

数据来源：数据库产品上市商用时间



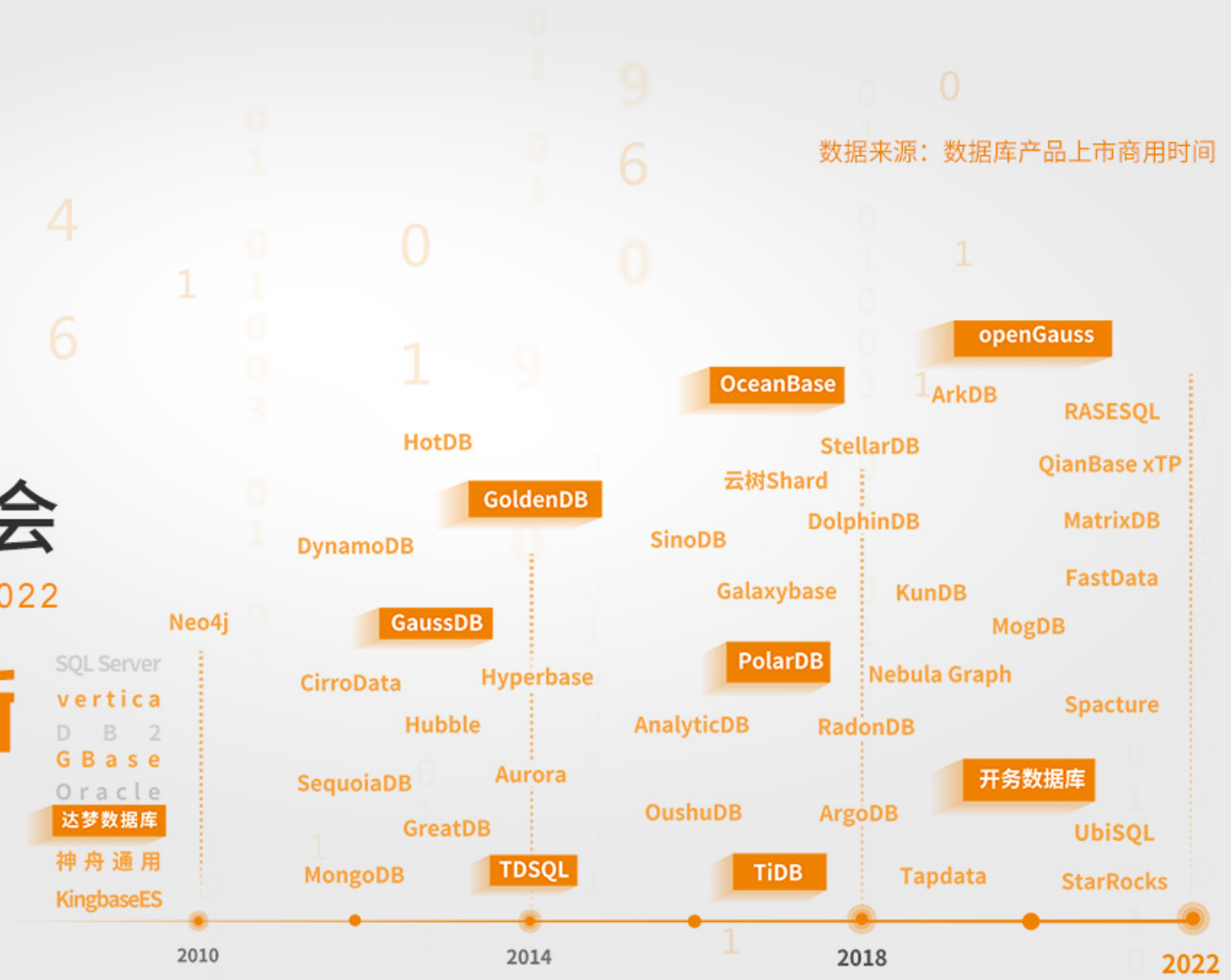
# 第十三届中国数据库技术大会

DATABASE TECHNOLOGY CONFERENCE CHINA 2022

## 数据智能 价值创新



线上直播 | 2022/12/14-16



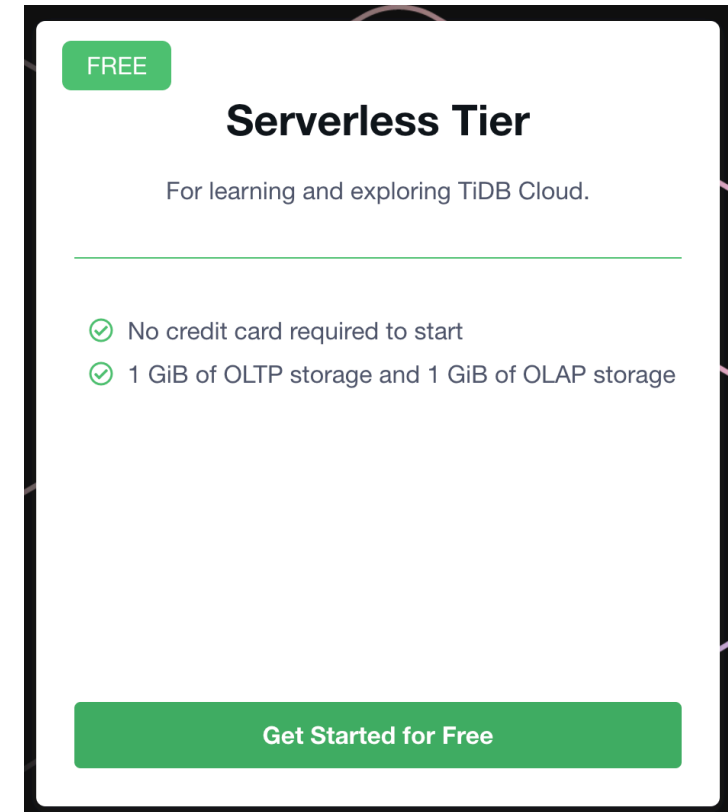
# Dive Deep Into TiDB's Columnar Storage Engine

韦万 @PingCAP

# TiDB Introduction

**TiDB** is an open-source NewSQL database that supports HTAP workloads. It is MySQL compatible and features horizontal scalability, strong consistency, and high availability.

And there is a serverless free TiDB available for every developers at [tidbcloud.com](https://tidbcloud.com).  
Get ready in 20 seconds!



# Agenda

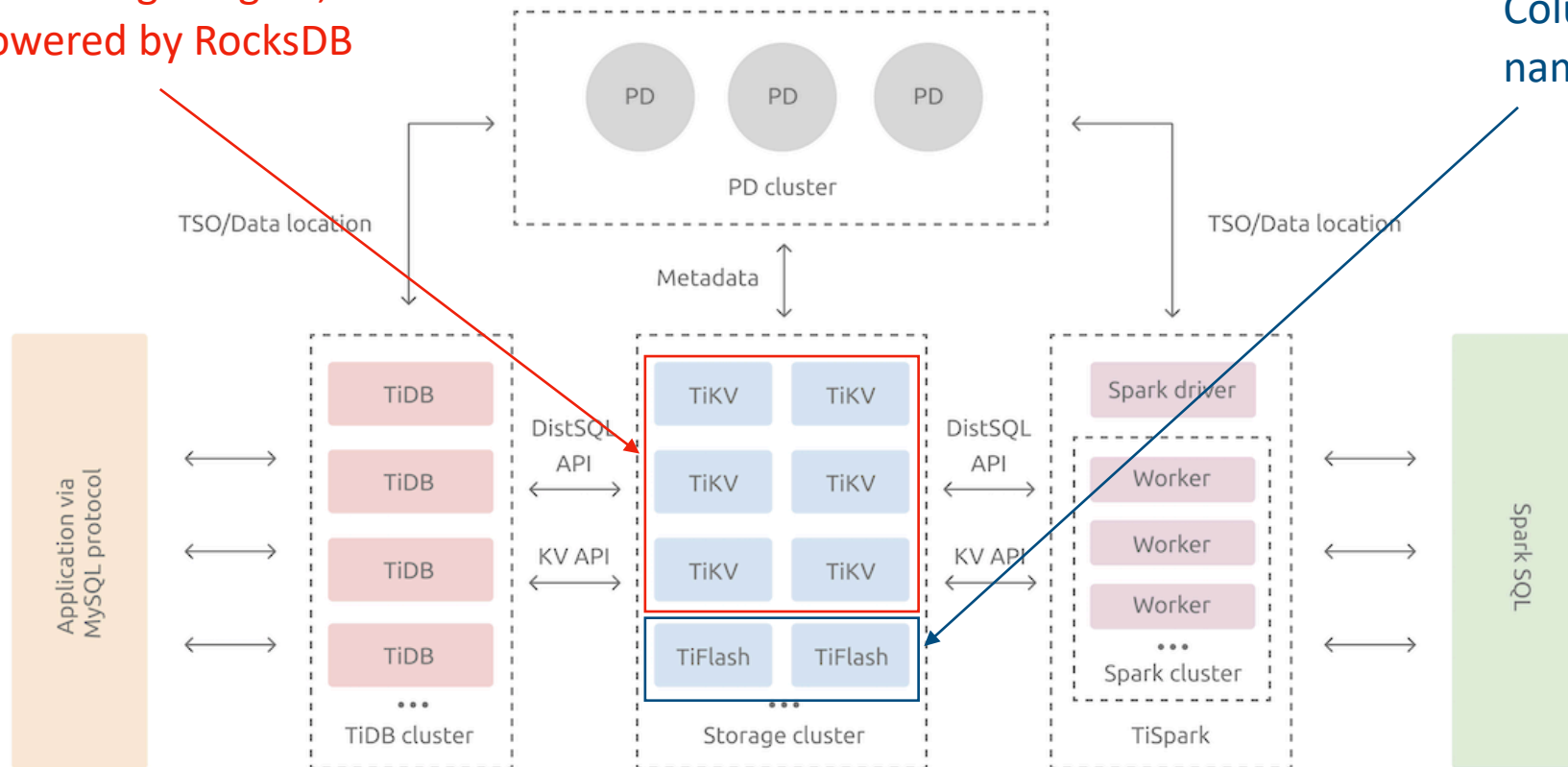
- How does TiDB handle HTAP workloads
- Dive deep into Delta Tree, the columnar storage engine of TiDB
- Cloud-native evolution

How does TiDB handle HTAP workloads?

# TiDB Introduction

Row storage engine,  
powered by RocksDB

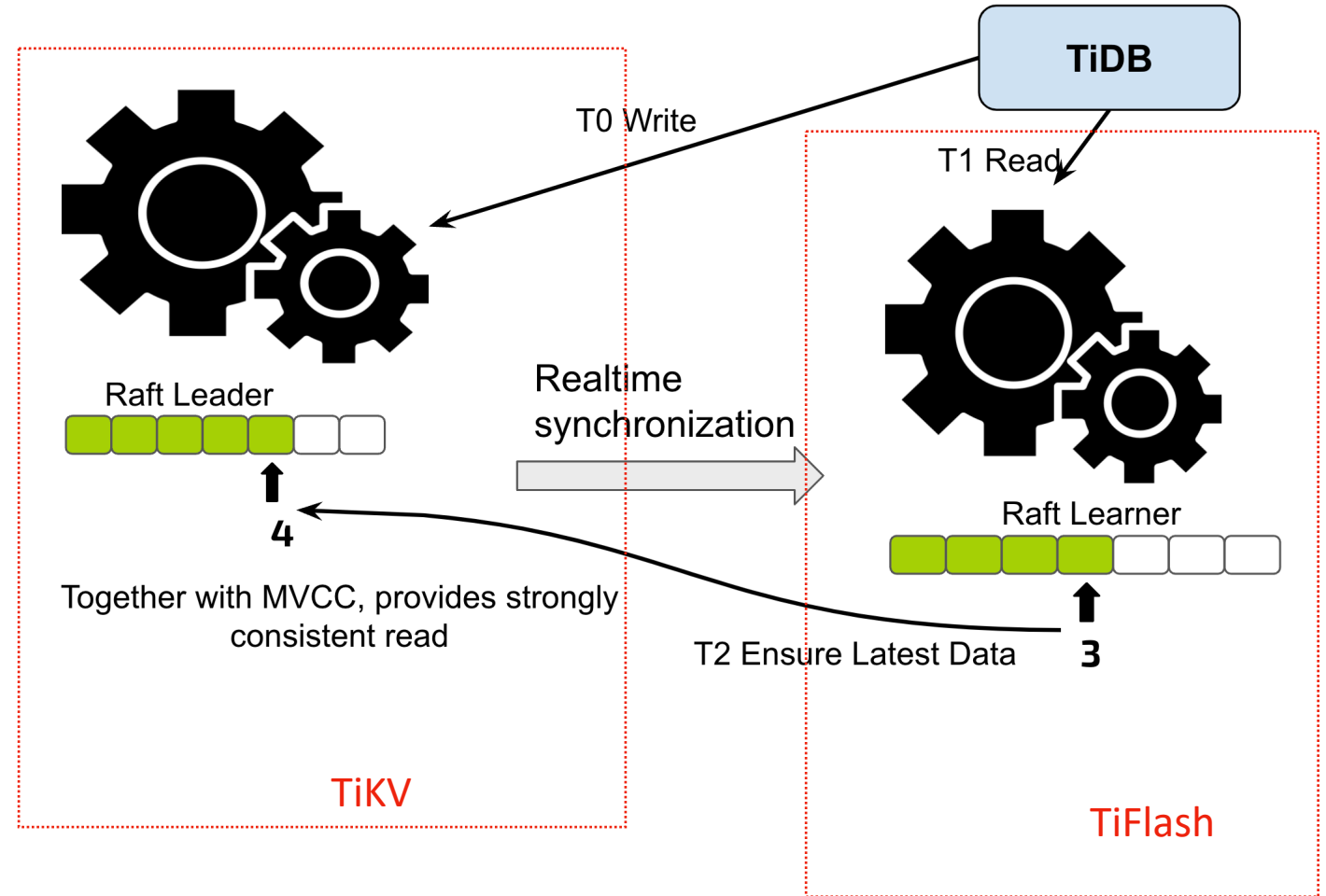
Columnar storage engine,  
named as Delta Tree



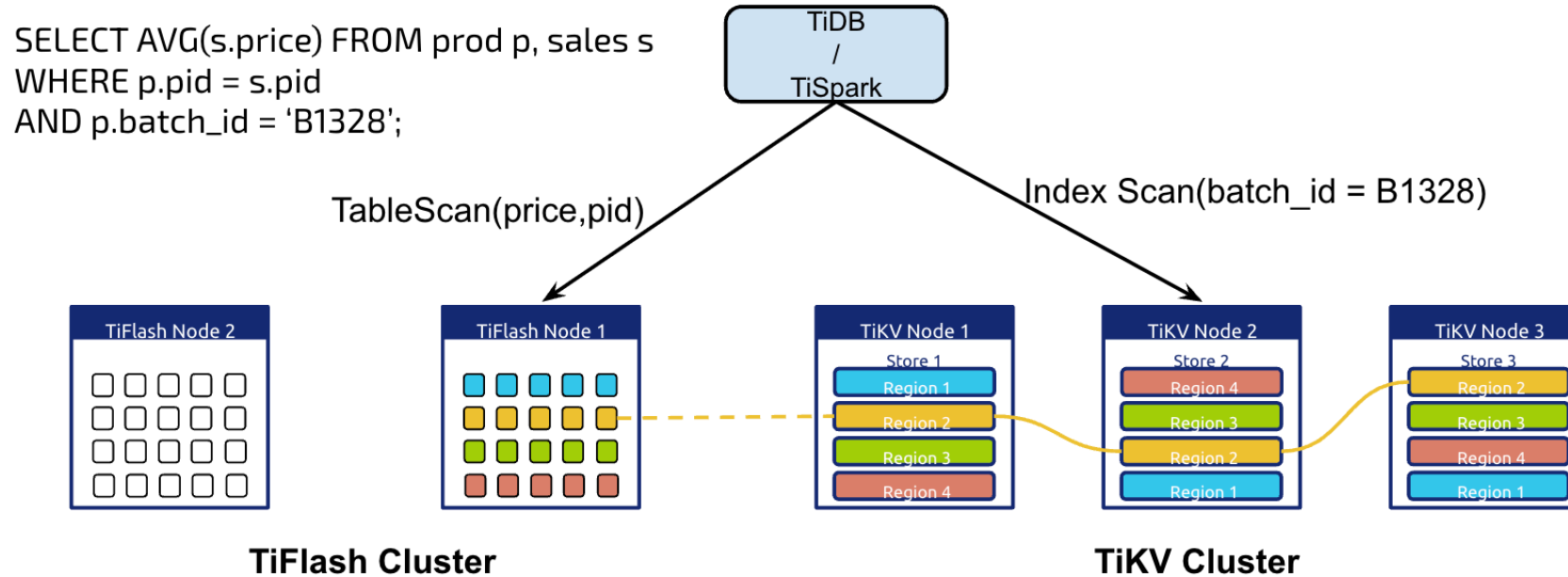


# Two storage engines work together to empower HTAP ability

- TiKV is for OLTP, and TiFlash is for OLAP
- TiKV synchronizes data updates in real-time to TiFlash, via raft protocol.
- Reads on TiKV and TiFlash are strong consistent, with no delay.
- Read consistency is guaranteed by learner-read mechanism.



# Optimizer utilize both column and row storages.





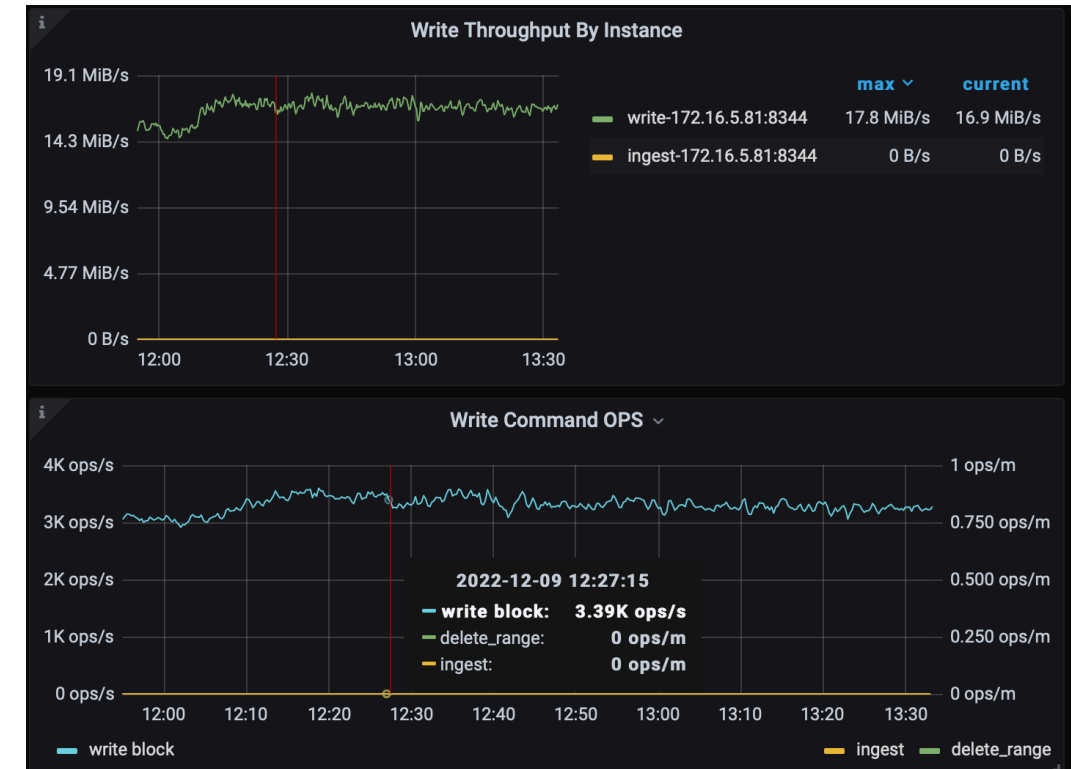
Dive deep into Delta Tree, the columnar storage engine of TiDB

# Columnar Storage Engine

Parquet is handy, isn't it enough? No!


We need **real-time update** with **high OPS**.

We need **MVCC**, to support **transactional snapshot isolation**.

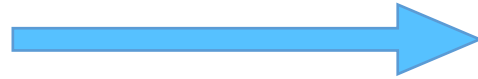


A typical write throughput of a TiFlash node

# The basic ideas of Delta Tree Storage engine

- Transform updates and deletes into **upserts**.
- MVCC ability get 

pk	ts	del	col_a	col_b
1000	50	0	好	当
1000	60	0	雨	春
2000	20	0	知	乃
2000	22	1	时	发
3000	70	0	节	生

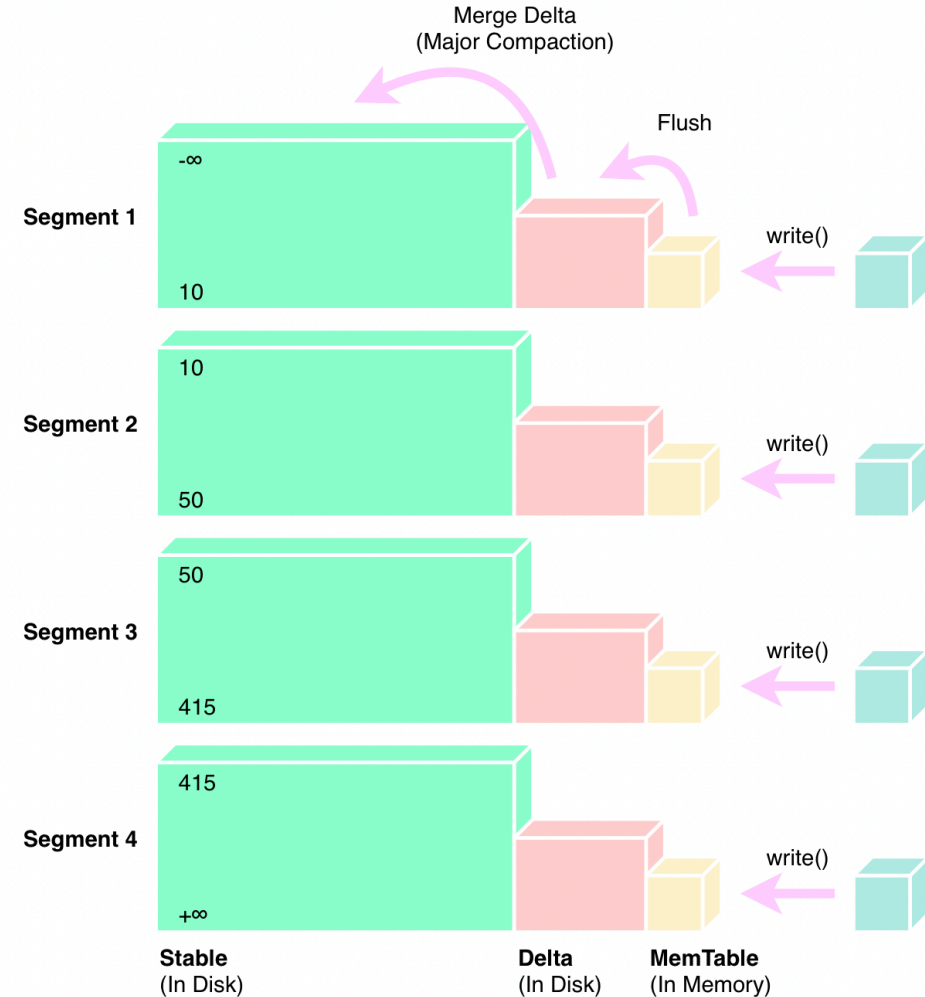


select \* from T with read\_ts = 65

pk	ts	del	col_a	col_b
1000	60	0	雨	春
3000	70	0	节	生

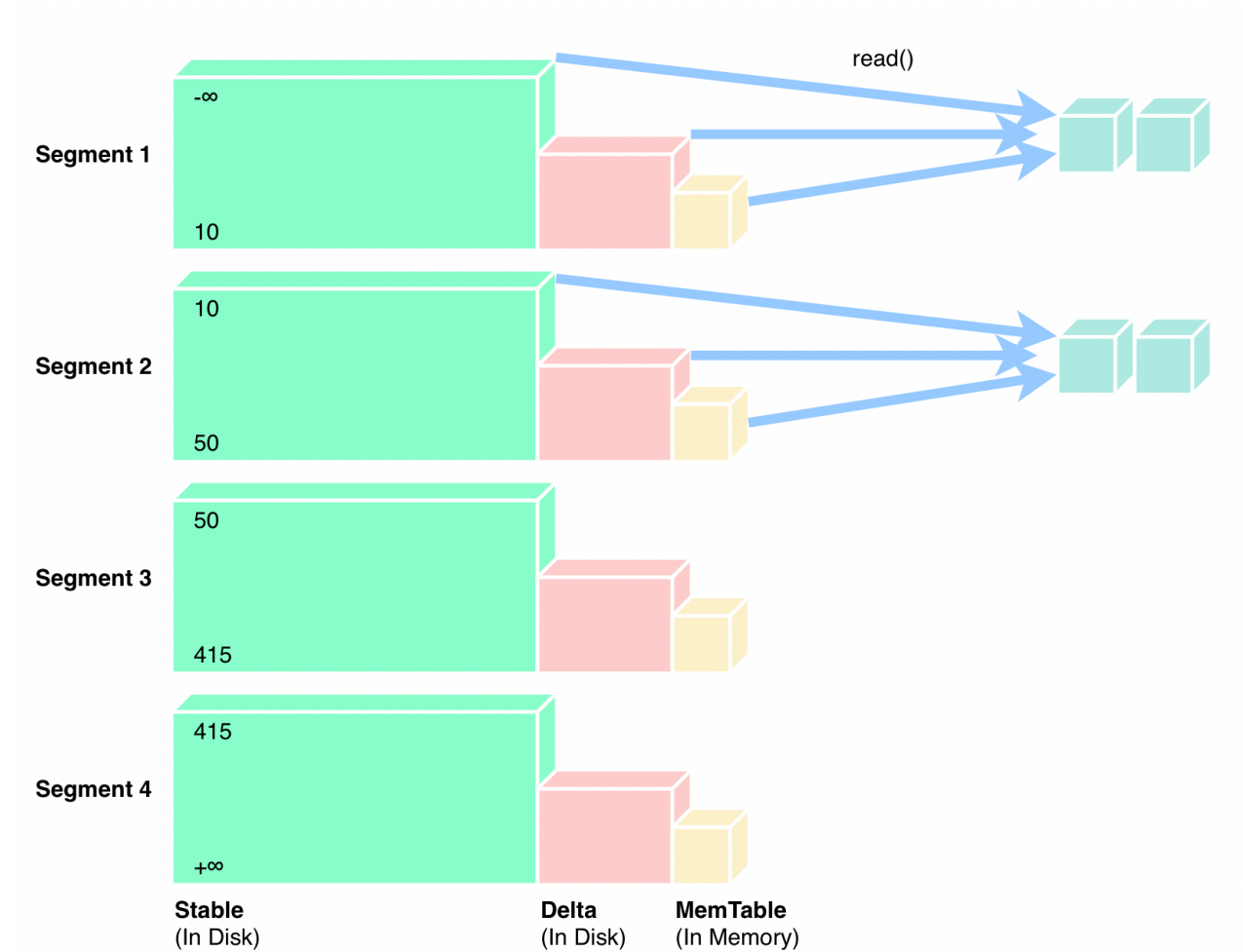
# The basic ideas of Delta Tree Storage engine

- Split the data by PK range into many Segments
- Each segment is a small LSM-Tree, with only 2 layers
  - Delta layer and write cache, i.e. memtable
  - Stable layer



# The basic ideas of Delta Tree Storage engine

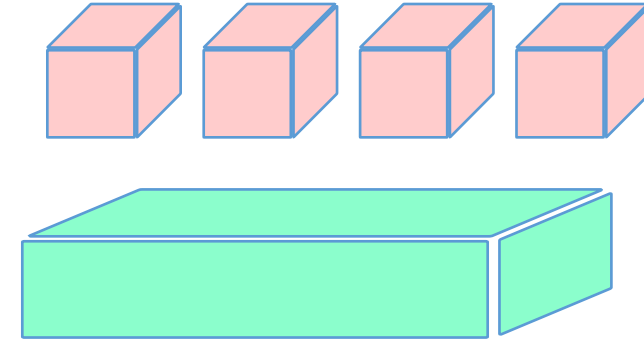
- Segments are read in parallel, naturally.
- Fewer layers brings faster reads
  - Fewer layers to sort merge
- Segments are compact in parallel in separate ranges, brings smaller write amplification



# How to store those column files?

**Drawback:** many small fragment column files in delta layer

- Column files in delta layer: 64KiB ~ 16MiB
  - Millions per node
- Column files in stable layer: 128MiB ~ 1GiB
  - Thousands per node
- Many meta data objects needed to be persisted to disk
  - Millions per node

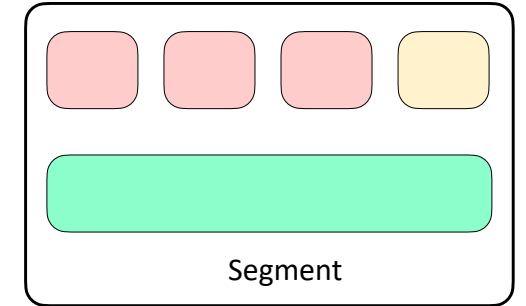
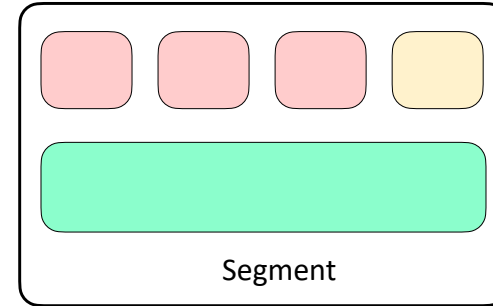
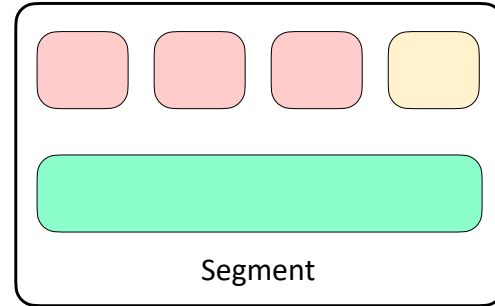


We persist everything in Delta Tree into PageStorage

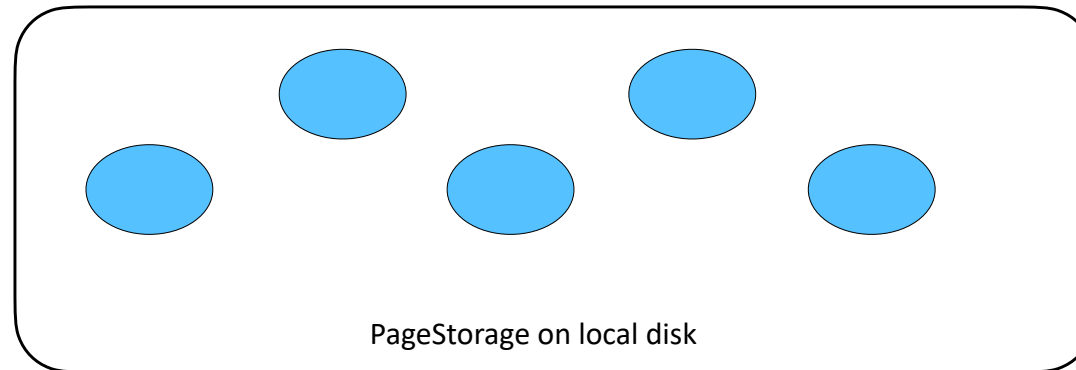


# Storage structure of DeltaTree

- All storage engine logic is implemented in this layer
- Meta & Cache
- In memory



- All IO logic is implemented in this layer
- All data are serialized into Pages and stored in PageStorage on disk
- PageStorage is a local object storage

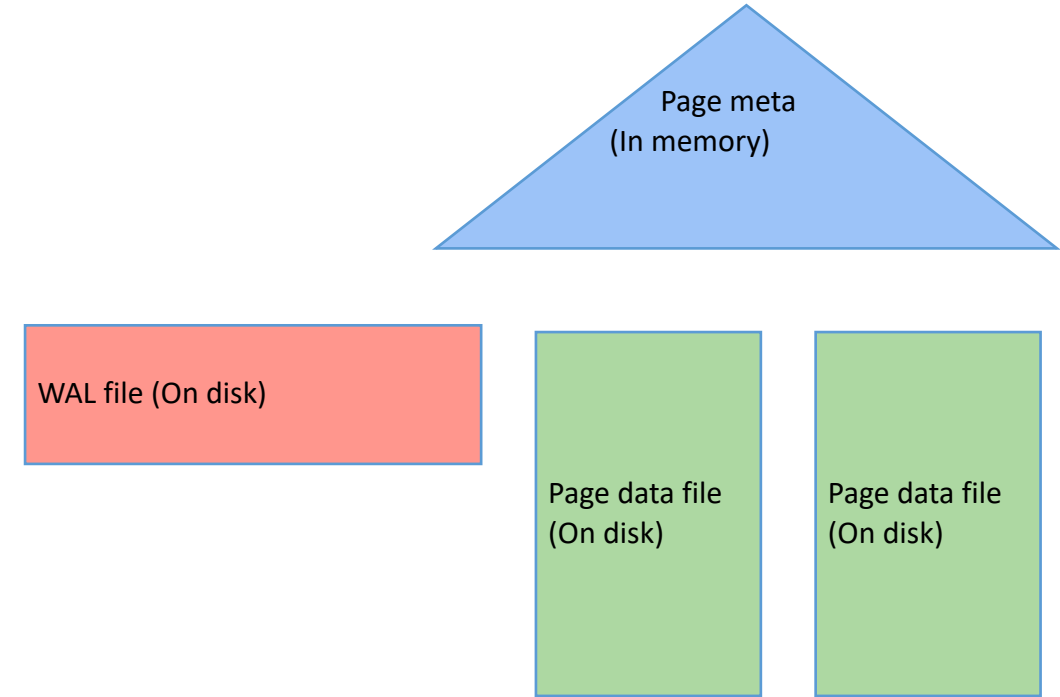


# Key features of PageStorage

1. A key-value storage. Can store a large number of various sized pages
  - Millions of pages
  - PageId (i.e. key) is int64(will support binary later), and value is binary object(i.e. Page)
2. Support write batch
  - To group several writes into an atomic write
3. Support snapshot
  - Delta Tree heavily depends on it to support snapshot read
4. Support reference pages
  - Just like hard link in a file system. Used by Segment split and others.
5. Support external pages
  - The real page content is store at somewhere else (e.g. a regular file), but managed by PageStorage
6. Low read/write latency, high read/write throughput

# Basic ideas of PageStorage

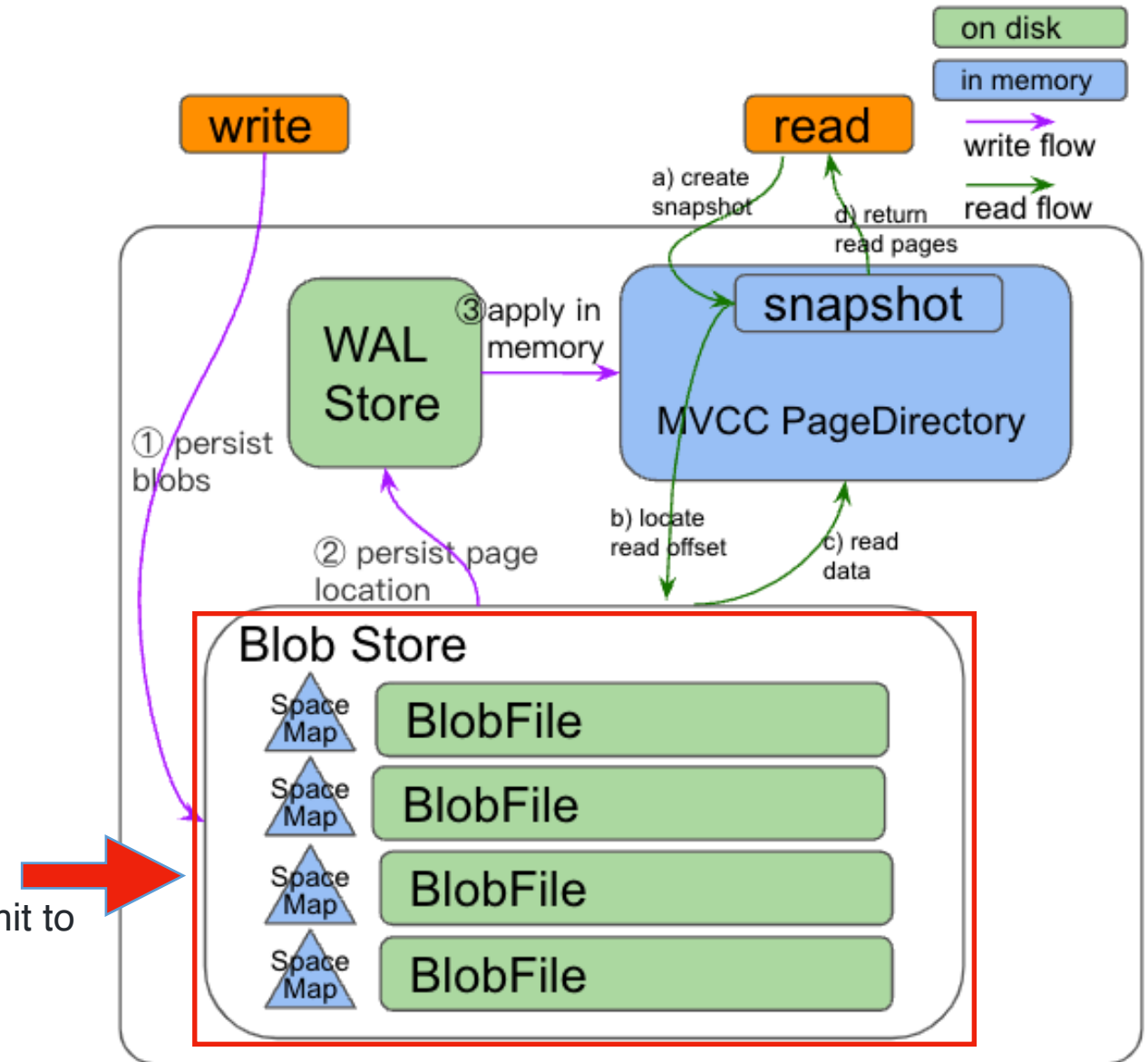
- Store the page meta in memory.
  - Key, file id, data offset, size, checksum
  - Fast read. At most one IO to read a page
  - The memory consumption is limited
- WAL file only stores Page meta
  - Easy to support multiple writable data files
  - Fast to do compaction. WAL file is small, and data files rarely need to clean up.
  - First write page data to data file, then commit to WAL file



# PageStorage write process

Blob Store is where page data actually stored

- Blob File
  - Store pages data
  - Support multiple disk
- Space Map
  - R-B tree
  - Records the free space in BlobFiles
- Write process:
  1. Select a Blob File to write
  2. Find a suitable free space in Space Map
  3. Write into the selected Blob File
  4. Write meta data into WAL, commit to disk
  5. Update meta data in MVCC PageDirectory, commit to memory

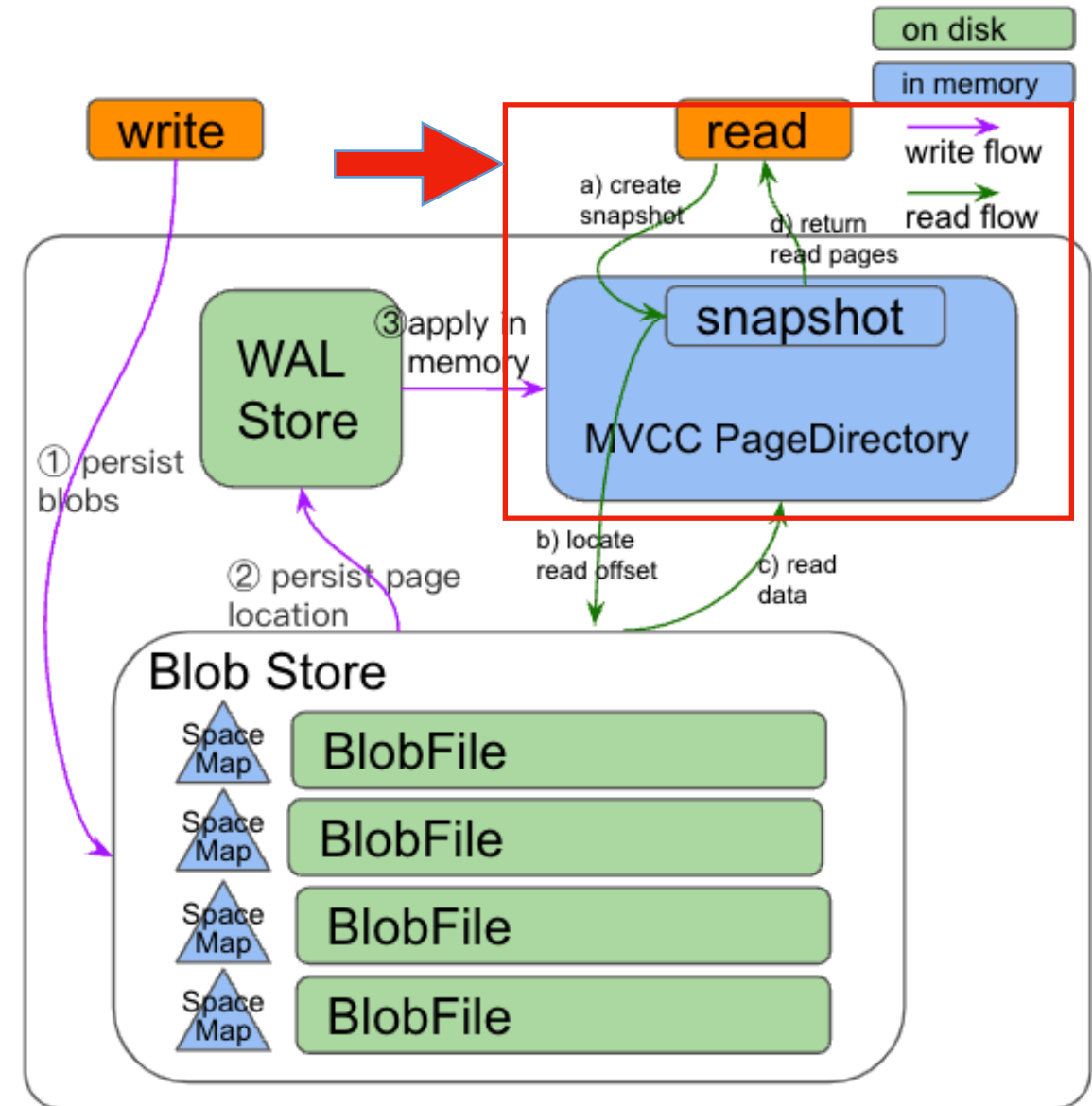


# PageStorage read process

PageDirectory stores all info required to access pages

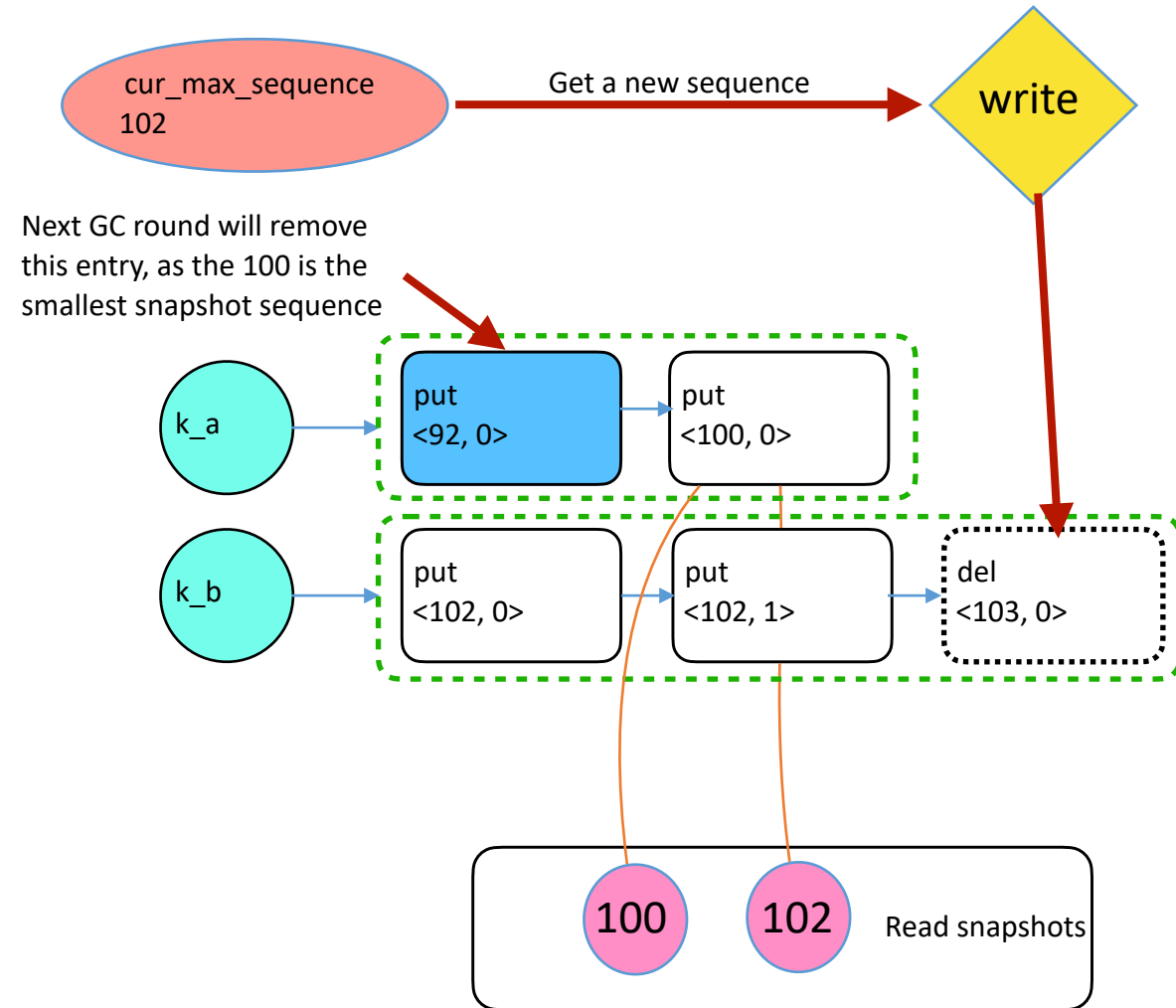
- Read process:

1. Get a snapshot of PageDirectory
2. Collect required info from the snapshot, including each pages' file\_id, offset, etc.
3. Do reading on Blob Files



# PageStorage PageDirectory & GC

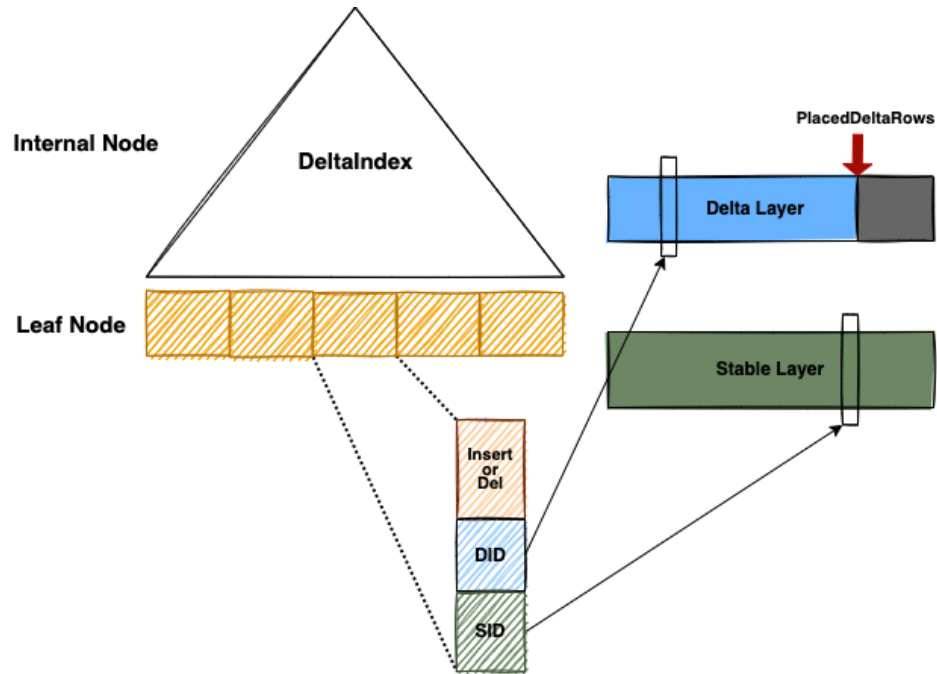
- PageDirectory is an in memory sorted map
  - Key is PageId, value is the page's edit entry list
- Update operations:
  1. Put
  2. Delete
  3. Reference
  4. Put External
- A version <sequence, epoch> attaches to each update operation
  - sequence - operation counter
  - epoch - GC counter
- GC periodically remove useless entries
  - The entries with smaller sequence than smallest snapshot sequence
- GC also move Page data to another Blob File, to remove the low use ratio Blob File
  - epoch increase by 1 after move



Some tricks to improve query performance of Delta Tree

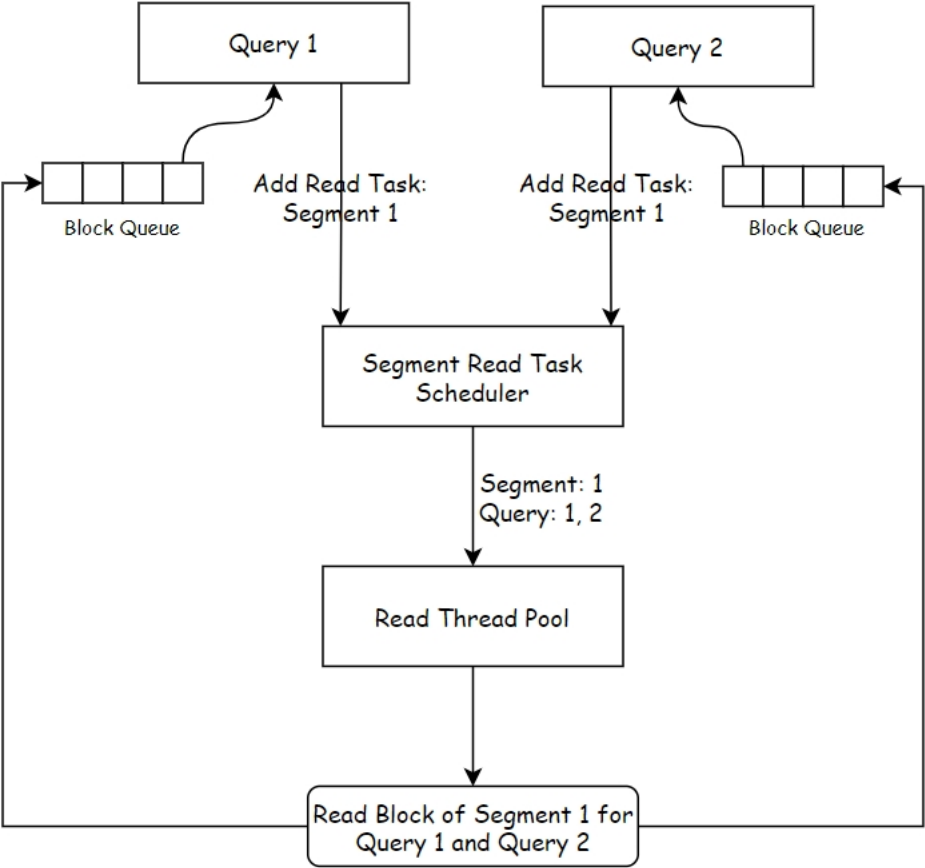


# Use DeltaIndex to accelerate scan speed



The scan speed of DeltaTree is 3x of ClickHouse in SELECT ... FINAL

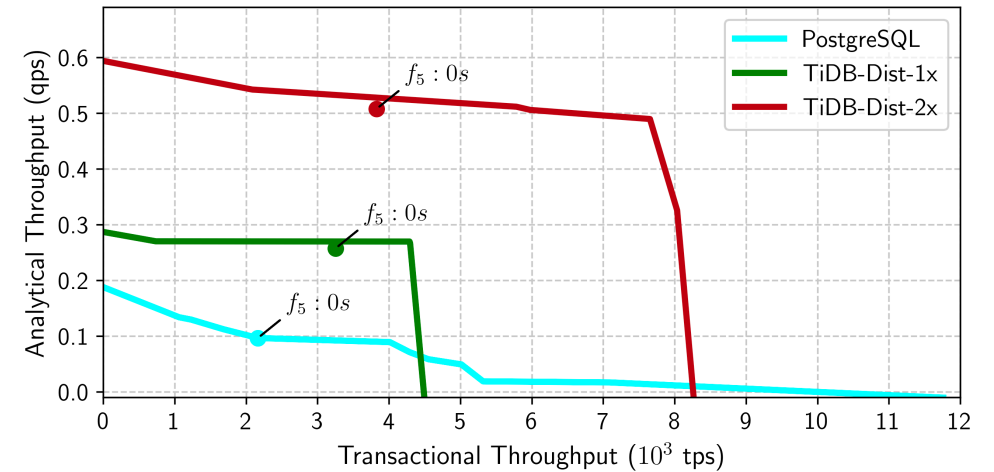
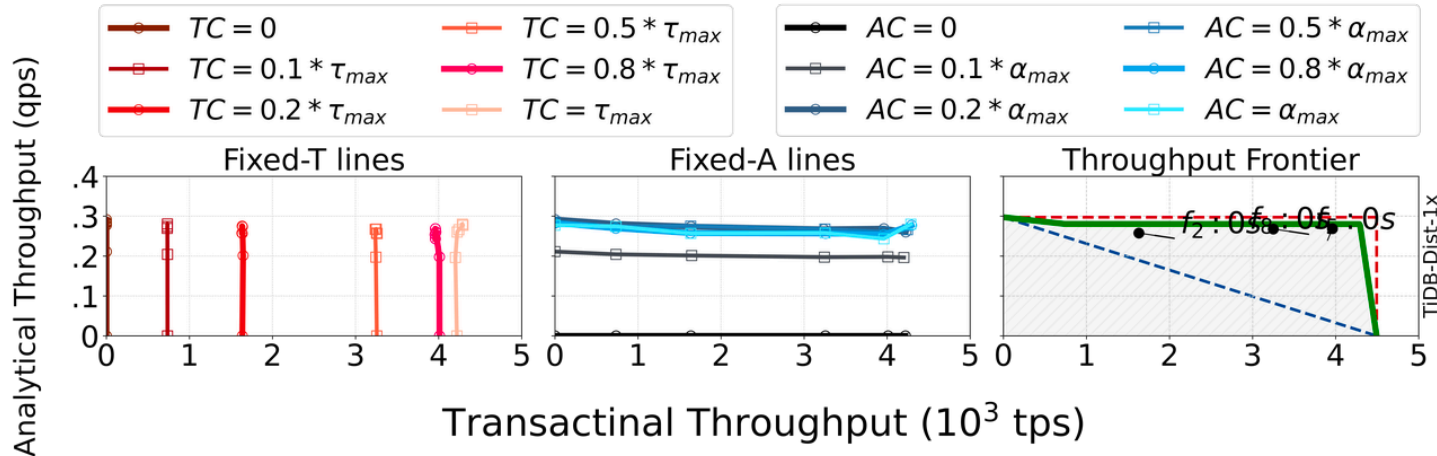
# Data Sharing



How does the combination of TP and AP work?

# TiDB HTAP performance in 6.x

- The avg latency of AP queries in typical HTAP workloads was decreased by 30% to 50% comparing with 5.x.
- Better isolation between OLTP and OLAP workloads ([HATrick Bench](#))
- Better scalability

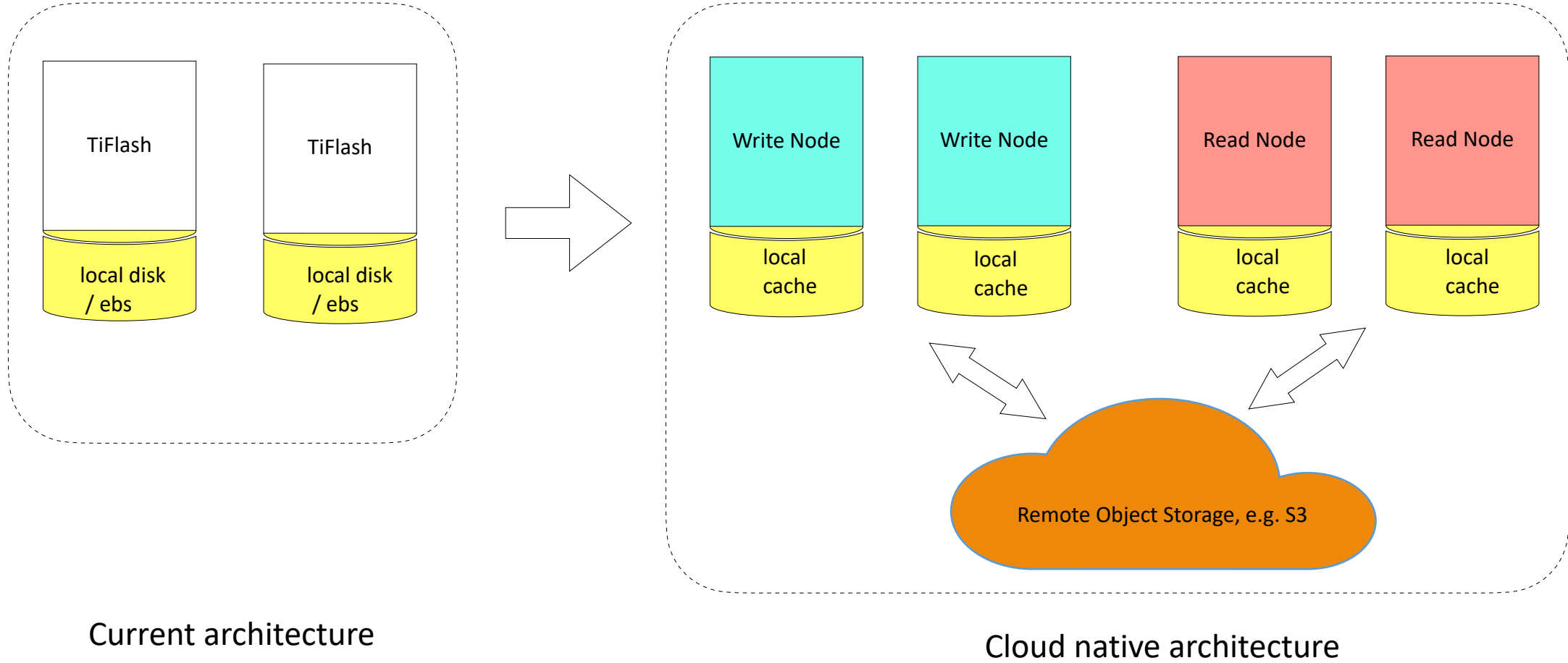


The next move: cloud-native evolution

# What is the benefit of cloud-native?

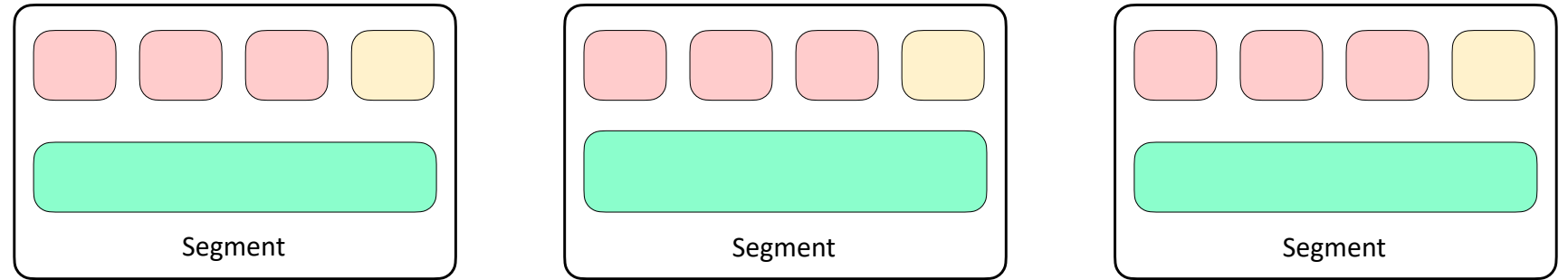
- Scale fast, real fast
- Higher availability
- Pay as you go

# The evolution to cloud-native (coming soon)

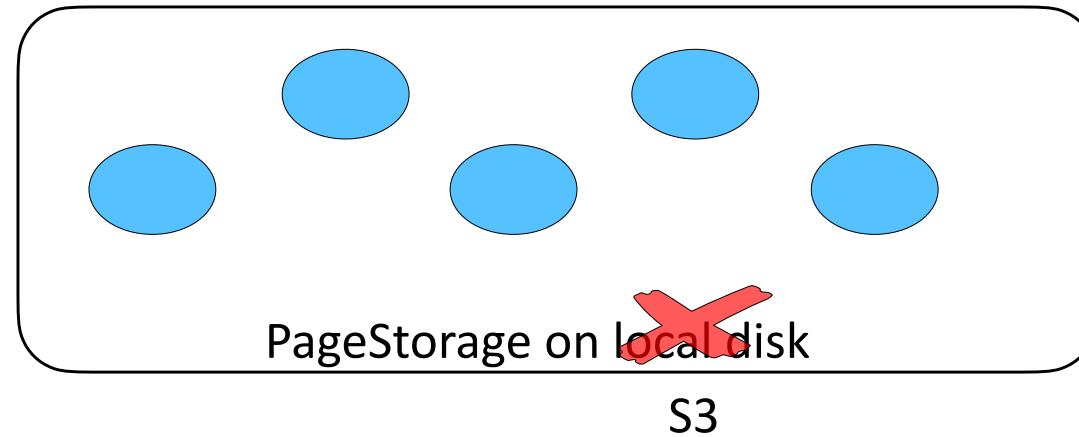




# The evolution to cloud-native (coming soon)



- Only need to change the storage location of PageStorage
- Local -> Remote (S3)



# What is the benefit of cloud-native?

- Scale fast, real fast
  - Read node is stateless. Scale instantly!
  - Write node only host small data, and can be recover fast. Scale fast!
- Higher availability
  - S3 provides better HA
  - Scale fast leads to better HA
- Pay as you go
  - Compute pool (read nodes)
  - Shared storage nodes by multiple clusters (write nodes)
- Cloud native != cloud only
  - Both TiDB Cloud and on premise cluster benefit from cloud-native cluster

# THANKS

