

Apache IoTDB

成員分工

- 引言與架構解析：詹茗偉、陳肇廷、鄭睿宏、賴光禹
- 本機安裝：張發貴
- 相近技術的比較：黃思璇、黃甄浥
- 結語 & demo 影片：潘煜智
- 投影片製作：張發貴、黃思璇、黃甄浥

片 紅

- Introduction
- 架構解析
- 實驗記錄與心得
- Evaluation and comments
- 總結

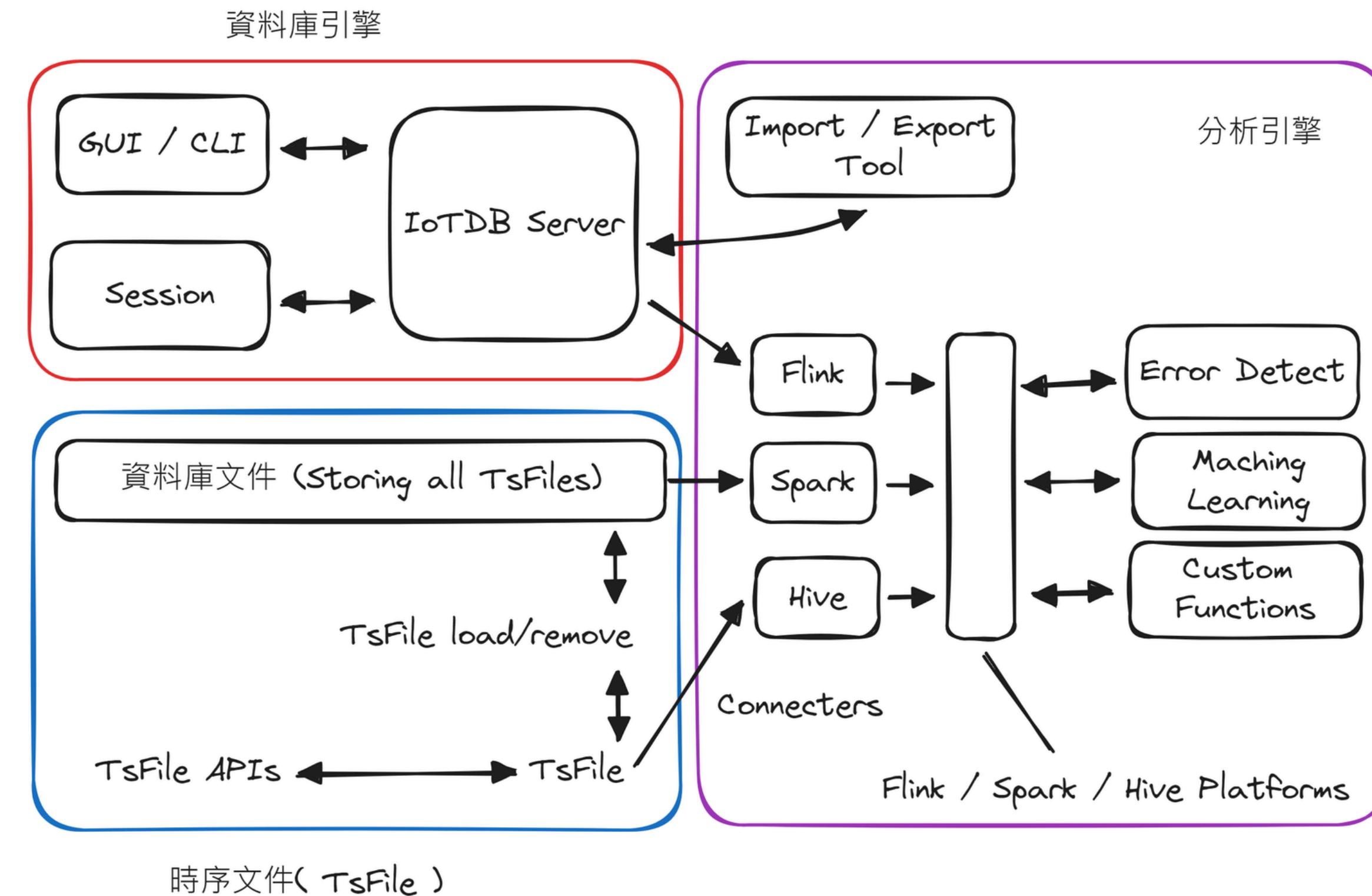
Introduction

核心概念

- 提供一個可以很好操作 time series data 的 Database system
- 屬於 Time Series DBMS (TSDB)
- 為各種物聯網設備傳感器設計，可以支援大量包含時間戳記的數據
- 基於時序文件 TsFile 、資料庫引擎，與分析引擎所建構的資料庫



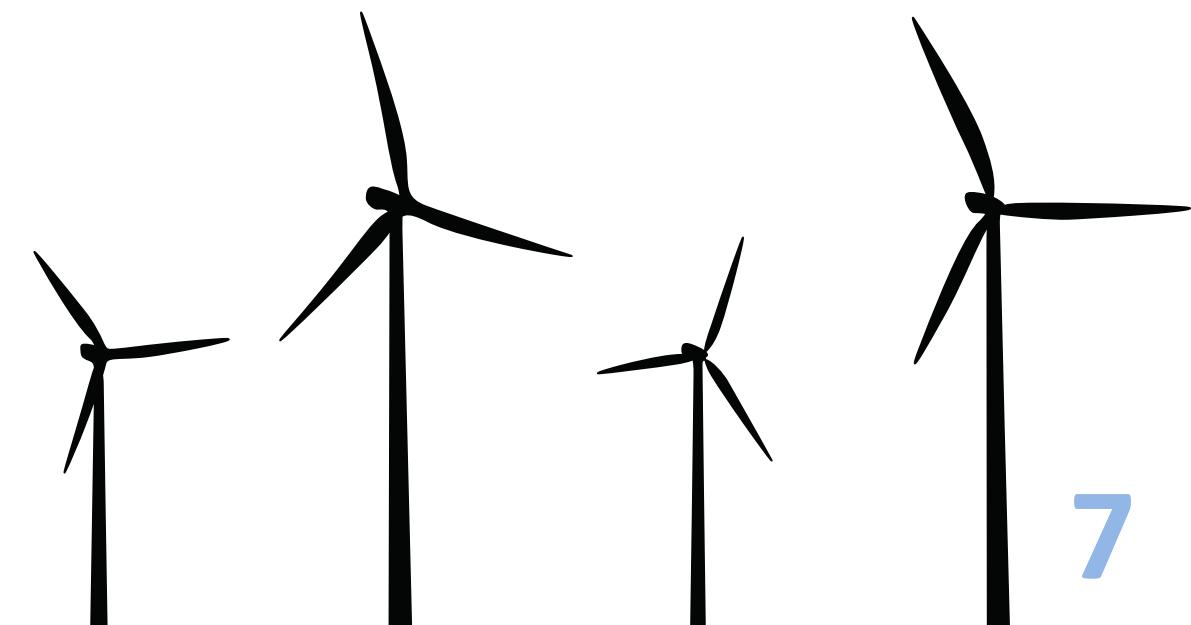
Overall Approach



Example

風力發電機

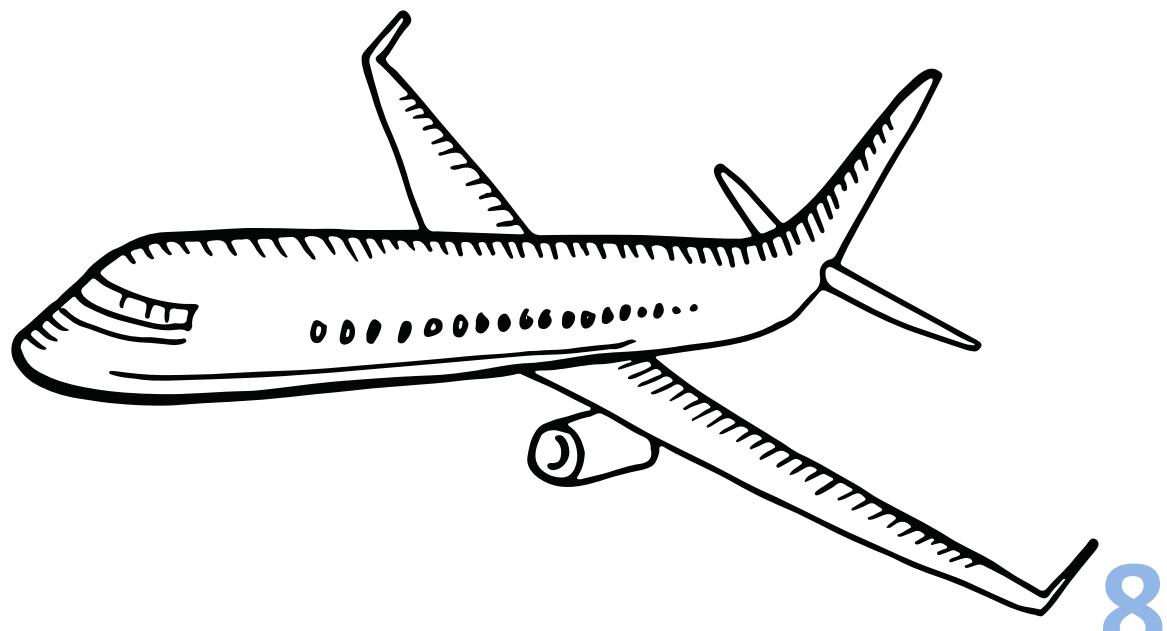
兩萬台風力發電機，每年產生 120PB 的數據



Example - 2

飛機

一架飛機有八萬多個傳感器，設備多、採集頻率高、採集規則複雜

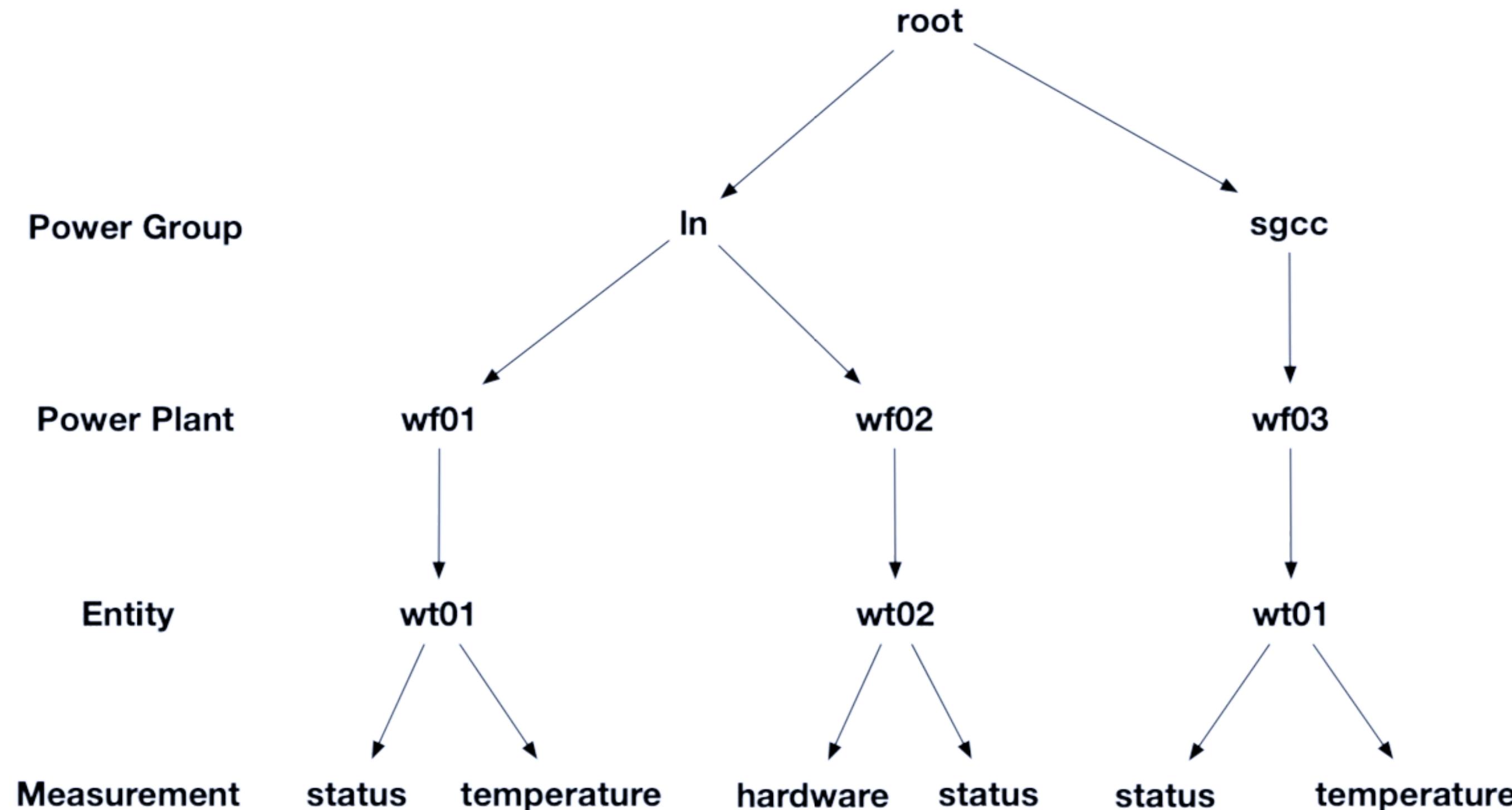


架構解析

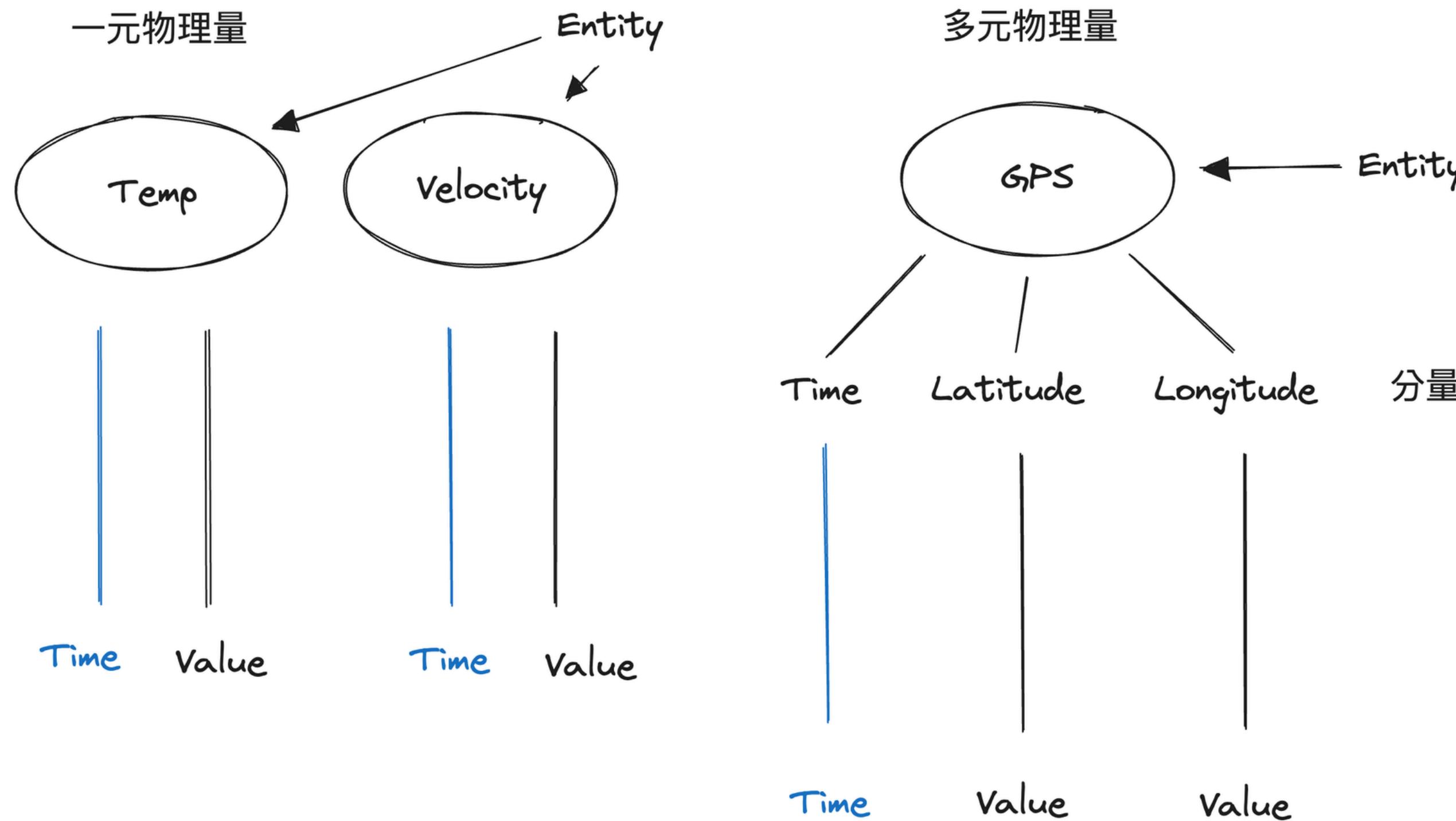
Explanation

- 比起傳統的 Database ， IoTDB 使用一種樹狀結構 (TsFile) ，儲存數據。
- 其中樹狀結構的起始點稱為 "root" ，以下還分多層，分別是 Group, Entity & Measurement (物理量)。

以風力發電機為例，root 下面分為不同的 Power Group，不同 group 中有不同發電機，發電機各自為一實體，各自有其物理量。



物理量的表示法可以是多元或是一元。一元的物理量 (Measurement) 通常稱為時間序列 (Timeseries) ，多元物理量通常稱為 (Aligned Timeseries) 。再插入多元物理量時，某列的某行是允許空值的。



IOTDB server : Engine + Storage

Storage:

- 底層儲存結構(TSFile)

Engine:

- QueryEngine
- StorageEngine

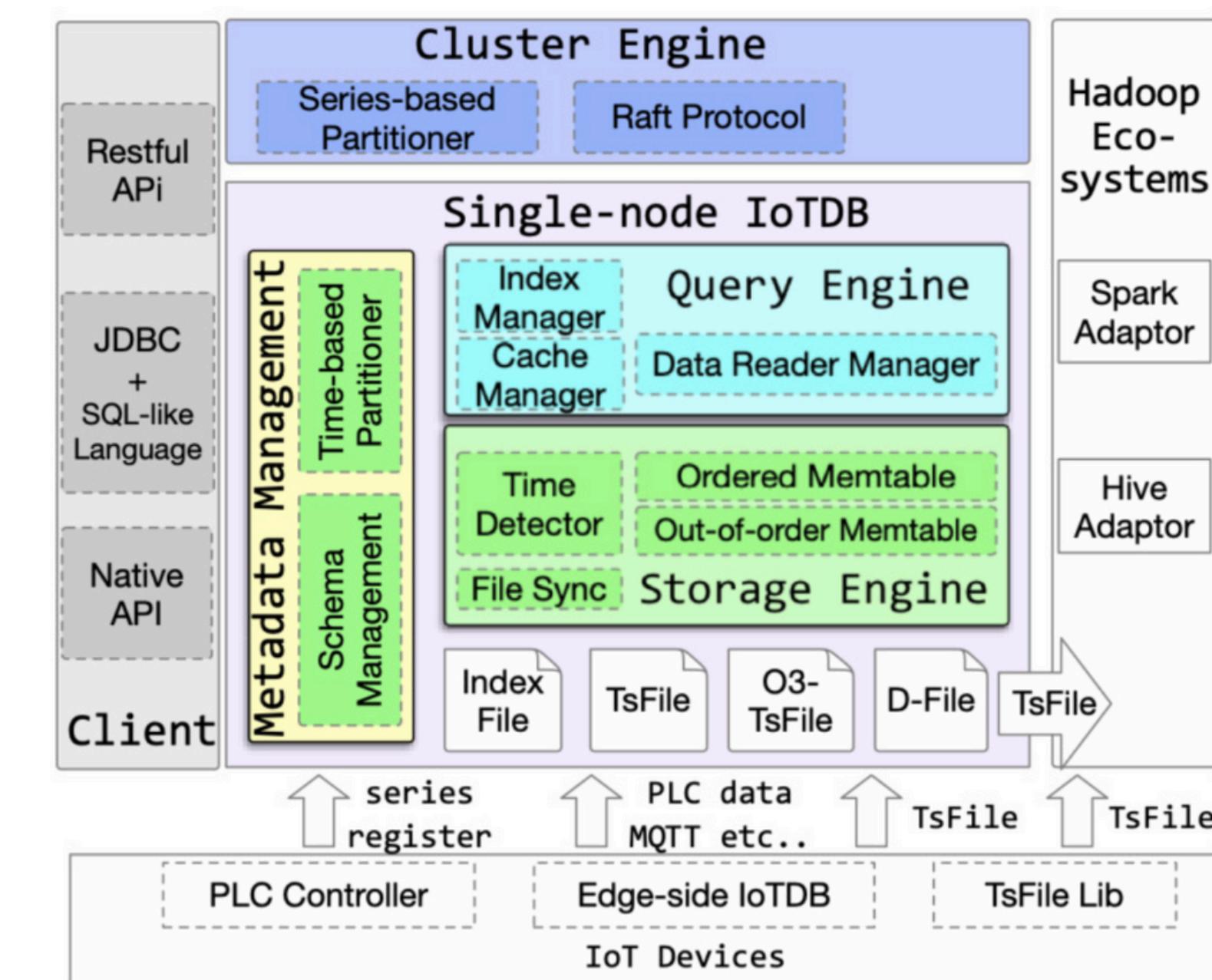


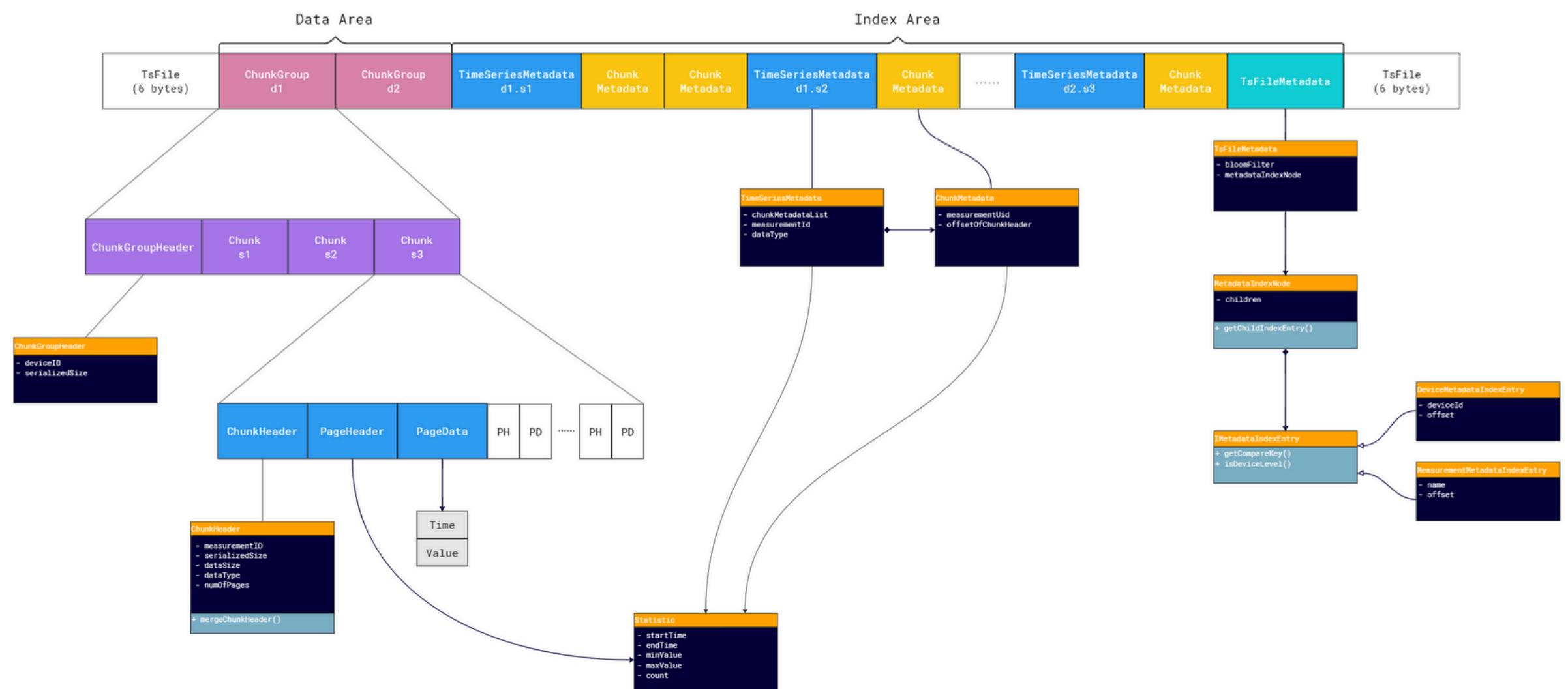
Figure 1: Main modules of IoTDB

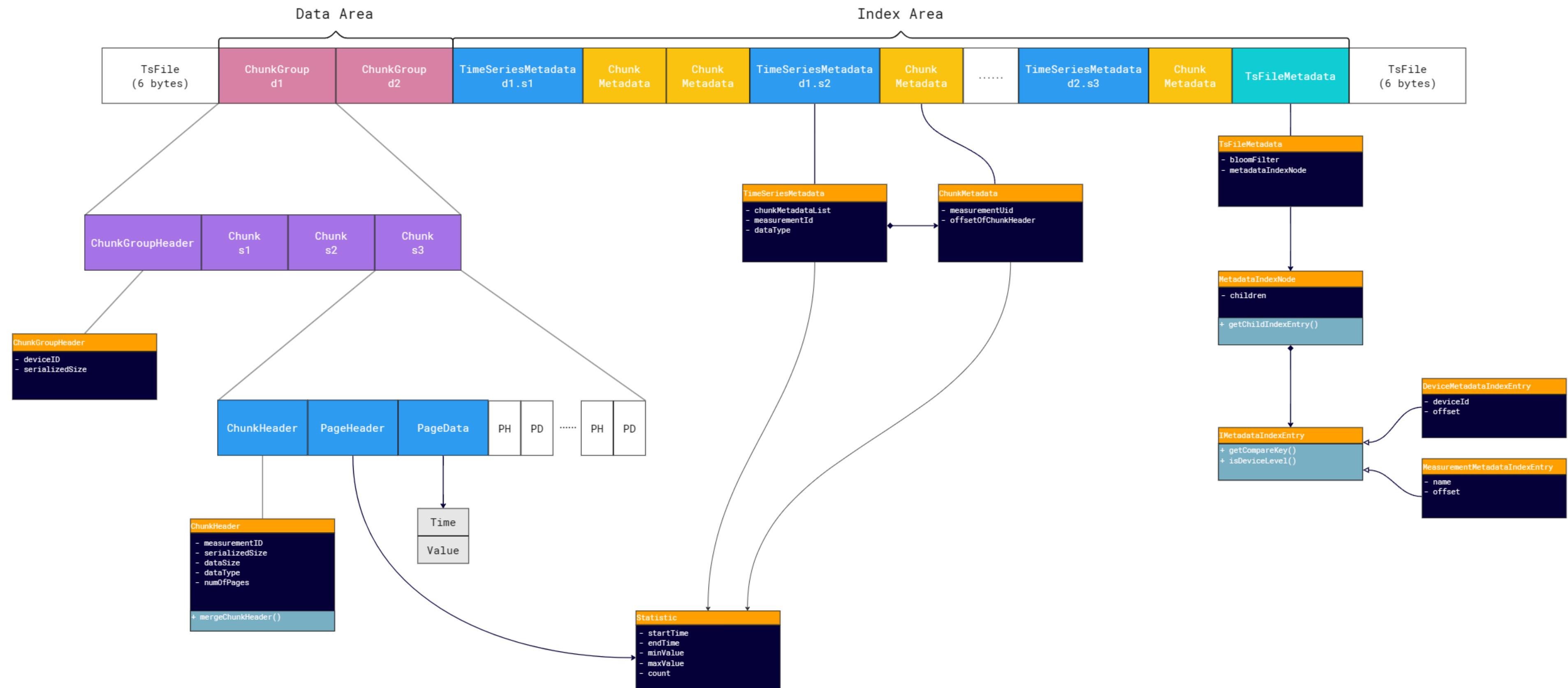
miro

TsFile

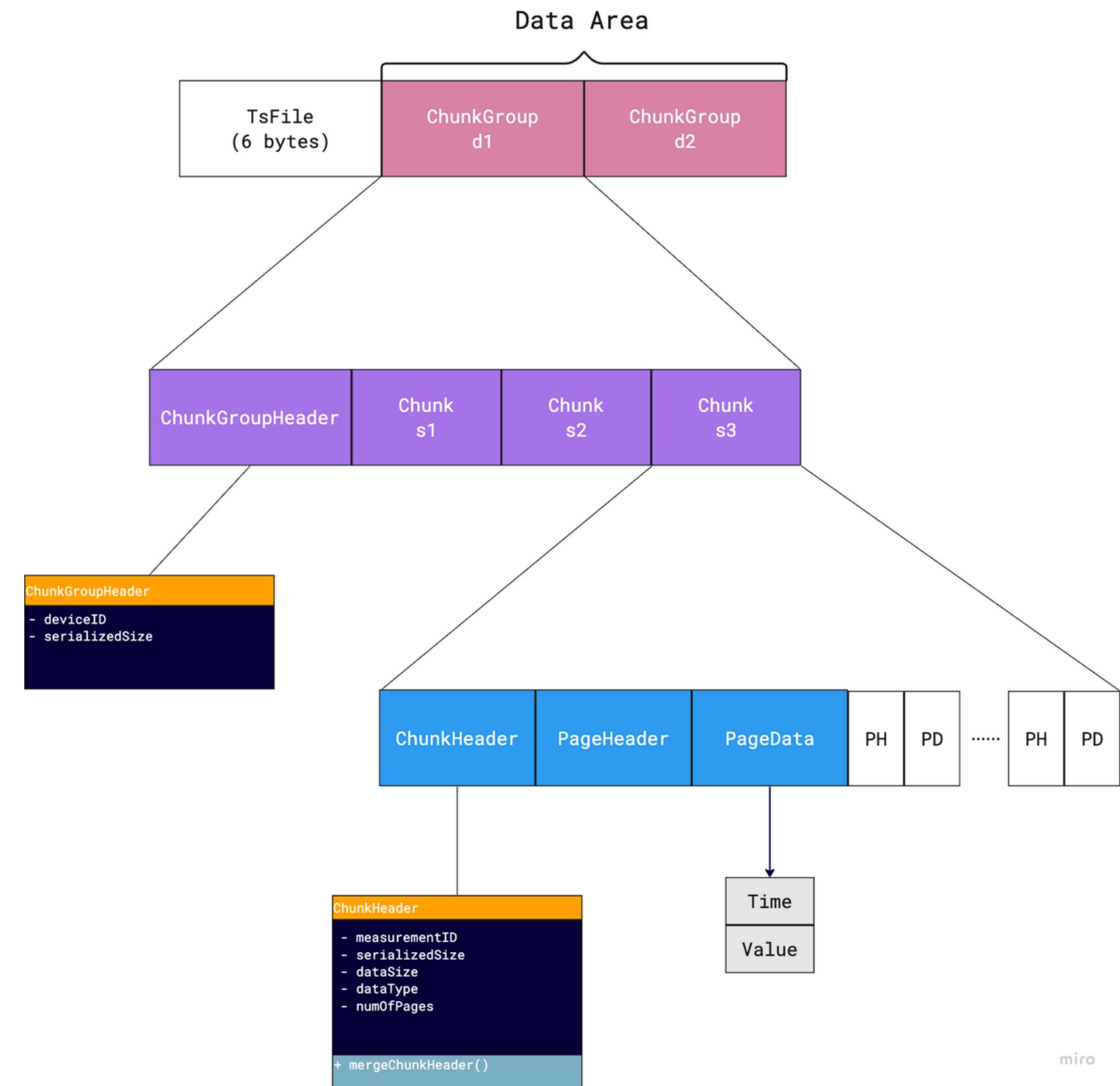
IoTDB的主要文件儲存格式：

- ChunkGroup : 設備
- Chunk : 測點
- Page : 資料



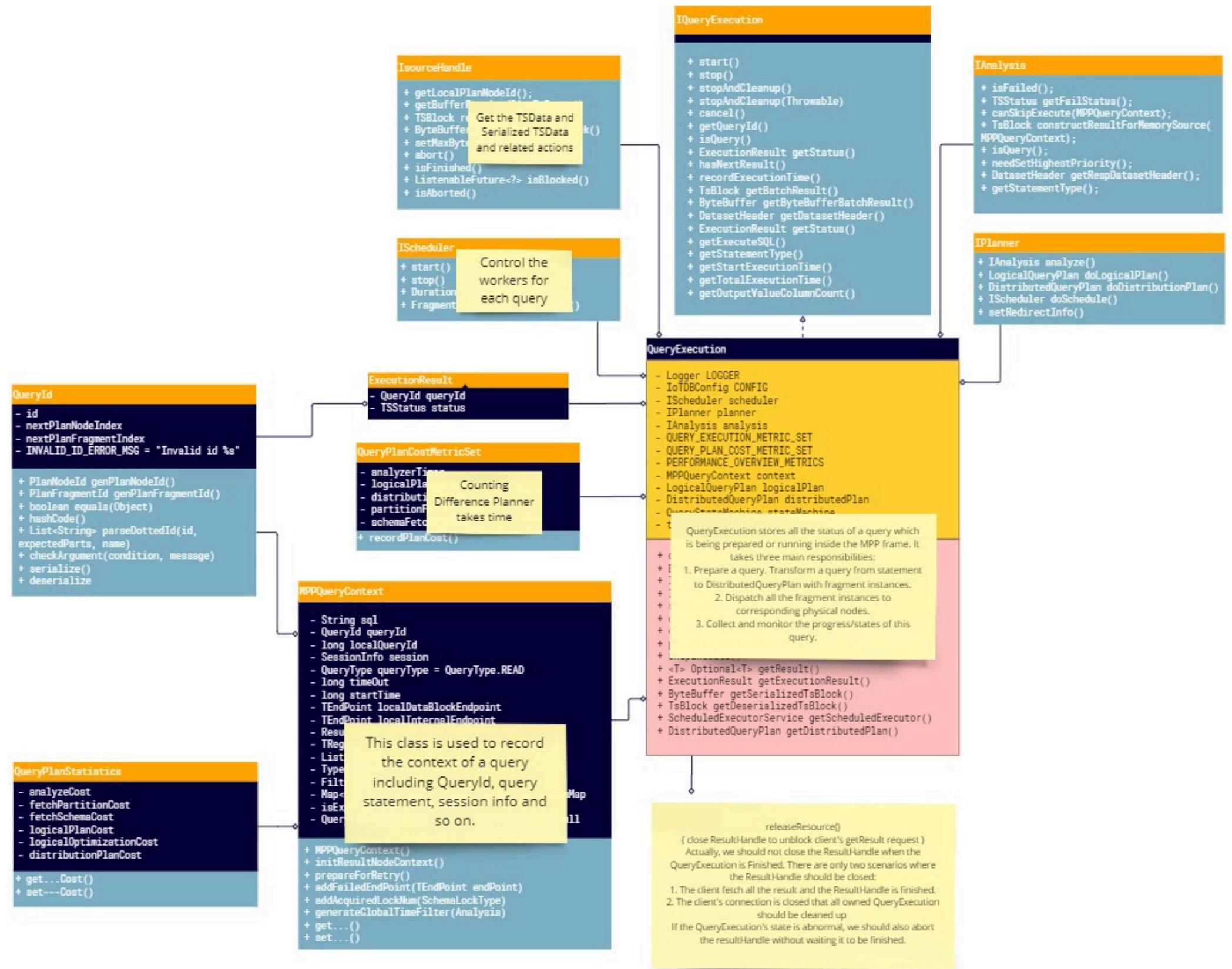


TsFile

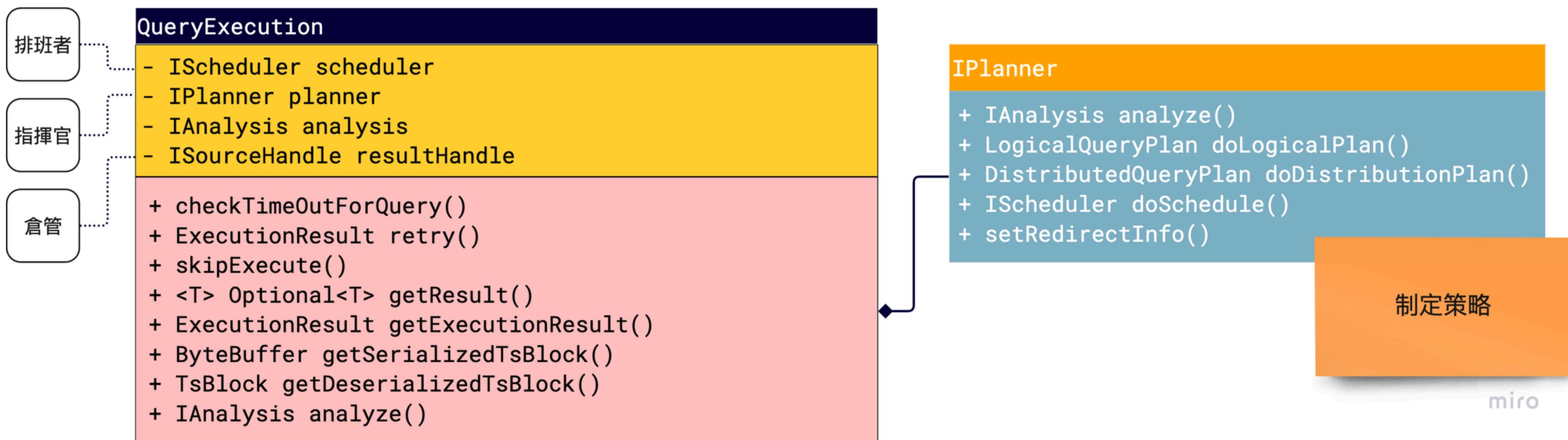


miro

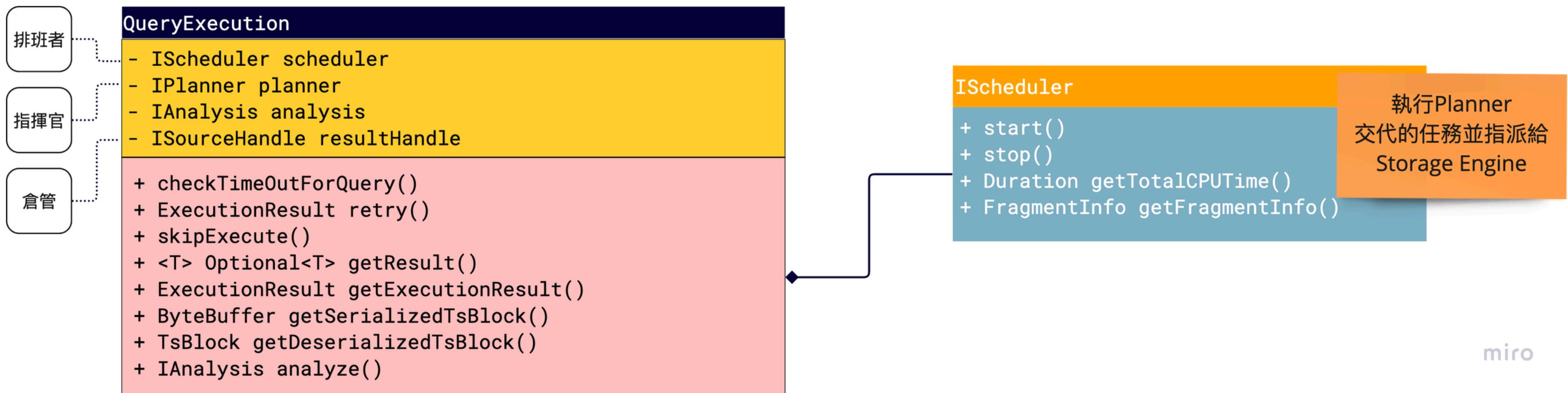
QueryEngine



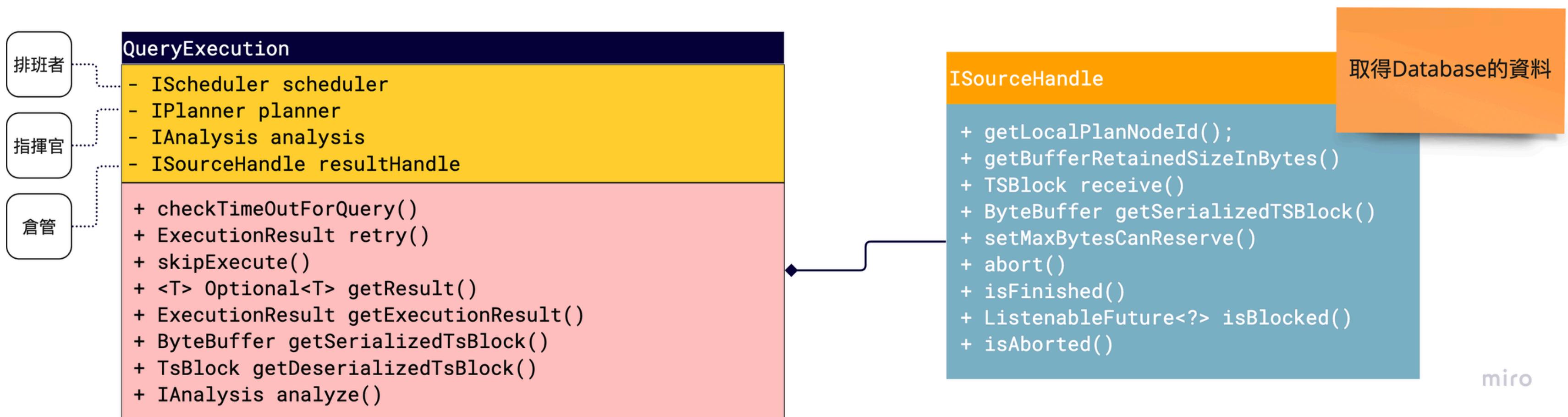
QueryEngine



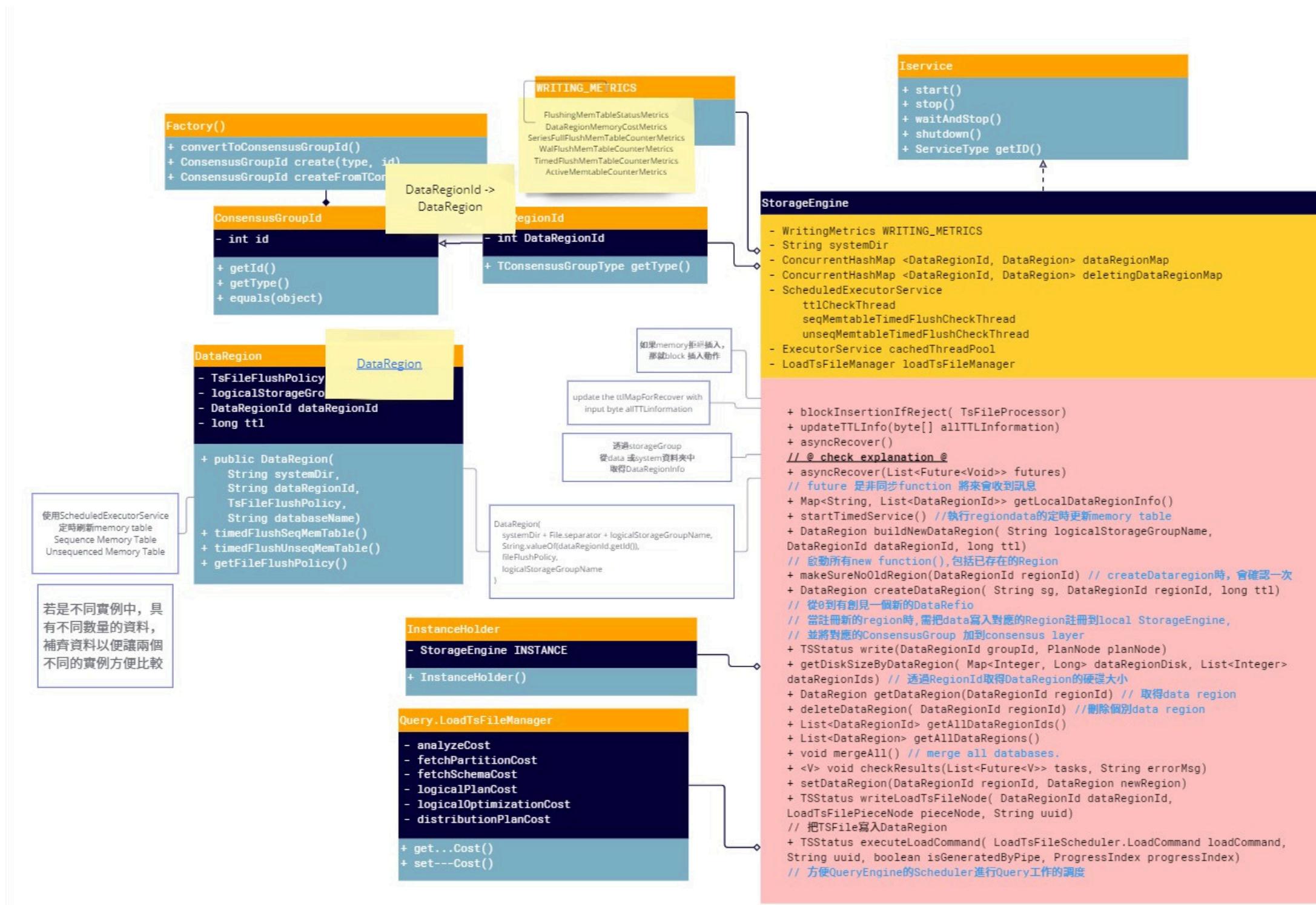
QueryEngine



QueryEngine



Storage Engine



Storage Engine

DataRegion: 一個 DataRegion 可包含多個 TsFiles ，並且管理對於這些 data 的動作，包括插入、Query 或是在不同 Node 之間的複製。

StorageEngine

- String systemDir
- ConcurrentHashMap <DataRegionId, DataRegion> dataRegionMap
- ConcurrentHashMap <DataRegionId, DataRegion> deletingDataRegionMap
- LoadTsFileManager loadTsFileManager

miro

Storage Engine

StorageEngine

- String systemDir
 - ConcurrentHashMap <DataRegionId, DataRegion> dataRegionMap
 - ConcurrentHashMap <DataRegionId, DataRegion> deletingDataRegionMap
 - LoadTsFileManager loadTsFileManager
-
- + Map<String, List<DataRegionId>> getLocalDataRegionInfo()
 - + DataRegion createDataRegion(String sg, DataRegionId regionId, long ttl)
 - + DataRegion getDataRegion(DataRegionId regionId) // 取得data region
 - + List<DataRegionId> getAllDataRegionIds()
 - + List<DataRegion> getAllDataRegions()

可以對 DataRegion
進行操作

miro

Storage Engine

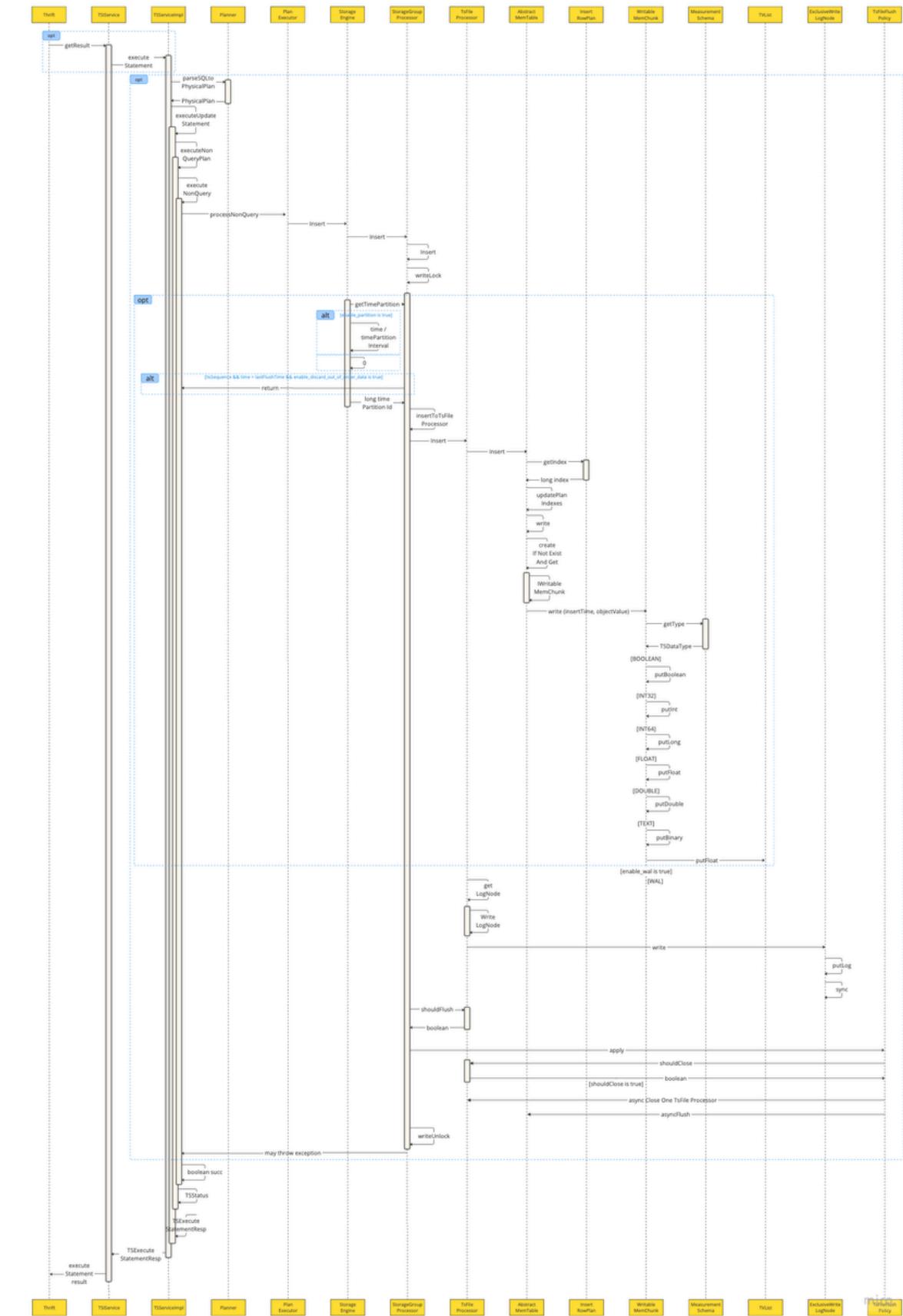
StorageEngine

```
- String systemDir  
- ConcurrentHashMap <DataRegionId, DataRegion> dataRegionMap  
- ConcurrentHashMap <DataRegionId, DataRegion> deletingDataRegionMap  
- LoadTsFileManager loadTsFileManager  
  
+ void mergeAll() // merge all databases.  
+ void setDataRegion(DataRegionId regionId, DataRegion newRegion)  
+ void deleteDataRegion( DataRegionId regionId)  
+ TSStatus writeLoadTsFileNode( DataRegionId dataRegionId, LoadTsFilePieceNode  
pieceNode, String uuid)  
+ TSStatus write(DataRegionId groupId, PlanNode planNode)  
// 方便 QueryEngine 的 Scheduler 進行 Query 工作的調度  
+ TSStatus executeLoadCommand( LoadTsFileScheduler.LoadCommand loadCommand, String  
uuid, boolean isGeneratedByPipe, ProgressIndex progressIndex)
```

或是直接寫入新的
TSFile 並回傳狀態

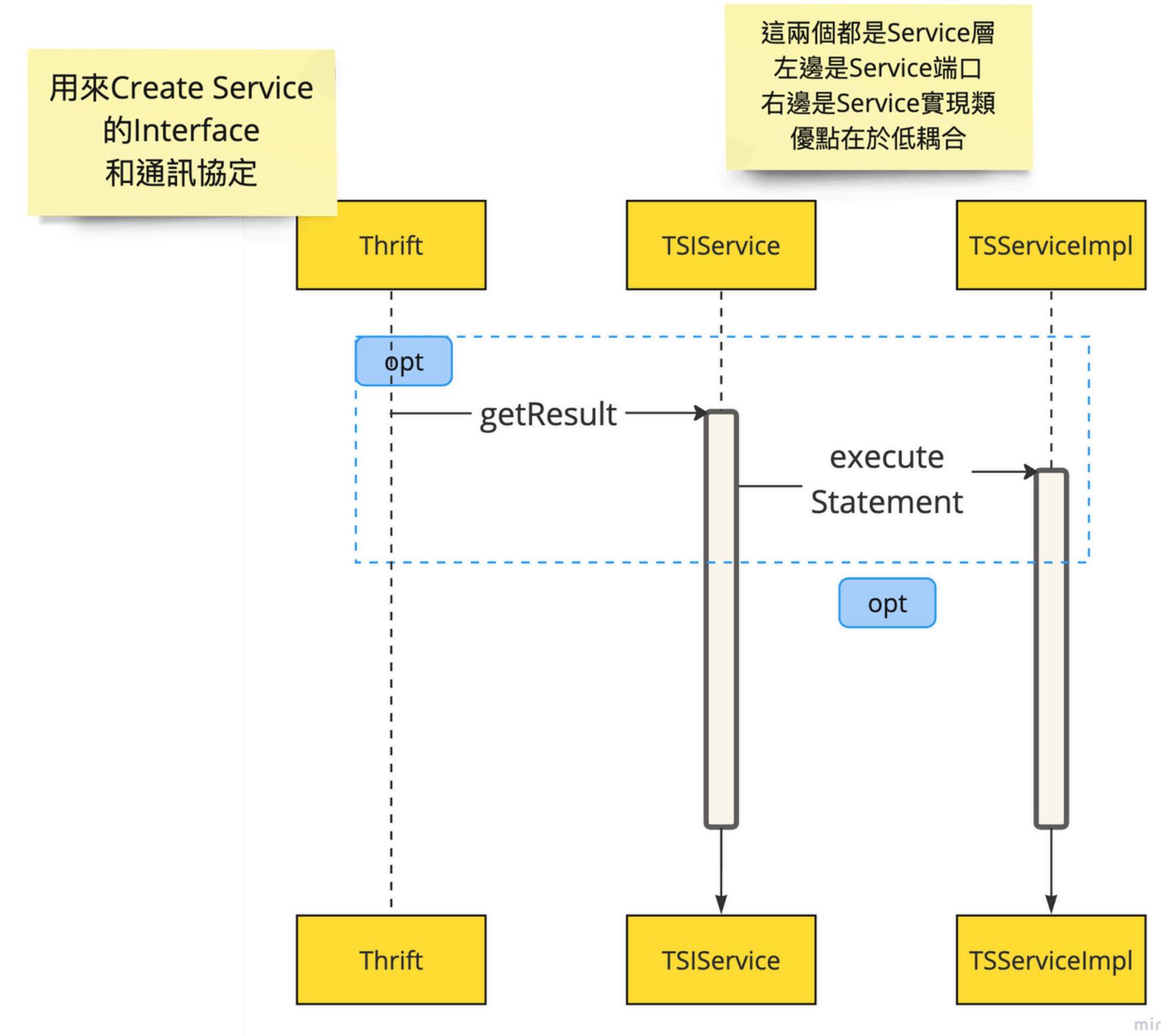
miro

Sequence Diagram



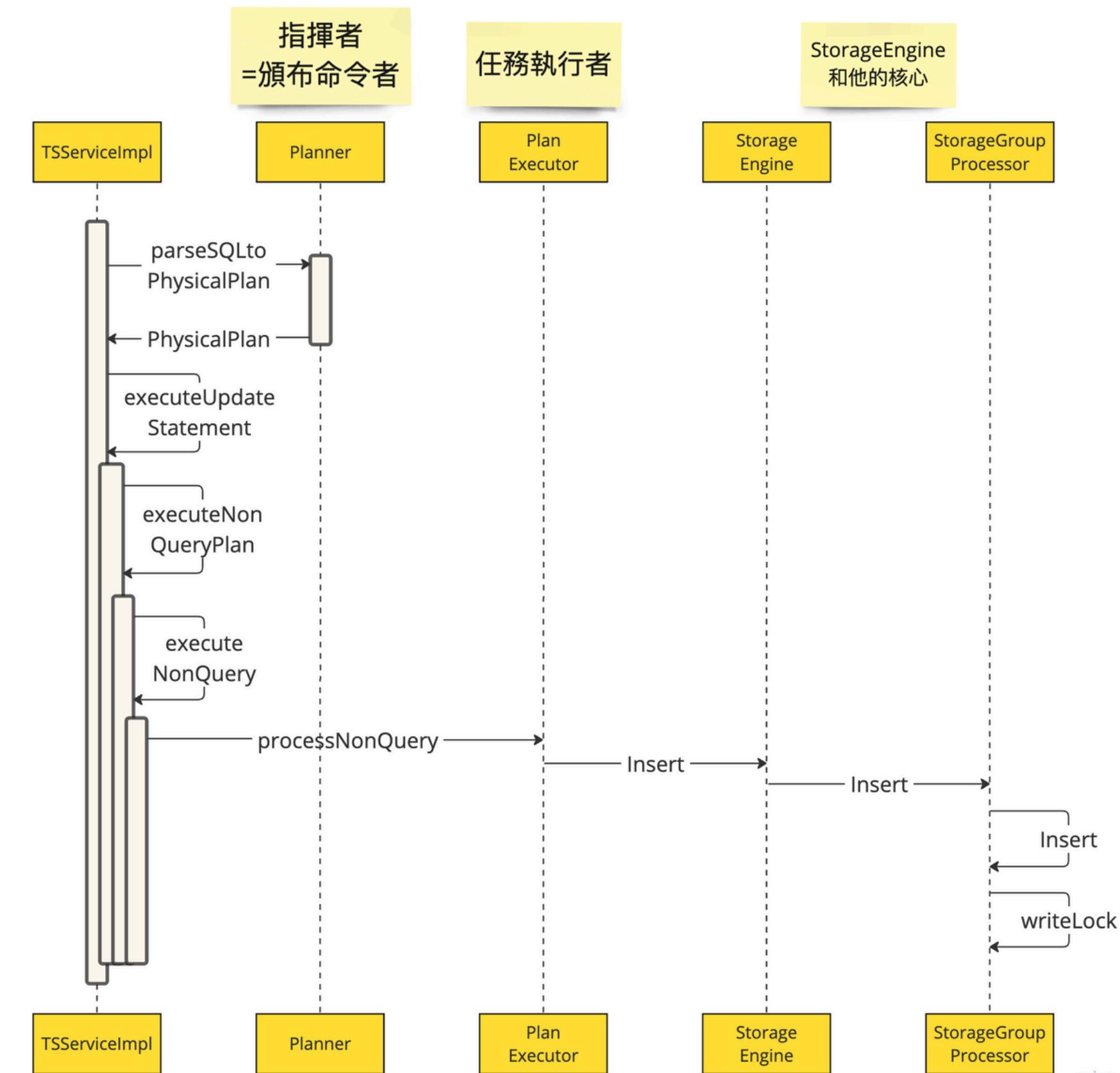
IOTDB Service:

Thrift負責Service的接口和通訊協議
在IOTDB client API使用時
Java、C++、NodeJS、Go、Rust 等
都需要安裝並使用Thrift



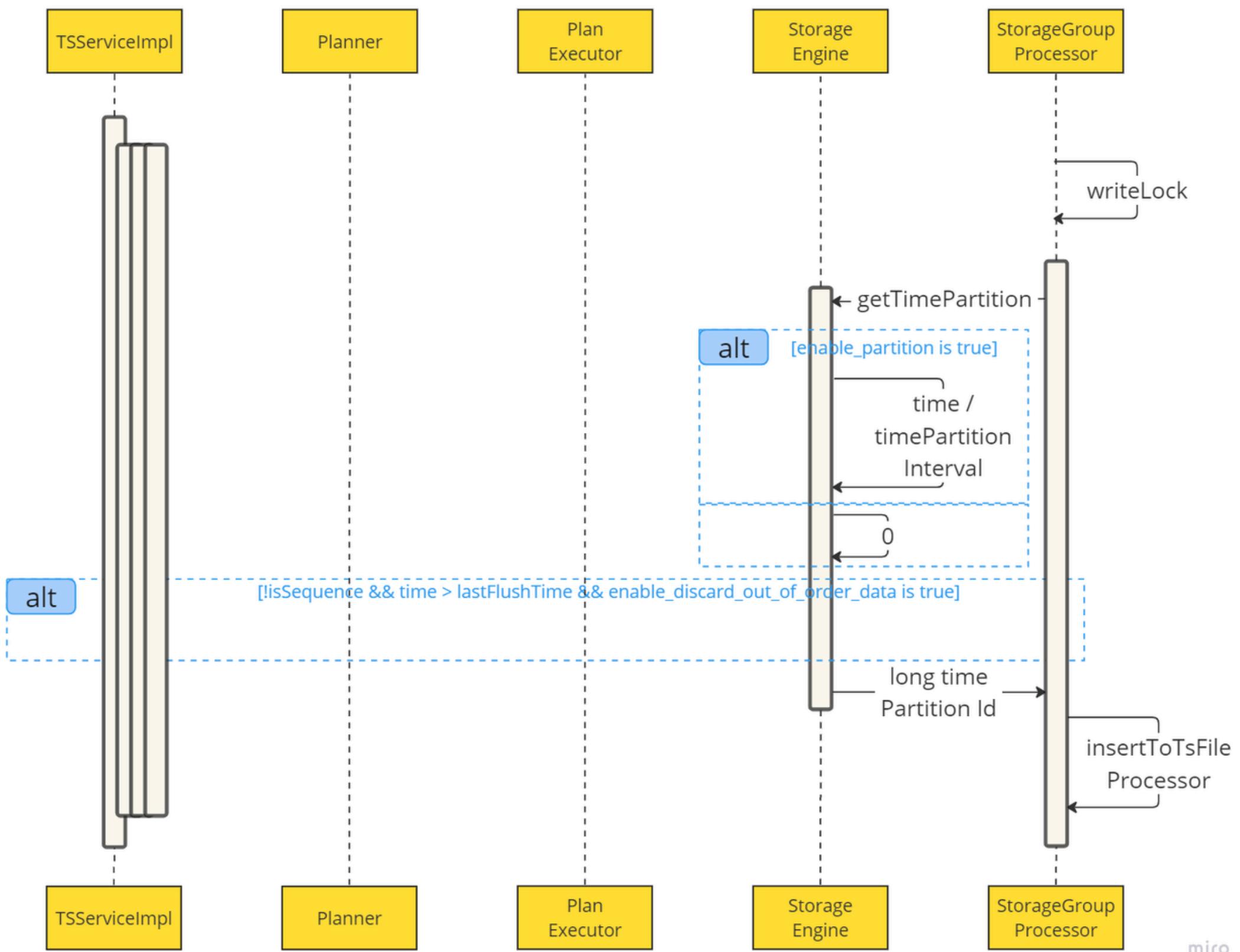
執行IOTDB Server

QueryEngine 中的 Planner
會將要處理的任務
分配給 Storage Engine 執行



對 Plan 做時間標記

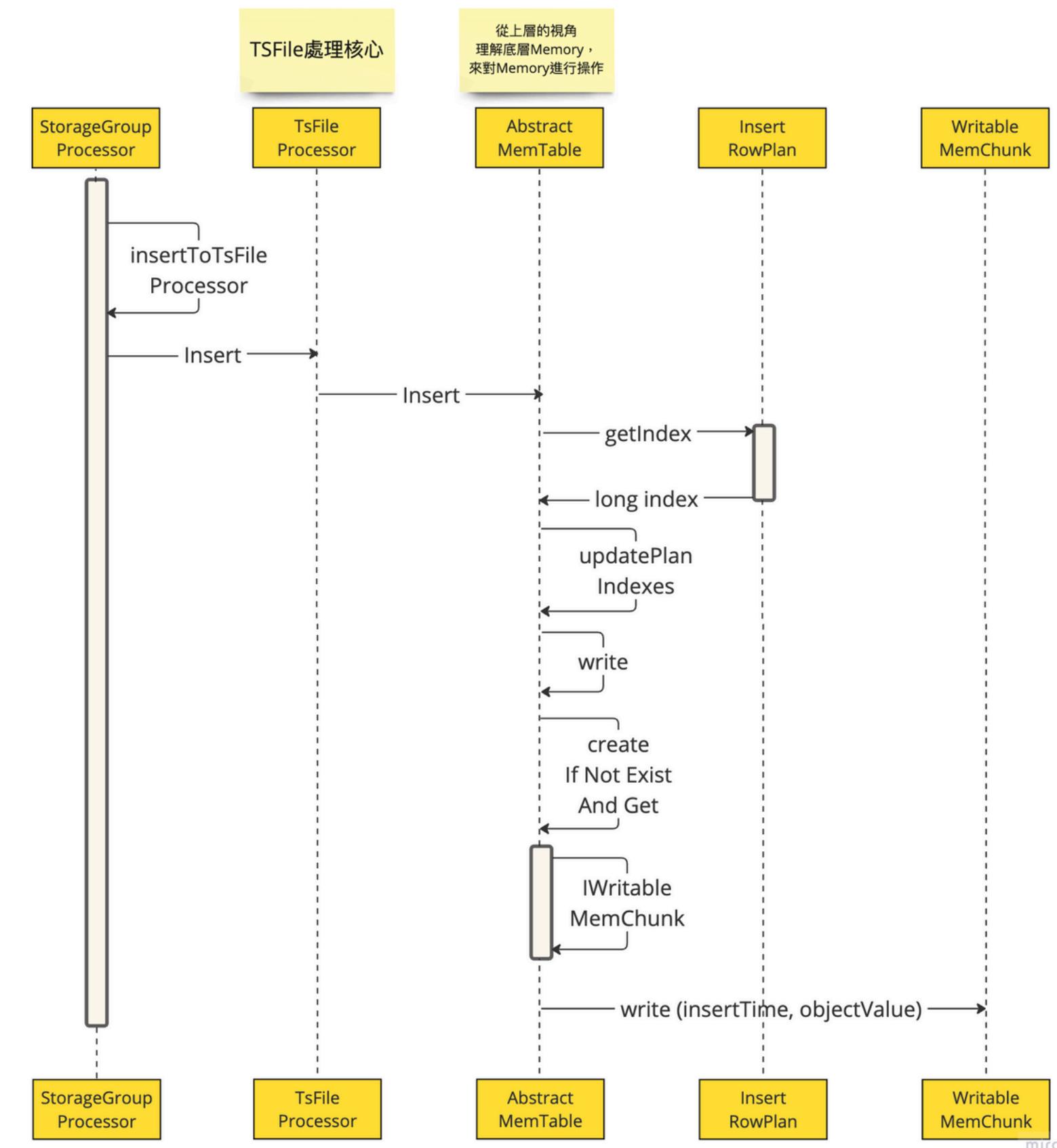
將 TSFile Processor 的 WriteLock 解鎖
對傳入 Storage Engine 的 Plan 加上時
間後交給 TSFile Processor



miro

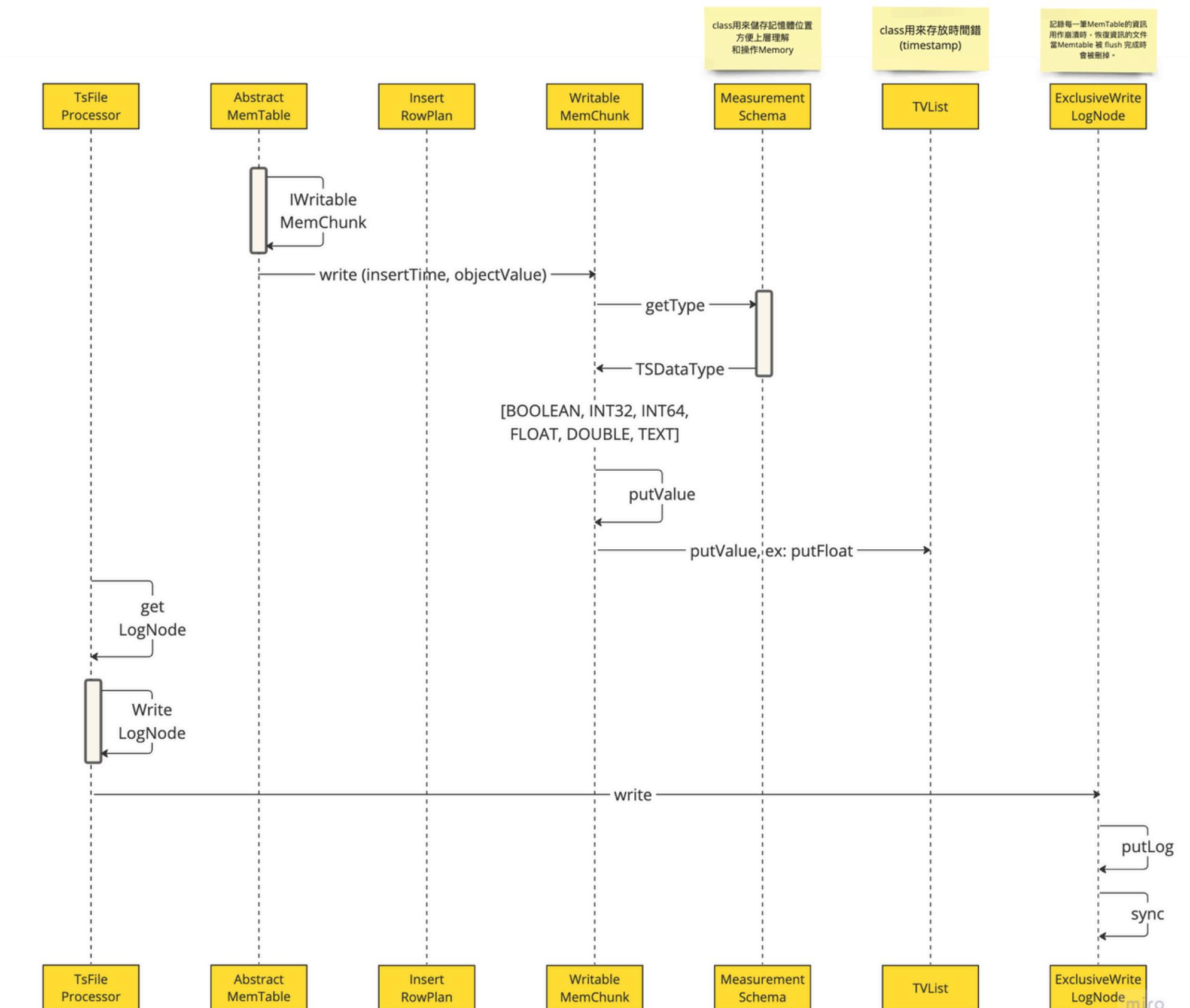
準備寫入Memory

TSFile Processor 將
帶有時間標記的 Plan
做寫入記憶體的動作

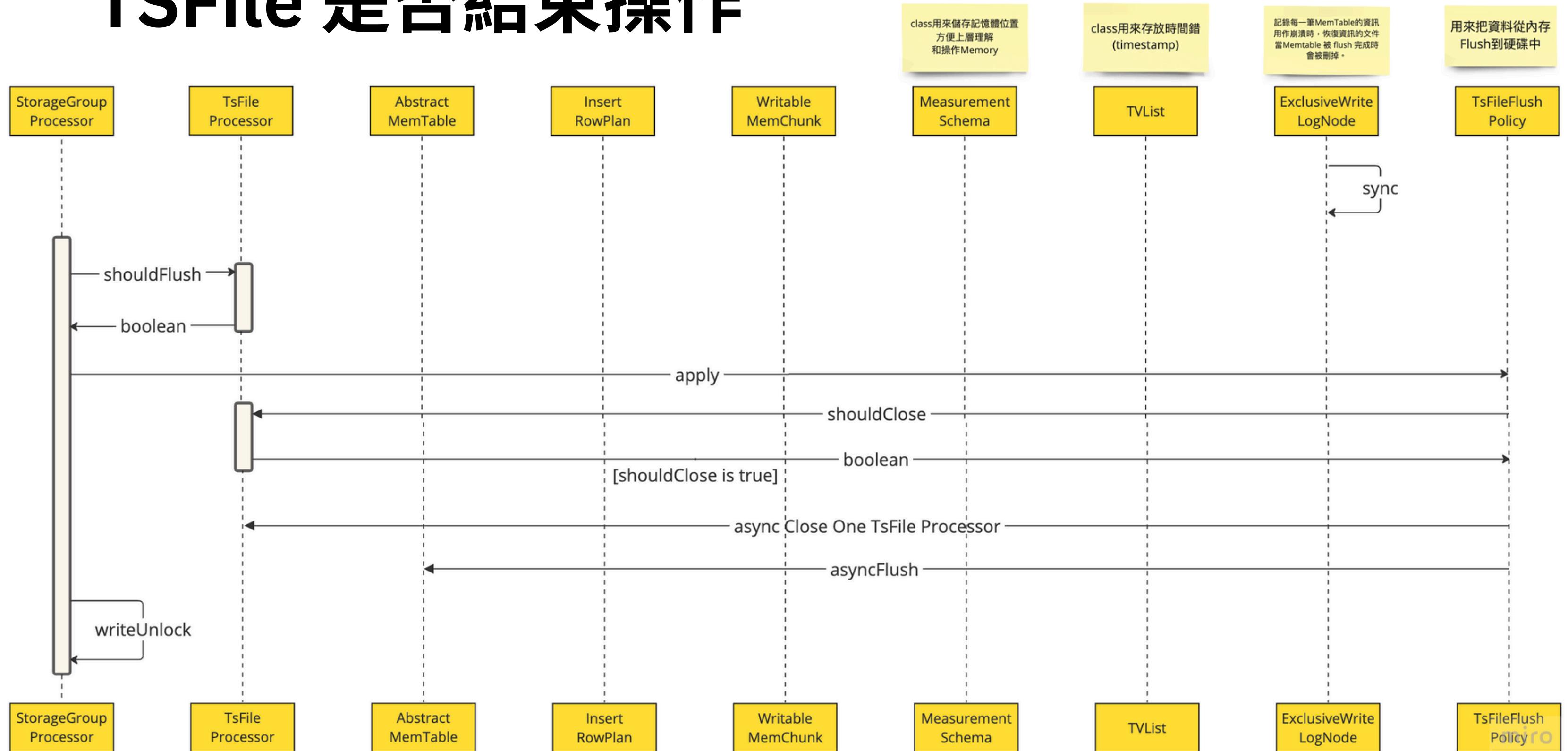


寫入Memory的過程

在寫入 Memory 之前，
LogNode 會先紀錄每筆 MemTable
用來作為崩潰時的資料備份
而當資料從 RAM 更新至 Memory 時
LogNode 的備份會被刪除

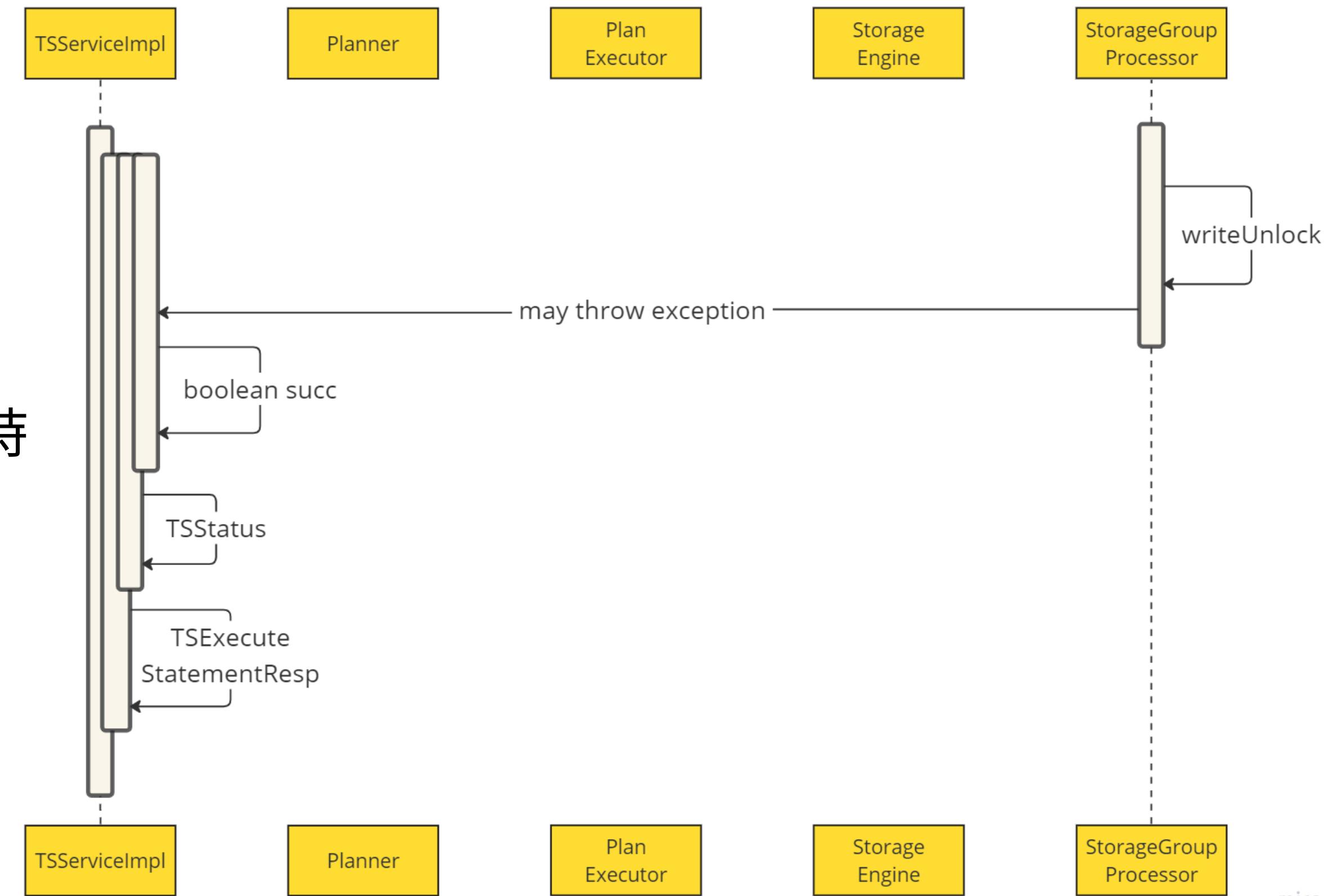


同步資訊，並告知 TSFile 是否結束操作



錯誤回報

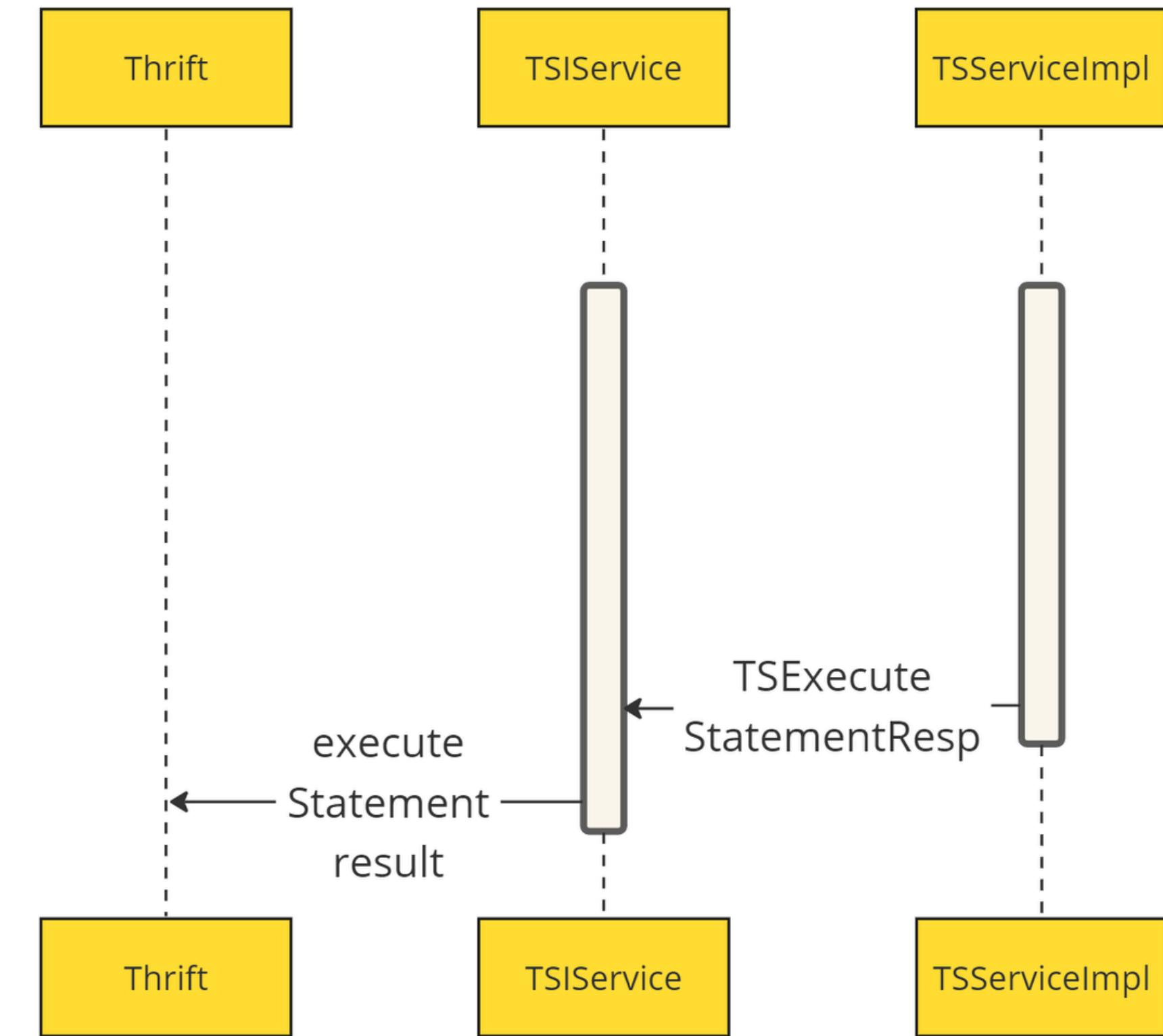
當資料已被存入記憶體時
若有 Error Message
則會回傳給 TSService



miro

完成操作

回傳結果最後會被傳給 Thrift
讓 Client 知道是否寫入成功
或是了解錯誤發生原因

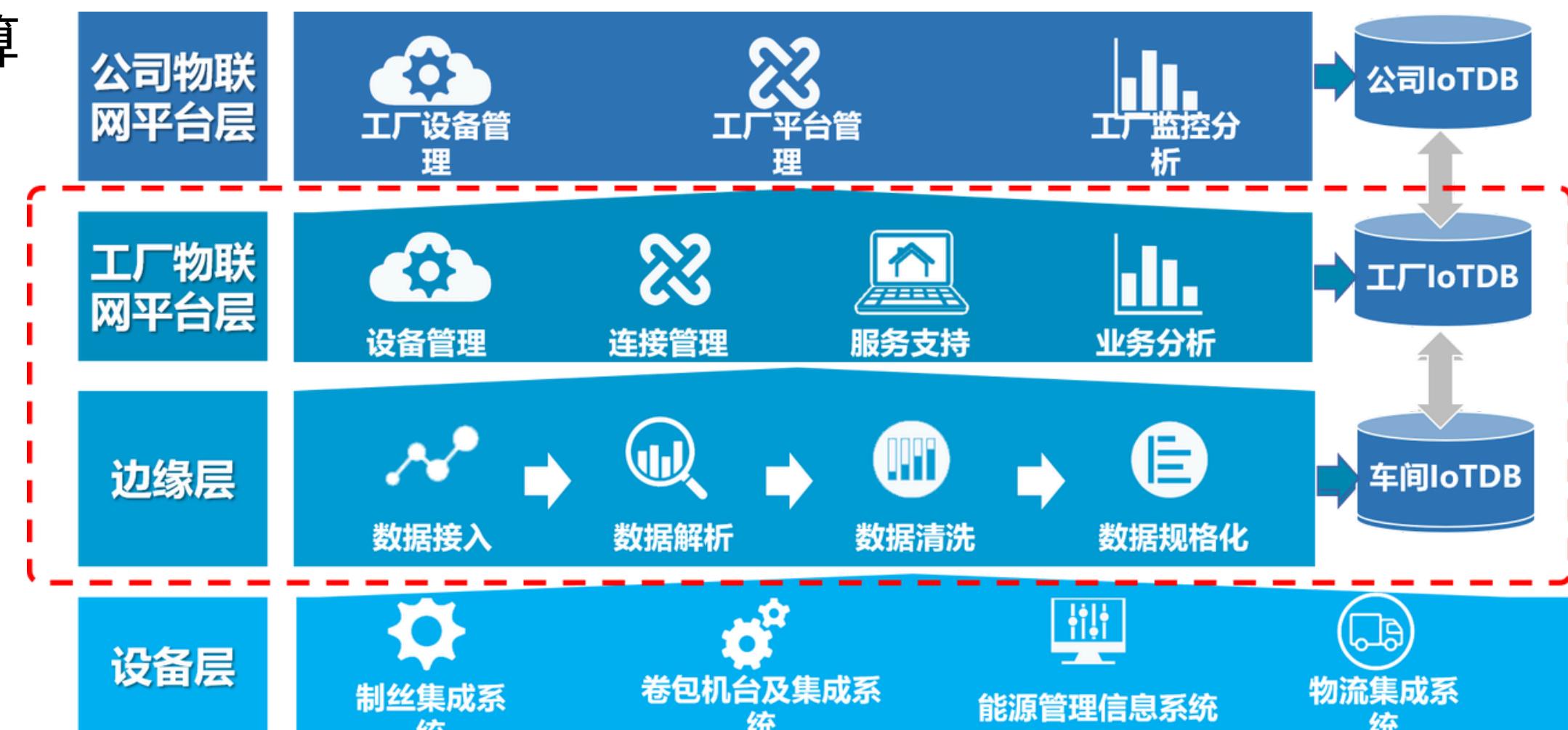


miro

如何解決問題

Case: Smart Factory

- Cloud-Edge Collaboration
- 大量時序資料處理、邊緣計算
- 資料得以流通在公司各層



Evaluation and Comments

和相近技術的比較

InfluxDB

- 原生 TSDB
- 介面：InfluxQL 和 HTTP API



OpenTSDB 與 KairosDB

- 基於 NoSQL 的 TSDB
- 介面：RESTful API
- 為相似的兩種資料庫
- 相異處：OpenTSDB 基於 HBase / KairosDB 基於 Cassandra



TimescaleDB

- 構建在 PostgreSQL 之上的 TSDB
- 介面：SQL



基礎功能比較

TSDB	IoTDB	InfluxDB	OpenTSDB	KairosDB	TimescaleDB
OpenSource	+	+	+	+	+
SQL-like	+	+	-	-	++
Schema	Tree-based, tag-based	tag-based	tag-based	tag-based	Relational
Writing out-of-order data	+	+	+	+	+
Schema-less	+	+	+	+	+
Batch insertion	+	+	+	+	+
Time range filter	+	+	+	+	+
Order by time	++	+	-	-	+
Value filter	+	+	-	-	+
Downsampling	++	+	+	+	+
Fill	++	+	+	-	+
LIMIT	+	+	+	+	+
SLIMIT	+	+	-	-	?
Latest value	++	+	+	-	+

基礎功能比較

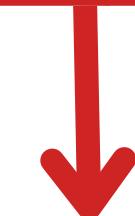
TSDB	IoTDB	InfluxDB	OpenTSDB	KairosDB	TimescaleDB
Schema	Tree-based, tag-based	tag-based	tag-based	tag-based	Relational



- 在許多工業場景中，設備的管理是分層的
- 在許多現實世界的設備中，標籤名稱是不變的。例如，風力渦輪機製造商透過其位於哪個國家、所屬的發電場名稱以及其在發電場中的 ID 來識別其風力渦輪機。（“root.the-country-name.the-farm-name.the-id”）

基礎功能比較

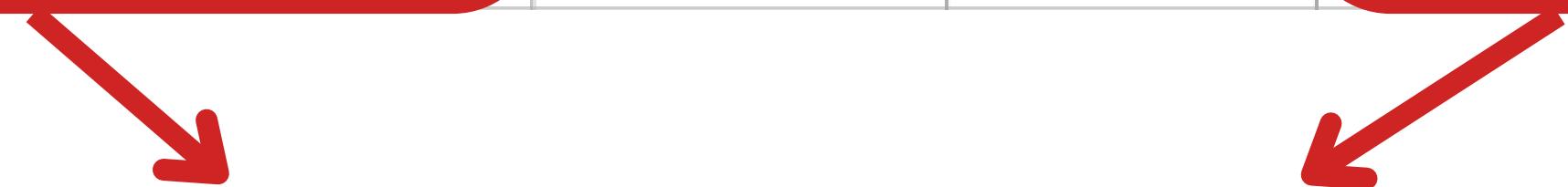
TSDB	IoTDB	InfluxDB	OpenTSDB	KairosDB	TimescaleDB
Order by time	++	+	-	-	+



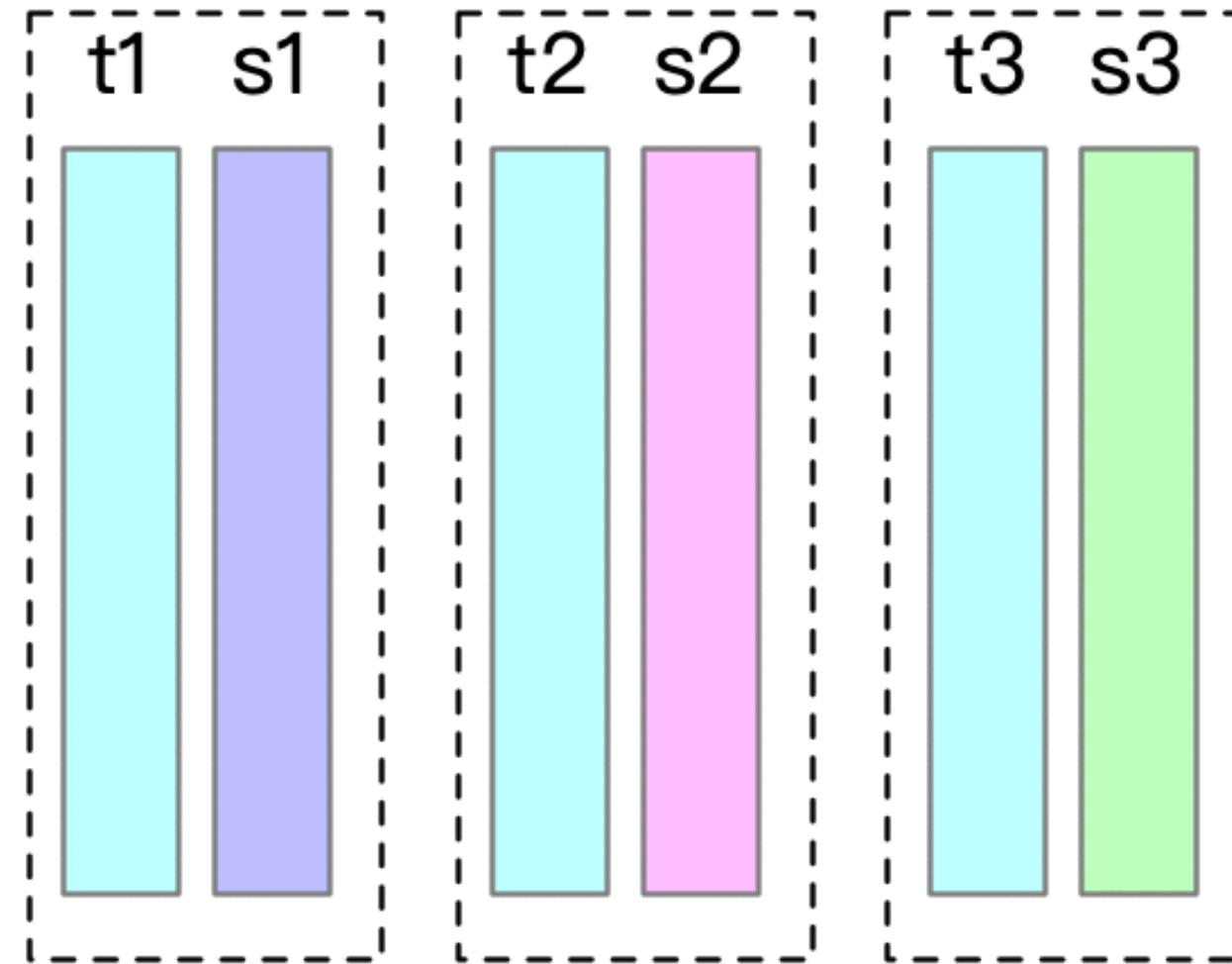
不支持「多條」時間序列的按時間戳排序

基礎功能比較

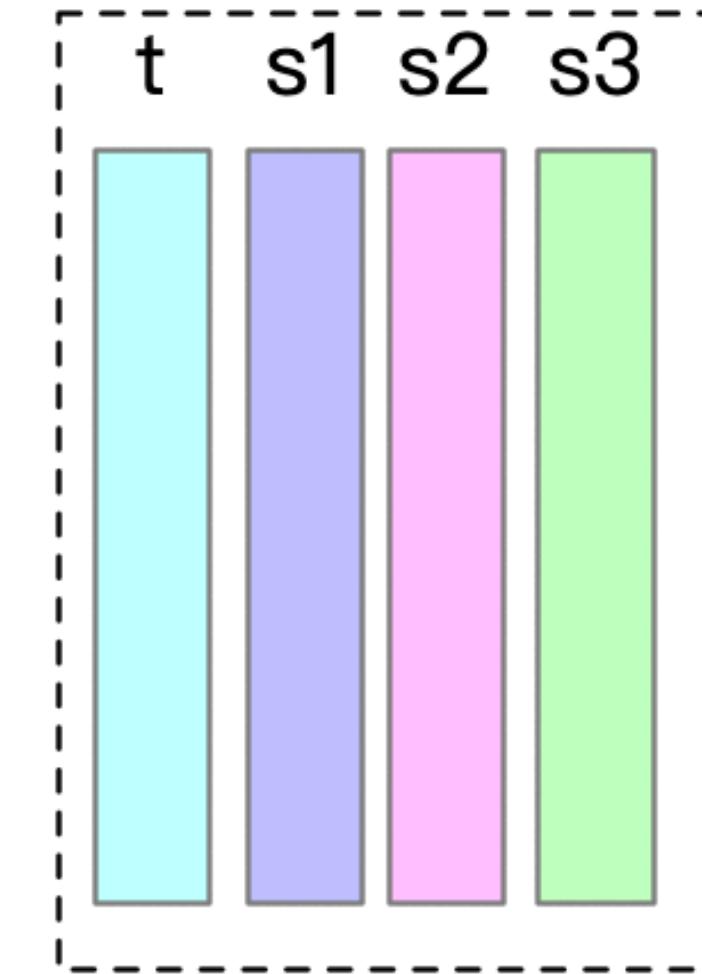
TSDB	IoTDB	InfluxDB	OpenTSDB	KairosDB	TimescaleDB
Order by time	++	+	-	-	+



支持「多條」時間序列的按時間戳排序



timeseries
s1, s2, s3



aligned timeseries
(s1,s2,s3)

IoTDB

基礎功能比較

- IoTDB 通過 align by time 在資料「寫入時」進行對齊來實現

timestamp	風場 1 的風速	風場 1 中風機 1 所產生電能
1	5.0	13.1
2	6.0	13.3
3	NULL	13.1

基礎功能比較

- InfluxDB 是通過在「查詢時」進行對齊來實現

timestamp	時間序列名	值
1	風場 1 的風速	5.0
1	風場 1 中風機 1 所產生電能	13.1
2	風場 1 的風速	6.0
2	風場 1 中風機 1 所產生電能	13.3
3	風場 1 中風機 1 所產生電能	13.1

與傳統 Relational DB 比較

- **PostgreSQL**

- 關聯式資料庫管理系統 (RDBMS)
- 和 IoTDB 皆是強大的數據庫管理系統
- 多用於處理非時序數據

與傳統 Relational DB 比較

特點	IoTDB	PostgreSQL
資料模型與結構	時序資料 (Tsfile)	關聯式資料、通用表格結構
查詢語言	類似 SQL 的查詢語言	標準 SQL
性能	通常具有較高的效能和較低的延遲，專為時序資料進行了最佳化	在複雜查詢和大規模資料處理時，效能可能受到限制，尤其在處理大量時序資料時
應用場景	智慧城市、工業物聯網等	金融、電商等

總結

Apache IoTDB 是一個專為處理 Time series data 而設計的開源數據庫系統，特別適用於物聯網設備和傳感器所產生的大量 Time series data

總結

- **主要功能**

- 提供高效的時序數據儲存和查詢功能
- 支援多種壓縮算法，提高數據儲存效率
- 可擴展性強，適用於處理大量的時序數據

總結

- **核心架構：**

- 基於 TsFile、資料庫引擎和分析引擎構建的分散式資料庫系統
- 以樹狀結構儲存數據
- 提供模板功能

- **應用場景：**

- 風力發電機監控系統
- 車聯網系統
- 智能工廠中生產設備的管理

總結

- 安裝與使用：
 - Apache IoTDB 可通過 Docker 或手動安裝的方式部署和運行
- 評估與建議：
 - 與其他時序數據庫相比，Apache IoTDB 在許多方面具有優勢，如資料壓縮、多實例同步、低延遲查詢等
 - IoTDB 的樹狀結構和支持多種壓縮算法等特性，使其在工業互聯網領域具有更好的適應性和性能

實驗紀錄與心得

Demo

包含

1. IoTDB 操作
2. Python SDK 的串接
3. RESTful Service (V2).

<https://www.youtube.com/watch?v=d1wpv5wL-Pw>

Docker Install

1. Docker Pull

```
chaotting@A6000-GPU:~$ docker pull apache/iotdb:1.3.0-standalone
1.3.0-standalone: Pulling from apache/iotdb
521f275cc58b: Pull complete
fe6a6404648d: Pull complete
8a0aabae808c: Pull complete
1594c0e637d5: Pull complete
7c5fea037d78: Pull complete
5b355240dfbf: Pull complete
950303a2b413: Pull complete
e77fa0ca1bae: Pull complete
4f4fb700ef54: Pull complete
9d0e1a3e5033: Pull complete
42fa77142943: Pull complete
15c7bc70c5e3: Pull complete
Digest: sha256:14749f800fca09c0290f61593fffeec727d54589055cbcfc2005365a500aeb8051
Status: Downloaded newer image for apache/iotdb:1.3.0-standalone
docker.io/apache/iotdb:1.3.0-standalone
```

Docker Install

2. Create docker bridge network & container

```
chaotting@A6000-GPU:~$ docker network create --driver=bridge --subnet=172.18.0.0/16 --gateway=172.18.0.1 iotdb
15e27cd971d779afba9576b0f90a1ede2f970f53258a72bc51d8601704787782
chaotting@A6000-GPU:~$ docker run -d --name iotdb-service \
    --hostname iotdb-service \
    --network iotdb \
    --ip 172.18.0.6 \
    -p 6667:6667 \
    -e cn_internal_address=iotdb-service \
    -e cn_seed_config_node=iotdb-service:10710 \
    -e cn_internal_port=10710 \
    -e cn_consensus_port=10720 \
    -e dn_rpc_address=iotdb-service \
    -e dn_internal_address=iotdb-service \
    -e dn_seed_config_node=iotdb-service:10710 \
    -e dn_mpp_data_exchange_port=10740 \
    -e dn_schema_region_consensus_port=10750 \
    -e dn_data_region_consensus_port=10760 \
    -e dn_rpc_port=6667 \
    apache/iotdb:1.3.0-standalone
4616534aa1e6783c601a2b3e771db6a904fa481588e4bd88034d5dd63ecd63c0
```

Docker Install

3. Execute the SQL

```
chaotting@A6000-GPU:~$ docker exec -ti iotdb-service /iotdb/sbin/start-cli.sh -h iotdb-service
-----
Starting IoTDB Cli
-----
IoTDB [version 1.3.0 (Build: d1326c5)]
Successfully login at iotdb-service:6667
IoTDB> █
```

Local Install

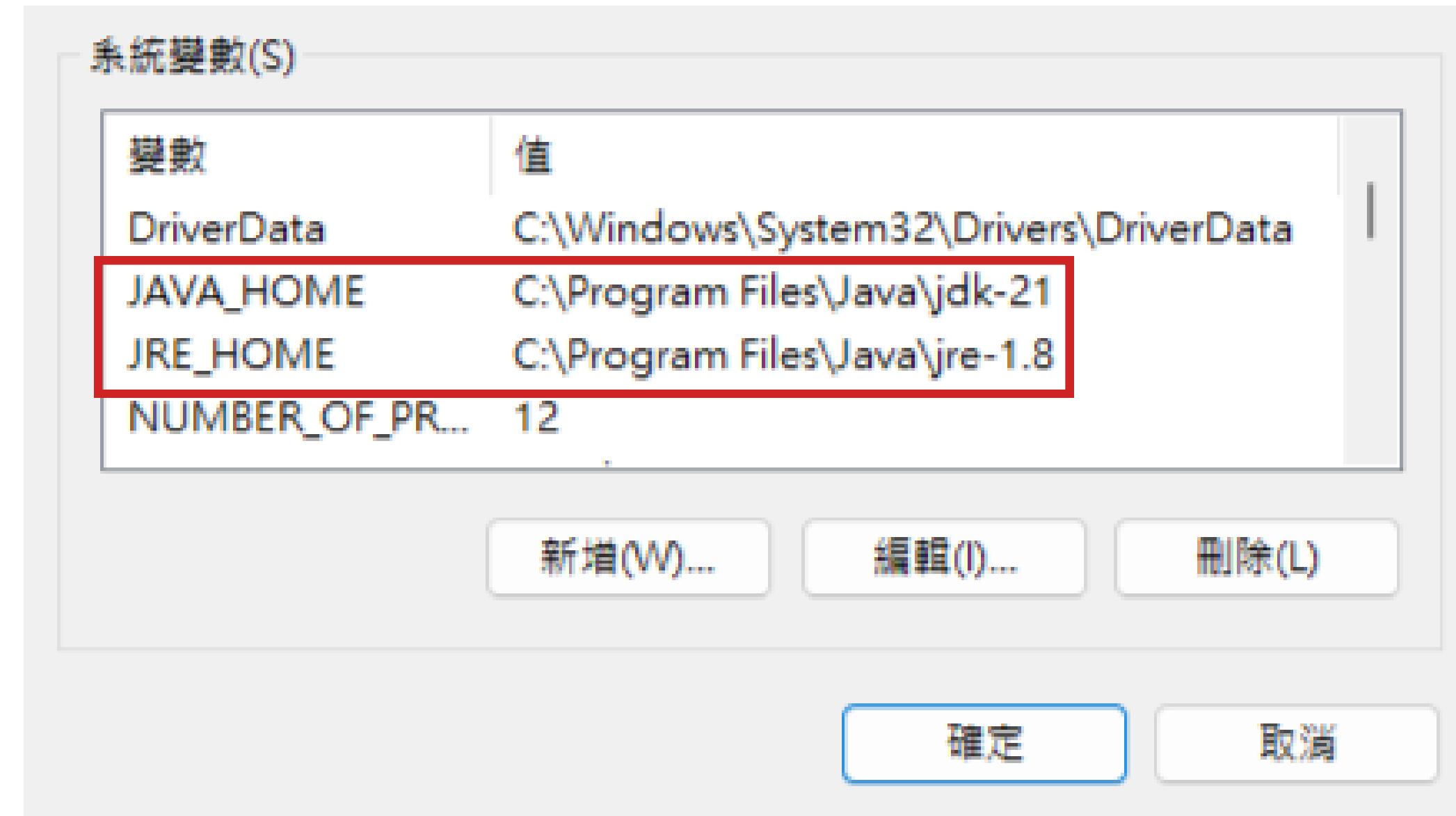
1. 下載 Java 和 Maven (java 21.0.1 2023-10-17 LTS \ Maven 3.9.6)

- <https://www.java.com/en/download/>
- <https://dlcdn.apache.org/maven/maven-3/3.9.6/binaries/apache-maven-3.9.6-bin.zip>

2. JAVA 安裝完成後，需要在環境變數裏新增兩個系統變數：

- 變數名稱：JAVA_HOME，變數值：C:\Program Files\Java\jdk-21
- 變數名稱：JRE_HOME，變數值：C:\Program Files\Java\jre-1.8

Local Install



Local Install

3. 下載 Maven 壓縮檔後：

- 解壓縮 Maven 到你想要的位置，例：C:\Program Files\apache-maven-3.9.6-bin
- 解壓縮后，Maven 的 bin 檔案記得放在環境變數的 Path 裏

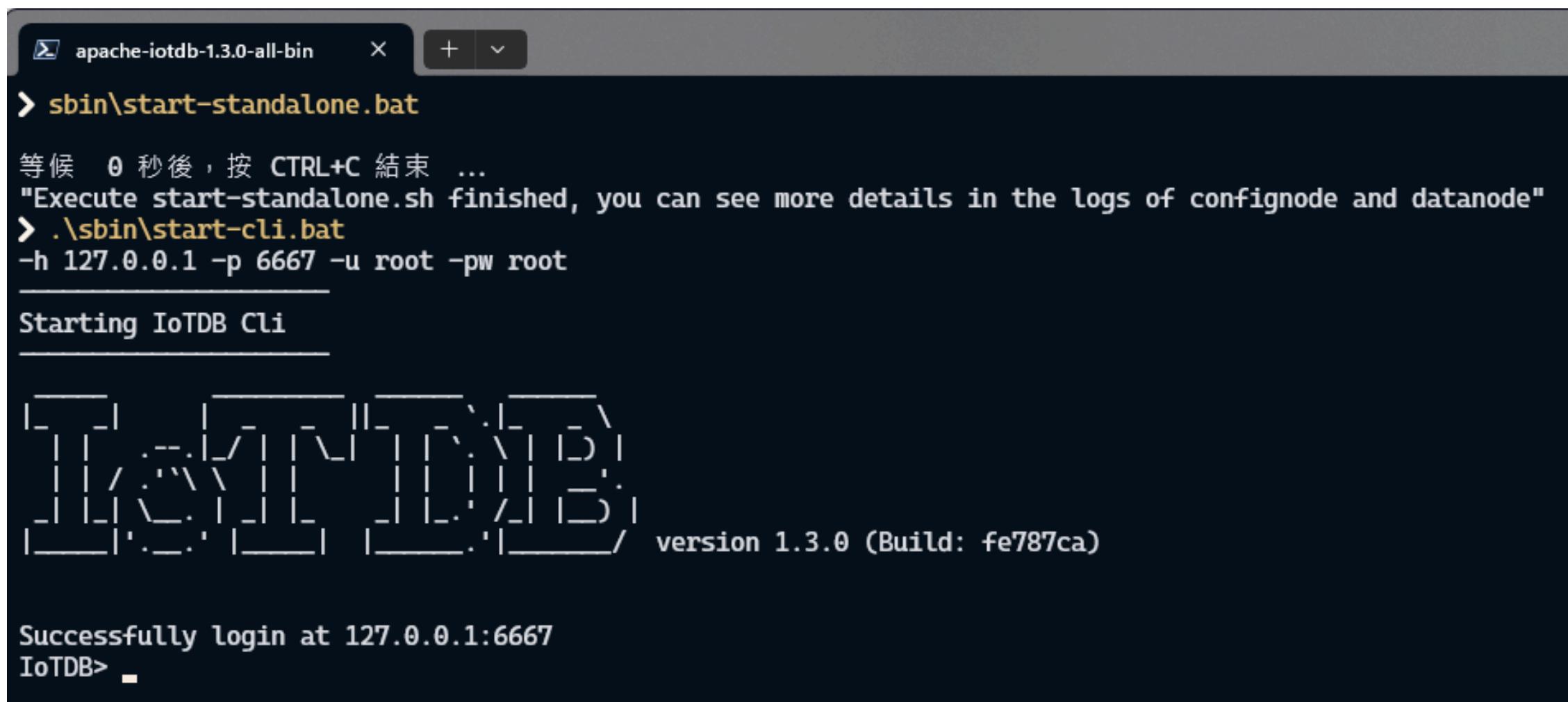
4. 下載 IoTDB

- <https://www.apache.org/dyn/closer.cgi/iotdb/1.3.0/apache-iotdb-1.3.0-all-bin.zip>
- 解壓縮在一個你記得的位置，範例：C:\Users\aaagui_666\apache-iotdb-1.3.0-all-bin

Local Install

5. 運行 IoTDB

- 在 apache-iotdb-1.3.0-all-bin 目錄下，執行：sbin\start-standalone.bat
 - 會跳出兩個視窗，但不用關掉
- 在 apache-iotdb-1.3.0-all-bin 目錄下，執行：sbin\start-cli.bat



The screenshot shows a terminal window titled "apache-iotdb-1.3.0-all-bin". It displays the command "sbin\start-standalone.bat" being run, followed by a message indicating the process has finished. Then, the command "sbin\start-cli.bat" is run with parameters "-h 127.0.0.1 -p 6667 -u root -pw root". The terminal then outputs "Starting IoTDB Cli" and a decorative logo consisting of various symbols like brackets and dots. Below the logo, it says "version 1.3.0 (Build: fe787ca)". Finally, it shows a successful login message: "Successfully login at 127.0.0.1:6667" and the prompt "IoTDB>".

```
apache-iotdb-1.3.0-all-bin
> sbin\start-standalone.bat
等候 0 秒後，按 CTRL+C 結束 ...
"Execute start-standalone.sh finished, you can see more details in the logs of confignode and datanode"
> .\sbin\start-cli.bat
-h 127.0.0.1 -p 6667 -u root -pw root
Starting IoTDB Cli
version 1.3.0 (Build: fe787ca)
Successfully login at 127.0.0.1:6667
IoTDB>
```

Reference

Only use Generative AI in conclusion.