ここがつらいよ分散SQLデータベース

Yoshitaka Koitabashi
Sr. Solution Architect
PingCAP株式会社

自己紹介

Yoshitaka KOITABASHI

PingCAP株式会社
Sr. Solution Architect
💙 Database / Serverless / Container

JAWS-UG
AWS User Group - Japan

Agenda

- PingCAPとは？
- 分散型SQLデータベース
- TiDBの裏側

  (アーキテクチャ/MVCC/分散トランザクション/Raft/ etc...)
- 実際の運用現場で直面する課題

PingCAPとは？

会社紹介

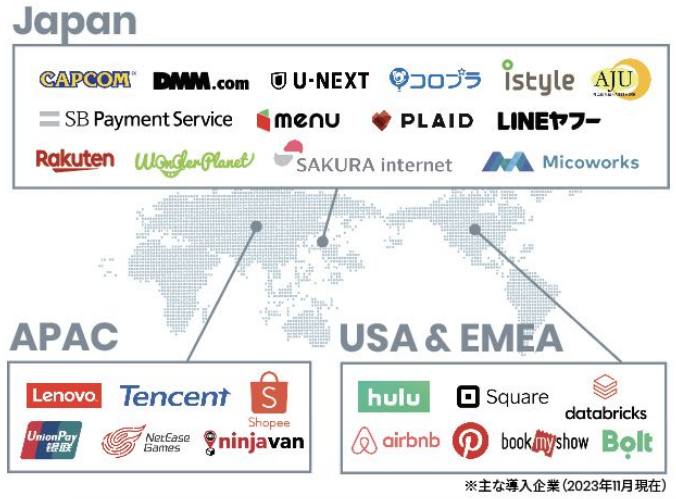
分散型SQLデータベース TiDBを継続的に開発

- 2015年設立後、OSSとしてTiDBを開発
- 2017年には商用版もリリースし、毎年メジャーアップデート
- フルマネージド型DBサービス「TiDB Cloud」の展開を強化

ワールドワイドでビジネス展開

- 800億以上の資金調達に成功し、積極的な事業展開
- **本社：米国カリフォルニア州**
- 2021年4月に日本支社設立
- **設立目的**
  1.日本のお客様の保守体制の構築
  2.日本市場の開拓
- 日本法人所在地
  東京都千代田区大手町2丁目6-4
  TOKYO TORCH 常盤橋タワー

TiDBはグローバルで3000社以上で採用



※主な導入企業（2023年11月現在）

活発な開発コミュニティを持つ、成熟した OSS

32,000+
Stars

3,000+
Large Scale Adoptors

1,200+
Ecosystem Contributors

2
CNCF Hosted Projects
TiKV, Chaos-Mesh

CLOUD NATIVE COMPUTING FOUNDATION

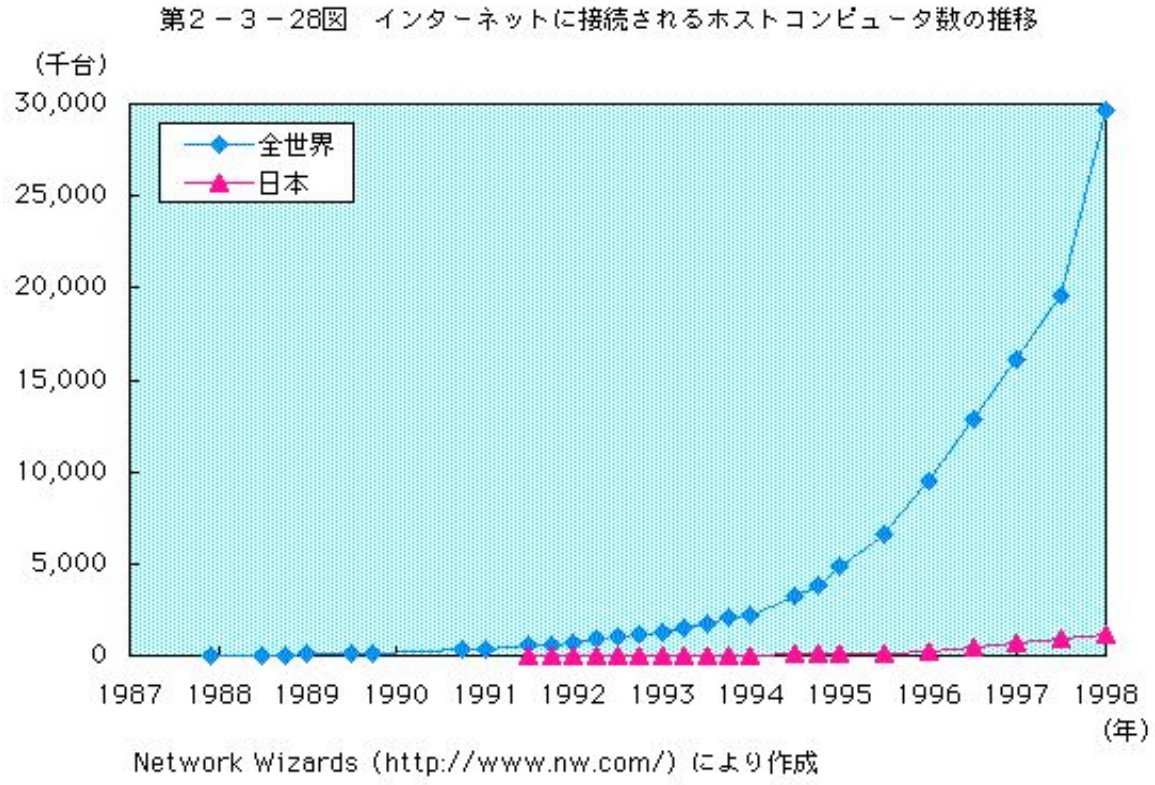
分散型SQLデータベース
≒ NewSQL

スケーラビリティが求められる背景

- 1990年以降、インターネット人口が急増
- 書き込みの負荷に対しマシン性能が必要
  => NoSQLが生まれる

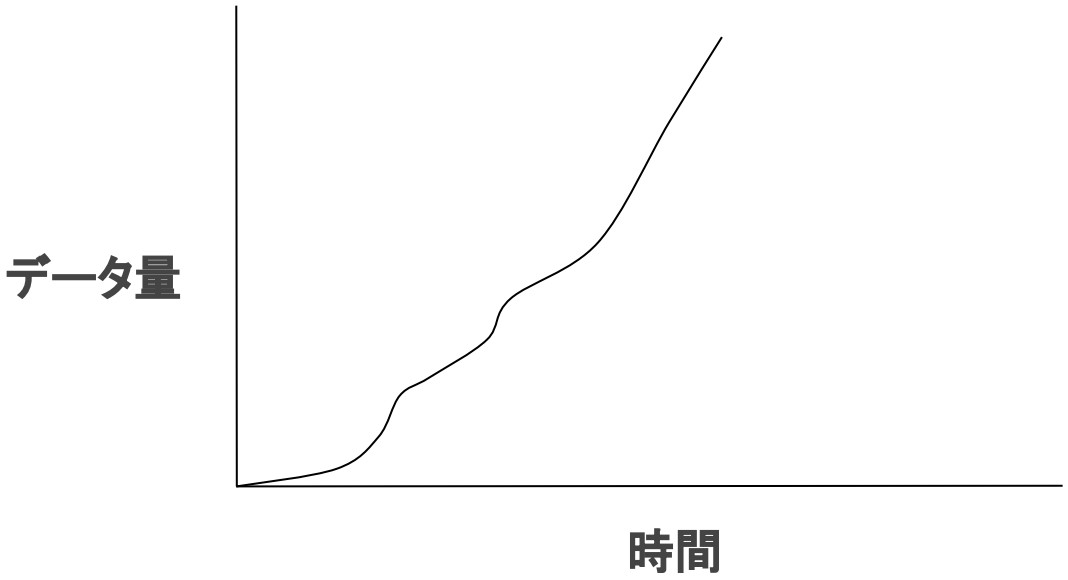
- 2006年にGoogleが論文を発表
  ”A Distributed Storage System for Structured Data”
  => カラム指向 DB: “BigTable”

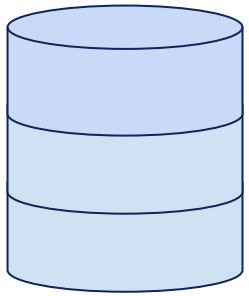
- 2008年にFacebookが”Cassandra”を発表
- 2012年にAWSが”DynamoDB”をLaunch



https://www.soumu.go.jp/johotsusintokei/whitepaper/ja/h10/html/98wp2-3-1f.html

データベース技術の変化





取り扱う
データの種類

求められる
クエリの性質

データベース技術の発展

データモデル
リレーショナル
オブジェクト
Key-Value
ドキュメント
グラフ

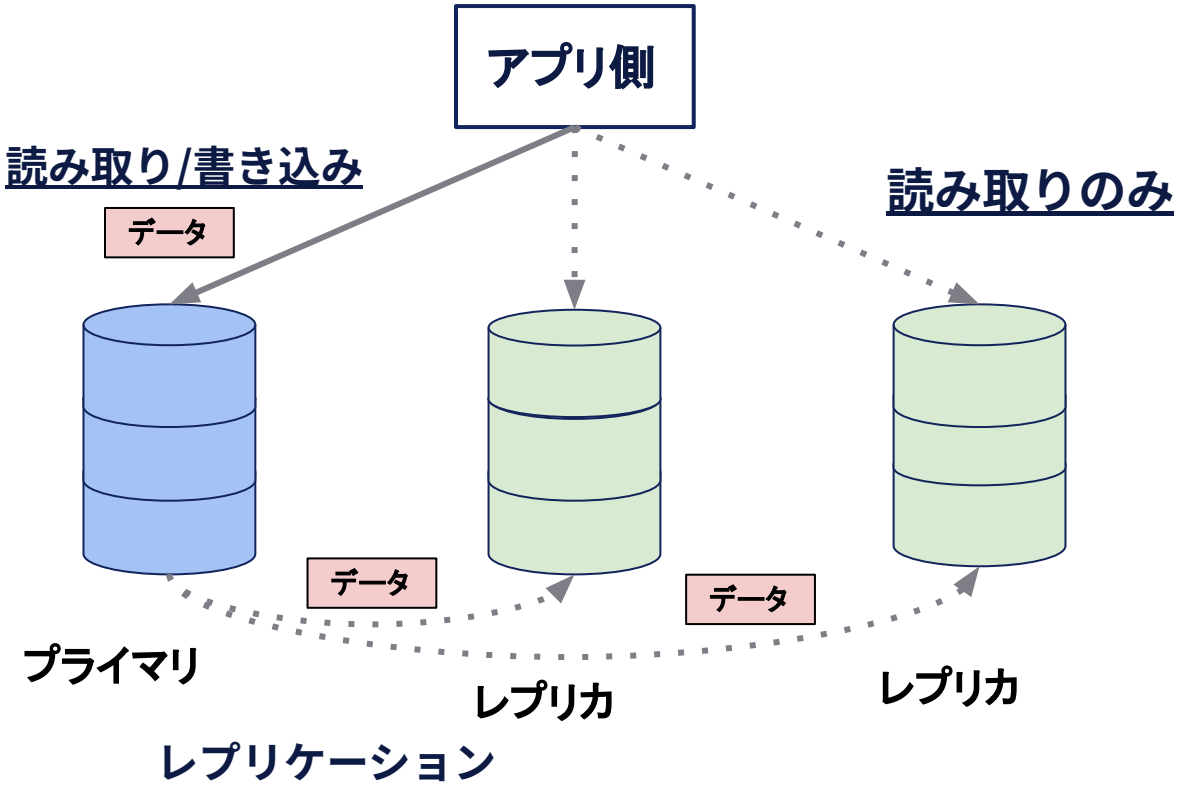
ストレージ
HDD
SSD
NVMe

データ構造
B+Tree
LSM Tree
Bitcask
Hash Table
Bitmap Index
AVL Tree
Graph

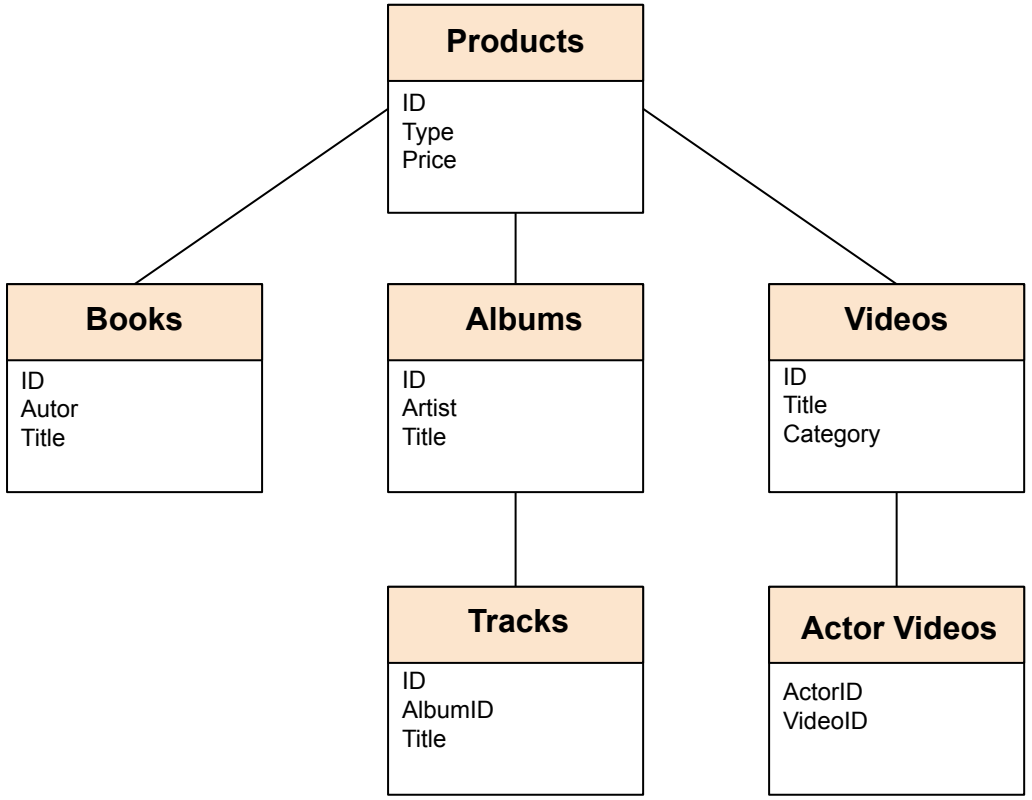
分散技術
共有ディスク
オブジェクトストア
Primary - Replica
パーティショニング
シャーディング
分散ストリーム
Quorum
分散合意
高精度時刻同期

https://speakerdeck.com/frsyuki/dbakitekutiyanobi-jiao-toxuan-ze

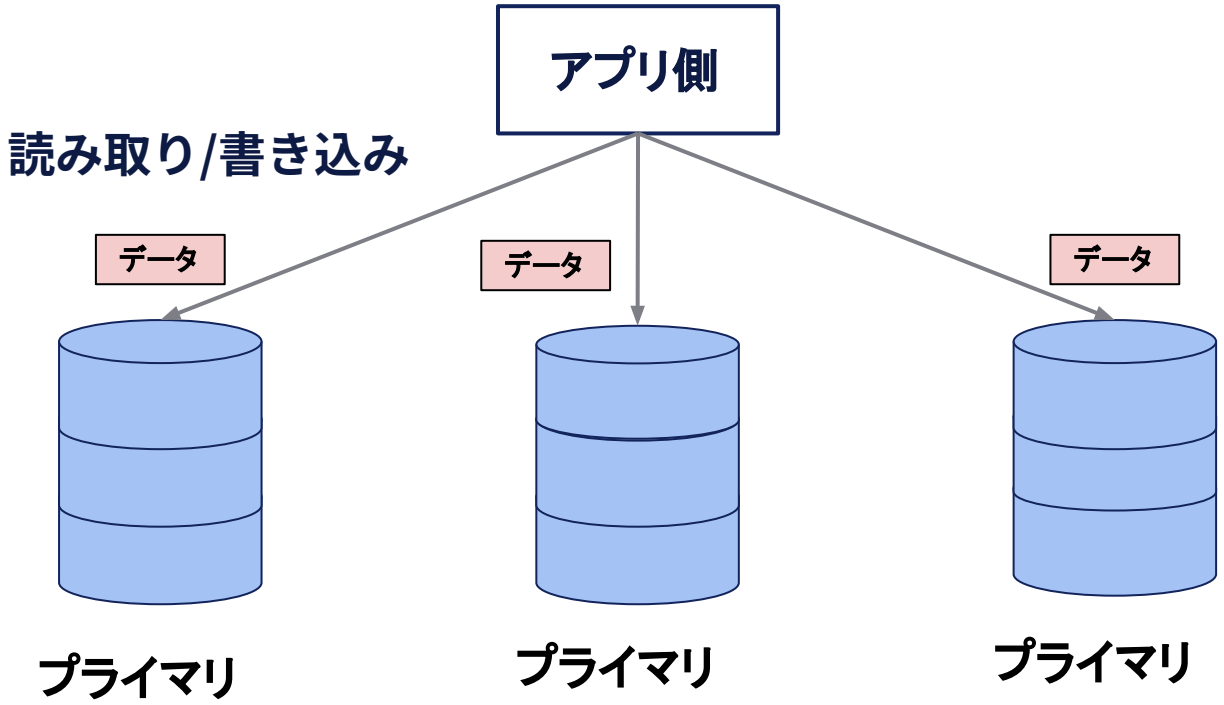
例: RDBMSアーキテクチャ / テーブル構造

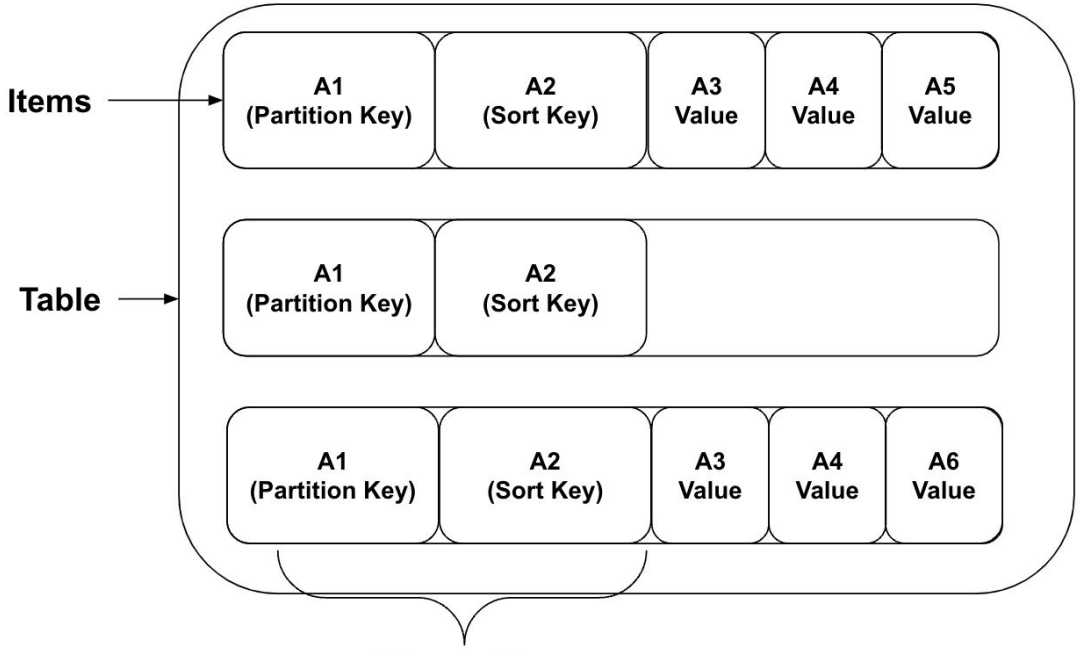


例: 製品(Products)データベース



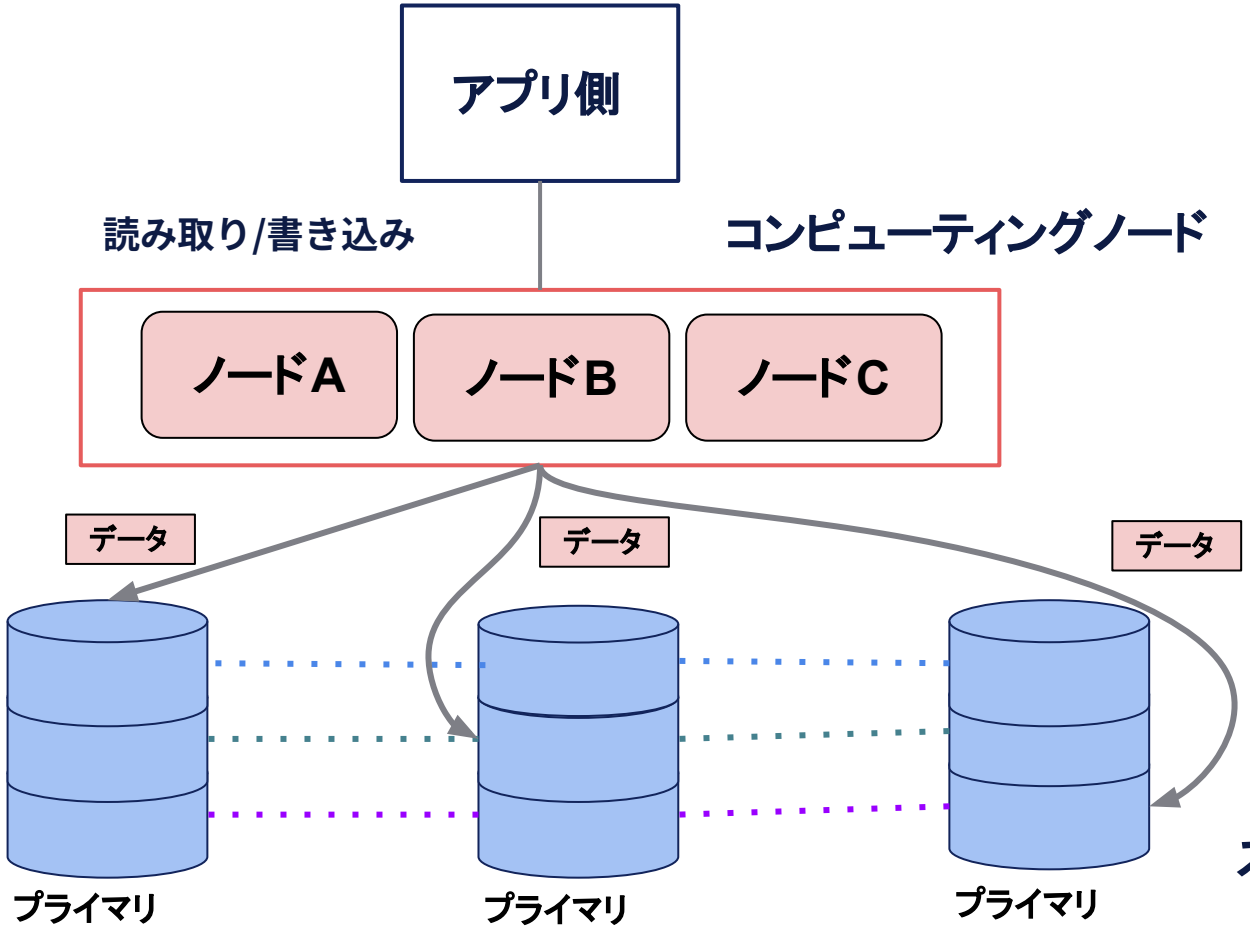
例: NoSQLアーキテクチャ / テーブル構造



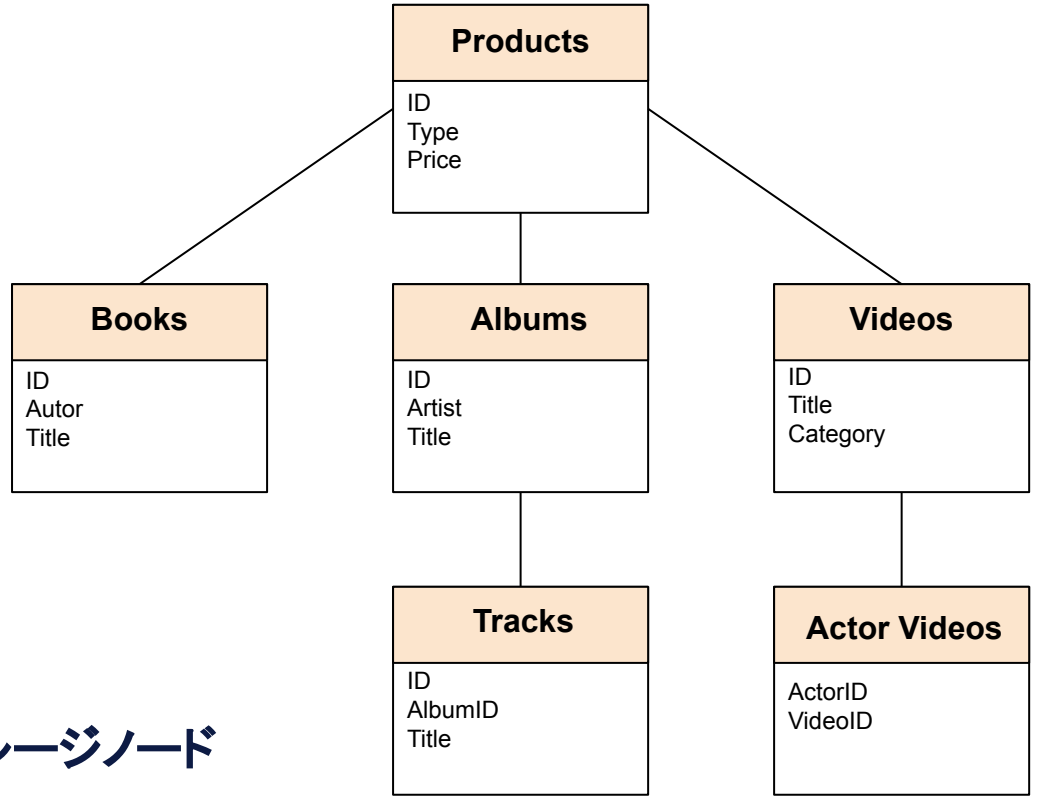


Product Table	
Partition key	Attributes
ProductId = bob	ProductType = videos, Price=$200
ProductId = fred	ProductType = Book, Price=$200

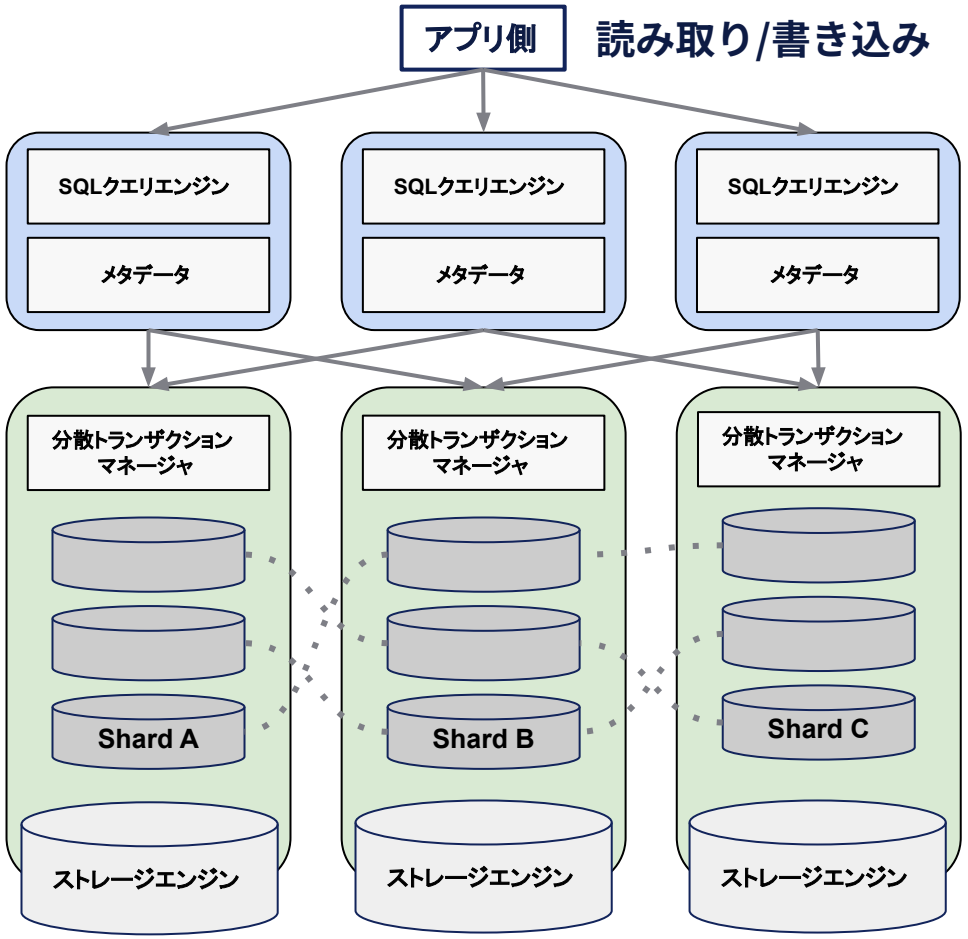
例: 分散型SQLデータベース ≒ NewSQL



例: 製品(Products)データベース



分散型SQLデータベースの内部構造



[コンピューティング層]

接続されたクライアント側から発行された SQLのパース処理や、統計情報を取り扱う

[ストレージ層]

実際のデータの読み書きを管理

2025年 分散DB/NewSQLの盛り上がり

日経XTechに特集記事掲載

第1回

分散DB「NewSQL」への乗り換え事例が相次ぐ、その仕組みとメリットをひも解く

熱田 麻貴　日経クロステック／日経コンピュータ

2025.01.31
有料会員限定

全3076文字

【「新世代ビジネスPC」とは】ノートPC入替を検討する企業へ提案する PR
【「AIプライベートクラウド」の可能性】AIとデータの“主権”は自社で持つ PR
DXベンチャーEARTHBRAIN、土木建設業界のグローバル課題に挑む PR

第2回

3大クラウドが分散DBの開発で競う、老舗のGoogle「Spanner」を追撃へ

熱田 麻貴　日経クロステック／日経コンピュータ

2025.02.04
有料会員限定

全3848文字

第3回

分散DB「Yugabyte」「TiDB」は自由度が強み、OracleやIBM製品の置き換えも狙う

熱田 麻貴　日経クロステック／日経コンピュータ

2025.02.06
有料会員限定

全2758文字

【「新世代ビジネスPC」とは】ノートPC入替を検討する企業へ提案する PR
【「AIプライベートクラウド」の可能性】AIとデータの“主権”は自社で持つ PR
DXベンチャーEARTHBRAIN、土木建設業界のグローバル課題に挑む PR

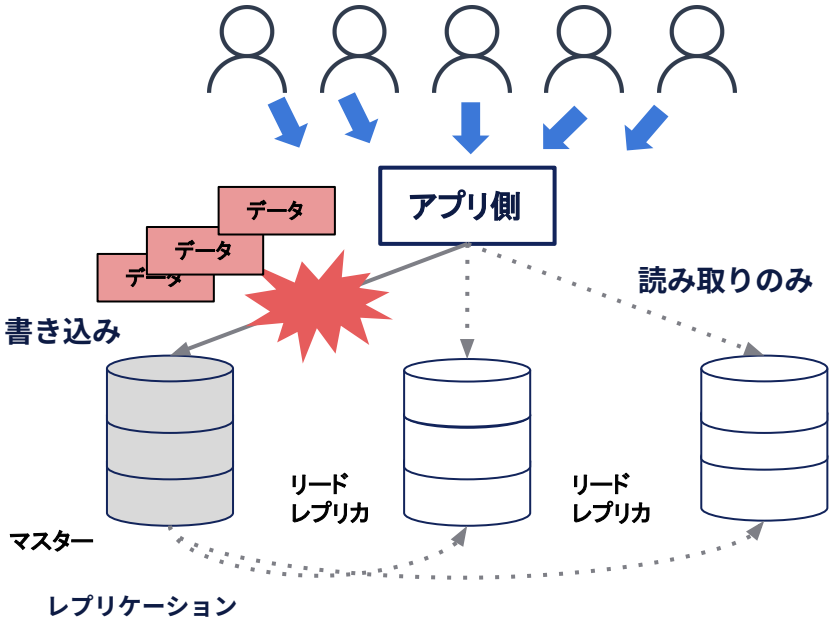
データを分散配置して拡張性を担保しながらSQLを利用できる大規模分散データベース、いわゆる「NewSQL」。今回は専業ベンダーの製品を取り上げる。YugabyteDBの「YugabyteDB Aeon」やPingCAPの「TiDB」などだ。クラウドベンダーの製品と異なり、提供形態や稼働環境の自由度が高く、オンプレミス環境にデータを保有しておけるメリットなどがある。

Amazon Aurora DSQLの登場

RDBMSと分散型SQLデータベース(NewSQL)との違い

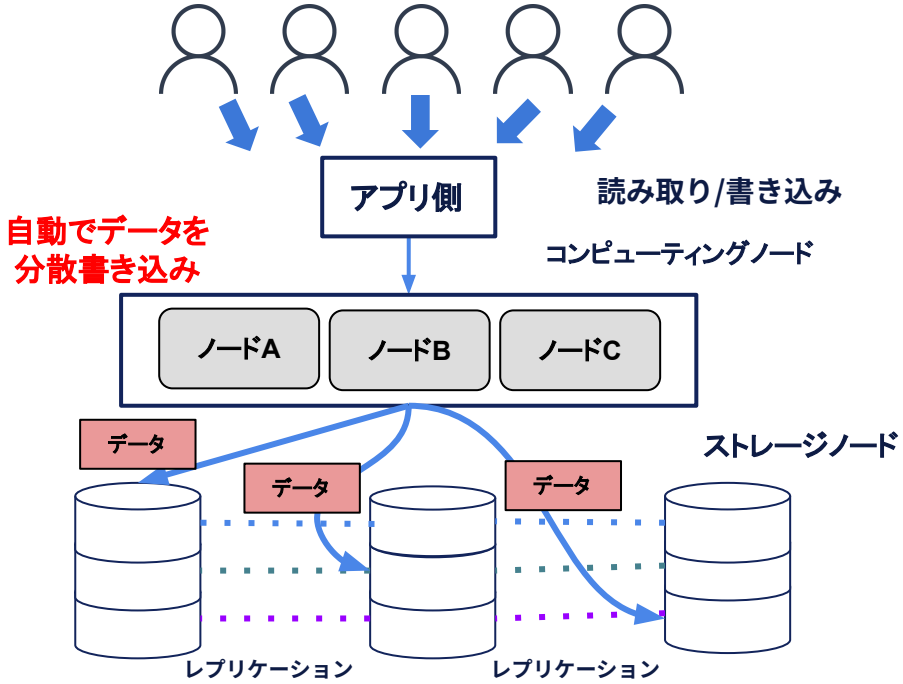
RDBMS

・大量の同時接続により書き込み負荷の集中
・リードレプリカのスケール限界

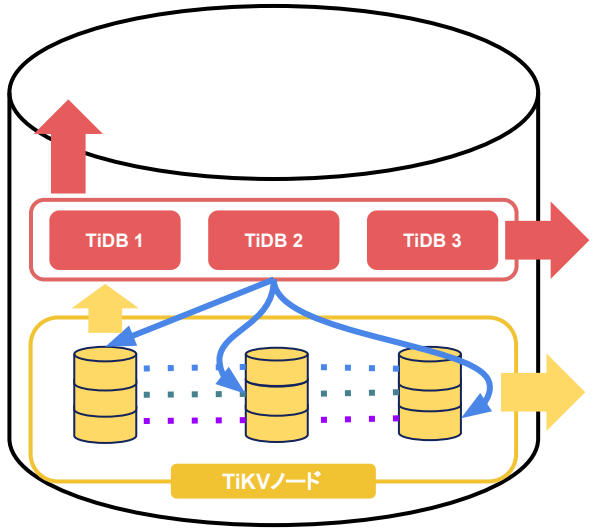


NewSQL (TiDB)

・大量の同時接続でも一定のレイテンシー
・オンラインで水平 /垂直に拡張可能

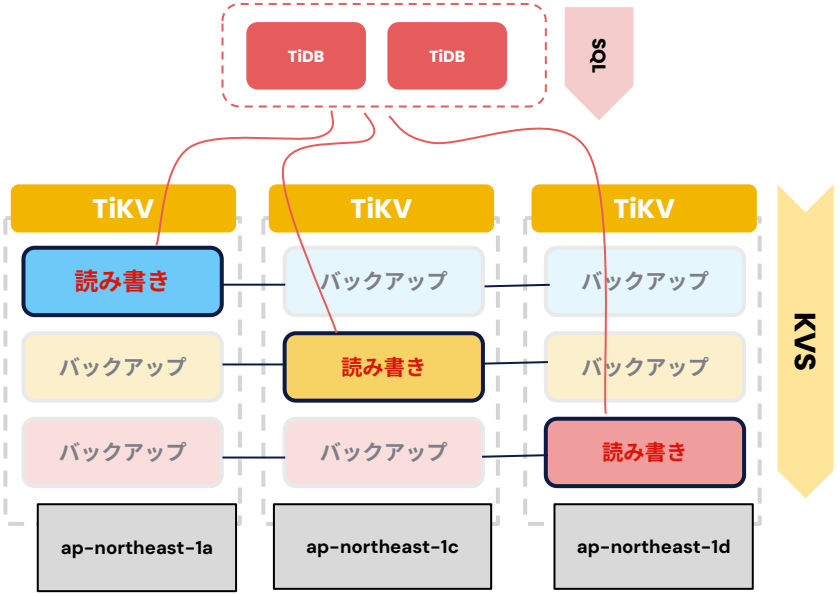


なぜ分散型SQLデータベースが注目されているのか



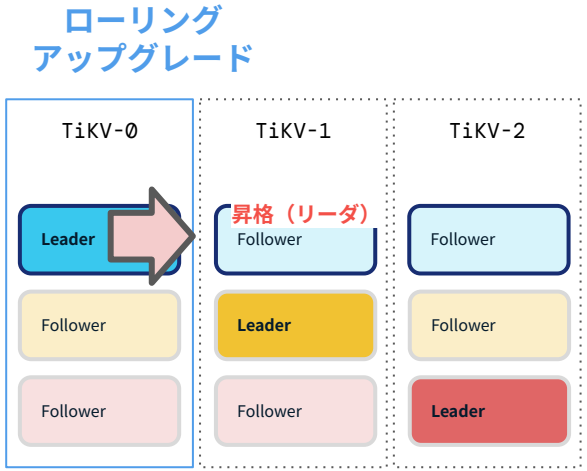
Scalable

ビジネスの成長に合わせて、
稼働するDBを止めずに拡張可能



BCP

インスタンス/DC
障害時も自動復旧



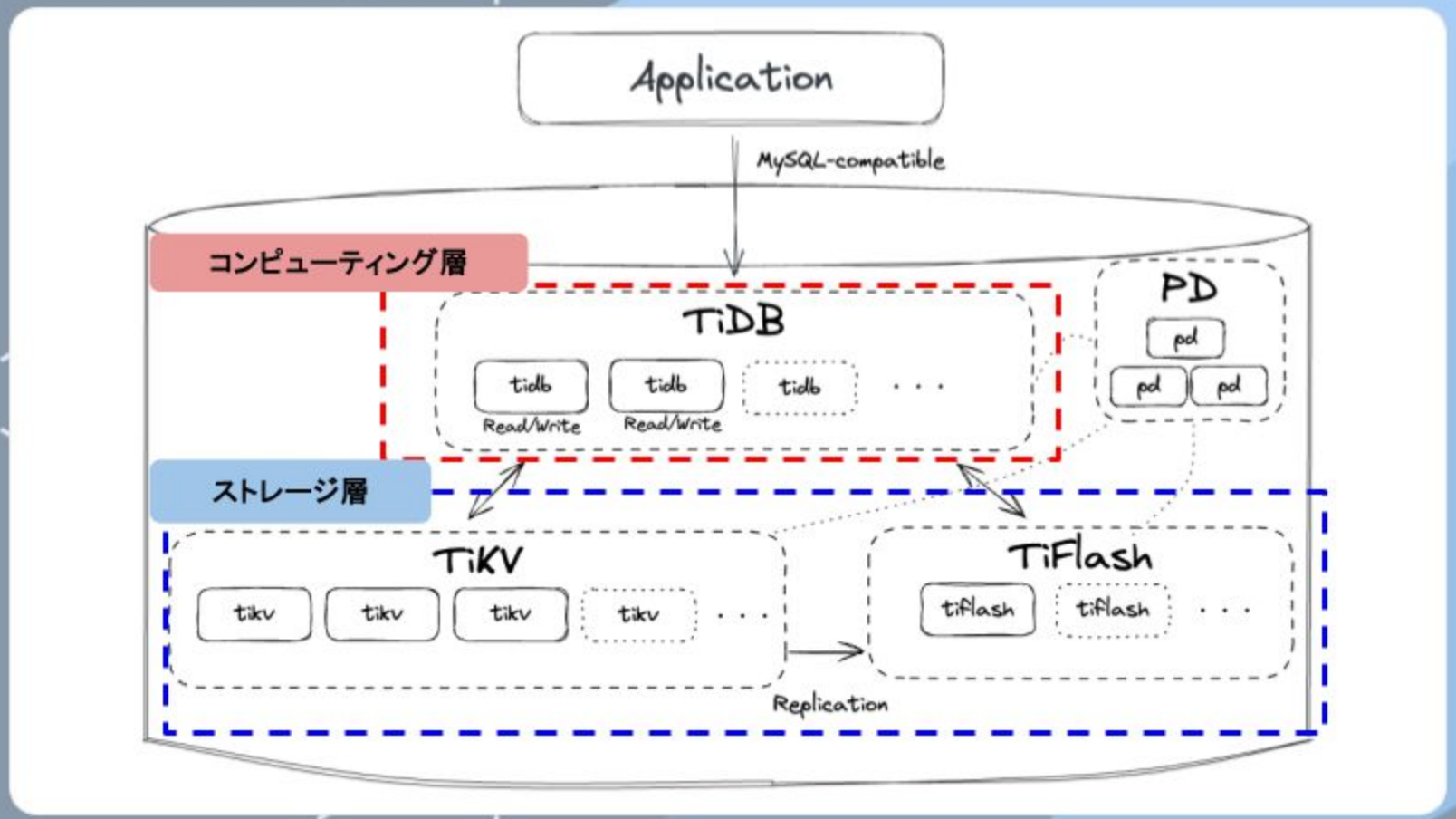
Resilient

アップグレード時
なども無停止

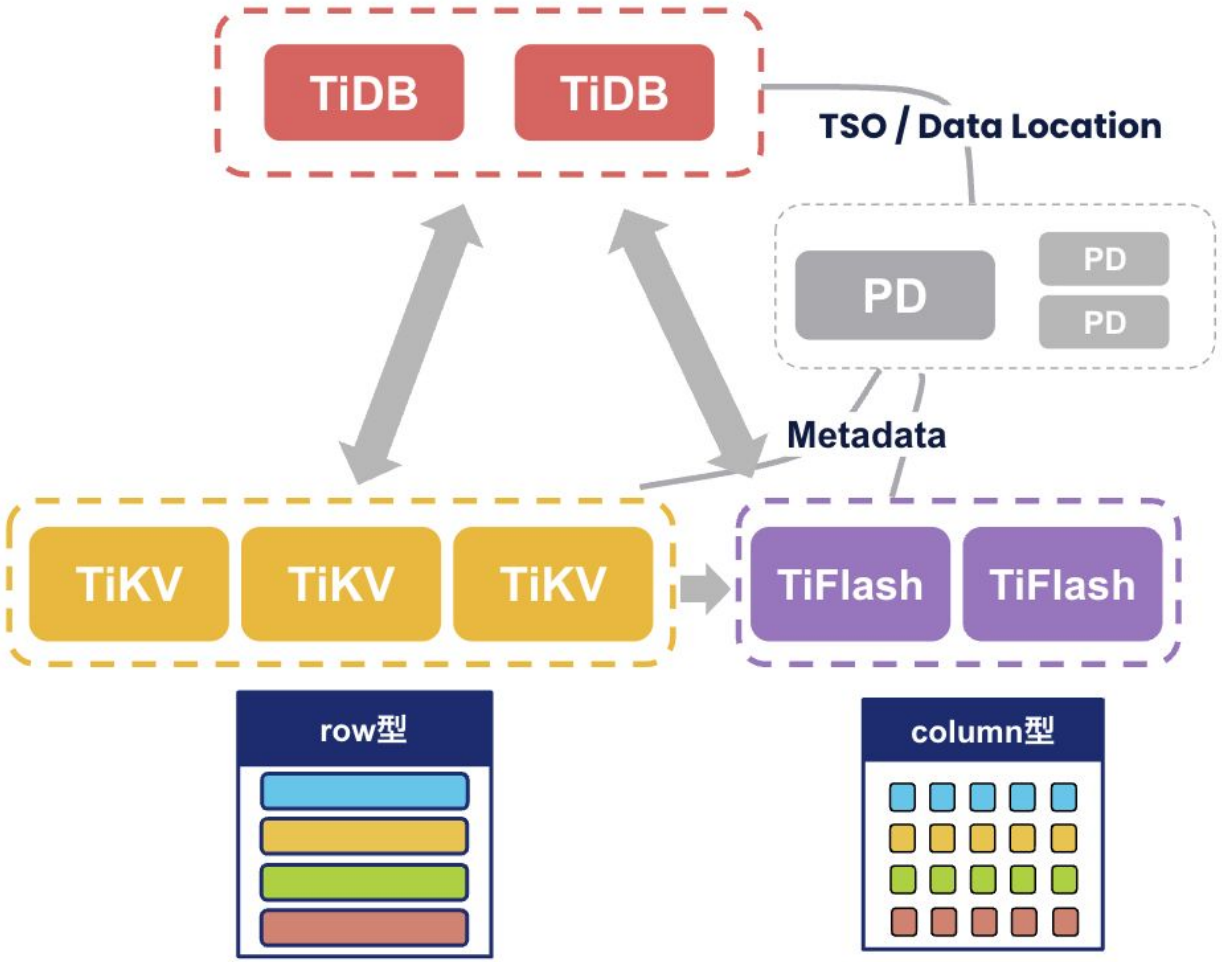
JAWS-UG
AWS User Group - Japan

TiDBの裏側
(アーキテクチャ/MVCC/分散トランザクション/Raft/ etc...)

全体アーキテクチャー


Application
MySQL-compatible
コンピューティング層
TiDB
tidb
tidb
tidb
Read/Write
Read/Write
PD
pd
pd
pd
ストレージ層
TiKV
tikv
tikv
tikv
tikv
TiFlash
tiflash
tiflash
Replication

TiDBのコンポーネント



コンピューティング TiDB

- SQL解析を行う *MySQLプロトコルをKVに変換
- ParseやOptimizeを担う
- HTAPではTiKV(row型)かTiFlash(column型)判断

ストレージ TiKV TiFlash

TiKV

- OLTP用途の分散Key-Value Store※RocksDB利用
- パーティショニングによるデータ分散管理
- Raftや2PCにより整合性担保

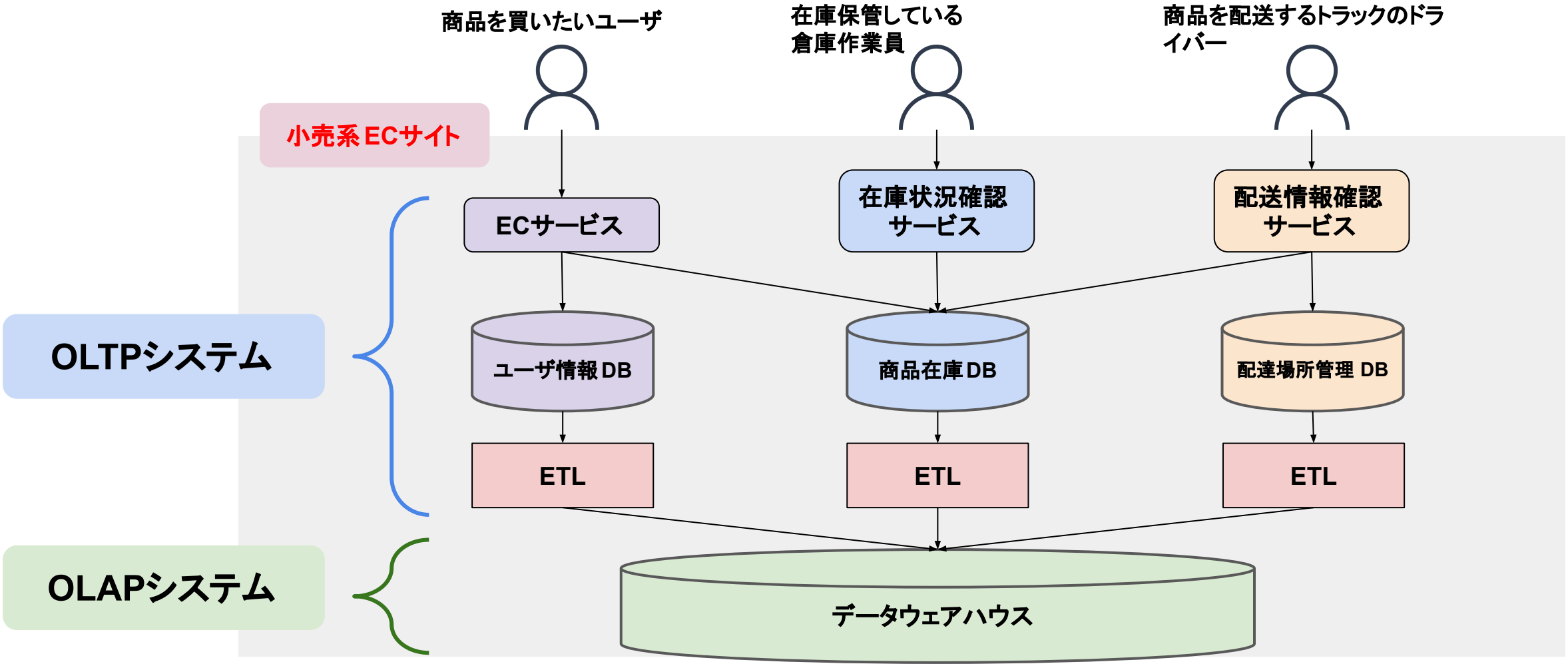
TiFlash *optional*

- HTAP利用時の追加コンポーネント
- OLAP用途のカラム型ストア

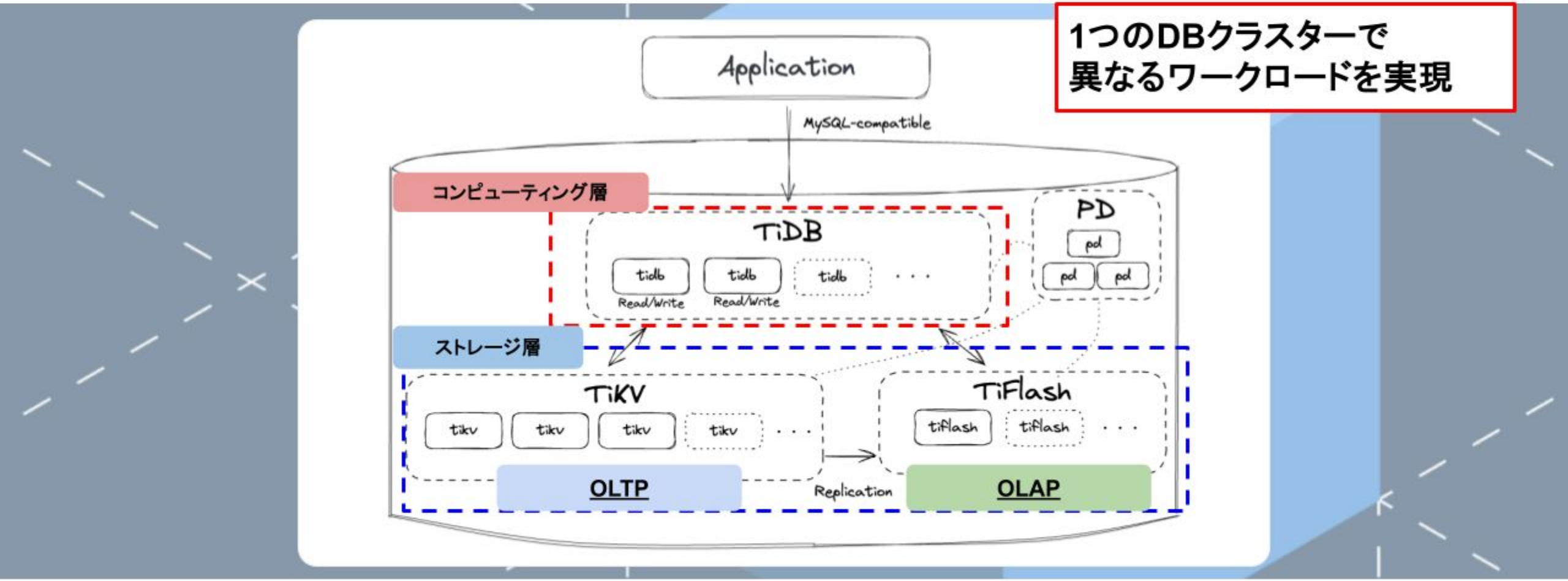
司令塔 PD

- データ配置場所の管理（Region管理）
- 分散トランザクションで利用するTSOを発行

よくあるOLTPワークロード / OLAPワークロード



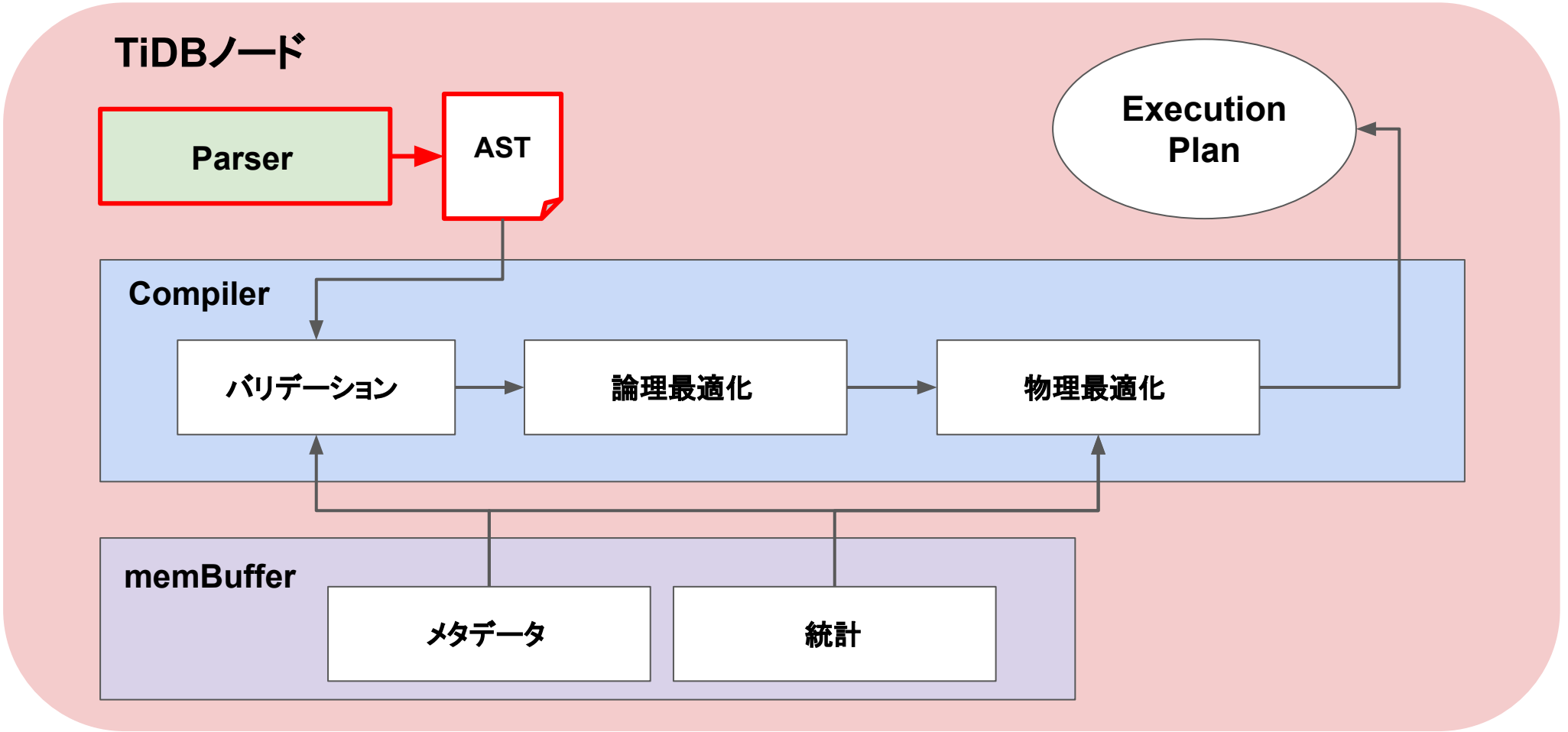
TiDBならではの機能として HTAP



JAWS-UG
AWS User Group - Japan

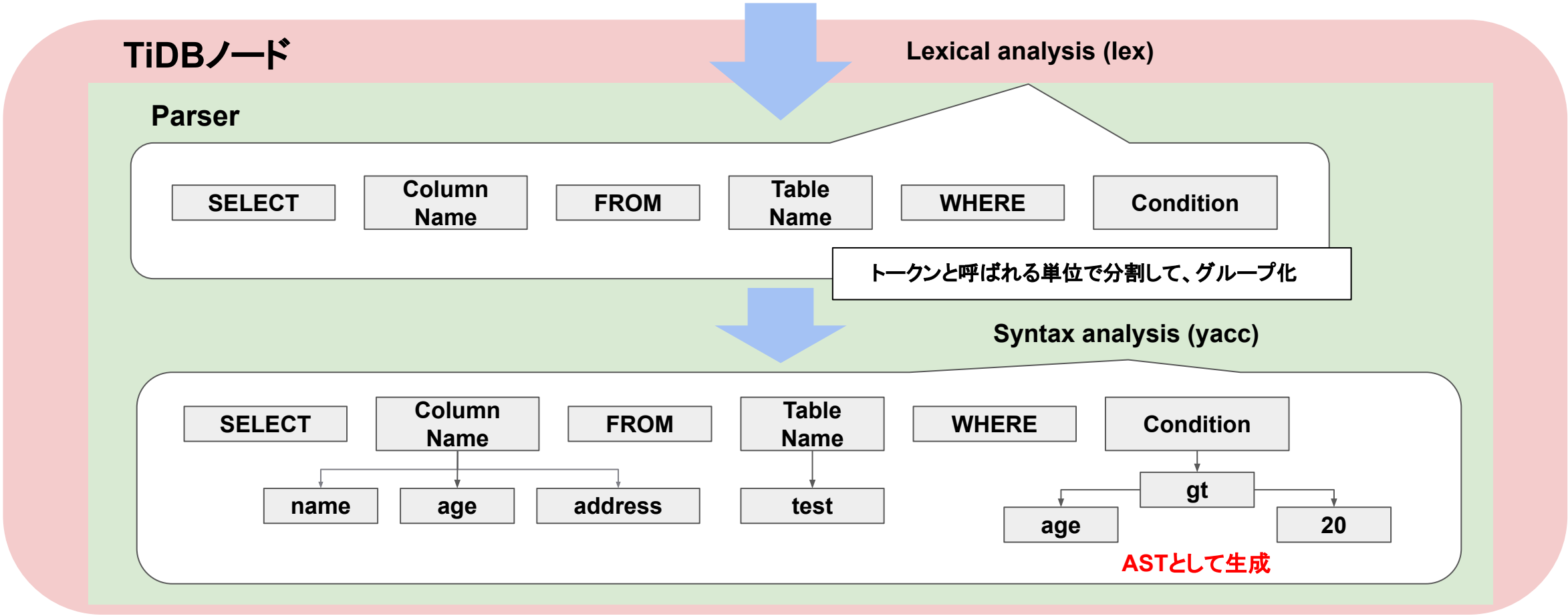
詳細な分散DBの仕組みへDeep Dive

TiDBの役割: SQLの変換とトランザクション



Parser処理

SELECT * FROM test WHERE age > 20;



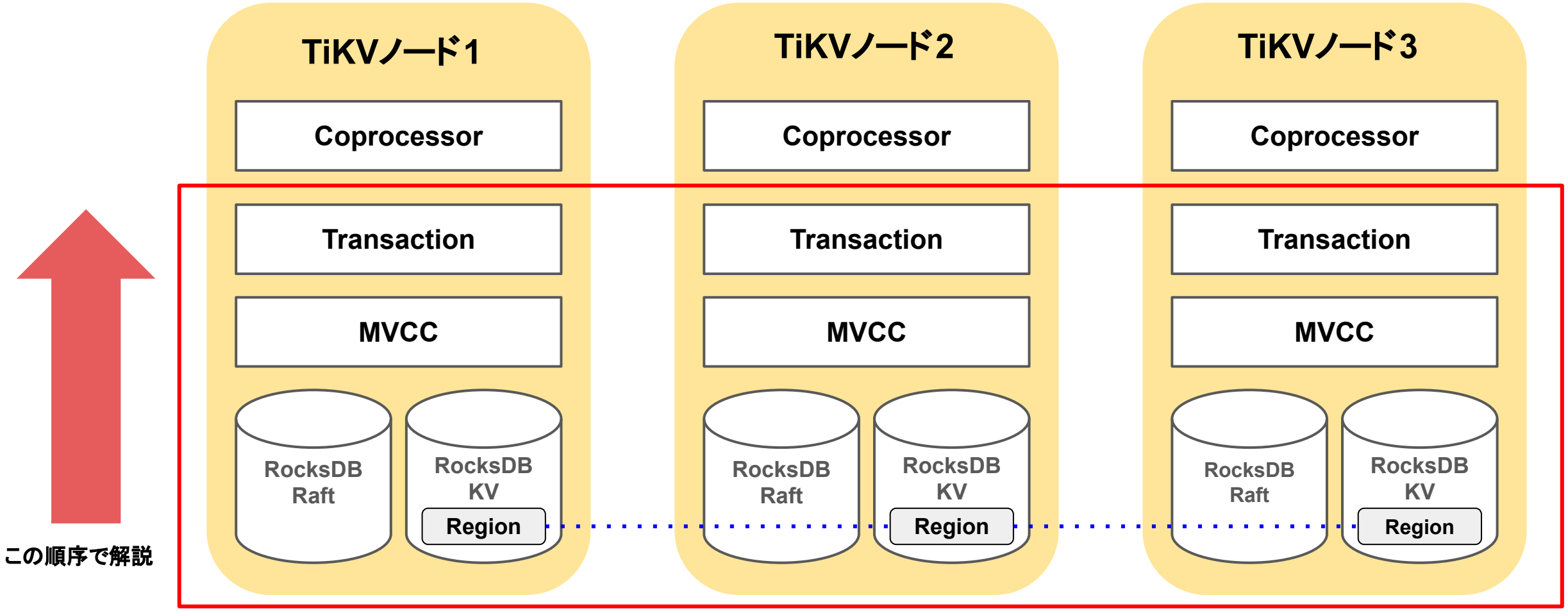
TiDBにおけるテーブルからKey-Valueへの変換

- 行データを <{table_id, primary key},{values*}> の形のKVに変換
- インデックスも <{index_columns*},rowid>の形のKVに変換

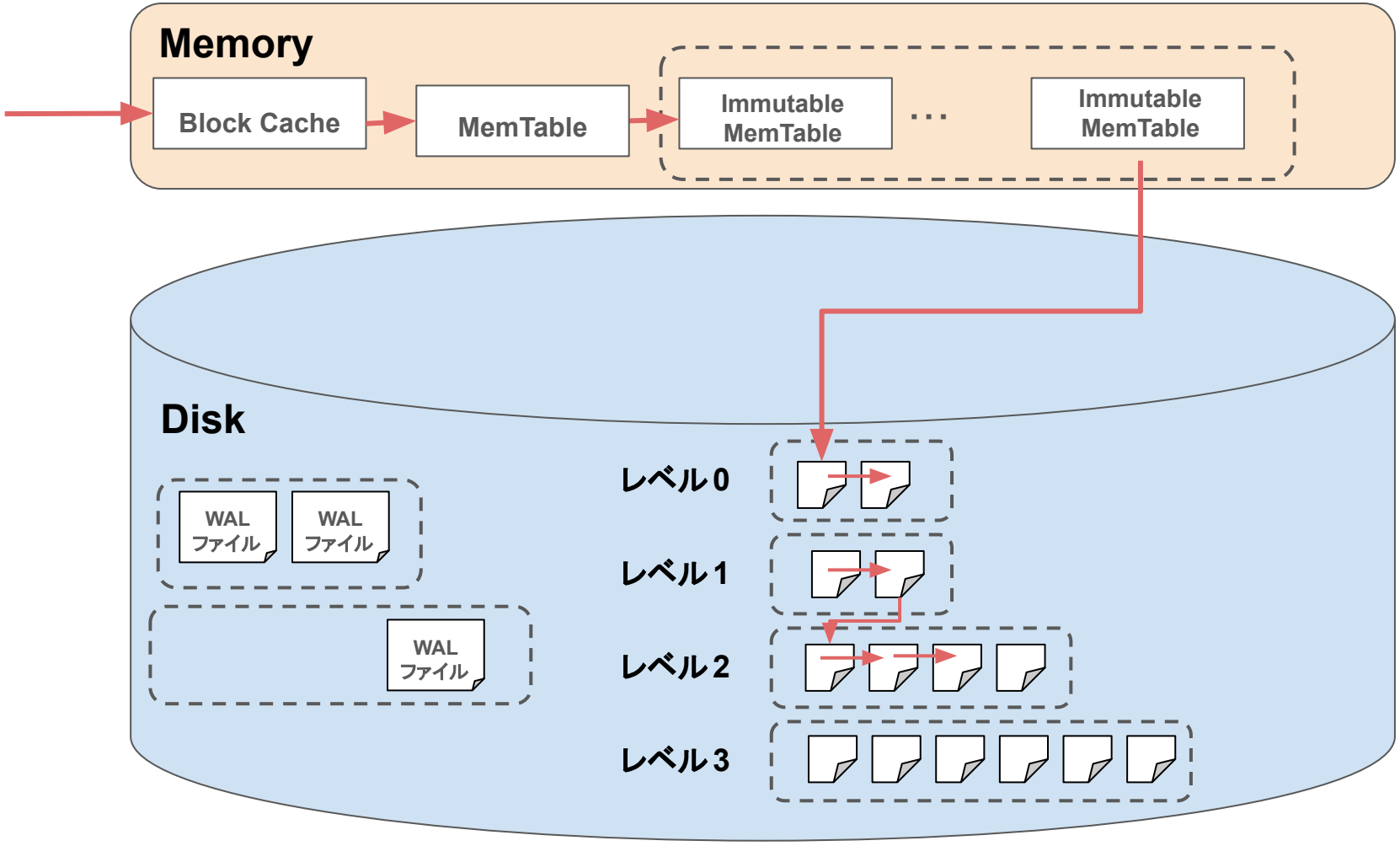
id	名前	誕生日	携帯電話	点数
r1	Tom	1982-09-28	1390811212	78
r2	Jack	1996-04-12	1801222187	91
r3	Frank	1982-09-28	1364571212	90
r4	Tony	1977-03-12	1391113134	65
r5	Jim	1992-07-19	1579915611	51
r6	Sam	1978-09-12	1713665011	97

Key	Value
tid_r1	Tom, 1982-09-28, 1390811212, 78
tid_r2	Jack, 1996-04-12, 1801222187, 91
tid_r3	Frank, 1982-09-28, 1364571212, 90
tid_r4	Tony, 1977-03-12, 1391113134, 65
tid_r5	Jim, 1992-07-19, 1579915611, 51
tid_r6	Sam, 1978-09-12, 1713665011, 97

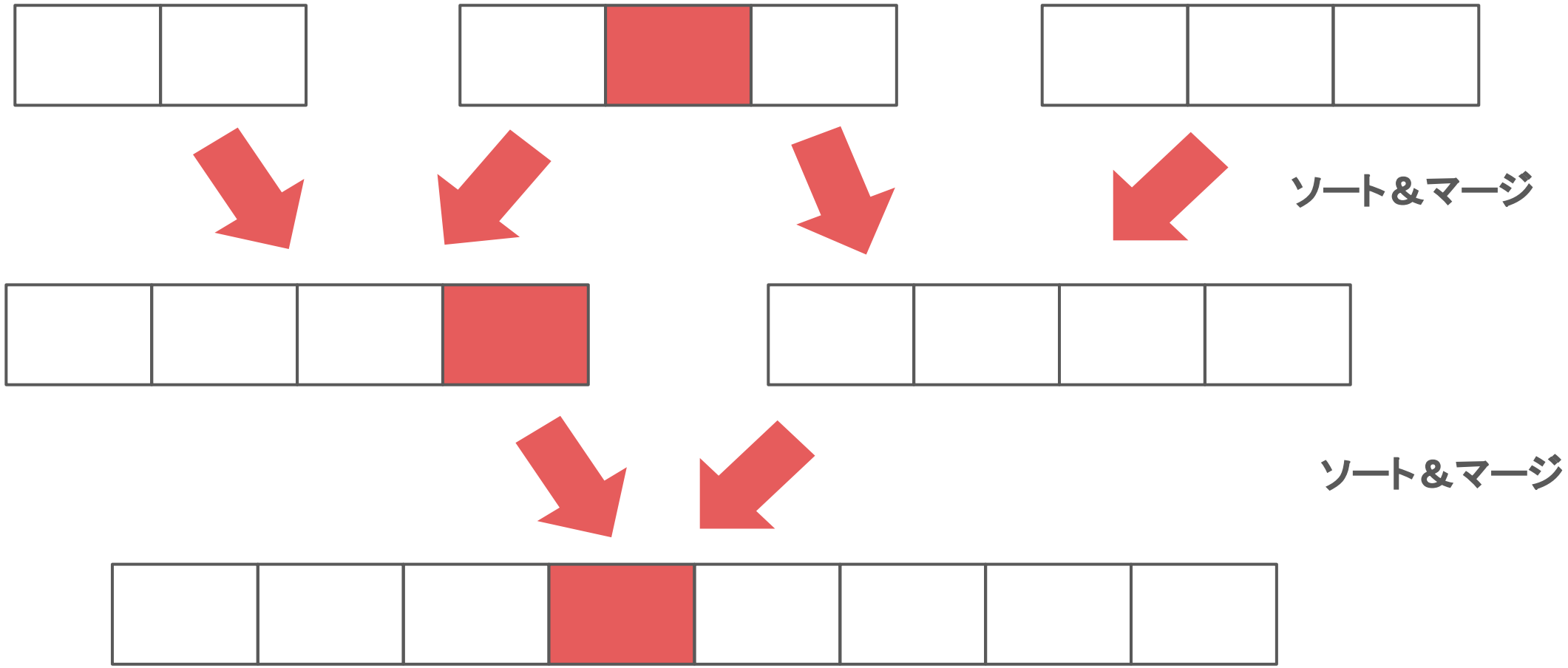
TiKVの役割: データの保存と一貫性



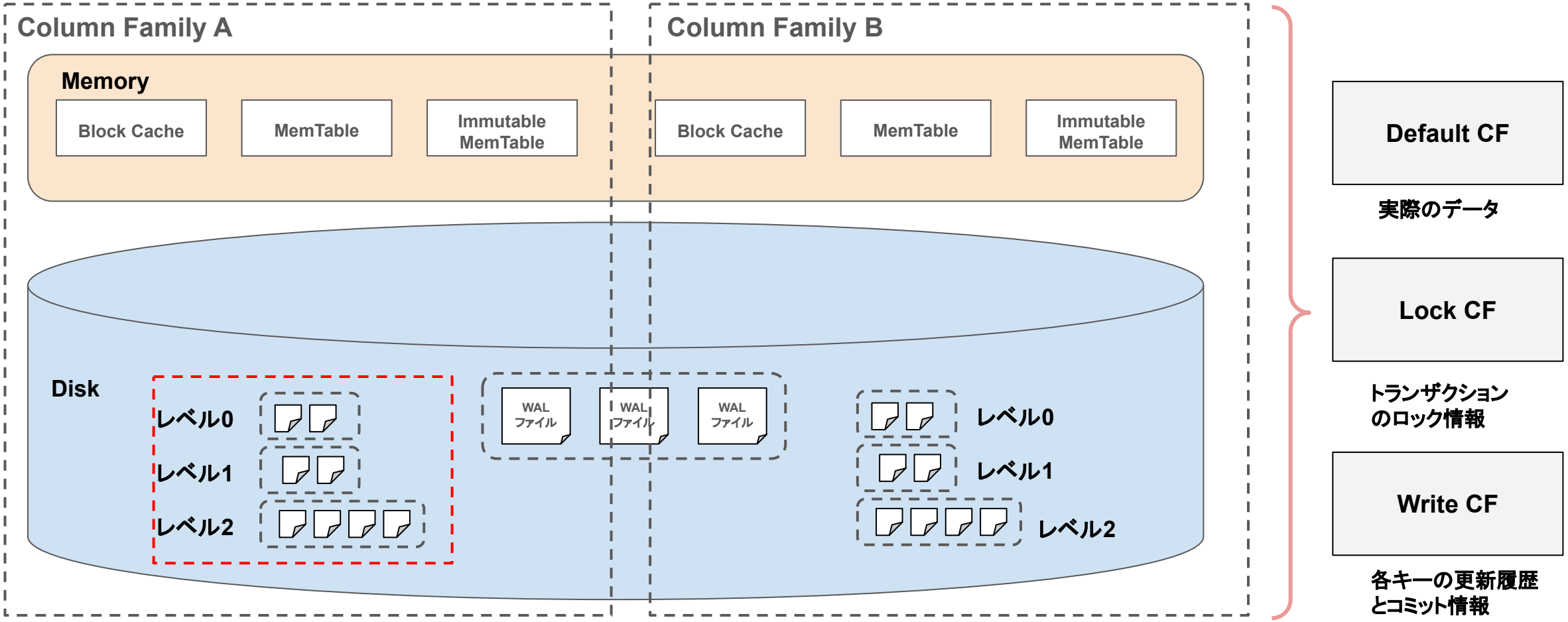
RocksDBの書き込み処理の特徴



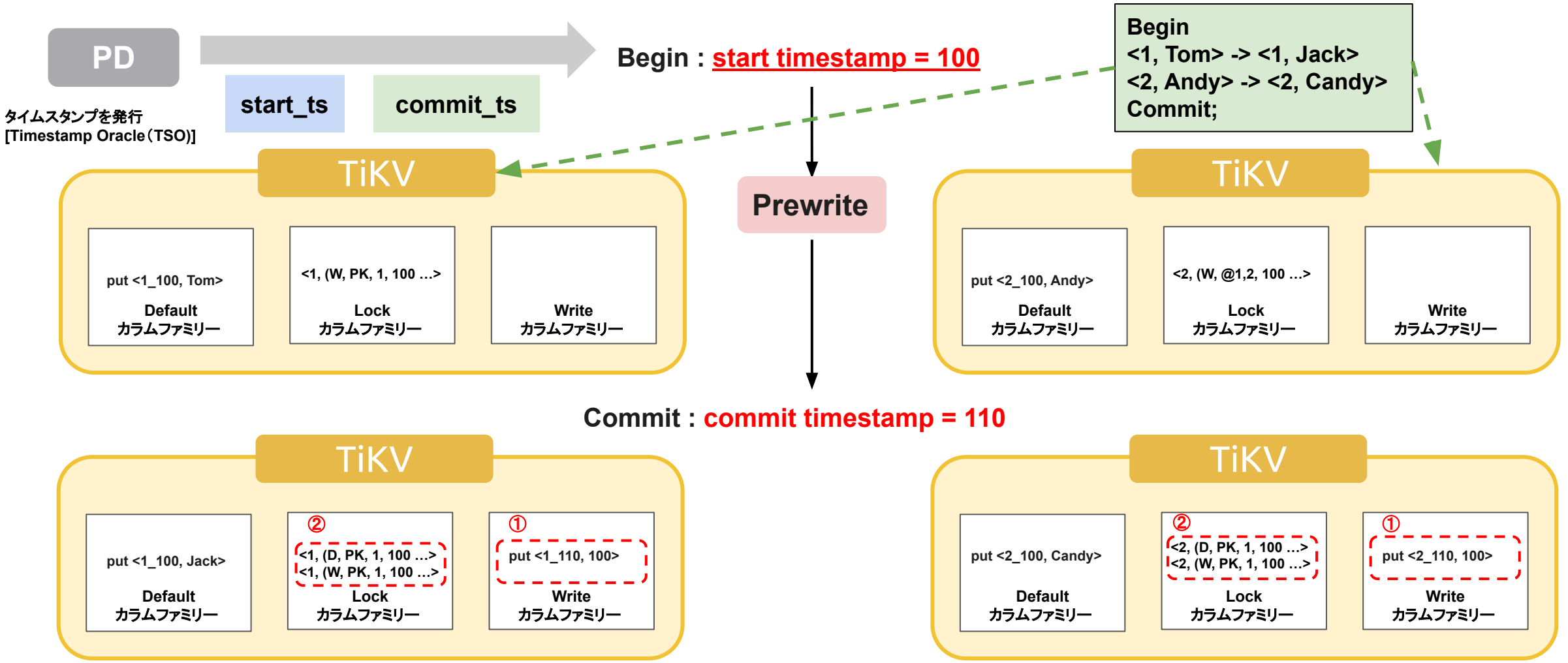
ログ構造化マージ(LSM)ツリーの動き



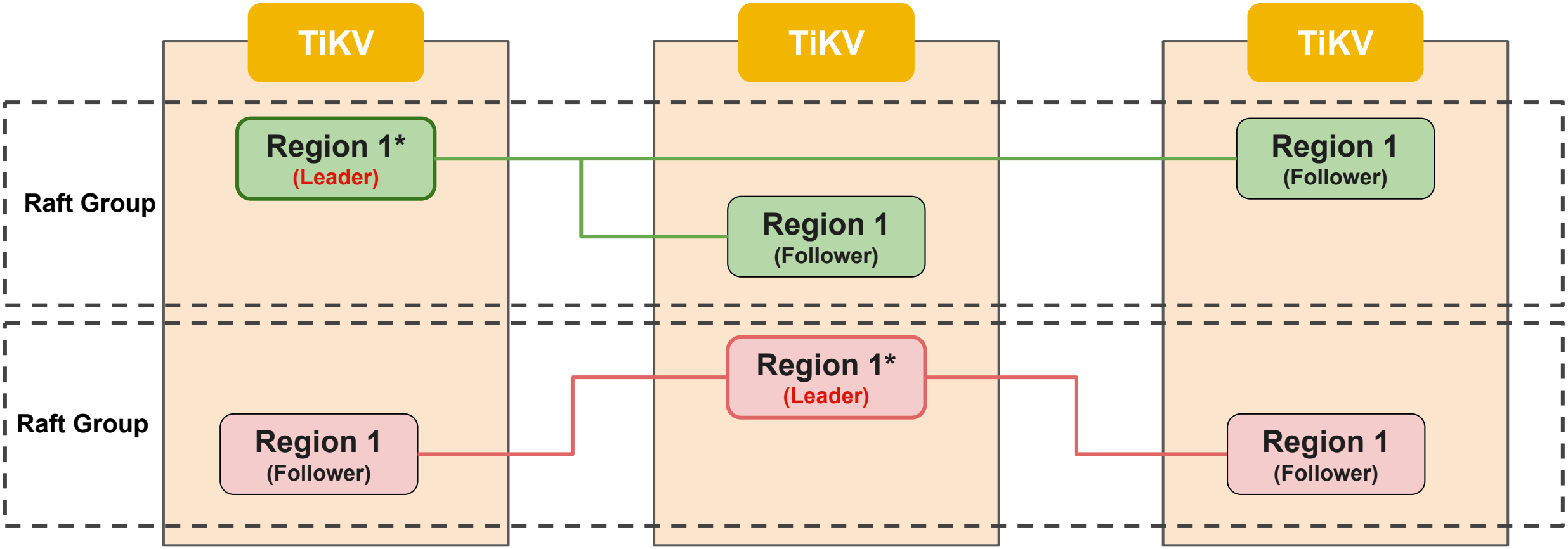
RocksDBにおけるCF(カラムファミリー)



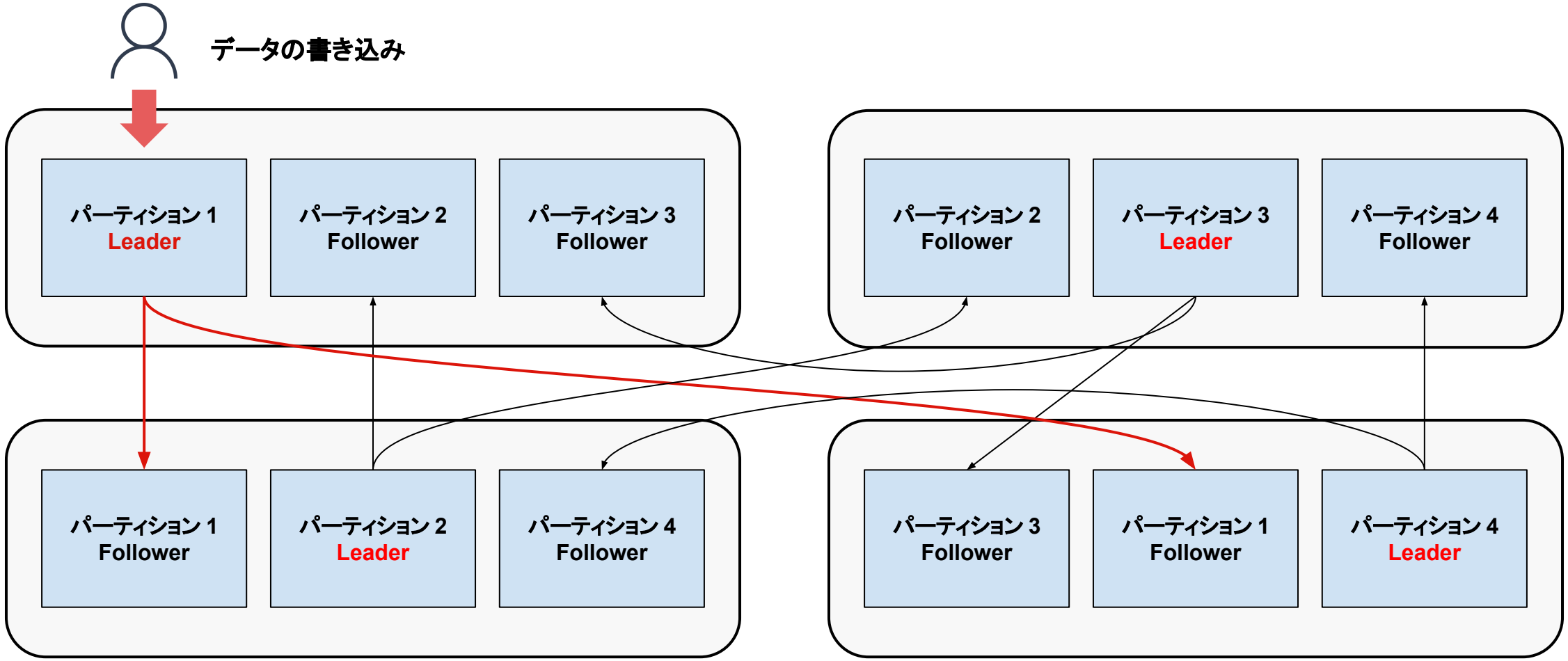
分散トランザクション / MVCC



Raftによるログレプリケーションの動き



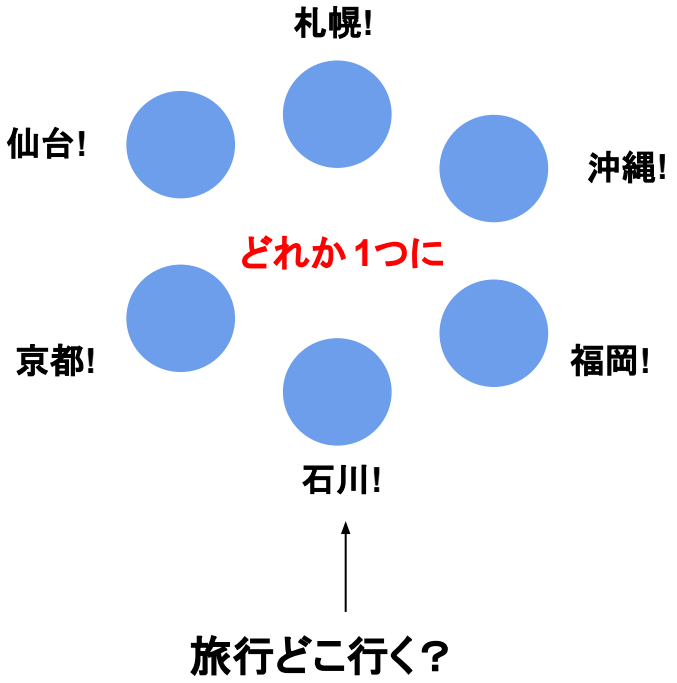
パーティショニングとレプリケーションの動き



レプリカによる可用性: Raft

分散システム : Consensus(合意)形成は重要

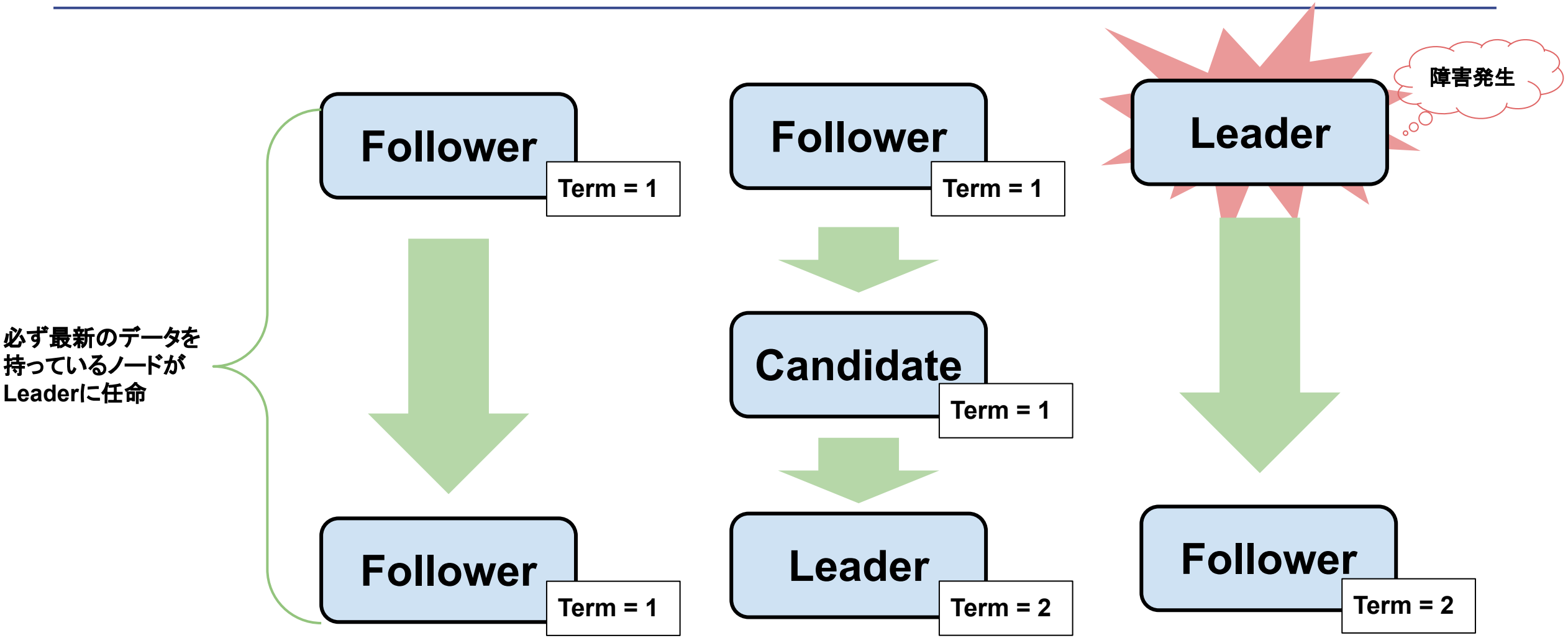
- 異なるReplicaを作り出さないようにする
- リーダーの決定や分散トランザクションに利用



一貫性を保つ合意アルゴリズム

- Raft => TiDBではこちらを採用
- Paxos

Raftによる選挙の動作

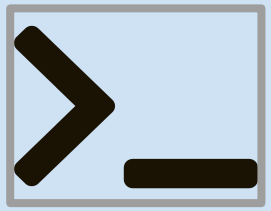


実際に使いたい場合どうすれば？？

TiDBの利用パターン

パターン 1

Instance



TiUP

TiDBのセットアップから運用まで
※ローカルでも使える

パターン 2

Kubernetes

HELM

TiDB Operator

K8sでのセットアップ・運用

パターン 3

Cloud

TiDB

aws Google Cloud Microsoft Azure

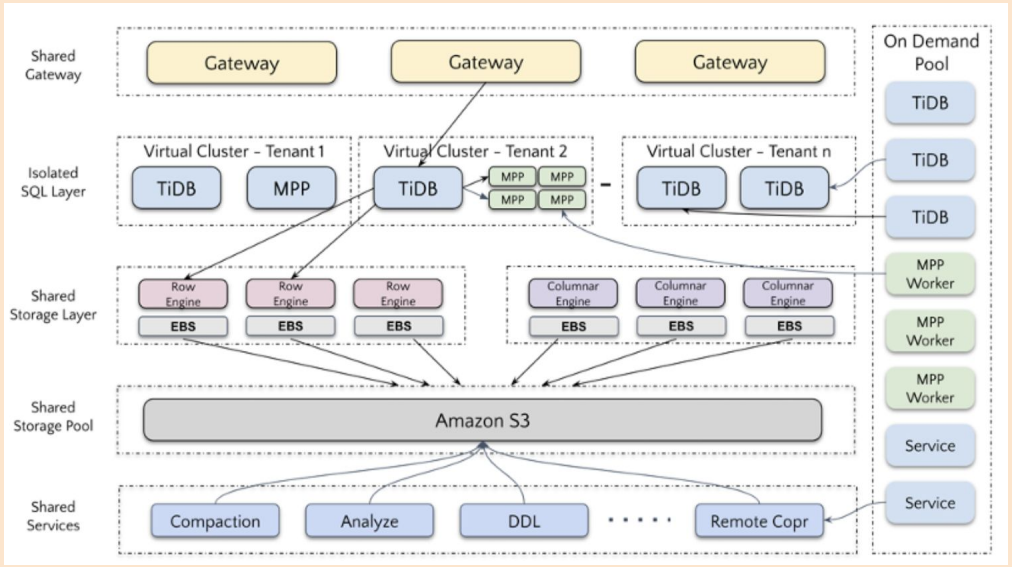
※coming soon

TiDB Cloud

フルマネージドDBaaS
ServerlessとDedicatedがある

TiDB Cloudのラインナップ

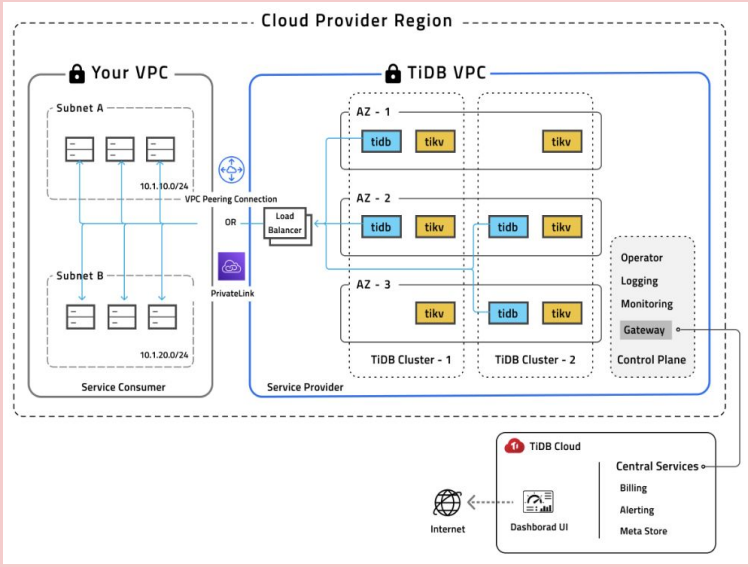
Serverless



マルチテナント型

- シングルAZ / Autoスケール ※マルチAZ版も予定
- BranchingやEdge Function用のDriver用意
- Vector Search※Betaもあり

Dedicated



専有型

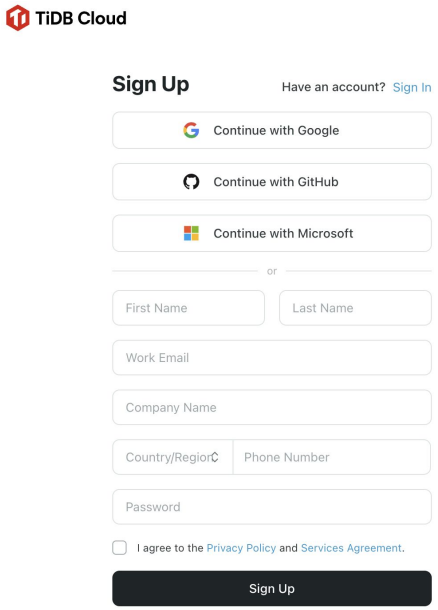
- マルチAZ / 各コンポーネントスケール(APIあり)
- MySQLからの移行ツールもフルで使える
- OSSライクに柔軟なチューニング可能

でも、お高いんでしょ？

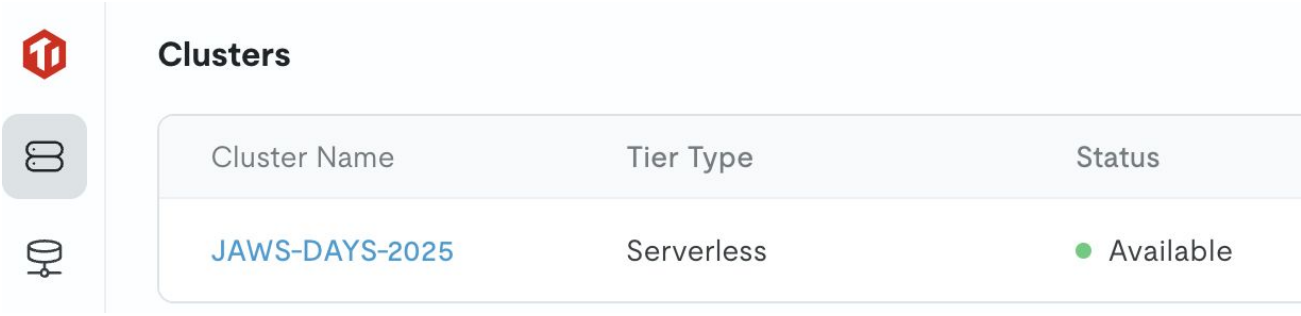
簡単に始められるTiDB Serverless

クレカ不要！
$0から利用可能

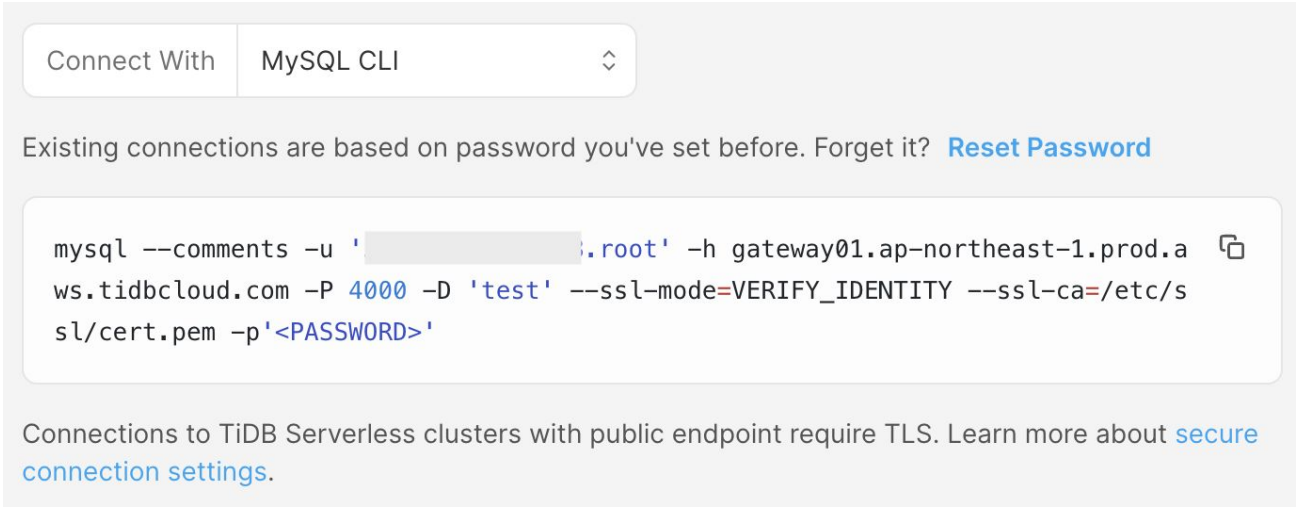
・1ヶ月間で行ベースデータ 5GiB /
列ベースデータ 5GiBを同時に保存可能
・5,000万RUを消費可能
※無料クラスター：1アカウント5つまで



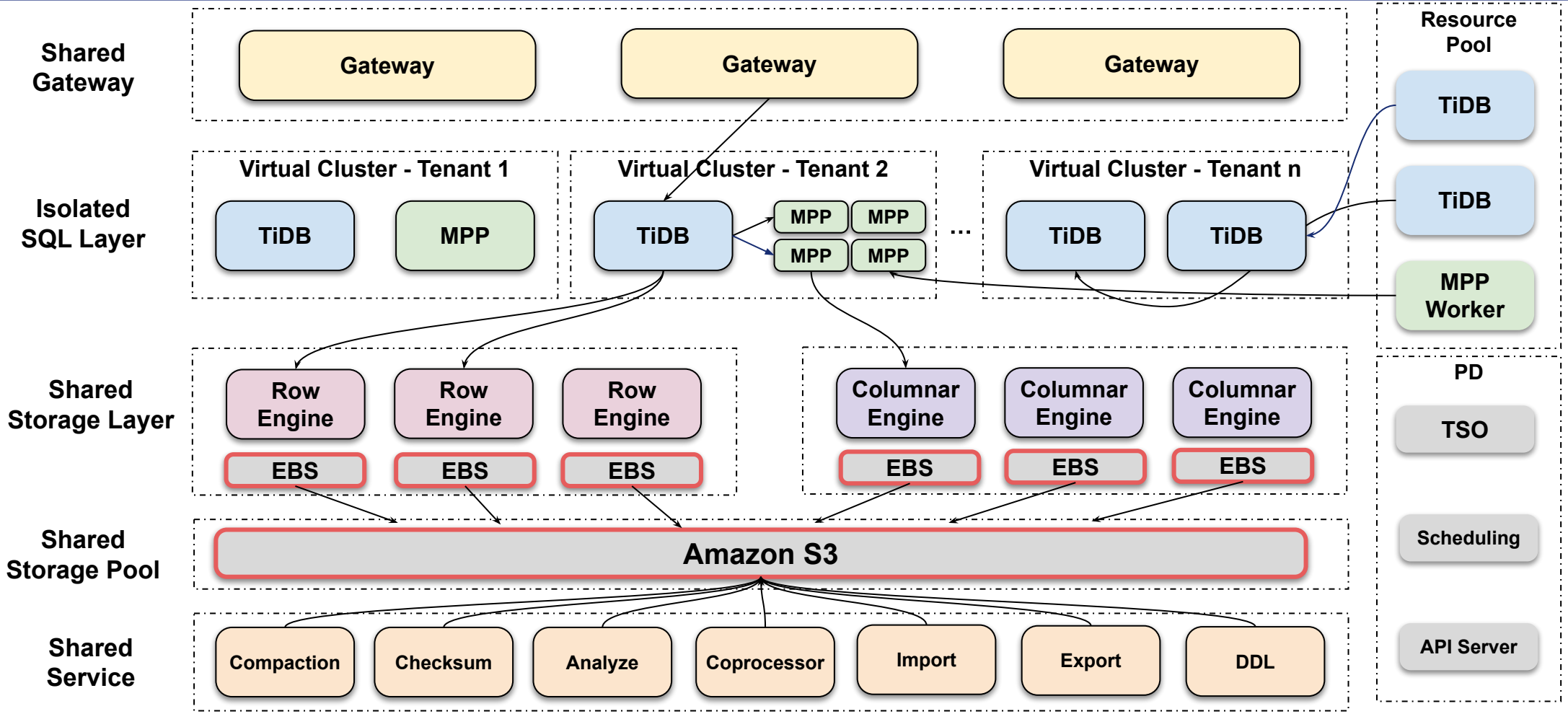
クラスタの起動10秒くらい



MySQLクライアントからの
接続時の情報も自動生成

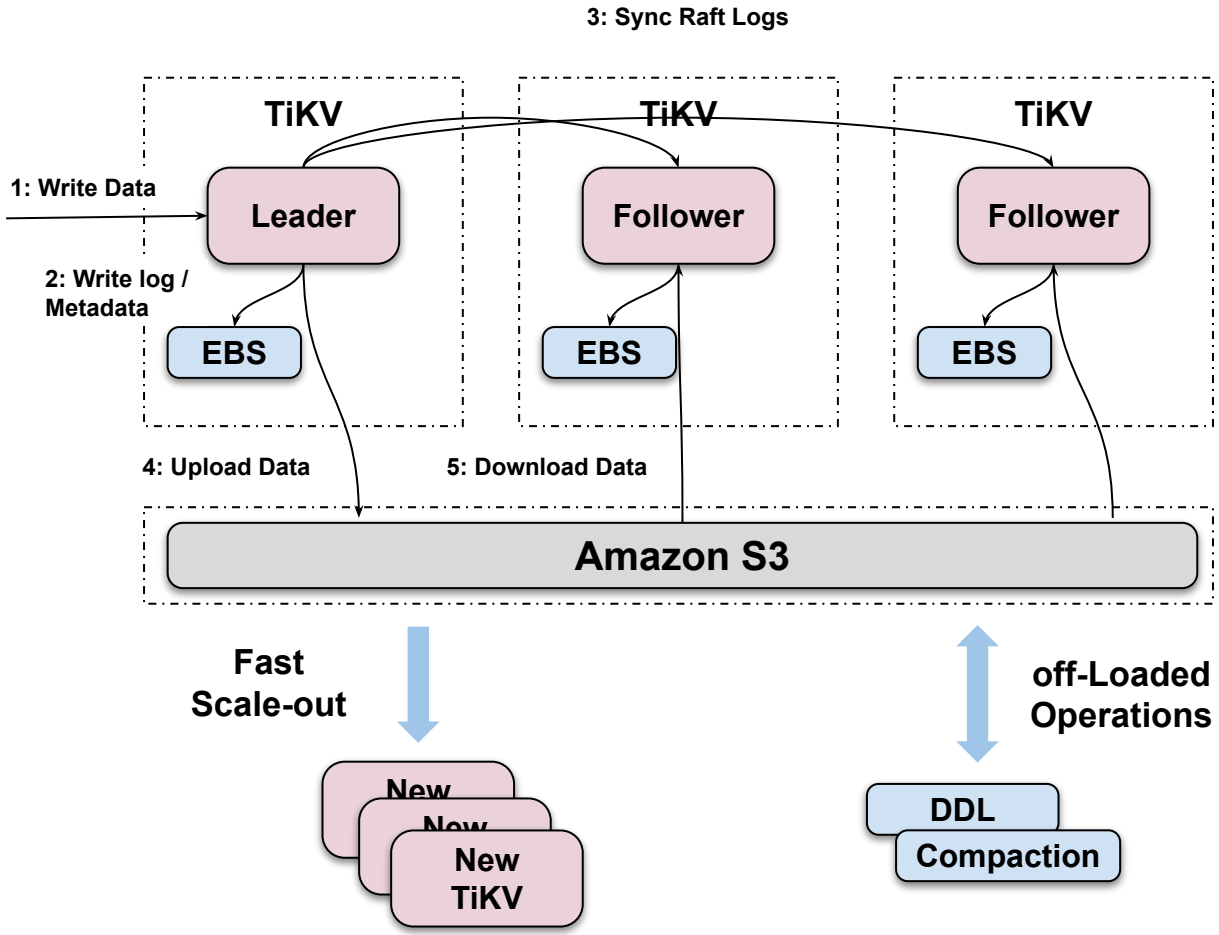


TiDB Serverless - Serverless DB on AWS



https://aws.amazon.com/jp/blogs/storage/how-pingcap-transformed-tidb-into-a-serverless-dbaas-using-amazon-s3-and-amazon-ebs/

Storage architecture data workflow



1. TiDBノードがTiKVへプッシュダウン
2. 最初にリーダーレプリカが存在するノードの EBSにWALとRaft ログが書き込まれる
3. Raft ログが残りのフォロワーレプリカに伝播
4. データをEC2インスタンスストアにキャッシュした後、TiDBは変更をS3に非同期的に書き込み
5. TiDB は他の2つのフォロワーレプリカにデータ読み込み信号を送信し、信号を受信すると、フォロワーレプリカはS3からデータを読み込み、マルチレプリカレプリケーションが完了

実際の運用現場で直面する課題

前提として: TiDBのテーブル構造

TABLE ※デフォルト：CLUSTERD

```
CREATE TABLE t (
`id` INT PRIMARY KEY /*T![clustered_index] CLUSTERD*/,
`name` VARCHAR(255) );
```

(Key) - (Value)

Primary Key - row data

Key (PK)
1001
1002
1003

Value (row data)
やまだ
いとう
さとう

- 1レコードを1つのKey-Valueで持つ
- Key：PK　/　Value：その他のColumn

INDEX

```
CREATE TABLE t(
  `id` INT PRIMARY KEY /*T![clustered_index] CLUSTERED */,
  `name` VARCHAR(255),
  INDEX id_name (`name`));
```

(Key) - (Value)

Index - Primary Key

Key (PK)
やまだ
いとう
さとう

Value (PK)
1001
1002
1003

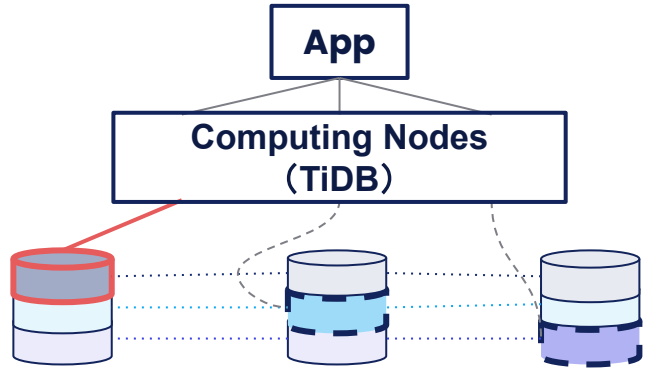
- Key：Index　/　Value：PK
- Index Lookupは2ホップでのアクセス
  ※①Index KV → ②Table（Row Data）KVの順

※実際のKeyにはTableの情報も含まれます

Writeの観点：PKを散らす

PKが連番

多数の近い値を主キーに挿入すると、
書き込みホットスポットを誘発
ex. Auto Incrementテーブルに大量INSERT



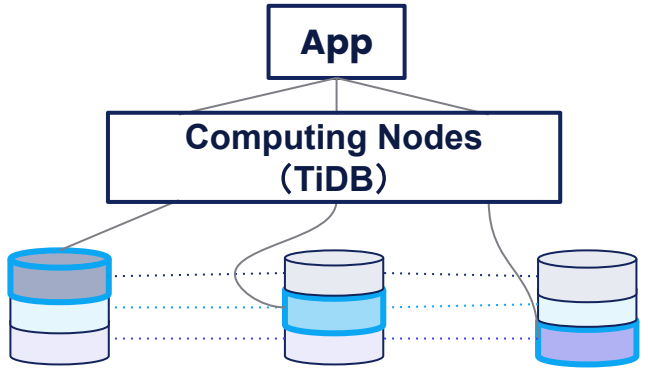
id（PK）	created at
1	4/1 12:00
2	4/1 12:01
・・・	・・・

id（PK）	created at
・・・	・・・
9999999	4/4 2:00

id（PK）	created at
・・・	・・・

PKがランダム ※バラつきあり

主キーにはバラつきのある値を利用
-> 負荷もバラけるためHotspot回避
※Auto RandomやUUIDなど



id（PK）	created at
100100001	4/1 12:00
100100002	4/1 12:03
100100003	4/1 12:06

id（PK）	created at
234100001	4/1 12:01
234100002	4/1 12:04
234100003	4/1 12:07

id（PK）	created at
368400001	4/1 12:02
368400002	4/1 12:05

Readの観点：PKの工夫

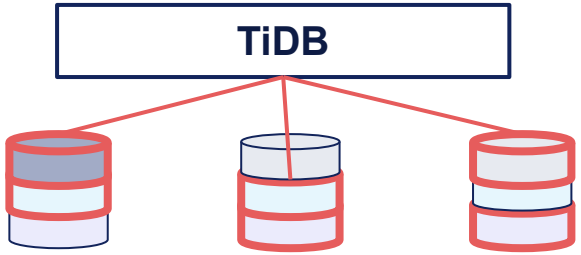
要件：ユーザー情報を時系列で取得する

```
SELECT * FROM `user_action_logs`
 WHERE `user_id` = ? AND `created_at` >= ?
 ORDER BY `created_at` DESC, `id` DESC LIMIT ?
```

サロゲートキー + Index

```
CREATE TABLE `user_action_logs` (
  `id` varchar(255) NOT NULL,
  `user_id` varchar(255) NOT NULL,
  `log_data` json DEFAULT NULL,
  `created_at` timestamp NOT NULL,
  `updated_at` timestamp NULL DEFAULT NULL,
  PRIMARY KEY (`id`) /*T![clustered_index] CLUSTERED */,
  KEY `user_action_logs_user_id_created_at_index` (`user_id`,`created_at`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin
```

id（PK）	user_id	created_at
asdijasoja	100001	8/19 12:00
zmpkogkp	100001	8/20 22:00
haoahapji	100001	8/21 5:00

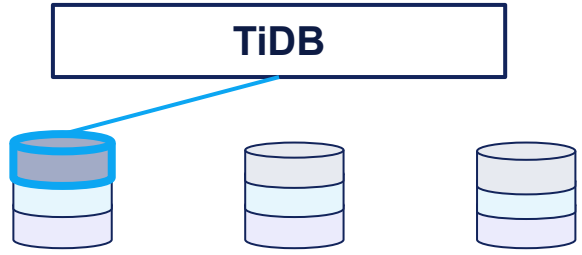


- PKがサロゲートキー（id）のため、
  取得したいデータが各TiKVノードに散らばっている } CPU負荷高め
- Indexからのアクセスのため2ホップ

PK

```
CREATE TABLE `user_action_logs` (
  `id` varchar(255) NOT NULL,
  `user_id` varchar(255) NOT NULL,
  `log_data` json DEFAULT NULL,
  `created_at` timestamp NOT NULL,
  `updated_at` timestamp NULL DEFAULT NULL,
  PRIMARY KEY (`user_id`, `created_at`, `id`) /*T![clustered_index] CLUSTERED */
  UNIQUE KEY `unique_idx_id` (`id`),
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin;
```

user_id（PK）	created_at（PK）	id（PK）
100001	8/19 12:00	asdijasoja
100001	8/20 22:00	zmpkogkp
100001	8/21 5:00	haoahapji



取得したいデータを特定のTiKVノードにおさめやすいので、効率が良い→SLELECTの条件に即してPKを設計しておくことがポイント
※RDBMSと同じ

Readの観点：Covering Index ※PK除いた全カラムのIndex

テーブルのKV

Primary Key - row data

id（PK）
1
2
3
4

user	department	created at
やまだ	AAA	4/1 12:00
いとう	AAA	4/1 22:00
さとう	BBB	4/2 5:00
いしだ	BBB	4/2 13:00

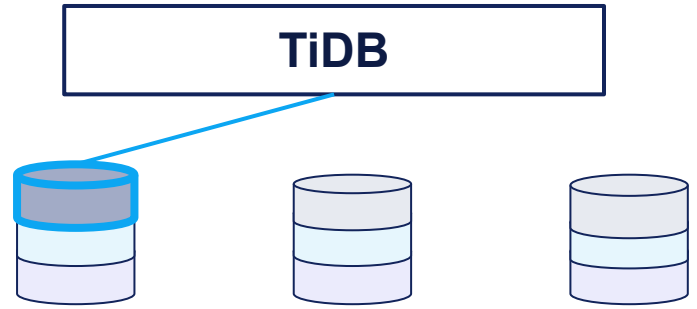
IndexのKV → こっちだけを見る

Index - Primary Key

user	department	created at
やまだ	AAA	4/1 12:00
いとう	AAA	4/1 22:00
さとう	BBB	4/2 5:00
いしだ	BBB	4/2 13:00

id（PK）
1
2
3
4

```
CREATE TABLE `user_action_logs` (
  `id` varchar(255) NOT NULL,
  `user_id` varchar(255) NOT NULL,
  `log_data` json DEFAULT NULL,
  `created_at` timestamp NOT NULL,
  `updated_at` timestamp NULL DEFAULT NULL,
  PRIMARY KEY (`id`) /*T![clustered_index] CLUSTERED */,
  INDEX idx_covering(`user_id`,`created_at`,`updated_at`,`log_data`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin
```



PKの工夫同様、取得したいデータを特定のTiKVノードにおさめやすい
→ デメリットはストレージが２倍になる（テーブルとIndexのKV）

OLTPチューニングの研究進む

帯域（150万QPS）やレイテンシに対する見解

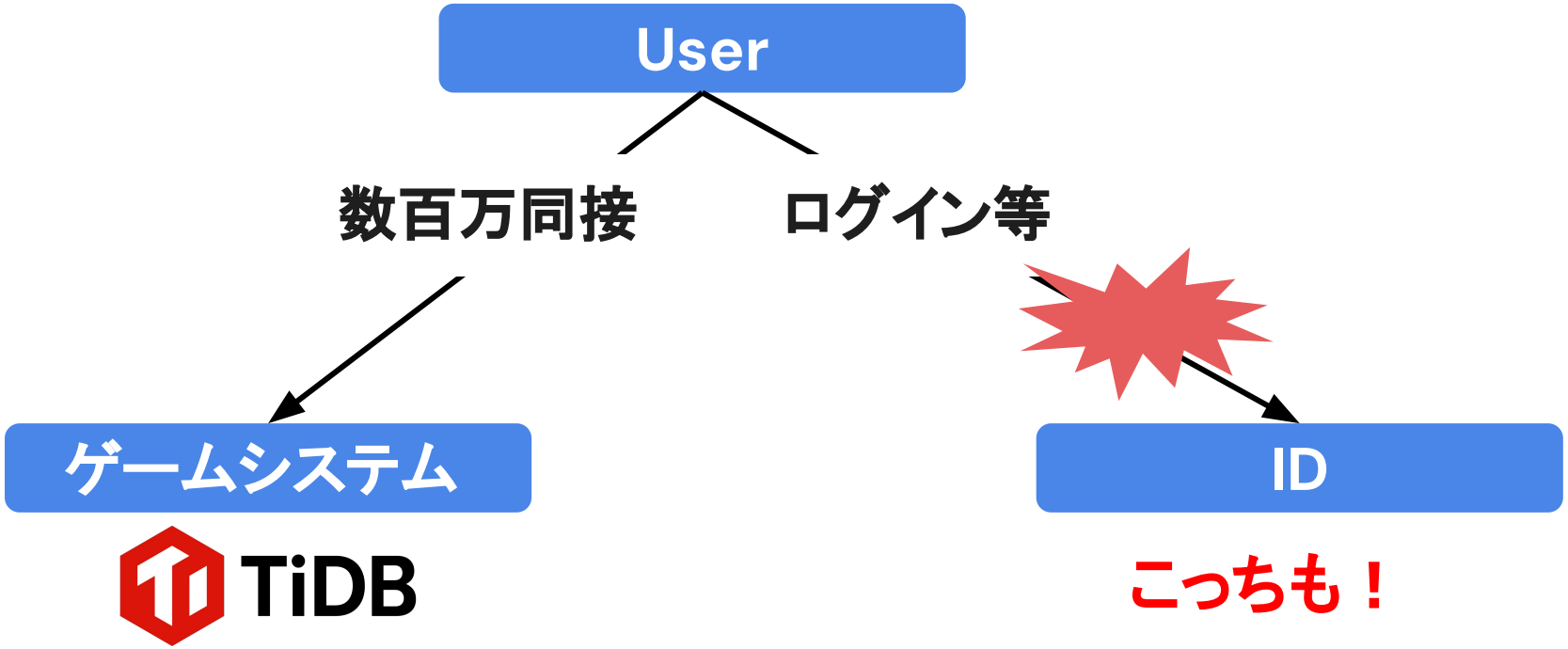
例えば、ワークロードとして Read heavyに効くパラメータとは？
（狙いは、レイテンシ減やリソース効率化を追及）

2024年には、TiKVのCPU使用率が高くなると Write latencyが上がる課題があり、
v8.0でstore-pool-ioパラメータのデフォルト値を変更 ： 0->1
=> Raft I/O タスクを処理するスレッドの許容数

https://docs.pingcap.com/tidb/stable/tikv-configuration-file#store-io-pool-size-new-in-v530

OLTPチューニングの研究進む

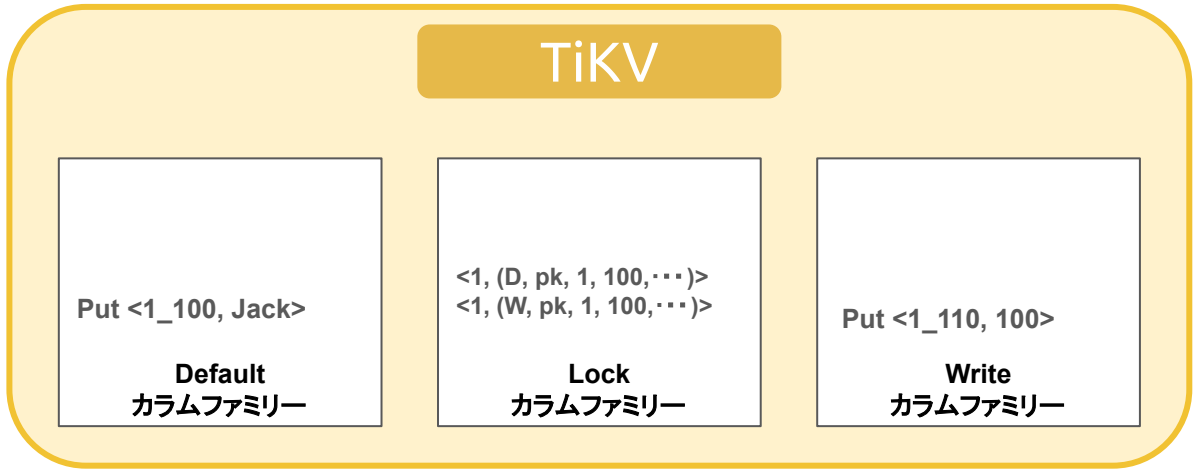
ゲーム業界トピック:ゲーム大規模化に伴って、 ID基盤での活用進む
（IDのせいでゲームが止まらない、課金・決済の内製化など）



更新ヘビーが起こす問題への対応

例えば、状態管理などのテーブルは更新が多く、 MVCCデータが多く溜まる

- TiKVは書込性能重視のため LSMTree(追記型）
- Write(update,delete)は基本的に追記
- ある程度の頻度（ compaction）で削除



- 履歴(MVCC)データが異常に多くなる
- 履歴データを含めた検索となる

-TiKV CPU増（コスパ）
-レイテンシ増

更新ヘビーが起こす問題への対応

TiDBとしての対応

1. GCで削除するの頻度を増やす
（parameter：compaction-filter->false）

GC in Compaction Filter

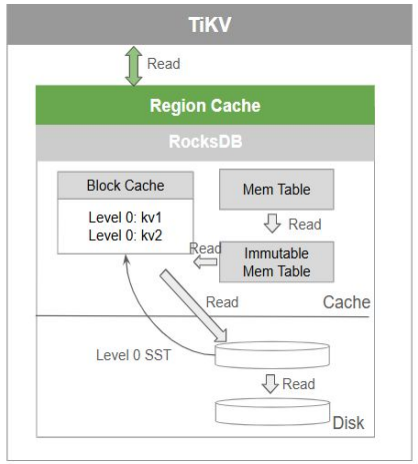
Based on the `DISTRIBUTED` GC mode, the mechanism of GC in Compaction Filter uses the compaction process of RocksDB, instead of a separate GC worker thread, to run GC. This new GC mechanism helps to avoid extra disk read caused by GC. Also, after clearing the obsolete data, it avoids a large number of left tombstone marks which degrade the sequential scan performance.

The following example shows how to enable the mechanism in the TiKV configuration file:

```
[gc]
enable-compaction-filter = true
```

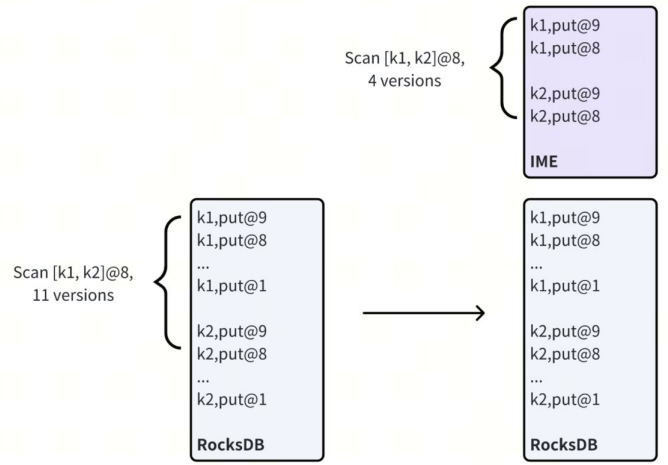
2. TiKVの中にキャッシュを作り最新のデータのみ読む機能
(in-memory cache)

8.4 New



3.TiKV内に、MVCC用のインメモリエンジンを搭載最新のMVCCバージョンをメモリにキャッシュ
(in-memory-engine)

8.5 New

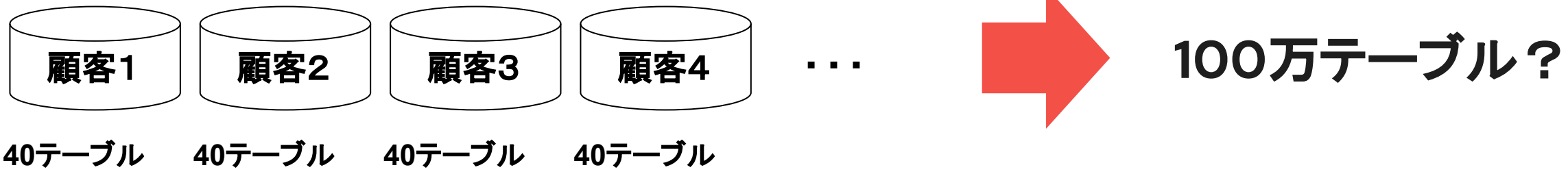


多テーブルが起こす問題への対応

B2B2C/つぎ足すシステムはテーブルが多くなりがち

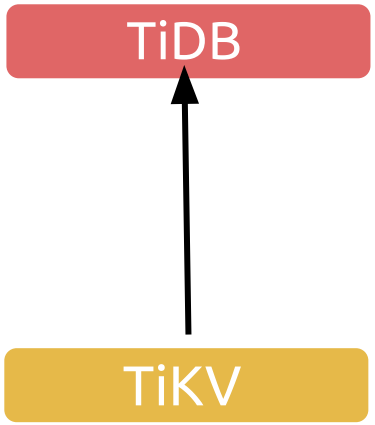
https://developer.atlassian.com/platform/forge/storage-reference/sql/

> Forge SQL is currently based on a self-hosted TiDB ↗ implementation, and we impose limitations on how the database can be used to support Forge use cases.
>
> We recommend that you avoid designing applications with TiDb-specific functionality.



多テーブルが起こす問題への対応

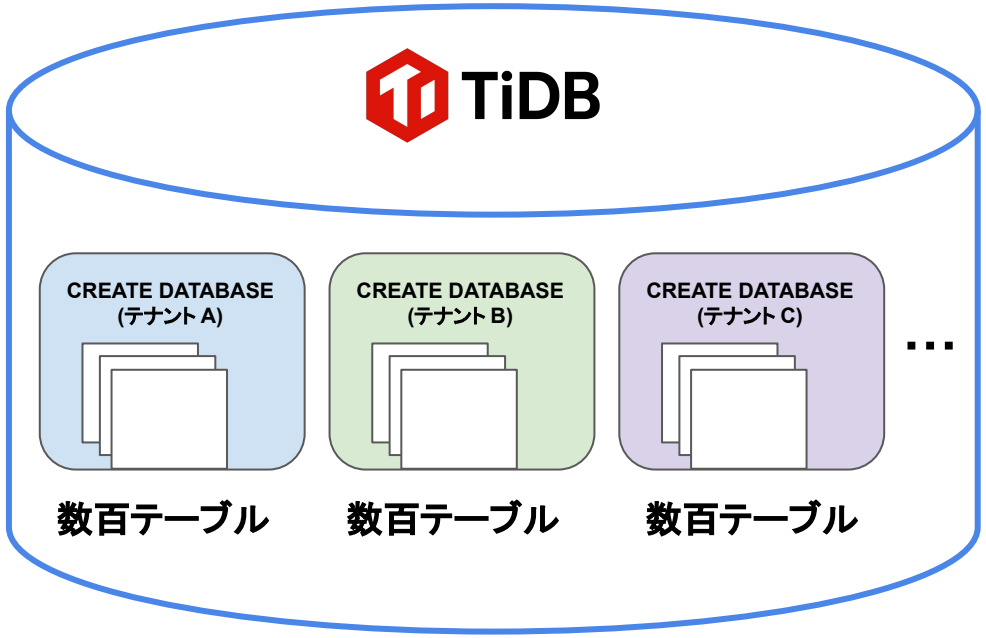
統計情報のロードに時間がかかる
（起動が遅い、クエリが非効率に）



[起動時]
統計情報の一部を載せて起動

[クエリ時]
統計情報（残り）

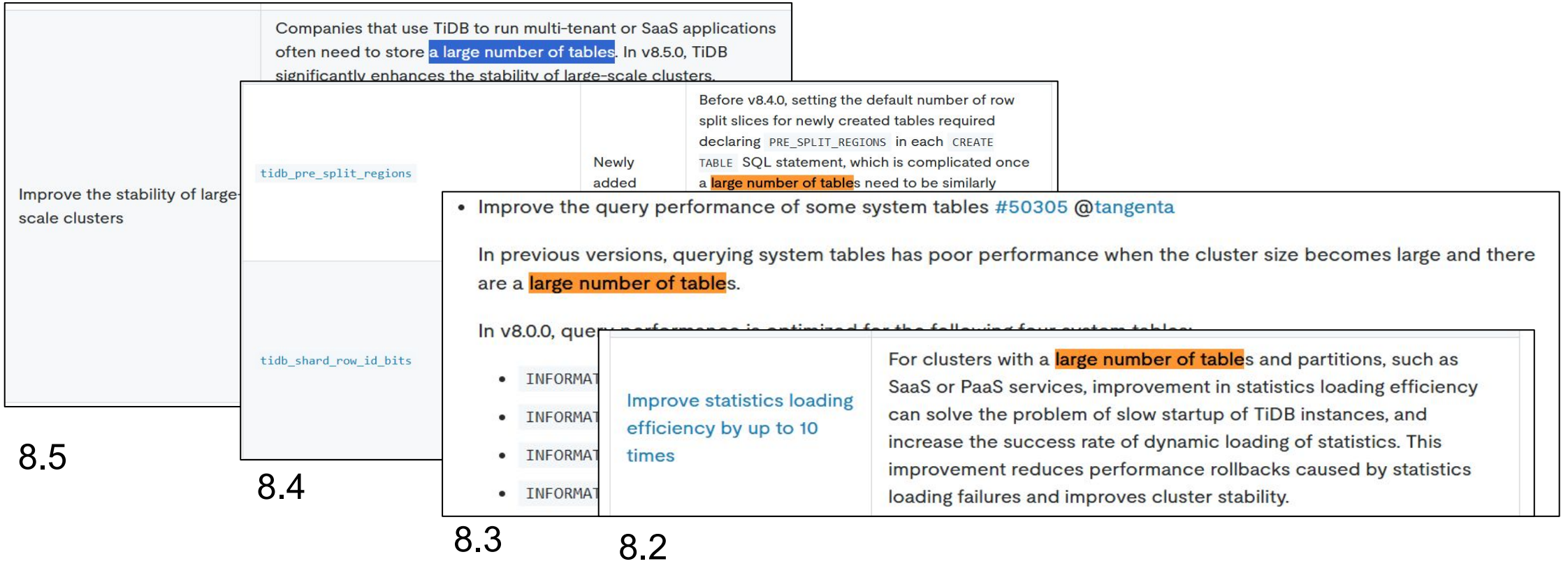
Information Schemaのレスポンス低下
（show~のせいでmigrationが遅い。）



多テーブルが起こす問題への対応

TiDBとしての改善

1年かけてエンハンスを続けてきました（統計情報のロード性能など）

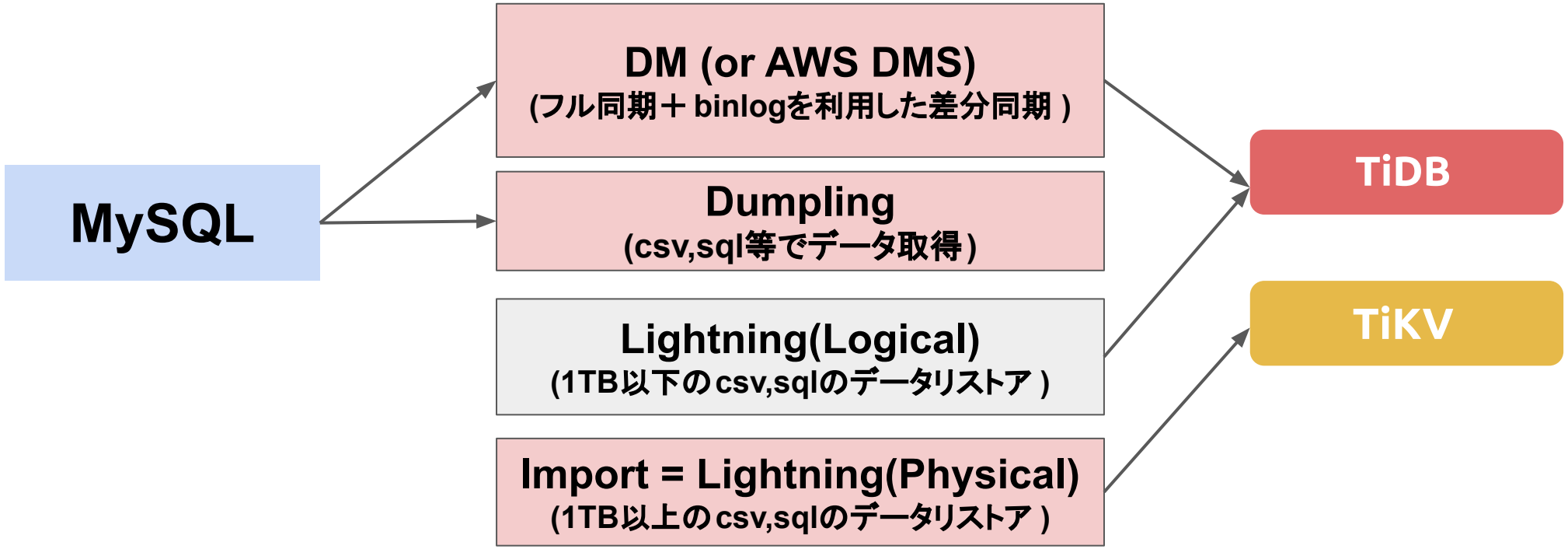


移行の高度化

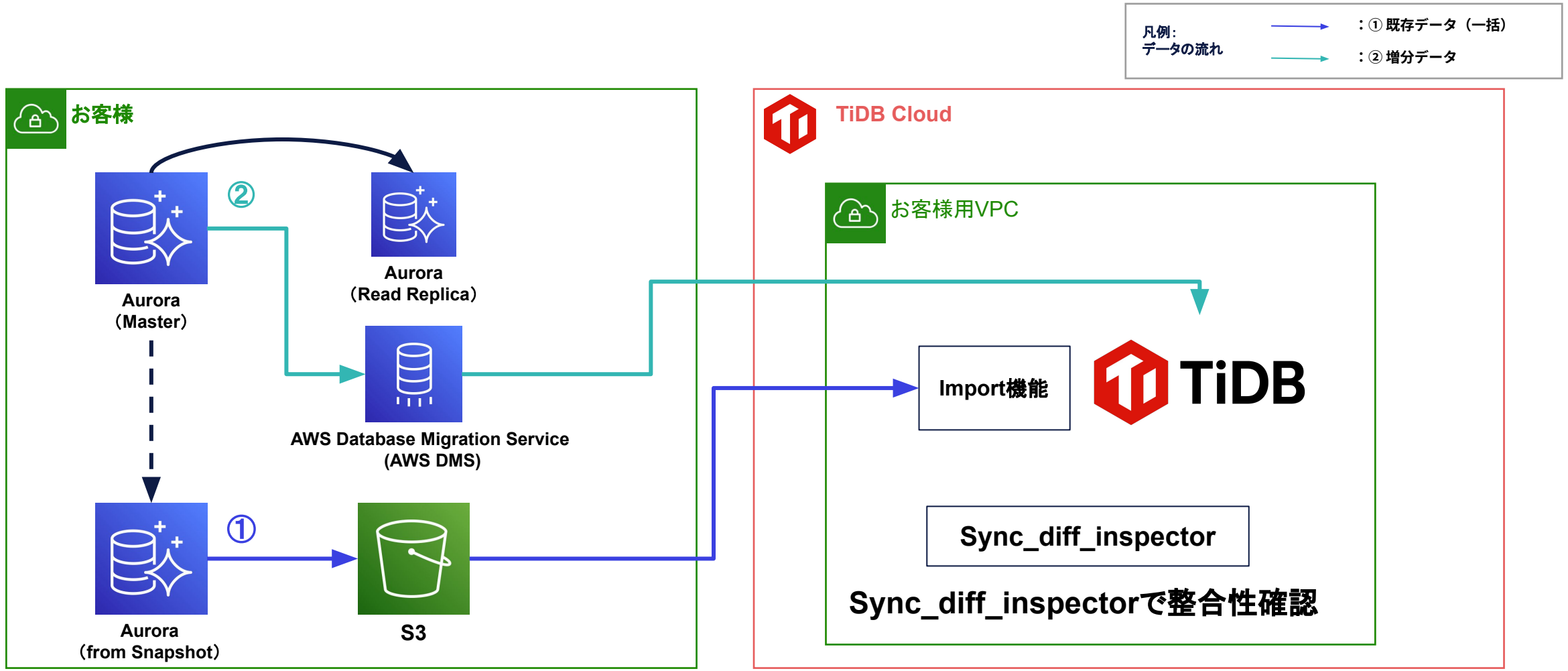
様々な移行方法

A. TiDBのエコシステムツールを利用

B. Dual Write (ノーダウンタイム)



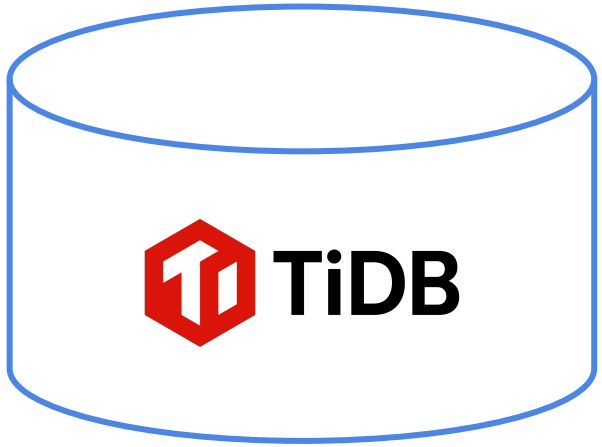
よくある移行手段と全体構成



移行の高度化

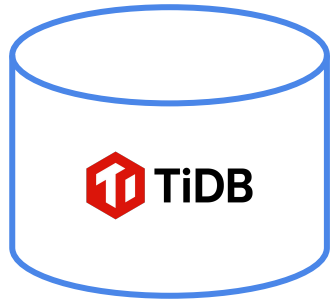
移行後のチューニングをセットで考えたキャパシティプラン

移行直後



CPU増・スロークエリ発生
- Indexが足りない
- クエリが非効率

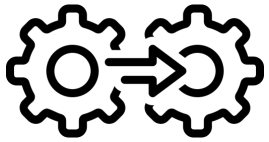
移行後



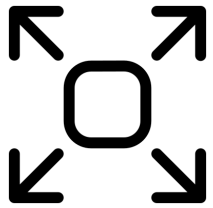
オンライン
スケールイン /ダウン

TiDBがあなたにもたらすもの。それは

TiDB



MySQL Compatibility



Horizontally Scaling

Zero Downtime

HTAP

Resilient

あなたのDBもキャリアもスケールさせましょう。
ブースでお待ちしてます

Ready for the Next Scale?