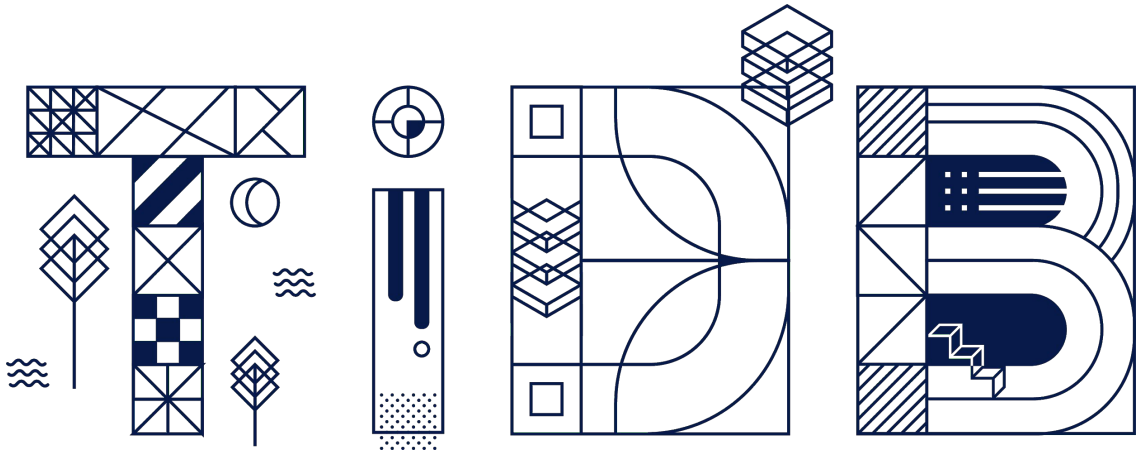
PingCAP TiDB

TiDB 悲观锁

Presented by: shirly



自我介绍

- 吴雪莲
- TiKV 研发工程师
- Transaction 和 Coprocessor 模块负责人
- 曾就职于盛大创新院、京东云、网易杭州研究院，从事公有云 对象存储系统的研发和设计。

Agenda

- 悲观锁要解决的问题
- MySQL 悲观锁行为
- TiDB 悲观事务
    - 等锁
    - 死锁
- 测试
- 使用

PingCAP

Part I - 悲观锁解决的问题

乐观事务

为什么加购物车的时候明明还有，但是我付款的时候提示没了呢？

可能是因为下单扣减库存是用了乐观锁吧。

啥玩意？我都没买到喜欢的裙子，你还乐观？

PingCAP

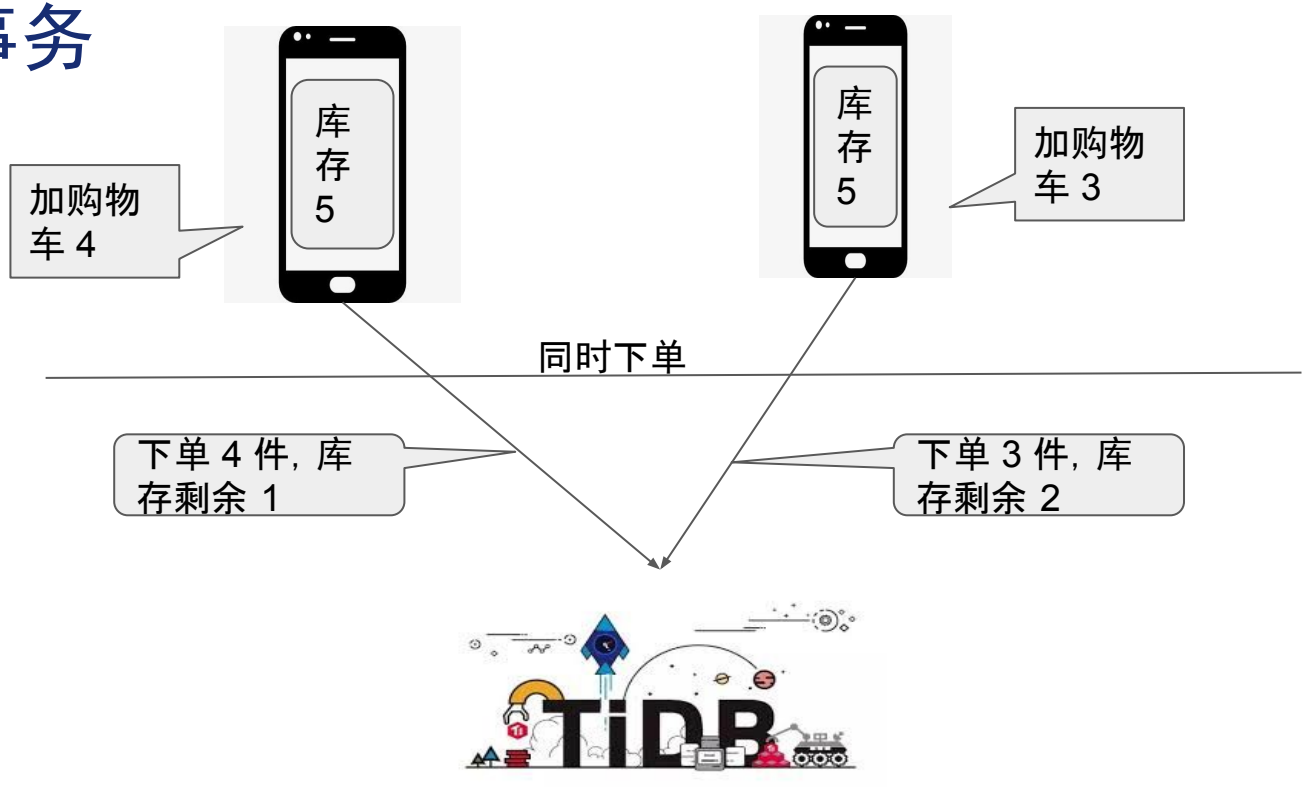
不是啦，这是并发控制的一种方式呢。

额、我听不懂，你给我讲讲。

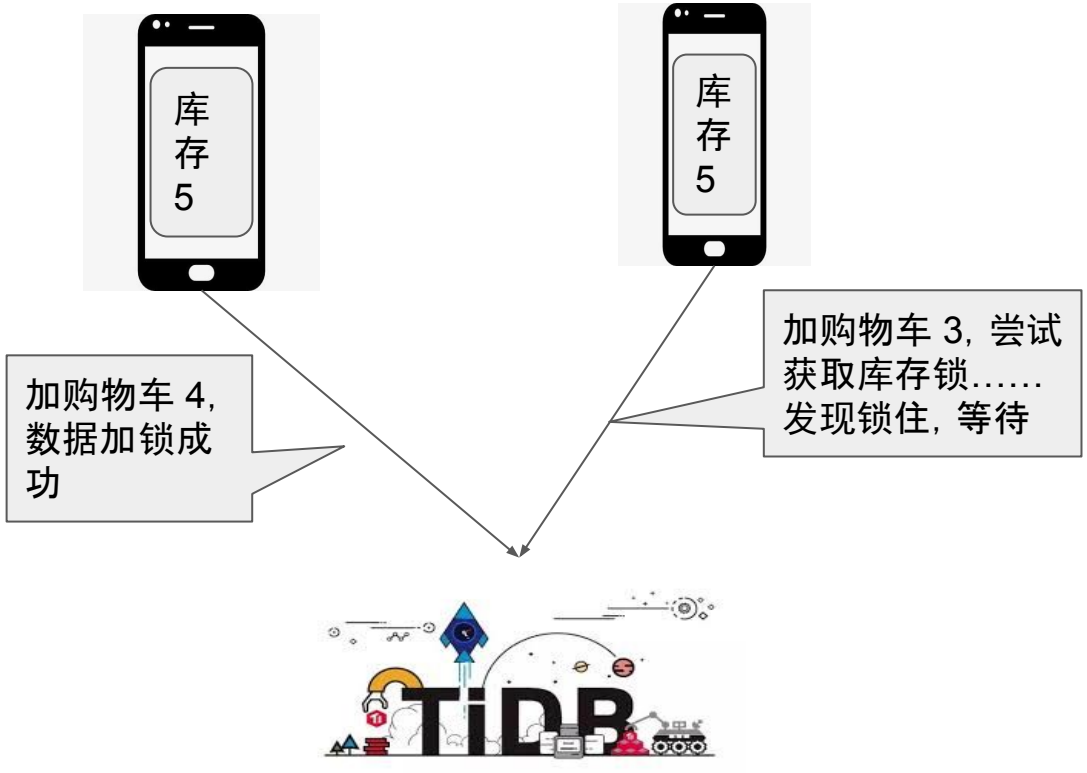
我有一个悲观锁，和一个乐观锁，你想听哪一个？

那，先听悲观锁吧。

乐观事务



悲观事务



乐观事务 VS 悲观事务

冲突检测的检测时间点：

- 乐观：加购物车成功 =》下单不一定成功
- 悲观：加购物车成功 =》下单一定成功

乐观事务存在的问题

- 提交不一定成功。
- 冲突需要手动重试。
- 冲突严重且重试代价大的场景性能差，失败率高。
- MySQL 用户习惯于悲观事务模型，应用侧一般不会重试。

悲观锁解决的问题

- 通过支持悲观事务，降低用户修改代码的难度甚至不用修改代码。
- 拓展 TiDB 的应用场景。

PingCAP

Part II - MySQL 悲观锁行为

PostgreSQL

Session A	*Session B*
> begin; > set transaction isolation level repeatable read;	> begin; > set transaction isolation level repeatable read;
> update test set value = value + 1 where id = 1; UPDATE 1	
	> update test set value = value + 1 where id = 1; block...
> commit; COMMIT	ERROR: could not serialize access due to concurrent update

MySQL

Session A	*Session B*
> begin;	> begin;
> update test set value = value + 1 where id = 1; Query OK, 1 row affected	
	> update test set value = value + 1 where id = 1; block...
> commit; Query OK, 0 rows affected	Query OK, 1 row affected
	> commit; Query OK, 0 rows affected

MySQL

Session A	*Session B*
> create table t (i int);	
> begin; > select * from t; Empty set	
	> insert into t values (1); Query OK, 1 row affected
> select * from t for update; 1 row in set	
> select * from t; Empty set	

MySQL 悲观锁行为

- LOCKING READS、UPDATE 和 DELETE 基于锁。
- 写写冲突不会 abort，基于锁的行为会读到最新已提交的数据。
- 写写冲突不一定会导致异常，写操作基于最新已提交的数据能避免大部分异常。

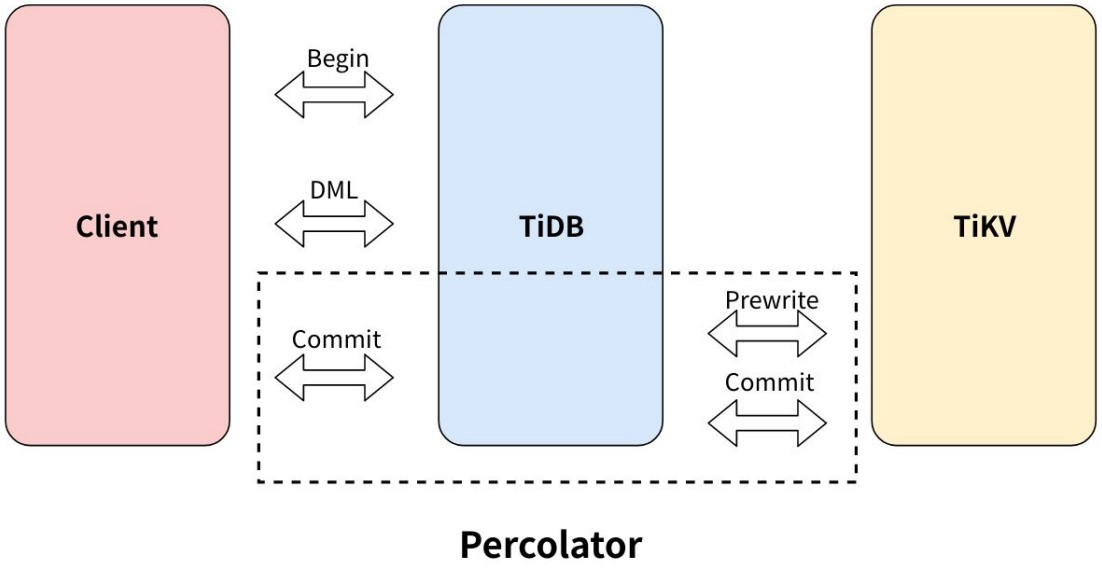
PingCAP

Part III - TiDB 悲观事务

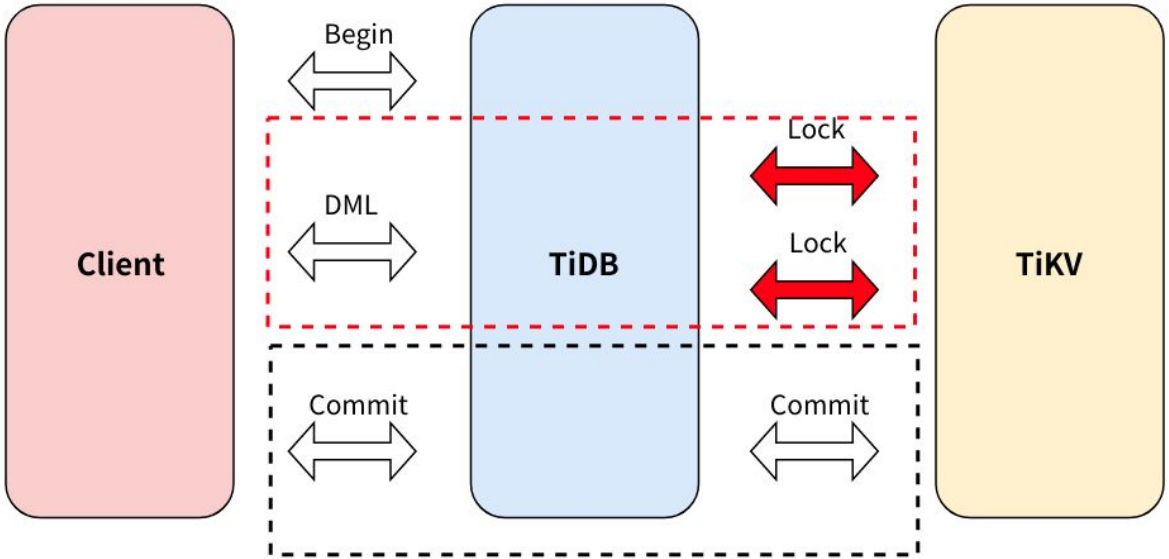
实现上的考虑

- 在现有实现基础上进行改造，减少改动。
- 兼容现有乐观事务模型，支持乐观、悲观混合使用。
- 兼容 MySQL 行为。

乐观事务

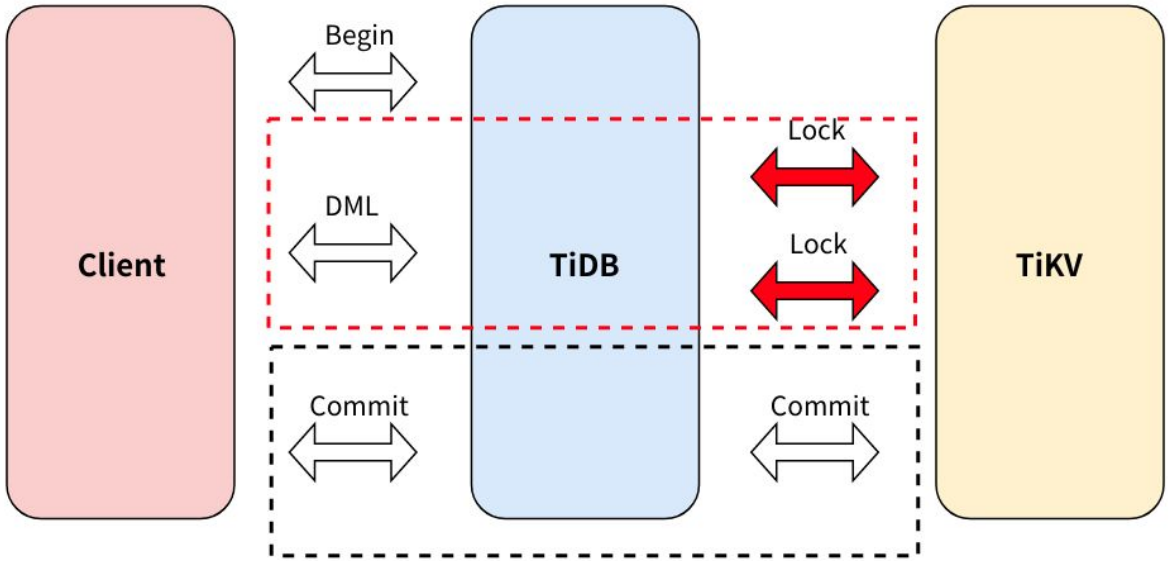

Client
Begin
DML
Commit
TiDB
Prewrite
Commit
TiKV
Percolator

初步想法

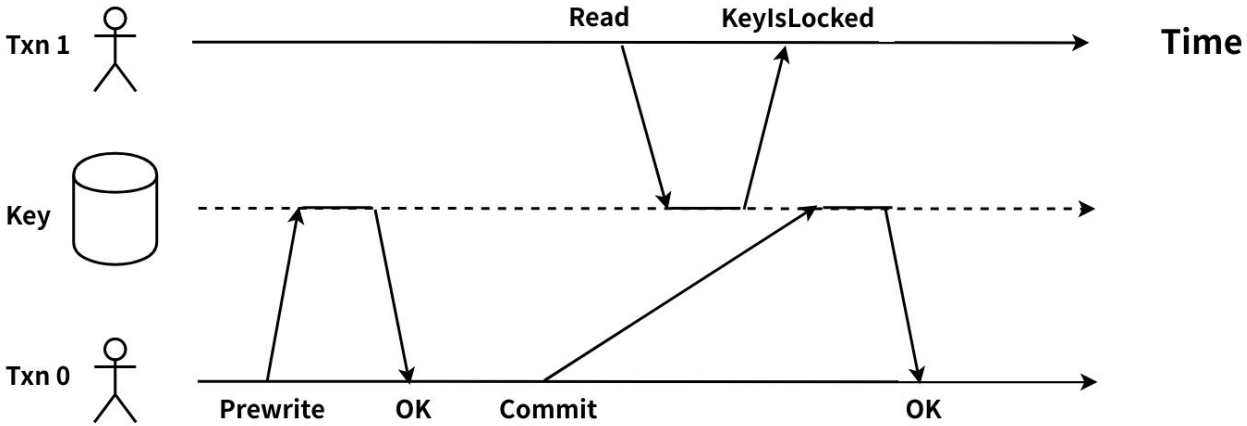


问题

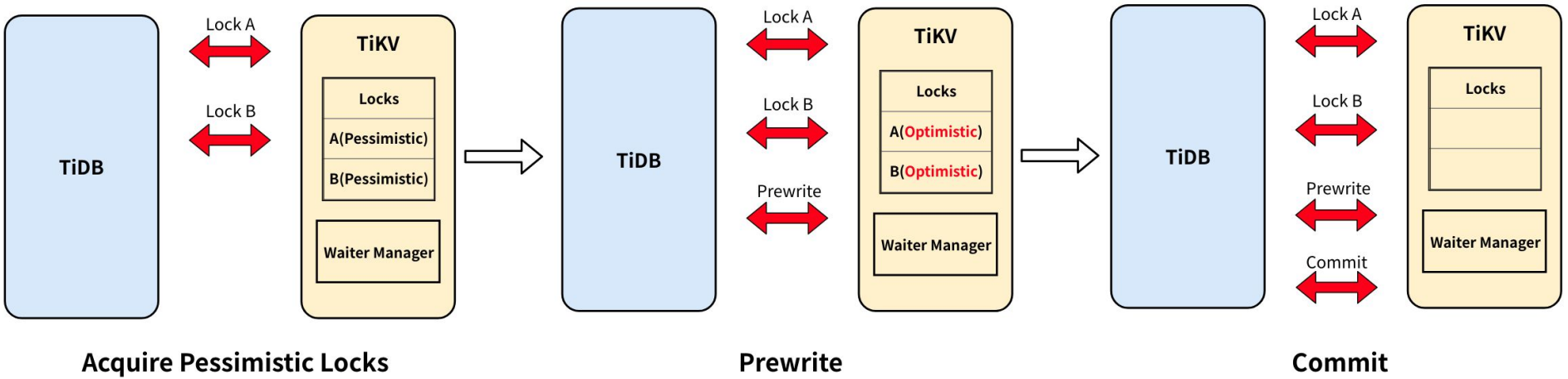
- 锁的存在会阻塞读，悲观锁存在时间长。



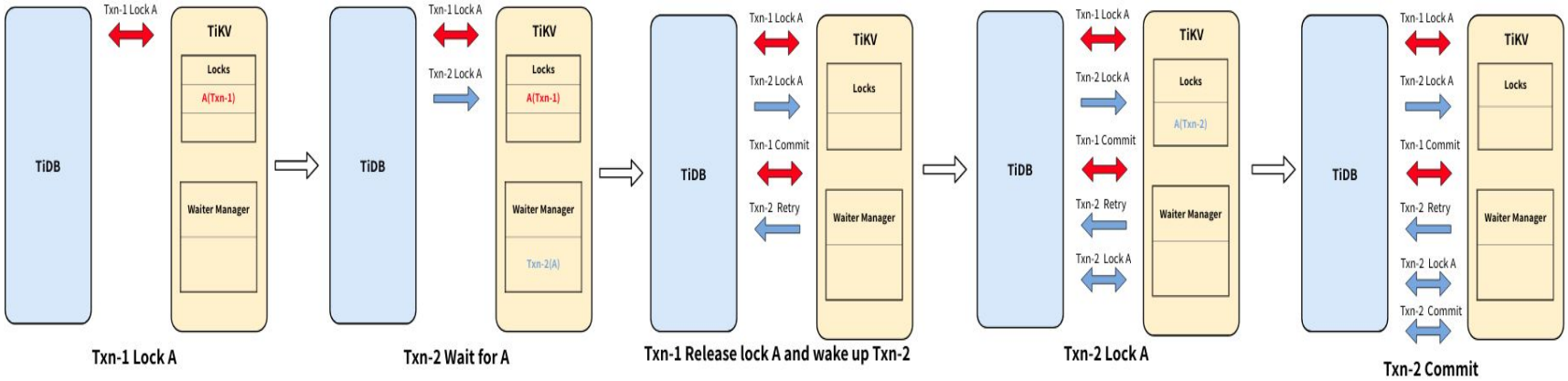
为什么 TiDB 乐观事务中锁会阻塞读？



悲观事务的实现


TiDB
Lock A
Lock B
TiKV
Locks
A(Pessimistic)
B(Pessimistic)
Waiter Manager
TiDB
Lock A
Lock B
Prewrite
TiKV
Locks
A(Optimistic)
B(Optimistic)
Waiter Manager
TiDB
Lock A
Lock B
Prewrite
Commit
TiKV
Locks
Waiter Manager
Acquire Pessimistic Locks
Prewrite
Commit

Pessimistic Lock —— 遇到其他事务的锁



PingCAP

PingCAP

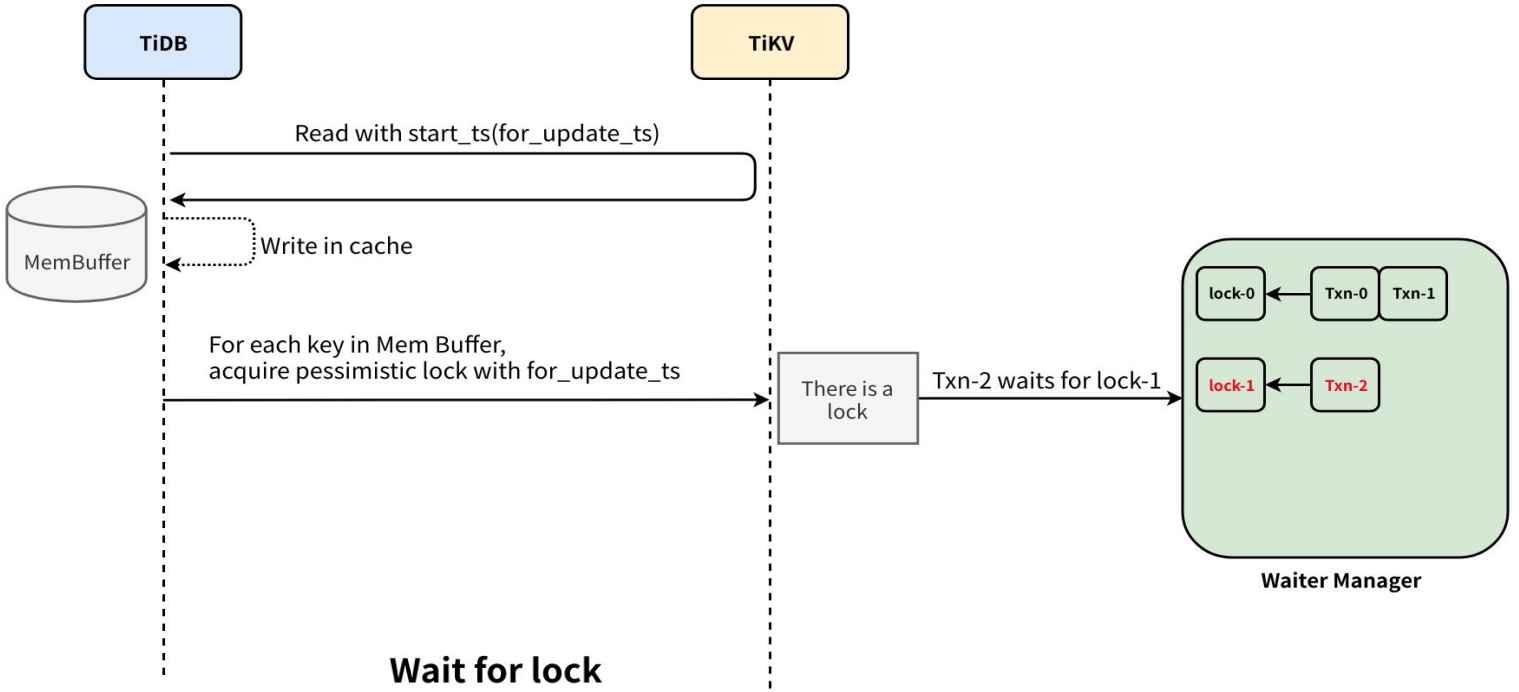
Part III.I - 等锁

在哪里等锁？

- 若在 TiDB 实现，要定期重试，不及时且代价大。
- 在 TiKV 中实现等锁，能够及时得知锁被释放。

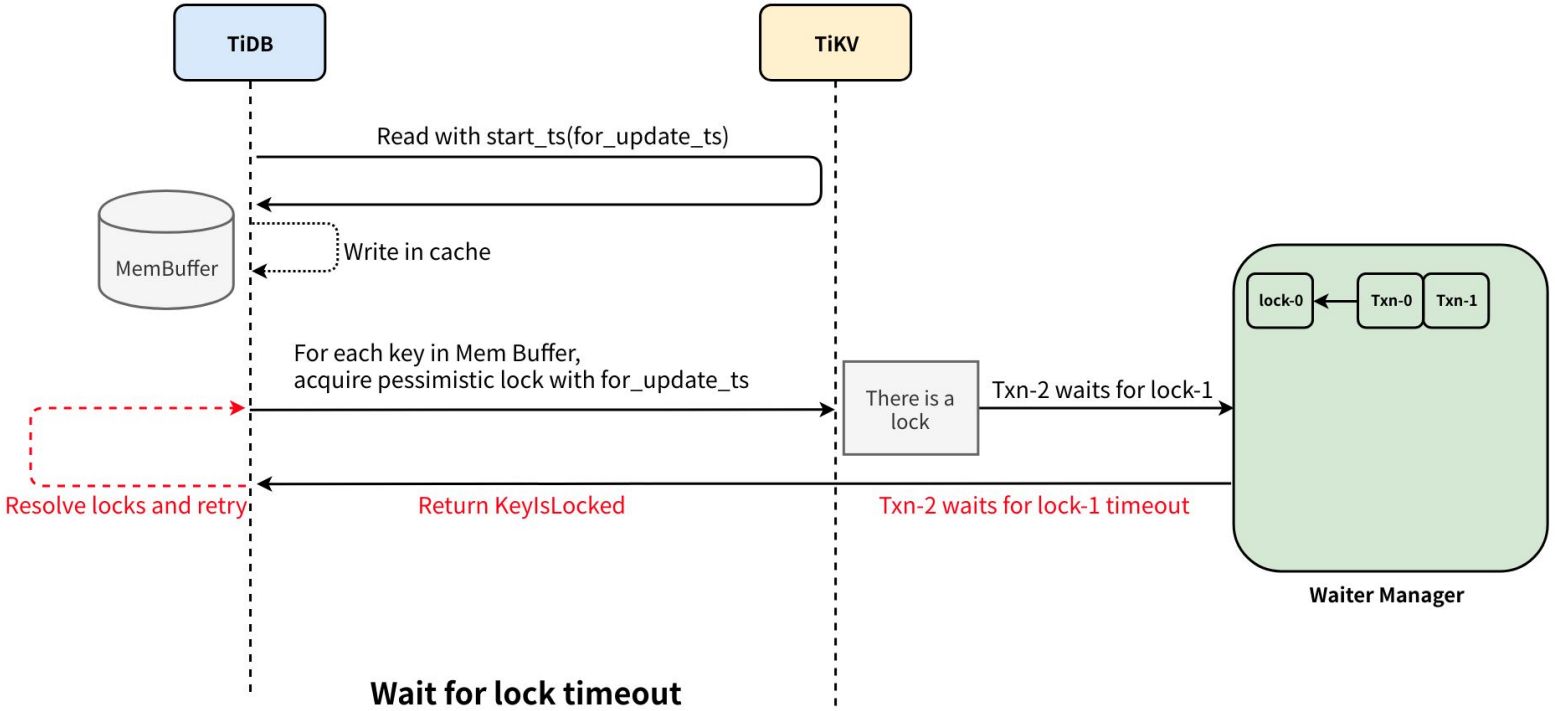
Waiter Manager

- 当遇到锁时，会在 TiKV 中等锁。



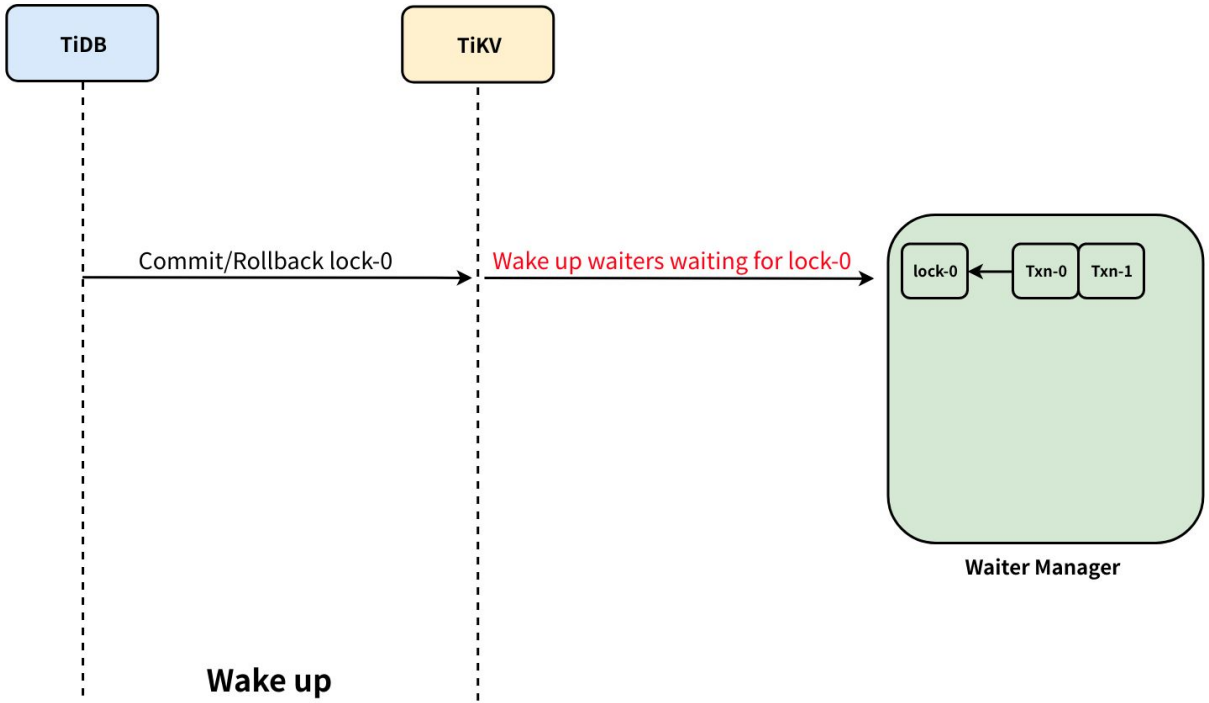
Waiter Manager- time out

- 等锁有超时时间，默认为 3s，可配置。



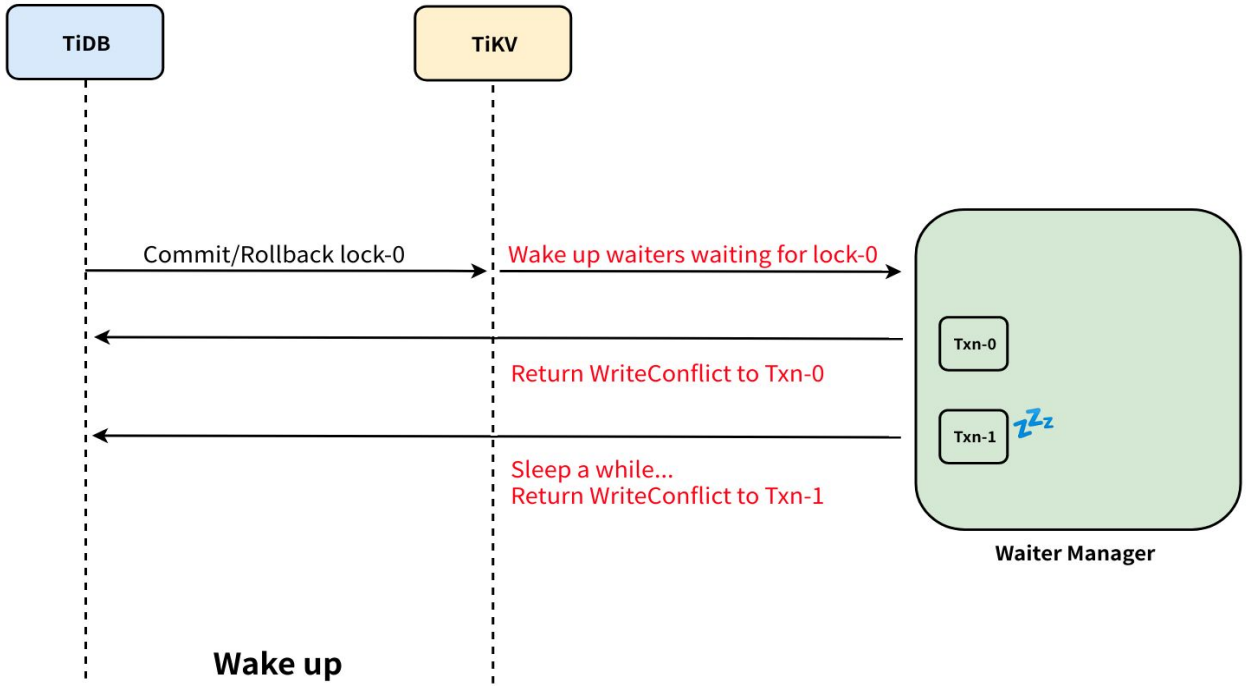
Waiter Manager-wake up

- 当锁被释放时(提交或回滚)，会唤醒等待的事务。



Waiter Manager

- 当有多个事务等待同一个锁时，会按照事务的 start_ts 排序来返回响应，让 start_ts 小的更有可能获取到锁。



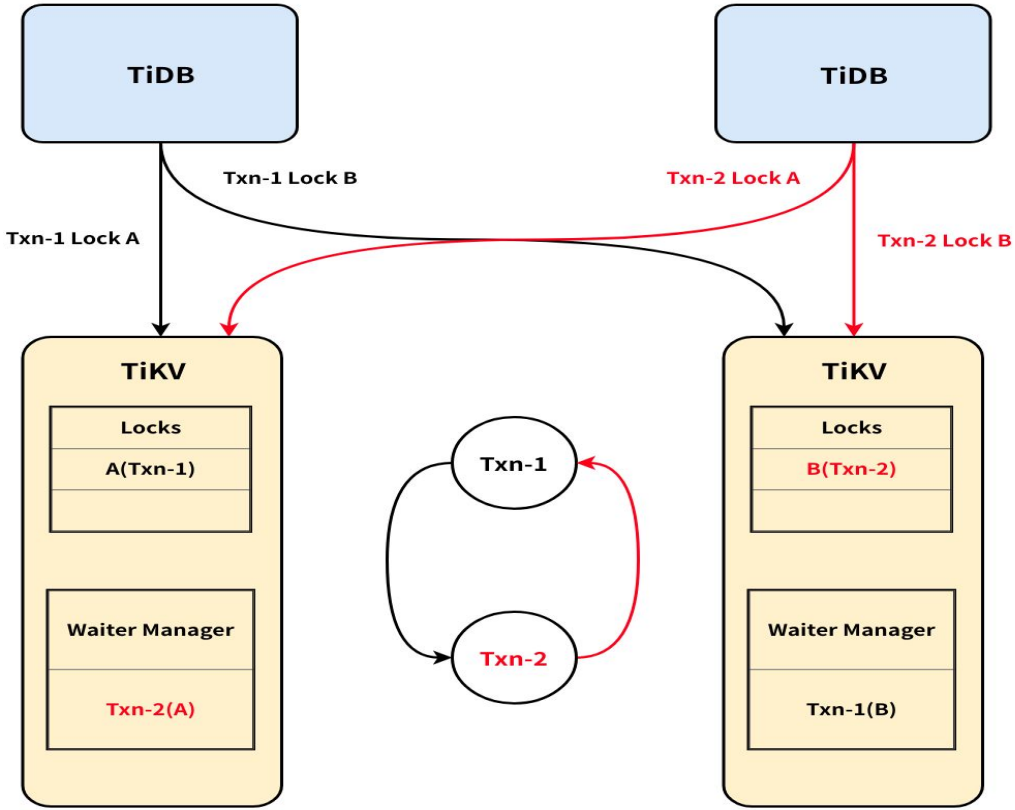
Waiter Manager

- 在纯乐观事务场景，不会有多余计算和唤醒。

PingCAP

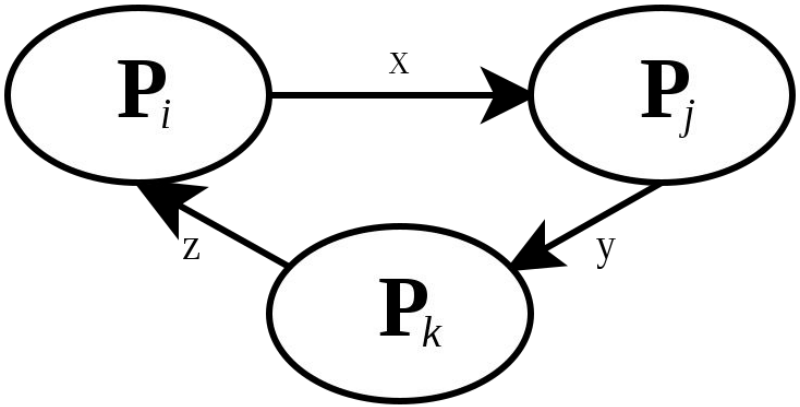
Part III.II - 死锁

死锁


TiDB
TiDB
Txn-1 Lock B
Txn-2 Lock A
Txn-1 Lock A
Txn-2 Lock B
TiKV
Locks
A(Txn-1)
Waiter Manager
Txn-2(A)
Txn-1
Txn-2
TiKV
Locks
B(Txn-2)
Waiter Manager
Txn-1(B)

死锁解决方案

- 死锁检测：维护全局的等待图，等锁时添加等待路径，拒绝在图中产生环的动作。



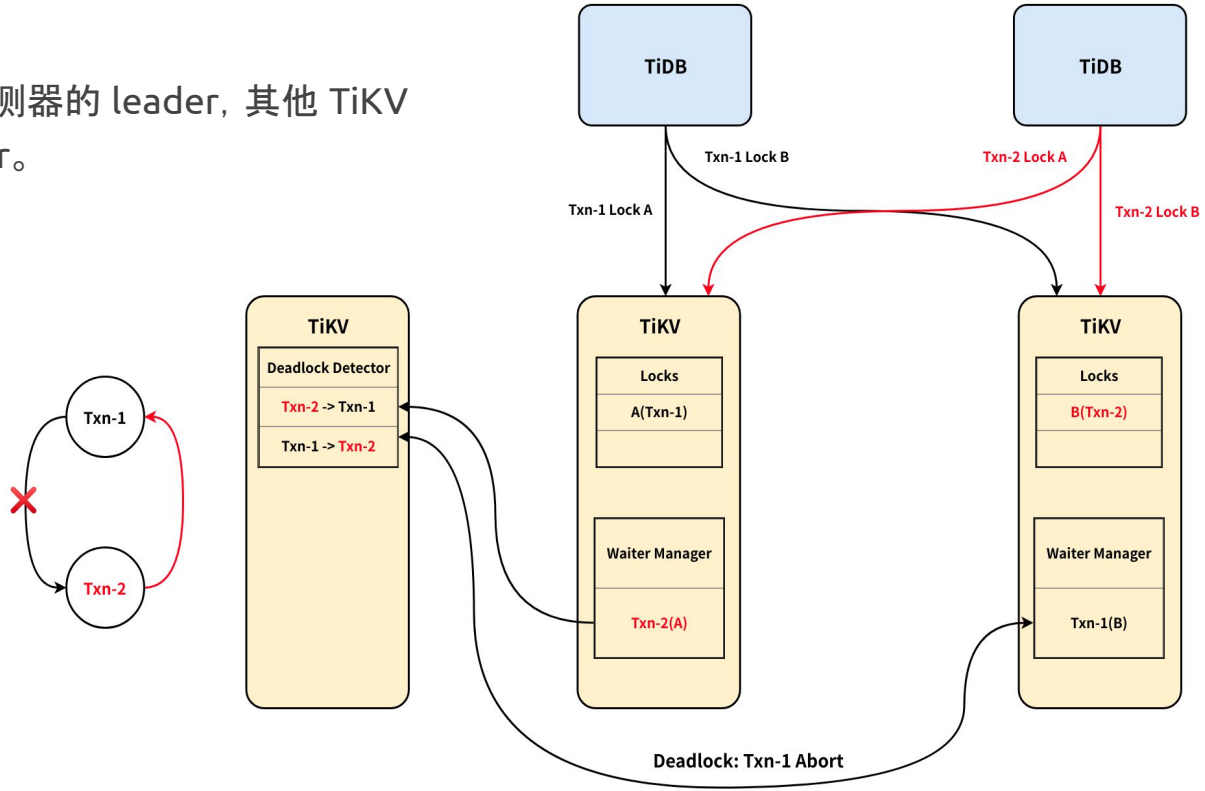
死锁解决方案

- 死锁避免：
  - 实现简单，无中心节点。
  - 只允许一种顺序的等待，事务回滚较多。

- 死锁检测：
  - 准确检测死锁，最小化事务回滚。
  - 需要有中心节点，实现复杂。

TiDB 的实现

- 在 TiKV 中实现死锁检测
  - 一台 TiKV 作为死锁检测器的 leader，其他 TiKV 发送等锁信息到 leader。



Leader 选举

- Region 1 的 leader 作为死锁检测器的 leader。
- 拥有 Region 1 的 TiKV 通过 Raft 得知 leader 变更。
- 其他 TiKV 通过 PD 得知 leader 变更。

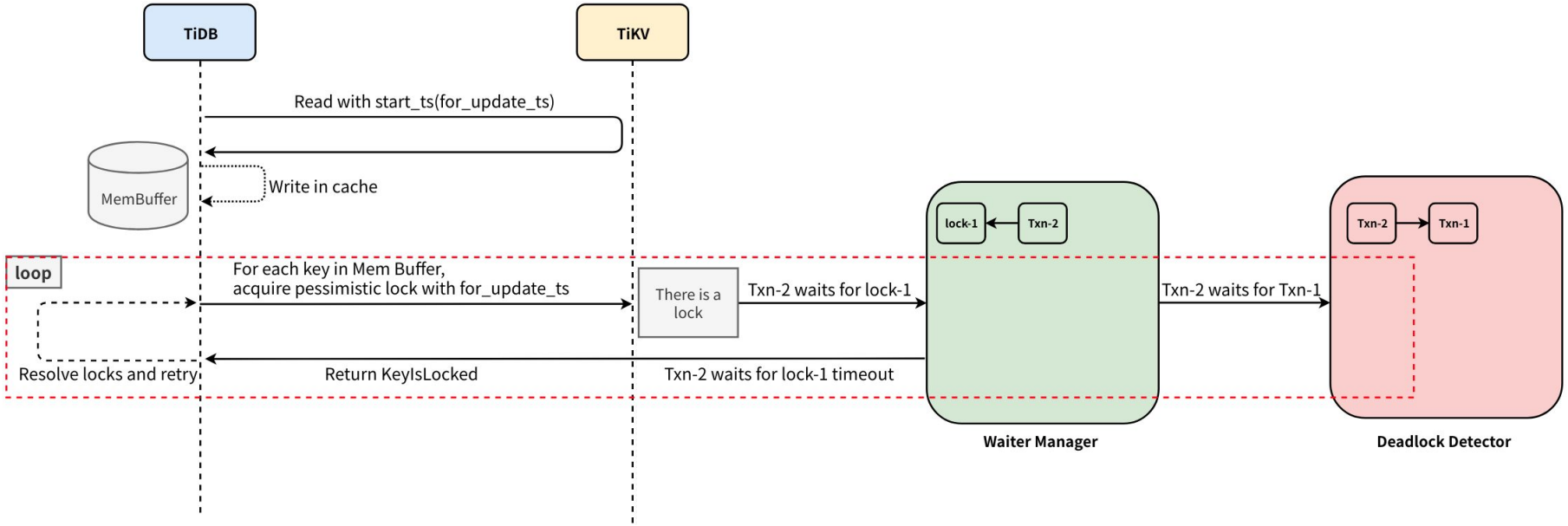
问题

- 等锁请求发送失败，如何处理？
- leader 还没选举出来，怎么处理？
- leader 变更了，新的 leader 没有之前的 wait-for graph，如何处理？

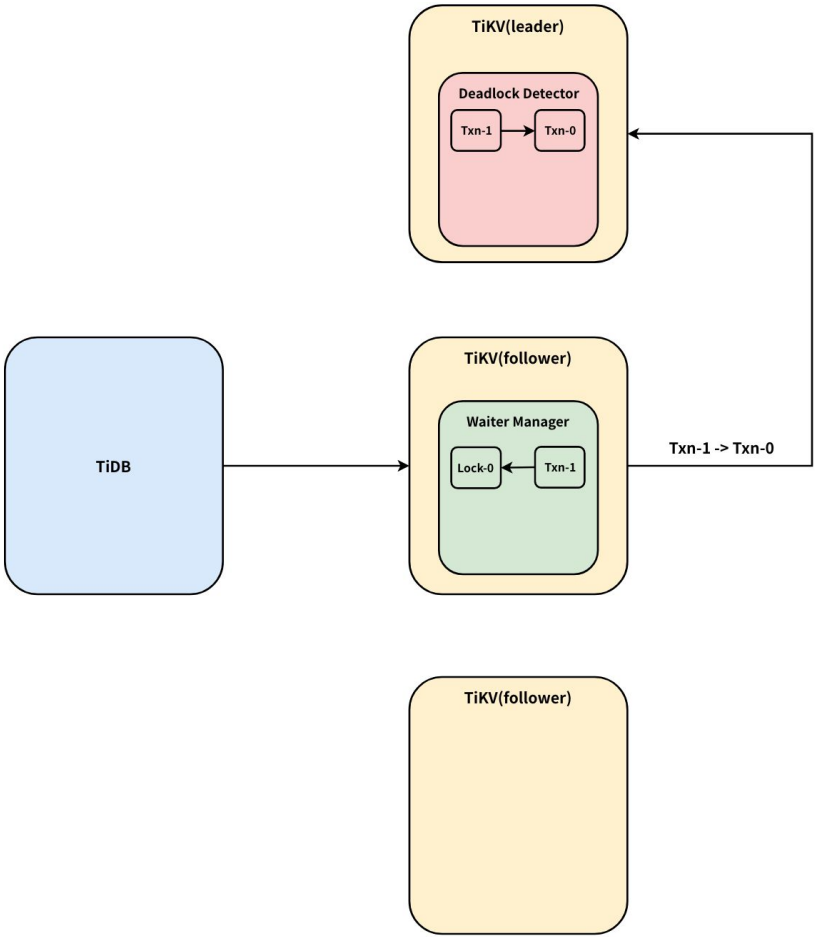
问题

- 等锁请求发送失败，如何处理？
- leader 还没选举出来，怎么处理？
- leader 变更了，新的 leader 没有之前的 wait-for graph，如何处理？
- 重试！

等锁发送失败

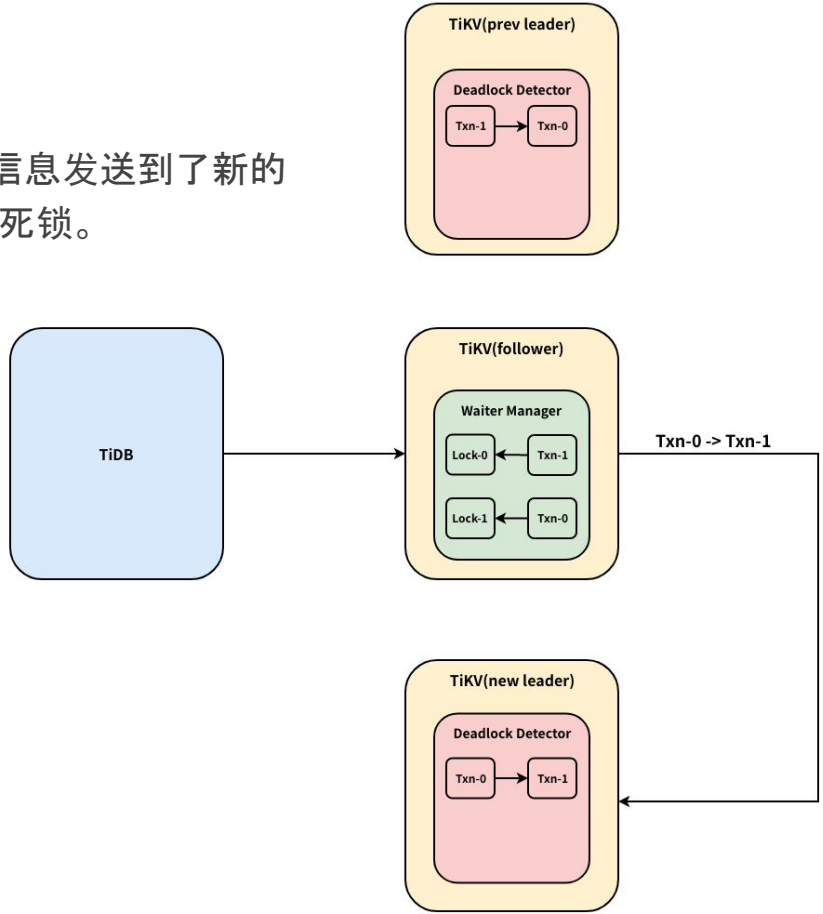

TiDB
TiKV
Read with start_ts(for_update_ts)
MemBuffer
Write in cache
loop
For each key in Mem Buffer,
acquire pessimistic lock with for_update_ts
There is a lock
Txn-2 waits for lock-1
lock-1
Txn-2
Txn-2 waits for Txn-1
Txn-2
Txn-1
Resolve locks and retry
Return KeyIsLocked
Txn-2 waits for lock-1 timeout
Waiter Manager
Deadlock Detector

Leader 变更


TiKV(leader)
Deadlock Detector
Txn-1
Txn-0
TiDB
TiKV(follower)
Waiter Manager
Lock-0
Txn-1
Txn-1 -> Txn-0
TiKV(follower)

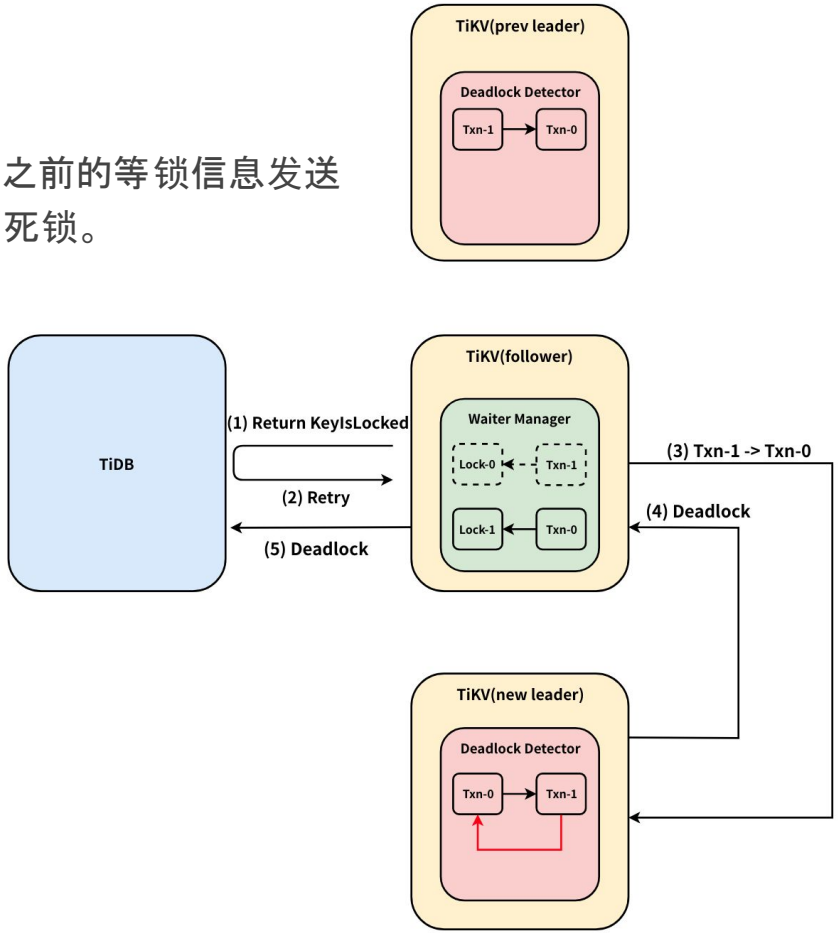
Leader 变更

- leader 变化了，等锁信息发送到了新的 leader 导致未检测出死锁。



Leader 变更

- 等锁超时，TiDB 重试，之前的等锁信息发送到新的 leader，检测到死锁。



单语句 rollback

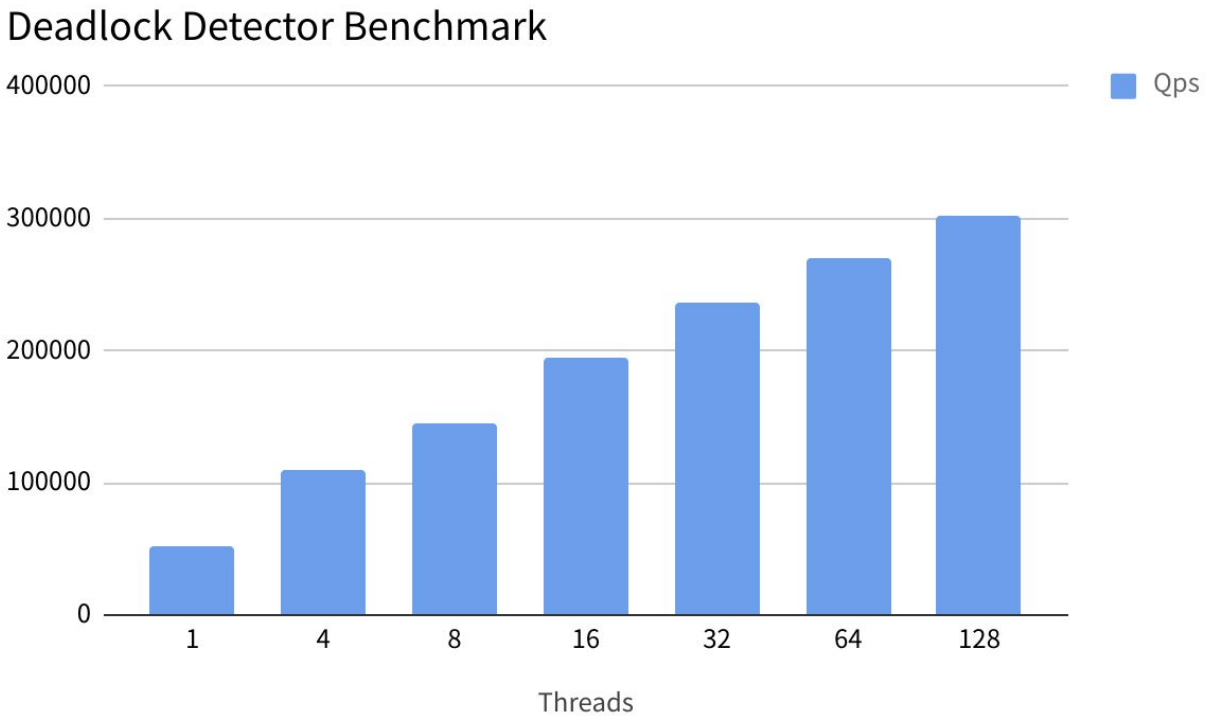
- 两条语句间可能发生死锁：
  - TiDB 按照 region 划分请求，并行加锁，无法控制加锁顺序。

- Percolator 的 Prewrite 有同样的问题。

单语句 rollback

- TiKV 返回给 TiDB 的 deadlock 错误中包含导致死锁的事务正在等待的 key 信息。
- 若 TiDB 发现该 key 为当前语句持有，会删除该语句持有的悲观锁，等待并重试。

死锁检测器性能


Deadlock Detector Benchmark
400000
300000
200000
100000
0
Qps
1
4
8
16
32
64
128
Threads

PingCAP

Part IV - 测试

测试

- TiDB 所有已有的集成测试
- TiDB 所有已有的薛定谔测试
- Jepsen 测试：纯乐观、纯悲观、乐观悲观混合定期测试。
- 新增的薛定谔测试与集成测试。

PingCAP

Part V - 使用

悲观事务使用方法

进入悲观事务模式有以下三种方式:

- 执行 `BEGIN PESSIMISTIC;` 语句开启的事务，会进入悲观事务模式。 可以通过写成注释的形式 `BEGIN /*!90000 PESSIMISTIC */;` 来兼容 MySQL 语法。
- 执行 `set @@tidb_txn_mode = 'pessimistic';` ，使这个 session 执行的所有显式事务（即非 autocommit 的事务）都会进入悲观事务模式。
- 执行 `set @@global.tidb_txn_mode = 'pessimistic';` ，使之后整个集群所有新创建的 session 都会进入悲观事务模式执行显式事务。

乐观事务

Session A	*Session B(no auto retry)*
> begin **optimistic**;	> begin optimistic;
> update test set v = v + 1 where k = 1;	**> update test set v = v + 1 where k = 1;**
> commit;	
	> commit; **ERROR 8005 (HY000): Write conflict, txnStartTS 410234613249343489 is stale [try again later]**

悲观事务

Session A	*Session B*
> begin pessimistic;	> begin pessimistic;
> update test set v = v + 1 where k = 1;	
	> update test set v = v + 1 where k = 1; **block...**
> commit;	**Query OK, 1 row affected (0.00 sec)**
	> commit; **Query OK, 0 row affected (0.00 sec)**

SELECT … FOR UPDATE

Session A	*Session B*
> begin pessimistic;	
> select * from test;(SI) +------+------+ | k | v | +------+------+ | 1 | 1 | +------+------+	
	> update test set **v = v + 1** where k = 1;
> select * from test for update;(RC) +------+------+ | k | v | +------+------+ | 1 | 2 | +------+------+	

DEADLOCK

Session A	*Session B*
> begin pessimistic;	> begin pessimistic;
> update test set v = 2 where k = 1;	**>** update test set v = 1 where k = 2;
> update test set v = 1 where k = 2;	
	> update test set v = 2 where k = 1; **ERROR 1213 (40001): Deadlock found when trying to get lock; try restarting transaction**
> commit;	

PingCAP

Thank You!