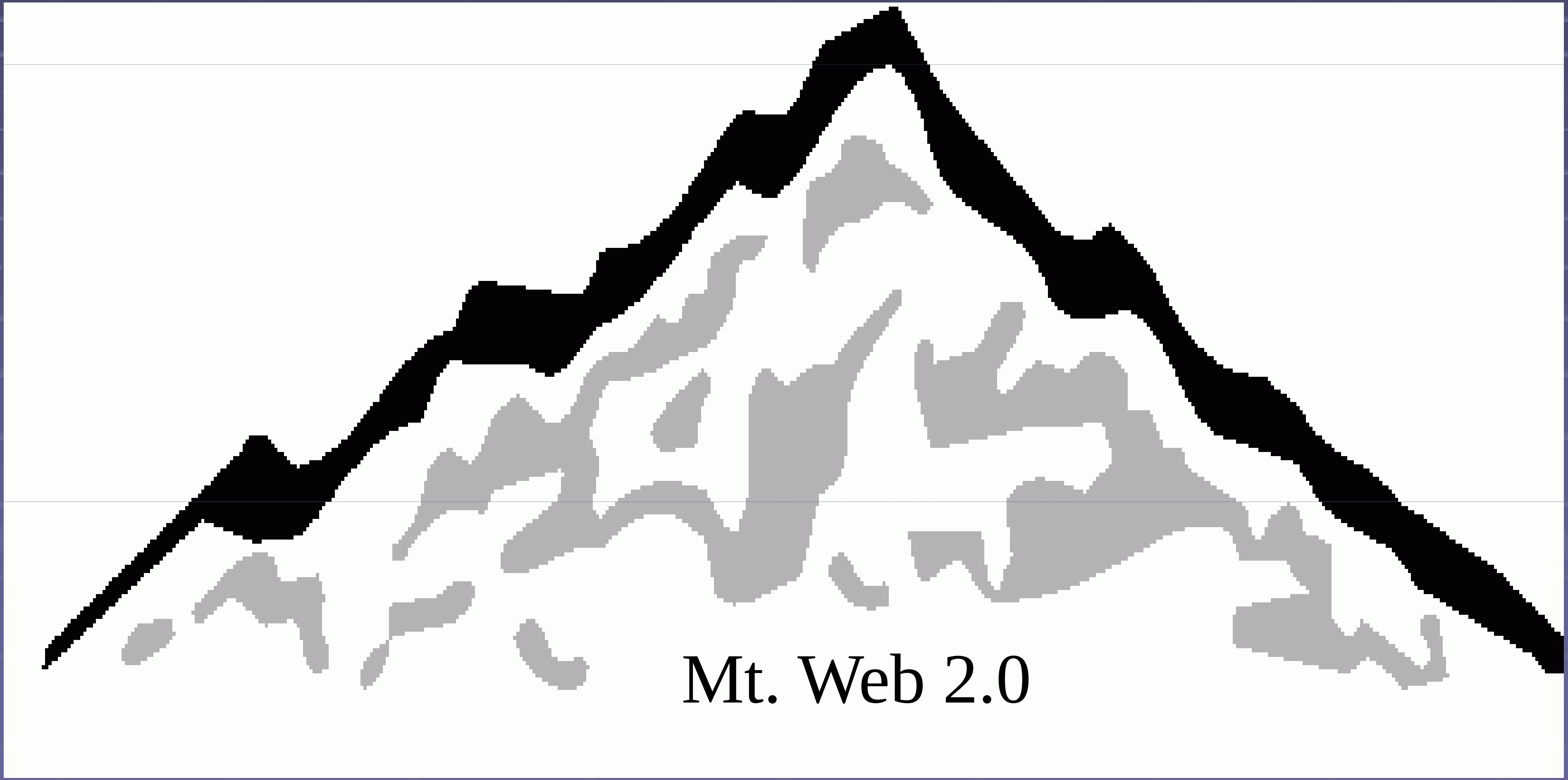
Hypertable

Doug Judd

Zvents, Inc.

Background

Web 2.0 = Data Explosion



Traditional Tools Don’t Scale Well

- Designed for a single machine
- Typical scaling solutions
  - ad-hoc
  - manual/static resource allocation

The Google Stack

- Google File System (GFS)
- Map-reduce
- Bigtable

What is Hypertable?

- Massively scalable database, modelled after Bigtable
- Open Source (GPL)
- Supports massive tables
- Data is sorted (indexed) by a single primary key (row key)

What is Hypertable not?

- A relational database (no joins)
- A transaction system

Hypertable Improvements Over Traditional RDBMS

- Scalable
- High **random** insert, update, and delete rate

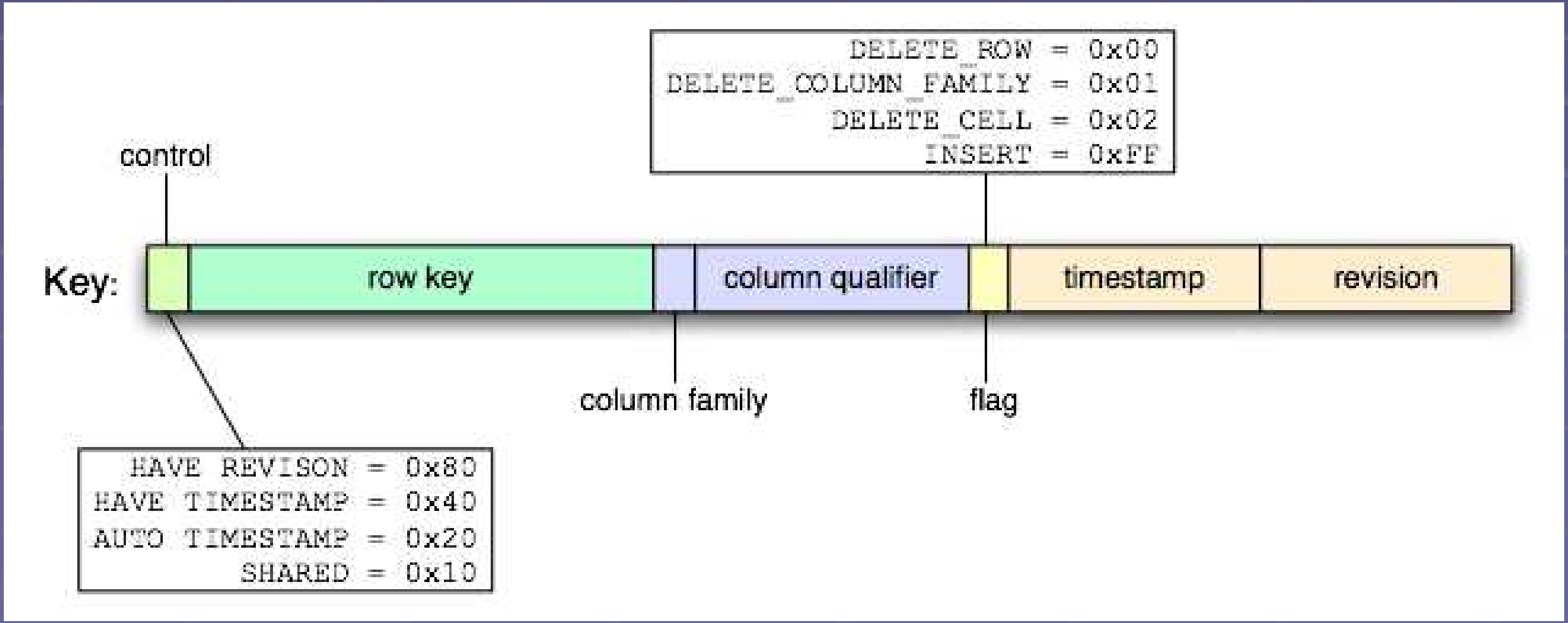
Architectural Overview

Data Model

- Sparse, two-dimensional table with cell versions
- Cells are identified by a 4-part key
  - Row (string)
  - Column Family (byte)
  - Column Qualifier (string)
  - Timestamp (long integer)

Anatomy of a Key

- Column Family is represented with 1 byte
- Timestamp and revision are stored big-endian ones-compliment
- Simple byte-wise comparison



hypertable.org

Table: Visual Representation

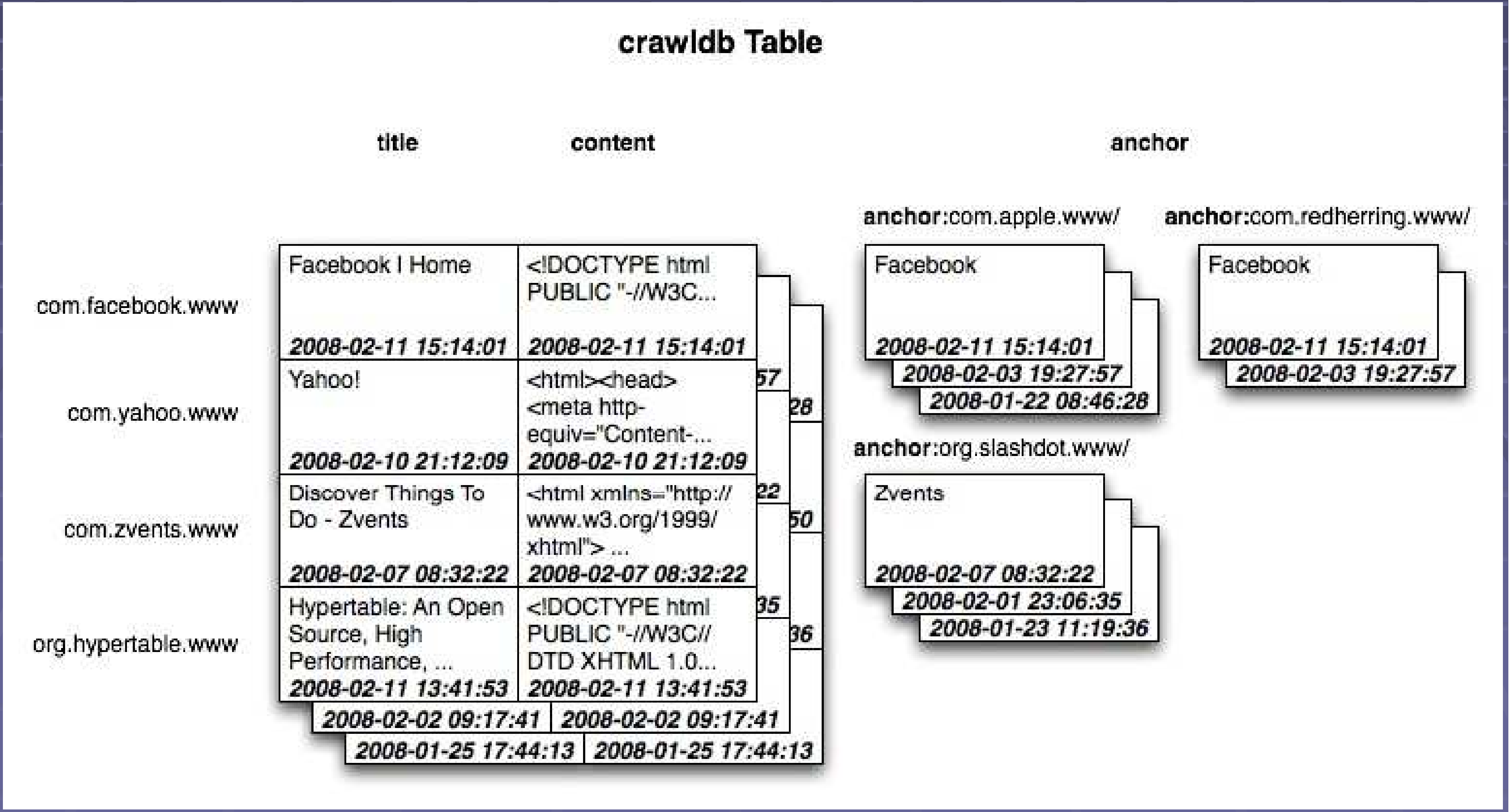
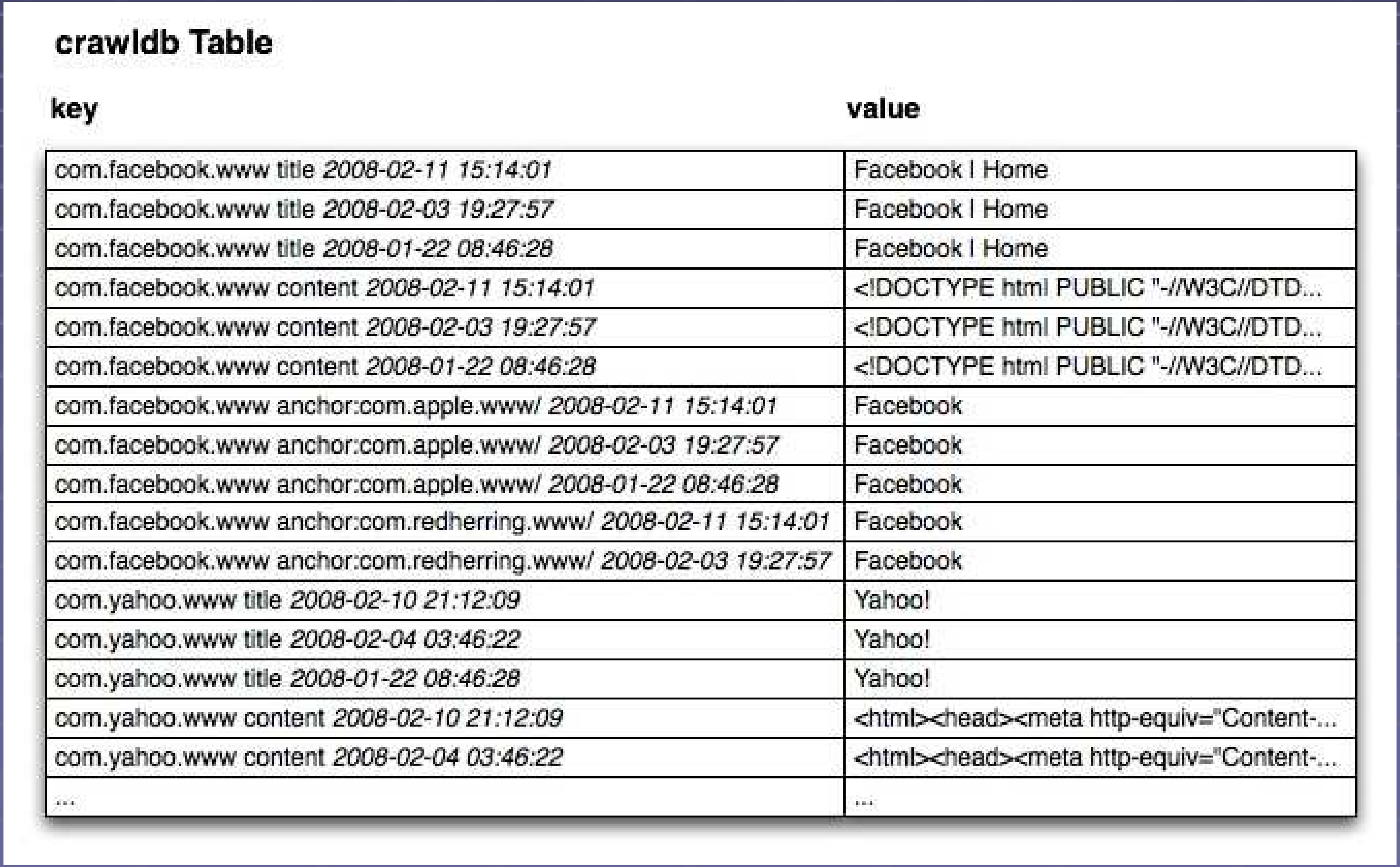


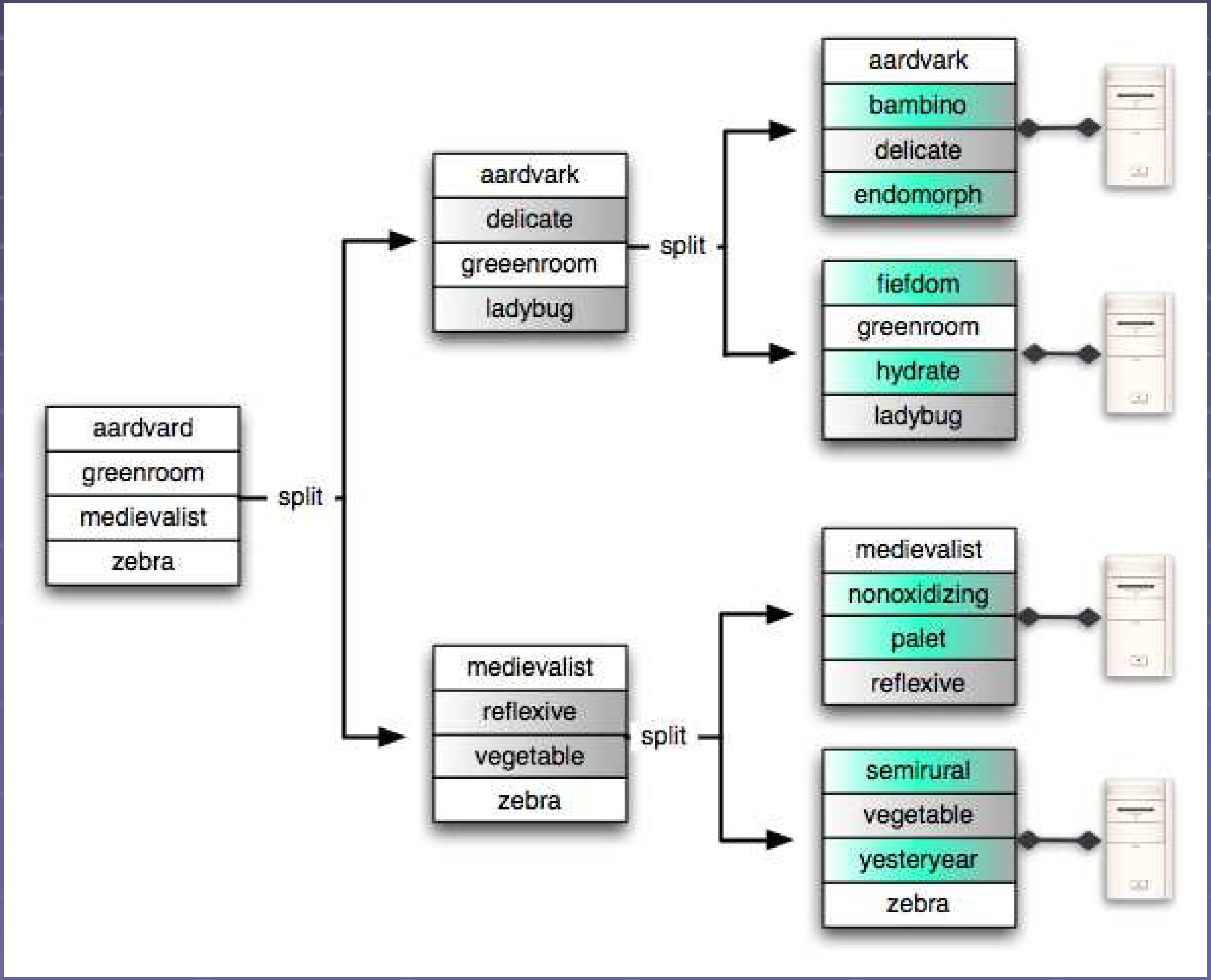
Table: Actual Representation



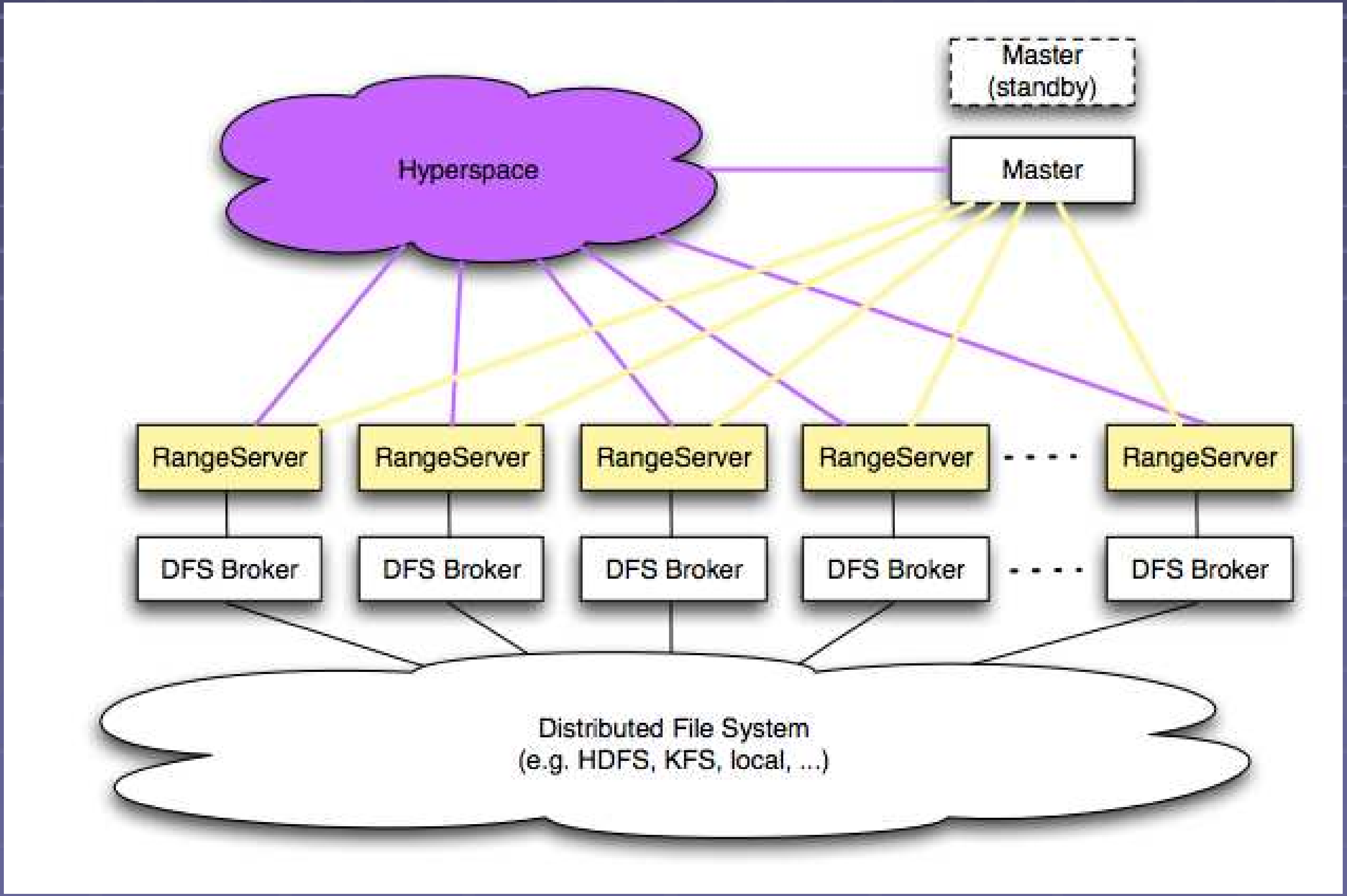
crawldb Table

key	value
com.facebook.www title *2008-02-11 15:14:01*	Facebook I Home
com.facebook.www title *2008-02-03 19:27:57*	Facebook I Home
com.facebook.www title *2008-01-22 08:46:28*	Facebook I Home
com.facebook.www content *2008-02-11 15:14:01*	<!DOCTYPE html PUBLIC "-//W3C//DTD...
com.facebook.www content *2008-02-03 19:27:57*	<!DOCTYPE html PUBLIC "-//W3C//DTD...
com.facebook.www content *2008-01-22 08:46:28*	<!DOCTYPE html PUBLIC "-//W3C//DTD...
com.facebook.www anchor:com.apple.www/ *2008-02-11 15:14:01*	Facebook
com.facebook.www anchor:com.apple.www/ *2008-02-03 19:27:57*	Facebook
com.facebook.www anchor:com.apple.www/ *2008-01-22 08:46:28*	Facebook
com.facebook.www anchor:com.redherring.www/ *2008-02-11 15:14:01*	Facebook
com.facebook.www anchor:com.redherring.www/ *2008-02-03 19:27:57*	Facebook
com.yahoo.www title *2008-02-10 21:12:09*	Yahoo!
com.yahoo.www title *2008-02-04 03:46:22*	Yahoo!
com.yahoo.www title *2008-01-22 08:46:28*	Yahoo!
com.yahoo.www content *2008-02-10 21:12:09*	<html><head><meta http-equiv="Content-...
com.yahoo.www content *2008-02-04 03:46:22*	<html><head><meta http-equiv="Content-...
...	...

Table: Growth Process

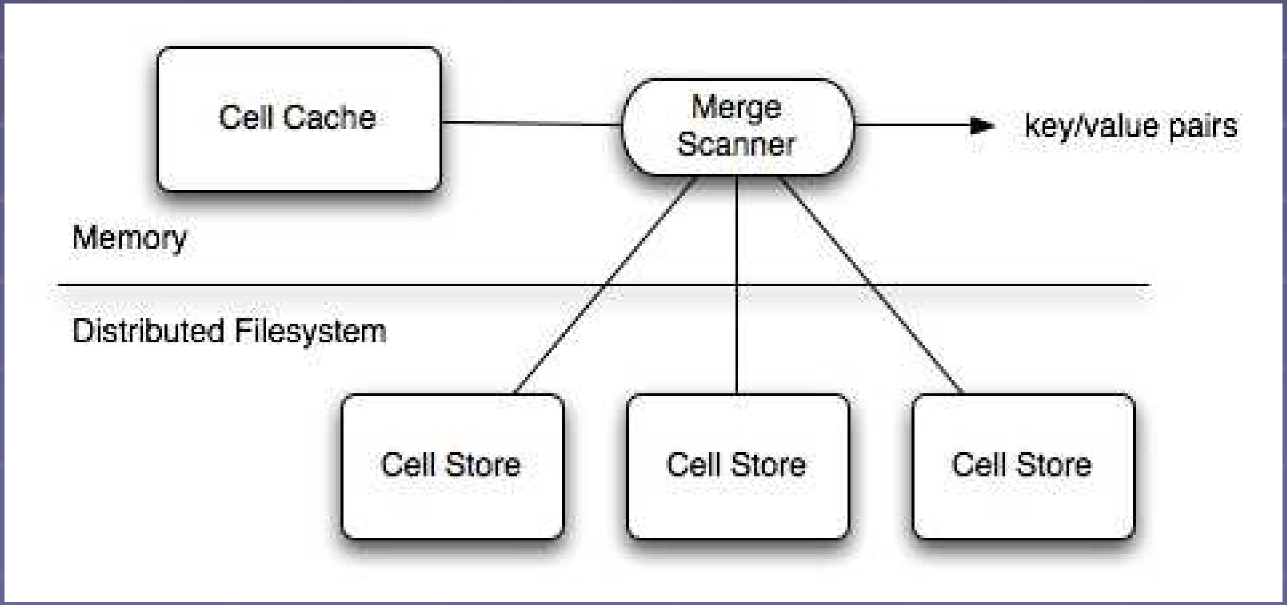


System Overview



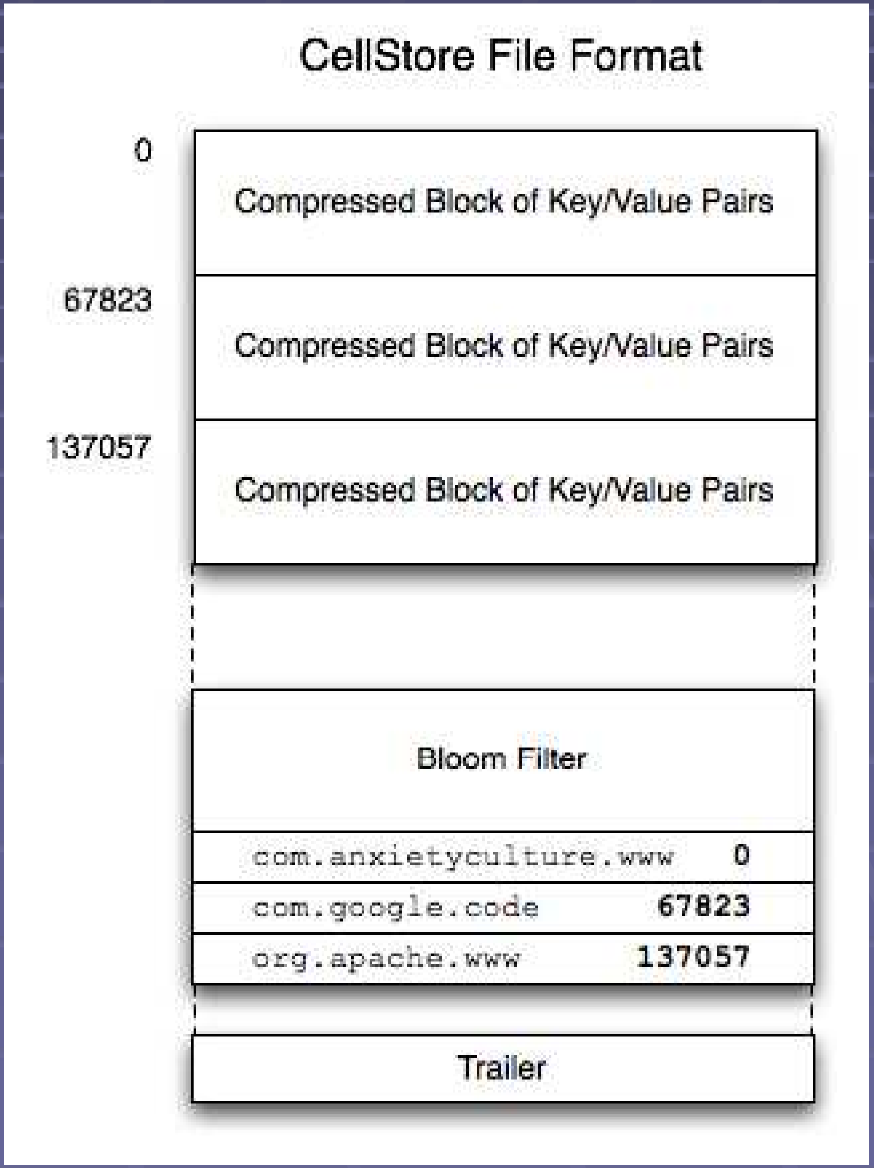
Range Server

- Manages ranges of table data
- Caches updates in memory (CellCache)
- Periodically spills (compacts) cached updates to disk (CellStore)



Range Server: CellStore

- Sequence of 65K blocks of compressed key/value pairs



Range Server: Write Ahead Commit Log

- Persists all modifications (inserts and deletes)
- Written into underlying DFS

Master

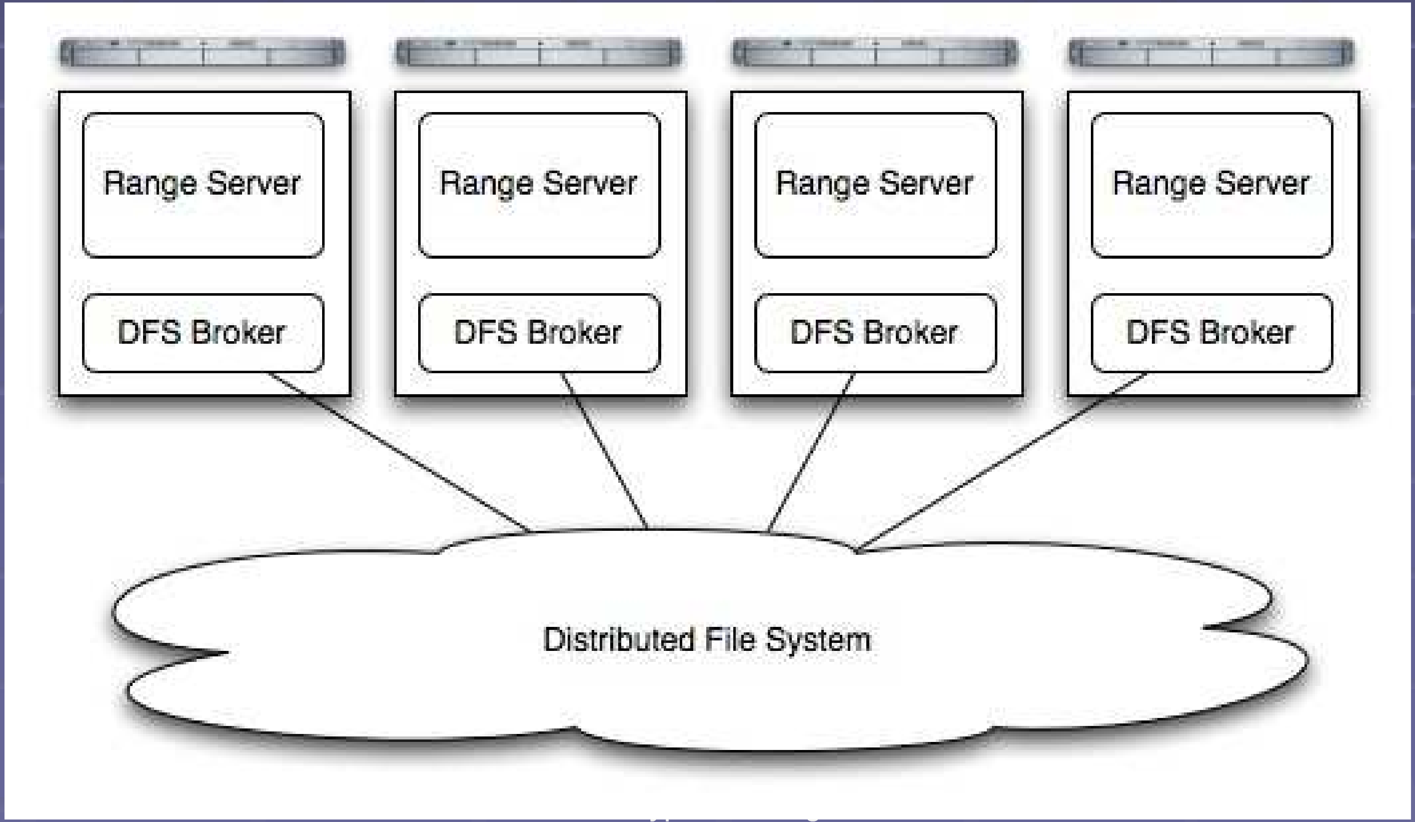
- Single Master (hot standbys)
- Directs meta operations
  - CREATE TABLE
  - DROP TABLE
  - ALTER TABLE
- Handles recovery of RangeServer
- Manages RangeServer Load Balancing
- Client data does **not** move through Master

Hyperspace

- Chubby equivalent
  - Distributed Lock Manager
  - Filesystem for storing small amounts of metadata
  - Highly available
- "Root of distributed data structures"

Filesystem Broker Architecture

- Hypertable can run on top of any distributed filesystem (e.g. KFS, HDFS, etc.)



Client API

```
class Client {

  void create_table(const String &name,
                    const String &schema);

  Table *open_table(const String &name);

  String get_schema(const String &name);

  void get_tables(vector<String> &tables);

  void drop_table(const String &name,
                  bool if_exists);
};
```

Client API (cont.)

```
class Table {
  TableMutator *create_mutator();
  TableScanner *create_scanner(ScanSpec &scan_spec);
};
class TableMutator {
  void set(KeySpec &key, const void *value, int value_len);
  void set_delete(KeySpec &key);
  void flush();
};
class TableScanner {
 bool next(CellT &cell);
};
```

Client API (cont.)

```
class ScanSpecBuilder {
  void set_row_limit(int n);
  void set_max_versions(int n);
  void add_column(const String &name);
  void add_row(const String &row_key);
  void add_row_interval(const String &start, bool sinc,
                        const String &end, bool einc);
  void add_cell(const String &row, const String &column);
  void add_cell_interval(…)
  void set_time_interval(int64_t start, int64_t end);
  void clear();
  ScanSpec &get();
}
```

Language Bindings

- Currently C++ only
- Thrift Broker will provide bindings for:
  - Java
  - Python
  - PHP
  - Ruby
  - Erlang
  - Perl
  - Others (Haskell, C#, Cocoa, Smalltalk, and Ocaml)

Optimizations

Compression

- Cell Stores store compressed blocks of key/value pairs
- Commit Log stores compressed blocks of updates
- Supported Compression Schemes
  - zlib (--best and --fast)
  - lzo
  - quicklz
  - bmz
  - none

Caching

- Block Cache
  - Caches CellStore blocks
  - Blocks are cached uncompressed
- Query Cache
  - Caches query results
  - TBD

Bloom Filter

- Negative Cache
- Probabilistic data structure
- Indicates if key is **not** present

Concurrency

- MVCC
- Bigtable uses copy-on-write

Access Groups

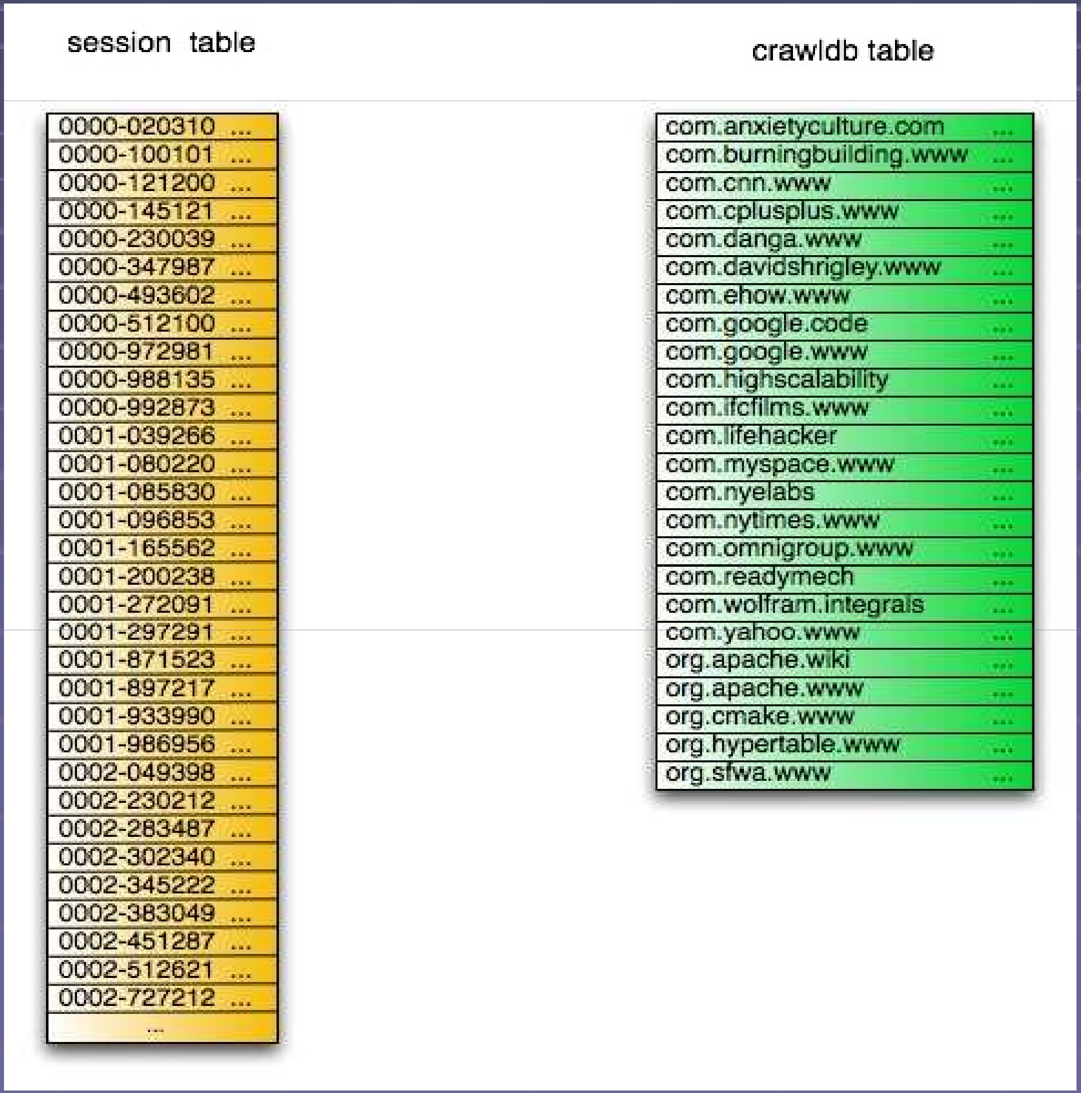
- Provides control of physical data layout -- hybrid row/column oriented
- Improves performance by minimizing I/O

```
CREATE TABLE crawldb {
  Title MAX_VERSIONS=3,
  Content MAX_VERSIONS=3,
  PageRank MAX_VERSIONS=10,
  ClickRank MAX_VERSIONS=10,
  ACCESS GROUP default (Title, Content),
  ACCESS GROUP ranking (PageRank, ClickRank)
};
```

Keys To Performance

- C++
- Asynchronous communication

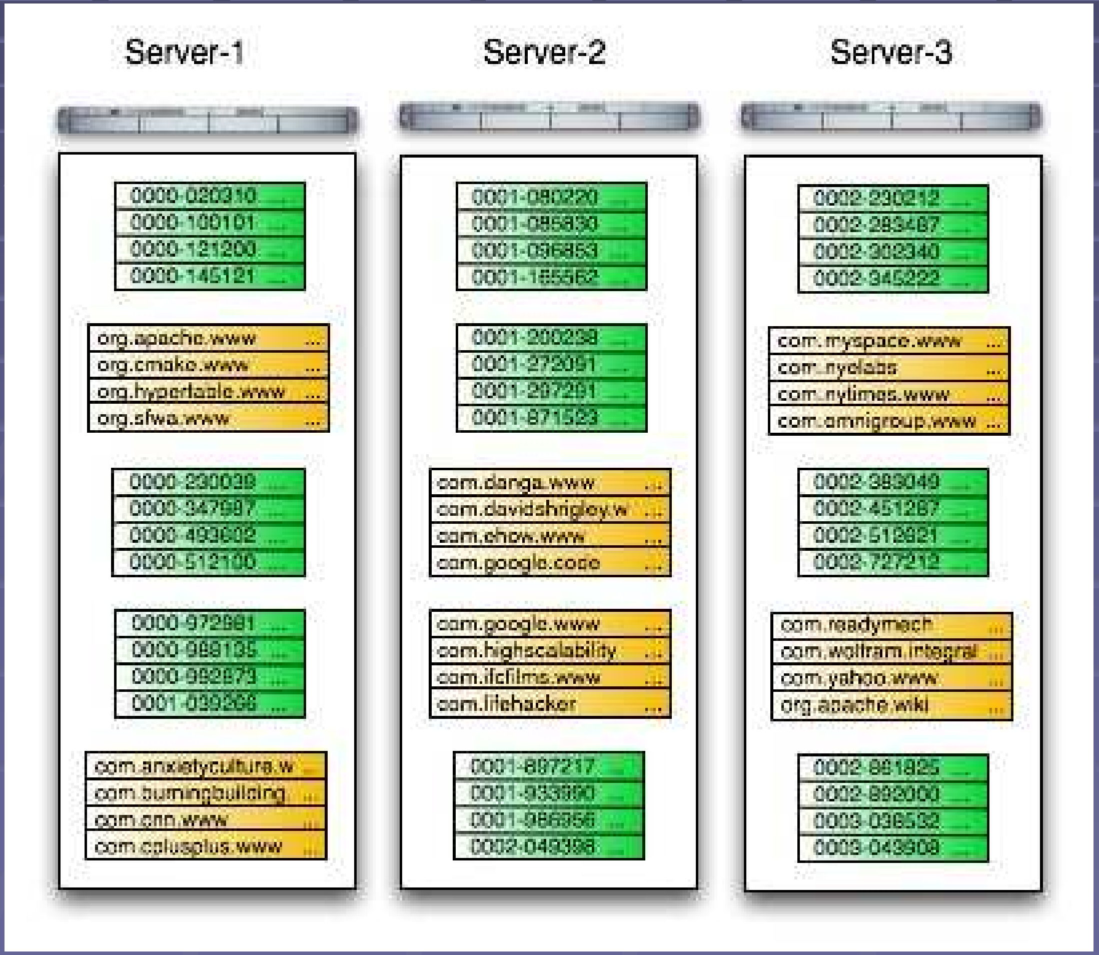
Scaling (part I)



session table	
0000-020310	...
0000-100101	...
0000-121200	...
0000-145121	...
0000-230039	...
0000-347987	...
0000-493602	...
0000-512100	...
0000-972981	...
0000-988135	...
0000-992873	...
0001-039266	...
0001-080220	...
0001-085830	...
0001-096853	...
0001-165562	...
0001-200238	...
0001-272091	...
0001-297291	...
0001-871523	...
0001-897217	...
0001-933990	...
0001-986956	...
0002-049398	...
0002-230212	...
0002-283487	...
0002-302340	...
0002-345222	...
0002-383049	...
0002-451287	...
0002-512621	...
0002-727212	...
...	

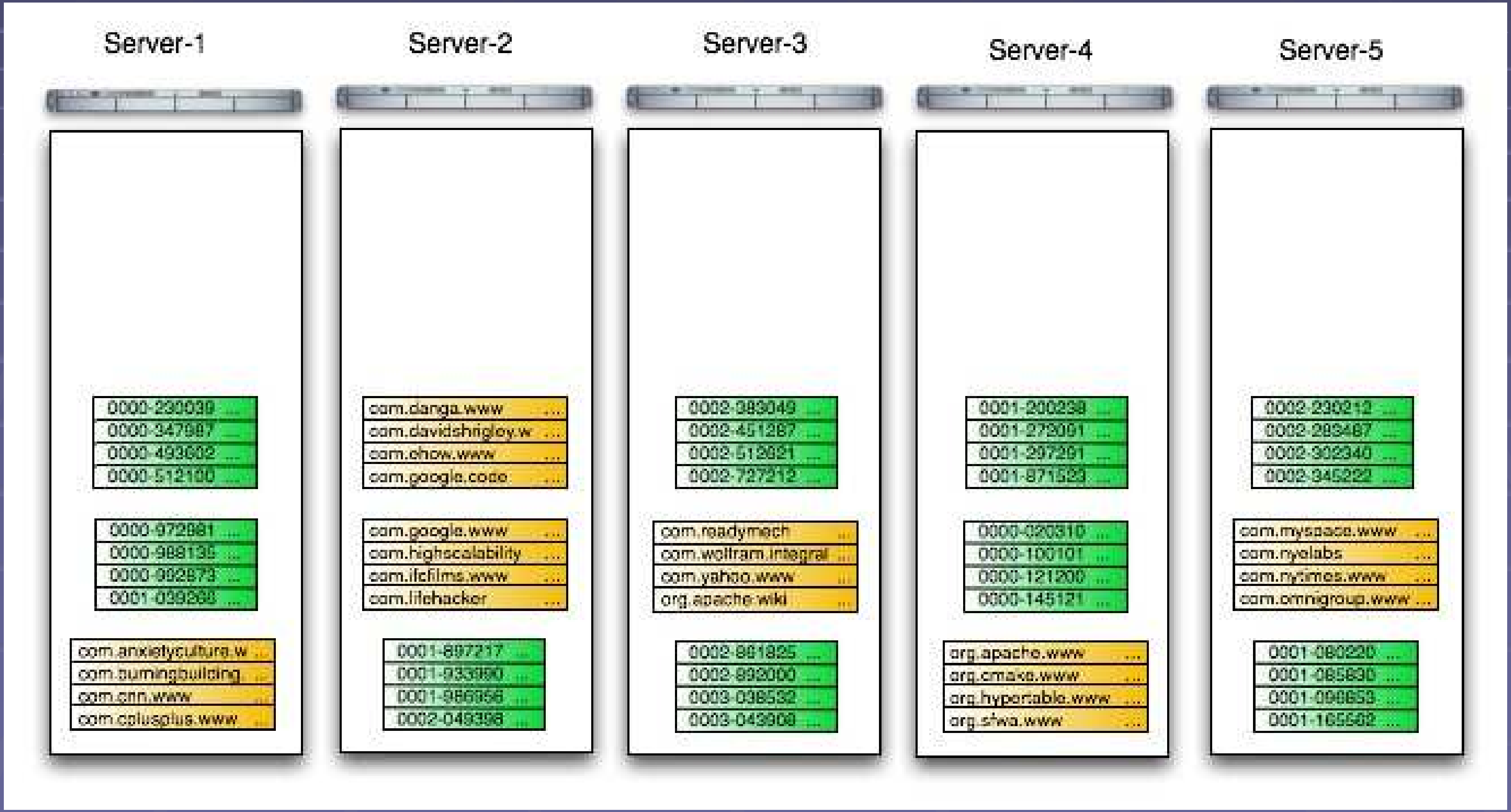
crawldb table	
com.anxietyculture.com	...
com.burningbuilding.www	...
com.cnn.www	...
com.cplusplus.www	...
com.danga.www	...
com.davidshrigley.www	...
com.ehow.www	...
com.google.code	...
com.google.www	...
com.highscalability	...
com.ifcfilms.www	...
com.lifehacker	...
com.myspace.www	...
com.nyelabs	...
com.nytimes.www	...
com.omnigroup.www	...
com.readymech	...
com.wolfram.integrals	...
com.yahoo.www	...
org.apache.wiki	...
org.apache.www	...
org.cmake.www	...
org.hypertable.www	...
org.sfwa.www	...

Scaling (part II)



hypertable.org

Scaling (part III)



Performance Test (AOL Query Logs)

- 75,274,825 inserted cells
- 8 node cluster
  - 1 1.8 GHz Dual-core Opteron
  - 4 GB RAM
  - 3 x 7200 RPM SATA drives
- Average row key: 7 bytes
- Average value: 15 bytes
- Replication factor: 3
- 4 simultaneous insert clients
- 500K **random** inserts/s
- 680K scanned cells/s

Performance Test II

- Simulated AOL query log data
- 1TB data
- 9 node cluster
  - 1 2.33 GHz quad-core Intel
  - 16 GB RAM
  - 3 x 7200 RPM SATA drives
- Average row key: 9 bytes
- Average value: 18 bytes
- Replication factor: 3
- 4 simultaneous insert clients
- Over 1M **random** inserts/s (sustained)

Project Status

- Currently in “alpha”
  - About to release version 0.9.1.0
- “beta” release at the end of January
- Load balancing will come in 1.1 release

Questions?

- www.hypertable.org