APACHE HBASE × MI

HBASECON ASIA2019

THE COMMUNITY EVENT FOR APACHE HBASE™

HBASECON ASIA2019

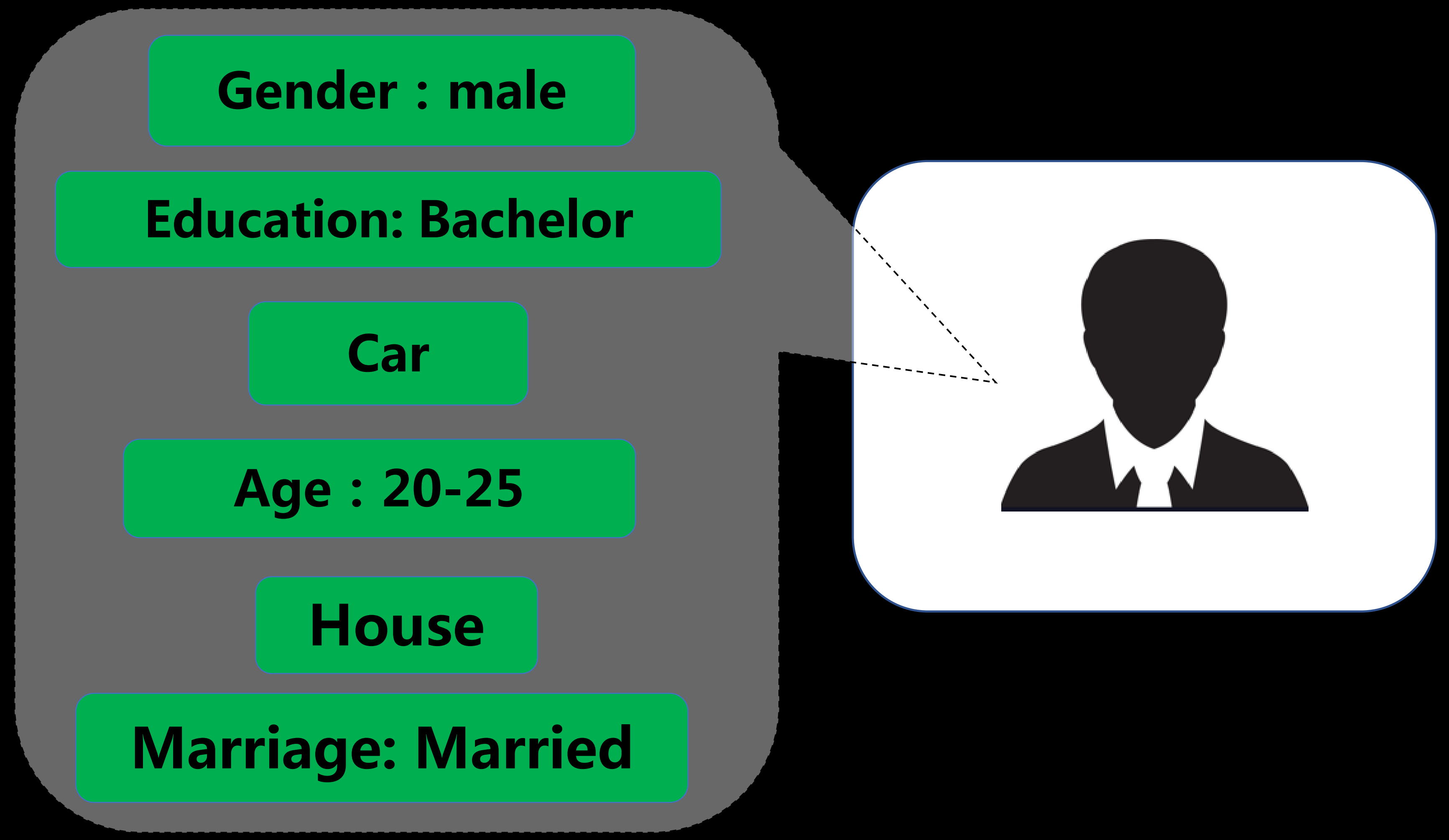
Distributed Bitmap Index Solution

Xingjun Hao

Huawei

Motivation

- Motivation for designing software
  - HBase is suitable for storing massive tag data



Gender : male

Education: Bachelor

Car

Age : 20-25

House

Marriage: Married

	Info:Gender	Info:Age	Car:Brand	House:Address
Entity1	Male	20_25	Audi	
Entity2	Male	25_30		Urban
Entity3	Female	25_30	Audi	
Entity4	Male	20_25		Suburbs

Hbase Data Model is suitable for tag data storage

1. Distributed LSM-based storage: PB-level storage and good write performance

1. Sorted RowKey -> Support Quick Point Query and Range Query

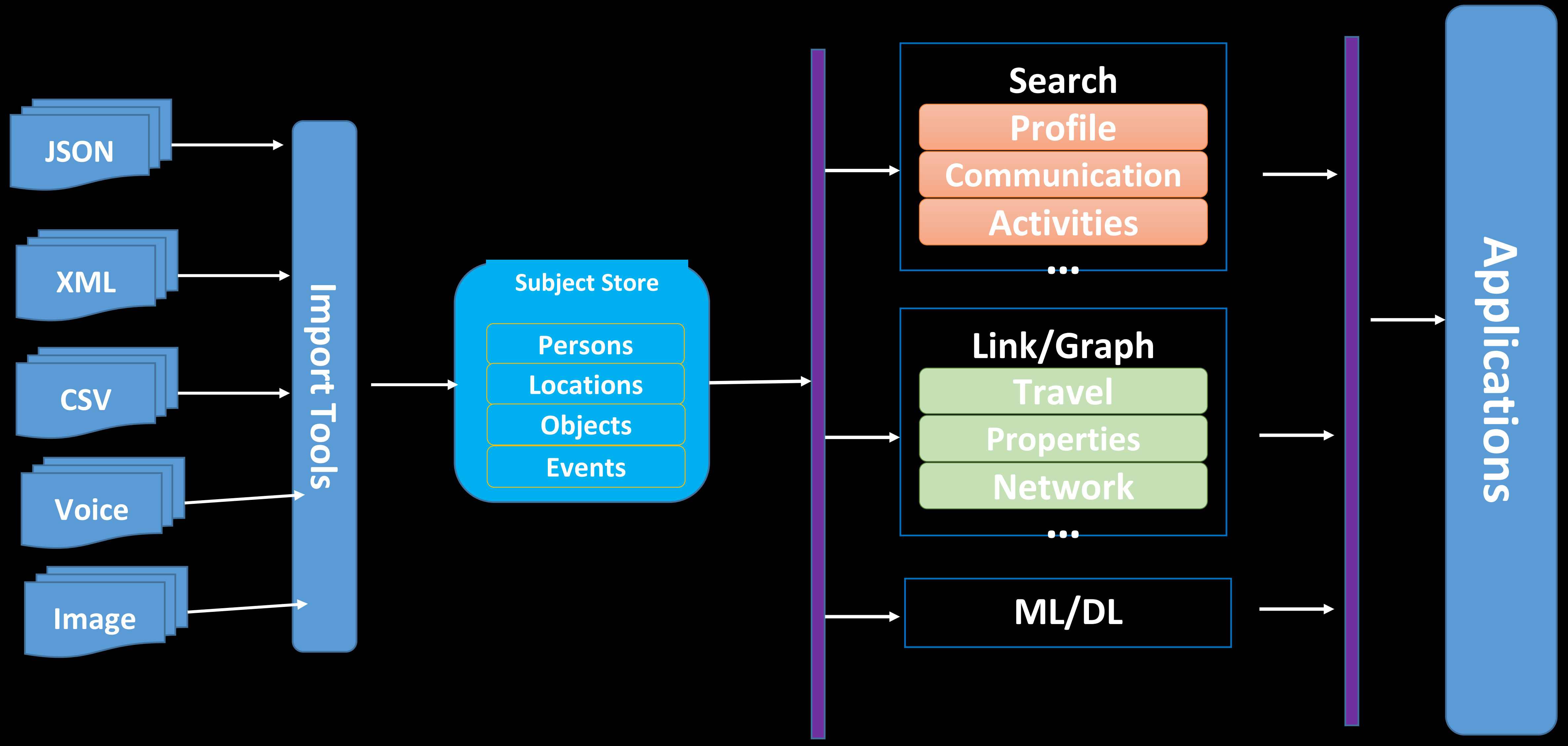
2. Columns -> Support Each entity has a custom tag schema.

3. Cell -> Can have multi-value, can be empty.

- Lack of efficient indices when processing ad hoc queries

Scenrio	Advantage
Get(“RowKeyX”)/Scan(“RowKeyX” -> “RowKeyY”)	Good
Put	Good
Fexiable	Good
Ad-hoc Query(“TagA AND TagB AND (TagX OR TagY)”)	Poor

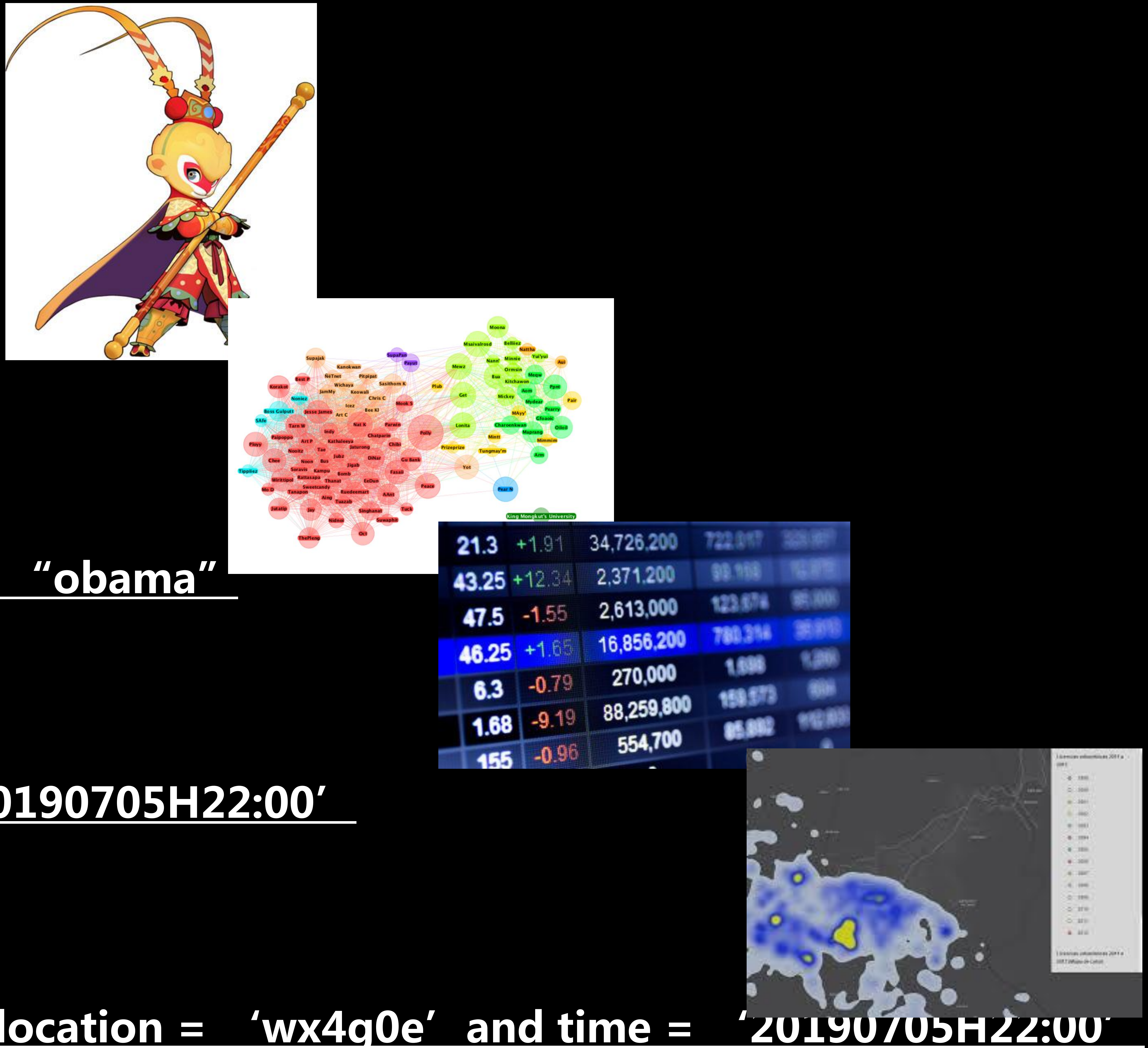
Example: Security project



- Production data rate
  - ~ 1 TB per day
  - Storage data of a year ~ 400TB
  - each event ~1 KB in size
- Consume data rate
  - 1000 queries / second
  - Desire to get the 300 rows collection within 100ms in per query
- Engineer may query any of the 500 attributes
  - Each query may involve conditions on 5 ~ 8 attributes. Ad-hoc queries
  - Eg. select * from table WHERE (location = “area-A”)
  - select * from table WHERE (location = “area-A” AND time = “20190705”)
  - select * from table WHERE (location = "area-A” OR location = "area-B” AND time = “20190705”)

HBase can't satisfy this scene.

Applications Involve Massive Tags



AI : select * from pictures where theme = “monkey”

Graph Computing : select * from graphs where edge = “obama”

Time series: select * from timeseries where time = ‘20190705H22:00’

Spatial temporal: select * from spatialtemporal where location = ‘wx4g0e’ and time = ‘20190705H22:00’

Searching and Indexing Requirements

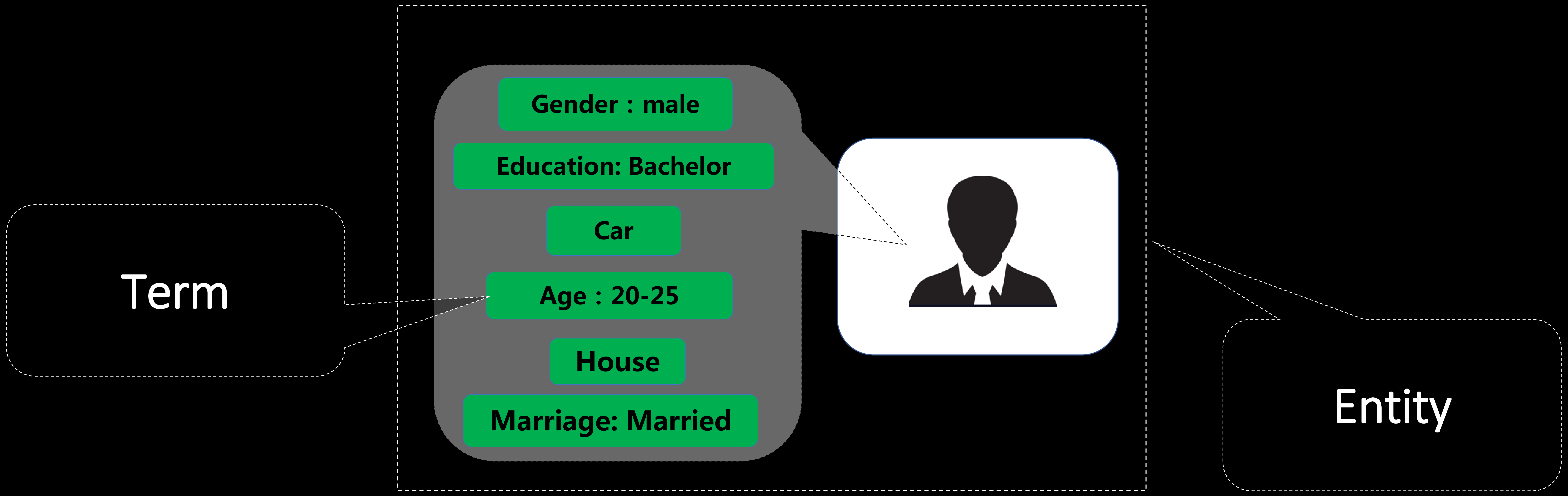
- Some common features of the large tag datasets
  - Read-mostly
  - Large high-dimensional data: millions or billions of records, each record with tens or hundreds of attributes
  - Many queries are high-dimensional point queries or partial range queries
  - Most users desire to modify queries interactively

- Existing database software not specialized for these tasks
  - Secondary index on HBase: slow, low storage efficiency
  - ES/lucene: cannot be updated frequently

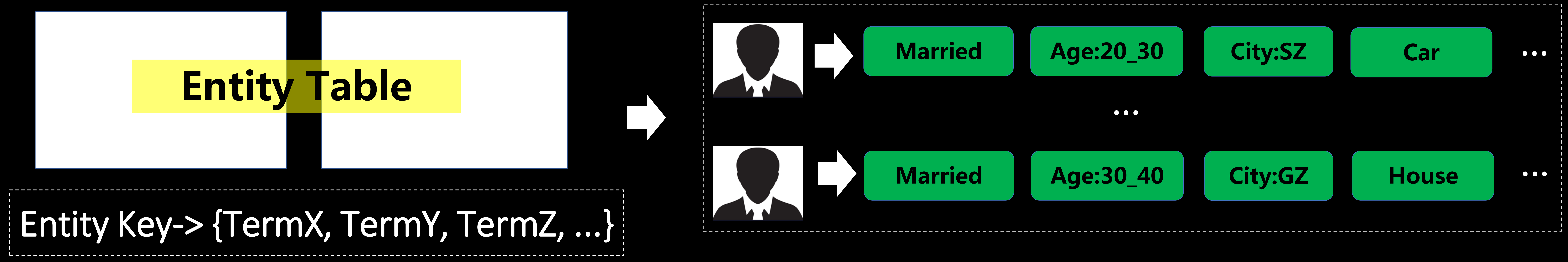
Issues to Be Discussed

- Framework : Organization of data on HBase
  - Data Organization: An entity table is used to store primary data, while an index table to to store bitmap index data.
  - Index Schema: The Bitmap index is actually an inverted index based bitmap index framework.
- Implement :
  - Index implement: Coprocessor-based bitmap index building and querying.
  - Index Data partition.
- API
  - Write Data with HBase API
  - Normal Query/ Paging Query/ Top-N Query/ Counting Query/ Sample Query

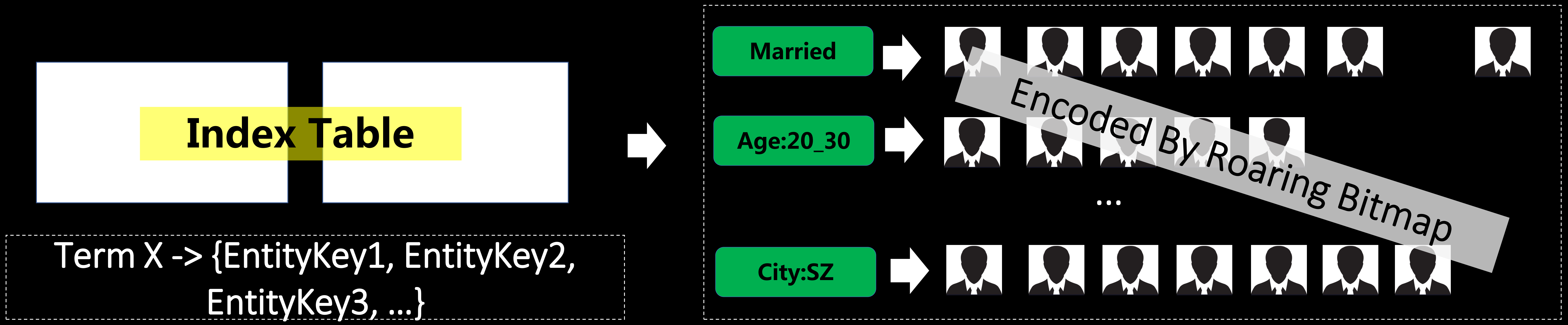
Framework: Concept



Framework: Organization of data Overview



Entity Key-> {TermX, TermY, TermZ, ...}



Term X -> {EntityKey1, EntityKey2, EntityKey3, ...}

Framework: An Example of Organization of data

Entity Table

Entity Key-> {TermX, TermY, TermZ, …}

value

ColumnFamily:Column

Rowkey

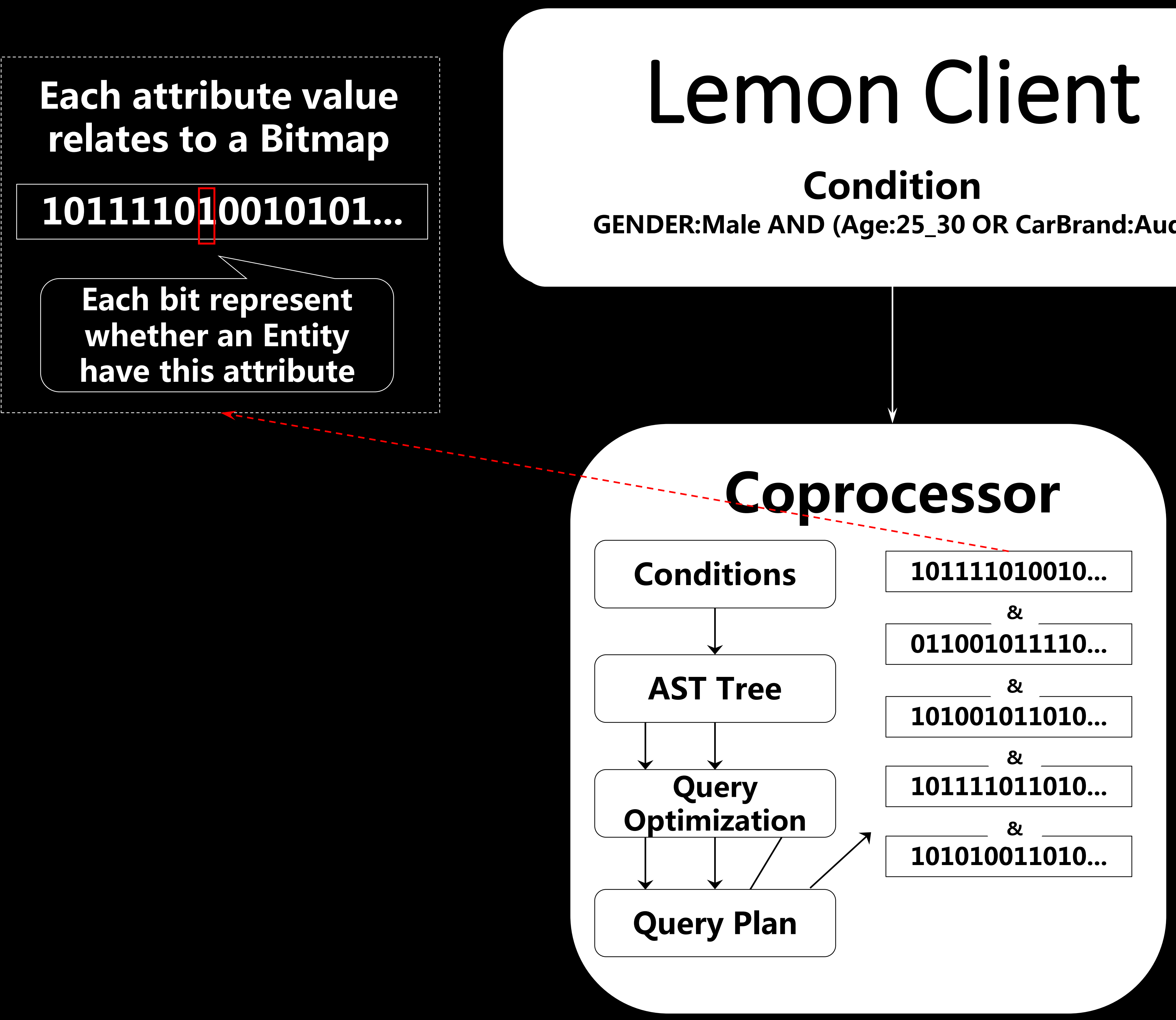
	Info:Gender	Info:Age	Car:Brand	House:Address
Entity0	Male	20_25	Audi	
Entity1	Male	25_30	Audi	
Entity2	Female	25_30		Urban
Entity3	Male	20_25		Suburbs

Index Table

Term X -> {EntityKey1, EntityKey2, EntityKey3, …}

	Index: 1101	Index: 0110	Index: 1100
Gender:Male	B		
Age:25_30		B	
Car:Brand			B

Framework: Index Schema



Each attribute value relates to a Bitmap

1011110100010101...

Each bit represent whether an Entity have this attribute

Lemon Client

Condition

GENDER:Male AND (Age:25_30 OR CarBrand:Audi)

Coprocessor

Conditions

AST Tree

Query Optimization

Query Plan

101111010010...

&

011001011110...

&

101001011010...

&

101111011010...

&

101010011010...

GENDER:Male AND (Age:25_30 OR CarBrand:Audi)

❶ Recevie Query Conditions

	Index: 1101	Index: 0110	Index: 1100
Gender:Male	B		
Age:25_30		B	
CarBrand:Audi			B

Gender:Male	1	1	0	1
Age:25_30	0	1	1	0
CarBrand:Audi	1	1	0	0

e. g. 1 AND （0 OR 1） = 1

② Bitmap Computing

1	1	0	0

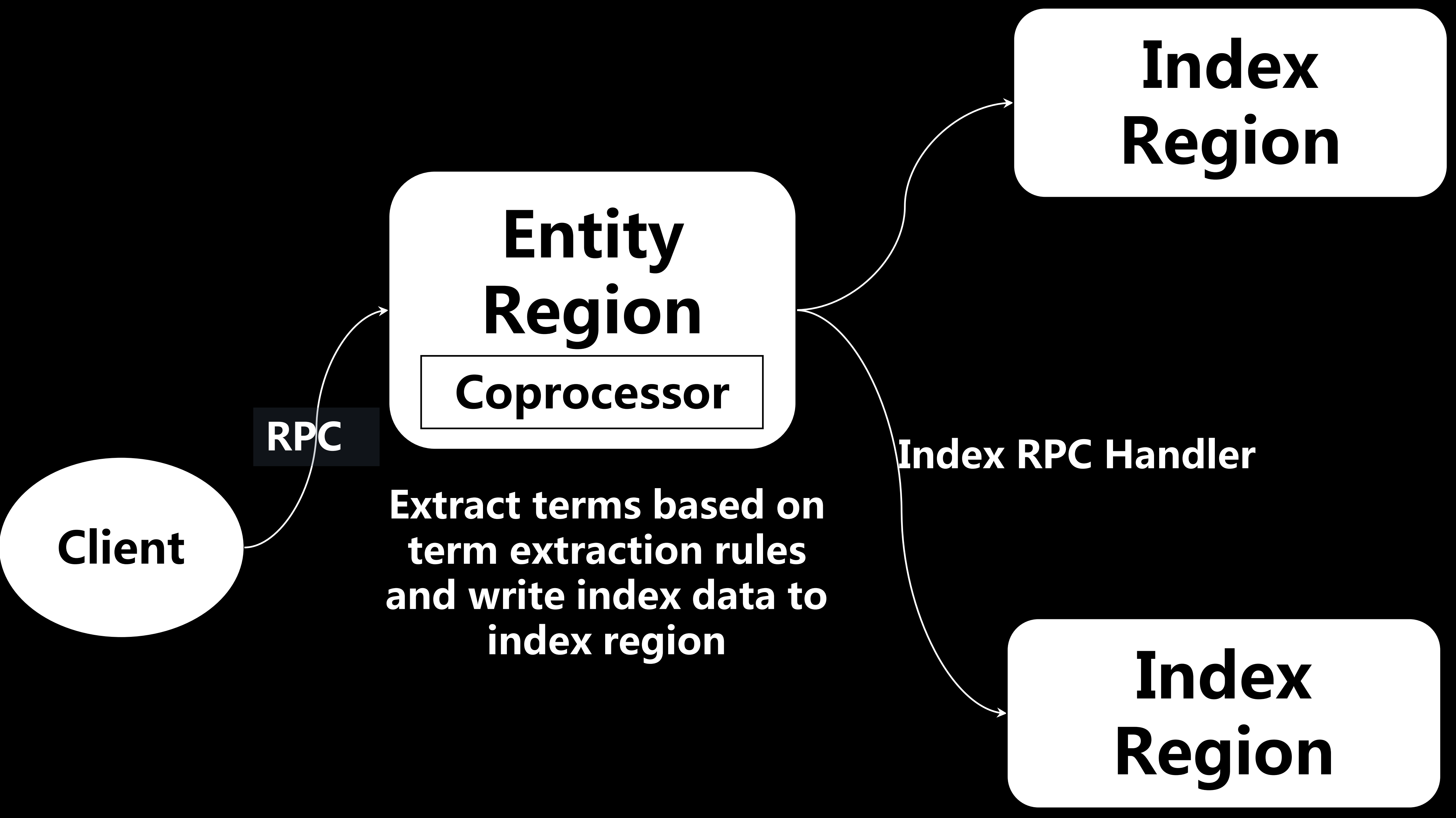
③ Fetch Entities And Return

	Info:Gender	Info:Age	Car:Brand
Entity0	Male	20_25	Audi
Entity1	Male	25_30	Audi

Issues to Be Discussed

- Framework : Organization of data on HBase
  - Data Organization: An entity table is used to store primary data, while an index table to store bitmap index data.
  - Index Schema: The Bitmap index is actually an inverted index based bitmap index framework.
- **Implement :**
  - Index implement: Coprocessor-based bitmap index building and querying.
  - Index Data partition.
- API
  - Write Data with HBase API
  - Normal Query/ Paging Query/ Top-N Query/ Counting Query/ Sample Query

Implement: Coprocessor-based bitmap index building



Extract terms based on term extraction rules and write index data to index region

RowKey	Column Family (I)	
U0100100	Age	City
	20_30	SZ
U0200111	Age30_40	City_GZ
	30_40	GZ

1

Term
FieldName: Age, FieldValue: 20_30

Term
FieldName: City, FieldValue: SZ

2

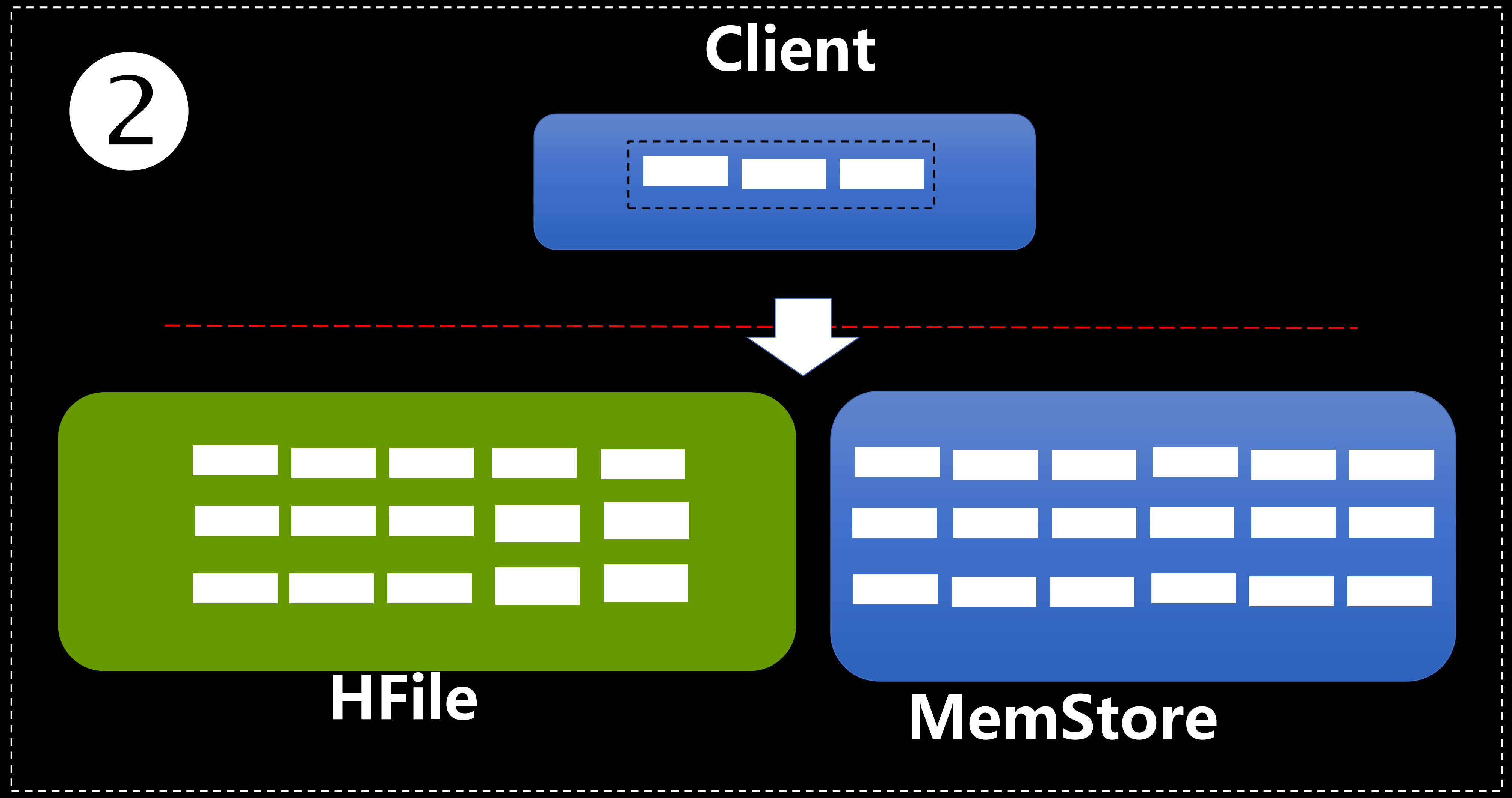
Column Family (B)	Column Family (I)
U0100100 -> 1	
U0200111 -> 2	
1 -> U0100100	
2 -> U0200111	
	Age20_30 -> {1}
	Age30_40 -> {2}
	City_GZ -> {2}
	City_SZ -> {1}
	Married -> {1,2}

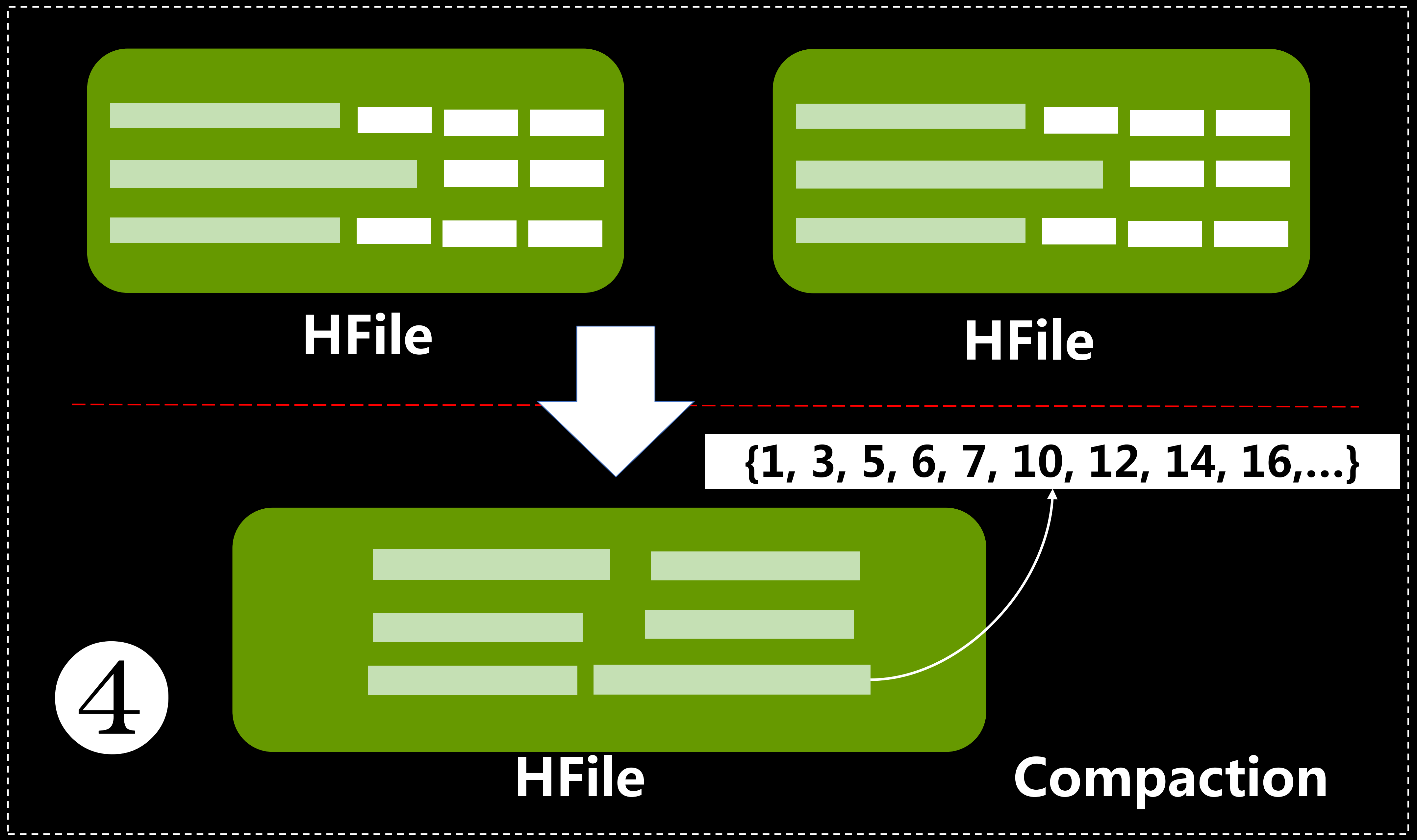
Assign a ID to each Entity

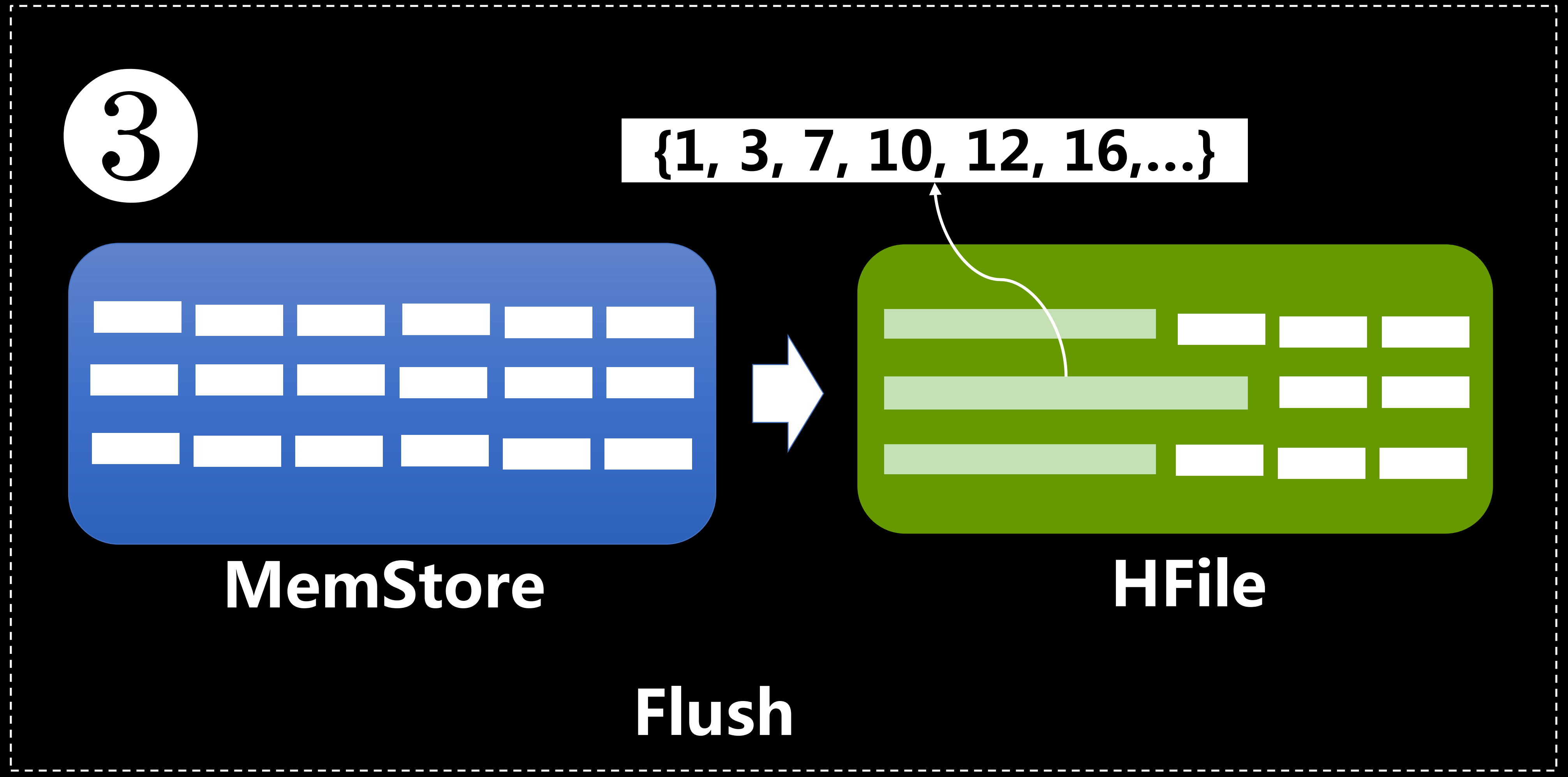
Inverted index of Term to Entity ID

2. Only build the inverted index when write to MemStore.

Implement: Coprocessor-based bitmap index building (continued)







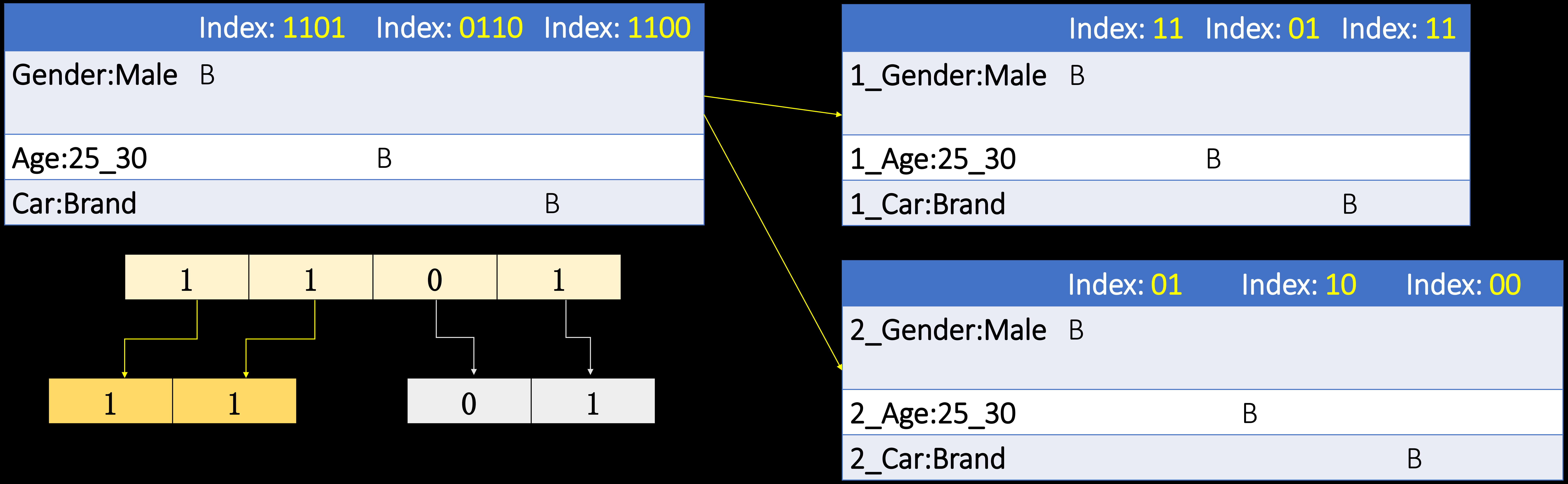
3. Flush phase: build the bitmap index of the HFiles.
4. Compaction phase: rebuild bitmap index when merge HFiles.

Implement: Data Partition

if there are 10 billions entities contains the same term? The bitmap will be about 1GB.

Harm read performance

Idea: Each bitmap index is responsible for only a portion of the entity

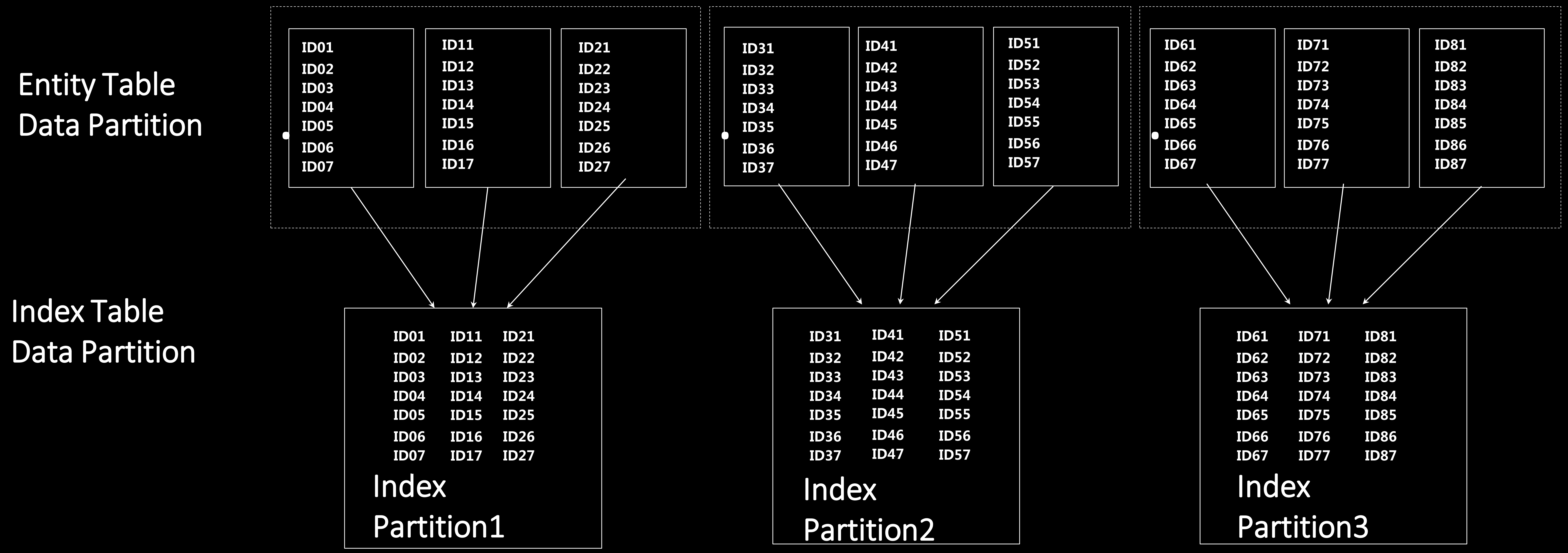


	Index: 1101	Index: 0110	Index: 1100
Gender:Male	B		
Age:25_30		B	
Car:Brand			B

	Index: 11	Index: 01	Index: 11
1_Gender:Male	B		
1_Age:25_30		B	
1_Car:Brand			B

	Index: 01	Index: 10	Index: 00
2_Gender:Male	B		
2_Age:25_30		B	
2_Car:Brand			B

Implement: Data Partition



Entity Table
Data Partition

ID01
ID02
ID03
ID04
ID05
ID06
ID07

ID11
ID12
ID13
ID14
ID15
ID16
ID17

ID21
ID22
ID23
ID24
ID25
ID26
ID27

ID31
ID32
ID33
ID34
ID35
ID36
ID37

ID41
ID42
ID43
ID44
ID45
ID46
ID47

ID51
ID52
ID53
ID54
ID55
ID56
ID57

ID61
ID62
ID63
ID64
ID65
ID66
ID67

ID71
ID72
ID73
ID74
ID75
ID76
ID77

ID81
ID82
ID83
ID84
ID85
ID86
ID87

Index Table
Data Partition

ID01	ID11	ID21
ID02	ID12	ID22
ID03	ID13	ID23
ID04	ID14	ID24
ID05	ID15	ID25
ID06	ID16	ID26
ID07	ID17	ID27

Index
Partition1

ID31	ID41	ID51
ID32	ID42	ID52
ID33	ID43	ID53
ID34	ID44	ID54
ID35	ID45	ID55
ID36	ID46	ID56
ID37	ID47	ID57

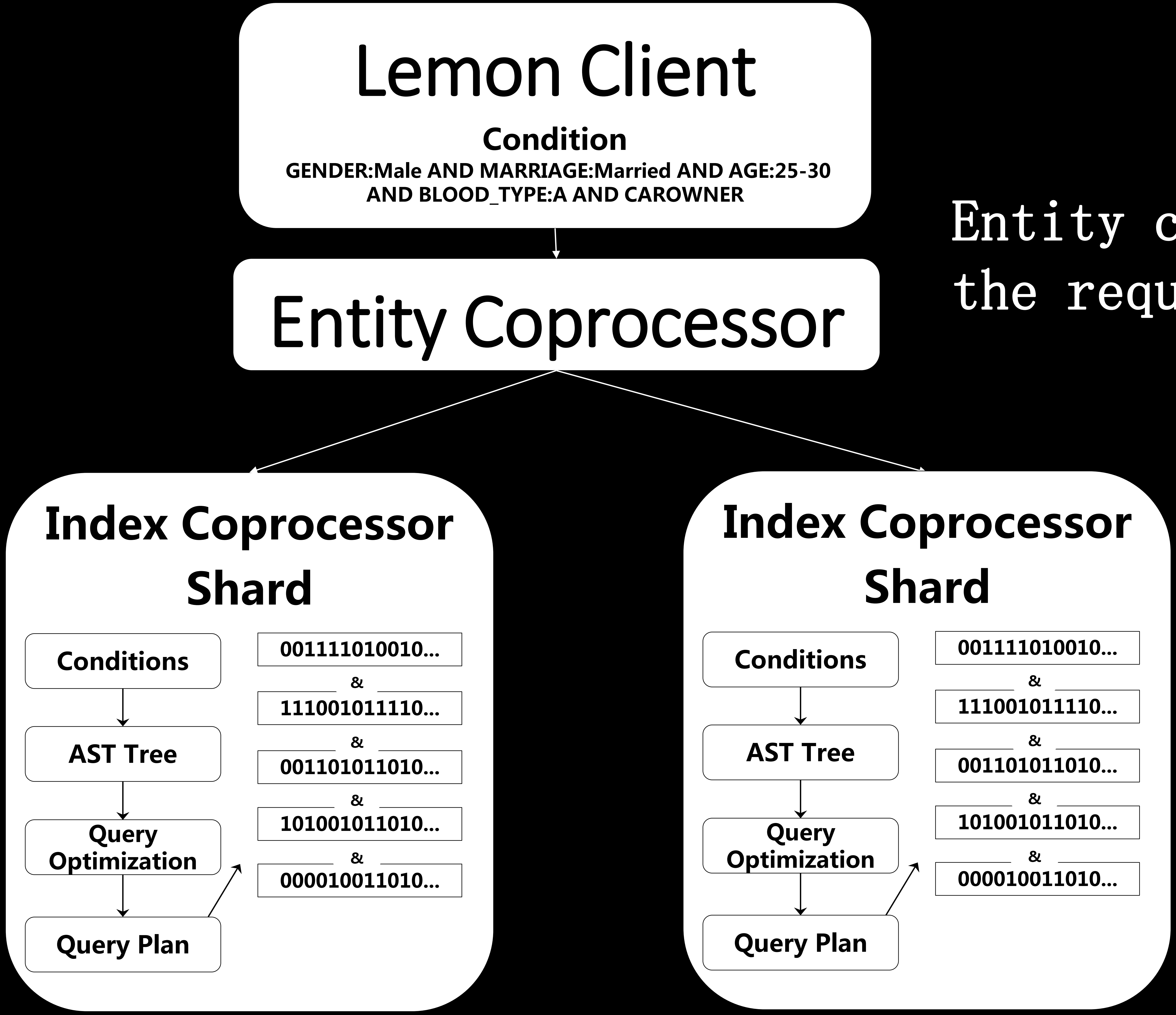
Index
Partition2

ID61	ID71	ID81
ID62	ID72	ID82
ID63	ID73	ID83
ID64	ID74	ID84
ID65	ID75	ID85
ID66	ID76	ID86
ID67	ID77	ID87

Index
Partition3

1. The number of regions of entity table and number of shard of index table are specified by user
2. 1 shard is response for 1 or more regions of entity table.

Implement: Coprocessor-based bitmap index query



Lemon Client

Condition

GENDER:Male AND MARRIAGE:Married AND AGE:25-30 AND BLOOD_TYPE:A AND CAROWNER

Entity Coprocessor

Index Coprocessor Shard

Conditions

AST Tree

Query Optimization

Query Plan

001111010010...

&

111001011110...

&

001101011010...

&

101001011010...

&

000010011010...

Index Coprocessor Shard

Conditions

AST Tree

Query Optimization

Query Plan

001111010010...

&

111001011110...

&

001101011010...

&

101001011010...

&

000010011010...

Entity coprocessor is response for distributing the requests to Index Coprocessors

Issues to Be Discussed

- Framework : Organization of data on HBase
  - Data Organization: An entity table is used to store primary data, while an index table to store bitmap index data.
  - Index Schema: The Bitmap index is actually an inverted index based bitmap index framework.

- Implement
  - Index implement: Coprocessor-based bitmap index building and querying.
  - Index Data partition.

- **API**
  - Write Data with HBase API
  - Normal Query/ Paging Query/ Top-N Query/ Counting Query/ Sample Query

API

- Put: Write interfaces are the same as hbase.
- Query: Query Grammar

Query grammar in BNF：

```
Query ::= ( Clause )+
Clause ::= ["AND", "OR", "NOT"] ([Field:]Value | "(" Query ")" )
```

- A Query is a series of clauses. Each Clause can also be a nested query
- Supports AND/OR/NOT operators. AND indicates this clause is required, NOT indicates this clause is prohibited, OR indicates this clause should appear in the matching results. The default operator is OR is none operator specified.
- Parenthese “(” “) ” can be used to improve the priority of a sub-query

Query(1): Normal Query/ Paging Query/ Top-N Query

Query records that meet the combined label criteria for "City: Shenzhen AND Age:20_30", and request the first time to retrieve 10 records:

```
LemonTable lemonTable = new LemonTable(table);

LemonQuery query = LemonQuery.builder()
  .setQuery("City:Shenzhen AND Age:20_30")
  .setCaching(10)
  .build();

ResultSet resultSet = lemonTable.query(query);

// Data records that are cached to the Client side can be accessed as follows:
List<EntityEntry> entries = resultSet.listRows();

// Get 20 rows of records from index position 100
resultSet.listRows(100, 20);

// Top 10 records can be obtained by:
List<EntityEntry> entries = resultSet.listRows(10);
```

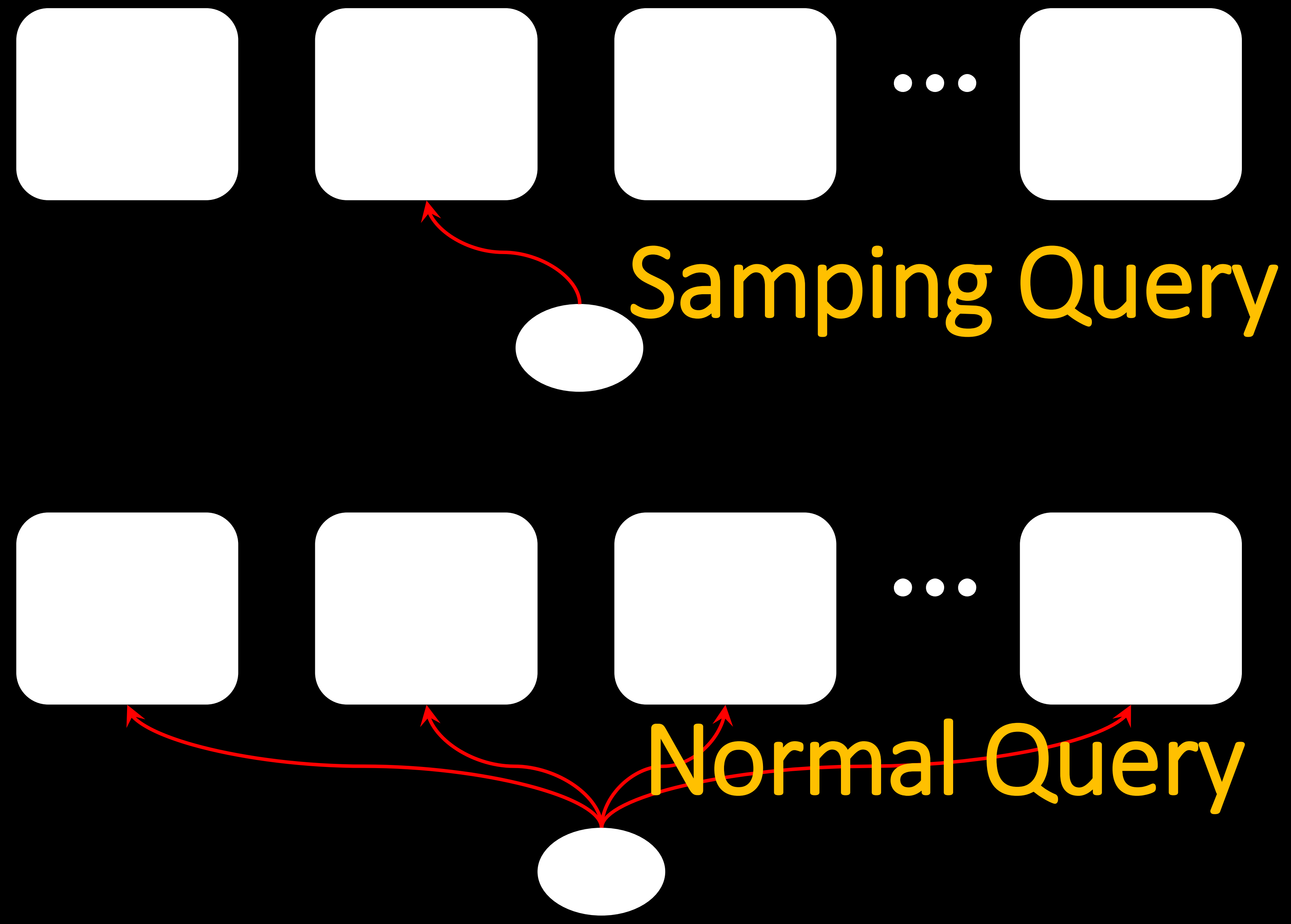
Query(2): Count Query

```
LemonTable lemonTable = new LemonTable(table);

LemonQuery query = LemonQuery.builder()
    .setQuery("City:Shenzhen AND (Age:10_20 OR Age:20_30) AND Occupation:Engineer")
    //Counting
    .setCountOnly()
    .addFamily(TableTmpl.FAM_M)
    .build();

ResultSet resultSet = lemonTable.query(query);

// Read count.
int count = resultSet.getCount();
```

Query(3): Sampling Query

The result of a random query for a data shard (normal query sends requests to all data shards):

```
LemonQuery query = LemonQuery.builder()
    .setQuery("City:Shenzhen AND (Age:10_20 OR Age:20_30)")
    .setSampling()
    .addFamily(TableTmpl.FAM_M)
    .setCaching(CACHING)
    .build();

ResultSet resultSet = lt.query(query);
// List all the caching rows.
List<EntityEntry> entries = resultSet.listRows();
```



Future

1. Better Bitmap Memory Management.
2. Range Query
3. ASync HBase client
4. Bitmap Calculation On FPGA

HBASECON ASIA2019

Lightweight SQL Engine – Lemon SQL

Zhi Liu

Huawei

Agenda

- Why Lemon SQL
- What to do, and what not to do
- How to do
- Lemon SQL current
- Lemon SQL future

Agenda

- Why lemon SQL
- What to do, and what not to do
- How to do
- Lemon SQL current
- Lemon SQL future

Why Lemon SQL?

Why not phoenix?

Why Lemon SQL?

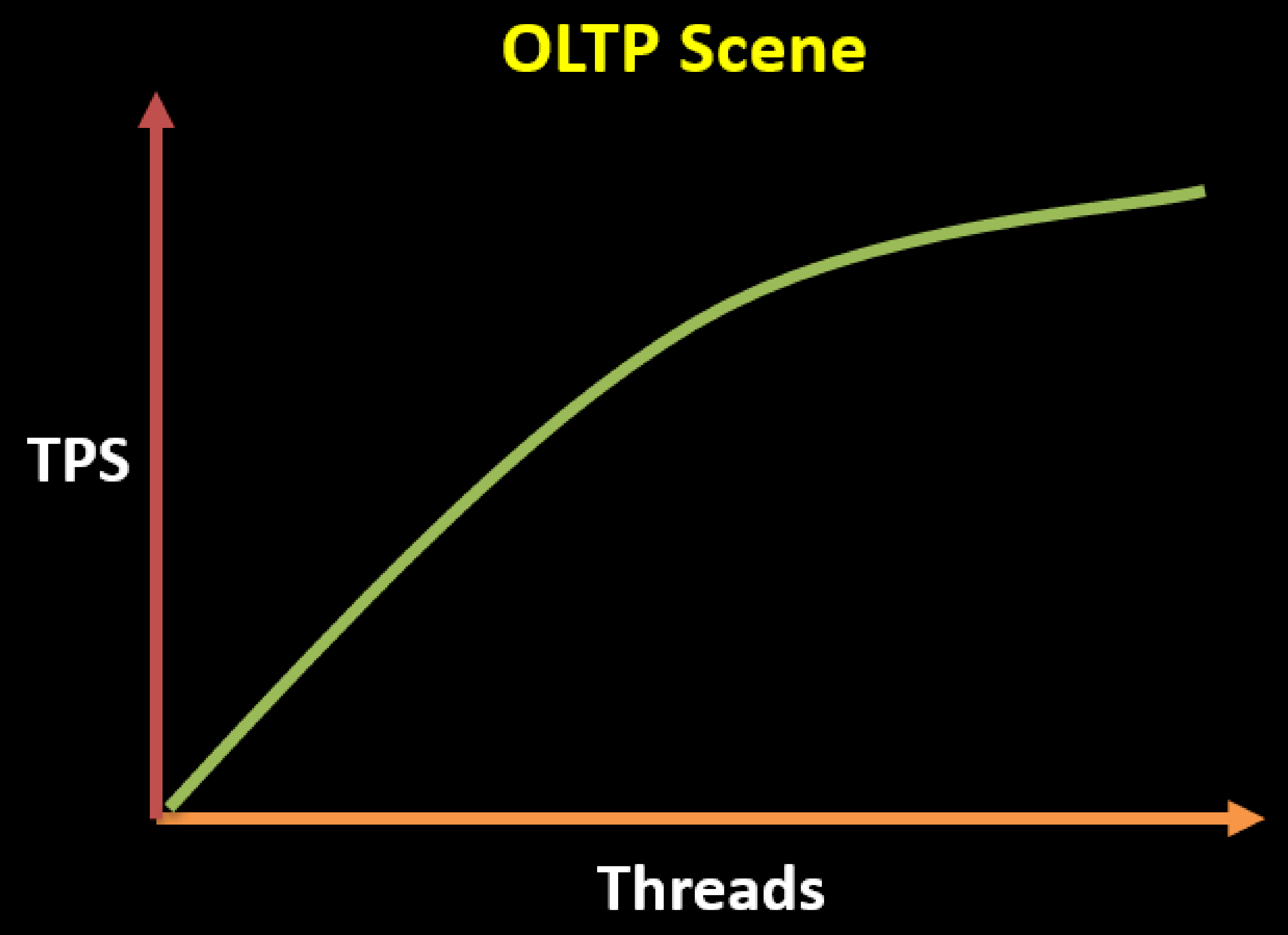
Problems of phoenix:

1. Too heavy

   Transaction, Join, View, Index, …
   Code lines: 30w+

2. Low performance on OLTP scene

3. Poor functional scalability



Agenda

- Why Lemon SQL
- What to do, and what not to do
- How to do
- Lemon SQL current
- Lemon SQL future

What to do, and what not to do

For OLTP, not for OLAP

Query Scene	Support?
SELECT * FROM table WHERE key = ‘value1’	YES
SELECT * FROM table WHERE key > ‘value1’ AND key < ‘value2’	YES
SELECT count(*) FROM table WHERE key > ‘value1’ AND key < ‘value2’ GROUP BY key	YES
…	
SELECT * FROM table	NO
SELECT * FROM table WHERE key = ‘value1’ ORDER BY col	NO
SELECT * FROM table1, table2 WHERE table1.key = ‘value1’ AND table1.key = table2.key	NO
SELECT count(*) FROM table WHERE key > ‘value1’ AND key < ‘value2’ GROUP BY col	NO
…	

Agenda

- Why Lemon SQL
- What to do, and what not to do
- How to do
- Lemon SQL current
- Lemon SQL future

How to do

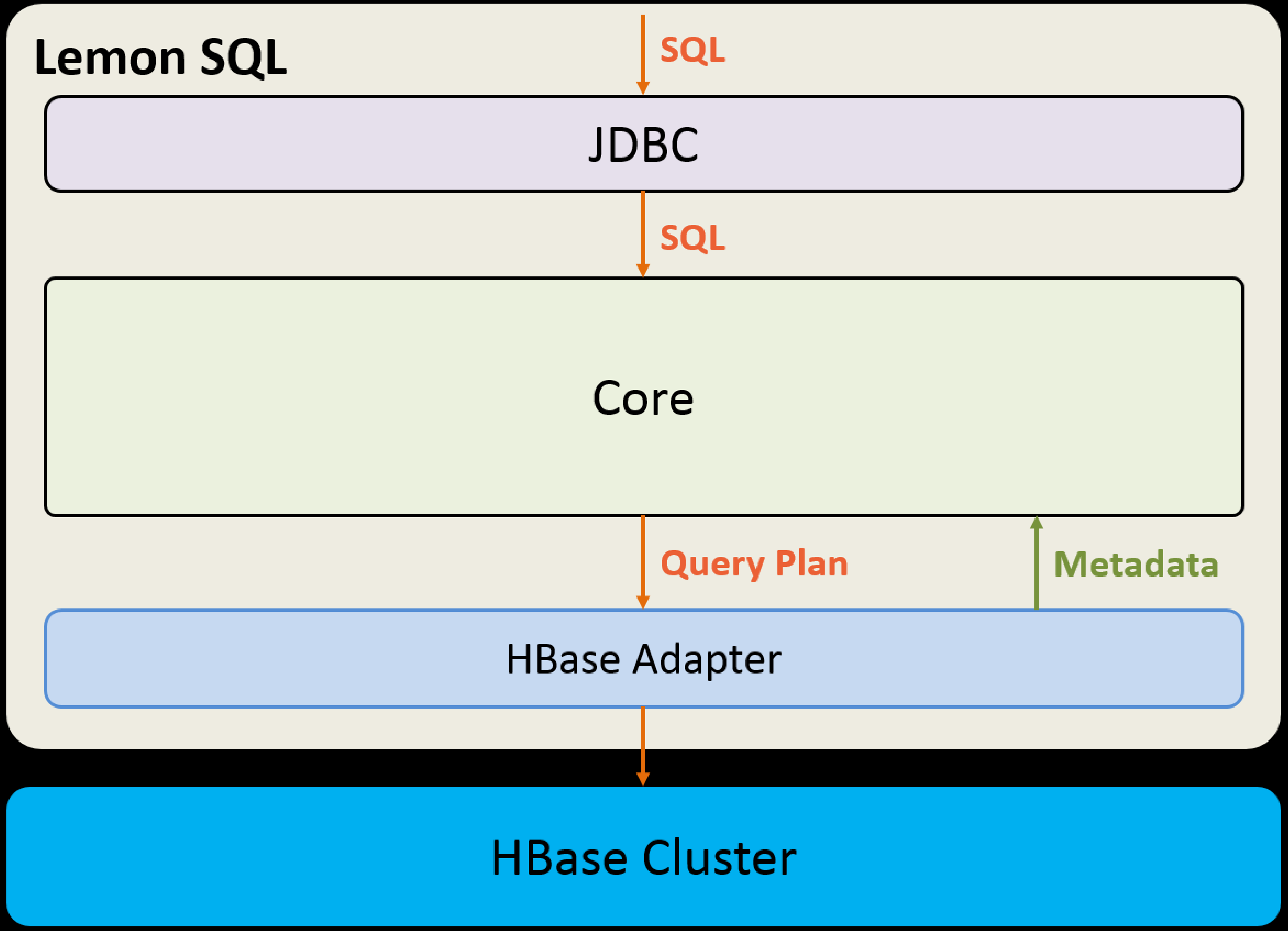
Key targets of new SQL engine:

- Lightweight
- High concurrency, high performance
- High functional scalability

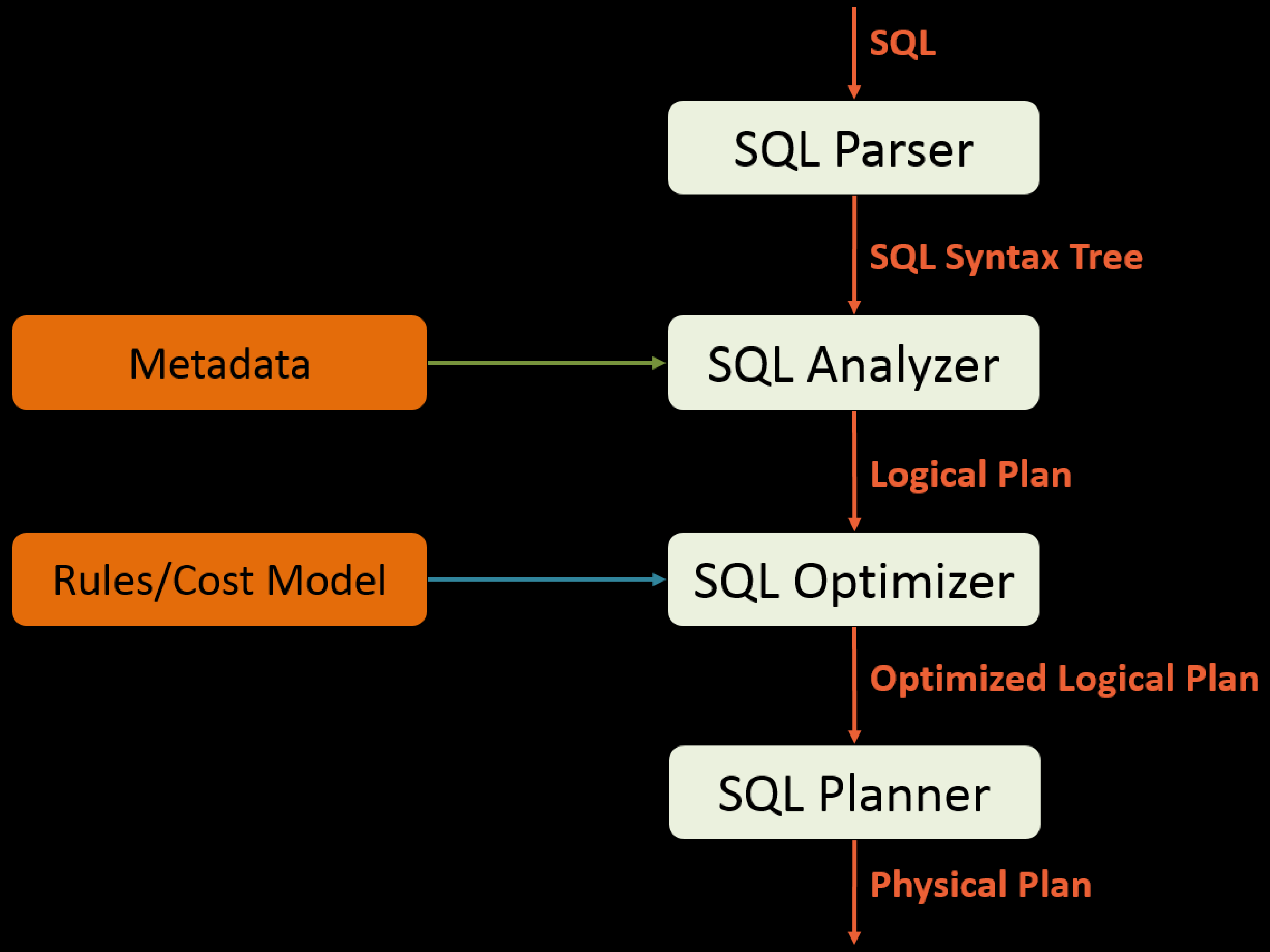
Agenda

- Why Lemon SQL
- What to do, and what not to do
- How to do
- **Lemon SQL current**
- Lemon SQL future

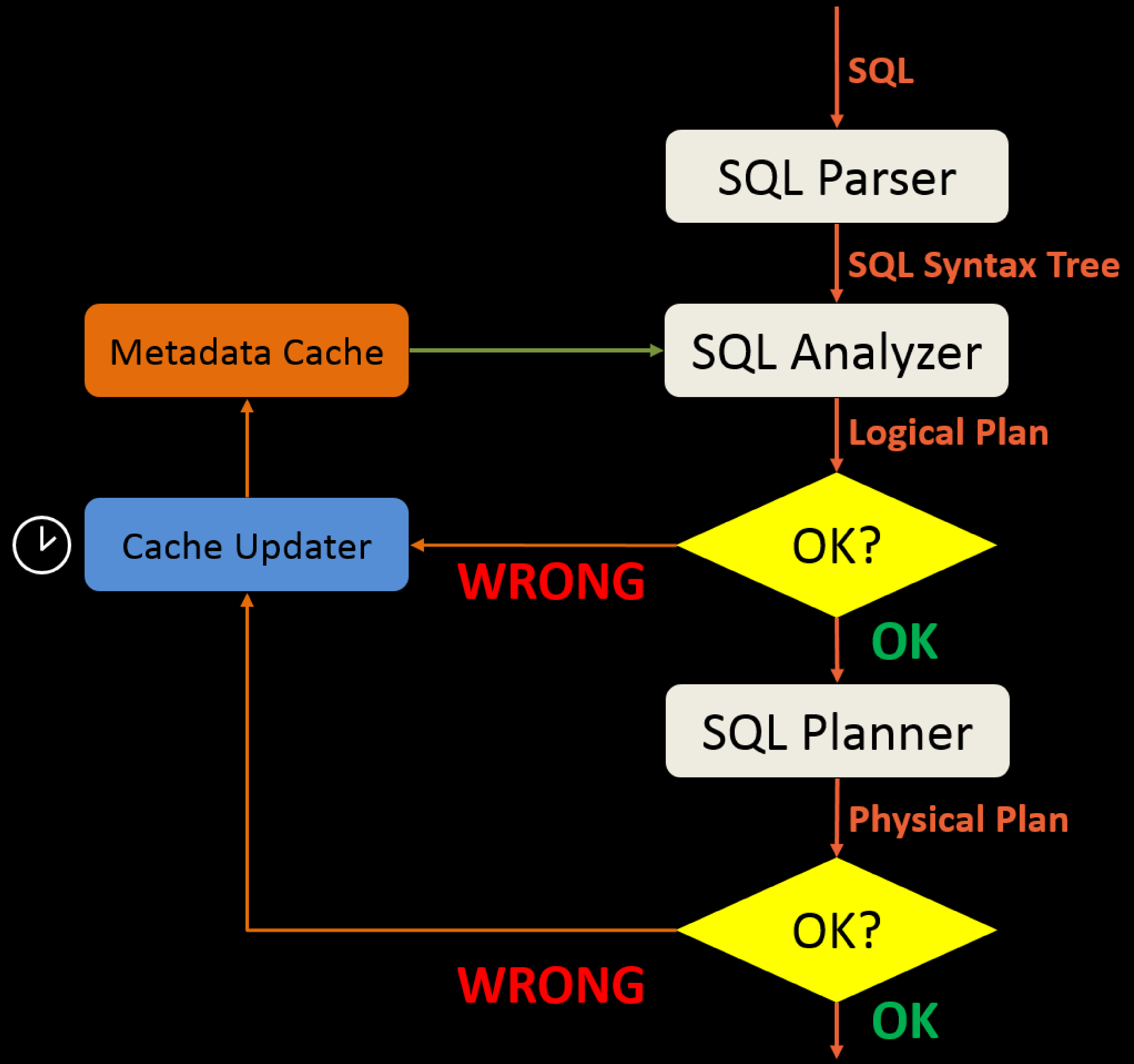
Architecture



Architecture



Metadata Cache Policy



Metadata Manage Policy

- Cannot update column value type
- The new column can only be a nullable column
- Cannot delete the primary key column
- …

Syntax Analyzer

	Grammar File	SQL Object Model	AST to SQL Object Converter	Time complexity
JavaCC	Required	Required	--	N
Antlr 4	Required	Required	Required	N
Lemon SQL	--	Required	--	1

Syntax Analyzer

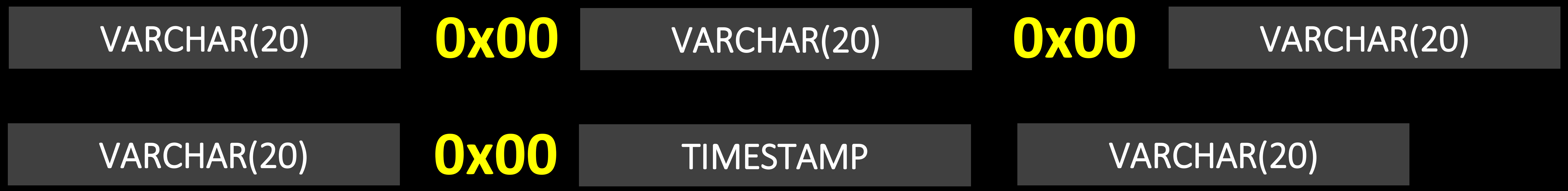
SELECT age + 1 FROM employee WHERE name='Zhang San'

1000000 times

	Used Time
JavaCC	138135ms
Antlr 4	17146ms
Lemon SQL	7594ms

Row Key Format

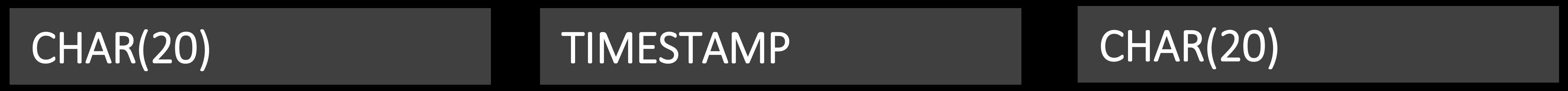
Separated by 0x00:



VARCHAR(20) 0x00 VARCHAR(20) 0x00 VARCHAR(20)

VARCHAR(20) 0x00 TIMESTAMP VARCHAR(20)

Alignment:



CHAR(20) TIMESTAMP CHAR(20)

```
CREATE TABLE test (col1 CHAR(20), col2 TIMESTAMP CONSTRAINT pk PRIMARY KEY (col1, col2))
OPTIONS ('row_key_codec_type' = 'ALIGNMENT')
```

Lemon SQL vs Phoenix

Test Data

```
CREATE TABLE monitor_data (
          device_id CHAR(5) NOT NULL,
          report_time VARCHAR NOT NULL,
          indicator0 BIGINT ,
          indicator1 BIGINT ,
          indicator2 BIGINT ,
          indicator3 BIGINT ,
          indicator4 BIGINT ,
          indicator5 BIGINT ,
          indicator6 BIGINT ,
          indicator7 BIGINT ,
          indicator8 BIGINT ,
          indicator9 BIGINT
          CONSTRAINT pk PRIMARY KEY (device_id, report_time)
)
```

20000000 rows

Lemon SQL vs Phoenix

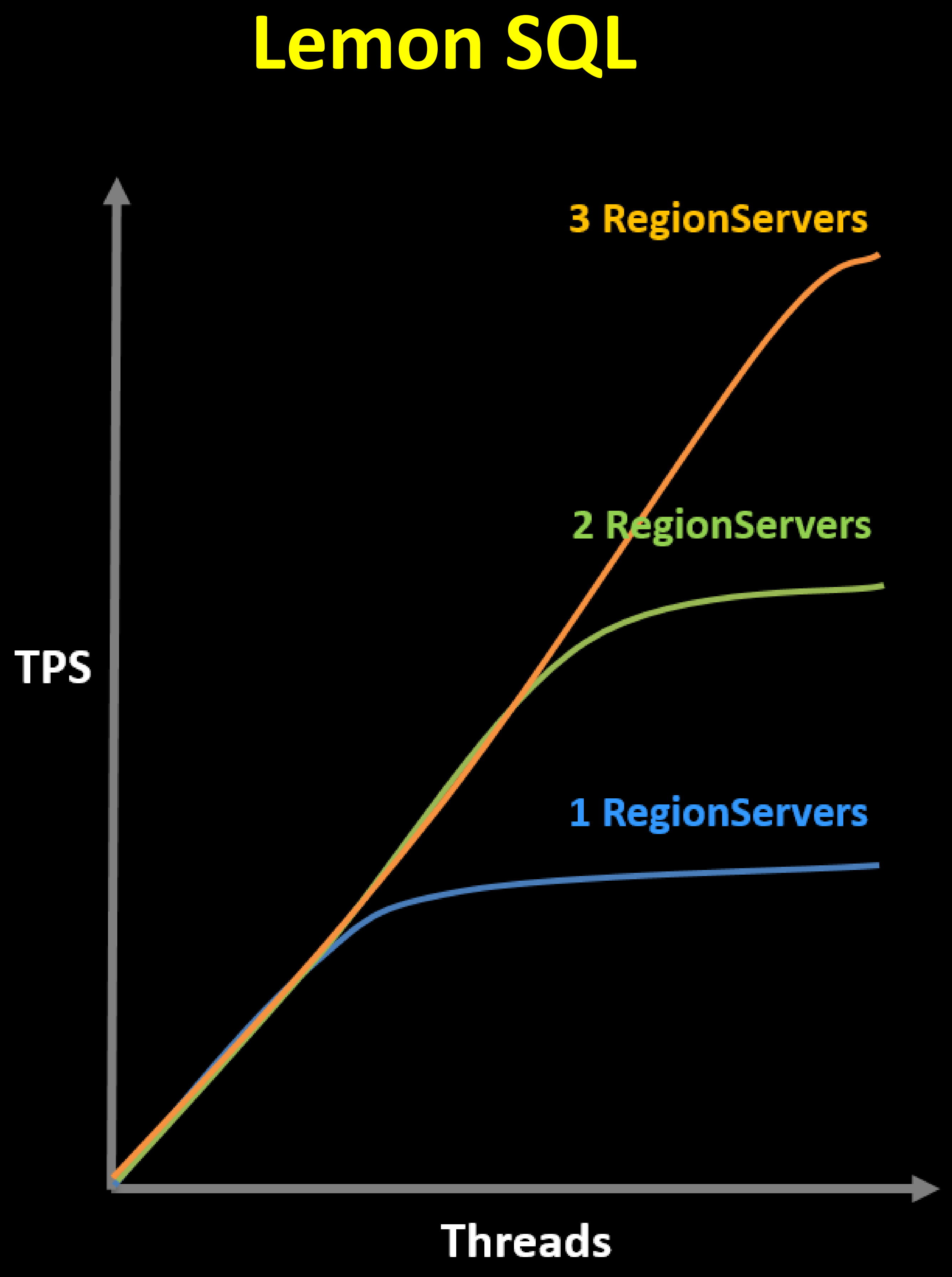
	Lemon SQL	Phoenix
Code lines	5w+	30w+

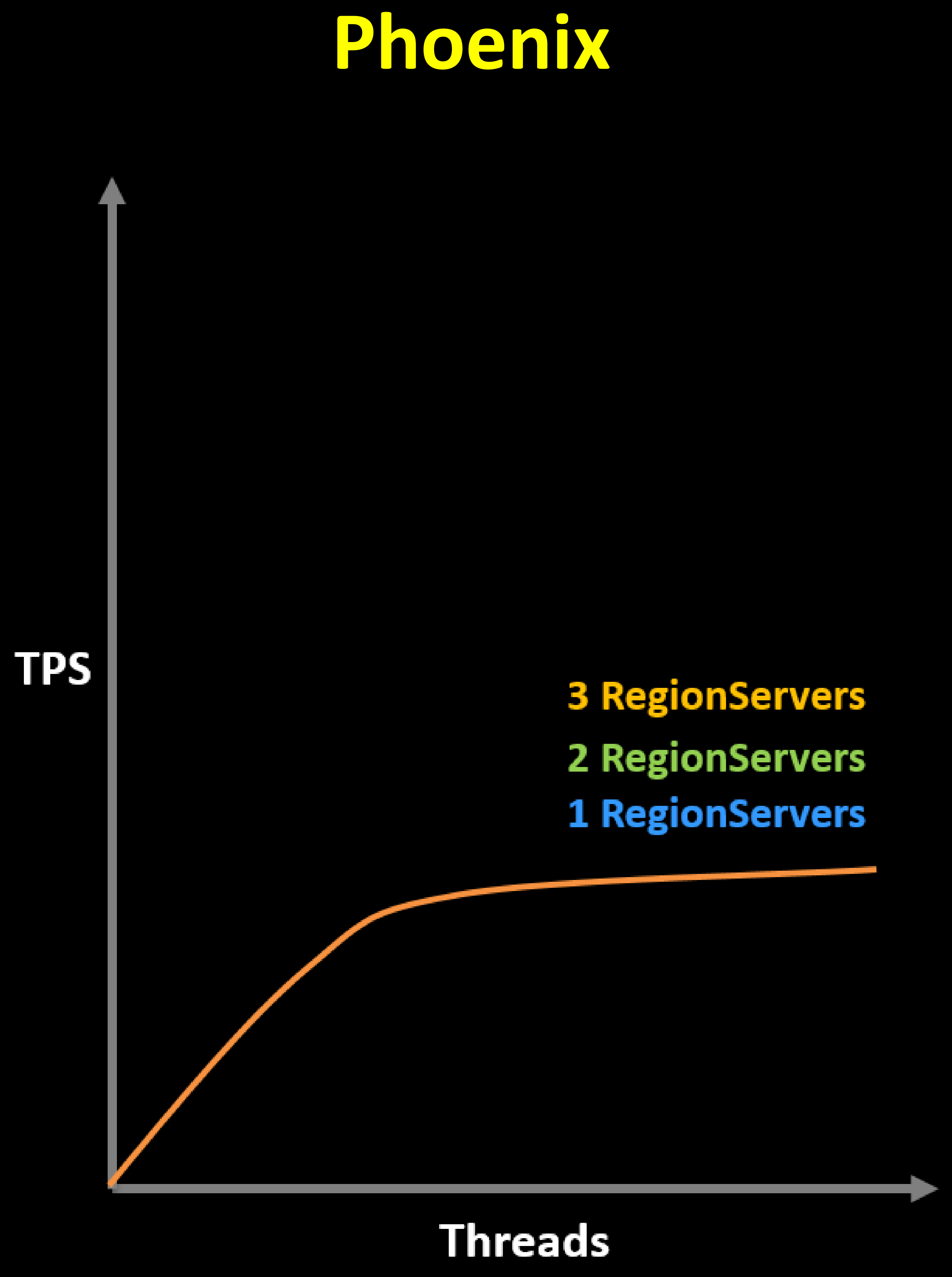
1 Thread 1 Row
SELECT * FROM monitor_data
WHERE device_id = ‘00100’ AND report_time = ‘2019-06-20 00:00:00’
	2680 TPS	690 TPS

1 Thread 10 Rows
SELECT * FROM monitor_data
WHERE device_id = ‘00100’ AND report_time > ‘2019-06-20 00:00:00’ AND report_time <= ‘2019-06-20 00:01:00’
	1770 TPS	630 TPS

1 Thread 200 Rows
SELECT * FROM monitor_data
WHERE device_id = ‘00100’
	280 TPS	237 TPS

Lemon SQL vs Phoenix





Agenda

- Why Lemon SQL
- What to do, and what not to do
- How to do
- Lemon SQL current
- Lemon SQL future

Future

- Full text index
- Row to column
- ……

Future — Row to column

Table Definition:

```
CREATE TABLE person (id CHAR(20), gender VARCHAR CONSTRAINT pk PRIMARY KEY (id))
```

HBase Table Structure:

Row Key	F:gender_male	F:gender_female

Query:

```
SELECT id FROM person WHERE gender = ‘male’
```

Physical Plan:

```
Scan (F:gender_male)
```

HUAWEI

华为云

MRS

表格存储服务

Zhi Liu

liuzhi5@huawei.com

Xingjun Hao

haoxingjun@huawei.com

Thanks !