https://hao-ai-lab.github.io/dsc204a-w24/

# DSC 204A: Scalable Data Systems
# Winter 2024

Machine Learning Systems

Big Data

Cloud

Foundations of Data Systems

# Data indexes

- Straw-man design (bash script, get, set, append-only)
  - Fast write
  - Slow read
  - Large storage space.
- Hash table (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
  - Segments are sorted
  - Index is sparse and small
  - Compaction is based on merge sort
  - But how to get the segments sorted in first place?

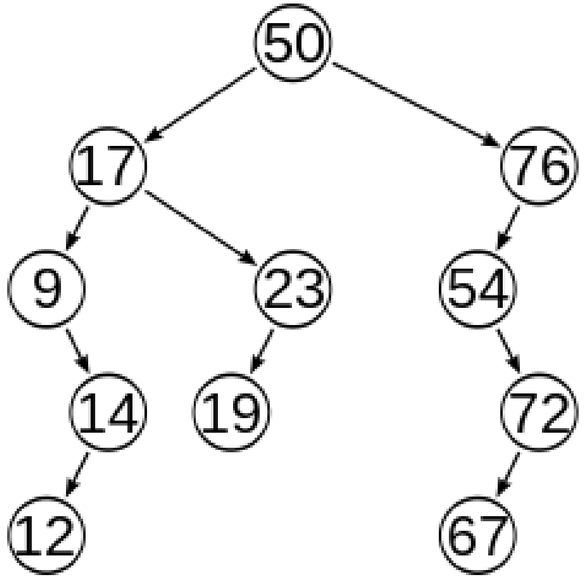
How do you get your data to be sorted by key in the first place?

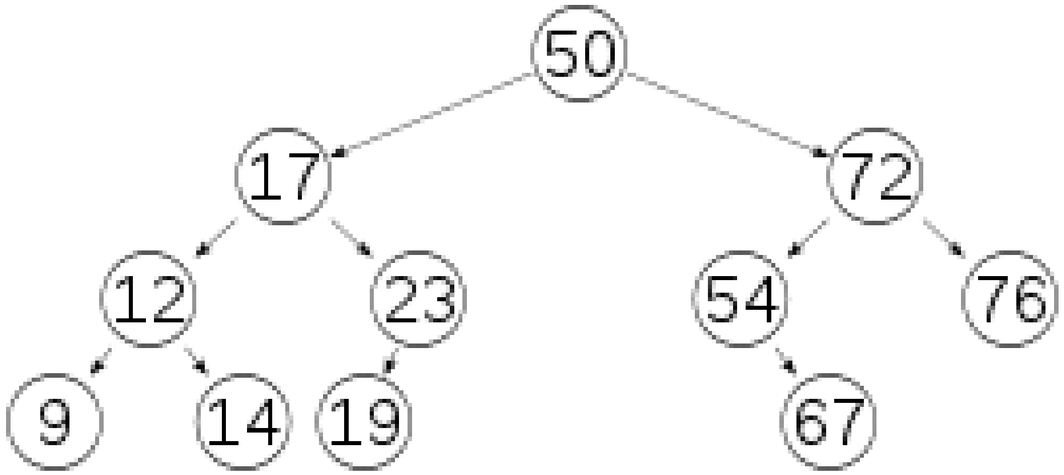
# LSMTable: Augment SSTable with MemTable

- 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)

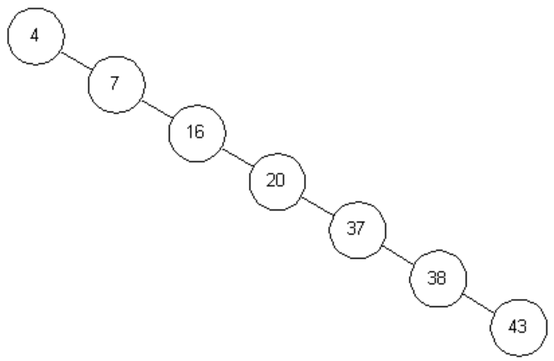


# AVL v.s Binary Search Tree

| 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.
  - Merge Sort

# 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.

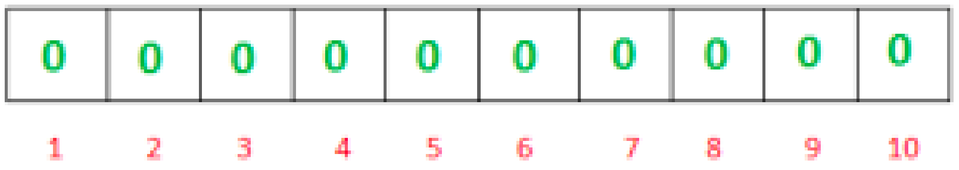
Initialization (m):

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

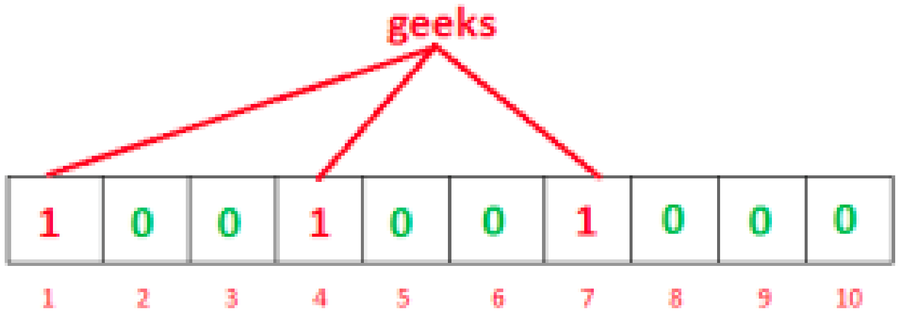
Three hashing functions (k): h1, h2, h3

# Bloom filters (read and write)

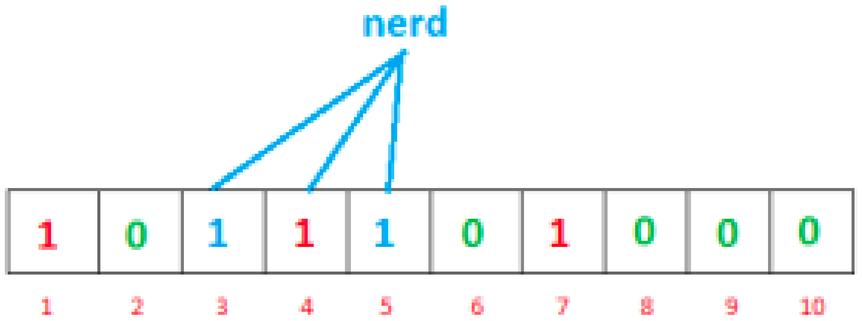
A set of words: {“geeks”, “nerd”}




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



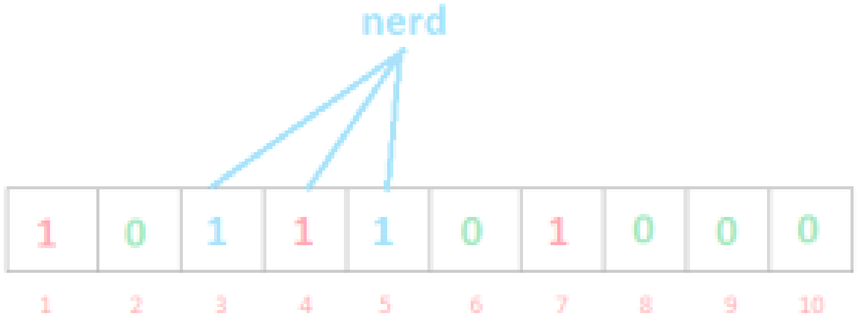

h1(“nerd”) % 10 = 3
h2(“nerd”) % 10 = 5
h3(“nerd”) % 10 = 4



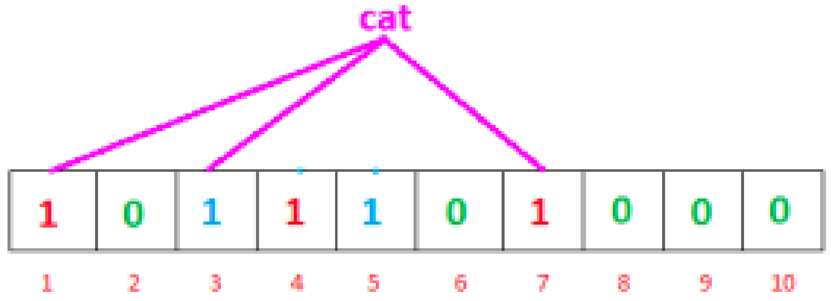
# Bloom filters - False positive

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

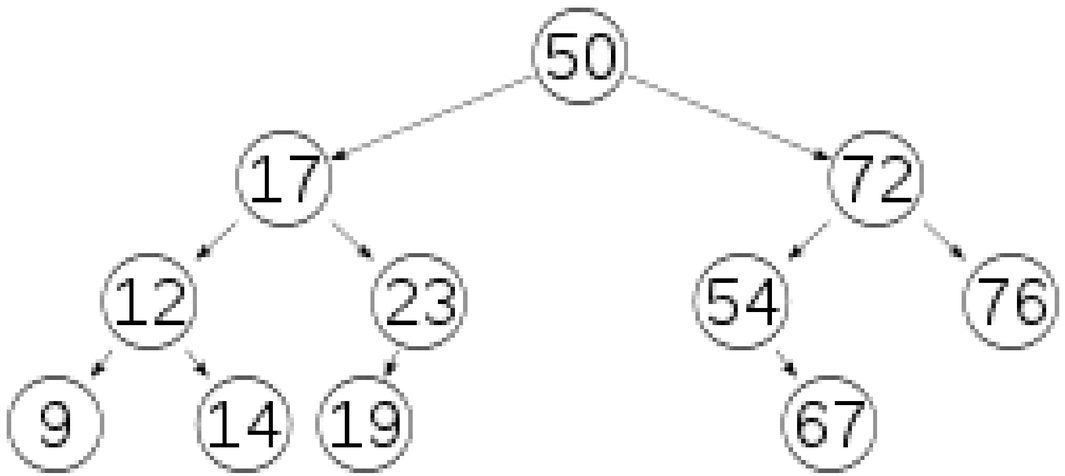

h1(“nerd”) % 10 = 3
h2(“nerd”) % 10 = 5
h3(“nerd”) % 10 = 4



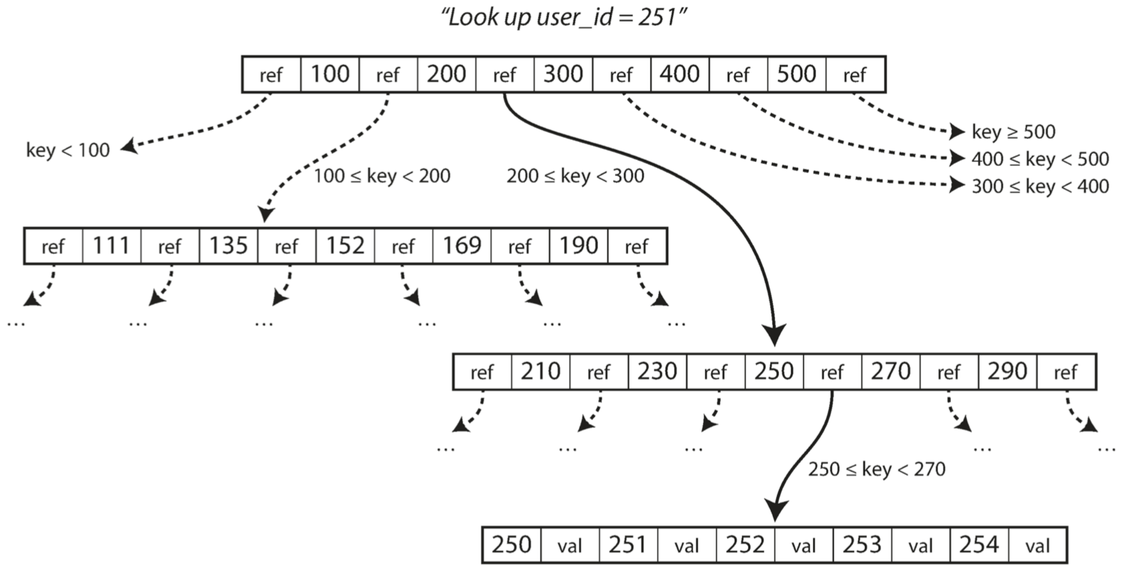

h1(“cat”) % 10 = 1
h2(“cat”) % 10 = 3
h3(“cat”) % 10 = 7



# B-tree



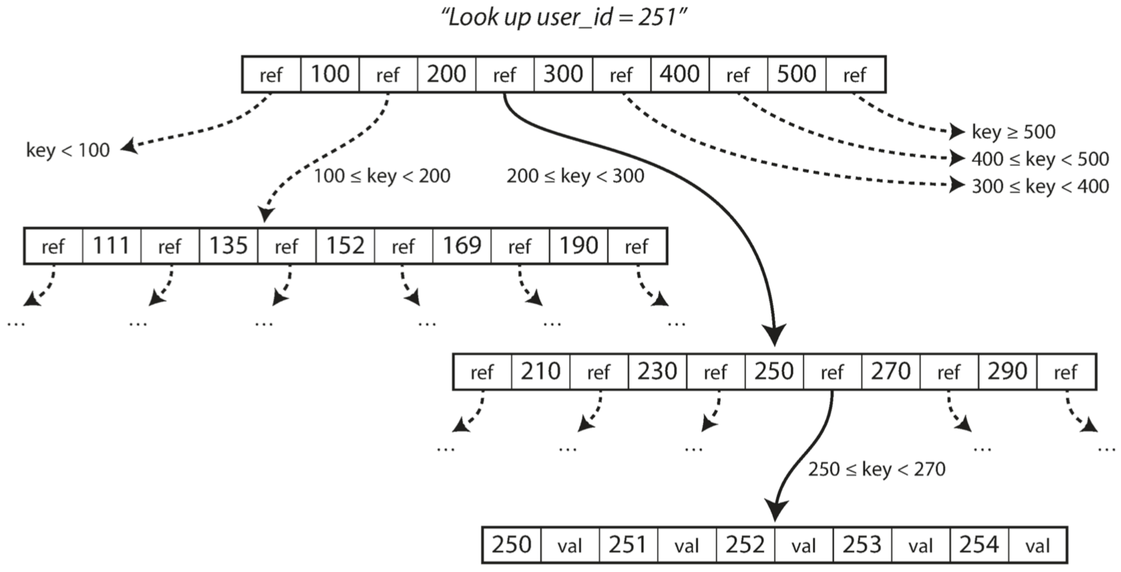

Self-balanced BST




B-Tree

# B-tree (Will not be in Exam)

- 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

# Trends: In-memory database

Why so much complexity?

- Magnetic Disks and SSDs are awkward to deal with.
- Slow, Do not support random address access.
- But they are durable/persistent and cheap.

Hardware trends

- RAM becomes cheaper and larger.
- Battery powered RAM.

Notable In-memory database implementation

- Memcached, Memsql, Oracle TimesTen, Redis

Advantages

- Not because disk is slower.
  - Modern OSs do caching well.
- Reason 1: data serialization is eliminated
  - Data representations in the memory and the disk
- Reason 2: Simpler implementations.
  - Cost: Disk < Memory < Developers (you?)

# Next

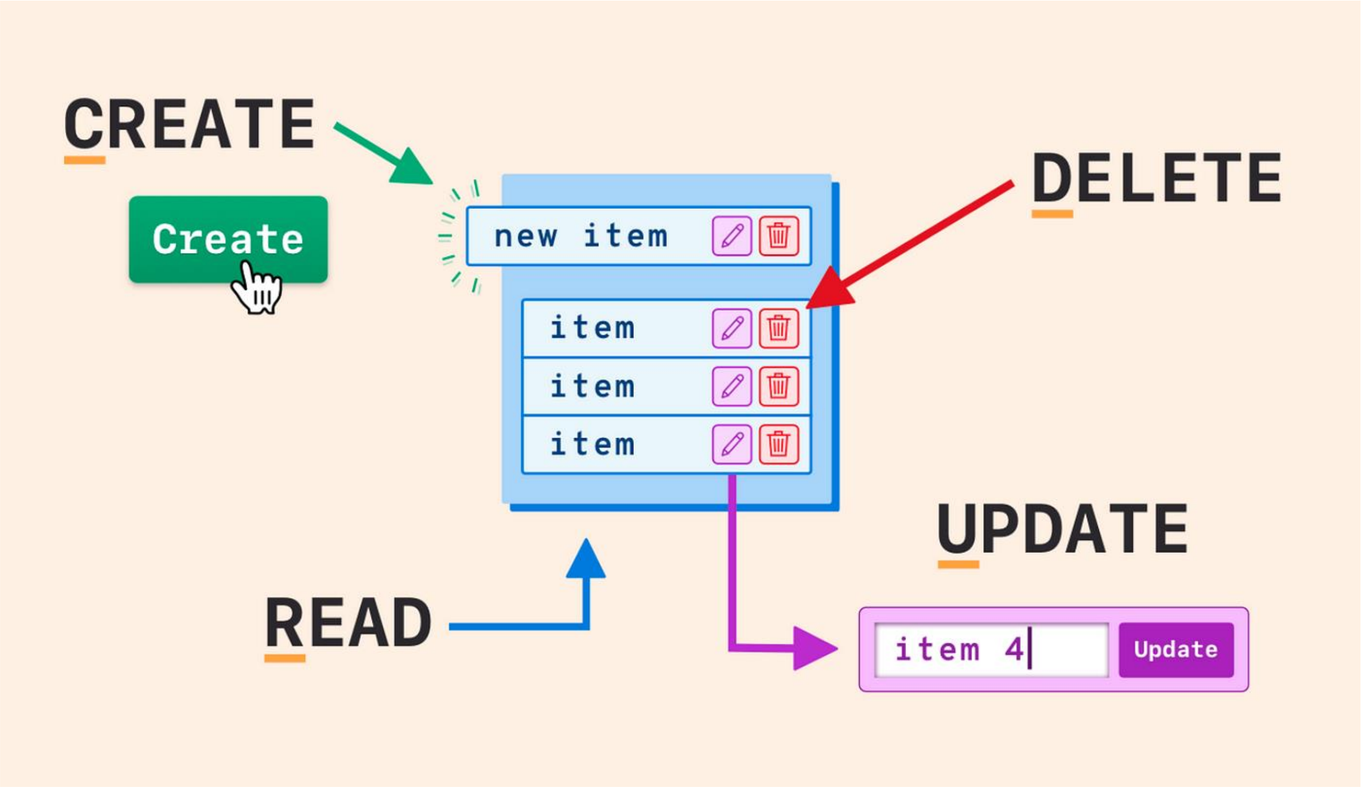
- File system
- Database
- **Data Warehouse and Column Storage**

# Data Warehouse and Column Storage

- OLTP v.s. OLAP
- Data warehousing
- Schemas for Analytics
- Column-oriented storage
- Data cubes and materialized views

# CRUD


I'm a Database Developer,
all what I do is
Create
Read
Update
Delete


CREATE
Create
new item
item
item
item
DELETE
UPDATE
READ
item 4
Update

# Database transactions

- Make sale
- Place an order
- Pay an employee's salary
- Comment a blog post
- Act in games
- Add/remove contract to an address book

Online transaction processing (OLTP)

# Walmart Beer and Diaper (1988)

Forbes 1988

- Unexpected correlation:
  - Sales of diapers and beer



https://www.forbes.com/forbes/1998/0406/6107128a.html?sh=2574a9316260

# Data analytics

- What was the total revenue of each of our stores in Jan?
- How many more bananas that usual did we sell during our latest data?
- Which brand of baby food is most often purchased together with brand X diapers?

Online analytic processing (OLAP)

# OLTP v.s. OLAP

| Property | Transaction processing systems (OLTP) | Analytic systems (OLAP) |
|---|---|---|
| Main read pattern | Small number of records per query, fetched by key | Aggregate over large number of records |

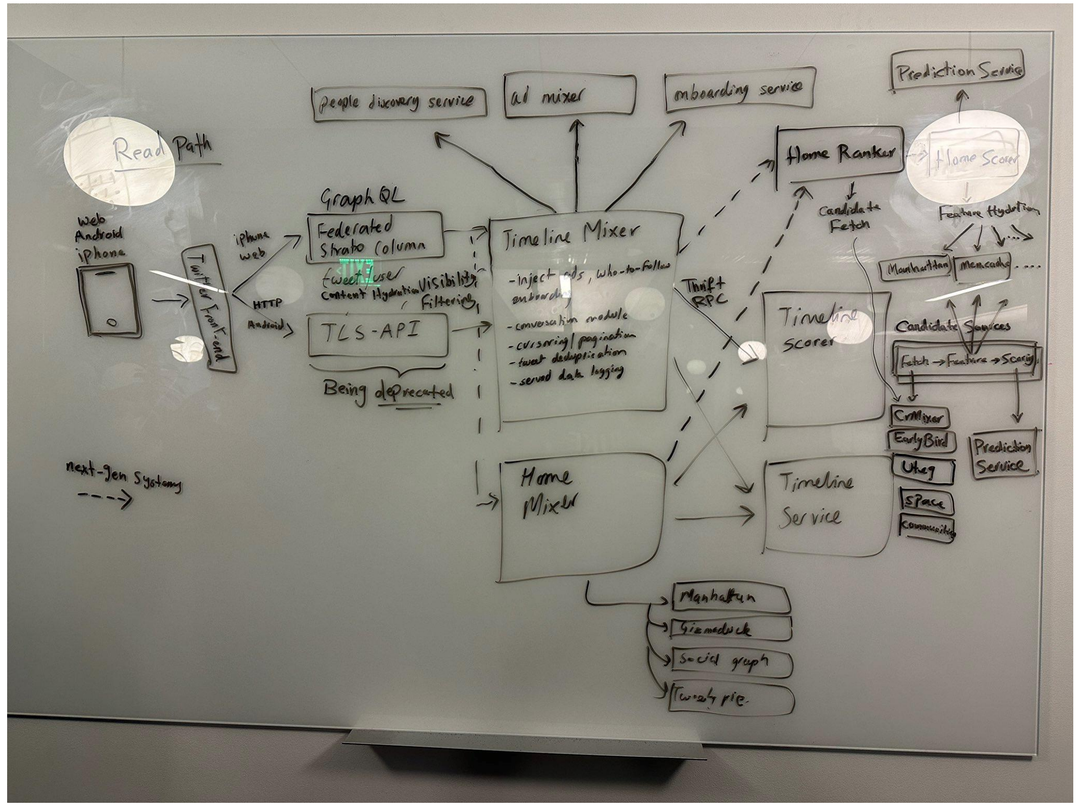
# OLTP v.s. OLAP

| Property | Transaction processing systems (OLTP) | Analytic systems (OLAP) |
|---|---|---|
| Main read pattern | Small number of records per query, fetched by key | Aggregate over large number of records |
| Main write pattern | Random-access, low-latency writes from user input | Bulk import (ETL) or event stream |
| Primarily used by | End user/customer, via web application | Internal analyst, for decision support |
| What data represents | Latest state of data (current point in time) | History of events that happened over time |
| Dataset size | Gigabytes to terabytes | Terabytes to petabytes |

# Today's topic

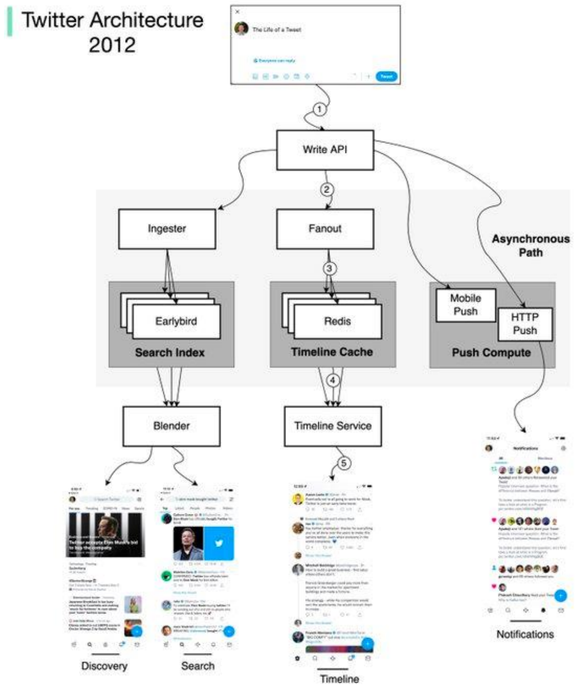
- OLTP v.s. OLAP
- Data warehousing
- Schemas for Analytics
- Column-oriented storage

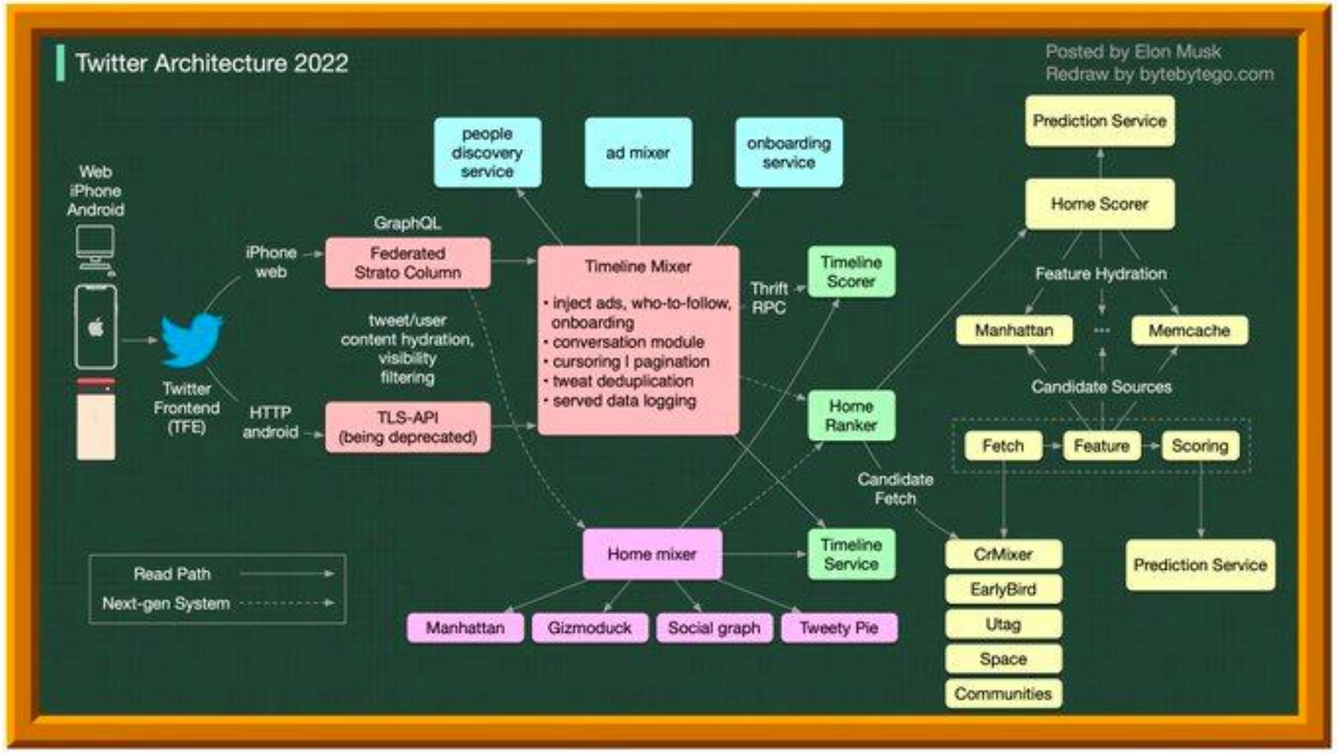
# Transaction systems are complex.


Elon Musk's Twitter System Design Diagram Explained
https://www.youtube.com/watch?v=_Y5aGCOkymQ

# Transaction systems need to be highly available.



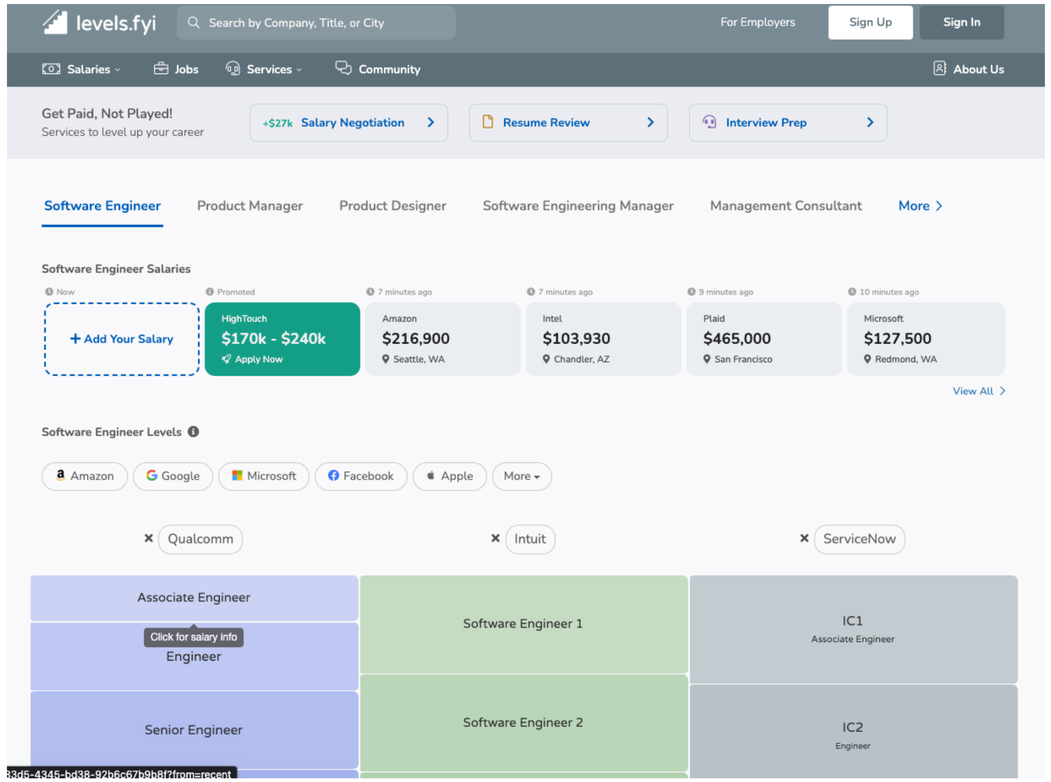

- Low latency.
- Highly available.
- Ad hoc analytic queries are expensive.

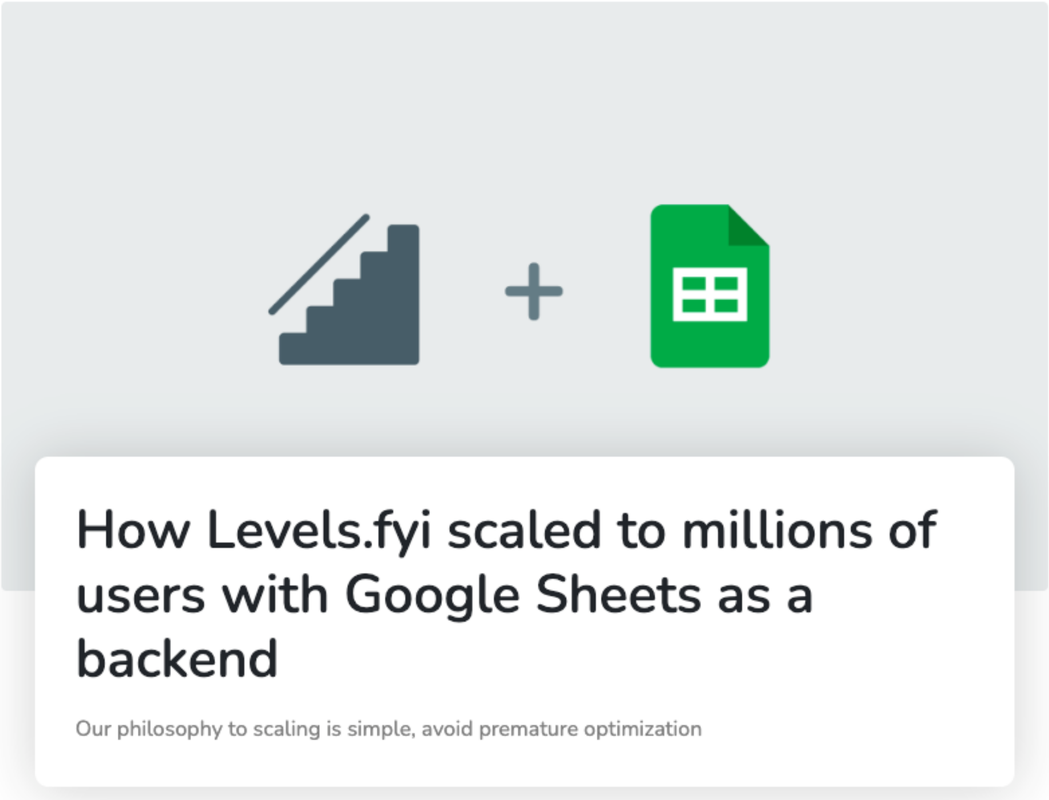
https://twitter.com/alexxubyte/status/1594008281340530688

# Data warehouse

- A separate database that analysts can query to their hearts' content, without affecting OLTP operations.
- Maintain a read-only copy for analytic purposes.
- Only exist in almost all large enterprises.

# Small companies?

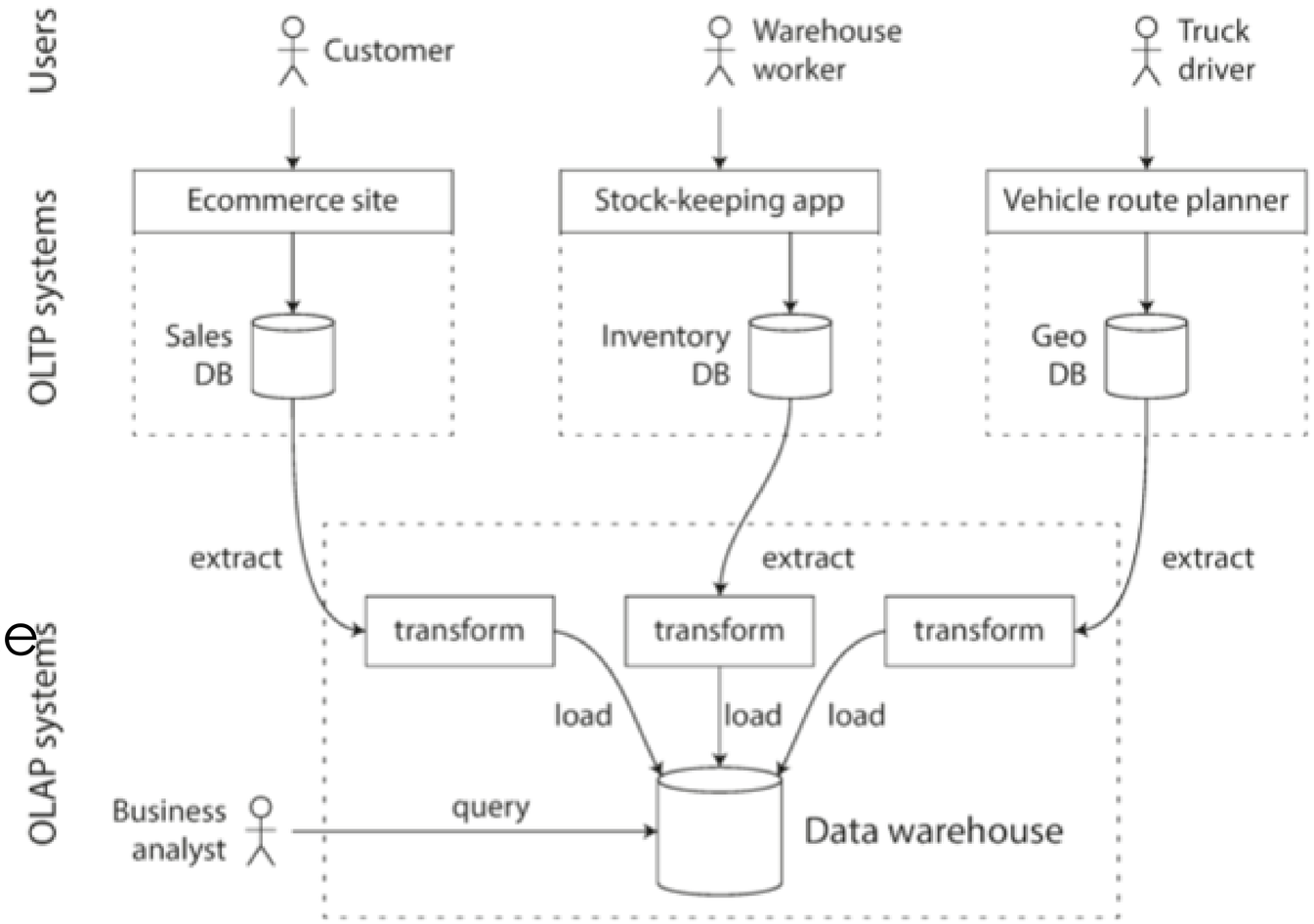



https://www.levels.fyi/blog/scaling-to-millions-with-google-sheets.html

# Extract-Transform-Load (ETL)

- Extract
  - Periodica data dump
  - Continuous streaming
- Transform
  - Analysis-friendly schema
  - Data cleaning
- Load into a data warehouse



# Why data warehouse?

- Separation of concerns
  - Performance (reliability, latency)
  - Expertise requirement, management
- The indexes in last lecture (e.g., SSTable, B-tree) are good for reading and writing a single record.
  - But are not good at answering analytic queries.

# How do you interact with OLAP & OLTP

- SQL query interface
  - *Select * from*
  - "A database system can be considered mature when it has an SQL query interface".
  - Both OLAP and OLTP
- OLAP:
  - More and more codeless user interfaces.
  - Note: This is a big market of innovations

# More Stories?

TRIFACTA

400M

Streamlit

800M / ~30 persons

databricks

43B

snowflake®

73B