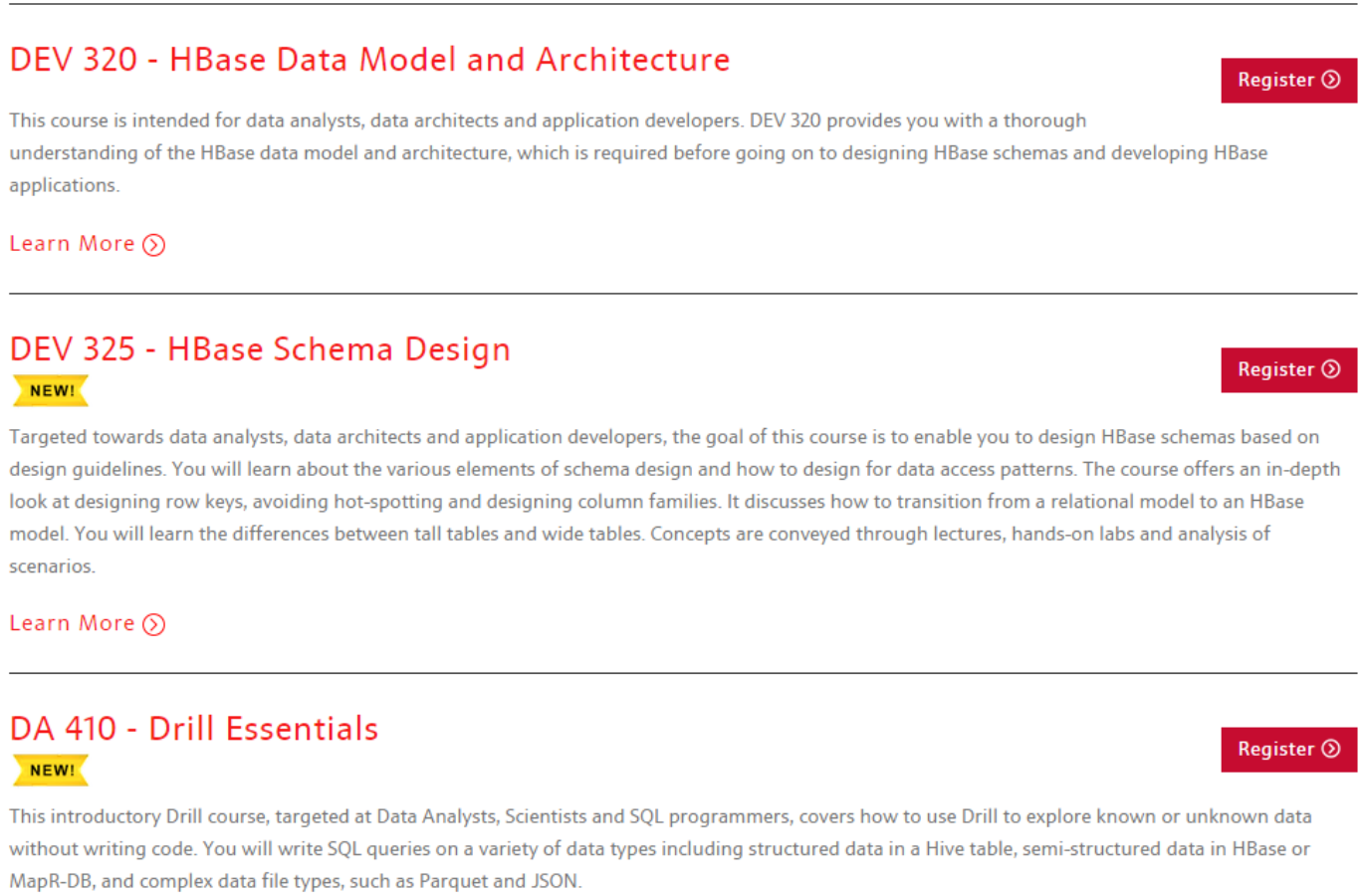
Scale Your SQL Database with SQL, Hadoop, Drill, JSON, NoSQL and HBase

Carol McDonald cmcdonald@mapr.com

http://answers.mapr.com/

Free Hadoop On Demand Training

- https://www.mapr.com/services/mapr-academy/big-data-hadoop-online-training



DEV 320 - HBase Data Model and Architecture

Register

This course is intended for data analysts, data architects and application developers. DEV 320 provides you with a thorough understanding of the HBase data model and architecture, which is required before going on to designing HBase schemas and developing HBase applications.

Learn More

DEV 325 - HBase Schema Design

NEW!

Register

Targeted towards data analysts, data architects and application developers, the goal of this course is to enable you to design HBase schemas based on design guidelines. You will learn about the various elements of schema design and how to design for data access patterns. The course offers an in-depth look at designing row keys, avoiding hot-spotting and designing column families. It discusses how to transition from a relational model to an HBase model. You will learn the differences between tall tables and wide tables. Concepts are conveyed through lectures, hands-on labs and analysis of scenarios.

Learn More

DA 410 - Drill Essentials

NEW!

Register

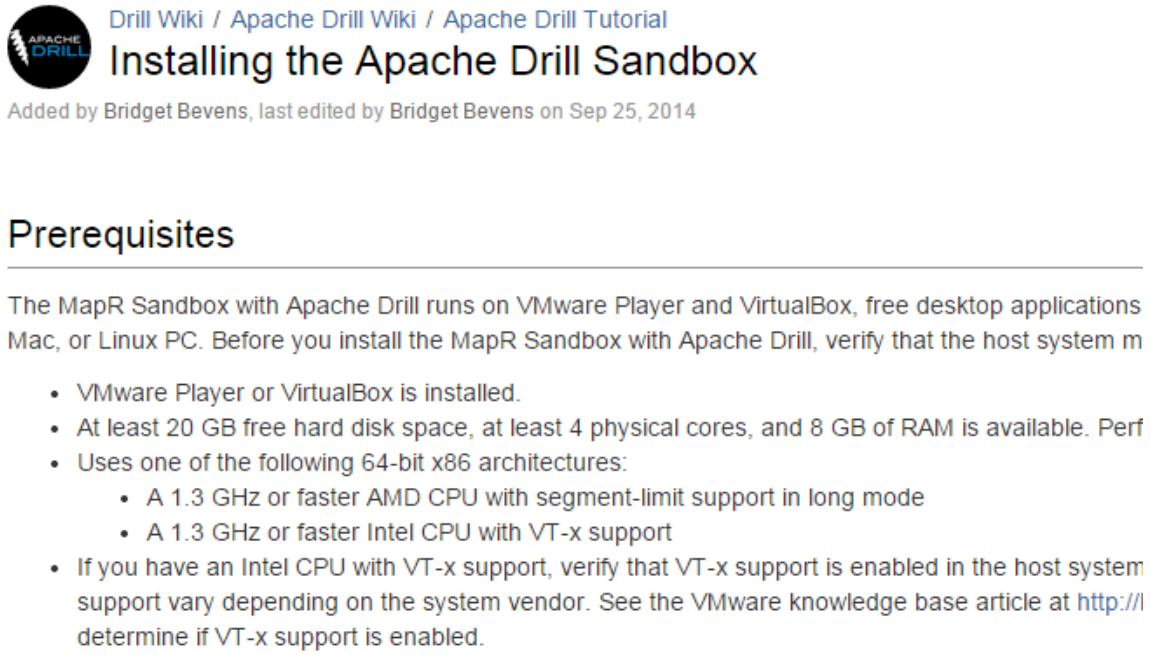
This introductory Drill course, targeted at Data Analysts, Scientists and SQL programmers, covers how to use Drill to explore known or unknown data without writing code. You will write SQL queries on a variety of data types including structured data in a Hive table, semi-structured data in HBase or MapR-DB, and complex data file types, such as Parquet and JSON.

Objectives of this session

- What is HBase?
  - Why do we need NoSQL / HBase?
  - Overview of HBase & HBase data model
  - HBase Architecture and data flow
- Design considerations when migrating from RDBMS to HBase
- MapReduce and HBase
- Hive and HBase
- Drill and HBase

Prerequisite for Hands-On-Labs (before Lecture)

- Install a one-node MapR Sandbox on your laptop
  - Download install the MapR Sandbox:
  - **https://cwiki.apache.org/confluence/display/DRILL/Installing+the+Apache+Drill+Sandbox**



Drill Wiki / Apache Drill Wiki / Apache Drill Tutorial

Installing the Apache Drill Sandbox

Added by Bridget Bevens, last edited by Bridget Bevens on Sep 25, 2014

Prerequisites

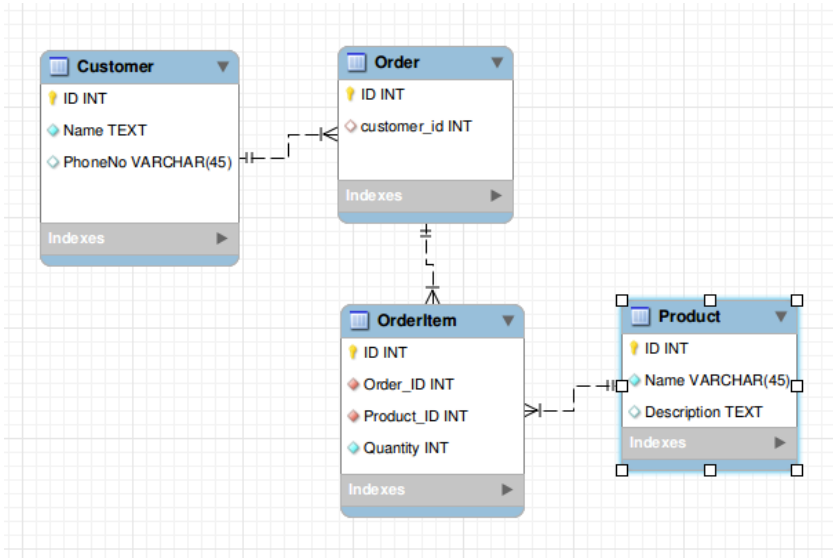
The MapR Sandbox with Apache Drill runs on VMware Player and VirtualBox, free desktop applications
Mac, or Linux PC. Before you install the MapR Sandbox with Apache Drill, verify that the host system m

- VMware Player or VirtualBox is installed.
- At least 20 GB free hard disk space, at least 4 physical cores, and 8 GB of RAM is available. Perf
- Uses one of the following 64-bit x86 architectures:
  - A 1.3 GHz or faster AMD CPU with segment-limit support in long mode
  - A 1.3 GHz or faster Intel CPU with VT-x support
- If you have an Intel CPU with VT-x support, verify that VT-x support is enabled in the host system
  support vary depending on the system vendor. See the VMware knowledge base article at http://
  determine if VT-x support is enabled.

Why do we need NoSQL / HBase?

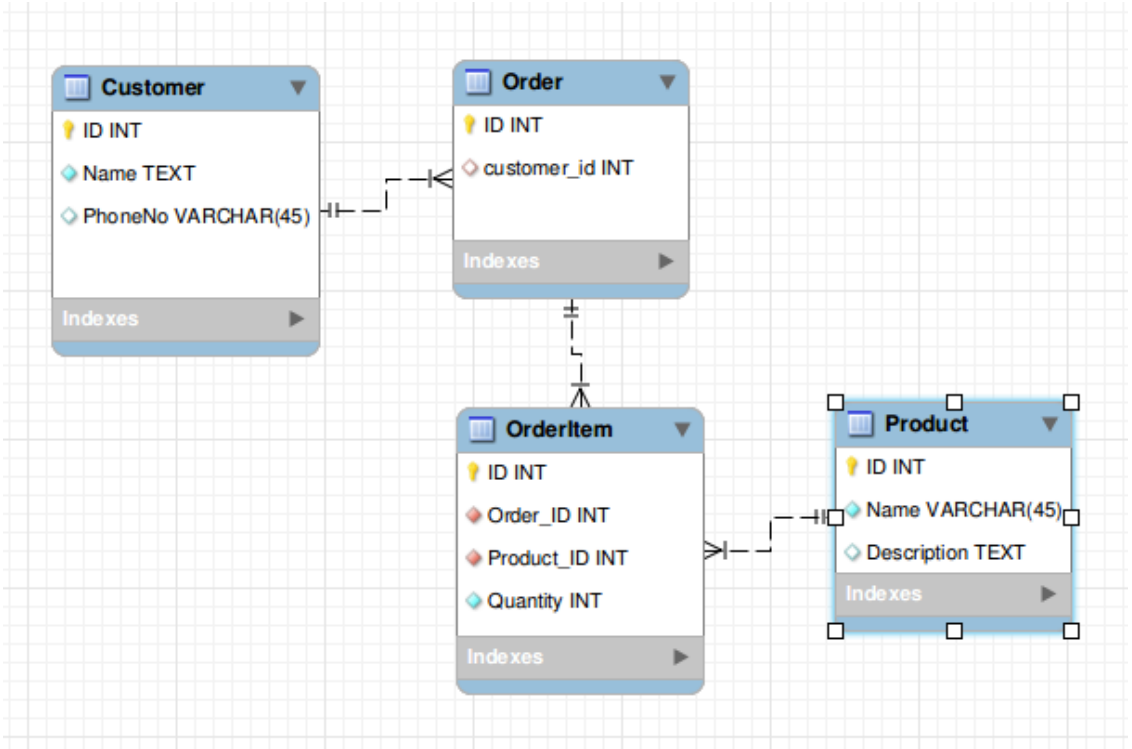
Relational Model

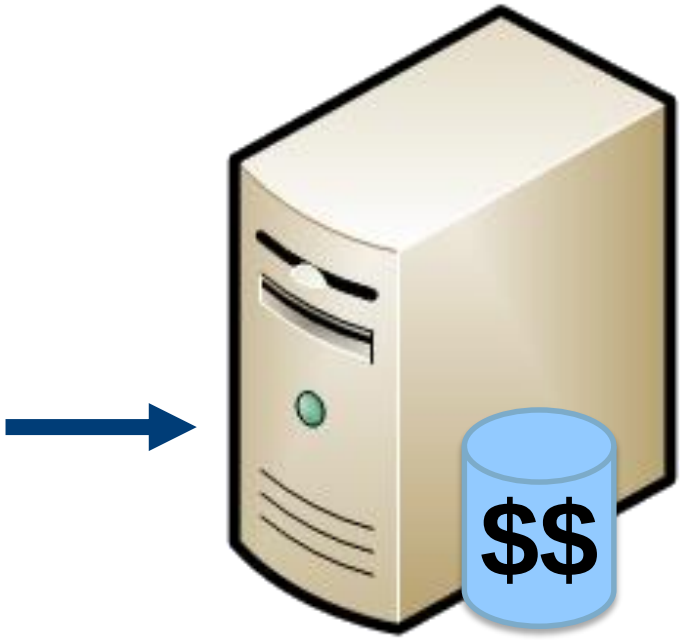
- Pros
  - **Standard** persistence model
  - **Transactions handle**
    - **concurrency , consistency**
  - efficient and robust structure for storing data



Relational Databses vs HBase - Scaling

RDBMS - Scale UP approach

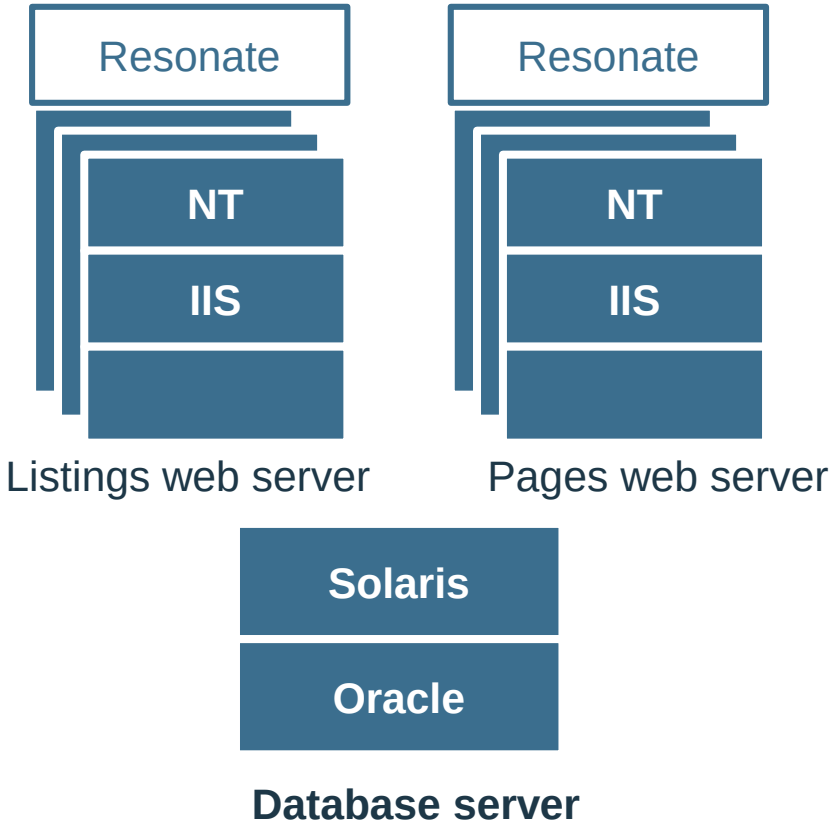




Vertical scale = **big box**

RDBMS Scaling Up Example - eBay

- Back End Oracle Database server **scaled vertically to a larger machine** (Sun E10000)



http://www.addsimplicity.com/downloads/eBaySDForum2006-11-29.pdf

What changed to bring on NoSQL?

Lots of data, the need to scale horizontally

Horizonal scale: split table by rows into partitions across a cluster

Key	colB	colC
val	val	val
xxx	val	val

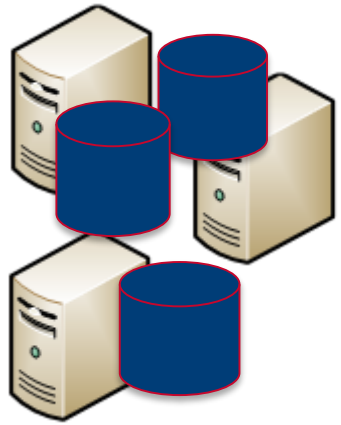
id 1-1000

Key	colB	colC
val	val	val
xxx	val	val

id 1000-2000

Key	colB	colC
val	val	val
xxx	val	val

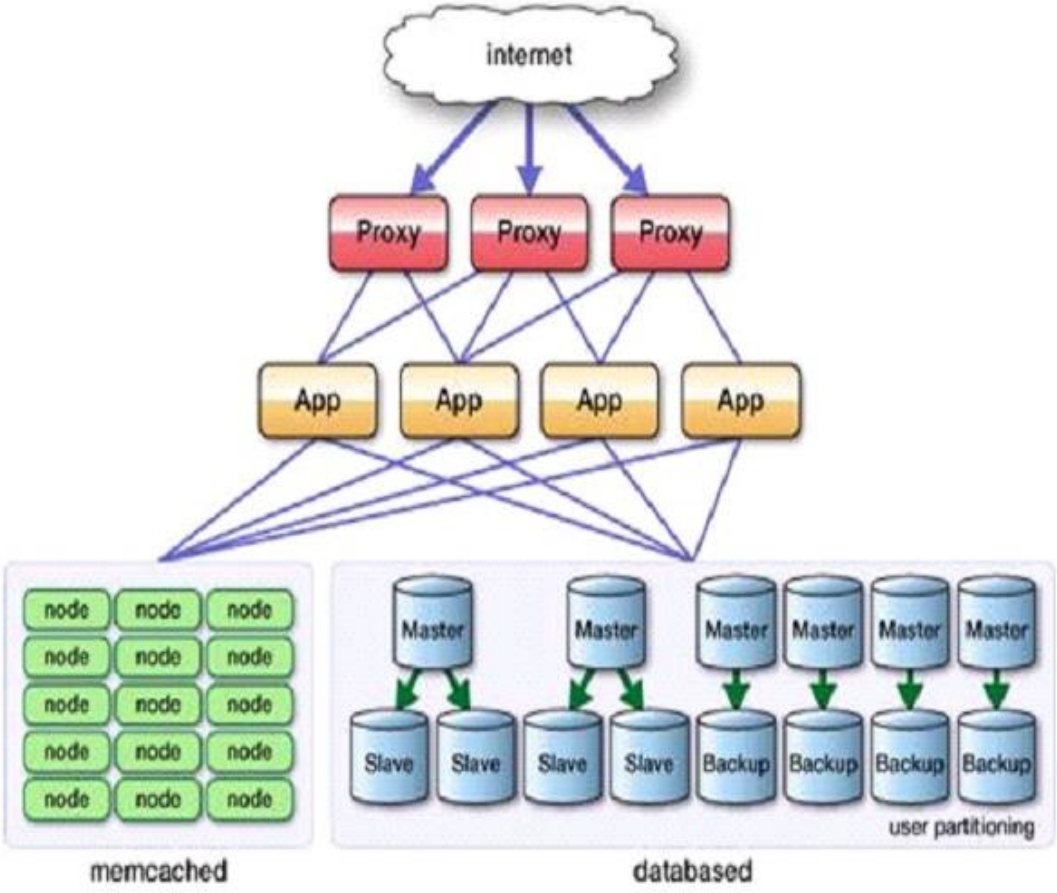
id 2000=3000



- **Horizontal** scaling
  - Cheaper than vertical
  - parallel execution
- Relational databases were **not designed to do this** automatically

Facebook 2010

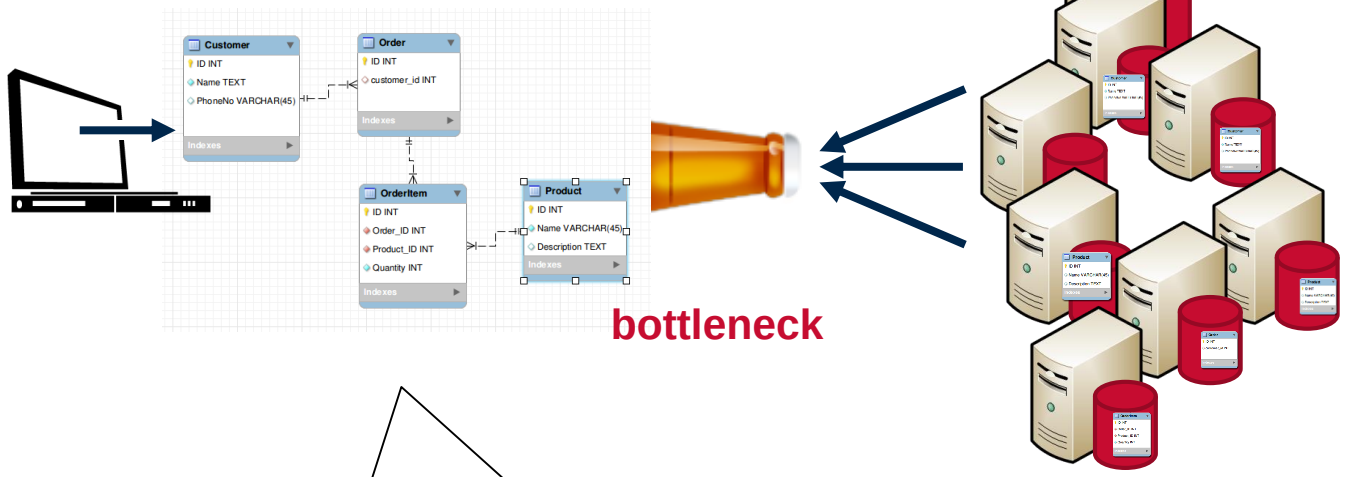
- 9000 memcache instances
- **4000 Shards mysql**



http://gigaom.com/2011/07/07/facebook-trapped-in-mysql-fate-worse-than-death/

Relational Databases vs. HBase – Data Storage Model

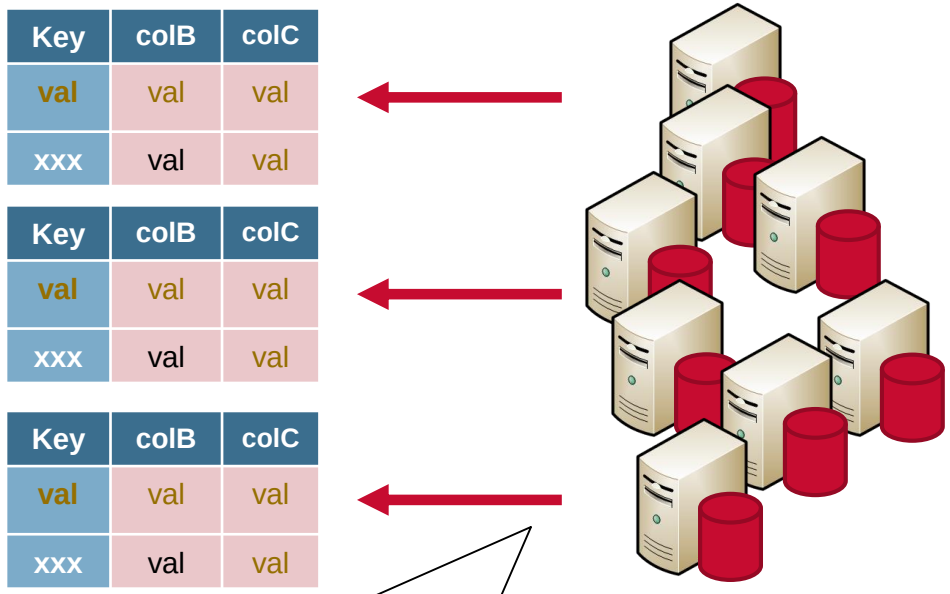
RDBMS



Distributed Joins, Transactions **do not scale**

Storage Model

HBase



Data that is accessed together is stored together

Hbase designed for Distribution, Scale, Speed

Big Table

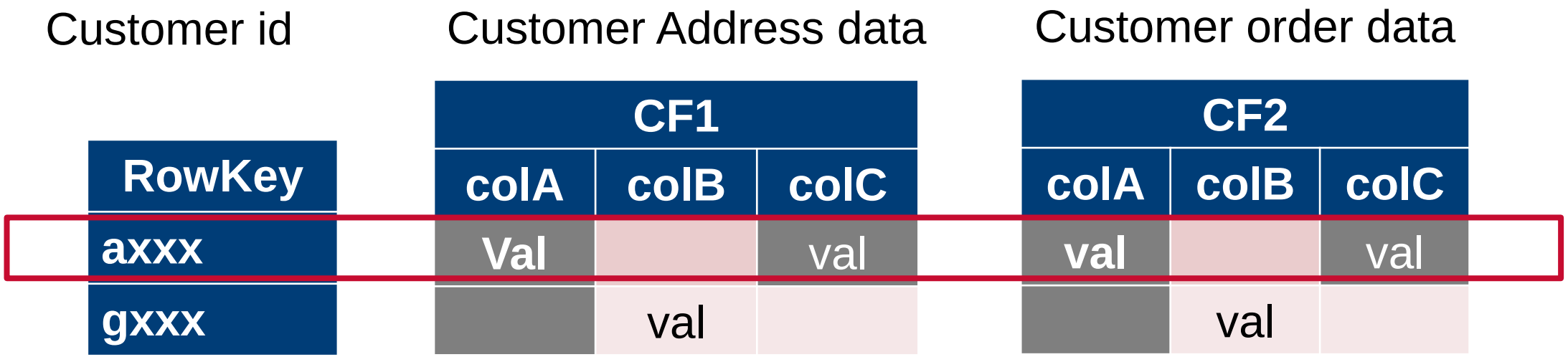
- Distributed Storage System
- Paper published in 2006.

Google File System

MapReduce

Runs on commodity hardware
Designed to Scale

HBase is a ColumnFamily oriented Database

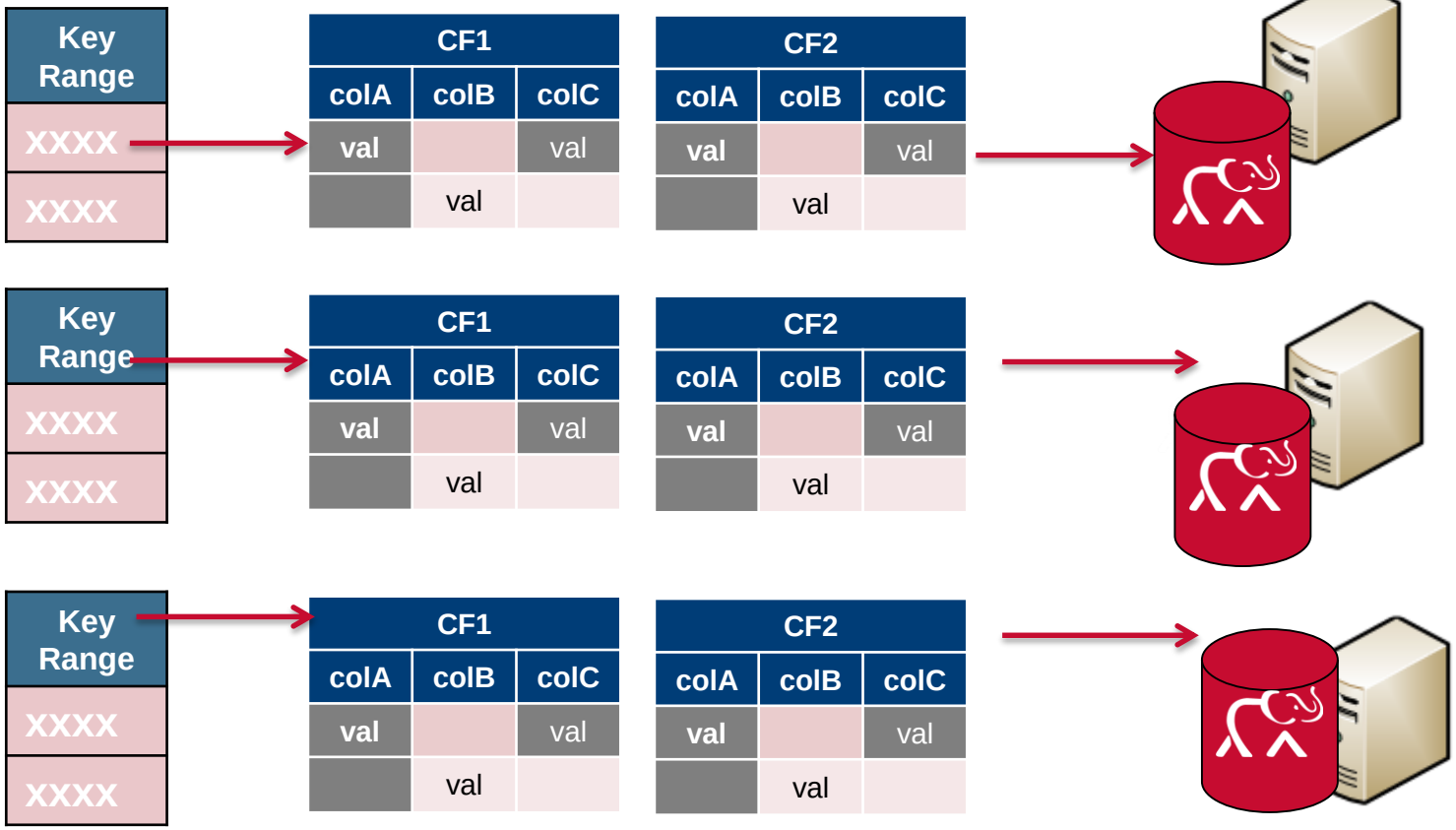


Data is accessed and stored together:

- RowKey is the primary index
- Column Families group similar data by row key

HBase is a Distributed Database

Put, Get by Key



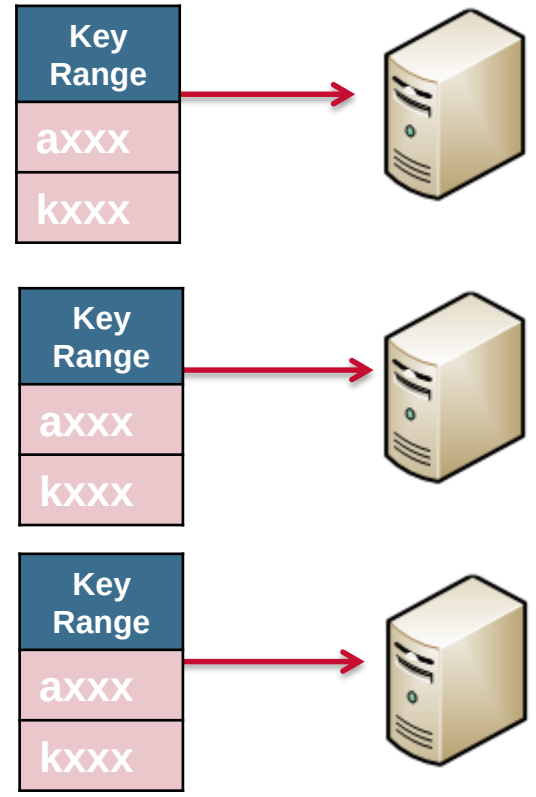
Data is automatically distributed across the cluster

- **Key range** is used for **horizontal partitioning**

Column Family Databases

distributed data stored and accessed together:

- Pros
  - **scalable**
  - **Fast Writes and Reads**
- Cons
  - **No** joins
  - No dynamic queries
  - Need to **know** how data will be **queried in advance**



Hbase Data Model

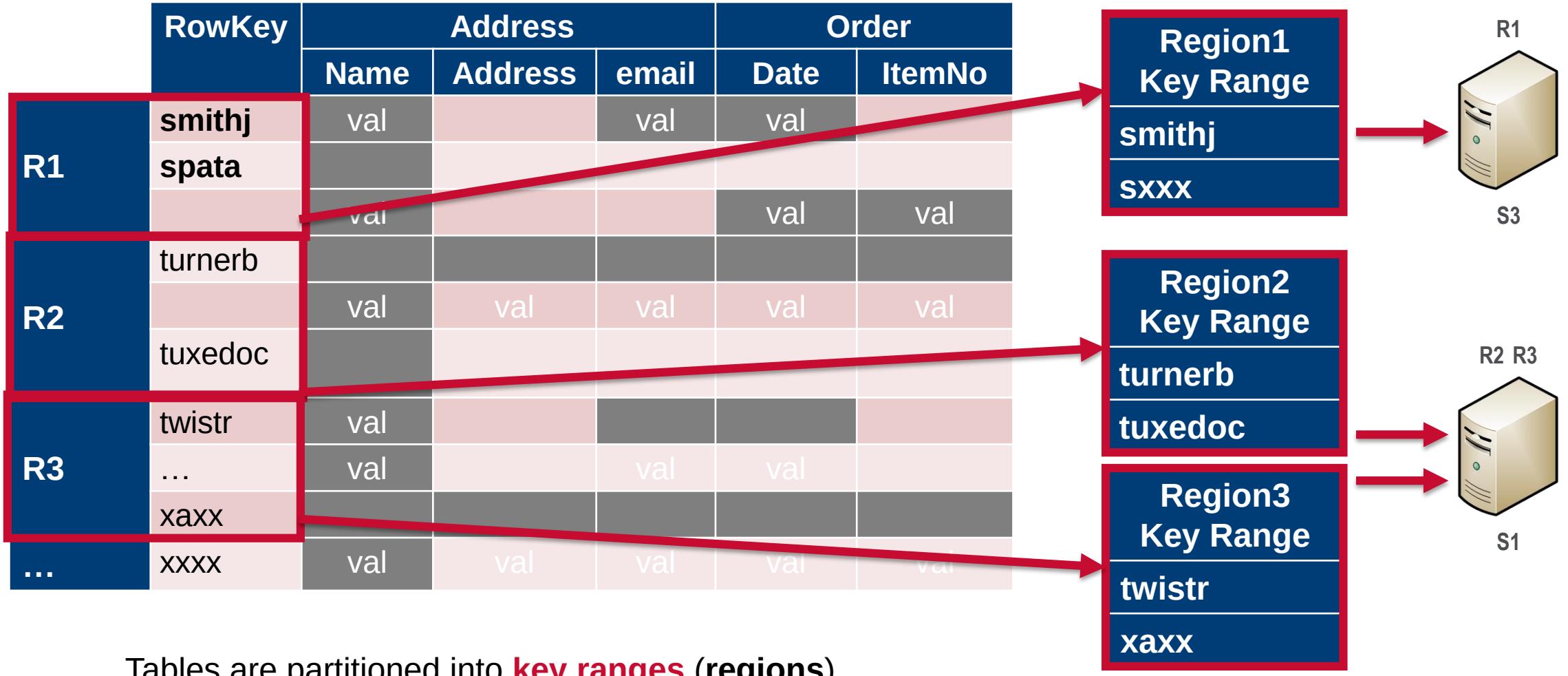
HBase Data Model- Row Keys

RowKey	Address			Order				…
	street	city	state	Date	ItemNo	Ship Address	Cost	
smithj	val		val	val			val	
spata								
sxxxx	val			val	val	val		
…								
turnerb	val	val	val	val	val	val	val	
…								
	val							
twistr	val		val	val			val	
…								
zaxx	val	val	val	val	val	val		
zxxx	val						val	

Row Keys: identify the rows in an HBase table.

Tables are split into Regions = contiguous keys

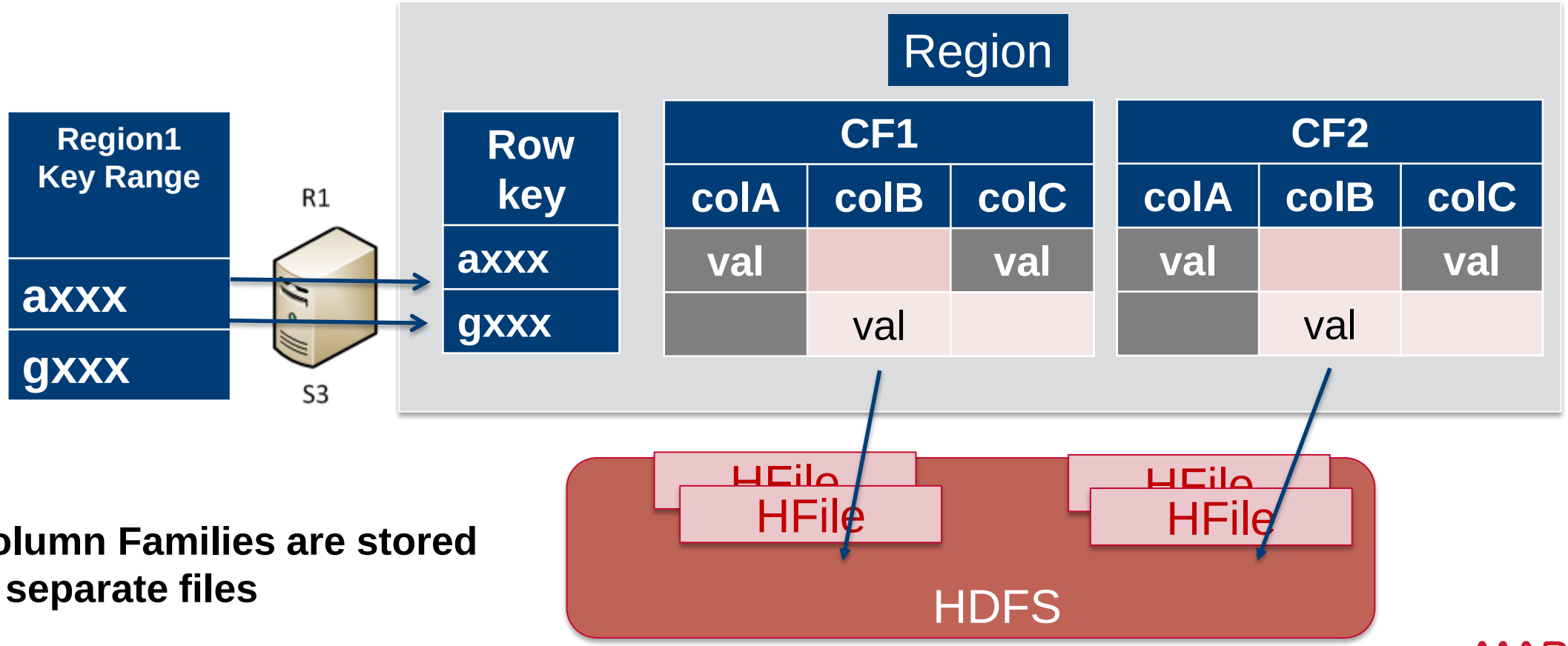
	RowKey	Address			Order	
		Name	Address	email	Date	ItemNo
R1	smithj	val		val	val	
	spata					
		val			val	val
R2	turnerb					
		val	val	val	val	val
	tuxedoc					
R3	twistr	val				
	…	val		val	val	
	xaxx					
…	xxxx	val	val	val	val	val



Tables are partitioned into **key ranges** (**regions**)
Region= served by nodes (Region Servers)
Regions are spread across cluster

Column Families are stored Separately

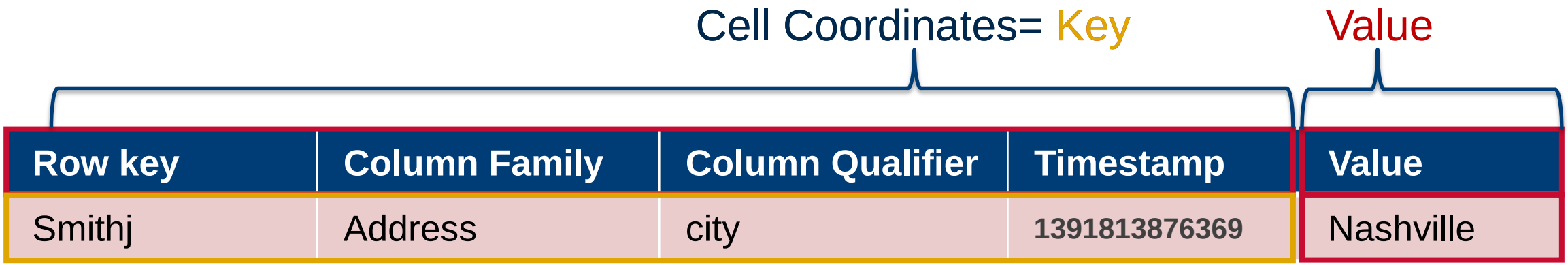
- column families are stored and can be accessed separately



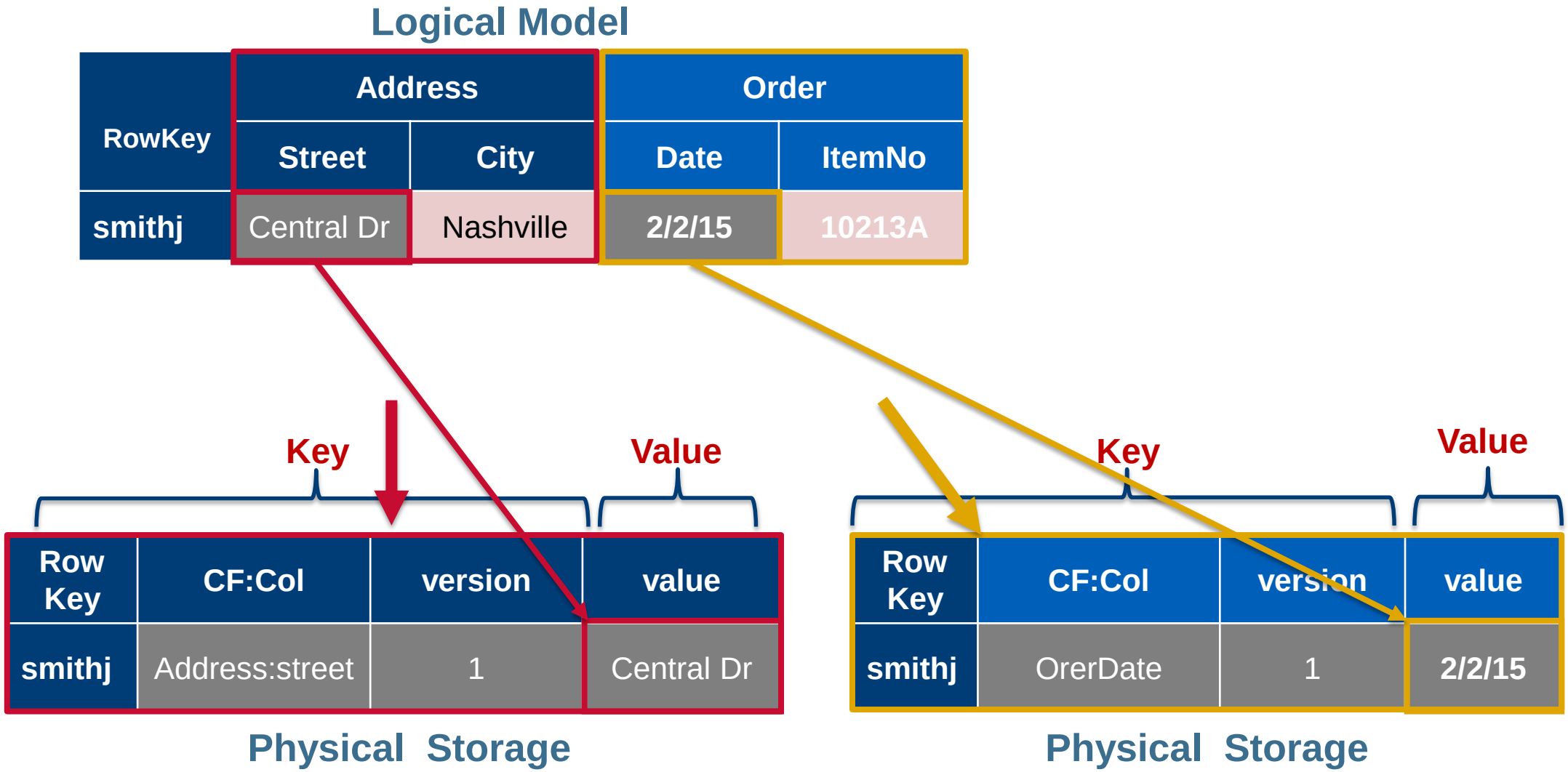
Column Families are stored in separate files

HBase Data Model - Cells

- Data is stored in **Key Value** format
- Value for each **cell** is specified by complete coordinates:
  - (Row key, ColumnFamily, Column Qualifier, timestamp) => Value



Logical Data Model vs Physical Data Storage



Sparse Data with Cell Versions

	CF1:colA	CF1:colB	CF1:colC
Row1	@time7: value3		
Row10	@time2: value1	@time2: value1	
Row11	@time6: value2		
Row2	@time4: value1		@time4: value1

Versioned Data

Number of versions can be configured. Default number equal to 1

put, adds new cell

Key	CF:Col	version	value
smithj	Address:street	v3	19th Ave
smithj	Address:street	v2	Main St
smithj	Address:street	v1	Central Dr

Logical Data Model vs Physical Data Storage

Row Key	CF1		CF2	
	ca	cb	ca	cd
ra	1		2	
rb		3,4		
rc	5		6,7	8

Logical Model

Column families are stored separately
Row keys, Qualifiers are sorted **lexicographically**

Physical Storage

Key Value

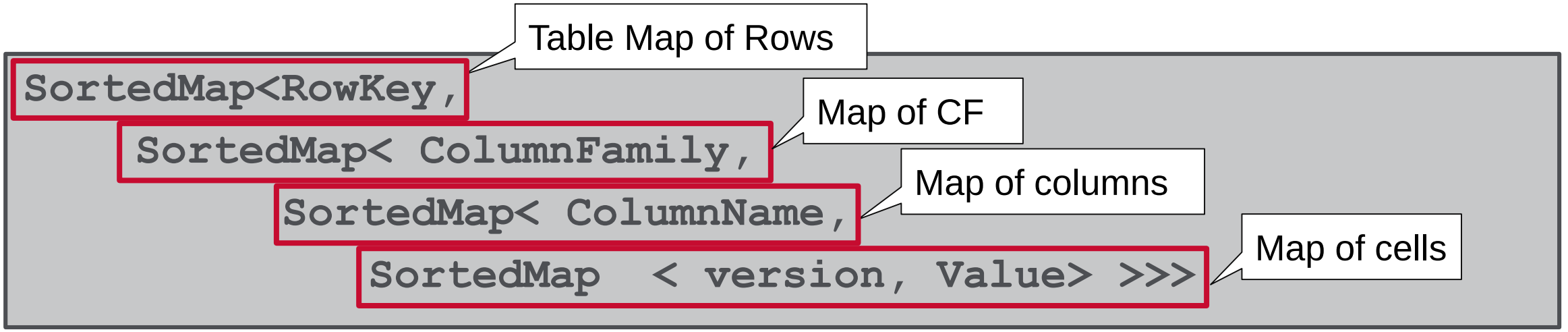
Key	CF1:Col	version	value
ra	cf1:ca	1	1
rb	cf1:cb	2	4
rb	cf1:cb	1	3
rc	cf1:ca	1	5

Key Value

Key	CF2:Col	version	value
ra	cf2:ca	1	2
rc	cf2:ca	2	7
rc	cf2:ca	1	6
rc	cf2:cd	1	8

HBase Table is a Sorted Map of Maps

In Java this is the table:



Key	CF:Col	version	value
smithj	Address:street	v2	Main St
smithj	Address:street	v1	Central Dr
spata	Address:street	v1	High Ave
turnerb	Address:street	v1	Cedar St

Key	CF:Col	version	value
smithj	Order:Date	v1	2/2/15
spata	Order:Date	v1	1/31/14
turnerb	OrderDate	v1	7/8/14

HBase Table - SortedMap

```
<smithj,<Address, <street, <v1, Central Dr>>
                  <street, <v2, Main St>>
              <Order <Date, <v1, 2/2/15>>>
<spata,<Address, <street, <v1,High Ave>>
              <Order <Date, <v1, 1/31/14>>>
<turnerb,<Address, <street, <v1,Cedar St>>
              <Order <Date, <v1, 7/8/14>>>
```

Key	CF:Col	version	value
smithj	Address:street	v2	Main St
smithj	Address:street	v1	Central Dr
spata	Address:street	v1	High Ave
turnerb	Address:street	v1	Cedar St

Key	CF:Col	version	value
smithj	Order:Date	v1	2/2/15
shawa	Order:Date	v1	1/31/14
turnerb	OrderDate	v1	7/8/14

Basic Table Operations

- **Create Table, define Column Families before data is imported**
  - but not the rows keys or number/names of columns
- Low level API, technically more demanding
- **Basic data access operations (CRUD):**

put	Inserts data into rows (both create and update)
get	Accesses data from one row
scan	Accesses data from a range of rows
delete	Delete a row or a range of rows or columns

Create HBase Table – Using HBase Shell

```
hbase> create '/user/user01/Customer', {NAME =>'Address'} , {NAME =>'Order'}
hbase> put '/user/user01/Customer', 'smithj', 'Address:street', 'Central Dr'
hbase> put '/user/user01/Customer', 'smithj', 'Order:Date', '2/2/15'
hbase> put '/user/user01/Customer', 'spata', 'Address:city', 'Columbus'
hbase> put '/user/user01/Customer', 'spata', 'Order:Date', '1/31/14'
```

Row Key	Address			Order	
	street	city	state	Date	ItemNo
smithj	Central Dr	Nashville	TN	2/2/15	10213A
spata	High Ave	Columbus	OH	1/31/14	23401V
turnerb	Cedar St	Seattle	WA	7/8/14	10938A

Create HBase Table – Using HBase Shell

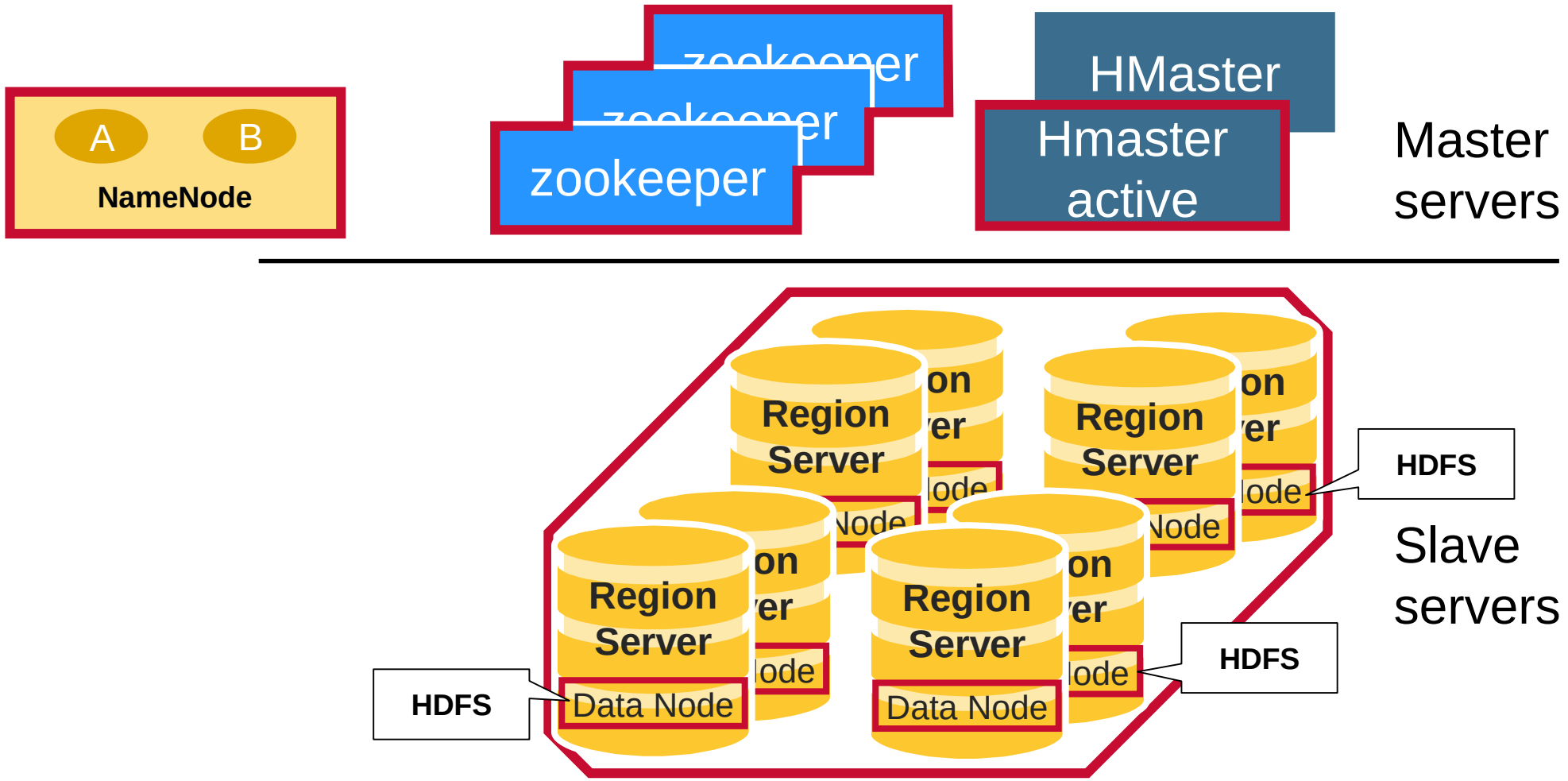
```
hbase> get '/user/user01/Customer', 'smithj'

hbase> scan '/user/user01/Customer'

hbase> describe '/user/user01/Customer'
```

HBase Architecture

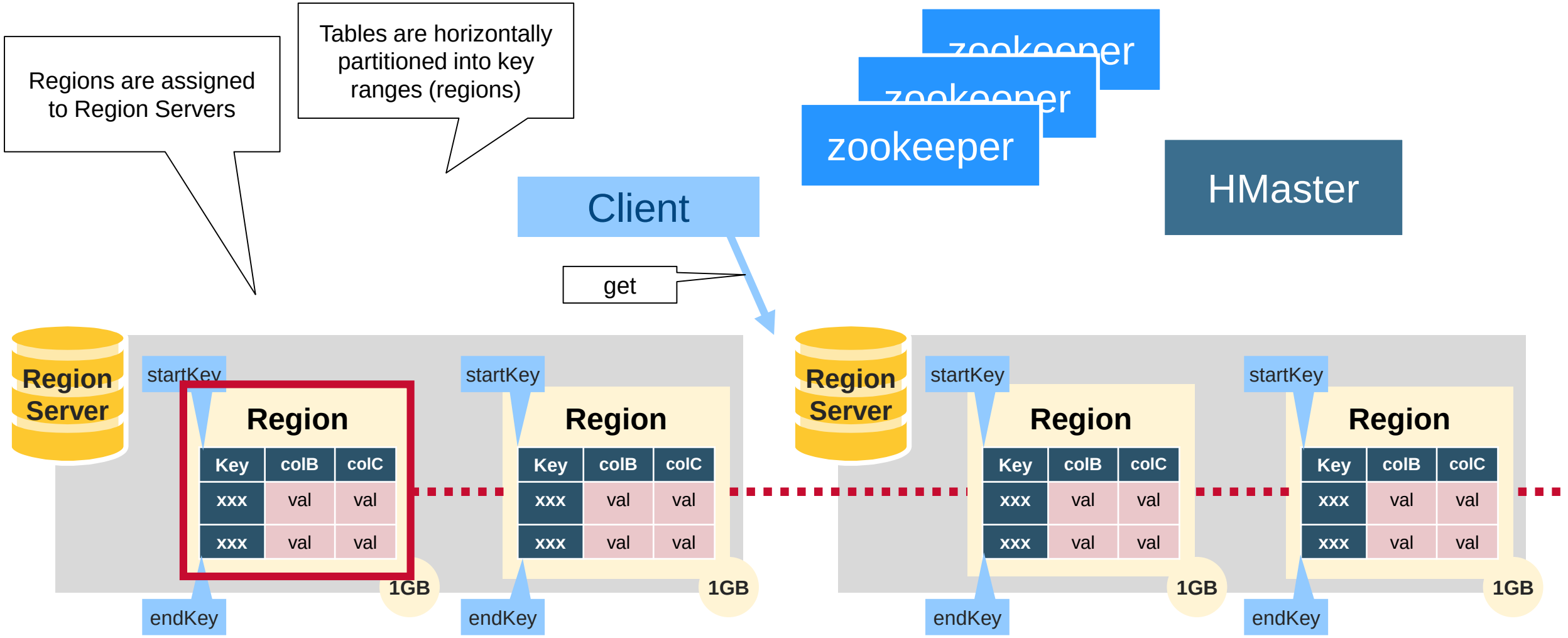
Data flow for Writes, Reads

Designed to Scale

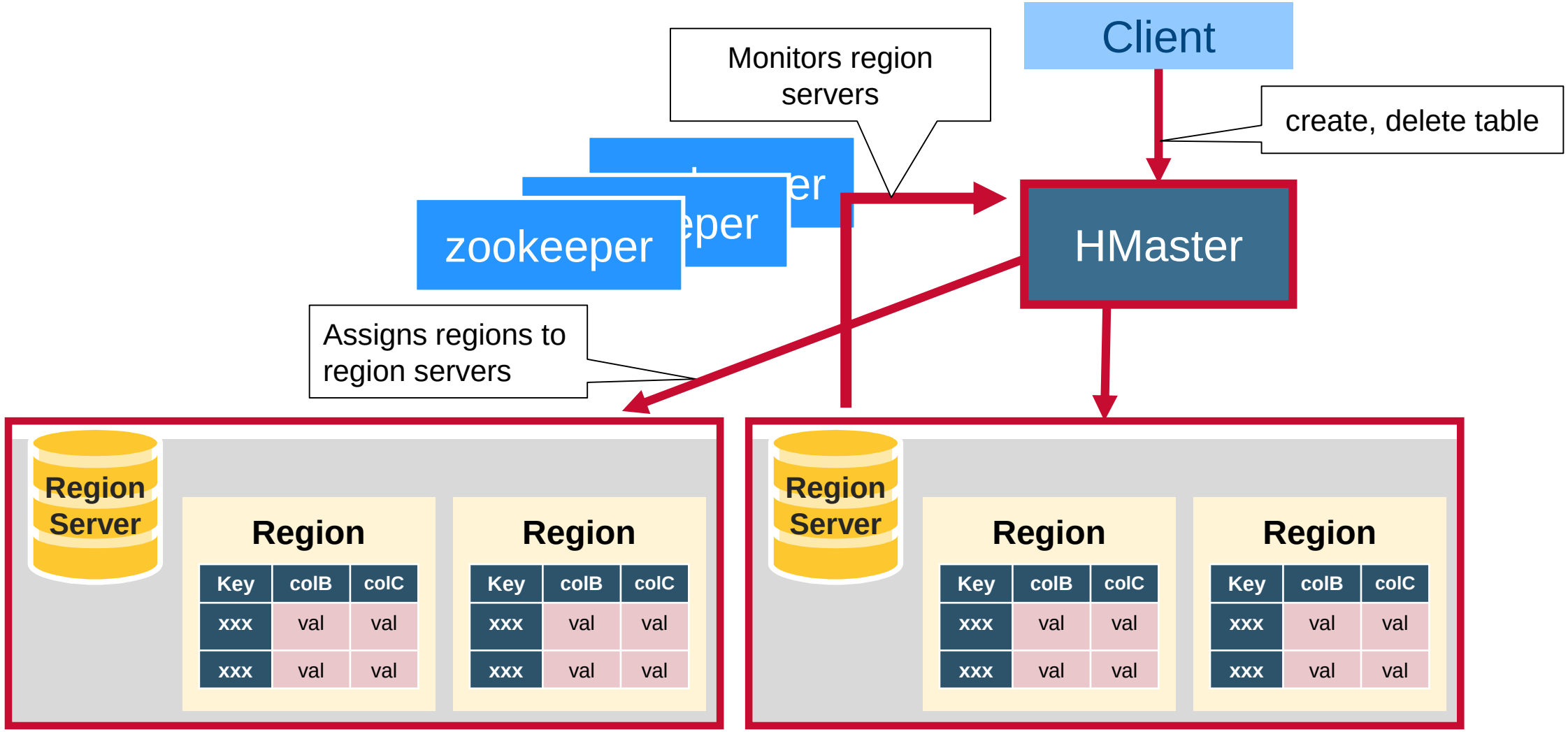
HBase Architectural Components



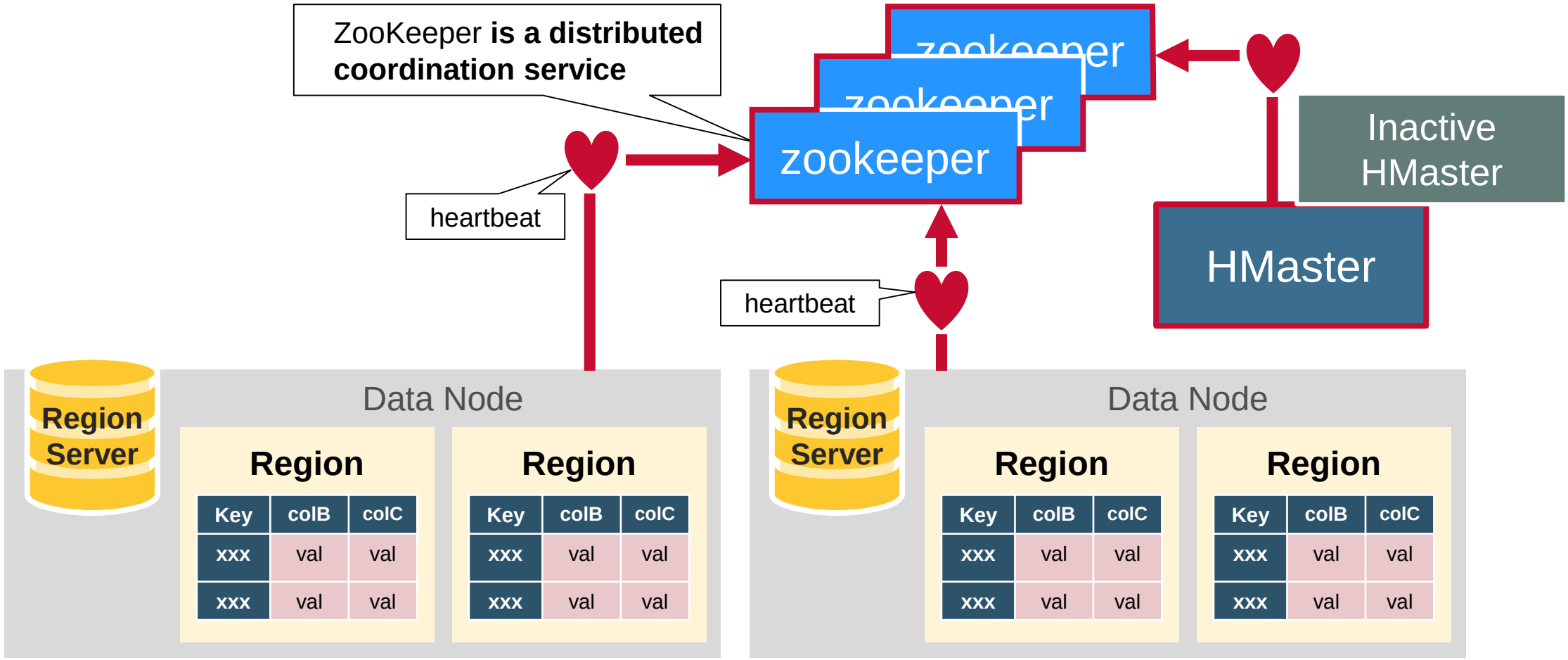
Regions



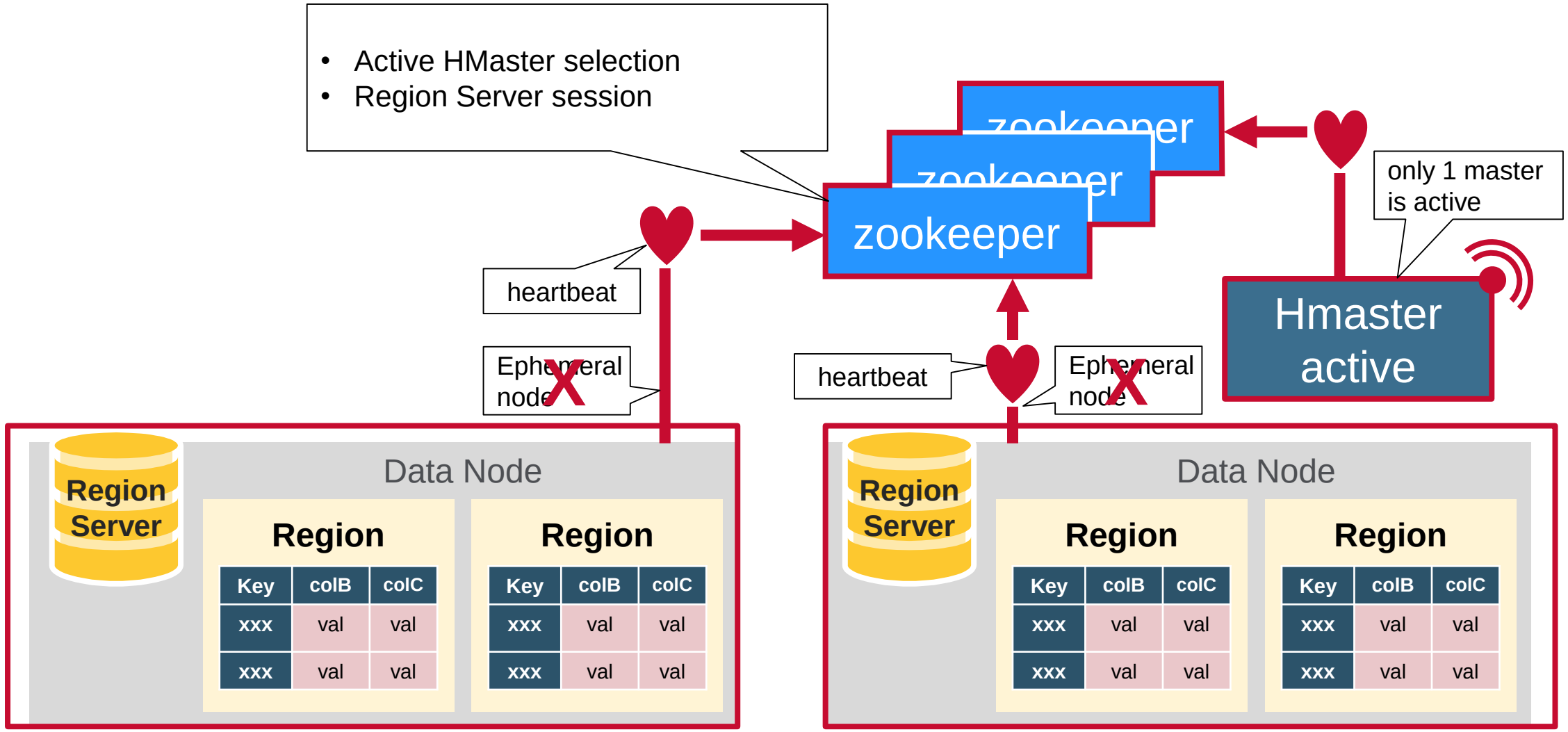
HBase HMaster



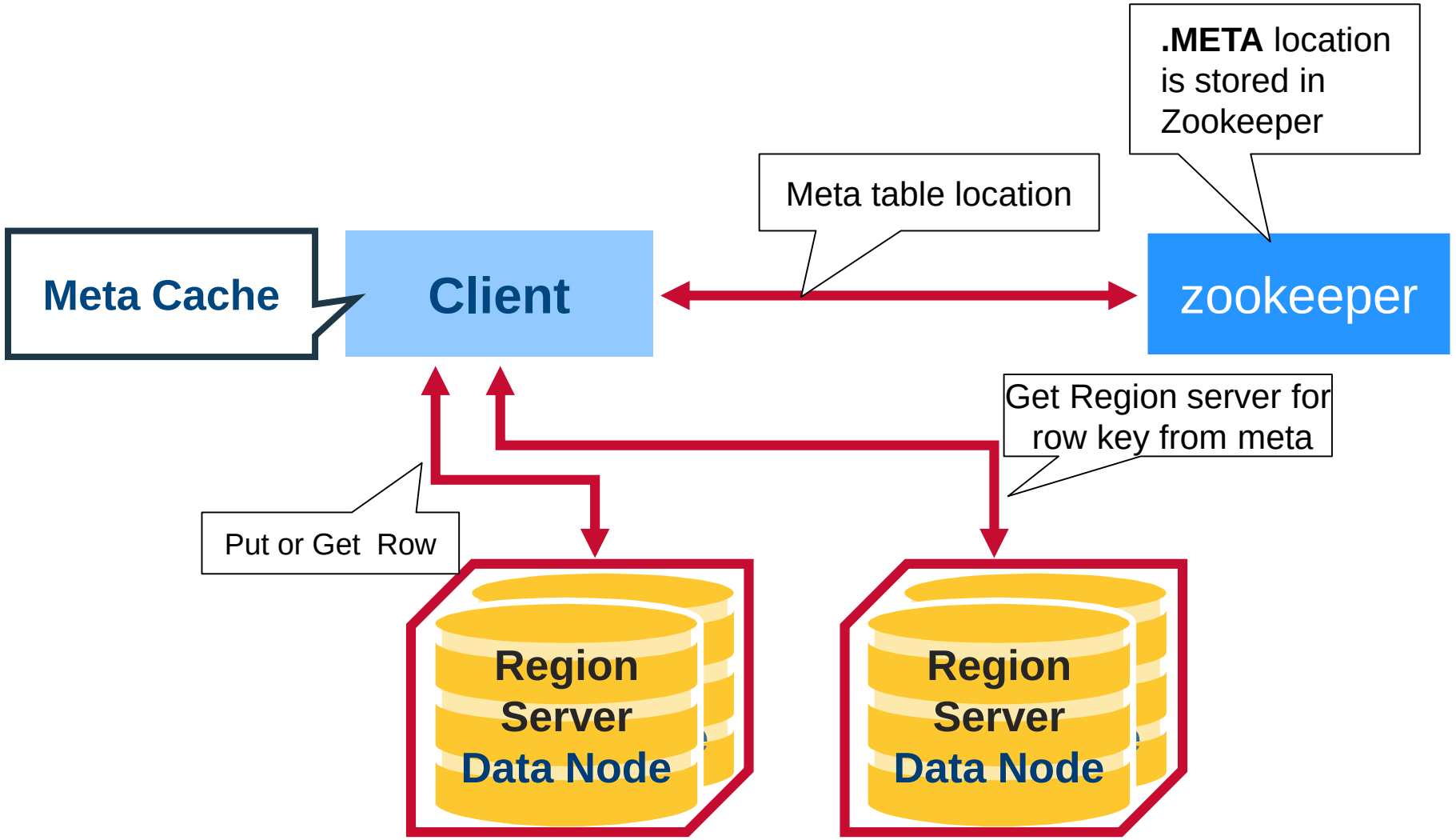
Zookeeper The Coordinator



How the Components Work Together



HBase First Read or Write



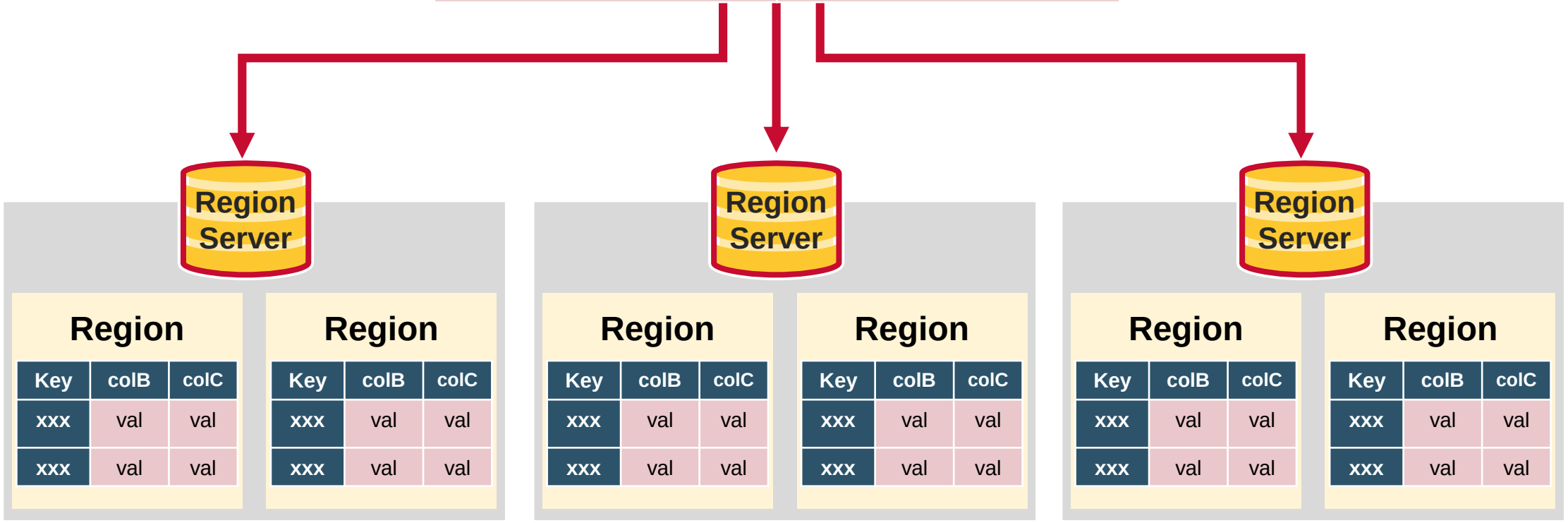
HBase Meta Table

Meta table is used to find the Region for a given Table key

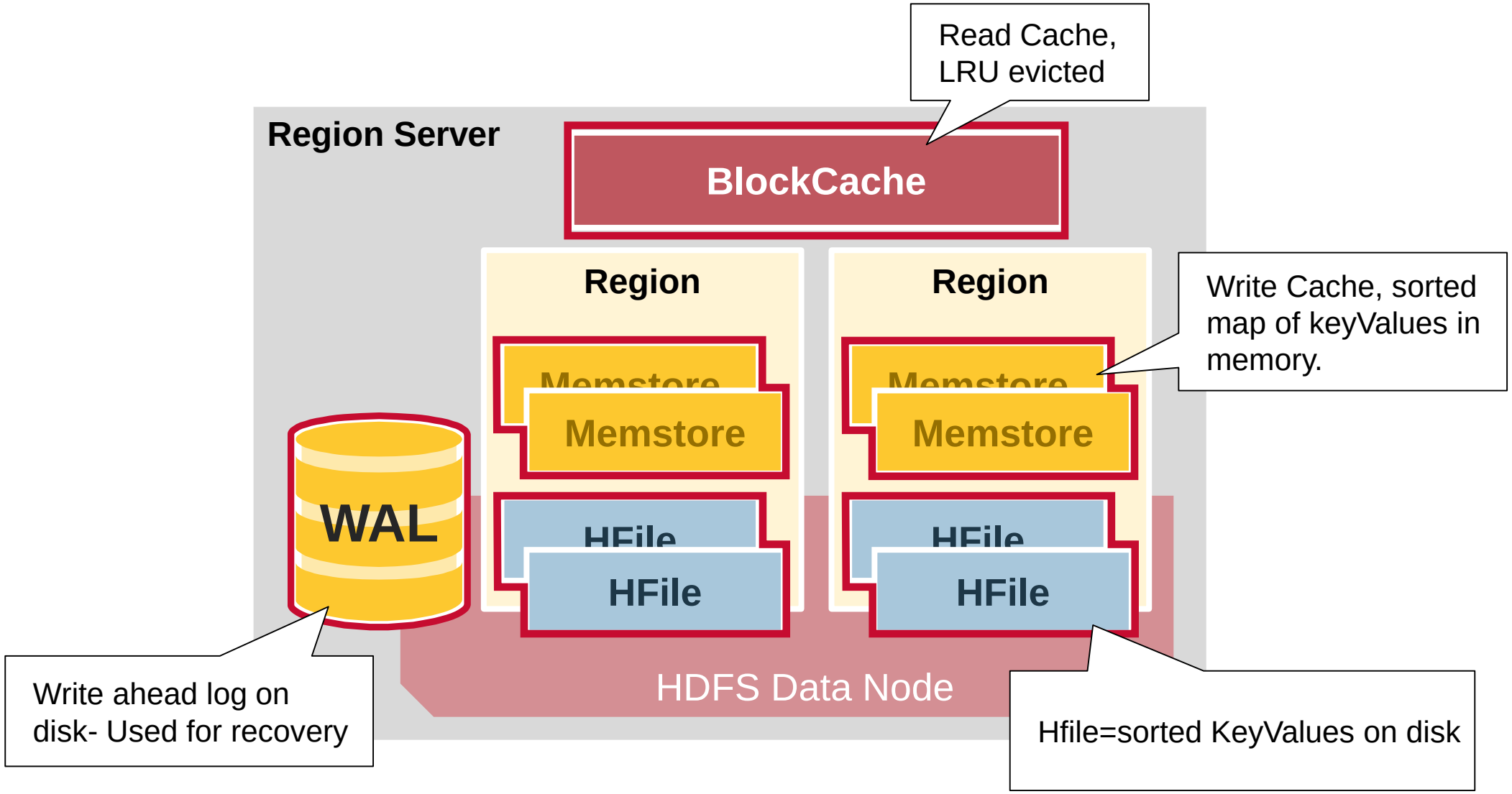
META table

Row key	Value
table,key,region	region server

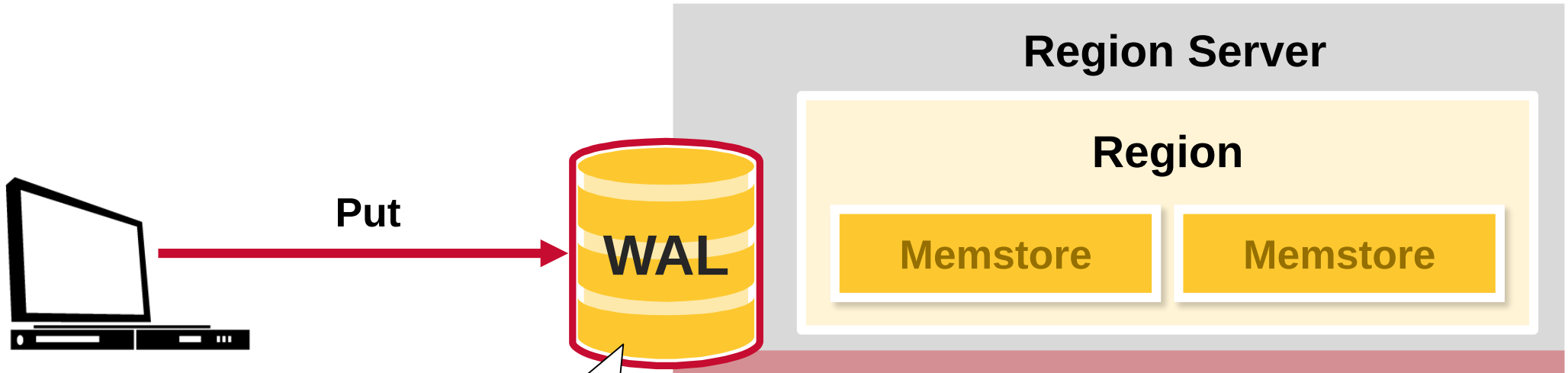
B tree



Region Server Components



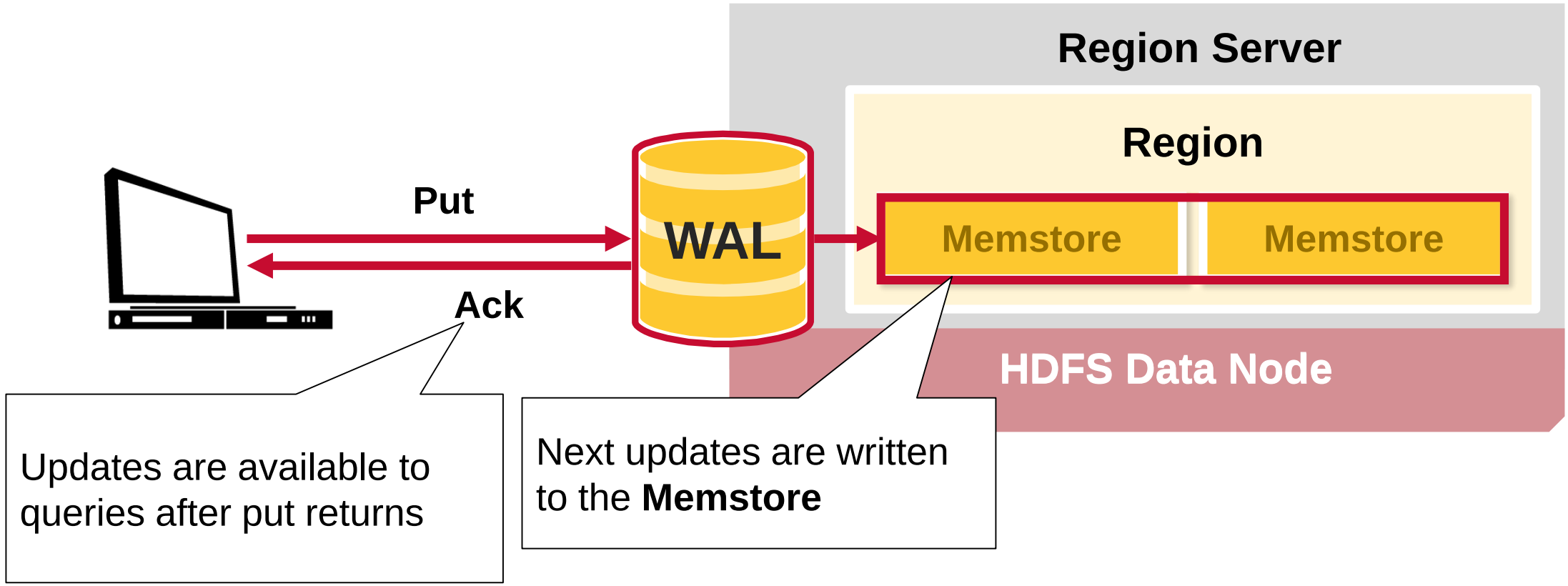
HBase Write Steps



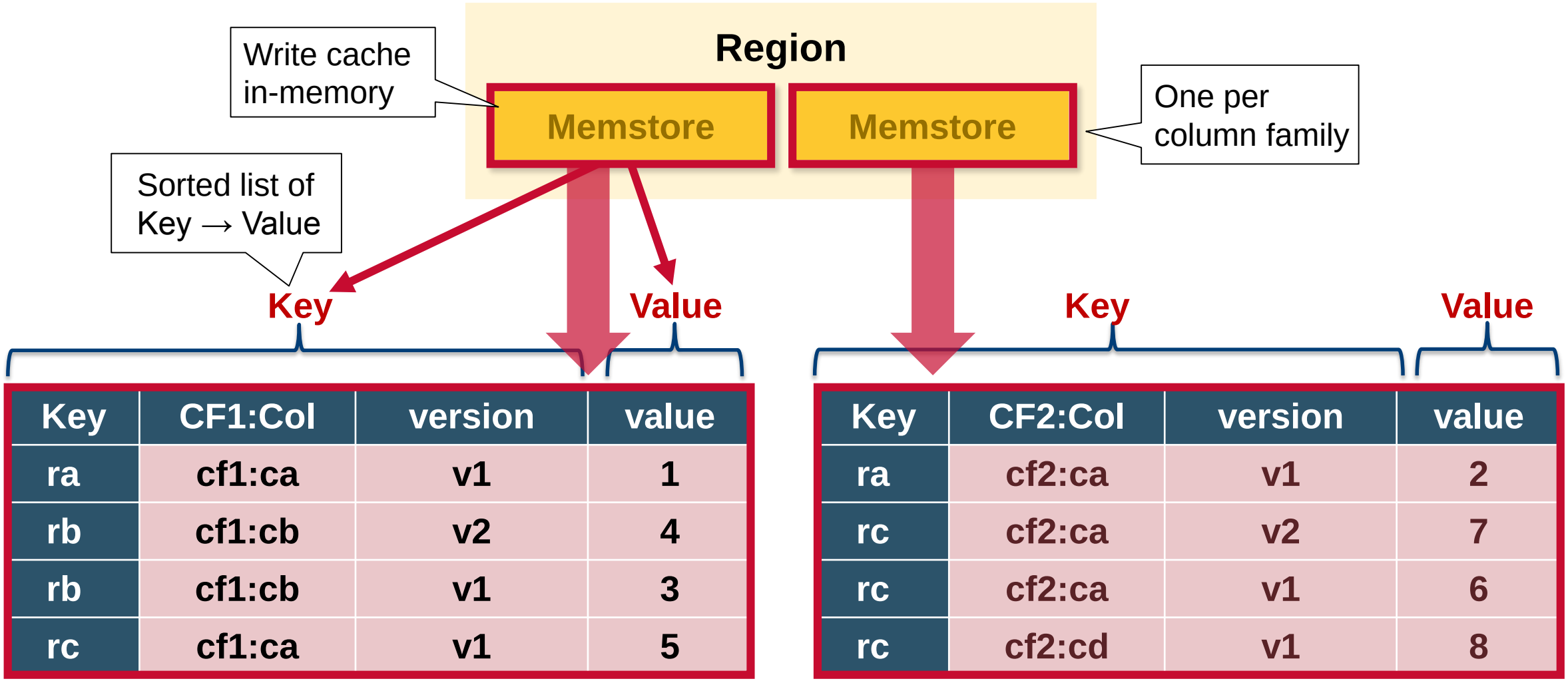
Each incoming record written to **WAL** for durability:

- log on disk
- updates appended sequentially

HBase Write Steps – (2)



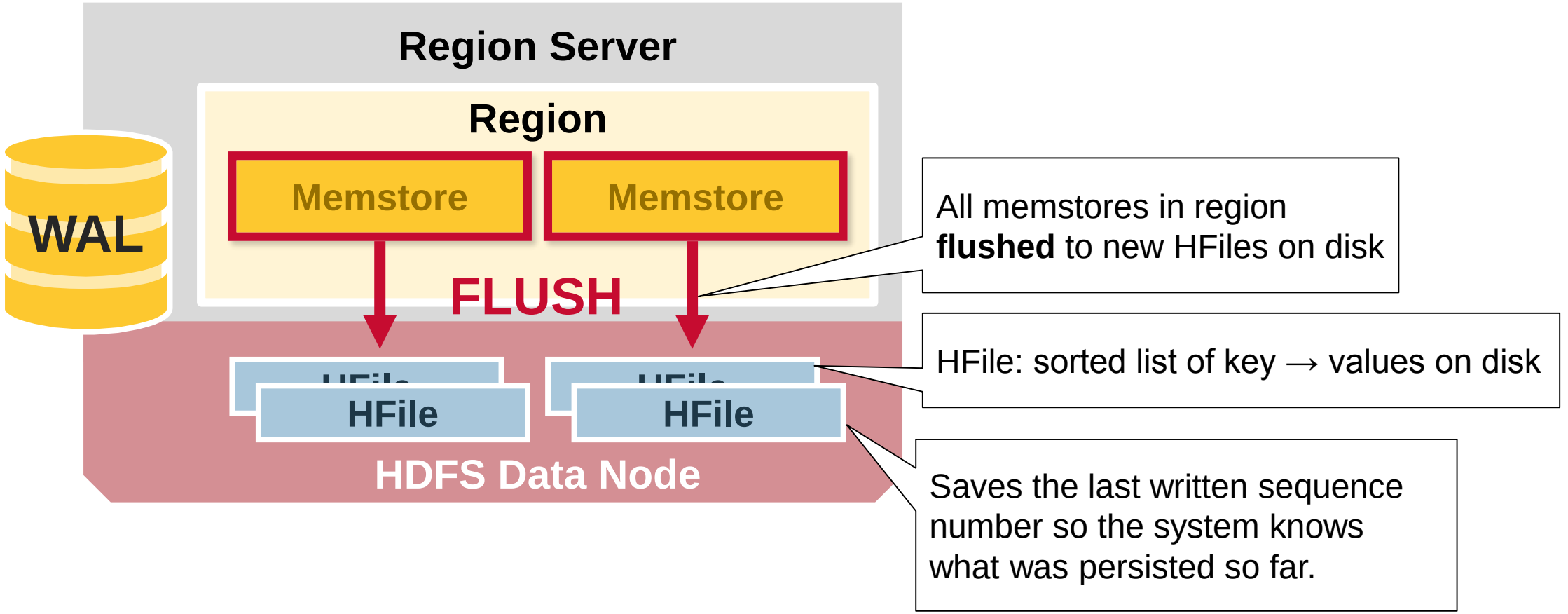
HBase Memstore



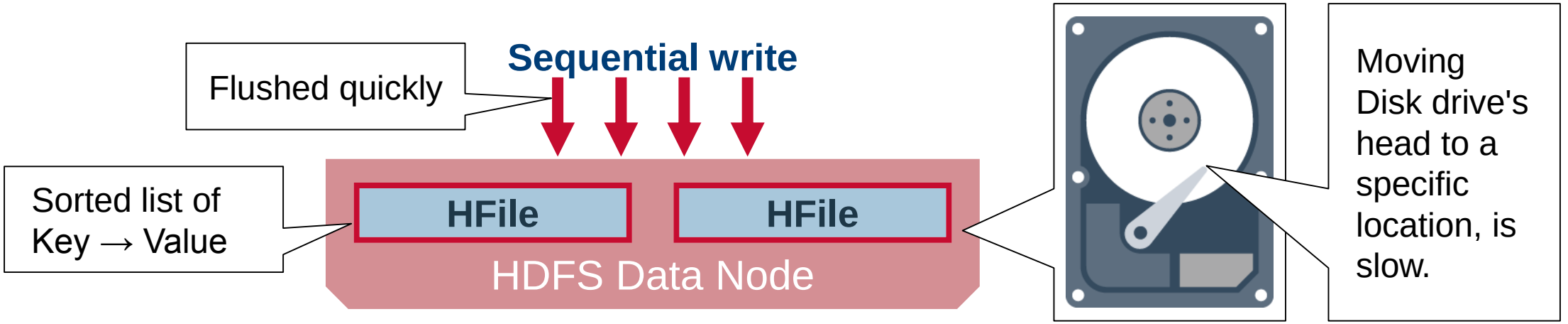
Key	CF1:Col	version	value
ra	cf1:ca	v1	1
rb	cf1:cb	v2	4
rb	cf1:cb	v1	3
rc	cf1:ca	v1	5

Key	CF2:Col	version	value
ra	cf2:ca	v1	2
rc	cf2:ca	v2	7
rc	cf2:ca	v1	6
rc	cf2:cd	v1	8

HBase Region Flush



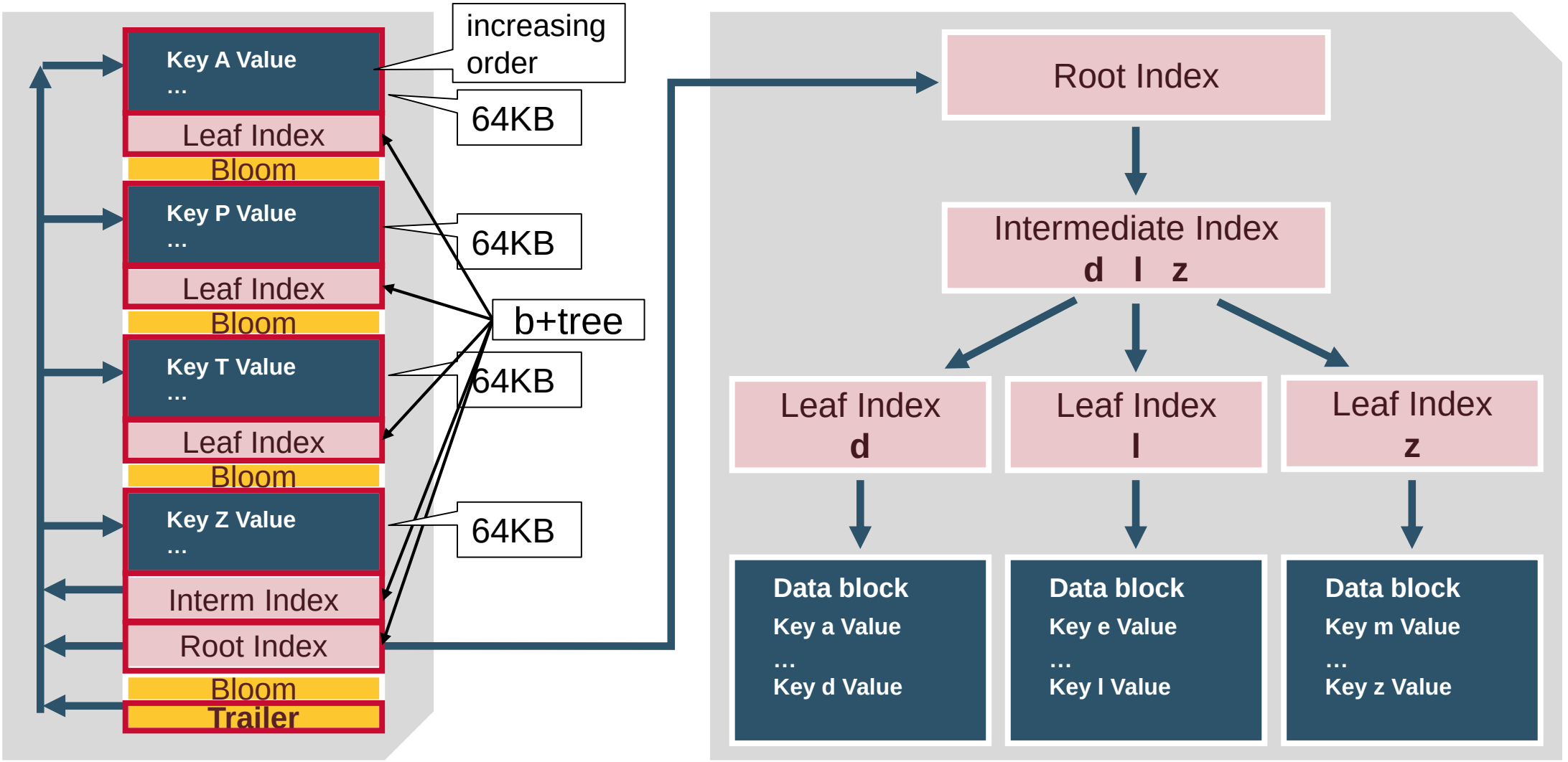
HBase HFile



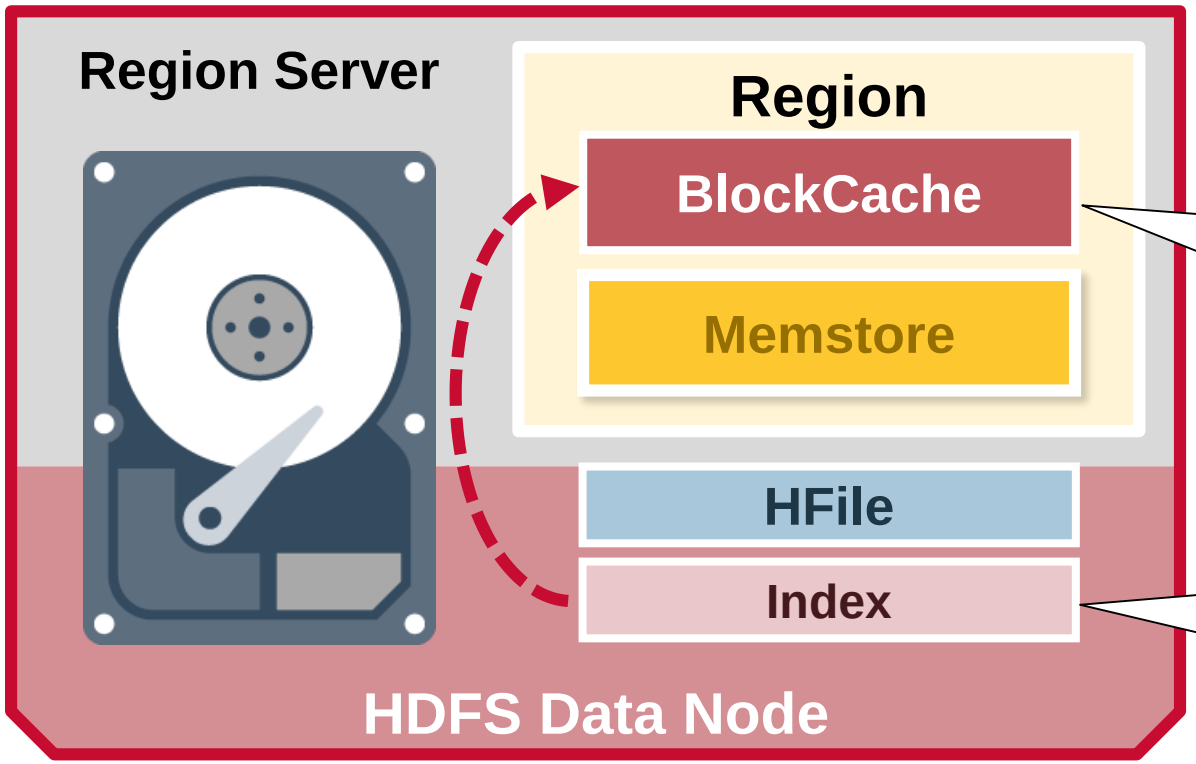
Key	CF1:Col	version	value
ra	cf1:ca	v1	1
rb	cf1:cb	v2	4
rb	cf1:cb	v1	3
rc	cf1:ca	v1	5

Key	CF2:Col	version	value
ra	cf2:ca	v1	2
rc	cf2:ca	v2	7
rc	cf2:ca	v1	6
rc	cf2:cd	v1	8

HBase HFile Structure



HFile Index



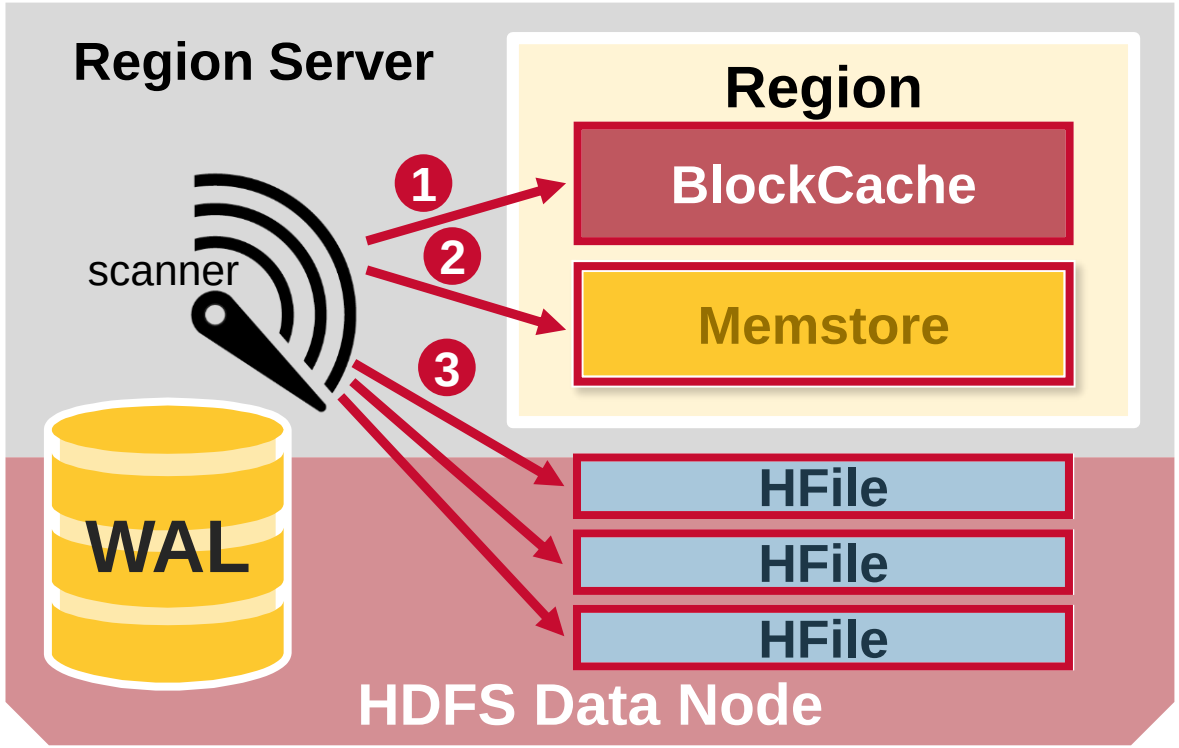
Lookups can be performed with a single disk seek

Hfile Index is kept in Block Cache (Memory)

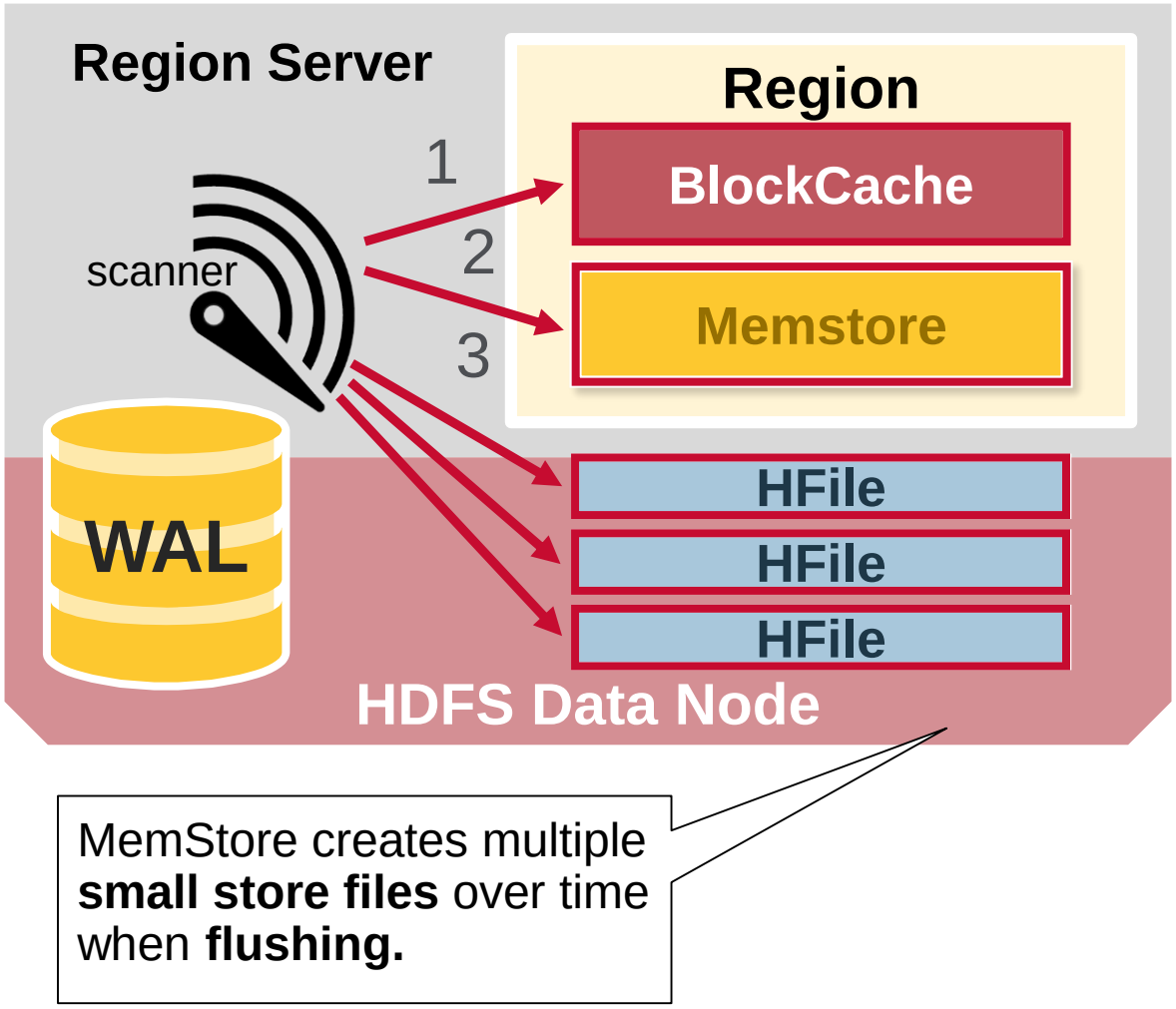
Index is loaded when the HFile is opened

HBase Read Merge

1. First the scanner looks for the Row KeyValues in the Block cache
2. Next the scanner looks in the Memstore
3. If all row cells not in memstore or blockCache, look in HFiles



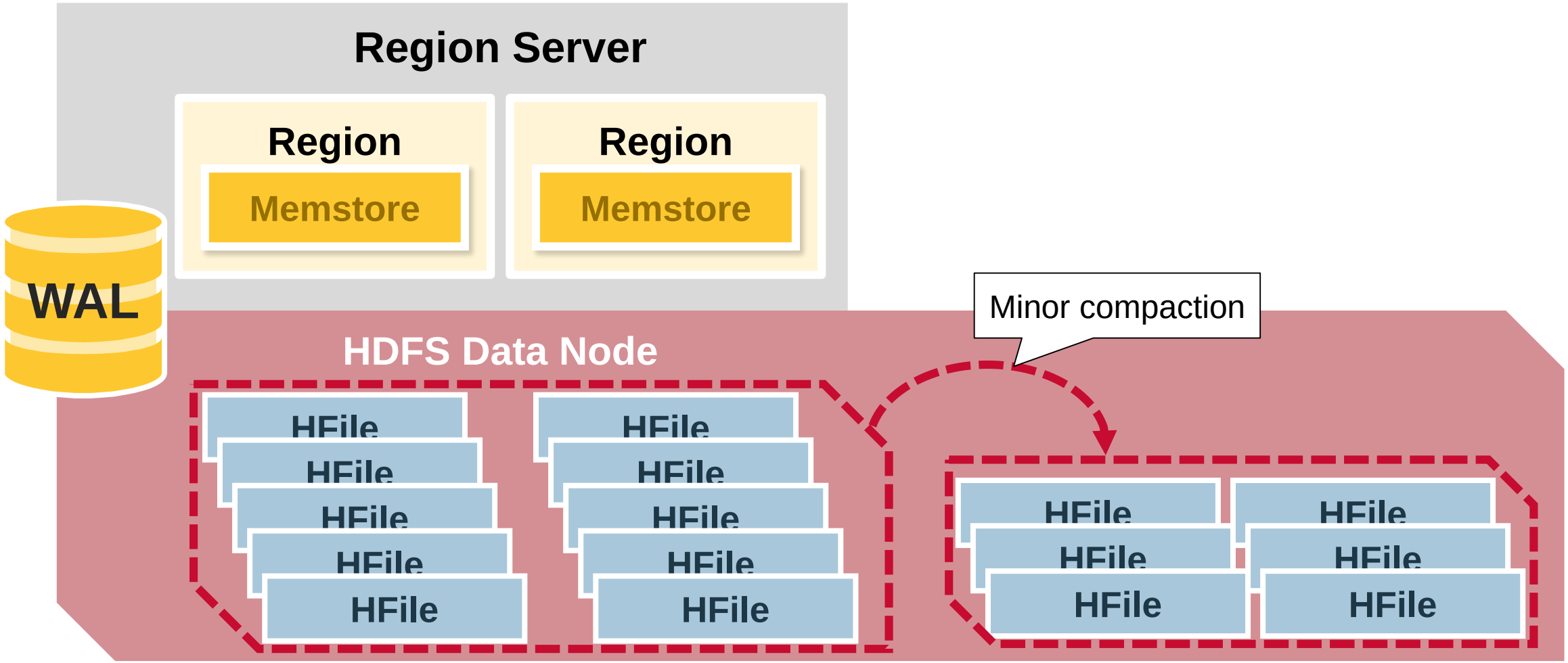
HBase Read Merge



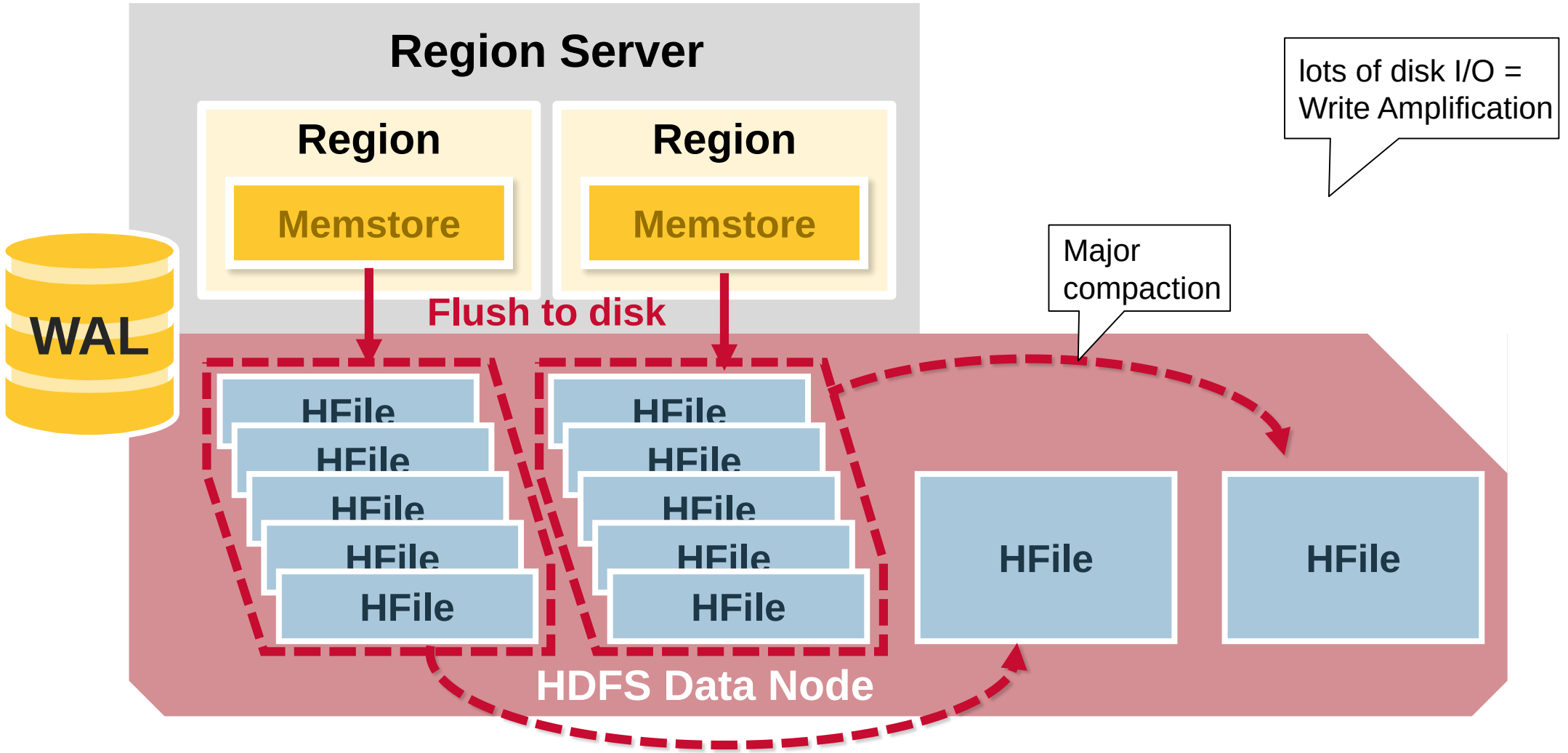
Read Amplification

- multiple files have to be examined

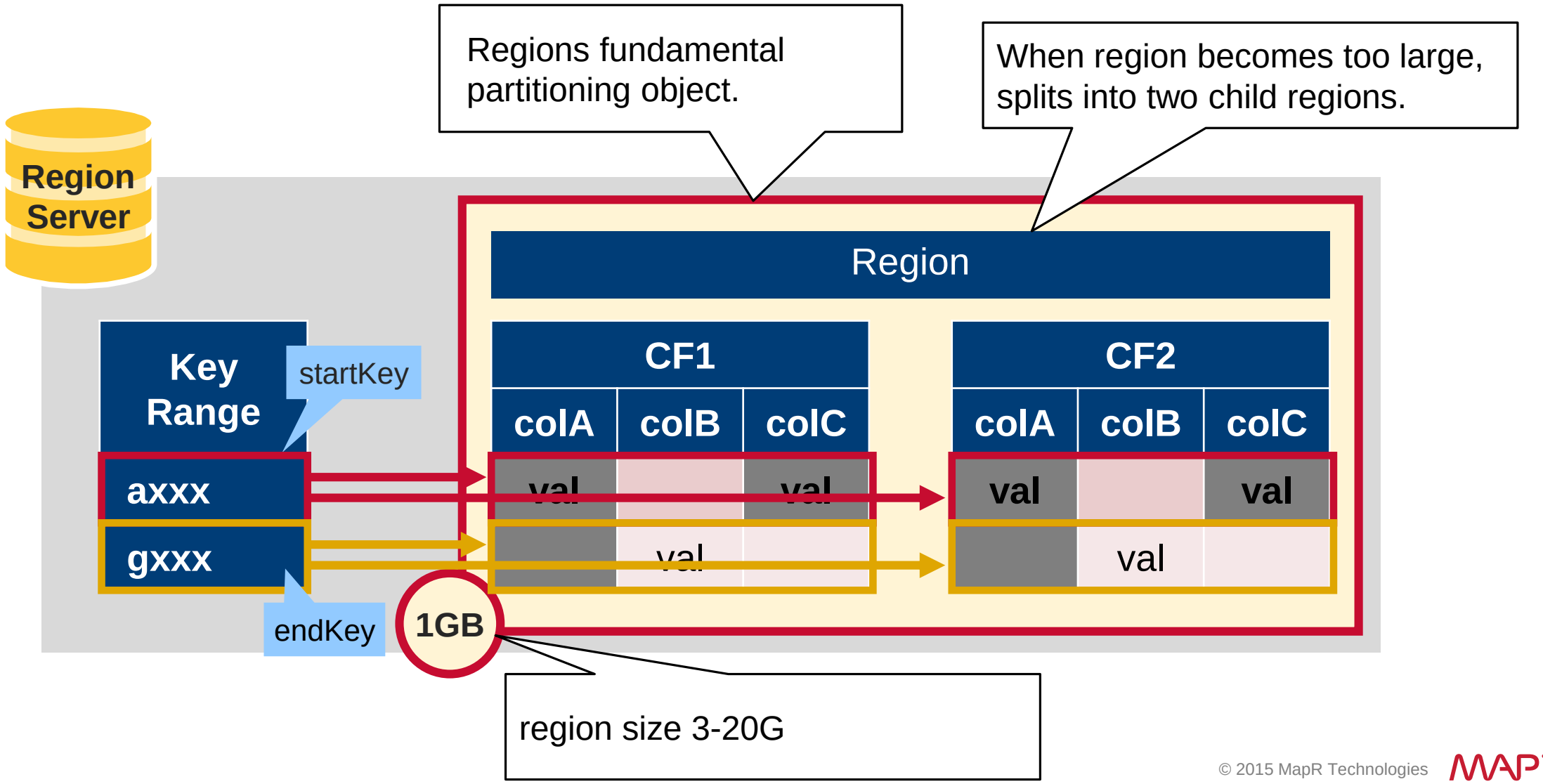
HBase Minor Compaction



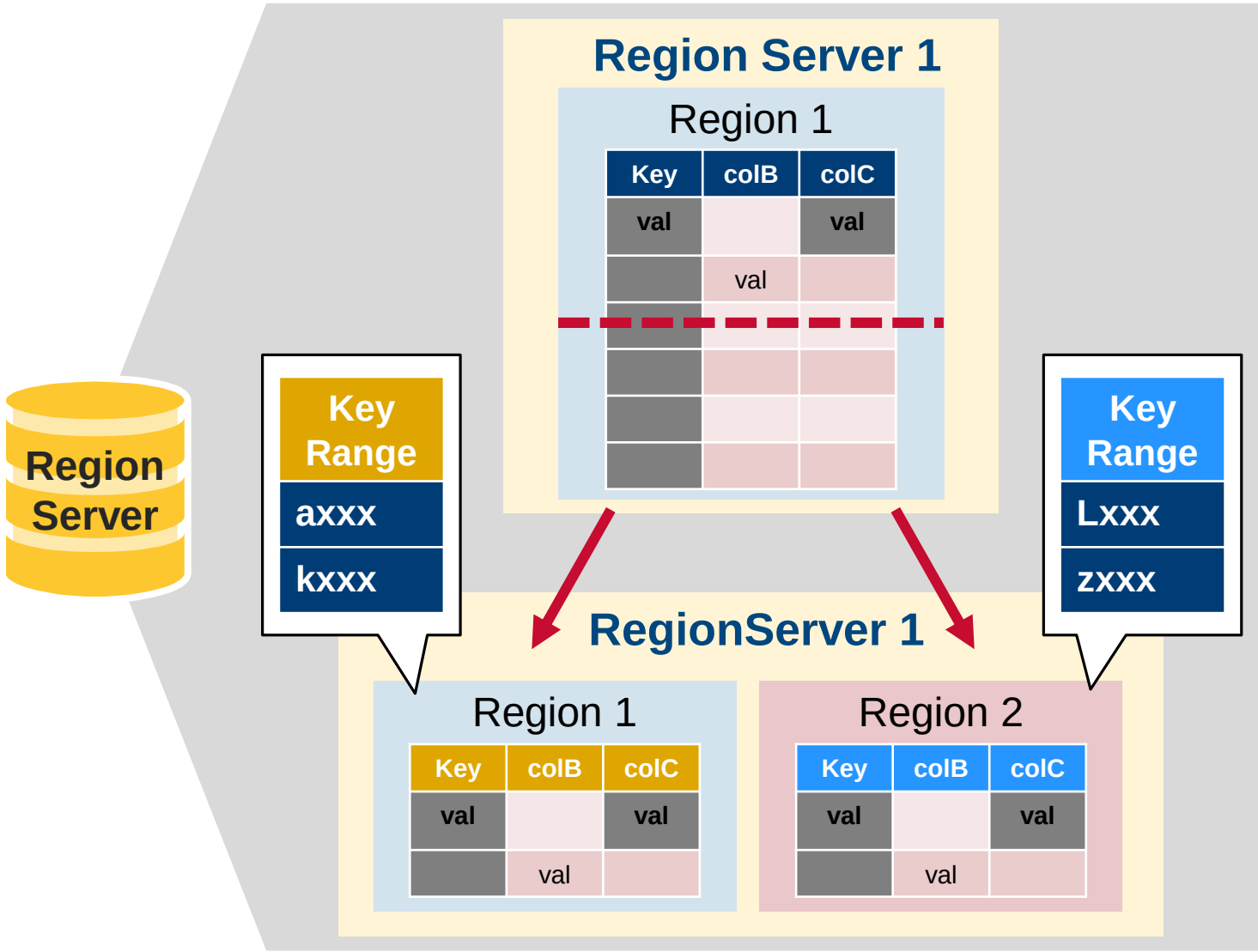
HBase Major Compaction



Region = contiguous keys

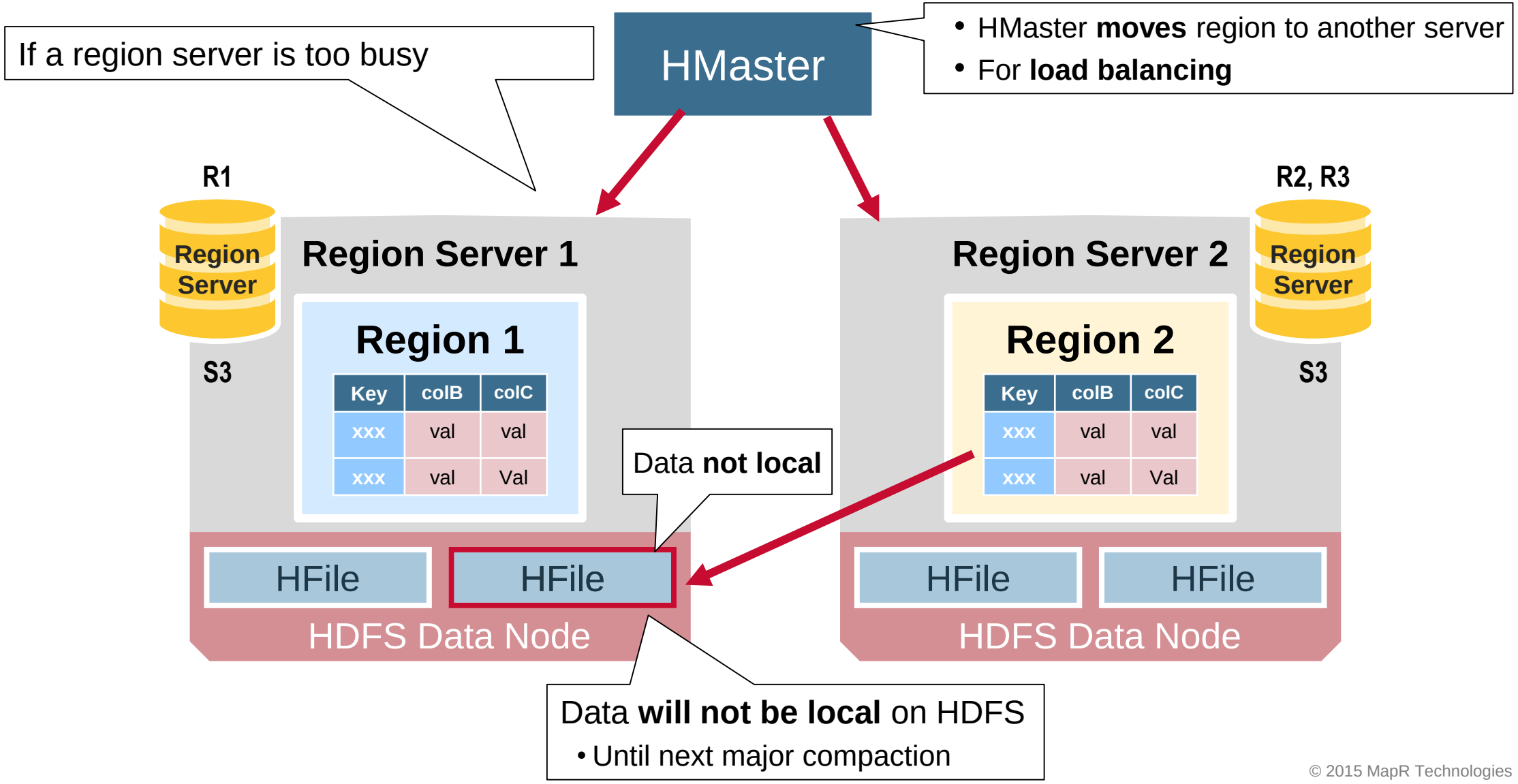


Region Split

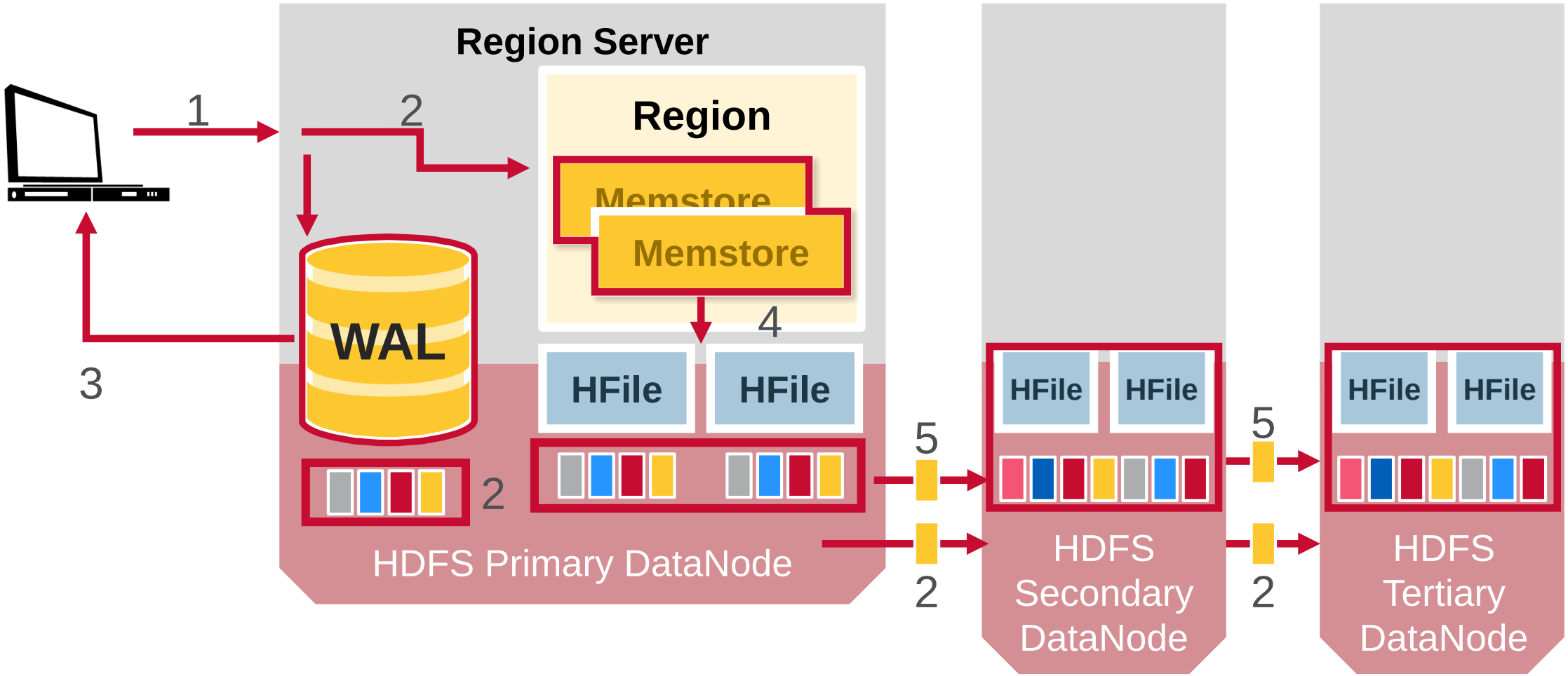


when region size > hbase.hregion.max.filesize → split

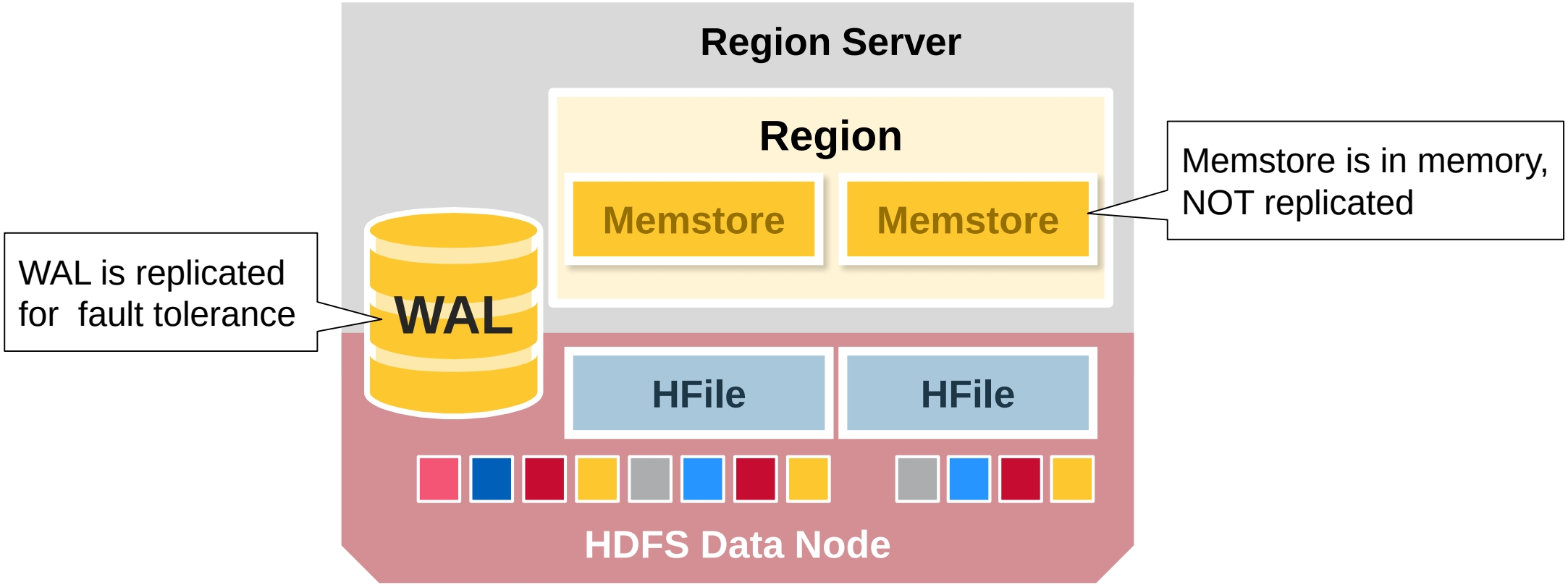
Region Load Balancing



HDFS Data Replication

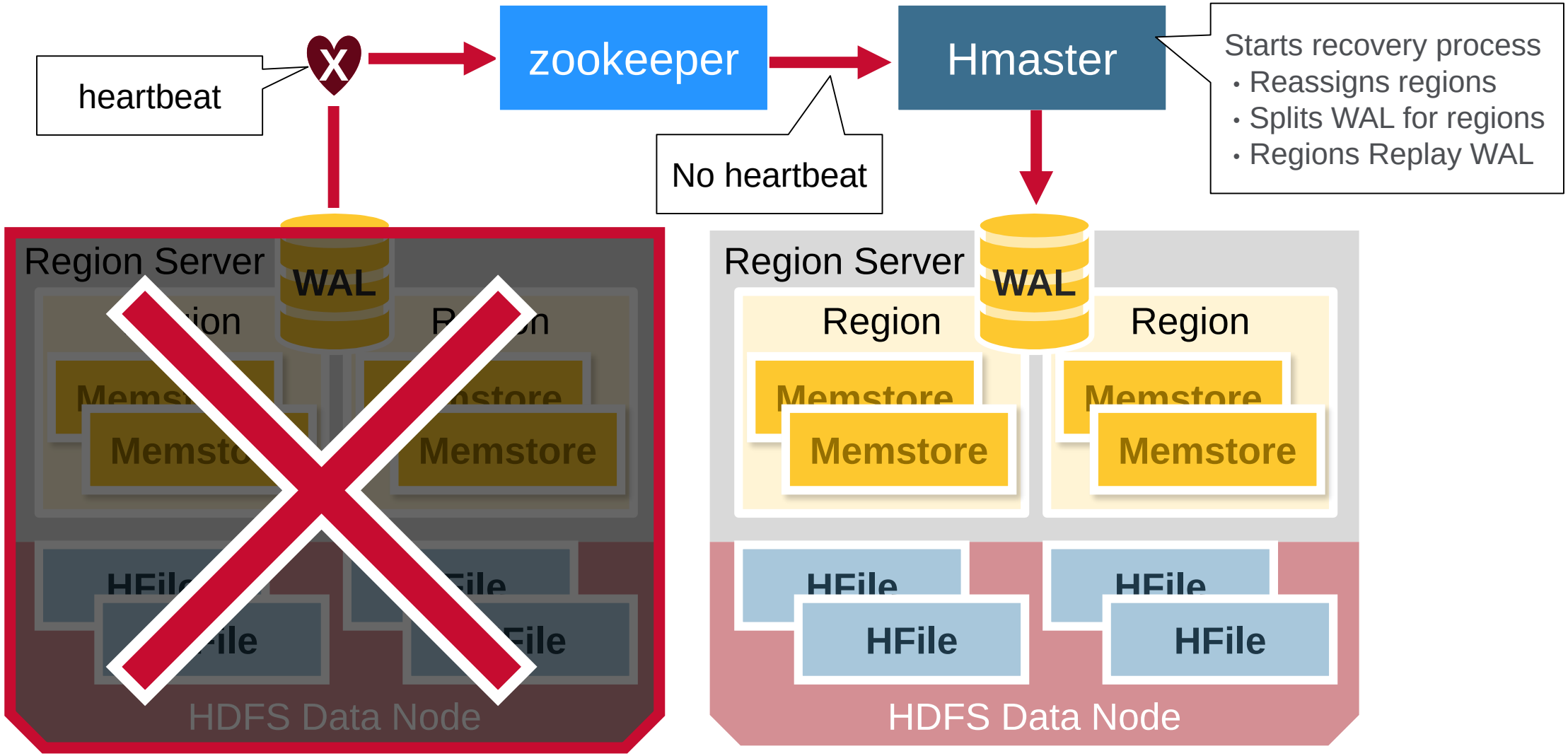


HDFS Data Replication – (2)

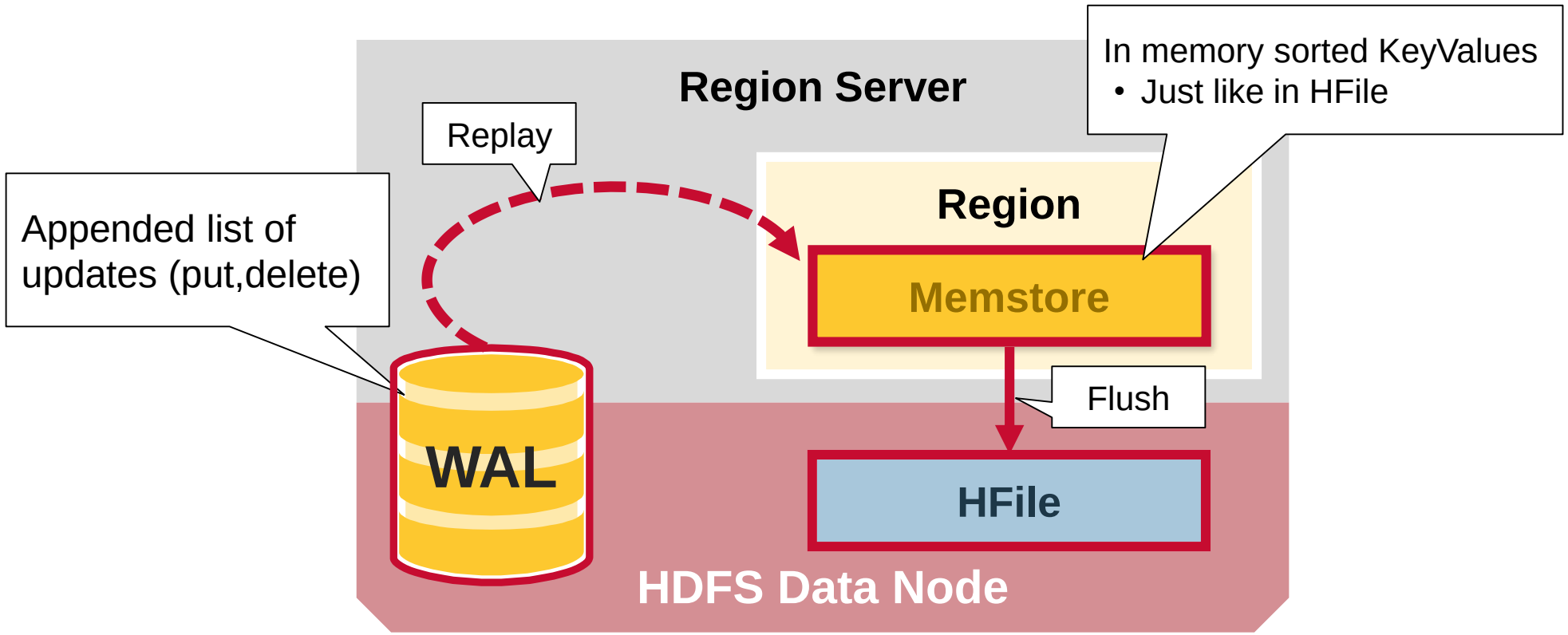


How does HBase recover updates not persisted to HFiles?

HBase Crash Recovery



Data Recovery



Apache HBase Architecture Benefits

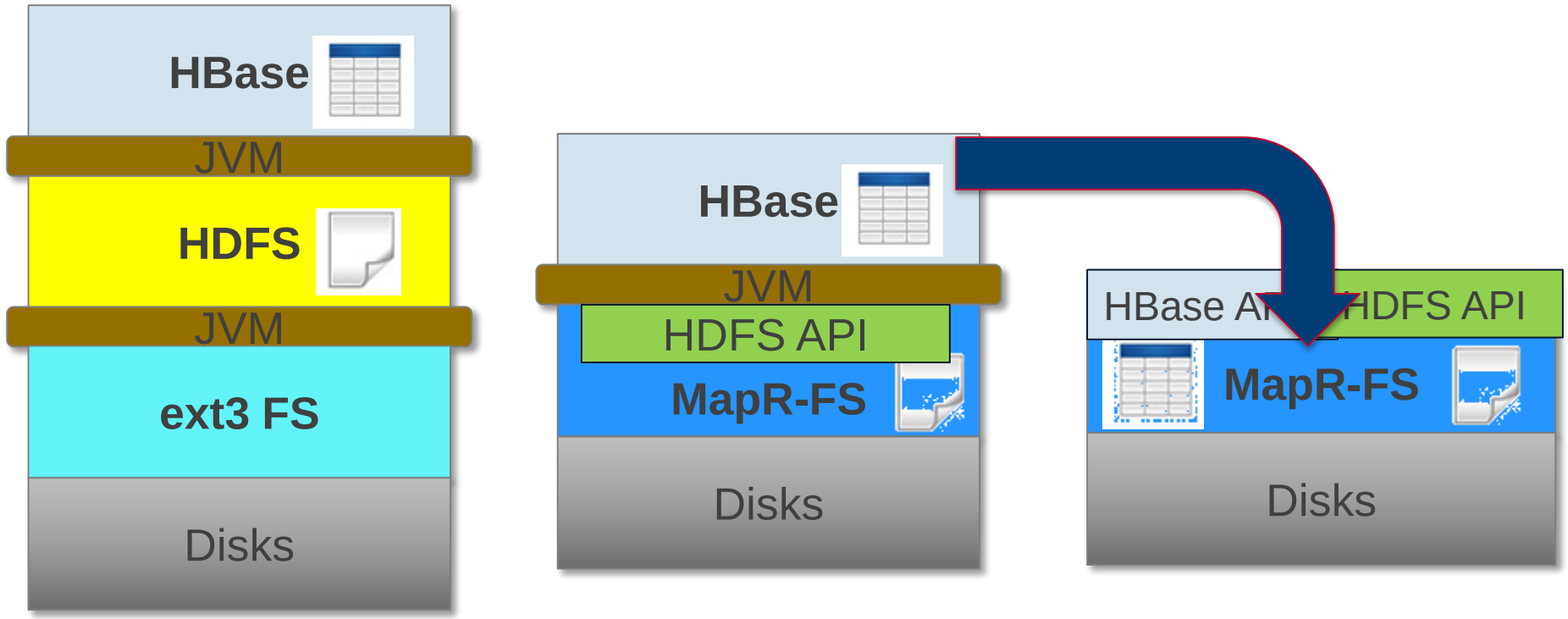
Better than many NoSQL data store solutions, hence its popularity

- **Strong consistency model**
  - When a write returns, all readers will see same value
- **Scales automatically**
  - Regions **split** when data grows too large
  - Uses HDFS to **spread** and **replicate data**
- **Built-in recovery**
  - Using **Write Ahead Log** (similar to journaling on file system)
- **Integrated with Hadoop**
  - MapReduce on HBase is straightforward

MapR-DB

MapR-DB and Files in a Unified Storage Layer

- MapR**-DB Tables and** MapR-FS **Files** in a **unified read/write filesystem**.



Apache HBase on Hadoop

Apache HBase on MapR Filesystem

Mapr-DB Integrated into Filesystem

Tables Integrated into MapR read/write File System:

- **MapR-DB tables use the HBase data model and API**
- **Key differences between MapR tables and Apache HBase**
  - **Tables** part of the **MapR Read/Write File** system
  - **Guaranteed data locality**
  - **Smarter load balancing**
    - **Uses container Replicas**
  - **Smarter fail over**
    - **Uses container replicas**
  - Multiple small WALs
    - **Faster recovery**
  - **No compaction !**

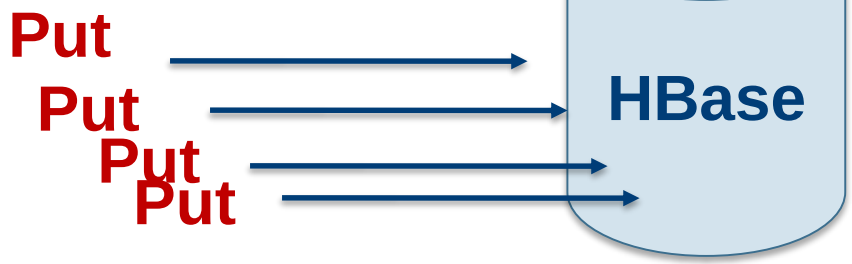
HBase Use Cases

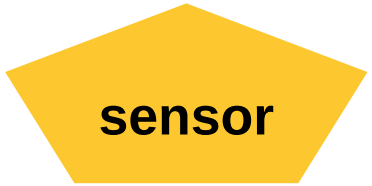
3 Main Use Case Categories

- Time Series Data, Stuff with a Time Stamp
  - Sensor, System Metrics, Events, log files
  - Stock Ticker, User Activity
  - Hi Volume, Velocity Writes

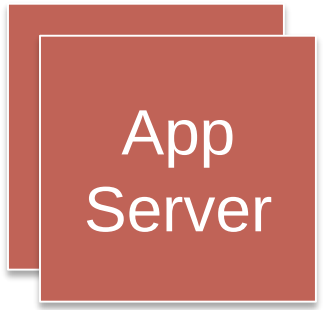
OpenTSDB

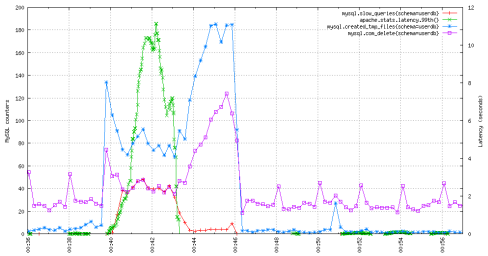
Event time stamped data





read

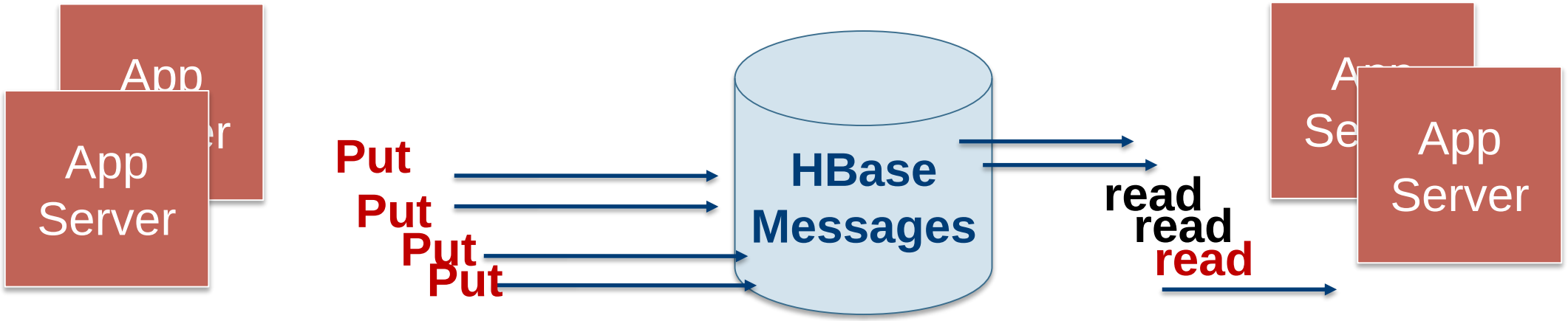




Data for real-time monitoring.

3 Main Use Case Categories

- Information Exchange
  - email, Chat, Inbox: **Facebook**
  - Hi Volume, Velocity Write/Read

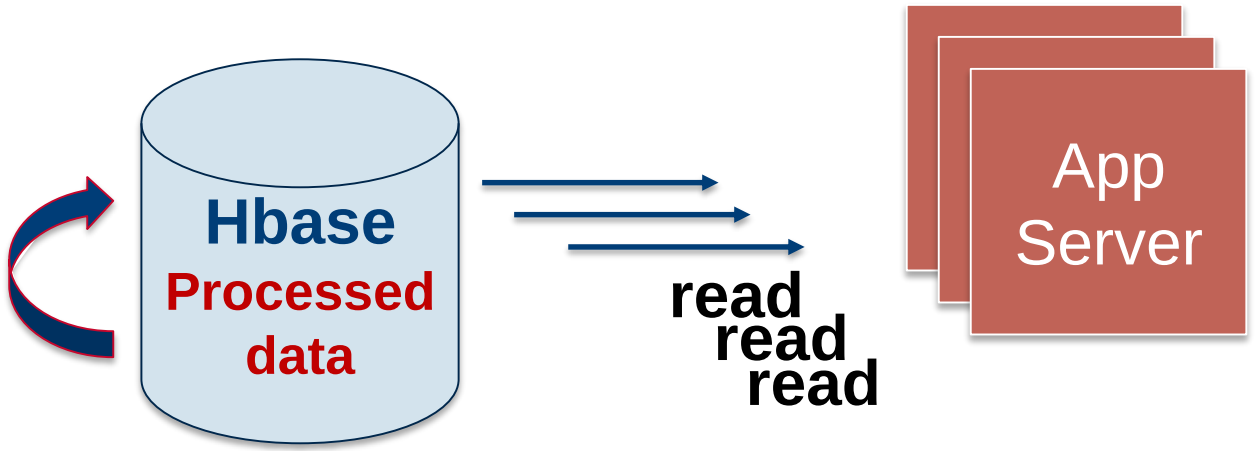


https://www.facebook.com/UsingHbase

3 Main Use Case Categories

- Content Serving, Web Application Backend
  - Online Catalog: Gap, World Library Catalog.
  - Search Index: ebay
  - Online Pre-Computed View: Groupon, Pinterest
  - Hi Volume, Velocity Reads

Bulk Import
Pre-Computed
Materialized View



Agenda

- Why do we need NoSQL / HBase?
- Overview of HBase & HBase data model
- HBase Architecture and data flow
- Demo/Lab using HBase Shell
  - Create tables and CRUD operations using MapR Sandbox
- Design considerations when migrating from RDBMS to HBase
- HBase Java API to perform CRUD operations
  - Demo / Lab using Eclipse, HBase Java API & MapR Sandbox
- How to work around transactions

Schema Design Guidelines

- HBase tables ≠ Relational tables!
  - HBase Design for Access Paterns

Use Case Example

Record Stock Trade Information in a Table

Trade data

Trade

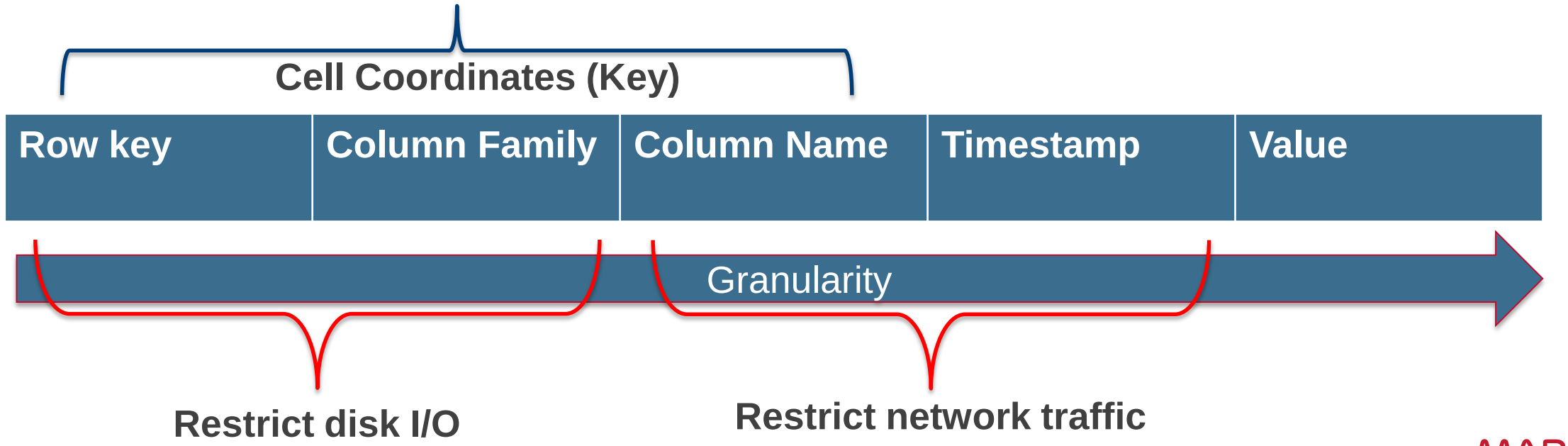
- Timestamp
- Stock symbol
- Price per share
- Volume of trade

Example

- 1381396363000 (epoch timestamp with millisecond granularity)
- AMZN
- $304.66
- 1333 shares

Intelligent keys

- Only the row keys are indexed
- **Compose the key** with attributes used for **searching**
  - Composite key : 2 or more **identifying** attributes
  - Like **multi-column index design** in RDB



Composite Keys

- Use composite row key:
  - Include multiple elements in the row key
  - Use a separator or fixed length
- Example row key format:

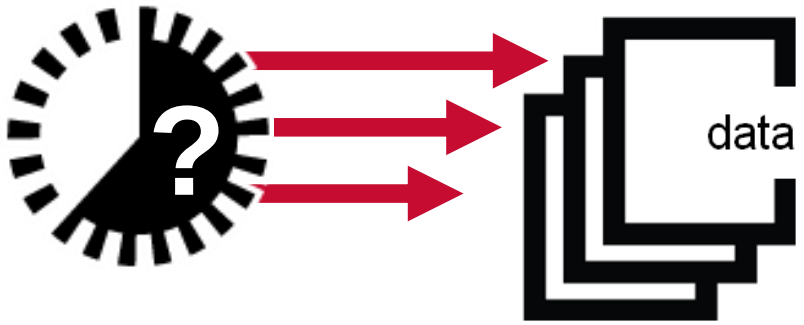
SYMBOL	+	DATE (YYYYMMDD)

  - Ex: GOOG_20131012
- Scans can use partial keys
  - Ex: “GOOG” or "GOOG_2014"

Consider Access Patterns for Application

How will data be retrieved?

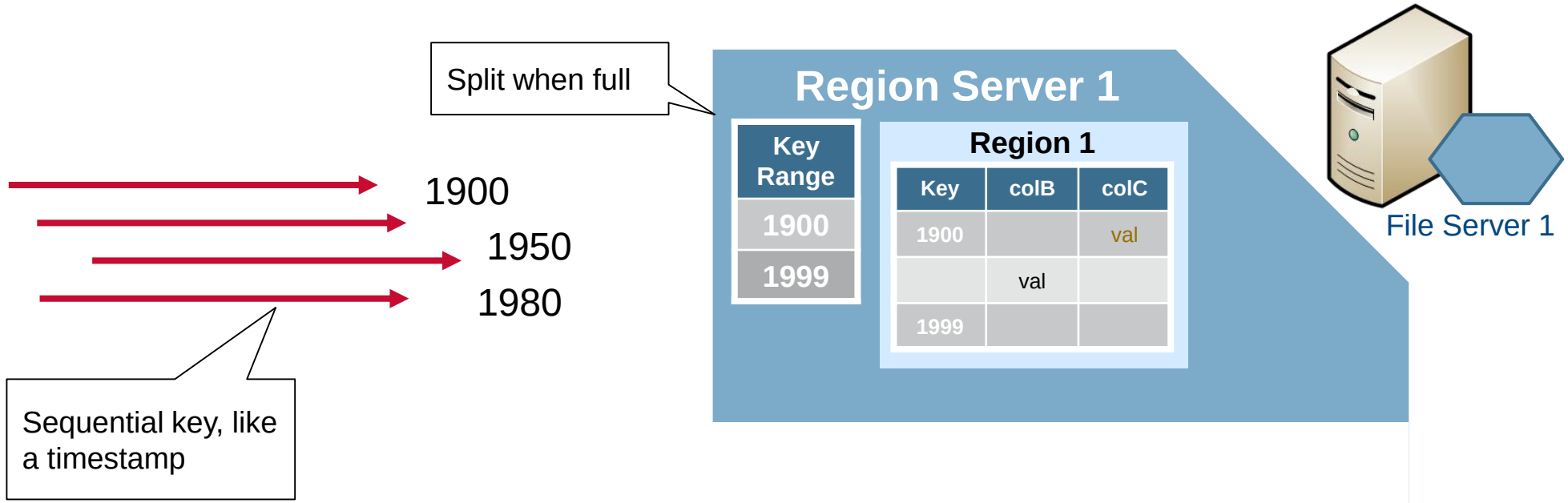
- By date? By hour? By companyId?
  - ***Rowkey design***



What if the Date/Timestamp is leftmost ?

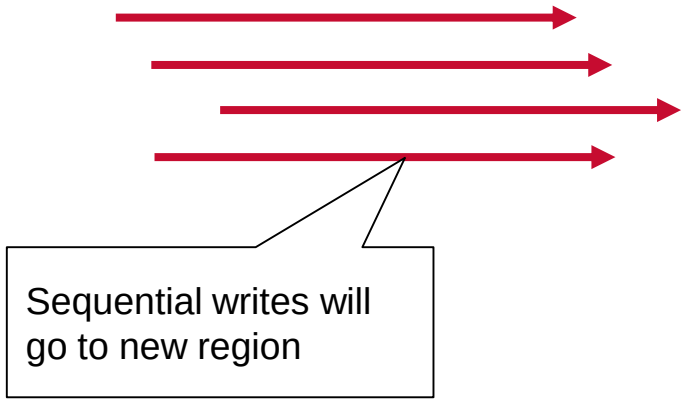
Key				
1391813876369_AMZN				
1391813876370_AMZN				
1391813876371_GOOG				

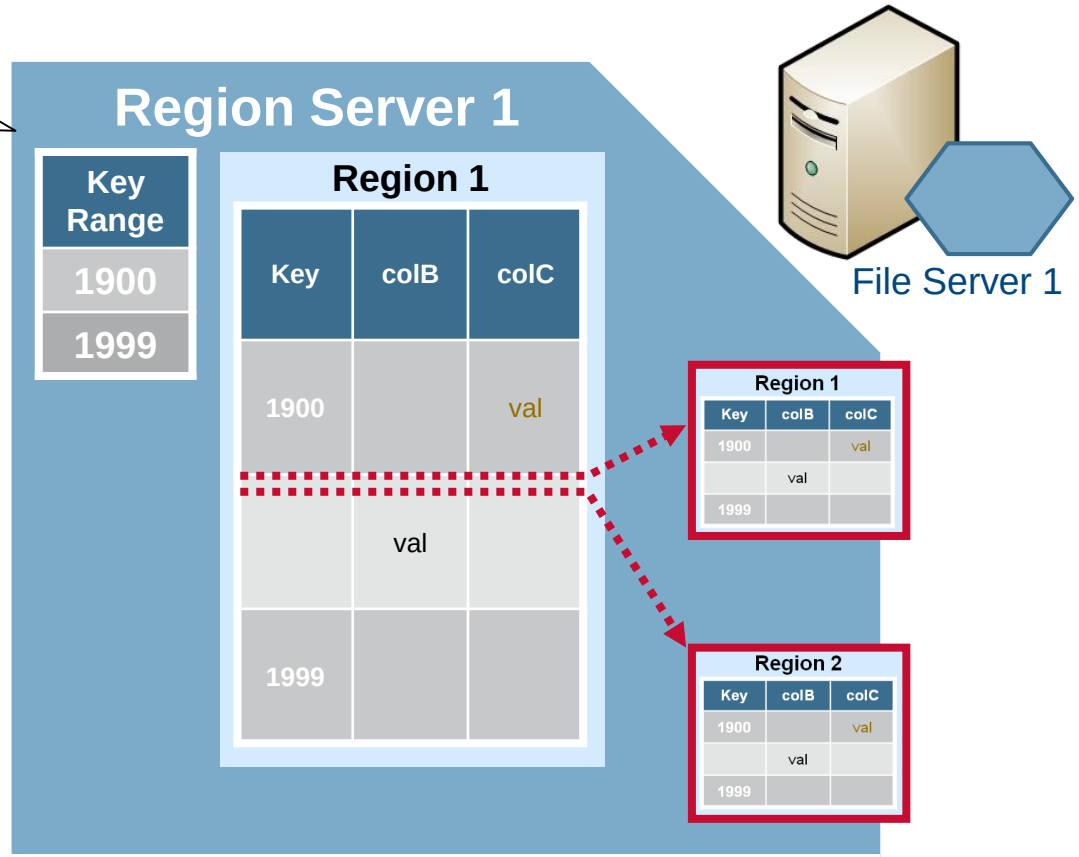
Hot-Spotting & Region Splits



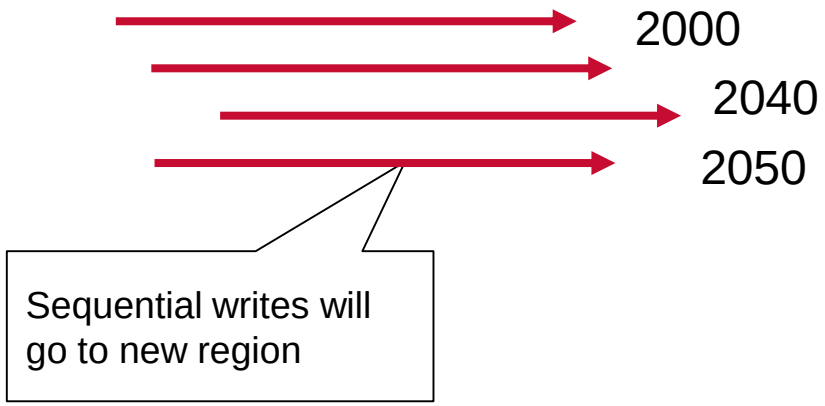
Hot-Spotting & Region Splits

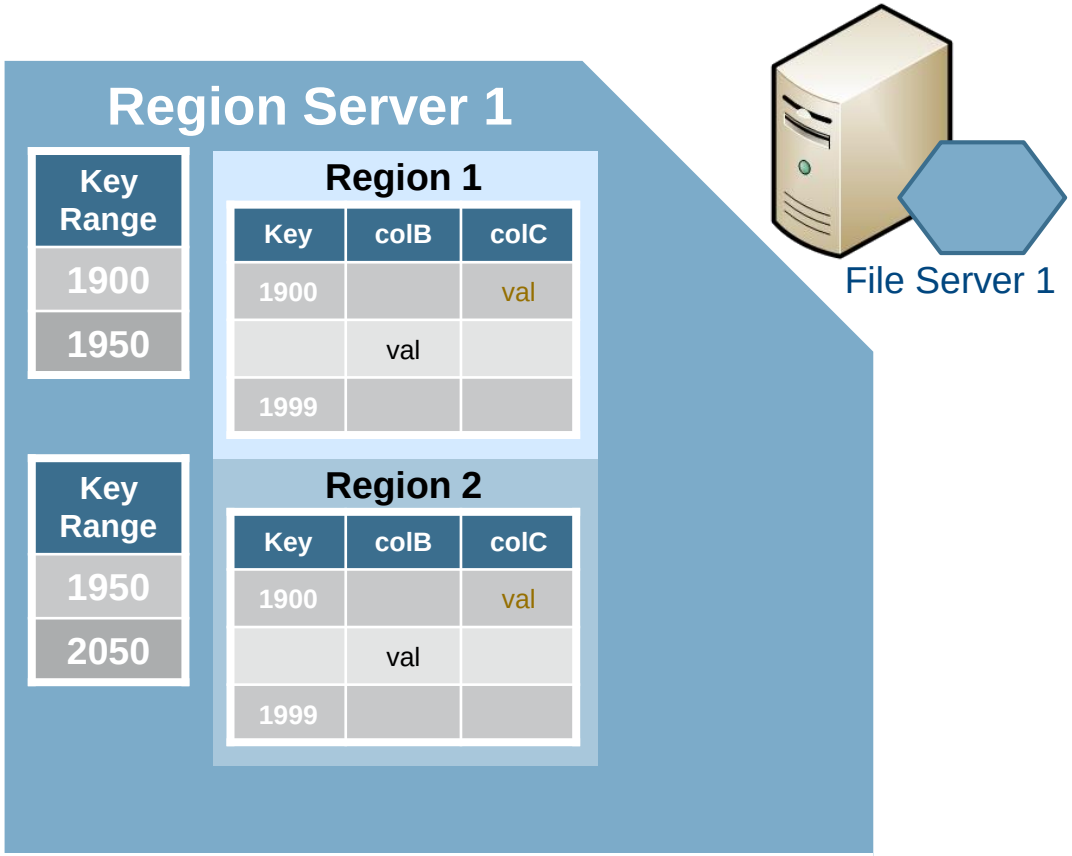
Region Server Creates two new regions, each with **half of the original regions keys.**



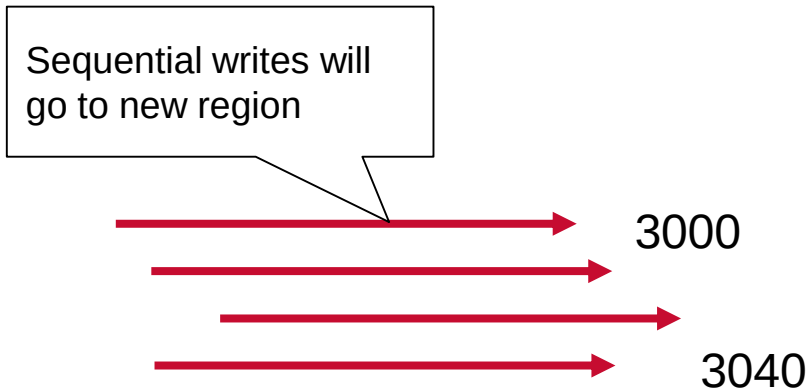


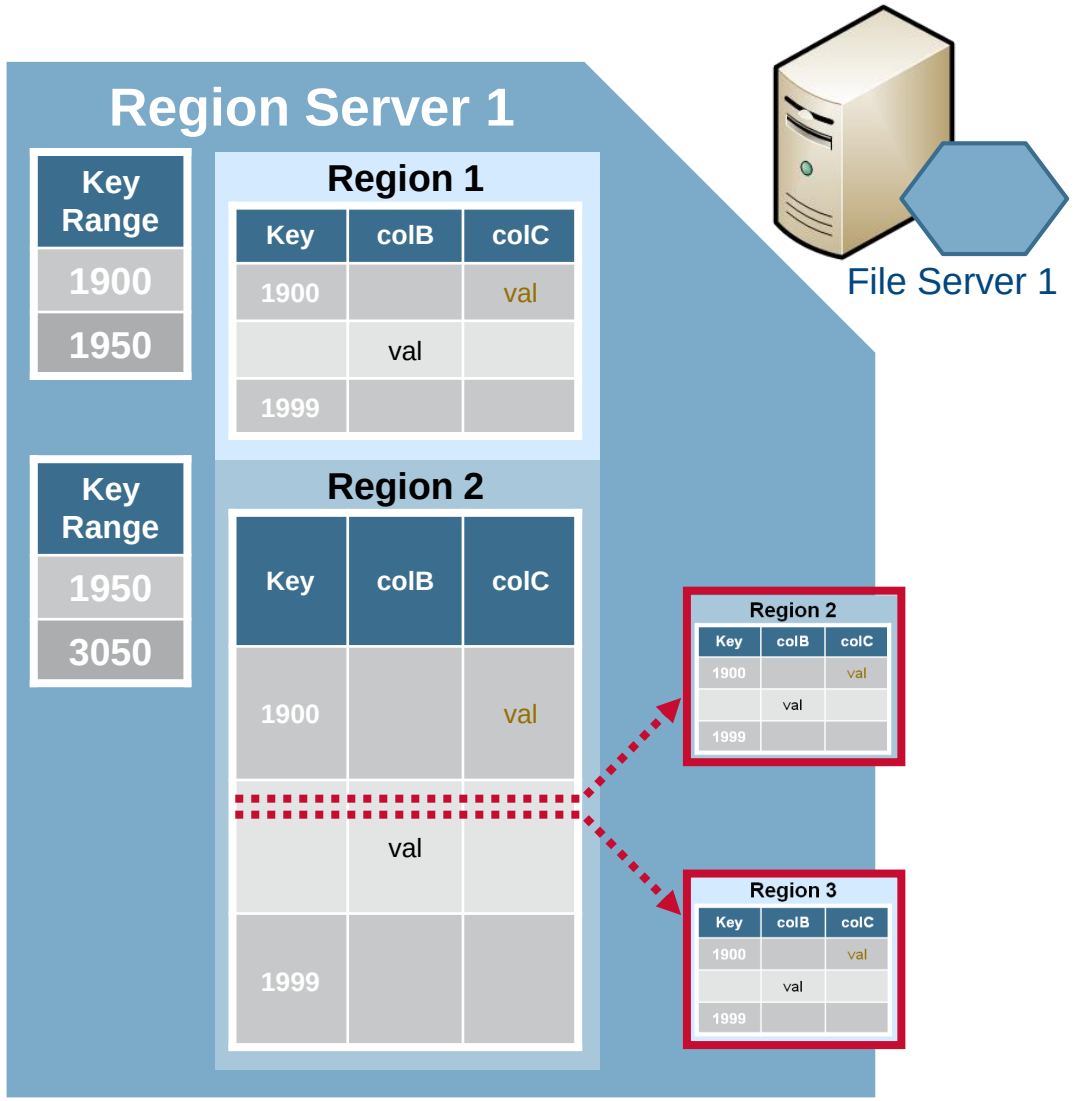
Hot-Spotting & Region Splits



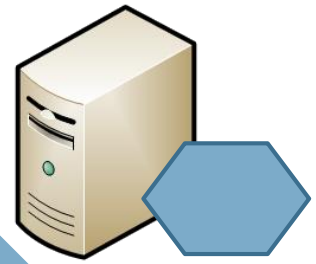


Hot-Spotting & Region Splits





Hot-Spotting & Region Splits



Region Server 1

Key Range
1900
1950

Region 1

Key	colB	colC
1900		val
	val	
1999		

Key Range
1950
2050

Region 2

Key	colB	colC
1900		val
	val	
1999		

Key Range
2051
3050

Region 3

Key	colB	colC
1900		val
	val	
1999		

Regions split as the table grows

Sequential writes will go to new region

3041

3045

3050

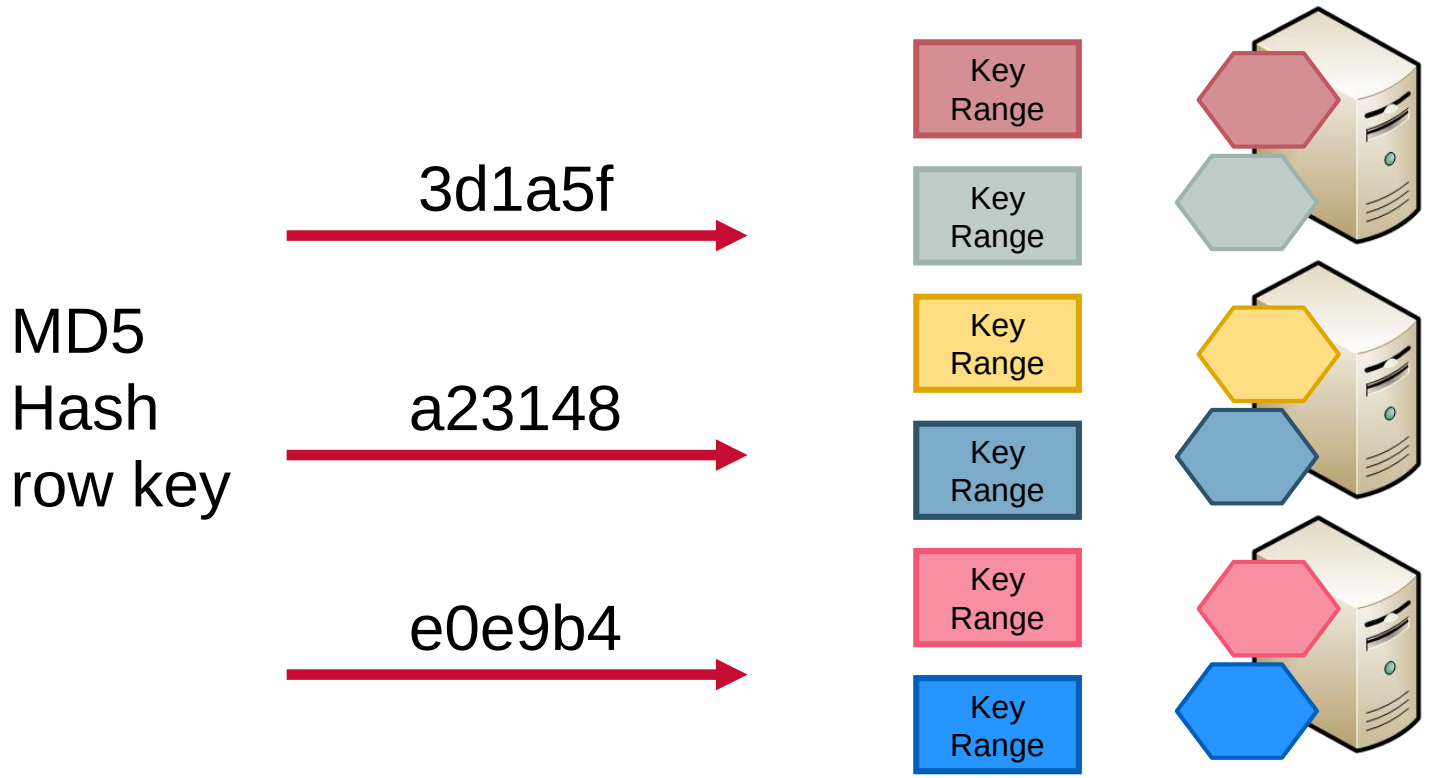
Hot-Spotting Summary

- Caused by **row keys** that are ***written*** in **sequential** order
  - Ex: row keys written in order, 0000, 0001, 0002, 0003…
- All writes go to only one server at a time
  - **Bottlenecks** **write** performance
- Results in **Inefficient splitting**
  - Regions fill to half the maximum, but never more.

Random Keys

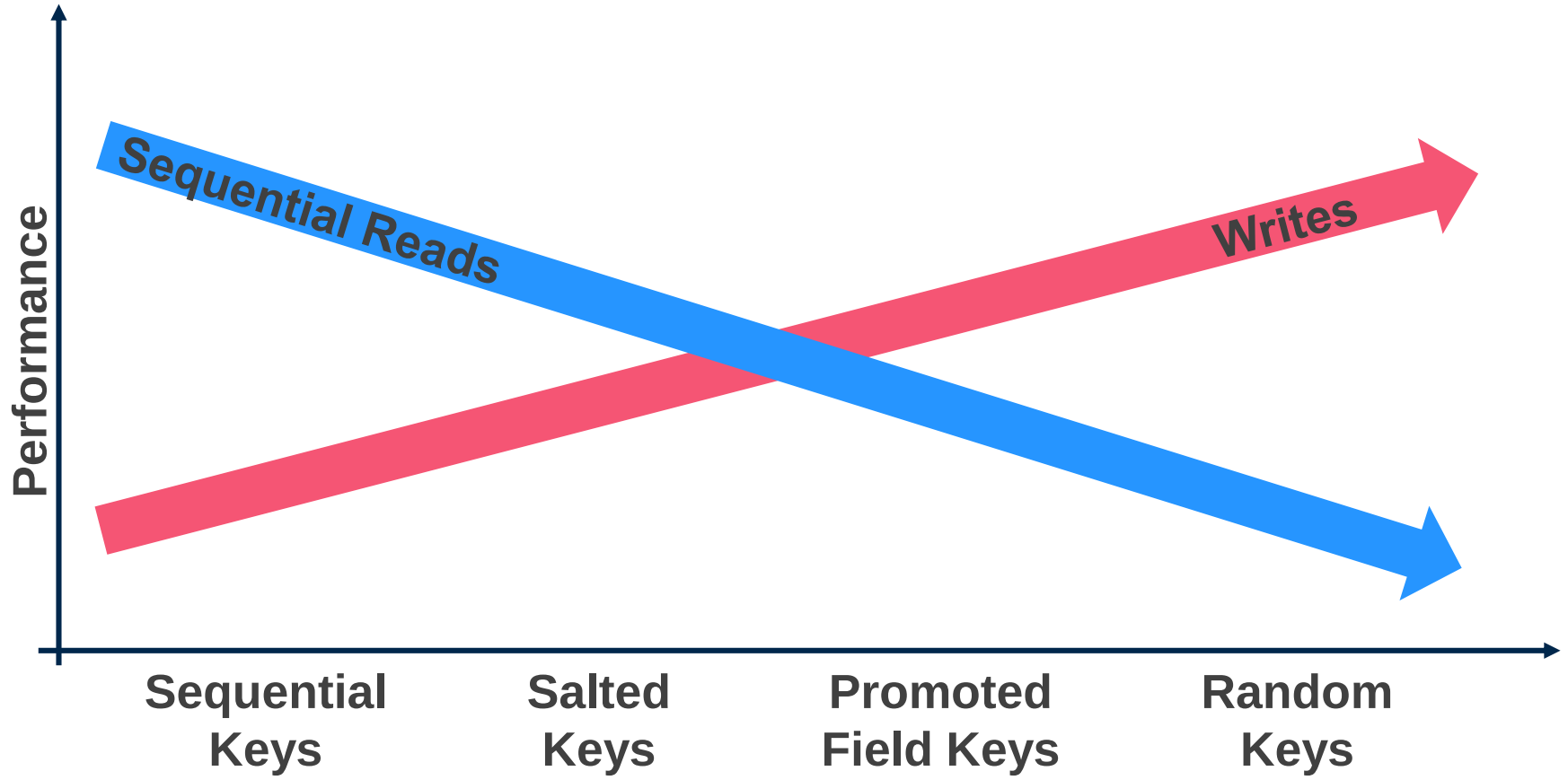
Random writes will go to different regions
If table was pre-split or **big enough** to have split

```
d = MessageDigest.getInstance("MD5");
byte[] prefix = d.digest(Bytes.toBytes(s));
```



Sequential vs. Random keys

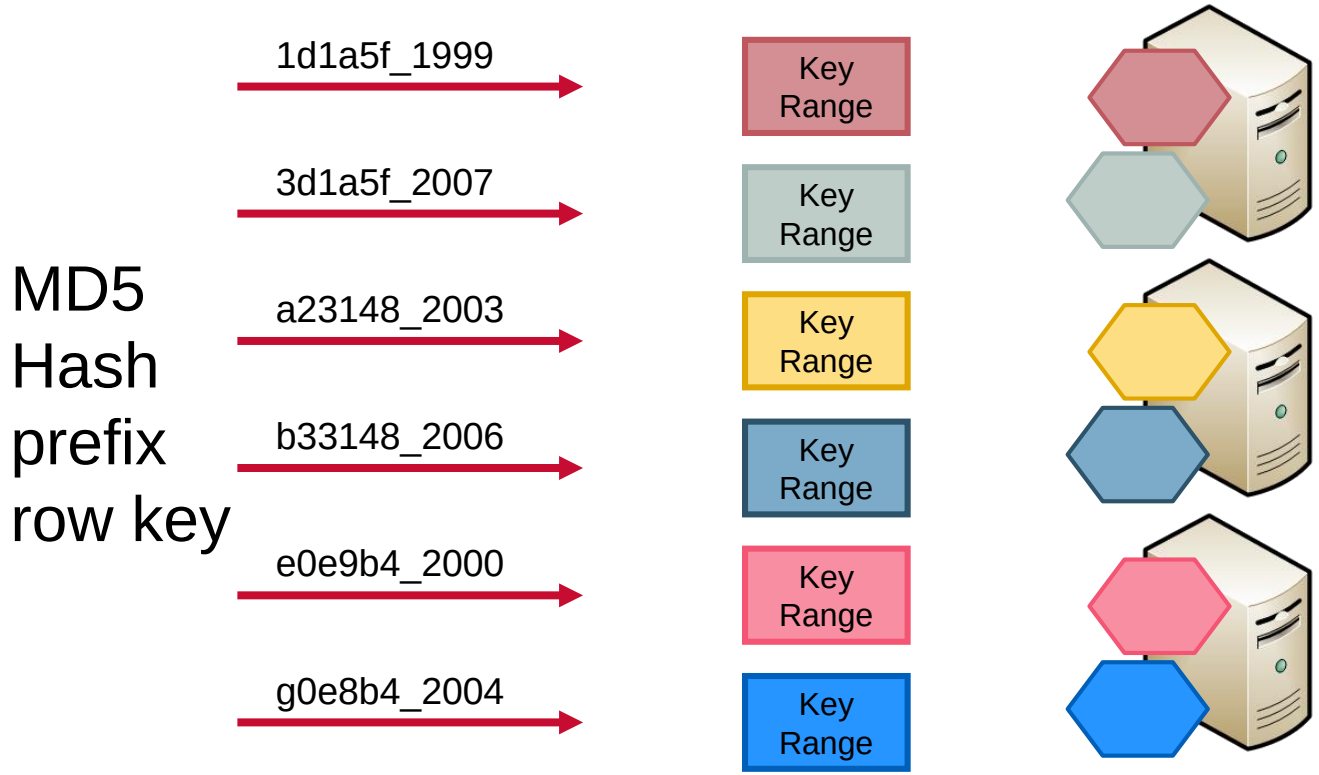
Random is better for **writing** , but **sequential** is better for **scanning** row keys



Prefix with a Hashed Field Key

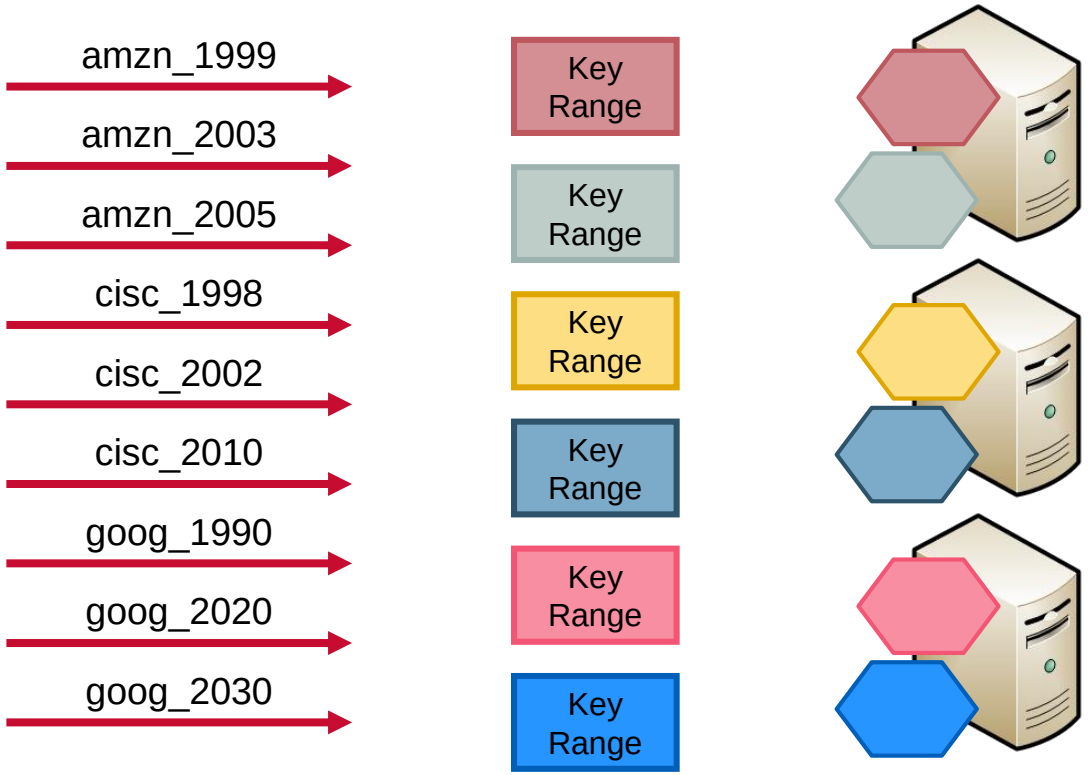
- Prefix the row key with a (shortened) hash:

```
byte[] hash = d.digest(Bytes.toBytes(fieldkey));
Bytes.putBytes(rowkey, 0, hash, 0, length);
```



Prefix, Promote a Field Key

- Prefix or promote identifying/searchable value to front of key



Consider Access Patterns for Application

- Which trade data needs fastest access (or most frequent)?
  - ***Row key ordering***
- What if you want to retrieve the stocks by symbol & date?
  - **Scan by: STOCKSYMBOL_TIMESTAMP**

SYMBOL	+	timestamp

Row Key				
AMZN_1391813876369				
AMZN_1391813876370				
GOOG_1391813876371				

- What if you usually want to retrieve the most recent?

Last In First Out Access: Use Reverse-Timestamp

- Row keys are sorted in increasing order
- For fast access to **most-recent** writes:
  - **Design composite** row key with **reverse-timestamp** that **decreases** over time.
  - **Scan by row key prefix Decreasing: [MAXTIME–TIMESTAMP]**
    - **Ex:** `Long.MAX_VALUE-date.getTime()`

SYMBOL	+	Reverse timestamp

Key				
AMZN_98618600666				
AMZN_98618600777				
GOOG_98618608888				

*Long.MAX_VALUE = 2^{63} -1

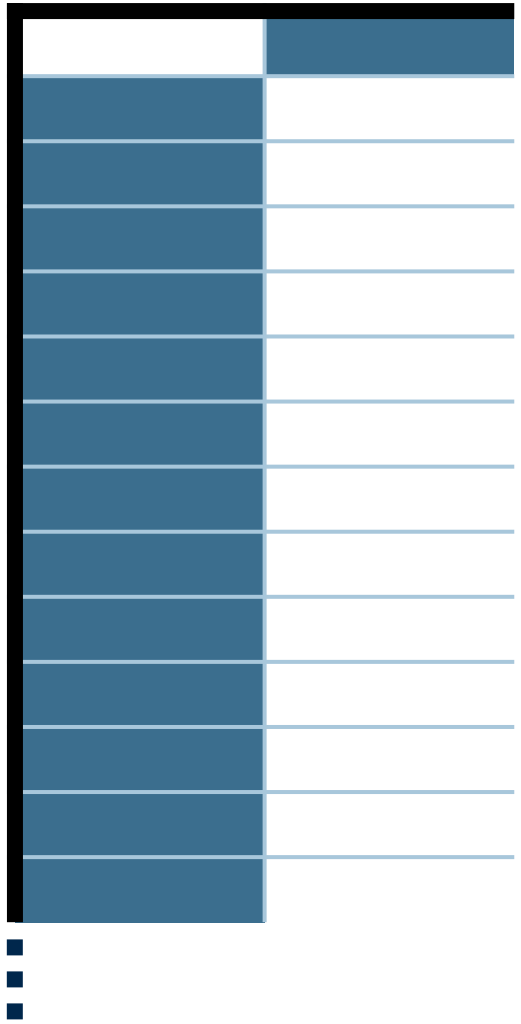
Consider Access Patterns for Application

How will data be retrieved?

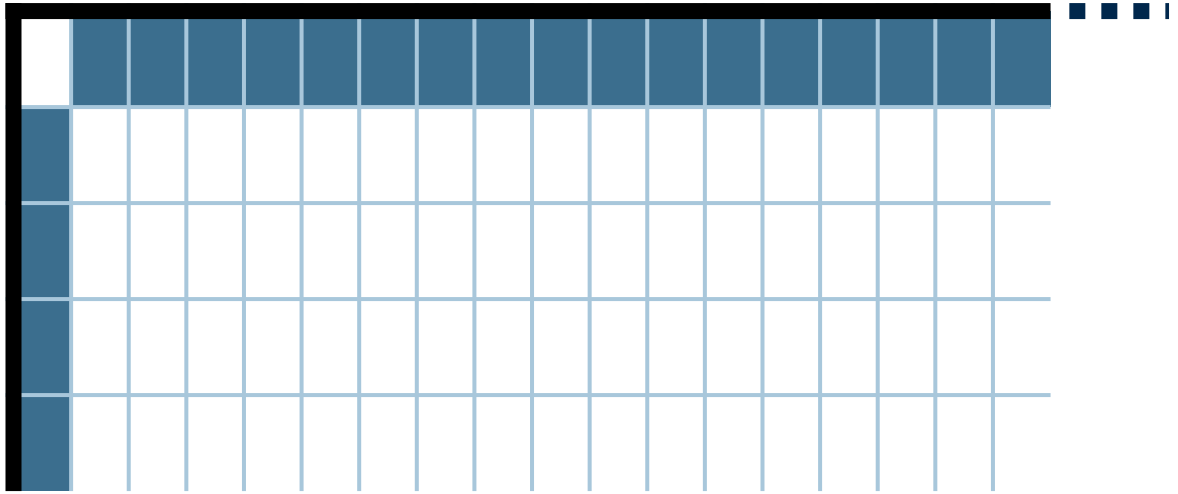
- What are the needs for atomicity of transactions?
  - ***Column design***
  - **More** Values in a **single row**
    - Works well to get or **update multiple values**

Tall or Flat Tables

Tall Narrow



Flat Wide



Consider Access Patterns for Application

- Are Price & Volume data typically accessed together, or are they unrelated?
  - ***Column family structure***

- Column Families
  - Group data that will be read & stored together

Tall Table for Stock Trades

Row key format: **SYMBOL** | **+** | **Reverse timestamp**

Ex: AMZN_98618600888

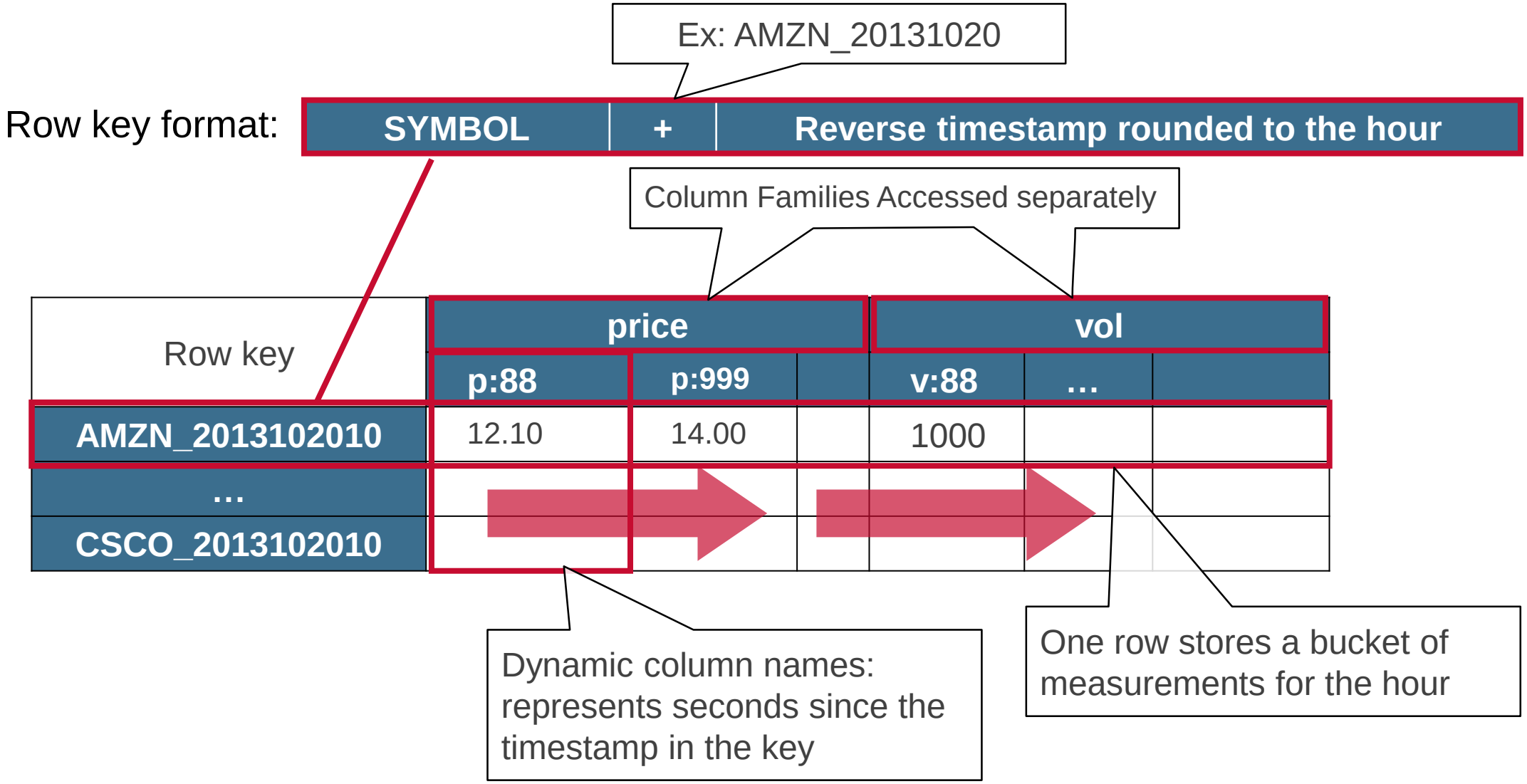
Group data that will be read & stored together

Row key	CF: CF1	
	CF1:price	CF1:vol
…	…	…
AMZN_98618600666	12.34	2000
AMZN_98618600777	12.41	50
AMZN_98618600888	12.37	10000
…	…	…
CSCO_98618600777	23.01	1000
…		

Consider Access Patterns for Application

- Are Price & Volume data typically accessed separately ?
  - ***Column family structure***
- Column Families
  - Separate data that will be **not** be read together
- Columns
  - Column names are dynamic

Medium Wide Table for Stock Trades



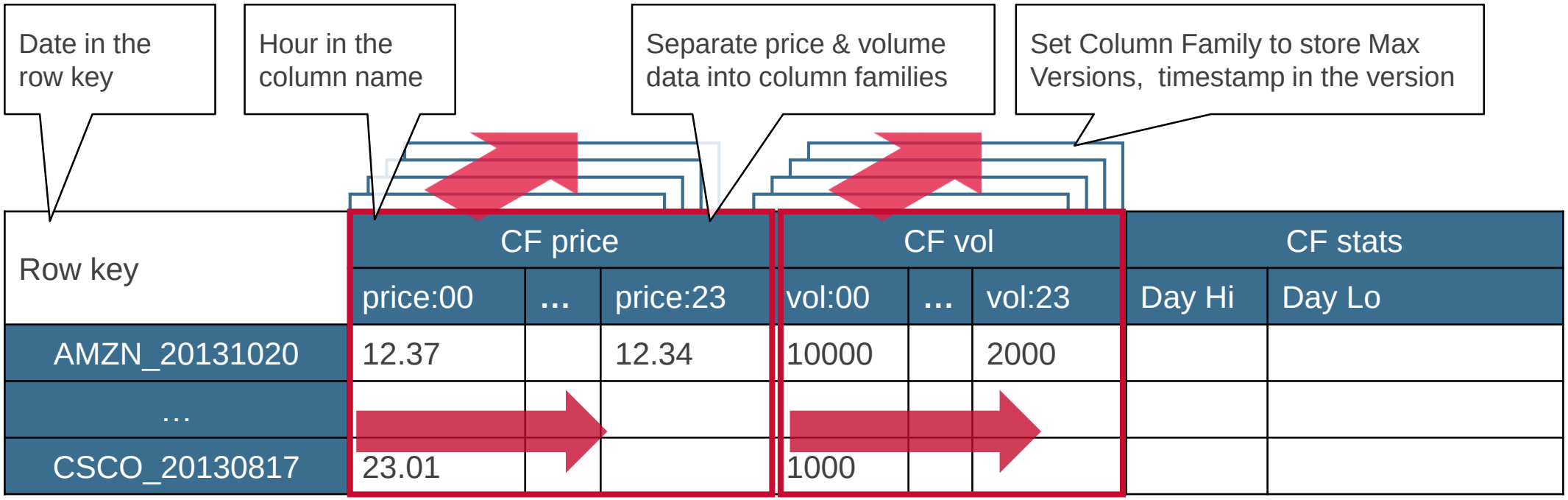
Consider Access Patterns for Application

- Column Families
  - How many Versions?
    - ***Max Versions***

Wide Table for Stock Trades

Row key format: **SYMBOL** + **date YYYYMMDD**

Ex: AMZN_20131020



Row key	CF price			CF vol			CF stats	
	price:00	...	price:23	vol:00	...	vol:23	Day Hi	Day Lo
AMZN_20131020	12.37		12.34	10000		2000		
...								
CSCO_20130817	23.01			1000				

Flat-Wide Vs. Tall-Narrow Tables

- Tall-Narrow provides better **query granularity**
  - Finer grained Row Key
  - Works well with scan

- Flat-Wide supports built-in row **atomicity**
  - More Values in a single row
    - Works well to update multiple values (row atomicity)
    - Works well to get multiple associated values

Column names can be dynamic, every row does not need to have same columns

Lesson: Schemas can be very flexible and can even change on the fly

Lesson: Have to know the queries to design in performance

Lab – Import airline data into HBase

$ tail ontime.csv

2014,1,1,31,5,**2014-01-31**,WN,N7704B,228,TUS,LAS,1946,46.00,1958,43.00,0.00,,
75.00,60.00,365.00,11.00,0.00,0.00,0.00,32.00,

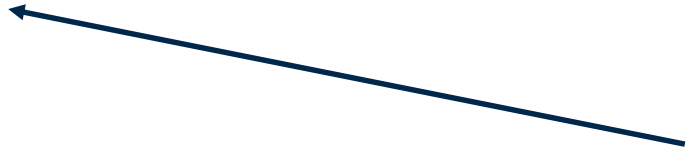
$ wc -l ontime.csv
471949 lines of flight information

Import command:

-Dimporttsv.columns=$CF:year,$CF:qtr,$CF:month,$CF:dom,$CF:dow,HBASE_ROW_KEY,
$CF:carrier,$CF:tailnum,$CF:flightnumber,$CF:origin,$CF:dest,$CF:deptime,$CF:depdelay,$CF
:arrtime,$CF:arrdelay,$CF:cncl,$CF:cnclcode,$CF:elaptime,$CF:airtime,$CF:distance,$CF:carri
erdelay,$CF:weatherdelay,$CF:nasdelay,$CF:securitydelay,$CF:aircraftdelay

Lab – Import airline data into HBase

2014,1,1,31,5,**2014-01-31**,WN,N7704B,228,TUS,LAS,1946,46.00,1958,43.00,0.00,,75.00,60.00,365.00,11.00,0.00,0.00,0.00,32.00,



HBASE_ROW_KEY

Table: airline

Row-key Date	cf1										
	year	Qtr	month	dom	dow	carrier	tailnum	Flight number	origin	dest	…
2014-01-31	2014	1	1	31	5	WN	N7704B	228	TUS	LAX	…

Why does the table only have 31 rows ??

$ hbase shell
hbase(main):003:0> count '/tables/airline'
31 row(s) in 0.0560 seconds

hbase(main):003:0> scan '/tables/airline', {LIMIT => 1}

ROW	COLUMN+CELL
2014-01-01	column=cf1:aircraftdelay, timestamp=1424960635661, value=0.00
2014-01-01	column=cf1:airtime, timestamp=1424960635661, value=201.00
2014-01-01	column=cf1:arrdelay, timestamp=1424960635661, value=26.00
2014-01-01	column=cf1:arrtime, timestamp=1424960635661, value=1311
2014-01-01	column=cf1:carrier, timestamp=1424960635661, value=WN
2014-01-01	column=cf1:carrierdelay, timestamp=1424960635661, value=0.00
2014-01-01	column=cf1:cncl, timestamp=1424960635661, value=0.00
2014-01-01	column=cf1:cnclcode, timestamp=1424960635661, value=
2014-01-01	column=cf1:depdelay, timestamp=1424960635661, value=1.00
2014-01-01	column=cf1:deptime, timestamp=1424960635661, value=0931
2014-01-01	column=cf1:dest, timestamp=1424960635661, value=MCO
2014-01-01	column=cf1:distance, timestamp=1424960635661, value=1142.00
2014-01-01	column=cf1:dom, timestamp=1424960635661, value=1
2014-01-01	column=cf1:dow, timestamp=1424960635661, value=3
2014-01-01	column=cf1:dummy, timestamp=1424960635661, value=
2014-01-01	column=cf1:elaptime, timestamp=1424960635661, value=195.00
2014-01-01	column=cf1:flightnumber, timestamp=1424960635661, value=1147
2014-01-01	column=cf1:month, timestamp=1424960635661, value=1
2014-01-01	column=cf1:nasdelay, timestamp=1424960635661, value=26.00
2014-01-01	column=cf1:origin, timestamp=1424960635661, value=MHT
2014-01-01	column=cf1:qtr, timestamp=1424960635661, value=1
2014-01-01	column=cf1:securitydelay, timestamp=1424960635661, value=0.00
2014-01-01	column=cf1:tailnum, timestamp=1424960635661, value=N264LV
2014-01-01	column=cf1:weatherdelay, timestamp=1424960635661,
2014-01-01	value=0.00
	column=cf1:year, timestamp=1424960635661, value=2014

1 row

Lab – Import airline data into HBase

mapping to Row Key and Columns:

timing:year=2014,
timing:qtr=1,
timing:month=1,
timing:dom=31,
timing:dow=5,
HBASE_ROW_KEY_3=2014-01-31, // date
HBASE_ROW_KEY_1=WN, // carrier
info:tailnum=N7704B,
HBASE_ROW_KEY_2=228, // flight number
HBASE_ROW_KEY_4=TUS, // orig
HBASE_ROW_KEY_5=LAS, // dest
timing:deptime=1946,
delay:depdelay=46.00,

timing:arrtime=1958,
delay:arrdelay=43.00,
info:cncl=0.00,
info:cnclcode="",
stats:elaptime=75.00,
stats:airtime=60.00,
stats:distance=365.00,
delay:carrierdelay=11.00,
delay:weatherdelay=0.00,
delay:nasdelay=0.00,
delay:securitydelay=0.00,
delay:aircraftdelay=32.00,

Scans are limited

```
$ hbase shell

scan '/tables/airline'
scan  '/tables/airline',  { STARTROW => '2014'}

scan  '/tables/airline',  { STARTROW => '2014-01-20', STOPROW => '2014-01-21'}
```

1 row

Lab – Import airline data into HBase

Import mapping to Row Key and Columns:

Row-key Carrier-Flightnumber-Date-Origin-destination	delay			info			stats		timing	
	Air Craft delay	Arr delay	Carrier delay	cncl	Cncl code	tailnum	distance	elaptime	arrtime	Dep time
AA-1-2014-01-01-JFK-LAX		13		0		N7704	2475	385.00	359	…

Do you see the advantages to this row key, CFs ?

$ hbase shell
hbase(main):003:0> count '/tables/airline'
471,949 row(s)
hbase(main):003:0>scan '/tables/airline',
{STARTROW => 'WN-228-2014-01-31-TUS-LAS',
LIMIT => 1}

ROW	COLUMN+CELL
WN-228-2014-01-31-TUS-LAS	column=delay:aircraftdelay, timestamp=1425775096289, value=32.00
WN-228-2014-01-31-TUS-LAS	column=delay:arrdelay, timestamp=1425775096289, value=43.00
WN-228-2014-01-31-TUS-LAS	column=delay:carrierdelay, timestamp=1425775096289, value=11.00
WN-228-2014-01-31-TUS-LAS	column=delay:depdelay, timestamp=1425775096289, value=46.00
WN-228-2014-01-31-TUS-LAS	column=delay:nasdelay, timestamp=1425775096289, value=0.00
WN-228-2014-01-31-TUS-LAS	column=delay:securitydelay, timestamp=1425775096289, value=0.00
WN-228-2014-01-31-TUS-LAS	column=delay:weatherdelay, timestamp=1425775096289, value=0.00
WN-228-2014-01-31-TUS-LAS	column=info:cncl, timestamp=1425775096289, value=0.00
WN-228-2014-01-31-TUS-LAS	column=info:cnclcode, timestamp=1425775096289, value=
WN-228-2014-01-31-TUS-LAS	column=info:dummy, timestamp=1425775096289, value=
WN-228-2014-01-31-TUS-LAS	column=info:tailnum, timestamp=1425775096289, value=N7704B
WN-228-2014-01-31-TUS-LAS	column=stats:airtime, timestamp=1425775096289, value=60.00
WN-228-2014-01-31-TUS-LAS	column=stats:distance, timestamp=1425775096289, value=365.00
WN-228-2014-01-31-TUS-LAS	column=stats:elaptime, timestamp=1425775096289, value=75.00
WN-228-2014-01-31-TUS-LAS	column=timing:arrtime, timestamp=1425775096289, value=1958
WN-228-2014-01-31-TUS-LAS	column=timing:deptime, timestamp=1425775096289, value=1946
WN-228-2014-01-31-TUS-LAS	column=timing:dom, timestamp=1425775096289, value=31
WN-228-2014-01-31-TUS-LAS	column=timing:dow, timestamp=1425775096289, value=5
WN-228-2014-01-31-TUS-LAS	column=timing:month, timestamp=1425775096289, value=1
WN-228-2014-01-31-TUS-LAS	column=timing:qtr, timestamp=1425775096289, value=1
WN-228-2014-01-31-TUS-LAS	column=timing:year, timestamp=1425775096289, value=2014

1 row

Comparing Relational vs HBase
Schema design And more Complex Schema design

Better Scans

$ hbase shell

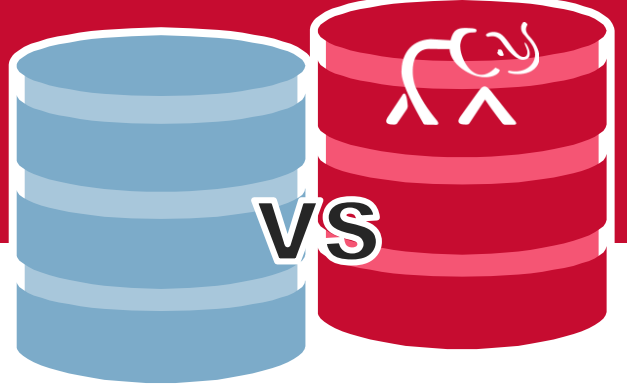
scan '/tables/airline', { STARTROW => 'AA-1-2014-01-01-JFK-LAX', STOPROW => 'AA-10‘}

scan '/tables/airline', {COLUMNS=>['delay'], STARTROW => ‘DL'}

scan '/tables/airline', FILTER=>"ValueFilter(=,'binary:239.00')"

1 row

Relational vs. HBase Schemas

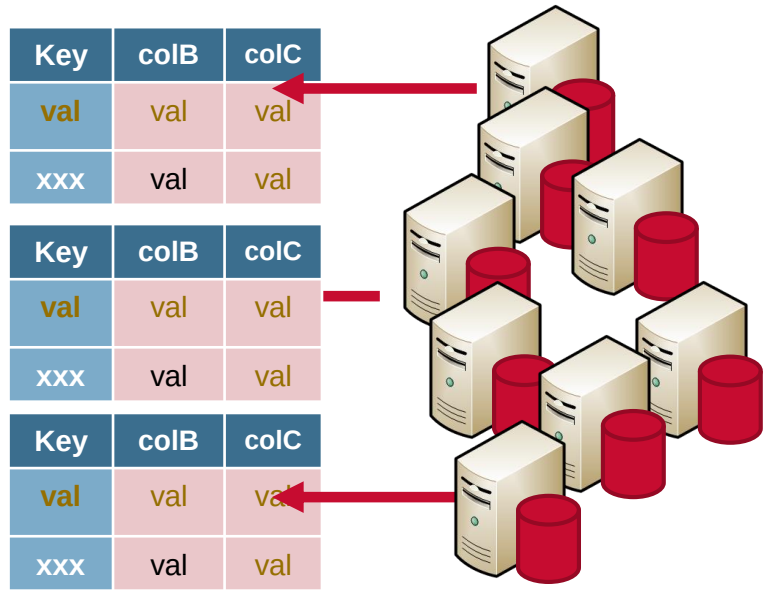


- Relational design
  - Data centric, focus on entities & relations
  - Query joins
    - New views of data from different tables easily created

      Does not scale across cluster

- HBase is designed for clustering:
  - Distributed data is stored and accessed together
  - Query centric, focus on how the data is read
  - Design for the questions

HBase



Data that is accessed together is stored together

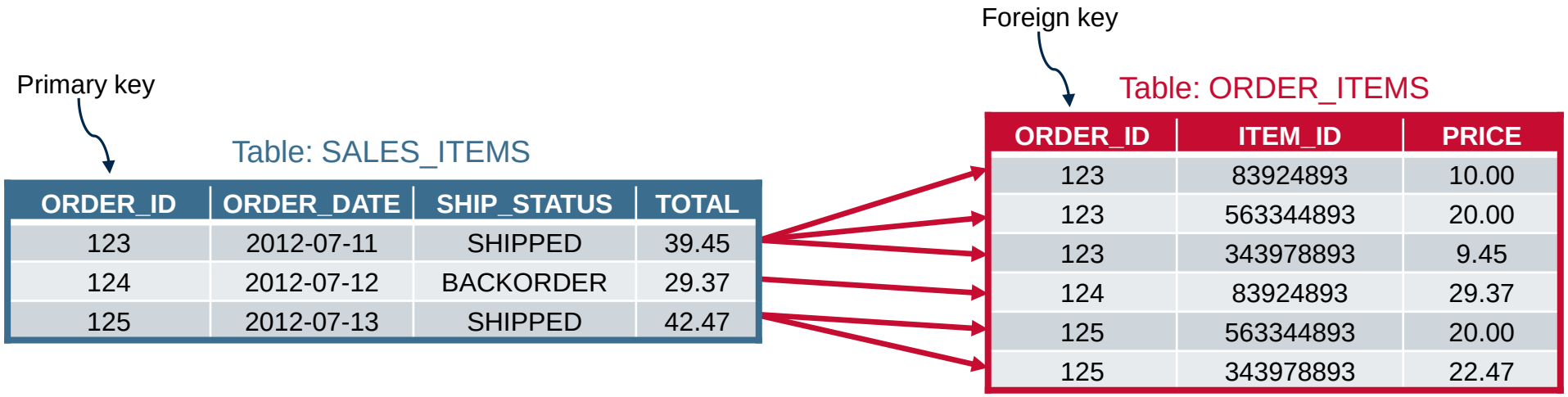
Normalization

Goal of Normalization

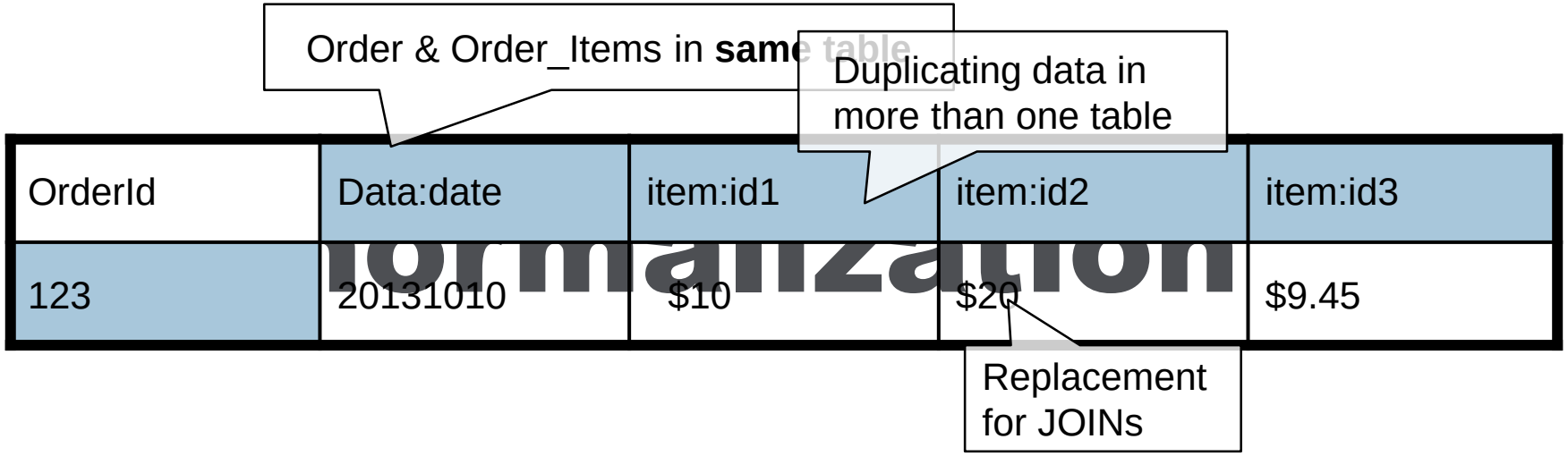
- Eliminate redundant data
- Put repeating information in its own table

Normalized database

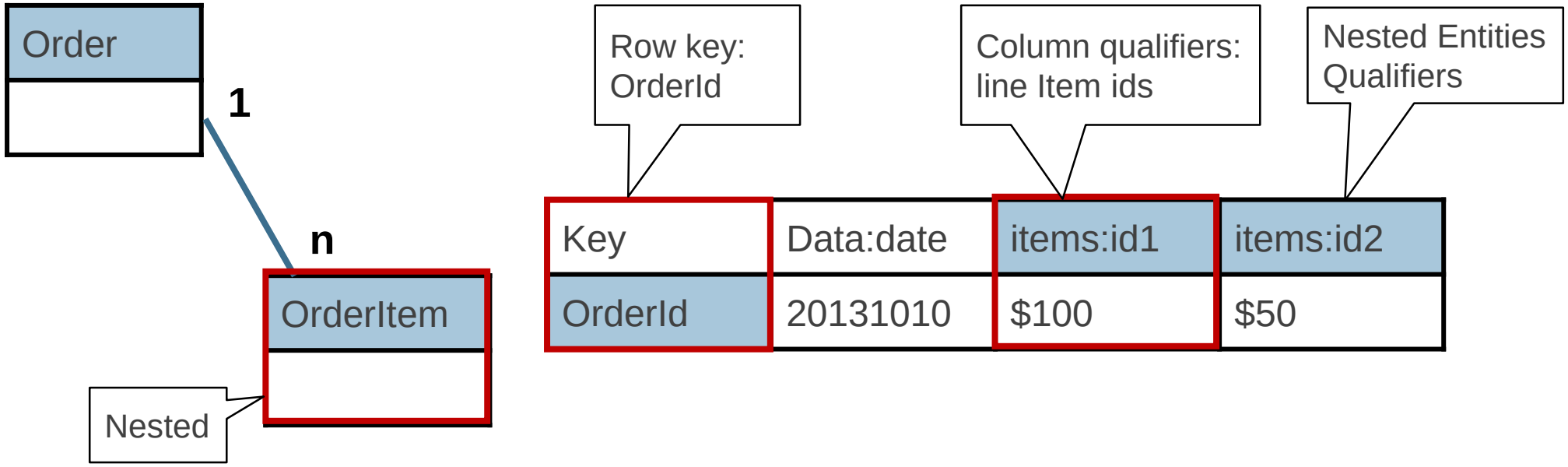
- Causes joins
  - Data has to be retrieved from more tables
  - Queries can take more time



Denormalization



Parent-Child Relationship –Nested Entity



A one-to-many relationship can be modeled as a single row

Embedded, Nested Entity

Reads are faster across a cluster

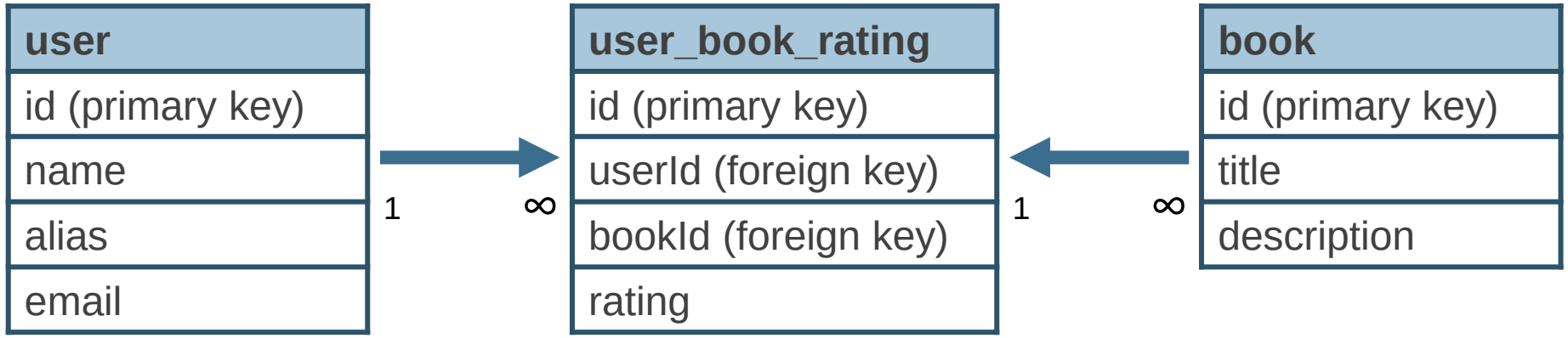
retrieve data about entity and related entities in **one read**

Many to Many Relationship - RDBMS

Queries

- Get name for user x
- Get tiltle for book x
- Get books and corresponding ratings for userId x
- Get all userids and corresponding ratings for book y

Online book store



Many to Many Relationship - HBase

Queries

- Get books and corresponding ratings for userId x
- Get all userids and corresponding ratings for book y

User table Column family for book ratings by userid for bookids

Key	**data**:fname	…	**rating:bookid1**	rating:bookid2
userid1			5	4

Book table Column family for ratings for bookid by userid

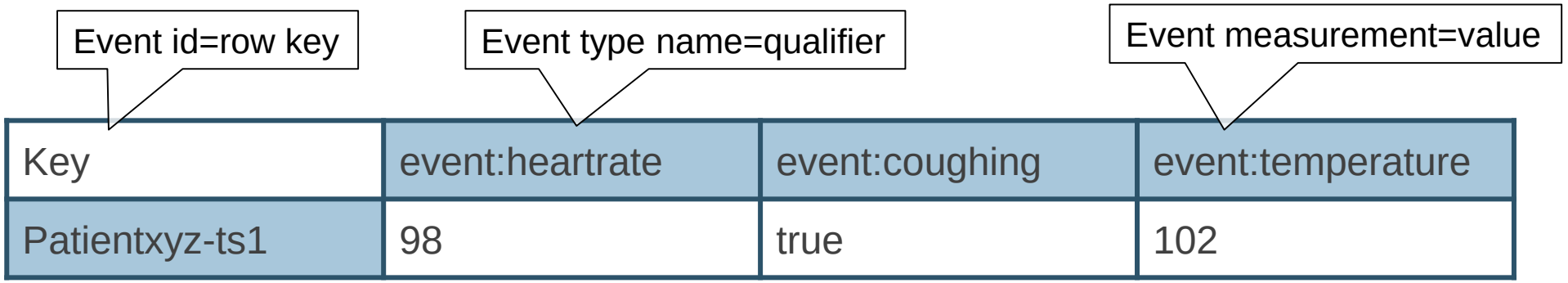
Key	**data:title**	…	**rating:userid1**	rating:userid2
bookid1			5	4

Generic Data, Event Data, Entity-Attribute-Value

- **Generic** table: Entity, Attributes, Values
  - Event Id, Event attributes, Values
  - Object-property-value , name-value pairs , **schema-less**

```
patientXYZ-ts1, Temperature , "102"
patientXYZ-ts1, Coughing, "True"
patientXYZ-ts1, Heart Rate, "98"
```

- Advantage of HBase
  - Define columns on the fly, put attribute name in column qualifier
  - Group data by column families

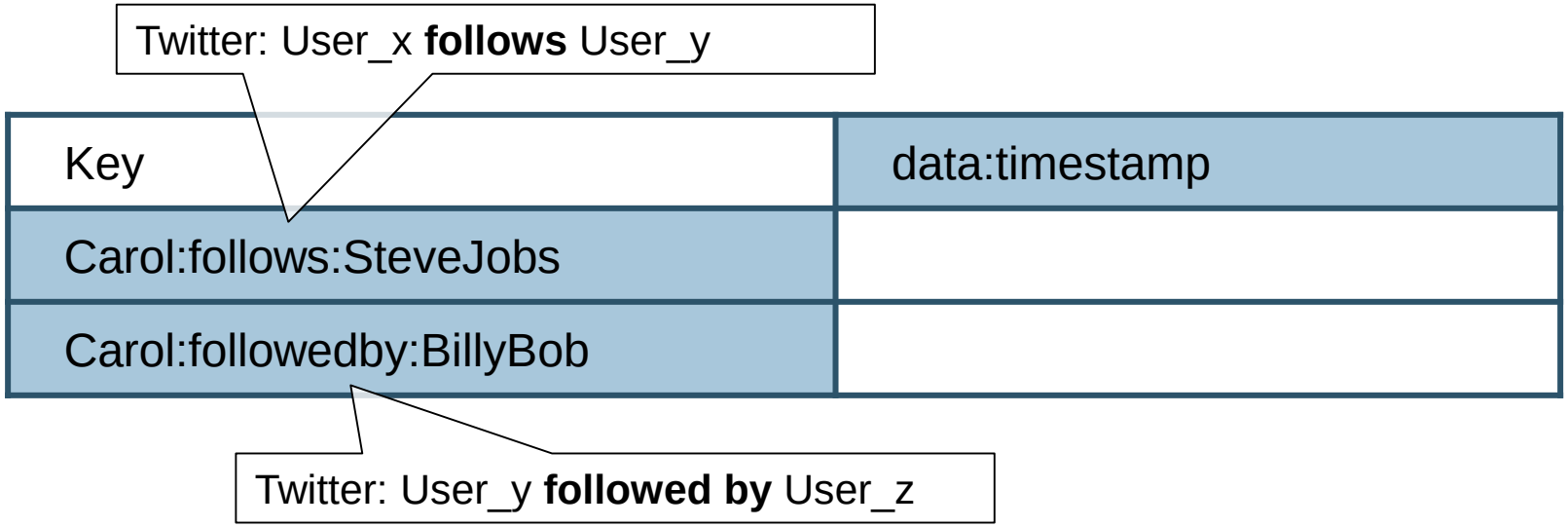


Key	event:heartrate	event:coughing	event:temperature
Patientxyz-ts1	98	true	102

Self Join Relationship - HBase

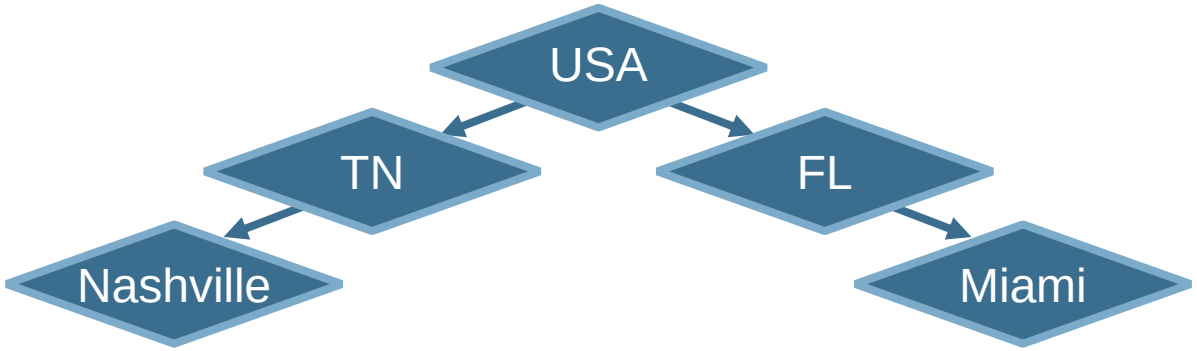
Queries

- Get all users who Carol follows
- Get all users following Carol



Tree, Graph Data

- Row=node
- Row key=node id
- Parents, children in columns
  - Col Qualifier=edge id

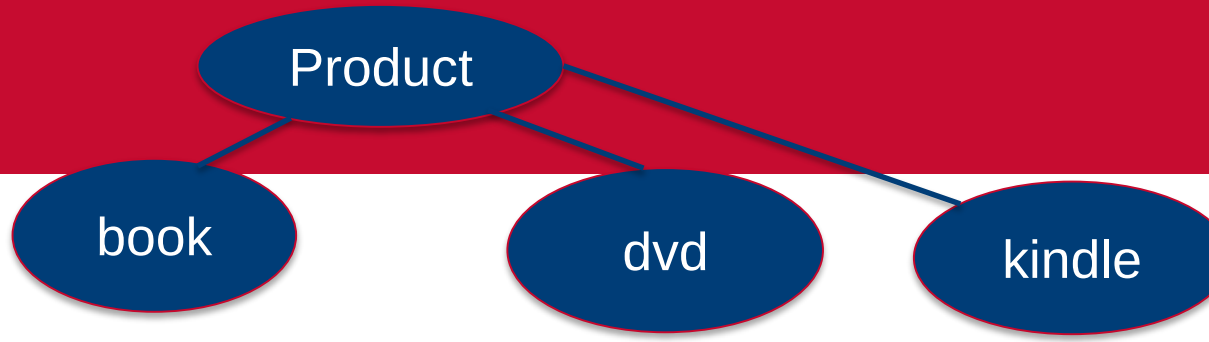


parent

children

Key	P:USA	P:TN	P:FL	C:TN	C:FL	C:Nashvl	C:Miami
USA				state	state		
TN	country					city	
FL	country						city
Nashville		state					
Miami			state				

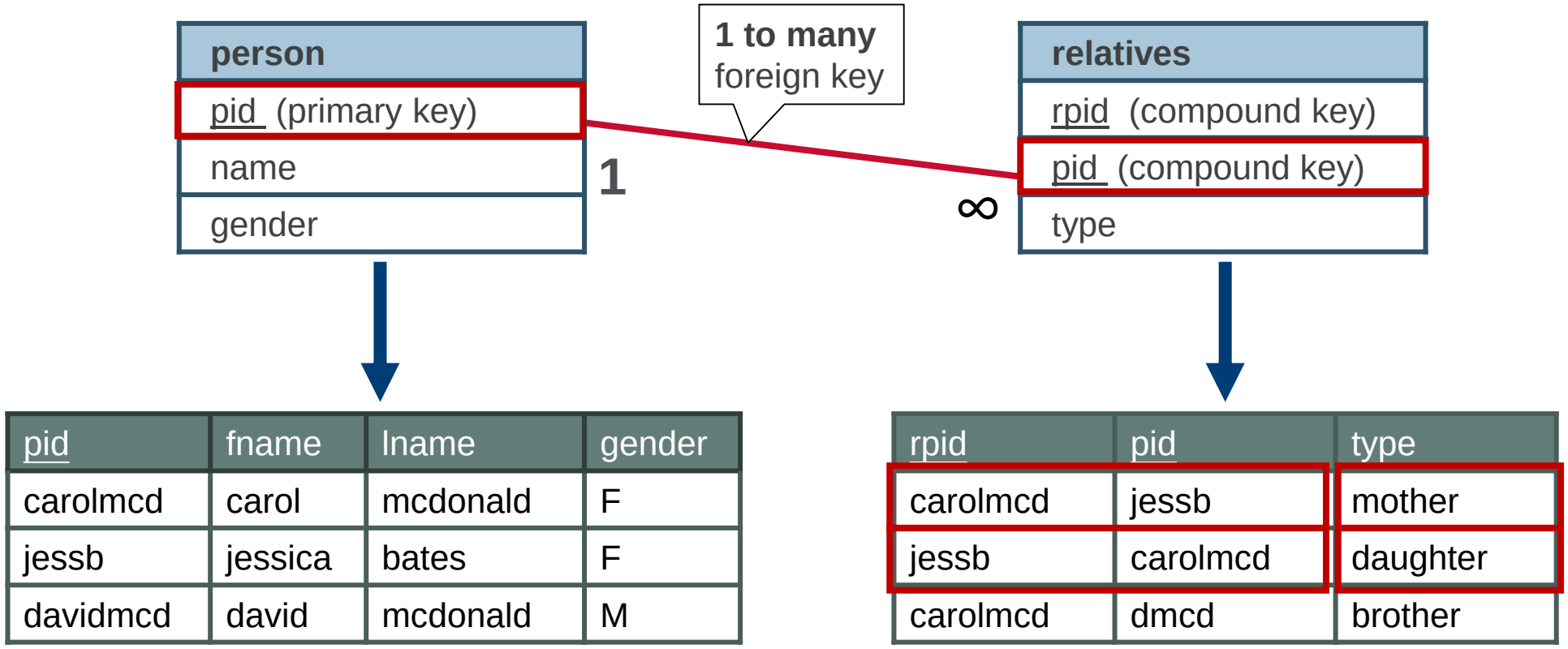
Inheritance Mapping



- Online Store Example Product table
  - Put **sub class type abbreviation** in **key prefix** for searching
  - Columns do not all have to be the same for different types

Key	price	title	details	model
Bok+id1	10	HBase		
Dvd+Id2	15	stones		
Kin+Id3	100			fire

Use Case 1 – Person's Relatives



pid	fname	lname	gender
carolmcd	carol	mcdonald	F
jessb	jessica	bates	F
davidmcd	david	mcdonald	M

rpid	pid	type
carolmcd	jessb	mother
jessb	carolmcd	daughter
carolmcd	dmcd	brother

get all the relatives for a userid ?

HBase Modeling Concepts

1. Identify Entities
2. Identify Queries
3. Identifying Attributes
4. Non-identifying attributes
5. Relationships
6. Secondary index (foreign keys)

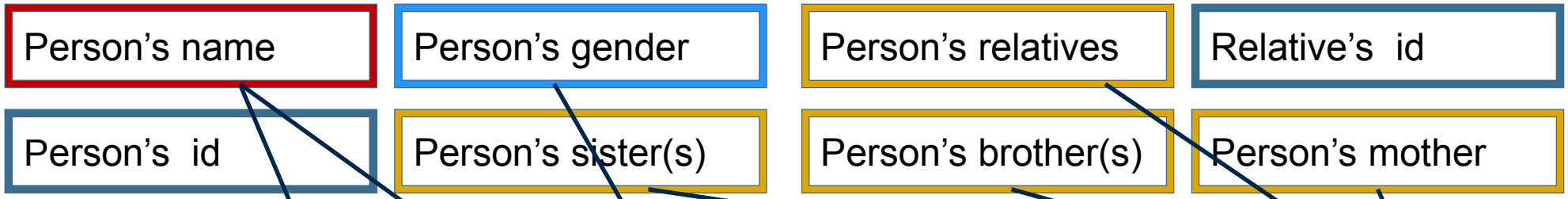
HBase Modeling Concepts

1. Identify Entities
2. Identify Queries
   - What information accessed together in one get
   - What information needs scanning
3. Identifying Attributes
4. Non-identifying attributes
5. Relationships
6. Secondary index (foreign keys)

Use Case 1 - Queries

2 Identify queries

get all the relatives for a userid ?



pid	fname	lname	gender
carolmcd	carol	mcdonald	F
jessb	jessica	bates	F
davidmcd	david	mcdonald	M

rpid	pid	type
carolmcd	jessb	mother
jessb	carolmcd	daughter
carolmcd	dmcd	brother

HBase Modeling Concepts

1. Identify Entities
2. Identify Queries
3. Identifying Attributes
4. Non-identifying attributes
5. Relationships
6. Secondary index (foreign keys)

Use Case 1 – Identifying & Non-Identifying Attributes

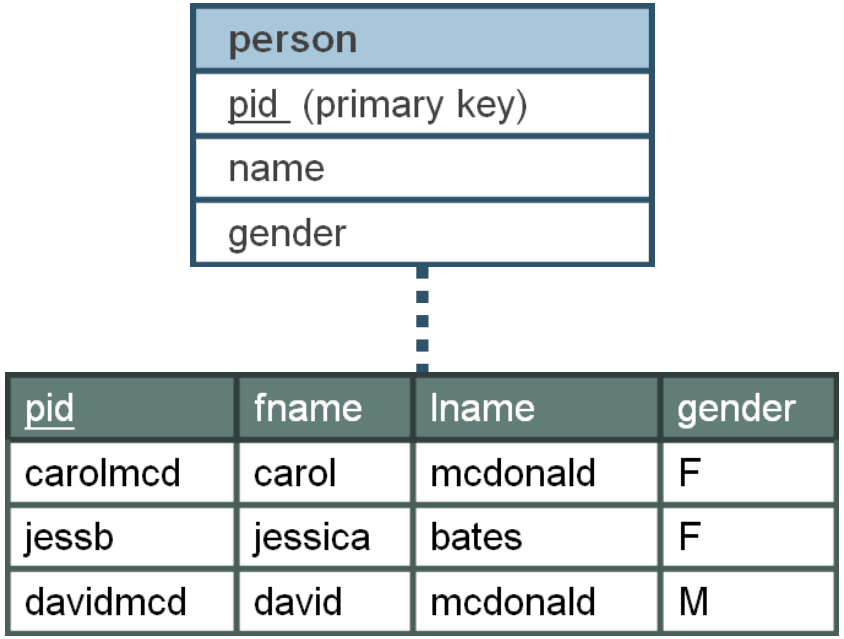
3

Identifying attributes used in queries

Identify which attributes can be used to identify unique instances of the entity (entity = Person):

A. pid

B. pid + lname + fname



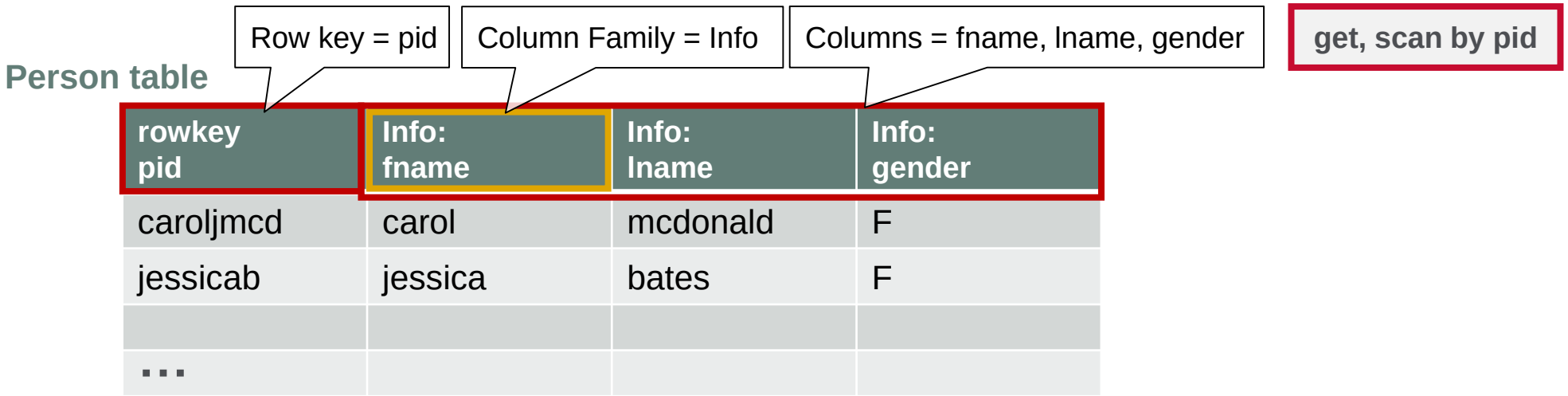
Identifying & Non Identifying Attributes – Option A

3 Identifying attributes used in queries

Identify which attributes can be used to identify unique instances of the entity (entity = Person):

A. pid

4 Entity: Person
Identifying attribute**: pid**
Non-identifying attributes: **fname, lname, gender**

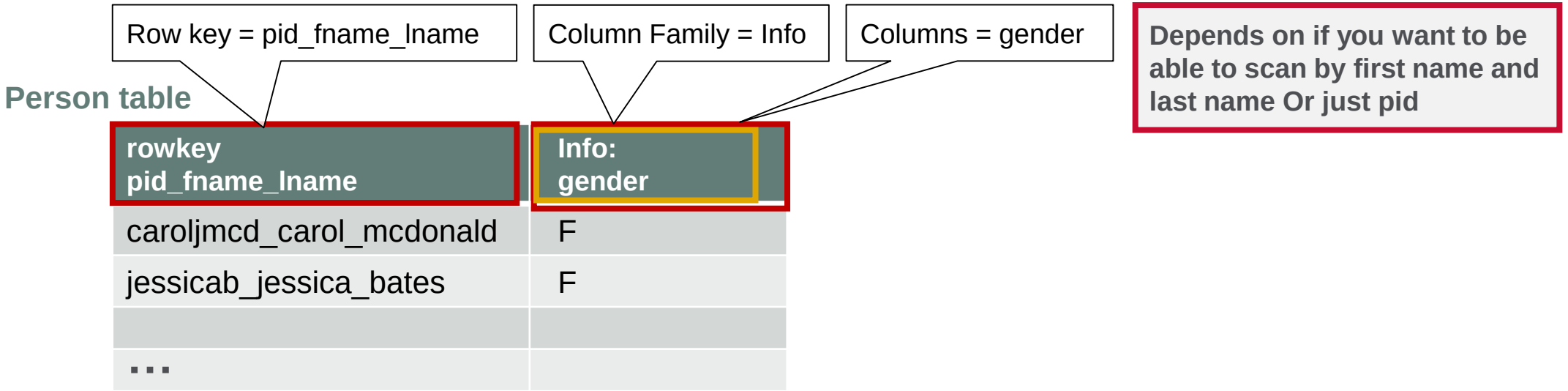


Identifying & Non Identifying Attributes – Option B

3 Identifying attributes used in queries
Identify which attributes can be used to identify unique instances of the entity (entity = Person):

B. pid + lname + fname

4 Entity: Person
Identifying attribute: **pid, fname, lname**
Non-identifying attributes: **gender**



HBase Modeling Concepts

1. Identify Entities
2. Identify Queries
3. Identifying Attributes
4. Non-identifying attributes
5. Relationships
   - One to many -> nested or embedded entities in column family
6. Secondary index (foreign keys)

Use Case 1 - Relationships

5 One to many → nested or embedded entities in column family

How to model?

Parent identifying attribute(s) = row key

Child identifying attributes = column qualifiers

Use Case 1 – Relationships (2)

5 **What are the child identifying attributes?**
(Click the column heading(s) from the table on the right.)

relatives
rpid (compound key)
pid (compound key)
type

rpid	pid	type
carolmcd	jessb	mother
jessb	carolmcd	daughter
carolmcd	dmcd	brother

Based on which attribute you use as column qualifier, you have different options.

Use Case 1 – Relationships (3)

5

Option A	Option B
Child identifying attribute = type	Child identifying attribute = rpid

Relationships #A

5

Option A

Child identifying attribute = type

Info column family

relation column family

Requires keeping a counter for number of sisters, brothers, uncles…

Person table

rowkey	info: fname	info: lname	info: gender	relation: brother1	relation: daughter1	relation: daughter2	relation: mother	relation: sister1	…
pid									
caroljmcd	carol	mcdonald	F	davidmcd	jessicab	sarahb			
jessicab	jessica	bates	F				caroljmcd	sarahb	
sarahb							caroljmcd	jessicab	
davidmcd	david							carol	

Relationships #2

5

Option B

Child identifying attribute = rpid

Info column family

relation column family

Dynamic column name

Person table

rowkey	info: fname	info: lname	info: gender	…	relation: jessicab	relation: sarahb	relation: caroljmcd	relation: davidmcd	…
pid									
caroljmcd	carol	mcdonald	F		daughter	daughter		brother	
jessicab	jessica	bates	F			sister	mother	uncle	
sarahb	sarah		F		sister		mother	uncle	
davidmcd	david		M		uncle	uncle	brother		

HBase Modeling Concepts

1. Identify Entities
2. Identify Queries
3. Identifying Attributes
4. Non-identifying attributes
5. Relationships
6. Secondary index (foreign keys)
   - Put identifying data in lookup table

Use Case 1 – Secondary index

6

Secondary index (foreign keys)

- Put identifying data in lookup table

Person table

rowkey	info: fname	Info: lname	Info: gender
caroljmcd	carol	mcdonald	F
jessicab	jessica	bates	F
…			

Relatives lookup table

rowkey	Info:ts
caroljmcd_mother_jessicab	125666
caroljmcd_mother_sarahb	125675
caroljmcd_brother_davidmcd	
jessicab_daughter_caroljmcd	
jessicab_sister_sarahb	

Requires querying 2 tables
Good solution if relationships (relatives) need frequent updating

Summary – Use Case 1

1. Entity = Person
2. Queries – get person information

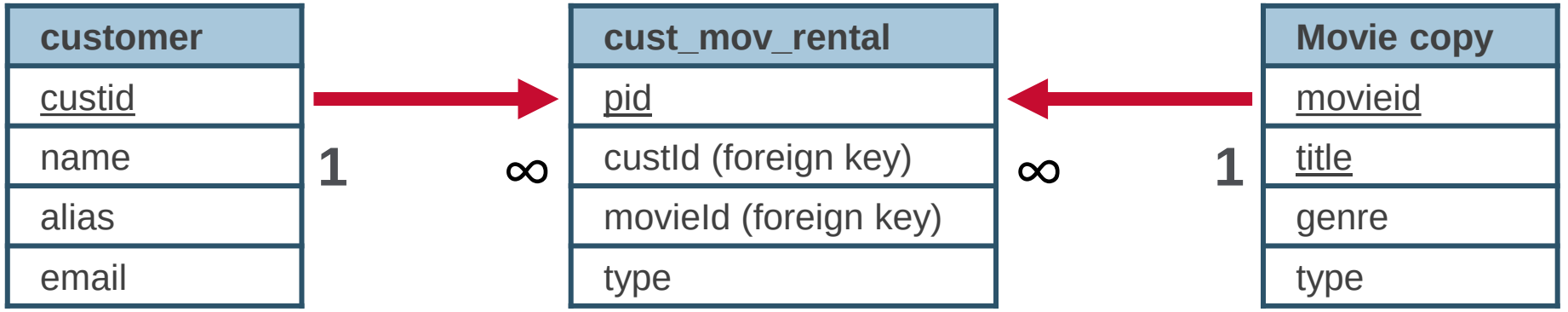
   Get all relatives for person
   Get all sisters for person, etc.

3. Identifying attributes → row key
   pid, lname, fname
4. Non-identifying attributes: gender

5. Relationships:
   1. Use type → need counter
   2. Use rpid → dynamic

6. Secondary index
   3. Use foreign key → lookup table

Use Case 2 - Customers Movies Rental

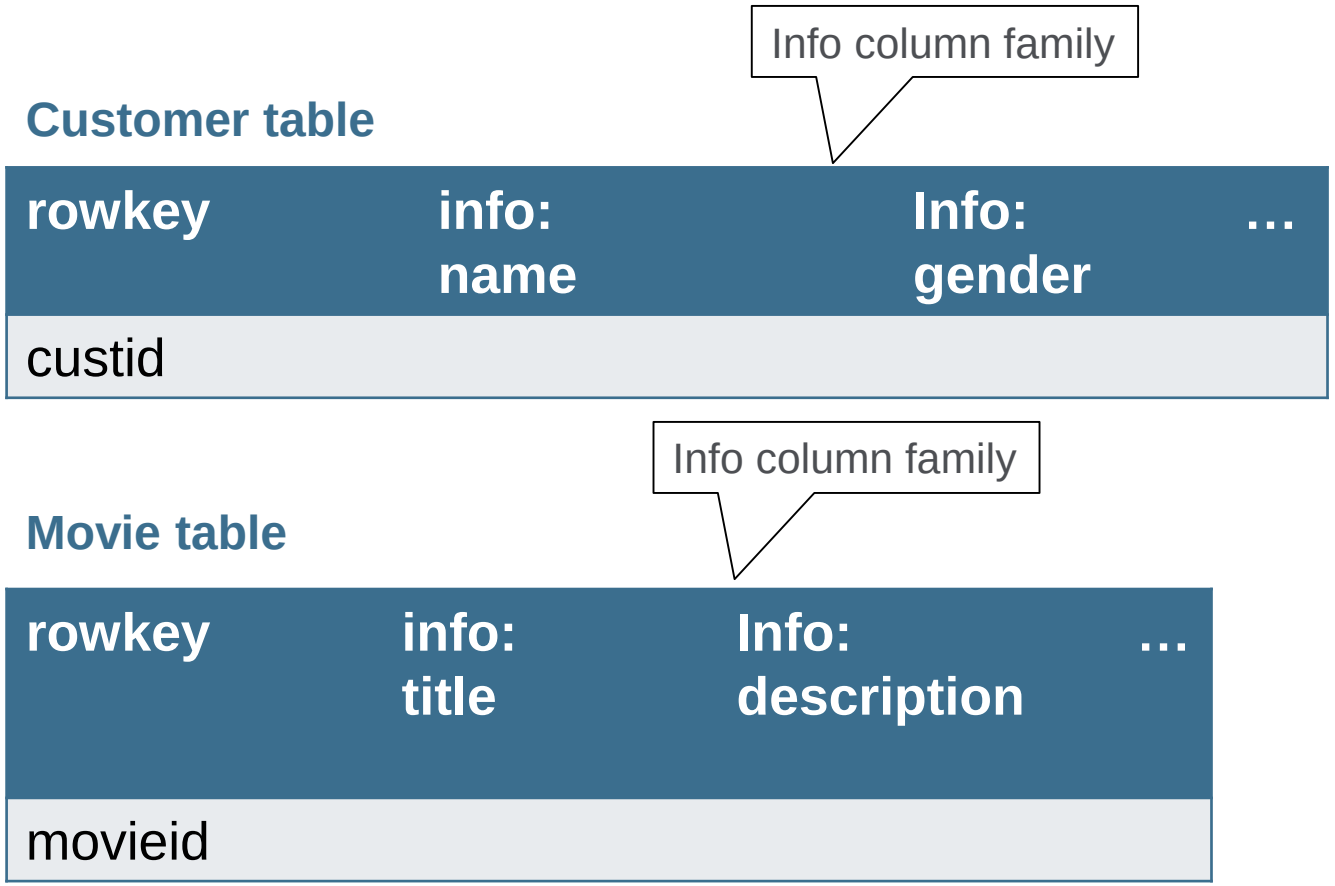


pid	Name	email
carolj	carol	cm@y
jessb	jessica	
dmcd	david	

pid	rid	type
carolj	TS1	DVD
jessb	BC1	online

movieid	title	genre
TS1	Toy story	drama
BC1	Big Chill	drama

Identifying, Non Identifying Attributes #1



HBase Modeling

- Identifying Attributes used in queries-> row key
- Non identifying attributes -> columns
- **Put identifying data for relationships:**
  - **in a column family column qualifier**
  - **Or put it in a lookup table**
- **Relationships:**
  - **Many to many -> 2 tables:**
    - **?**

Relationships #1

Info column family

rental column family

Customer table

rowkey	info: name	Info: gender	…	rental: movieid1	rental: movieid2	…
custid1				dvd		
custid2					online	

Info column family

rental column family

Movie table

rowkey	info: title	Info: description	…	rental: custid1	rental: custid2	…
movieid1				dvd		
movieid2					online	

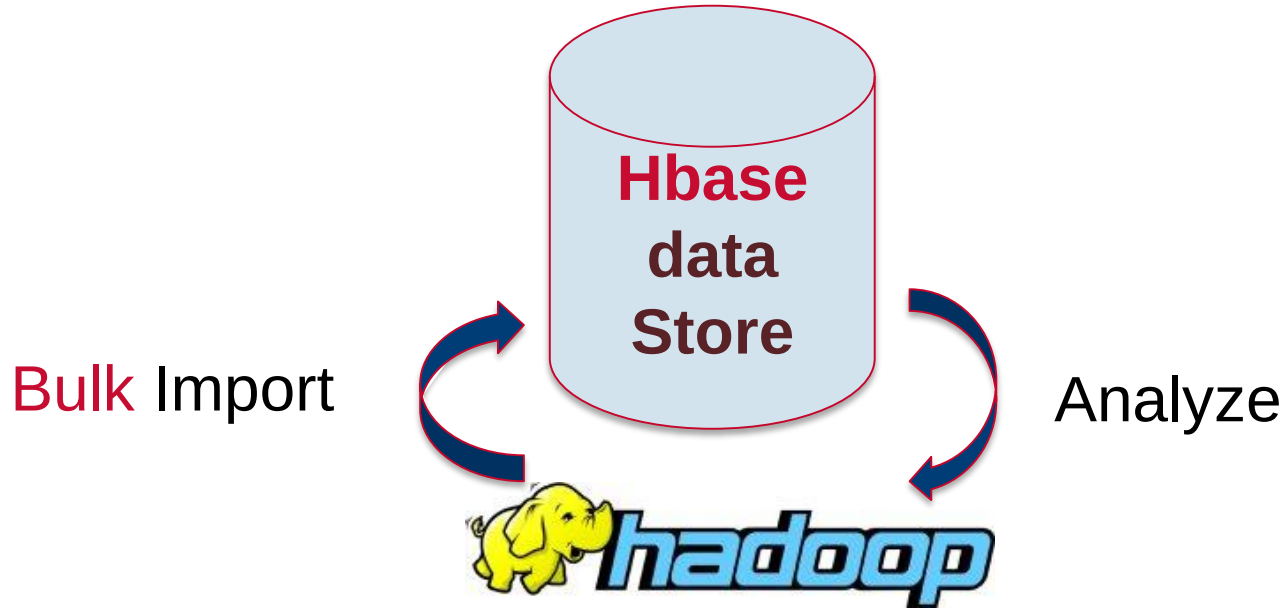
Learning Goals

- Using MapReduce or Hive with HBase

Data access patterns

Use Cases:

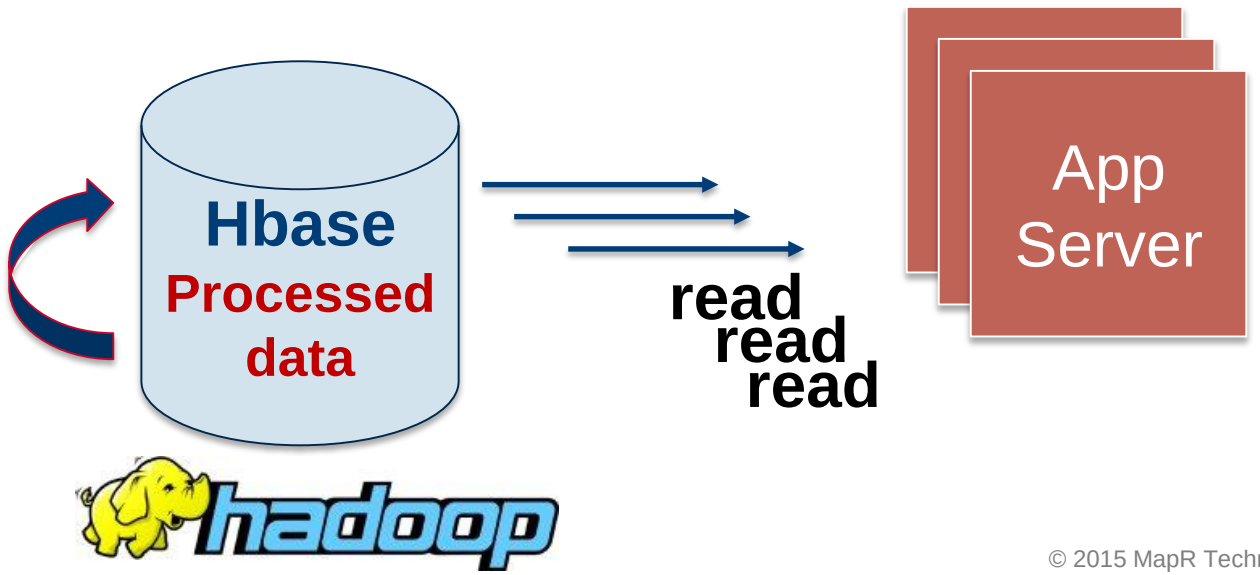
- **Large scale offline** ETL analytics, Generating derived data
  - Bulk Import



Data access patterns

Use Cases:

- **Materialized** View, Pre-Calculated Summaries
  - Map Reduce to **update schema offline**
  - **Online Catalog, Online Dashboard**

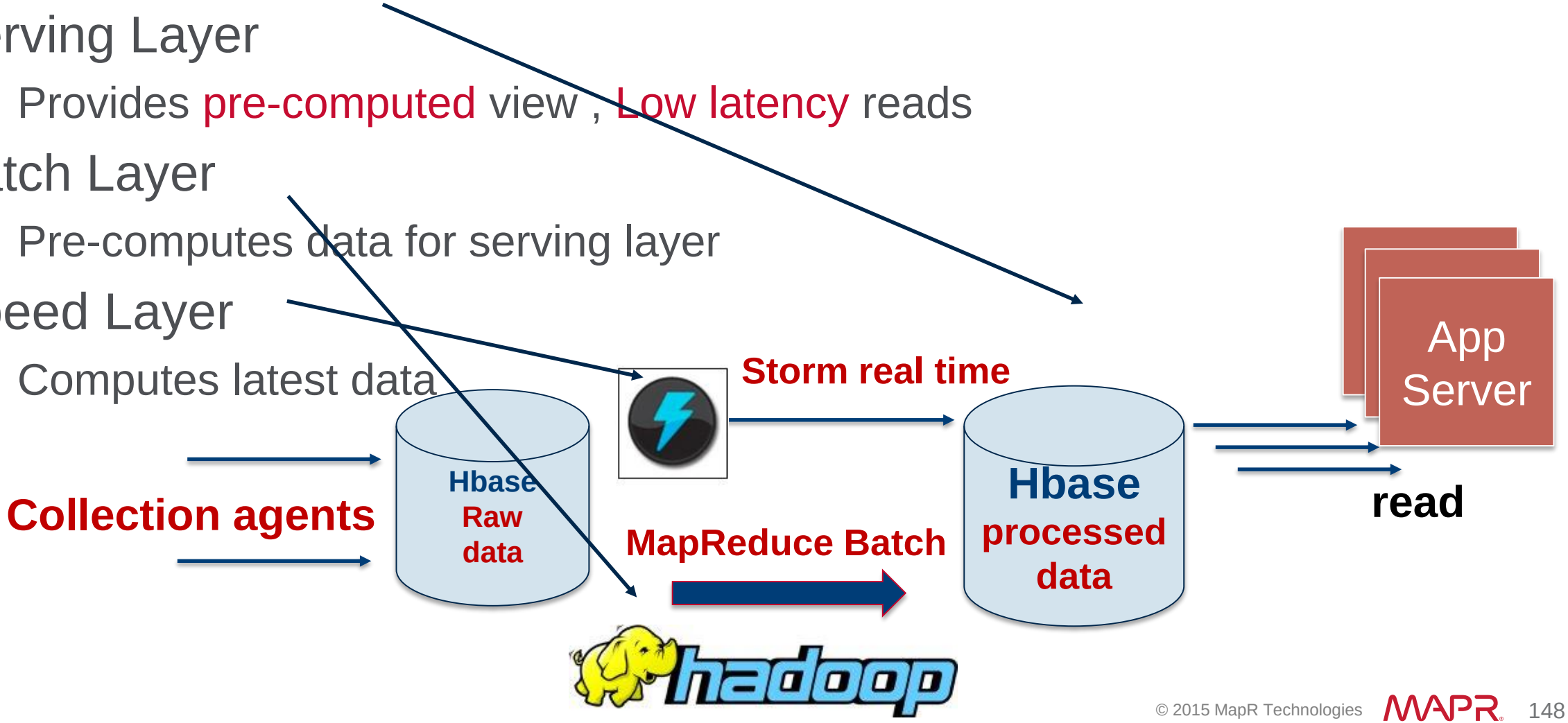


Bulk Import

Data access patterns

Lamba architecture

- Serving Layer
  - Provides pre-computed view , Low latency reads
- Batch Layer
  - Pre-computes data for serving layer
- Speed Layer
  - Computes latest data



Example: Word Count

"The time has come," the Walrus said,
"To talk of many things:
Of shoes—**and** ships—**and** sealing-wax

the, 1
time, 1
has, 1
come, 1
…
and, 1
…
and, 1
…

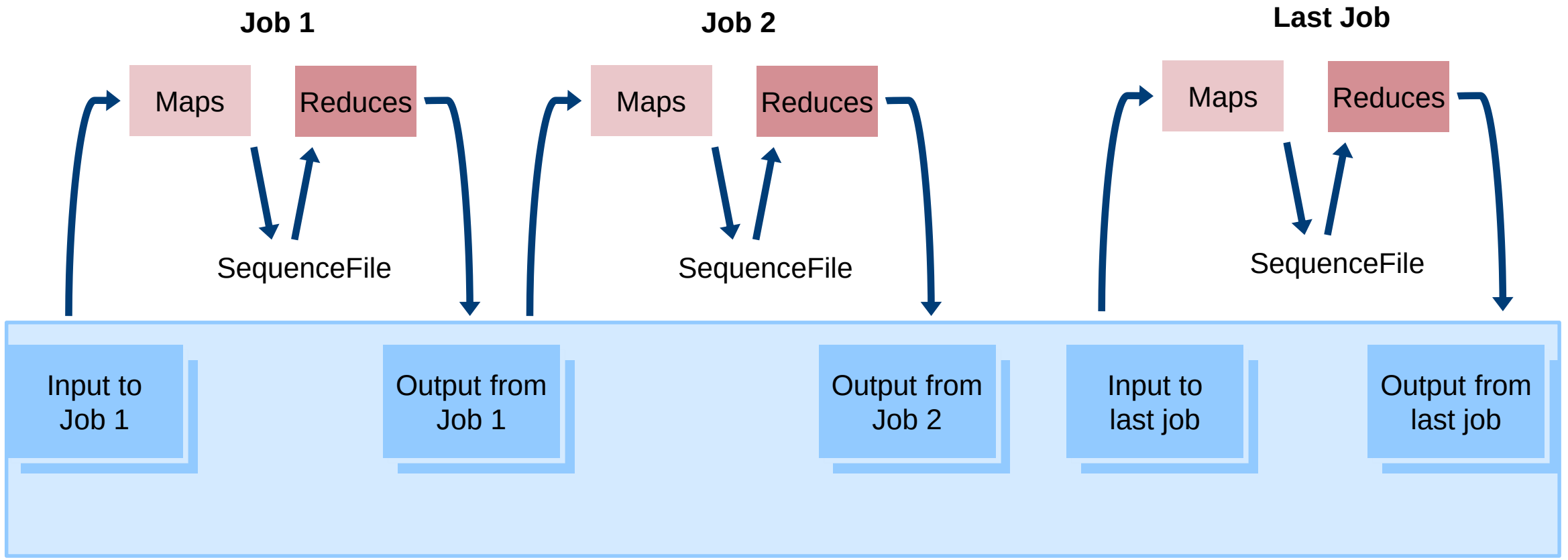
and, [1, **1**, 1]
come, [1,1,1]
has, [1,1]
the, [1,1,1]
time, [1,1,1,1]
…

and, 12
come, 6
has, 8
the, 4
time, 14
…

Input **Map** **Shuffle and Sort** **Reduce** **Output**

Reduce

Typical MapReduce Workflows



MapReduce Design Patterns

- Summarization
  - Inverted index, counting
- Filtering
  - Top ten, distinct
- Aggregation
- Data Organziation
  - partitioning
- Join
  - Join data sets
- Metapattern
  - Job chaining

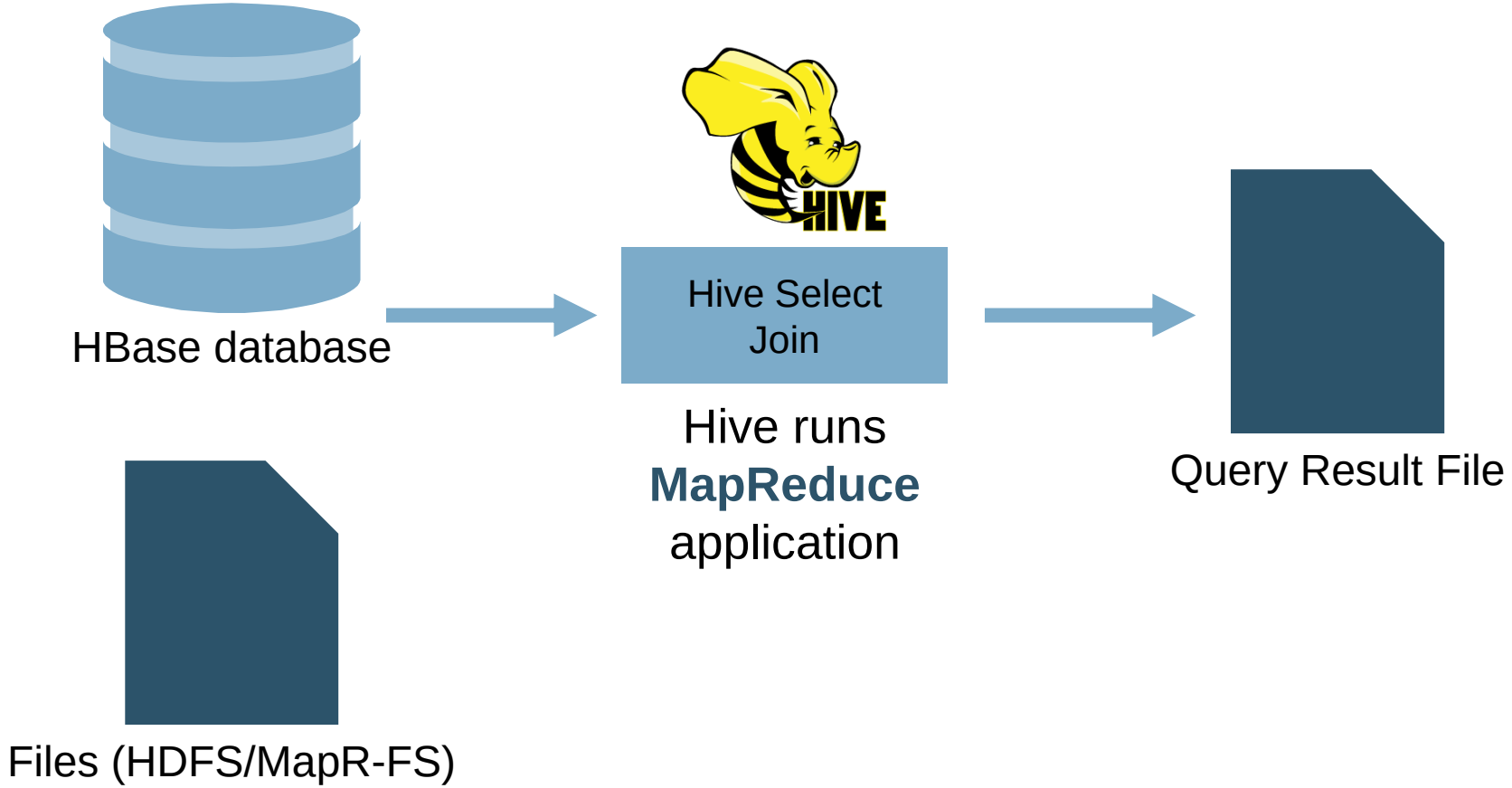
MapReduce Can be Complex

```
public void reduce(Text key, Iterable<Text> values, Context context) throws
IOException, InterruptedException {
for (Text value: values) {
      compositeString = value.toString();
      compositeStringArray = compositeString.split("_");
      tempYear = new Text(compositeStringArray[0]);
      tempValue = new Long(compositeStringArray[1]).longValue();
      if(tempValue < min) {
         min=tempValue;
         minYear=tempYear;
      }
   }
   Text keyText = new Text("min" + "(" + minYear.toString() + "): ");
   context.write(keyText, new FloatWritable(min));
 }
```

What is Hive?

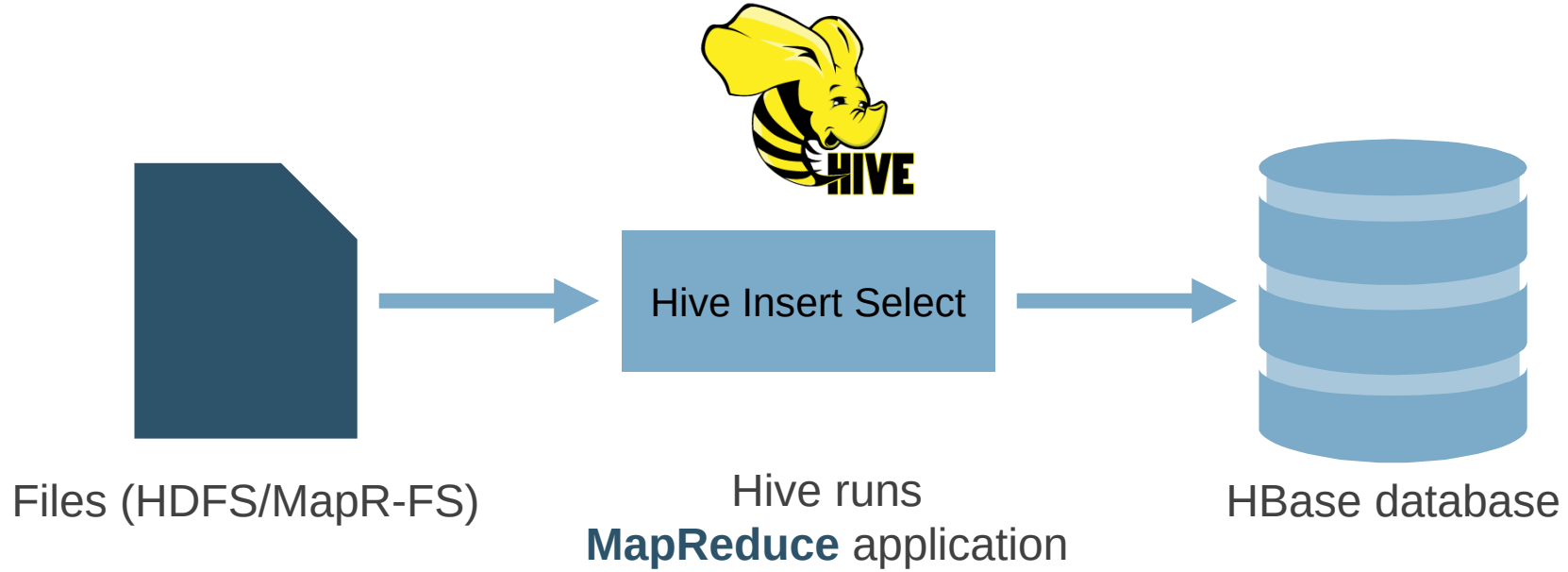
- Data Warehouse on top of Hadoop
  - Gives ability to query without programming
  - Used for analytical querying of data
- SQL like execution for Hadoop
- SQL evaluates to MapReduce code
  - Submits jobs to your cluster

Using HBase as a MapReduce/Hive Source



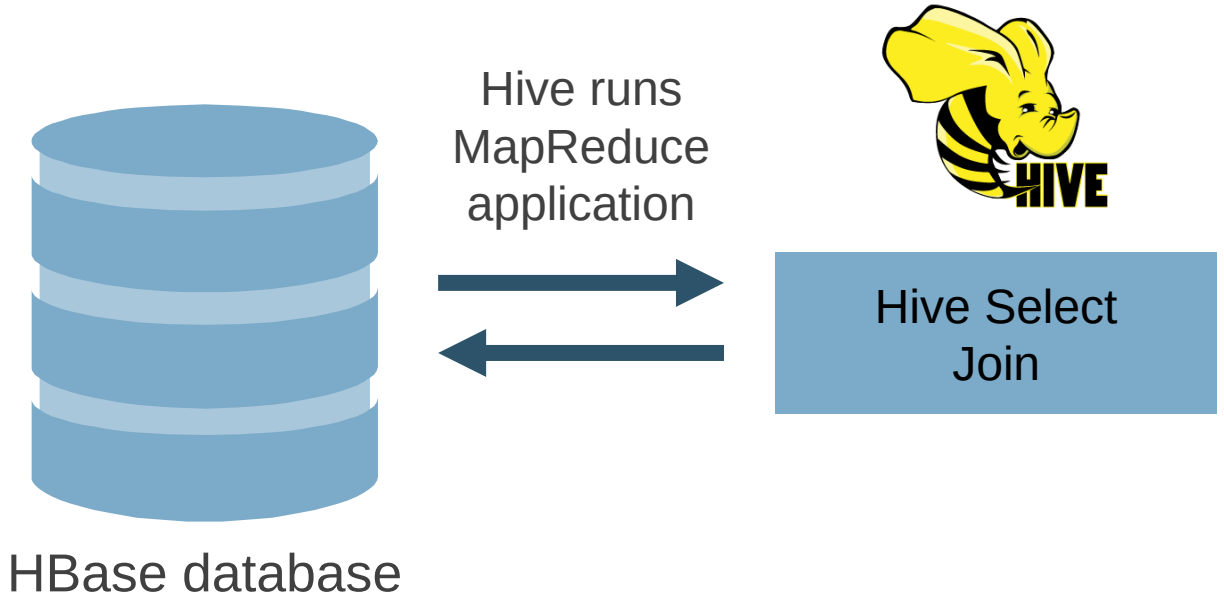
EXAMPLE: Data Warehouse for Analytical Processing queries

Using HBase as a MapReduce or Hive Sink



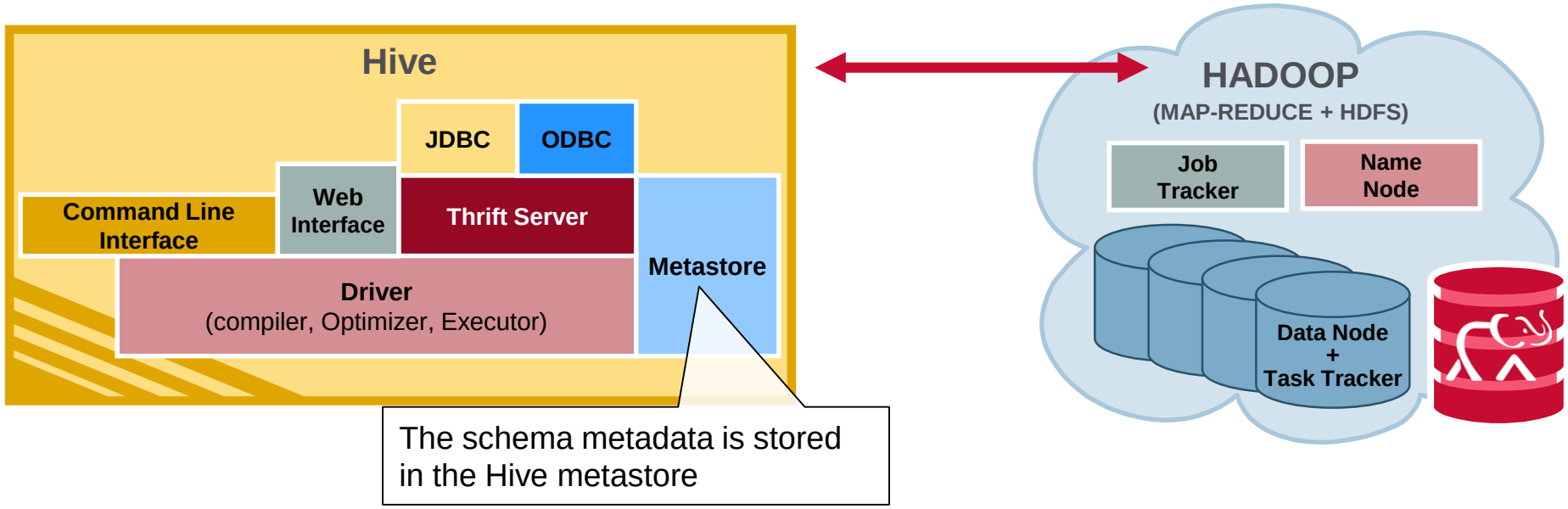
EXAMPLE: bulk load data into a table

Using HBase as a Source & Sink



EXAMPLE: calculate and store summaries, Pre-Computed, Materialized View

Hive Metastore



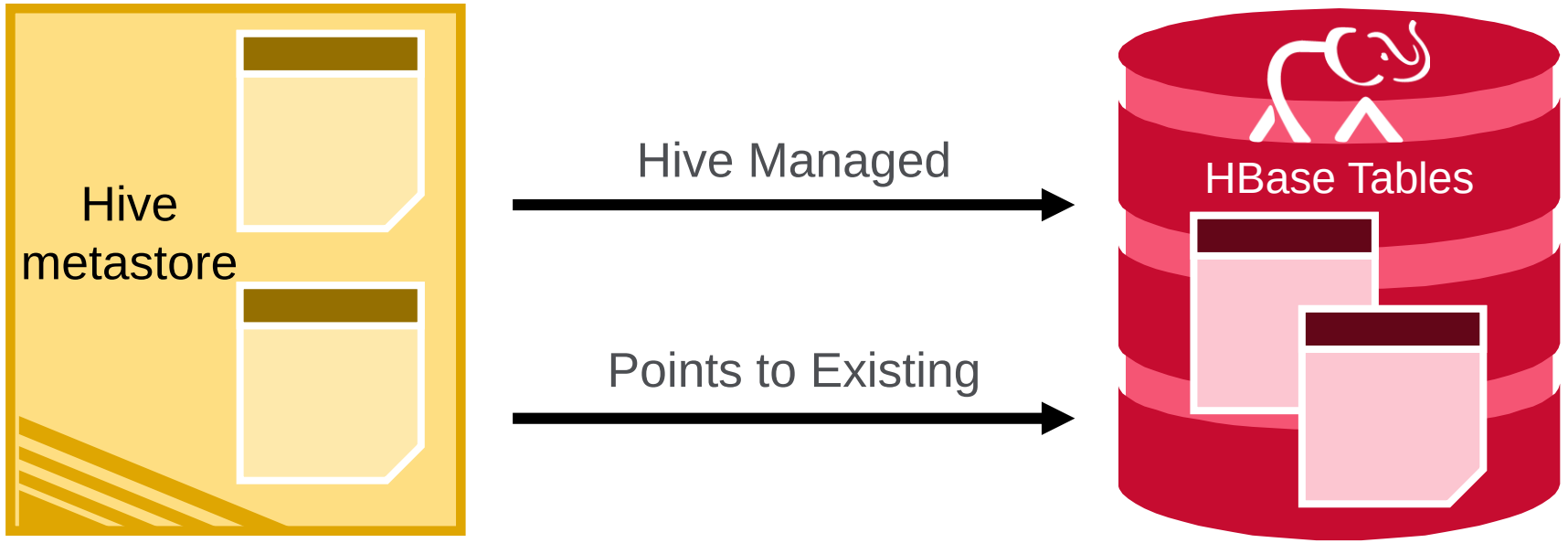
Hive Table definition

key string	price bigint	vol bigint

HBase trades_tall Table

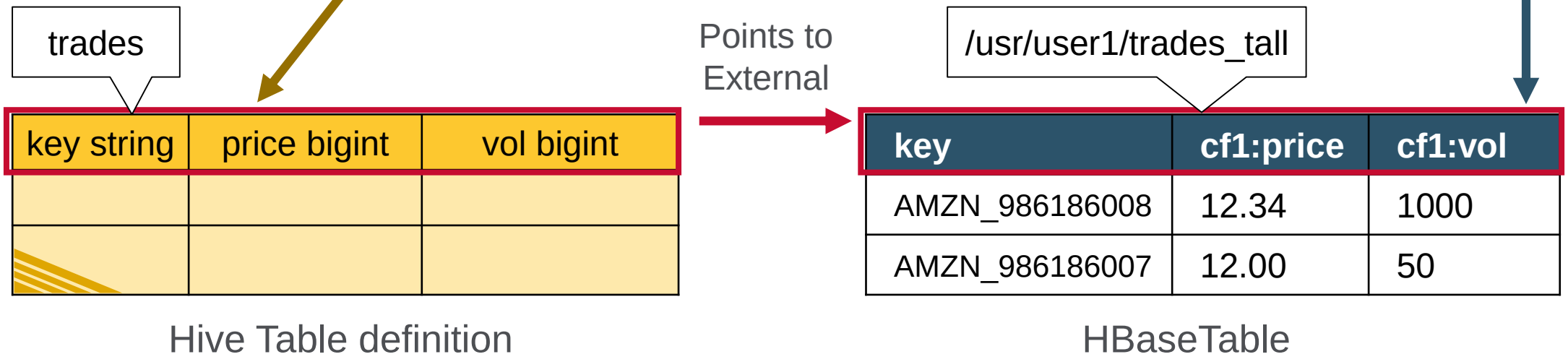
key	cf1:price	cf1:vol
AMZN_986186008	12.34	1000
AMZN_986186007	12.00	50

Hive HBase



Hive HBase – External Table

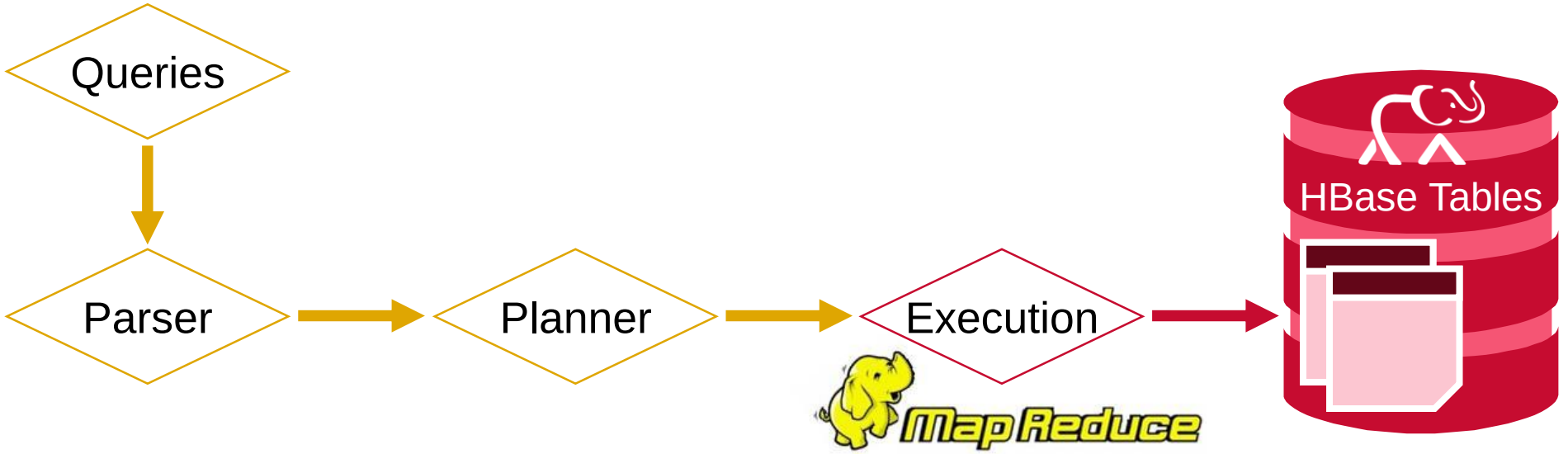
```
CREATE EXTERNAL TABLE trades(key string, price bigint, vol bigint)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ("hbase.columns.mapping"= ":key,cf1:price#b,cf1:vol#b")
TBLPROPERTIES ("hbase.table.name" = "/usr/user1/trades_tall");
```



Hive HBase – Hive Query

SQL evaluates to MapReduce code

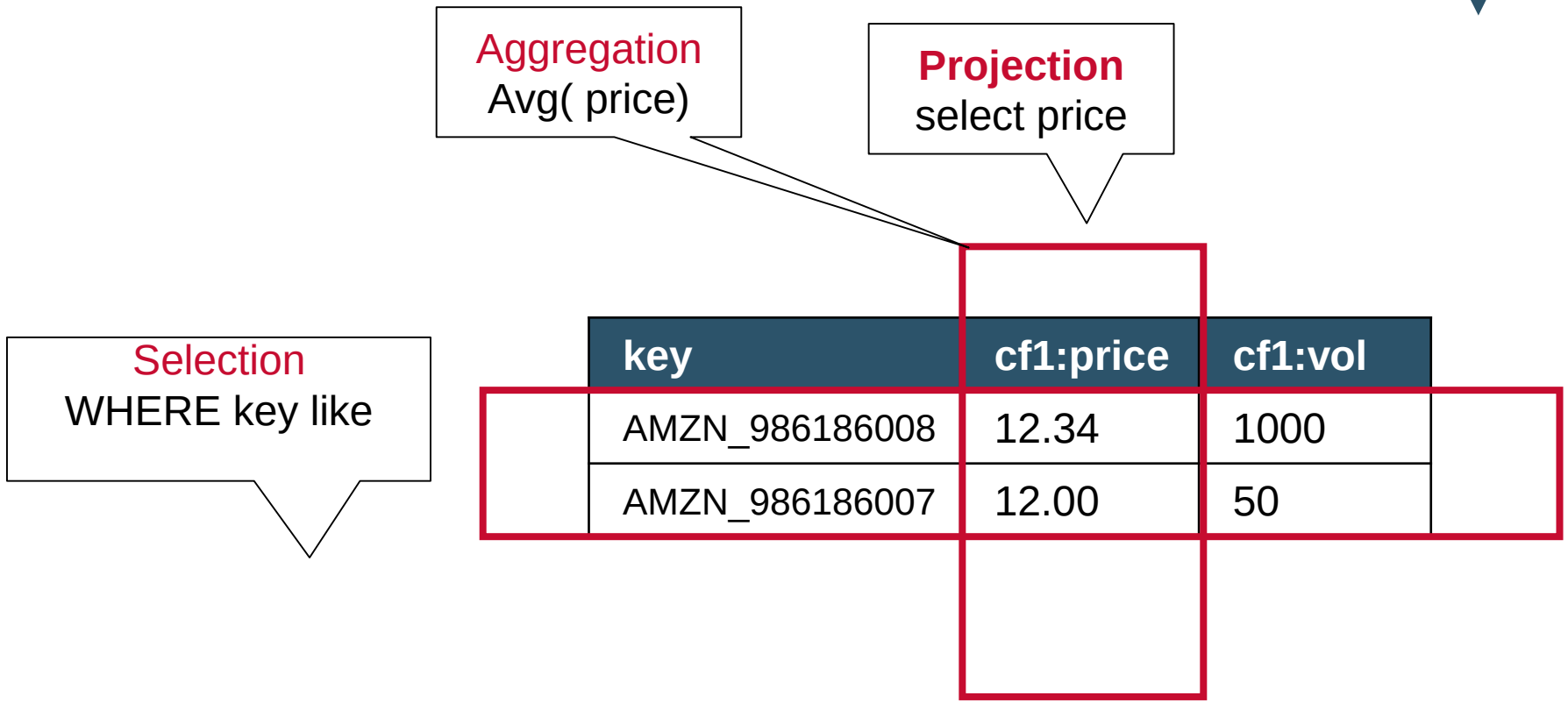
```
SELECT AVG(price) FROM trades WHERE key LIKE "GOOG" ;
```



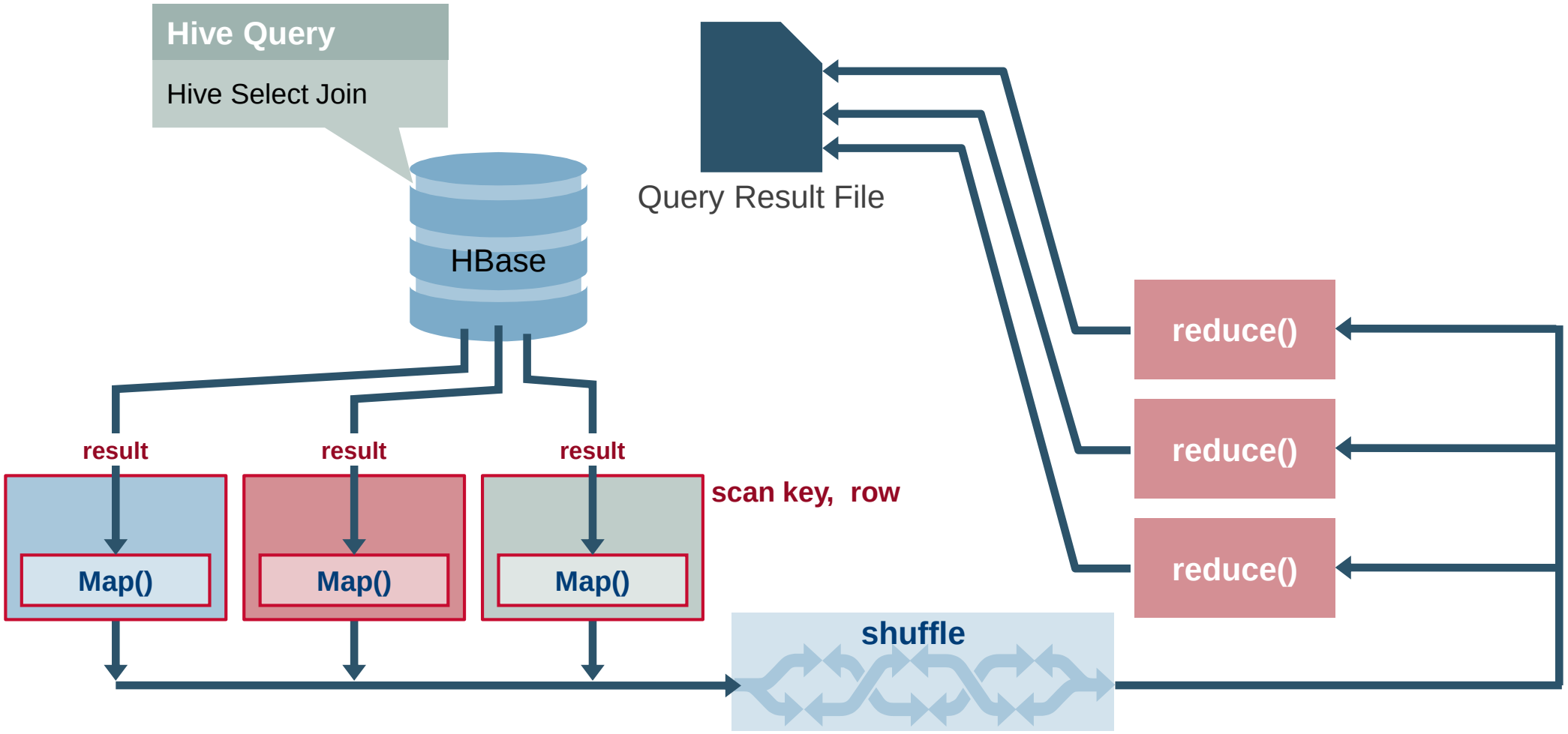
Hive HBase – External Table

SQL evaluates to MapReduce code

```
SELECT AVG(price) FROM trades WHERE key LIKE "AMZN" ;
```



Hive Map Reduce

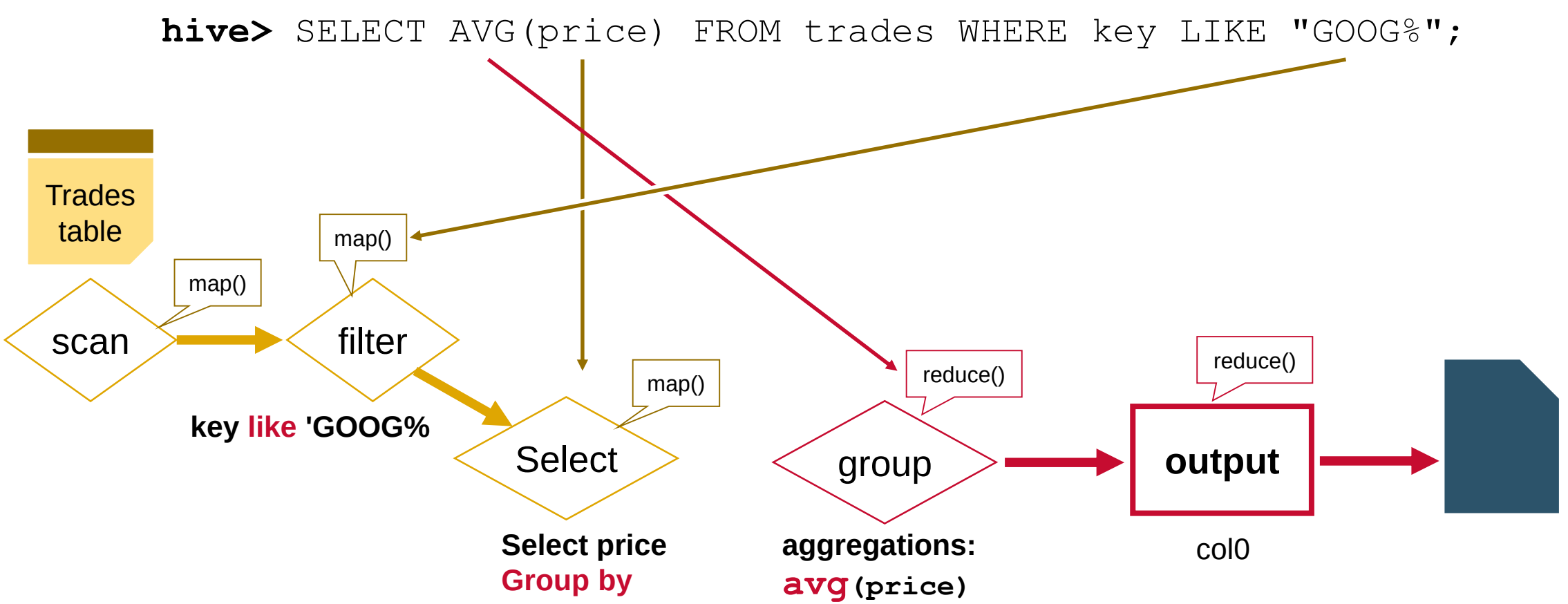


Hive Query Plan

```
EXPLAIN SELECT AVG(price) FROM trades WHERE key LIKE "GOOG%";
```

STAGE PLANS:
Stage: Stage-1
Map Reduce
Map Operator Tree:
TableScan
Filter Operator
predicate: (key like 'GOOG%') (type: boolean)
Select Operator
Group By Operator
Reduce Operator Tree:
Group By Operator
Select Operator
File Output Operator

Hive Query Plan – (2)



Some Hive Design Patterns

- Summarization
  - **Select min(delay), max(delay), count(*) from flights group by carrier;**
- Filtering
  - `SELECT * FROM trades WHERE key LIKE "GOOG%";`
  - `SELECT price FROM trades DESC LIMIT 10 ;`
- Join

```
SELECT tableA.field1, tableB.field2 FROM tableA
JOIN tableB
ON tableA.field1 = tableB.field2;
```

Lab – Query HBase airline data with Hive

Import mapping to Row Key and Columns:

Row-key Carrier-Flightnumber-Date-Origin-destination	delay			info			stats		timing	
	Air Craft delay	Arr delay	Carrier delay	cncl	Cncl code	tailnum	distance	elaptime	arrtime	Dep time
AA-1-2014-01-01-JFK-LAX		13		0		N7704	2475	385.00	359	…

Count number of cancellations by reason (code)

$ hive

hive> explain select count(*) as cancellations, cnclcode from flighttable where cncl=1 group by cnclcode order by cancellations asc limit 100;

```
OK
STAGE DEPENDENCIES:
  Stage-1 is a root stage
  Stage-2 depends on stages: Stage-1
  Stage-0 is a root stage
STAGE PLANS:
  Stage: Stage-1
    Map Reduce
      Map Operator Tree:
          TableScan
            Filter Operator
              Select Operator
                Group By Operator
                  aggregations: count()
                  Reduce Output Operator
      Reduce Operator Tree:
        Group By Operator
          aggregations: count(VALUE._col0)
          Select Operator
            File Output Operator
  Stage: Stage-2
    Map Reduce
      Map Operator Tree:
          TableScan
            Reduce Output Operator
      Reduce Operator Tree:
        Extract
          Statistics: Num rows: 0 Data size: 0 Basic stats: NONE Column stats: NONE
          Limit
            File Output Operator
  Stage: Stage-0
    Fetch Operator
      limit: 100
```

2 MapReduce jobs

$ hive
hive> select count(*) as cancellations, cnclcode **from** flighttable **where** cncl=1 **group by** cnclcode **order by** cancellations **asc** limit 100;

```
Total jobs = 2
MapReduce Jobs Launched:
Job 0: Map: 1  Reduce: 1   Cumulative CPU: 13.3 sec   MAPRFS Read: 0
MAPRFS Write: 0 SUCCESS
Job 1: Map: 1  Reduce: 1   Cumulative CPU: 1.52 sec   MAPRFS Read: 0
MAPRFS Write: 0 SUCCESS
Total MapReduce CPU Time Spent: 14 seconds 820 msec
OK
4598    C
7146    A
```

Find the longest airline delays

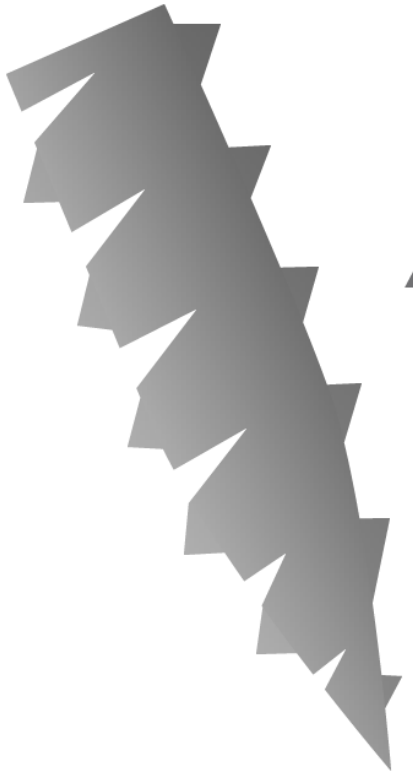
$ hive

hive> select arrdelay,key from flighttable where arrdelay > 1000 order by arrdelay desc limit 10;

```
MapReduce Jobs Launched:
Map: 1  Reduce: 1
OK
1530.0  AA-385-2014-01-18-BNA-DFW
1504.0  AA-1202-2014-01-15-ONT-DFW
1473.0  AA-1265-2014-01-05-CMH-LAX
1448.0  AA-1243-2014-01-21-IAD-DFW
1390.0  AA-1198-2014-01-11-PSP-DFW
1335.0  AA-1680-2014-01-21-SLC-DFW
1296.0  AA-1277-2014-01-21-BWI-DFW
1294.0  MQ-2894-2014-01-02-CVG-DFW
1201.0  MQ-3756-2014-01-01-CLT-MIA
1184.0  DL-2478-2014-01-10-BOS-ATL
```

Lab

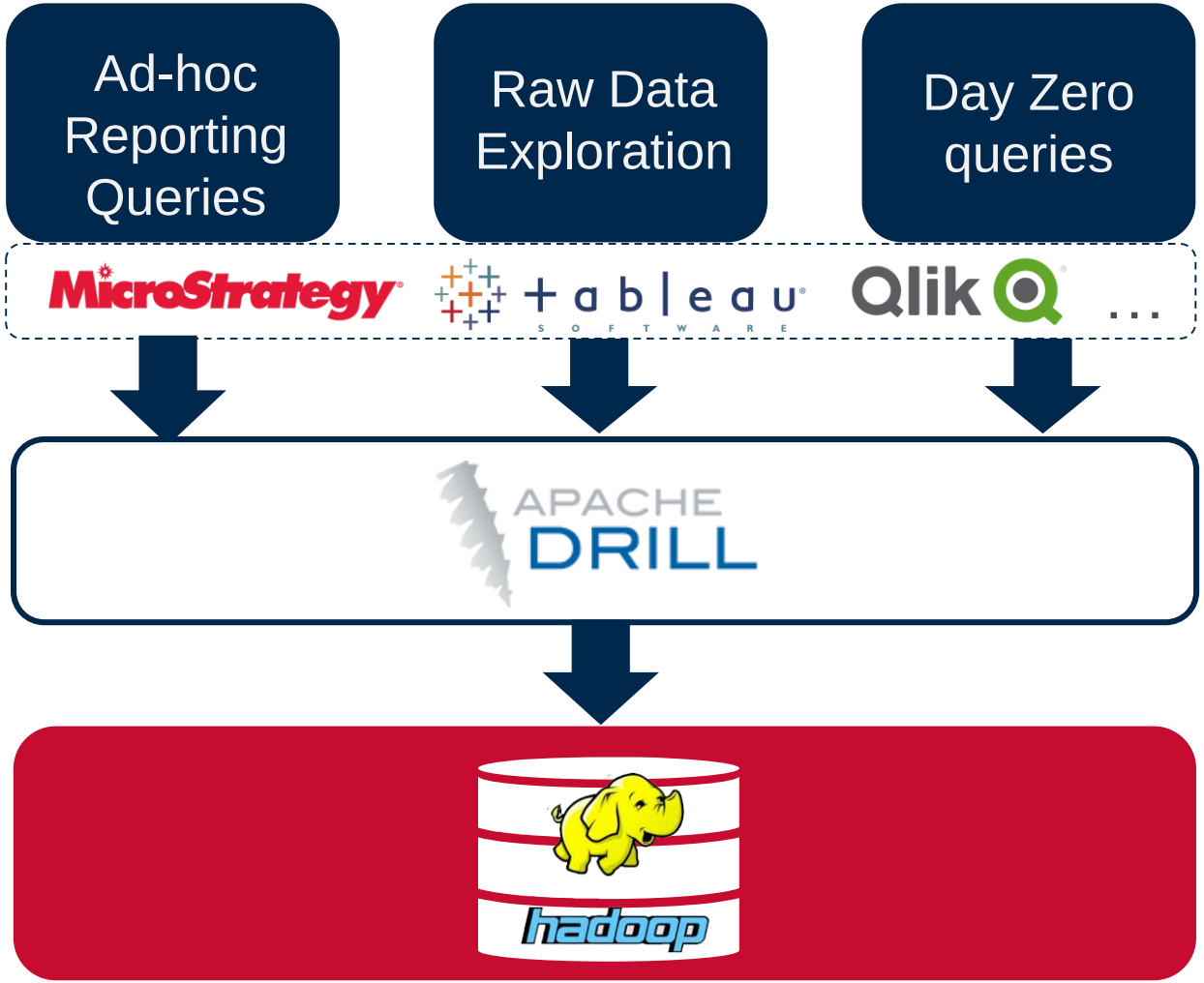
- Hive



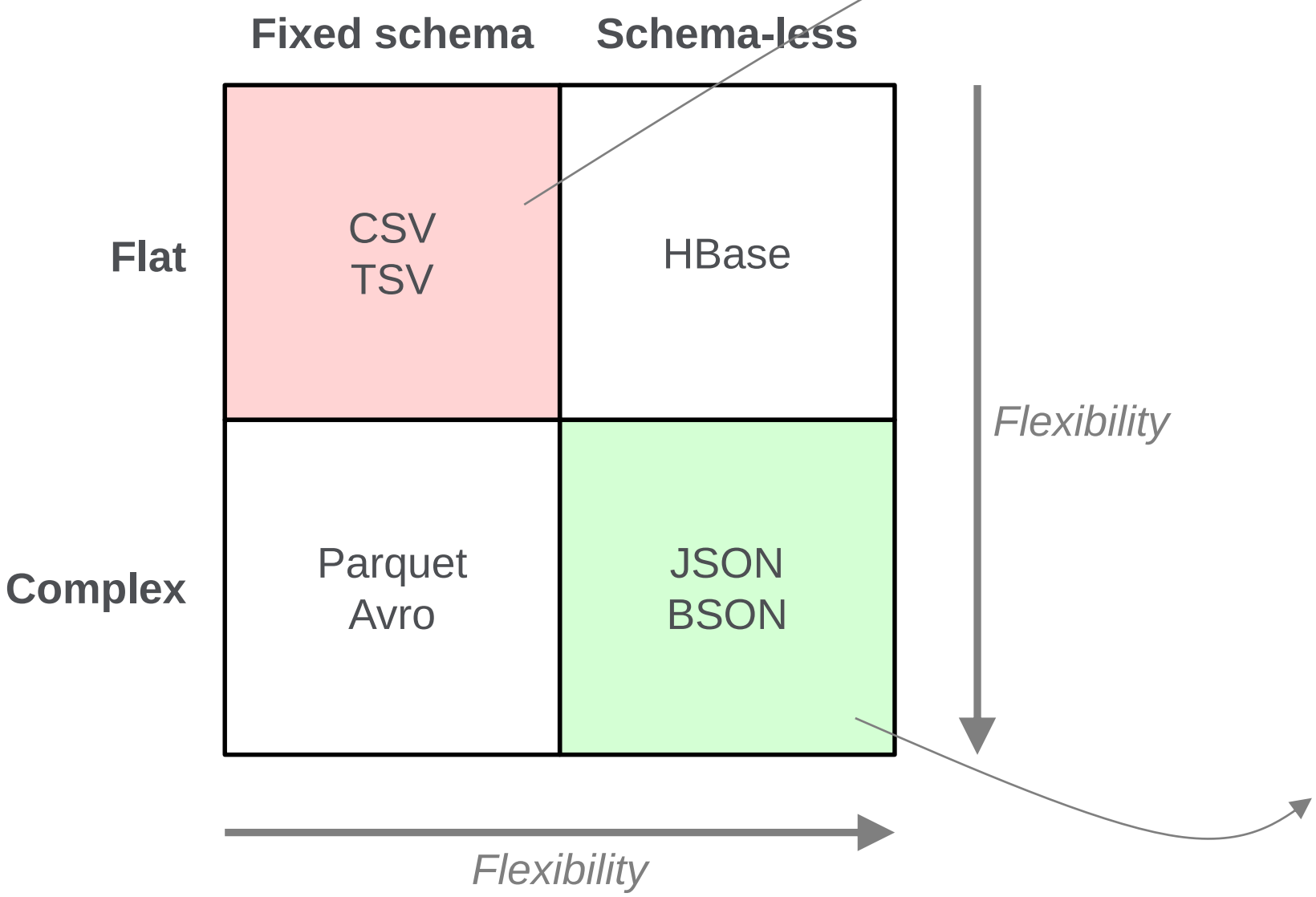
APACHE DRILL

Self-Service Data Exploration

Direct access to Hadoop data from familiar BI / Analytics tools- ANSI SQL compatible



Drill's Data Model is Flexible



RDBMS/SQL-on-Hadoop table

Name	Gender	Age
Michael	M	6
Jennifer	F	3

Apache Drill table

```
{
  name: {
    first: Michael,
    last: Smith
  },
  hobbies: [ski, soccer],
  district: Los Altos
}
{
  name: {
    first: Jennifer,
    last: Gates
  },
  hobbies: [sing],
  preschool: CCLC
}
```

Drill Supports *Schema Discovery On-The-Fly*

Schema Declared In Advance

- Fixed schema
- Leverage schema in centralized repository (Hive Metastore)

Schema Discovered On-The-Fly

- Fixed schema, evolving schema or schema-less
- Leverage schema in centralized repository or self-describing data

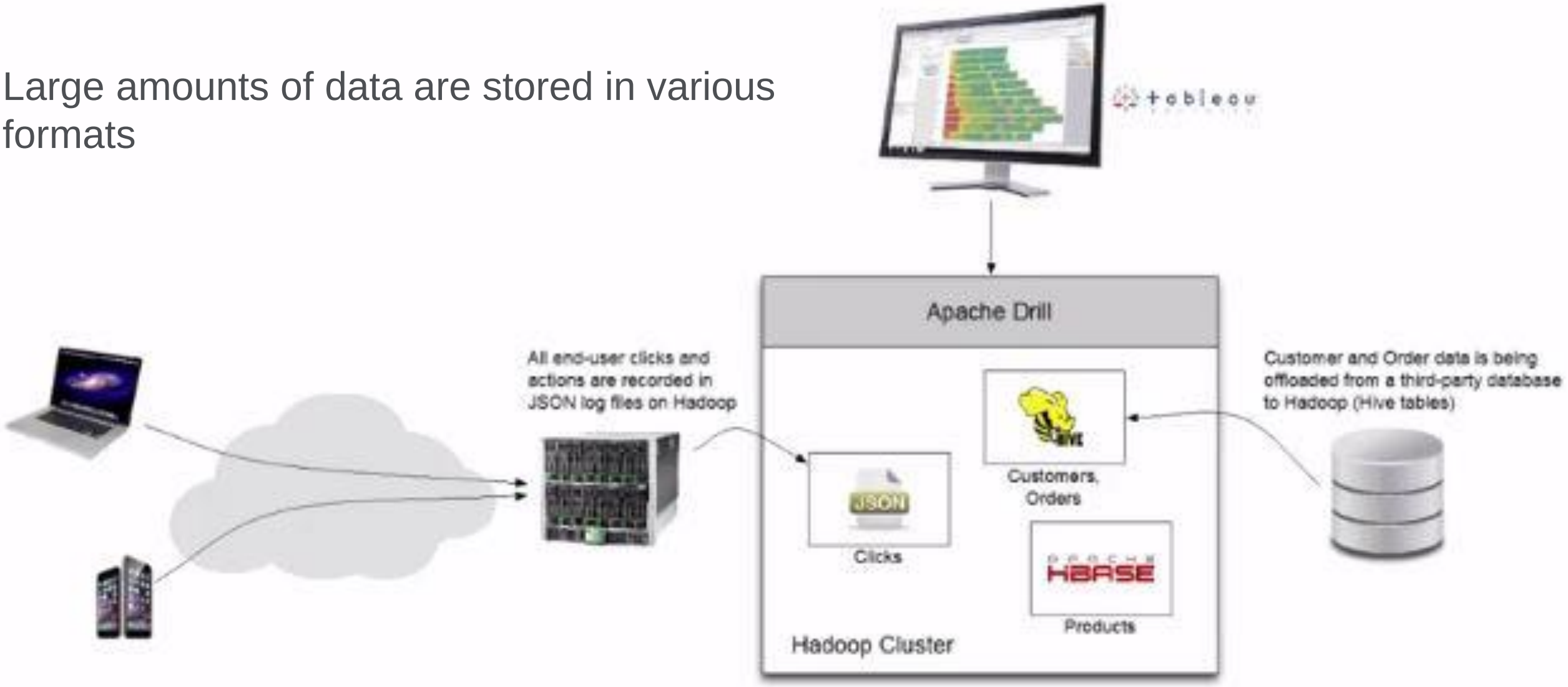
SCHEMA ON WRITE

SCHEMA BEFORE READ

SCHEMA ON THE FLY

Use Case: Online Retail Business

Large amounts of data are stored in various formats



Use Case: Online Retail Business

Data Analyst

Analyst explores various data sources using Apache Drill

Web, mobile log Files of
User clicks on online retail web Site, Mobile app

Orders

Customer Profile
Products

Hive Data

- Orders
  - Hive

Order schema

order_id	month	cust_id	state	Prod_id	Order_total
67212	June	10001	ca	909	13

HBase Data

- Products Data
  - HBase

rowkey	CF: details		CF: pricing
	category	name	price
10	laptop	sony	1000

- Customer Data
  - Hbase

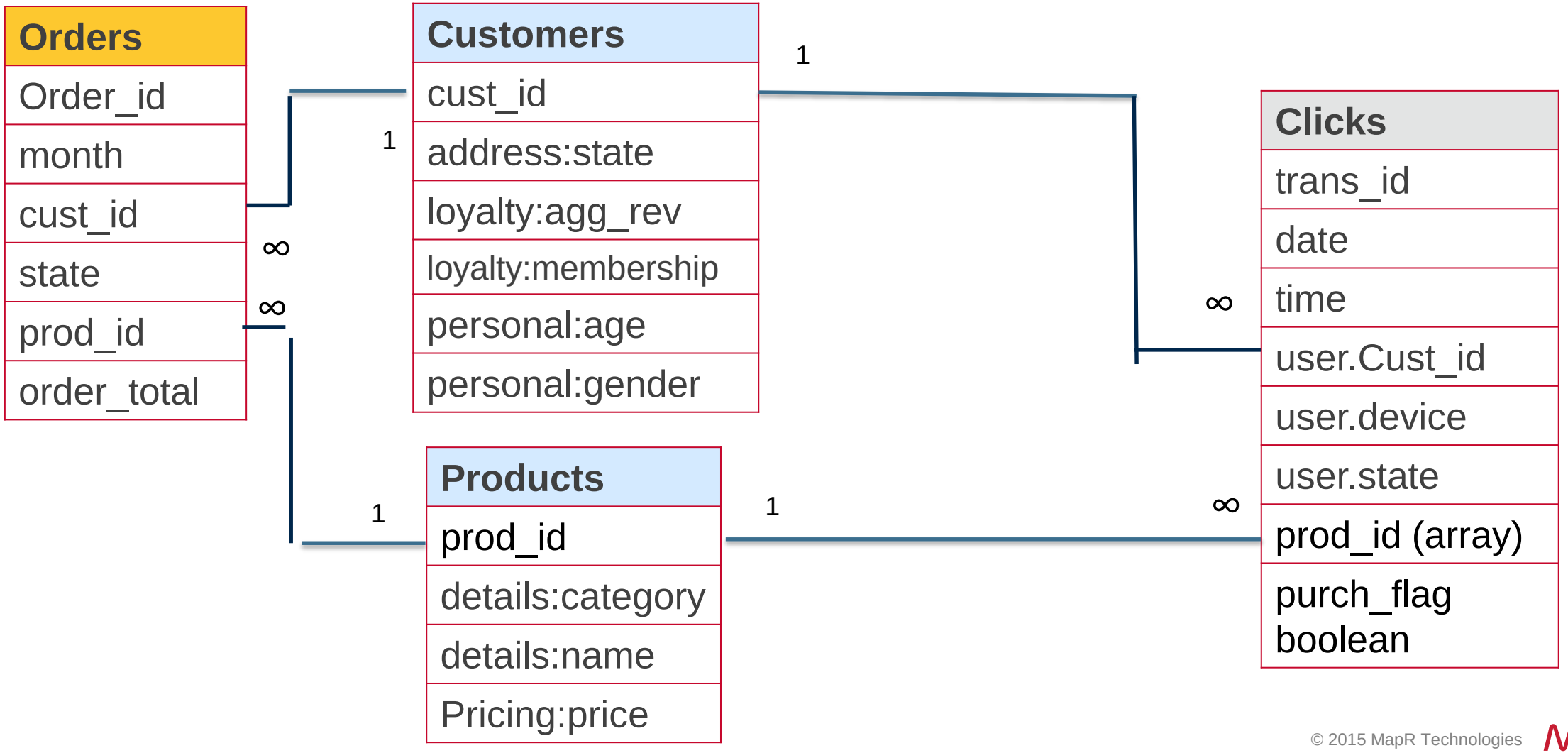
rowkey	CF: address	CF: loyalty		CF:personal	
	state	agg_rev	membership	age	gender
1	"va"	197.00	"silver"	"15-20"	"FEMALE"

Log Data

- User Web clicks logs
  - JSON files

trans_id	date	user_info.cust_id	user_info.device	user_info.state	trans_info.prod_id	trans_info.purch_flag
67212	June	10001	“iOS5”	“ca”	[174,2]	false

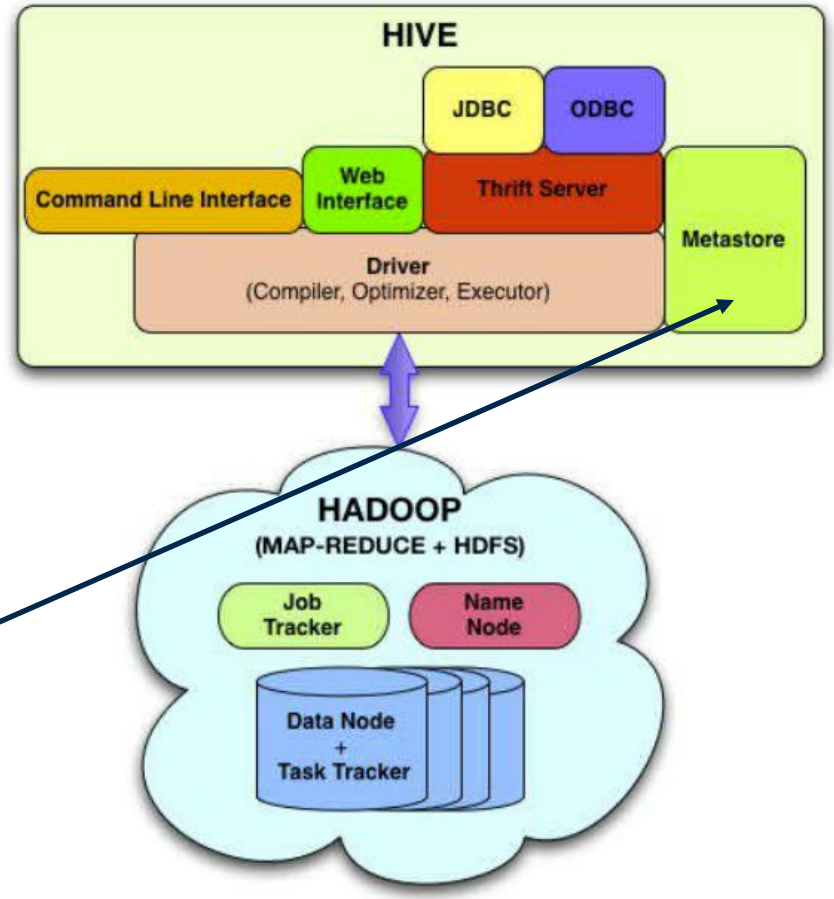
High Level “Relations” between Retail data



Use Case

- Explore the Use Case Data
  - Hive
  - HBase
  - **Files**

Drill can read Hive Metastore



- With Hive you have to define a schema for files or Hbase tables
- The schema metadata is stored in the Hive metastore
- Drill can read the Hive metastore

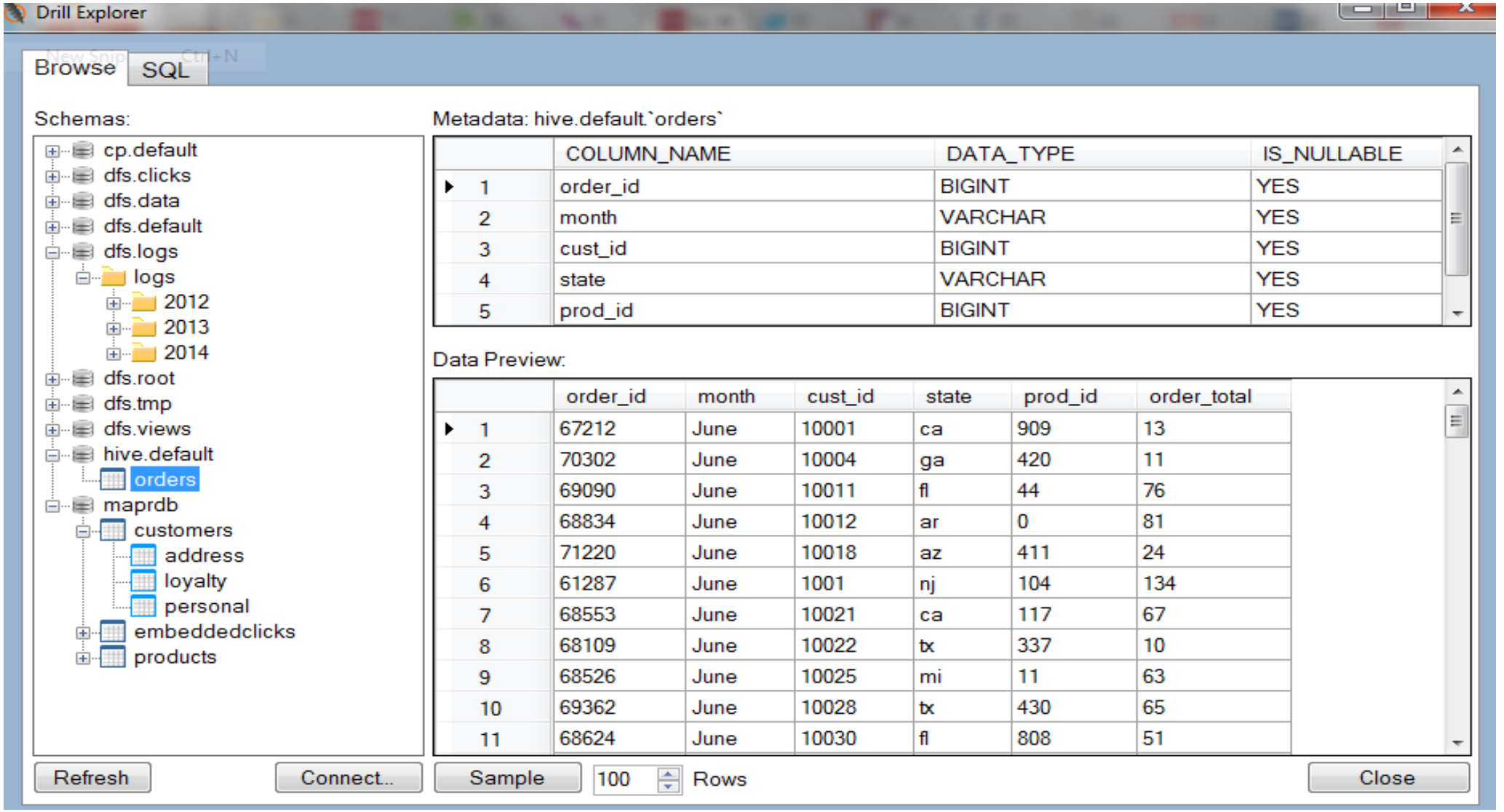
Order schema

order_id	month	cust_id	state	Prod_id	Order_total

Metadata for file → **/data/orders/month2.log.csv**

15552	February	10005	CA	909	10

Exploring the Hive Order Tables



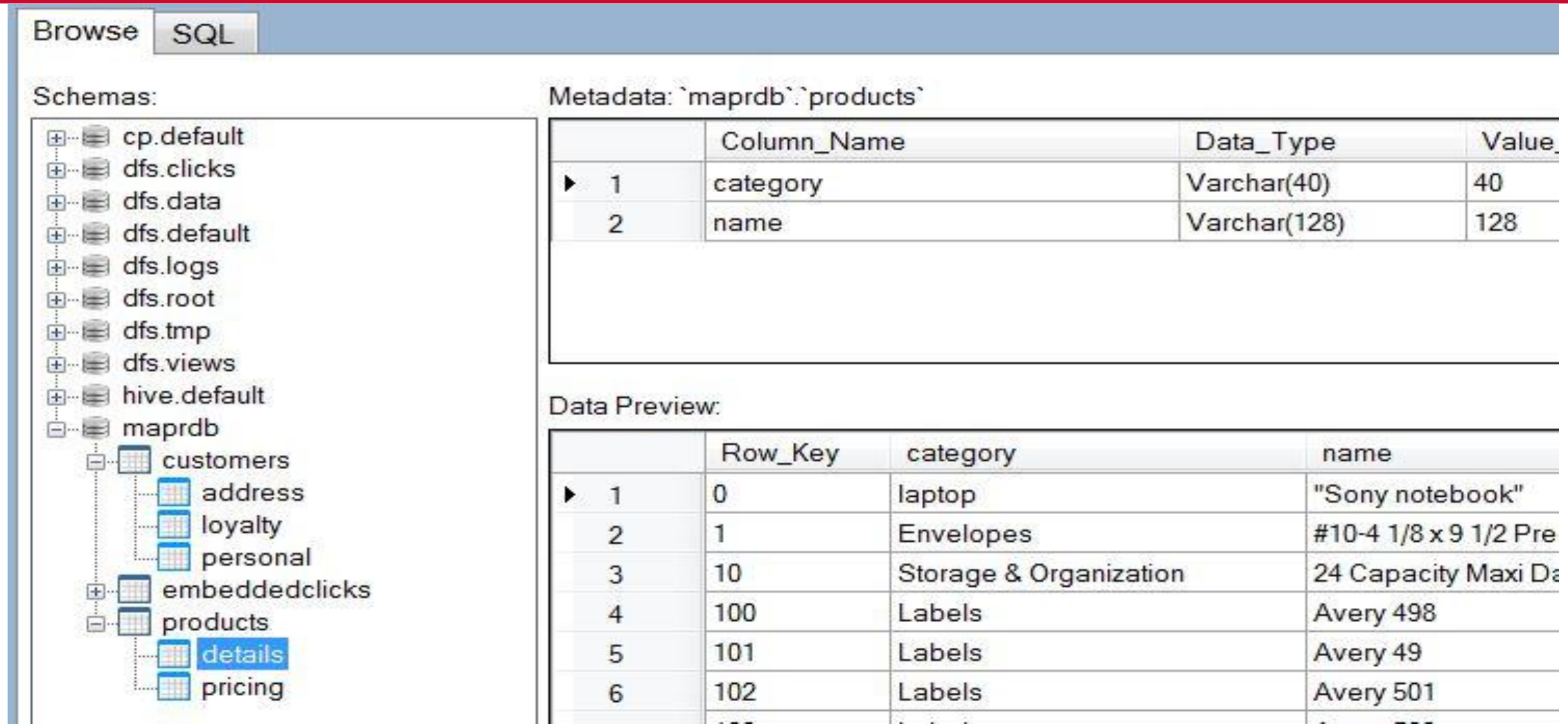
HBase Data

HBase is “schema less”

- Drill allows direct queries on HBase Tables, without mapping to a schema (like with Hive or Impala)

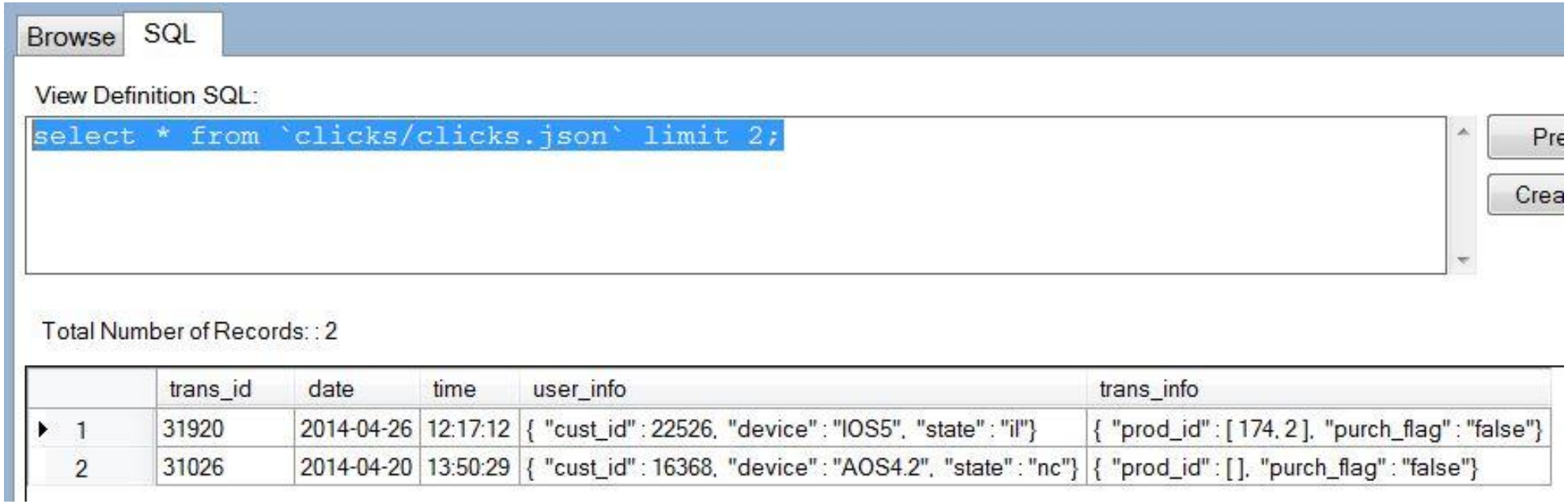
RowKey	CF1			CF2				…
	colA	colB	colC	colA	colB	colC	colD	
axxx	val		val	val			val	
…								

Exploring the HBase Tables



Query the Clicks File

```
> use dfs.clicks;
> SELECT * FROM  `clicks/clicks.json`  limit 2;
```



Browse | SQL

View Definition SQL:

```
select * from `clicks/clicks.json` limit 2;
```

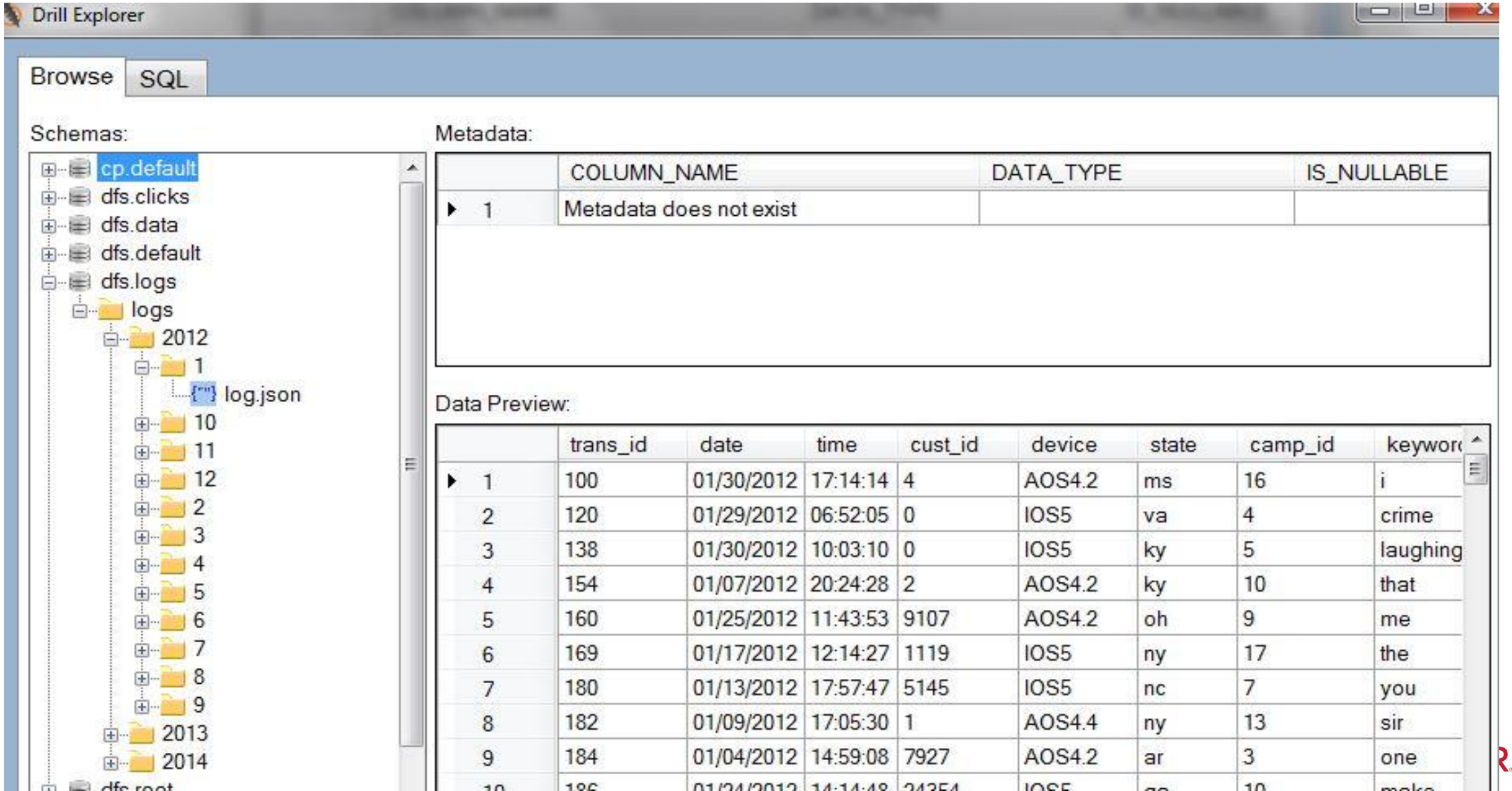
Pre

Crea

Total Number of Records: : 2

	trans_id	date	time	user_info	trans_info
▸ 1	31920	2014-04-26	12:17:12	{ "cust_id" : 22526, "device" : "IOS5", "state" : "il"}	{ "prod_id" : [174, 2], "purch_flag" : "false"}
2	31026	2014-04-20	13:50:29	{ "cust_id" : 16368, "device" : "AOS4.2", "state" : "nc"}	{ "prod_id" : [ ], "purch_flag" : "false"}

Explore the logs directory



Query the logs directory

```
use dfs.logs;
SELECT *
FROM  logs limit 100;
```

Browse | SQL

View Definition SQL:

```
SELECT * FROM dfs.logs.`logs` limit 400
```

Total Number of Records: : 400

	dir0	dir1	trans_id	date	time	cust_id	device	state	camp_id	keywords
219	2014	8	36017	08/02/2014	08:44:33	6	AOS4.2	ak	13	people
220	2014	8	36044	08/02/2014	18:48:36	29	IOS5	ms	7	oh
221	2014	8	36061	08/04/2014	18:53:31	1	IOS5	la	2	it's
222	2013	2	12115	02/23/2013	19:48:24	3	IOS5	az	5	who's
223	2013	2	12127	02/26/2013	19:42:03	11459	IOS5	wa	10	for
224	2013	2	12138	02/09/2013	05:49:01	1	IOS6	ca	7	minutes
225	2013	2	12139	02/23/2013	06:58:20	1	AOS4.4	ms	7	i

Data Sources and Storage Plugins Configuration

Data Source is in the Query

```
SELECT *
FROM   dfs.clicks.`clicks/clicks.json`
```

where you want it from

<storage plugin>.<workspace>.<tablename>

OPTIONAL

A *storage plugin name*
- DFS (default)
- HBase
- Hive

A *workspace*
- Sub-directory
- Hive database

A table
- pathname
- HBase table
- Hive table

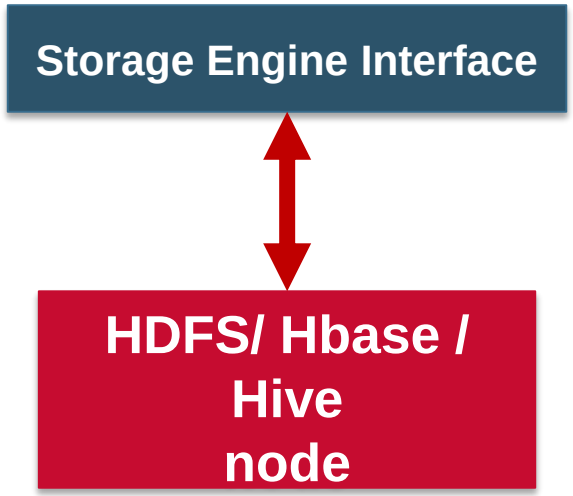
Data Source is in the Query

SELECT *
FROM **dfs.clicks.`clicks/clicks.json`**

where you want it from

Storage system type

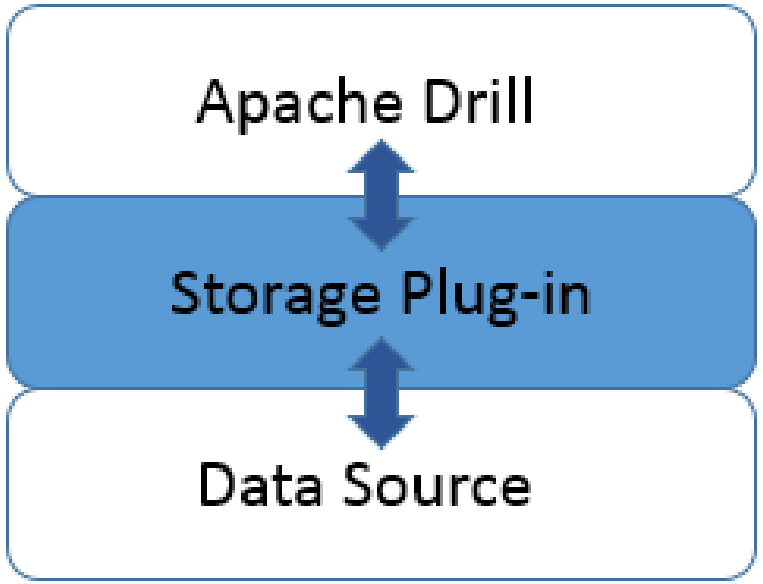
A *storage plugin name*
- **DFS (default)**
- HBase
- Hive



DFS node

Storage plugins

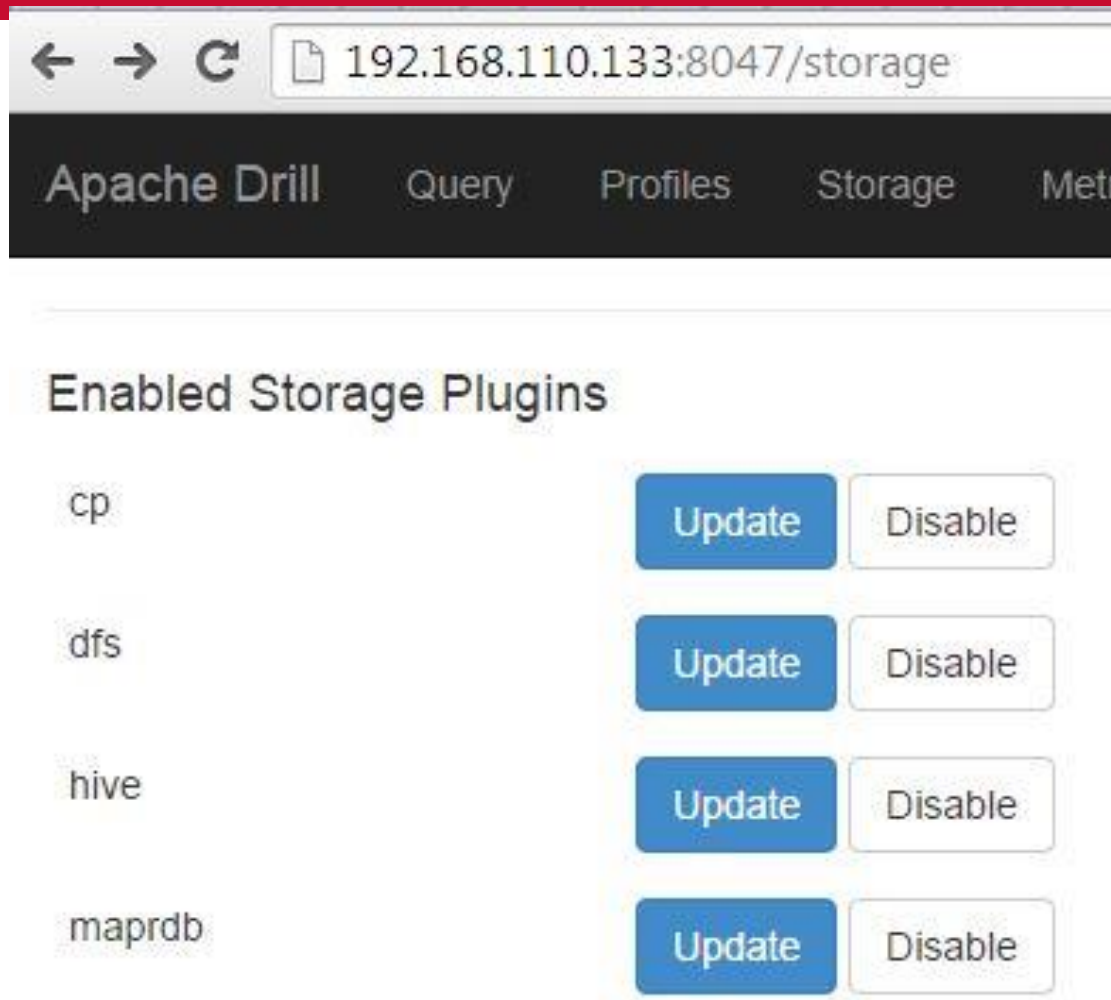
- Storage plugins provide:
  - Metadata from data source
- Storage plugins perform scanner and writer functions



Specifying the data source

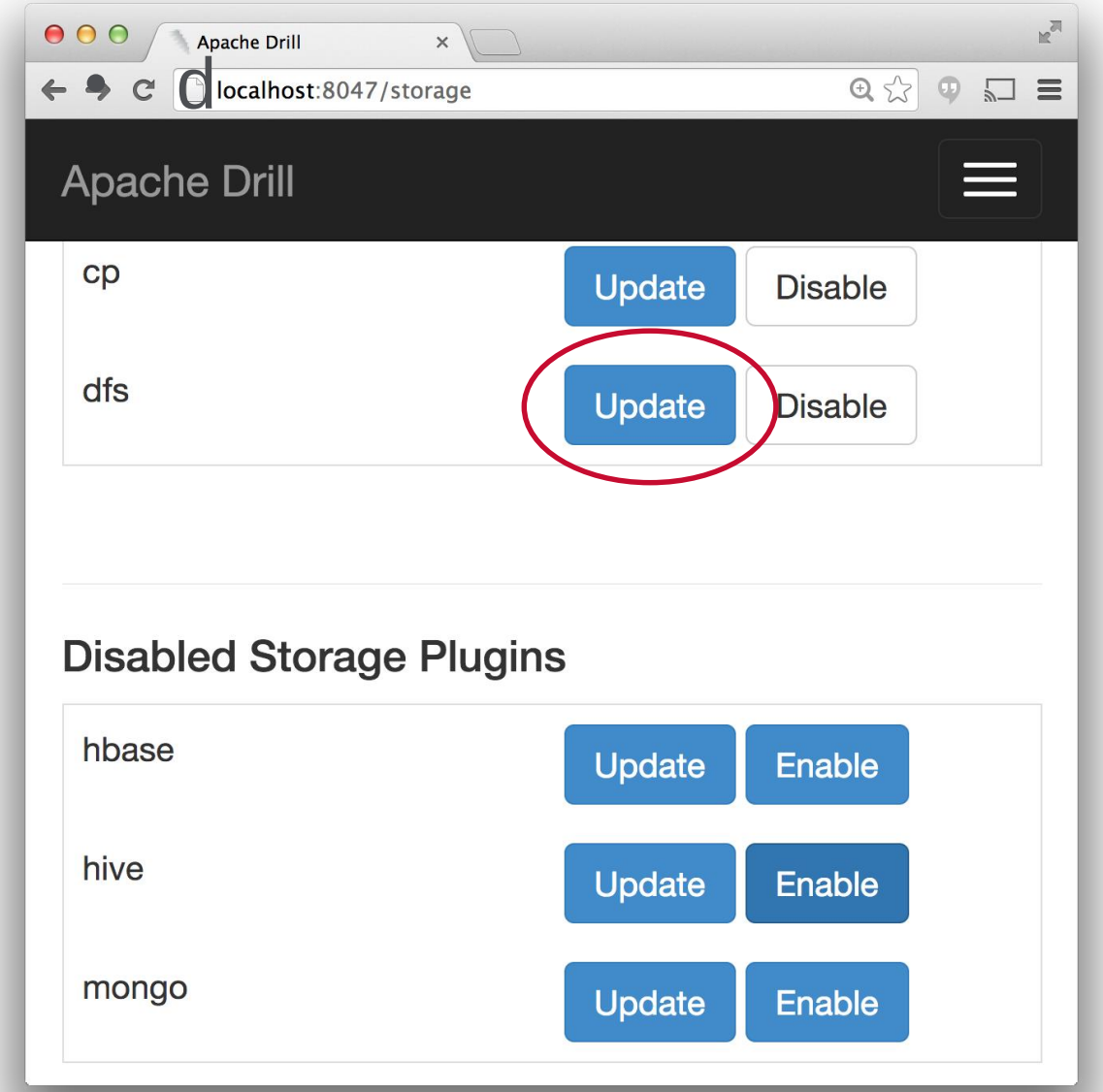
Storage Plugin Registration

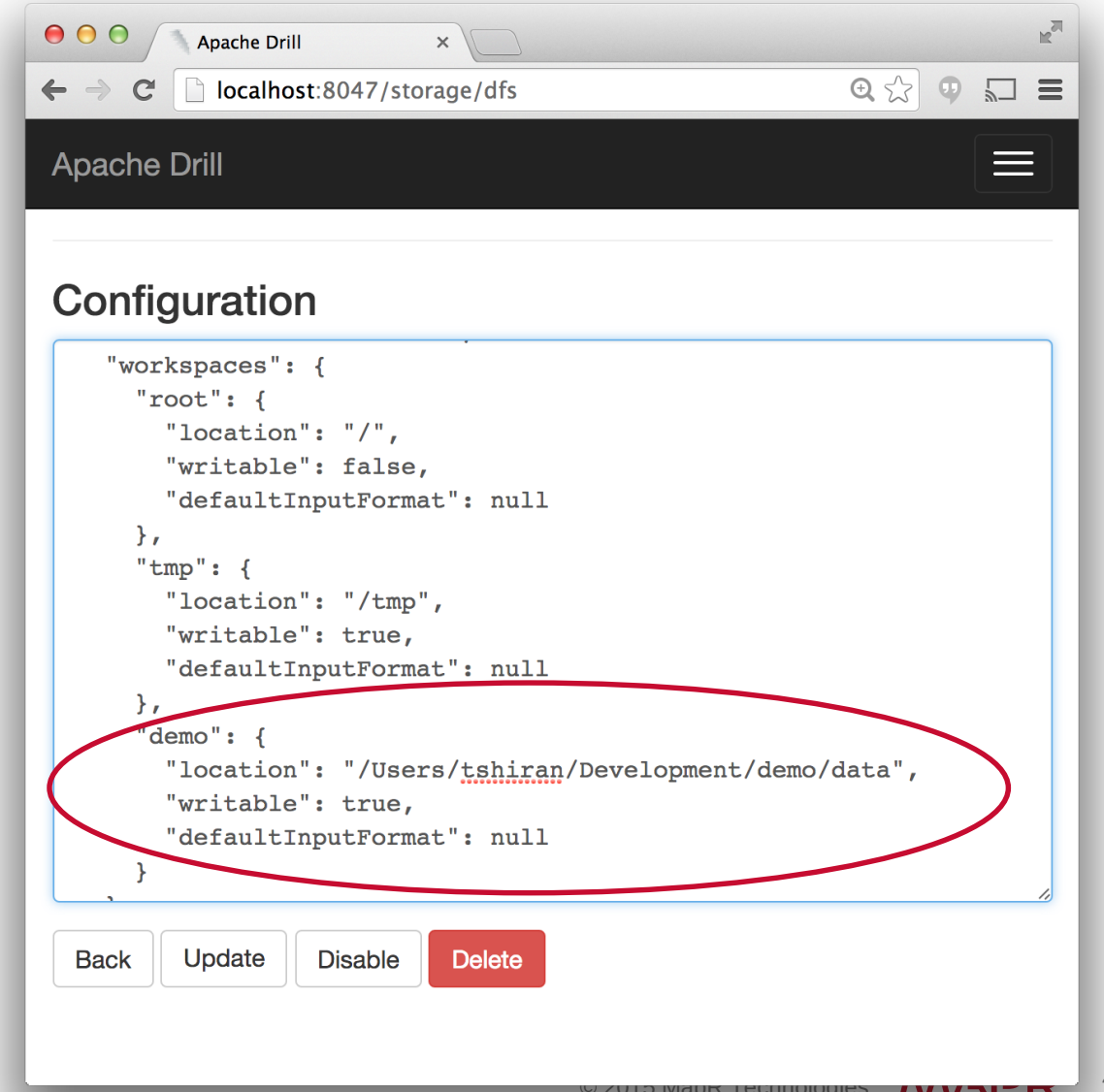
FROM **dfs.clicks.`clicks/clicks.json`**



https://cwiki.apache.org/confluence/display/DRILL/Storage+Plugin+Registration

Define Workspaces in the DFS Storage Plugin





Query Basics Data Source Workspace

workspace defines a schema to query

```
SELECT *
FROM   dfs.clicks.`clicks/clicks.json`
```

OPTIONAL

A *workspace*
- **Sub-directory**

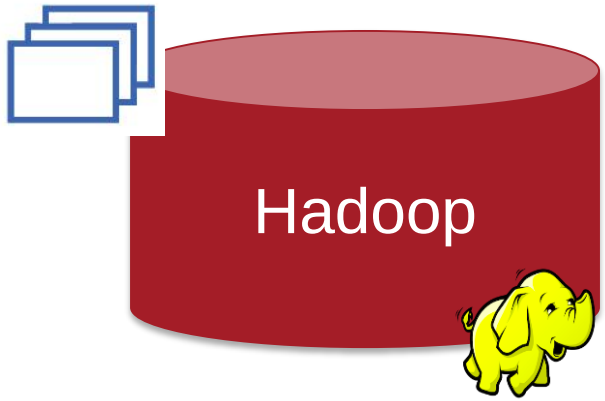
```
{
  "type" : "file",
  "enabled" : true,
  "connection" : “maprfs:///",
  "workspaces" : {
    “clicks" : {
       "location" : "/mapr/data",
       "writable" : false,
       "storageformat" : json
    }
}
```

https://cwiki.apache.org/confluence/display/DRILL/Workspaces

Data Source table

```
SELECT *
FROM   dfs.clicks.`clicks/clicks.json`
```

where you want it from



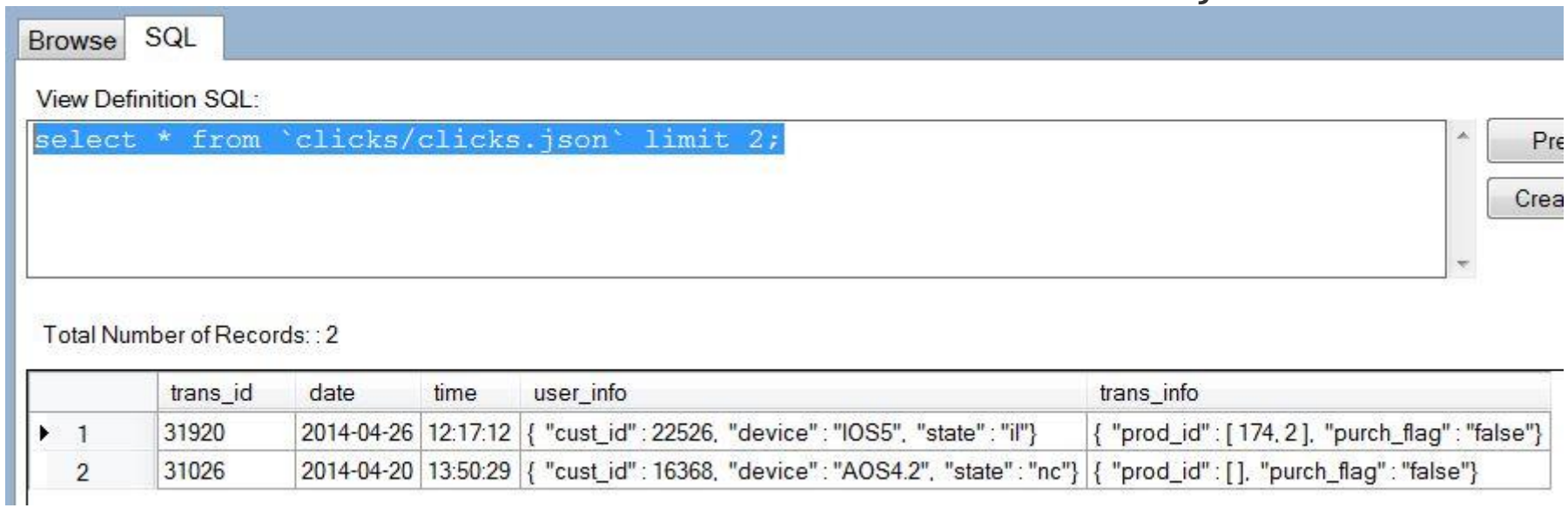
A *table*
- **pathname**
- HBase table
- Hive table

Setting the Workspace

- You can set a workspace:

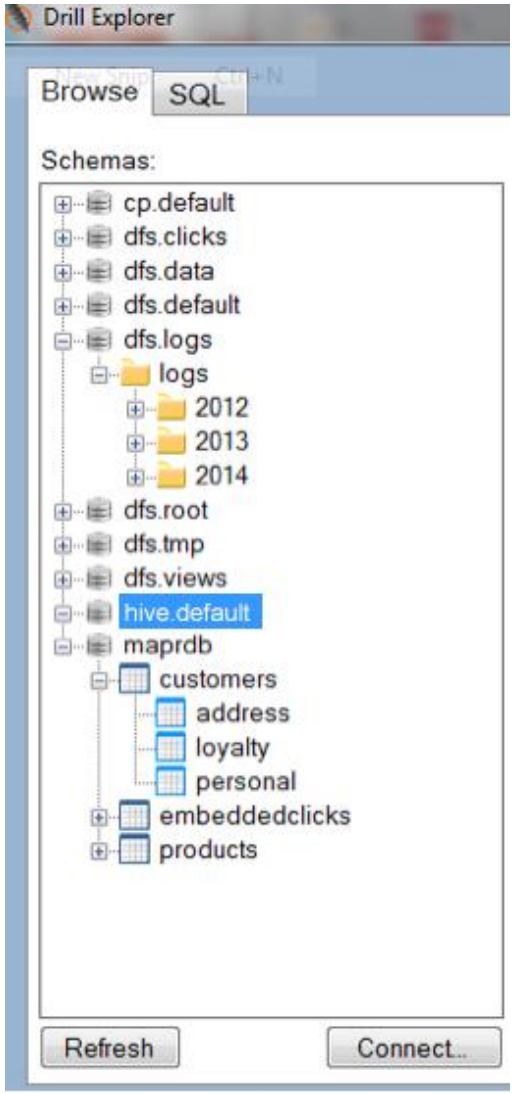
```
drill:> use dfs.clicks;
drill:> SELECT * FROM  `clicks/clicks.json`
```

```
{
  "type" : "file",
  "enabled" : true,
  "connection" : “maprfs:///",
  "workspaces" : {
    “clicks" : {
      "location" : "/mapr/data",
      "writable" : false,
      "storageformat" : json
    }
  }
}
```



	trans_id	date	time	user_info	trans_info
1	31920	2014-04-26	12:17:12	{ "cust_id" : 22526, "device" : "IOS5", "state" : "il"}	{ "prod_id" : [174, 2], "purch_flag" : "false"}
2	31026	2014-04-20	13:50:29	{ "cust_id" : 16368, "device" : "AOS4.2", "state" : "nc"}	{ "prod_id" : [ ], "purch_flag" : "false"}

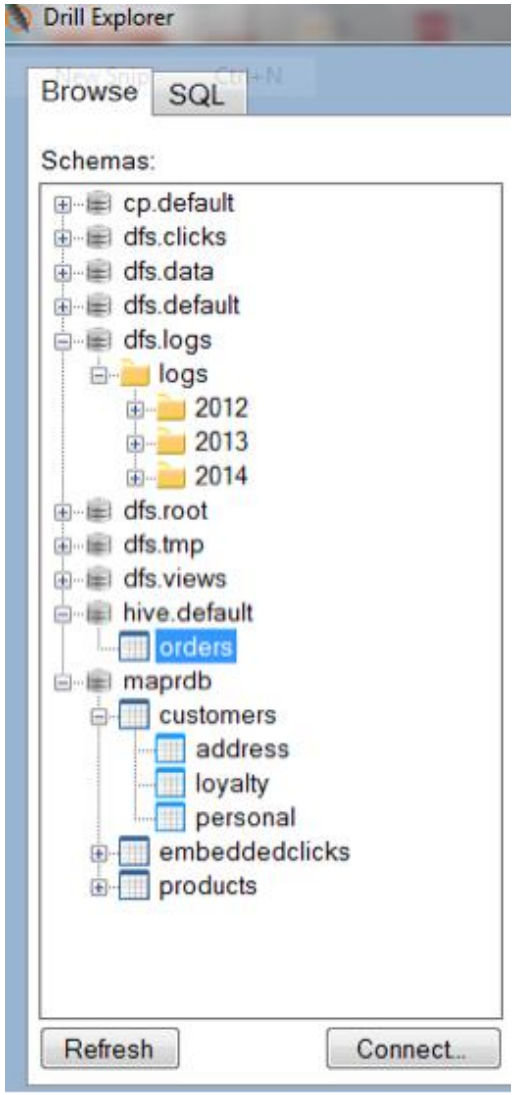
Define Data Interaction & Schema Detection



Storage system type

```
SELECT `month`, SUM(order_total)
as sales FROM hive.orders GROUP
BY `month` ORDER BY sales desc;
```

Define Data Interaction & Schema Detection



Storage system type

```
SELECT `month`, SUM(order_total)
as sales FROM hive.orders GROUP
BY `month` ORDER BY sales desc;
```

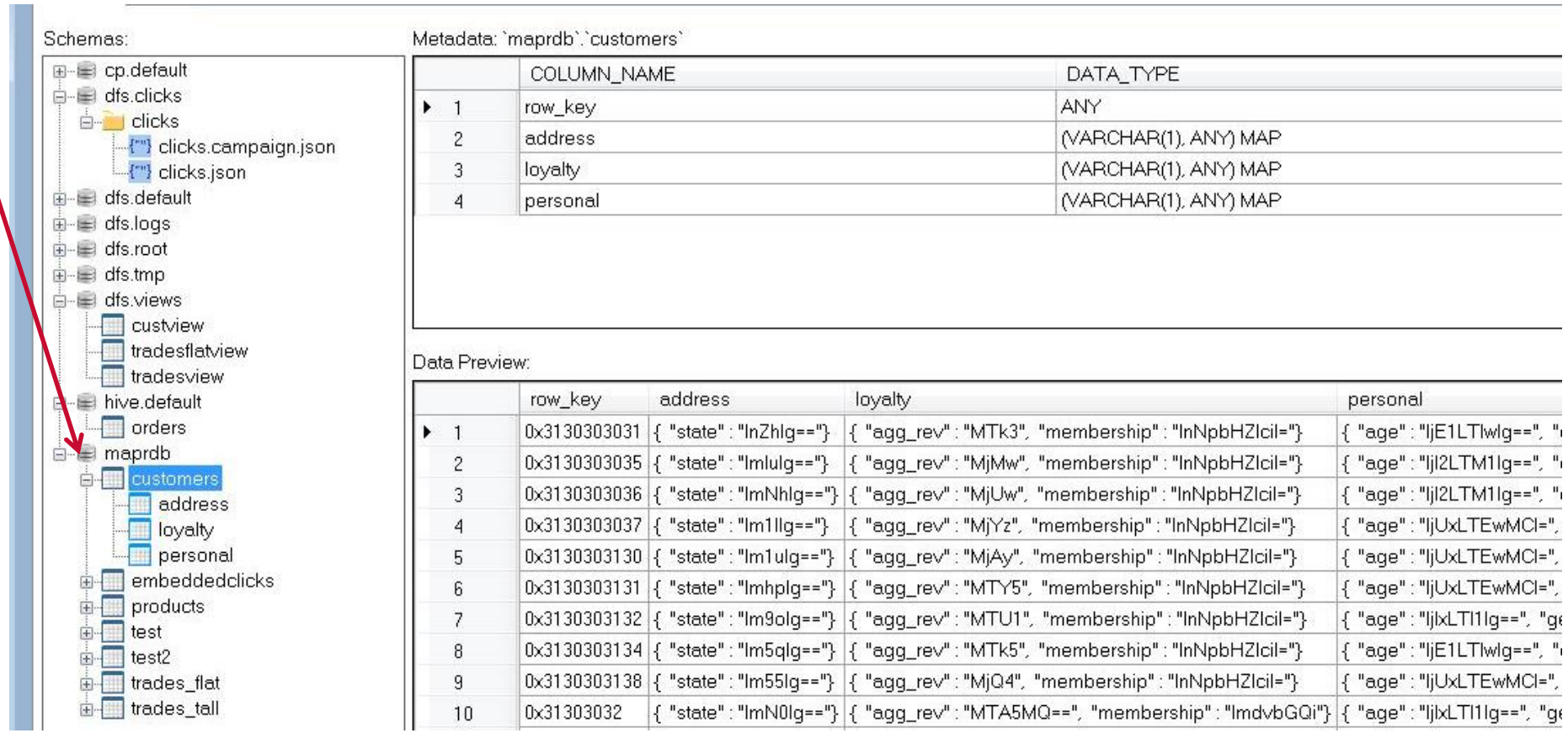
Table Name

Complex Query Examples

```
SELECT cast(cust.personal.name as varchar(40)) as cust_name,
clk.prod_id, clk.`date`
FROM maprdb.customers as cust
```

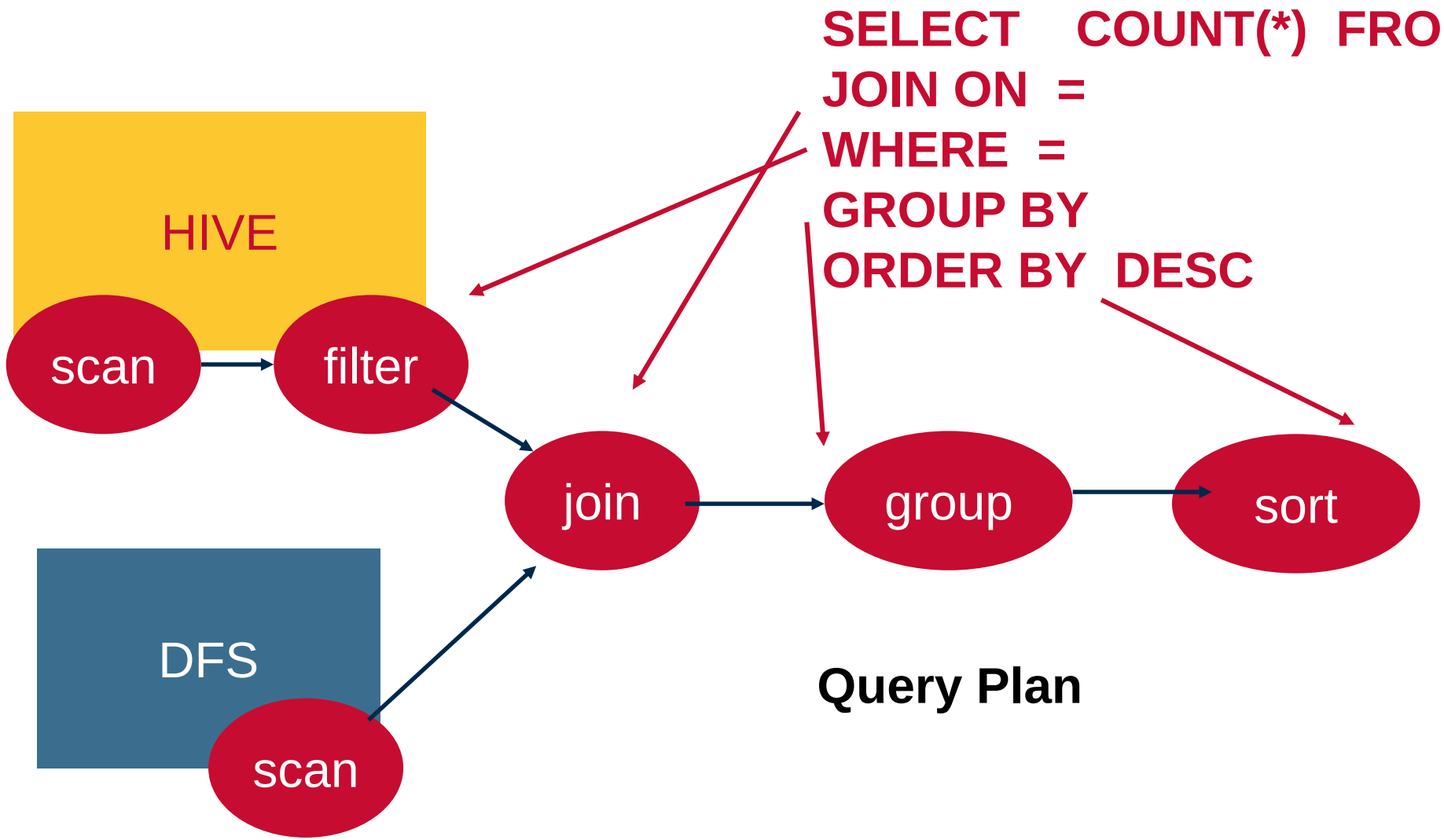
Storage system type

Table Name



Drill Architecture and flow

SQL is the WHAT, now we will look at the HOW



Understanding the Explain plan

Business Scenario

- Understanding the Explain plan for this query
  - Which device do customers who made orders use ?

```
SELECT t.user_info.device device, t.user_info.cust_id cust_id, o.order_id
FROM dfs.clicks.`clicks/clicks.json` t, hive.orders o
WHERE t.user_info.cust_id=o.cust_id ;
```

JOIN ON

id	date	cust_id	device	state	prod_id	purch_flag
67212	June	10001	“iOS5”	“ca”	[174,2]	false

Web log clicks

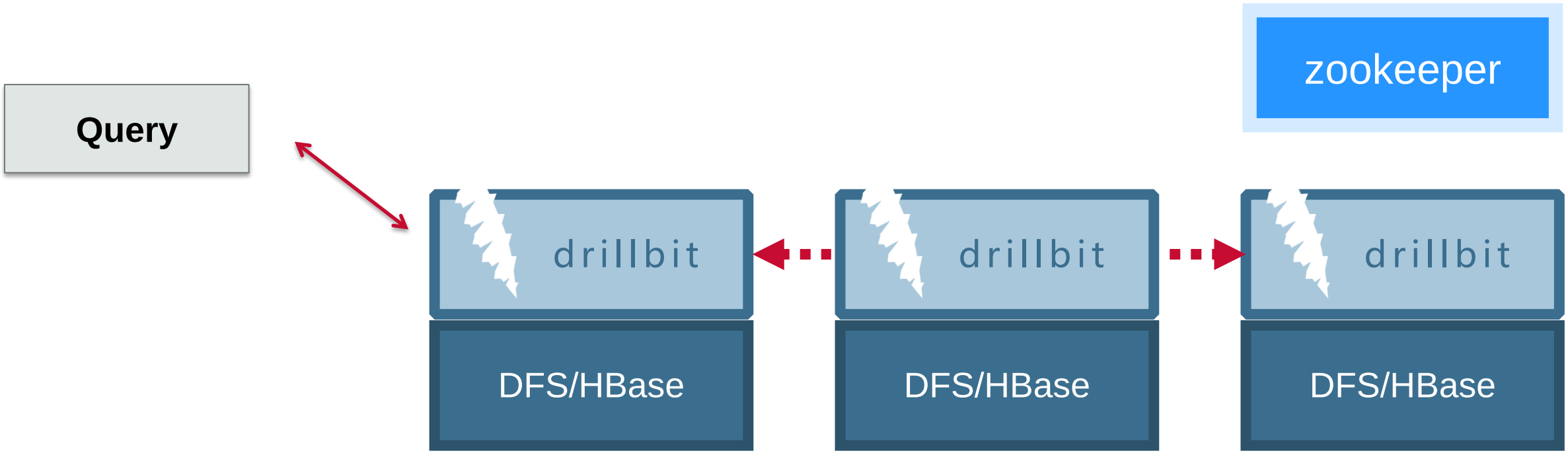
order_id	month	cust_id	state	Prod_id	total
67212	June	10001	ca	909	13
70302	May	10004	ga	420	11

HIVE orders

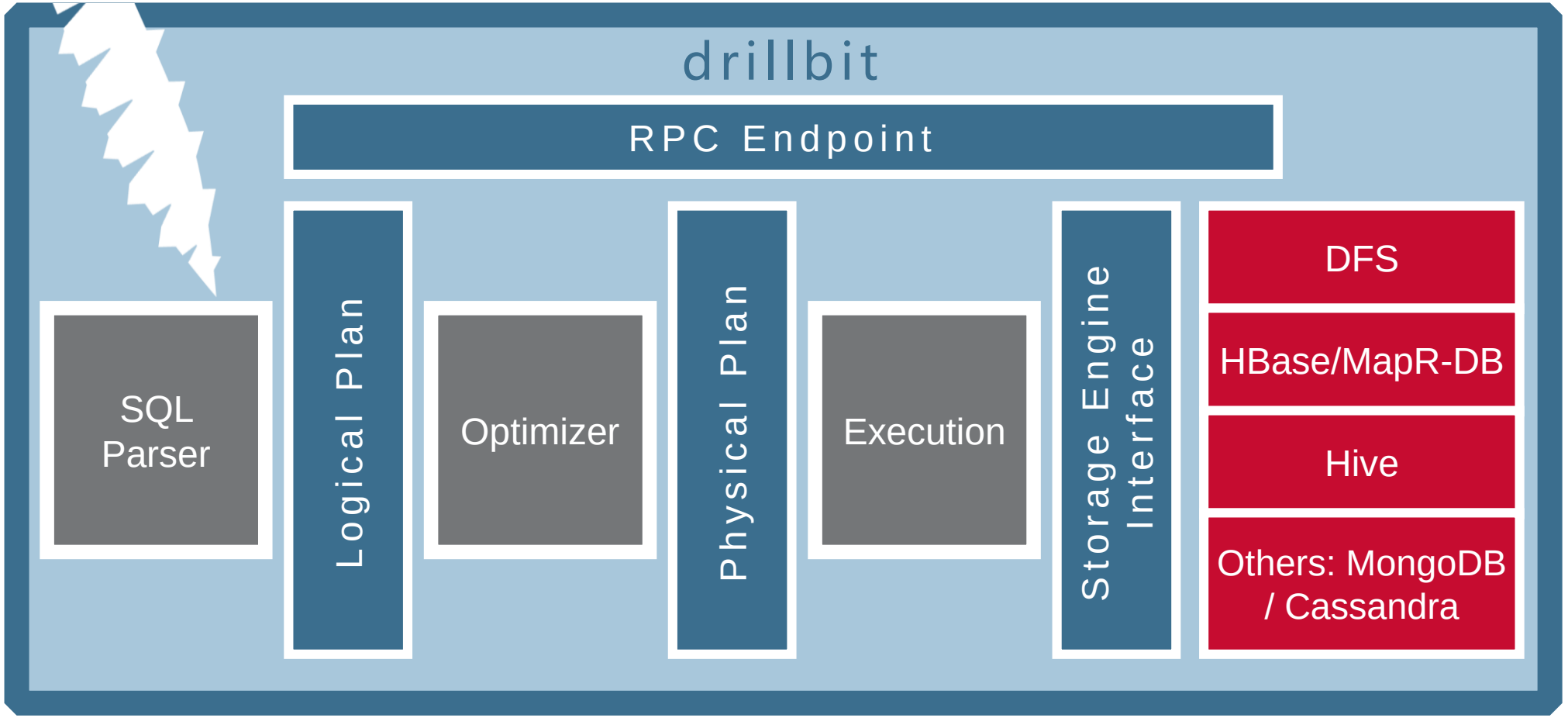
How Does It Work?

```
SELECT device, cust_id, order_id
FROM  clicks.json t, hive.orders o
WHERE t.cust_id=o.cust_id
```

- "Drillbits" run on each node
  - designed to maximize data locality
- Coordination, query planning, optimization, scheduling, and execution are distributed

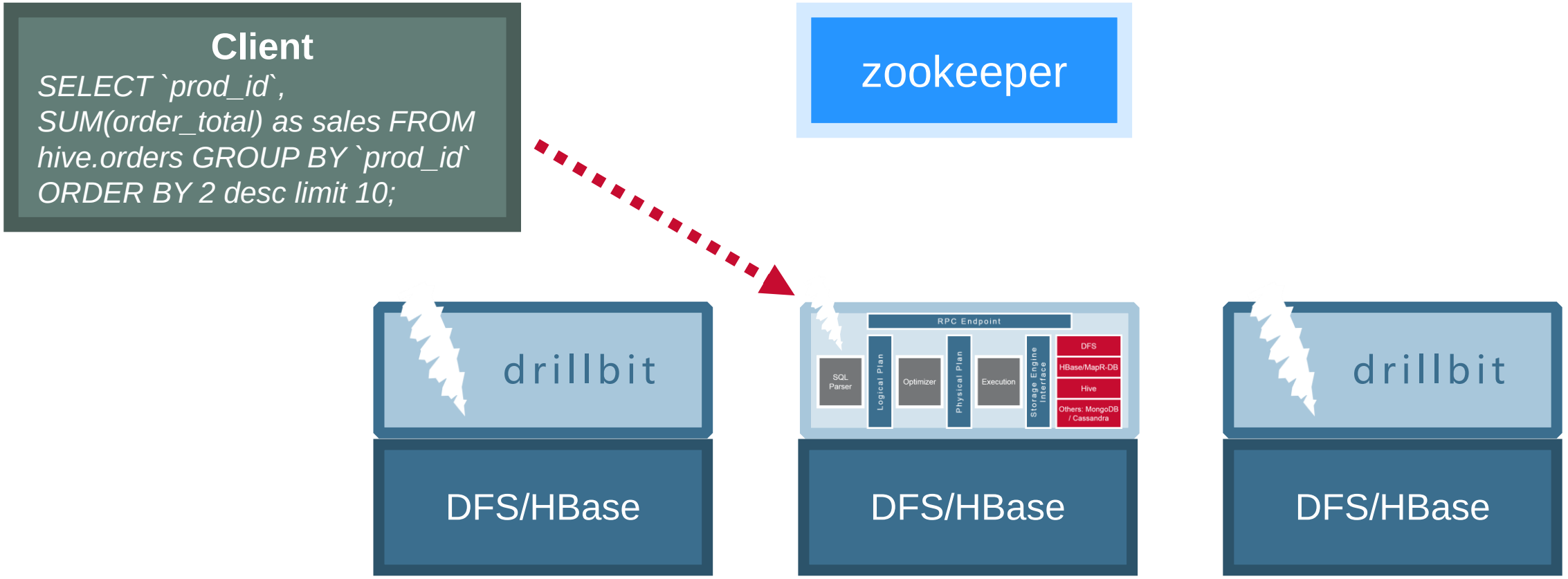


Core Modules within a Drillbit



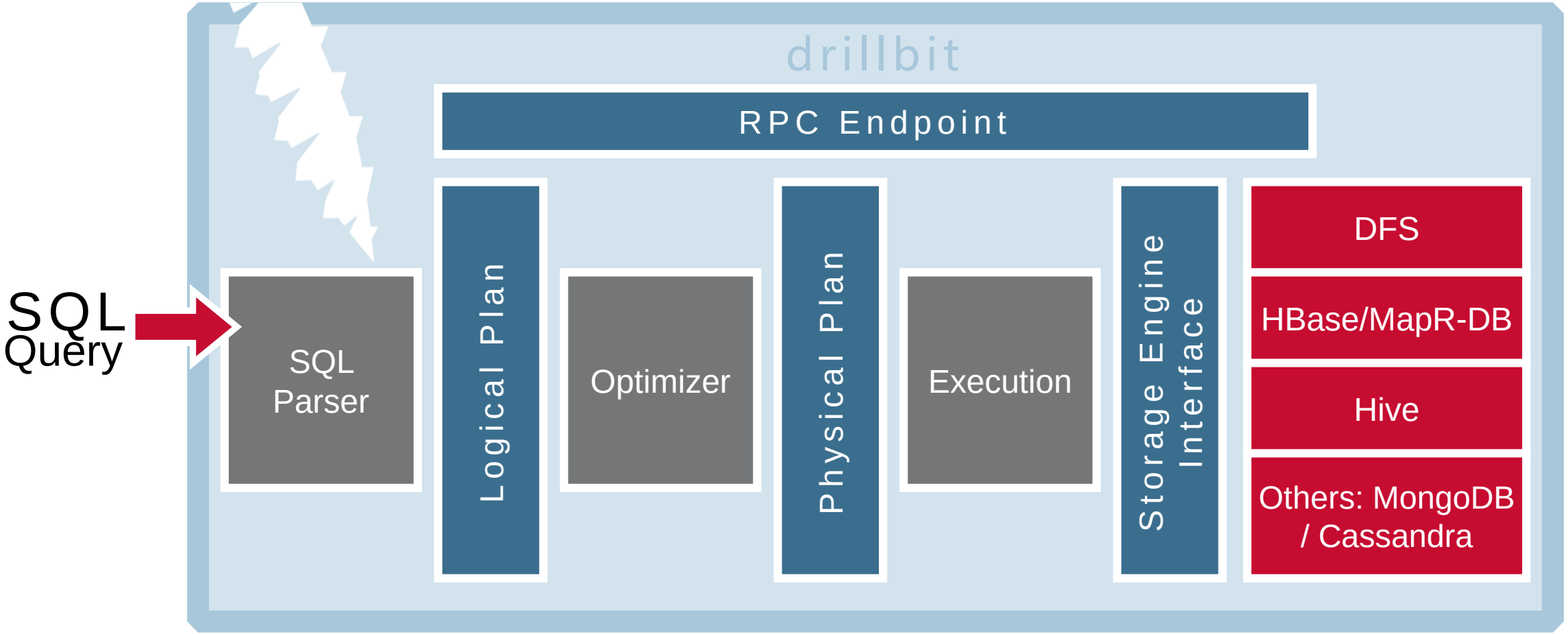
Query Execution

1. Query comes to any Drillbit (JDBC, ODBC, CLI)



Query Execution Plan

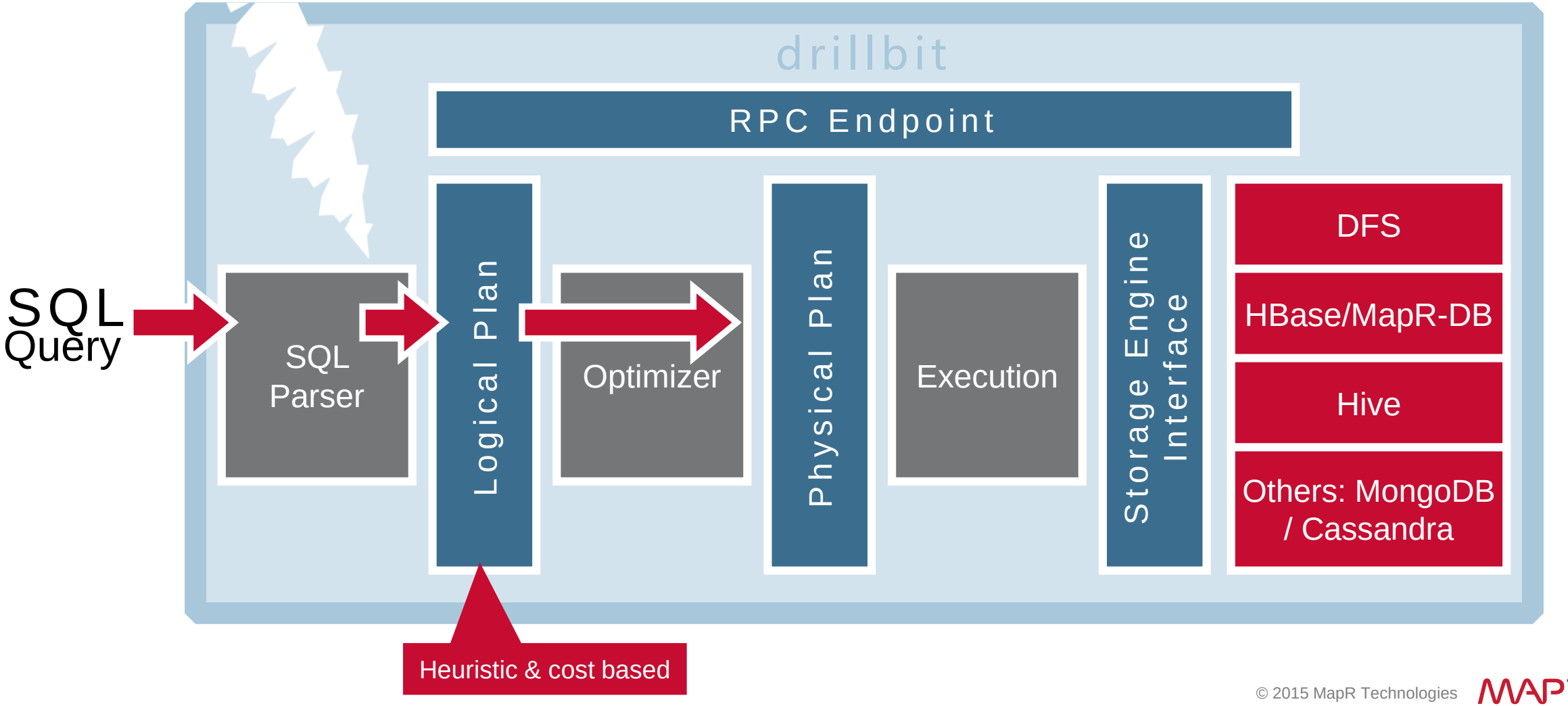
2. Drillbit generates execution plan based on query optimization & locality



SQL What we want to do (analyst friendly)

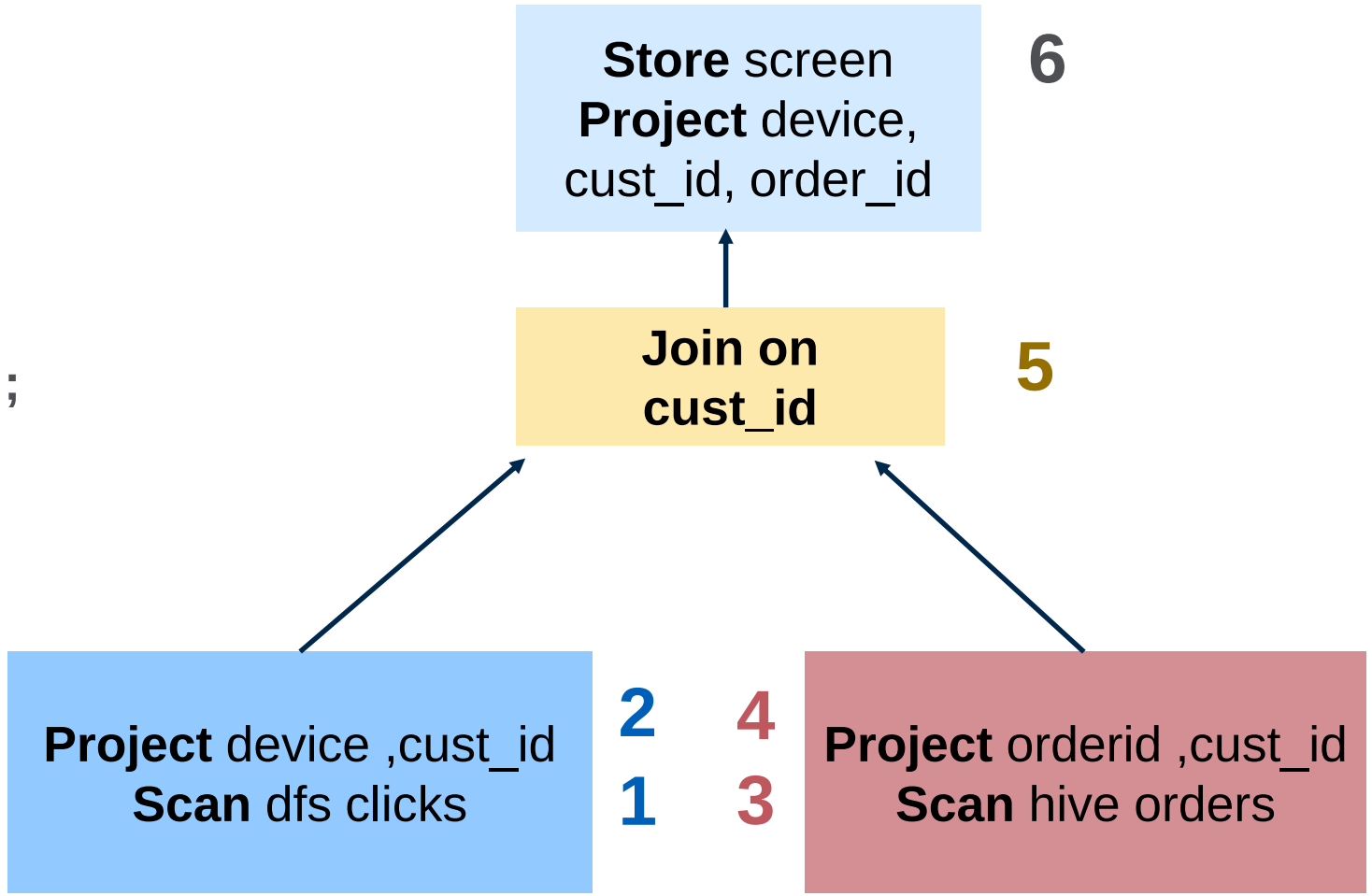
Query Execution Plan

Logical Plan: What we want to do (language agnostic, computer friendly)



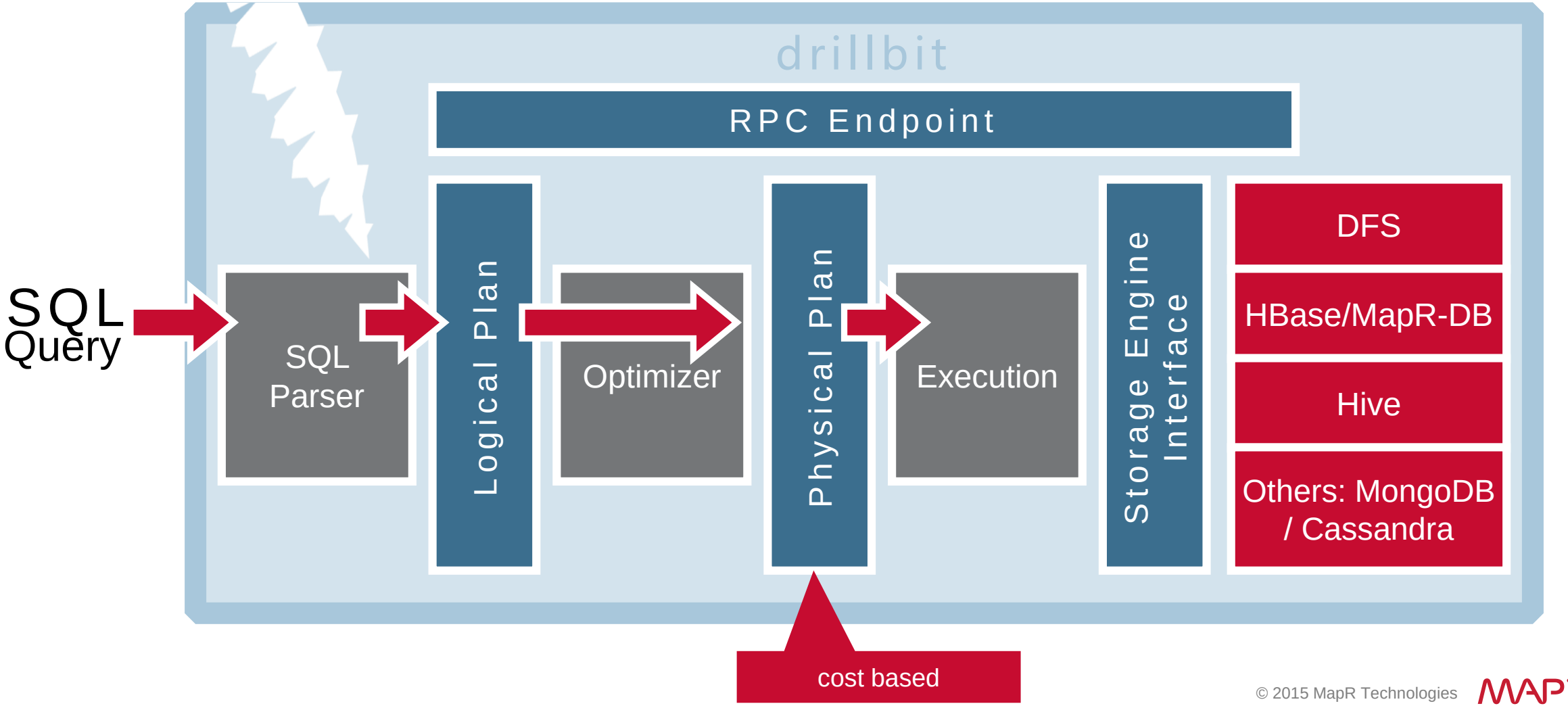
Logical Plan

```
SELECT t.user_info.device
device, t.user_info.cust_id
cust_id, o.order_id
FROM
dfs.clicks.`clicks/clicks.json`
t, hive.orders o
WHERE
t.user_info.cust_id=o.cust_id ;
```



Query Execution Plan

Physical Plan: How we want to do it (the best way we can tell)



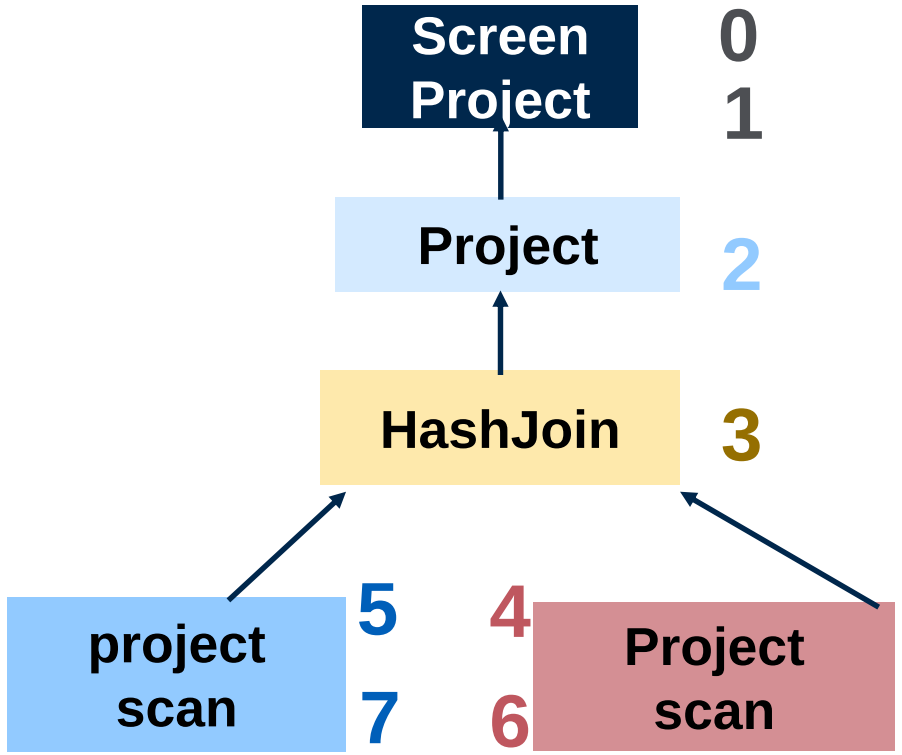
Drill Generates a Physical Plan, based on available operations

- A Physical Plan is composed of major Fragments
- Each operator has a particular **OperatorId.**
- You can capture a JSON representation of this plan, modify manually and submit

```
> !set maxwidth 10000
> explain plan for select * from customer limit 5;
…
00-00    Screen
00-01      Project
00-02        Project
00-03          HashJoin
00-04            Project
00-05              Scan
00-06            Project
00-07              Scan
```

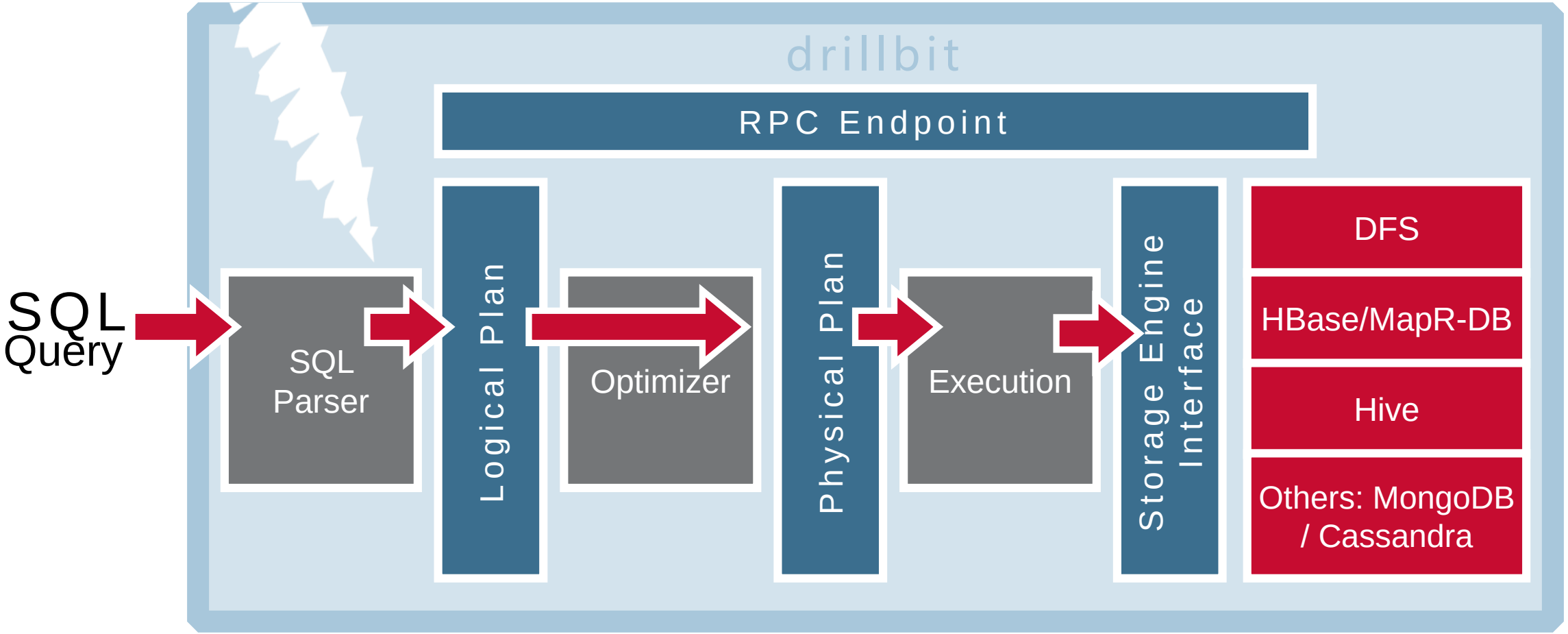
Major Fragment Id

Operator Id



Query Execution Plan

Execution Plan (fragments): Where we want to do it

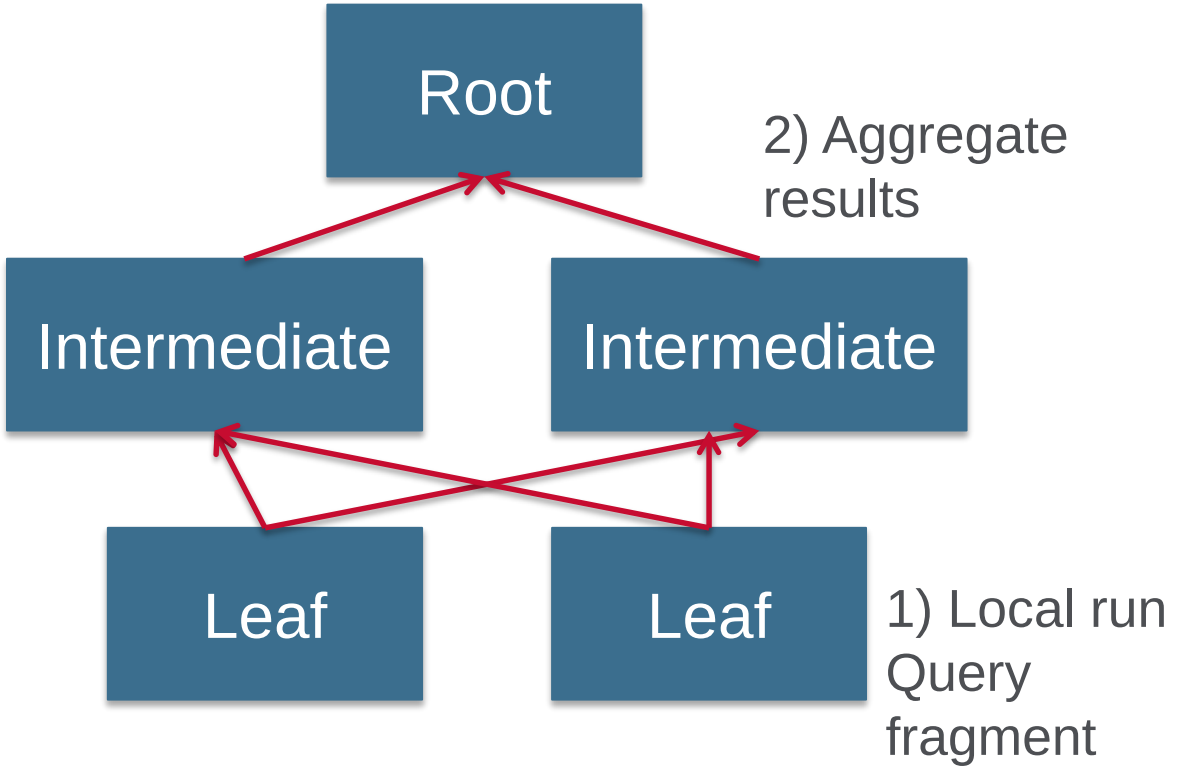


Plan fragments sent to drill bits

Query Execution Plan tree

Each execution plan has:

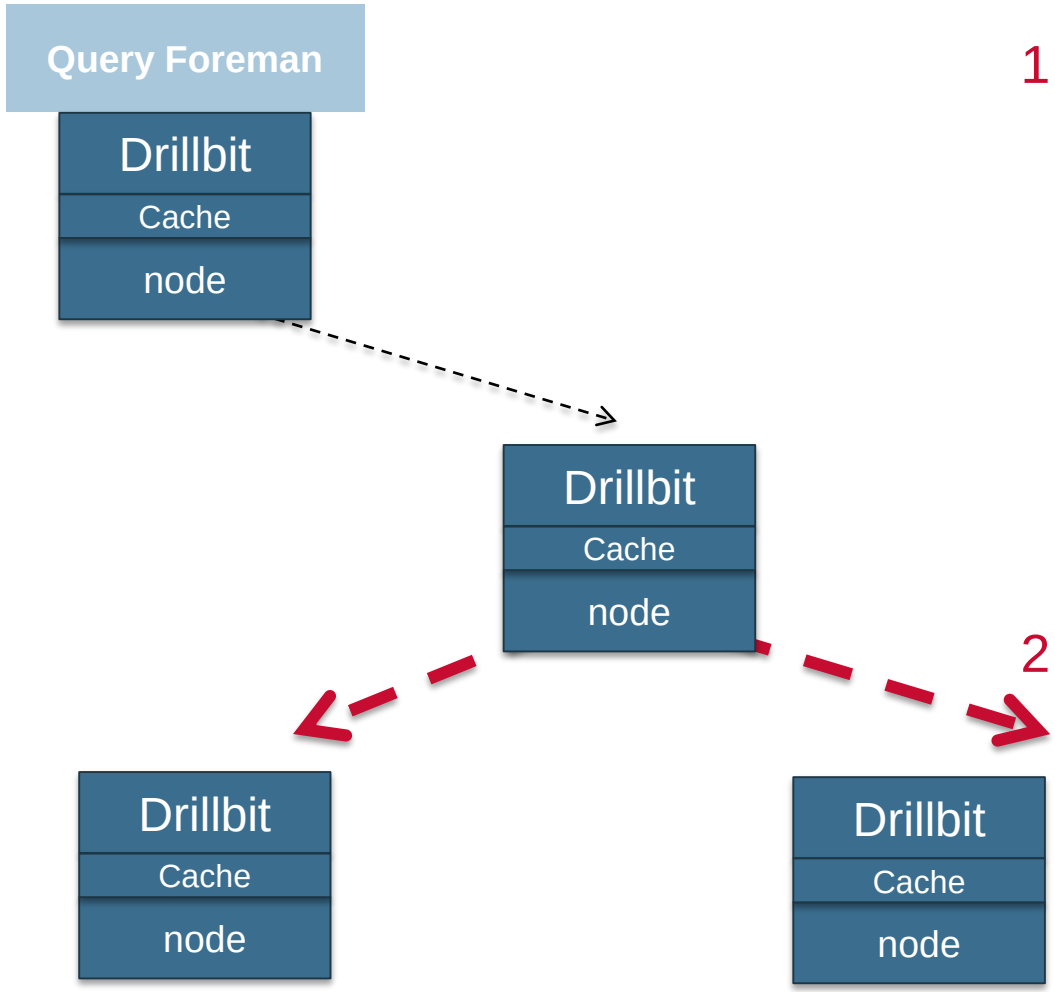
- One **root** fragment
1. **Leaf fragments**
   - **1st** to run
   - Leaf nodes **run query** fragment
- Intermediate fragments



Foreman manages execution

SQL Query →

```
SELECT device, cust_id, order_id
FROM  clicks.json t, hive.orders o
WHERE t.cust_id=o.cust_id ;
```



1. Originating Drillbit manages query execution

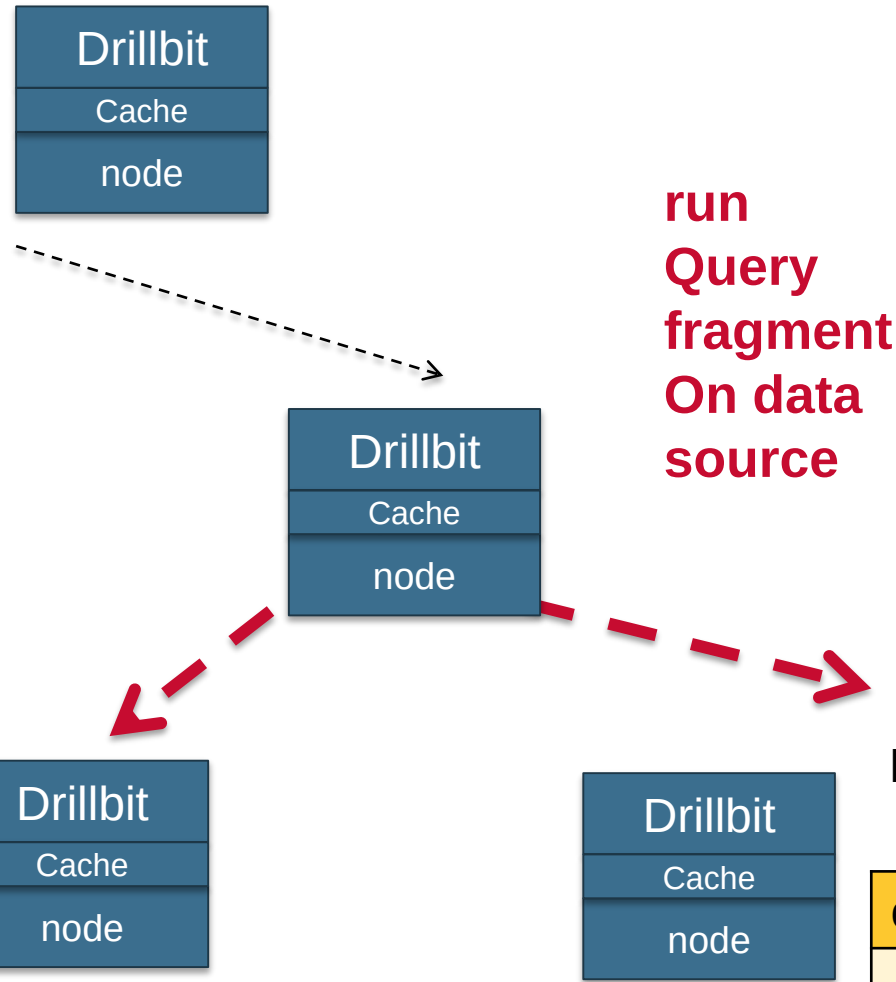
2. leafs run query.

cust_id	device
10001	“iOS5”
10004	AOS4.2

order_id	cust_id
67212	10001
70302	10004

Leaf fragments pass back results

Leaf nodes run query fragment on data source and send results up the tree




```
SELECT device, cust_id, order_id
FROM  clicks.json t, hive.orders o
WHERE t.cust_id=o.cust_id ;
```

Project device ,cust_id
Scan dfs clicks

cust_id	device
10001	“iOS5”
10004	AOS4.2

Project orderid ,cust_id
Scan hive orders

order_id	cust_id
67212	10001
70302	10004

Columnar Storage

- Drill optimizes for Columnar Storage
  - Parquet http://parquet.incubator.apache.org/
- Only retrieve columns participating in query

Logical table representation

a	b	c
a1	b1	c1
a2	b2	c2
a3	b3	c3
a4	b4	c4
a5	b5	c5

Row layout

a1	b1	c1	a2	b2	c2	a3	b3	c3	a4	b4	c4	a5	b5	c5

Column layout

a1	a2	a3	a4	a5	b1	b2	b3	b4	b5	c1	c2	c3	c4	c5

Columnar Execution

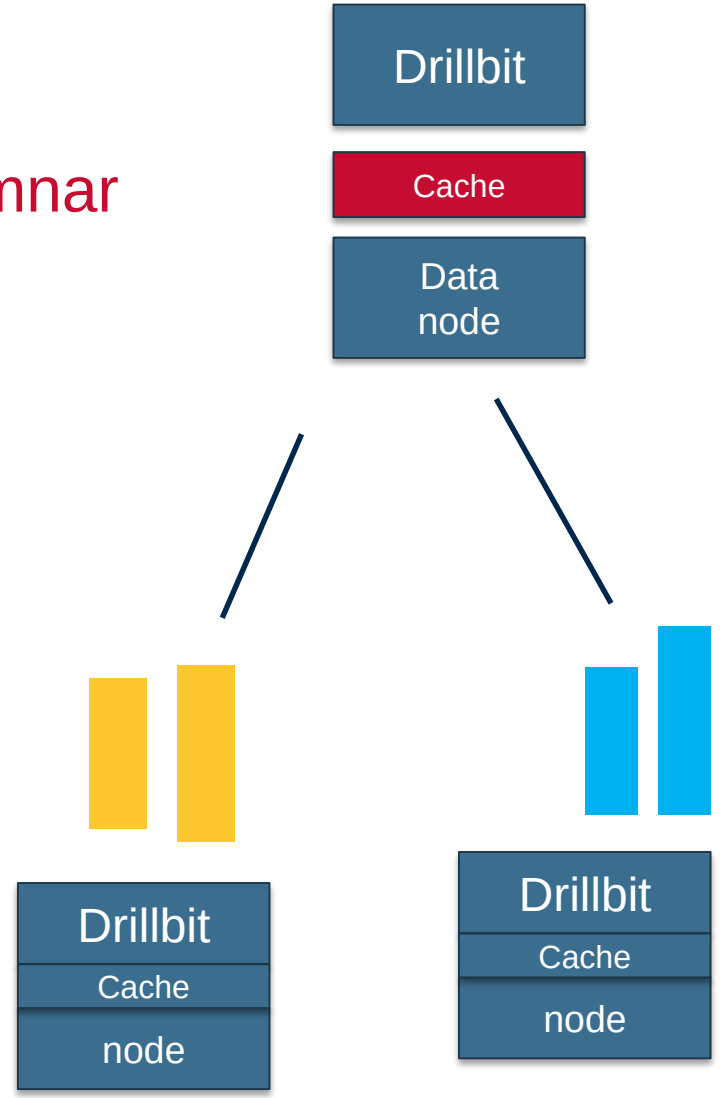
- Drill optimizes for Columnar Execution
  - in-memory data model that is hierarchical and columnar

order id	Cust_id
101	abc
102	def
104	ghi
105	jkl
108	mno
112	pqr

Data node

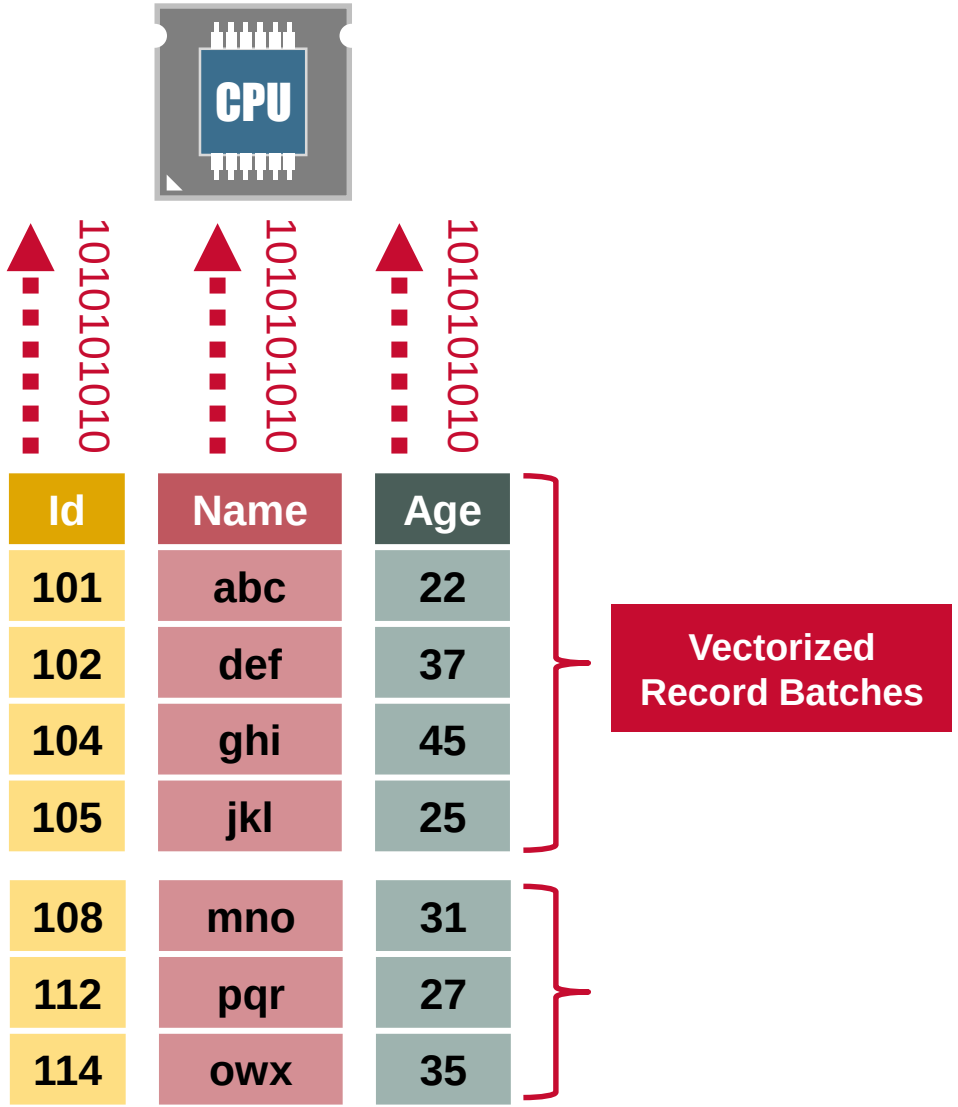
devic	Cust_id
101	abc
102	def
104	ghi
105	jkl
108	mno
112	pqr

Data node



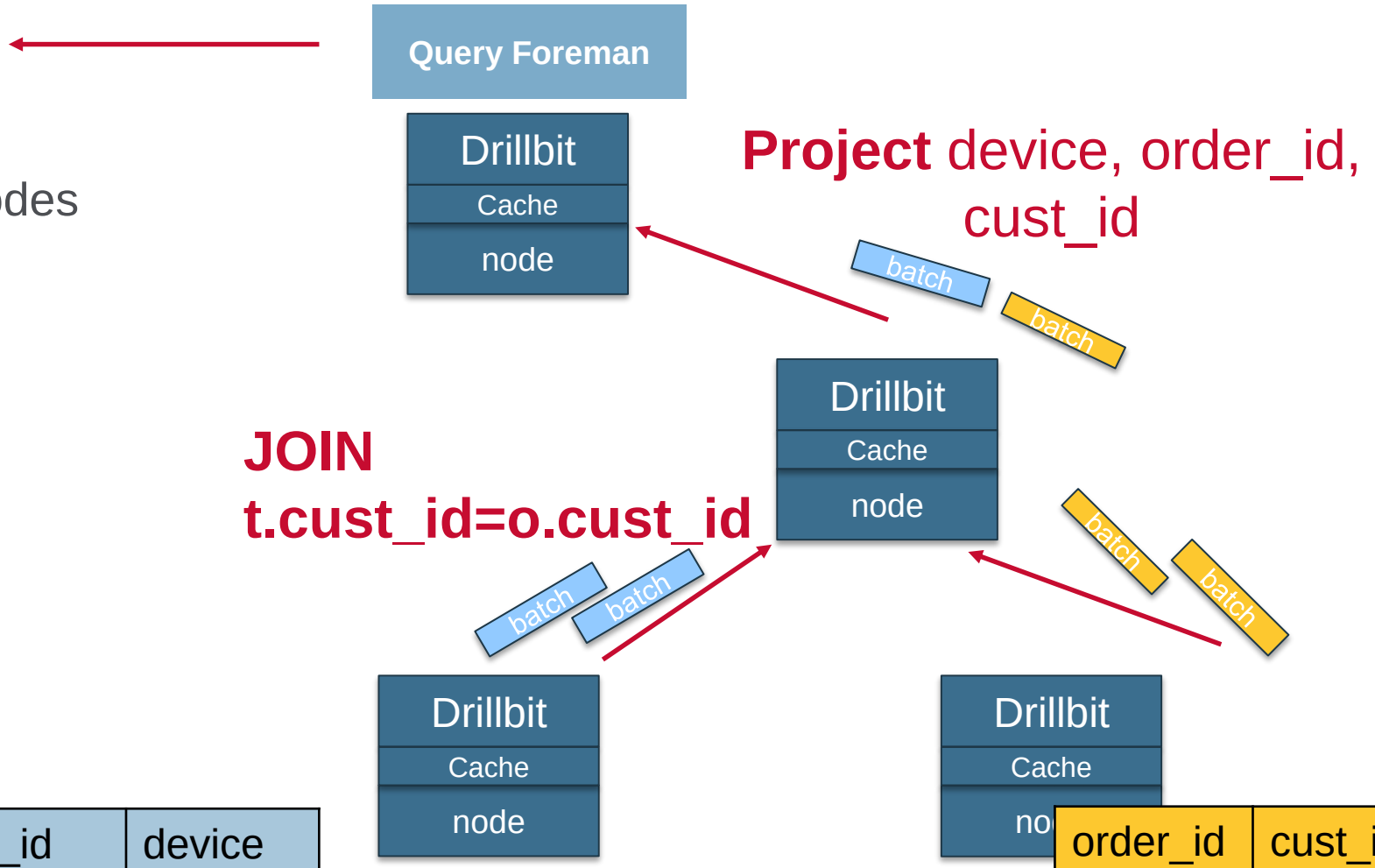
Vectorization

sets of columns values



Data is returned, filtered

- Intermediate fragments **start** when they **receive** data
- **aggregates results** from leaf nodes and
  - send results up the tree



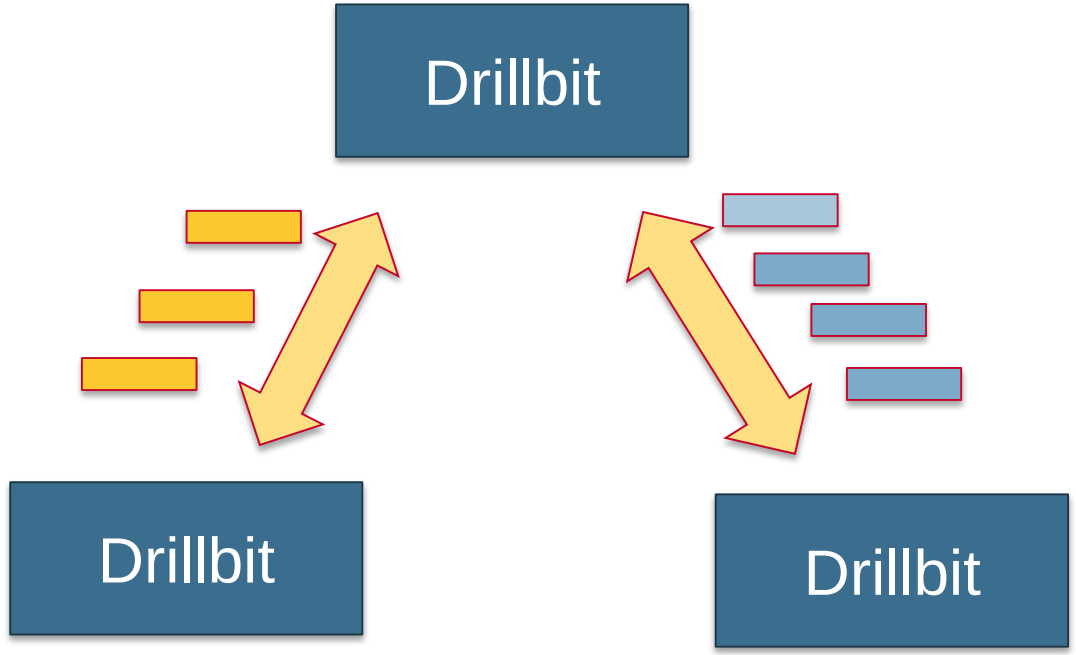

```
SELECT device, cust_id, order_id
FROM  clicks.json t, hive.orders o
WHERE t.cust_id=o.cust_id ;
```

cust_id	device
10001	“iOS5”
10004	AOS4.2

order_id	cust_id
67212	10001
70302	10004

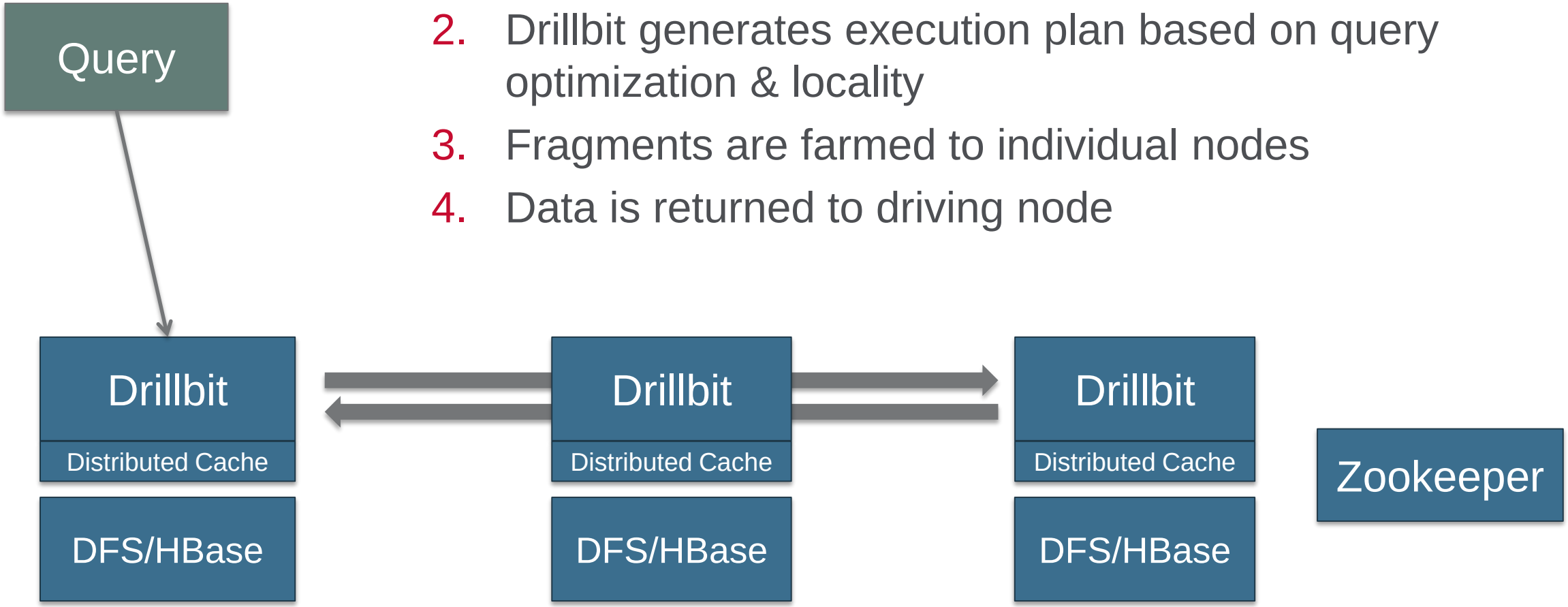
Pipelining

- Record batches are pipelined between nodes
  - Operators works on a batch
- query execution happens in-memory
  - (as much as possible)



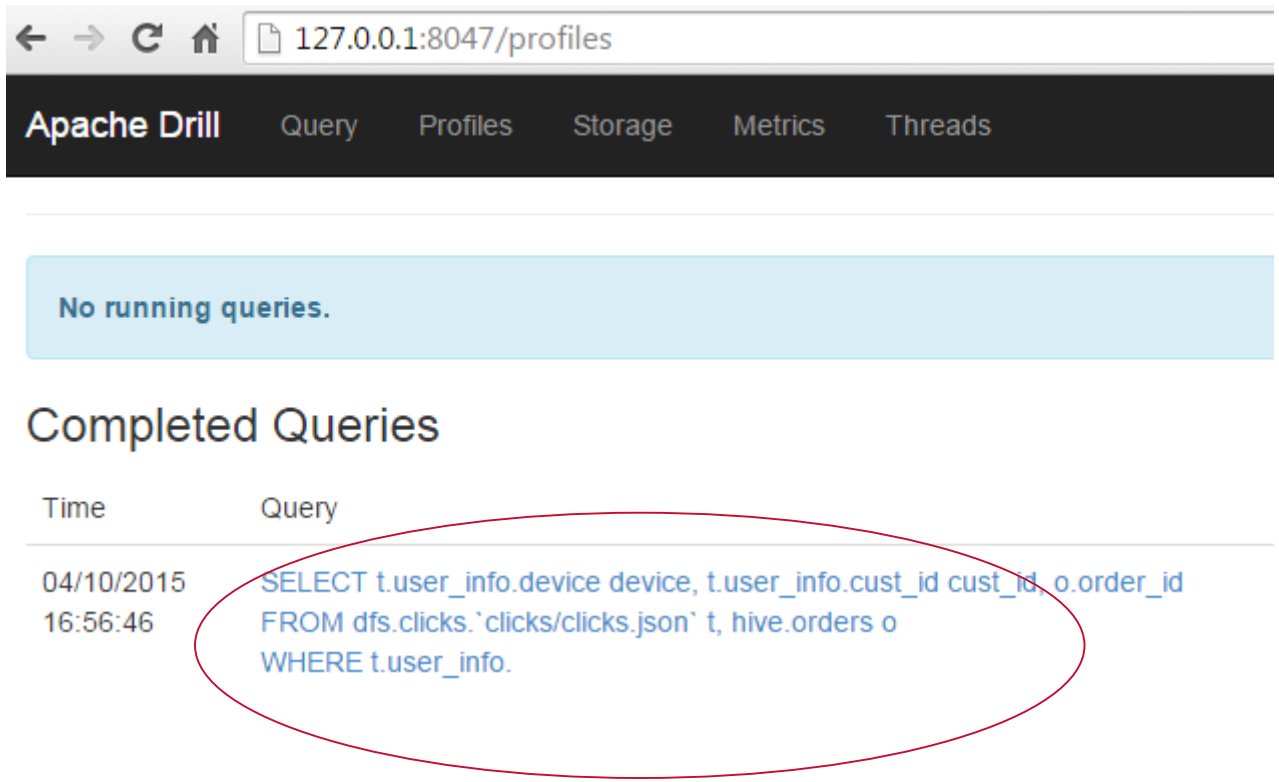
Drill query execution at a glance

1. Query comes to any Drillbit
2. Drillbit generates execution plan based on query optimization & locality
3. Fragments are farmed to individual nodes
4. Data is returned to driving node

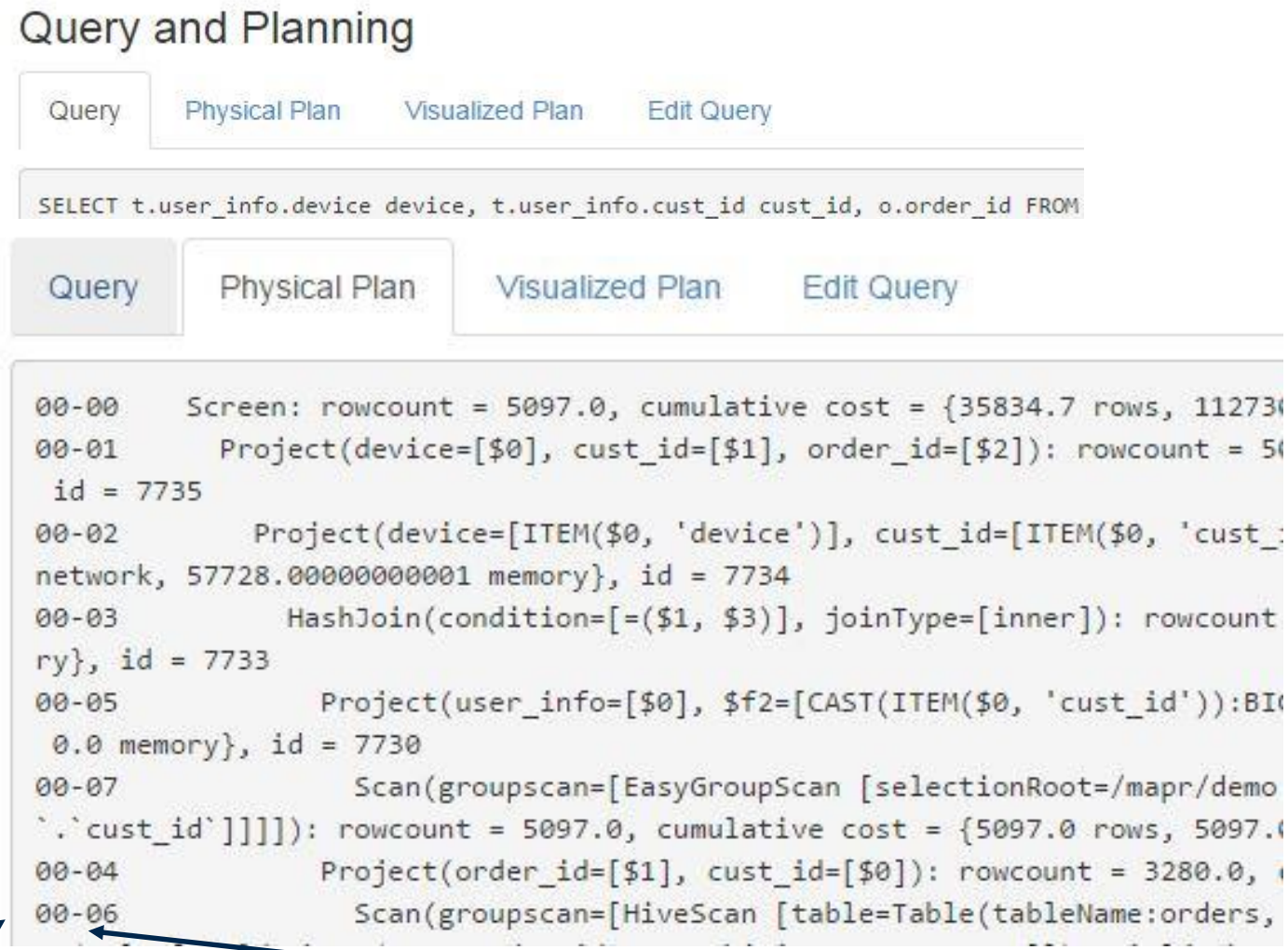


Drill Query Profiles

- For each query, Drill generates a unique query id
- Profiles are saved under this identifier



WebUI: Query and Physical Plan



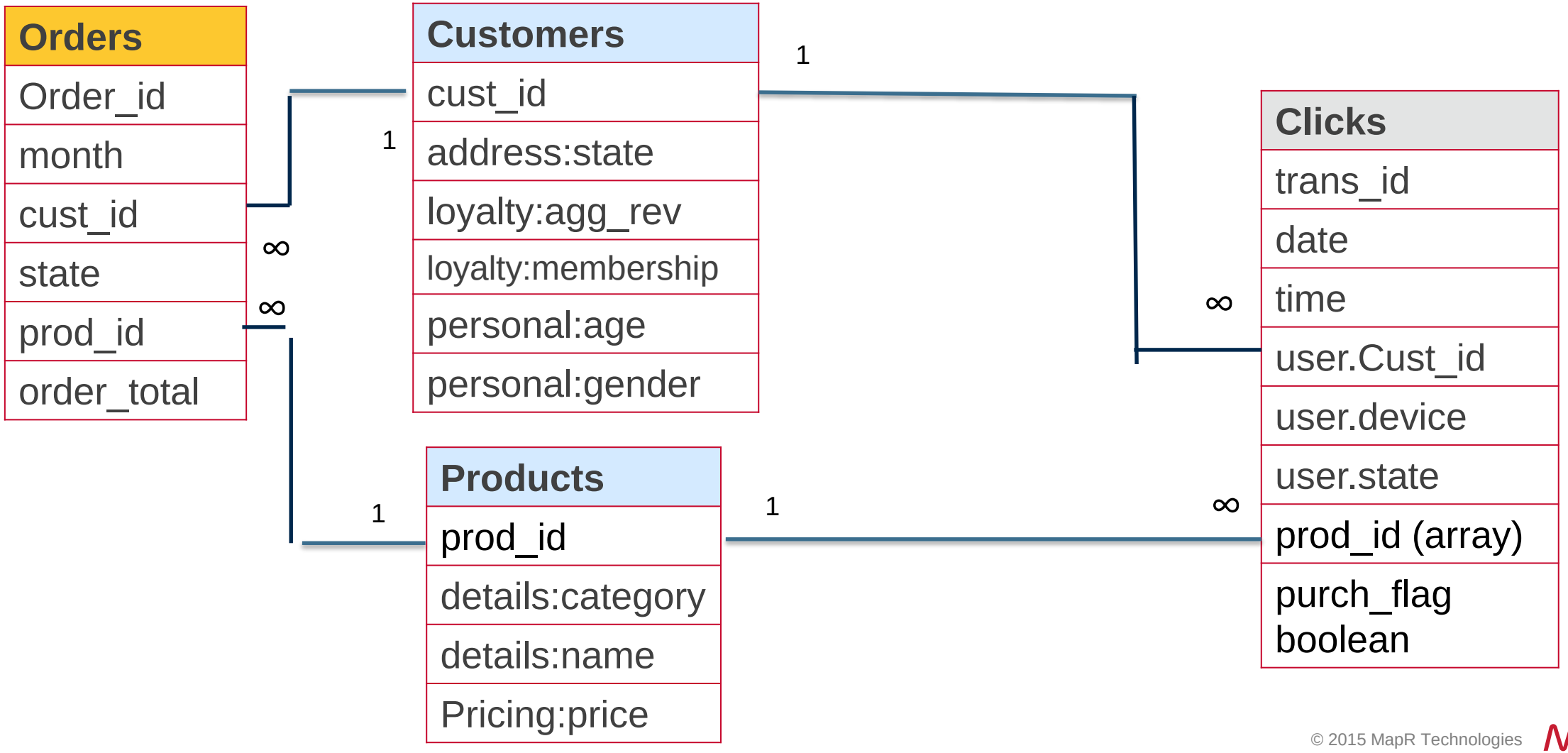
Major Fragment Id Operator Id

WebUI: Visualized Plan

Combining and Aggregating Data from Files, Hive, and HBase using Drill and ANSI SQL

- GROUP BY
- UNION
- JOIN
- SUBQUERY
- CAST
- CREATING A VIEW
- CREATING A TABLE

High Level “Relations” between Retail data



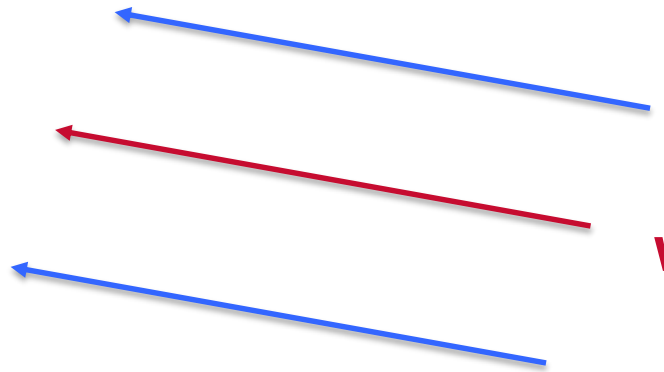
SQL Basic Query Syntax

```
SELECT select_list
[ FROM table_source ]
[ WHERE search_condition ]
[ GROUP BY group_by_expression ]
[ HAVING search_condition ]
[ ORDER BY order_expression [ ASC | DESC ] ]
```

ANSI SQL with no modifications

Query Basics Select

```
SELECT cust_id
FROM  hive.`orders`
WHERE state='ca'
```



Selects columns

where you want it from

Filters records to return

Aggregating Data from Files, Hive, and HBase using Drill and ANSI SQL

- Aggregate Functions, GROUP BY

UNION

JOIN

SUBQUERY

CAST

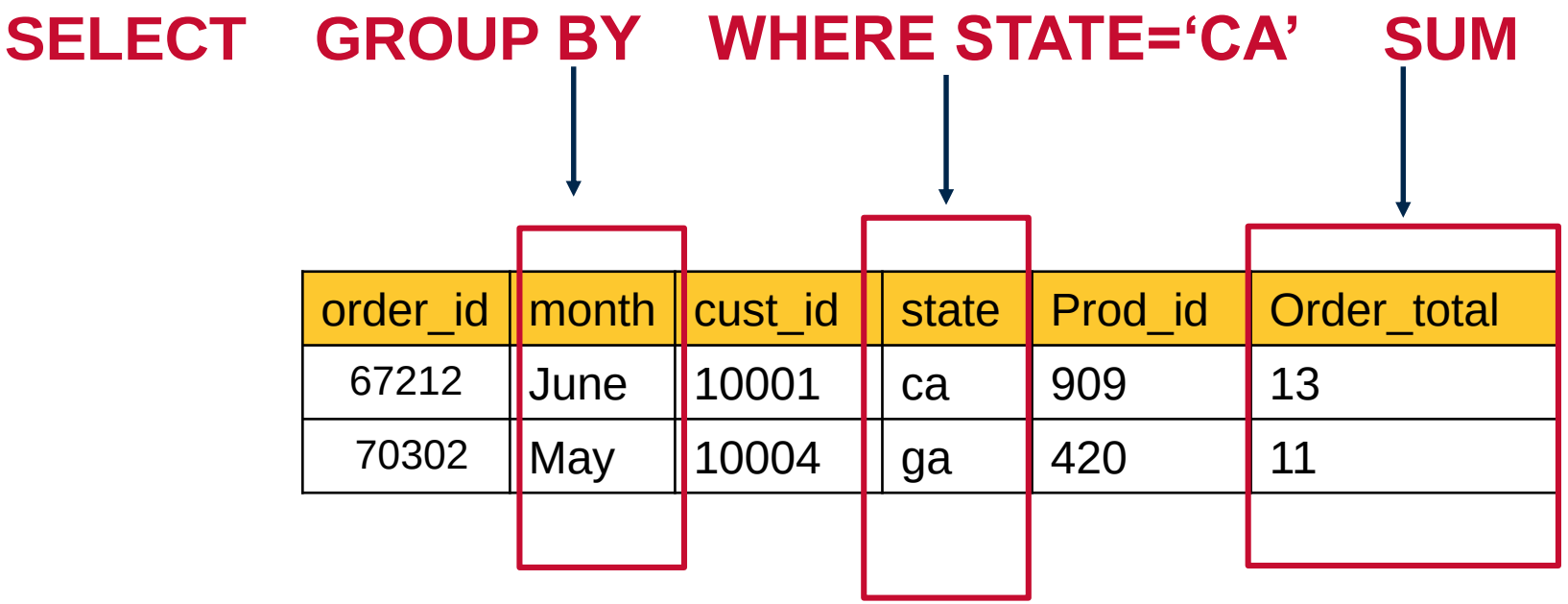
CREATING A VIEW

CREATING A TABLE

Summarizing Data with Aggregate Functions, GROUP BY

Business Scenario

- What are the Sales total by month for California ?



order_id	month	cust_id	state	Prod_id	Order_total
67212	June	10001	ca	909	13
70302	May	10004	ga	420	11

Query Basics Group by

```
SELECT  `month`, SUM(order_total) AS sales
FROM  hive.`orders`
WHERE state='ca'
GROUP BY `month`
ORDER BY 2 desc
```

ALIAS

Total Number of Records: : 10

	month	sales
1	May	119586
2	June	116322
3	April	101363
4	March	99540
5	July	90285
6	October	80090
7	August	71255
8	February	63527
9	September	45552
10	January	38799

Query from orders for sales totals by month

Query Basics: Aggregate Functions

```
avg(expression)
count(*)
count([DISTINCT] expression)
max(expression)
min(expression)
sum(expression)
```

Standard SQL clauses work in the same way in Drill queries as in relational database queries

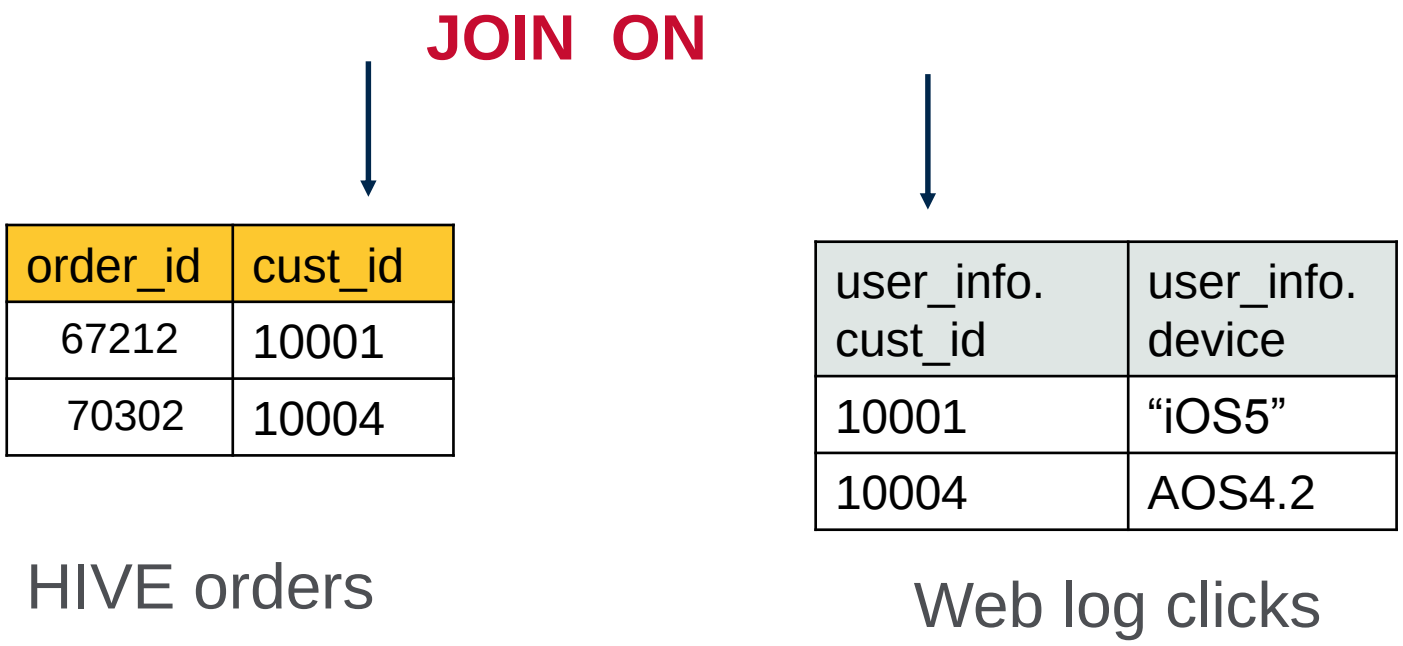
Use Drill with ANSI SQL

- GROUP BY
- UNION
- Combining Data from Files, Hive, and HBase using Drill and ANSI SQL
- JOIN
- SUBQUERY
- CAST
- CREATING A VIEW
- CREATING A TABLE

Combining Data from multiple tables with JOIN

Business Scenario

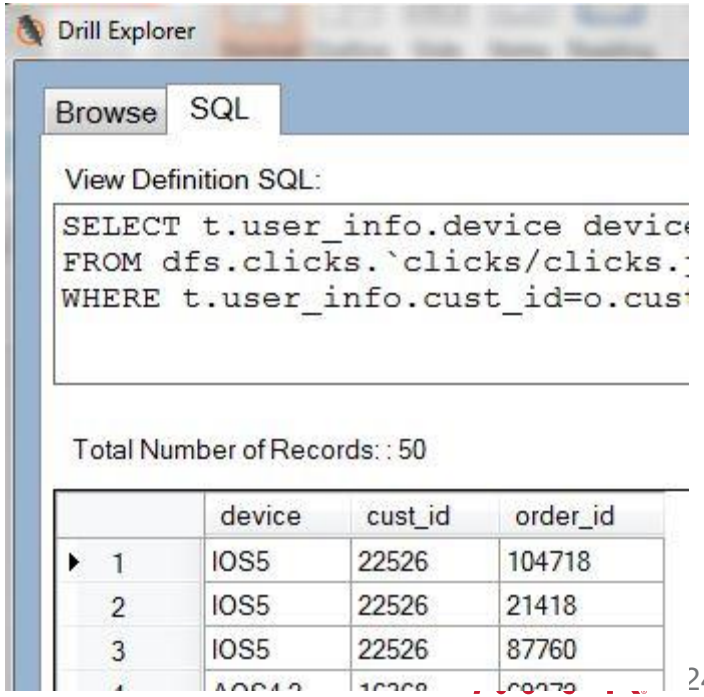
- Which device do customers who made orders use ?



Query Basics JOIN

```
SELECT  t.user_info.device device, t.user_info.cust_id cust_id, o.order_id
FROM  dfs.clicks.`clicks/clicks.json` t, hive.`orders` o
WHERE t.user_info.cust_id=o.cust_id
```

Query to get the device, and order id from the click log and hive orders, join on customer id



Cast function with HBase

Business Scenario

- Select an HBase column in a query

rowkey	CF: address	CF: loyalty		CF:personal		
	state	agg_rev	membership	age	gender	name
1	"va"	197.00	"silver"	"15-20"	"FEMALE"	Brittany Park

Selecting a Column with HBase

```
>SELECT t.personal.name FROM maprdb.customers t
LIMIT 1;
+------------+
|   EXPR$0   |
+------------+
|[B@6780874d |
+------------+
```

table ALIAS

rowkey	CF: address	CF: loyalty		CF:personal		
	state	agg_rev	membership	age	gender	name
1	"va"	197.00	"silver"	"15-20"	"FEMALE"	Brittany Park

CONVERT_FROM function with HBase

Business Scenario

- Convert HBase bytes to datatypes in a query

```
+------------+
|   EXPR$0   |
+------------+
|[B@6780874d |
+------------+
```

Data Preview:

	Row_Key	age	gender	name
1	10001	"15-20"	"FEMALE"	"Corrine Mecham"
2	10005	"26-35"	"MALE"	"Brittany Park"

rowkey	CF: address	CF: loyalty		CF:personal		
	state	agg_rev	membership	age	gender	name
1	"va"	197.00	"silver"	"15-20"	"FEMALE"	Brittany Park

Using convert_from with HBase

```
> SELECT convert_from(t.personal.name, 'UTF8') as pName
> FROM customers t LIMIT 2;
+-----------------+
|    pName        |
+-----------------+
| "Corrine Mecham" |
| "Brittany Park"  |
+-----------------+
2 rows selected
```

Table ALIAS

column ALIAS

convert_from with HBase

```
use maprdb;
select convert_from(row_key , 'UTF8') as cust_id,
convert_from(t.personal.name , 'UTF8') as name,
convert_from(t.personal.gender ,'UTF8') as gender,
convert_from(t.personal.age , 'UTF8') as age,
convert_from(t.address.state , 'UTF8') as state,
convert_from(t.loyalty.agg_rev, 'UTF8') as agg_rev,
convert_from(t.loyalty.membership, 'UTF8') as membership
from customers t limit 5;
```

cust_id	name	gender	age	state	agg_rev	membership
10001	"Corrine Mecham"	"FEMALE"	"15-20"	"va"	197	
10005	"Brittany Park"	"MALE"	"26-35"	"in"	230	
10006	"Rose Lokey"	"MALE"	"26-35"	"ca"	250	
10007	"James Fowler"	"FEMALE"	"51-100"	"me"	263	
10010	"Guillermo Koehler"	"OTHER"	"51-100"	"mn"	20	

Combining and Aggregating Data from Files, Hive, and HBase using Drill and ANSI SQL

- GROUP BY
- UNION
- JOIN
- SUBQUERY
- CONVERT_FROM
- CREATING A VIEW
- CREATING A TABLE

Simplify future Queries with a View

Business Scenario

- Make the **Converted HBase data easily available** in a **view**

```
+-------------+
|   EXPR$0    |
+-------------+
|[B@6780874d |
+-------------+
```

View Definition SQL:

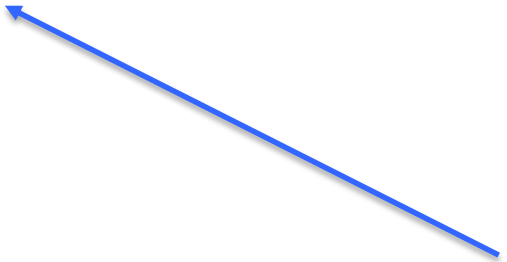
```
select convert_from(row_key , 'UTF8') as cust_id,
convert_from(t.personal.name , 'UTF8') as name,
convert_from(t.personal.gender ,'UTF8') as gender,
convert_from(t.personal.age , 'UTF8') as age,
```

Total Number of Records: : 5

	cust_id	name	gender	age	state	agg_rev	membership
1	10001	"Corrine Mecham"	"FEMALE"	"15-20"	"va"	197	"silver"
2	10005	"Brittany Park"	"MALE"	"26-35"	"in"	230	"silver"

Mutable Workspace for Views and/or Tables

use **dfs.mydata**



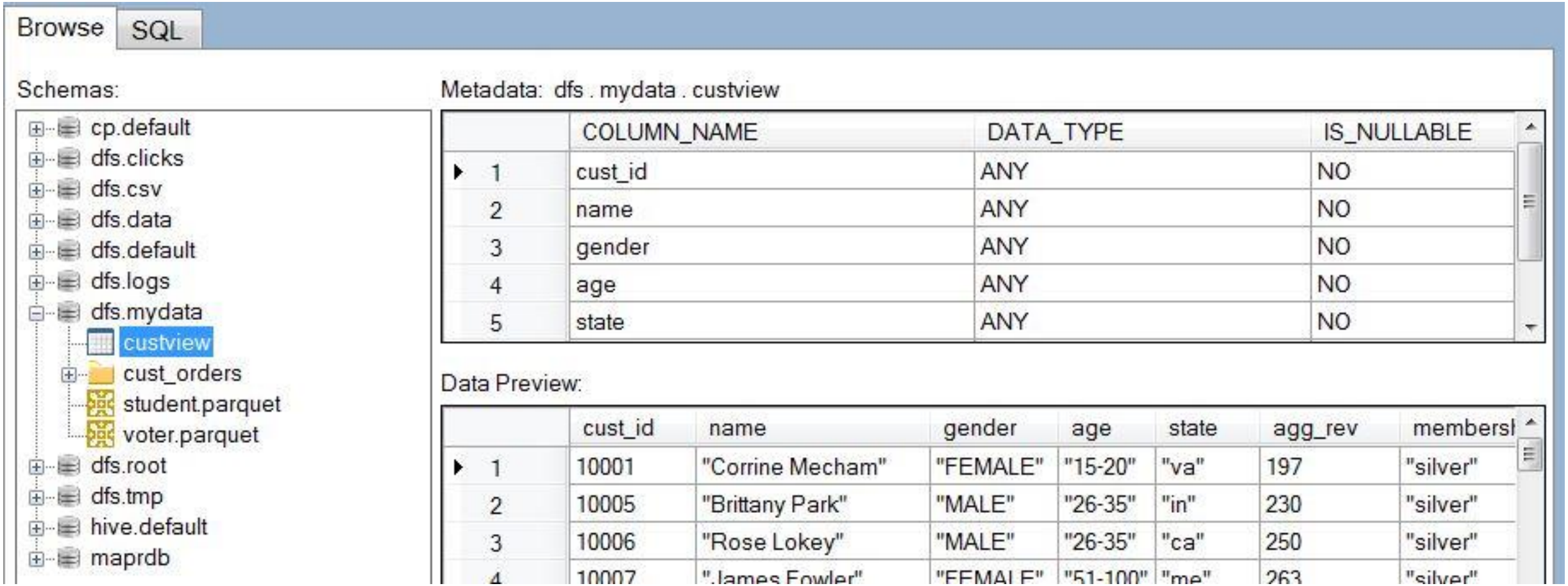
Mutable workspace

```
"type": "file",
"enabled": true,
"connection": "maprfs:///",
"workspaces": {
   “mydata": {
     "location": "/mapr/data/views",
     "writable": true,
     . . .
    },
}
```

Create a View

```
use dfs.mydata;

create or replace view custview as
select convert_from(row_key , 'UTF8') as cust_id,
convert_from(t.personal.name , 'UTF8') as name,
convert_from(t.personal.gender ,'UTF8') as gender,
convert_from(t.personal.age , 'UTF8') as age,
convert_from(t.address.state , 'UTF8') as state,
convert_from(t.loyalty.agg_rev, 'UTF8') as agg_rev,
convert_from(t.loyalty.membership, 'UTF8') as membership
from maprdb.customers t;
```

View created



Drill View Creates metadata in the filesystem

A view is simply a special file with a specific extension (.drill) :

$ cat /mapr/demo.mapr.com/data/views/custview.view.drill

metadata which was created for this view:

```
{
  "name" : "custview",
  "sql" : "SELECT CAST(`row_key` AS INTEGER) AS `cust_id`, CAST(`t`.`personal`['name'] AS VARCHAR(20)) AS `name`,
CAST(`t`.`personal`['gender'] AS VARCHAR(10)) AS `gender`, CAST(`t`.`personal`['age'] AS VARCHAR(10)) AS `age`,
CAST(`t`.`address`['state'] AS VARCHAR(4)) AS `state`, CAST(`t`.`loyalty`['agg_rev'] AS DECIMAL(7, 2)) AS `agg_rev`,
CAST(`t`.`loyalty`['membership'] AS VARCHAR(20)) AS `membership`\nFROM `maprdb`.`customers` AS `t`",
  "fields" : [ {
    "name" : "cust_id",
    "type" : "INTEGER"
  }, {
    "name" : "name",
    "type" : "VARCHAR",
    "precision" : 20
  }, {
    "name" : "gender",
…
```

What is a Drill View?

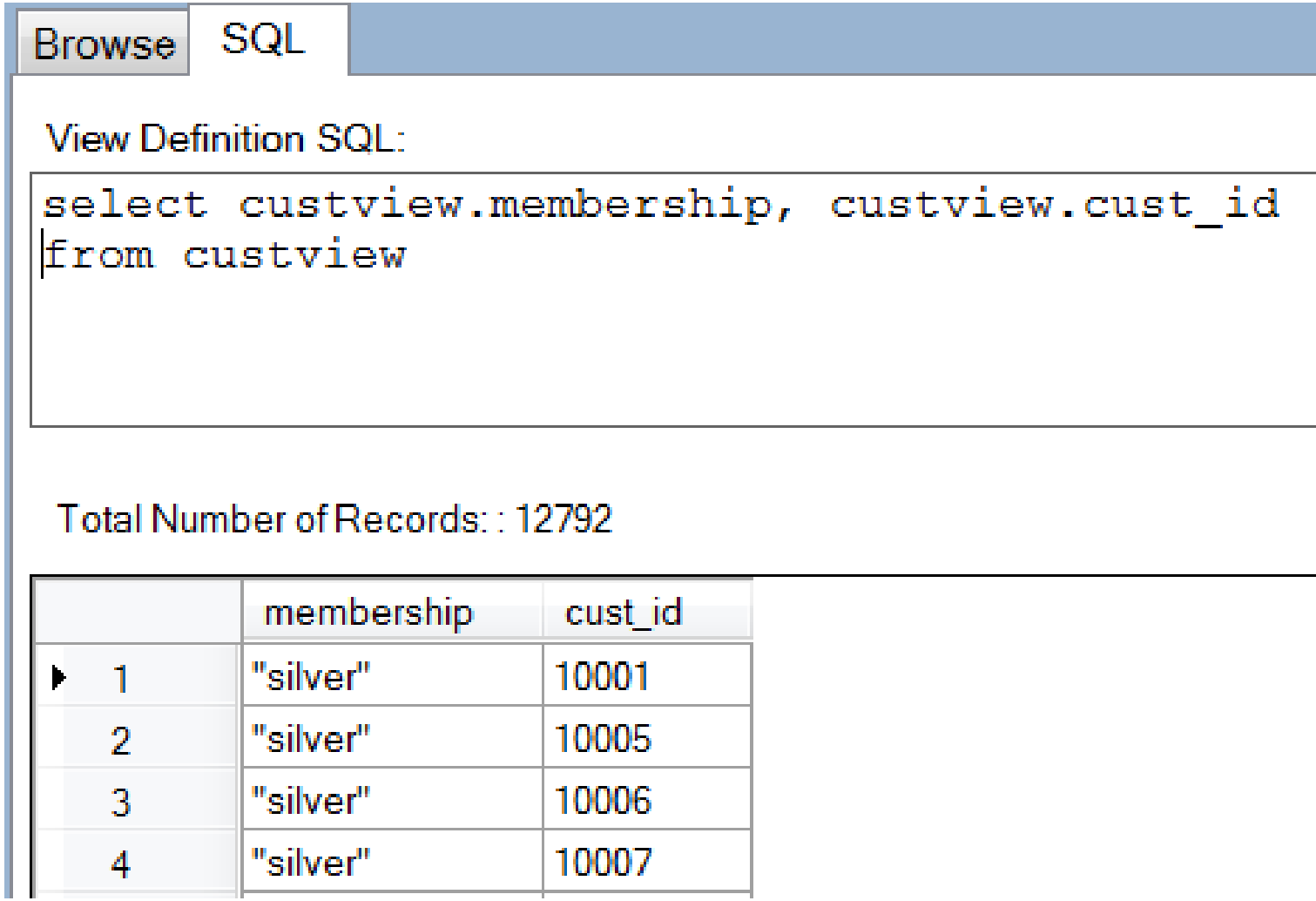
- A Drill view is a JSON file
- Stored metadata for a query
- no actual data
  - Accesses latest data
- View Can be used :
  - to simplify complex queries
  - Like a table
    - with BI tools like Tableau

How to Use a View ?

```
use dfs.mydata;
select * from custview limit 1;
+------------+------------------+------------+------------+------------+-------------+
|  cust_id   |    name          |   gender   |    age     |   state    |  agg_rev    |
+------------+------------------+------------+------------+------------+-------------+
| 10001      | "Corrine Mecham" | "FEMALE"   | "15-20"    | "va"       | 197.0 |
+------------+------------------+------------+------------+------------+-------------+
1 row selected
```

Using a View

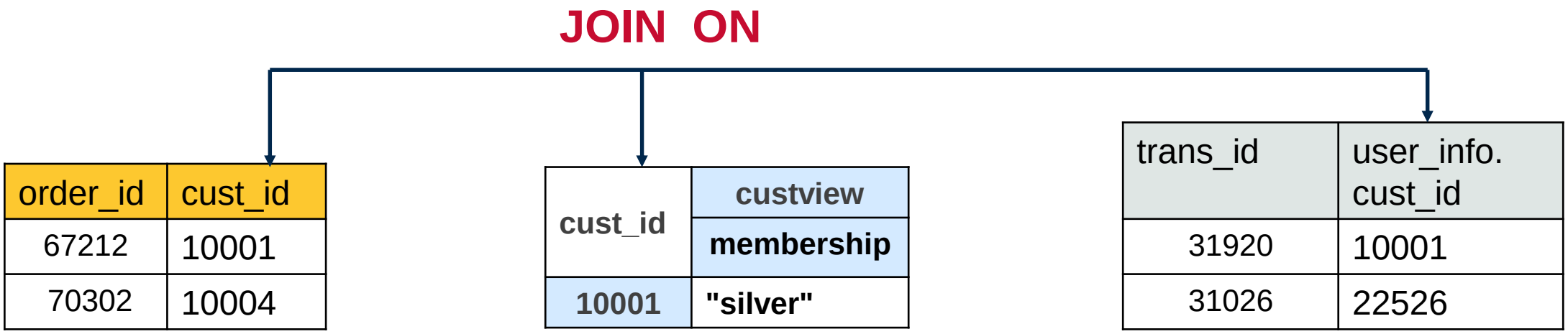
```
select
custview.membership,
custview.cust_id
from custview
```



How to Use a View with a JOIN

Business Scenario

- Show Customer Order information



HIVE orders

Customer **View**

Clicks json

Joining with a View

Join the customers view and the orders table:

```
select custview.membership, sum(orders.order_total) as
sales
from hive.orders, custview,
dfs.`/mapr/demo.mapr.com/data/nested/clicks/clicks.json` c
where orders.cust_id=custview.cust_id and
orders.cust_id=c.user_info.cust_id
group by custview.membership order by 2;
```

```
+------------+------------+
| membership |   sales    |
+------------+------------+
| "basic"    | 372866     |
| "silver"   | 728424     |
| "gold"     | 7050198    |
+------------+------------+
3 rows selected (11.374 seconds)
```

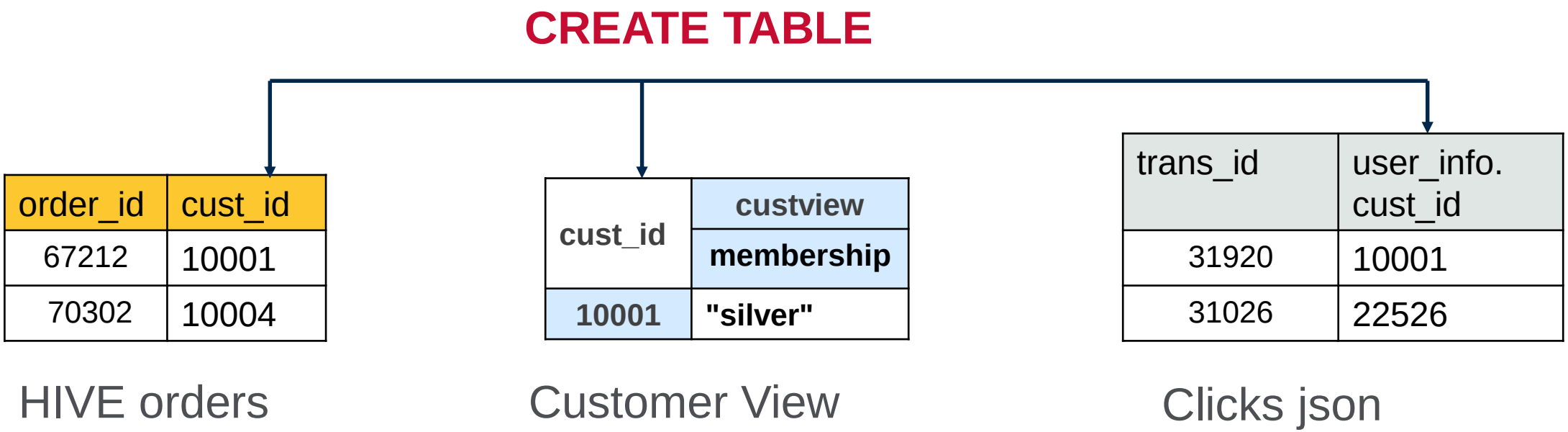
Combining and Aggregating Data from Files, Hive, and HBase using Drill and ANSI SQL

- GROUP BY
- UNION
- JOIN
- SUBQUERY
- CAST
- CREATING A VIEW
- CREATING A TABLE

Create a Table from multiple sources

Business Scenario

- Create a table with the Customer Order information from different data sources

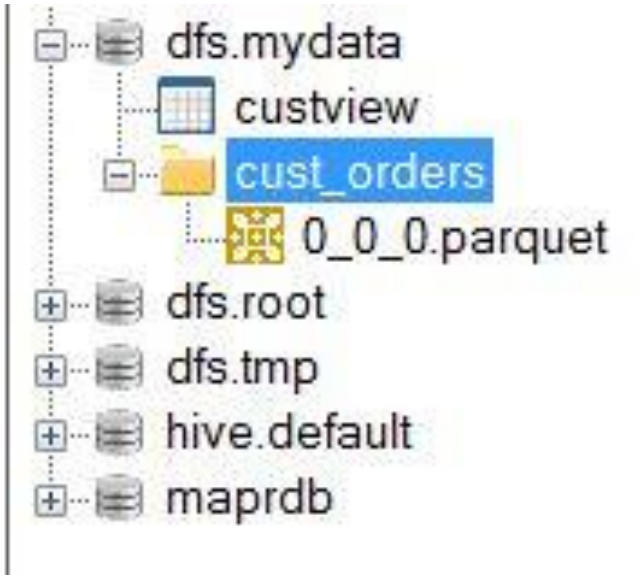


Create a Table from the previous Join

```
use dfs.mydata;

create table cust_orders as
select custview.membership, sum(orders.order_total) as sales
from hive.orders, custview,
dfs.`/mapr/demo.mapr.com/data/nested/clicks/clicks.json` c
where orders.cust_id=custview.cust_id
and orders.cust_id=c.user_info.cust_id
group by custview.membership order by 2;
```

Table created

- The “CREATE TABLE AS” command creates a directory that contains parquet files with the table data
  - Parquet is a columnar storage format



Data Preview:

	membership	sales
1	"basic"	372866
2	"silver"	728424
3	"gold"	7050198

Query the Table Created

Query the table created:

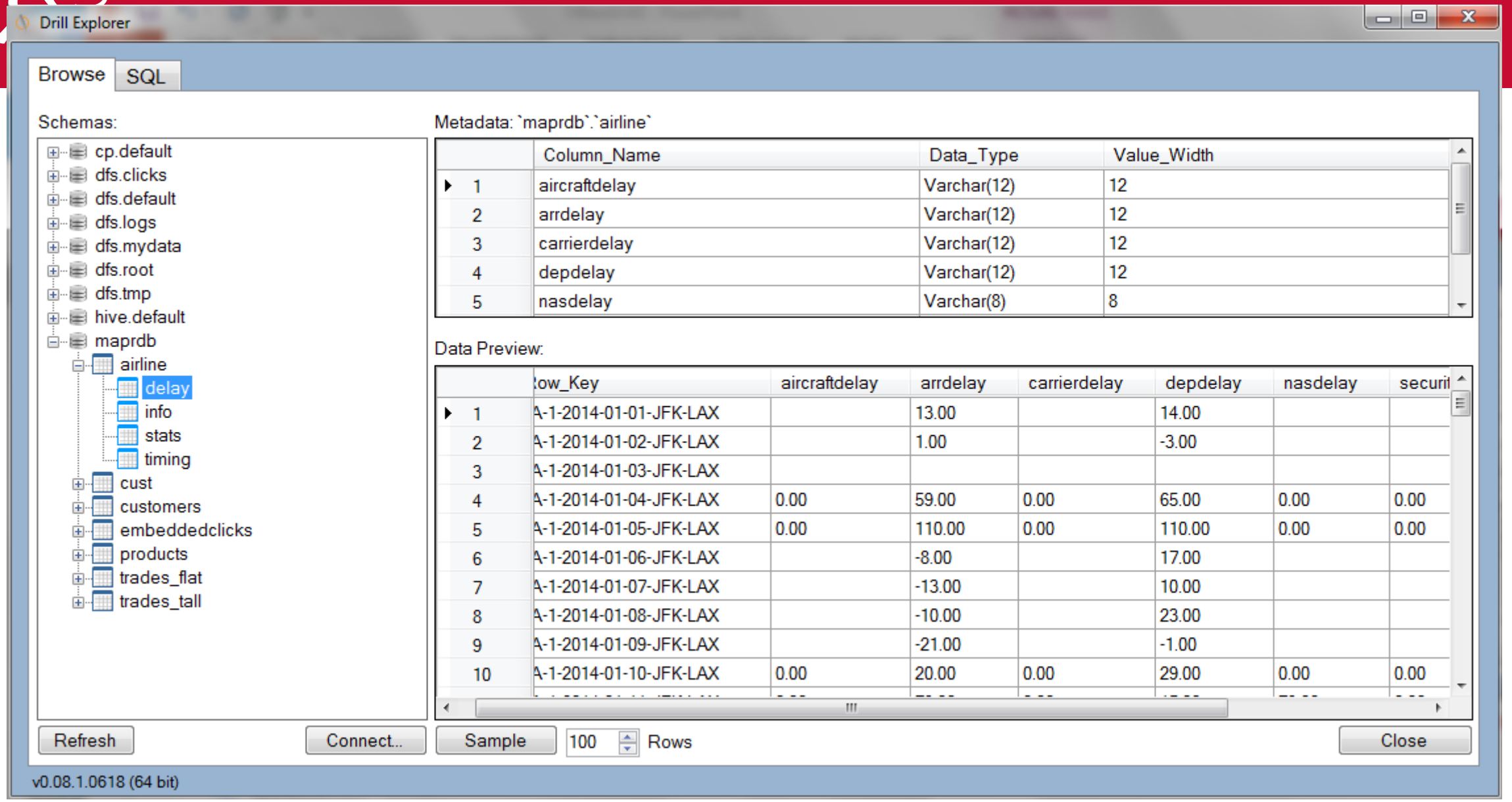
```
select * from cust_orders;
```

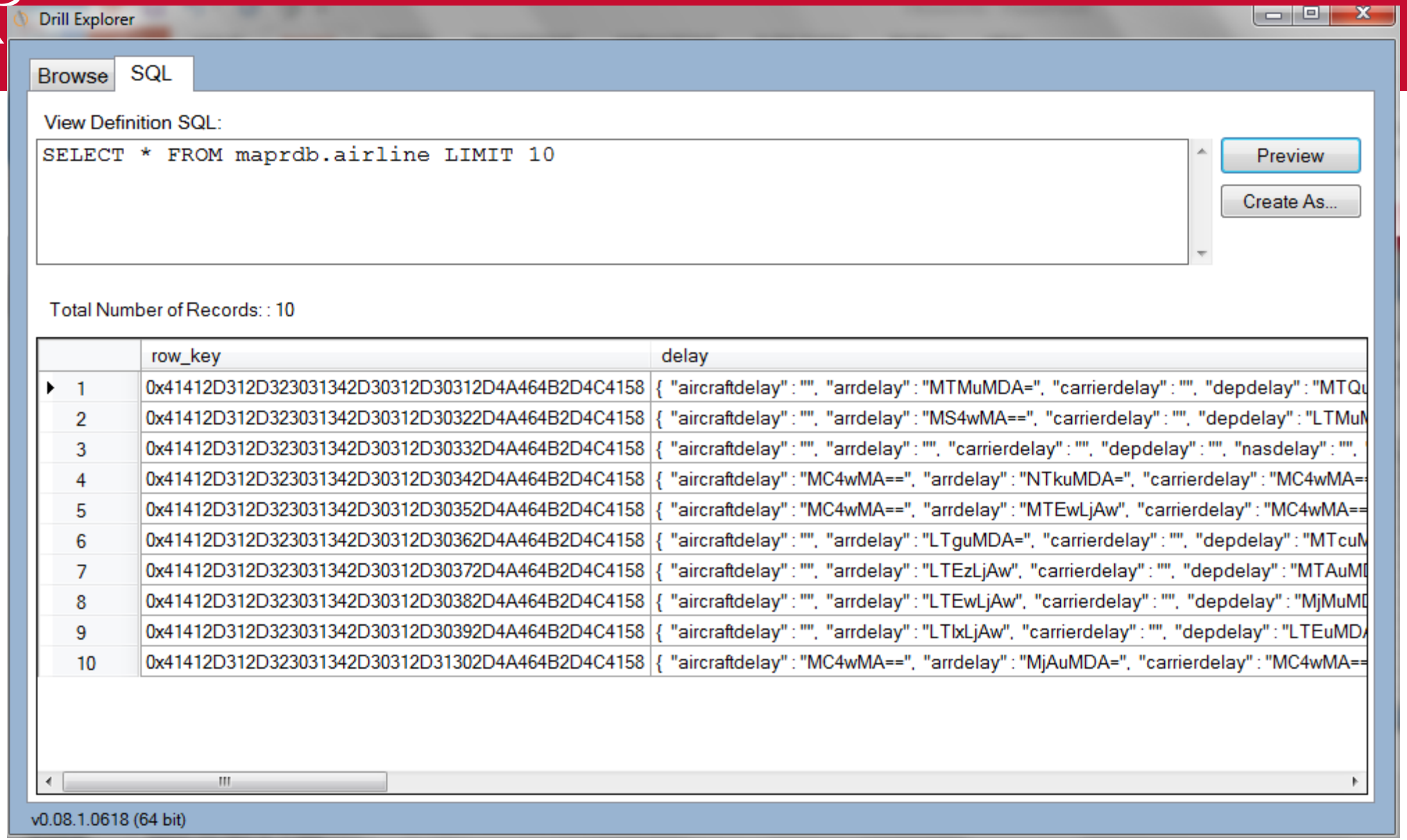
```
+------------+------------+
| membership |   sales    |
+------------+------------+
| "basic"    | 372866     |
| "silver"   | 728424     |
| "gold"     | 7050198    |
+------------+------------+
3 rows selected (0.073 seconds)
```

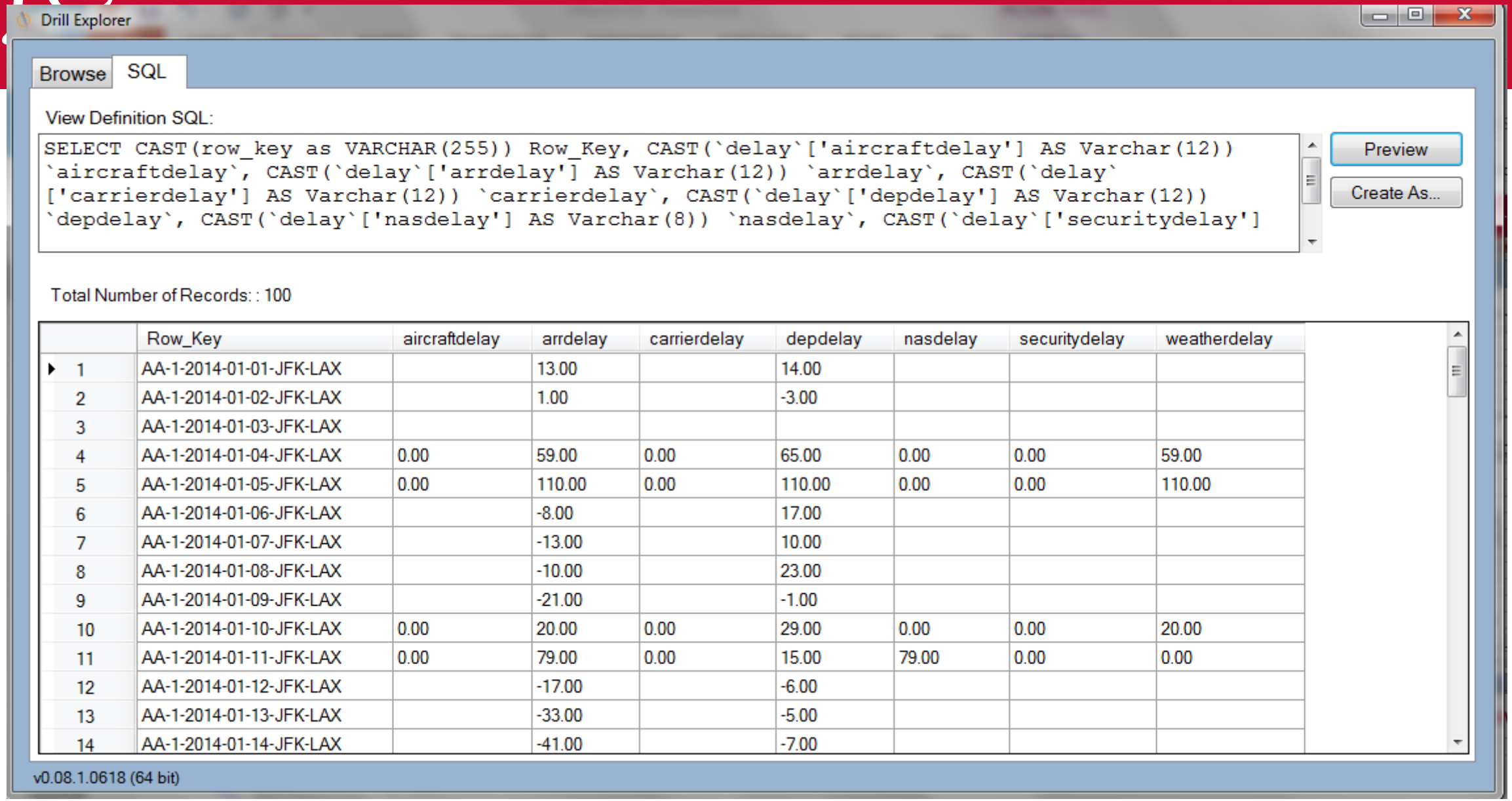
Lab – Query HBase airline data with Drill

Import mapping to Row Key and Columns:

Row-key Carrier-Flightnumber-Date-Origin-destination	delay			info			stats		timing	
	Air Craft delay	Arr delay	Carrier delay	cncl	Cncl code	tailnum	distance	elaptime	arrtime	Dep time
AA-1-2014-01-01-JFK-LAX		13		0		N7704	2475	385.00	359	…







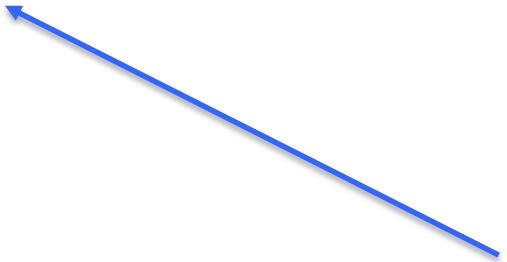
	Row_Key	aircraftdelay	arrdelay	carrierdelay	depdelay	nasdelay	securitydelay	weatherdelay
1	AA-1-2014-01-01-JFK-LAX		13.00		14.00			
2	AA-1-2014-01-02-JFK-LAX		1.00		-3.00			
3	AA-1-2014-01-03-JFK-LAX							
4	AA-1-2014-01-04-JFK-LAX	0.00	59.00	0.00	65.00	0.00	0.00	59.00
5	AA-1-2014-01-05-JFK-LAX	0.00	110.00	0.00	110.00	0.00	0.00	110.00
6	AA-1-2014-01-06-JFK-LAX		-8.00		17.00			
7	AA-1-2014-01-07-JFK-LAX		-13.00		10.00			
8	AA-1-2014-01-08-JFK-LAX		-10.00		23.00			
9	AA-1-2014-01-09-JFK-LAX		-21.00		-1.00			
10	AA-1-2014-01-10-JFK-LAX	0.00	20.00	0.00	29.00	0.00	0.00	20.00
11	AA-1-2014-01-11-JFK-LAX	0.00	79.00	0.00	15.00	79.00	0.00	0.00
12	AA-1-2014-01-12-JFK-LAX		-17.00		-6.00			
13	AA-1-2014-01-13-JFK-LAX		-33.00		-5.00			
14	AA-1-2014-01-14-JFK-LAX		-41.00		-7.00			

Create a View

```
use dfs.mydata;

create or replace view airdelayview as
SELECT convert_from(row_key ,'UTF8') as rowkey,
convert_from(t.delay.aircraftdelay,'UTF8') as aircraftdelay,
convert_from(t.delay.arrdelay,'UTF8') AS arrdelay,
convert_from(t.delay.carrierdelay,'UTF8') AS carrierdelay,
convert_from(t.delay.depdelay,'UTF8') AS depdelay,
convert_from(t.delay.nasdelay,'UTF8') AS nasdelay,
convert_from(t.delay.securitydelay,'UTF8') AS securitydelay,
convert_from(t.delay.weatherdelay,'UTF8') AS weatherdelay
 FROM maprdb.airline t;
```

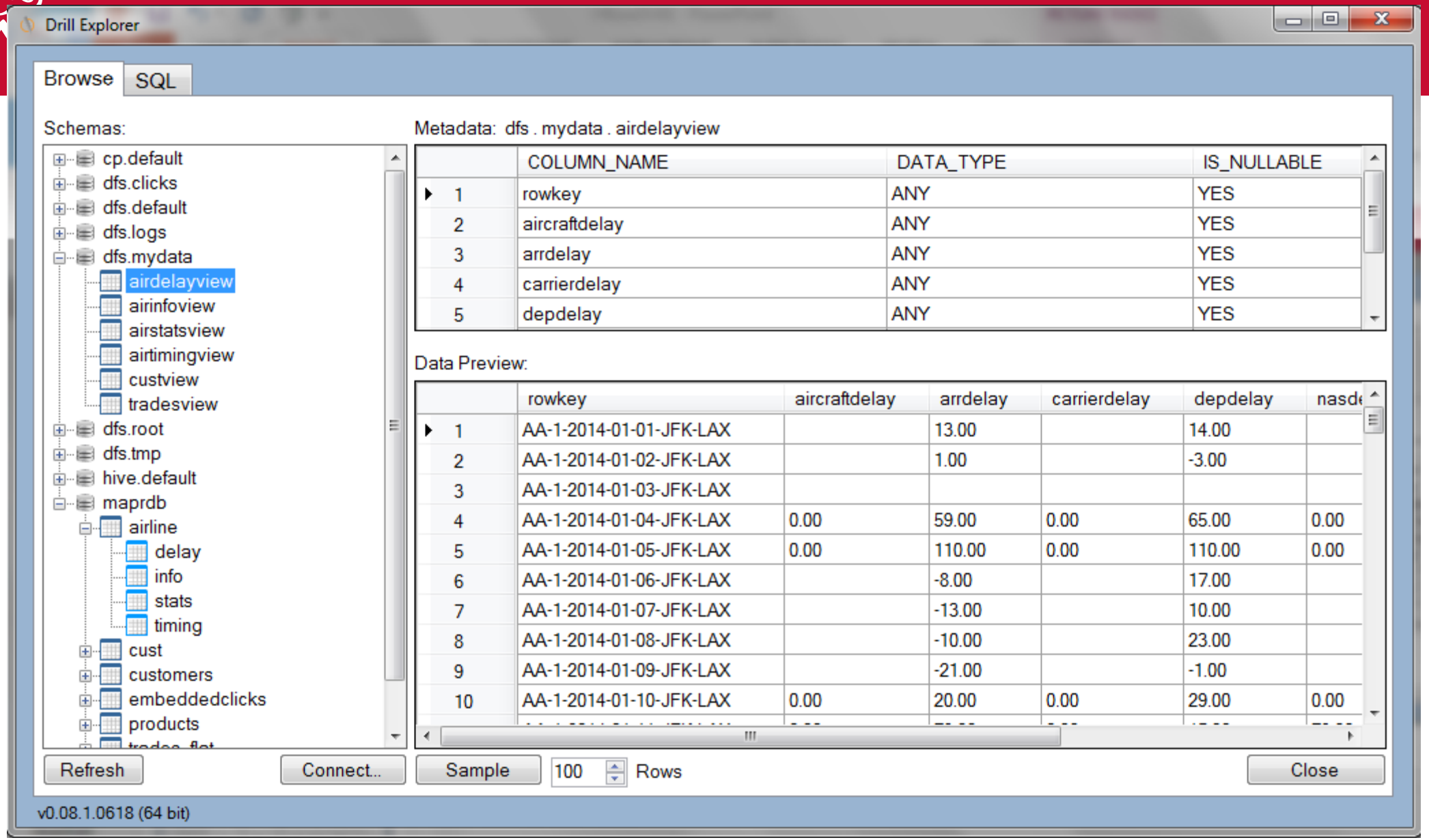
Mutable Workspace for Views and/or Tables

use **dfs.mydata**



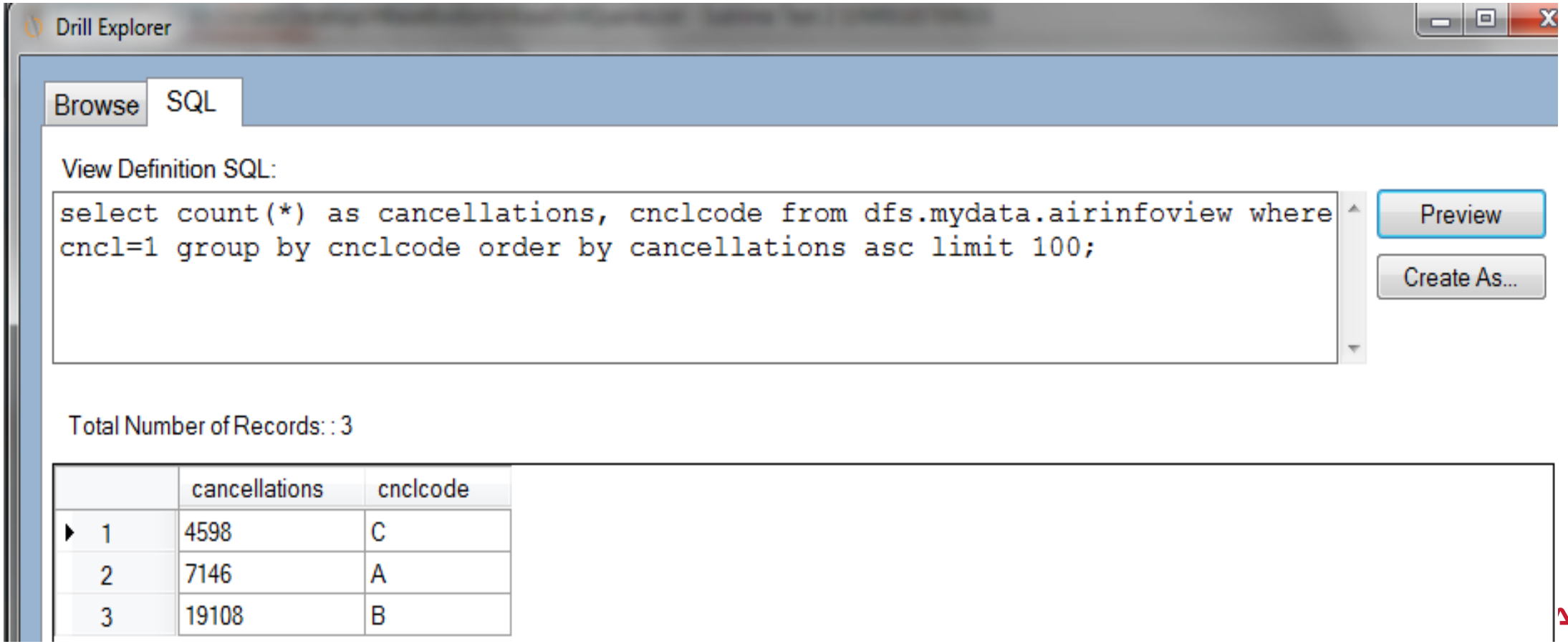
Mutable workspace

```
"type": "file",
"enabled": true,
"connection": "maprfs:///",
"workspaces": {
   “mydata": {
     "location": "/mapr/data/views",
     "writable": true,
     . . .
    },
}
```



Count number of cancellations by reason (code)

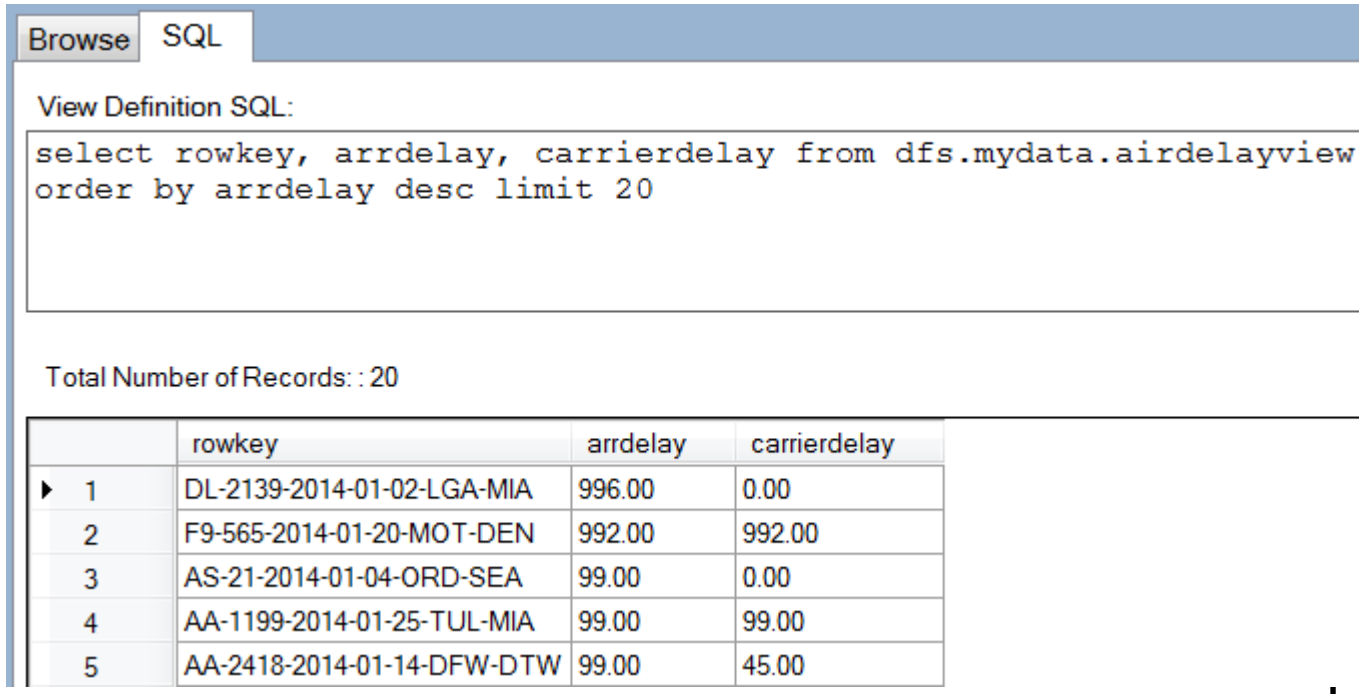
```
select count(*) as cancellations, cnclcode from
dfs.mydata.airinfoview where cncl=1 group by cnclcode order
by cancellations asc limit 100;
```



	cancellations	cnclcode
1	4598	C
2	7146	A
3	19108	B

Find the longest airline delays

select rowkey, arrdelay, carrierdelay from dfs.mydata.airdelayview order by arrdelay desc limit 20;



Browse | SQL

View Definition SQL:

```
select rowkey, arrdelay, carrierdelay from dfs.mydata.airdelayview
order by arrdelay desc limit 20
```

Total Number of Records: : 20

	rowkey	arrdelay	carrierdelay
1	DL-2139-2014-01-02-LGA-MIA	996.00	0.00
2	F9-565-2014-01-20-MOT-DEN	992.00	992.00
3	AS-21-2014-01-04-ORD-SEA	99.00	0.00
4	AA-1199-2014-01-25-TUL-MIA	99.00	99.00
5	AA-2418-2014-01-14-DFW-DTW	99.00	45.00

1 row

- Complex Data Types
  - Map, Array
- Query JSON file with Map Data type
- Query JSON file with Array Data type
- Join Hive and JSON file with complex data types
- Create a table from Join Hive and JSON file with complex data types
- Query Directories

Complex Data Types

- MAP
  - set of name/value pairs

  “user_info”: {"cust_id":22526,"device":"IOS5","state":"il"}

- ARRAY
  - a repeated list of values

  "prod_id":[174,2]

	trans_id	date	time	user_info	trans_info
1	31920	2014-04-26	12:17:12	{ "cust_id" : 22526, "device" : "IOS5", "state" : "il"}	{ "prod_id" : [174, 2], "purch_flag" : "false"}
2	31026	2014-04-20	13:50:29	{ "cust_id" : 16368, "device" : "AOS4.2", "state" : "nc"}	{ "prod_id" : [ ], "purch_flag" : "false"}
3	33848	2014-04-10	04:44:42	{ "cust_id" : 21449, "device" : "IOS6", "state" : "oh"}	{ "prod_id" : [582], "purch_flag" : "false"}
4	32383	2014-04-18	06:27:47	{ "cust_id" : 20323, "device" : "IOS5", "state" : "oh"}	{ "prod_id" : [710, 47], "purch_flag" : "false"}
5	32359	2014-04-19	23:13:25	{ "cust_id" : 15360, "device" : "IOS5", "state" : "ca"}	{ "prod_id" : [0, 8, 170, 173, 1, 124, 46, 764, 30,

Selecting a column from a JSON Map Data Type

Business Scenario

- Select JSON Map "columns" cust_id, device… in a query

	trans_id	date	time	user_info	trans_info
1	31920	2014-04-26	12:17:12	{ "cust_id" : 22526, "device" : "IOS5", "state" : "il"}	{ "prod_id" : [174, 2], "purch_flag" : "false"}
2	31026	2014-04-20	13:50:29	{ "cust_id" : 16368, "device" : "AOS4.2", "state" : "nc"}	{ "prod_id" : [ ], "purch_flag" : "false"}
3	33848	2014-04-10	04:44:42	{ "cust_id" : 21449, "device" : "IOS6", "state" : "oh"}	{ "prod_id" : [582], "purch_flag" : "false"}
4	32383	2014-04-18	06:27:47	{ "cust_id" : 20323, "device" : "IOS5", "state" : "oh"}	{ "prod_id" : [710, 47], "purch_flag" : "false"}
5	32359	2014-04-19	23:13:25	{ "cust_id" : 15360, "device" : "IOS5", "state" : "ca"}	{ "prod_id" : [0, 8, 170, 173, 1, 124, 46, 764, 30,

Results of Query JSON with Complex Data Types

table.column.column notation :

```
select t.user_info.cust_id
as custid,
t.user_info.device
as device,
t.user_info.state
as state
from `clicks/clicks.json` t
limit 5;
```

Browse | SQL

View Definition SQL:

```
select t.user_info.cust_id as custid,
t.user_info.device as device,
t.user_info.state as state
from `clicks/clicks.json` t limit 5;
```

Total Number of Records: : 5

	custid	device	state
1	22526	IOS5	il
2	16368	AOS4.2	nc
3	21449	IOS6	oh
4	20323	IOS5	oh
5	15360	IOS5	ca

Result of Query with Map Data Type

table.column.column notation :

```
select
t.trans_info.prod_id
as prodid,
t.trans_info.purch_flag
as purchased
from
`clicks/clicks.json` t
limit 5;
```

Browse | SQL

View Definition SQL:

```
select t.trans_info.prod_id as prodid,
t.trans_info.purch_flag as purchased
from `clicks/clicks.json` t limit 5;
```

Total Number of Records: : 5

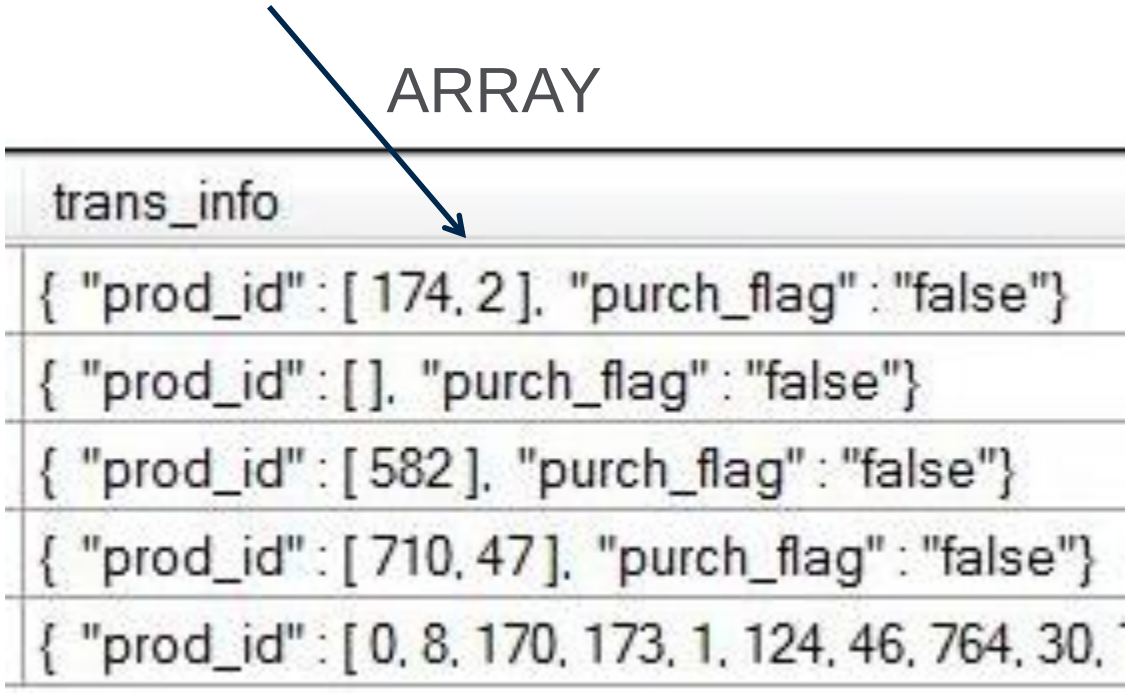
ARRAY

	prodid	purchased
1	[174, 2]	false
2	[]	false
3	[582]	false
4	[710, 47]	false
5	[0, 8, 170, 173, 1, 124, 46, 764, 30, 711, 0, 3, 25]	true

Selecting a column from a JSON Map Data Type

Business Scenario

- Select a Value in a JSON Array in a query
- **t.trans_info.prod_id[0]**



Query JSON with Arrays

Get first product Id in prod_id array

```
select t.user_info.cust_id as cust_id,
t.trans_info.prod_id[0] as prod_id
from `clicks/clicks.json` t;
```

```
+------------+------------+
|  cust_id   |  prod_id   |
+------------+------------+
| 2587       | 174        |
| 1772       |            |
| 1429       | 582        |
+------------+------------+
```

Query JSON with Arrays

Business Scenario

- **Which customers searched on at least 10 products?**

```
t.trans_info.prod_id[10]
is not null
```

trans_id	date	user_info. cust_id	user_info. device	user_info. state	trans_info. prod_id	trans_info. purch_flag
31920	2014-04-26	10001	“iOS5”	“ca”	[174,2]	false
31026	2014-04-26	22526	AOS4.2	“nc”	[420]	true

clicks.json file

Query JSON with Arrays

Which customers searched on at least 10 products?

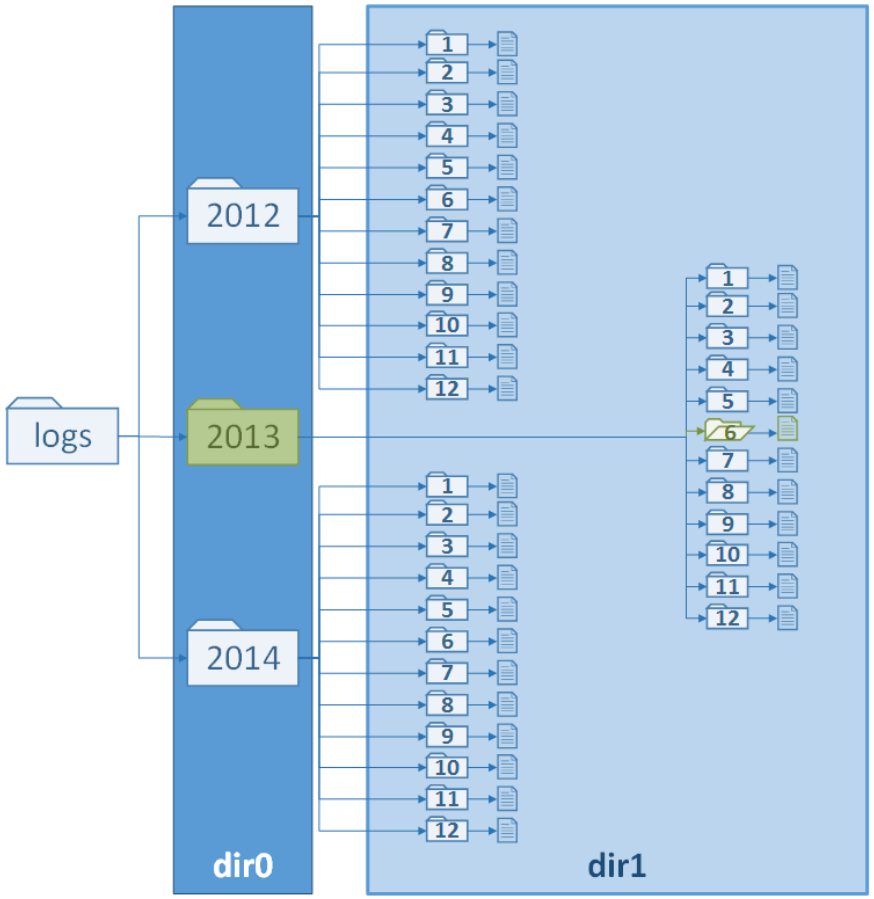
```
select t.user_info.cust_id as cust_id,
t.trans_info.prod_id[10] as prod_id
from `clicks/clicks.json` t
where t.trans_info.prod_id[10] is not null order by trans_id;
```

```
+------------+------------+
|  cust_id   |  prod_id   |
+------------+------------+
| 2587       | 156        |
| 1772       | 62         |
| 1429       | 18         |
| 4344       | 29         |
| 1818       | 0          |
+------------+------------+
```

Directories Partitions

- When Querying Files
  - partitions are simply nested subdirectories of the table directory.

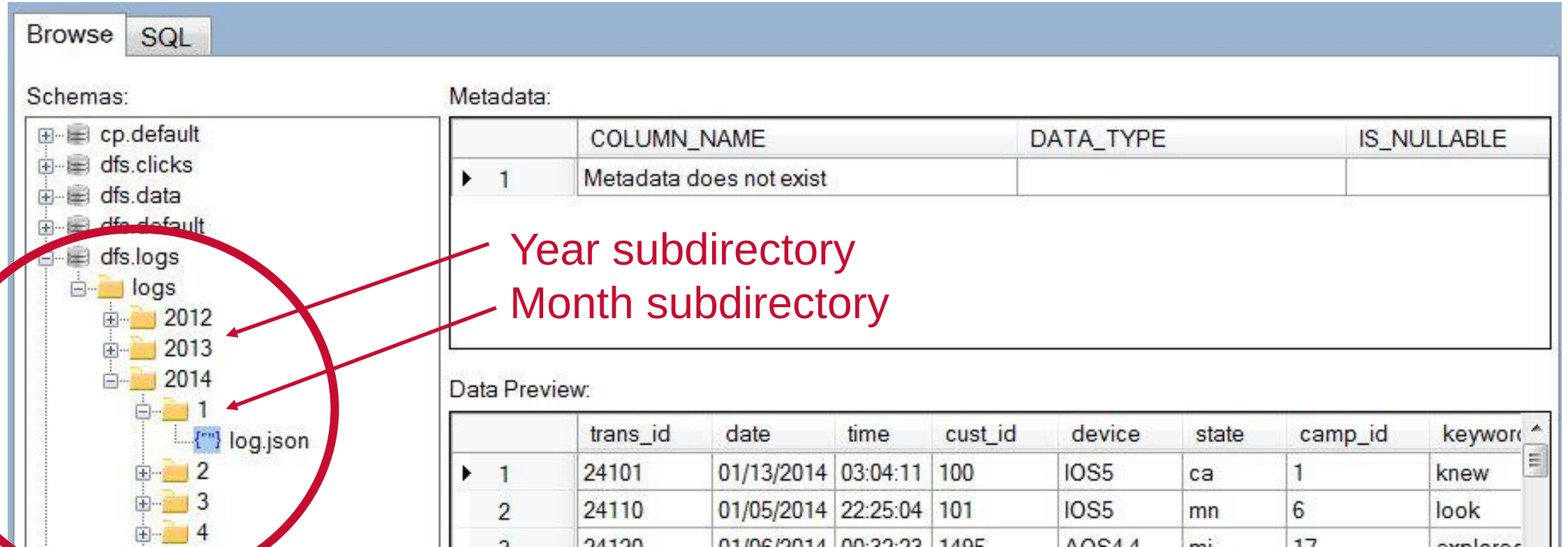
```
drill> select * from logs where dir0=2013 and dir1=6;
```



Query Partitioned directories

Business Scenario

- query files in directories and subdirectories in a single SELECT statement



Query directory logs

Business Scenario

- Query file data from logs top directory

Browse | SQL

View Definition SQL:

```
select * from logs limit 10;
```

Prev

Create

Year subdirectory

Month subdirectory

Total Number of Records: : 10

	dir0	dir1	trans_id	date	time	cust_id	device	state	camp_id	keywords	prod_id
1	2014	8	24181	08/02/2014	09:23:52	0	IOS5	il	2	wait	128
2	2014	8	24195	08/02/2014	07:58:19	243	IOS5	mo	6	hmm	107
3	2014	8	24204	08/01/2014	12:10:27	12048	IOS6	il	1	marge	324
4	2014	8	24222	08/02/2014	16:28:37	2488	IOS6	pa	2	to	391
5	2014	8	24227	08/02/2014	07:14:00	154687	IOS5	wa	2	on	376

Query subdirectory

Business Scenario

- Query log file data for the year 2014

Year subdirectory

	dir0	dir1	trans_id	date	time	cust_id	device	state	camp_id	keywords	prod_id
1	2014	8	24181	08/02/2014	09:23:52	0	IOS5	il	2	wait	128
2	2014	8	24195	08/02/2014	07:58:19	243	IOS5	mo	6	hmm	107
3	2014	8	24204	08/01/2014	12:10:27	12048	IOS6	il	1	marge	324
4	2014	8	24222	08/02/2014	16:28:37	2488	IOS6	pa	2	to	391
5	2014	8	24227	08/02/2014	07:14:00	154687	IOS5	wa	2	on	376
6	2014	8	24249	08/03/2014	23:18:59	1	IOS5	nv	1	hey	348
7	2014	8	24261	08/04/2014	23:32:16	7389	IOS5	mn	7	kapur	65
8	2014	8	24287	08/02/2014	18:21:26	2	IOS5	tx	13	church	283
9	2014	8	24323	08/04/2014	03:22:13	447	AOS4.4	in	9	here	496
10	2014	8	24378	08/01/2014	14:46:42	14006	IOS6	il	8	cecil	162

Query Selecting a subdirectory

Business Scenario

- Query log file data for the year 2014

```
> select * from logs where dir0='2014' ;
```

```
+------------+------------+------------+------------+------------+------------+
|    dir0    |    dir1    |  trans_id  |    date    |    time    |  cust_id   |
+------------+------------+------------+------------+------------+------------+
| 2014       | 8          | 24181      | 08/02/2014 | 09:23:52   | 0          |
| 2014       | 8          | 24195      | 08/02/2014 | 07:58:19   | 243        |
```

. . .

Query subdirectory

Business Scenario

- Query log file data for the month of August

Month subdirectory

	dir0	dir1	trans_id	date	time	cust_id	device	state	camp_id	keywords	prod_id
1	2014	8	24181	08/02/2014	09:23:52	0	IOS5	il	2	wait	128
2	2014	8	24195	08/02/2014	07:58:19	243	IOS5	mo	6	hmm	107
3	2014	8	24204	08/01/2014	12:10:27	12048	IOS6	il	1	marge	324
4	2014	8	24222	08/02/2014	16:28:37	2488	IOS6	pa	2	to	391
5	2014	8	24227	08/02/2014	07:14:00	154687	IOS5	wa	2	on	376
6	2014	8	24249	08/03/2014	23:18:59	1	IOS5	nv	1	hey	348
7	2014	8	24261	08/04/2014	23:32:16	7389	IOS5	mn	7	kapur	65
8	2014	8	24287	08/02/2014	18:21:26	2	IOS5	tx	13	church	283
9	2014	8	24323	08/04/2014	03:22:13	447	AOS4.4	in	9	here	496
10	2014	8	24378	08/01/2014	14:46:42	14006	IOS6	il	8	cecil	162

Query Selecting subdirectories

Business Scenario

- Query log file data for the month of August

> select * from logs where dir1='8' ;

```
+------------+------------+------------+------------+------------+------------+
|    dir0    |    dir1    |  trans_id  |    date    |    time    |  cust_id   |
+------------+------------+------------+------------+------------+------------+
| 2014       | 8          | 24181      | 08/02/2014 | 09:23:52   | 0          |
| 2014       | 8          | 24195      | 08/02/2014 | 07:58:19   | 243        |
| 2014       | 8          | 24204      | 08/01/2014 | 12:10:27   | 12048      |
```

Aggregate counts with subdirectories

Business Scenario

- **What is the count of customer clicks for August 2014 ?**

Year= 2014 Month=8 Count distinct

	dir0	dir1	trans_id	date	time	cust_id	device	state	camp_id	keywords	prod_id
1	2014	8	24181	08/02/2014	09:23:52	0	IOS5	il	2	wait	128
2	2014	8	24195	08/02/2014	07:58:19	243	IOS5	mo	6	hmm	107
3	2014	8	24204	08/01/2014	12:10:27	12048	IOS6	il	1	marge	324
4	2014	8	24222	08/02/2014	16:28:37	2488	IOS6	pa	2	to	391
5	2014	8	24227	08/02/2014	07:14:00	154687	IOS5	wa	2	on	376
6	2014	8	24249	08/03/2014	23:18:59	1	IOS5	nv	1	hey	348
7	2014	8	24261	08/04/2014	23:32:16	7389	IOS5	mn	7	kapur	65
8	2014	8	24287	08/02/2014	18:21:26	2	IOS5	tx	13	church	283
9	2014	8	24323	08/04/2014	03:22:13	447	AOS4.4	in	9	here	496
10	2014	8	24378	08/01/2014	14:46:42	14006	IOS6	il	8	cecil	162

Aggregate counts with subdirectories

Business Scenario

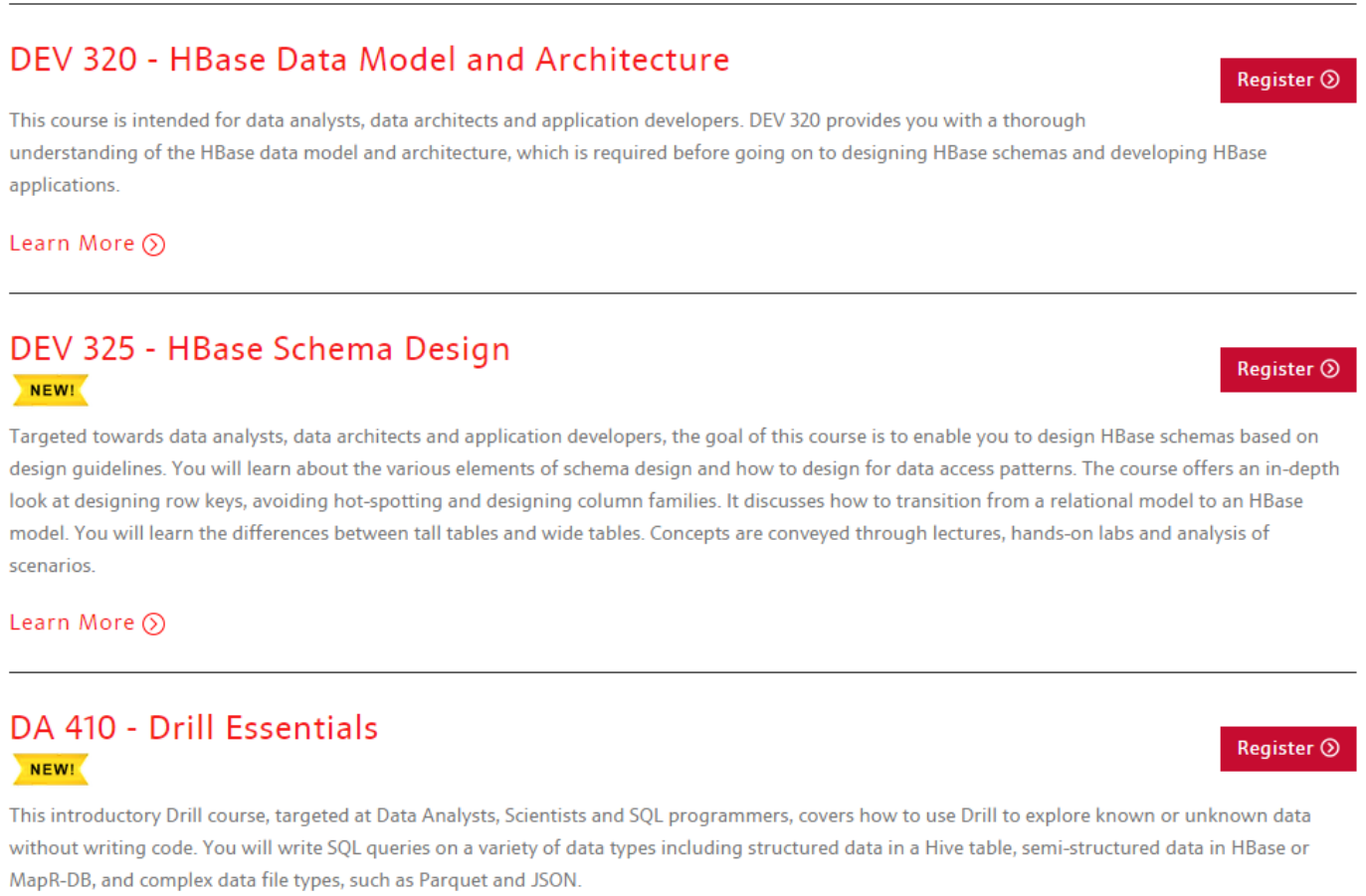
- **What is the count of customer clicks for August 2014 ?**

```
select count(distinct cust_id)
from logs
where dir0=2014 and dir1=8;
```

```
+------------+
|   EXPR$0   |
+------------+
| 167        |
+------------+
1 row selected
```

Free Hadoop On Demand Training

- https://www.mapr.com/services/mapr-academy/big-data-hadoop-online-training



DEV 320 - HBase Data Model and Architecture

Register

This course is intended for data analysts, data architects and application developers. DEV 320 provides you with a thorough understanding of the HBase data model and architecture, which is required before going on to designing HBase schemas and developing HBase applications.

Learn More

DEV 325 - HBase Schema Design

NEW!

Register

Targeted towards data analysts, data architects and application developers, the goal of this course is to enable you to design HBase schemas based on design guidelines. You will learn about the various elements of schema design and how to design for data access patterns. The course offers an in-depth look at designing row keys, avoiding hot-spotting and designing column families. It discusses how to transition from a relational model to an HBase model. You will learn the differences between tall tables and wide tables. Concepts are conveyed through lectures, hands-on labs and analysis of scenarios.

Learn More

DA 410 - Drill Essentials

NEW!

Register

This introductory Drill course, targeted at Data Analysts, Scientists and SQL programmers, covers how to use Drill to explore known or unknown data without writing code. You will write SQL queries on a variety of data types including structured data in a Hive table, semi-structured data in HBase or MapR-DB, and complex data file types, such as Parquet and JSON.

More Info

- https://www.mapr.com/blog/how-use-sql-hadoop-drill-rest-json-nosql-and-hbase-simple-rest-client#.VT0AUCFViko
- https://cwiki.apache.org/confluence/display/DRILL/Apache+Drill+Tutorial
- https://www.dropbox.com/home/bigdatatechcon

Next Steps

Q&A

Engage with us!

@mapr maprtech

mapr-technologies MapR

yourname@mapr.com maprtech