HBase... And Lewis Carroll!

jwfbean@cloudera.com

Twitter, LinkedIn: @jwfbean

Introduction

- 2010: Cloudera Solutions Architect
- 2011: Cloudera TAM/DSE
- 2012-2013: Cloudera Training focusing on Partners and Newbies
- 2H/2013: Partner Engineering focusing on ISV certifications

- Prior experience as an SE in Business Intelligence, Middleware and Data Integration

Our Discussion

- Crash Coure in HBase
- HBase Best Practices (from Lewis Carroll)

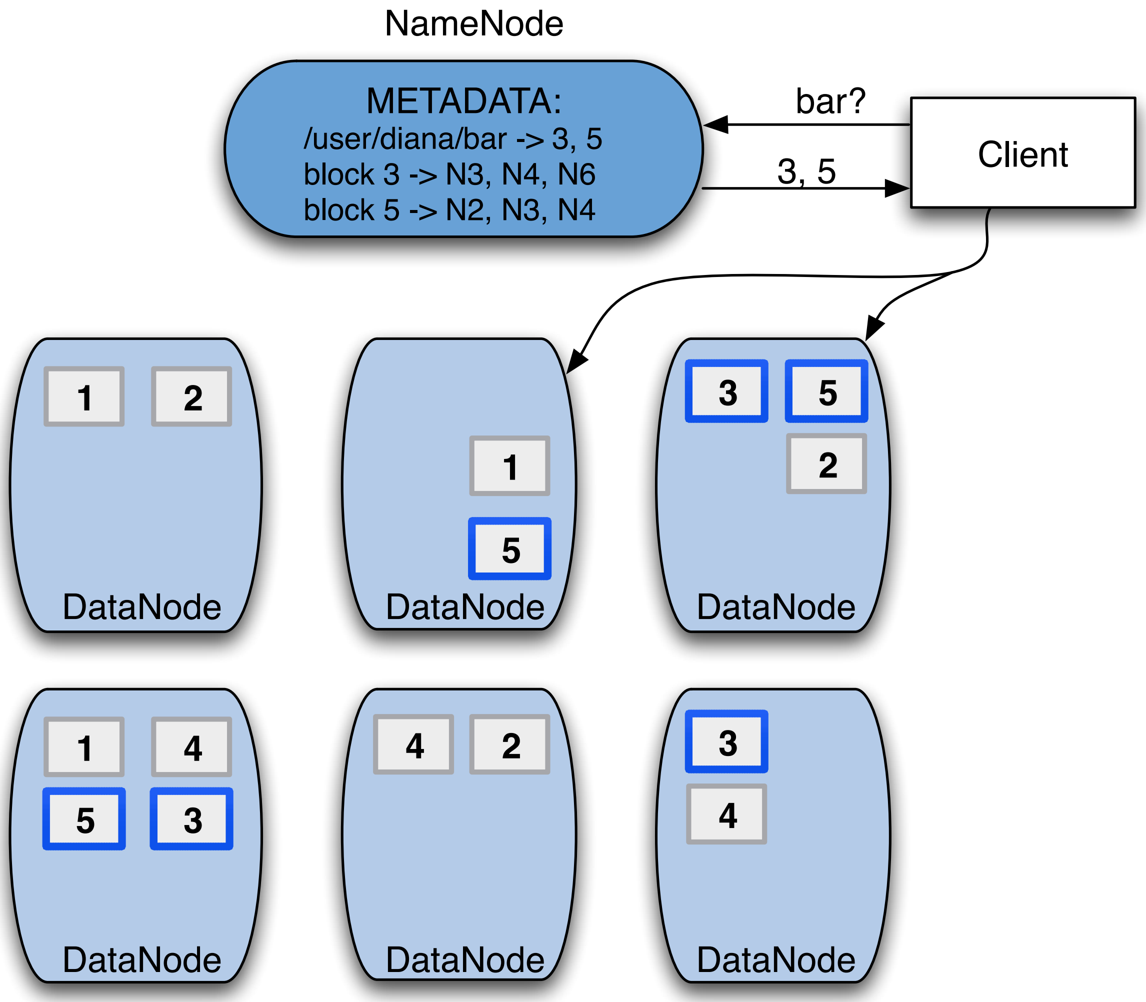
- Goal: A broad understanding of HBase
- A feeling for the tradeoffs and considerations

“The Three V’s”

More…

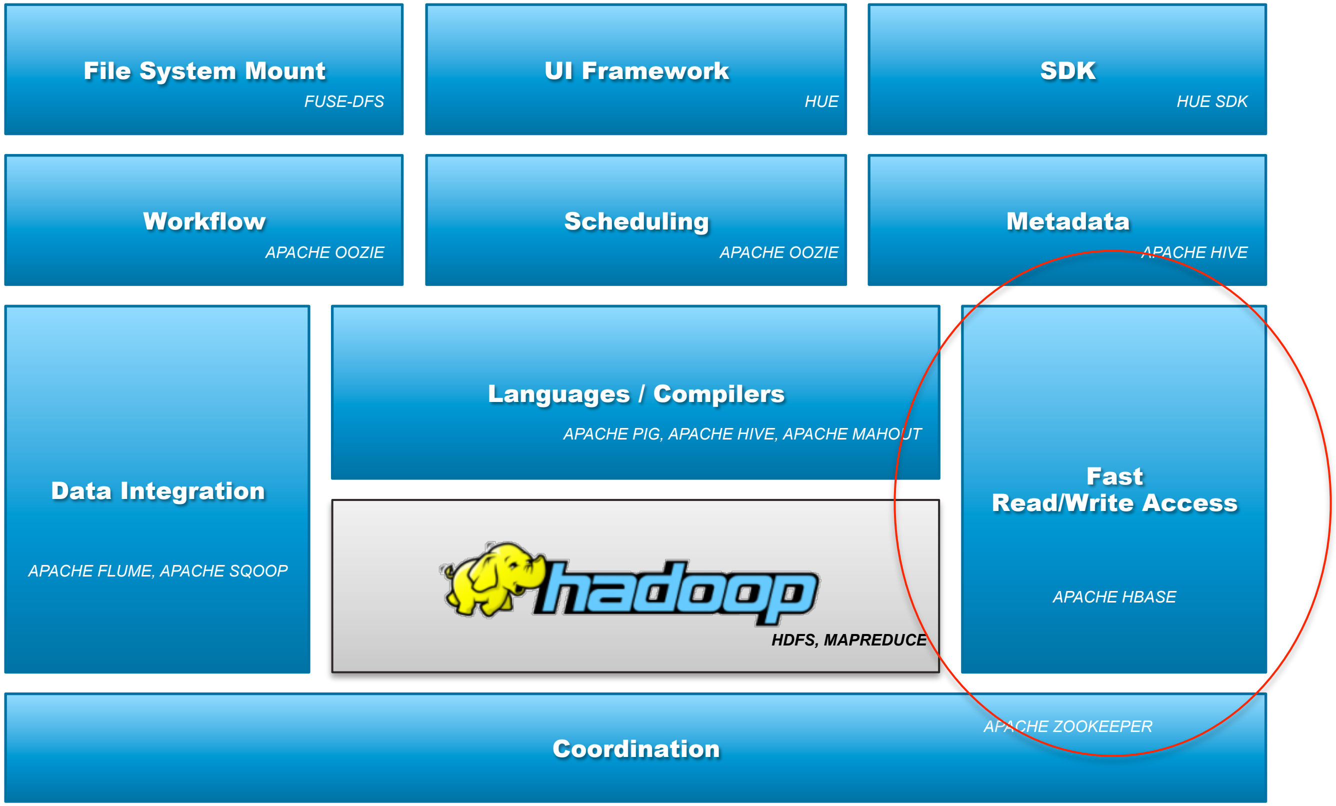
- Volume: Data than ever. (Bottlenecks and costly storage)
- Variety: Types of data than ever. (Expensive/ ineffective to model/schema)
- Velocity: Faster than ever. (Hard to capture, move, analyze in a timely way)

Crash course in HDFS



- Big blocks (64MB)
- Write once
- Batch Parallel processing via MapReduce
- Schema on Read

Cloudera's Distribution including Apache Hadoop (CDH)



File System Mount
FUSE-DFS

UI Framework
HUE

SDK
HUE SDK

Workflow
APACHE OOZIE

Scheduling
APACHE OOZIE

Metadata
APACHE HIVE

Data Integration
APACHE FLUME, APACHE SQOOP

Languages / Compilers
APACHE PIG, APACHE HIVE, APACHE MAHOUT

hadoop
HDFS, MAPREDUCE

Fast Read/Write Access
APACHE HBASE

Coordination
APACHE ZOOKEEPER

What is HBase

- Apache Managed Open Source Project
- Sparse, Multidimensional, Sorted Map
- Based on Google’s “Big Table” paper
- Implemented on top of HDFS
    - Linearly Scalable
    - Fault Tolerant

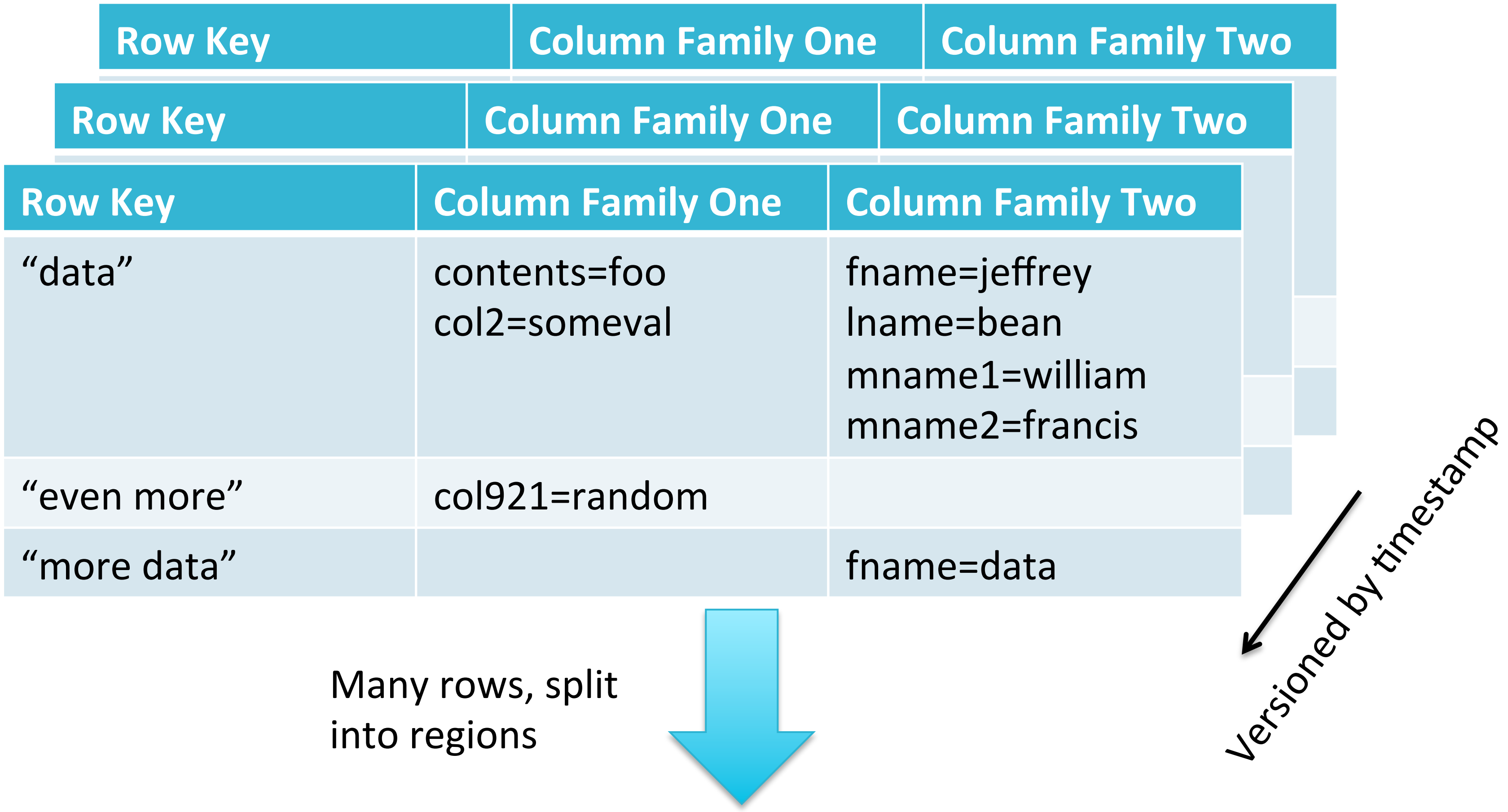
- Strongly Consistent
- Uses a “Log Structured Merge Tree”

When to use HBase

- When you need random access
- To huge data sets with huge concurrency
- With a well-defined access pattern

- (or a scalable cache)

An HBase Table



Row Key	Column Family One	Column Family Two

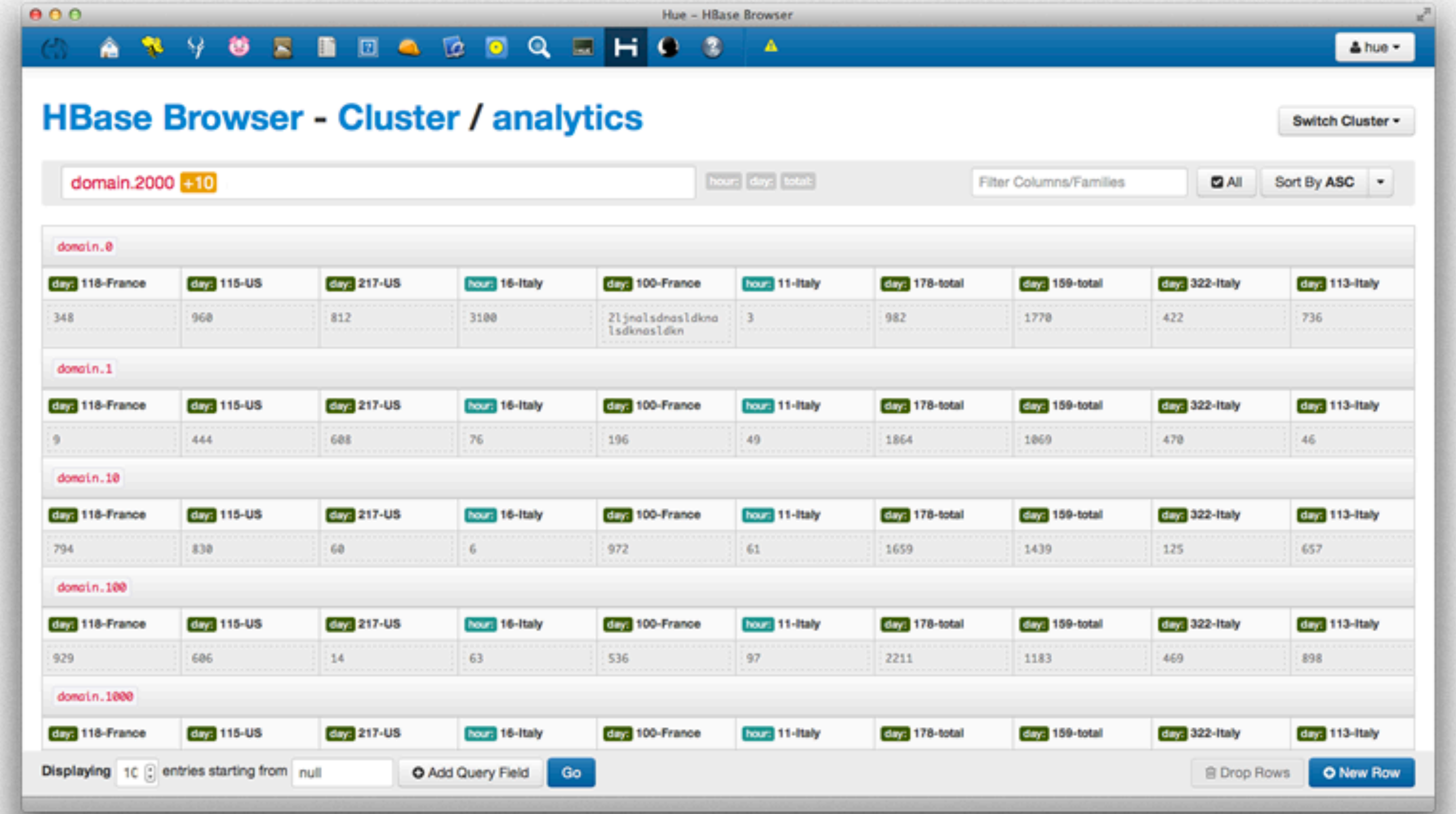
“data”	contents=foo
	col2=someval
		fname=jeffrey
		lname=bean
		mname1=william
		mname2=francis

“even more”	col921=random	
“more data”		fname=data

Many rows, split into regions

Versioned by timestamp

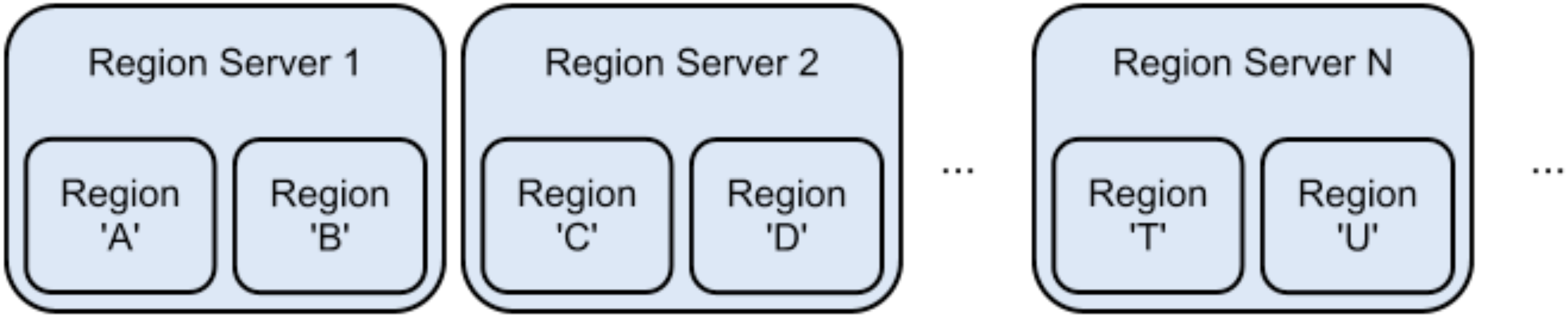
Visualized by Hue



HBase Browser - Cluster / analytics

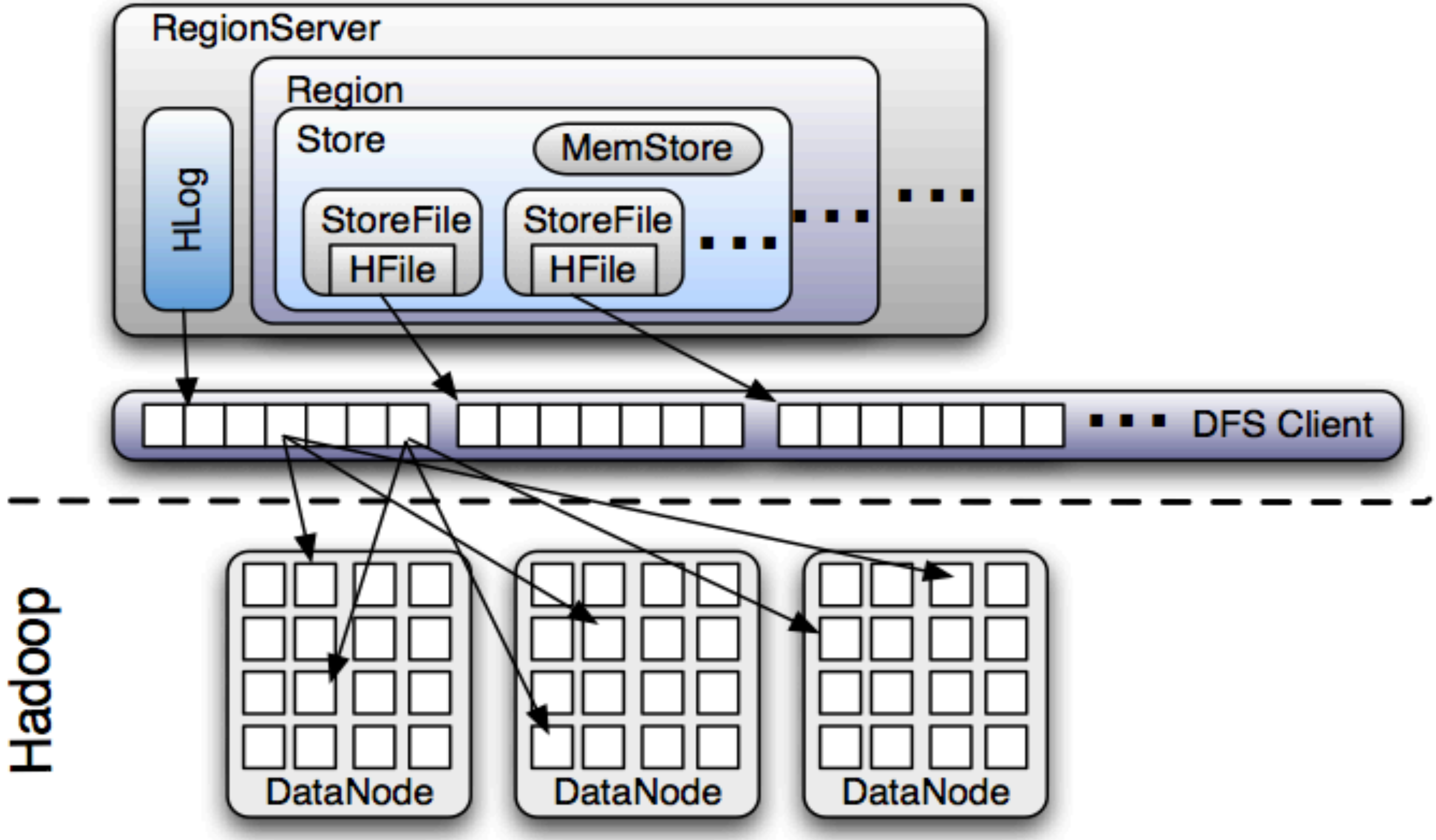
	day: 118-France	day: 115-US	day: 217-US	hour: 16-Italy	day: 100-France	hour: 11-Italy	day: 178-total	day: 159-total	day: 322-Italy	day: 113-Italy
domain.0	348	960	812	3100	2ljnalsdnasldkna lsdknasldkn	3	982	1770	422	736
domain.1	9	444	608	76	196	49	1864	1069	470	46
domain.10	794	830	60	6	972	61	1659	1439	125	657
domain.100	929	606	14	63	536	97	2211	1183	469	898
domain.1000										

Tables distributed across regions (AKA “shards”)

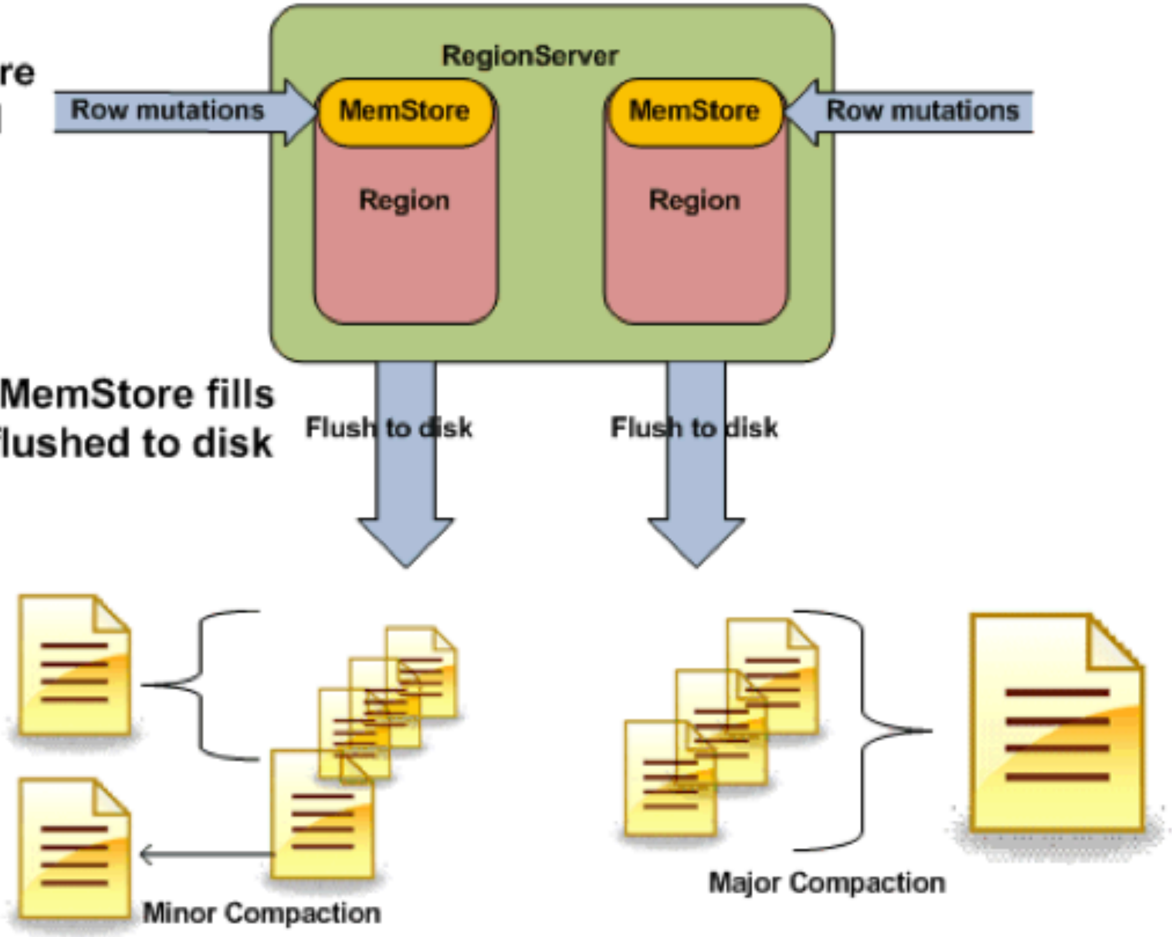


From kiji.org

RegionServers and HDFS



Flushes and Compactions



1. Rows are updated

2. When MemStore fills up, it is flushed to disk

3. Several small store files are compacted together

4. Each day the store files are compacted into a single file and cleaned up

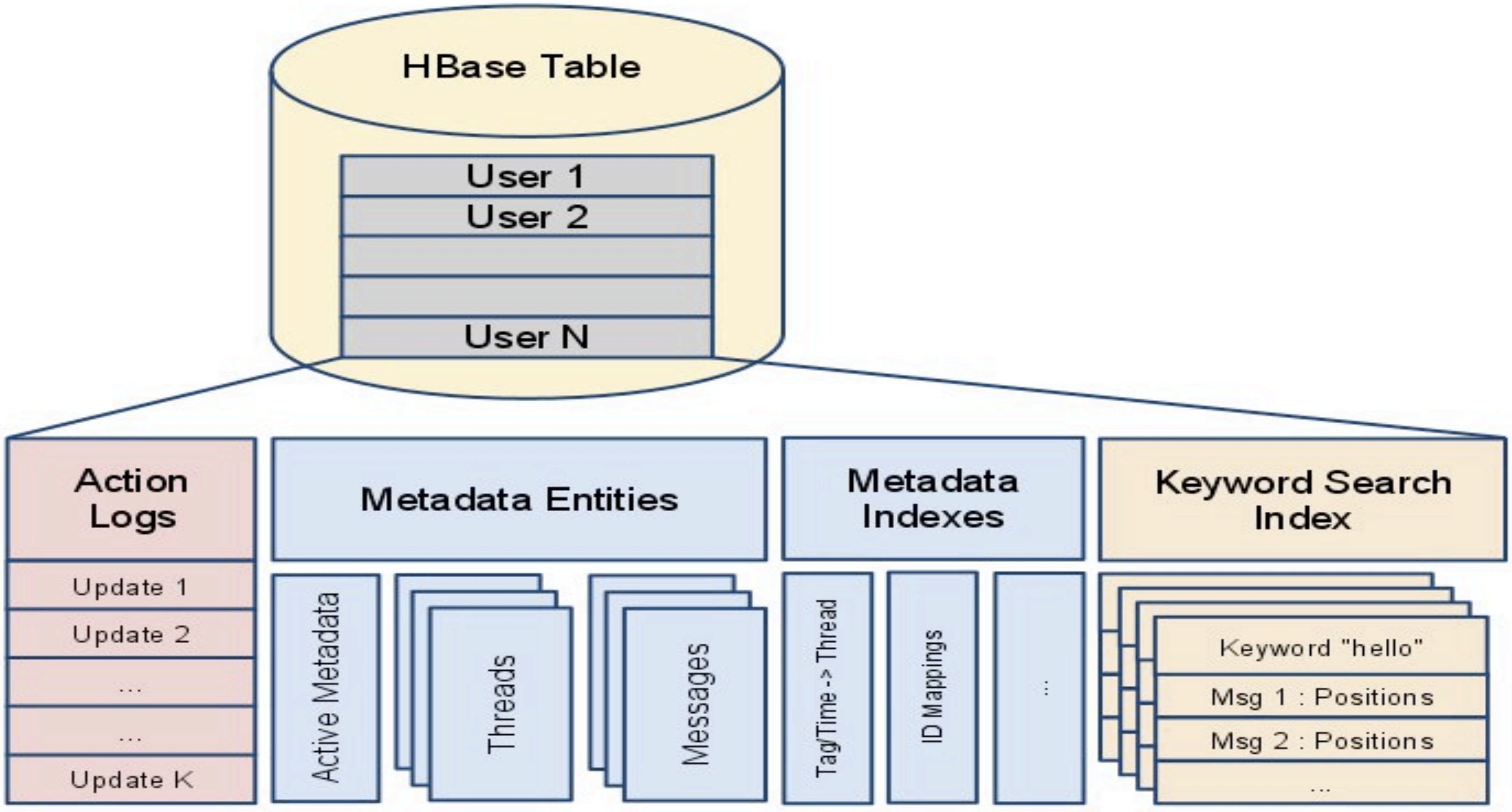
Getting at Data

- Very constrained query semantics (get/put/scan)
- Data stored as un-typed byte arrays
- Lexicographically sorted by row key
- Co-accessed data co-located by column family
- Java API, HBase shell, REST, Thrift, HUE

- Hive or Impala support for ad hoc query

A REAL HBase table

https://www.facebook.com/notes/facebook-engineering/inside-facebook-messages-application-server/10150162742108920



We get questions like...

- "Can HBase handle 2 million queries per second?"
- "Can I have some HBase performance numbers?"
- "Can I have sub-second query response?"
- "What's the maximum write throughput supported by HBase?"
- "Can HBase serve 500,000 concurrent queries over 4 petabytes of data?"

Or questions like...

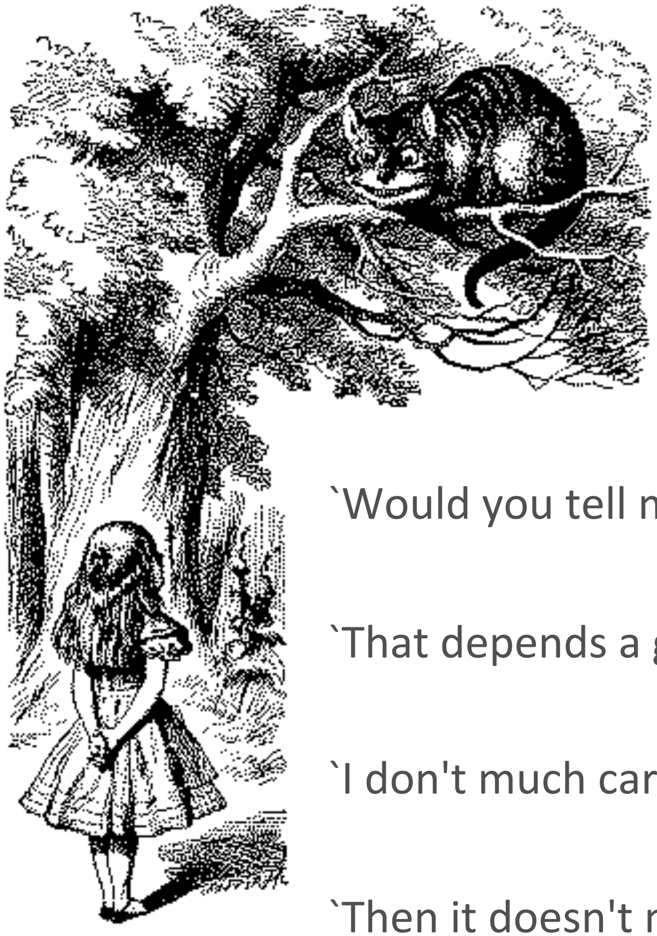
- What's the right value for:
  - hbase.hregion.max.filesize?
  - hbase.hregion.memstore.block.multiplier?
  - hbase.hregion.memstore.flush.size?
  - hbase.hstore.compaction.threshhold?
  - Java heap size for the regionserver process?

We get questions like...

- How big should my cluster be?
- What kind of nodes should I use?
- What column famillies do I need?
- What should my row key be?

And the answer...?

- It depends!



`Would you tell me, please, which way I ought to go from here?'

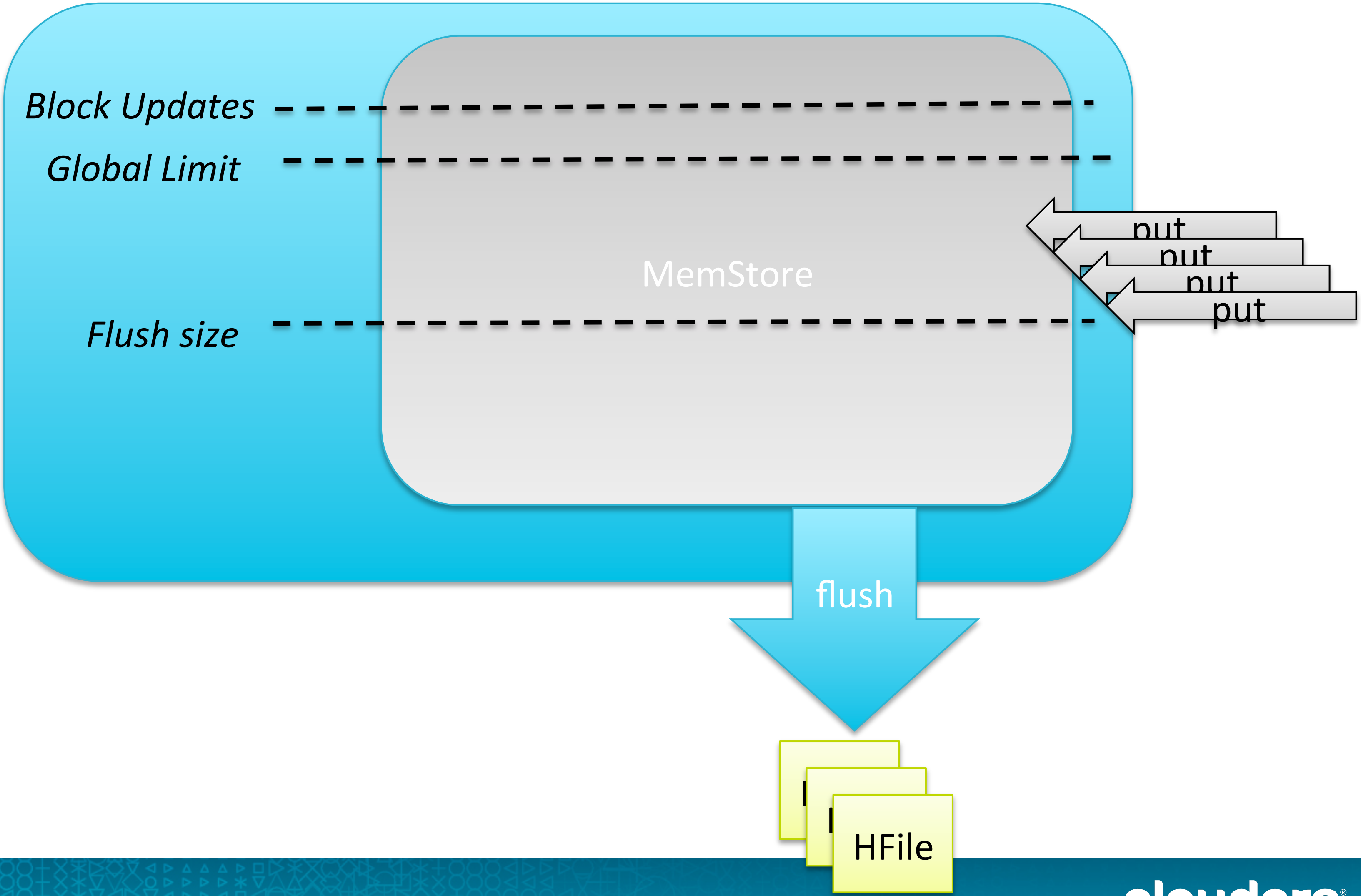
`That depends a good deal on where you want to get to,' said the Cat.

`I don't much care where--' said Alice.

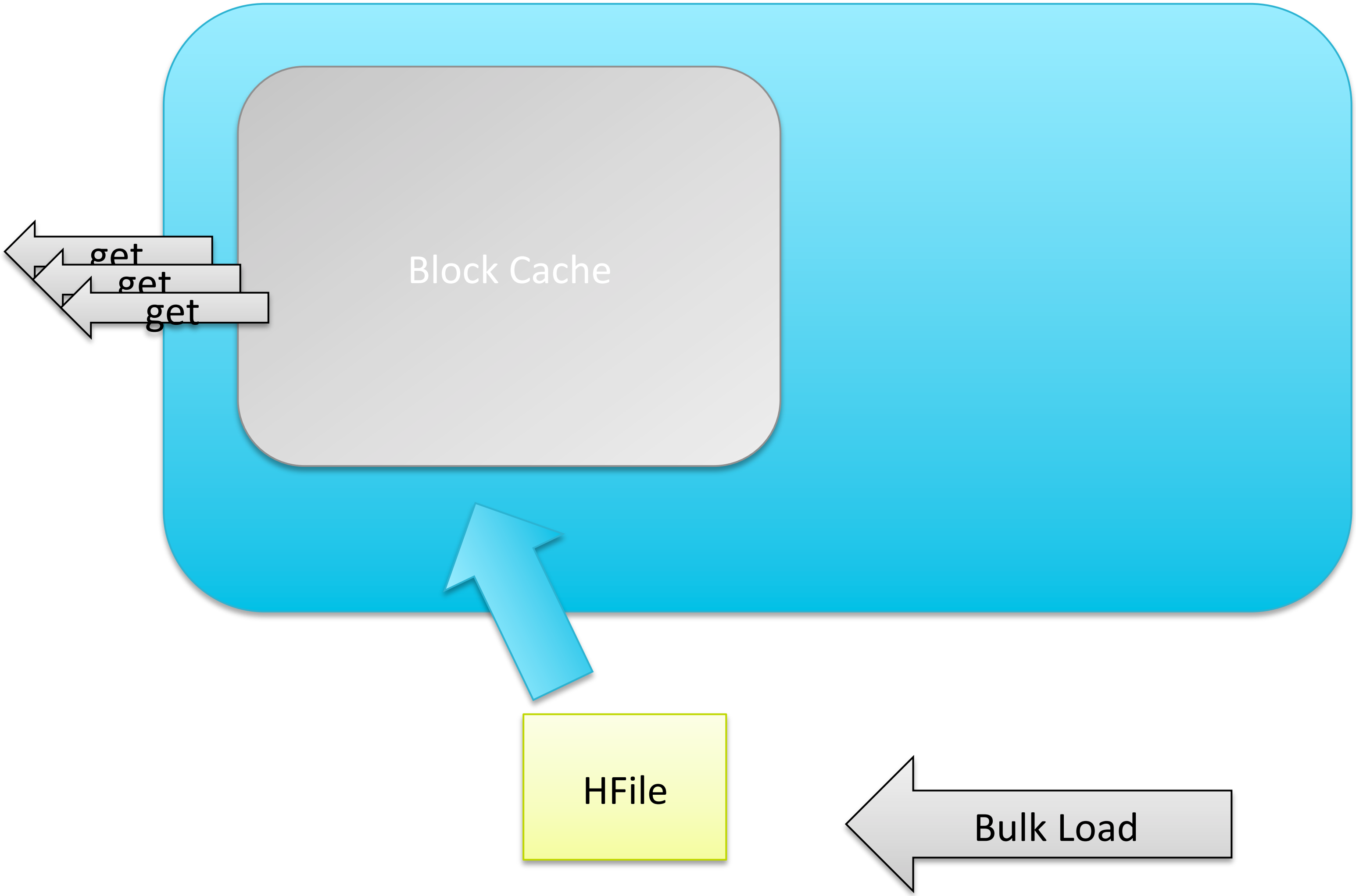
`Then it doesn't matter which way you go,' said the Cat.

Tune for the workload!

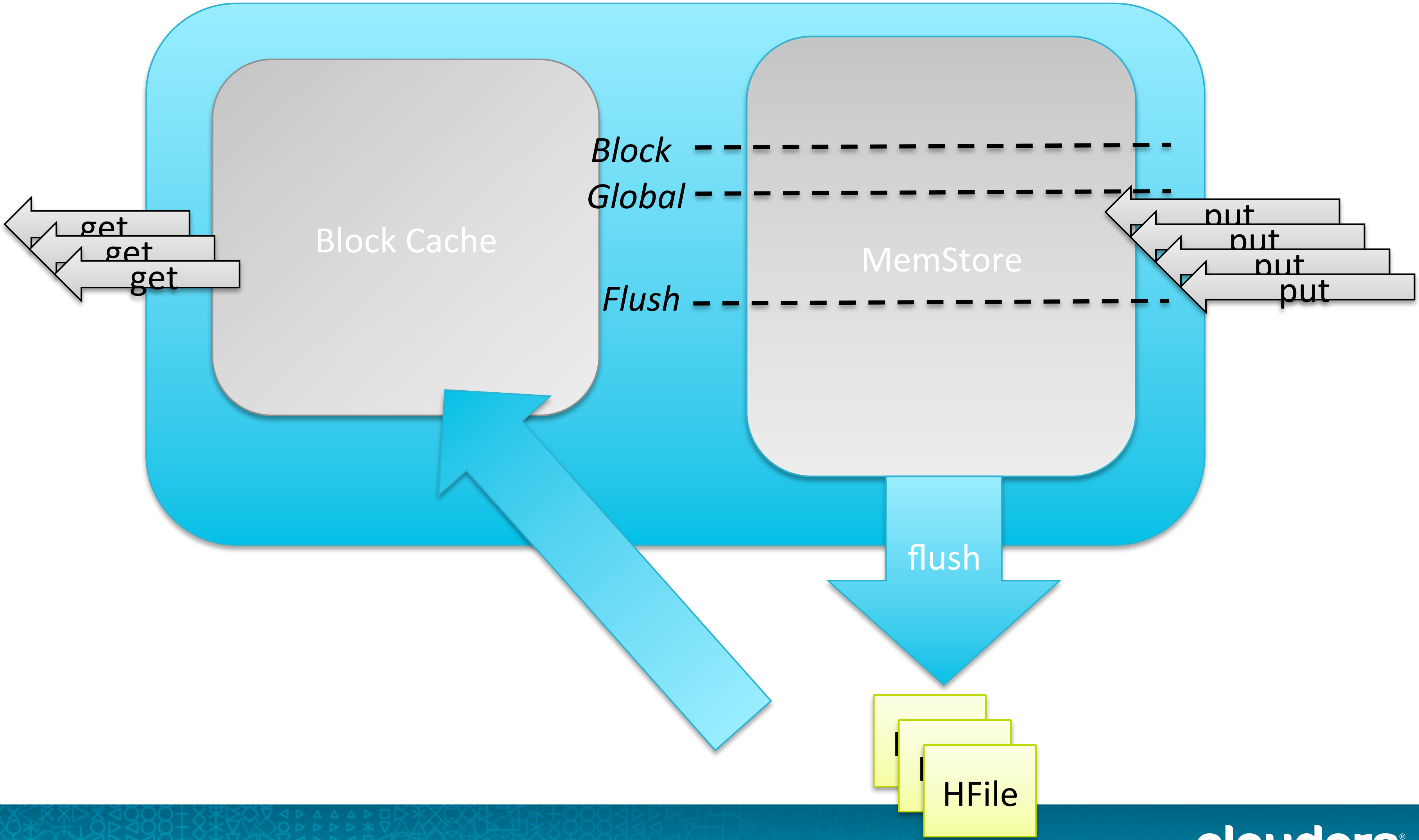
Write heavy workload: bigger memstore



Read Heavy Workload: bigger block cache

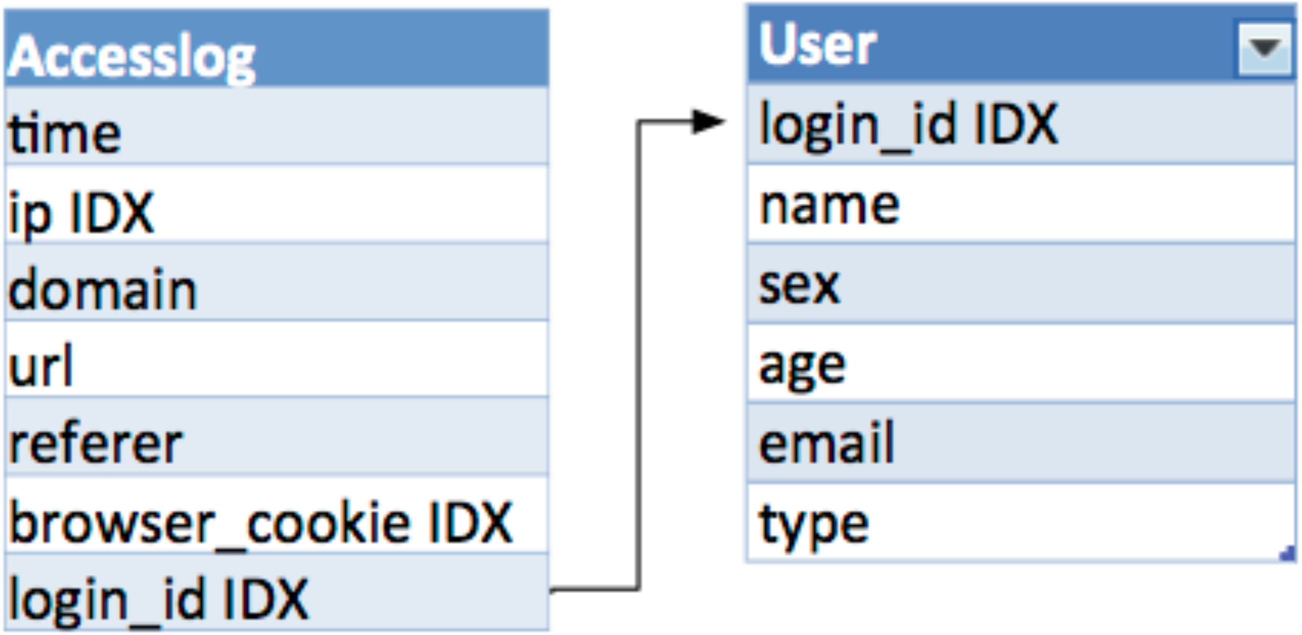


Mixed Workload: Tuned



Design Schema for Access Pattern

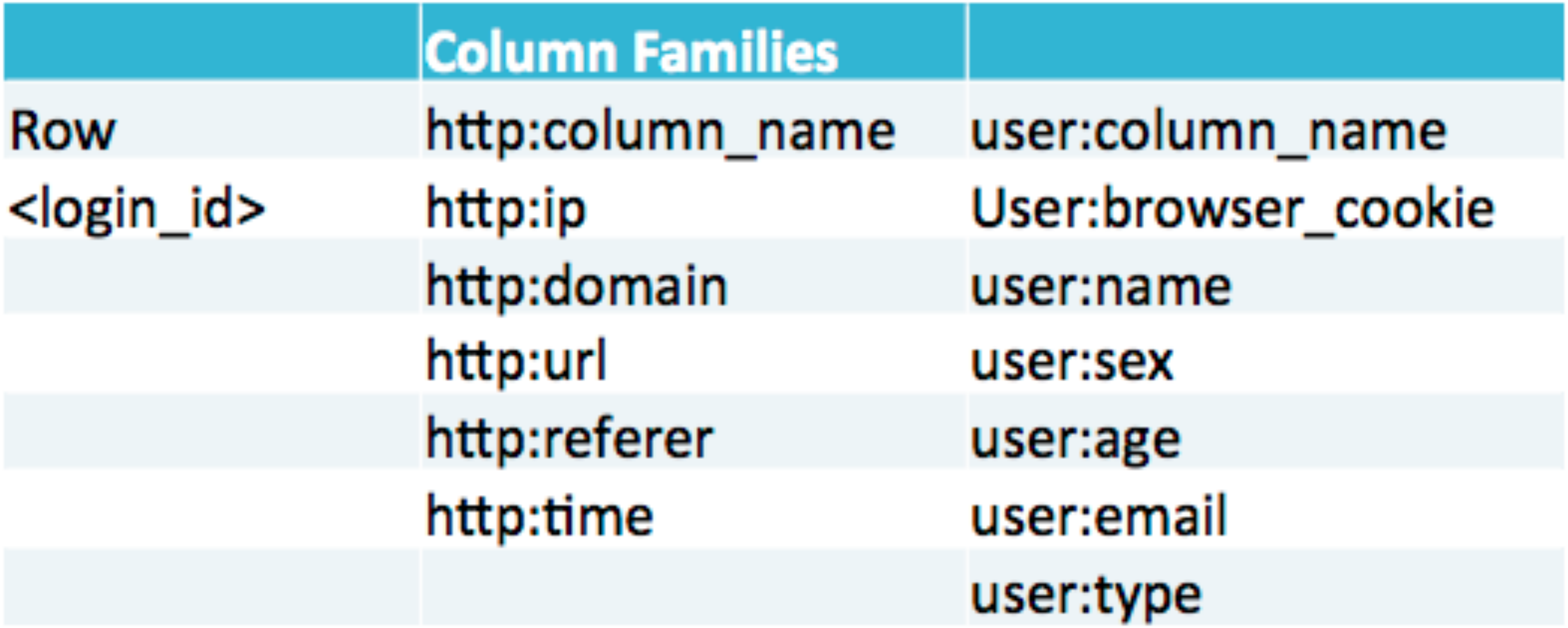
Web Clicks Schema: An RDBMS



Accesslog
time
ip IDX
domain
url
referer
browser_cookie IDX
login_id IDX

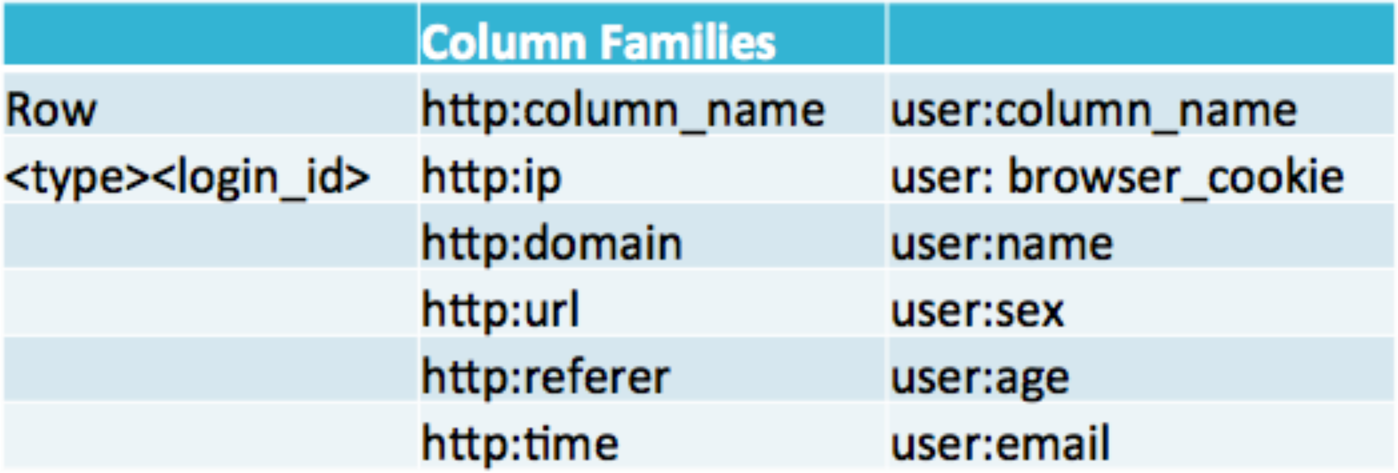
User
login_id IDX
name
sex
age
email
type

Choosing a Row Key and Column Families



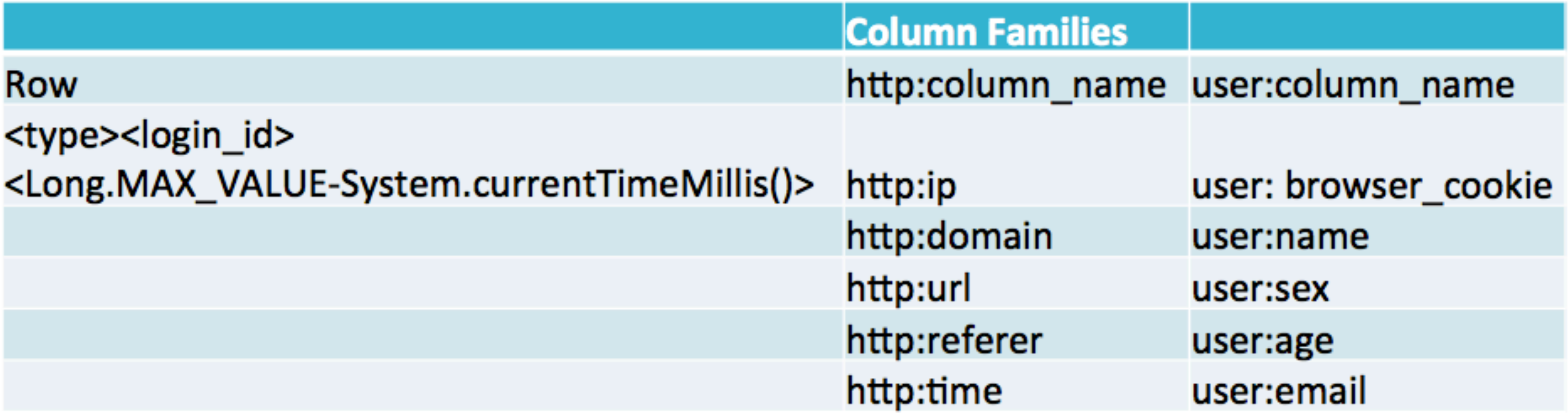
	Column Families	
Row	http:column_name	user:column_name
<login_id>	http:ip	User:browser_cookie
	http:domain	user:name
	http:url	user:sex
	http:referer	user:age
	http:time	user:email
		user:type

Avoiding Hot Spots with Promoted Field Key



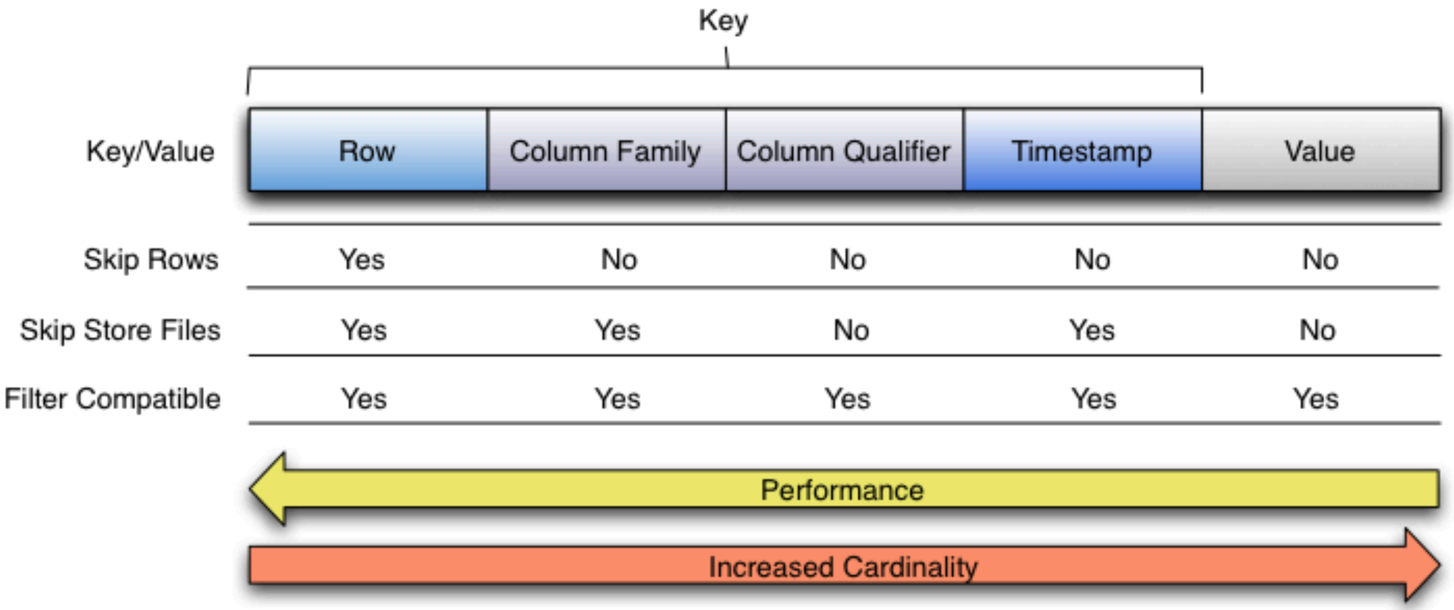
	Column Families	
Row	http:column_name	user:column_name
<type><login_id>	http:ip	user: browser_cookie
	http:domain	user:name
	http:url	user:sex
	http:referer	user:age
	http:time	user:email

Controlling Display order by Reverse Timestamp



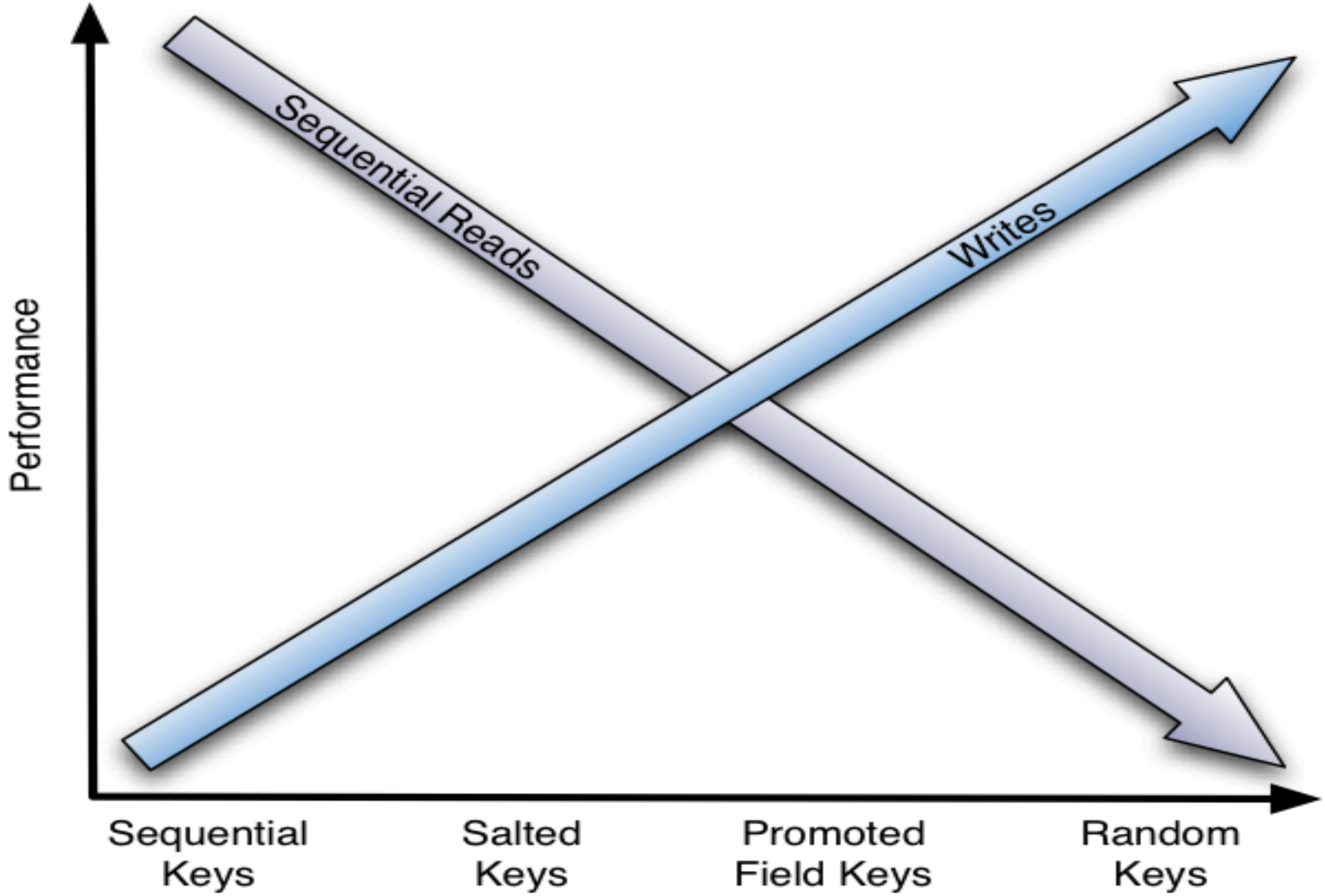
	Column Families	
Row	http:column_name	user:column_name
<type><login_id> <Long.MAX_VALUE-System.currentTimeMillis()>	http:ip	user: browser_cookie
	http:domain	user:name
	http:url	user:sex
	http:referer	user:age
	http:time	user:email

Design Schema for Performance

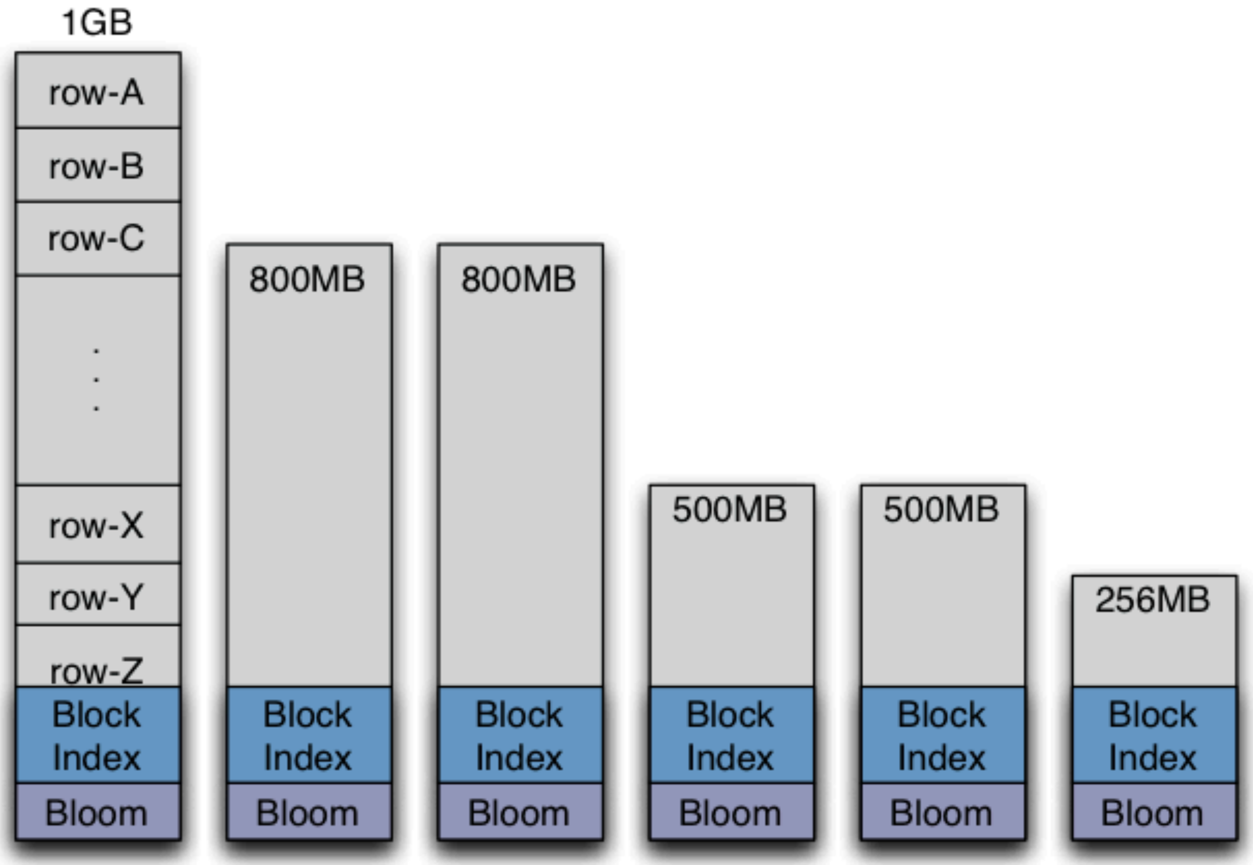


	Key				
Key/Value	Row	Column Family	Column Qualifier	Timestamp	Value
Skip Rows	Yes	No	No	No	No
Skip Store Files	Yes	Yes	No	Yes	No
Filter Compatible	Yes	Yes	Yes	Yes	Yes

Design Schema for Performance



Bloom Filters Help Reads...



dex	Yes	Yes	Yes	Yes	Yes	Yes	-> 6 seeks/blocks loade
ilter	No	No	Maybe	No	No	Maybe	-> 2 seeks/blocks loade

…Unless you frequently update most rows

Additional Tradeoffs

Tall/Narrow tables

- Split efficiently
- Logical rows span physical rows

- Good for scans

- Generally recommended

But... Flat-Wide Tables

- ... Are good for random gets

Cell size

- Big Cells: Grow HBase “block size”

- But that breaks performance on small cells

Other Concerns

Don't Colocate MapReduce and HBase

- ...Unless it's to read or write from HBase

Do automatic major compaction...

- ...But not as specified in the default

Split regions…

- …But not automatically, as is default

Additional Considerations

- WAL on puts trades performance for durability
- Random gets v sequential scans affect cache considerations
- Region size and cluster size affect query throughput

Improper choices...

"In that direction lives instability: and in that direction, lives unavailability. Visit either you like. They're both mad."

So the recommendation?

- Test environment matches production
- Test suite matches application
- Expect to iterate
- Expect to redesign
- Use Cloudera Manager to detect suboptimal conditions

- HBase is dependent upon the application

Lastly

- A good use case
- Properly configured
- With a well understood application
- Will scale!
- Ad hoc query won't (use Impala).

More information…

- HUE: http://gethue.com
- Oreilly's *HBase, the Definitive Guide* by Clouderan Lars George
- Cloudera University: http://university.cloudera.com/
- http://hbase.apache.org and mailing lists
- Cloudera Forums
- HBase at eBay, Facebook, StumbleUpon

cloudera®

Ask Bigger Questions