BITS Pilani
Pilani | Dubai | Goa | Hyderabad

DSECL ZG 522: Big Data Systems

Session 8: Hadoop Ecosystem: PIG, HIVE,HBASE

Janardhanan PS

Professor

janardhanan.ps@wilp.bits-pilani.ac.in

Topics for today

- Hadoop ecosystem technologies
  - ✓ **Pig - Scripting on top of MapReduce**
  - ✓ Hive - Data warehouse
  - ✓ HBase - Key-value store

Learn basics about

A. Applicability

B. Architecture

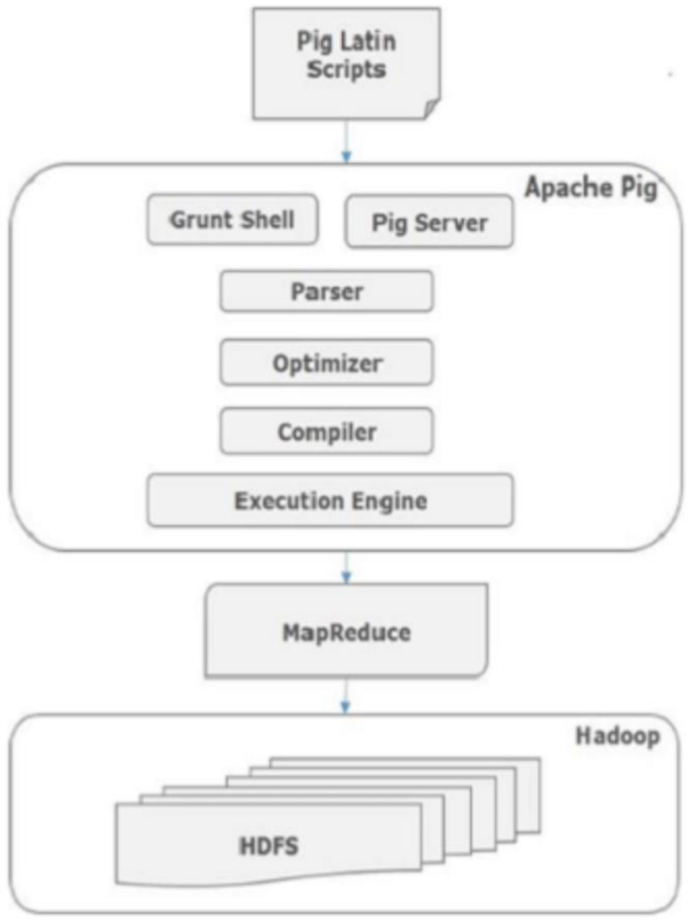
C. Data manipulation usage

Why PIG ?

- Hadoop MapReduce needs a developer to write UDFs for Map/Reduce using Java, Python etc.
- Pig gives developers a simpler scripting interface to manipulate data using a higher level programming interface
    - ✓ Similar to SQL - so becomes simpler for data engineers
- Pig script is translated to map reduce tasks by the Pig runtime and executed on the Hadoop cluster
    - ✓ However automated code may be less efficient than a native MR program
- Not an acronym - Pig can consume and process any data
- Developed in Yahoo

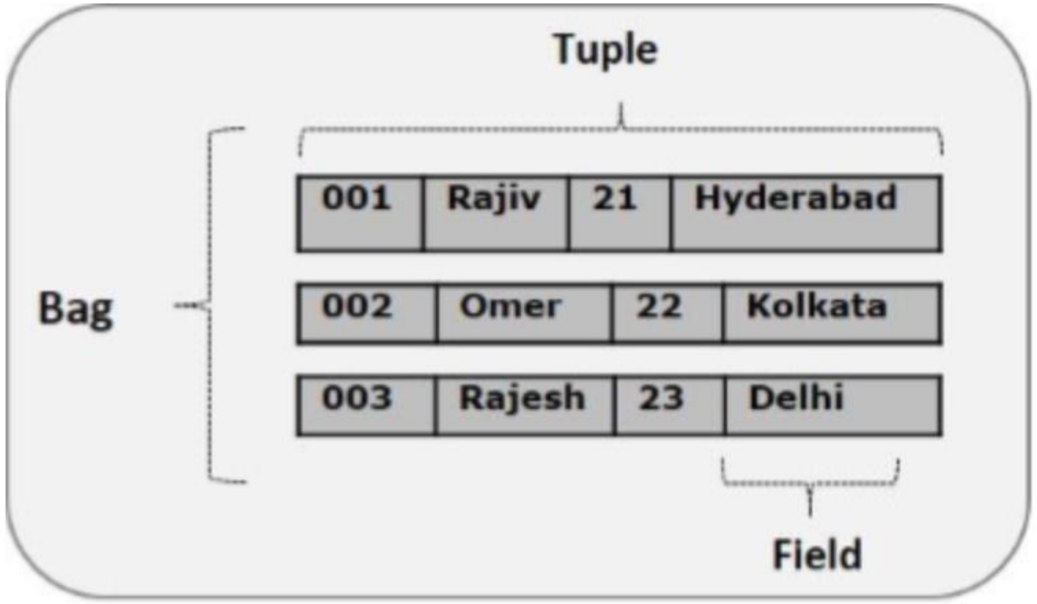
Architecture

- Programs are written in Pig Latin - a scripting language
- Pig Latin scripts are processed by the following components
  - ✓ Parser: Creates a DAG with syntactic and type checks. Nodes are operator and edges are data flows. This is a logical plan.
  - ✓ Optimizer: Removes unnecessary data or columns.
  - ✓ Compiler: Generates MR jobs and optimises the execution order.
  - ✓ Execution engine: Run the MR jobs on the Hadoop platform
  - ✓ Execution mode: Can work in local JVM for dev/test or by default on Hadoop cluster



Data Model

- Atom: Basic data types like byte, int, float, byte array.
  - ✓ e.g. andy, 67, …
- Tuple: Ordered set of fields of various types, like a row in RDBMS table.
  - ✓ e.g. (andy, 25)
- Bag: An unordered set of tuples. A bag can be inside a tuple - called an inner bag.
  - ✓ e.g. {(andy, 65), (ram, 23, 6000)}
  - ✓ e.g. inner bag: (1,{(1,2,3)})
- Map: A key-value pair, where a key is a char array type and value can be any type.
  - ✓ e.g. [name#andy, age#25]
- Relation: A bag but not an inner bag which can be inside a tuple. Like a table in RDBMS.


Tuple
Bag
001 | Rajiv | 21 | Hyderabad
002 | Omer | 22 | Kolkata
003 | Rajesh | 23 | Delhi
Field

PIG Analysis scenario 1: Count in groups

Count the number of transactions by item type for sales in India

```
grunt> geosales = LOAD 'geosales.csv' using PigStorage(',') as
(index:long,region:chararray,country:chararray,item_type:chararray,sales_channel:chararray,order
_priority:chararray,order_date:Datetime,order_id:chararray,ship_date:Datetime,units_sold:int,unit_
price:float,unit_cost:float,total_revenue:double,total_cost:double,total_profit:double);
grunt> india_data = FILTER geosales BY country=='India';
grunt> groupbyitem = group india_data by item_type;
grunt> countbyitem = FOREACH groupbyitem GENERATE CONCAT((chararray)$0,
CONCAT(':',(chararray)COUNT($1)));
grunt> STORE countbyitem INTO 'pig-output' USING PigStorage('\t');
```

```
Meat:39
Cereal:41
Fruits:53
Snacks:34
Clothes:48
Baby Food:59
Beverages:42
Cosmetics:52
Household:33
Vegetables:43
Personal Care:48
Office Supplies:48
```

PIG Analysis scenario 2: Basic stats

Avg/Max/Min/Sum revenue by item type for sales in India

```
grunt> revenue_data = FOREACH geosales GENERATE country, item_type, total_revenue ;
grunt> india_revenue_data = FILTER revenue_data BY country=='India';
grunt> india_item_groups = GROUP india_revenue_data by item_type;
grunt> avg_revenue_by_item = FOREACH india_item_groups GENERATE $0,
AVG(india_revenue_data.total_revenue);
grunt> describe avg_revenue_by_item;
avg_revenue_by_item: {group: chararray,double}
grunt> dump avg_revenue_by_item;
```

(Meat	2275631.3892307696)
(Cereal	1116549.6341463416)
(Fruits	46948.560000000005)
(Snacks	926205.4764705882)
(Clothes	528935.69)
(Baby Food	1401703.538983051)
(Beverages	240074.4047619048)
(Cosmetics	2583179.3846153845)
(Household	3683585.242424242)
(Vegetables	745693.3934883721)
(Personal Care	394076.519375)
(Office Supplies	3242835.86375)

PIG Analysis scenario 3: Sort

Top revenue item in India / Sort items by revenue

```
grunt> revenue_data = FOREACH geosales GENERATE country, item_type,
total_revenue ;

grunt> india_revenue_data = FILTER revenue_data BY country=='India';

grunt> india_item_groups = GROUP india_revenue_data by item_type;

grunt> item_grand_totals = FOREACH india_item_groups GENERATE $0,
SUM(india_revenue_data.total_revenue) as grand_total;

grunt> sorted_items = ORDER item_grand_totals BY grand_total DESC;
```

Topics for today

- Hadoop ecosystem technologies
  - ✓Pig
  - ✓**Hive**
  - ✓HBase

Learn basics about

A. Applicability

B. Architecture

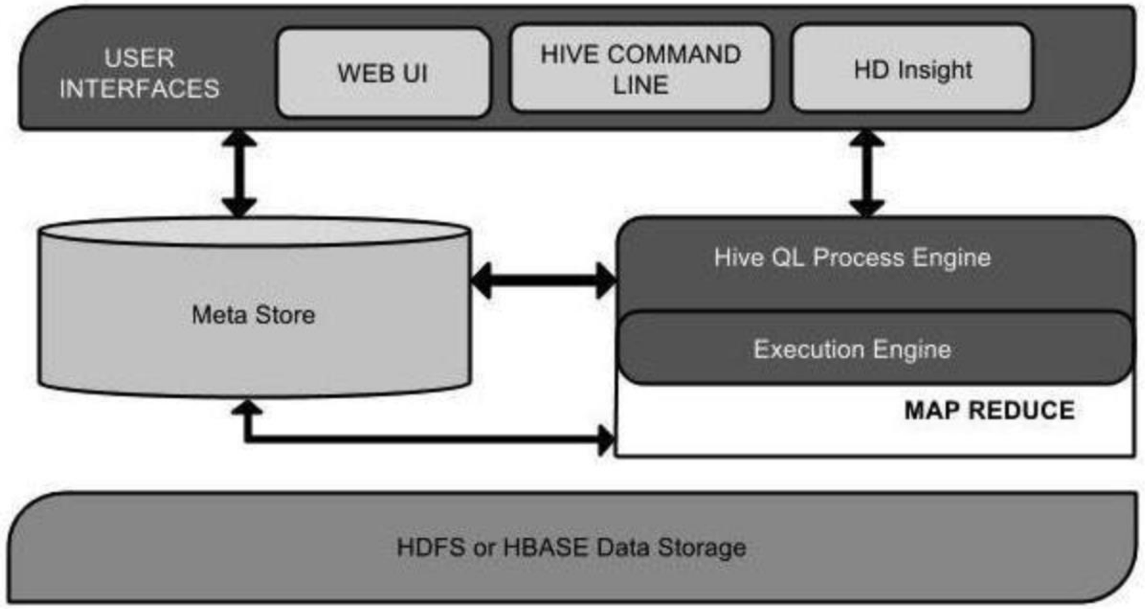
C. Data manipulation usage

Why Hive

- Provides a way to process large structured data in Hadoop
- Data Warehouse on Hadoop / HDFS
- SQL like query interface
  - ✓ But it is not an RDBMS with transactions
  - ✓ Not for real-time queries with row level updates
- Meant for OLAP type queries
- Initially in Facebook and now an Apache project

Hive Architecture

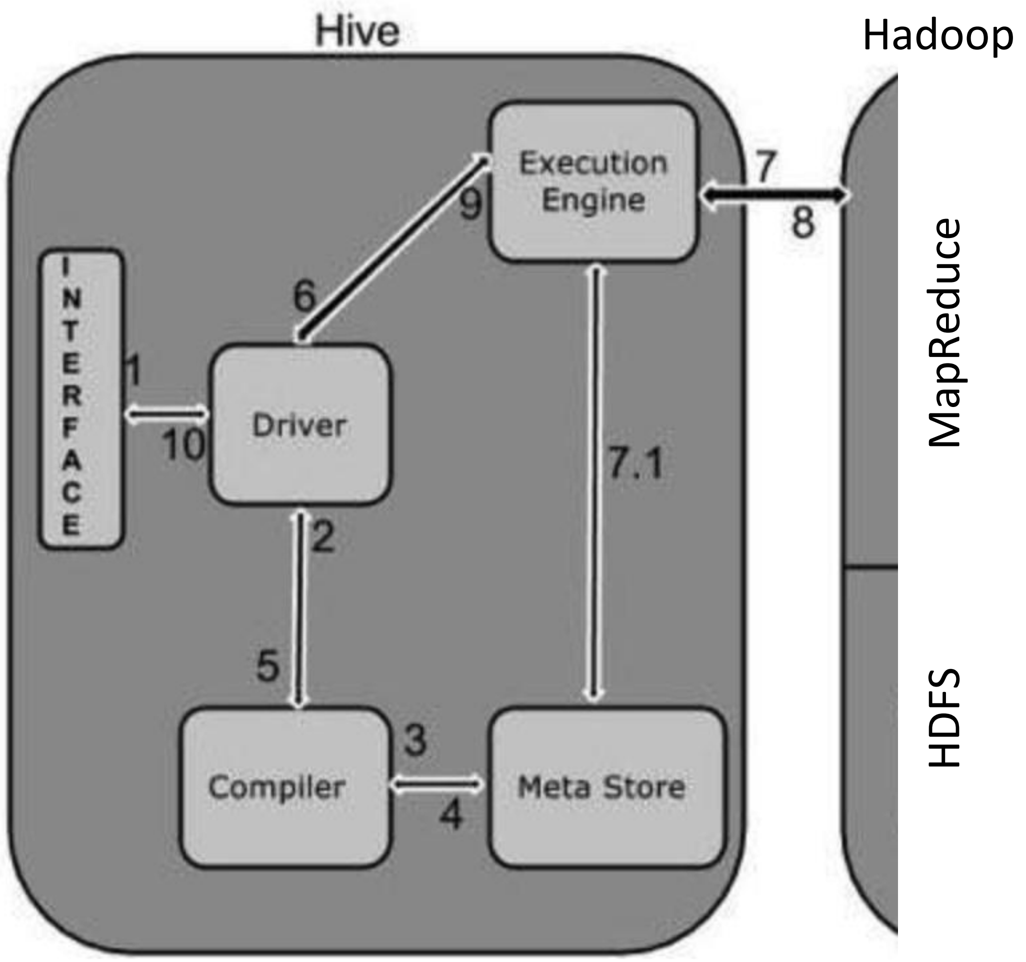
- Meta Store: Meta-data or schema is stored in a relational DB (default Derby, MySQL, ..) containing structure of tables, partitions, columns, data types, [de]-serializers to read/write data, HDFS files or HBase meta-data where data is stored.
- Provides SQL-like interface called HiveQL
- Execution engine with a Compiler (in HiveQL process engine) that consults meta-data to translate a query into MapReduce jobs
- Data in HDFS or in HBase



https://www.tutorialspoint.com/hive/hive_introduction.htm

Hive Query execution flow

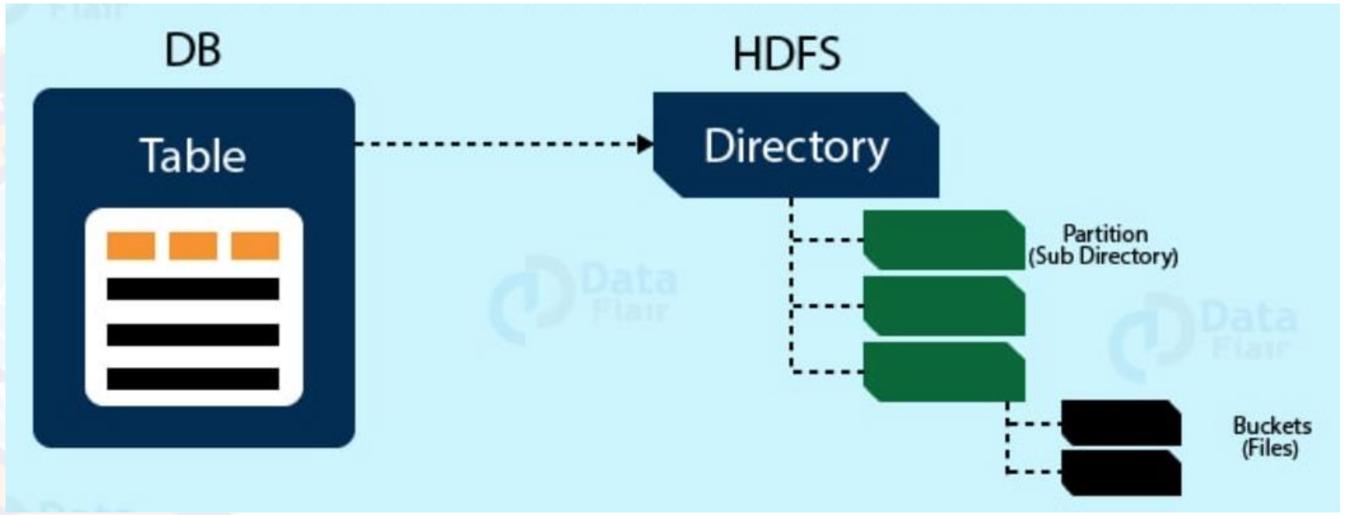
1. HiveQL query sent to driver from Interface.
2. Driver sends to Compiler to create a query plan
3. Compiler builds an abstract syntax tree (AST) and then semantic analysis of the query to create a query plan graph (DAG). For this it consults the Meta Store.
4. Meta Store responds to Compiler on requests.
5. Compiler (post optimization of the DAG) responds to Driver with the final plan.
   - ✓ Possible optimizations are combining consecutive joins, splitting tasks - e.g. combiners before reducer etc.
6. Driver sends to Execution Engine that essentially executes the query plan via Hadoop MapReduce + data in HDFS or HBase and finally result is sent to Hive interface. (steps 7-10)



https://www.tutorialspoint.com/hive/hive_introduction.htm

Data model

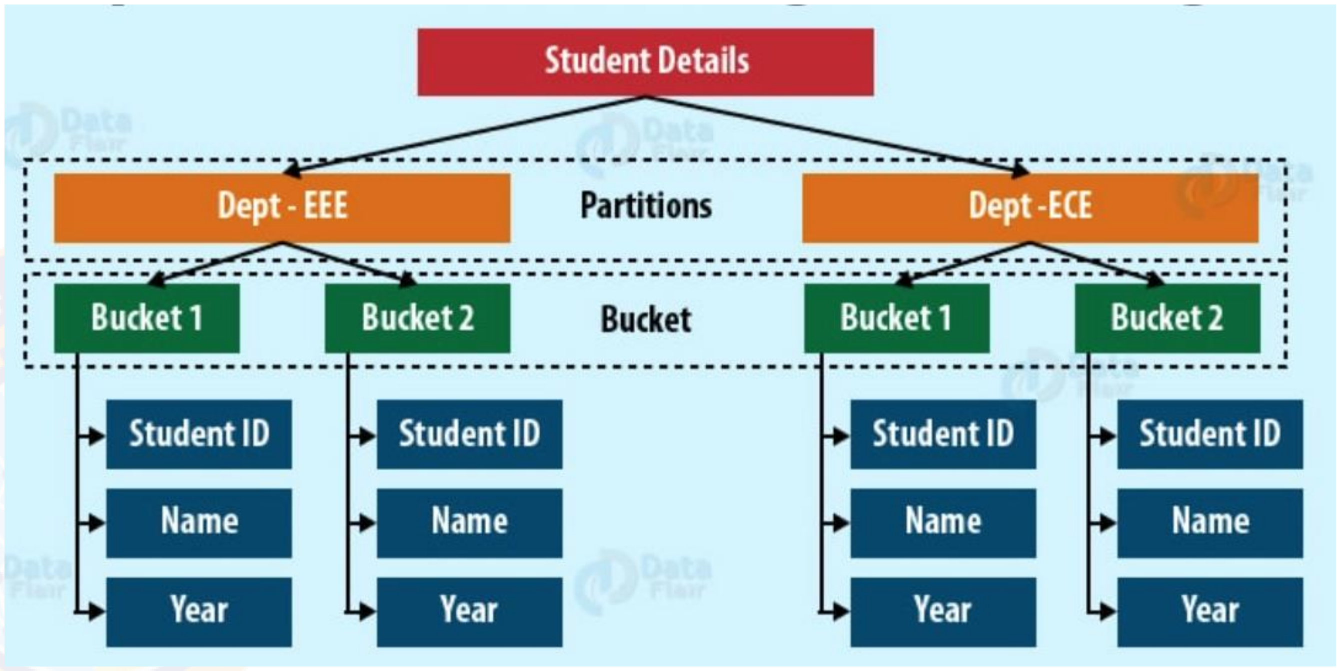
- Table
  - ✓ Same as RDBMS tables
  - ✓ Data is stored in a HDFS directory
  - ✓ Managed tables: Hive stores and manages in warehouse dir
  - ✓ External tables: Hive doesn't manage but records the path. So actual data can be brought in after create external table in Hive.
  - ✓ Tables are split into Partitions and then into Buckets



https://data-flair.training/blogs/hive-data-model/

Data model

- Partition
  - ✓ When creating a table a key can be used to split data into partitions - implemented as separate sub-dirs with table dir on HDFS
  - ✓ e.g. student data partitioned by dept id
- Bucket
  - ✓ Additional level of sub-division within a partition based on hash of some column to make some queries efficient
  - ✓ These map to actual files on HDFS / HBase



CREATE TABLE student_details (id int, name varchar(50), year int) PARTITIONED BY (dept varchar(3)) CLUSTERED BY (year) SORTED BY (id ASC) INTO N BUCKETS

↑

put year in same bucket but control total bucket count to keep their sizes balanced

https://data-flair.training/blogs/hive-data-model/

Data partitioning vs bucketing

Partitioning

- Distributes execution load horizontally.
  - Map tasks can be given partitions
- Faster execution of queries with the low volume of data takes place. E.g. search population from Vatican City returns faster than China

However,

- May have too many small partitions with too many directories.
- Effective for low volume data. But there some queries like group by on high volume of data take a long time to execute. For e.g., grouping population of China will take longer than Vatican City.

Bucketing

- Done when partition sizes vary a lot or there are too many partitions
- Provides faster query response within smaller segments of data. Like further indexing beyond partition keys.
- Almost equal volumes of data in each bucket — so joins at Map side will be quicker.
- Enables pre-sorting on smaller at data sets. Makes Map side merge sorts even more efficient.

However,

- Can define a number of buckets during table creation. But loading of an equal volume of data has to be done manually* by programmers.

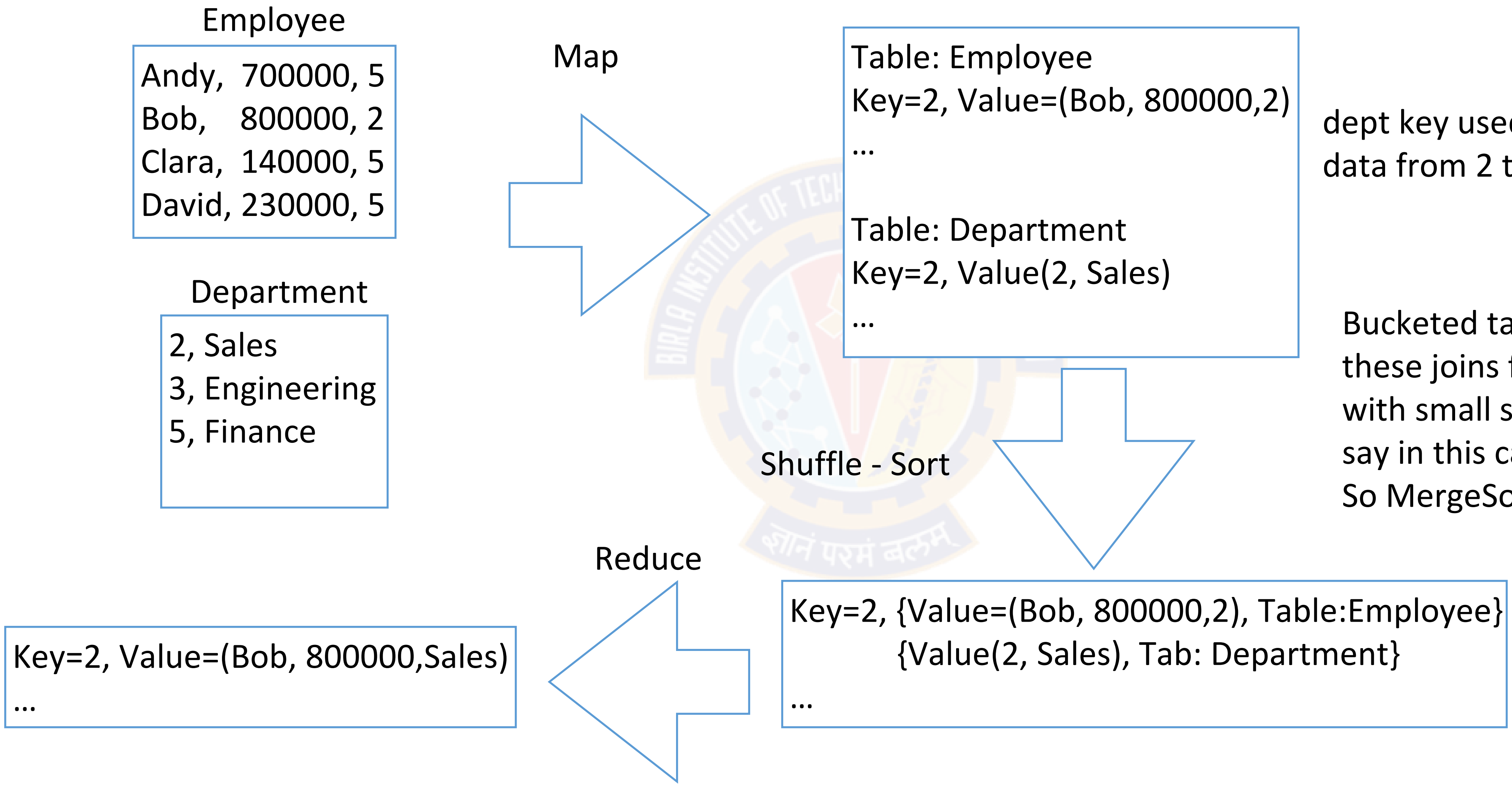
* Like ETL - this is a Data Warehouse !

Example of Partitions and Buckets

```
CREATE TABLE bucketed_user (
       firstname VARCHAR(64),
        lastname  VARCHAR(64),
        address   STRING,
        city  VARCHAR(64),
        state  VARCHAR(64),
        post      STRING,
        phone1    VARCHAR(64),
        phone2    STRING,
        email     STRING,
        web       STRING
)
COMMENT ‘A bucketed sorted user table’
PARTITIONED BY (country VARCHAR(64))
CLUSTERED BY (state) SORTED BY (city) INTO 32 BUCKETS
STORED AS SEQUENCEFILE;
```

Create partitions based on country
Group state records into same bucket
Control total number of buckets
Store as binary file to save space

A point about Map-side joins and buckets



Employee

Andy, 700000, 5
Bob, 800000, 2
Clara, 140000, 5
David, 230000, 5

Department

2, Sales
3, Engineering
5, Finance

Map

Table: Employee
Key=2, Value=(Bob, 800000,2)
…

Table: Department
Key=2, Value(2, Sales)
…

Shuffle - Sort

Key=2, {Value=(Bob, 800000,2), Table:Employee}
{Value(2, Sales), Tab: Department}
…

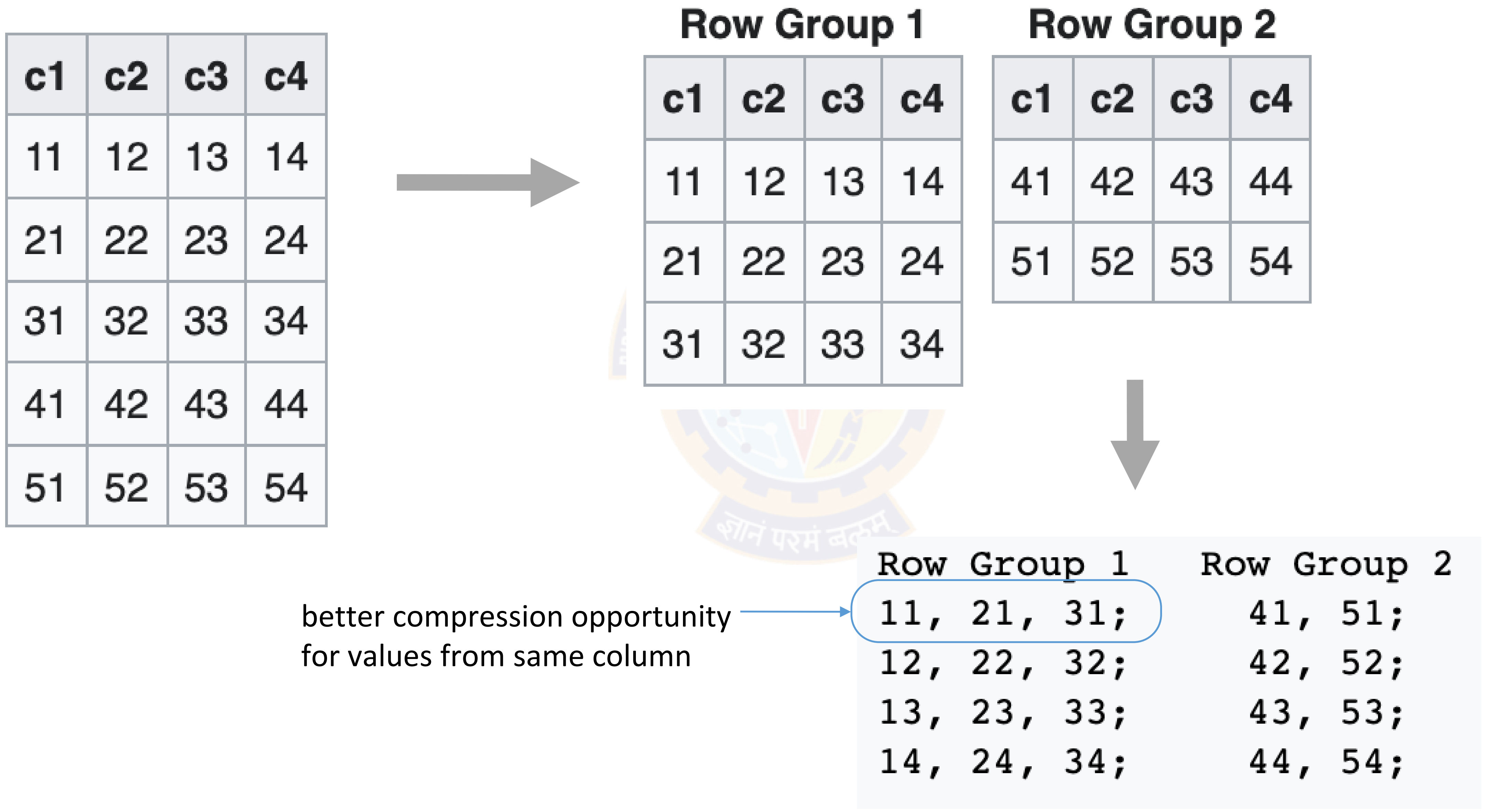
Reduce

Key=2, Value=(Bob, 800000,Sales)
…

dept key used as join key for data from 2 tables

Bucketed tables helps to make these joins faster in memory with small sized files sorted, say in this case by DeptID. So MergeSort works faster.

Storage format - Record Columnar file (RC file or ORC file)



c1	c2	c3	c4
11	12	13	14
21	22	23	24
31	32	33	34
41	42	43	44
51	52	53	54

Row Group 1

c1	c2	c3	c4
11	12	13	14
21	22	23	24
31	32	33	34

Row Group 2

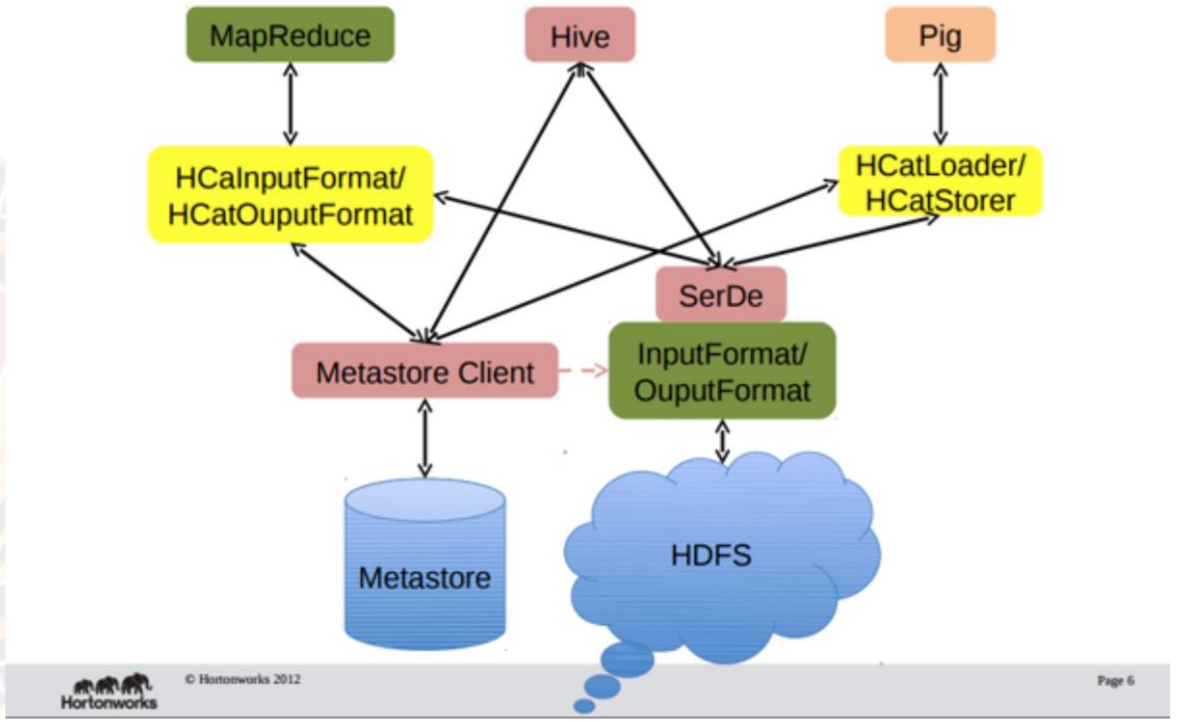
c1	c2	c3	c4
41	42	43	44
51	52	53	54

```
Row Group 1    Row Group 2
11, 21, 31;      41, 51;
12, 22, 32;      42, 52;
13, 23, 33;      43, 53;
14, 24, 34;      44, 54;
```

better compression opportunity
for values from same column

Broader use of Hive meta-store: HCatalog service

- Hive uses a Meta Store for structured data in tables with data on HDFS
- MapReduce programs, Pig scripts can independently access HDFS
- HCatalog: Hive MetaStore exposed as a REST service for other Hadoop ecosystem tech to use
  - ✓ E.g. Hadoop MapReduce Java code can use HCatalog HCatLoader and HCatStore implementations of Hadoop InputFormat and OutputFormat.
  - ✓ So read data from Hive and save results back to Hive
- Can be used to coordinate tasks working on same data across MR and Pig



```
a = LOAD 'hdfs://localhost:9000/user/hadoop/sales.csv'
USING PigStorage(',') AS
(shop:chararray,employee:chararray,sales:int);
```

⇨

```
a = LOAD 'sales.csv' using HCatLoader();
```

Pig Vs Hive

Pig	Hive
Procedural Data Flow Language	Declarative SQLish Language
For Programming	For creating reports
Mainly used by Researchers and Programmers	Mainly used by Data Analysts
Operates on the client side of a cluster.	Operates on the server side of a cluster.
Does not have a dedicated metadata database.	Makes use of exact variation of dedicated SQL DDL language by defining tables beforehand.
Pig is SQL like but varies to a great extent.	Directly leverages SQL and is easy to learn for database experts.
Pig supports Avro file format.	Hive does not support it.

Topics for today

- Hadoop ecosystem technologies
  - ✓ Pig
  - ✓ Hive
  - ✓ **HBase**

Learn basics about

A. Applicability

B. Architecture

C. Data manipulation usage

HBase

- HBase is the Hadoop database
- HBase is a NoSQL database which is:
  - ✓Consistent (C)
  - ✓Partition Tolerant (P)
- It is a key value store where we can have the value and a key
- In HBase we have got the concepts of table and columns
- Columns are grouped together into column families
- To access columns of a single column family, it makes it faster as it does not scan other column families
- Use HBase when you need random, real time read / write access to Bigdata

Why HBase ?

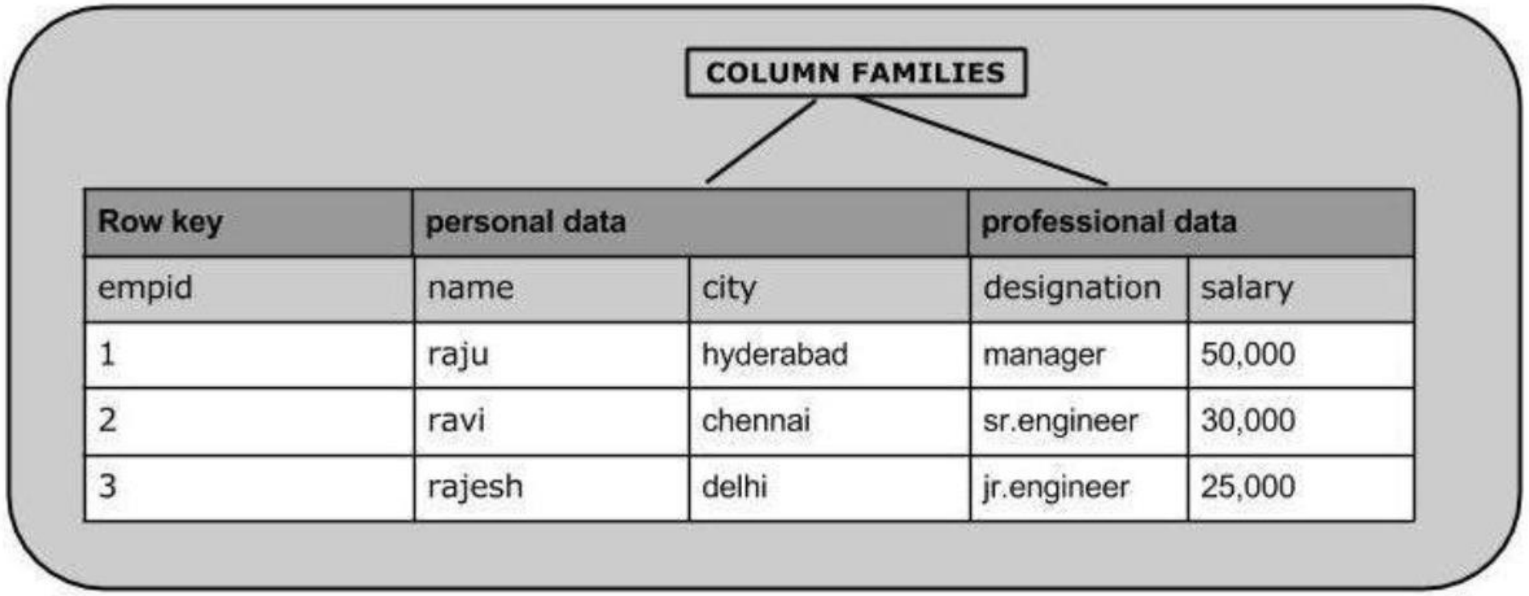
- HDFS provides sequential access to data
- HBASE provides random access capability of large files in HDFS
    - ✓ Hash tables used for indexing of HDFS files
- Key-value store with no fixed schema as in RDBMS
    - ✓ so can be used for structured and semi-structured data
    - ✓ type of NoSQL database
- Built for wide tables with many attributes
    - ✓ de-normalized data
- Column oriented storage
    - ✓ Tuned for analytical queries that access specific columns
- Strongly consistent because read is on latest write of a data item
- Origins of idea from Google BigTable (on GFS) leading to a Hadoop project (on HDFS)

When to use HBase

- Large data volume where many distributed nodes are needed, e.g. 100s of millions of rows
- Need to lookup data in a large data store
  - ✓ that's the key aspect of HBase on HDFS
- Need linear scalability with data volume
- None of these are required : transactional guarantees, secondary indices, DB triggers, complex queries
- Can add enough commodity hardware to scale with favourable cost-performance ratio

Columnar storage

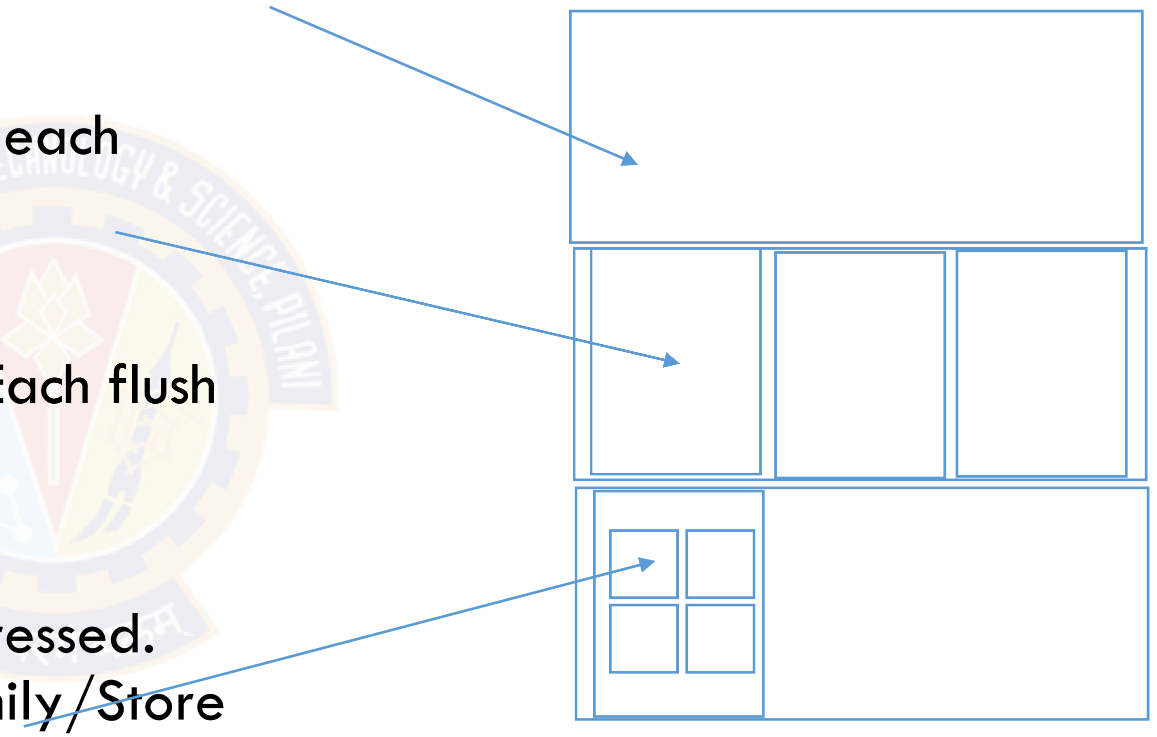
- Data in a row is a collection of column families with each column being a key-value pair
- A column values are stored together on disk
- Each cell value also has a timestamp



Row key	personal data		professional data	
empid	name	city	designation	salary
1	raju	hyderabad	manager	50,000
2	ravi	chennai	sr.engineer	30,000
3	rajesh	delhi	jr.engineer	25,000

HBase storage structure

- Region - Continuous sorted set of rows stored together (using row-key to sort).
  - ✓ HBase tables are split into regions.*
- A region has multiple Column Families with each column family having a Store
- Each Store has
  - ✓ MemStore**: Write buffer for clients. Each flush creates new StoreFile/HFile on disk.
  - ✓ StoreFiles can be compacted
- Each StoreFile has Blocks that can be compressed. Blocksize can be configured per ColumnFamily/Store level.

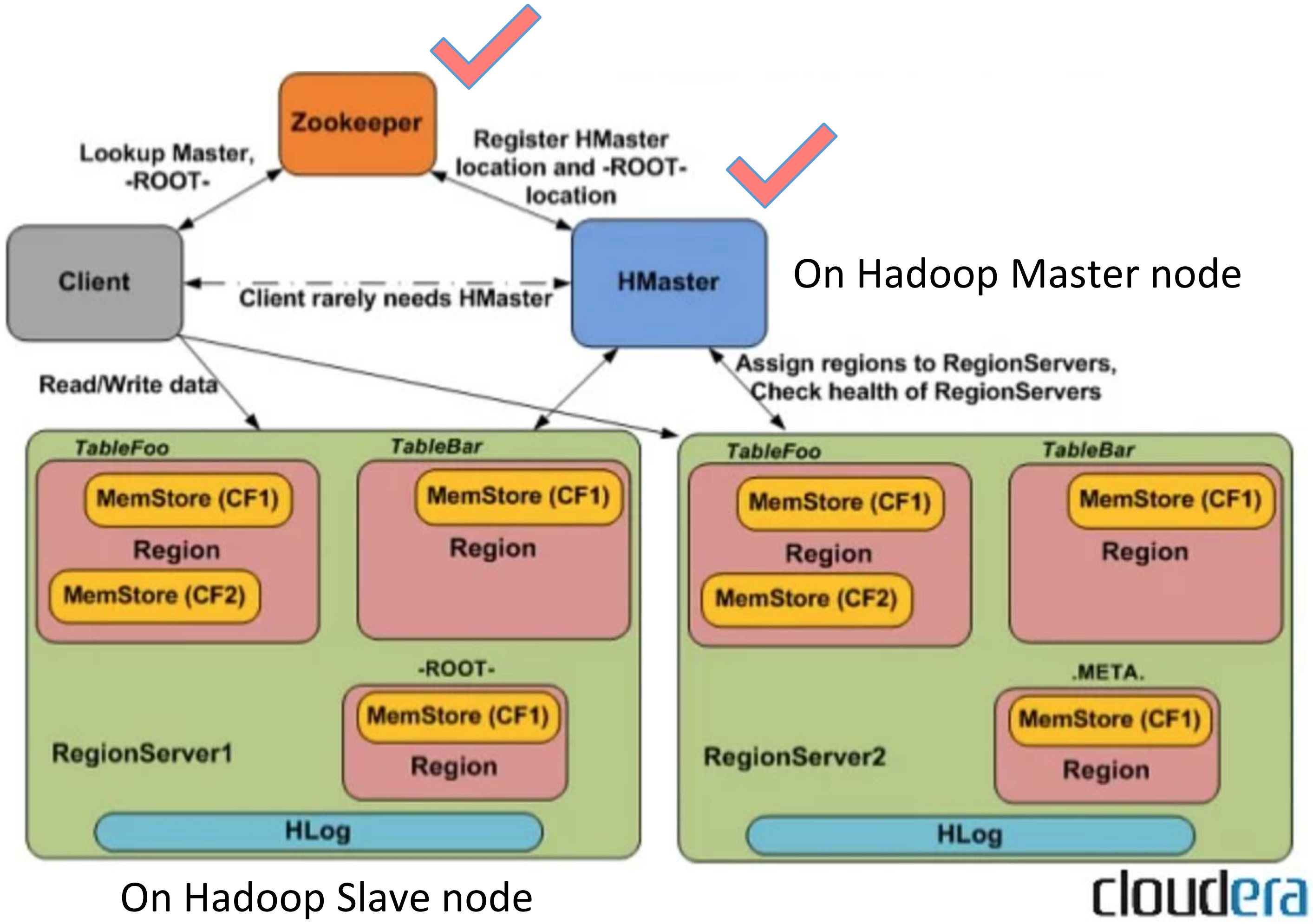


** configured in : hbase-site.xml/hbase.hregion.memstore.flush.size

* configured in: hbase.hregion.max.filesize

Architectural components - Master

- HMaster: Single controller on master node.
  - ✓ Assigns regions to RegionServers and checks health.
  - ✓ Failover control
  - ✓ DDL / meta-data operations
- Zookeeper cluster (separate nodes):
  - ✓ Client communication
  - ✓ Track failures with heartbeat
  - ✓ Maintain cluster config data

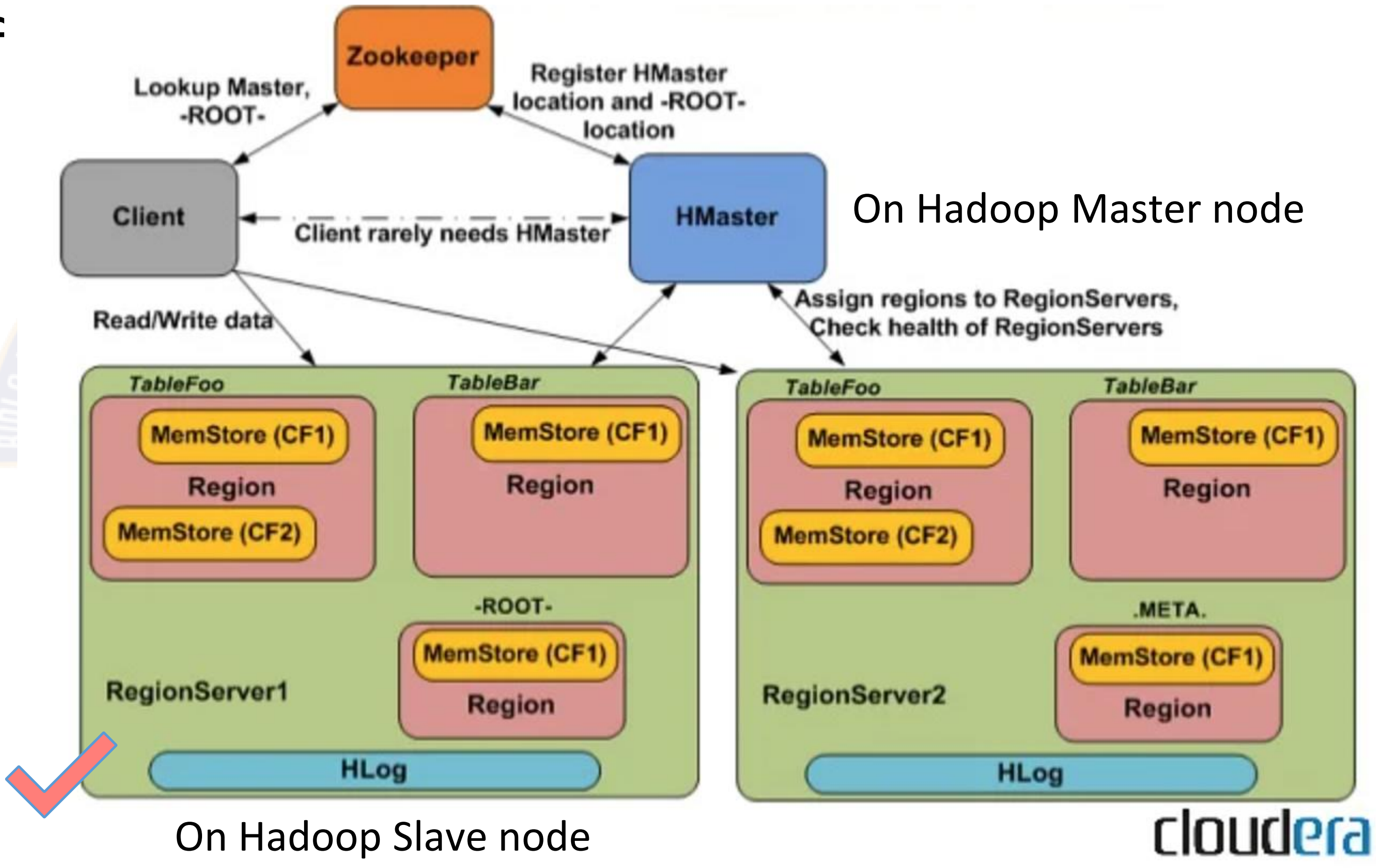


On Hadoop Slave node

CF : Column Family

Architectural components - Slaves

- RegionServer: In-charge of set of regions on a slave node / HDFS DataNode
  - ✓ Worker nodes handling read/write/delete requests from clients
  - ✓ HLog or Write Ahead Logs (WAL) for MemStore (look in hbase/WALs/ on HDFS)



On Hadoop Slave node

CF : Column Family

Find HBase objects on HDFS

```
For data:
/hbase
     /<Table> (Tables in the cluster)
          /<Region>  (Regions for the table)
               /<ColumnFamily> (ColumnFamilies for the Region)
                    /<StoreFile>  (StoreFiles for the ColumnFamily)
For WAL logs:
/hbase
     /.logs
          /<RegionServer>    (RegionServers)
               /<HLog>           (WAL HLog files for the RegionServer)

To see list of regions and utilisation per region of a table:
hadoop fs -du /hbase/<table name>
```

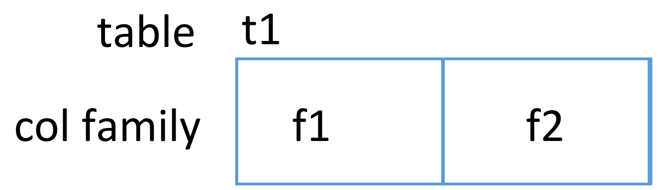
Find HBase objects on HDFS - Example

```
> ls -l $HOME/hbase/hbase-2.4.4/bin/tmp/hbase
drwxr-xr-x   6 anindya  staff   192 Jun 27 21:44 MasterData
drwxr-xr-x   3 anindya  staff    96 Jun 28 10:30 WALs
drwxr-xr-x   2 anindya  staff    64 Jun 27 21:44 archive
drwxr-xr-x   2 anindya  staff    64 Jun 27 21:44 corrupt
drwxr-xr-x   4 anindya  staff   128 Jun 27 21:44 data
-rw-r--r--   1 anindya  staff    42 Jun 27 21:44 hbase.id
-rw-r--r--   1 anindya  staff     7 Jun 27 21:44 hbase.version
drwxr-xr-x   2 anindya  staff    64 Jun 27 21:44 mobdir
drwxr-xr-x  62 anindya  staff  1984 Jun 29 01:14 oldWALs
drwx--x--x   2 anindya  staff    64 Jun 27 21:44 staging
```

Configured path for HBase files

Write Ahead Logs

Data files



```
> ls -l $HOME/hbase/hbase-2.4.4/bin/tmp/hbase/data/default

drwxr-xr-x  5 anindya  staff  160 Jun 27 21:44 t1
```

Table

```
> ls -l /Users/anindya/hbase/hbase-2.4.4/bin/tmp/hbase/data/default/t1/e35e1cb65f5fd84bb4201353d1764365
drwxr-xr-x  3 anindya  staff  96 Jun 29 08:10 f1
drwxr-xr-x  3 anindya  staff  96 Jun 27 23:05 f2
drwxr-xr-x  3 anindya  staff  96 Jun 28 10:30 recovered.edits
```

Column families

```
> ls -l /Users/anindya/hbase/hbase-2.4.4/bin/tmp/hbase/data/default/t1/e35e1cb65f5fd84bb4201353d1764365/f1

-rw-rw-rw-  1 anindya  staff  4891 Jun 27 22:53 01d6868c5ed4426393ab08815971b021
```

HDFS file storing a block

HBase DML using shell - create

```
> create 't1', 'f1', ‘f2'
> describe 't1'
Table t1 is ENABLED
t1
COLUMN FAMILIES DESCRIPTION
{NAME => 'f1', BLOOMFILTER => 'ROW', IN_MEMORY => 'false', VERSIONS => '1',
KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', COMPRESSION
=> 'NONE', TTL => 'FOREVER', MIN_VERSIONS => '0', BLOCKCACHE => 'true',
BLOCKSIZE => '65536', REPLICATION_SCOPE => '0'}

{NAME => 'f2', BLOOMFILTER => 'ROW', IN_MEMORY => 'false', VERSIONS => '1',
KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', COMPRESSION
=> 'NONE', TTL => 'FOREVER', MIN_VERSIONS => '0', BLOCKCACHE => 'true',
BLOCKSIZE => '65536', REPLICATION_SCOPE => '0'}

2 row(s)
Quota is disabled
Took 0.2327 seconds
```

HBase DML using shell - updates / inserts

row key

```
> put 't1', 1, 'f1:name', ‘andy'
> put 't1', 1, 'f1:name', 'ram'
> put 't1', 1, 'f2:salary', 10000
> scan 't1'
ROW  COLUMN+CELL
 1   column=f1:name, timestamp=2021-06-27T21:54:17.434, value=ram
 1   column=f2:salary, timestamp=2021-06-27T22:02:22.730, value=10000
1 row(s)
> put 't1', 2, 'f1:name', ‘andy’
> put 't1', 2, 'f2:salary', 20000
> scan ‘t1’
ROW  COLUMN+CELL
 1   column=f1:name, timestamp=2021-06-27T21:54:17.434, value=ram
 1   column=f2:salary, timestamp=2021-06-27T22:02:22.730, value=10000
 2   column=f1:name, timestamp=2021-06-27T22:04:26.820, value=andy
 2   column=f2:salary, timestamp=2021-06-27T22:05:29.310, value=20000
2 row(s)
```

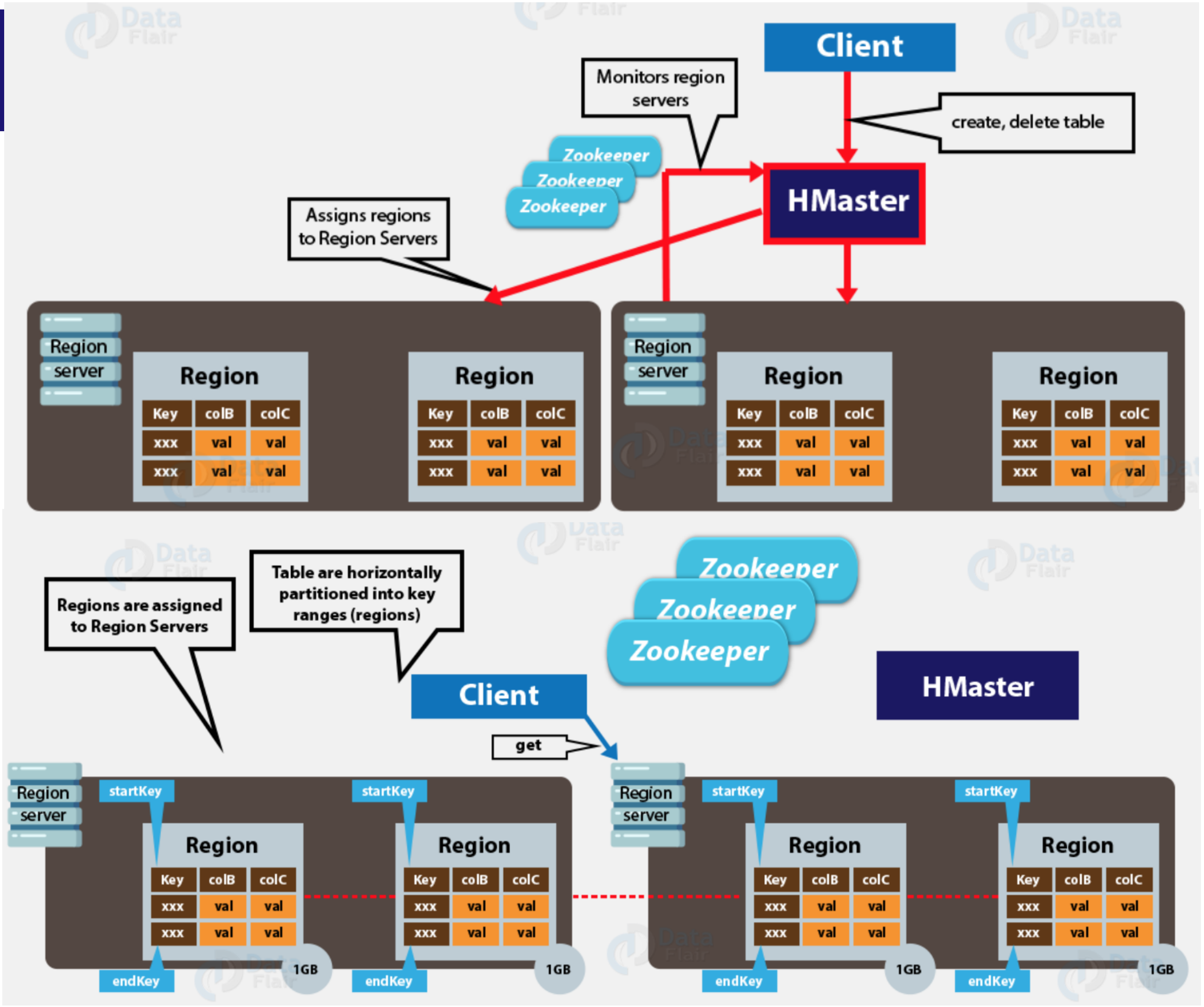
HBase DML using shell - filtering

```
> import org.apache.hadoop.hbase.filter.SingleColumnValueFilter
> import org.apache.hadoop.hbase.filter.CompareFilter
> import org.apache.hadoop.hbase.filter.BinaryComparator


> scan 't1', {COLUMNS=>['f1:name','f2:salary'],
FILTER=>SingleColumnValueFilter.new(Bytes.toBytes('f2'),Bytes.toBytes('salary'),Co
mpareFilter::CompareOp.valueOf('GREATER'),BinaryComparator.new(Bytes.toBytes('1200
0')))}
ROW  COLUMN+CELL
 2   column=f1:name, timestamp=2021-06-27T22:04:26.820, value=andy
 2   column=f2:salary, timestamp=2021-06-27T22:05:29.310, value=20000
1 row(s)
Took 0.0170 seconds
```

Client operations

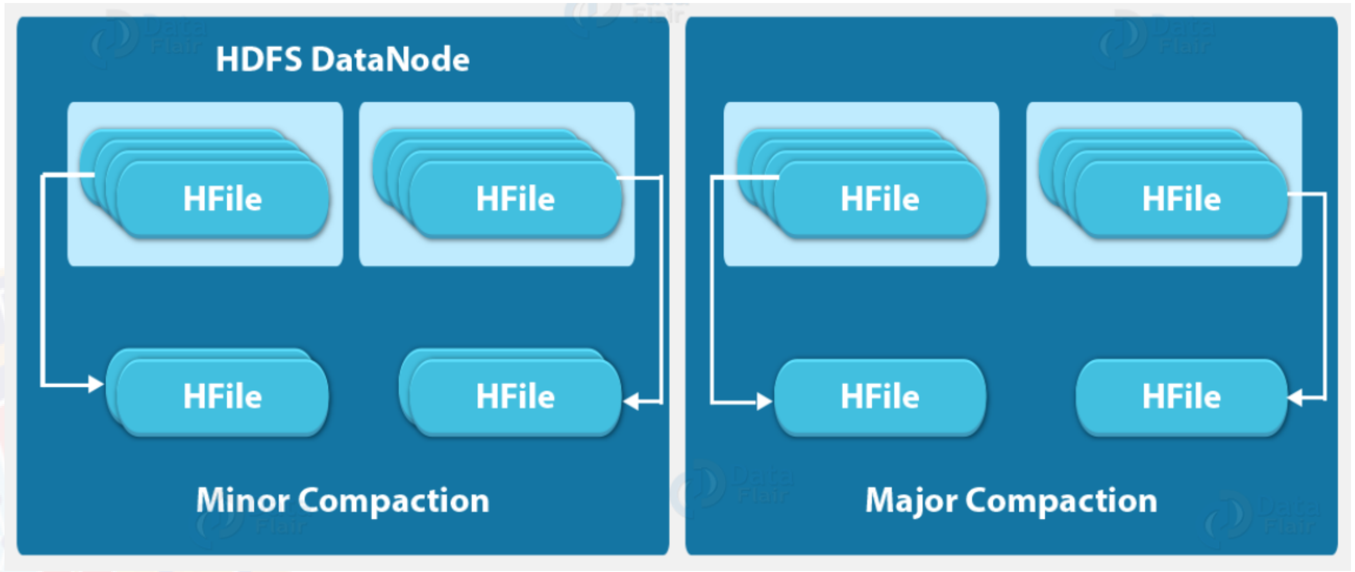
- Create / Delete via HMaster
- Meta-data from HMaster
- Actual Read / Write via RegionServer



https://data-flair.training/blogs/hbase-architecture/

File compaction

- Regions get split on the same node when they grow large.
- Typically RegionServers manage all regions on same node.
- Load balancing, HDFS replication, or failure may have RegionServers managing regions with non-local StoreFiles.
- Compaction is about
  - ✓ merging StoreFiles into larger files
  - ✓ merging Regions
  - ✓ making sure RegionServers and corresponding region files are on same node.



Merge smaller files into larger files on same ColumnFamily on same node

Merge all files into one for same Column Family possibly across nodes

Drops deleted cells also

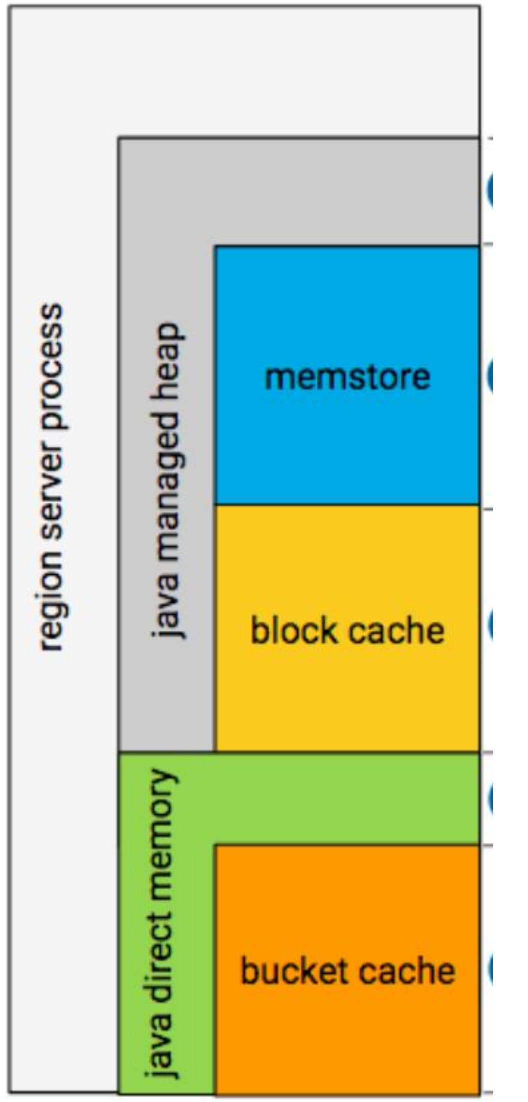
https://data-flair.training/blogs/hbase-architecture/

Splitting tables into Regions

- Automatic splitting happens based on size
- However in some cases, user may want to control splitting
  - ✓ Hot spots for data access
  - ✓ For real-time / time-series data, last region is always active
  - ✓ Load balancing
- hbase> create 'test_table', 'f1', SPLITS=> ['a', 'e', 'i', 'o', ‘u']
  - ✓ Splitting by sorted row keys starting with letters region1:a-d, region 2:e-h, …
- Can also merge regions:
  - ✓ hbase> merge_region region1 region2

Caching on RegionServer

- MemStore is for writes
- Recent edits can also be read from MemStore
- Otherwise read caching is done in BlockCache
- HFile is a collection of Blocks - so Block is the smallest unit of IO at HBase level
- HBase blocks are 4 types
  - ✓ DATA : Contain user data
  - ✓ META : Contain meta-data about the HFile
  - ✓ INDEX, BLOOM: Index DATA blocks to access cell level
- Types of caches in recent HBase
  - ✓ LRUBlockCache : Within JVM Heap - so will be impacted when GC runs
  - ✓ SlabCache, BucketCache : Can use memory outside Heap to avoid GC pressure



https://blog.cloudera.com/hbase-blockcache-101/

Approximate Sizing guidance

- 10 GB per region
- RegionServer : 100 regions x 10GB = 1 TB
  - ✓ Typical for PB data storage use cases
- Cache size: 10GB : 1% - which may barely cover block indices

Next Session:

More Hadoop ecosystem tech