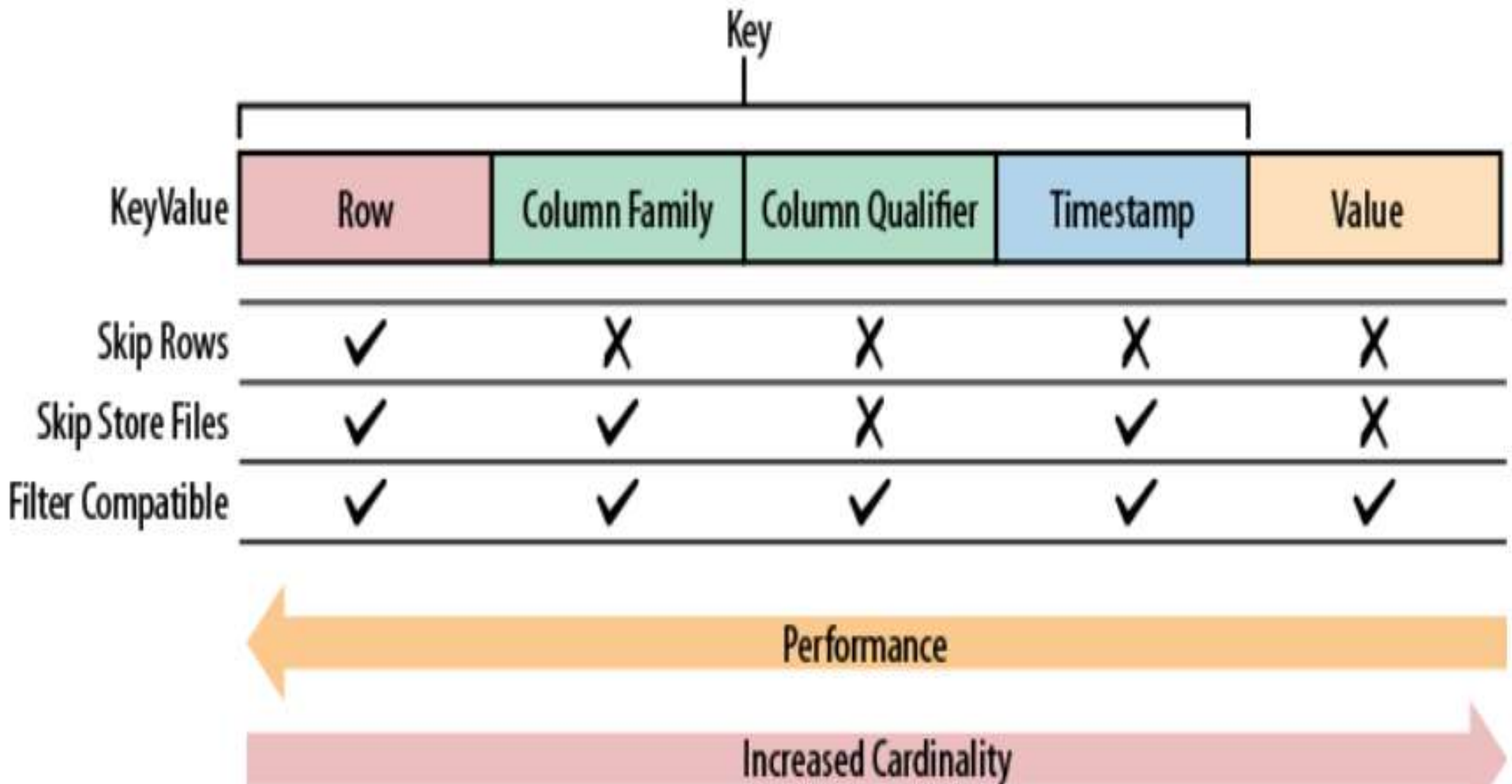
Key Design

- Two fundamental key structures
  - Row Key
  - Column Key
    - A column family name + a column qualifier
- Use these keys
  - to solve commonly found problems when designing storage solutions
- Logical V.S. Physical layout

Logical V.S. Physical layout



Read Performance and Query Criteria



Key Design – Tall-Narrow V.S. Flat-Wide Tables

- Tall-narrow table layout
  - A table with few columns but many rows
- Flat-wide table layout
  - Has fewer rows but many columns

- Tall-narrow table layout is recommended
  - Due to a single row could outgrow the maximum file/region size and work against the region split facility under Flat-wide table design

Key Design – Tall-Narrow V.S. Flat-Wide Tables

- A email system as example
  - Flat-wide layout

<userId> : <colfam> : <messageId> : <timestamp> : <email-message>

12345 : data : 5fc38314-e290-ae5da5fc375d : 1307097848 : "Hi Lars, ..."
12345 : data : 725aae5f-d72e-f90f3f070419 : 1307099848 : "Welcome, and ..."
12345 : data : cc6775b3-f249-c6dd2b1a7467 : 1307101848 : "To Whom It ..."
12345 : data : dcbee495-6d5e-6ed48124632c : 1307103848 : "Hi, how are ..."

  - Tall-narrow

<userId>-<messageId> : <colfam> : <qualifier> : <timestamp> : <email-message>

12345-5fc38314-e290-ae5da5fc375d : data : : 1307097848 : "Hi Lars, ..."
12345-725aae5f-d72e-f90f3f070419 : data : : 1307099848 : "Welcome, and ..."
12345-cc6775b3-f249-c6dd2b1a7467 : data : : 1307101848 : "To Whom It ..."
12345-dcbee495-6d5e-6ed48124632c : data : : 1307103848 : "Hi, how are ..."

Partial Key Scans

Command	Description
<userId>	Scan over all messages for a given user ID.
<userId>-<date>	Scan over all messages on a given date for the given user ID.
<userId>-<date>-<messageId>	Scan over all parts of a message for a given user ID and date.
<userId>-<date>-<messageId>-<attachmentId>	Scan over all attachments of a message for a given user ID and date.

- Make sure to pad the value of each field in composite row key, to ensure the right sorting order you expected

Partial Key Scans

- Set startRow and stopRow
  - Set startRow with exact user ID
    - *Scan.setStartRow(...)*
  - Set stopRow with user ID + 1
    - *Scan.setStopRow(...)*
- Control the sorting order
  - *Long.MAX_VALUE - <date-as-long>*
  -

```
String s = "Hello,";
for (int i = 0; i < s.length(); i++) {
  print(Integer.toString(s.charAt(i) ^ 0xFF, 16));
}
b7 9a 93 93 90 d3
```

Time Series Data

- Dealing with stream processing of event
- Most common use case is time series data
  - Data could be coming from
    - A sensor in a power grid
    - A stock exchange
    - A monitoring system for computer systems
  - Their row key represents the event time
- The sequential, monotonously increasing nature of time series data
  - Causes all incoming data to be written to the same region
  - Hot spot issue

Time Series Data

- Overcome this problem
  - By prefixing the row key with a nonsequential prefix

- Common choices
  - Salting
  - Field swap/promotion
  - Randomization

Time Series Data - Salting

- Use a salting prefix to the key that guarantees a spread of all rows across all region servers

```
byte prefix = (byte) (Long.hashCode(timestamp) % <number of regionservers>);
byte[] rowkey = Bytes.add(Bytes.toBytes(prefix), Bytes.toBytes(timestamp);
```

- Which results

```
0myrowkey-1
0myrowkey-4
1myrowkey-2
1myrowkey-5
...
```

Time Series Data - Salting

- Access to a range of rows must be fanned out
- Read with <*number of region servers*> get or scan calls

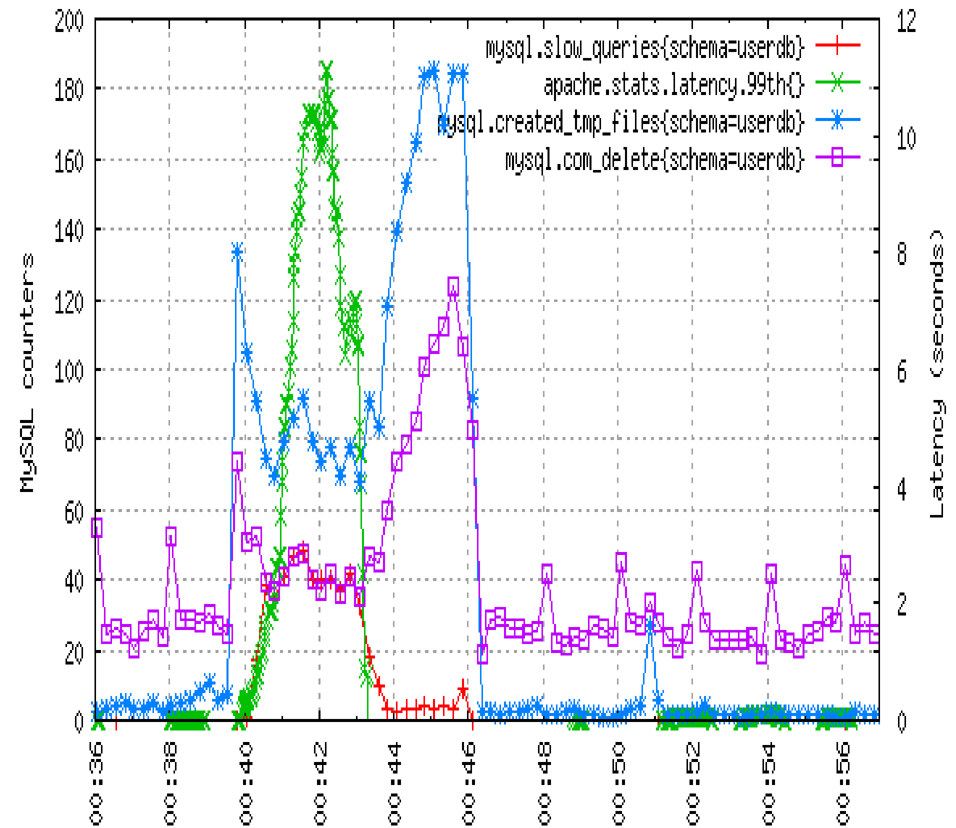
- Is it good or not good ?
  - Use multiple threads to read this data from distinct servers
  - Need more further study on the access pattern and try run

Time Series Data – Field swap/Promotion

- Use the composite row key concept
  - Move the timestamp to a secondary position in the row key
- If you already have a row key with more than one field
  - Swap them
- If you have only the timestamp as the current row key
  - Promote another field from the column keys into the row key
  - Promote even the value
- You can only access data, especially time ranges, for a given swapped or promoted field

Time Series Data – Field swap/Promotion Usecase

- OpenTSDB
  - A time series database
  - Store metrics about servers and services, gathered by external collection agents
  - All of the data is stored in HBase
  - System UI enables users to query various metrics, combining and/or downsampling them—all in real time



- The schema promotes the metric ID into the row key
  - *<metric-id><base-timestamp>...*

http://opentsdb.net/

Time Series Data – Field swap/Promotion Usecase

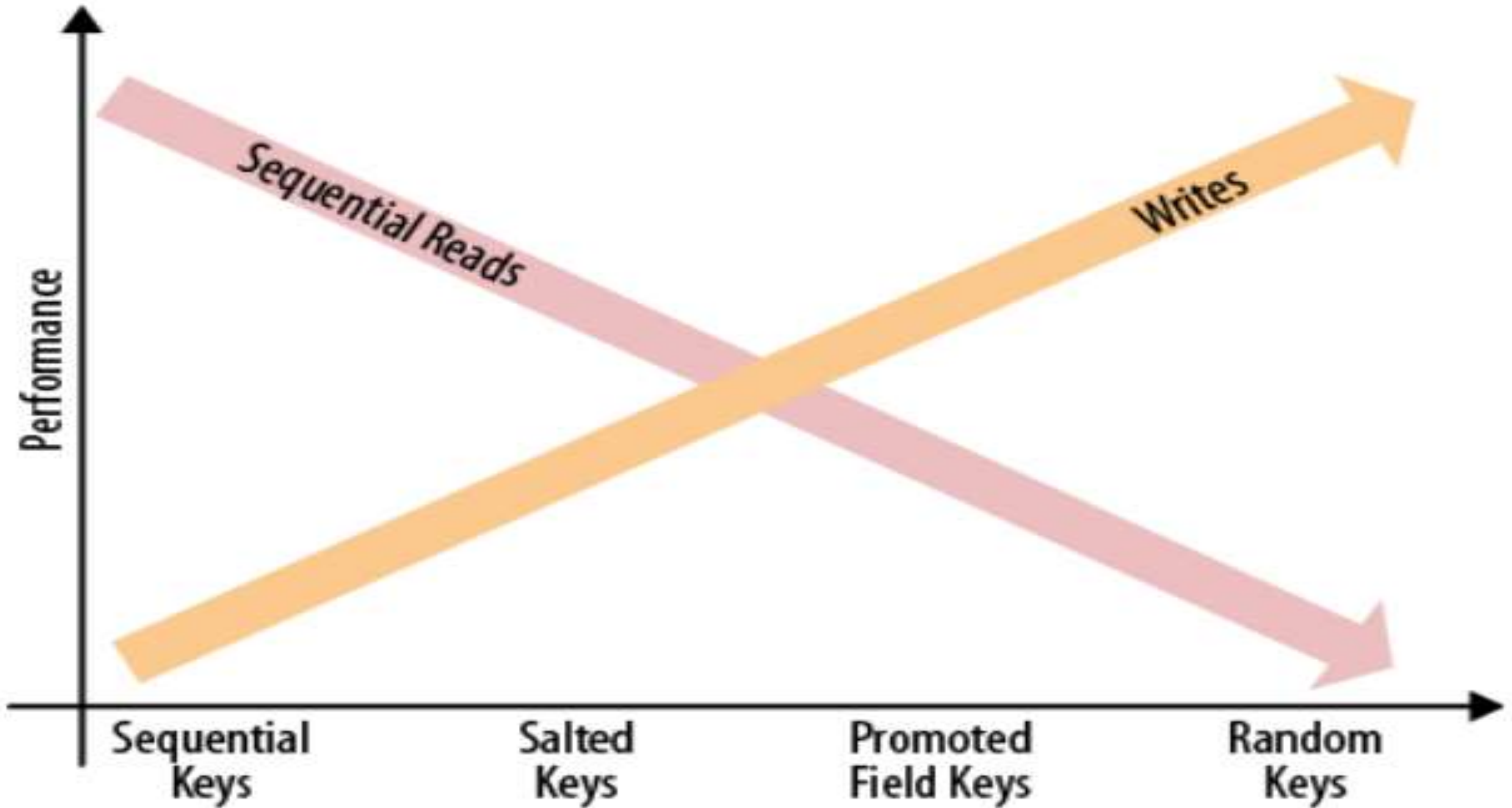
- Example

```
myservice.latency.avg 1292148123 42 reqtype=foo host=web42
```

```
[0, 0, -69, 77, 4, -99, 32, 0, 0, 1, 0, 1, 11, 0, 0, 2, 0, -7, 42]
 `-------'  `------------'  `-----'  `------'  `-----'  `-------'
 metric ID  base timestamp  name ID  value ID  name ID  value ID
                            `----------------'  `---------------'
                                first tag          second tag
```

OpenTSDB Schema - http://opentsdb.net/schema.html

Time Series Data



Time-Ordered Relations

- You can also store related, time-ordered data
  - By using the columns of a table
- Since all of the columns are sorted per column family
  - Treat this sorting as a replacement for a secondary index
  - For a small number of indexes, you can create a column family for them
    - If the large amount of indexes, you shall consider the Secondary-Indexes approaches in later of this ppt
- HBase currently (0.95) does not do well with anything above two or three column families
  - Due to flushing and compactions are done on a per Region basis
    - Can make for a bunch of needless i/o loading

http://hbase.apache.org/book/number.of.cfs.html

Time-Ordered Relations – Example

- Colum name = *<indexId> + "-" + <value>*
- Column value
  - Key in data column family
  - Redundant values from data column family for performance
    - Denormalization

```
... //data
12345 : data : 5fc38314-e290-ae5da5fc375d : 1307097848 : "Hi Lars, ..."
12345 : data : 725aae5f-d72e-f90f3f070419 : 1307099848 : "Welcome, and ..."
12345 : data : cc6775b3-f249-c6dd2b1a7467 : 1307101848 : "To Whom It ..."
12345 : data : dcbee495-6d5e-6ed48124632c : 1307103848 : "Hi, how are ..."
... //ascending index for from email address
12345 : index : idx-from-asc-mary@foobar.com : 1307099848 : 725aae5f-d72e...
12345 : index : idx-from-asc-paul@foobar.com : 1307103848 : dcbee495-6d5e...
12345 : index : idx-from-asc-pete@foobar.com : 1307097848 : 5fc38314-e290...
12345 : index : idx-from-asc-sales@ignore.me : 1307101848 : cc6775b3-f249...
...// descending index for email subjects
12345 : index : idx-subject-desc-\xa8\x90\x8d\x93\x9b\xde : \
 1307103848 : dcbee495-6d5e-6ed48124632c
12345 : index : idx-subject-desc-\xb7\x9a\x93\x93\x90\xd3 : \
 1307099848 : 725aae5f-d72e-f90f3f070419
```

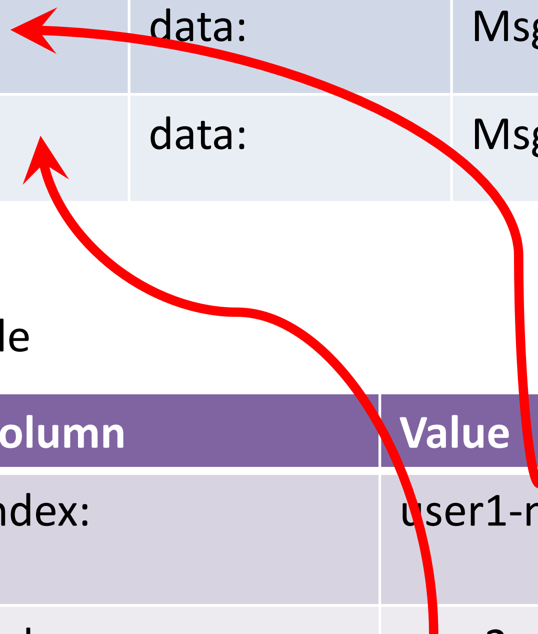
Secondary Indexes

- HBase has <u>no native support</u> for secondary indexes
  - There are use cases that need them
  - Look up a cell with not just the primary coordinates
    - The row key, column family name and qualifier
  - But also an alternative coordinate
    - Scan a range of rows from the main table, but ordered by the secondary index
- Secondary indexes store a <u>mapping between the new coordinates and the existing ones</u>

Secondary Indexes – Sample I

Data Table

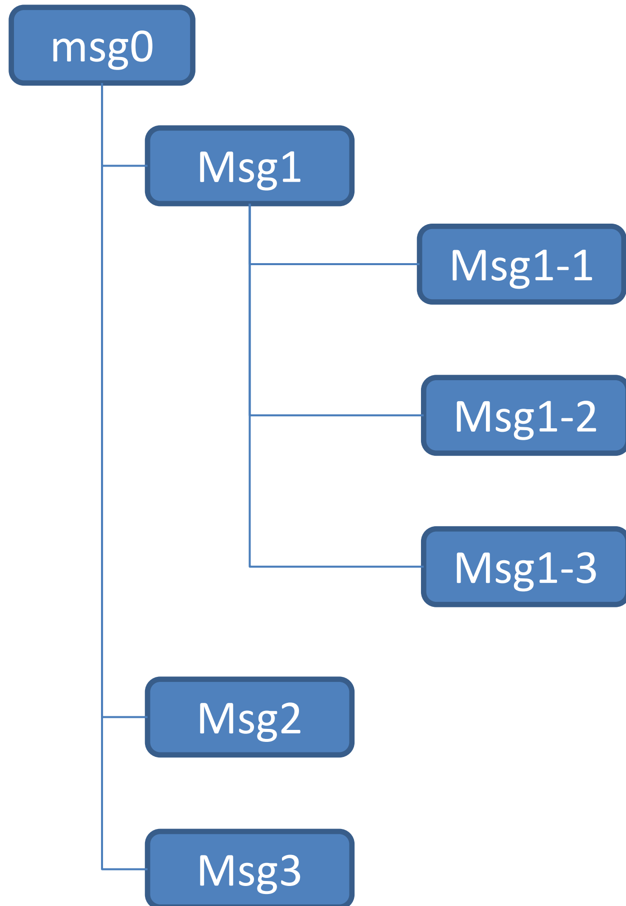
RowKey	Column	Value
user1-msg1	data:	Msg…
user2-msg2	data:	Msg…



Sender Receiver Index Table

RowKey	Column	Value
user1-user2-timetamp	index:	user1-msg1
user2-user3-timestamp	index:	user2-msg2

Secondary Indexes – Sample II



RowKey	Column	Value
msg0-1msg1	data:	usern-msgn
msg0-1msg1-1msg1-1	data:	usern-msgn
msg0-1msg1-2msg1-2	data:	usern-msgn
…	data:	usern-msgn
msg0-2msg2	data:	usern-msgn

Secondary Indexes - Client-managed

- Moving the responsibility into the application layer

- Combines a data table and one (or more) lookup/mapping tables

- Write data
  - Into the data table, also updates the lookup tables

- Read data
  - Either a direct lookup in the main table
  - A lookup in secondary index table, then retrieve data from main table

Secondary Indexes - Client-managed

- Atomicity
  - No cross-row atomicity
  - Writing to the secondary index tables first, then write to the data table at the end of the operation
  - Use asynchronous, regular pruning jobs

- It is hardcoded in your application
  - Needs to evolve with overall schema changes, and new requirements

Secondary Indexes - Coprocessor

- Implement an indexing solution based on coprocessors
  - Using the server-side hooks, e.g. *RegionObserver*
  - Use coprocessor to load the indexing layer for every region, which would subsequently handle the maintenance of the indexes
  - Use of the scanner hooks to transparently iterate over a normal data table, or an index-backed view on the same
  - Currently in development
- JIRA ticket
  - https://issues.apache.org/jira/browse/HBASE-2038