CS 6400 A

Database Systems Concepts and Design

Lecture 25

11/24/25

Announcements

- Assignment 3 due today
- Project final report and code due next Monday
- Schedule this week:
  - No lecture on Wednesday; no office hour
- Final exam (take home):
  - Released Dec 4 (Thursday) at 3PM; due Dec 5 (Friday) at 9PM.
  - Practice exam on canvas
  - Review lecture next Monday

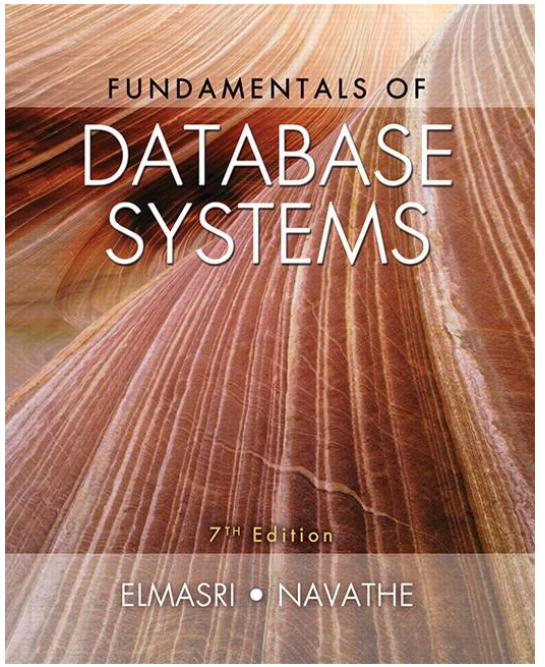
Agenda

1. NoSQL
2. Course Summary

Reading Materials

Fundamental of Database Systems (7th Edition)

- Chapter 24: NOSQL Databases and Big Data Storage Systems



Research Papers:

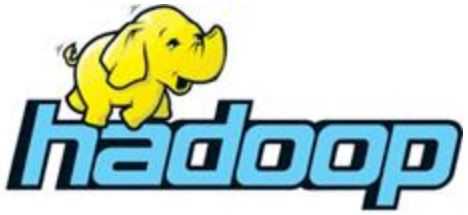
- The LSM Tree Paper

Review: SQL history and motivation

Initially developed in the early 1970's

By 1986, ANSI and ISO standard groups standardize SQL

- New versions of standard published in 1989, 1992, and more up to 2016



"Dark times" in 2000s

- NoSQL for Web 2.0
- Google's BigTable, Amazon's Dynamo
- Are relational databases dead?

NewSQL systems in 2010s

- Bring back ACID

1. NoSQL

NoSQL

NoSQL stands for “Not Only SQL” or “Not Relational”

- relax on consistency requirements to gain efficiency and scalability

Designed to scale simple “OLTP”-style application loads

- to provide good horizontal scalability for simple read/write database operations distributed over many servers

Originally motivated by Web 2.0 applications

- these systems are designed to scale to thousands or millions of users

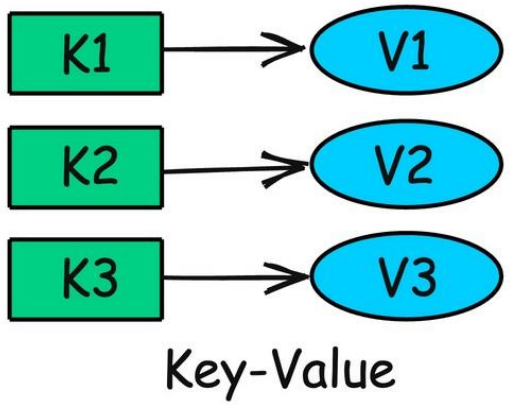
Example NoSQL systems

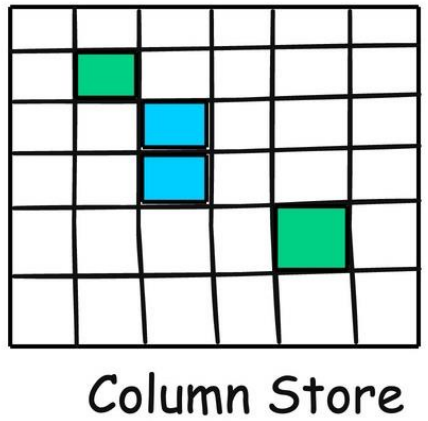
Key-value store:
DynamoDB, Redis

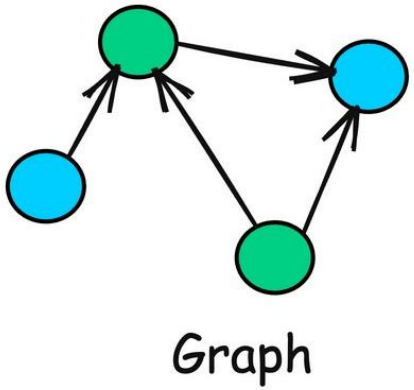
Document store:
MongoDB, CouchDB

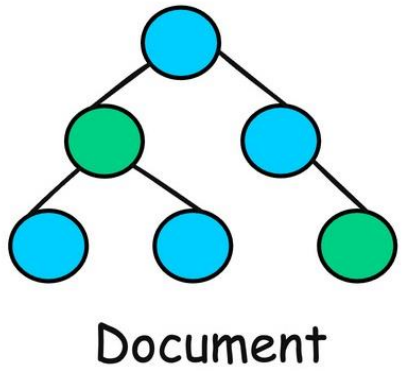
Wide-column stores:
Cassandra, BigTable

Graph database:
Neo4j, AWS Neptune









NoSQL vs RDBMS

Two important distinctions:

Consistency Model:

- ACID (Strong Consistency) vs BASE (Eventual Consistency)

Data structure:

- B+ tree vs LSM Tree

Problem with ACID

Recall ACID for RDBMS desired properties of transactions

Challenge: maintaining ACID across multiple servers requires coordination

- Coordination = latency + reduced availability

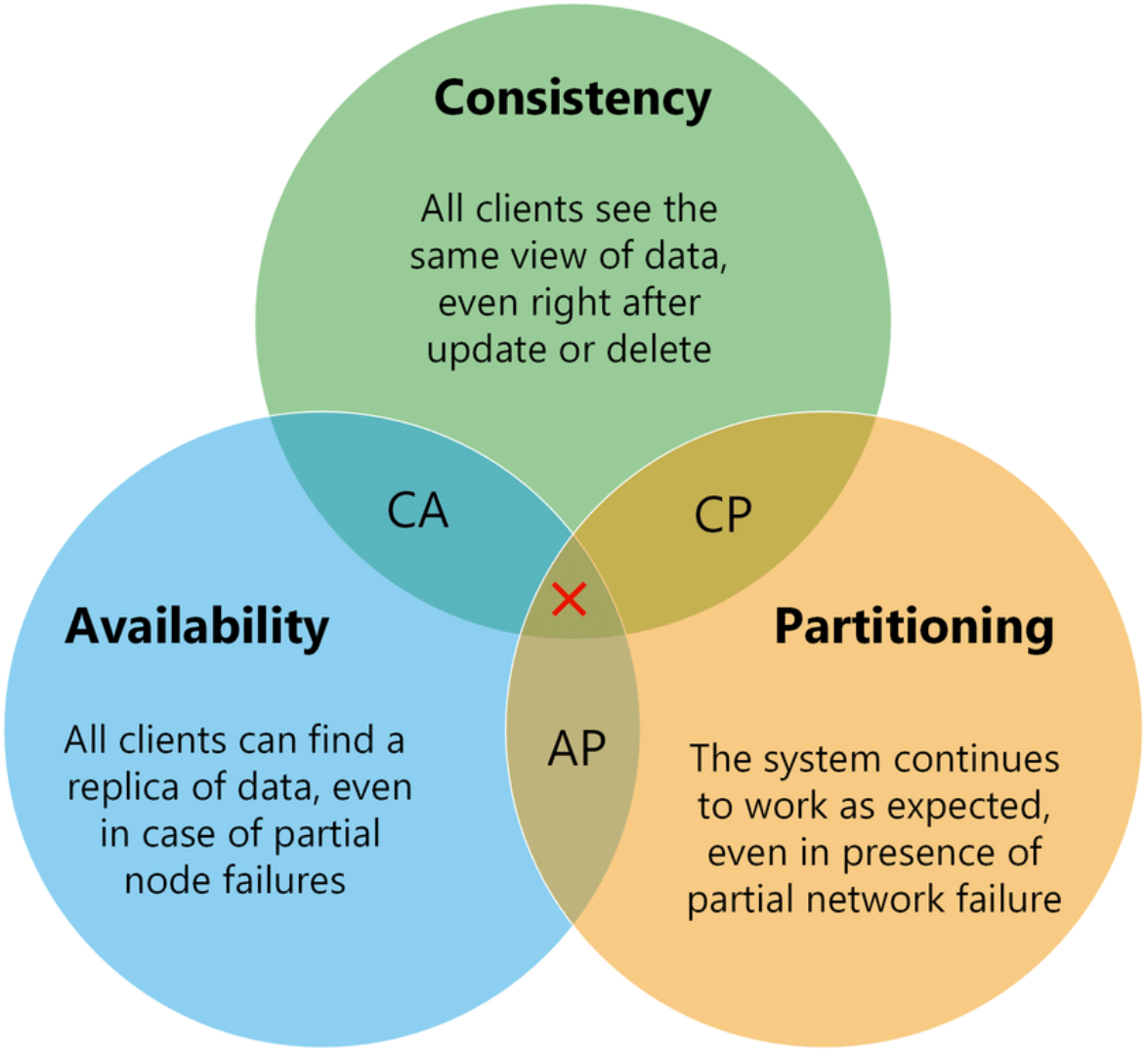
Example: adding an item to shopping cart

- Option 1: System says "I can't guarantee your cart is consistent across all servers, so I'm refusing your request"
- Option 2: System says "I'll accept your cart addition on this datacenter, we'll sync up later"

CAP Theorem

Any distributed data store can provide only two of the following three guarantees: **C**onsistency, **A**vailability, and **P**artition-tolerance.

AP: eventual consistency
CP: strong consistency



NoSQL systems generally give up consistency (AP system)

Image source: https://medium.com/nerd-for-tech/understand-cap-theorem-751f0672890e

ACID vs BASE

NOSQL systems typically do not provide ACID: they sacrifice strong consistency for availability/performance

- Basically Available
- Soft state
- Eventually consistent

The systems differ in how much they give up

- e.g., most of the systems call themselves “eventually consistent”, meaning that updates are eventually propagated to all nodes
- Many modern NoSQL systems offer tunable consistency - you can choose between strong consistency (more ACID-like) and eventual consistency (BASE) depending on your need

Problem with B+-Trees

Optimized for reads, not for writes (insert, update, delete)

Write amplification:

- ratio of the amount of data written to the storage device versus the amount of data written to the database
- Inserting and deleting a record could require updating multiple pages on disk (many random disk I/Os)

NoSQL systems face extreme *write workloads*:

- High write throughput: social media posts, IoT sensors, logs
- Distributed writes across many nodes
- Append-heavy workloads

Log-structured Merge Tree (LSM Tree)

Proposed by O'Neil, Cheng, and Gawlick in 1996

Uses write-optimized techniques to significantly speed up inserts

Used by many NoSQL systems:

- e.g., Bigtable, Cassandra, Dynamo, HBase, RocksDB, InfluxDB

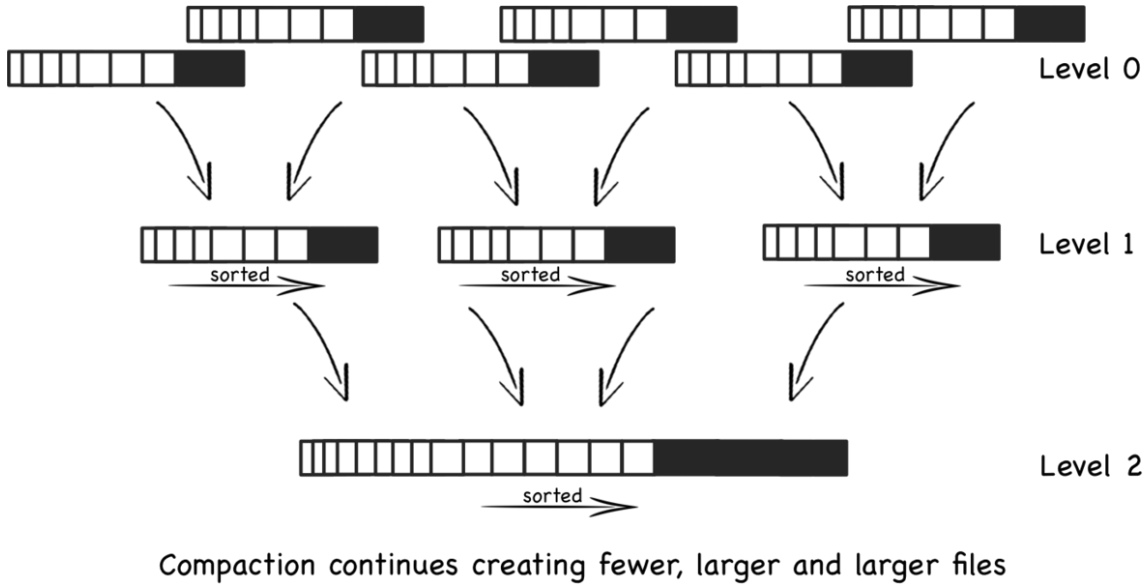


Image source: https://en.wikipedia.org/wiki/Log-structured_merge-tree

LSM Tree: Overview

Log-structured · Merge · Tree

Log-structured

Data is written sequentially in append-only fashion, like writing to a log file

Merge

As data evolves, sequentially written runs of key-value pairs are merged

- Runs of data are indexed for efficient lookup
- Merges happen only after much new data is accumulated

Tree

The hierarchy of key-value pair runs form a tree

- Searches start at the root, progress downwards

LSM Tree: Overview [O'Neil, Cheng, Gawlick '96]

An LSM-tree comprises a hierarchy of levels of increasing size

- All data inserted into in-memory tree (C_0); no I/O cost at this step
- Larger on disk levels ($C_{i>0}$) hold data that does not fit into memory

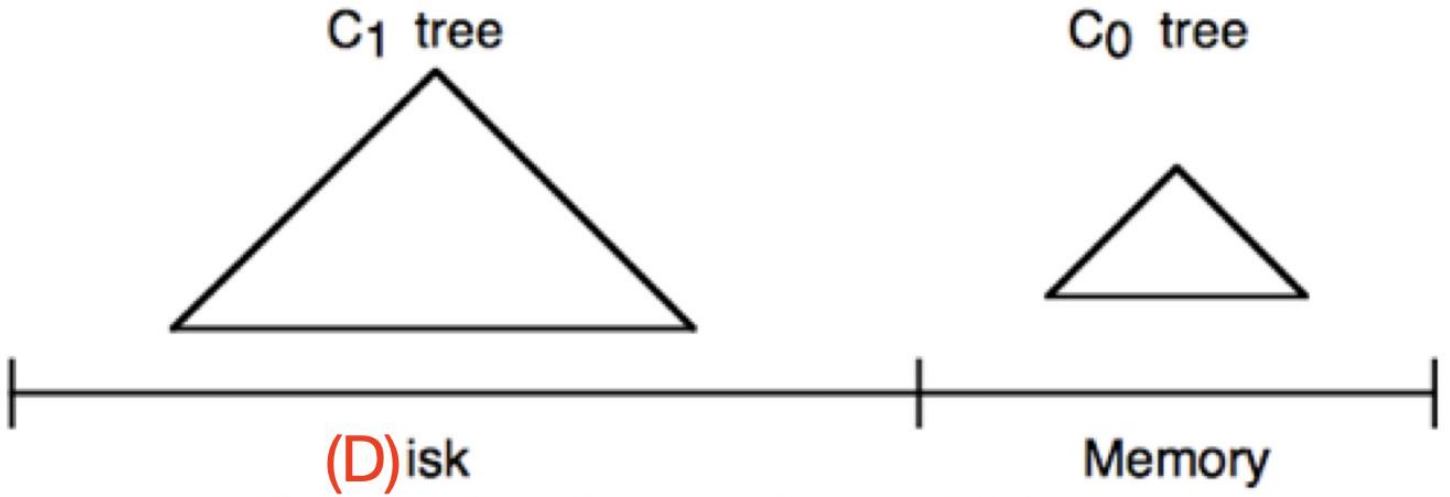


Figure 2.1. Schematic picture of an LSM-tree of two components

LSM Tree: Overview [O'Neil, Cheng, Gawlick '96]

When a level exceeds its size limit, its data is merged and rewritten

- Also called "compaction"
- Higher level is always merged into next lower level (C_i merged with C_{i+1})
- Merging always proceeds top down

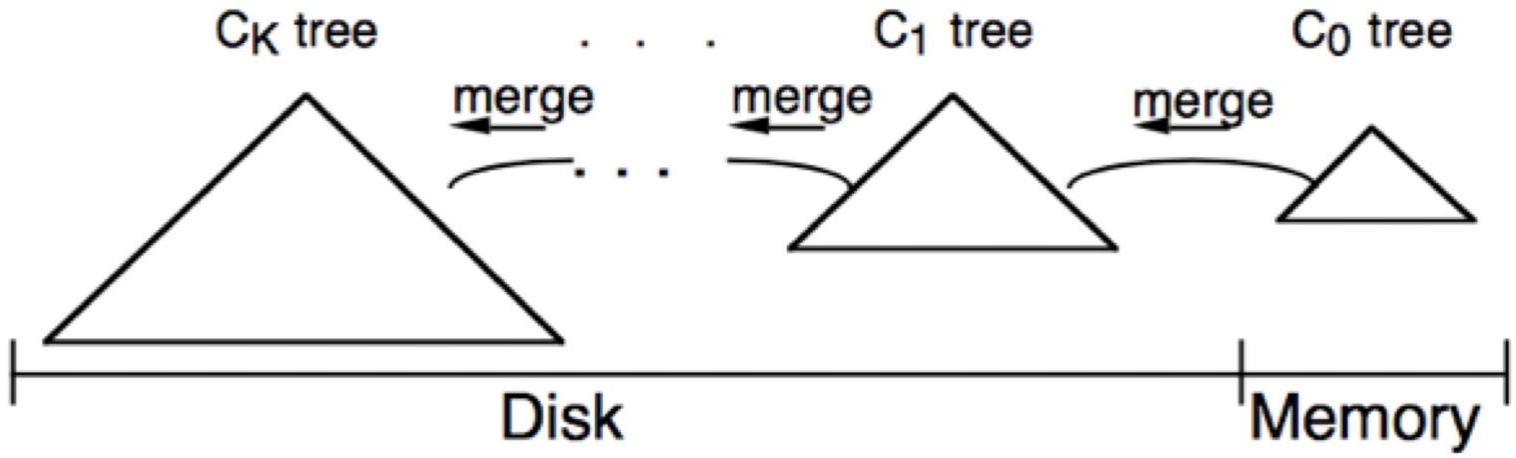


Figure 3.1. An LSM-tree of K+1 components

LSM Tree: Inserts

New data is written to an in-memory buffer (MemTable)

- typically organized as a balanced tree (like a Red-Black tree) to maintain sorted order
- Append-only log

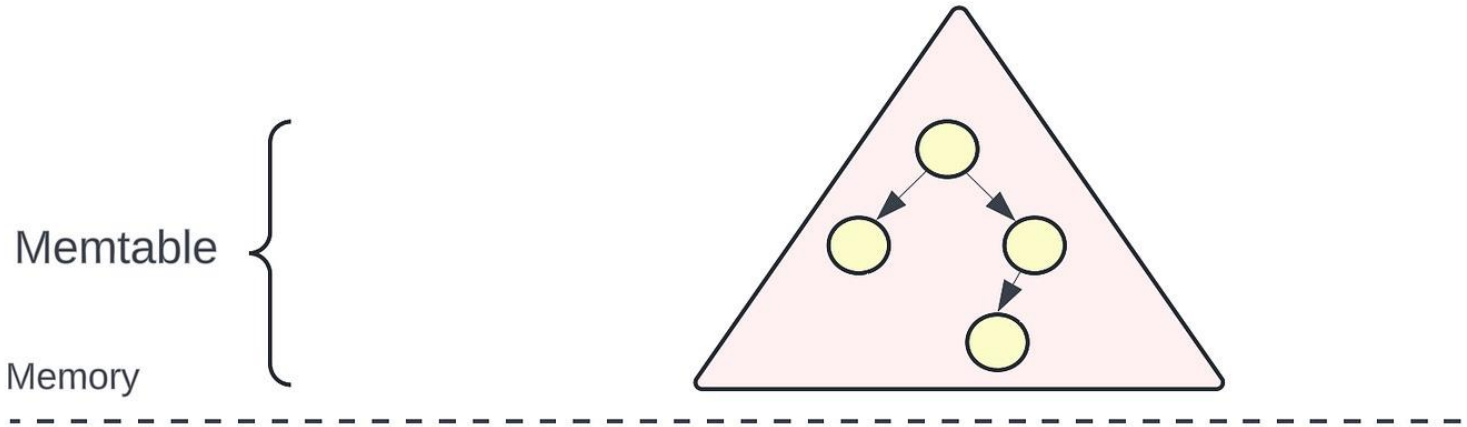


Image source: https://vivekbansal.substack.com/p/what-is-lsm-tree

LSM Tree: Inserts

New data is written to an in-memory buffer (MemTable)

- typically organized as a balanced tree (like a Red-Black tree) to maintain sorted order

When the MemTable fills up, it's flushed to disk as a new SSTable file (the “run”) in Level 0

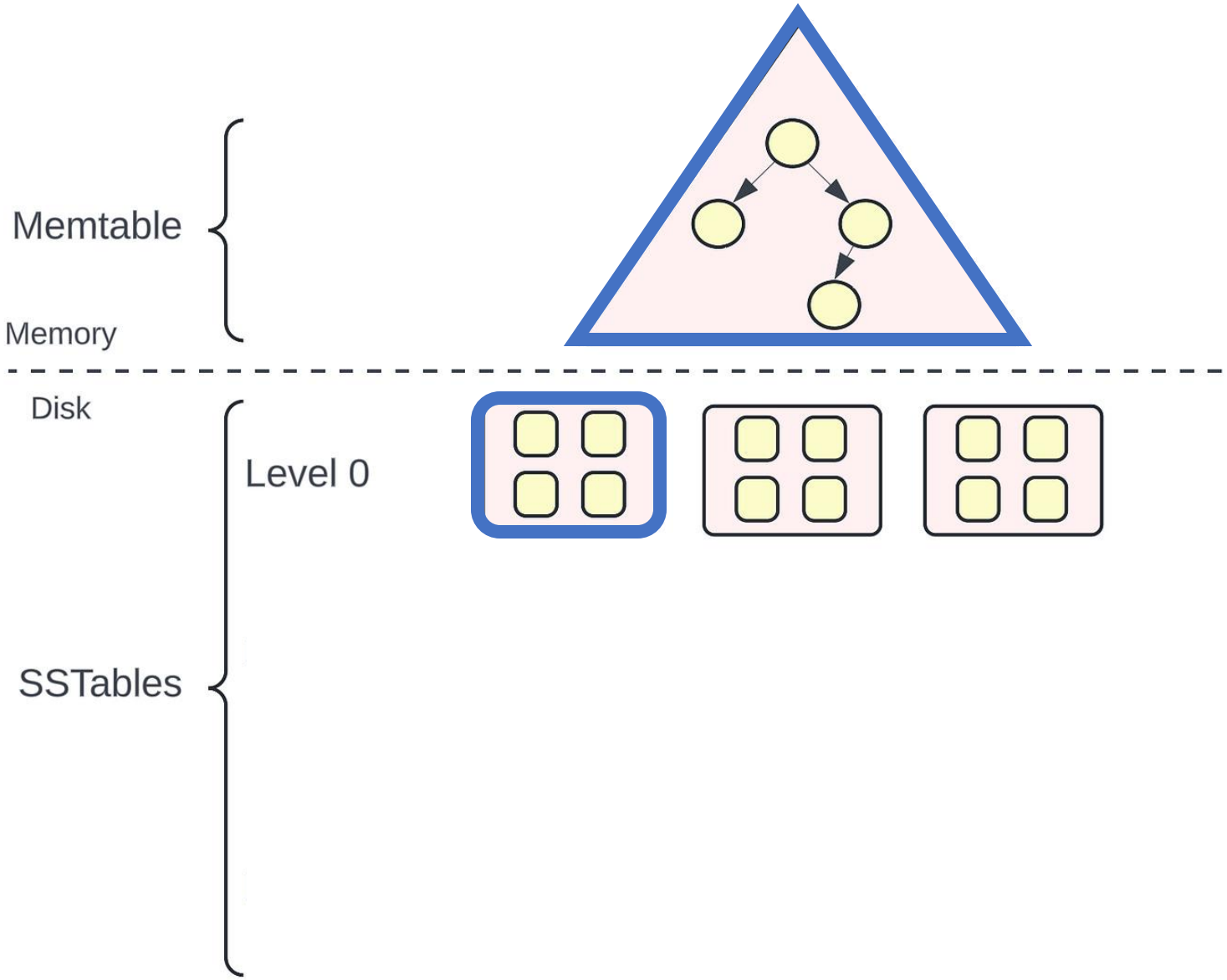


Image source: https://vivekbansal.substack.com/p/what-is-lsm-tree

LSM Tree: Inserts

Once too many L0 files exist, compaction merges them into L1

- k-way merge sort
- Input: L0 files + L1 files with overlapping key ranges
- Output: new L1 files with non-overlapping ranges

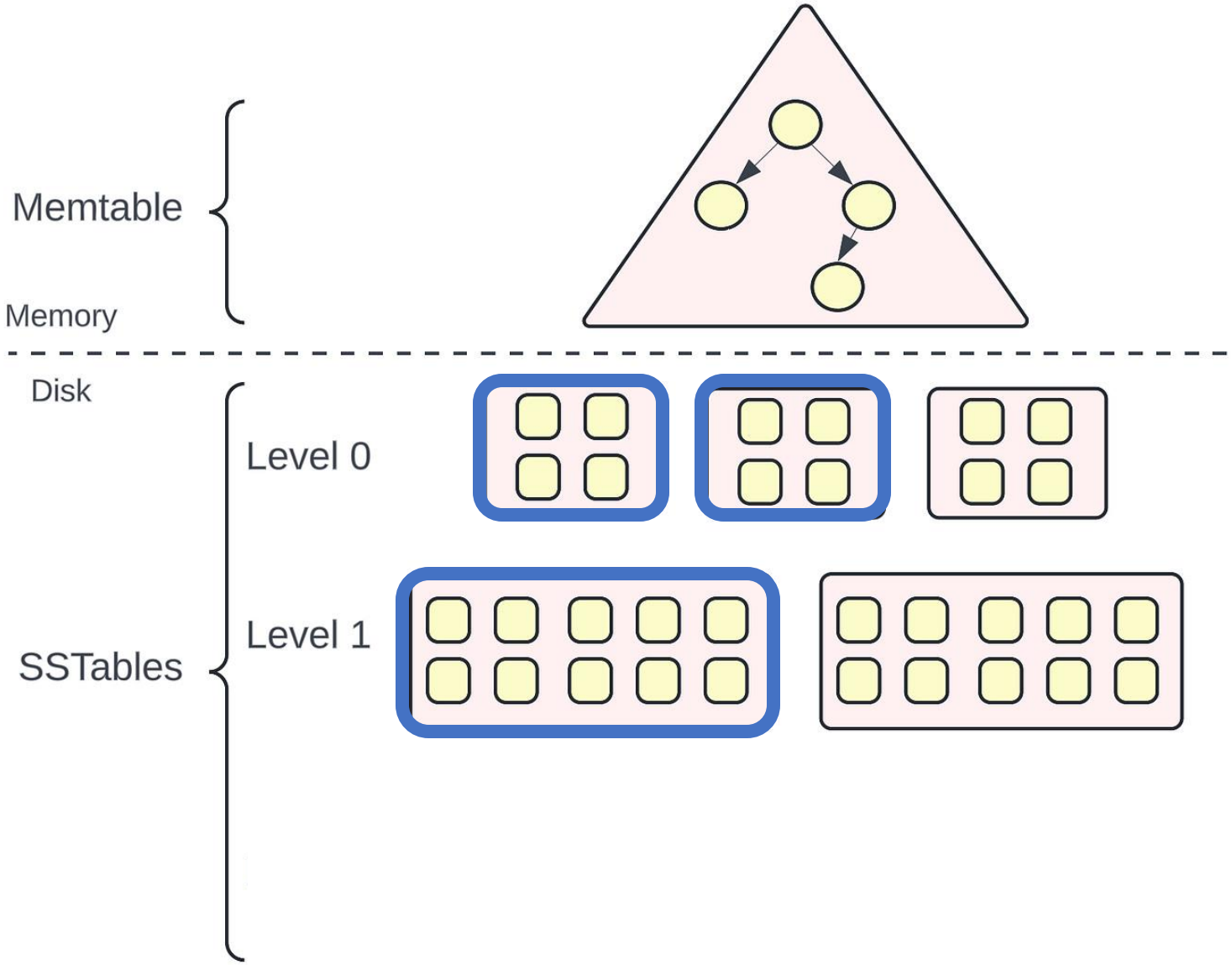


Image source: https://vivekbansal.substack.com/p/what-is-lsm-tree

LSM Tree: Inserts

Once too many L0 files exist, compaction merges them into L1

Similarly, when there are too many L1 files, compaction merges them into L2

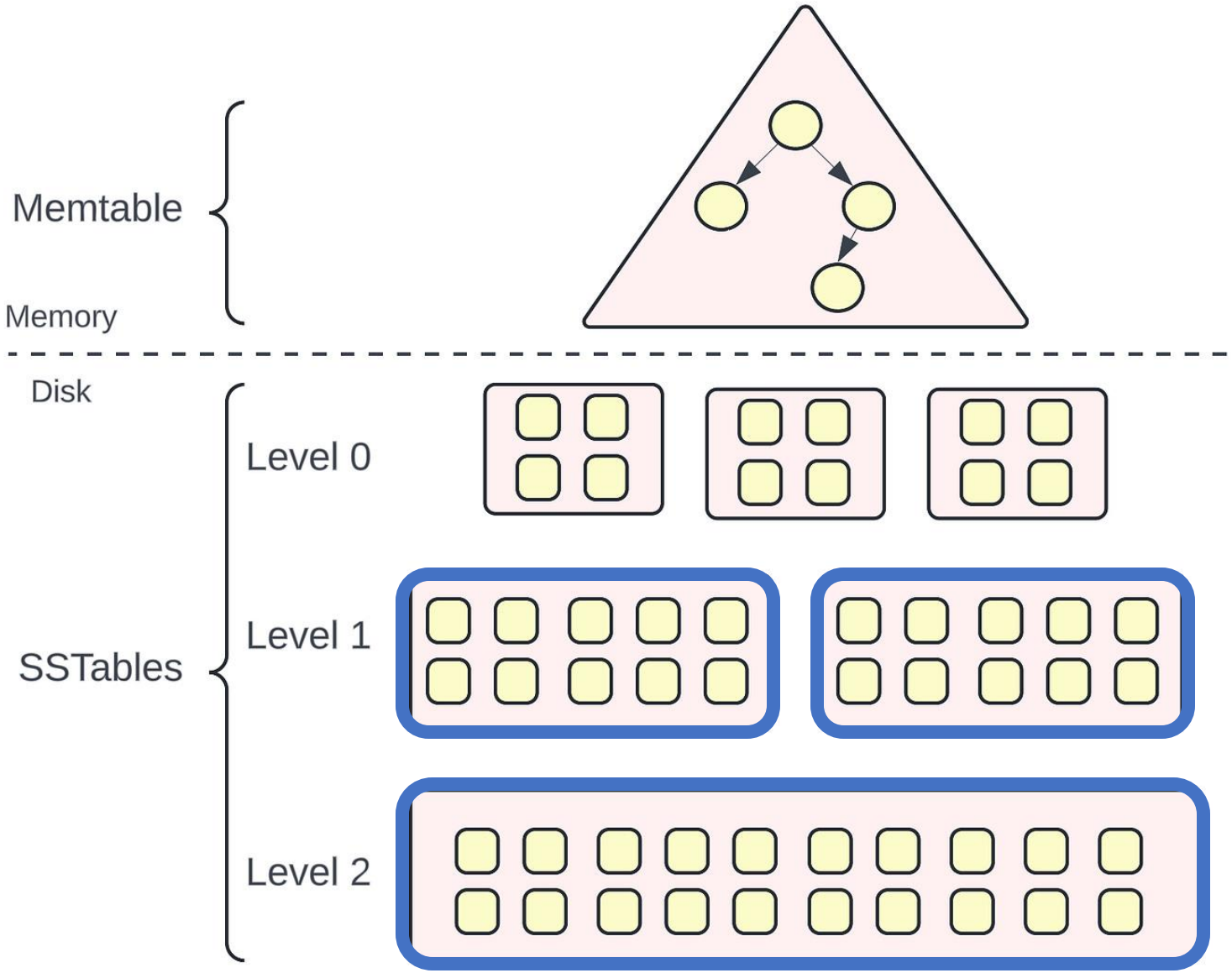


Image source: https://vivekbansal.substack.com/p/what-is-lsm-tree

LSM Tree: Inserts

Once too many L0 files exist, compaction merges them into L1

Similarly, when there are too many L1 files, compaction merges them into L2

Different compaction strategies:

- Size tiered compaction strategy (bigger files in deeper levels)
- Leveled compaction strategy (more files per level)

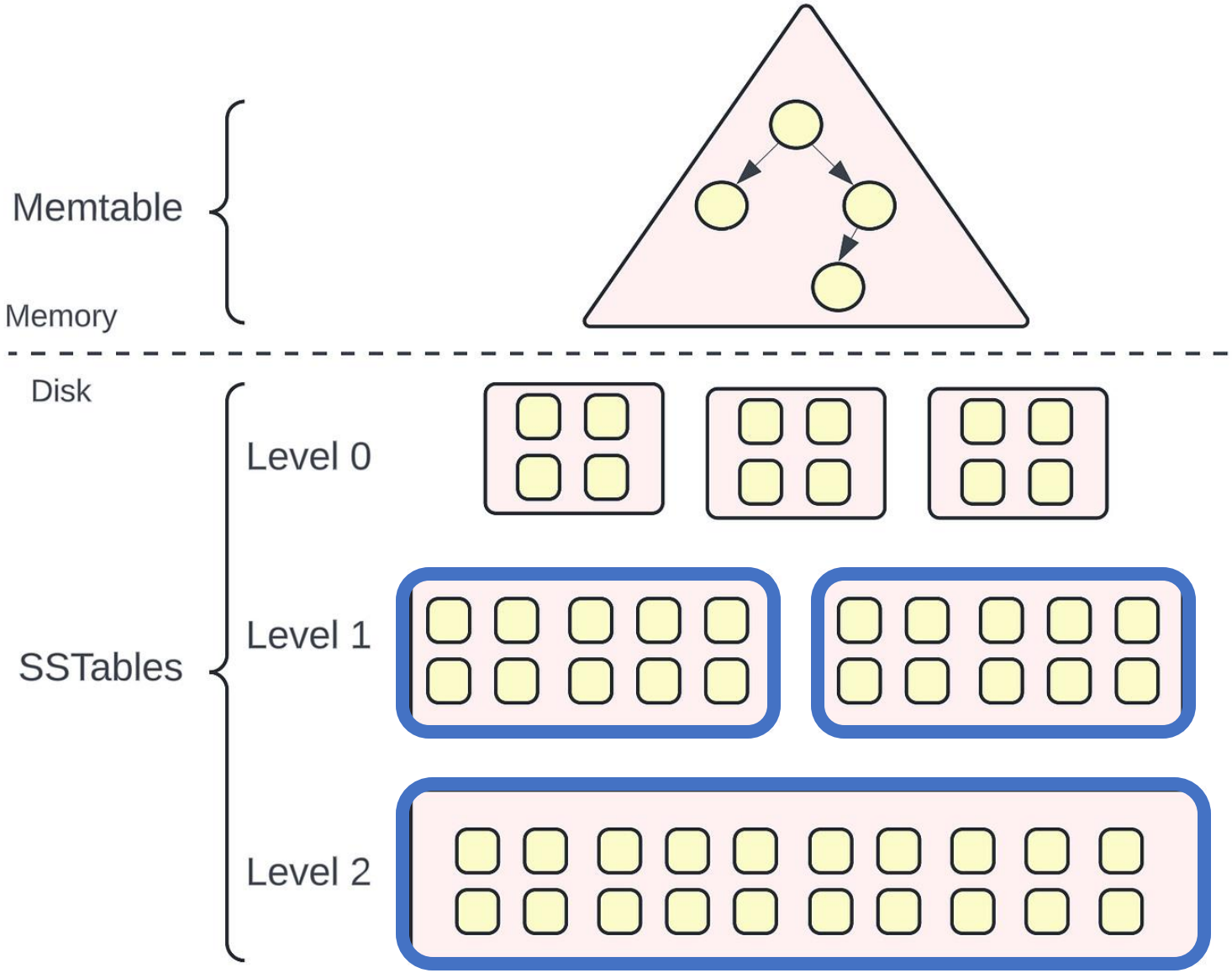


Image source: https://vivekbansal.substack.com/p/what-is-lsm-tree

LSM Tree: Lookups

Check Memtables

Check SSTables:

- Bloom Filter:
  - check existence of key
- Summary Tables:
  - e.g., store the range of keys in each SSTable
  - Skip irrelevant files

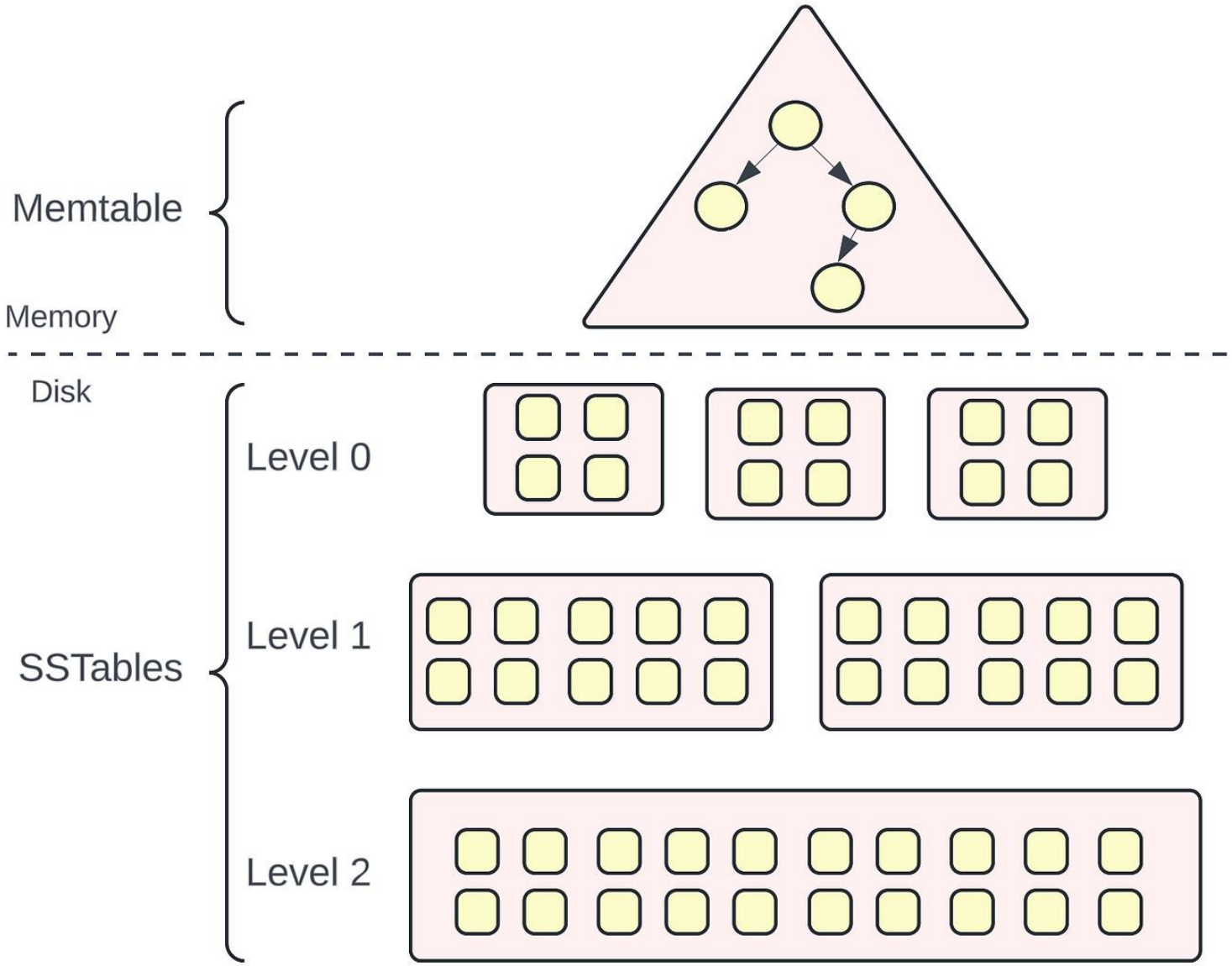


Image source: https://vivekbansal.substack.com/p/what-is-lsm-tree

Summary: B+ Tree vs LSM Tree

	B+ Tree	LSM Tree
Structure	Balanced tree with internal nodes and leaves	Multiple levels, memtable, and SSTables

Summary: B+ Tree vs LSM Tree

	B+ Tree	LSM Tree
Structure	Balanced tree with internal nodes and leaves	Multiple levels, memtable, and SSTables
Reads	Fast lookups and range queries	Potential for slower reads (multiple SSTables may need to be checked)

Summary: B+ Tree vs LSM Tree

	B+ Tree	LSM Tree
Structure	Balanced tree with internal nodes and leaves	Multiple levels, memtable, and SSTables
Reads	Fast lookups and range queries	Potential for slower reads (multiple SSTables may need to be checked)
Writes	Slower, as it can involve rebalancing, splitting and merging nodes	Faster, append-only for the initial writes

Summary: B+ Tree vs LSM Tree

	B+ Tree	LSM Tree
Structure	Balanced tree with internal nodes and leaves	Multiple levels, memtable, and SSTables
Reads	Fast lookups and range queries	Potential for slower reads (multiple SSTables may need to be checked)
Writes	Slower, as it can involve rebalancing, splitting and merging nodes	Faster, append-only for the initial writes
Typical Use Cases	Relational databases, read-heavy workloads	NoSQL databases, log-structured systems, write-heavy workloads

2. Course Summary

Course Summary

We learned…

1. How to query a database

1. Intro

2-3. SQL

4. ER Diagrams

5. Relational Algebra

6-7. DB Design

8-13. Index, Storage

14-17. QO

18-21. TXNs

22-25. Beyond RDBMS

Course Summary

We learned…

1. How to query a database
2. How to design a database

1. Intro

2-3. SQL

4. ER Diagrams

5. Relational Algebra

6-7. DB Design

8-13. Index, Storage

14-17. QO

18-21. TXNs

22-25. Beyond RDBMS

Course Summary

We learned…

1. How to query a database
2. How to design a database
3. How records are stored and indexed

1. Intro

2-3. SQL

4. ER Diagrams

5. Relational Algebra

6-7. DB Design

8-13. Index, Storage

14-17. QO

18-21. TXNs

22-25. Beyond RDBMS

Course Summary

We learned…

1. How to query a database
2. How to design a database
3. How records are stored and indexed
4. **How to optimize the performance of a database**

1. Intro

2-3. SQL

4. ER Diagrams

5. Relational Algebra

6-7. DB Design

8-13. Index, Storage

14-17. QO

18-21. TXNs

22-25. Beyond RDBMS

Course Summary

We learned…

1. How to query a database
2. How to design a database
3. How records is stored and indexed
4. How to optimize the performance of a database
5. How to handle concurrent user requests and crashes/aborts

1. Intro

2-3. SQL

4. ER Diagrams

5. Relational Algebra

6-7. DB Design

8-13. Index, Storage

14-17. QO

18-21. TXNs

22-25. Beyond RDBMS

Course Summary

We learned…

1. How to query a database
2. How to design a database
3. How records is stored and indexed
4. How to optimize the performance of a database
5. How to handle concurrent user requests and crashes/aborts
6. How RDBMS relates to OLAP, Distributed Query Processing, Parallel and Distributed DBMS, NoSQL etc.

1. Intro

2-3. SQL

4. ER Diagrams

5. Relational Algebra

6-7. DB Design

8-13. Index, Storage

14-17. QO

18-21. TXNs

22-25. Beyond RDBMS

Relational databases => Data-intensive systems

Most important computer applications must manage, update and query datasets

- Bank, store, search app…

Data quality, quantity & timeliness becoming even more important with AI

- Machine learning = algorithms that generalize from data

Relational databases => Data-intensive systems

Relational databases are the most popular type of data-intensive system (MySQL, Oracle, etc)

Many other systems facing similar concerns: key-value stores, streaming systems, ML frameworks, your custom app?

Reliability in the face of crashes, bugs, bad user input, etc
Concurrency: access by multiple users
Performance: throughput, latency, etc
Access interface from many, changing apps
Security and data privacy (not covered in this course)

Beyond this class

Classes:

- CS 4420/6422: Database System Implementation
- CS 4423/6423: Advanced Database System Implementation
- CS 6220: Big Data Systems and Analytics
- CS 8803-LRV: Large Scale & Real-Time Visual Analysis
- CS 8803-DML: Data-Centric Machine Learning

DB research at GT:

- Data Systems and Analytics Group