NOSQL
AND CASSANDRA

@rantav

ABOUT MYSELF

@rantav

@reversim

reversim.com

prettyprint.me

github.com/rantav

speakerdeck.com/u/rantav/

ABOUT MYSELF

Full stack developer

User of Cassandra

Contributor to Cassandra

Creator of Hector

User of MongoDB


Redis
HBASE
Cassandra
mongoDB
riak
CouchDB
relax
Tokyo
Cabinet
8192PB
Scalaris
Project Voldemort
Neo4j
the graph database
membase

NOSQL

NOSQL

A Multi headed beast?

WHAT'S WRONG WITH SQL?

- **Nothing really wrong**
- But **some problems** - easier to solved with no:sql
- Examples:
  - High scale (reads, write, data size)
  - Data structures (sorted sets, hashes)
  - Graph processing

RELATIONAL DATABASES - SWISS ARMY KNIFE?

- You used to be able to do **everything** with Relational Databases

- Until you reach:
  - **Scaling** limits
  - **$$$** limits
  - **Flexibility** limits
  - **Performance** limits

NOSQL USE CASES

- Large amounts of data (linear horizontal scalability)
- Massive writes
- Speedy key-value access
- Graphs
- High availability
- Easier for ops (well, for some...)
- Easier for programmers

TAXONOMY

KEY-VALUE CACHING

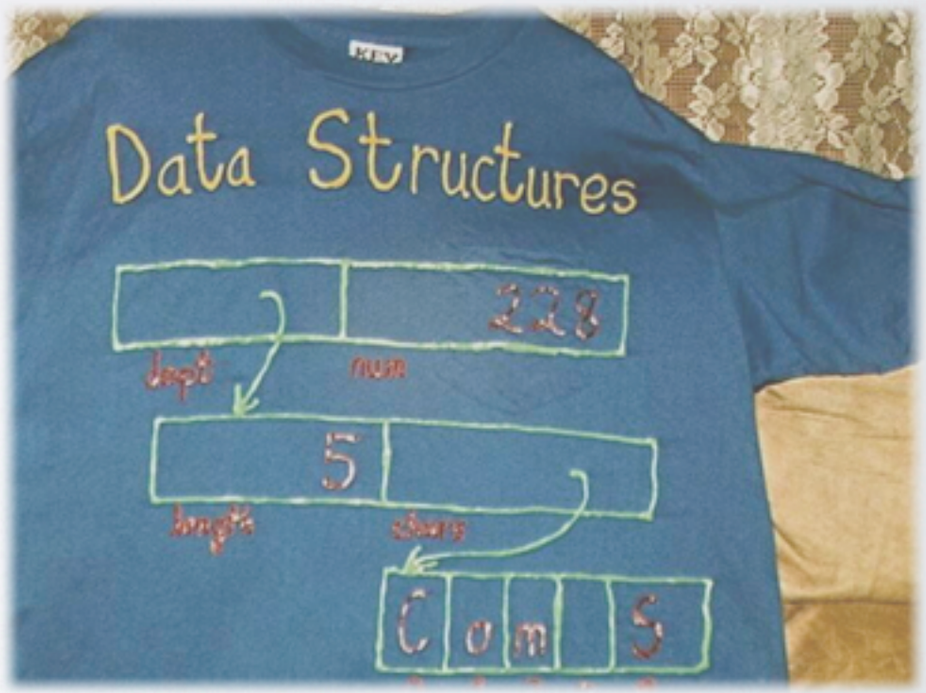
- **memcached**
- repcached
- coherence
- infinispan
- eXtreme scale
- jboss cache
- velocity
- Terracotta

KEY-VALUE STORE

- **DynamoDB**
- voldemort
- Dynomite
- SubRecord
- Mo8onDb
- Dovetaildb
- **Tokyo Tyrant**
- lightcloud
- NMDB
- luxio
- **memcachedb**
- **CouchBase**
- actord
- **Riak**

DATA STRUCTURE STORES

- **Redis**

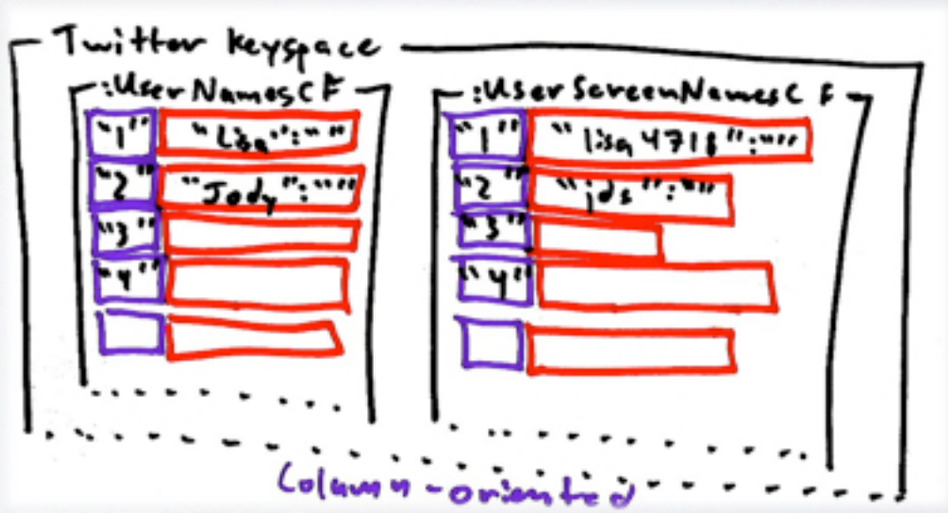


DOCUMENT STORE

- **CouchDB**
- **MongoDB**
- Jackrabbit
- XML Databases
- ThruDB
- CloudKit
- Perservere
- Scalaris

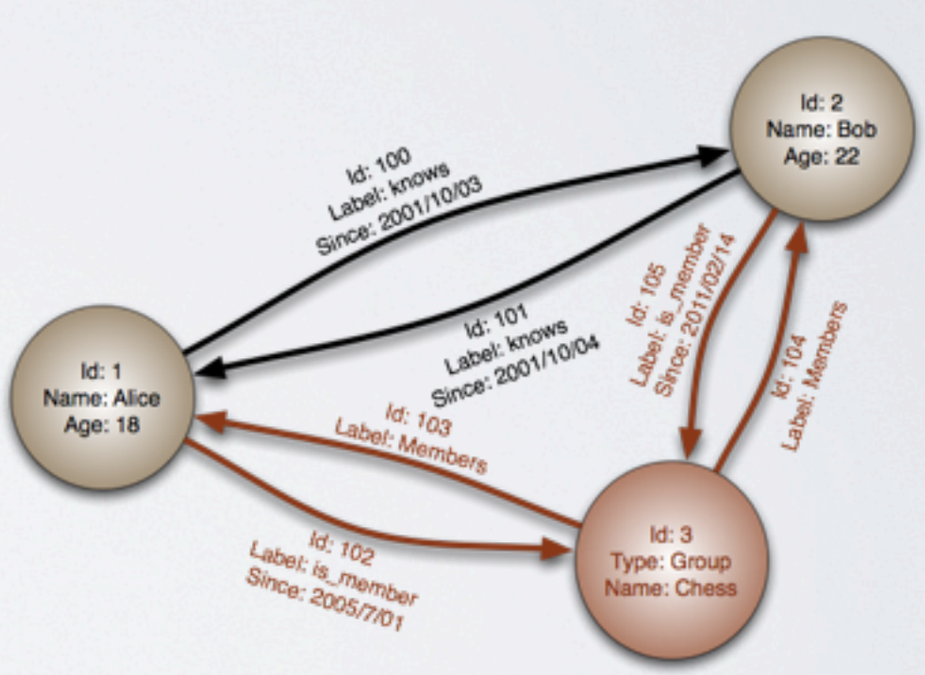
COLUMNAR STORES

- **BigTable**
- **Hbase**
- **Cassandra**
- Hypertable
- KAI
- OpenNeptune
- Qbase
- KDI



GRAPH DATABASES

- **Neo4J**
- VertexDB
- FlockDB
- DEX
- OrientDB



- Decentralized (p2p)
- Very high scale
- Column oriented
- Replication and Multi-DC
- Fault tolerant
- Tunable consistency
- Tunable Caching
- MapReduce
- Query language (CQL)
- Thrift

- Document Database
- Ad-hoc queries
- Indexing
- Replication, Sharding
- MapReduce
- File storage
- Capped collections
- Flexible schema

- Key-Value store
- Data structure server
- (Mostly) in-memory
- Very high performance
- Master-slave setup
- Pub/Sub
- Atomic, fast operations:
  - Sets
  - Sorted Sets
  - Strings
  - Hashes

Reminder

- ACID
  - Atomicity
  - Consistency
  - Isolation
  - Durability
- Transactional

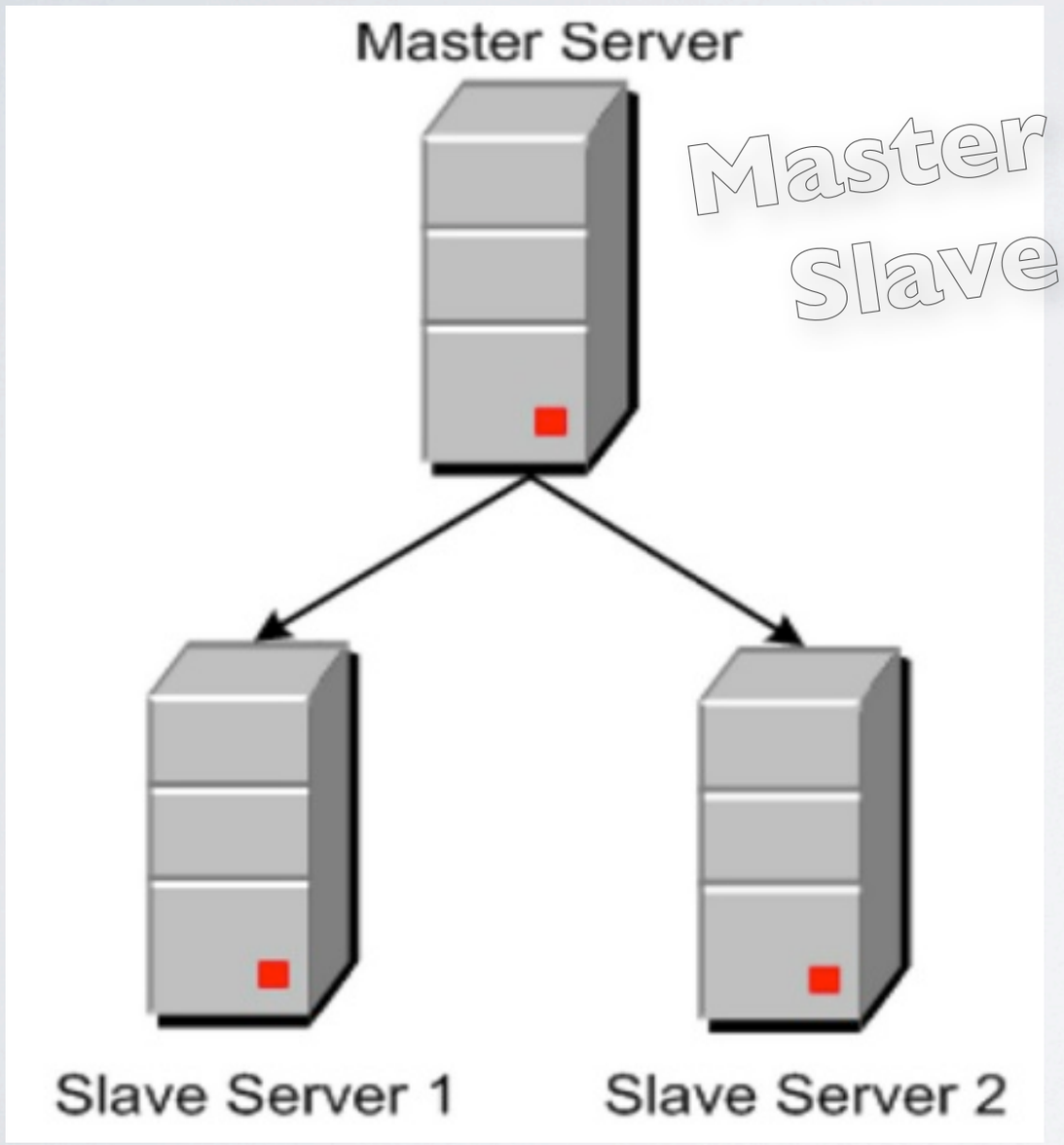
- SQL
- Stored Procedures
- (Usually) optimized for reads
- MySQL Cluster
  - NoSQL and SQL mix

TODAY'S FOCUS:
SCALING

- "Internet scale" data size
- High read/write rates
- Frequent schema changes
- **Social** apps: not **banks**

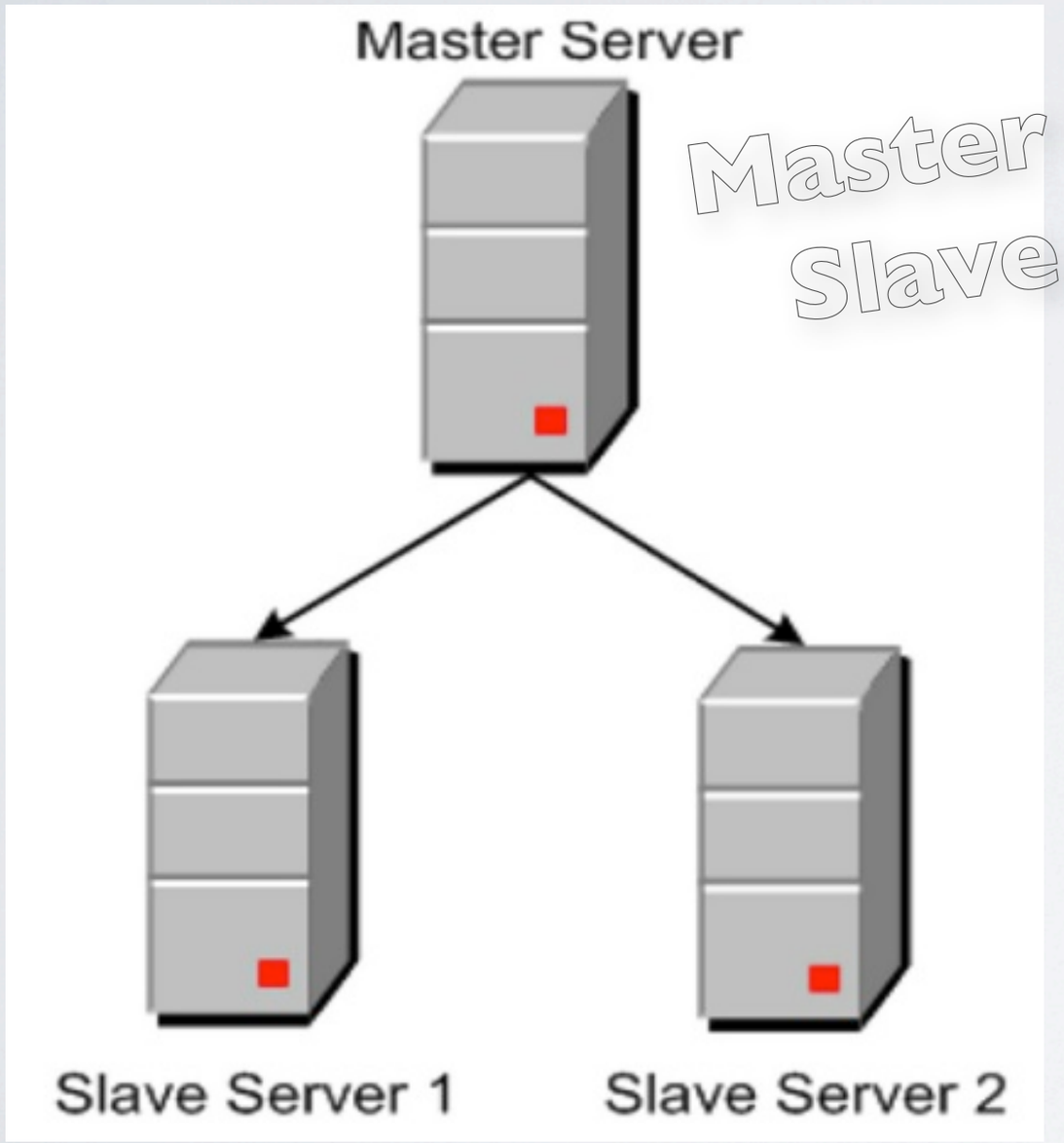
  ⇒can relax ACID

TRADITIONAL SCALING

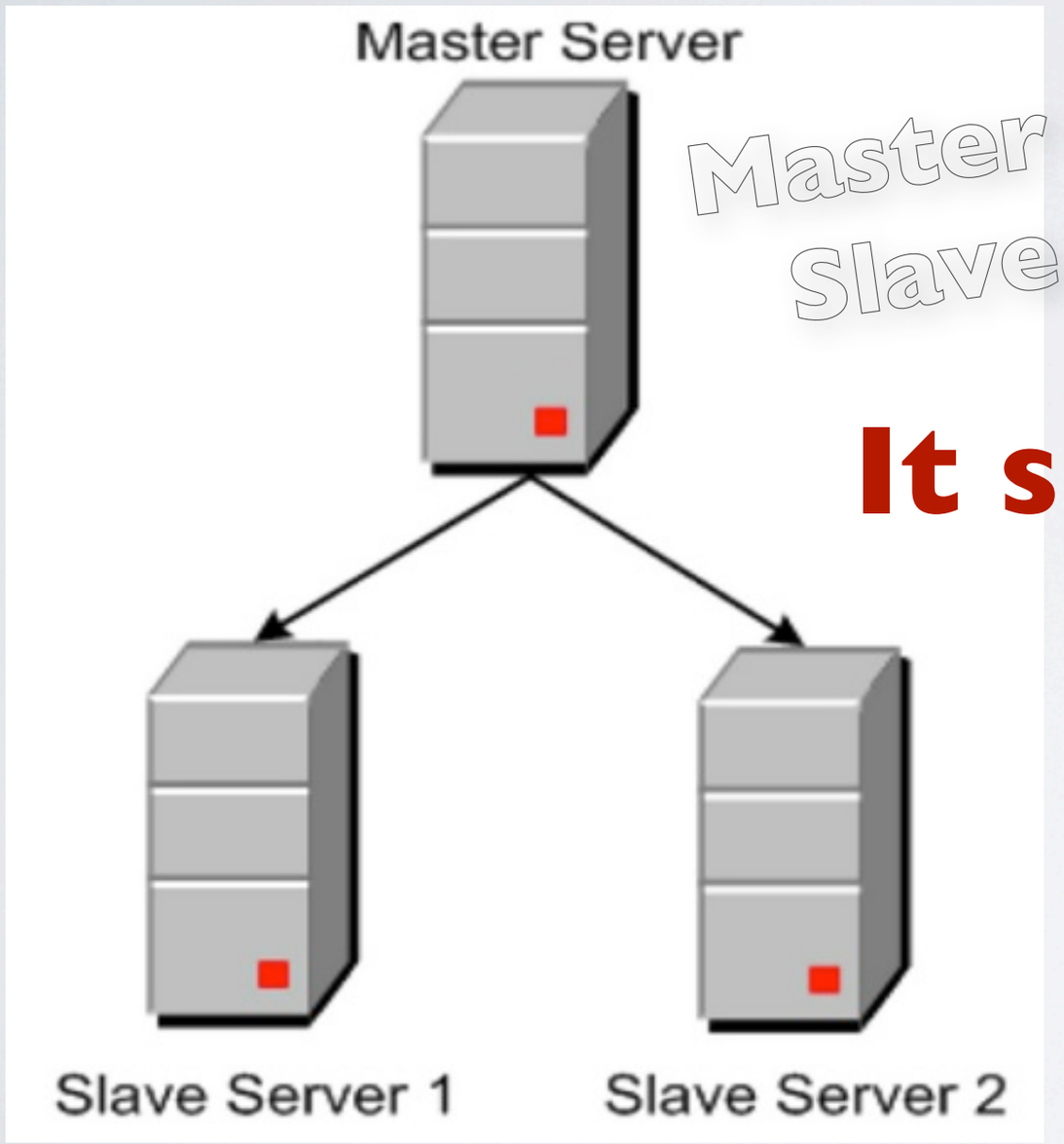


Master - Slave

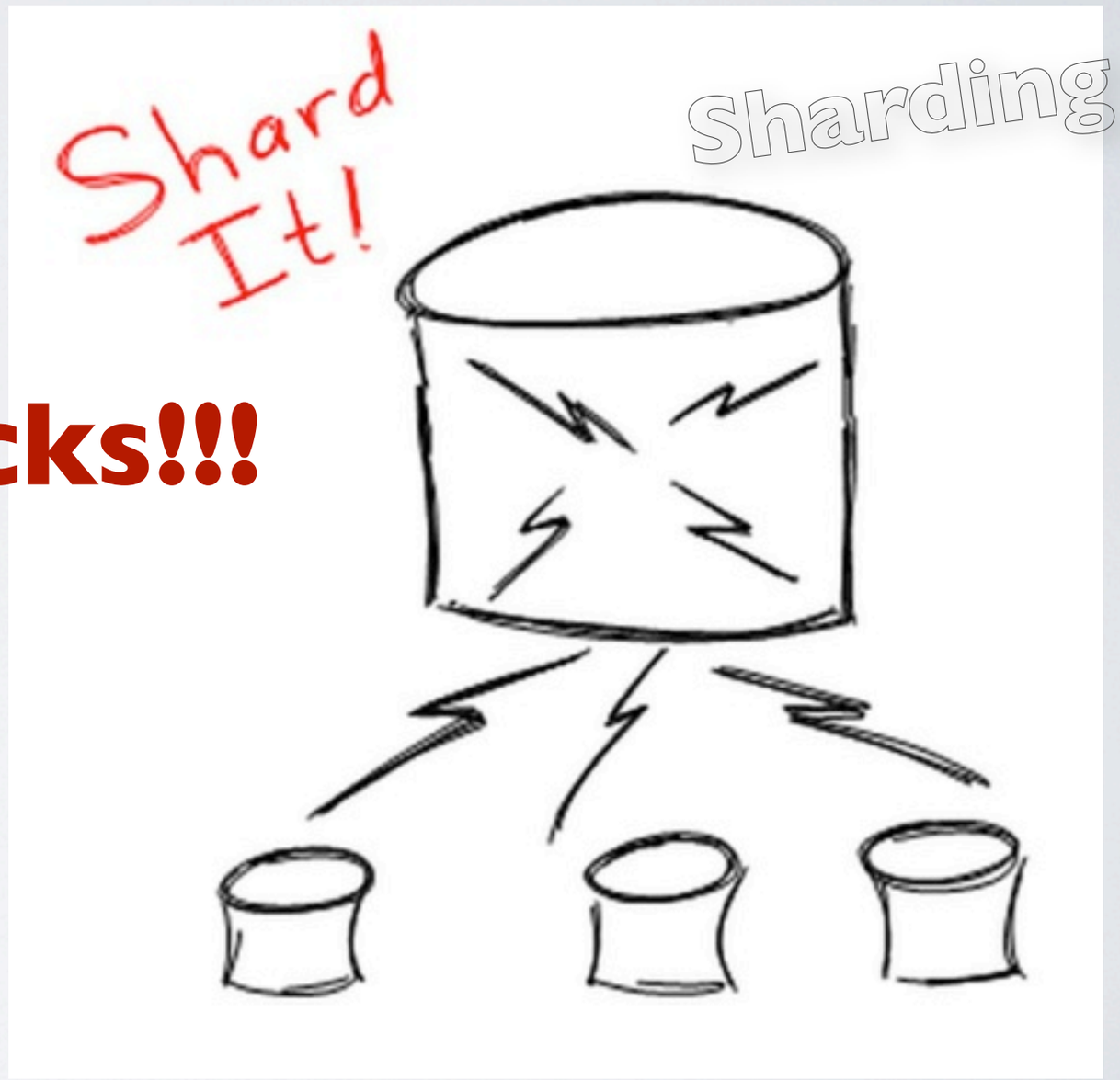
TRADITIONAL SCALING



TRADITIONAL SCALING



It sucks!!!



CASSANDRA

- Developed at Facebook
- Now used by many .com
- BigTable data model: **Columnar**
- Dynamo **Eventual Consistency**
- Apache **open source**
- Implemented in **Java**

CASSANDRA

- Developed at Facebook
- Now used by many .com
- BigTable data model: **Columnar**
- Dynamo **Eventual Consistency**
- Apache **open source**
- Implemented in **Java**

Goal:
Easy Scaling

WHAT IS EVENTUAL CONSISTENCY?

- Full Consistency: One you write, it can be read
- Eventual Consistency: Given enough time, **eventually** data can be read

Observation:

Eventual Consistency ≠ A**C**ID

ACADEMIC

DIFFERENT LEVELS OF EVENTUAL CONSISTENCY

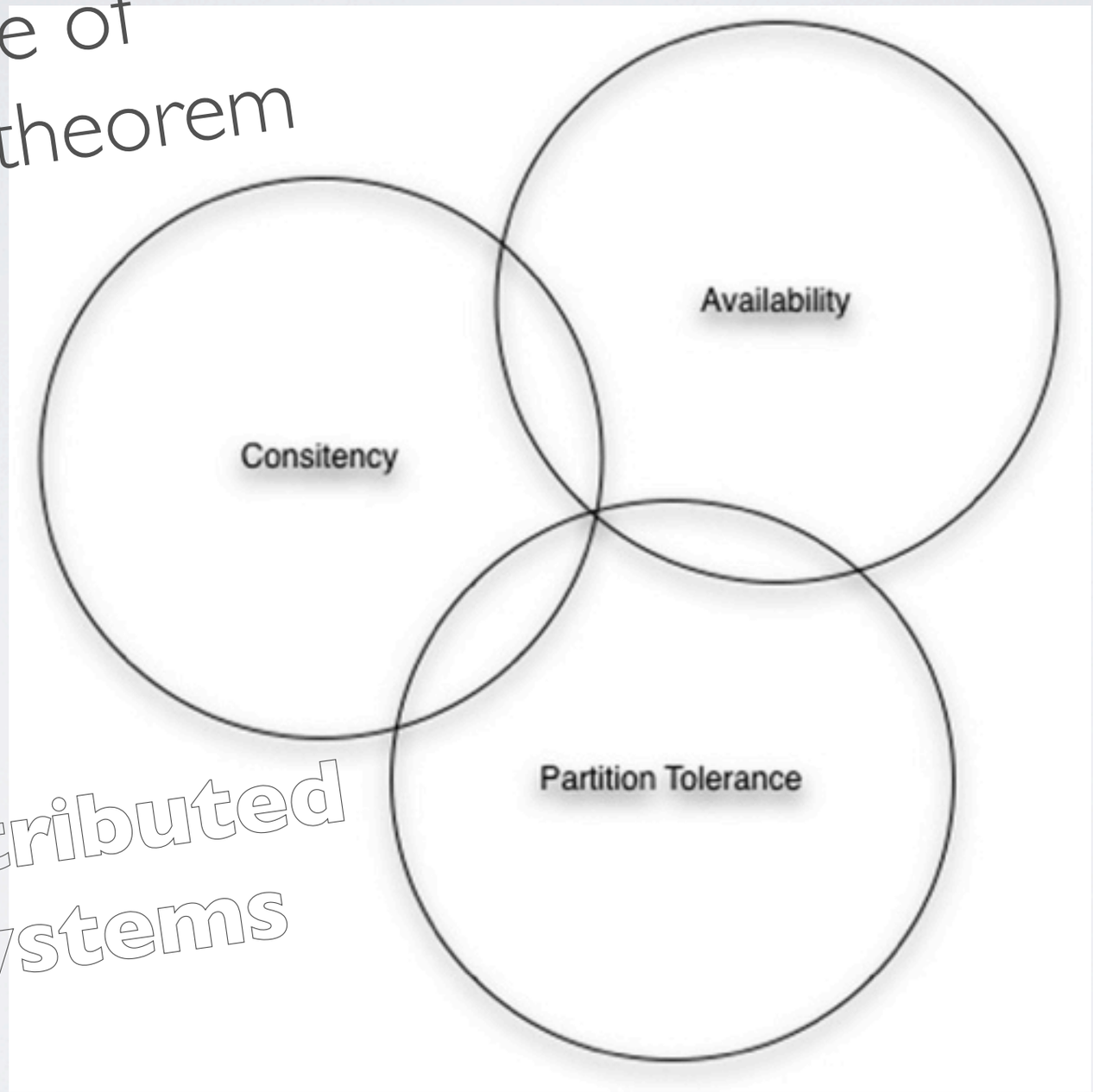
- Causal consistency
- Read-your-writes consistency
- Session consistency
- Monotonic read consistency
- Monotonic write consistency

ACADEMIC

WHY EVENTUAL CONSISTENCY AT ALL?

WHY EVENTUAL CONSISTENCY AT ALL?

Because of the **CAP** theorem



Distributed Systems

Academic

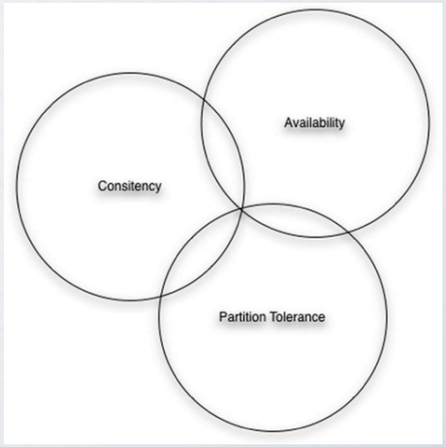
EVENTUAL CONSISTENCY DOWN TO EARTH

How it's done in Cassandra

N - Number of replicas (nodes) for any data item

W - Number or nodes a write operation blocks on

R - Number of nodes a read operation blocks on



NRW - TYPICAL VALUES

W=1 ⇒ Block until **first** node writes successfully

W=N ⇒ Block until **all** nodes writes successfully

W=0 ⇒ **Async** writes

NRW - TYPICAL VALUES

R=1 ⇒ Block until the **first** node returns an answer

R=N ⇒ Block until **all** nodes return an answer

R=0 ⇒ Doesn't make sense

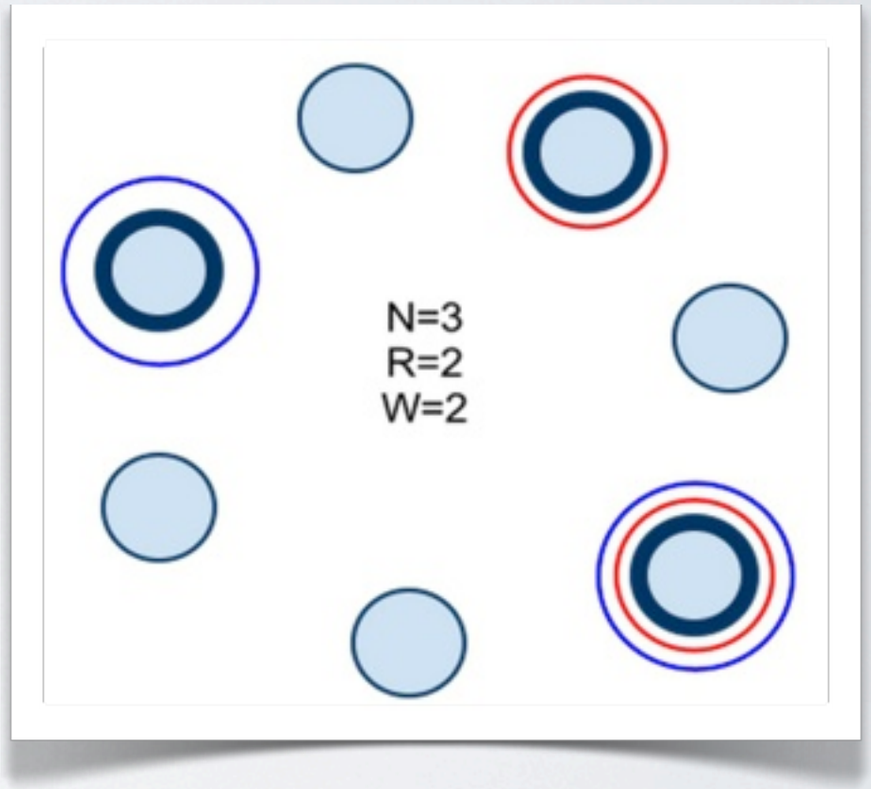
NRW - TYPICAL VALUES

QUORUM

R = N/2+1

W = N/2+1

⇒ Fully consistent



NRW - OBSERVATION

Observation:

When R + W < N ⇒ Weak consistency

DATA MODEL IN CASSANDRA

Forget SQL

DATA MODEL - VOCABULARY

- **Keyspace** – like namespace for unique keys.
- **Column Family** – very much like a table… but not quite.
- **Key** – a key that represent row (of columns)
- **Column** – representation of value with:
  - Column name
  - Value

DATA MODEL

```
Users:                                          CF
    ran:                                        ROW
        emailAddress: foo@bar.com,              COLUMN
        webSite: http://bar.com                 COLUMN
    f.rat:                                      ROW
        emailAddress: f.rat@mouse.com           COLUMN
Stats:                                          CF
    ran:                                        ROW
        visits: 243                             COLUMN
```

DATA MODEL

```
Users:                                  CF
    ran:                                ROW
        emailAddress: foo@bar.com,      COLUMN
        webSite: http://bar.com         COLUMN
    f.rat:                              ROW
        emailAddress: f.rat@mouse.com   COLUMN
Stats:                                  CF
    ran:                                ROW
        visits: 243                     COLUMN
```

Sparse Columns

DATA MODEL

```
Songs:
    Meir Ariel:
        Shir Keev: 6:13
        Tikva: 4:11
        Erol: 6:17
        Suetz: 5:30
        Dr Hitchakmut: 3:30
    Mashina:
        Rakevet Layla: 3:02
        Optikai: 5:40
```

Go wild with column names

THE API

- Thrift
- CQL
- Client libraries:
  - Hector
  - PyCassa
  - Helenus
  - Aquiles
  - Many more...

READ API

```
get                          get_ranage_slices

get_slice                    get_indexed_slices

get_count                    execute_cql_query

multiget

multiget_slice
```

THE TRUE API

```
get(keyspace, key, column_path, consistency)

get_slice(ks, key, column_parent, predicate, consistency)

multiget(ks, keys, column_path, consistency)
```

WRITE API

```
insert

add

remove

remove_counter

batch_mutate
```

THE TRUE WRITE API

```
insert(..., consistency)

add(..., consistency)

remove(..., consistency)

remove_counter(..., consistency)

batch_mutate(..., consistency)
```

METADATA API

```
describe_schema_versions

describe_keyspaces

describe_cluster_name

describe_version

describe_ring
```

DDL API

```
system_add_column_family

system_drop_column_family

system_add_keyspace

system_drop_keyspace

system_update_keyspace

system_update_column_family
```

CQL API

Thrift: `execute_cql_query`

```
cqlsh> SELECT key, state FROM users;

cqlsh> INSERT INTO users (key, full_name,
birth_date, state) VALUES ('bsanderson',
'Brandon Sanderson', 1975, 'UT');
```

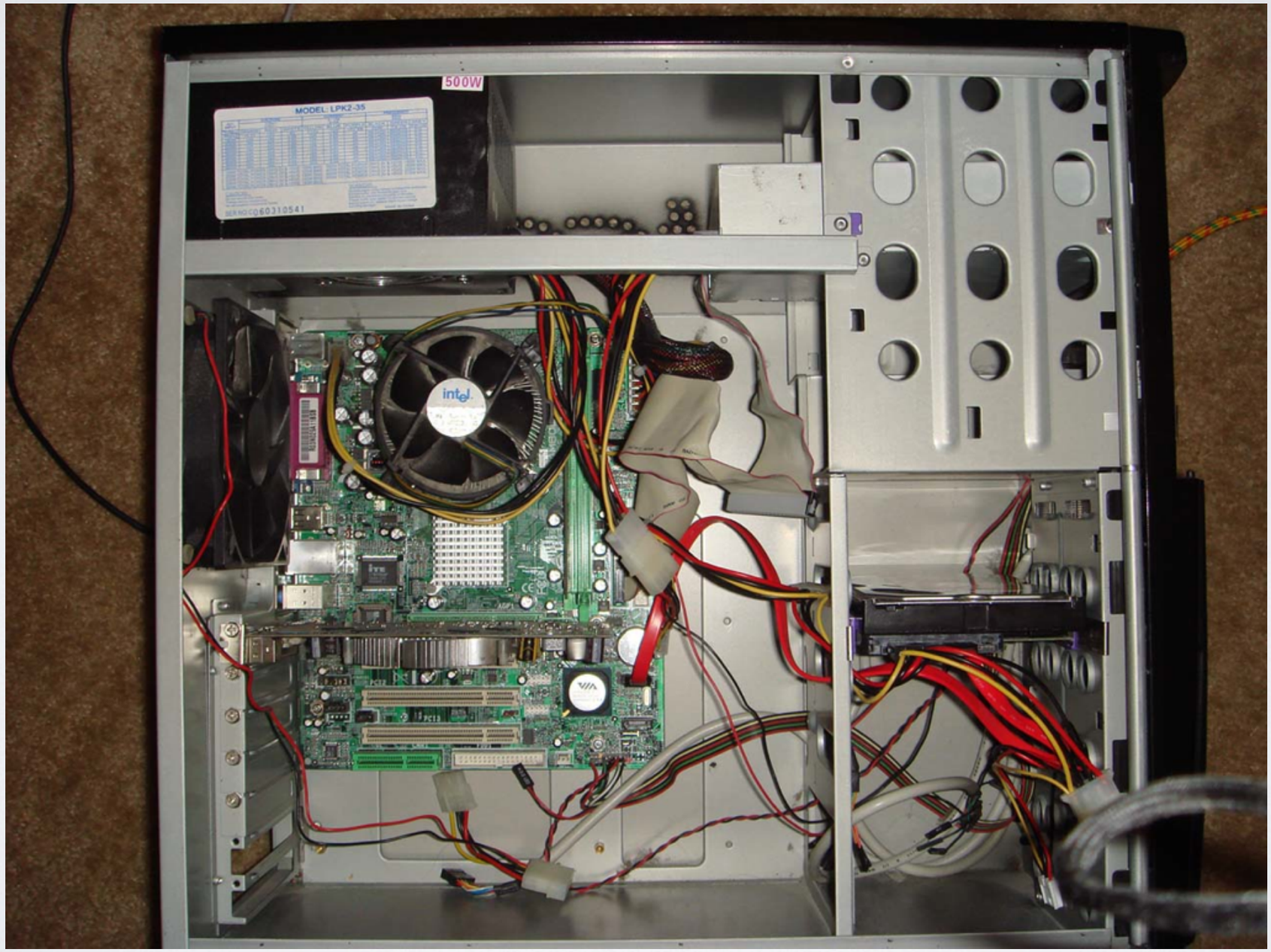
JAVA CODE (HECTOR)

```
/**
 * Insert a new value keyed by key
 *
 * @param key   Key for the value
 * @param value the String value to insert
 */
public void insert(final String key, final String value) {
  Mutator m = createMutator(keyspaceOperator);
  m.insert(“key”,
           “columnFamily”,
           createColumn(“columnName”, value));
}
```

JAVA CODE (HECTOR)

```
/**
 * Get a string value.
 *
 * @return The string value; null if no value exists for the given key.
 */
public String get(final String key) throws HectorException {
  ColumnQuery<String, String> q = createColumnQuery(keyspaceOperator, serializer, serializer);
  Result<HColumn<String, String>> r = q.setKey(key).
      setName("column").
      setColumnFamily("columnFamily").
      execute();
  HColumn<String, String> c = r.get();
  return c == null ? null : c.getValue();
}
```

CASSANDRA INTERNALS



AGENDA

- Background and history
- Architectural Layers
- Transport: (Thrift)
- Write Path
- Read Path
- Compactions
- Bloom Filters
- Gossip
- Deletions
- and more...

BACKGROUND

- Required reading:
  - **BigTable** http://labs.google.com/papers/bigtable.html
  - **Dynamo** http://www.allthingsdistributed.com/2007/10/amazons_dynamo.html

FROM DYNAMO

- Symmetric **p2p** architecture
- **Gossip** based discovery and error detection
- **Distributed** key-value store
- Pluggable **partitioning**
- Pluggable **topology discovery**
- **Eventual consistent** and Tunable per operation

FROM BIGTABLE

- **Sparse** Column oriented sparse array
- **SSTable** disk storage
- Append-only **commit log**
- **Memtable** (buffering and sorting)
- **Compactions**
- High write performance

ARCHITECTURAL LAYERS

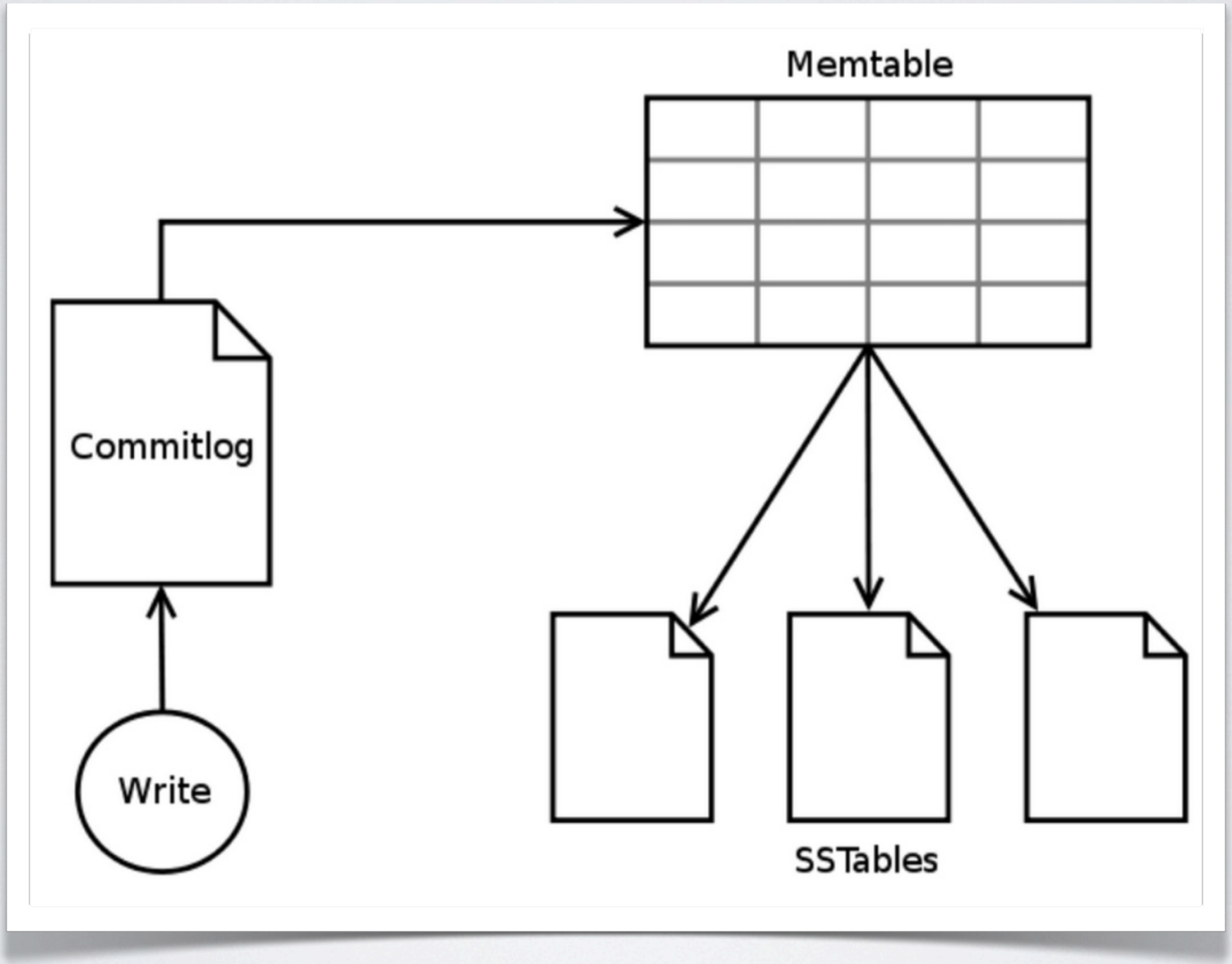
Cluster Management

- Messaging service
- Gossip
- Failure detection
- Cluster state
- Partitioner
- Replication

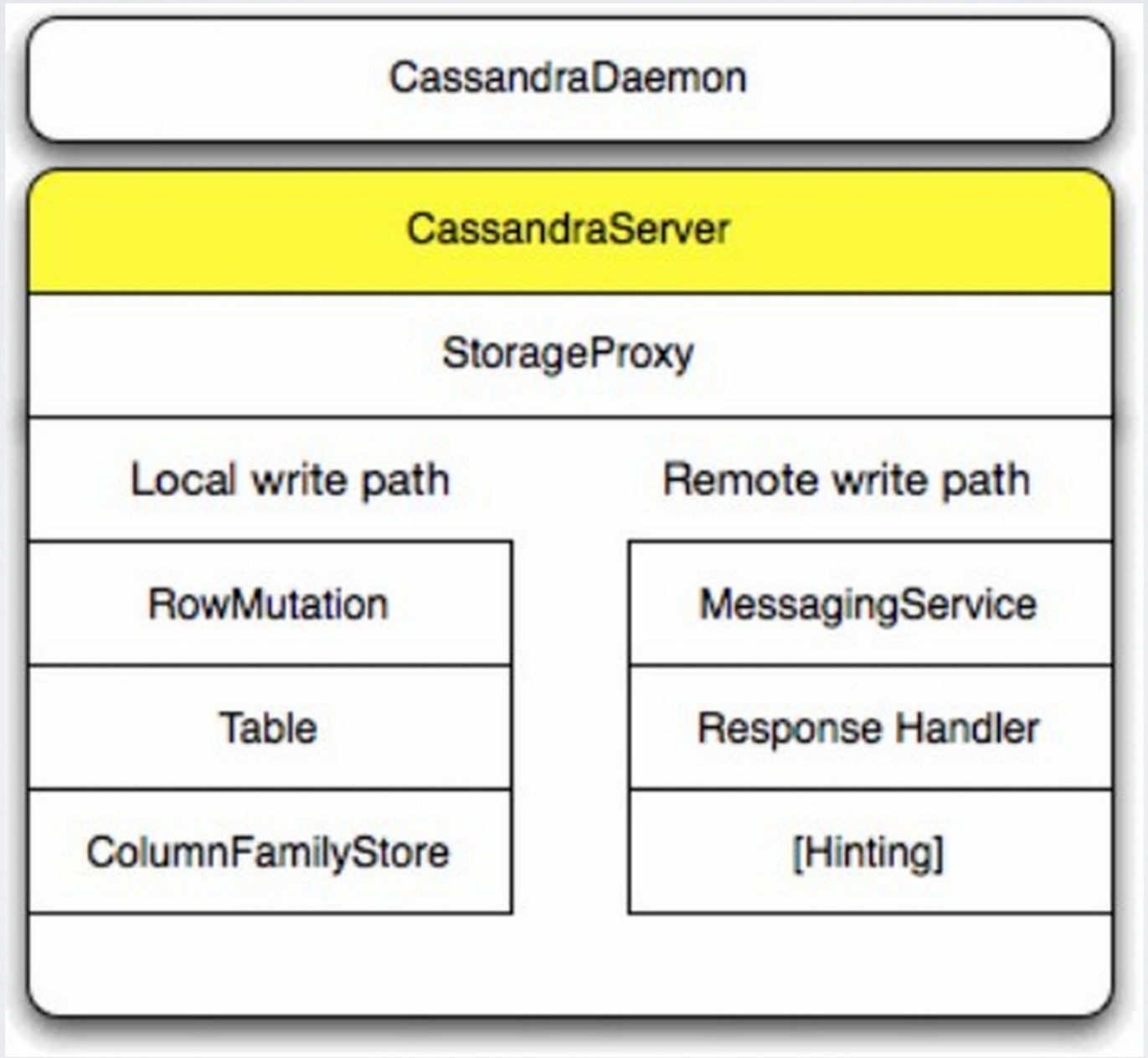
Host Management

- Commit log
- Memtable
- SSTable
- Indexes
- Compaction

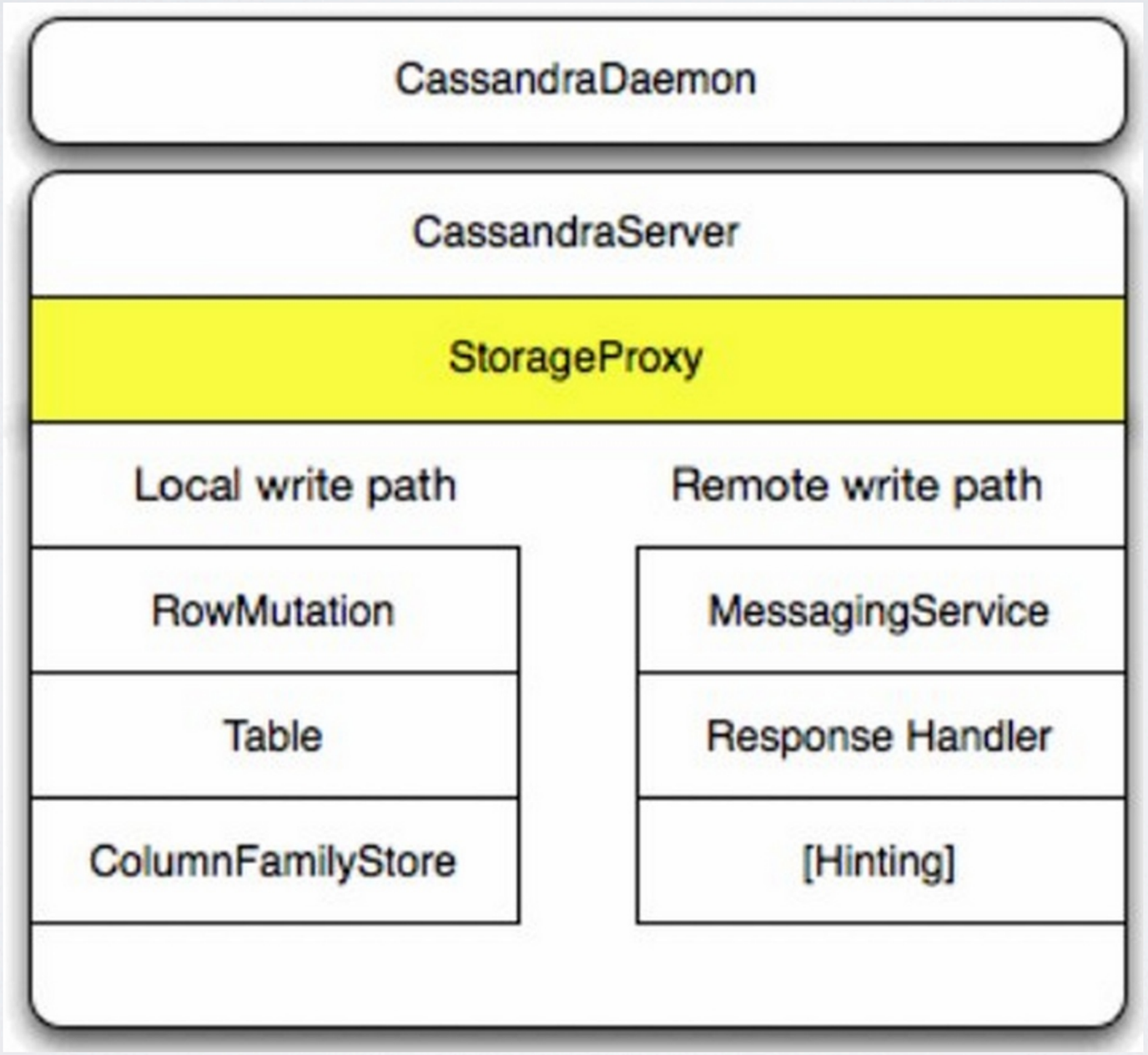
LOCAL WRITE PATH



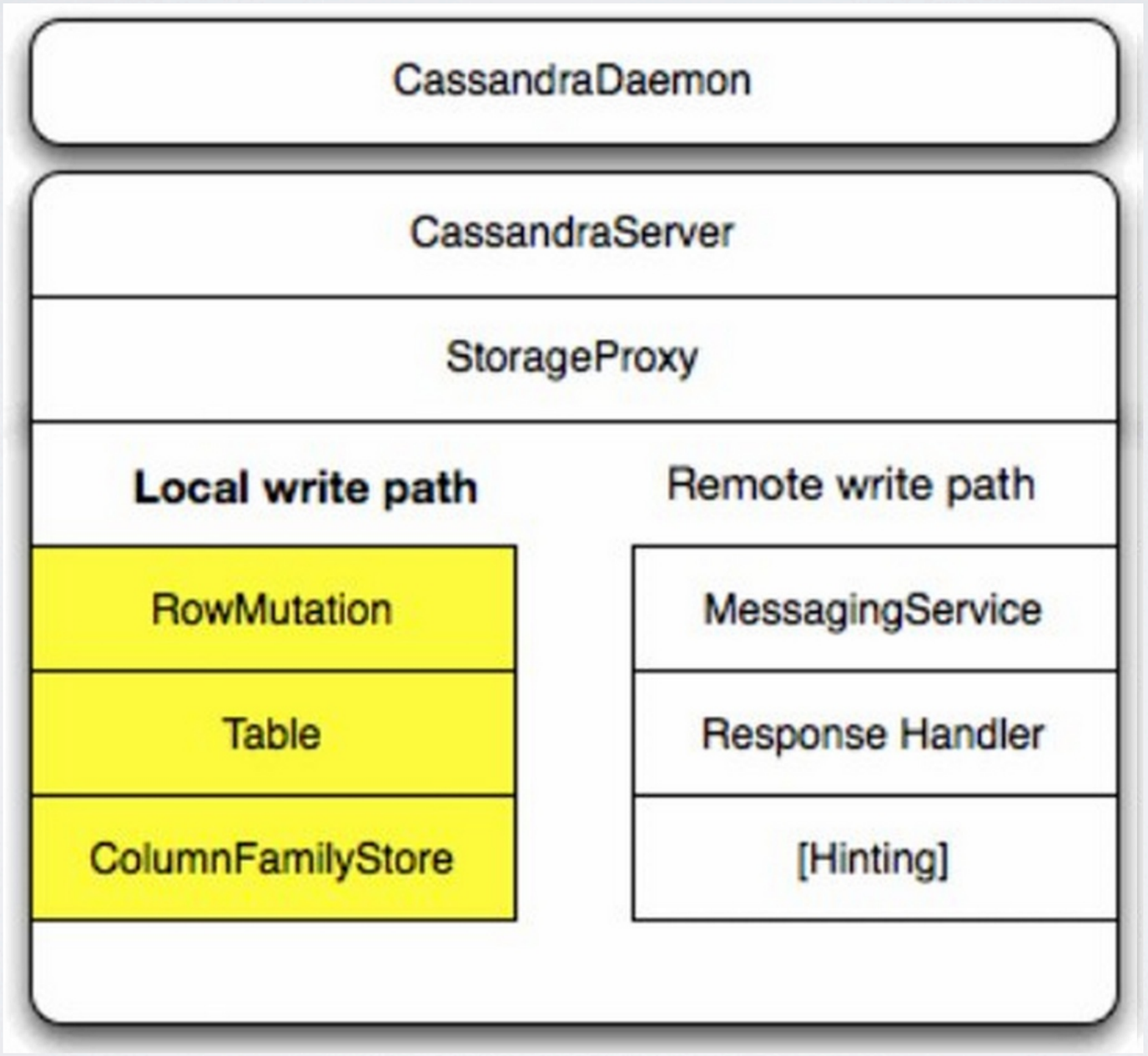
NETWORK WRITE PATH



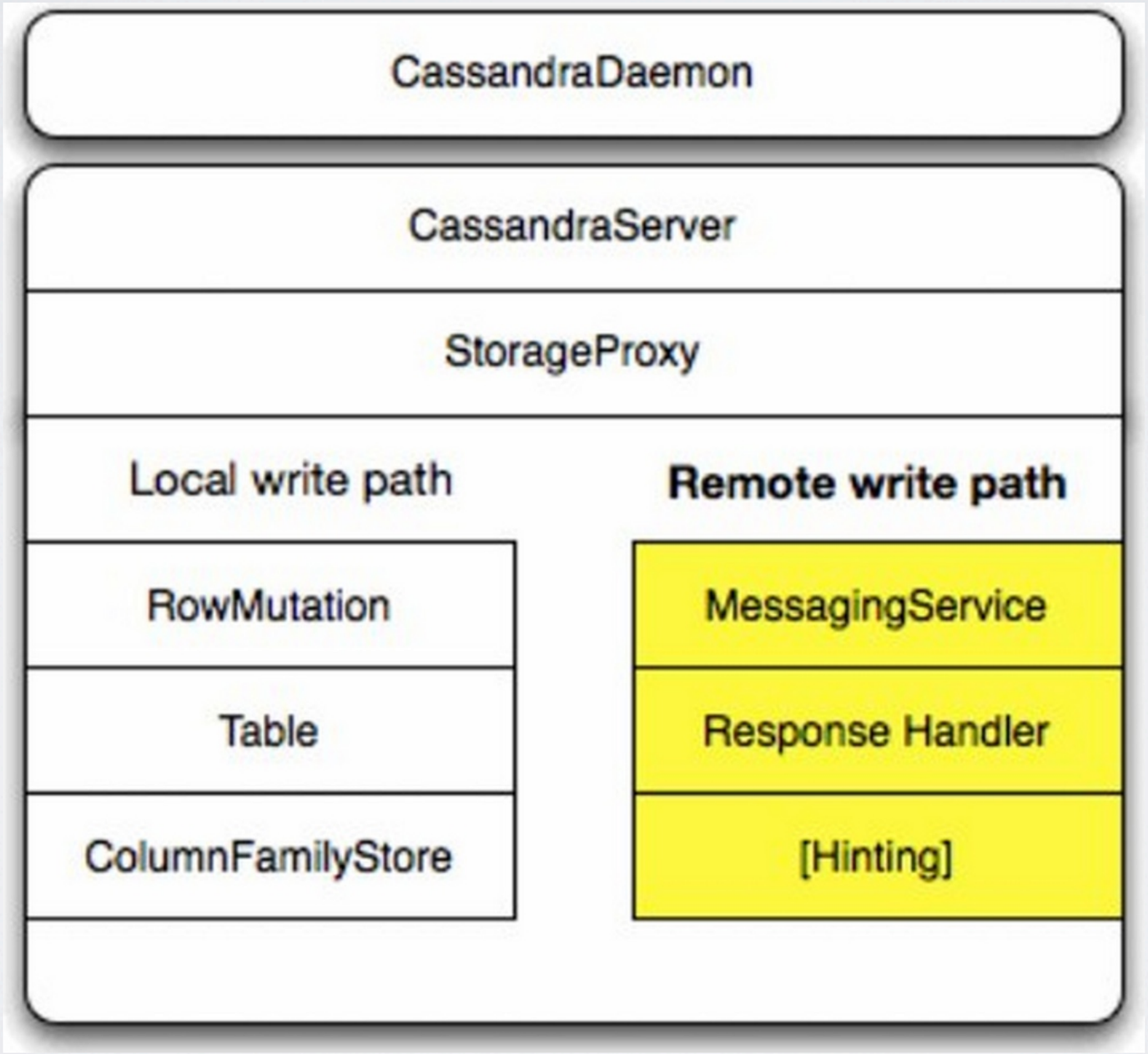
NETWORK WRITE PATH



NETWORK WRITE PATH



NETWORK WRITE PATH



WHAT ARE **MEMTABLES**?

- In-memory representation of recently written data
- When the table is full, it's sorted and then flushed to disk ⇒ sstable

WHAT ARE **SSTABLES**?

Sorted Strings Tables

- Immutable
- On-disk
- Sorted by a string key
- In-memory index of elements
- Binary search (in memory) to find element location
- Bloom filter to reduce number of unneeded binary searches.

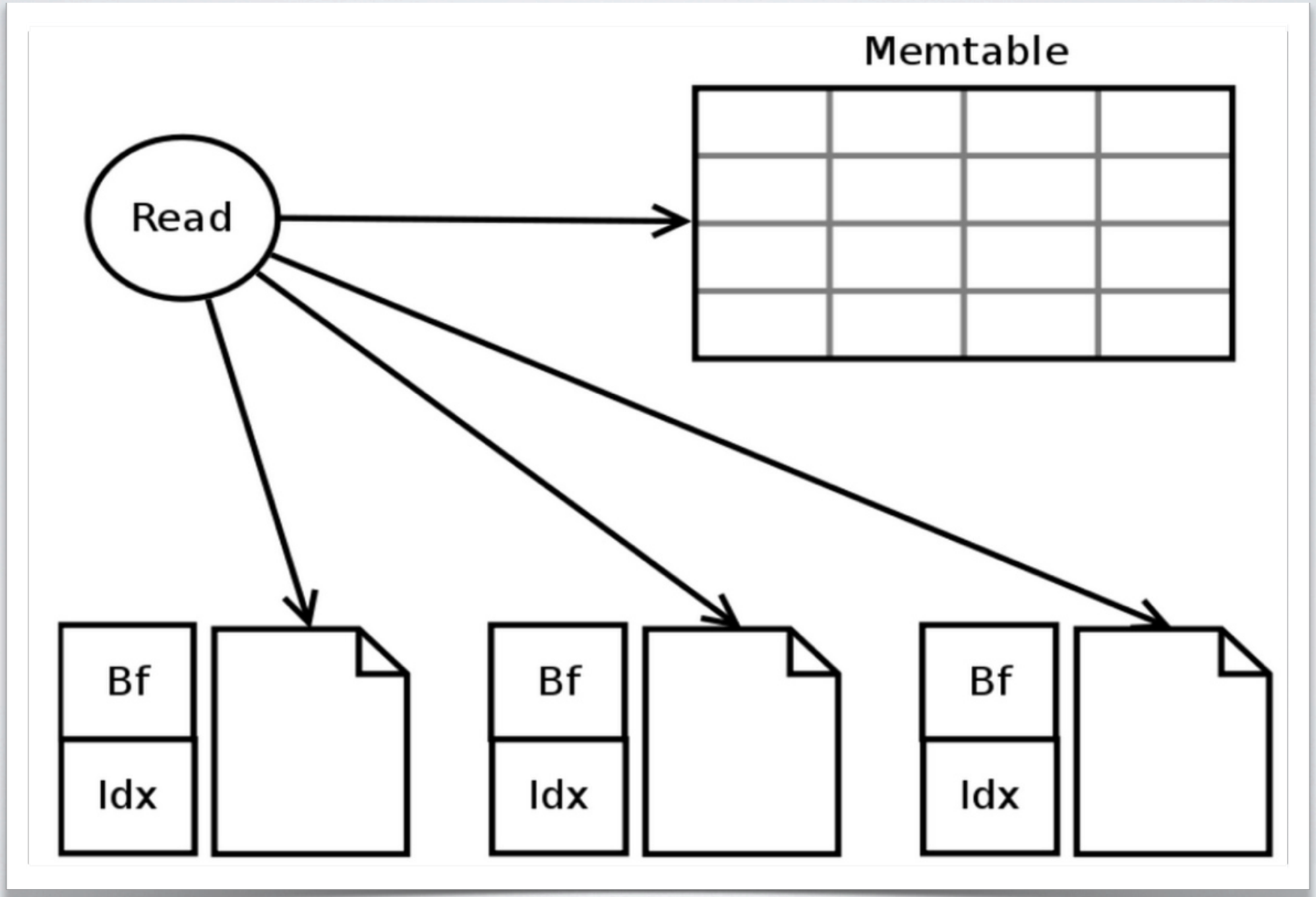
WRITE PROPERTIES

- **No Locks** in the critical path
- **Always available** to writes, even if there are failures.
- **No reads**
- **No seeks**

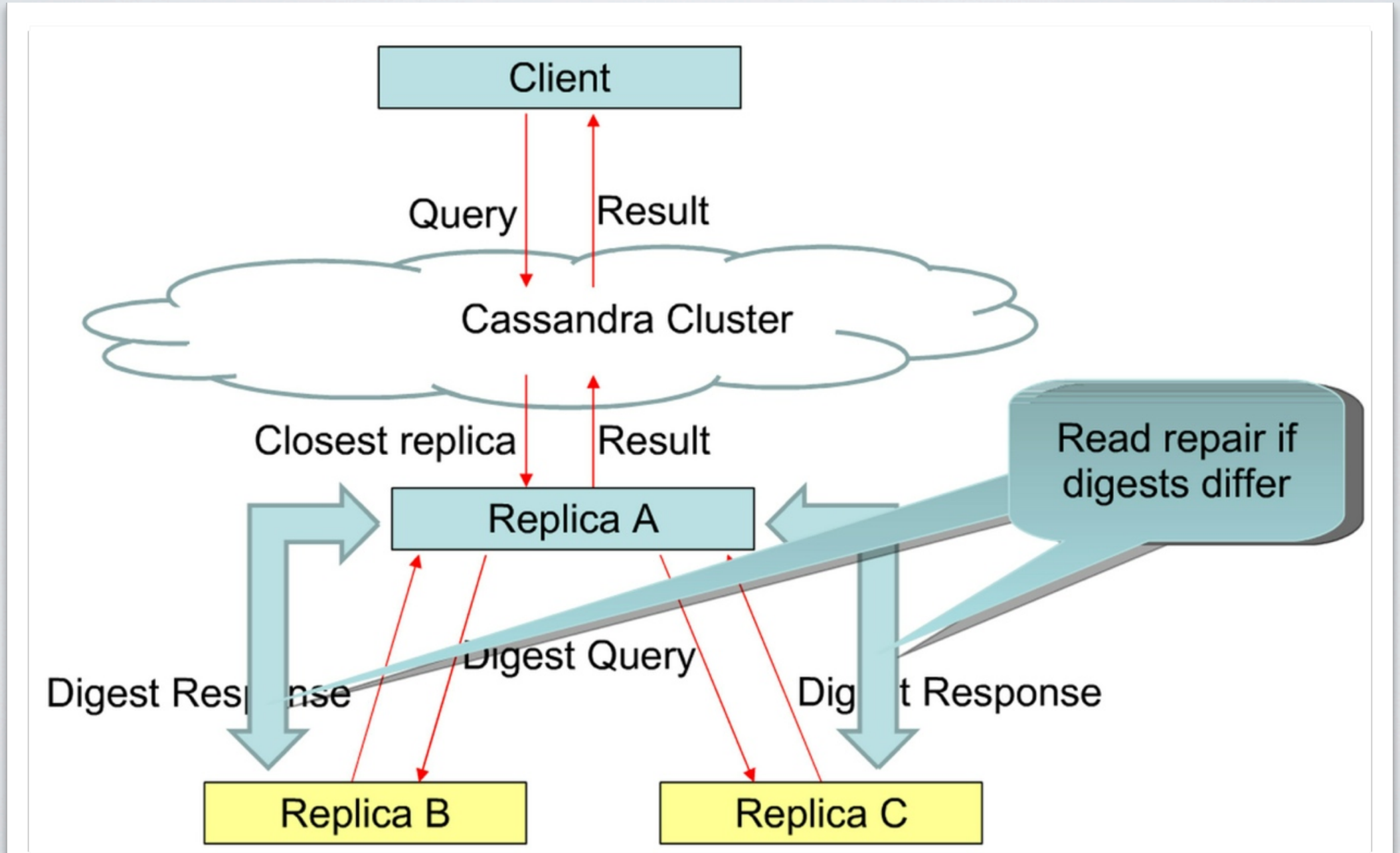
  ⇒Fast

- **Atomic** within a Row

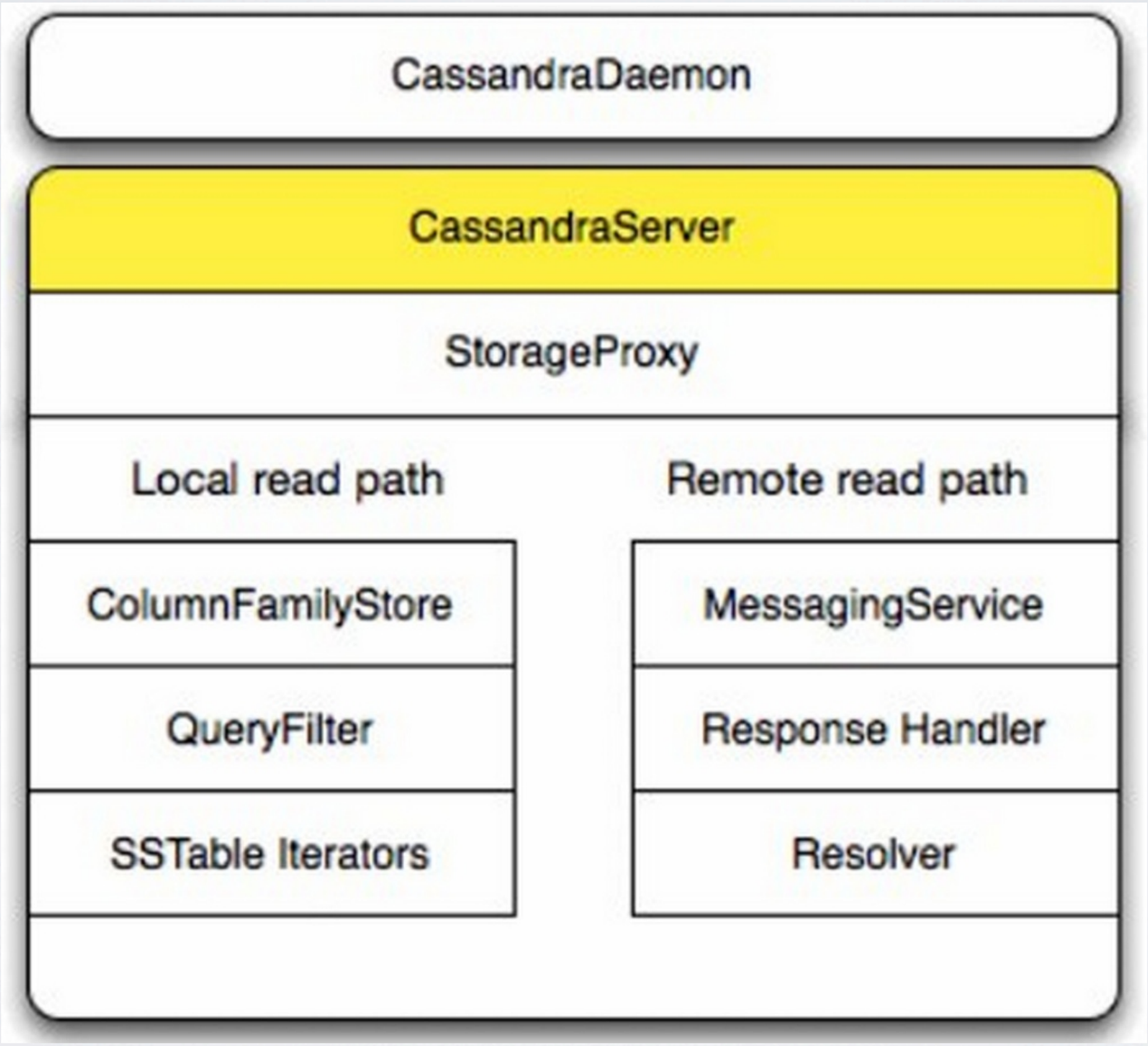
LOCAL READ PATH



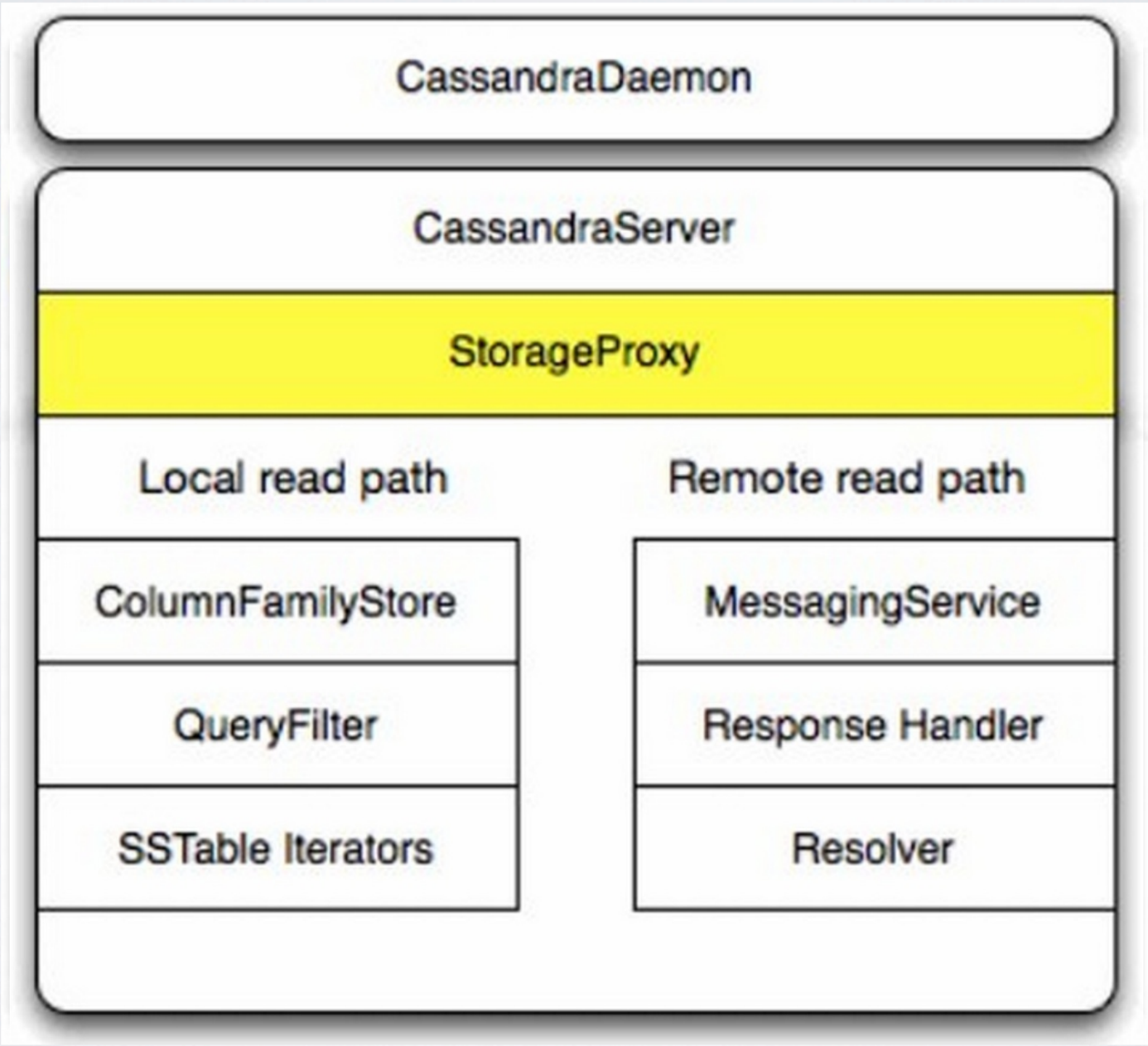
NETWORK READ PATH



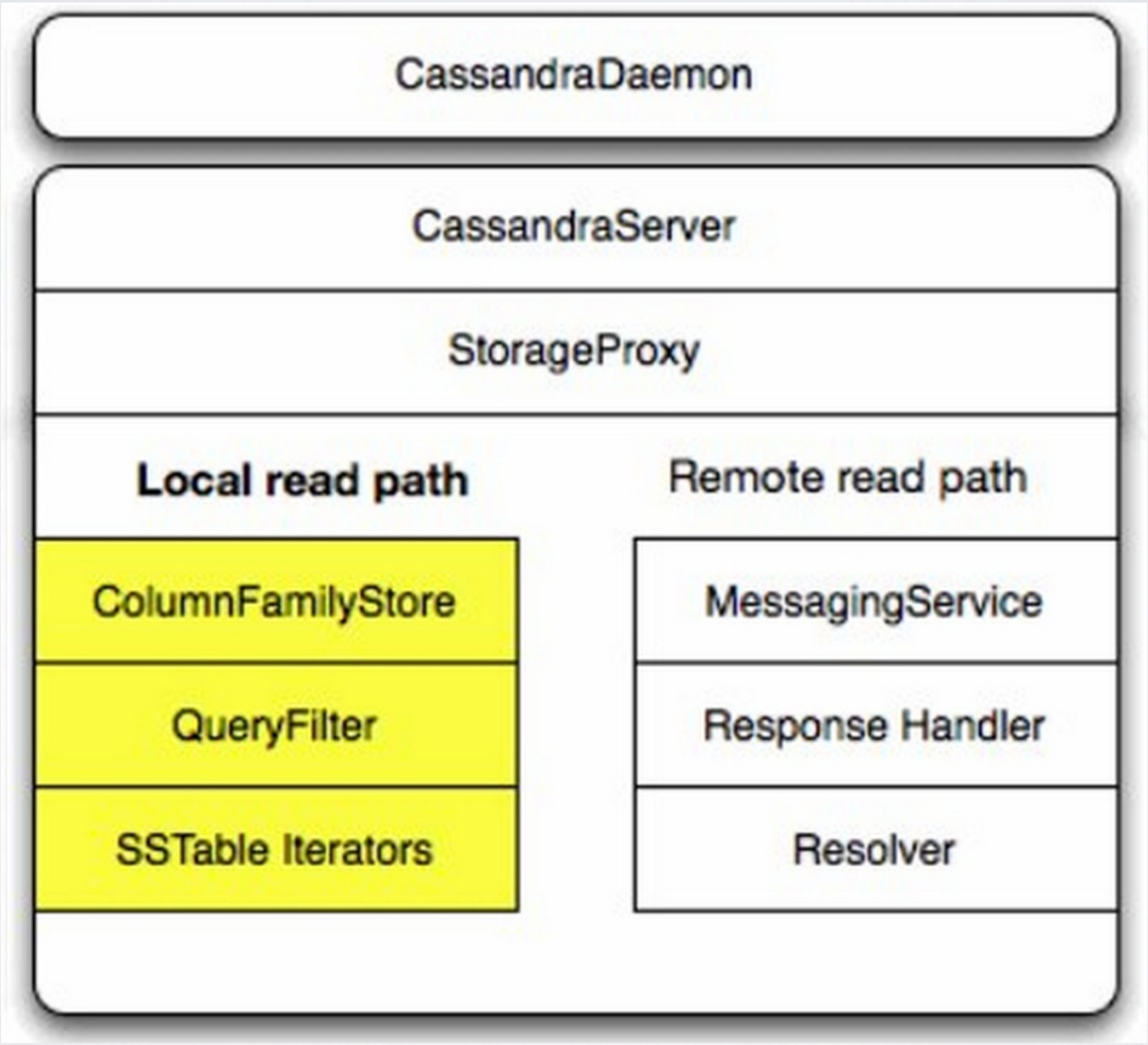
NETWORK READ PATH



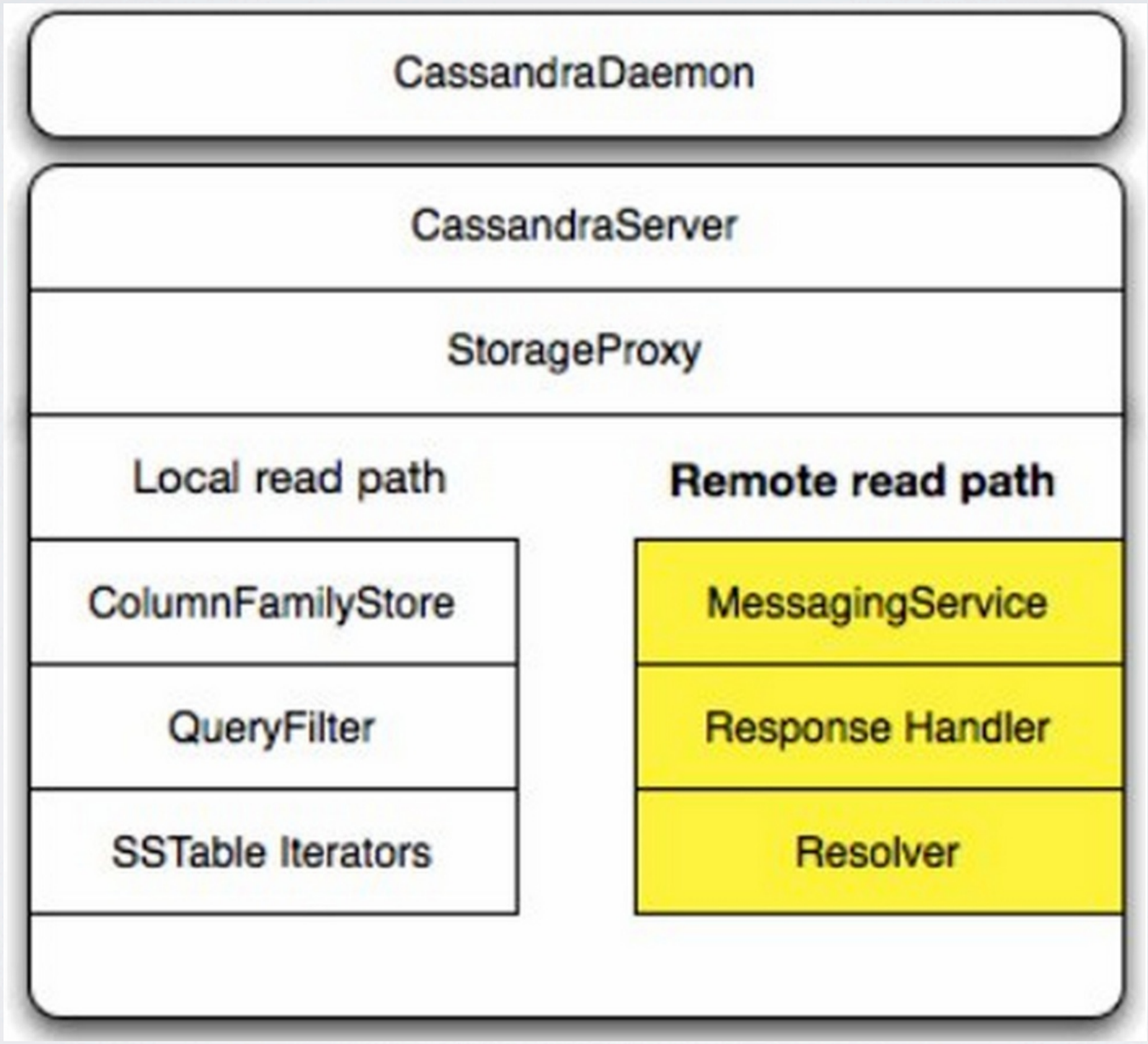
NETWORK READ PATH



NETWORK READ PATH



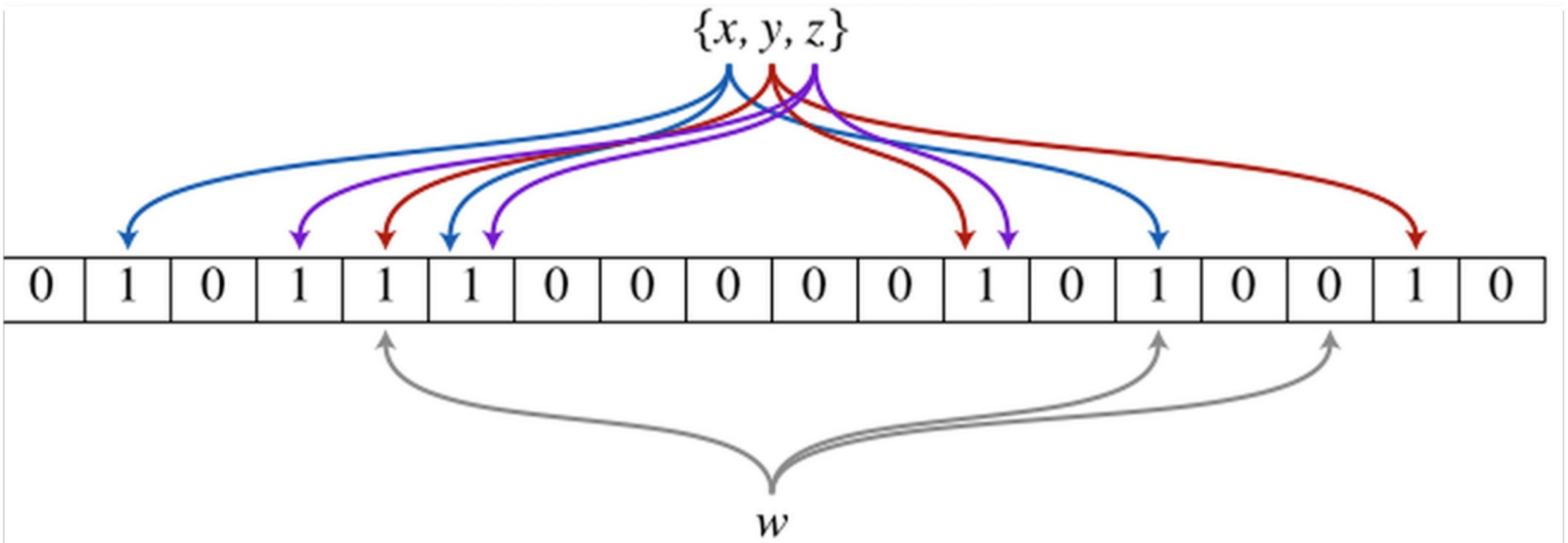
NETWORK READ PATH



READ PROPERTIES

- Read multiple SSTables
- Slower than writes (but still fast)
- Seeks can be mitigated with more RAM
- Uses probabilistic bloom filters to reduce lookups.
- Extensive (optional) caching
  - Key Cache
  - Row Cache
- Excellent monitoring

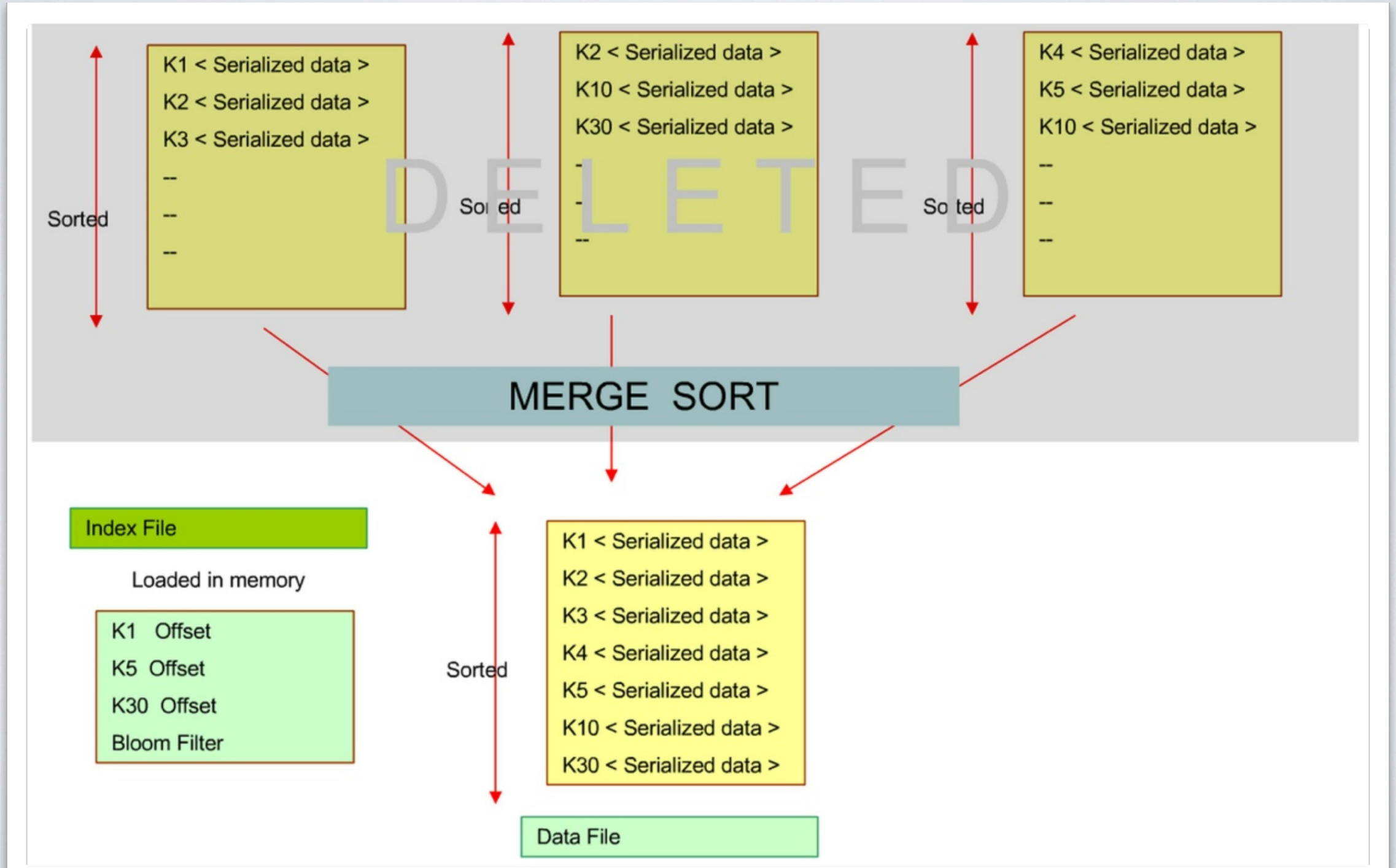
BLOOM FILTERS



BLOOM FILTERS

- Space efficient probabilistic data structure
- Test whether an element is a member of a set
- Allow false positive, but not false negative
- k hash functions
- Union and intersection are implemented as bitwise OR, AND

COMPACTIONS



COMPACTIONS

- Merge keys
- Combine columns
- Discard tombstones
- Merge bloom filters (bitwise OR operation)

GOSSIP

- p2p
- Enables seamless nodes addition.
- Rebalancing of keys
- Fast detection of nodes that goes down.
- Every node knows about all others - no master.

DELETIONS

The problem

1. Node A goes down

2. Client sends **`DELETE x`** for data in A

3. Node B, which is a replica of A accepts the DELETE x

4. Node A goes up again

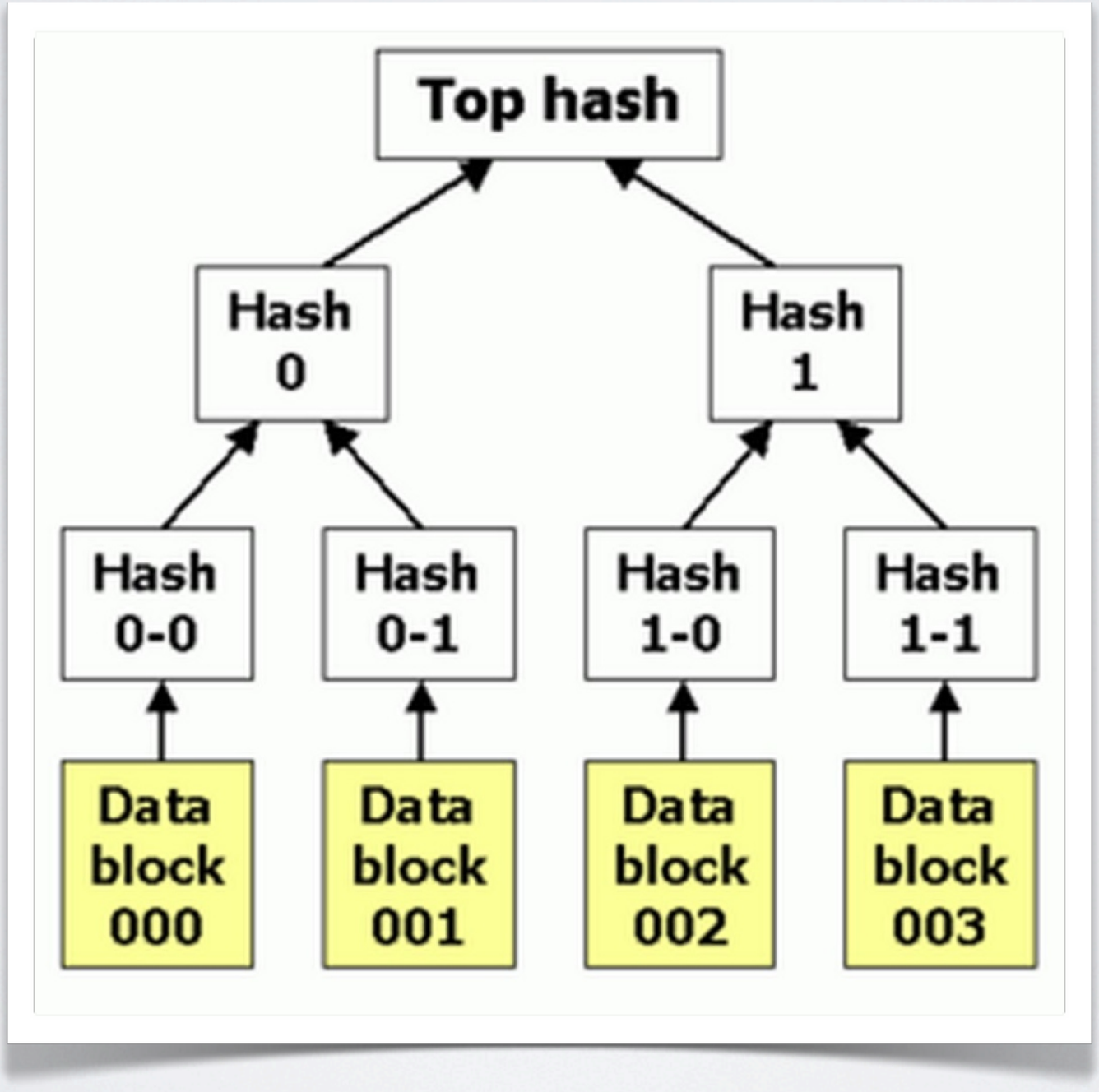
⇒A thinks B misses x so sends **`WRITE x`** to B

DELETIONS

The solution

- Deletion marker (tombstone) necessary to suppress data in older SSTables, until compaction
- Read repair complicates things a little
- Eventually consistent complicates things more
- Solution: configurable delay before tombstone GC, after which tombstones are not repaired

ANTI ENTROPY SERVICE AND MERKLE TREES



HINTED HANDOFF

REFERENCES

- This presentation: https://speakerdeck.com/u/rantav/p/nosql-and-cassandra-at-ibm
- http://highscalability.com/blog/2009/11/5/a-yes-for-a-nosql-taxonomy.html
- http://en.wikipedia.org/wiki/NoSQL#Taxonomy
- http://en.wikipedia.org/wiki/Apache_Cassandra
- http://www.datastax.com/dev/blog/whats-new-in-cassandra-1-0-performance
- http://wiki.apache.org/cassandra/ClientOptions

REFERENCES

- http://horicky.blogspot.com/2009/11/nosql-patterns.html
- http://s3.amazonaws.com/AllThingsDistributed/sosp/amazon-dynamo-sosp2007.pdf
- http://labs.google.com/papers/bigtable.html
- http://bret.appspot.com/entry/how-friendfeed-uses-mysql
- http://www.julianbrowne.com/article/viewer/brewers-cap-theorem
- http://www.allthingsdistributed.com/2008/12/eventually_consistent.html
- http://wiki.apache.org/cassandra/DataModel

REFERENCES

- http://incubator.apache.org/thrift/
- http://www.eecs.harvard.edu/~mdw/papers/quals-seda.pdf