LSM(Log-Structured Merge Tree)

LSM Tree——分布式存储系统（BigTable）的理论模型

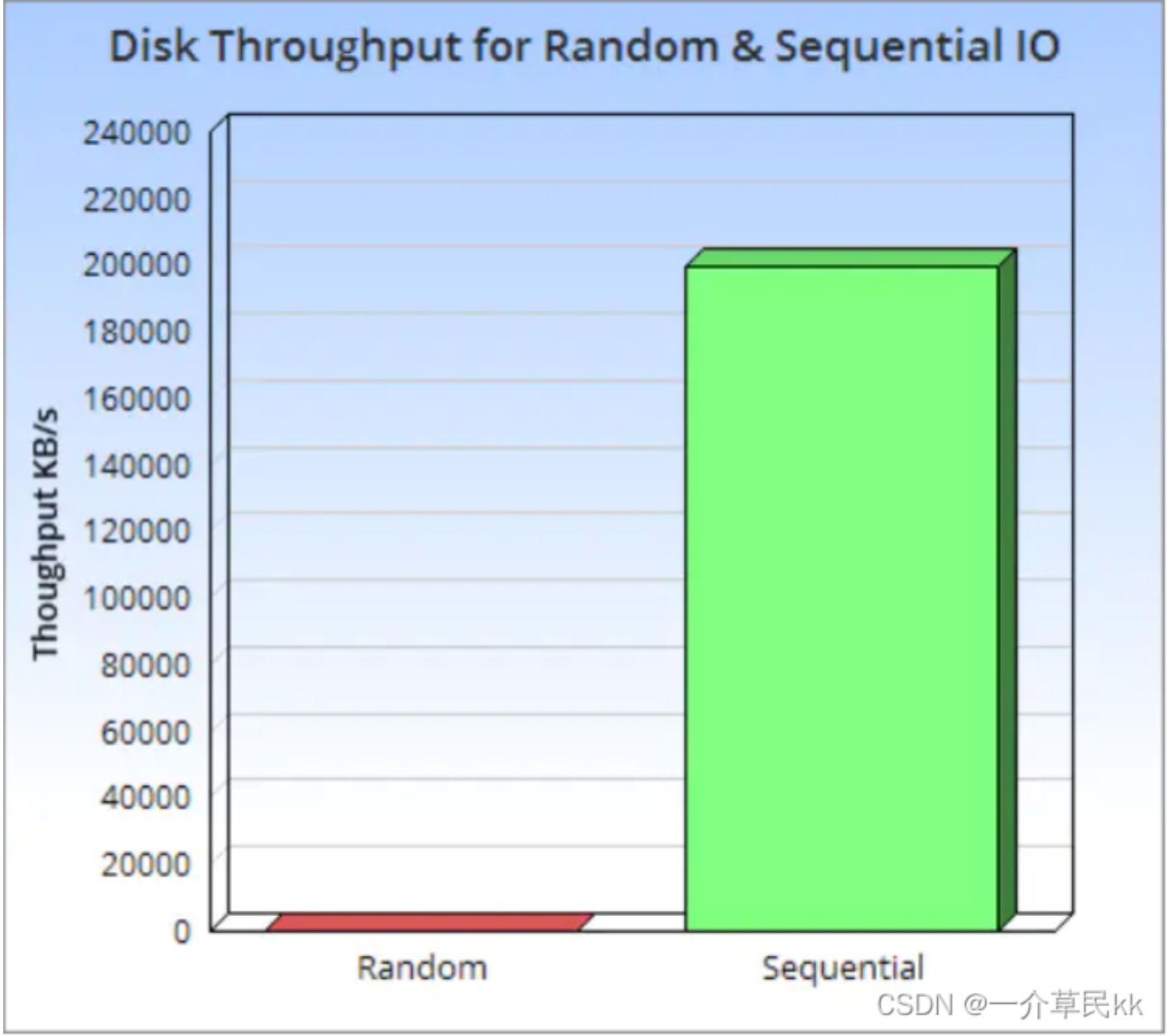
一、什么是LSM Tree
二、基本原理简述
2.1 SSTable和Level
2.2 分布式存储系统（BigTable）
2.2.1 数据模型
2.2.2 组件
三、LSM Tree框架图
四、总结
参考：

一、什么是LSM Tree

LSM Tree全称**日志结构合并树**（**Log-Structured Merge Tree**）。对于存储介质为*磁盘*或*固态盘*的数据库，长期以来主流使用B+树这种索引结构来实现快速数据查找。当数据量不太大时，B+树读写性能表现非常好。但是在海量数据情况下，B+树越来越高，由于B+树更新和删除数据时需要沿着B+树逐层进行 页分裂 和 页合并，严重影响数据写入性能。为了解决这种情况，google在论文《*Bigtable: A Distributed Storage System for Structured Data*》中介绍了一种新的数据组织结构LSM Tree（Log-Structured Merge Tree），LSM Tree是其中着重介绍的一种新的*分布式存储系统*的一个*理论模型*。LSM Tree的存储架构设计在*机械盘*时代大方异彩，同时也是一个把 顺序写 发挥到极致的设计架构，核心之一是 log文件 。当前比较流行的NoSQL数据库，如Cassandra、RocksDB、HBase、LevelDB等，newSQL数据库，如TIDB，都是使用LSM Tree来组织磁盘数据的。

二、基本原理简述

LSM Tree是一个 分层 、 有序 、 针对块存储设备 （*HDD*和*SSD*）特点而设计的**数据存储结构**。他的核心理论基础还是磁盘的顺序写速度比随机写速度快非常多，即使是SSD，由于块擦除和垃圾回收的影响，顺序写速度还是比随机写速度快很多。



2.1 SSTable和Level

LSM Tree将存储数据切分为一系列的SSTable（*Sorted String Table*），并将SSTable分为 `Level 0 - Level N`。每个SSTable内的数据都是 有序 的任意字节组，并且SSTable一旦写入磁盘中就像日志一样 无法修改 。对于LSM Tree若需要修改或者删除数据并不是直接对旧数据进行操作，而是将新数据写入新的SSTable中。若需删除数据则是写一个相应数据的删除标记的记录到一个新的SSTable中。依照这种方法也确保了LSM Tree写数据时对磁盘的操作都是顺序块写入操作，而没有随机写操作。

LSM Tree这种独特的写入方式，导致在查找数据的时候，LSM Tree 不能像B+ Tree那样在一个统一的索引表中进行查找，而是从最 新的SSTable 到 老的SSTable 中依次进行查找。如果在新的SSTable中找到了需要查找的 数据 或者相应的 删除标记，则直接返回查找结果；若没找到，再到老的SSTable中进行查找，直到老的SSTable查找完。为了提高查找效率，LSM Tree对SSTable进行分层、有序组织，就是将SSTable组织成 多层，同一层可以有多个SSTable，并且每个SSTable内的数据是有序的，前一个SSTable的最大数据值小于后一个SSTable的最小数据值（实际情况更加复杂），以达到提升在同一层SSTable的查询效率。同时，LSM Tree会将多个SSTable合并（Compact）为一个新的SSTable，这样可以减少SSTable的数量，同时把修改前的数据或删除的数据真正从SSTable中删除，减少SSTable的大小（这就是LSM中 `Merge` （合并）的由来），对提高查找性能起到了极其重要的作用。

2.2 分布式存储系统（BigTable）

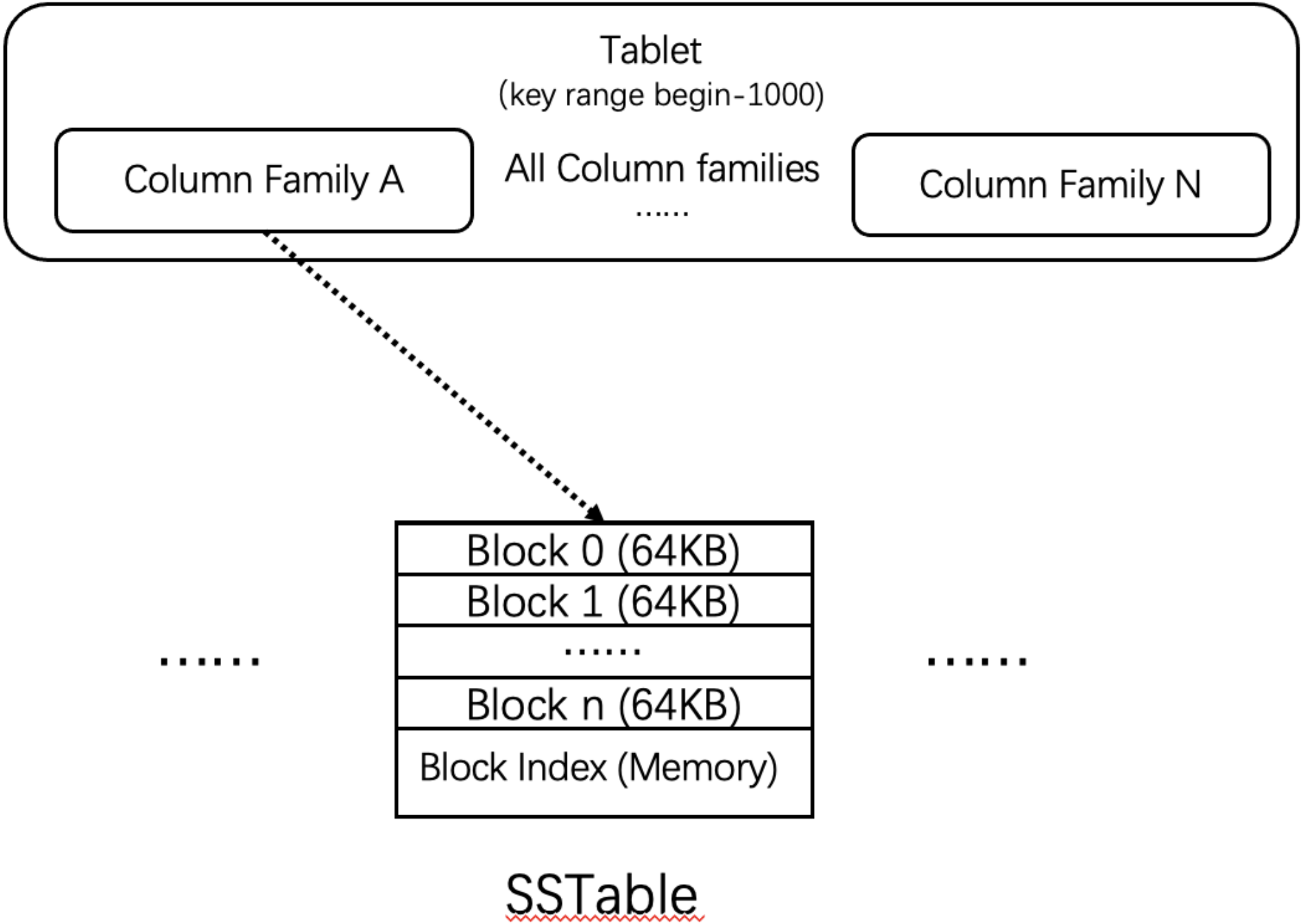
2.2.1 数据模型

BigTable本质上是一个为稀疏结构化数据设计的分布式多维排序Map。这里的“稀疏”指的是对于某一个结构化对象，有大量 属性 是 缺失 的，且多个对象之间缺失的 属性分布 也很 分散 。Map的Key由三个部分组成，分别是Row、Column Family:Column、Timestamp。

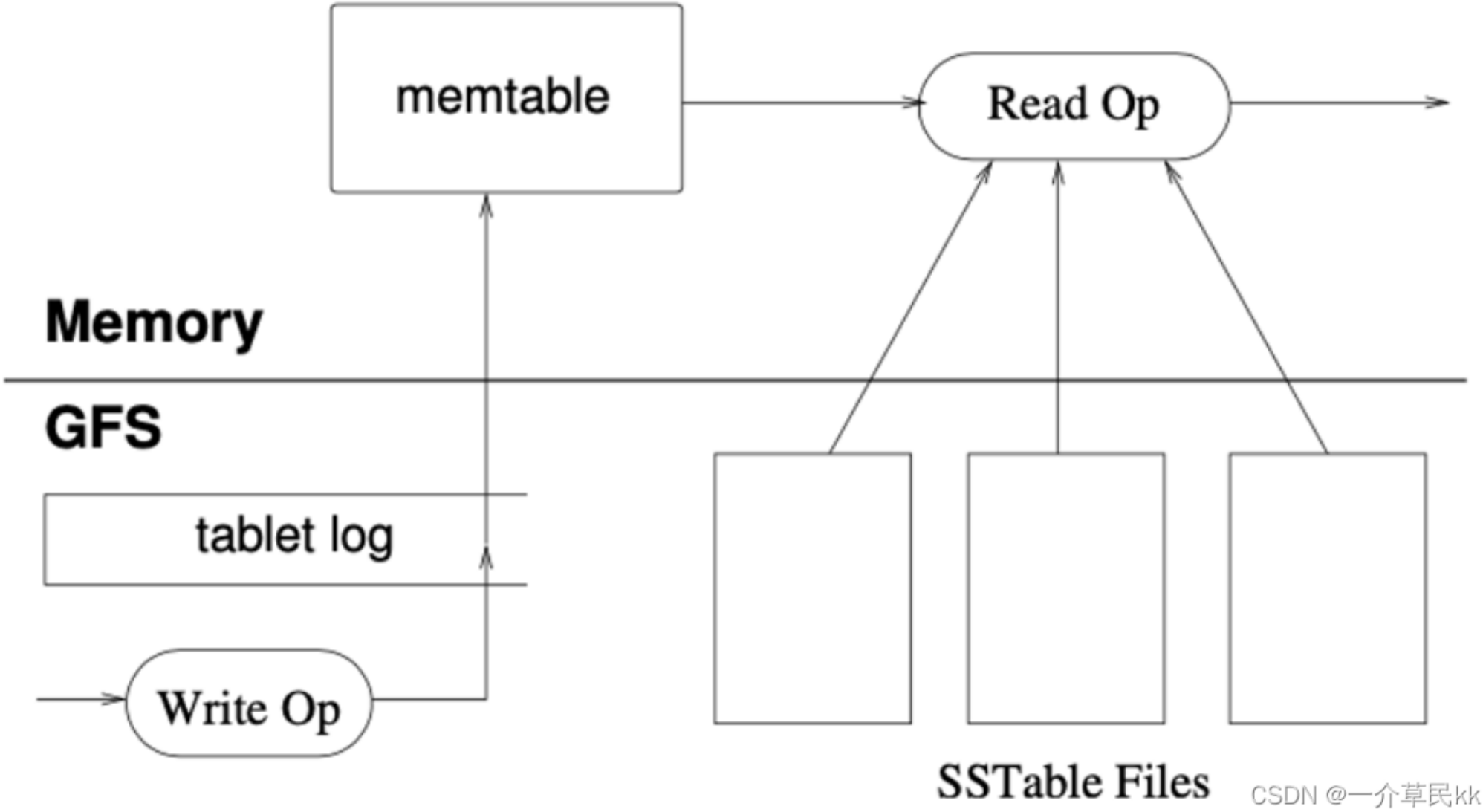
- **Row**（行）
  行是结构化对象的一个实例。在BigTable中行是根据Key用 字典序排序存储 的。BigTable支持单行级别的 原子更新操作，客户端无须在并行调用场景关系行操作的原子性。
- **Column Family**（列簇）：**Column**（列）
  列簇是多个列的集合。一般情况下，存储在同一个列簇中的数据具有同样的类型，并且会压缩在一起。列簇还是 访问权限控制 的 最小单元 。
- **Timestamp**（时间/版本）
  Timestamp是一个64-bit的时间版本号。BigTable可以存储同一份数据的 多个版本，并支持按照版本进行访问。Timestamp可以由客服端自行指定，或由BigTable实时生成。

2.2.2 组件

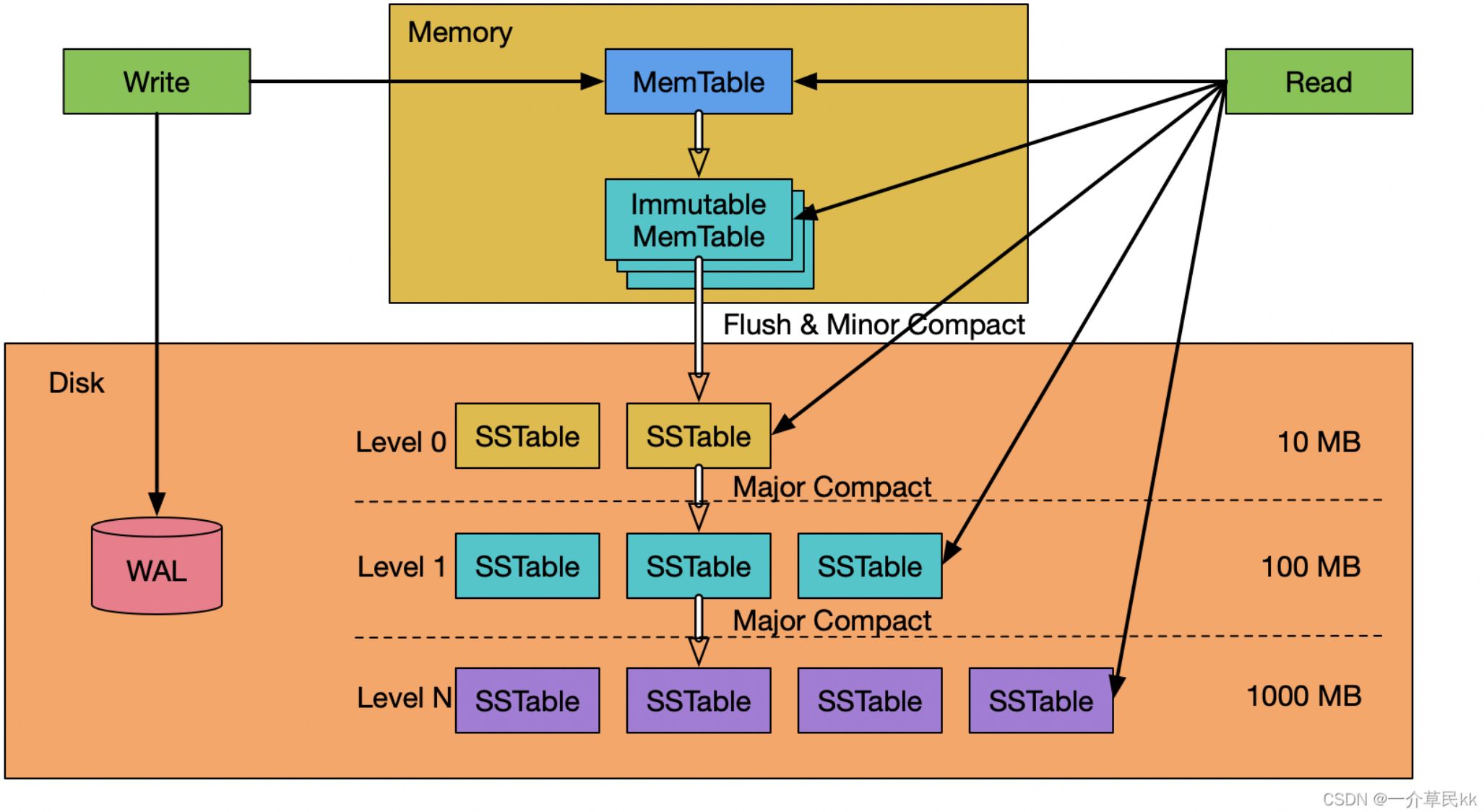
- Tablet
  Tablet是数据调度的 最小单位，由一个**Key Range**组成，包含了这个Key Range中的所有数据。每一行Key又包含了多个列簇，同一个列簇内的数据被**压缩**并存放在SSTable中。
  SSTable是Google内部的持久化KV存储结构，将数据按Key排序存储实现高效检索。SSTable被创建过后，不可对其内容进行更改，属于“**只读结构**”。
  SSTable内部由多个 `Block` 组成，每个Block默认大小为**64KB**，在SSTable的最后存放Block的索引信息。
  使用SSTable时，首先将Block索引信息加载到**内存**中；检索时，首先通过二分查找检索 `Block索引` 确定Block，再从磁盘读取该Block的数据。这样设计可以更好的利用磁盘的顺序读取特性，提高 查找性能 。同时SSTable也可以直接 映射 到**内存**中。



- MemTable
  前面介绍到的SSTable是不可变的，若需对BigTable实现增删改操作就要用到MemTable。
  MemTable是存放在内存中的**数据结构**，当发生`变更操作`时，首先对MemTable进行写入；同样，读取数据时，就同时读取SSTable和MemTable，将结果`合并`。随着数据持续写入，MemTable不断增长，被写满后，会重新创建一个**新的MemTable**，**老的MemTable**被`锁定`，然后Merge（合并）到新的SSTable中。
  由于MemTable存放在内存的缘故，当机器发生故障进程突然被`kill`时，MemTable中的数据会丢失。为了保证系统的稳定高可用，BigTable通过`磁盘Log`记录了所有的**Commit操作**，当发生数据丢失时，可以通过Log和SSTable的时间戳，及时**恢复**MemTable中的数据。



三、LSM Tree框架图



从图中可知，LSM Tree的数据主要由Memory（内存）和Disk（磁盘）组成。Memory主要由一个`MemTable`和一个或多个`Immutable MemTable`组成。Disk主要由分布在多级**Level**的`SSTable`组成。Level级数越**小**表示处于该Level的SSTable越**新**，Level级数越**大**表示处于该Level的SSTable越**老**，最大级数（Level N）大小由系统设定。在此处Disk中最小的级数是Level 0，也会有系统把Memory中的Immutable MemTable定义为Level 0，即Disk中的数据从Level 1开始，这只是Level定义不同，不影响系统的工作流程和对系统的理解。

WAL（Write Ahead LOG）严格来说不是LSM Tree结构的一部分，但是实际系统中，WAL是数据库不可或缺的一部分，WAL的结构和作用跟其他数据库一样，是一个只能在尾部以**Append Only**方式追加记录的`日志结构文件`，他用来当系统崩溃重启时重放操作，使MemTable和Immuntable MemTabel中未持久化到磁盘中的数据不会丢失，这是通过**LOG**保证存储在内存中数据不丢失的一个方法。

MemTable往往是一个`跳表`（Skip List）组织的有序数据结构（也可以是**有序数组**或者**红黑树**等`二叉搜索树`），跳表既支持高效的动态插入数据，对数据进行排序，也支持高效的对数据进行**精确查找**和**范围查找**。

SSTable一般由一组`数据block`和一组`元数据block`组成。元数据block存储了SSTable数据block的**描述信息**，如**索引**、**BloomFilter**（布隆过滤器）、**压缩**、**统计**等信息。因为SSTable是不可更改的，且是有序的，索引往往采用`二分数组结构`就行了。为了节省存储空间以及提高数据块block的读写效率，可以对数据block进行压缩。

四、总结

- LSM Tree以**牺牲**部分`读取性能`为代价**提高**`写入性能`，通常适用于类似时序数据库这类写多读少的场景。
- LSM Tree主要利用磁盘`顺序访问`要比`随机访问`快很多的思想实现**提高写入性能**的特点。
- LSM Tree是一种`分层的`、`有序的`、`基于磁盘`的**数据结构**。设计思想是先将数据的操作存到**内存**中，并由**LOG记录**，当内存的`MemTable`**达到阈值**时就通过`归并排序`方式**合并**放到`磁盘队尾`。
- LSM Tree**提升读性能**的优化策略主要是使用`布隆过滤器`、`多路归并机制`等。

参考：

经典论文研读：《Bigtable: A Distributed Storage System for Structured Data》
LSM Tree原理详解
LSM-Tree介绍