

# THE DATA SCIENCE LAB

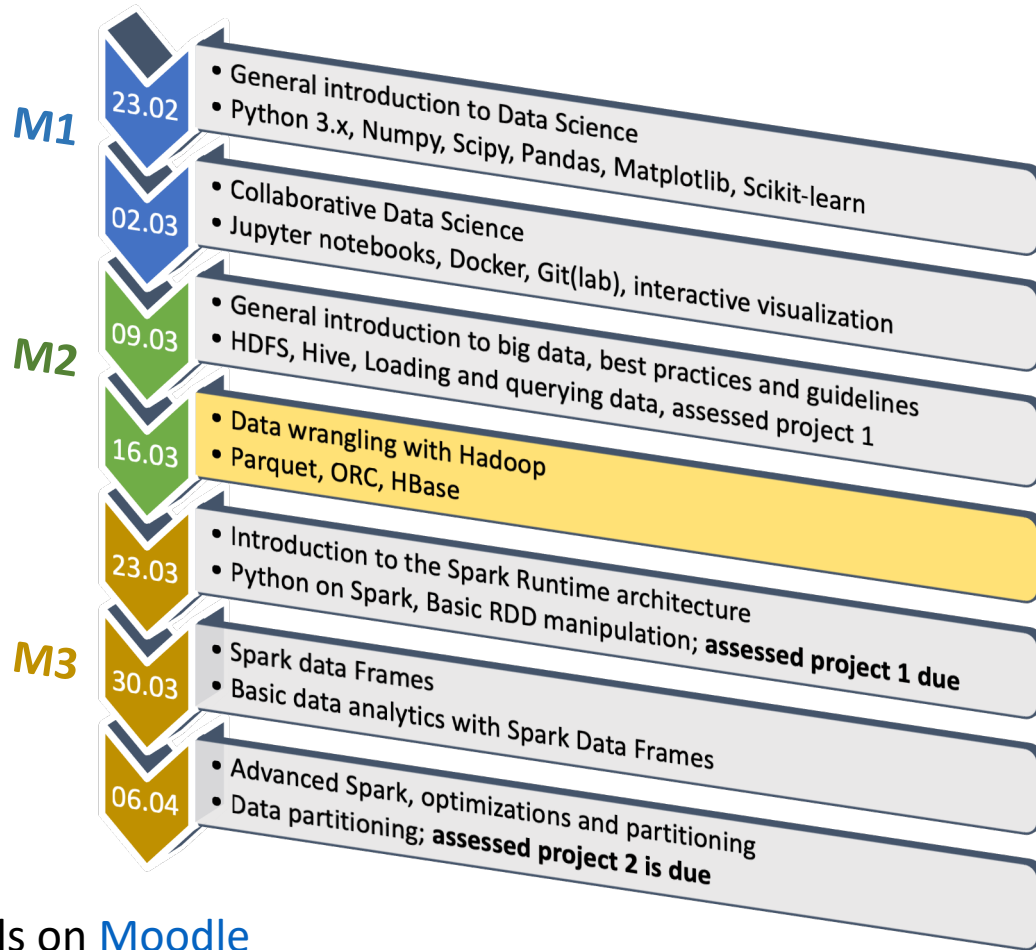
## Data Wrangling with Hadoop

COM 490 – Spring 2022

Week 4

**THIS CLASS WILL BE RECORDED**

# Agenda Spring 2022



\*Details on [Moodle](#)

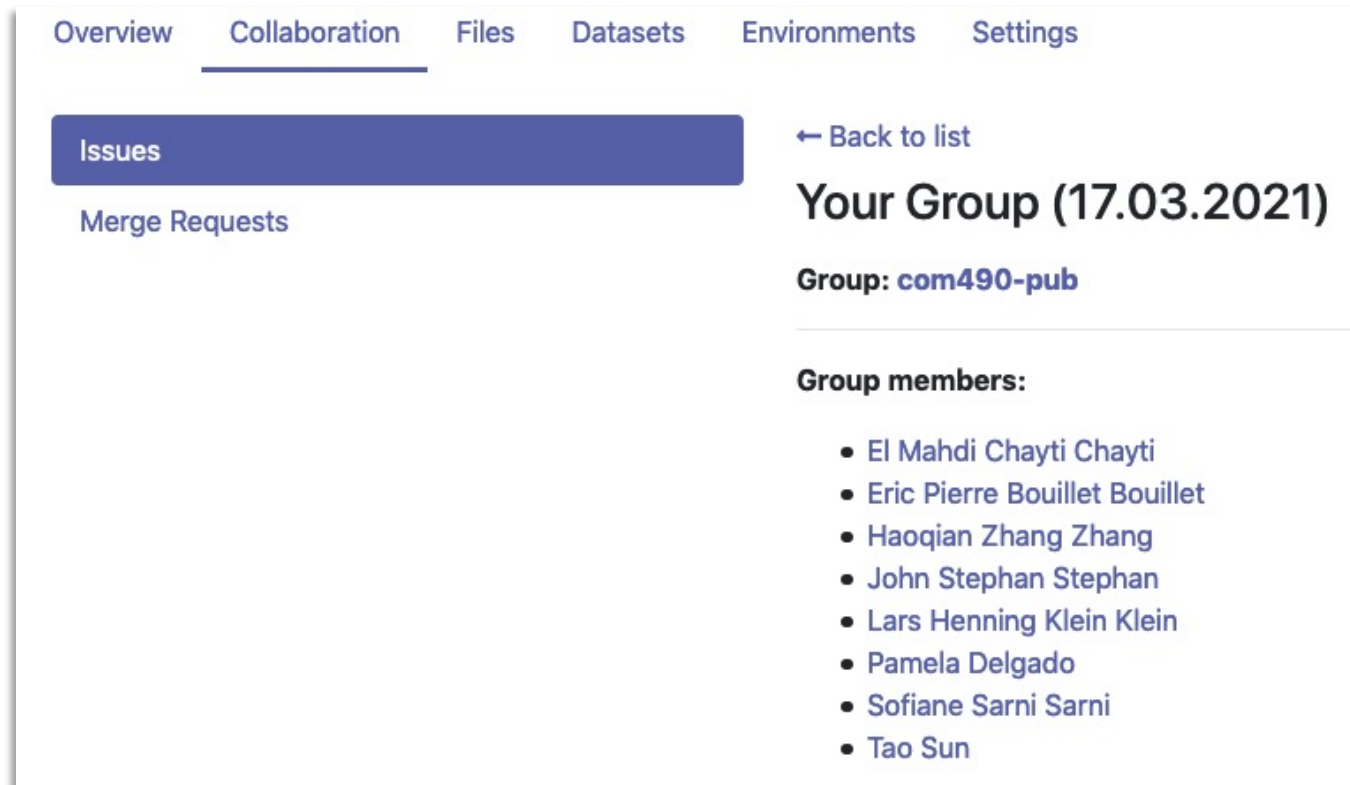
# Week 3 – Questions?



# Week 4 – Homework 1 : Checkpoint

Have you seen this in your homework project ?

**If not, contact us !**



The screenshot displays the DVC Collaboration web interface. At the top, there are navigation tabs: Overview, Collaboration (which is selected and underlined), Files, Datasets, Environments, and Settings. On the left side, under the Collaboration tab, there are two menu items: 'Issues' (highlighted with a blue background) and 'Merge Requests'. The main content area on the right shows a '← Back to list' link at the top. Below it, the title 'Your Group (17.03.2021)' is displayed in a large font. Underneath the title, it says 'Group: com490-pub'. A horizontal line separates this from the 'Group members:' section, which contains a bulleted list of eight names: El Mahdi Chayti Chayti, Eric Pierre Bouillet Bouillet, Haoqian Zhang Zhang, John Stephan Stephan, Lars Henning Klein Klein, Pamela Delgado, Sofiane Sarni Sarni, and Tao Sun.

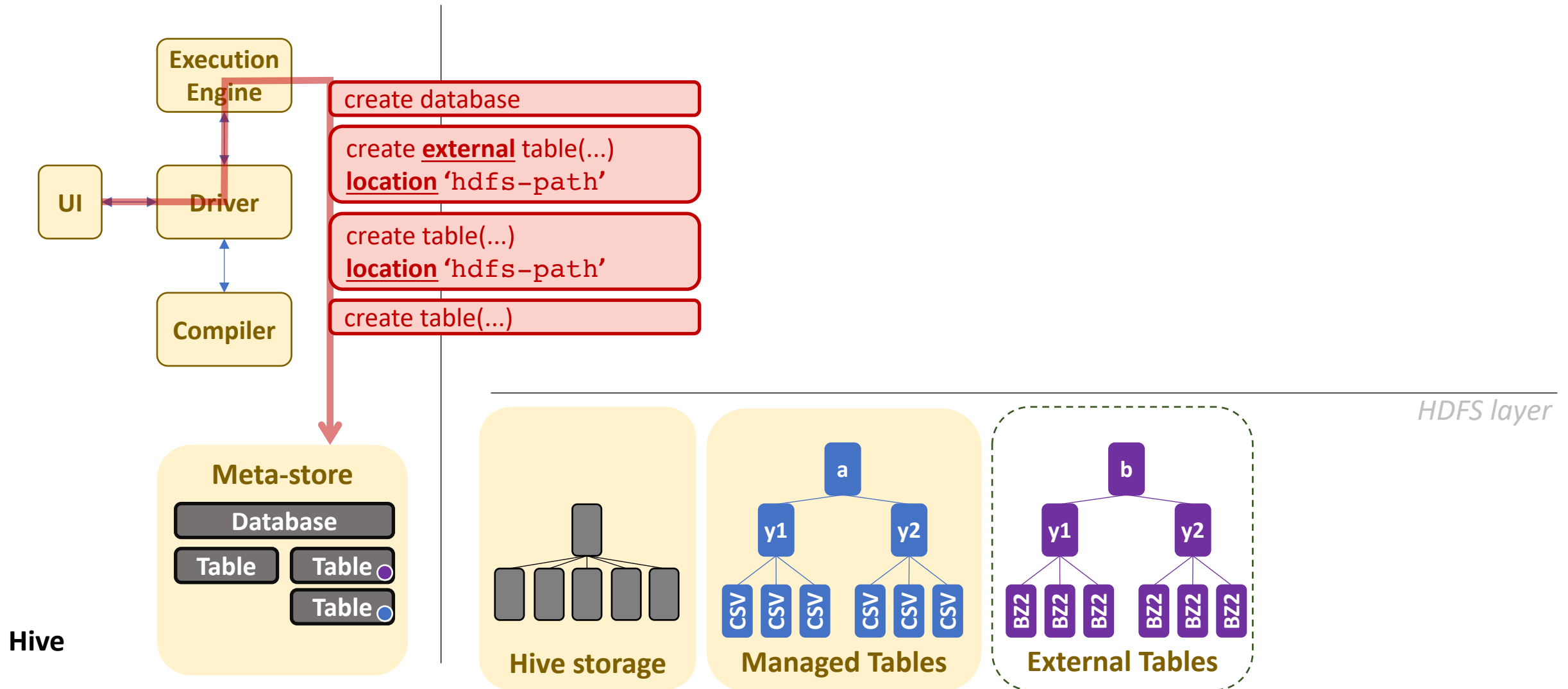
# Today's Agenda

- Data Wrangling with Hive
- HBase
- Exercises week 4
  - Data wangling with Hive, CSV and ORC
  - Hive on JSON with Twitter
  - Fist steps with HBase, Hive over HBase

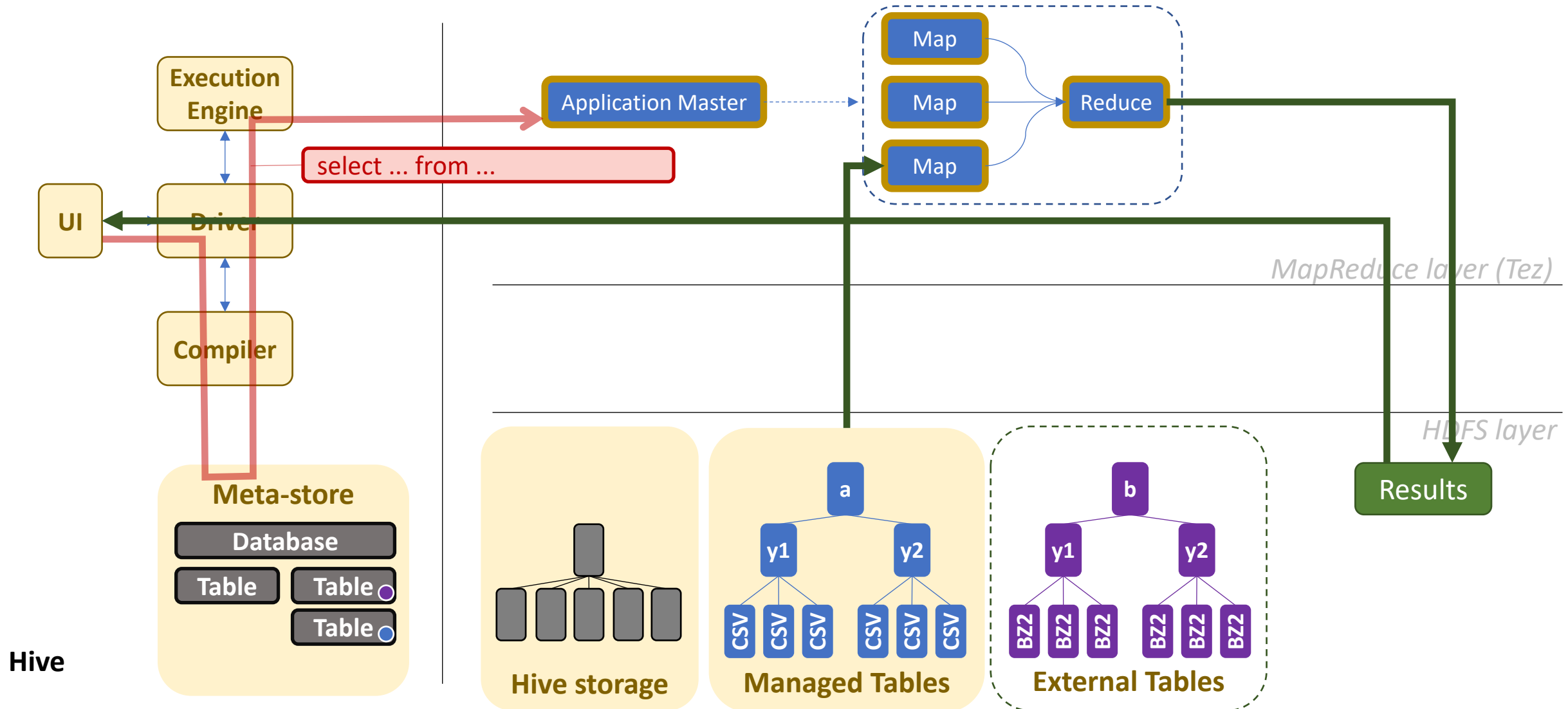
# Data Wrangling with Hive



# Hive Tables



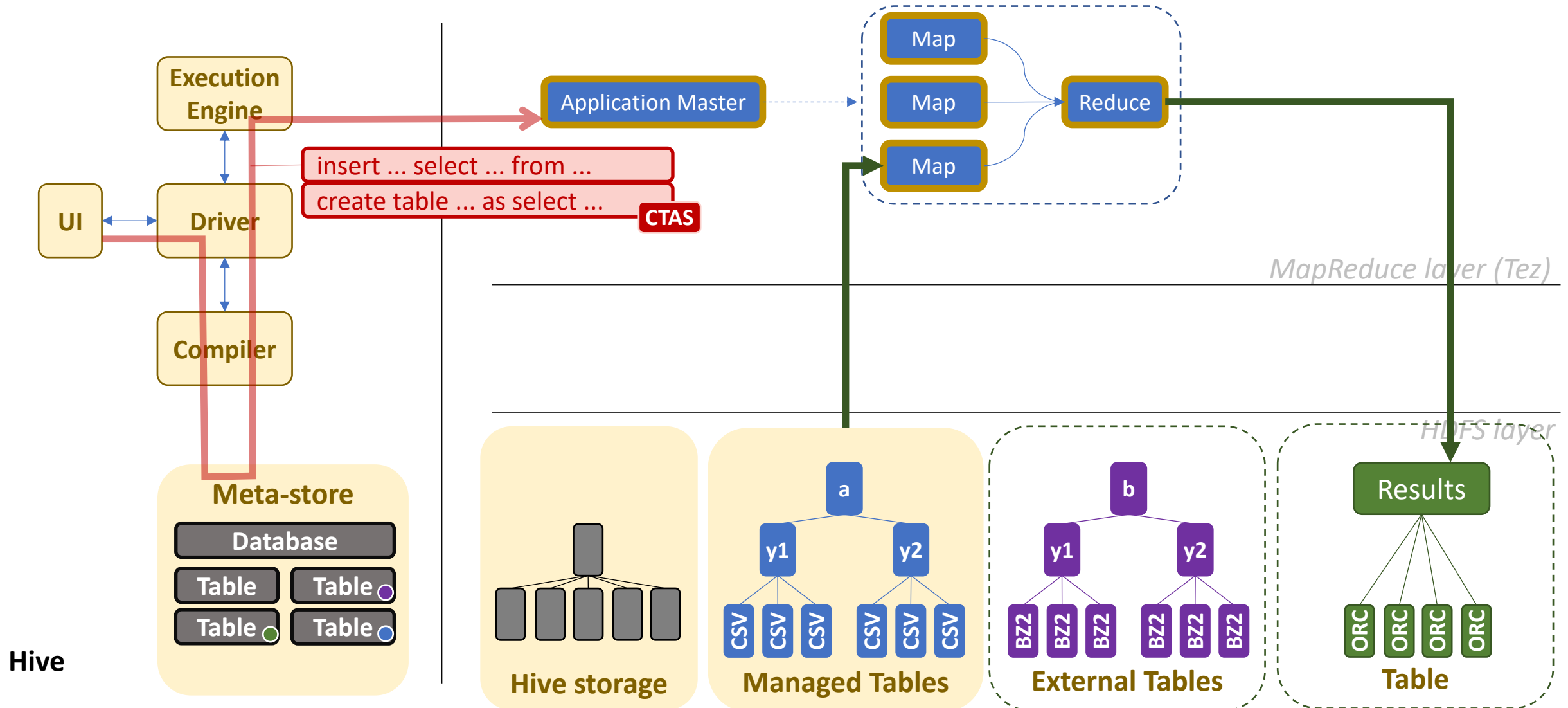
# Hive Tables



Hive



# Hive Tables



Hive

# Row versus Column Data Base

- Row-oriented databases
  - Designed to efficiently return data for an entire row
  - Best suited for Online Transaction Processing (OLTP), for retrieving information about a few objects:
    - E.g. return address or other attributes of a person (identified by a unique key)
- Column-oriented
  - Designed to efficiently process data analytics on entire columns
  - Best suited for Online Analytical Processing (OLAP)
    - E.g. compute statistics such as average or variance on values or group of values in a column

# Popular HDFS Storage Format

- Storage format - this is how data is physically stored in the HDFS blocks
  - **Plain text** (csv, json, xml, ...),
    - Row-oriented (most of the time)
    - This often the format you get from external sources
    - Best for OLTP
    - Batch and (often) stream processing
    - Splittable (if one line per record)
  - **Parquet**
    - Column-oriented, best for OLAP
    - Integrated compression: None, SNAPPY, ZLIB
    - Splittable
    - For write once, read many (WORM)
    - Batch processing only
  - **ORC**
    - Column-oriented, in collections of rows (stripes of 250MB), best for OLAP
    - Indexed
    - Splittable (per stripes).
    - integrated compression: None, SNAPPY, ZLIB
    - Optimized for WORM
    - Batch processing only
  - **Avro**
    - Row-oriented,
    - Splittable
    - Support block level compression
    - Best for OLTP
    - Support schema evolution



# HDFS vs Hive

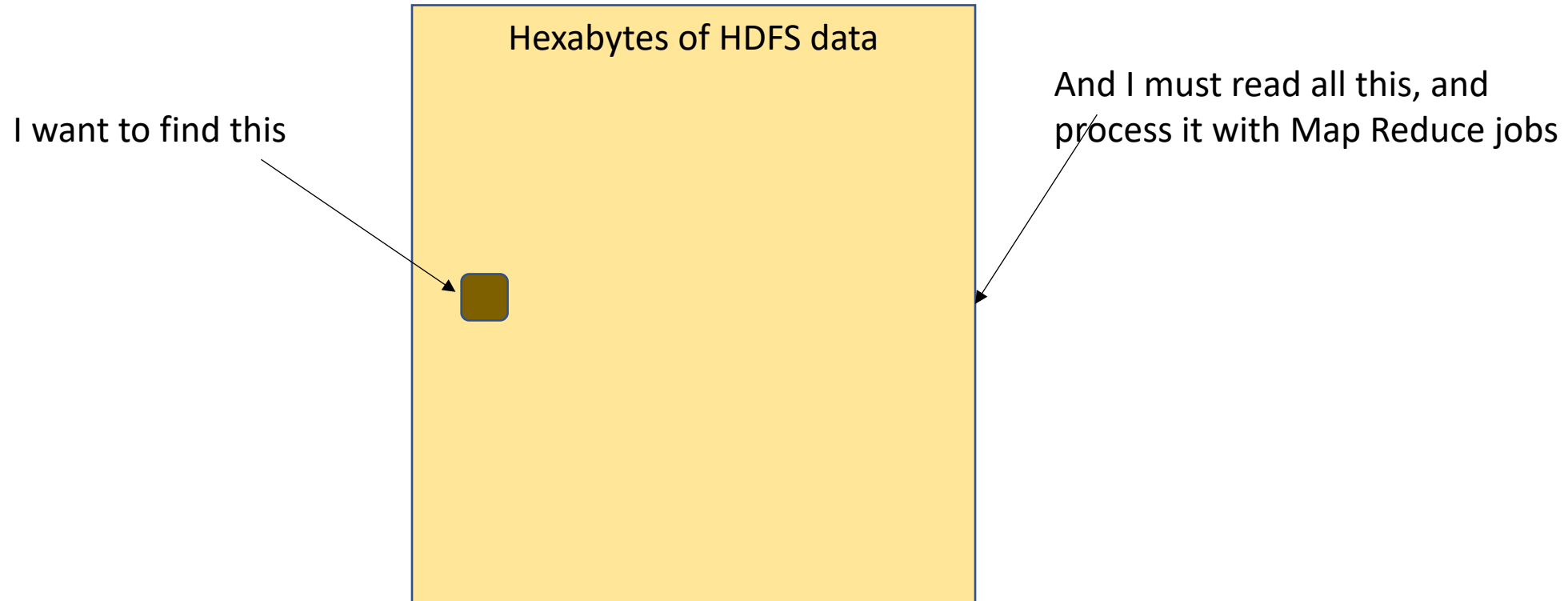
## HDFS

- Scalable, distributed, cost effective, reliable
- Store raw data in CSV, JSON, ORC, PARQUET, ...
- Not optimized for fine grained access
- Best for parking data and for batch access
- Not ideal for for complex queries (select, join, ...)

## Hive

- Big data warehouse
- Built on top of Hadoop (Tez)
- Store in HDFS, and SerDe-supporting storages
- SQL interface
- Best for relational queries in batch on huge data sets, e.g. big table joint etc.

# Between HDFS and Hive – the Gap



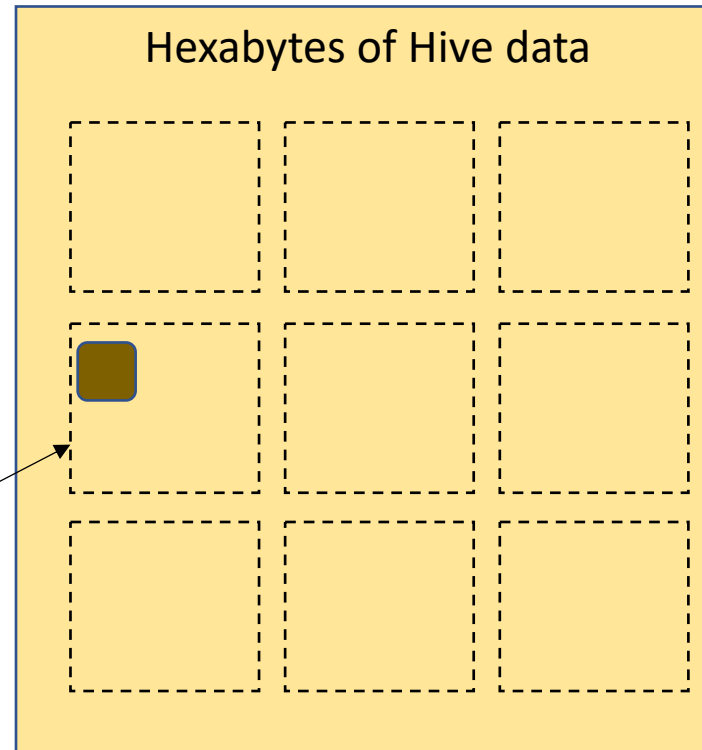
# Between HDFS and Hive – the Gap

Partitioning, and  
bucketization can help

....

Up to a point

I only need to search this area



Partitioning: distinct values (e.g. year) have each a distinct partition (file). Each partition correspond to one value.

Bucketing: values are hashed and randomly distributed in n-bucket (n given). A bucket may contain multiple values, but any given value always end up in the same bucket.



# No SQL databases

We can do better, but there is always a tradeoff

- Faster queries, but less expressivity in the query
  - E.g. allow one way to search for data using keys only (no-SQL)
- Achieve near-real time random Read and Write on arbitrarily large number of rows

# HBase

- Key value store
  - Data is indexed by keys, which points to collections of columns.
  - Columns are grouped in column families
  - Column families are declared when table is created
  - Columns (names and values) are conjured on the fly when rows are created or updated
- Wide column store
  - Hbase can handle billions of rows containing millions of columns
- Sparse
  - Empty columns do not take space in HBase, in fact they do not exist from HBase standpoint
- Versioned
  - Each column values can have a fixed number of versions, changing a value creates a new version
  - Accessing a value (a Cell in Hbase parlance) at a given version is done with the tuple:
    - namespace:table—name {row-key, column-family:column-name, version }

# HBase

- Anatomy of Hbase rows

<u>Row-Key</u>	<u>Column family cf1</u>	<u>Column family cf2</u>
0123456abcdef	col1={ 1, 2, 3 }, col2={"a", "b" }	col3={ "v1", "v2" }
0123456ghijklm	col1={ 4, 5, 6 }	col3={ "v3", "v4" }
....		

- Most common DDL operations

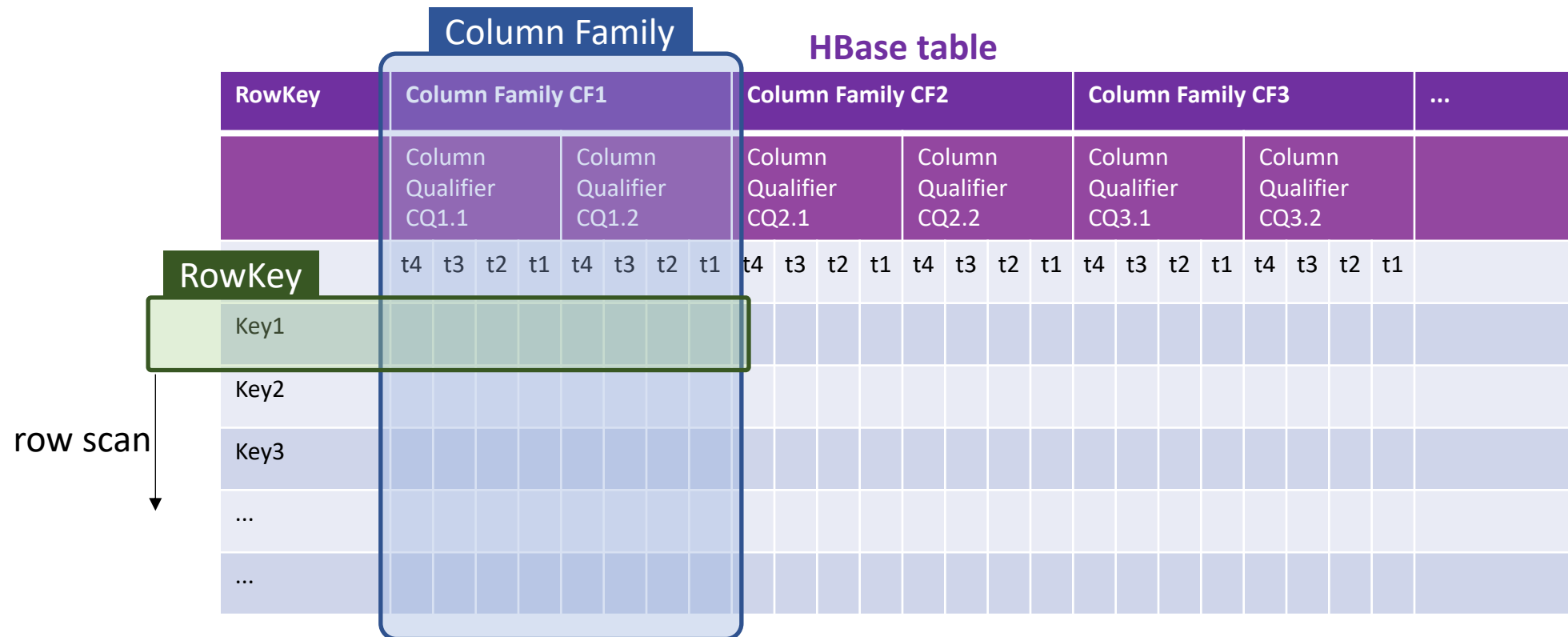
- create table 'namespace:tablename', { family1 properties }, { family2 properties }
- enable/disable table 'namespace:tablename'
- drop tables 'namespace:tablename'
- list 'namespace:.\*'

- Most common DML operations

- put 'namespace:table', 'row-key', 'cf1:col1', value, [ 'version-ts' ]
- get 'namespace:table', 'row-key', [ time range, column, versions ]
- scan 'namespace:table', [ row range, time range, filters, ...]

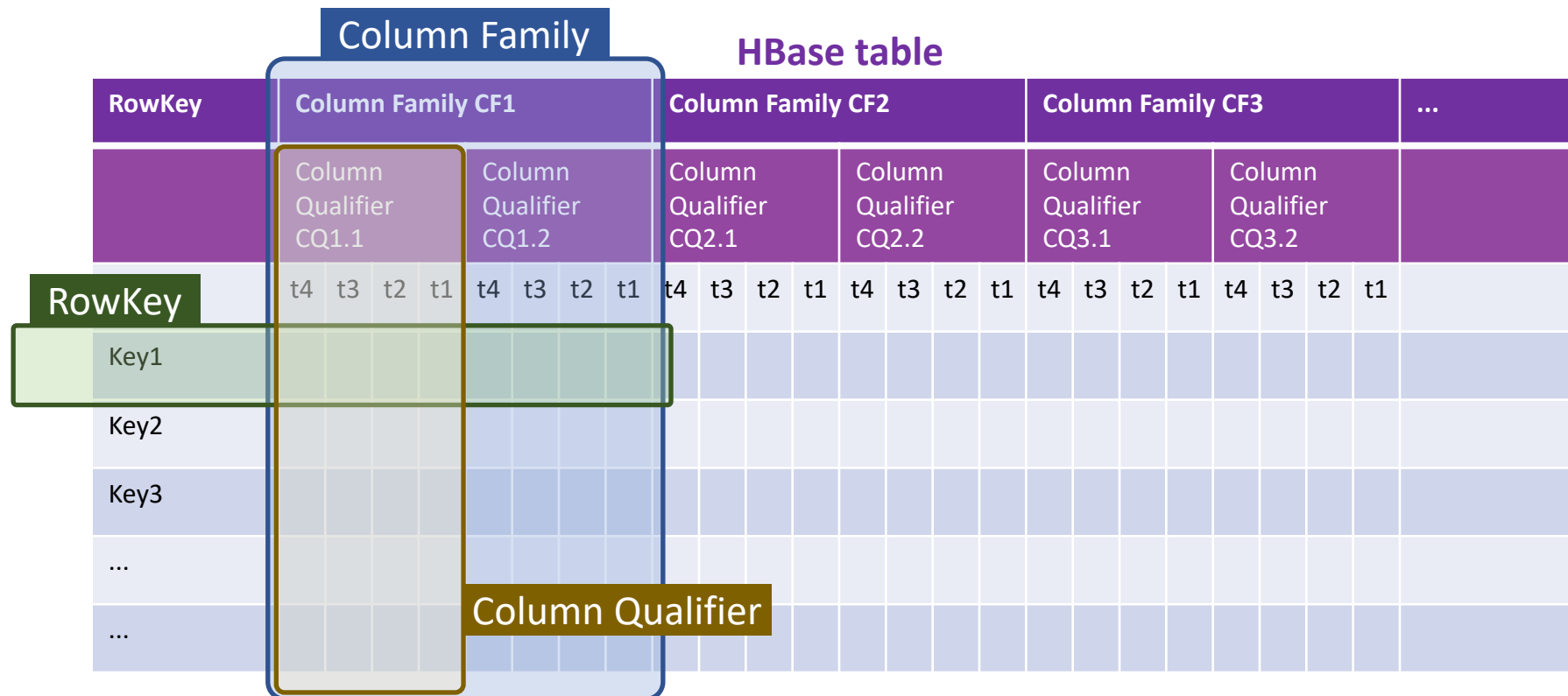


# HBase Column Oriented Data Model



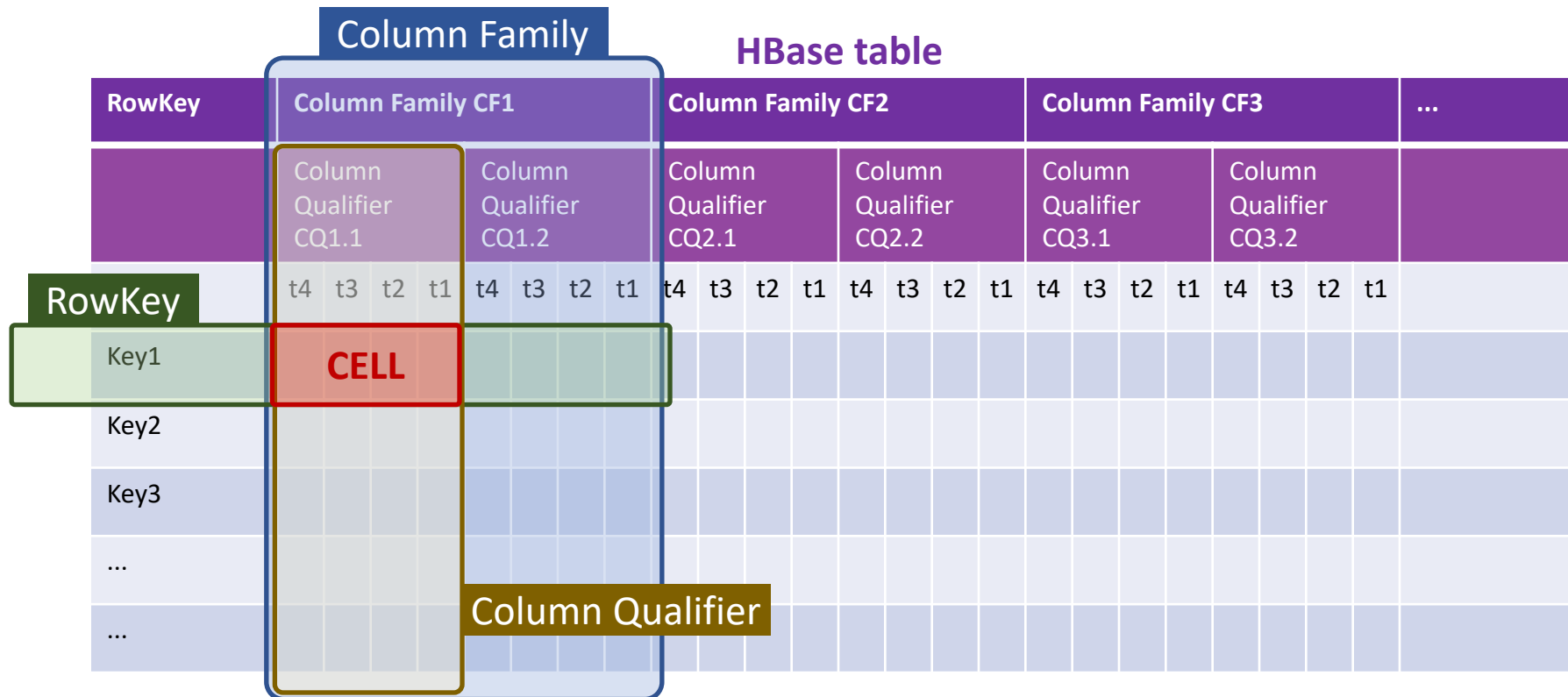
- Values of a same column family are stored together
- Values of a column family can be read in a single seek (best for OLAP)
- Column families are declared at table creation time

# HBase Column Oriented Data Model



- Column qualifier are conjured-up on the fly
- Technically they use space in a RowKey only if they are declared in that RowKey
- Everywhere else, they don't exist

# HBase Column Oriented Data Model



- **RowKey** + **Column Family**/**Column Qualifier** = **CELL**
- Values are stored in cells, with timestamps (date and time)
- It is possible to retrieve earlier versions (earlier timestamps) of a cell's value



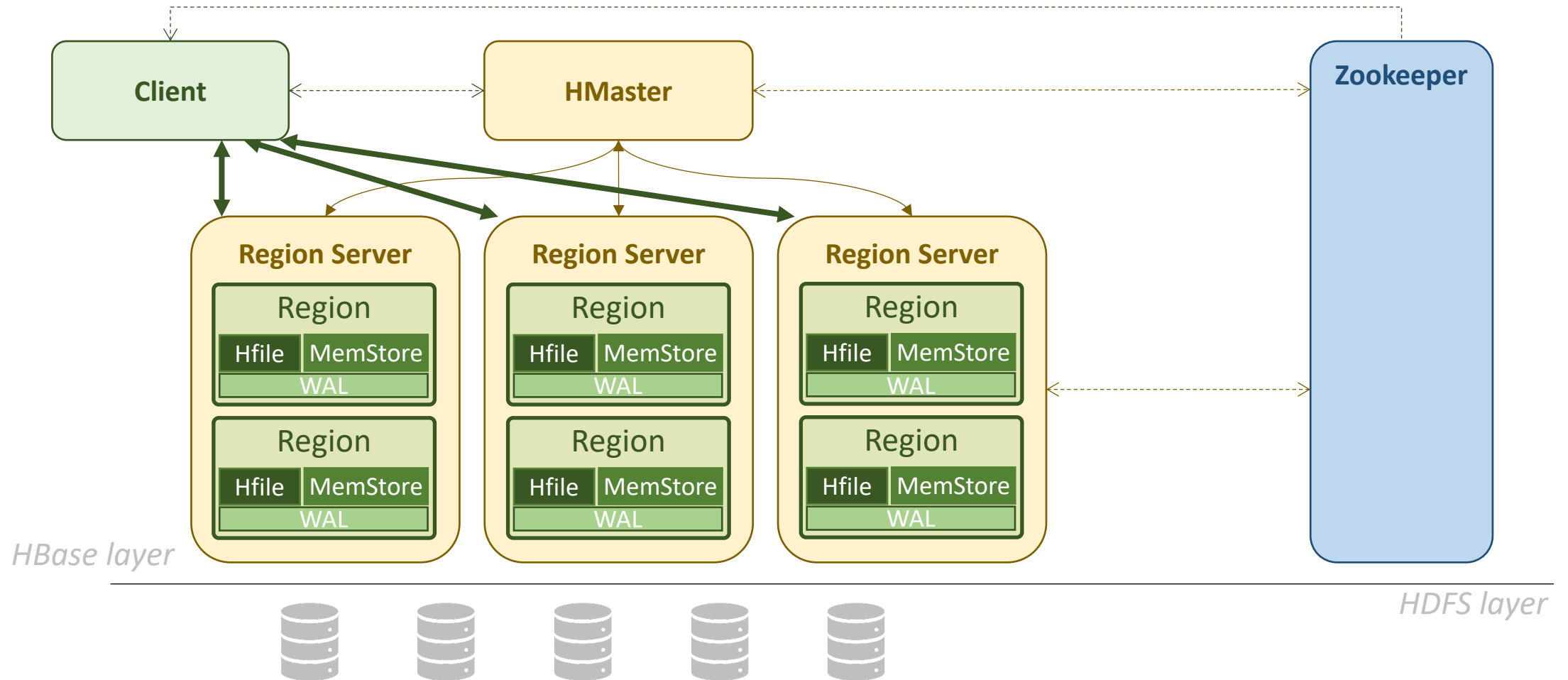
# HBase Architecture – Region decomposition

**HBase table**

RowKey	Column Family CF1								Column Family CF2								Column Family CF3								...
	Column Qualifier CQ1.1				Column Qualifier CQ1.2				Column Qualifier CQ2.1				Column Qualifier CQ2.2				Column Qualifier CQ3.1				Column Qualifier CQ3.2				
Region	t4	t3	t2	t1	t4	t3	t2	t1	t4	t3	t2	t1	t4	t3	t2	t1	t4	t3	t2	t1	t4	t3	t2	t1	
Key1																									
Key2																									
Key3																									
...																									
...																									

- HBase tables are divided into regions
- A region contains all the values of a column family within a sorted RowKey range
- By default regions are 250MB large

# HBase Architecture



# HBase vs Hive

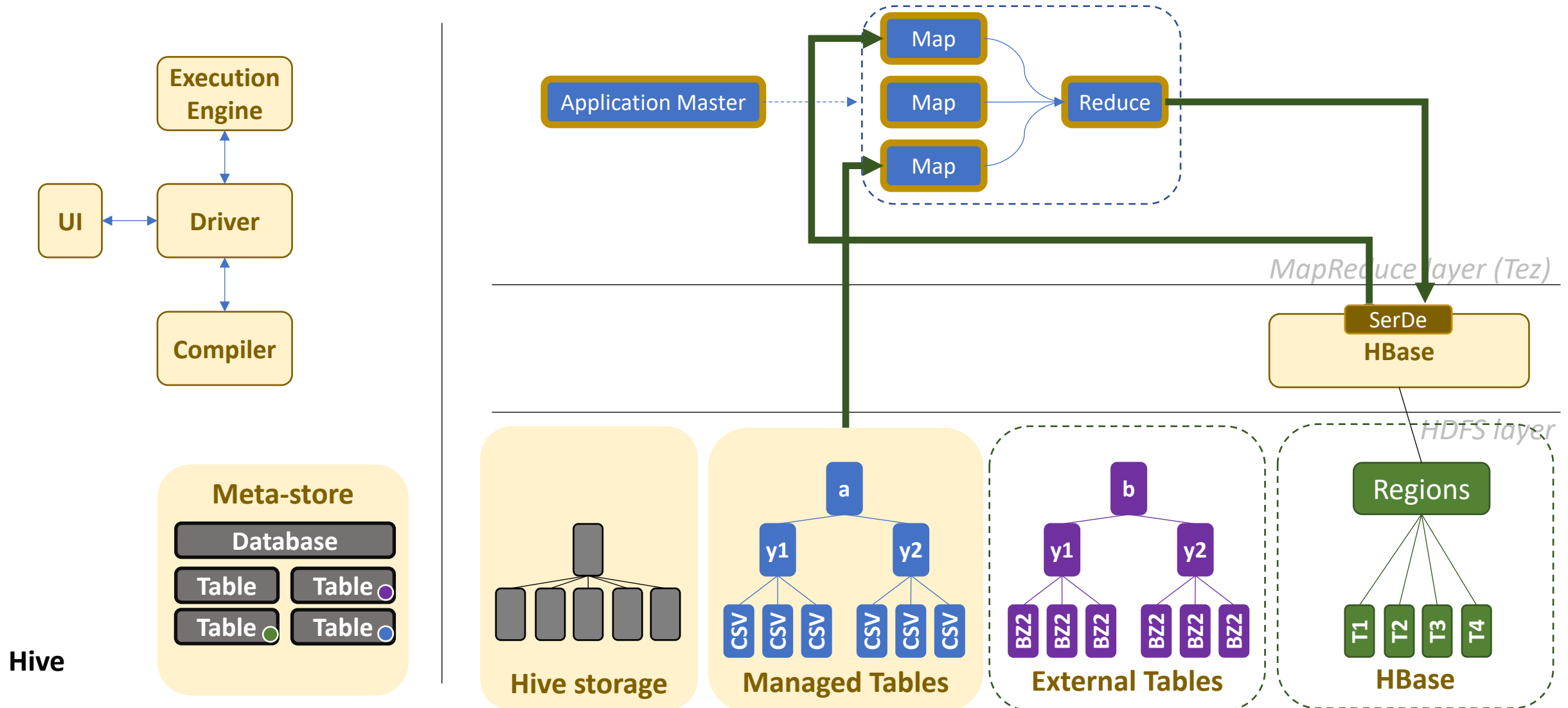
## Hive

- Big data warehouse
- Built on top of Hadoop (Tez)
- Store in HDFS, and SerDe-supporting storages
- Best for relational queries in batch on huge data sets, e.g. big table joint etc.

## HBase

- N(ot)o(nly) SQL
- Built on top of HDFS
- Optimize access to HDFS (partitioning, bucketing, tables compaction)
- Best for random access queries and real-time/fast data insetion
- Not for complex/RDBMS type of queries

# Hive on HBase



Hive



# In sum - HDFS vs Hive vs HBase

HDFS	Hive	HBase
Distributed file systems	Big data warehouse	Key-value store Wide column store
	Build on top of Hadoop	Build on top of HDFS
Store on commodity disk storage	Store in HDFS, Hbase and other storages	Store in HDFS storage
Best for parking data. Not for random access queries	Best for relational queries in batch on huge data sets Not for fast random access queries	Best for random access queries and real-time/fast data. Not for complex/relational queries

# Module 2 – Self-evaluation



## I know

- What is HDFS and how it works
- When to use HDFS
- What is Hive and how it works
- When to use Hive
- What is HBase and how it works
- When to use HBase
- When it is best to use Streams or Batch
- What is Map Reduce and best practices (optimizations)
- Most popular storage encodings: ORC, Parquet, ...

## I am able to

- Copy data to and from HDFS
- Explore HDFS arborescence
- Move data around in HDFS
- Modify ownership and HDFS access rights
- Create and manage Hive databases
- Create Hive external and managed Hive table
- Query Hive tables

# Start your engines

<https://dslab2022-renku.epfl.ch/projects/com490/lab-course>