CarbonData : A New Hadoop File Format For Faster Data Analysis

www.huawei.com

HUAWEI TECHNOLOGIES CO., LTD.

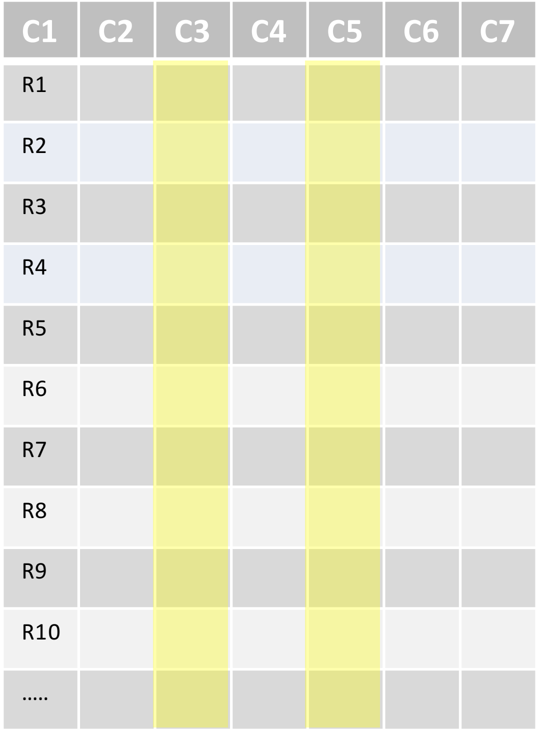
HUAWEI

Outline

- **Use Case & Motivation : Why introducing a new file format?**
- CarbonData File Format Deep Dive
- Framework Integrated with CarbonData
- Demo & Performance Comparison
- Future Plan

Use case: Sequential scan

- Full table scan
  - Big scan(**all rows**, no filter)
  - Only fetch a few columns of the table
- Common usage scenario:
  - ETL job
  - Log Analysis

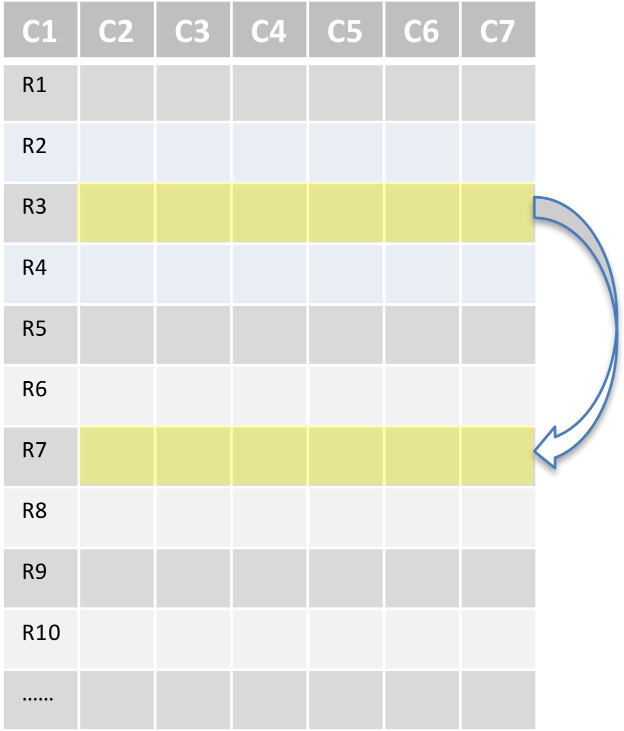


C1	C2	C3	C4	C5	C6	C7
R1						
R2						
R3						
R4						
R5						
R6						
R7						
R8						
R9						
R10						
…..						

Use case: Random Access

- Predicate filtering on many columns(point query
  - Row-key query (like HBase)
  - Narrow scan but might fetch **all columns**
  - Requires second/sub-second level low-latency

- Common usage scenario:
  - Operational query
  - User profiling

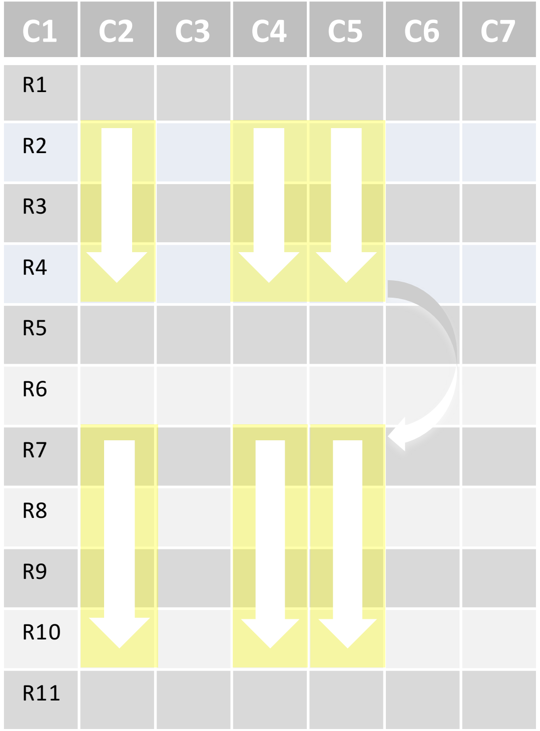


C1	C2	C3	C4	C5	C6	C7
R1						
R2						
R3						
R4						
R5						
R6						
R7						
R8						
R9						
R10						
……						

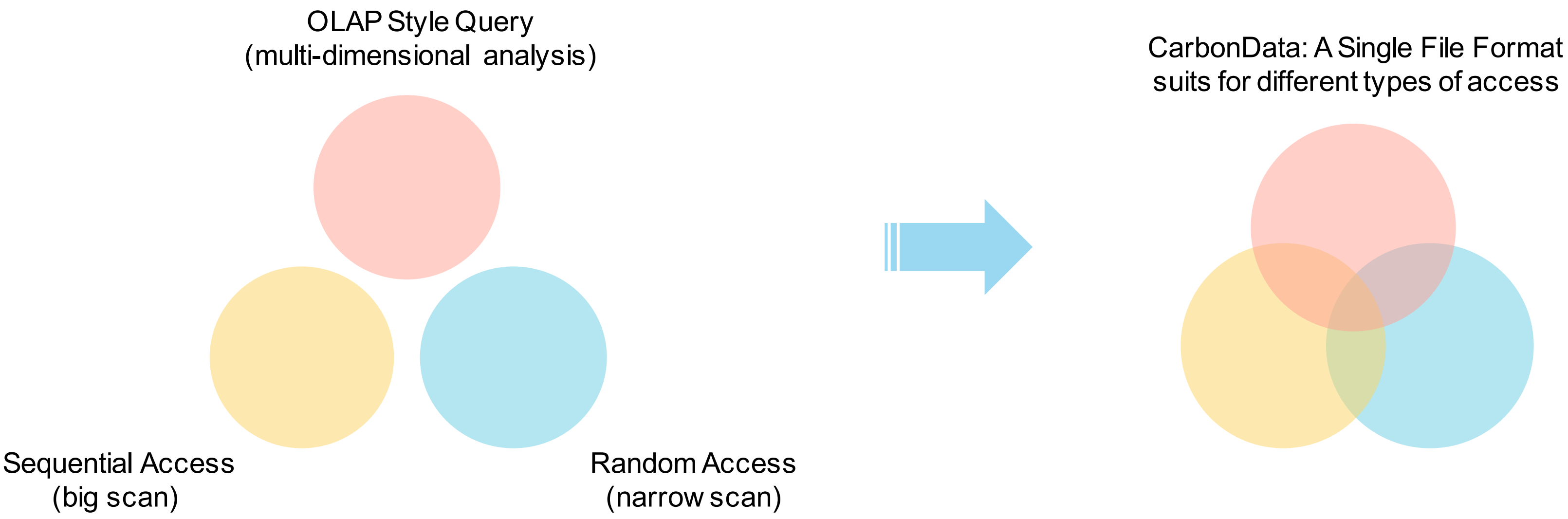
Use case: OLAP-Style Query

- Interactive data analysis for any dimensions
  - Involves aggregation / join
  - Roll-up, Drill-down, Slicing and Dicing
  - Low-latency ad-hoc query

- Common usage scenario:
  - Dash-board reporting
  - Fraud & Ad-hoc Analysis



Motivation



OLAP Style Query
(multi-dimensional analysis)

Sequential Access
(big scan)

Random Access
(narrow scan)

CarbonData: A Single File Format
suits for different types of access

Why CarbonData

Based on the below requirements, we investigated existing file formats in the Hadoop eco-system, but **we could not find a suitable solution that can satisfy all the requirements at the same time,** so we start designing CarbonData.

- *Support big scan & only fetch a few columns*
- *Support primary key lookup response in sub-second.*
- *Support interactive OLAP-style query over big data which involve many filters in a query, this type of workload should response in seconds.*
- *Support fast individual record extraction which fetch all columns of the record.*
- *Support HDFS so that customer can leverage existing Hadoop cluster.*

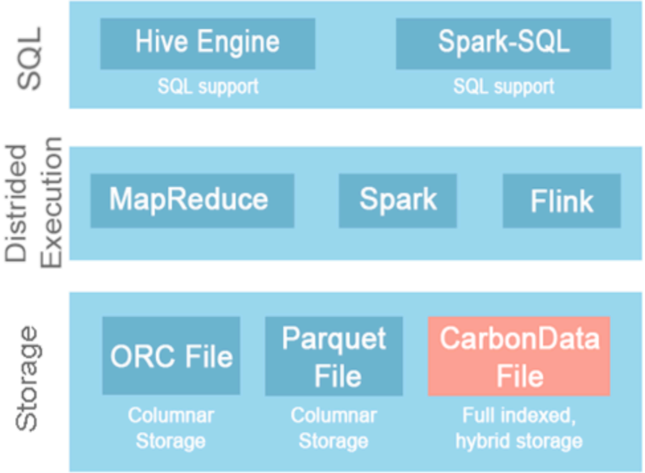
When we investigated Parquet/ORC, it seems they work very well for R1 and R5, but they does not meet for R2,R3,R4. So we designed CarbonData mainly to add following differentiating features:

- *Stores data along with index: it can significantly accelerate query performance and reduces the I/O scans and CPU resources, where there are filters in the query. CarbonData index consists of multiple level of indices, a processing framework can leverage this index to reduce the task it needs to schedule and process, and it can also do skip scan in more finer grain unit (called blocklet) in task side scanning instead of scanning the whole file.*
- *Operable encoded data :Through supporting efficient compression and global encoding schemes, can query on compressed/encoded data, the data can be converted just before returning the results to the users, which is "late materialized".*
- *Column group: Allow multiple columns to form a column group that would be stored as row format. This reduces the row reconstruction cost at query time.*
- *Supports for various use cases with one single Data format : like interactive OLAP-style query, Sequential Access (big scan), Random Access (narrow scan).*

Design Goals

- **Low-Latency** for various types of data access pattern
- Allow **fast query on fast data**
- Ensure **Space Efficiency**
- General format available on **Hadoop-ecosystem**

- Read-optimized **columnar storage**
- Leveraging **multi-level Index** for low-latency
- Support **column group** to leverage the benefit of row-based
- Enables dictionary encoding for **deferred decoding** for aggregation
- Broader Integration across Hadoop-ecosystem



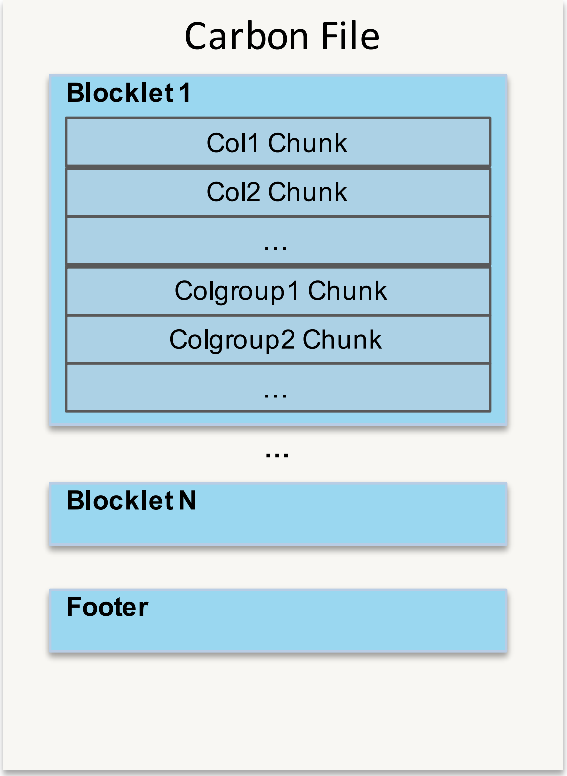
Outline

- Use cases & Motivation: Why introducing a new file format?
- **CarbonData File Format Deep Dive**
- Framework Integrated with CarbonData
- Demo & Performance Comparison
- Future Plan

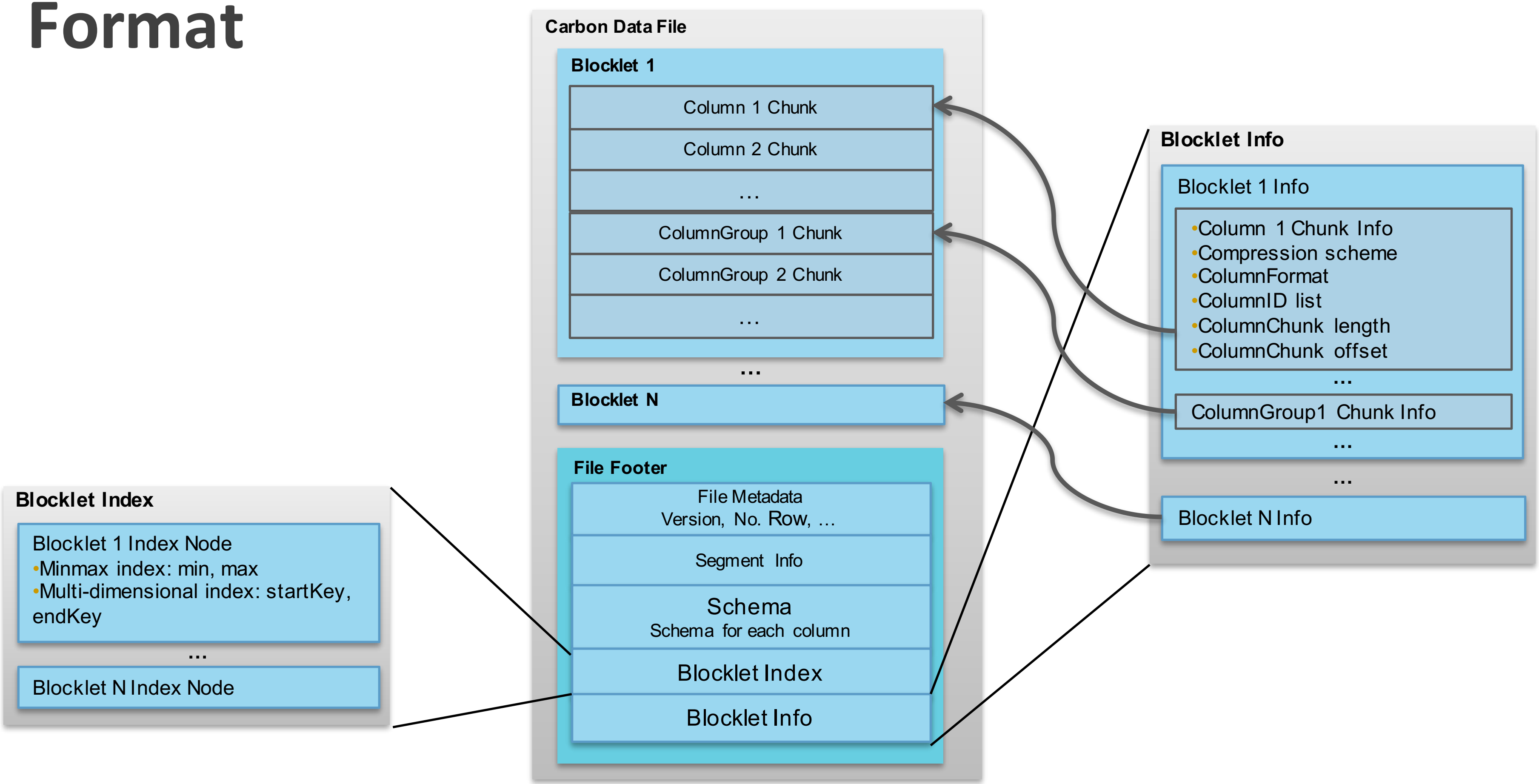
CarbonData File Structure

- **Blocklet : A set of rows in columnar format**
- **Column chunk : Data for one column/column group in a Blocklet**
  - Allow multiple columns forms a column group & stored as row-based
  - Column data stored as sorted index
- **Footer : Metadata information**
  - File level metadata & statistics
  - Schema
  - Blocklet Index & Blocklet level Metadata

Remark : One CarbonData file is a HDFS block.



Format



Carbon Data File

Blocklet 1

Column 1 Chunk

Column 2 Chunk

…

ColumnGroup 1 Chunk

ColumnGroup 2 Chunk

…

…

Blocklet N

File Footer

File Metadata
Version, No. Row, …

Segment Info

Schema
Schema for each column

Blocklet Index

Blocklet Info

Blocklet Info

Blocklet 1 Info

- Column 1 Chunk Info
- Compression scheme
- ColumnFormat
- ColumnID list
- ColumnChunk length
- ColumnChunk offset

…

ColumnGroup1 Chunk Info

…

…

Blocklet N Info

Blocklet Index

Blocklet 1 Index Node

- Minmax index: min, max
- Multi-dimensional index: startKey, endKey

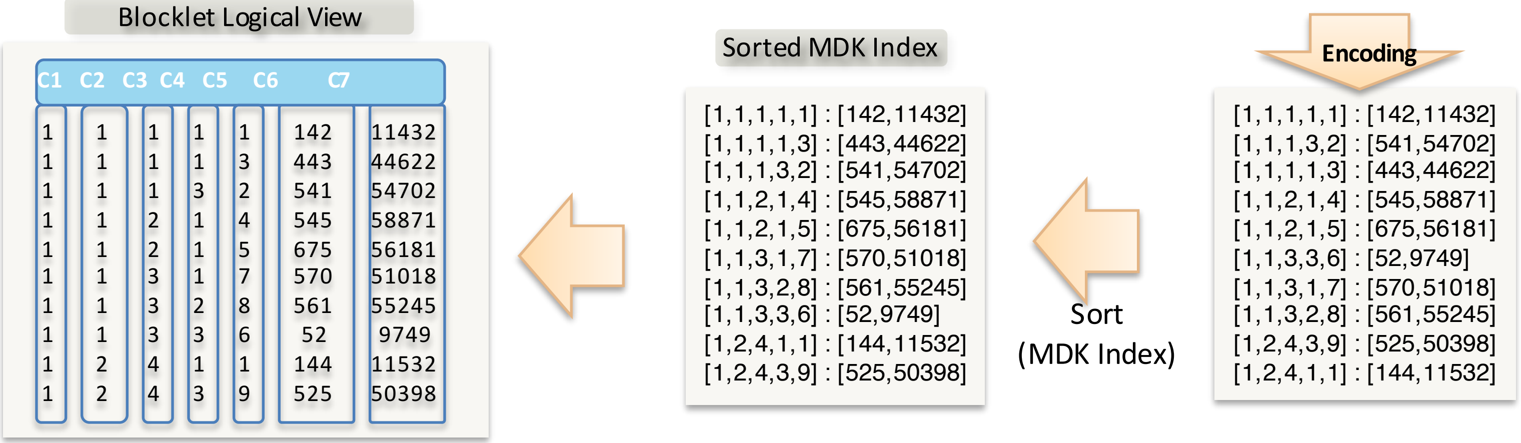
…

Blocklet N Index Node

Blocklet

- Data are sorted along MDK (multi-dimensional keys)
    - data stored as index in columnar format

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



[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]

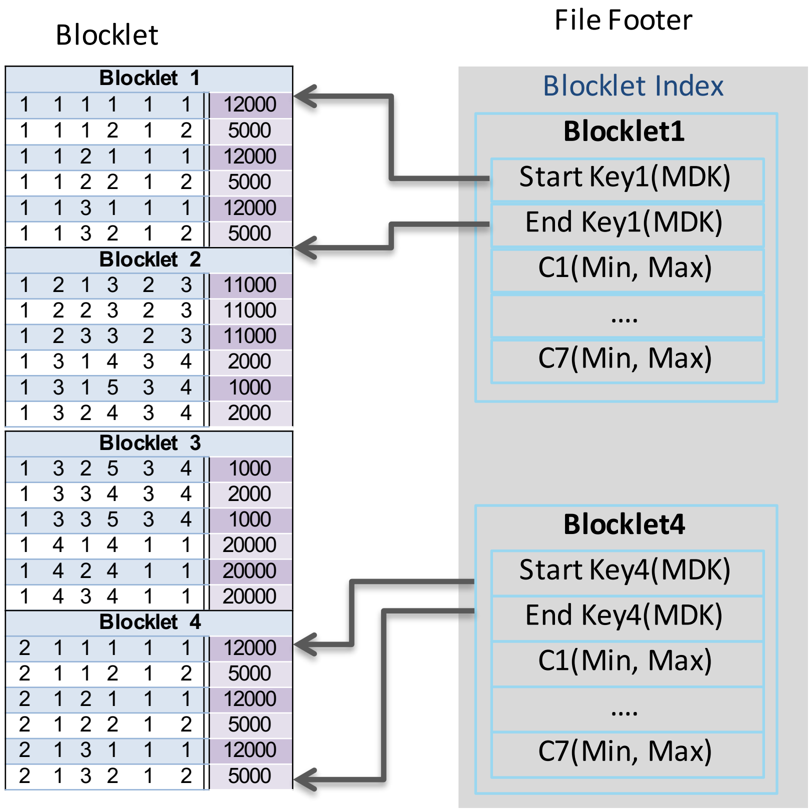
Sorted MDK Index

[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 Logical View

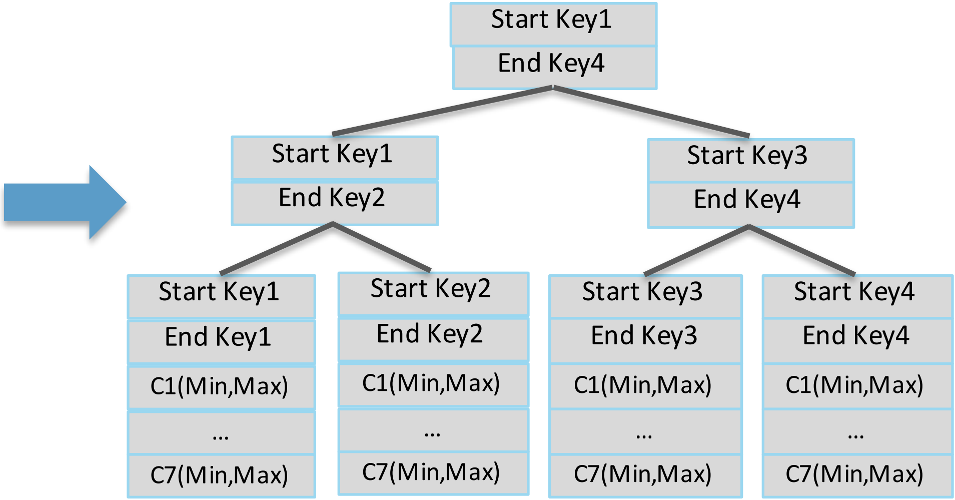
C1	C2	C3	C4	C5	C6	C7
1	1	1	1	1	142	11432
1	1	1	1	3	443	44622
1	1	1	3	2	541	54702
1	1	2	1	4	545	58871
1	1	2	1	5	675	56181
1	1	3	1	7	570	51018
1	1	3	2	8	561	55245
1	1	3	3	6	52	9749
1	2	4	1	1	144	11532
1	2	4	3	9	525	50398

File Level Blocklet Index



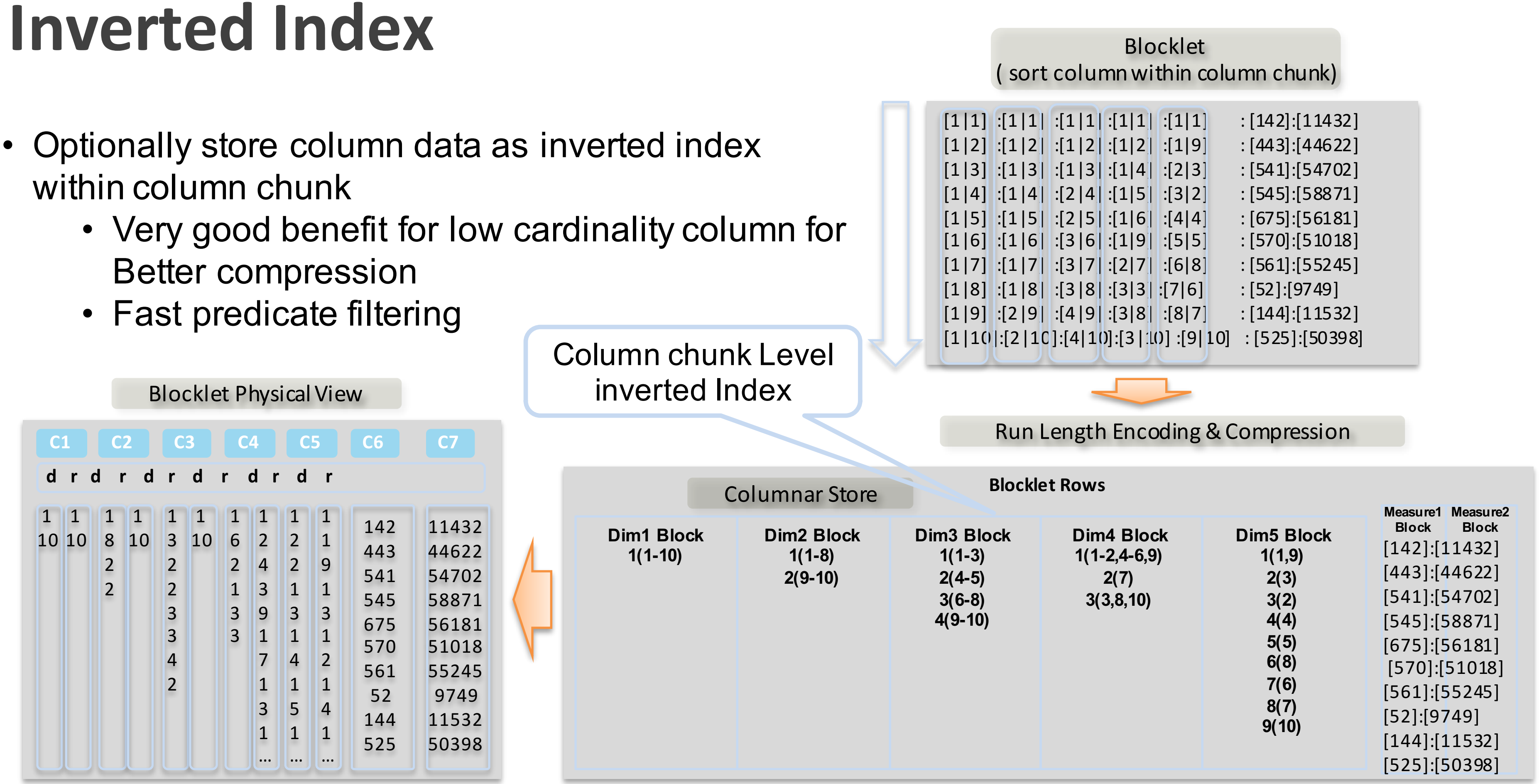
Blocklet 1						
1	1	1	1	1	1	12000
1	1	1	2	1	2	5000
1	1	2	1	1	1	12000
1	1	2	2	1	2	5000
1	1	3	1	1	1	12000
1	1	3	2	1	2	5000
Blocklet 2						
1	2	1	3	2	3	11000
1	2	2	3	2	3	11000
1	2	3	3	2	3	11000
1	3	1	4	3	4	2000
1	3	1	5	3	4	1000
1	3	2	4	3	4	2000
Blocklet 3						
1	3	2	5	3	4	1000
1	3	3	4	3	4	2000
1	3	3	5	3	4	1000
1	4	1	4	1	1	20000
1	4	2	4	1	1	20000
1	4	3	4	1	1	20000
Blocklet 4						
2	1	1	1	1	1	12000
2	1	1	2	1	2	5000
2	1	2	1	1	1	12000
2	1	2	2	1	2	5000
2	1	3	1	1	1	12000
2	1	3	2	1	2	5000

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



Inverted Index

- Optionally store column data as inverted index within column chunk
  - Very good benefit for low cardinality column for Better compression
  - Fast predicate filtering



Blocklet
(sort column within column chunk)

[1|1]	:[1|1|	:[1|1|	:[1|1|	:[1|1]	: [142]:[11432]
[1|2]	:[1|2|	:[1|2|	:[1|2|	:[1|9]	: [443]:[44622]
[1|3]	:[1|3|	:[1|3|	:[1|4|	:[2|3]	: [541]:[54702]
[1|4]	:[1|4|	:[2|4|	:[1|5|	:[3|2]	: [545]:[58871]
[1|5]	:[1|5|	:[2|5|	:[1|6|	:[4|4]	: [675]:[56181]
[1|6]	:[1|6|	:[3|6|	:[1|9|	:[5|5]	: [570]:[51018]
[1|7]	:[1|7|	:[3|7|	:[2|7|	:[6|8]	: [561]:[55245]
[1|8]	:[1|8|	:[3|8|	:[3|3|	:[7|6]	: [52]:[9749]
[1|9]	:[2|9|	:[4|9|	:[3|8|	:[8|7]	: [144]:[11532]
[1|10|	:[2|10]	:[4|10]	:[3|10]	:[9|10]	: [525]:[50398]

Run Length Encoding & Compression

Column chunk Level inverted Index

Columnar Store

Blocklet Rows

Dim1 Block	Dim2 Block	Dim3 Block	Dim4 Block	Dim5 Block	Measure1 Block	Measure2 Block

1(1-10)	1(1-8)
	2(9-10)
		1(1-3)
		2(4-5)
		3(6-8)
		4(9-10)
			1(1-2,4-6,9)
			2(7)
			3(3,8,10)
				1(1,9)
				2(3)
				3(2)
				4(4)
				5(5)
				6(8)
				7(6)
				8(7)
				9(10)
					[142]:[11432]
					[443]:[44622]
					[541]:[54702]
					[545]:[58871]
					[675]:[56181]
					[570]:[51018]
					[561]:[55245]
					[52]:[9749]
					[144]:[11532]
					[525]:[50398]	

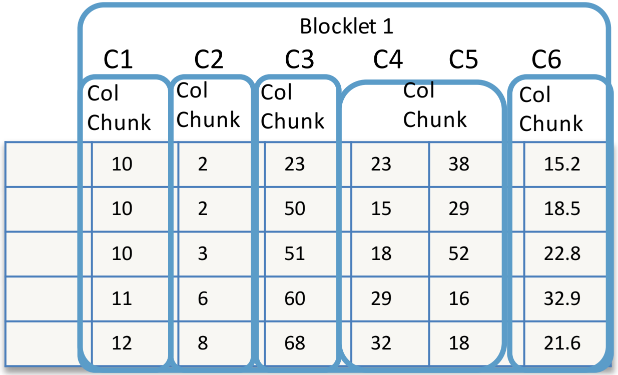
Blocklet Physical View

d r d r d r d r d r d r

C1		C2		C3		C4		C5		C6	C7
1	1	1	1	1	1	1	1	1	1	142	11432
10	10	8	10	3	10	6	2	2	1	443	44622
		2		2		2	4	2	9	541	54702
		2		2		1	3	1	1	545	58871
				3		3	9	3	3	675	56181
				3		3	1	1	1	570	51018
				4			7	4	2	561	55245
				2			1	1	1	52	9749
							3	5	4	144	11532
							1	1	1	525	50398
							…	…	…		

Column Group

- Allow multiple columns form a column group
  - stored as a single column chunk in row-based format
  - suitable to set of columns frequently fetched together
  - saving stitching cost for reconstructing row



Blocklet 1

	C1	C2	C3	C4	C5	C6
	Col Chunk	Col Chunk	Col Chunk	Col Chunk		Col Chunk
	10	2	23	23	38	15.2
	10	2	50	15	29	18.5
	10	3	51	18	52	22.8
	11	6	60	29	16	32.9
	12	8	68	32	18	21.6

Nested Data Type Representation

Arrays

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

Struts

- 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

Encoding & Compression

- Efficient encoding scheme supported:
  - DELTA, RLE, BIT_PACKED
  - Dictionary: table level global dictionary
- Compression Scheme: Snappy

Big Win:

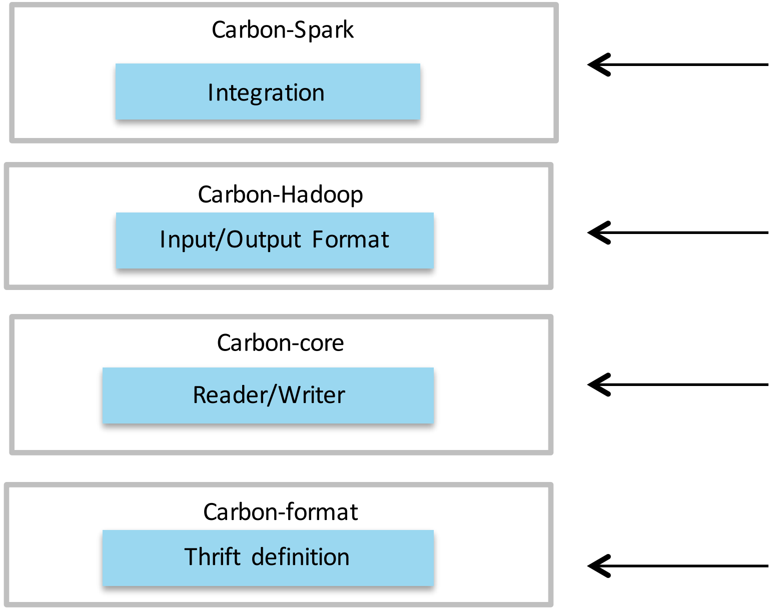
- Speedup Aggregation
- Reduce run-time memory footprint
- Enable deferred decoding
- Enable fast distinct count

Compression ratio : 1/3

Outline

- Use Case & Motivation: Why introducing a new file format?
- CarbonData File Format Deep Dive
- **Framework Integrated with CarbonData**
- Demo & Performance Comparison
- Future Plan

CarbonData Modules(to understand more)



Integration of Carbon with Spark including query optimization

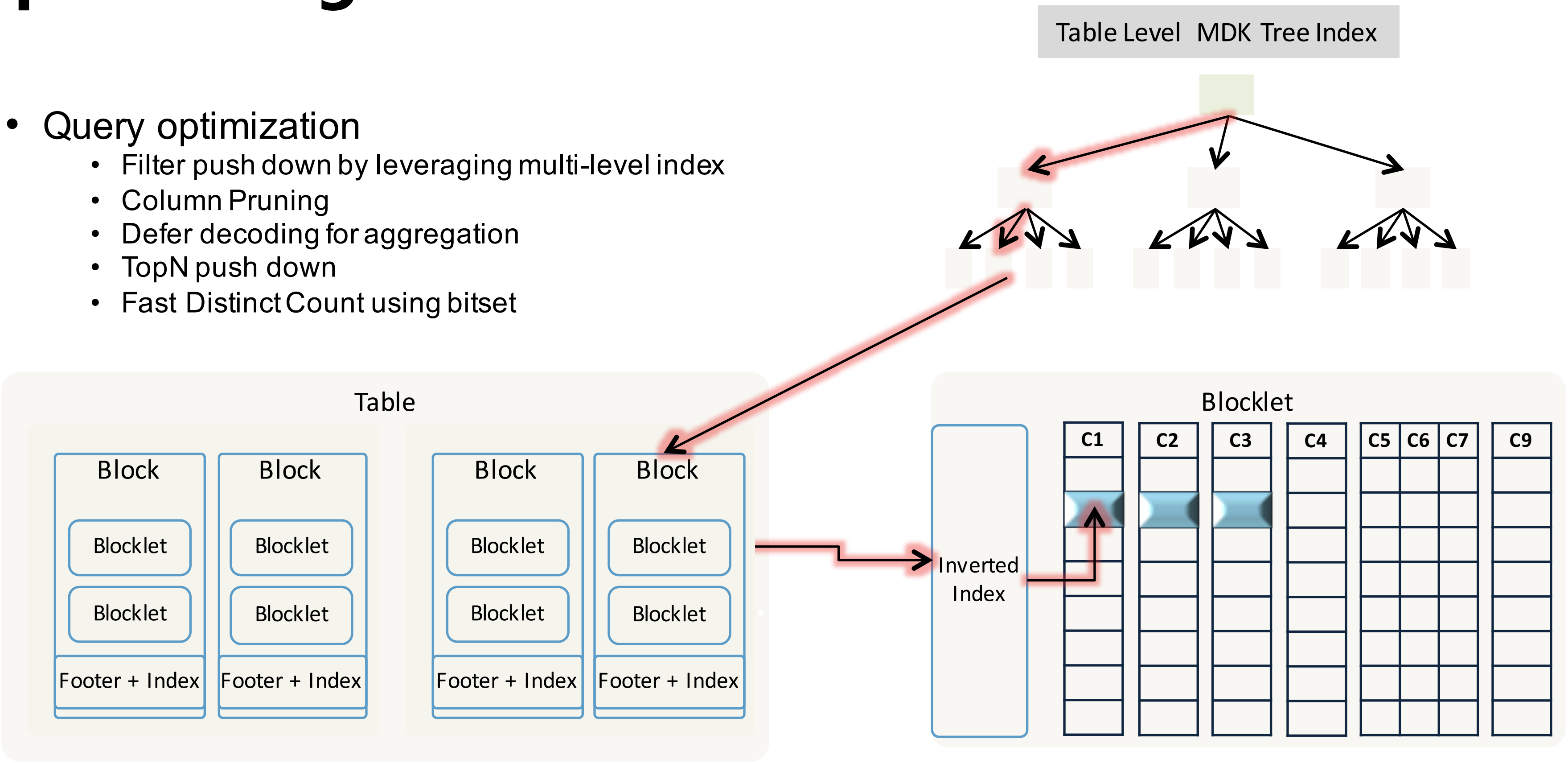
Provide Hadoop Input/Output Format interface

Core component of format implementation for reading/writing Carbon data

Language Agnostic Format Specification

Spark Integration

- Query optimization
  - Filter push down by leveraging multi-level index
  - Column Pruning
  - Defer decoding for aggregation
  - TopN push down
  - Fast Distinct Count using bitset



Outline

- Use Case & Motivation: Why introducing a new file format?
- CarbonData File Format Deep Dive
- Framework Integrated with CarbonData
- **Demo & Performance Comparison**
- Future Plan

DEMO and Performance Comparison

数据量超过1亿行记录，性能比较会更明显。

因个人便携配置有限，本例子只以100万行纪录为demo演示：

1.Spark SQL里查询csv格式数据性能：

```
import org.apache.spark.sql.catalyst.util._
import org.apache.spark.sql.SQLContext
var df = sqlContext.read.format("com.databricks.spark.csv").option("header", "true").option("inferSchema", "true").load("./carbondata/hzmeetup.csv”)
benchmark { df.filter($"country" === "china" and $"name" === "hangzhou" and $"seq" < 100000).count }
```

2.Spark SQL里查询parquet格式数据性能：

```
import org.apache.spark.sql.{CarbonContext, DataFrame, Row, SaveMode, SQLContext}
df.write.mode(SaveMode.Overwrite).parquet("./carbondata/parquet")
val parquetdf = sqlContext.parquetFile("./carbondata/parquet")
benchmark { parquetdf.filter($“country” === “china” and $“name” === “hangzhou” and $“seq” < 100000).count }
```

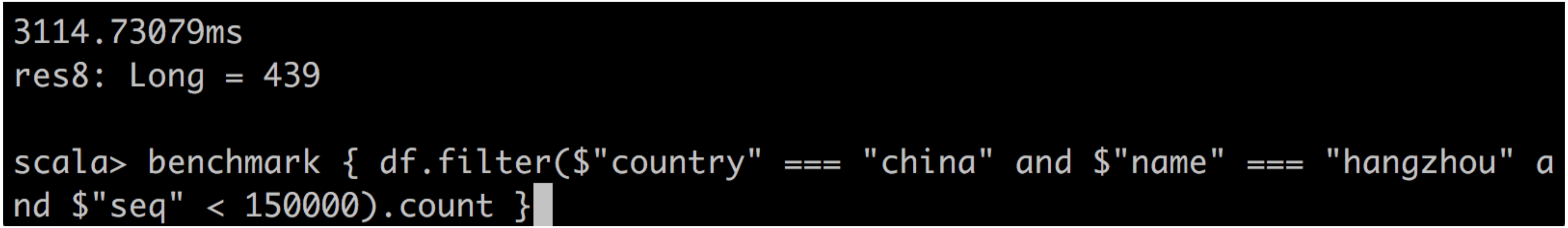
3. Spark SQL里查询CarbonData格式数据性能：

请参照CarbonData github例子，初始化CarbonContext。

```
cc.sql("CREATE TABLE meetupTable (seq Int, name String, country String,age Int) STORED BY 'org.apache.carbondata.format'")
cc.sql("LOAD DATA LOCAL INPATH './carbondata/hzmeetup.csv' INTO TABLE meetupTable”)
val carbondf = cc.read.format("org.apache.spark.sql.CarbonSource").option("tableName", "meetupTable").load()
benchmark { carbondf.filter($“country” === “china” and $“name” === “hangzhou” and $“seq” < 100000).count }
```

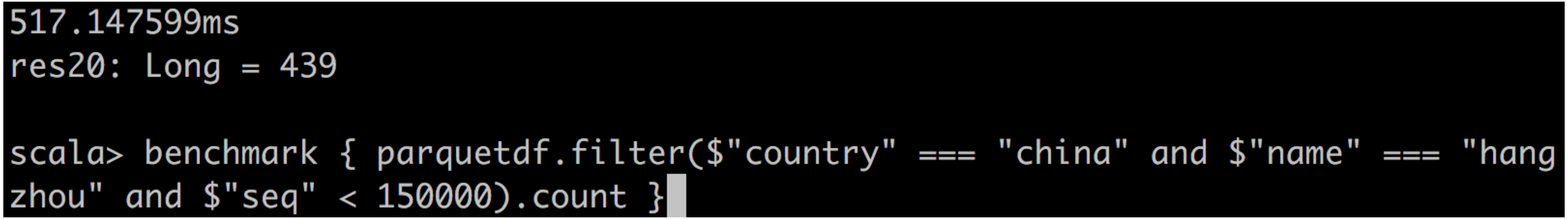
注：以上代码因拷贝原因，双引号和单引号可能被转了格式，在Spark shell里需要修正。具体步骤，请参看github中wiki的quick start

CSV,Parquet,CarbonData test result :



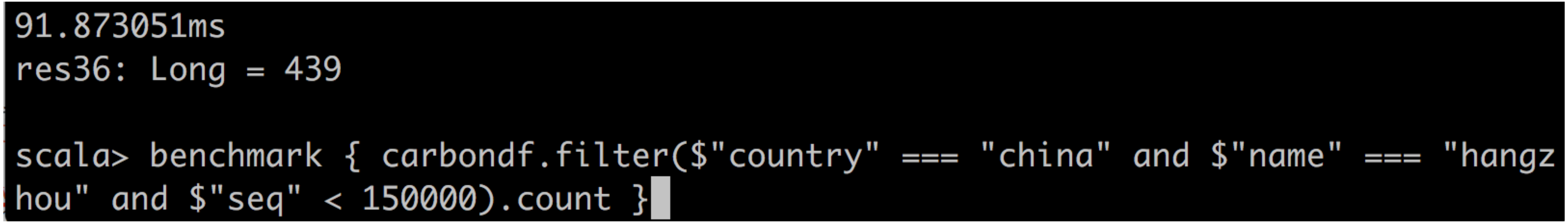
```
3114.73079ms
res8: Long = 439

scala> benchmark { df.filter($"country" === "china" and $"name" === "hangzhou" a
nd $"seq" < 150000).count }
```



```
517.147599ms
res20: Long = 439

scala> benchmark { parquetdf.filter($"country" === "china" and $"name" === "hang
zhou" and $"seq" < 150000).count }
```



```
91.873051ms
res36: Long = 439

scala> benchmark { carbondf.filter($"country" === "china" and $"name" === "hangz
hou" and $"seq" < 150000).count }
```

Outline

- Motivation: Why introducing a new file format?
- CarbonData File Format Deep Dive
- Framework Integrated with CarbonData
- Demo & Performance Comparison
- **Future Plan**

Future Plan

- Upgrade to Spark 2.0
- Integrate with BI tools
- Add append support
- Support pre-aggregated table
- Broader Integration across Hadoop-ecosystem

Community

- CarbonData is Apache, welcome contribution to our Github: https://github.com/apache/incubator-carbondata

Thank you

www.huawei.com

The information in this document may contain predictive statements including, without limitation, statements regarding the future financial and operating results, future product portfolio, new technology, etc. There are a number of factors that could cause actual results and developments to differ materially from those expressed or implied in the predictive statements. Therefore, such information is provided for reference purpose only and constitutes neither an offer nor an acceptance. Huawei may change the information at any time without notice.