

LSM Tree and SS Tables Explained

What is an LSM Tree?

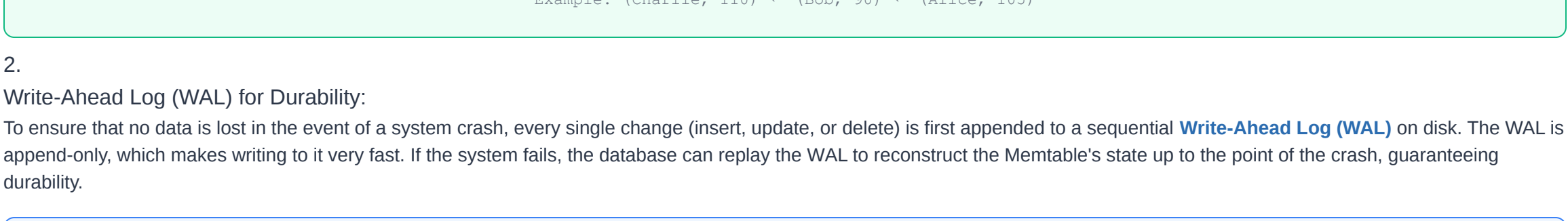
LSM Tree (Log-Structured Merge Tree) is a write-optimized data structure commonly used in modern databases like LevelDB, RocksDB, and Cassandra. It's designed to handle high write throughput efficiently and is structured as:

- An in-memory balanced binary search tree (like a Red-Black Tree or AVL Tree).
- On-disk immutable sorted files called SSTables.

🔧 How It Works – Step by Step

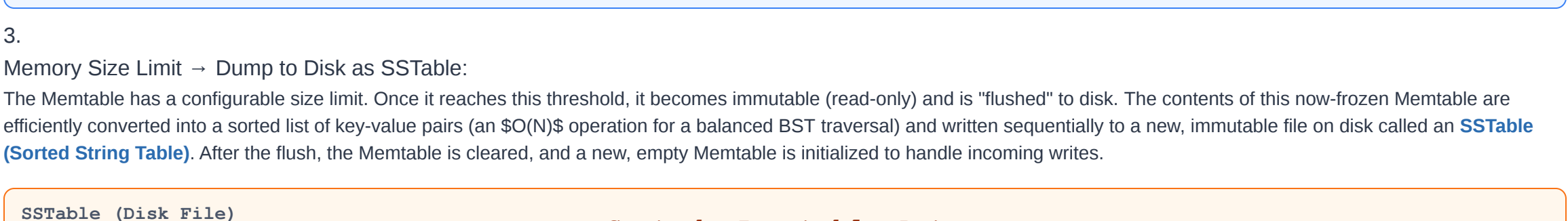
1.

Start in Memory (The Memtable):
Reads and writes first go to a memory-resident balanced tree, known as the **Memtable**. This structure allows for incredibly fast operations (inserts, updates, and point reads are all $O(\log n)$ because all data access occurs in RAM, avoiding costly disk I/O at this stage.



2.

Write-Ahead Log (WAL) for Durability:
To ensure that no data is lost in the event of a system crash, every single change (insert, update, or delete) is first appended to a sequential **Write-Ahead Log (WAL)** on disk. The WAL is append-only, which makes writing to it very fast. If the system fails, the database can replay the WAL to reconstruct the Memtable's state up to the point of the crash, guaranteeing durability.



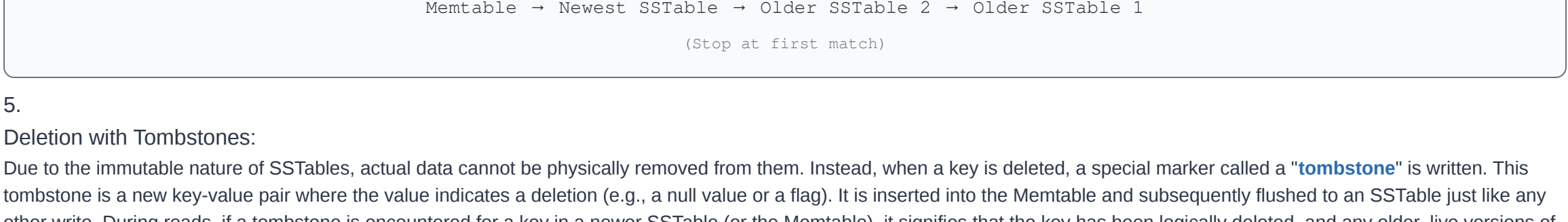
3.

Memory Size Limit → Dump to Disk as SSTable:
Due to the immutable nature of SSTables, actual data cannot be physically removed from them. Instead, when a key is deleted, a special marker called a **"tombstone"** is written. This tombstone is a new key-value pair where the value indicates a deletion (e.g., a null value or a flag). It is inserted into the Memtable and subsequently flushed to an SSTable just like any other write. During reads, if a tombstone is encountered for a key in a newer SSTable (or the Memtable), it signifies that the key has been logically deleted, and any older, live versions of that key in older SSTables are ignored.



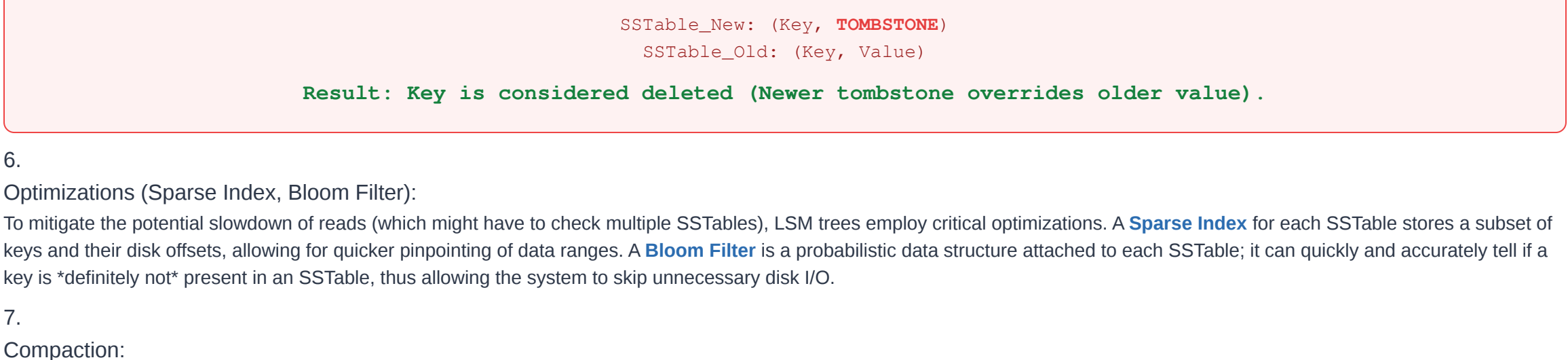
4.

Reading Data:
When a read request comes in for a specific key, the LSM tree follows a hierarchical search strategy to find the most recent version of that key. It first checks the active in-memory Memtable. If the key is not found there, it then proceeds to search the on-disk SSTables, always starting from the newest SSTable and moving backward chronologically to older ones. This is because newer SSTables contain more recent data, overriding older versions. As soon as the key is found, the search stops, ensuring the latest value is returned. Each SSTable can be binary searched efficiently due to its sorted nature.



5.

Deletion with Tombstones:
Due to the immutable nature of SSTables, actual data cannot be physically removed from them. Instead, when a key is deleted, a special marker called a **"tombstone"** is written. This tombstone is a new key-value pair where the value indicates a deletion (e.g., a null value or a flag). It is inserted into the Memtable and subsequently flushed to an SSTable just like any other write. During reads, if a tombstone is encountered for a key in a newer SSTable (or the Memtable), it signifies that the key has been logically deleted, and any older, live versions of that key in older SSTables are ignored.

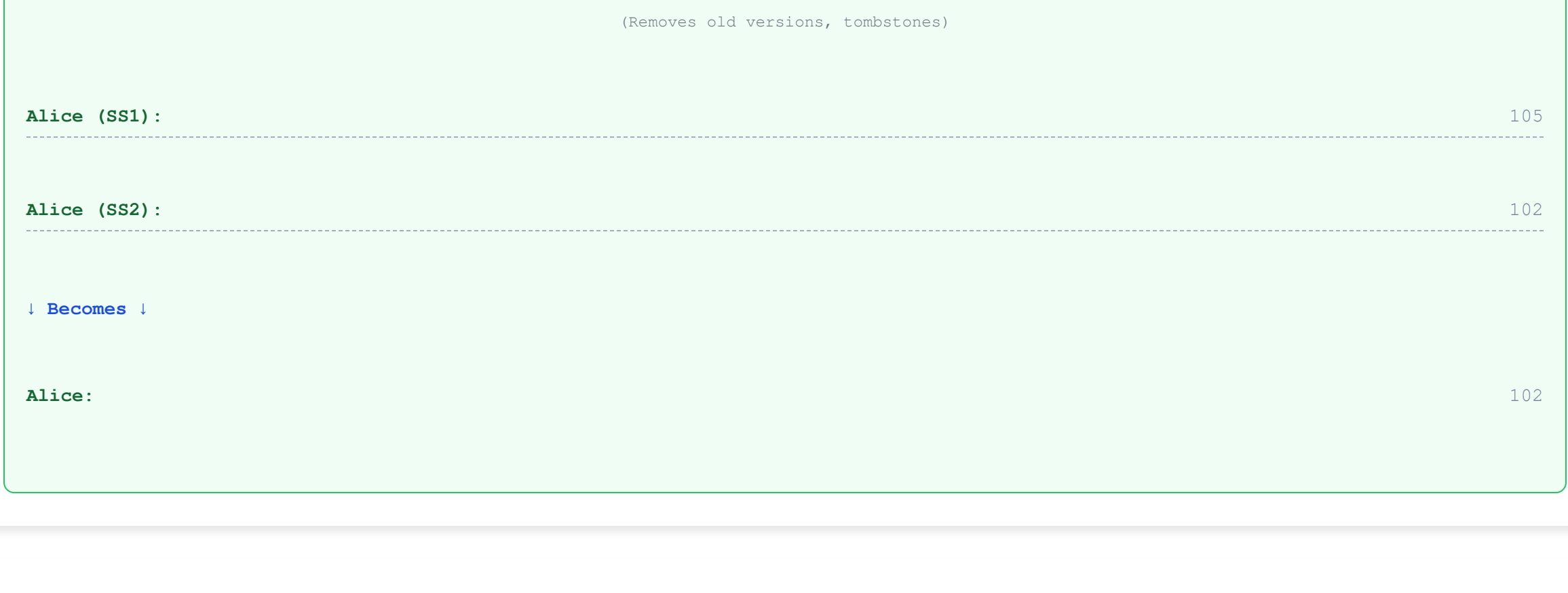


6.

Optimizations (Sparse Index, Bloom Filter):
To mitigate the potential slowdown of reads (which might have to check multiple SSTables), LSM trees employ critical optimizations. A **Sparse Index** for each SSTable stores a subset of keys and their disk offsets, allowing for quicker pinpointing of data ranges. A **Bloom Filter** is a probabilistic data structure attached to each SSTable; it can quickly and accurately tell if a key is "definitely not" present in an SSTable, thus allowing the system to skip unnecessary disk I/O.

7.

Compaction:
This is a vital background process that continuously merges multiple SSTables into new, consolidated ones. The primary goals of **compaction** are to reclaim disk space by removing duplicate versions of keys (keeping only the most recent) and physically purging keys that have been marked by tombstones. Compaction also helps maintain read performance by reducing the total number of SSTables that a read operation might need to check. This process happens asynchronously, like garbage collection, to minimize impact on foreground read/write operations.



1. The Memtable (In-Memory Component)

The **Memtable** is the heart of an LSM Tree. It's a small, in-memory data structure (often a balanced binary search tree like a Red-Black Tree or AVL Tree) that handles all incoming writes (inserts, updates, deletes).

Every write operation is first recorded in a **Write-Ahead Log (WAL)** on disk for durability, then applied to the Memtable. The WAL ensures that even if the system crashes, the data in the Memtable can be recovered by replaying the log.

