CarbonData: 大数据交互式分析实践

李昆 2017-05

Agenda

- 为什么需要CarbonData
- CarbonData介绍
- 性能测试
- 应用案例
- 未来计划

企业中包含多种数据应用，从商业智能、批处理到机器学习

Report & Dashboard

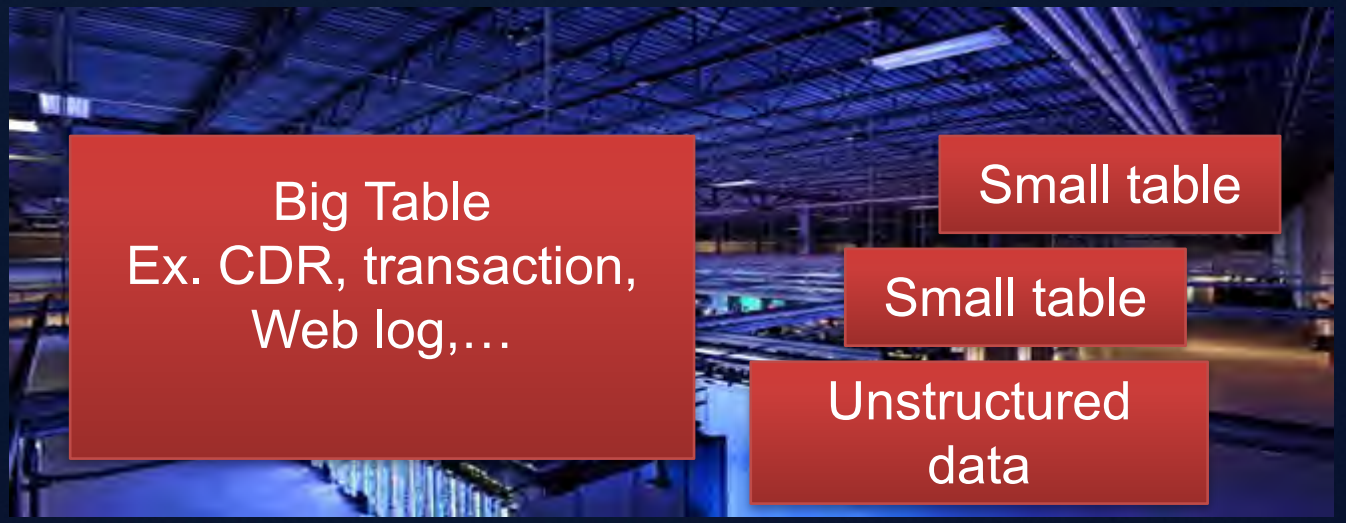
OLAP & Ad-hoc

Batch processing

Machine learning

Realtime Analytics

data



来自数据的挑战

- Data Size
  - Single Table >10 B
  - Fast growing

百亿级数据量

- Multi-dimensional
  - Every record > 100 dimension
  - Add new dimension occasionally

多维度

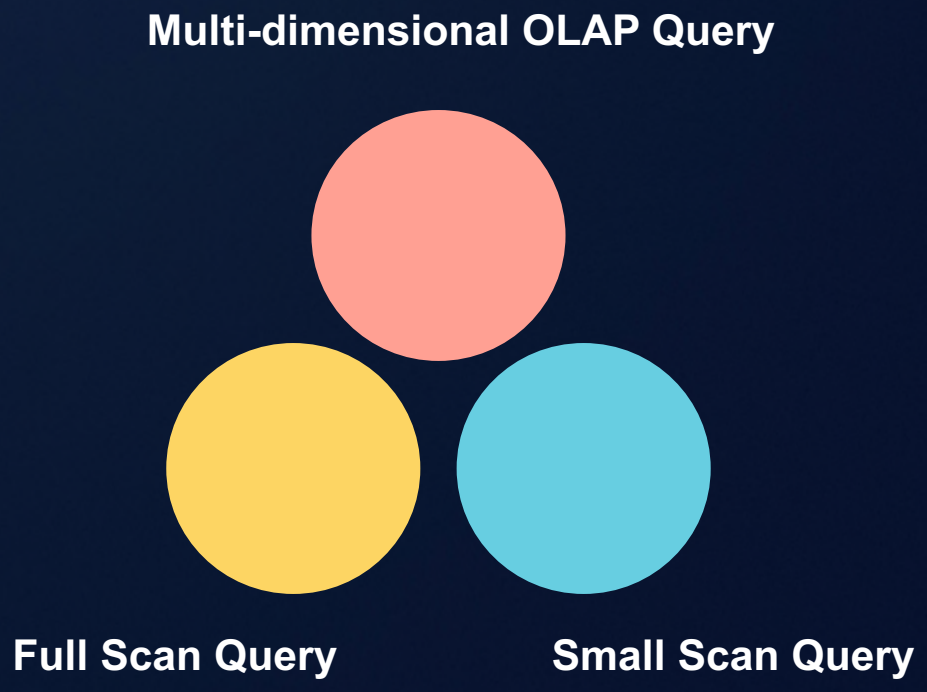
- Rich of Detail
  - Billion level high cardinality
  - 1B terminal * 200K cell * 1440 minutes = 28800 (万亿)

细粒度

来自应用的挑战

- Enterprise Integration **企业应用集成**
  - SQL 2003 Standard Syntax
  - BI integration, JDBC/ODBC

- Flexible Query **灵活查询 无固定模式**
  - Any combination of dimensions
  - OLAP Vs Detail Record
  - Full scan Vs Small scan
  - Precise search & Fuzzy search



How to choose storage?

如何构建数据平台？

选择1: NoSQL Database

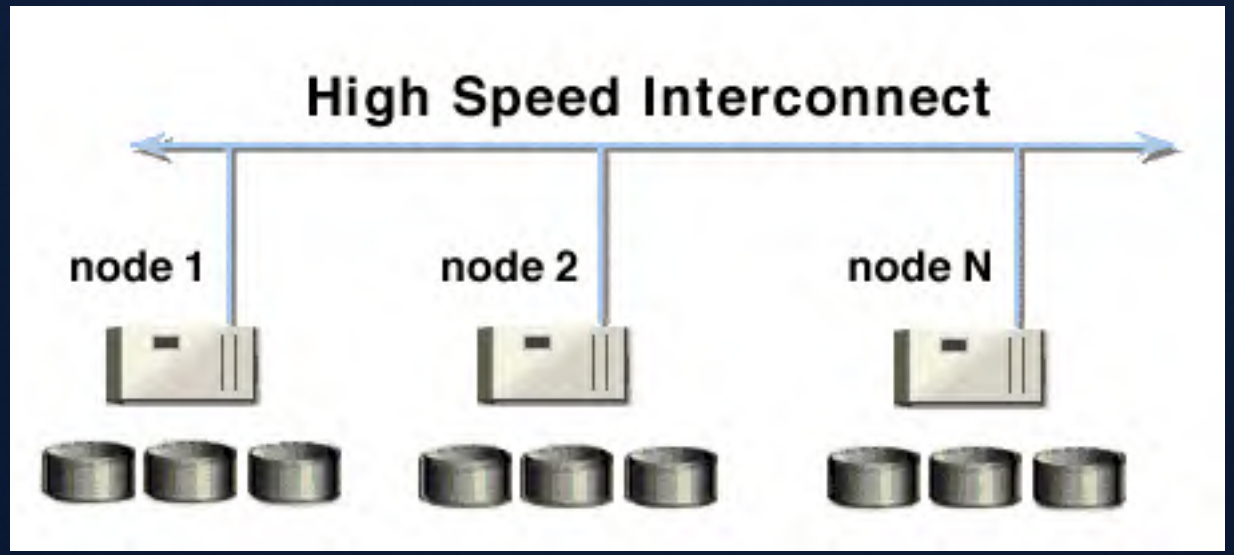
Key-Value store: low latency, <5ms

只能通过Key访问，一键一值
适合实时应用对接，不适合分析型应用

Type	Examples
Key-Value Store	redis riak
Wide Column Store	APACHE HBASE Cassandra

Row Key	Timestamp	Customer		Sales	
Customer Id		Name	City	Product	Amount
101	T1	Suresh	Hyderabad		300
101	T2	Suresh Reddy		Books	
102	T1	Lavya Gavshinde	Indore	Fan	600
102	T2	Lavya			570
102	T3		Bhopal		
103	T1	Anurag	Raipur	Laptop	40000
104	T1	Deepesh	Delhi	Bike	32000

选择2：Parallel database



- Parallel scan + Fast compute

细粒度控制并行计算，适合中小规模数据分析（数据集市）

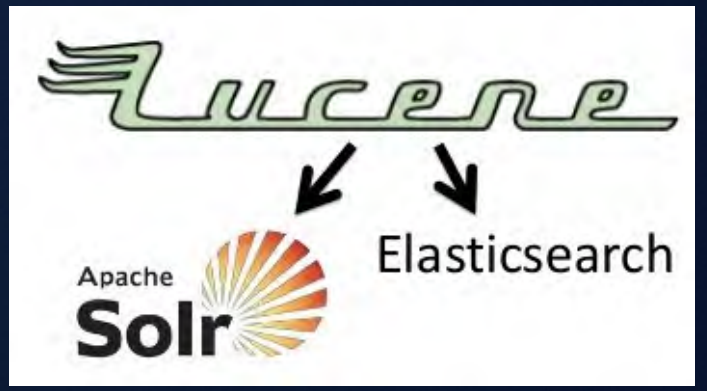
- Questionable scalability and fault-tolerance
  - Cluster size < 100 data node
  - Not suitable for big batch job

扩展能力有上限
查询内容错能力弱
不适合海量数据分析（企业级数仓）

选择3: Search engine

- All column indexed
- Fast searching
- Simple aggregation

适合多条件过滤，文本分析



- Designed for search but not OLAP
- Not for TopN, join, multi-level aggregation

无法完成复杂计算

- 3~4X data expansion in size

数据膨胀

- No SQL support

专用语法，难以迁移

选择4: SQL on Hadoop

- Modern distributed architecture, scale well in computation.
  - Pipeline based: Impala, Drill, Flink, …
  - BSP based: Hive, SparkSQL

并行扫描+并行计算
适合海量数据计算

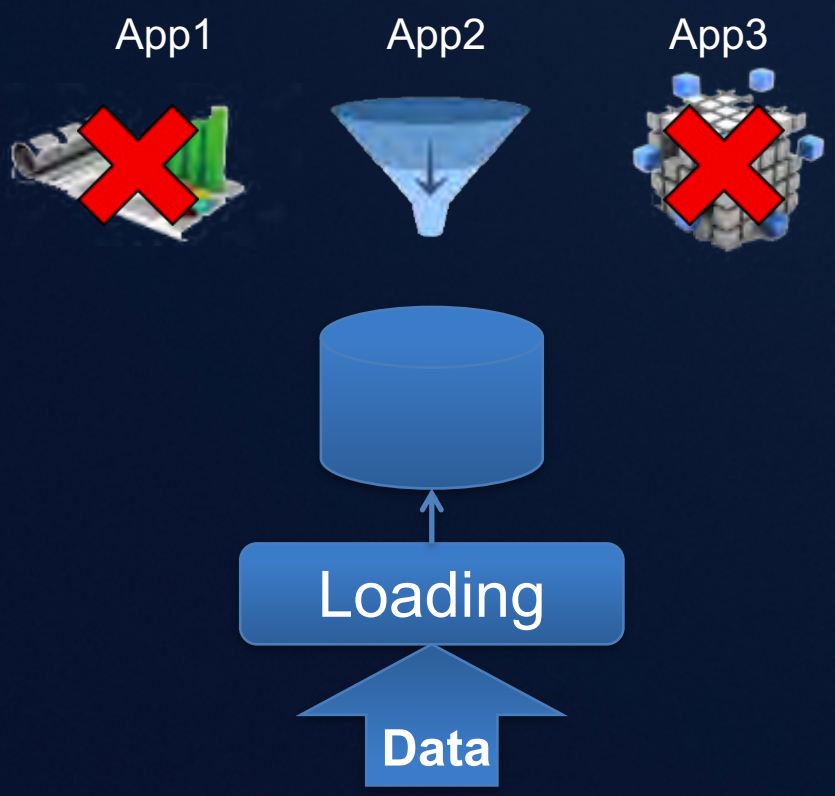
- BUT, still using file format designed for batch job
  - Focus on scan only
  - No index support, not suitable for point or small scan queries

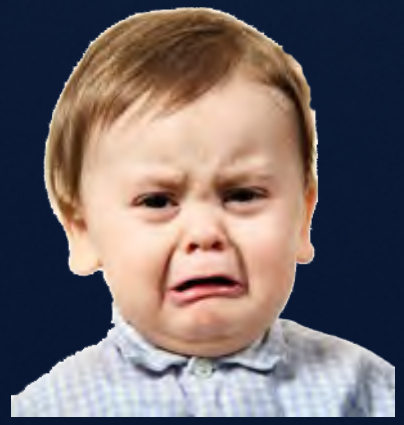
仍然使用为批处理设计的存储，场景受限

架构师如何选择？

Choice 1: Compromising

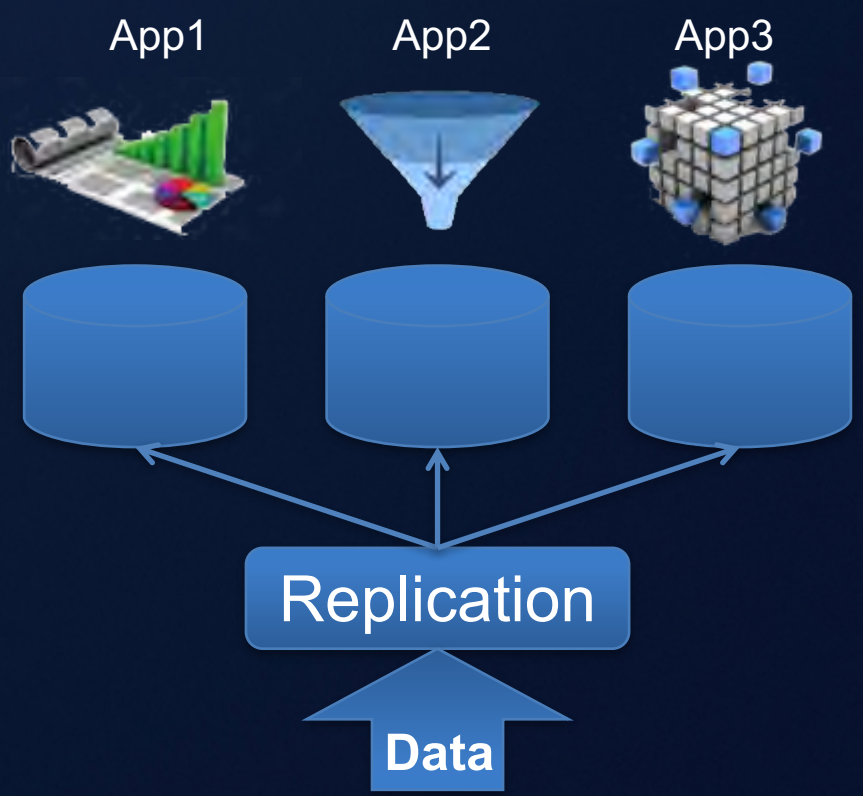
做出妥协，只满足部分应用





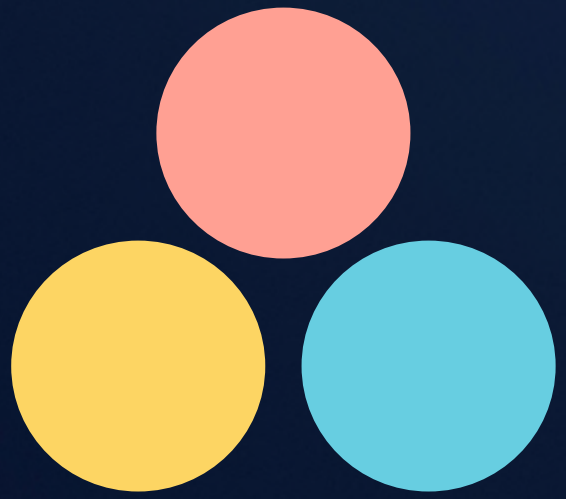
Choice 2: Replicating of data

复制多份数据，满足所有应用



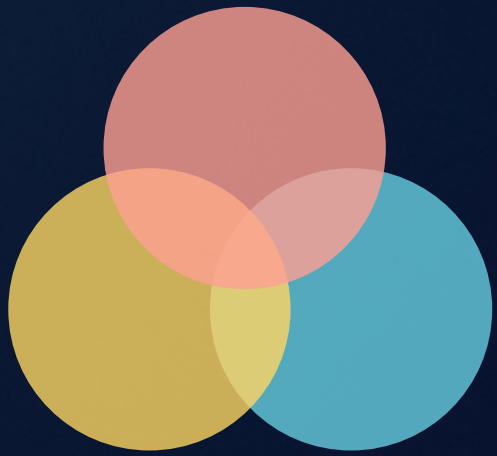
CarbonData目标：
一份数据满足多种业务需求，与大数据生态无缝集成

Multi-dimensional OLAP Query



Full Scan Query　　Small Scan Query

CarbonData: Unified Storage



一份数据满足多种分析场景
详单过滤，海量数仓，数据集市，…

Apache CarbonData社区介绍

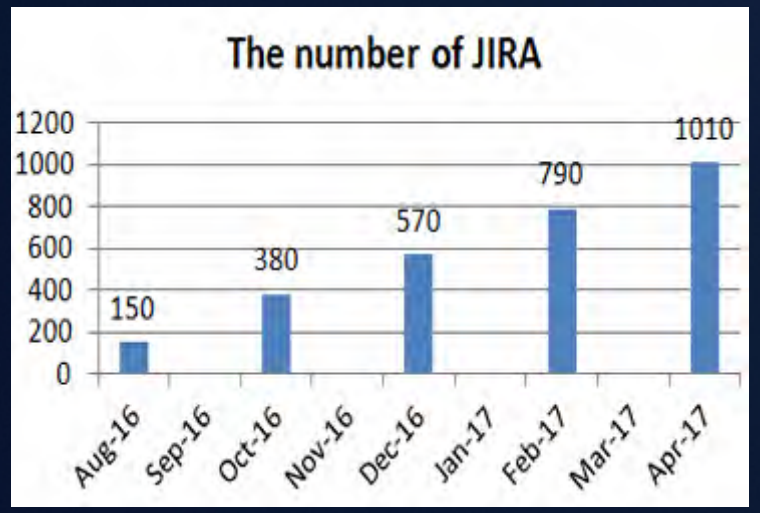
Apache CarbonData

- 2016年6月，进入Apache孵化器
- 2016年9月，第一个生产系统部署
- 2017年4月，Apache毕业，成为Apache顶级项目

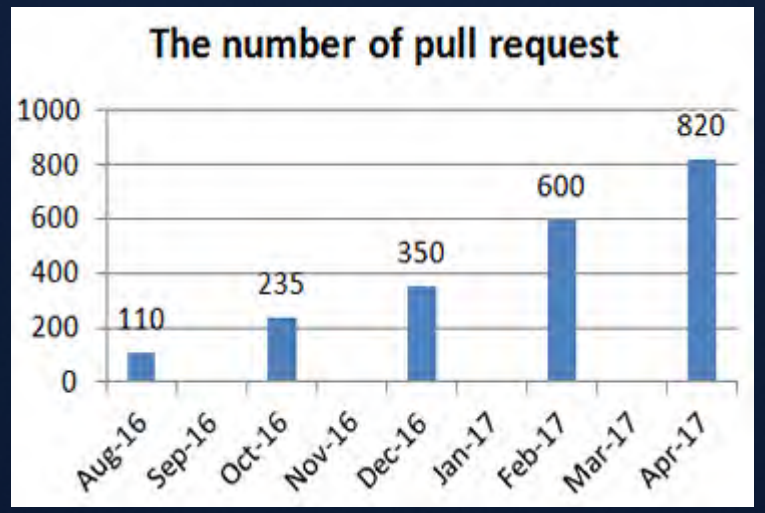
http://carbondata.apache.org

- 共发布：
  - 5个稳定版本
  - 最新版本： **1.1.0**
- 目前可访问CarbonData的计算引擎：
  - Spark, Presto, Hive, Flink

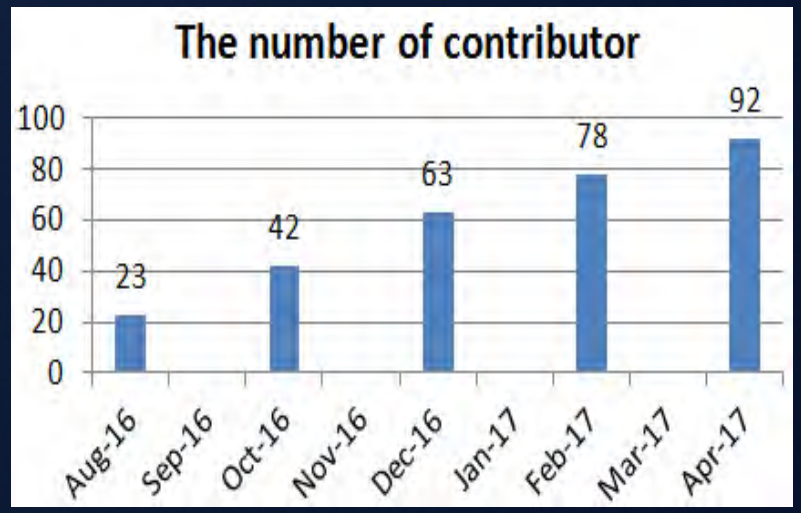
Growing Community



每个月约100个JIRA



每个月约100个PR



每两个月新增约15个贡献者

- **Diversity organization:**
- **In production:**

HUAWEI

感谢社区用户的贡献

- Partition
  - 上汽集团
- Bitmap Encoding
  - 上汽集团
- Presto Integration
  - 携程
- Hive Integration
  - 滴滴、Knoldus

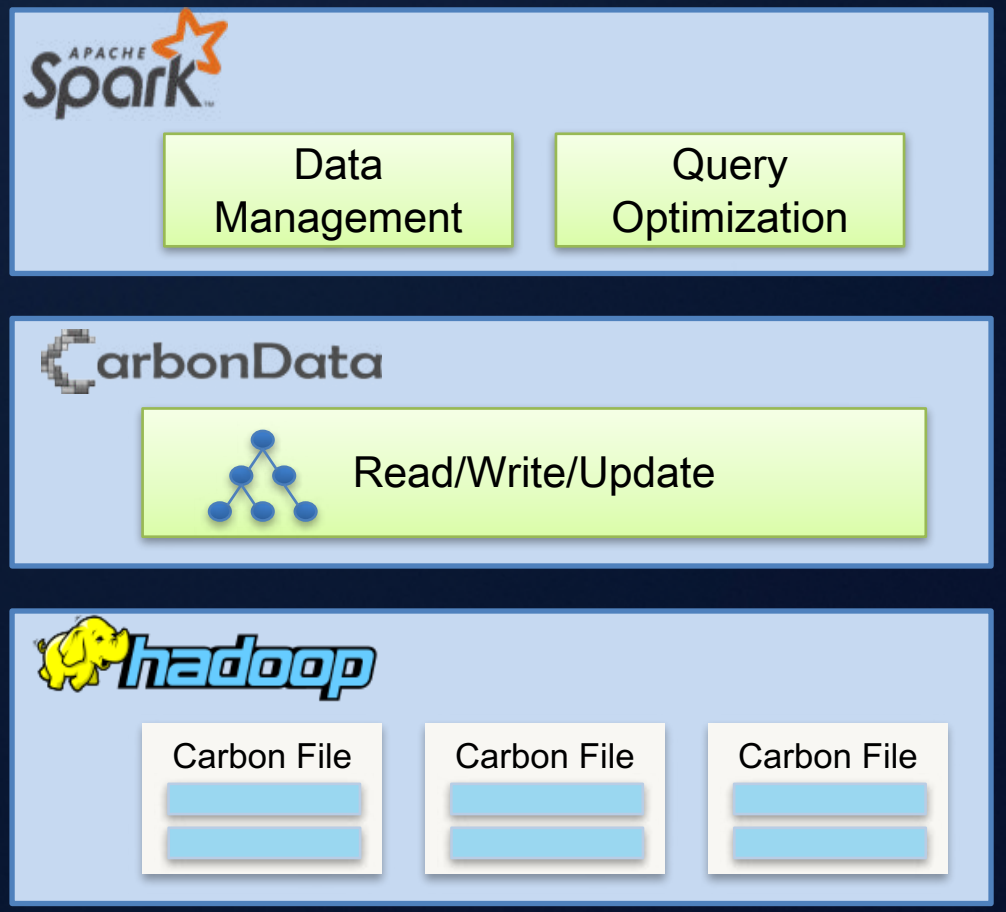
CarbonData + Spark:
打造大数据交互式分析引擎

CarbonData大数据生态

- 支持Spark、Hive、Presto、Flink
- 内置Hadoop和Spark深度优化
  - Hadoop: > 2.2
  - Spark 1.5, 1.6, 2.1

- 接口
  - SQL
  - DataFrame API

- 支持操作:
  - 查询：支持SparkSQL优化器
  - 数据管理：批量入库、更新、删除、合并（Compaction）、增删列



使用方式：入库

- SQL

```
CREATE TABLE tablename (name String, PhoneNumber String)
STORED BY “carbondata”
TBLPROPERTIES (...)

LOAD DATA [LOCAL] INPATH 'folder path' [OVERWRITE] INTO TABLE tablename
OPTIONS(...)

INSERT INTO TABLE tablennme  select_statement1 FROM table1;
```

- Dataframe

```
df.write
   .format(“carbondata")
   .options("tableName“, “t1"))
   .mode(SaveMode.Overwrite)
   .save()
```

使用方式：查询

- SQL

```
SELECT project_list FROM t1
WHERE cond_list
GROUP BY columns
ORDER BY columns
```

- Dataframe

```
df = sparkSession.read
          .format(“carbondata”)
          .option(“tableName”, “t1”)
          .load(“path_to_carbon_file”)

df.select(…).show
```

使用方式：更新和删除

Modify one column in table1

```
UPDATE table1 A
SET A.REVENUE = A.REVENUE - 10
WHERE A.PRODUCT = 'phone'
```

phone, ~~70~~ 60
car,100
phone, ~~30~~ 20

Modify two columns in table1 with values from table2

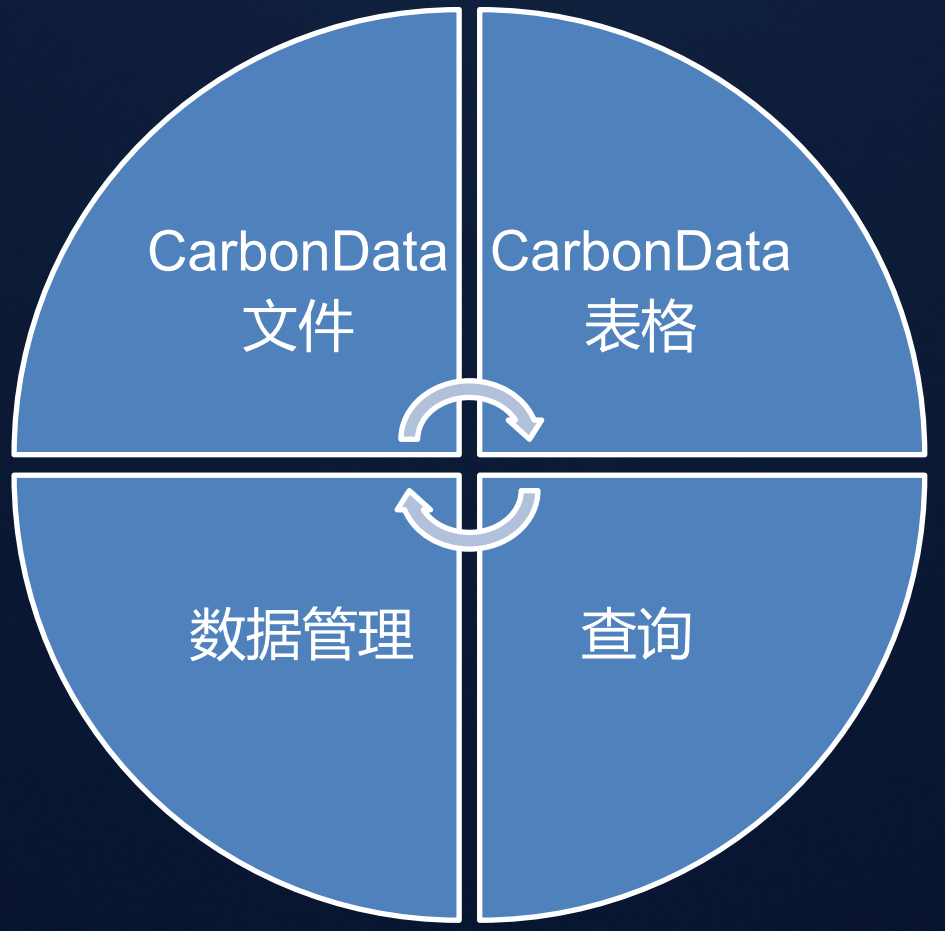
```
UPDATE table1 A
SET (A.PRODUCT, A.REVENUE) =
     (
      SELECT PRODUCT, REVENUE
      FROM table2 B
      WHERE B.CITY = A.CITY AND B.BROKER = A.BROKER
     )
WHERE A.DATE BETWEEN '2017-01-01' AND '2017-01-31'
```

Delete records in table1

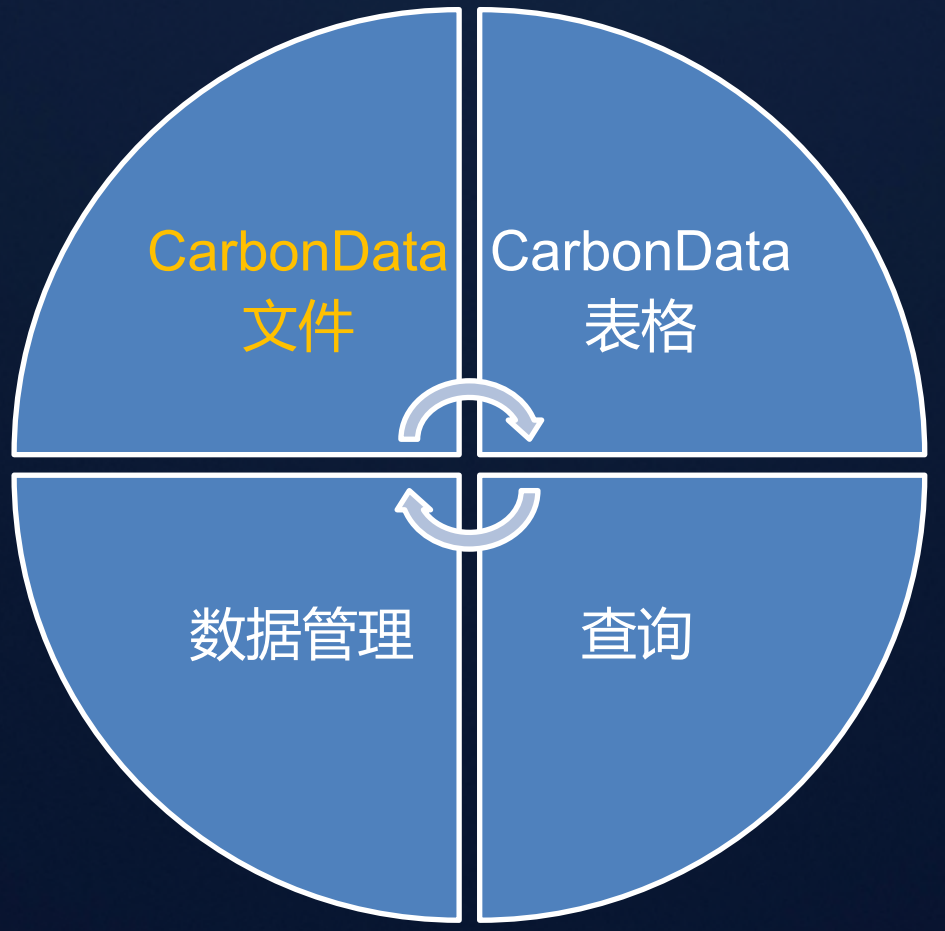
```
DELETE FROM table1 A
WHERE A.CUSTOMERID = '123'
```

~~123, abc~~
456, jkd

CarbonData介绍

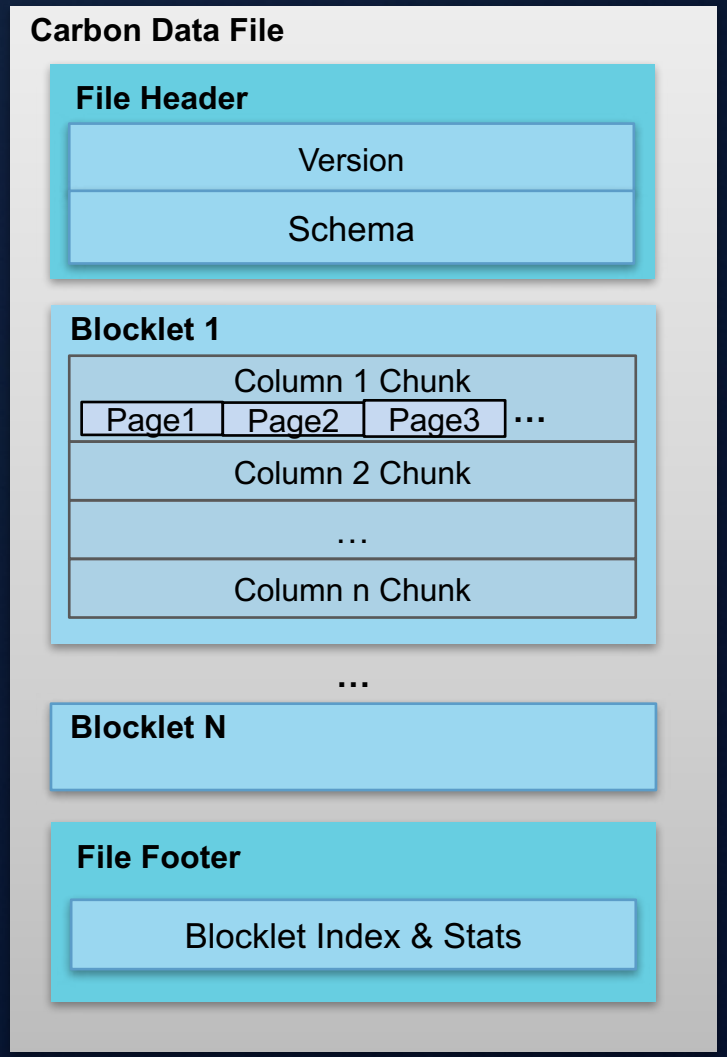


CarbonData介绍



CarbonData文件格式

- Blocklet: 文件内的数据块
  - Data are sorted along MDK (multi-dimensional keys)
  - Clustered data enabling efficient filtering and scan
- Column chunk: Blocklet内的列数据
  - 一次IO单元
  - 内部分为多个Page，解压单元
- 元数据和索引信息
  - Header：Version，Schema
  - Footer： Blocklet Index & statistics
- 内置索引
  - Multi-dimensional Index
  - Statistics: Column Min/Max, Cardinality



索引建立过程介绍

- 数据和索引合一存储，数据即索引
- 多维索引 (Multi-Dimensional Key)
- 全局字典编码

Years	Quarters	Months	Territory	Country	Quantity	Sales
2003	QTR1	Jan	EMEA	Germany	142	11,432
2003	QTR1	Jan	APAC	China	541	54,702
2003	QTR1	Jan	EMEA	Spain	443	44,622
2003	QTR1	Feb	EMEA	Denmark	545	58,871
2003	QTR1	Feb	EMEA	Italy	675	56,181
2003	QTR1	Mar	APAC	India	52	9,749
2003	QTR1	Mar	EMEA	UK	570	51,018
2003	QTR1	Mar	Japan	Japan	561	55,245
2003	QTR2	Apr	APAC	Australia	525	50,398
2003	QTR2	Apr	EMEA	Germany	144	11,532

Encoding

[1,1,1,1,1] : [142,11432]
[1,1,1,3,2] : [541,54702]
[1,1,1,1,3] : [443,44622]
[1,1,2,1,4] : [545,58871]
[1,1,2,1,5] : [675,56181]
[1,1,3,3,6] : [52,9749]
[1,1,3,1,7] : [570,51018]
[1,1,3,2,8] : [561,55245]
[1,2,4,3,9] : [525,50398]
[1,2,4,1,1] : [144,11532]

Sort
(MDK Index)

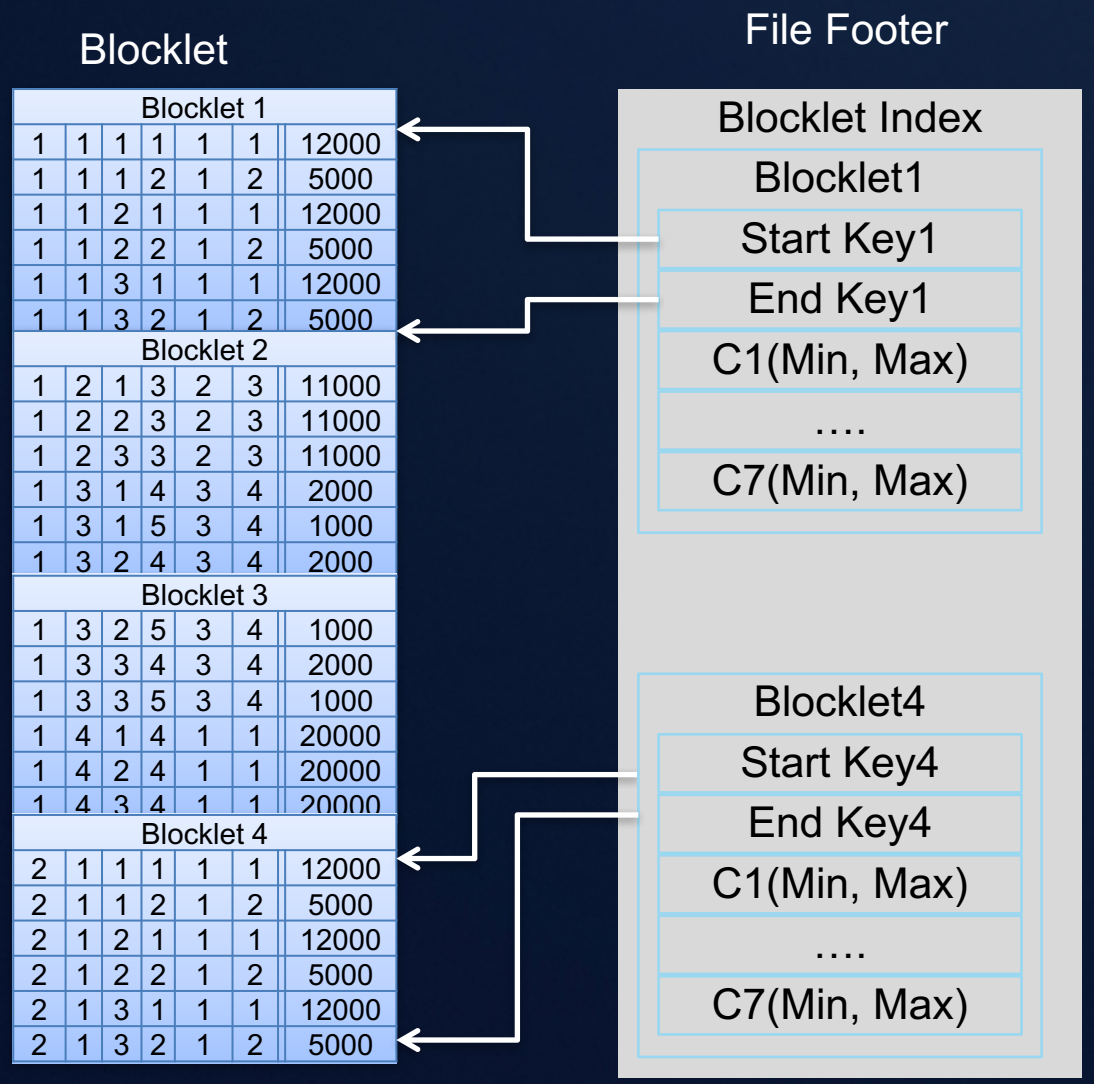
Sorted MDK Index

[1,1,1,1,1] : [142,11432]
[1,1,1,1,3] : [443,44622]
[1,1,1,3,2] : [541,54702]
[1,1,2,1,4] : [545,58871]
[1,1,2,1,5] : [675,56181]
[1,1,3,1,7] : [570,51018]
[1,1,3,2,8] : [561,55245]
[1,1,3,3,6] : [52,9749]
[1,2,4,1,1] : [144,11532]
[1,2,4,3,9] : [525,50398]

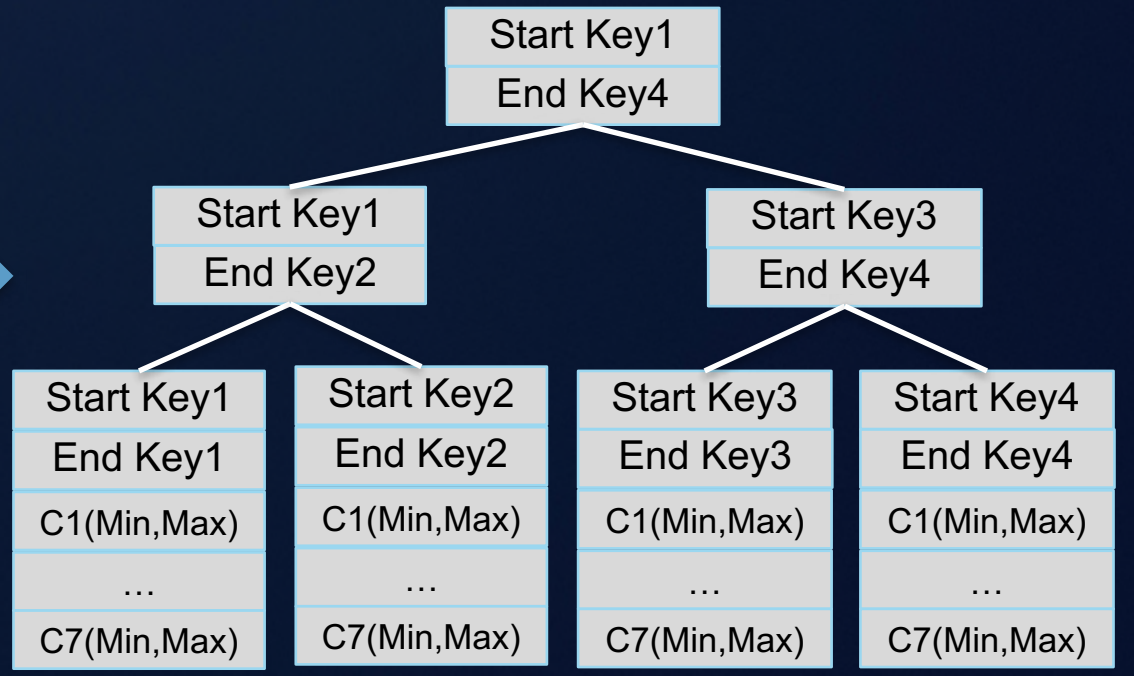
Blocklet (Columnar view)

C1	C2	C3	C4	C5	C6	C7
1	1	1	1	1	142	11432
1	1	1	1	3	443	44622
1	1	1	3	2	541	54702
1	1	2	1	4	545	58871
1	1	2	1	5	675	56181
1	1	3	1	7	570	51018
1	1	3	2	8	561	55245
1	1	3	3	6	52	9749
1	2	4	1	1	144	11532
1	2	4	3	9	525	50398

索引建立过程介绍（二）



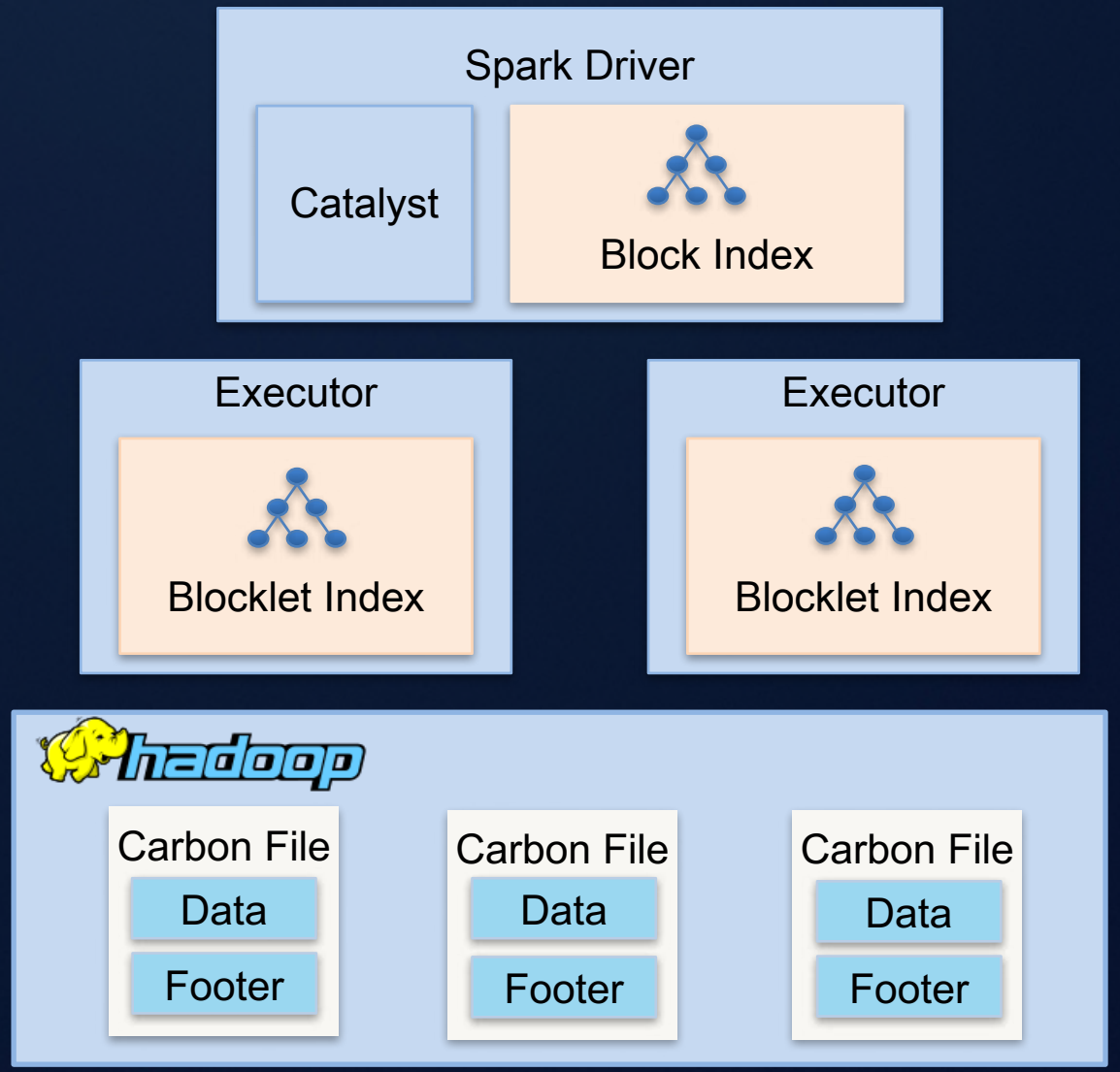
- Build in-memory file level MDK index tree for filtering
- Major optimization for efficient scan



利用两级索引架构减少Spark Task数和磁盘IO

- 第一级：文件级索引
用于过滤文件（HDFS Block），避免扫描不必要的文件，减少多达95%的Spark Task

- 第二级：Blocklet索引
用于过滤文件内部的Blocklet，避免扫描不必要的Blocklet，减少磁盘IO



支持嵌套数据类型：Array, Struct

Array

- Represented as a composite of two columns
- One column for the element value
- One column for start index & length of Array

Struct

- Represented as a composite of finite number of columns
- Each struct element is a separate column

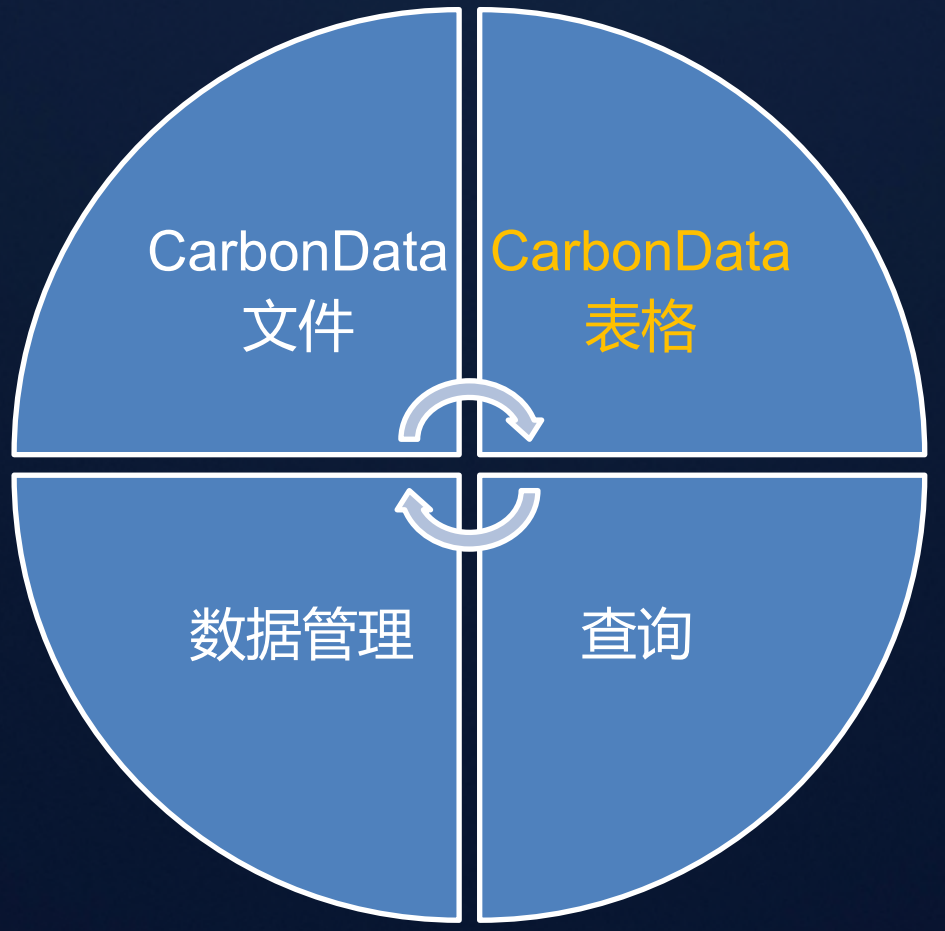
Name	Array<Ph_Number>
John	[192,191]
Sam	[121,345,333]
Bob	[198,787]

Name	Array [start,len]	Ph_Number
John	0,2	192
Sam	2,3	191
Bob	5,2	121
		345
		333
		198
		787

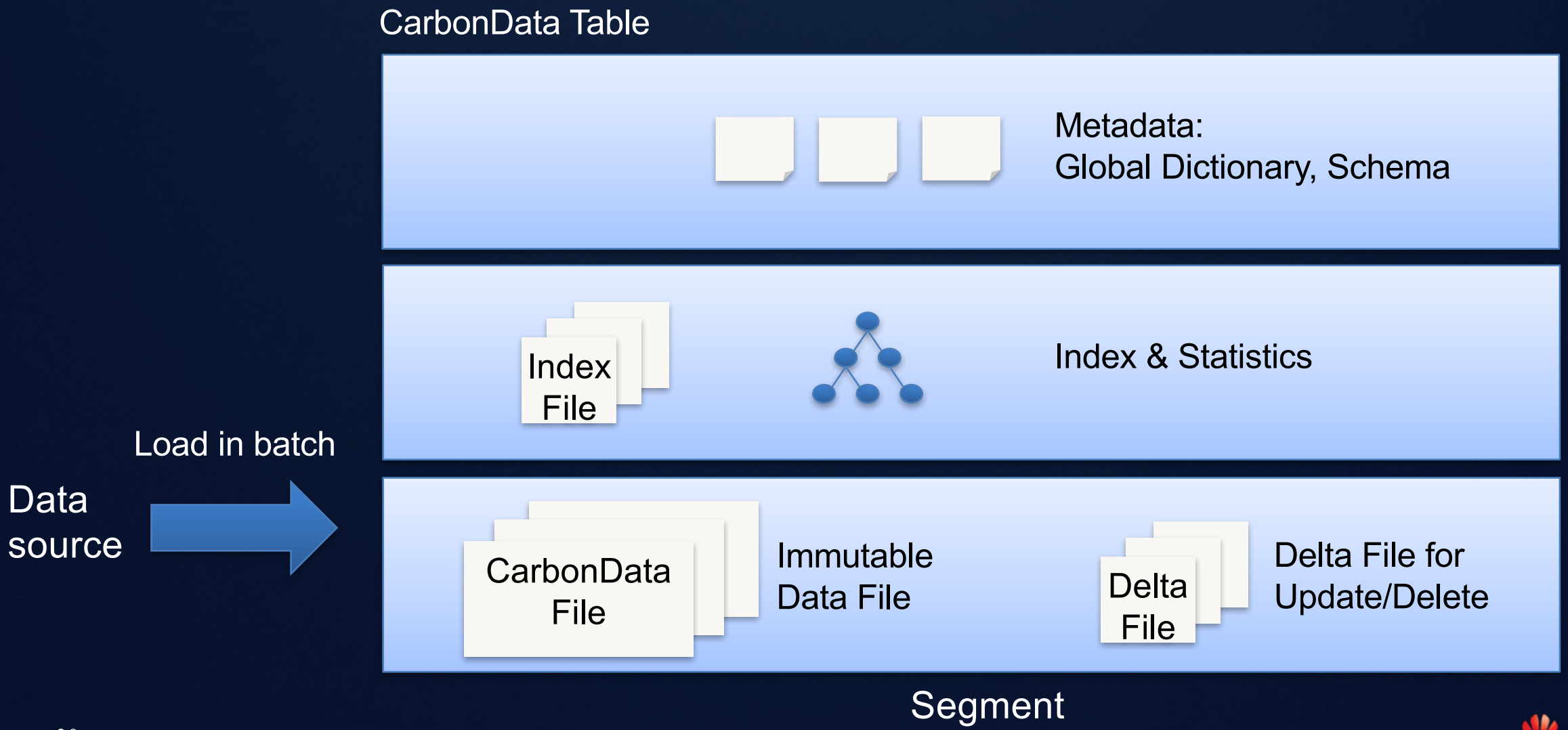
Name	Info Strut<age,gender>
John	[31,M]
Sam	[45,F]
Bob	[16,M]

Name	Info.age	Info.gender
John	31	M
Sam	45	F
Bob	16	M

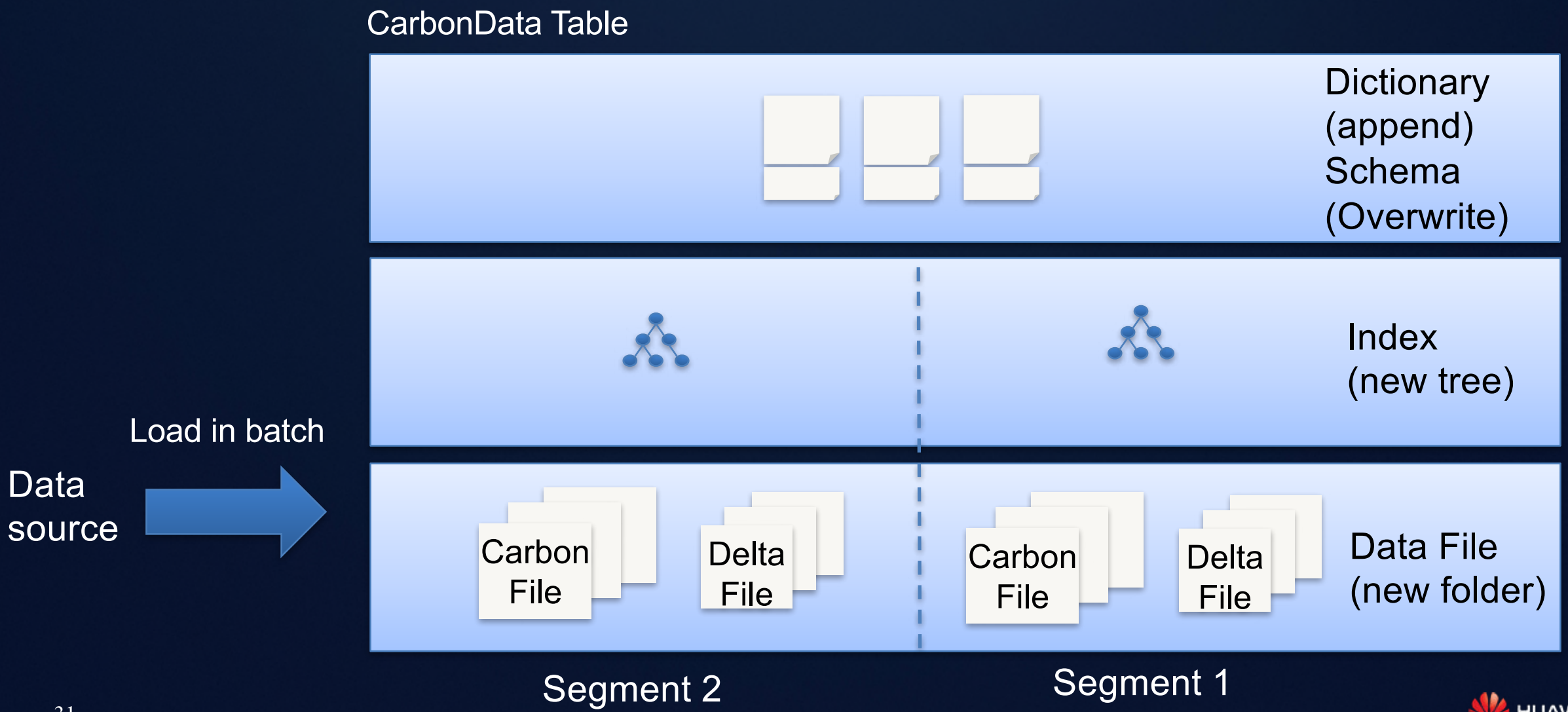
CarbonData介绍



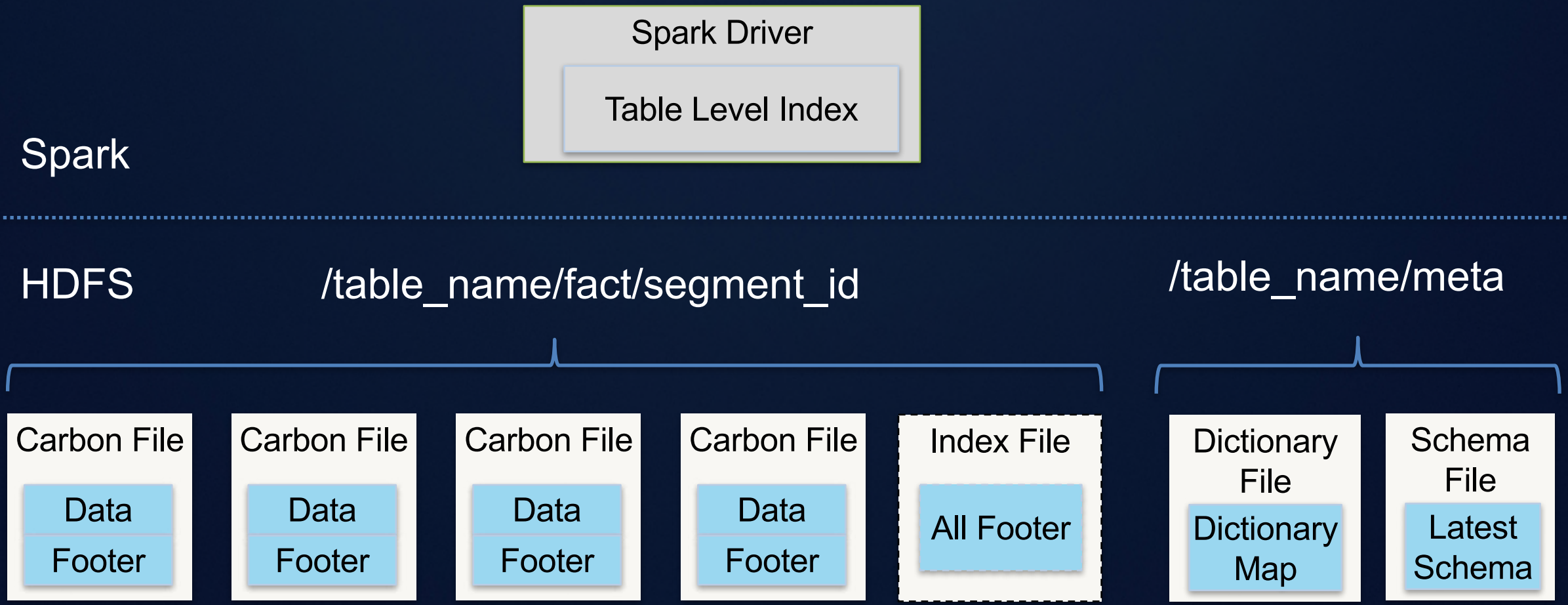
CarbonData表格组成



CarbonData表格组成：多次入库

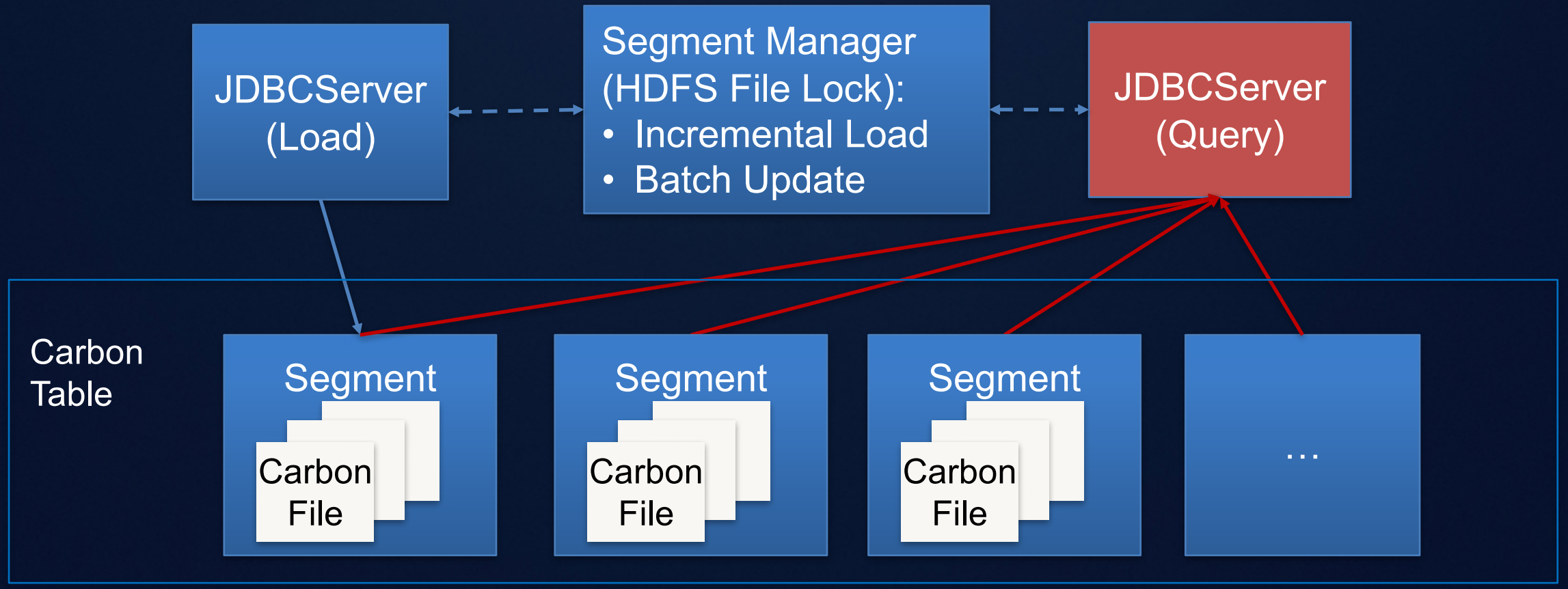


CarbonData表格物理存储

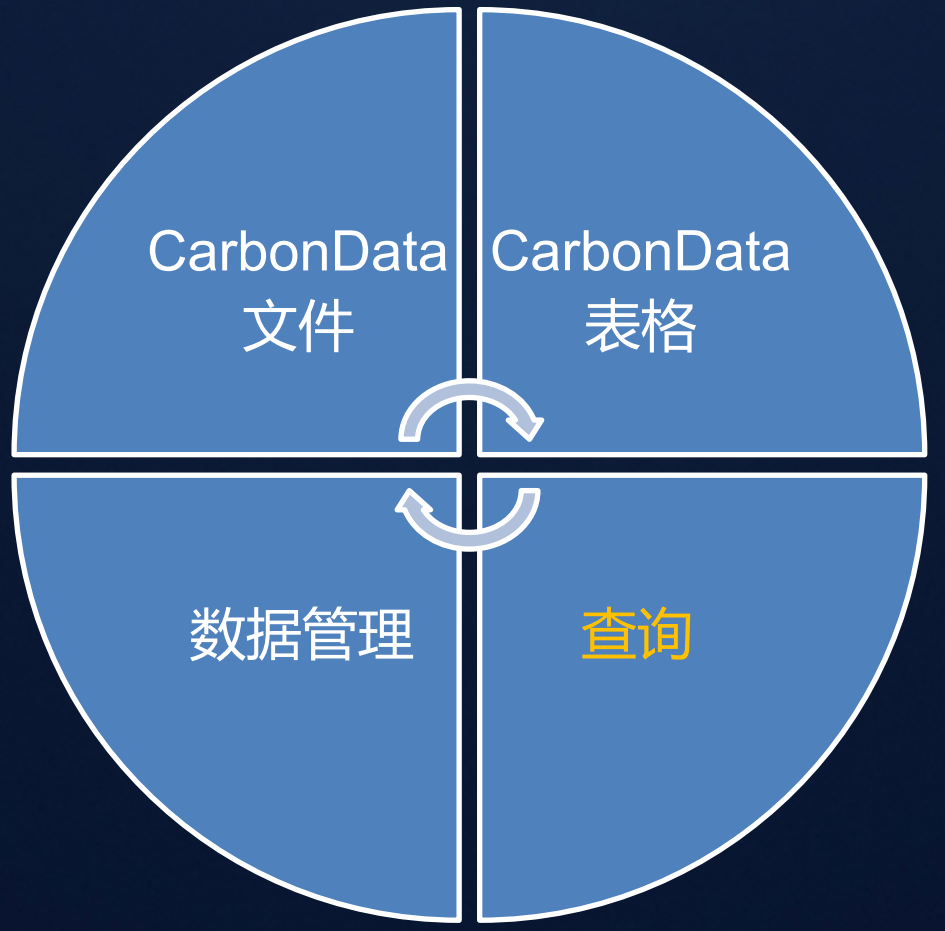


Segment管理

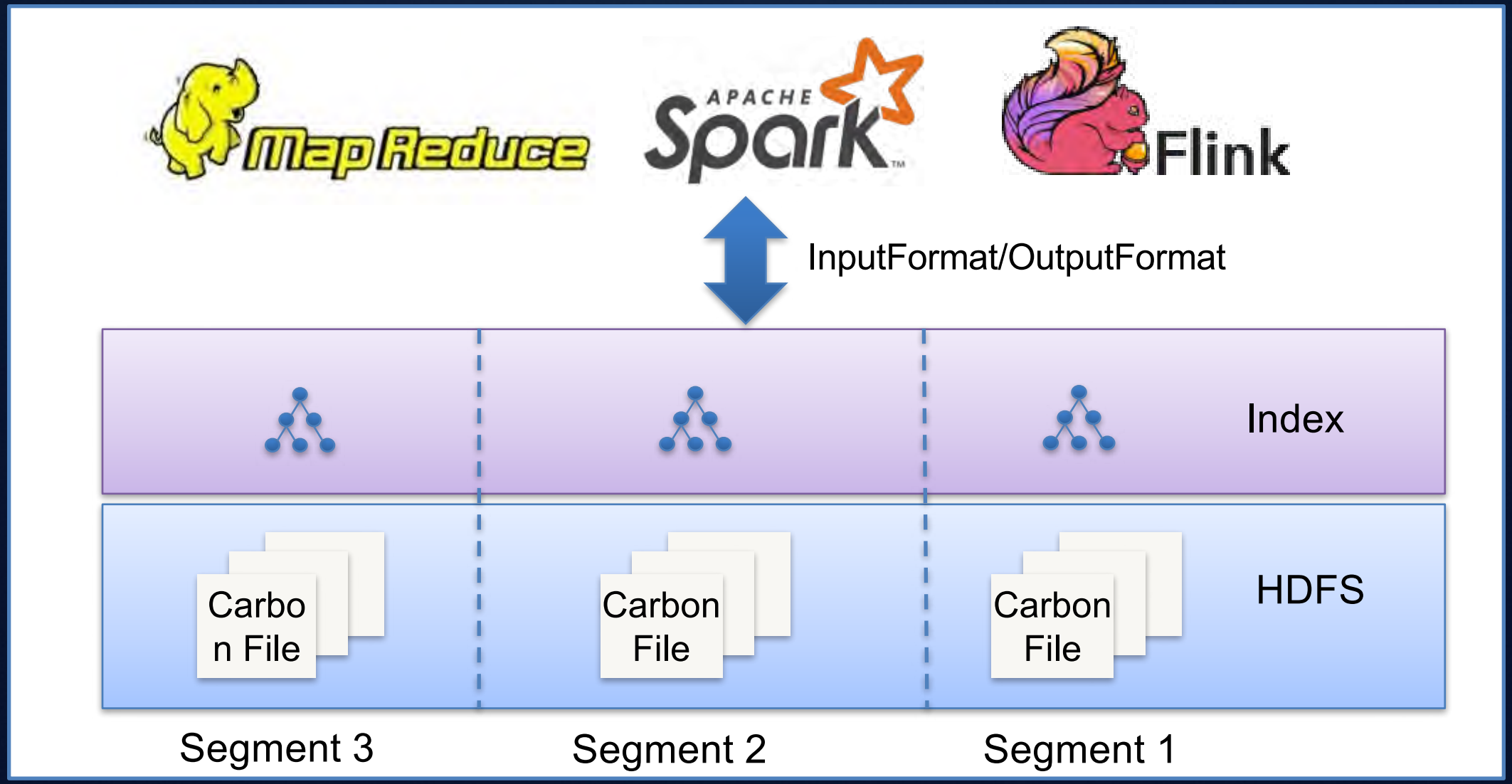
Leveraging HDFS File Lock to manage the Segment State, Data Update



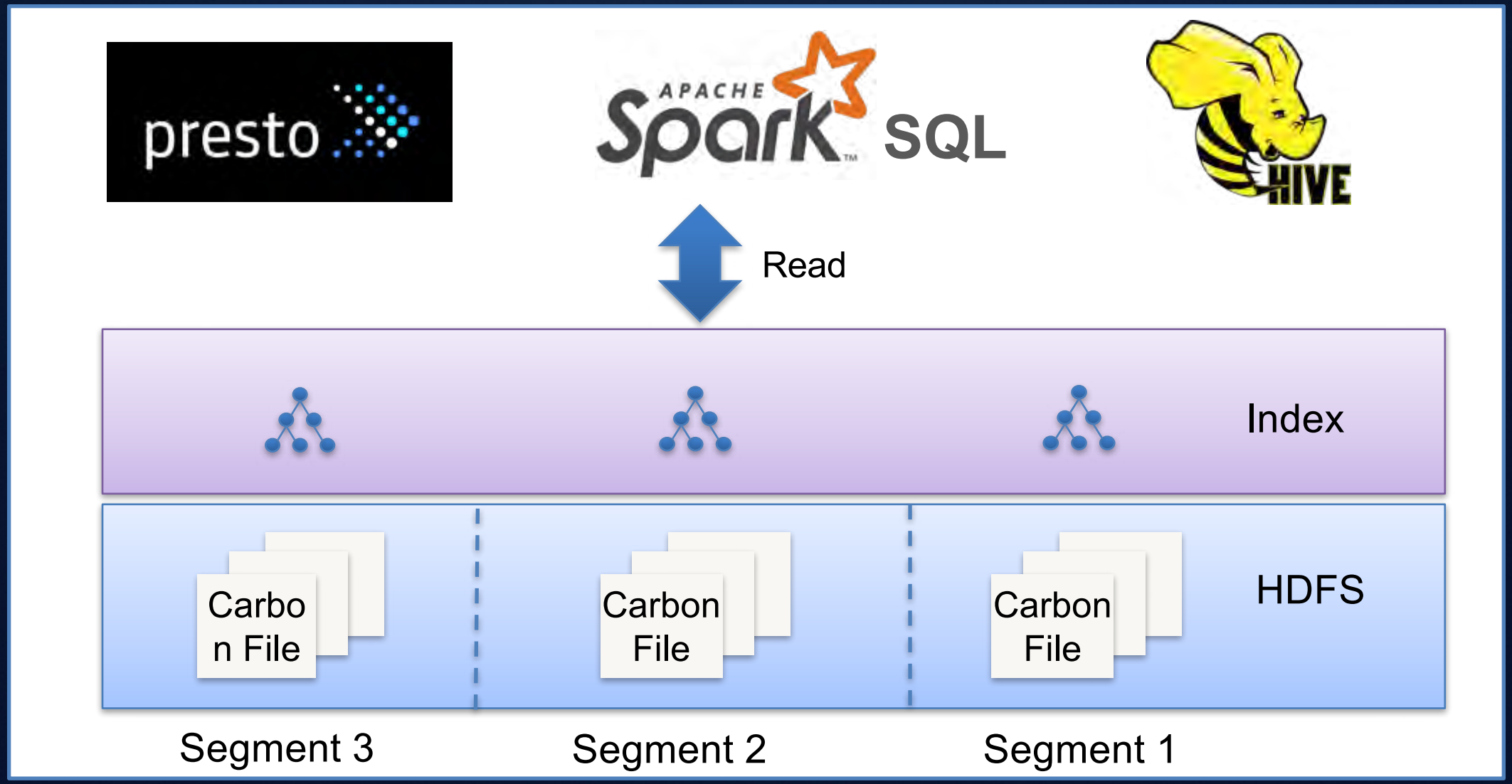
CarbonData介绍



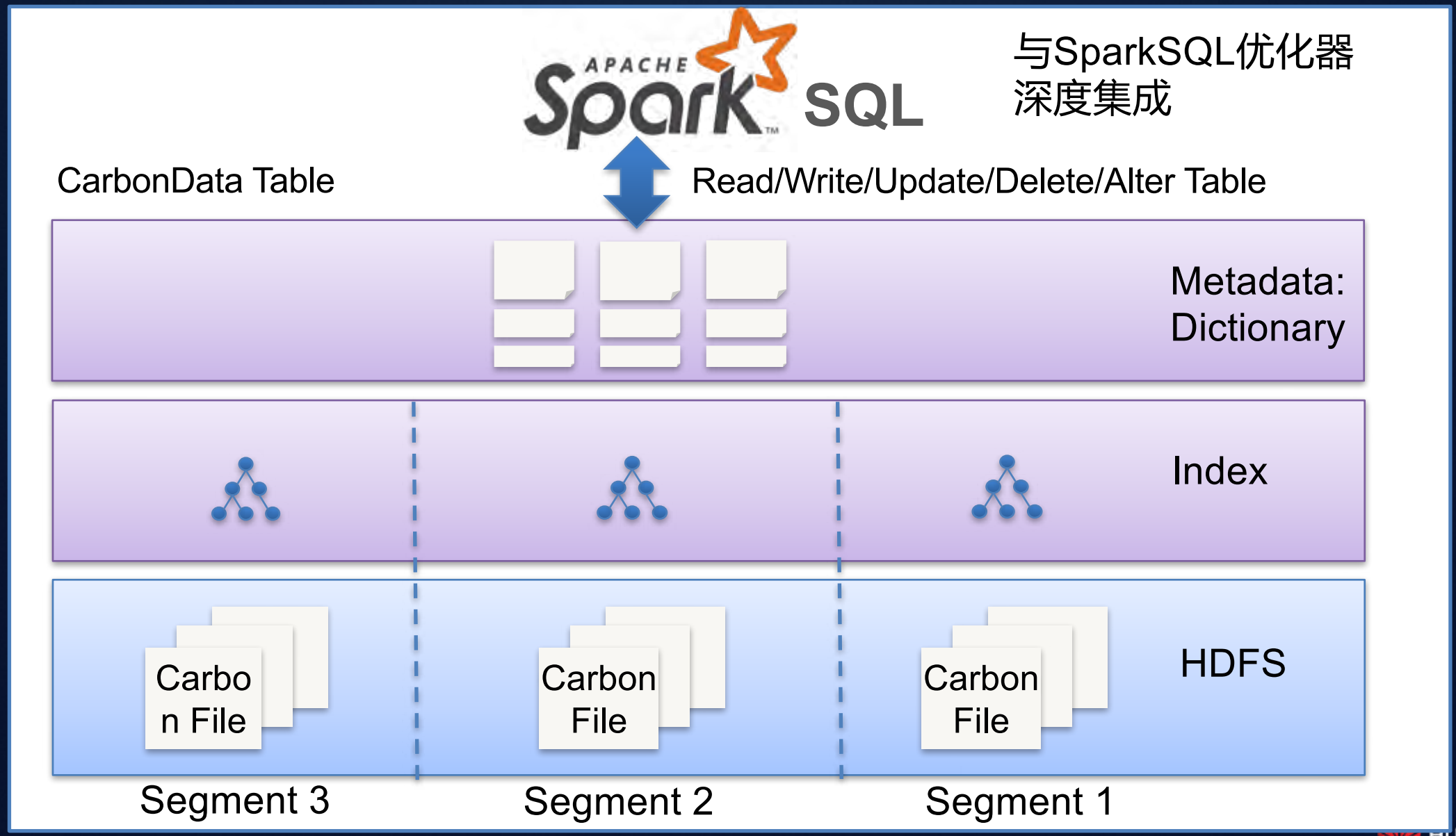
文件级对接：MapReduce



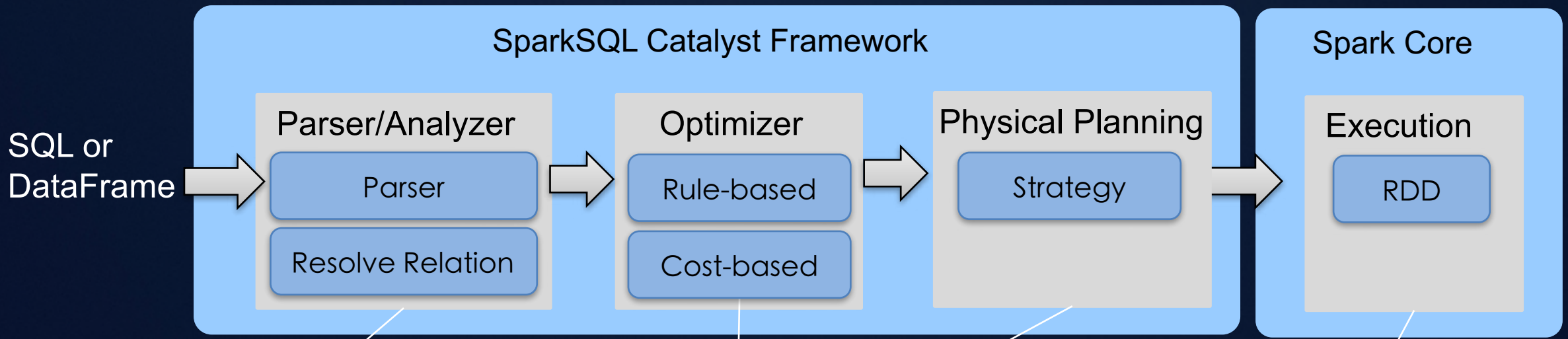
SQL引擎对接：表格



SparkSQL深度对接



SparkSQL Data Source对接



Carbon Data Source

New SQL syntax
- DML related statement

Carbon-specific optimization rule:
- Lazy Decode leveraging global dictionary

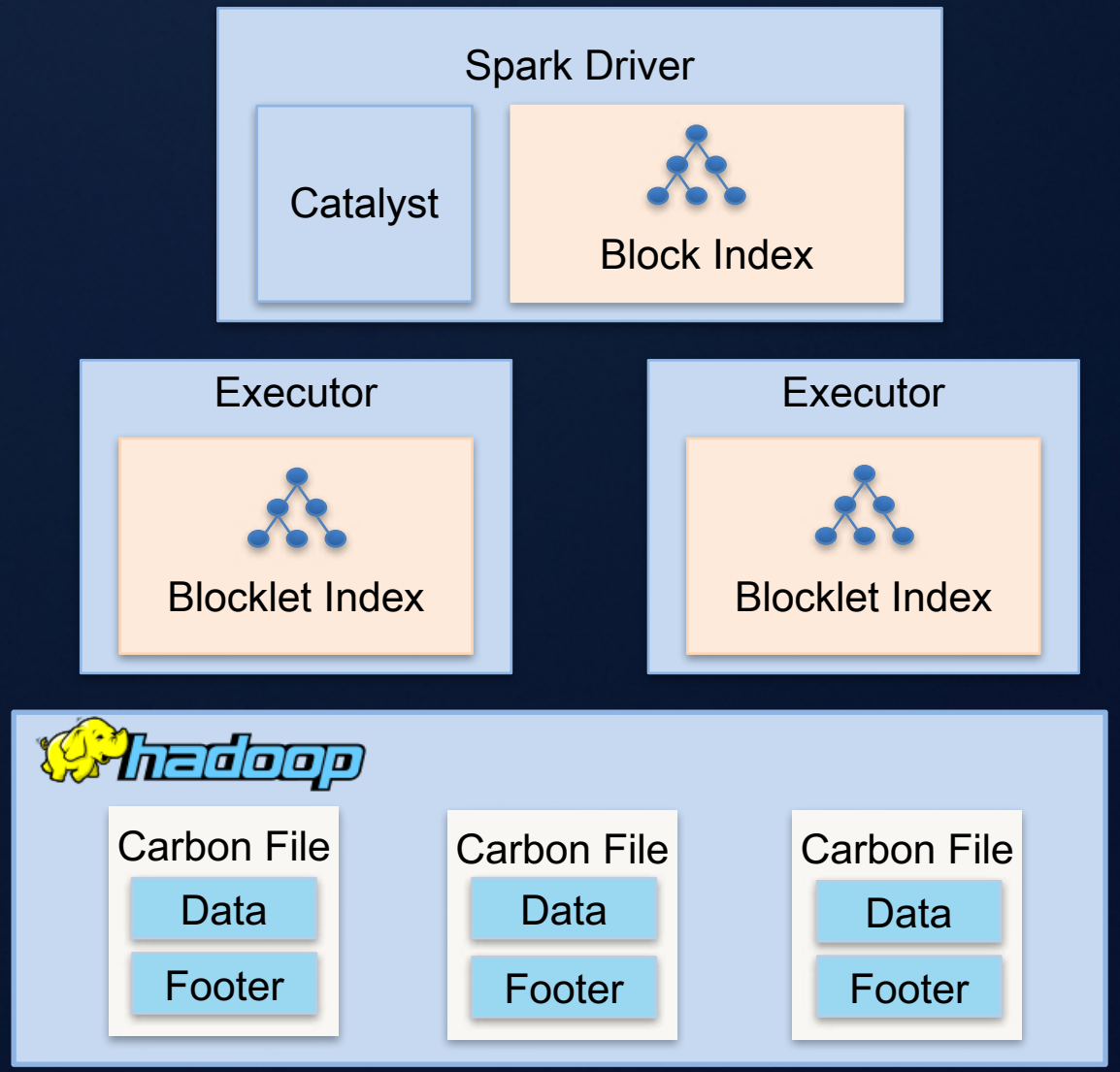
CarbonScanRDD:
- Leveraging multi level index for efficient filtering and scan

DML related RDDs

利用两级索引架构减少Spark Task数和磁盘IO

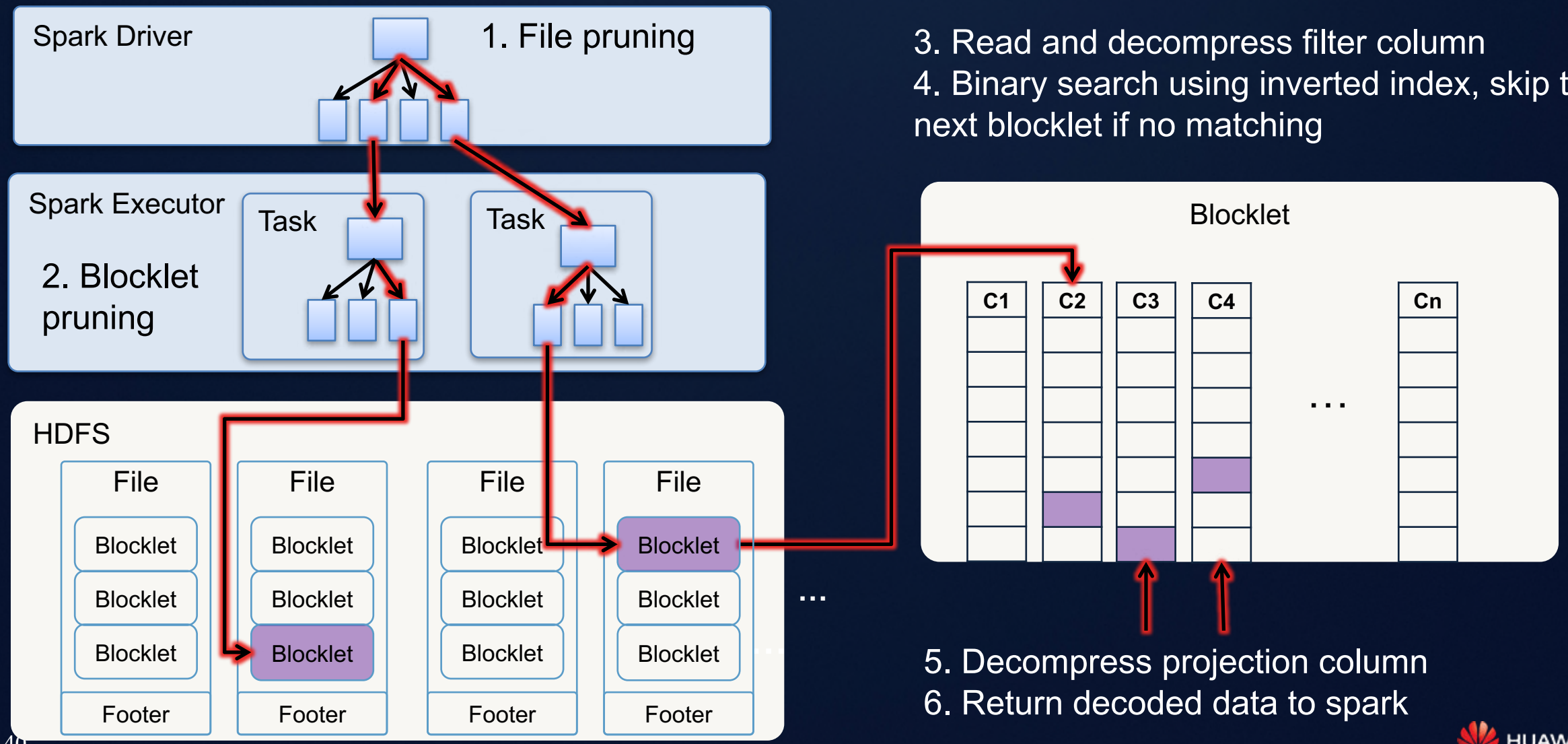
- 第一级：文件级索引
用于过滤文件（HDFS Block），避免扫描不必要的文件，减少多达95%的Spark Task

- 第二级：Blocklet索引
用于过滤文件内部的Blocklet，避免扫描不必要的Blocklet，减少磁盘IO

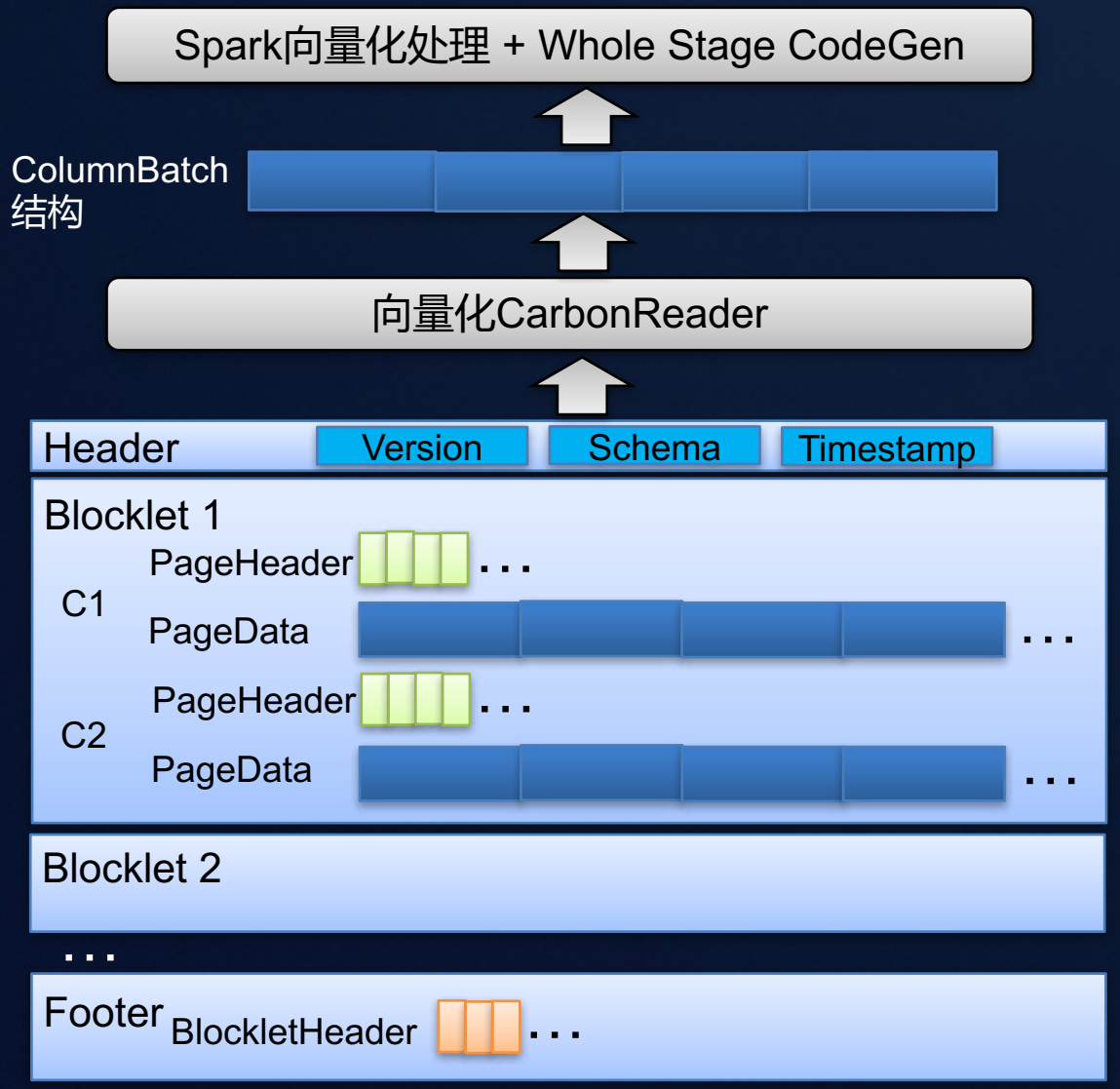


一个完整的过滤查询过程

```
SELECT c3, c4 FROM t1 WHERE c2='beijing'
```



文件内扫描优化，与Spark向量化处理对接



大颗粒IO单元（Carbon V3格式）：

•Blocklet内部一个Column Chunk 是一个IO单元，Blocklet按大小切分，默认64MB，大概有100万行记录。大颗粒顺序读提升扫描性能。

跳跃解码（Carbon V3格式）：

•增加数据页Page概念，按Page进行过滤和解码，减少不必要的解码解压缩，提升CPU利用率

向量化解码 + Spark向量化处理

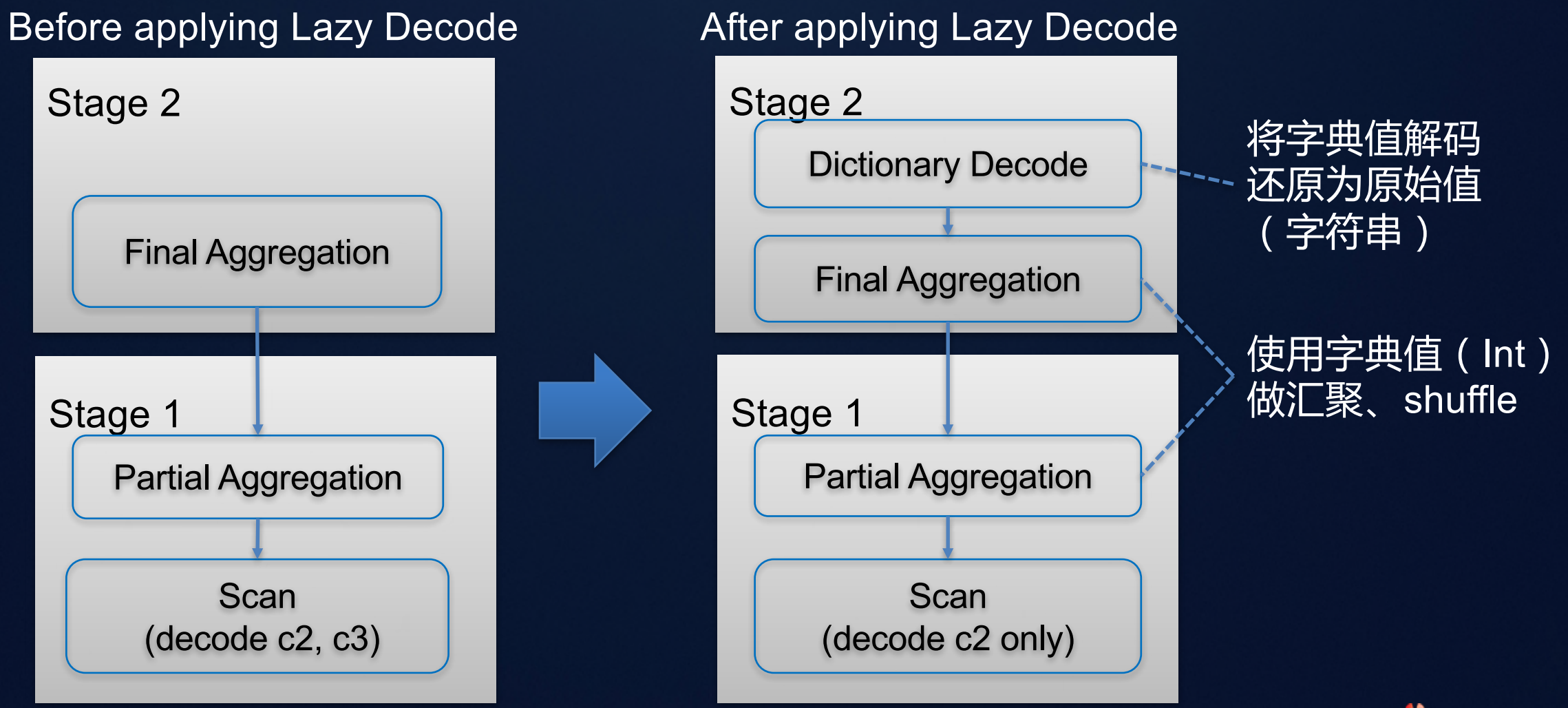
•解码和解压缩采用向量化处理，与Spark2.1向量化、Codegen结合，在扫描+聚合场景下提升性能4X

堆外内存：

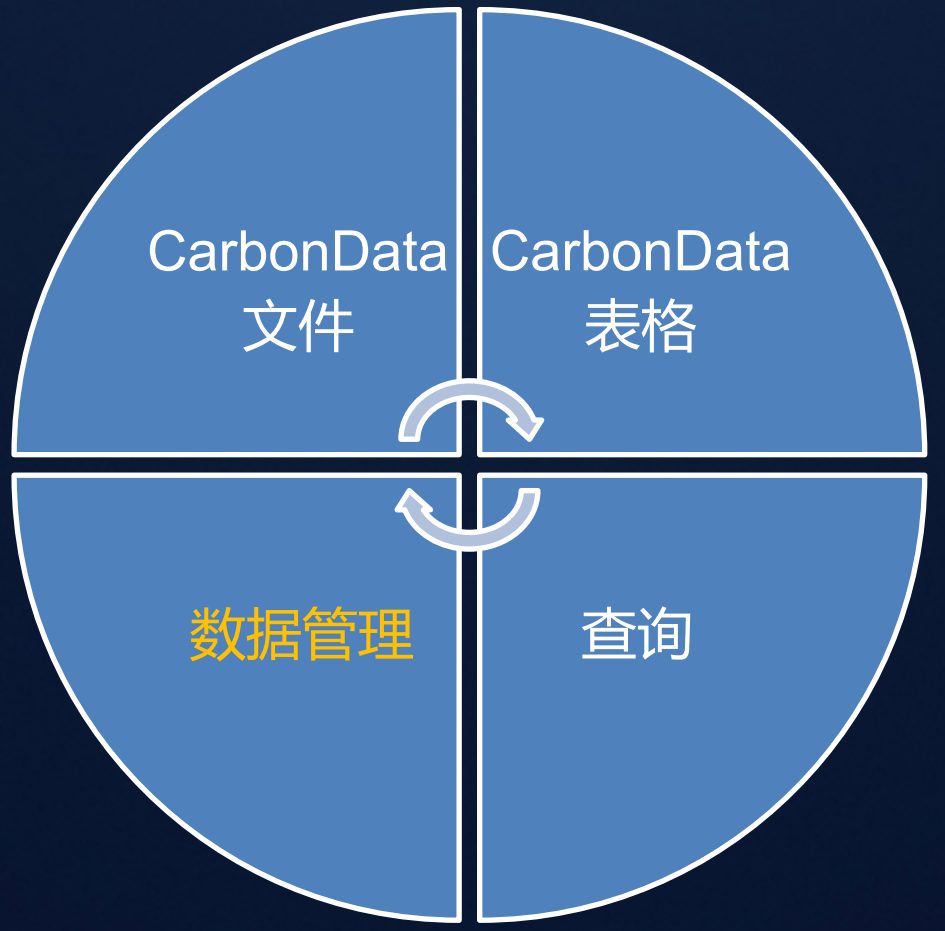
•处理大结果集时，解码解压缩过程中堆外完成，减少GC

利用全局字典做延迟解码，提升GroupBy性能

```
SELECT c3, sum(c2) FROM t1 GROUP BY c3
```



CarbonData介绍



增量入库：排序Key和排序范围

```
CREATE TABLE table(c1 STRING, c2 INT, c3 STRING, c4 INT, c5 DOUBLE)
STORED BY “carbondata”
TBLPROPERTIES (‘SORT_COLUMNS’=‘c2, c3’)

LOAD DATA INPATH ‘source path’ INTO TABLE table
OPTIONS (‘SORT_SCOPE’=‘NO_SORT/BATCH_SORT/LOCAL_SORT’)
```

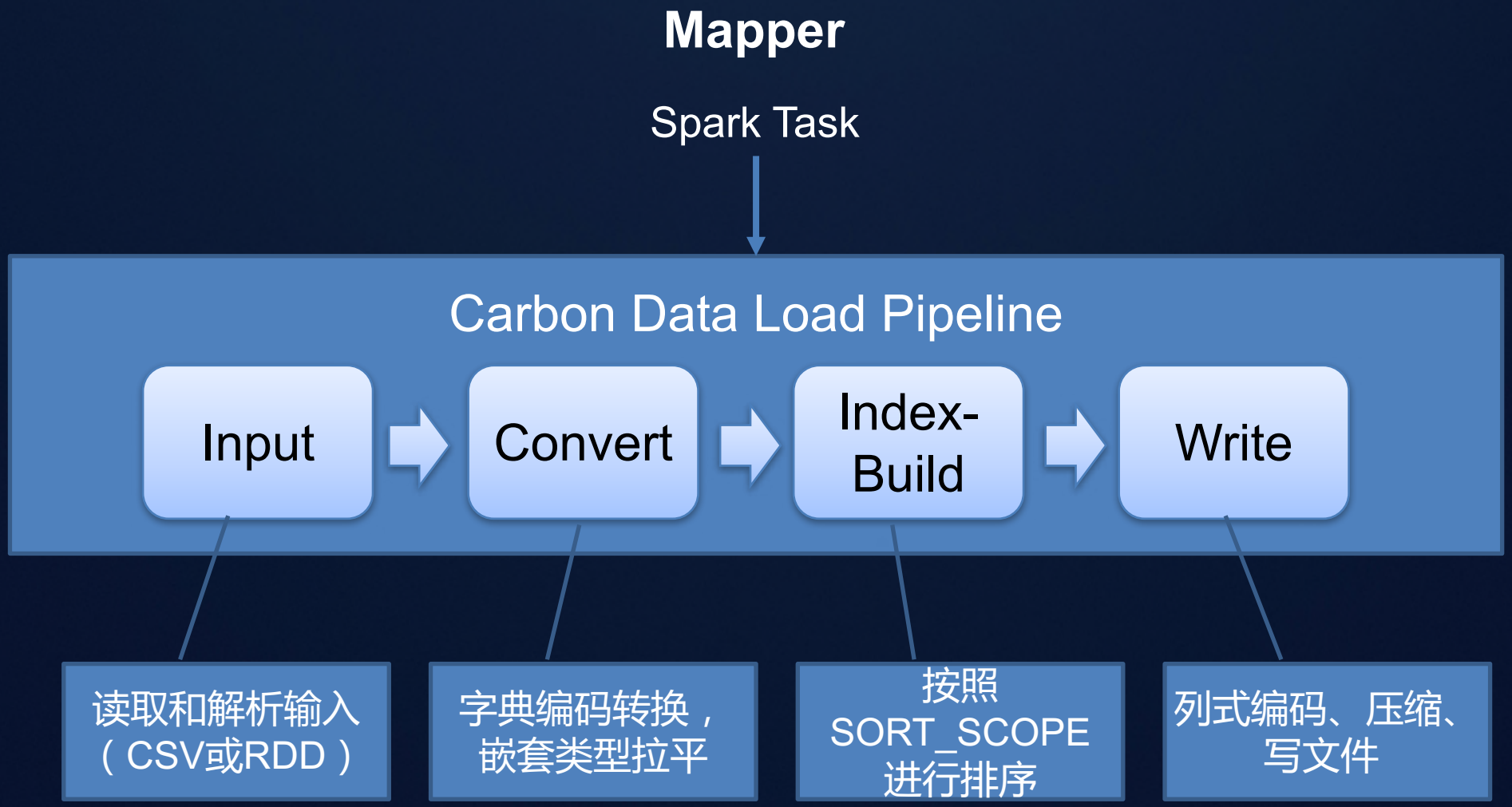
通过SORT_COLUMNS指定需要排序的Key。
如果不指定，系统会使用默认规则（查询性能最大化）：

- 在所有维度列上建立MDK索引
- 排序范围是LOCAL_SORT

每次入库时可指定三种排序范围：

- NO_SORT：不排序，在Compaction时建立索引。
- BATCH_SORT：使用配置的内存大小进行“尽量排序”
- LOCAL_SORT：在节点内部排序（Merge Sort）

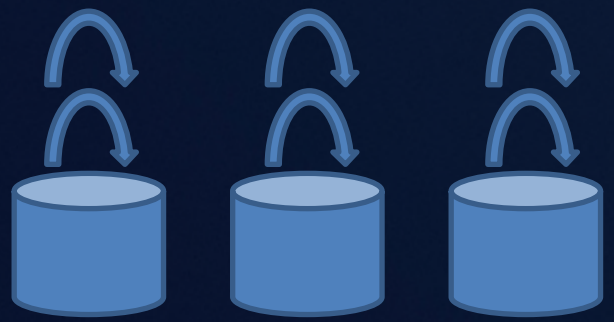
入库流水线介绍



全局字典的生成方式

Two Pass

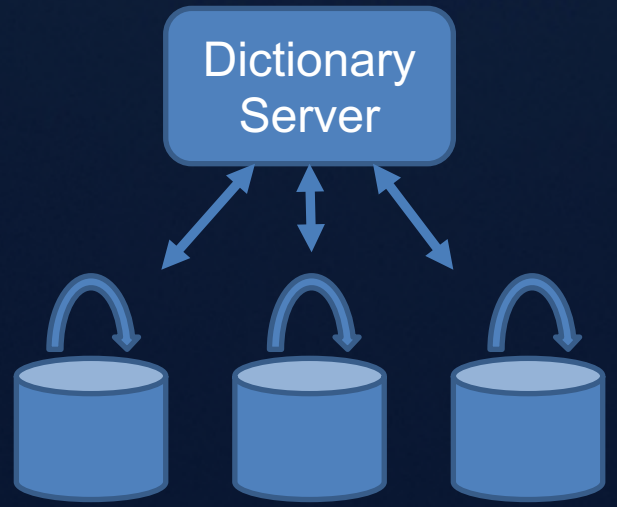
Job1：生成字典
Job2：执行入库流程



场景：首次入库

Single Pass

由Dictionary Server在线生成字典



场景：非首次入库；
若维度Cardinality不高，也可用于首次入库

Pre-defined Dictionary

入库前手工生成字典

```
LOAD DATA 'path' INTO TABLE t
OPTIONS('COLUMNDICT'='nplist')

nplist := column_name:path,…
```

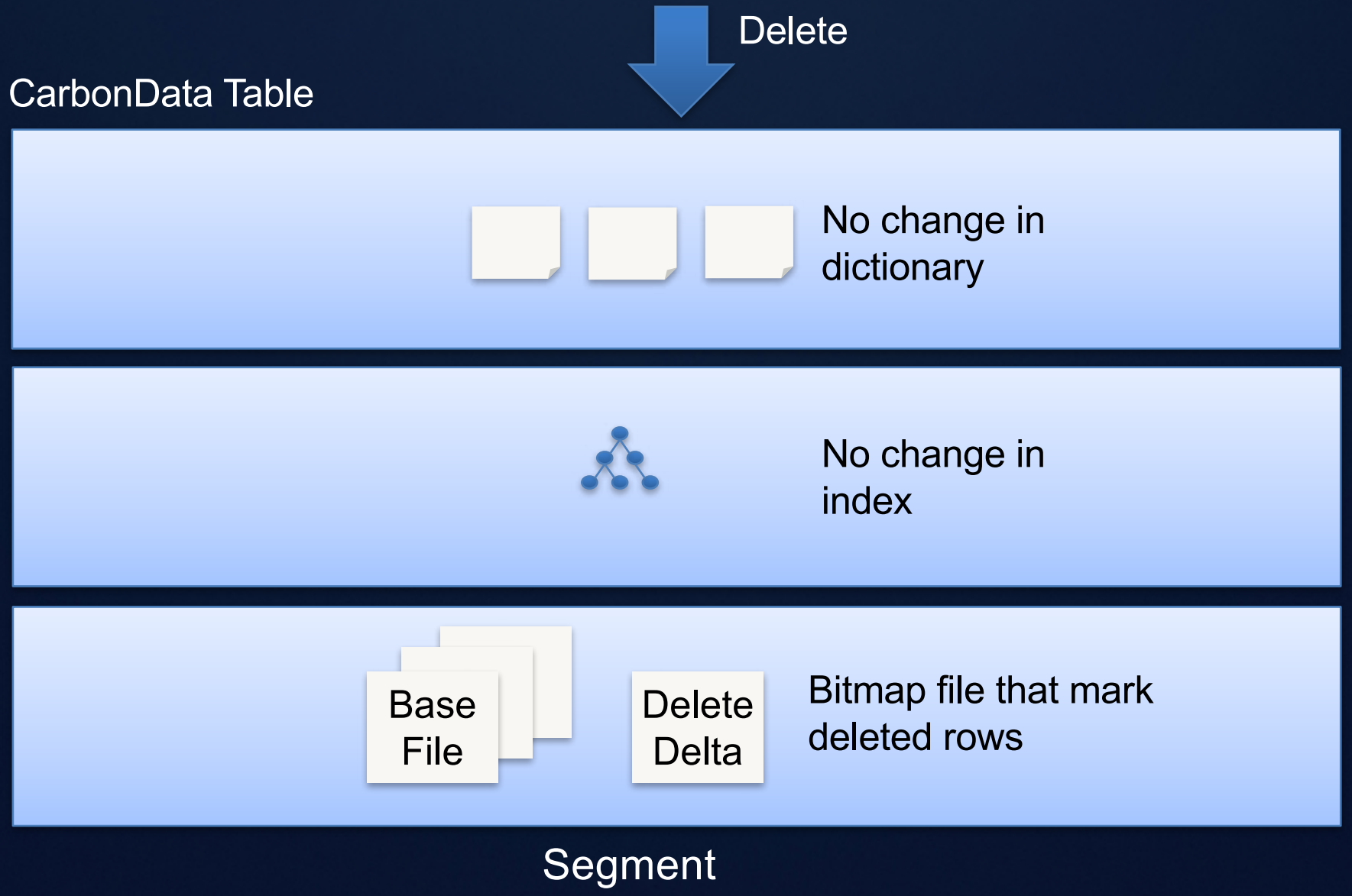
场景：通过静态维表预生产字典

数据更新和删除

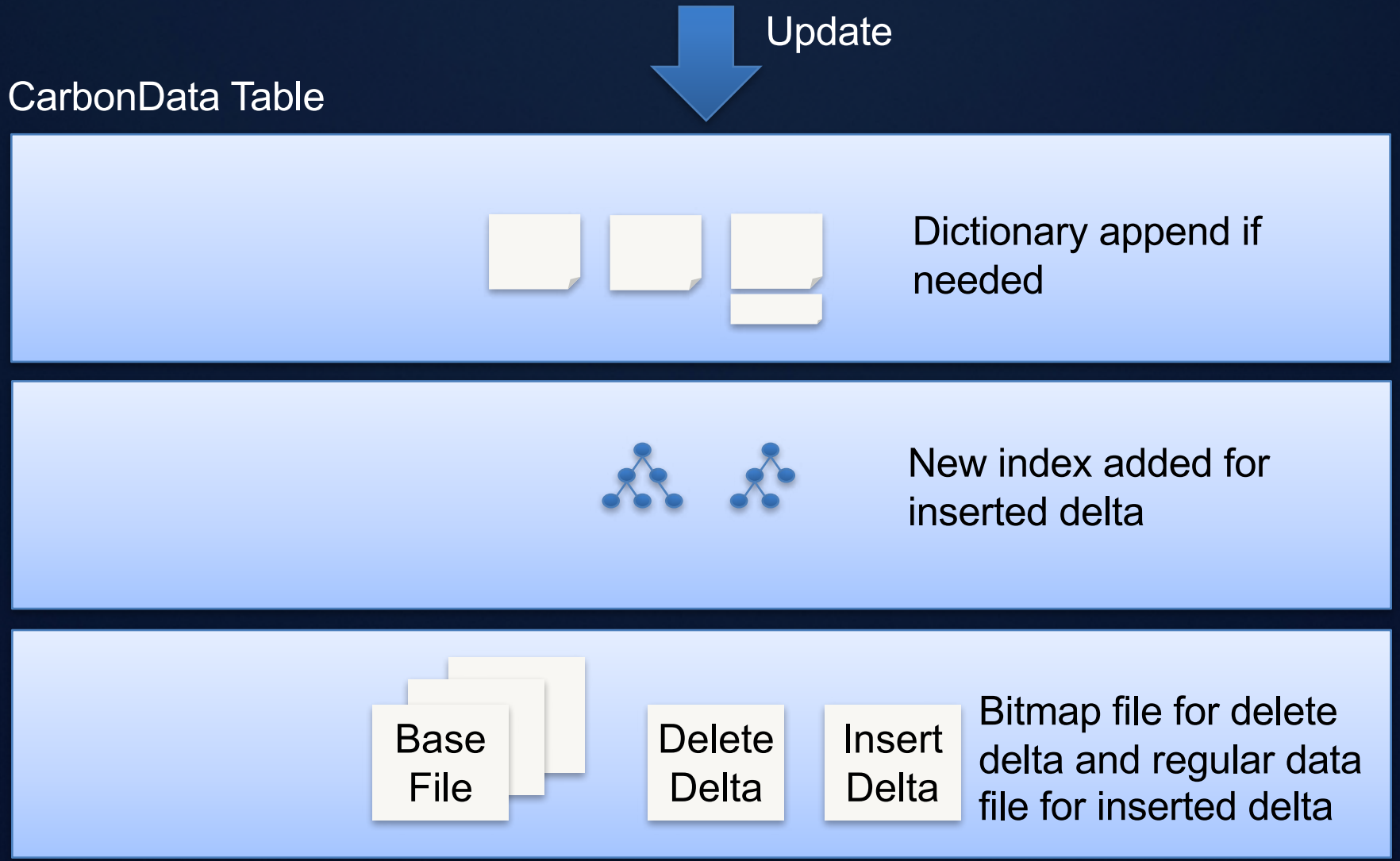
- 场景：
  - 批量更新的事实表，如更新某些度量值，ID值
  - 缓慢变化的维表，如刷新用户信息

- ACID属性：
  - 支持更新语句原子性（要么成功要么失败），一致性（更新串行化）
  - 更新过程中不影响查询
  - 适用于OLAP型+批量更新场景，不支持毫秒级实时更新，不支持OLTP类应用

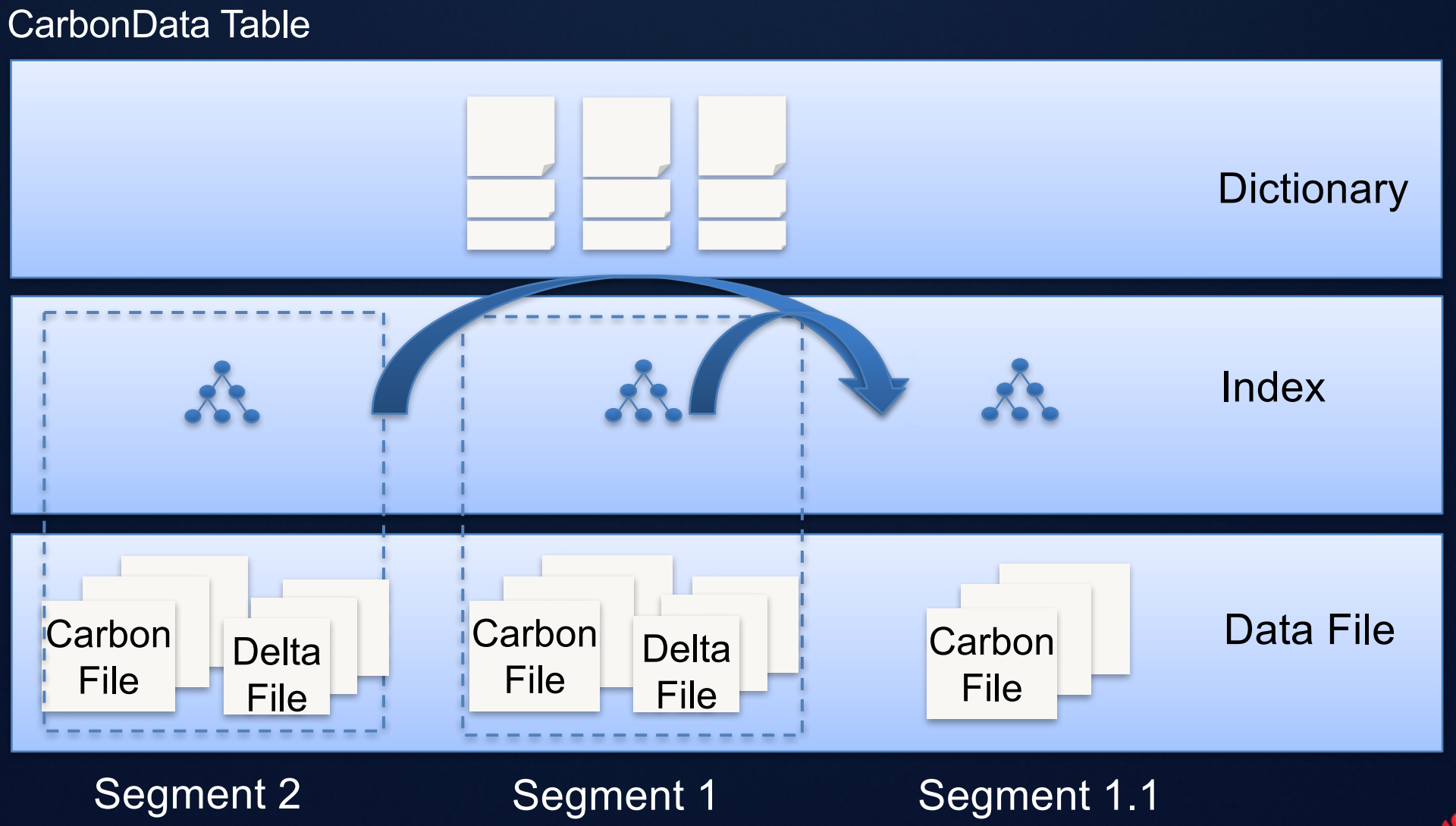
删除数据


Delete
CarbonData Table
No change in dictionary
No change in index
Base File
Delete Delta
Bitmap file that mark deleted rows
Segment

数据更新


Update
CarbonData Table
Dictionary append if needed
New index added for inserted delta
Base File
Delete Delta
Insert Delta
Bitmap file for delete delta and regular data file for inserted delta
Segment

Compaction



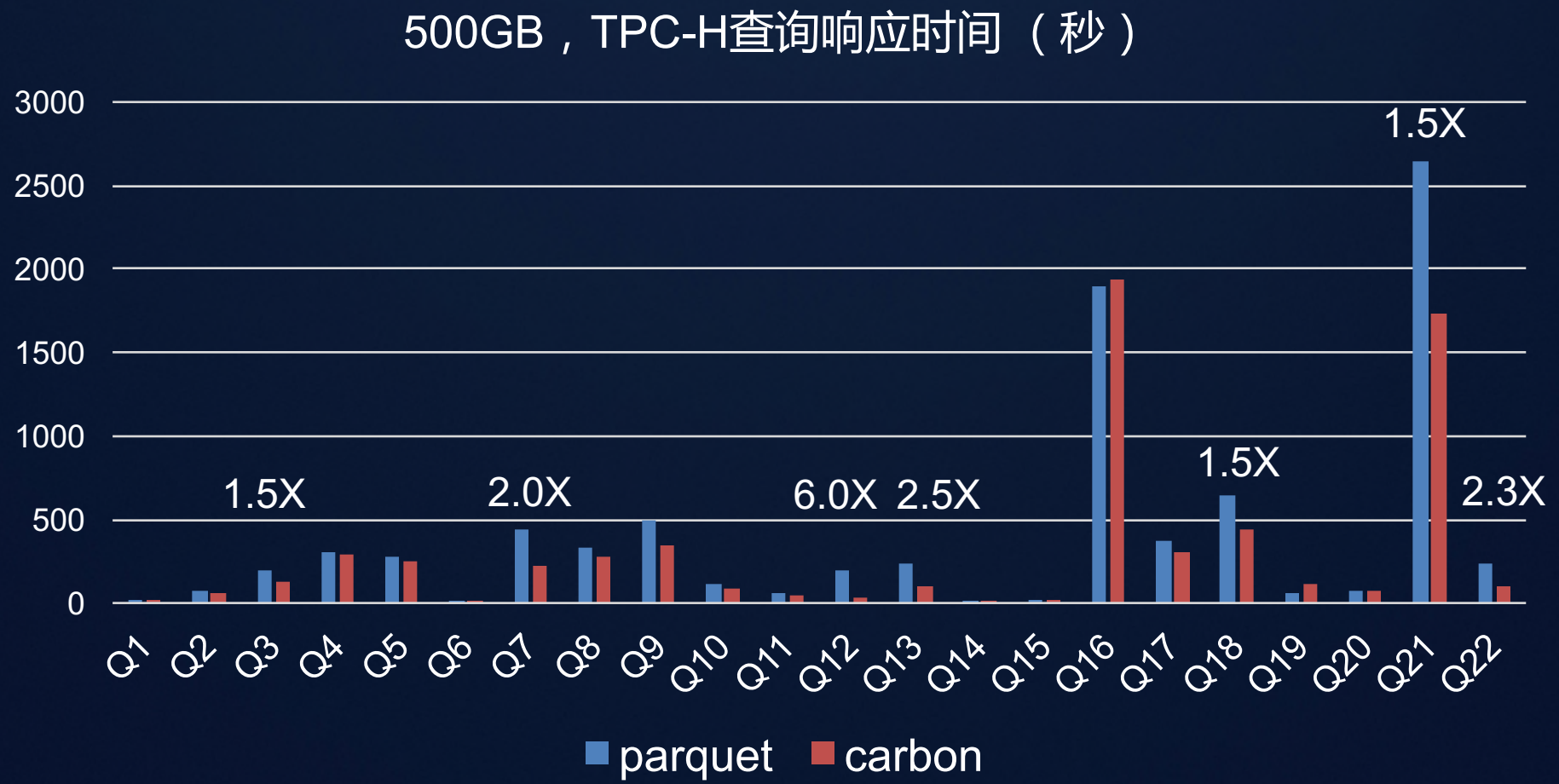
性能测试

测试用例

1. TPC-H benchmark 测试 (500GB)
2. 真实场景测试 (十亿级数据)
3. 可扩展性测试 (100个节点，万亿级数据，103TB)

- 存储层
  - Parquet:
    - 分区列：time column (c1)
  - CarbonData:
    - 多维索引：c1~c10
- 计算层
  - Spark 2.1

TPC-H: 查询速度



为什么Query12快了6倍？

```
/* TPC_H Query 12 - Shipping Modes and Order Priority */
SELECT L_SHIPMODE,
SUM(CASE WHEN O_ORDERPRIORITY = '1-URGENT' OR O_ORDERPRIORITY = '2-
HIGH' THEN 1 ELSE 0 END) AS HIGH_LINE_COUNT,
SUM(CASE WHEN O_ORDERPRIORITY <> '1-URGENT' AND O_ORDERPRIORITY <> '2-
HIGH' THEN 1 ELSE 0 END ) AS LOW_LINE_COUNT
FROM ORDERS, LINEITEM
WHERE O_ORDERKEY = L_ORDERKEY AND L_SHIPMODE IN ('MAIL','SHIP')
AND L_COMMITDATE < L_RECEIPTDATE AND L_SHIPDATE < L_COMMITDATE AND L_RECEI
PTDATE >= '1994-01-01'
AND L_RECEIPTDATE < dateadd(mm, 1, cast('1995-09-01' as date))
GROUP BY L_SHIPMODE
ORDER BY L_SHIPMODE
```

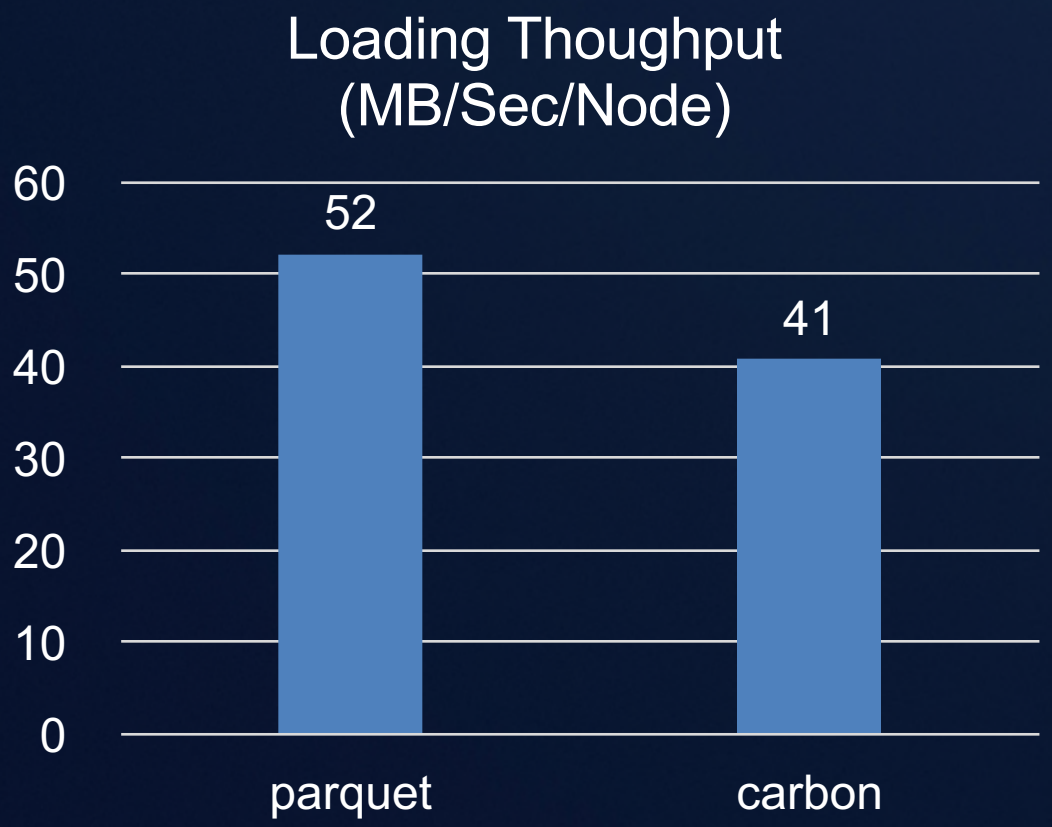
Parquet资源使用(nmon)

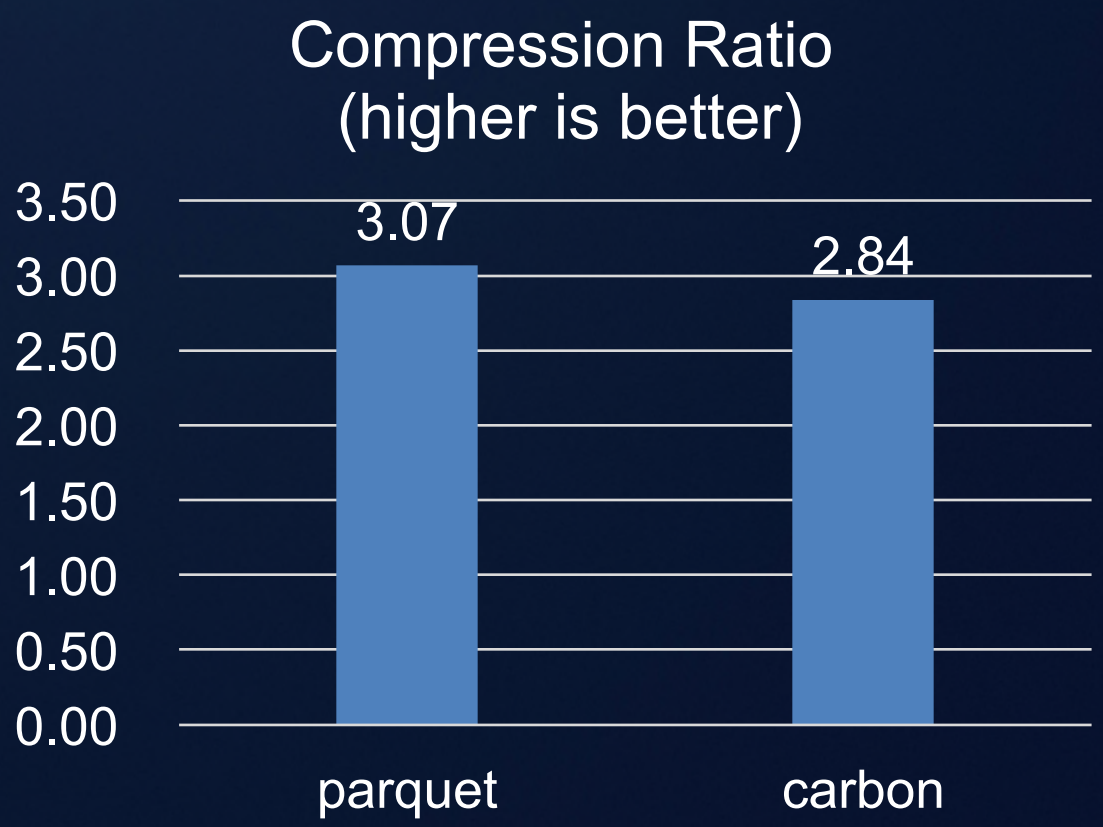
name	id	planhash	time	node	usr_time	ker_time	iowait_time	cpu(%)	read(mb)	write(mb)	recv(mb)	send(mb)	net_rate
Q12	Q12_0	0	195.65	10.19.89.49	1109.14	515.06	678.92	17.37	22257.06	5131.96	24416.92	23965.18	248.41
				10.19.89.53	1079.07	213.3	841.56	13.8	14249.69	5593.95	28160.61	28371.8	289.73
				10.19.89.51	1023.15	326.7	667.33	14.41	20962.34	6019.0	25788.4	25751.43	264.18
				avg	1070.45	351.69	729.27	15.19	19156.36	5581.64	26121.98	26029.47	267.44

Carbon资源使用(nmon)

name	id	planhash	time	node	usr_time	ker_time	iowait_time	cpu(%)	read(mb)	write(mb)	recv(mb)	send(mb)	net_rate
Q12	Q12_0	0	33.56	10.19.89.49	368.57	95.83	59.1	28.88	3166.83	2952.19	4734.1	5581.39	307.88
				10.19.89.53	443.57	97.19	15.38	33.63	2054.84	3878.42	8263.88	6940.31	453.87
				10.19.89.51	420.2	82.04	40.94	31.23	4796.36	3667.1	7500.64	7704.11	453.77
				avg	410.78	91.69	38.47	31.25	3339.34	3499.24	6832.87	6741.94	405.17

TPC-H: 入库速度和压缩率





过滤测试（真实场景）

过滤查询，c1到c10组合条件过滤

	Filter								响应时间(sec)		Task数	
Query	c1	c2	c3	c4	c5	c6	…	c10	Parquet	CarbonData	Parquet	CarbonData
Q1	✓	✓	✓	✓					6.4	1.3	55	5
Q2		✓	✓						65	1.3	804	5
Q3		✓		✓					71	5.2	804	9
Q5		✓			✓				64	4.7	804	9
Q4			✓	✓					67	2.7	804	161
Q6			✓			✓			62	3.7	804	161
Q7				✓		✓			63	21.85	804	588
Q8								✓	69	11.2	804	645

Carbon通过利用索引，减少Task任务下发，同时单Task内减少磁盘IO

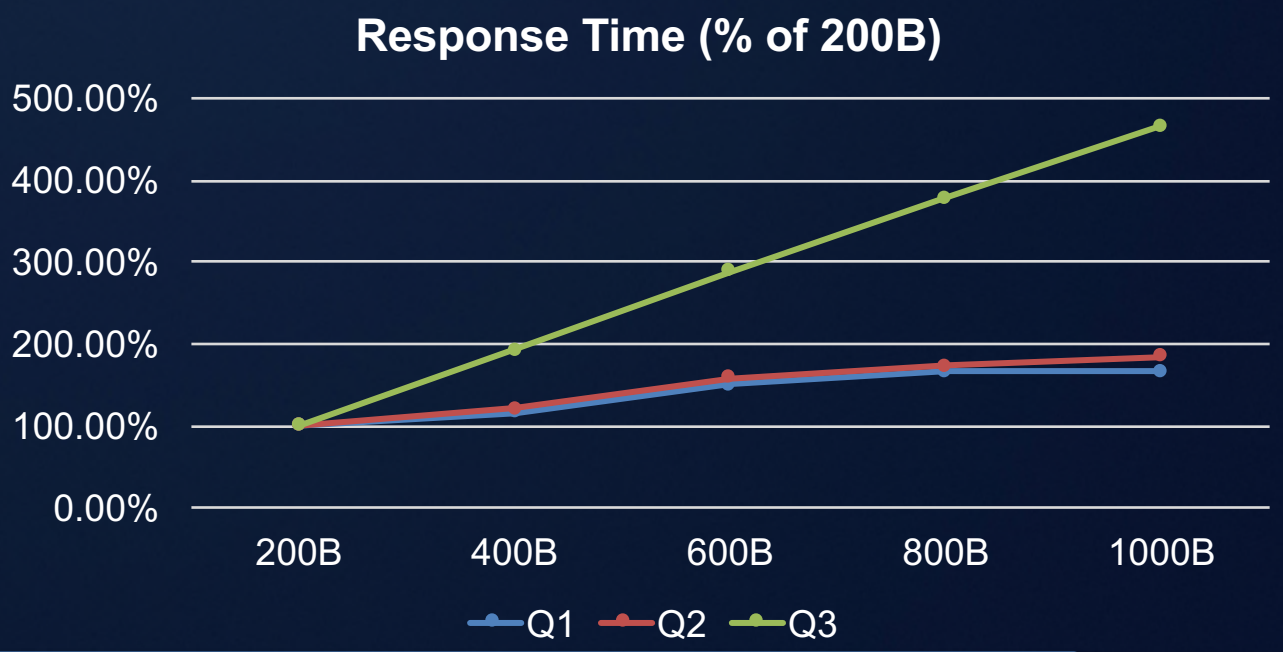
大数据集测试

数据：2000亿到1万亿数据 (中国某省份半年数据)

集群: 70 nodes, 1120 cores

查询:

Q1: filter (c1~c4), select *
Q2: filter (c10), select *
Q3: full scan aggregate



结果

- 过滤查询：5倍数据量，时间增加<1倍。
- 汇总分析：有效利用计算资源，可线性扩展

成功案例介绍

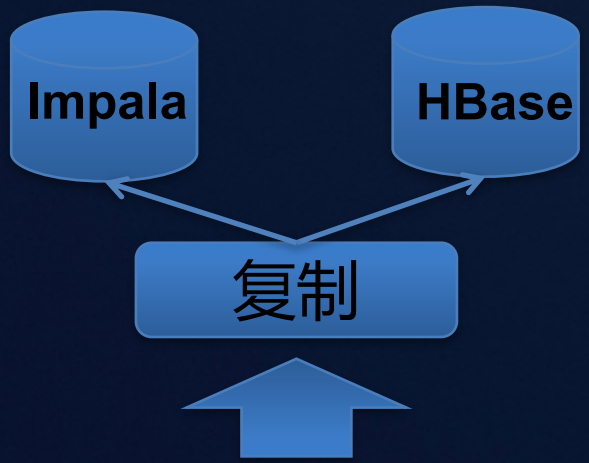
银行：使用CarbonData解决性能、规模、集群资源利用问题

业务场景：用户需要同时做复杂分析和多维查询，支持大规模数据

优化前

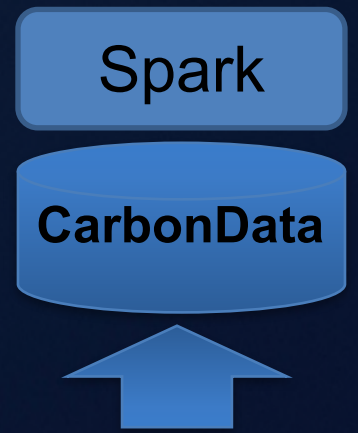
复杂分析：
1.**只适合Join/Orderby的业务**，不适合做点查；
2.**扩展性不足**，不能支撑上万亿数据查询

多维查询：
1、对6个关键维度查询，**需要建立4个二级索引，数据膨胀太大**，数据同步复杂
2、无法用YARN统一资源管理

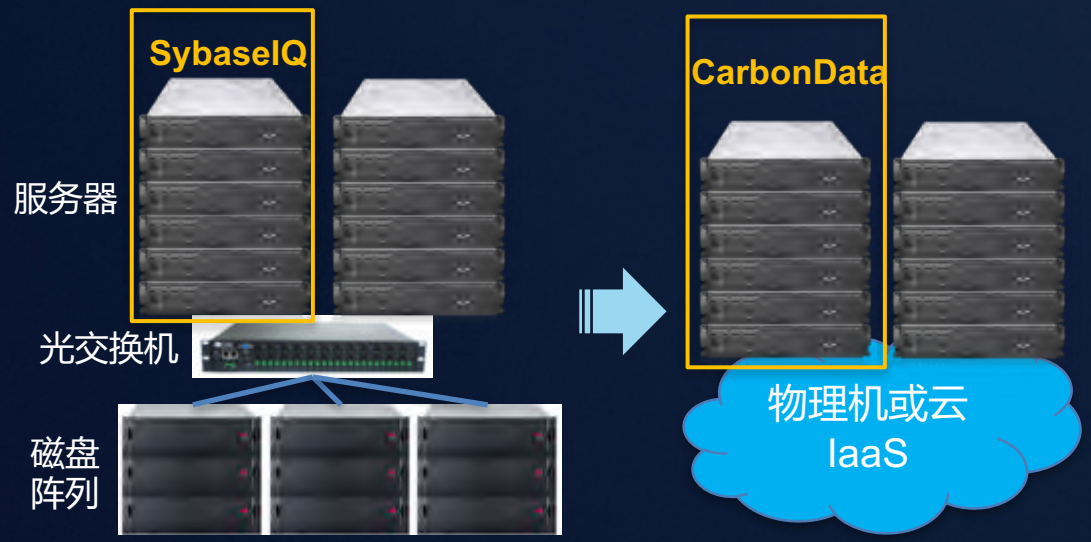


优化后

1. 一份数据完成复杂分析和多维点查，简化存储
2. 支持千亿数据量分析，秒级响应，高扩展性



电信：数仓分析场景，替换SybaseIQ和磁盘阵列



典型业务场景优化对比（忙时性能对比）：

业务	SQL特征	优化前	优化后	提升
业务1	多过滤（IO型）	10~20S	3S	5倍
业务2	高维计数TopN（全表扫描+用户数统计）	15S	7S	3倍
业务3	多过滤（IO型）	20S-30S	3S	8倍
业务3	高维聚合（千万级汇聚+全表扫描）	超时	20~30S	--

Carbon忙闲时性能对比（闲时并发15，忙时并发50）：

业务	闲时	忙时
业务1	2S	3~5S
业务2	10S	20~30S
业务3	3S	4~5S

原方案

- **成本高**：需要部署光交换机、磁盘阵列
- **无法云化**：数据存储在磁阵上，无法支持容器、VM部署
- **查询性能差**：在忙时查询性能无法满足要求，需要扩容磁阵

新方案

- **使用Carbon替换**：整个产品融合为一个HDFS存储，不依赖磁阵。
- **性能提升**：Carbon索引提升IO效率和分布式计算效率，**提升整体性能3~8倍**
- **规模**：迁移15个业务，1000+张表；每15分钟、1小时、1天加载一次数据

Hulu公司：上线CarbonData，提升OLAP业务查询效率

5 Carbondata在hulu的优化

hulu的部分广告数据的查询以宽表查询为主。数据的特点是：

- filter过滤掉95%~98%的数据，只有2%~5%的数据参与aggregation
- filter索引一般命中5~10列，大部分列distinct值小于100。
- select的列有100+，列式存储组装所花费的时间长。

在上述场景下，由于carbondata比parquet和ORC拥有更细粒度的索引，支持行式存储，我们采用carbondata(版本0.2.0)格式以加快查询速度。carbondata的查询引擎采用sparkSQL（目前carbondata仅支持sparkSQL），在此基础上进一步优化。

不同OLAP引擎下，carbondata的实验结果对比如表3所示：

查询条件	查询时间(s)
Presto	290
Impala	76
Optimized Impala	36
carbondata GC + Speculation	25
carbondata GC + Speculation + shuffle + 10tasks + 256M Block + MDK调优	21

CarbonData 在Hulu OLAP引擎的优化与应用：
http://mp.weixin.qq.com/s/XFjUdVvbD-RMWAvJID0zmA

总结：CarbonData的优秀DNA

查询：

- 两级索引，减少IO：**适合ad-hoc查询**，任意维度组合查询场景
- 延迟解码，向量化处理：适合全表扫描、**汇总分析**场景

数据管理：

- 增量入库，多级排序可调：**由用户权衡**入库时间和查询性能
- 增量更新，批量合并：支持**快速更新**事实表或维表，闲时做Compaction合并

大规模：

- 计算与存储分离：支持从GB到PB大规模数据，**万亿数据秒级响应**

部署：

- Hadoop Native格式：与大数据生态无缝集成，**利用已有Hadoop集群资产**

欢迎参与Apache CarbonData社区

- website: http://carbondata.apache.org
- Code: https://github.com/apache/carbondata
- JIRA: https://issues.apache.org/jira/browse/CARBONDATA
- Mail list: dev@carbondata.apache.org, user@carbondata.apache.org

- 欢迎在Maillist上提问，共同探讨和开发CarbonData新特性