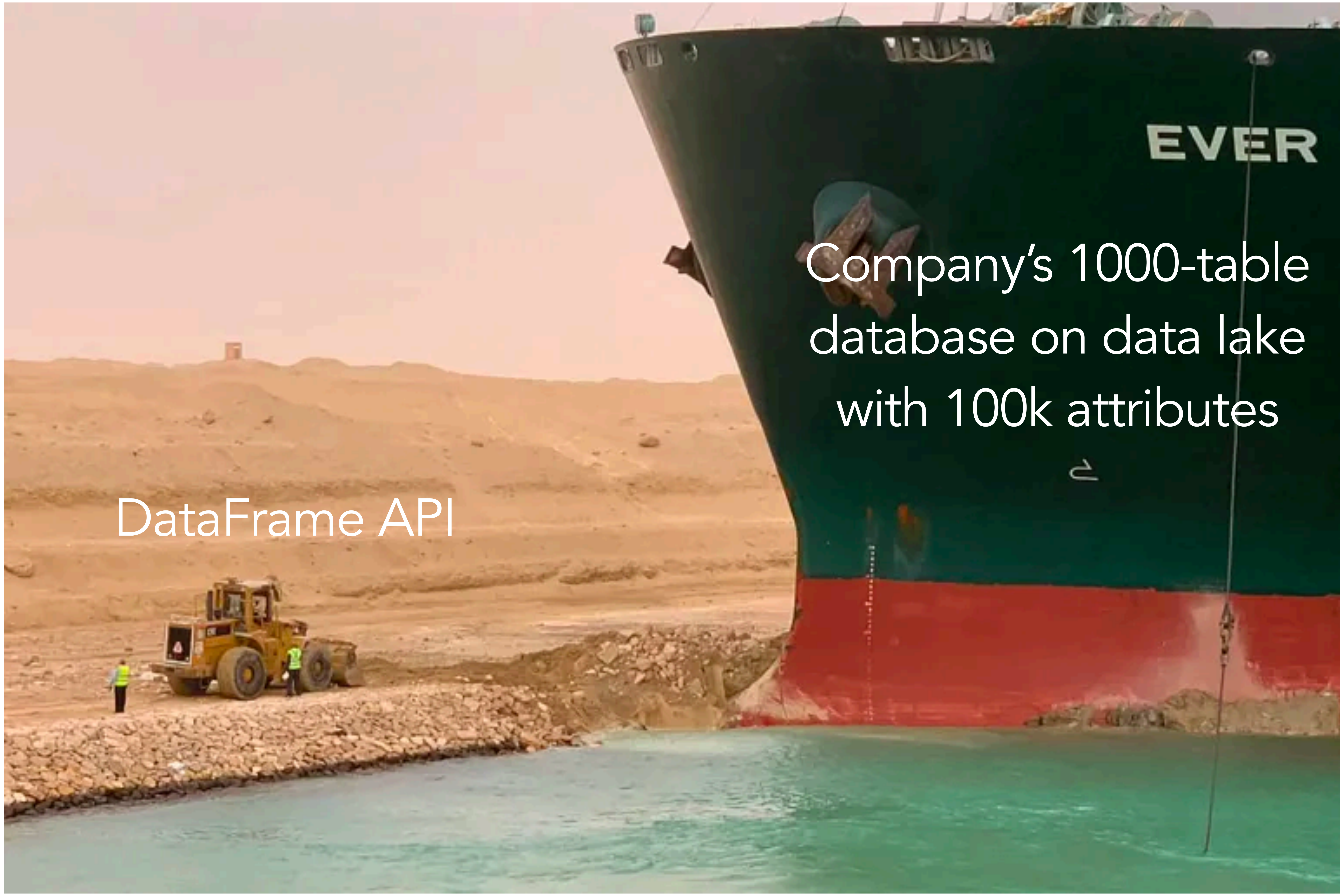
DSC 204a Scalable Data Systems

- Haojian Jin



Meme idea credit: https://datasystemsfun.tumblr.com/

Where are we in the class?

Foundations of Data Systems (2 weeks)

- Digital representation of Data → Computer Organization → Memory hierarchy → Process → Storage

Scaling Distributed Systems (3 weeks)

- Cloud → Network → **Distributed storage** → Partition and replication (HDFS) → Distributed computation

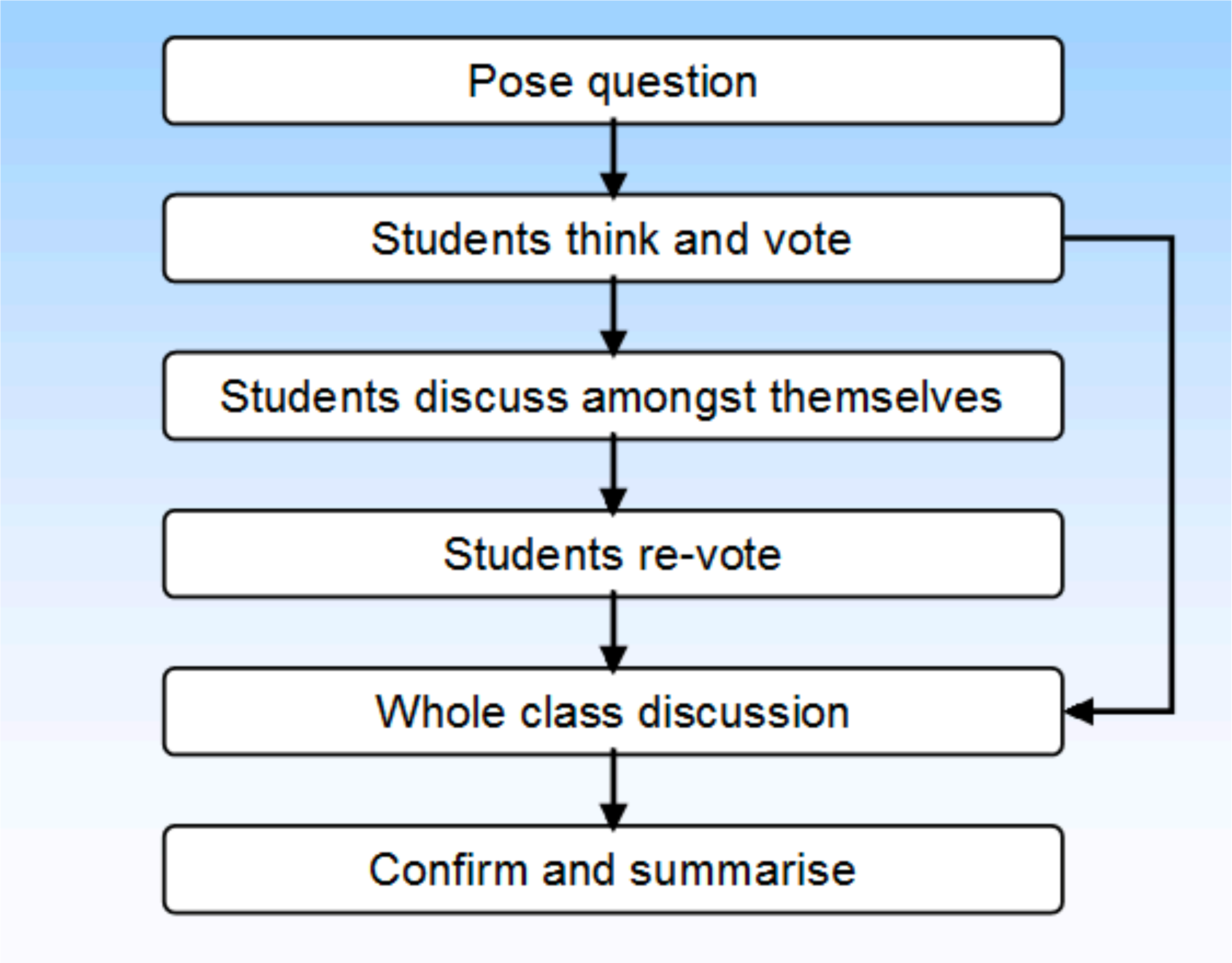
Data Processing and Programming model (5 weeks)

- Data Models evolution → Data encoding evolution → → IO & Unix Pipes → Batch processing (MapReduce) → Stream processing (Spark)

Today's activities

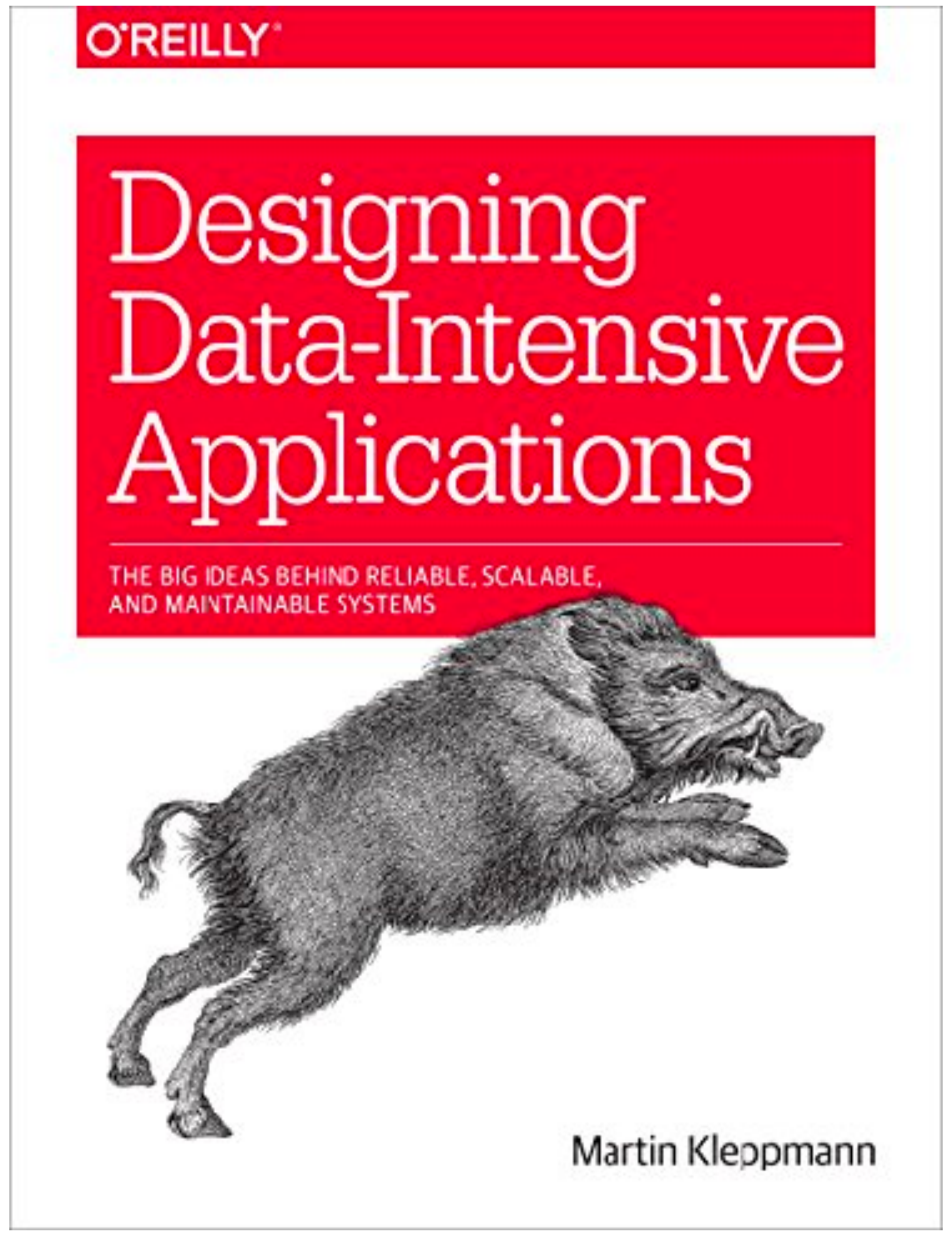
- PIA.
- Chapter 3 Storage and Retrival
  - Hash Indexes
  - SSTables and LSM-Trees
  - B-Trees
  - Other indexing structures

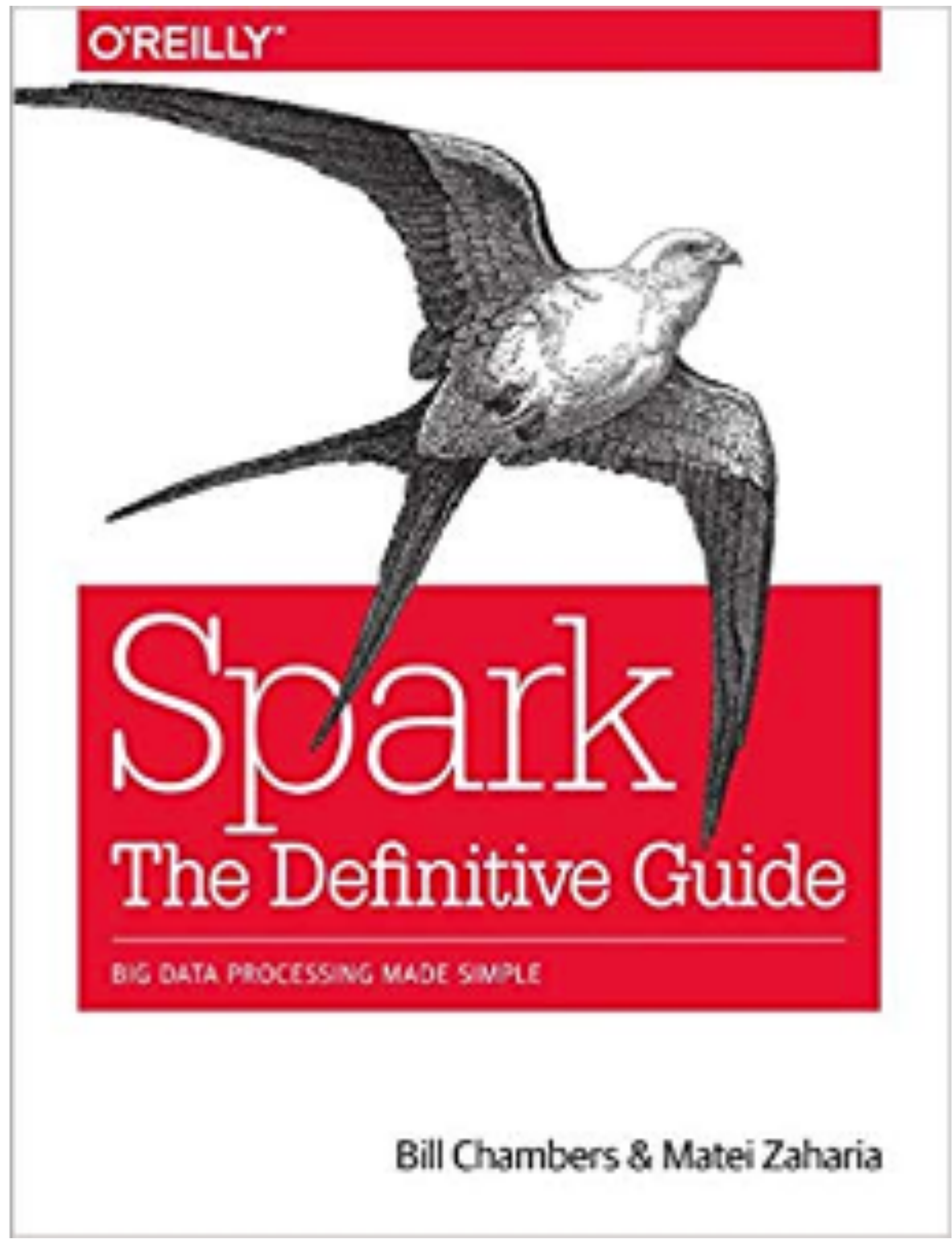
Peer instruction activity

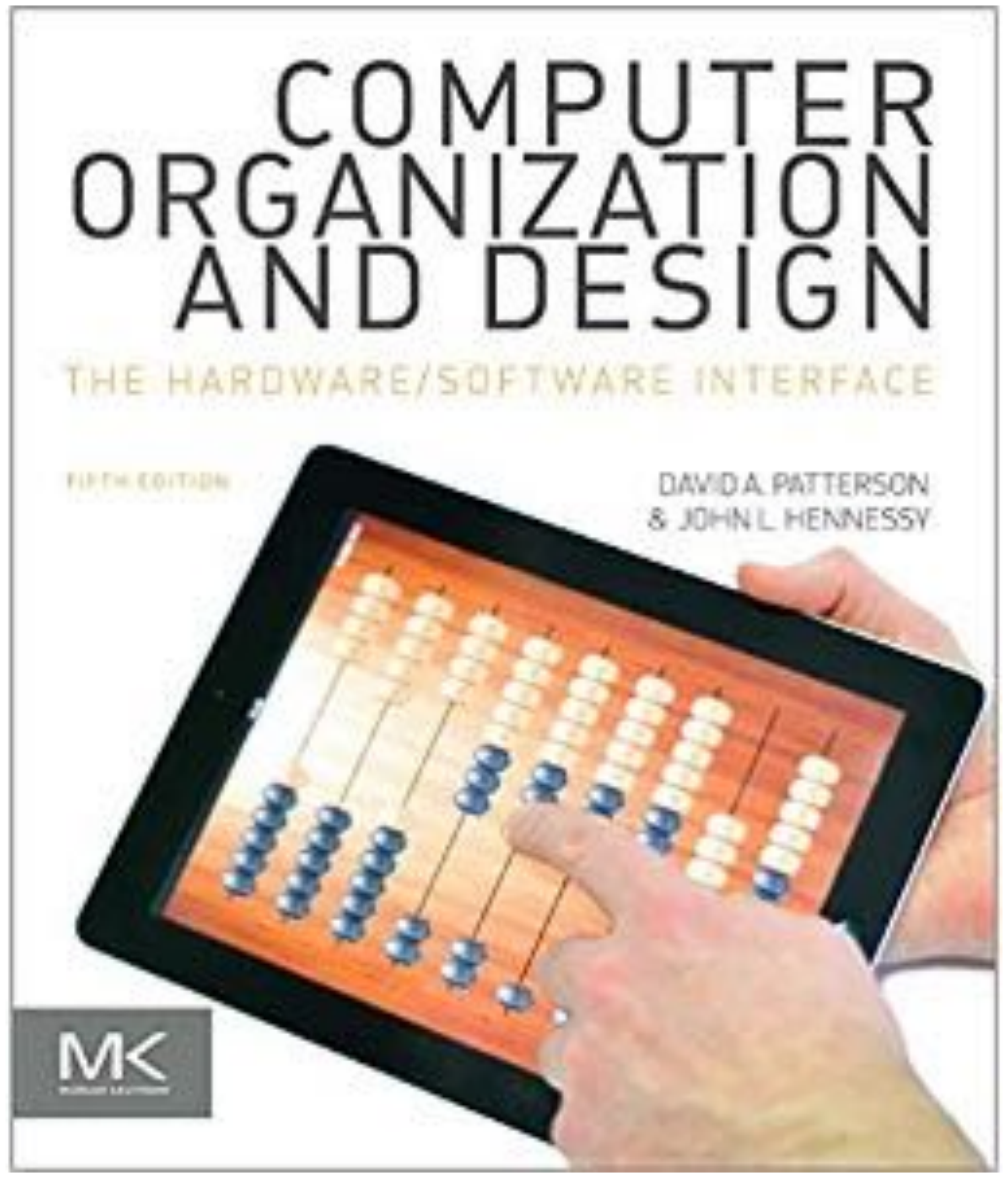


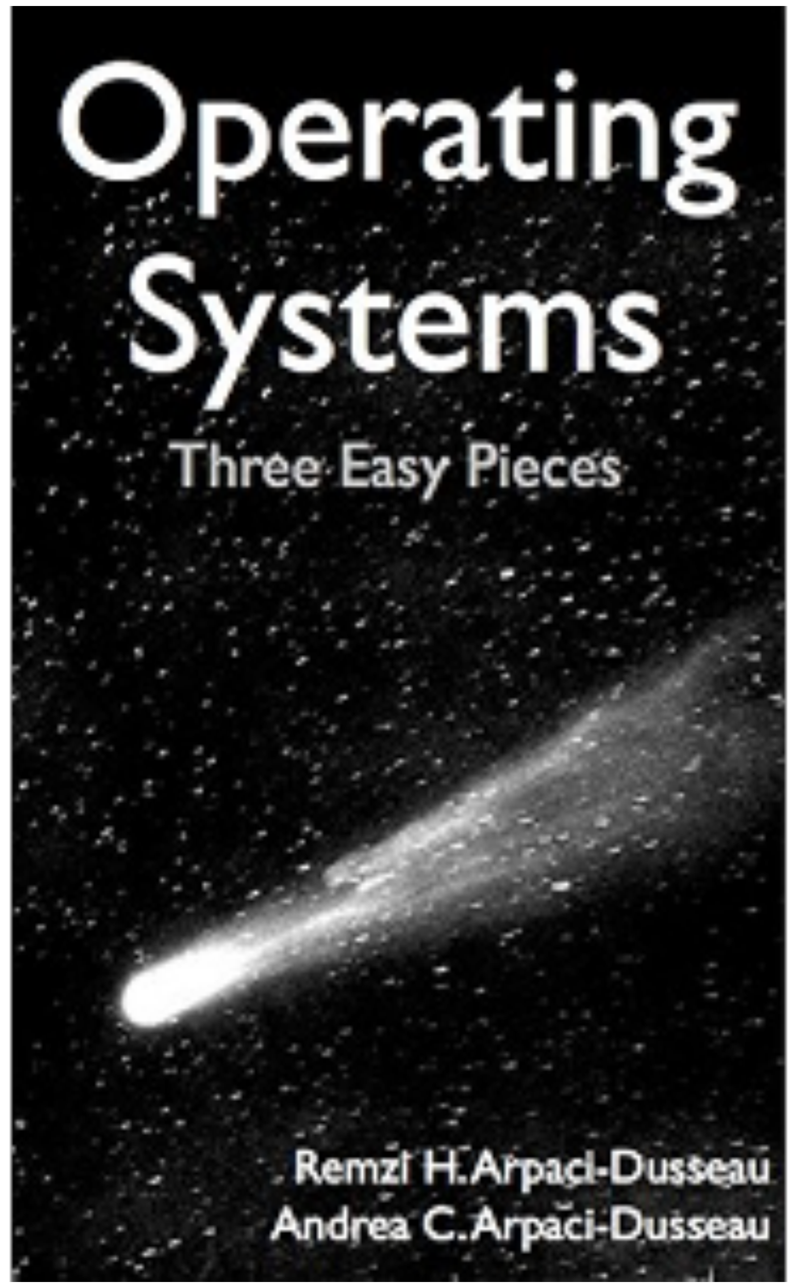
Suggested Textbooks

Computer systems are about carefully layering levels of abstraction.









Hands on experience

Background

The simplest database (demo)

```bash
#!/bin/bash

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

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

1. Search the lines that start with a parameter.

2. Only output the value part.

3. Only output the last line.

The simplest database (write)

```bash
#!/bin/bash

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

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

- Append only.
  - Writing is efficient.
  - Application:
    - Database Log
      - More real world challenges.
        - Concurrency
        - Disk space
        - Handling errors
        - …

The simplest database (read)

```bash
#!/bin/bash

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

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

- Always output the latest matched line.
    - Read is super slow.
        - Scan entire database.
        - The cost of lookup O(n).
            - Double lines => Double time

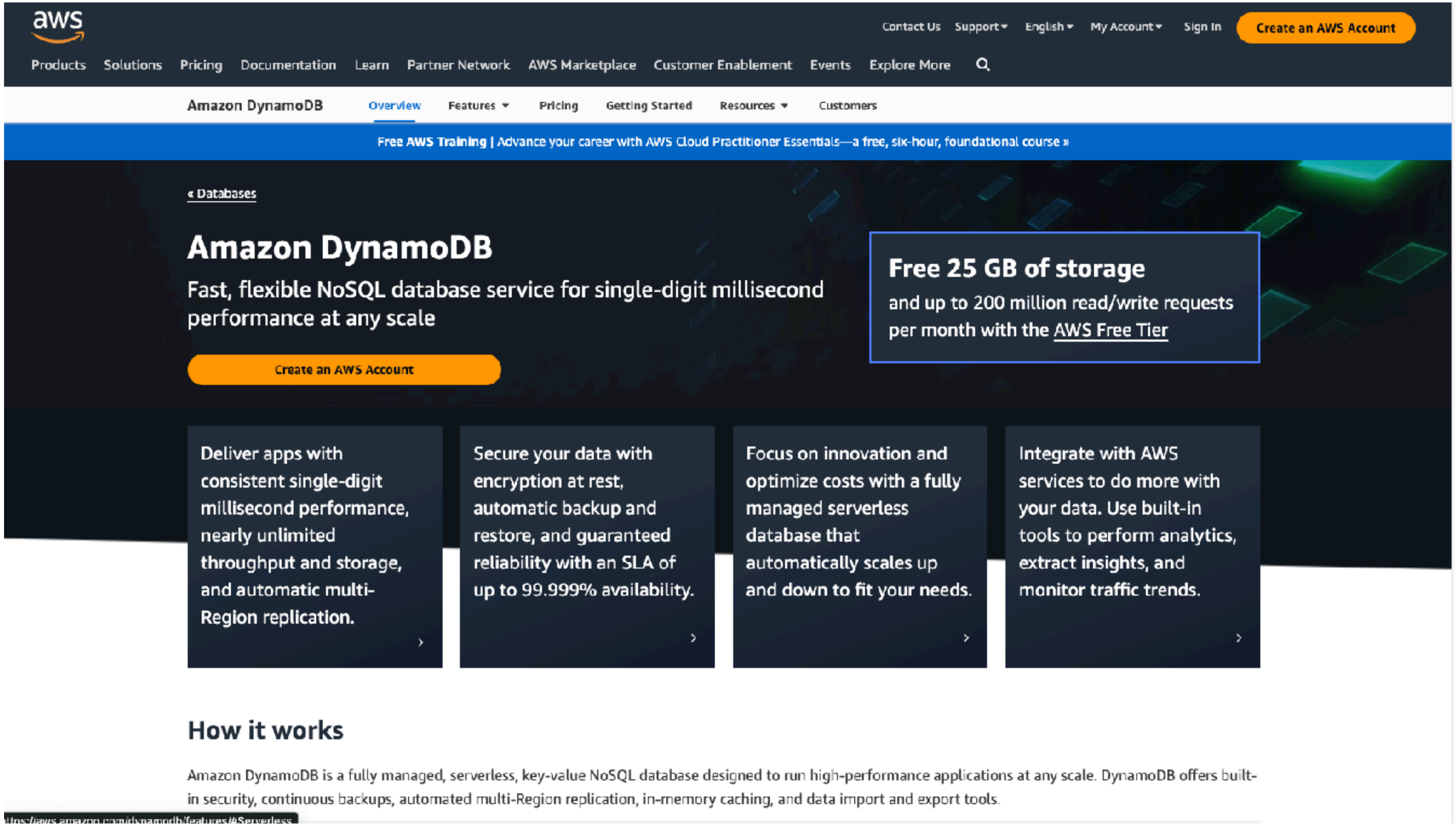
Improvement: Index

- Keep some additional metadata on the side, which acts as a signpost and helps you to locate the data you want.
- Faster to find the data.
- Update/remove/add the index is cheap.
- No free lunch!
  - Slows down the write.
  - Often needs to update the index.

Your role!

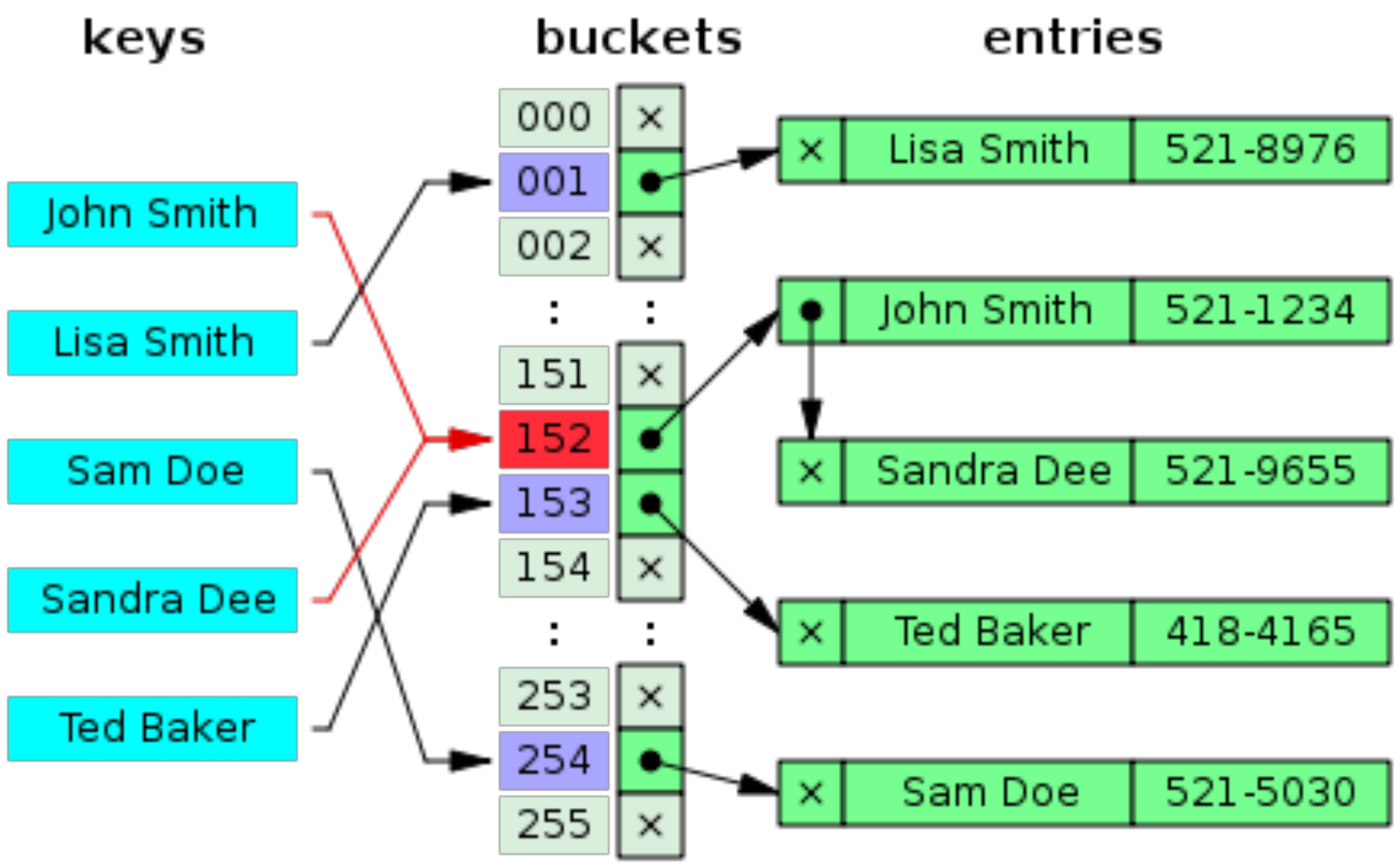
- **Well-chosen indexes speed up read queries, but every index slows down writes.**
  - DB do not index everything by default.
  - App developers or db admins choose indexes manually.
    - Based on domain knowledge.
  - Balance the tradeoffs.

Key-value stores (a common index)



Hash map/table

- **A hash table is a very fast approach to dictionary storage**
  - hash functions, separate chaining, linear probing
  - Search, insert, delete: ~ O(1).



Time Complexity

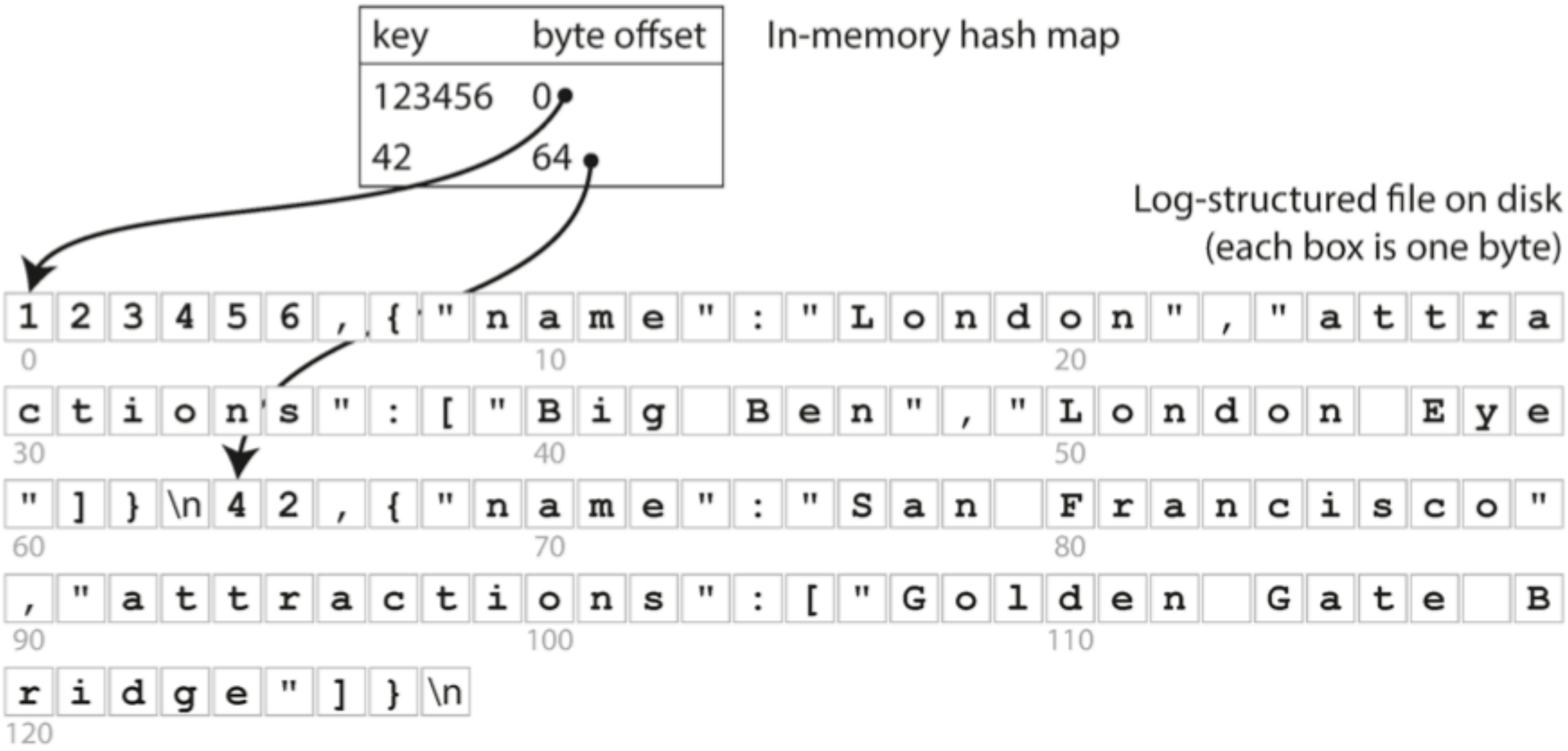
Average Case	Add	Remove	Search
Array	O(1)	O(n)	O(n)
Sorted Array	O(n)	O(lg n)	O(lg n)
Linked List	O(1)	O(n)	O(n)
BST	O(lg n)	O(lg n)	O(lg n)
Hash Table	~O(1)	~O(1)	~O(1)

Note: For sorted array and BST, keys have to be ordered.

8

See details in https://algs4.cs.princeton.edu/lectures/keynote/34HashTables.pdf

Hash map in Memory Hierarchy



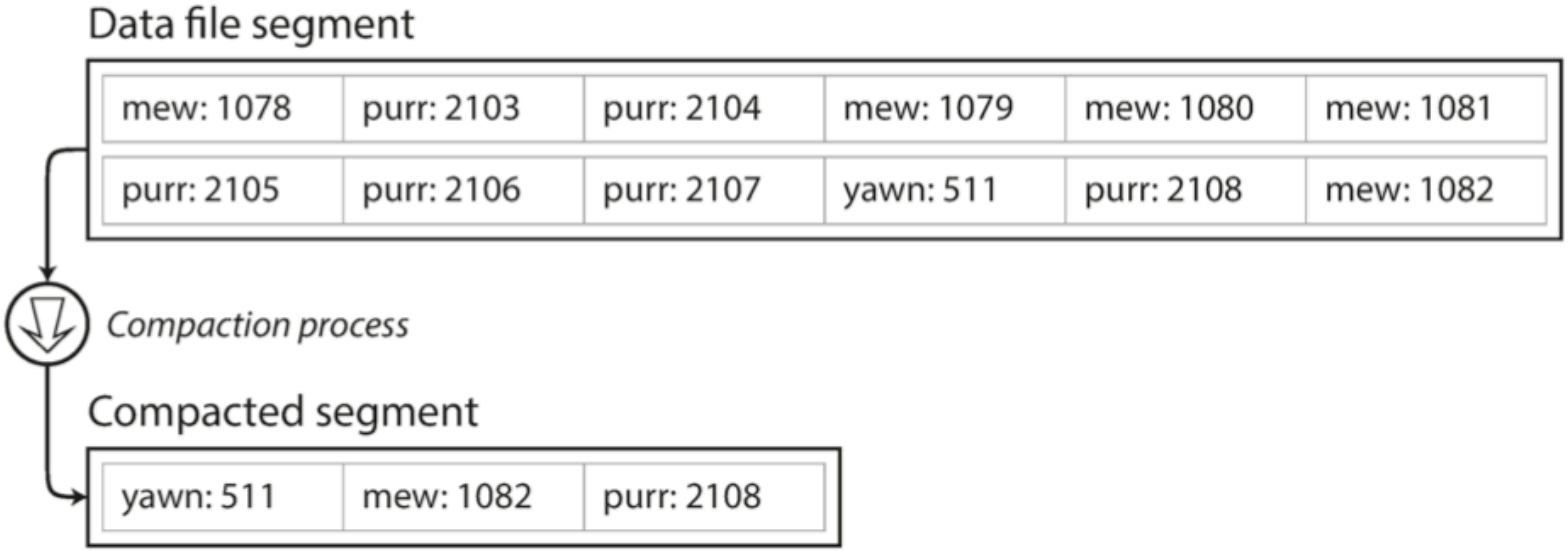
- Bitcask
- High performance reads and writes.
- Capacity:
  - All keys need to fit in the available RAM.
  - Values can be load from a disk. Much larger!!!

An example application:

- Track the number of times a video has been played.
  - Increment every time someone hits the play button.
- Memory capacity
  - 64 GB
  - URL: 2048 char = 2048 byte = 2KB
  - 64 GB/2KB = 32 million.
- Note: YouTube has over 800 million videos.
- When to keep all keys in memory?
  - Lots of writes,
  - not much distinct keys,
  - a large number of writes per-day.

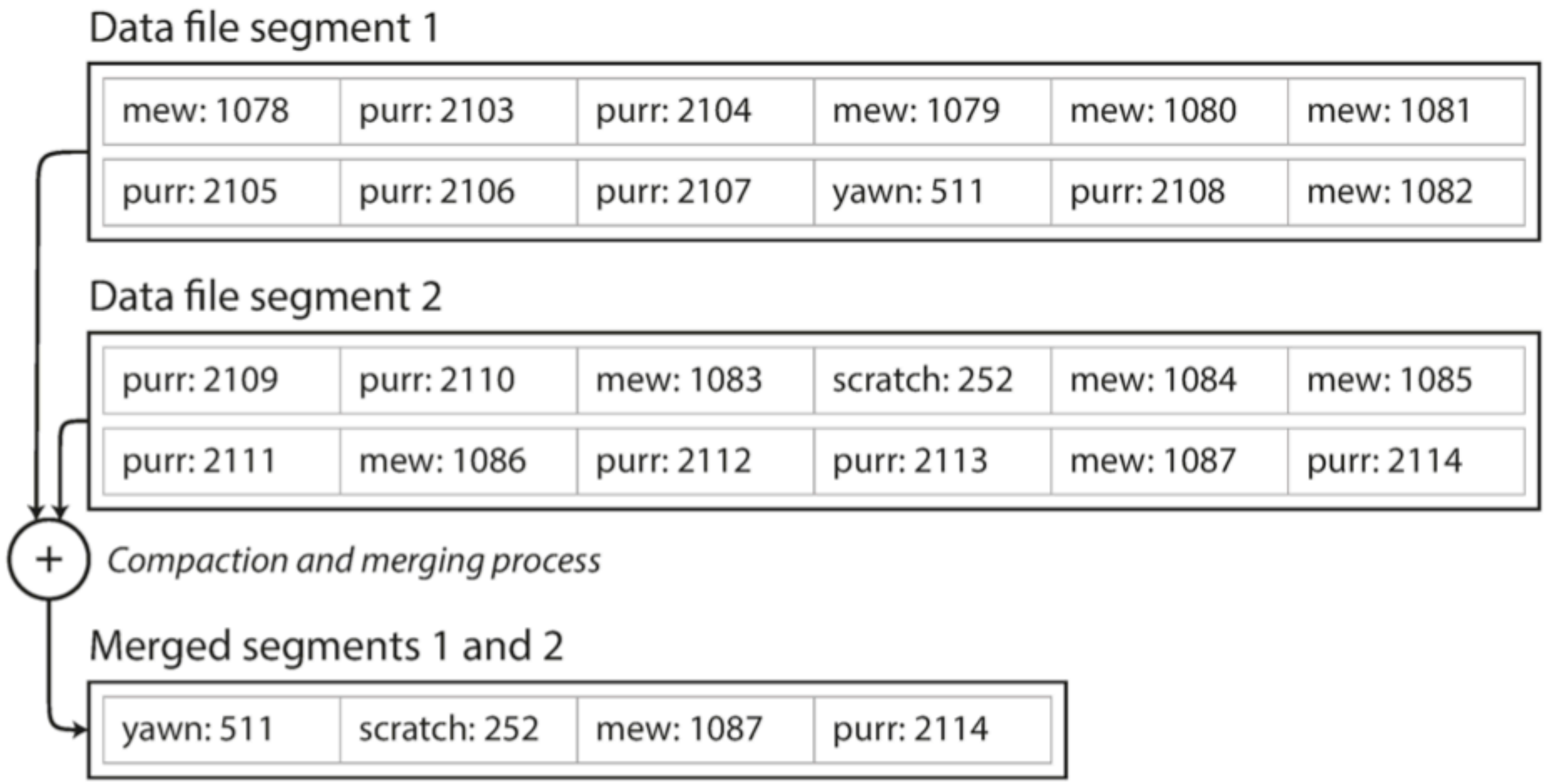
Run out of disk space? Segment compaction

- Segments of a certain size.
- Perform compaction.
  - Throw away duplicate logs and keep only the most recent update.

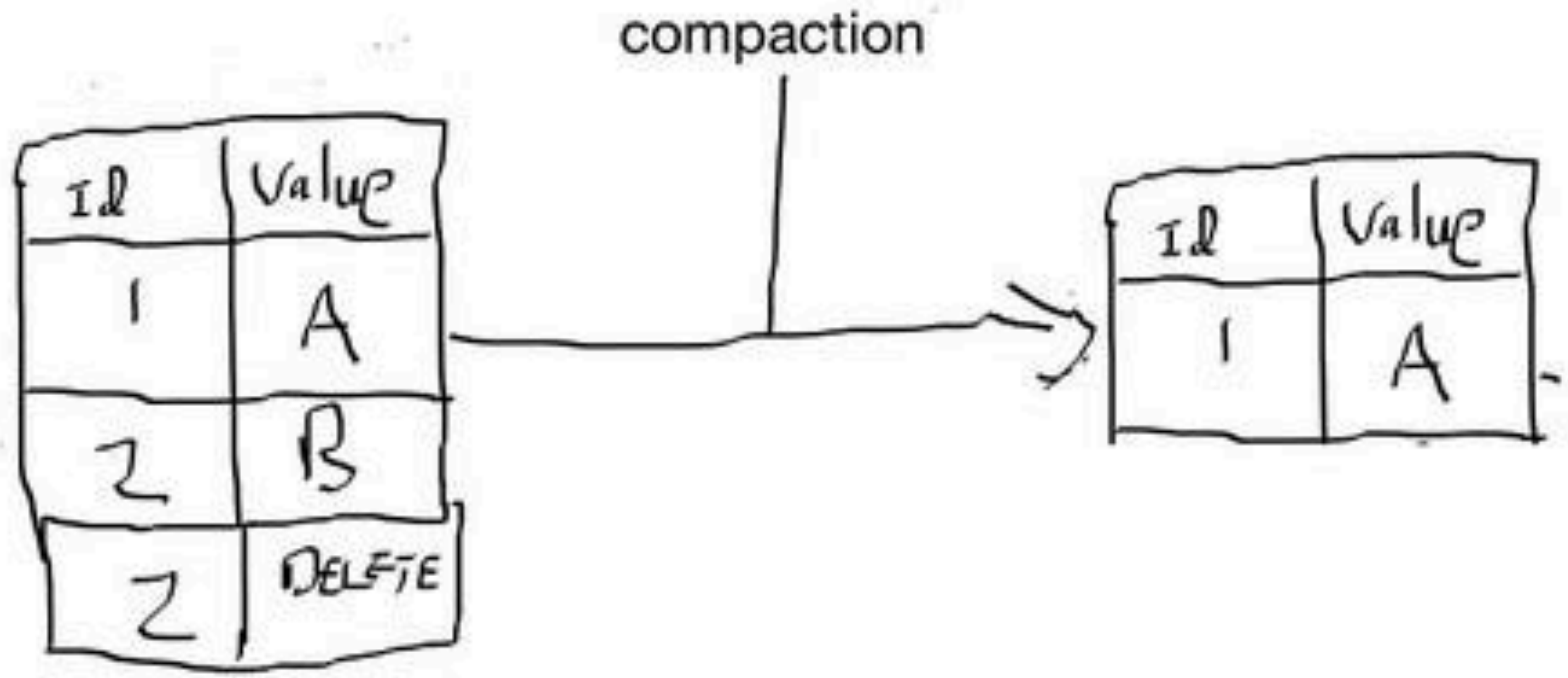


Segment compaction

- Frozen segments. Never modified.
    - Only merge frozen segments and write the output to a new file.
- The read and write can work as normal using the old segment files.
- After the merging,
    - Read requests from the merged file.
    - Delete old segment files



How to delete a record?



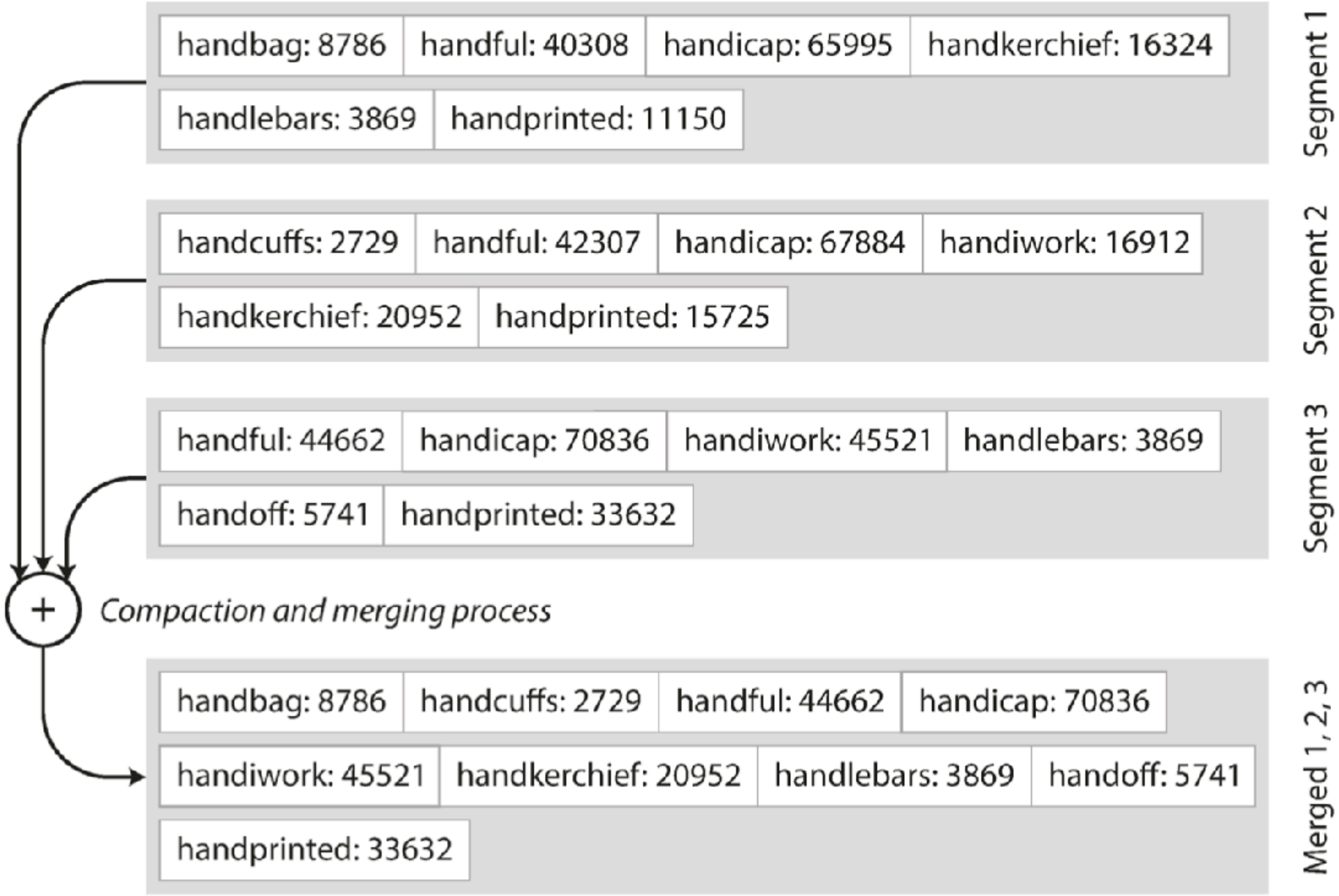
Crash recovery

- Restart a database.
  - Segments are often large.
    - Loading is slow.
    - Store the segments' hash maps on disk.
- Partially written records. e.g., lose power?
  - Checksums for each record.
  - Detect and ignore corrupted parts.

Hash Table Index

- Advantages (Append-only & imputable):
  - Very fast write.
    - Recall how hard drive works.
  - Simple concurrency and crash recovery.
    - No need to worry about partially written records.
  - Avoid the problems of fragmented data files.
- Disadvantage
  - The hash table index must fit in memory.
    - Can we put hash table index on disk? => DDIA p75.
  - **Range queries are not efficient.**

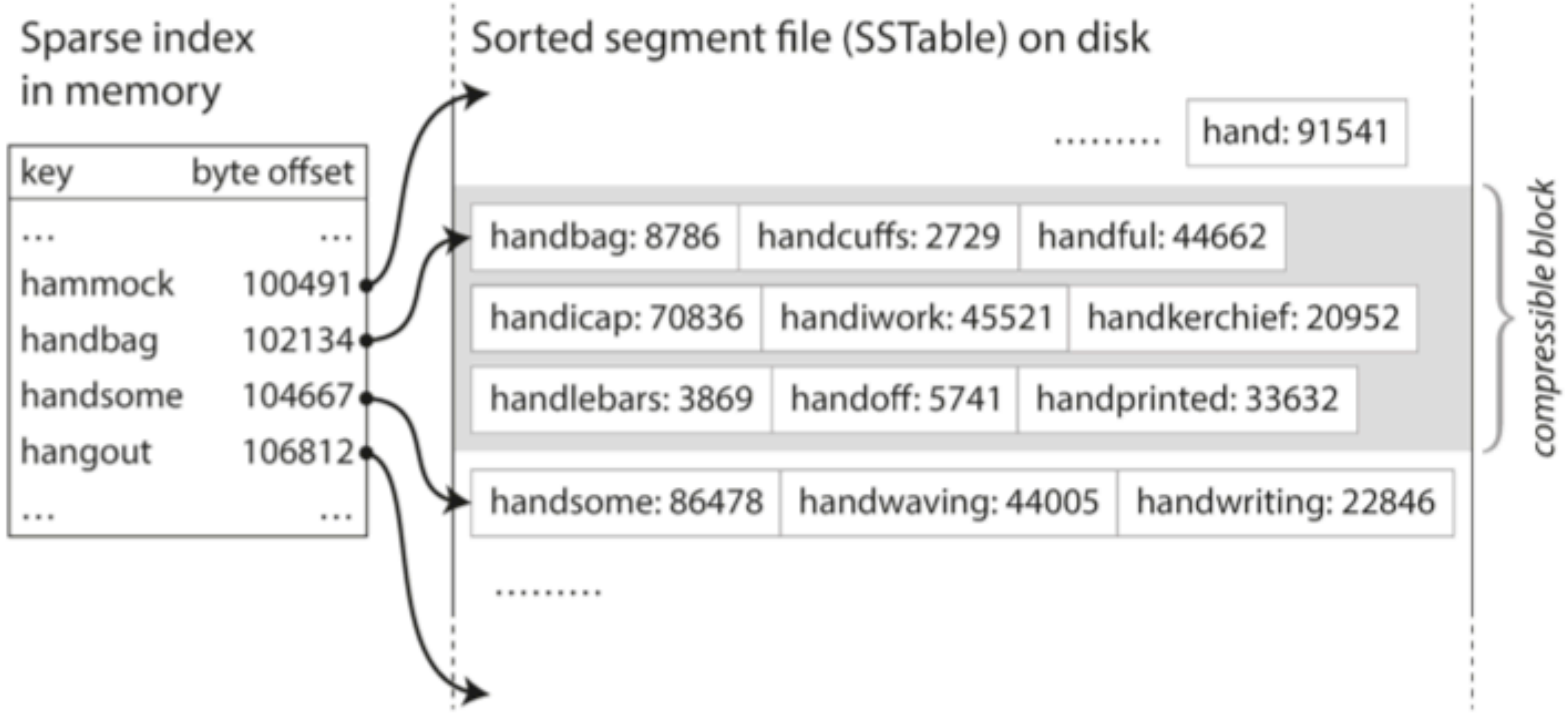
SSTable (sorted string table)



- Change the format of the segment files
  - Sorted by keys

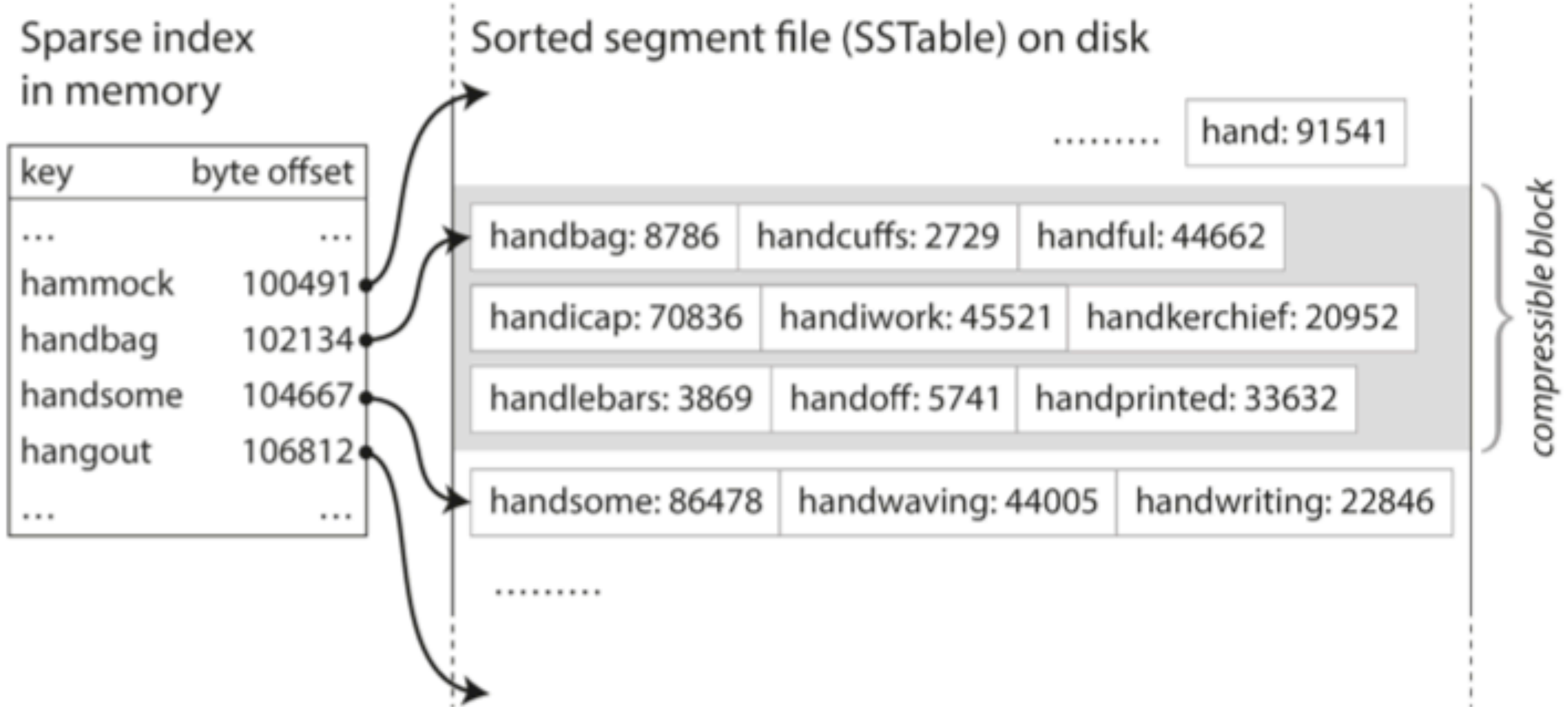
SSTable

- Merging segments is simple and efficient, **even if the files are bigger than the available memory.**
  - Merge sort: https://en.wikipedia.org/wiki/Merge_sort
- No longer need to keep an index of all the keys in memory.
  - Jump to the range.
  - Similar idea as Hash table.



SSTable

- Sparse in-memory index
- Each segment file for a few kilobytes.
- “Better idea”:
  - Assume that the keys and values had a fixed size, use binary search on a segment file and avoid the in-memory index.
    - Only useful in special applications.
- Compressible blocks.



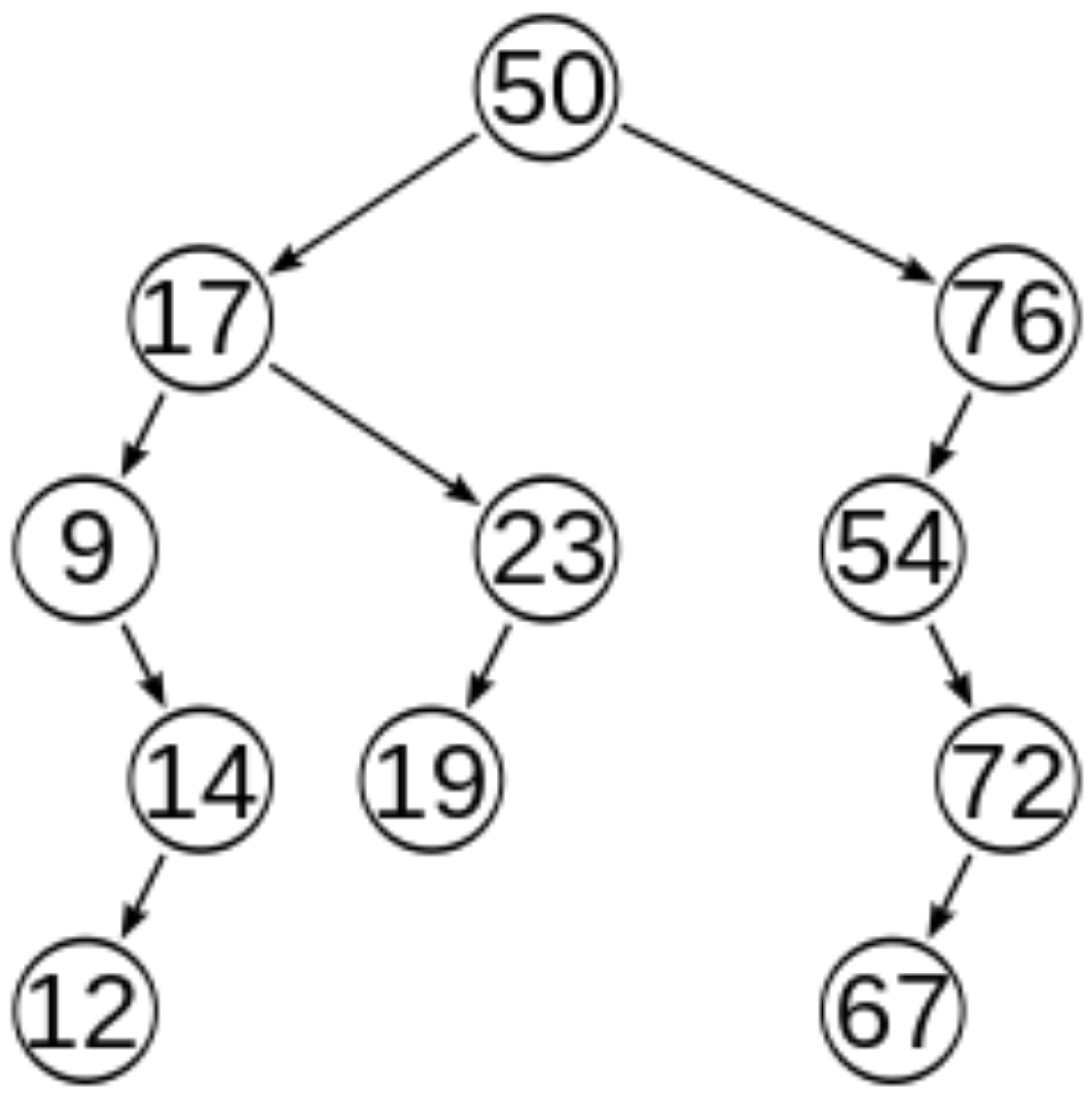
How do you get your data to be sorted by key in the first place?

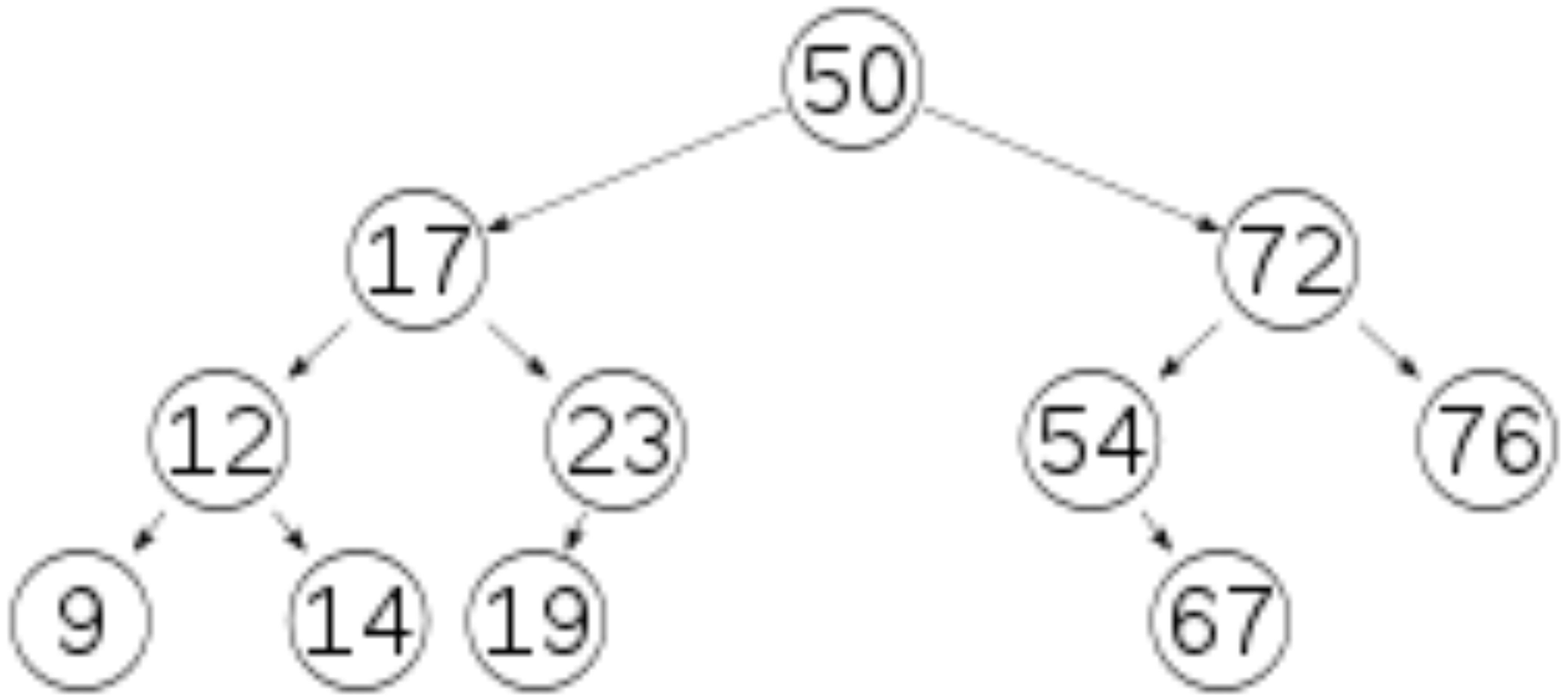
Memtable: Sorted structure in memory

- Easier to manipulate data in memory than disk.
  - Why?
- Maintain a sorted data structure in memory.

Self-balanced trees

- Any node-based **binary search tree** that automatically keeps its height (maximal number of levels below the root) small in the face of arbitrary item insertions and deletions.
  - E.g., Red-black trees or AVL trees
  - Height O(log n)





Complexity Comparison of Various Structures

Operation	Sequential List (Sorted Array)	Linked List	AVL Tree
Search for x	$O(\log n)$	$O(n)$	$O(\log n)$
Search for kth item	$O(1)$	$O(k)$	$O(\log n)$
Delete x	$O(n)$	$O(1)$[1]	$O(\log n)$
Delete kth item	$O(n - k)$	$O(k)$	$O(\log n)$
Insert x	$O(n)$	$O(1)$[2]	$O(\log n)$
Output in order	$O(n)$	$O(n)$	$O(n)$

[1]Doubly linked list and position of x known.

[2]Position for insertion known

10-16

AVL tree		
Type	Tree	
Invented	1962	
Invented by	G.M. Adelson-Velskii and E.M. Landis	
Time complexity in big O notation		
	Average	Worst case
Space	O(n)	O(n)
Search	O(log n)	O(log n)
Insert	O(log n)	O(log n)
Delete	O(log n)	O(log n)

Binary search tree		
Type	tree	
Invented	1960	
Invented by	P.F. Windley, A.D. Booth, A.J.T. Colin, and T.N. Hibbard	
Time complexity in big O notation		
Algorithm	**Average**	**Worst case**
Space	$O(n)$	$O(n)$
Search	$O(\log n)$	$O(n)$
Insert	$O(\log n)$	$O(n)$
Delete	$O(\log n)$	$O(n)$