Hadoop’s ecosystem

1000 players vie for attention

What is Hadoop?

- Hadoop Common – Libraries and utilities
- Hadoop Distributed File System (HDFS) – A distributed file-system
- Hadoop YARN – A resource-management platform
- Hadoop MapReduce – A programming model for large scale data processing

What is Spark?

- A programming model for large scale data processing
- Doesn’t need Hadoop to run
- Can use HDFS, S3, Cassandra, MongoDB and Swift
- Is Spark running on AWS and S3 still part of Hadoop’s ecosystem?

Things we really need

- **A processing engine**
- **SQL queries for semi-structured and structured data**
- **Access to our favorite analytic tools**
  - R
  - machine learning models
  - data mining tools
  - graph algorithms
- **Ingest from multiple sources**
  - IOT
  - Scientific results
  - RDBMS
- **Somebody else to handle the hardware and setup...**

Processing engines for Hadoop

- **MapReduce (MR2)** - Batch, cold data, best scaling.
- **Spark** - Batch and microbatch. Innovative, unstable.
- **Flink** - Batch and streaming. Stable, compatible with Spark and MR2. Bank-worthy.
- **Storm** - Very small code-base. Simple, performant processing engine, covered in IOT course.

4 open source ways to use SQL with Hadoop

- **Apache Hive** - works with **Spark** or **MR2**
- **Apache SparkSQL** - works with **Spark**, reuses parts of **Hive**
  - Introduction next week
- **Apache Phoenix -** SQL for **HBase** (SQL for NoSQL)
  - Designed for OLTP
  - SQL interface to HBase
  - History: started at SalesForce, Apache project since 2014
- **Apache Drill -** based on **Google Dremel.** Source agnostic: different query interfaces for different data stores

6 proprietary SQL interfaces for Hadoop

- **Cloudera Impala** - Rewrite of Dremel/Drill
- **Facebook Presto** - Drill-like, on top of Hive and Cassandra
- **Hortonworks Stinger** - Hive on drugs
- **Pivotal HAWQ** - Proprietary, supposedly fast.
- **Oracle Big Data SQL** - Drill-like. Works with Oracle 12c and up.
- **IBM BigSQL** - Works with IBM's Hadoop, InfoSphere BigInsights.

Access to analytics

R: RHadoop, Rhipe, Hadoop Streaming with R, SparkR

Machine learning:

- **Mahout - machine learning for MR2**
- **H2O - nodes/contexts provide access H2O libraries**
  - H2O nodes are started by MR2 Mappers
  - H2O contexts are started by Spark executors
- **Spark MLlib - machine learning for Spark**
- **Flink ML - machine learning for Flink**

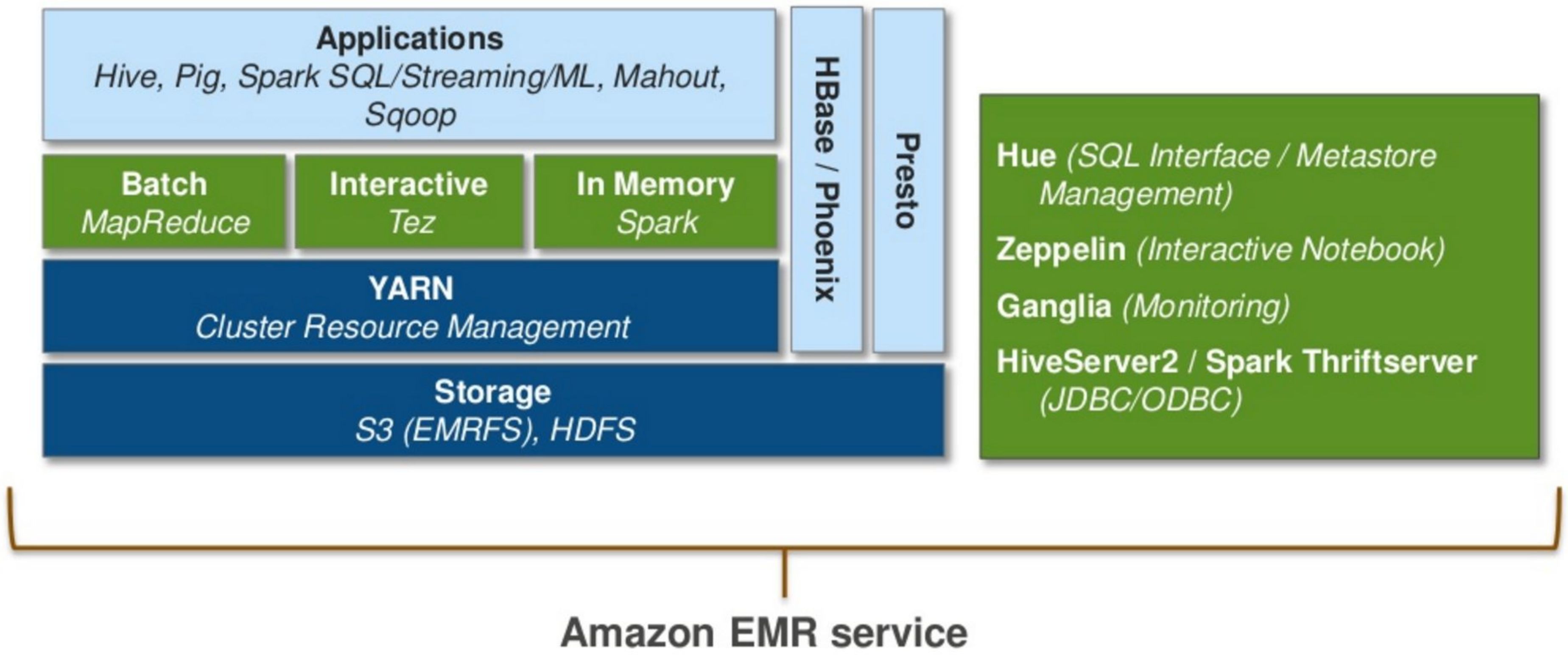
Graph algorthms:

- **Giraph (from Pregel)**
- **Titan and Faunus (Aurelius/DataStax)**
- **GraphX (with GraphFrames on Spark)**
- **Oracle Big Data spatial and graph analysis (on HBase)**

Ingest from multiple sources

- **RDBMS: Sqoop**
- **Multiple sources: Apache Flume**
- **Streaming source:**
    - Kafka
    - SparkStreaming
    - Flink
- **OLAP: Apache Kylin - SQL UI, HBase**

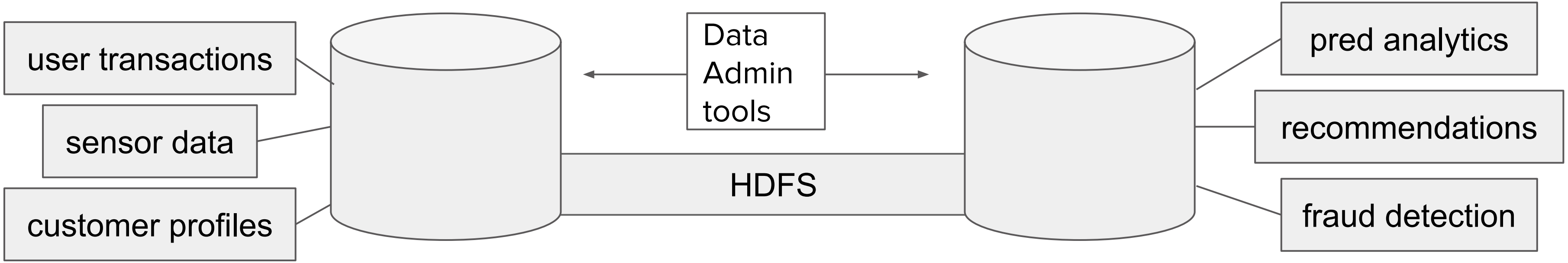
The AWS ecosystem for Hadoop



NoSQL: HBase

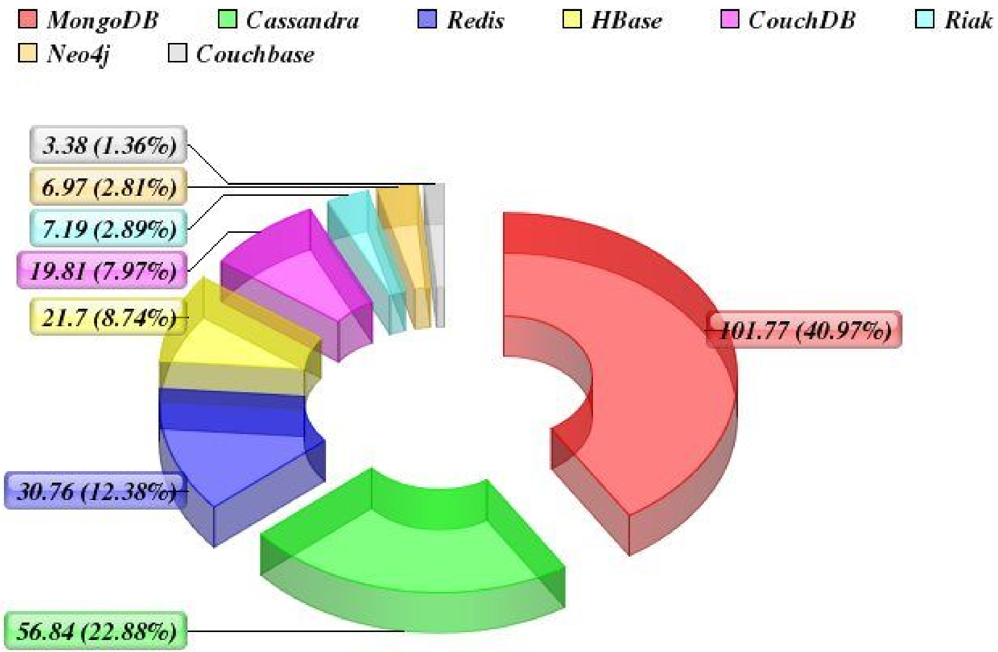
Using sparse, distributed, persistent multidimensional sorted maps.

incremental, horizontal scaling
varying/changing data formats



NoSQL	Hadoop
transactions	immutable data
real-time/interactive	batch/micro-batch
fast reads/writes	massive compute power

Many available NoSQL databases



A quick survey of lists on the web: 50+ NoSQL databases

Hive and MongoDB

```
CREATE EXTERNAL TABLE stock_prices
(
id STRUCT,
      Symbol STRING,
      Date STRING,
      Open DOUBLE,
      High DOUBLE,
      Low DOUBLE,
      Close DOUBLE,
      Volume INT
)
STORED BY 'com.mongodb.hadoop.hive.MongoStorageHandler'
WITH SERDEPROPERTIES('mongo.columns.mapping'='{"id":"_id","Symbol":"Symbol", "Date":"Date",
"Open":"Open", "High":"High", "Low":"Low", "Close":"Close", "Volume":"Volume"}')
TBLPROPERTIES('mongo.uri'='mongodb://localhost:27017/stocks.stock_prices');
```

Building a movie recommendation system.

- Find the entire example on github
- Download MongoDB from mongodb.com
- Download Spark from here
- Read the MongoDB-Spark connector documentation
- Sign up for the MongoDB University Course

Cassandra and Hadoop

Vendor Datastax positions Cassandra *vs* Hadoop.

History: developed at Facebook for inbox search. Facebook now uses HBase for search indexing.

Features:

- Cassandra has great performance
- Cassandra is easier to use and administer than HBase
- However, it is not integrated with Hadoop
    - Often used in architectures that also incorporate Hadoop
    - Because it doesn't use HDFS
- An illuminating comment manager at Google: "...start with Cassandra and, if/when it starts to tip-over, go to HBase"

Heavy users

Yahoo:

- HBase with Omid - 100,000 transactions per second.
- Flurry : 2,000+ node Hadoop/HBase cluster, mobile analytics
- http://www.slideshare.net/HBaseCon/hbasecon-2015-hbase-operations-in-a-flurry

Facebook - now running pure Apache HBase (HBaseCon, May 24, 2016)

- online transaction processing workloads - all messaging
- online analytics processing
- internal monitoring system
- Nearby Friends
- search indexing
- streaming data analysis

Use at Google

Congruent with use at Google as BigTable:

- Bigtable: A Distributed Storage System for Structured Data
- HBase is a *faithful* rewrite of BigTable for open source
- Google is a central contributor to HBase with 1.0

“Anything at Google that is persisted is on Big Table somewhere”

- Max Luebbe at HBaseCon2015

Other users

Pinterest - 13 production HBase clusters, used for mobile messaging

Bloomberg - all HBase - “chosen for latency and never losing our data”

Flipboard - uses AWS

Others: Airbnb, Alibaba, Apple, Box, DropBox, Finra, Salesforce, Visa, Xiaomi

Why?

Reliable, low-latency, random-access to HDFS

Why not?

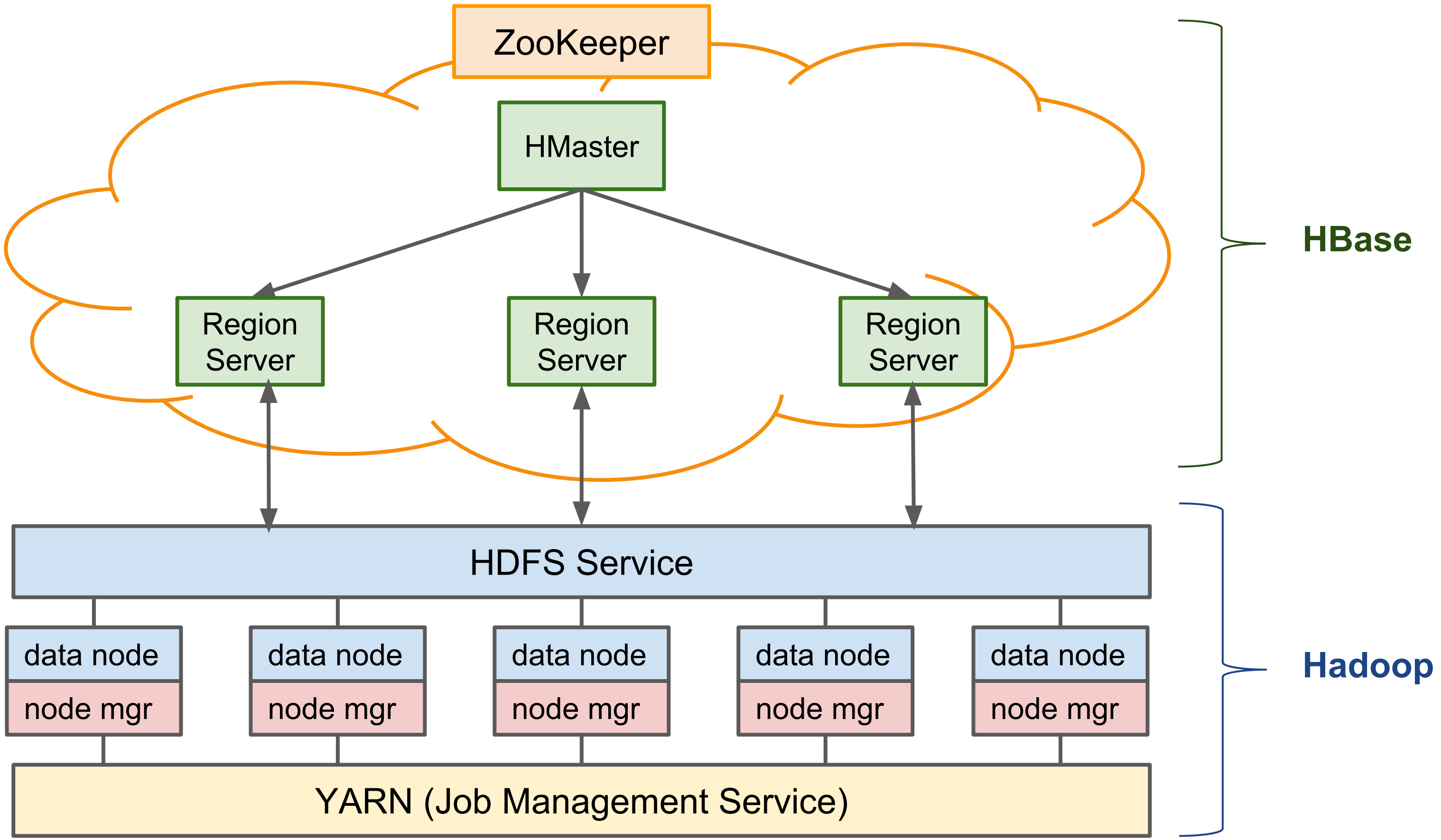
It’s hard to administer

Readings

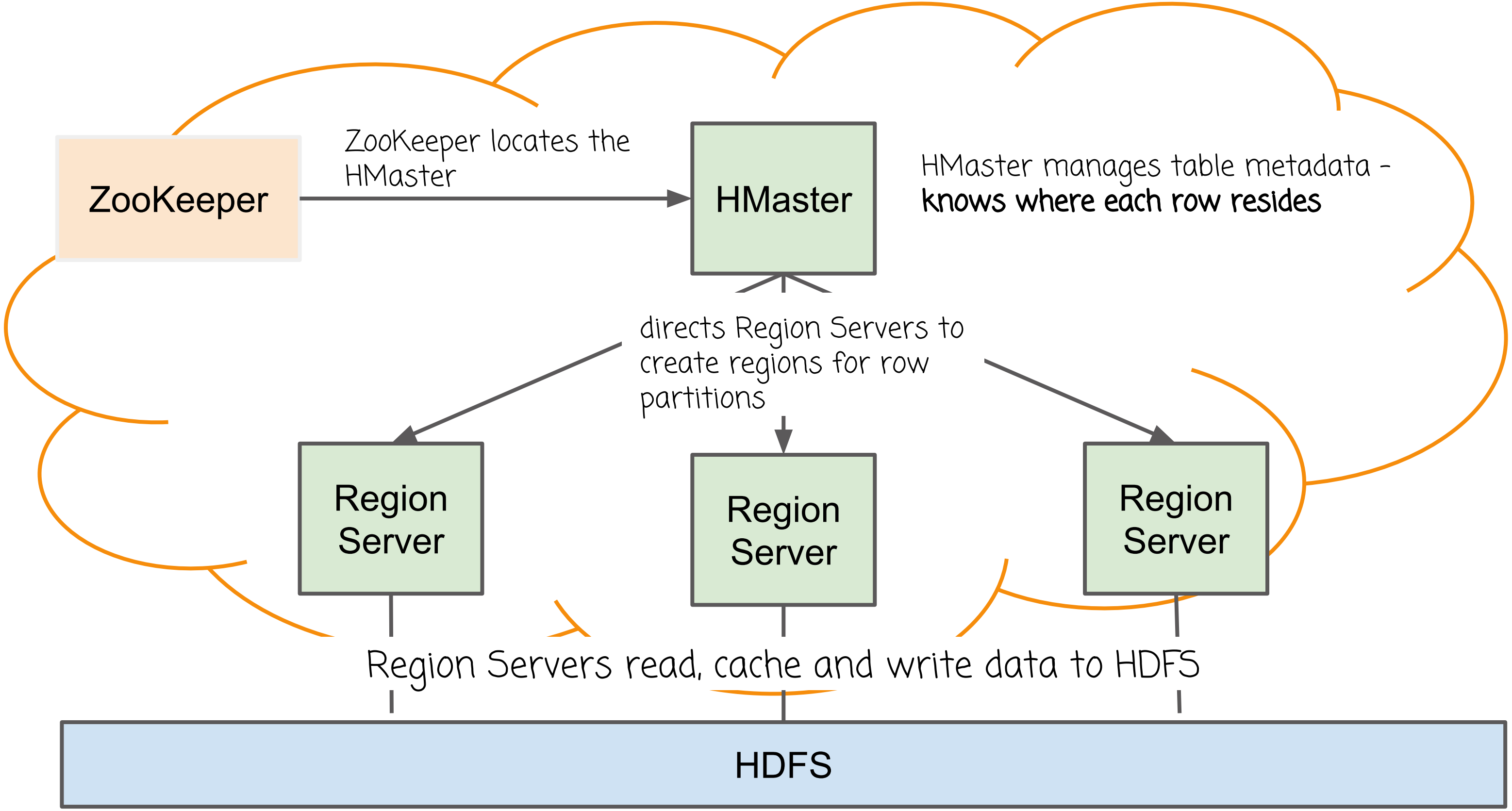
- Free text: **hbase.apache.org/book.html**
- Recommended: HBase: The Definitive Guide, Lars George, O'Reilly 2011, Pages 75 - 150
- Seminal work: **Bigtable: A Distributed Storage System for Structured Data**, OSDI, 2006. (Team from Google)

Topics

- **HBase architecture w/ Zookeeper**
- **HBase characteristics**
- **HBase example**
- **The well-designed key**


ZooKeeper
HMaster
Region Server
Region Server
Region Server
HDFS Service
data node
node mgr
data node
node mgr
data node
node mgr
data node
node mgr
data node
node mgr
YARN (Job Management Service)
HBase
Hadoop

HBase is a distributed database with a master node (HMaster) and slave nodes (Region Servers)



Topics

- Physical architecture
- **HBase characteristics**
- HBase example
- The well-designed key

HBase characteristics

"A Bigtable is a sparse, distributed, persistent, multidimensional sorted map."

- Bigtable: A Distributed Storage System for Structured Data

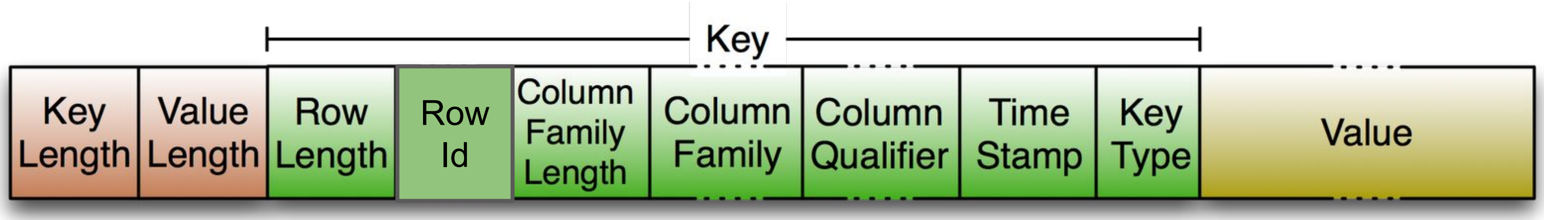
- **sparse -** one very wide table (many columns), most cells are empty
- **distributed** over a cluster
- **persisted** in HDFS
- **multiple aspects of a key -** row id, column id, time
- all data is **sorted lexically** by row id, column id and time
- Each data element is a **key-value** pair, so the whole table is a **map**

Keys

A key in HBase is a compound key

- Row Id
- Column Family
- Column Qualifier
- Time stamp

When you retrieve a key-value from HBase it looks like:



Primary key - Row ID

ROW ID is the primary key

- Row ids are used to partition a table into regions
- Groups of contiguous rows are stored in regions on Region Servers

  rows 1 - 2000 -- region 1

  rows 2001 - 4000 -- region 2

  …

  rows 5,000,001 - 5,002,000 -- region 1? yes, for balance

- HMaster stores lookup data: row id:region server

Key - Column family

COLUMN FAMILY defines columnar storage

- **Groups together a "meaningful" set of columns**
  - Most queries only access one or a few columns (scoped queries)
    - **Example:**
      - examining employee compensation - compensation family
      - examining employee distribution - location family
  - Data types - columns in a family may hold the same kind of data
    - **columns may hold images or text or numerical data**
  - Families are the basis of storage: family-oriented storage in HDFS
- **Column families are compressed together**
- **Column families have the same security**
  - can create ACLs for families

Key - Column Qualifiers

COLUMN QUALIFIER are defined dynamically:

- **A column family and a column qualifier together define a “column”**
  - A column id is usually written as (family:qualifier)
  - Specify a qualifier when adding data, *not* during table creation
- **You can add as many qualifiers as you want (millions)**
- **This is what makes the table “big”**

Key - Time stamp

TIME STAMP:

- **Time stamp is assigned to a new row by default**
  - You can assign your own - with care
- **Time stamps are used to “version” data**
  - Once you create a row of data, it is immutable
  - To change the data, you create a new “version”
- **Versions are used to “age” data out of the system**
  - You can specify how many versions to keep
  - You can specify how long to keep a version (TTL or time to live)
  - Default number of versions to keep = 1
    - **previous versions are deleted**
    - **deletes don’t take place immediately - the data is “marked” for deletion**

Topics

- Physical architecture
- HBase characteristics
- **HBase example**
- The well-designed key

Original use case: the webtable

- **Google needed a way to store data about webpages**
- **Requirements - store all the data for *each* key page and keep information about the back-links from other pages.**

Pages that “in-link” or “back-link” to the key page

The original BigTable: *webtable*

- **Store the data for *each* page and include back-links**
- **Requirements - saving everything needed about a webpage**
  - Requirement 1: Save the contents of the page.
    - **Want to save multiple, timestamped versions**
    - **Used for search**
  - Requirement 2: Store the back-links
    - **Back-links are the URLs of pages that link to the current page**
    - **Used for page-rank**
  - Requirement 3: Save metadata about the page
    - **last view, number of visitors, number of ad clicks, size, revenue generated.**
    - **Used for ad placement**

The keys of *webtable*

- **good row key design: use the reverse URL for the domain**
  - $\underbrace{\text{com.google.sites}}_{\text{reverse URL}}$/site/hadoop30088/home

- **useful column families:**
  - CONTENT - All versions of the page's html, images, etc.
    - **Used for search**
  - BACKLINKS - pages with links into this page
    - **Used for pagerank**
  - META - last view, # of visitors, # of ad clicks, size, revenue
    - **Used for ad placement**

- **time stamps: actual time the pages are fetched.**

Results from ‘scanning’ the table

Note: can put data in the qualifier

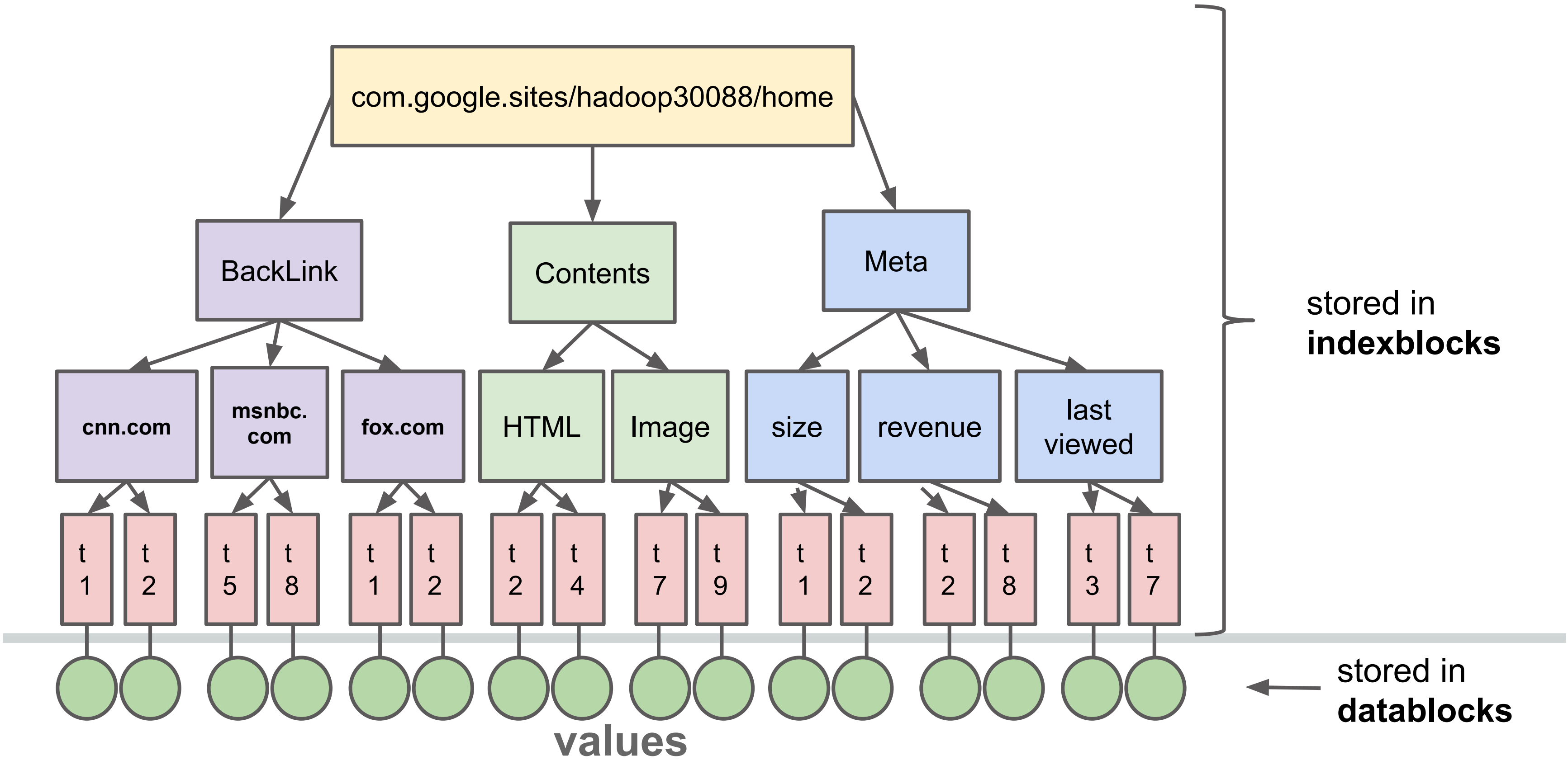
rowid	column family	column qualifier	time stamp	
com.google.sites/hadoop30088/home	backlinks	com.cnn.www/...	t2	“important hadoop site”
com.google.sites/hadoop30088/home	backlinks	com.cbs.www/...	t1	“wow. great website.”
com.google.sites/hadoop30088/home	backlinks	com.fox.www/...	t1	“left-wing propaganda.”
com.google.sites/hadoop30088/home	content	html	t1	<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN""http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"><html xmlns="http://www.w3.org/1999/xhtml" itemscope="" itemtype="http://schema.org/WebPage">..
com.google.sites/hadoop30088/home	content	html	t2	<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN""http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"><html xmlns="http://www.w3.org/1999/xhtml" itemscope="" itemtype="http://schema.org/WebPage">..
com.google.sites/hadoop30088/home	meta	size	t1	1820
com.google.sites/hadoop30088/home	meta	revenue	t2	82301
com.google.sites/hadoop30088/home	meta	last view	t2	t2

***Actual file is not easy to read - each line is a byte array...**

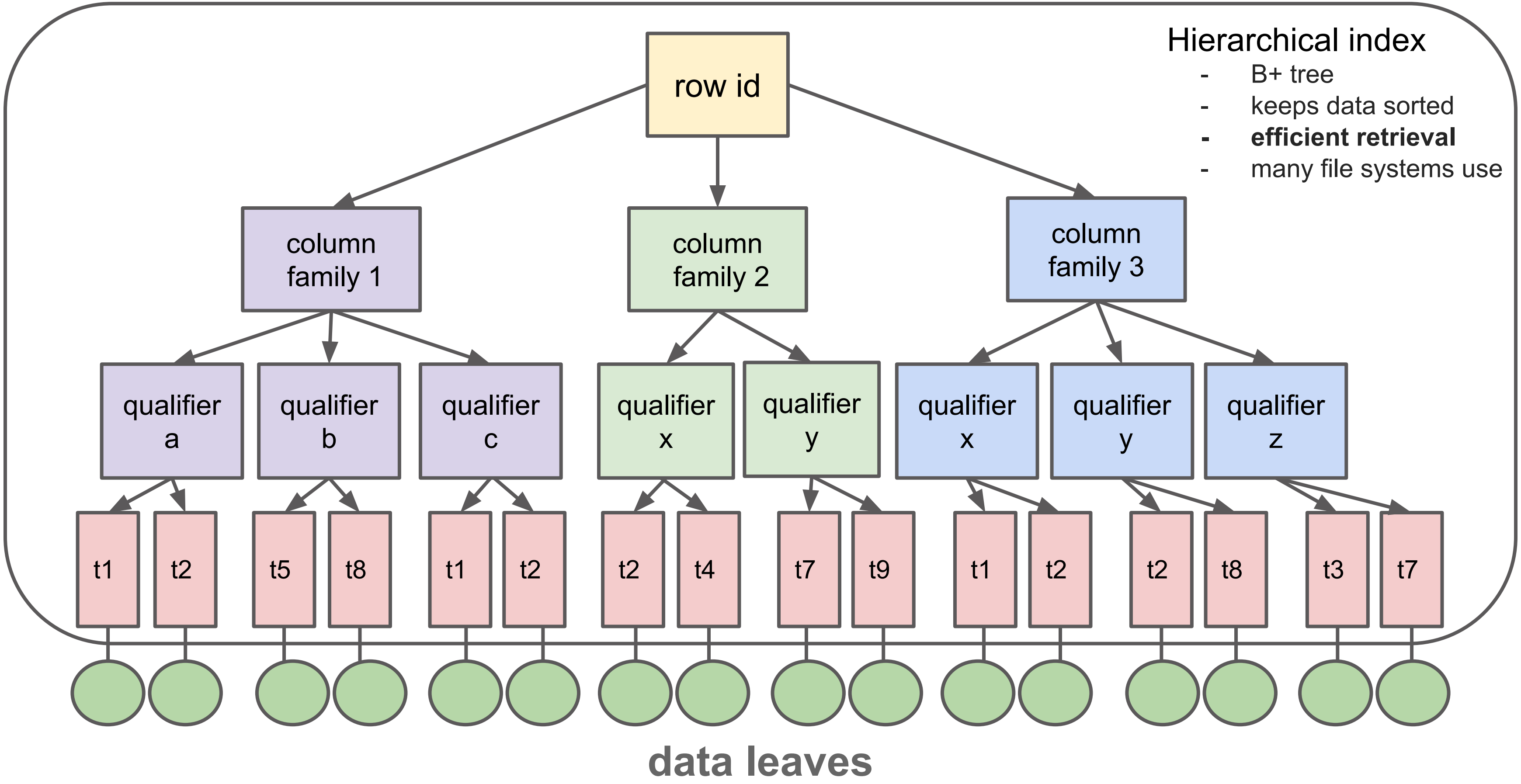
A view of a row's logical record

rowid	column family	column qualifier	time	value
com.google.sites/ hadoop30088/home	backlinks	com.cnn.www/...	t1	"important hadoop site"
		com.cmsnbc.www/...	t2	"right-wing propaganda."
		com.fox.www/...	t1	"left-wing propaganda."
	content	html	t1	<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN ""http://www.w3.org/TR/xhtml1/DTD/xhtml1-
			t2	<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN ""http://www.w3.org/TR/xhtml1/DTD/xhtml1-
	meta	size	t1	1820
		revenue	t2	82301
		last view	t2	t2

The logical record, rotated.



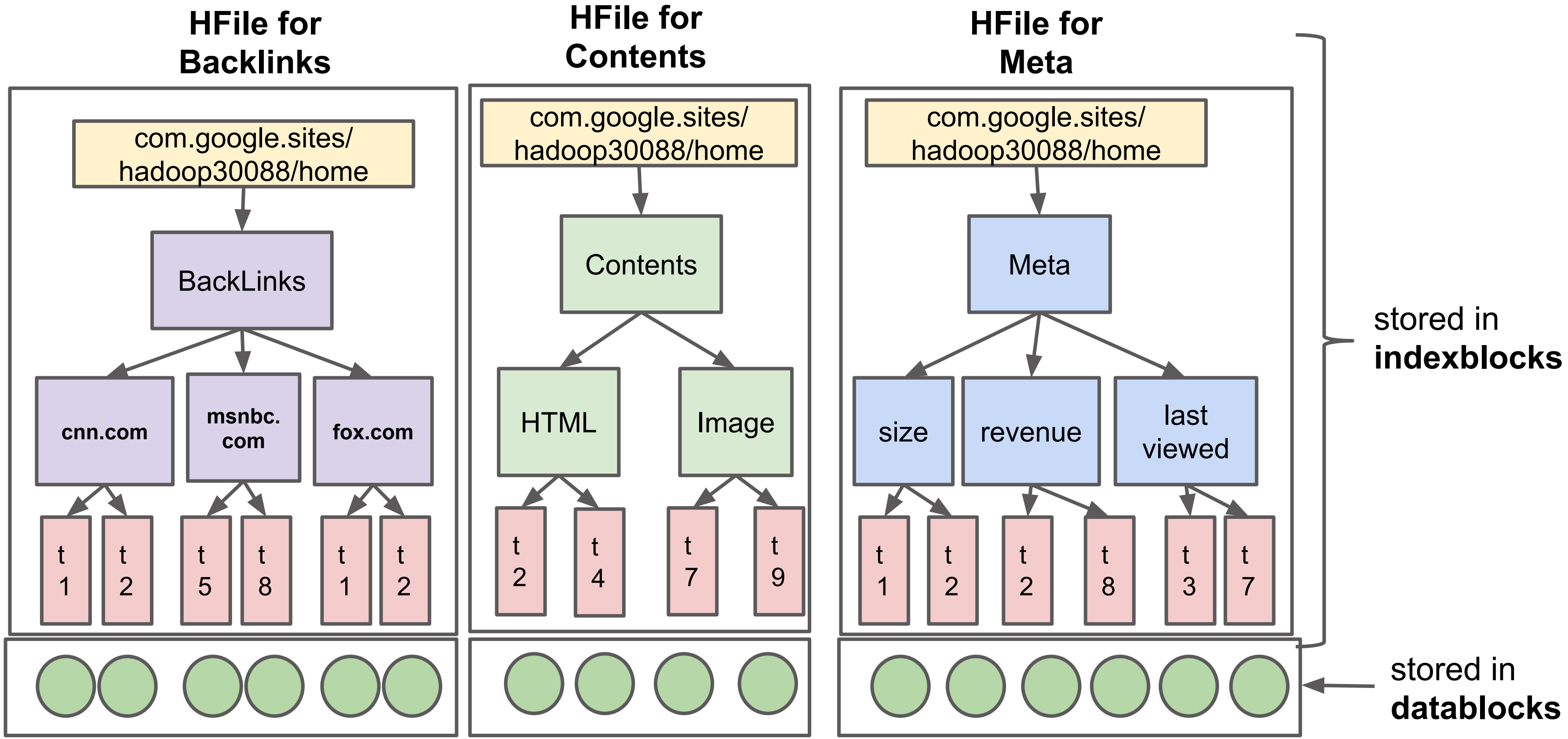
Hierarchical key structure



Hierarchical index

- B+ tree
- keeps data sorted
- **efficient retrieval**
- many file systems use

The logical record, stored in a file.



Data-locality and the well-designed row id

- **Core concept: Ranges of row keys should form *useful groups***

- ***Example: the reverse URL for domains***
  - *Sorting on the reverseURL causes pages in a domain to group together.*

- **In HBase, when you retrieve a row, it retrieves and caches the rows around it.**
  - If sorted row keys naturally cluster, then HBase caching will retrieve the cluster.
  - Therefore: design a key by clustering (e.g. with nearest neighbor or LSH)
  - Create a row id that reflects cluster membership.

On the other hand… there are Hotspots

- **Hotspot: when access patterns result in a data server being swamped**
  - Solution: alter the row key so data is spread over servers
    - **Salting: adding a new, random prefix**
    - **Hashing: deterministically adding a prefix to certain rows**
    - **Reversing the key (e.g. reversed URL by Google)**
  - You will need to balance this against the usefulness of data locality.
    - **That is, as soon as you spread the hot spot out, you lose data locality**
    - ***sigh***

Keys: important points

- **Row key design is the most important aspect of table design.**
  - Indexing by the master is based on the Row key.
  - Tables are sorted based on the Row key.
  - Region Servers hold contiguous sets of rows
- **Column families**
  - store columns with the same access patterns or data types.
  - use columns to control compression and security.
  - column families are stored together, as files.
- **Time stamps are used for versioning.**
  - Data is not changed, new data is created and time stamped.
  - New versions are served first
  - Old data can be removed by setting time-to-live (TTL) or Max Versions

References for key design

- **smart key design promotes data locality**
  - *see HBase: The Definitive Guide, Chap 9.*
- **but.. smart key design also avoids “hotspotting” with salting**
  - see http://hbase.apache.org/book.html#rowkey.design
- **… deciding when to salt is why you are paid well.**

Loading HBase

Bulk loads via MapReduce or Spark

The problem of scampering employees

2015 was a brutal year for turnover in oil companies

We want to keep track of employees who quit

Employees who quit after less than two years of employment

- **we say they “scampered”**
- **we will create a table called scamper**
- **answer the board’s questions:**
  - how many were “critical” employees?
  - did they have a hideous manager?
  - were they being relocated?

How do I load data files into HBase? (1)

- **load csv data into HDFS**
  - example: hadoop fs -put quickQuits emp_data/scampered

info in "quickQuits": name, salary, appraisal, duration, scampered, site, relocated, manager

- **create the table using the hbase shell**

```
hbase> create 'scamper', ‘emp’, ‘loc’, ‘dept’, {SPLITS => ['g', 'm', 'r', 'w']}
```

table name

columns

specify split points

- splits file into parts
- example: splits on first letter of name
- defines 5 regions: (a-f, g-l, m-q, r-w, w-z)

How do I load data files into HBase? (2)

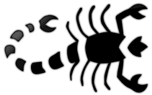
- **generate the hfiles (this command runs a Map Reduce job)**

```
$ hbase org.apache.hadoop.hbase.mapreduce.ImportTsv \
  -Dimporttsv.separator=, \
  -Dimporttsv.bulk.output=HFileScamper \
  -Dimporttsv.columns=HBASE_ROW_KEY,emp:salary, emp:appraisal,\
  emp:duration, emp:scampered, loc:site, loc:relocated, \
  dept:manager \
  scamper emp_data/scampered
```

- **would this be faster in Spark?**

How do I load data files into HBase? (3)

- Make sure the hfiles you created are assessible to HBase



```
sudo -u hdfs hadoop fs -chown -R hbase:hbase/user/emp_data/scampered
```

- Finally, load the hfiles into hbase directly

```
$ hbase org.apache.hadoop.hbase.mapreduce.LoadIncrementalHFiles \
hdfs://HFileScamper scamper
```

What if I need to cleanup the data first?

1. **Create and run a MapReduce or Spark job to switch-up the data.**
   - Output the results with HFileOutputFormat
   - The output will be an HFile (e.g. HFileScamper)

2. **Run completebulkload**

```
$ hadoop jar hbase-VERSION.jar completebulkload /user/emma/HFileScamper scamper
```

Note: ***this will create the create 'scamper' if you haven't already done so***

Examining the new table

$ hbase shell

```
hbase> list
TABLE
scamper
```

```
hbase> describe ‘scamper’
DESCRIPTION
 'scamper', {NAME => 'emp', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER =>
'ROW', REPLICATION_SCOPE => '0', VERSIONS => '1', COMPRESSION => 'NONE',
MIN_VERSIONS => '0’, TTL => 'FOREVER', KEEP_DELETED_CELLS => 'false', BLOCKSIZE
=> '65536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}
```

```
hbase> scan ‘scamper’
... whoops, I just printed all the rows in scamper to the console...
```

HBase shell

Basic commands

Using the hbase shell

- **Ruby shell used for simple operations/prototyping**
- **For complex operations use Java API**
  - batched ‘puts’ and ‘gets’
  - cell-level security
  - complex filtering on results
- **As time goes on, the shell has become more “powerful”**
  - can now filter in the shell
  - may have problems keeping up with the Java API in HBase 10

Creating an empty ‘scamper’ table

```
$ hbase shell
```

```
hbase> create ‘scamper’, 'emp', ‘loc’, ‘dept’
```

table name

column families
(usually < 100 families)

forgot something?

```
hbase> help
hbase> table_help
```

Altering the table

```
hbase> disable 'scamper'
hbase> alter 'scamper', ‘team’
hbase> alter 'scamper', {NAME=> ‘emp’, VERSIONS=‘5’}
hbase> enable 'scamper'
hbase> describe ‘scamper’
```

add column family ‘team’

change the number of versions kept for ‘emp’

Describing ‘scamper’

```
hbase> describe ‘scamper’
DESCRIPTION
 'scamper', {NAME => 'emp', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER =>
'ROW', REPLICATION_SCOPE => '1', VERSIONS => '3', COMPRESSION => 'NONE',
MIN_VERSIONS => '0’, TTL => 'FOREVER', KEEP_DELETED_CELLS => 'false', BLOCKSIZE
=> '65536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}
```

Anything described you can set during create

- you are usually working on column family characteristics
- example:
    - create ‘scamper’, {NAME=>’emp’, COMPRESSION=>’LZO’, VERSIONS=>’5’, IN_MEMORY=>’true’}

Add a row to ‘scamper’ table

```
hbase > put 'scamper',  'Kulmani891', 'emp:salary', '17200'
```

table

row id

family : qualifier
column id

value

Result: a new {key,value} in ‘scamper’ table

key value

{ **Kumani891, emp:salary,** <*timestamp**>, 17200 }

Since we didn’t give a timestamp, the system automatically created one.

Adding your own timestamp

```
hbase > import java.text.SimpleDateFormat
hbase > import java.text.ParsePosition

hbase > SimpleDateFormat.new("yy/MM/dd HH:mm:ss").parse("08/08/16 20:56:29",
ParsePosition.new(0)).getTime()
1218920189000

hbase > put ‘scamper’, ‘ellen1232’, ‘emp:scampered’, ‘true’, 1218920189000
```

Usually, you would let HBase assign the timestamp

Viewing and changing data

```
hbase > scan ‘scamper’
 ROW                 COLUMN + CELL
row1 column = emp:scampered, timestamp = 1418051555, value = false
row1 column = emp:title, timestamp = 1418051555,value = jedi
row1 column = emp:bonus, timestamp = 1418051555,value = 0
1 row(s) in 0.0100 seconds

hbase > put ‘scamper’, ‘Kulmani891’, ‘emp:scampered’, ‘true’
0 row(s) in 0.0400 seconds

hbase > scan ‘scamper’
  ROW                 COLUMN + CELL
row1 column = emp:scampered, timestamp = 1427650516, value = true
row1 column = emp:title, timestamp = 1418051555,value = jedi
row1 column = emp:bonus, timestamp = 1418051555,value = 0
1 row(s) in 0.0100 seconds
```

Scanning the table

- Scan the table - this returns all the data (!) and shows how it is organized

```
hbase> scan ‘scamper’
```

- Scan emp and dept families and limit to 10 results

```
hbase> scan ‘scamper’, {COLUMNS=>[‘emp’,’dept’], LIMIT=>10}
```

- Scan within a time period

```
hbase> scan ‘scamper’, {TIMERANGE => [1303668804, 1303668904]}
```

Scanning in rows and columns

To scan a **range of rows**, specify a startrow and an endrow -- ranges are not inclusive

```
#Example:  returns info about ‘corey’ but does not return info about ‘corfu’  - like (x,y] interval
hbase> scan 'scamper, {STARTROW => ‘corey’, ENDROW=> ‘corfu’}
```

Another approach is to use a prefix filter - **filter for a row id prefix**

```
# Example:  this will get all the scamperers with name starting with ‘c’
hbase> scan 'scamper', {FILTER => "PrefixFilter('c')”}
```

Can **filter for column ids** using SingleColumnValueFilter

```
#Example:  find everyone working at Whiting who scampered
hbase > scan ‘scamper’,
        {FILTER => “(SingleColumnValueFilter( 'loc' , 'site',  =>, 'binary:Whiting') )”}
```

Filter parameters

- **filters compare and match**
    - Compare operators:
        - **<, <=, =, !=, =>, >**
    - Matching:
        - **binary:**val **-** exactly matches val
        - **regexstring:**pattern - matches the pattern
        - **substring:**str - contains str

Commonly used filters

- **RowFilter**
  - `Example:` `RowFilter(=, ‘regexstring:Kulmani*’)`
  - retrieves all key-values with row ids containing ‘Kulmani’
- **Family filter**
  - Example: `FamilyFilter(=, ‘binary:emp’)`
  - returns all key-values in the ‘emp’ family
- **Value filter**
  - Example: `ValueFilter(=>, ‘binary:200000’)`
  - matches all key-values with a value of 200000

Combining filters

You can combine filters (AND, OR)

- AND - must pass both the filters
- OR - pass at least one of the filters

```
hbase> scan ‘scamper’, COLUMNS => ['emp'], {FILTER =>
     “(RowFilter (=,binary:‘Kulmani912’)
     AND
     (QualifierFilter (>=, ‘binary:salary’)))
     AND
     (TimestampsFilter (36501325, 36501697))”
}
```

More filters

- **SingleColumnValueExcludeFilter**
  - same as SingleColumnValueFilter except it excludes the given column
- **PrefixFilter**
  - Based on prefix of row keys
- **FirstKeyOnlyFilter**
  - Returns the key of the first key-value pair
- **Many more…**
- **You can write your own**

List of filters:
http://www.cloudera.com/content/cloudera/en/documentation/core/latest/topics/admin_hbase_filtering.html

Original documentation: see Filter Language.docx in

https://issues.apache.org/jira/browse/HBASE-4361

Getting a single row or parts of a row

Multiple rows:

```
hbase > scan ‘scamper’...
```

For just one row at a time:

specify an exact key

```
hbase> get 'scamper', 'Kulmani856'
hbase> get 'scamper', 'Kulmani856', {COLUMN => 'emp’}
hbase> get 'scamper', 'Kulmani856', {COLUMN => ['emp', 'dept']}
hbase> get 'scamper', 'Kulmani856', {COLUMN => 'emp', TIMESTAMP => 18293471}
hbase> get 'scamper', 'Kulmani856', {COLUMN => 'emp', VERSIONS => 4}
```

Keep what you have done and run as a script

```
>> vi ~/.irbrc
```

```
require 'irb/ext/save-history'
IRB.conf[:SAVE_HISTORY] = 100
IRB.conf[:HISTORY_FILE] = "#{ENV['HOME']}/.irb_history"
Kernel.at_exit do
    IRB.conf[:AT_EXIT].each do |i|
        i.call
    end
end
```

Create the script from the log of shell commands in .irb_history

Run the script

```
>>  ${HBASE_HOME}/bin/hbase shell PATH_TO_SCRIPT
```

Essential knowledge

- create
- alter
- put
- describe
- scan, scan with filters
- get

----

loading files with ImportTSV

loading files created by MapReduce with completebulkload

HBase tech

Cool aspects of HBase

Why is HBase fast?

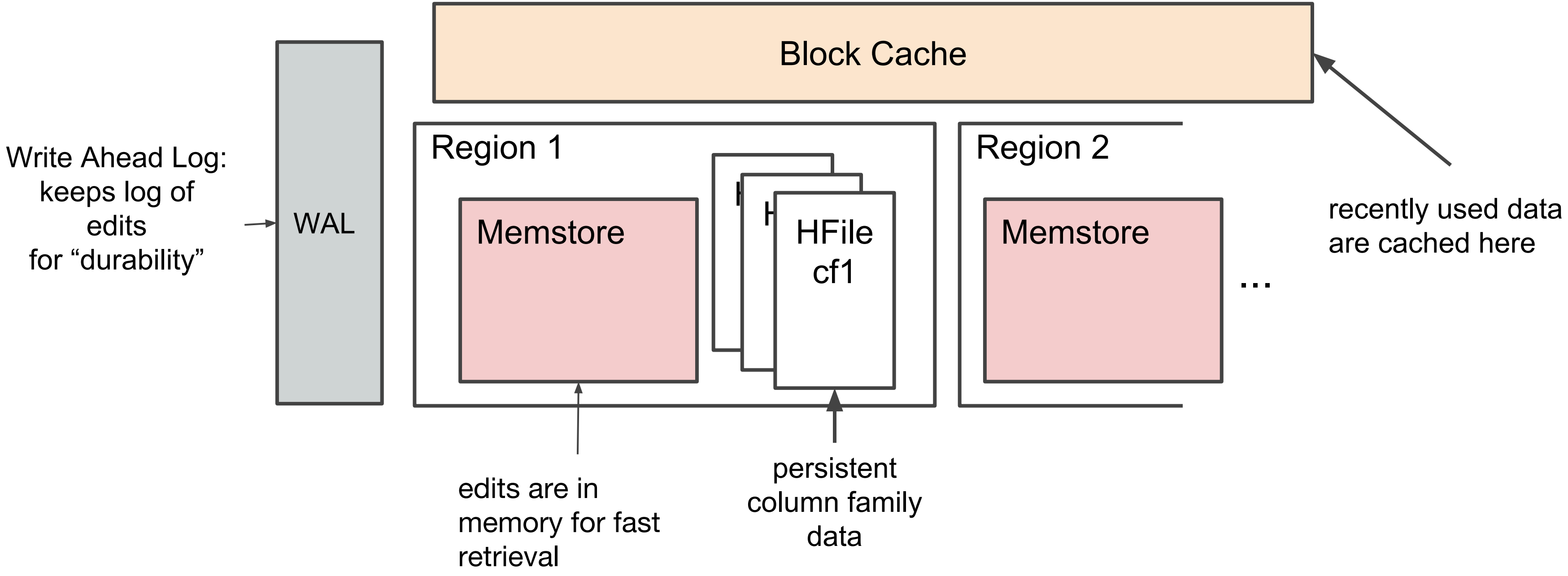
Fast lookups

- **Bloom filters**
- **Caching**

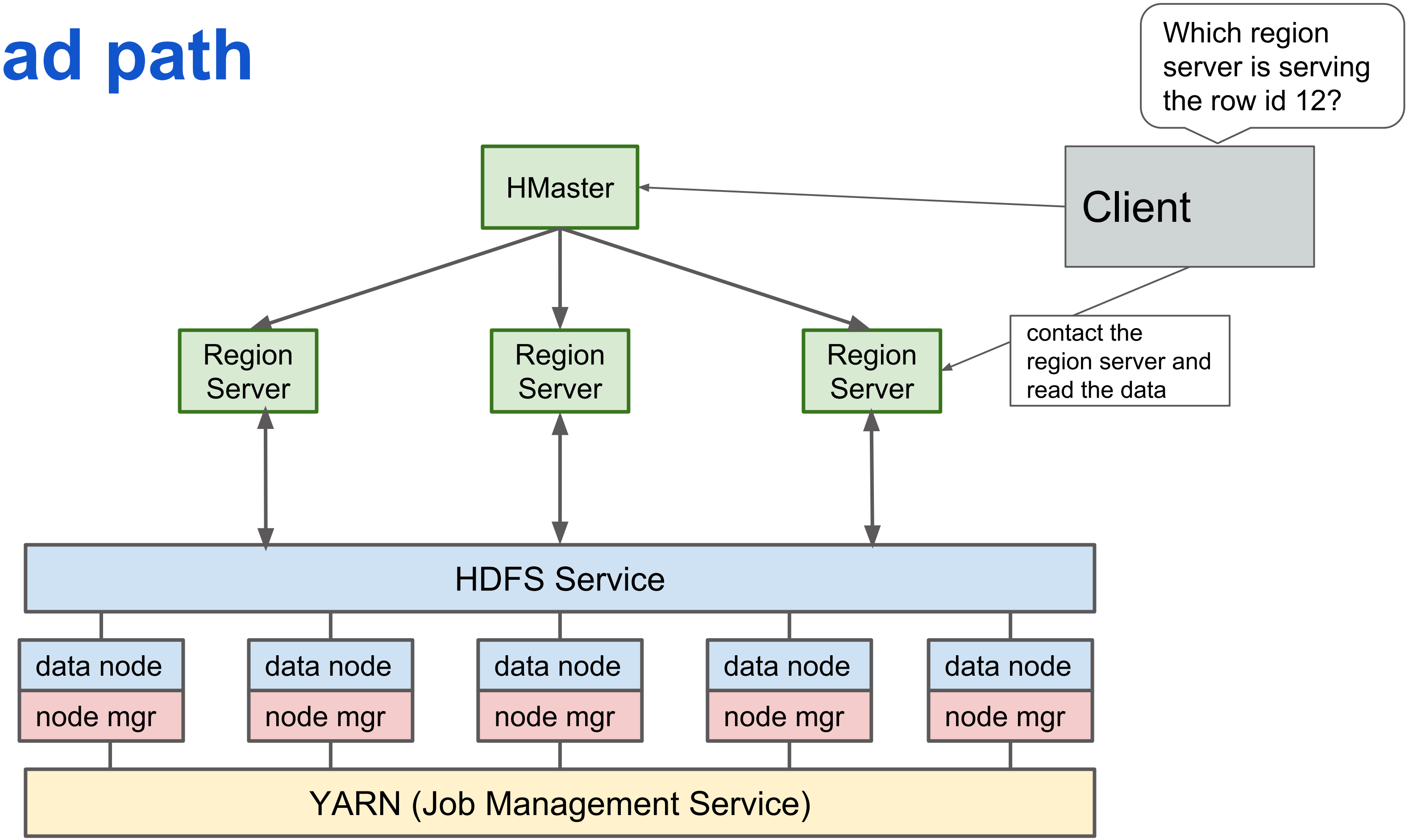
Fast writes

- **A hierarchical approach to writing called LSM (log structured merge)**
- **Uses different kinds of storage (memory, files, remote files)**
- **Writes over different time-frames**
  - Memory writes are immediate
  - Files writes occur fairly frequently (minor compaction events)
  - Remote files occur only on “major” compaction events

Components of the Region Server

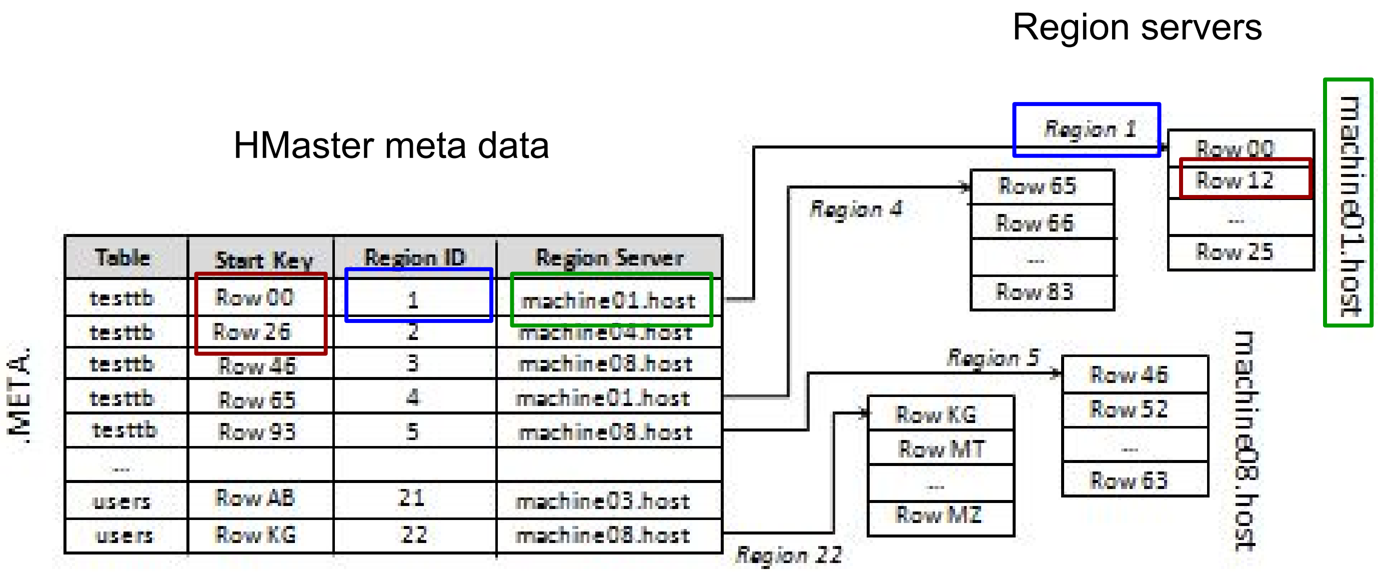


read path



Which region server is serving the row id 12?
Client
HMaster
Region Server
Region Server
Region Server
contact the region server and read the data
HDFS Service
data node
node mgr
data node
node mgr
data node
node mgr
data node
node mgr
data node
node mgr
YARN (Job Management Service)

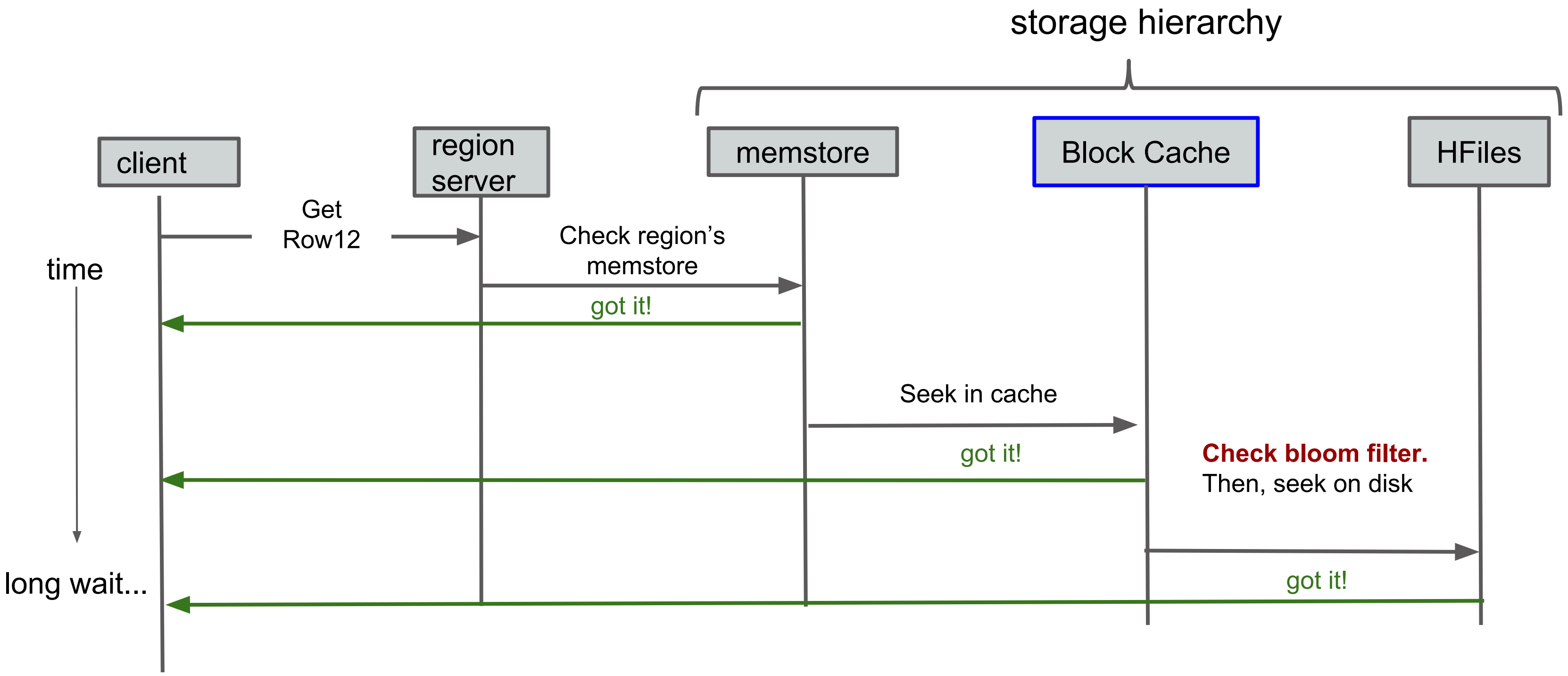
Region server lookup



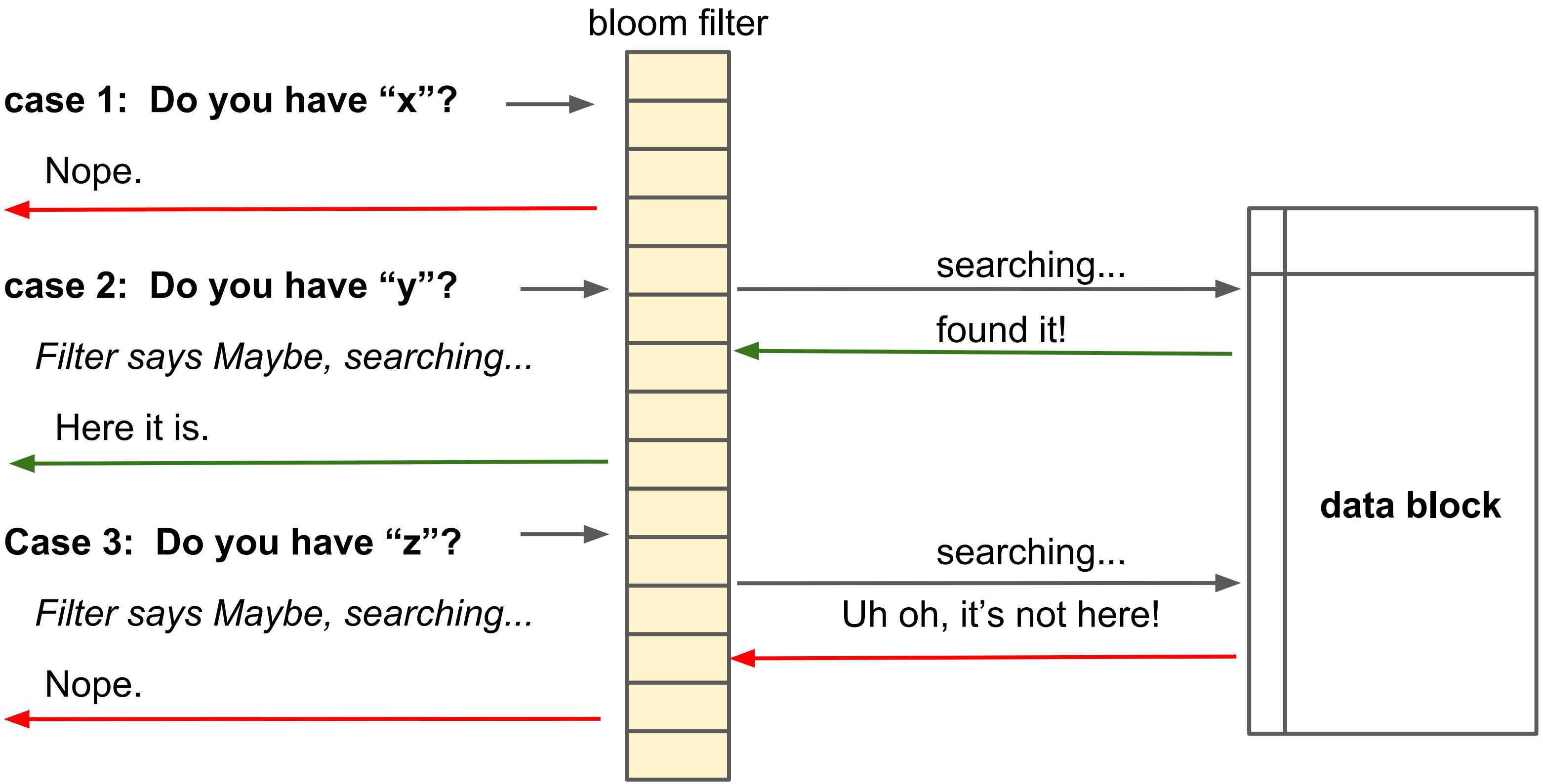
Question: What if the row isn't on the region server?

- HMaster initiates fetch of row data from HDFS.
- Want more? Go here...

Hierarchical read on the Region Server



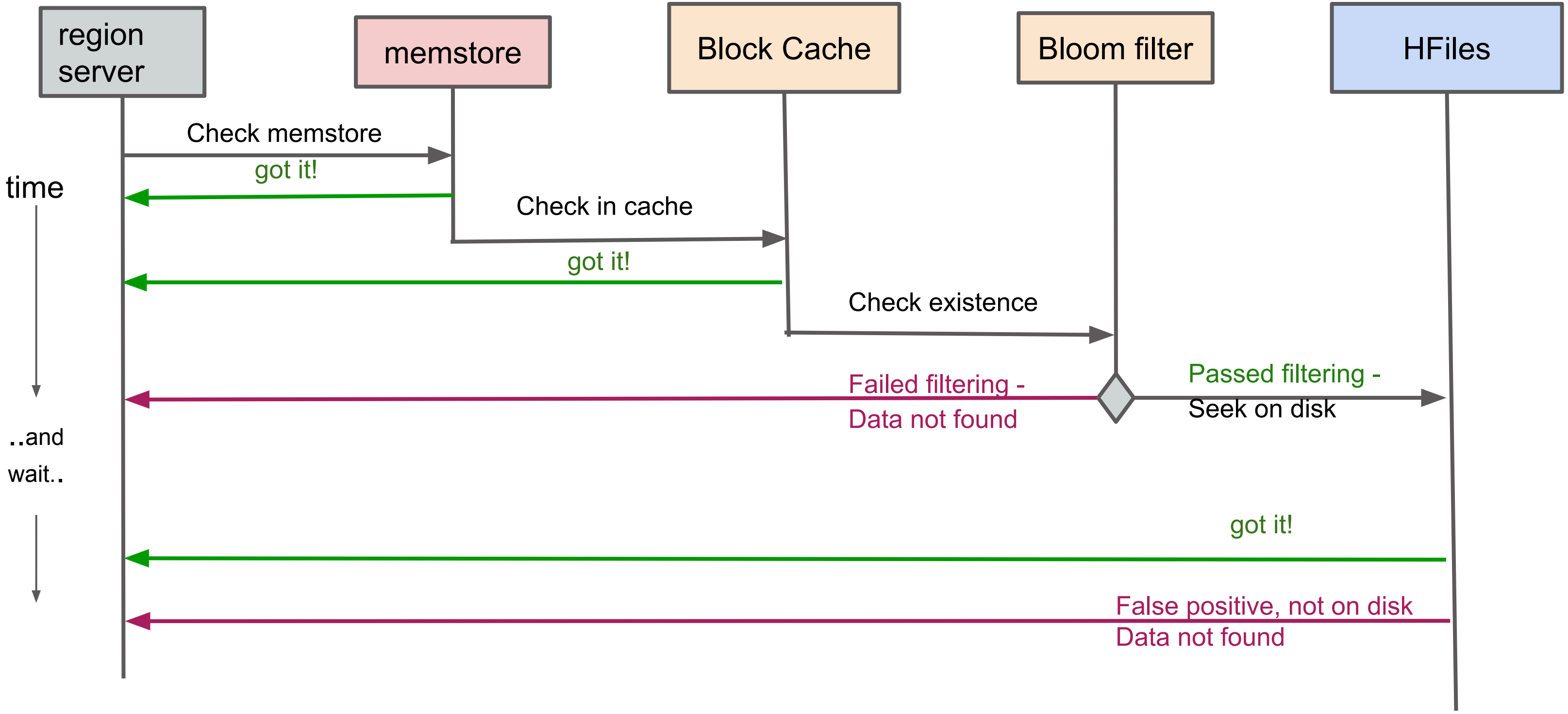
Bloom filters - intro



Bloom filters - fewer unnecessary “seeks”

- **Checks if data is on the disk before “seeking”**
- **Replies**
  - “No” - definitely not there
  - “Maybe” - probability of being on disk is *high*
    - **How high? Depends on the filter**
    - **Row filter - built only for Row ids**
    - **RowCol filter - built for both Row and Column ids**
      - - more reliable, more finely grained
      - - has to be rebuilt more often as data changes

Data retrieval with a bloom filter



Bloom filters - how they work

- **Easily built for B+ trees (like our key-tree!)**
- **Create the Bloom hashtable:**
    - Use the key’s components - add a hash for level of the tree
    - Combine to the hashes into a single hash
    - Add to the Bloom hashtable
- **To fetch, first query the filter:**
    1. hash the key that is sought -> create queryHash
    2. lookup queryHash in Bloom hashtable
    3. Matches? The data *might* really be in storage.

Setting a bloom filter

```
hbase> create 'scamper',{NAME => 'emp', BLOOMFILTER =>
'ROWCOL'}, ‘dept’
```

BloomFilter options:

- None
- `ROW`: checks for row ids (default)
- `ROWCOL`: checks for row ids and columns

More on bloom filters

Delve into how they work: Why bloom filters work the way they do.

See them at work on a B+ tree: Early use of bloom filters for a B+ tree

What is the Block Cache?

Caches recently used rows - and their neighbors!

Caches *all* row indices on the server and the bloom filters

All cache types use LRU (least recently used) eviction policy.

- **LRUCache (default) - all on-heap**
- **BucketCache: all off-heap**
- **CombinedBlockCache:** Uses both on and off-heap - nice!

Want more?
http://hortonworks.com/blog/blockcache-showdown-hbase/

Configuration: BlockCache=true

BlockCache=false

Why use the Block Cache?

- **Caches rows of data**
  - If you read one row, HBase will cache the surrounding rows, too.
  - If you stop using the data, the cache will eventually evict the data

- **Important: index and bloom blocks are always in the cache**
  - Indexes and bloom filters are the heart of HBase - it needs them
  - Hence, IndexBlock and Bloom blocks are always cached.

Summary: Fast lookups

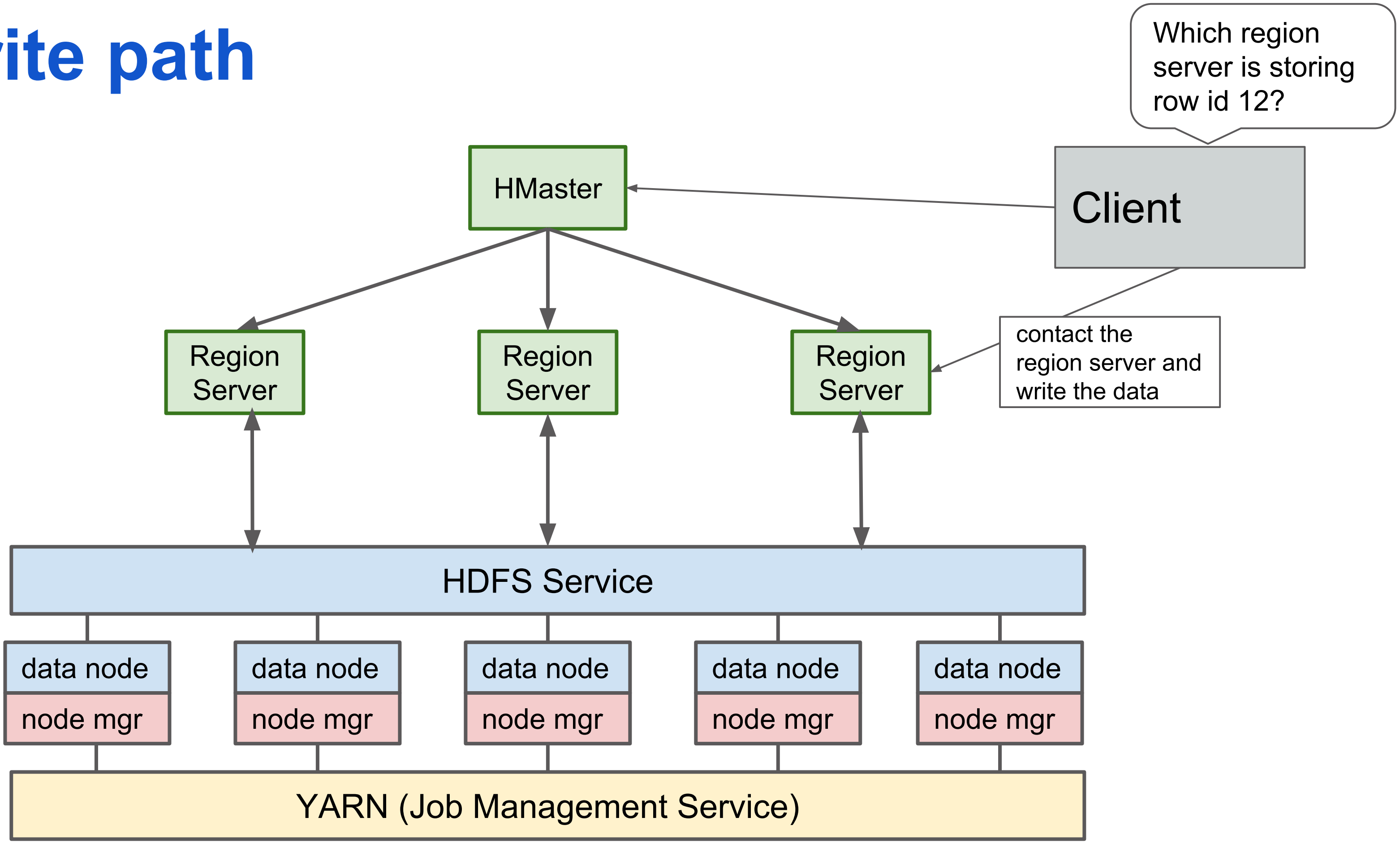
Fast hierarchical lookup for data:

- Zookeeper finds the HMaster
- HMaster finds the desired row in region servers.
- Client contacts the region server and initiates search:
  1. Memstore (insanely fast)
  2. Block Cache (usually really fast)
  3. Check the Bloom filter
  4. HFiles (slow.. disk access)

Why computer scientists love caching:

Main memory reference: 10 ns
Disk seek: 10,000,000 ns

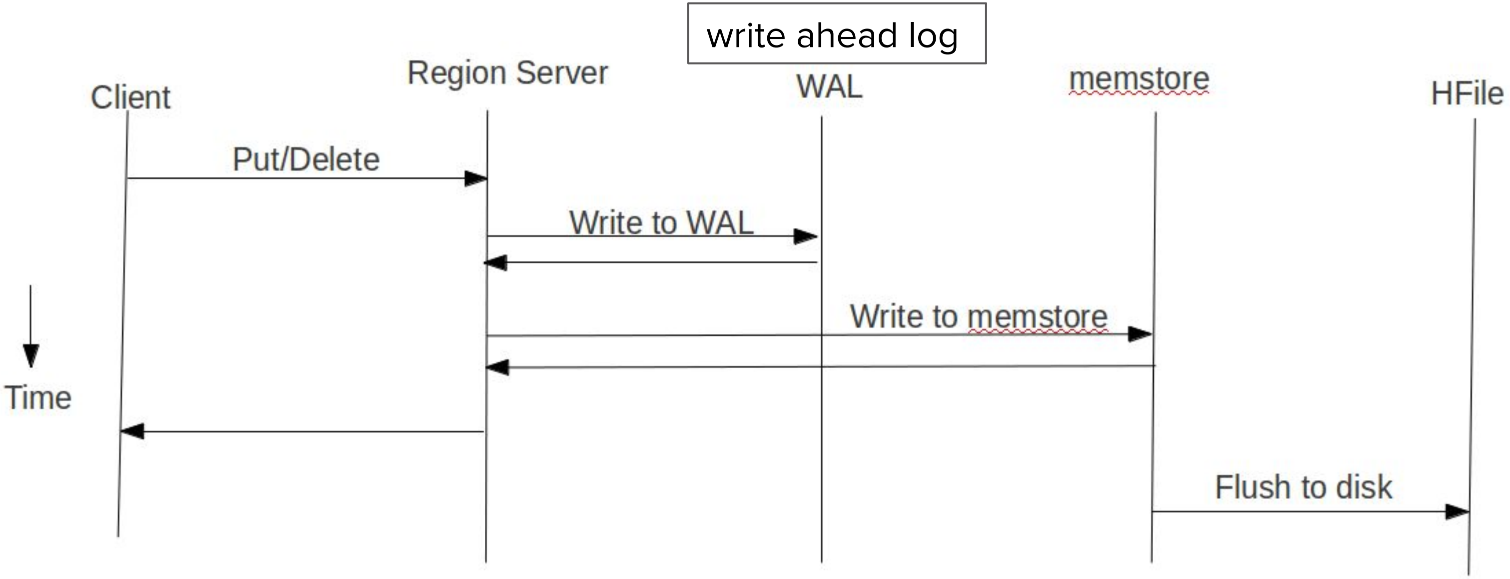
write path


Which region server is storing row id 12?
HMaster
Client
Region Server
Region Server
Region Server
contact the region server and write the data
HDFS Service
data node
node mgr
data node
node mgr
data node
node mgr
data node
node mgr
data node
node mgr
YARN (Job Management Service)

Fast writes: The LSM Tree in HBase

- **Log structured merge trees (LSM)**
  - used in hierarchical storage
    - periodic writes to the next (lower and slower) level of storage
    - each write usually involves a merge
  - HBase hierarchy:
    - Memory buffers (Memstore)
    - Local disk (HFile on the Region Server)
    - Disk on another machine (HFile in HDFS)
- **Another “borrow” from file system design**

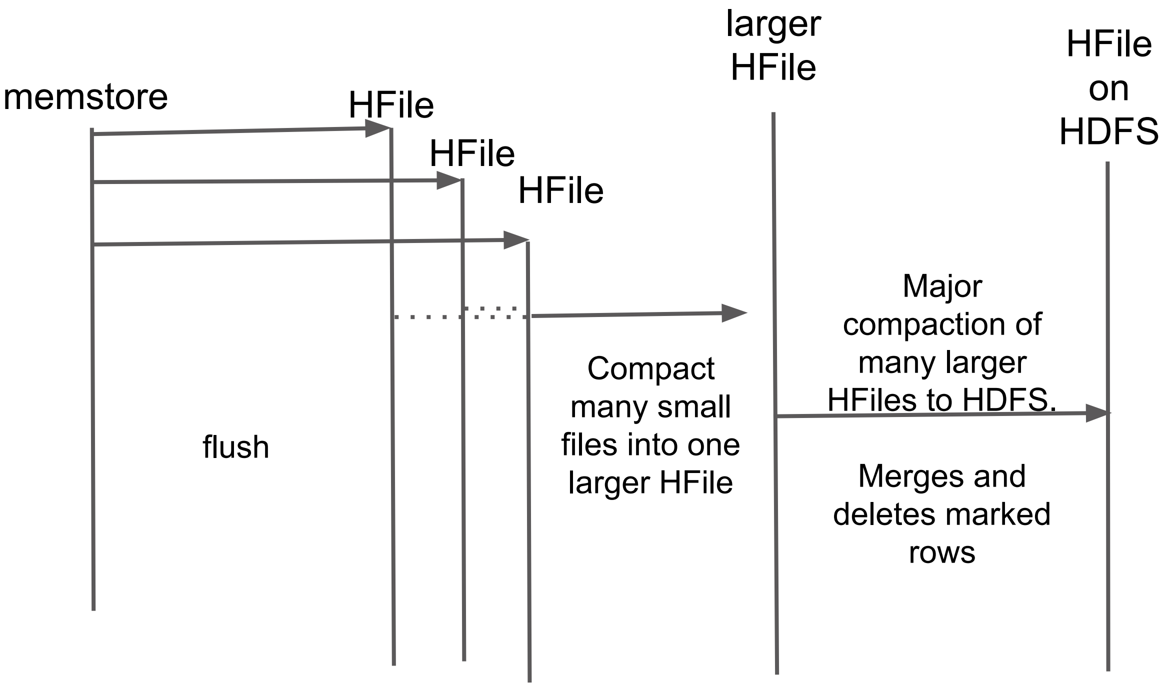
The write path



HBase Write Path

http://blog.cloudera.com/blog/2012/06/hbase-write-path/

LSM - Compaction events



Stages in HBase data reorganization

- Data events are recorded to a WAL on HDFS, for durability
  - After fails, edits in WAL are replayed during recovery
  - WAL appends are **immediate**, in critical write-path
- Data is collected in "MemStore", until a "flush" writes to HFiles
  - Flush is automatic, based on configuration (size, or staleness interval)
  - Flush clears WAL entries corresponding to MemStore entries
  - Flush is **deferred**, not in critical write-path
- HFiles are merge-sorted during "Compaction"
  - Small files compacted into larger files (minor and major compaction)
  - Old records discarded (major compaction only)
  - Very expensive in terms of disk and network time, **infrequent**

Key points

- **HBase is an engine for finding and changing data in HDFS files**
- **HBase's primary strengths are:**
    - Caching of data using the Memstore and the Block Cache
    - Deep indexing of data using a B+ tree
    - Fast lookup of indexed data using Bloom filters
    - Column-oriented storage
    - Hierarchical storage and LSM based writing

References

HBase 1.0 documentation from apache:

http://hbase.apache.org/book.html

HBase and BigTable:

http://jimbojw.com/wiki/index.php?title=Understanding_Hbase_and_BigTable

Up-and-coming: Spark on HBase

- Full access to HBase in a map or reduce stage
- Ability to do a bulk load
- Ability to do bulk operations like get, put, delete
- Ability to be a data source to SQL engines

https://blog.cloudera.com/blog/2015/08/apache-spark-comes-to-apache-hbase-with-hbase-spark-module/

Extra slides

Topics we don’t have time for

Flume

Ingesting data from many sources

Flume

- **Flume is typically used to ingest log files from real-time systems such as Web servers, firewalls and mailservers into HDFS**
- **Currently in use in many large organizations, ingesting millions of events per day**
  - At least one organization is using Flume to ingest over 200 million events per day
- **Flume is typically installed and configured by a system administrator**
  - Check the Flume documentation if you intend to install it yourself

Loading real-time data using Flume

- **Flume is a distributed, reliable, available service for efficiently moving large amounts of data as it is produced**
  - Ideally suited to gathering logs from multiple systems and inserting them into HDFS as they are generated
- **Flume is Open Source**
  - Initially developed by Cloudera
- **Flume's design goals:**
  - Reliability
  - Scalability
  - Extensibility

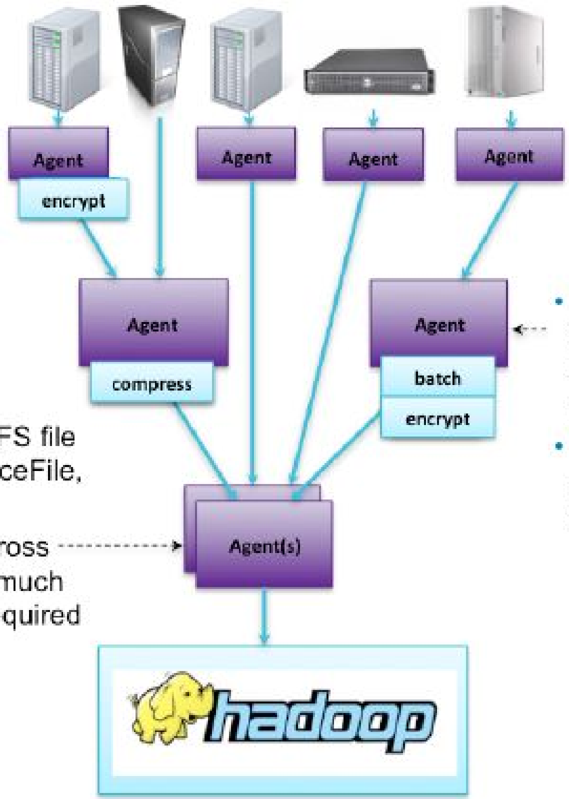
Flume Reliability

- **Channels provide Flume's reliability**
- **Memory Channel**
  - Data will be lost if power is lost
- **File Channel**
  - Data stored on disk
    - Guarantees durability of data in face of a power loss
- **Data transfer between Agents and Channels is transactional**
  - A failed data transfer to a downstream agent rolls back and retries
- **Can configure multiple Agents with the same task**
  - e.g., two Agents doing the job of one "collector" – if one agent fails then upstream agents would fail over

Flume Scalable and Extensible

- **Scalability**
  - The ability to increase system performance linearly by adding more resources to the system
  - Flume scales horizontally
    - As load increases, more machines can be added to the configuration
- **Extensibility**
  - The ability to add new functionality to a system
- **Flume can be extended by adding Sources and Sinks to existing storage layers or data platforms**
  - General Sources include data from files, syslog, and standard output from a process
  - General Sinks include files on the local filesystem or HDFS
  - Developers can write their own Sources or Sinks

Flume: High-Level Overview

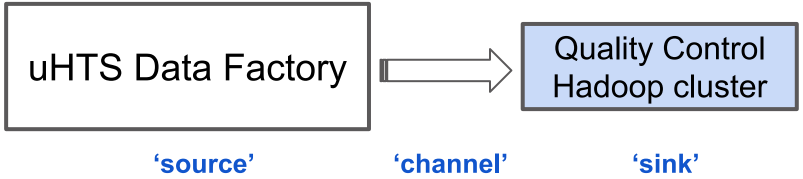


- Writes to multiple HDFS file formats (text, SequenceFile, JSON, Avro, others)
- Parallelized writes across many collectors – as much write throughput as required

- Optionally process incoming data: perform transformations, suppressions, metadata enrichment
- Each agent can be configured with an in-memory or durable channel

Flume Agent Characteristic

- **Each Flume agent has a *source*, a *sink* and a *channel***
- **Source**
  - Tells the node where to receive data from
- **Sink**
  - Tells the node where to send data to
- **Channel**
  - A queue between the Source and Sink
  - Can be in-memory only or ‘Durable’
    - Durable channels will not lose data if power is lost



Making it work

Configuration

- The configuration defines the sources, channels and sinks.
- Sources, channels and sinks are defined **per agent**

DataSource code

- Manage processing of a datastream created by the data factory
- Implements Source interface: configure(), start() and stop() methods.
- Opportunity to filter data as it streams into your “sink”.

Run statement:

```
$ bin/flume-ng agent --conf conf --conf-file myconfig.conf \
--name DataFactoryAgent -Dflume.root.logger=INFO,console
```

Configuration example

```
DataFactoryAgent.sources = DataFactory
DataFactoryAgent.channels = MemChannel
DataFactoryAgent.sinks = HDFS


DataFactoryAgent.sources.DataFactory.type = DataFactorySource
DataFactoryAgent.sources.DataFactory.channels = MemChannel
DataFactoryAgent.sources.DataFactory.keywords = <parameters for the DataFactorySource>

DataFactoryAgent.channels.MemChannel.type = memory
DataFactoryAgent.channels.MemChannel.capacity = 10000
DataFactoryAgent.channels.MemChannel.transactionCapacity = 100

DataFactoryAgent.sinks.HDFS.channel = MemChannel
DataFactoryAgent.sinks.HDFS.type = hdfs
DataFactoryAgent.sinks.HDFS.hdfs.path = hdfs://hadoop1:8020/user/factory/data/%Y/%m/%d/%H/
DataFactoryAgent.sinks.HDFS.hdfs.fileType = DataStream
DataFactoryAgent.sinks.HDFS.hdfs.writeFormat = Text
DataFactoryAgent.sinks.HDFS.hdfs.batchSize = 1000
DataFactoryAgent.sinks.HDFS.hdfs.rollCount = 10000
```

Data Factory Source

```
public class DataFactorySource extends AbstractSource implements EventDrivenSource, Configurable {

  private DataFactoryStream datafactoryStream;

  @Override
  public void configure(Context context) {
    Configuration cb = new ConfigurationBuilder();
    datafactoryStream = new DataFactoryStreamFactory(cb.build()).getInstance();
  }

  @Override
  public void start() {
    final ChannelProcessor channel = getChannelProcessor();
    EventListener listener = new EventListener() {
            // listen for events to filter or process - filter for exceptions or stalls in data flow
     };
    datafactoryStream.addListener(listener);
    super.start();
  }

  @Override
  public void stop() {
    datafactoryStream.shutdown();
    super.stop();
  }
}
```

The raw data stream from the factory

Stream initialization using Flume config file

Starts the stream processing thread. Can listen to events on the stream and do additional processing.

Stops the stream processing thread. Shutdown the stream.

Going deeper

To set up your own flume agent see:

https://flume.apache.org/FlumeUserGuide.html

- **the example in the guide is very easy to setup and run.**

Streaming Architectures

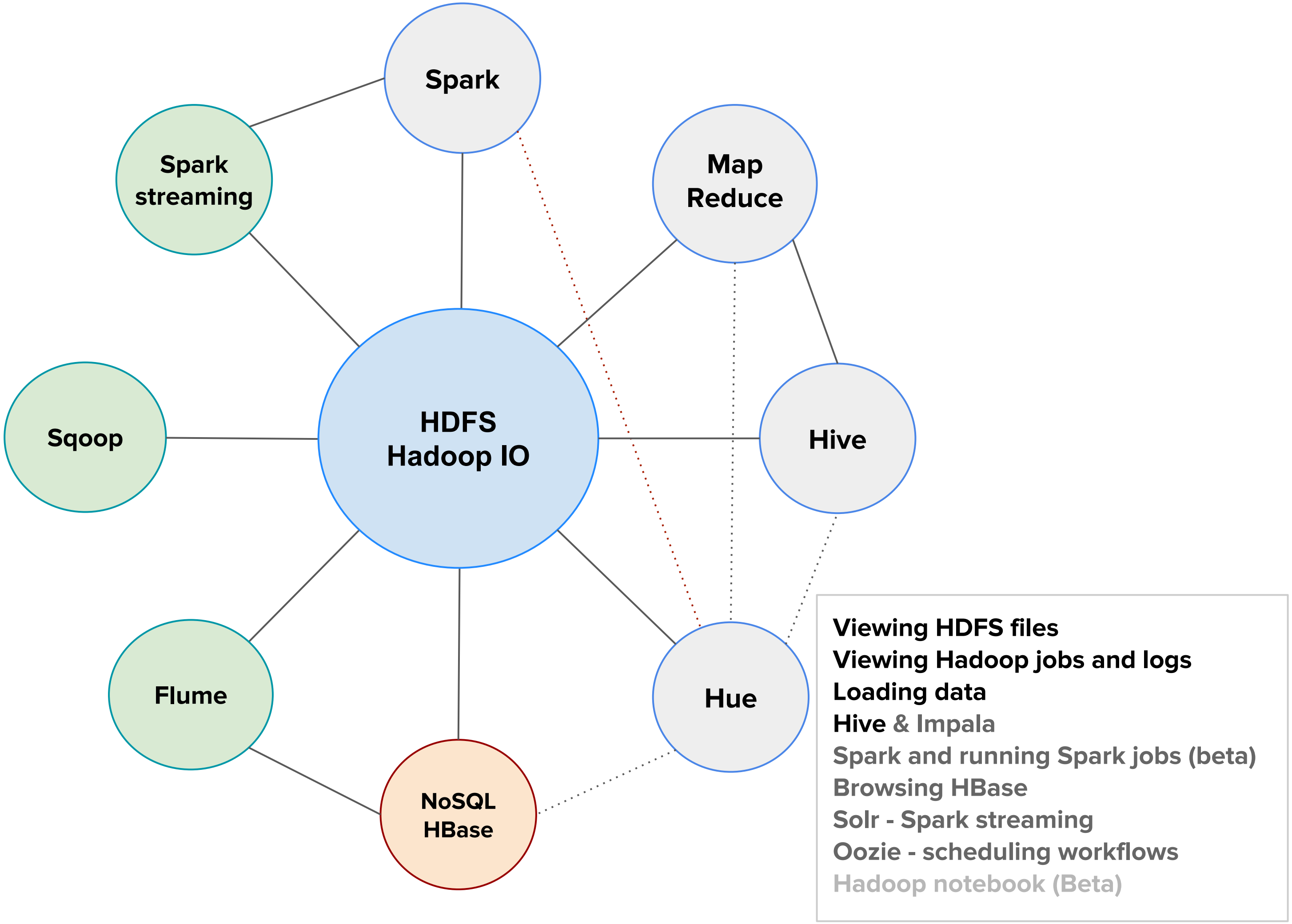
Kafka, Storm and HBase

Agenda

Overview of a distributed, streaming platform

Kafka

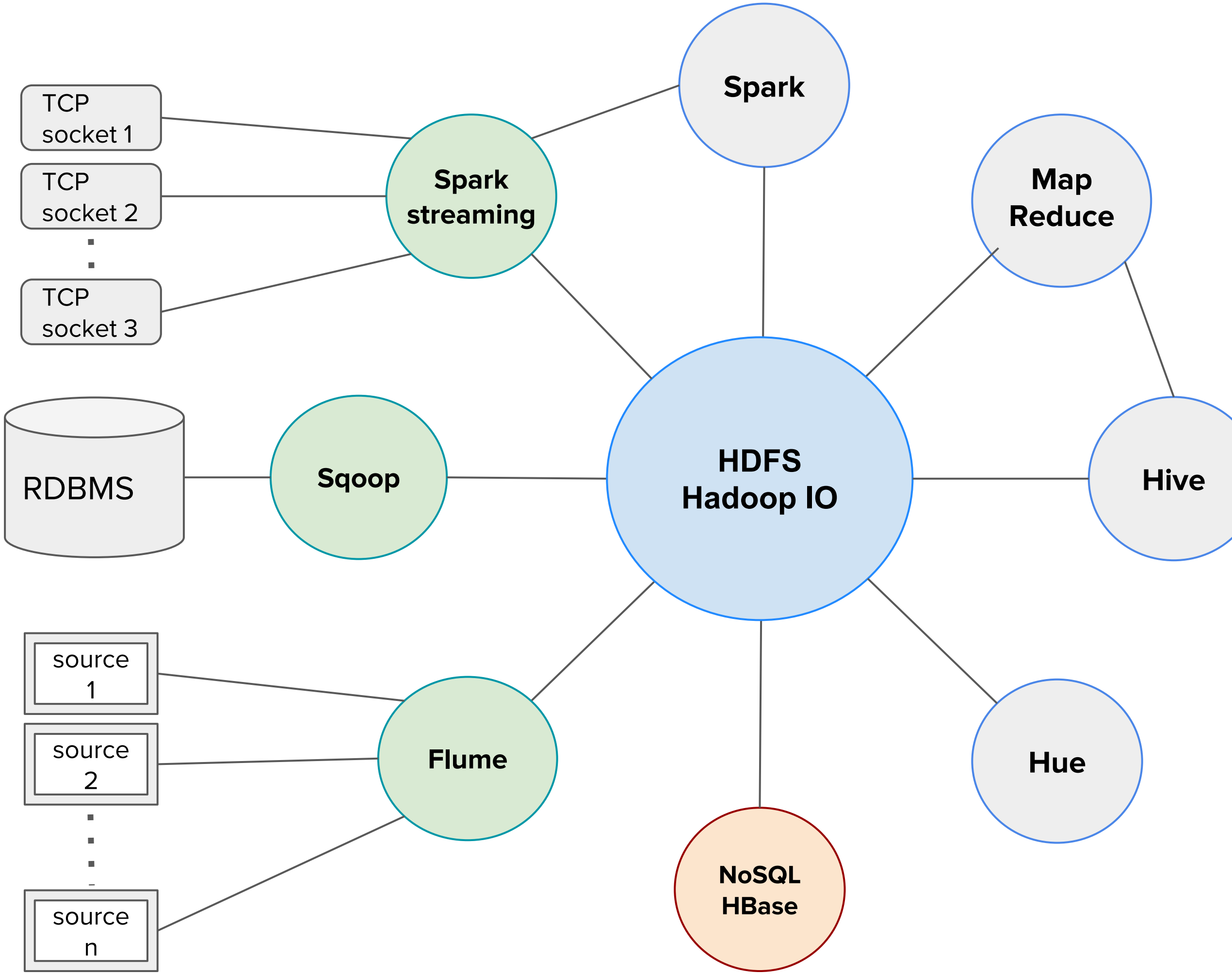
Storm - a tale for another day



Spark
Spark streaming
Map Reduce
Sqoop
HDFS Hadoop IO
Hive
Flume
NoSQL HBase
Hue
Viewing HDFS files
Viewing Hadoop jobs and logs
Loading data
Hive & Impala
Spark and running Spark jobs (beta)
Browsing HBase
Solr - Spark streaming
Oozie - scheduling workflows
Hadoop notebook (Beta)

Loading data
the hardest part

- time consuming
- many different formats
- many different solutions
  - Sqoop for databases
  - Flume for instruments
  - Hadoop IO for files
  - Streaming for sockets



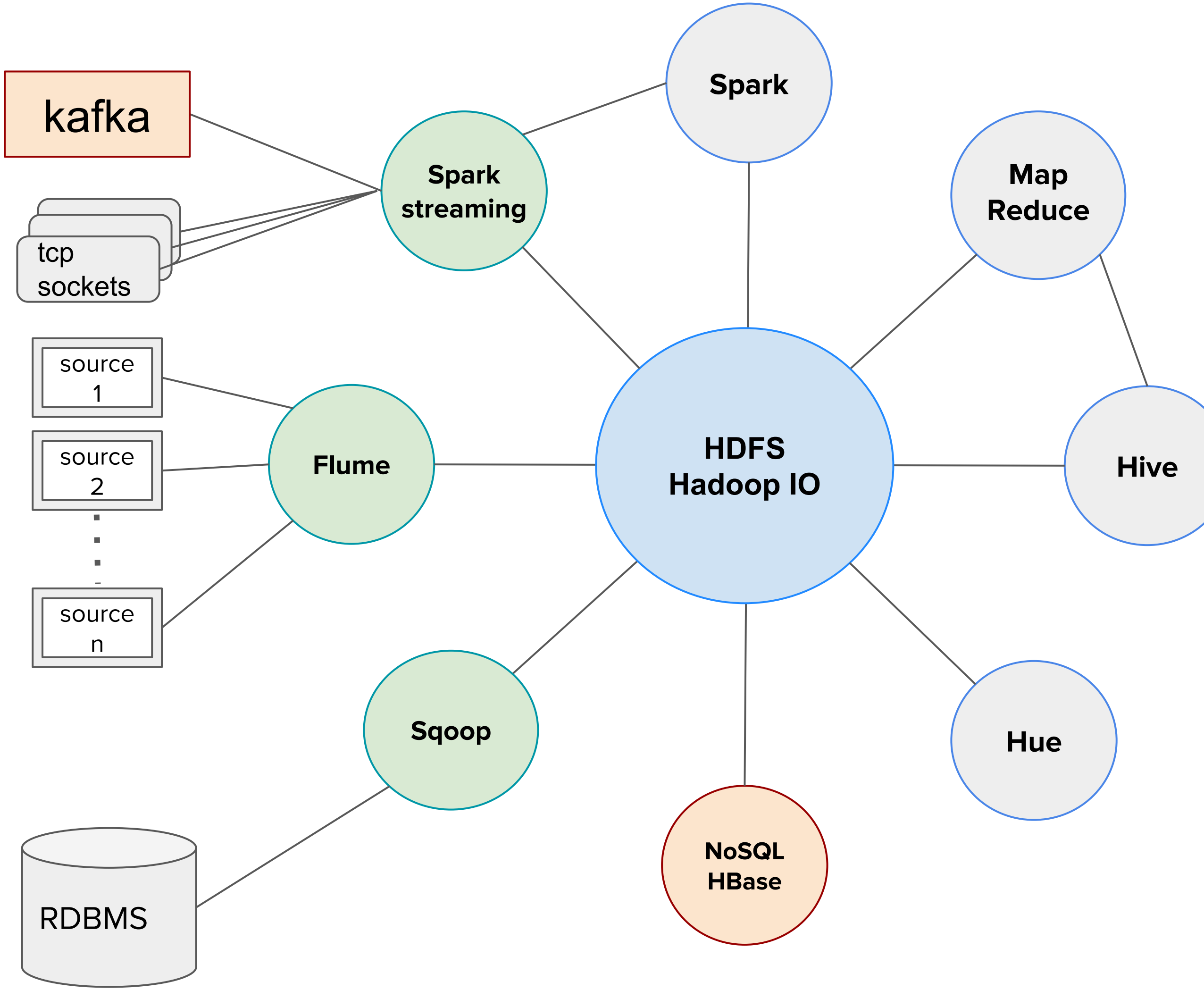
Loading data
the hardest part

Spark streaming inputs

- TCP sockets
- Kafka
- Flume
- HDFS/S3
- Kinesis

A great solution is Kafka

- doesn't solve all problems
- fast, distributed processing
- smart: uses the OS



Flume vs Kafka

Kafka presents a fast

Flume, however, provides an interceptor that can do event processing

Cloudera suggests wrapping Flume in Kafka

http://blog.cloudera.com/blog/2014/11/flafka-apache-flume-meets-apache-kafka-for-event-processing/

https://www.cloudera.com/documentation/kafka/latest/topics/kafka_flume.html

Kafka

http://kafka.apache.org/documentation.html

Kafka features

Designed as a unified platform for handling all the real-time data feeds for a large company.

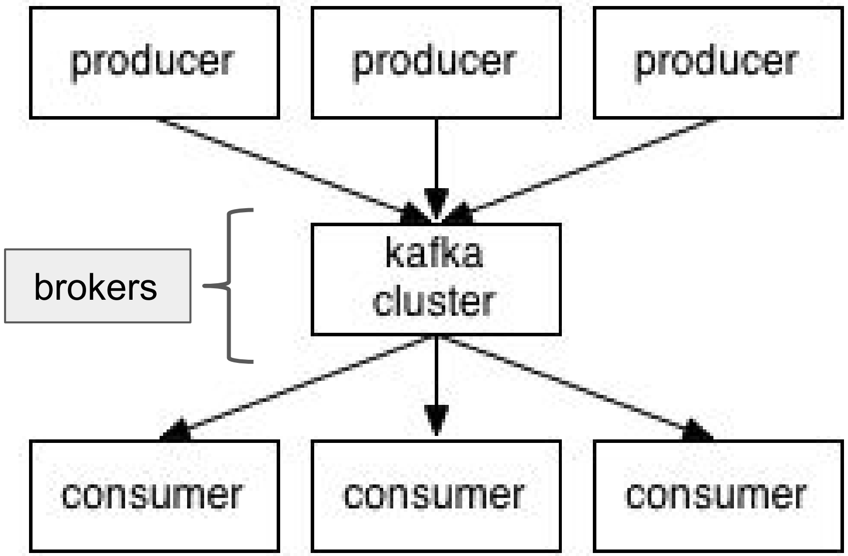
Supports

- **high volume event streams**
- **large data backlogs**
- **low-latency message delivery**
- **creation of new, derived feeds**
- **fault-tolerance**

Result: a design akin to a database log

Kafka terminology

- **topics**: categories for message feeds
- **producers**: publish messages on a topic
- **brokers**: servers forming a **kafka cluster**
- **consumers**:
  - subscribe to topics
  - process a feed of published messages



Kafka overview

Kafka is a messaging system

- Distributed over multiple servers called “brokers”
- Processes messages in sets called “topics”
    - Topics are partitioned and distributed over the brokers
    - Brokers cooperate to maintain “topics”
    - Replication for reliability
- Commit logs
- publisher-subscriber
- uses Zookeeper

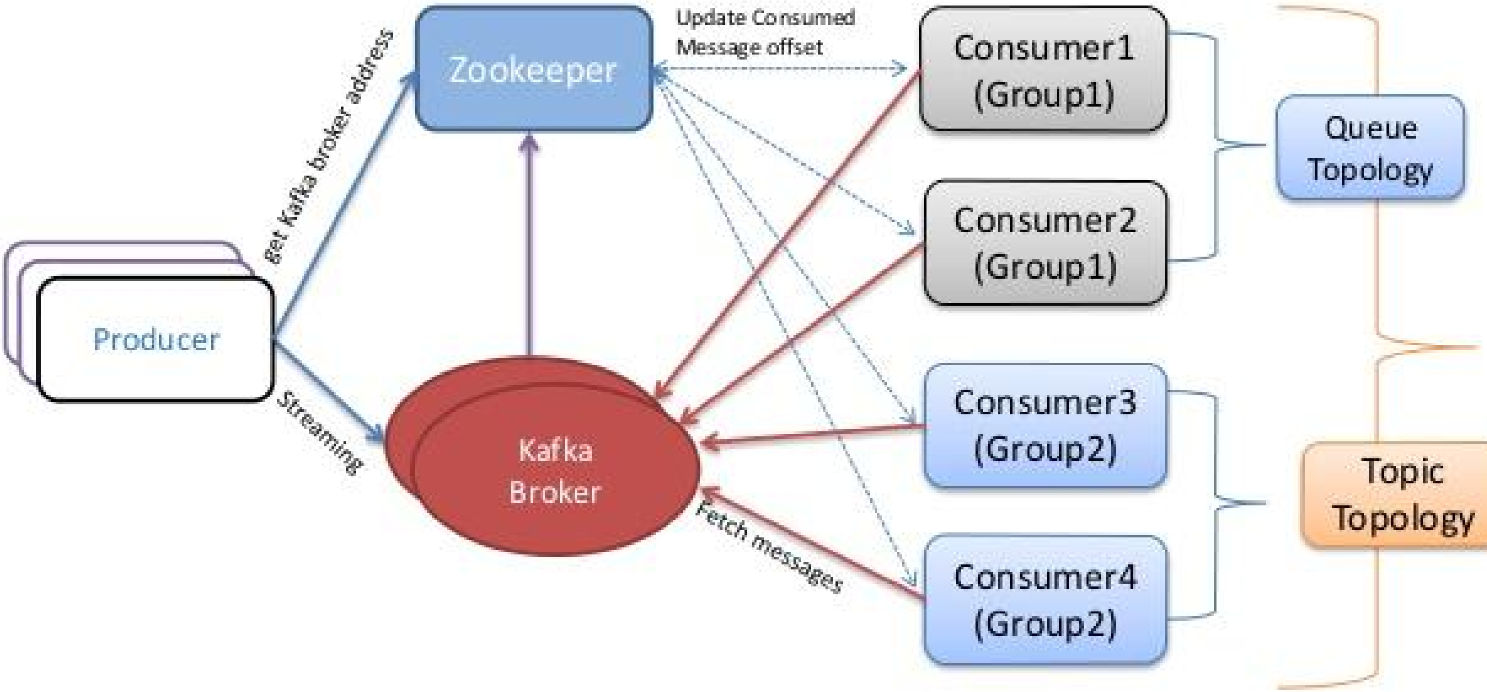
Aside: Kafka uses Java and the OS to advantage

Java supplants Scala (0.9 and 0.10): new Producers and Consumers

Main strength: Kafka has befriended the OS

Real time transfer

Broker does not ***Push*** messages to Consumer, Consumer ***Polls*** messages from Broker.



Anatomy of a topic

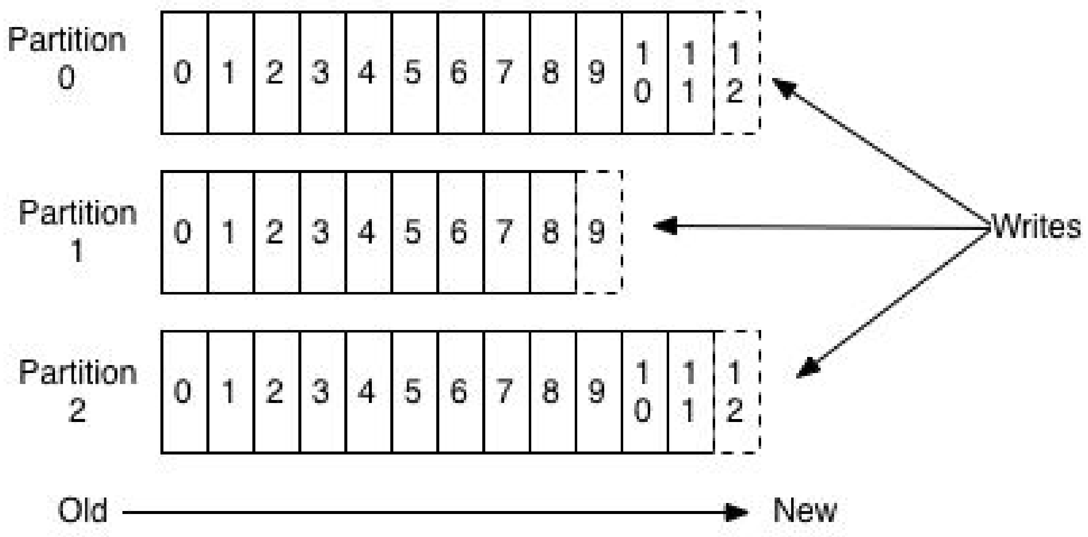
Data is partitioned when consumed

- partitions are spread across brokers
- data persists indefinitely
- time to live is configurable

Messages are immutable

Consumers can consume any message

- every message has an ‘offset’
- can consume most recent
- can consume using the offset



Fault tolerance and load balancing

- Leaders
  - Each partition has one server which acts as the "leader"
  - A leader handles all read and write requests for the partition
  - If the leader fails, one of the followers will automatically become the new leader
- Followers
  - For each partition, zero or more servers which act as "followers".
  - Followers replicate the leader.
  - Number of followers depends on replication number.
- Sometimes a leader, sometimes a follower
  - Each server acts as a leader for some of its partitions and a follower for others
  - load is well balanced within the cluster.

Producers

- Producers publish data to the topics of their choice.
- The producer decides which message to assign to which topic and partition
- Partitions may be assigned two ways:
  - May be done round-robin -- simply to balance load
  - May be done according to some semantic function (e.g. partition based on a k
    message)

Consumers

Consumers can obtain data in two ways

- subscribing to a topic
- using an assigned queue

Queues

- a pool of consumers reads from a server
- each message goes to one of them

Subscribed are broadcast to all consumers

Quickstart - with .sh files provided with Kafka

- download the code
- start zookeeper on localhost:2181, start kafka on localhost:9092
- give zookeeper a topic

```
> bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1
--partitions 1 --topic test
```

- send a message to Kafka

```
> bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test
```

- consume the message using zookeeper for directions to the broker

```
> bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic test
--from-beginning
```

http://kafka.apache.org/documentation.html#quickstart

More hands-on

To work with Kafka using the Java API:

https://github.com/tilogaat/HelloKafka

Kafka ❤ sequential disk access

- Kafka stores and caches messages on the file system
- Random reads are ~ 100,000 times slower than sequential access
  - Independent of medium - true for disk, SSD and tape
  - Example: six 7200rpm SATA RAID-5 array
    - random writes: 100k/sec
    - linear writes: 600MB/sec
- The OS is your friend
  - The modern OS diverts all free-memory to disk caching
  - Writes to disk (don't "flush") => the messages likely remain in the **pagecache**
  - using the filesystem and relying on pagecache is superior to an in-memory ca

*The Pathologies of Big Data, ACM Queue. http://queue.acm.org/detail.cfm?id=1563874

Why Kafka doesn’t use fancy storage paradigms

Typical queues used for messaging employ BTrees - O(log N)

- Incur the wrath of seek times
- Actual access times are superlinear

Kafka appends to store - flat structure, fast sequenctial access

With access to almost unlimited disk space - no need for deletes

Efficiency

Primary use cases is web activity data

- very high volume
- one page view may generate dozens of writes
- multiple consumers for the activity data

Two primary problems (now that reading and writing are efficient)

- too many small reads and writes (small IO problem)
- excessive byte copying

Solutions

- bucket up the messages into “message sets” and then compress
- use optimized pagecache to NIC buffer via **sendfile system call**
- both solutions depend on using the same message format on the producer, broker and consumer

Message guarantees

3 possible message delivery guarantees that could be provided:

- ***At most once***—Messages may be lost but are never redelivered.
- ***At least once***—Messages are never lost but may be redelivered.
- ***Exactly once***—this is what people actually want, each message is delivered once and only once.

Two basic problems:

- the durability guarantees for publishing a message - solved for Kafka!
- the guarantees when consuming a message - may be implemented by user.

Kafka guarantees at-least-once delivery

New: Kafka streaming (0.10)

- Raw input data consumed
- Data is then aggregated, enriched, or transformed into new topics

Example: A processing pipeline for recommending news articles

- producer: crawls article content from RSS feeds and publishes to an "articles" topic
- Kafka streaming normalizes (deduplicates) and publishs unique articles content to a new topic
- Kafka streaming could also add processing to find recommended content - another new topic.

Alternative stream processing tools include **Apache Storm** and **Apache Samza**.

A tale for another day: Storm

What follows is an overview

Connective devices and connected data

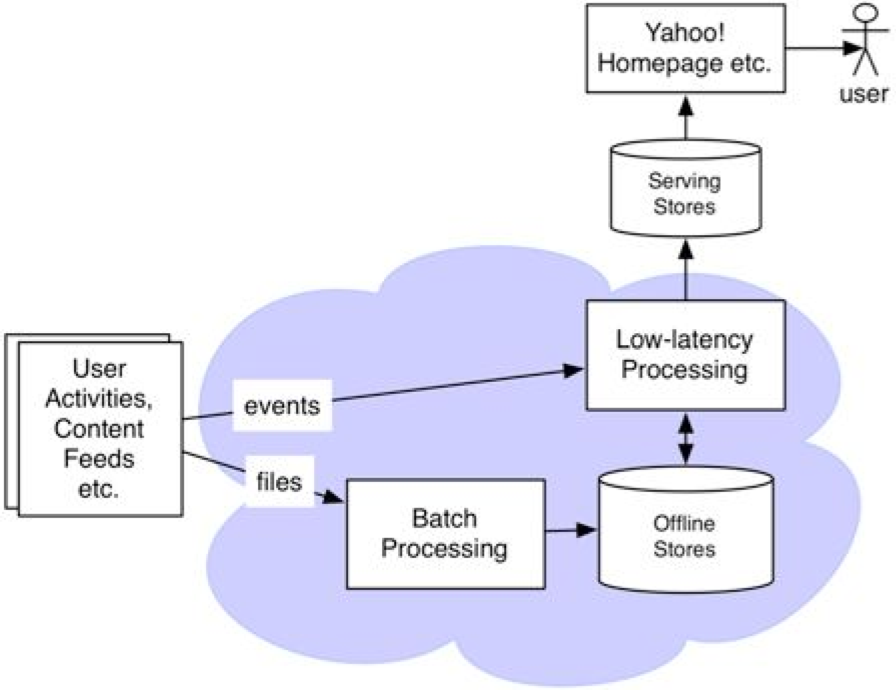
Tremendous amount of connected data

- events
  - clicks (and time between clicks)
  - search
- infer knowledge about user
  - analytics
  - targeting
  - personalization
  - from information to knowledge
    - how likely am I to be interested in fashion
    - intensity - 5 minutes or 20 days
  - user model

Yahoo! - problems with batch processing

- 20 billion events per day
  - compute user profiles and models in batch
  - jobs run 15 minutes to 3 hours
- still have signals coming in during batch
  - are the model assertions still true?
  - model is 15 minutes to 3 hours old
- ok for some analytics
- batch model
  - prior probabilities: probability of a user being a certain type
- need smaller windows
  - newer data can mix in
  - revise scoring on results using posterior probabilities
  - enter Storm

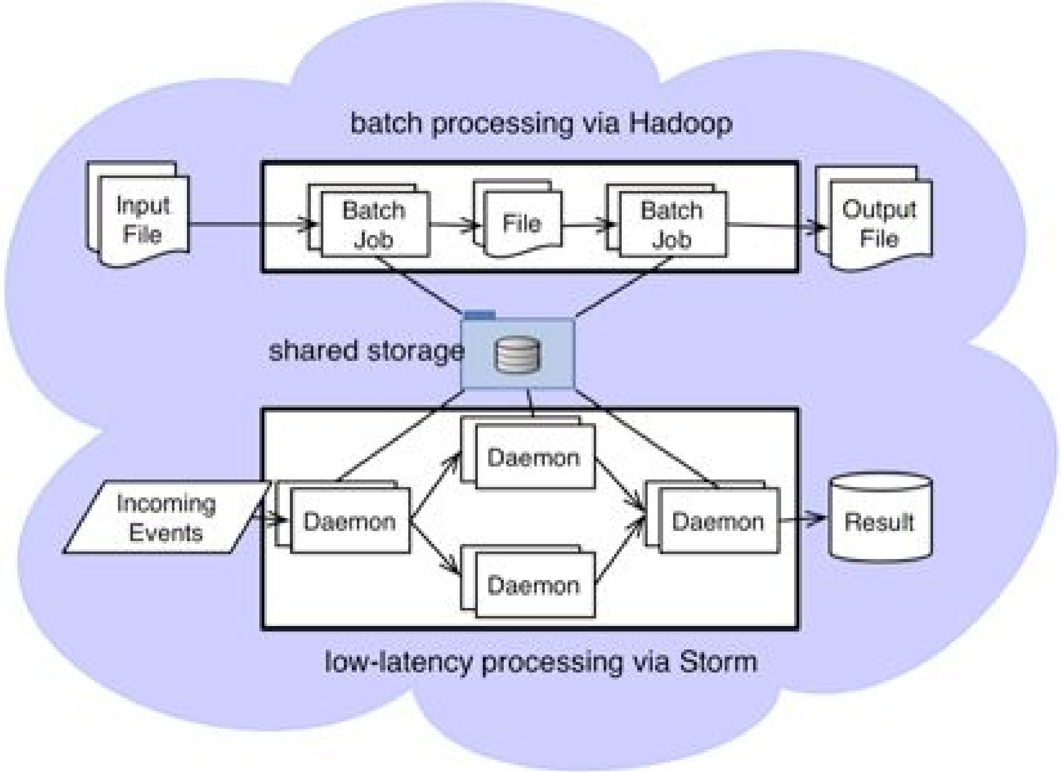
Using batch and microbatch simultaneously



Two systems

- MapReduce: large, batch model
- Storm: updates from streaming
- At Yahoo:
  - Storm: under 5 seconds
  - MapReduce: 1 - 3 hours

Both systems run on YARN



A topology in Storm wires
`data` and *functions* via a DAG.

Executes on many machines
like a MR job in Hadoop.

Another way to look at WordCount

```
( (1.1.1.1, "foo.com")
  (2.2.2.2, "bar.net")
  (3.3.3.3, "foo.com")
  (4.4.4.4, "foo.com")
  (5.5.5.5, "bar.net") )
```

DNS queries

⬇

```
("foo.com", "bar.net", "foo.com",
 "foo.com", "bar.net")
```

f

⬇

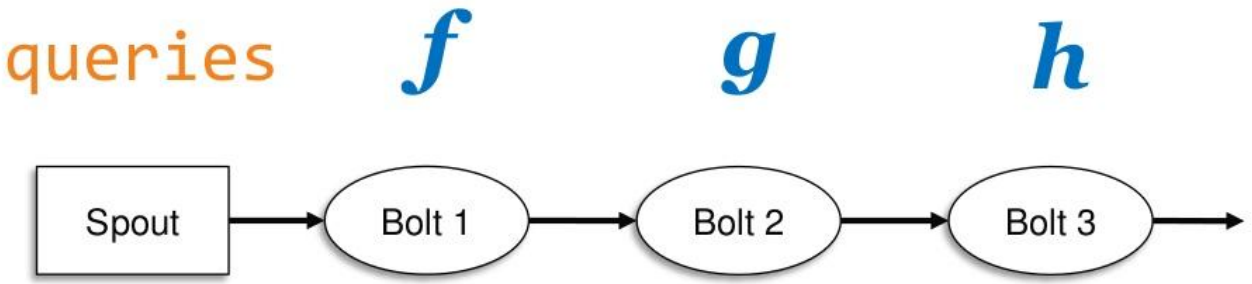
```
{"bar.net" -> 2, "foo.com" -> 3}
```

g

⬇

```
( ("foo.com", 3)
  ("bar.net", 2) )
```

h


queries
f
g
h
Spout
Bolt 1
Bolt 2
Bolt 3

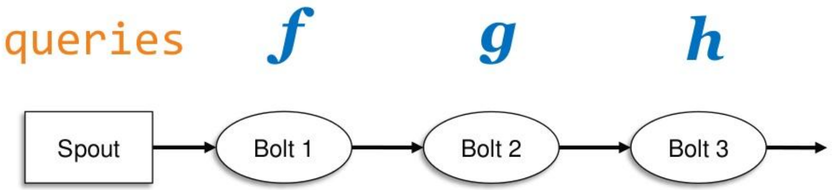
WordCount in Clojure

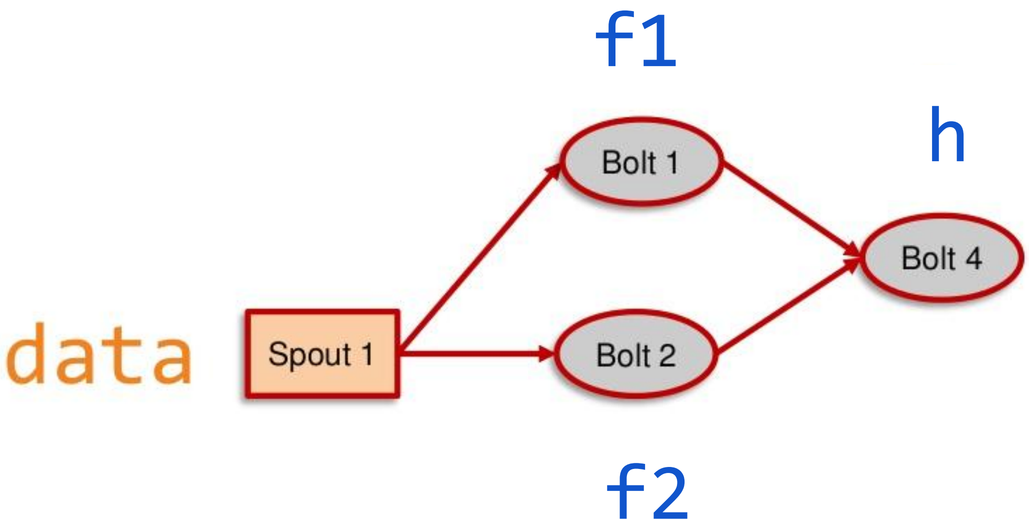
```
user> queries
(("1.1.1.1" "foo.com") ("2.2.2.2" "bar.net")
 ("3.3.3.3" "foo.com") ("4.4.4.4" "foo.com")
 ("5.5.5.5" "bar.net"))

user> (map second queries)
("foo.com" "bar.net" "foo.com" "foo.com" "bar.net")

user> (frequencies (map second queries))
{"bar.net" 2, "foo.com" 3}

user> (sort-by val > (frequencies (map second queries)))
(["foo.com" 3] ["bar.net" 2])
```





DAG: h(f1(data), f2(data))

Key points

- Storm user
  - Defines the DAG (topology of the job)
  - Provides the code
- Storm has a small codebase (~ 10,000 lines)
- Runs within its own architecture
- Want more?

http://www.michael-noll.com/blog/2014/09/15/apache-storm-training-deck-and-tutorial/

Extra info on HBase

More we don’t have time for

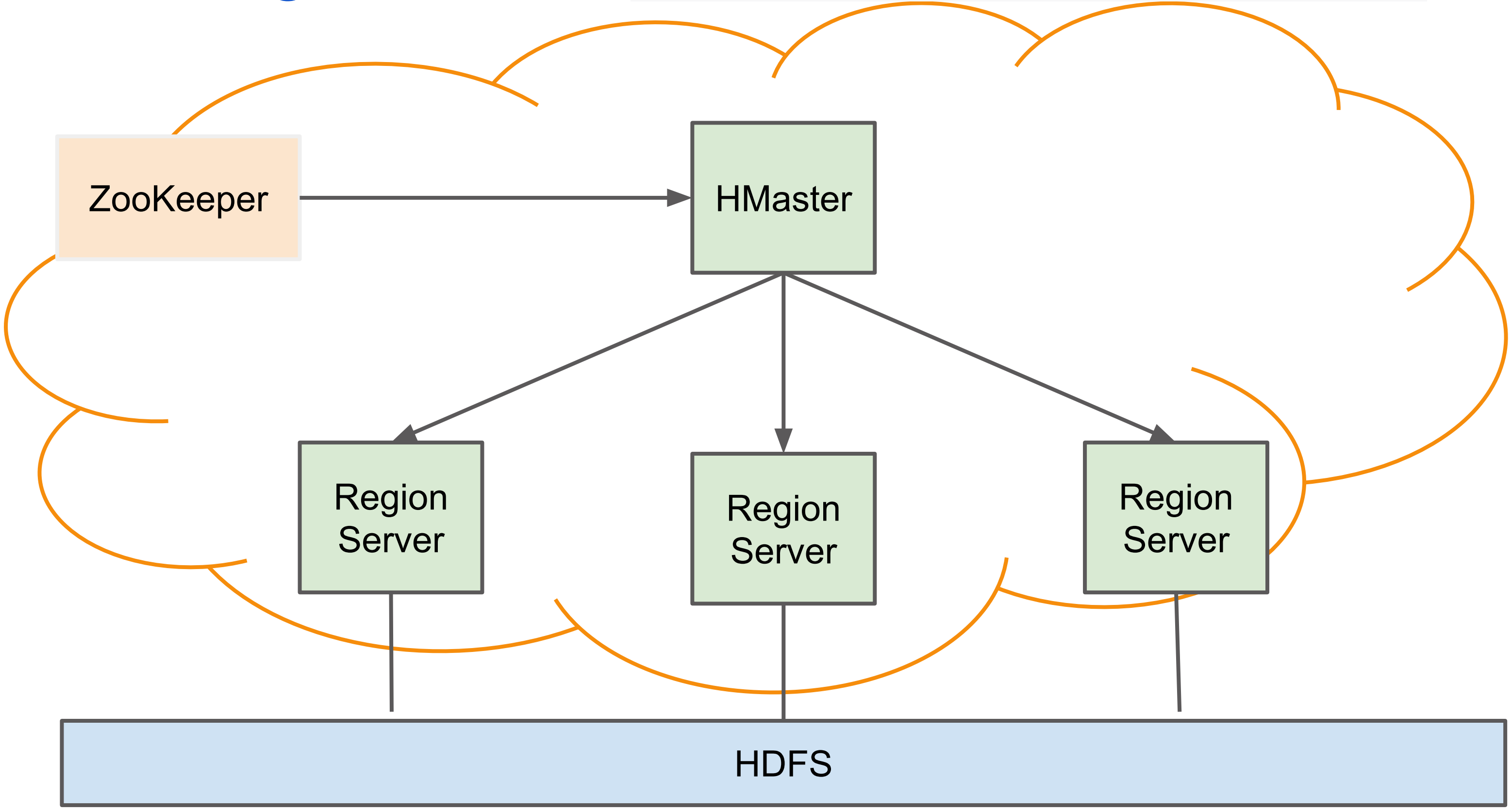
Installing HBase (admins only)

Decent guides:

- **Standalone installation**
  - assumes you already have hadoop installed as a single node
  - http://hbase.apache.org/book.html#quickstart
- **Distributed installation:**

  http://hbase.apache.org/book.html#configuration

Starting HBase: `$ bin/start-hbase.sh`



Checking HBase with JPS

Standalone mode

your machine

```
$ jps
20355 Jps
20137 HMaster
...
```

HMaster JVM: runs ZooKeeper, HMaster and Region Servers all in one JVM

Fully distributed mode

masternode

```
$ jps
20355 Jps
20071 HQuorumPeer
20137 HMaster
```

backup master

```
$ jps
15930 HRegionServer
16194 Jps
15838 HQuorumPeer
16010 HMaster
```

region servers

```
$ jps
15930 HRegionServer
16194 Jps
15838 HQuorumPeer
```

How data are stored in HFiles

File is broken into "blocks":

- **IndexBlock -** the full index for the data
- **DataBlock -** the actual data values
- **BloomBlock -** a hash of hashes based on the index block
- **MetaBlock -** info about the file and file organization

want more?
http://blog.cloudera.com/blog/2012/06/hbase-io-hfile-input-output/

Viewing the logs

In your VM, you can see the logs here:

quickstart.cloudera:60010/logs

HMaster logs

Click on "hbase-hbase-master-quickstart.cloudera.log":

- **Beginning of log:**
  - shows limit.conf for the master
  - shows environment variables for the master's CLI
  - shows environment for the ZooKeeper client
  - you can see the Master connecting to ZooKeeper
- **… a few hundred of startup log messages ...**
- **Near the end of log:**
  - shows the operation for create (search for *create 'scamper'*)
  - Note the time: 2015-06-09 06:.. etc

Region Server logs

Click on “hbase-hbase-regionserver-quickstart.cloudera.log”:

- **Beginning of log - same as master.log**
- **… a few hundred of startup log messages ...**
- **One second after you created the table:**
    - Opens ‘scamper’
    - Checks for edits
- **After you disabled the table:**
    - HRegion: Started closing ‘scamper’
    - Flushed the memstore and added table to hdfs
    - HRegion: Closed ‘scamper’