THE DATA SCIENCE LAB
Data Wrangling with Hadoop

COM 490 – Module 2b

Week 4

Agenda 2025 - Module 2b

19.02	Introduction to Data Science with Python
26.02	(Bigger) Data Science with Python
05.03	Introduction to Big Data Technologies
12.03	Big Data Wrangling with Hadoop
19.03	Advanced Big Data Queries
26.03	Introduction to Spark
02.04	Spark Data Frames
09.04	Advanced Spark
16.04	Introduction to Stream Processing
30.04	Stream Processing with Kafka
07.05	Advanced Stream Processing
14.06	Final Project Q&A
22.05	Final Project Videos Due before midnight
28.05	Oral Sessions

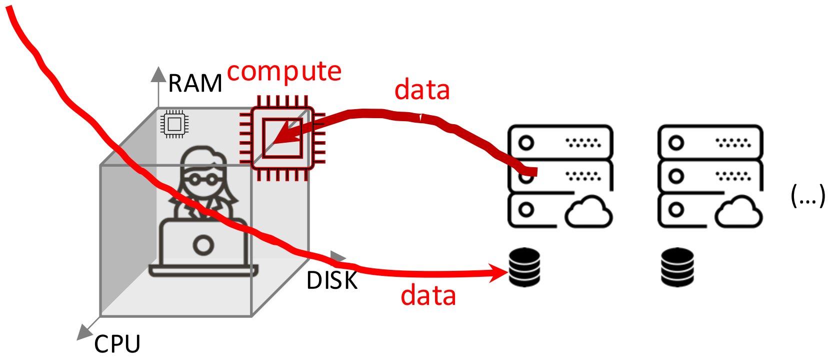
Module 2a – Questions?

Objective Module 2a

- **You have formed the groups**
  - Otherwise contact us
- **Understand fundamental concepts of the big data journey**
  - Distributed computing and challenges
  - **Scale-out** vs **Scale-up**, **Hive Partitioning**, Predicate Push down, HDFS, **Splittable data format**, **Map Reduce**, C.A.P theorem tradeoffs, out-of-core computing (and what to do when pandas runs out-of-memory)
  - Understand why HDFS and Map Reduce work well together
  - Get an understanding of the various Hadoop technologies and their applications - can group technologies into groups that offer similar features (storage, data warehouse, nosql, …)
- **You can navigate HDFS and manage data on HDFS (exercises)**

Module 2a - Review

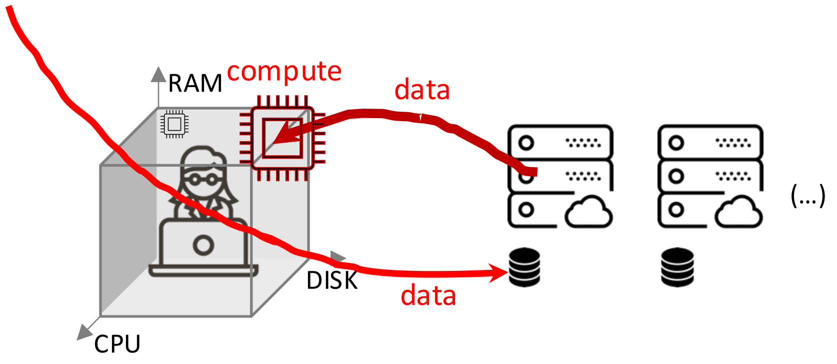
Module 2a - Lab



- **Transferred data to HDFS**: We uploaded data to the HDFS for storage and processing.
- **Executed data query tasks locally**: We performed data analysis using local tools (e.g., DuckDB) to query the data.
    - Data was copied from HDFS to the local machine to perform computations **out-of-core**, preventing potential **Out-of-Memory** errors by processing data in smaller chunks.

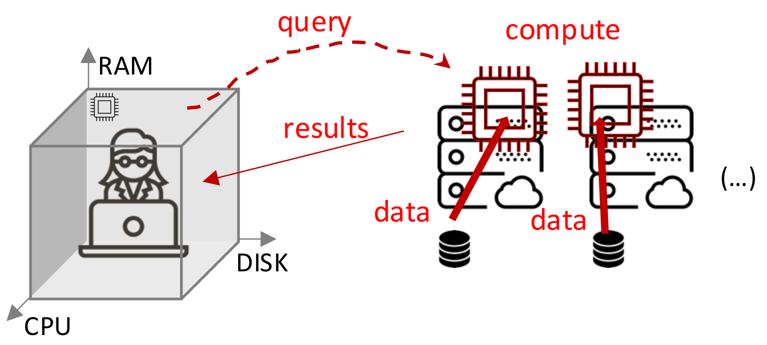
Module 2b

Module 2a - Lab



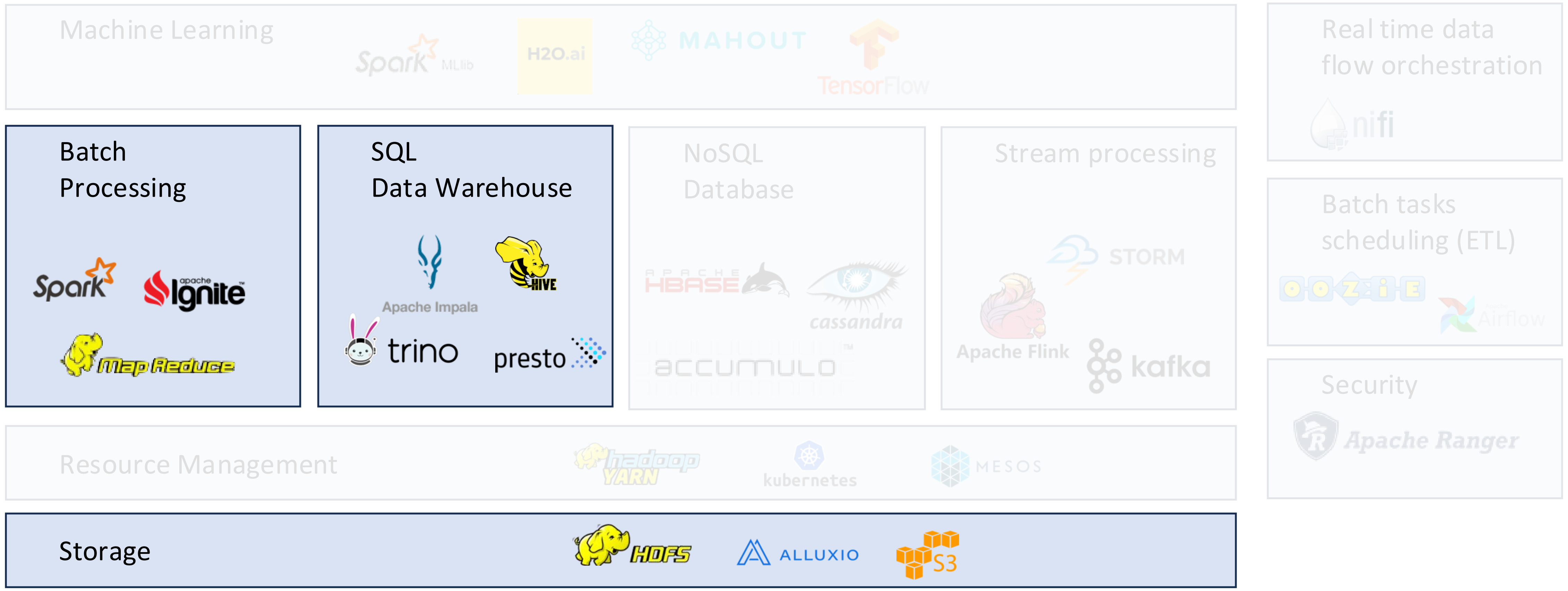
- **Transferred data to HDFS**: We uploaded data to the HDFS for storage and processing.
- **Executed data query tasks locally**: We performed data analysis using local tools (e.g., DuckDB) to query the data.
  - Data was copied from HDFS to the local machine to perform computations **out-of-core**, preventing potential **Out-of-Memory** errors by processing data in smaller chunks.

Module 2b - Lab



- **Run analytics tasks in parallel on the cluster**: We will leverage the distributed computing power of the cluster to execute analytics tasks concurrently (more RAM, more CPU)
- **Process data directly on HDFS**: Instead of moving data to local storage, we will perform analytics directly on the HDFS data

Addressing the Big Data Challenge – Data Warehouse(*)



(*) And data lakehouses

Today's Agenda

- Data Warehouses Concepts
  - Hive (lecture), Trino (lab)
- Exercises Module 2b
  - Data wangling with Trino, CSV and PARQUET

Data Warehouse

What is a Data Warehouse?

- Designed to provide an interface for **querying large amounts of structured data**
  - Data is typically stored in **external storage systems** such as **HDFS**, S3, or other distributed file systems
- Optimized for:
  - **Analytical processing** (OLAP) rather than transactional (OLTP)
  - **Complex queries** and **historical data analysis**
- High level query language, e.g. HiveQL (similar to SQL)

Data Definition Language (DDL) – E.g. HiveQL

```
CREATE DATABASE IF NOT EXISTS  mydatabase;
```

Create a database – this is like a namespace.

```
USE mydatabase;
```

Make *mydatabase* the **default** (not required but useful)

```
CREATE EXTERNAL TABLE mydatabase.mytable (
            betriebstag          STRING,
            fahrt_bezeichner     STRING,
            betreiber_id         STRING
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ";"
STORED AS TEXTFILE
LOCATION '/data/sbb/csv/istdaten';
```

Create table *mytable* in the database *mydatabase*

The **schema**: comma separated list of column names and their types

Storage **format**

Optional Location of data, on HDFS in example

https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL

Schema On Read

- ***Schema on READ***
  - You write the data on store (e.g. HDFS, or S3) first, and then you apply a schema later, when the data is being read off of the store; the same data at the given physical location can be accessed using different schemas
  - **Pros**:
    - No need to plan ahead, save the data and decide of the schema later
    - Support for **Schema Evolution**, (e.g. PARQUET, Avro, …): as you add more data to the store, new data do not need to be exactly the same schema as the old data
  - **Cons**: limited type safety, analytics must be ready to handle bad data; limited opportunities for "schema-aware" optimization

- **Versus. Traditional RDBMS** - *Schema on Write*
  - You must decide what is the schema of the database table first, then you can write data to the tables. The schema cannot change after you created the table.
  - **Pros**: better type safety; optimization possible if schema is known beforehand
  - **Cons**: you must plan ahead; it is harder to have different views of same data

Data Definition Language (DDL)

```
CREATE EXTERNAL TABLE mydatabase.mytable (
            [...]
)
...
LOCATION '/data/csv/sbb/istdaten';
```

Hive

```
CREATE TABLE catalog.schema.mytable (
            [...]
)
...
WITH(
            external_location '/data/csv/sbb/istdaten'
);
```

Trino

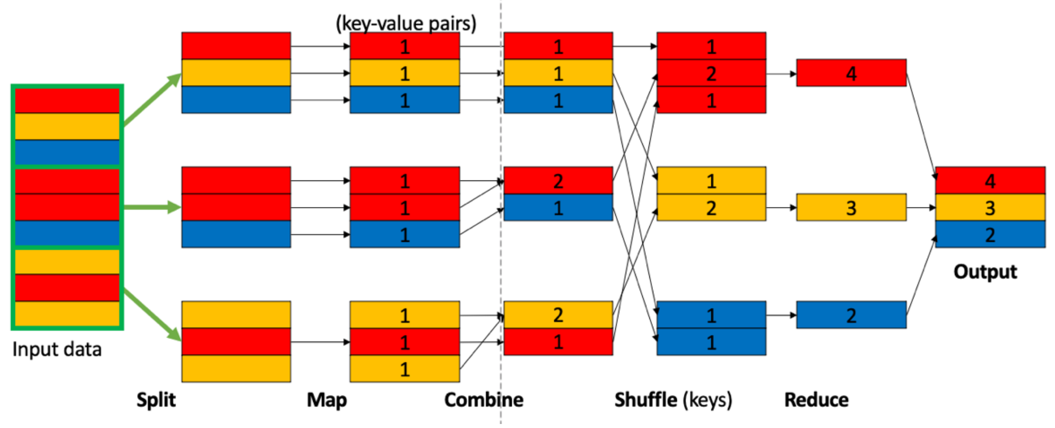
- If EXTERNAL
  - You manage the data
  - The data is not deleted when you DROP the table (only the meta-data of the table definitions)
- If not EXTERNAL
  - Data is managed by Hive
  - The data is deleted when you DROP the table !!!!
- In Hive EXTERNAL is NOT the default
- On pre-existing data, make it EXTERNAL

Data Retrieval Queries – E.g. HiveQL

```
SELECT product_id, COUNT(*) FROM mydatabase.mytable
        GROUP BY product_id;
```

SQL like select query

is automatically converted and executed as a map reduce application

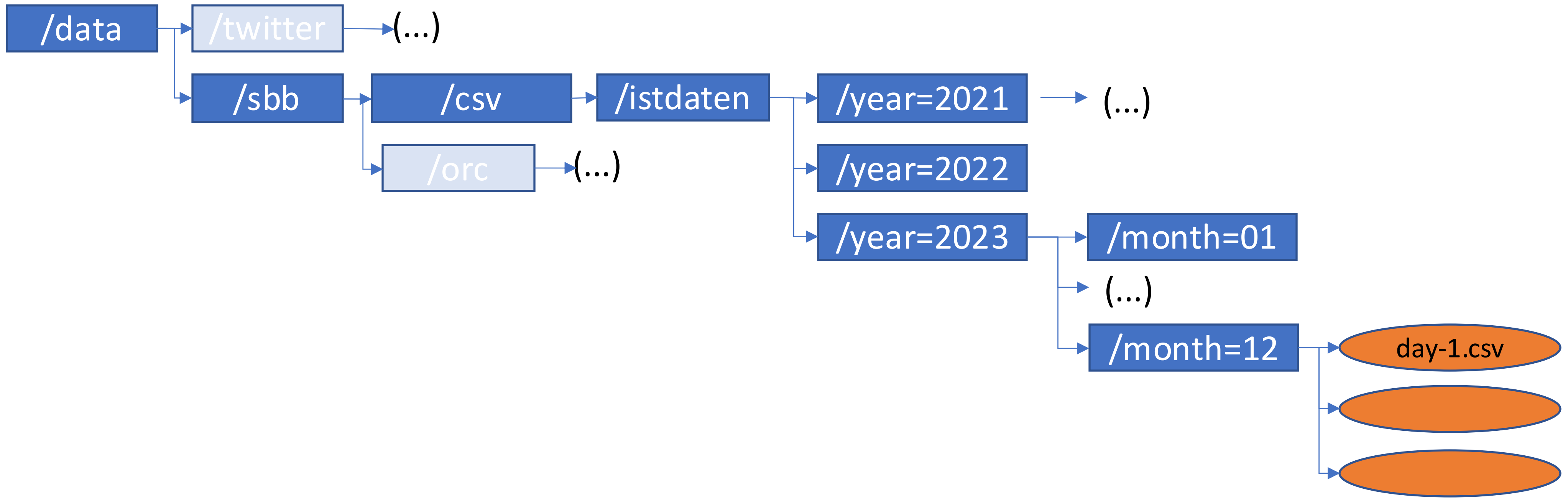


https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Select

Data Storage and Partitioning

- Hive Partitioning

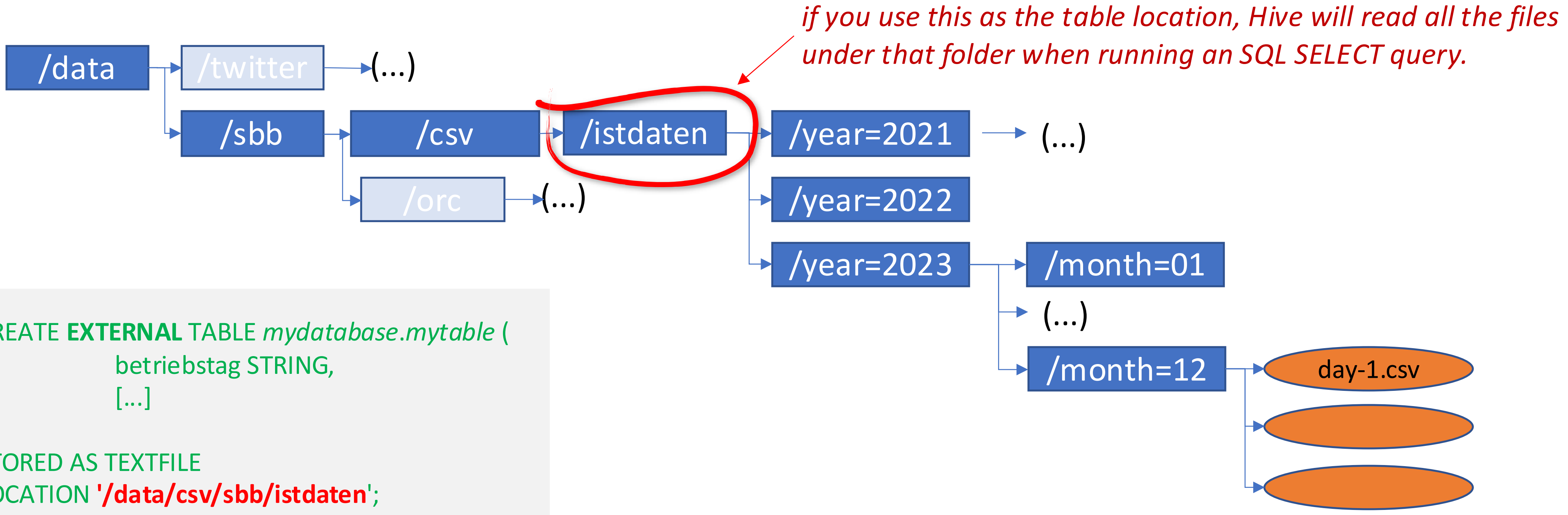
Example, file structure found on HDFS



Data Storage and Partitioning

- Hive Partitioning

Example, file structure found on HDFS



if you use this as the table location, Hive will read all the files under that folder when running an SQL SELECT query.
/data
/twitter
(...)
/sbb
/csv
/istdaten
/year=2021
(...)
/orc
(...)
/year=2022
/year=2023
/month=01
(...)
/month=12
day-1.csv

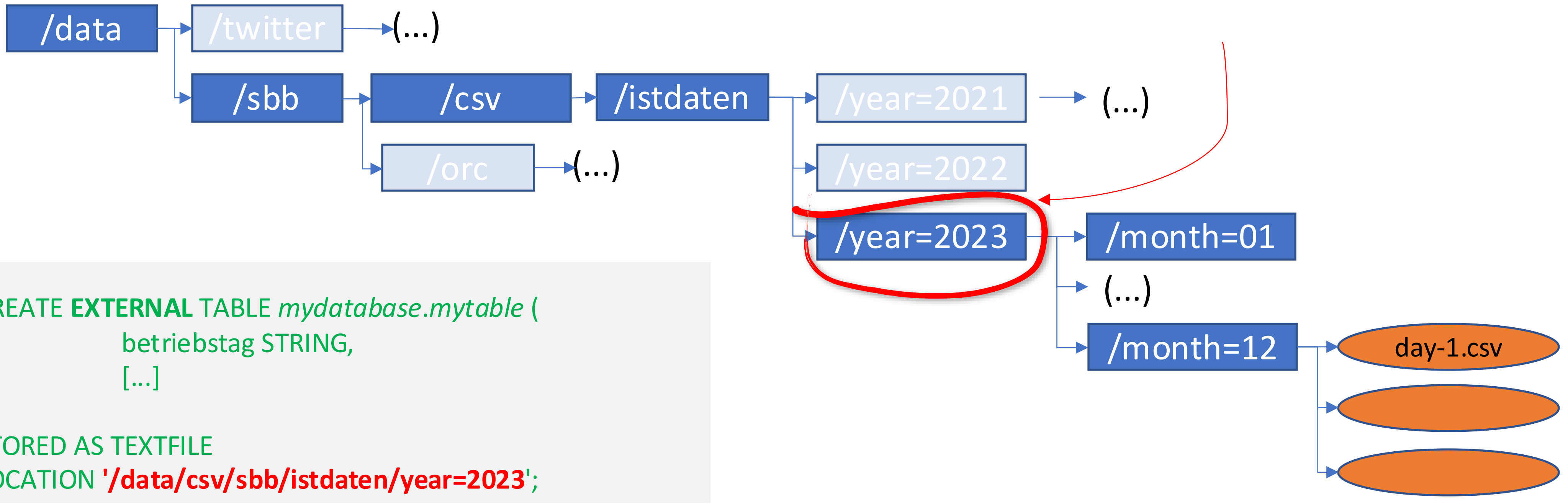
```
CREATE EXTERNAL TABLE mydatabase.mytable (
            betriebstag STRING,
            [...]
)
STORED AS TEXTFILE
LOCATION '/data/csv/sbb/istdaten';
```

Data Storage and Partitioning

- Hive Partitioning

Example, file structure found on HDFS

If you create the tables using a location in a sub-folder, Hive will only look under that folder …



```
CREATE EXTERNAL TABLE mydatabase.mytable (
        betriebstag STRING,
        [...]
)
STORED AS TEXTFILE
LOCATION '/data/csv/sbb/istdaten/year=2023';
```

Data Storage and Partitioning

- Hive Partitioning

```
CREATE EXTERNAL TABLE mydatabase.mytable (
        betriebstag STRING,
        [...]
)
PARTITIONED BY (year INT, month INT)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ";"
STORED AS TEXTFILE
LOCATION '/data/csv/sbb/istdaten';
```

You are giving a hint that the folder structure under **istdaten** is organized into year folders containing month folders. They CANNOT be the same column names as in the schema

```
MSCK REPAIR TABLE myatabase.mytable  [ADD PARTITIONS];
```

Here Hive will use the above partition hint, and will look for folders year=.../month=... and will create columns corresponding to those folders.

```
SELECT COUNT(*) FROM mydatabase.mytable WHERE year=2022
AND month=12 AND betriebstag='01.12.2022';
```

Partitioning columns are used in Hive SQL queries like any other columns. Hive will optimize the query to read only data found in the corresponding folders.

ETL - Extract Transform Load

Data Warehouses not only handle OLAP queries but also serve as powerful tools for transforming raw data through the ETL process:

- **Extract**:
  - Pull data from various sources (databases, HDFS, S3, APIs, etc.)
  - Sources can be structured, semi-structured, or unstructured
- **Transform**: (SELECT …)
  - Clean and process the raw data (e.g., standardization, filtering, handling missing values).
  - Apply business rules and aggregate data for analytics
- **Load**:
  - Store the transformed data into tables within the data warehouse, e.g. on HDFS, S3 – (CREATE AS SELECT, INSERT).
  - Data is now ready for analysis and querying.

Create Table as Select (CTAS) – E.g. HiveQL

Example of CTAS:

```
CREATE TABLE mydatabase.mynewtable
STORED AS PARQUET
LOCATION '/data/parquet/sbb/istdaten'
            AS SELECT * FROM mydatabase.mytable;
```

Create Table As Select (CTAS), note that in the new table:

- We do not need to specify the schema of the new table (it is derived from the source table
- We can specify a different storage format.
- You can specify a partitioning (since 3.2.0)

This is convenient if you want for instance to create a ORC or PARQUET format from a data in CSV format.

https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-CreateTableAsSelect(CTAS)

INSERT/LOAD – E.g. HiveQL

Example of CREATE + INSERT:

```
CREATE EXTERNAL TABLE mydatabase.mynewtable (
            betriebstag STRING,
            ...
)
PARTITIONED BY (year INT, month INT)
STORED AS PARQUET
LOCATION '/data/com-490/parquet/sbb/istdaten';


INSERT INTO TABLE mydatabase.mynewtable
SELECT * FROM mydatabase.mytable;
```

Create table and insert from another table. This option provides more flexibility (partitioning, clustering, external tables etc) than CTAS.

https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DML

Storage Formats

```
CREATE EXTERNAL TABLE mydatabase.mynewtable (
            betriebstag STRING,
            ...
)
PARTITIONED BY (year INT, month INT)
STORED AS PARQUET
LOCATION '/data/com-490/parquet/sbb/istdaten';
```

Storage formats: ORC, PARQUET, TEXTFILE (with additional field/record separator, header parameters, etc)

https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DML

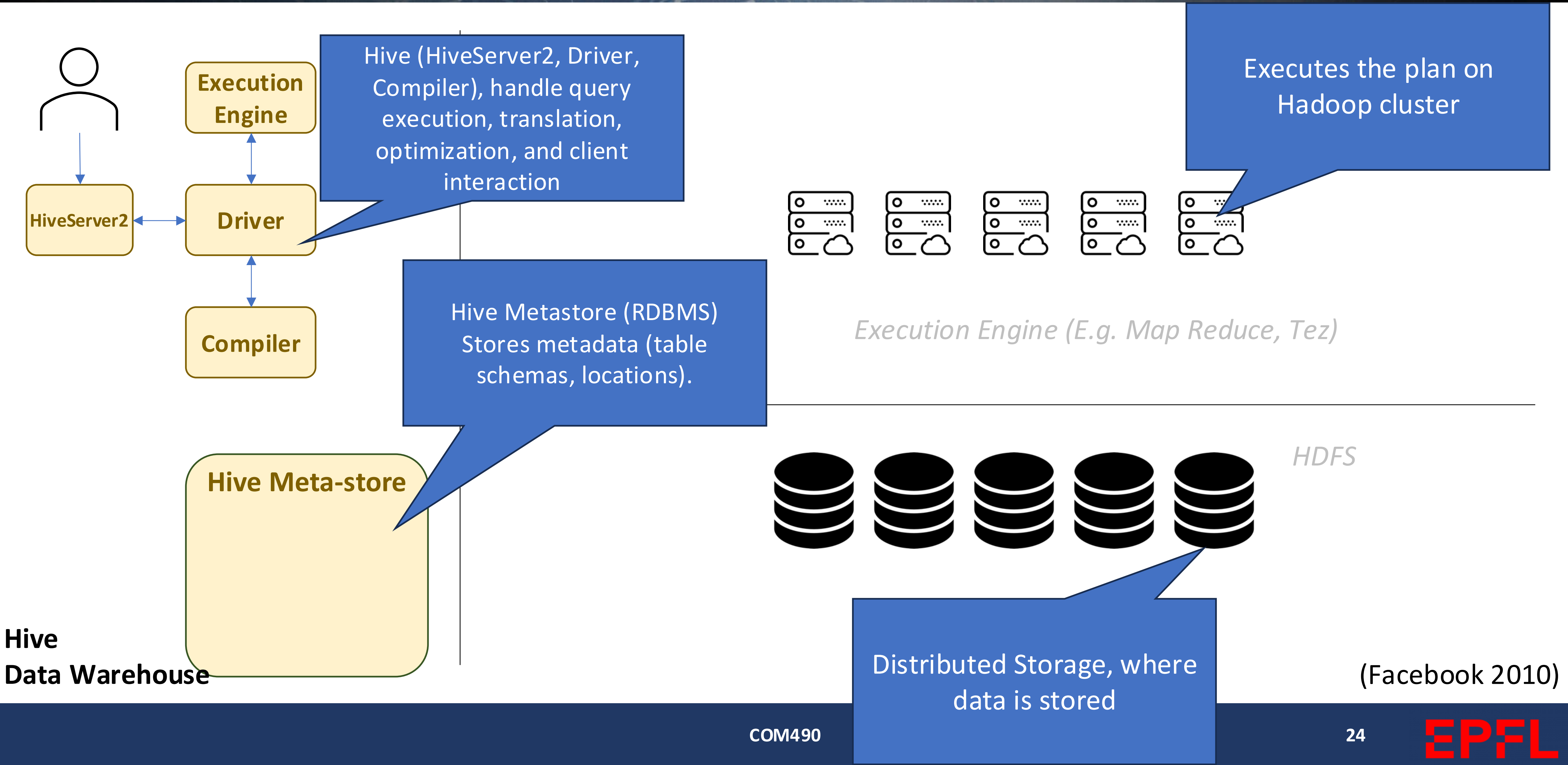
Reminder - Popular Storage Formats

- **Plain text (**csv, json, xml, …),
  - Row-oriented (most common)
  - Often sourced externally
  - Best for OLTP
  - Compression: None, Gzip, Bzip2, ...
  - Batch and stream processing
  - Splittable (if one line per record, depend on compression)
- **Parquet**
  - Column-oriented, ideal for OLAP workload
  - Integrated compression: SNAPPY, ZLIB, ZSTD, …
  - Splittable
  - Best suited for write once, read many (WORM)
  - Batch processing only
- **ORC**
  - Column-oriented, optimized for OLAP
  - Data stored in stripes (typically 250MB)
  - Indexed, splittable
  - Integrated compression: SNAPPY, ZLIB, ZSTD, …
  - Optimized for WORM
  - Batch processing only
- **Avro**
  - Row-oriented,
  - Splittable
  - Block level compression
  - Best for OLTP
  - Support schema evolution
- HDF5 / NetCDF4
  - Hierarchical, Multidimensional (D > 2)
  - Optimized for large datasets
  - Compression: ZLIB, SZIP, …
  - Splittable (with chunks)
  - Best for scientific and high-performance computing

Other Data Warehouses

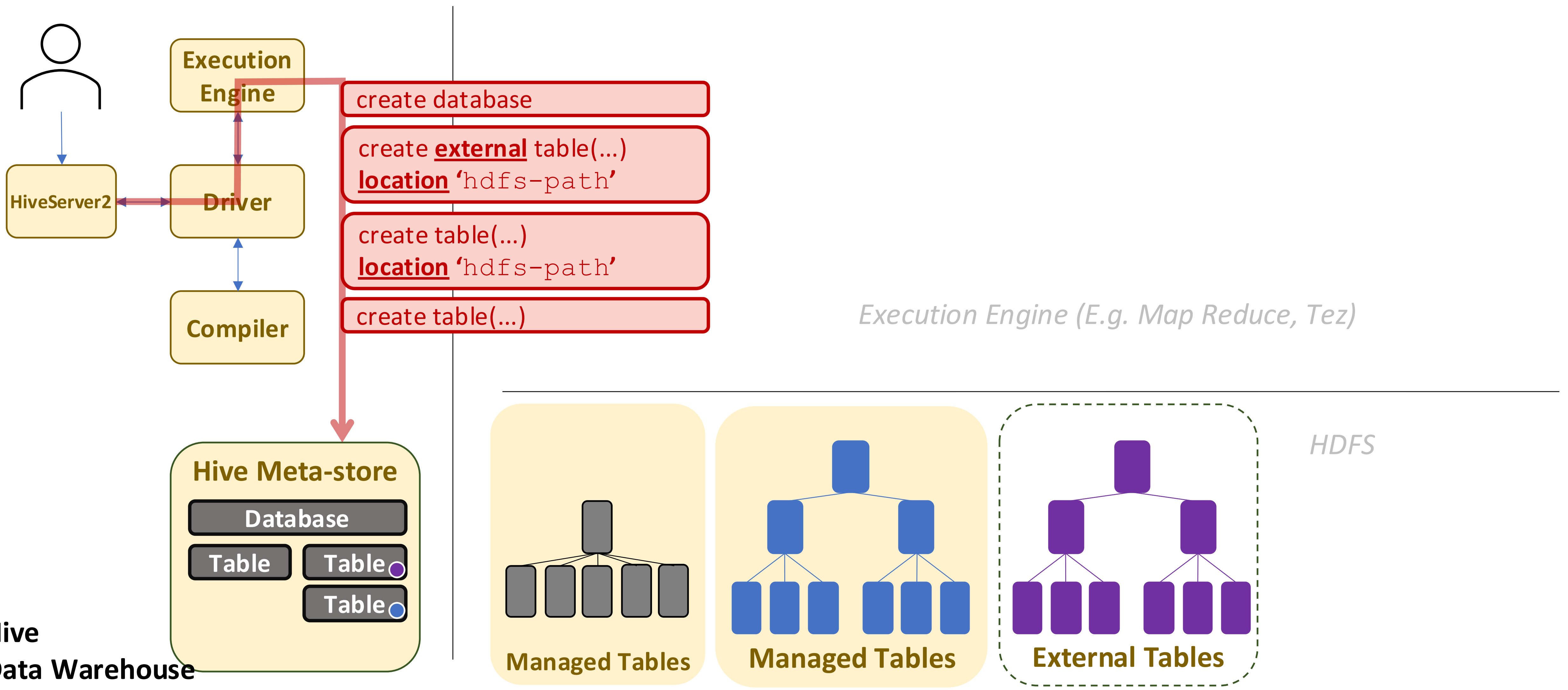
- Open Source
  - Hive - serverless
  - Presto, Trino (a fork of Presto) - server based
- Cloud
  - Amazon AWS: Athena (Trino)
  - Microsoft Azure: Synapse Analytics (T-SQL)
  - Google: BigQuery

Apache Hive Under the Hood


Execution Engine
HiveServer2
Driver
Compiler
Hive Meta-store
Hive (HiveServer2, Driver, Compiler), handle query execution, translation, optimization, and client interaction
Hive Metastore (RDBMS) Stores metadata (table schemas, locations).
Executes the plan on Hadoop cluster
Execution Engine (E.g. Map Reduce, Tez)
HDFS
Distributed Storage, where data is stored
Hive
Data Warehouse

(Facebook 2010)

Hive Under the Hood



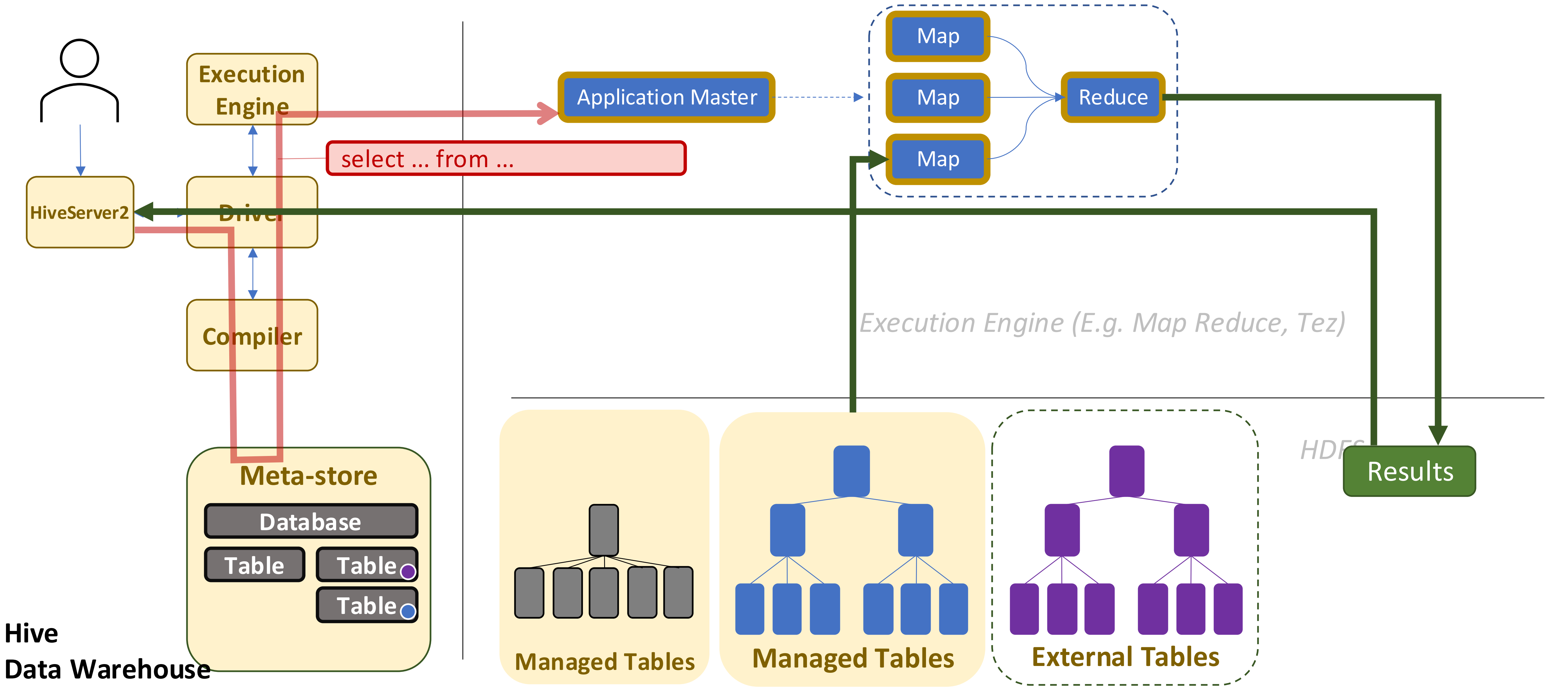
create database

create **external** table(…)
location 'hdfs-path'

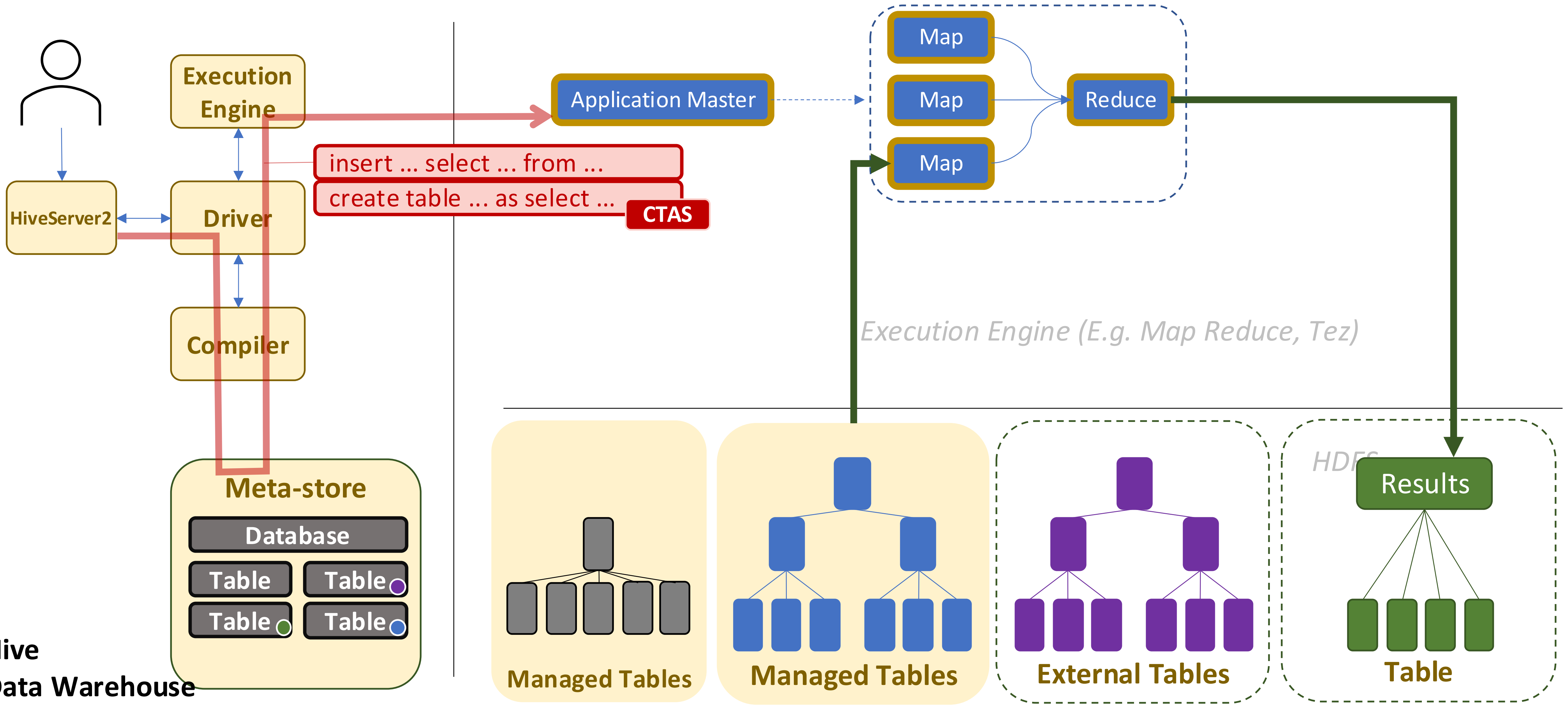
create table(…)
location 'hdfs-path'

create table(…)

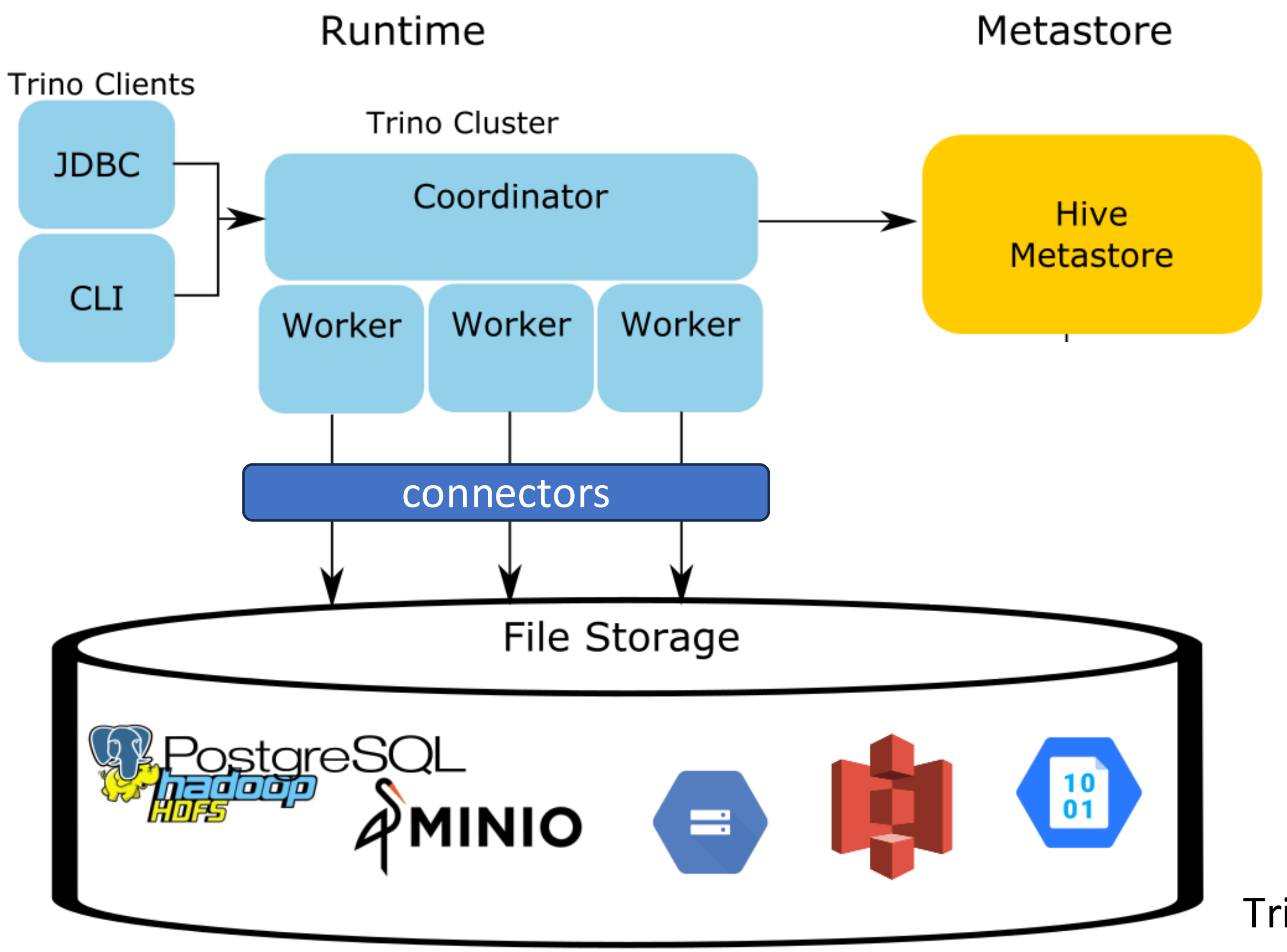
Hive Under the Hood



Hive Under the Hood



Trino Under the Hood



Trino - fork of Presto (Facebook 2013)

Start your engines

https://dslabgit.datascience.ch/course/2025/module-2b