基于CarbonData构建万亿级数据仓库

李昆 2018-12

Agenda

- CarbonData的适用场景
- CarbonData的4个使用层次
- Demo

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

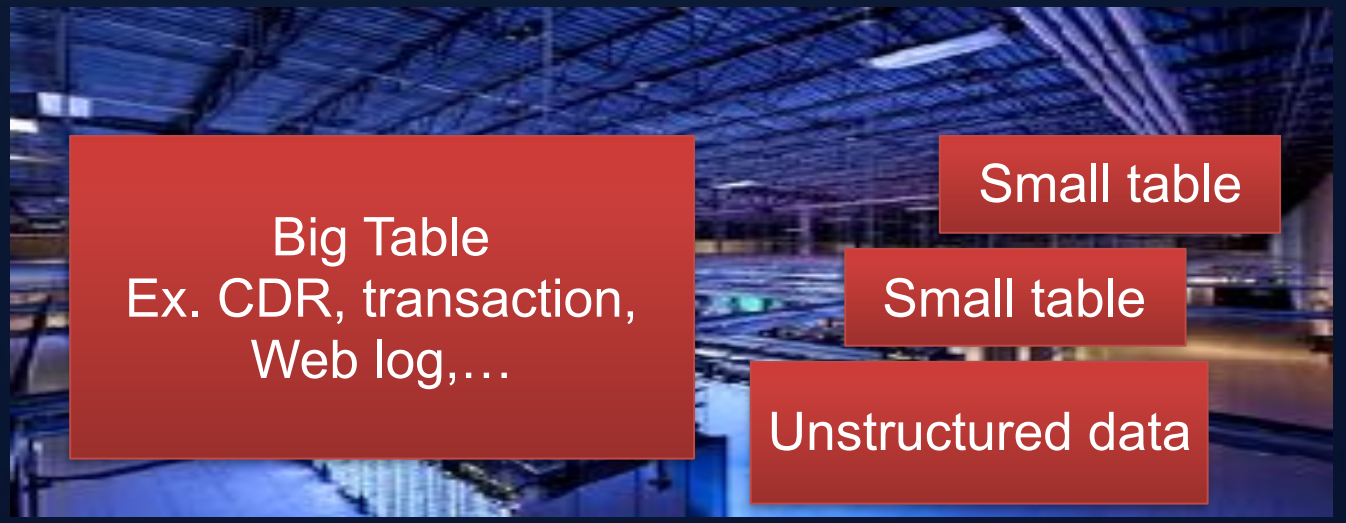
OLAP & Ad-hoc

Batch processing

Machine learning

Realtime Analytics

data



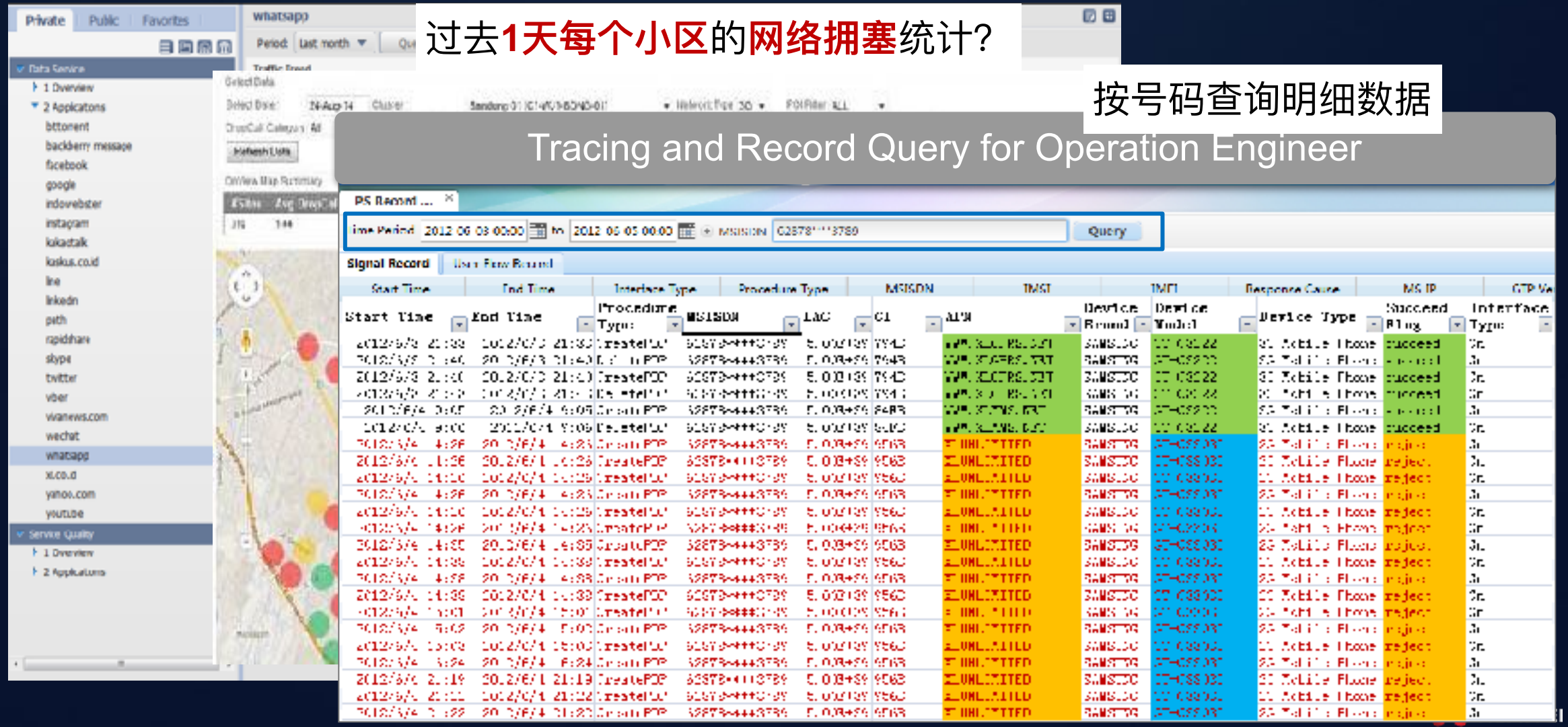
数据应用举例

过去**1天**使用**Whatapp应用**的**终端按流量排名**情况?

过去**1天每个小区**的**网络拥塞**统计?

按号码查询明细数据

Tracing and Record Query for Operation Engineer



实时应用　明细数据分析　交互式分析

数据产品A　数据产品B　数据产品C　数据产品D　数据产品E　数据产品F

标签画像库　关系库　XX业务库

主题库

流处理

全量历史库

流水线开发和调度

ODS（临时库）

数据源
(生产系统)

HUAWEI

典型诉求

核心诉求：

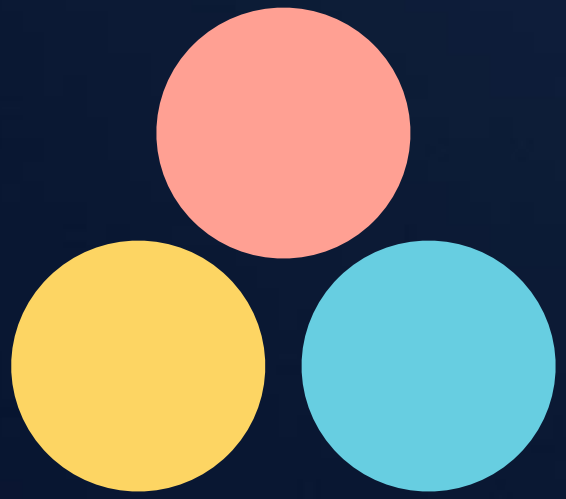
1. 能稳定承载海量数据，PB级
2. 支持多种workload：增量，更新，编程，SQL，…
3. 性能：快速入库，快速分析
4. 数据源对接：开源生态，传统DB，…

核心诉求：

1. 高性能：复杂SQL优化
2. 高性能：特殊索引，全文，图，bitmap，…
3. 易开发，标准SQL

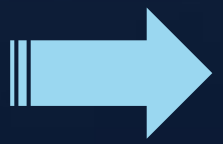
CarbonData目标：
一份数据满足多种业务需求，与大数据生态无缝集成

Multi-dimensional OLAP Query

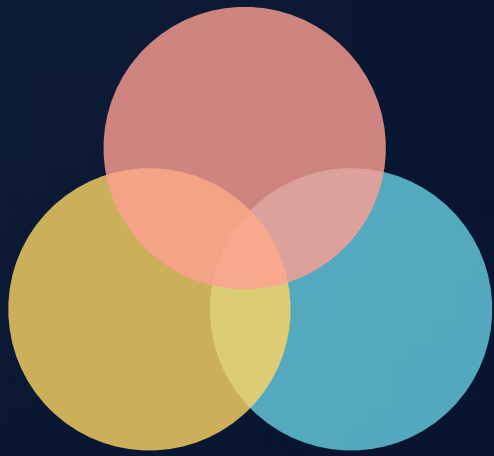


Full Scan Query

Small Scan Query



CarbonData: Unified Storage



一份数据满足多种分析场景

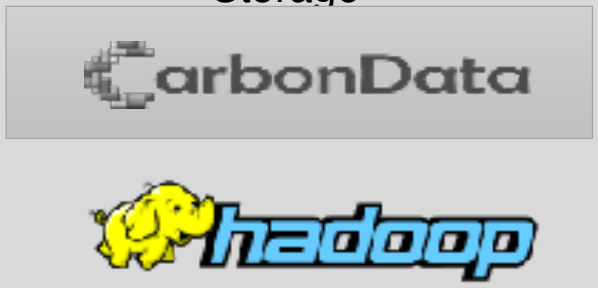
详单过滤，海量数仓，数据集市，…

Apache CarbonData

- Apache Incubator Project since June, 2016
- Apache releases
  - >10 stable releases
  - Coming release**: 1.5.0, 2018-09**
- Contributors:

Compute

Storage



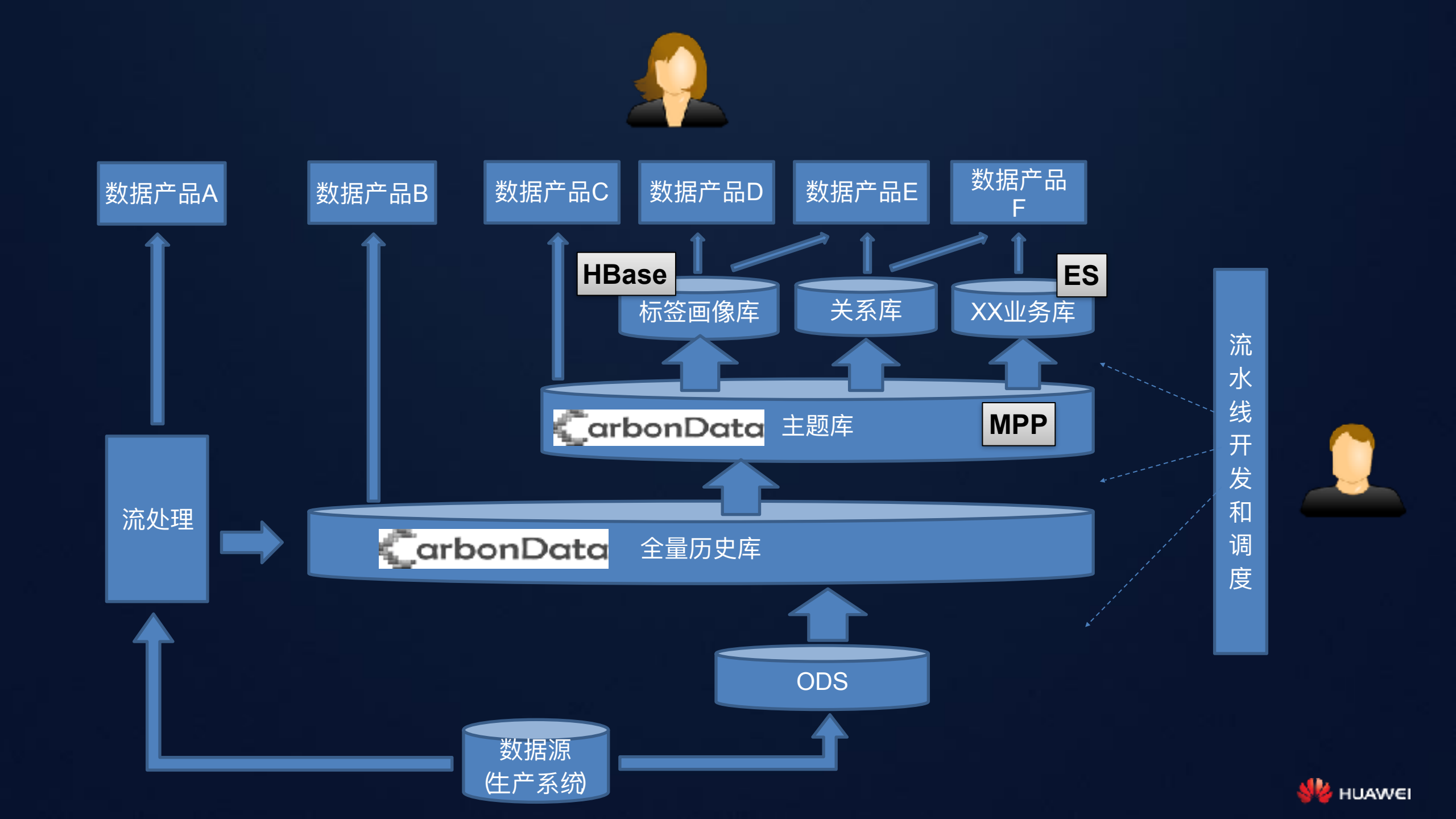
方案对比


不支持或很弱
中规中矩
擅长
HDFS +CB
HDFS +PQ
ES
HBase
Kudu
MPP
流写入
批量写入
批量更新
实时更新
明细数据查询
批量计算
机器学习
海量
多租
批量写入
汇总统计
特殊索引
标准SQL
机器学习
全量历史库
主题库/业务库

CarbonData目标场景

- 以Hadoop为基础的数仓：海量数据，多租户，多计算引擎访问同一份数据
- TB~PB级明细数据的多维过滤分析（索引）、OLAP分析（预汇聚）
- 分钟级追加（批量或流式）
- 批量更新、删除，拉链表

- 非目标场景：
  - OLTP：实时追加、更新、删除
  - 毫秒级点查
  - 毫秒级OLAP


数据产品A
数据产品B
数据产品C
数据产品D
数据产品E
数据产品F
HBase
ES
标签画像库
关系库
XX业务库
CarbonData 主题库
MPP
流处理
CarbonData 全量历史库
ODS
数据源
(生产系统)
流水线开发和调度
HUAWEI

CarbonData的4个使用层次

CarbonData的4个使用层次

1. 基本能力：
   - 列存+事务管理+更新 => 性能等同于Parquet + 多引擎共用一份数据
2. 使用索引 => 提升海量数据的过滤性能（类搜索）
3. 使用预汇聚 => 提升复杂汇聚分析性能
4. 使用流式入库 => 提升实时性，同时避免小文件

CarbonData的4个使用层次

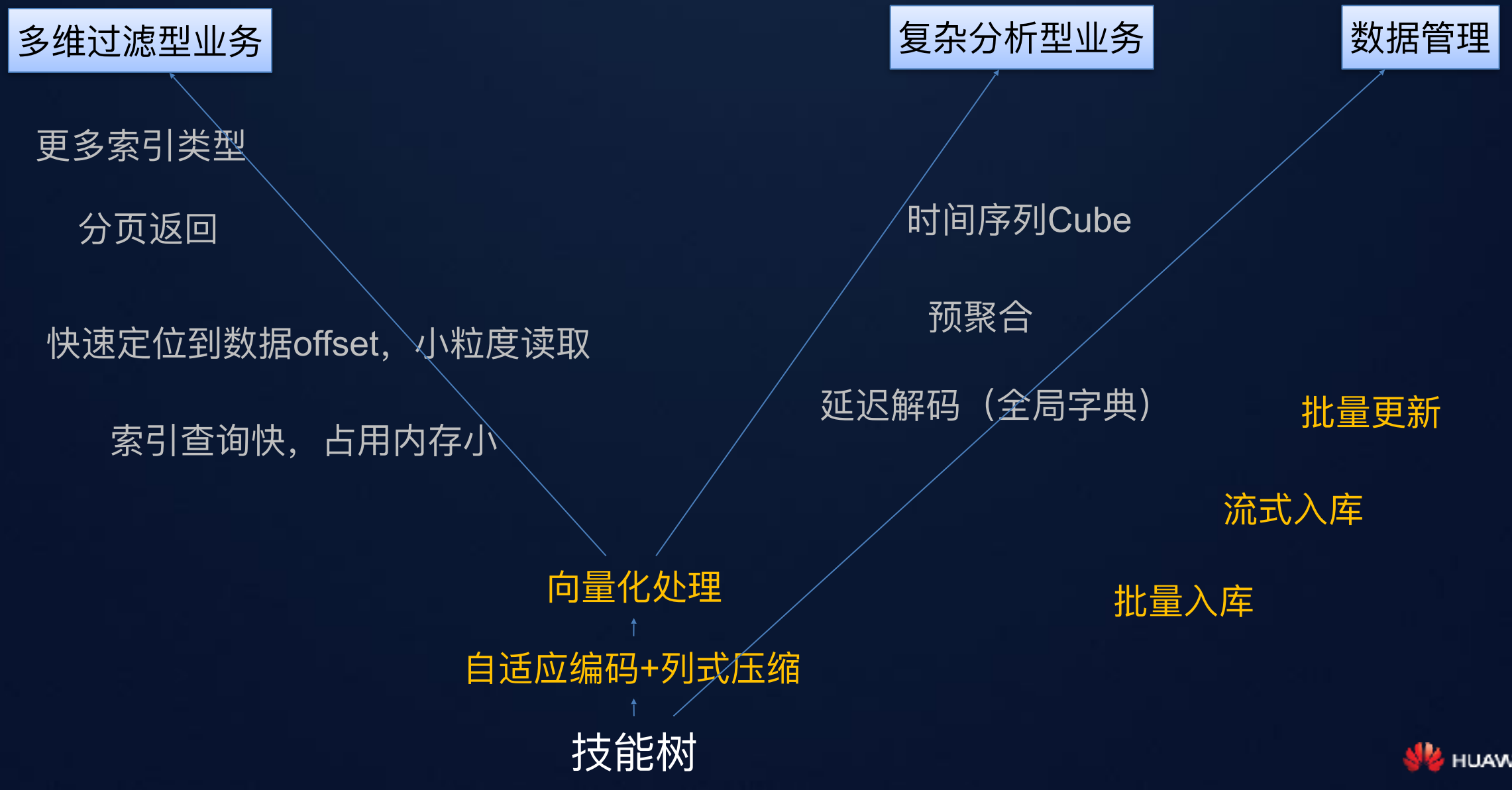
1. 基本能力：
   - 列存+事务管理+更新 => **性能等同于Parquet** + 多引擎共用一份数据

2. 使用索引 => 提升海量数据的过滤性能（类搜索）

3. 使用预汇聚 => 提升复杂汇聚分析性能

4. 使用流式入库 => 提升实时性，同时避免小文件

CarbonData技能树



创建表格和入库

- SQL

```
CREATE TABLE tablename (name string, age int)
STORED AS carbondata
TBLPROPERTIES ('SORT_COLUMNS'='name',
'SORT_SCOPE'='..')

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()
```

- Hive语法
- SORT_COLUMNS：排序的列（提升过滤性能）
- SORT_SCOPE：排序范围（控制入库速度）
  - NO_SORT
  - LOCAL_SORT
  - BATCH_SORT
  - GLOBAL_SORT
- 兼容DataFrame API

Carbon数据文件格式

列存格式 + 多级索引，可跳读或顺序扫描

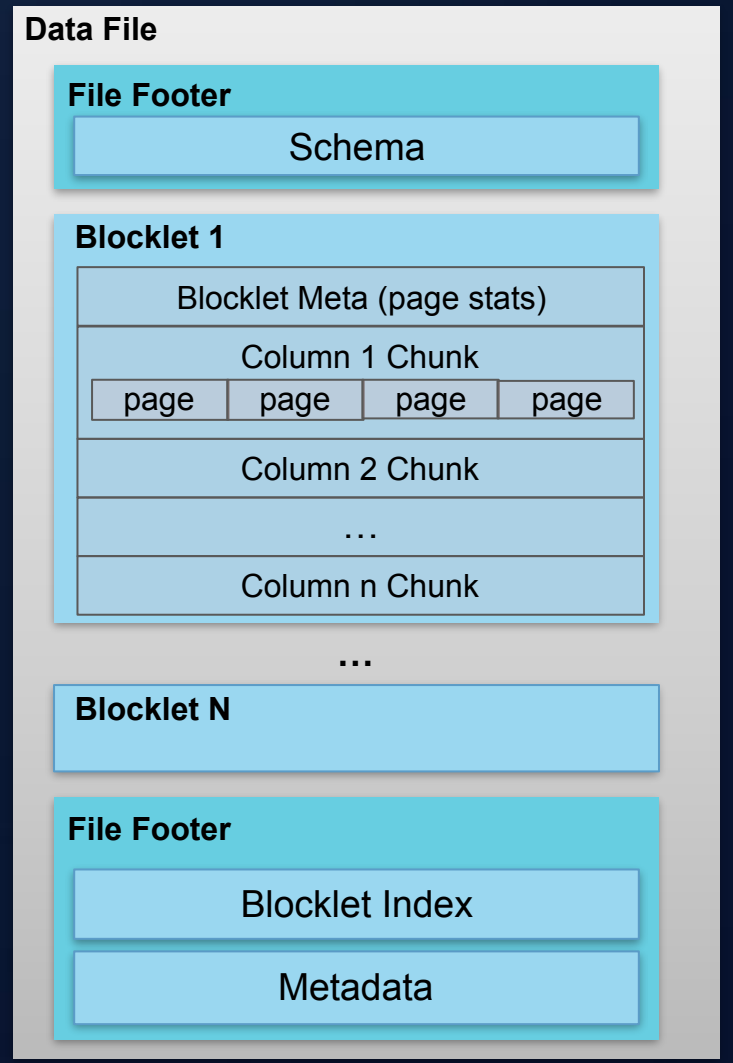
•Page：包含32000行数据（一列），是一个压缩单元，可单独压缩解压缩

• Column Chunk: 包含多个Page，是一次IO读的单元，解码时一次读取一个Column Chunk

• Blocklet: 包含所有列的Column Chunk (默认64MB)

•Header：包含文件版本号、Schema

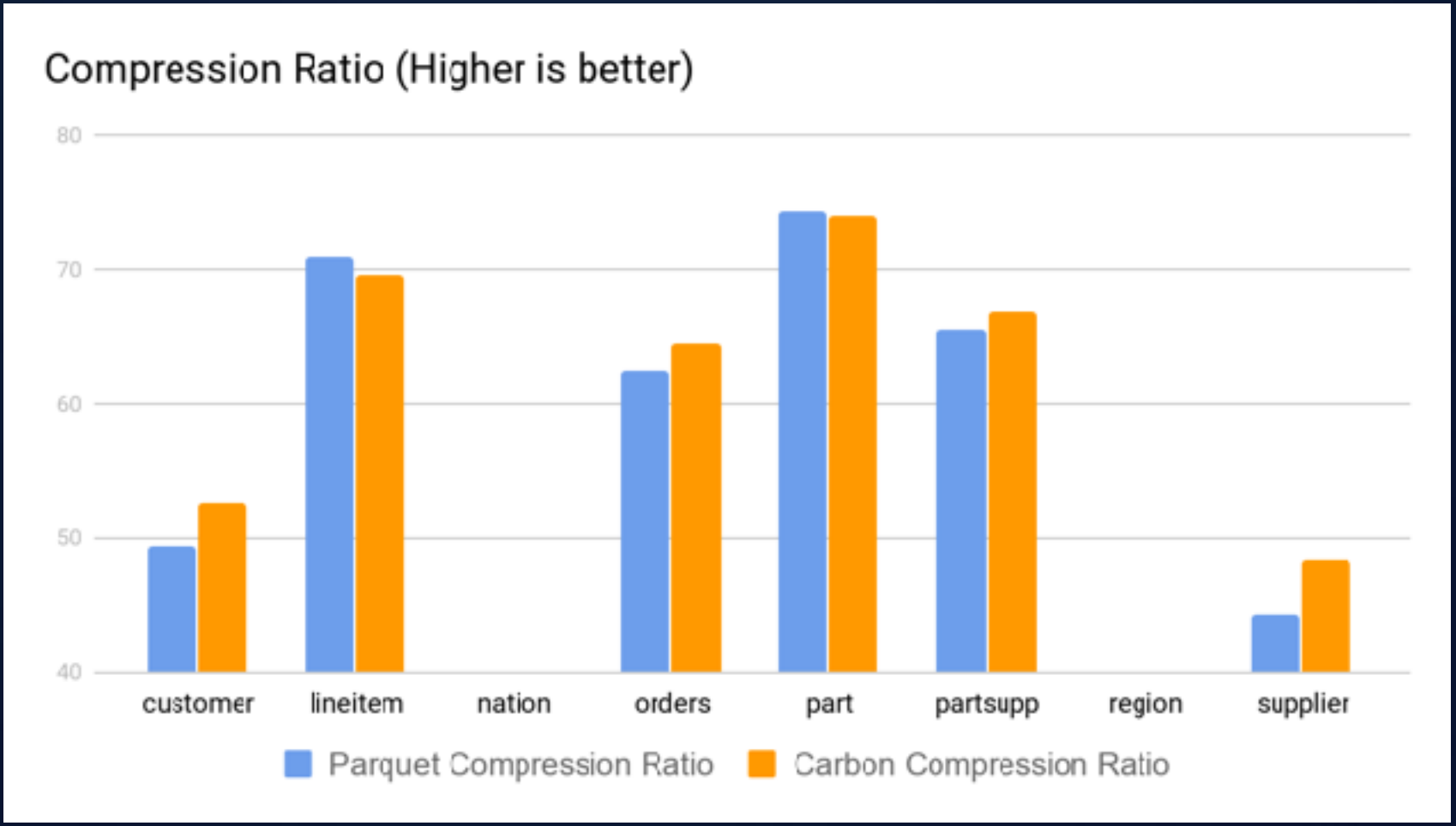
•Footer: 包含索引和其他Meta



TPC-H: 压缩率对比

Carbon NO_SORT表
Parquet 非分区表

500GB
3节点集群

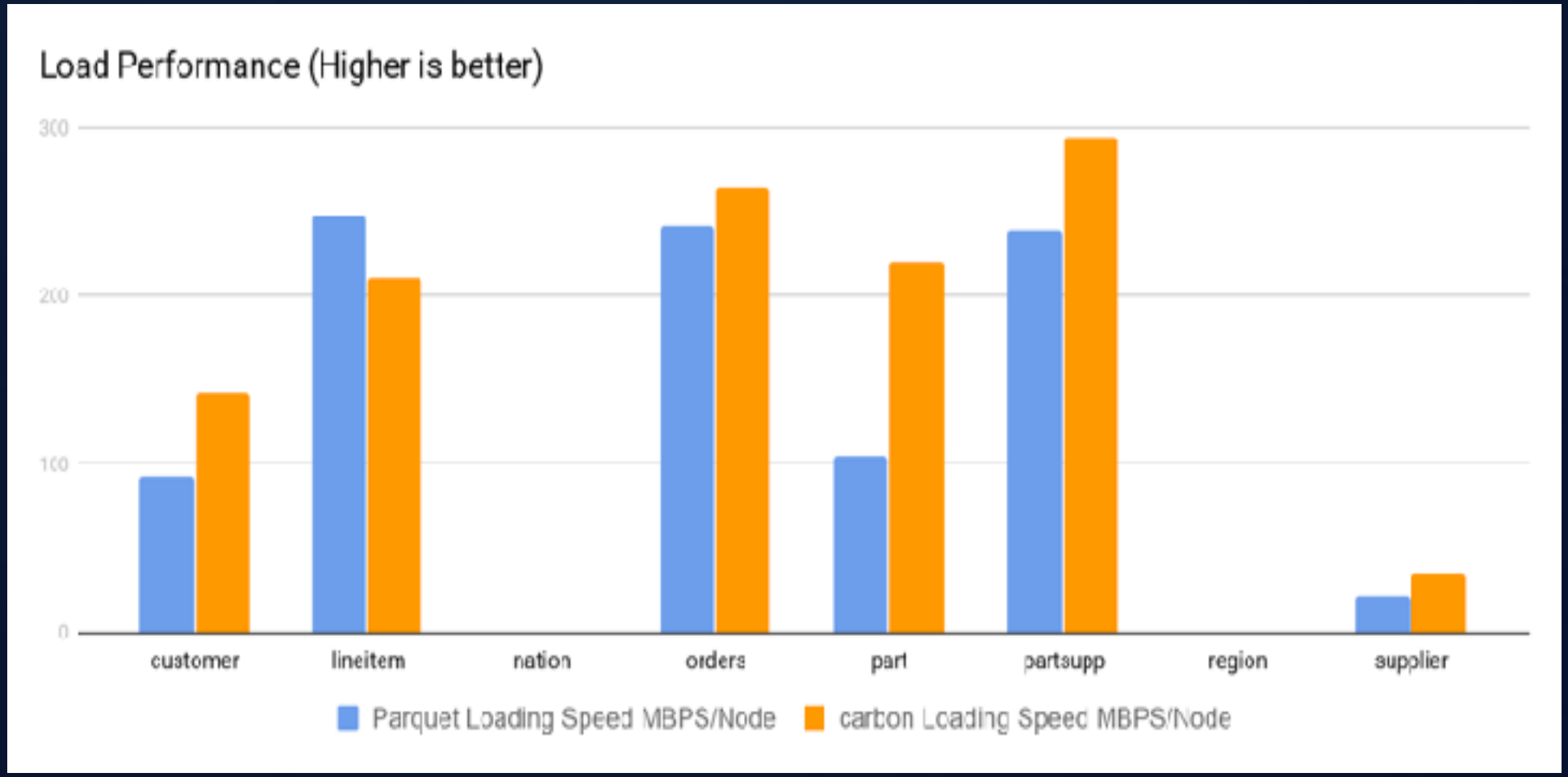


压缩率 =（原始大小-入库后大小）/ 入库后大小

TPC-H: 入库性能对比

Carbon NO_SORT表
Parquet 非分区表

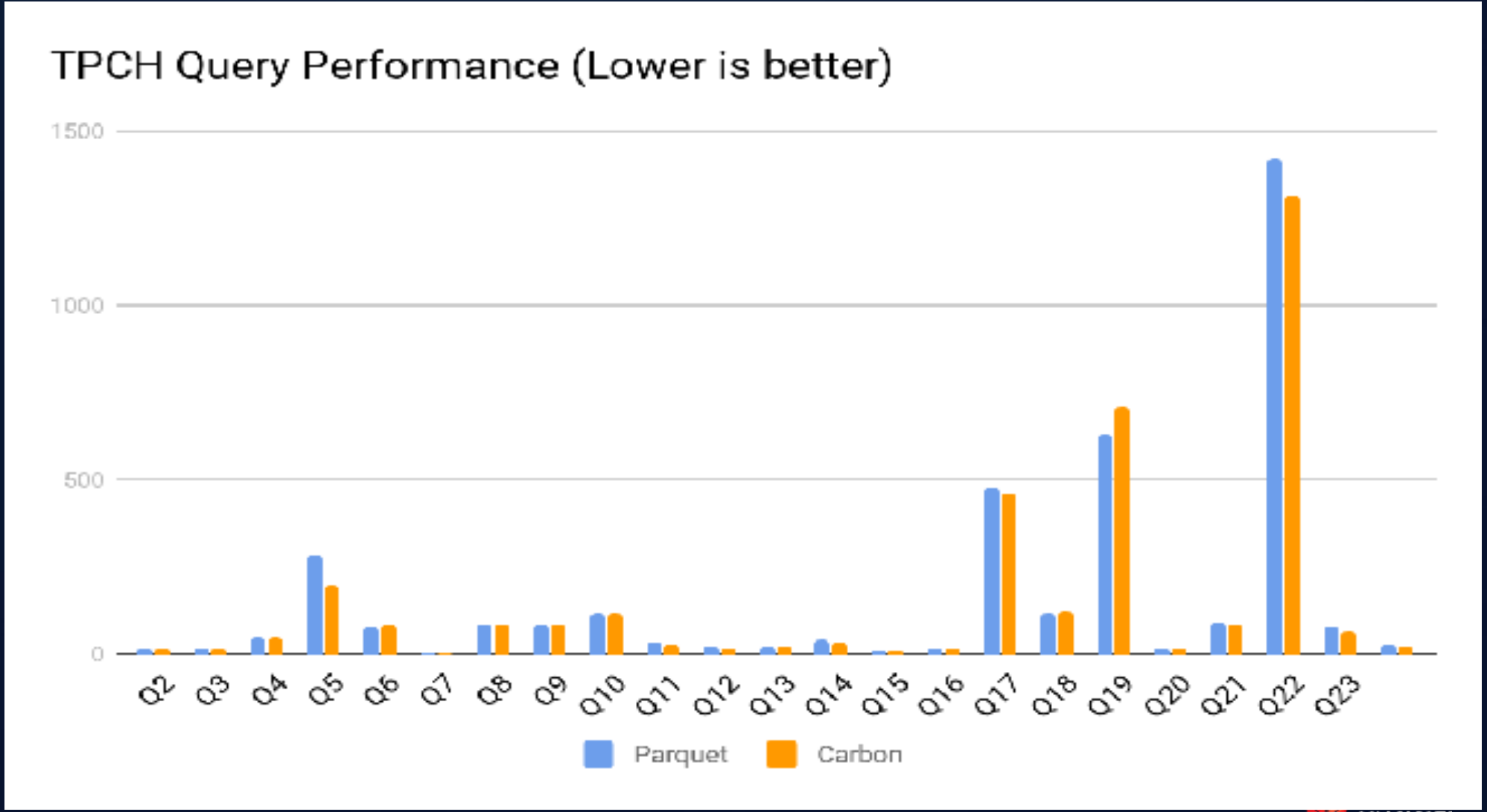
500GB
3节点集群



TPC-H查询性能对比

Carbon NO_SORT表
Parquet 非分区表

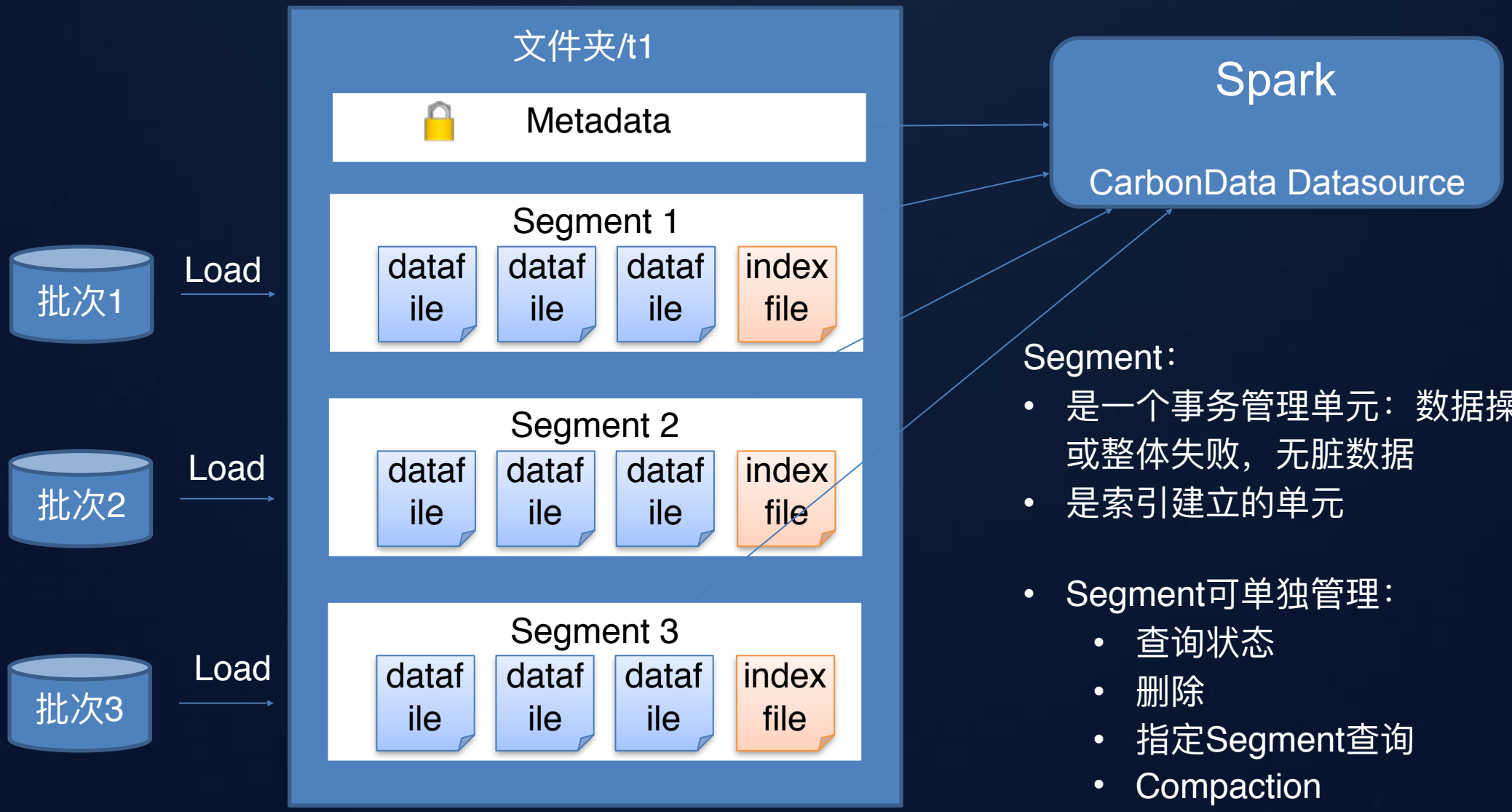
500GB
3节点集群



CarbonData的4个使用层次

1. 基本能力：
   - 列存+事务管理+更新 => 性能等同于Parquet + **多引擎共用一份数据**
2. 使用索引 => 提升海量数据的过滤性能（类搜索）
3. 使用预汇聚 => 提升复杂汇聚分析性能
4. 使用流式入库 => 提升实时性，同时避免小文件

Carbon表格数据组织



Segment：

- 是一个事务管理单元：数据操作整体成功或整体失败，无脏数据
- 是索引建立的单元

- Segment可单独管理：
  - 查询状态
  - 删除
  - 指定Segment查询
  - Compaction

Segment管理

- 查询表格的segments元数据

```
SHOW HISTORY SEGMENTS FOR TABLE t1 LIMIT 100
```

```
0: jdbc:hive2://carbon1:10000> show segments for table lineitem;
+------------------+----------+--------------------------+--------------------------+-----------+-------------+-----------+------------+--+
| SegmentSequenceId |  Status  |     Load Start Time      |      Load End Time       | Merged To | File Format | Data Size | Index Size |
+------------------+----------+--------------------------+--------------------------+-----------+-------------+-----------+------------+--+
| 3                | Success  | 2018-09-24 23:48:11.789  | 2018-09-24 23:48:12.466  | NA        | COLUMNAR_V3 | 430.94KB  | 2.42KB     |
| 1                | Success  | 2018-09-24 21:57:51.74   | 2018-09-24 22:01:25.968  | NA        | COLUMNAR_V3 | 2.02GB    | 26.48KB    |
+------------------+----------+--------------------------+--------------------------+-----------+-------------+-----------+------------+--+
2 rows selected (0.046 seconds)
```

- 删除segments （使用segment ID或入库时间）

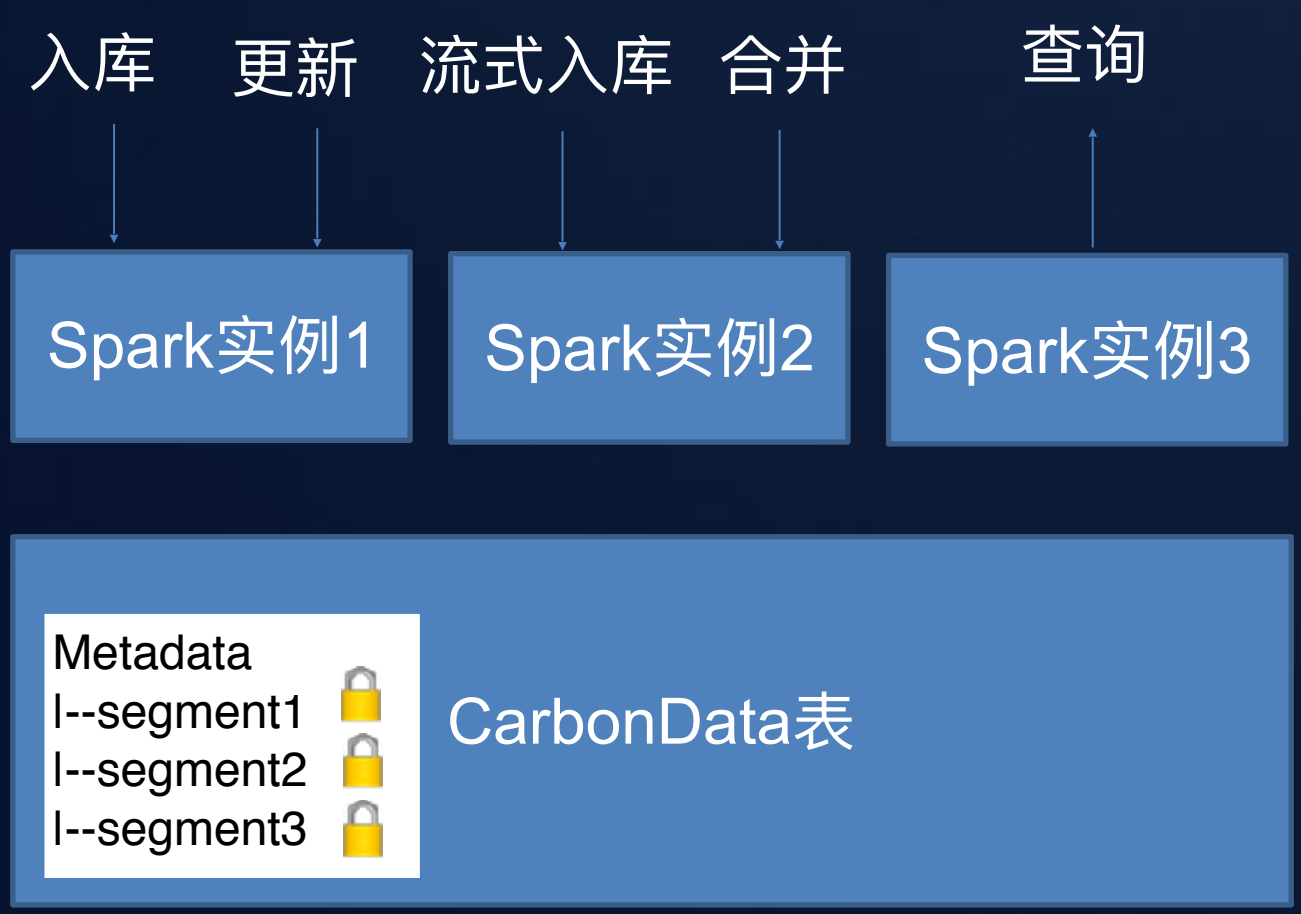
```
DELETE FROM TABLE t1 WHERE SEGMENT.ID IN (98, 97, 30)
```

```
DELETE FROM TABLE t1 WHERE SEGMENT.STARTTIME BEFORE '2017-06-01 12:05:06'
```

- 查询指定segment

```
SET carbon.input.segments.default.t1 = 100, 99
```

多Spark实例共用一份数据

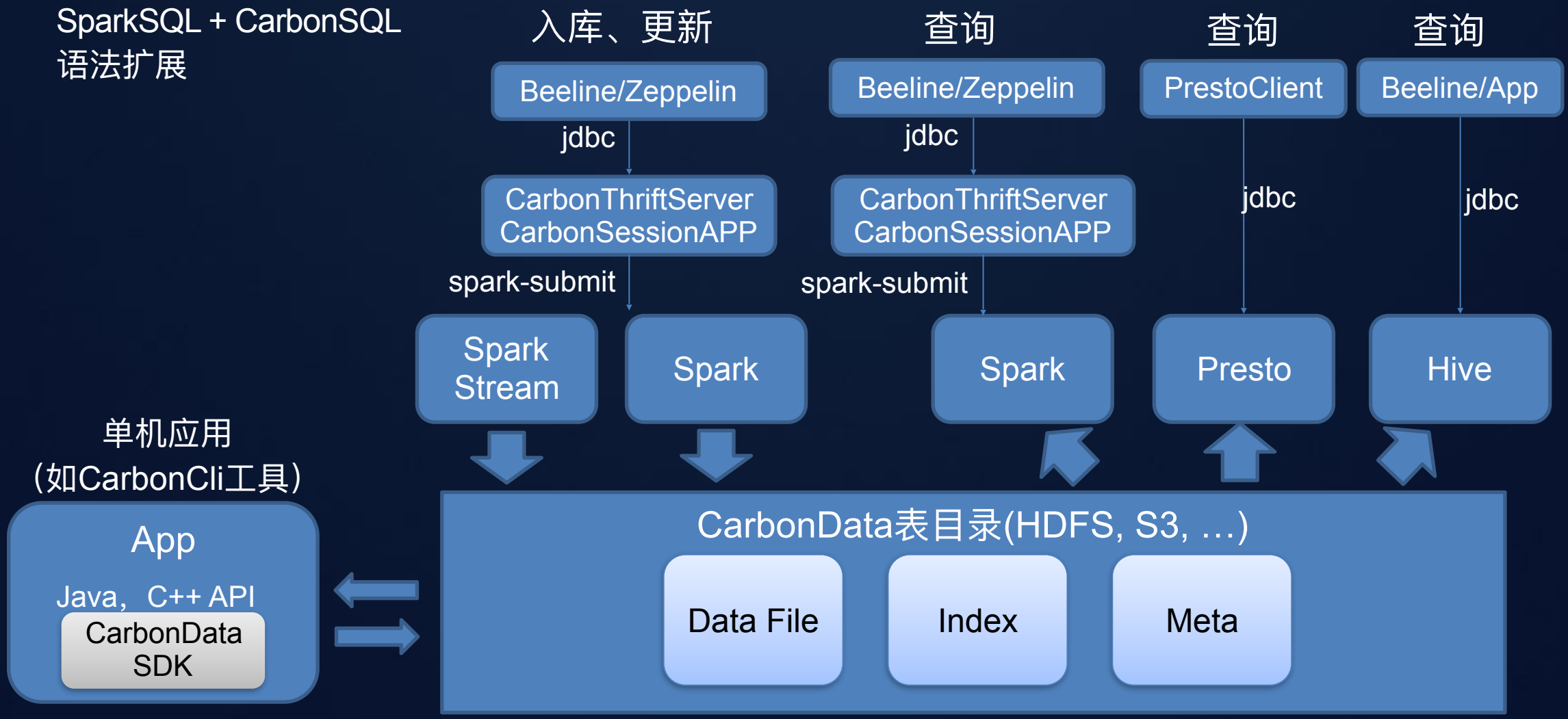


1. Segment锁
   - 支持并发入库、合并
   - 支持串行更新，不支持并发更新

2. 多个Spark实例并发操作同一份数据，有事务保证
   - 入库整体成功或整体失败
   - 边入库查询边查询
   - 边入库边更新
   - 边更新边查询
   - 边合并边查询
   - 入库+构建二级索引
   - 入库+构建预汇聚表

多引擎共用一份数据

SparkSQL + CarbonSQL
语法扩展



数据更新和删除

更新table1的revenue列

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

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

用join结果更新第一个表（拉链表场景）

```
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'
```

删除table1中的某些记录

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

~~123, abc~~
456, jkd

分区表

创建分区表

```
CREATE TABLE sale(id string, quantity int ...)
PARTITIONED BY (country  string, state string)
STORED AS carbondata
```

入库静态分区表

```
LOAD DATA LOCAL INPATH 'folder path' INTO TABLE sale PARTITION (country = 'US',
state = 'CA')
```

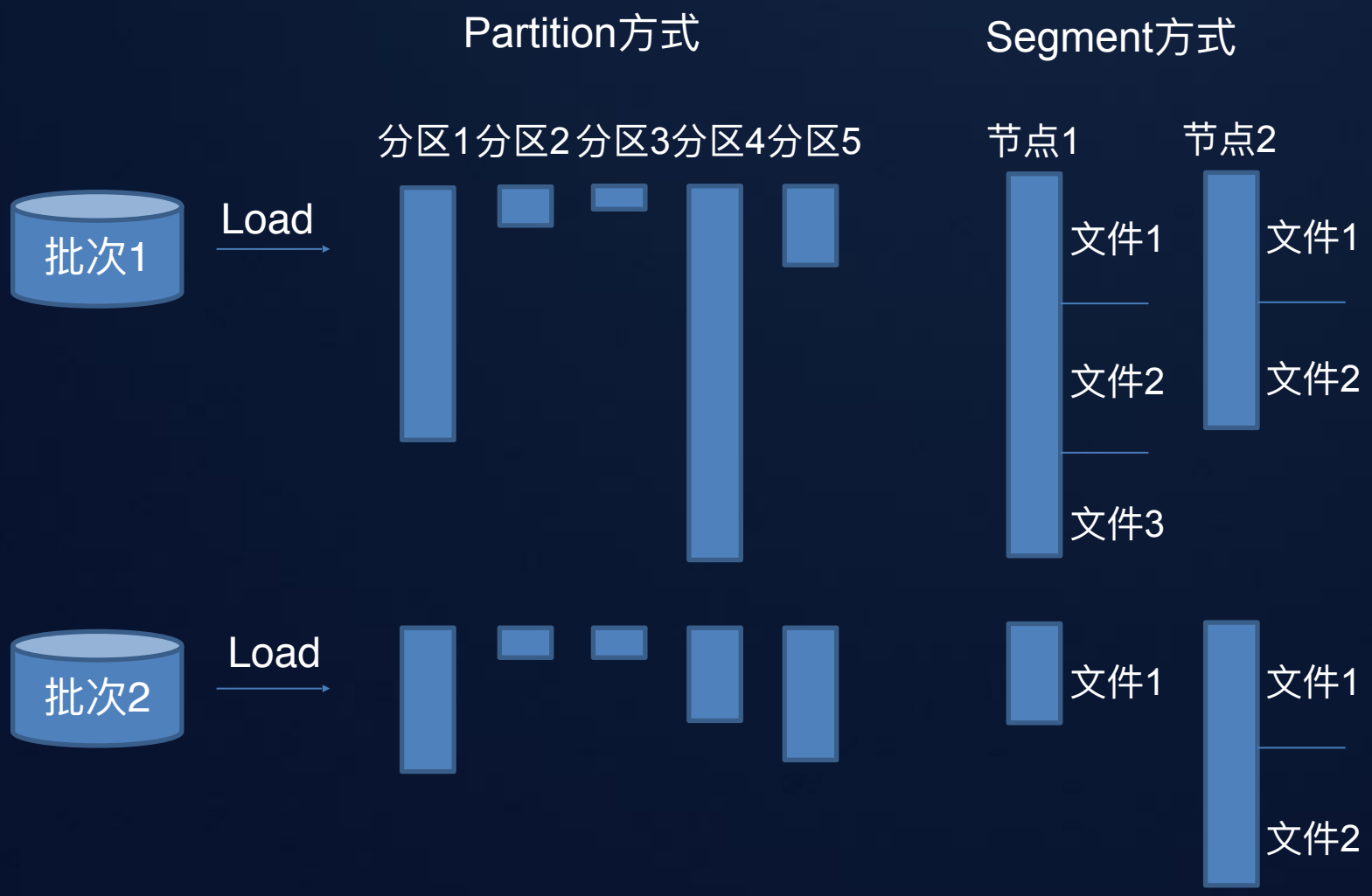
```
INSERT INTO TABLE sale PARTITION (country = 'US', state = 'AL')
SELECT <columns list excluding partition columns> FROM another_sale
```

入库动态分区表

```
LOAD DATA LOCAL INPATH 'folder path' INTO TABLE
```

```
INSERT INTO TABLE sale SELECT <columns list including partition columns> FROM
another_sale
```

Partition与Segment对比



	分区	Segment
入库	慢 shuffle	快 不需shuffle
均衡性	可能有倾斜，导致小文件	较均衡，按固定大小分割文件
扫描性能	可能由于小文件导致扫描差	文件大小固定，有利于扫描
数据聚集效果	好，直接定位分区	稍差 在多个节点上分布
过滤性能	分区列好，非分区列差	较好（主索引+二级索引）

CarbonData的4个使用层次

1. 基本能力：
   - 列存+事务管理+更新 => 性能等同于Parquet + 多引擎共用一份数据
2. **使用索引 => 提升海量数据的过滤性能（类搜索）**
3. 使用预汇聚 => 提升复杂汇聚分析性能
4. 使用流式入库 => 提升实时性，同时避免小文件

创建表格和入库

- SQL

```
CREATE TABLE tablename (name string, age int)
STORED AS carbondata
TBLPROPERTIES ('SORT_COLUMNS'='name',
'SORT_SCOPE'='..')

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()
```

- Hive语法
- SORT_COLUMNS：排序的列（提升过滤性能）
- SORT_SCOPE：排序范围（控制入库速度）
  - NO_SORT
  - LOCAL_SORT
  - BATCH_SORT
  - GLOBAL_SORT
- 兼容DataFrame API

DataMap管理

```
CREATE DATAMAP [IF NOT EXISTS] datamap_name [ON TABLE main_table]
  USING "datamap_provider"
  [WITH DEFERRED REBUILD]
  DMPROPERTIES ('key'='value', ...)
  AS
    SELECT statement
```

```
DROP DATAMAP [IF NOT EXISTS] datamap_name [ON TABLE main_table]
```

```
SHOP DATAMAP ON TABLE main_table
```

二级索引: 加速过滤查询

- bloomfilter
- Lucene
- Your idea

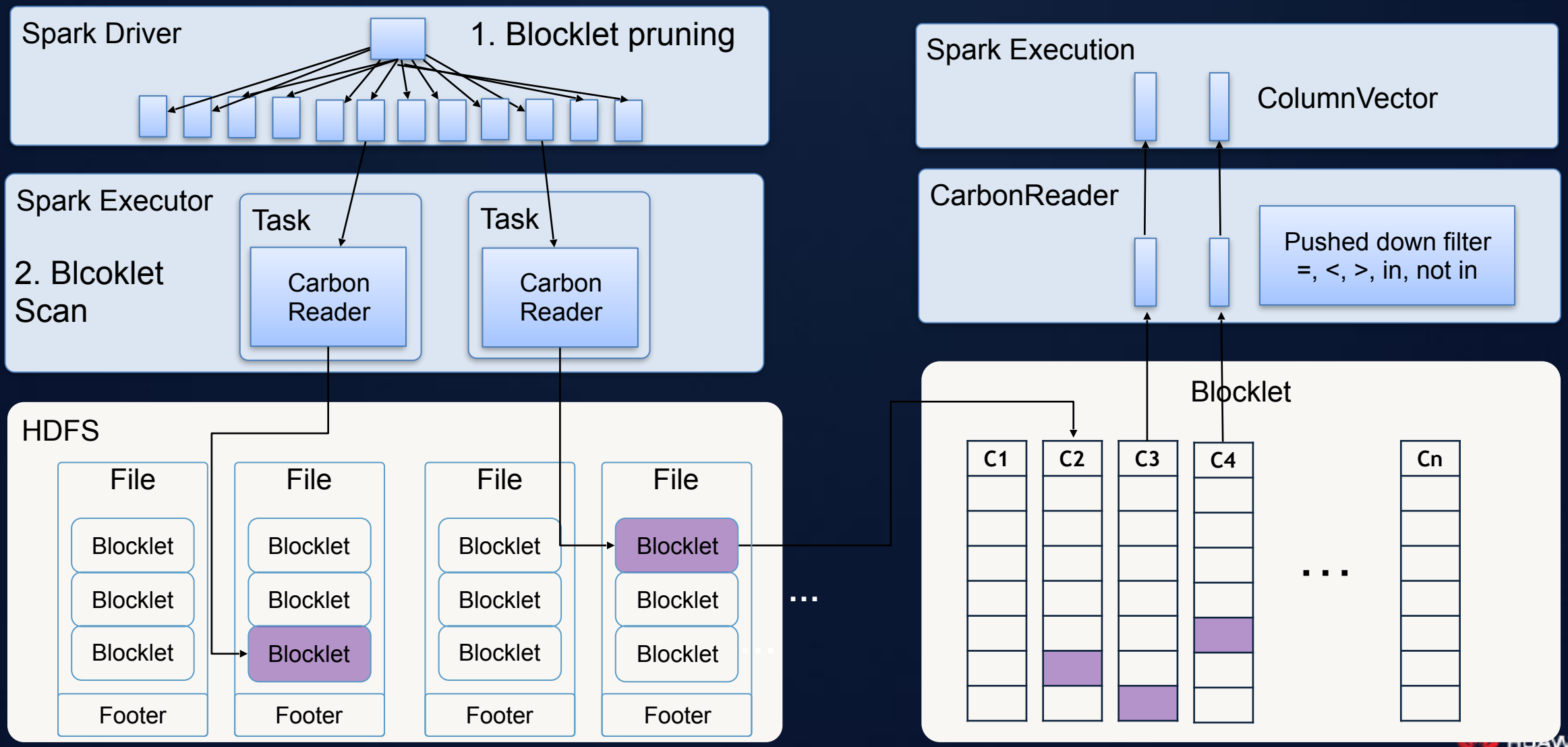
BloomFilter DataMap

```
CREATE DATAMAP [IF NOT EXISTS] datamap_name
   ON TABLE main_table
   USING 'bloomfilter’
DMPROPERTIES (
  ‘INDEX_COLUMNS'='city, name’,
  'BLOOM_SIZE'='640000’,
  'BLOOM_FPP'='0.00001’,
  'BLOOM_COMPRESS'='true')
```

- 适合做高基数列过滤，例如：手机号码，车牌等ID
- 入库时自动建索引或手动重建

CarbonData查询流程：索引+向量化

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



Demo1

- Q1: 在第1，2个排序列上过滤
  - select count(*) from lineitem where l_shipdate>'1992-05-03' and l_shipdate<'1992-05-05' and l_returnflag='R';
- Q2: 在第4个排序列上过滤
  - select count(*) from lineitem where l_receiptdate='1992-05-03';
- Q3: 在非排序列上过滤
  - select * from lineitem where l_partkey=123456;
- Q4: 扫描l_suppkey列，count行数
  - select count(l_suppkey) from lineitem;

Query	carbon	carbon_ls	parquet	parquet_par
Q1	45	0.3	46	0.6
Q2	27	0.6	27	38
Q3	2.9	3.3	17	17
Q4	3	3.3	6.6	6.6

Minor Compaction：基于Segment个数进行合并

设置合并策略

```
SET carbon.compaction.level.threshold = 4,2
```

手动触发Compaction

```
ALTER TABLE table_name COMPACT 'MINOR'
```

自动触发Compaction

```
SET carbon.enable.auto.load.merge = true
```

```
0: jdbc:hive2://carbon1:10000> show segments for table lineitem;
+------------------+-----------+-------------------------+-------------------------+-----------+-------------+-----------+------------+--+
| SegmentSequenceId |  Status   |     Load Start Time     |     Load End Time       | Merged To | File Format | Data Size | Index Size |
+------------------+-----------+-------------------------+-------------------------+-----------+-------------+-----------+------------+--+
| 11               | Success   | 2018-09-25 00:12:38.167 | 2018-09-25 00:12:38.789 | NA        | COLUMNAR_V3 | 430.94KB  | 2.43KB     |
| 10               | Success   | 2018-09-25 00:12:36.041 | 2018-09-25 00:12:36.634 | NA        | COLUMNAR_V3 | 430.94KB  | 2.43KB     |
| 9                | Compacted | 2018-09-25 00:11:54.541 | 2018-09-25 00:11:55.143 | 6.1       | COLUMNAR_V3 | 430.94KB  | 2.38KB     |
| 8                | Compacted | 2018-09-25 00:11:44.913 | 2018-09-25 00:11:45.554 | 6.1       | COLUMNAR_V3 | 430.94KB  | 2.42KB     |
| 7                | Compacted | 2018-09-25 00:11:38.273 | 2018-09-25 00:11:38.982 | 6.1       | COLUMNAR_V3 | 430.94KB  | 2.42KB     |
| 6.1              | Success   | 2018-09-25 00:11:54.541 | 2018-09-25 00:11:55.878 | NA        | COLUMNAR_V3 | 668.41KB  | 2.43KB     |
| 6                | Compacted | 2018-09-25 00:11:28.183 | 2018-09-25 00:11:28.766 | 6.1       | COLUMNAR_V3 | 430.94KB  | 2.42KB     |
| 5                | Compacted | 2018-09-25 00:09:33.949 | 2018-09-25 00:09:34.594 | 1.1       | COLUMNAR_V3 | 430.94KB  | 2.38KB     |
| 4                | Compacted | 2018-09-25 00:09:31.689 | 2018-09-25 00:09:32.28  | 1.1       | COLUMNAR_V3 | 430.94KB  | 2.42KB     |
| 3                | Compacted | 2018-09-25 00:09:15.968 | 2018-09-25 00:09:16.665 | 1.1       | COLUMNAR_V3 | 430.94KB  | 2.42KB     |
| 1.1              | Success   | 2018-09-25 00:09:33.949 | 2018-09-25 00:10:33.208 | NA        | COLUMNAR_V3 | 2.02GB    | 26.50KB    |
| 1                | Compacted | 2018-09-25 00:05:30.883 | 2018-09-25 00:07:45.809 | 1.1       | COLUMNAR_V3 | 2.02GB    | 26.40KB    |
+------------------+-----------+-------------------------+-------------------------+-----------+-------------+-----------+------------+--+
```

删除已合并的Segment数据

```
CLEAN FILES FOR TABLE carbon_table
```

Major Compaction：基于Segment大小进行合并

设置合并策略

```
SET carbon.major.compaction.size = 10240
```

手动触发Compaction

```
ALTER TABLE table_name COMPACT MAJOR'
```

Custom Compaction：指定Segment合并

指定Segment 2,3,4进行合并

```
ALTER TABLE table_name COMPACT 'CUSTOM' WHERE SEGMENT.ID IN (2,3,4)
```

CarbonData的4个使用层次

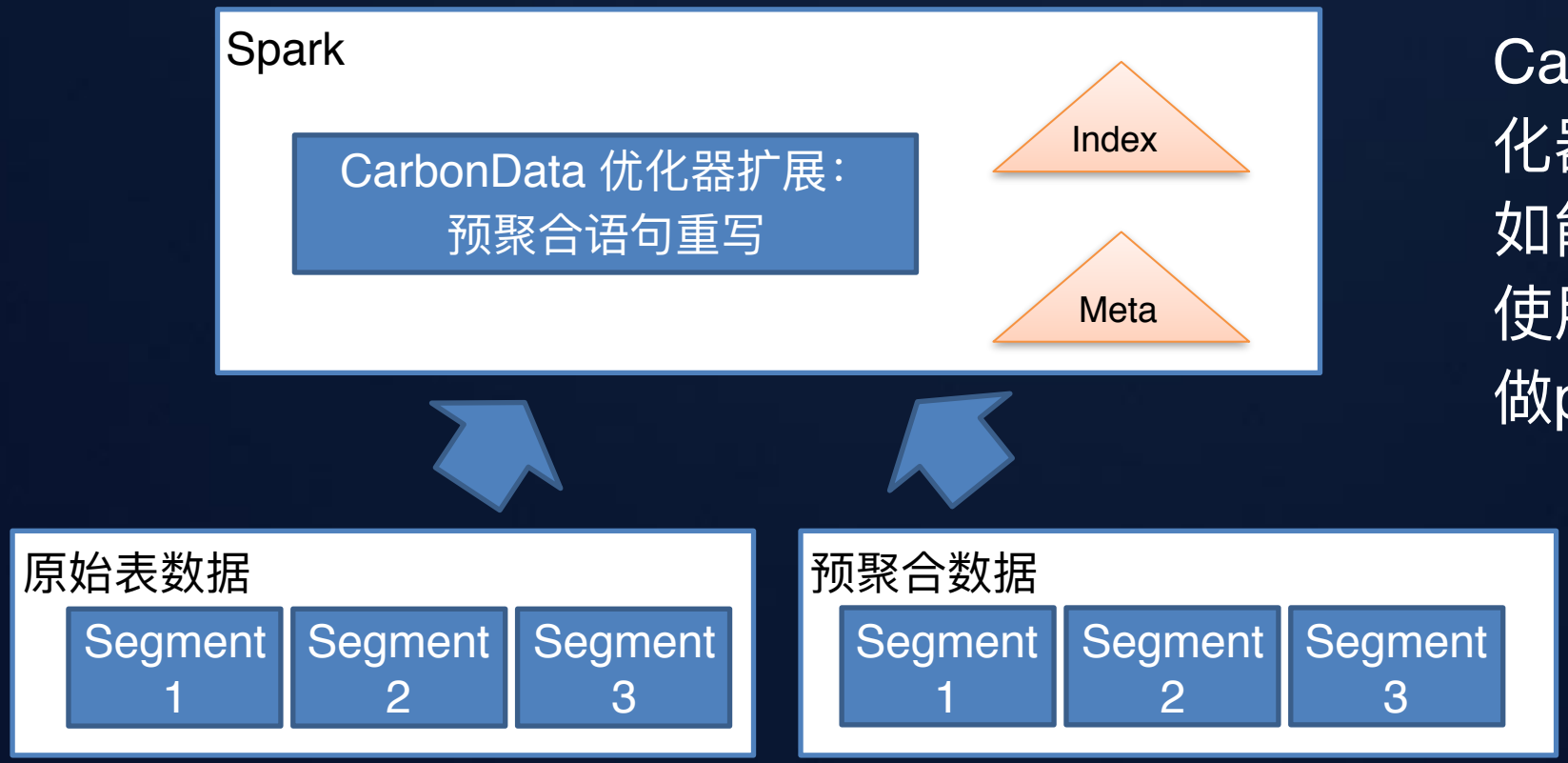
1. 基本能力：
   - 列存+事务管理+更新 => 性能等同于Parquet + 多引擎共用一份数据
2. 使用索引 => 提升海量数据的过滤性能（类搜索）
3. **使用预汇聚 => 提升复杂汇聚分析性能**
4. 使用流式入库 => 提升实时性，同时避免小文件

Pre-aggregate DataMap

```
CREATE DATAMAP agg_sales ON TABLE sales
  USING "preaggregate"
AS
  SELECT country, sex, sum(quantity), avg(price)
  FROM sales
  GROUP BY country, sex
```

- 支持的预汇聚函数：SUM，AVG，MAX，MIN，COUNT
- 入库时自动建索引或手动重建

分析性业务加速



Carbon对SparkSQL优化器扩展：
如能匹配预汇聚表，则使用代价最小的预汇聚做plan转换

Demo2

- 预汇聚加速TPC-H Q1
  - select l_returnflag, l_linestatus, sum(l_quantity) as sum_qty, sum(l_extendedprice) as sum_base_price, sum(l_extendedprice*(1-l_discount)) as sum_disc_price, sum(l_extendedprice*(1-l_discount)*(1+l_tax)) as sum_charge, avg(l_quantity) as avg_qty, avg(l_extendedprice) as avg_price, avg(l_discount) as avg_disc, count(*) as count_order from lineitem
    where l_shipdate <= date('1998-09-02')
    group by l_returnflag, l_linestatus
    order by l_returnflag, l_linestatus;

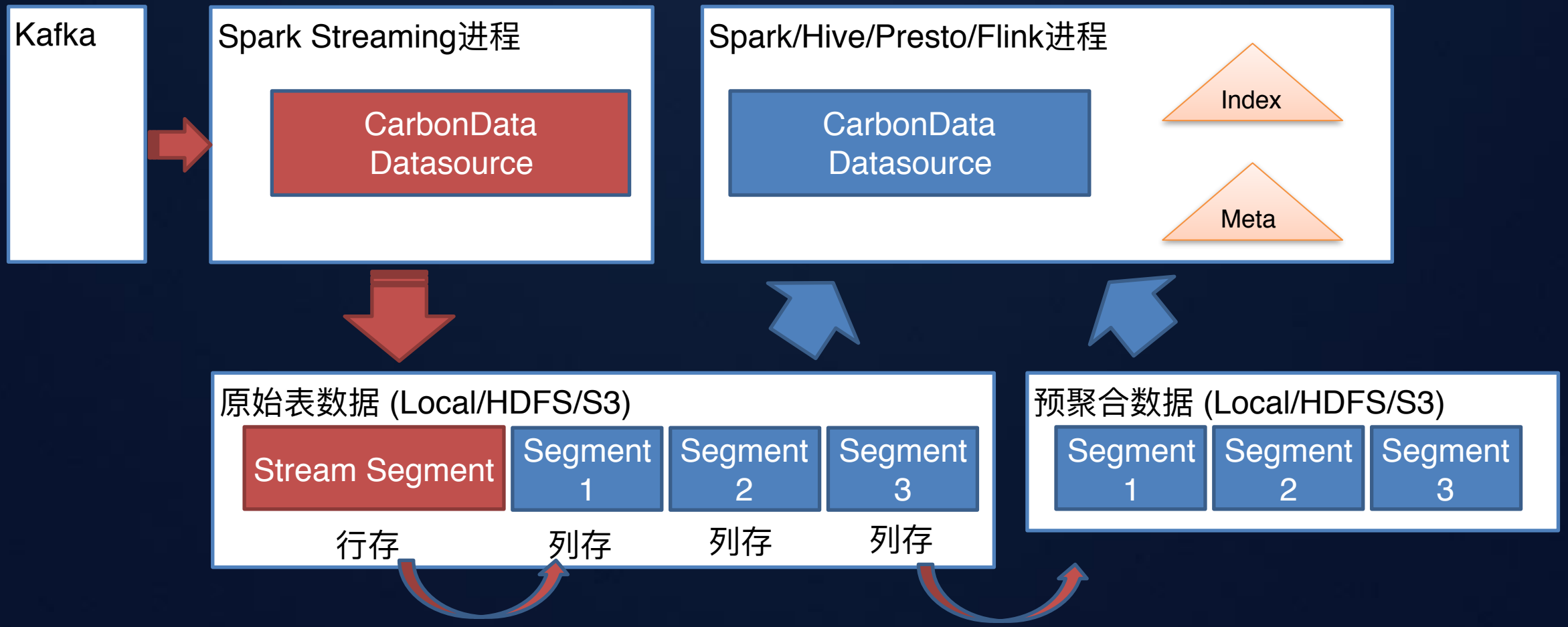
carbon	carbon_ls	parquet	parquet_par	carbon with preagg
12	13	13	14	1.3

CarbonData的4个使用层次

1. 基本能力：
   - 列存+事务管理+更新 => 性能等同于Parquet + 多引擎共用一份数据
2. 使用索引 => 提升海量数据的过滤性能（类搜索）
3. 使用预汇聚 => 提升复杂汇聚分析性能
4. **使用流式入 => 提升实时性，同时避免小文件**

流式入库，准实时数据分析

- 支持行列混合数据统一查询，数据延迟小，且避免小文件
- 支持流式数据和历史预聚合数据的SQL优化



流式入库

创建流数据源（以kafka为例）

```
CREATE TABLE source (sensorId String, temperature double)
TBLPROPERTIES ('format'='kafka', streaming='source',
'record_format'='csv|json')
```

创建目标表（carbon表）

```
CREATE TABLE sink (sensorId String, temperature double)
STOREAD AS 'carbondata'
TBLPROPERTIES (streaming='true')
```

启动流式入库作业（SparkStreaming作业）

```
CREATE STREAM stream ON TABLE sink
STMPROPERTIES (''trigger'='ProcessingTime', 'interval'='10 seconds')
AS SELECT * FROM source WHERE temperature > 30.0
```

停止作业

```
STOP STREAMS ON TABLE sink
```

Demo

- 演示Kafka流式入库，边入库边查询

CarbonData成功案例

电信详单分析场景，CarbonData替换Impala

- 5分钟入库，5分钟新增150GB原始数据，70张表
- 6个worker，共28*6core，一半做入库一半做查询（84core）
- 每天平均每张表新增600GB，所有表共新增42TB
- 最大的表一天新增1.5TB，80亿记录

加载：90MB每秒每worker，半小时一次compaction，平均2分钟入完

CarbonData替换Impala后效果：

- 指定号码查并join维表，38亿记录，Carbon10秒。Impala 60秒 (6X)
- 20并发：Carbon 55秒，Impala 450秒 (8X)
- 复杂join查询：11亿记录，26秒，50秒。

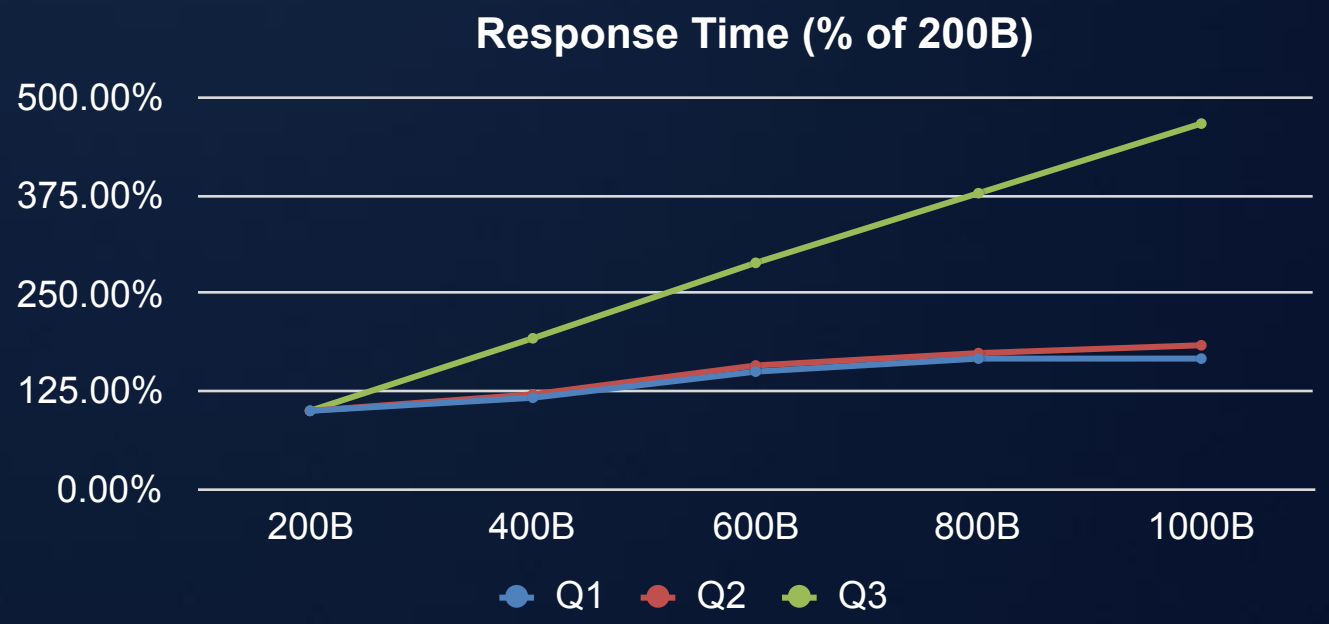
安防数据分析：单表万亿数据，秒级返回

数据：2000亿到1万亿数据

集群: >300 nodes, >10000 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。
  - 通过指定Segment查询实现分页，3秒返回首批数据。
- 汇总分析：有效利用计算资源，可线性扩展

总结

1. 基本能力：
   - 列存+事务管理+更新 => 性能等同于Parquet + 多引擎共用一份数据

     计算存储分离的PB级数仓
2. 使用索引 => 提升海量数据的过滤性能（类搜索）

   1~100X提升
3. 使用预汇聚 => 提升复杂汇聚分析性能

   1~10X提升
4. 使用流式入库 => 提升实时性，同时避免小文件

   10秒~分钟级表格刷新

以Hadoop为基础的新型数仓：海量数据，多租户，多计算引擎访问同一份数据，针对结构化搜索/OLAP优化，实时追加，批量更新

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