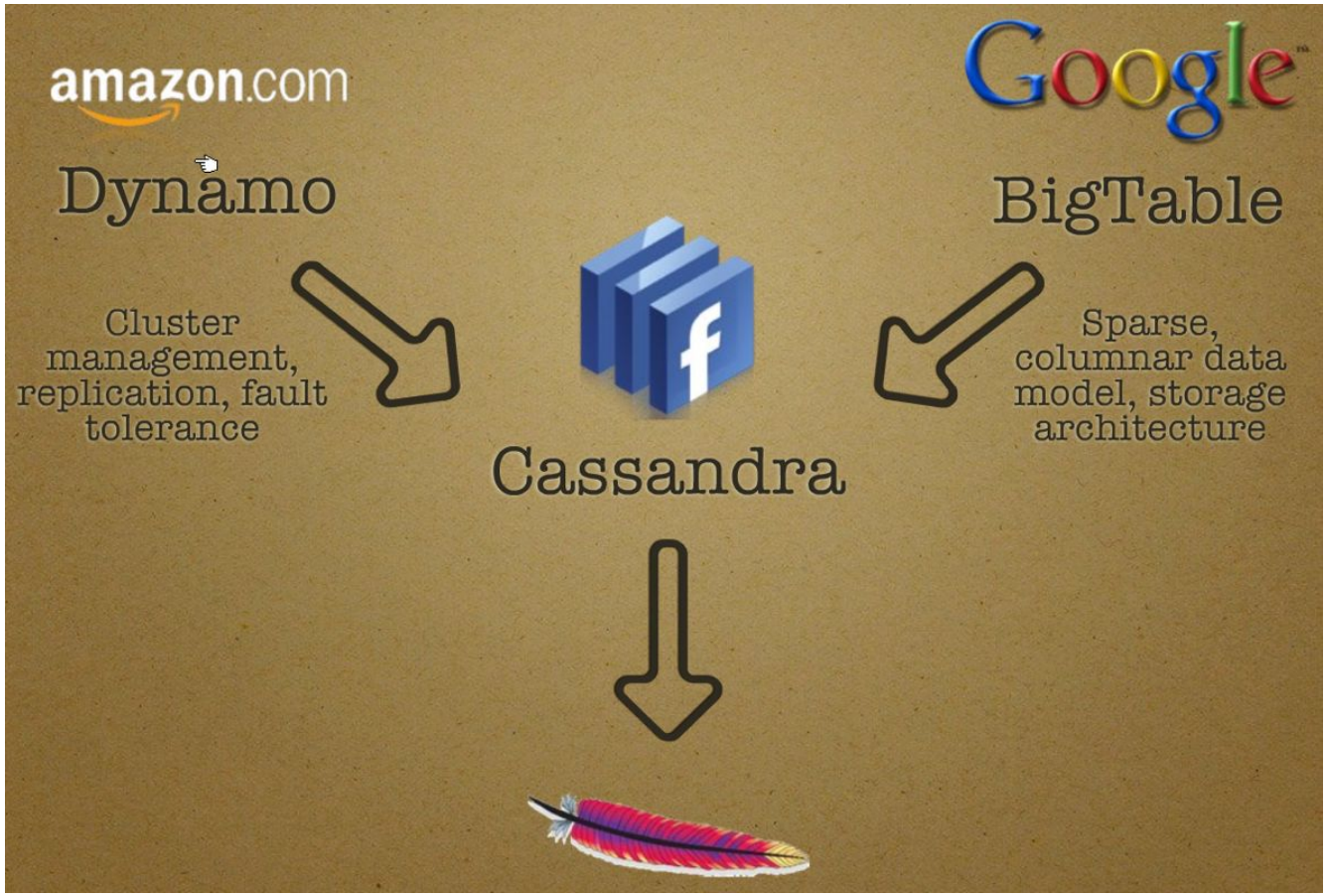
Internals

Ran
@outbrain

Required Reading ;-)

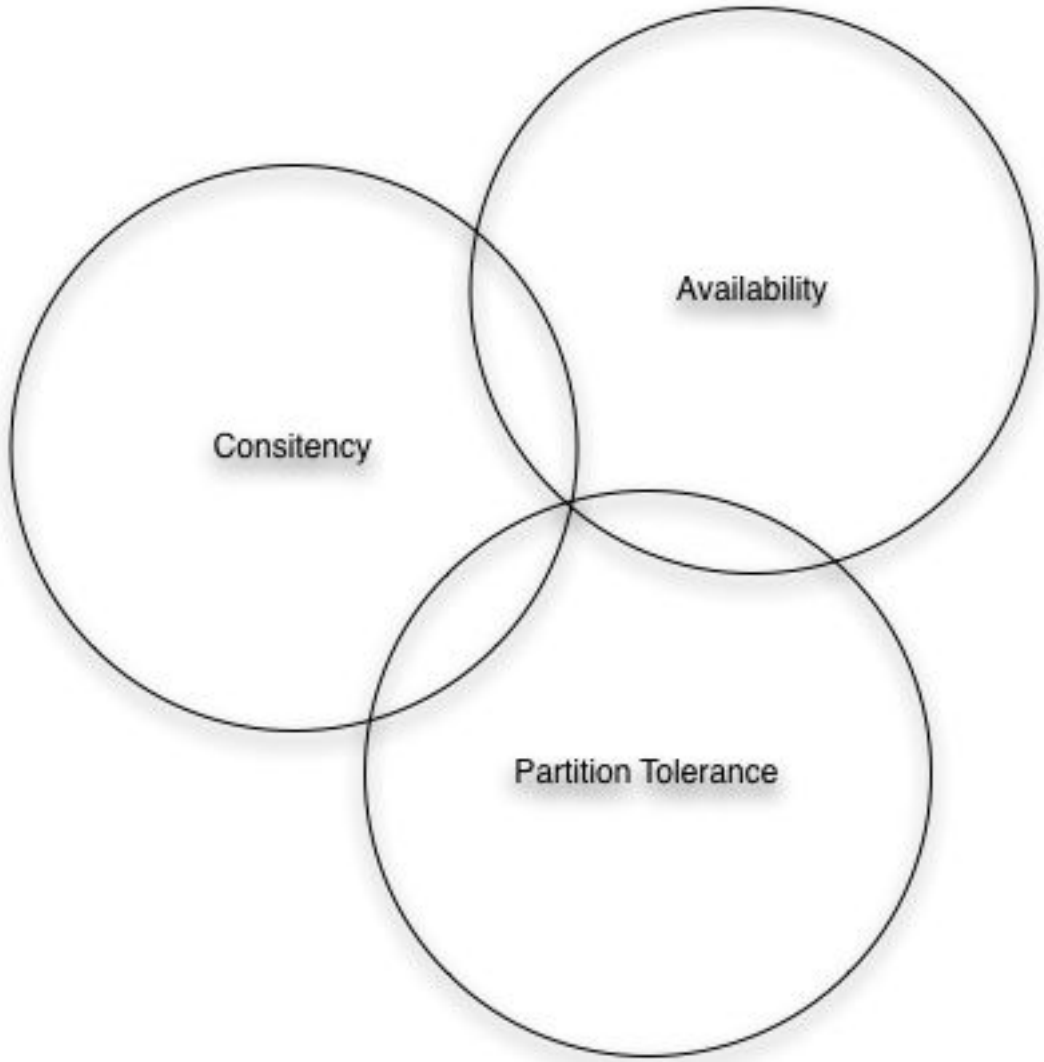
BigTable http://labs.google.com/papers/bigtable.html

Dynamo http://www.allthingsdistributed.com/2007/10/amazons_dynamo.html



Basics - CAP

You have to choose two



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)
  - Immutable sstable files
  - Compactions
  - High write performance

Architecture Layers

Cluster Management
Messaging service
Gossip
Failure detection
Cluster state
Partitioner
Replication

Single Host
Commit log
Memtable
SSTable
Indexes
Compaction

Consistency
Tombstones
Hinted handoff
Read repair
Bootstrap
Monitoring
Admin tools

Memtables

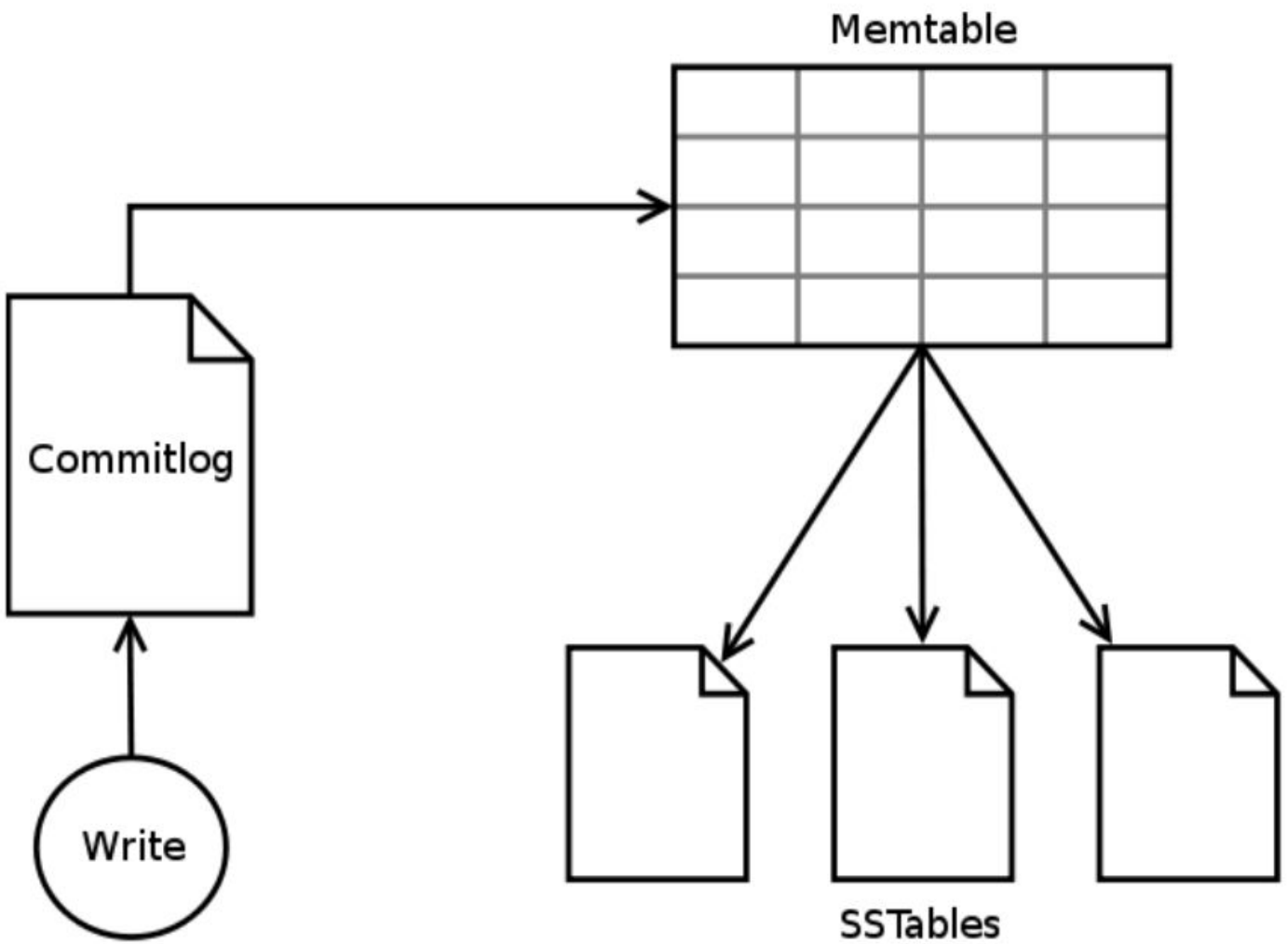
- In-memory representation of recently written data
- When the table is full, it's sorted and then flushed to disk -> sstable

SSTables

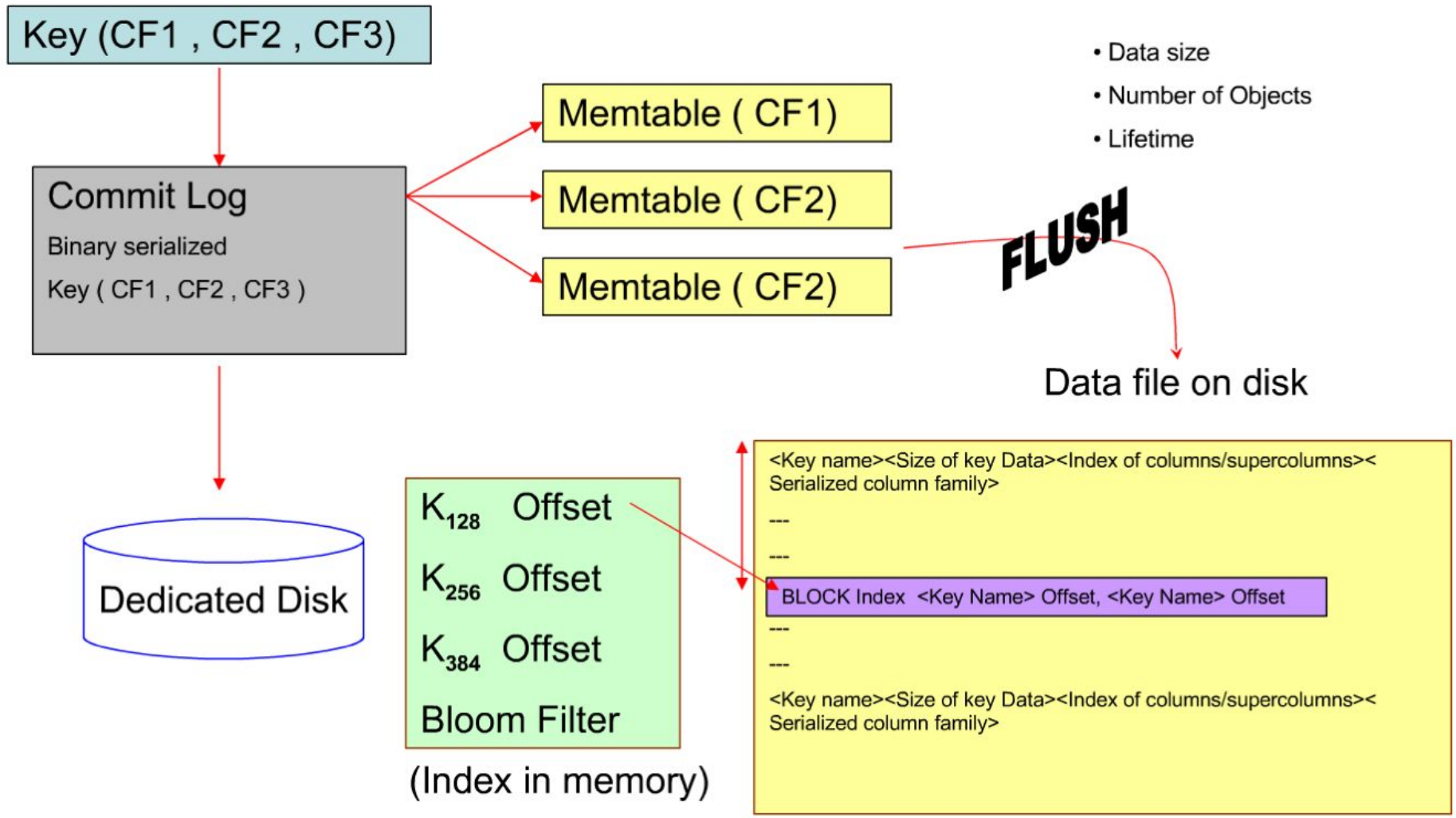
Sorted Strings Tables

- 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 Path



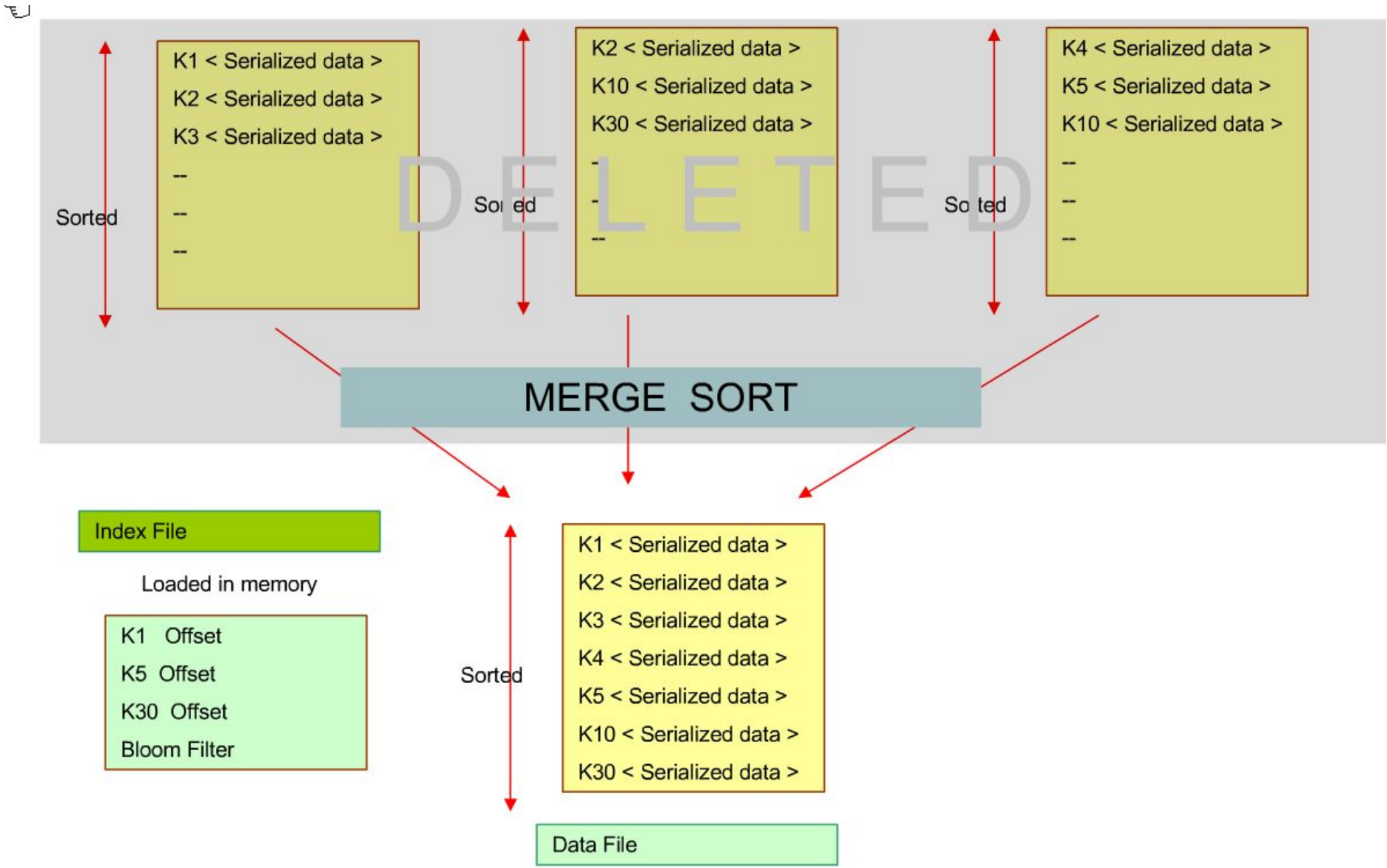
Write Path


Key (CF1 , CF2 , CF3)
Commit Log
Binary serialized
Key (CF1 , CF2 , CF3)
Memtable (CF1)
Memtable (CF2)
Memtable (CF2)
• Data size
• Number of Objects
• Lifetime
FLUSH
Data file on disk
Dedicated Disk
K128 Offset
K256 Offset
K384 Offset
Bloom Filter
(Index in memory)
<Key name><Size of key Data><Index of columns/supercolumns><
Serialized column family>

BLOCK Index <Key Name> Offset, <Key Name> Offset

<Key name><Size of key Data><Index of columns/supercolumns><
Serialized column family>

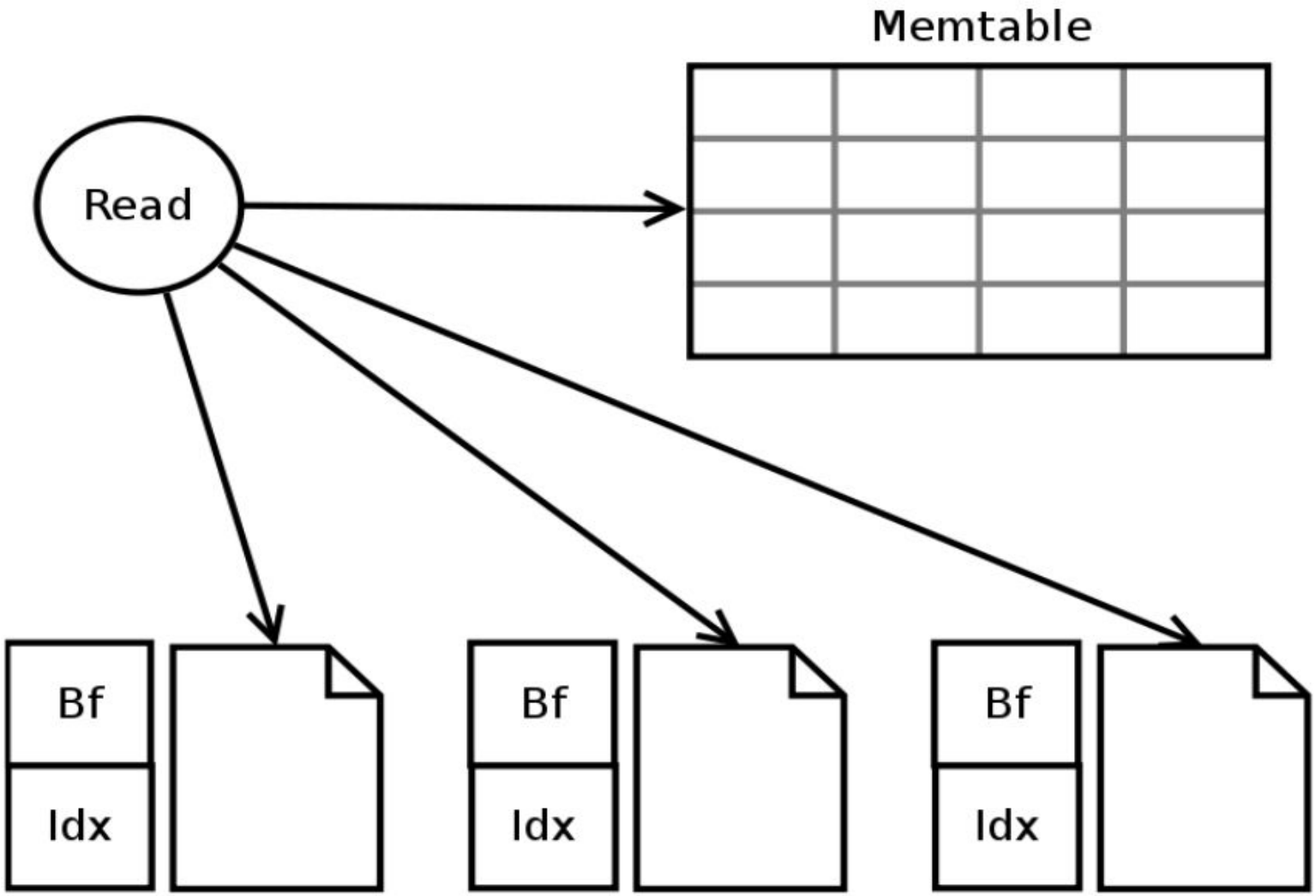
Compactions



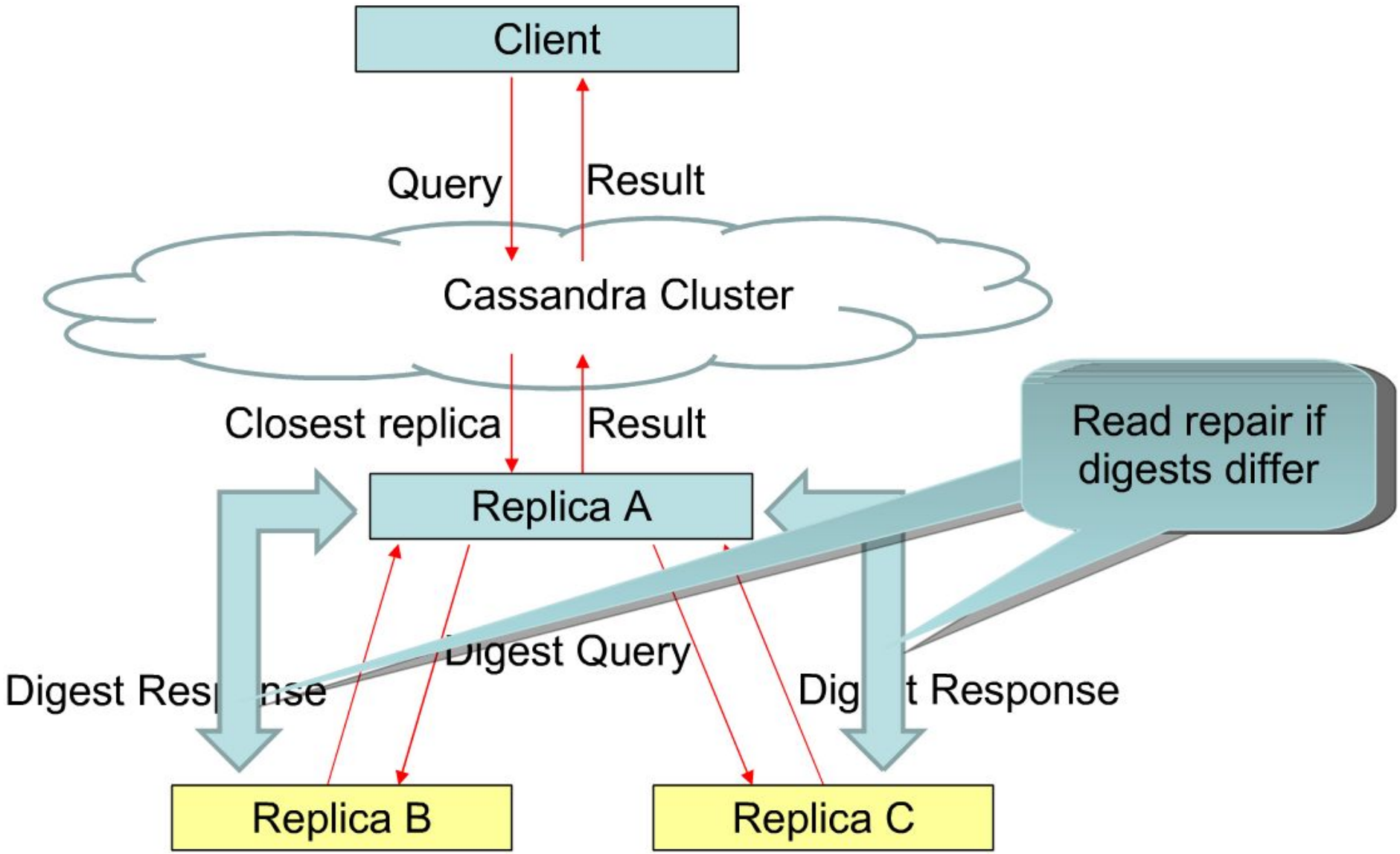
Write Properties

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

Read Path



Reads

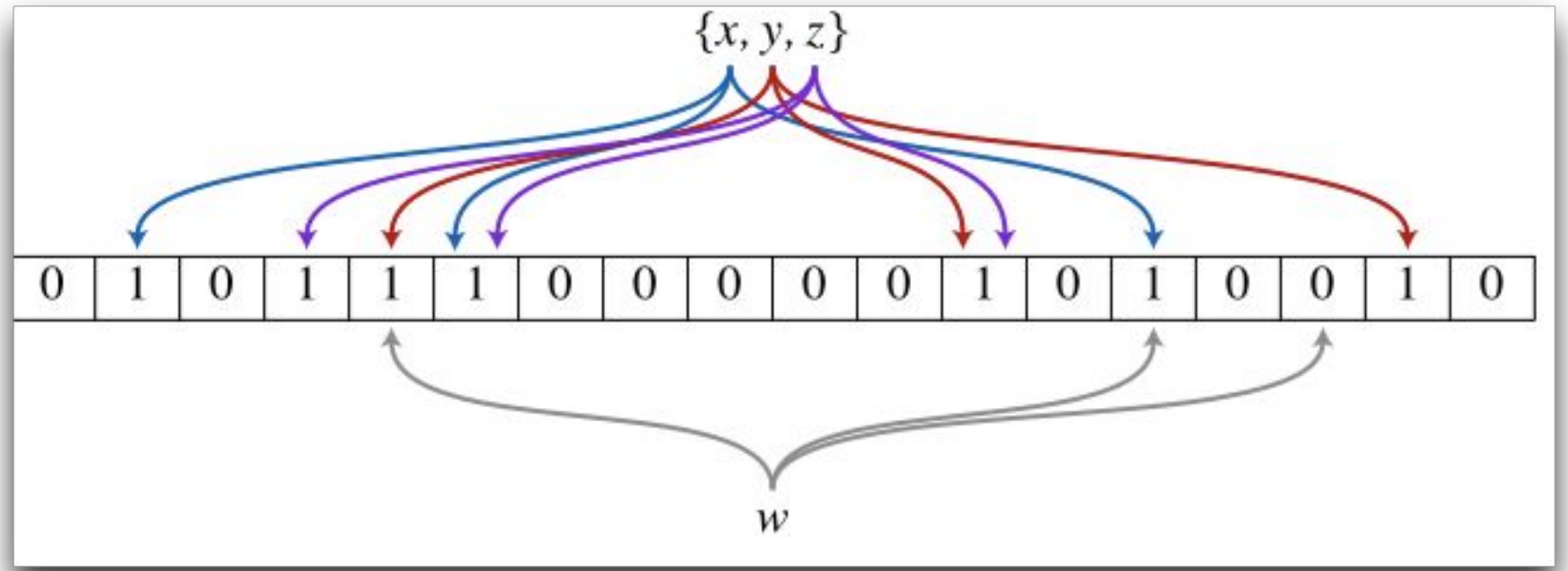

Client
Query
Result
Cassandra Cluster
Closest replica
Result
Read repair if digests differ
Replica A
Digest Query
Digest Response
Replica B
Replica C

Read Properteis

- Read multiple SSTables
- Slower than writes (but still fast)
- Seeks can be mitigated with more RAM
- Uses probabilistic bloom filters to reduce lookups.

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

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

- Large and Small compactions

Deletions

- 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

Extra Long list of subjects

commit log
SEDA
Gossip - failure detection and node discovery
anti entropy
hinted handoff
repair on read
timestamps -> vector clocks
consistent hashing
merkle trees