DESIGNING DATA-INTENSIVE APPLICATIONS

CHAPTER 3

DATABASE INDEX

LAYER 0

PEOPLE | ORDERS | ACTIONS | COMPANIES

LAYER 1

DATA STRUCTURE | OBJECTS

LAYER 2

JSON | XML | TABLES | GRAPHS

LAYER 3

BYTES IN RAM | FILES ON DISK

LAYER 4

E-CURRENTS | MAGNETIC FIELDS | LIGHT PULSES

WE HAVE 2 FAMILIES OF STORAGE ENGINES

LOG-STRUCTURED

PAGE ORIENTED

SIMPLEST DATABASE IN THE WORLD

```
#!/bin/bash

db_set () {
    echo "$1,$2" >> database
}

db_get () {
    grep "^$1," database | sed -e "s/^$1,//" | tail -n 1
}
```

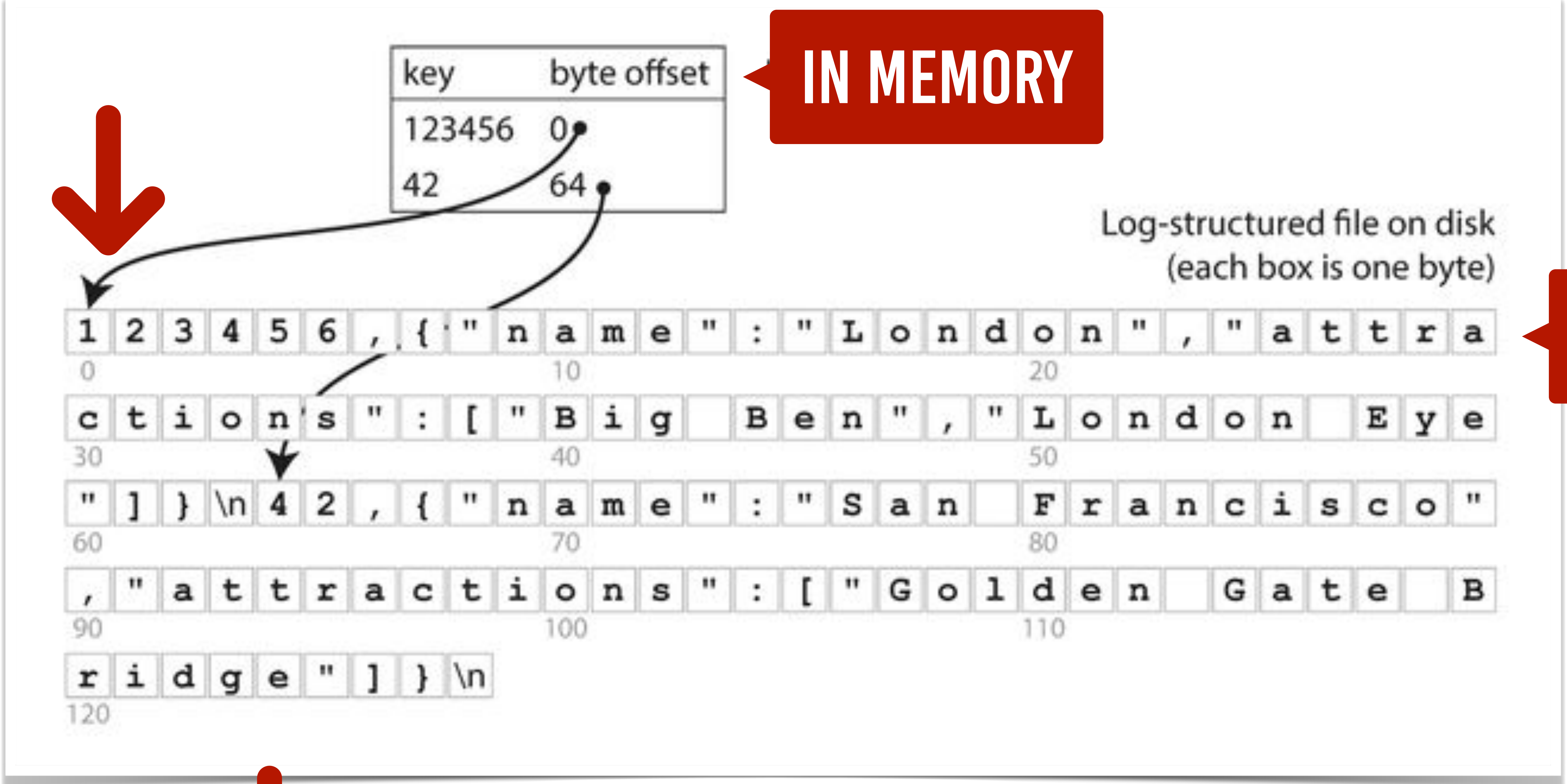
```
$ db_set 123456 '{"name":"London","attractions":["Big Ben","London Eye"]}'

$ db_set 42 '{"name":"San Francisco","attractions":["Golden Gate Bridge"]}'

$ db_get 42
{"name":"San Francisco","attractions":["Golden Gate Bridge"]}
```

WHAT IS INDEX?

HASH INDEX

- WHAT IS HASH TABLE ??
- WHAT IS HASH FUNCTION ??

AHMED → HASH FUNCTION → ARRAY INDEX



IN MEMORY

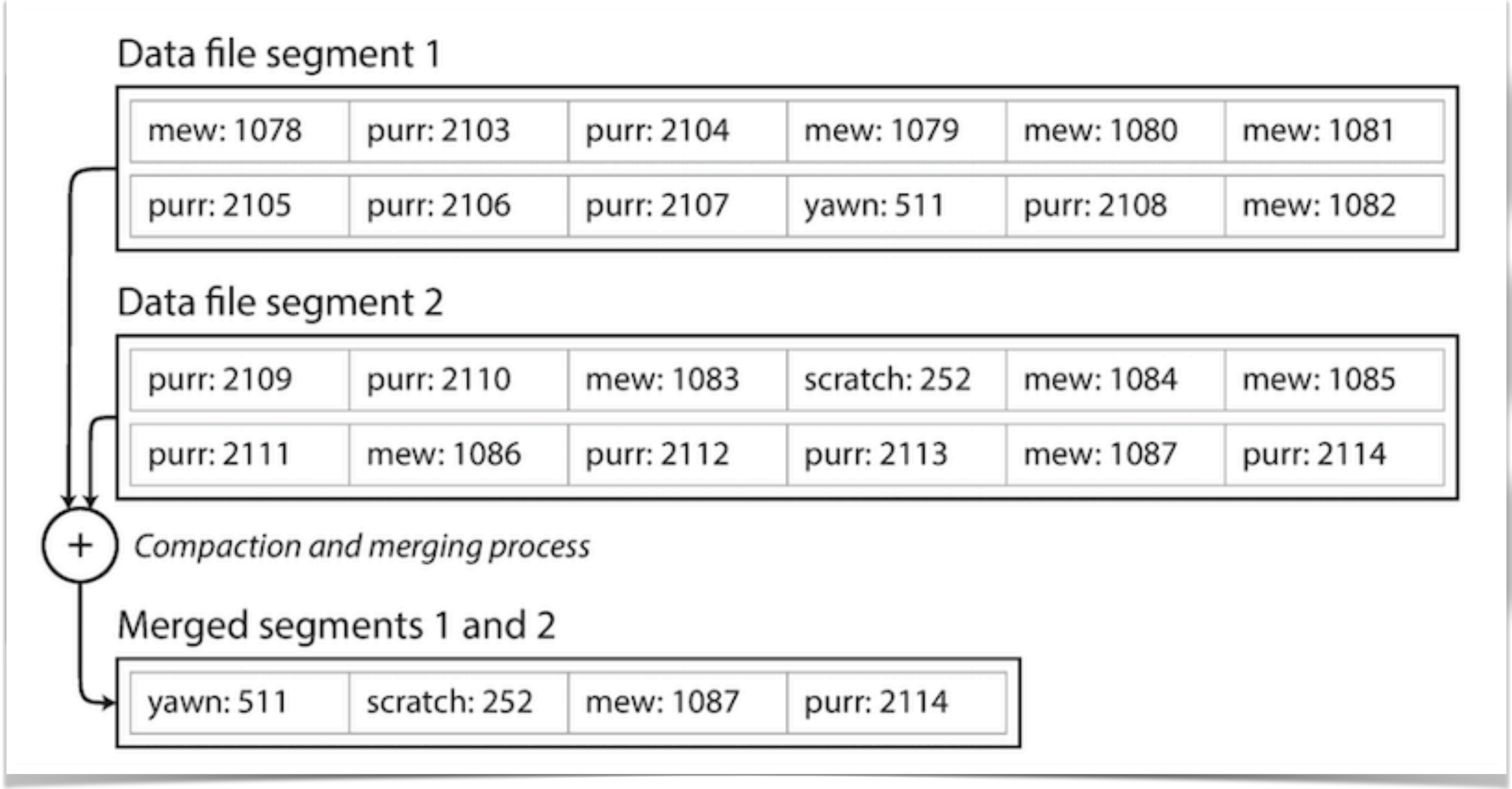
ON DISK

```
$ db_set 123456 '{"name":"London","attractions":["Big Ben","London Eye"]}'

$ db_set 42 '{"name":"San Francisco","attractions":["Golden Gate Bridge"]}'

$ db_get 42
{"name":"San Francisco","attractions":["Golden Gate Bridge"]}
```

BITCASK

RIAK DB

WHAT ARE THE ISSUES IN THIS WAY?



DISK OUT OF SPACE

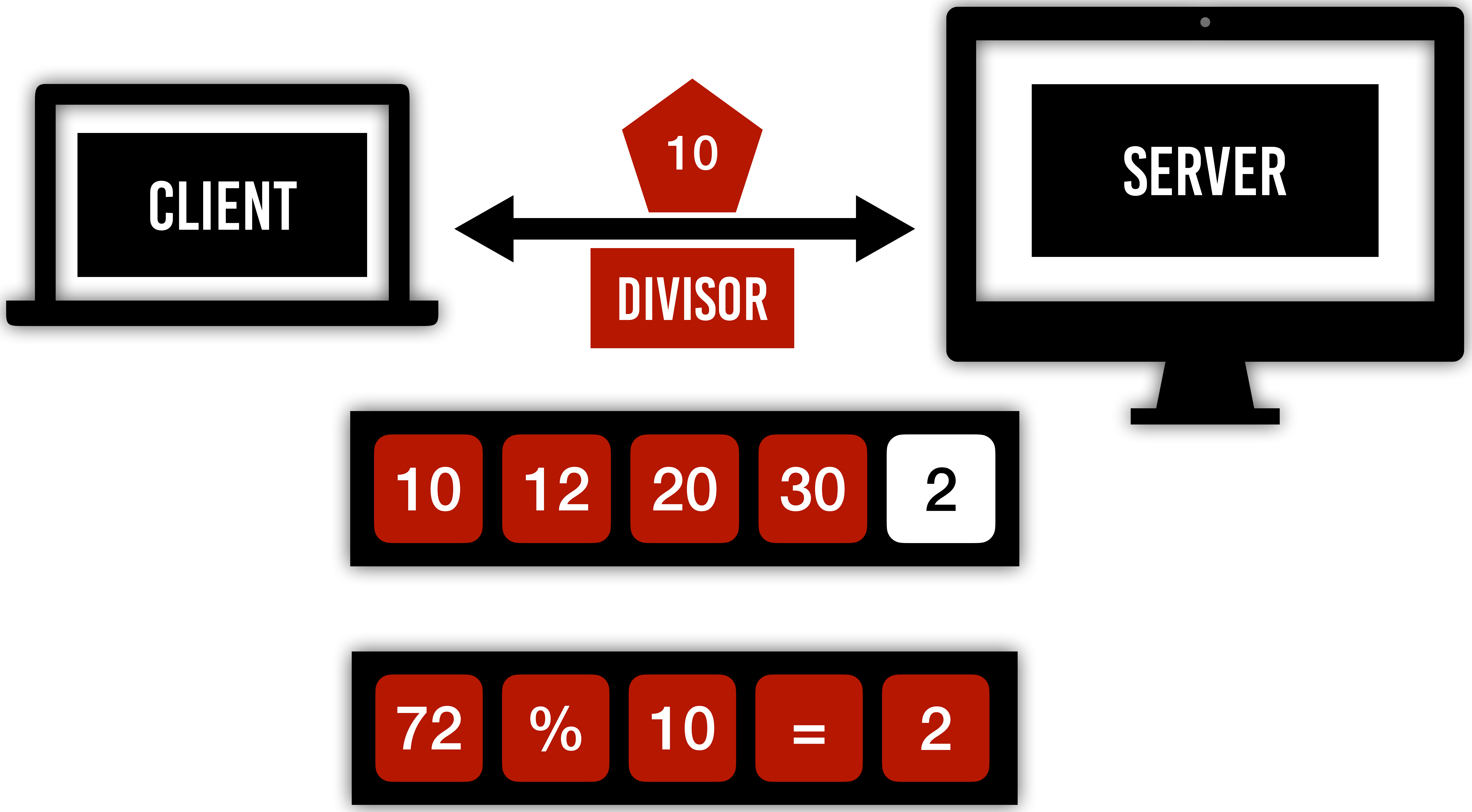
SEGMENTS

COMPACTION

WE STILL HAVE SOME ISSUES

- BINARY
- TOMBSTONE
- SNAPSHOT
- ONE WRITE THREAD
- CHECKSUM

WHAT IS THE CHECKSUM?!



PROS OF APPEND LOG WAY

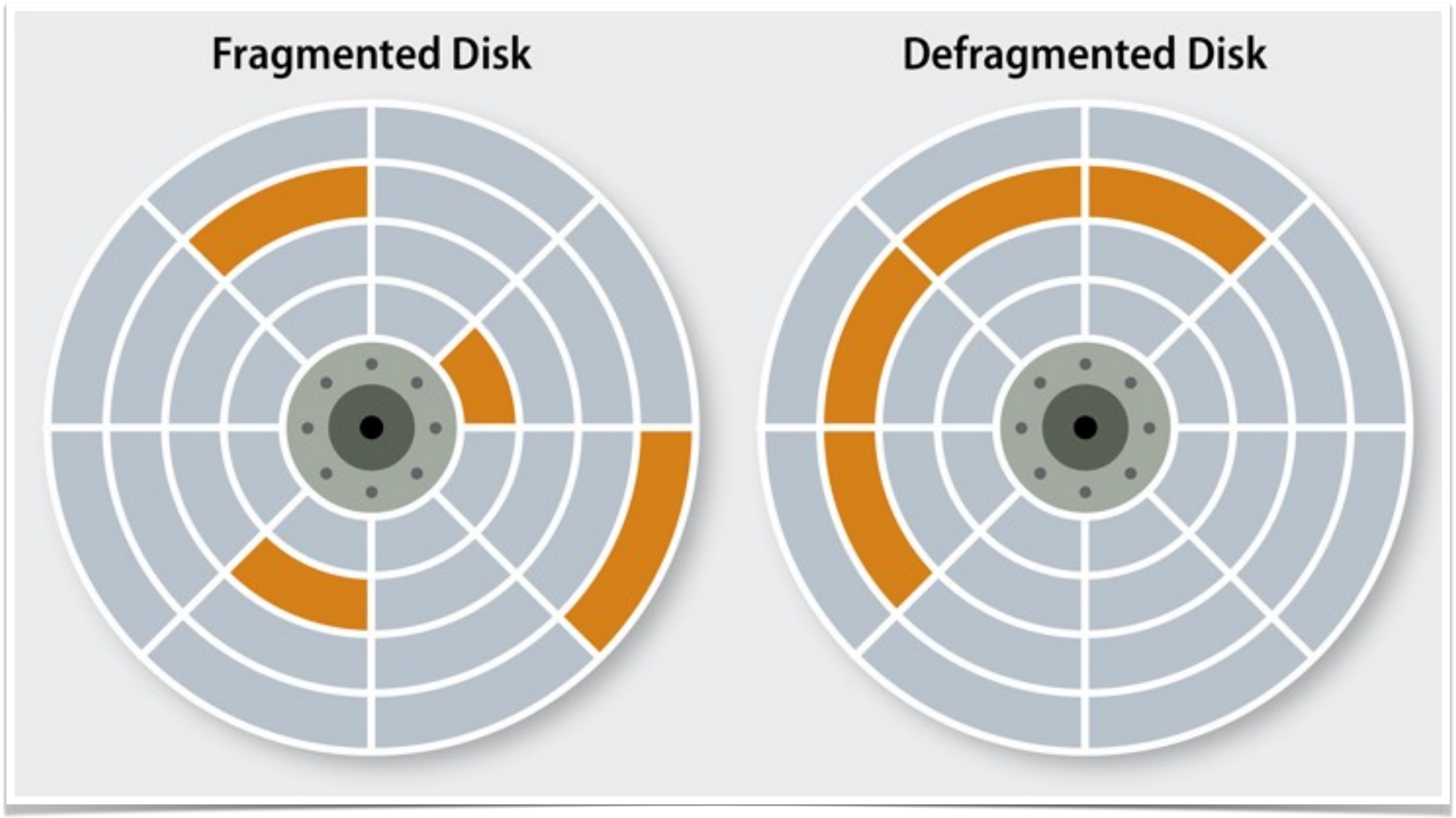
FASTER

CRASH RECOVERY

CONCURRENCY CONTROLL

NO DATA FRAGMENTATION

WHAT IS DATA FRAGMENTATION?

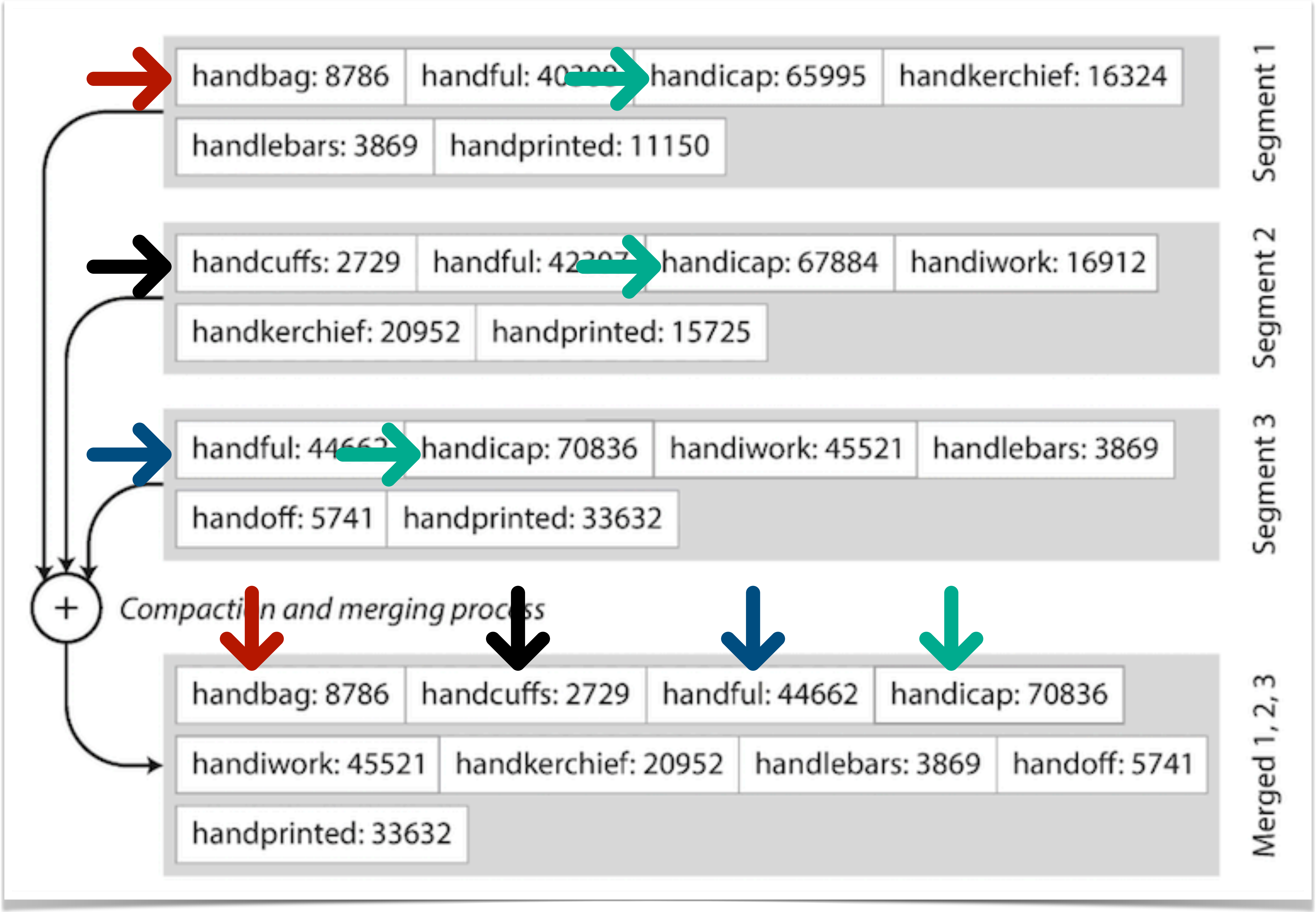


LIMITATIONS OF APPEND LOG WAY

MEMORY ISSUE

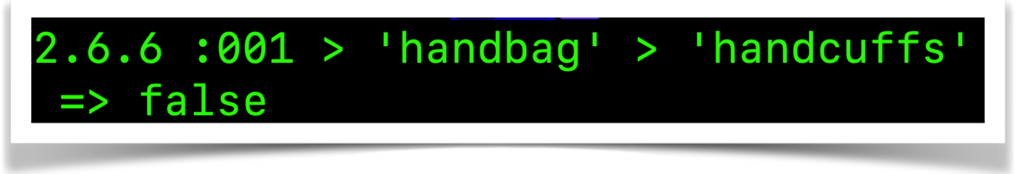
RANGE QUERY

SSTABLES "SORTED STRING TABLES"

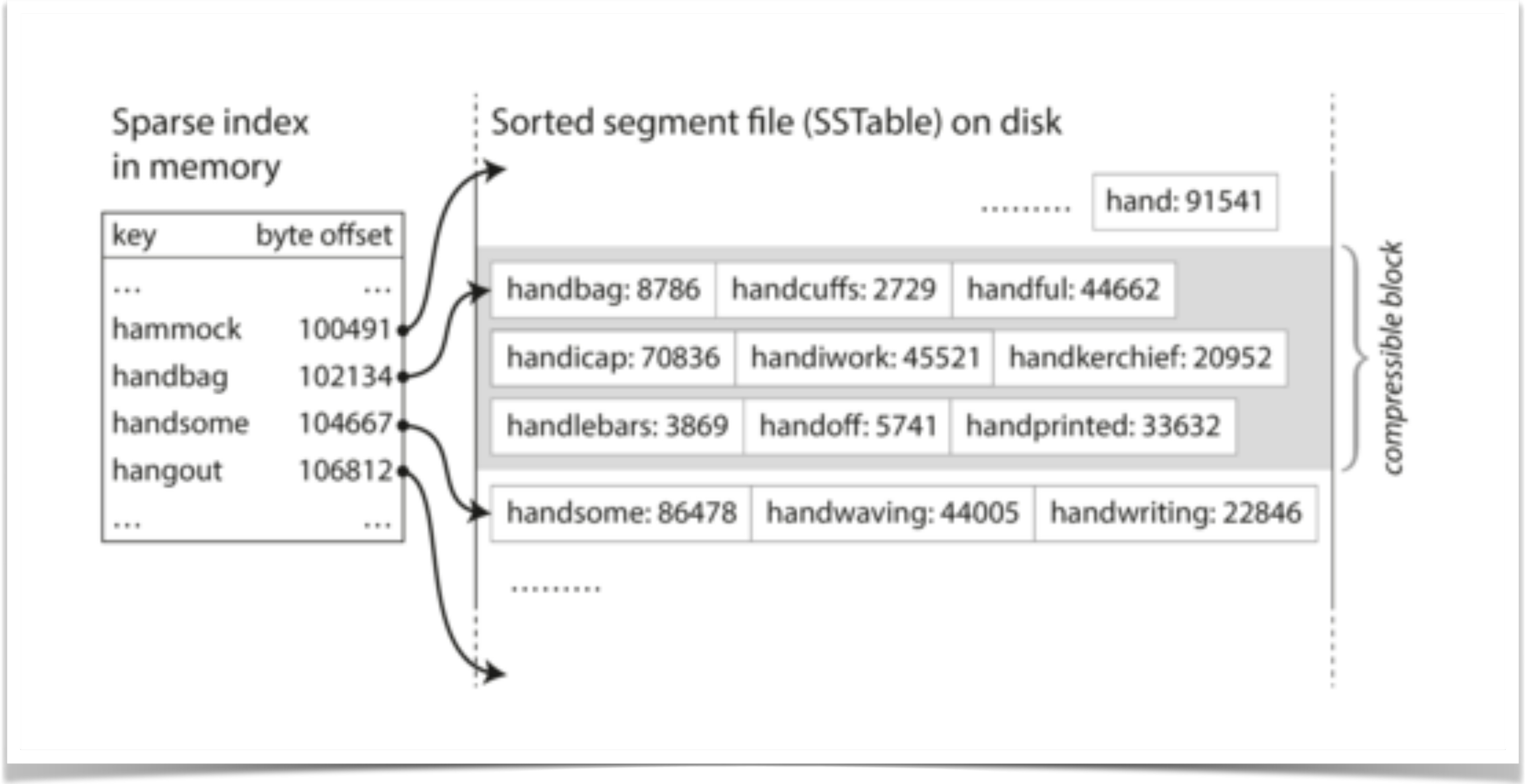


KEYS SORTED

KEYS UNIQUE AFTER MERGE



WHAT IS THE BENEFITS FROM THE SORTING?

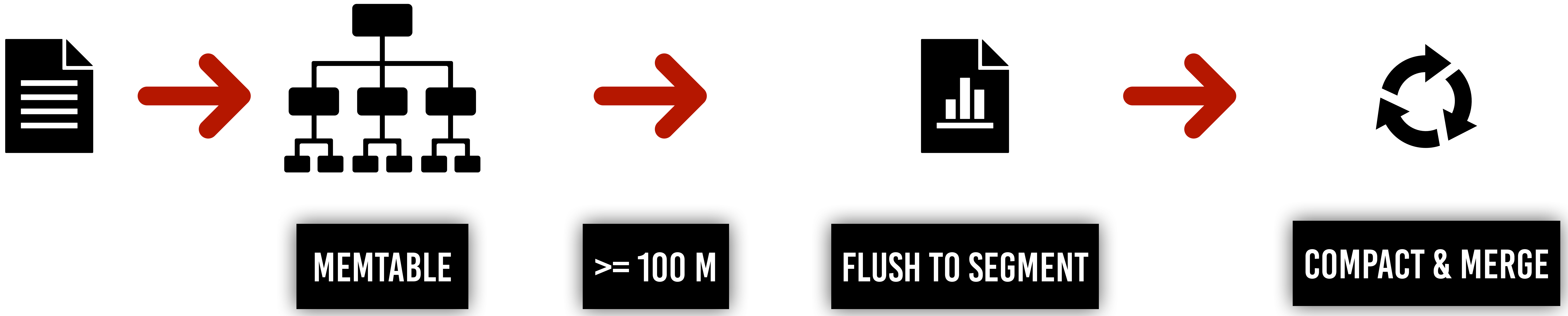


SEGMENT 1

SEGMENT 2

MERGED SEGMENT

SSTABLES FLOW



LUCENE INDEX IN APACHE SOLR

<indexConfig> in solrconfig.xml

The `<indexConfig>` section of `solrconfig.xml` defines low-level behavior of the Lucene index writers.

By default, the settings are commented out in the sample `solrconfig.xml` included with Solr, which means the defaults are used. In most cases, the defaults are fine.

```
<indexConfig>
  ...
</indexConfig>
```

Writing New Segments

The following elements can be defined under the `<indexConfig>` element and define when new segments are written ("flushed") to disk.

ramBufferSizeMB

Once accumulated document updates exceed this much memory space (defined in megabytes), then the pending updates are flushed. This can also create new segments or trigger a merge. Using this setting is generally preferable to `maxBufferedDocs`. If both `maxBufferedDocs` and `ramBufferSizeMB` are set in `solrconfig.xml`, then a flush will occur when either limit is reached. The default is `100` MB.

```
<ramBufferSizeMB>100</ramBufferSizeMB>
```

maxBufferedDocs

Sets the number of document updates to buffer in memory before they are flushed as a new segment. This may also trigger a merge. The default Solr configuration sets to flush by RAM usage (`ramBufferSizeMB`).

```
<maxBufferedDocs>1000</maxBufferedDocs>
```

Merging Index Segments

The following settings define when segments are merged.

mergePolicyFactory

Defines how merging segments is done.

The default in Solr is to use `TieredMergePolicy`, which merges segments of approximately equal size, subject to an allowed number of segments per tier.

Other policies available are the `LogByteSizeMergePolicy` and `LogDocMergePolicy`. For more information on these policies, please see the MergePolicy javadocs.

```
<mergePolicyFactory class="org.apache.solr.index.TieredMergePolicyFactory">
  <int name="maxMergeAtOnce">10</int>
  <int name="segmentsPerTier">10</int>
  <double name="forceMergeDeletesPctAllowed">10.0</double>
  <double name="deletesPctAllowed">33.0</double>
</mergePolicyFactory>
```

Controlling Segment Sizes

The most common adjustment users make to the configuration of `TieredMergePolicy` (or `LogByteSizeMergePolicy`) are the "merge factors" to change how many segments should be merged at one time and, in the `TieredMergePolicy` case, the maximum size of an merged segment.

For `TieredMergePolicy`, this is controlled by setting the `maxMergeAtOnce` (default `10`), `segmentsPerTier` (default `10`) and `maxMergedSegmentMB` (default `5000`) options.

`LogByteSizeMergePolicy` has a single `mergeFactor` option (default `10`).

To understand why these options are important, consider what happens when an update is made to an index using `LogByteSizeMergePolicy`: Documents are always added to the most recently opened segment. When a segment fills up, a new segment is created and subsequent updates are placed there.

If creating a new segment would cause the number of lowest-level segments to exceed the `mergeFactor` value, then all those segments are merged together to form a single large segment. Thus, if the merge factor is `10`, each merge results in the creation of a single segment that is roughly ten times larger than each of its ten constituents. When there are 10 of these larger segments, then they in turn are merged into an even larger single segment. This process can continue indefinitely.