Alibaba Group 阿里巴巴集团

RocksDB/MyRocks源码学习—读

基础架构事业群–数据库技术–数据库内核

王禹杰

内容

- RocksDB读流程
- Handler层
- RocksDB的其他部分

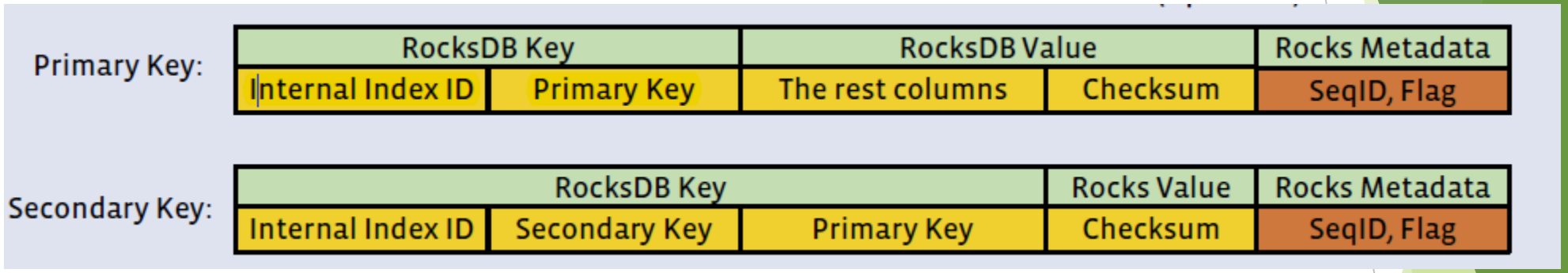
RocksDB读流程

存储格式

- 除L0层之外，数据在一层中有序，排序的依据依次是User Key升序，Sequence Number降序，类型降序。
- 除L0层之外，一层中的各文件所存储的Key范围不会重叠。
- L0层的文件内部有序，但各文件之间可能有重叠。

RocksDB读流程

- PK
- SK



- Key格式
  - Internal Key

  | User key (string) | sequence number (7 bytes) | value type (1 byte) |
  - Lookup Key

  | Size (int32变长)| User key (string) | sequence number (7 bytes) | value type (1 byte) |

* 图来自 Yoshinori Matsunobu的 MyRocks Deep Dive

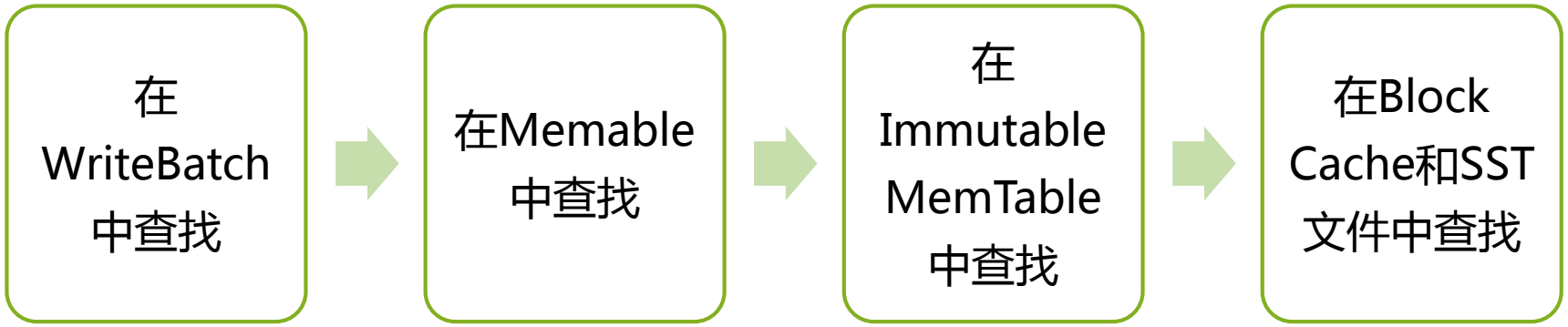
RocksDB读流程

Value Type

```
enum ValueType : unsigned char {
  kTypeDeletion = 0x0,
  kTypeValue = 0x1,
  kTypeMerge = 0x2,
  kTypeLogData = 0x3,                    // WAL only.
  kTypeColumnFamilyDeletion = 0x4,       // WAL only.
  kTypeColumnFamilyValue = 0x5,          // WAL only.
  kTypeColumnFamilyMerge = 0x6,          // WAL only.
  kTypeSingleDeletion = 0x7,
  kTypeColumnFamilySingleDeletion = 0x8,  // WAL only.
  kTypeBeginPrepareXID = 0x9,             // WAL only.
  kTypeEndPrepareXID = 0xA,               // WAL only.
  kTypeCommitXID = 0xB,                   // WAL only.
  kTypeRollbackXID = 0xC,                 // WAL only.
  kTypeNoop = 0xD,                        // WAL only.
  kMaxValue = 0x7F                        // Not used for storing records.
};
```

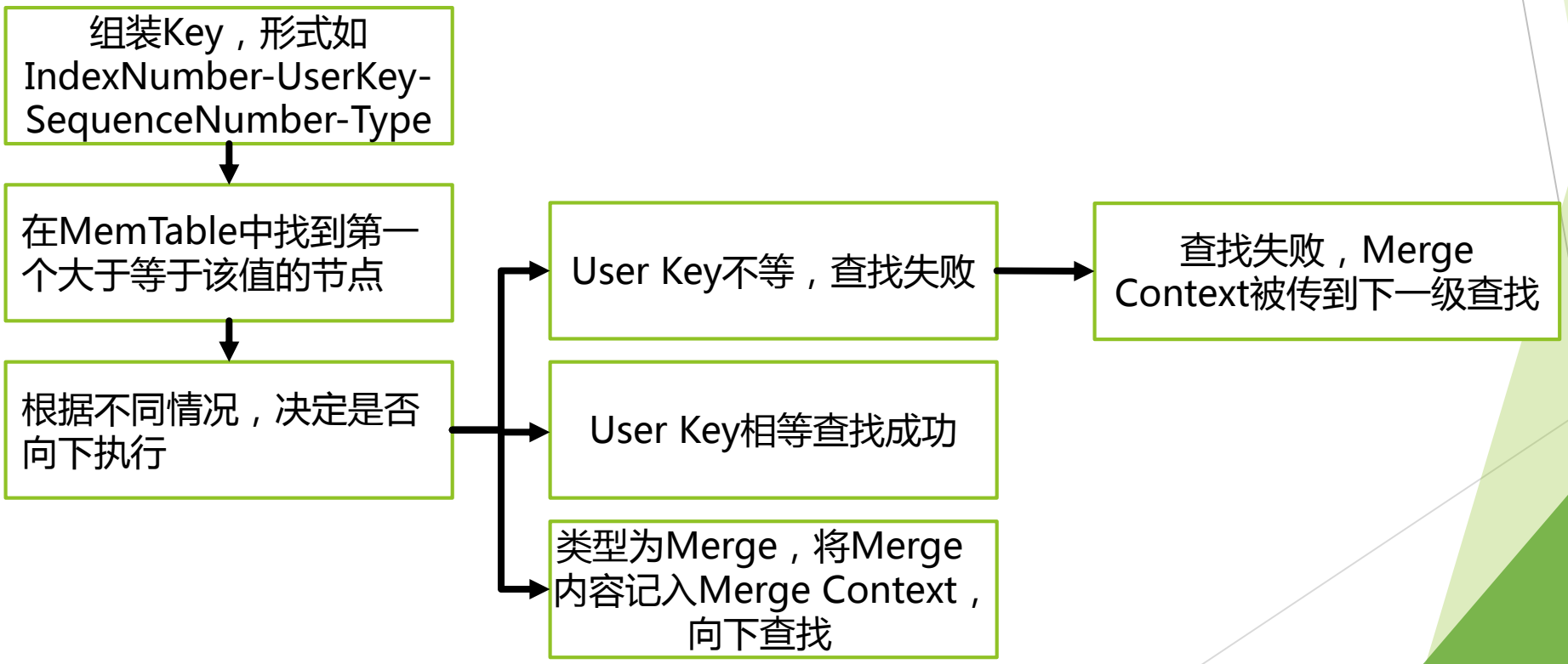
RocksDB读流程

select * from t where id=1



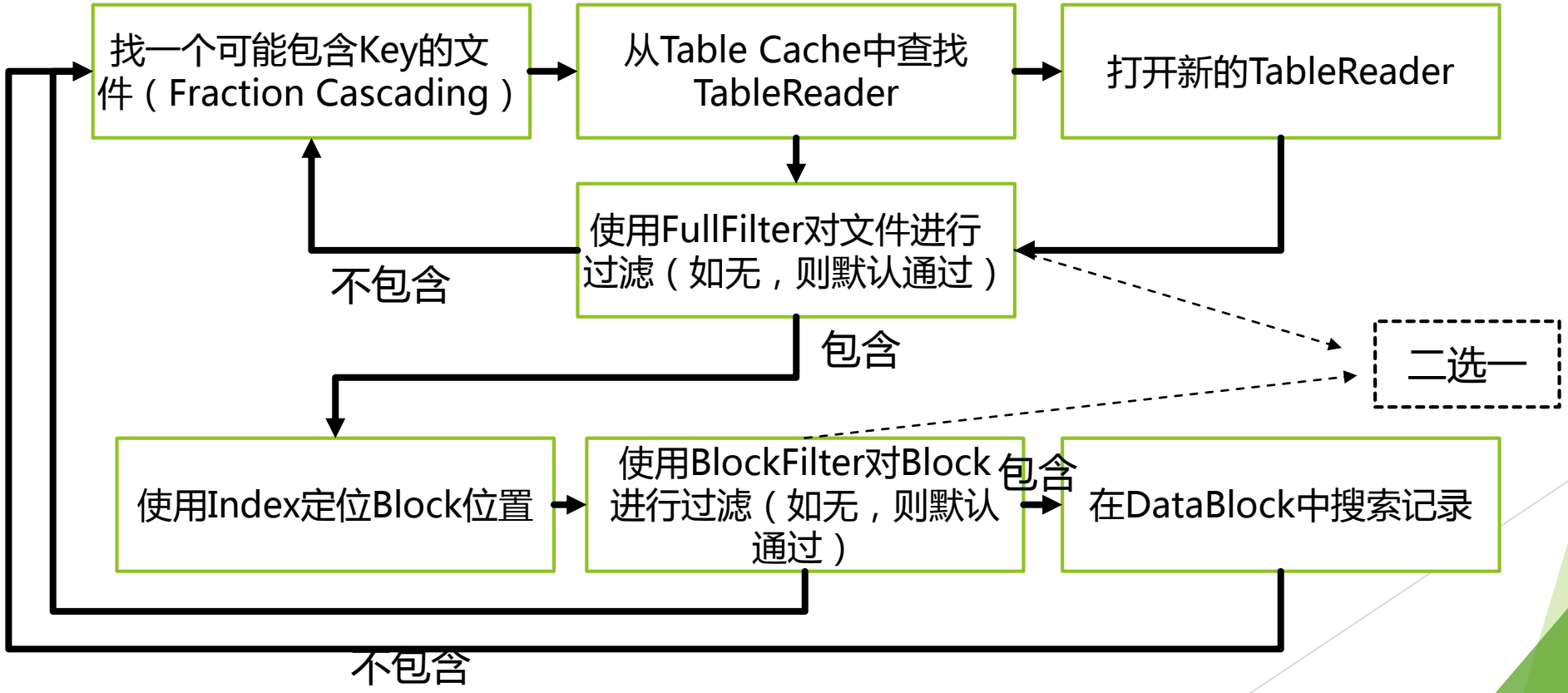
RocksDB读流程

select * from t where id=1



RocksDB读流程

select * from t where id=1



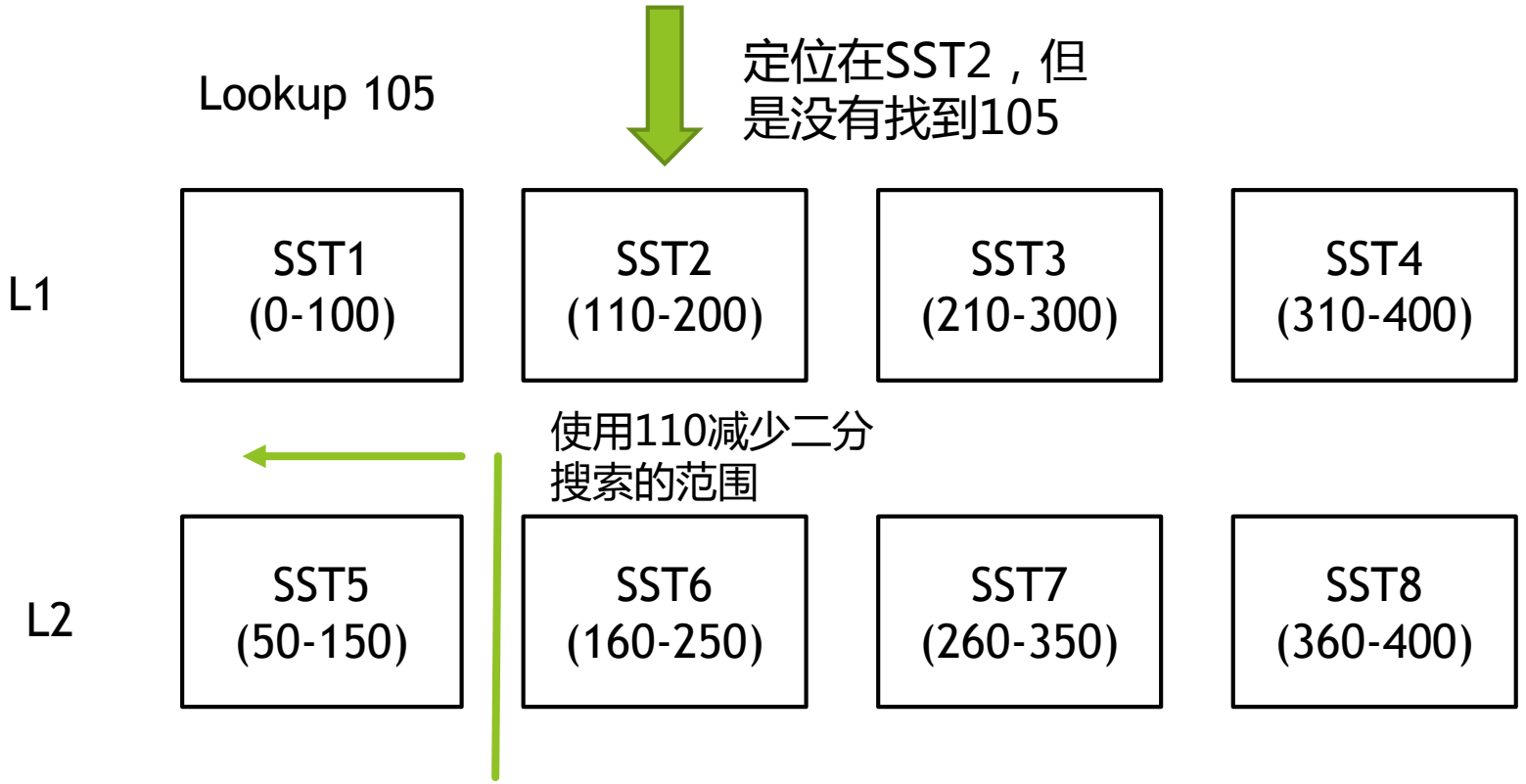
RocksDB读流程

Fraction Cascading

- 1层以上并且该层文件数多于3才会执行。
- 在某一层进行查询时，会根据比较的结果设置查询的上限和下限。
- 在下一层进行查询时，会使用这个查询的上限和下限减少文件搜索的范围。

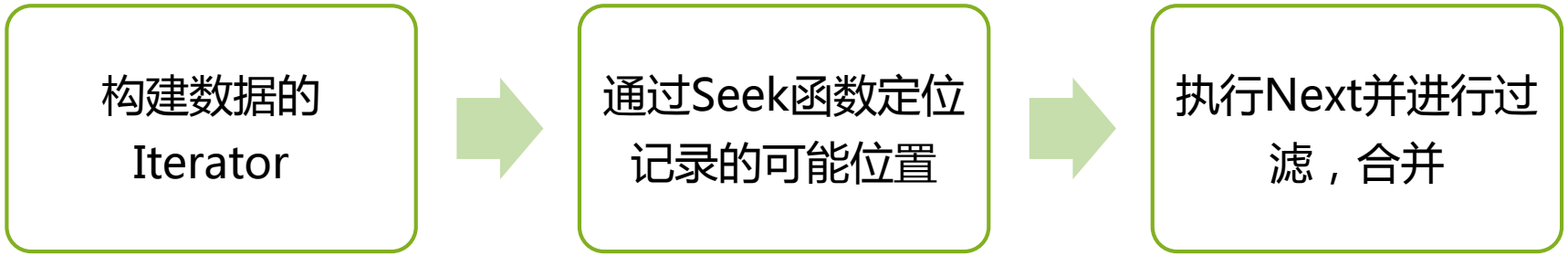
RocksDB读流程

Fraction Cascading



RocksDB读流程

select * from t where id>=1

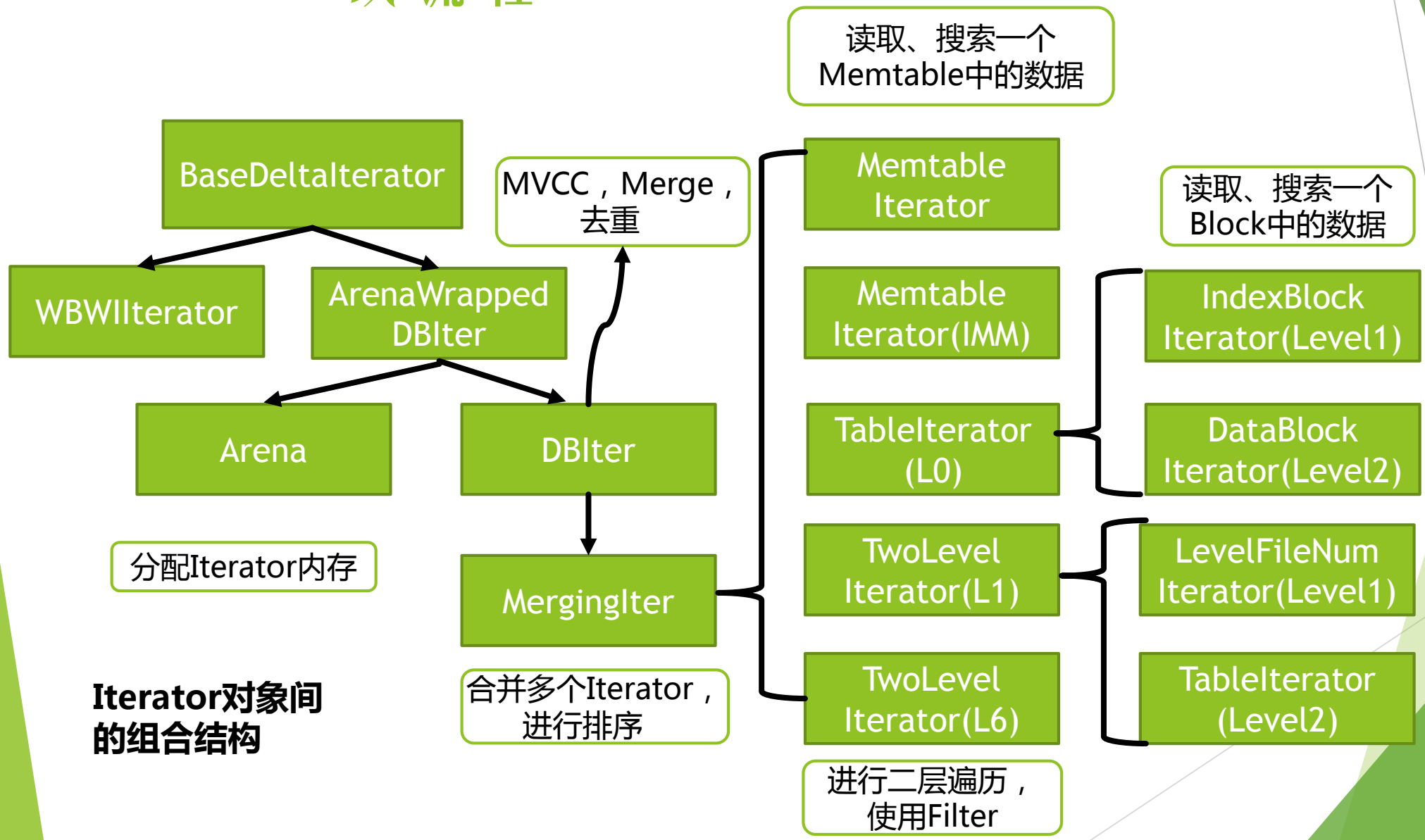


RocksDB读流程

select * from t where id>=1

- Iterator
  - Seek
  - Next
  - Prev
  - SeekToFirst
  - SeekToLast
  - key
  - value

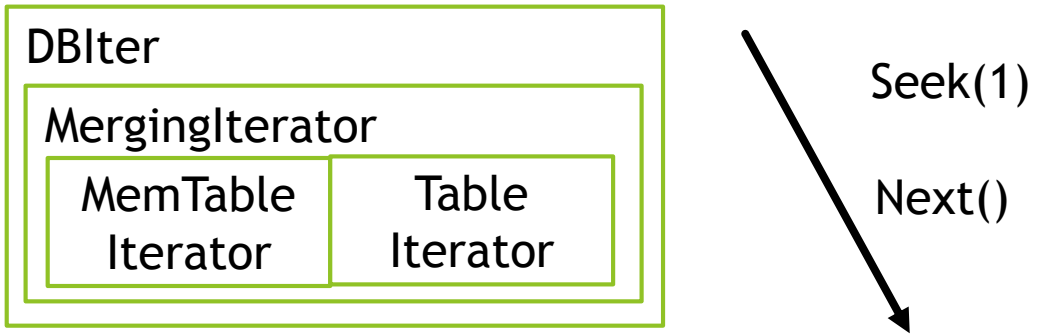
RocksDB读流程



Iterator对象间的组合结构

RocksDB读流程

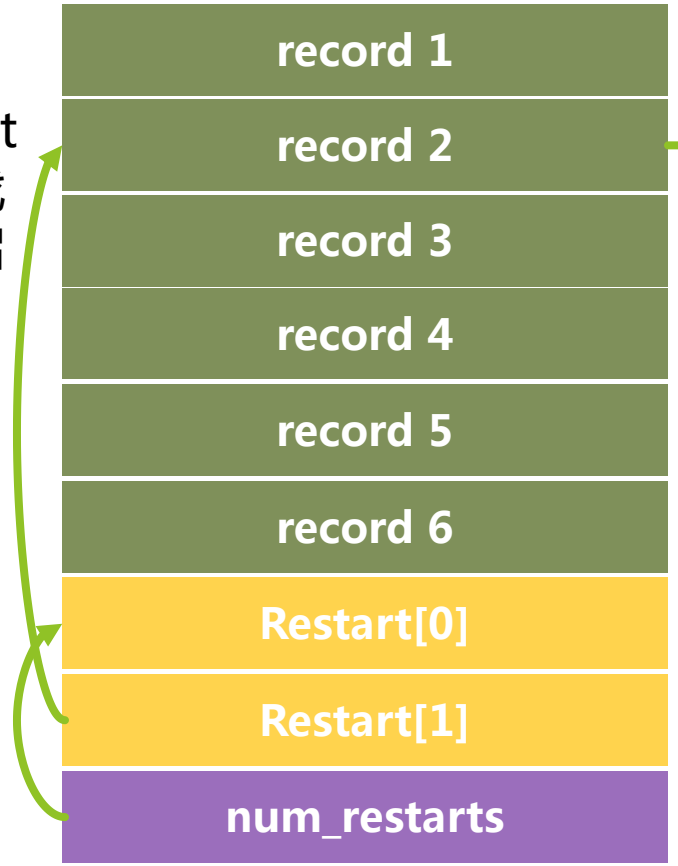
select * from t where id>=1



上层Iterator的Seek或Next函数会调用下层Iterator的Seek或Next函数，取决于上层Iterator的工作内容，下层Iterator的相应函数可能会调用多次。

RocksDB读流程

select * from t where id>=1 Block内部搜索



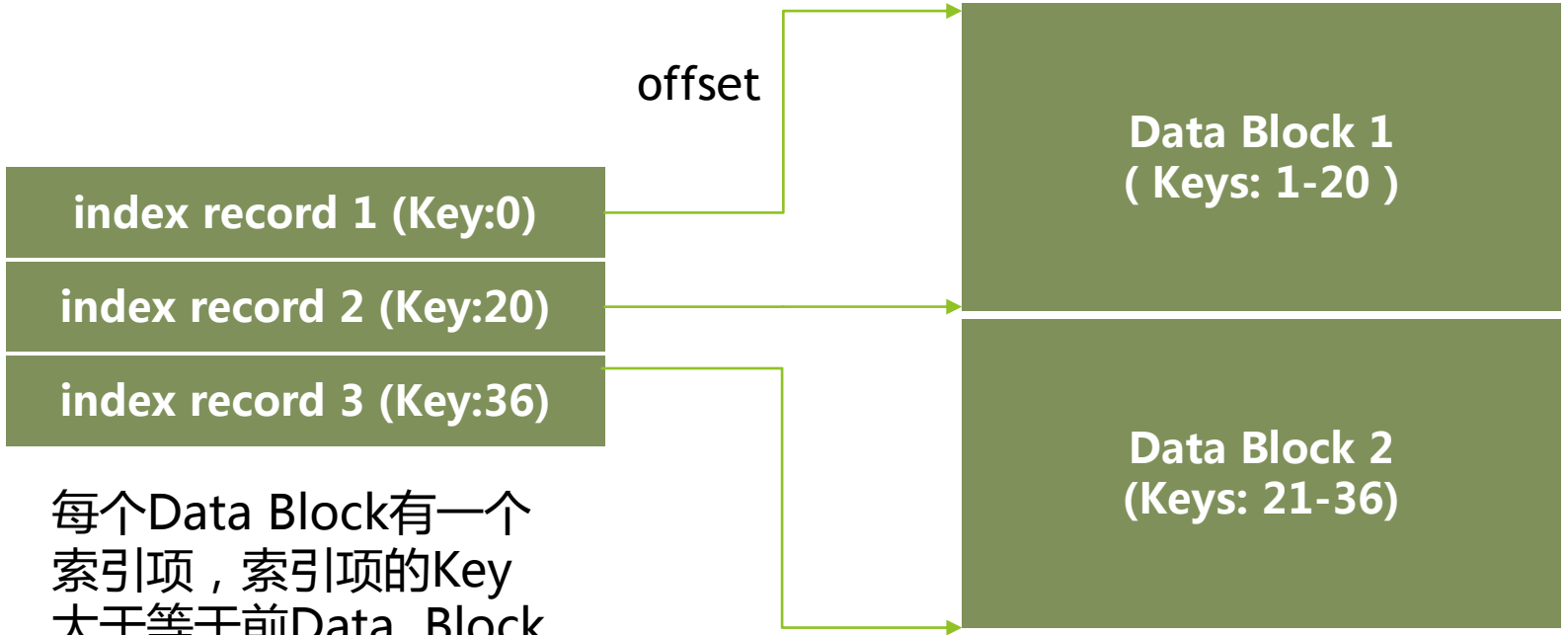
2. 在Restart Point区域二分查询，找到恰好较小的重启点。

1. 从结尾读取num_restarts获得Restart数组的起始位置，由于每个Restart项长度固定，可以计算出第i个项的位置。

3. 使用重启点定位一条Record，向下顺序搜索，找到恰好较大或等于的记录。

RocksDB读流程

select * from t where id>=1 Index搜索

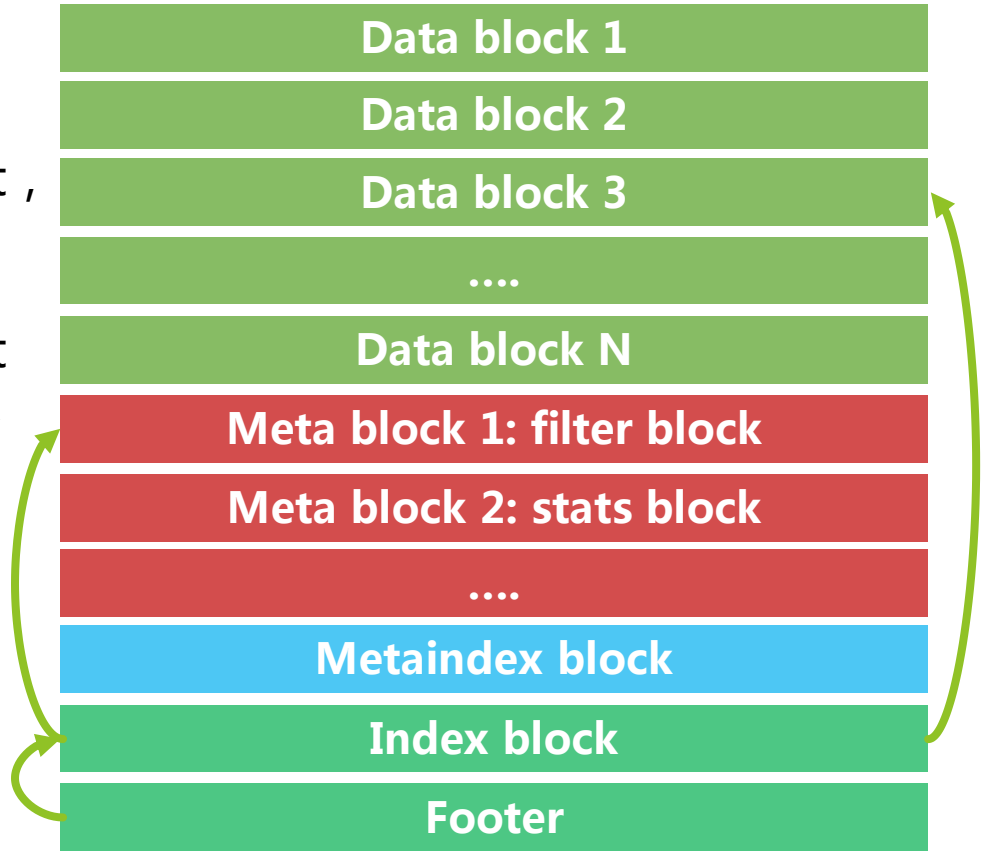


每个Data Block有一个索引项，索引项的Key大于等于前Data Block的最大Key，小于后Data Block的最小Key.

RocksDB读流程

select * from t where id>=1 文件内部搜索

2. 使用Index Block得到offset，计算filter编号，然后使用Filter Block中的offset array获得filter，使用filter与Key判断是否Match



3. 根据offset，获取Data Block

1. 从Footer定位IndexBlock

RocksDB读流程

select * from t where id>=1 层级搜索

- 通过VersionStroageInfo类可以获得文件元数据，包括每个文件包含Key的最大值和最小值等，这些信息随文件的变化而维护。
- 通过二分搜索找到可能包含Key的文件。
- 对于第0层，由于范围重叠，必须要扫描每个文件。

RocksDB读流程

select * from t where id>=1 合并与遍历

使用堆排序将多层列表合并为一个 → 在遍历这个有序列表时，去除新版本元素 → 去除重复、被删除的元素，并做Merge → 将WriteBatch的操作更新到列表上

RocksDB读流程

MVCC

- 当前事务的写入在提交前存储在自己的WriteBatch中，其他事务读取不到。
- RR隔离级别下，事务的读取在第一次执行读取操作时获取SequenceNumber。
- 读取过程中会过滤大于SequenceNumber的记录。

RocksDB读流程

MVCC

- 在等值查询和范围查询中略有不同。
- 等值查询中，当在Iterator中执行Seek操作时，由于Key的比较规则，SequenceNumber较大的项已被跳过，而在Next操作中，不会找到不等于目标Key的项；
- 而在范围查询中Sequence Number较大的项可能在Next操作中被取出来，因此需要再次进行过滤。

RocksDB读流程

MVCC

Select * from t where id >= 5;

给出一个Seek和Next过程中实现MVCC的例子，假设有列表(元素格式为Key：SequenceNumber)：[5:10, 5:9, 5:8, 6:10, 6:9, 6:8]，当前快照Sequence Number值为9.

Seek(5)

[5:10, **5:9**, 5:8, 6:10, 6:9, 6:8]

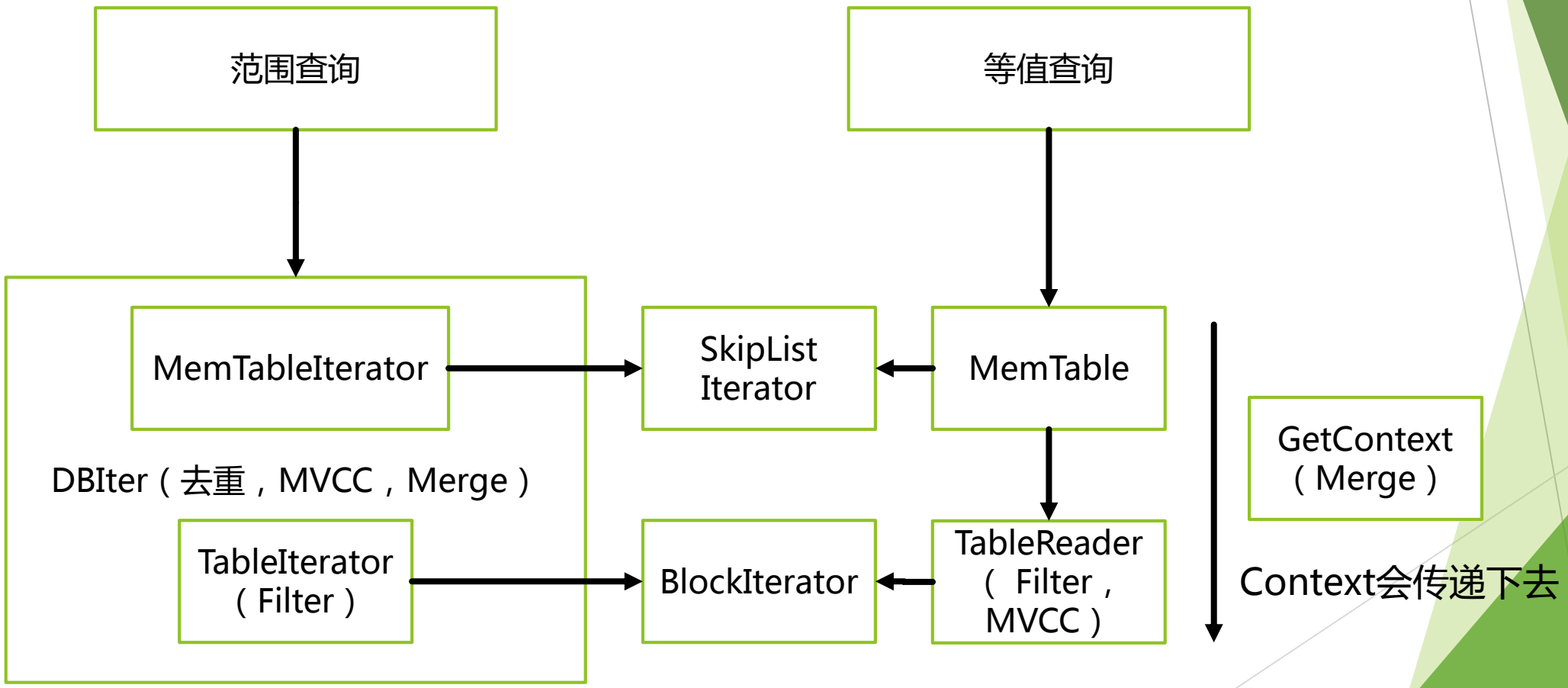
Next

1, 向前一步，得到[5:10, 5:9, **5:8**, 6:10, 6:9, 6:8]，发现重复，跳过。

2, 向前两步，得到[5:10, 5:9, 5:8, **6:10**, 6:9, 6:8]，Sequence Number不符合跳过。

3, 向前三步，得到[5:10, 5:9, 5:8, 6:10, **6:9**, 6:8]，符合，Next结束。

RocksDB读流程


范围查询
等值查询
MemTableIterator
SkipList Iterator
MemTable
GetContext（Merge）
DBIter（去重，MVCC，Merge）
TableIterator（Filter）
BlockIterator
TableReader（Filter，MVCC）
Context会传递下去

Handler层

信息：

- 记录行数
- 索引数据大小
- 数据大小
- 每个Key的记录数
- 每个Key Prefix的不同Key数
- 在Flush，Compaction和手动执行ANALYZE TABLE时更新

Handler层

索引选择：

利用之前的数据并通过较为简单的查询，records_in_range实现了对于两个Key之间的记录数的估算。主要的估算的公式为：

ret = rows*sz/disk_size

其中rows和disk_size已经在上面一阶段求出，sz表示两个Key之间的数据所占用的disk_size，sz的估算逻辑最终落到BlockBasedTable::ApproximateOffsetOf通过IndexIterator的一次Seek获得key所在Block的Offset.

当这个代价计算相等的情况下，默认先选择Primary Key.

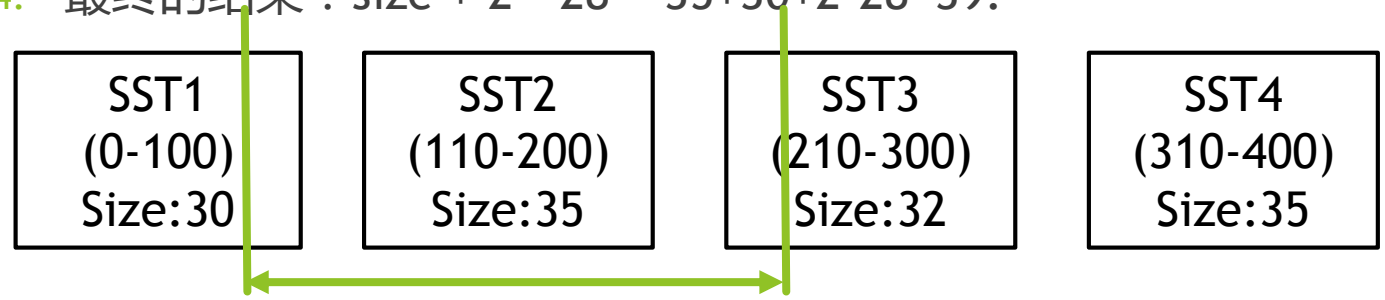
但代价相等且满足索引读（covering index）的情况下选择Secondary Key.

Handler层

索引选择：

假设有4个SST文件，我们要找的范围是99-211。

1. 定位99所在的文件根据文件的minKey和maxKey元数据，定位到时SST1，之后通过SST1的Index搜索99所在的块（Seek），找到的Offset为28.
2. 设置size=0.
3. 遍历SST1（由99定位得到）以及之后的文件的元数据，如果211不在范围，size加上文件大小，如果211在范围内，则定位211所在块，假设是在SST3的Offset为2的位置。
4. 最终的结果：size + 2 – 28 = 35+30+2-28=39.



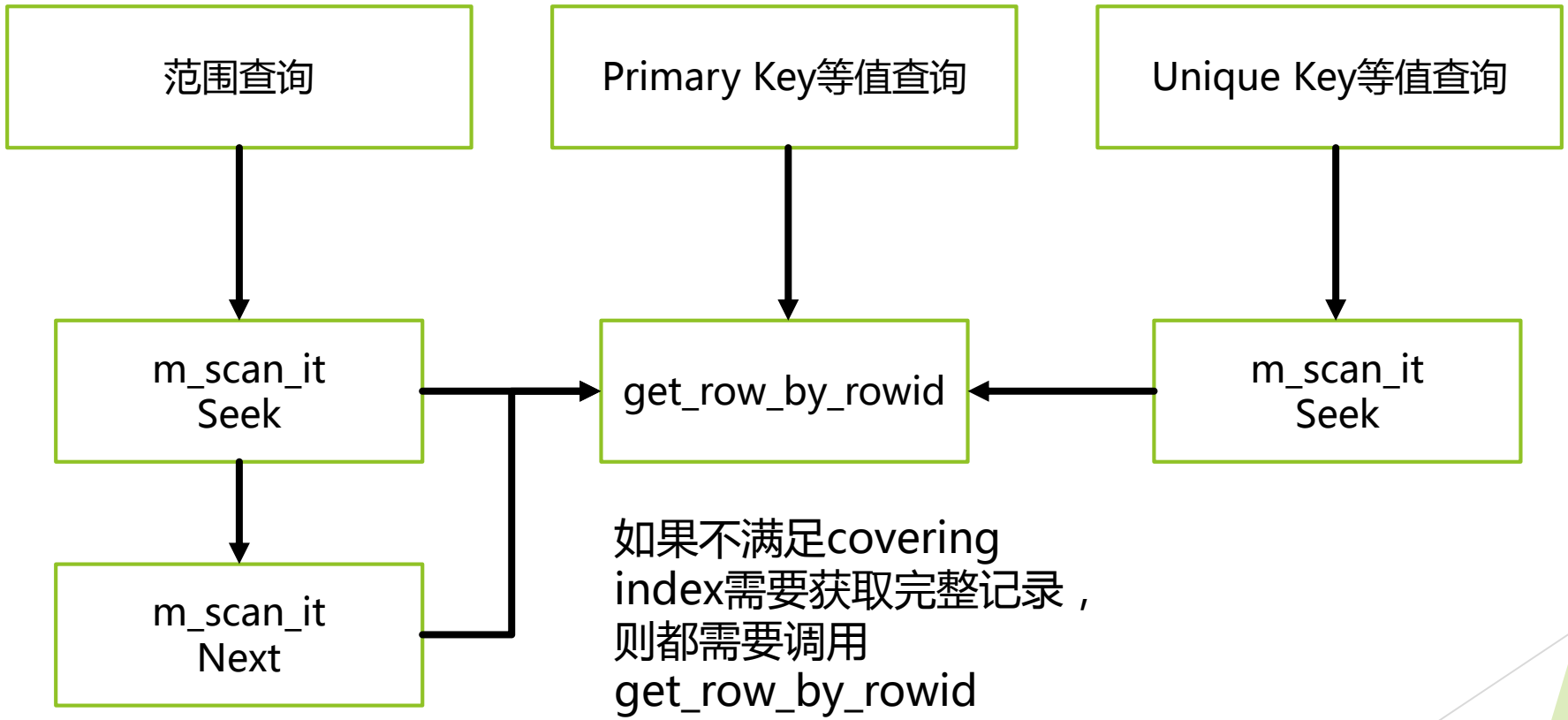
Handler层

- 最大User Key为要查找Key所属的IndexNumber加一，当执行select * from t order by id desc时，需要定位最大的id，这时Seek使用的Key就是这个。
- 最小User Key为要查找的IndexNumber，当执行select * from t时，需要定位最小的id，这时Seek使用的Key就是这个。
- 下一个InternalKey，计算方式为IndexNumber + Key + 0（SequenceNumber 7bytes） + 0（Type Delete 1 byte）.根据之前的排序方式，这个InternalKey在Key相等的情况下最大而且一定不存在，因此可以定位下一个Key。

Handler层

- 从handler层来看共有两条路径可以使用
  - 使用Key可以直接获得数据，则调用index_read_map或get_row_by_rowid，进入rocksdb后，调用DBImpl::Get获取一条记录，这条路径也会通过TableIterator的Seek函数找到目标行。
  - 需要遍历获得数据，则调用m_scan_it的Seek和Next，进入rocksdb后使用刚才描述的BaseDeltaIterator进行遍历，如果使用的Iterator是在Secondary Key上进行遍历，则会从Secondary Key上解析出Primary Key，再使用Primary Key进入等值的读取流程。

Handler层


范围查询
Primary Key等值查询
Unique Key等值查询
m_scan_it
Seek
get_row_by_rowid
m_scan_it
Seek
m_scan_it
Next
如果不满足covering index需要获取完整记录，则都需要调用get_row_by_rowid

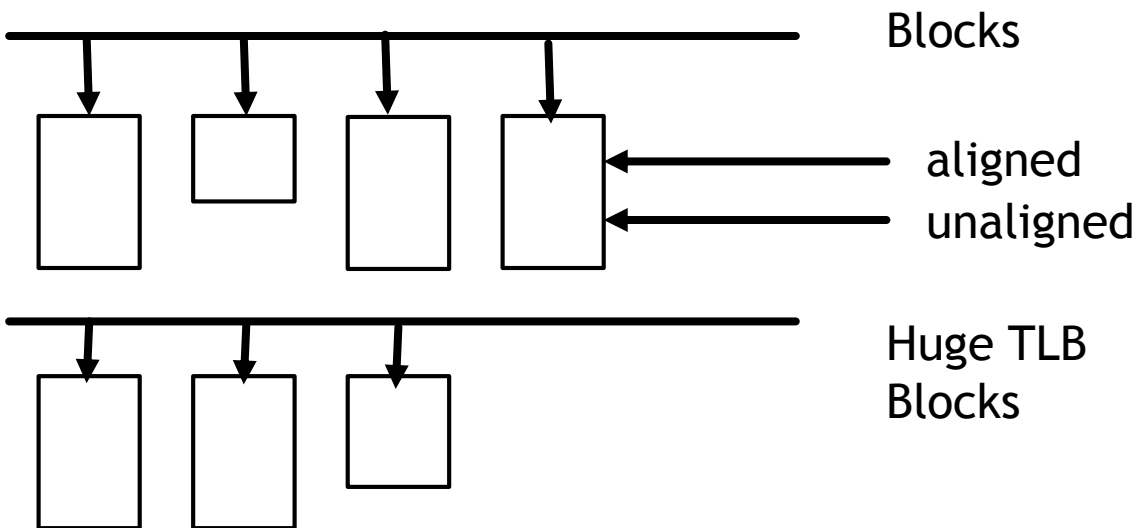
其他部分

- Arena
- Block Cache和Table Cache
- WriteBatchWithIndex
- SuperVersion

其他部分

Arena

- 在Iterator层次创建的过程中为Iterator分配空间。
- 大于四分之一块大小时分配一个单独的块，单独使用。



其他部分

Block Cache和Table Cache

- 缓存默认使用ShardedLRUCache实现
- Block Cache→使用文件编号和Offset作为Key查找Block
- Table Cache→使用文件名查找TableReader

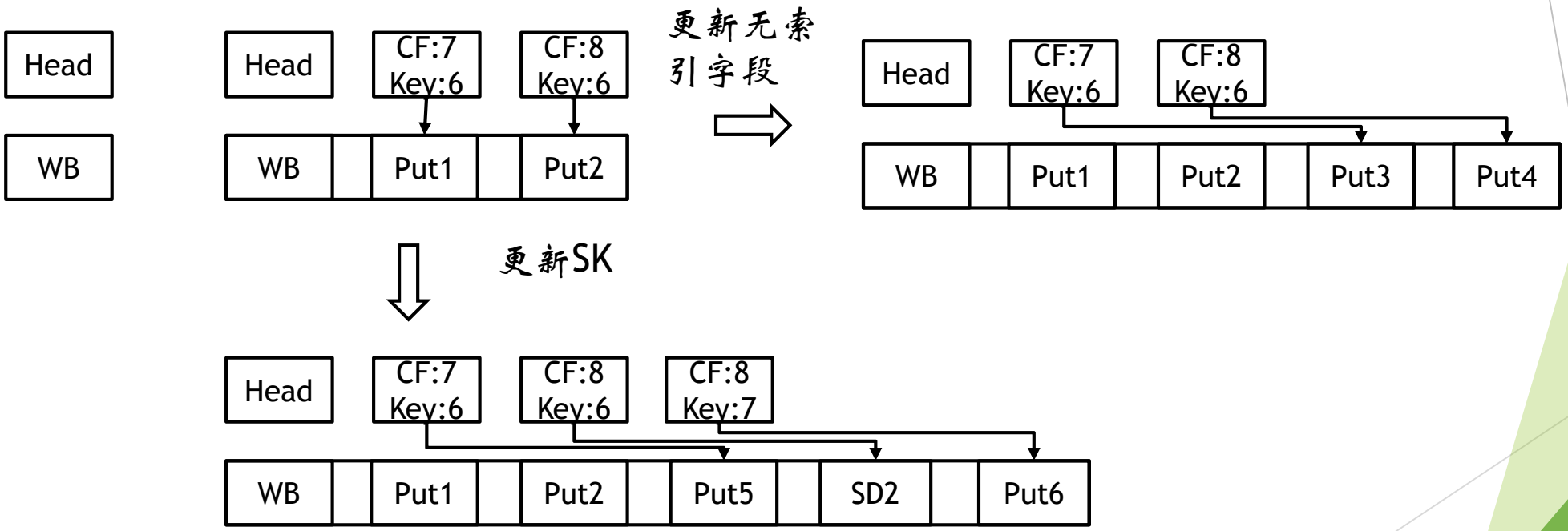
其他部分

WriteBatchWithIndex

- WriteBatch，主体是一个字符串，可以解析出其中的操作
- Index，是一个SkipList，索引WriteBatch中的内容

RocksDB读流程

WriteBatchWithIndex



其他部分

SuperVersion

- SuperVersion引用了当前Iterator所需搜索的内容，包括一个应用SST文件的Version，以及MemTable，Immutable MemTable
- 在Iterator遍历数据库时，SuperVersion保证了遍历所需的SST文件，MemTable不会被删除。

MyRocks读过程分析

谢谢大家！