Introduction to Real Time Big Data Analysis

Who are we?

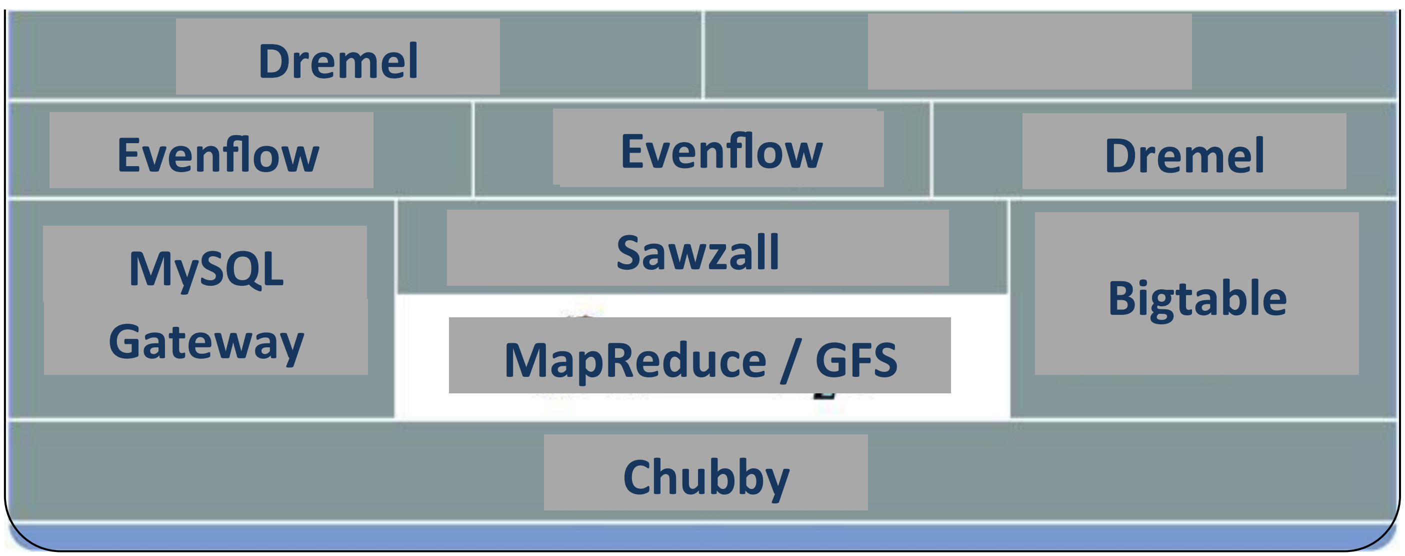
- Don Brown
- Adam Kunicki
- Amit Nithianandan
- Lee Sheng

Logistics

Who are you?

WWGD?

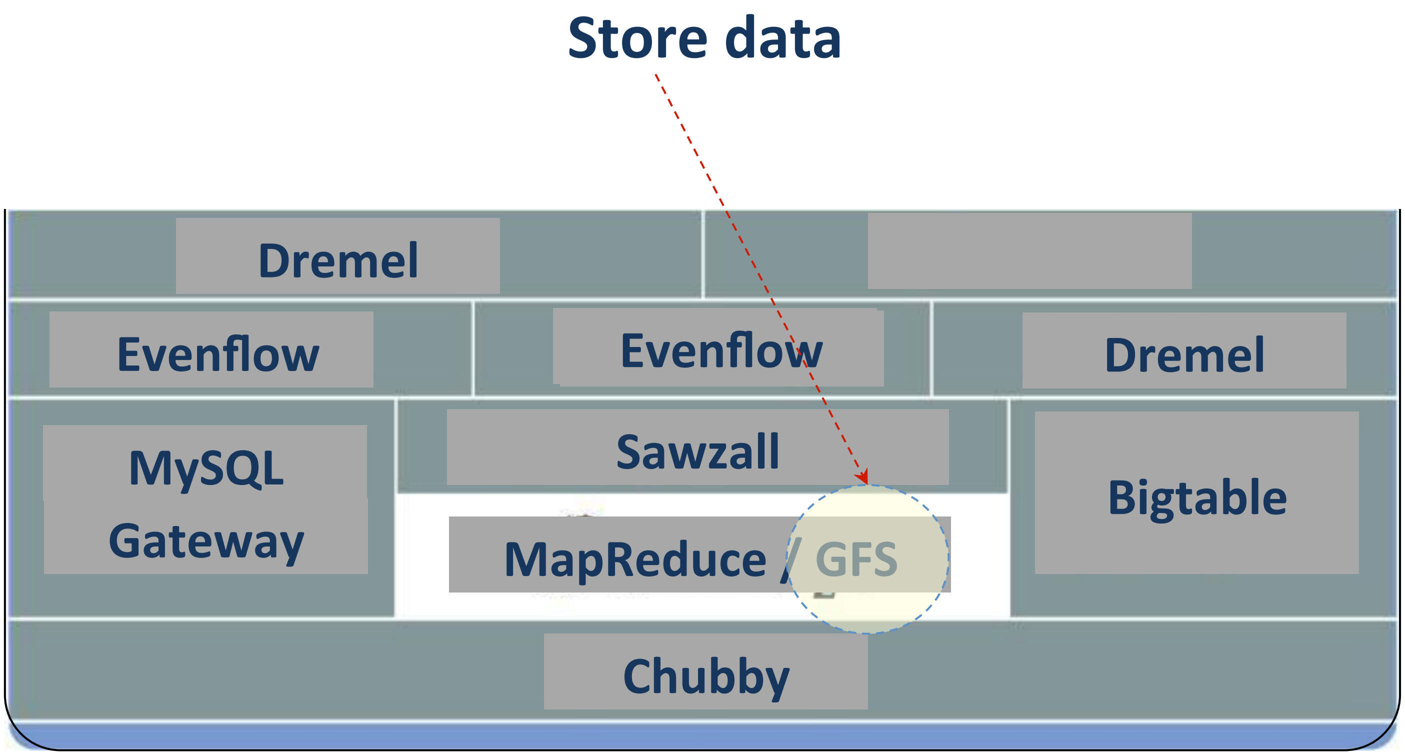
Google



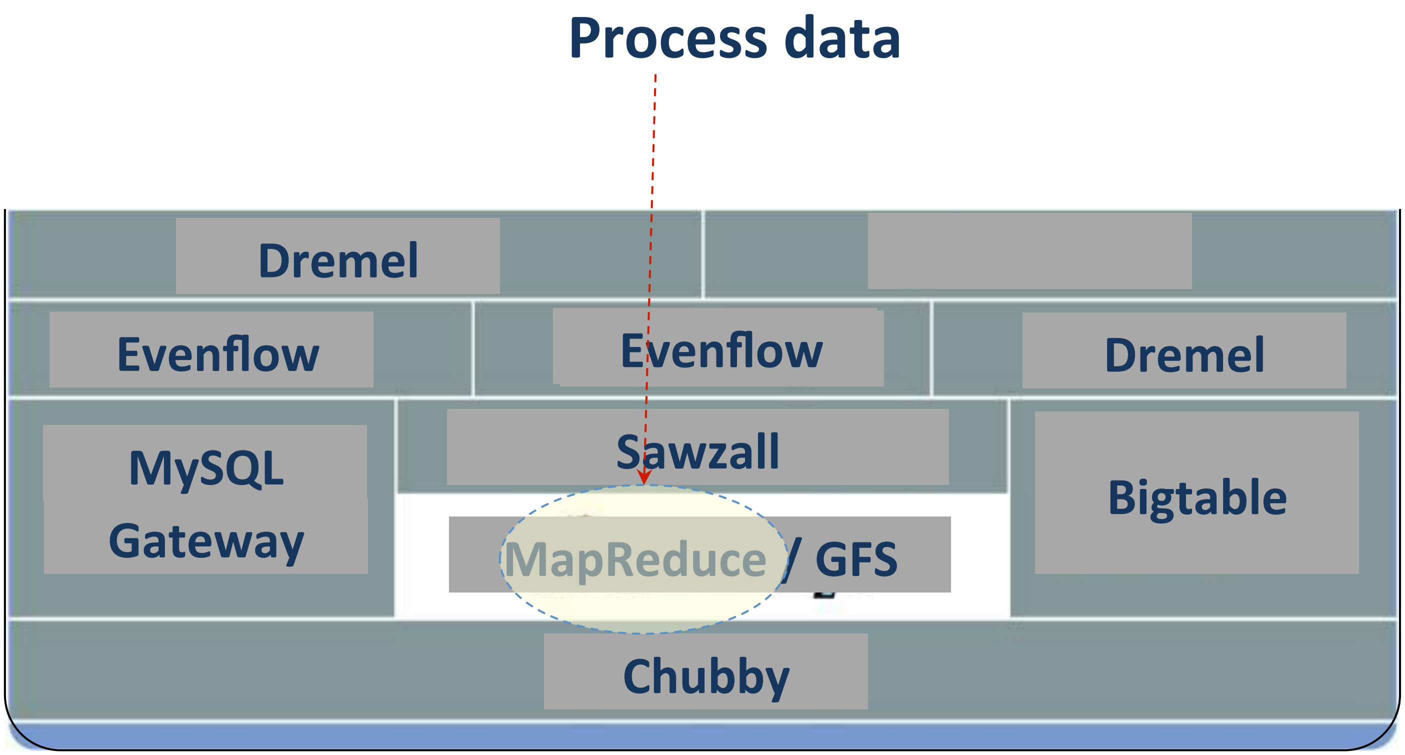
Dremel
Evenflow
Evenflow
Dremel
MySQL Gateway
Sawzall
MapReduce / GFS
Bigtable
Chubby

When they started to get big data, what did Google build?

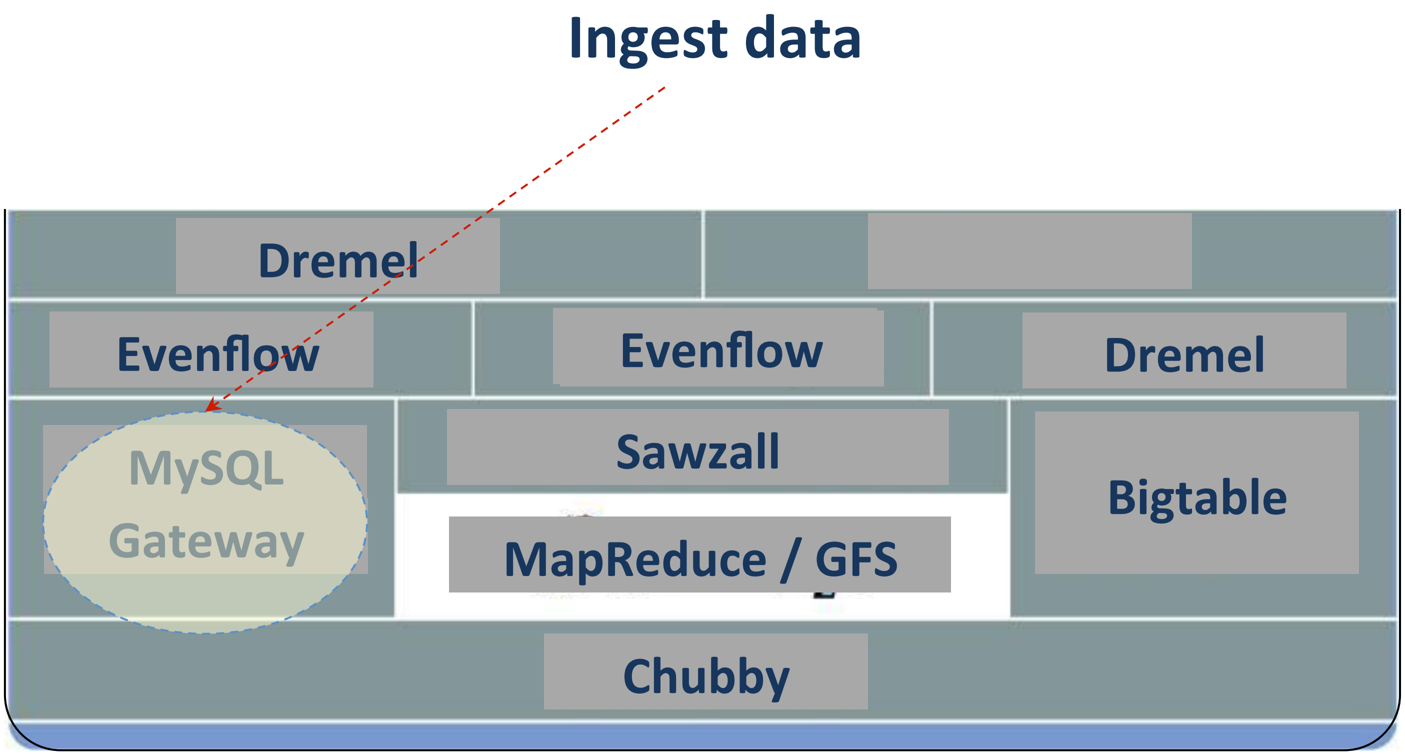
Google



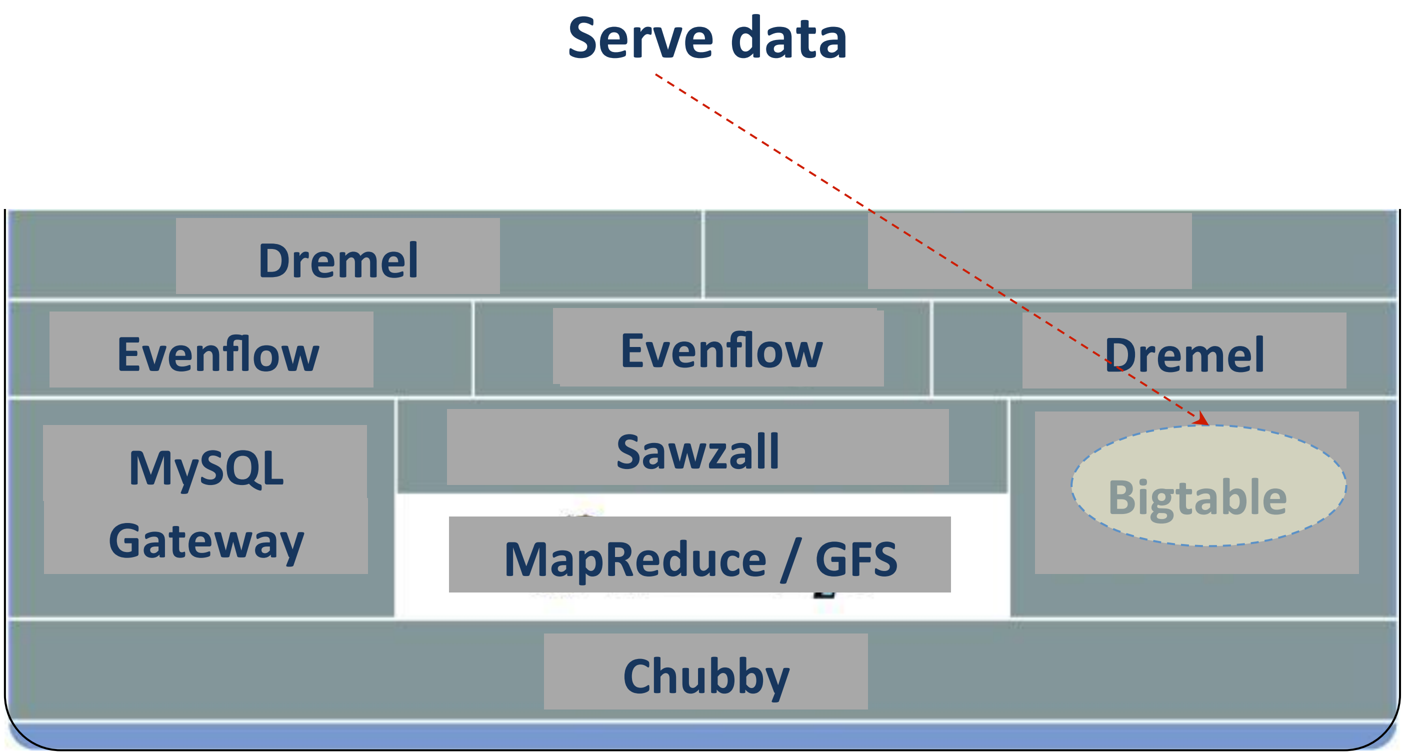
When they started to get big data, what did Google build?



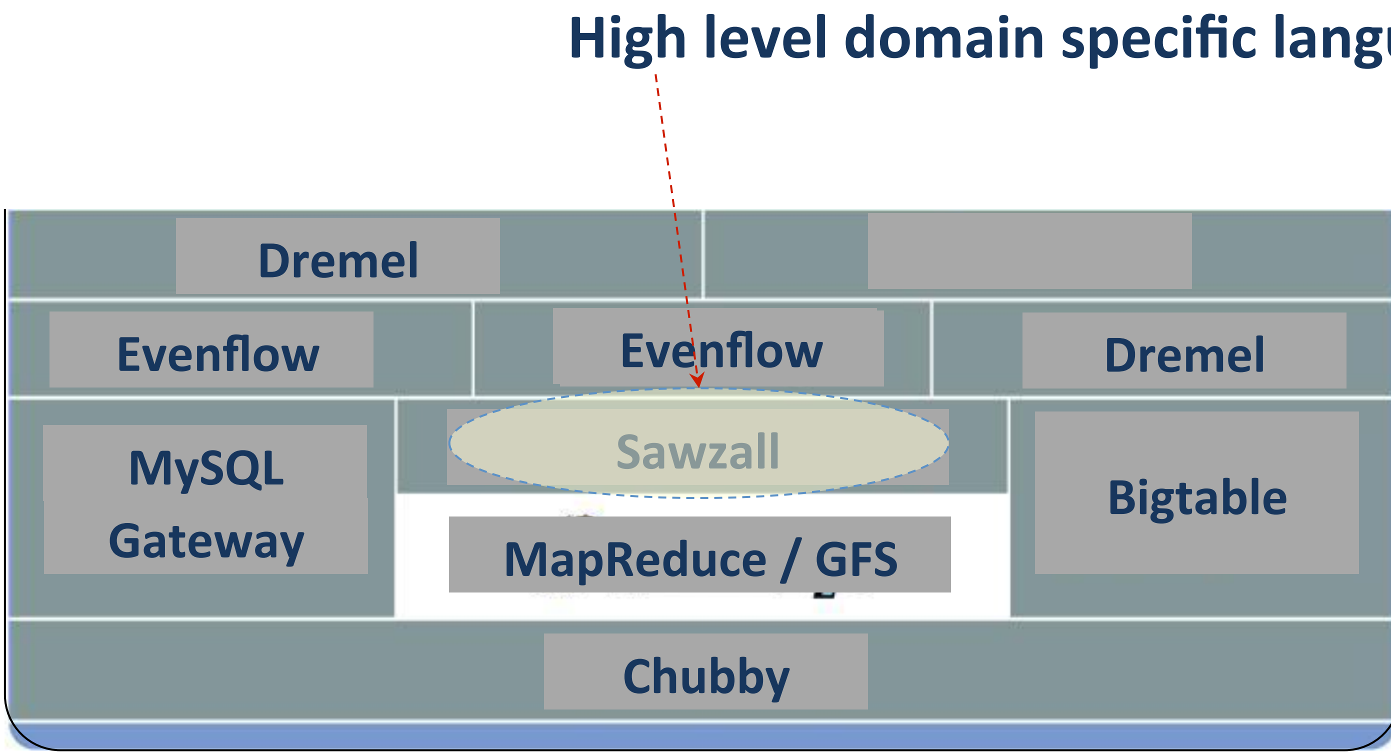
When they started to get big data, what did Google build?



When they started to get big data, what did Google build?



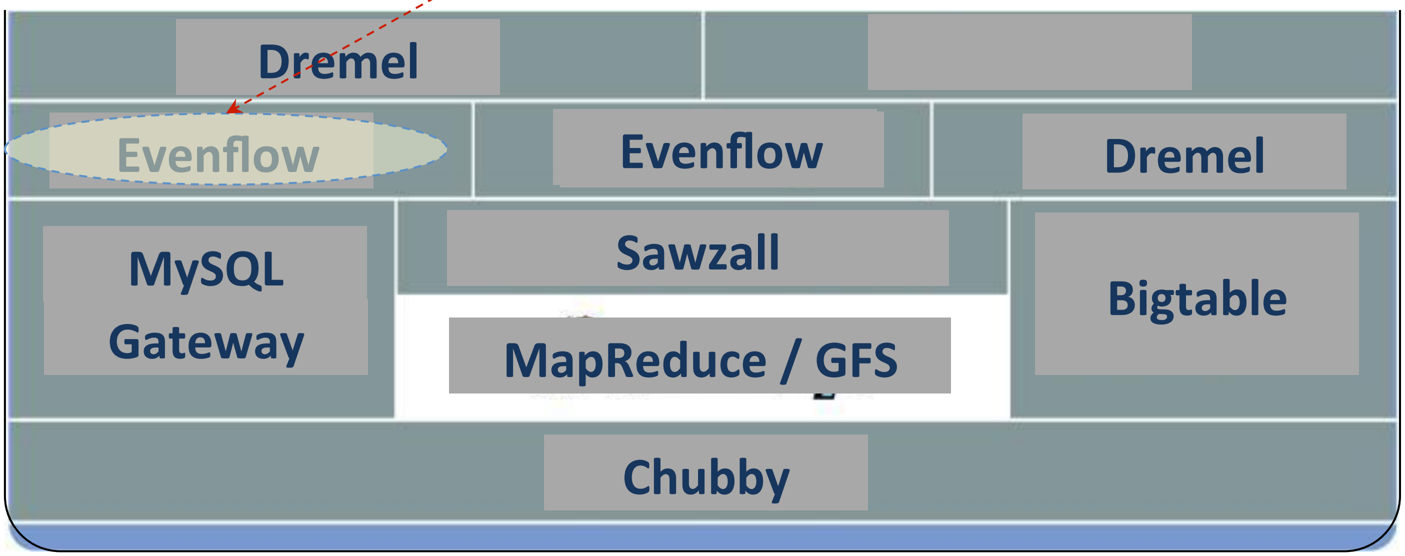
When they started to get big data, what did Google build?



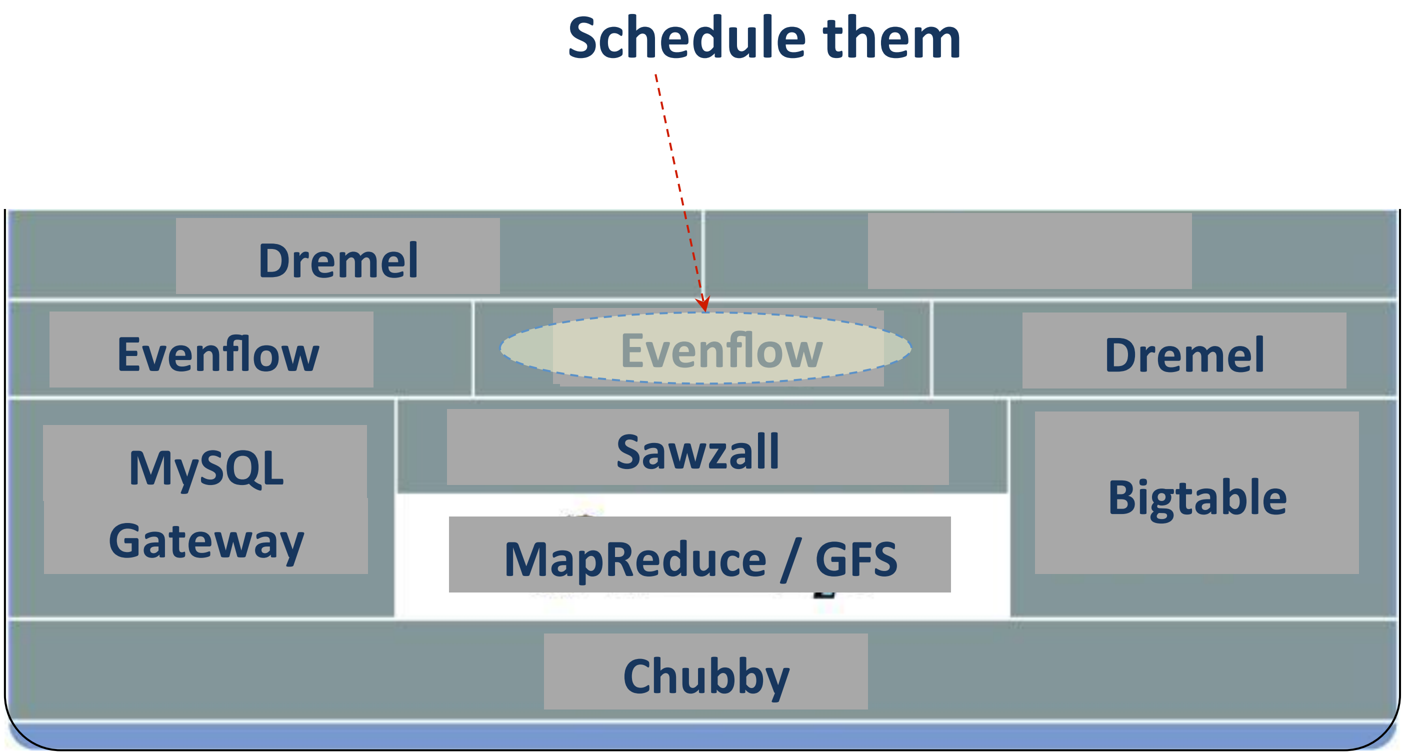
When they started to get big data, what did Google build?

Google

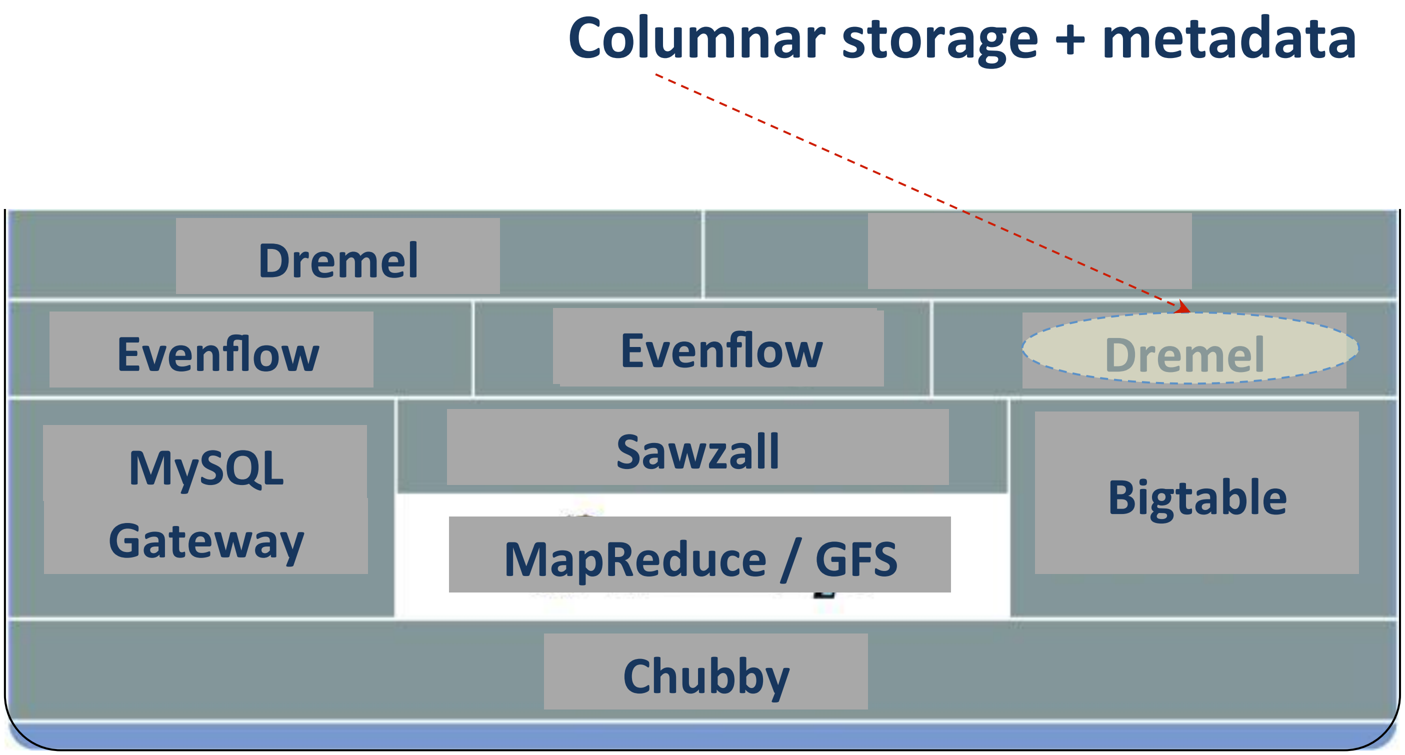
Chain together complex workloads



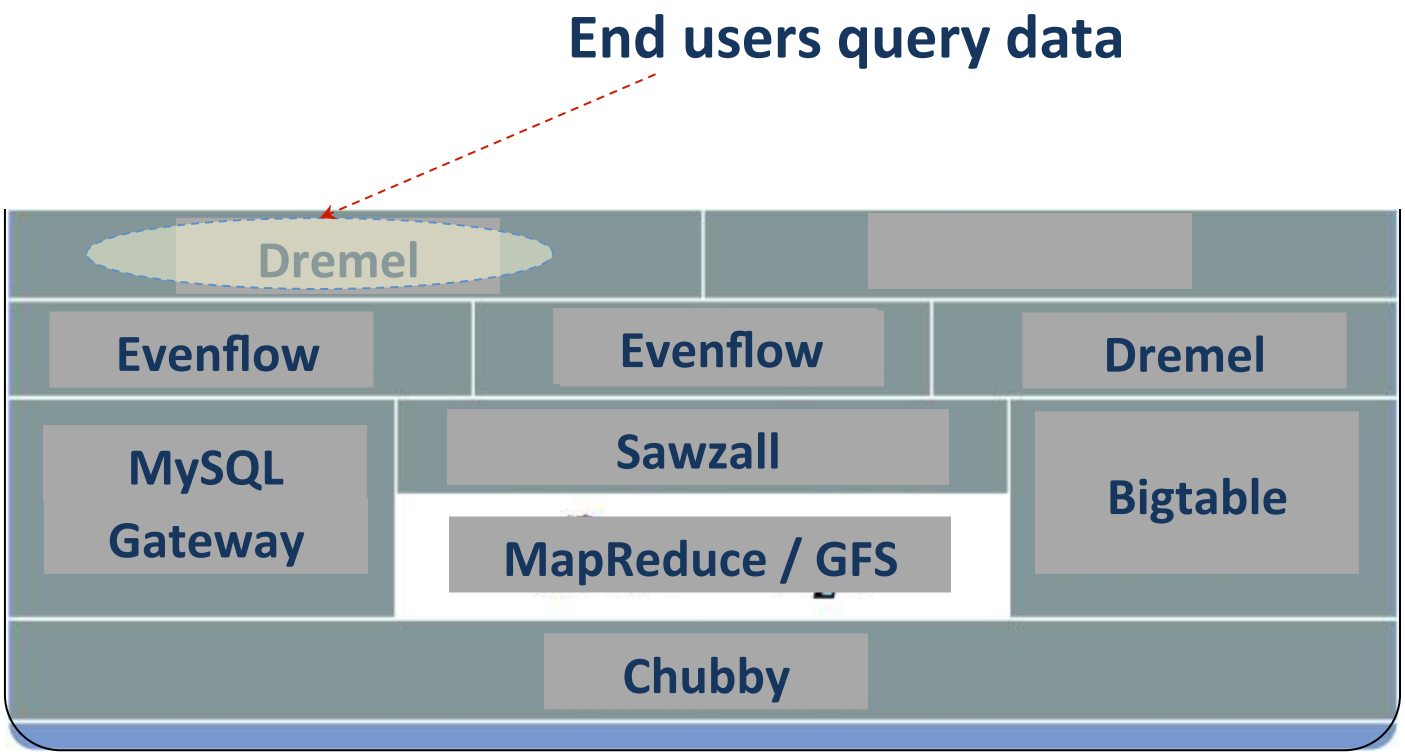
When they started to get big data, what did Google build?



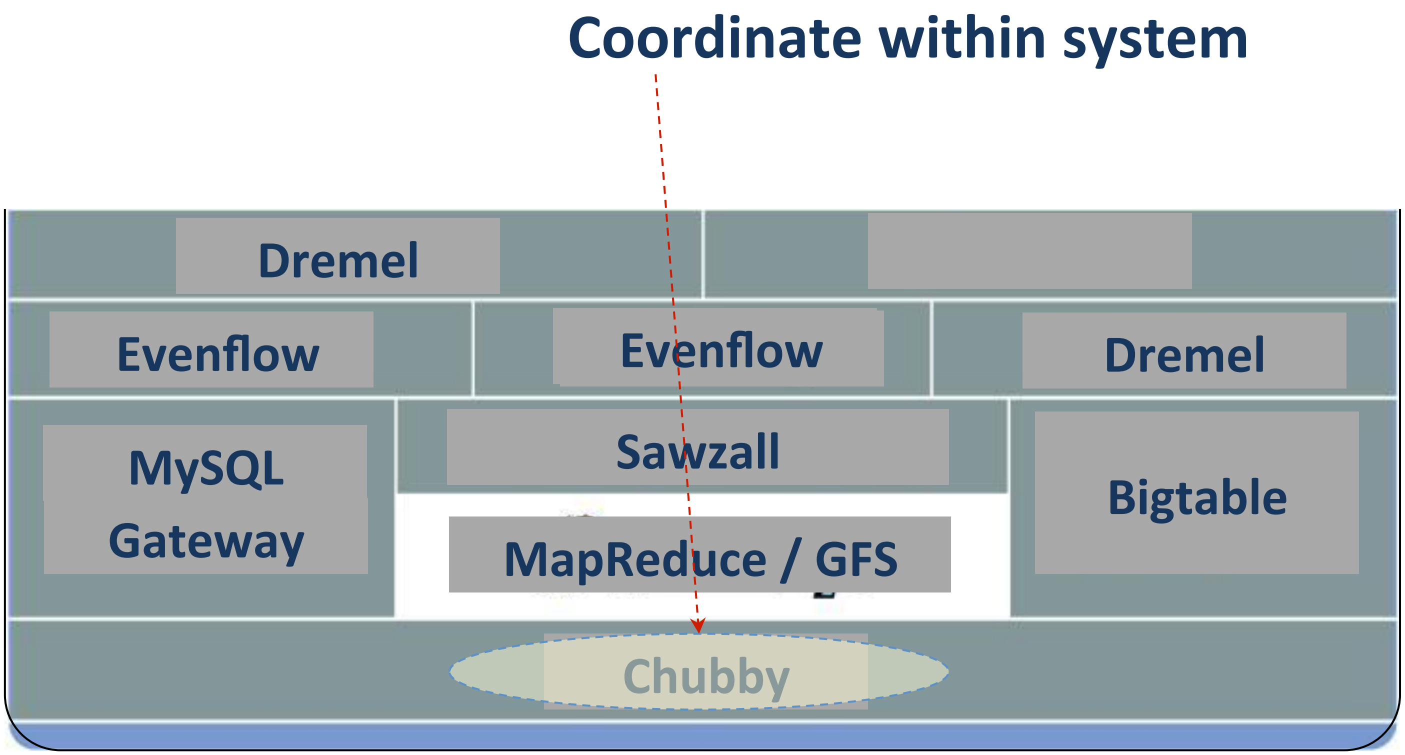
When they started to get big data, what did Google build?



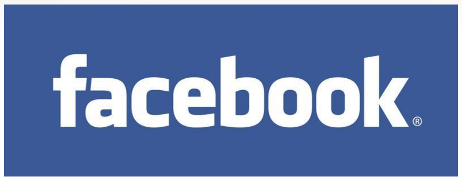
When they started to get big data, what did Google build?

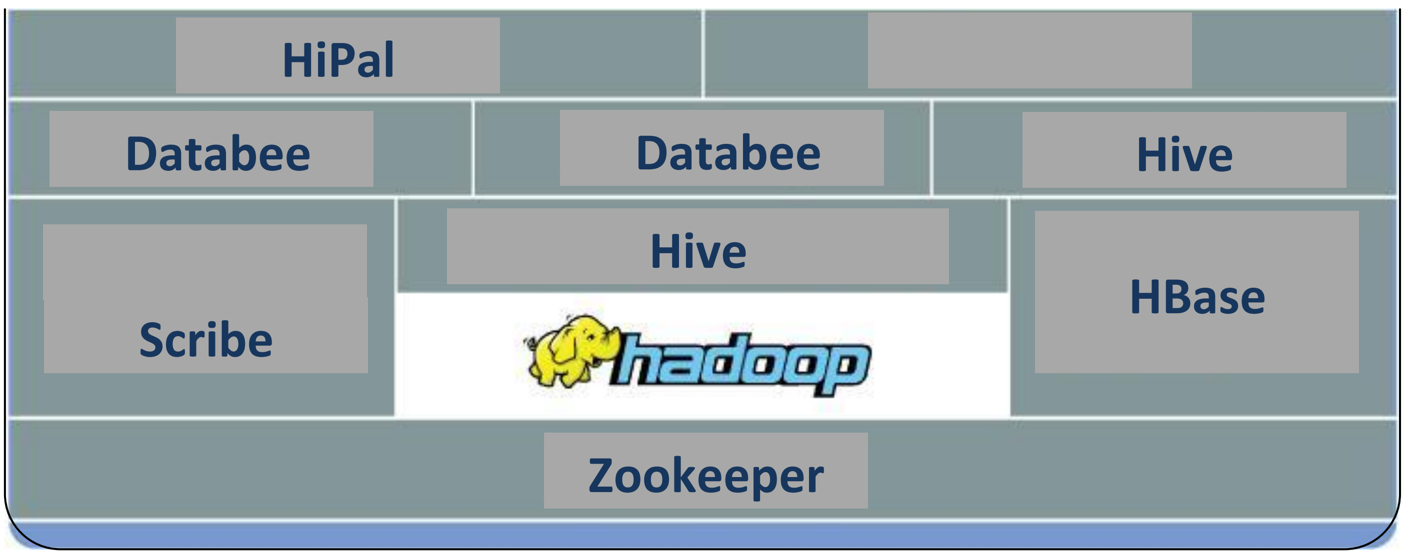


When they started to get big data, what did Google build?

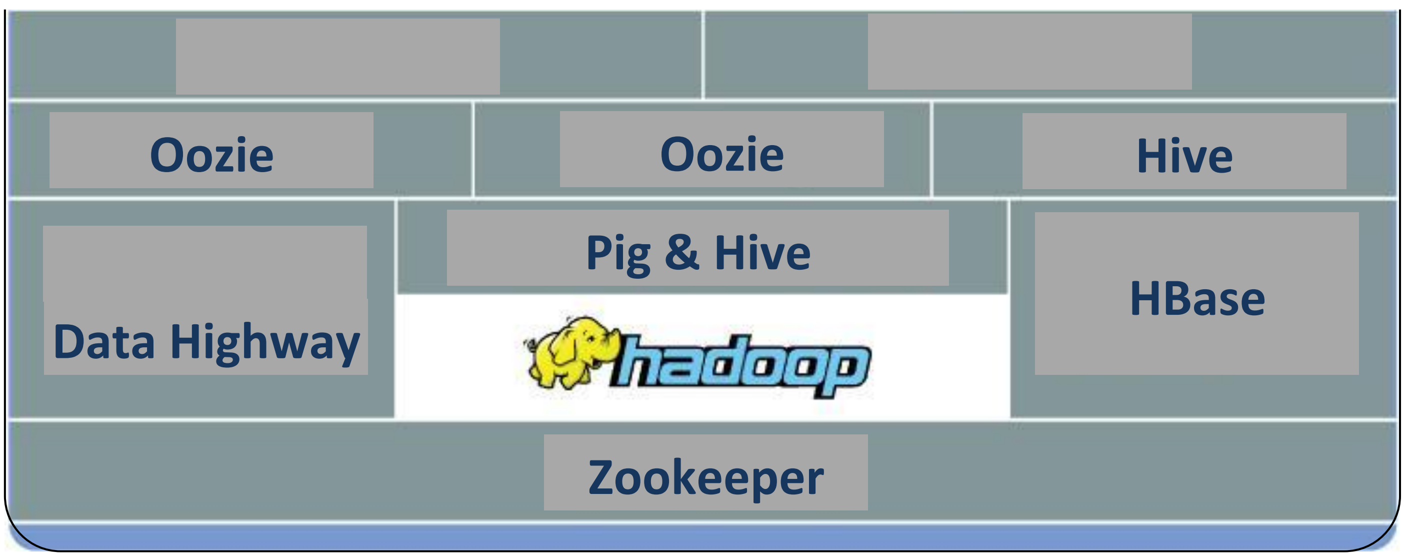


Lather…




HiPal
Databee
Databee
Hive
Hive
Scribe
hadoop
HBase
Zookeeper

Rinse…


Oozie
Oozie
Hive
Pig & Hive
Data Highway
hadoop
HBase
Zookeeper

Repeat…


Azkaban
Azkaban
Sqoop
Kafka
Pig
hadoop
Voldemort
Zookeeper

Hadoop is the Data Operating System

- Originally created by Doug Cutting and Mike Cafarella in 2004
- Original problem was a simple web crawler
- It has grown into a general purpose distributed storage and computation framework
- Based on the reference papers issued by Google

When Should I Use Hadoop

"When you need to store and process an enormous quantity of data of uncertain value."

--Josh Wills

Hadoop Introduction

Two components

- Hadoop Distributed File System
- Hadoop MapReduce

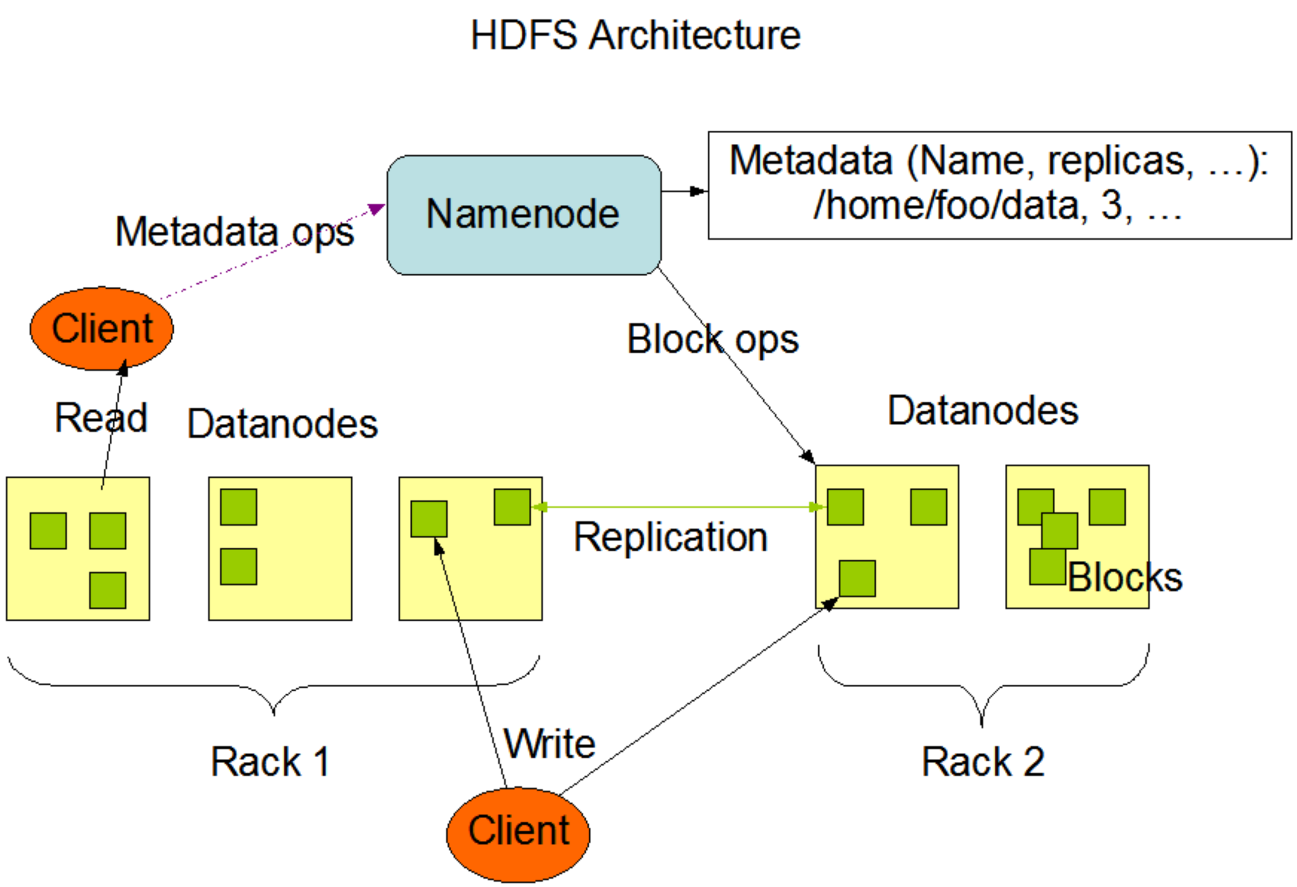
Major Contributors

- Cloudera
- Hortonworks
- Facebook
- Yahoo

Hadoop Distributed Filesystem

- Store petabytes (currently) of replicated data across thousands of heterogeneous nodes
- Master/Slave Architecture
  - Data is broken into 64MB or 128MB blocks and replicated in triplicate across the cluster
  - The Master or NameNode contains a location of all the blocks
  - The Slaves manage the blocks on their local filesystems
- Built on commodity hardware: no SAN, no RAID (they are actually a handicap

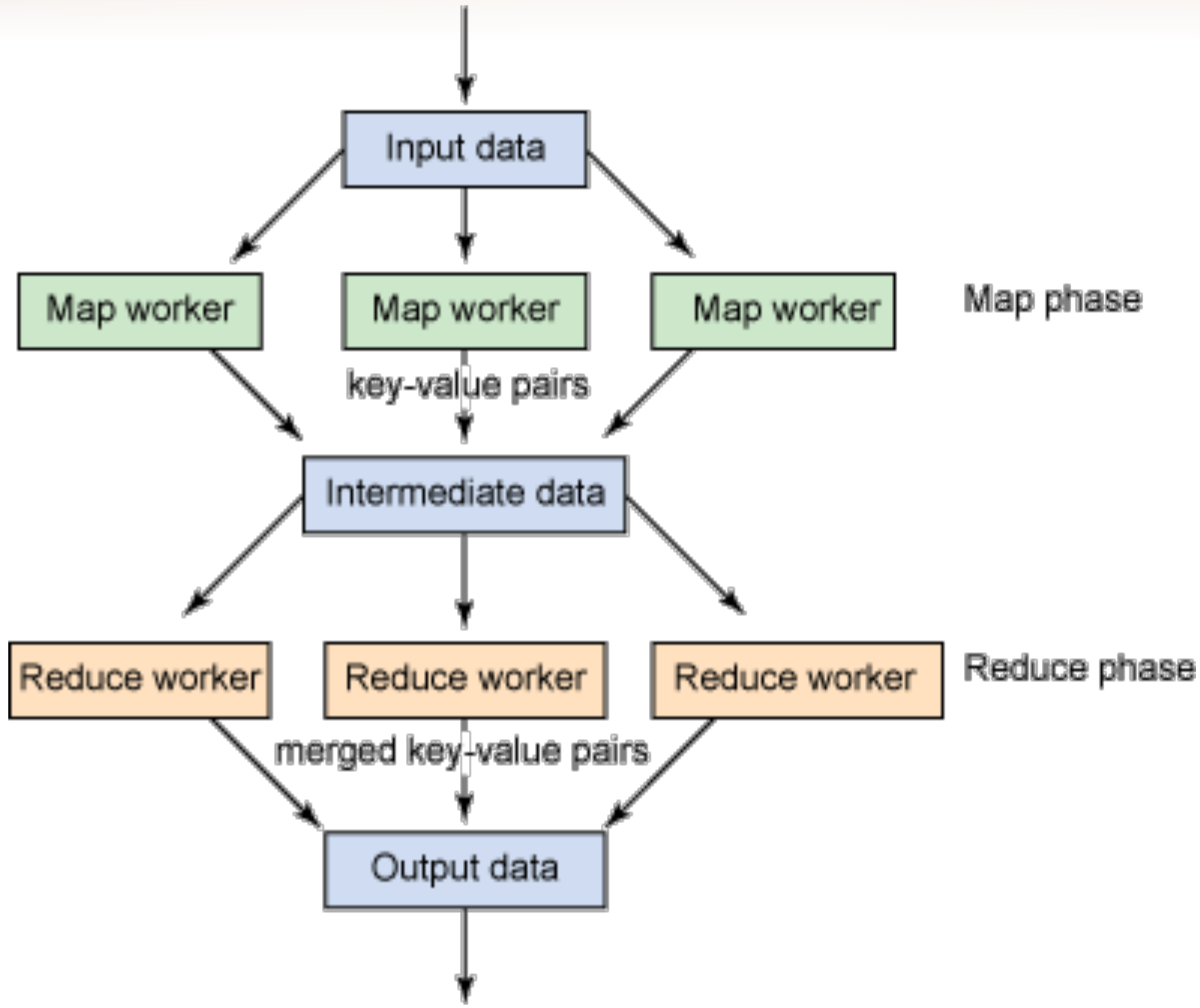
HDFS Architecture



MapReduce

- MapReduce != Hadoop
- MapReduce is a programming model for data processing in batch
- Existed for over 50 years
- The Hadoop implementation of MapReduce offers:
    - Fault Tolerance
    - Parellalization
    - Distribution
    - Data Locality

MapReduce Data Flow



Can You Figure Out What This Does?

```
users = load 'users.csv' as (username: chararray, age: int);
users_1825 = filter users by age >= 18 and age <= 25;

pages = load 'pages.csv' as (username: chararray, url: chararray);

joined = join users_1825 by username, pages by username;
grouped = group joined by url;
summed = foreach grouped generate group as url, COUNT(joined) AS views;
sorted = order summed by views desc;
top_5 = limit sorted 5;

store top_5 into 'top_5_sites.csv';
```

How About This?

You Probably Don't Actually Want to Write MapReduce

Popular abstractions:

- Pig
- Hive
- Cascading

Premature Optimization is the Root of All Evil

- Performance hit is dependent upon the types of jobs being run, but is commonly measured around 5-20% using the aformentioned abstractions
- Identify slow portions of the processing pipeline and recode those in MapReduce if necessary

Prominent Users

- Bank of America, Walmart, GE, Apple, JPMC, Home Depot, JPMC, Deutsche Bank
- 80% of the cellular traffic in this country hits a Hadoop cluster at some point

What Hadoop is Actually Used For….

ETL OFFLOAD

analytics

Hadoop is…

- designed to store and process petabyte style datasets in batch
- does not work well with most real time use cases
- does not handle small files well
- plays no favorites in terms of structured, semi-structured, and unstructured data

Random read/write, low latency storage options in Hadoop

What is HBase?

Distributed

Column-Oriented

Multidimensional

Highly-Available

Low latency

Commodity Storage

Why HBase?

How is this better than......?

NoSQL is a very broad term encompassing:

- Key value stores: MongoDB, CouchDB
- Document Stores: MongoDB, CouchDB
- Graph Databases: Neo4J, FlockDB
- Column-oriented Stores: HBase, Accumulo

Things to Consider When Choosing a NoSQL Solution

Consistency Model: Strict or Eventual

Persistence Model: In Memory or Persistent

Writes or Reads?

Scaling Limits

Cassandra

- Eventual Consistency on write
- Fairly easy to configure
  - Supports automatic peer discovery
- No single point of failure (all nodes provide all required functions)
- No support for atomic row-level operations

- HBase
- Write consistent
- Supports sequential scanning of rows
- Integrated with hadoop ecosystem (MapReduce)
- Coprocessor interface for running custom retrieval and data transformation code
- Stores all values as byte-arrays
- Hive integration

accumulo™

- Write consistent
- Supports sequential scanning of rows
- Integrated with hadoop ecosystem (MapReduce)
- Native support for cell-level security
- Iterator interface for custom data retrieval and transformation code
- Stores all values as byte-arrays

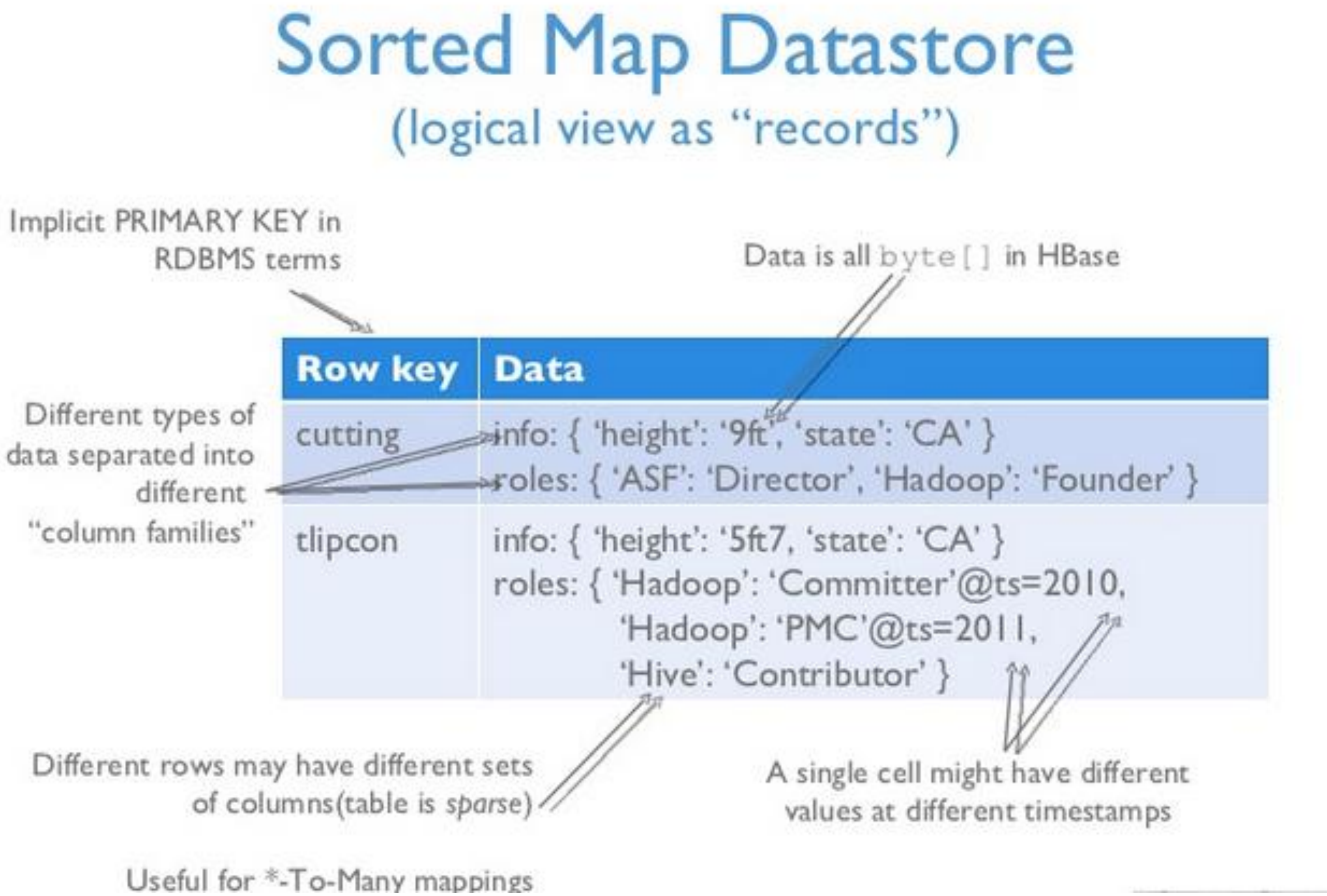
What do all these have in common?

- All are key/value based stores
- RowKey - Family - Qualifier -> Value
- Ideal for entity-centric storage of data

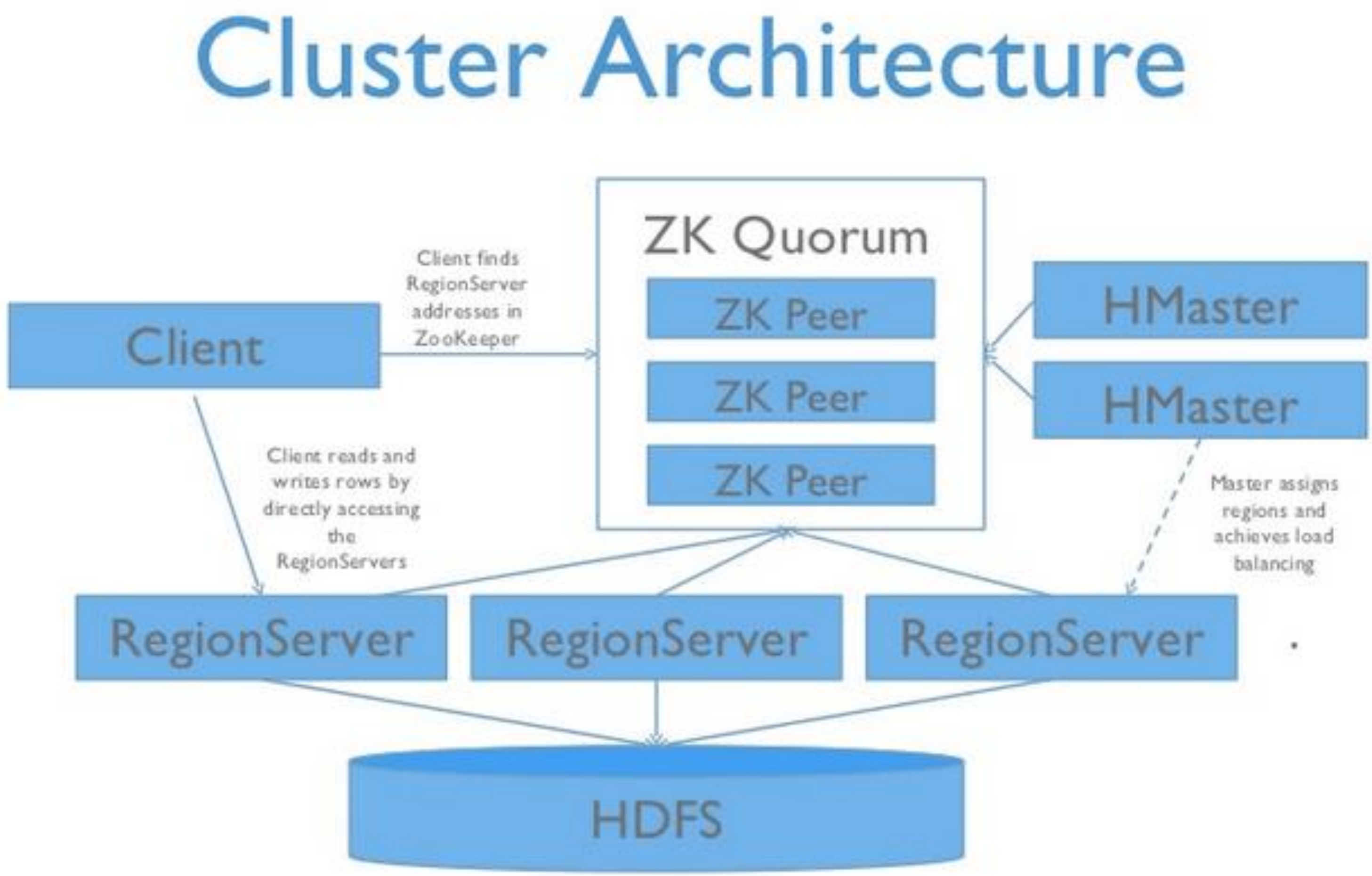
HBase Table Basics

- Tables are lexographically sorted by rowkey
- Table schema defines its Column Families (and only the Column Families)
- Any number of columns (millions)
- Any number of versions
- Unlike row-oriented databases, NULLs are free
- Everything is stored as byte arrays

HBase Table Example Layout



HBase Component Architecture



HBase Access Options

- Java API
- REST/HTTP
- Apache Thrift
- Hive/Pig

HBase API

- get
- put
- scan
- increment
- check and put
- delete
- MapReduce

HBase almost suits our needs

Pros:

- Random access reads and writes
- High throughput, low latency
- Integrates with Hadoop HDFS and MapReduce

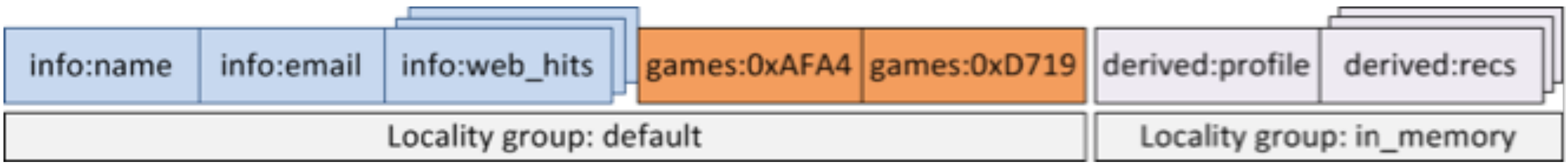
Cons:

- Raw byte array API for data
- Schema design is difficult

Entity-Centric Storage

- Kiji is designed to store entities
- All the data for a particular entity should be stored together, for easy access later on
  - Profile information, segmentation
  - Transactions, clicks, events

Kiji Data Model

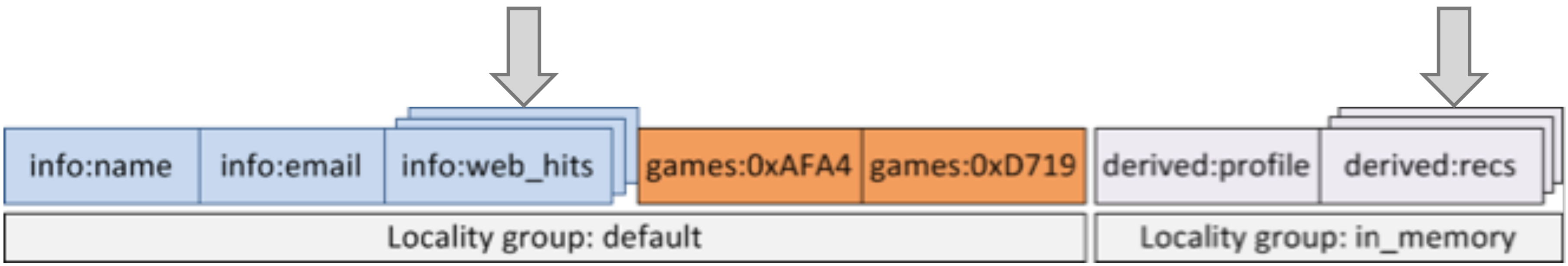


- Every row is identified by an **entity ID**
- Data is physically organized by **locality groups**
- Data is logically organized by **column families**
- Columns are identified with a **column qualifier**

What is an entity?

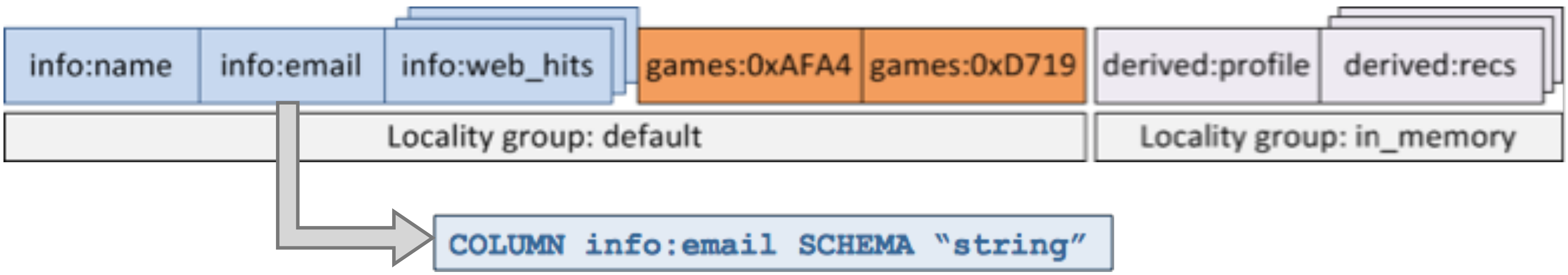
- In databases, an entity is an important thing or object that we want to capture data about
  - Customers, Users, Accounts, Devices
- Entity-Relationship Diagrams are the result of an inadequate storage system
  - Complex joins necessary to model an entire entity

Kiji Data Model: Time



- Data is also stored along a time dimension
  - A particular cell is identified by four coordinates (Entity ID, Column Family, Column Qualifier, Timestamp)
- Kiji can be used to store and analyze a timeseries

Kiji Data Model: Column Schemas



- Columns store Avro data
  - Primitives or complex records
  - Enables schema evolution
- Richer than storing binary data

Column Families

- **Group** Families
  - Store a fixed, enumerated set of qualifiers
  - Each qualifier has a specified schema
  - Space-efficient
- **Map** Families
  - Map from a string to a value
  - Map key acts as a column qualifier
  - All map values must have the same schema
  - Example use cases: URLs, dynamic category names, separating account transactions

Locality Groups

- Analogous to HBase column families
- Assigns physical properties to data
  - TTL, MAXVERSIONS, INMEMORY, COMPRESSED WITH
  - Special values INFINITY and FOREVER
- Column families belong to a locality group
- Locality groups follow the same rules as HBase column families
  - **Don't have more than three or four of them**
  - http://hbase.apache.org/book/number.of.cfs.html

Layout Definition

- Tables can be defined with a JSON layout file or with DDL scripts
- Layout information is stored in HBase
  - Stored in a metadata table as Avro records
  - Kiji maps from column names to aliases in HBase
    - Saves on disk space

Example Layout DDL

```
CREATE TABLE users
  WITH DESCRIPTION 'Usernames and emails'
  ROW KEY FORMAT HASHED
  WITH LOCALITY GROUP default (
    MAXVERSIONS = INFINITY,
   COMPRESSED WITH GZIP,
   FAMILY info WITH DESCRIPTION 'basic information' (
     name "string" WITH DESCRIPTION 'the username',
     email "string”,
      address CLASS com.wibidata.MailingAddress
    )));
```

Row Key Formats

```
...ROW KEY FORMAT HASHED...
```

- Hash-Prefixed
  - Retains human-readable keys
- Hashed
  - Hashing improves key distribution across the HBase cluster
- Raw
  - Keys are managed by the application
- Composite
  - Hierarchical data

Composite Row Keys

- Key is composed of one or more pieces
- Specify which components are used in a hash
- Typically, prefix hash will be used
  - Retains the ability to filter over components
- string, int, long, and null components are supported
- Non-string components can help save disk space

Serialization Needs

- Support rich data types
  - Primitives, records, maps, lists
- Compact representation and fast performance
- Support schema evolution
  - Old code should be able to read new records
  - New code should be able to read old records
- Provide cross-language APIs

Serialization with AVRO

- Apache Avro provides flexible serialization
- All data is written along with its "writer schema"
- Reader schema may differ from the writer's

```
record LongList {
    long value;
    union { null, LongList } next;
}
```

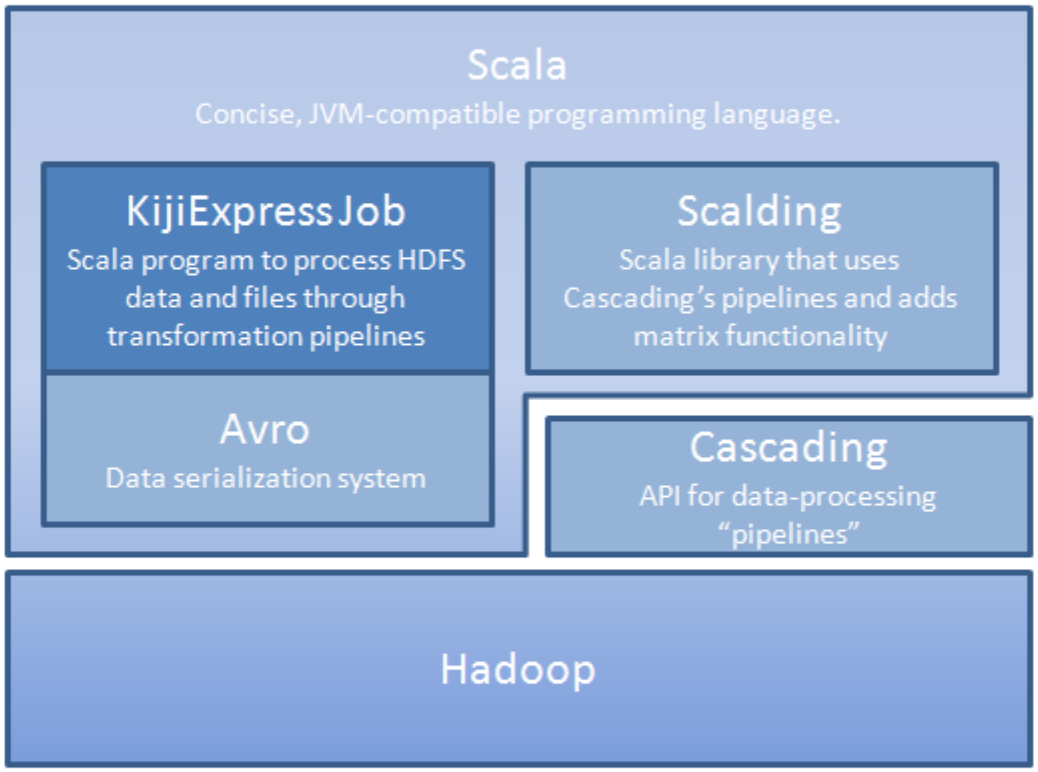
Hive

SQL-like queries with HiveQL

KijiExpress

Analyze data with Scalding and build ML models

KijiExpress - Introduction



*http://docs.kiji.org/tutorials/express-recommendation/0.6.0/express-language

KijiExpress - Overview

- Leverages Twitter's Scalding to allow developers/analysts to build complex Hadoop-based workflows using a Scala based DSL.
- "Compiles" down to a series of MapReduce jobs executed on the cluster.
- KijiExpress adds ability to read from/write to Kiji tables and specialized data structures/ operations to work with Kiji data.

Quick intro to Scalding...

- Open source project from Twitter (Apache License)
  - https://github.com/twitter/scalding
- Scala library wrapping Java Cascading library
  - Cascading allows you to construct high level Hadoop workflows in Java.
  - http://www.cascading.org
- API Reference:
  - https://github.com/twitter/scalding/wiki/Fields-based-API-Reference

Major Concepts - Data Movement

- Taps - Data input/output connectors
  - Source
    - Where data enters a data flow. Can come from any number of places. TSV, SequenceFiles, Kiji tables.
  - Sink
    - Where data is written to. This too can go to TSV, SequenceFiles, Kiji tables.
- Pipes
  - Data “conduits” through which Tuples flow. Perform computation on data and chain together to create data work flow.
- Tuples
  - Single collection of data elements. Conceptually equivalent to a

Major Concepts - Data Processing

- map/flatMap
  - **map** - Apply a function over all tuples and produce extra fields. (i.e. take the “name” field and produce “first_name” and “last_name” fields).
  - **flatMap** - Similar to map but will produce new “rows” with one or more additional fields. (i.e. take a single string and produce new “rows” each with a single “word” field.)
- filter
  - Apply a function over all tuples that will **restrict** the pipe to only those tuples where some condition is true. (i.e. include tuples where the phone number field is a valid number).
- groupBy/aggregation
  - Group on one or more columns and execute reduce operations on each group. (e.g. group by email address and count the number of messages received).
  - Invoking groupBy will add a MapReduce reduce phase to your flow. These can be costly!
- joins
  - Join one or more pipes together to produce a new pipe. Standard join semantics (inner, left, right) along with special joins to handle asymmetries in the size of the pipes. (e.g. join a large pipe of user data with a smaller pipe containing IP => location data)

KijiExpress Concepts

- KijiInput
- KijiSlice
- KijiOutput
- Working with Avro data

Example: Create a pipe to compute top 10 E-Mail senders

1. Start with an input such as KijiInput or Tsv
2. Map the input to fields. In this case the sender in the “from” field of the data
3. Group by sender and count the number of values in each group
4. Do a global descending sort
5. Output only the first ten values
6. Write to a sink, in this case a Tsv

Slightly more Complicated Example

What if we want to compute who sent the most emails to any other person?

Computing Term Frequency - Inverse Document Frequency

TF-IDF allows us to determine the importance of terms to a document within a corpus.

TFIDF

For a term i in document j:

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

$tf_{i,j}$ = number of occurrences of i in j
df_i = number of documents containing i
N = total number of documents

Computing Term Frequency - Inverse Document Frequency

This can be done in as little as four lines of code once you've prepared your input!

Compute a matrix representing Term Frequency (tf)

	word1	word2	...	wordN
email1	1	2		tf
email2	0	1		tf
...				tf
emailN	3	1		tf

Compute a matrix representing Document Frequency (df)

	word1	word2	...	wordN
email1	1	1		df
email2	0	1		df
...				df
emailN	1	1		df

word1	2
word2	3
...	

Compute the Inverse Document Frequency Vector

word1	2
word2	3
...	

	word1	word2	...	wordN
1	2	3		df

	word1	word2
1	log2(N/2)	log2(N/3)

Compute the Idf matrix

	word1	word2	...	wordN
email1	1	2		tf
email2	0	1		tf
...				tf
emailN	3	1		tf

Zip operation

	word1	word2	...	wordN
email1	(1,log2(N/2))	(2,log2(N/3))		(tf, idf)
email2	(0,log2(N/2))	(1,log2(N/3))		(tf, idf)
...				(tf, idf)
emailN	(3,log2(N/2))	(1,log2(N/3))		(tf, idf)

Compute the Tf-Idf matrix

	word1	word2	...	wordN
email1	1 * log2(N/2)	2 * log2(N/3)		(tf, idf)
email2	0 * log2(N/2)	1 * log2(N/3)		(tf, idf)
...				(tf, idf)
emailN	3 * log2(N/2)	1 * log2(N/3)		(tf, idf)

Zip operation ↓

	word1	word2	...	wordN
email1	log2(N/2)	2log(N/3)		(tf, idf)
email2	0	log2(N/3)		(tf, idf)
...				(tf, idf)
emailN	3log2(N/2)	log2(N/3)		(tf, idf)