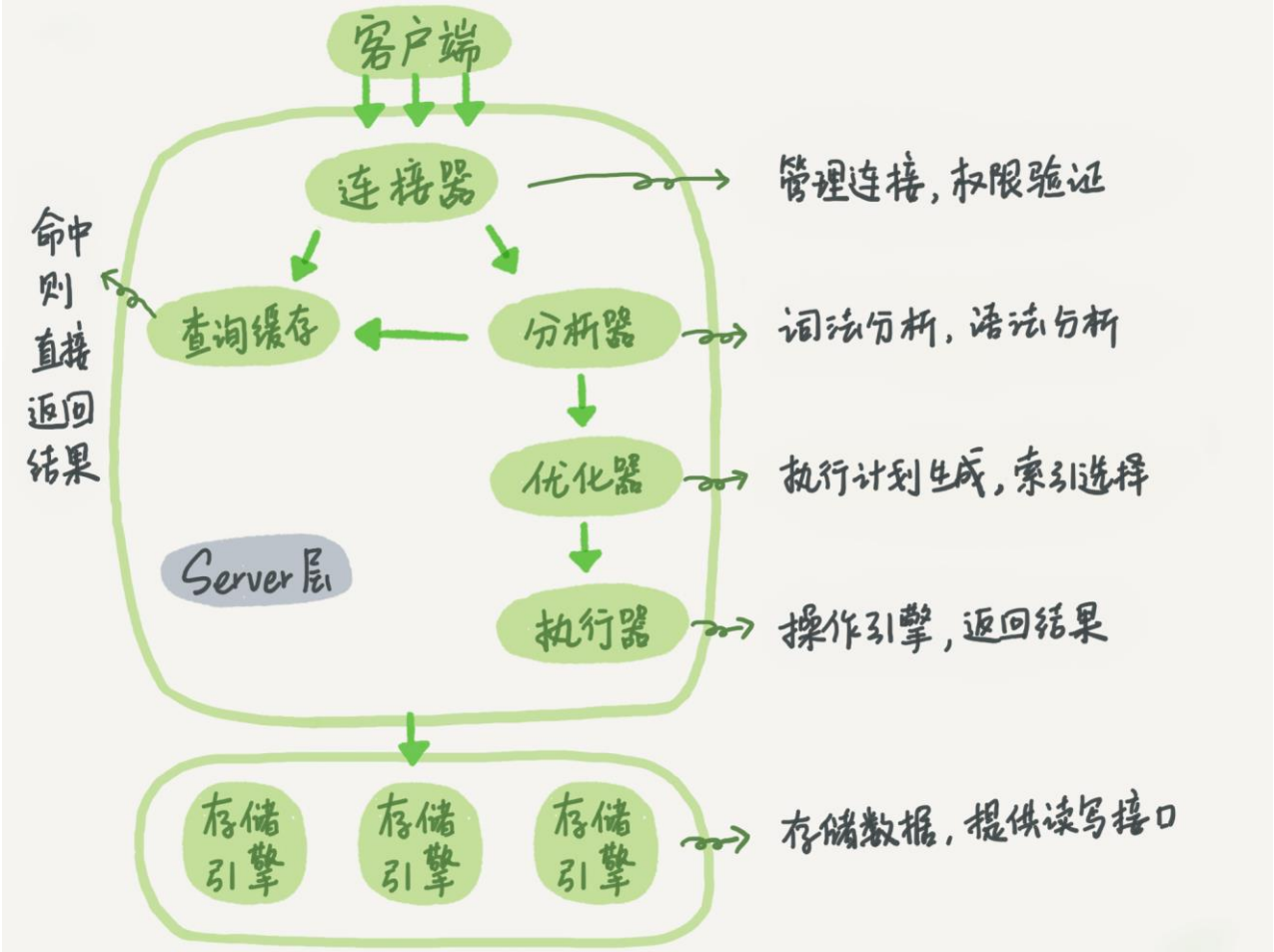
mysql 数据库分享

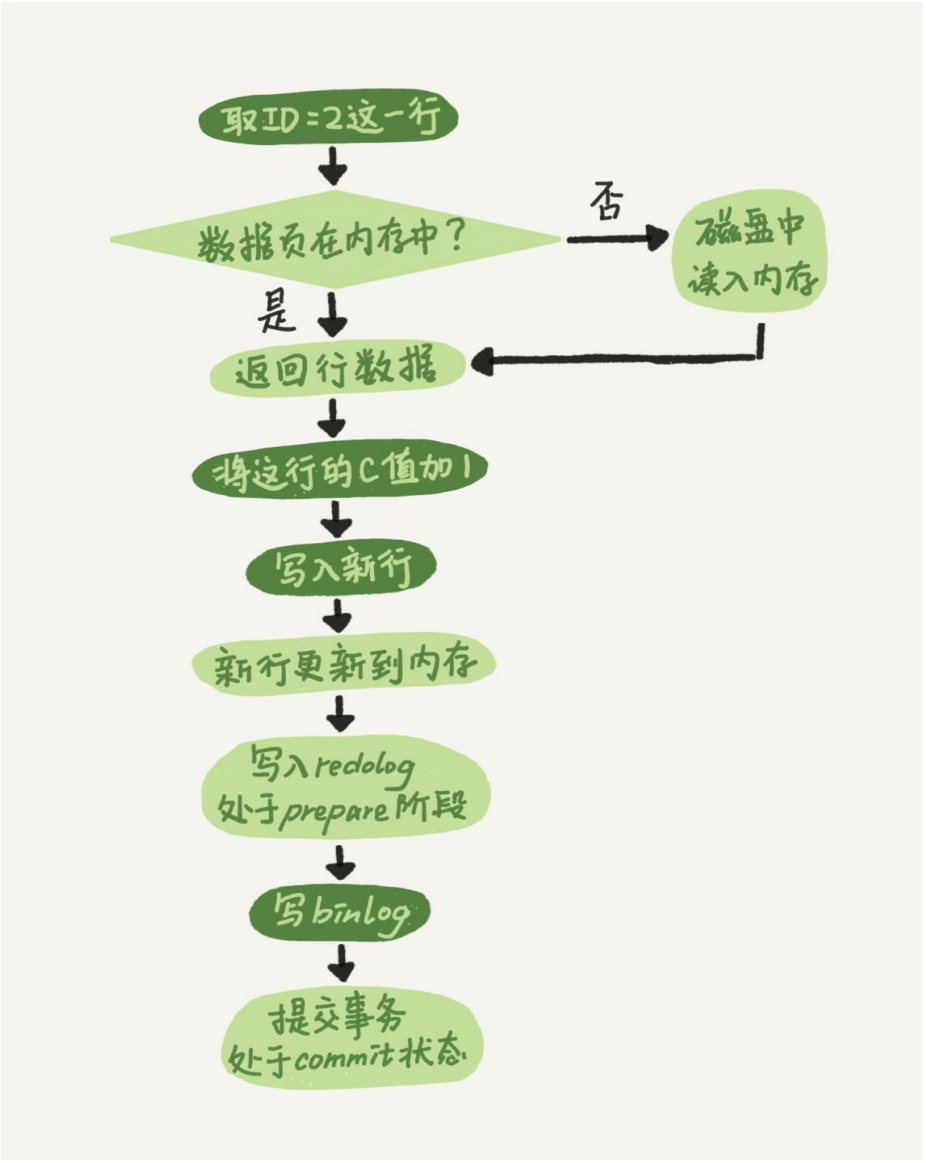
徐过

2020.1.7

MySQL 的逻辑架构图



MySQL update 时执行器与引擎的交互



WAL 技术，WAL 的全称是 Write-Ahead Logging，它的关键点就是先写日志，再写磁盘，具体来说，当有一条记录需要更新的时候，InnoDB 引擎就会先把记录写到 redo log里面，并更新内存，这个时候更新就算完成了。同时，InnoDB 引擎会在适当的时候，将这个操作记录更新到磁盘里面，而这个更新往往是在系统比较空闲的时候做。

1. 物理日志redo log

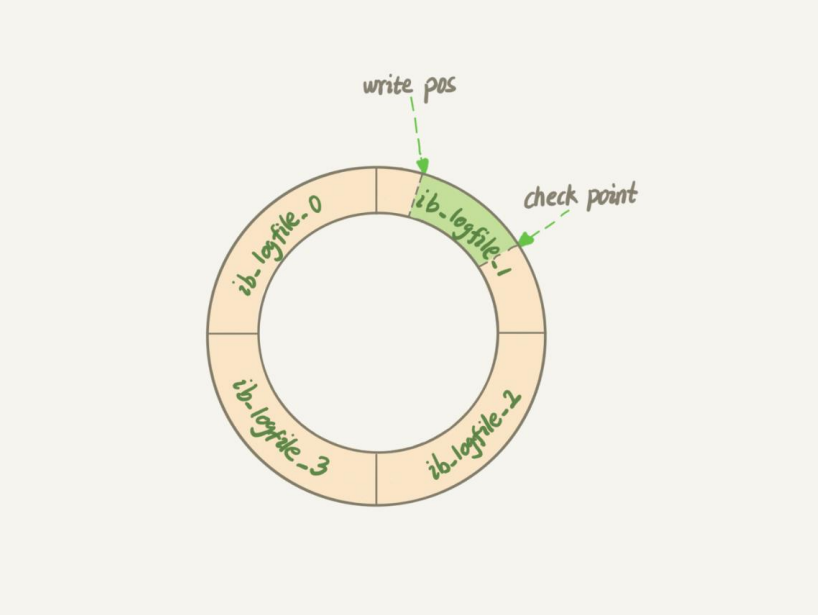
   InnoDB 引擎特有， 记录的是“在某个数据页上做了什么修改”，提供 crash-safe 能力。

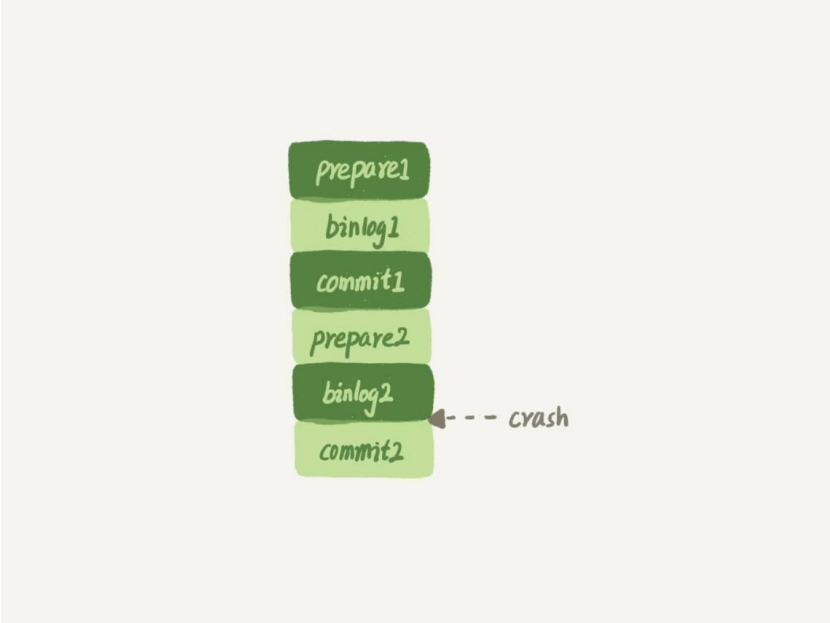
2. 逻辑日志binlog

   MySQL Server层，记录的是这个语句的原始逻辑，比如“给 ID=2 这一行的 c 字段加 1 ”, 提供备份以及恢复能力。

为什么MYSQL同时需要redo log和binlog两种日志

这两种日志有以下三点不同。
1. redo log 是 InnoDB 引擎特有的；binlog 是 MySQL 的 Server 层实现的，所有引擎都可以使用。
2. redo log 是物理日志，记录的是“在某个数据页上做了什么修改”；binlog 是逻辑日志，记录的是这个语句的原始逻辑，比如“给 ID=2 这一行的 c 字段加 1 ”。
3. redo log 是循环写的，空间固定会用完；binlog 是可以追加写入的。“追加写”是指 binlog 文件写到一定大小后会切换到下一个，并不会覆盖以前的日志。





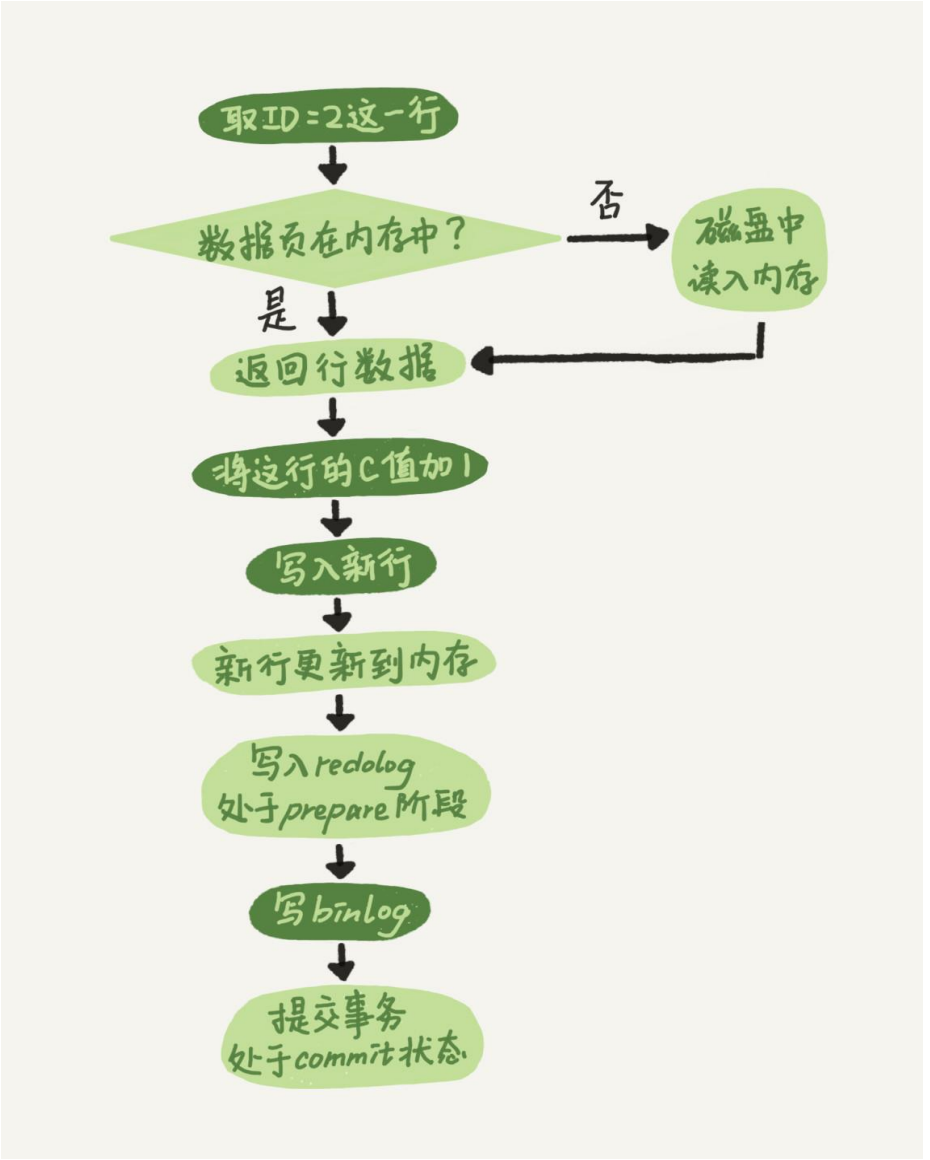
为什么MYSQL需要两阶段提交

1. 先写 redo log 后写 binlog

redo log 写完之后系统崩溃，将c值恢复为1，binlog 没写，备份或者恢复数据库时c值为0。

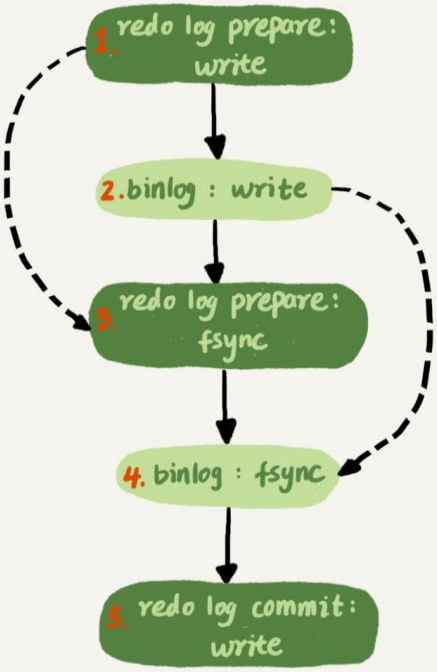
2. 先写 binlog 后写 redo log

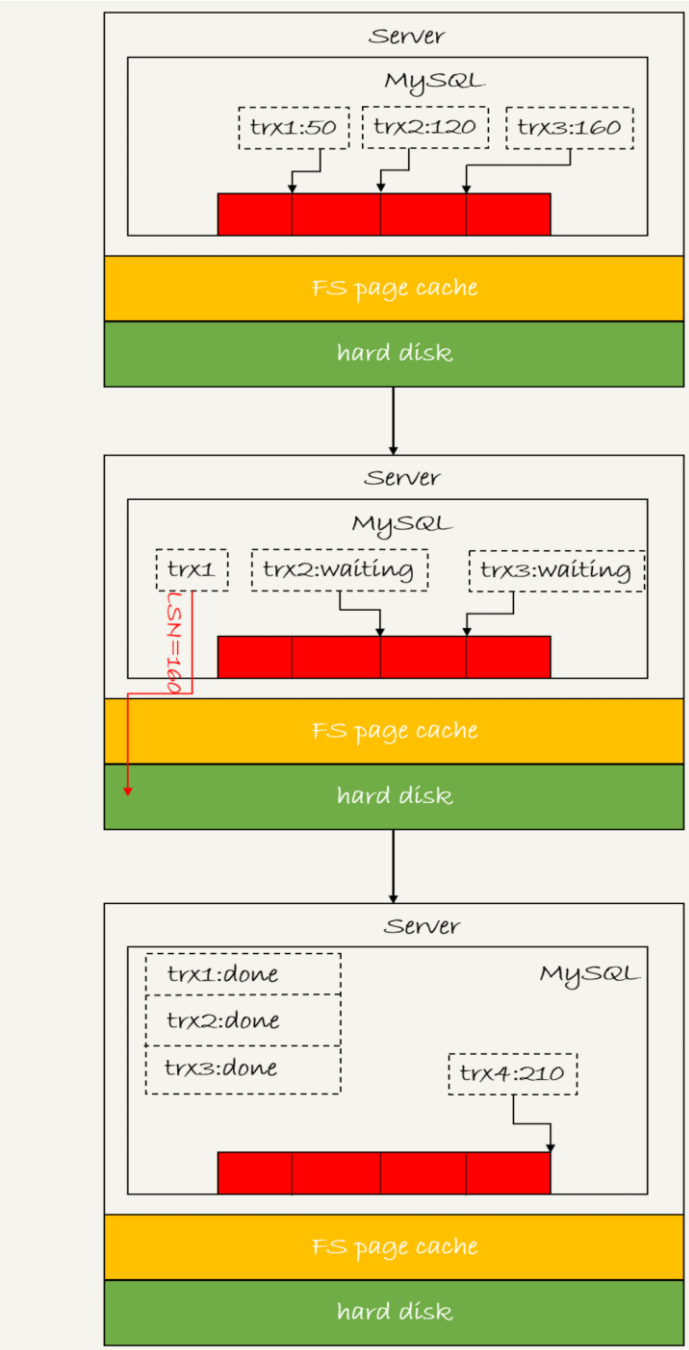
如果在 binlog 写完之后 crash，由于 redo log 还没写，崩溃恢复以后这个事务无效，所以这一行 c 的值是 0。但是 binlog 里面已经记录了“把 c 从 0 改成 1”这个日志。所以，在之后用 binlog 来恢复的时候就多了一个事务出来，恢复出来的这一行 c 的值就是 1，与原库的值不同。



组提交（group commit）机制

日志逻辑序列号（log sequence number，LSN ），用来对应 redo log 的一个个写入点,每次写入长度为 length 的 redo log，LSN 的值就会加上 length。





binlog 格式

mysql> delete from t where a>=4 and t_modified<='2018-11-10' limit 1;

1. statement

```
| master.000001 | 5889 | Anonymous_Gtid |         1 |        5954 | SET @@SESSION.GTID_NEXT= 'ANONYMOUS'                                                                |
| master.000001 | 5954 | Query          |         1 |        6041 | BEGIN                                                                                               |
| master.000001 | 6041 | Query          |         1 |        6197 | use `test`; delete from t /*comment*/  where a>=4 and t_modified<='2018-11-10' limit 1              |
| master.000001 | 6197 | Xid            |         1 |        6228 | COMMIT /* xid=61 */                                                                                 |
```

2. row

```
BEGIN
/*!*/;
# at 9045
#181229 23:32:22 server id 1  end_log_pos 9092 CRC32 0xdbfc0a8c         Table_map: `test`.`t` mapped to number 226
# at 9092
#181229 23:32:22 server id 1  end_log_pos 9140 CRC32 0x0cda8921         Delete_rows: table id 226 flags: STMT_END_F

BINLOG '
hpMnXBMBAAAALwAAAIQjAAAAAOIAAAAAAAEABHRlc3QAAXQAAwMDEQEAAowK/Ns=
hpMnXCABAAAAMAAAALQjAAAAAOIAAAAAAAEAAgAD//gEAAAABAAAAFvl9VAhidoM
'/*!*/;
### DELETE FROM `test`.`t`
### WHERE
###   @1=4 /* INT meta=0 nullable=0 is_null=0 */
###   @2=4 /* INT meta=0 nullable=1 is_null=0 */
###   @3=1541797200 /* TIMESTAMP(0) meta=0 nullable=0 is_null=0 */
# at 9140
#181229 23:32:22 server id 1  end_log_pos 9171 CRC32 0x1beb44f1         Xid = 68
COMMIT/*!*/;
SET @@SESSION.GTID_NEXT= 'AUTOMATIC' /* added by mysqlbinlog */ /*!*/;
```

binlog 格式

mysql> insert into t values(10,10, now());

3. mixed

利用 statment 格式的优点，同时又避免了数据不一致的风险。

```
| master.000001 | 2738 | Query          |         1 |        2825 | BEGIN

| master.000001 | 2825 | Query          |         1 |        2942 | use `test`; insert into t values(100, 1, now())

| master.000001 | 2942 | Xid            |         1 |        2973 | COMMIT /* xid=41 */
```

```
BEGIN
/*!*/;
# at 2825
#181230  1:11:31 server id 1  end_log_pos 2942 CRC32 0x0ecd5082        Query   thread_id=4     exec_time=0     error_code=0
SET TIMESTAMP=1546103491/*!*/;
insert into t values(100, 1, now())
/*!*/;
# at 2942
#181230  1:11:31 server id 1  end_log_pos 2973 CRC32 0x09877081        Xid = 41
COMMIT/*!*/;
```

事务

SQL 标准的事务隔离级别包括：读未提交（read uncommitted）、读提交（read committed）、可重复读（repeatable read）和串行化（serializable ）。

1. 读未提交是指，一个事务还没提交时，它做的变更就能被别的事务看到，**没有视图概念。**

2. 读提交是指，一个事务提交之后，它做的变更才会被其他事务看到，**每个 SQL 语句开始执行的时候创建视图。**

3. 可重复读是指，一个事务执行过程中看到的数据，总是跟这个事务在启动时看到的数据是一致的，**事务启动时创建视图。**

4. 串行化，顾名思义是对于同一行记录，“写”会加“写锁”，“读”会加“读锁”。当出现读写锁冲突的时候，后访问的事务必须等前一个事务执行完成，才能继续执行，**不需要视图。**

事务A	事务B
启动事务 查询得到值1	启动事务
	查询得到值1
	将1改成2
查询得到值V1	
	提交事务B
查询得到值V2	
提交事务A	
查询得到值V3	

1. 读未提交， 则 V1、V2、V3 是 2。

2. 读提交，则 V1 是 1，V2 的值是 2。事务 B 的更新在提交后才能被 A 看到。所以， V3 的值也是 2。

3. 可重复读，则 V1、V2 是 1，V3 是 2。

4. 串行化，则在事务 B 执行“将 1 改成 2”的时候，会被block住。直到事务 A 提交后，事务 B 才可以继续执行。所以从 A 的角度看， V1、V2 值是 1，V3 的值是 2。

视图

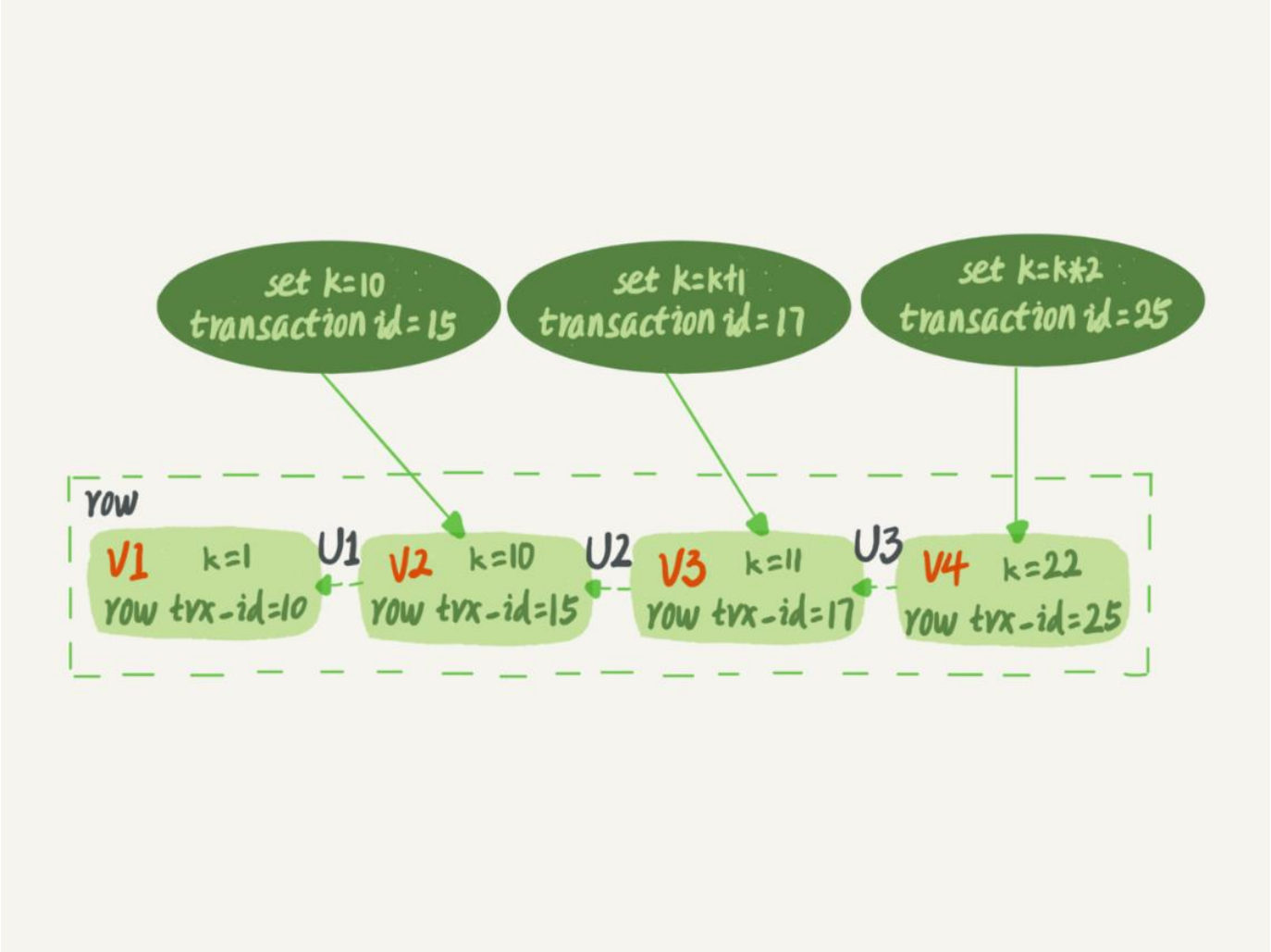
```
mysql> CREATE TABLE `t` (
`id` int(11) NOT NULL,
`k` int(11) DEFAULT NULL, PRIMARY KEY (`id`)
) ENGINE=InnoDB;

insert into t(id, k) values(1,1);
```

事务A	事务B	事务C
start transaction with consistent snapshot;		
	start transaction with consistent snapshot;	
		update t set k=k+1 where id=1;
	update t set k=k+1 where id=1; select k from t where id=1;	
select k from t where id=1; commit;		
	commit;	

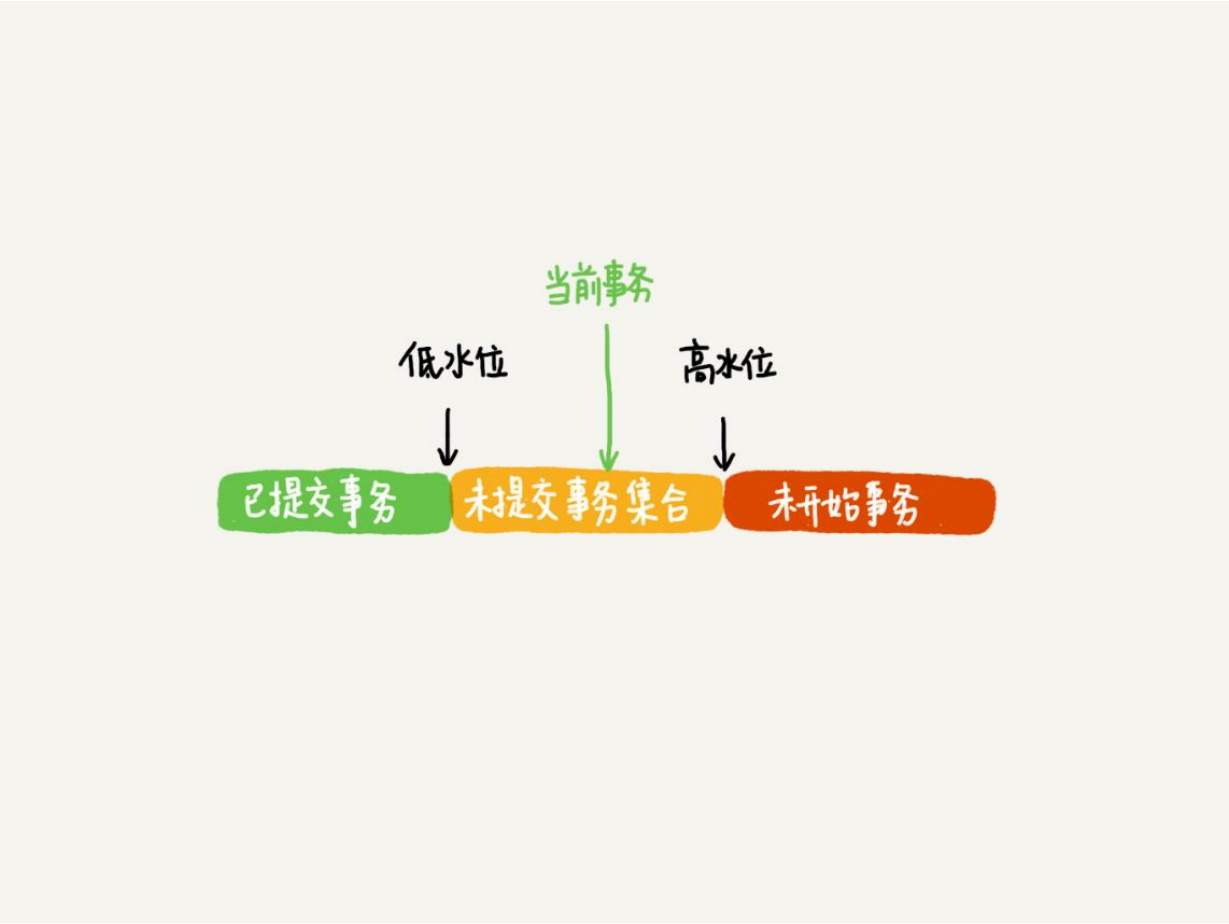
MVCC (Multi-Version Concurrency Control)

数据表中的一行记录，可能有多个版本 (row)，每个版本有自己的 row trx_id。



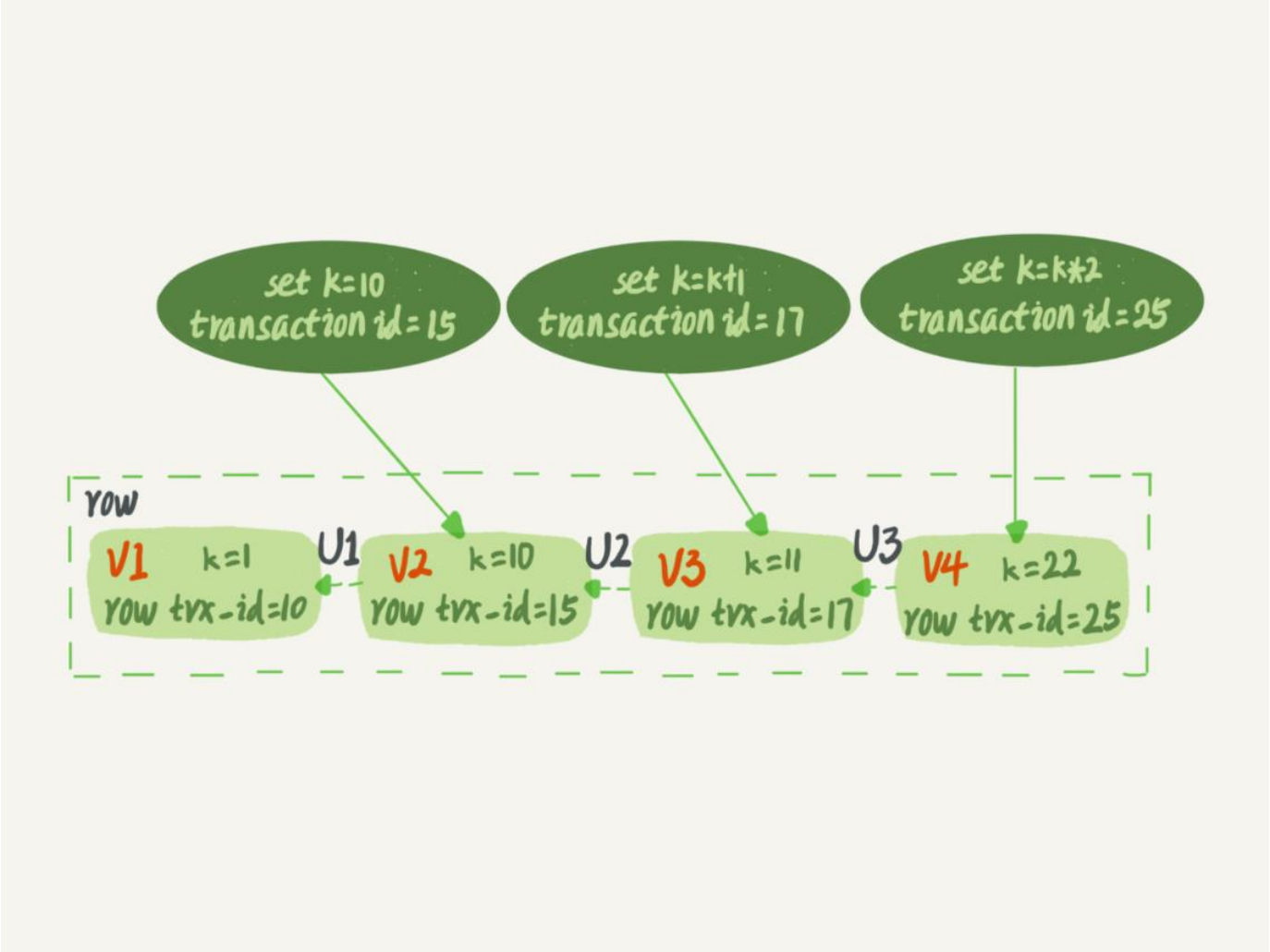
视图 （快照）

以事务启动的时刻为准，如果一个数据版本是在该事务启动之前生成的，就认；如果该事务启动以后才生成的，就不认这个值，必须要找到它的上一个版本。



视图 （快照）

比如，如果有一个事务，id为20，低水位为18，那么当它访问这一行数据时，就会从 V4 通过 U3 计算出 V3，所以在它看来，这一行的值是 11。

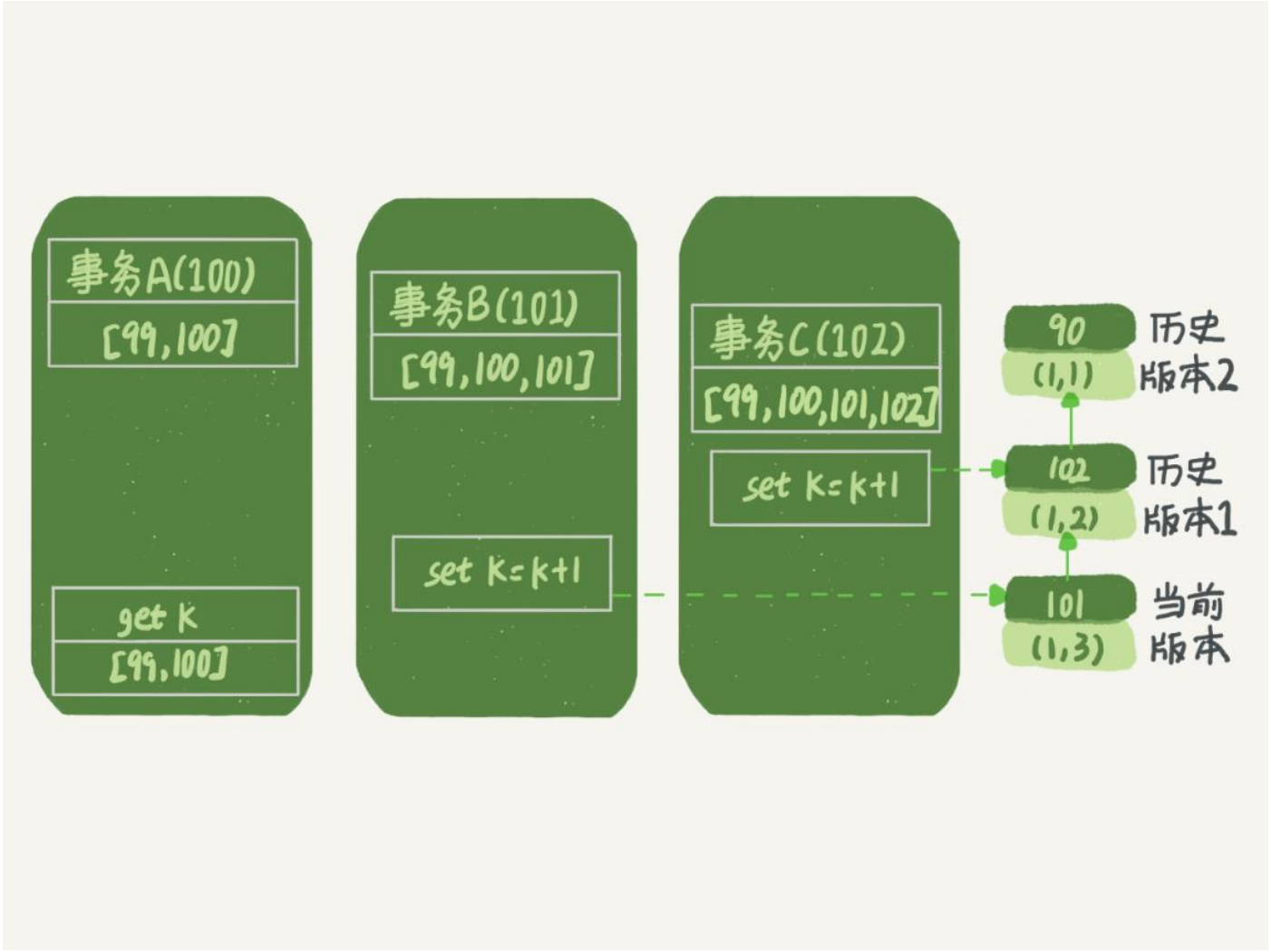


视图

事务A	事务B	事务C
start transaction with consistent snapshot;		
	start transaction with consistent snapshot;	
		update t set k=k+1 where id=1;
	update t set k=k+1 where id=1; select k from t where id=1;	
select k from t where id=1; commit;		
	commit;	

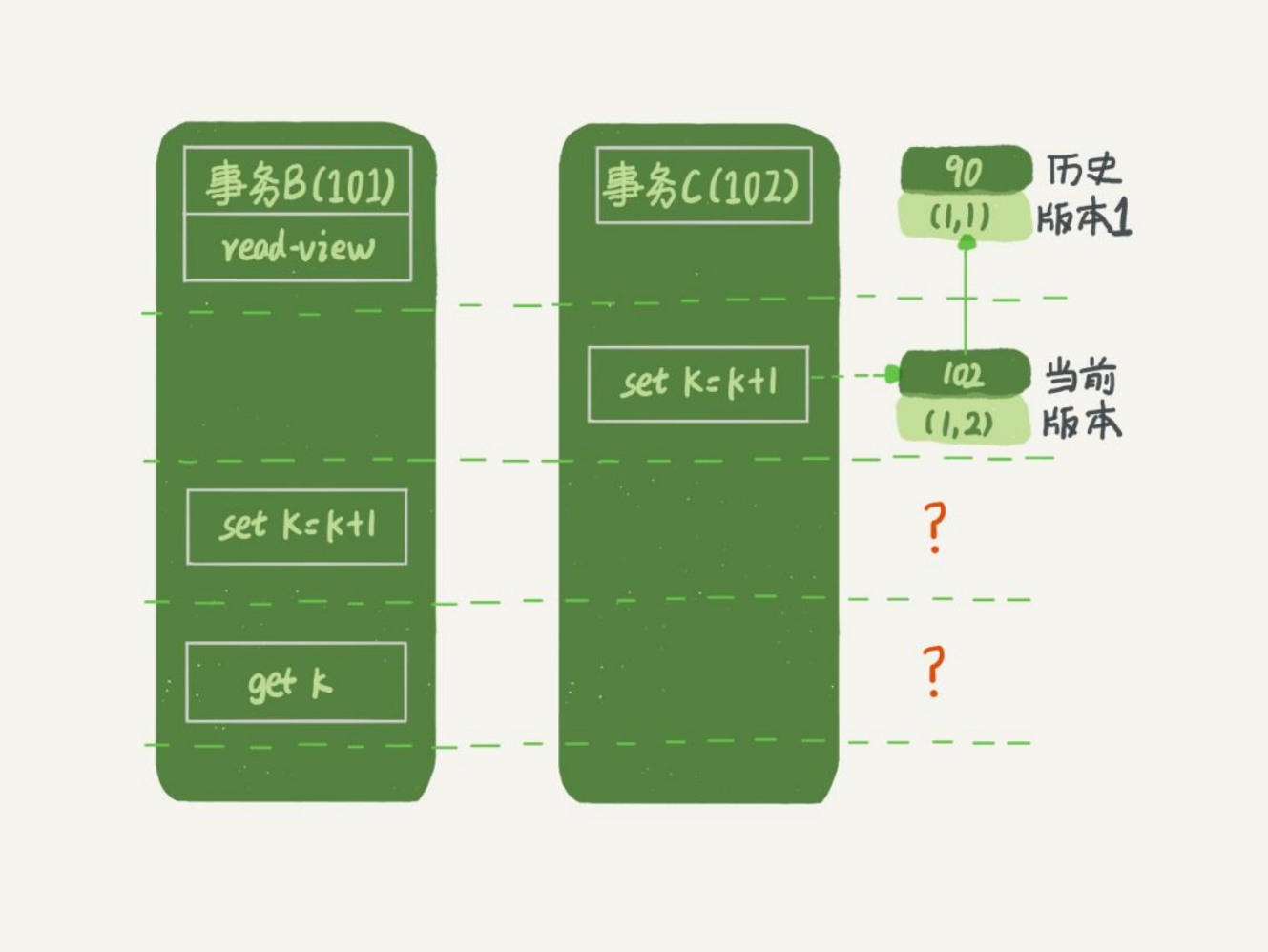
视图 （快照）

事务 A 查询数据逻辑图



视图 （快照）

事务 B 查询数据逻辑图



Thanks