

LevelDB存储引擎

网易杭州研究院---胡争（博客：openinx.github.io）

LevelDB简介

- 基于LSM实现的KV存储引擎
- Google实现/单机 C++
- 写入性能极高(40W/s)
- 读性能比写性能差(6W/s)
- 支持快照读一致性
- 支持崩溃恢复
- 采用snappy数据压缩

LSM 相关产品

BigTable

RocksDB

HBase

Cassandra

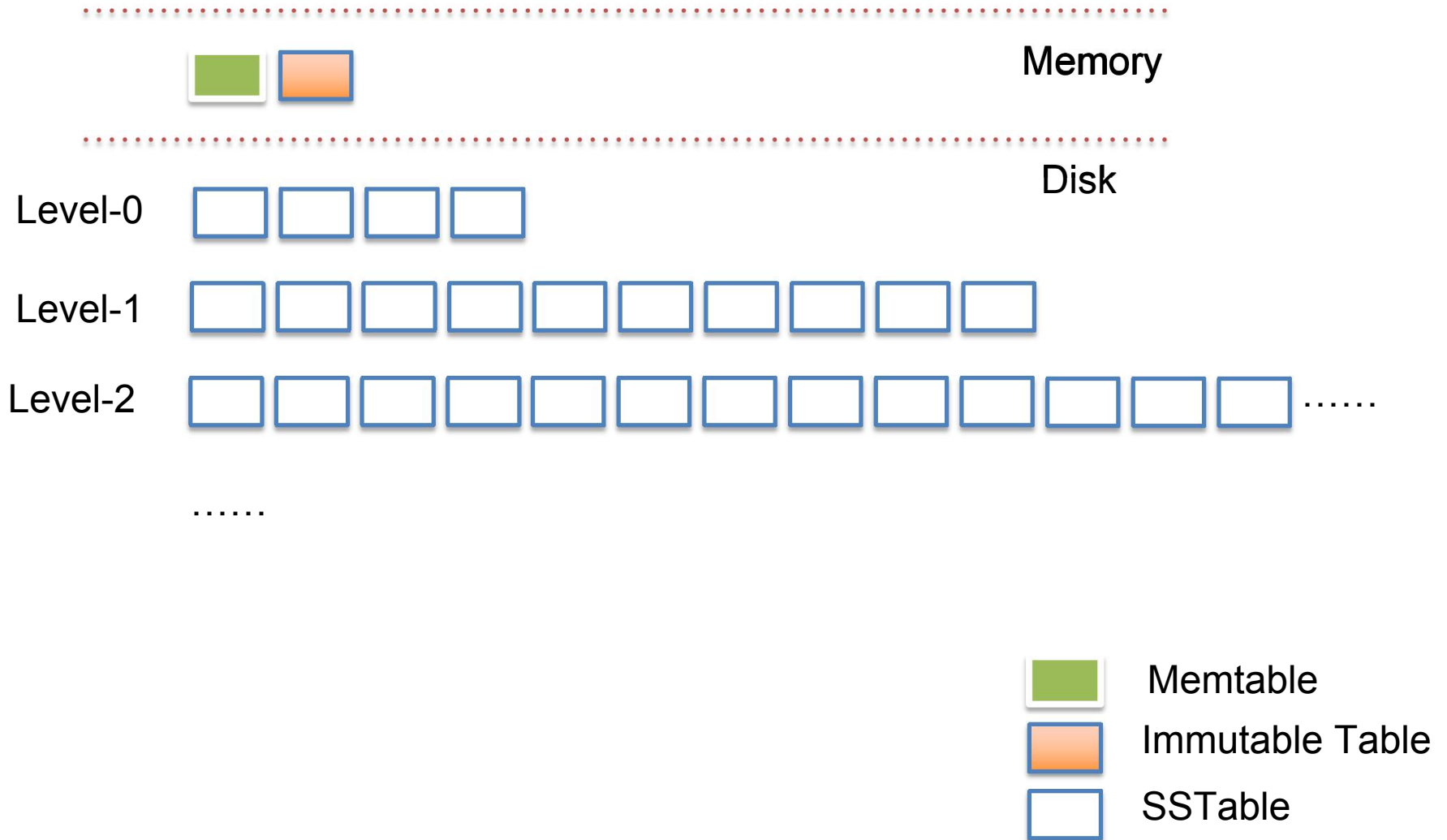
WiredTiger

LevelDB

InfluxDB

Part.1 数据结构

LevelDB-LSM结构

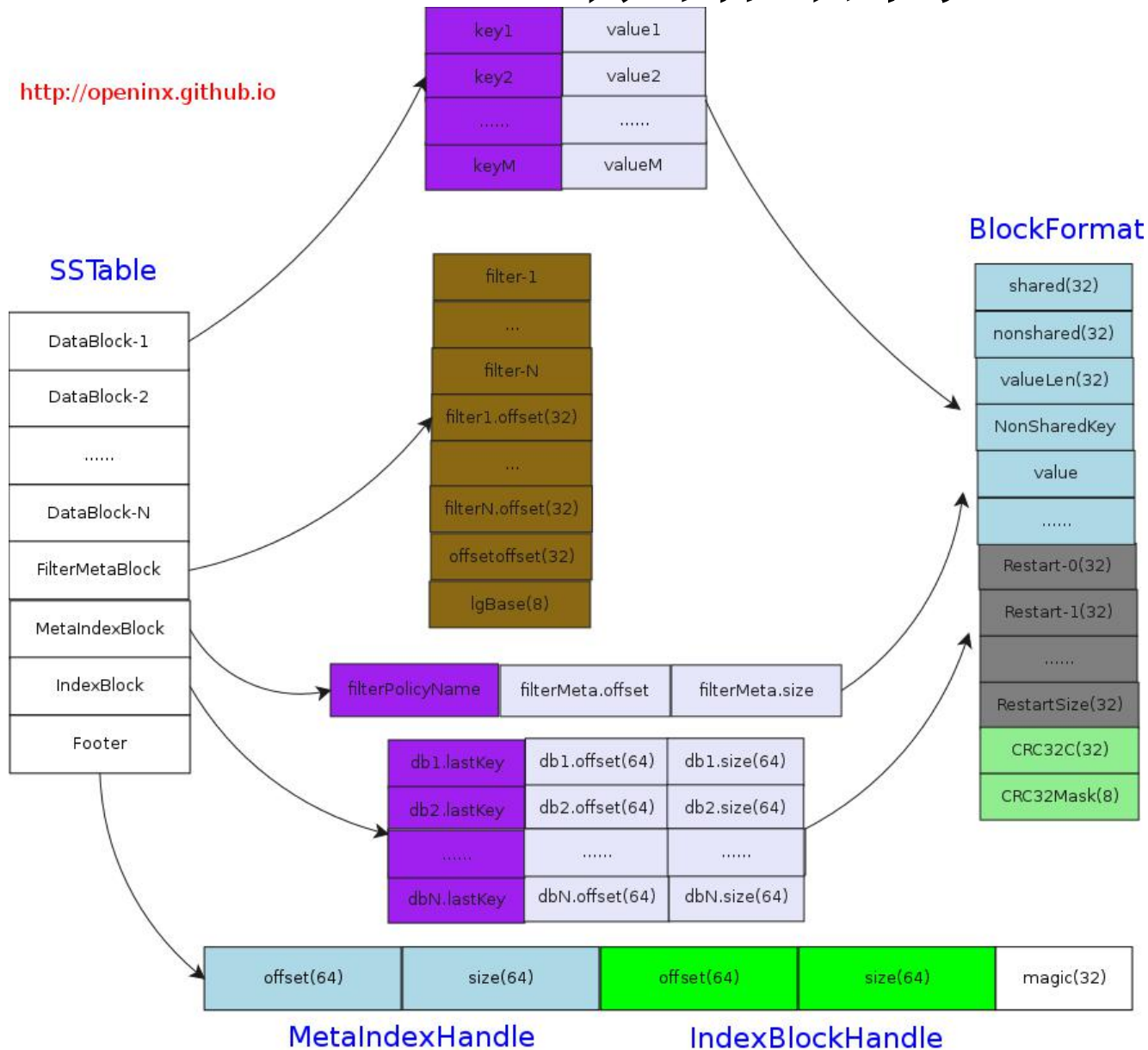


LevelDB-LSM性质

- Memtable/ImmutableTable
 - 4M左右内存块（可调整）
 - 存放有序键值对
 - 内存结构为skiplist
 - 数据先写入Memtable，再通过Compaction方式异步下推落盘。
- SSTable
 - 层次结构（而不是严格树形结构）
 - SSTable存放有序键值对
 - Level-0层SSTable之间可能存在Range交集
 - Level-i ($i > 0$) 层SSTable之间不存在Range交集
 - Level-i层SSTable个数数量级在 10^i
 - **Level-i层中每一个SSTable最多与Level-(i+1)层的10个SSTable有交集**
 - **数据新鲜度: Memtable > ImmutableTable > Level0 > Level1 > ...**

SSTable存储结构

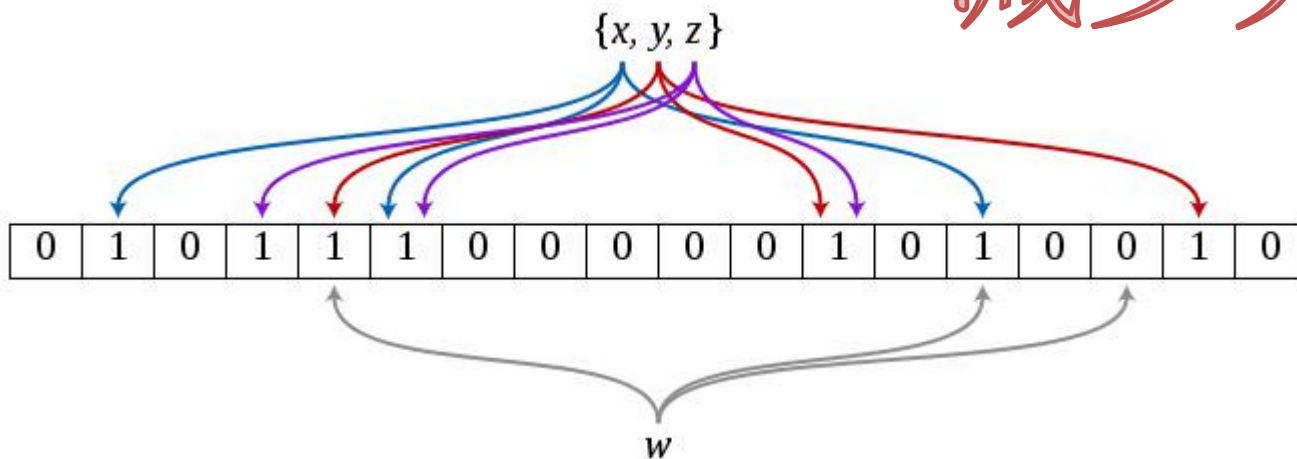
<http://openinx.github.io>



BloomFilter原理

- 随机化算法 ([BloomFilter](#))
- 解决的根本问题：
 - 判断一组key是否在集合中。
 - 答案只有两种：可能存在 or 不存在。
 - 每个键值消耗10bit，理论上可以保证可能存在出错的概率<1%.
- 特点：
 - 随机化
 - 空间换时间
 - 过滤不必要的查询

减少无效IO



前缀压缩

- 压缩规则：
 - 一个restart-point对应一个前缀压缩组；
 - 一个key和上一个key公共前缀长度 >0 ，则key放入相同前缀压缩组；
 - 一个key和上一个key公共前缀长度 $=0$ ，则新建一个前缀压缩组。
- 查找规则：
 - 多个前缀压缩组之间二分查找；
 - 前缀压缩组内线性查找；
- 配置参数：block_restart_interval
 - 前缀压缩组内键值数 $< \text{block_restart_interval}$ 。
 - 防止一个前缀组内数据太多，造成 $O(n)$ 线性时间查找。

减少磁盘数据量

Part.3 接口实现

Put/Delete操作

- 将(write, key, value)包装成writer放入writers FIFO队列;
- 等待writers队列writer之前写操作结束;
- 检查Memtable是否超过4M, 若超过4M, 则等待Compaction腾出Memtable空间。
- 获取last_sequence, 包装成(write, internalKey, value)。
 - 其中InternalKey :=
[UserKey][SequenceNumber(56Bit)][ValueType(8Bit)]
 - ValueType := kTypeDeletion | kTypeValue
- 写redo日志 (当客户端打开sync选项, 则sync到文件系统)
- 将key-value插入到Memtable。

一次顺序日志写 + 一次内存写

Get操作

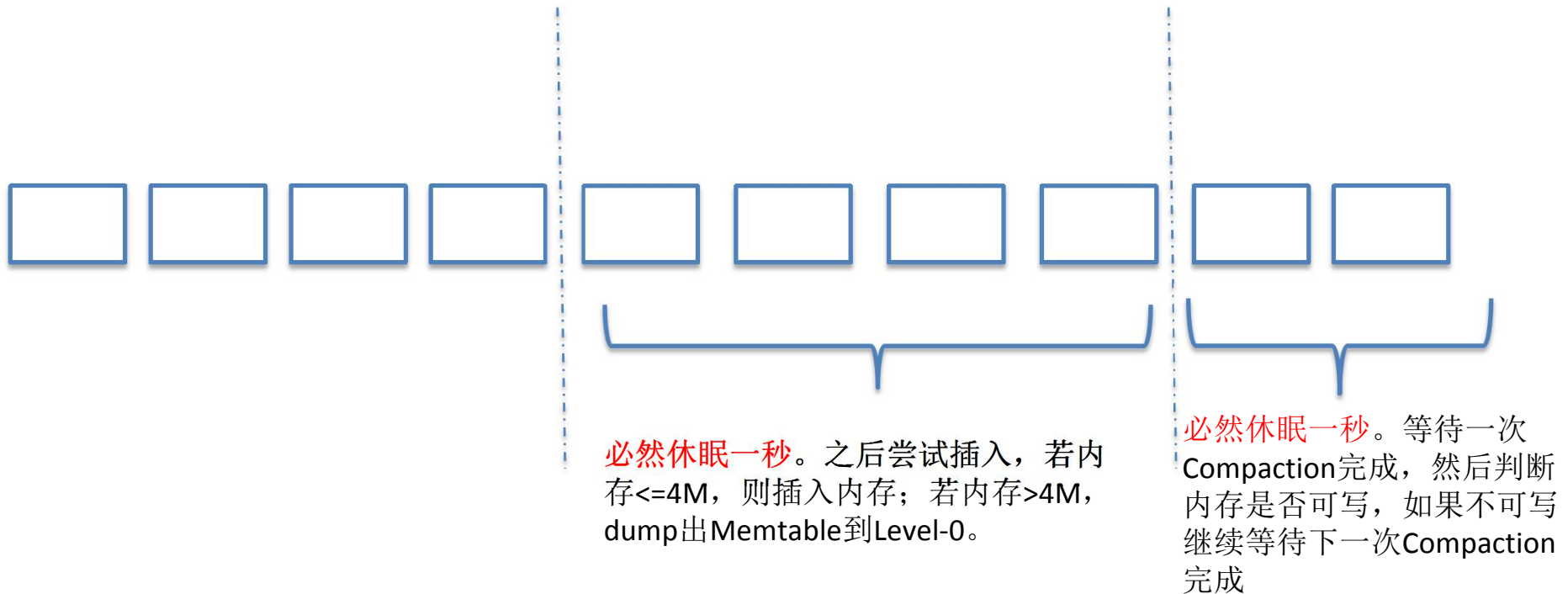
- value Get(key)
- 流程：
 - 获取最新的VersionSet。
 - 获取最新VersionSet的last_sequence。
 - lookupKey :=
[UserKey][SequenceNumber(56Bit)][ValueType(8Bit)]
 - 如果Memtable中存在lookupKey, 则返回;
 - 找出所有与key有交集的SSTable, 设该集合为TmpSSTableList.
 - 按照logNumber顺序依次遍历TmpSSTableList, 在SSTable中查找key, 若找到则停止。

最坏情况下:

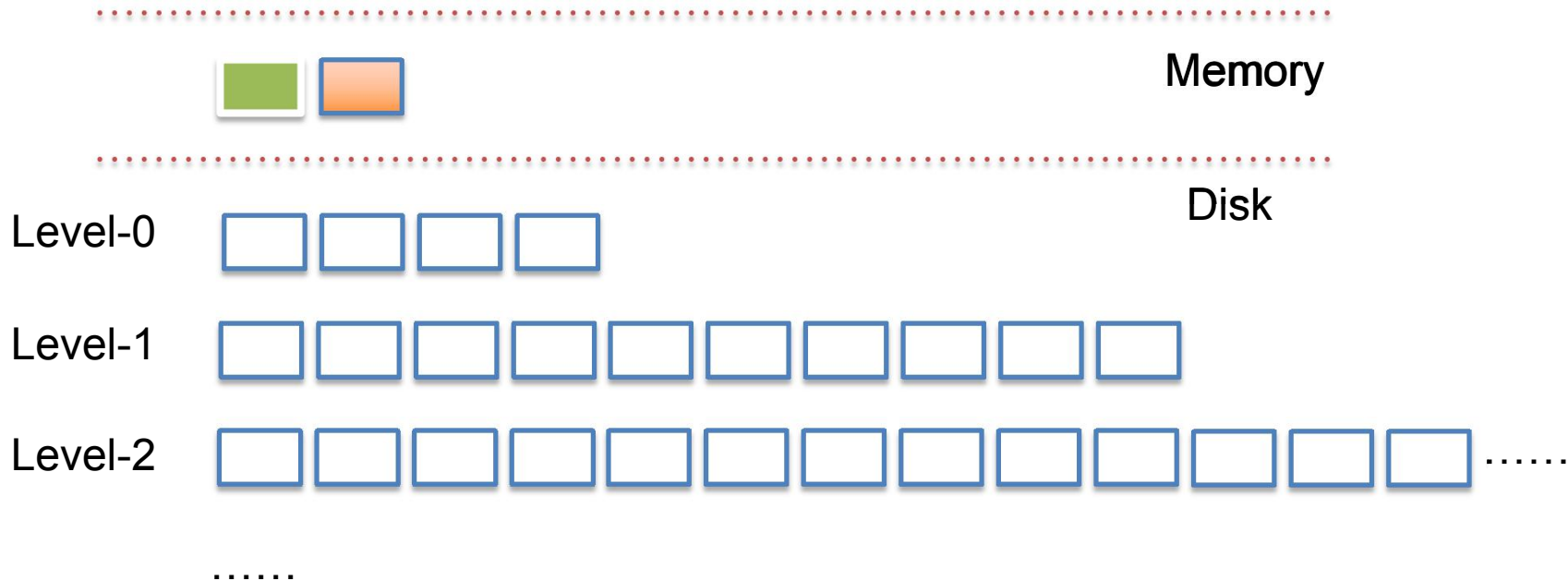
- 一次Memtable读;
- 读Level-0层所有SSTable;
- 读一次Level-1层SSTable;
- 读一次Level-i层SSTable;
- . . .




Get与Compaction

- Level-0层
 - 当sst个数 ≥ 4 时，休眠1秒，再尝试插入。
 - 当sst个数 ≥ 8 时，等待一次Compaction完成，然后判断内存是否可写，当可写时，插入内存。



一致性读



-  Memtable
-  Immutable Table
-  SSTable

Part.4 Compaction

Minor Compaction



Memory



Disk



Memory



Disk



Minor Compaction优化

- **PickLevelForMemTableOutput**

- 宗旨：把Memtable放入到尽量深的Level上
- 条件
 - Memtable与Level-0无交集；（排除Memtable放Level-1造成Level-1层的SST之间有交集）
 - Memtable与Level-(i+1)有交集，则放在i层；
 - Memtable与Level-(i+2)层交集数据量超过25*2M，则放i层；

将Memtable尽量放到下层上，可以减少低层向下层Compaction的次数

Major Compaction



Memory



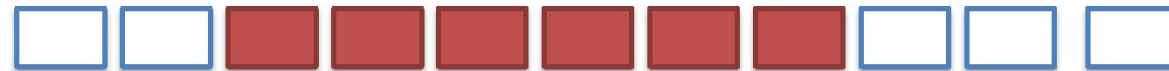
Disk



Memory



Disk

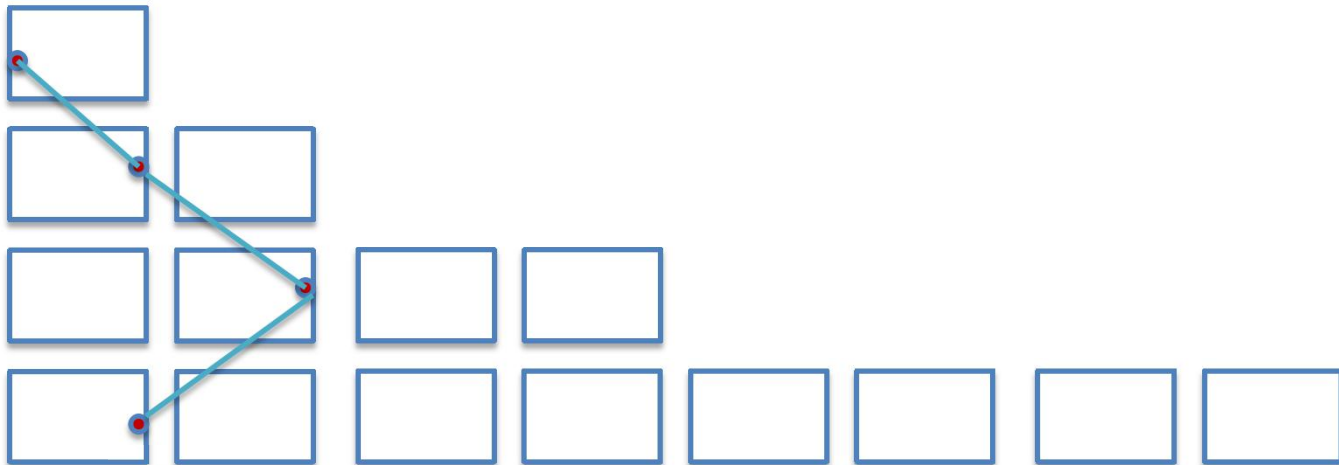


Compaction触发条件

- Compaction触发条件：
 - Memtable内存空间>4M
 - 调用Get这个API的过程中，发现seek的第一个sstable的AllowedSeek消耗完了（优化）；
 - 第0层的sstable超过8个；
 - 第 $i(i>0)$ 层的所有sstable的占用空间超过 10^i M；

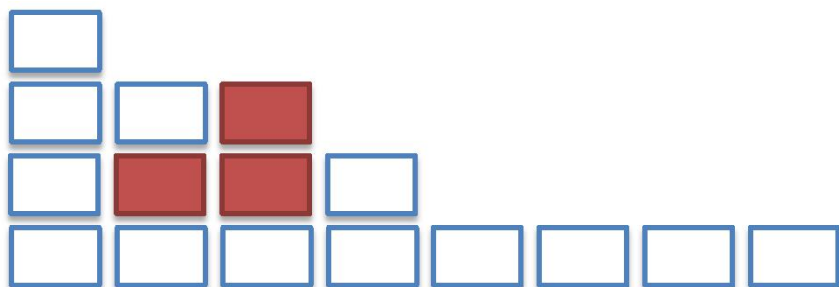
Pick-Compaction

- 如何选择SSTable，进行Compaction？
 - seek_compaction:
 - Get操作第一次Seek的SST的allowed_seeks被用完。
 - size_compaction:
 - 选择 $\text{MAX}(\text{TotalFileSize}(\text{level}[i]) / 10^i)$ 最大的层，该层数据失衡指数最大，因此要优先Compaction。
 - 选择该层CompactionPointer之后的第一个SSTable。保证Compaction操作依次落在同一层的各个SSTable。

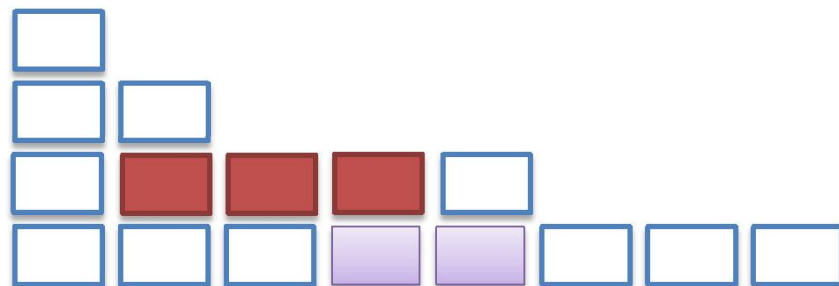


Compaction 与多版本

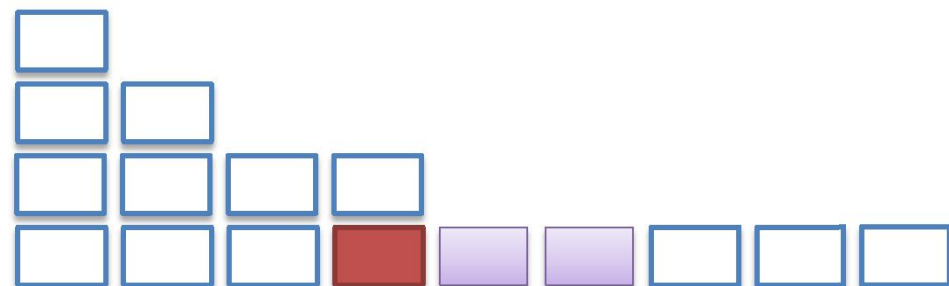
在Snapshot Read时，发生了Compaction怎么办？



Version-1



Version-2



Version-3

Thank you