上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

IPADS
INSTITUTE OF PARALLEL AND DISTRIBUTED SYSTEMS

Project LSM-KV: 基于 LSM 树的键值存储系统

董明凯

上海交通大学 并行与分布式系统研究所

https://ipads.se.sjtu.edu.cn

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

Database	Data layout	Compaction Trigger					Compaction Granularity				Data Movement Policy							
		Level saturation	#Sorted runs	File staleness	Space amp.	Tombstone-TTL	Level	Sorted run	File (single)	File (multiple)	Round-robin	Least overlap (+1)	Least overlap (+2)	Coldest file	Oldest file	Tombstone density	Expired TS-TTL	N/A (entire level)
RocksDB [30], Monkey [22]	Leveling / 1-Leveling	✓		✓					✓	✓		✓		✓	✓	✓		
	Tiering		✓		✓	✓		✓										✓
LevelDB [32], Monkey (J.) [21]	Leveling	✓							✓		✓	✓	✓					
SlimDB [47]	Tiering	✓							✓	✓								✓
Dostoevsky [23]	*L*-leveling	✓^{L}	✓^{T}				✓^{L}	✓^{T}				✓^{L}						✓^{T}
LSM-Bush [24]	Hybrid leveling	✓^{L}	✓^{T}				✓^{L}	✓^{T}				✓^{L}						✓^{T}
Lethe [51]	Leveling	✓				✓			✓	✓		✓					✓	
Silk [11], Silk+ [12]	Leveling	✓							✓	✓	✓							
HyperLevelDB [35]	Leveling	✓							✓		✓	✓	✓					
PebblesDB [46]	Hybrid leveling	✓							✓	✓								✓
Cassandra [8]	Tiering		✓	✓		✓		✓										✓
	Leveling	✓				✓			✓	✓		✓				✓	✓	
WiredTiger [62]	Leveling	✓					✓											✓
X-Engine [34], Leaper [63]	Hybrid leveling	✓							✓	✓		✓			✓			
HBase [7]	Tiering		✓					✓										✓
AsterixDB [3]	Leveling	✓					✓											✓
	Tiering		✓					✓										✓
Tarantool [57]	*L*-leveling	✓^{L}	✓^{T}				✓^{L}	✓^{T}										✓
ScyllaDB [55]	Tiering		✓	✓		✓		✓										✓
	Leveling	✓				✓			✓	✓		✓				✓	✓	
bLSM [56], cLSM [31]	Leveling	✓							✓		✓							
Accumulo [6]	Tiering	✓	✓			✓		✓										✓
LSbM-tree [58, 59]	Leveling	✓					✓											✓
SifrDB [44]	Tiering	✓								✓								✓

https://upload.wikimedia.org/wikipedia/commons/thumb/d/da/Rocksdb-icon.svg/220px-Rocksdb-icon.svg.png
https://dbdb.io/media/logos/leveldb-horizontal.png.280x250_q85.png
Constructing and Analyzing the LSM Compaction Design Space, VLDB

键值存储的基本操作

- 插入 insert/set/put (key, value)
  - 设置键 key 的值为 value
- 删除 delete/remove/del (key)
  - 删除键 key 和其值
- 查询 search/query/lookup/find/get (key) -> value
  - 获取键 key 的值
- 区间查询 scan(key1, key2)【不用做】
  - 获取键在 key1 ~ key2 之间的所有键值对

LSM Tree 基本结构【复合的数据结构：内存+磁盘】

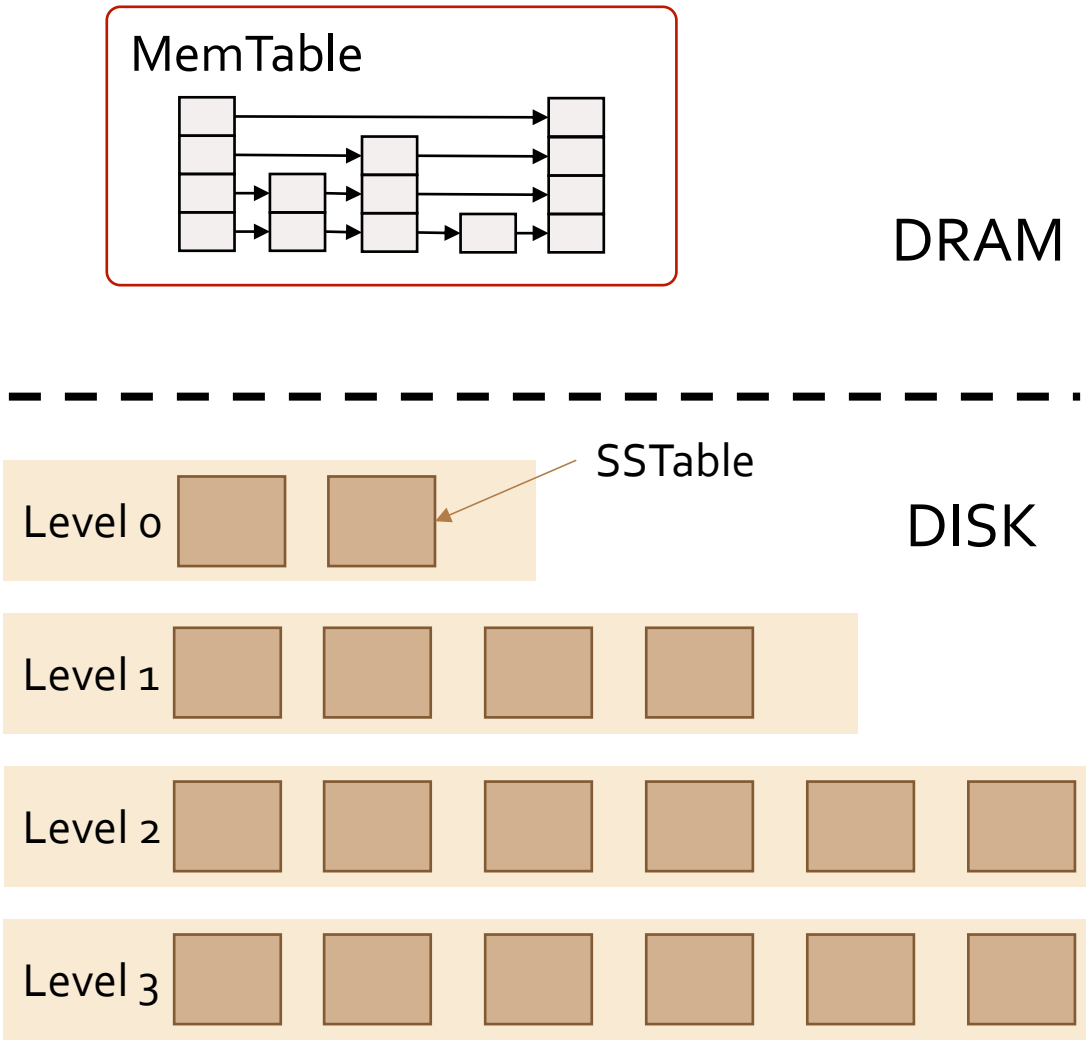
内存存储 MemTable

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

磁盘存储 SSTable

【sorted strings table】

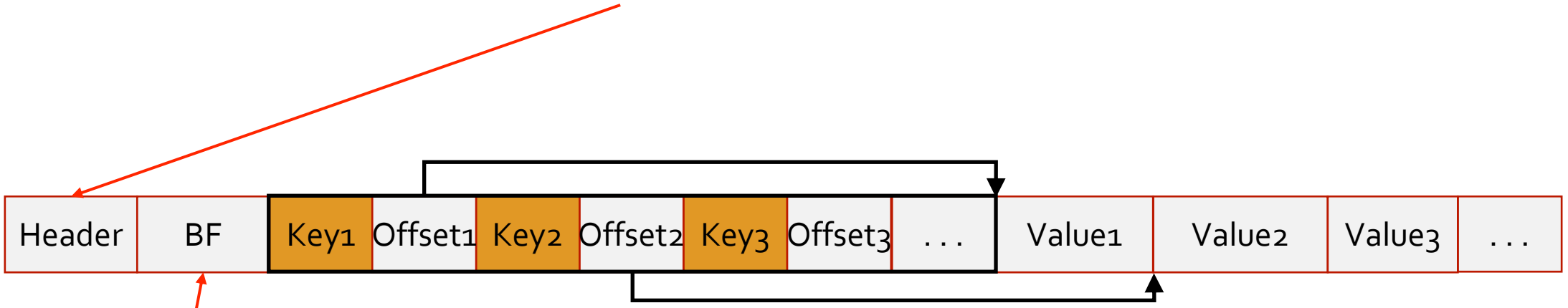
- 分层保存持久化数据
- 每层有多个固定大小的**只读**文件（SSTable）
- 每个文件中保存的 key 是**有序**的
- 越下层文件数量越多，比例是预设的
- Leveling层中文件保存的 key 区间不相交
- Tiering层中文件保存的 key 区间可相交
- Level0为Tiering



SSTable: Sorted Strings Table

SSTable 存储格式【里面存的东西是有序的】

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



10240 字节的 Bloom Filter

索引区

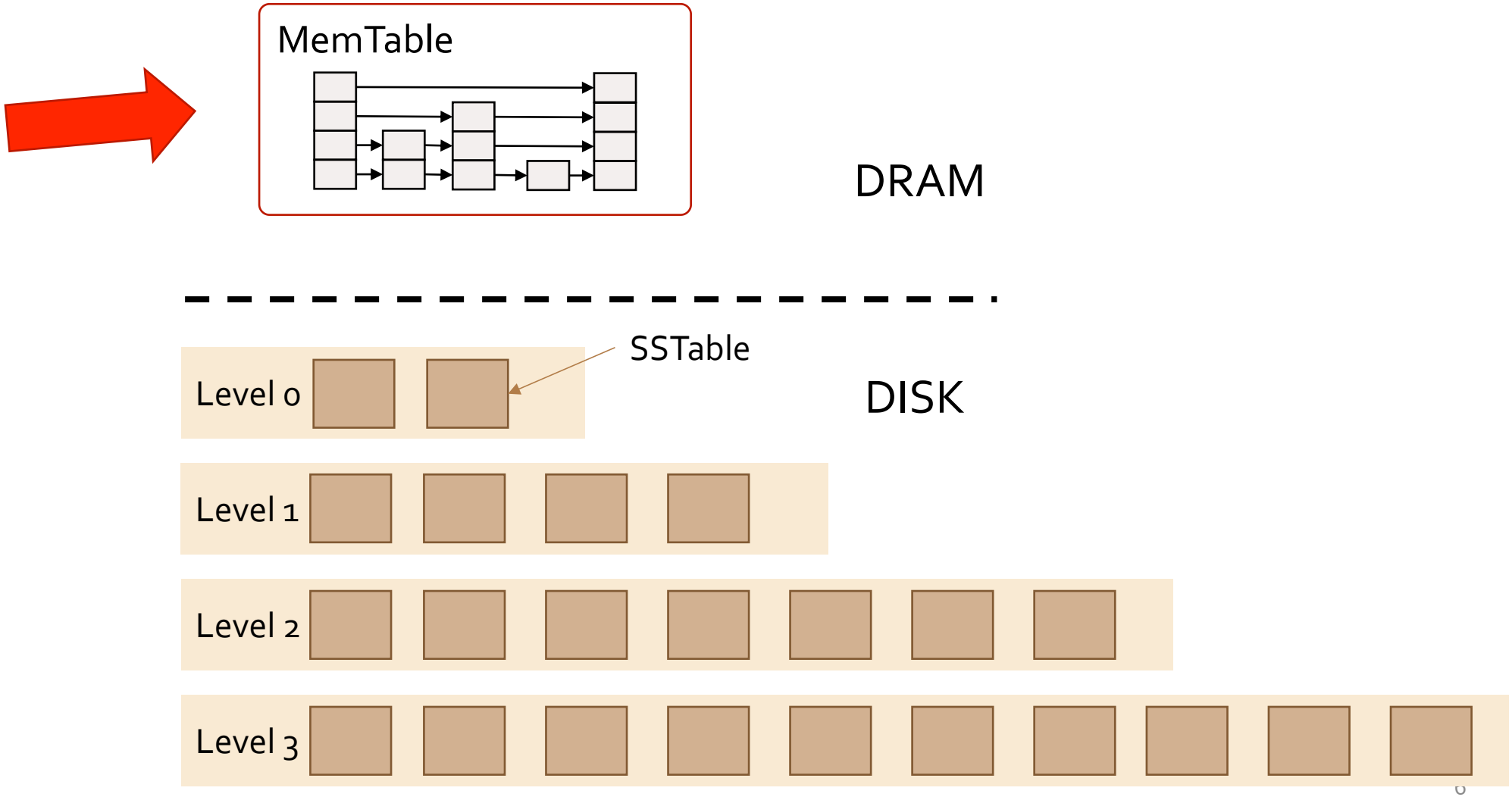
数据区

整个文件，不超过 2 MB （2*1024*1024字节）

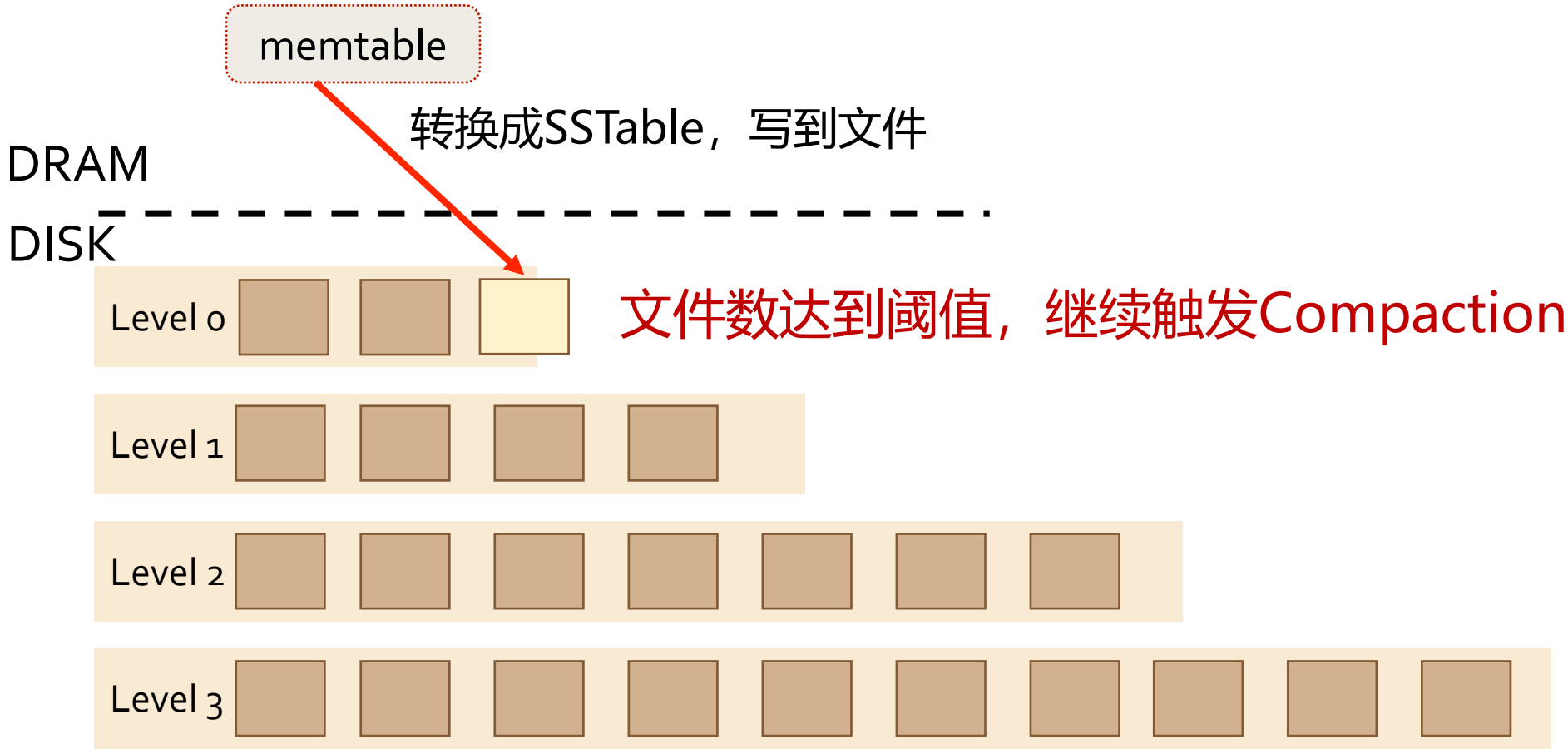
PUT(K, V)【写入】

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

PUT(K, V)



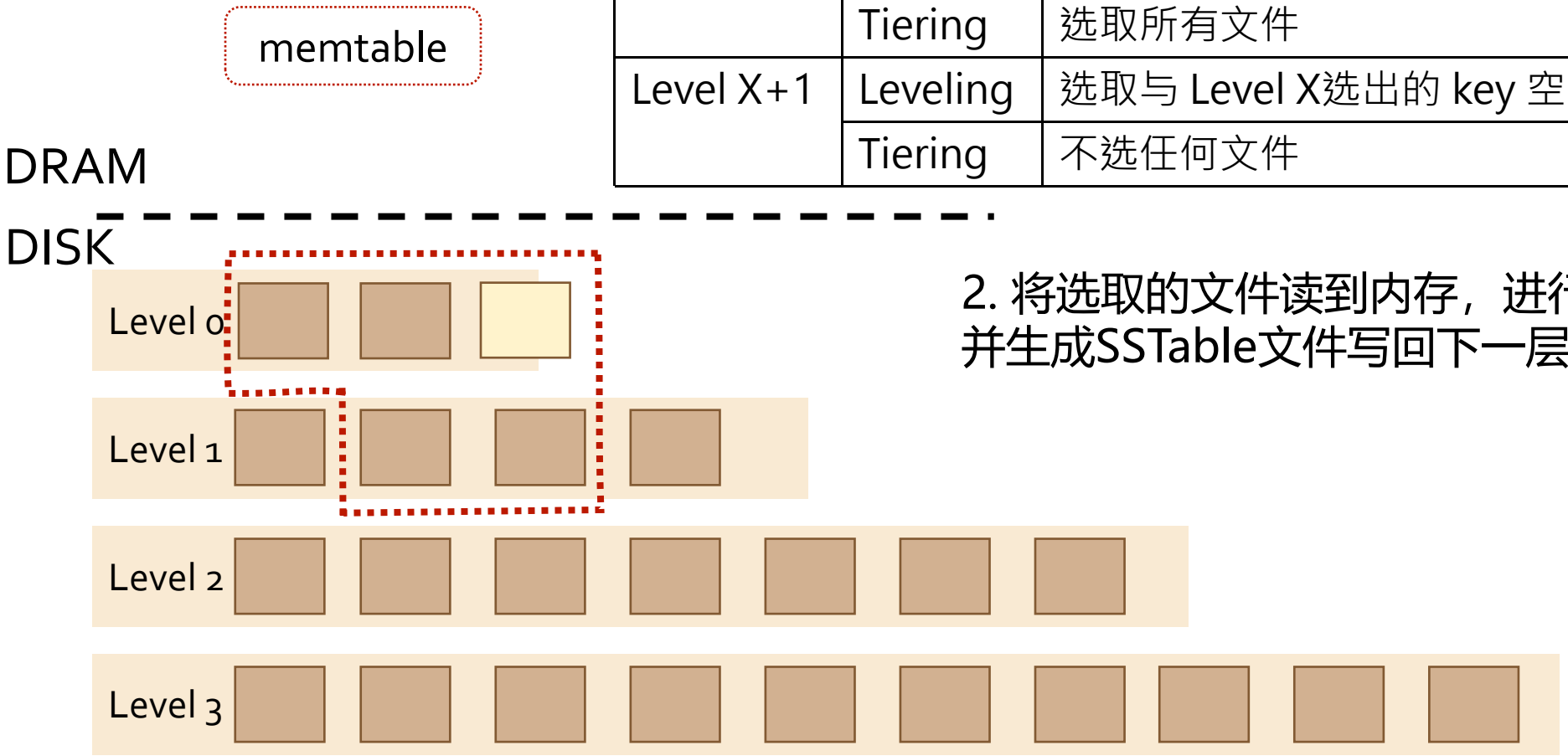
Compaction


memtable
转换成SSTable，写到文件
DRAM
DISK
Level 0
文件数达到阈值，继续触发Compaction
Level 1
Level 2
Level 3

Compaction

1. 选取SSTable文件：

层	层模式	选取方法
Level X	Leveling	按规则选取超出数量限制的文件
	Tiering	选取所有文件
Level X+1	Leveling	选取与 Level X选出的 key 空间重叠的文件
	Tiering	不选任何文件

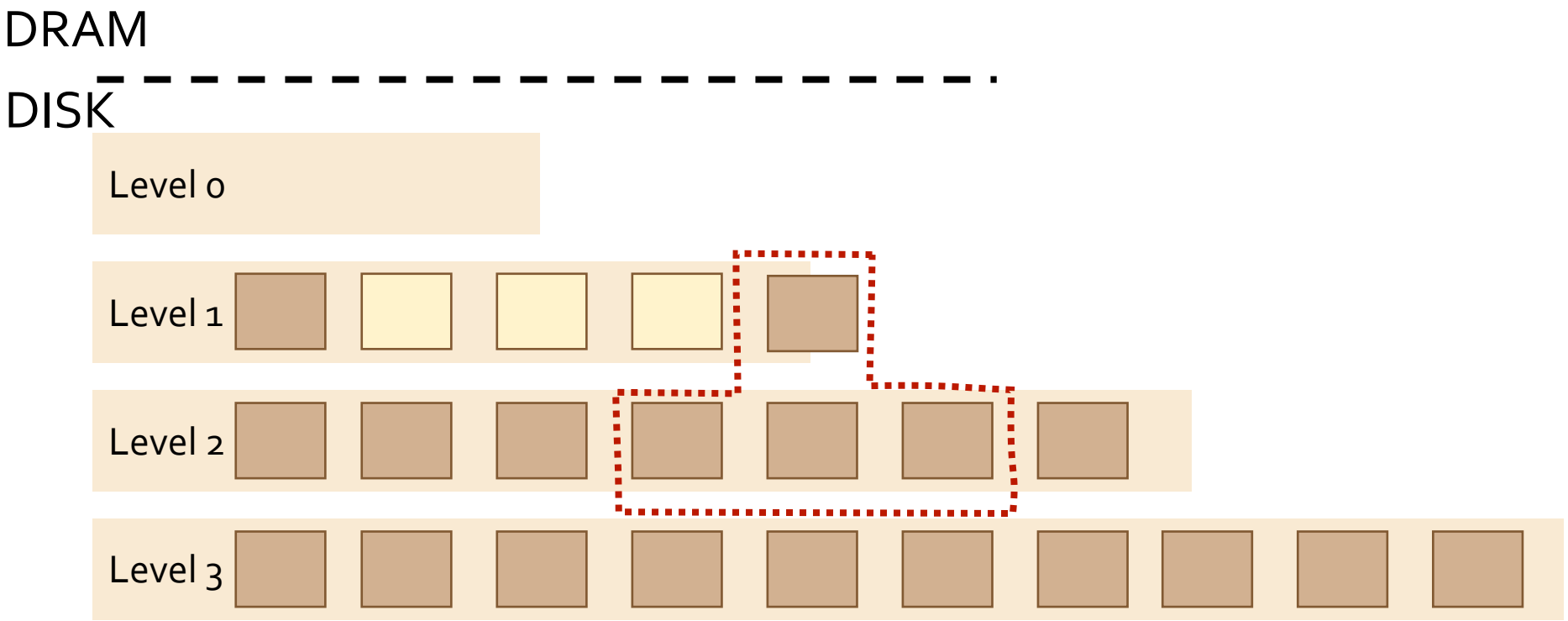


2. 将选取的文件读到内存，进行归并排序，并生成SSTable文件写回下一层（Level 1）

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

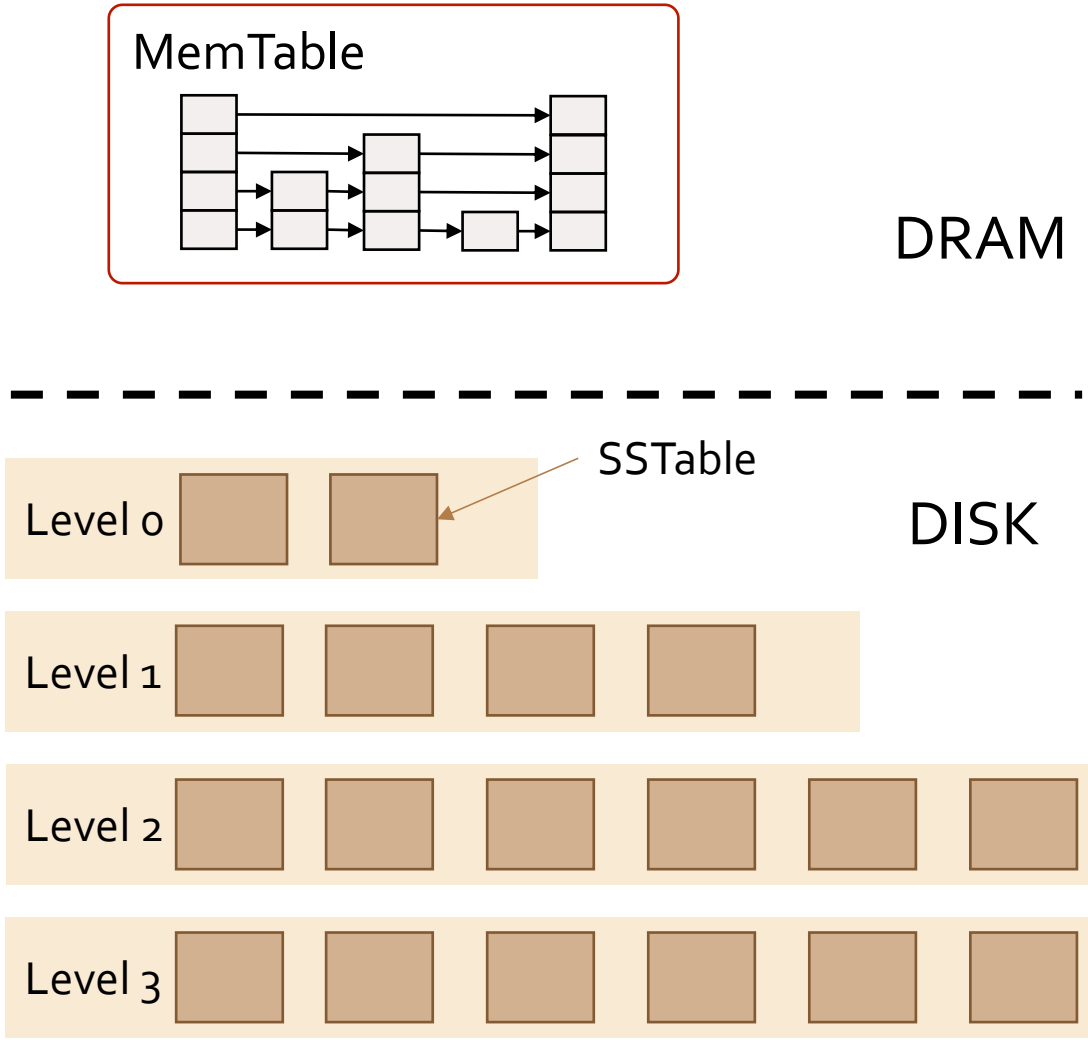


DELETE(K)【删除比较简单】

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

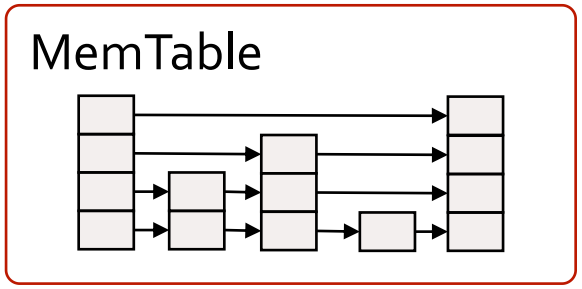
该标记同样会被 Compaction，
直到最后一层才可以被删除

Q：为什么不能提前删除?



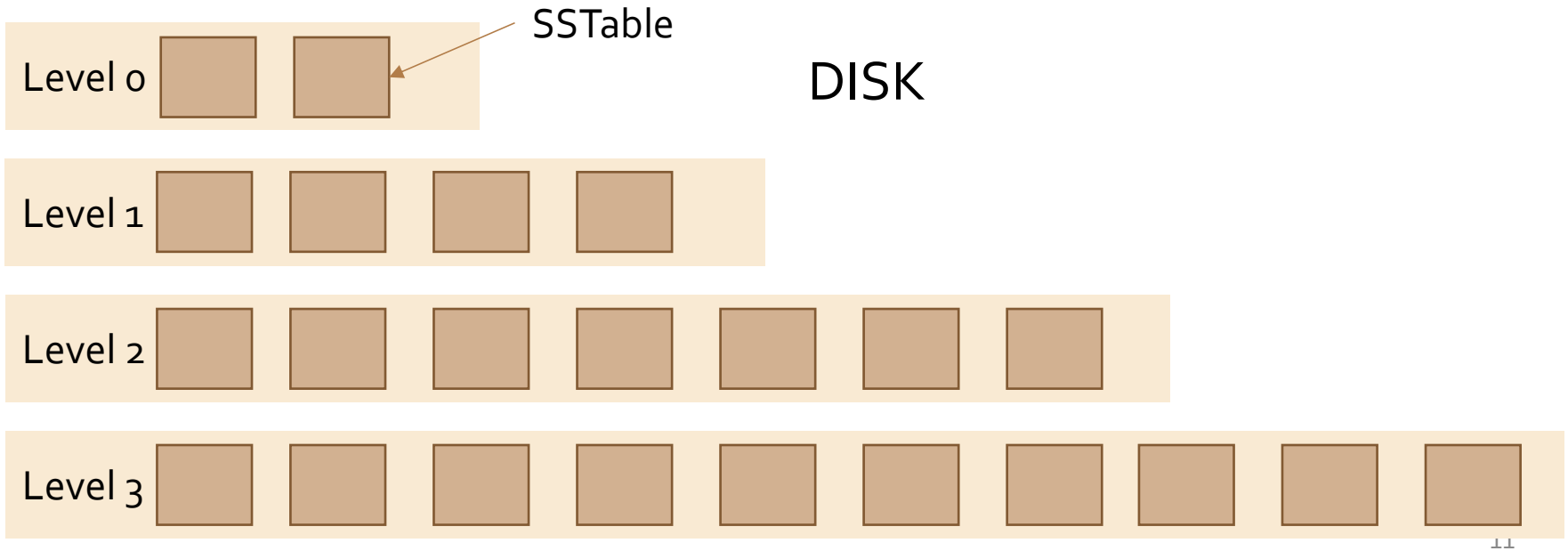
GET(K)【查找】

GET(K)

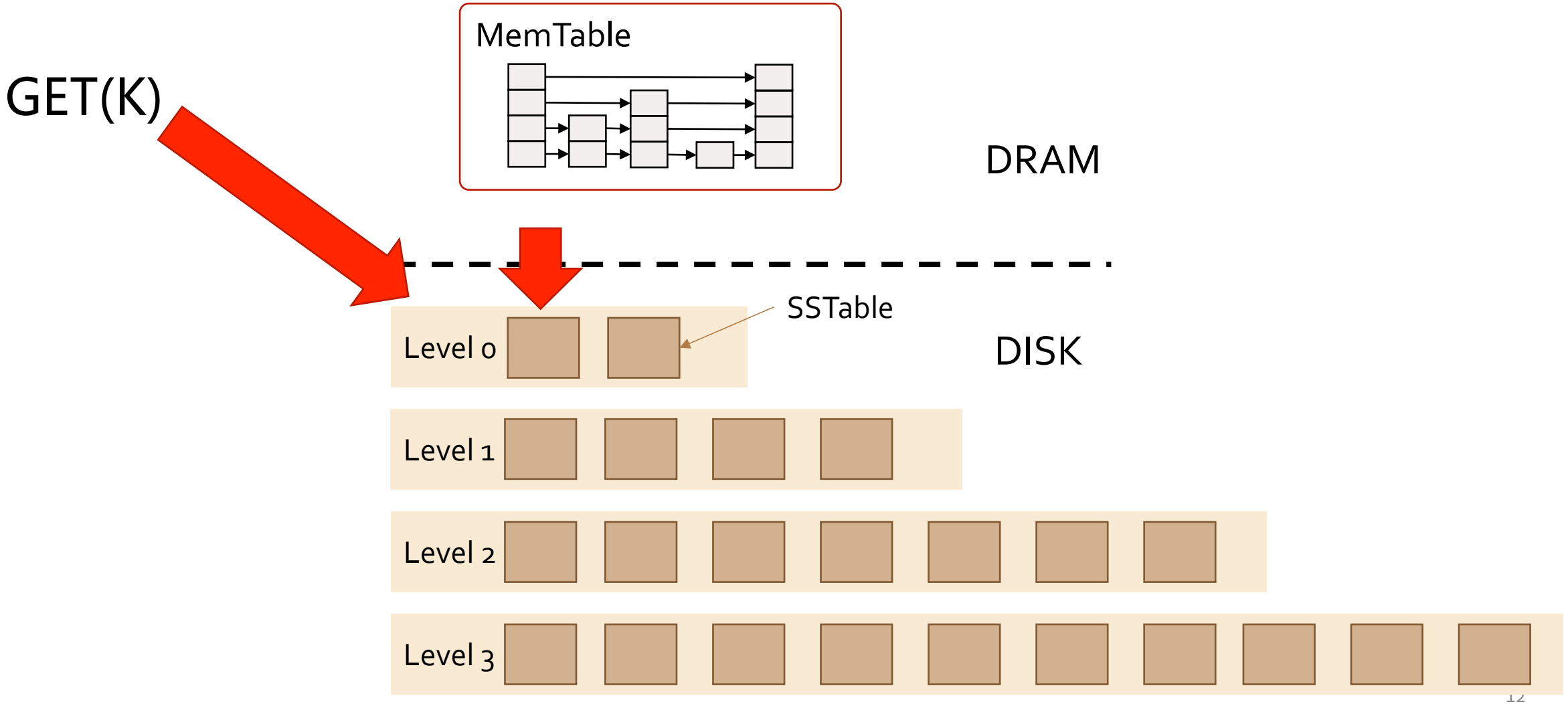


DRAM

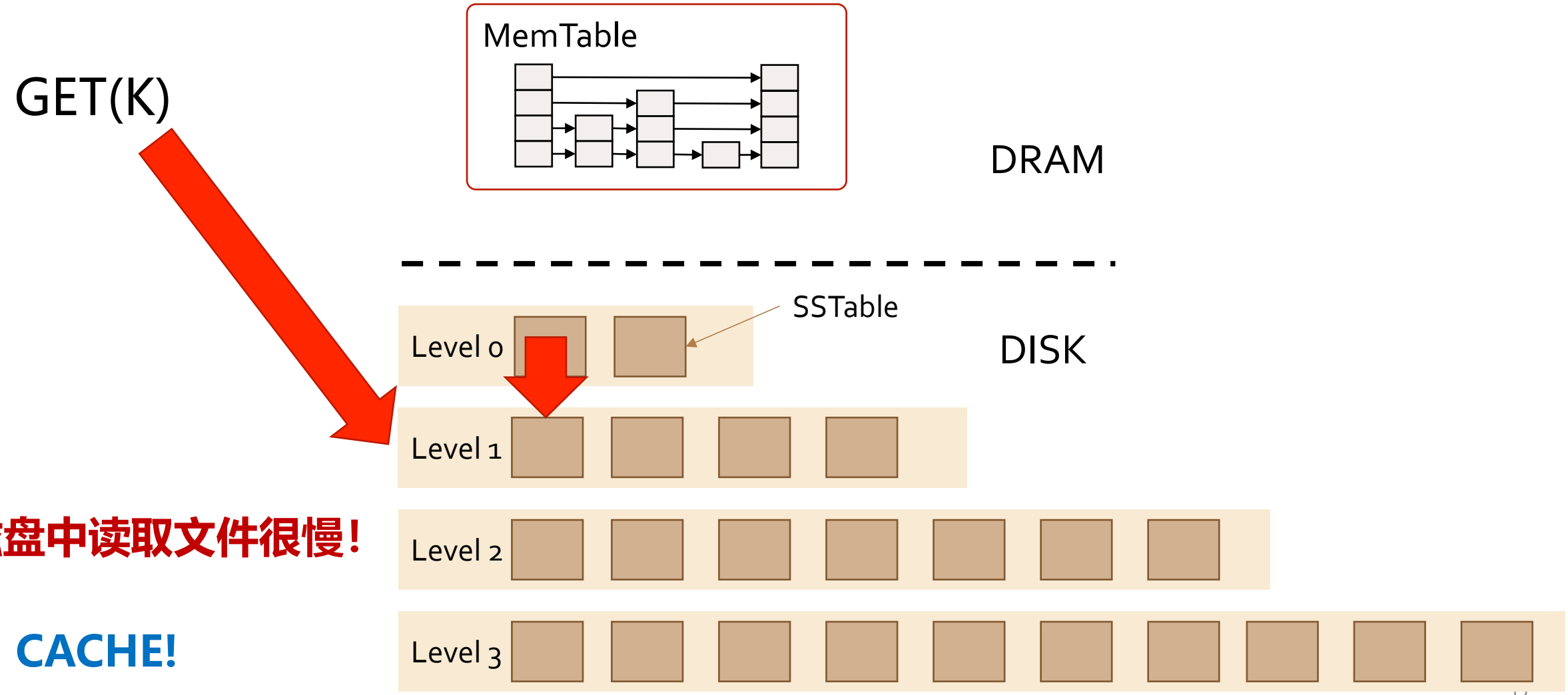
DISK



GET(K)


GET(K)
MemTable
DRAM
SSTable
DISK
Level 0
Level 1
Level 2
Level 3

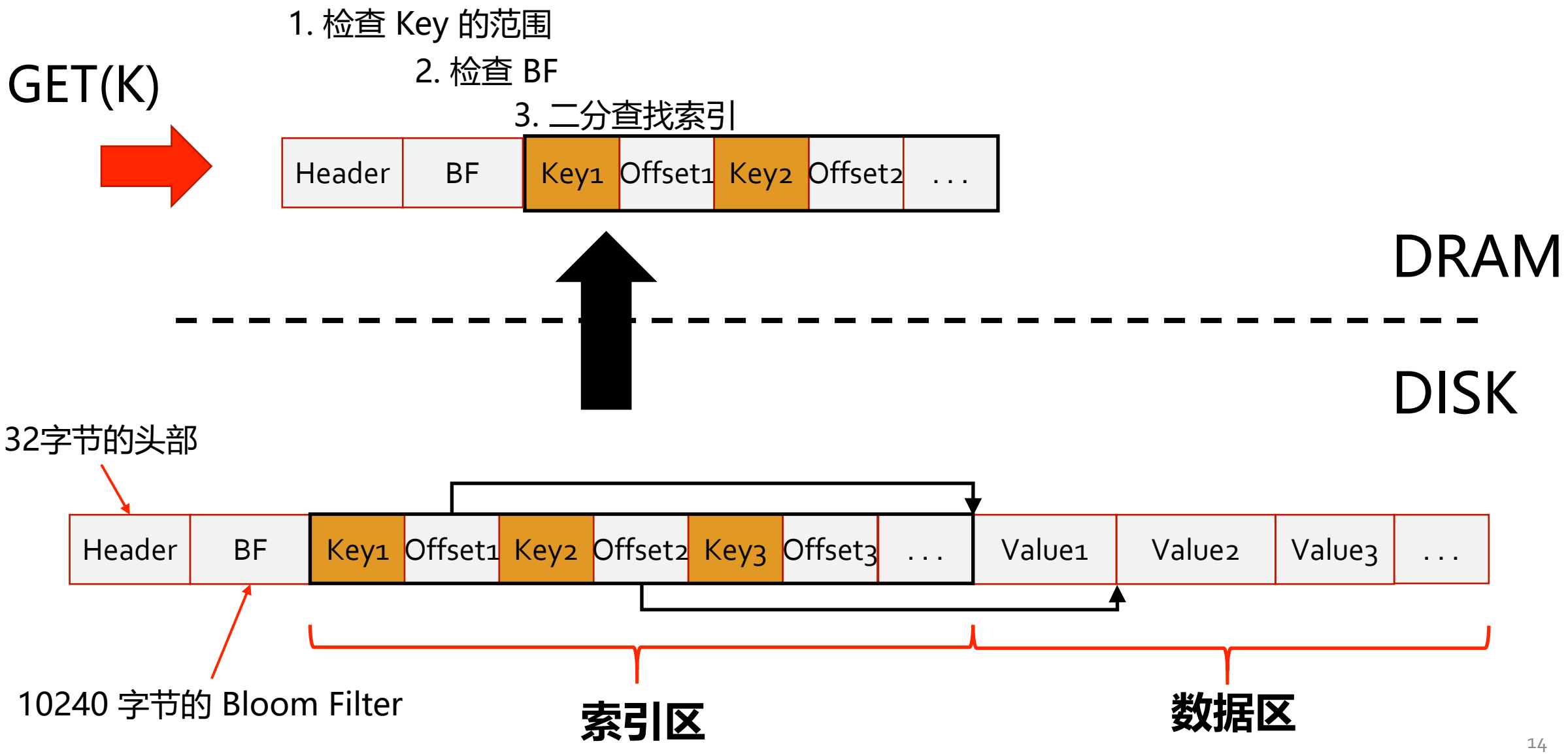
GET(K)



从磁盘中读取文件很慢!

CACHE!

优化：将索引缓存在DRAM中

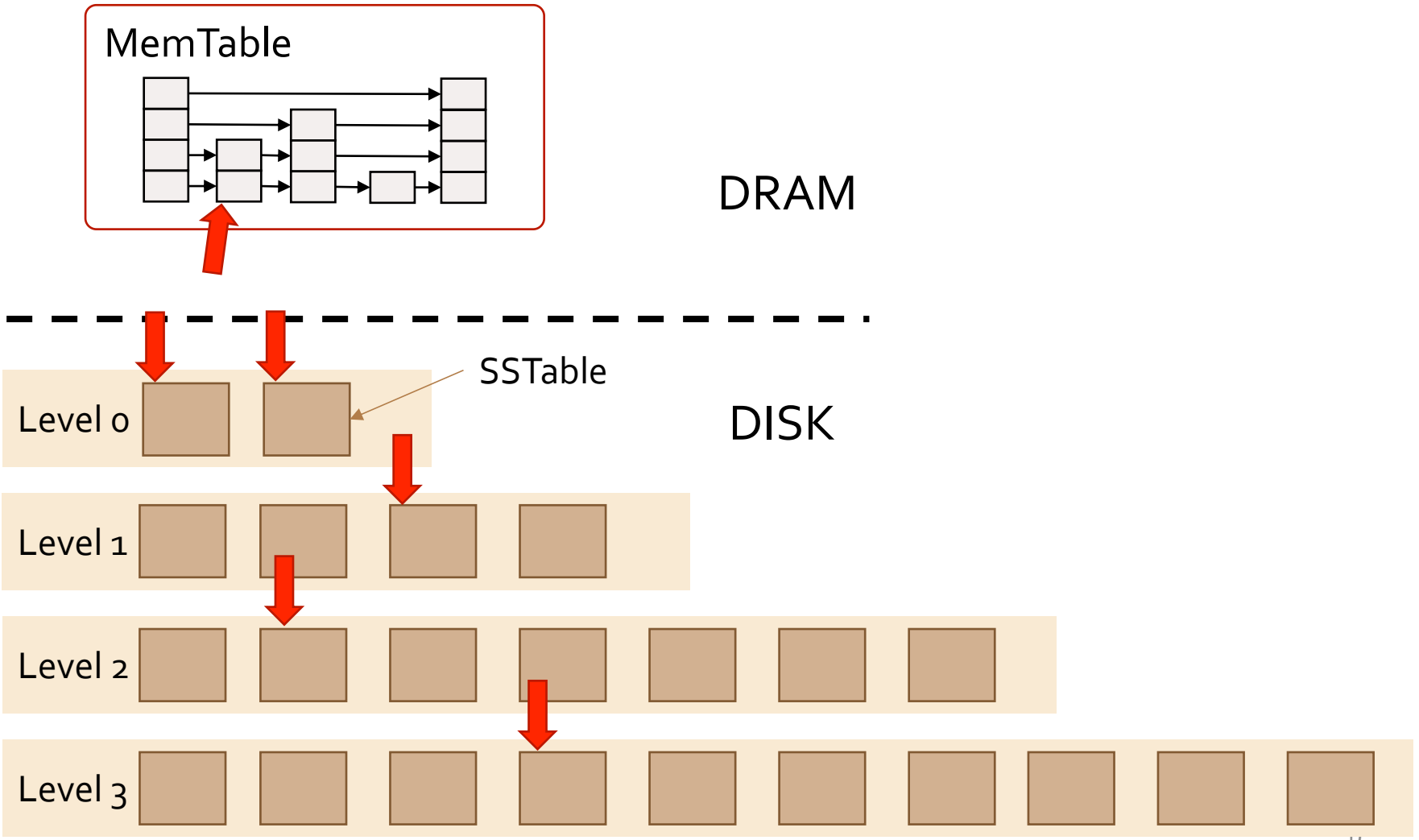


可选：SCAN(K1, K2)

SCAN(K1, K2)

一种方法：
1. 维护多个指针
2. 使用最小堆排序（排序）

能否使用
```
for (k = K1; k <= K2; ++k)
    GET(k)
```
?



启动与 Reset 操作【你关了电脑，游戏数据不能没了！】

- 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【每次检查上次剩下来的sstable】

- 在 reset() 操作被调用时，会清空所有数据【包括跳表和所有的sstable】
- 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，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
    - 对比：比较不同Levle配置对性能的影响
  - 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 键值存储系统优化

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
  - …

分阶段提交

- 阶段 1：第 4~5 周
  - 使用跳表实现 MemTable，完成基本的 PUT、GET、DEL、SCAN、RESET 操作；
  - 实现 SSTable 的生成，此阶段没有 Compaction，因此 Level 0 无限大；
- 阶段 2：第 6~7 周
  - 实现 BloomFilter、缓存等，此阶段依然没有 Compaction
- 阶段 3：第 8~10 周
  - 实现 Compaction 和其余逻辑；
- 阶段 4（即最终提交）：第 11~12 周
  - 完成测试实验和报告；
- 前 3 个阶段的提交不进行内容检查，但未提交会扣分。