

Project 2 LSM-KV: 基于 LSM 树的键值存储系统

Basic 解释器的项目情况

- 增加了图形化
- 增加了一个语法树的输出 (Pretty Print)

- 造成了大量的边界情况
 - 大量“需求补丁”
- 为什么要模块化?
- 为什么要面向对象?
- 为什么要代码风格?
- 为什么要写注释?

- 原本考虑 LSM-KV 图形化

➢ 做两个弱弱的图形化不如把其中一个做得更好

BASIC 解释器 Project 问题汇总

【问题】清除某一行代码的命令

【回答】文档里面有说明。

【问题】LOAD 指令

【回答】LOAD 指令应触发 Qt 中的文件选择框，让用户选择一个文件内容为纯文本。在选择了文件之后，会将文件中的每一行依次在“命令输入框”中输入并执行 BASIC 程序。

【问题】界面中的三个按钮

【回答】界面中的三个按钮相当于其对应的三个指令的“快捷方式”，相当于在最下方“命令输入框”中输入对应的命令并按回车。

【问题】是否可以直接在代码框中编辑

【回答】不能，整个界面只有“命令输入框”是可以编辑的，其他的均是 read-only，只展示内容。

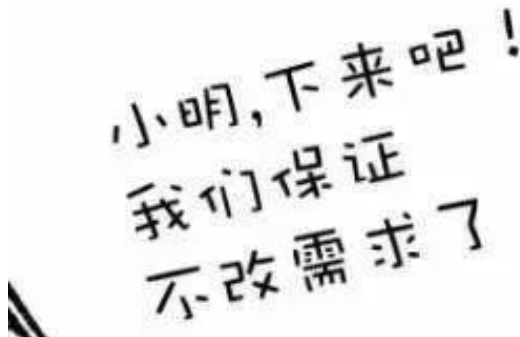
【问题】是否要求文件保存

【回答】项目中只要求 LOAD 指令，从文件中读取命令；不要求保存代码框中的代码。如果有其他指令，可以自己实现，命令名称



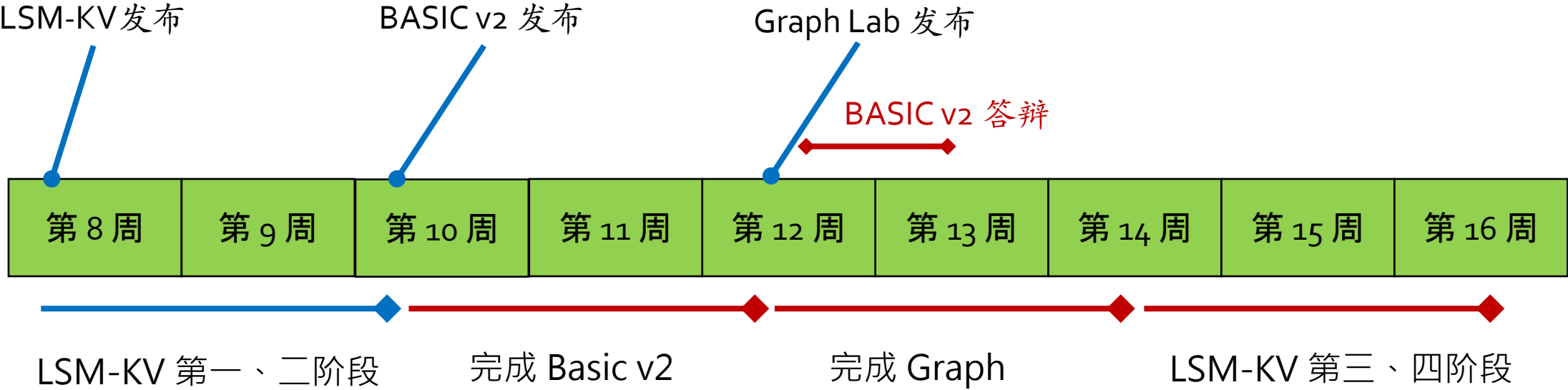
总是让你始料未及
和我们

Basic v2

- 明确边界问题
 - 提出一些新的功能/修改一些对现有功能的需求
 - 模拟“甲方爸爸”对需求的变更
 - 安排答辩
 - 分数:
 - v1 放松要求占 40%
 - v2 占 60%
- 



推荐的完成时间表



LSM Tree 键值存储系统

- 基于 Log-Structured Merge Tree (LSM)
- LSM Tree 是 Google 开源项目 LevelDB 和 Facebook 开源项目 RocksDB 的核心数据结构
- 是近年来存储学术会议中的热门话题



LSM Tree 键值存储系统

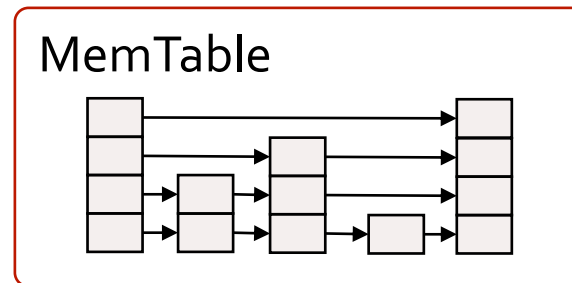
基本操作

- PUT(K,V): 设置键K的值为V
- GET(K): 获取键K的值
- SCAN(K1, K2): 获取键在K1~K2之间的所有键值对（使用迭代器）
- DELETE(K): 删除键K和其值

LSM Tree 基本结构

内存存储 MemTable

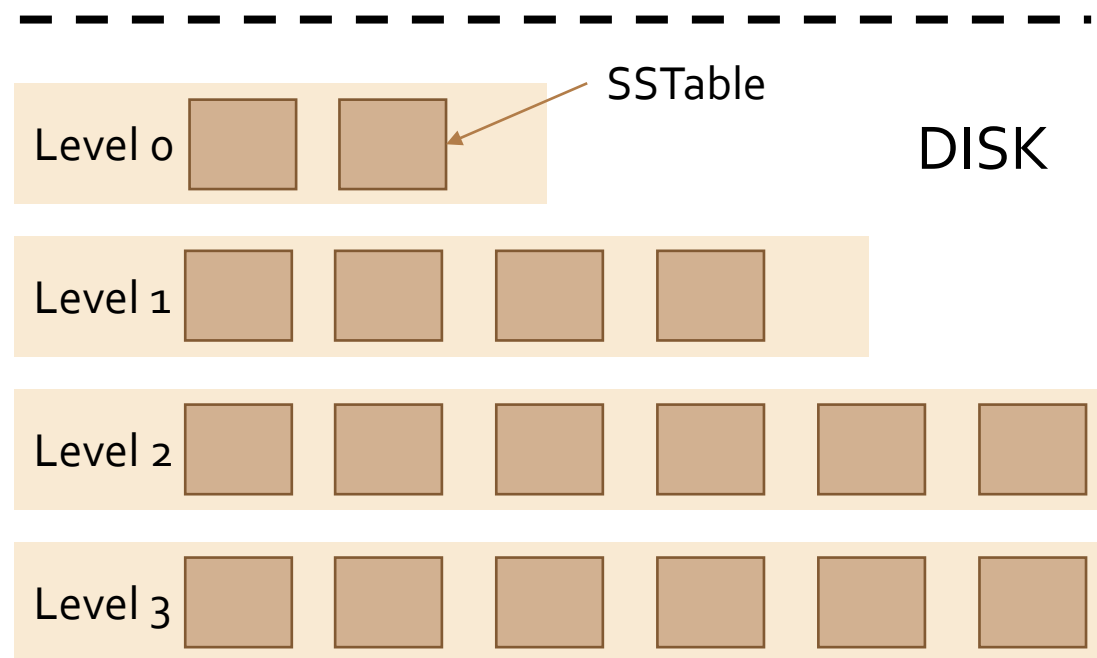
- 常用的是跳表 (skip-list)
- 新写入的数据均被保存在 MemTable 中



DRAM

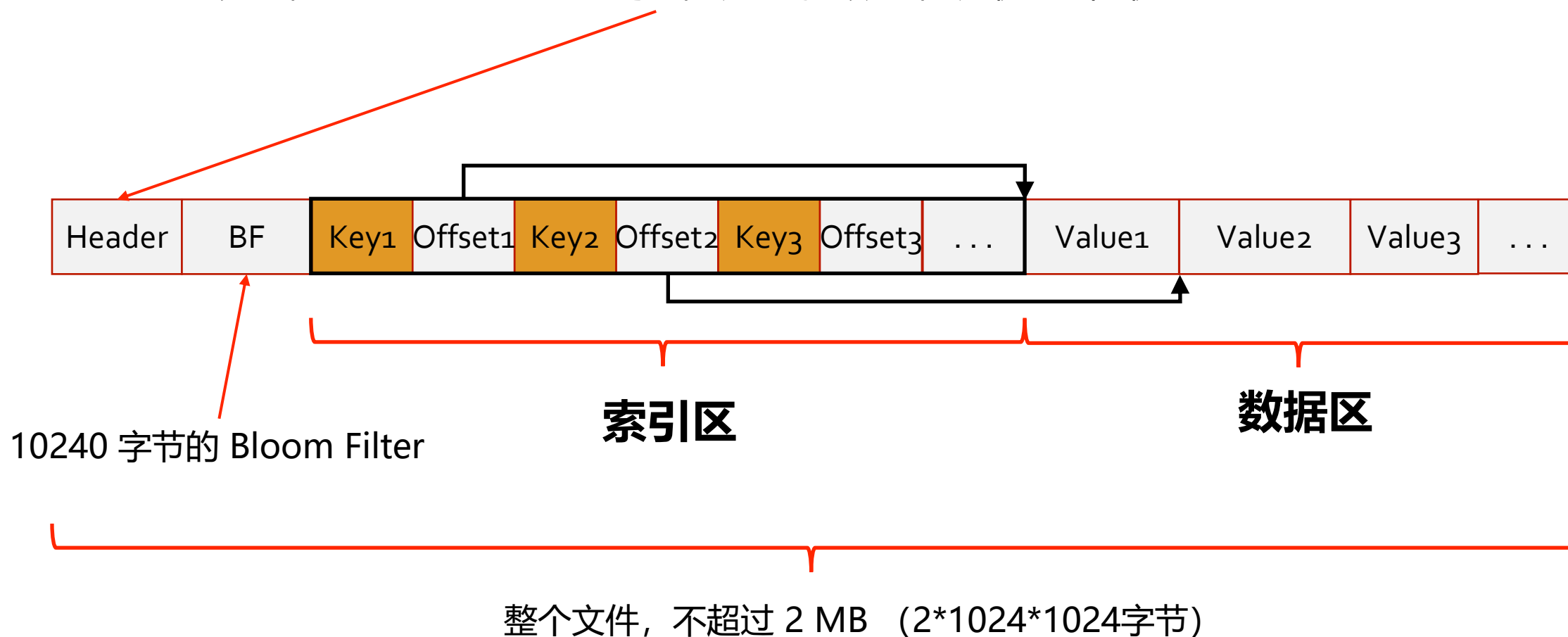
磁盘存储 SSTable

- 分层保存持久化数据
- 每层有多个固定大小的**只读**文件 (SSTable)
- 每个文件中保存的 key 是**有序**的
- 越下层文件数量越多, 比例是预设的
- 除第 0 层外, 同一层中文件保存的 key 区间不相交



SSTable 存储格式

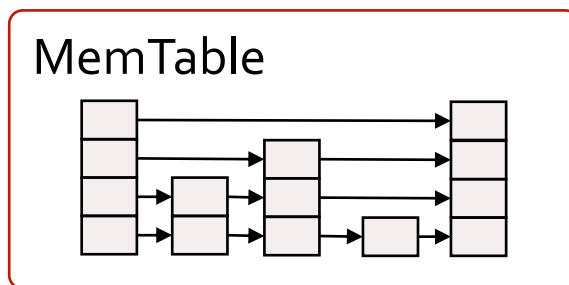
32字节的头部，包括：SSTable的时间戳，键值对的数量，键最小值和最大值



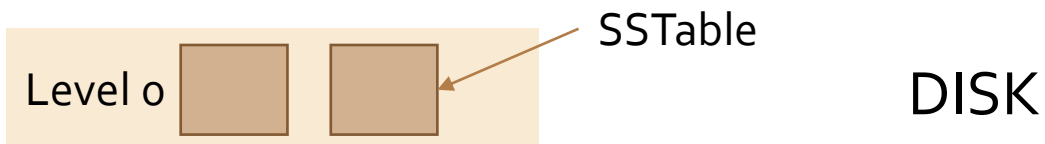
PUT(K, V)

1. 在 MemTable 中插入和覆盖
2. 数据量达到阈值, 触发 Compaction

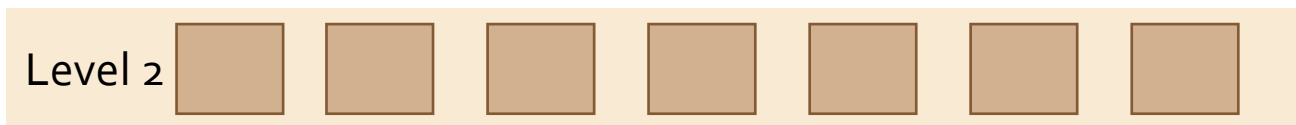
PUT(K, V)



DRAM



DISK



Compaction



Compaction

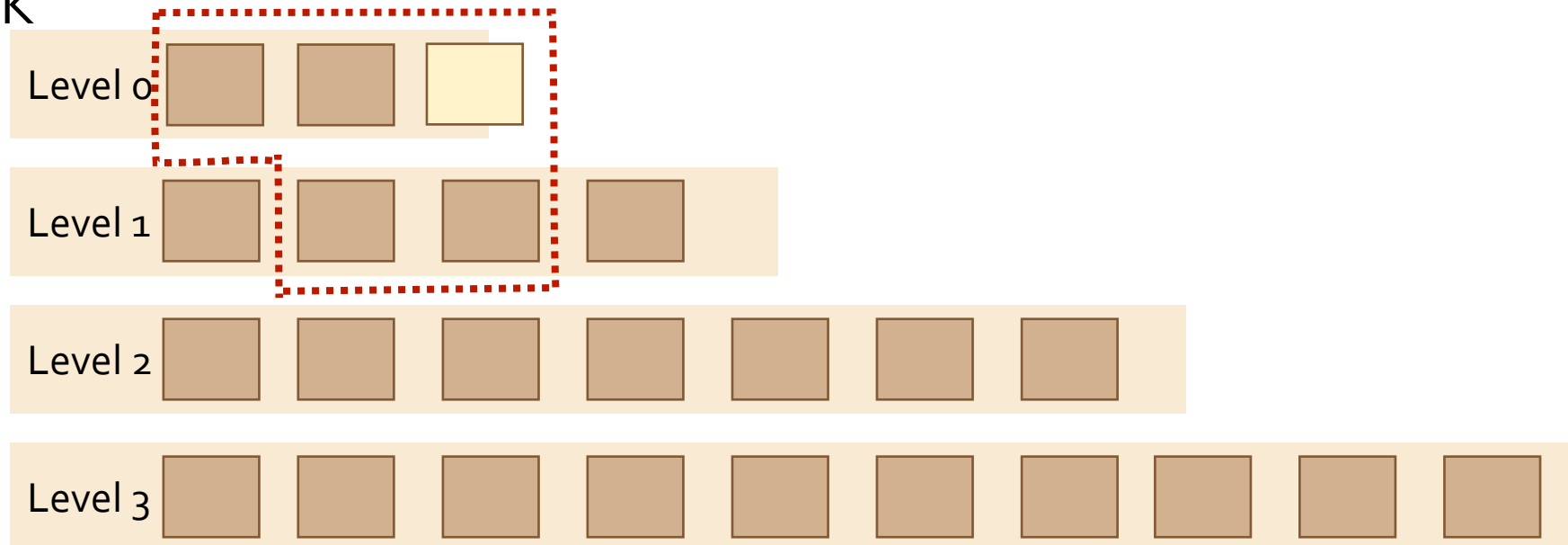
1. 先统计 Level 0 层中所有 STable 所覆盖的键的区间
然后在 Level 1 层中找到与此区间有交集的所有 STable 文件。

memtable

2. 将涉及到的文件读到内存，进行归并排序，
并生成SSTable文件写回下一层（Level 1）

DRAM

DISK



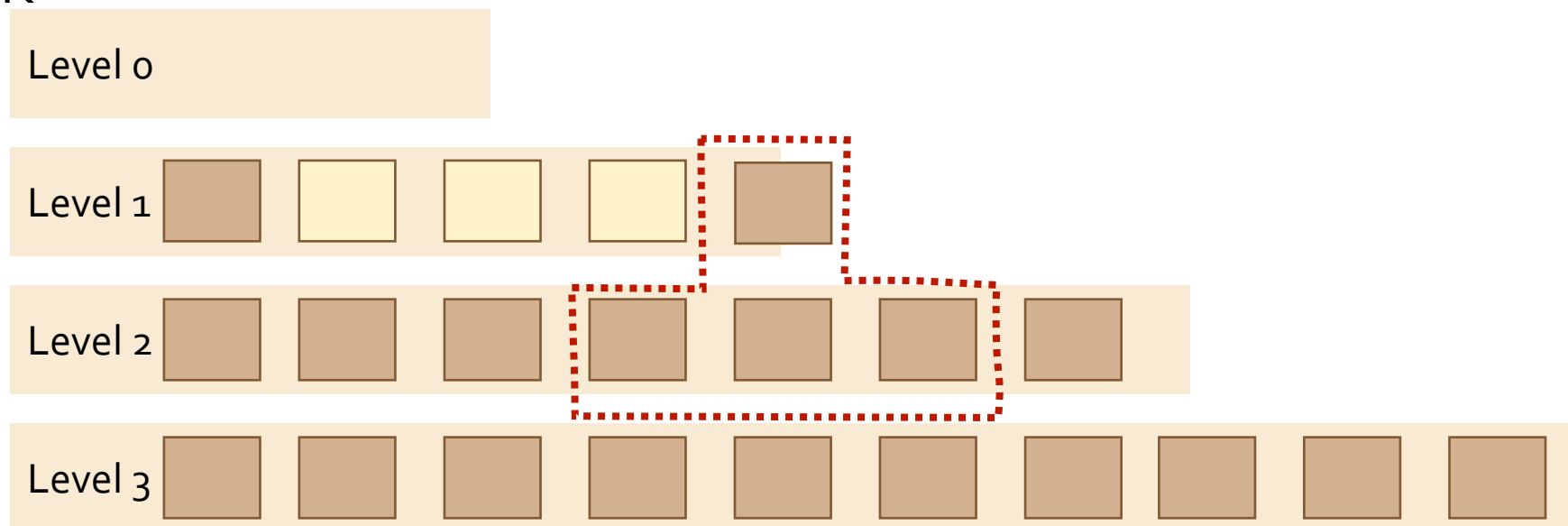
Compaction

memtable

2. 若下一层依然超出阈值，选取文件，继续向更下一层做compaction

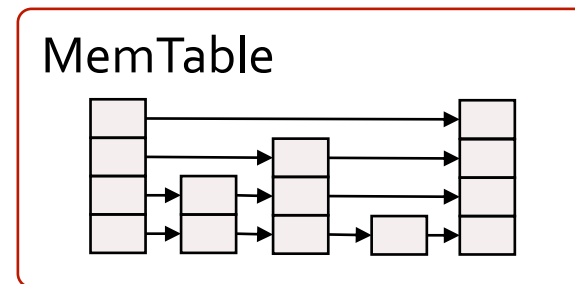
DRAM

DISK

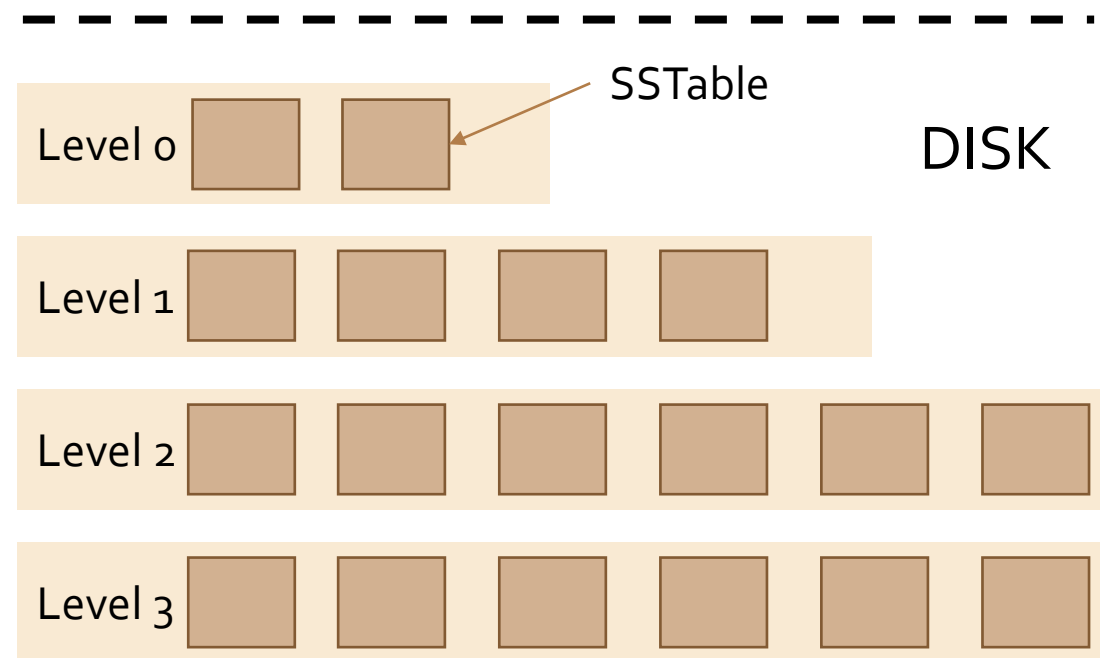


DELETE(K)

1. 删除 MemTable 中的对应键值对
2. 插入一个特殊的键值，标记 Key 被删除
该标记同样会被 Compaction，
直到最后一层才可以被删除

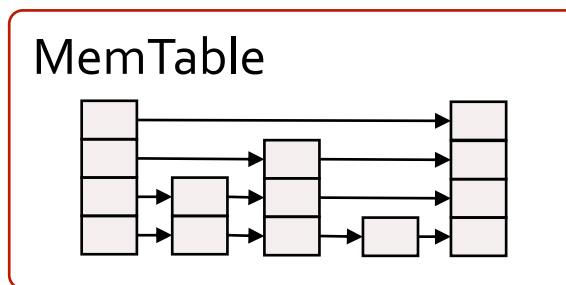


DRAM

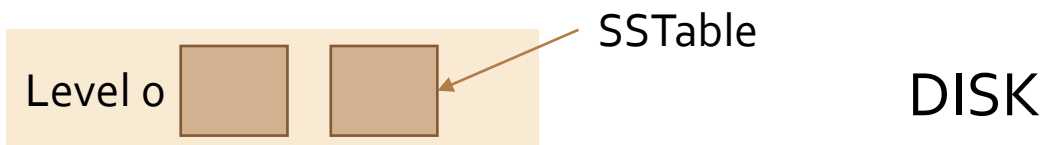


GET(K)

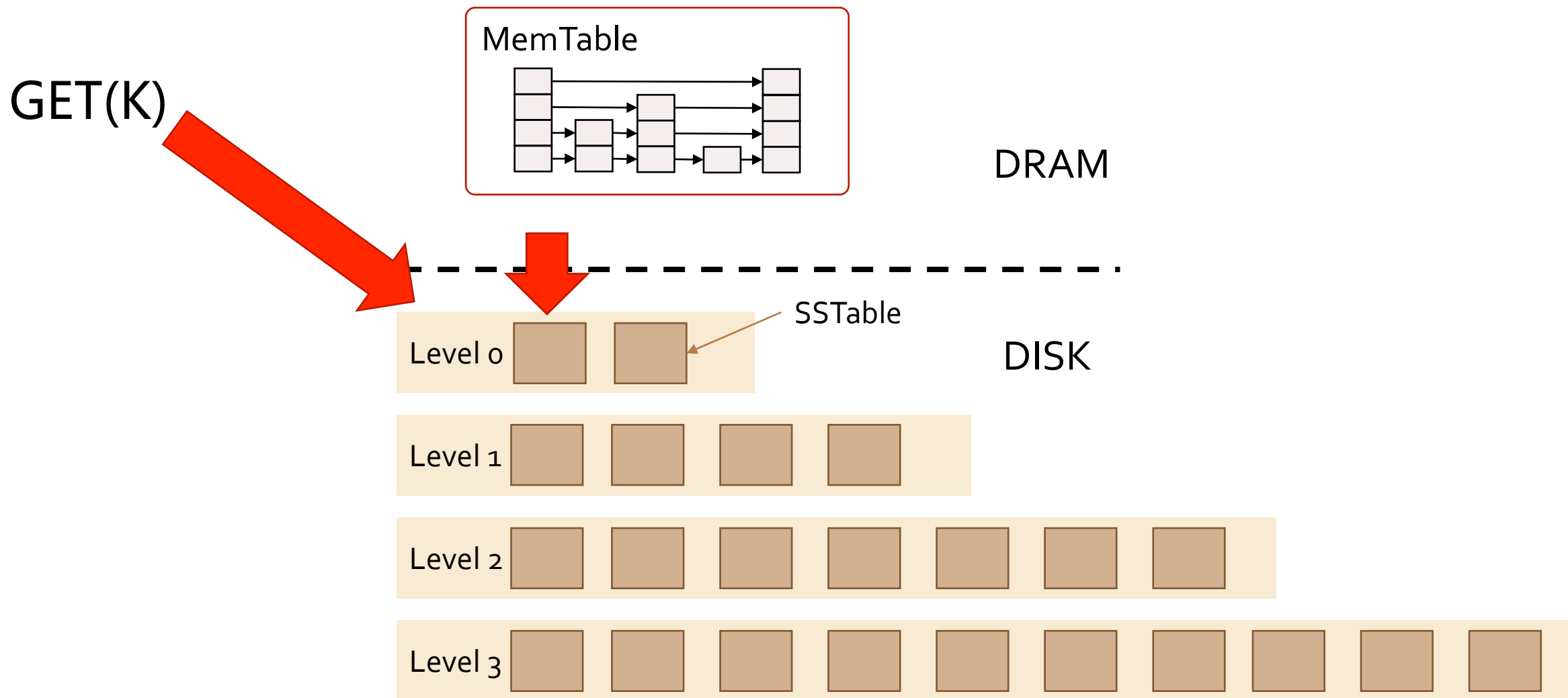
GET(K)



DRAM

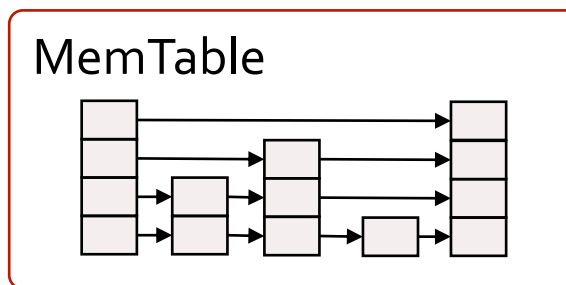


GET(K)

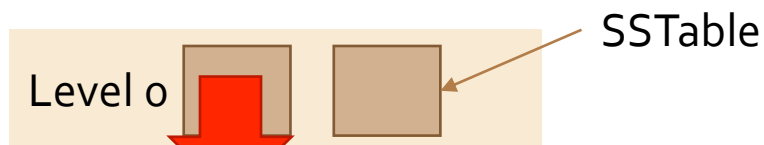


GET(K)

GET(K)



DRAM



DISK

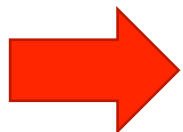


从磁盘中读取文件很慢!

CACHE!

优化：将索引缓存在DRAM中

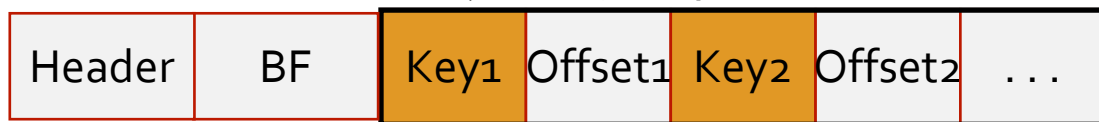
GET(K)



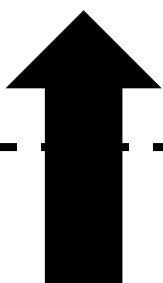
1. 检查 Key 的范围

2. 检查 BF

3. 二分查找索引



DRAM



DISK

32字节的头部



10240 字节的 Bloom Filter

索引区

数据区

启动与 Reset 操作

- LSM-KV 在启动时，会检查是否有此前保存下来的 SSTable 文件
- 并读取其中的内容
- 在 reset() 操作被调用时，会清空所有数据
- 在正常退出时，需将 MemTable 中的内容写成 SSTable

代码

- 给出了接口和部分测试代码（正确性测试，和持久化测试）
 - 请查看 README.md 文件，结合PDF文档
 - 请**不要使用绝对路径**，比如“ C:/project1/” , “ /home/student/proj1”
 - 不要使用 windows/linux 特定的函数和方法
 - 持久化测试可能需要 10G+ 硬盘大小，只要你确定自己没“耍花招”，可以修改其代码，将测试量变小。（我们测试时会使用新的测试）
- 性能测试和瓶颈分析
 - 熟悉软件测试方法和性能瓶颈分析
 - 测试目的：
 - 延迟：表现系统性能
 - 吞吐量：表现系统性能，并表现出 compaction 对性能的影响
 - 根据提供的 LaTeX 模板撰写报告

其他资料

- 请使用 c++14 标准
- 跨平台的文件操作相关函数已经在 utils.h 中给出
 - 主要用于创建/删除目录/删除文件/扫描文件等
- 二进制文件的访问
 - ostream & write(char* buffer, int count);
 - istream & read(char* buffer, int count);
 - 可参考 <http://c.biancheng.net/view/302.html>
https://zh.cppreference.com/w/cpp/io/basic_ifstream
https://zh.cppreference.com/w/cpp/io/basic_ofstream

LSM Tree 键值存储系统

可选的优化方向和方法

■ 方向

- 提升合并速度
- 提升读写性能
- 减少写放大
- 提升可靠性

■ 方法

- 增加 write-ahead-log: 保证写到MemTable的数据不会丢失
- ...

