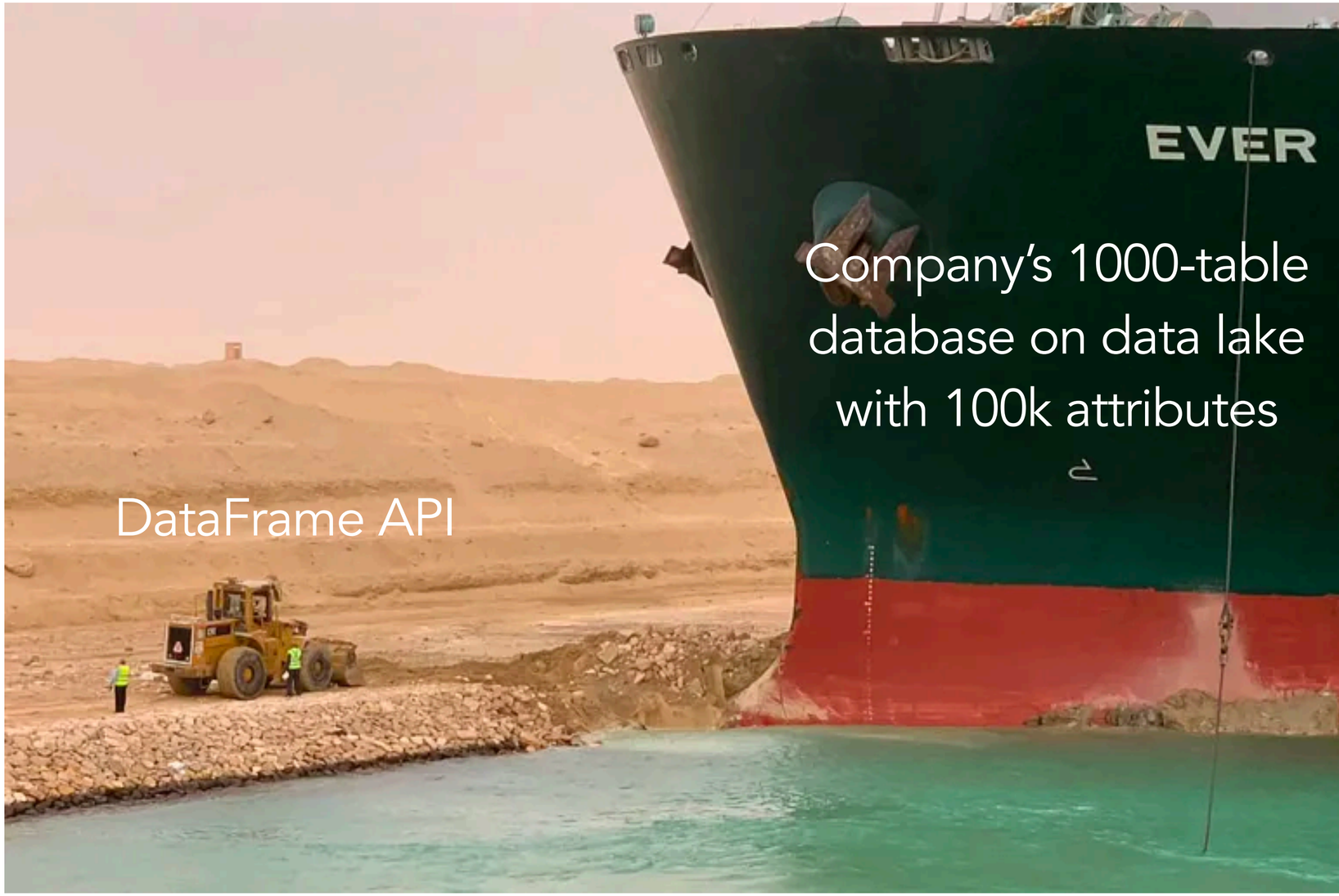
DSC 204a Scalable Data Systems

- 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

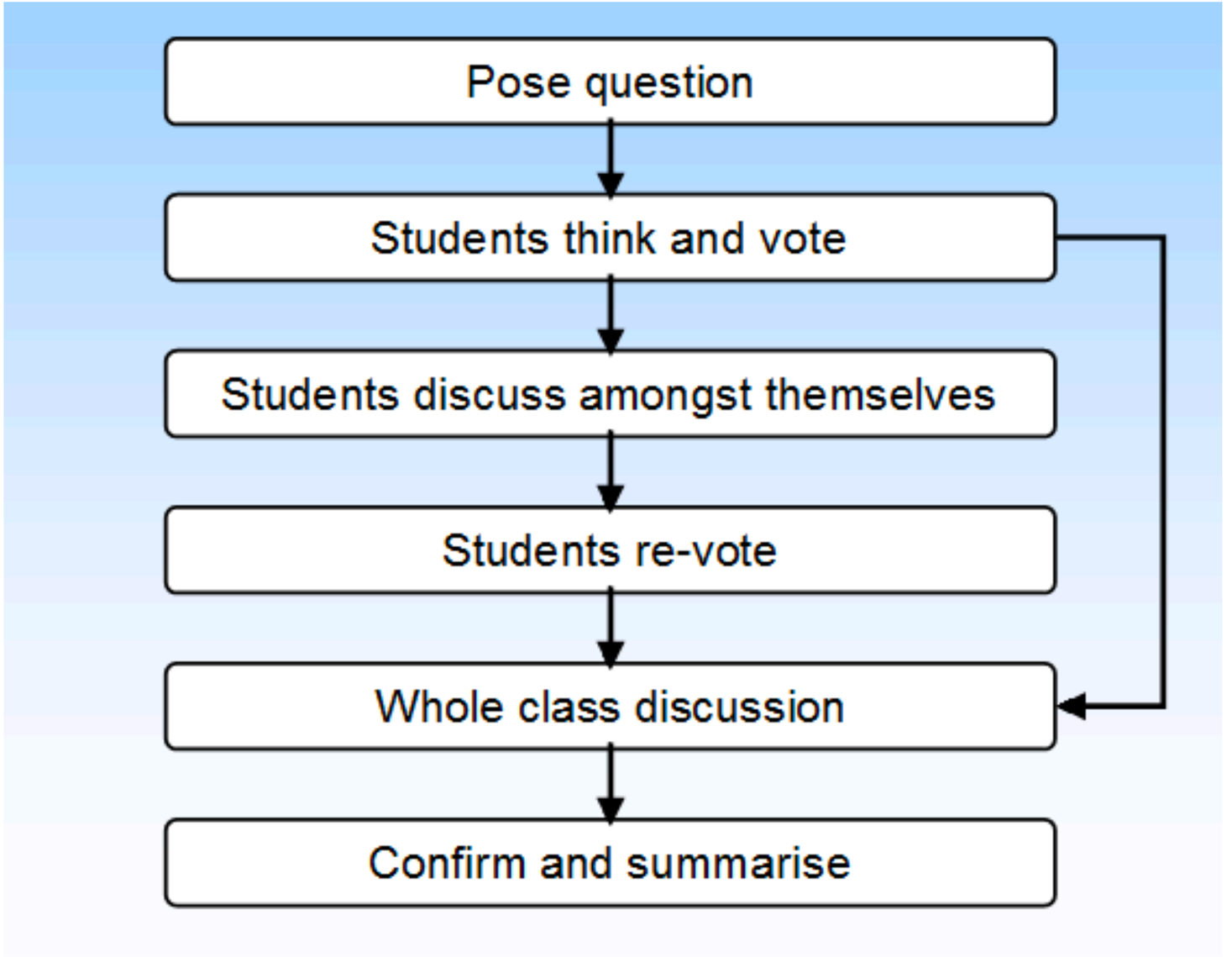
Data Processing and Programming model (5 weeks)

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

Today’s activities

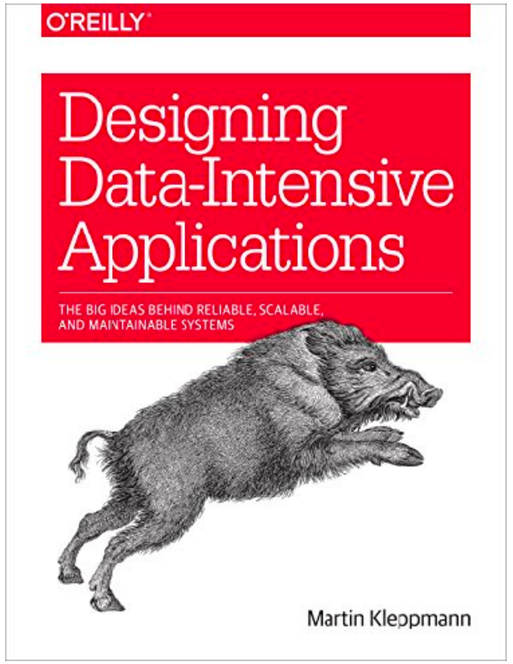
- PIA.
- Chapter 3 Storage and Retrieval
  - Hash Indexes
  - SSTables and LSM-Trees
  - B-Trees
  - Other indexing structures

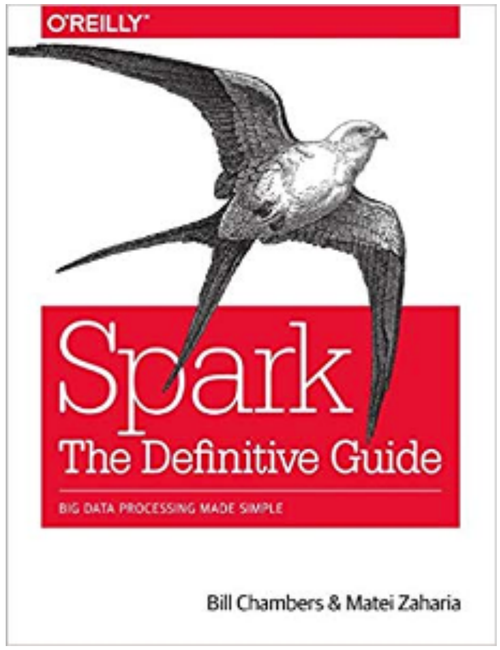
Peer instruction activity



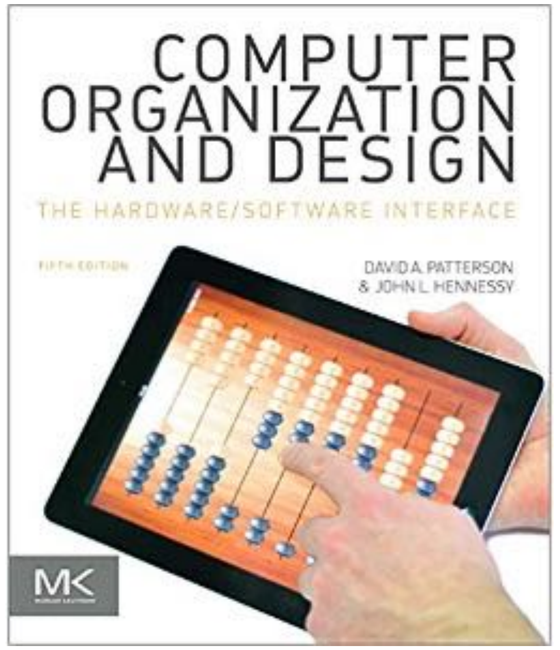
Suggested Textbooks

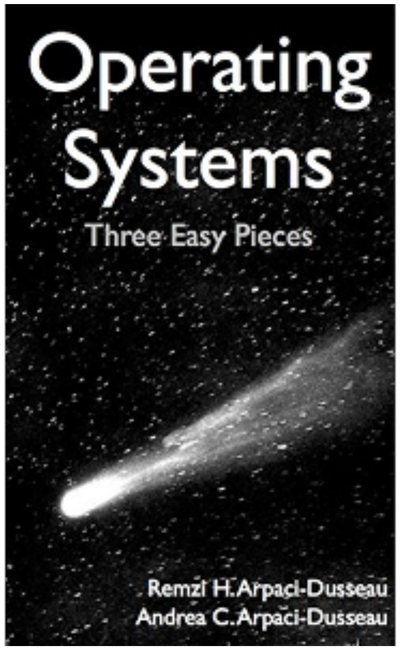
Computer systems are about carefully layering levels of abstraction.





Hands on experience





Background

The simplest database (demo)

```
#!/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)

```
#!/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

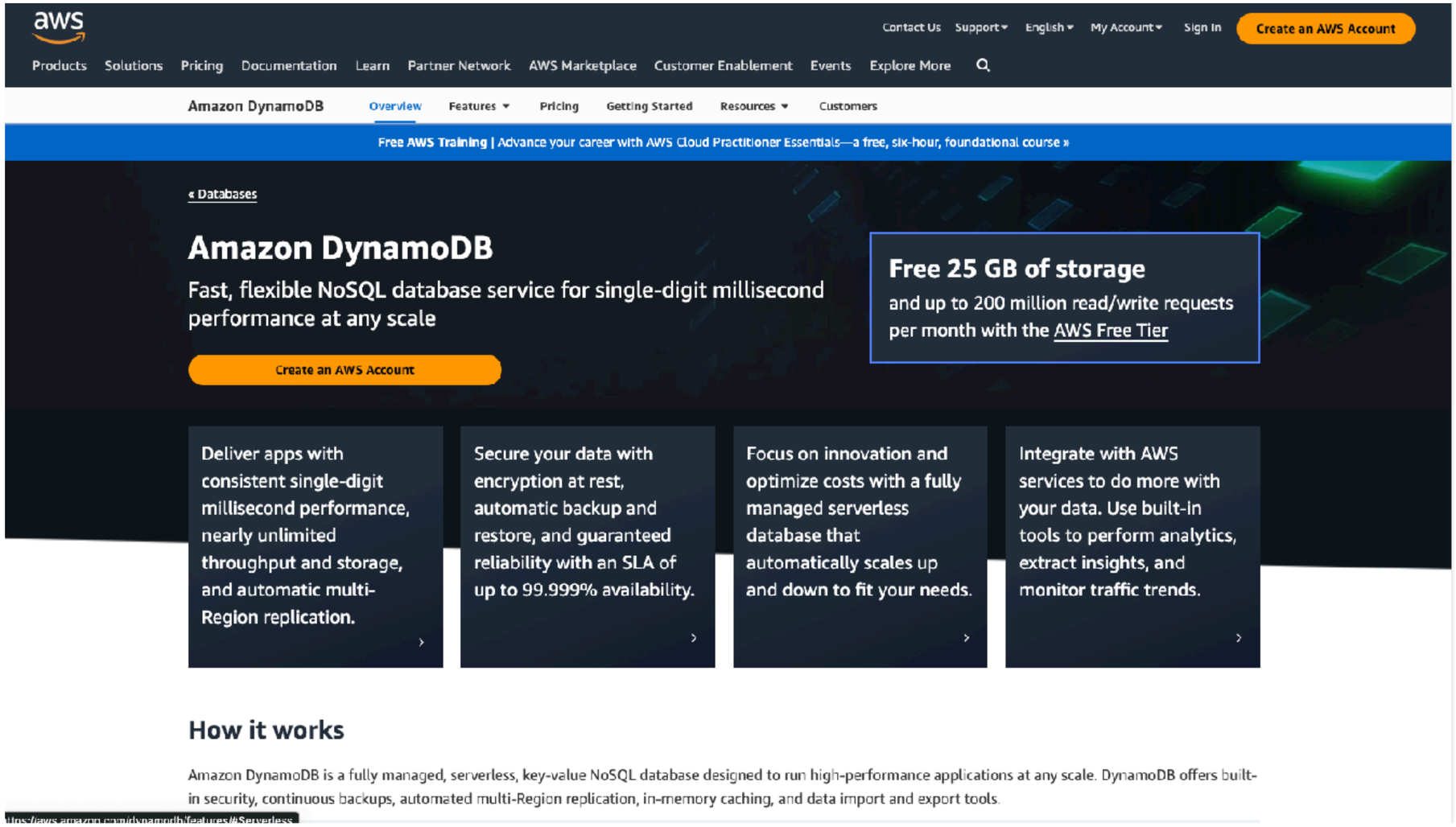
Improvement: Index

- 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!

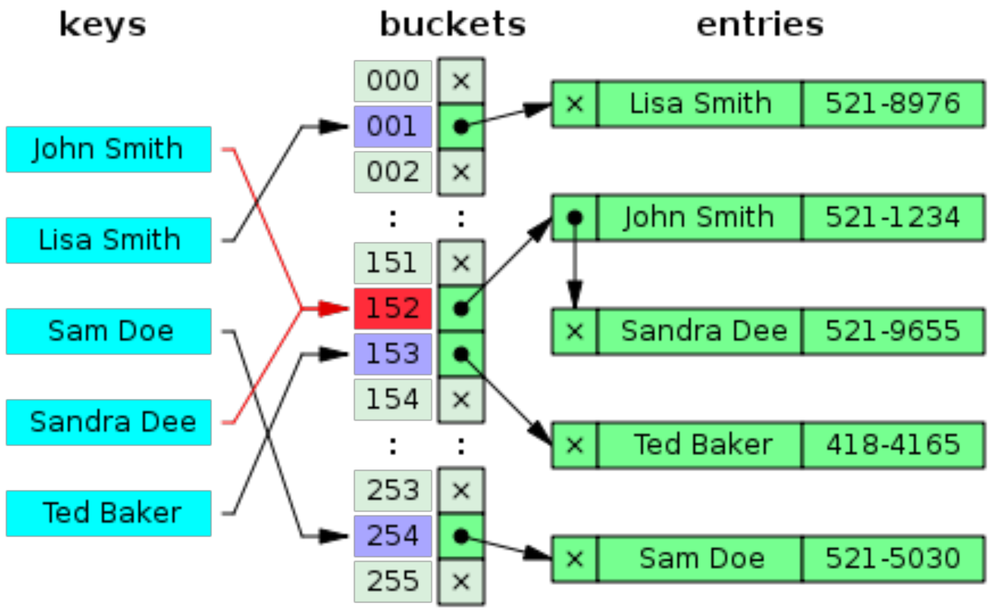
- **Well-chosen indexes speed up read queries, but every index slows down writes.**
  - DB do not index everything by default.
  - App developers or db admins choose indexes manually.
    - Based on domain knowledge.
  - Balance the tradeoffs.

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

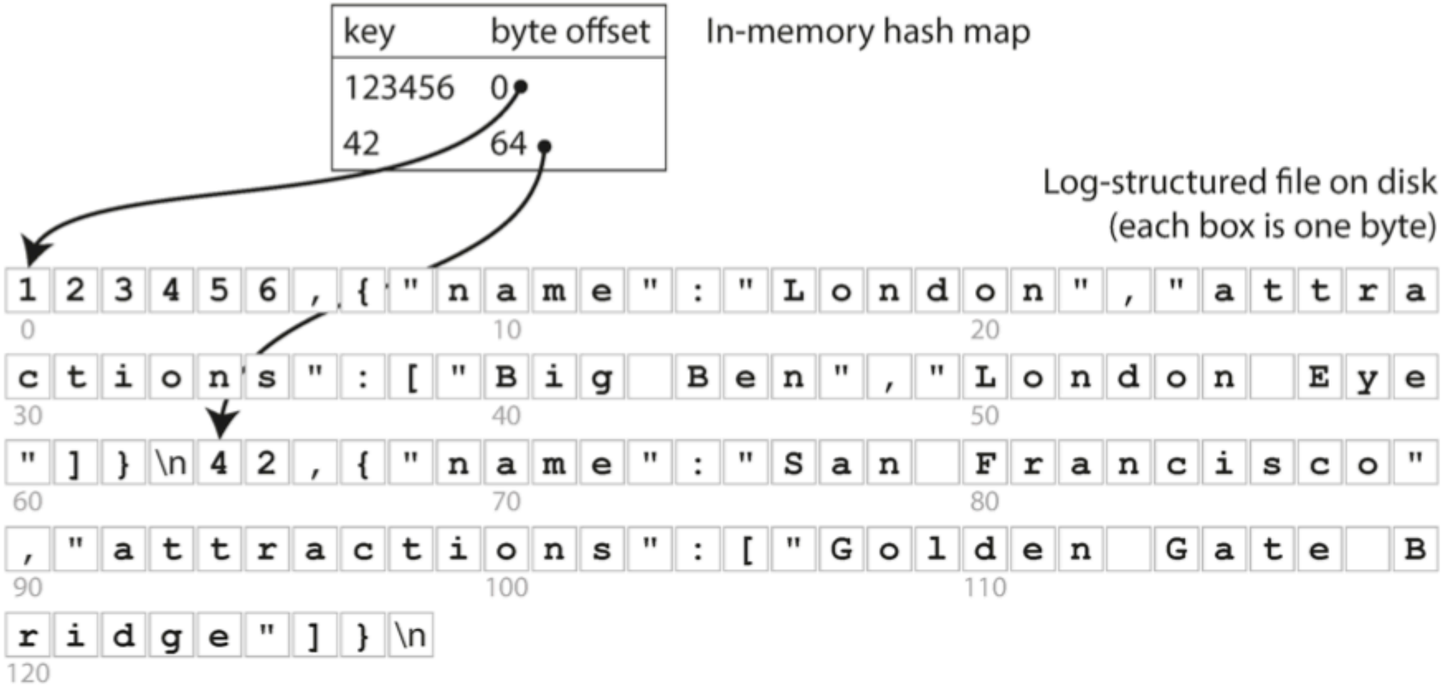
Average Case	Add	Remove	Search
Array	O(1)	O(n)	O(n)
Sorted Array	O(n)	O(lg n)	O(lg n)
Linked List	O(1)	O(n)	O(n)
BST	O(lg n)	O(lg n)	O(lg n)
Hash Table	~O(1)	~O(1)	~O(1)

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



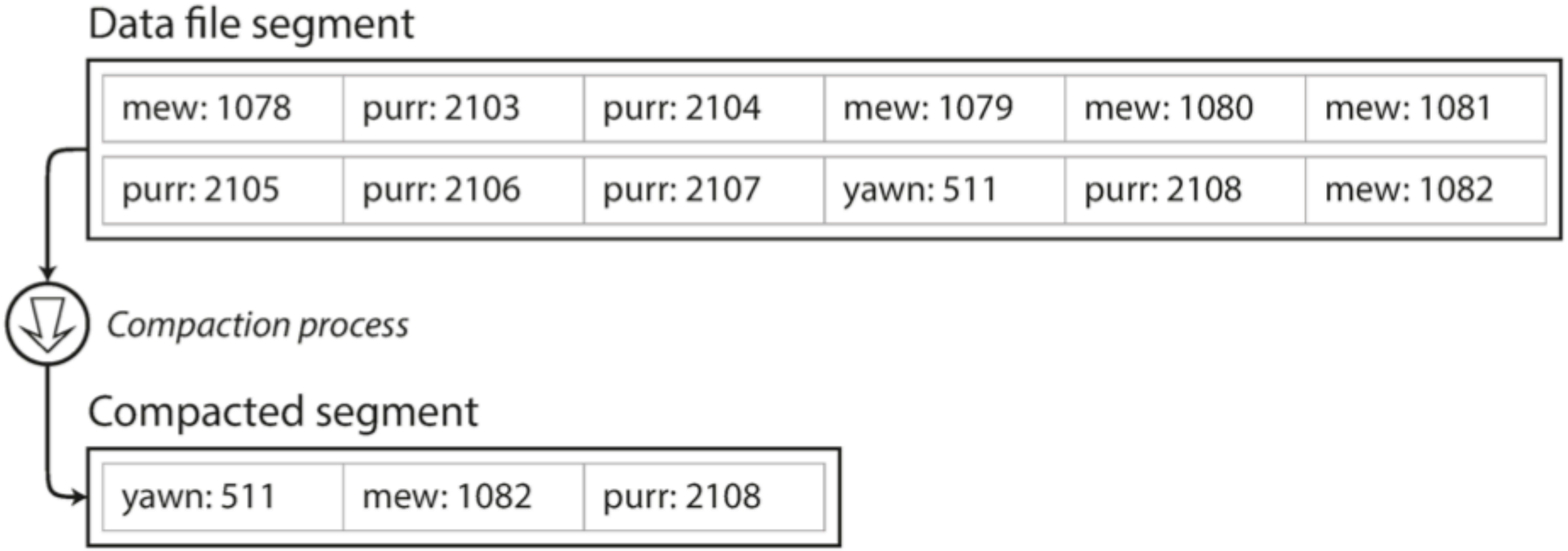
- Bitcask
- High performance reads and writes.
- Capacity:
  - All keys need to fit in the available RAM.
  - Values can be load from a disk. Much larger!!!

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.

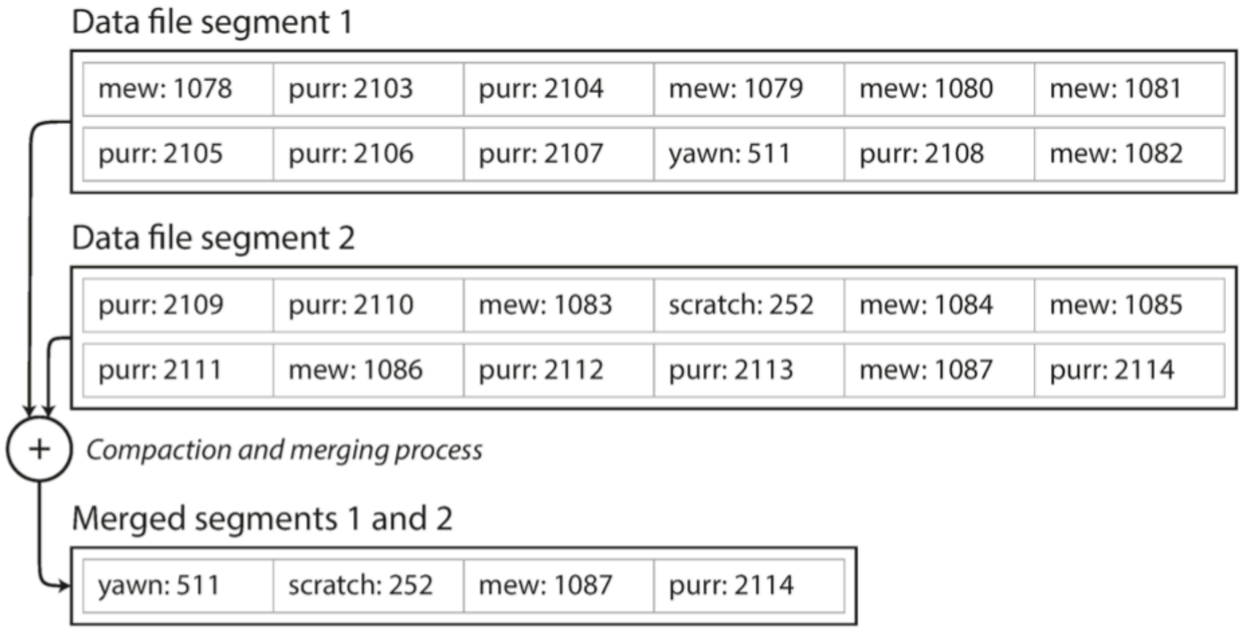
Run out of disk space? Segment compaction

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

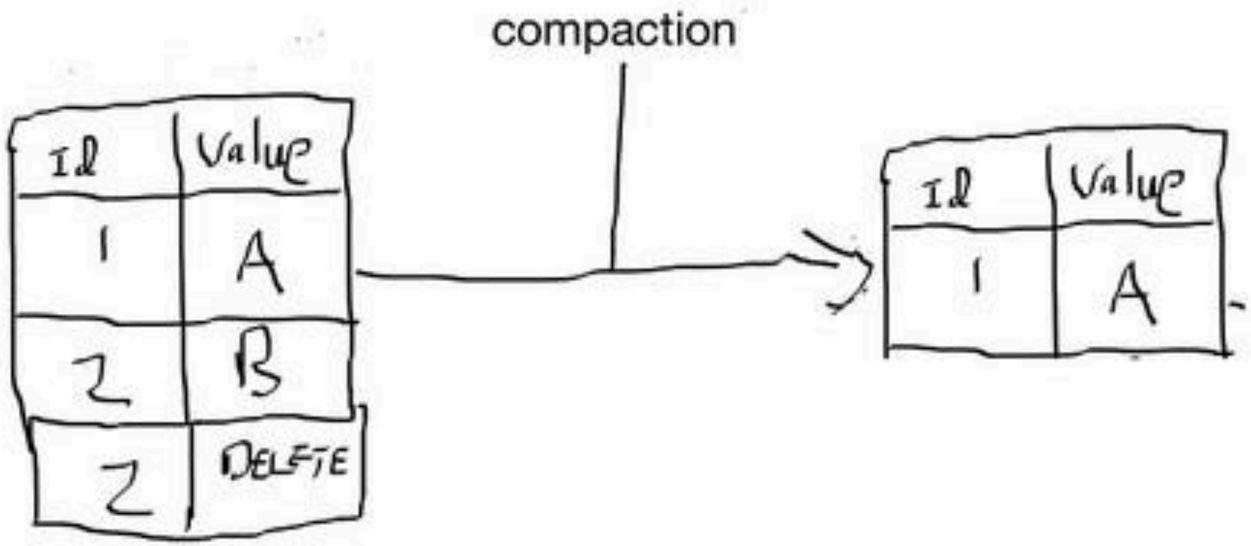


Segment compaction

- 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.**

Today's activities

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

Packet Delay

- Sum of a number of different delay components:
- Propagation delay on each link: time for a signal to propagate through the media
  - Proportional to the length of the link
- Transmission delay on each link: time it takes to push the packet's bits onto the link
  - Proportional to the packet size and 1/link speed
- Processing delay on each router: Time it takes a router to process the packet header
  - Depends on the speed of the router
- Queuing delay on each router: time the packet spends in routing queues
  - Depends on the traffic load and queue size

Some simple calculations. TCP. (mbps/kbps)

- Cross country latency
  - Distance/speed = 5 * 10^6m / 2x10^8m/s = 25 * 10^-3 s = 25ms
  - 50ms RTT
- Link speed (capacity) 100Mbps
- Packet size = 1250 bytes = 10 kbits
  - Packet size on networks usually = 1500 bytes across wide area or 9000 bytes in local area
- 1 packet takes
  - 10k/100M = .1 ms to transmit
  - 25ms to reach there
  - ACKs are small → so 0ms to transmit
  - 25ms to get back
- Effective bandwidth = 10kbits/50.1ms = 200kbits/sec ☹

https://www.quora.com/Which-is-faster-the-speed-of-electric-current-through-a-copper-wire-or-the-speed-of-light

Network speed calculation

Consider a point-to-point link 2 km in length. At what bandwidth would propagation delay (at a speed of 2 x 10^8 m/s) equal transmission delay for 100-byte packets?

Propagation delay = Distance / Propagation Speed
= 2000 m / (2 * 10^8) m/s
= 10^-5 s

Bandwidth = Size of packet / Transmit delay
= (100 * 8) bits / 10^-5 s
= 8 * 10^7 bits/s
= 80 Mbps

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

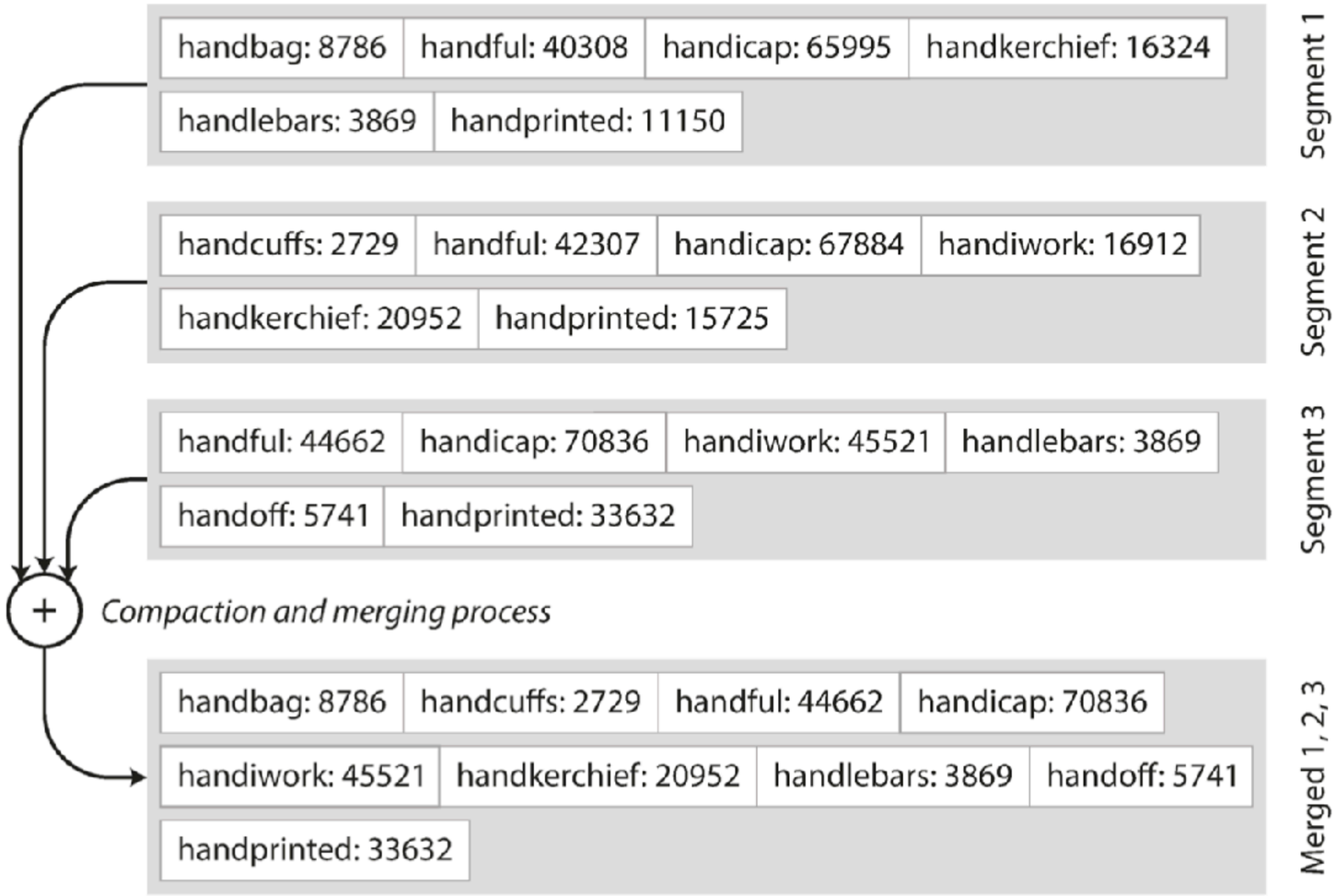
Data Processing and Programming model (5 weeks)

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

Data indexes

- Straw-man design (bash script, get, set, append-only)
  - Fast write
  - Slow read
  - Large storage space.
- Hashtable (all keys in the memory, all values on the disk, background compaction)
  - Fast write & read
  - Less storage space
  - All keys need to fit in memory.
- ????

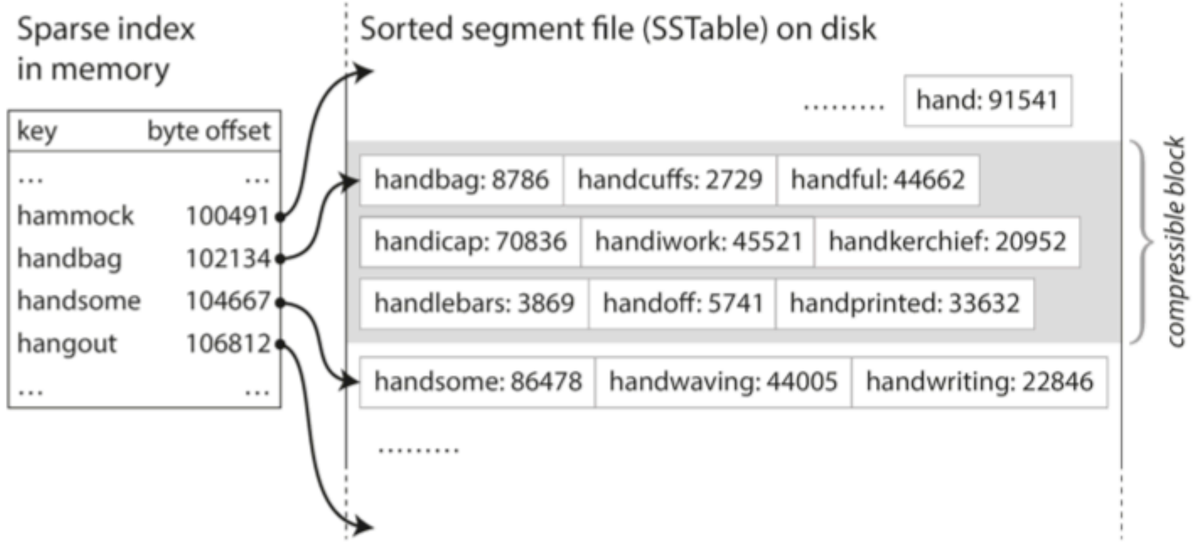
SSTable (sorted string table)



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

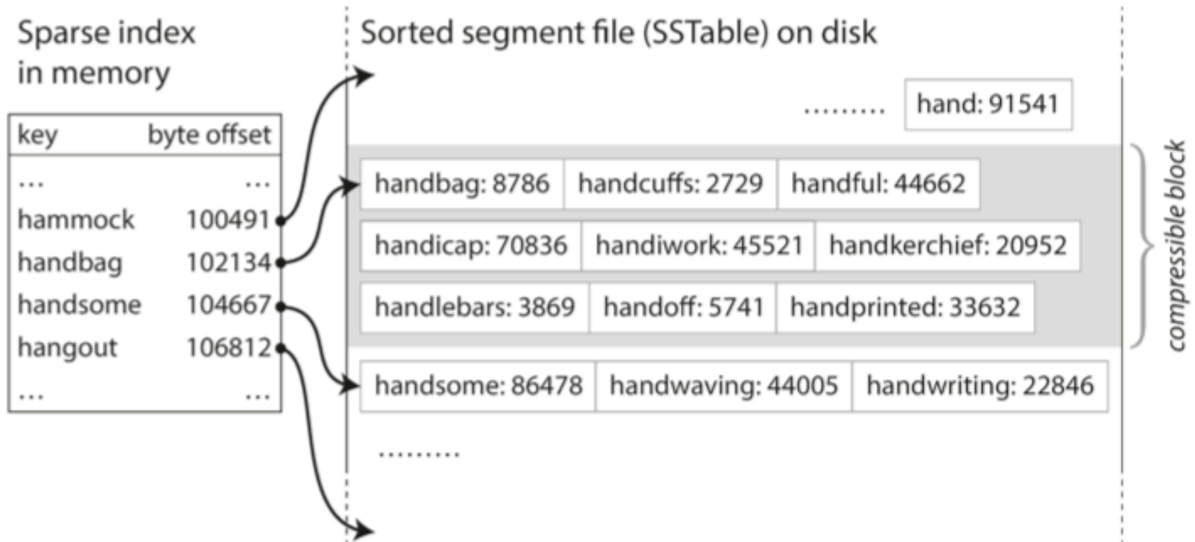
SSTable

- 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 implementation

- Sparse in-memory index
- Each segment file for a few KB-MB.
- “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.



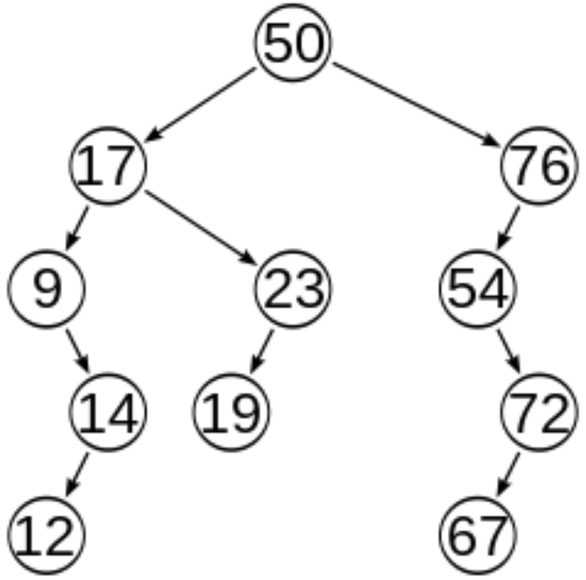
How do you get your data to be sorted by key in the first place?

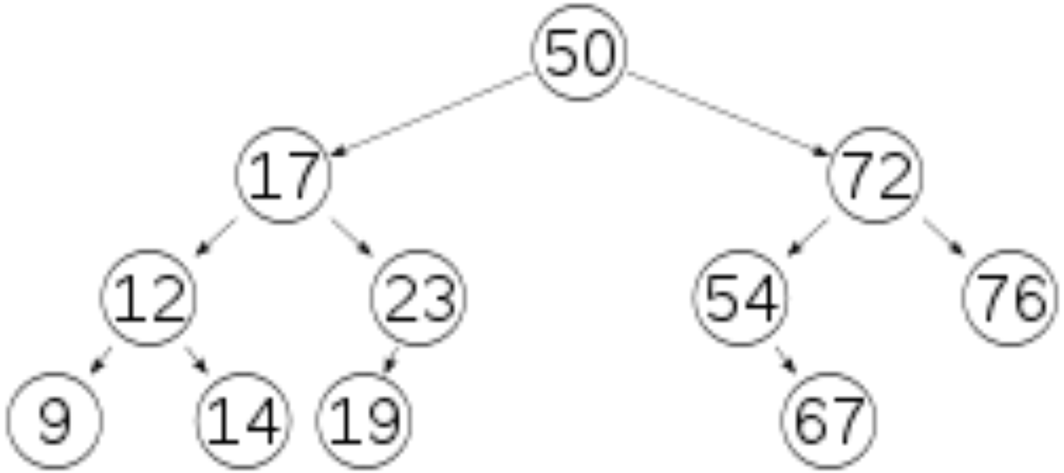
Memtable: Sorted structure in memory

- 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

AVL v.s Binary Search Tree

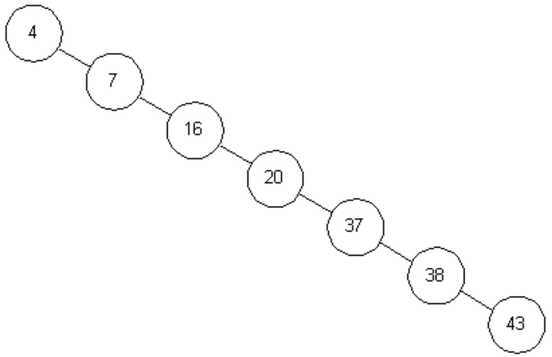
AVL tree		
Type	Tree	
Invented	1962	
Invented by	G.M. Adelson-Velskii and E.M. Landis	
Time complexity in big O notation		
	Average	Worst case
Space	O(n)	O(n)
Search	O(log n)	O(log n)
Insert	O(log n)	O(log n)
Delete	O(log n)	O(log n)

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	$O(\log n)$	$O(n)$



How a LSM (Log-structured merged-tree) storage engine works

- Write:
  - When a write comes in, add it to the memtable.
  - If the memtable > a threshold, save the memtable as the most recent segment.
- Read:
  - Check if the key in the memtable.
  - Then go through the segments.
- Background:
  - Merge and compact.

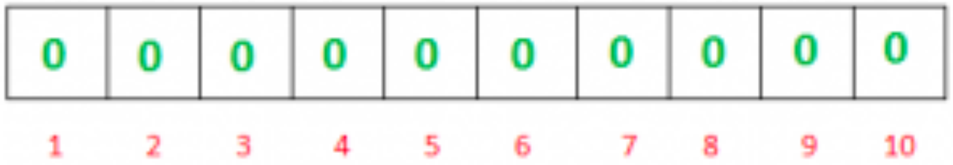
One issue of LSM

- What will happen if we want to look up **keys that do not exist in the database**?
  - Check the memtable
  - Check the segments all the way back to the oldest
- Optimization:
  - Use a bloom filter to test whether a key exist.

Bloom filters

- A space efficient **probabilistic** data structure
  - It can test whether an element is a member of a set.
  - Computation: O(k) and Space: O(m).
- **Cost: probabilistic?**
  - False positive:
    - It might tell that an element is a member of a set while it is not.

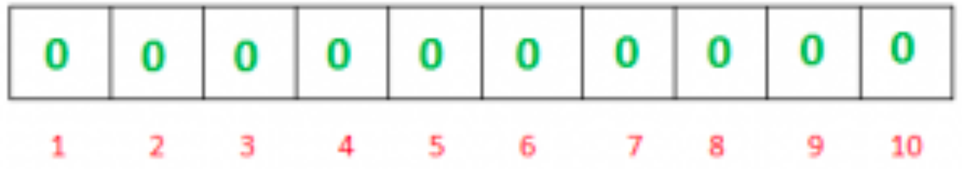
Initialization (m):



Three hashing functions (k): h1, h2, h3

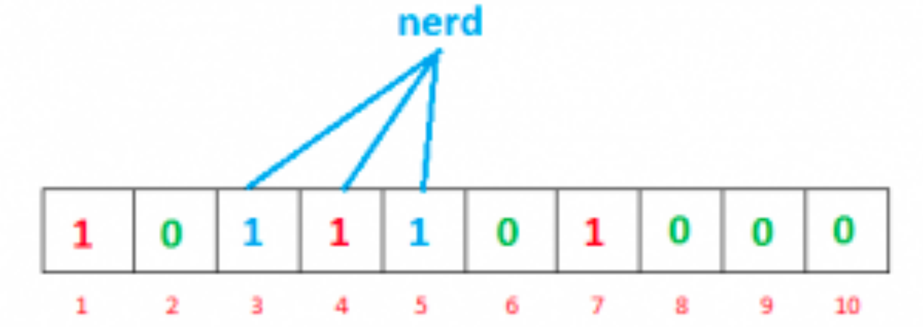
Bloom filters (read and write)

A set of words: {“geeks”, “nerd”}



h1(“geeks”) % 10 = 1
h2(“geeks”) % 10 = 4
h3(“geeks”) % 10 = 7

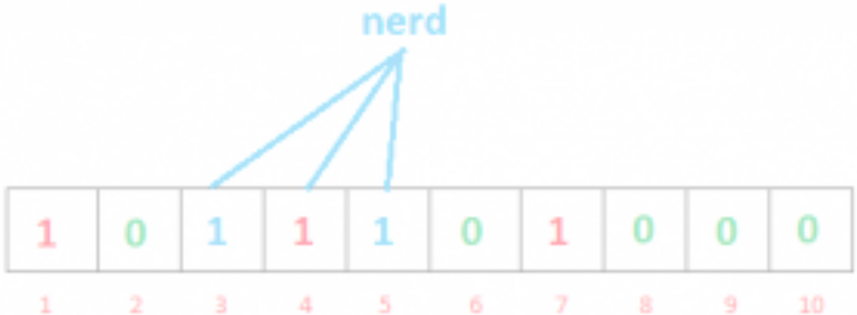
h1(“nerd”) % 10 = 3
h2(“nerd”) % 10 = 5
h3(“nerd”) % 10 = 4



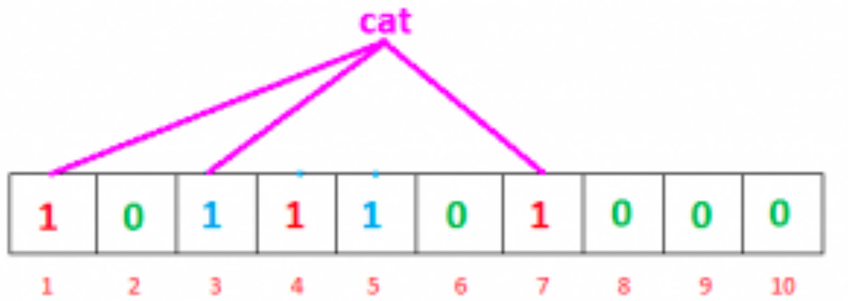
Bloom filters - False positive

h1("geeks") % 10 = 1
h2("geeks") % 10 = 4
h3("geeks") % 10 = 7

h1("nerd") % 10 = 3
h2("nerd") % 10 = 5
h3("nerd") % 10 = 4



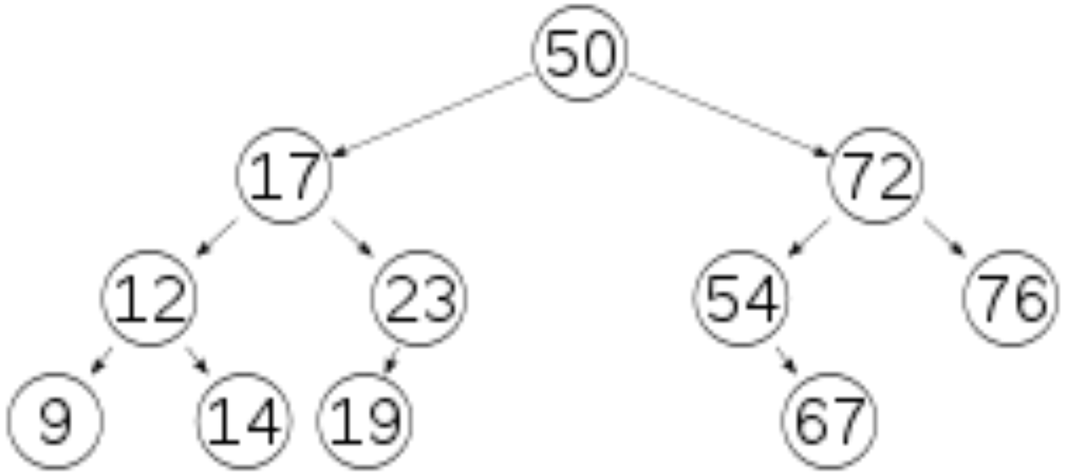
h1("cat") % 10 = 1
h2("cat") % 10 = 3
h3("cat") % 10 = 7



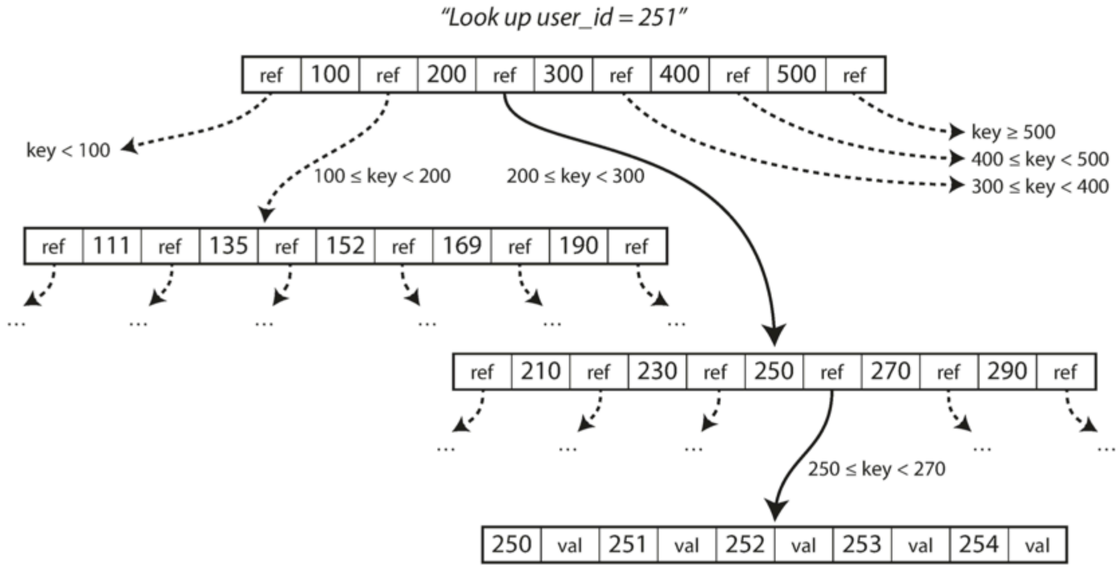
Data indexes

- Straw-man design (bash script, get, set, append-only)
  - Fast write
  - Slow read
  - Large storage space.
- Hashtable (all keys in the memory, all values on the disk, background compaction)
  - Fast write & read
  - Less storage space
  - All keys need to fit in memory.
- **SSTable (HashTable + Sorted Segment + Sparse keys in the memory)**
  - **Works even if the size of keys in dataset is bigger than the memory.**
  - **Good performance for ranging queries as well.**
  - **Further compression**

B-tree



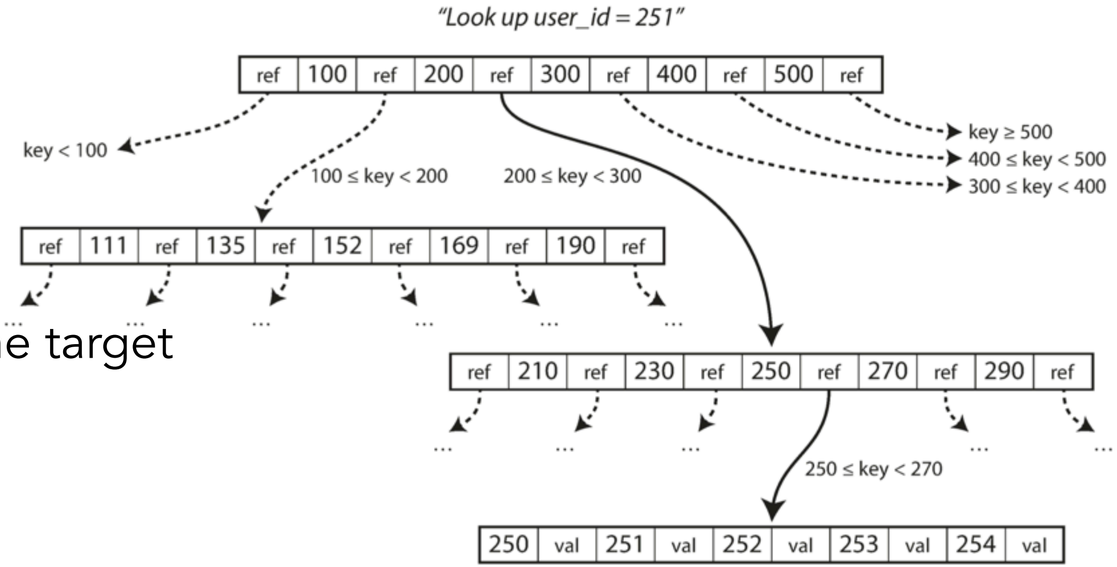
Self-balanced BST



B-Tree

B-tree

- Corresponds more closely to the underlying hardware, as disks are also arranged in fixed-size blocks.
- Root = kept in main memory.
  - Loaded into memory when needed.
- Not append only.
  - Search for the leaf page containing the target key
  - Change **the value** in that page
  - Write the page back to disk.
  - Do not change the references.



B-Tree

Recall Lecture 4 (Memory hierachy):

What's Inside A Disk Drive?

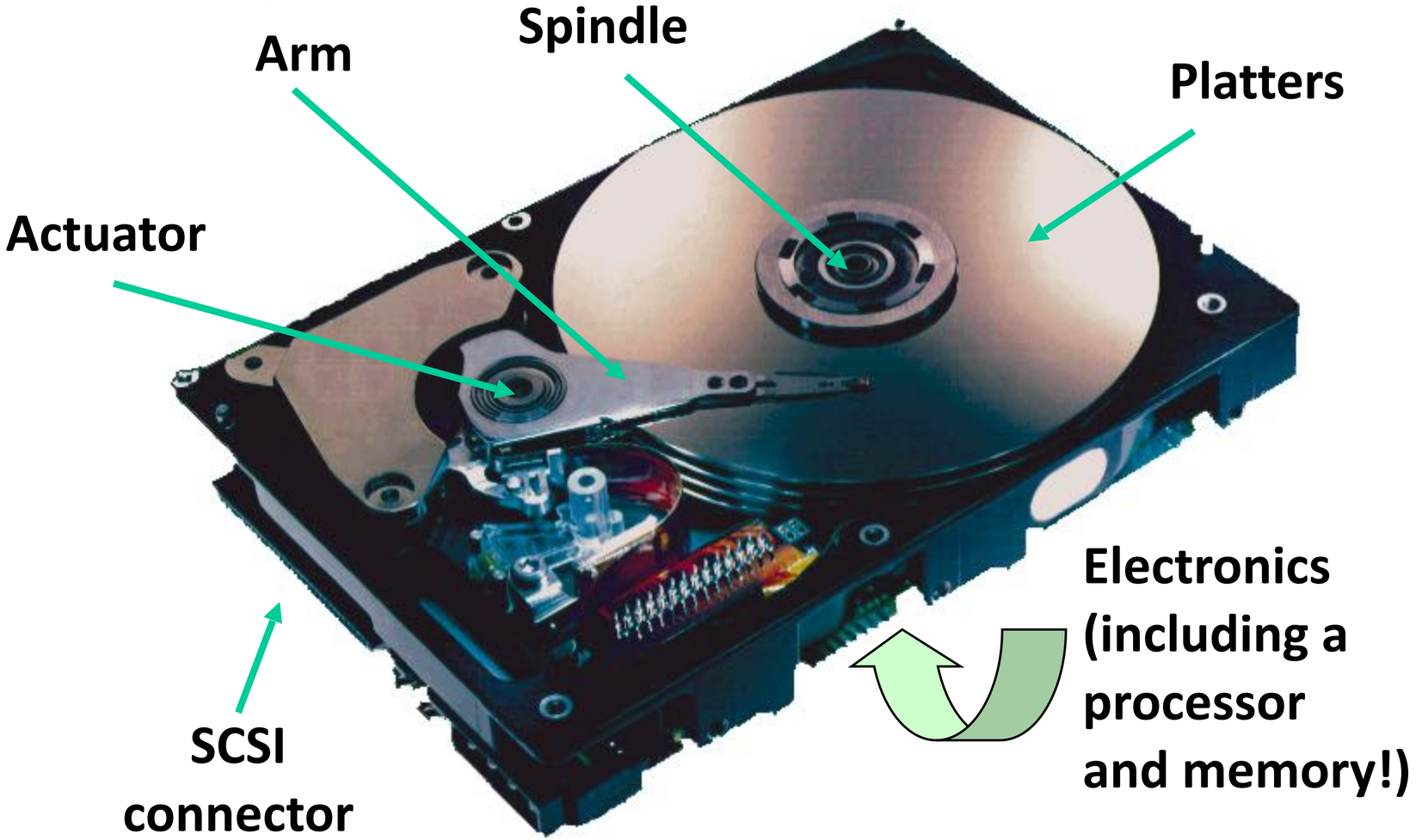
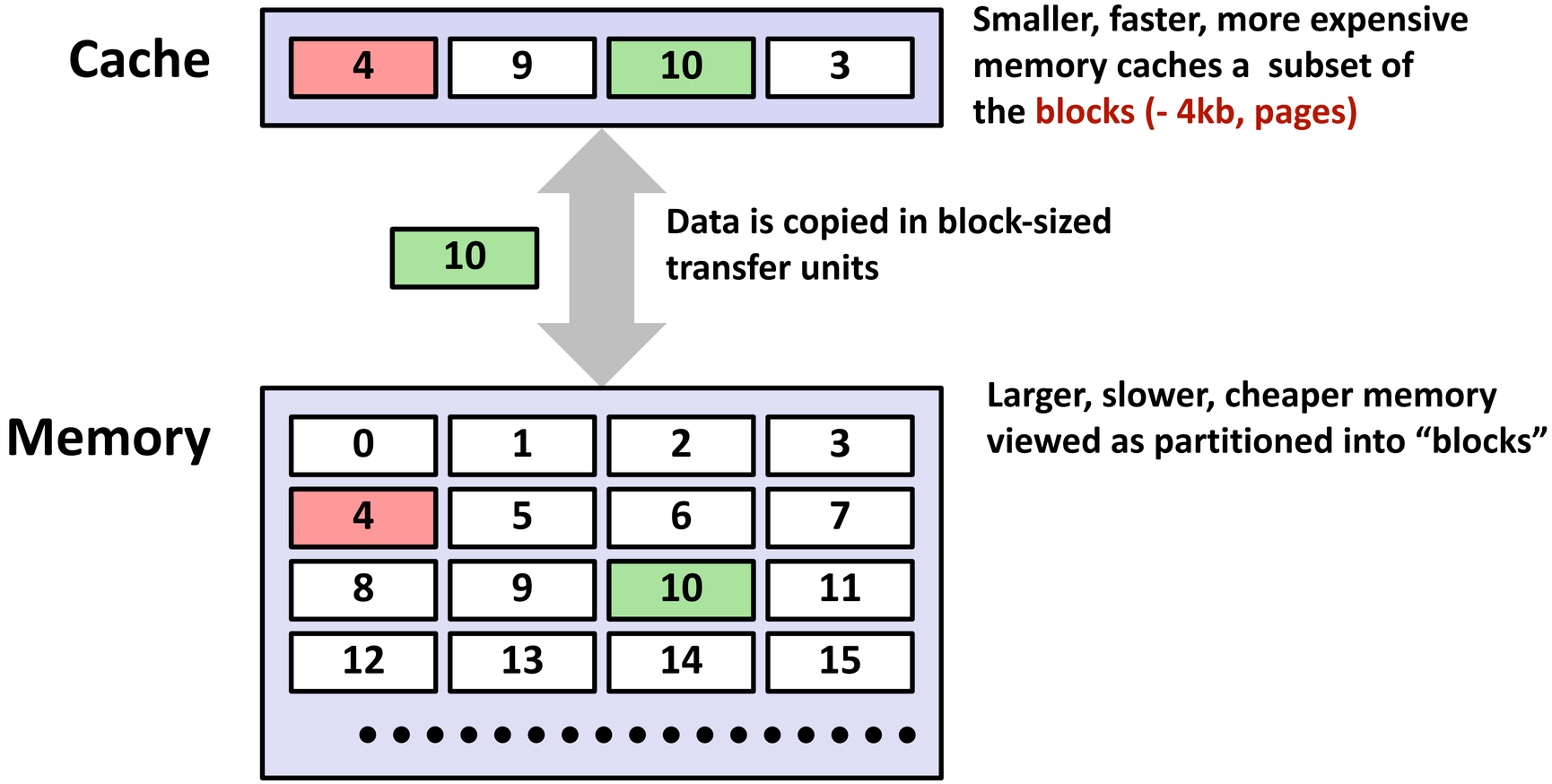


Image courtesy of Seagate Technology

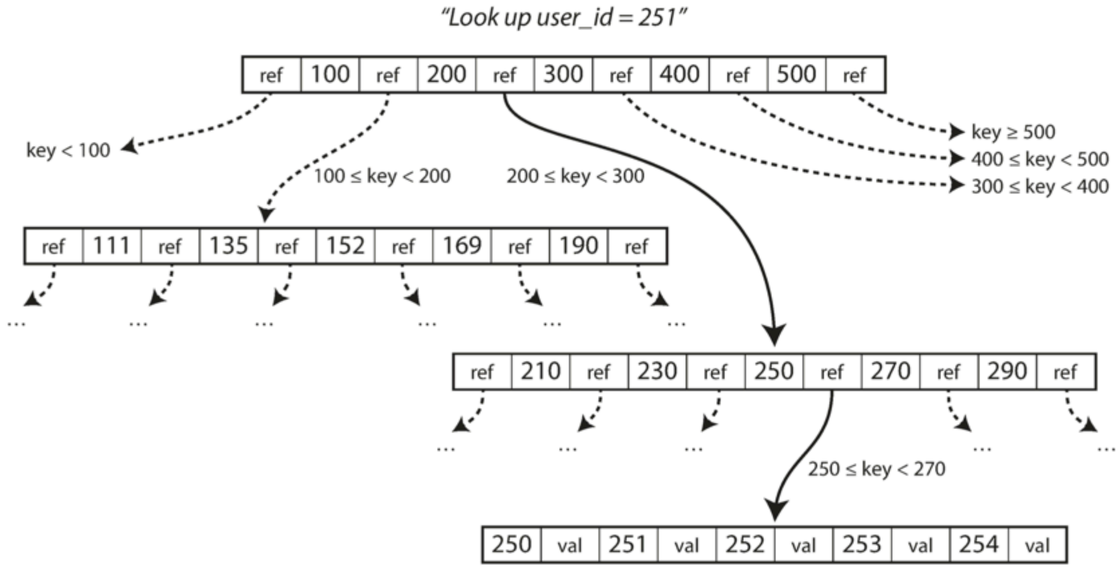
Recall Lecture 4 (Memory hierachy):

General Cache Concepts



B-tree

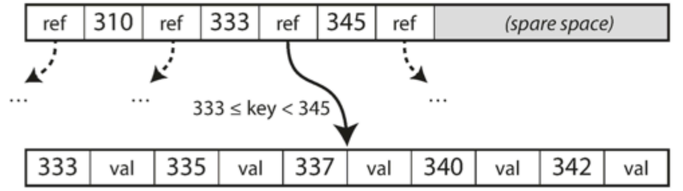
- Branching factors:
  - The number of references to child pages.
    - Typically several hundred.
- I/O is proportional to tree height.
  - Height can be less than BST.
- Fit more volume of data into the memory.
  - Most DBs are 3 or 4 levels deep.
  - A four-level tree of 4KB pages with a branching factor of 512 can store up to 256 TB.
    - (512^4) x 4kb = 256 TB (Disk)
    - Memory?
- B-tree was invented in 1970s.



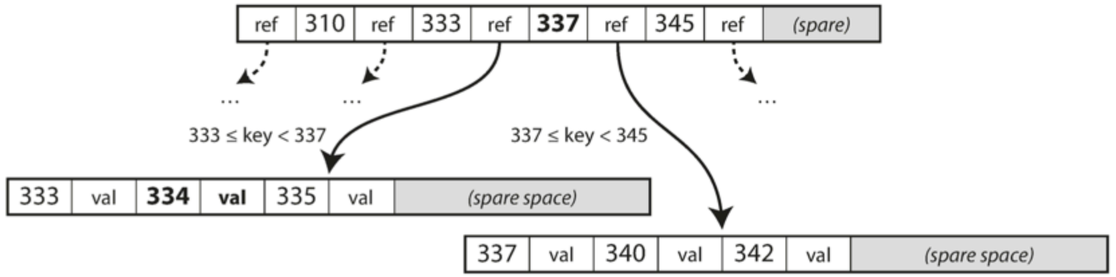
B-Tree

Page splitting in B-tree

- What if we want to add a key and there is not enough space?
  - Split a page in a B-tree.
- B-tree is also a self-balance tree.



After adding key 334:



LSM-trees v.s. B-trees

- LSM-Trees
  - Faster for writes
    - Append-only
  - Slower for reads
    - Need to check multiple data structures
      - At different stages of compactions
  - Better compression
  - Higher CPU usages
  - What if write too fast? => Compaction configuration.
- B-trees
  - Faster for reads
    - Consistent data structure.
  - Slower for writes
    - Need to write to a log to address the implications of append-only.
  - Storage Fragmentation

In-memory database

- Why so much complexity?
  - Magnetic Disks and SSDs are awkward to deal with.
  - Slow, Donot support random address access.
  - But they are durable/persistent and cheap.
- New trends
  - RAM becomes cheaper and larger.
  - Battery powered RAM.
- In-memory database
  - Memcached, Memsql, Oracle TimesTen, Redis

In-memory database

- Multiple implementations.
  - Use in-memory database for caches only
  - Use disks as an append-only log only.
- Advantages
  - **Counter intuitive!**
    - **Not because disk is slower.**
      - **Modern OSs do caching well.**
    - **Because of the data serialization.**
      - **Data representations in the memory and the disk**
    - **Simpler implementations.**
    - **Cost: Disk < Memory < Developers**