Apache Phoenix: Transforming HBase into a SQL database

James Taylor
@JamesPlusPlus
http://phoenix.apache.org

About me

- Architect at Salesforce.com in Big Data group
  - Started Phoenix as internal project ~3 years ago
  - Open-source on Github ~1.5 years ago
  - Apache incubator for ~5 months
  - Graduated as Top Level Project in May 2014
- Engineer at BEA Systems
  - XQuery-based federated query engine
  - SQL-based complex event processing engine

Agenda

- What is Apache Phoenix?
- Why is it so fast?
- How does it help HBase scale?
- Roadmap
- Q&A

What is Apache Phoenix?

What is Apache Phoenix?

1. Turns HBase into a SQL database
    - Query Engine
    - MetaData Repository
    - Embedded JDBC driver
    - **Only** for HBase data

What is Apache Phoenix?

2. Fastest way to access HBase data
   - HBase-specific push down
   - Compiles queries into native HBase calls (no map-reduce)
   - Executes scans in parallel

SELECT * FROM t WHERE k IN (?,?,?)

Phoenix	Stinger (Hive 0.11)
0.04 sec	280 sec

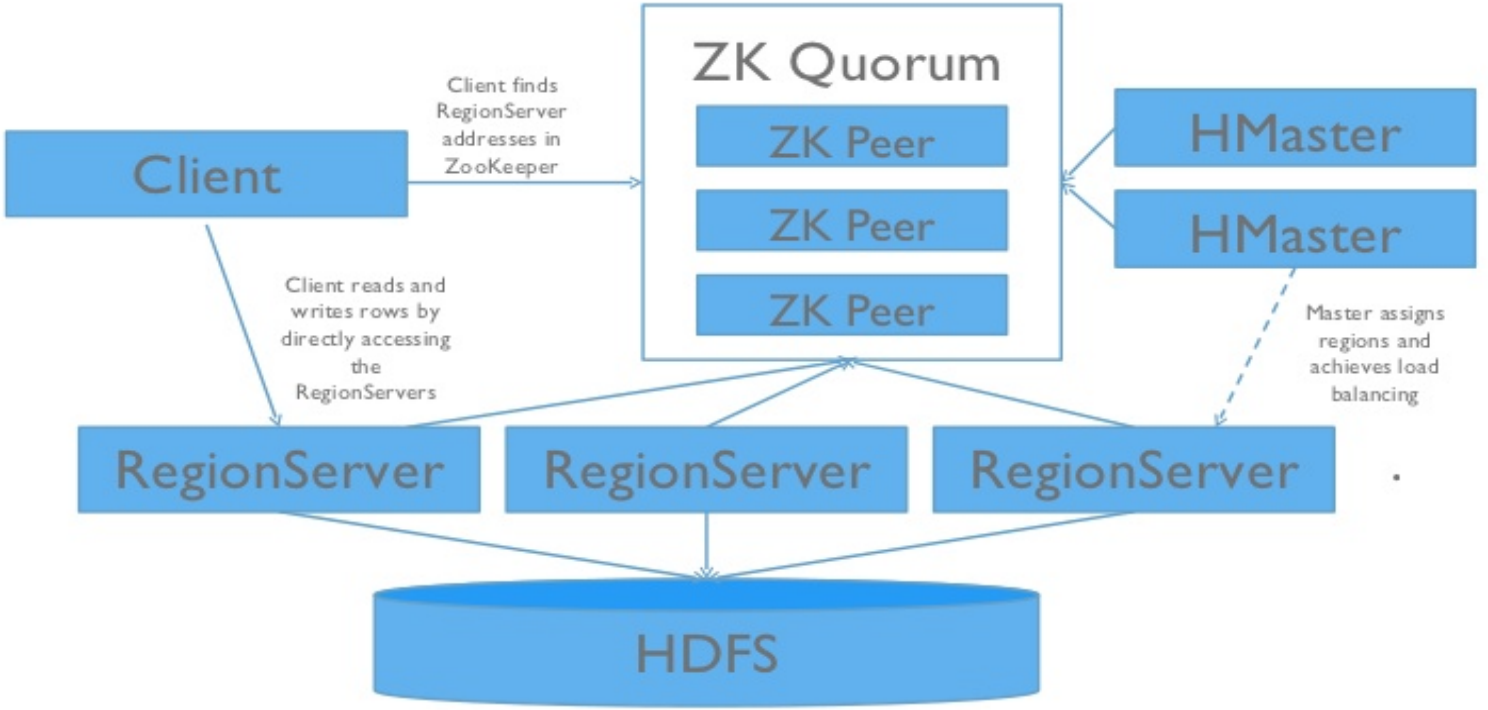
7,000x faster

* 110M row table

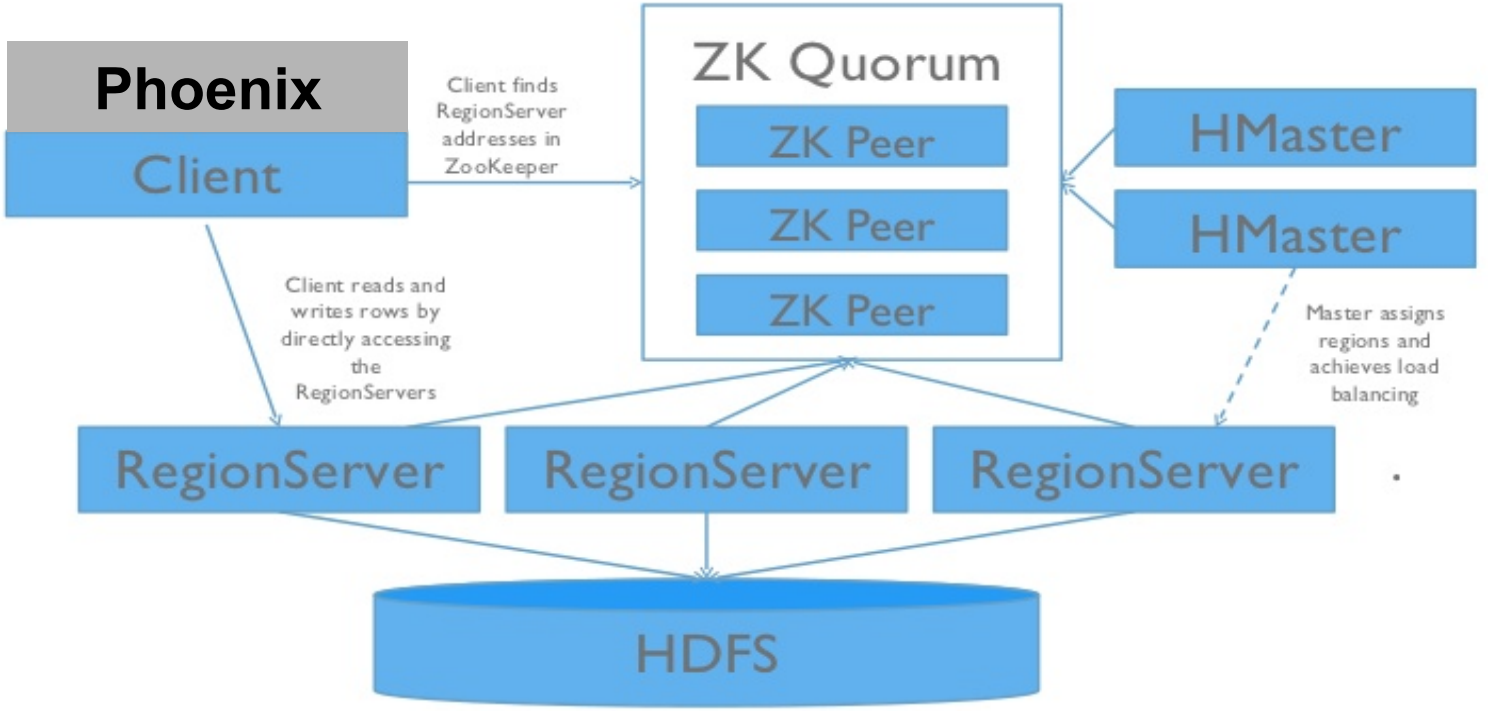
What is Apache Phoenix?

3. Lightweight
   - No additional servers required
   - 100% Java
   - Included in HDP 2.1 distribution
   - Available in Amazon EMR
   - Otherwise copy Phoenix jar into HBase lib directory on each RS

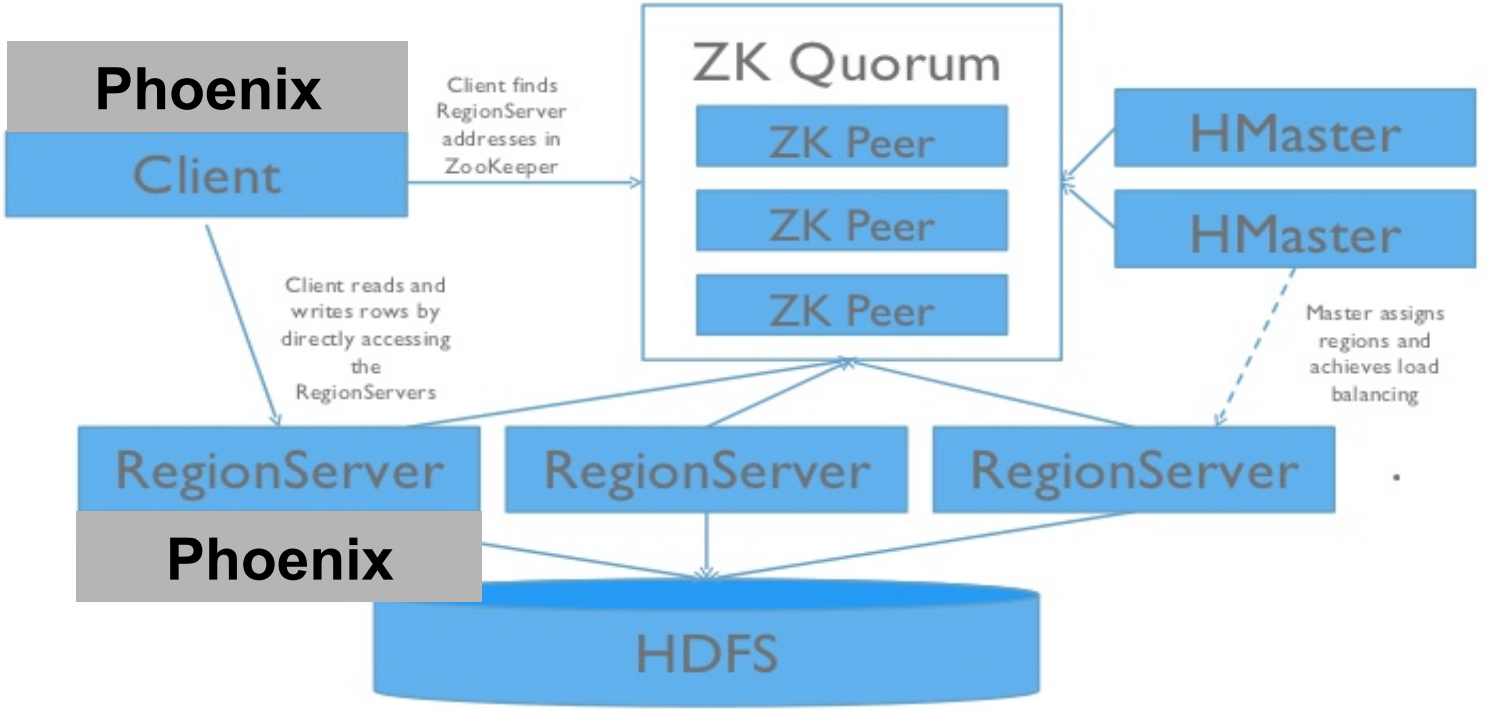
HBase Cluster Architecture



HBase Cluster Architecture



HBase Cluster Architecture



What is Apache Phoenix?

4. Integration-friendly
   - Map to existing HBase table
   - Integrate with Apache Pig
   - Integrate with Apache Flume
   - Integrate with Apache Sqoop (wip)

What is Apache Phoenix?

1. Turns HBase into a SQL database
2. Fastest way to access HBase data
3. Lightweight
4. Integration-friendly

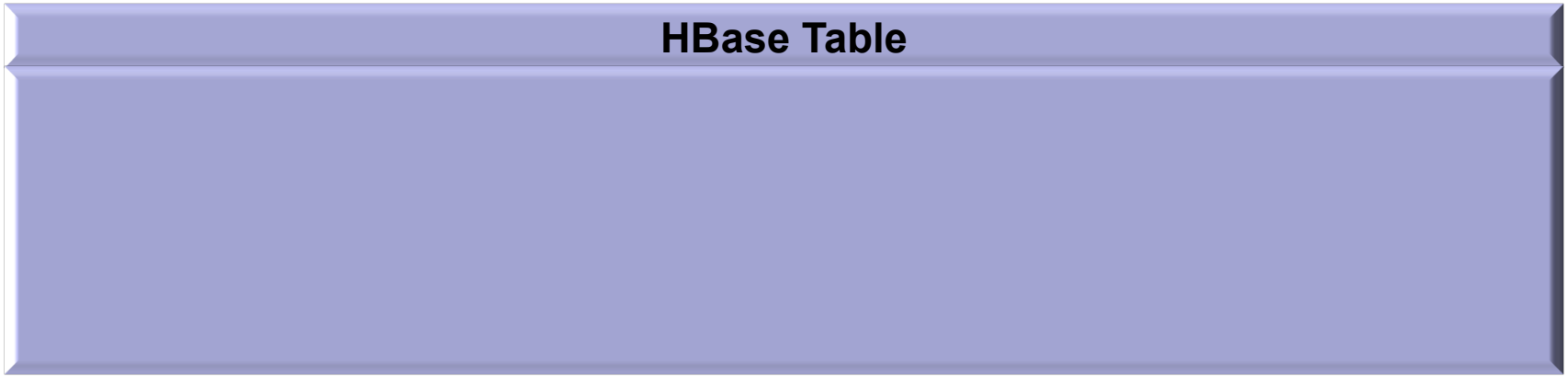
Why is Phoenix so fast?

Why is Phoenix so fast?

1. HBase
   - Fast, but “dumb” (on purpose)
2. Data model
   - Support for composite primary key
   - Binary data sorts naturally
3. Client-side parallelization
4. Push down
   - Custom filters and coprocessors

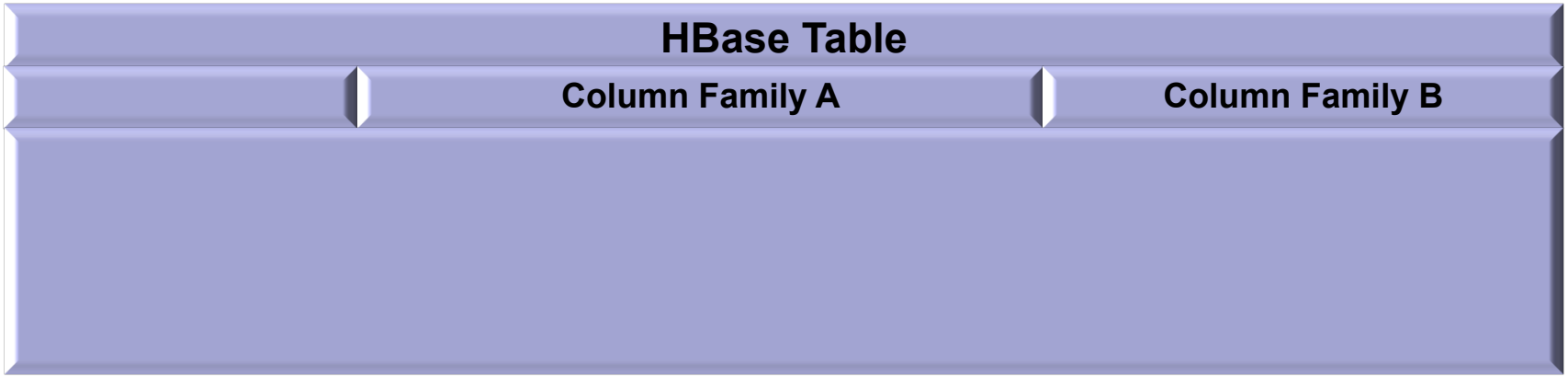
Phoenix Data Model

Phoenix maps HBase data model to the relational world



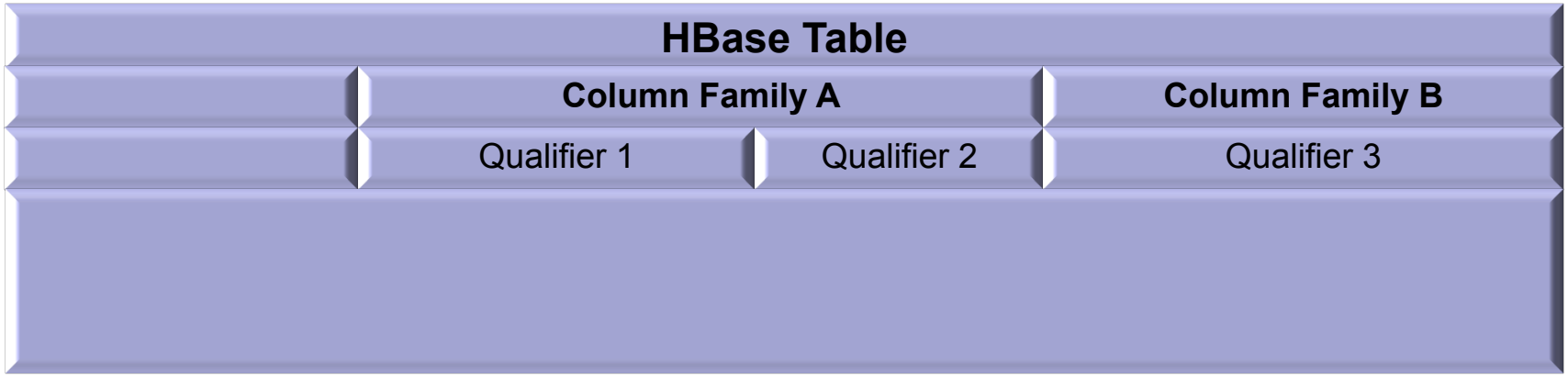
Phoenix Data Model

Phoenix maps HBase data model to the relational world



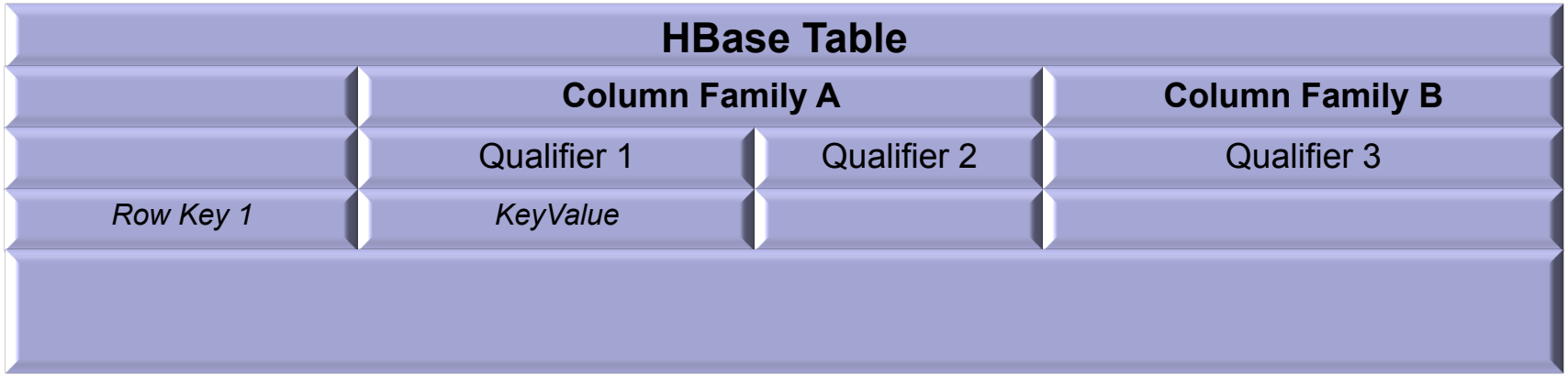
Phoenix Data Model

Phoenix maps HBase data model to the relational world



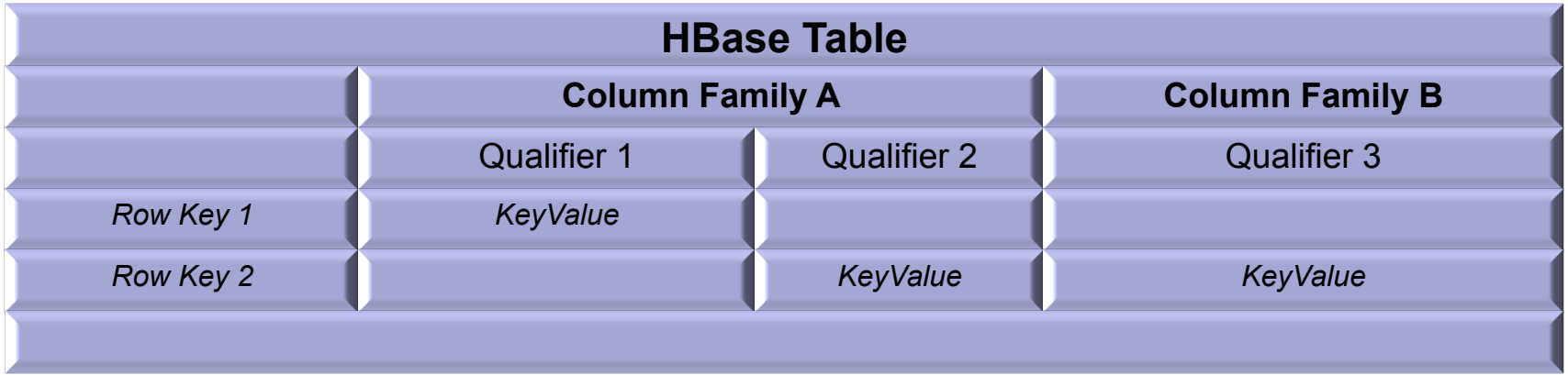
Phoenix Data Model

Phoenix maps HBase data model to the relational world



Phoenix Data Model

Phoenix maps HBase data model to the relational world



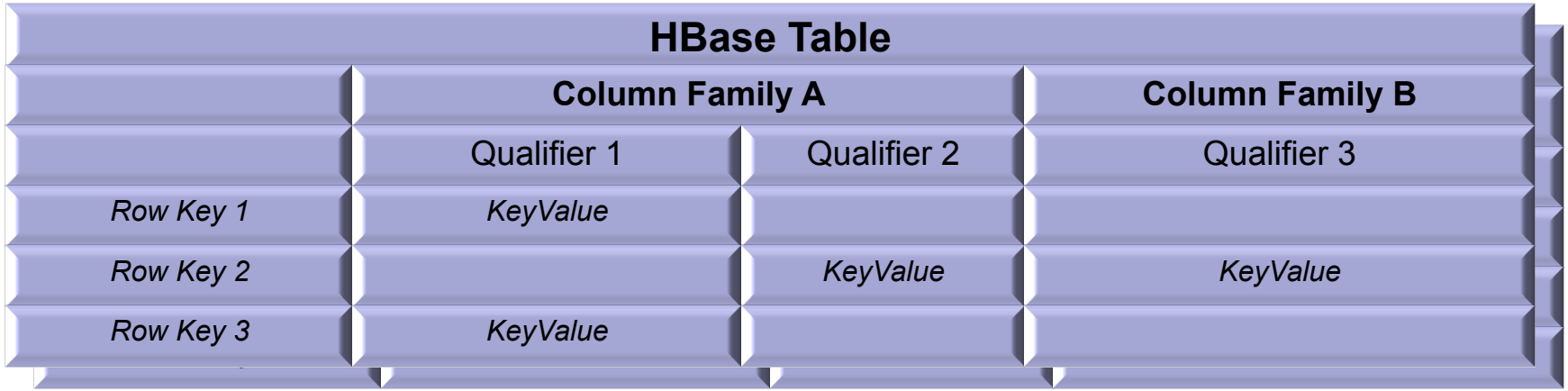
Phoenix Data Model

Phoenix maps HBase data model to the relational world

HBase Table			
	Column Family A		**Column Family B**
	Qualifier 1	Qualifier 2	Qualifier 3
Row Key 1	*KeyValue*		
Row Key 2		*KeyValue*	*KeyValue*
Row Key 3	*KeyValue*		

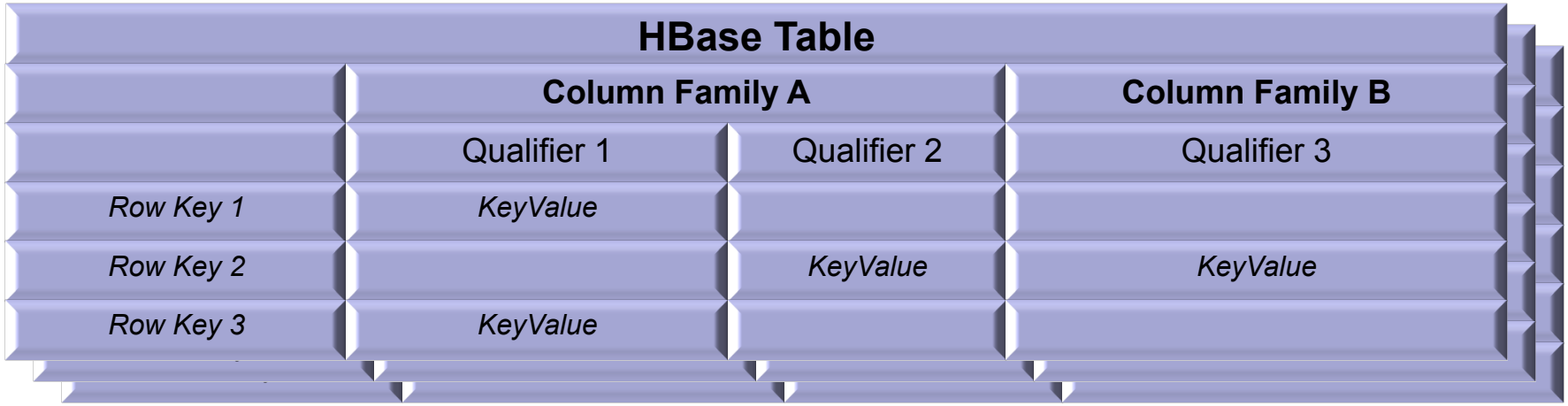
Phoenix Data Model

Phoenix maps HBase data model to the relational world



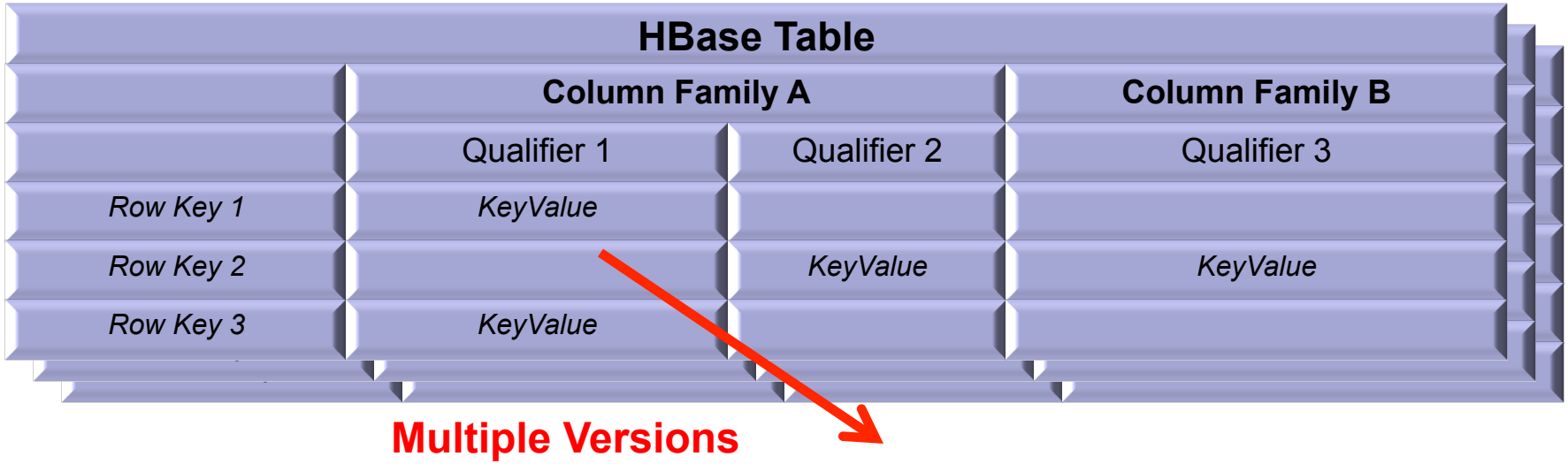
Phoenix Data Model

Phoenix maps HBase data model to the relational world



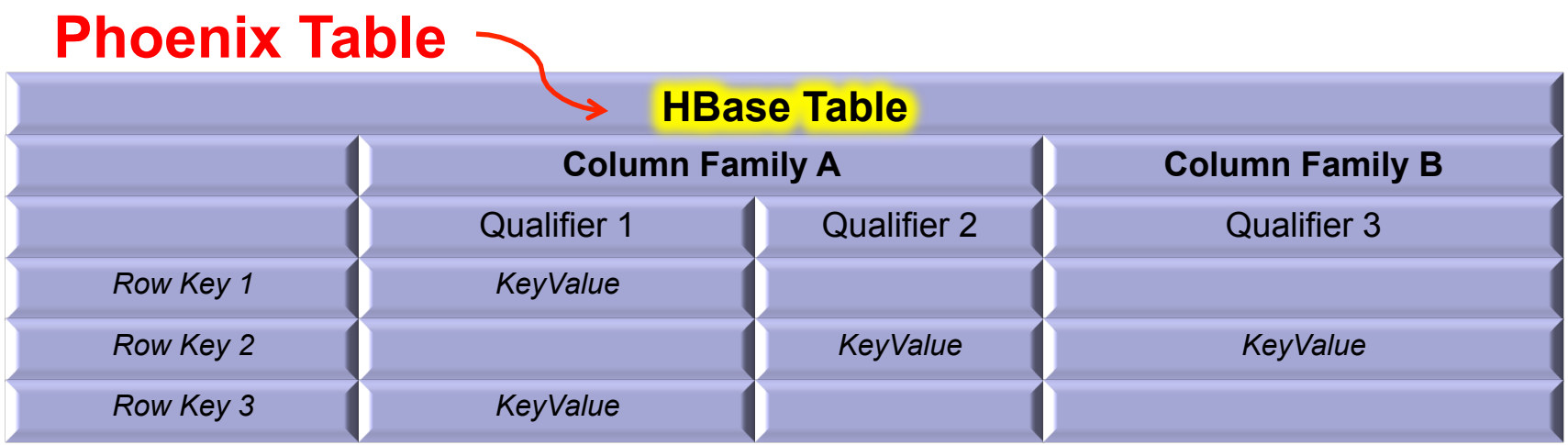
Phoenix Data Model

Phoenix maps HBase data model to the relational world



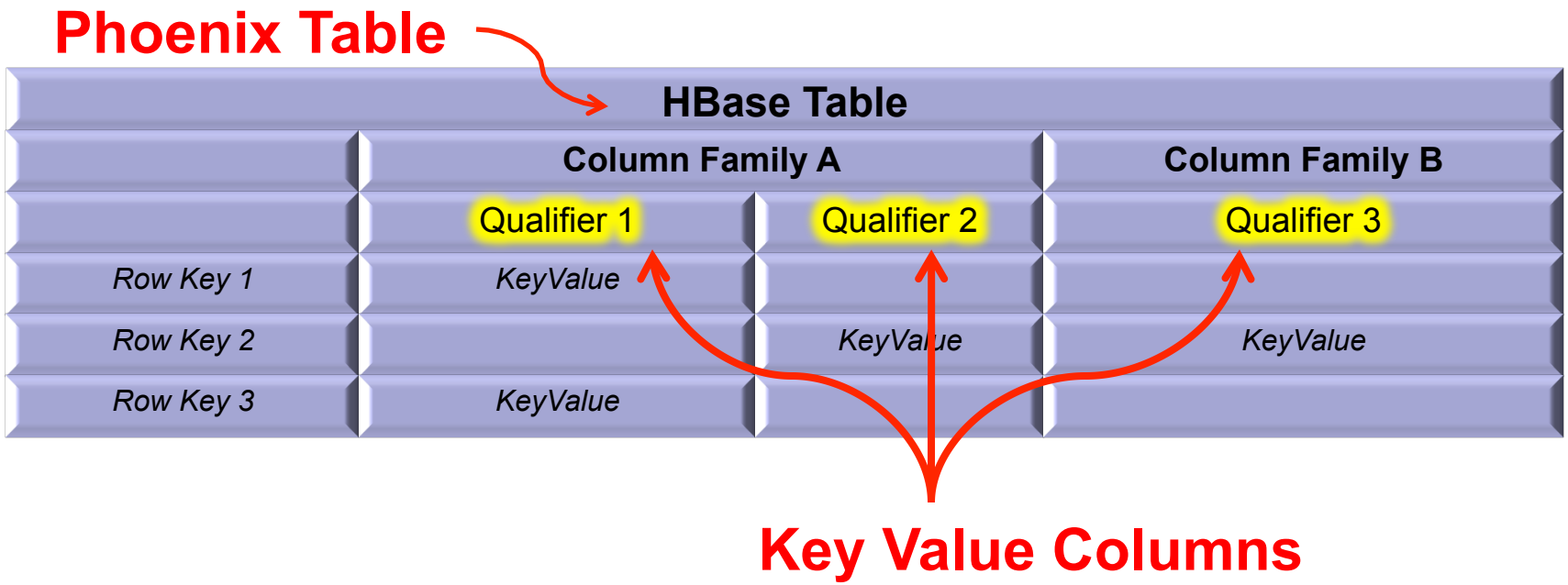
Phoenix Data Model

Phoenix maps HBase data model to the relational world



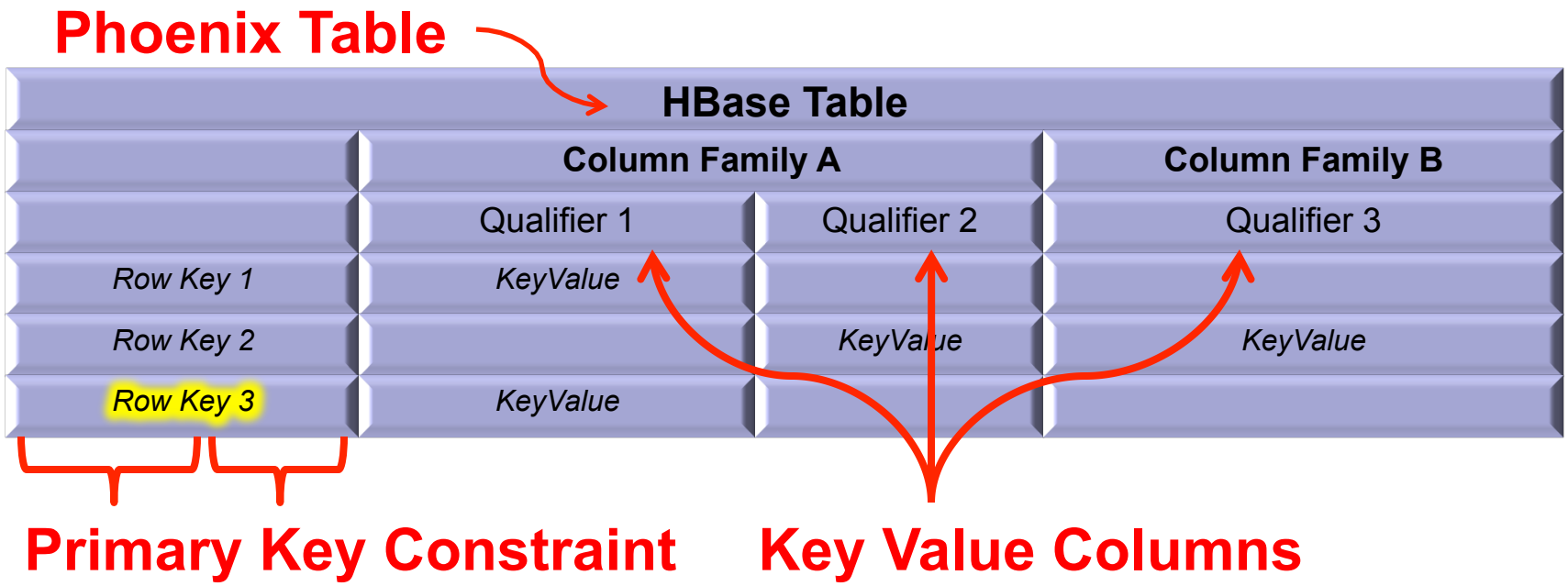
Phoenix Data Model

Phoenix maps HBase data model to the relational world



Phoenix Data Model

Phoenix maps HBase data model to the relational world



Example

Over metrics data for servers with a schema like this:

SERVER METRICS	
HOST	VARCHAR
DATE	DATE
RESPONSE_TIME	INTEGER
GC_TIME	INTEGER
CPU_TIME	INTEGER
IO_TIME	INTEGER

Row Key

Example

Over metrics data for servers with a schema like this:

SERVER METRICS	
HOST	VARCHAR
DATE	DATE
RESPONSE_TIME	INTEGER
GC_TIME	INTEGER
CPU_TIME	INTEGER
IO_TIME	INTEGER

Key Values

Example

DDL command looks like this:

```
CREATE TABLE SERVER_METRICS (
        HOST                VARCHAR,
        DATE                DATE,
        RESPONSE_TIME       INTEGER,
        GC_TIME             INTEGER,
        CPU_TIME            INTEGER,
        IO_TIME             INTEGER,
        CONSTRAINT pk PRIMARY KEY (HOST, DATE))
```

Example

With data that looks like this:

SERVER METRICS			
HOST + DATE		RESPONSE_TIME	GC_TIME
SF1	1396743589	1234	
SF1	1396743589		8012
…			
SF3	1396002345	2345	
SF3	1396002345		2340
SF7	1396552341	5002	1234
…			

Row Key

Example

With data that looks like this:

SERVER METRICS			
HOST + DATE		RESPONSE_TIME	GC_TIME
SF1	1396743589	1234	
SF1	1396743589		8012
…			
SF3	1396002345	2345	
SF3	1396002345		2340
SF7	1396552341	5002	1234
…			

Key Values

Phoenix Push Down: Example

```
SELECT host, avg(response_time)
FROM server_metrics
WHERE date > CURRENT_DATE() – 7
AND host LIKE ‘SF%’
GROUP BY host
```

Phoenix Push Down: Example

```
SELECT host, avg(response_time)
FROM server_metrics
WHERE date > CURRENT_DATE() – 7
AND host LIKE ‘SF%’
GROUP BY host
```

Phoenix Push Down: Example

```
SELECT host, avg(response_time)
FROM server_metrics
WHERE date > CURRENT_DATE() – 7
AND host LIKE ‘SF%’
GROUP BY host
```

Phoenix Push Down: Example

```
SELECT host, avg(response_time)
FROM server_metrics
WHERE date > CURRENT_DATE() – 7
AND host LIKE ‘SF%’
GROUP BY host
```

Phoenix Push Down: Example

```
SELECT host, avg(response_time)
FROM server_metrics
WHERE date > CURRENT_DATE() – 7
AND host LIKE ‘SF%’
GROUP BY host
```

Phoenix Push Down

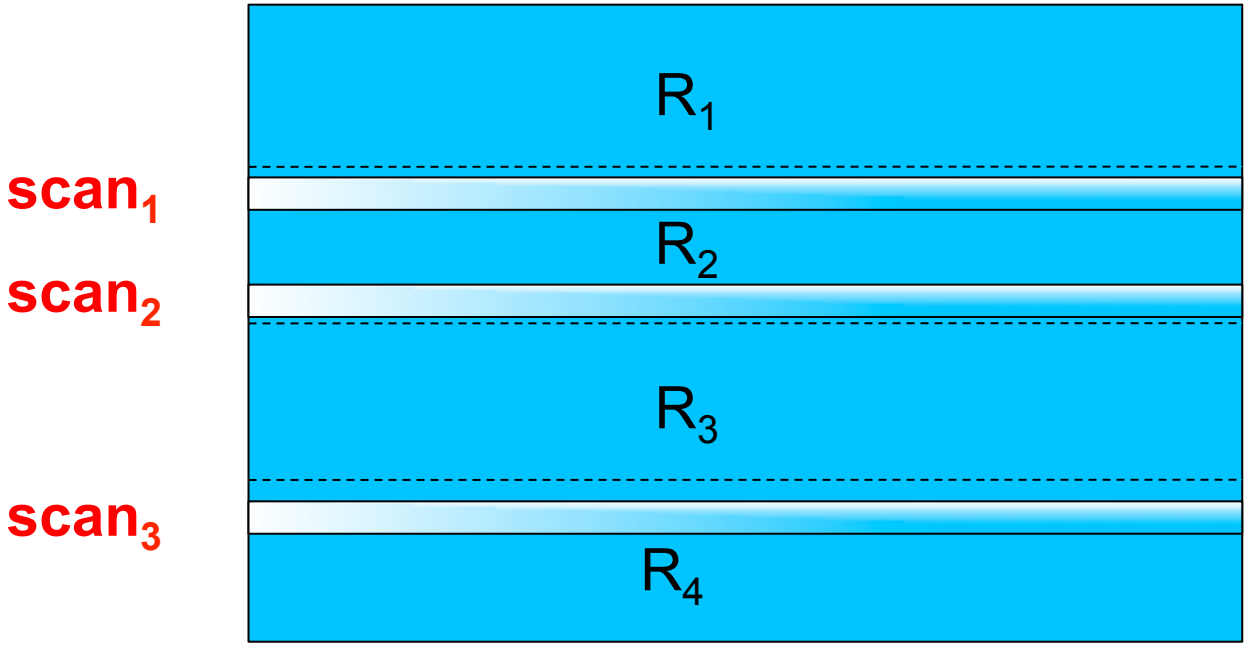
1. Skip scan filter
2. Aggregation
3. TopN
4. Hash Join

Phoenix Push Down: Skip scan

```
SELECT host, avg(response_time)
FROM server_metrics
WHERE date > CURRENT_DATE() – 7
AND host LIKE ‘SF%’
GROUP BY host
```

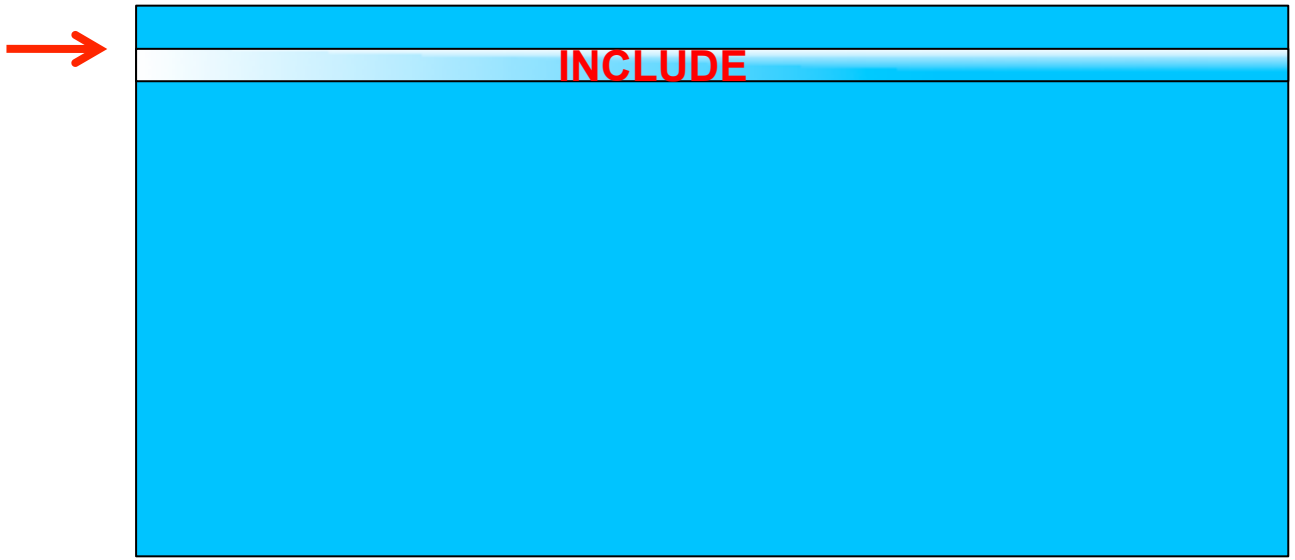
Phoenix Push Down: Skip scan

Phoenix Push Down: Skip scan
Client-side parallel scans



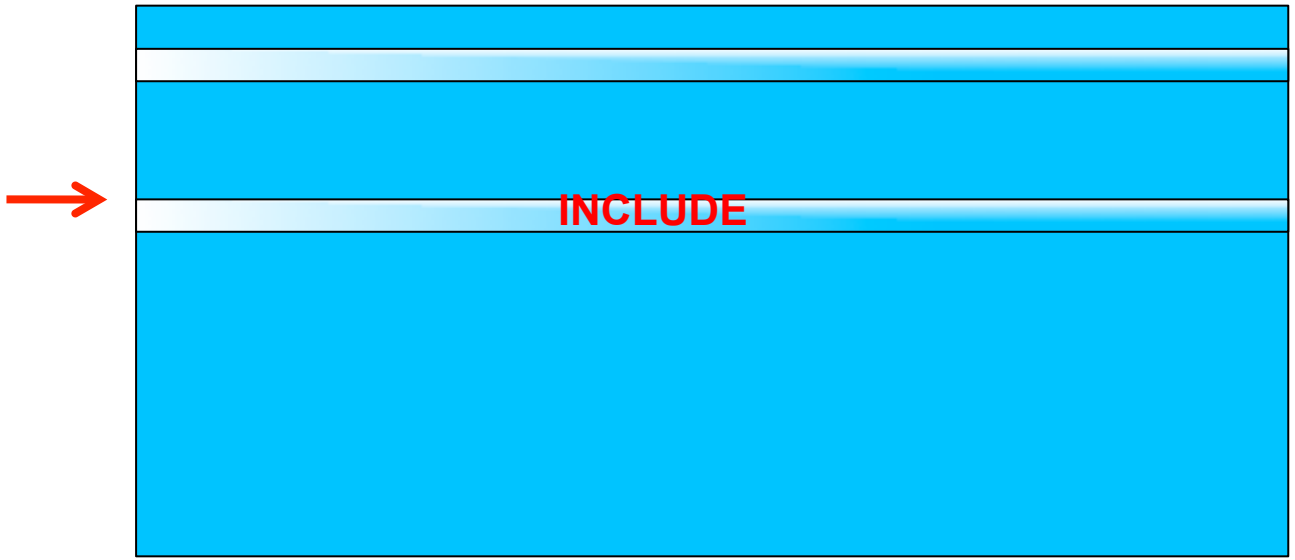
Phoenix Push Down: Skip scan Server-side filter

Phoenix Push Down: Skip scan Server-side filter



Phoenix Push Down: Skip scan Server-side filter

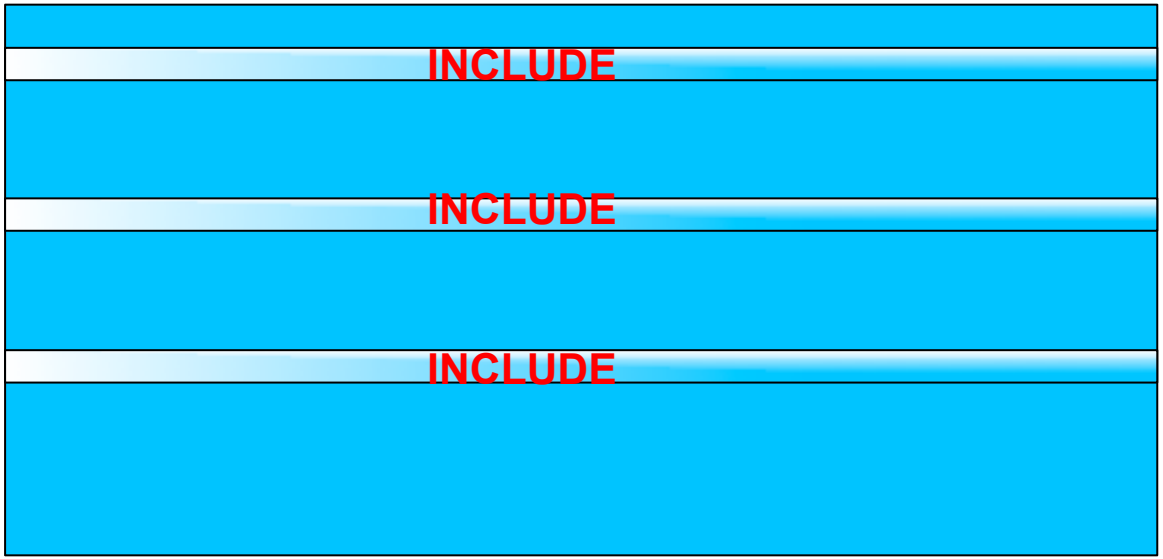
Phoenix Push Down: Skip scan Server-side filter



Phoenix Push Down: Skip scan Server-side filter

Phoenix Push Down: Skip scan Server-side filter

Phoenix Push Down: Skip scan Server-side filter



Phoenix Push Down: Aggregation

```
SELECT host, avg(response_time)
FROM server_metrics
WHERE date > CURRENT_DATE() – 7
AND host LIKE ‘SF%’
GROUP BY host
```

Phoenix Push Down: Aggregation
Aggregate on server-side

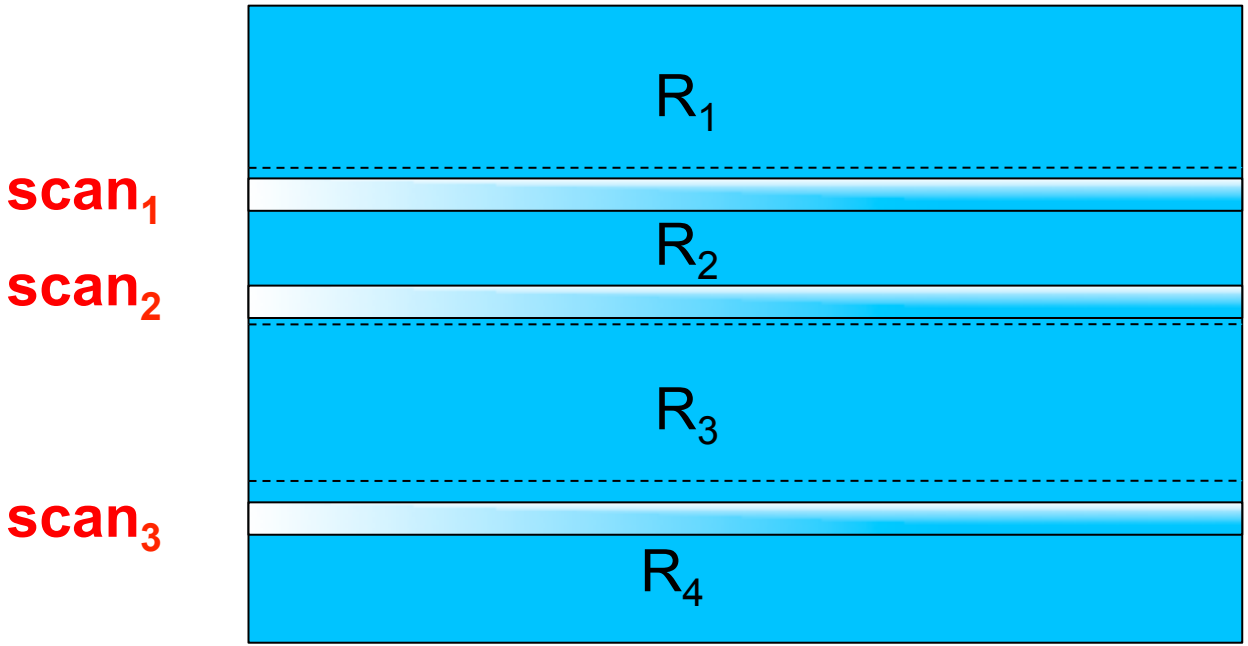
SERVER METRICS				
HOST	DATE	KV_1	KV_2	KV_3
SF1	Jun 2 10:10:10.234	239	234	674
SF1	Jun 3 23:05:44.975		23	234
SF1	Jun 9 08:10:32.147	256	314	341
SF1	Jun 9 08:10:32.147	235	256	
SF1	Jun 1 11:18:28.456		235	23
SF1	Jun 3 22:03:22.142	234		314
SF1	Jun 3 22:03:22.142	432	234	256
SF2	Jun 1 10:29:58.950	23	432	
SF2	Jun 2 14:55:34.104	314	876	23
SF2	Jun 3 12:46:19.123	256	234	314
SF2	Jun 3 12:46:19.123		432	
SF2	Jun 8 08:23:23.456	876	876	235
SF2	Jun 1 10:31:10.234	234	234	876
SF3	Jun 1 10:31:10.234	432	432	234
SF3	Jun 3 10:31:10.234		890	
SF3	Jun 8 10:31:10.234	314	314	235
SF3	Jun 1 10:31:10.234	256	256	876
SF3	Jun 1 10:31:10.234	235		234
SF3	Jun 8 10:31:10.234	876	876	432
SF3	Jun 9 10:31:10.234	234	234	
SF3	Jun 3 10:31:10.234		432	276
…	…	…	…	…

SERVER METRICS	
HOST	AGGREGATE VALUES
SF1	3421
SF2	2145
SF3	9823

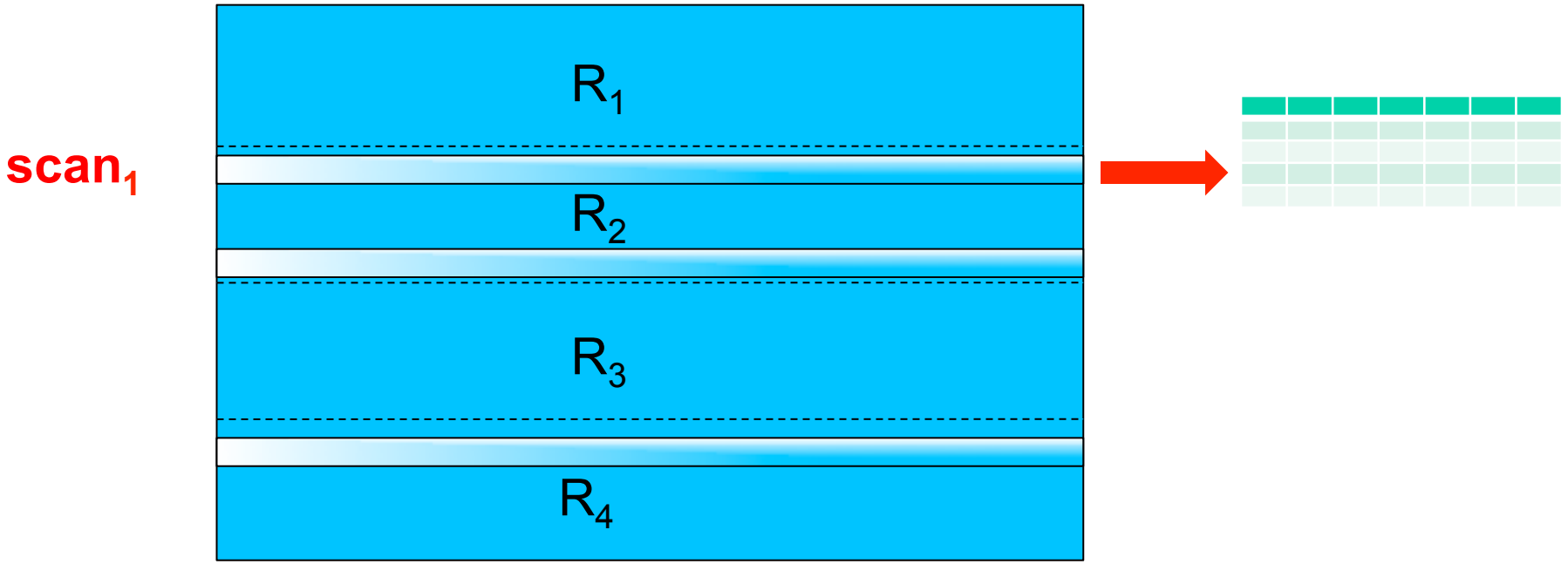
Phoenix Push Down: TopN

```
SELECT host, date, gc_time
FROM server_metrics
WHERE date > CURRENT_DATE() – 7
AND host LIKE ‘SF%’
ORDER BY gc_time DESC
LIMIT 5
```

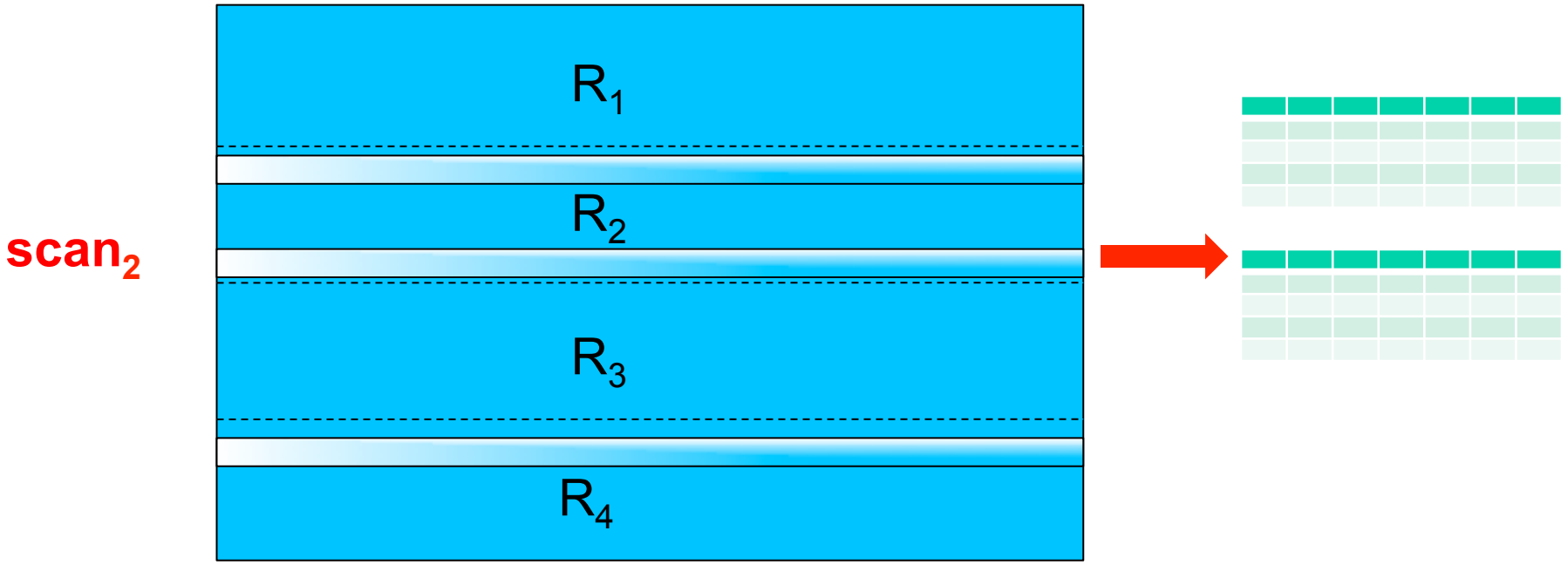
Phoenix Push Down: TopN
Client-side parallel scans



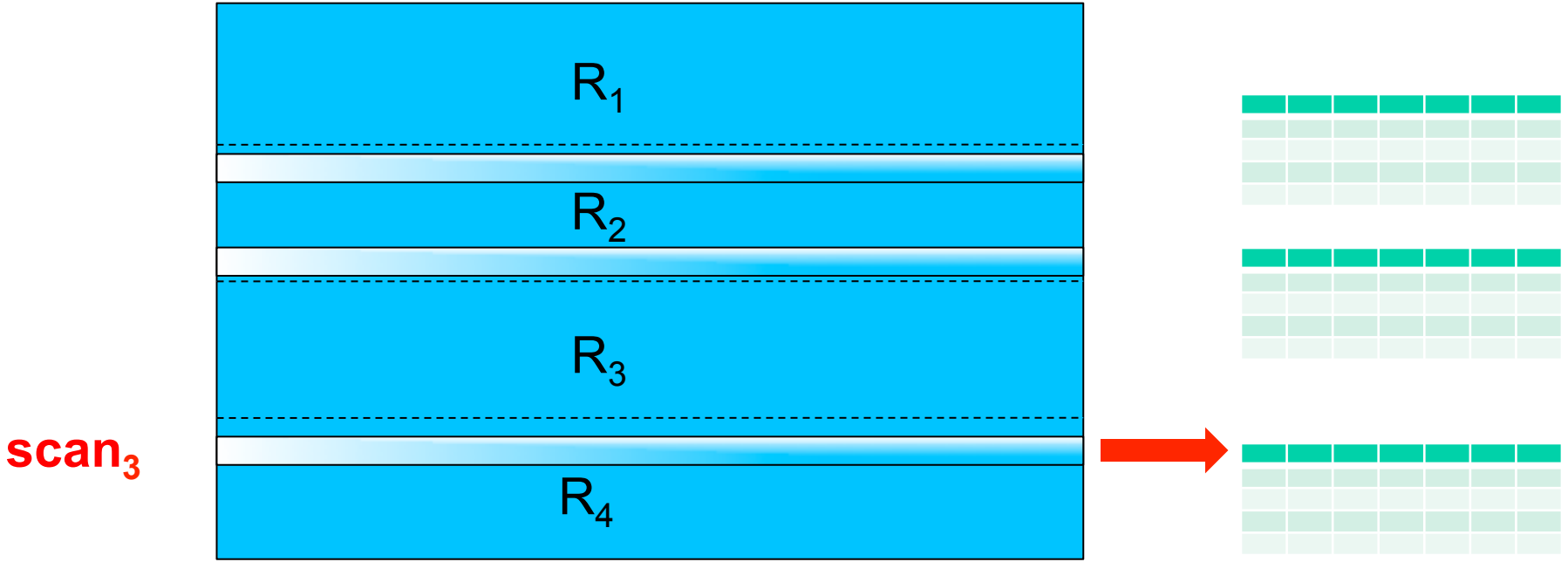
Phoenix Push Down: TopN
Each region holds N rows



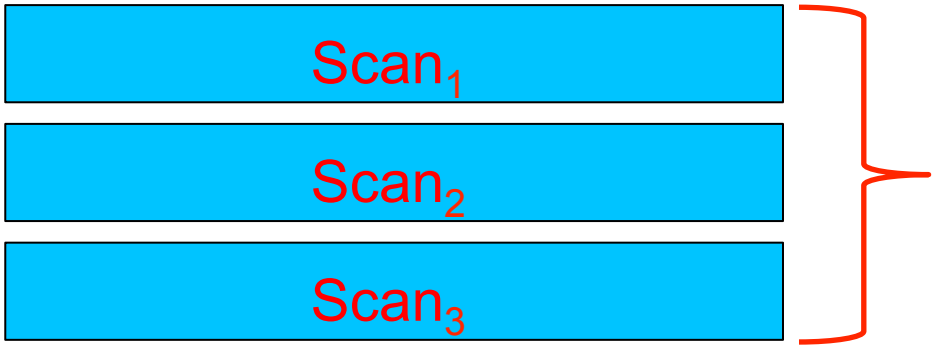
Phoenix Push Down: TopN
Each region holds N rows



Phoenix Push Down: TopN
Each region holds N rows



Phoenix Push Down: TopN
Client-side final merge sort



SERVER METRICS		
HOST	DATE	GC_TIME
SF3	Jun 2 10:10:10.234	22123
SF5	Jun 3 23:05:44.975	19876
SF2	Jun 9 08:10:32.147	11345
SF2	Jun 1 11:18:28.456	10234
SF1	Jun 3 22:03:22.142	10111

Phoenix Push Down: TopN Secondary Index

```
CREATE INDEX gc_time_index
ON server_metrics (gc_time DESC, date DESC)
INCLUDE (response_time)
```

GC_TIME_INDEX	
GC_TIME	INTEGER
DATE	DATE
HOST	VARCHAR
RESPONSE_TIME	INTEGER

Row Key

Phoenix Push Down: TopN Secondary Index

```
CREATE INDEX gc_time_index
ON server_metrics (gc_time DESC, date DESC)
INCLUDE (response_time)
```

GC_TIME_INDEX	
GC_TIME	INTEGER
DATE	DATE
HOST	VARCHAR
RESPONSE_TIME	INTEGER

Key Value

Phoenix Push Down: TopN
Secondary Index

- Original query doesn’t change
- Phoenix rewrites query to use index table
- All referenced columns must exist in index table for it to be considered
- Stats coming soon!

Phoenix Push Down: Hash Join

```
SELECT m.*, i.location
FROM server_metrics m
JOIN host_info i ON m.host = i.host
WHERE m.date > CURRENT_DATE() – 7
AND i.location = ‘SF’
ORDER BY m.gc_time DESC
LIMIT 5
```

Phoenix Push Down: Hash Join Separate LHS and RHS

```
SELECT m.*, i.location
FROM server_metrics m
JOIN host_info i ON m.host = i.host
WHERE m.date > CURRENT_DATE() – 7
AND i.location = ‘SF’
ORDER BY m.gc_time DESC
LIMIT 5
```

Phoenix Push Down: Hash Join Separate LHS and RHS

```
SELECT m.*, i.location
FROM server_metrics m
JOIN host_info i ON m.host = i.host
WHERE m.date > CURRENT_DATE() – 7
AND i.location = ‘SF’
ORDER BY m.gc_time DESC
LIMIT 5
```

Phoenix Push Down: Hash Join Separate LHS and RHS

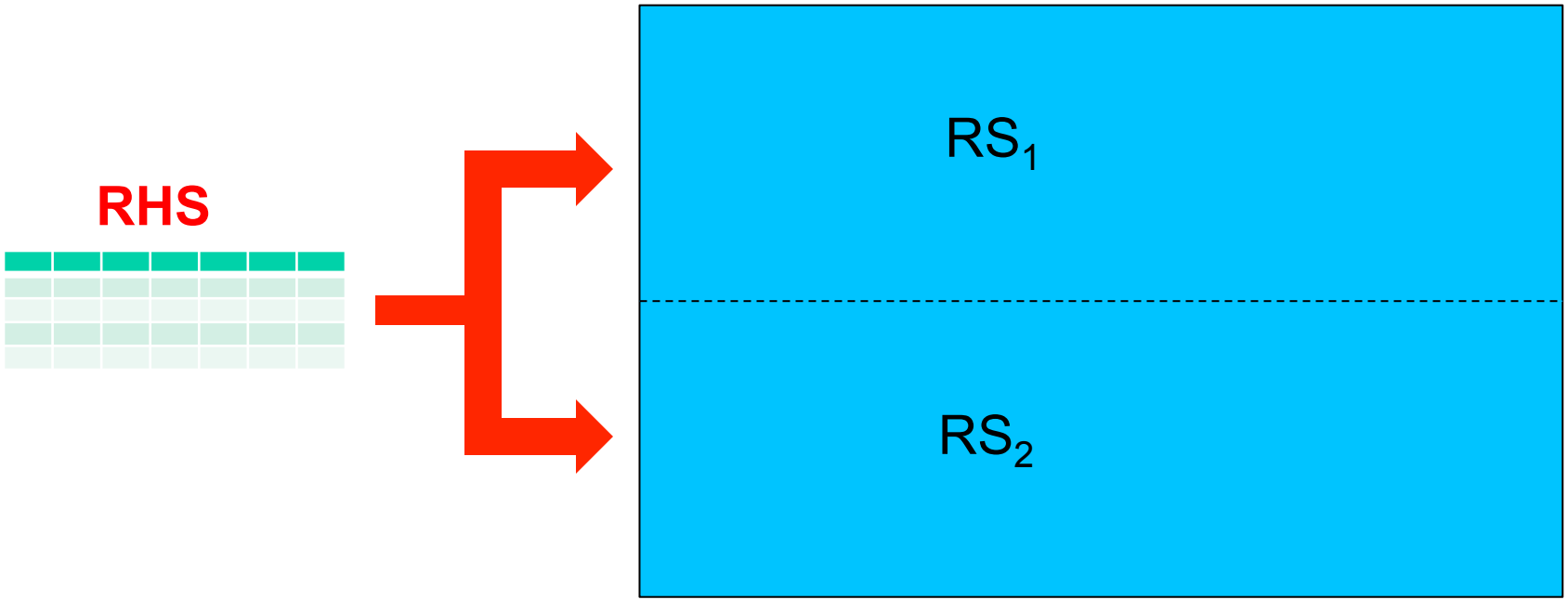
LHS

```
SELECT *
FROM server_metrics
WHERE date >
CURRENT_DATE() – 7
ORDER BY gc_time DESC
LIMIT 5
```

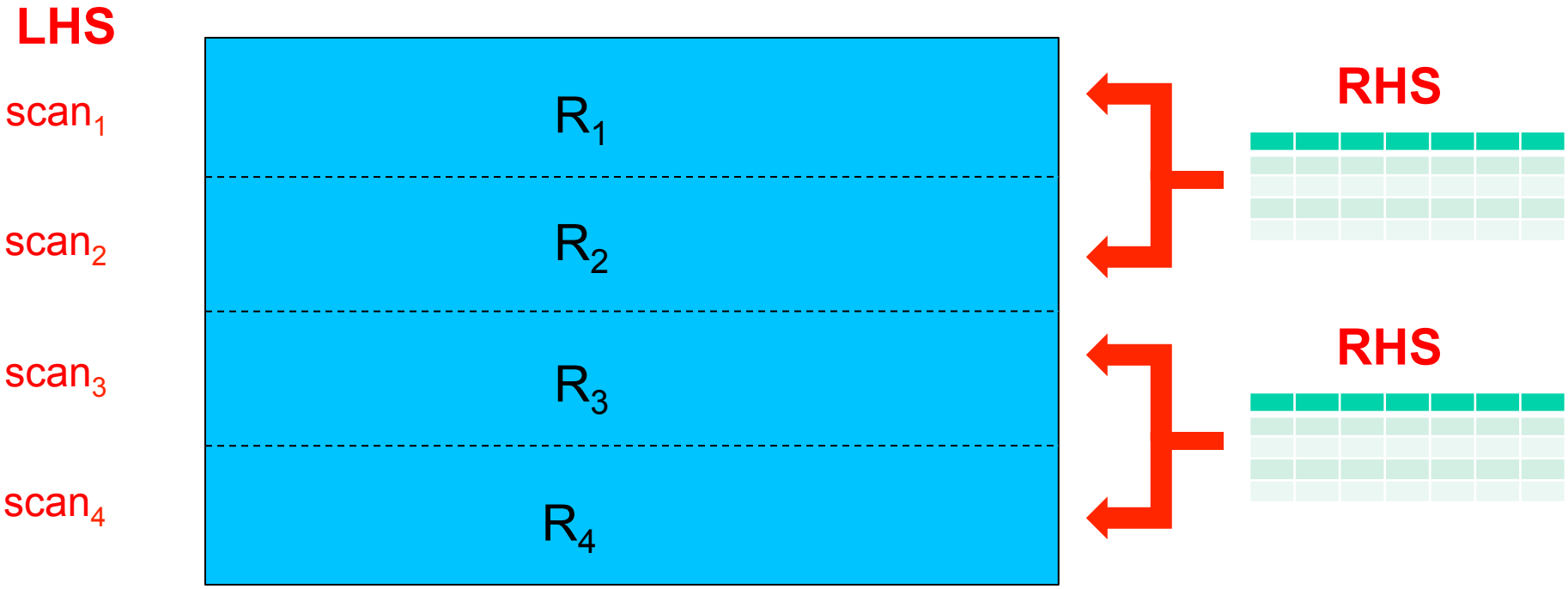
RHS

```
SELECT location
FROM host_info
WHERE location = ‘SF’
```

Phoenix Push Down: Hash Join
Execute & broadcast RHS to each RS



Phoenix Push Down: TopN
Server-side map lookup during scan



How does Phoenix help HBase scale?

How does Phoenix help HBase scale?

1. Phoenix allows multiple tables to share same physical HBase table
    - Updateable VIEW
    - Multi-tenant TABLE + tenant-specific VIEW
    - Support for secondary indexes on VIEWs

How does Phoenix help HBase scale?

2. HBase wants small # of big tables instead of large # of small tables
   - Each region for each column family of each table consumes resources

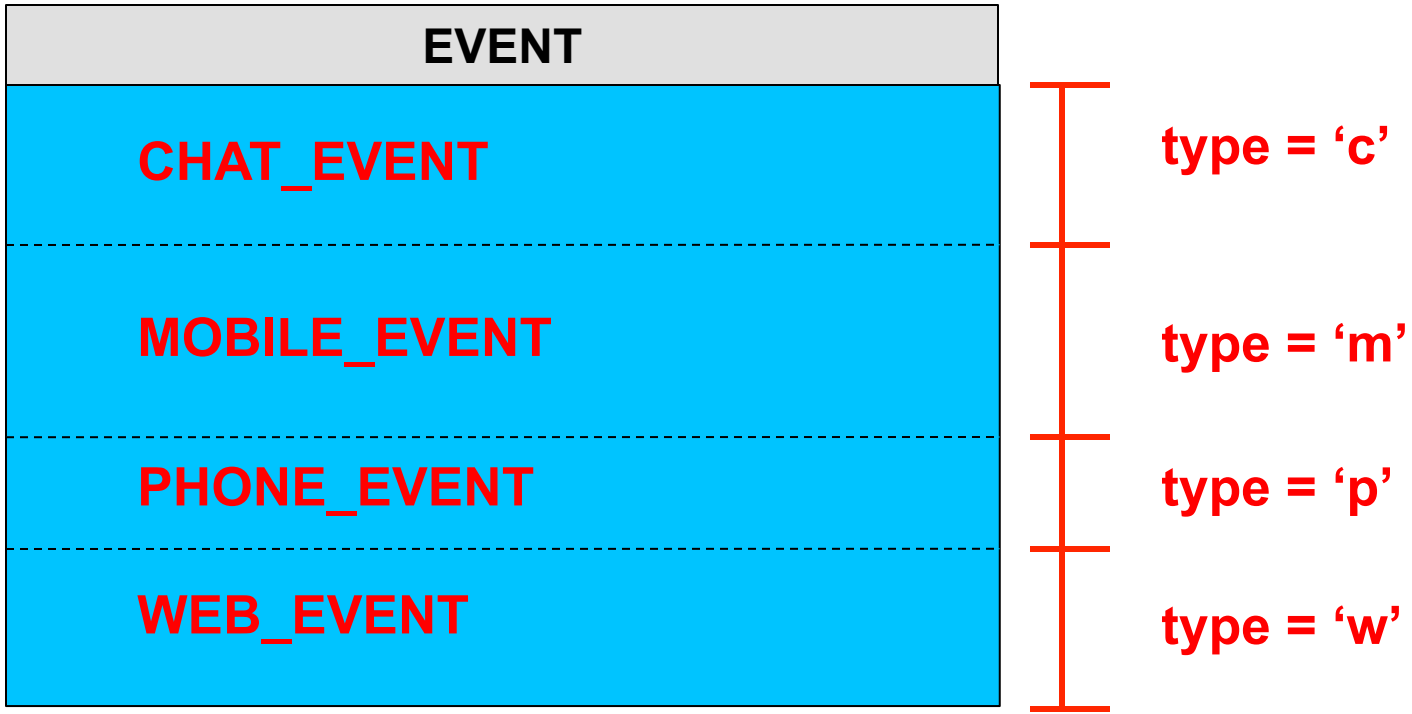
Phoenix Shared Tables: VIEW

```
CREATE TABLE event (
    type CHAR(1),
    event_id BIGINT,
    created_date DATE,
    created_by VARCHAR,
    CONSTRAINT pk PRIMARY KEY (type, event_id));
```

```
CREATE VIEW web_event (
    referrer VARCHAR) AS
SELECT * FROM event
WHERE type='w';
```

- Includes columns from TABLE
- Cannot define PK
- Updateable if only equality expressions separated by AND

Phoenix Shared Tables: VIEW
Same physical HBase table



Phoenix Shared Table: MULTI_TENANT

```
CREATE TABLE event (
    tenant_id VARCHAR,
    type CHAR(1),
    event_id BIGINT,
    created_date DATE,
    created_by VARCHAR,
    CONSTRAINT pk PRIMARY KEY (tenant_id, type, event_id))
   MULTI_TENANT=true;
```

First PK column identifies tenant ID

Phoenix Shared Table: MULTI_TENANT

```
CREATE VIEW web_event (
      referrer VARCHAR) AS
SELECT * FROM event
WHERE type='w';
```

Tenant-specific connection

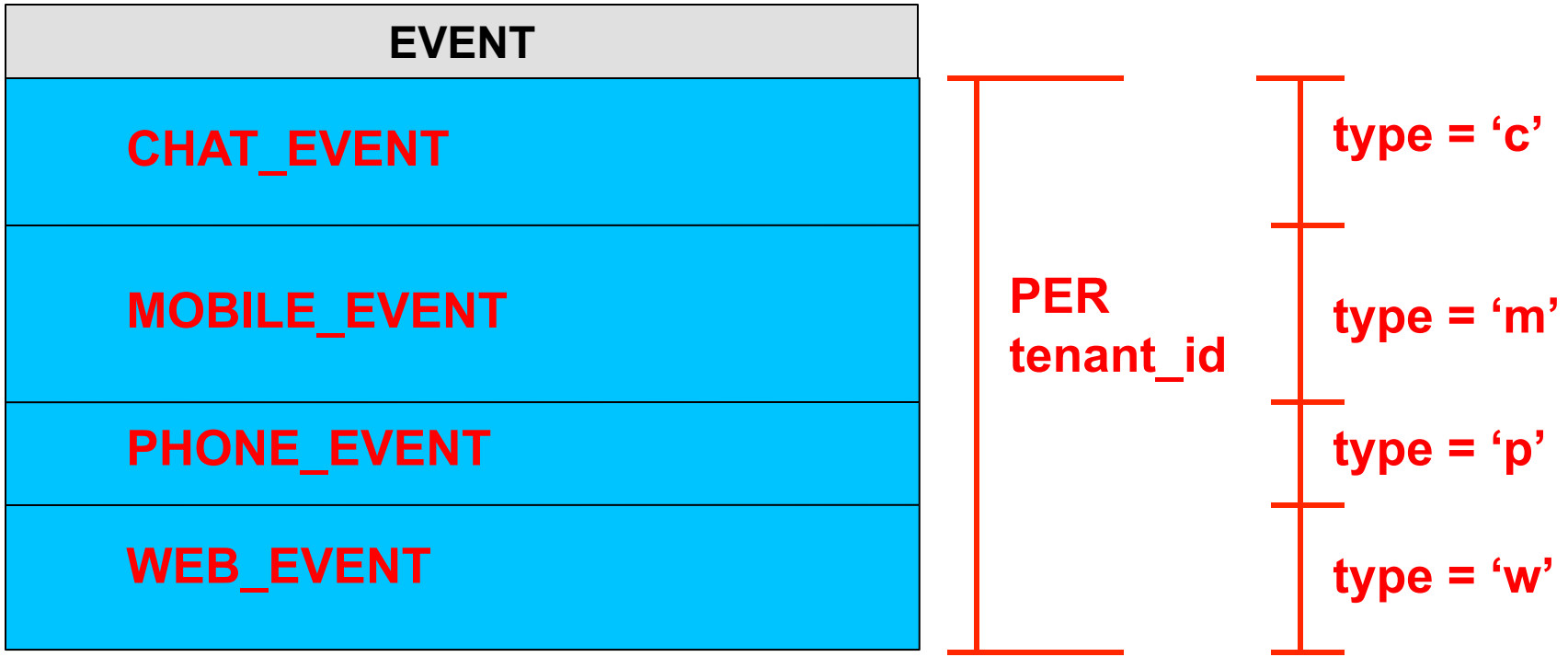
```
DriverManager.connect("jdbc:phoenix:localhost;tenantId=me");
```

```
CREATE VIEW my_web_event AS
SELECT * FROM web_event;
```

Tenant-specific view

Phoenix Shared Tables: MULTI_TENANT
Same physical HBase table

Phoenix Shared Tables: MULTI_TENANT Same physical HBase table



Phoenix Shared Tables: MULTI_TENANT

- Tenant-specific connection may only see and operate on *their* data
  - MetaData APIs honor this
  - Phoenix automatically manages scan ranges
- Primary key constraint of base table may not be changed
  - Indexes in separate shared table may be added to a VIEW
- DDL operations restricted
  - No ALTER of base table
  - No DROP of columns referenced in WHERE clause

Phoenix Roadmap

- ~~Derived/nested tables~~ (in 3.1/4.1)
- ~~Local Indexes~~ (in 4.1)
- Transactions
- More Join strategies
- Correlated sub-queries
- Cost-based query optimizer
- OLAP extensions

Thank you!
Questions/comments?