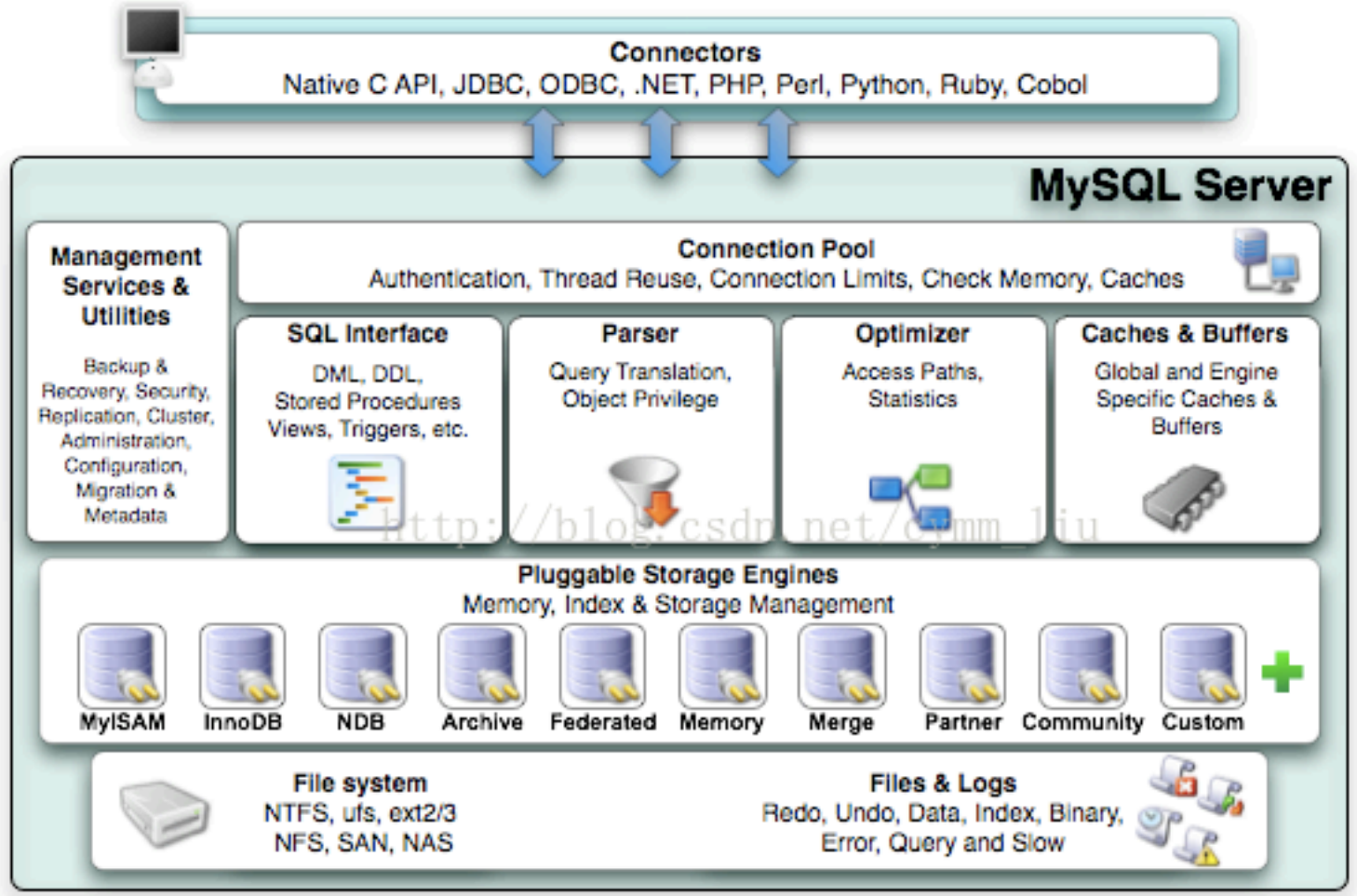
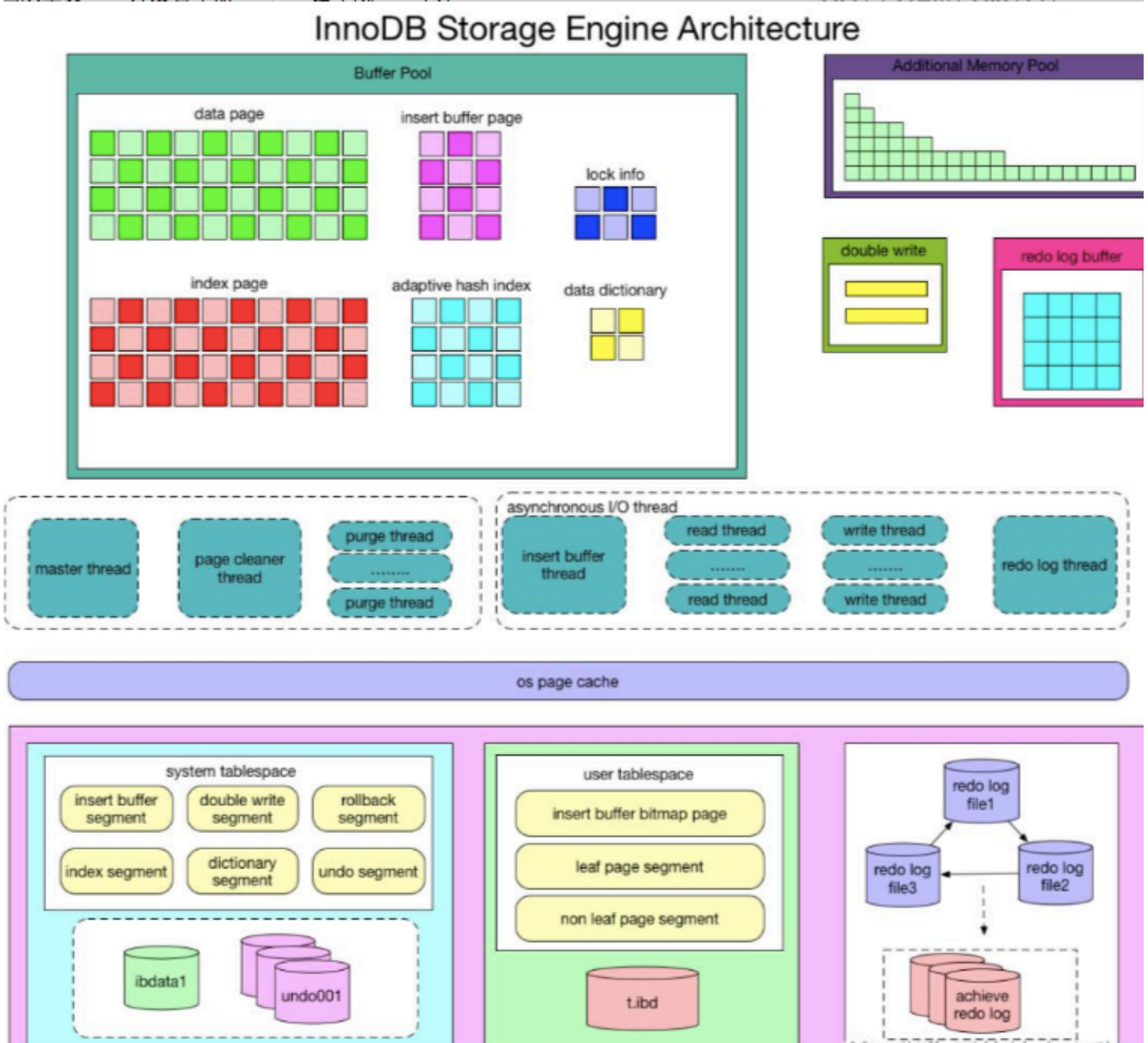
innodb

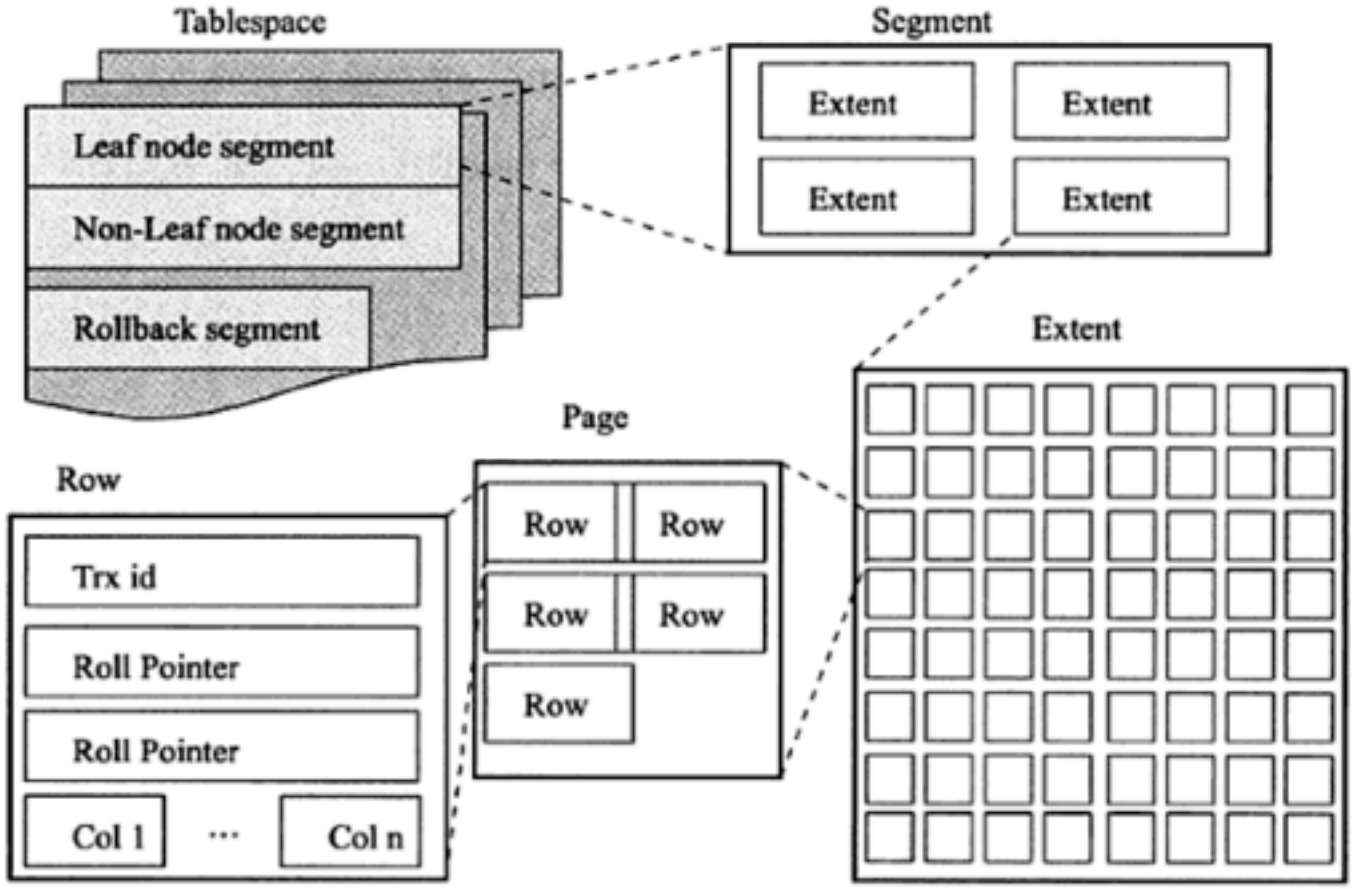
Mysql架构


Connectors
Native C API, JDBC, ODBC, .NET, PHP, Perl, Python, Ruby, Cobol
MySQL Server
Management Services & Utilities
Backup & Recovery, Security, Replication, Cluster, Administration, Configuration, Migration & Metadata
Connection Pool
Authentication, Thread Reuse, Connection Limits, Check Memory, Caches
SQL Interface
DML, DDL, Stored Procedures Views, Triggers, etc.
Parser
Query Translation, Object Privilege
Optimizer
Access Paths, Statistics
Caches & Buffers
Global and Engine Specific Caches & Buffers
Pluggable Storage Engines
Memory, Index & Storage Management
MyISAM
InnoDB
NDB
Archive
Federated
Memory
Merge
Partner
Community
Custom
File system
NTFS, ufs, ext2/3 NFS, SAN, NAS
Files & Logs
Redo, Undo, Data, Index, Binary, Error, Query and Slow

Innodb

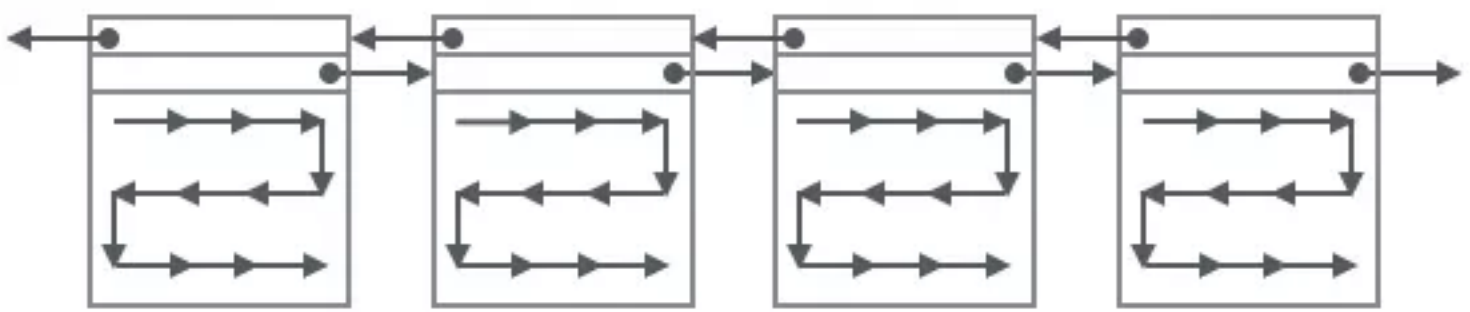

InnoDB Storage Engine Architecture
Buffer Pool
data page
insert buffer page
lock info
index page
adaptive hash index
data dictionary
Additional Memory Pool
double write
redo log buffer
master thread
page cleaner thread
purge thread
.....
purge thread
asynchronous I/O thread
insert buffer thread
read thread
.....
read thread
write thread
.....
write thread
redo log thread
os page cache
system tablespace
insert buffer segment
double write segment
rollback segment
index segment
dictionary segment
undo segment
ibdata1
undo001
user tablespace
insert buffer bitmap page
leaf page segment
non leaf page segment
t.ibd
redo log file1
redo log file3
redo log file2
achieve redo log

逻辑存储结构


Tablespace
Leaf node segment
Non-Leaf node segment
Rollback segment
Segment
Extent
Extent
Extent
Extent
Extent
Page
Row
Row
Row
Row
Row
Row
Trx id
Roll Pointer
Roll Pointer
Col 1
...
Col n

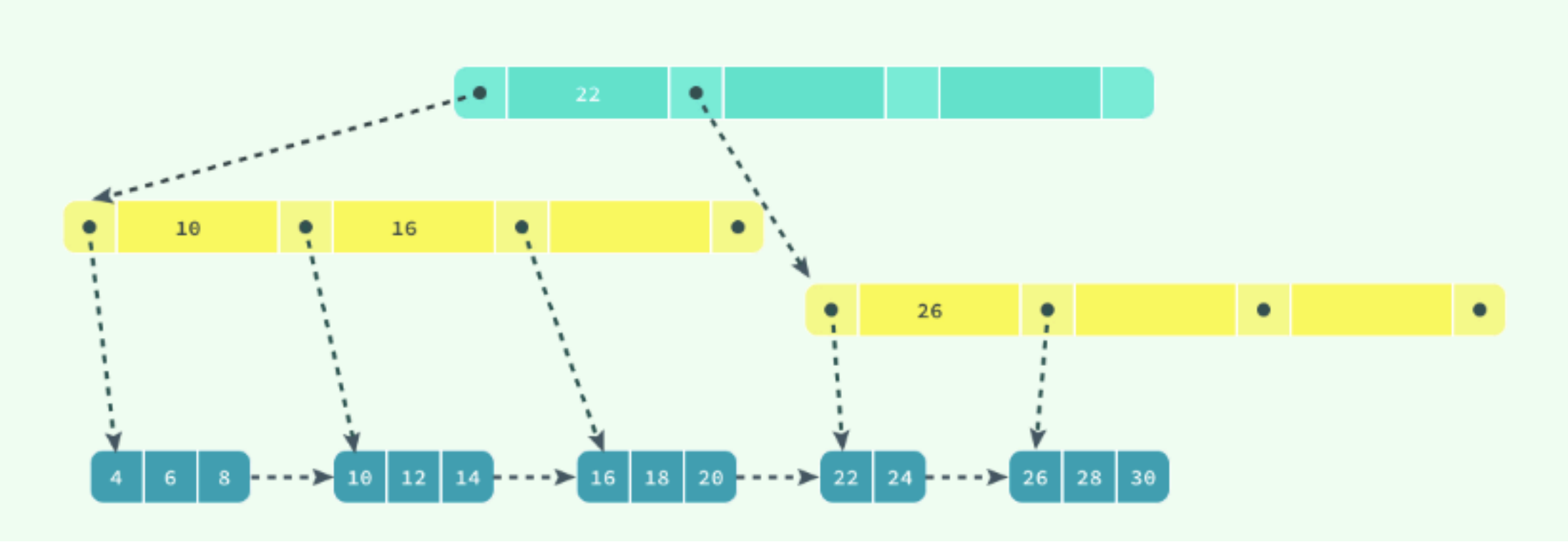
记录和页

- 页之间双向链表，页内记录单向链表

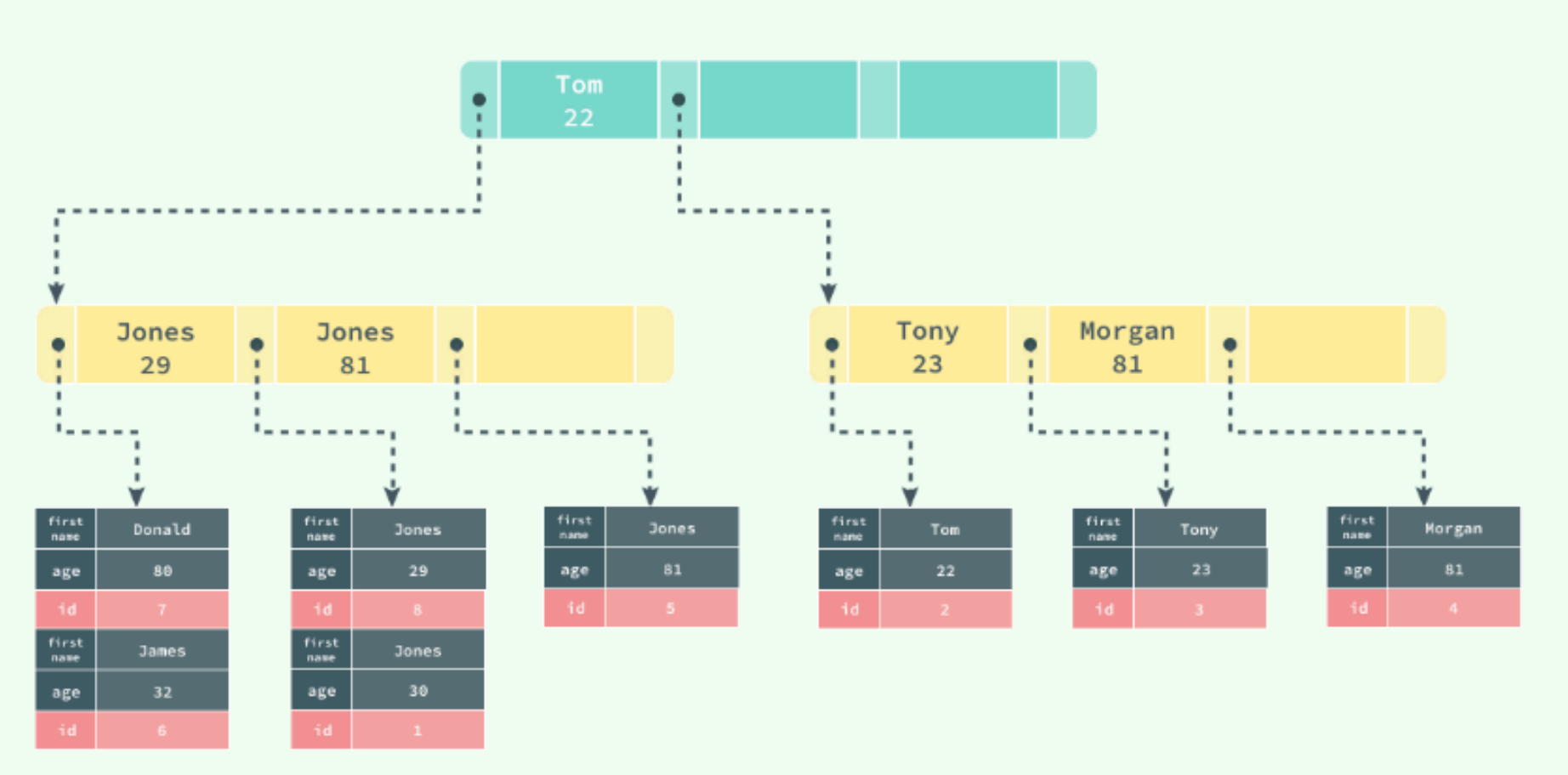


双向链表和单向链表

索引-聚簇索引



索引-辅助索引


Tom
22
Jones
29
Jones
81
Tony
23
Morgan
81
first name
Donald
age
80
id
7
first name
James
age
32
id
6
first name
Jones
age
29
id
8
first name
Jones
age
30
id
1
first name
Jones
age
81
id
5
first name
Tom
age
22
id
2
first name
Tony
age
23
id
3
first name
Morgan
age
81
id
4

锁

排他锁：update、delete、insert、select for update
共享锁：select lock in share mode

锁的算法

- Record Lock
- Gap Lock
- Next-Key Lock

锁的算法

- 等值查询M，非唯一索引的加锁逻辑 (M->pre-rec, M]，(M, M->next-rec]
- 等值查询M，唯一键的加锁逻辑 [M], next-lock 降级为 record locks
- >=，加锁逻辑 (M->pre_rec, M]，(M, M->next-rec]....(∞]
- >，加锁逻辑 (M, M->next-rec] ... (∞]
- <=，加锁逻辑 (-∞] ... (M, M->next-rec]
- <，加锁逻辑 (-∞] ... (M->rec, M]

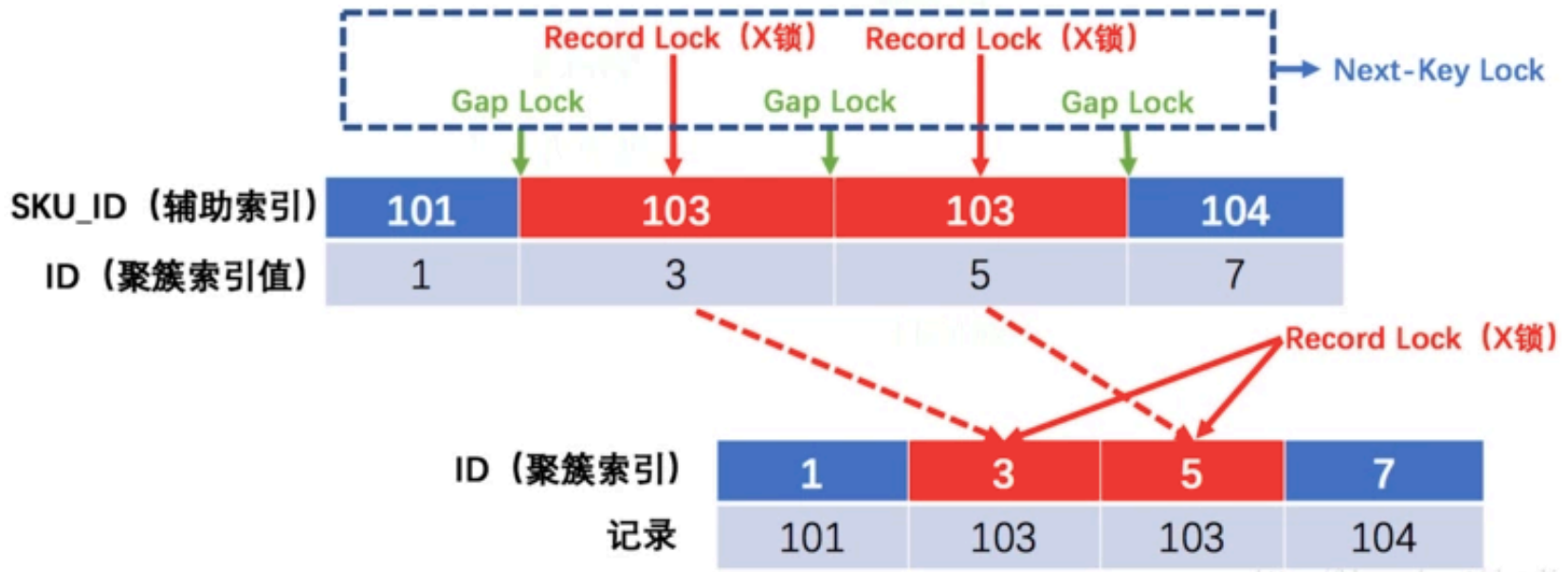
锁的算法

3、等值查询使用辅助索引

行锁算法示例——等值查询使用辅助索引

TABLE: stock(id primary key, sku_id key)

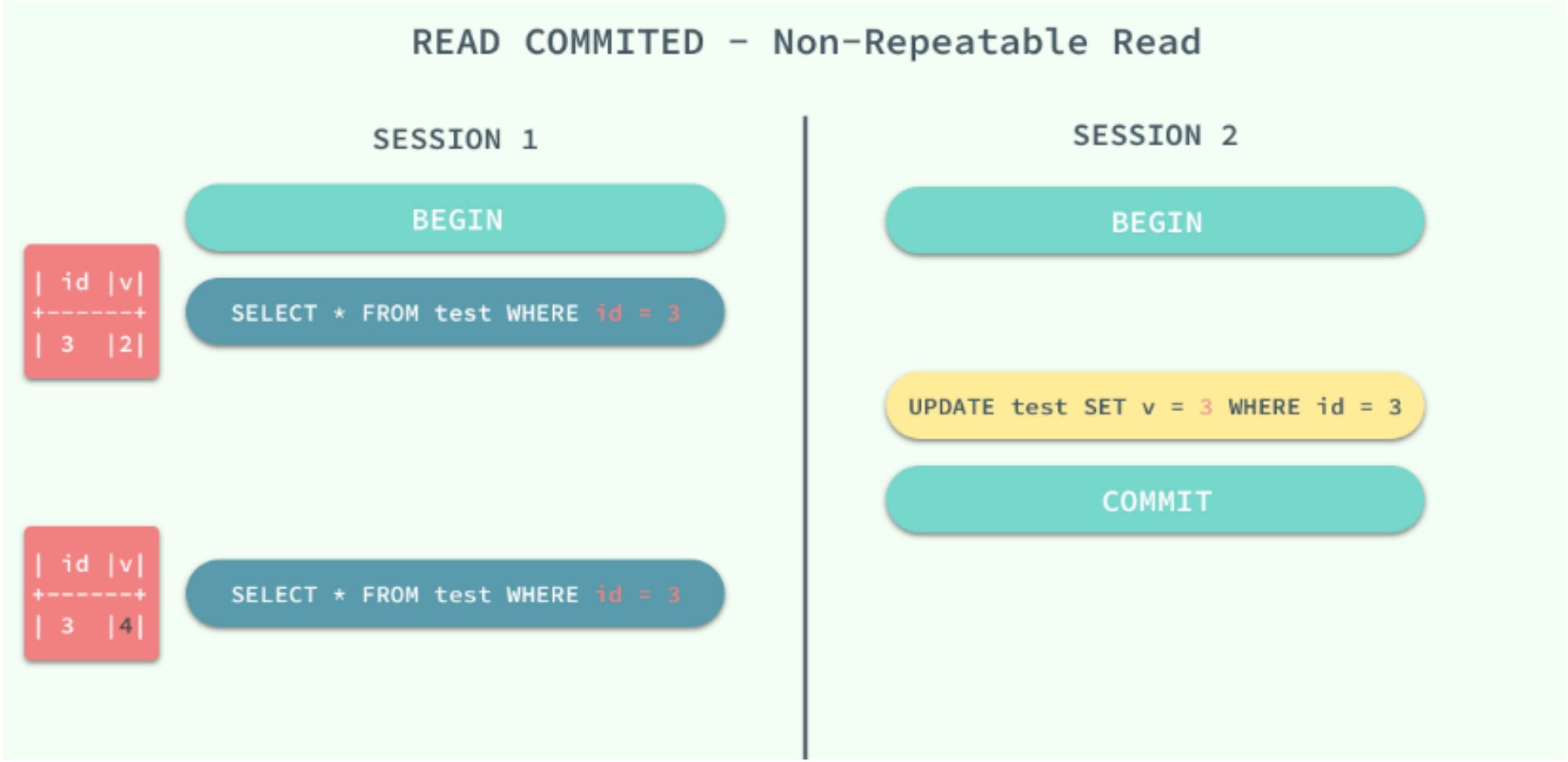
SQL: select * from stock where sku_id = 1000003 for update;



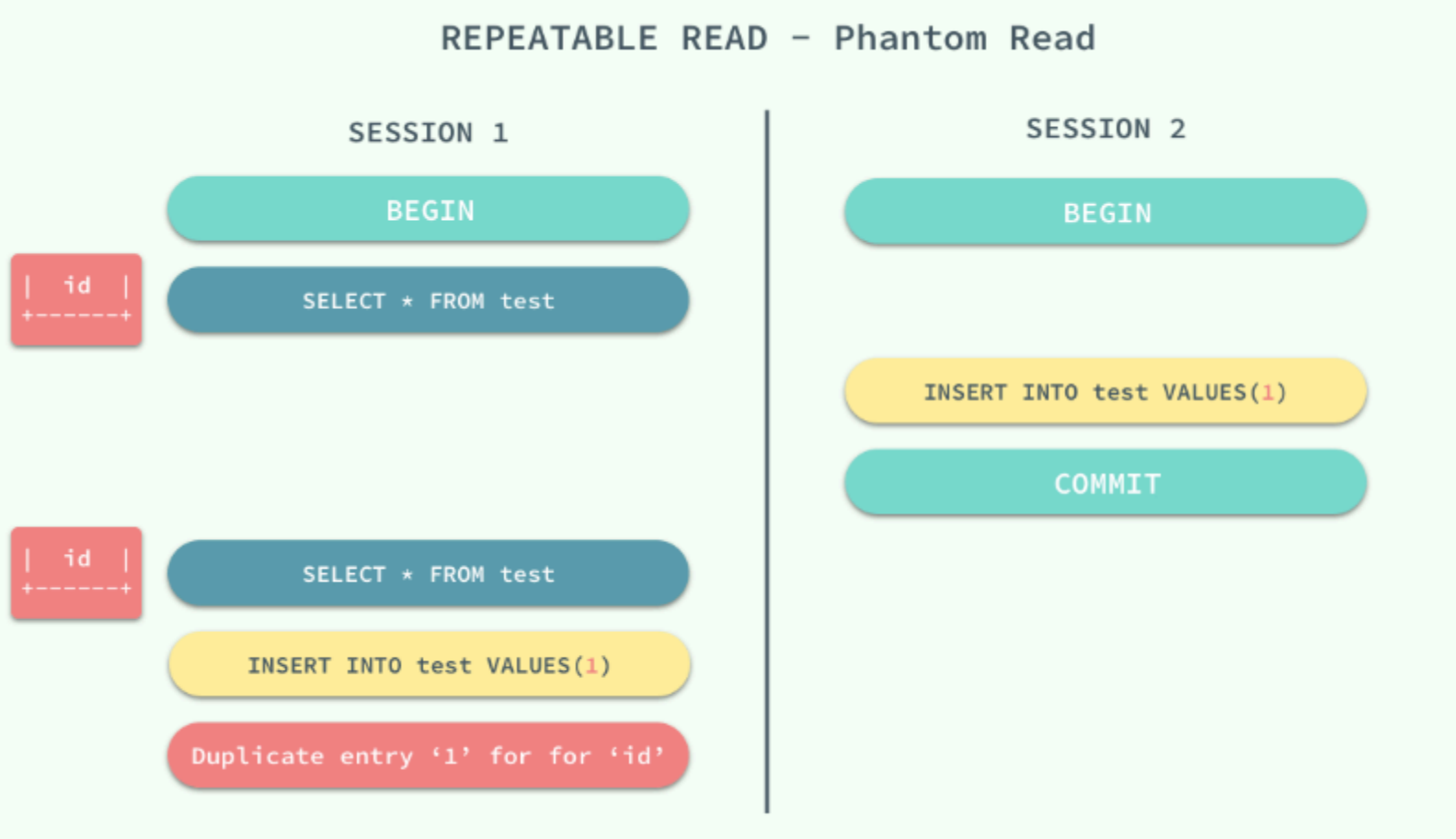
事务隔离-读未提交


READ UNCOMMITED - Dirty Read
SESSION 1
SESSION 2
BEGIN
BEGIN
| id |
+------+
SELECT * FROM test
INSERT INTO test VALUES(1)
| id |
+------+
| 1 |
SELECT * FROM test

事务隔离-读已提交


READ COMMITED - Non-Repeatable Read
SESSION 1
SESSION 2
BEGIN
BEGIN
| id |v|
+----+
| 3 |2|
SELECT * FROM test WHERE id = 3
UPDATE test SET v = 3 WHERE id = 3
COMMIT
| id |v|
+----+
| 3 |4|
SELECT * FROM test WHERE id = 3

事务隔离-重复读


REPEATABLE READ - Phantom Read
SESSION 1
SESSION 2
BEGIN
BEGIN
| id |
+------+
SELECT * FROM test
INSERT INTO test VALUES(1)
COMMIT
| id |
+------+
SELECT * FROM test
INSERT INTO test VALUES(1)
Duplicate entry '1' for for 'id'

缓冲池

- 读取页的时候，先判断是否在缓冲池，不在的时候从磁盘读取，然后放入缓冲池
- 修改也是修改缓冲池中的页（脏页），会根据checkpoint机制，刷新到磁盘
- LRU（最近最少使用）管理

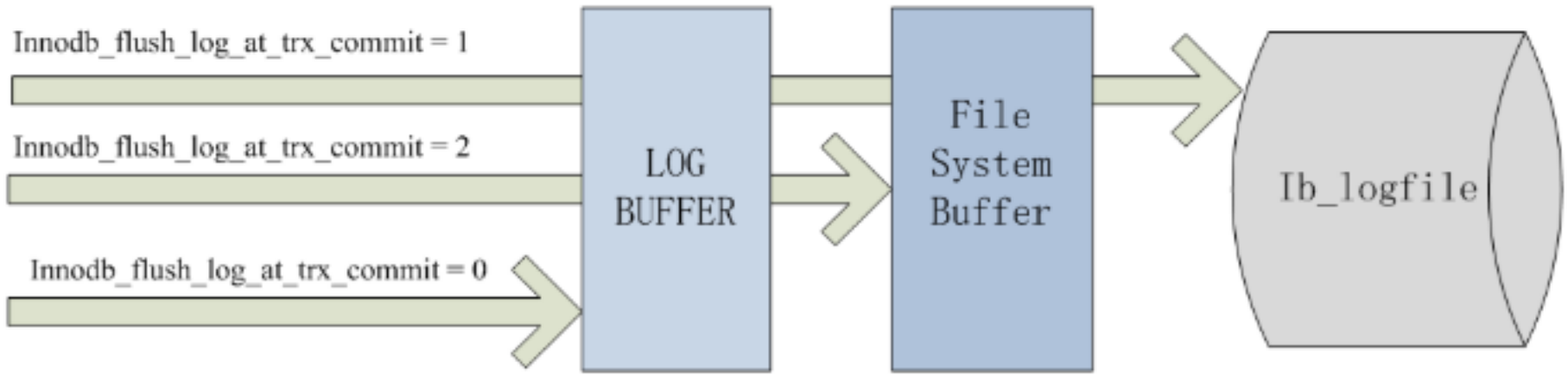
Redo log

- 用于数据恢复，遵循事务的持久性要求
- 页面修改，先写重写日志，再修改页（write ahead log策略），日志一定比数据页先写回磁盘
- 记录每个页的更改的物理情况，如偏移量800，写”aaa”记录
- 由于把对磁盘的随机写入(写数据)转换成了顺序的写入(写redo日志)，性能有很大幅度的提高

Redo log刷新时间

- Master thread每秒将重做日志缓冲刷新到重做日志文件
- 每个事务提交
- 重做缓冲池剩余空间小于1/2

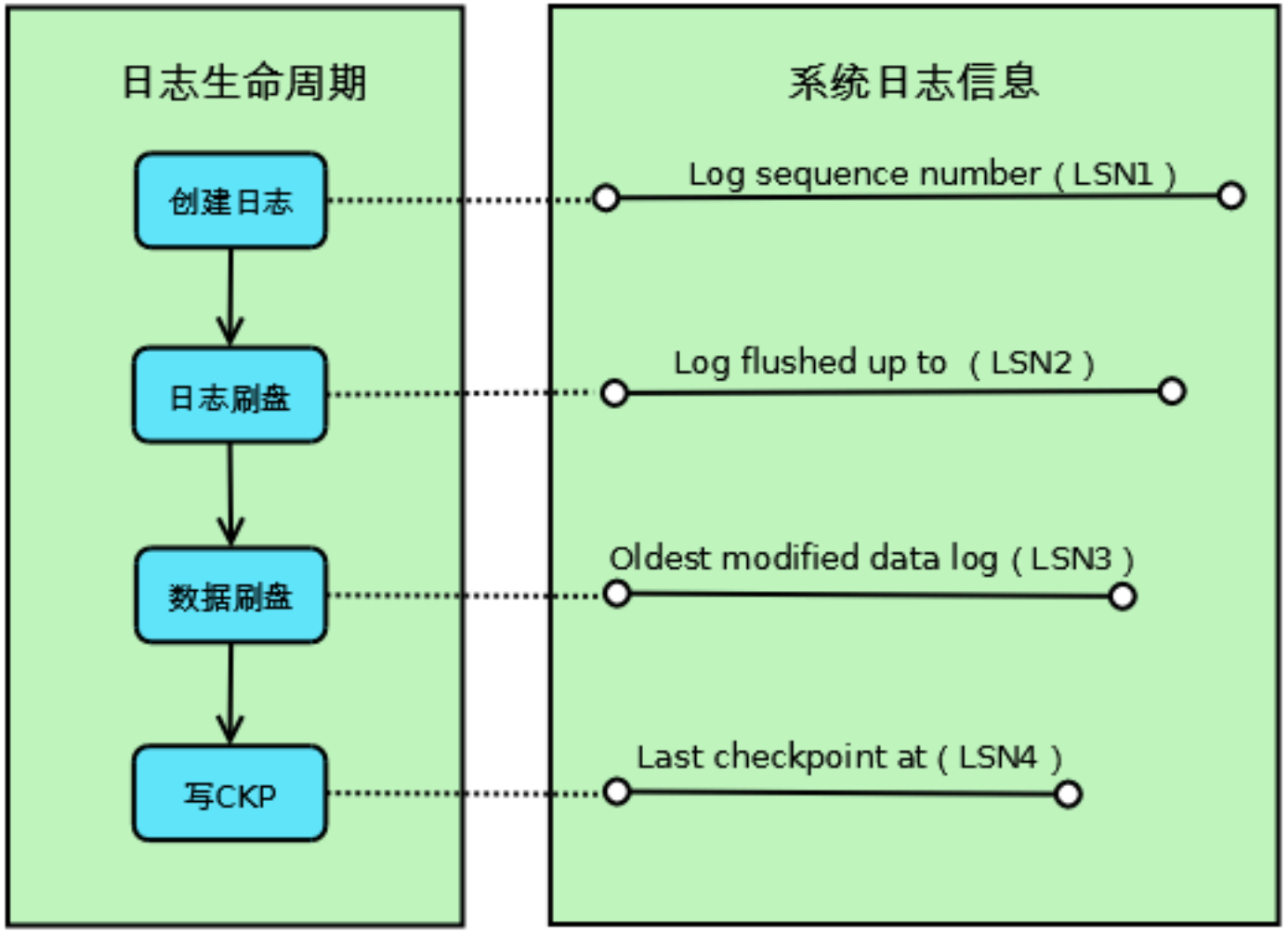
Redo log


Innodb_flush_log_at_trx_commit = 1
Innodb_flush_log_at_trx_commit = 2
Innodb_flush_log_at_trx_commit = 0
LOG BUFFER
File System Buffer
Ib_logfile

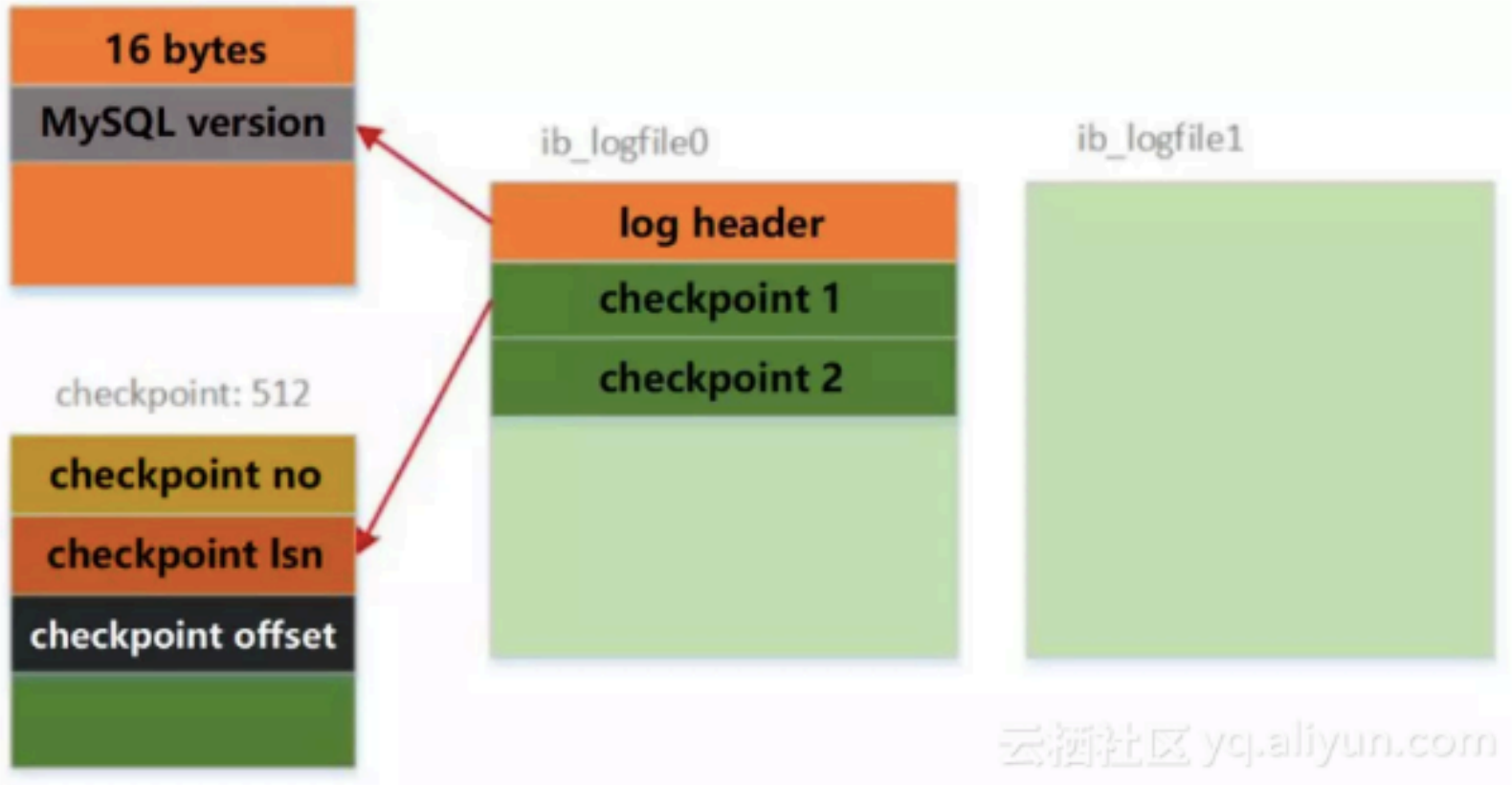
Checkpoint

- 检查点，表示脏页写入到磁盘的时候，所以检查点也就意味着脏数据的写入。
- 目的
  - 缩短数据库的恢复时间
  - 缓冲池不够用时，将脏页刷新到磁盘
  - 重做日志不可用时，刷新脏页
- 分类
  - sharp checkpoint：数据库正常关闭时
  - fuzzy checkpoint：master thread checkpoint、flush_lru_list checkpoint、async/sync flush checkpoint、dirty page too much checkpoint

LSN（Log Sequence Number）



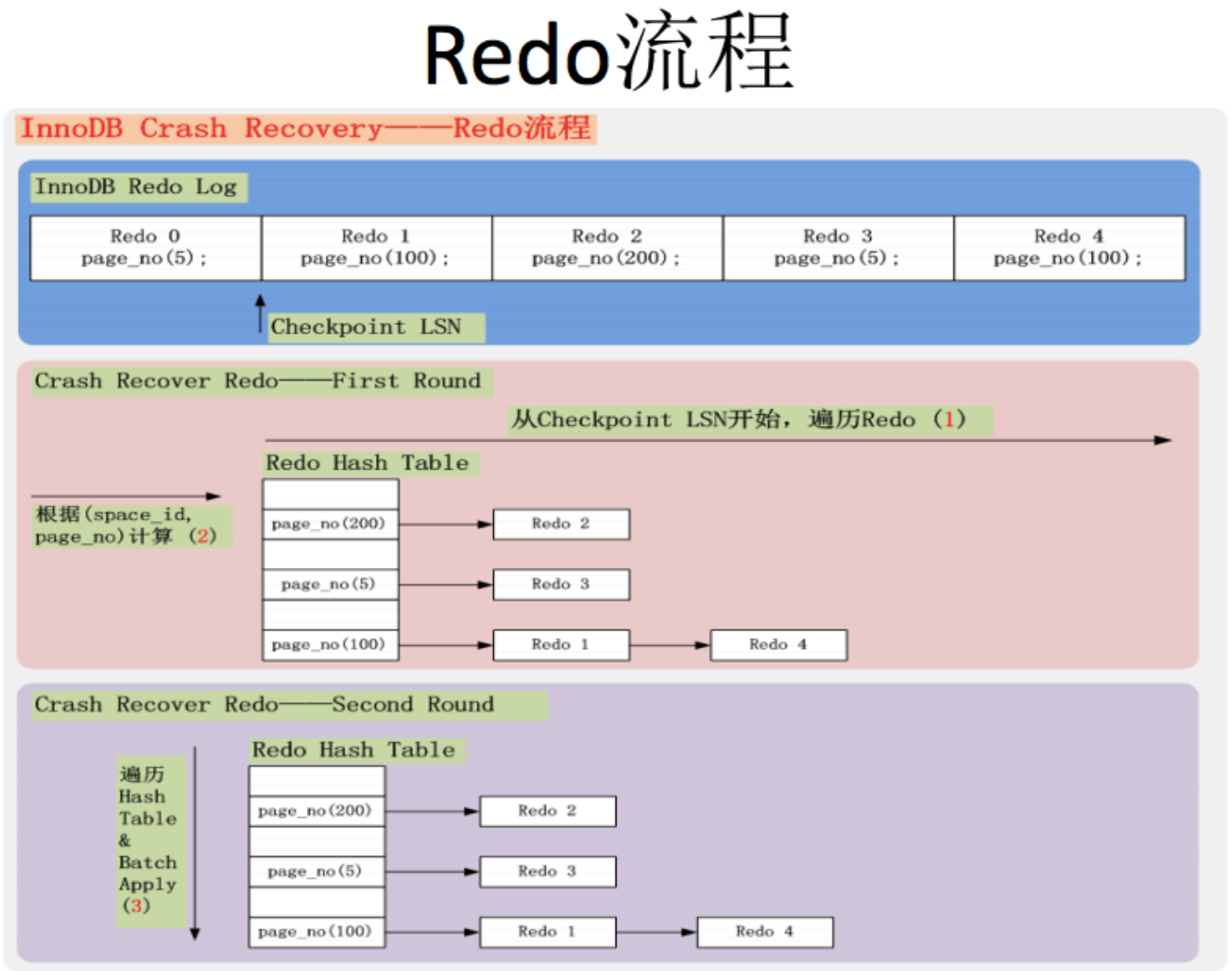
Checkpoint


16 bytes
MySQL version
ib_logfile0
log header
checkpoint 1
checkpoint 2
ib_logfile1
checkpoint: 512
checkpoint no
checkpoint lsn
checkpoint offset
云栖社区 yq.aliyun.com

Checkpoint

- checkpoint no：主要保存的是checkpoint号，因为InnoDB有两个checkpoint，通过checkpoint号来判断哪个checkpoint更新
- checkpoint lsn：数据刷盘的LSN，确保在该LSN前面的数据页都已经落盘，不再需要通过redo log进行恢复
- checkpoint offset：主要记录了该checkpoint产生时，redo log在ib_logfile中的偏移量，通过该offset位置就可以找到需要恢复的redo log开始位置。

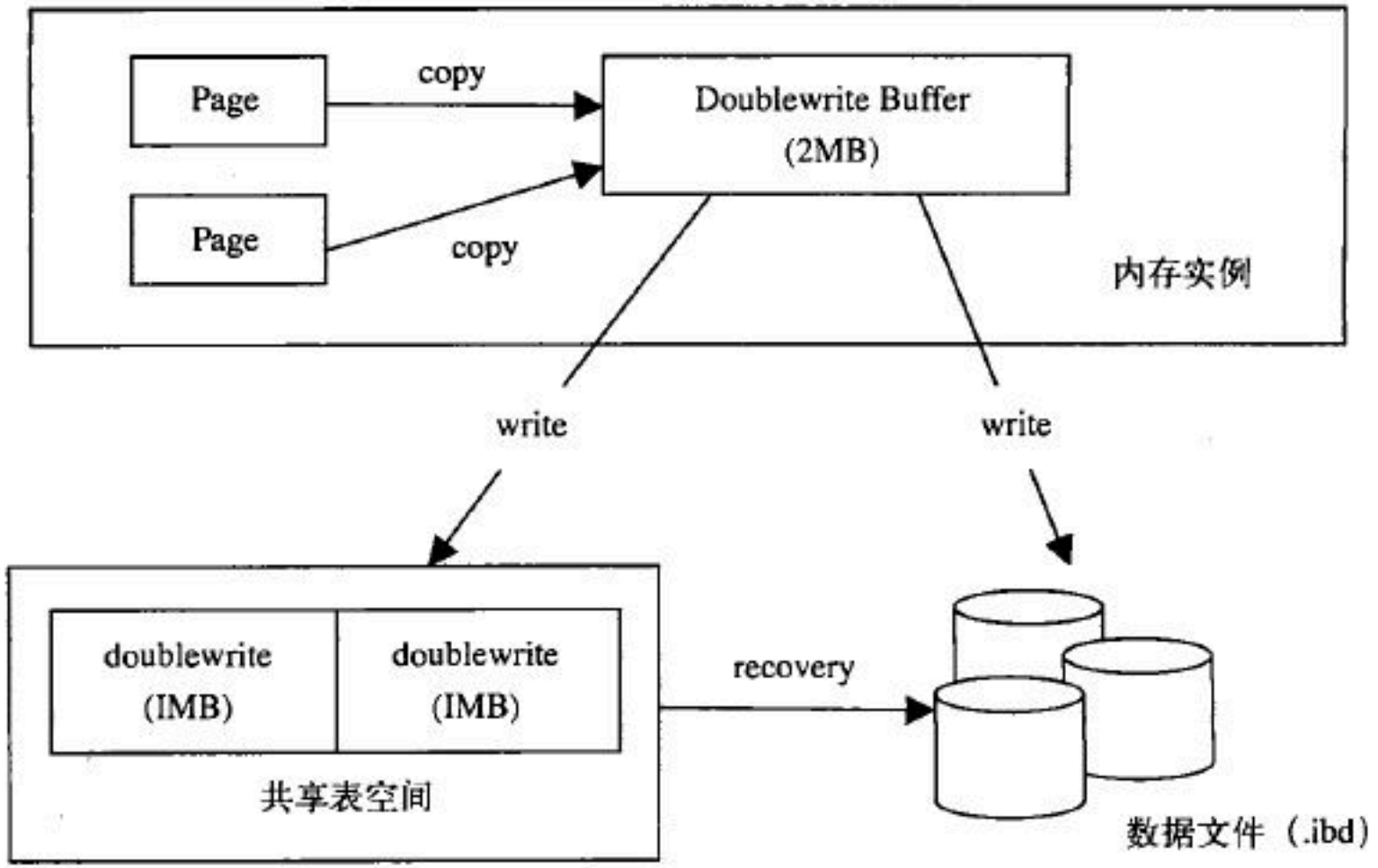
Cash recovery



两次写

- InnoDB 的Page Size一般是16KB，将数据写入到磁盘是以Page为单位进行操作的。16K的数据，写入4K 时，发生了系统断电/os crash ，只有一部分写是成功的，这种情况下就是 partial page write 问题。

两次写


Page
copy
Doublewrite Buffer
(2MB)
Page
copy
内存实例
write
write
doublewrite
(IMB)
doublewrite
(IMB)
共享表空间
recovery
数据文件（.ibd）

插入缓冲

- 性能提升
- Master Thread 每秒和每10秒的合并操作。
- 只对于非聚集索引（非唯一）的插入和更新有效，对于每一次的插入不是写到索引页中，而是先判断插入的非聚集索引页是否在缓冲池中，如果在则直接插入；若不在，则先放到Insert Buffer中，再按照一定的频率进行合并操作。这样通常能将多个插入合并到一个操作中，提升插入性能

Undo log

- Undo log可以用来做事务的回滚操作，保证事务的原子性。同时可以用来构建数据修改之前的版本，支持多版本读。
- 一个事务最多可以有两个undo slot
  - insert undo slot, 用来存储这个事务的insert undo，里面主要记录了主键的信息，方便在回滚的时候快速找到这一行
  - update undo slot，用来存储这个事务delete/update产生的undo，里面详细记录了被修改之前每一列的信息，便于在读请求需要的时候构造。
- undo日志在Buffer Pool中有对应的数据页，与普通的数据页一起管理，依据LRU规则也会被淘汰出内存，后续再从磁盘读取。与普通的数据页一样，对undo页的修改，也需要先写redo日志。

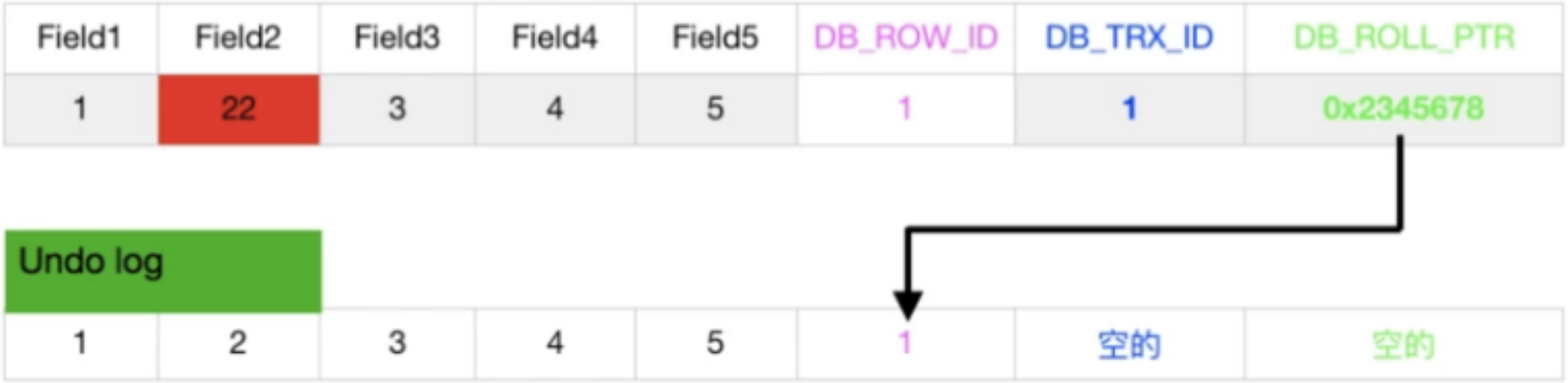
Undo log

- insert undo log : 事务对insert新记录时产生的undo log, 只在事务回滚时需要, 并且在事务提交后就可以立即丢弃
- update undo log : 事务对记录进行delete和update操作时产生的undo log, 不仅在事务回滚时需要, 一致性读也需要，所以不能随便删除，只有当数据库所使用的快照中不涉及该日志记录，对应的回滚日志才会被purge线程删除

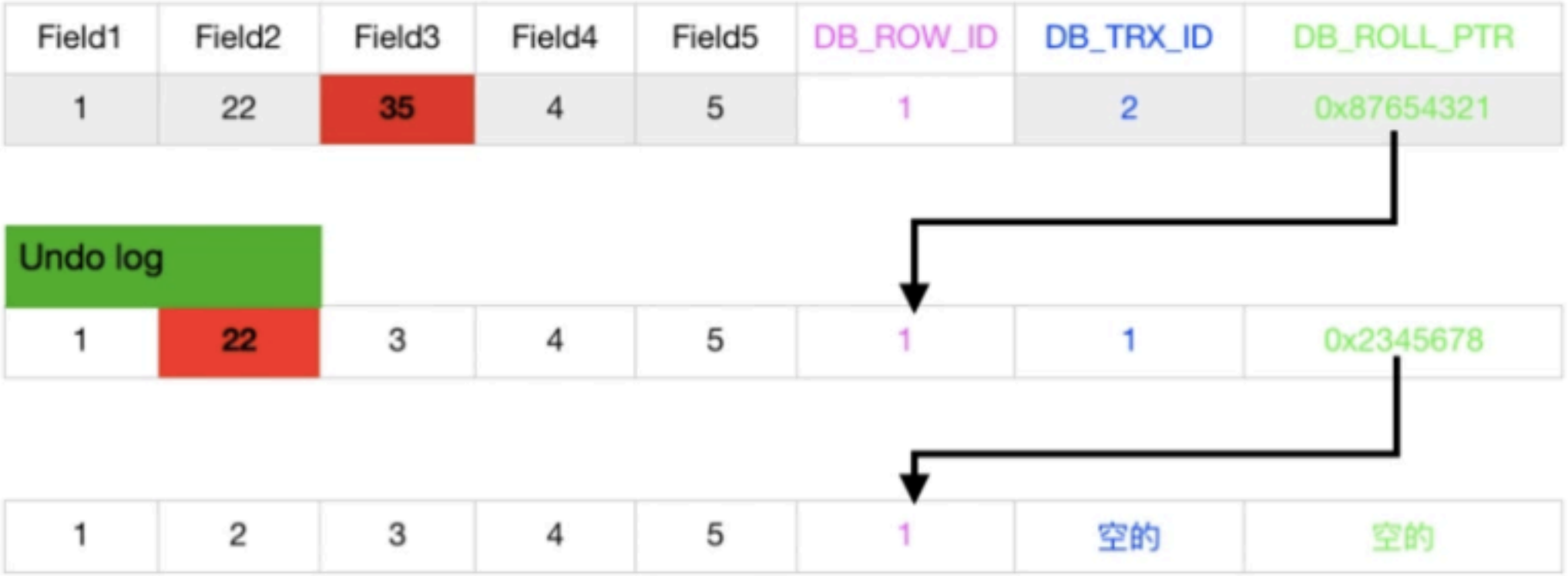
多版本控制mvcc

- 指的是一种提高并发的技术。最早的数据库系统，只有读读之间可以并发，读写，写读，写写都要阻塞。引入多版本之后，只有写写之间相互阻塞，其他三种操作都可以并行，这样大幅度提高了InnoDB的并发度。
- 事务开启第一个select会生成快照read view，保存当前事务的信息
  - 查看当前所有的未提交并活跃的事务，存储在快照的数组中descriptors
  - 选取未提交并活跃的事务中最小的事务ID，记录在快照的up_limit_id
  - 选取当前最大事务id+1，记录在low_limit_id中
- 遍历undo log找到满足可见性的记录，事务id< up_limit_id可见，事务id > low_limit_id不可见

多版本控制mvcc


Field1
Field2
Field3
Field4
Field5
DB_ROW_ID
DB_TRX_ID
DB_ROLL_PTR
1
22
3
4
5
1
1
0x2345678
Undo log
1
2
3
4
5
1
空的
空的

多版本控制mvcc


Field1
Field2
Field3
Field4
Field5
DB_ROW_ID
DB_TRX_ID
DB_ROLL_PTR
1
22
35
4
5
1
2
0x87654321
Undo log
1
22
3
4
5
1
1
0x2345678
1
2
3
4
5
1
空的
空的

purge

- 在执行delete或update操作时，实际旧记录没有真正删除（可能对活跃事务仍旧可见），只是在记录上打了一个删除标记，而是在事务提交后，由purge线程真正删除物理记录
- 在事务提交的时候，会把update undo加入到一个全局链表(history list)中，链表按照事务提交的顺序排序，保证最先提交的事务的update undo在前面，这样Purge线程就可以从最老的事务开始做清理

binlog

- 记录数据库执行更改的所有sql语句
- Mysql server记录
- 事务开始，记录更改操作到二进制日志缓冲中（默认32K），事务提交时写入二进制文件（默认1G）
- 主从同步