Apache CarbonData 2.0

Agenda

- CarbonData Background
- 2.0 New Features
  - Data access
  - Data analysis
- Upgrade Suggestion

Apache CarbonData

- 2014–2016: Internal R&D
- 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:

Typical Data Analysis Scenarios

O&M

Data insight

BI report

Query Details

Interactive analysis

Batch calculation

Data

Production database Transactions

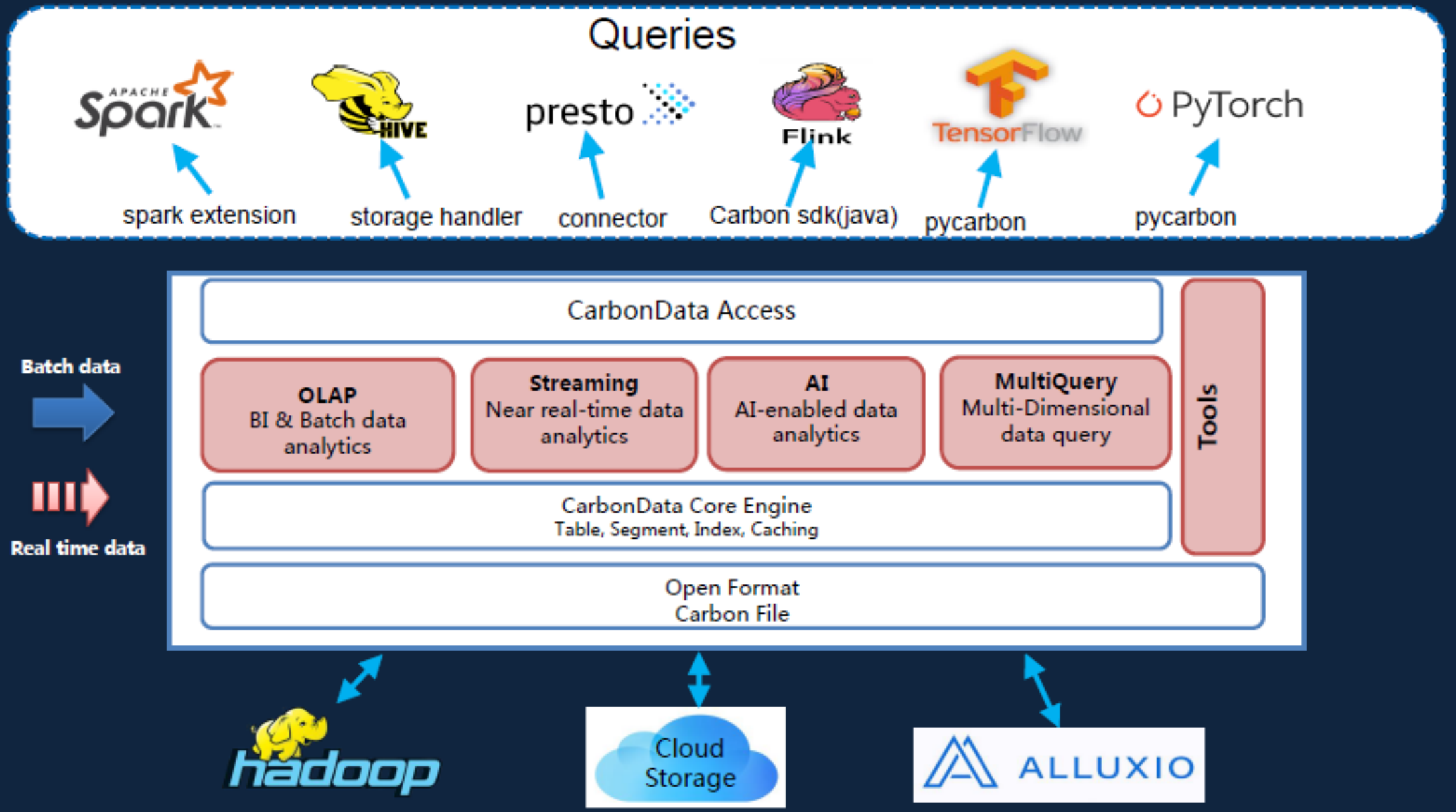
App logs

Log Analysis: Fault Locating
Feature: Real-time data query by user ID and device ID

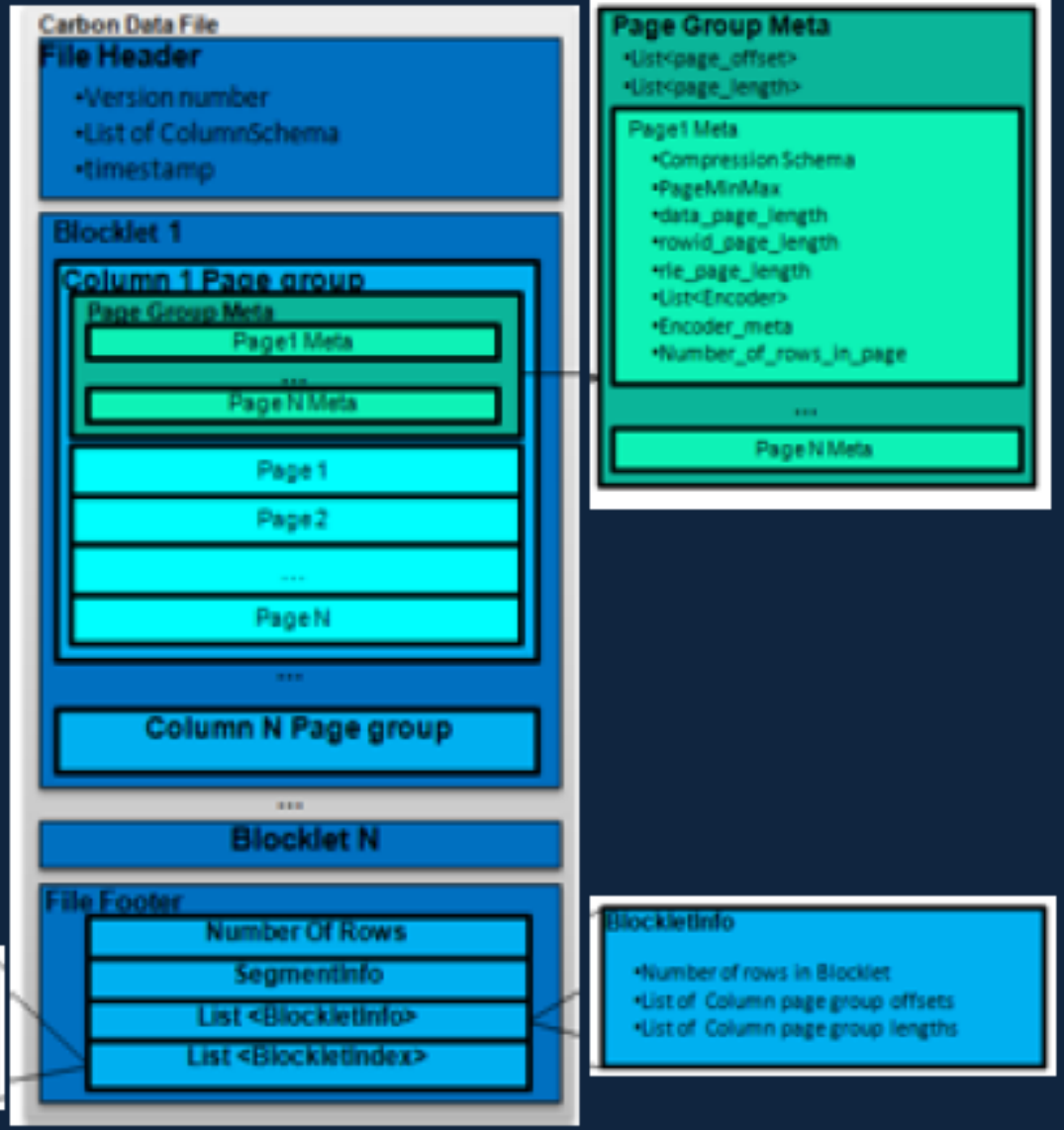
Interactive analysis: Generate insights and forecasts.
Features: multi-dimensional, variable mode, flexible computing, and massive data

Report calculation: BI report
Features: periodic summary statistics, service data change, and database data synchronization

Carbon Data Architecture



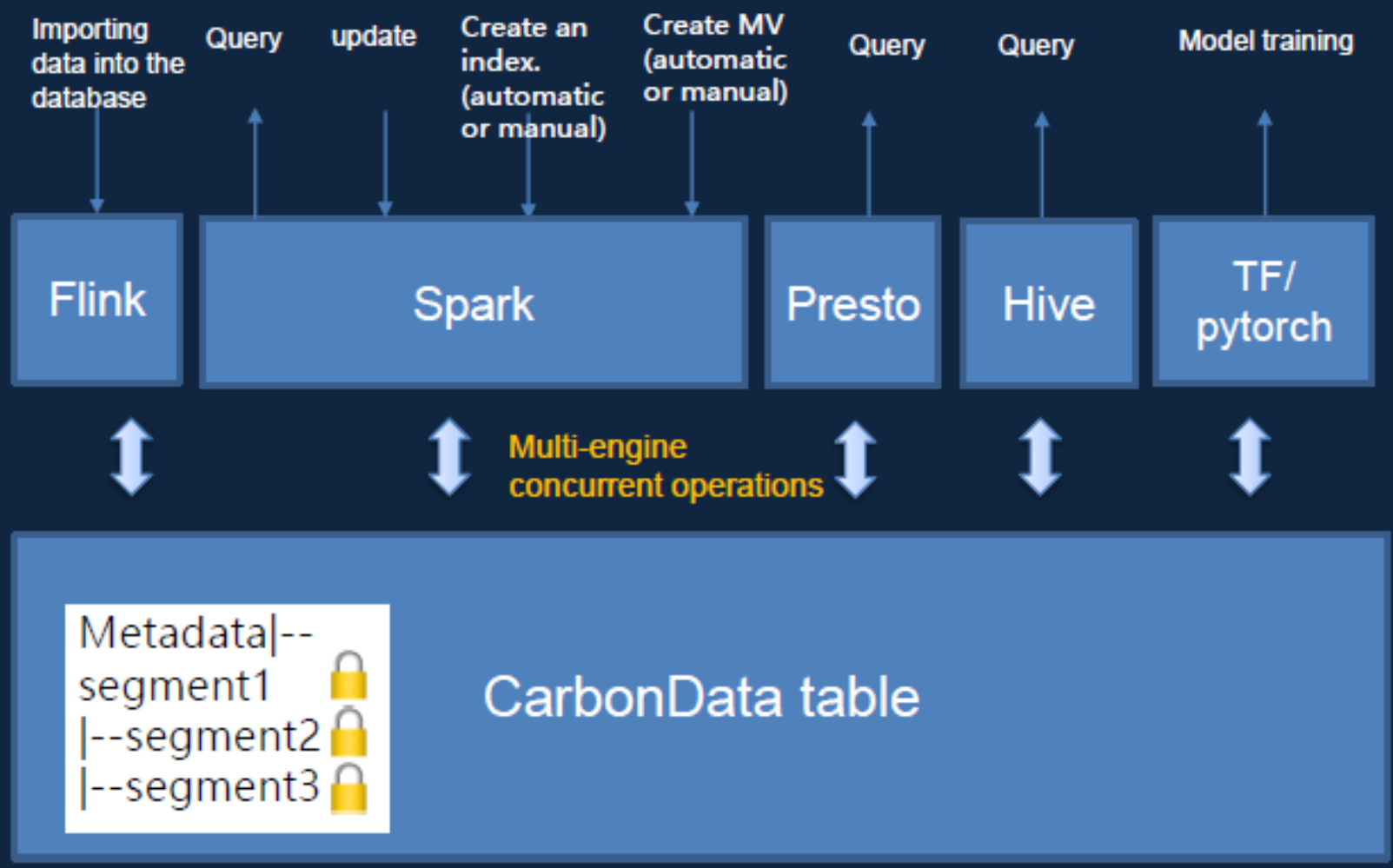
CarbonData columnar file format



- 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

CarbonData ACID



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

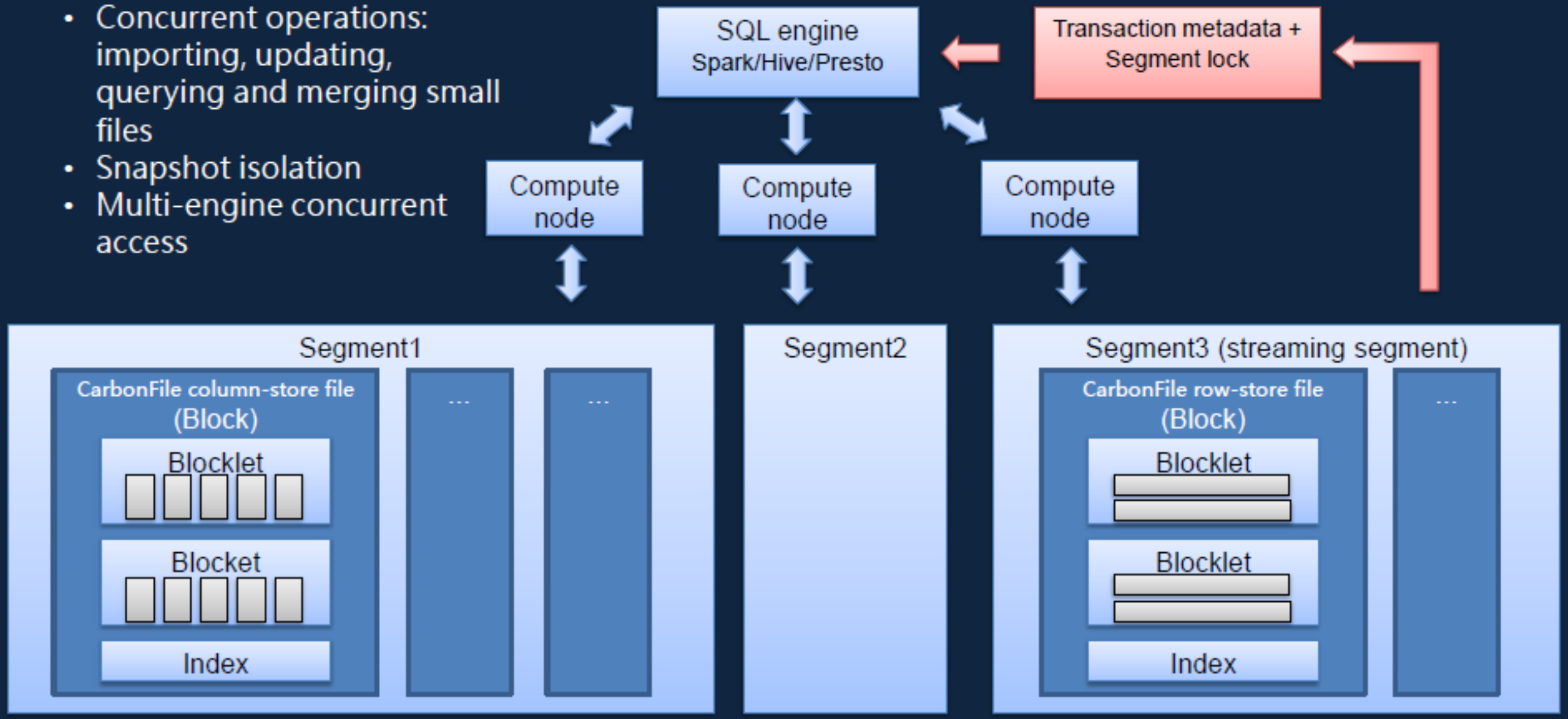
5.Tensorflow and pytorch: model training

6.SDK: Java, Python, C++

CarbonData ACID

It's either a success or a failure.

- Concurrent operations: importing, updating, querying and merging small files
- Snapshot isolation
- Multi-engine concurrent access



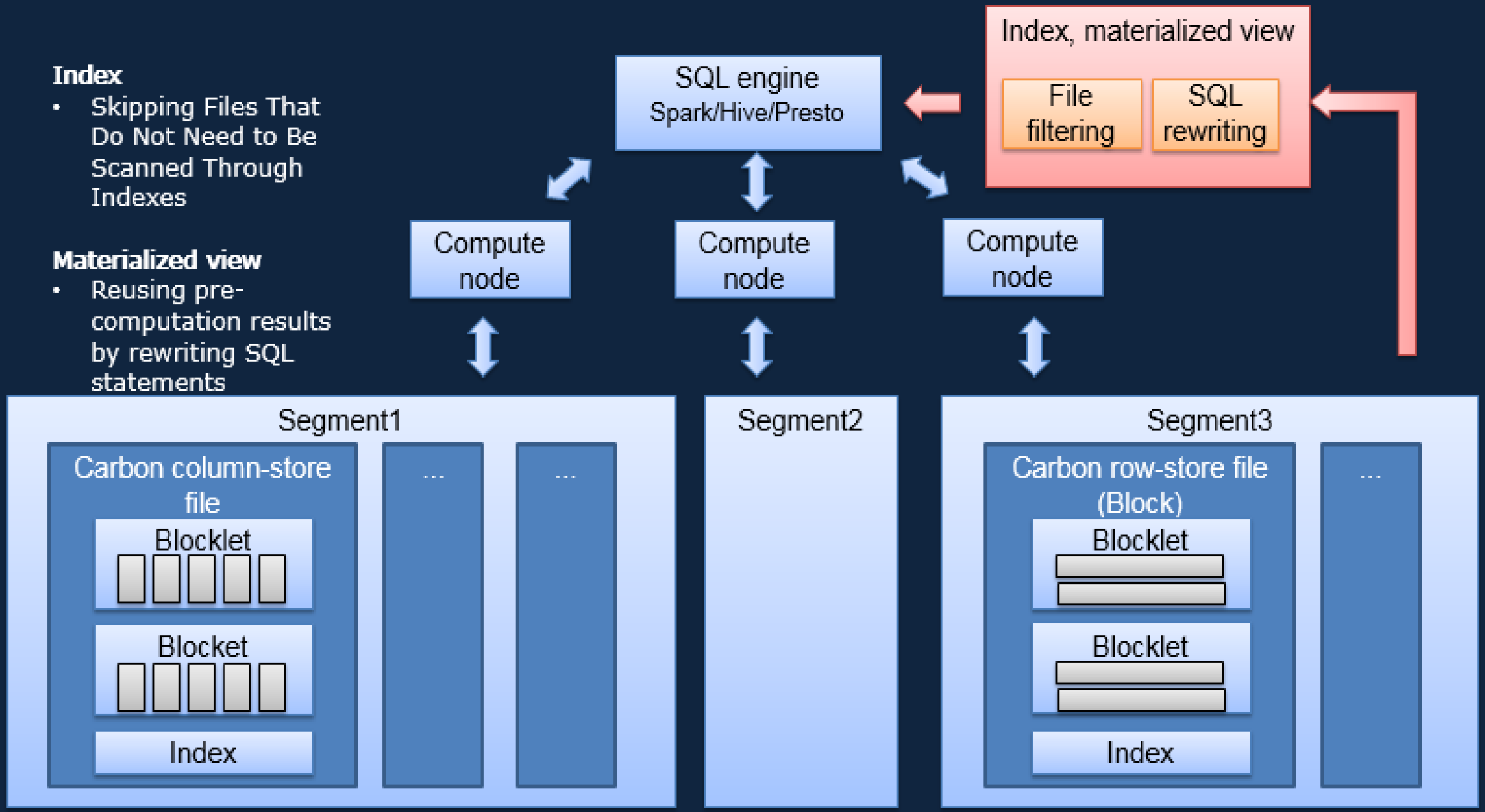
CarbonData : Query Acceleration

Index

- Skipping Files That Do Not Need to Be Scanned Through Indexes

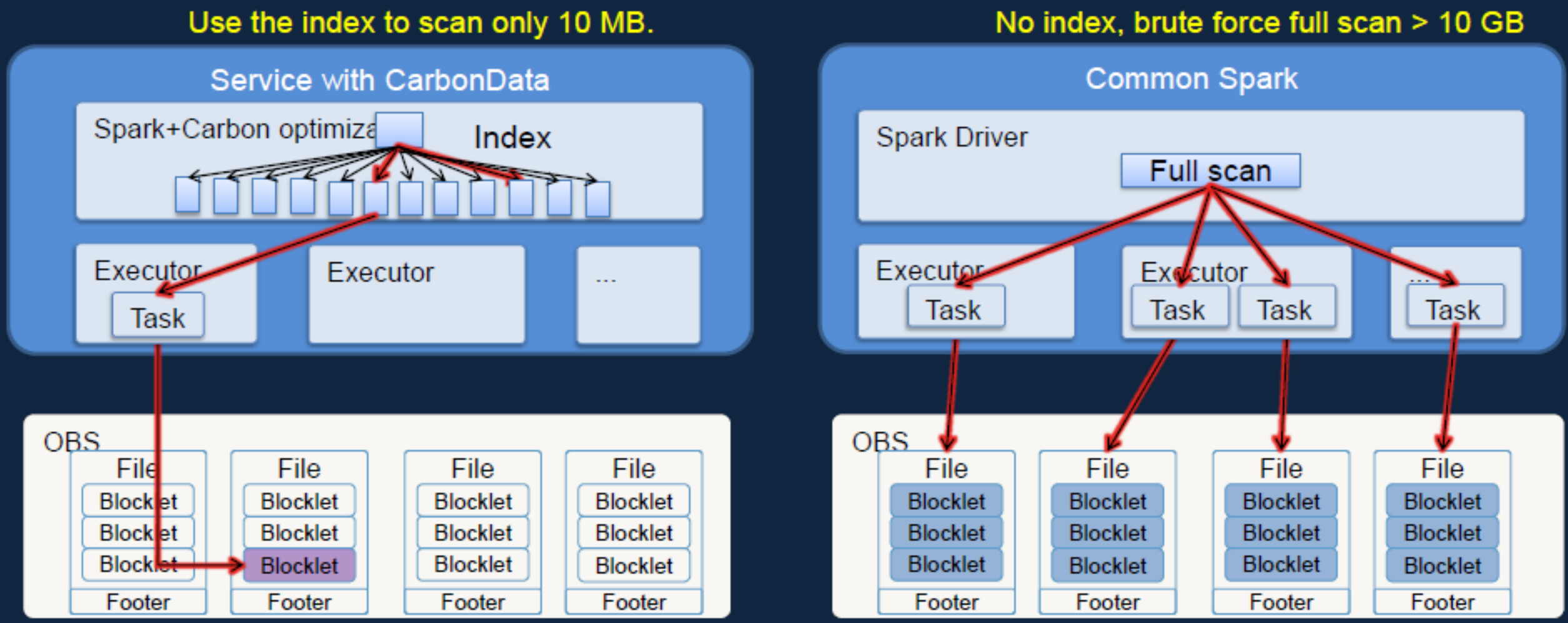
Materialized view

- Reusing pre-computation results by rewriting SQL statements



Carbon index optimization

```
SELECT city, app FROM t1 WHERE userId= '18699887362'
```



100x performance improvement in point query scenarios

Agenda

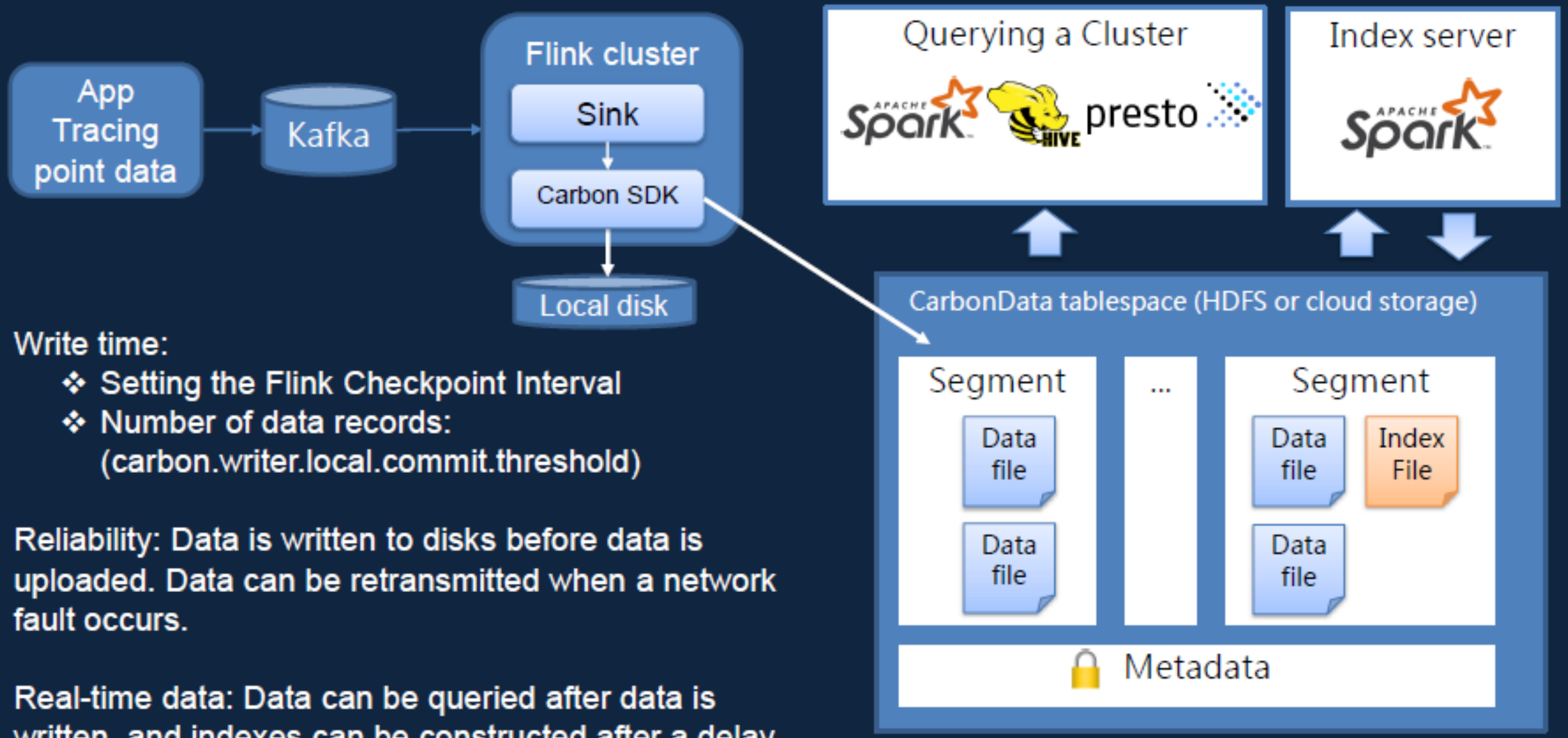
- CarbonData Background
- 2.0 New Features
  - Data access
  - Data analysis
- Upgrade Suggestion

CarbonData 2.0 New Features

- Data access:
  - Flink stream import
  - Database real-time data synchronization
  - Hive inbound and Presto inbound (2.1)
  - The Spark insert performance is doubled and the time is reduced by half.
  - The CSV, TXT, JSON, Parquet, ORC, CarbonFile format is supported in a table.

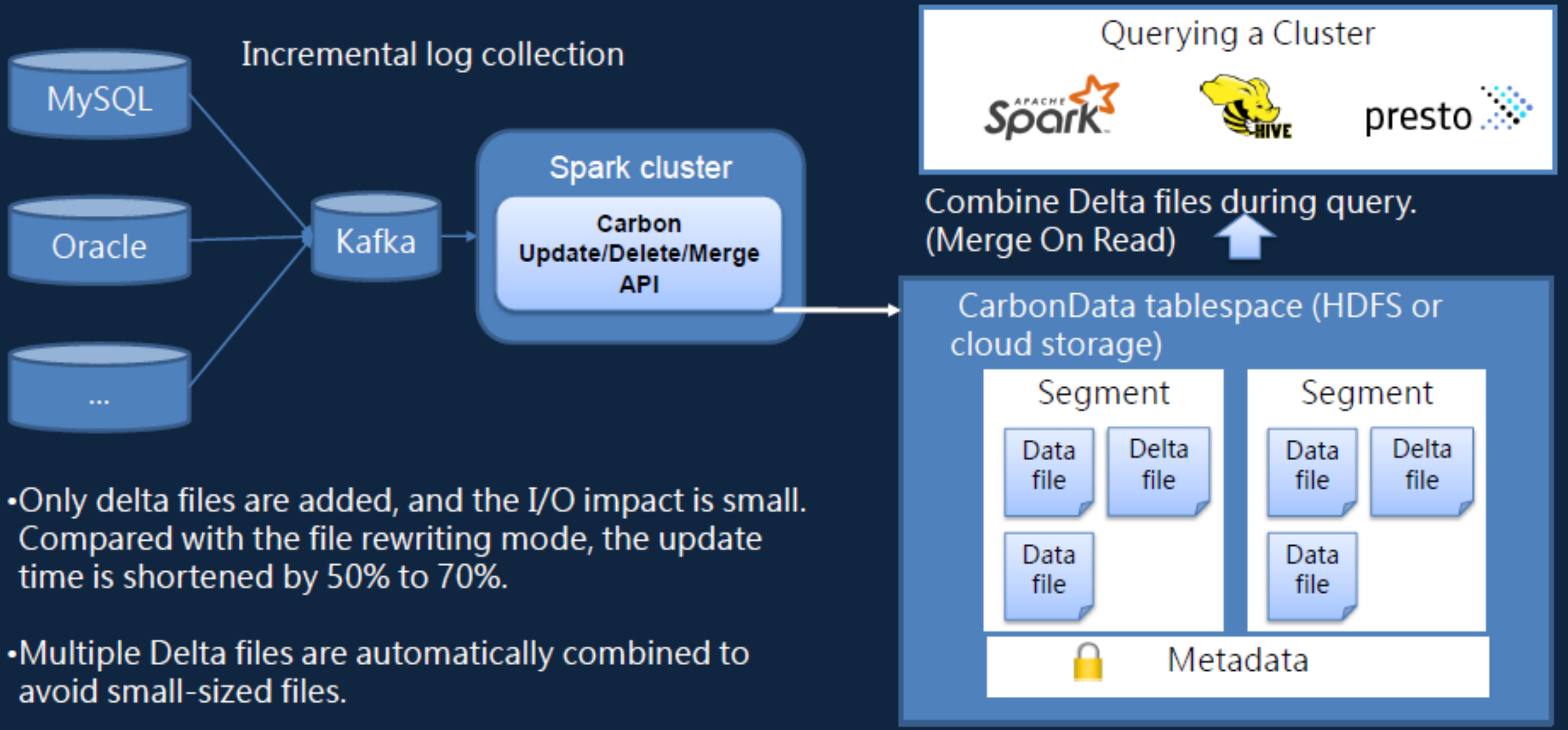
- Data query:
  - Spark Extension
  - Unify the index syntax, add the index server, SI index, and Geo index.
  - Unify the MV syntax, support time series data, and support Parquet/ORC tables.
  - Supports unstructured data, interconnection with TensorFlow, and pytorch deep neural network model training.

Flink + CarbonData real-time stream 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.

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.

Merge API Example

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()
```

Importing Hive and Presto Data to Carbon Tables

- Write Hive data to Carbon tables:
  - Non-transactional tables (similar to common Hive tables) do not support ACIDs.

```
CREATE TABLE hive_table (...)
STORED BY' org.apache.carbondata.hive.CarbonStorageHandler'

INSERT INTO hive_table SELECT * FROM source
SELECT * FROM hive_table
```

- Write Presto into Carbon Table:
  - Under development (planned version 2.1 : Oct-30)
  - Support transaction tables and non-transaction tables

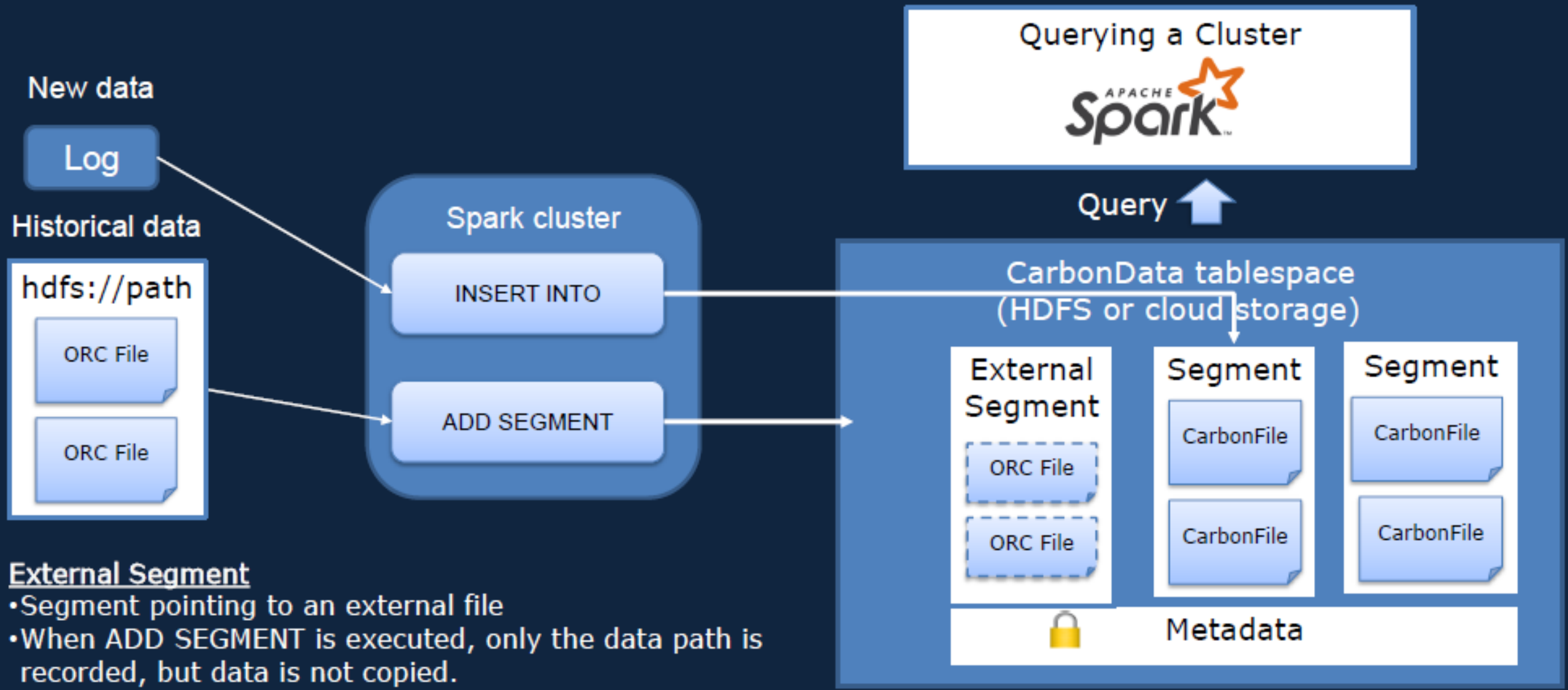
2X performance improvement of the Spark Insert

- Performance improvement points:
  - Avoid the conversion from Spark InternalRow to Carbon Row.
  - Avoid multiple data conversions during bad record processing.
  - Avoid adjusting the column order during index building.

- Types of Carbon tables supported.
  - Indexed, no index
  - Partitioned, no partition
  - transaction table
  - Importing MVs
  - Importing Flink Stream Data to the Database

Scenario (15 GB)	CarbonData 1.6	CarbonData 2.0
Inserting data from the Parquet table to the Carbon table	800 seconds	420 seconds

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)

CarbonData 2.0 new Features

- Data access:
  - Flink stream import
  - Database real-time data synchronization
  - Hive inbound and Presto inbound (2.1)
  - The Spark insert performance is doubled and the time is reduced by half.
  - The CSV, TXT, JSON, Parquet, ORC, CarbonFile format is supported in a table.

- Data query:
  - Spark Extension
  - Unify the index syntax, add the index server, SI index, and Geo index.
  - Unify the MV syntax, support time series data, and support Parquet/ORC tables.
  - Supports unstructured data, interconnection with TensorFlow, and pytorch deep neural network model training.

Spark Extension

- Standard extension mode of the Spark community

```
// CarbonData 1.x
import org.apache.spark.sql.CarbonSession._
val spark = SparkSession
.builder()
.master(masterUrl)
.enableHiveSupport()
.getOrCreateCarbonSession()
```

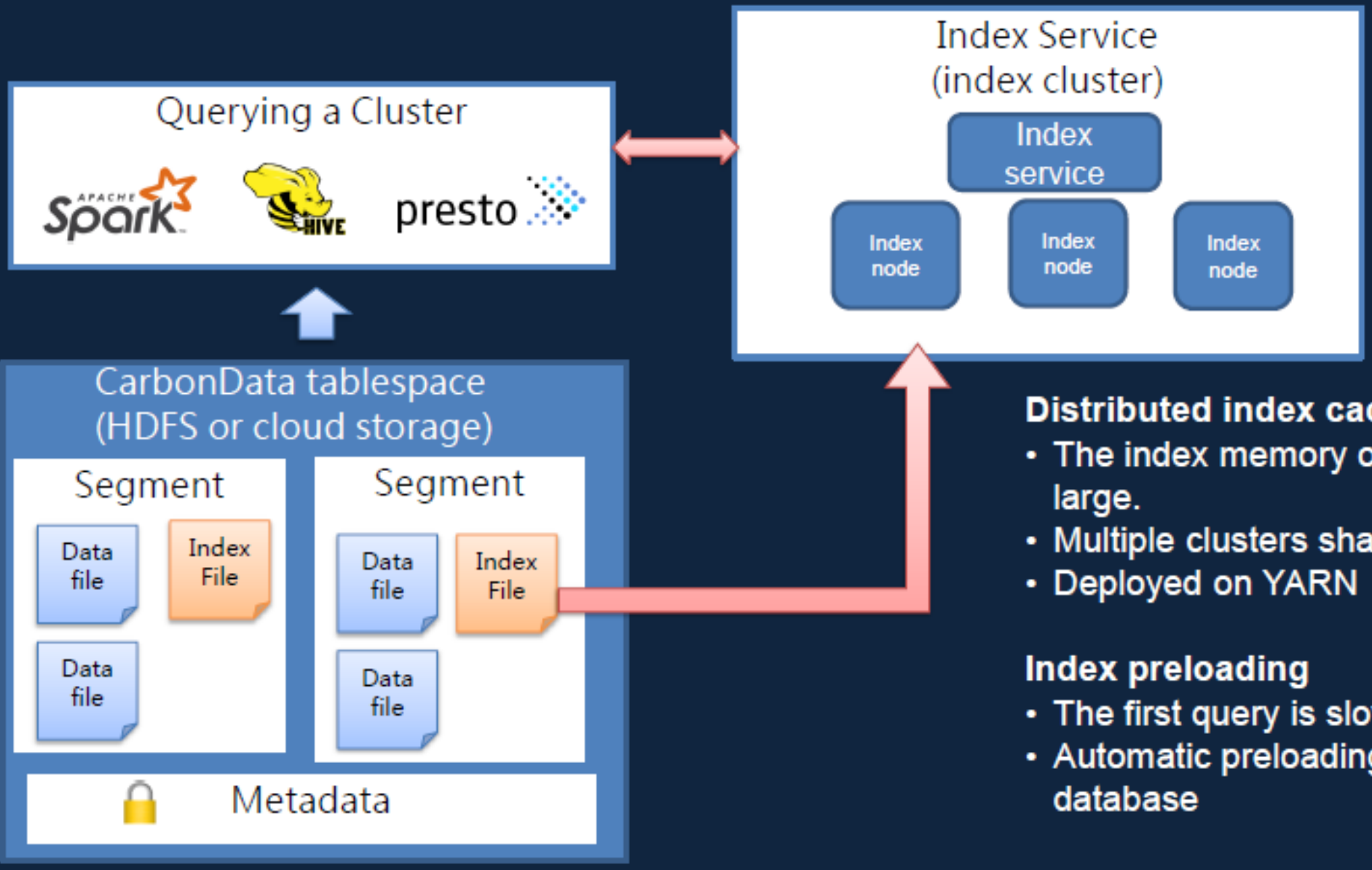
```
// CarbonData 2.0
val spark = SparkSession
.builder()
.master(masterUrl)
.enableHiveSupport()
 .config("spark.sql.extensions", "org.apache.spark.sql.CarbonExtensions")
.getOrCreate ()
```

All CarbonSession features are supported.

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

CarbonSession is still supported in 2.0, but it is recommended to not use it (to be discarded in the future).

Index Service



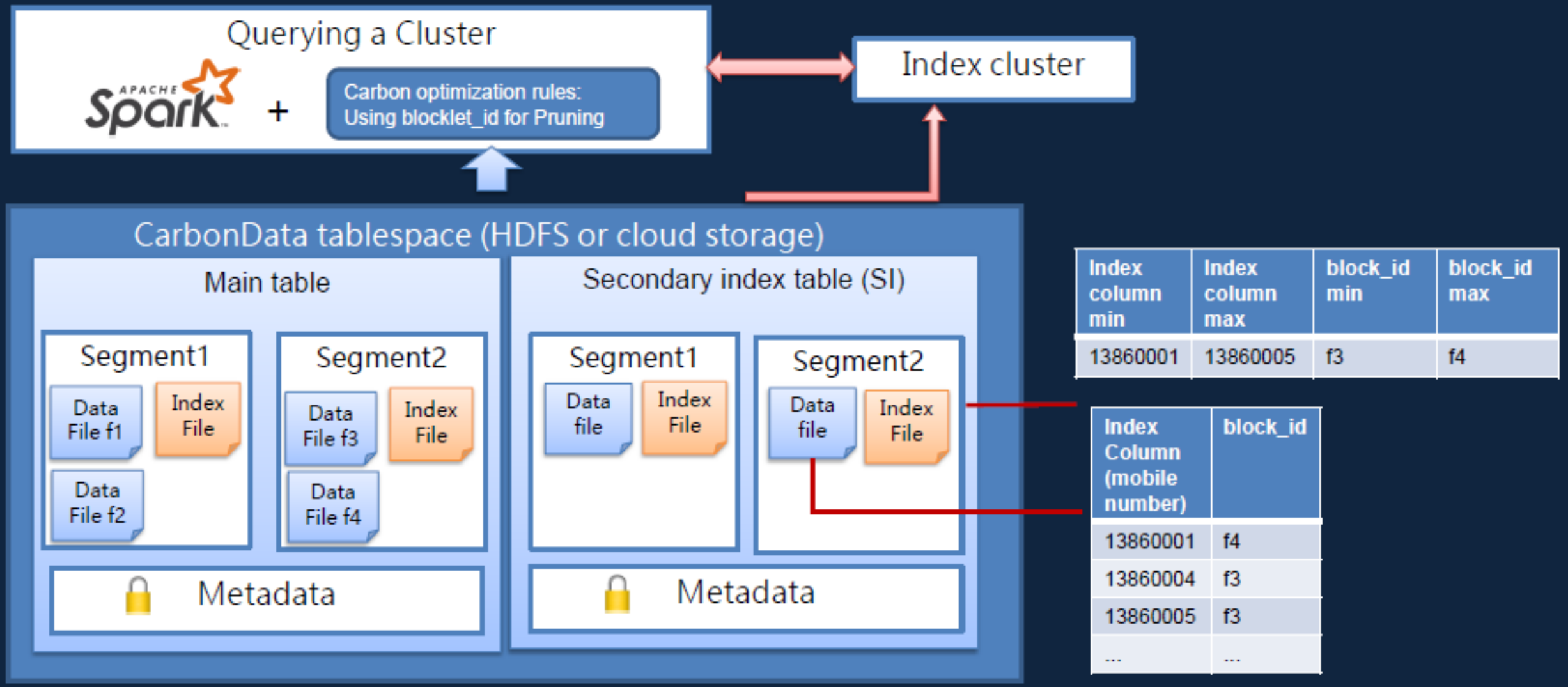
Distributed index cache

- The index memory on the driver side is too large.
- Multiple clusters share one index.
- Deployed on YARN

Index preloading

- The first query is slow.
- Automatic preloading after data is saved to the database

Secondary index



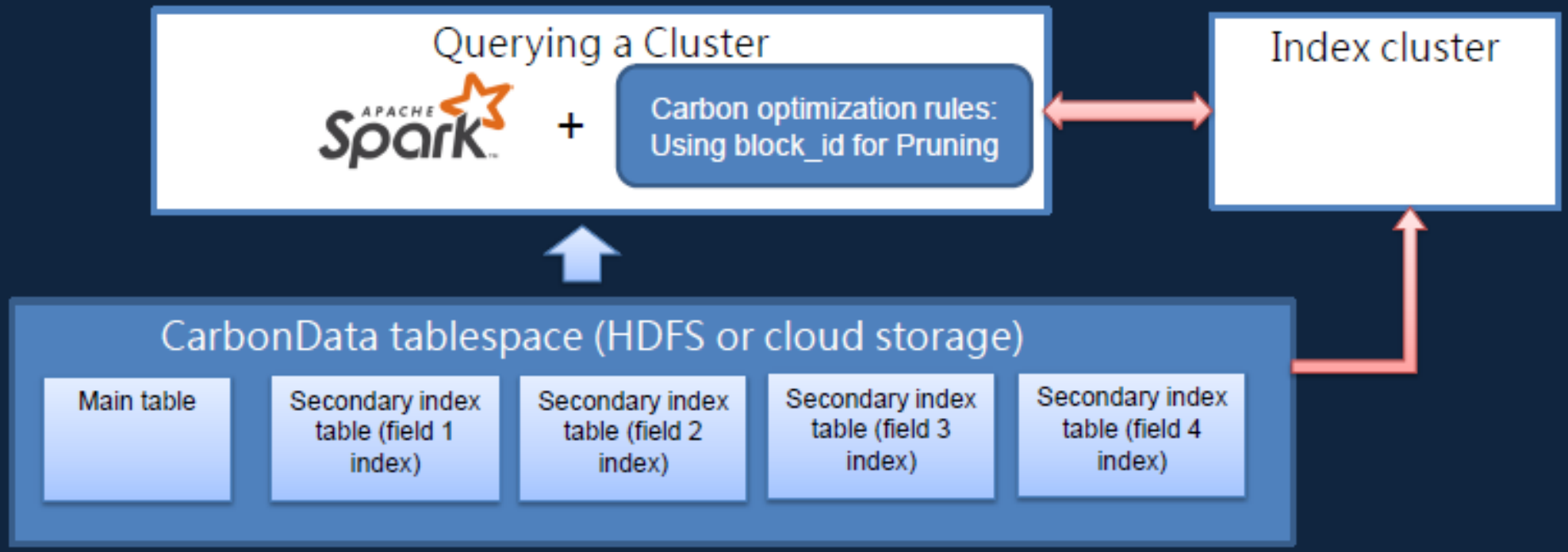
Index column min	Index column max	block_id min	block_id max
13860001	13860005	f3	f4

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.



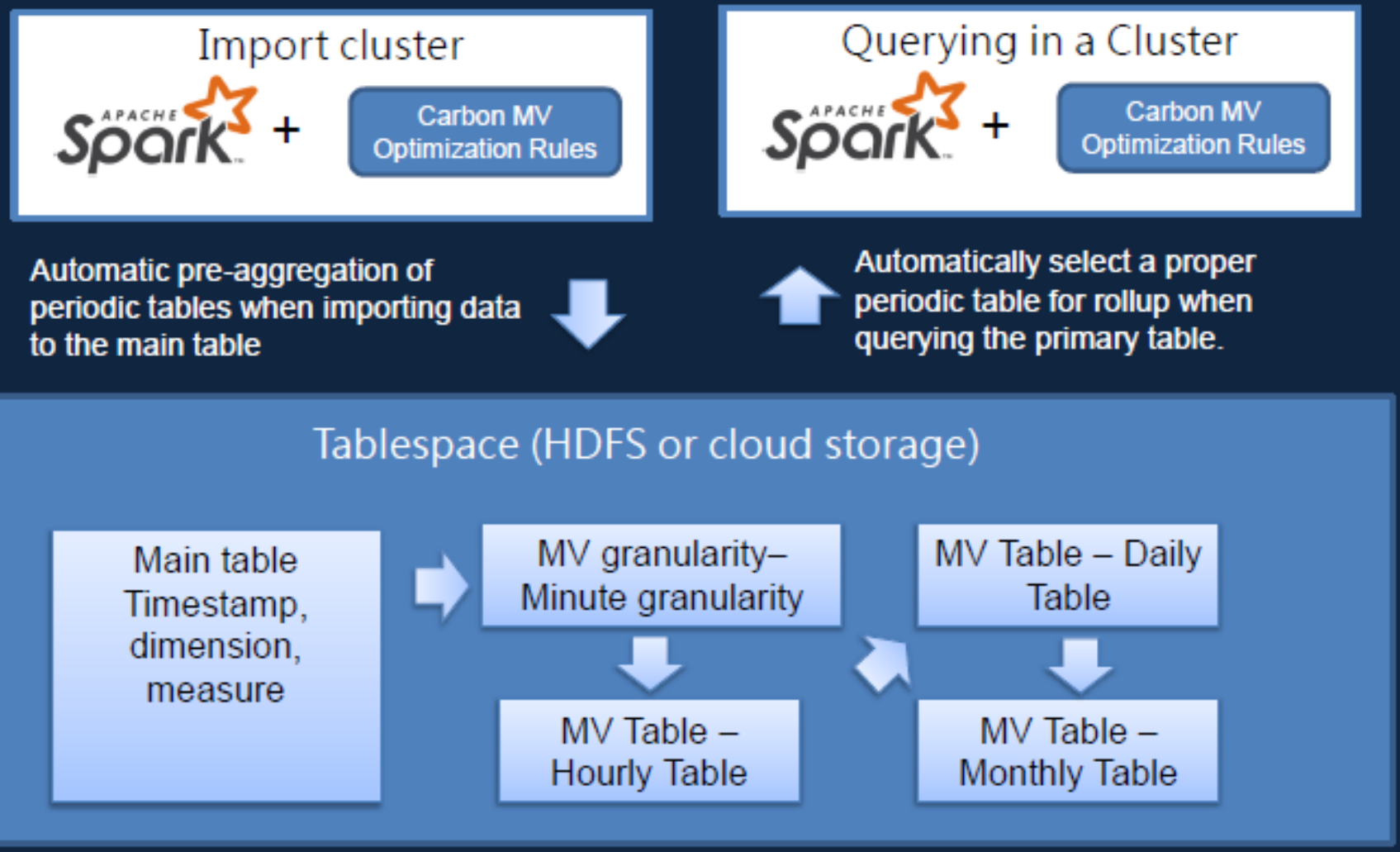
The index syntax is consistent with that of Hive

```
// Create an index.
CREATE INDEX [IF NOT EXISTS] index_name
ON TABLE table_name (column_name, ...)
AS index_provider
[WITH DEFERRED REFRESH]
[PROPERTIES('key'='value')]

index_provide := bloomfilter | lucene | carbondata

// Display the index.
SHOW INDEXES on table_name

// Delete an index.
DROP INDEX [IF EXISTS] index_name on table_name

// Refresh the index (by segment).
REFRESH INDEX index_name
ON table_ame
[WHERE SEGMENT.ID IN (segment_id, ...)]
```

The MV syntax is consistent with that of Hive

```
// Create a materialized view.
CREATE MATERIALIZED VIEW [IF NOT EXISTS] mv_name
[WITH DEFERRED REFRESH]
AS select_statement

// Example
CREATE MATERIALIZED VIEW mv1
AS select a.city, max(b.gdp) from a join b on a.id = b.id group by a.city

// Display the materialized view.
SHOW MATERIALIZED VIEW

// Delete a materialized view.
DROP MATERIALIZED VIEW [IF EXISTS] mv_name

// Refresh the index. (The system automatically determines the segment to be refreshed and performs incremental update.)
REFRESH MATERIALIZED VIEW mv_name
```

Time series supported by MV



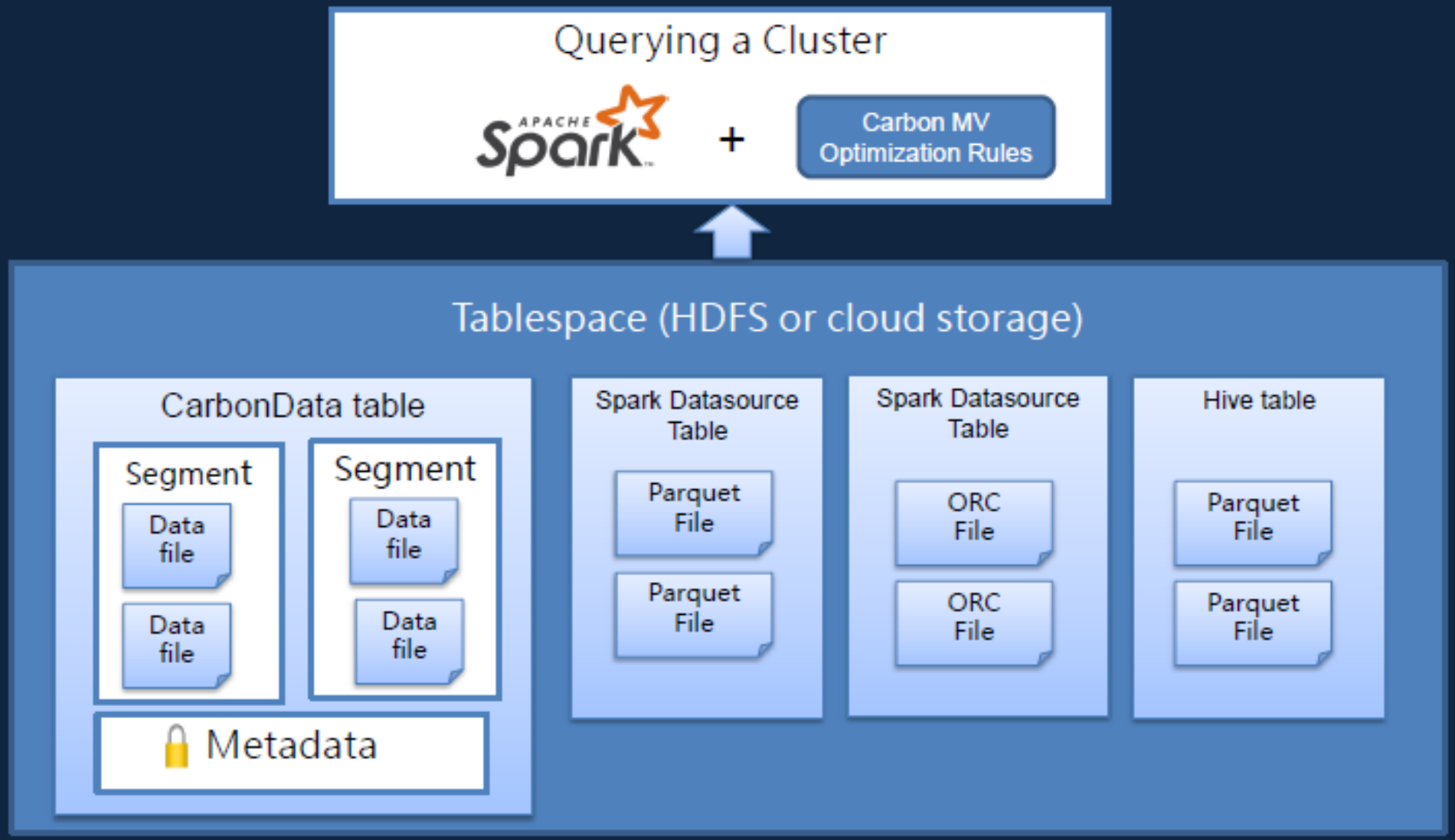
Time series MV example

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')
```

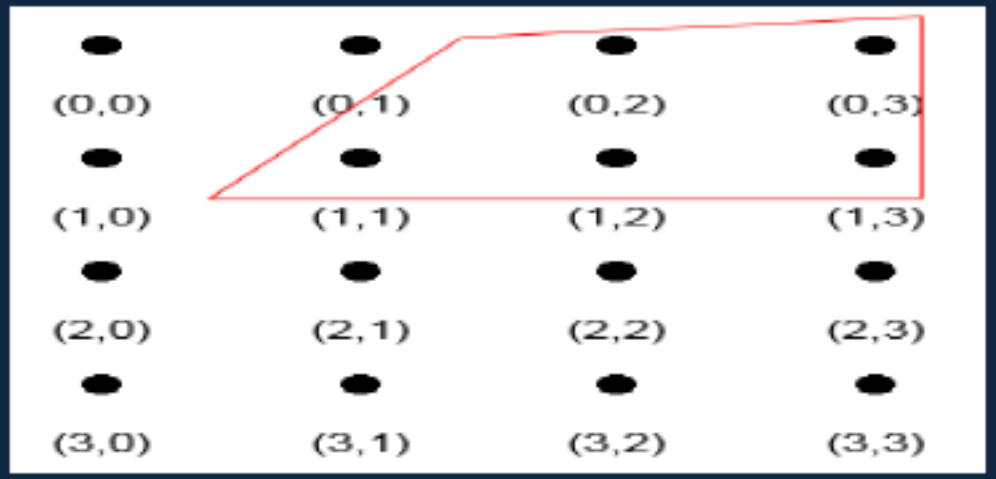
Restrictions: The time series MV does not support join statements and is replaced by common MVs.

MV supports non-Carbon tables



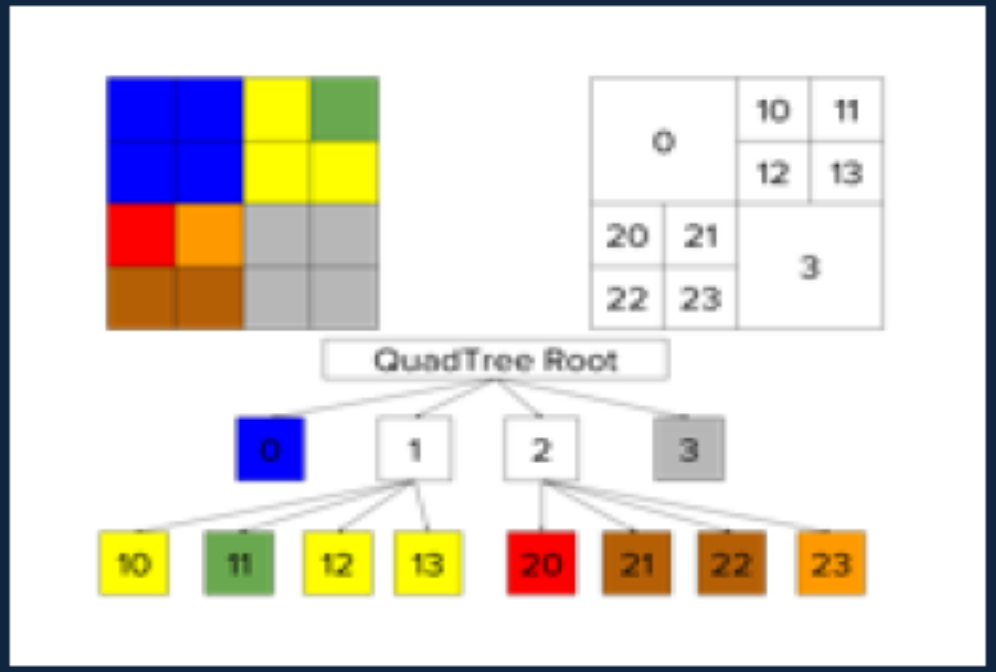
In addition to speeding up Carbon tables, you can also speed up Parquet, ORC tables.
Restrictions: There is no segment concept. Only full MV update is supported. Incremental update is not supported.

Geo spatial support



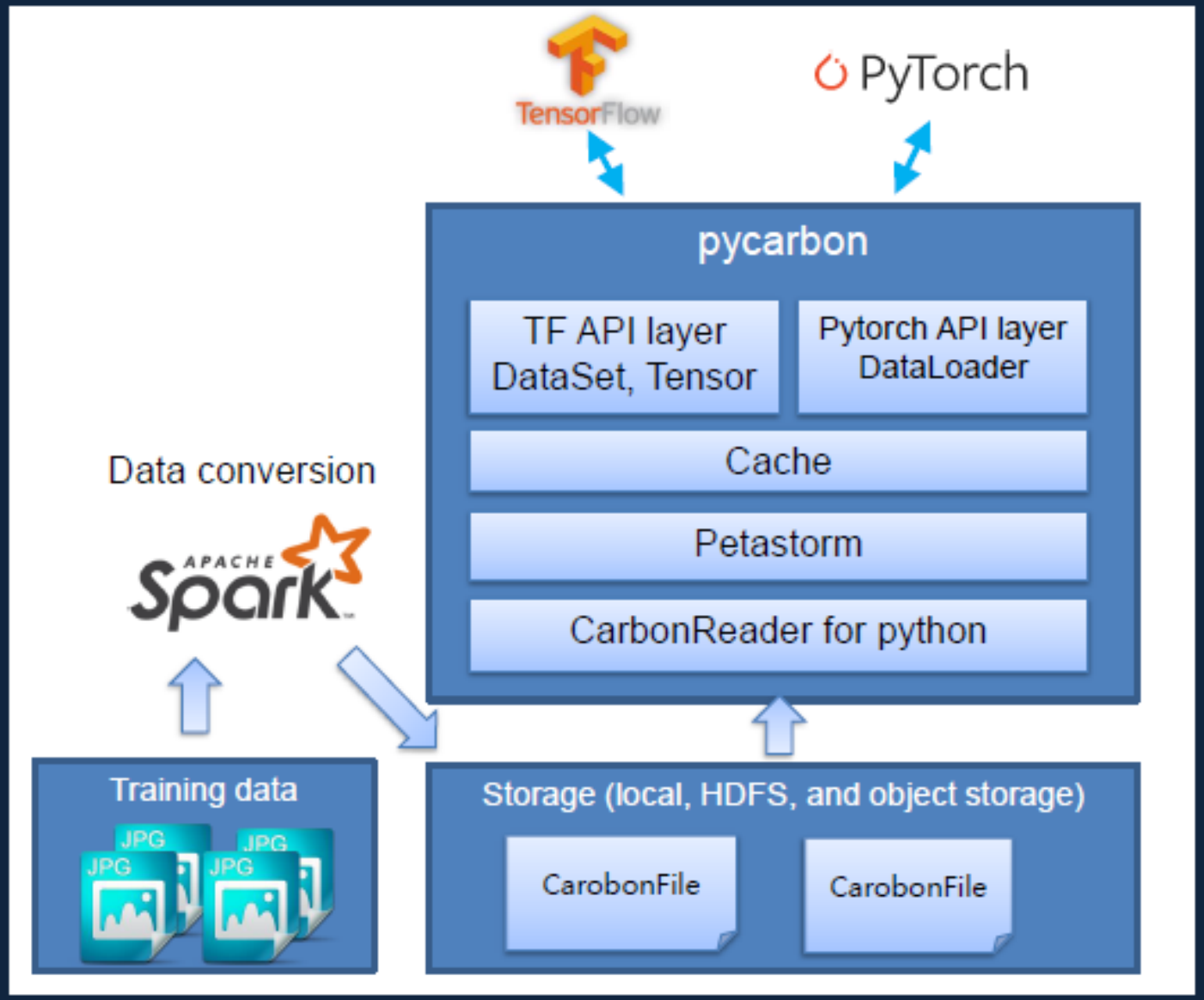
Records to hit with query are highlighted:

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

Accelerated AI model training

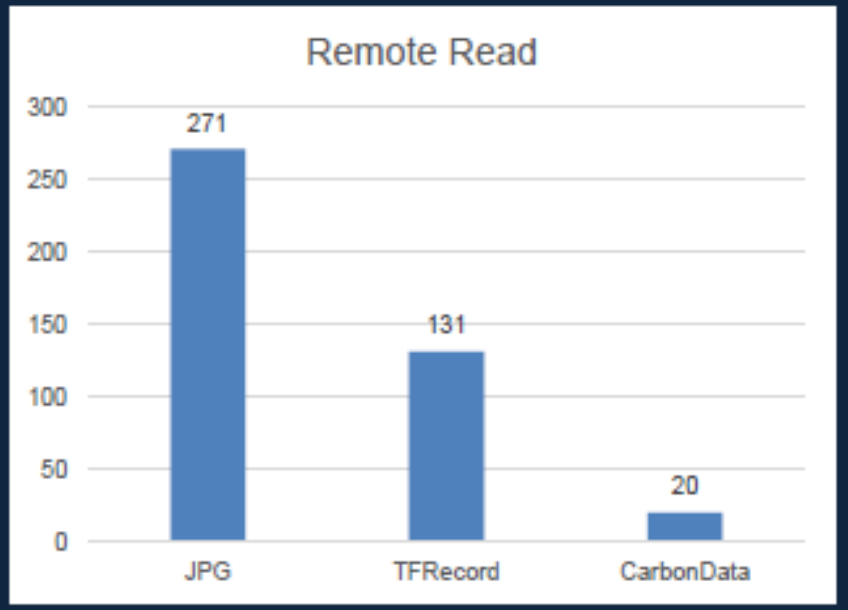


Facilitates model training by using the pycarbon + AI framework.

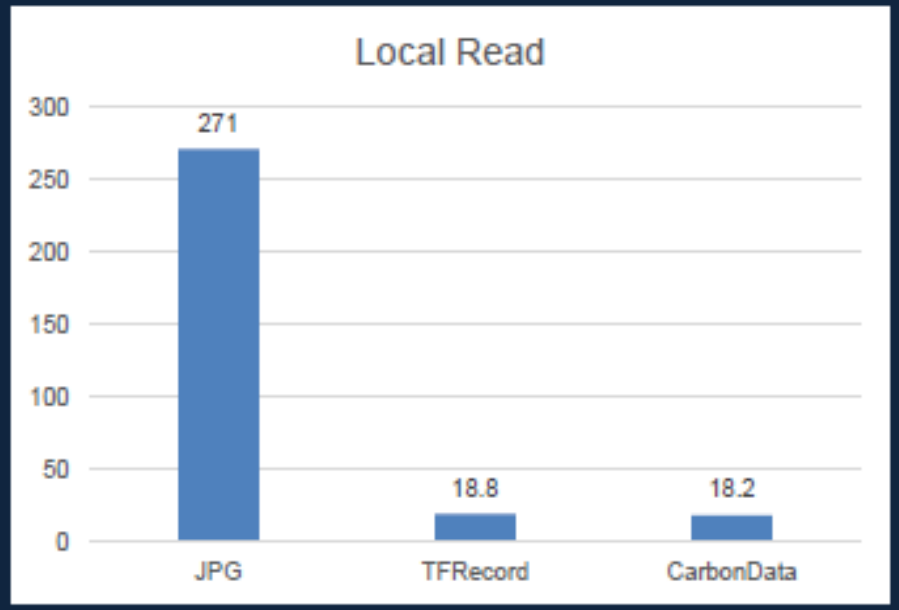
- After images are converted to Carbon files, the files are merged, which greatly improves the I/O efficiency.
- Cache: caches memory or local disks to avoid multiple remote read operations during training.
- Parallel processing: supports multi-thread parallel read.
- Out-of-order read: The read sequence 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

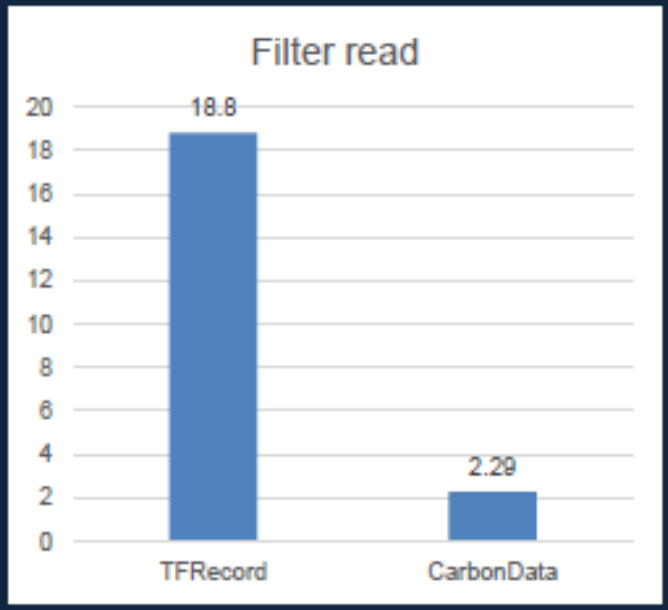
Dataset: 7800 images (1 GB) are extracted from ImageNet. Converted to Carbon and TFRecord files.
Field: 7 Columns: height, width, depth, imageName, imageBinary, txtName, txtContent
Storage: cloud storage (object storage)



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.

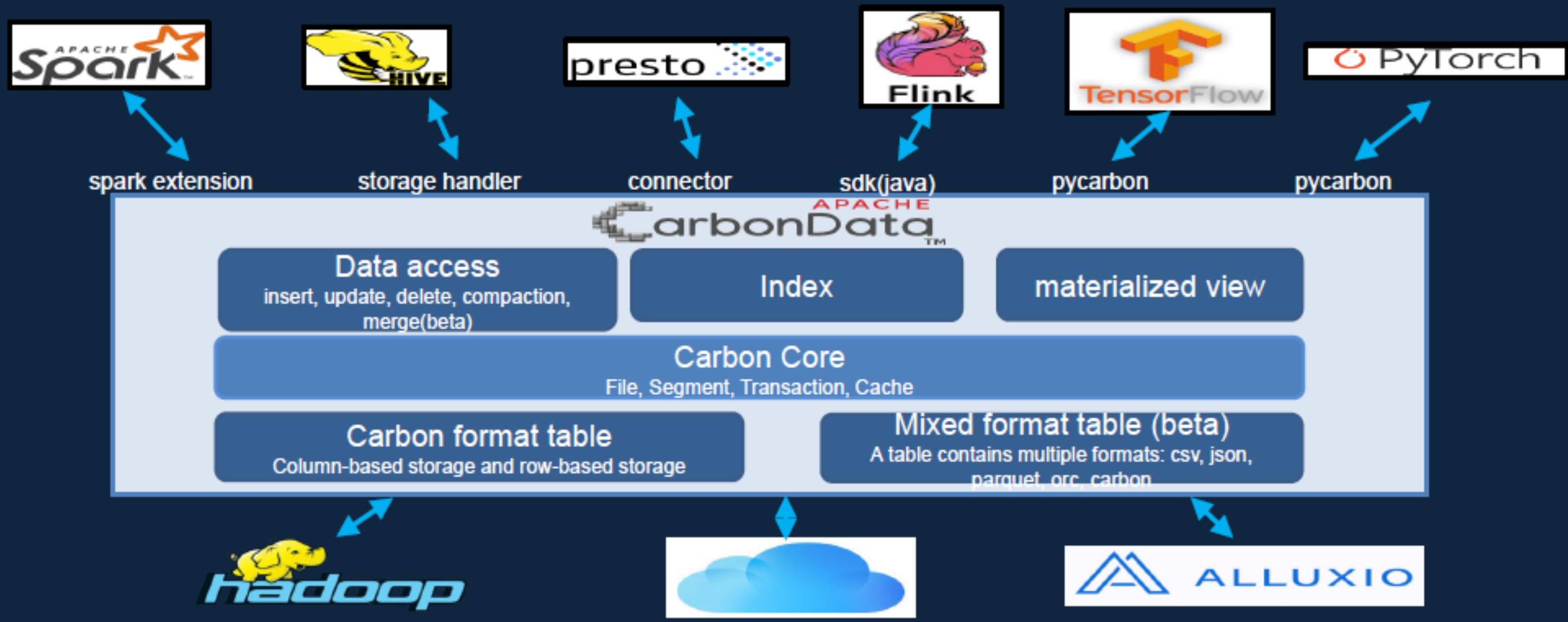


1300 images are filtered out from 7800 images as the training set. The I/O and time are six times shorter.

Upgrade Suggestion to CarbonData 2.x

- CarbonData 2.0.1 of the latest version is recommended.
- Only Spark 2.3 and Spark 2.4 are supported. Versions earlier than Spark 2.3 are not supported.
- Global dictionaries are no longer supported.
  Migration solution:
  - Recreate tables in the old system and use them in the new system.
- Pre-Aggregate DataMap is no longer supported.
  Migration solution:
  - Delete the DataMap from the old system.
  - Recreate indexes using the Index or MV syntax in the new system.
- Batch sorting is no longer supported.
  Migration solution:
  - In the old system, run the ALTER TABLE command to change SORT_SCOPE to NO_SORT, LOCAL_SORT, or GLOBAL_SORT.
- You are advised to use CarbonData in Spark Extension mode. CarbonSession will not be supported in the future.
- You are advised to set the data warehouse storage location using Spark/Hive. The carbon.storelocation attribute will no longer be supported in the future.
  - spark.sql.warehouse.dir
  - hive.metastore.warehouse.dir

Summary:



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

Love more community involvement & feedback

- 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