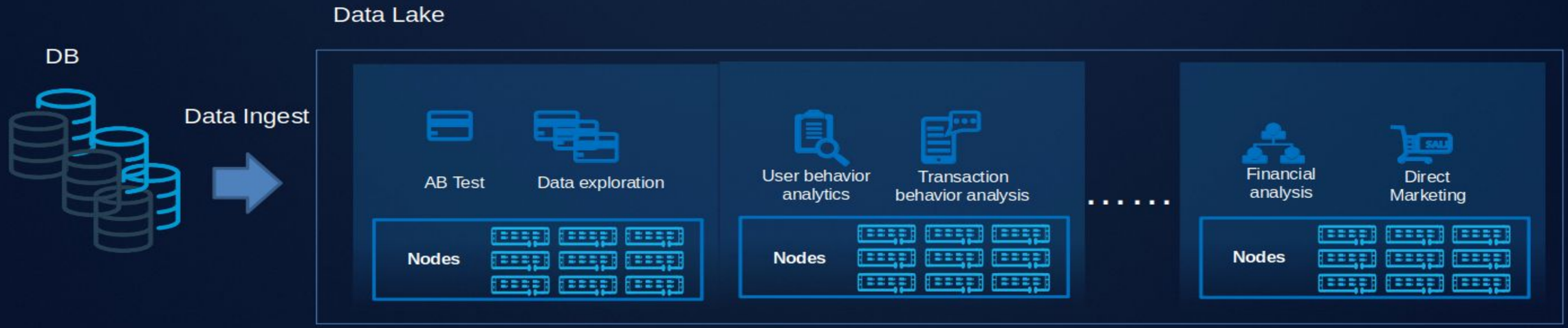
Apache CarbonData

Data storage for ACID ingest, Faster query and machine learning

Ajantha Bhat U
Apache CarbonData Committer & Senior Technical Lead
Huawei Technologies

CarbonData Background

Challenges



How to deal with the streaming mutable data

- Data update/merge is slow
- Only T+1 or even T+2 data synchronization can be supported

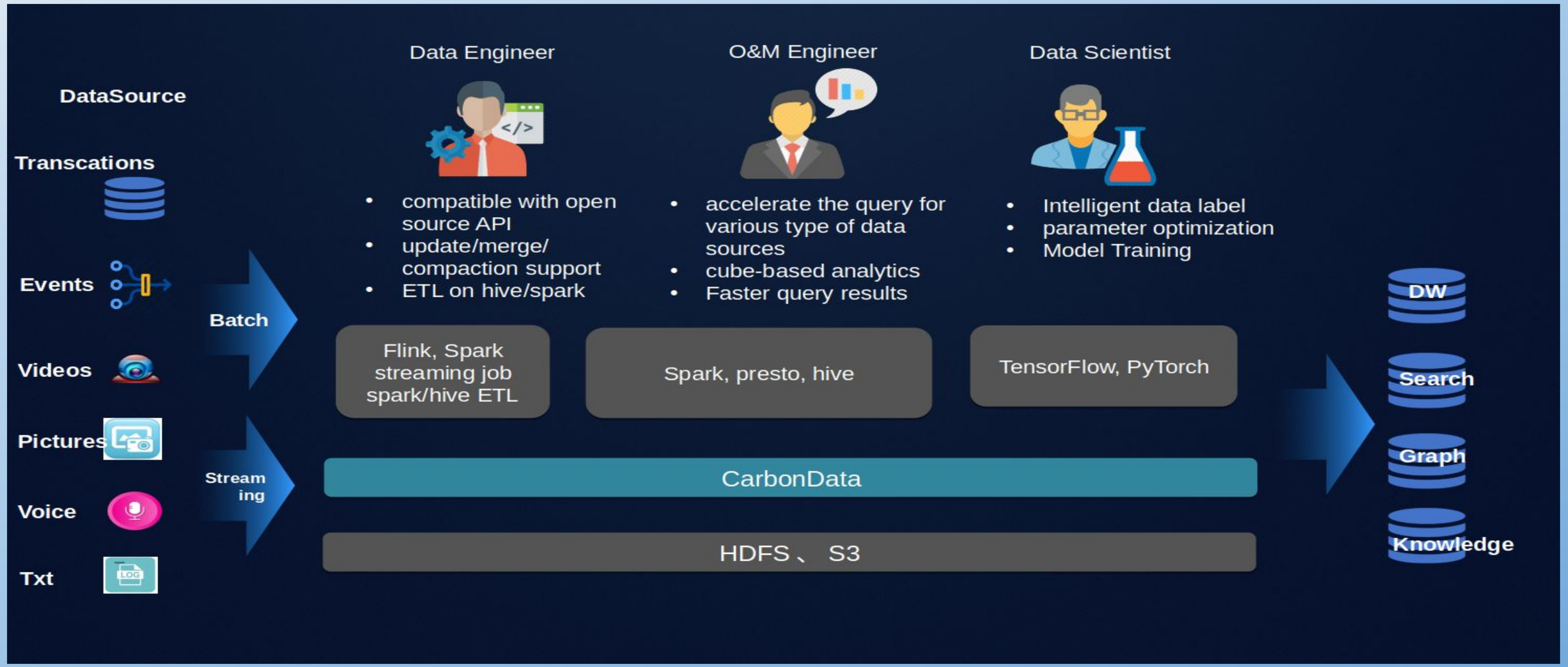
What technologies to use for indexing the data

- High costs and difficult O&M for NoSQL
- Query is Slow without Index (Spark, Hive and etc.)

How to analyze data with ML platforms

- Multiple copies of data is stored in data lake based the AI engine

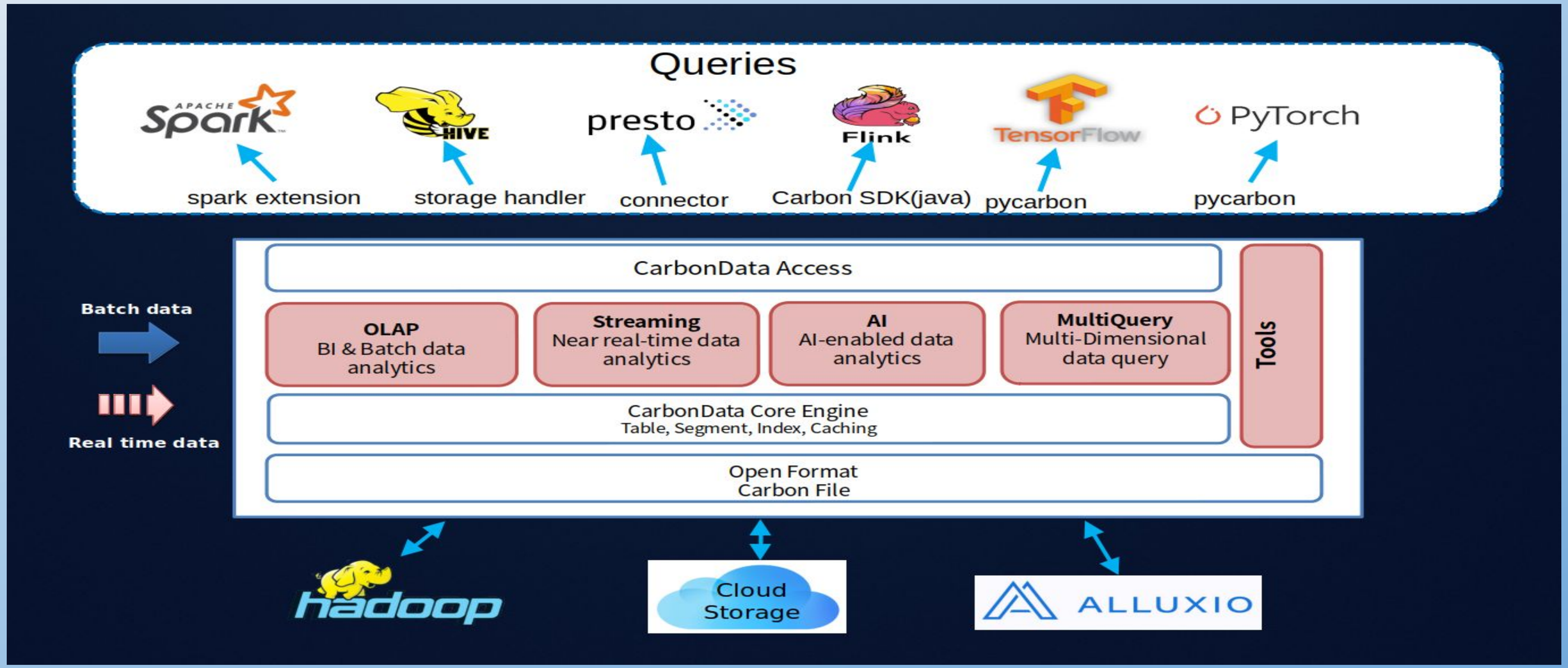
One Data, Zero Migration



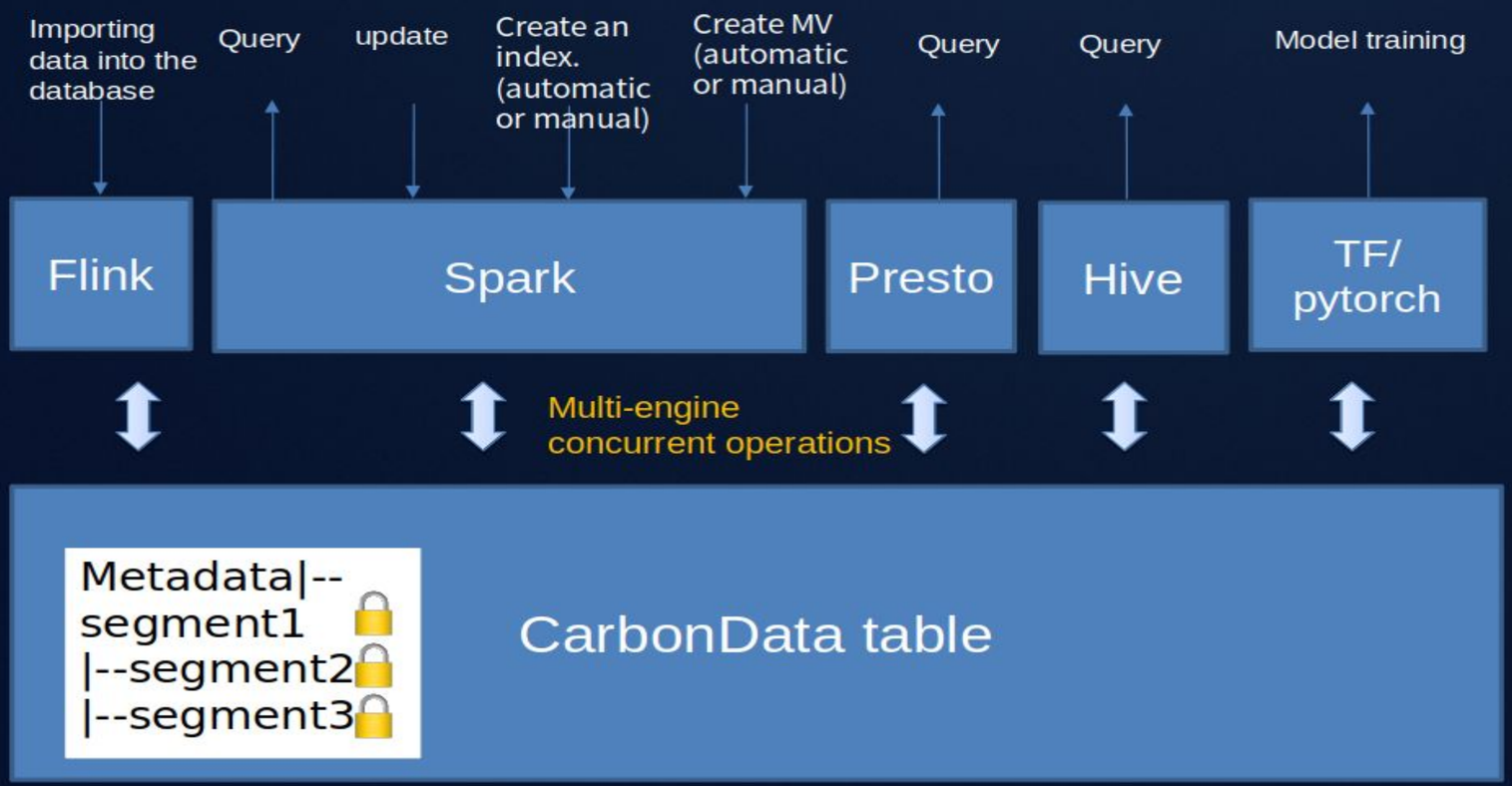
Apache CarbonData Project [Oldest Data Lake solution]

- 2014–2016: Internal R&D project in Huawei
- 2016-2017: Entered the Apache incubator and became an excellent incubator project of the year.
- June 2017: Become a top Apache project.
- Since 2018: PB-level large enterprises/ISVs have gone live > 50; Maximum number of records in a single table > 15 trillion
- Contributors from:

CarbonData Architecture [Storage Engine]



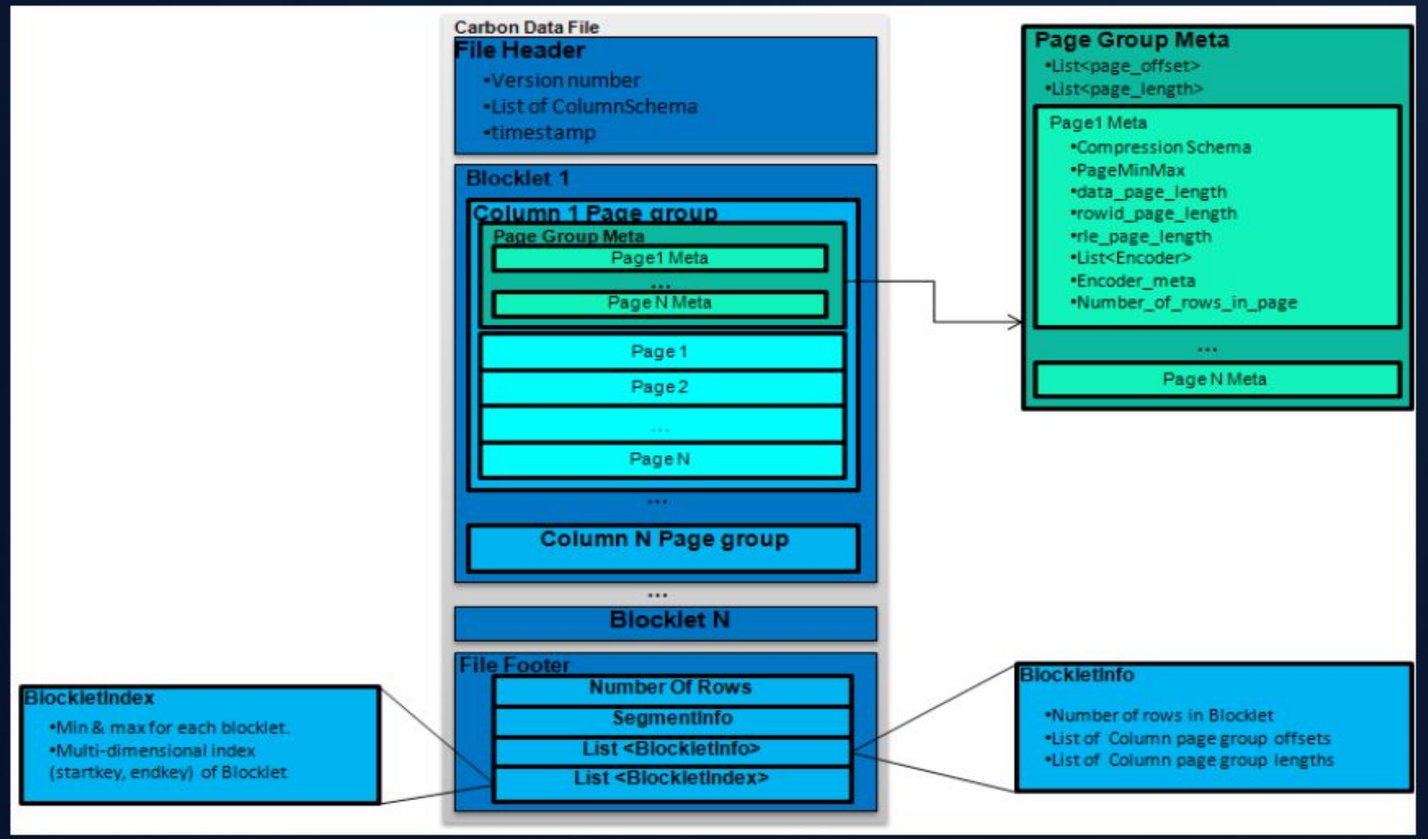
Storage Engine



All operations support the ACID capability.

1. Spark
   - Batch processing, interactive analysis, and machine learning
   - Insert, Update, delete, compaction, and merge
   - Create indexes and MVs.
2. Flink: streaming data import and real-time analysis
3. Presto: Interactive query
4. Hive: large-scale ETL
5. Tensorflow and pytorch: model training
6. SDK: Java, Python, C++

CarbonData columnar file format

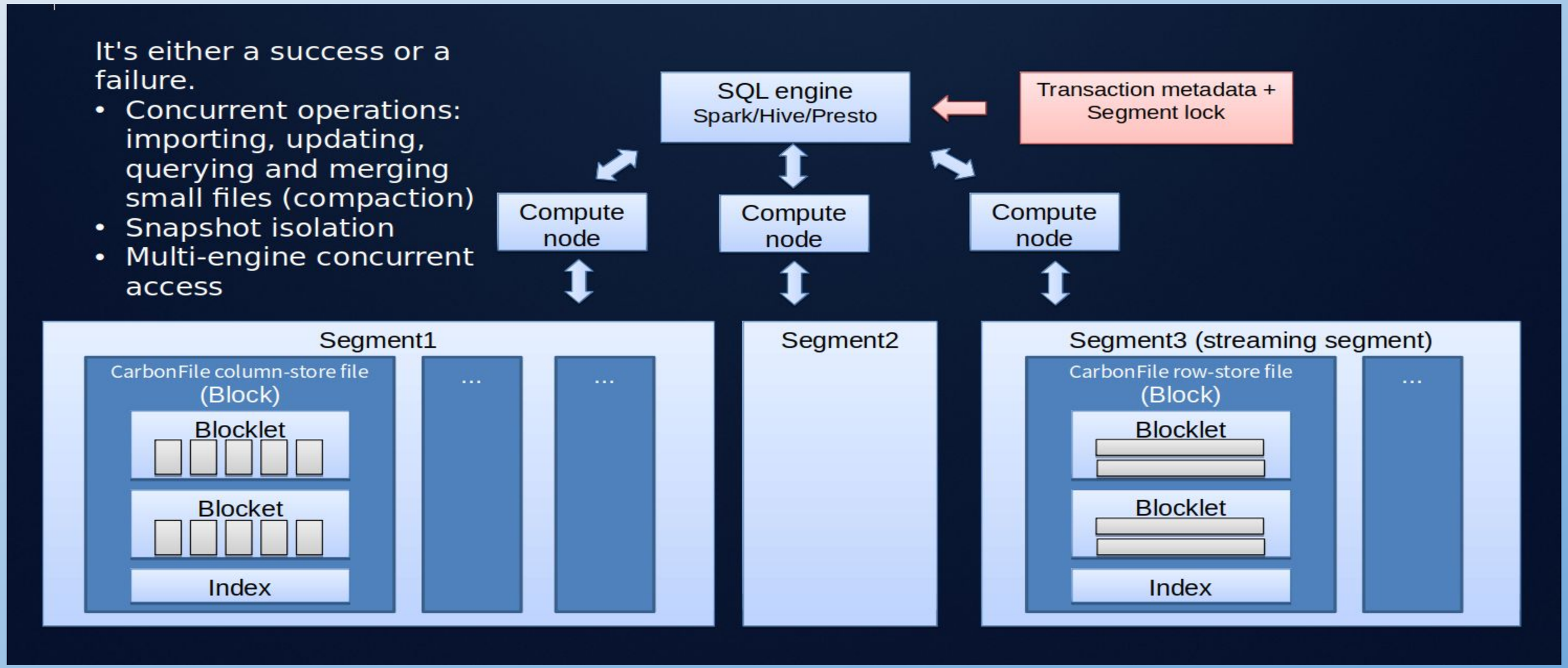


- Built-in Index columnar storage
  - Suitable for both batch and point query
- Built-in Index Type:
  - Min/Max index
  - Inverted index
- Encoding & Compression:
  - Local Dictionary, RLE, Delta
  - Snappy compression
- Data Type:
  - Primitive type and nested type
- Schema Evolution:
  - Add, Remove, Rename columns

- **Blocklet**: Set of rows stored in columnar format
- **Page** : Data for one column in a Blocklet (3200 entries or based on size)
- **Footer** : Metadata information

ACID Ingest

CarbonData ACID Ingest



Spark Extension

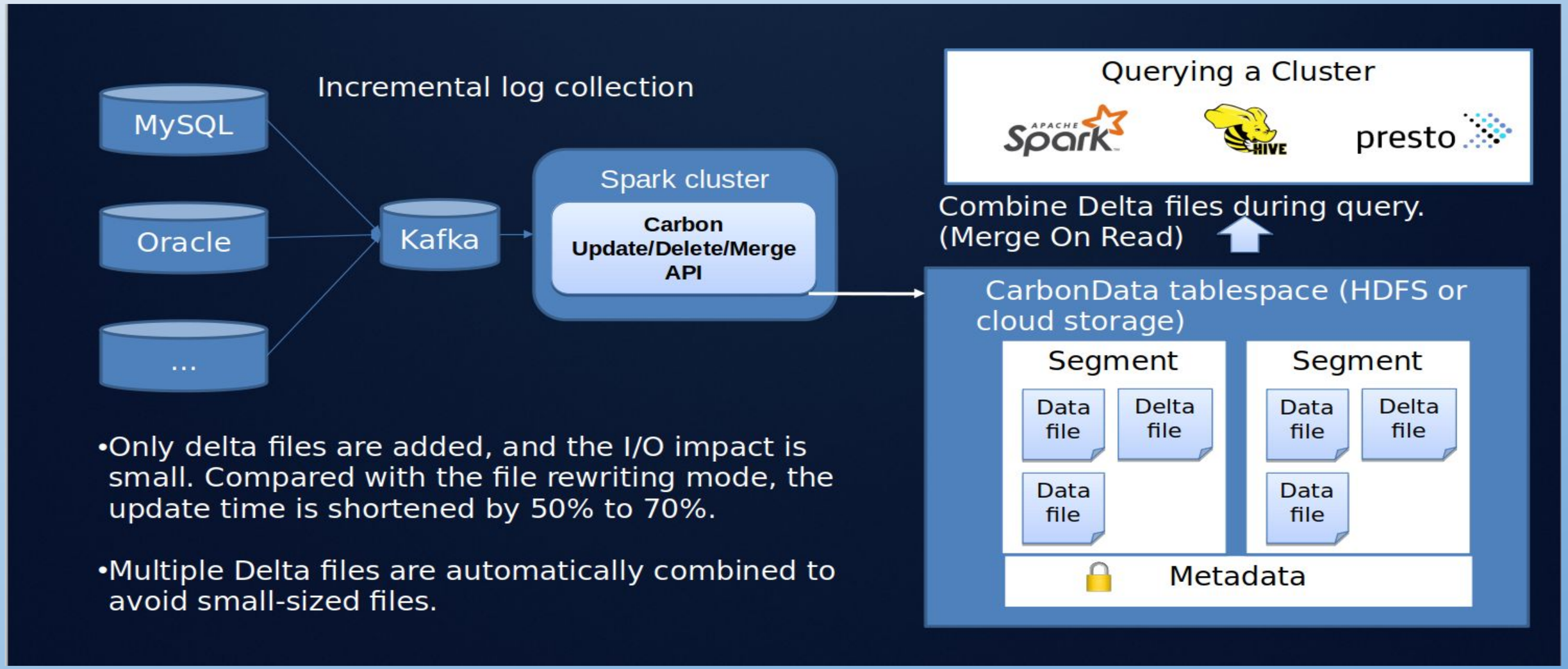
- Standard extension mode of the Spark community

```
// CarbonData 2.0
val spark = SparkSession
.builder()
.master(masterUrl)
.enableHiveSupport()
  .config("spark.sql.extensions",
"org.apache.spark.sql.CarbonExtensions")
.getOrCreate ()
```

All CarbonSession features are supported.

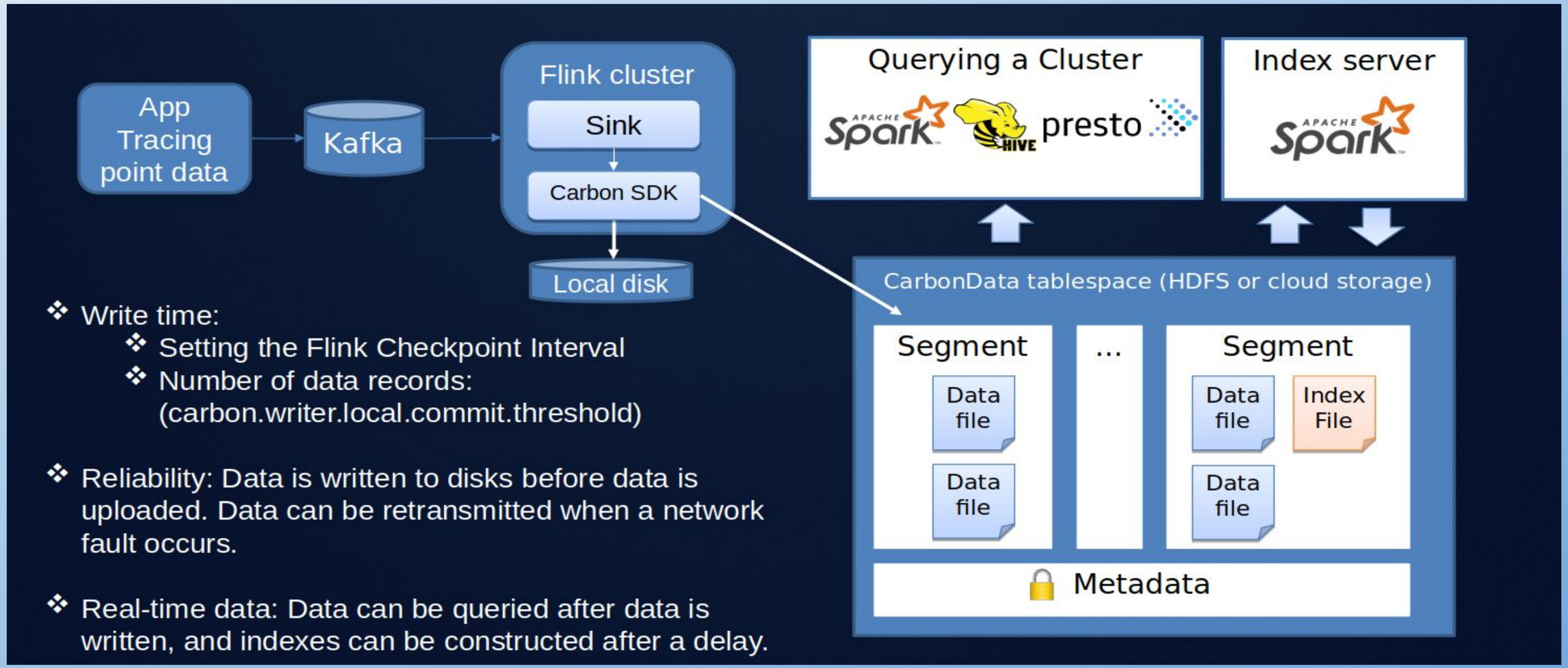
- Ingesting the Parser
- Injection optimization rule
- Inject physical planner

Spark Streaming and real-time database data synchronization



- Only delta files are added, and the I/O impact is small. Compared with the file rewriting mode, the update time is shortened by 50% to 70%.
- Multiple Delta files are automatically combined to avoid small-sized files.

Flink + CarbonData real-time streaming import



- Write time:
  - Setting the Flink Checkpoint Interval
  - Number of data records: (carbon.writer.local.commit.threshold)
- Reliability: Data is written to disks before data is uploaded. Data can be retransmitted when a network fault occurs.
- Real-time data: Data can be queried after data is written, and indexes can be constructed after a delay.

Merge API Example (CDC and SCD type 2)

change table

id	change_type	value
101	D	
100	U	'amy'
102	I	'tony'

merge

target table

id	value
100	'bob'
101	'jack'

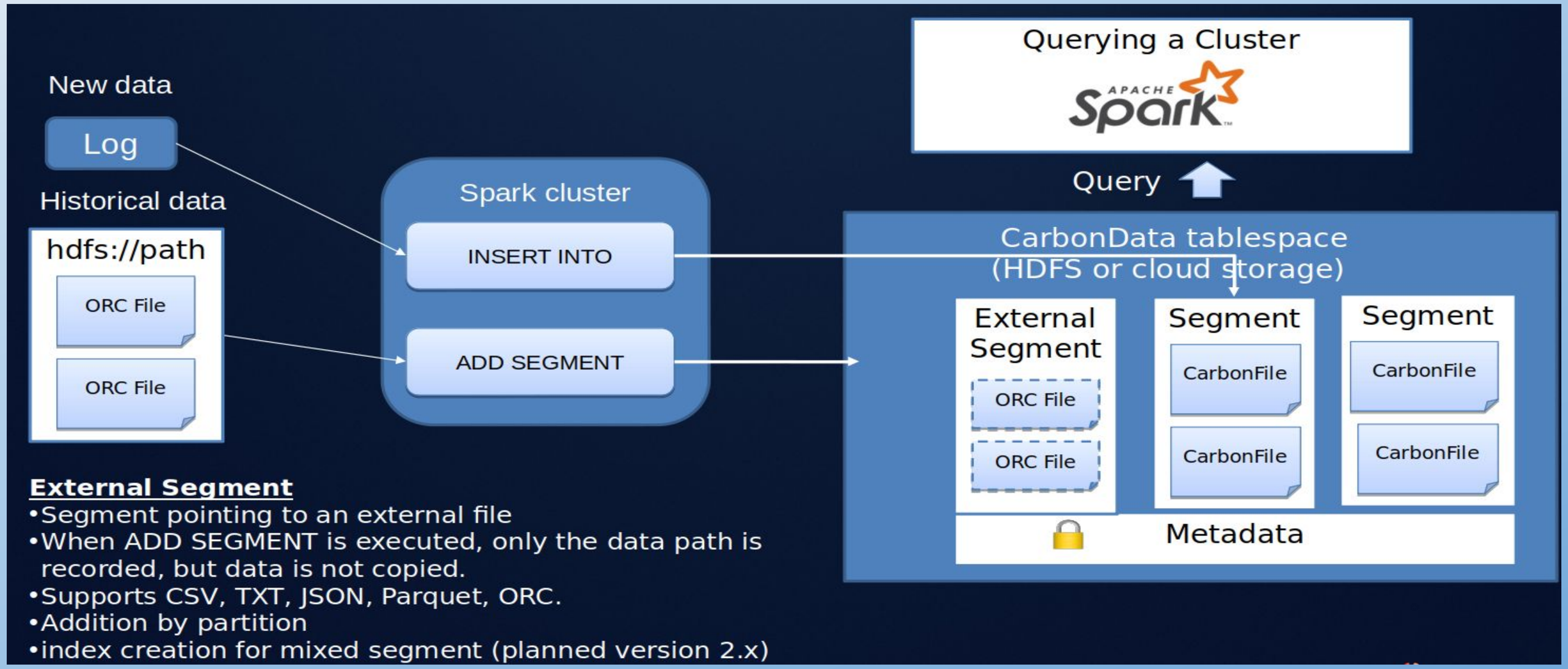
=

Updated target table

id	value
100	'amy'
102	'tony'

```
// Merge data in the change table to the target table.
targetDataFrame.as("A")
.merge(changeDataFrame.as("B"), "A.id = B.id")
.whenMatched("B.change_type = 'D'")
.delete()
.whenMatched("B.change_type = 'U'")
.updateExpr(Map("id" -> "B.id", "value" -> "B.value"))
.whenNotMatched("B.change_type ='I'")
.insertExpr(Map("id" -> "B.id", "value" -> "B.value"))
.execute()
```

Mixed format table (beta)

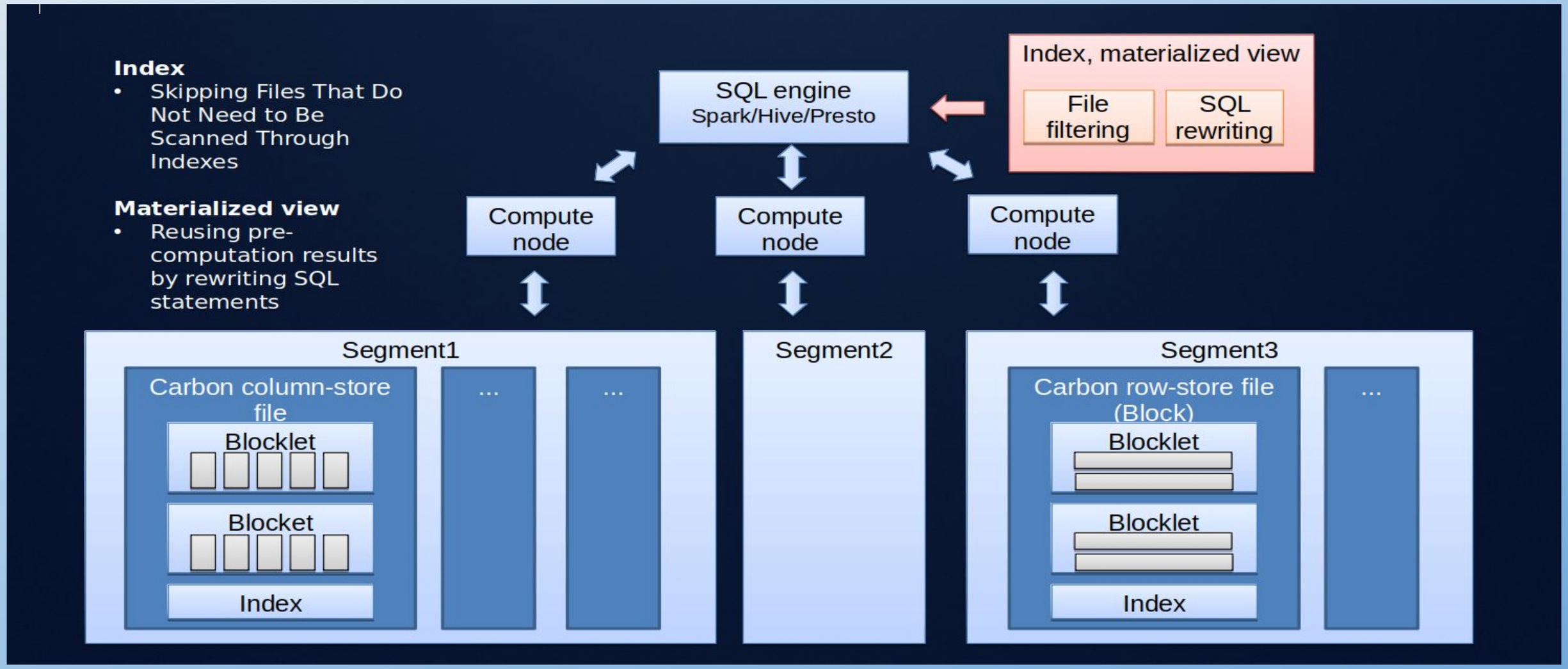


External Segment

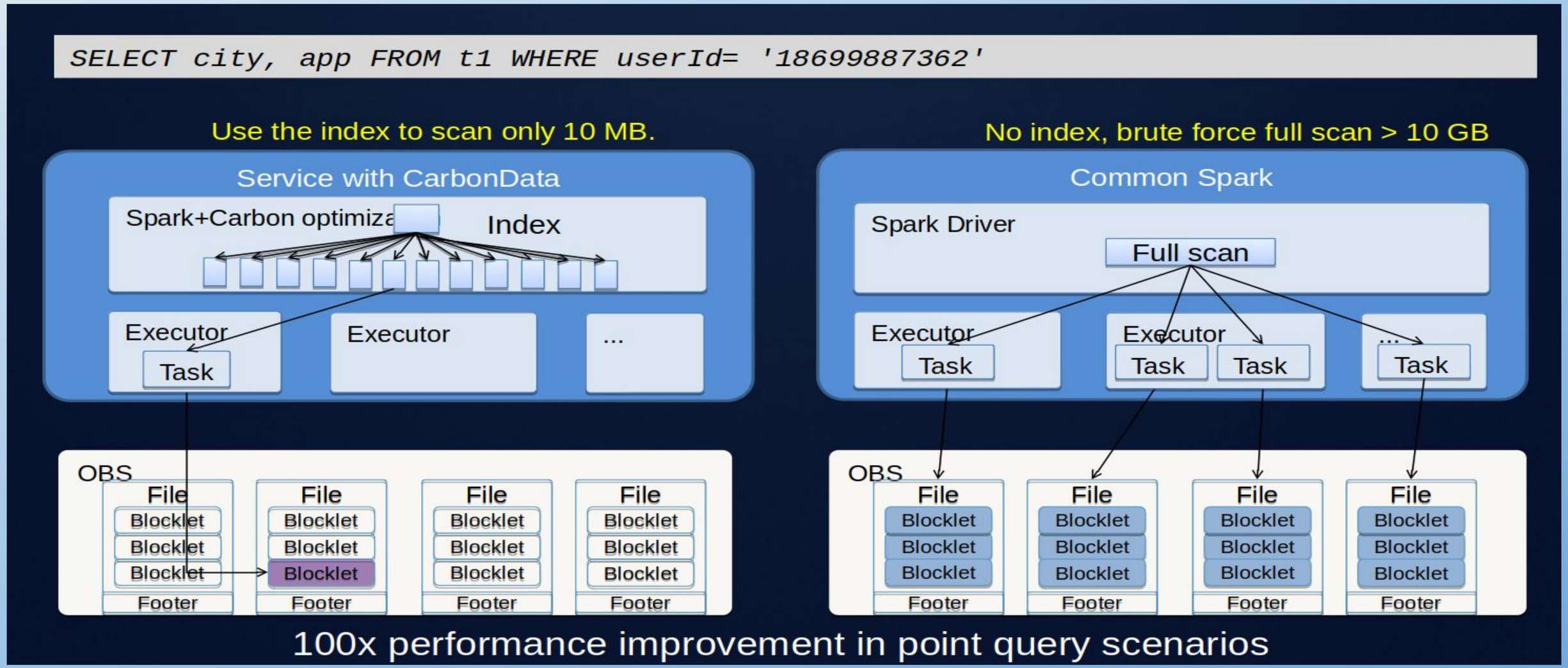
- Segment pointing to an external file
- When ADD SEGMENT is executed, only the data path is recorded, but data is not copied.
- Supports CSV, TXT, JSON, Parquet, ORC.
- Addition by partition
- index creation for mixed segment (planned version 2.x)

Faster Query Processing

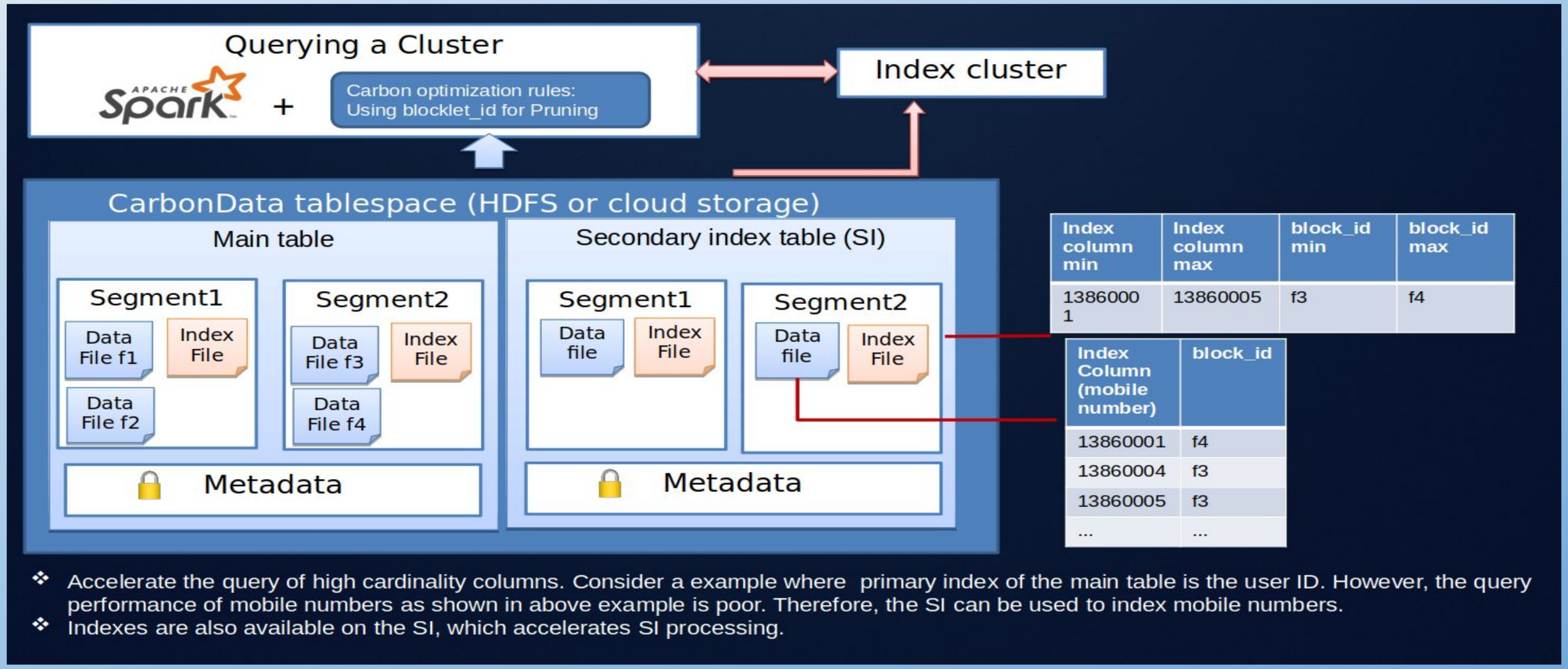
CarbonData for Faster query



Carbon index optimization [Min-max pruning, CG, FG indexes]



Secondary index [Location index table]



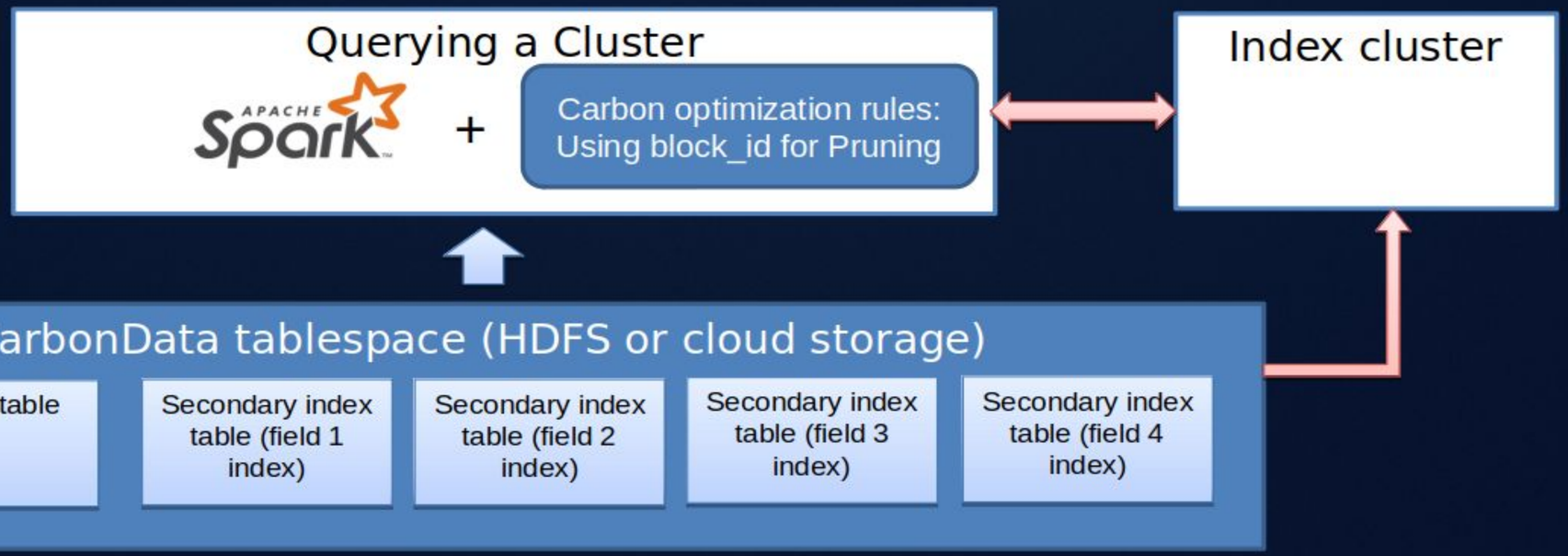
Index column min	Index column max	block_id min	block_id max
13860001	13860005	f3	f4

Index Column (mobile number)	block_id
13860001	f4
13860004	f3
13860005	f3
...	...

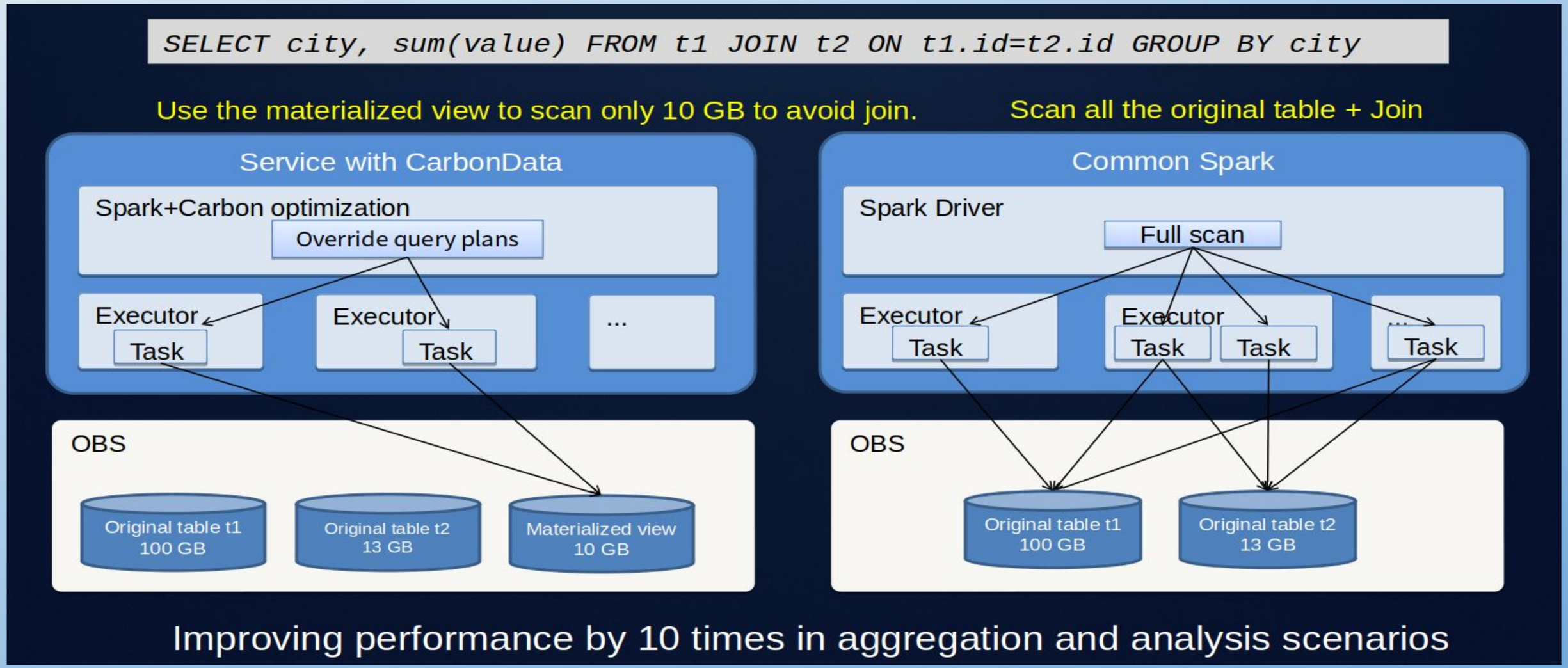
- Accelerate the query of high cardinality columns. Consider a example where primary index of the main table is the user ID. However, the query performance of mobile numbers as shown in above example is poor. Therefore, the SI can be used to index mobile numbers.
- Indexes are also available on the SI, which accelerates SI processing.

Multi-dimensional filtering using secondary indexes

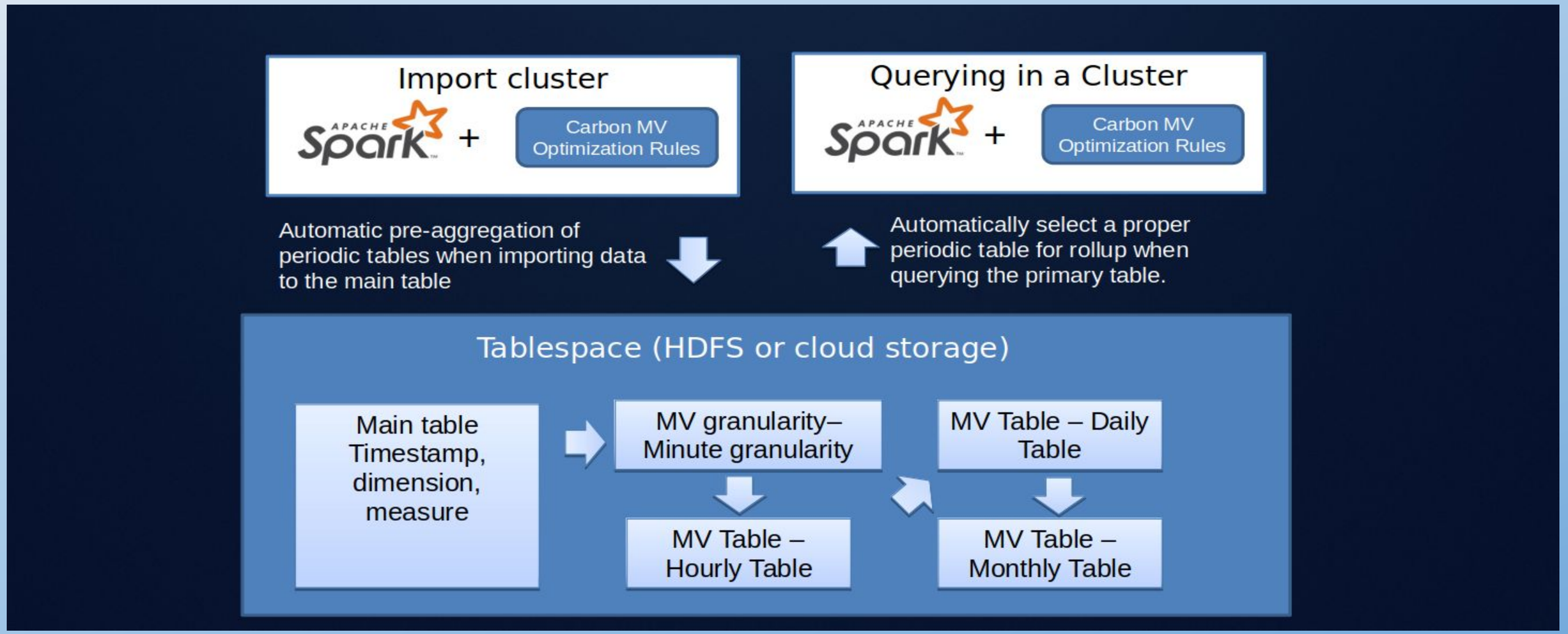
// Use the secondary index for filtering.
SELECT... WHERE: The value of field 1 is 10 and the value of field 2 is 20. Join two index tables, and then query the primary table.
SELECT... WHERE: field 1 = 10, field 3 = 30, or field 4 = 40, perform union between two index tables, and then query the primary table.



CarbonData materialized view optimization (MV)



Time series supported by MV



Time series MV example

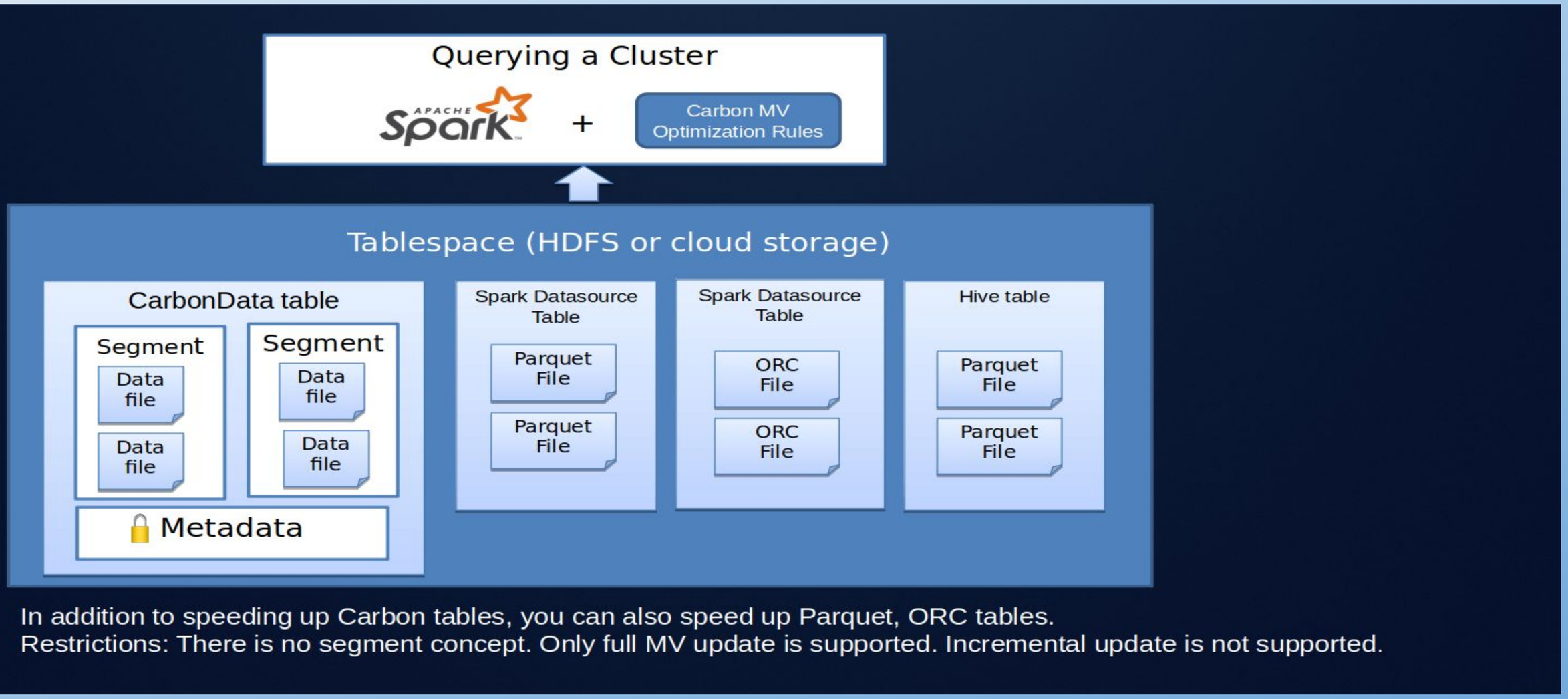
Granularity
year
month
week
day
hour
thirty_minute
fifteen_minute
ten_minute
five_minute
minute
second

```
// Create a materialized view.
CREATE MATERIALIZED VIEW avg_sales_minute
AS
SELECT timeseries(order_time, 'minute'),
avg(price)
FROM sales
GROUP BY series(order_time, 'minute')

// The following query statement uses the
materialized view:
SELECT timeseries(order_time, 'hour'),
avg(price)
FROM sales
GROUP BY series(order_time, 'hour')
```

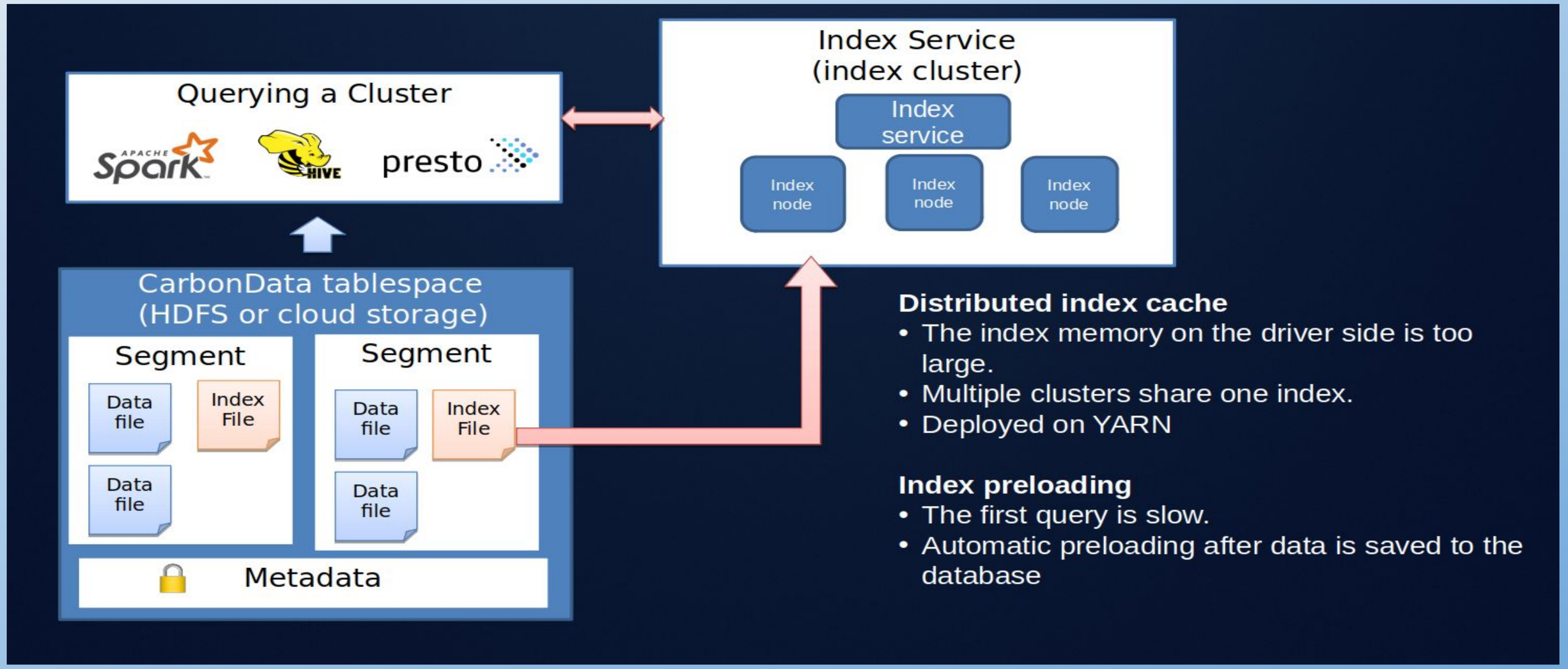
Restrictions: The time series MV does not support join statements and is replaced by common MVs.

MV supports non-Carbon tables

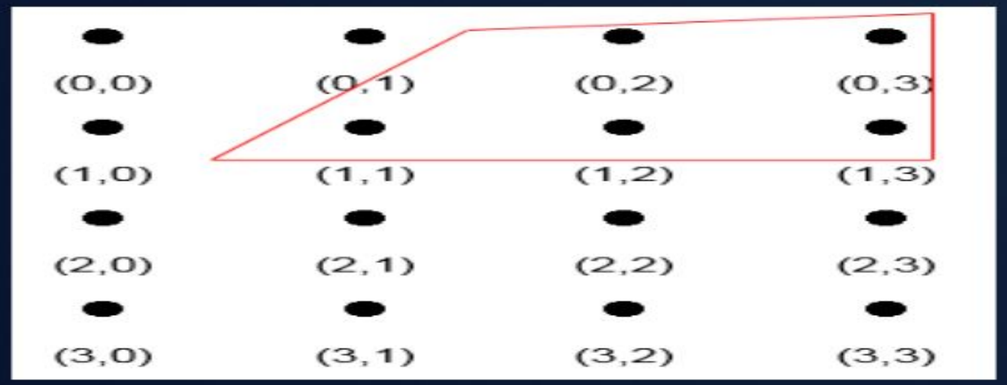


In addition to speeding up Carbon tables, you can also speed up Parquet, ORC tables.
Restrictions: There is no segment concept. Only full MV update is supported. Incremental update is not supported.

Index Service

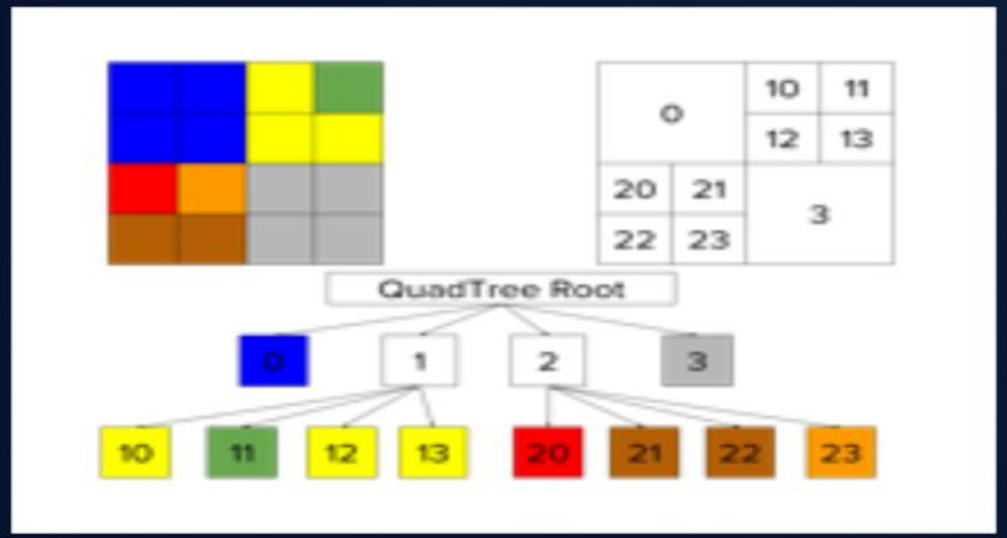


Geospatial support



Records to hit with query are highlighted:

latitude, longitude						
0,0		0,1		**0,2**		**0,3**
1,0		**1,1**		**1,2**		**1,3**
2,0		2,1		2,2		2,3
3,0		3,1		3,2		3,3
						end



- pluggable index generation support for geo spatial longitude, latitude columns (default Z order implementation)
- polygon query filter push down to scan layer for faster query performance

Adaptive & Delta encoding to reduce store size

C1: long
0x00000002
0x00000007
0x00000009
0x00000003
0x00000005
0x00000007

Min: 2
Max: 9

Schema: long (8 bytes)
Actual usage: byte (1 byte)

C2: long
12892712
12892727
12892713
12892743
12892725
12892742

Min: 12892712
Max: 12892743

Schema: long (8 bytes)
Actual usage: byte (1 byte)
(The storage value is max-value.)

C3: double
12.32
21.42
32.12
42.43
54.32
14.32

Min: 12.32
Max: 54.32
Scale: 2

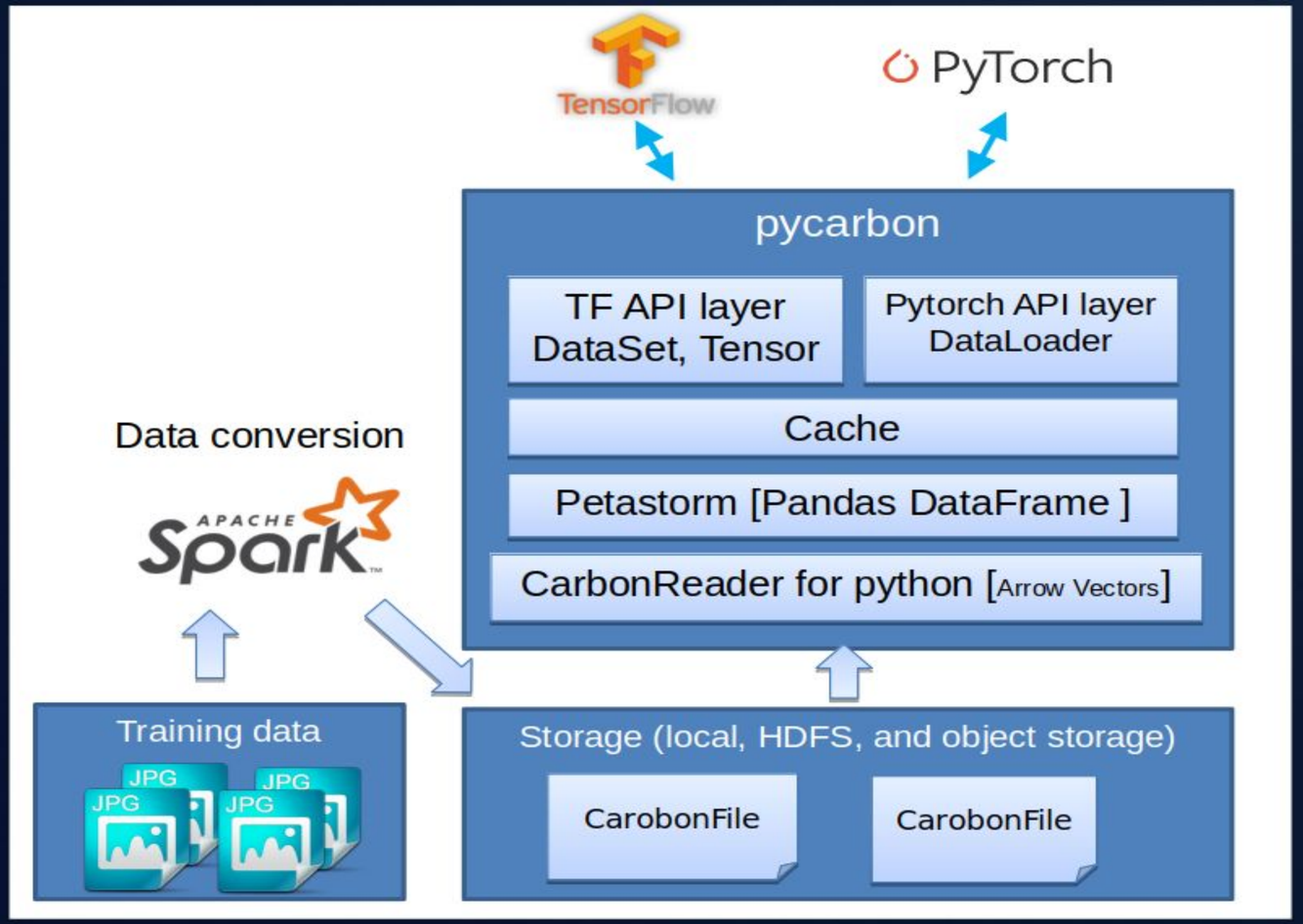
Schema: double (8 bytes)
Actual use: short (2 bytes)
(stored value x Math.pow(10, scale)

Other prominent features

- Global sort, local sort to sort the column data for better compression and faster query results
- Min max at segment level
- Specialized for cloud, it is not the flat folder. Virtually partitioned by segment concept and easy to prune the data for filter query as min max is maintained at segment level also.
- No need to list files as the metadata holds the file names.
- Reduce IO by having Index, merge index.
- Bitset pipe lining for multiple And filter

Machine Learning with CarbonData

Accelerated AI model training

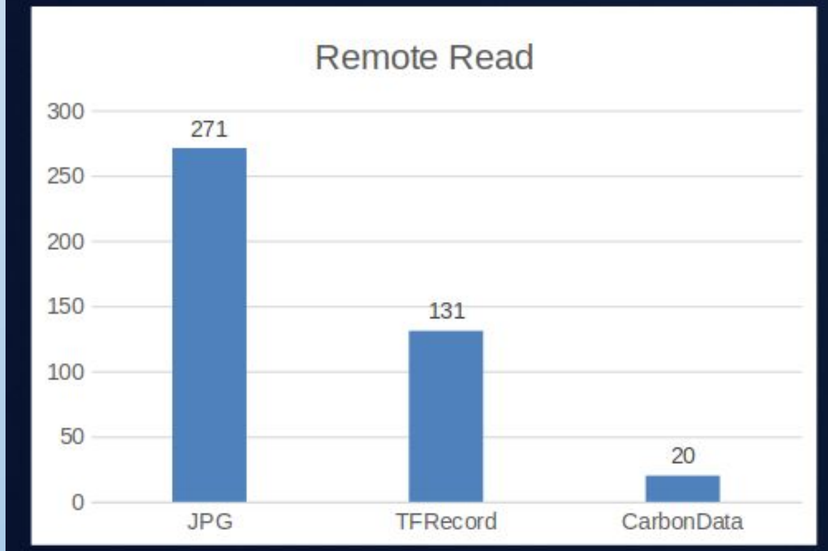


Facilitates model training by using the pycarbon + AI framework.

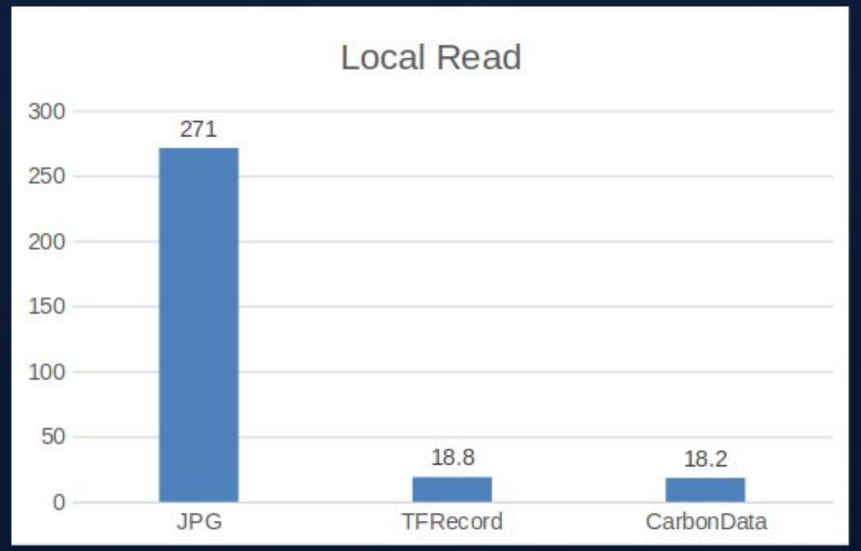
- After images are converted to Carbon files, the files are merged, which greatly improves the I/O efficiency.
- Cache: caches memory or local disks to avoid multiple remote read operations during training.
- Parallel processing: supports multi-thread parallel read.
- Out-of-order read: The read sequence of each training round is disordered, facilitating fast model convergence.
- Fast filtering: Compared with TFRecord, Carbon can quickly filter training sets based on column-store features.
- Supports interconnection with the TF and Pytorch native data structures.

ImageNet Dataset Read Performance Comparison

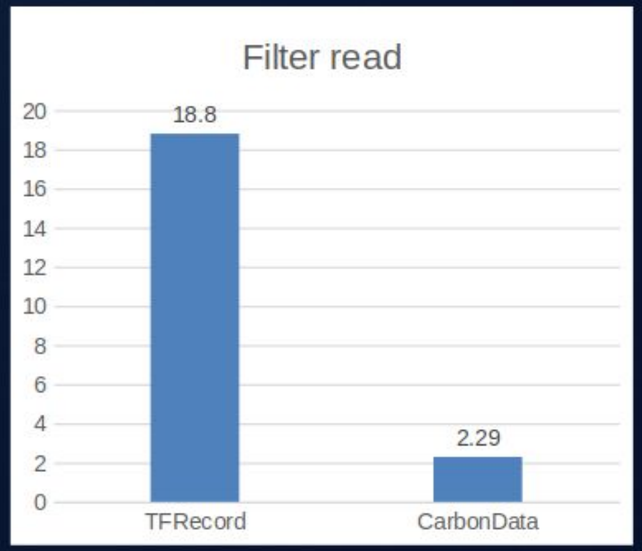
Dataset: 7800 images (1 GB) are extracted from ImageNet. Converted to Carbon and TFRecord files.
Field: 7 Columns: height, width, depth, imageName, imageBinary, txtName, txtContent
Storage: cloud storage (object storage)



10 times higher than JPG and 6 times higher than TFRecord



The analysis shows that TF does not support cloud storage.To avoid the TF bug, measure the download time and local read time.



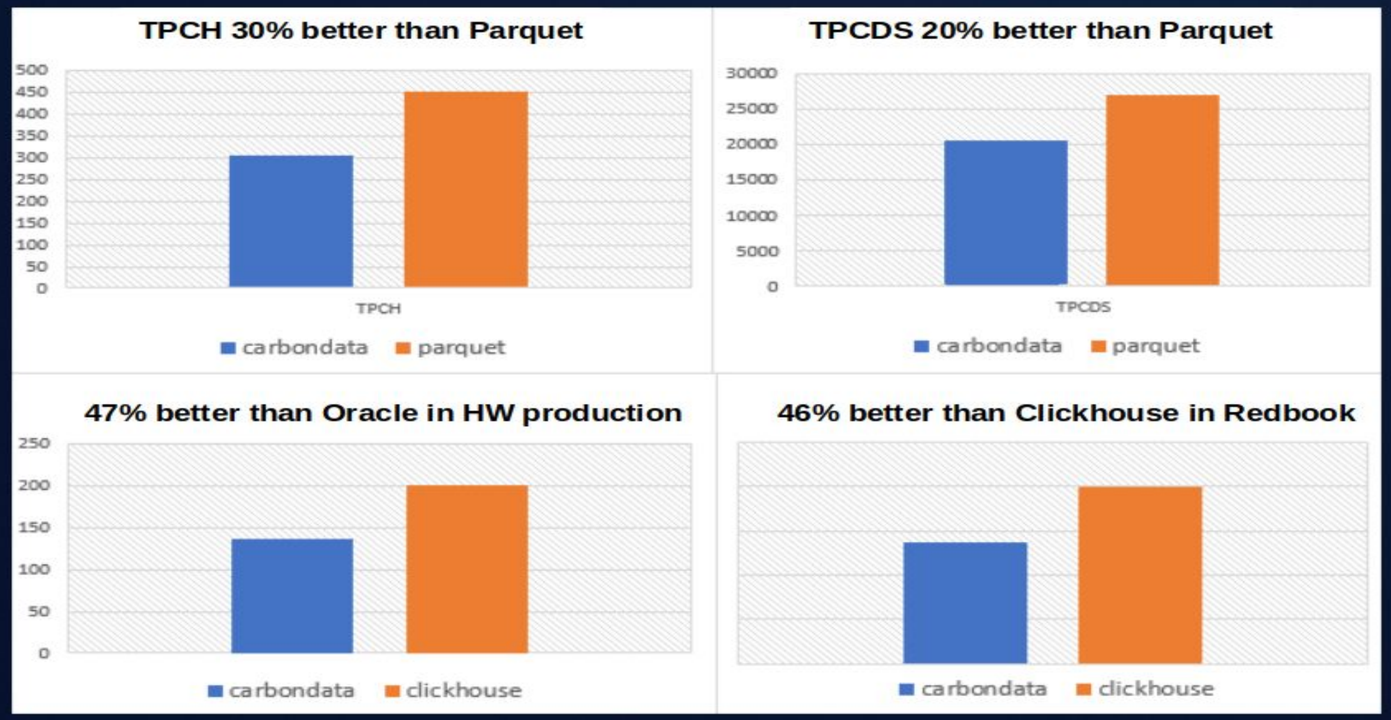
1300 images are filtered out from 7800 images as the training set. The I/O and time are six times shorter.

Customer Success stories

Customer success stories

✓ ETL and Ad-hoc Query better then Spark on Parquet/Oracle/Clickhouse

better compression, optimized for Object storage, etc.



- Default available in Huawei cloud and easy to integrate in any spark, hive, presto , flink services in GCP and AWS
- Used by Many customers in China like communication bank, Jinling, Redbook, Volkswagen, Huawei smart care, discovery, Alibaba
- Used by DBS bank Singapore

Summary

CarbonData Focus on Data Access,Analysis Performance and Big Data + AI Unified Storage

ACID Ingest

Empowers ingestion with **cloud-native ACID transactions and Streaming Merge**

Faster Query

Empowers Faster Query (seconds level on PB level data) with **Index, MV, and Index Service**

Machine Learning

With **Pycarbon,** Unified Storage for Bigdata and AI Engines

Special Thanks to the team [All the PMC members, Committers, contributors]

We love more community involvement & contributions

Subscribe to dev mailing list

- **Mail list:** dev@carbondata.apache.org, user@carbondata.apache.org
- **Mailing list Archive:** http://apache-carbondata-dev-mailing-list-archive.1130556.n5.nabble.com/
- **Slack:** https://join.slack.com/t/carbondataworkspace/shared_invite/zt-g8sv1g92-pr3GTvjrW5H9DVvNl6H2dg

Welcome any type of contribution: feature, documentation or bug report:

- **Code:** https://github.com/apache/carbondata
- **JIRA:** https://issues.apache.org/jira/browse/CARBONDATA
- **Website:** http://carbondata.apache.org
- **cwiki:** https://cwiki.apache.org/confluence/display/CARBONDATA/CarbonData+Home

Thank You

- Share this session on social media using **#DataLakeSummit**
- Interact with other attendees on Slack at **datalake-summit.slack.com**