Introduction to Real Time Big Data Analysis

What is WibiData?

Who are we?

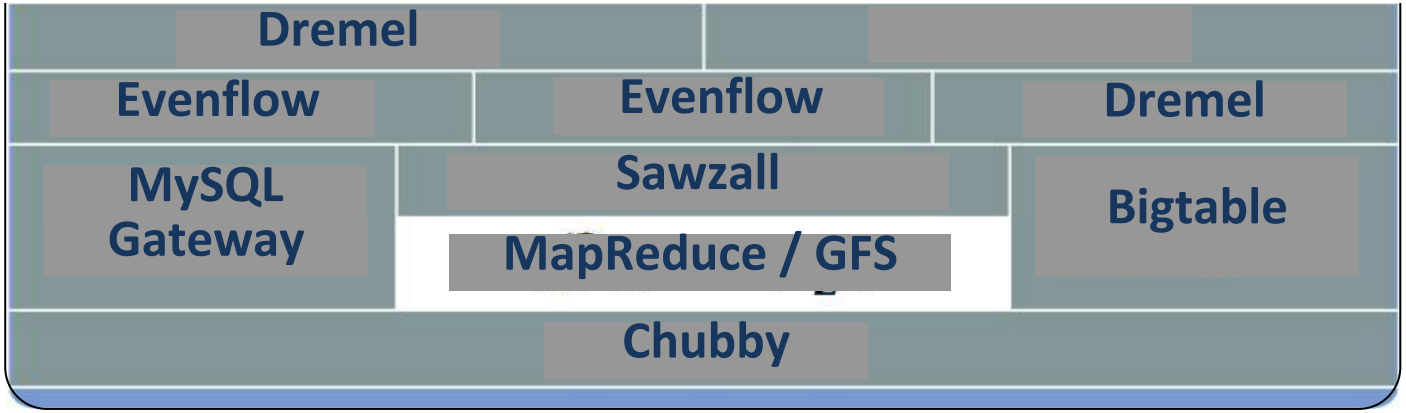
- Don Brown
- Adam Kunicki
- Amit Nithianandan
- Lee Sheng

Hadoop is a Data Operating System

- Originally created by Doug Cutting and Mike Cafarella in 2004
- Original use case was a simple web crawler
- It has grown into a general purpose distributed storage and computation framework

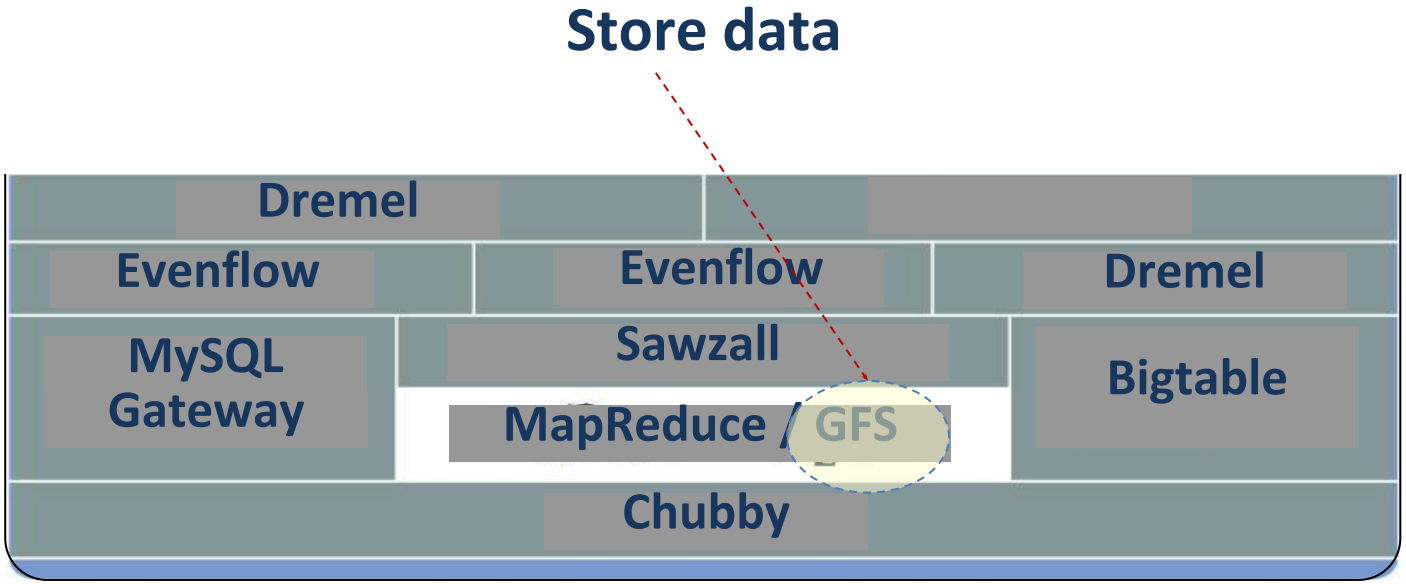
What did Google build?

Google



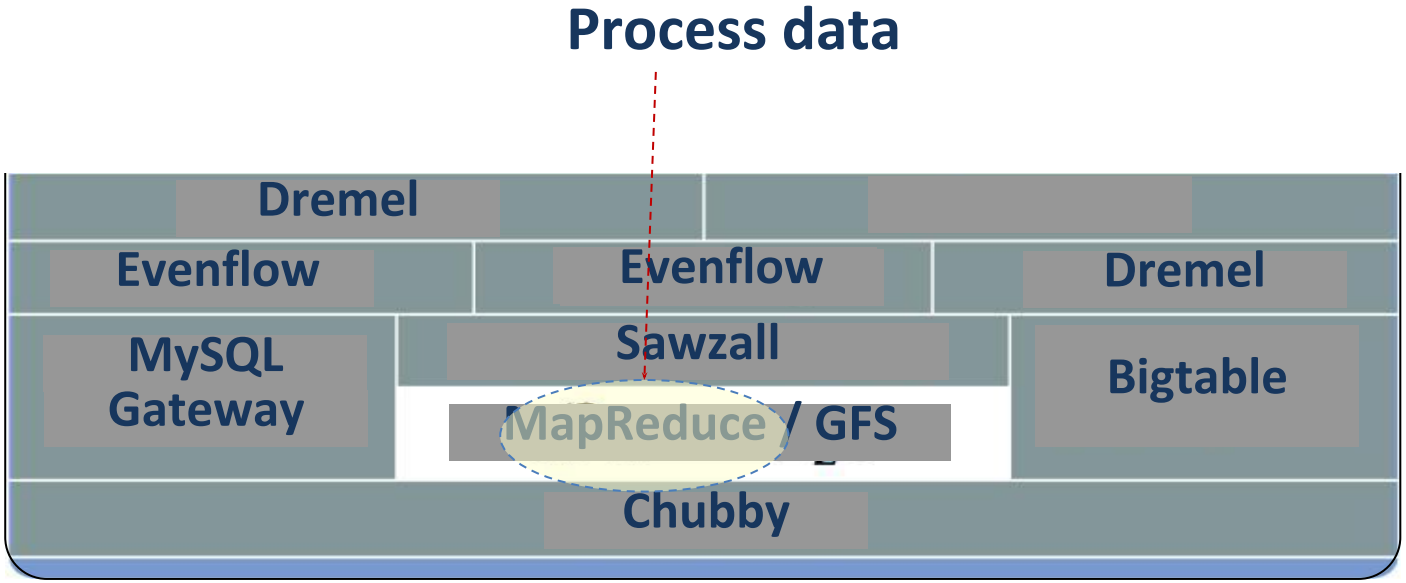
When they started to get big data, what did Google build?

Google



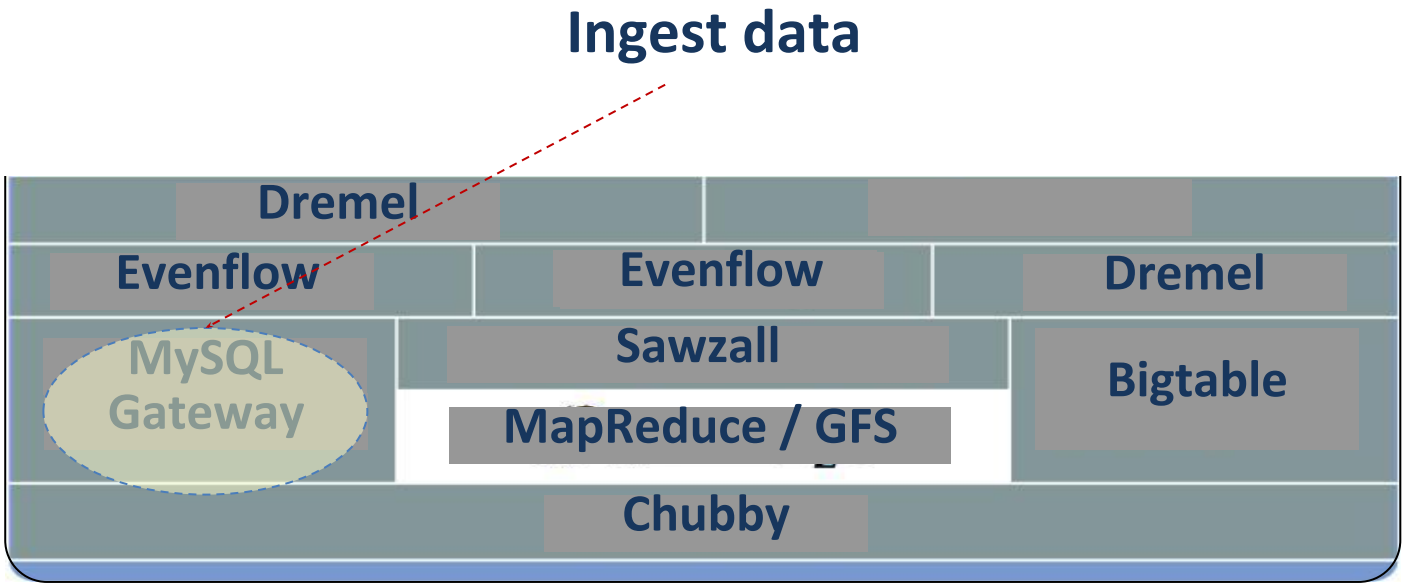
When they started to get big data, what did Google build?

Google



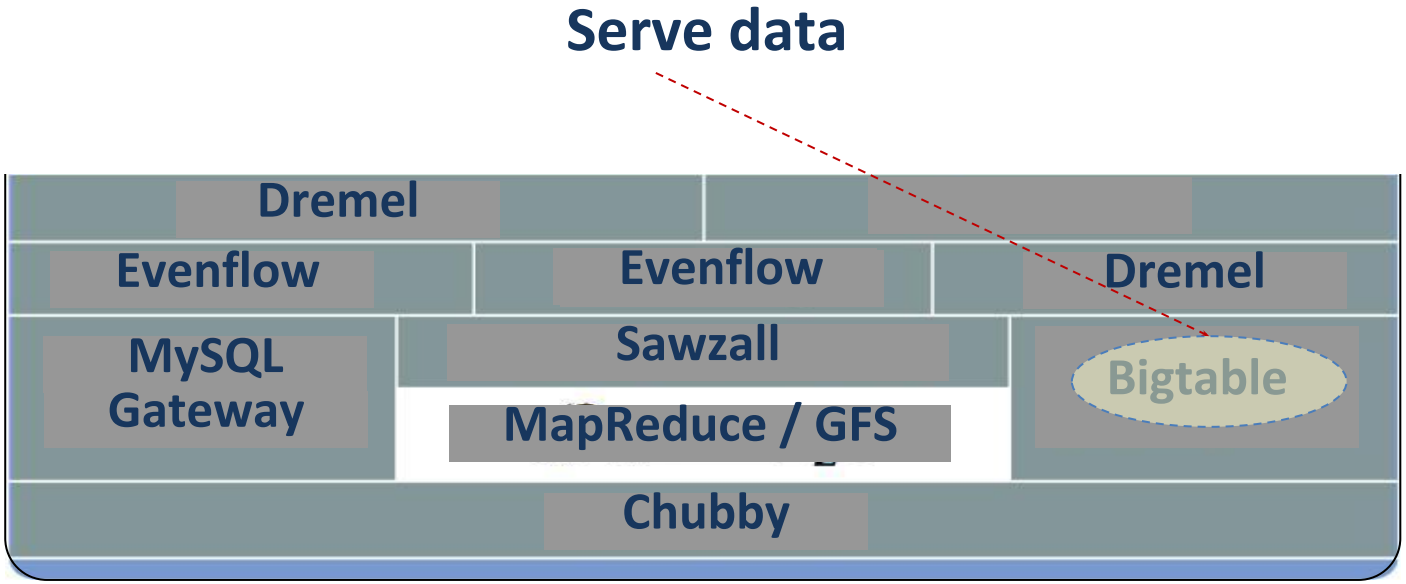
When they started to get big data, what did Google build?

Google



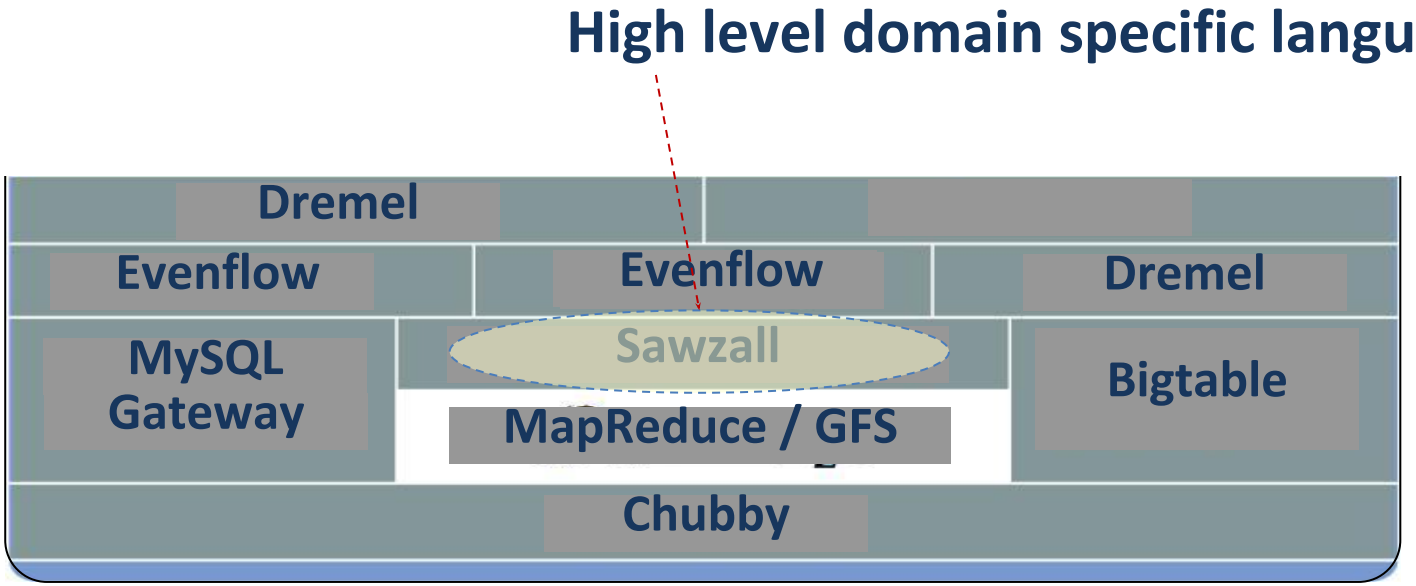
When they started to get big data, what did Google build?

Google



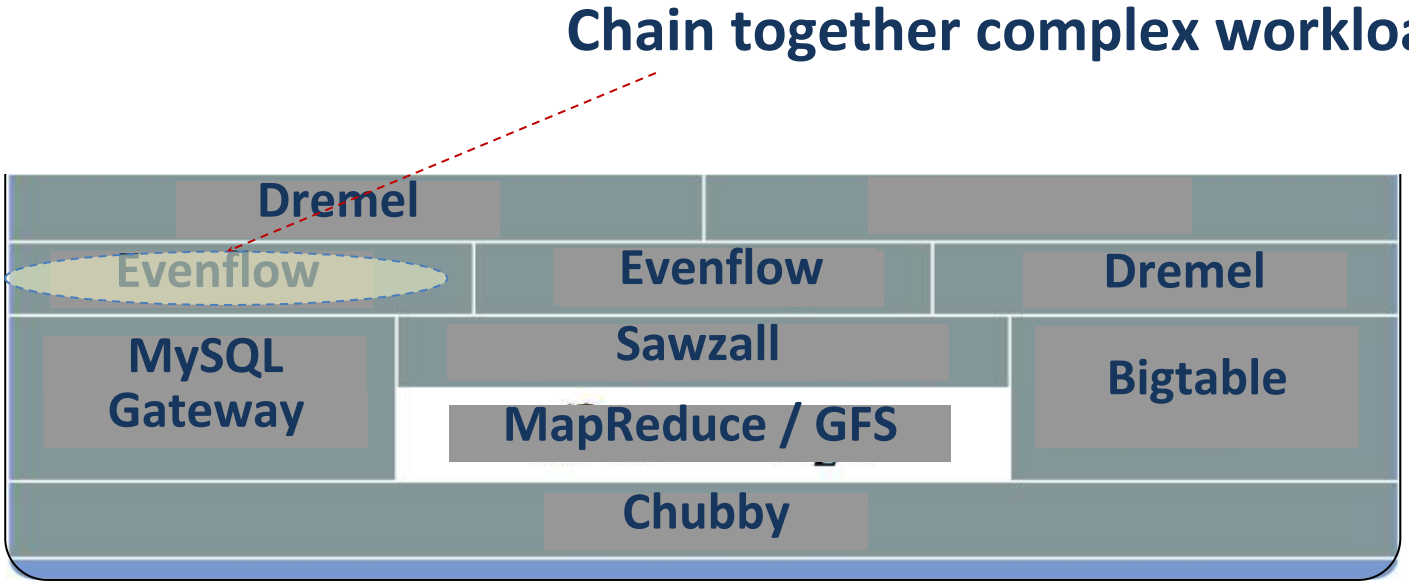
When they started to get big data, what did Google build?

Google



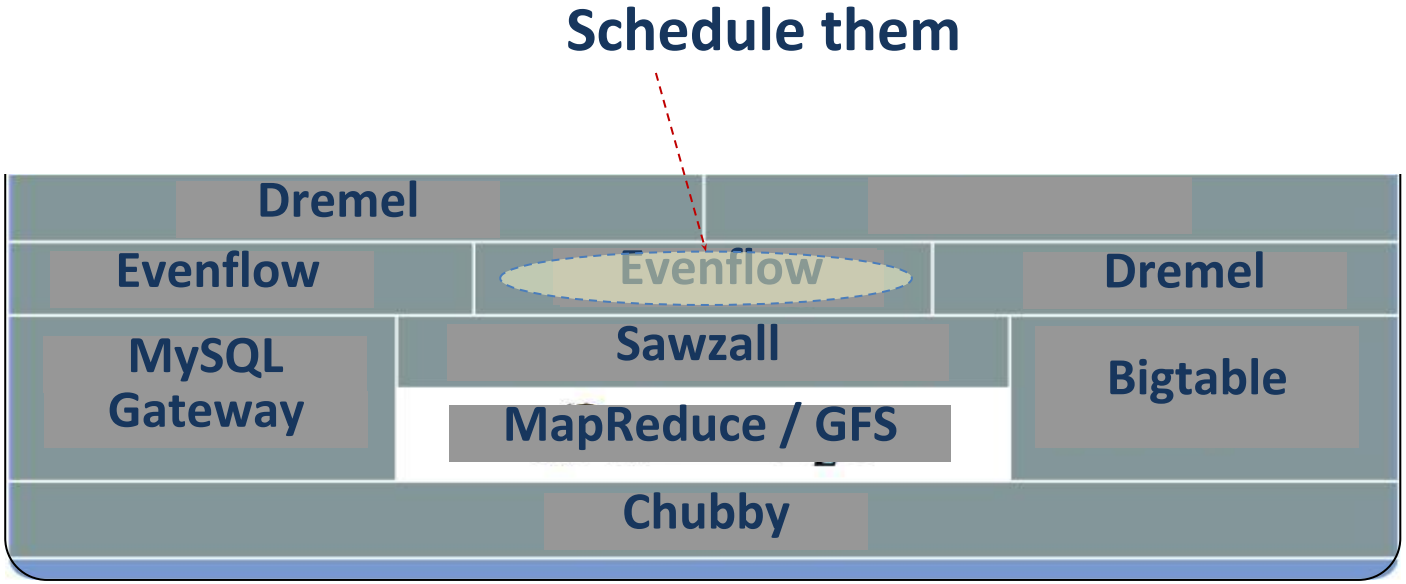
When they started to get big data, what did Google build?

Google



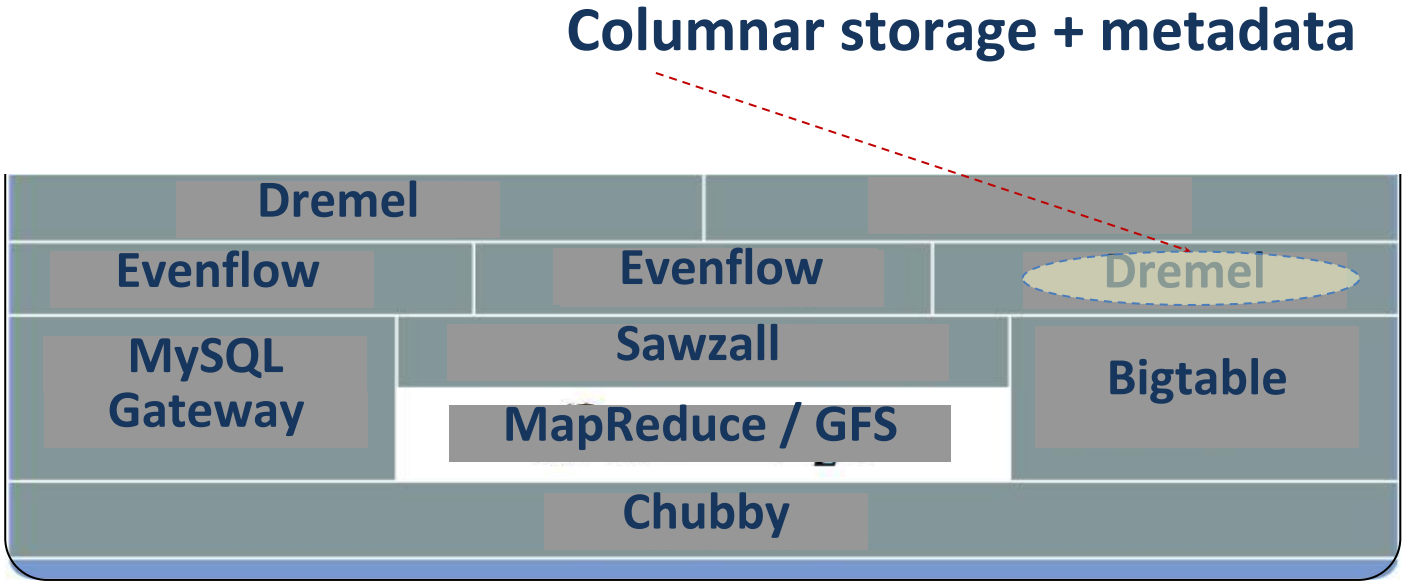
When they started to get big data, what did Google build?

Google



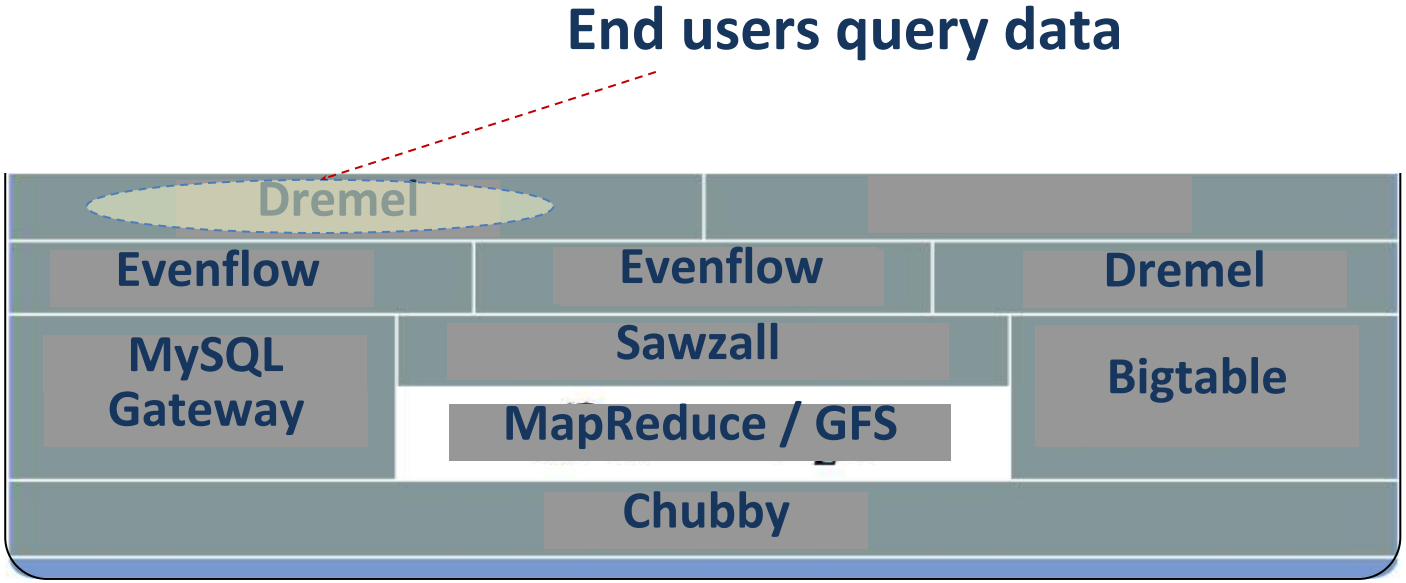
When they started to get big data, what did Google build?

Google



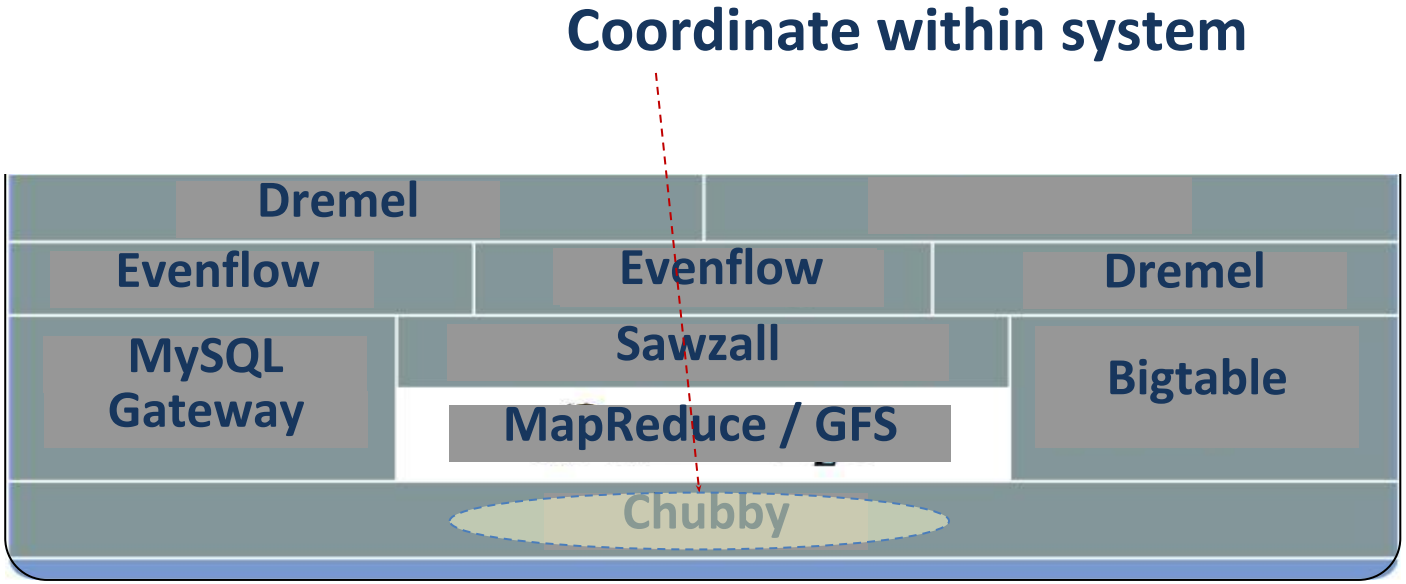
When they started to get big data, what did Google build?

Google



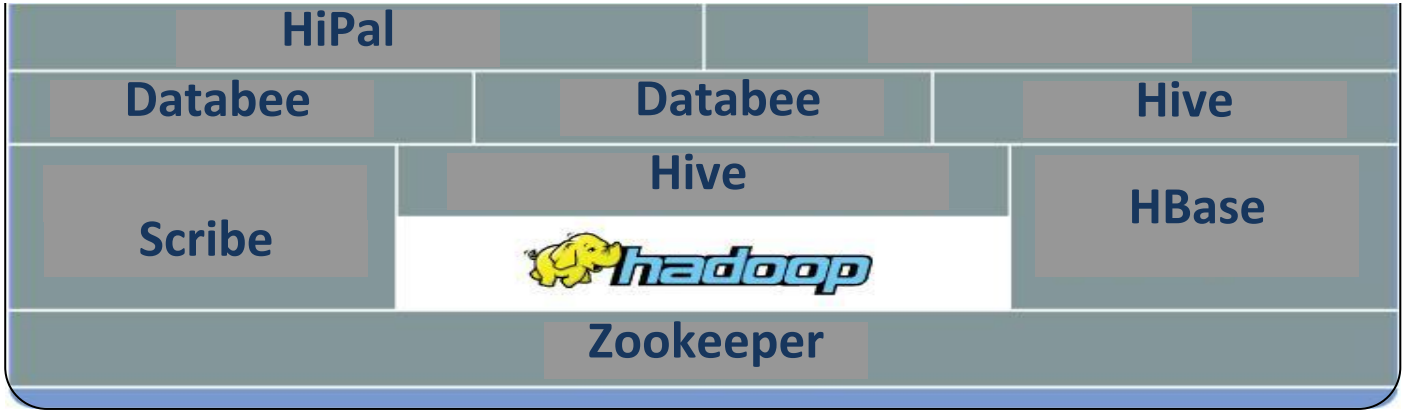
When they started to get big data, what did Google build?

Google



Later…

facebook



Rinse…


Oozie
Oozie
Hive
Pig & Hive
Data Highway
HBase
hadoop
Zookeeper

Repeat…


Azkaban
Azkaban
Sqoop
Kafka
Pig
hadoop
Voldemort
Zookeeper

What about low-latency access?

Several popular options

- Cassandra
- HBase
- Accumulo

- 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

APACHE HBASE

- 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

- 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 -> (Timestamp, Value)
- Ideal for entity-centric storage of data

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

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

HBase almost suits our needs

Pros:

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

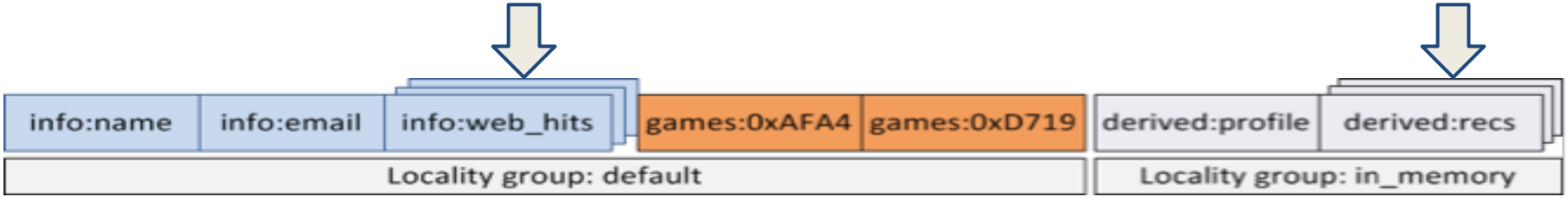
Cons:

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

Kiji Data Model

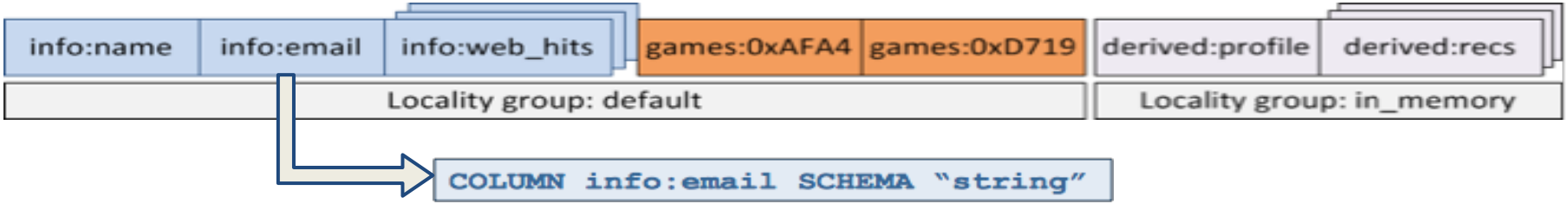
- Every row is identified by an **entity ID**
- Data is physically organized by **locality groups**
- Data is logically organized by **column families**
- Columns are identified with a **column qualifier**

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 

- 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

What is Hive?

- Apache Hive is a data warehousing system built on top of Hadoop. Built originally at Facebook and released to open sourced in 2009.
- The main purpose of Hive is for data analysis and ad hoc querying.
- Supports a variety of Hadoop file formats, as well as providing JDBC and ODBC drivers.
- Extensible both for input/outputs as well as for UDF(User defined functions).

Hive vs Impala

Hive

- Built on MapReduce
  - Slow spinup time as each query requires job creation
  - Reliability for the overall job as a result of MapReduce.
- Supports custom file formats
- Uses HCatalog Metastore to store table definitions.
- Runs queries written in HiveQL

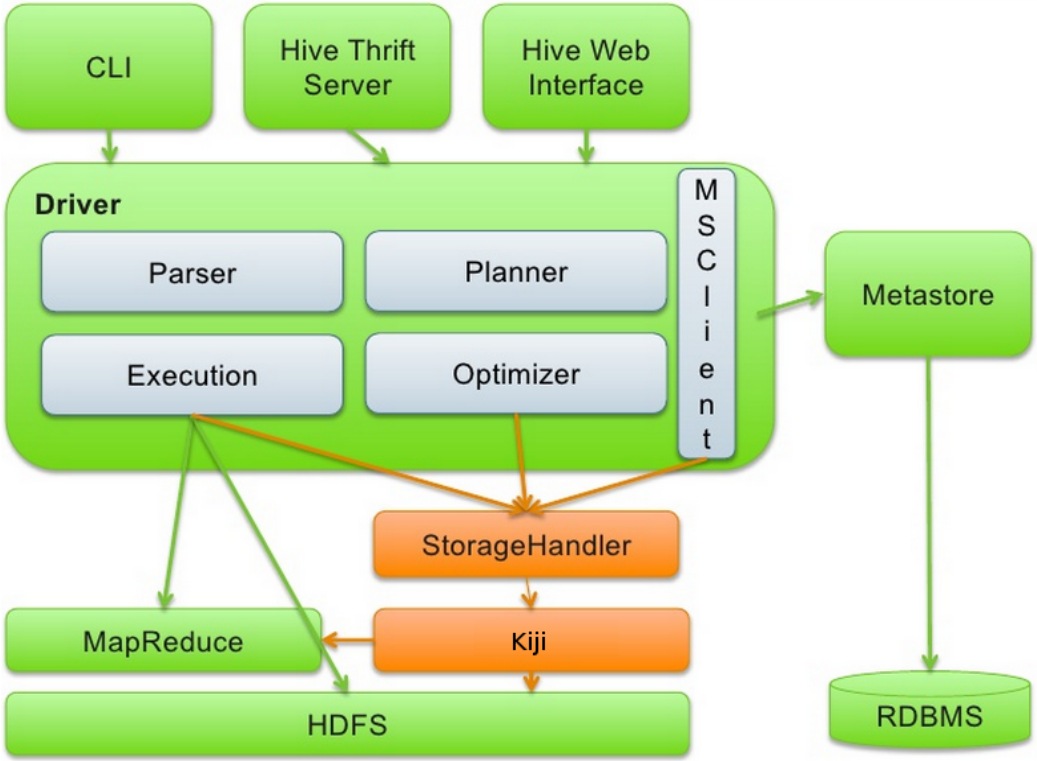
Impala

- Uses its own execution Engine
  - Faster performance for point queries.
  - No recovery if any execution node dies
- No pluggability
- Uses HCatalog Metastore to store table definitions.
- Runs queries written in HiveQL

Supported Storage Formats

- Files in HDFS
  - TEXTFILE
  - SEQUENCEFILE
  - RCFile
  - ORC (new RCFile for Stinger)
- JDBC
- Cassandra
- HBase, which is what leads us to supporting Hive in Kiji.

Hive architecture



Gaps between HiveQL and SQL

- Lack of in operator within statements.
  - No statements like: SELECT * FROM users WHERE name IN (‘John’, ‘Jane‘);
- Limited insert/update functionality.
- No self-referencing queries(reduces the clarity of SQL in queries).
- Only equijoins(no inequality joins)

Anatomy of a Hive Schema for Kiji

```
CREATE EXTERNAL TABLE users ( // EXTERNAL means that this particular table is not
                              // owned by Hive proper.
  user STRING, // 'user' refers to the name of the column in Hive.
               // STRING is the Hive type.
  track_plays ARRAY<STRUCT<ts: TIMESTAMP, value: STRING>>,
               // track_plays is an array of all the timestamped cells.
  next_song_rec STRUCT<ts: TIMESTAMP, value: STRING>
               // next_song_rec is the most recent timestamped cell.
)
STORED BY 'org.kiji.hive.KijiTableStorageHandler'
               // Use the Kiji classes for interfacing with data.
WITH SERDEPROPERTIES ( // KijiTableStorageHandler configuration.
  'kiji.columns' = ':entity_id,info:track_plays,info:next_song_rec[0]'
               // KijiRowExpressions that define(in order) what the Kiji source
               // of each Hive column is.
)
TBLPROPERTIES (
  'kiji.table.uri' = 'kiji://.env/kiji_music/users' // Kiji table URI
);
```

Enron Email Corpus Background

- 520,924 email messages from 150 Enron users, mostly senior management.
- Size: 423MB gzipped, 2.6GB uncompressed
- Original rowkey was to use the message id. But after realizing that each email appears multiple times, the rowkey [sender,timestamp] was used.
    - Made the assumption that any user can only send one email at a given timestamp. This resulted in 247,887 unique emails.
- Computed sentiment by extracting all the words and applying the sentiment score from the AFINN list. (Used a producer to compute this and write to a derived column).

Dataset downloaded from: https://www.cs.cmu.edu/~enron/

Hive Table Layout

```
CREATE EXTERNAL TABLE employee (
  mid ARRAY<STRUCT<ts: TIMESTAMP, value: STRING>>,
  fromStr STRUCT<ts: TIMESTAMP, value: STRING>,
  toStr ARRAY<STRUCT<ts: TIMESTAMP, value: STRING>>,
  subject ARRAY<STRUCT<ts: TIMESTAMP, value: STRING>>,
  body ARRAY<STRUCT<ts: TIMESTAMP, value: STRING>>
) STORED BY 'org.kiji.hive.KijiTableStorageHandler'
WITH SERDEPROPERTIES (
  'kiji.columns' = 'sent_messages:mid,sent_messages:from[0],
  sent_messages:to,sent_messages:subject,sent_messages:body '
) TBLPROPERTIES (
  'kiji.table.uri' = 'kiji://.env/enron_email/employee '
);
```

Who sends the most emails?

```
SELECT
  fromStr.value AS fromStr,
  size(mid) AS count
FROM employee
ORDER BY count DESC
LIMIT 10;
```

Who sends the most emails?

jeff.dasovich@enron.com 5405

kay.mann@enron.com 4758

sara.shackleton@enron.com 4393

tana.jones@enron.com 4055

pete.davis@enron.com 3910

vince.kaminski@enron.com 3759

chris.germany@enron.com 3570

matthew.lenhart@enron.com 2689

debra.perlingiere@enron.com 2412

gerald.nemec@enron.com 2222

Who emails whom the most?

```
SELECT
  fromStr,
  trim(splitToStr) AS toStr,
  count(1) AS count
FROM (
  SELECT
    fromStr.value AS fromStr,
    toLine.value AS toLine
  FROM employee
  LATERAL VIEW
    explode(toStr) tos AS toLine
) sq LATERAL VIEW
  explode(split(toLine,',')) tol AS splitToStr
GROUP BY fromStr, trim(splitToLine)
ORDER BY count DESC
LIMIT 10;
```

Who emails whom the most?

From	To	Count
pete.davis@enron.com	pete.davis@enron.com	3904
vince.kaminski@enron.com	vkaminski@aol.com	1039
michelle.nelson@enron.com	mike.maggi@enron.com	494
mike.maggi@enron.com	michelle.nelson@enron.com	452
kay.mann@enron.com	suzanne.adams@enron.com	449
j.kaminski@enron.com	vkaminski@aol.com	443
jeff.dasovich@enron.com	richard.shapiro@enron.com	443
tana.jones@enron.com	alan.aronowitz@enron.com	391

Who is Vincent Kaminski?

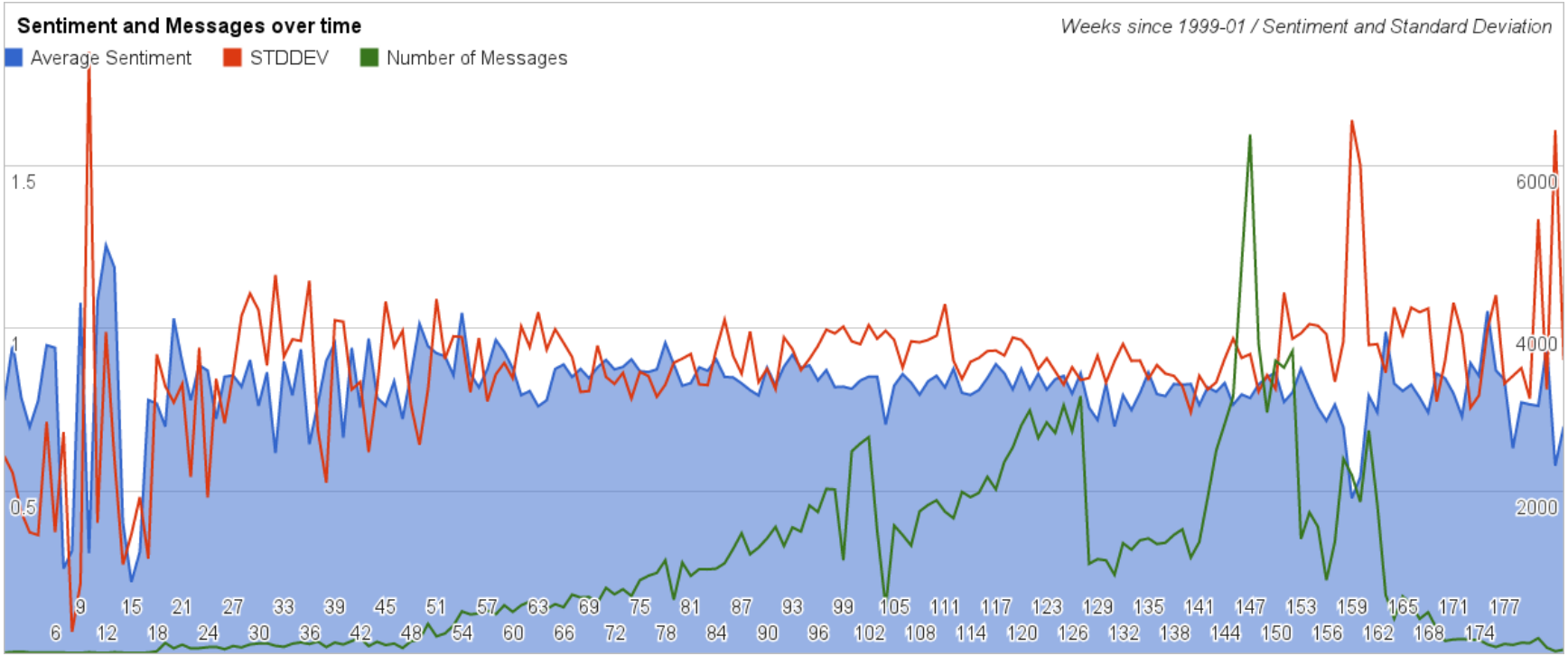
- Managing Director for Research at Enron.
- Repeatedly raised strong objections to accounting deals that Enron was doing.
- Choice lines like:
  - "so stupid that only Andrew Fastow could have come up with it,"
  - "heads the partnership wins, tails Enron loses."

http://www.washingtonpost.com/wp-dyn/content/article/2005/06/10/AR2005061000709.html

Sentiment at Enron over time

```
SELECT
  ((year(datelong.ts)-1999)*52+weekofyear(datelong.ts))
    AS weeknum,
  avg(sentiment.value) AS avgsentiment,
  stddev(sentiment.value) AS stddevsentiment,
  count(1) AS nummessages
FROM emails
WHERE regexp_replace(fromStr.value,".*@","")=="enron.com"
GROUP BY ((year(datelong.ts)-1999)*52+weekofyear(datelong.
ts));
```

Sentiment over time



Interesting looking dates:

- Peak of messages sent: Week 143(October 2001)
  - Enron had announced that they had to correct their financial statements for the last few years to resolve the accounting violations.
- Valley of average sentiment: Week 155(December 2001)
  - Buyout offer from competitor Dynegy failed, and Enron finally filed for bankruptcy.

Can we find negative words?

```
SELECT
  lword AS word,
  sum(sentiment) AS totalsentiment
FROM (
  SELECT
    lower(word) AS lword,
    sentiment.value AS sentiment
  FROM usersentemails
  LATERAL VIEW explode(sentences(body.value)[0]) wds AS word
  WHERE regexp_replace(fromStr.value,".*@","")=="enron.com"
) subquery
GROUP BY lword
ORDER BY totalsentiment ASC;
```

Negative words:

database	-21029.607485488057	1999	-4227.287998446729
dbcaps97data	-19852.214287519455	3dect	-4170.454455077648
error	-17593.24706810154	3dhou	-4167.271528244019
alias	-13795.813743025064	3djohn	-4161.744777917862
unknown	-11499.325352319516	forney	-3482.4385346332565
operation	-6188.143360715359	cannot	-2743.542536749039
initialize	-6108.375925928354	synchronizing	-2501.0873627215624
2a04	-6108.0	hourahead	-2286.577932715416
borland	-6091.149999856949	folder	-1952.093966498971
attempting	-5776.220677101519	failed	-1776.1406089728698
occurred	-5755.367470344063	insufficient	-1526.2367318486795
engine	-5730.411958107725	intervention	-1443.9731325190514
perform	-5361.119657802861	omnichair	-1336.0584192276
closed	-5211.192247746047	busypriority	-1323.977882862091
omni	-4293.765556707978	omniprincipal	-1321.485891699791

Negative words:

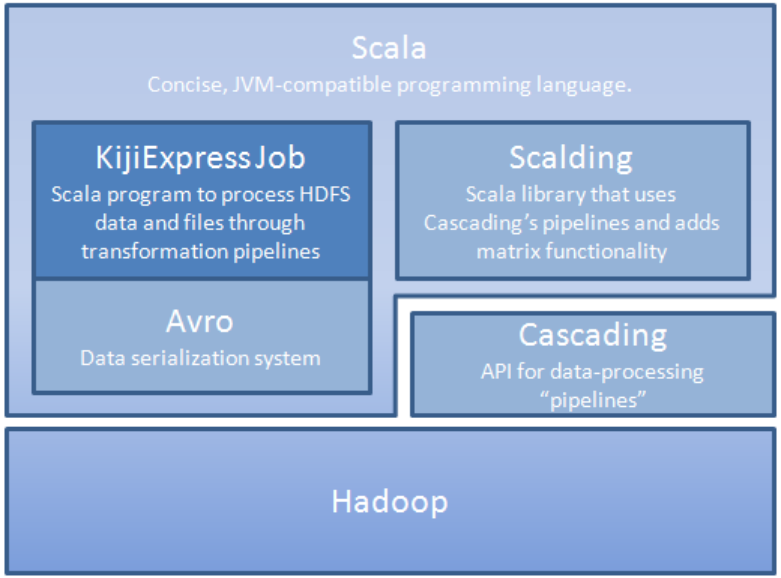
database	-21029.607485488057
dbcaps97data	-19852.214287519455
error	-17593.24706810154
alias	-13795.813743025064
unknown	-11499.325352319516
operation	-6188.143360715359
initialize	-6108.375925928354
2a04	-6108.0
borland	-6091.149999856949
attempting	-5776.220677101519
occurred	-5755.367470344063
engine	-5730.411958107725
perform	-5361.119657802861
closed	-5211.192247746047
omni	-4293.765556707978

1999	-4227.287998446729
3dect	-4170.454455077648
3dhou	-4167.271528244019
3djohn	-4161.744777917862
forney	**-3482.4385346332565**
cannot	-2743.542536749039
synchronizing	-2501.0873627215624
hourahead	-2286.577932715416
folder	-1952.093966498971
failed	-1776.1406089728698
insufficient	-1526.2367318486795
intervention	-1443.9731325190514
omnichair	-1336.0584192276
busypriority	-1323.977882862091
omniprincipal	-1321.485891699791

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

- **Tuple**
  - Single collection of data elements. Conceptually equivalent to a “row” of data that contains multiple fields.
- **Pipe**
  - Data conduits through which Tuples flow. Computation is performed on tuples flowing through pipes.

Major Concepts - Data Movement

- **Tap** - Data input/output connector
  - **Source**
    - Where data enters a flow. Can come from any number of places. TSV, SequenceFiles, Kiji tables.
  - **Sink**
    - Where tuples are written to at the end of the pipe. This too can go to TSV, SequenceFiles, Kiji tables.

Major Concepts - Data Processing

- map/flatMap/mapTo/flatMapTo
  - **map** - Applies a function over all tuples in a pipe 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.)
  - **mapTo/flatMapTo** variants do the exact same thing except discard any fields in the pipe that are not part of the output of the **mapTo** or **flatMapTo** call. Same as calling **project** after **map** or **flatMap** but more efficient.
- 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).

Major Concepts - Data Processing

- groupBy/aggregation
  - Group on one or more columns and execute reduce functions 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

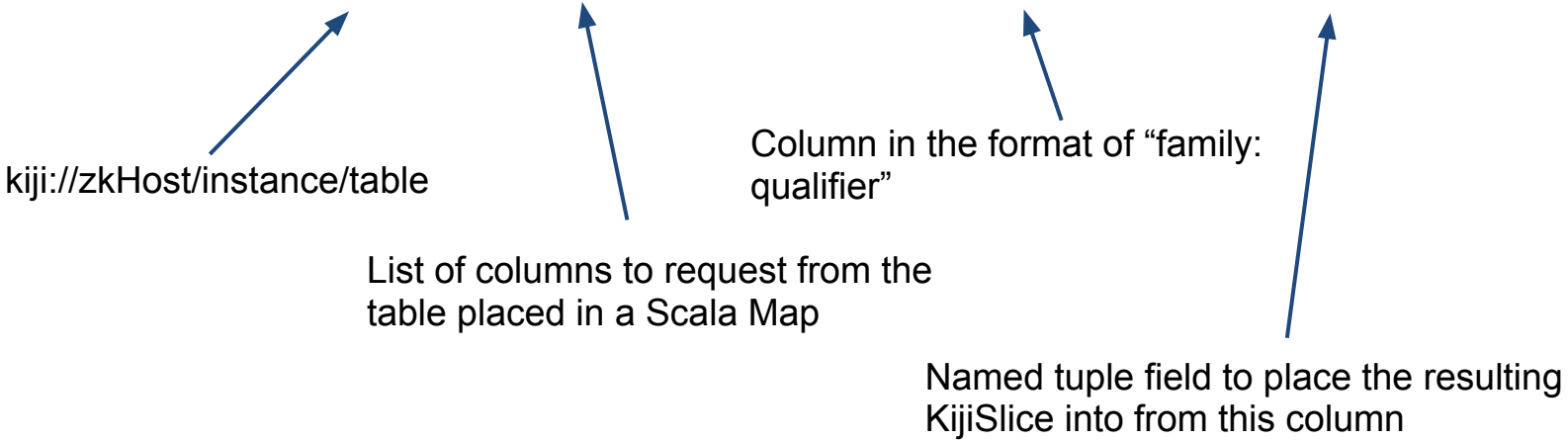
- **KijiInput -** Source for KijiTable
- **KijiSlice -** Group of cells from a KijiTable
- **KijiOutput -** Sink for KijiTable

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

1. Start with an input such as KijiInput
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

Understanding KijiInput

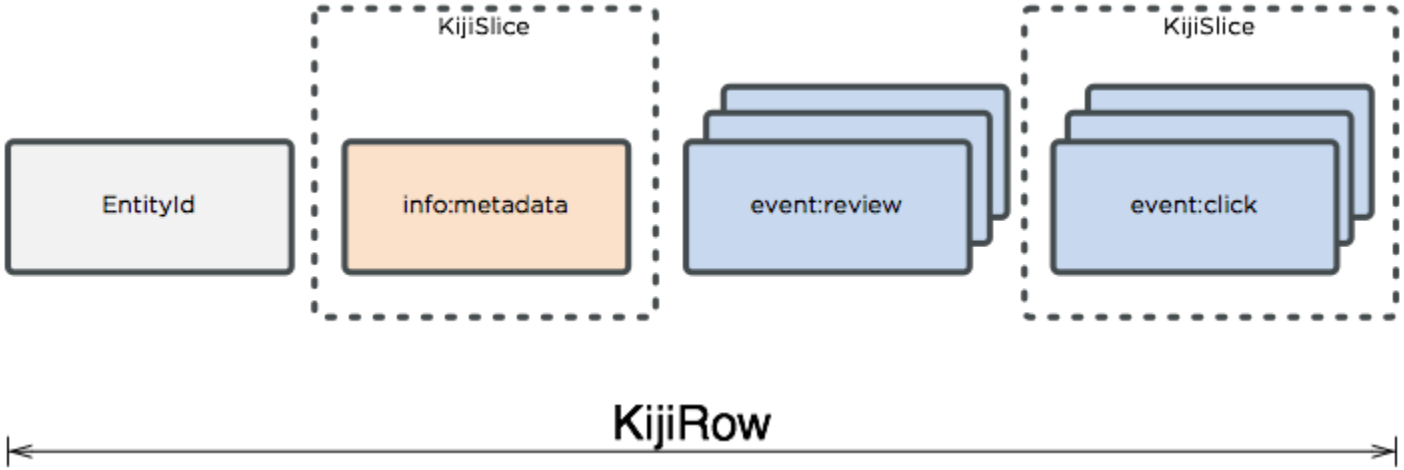
```
KijiInput(inputUri)(Map(Column("info:from") -> 'from))
```



What's in a KijiSlice anyway?

- Unlike flat text files, a column in HBase or a KijiTable is not one dimensional.
- A column contains cells which may have many versions (timestamp dimension).
- Also, each cell has an associated row key and family.
- This collection of cells and their metadata is represented in a KijiSlice.
- Typically we'll unpack cell values using the *flatMap* function.

KijiSlice



KijiInput(inputUri)(Map(Column(“info:metadata”) -> ‘md, Column(“event:click”, all) -> ‘clicks))

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 significance 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 Tf-Idf

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(email1,wordN)
email2	0	1		tf(email2,wordN)
...				tf(email3,wordN)
emailN	3	1		tf(email4,wordN)

Compute a matrix representing Document Frequency (df)

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

word1	2
word2	3
...	

Compute the Inverse Document Frequency Vector

word1	2
word2	3
...	

	word1	word2	...	wordN
1	2	3		df(wordN)

	word1	word2	...	wordN
1	log2(numDocs/2)	log2(numDocs/3)		log2(numDocs/df(wordN))

Compute the Idf matrix

	word1	word2	...	wordN
email1	1	2		tf(email1,wordN)
email2	0	1		tf(email2,wordN)
...				tf(email3,wordN)
emailN	3	1		tf(email4,wordN)

Zip operation

	word1	word2	...	wordN
email1	(1,log2(N/2))	(2,log2(N/3))		(tf(email1,wordN), idf1)
email2	(0,log2(N/2))	(1,log2(N/3))		(tf(email2,wordN), idf2)
...				(tf(email3,wordN), idf3)
emailN	(3,log2(N/2))	(1,log2(N/3))		(tf(email4,wordN), idf4)

Compute the Tf-Idf matrix

	word1	word2	...	wordN
email1	1 * log2(N/2)	2 * log2(N/3)		(tf(email1,wordN), idf1)
email2	0 * log2(N/2)	1 * log2(N/3)		(tf(email2,wordN), idf2)
...				(tf(email3,wordN), idf3)
emailN	3 * log2(N/2)	1 * log2(N/3)		(tf(email4,wordN), idf4)

Zip operation

	word1	word2	...	wordN
email1	log2(N/2)	2log(N/3)		(tf(email1,wordN), idf1)
email2	0	log2(N/3)		(tf(email2,wordN), idf2)
...				(tf(email3,wordN), idf3)
emailN	3log2(N/2)	log2(N/3)		(tf(email4,wordN), idf4)

Team Challenge

Can you implement a model that predicts when the next correspondence between two entities will be?