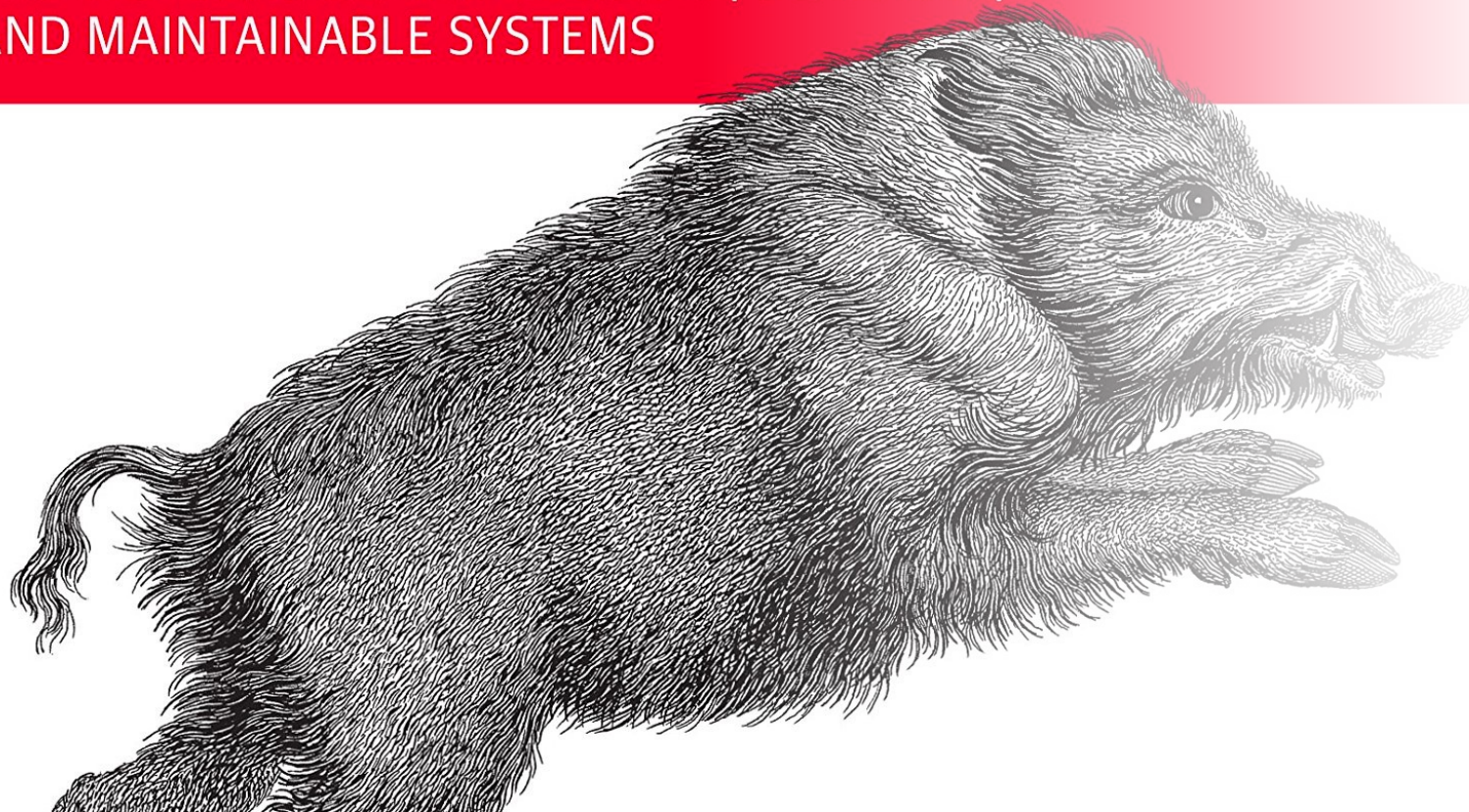
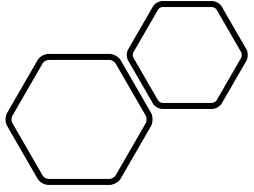


Data-Intensive — Applications

THE BIG IDEAS BEHIND RELIABLE, SCALABLE,
AND MAINTAINABLE SYSTEMS



Chapter 3 : Storage and Retrieval



What does a database do?



STORE SOME DATA



GIVE DATA WHEN WE
REQUEST FOR IT

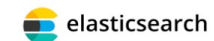
Storage Engines

- Log-structured
 - LSM Trees
 - SSTables
- Page-oriented
 - B-Trees

B-TREE



LSM TREE



Text file as a DB

```
db_set(){  
  echo "$1,$2" >>database  
}  
db_get(){  
  grep "^$1," database | sed -e  
  "s/^1,/" | tail -n 1  
}
```

```
$ db_set 123 'John'  
$ db_set 267 'Katie'  
$ db_get 267  
Katie  
$ db_set 267 'Bill'  
$ db_get 267  
Bill
```

```
$ cat database  
123, John  
267, Katie  
267, Bill
```

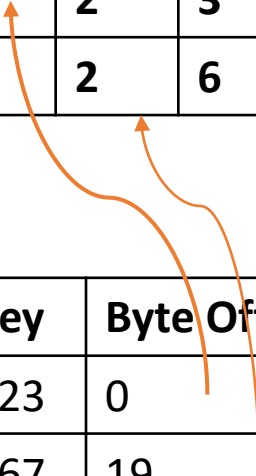
Index

- Additional structure derived from primary data.
- Keep metadata which helps to locate the data.
- Searching in different ways require different indexes.
- No effect on contents of the database, affects performance of queries.
- Slows down write queries, index needs to be updated.

Hash Indexes

0									10				17				
1	2	3	,	J	o	h	n	\n	2	6	7	,	K	a	t	i	e
\n	2	6	7	,	B	i	l	l									
18									26								

Key	Byte Offset
123	0
267	19



Compaction

Data file segment

Mew:1078	Purr:2103	Purr:2014	Mew:1079	Mew:1080	Mew:1081
Purr:2105	Purr:2106	Purr:2107	Yawn:511	Purr:2108	Mew:1082



Compaction process

Compacted Segment

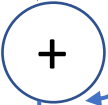
Yawn:511	Mew:1082	Purr:2018
----------	----------	-----------

Data file segment 1

Mew:1078	Purr:2103	Purr:2014	Mew:1079	Mew:1080	Mew:1081
Purr:2105	Purr:2106	Purr:2107	Yawn:511	Purr:2108	Mew:1082

Data file segment 2

Mew:1078	Purr:2103	Purr:2014	Mew:1079	Mew:1080	Mew:1081
Purr:2105	Purr:2106	Purr:2107	Yawn:511	Purr:2108	Mew:1082



Compaction and merging process

Merged segments 1 and 2

Yawn:511	Mew:1082	Purr:2018
----------	----------	-----------

Problems



FILE FORMAT



DELETING
RECORDS



CRASH RECOVERY



PARTIALLY
WRITTEN RECORDS



CONCURRENCY
CONTROL

SS Tables

- ***Sorted String Table***
- Sequence of key-value pairs is *sorted by key*.
- Each **key** appears **once** within each merged segment
- Merging segments is simple and efficient – uses merge sort and creates a merged segment file sorted by key
- No longer need to keep index of all keys in memory
- Read requests need to scan over several key value pairs in requested range, it is possible to group those records into a block.
- Each entry of sparse in memory index points to the start of the compressed block.

Data file segment 1

handbag:1078	handful:2103	handicap:2014
handkerchief:2105	handlebars:2106	handprinted:2107

Data file segment 2

handful:1078	handicap:2103	handiwork:2014
handlebars:2105	handoff:2106	handprinted:2109

Sparse index in memory

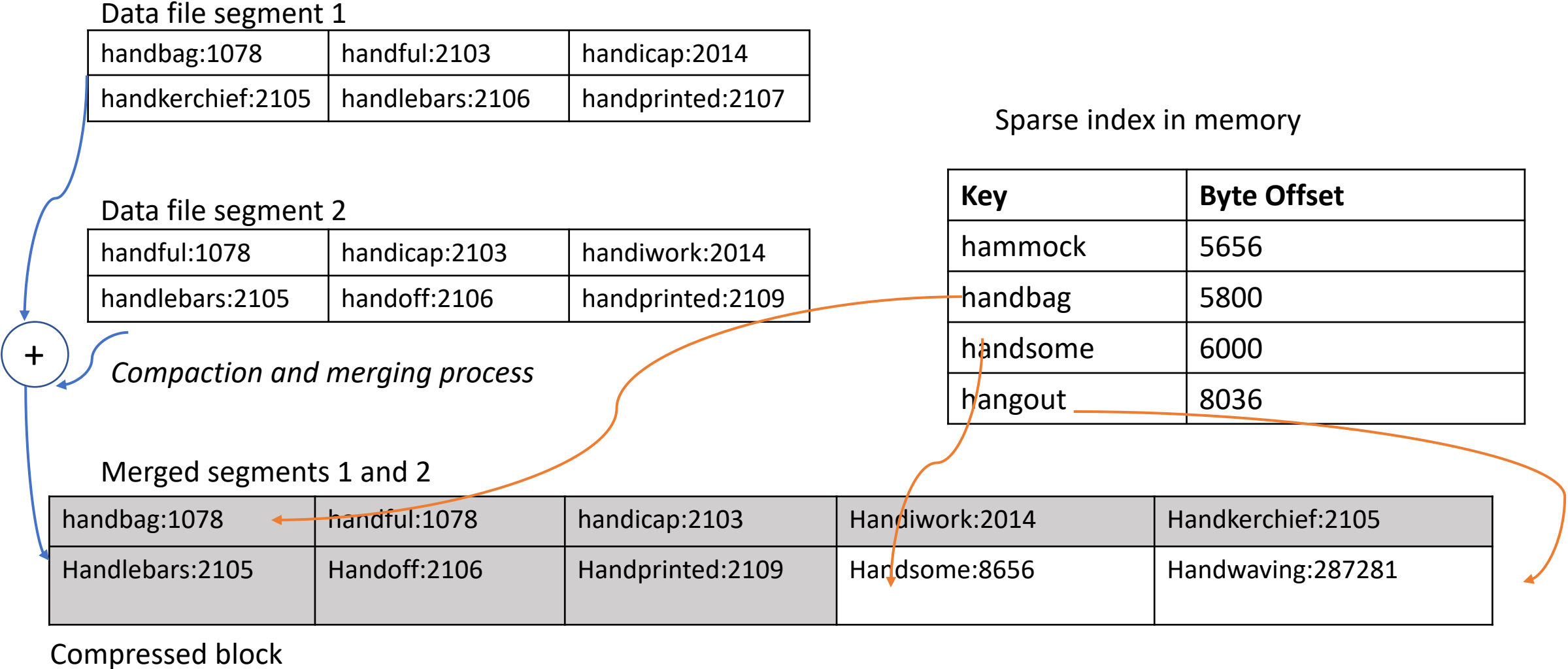
Key	Byte Offset
hammock	5656
handbag	5800
handsome	6000
hangout	8036

+
Compaction and merging process

Merged segments 1 and 2

handbag:1078	handful:1078	handicap:2103	Handiwork:2014	Handkerchief:2105
Handlebars:2105	Handoff:2106	Handprinted:2109	Handsome:8656	Handwaving:287281

Compressed block



How to sort the keys?

- Write comes in, add key to in-memory balanced tree DS. This in-memory tree is called memtable.
- When memtables crosses the threshold, flush it to disk as an SSTable file. The new SSTable file becomes the most recent segment file.
- To serve read request, first find key in memtable, then most recent on-disk segment and then next-older segment and so on..
- Run a merging and compaction process to combine segment files and discard overwritten or deleted values.
- Problem: When database crashes, most recent writes in memtables not yet written to disk are lost, we can maintain a simple log file for crash purposes.



B-Trees

- Value pairs sorted by keys, it allows range queries and efficient key-value lookups.
- Fixed size blocks vs variable segments in log structures trees.
- This corresponds to more closely the underlying hardware, as disks are also arranged in fixed size blocks.
- Each page can be identified via an address but on disk and not on memory.
- Page is found first and then updated whereas LSM is updated sequentially.
- Write operation in B-tree is to overwrite the page on disk without changing the location of the data and all its references.

Look up user id = 252

ref	100	ref	200	Ref	300	Ref	400	ref	500	ref
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

Key<100

Key >=500

100<=key<200

400<=key<500

200<=key<300

300<=key<400

ref	111	ref	135	Ref	152	Ref	169	ref	190	ref
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

ref	210	ref	230	Ref	250	Ref	270	ref	290	ref
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

250<=key<270

250	Val	251	Val	252	Val	253	Val	254	val
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

How to make B-Trees more reliable?

- Why reliability?
 - Overwrites the current page on disk
 - Database can crash at any point while updating the page references, end up with corrupted index and an orphan child.
- WAL
 - Write Ahead Log(Redo Log), append only file – writes all the B-tree modification before applying it to pages of tree itself.
 - Concurrency control is required.



B-Tree optimization

- Copy-on-write scheme (LMDB server)
- Save space in pages by not storing the entire key, but abbreviating it, leads to more keys into a page and higher branching factor with fewer levels.
- Pages can be positioned anywhere on disk, not necessary to store pages with nearby key ranges to be nearby on disk.
- Each page may have references to left and right sibling pages without jumping back to parent page.





B-Trees vs LSM-Trees

LSM Trees are **faster for writes** whereas B-trees for **reads**.

B-tree **index writes data at least twice** – WAL and tree page.

LSM – **writes multiple times** – due to repeated merging and compaction. **Write amplification** – Effect where one write to the database results in multiple writes to the disk - concern in SSDs.

LSM sustain **higher write throughput** than B-trees

LSM can be **compressed** better.

B-Trees might have some disk space unused due to **fragmentation** when a page is split whereas LSM periodically rewrite SSTables to remove fragmentation.

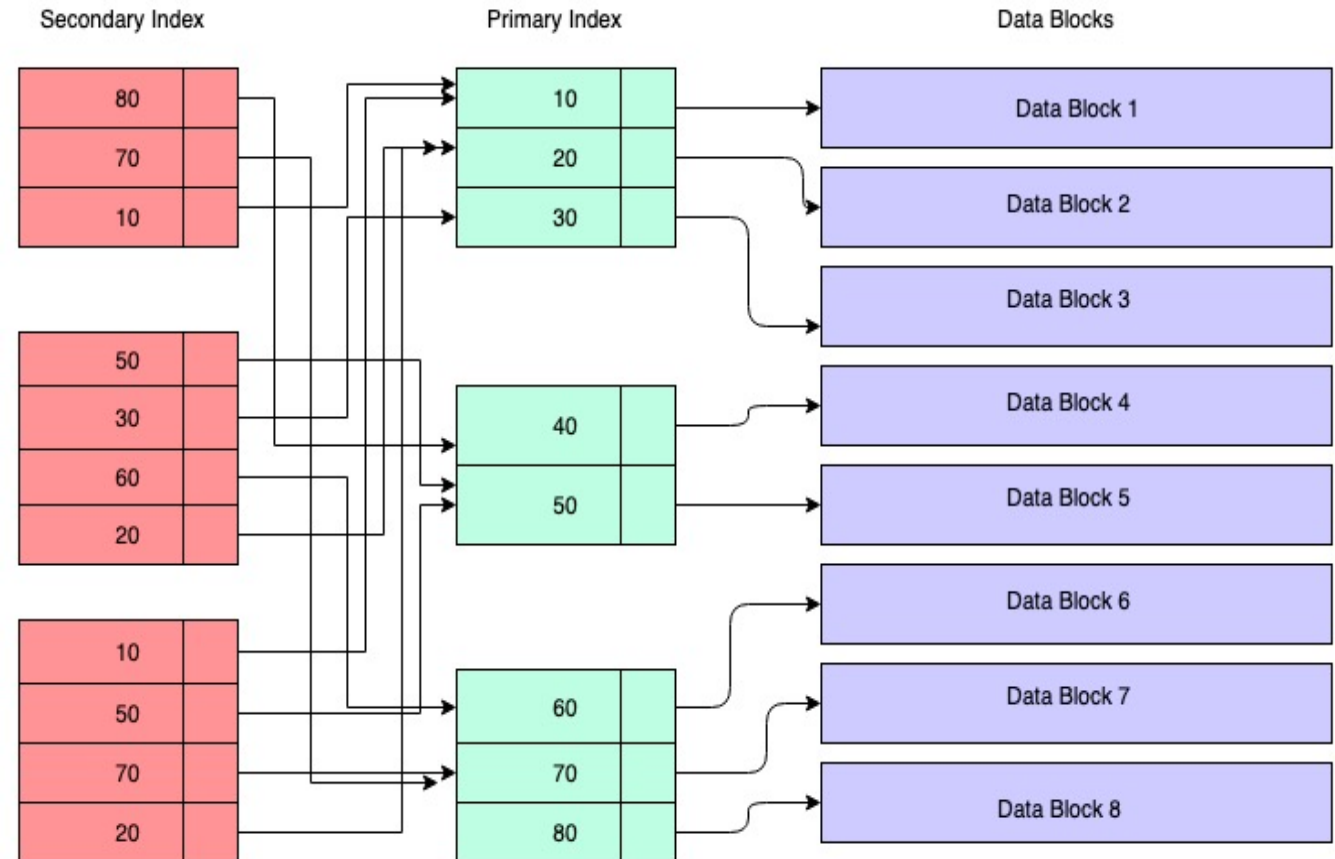
Compaction process in LSM can interfere with performance of ongoing reads and writes.



If write throughput is high and compaction is not configured properly, then number of unmerged segments increases, reads also slow down.

Key exists only once in B-tree index whereas in LSM may have multiple copies in different segments.

Other Indexing structures

- **Index** – (key,value) or (key,ref)
- **Primary Index** – uniquely identifies one row in a relational table
- **Secondary Index** – keys are not unique, there might be many rows with the same key.
- **Non clustering index** – storing only references to the data within the index



- 
- **Heap file** – where key,value are stored.
 - **Clustering index** – when hop from index to heap file is performance overhead, then leaf nodes are the actual data blocks on disk, since the index & data reside together. This kind of index physically organizes the data on disk as per the logical order of the index key. Primary key is always a clustered index.
 - **Covering index** – Compromise between clustering and non clustering index.
 - **Multi-column indexes**
- 

Summary



Log Structured Storage Engines



Hash indexes and SS Tables



Page Oriented Storage Engines



B-Trees



B-Trees vs LSM Trees



Other indexing structures



Thank You!