Two Phase Commit in TiDB

発表者: Yilin Chen

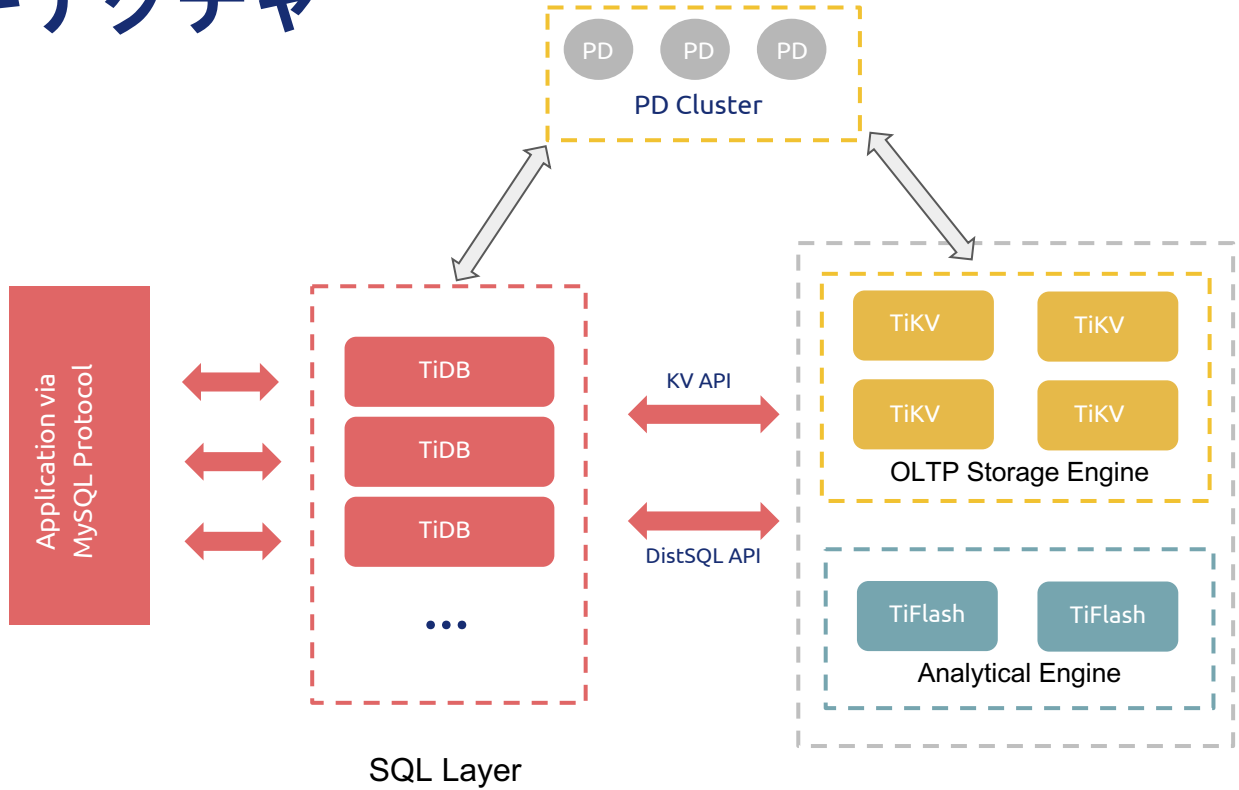
PingCAP

Part I - TiDBについて

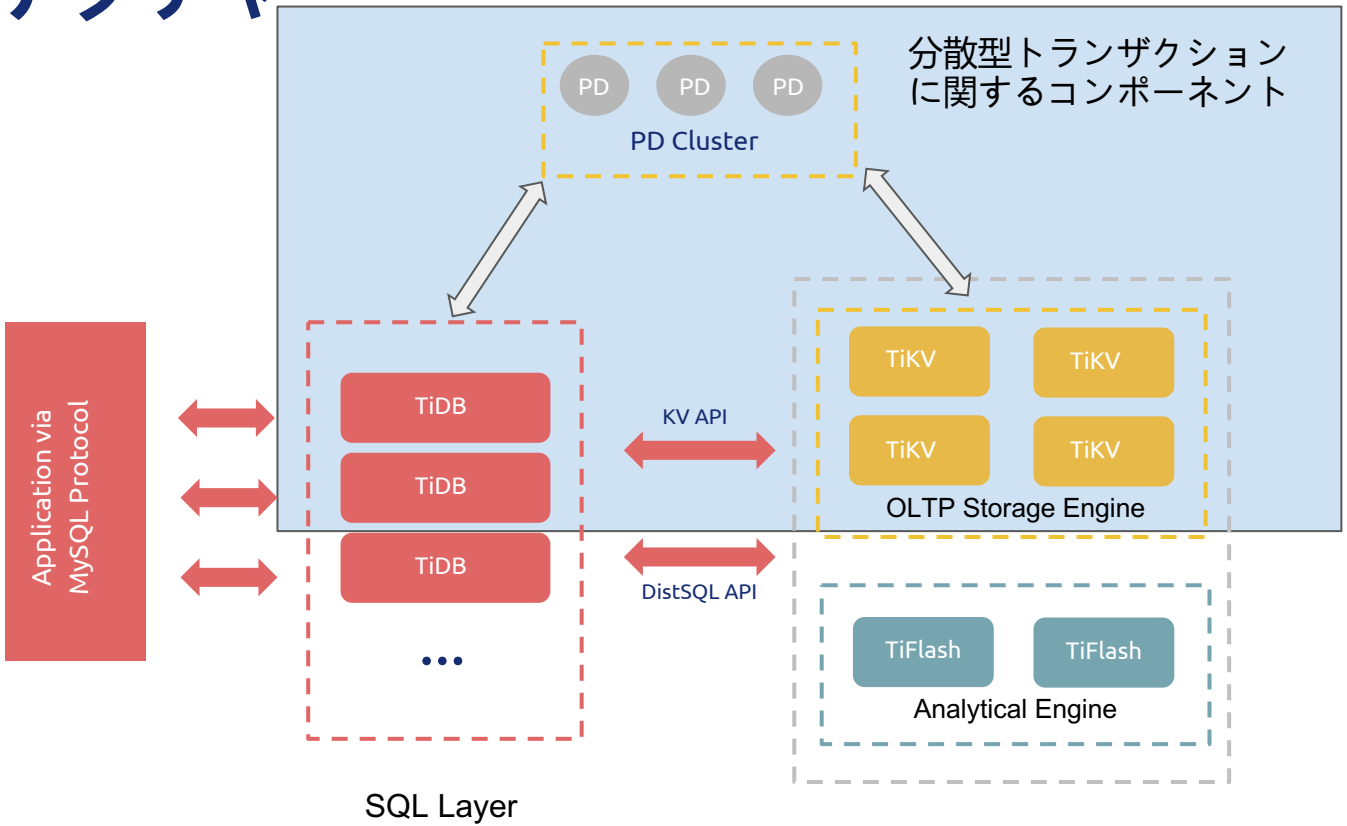
TiDB Highlights

- 水平方向のスケーラビリティ
- 高可用性
- 強固な一貫性
- MySQL互換性
- HTAP

アーキテクチャ


PD
PD
PD
PD Cluster
Application via MySQL Protocol
TiDB
TiDB
TiDB
...
SQL Layer
KV API
DistSQL API
TiKV
TiKV
TiKV
TiKV
OLTP Storage Engine
TiFlash
TiFlash
Analytical Engine

アーキテクチャ


分散型トランザクション
に関するコンポーネント
PD
PD
PD
PD Cluster
Application via
MySQL Protocol
TiDB
TiDB
TiDB
...
SQL Layer
KV API
DistSQL API
TiKV
TiKV
TiKV
TiKV
OLTP Storage Engine
TiFlash
TiFlash
Analytical Engine

TiKV Overview

- 分散KVストレージ
- CNCFの卒業プロジェクト
- 高可用性
- 高いスケーラビリティ
- トランザクションのサポート

Table Data Encoding

```
CREATE TABLE user_table (
        id INT,
        name VARCHAR(64),
        email VARCHAR(1024),
        PRIMARY KEY(id)
);
```

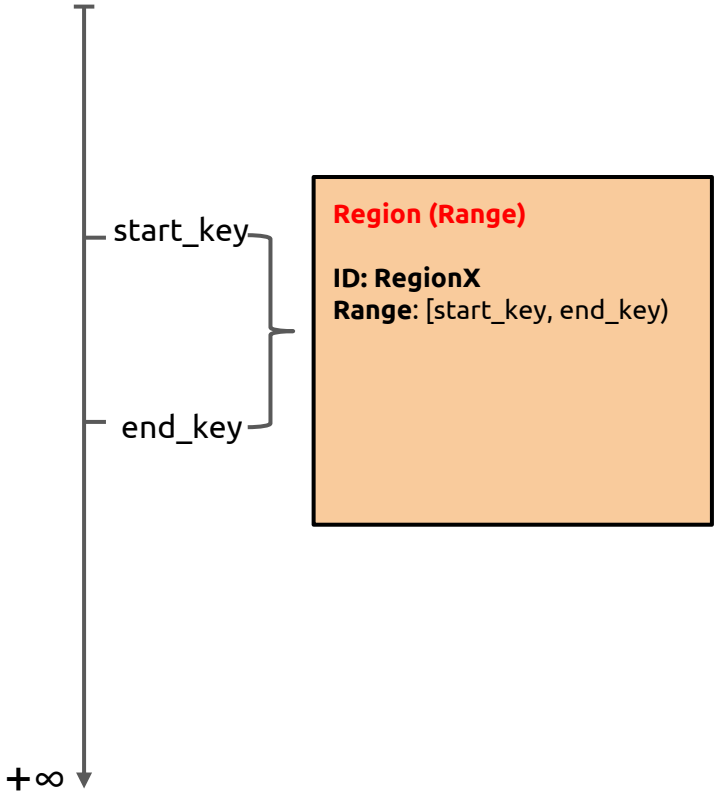
user_table		
1	dongxu	huang@pingcap.com
2	foo	bar@pingcap.com
...	...	...

Within TiKV:

t101_r1 => [1,dongxu,huang@pingcap.com]
t101_r2 => [2, foo, bar@pingcap.com]
t101_r... => ...

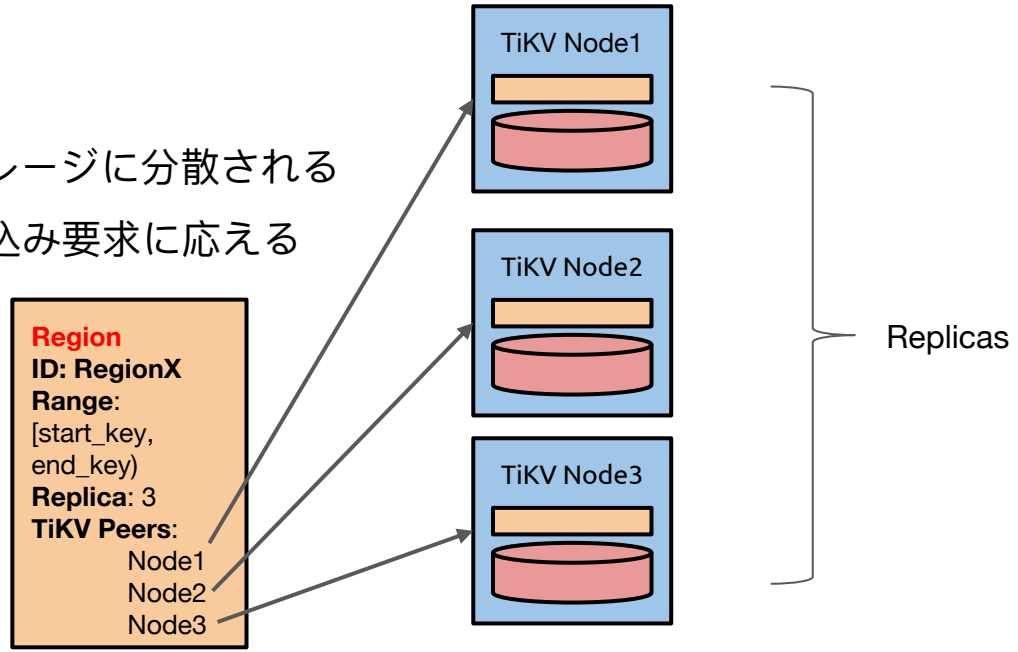
TiKVデータの構成

- Sorted Key-Value Map
  - キーと値の両方がバイト配列
  - キーはバイトオーダーでソートされる
- キー空間は分割されている
  - データは "Region "と呼ばれるチャンクに分割される

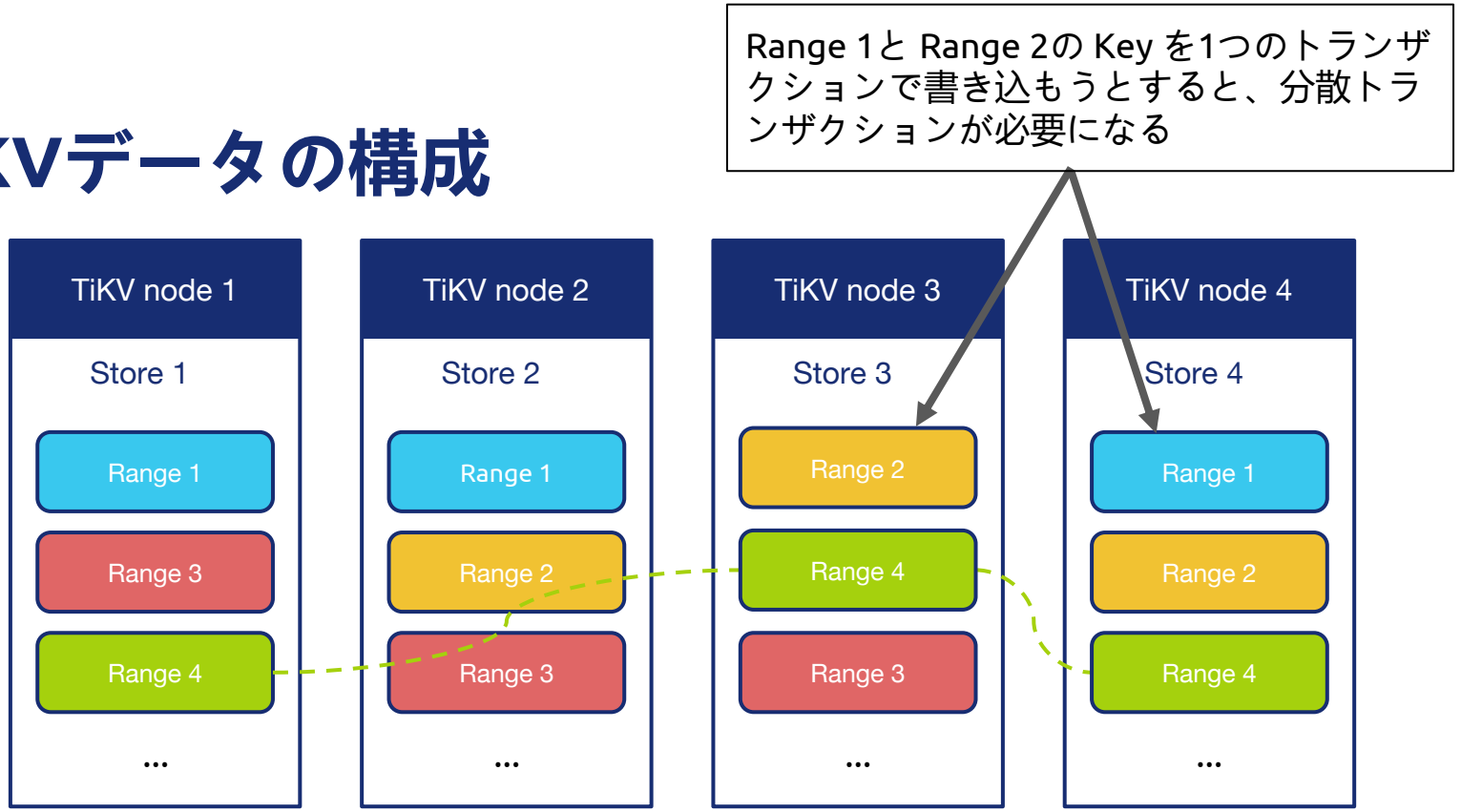


TiKVデータの構成

- 各 ***region*** は複数のレプリカを持つ
- 強い一貫性が保証されたレプリカ
- レプリカはRaftプロトコルより各ストレージに分散される
- ラフトグループのリーダーのみが書き込み要求に応える



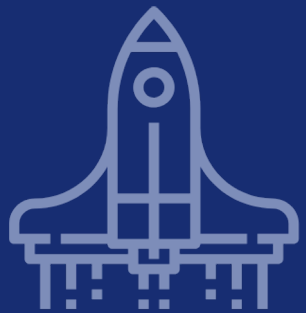
TiKVデータの構成



各データRangeは一つRaftグループ

PingCAP

Part II - Basic Percolator



Google Percolator

- Google BigTableのトランザクションモデル
- MVCCによるSnapshot Isolation
- 単一行トランザクションに基づいて複数行トランザクションを実施できる
- “Almost” **decentralized** 2PC
    - 中央のコーディネータがいない
    - The only single point is the timestamp allocator (**TSO**)
    - 高可用性を確保しやすい

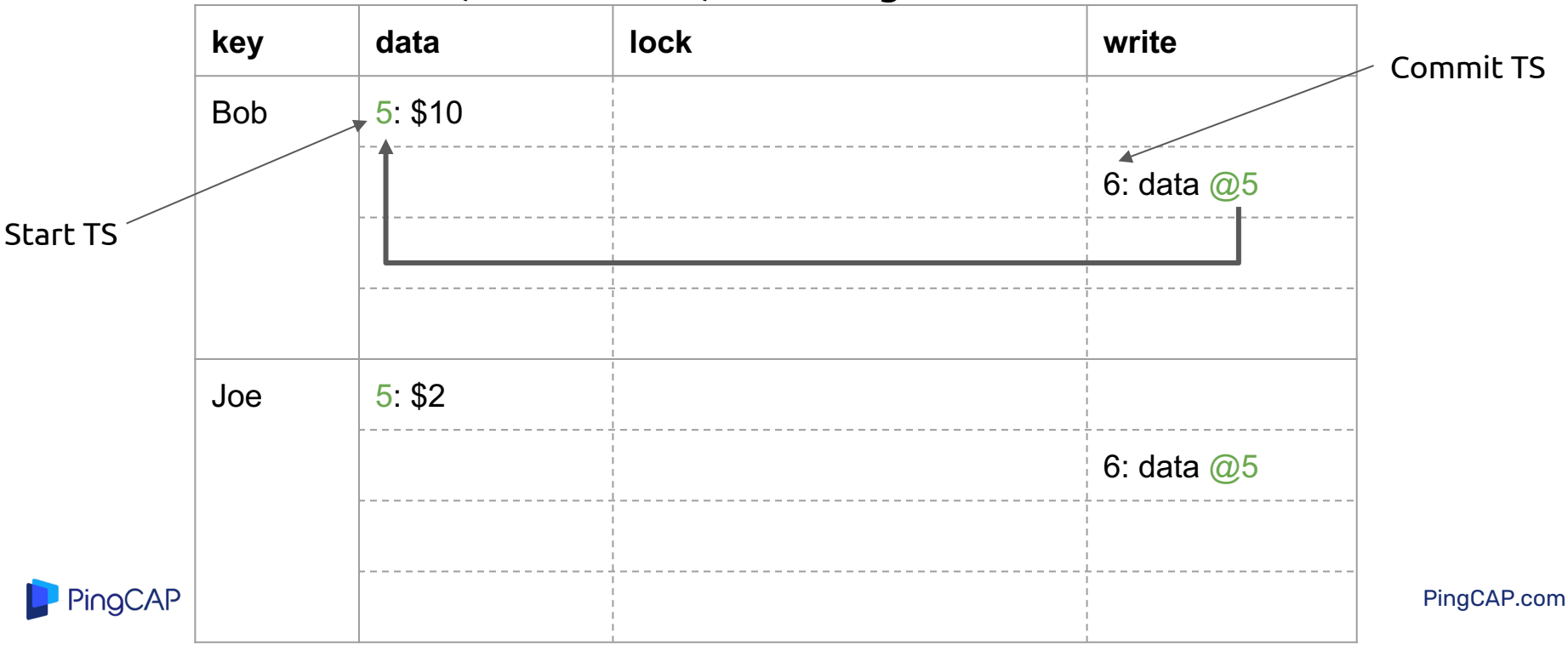
Schema

Column	Use
lock	Indicates an uncommitted transaction is writing
write	Committed data present
data	Store the data itself

Percolator Demo

各Keyは複数のバージョンを持つことができる

Bob have $10, Joe have $2, Bob will give Joe $7.



Phase 1: Prewrite

Select any key as primary. Write data and locks.

New Start TS from the TSO

key	data	lock	write
Bob	5: $10		
			6: data @5
	7: $3	7: I'm primary	
Joe	5: $2		
			6: data @5
	7: $9	7: primary @ Bob	

Phase 2: Commit Primary Lock

Remove the primary lock and add a write record.

key	data	lock	write
Bob	5: $10		
			6: data @5
	7:$3	~~7: I'm primary~~	
			8: data @7
Joe	5: $2		
			6: data @5
	7:$9	7: primary @ Bob	

New Commit TS from the TSO

Phase 2: Commit Primary Lock

Remove the primary lock and add a write record.

key	data	lock	write
Bob	5: $10		
			6: data @5
	7:$3	~~7: I'm primary~~	
			8: data @7
Joe	5: $2		
			5
	7:$	~~7: primary @ Bob~~	

プライマリロックがコミットされると、そのトランザクションはコミットされたとみなされます。

New commit TS from the TSO

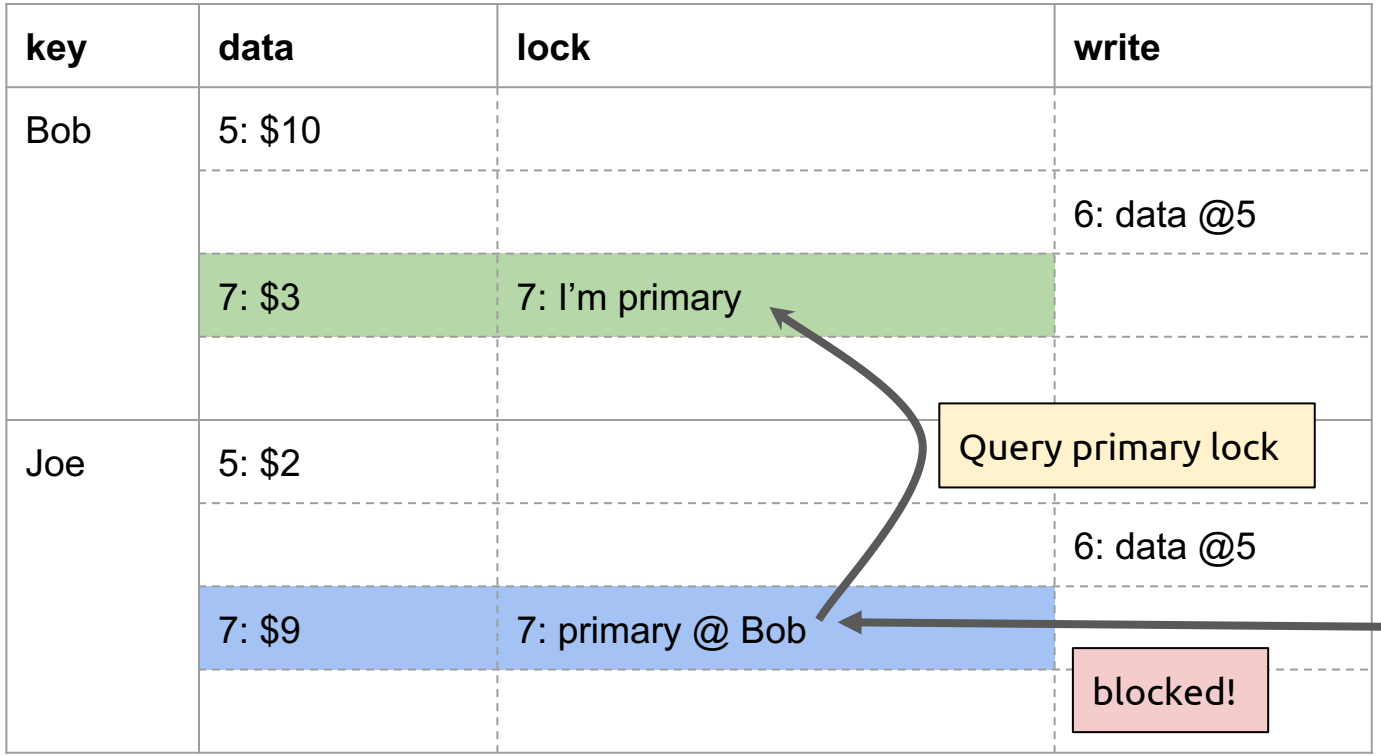
Phase 2: Commit Secondary Locks (*Async*)

Remove secondary locks and add write records for them.

key	data	lock	write
Bob	5: $10		
			6: data @5
	7:$3	~~7: I'm primary~~	
			8: data @7
Joe	5: $2		
			6: data @5
	7:$9	~~7: primary @ Bob~~	
			8: data @7

Encounter Lock -> Blocking

Read must be blocked if it encounters a lock with smaller TS



Why Blocking

Otherwise it may read stale data and may also break snapshot isolation.

key	data	lock	write
Bob	5: $10		
			6: data @5
	7: $3	7: I'm primary	
			8: data @7
Joe	5: $2		
			6: data @5
	7: $9	7: primary @ Bob	
			8: data @7

Read Joe @10

Encounter Lock -> Remove Lock

The blocking lock can be removed. And it means the transaction is aborted.

key	data	lock	write
Bob	5: $10		
			6: data @5
	~~7: $3~~	~~7: I'm primary~~	7: rollback
Joe	5: $2		
			6: data @5
	~~7: $9~~	~~7: primary @ Bob~~	7: rollback

Read Joe @10

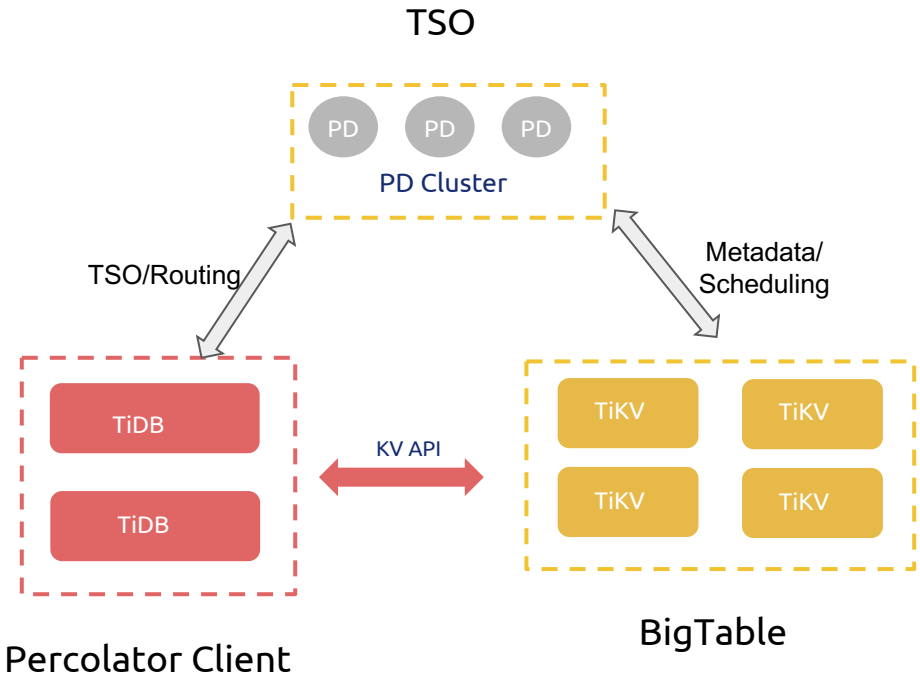
Encounter Lock

- ロックライターがある場合:
  - ライターがロックをコミットするまでリーダーをブロックする
- ロックwriterがダウンし、ロックをコミットできなくなった場合
  - Reader が読み続けられるようにロックを外す (トランザクションの中止を意味する) 。

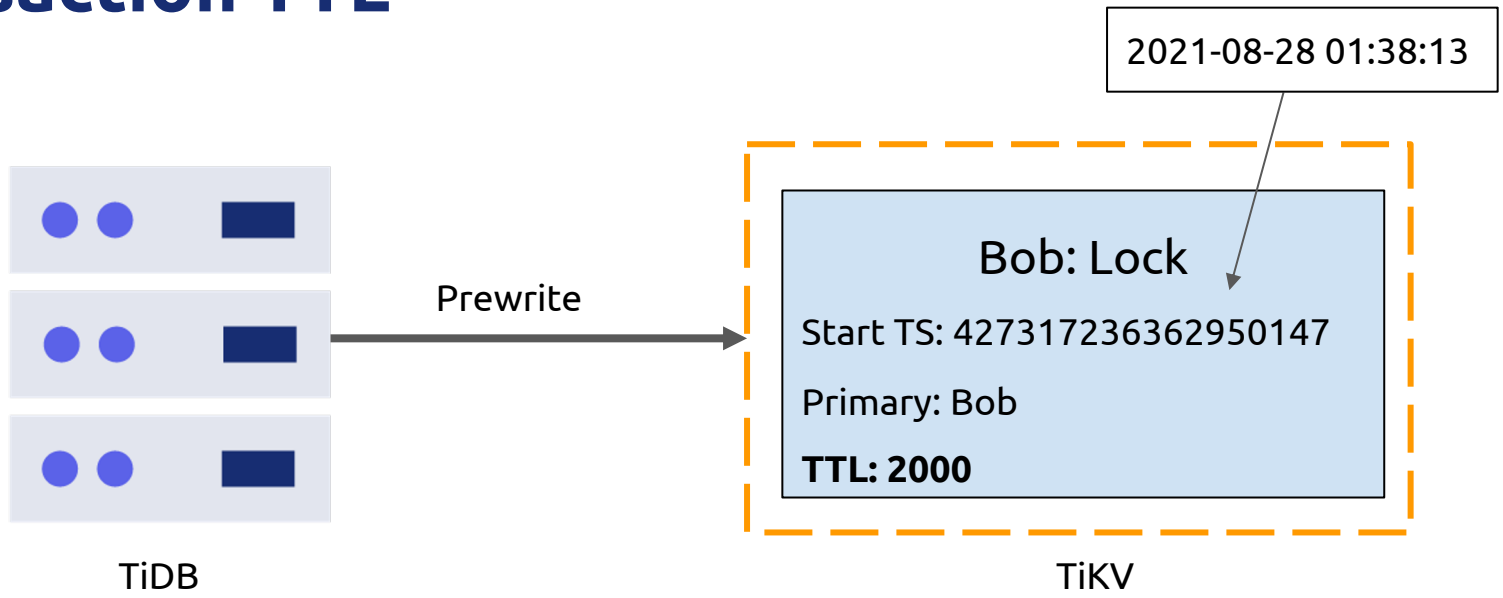
PingCAP

Part III - Percolator in TiDB

Percolator Concept Mapping

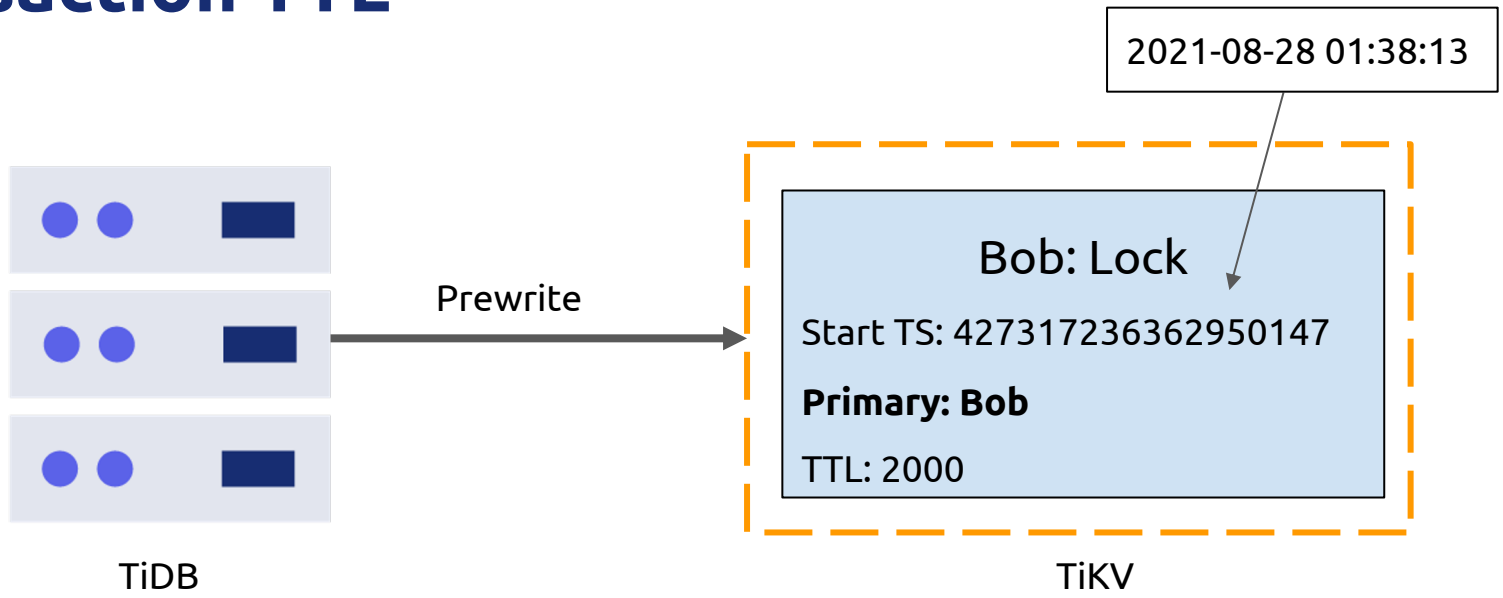


Transaction TTL



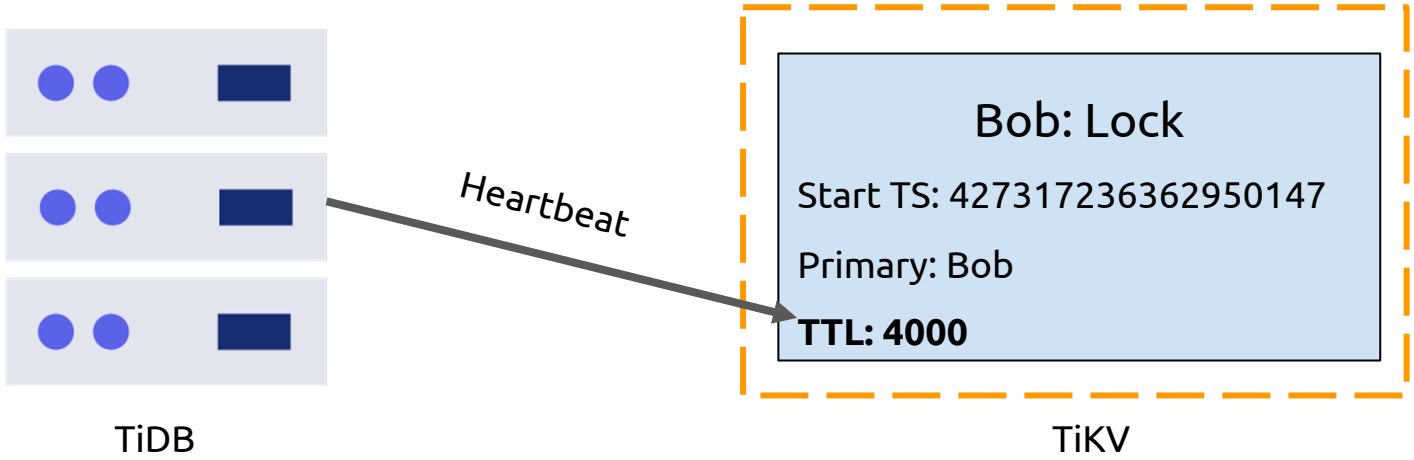
Start TS + TTL (ms) > Current TS => Expire (Writer is dead)

Transaction TTL



主キーのTTLは、トランザクションのTTLを表す

Transaction TTL: Heartbeat

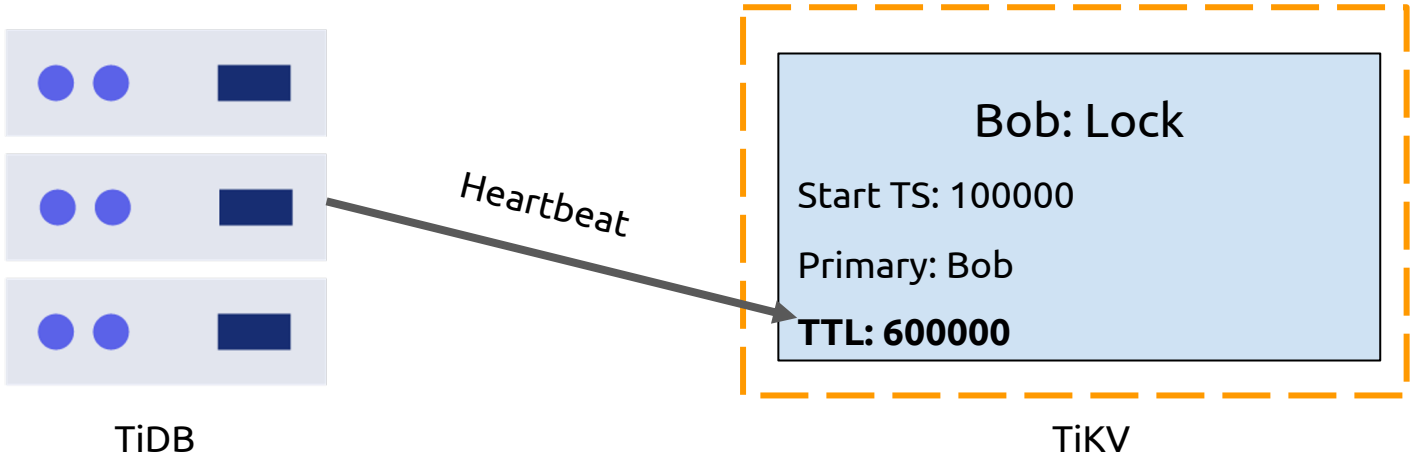


HeartbeatsでTTLを更新するので、トランザクションがいきなりに中止されることはない

PingCAP

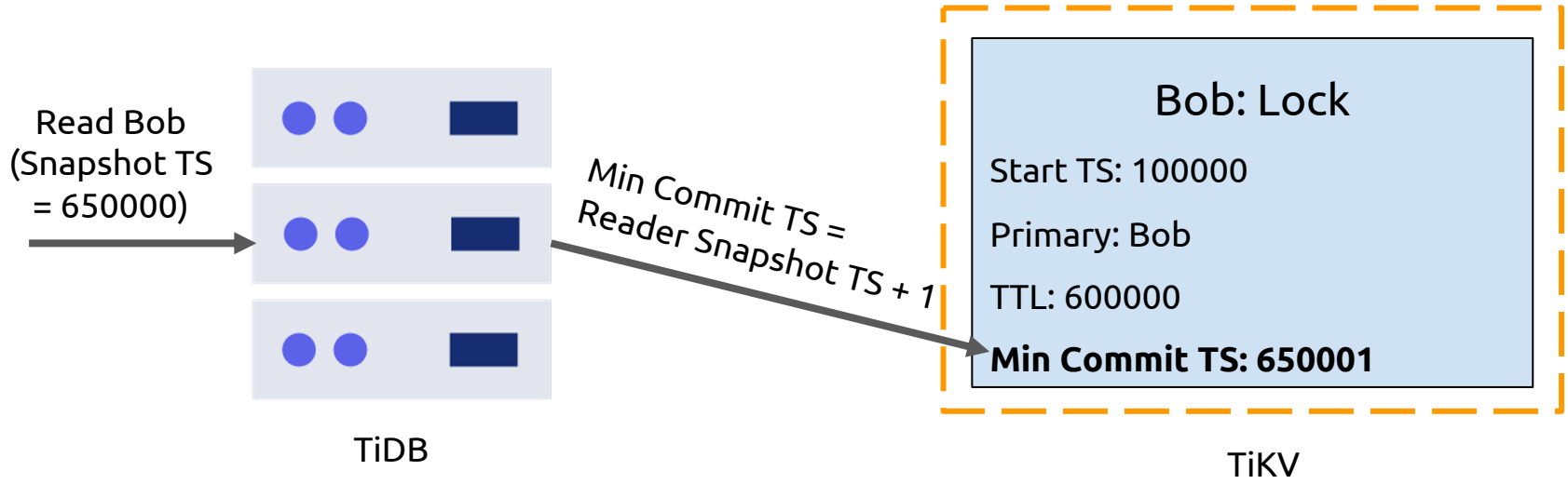
Part IV - Optimizations

Blocking reader?



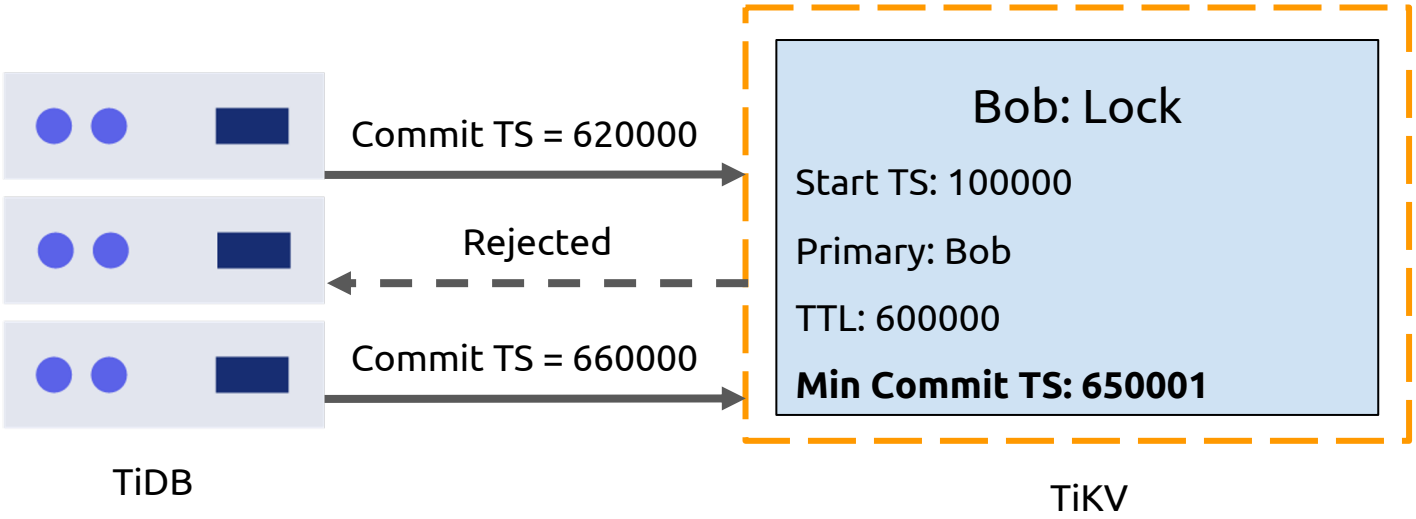
フェーズ1に時間がかかりすぎると、readerが長時間ブロックされてしまうのでは？

Optimization: Min Commit TS



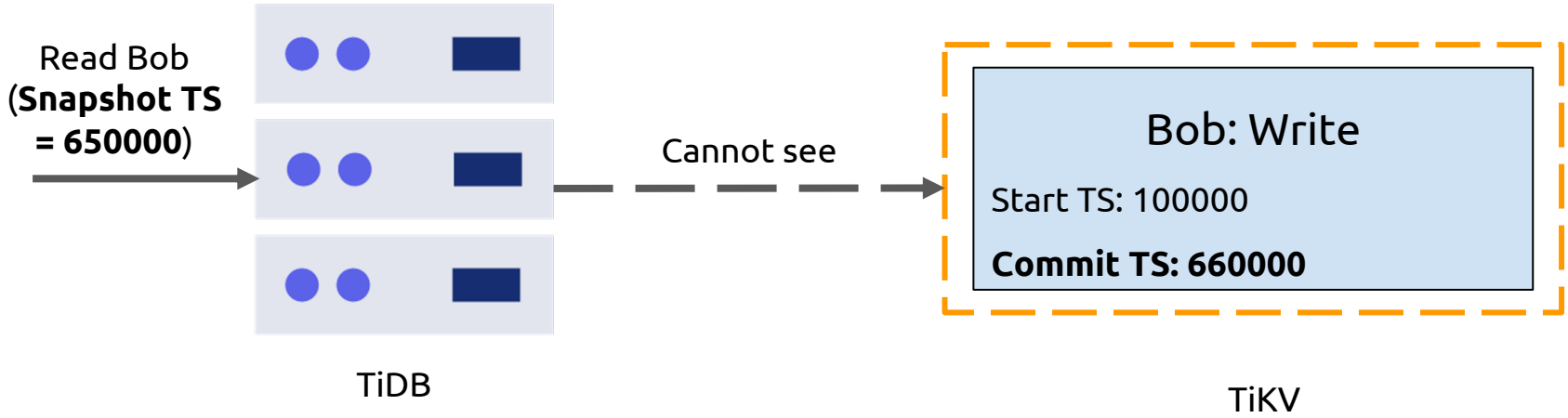
ロックに遭遇した場合、リーダーはMin Commit TSをスナップショットTS + 1に設定し、ロックを無視することができる

Optimization: Min Commit TS



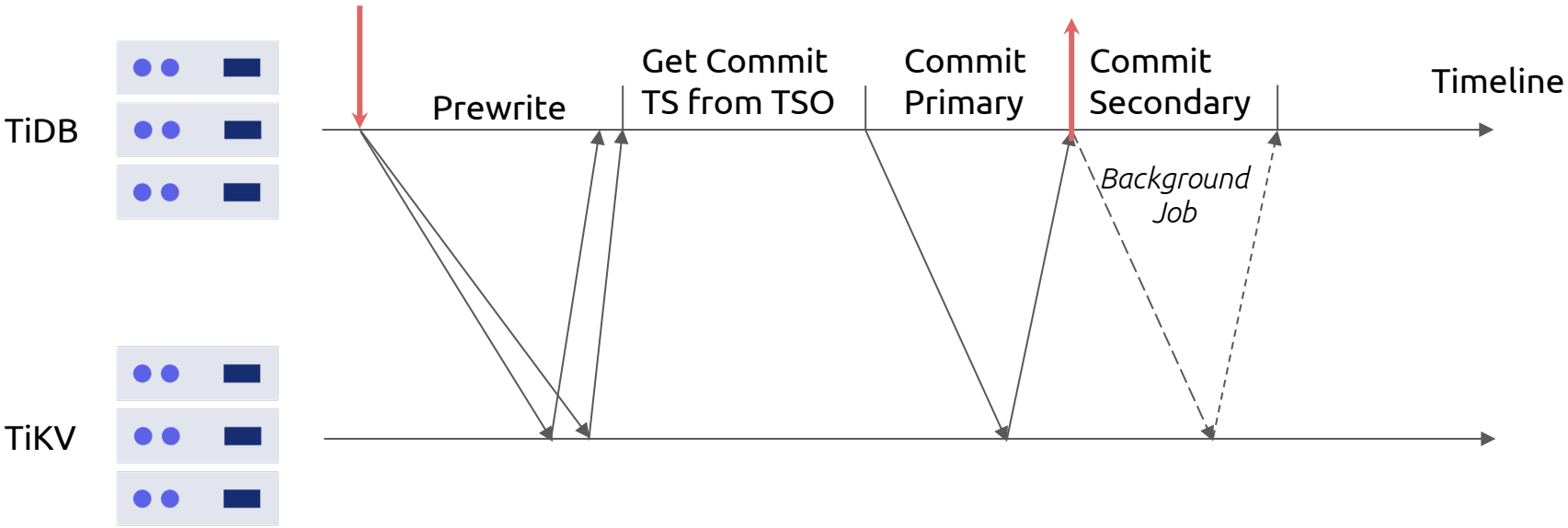
フェーズ2でCommit TS < Min Commit TSの場合、Commitは拒否される
TiDBはTSOから新しいコミットTSを取得して再試行する必要がある

Optimization: Min Commit TS



Commit TSはスナップショットTSよりも大きいので、新しくコミットされたレコードはリーダーのスナップショットを壊すことはない

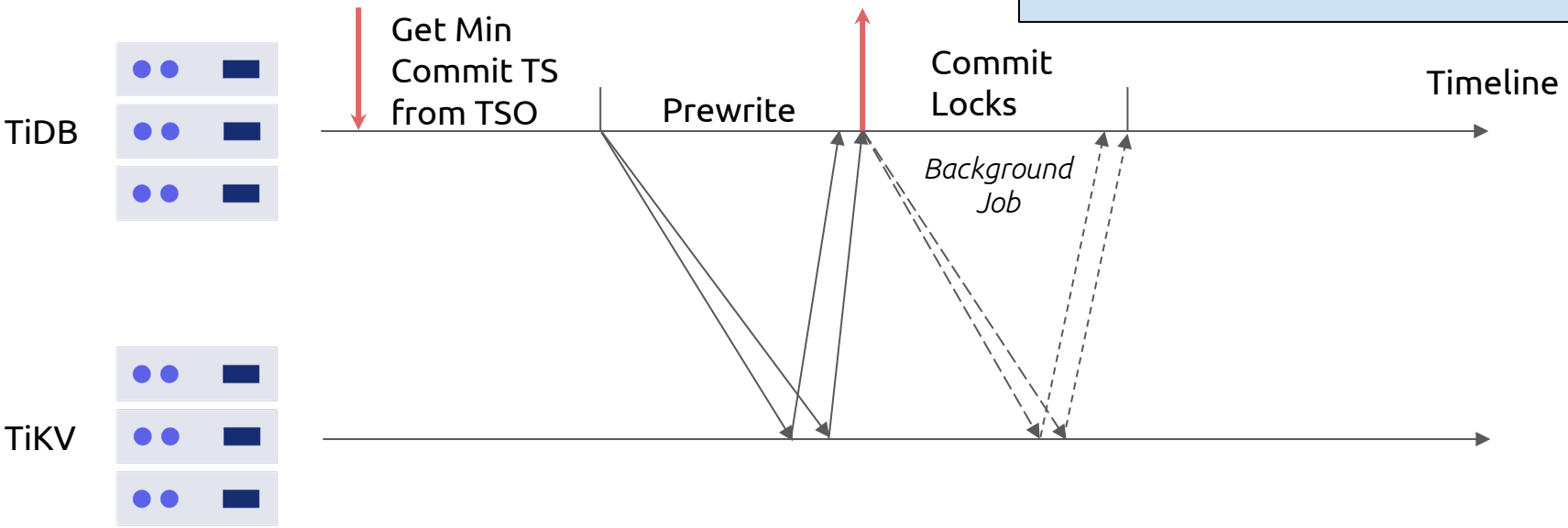
High latency



Percolator 2PCには、TiKVに対する少なくとも2つのRPCのレイテンシーが含まれる
各 RPC は Raft を通して複製する必要のある書き込み操作です。

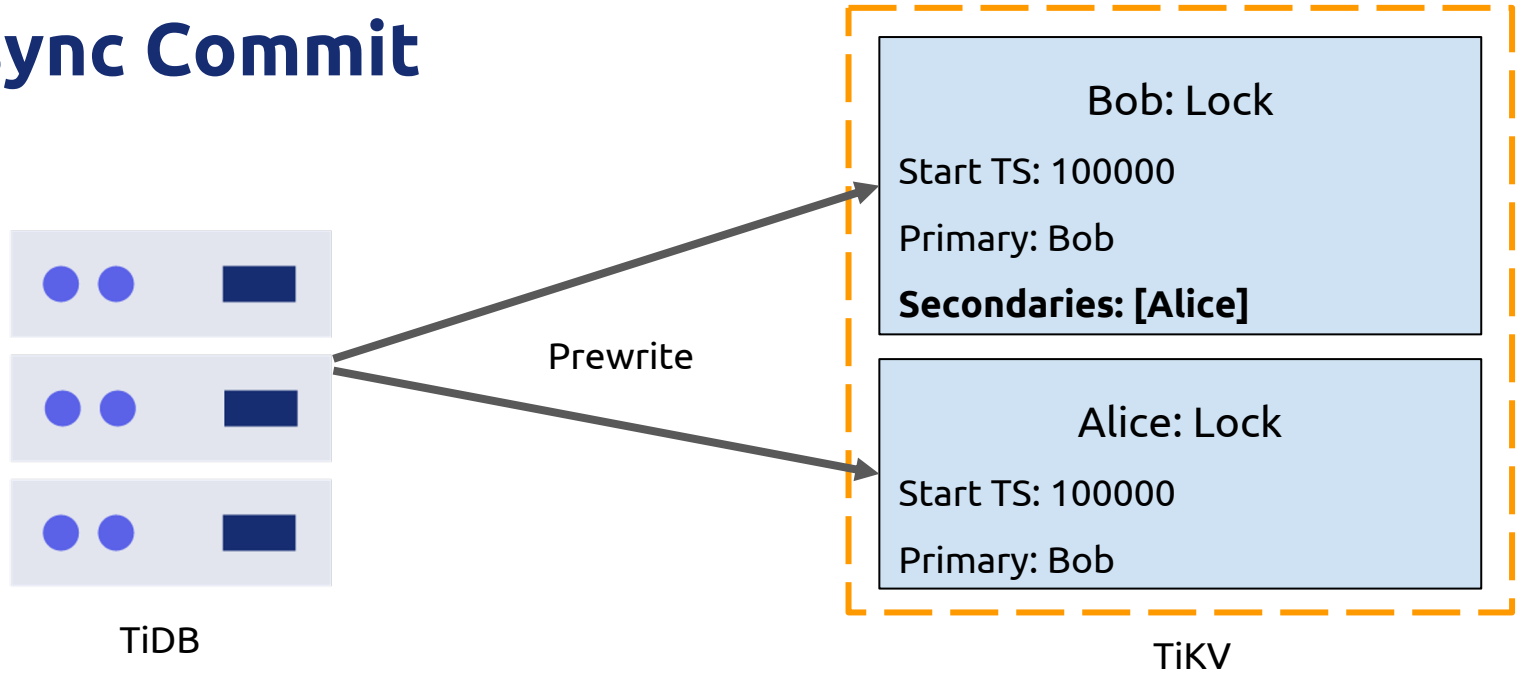
Optimization: Async Commit

Once all keys are prewritten successfully, the transaction is committed.



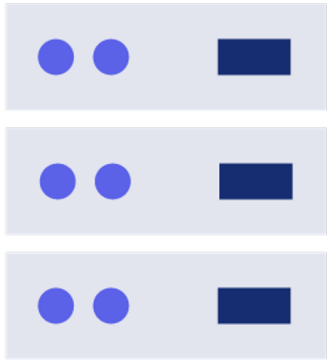
- 非同期コミットは、フェーズ2全体をバックグラウンドジョブにする
- TiKV RPCの1回分 (約40%) でクライアントのレイテンシは短縮されている

Async Commit



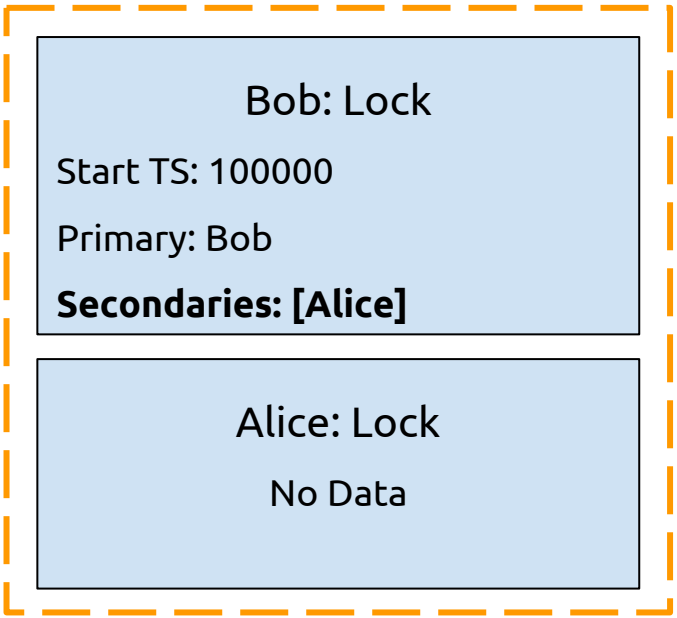
主な変更点は，プライマリロックにキーリストを追加したことです

Async Commit: Recovery



TiDB

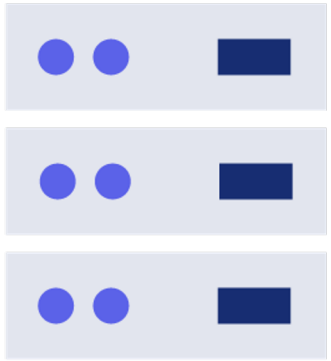
1. Encounter the lock of Bob
2. Bob is primary, check if all secondary locks exist
3. Alice does not exist, roll back all the locks



TiKV

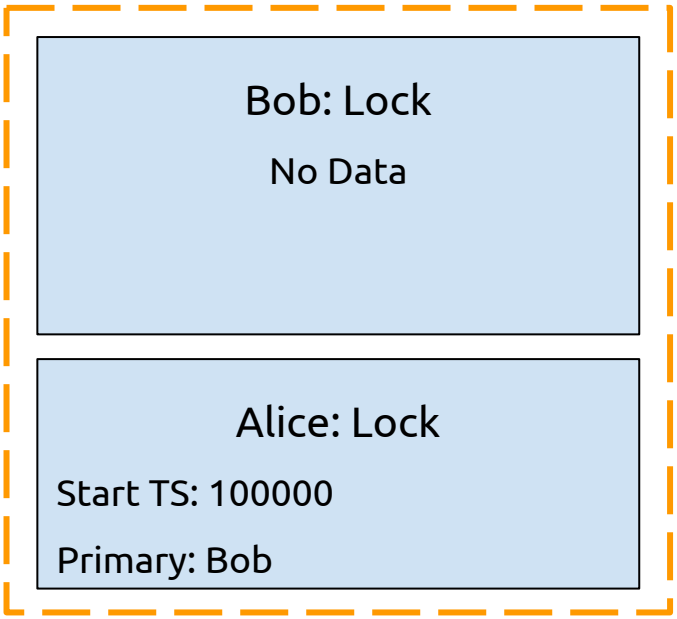
Secondary lockが存在しない場合は、トランザクションを中止する

Async Commit: Recovery



TiDB

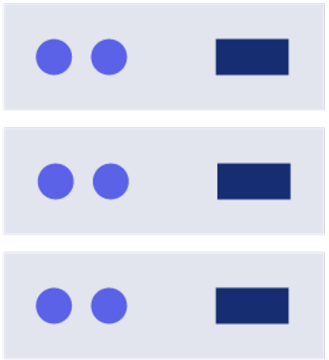
1. Encounter the lock of Alice
2. Check the primary lock Bob
3. Lock of bob does not exist, roll back all the locks



TiKV

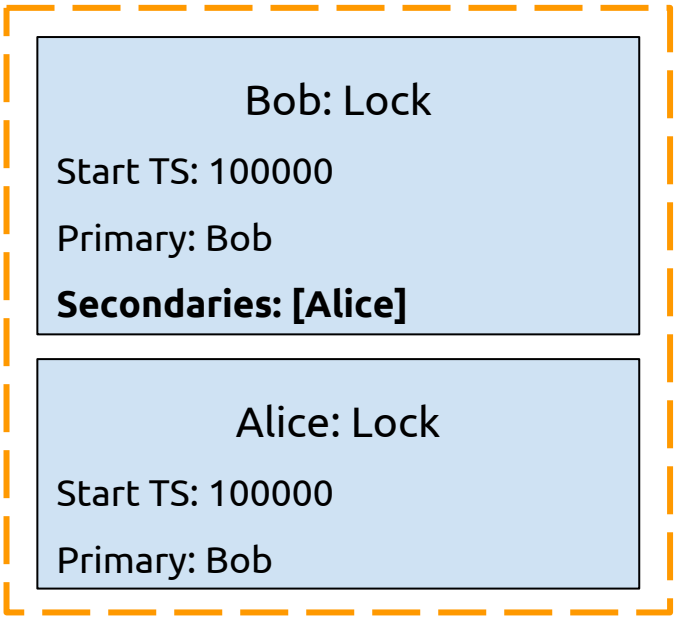
Primary lockが存在しない場合は、トランザクションを中止することができる

Async Commit: Recovery



TiDB

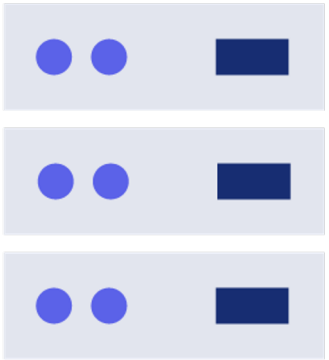
1. Encounter the lock of Alice
2. Check the primary lock Bob
3. Check if all secondary locks exist
4. Commit the locks



TiKV

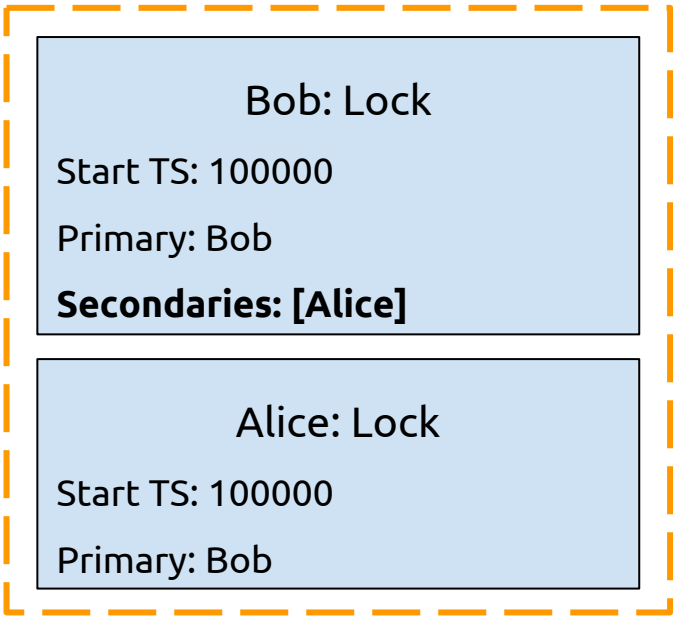
いずれかのロックをコミットする前にTiDBがクラッシュした場合、
他のTiDBは、リストを通じてすべてのロックが存在するかどうかを
確認し、フェーズ2を続行することができる

Async Commit: Recovery



TiDB

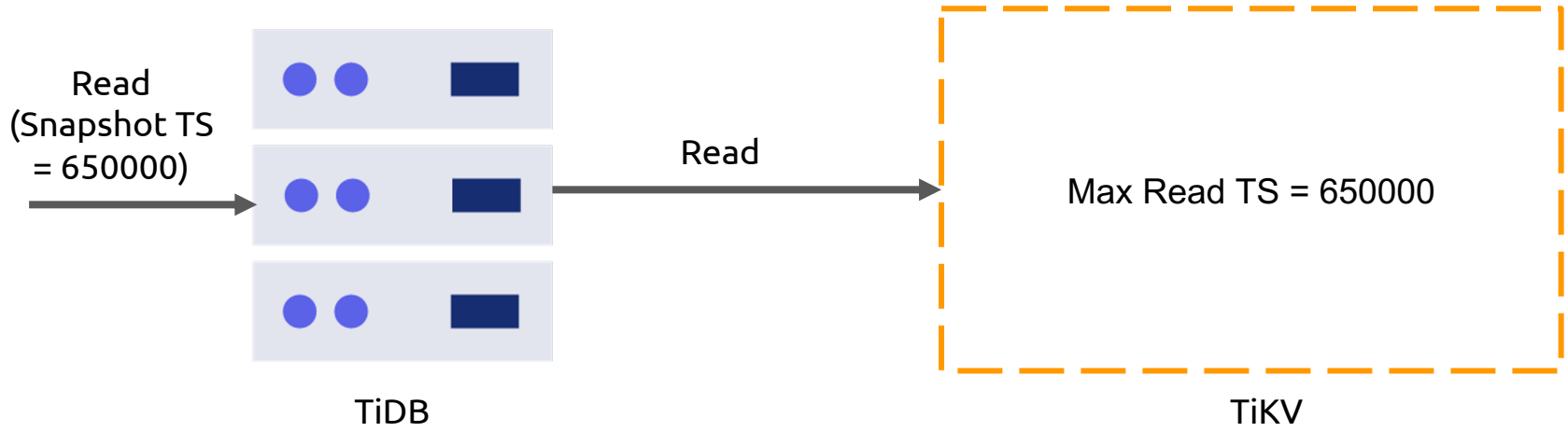
1. Encounter the lock of Alice
2. Check the primary lock Bob
3. Check if all secondary locks exist
4. Commit the locks



TiKV

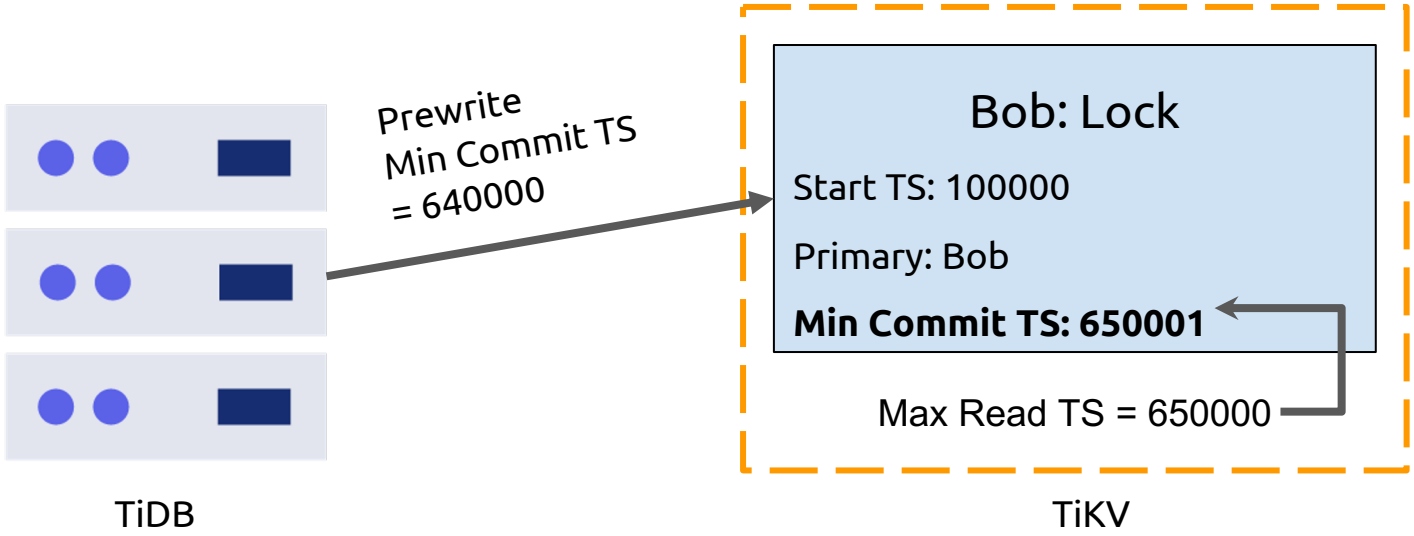
しかし、コミットTSはどうやって知ることができるのでしょうか？

Async Commit: Calculate Commit TS



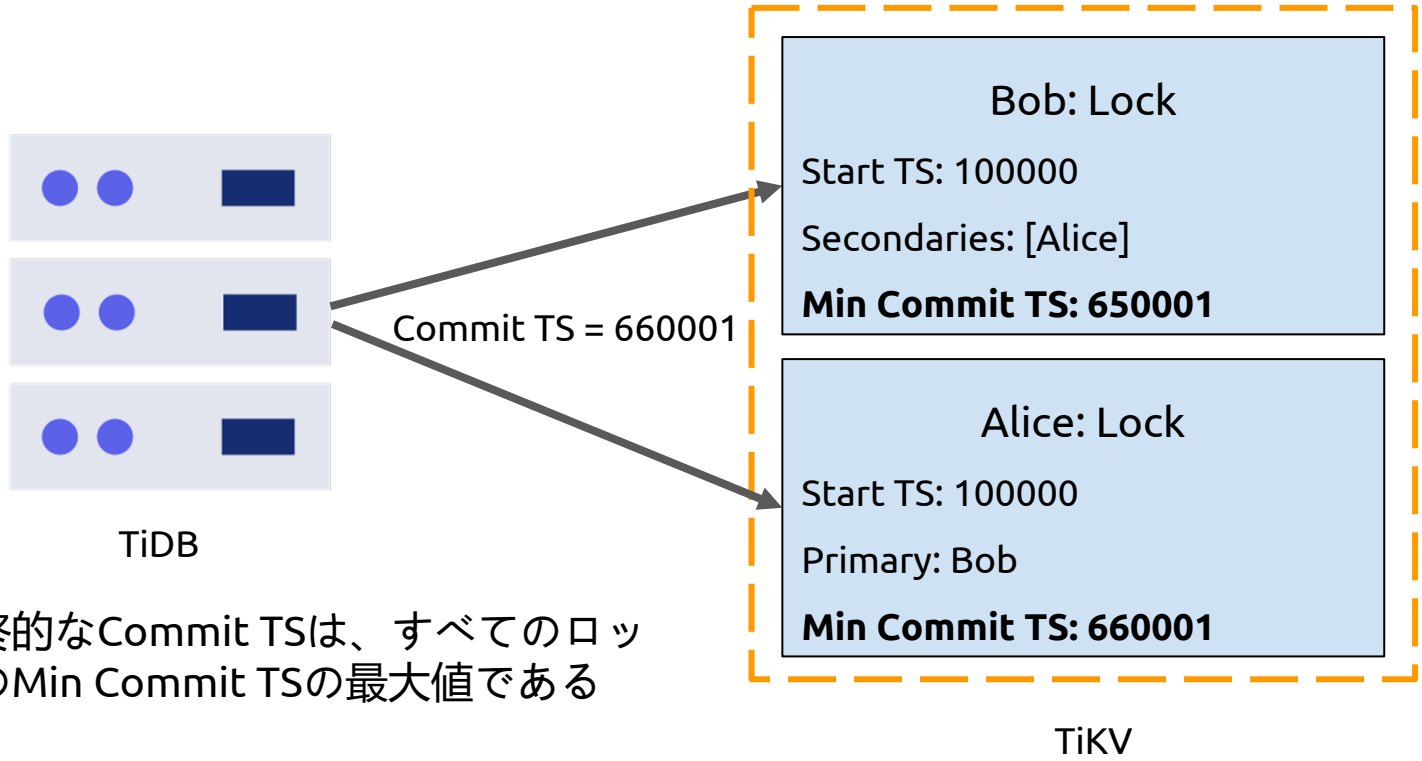
TiDBからの読み込みごとに、TiKVのMax Read TSが更新される

Async Commit: Calculate Commit TS



事前に書き込まれたロックのMin Commit TSは、TSOからの(Max Read TS + 1) とMin Commit TSの両方よりも大きい。

Async Commit: Calculate Commit TS



最終的なCommit TSは、すべてのロックのMin Commit TSの最大値である

Async Commit: Calculate Commit TS

- Commit TS > all readers' snapshot TS
  - ➢ 新しいcommitが前のreadersのスナップショットを壊さない (Snapshot Isolation)
- Commit TS does not exceed latest TS + 1
  - ➢ Commitは常に次の新しいスナップショットに表示される(Realtime)
- Commit TS > latest TS at the time of prewrite
  - ➢ 順序が逆にならない (Sequential)

PingCAP

Welcome

PingCAP株式会社募集中。分散システムにご趣味を持つ方は是非とも下記のメールとご連絡ください。

career@pingcap.co.jp

ご静聴ありがとうございます

PingCAP

Q & A