Introduction to

1. Column (NoSQL) databases
2. Hbase – a distributed column DB

Learning Objectives

- Understand (at conceptual level):
  - What is a column database and how it is different from a relational/SQL database
  - What is Hbase
  - Concepts of Hbase table, column families, attributes, timestamp, regions
  - Relationship between Hbase and Hadoop
  - The implementation model of Hbase
- Can insert values and updates to Hbase on paper

Reading

- Chapter1 of "HBase: The Definitive Guide" by Lars George https://www.safaribooksonline.com/library/view/hbase-the-definitive/9781449314682/

Column (NoSQL) DB

Jixue Liu

An example

Logic table: word frequencies in tweets

TwID	Happy	Angry	Party	Election
1	2	0	1	0
2	0	1	0	1
3	0	0	2	1
4	1	0	0	0
5	0	0	0	1

Physically

* Relational

Values of a row are stored together

t, H, A, P, E
1, 2, 0, 1, 0
2, 0, 1 ,0,1
3, 0, 0, 2,1
4, 1, 0, 0, 0
5, 0, 0, 0,1

* Column (NoSQL)

Values of a column are stored together

Happy:	Angry:	Party:	Election:
2:[1]	1:[2]	1:[1]	1:[2,3,5]
1:[4]		2:[3]	

Each column is a hash table use word freq. as key: frq:[TwIDs] pairs

Benefit of Column DB (1)

How many tweets mentioned 'Angry' exactly once?

Physically

* Relational

Values of a row are stored together

t, H, A, P, E
1, 2, 0, 1, 0
2, 0, 1 ,0,1
3, 0, 0, 2,1
4, 1, 0, 0, 0
5, 0, 0, 0,1

- Read each record, test if Angry>0
- 5 reads

* Column (NoSQL)

Values of a column are stored together

Happy:	Angry:	Party:	Election:
2:[1]	1:[2]	1:[1]	1:[2,3,5]
1:[4]		2:[3]	

- Read Angry for freq=1 and find ID list [2] – only one id, so 1 tweet.
- 1 read

Benefit of Column DB (2)

How many tweets mentioned ‘Angry’ once and ‘Party’ twice?

Physically

* Relational
  Values of a row are stored together

t, H, A, P, E
1, 2, 0, 1, 0
2, 0, 1 ,0,1
3, 0, 0, 2,1
4, 1, 0, 0, 0
5, 0, 0, 0,1

- Read each record, test if Angry=1 and Party=2
- 5 reads

* Column (NoSQL)
  Values of a column are stored together

Happy:	Angry:	Party:	Election:
2:[1] 1:[4]	1:[2]	1:[1] 2:[3]	1:[2,3,5]

- Read Angry for frq=1, find [TwIDs] of [2]
- Read Party for frq=2, find [3]
- 2 reads.
- No common IDs, 0 tweet does this.

Benefit of Column DB (2)

How many tweets mentioned 'Angry' once and 'Election' once?

Physically

* Relational

Values of a row are stored together

t, H, A, P, E
1, 2, 0, 1, 0
2, 0, 1 ,0,1
3, 0, 0, 2,1
4, 1, 0, 0, 0
5, 0, 0, 0,1

- Read each record, test if Angry=1 & Election=1
- 5 reads

* Column (NoSQL)

Values of a column are stored together

Happy:	Angry:	Party:	Election:
2:[1] 1:[4]	1:[2]	1:[1] 2:[3]	1:[2,3,5]

- Read Angry for frq=1, find [2]
- Read Election for frq=1, find [2,3,5]
- 2 reads.
- Comm.IDs = [2], 1 tweet does this.

Benefit of Column DB – not always

How many tweets mentioned two of the four words? (each word at least once!)

Physically

* Relational

Values of a row are stored together

t, H, A, P, E
1, 2, 0, 1, 0
2, 0, 1 ,0,1
3, 0, 0, 2,1
4, 1, 0, 0, 0
5, 0, 0, 0,1

- Read and count (nested loop)
- 5 reads

* Column (NoSQL)

Values of a column are stored together

Happy:	Angry:	Party:	Election:
2:[1]	1:[2]	1:[1]	1:[2,3,5]
1:[4]		2:[3]	

- Read Happy to get [1,4]
- Read Angry to get [2]
- Read Party to get [1,3]
- Read Election to get [2,3,5]
- ID=1 are in two lists, mentions two words.
- Same to ID=2, and ID=3.
- So three tweets do this.
- 5 reads, hard to compute
- Becomes expensive when the number of columns increases.

Update Column DB

Make tweet 1 to have 2 mentions of Angry, and no Happy

* Relational

Values of a row are stored together

t, H, A, P, E
1, 0, 2, 1, 0
2, 0, 1 ,0,1
3, 0, 0, 2,1
4, 1, 0, 0, 0
5, 0, 0, 0,1

* Column (NoSQL)

Values of a column are stored together

Happy:	Angry:	Party:	Election:
~~2:[1]~~	1:[2]	1:[1]	1:[2,3,5]
1:[4]	2:[1]	2:[3]	

Insertion to Column DB

Insert tweets with ID 6: (6, 1, 0, 2,2)

* Relational

  Values of a row are stored together

t, H, A, P, E
1, 0, 2, 1, 0
2, 0, 1 ,0,1
3, 0, 0, 2,1
4, 1, 0, 0, 0
5, 0, 0, 0,1
6, 1, 0, 2, 2

* Column (NoSQL)

  Values of a column are stored together

Happy:	Angry:	Party:	Election:
~~2:[1]~~	1:[2]	1:[1]	1:[2,3,5]
1:[4,6]	2:[1]	2:[3,6]	2:[6]

Column DB vs SQL DB

- Column
- Transaction management difficult
- Good for keyword based search

- SQL
- Support transaction management
- Complex queries

HBase

Spring 2013
WPI, Mohamed Eltabakh
Modified by Jixue Liu

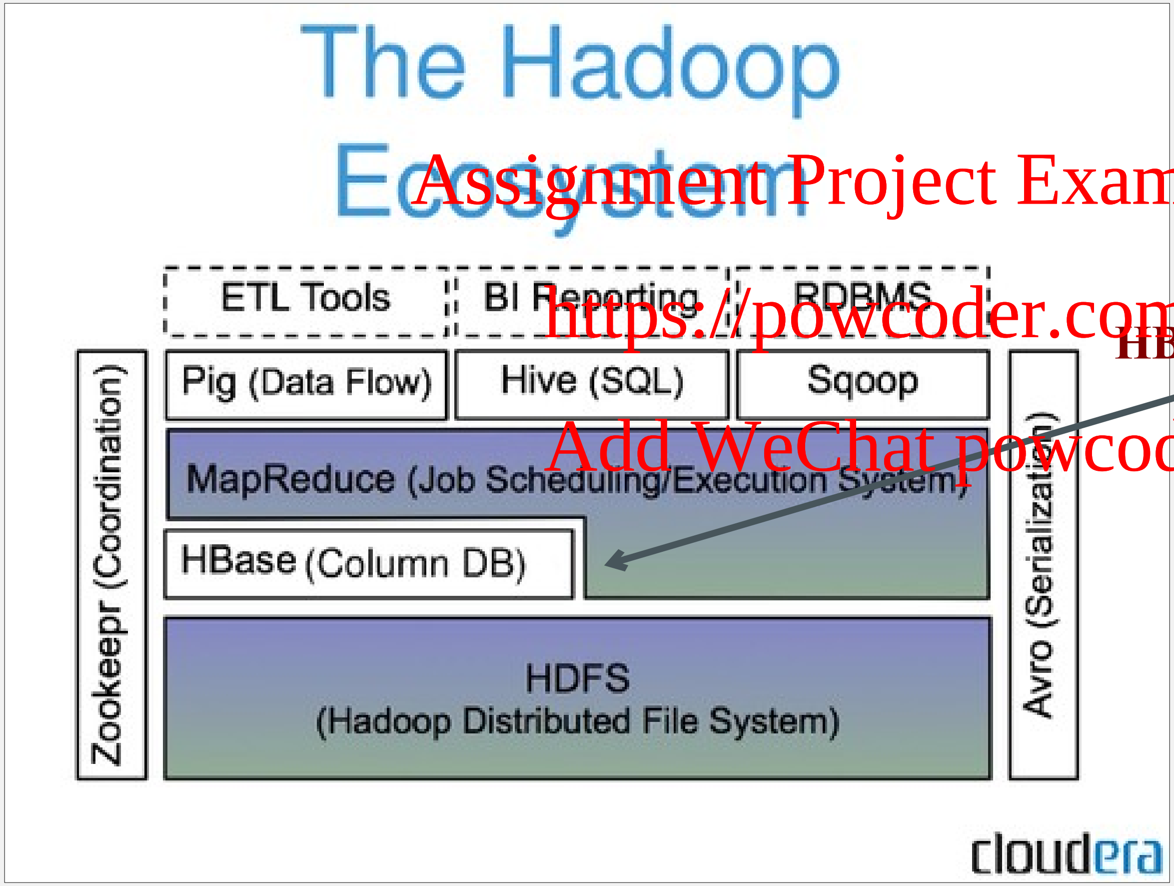
HBase: Overview

- **HBase is a distributed column-oriented data store built on top of HDFS**

- **HBase is an Apache open source project whose goal is to provide storage for the Hadoop Distributed Computing**

- **Data is logically organized into tables, rows and columns**

HBase: Part of Hadoop's Ecosystem



HBase is built on top of HDFS

HBase files are internally stored in HDFS

HBase Data Model

Hbase Logical model is a table – Hbase table

Headings are **attribute families, not attributes**.
Attributes are not at the schema level. **Attributes are part of data.**

Row key is primary key, and also the hash key

All data is of byte[ ]

One row can have multiple column families. Each column family can have many and different number of attribute:value pairs.

Row Key	Column Family “info”	Column Family “roles”
cutting	{‘height’:’9ft’, ‘state’:’CA’}	{‘ASF’:’Director’, ‘Hadoop’:’founder’}
tlipcon	{‘height’:’5ft7’, ‘state’:’CA’}	{‘Hadoop’:’Committer’@2010, ‘Hadoop’:’PMC’@2011, ‘Hive’:’Contributor’ }

Different rows have different attributes

In one cell, an attribute may have multiple values with different timestamps

Notes on Data Model

- HBase schema consists of several ***Tables – Hbase tables***
- Each table consists of a set of ***Column Families***
  - Attributes are not part of the schema
  - Because column names are encoded inside the cells
  - Different cells can have different columns
- Empty cells (if any) is not stored.
  - Cater for sparse data well.

Column Family "roles"
{'ASF':'Director', 'Hadoop':'founder'}
{'Hadoop':'Committer'@2010, 'Hadoop':'PMC'@2011, 'Hive':'Contributor' }

HBase Physical Model

Example: logical to physical

All cells (incl. versions) are collapsed down to <colFami:attr, ts, value> sorted

Row Key	Column Family "info"	Column Family "roles"
cutting	{'height':'9ft', 'state':'CA'}	{'ASF':'Director', 'Hadoop':'founder'}
tlipcon	{'height':'5ft7', 'state':'CA'}	{'Hadoop':'Committer'@2010, 'Hadoop':'PMC'@2011, 'Hive':'Contributor' }

Physical model

RowKey	colFami:attr	Time Stamp	Cell Value
cutting	info:height	7868	9ft
	info:state	4184	CA
	roles:ASF	7869	Director
	roles:Hadoop	7870	founder
tlipcon	info:height	7049	5ft7
	info:state	7446	CA
	roles:Hadoop	2011	PMC
	roles:Hadoop	2010	Committer
	roles:Hive	2011	contributor

HBase Regions

Each HTable is partitioned into ***regions.*** A region is stored on a node.

- Regions are counterpart to HDFS blocks

RowKey	colFami:atr	TimeStamp	CellValue
cutting	info:height	7868	9ft
	info:state	4184	CA
	roles:ASF	7869	Director
	roles:Hadoop	7870	founder
tlipcon	info:height	7049	5ft7
	info:state	7446	CA
	roles:Hadoop	2011	PMC
	roles:Hadoop	2010	Committer
	roles:Hive	2011	contributor

Form a region

Next two form another region

……

Assignment Project Exam Help

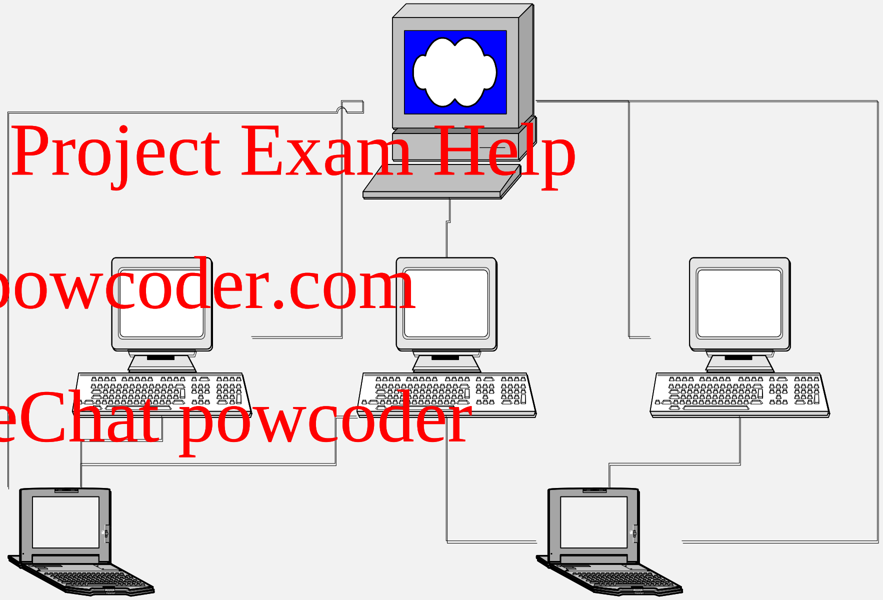
https://powcoder.com

HBase Architecture

Add WeChat powcoder

Three Major Components

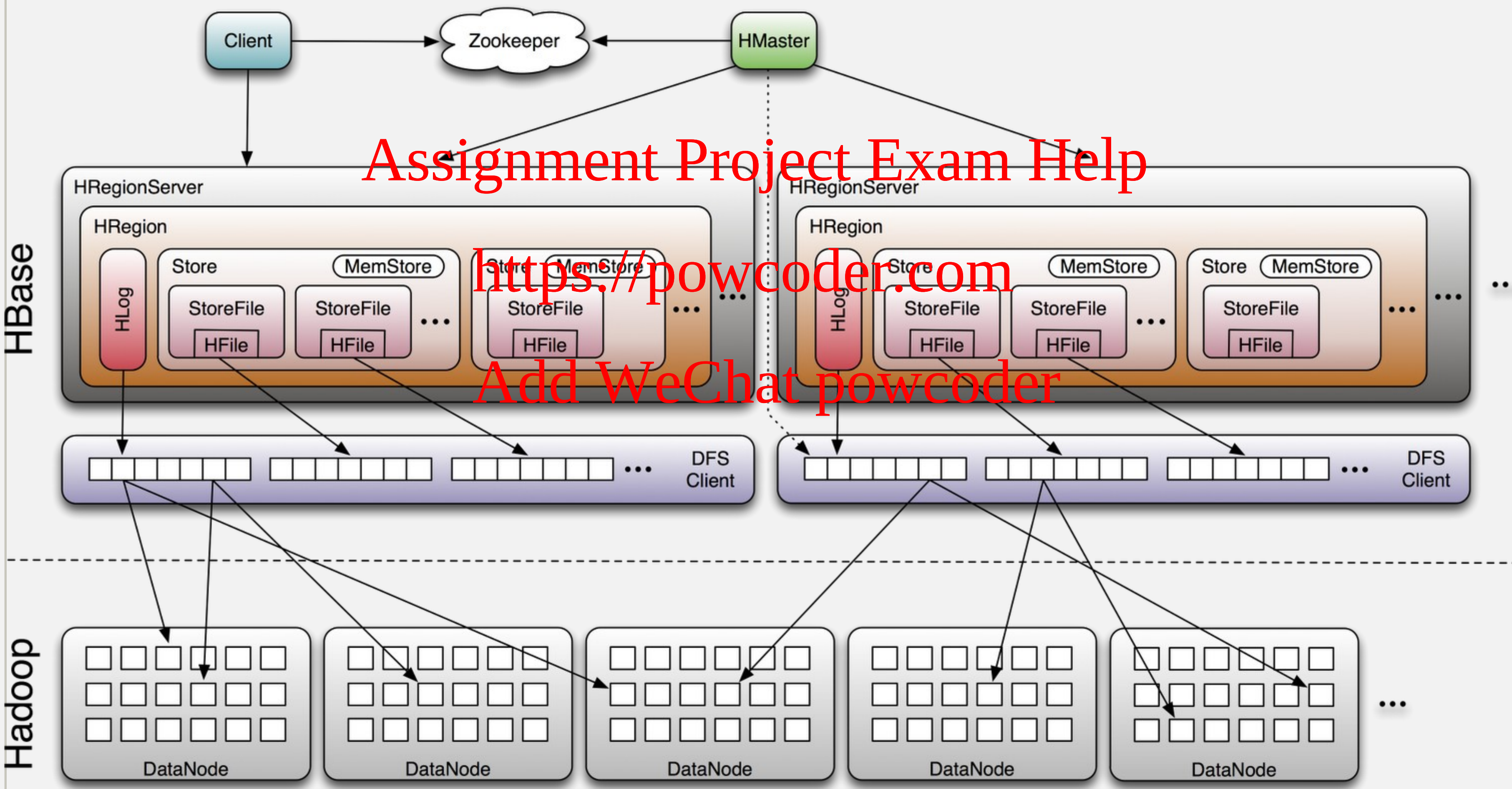
- The HBaseMaster
  - One master
- The HRegionServer
  - Many region servers
- The HBase client



HBase Components

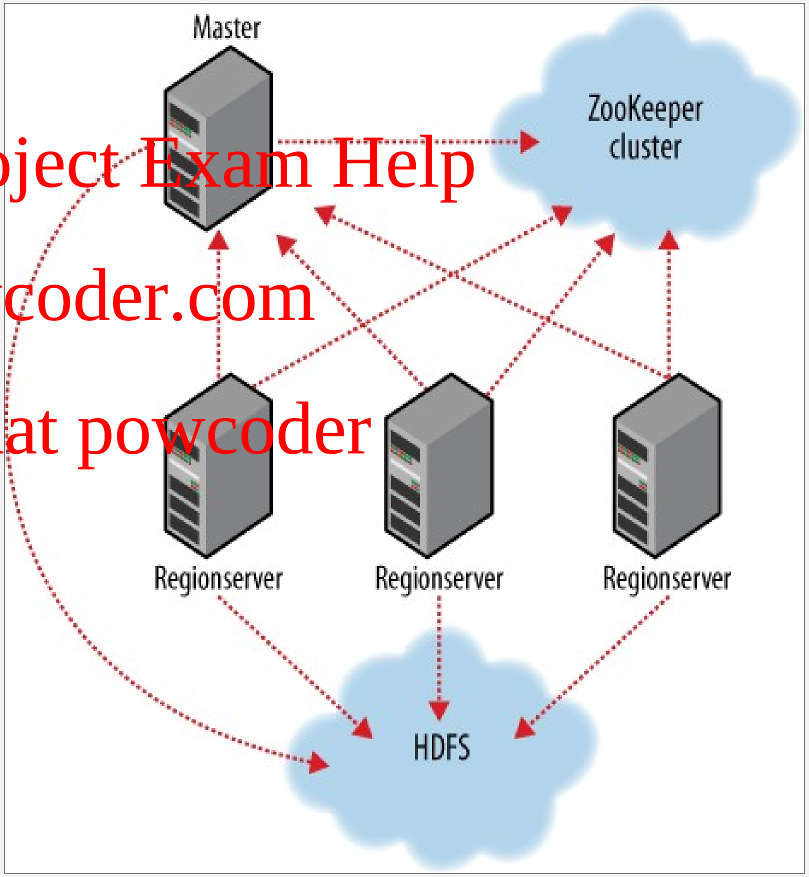
- **Region**
  - A subset of a table's rows, like horizontal range partitioning
  - Automatically done
- **RegionServer (many slaves)**
  - Manages data regions
  - Serves data for reads and writes (***using a log***)
- **Master**
  - Responsible for coordinating the slaves
  - Assigns regions, detects failures
  - Admin functions

Big Picture


Client
Zookeeper
HMaster
HBase
HRegionServer
HRegion
HLog
Store
MemStore
StoreFile
HFile
DFS Client
Hadoop
DataNode

ZooKeeper

- Manages synchronization, status, etc. of servers
- HBase depends on ZooKeeper
- By default HBase manages the ZooKeeper instance
  - E.g., starts and stops ZooKeeper
- HMaster and HRegionServers register themselves with ZooKeeper



Useful commands for shell

- Given schema:

  Row key personal data professional data

- The following commands are useful

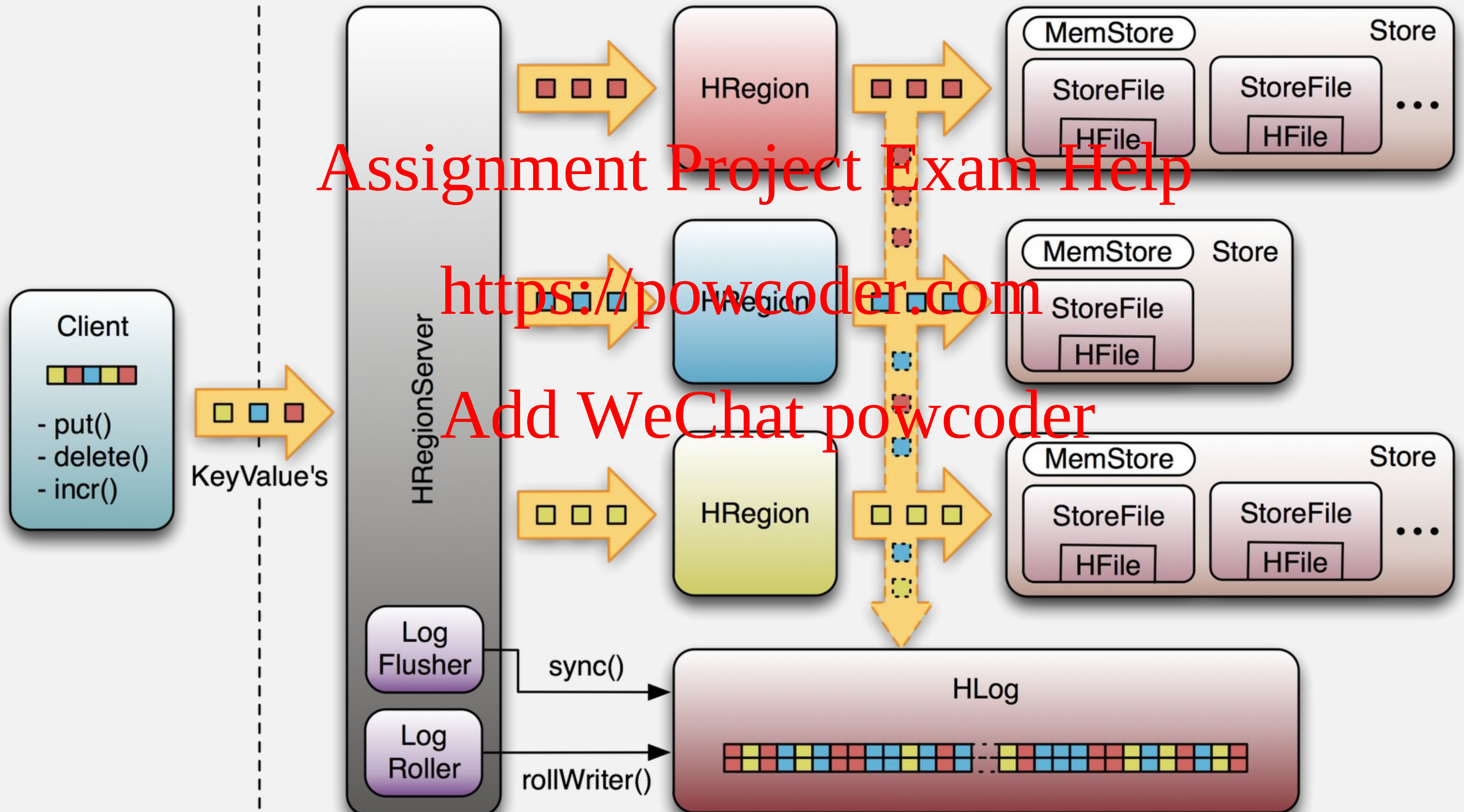

```
create 'emp', 'personaldata', 'professionaldata'
list
describe "emp"
put 'emp','1','personaldata:name','raju'
get 'emp','1'
scan 'emp',{COLUMNS => 'personal data:city', LIMIT=>10,
STARTROW => '3', ENDROW => '9.A'}
```

HBase: **Joins**

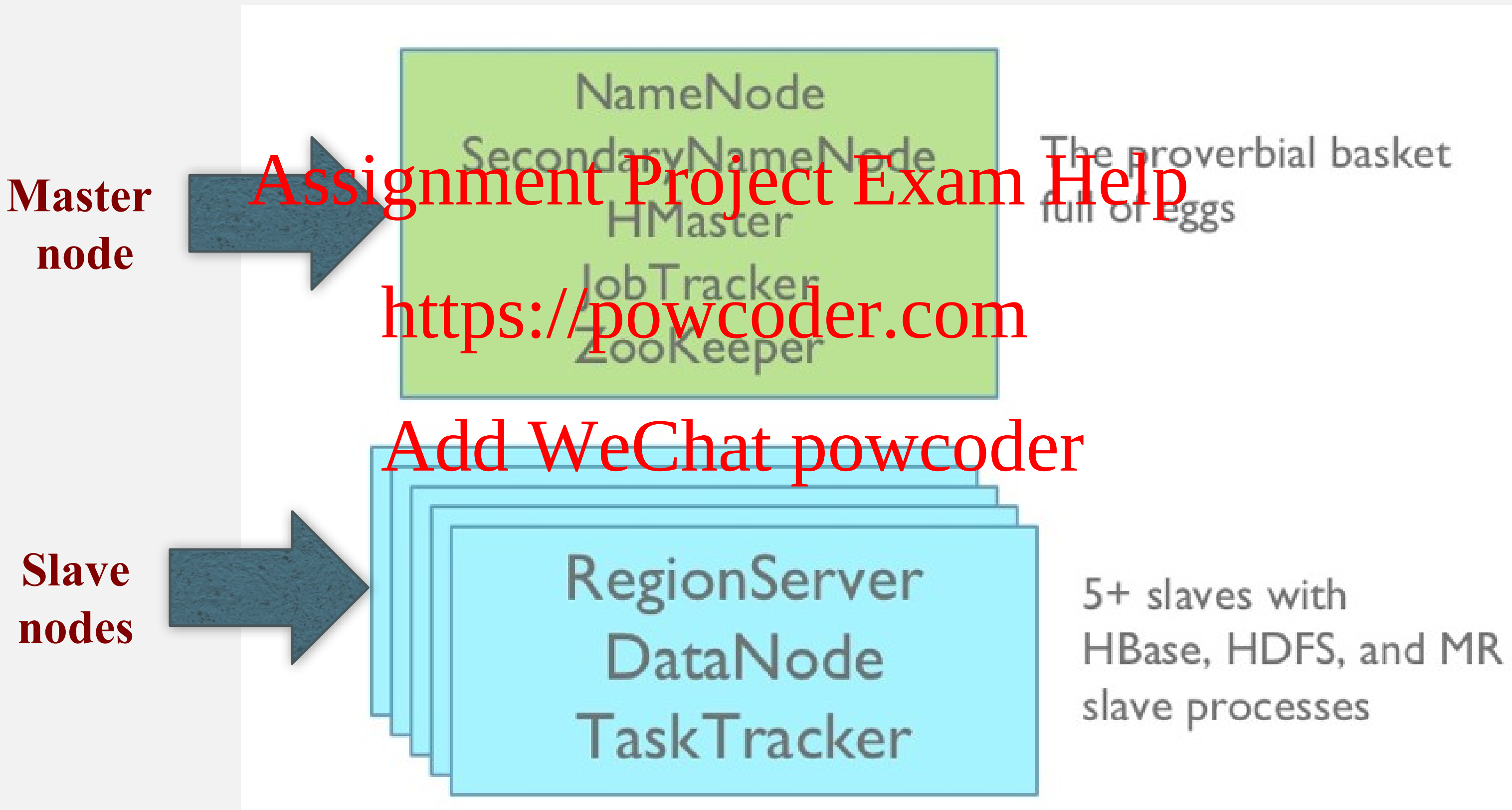
- HBase does not support joins

- Can be done in the application layer
  - Using scan() and get() operations

Logging Operations



HBase Deployment



HBase vs. HDFS

- Both are distributed systems that scale to hundreds or thousands of nodes

- ***HDFS*** is good for batch processing (scans over big files)
  - Not good for record lookup
  - Not good for incremental addition of small batches
  - Not good for updates

HBase vs. HDFS (Cont'd)

- ***HBase*** is designed to efficiently address the above points
  - Fast record lookup
  - Support for record-level insertion
  - Support for updates (not in place)

- HBase updates are done by creating new versions of values

HBase vs. HDFS (Cont'd)

	Plain HDFS/MR	HBase
Write pattern	Append-only	Random write, bulk incremental
Read pattern	Full table scan, partition table scan	Random read, small range scan, or table scan
Hive (SQL) performance	Very good	4-5x slower
Structured storage	Do-it-yourself / TSV / SequenceFile / Avro / ?	Sparse column-family data model
Max data size	30+ PB	~1PB

If application has neither random reads or writes ➔ Stick to HDFS

HBase vs. RDBMS

	RDBMS	HBase
Data layout	Row-oriented	Column-family-oriented
Transactions	Multi-row ACID	Single row only
Query language	SQL	get/put/scan/etc *
Security	Authentication/Authorization	Work in progress
Indexes	On arbitrary columns	Row-key only
Max data size	TBs	~1PB
Read/write throughput limits	1000s queries/second	Millions of queries/second

When to use HBase

- You need random write, random read, or both (*but not neither*)

- You need to do many thousands of operations per second on multiple TB of data

- Your access patterns are well-known and simple