GeekBrains

SQL в HADOOP

SQL to MapReduce, PIG vs Hive vs Impala, Hive QL

РАЗБОР ПРАКТИЧЕСКОГО ЗАДАНИЯ

РАЗБОР ПРАКТИЧЕСКОГО ЗАДАНИЯ

1. Может ли стадия Reduce начаться до завершения стадии Map? Почему?

Фаза Reduce состоит из трёх этапов: шафл, сортировка, свертка. Непосредственно свертка начнется только после окончания сортировки, а это невозможно пока не будут получены все данные от маппероов. При этом, этап шафла можно начать до завершения всех маппероов, изменив поле `mapreduce.job.reduce.slowstart.completedmaps` *в конфиге*

РАЗБОР ПРАКТИЧЕСКОГО ЗАДАНИЯ

1. Может ли стадия Reduce начаться до завершения стадии Map? Почему?

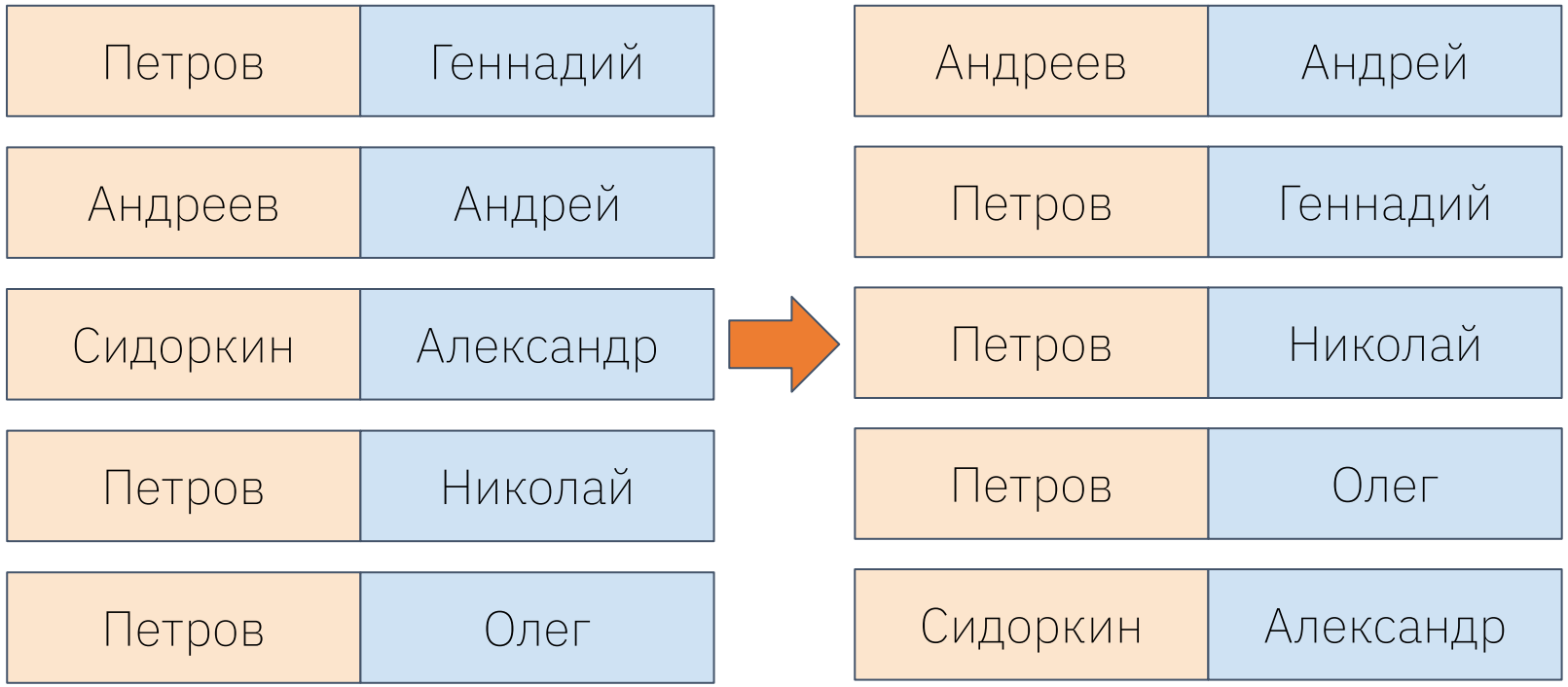
Фаза Reduce состоит из трёх этапов: шафл, сортировка, свертка. Непосредственно свертка начнется только после окончания сортировки, а это невозможно пока не будут получены все данные от мапперов. При этом, этап шафла можно начать до завершения всех мапперов, изменив поле `mapreduce.job.reduce.slowstart.completedmaps` *в конфиге*

Петров	Геннадий
Андреев	Андрей
Сидоркин	Александр
Петров	Николай
Петров	Олег

РАЗБОР ПРАКТИЧЕСКОГО ЗАДАНИЯ

1. Может ли стадия Reduce начаться до завершения стадии Map? Почему?

Фаза Reduce состоит из трёх этапов: шафл, сортировка, свертка. Непосредственно свертка начнется только после окончания сортировки, а это невозможно пока не будут получены все данные от мапперов. При этом, этап шафла можно начать до завершения всех мапперов, изменив поле `mapreduce.job.reduce.slowstart.completedmaps` *в конфиге*



РАЗБОР ПРАКТИЧЕСКОГО ЗАДАНИЯ

1. Может ли стадия Reduce начаться до завершения стадии Map? Почему?

Фаза Reduce состоит из трёх этапов: шафл, сортировка, свертка. Непосредственно свертка начнется только после окончания сортировки, а это невозможно пока не будут получены все данные от мапперов. При этом, этап шафла можно начать до завершения всех мапперов, изменив поле `mapreduce.job.reduce.slowstart.completedmaps` *в конфиге*

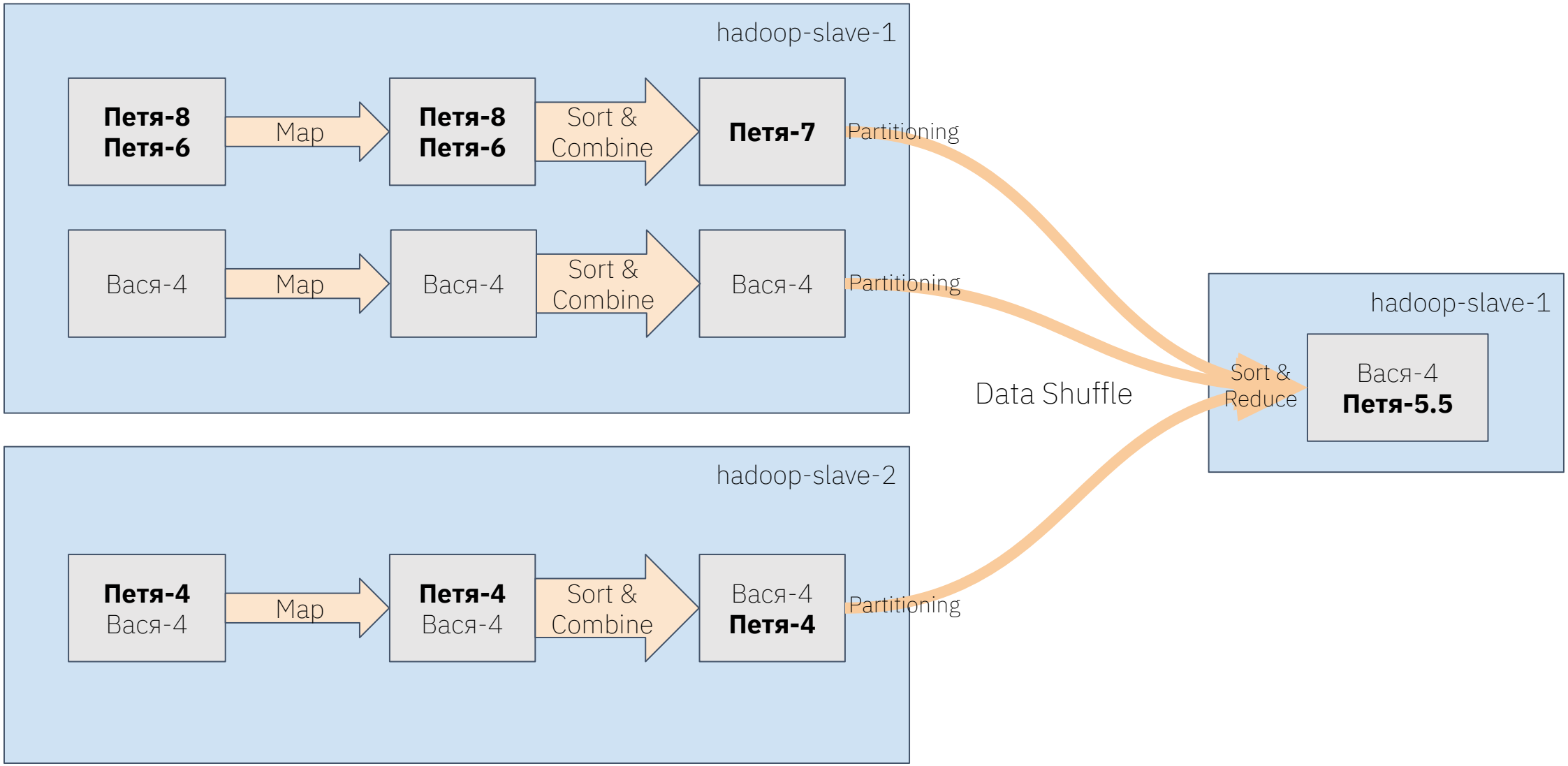
РАЗБОР ПРАКТИЧЕСКОГО ЗАДАНИЯ

6. На кластере лежит датасет, в котором ключами является id сотрудника и дата, а значением размер выплаты. Руководитель поставил задачу рассчитать среднюю сумму выплат по каждому сотруднику за последний месяц. В маппере вы отфильтровали старые записи и отдали ключ-значение вида: id-money. А в редьюсере суммировали все входящие числа и поделили результат на их количество. Но вам в голову пришла идея оптимизировать расчет, поставив этот же редьюсер и в качестве комбинатора, тем самым уменьшив шафл данных. Можете ли вы так сделать? Если да, то где можно было допустить ошибку? Если нет, то что должно быть на выходе комбинатора?

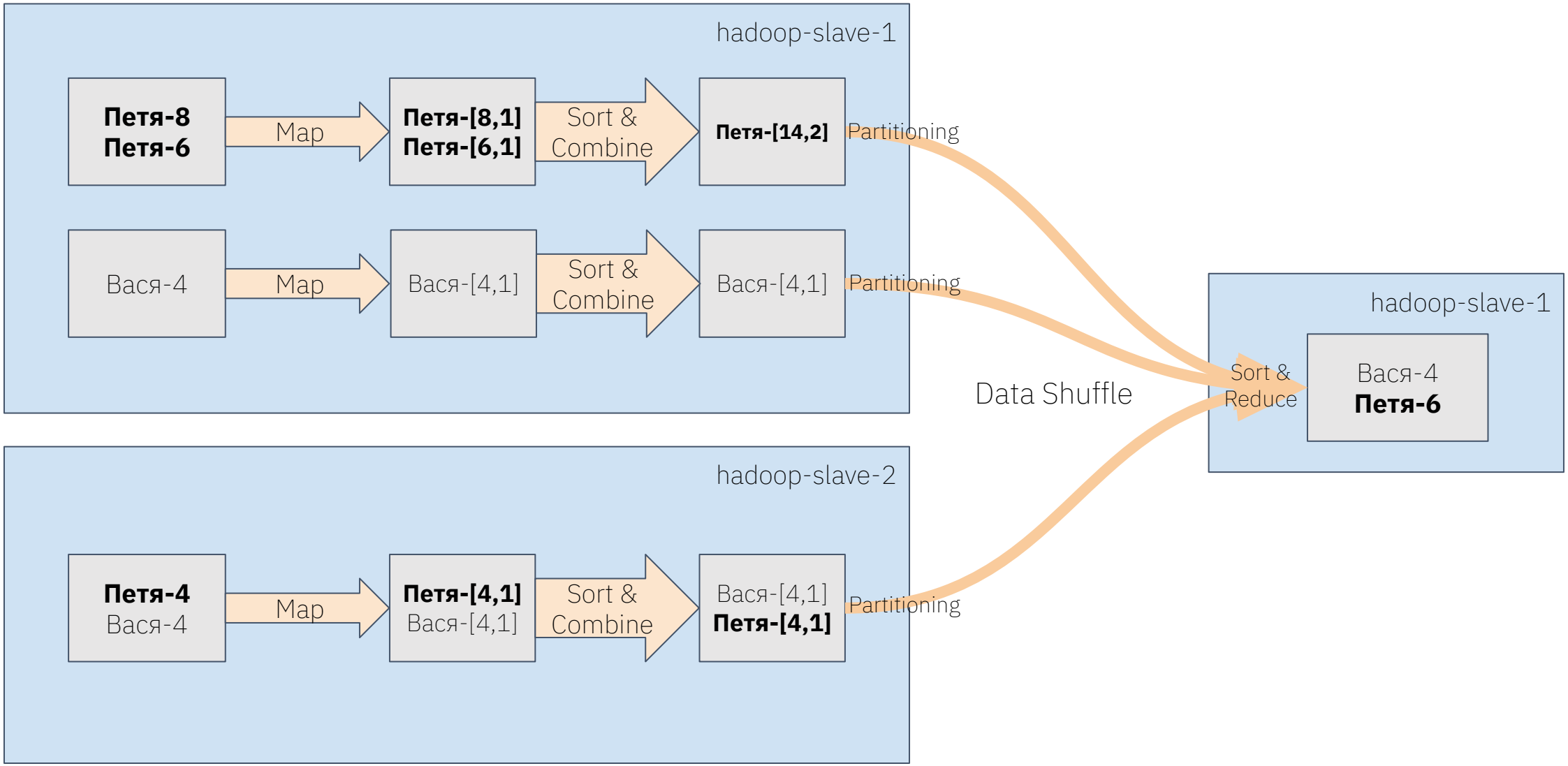
Мы не можем так сделать, так как количество начислений на каждой из нод кластера может быть разным. В таком случае, на комбинаторах будет потеряно это значение и среднее арифметическое будет неправильным.

Но мы можем написать комбинатор, который будет для каждого сотрудника отдавать пару значений: общая сумма и количество начислений. После чего на редьюсере объеденить все суммы и разделить на общее количество начислений.

РАЗБОР ПРАКТИЧЕСКОГО ЗАДАНИЯ: ОШИБКА



РАЗБОР ПРАКТИЧЕСКОГО ЗАДАНИЯ: ПРАВИЛЬНО



ИСТОРИЯ HIVE

ИСТОРИЯ ПОЯВЛЕНИЯ

Facebook начал использовать Hadoop в качестве решения для хранения непрерывно растущего потока данных

ИСТОРИЯ ПОЯВЛЕНИЯ

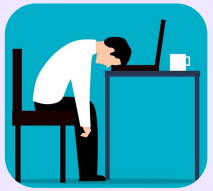
Facebook начал использовать Hadoop в качестве решения для хранения непрерывно растущего потока данных

Сложные запросы к данным в Hadoop требуют написания нескольких последовательных MapReduce задач на сотни строк кода и следить за их стабильностью

ИСТОРИЯ ПОЯВЛЕНИЯ

Facebook начал использовать Hadoop в качестве решения для хранения непрерывно растущего потока данных

Не все пользователи данных владеют Java или другим ЯП на достаточном уровне, чтобы написать задачу

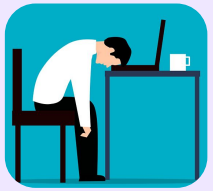


Сложные запросы к данным в Hadoop требуют написания нескольких последовательных MapReduce задач на сотни строк кода и следить за их стабильностью

ИСТОРИЯ ПОЯВЛЕНИЯ

Facebook начал использовать Hadoop в качестве решения для хранения непрерывно растущего потока данных

Не все пользователи данных владеют Java или другим ЯП на достаточном уровне, чтобы написать задачу



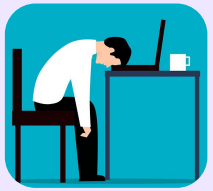
Сложные запросы к данным в Hadoop требуют написания нескольких последовательных MapReduce задач на сотни строк кода и следить за их стабильностью

Зато все, кому нужно работать с данными, владеют SQL

ИСТОРИЯ ПОЯВЛЕНИЯ

Facebook начал использовать Hadoop в качестве решения для хранения непрерывно растущего потока данных

Не все пользователи данных владеют Java или другим ЯП на достаточном уровне, чтобы написать задачу



Был разработан инструмент, который реализовал концепцию таблиц и колоночного представления данных поверх MapReduce, чтобы с ним можно было работать, как с SQL

Сложные запросы к данным в Hadoop требуют написания нескольких последовательных MapReduce задач на сотни строк кода и следить за их стабильностью

Зато все, кому нужно работать с данными, владеют SQL

APACHE HIVE

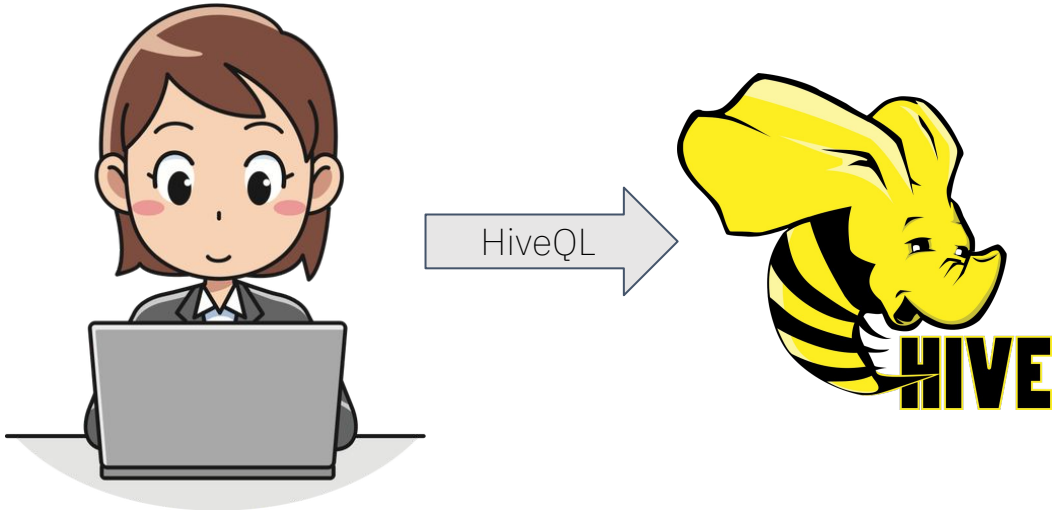
APACHE HIVE

Hive это система хранения данных, предоставляющая средства для работы и анализа больших датасетов, хранящихся в HDFS

Hive использует язык запросов HiveQL, который является подмножеством SQL

APACHE HIVE

APACHE HIVE



APACHE HIVE

Hive **преобразует** HiveQL запрос в серию MapReduce задач, выполняемых в кластере

АРХИТЕКТУРА

АРХИТЕКТУРА

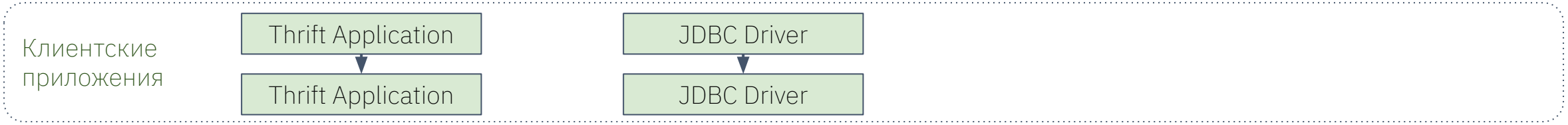
Клиентские
приложения

Hive поддерживает несколько типов клиентских приложений, которые могут быть написаны на разных языках программирования

АРХИТЕКТУРА

Thrift это RPC фреймворк, позволяющий вызывать удаленные методы других приложений

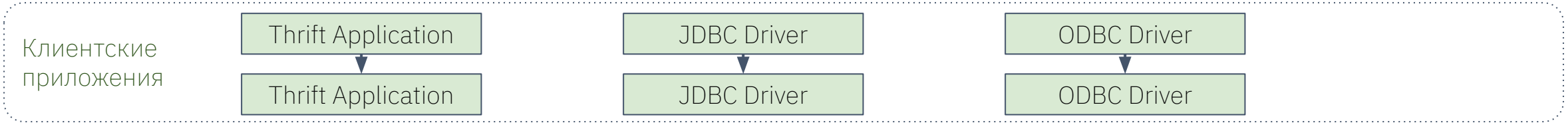
АРХИТЕКТУРА



JDBC — Java Database Connectivity

Позволяет JVM приложениям через JDBC драйвер работать с различными базами данных, отправляя на них SQL запросы и обрабатывающая ответы

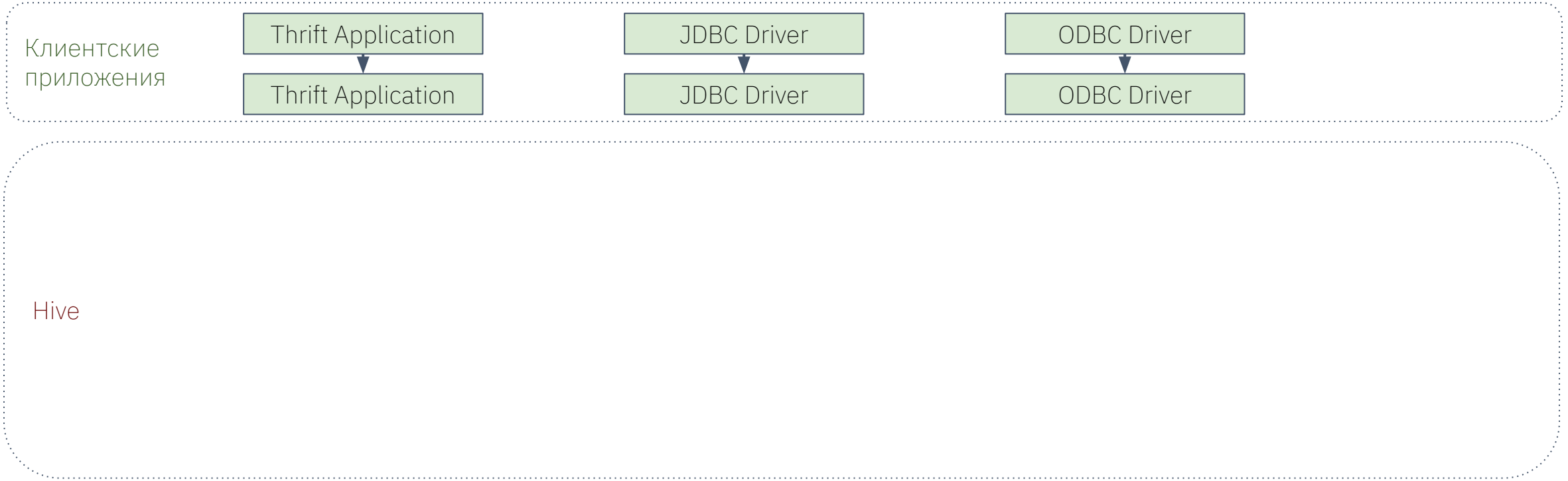
АРХИТЕКТУРА



ODBC — Open Database Connectivity

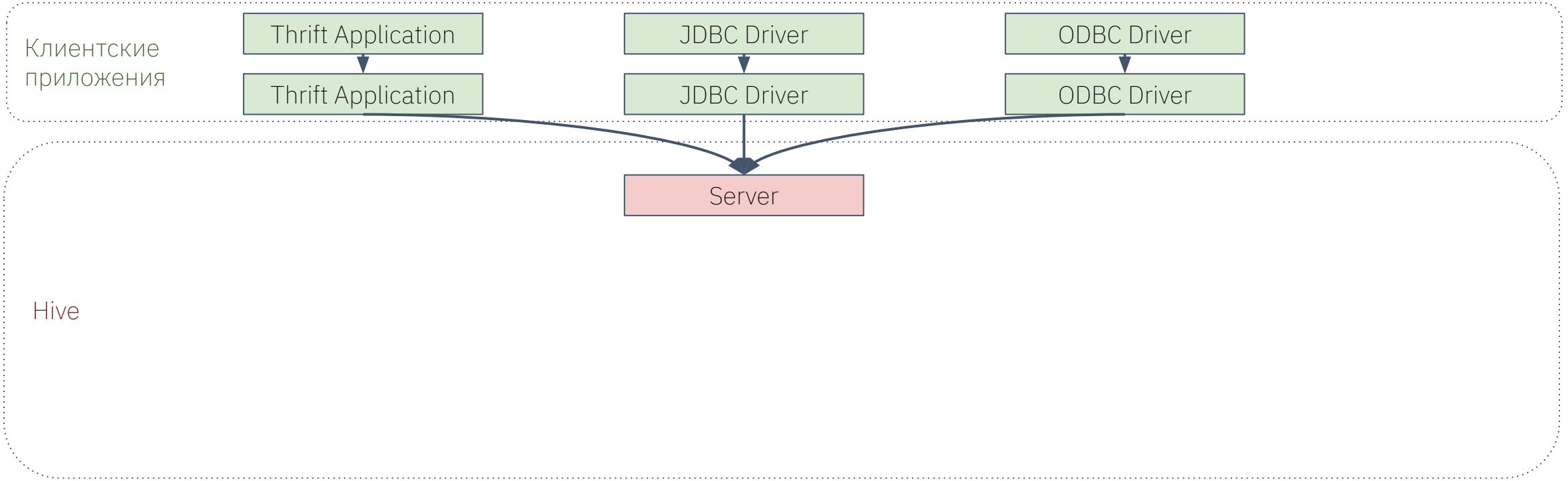
Аналогичная JDBC разработка от Microsoft

АРХИТЕКТУРА



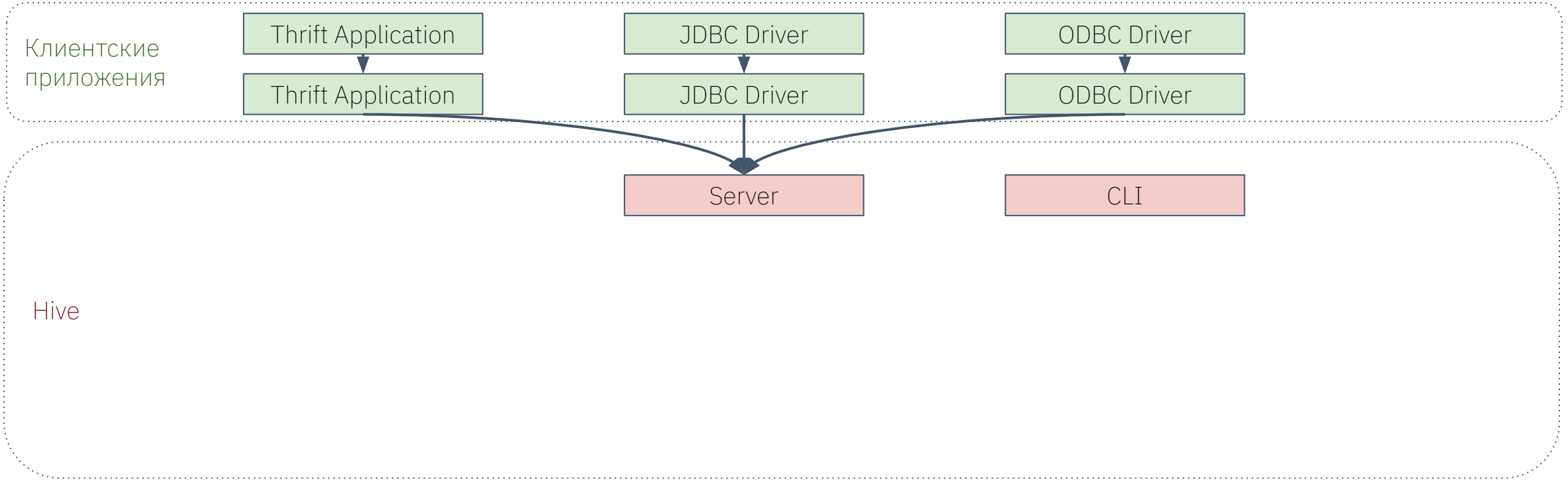
Hive состоит из нескольких сервисов, обеспечивающих его работу

АРХИТЕКТУРА



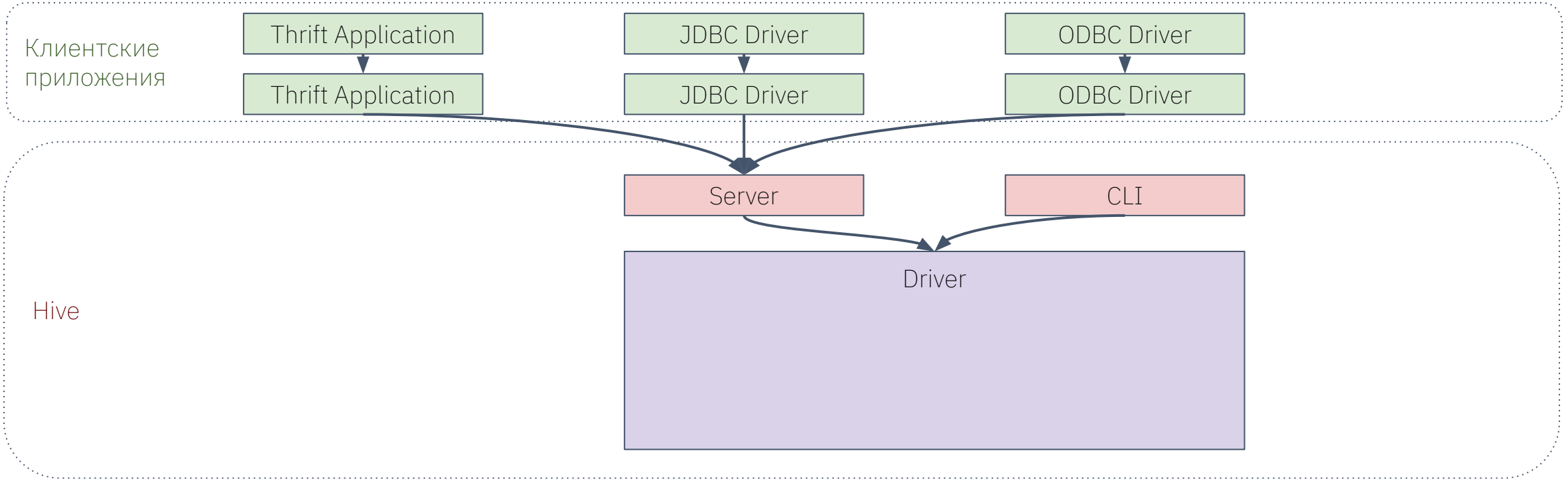
Все клиентские запросы отправляются на сервер

АРХИТЕКТУРА



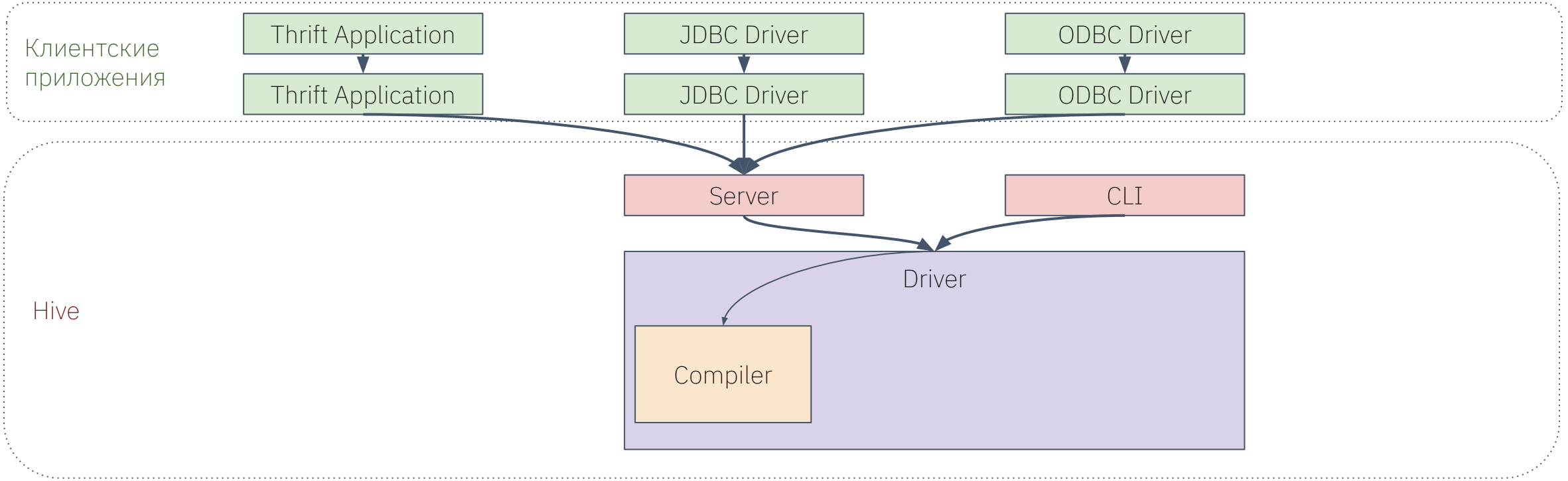
Интерактивная оболочка позволяет обрабатывать запросы напрямую

АРХИТЕКТУРА



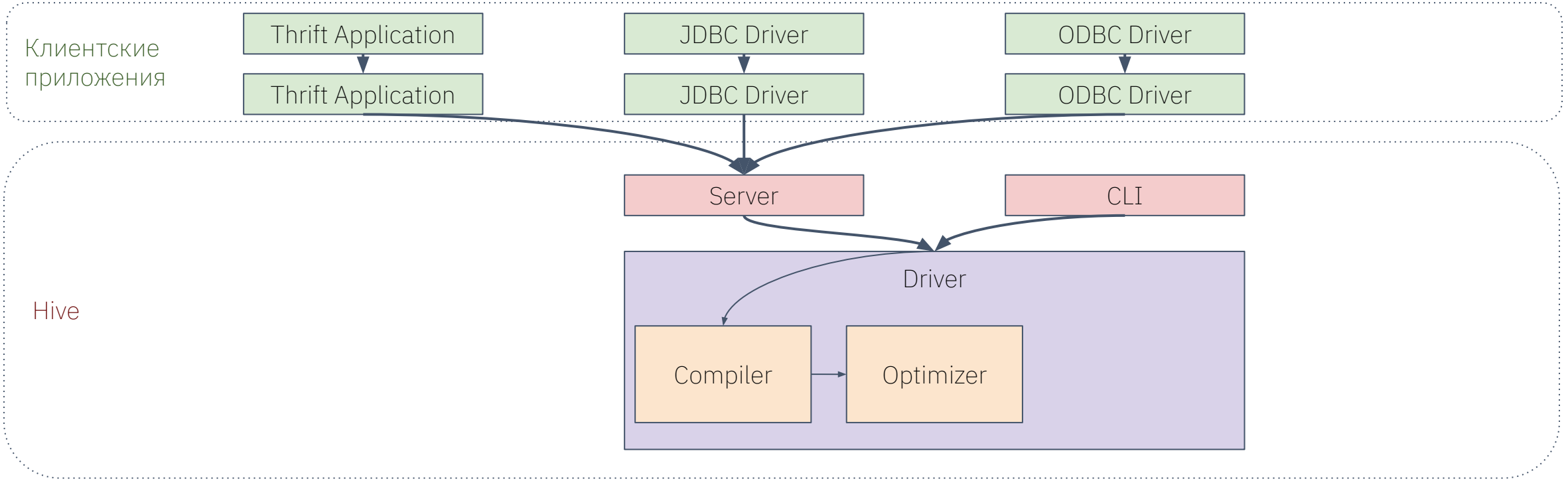
Драйвер отвечает за обработку и исполнение запросов и состоит из трех компонентов

АРХИТЕКТУРА



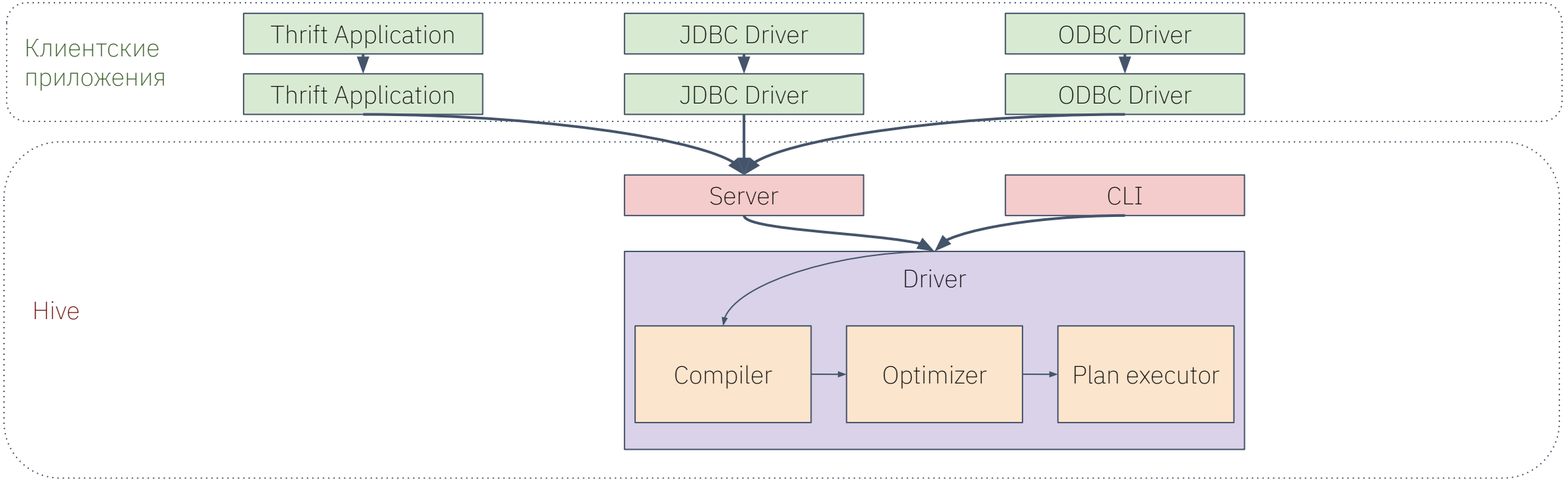
Компилятор анализирует HiveQL запрос и проверяет его на корректность

АРХИТЕКТУРА



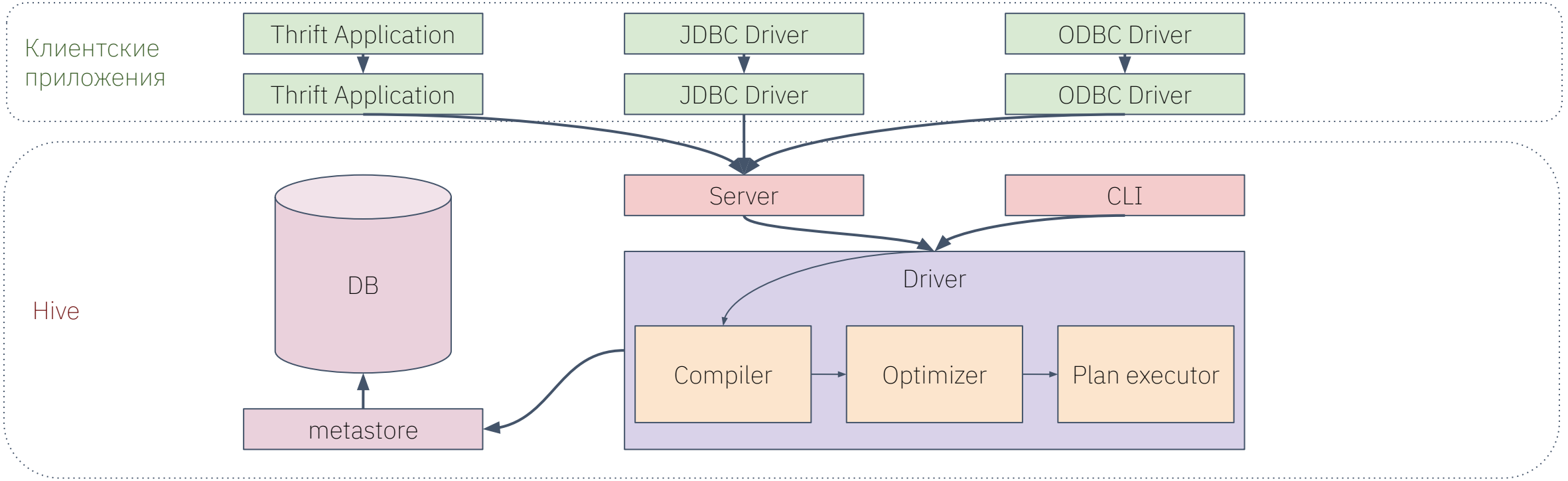
Оптимизатор составляет оптимальный план MapReduce и HDFS задач в виде графа

АРХИТЕКТУРА



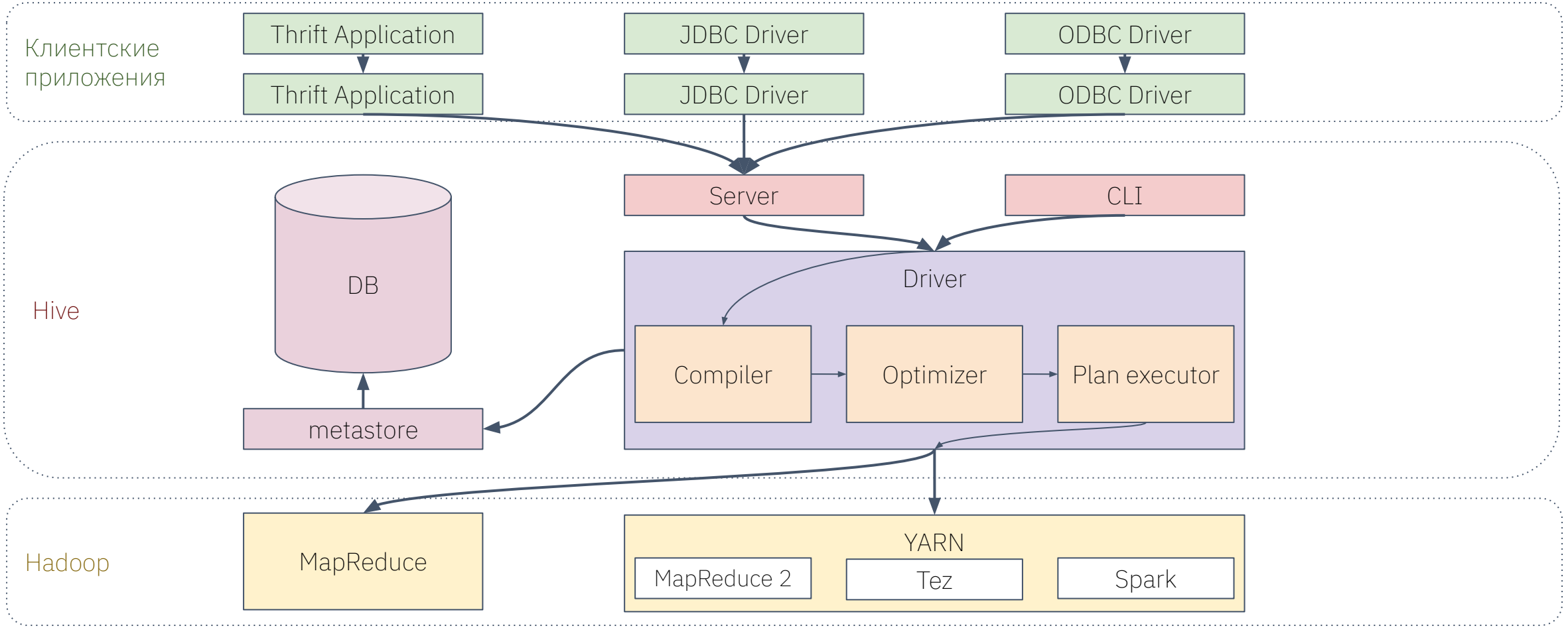
Исполнитель отвечает за запуск и исполнения графа задач

АРХИТЕКТУРА



Метахранилище содержит все метаданные Hive

АРХИТЕКТУРА



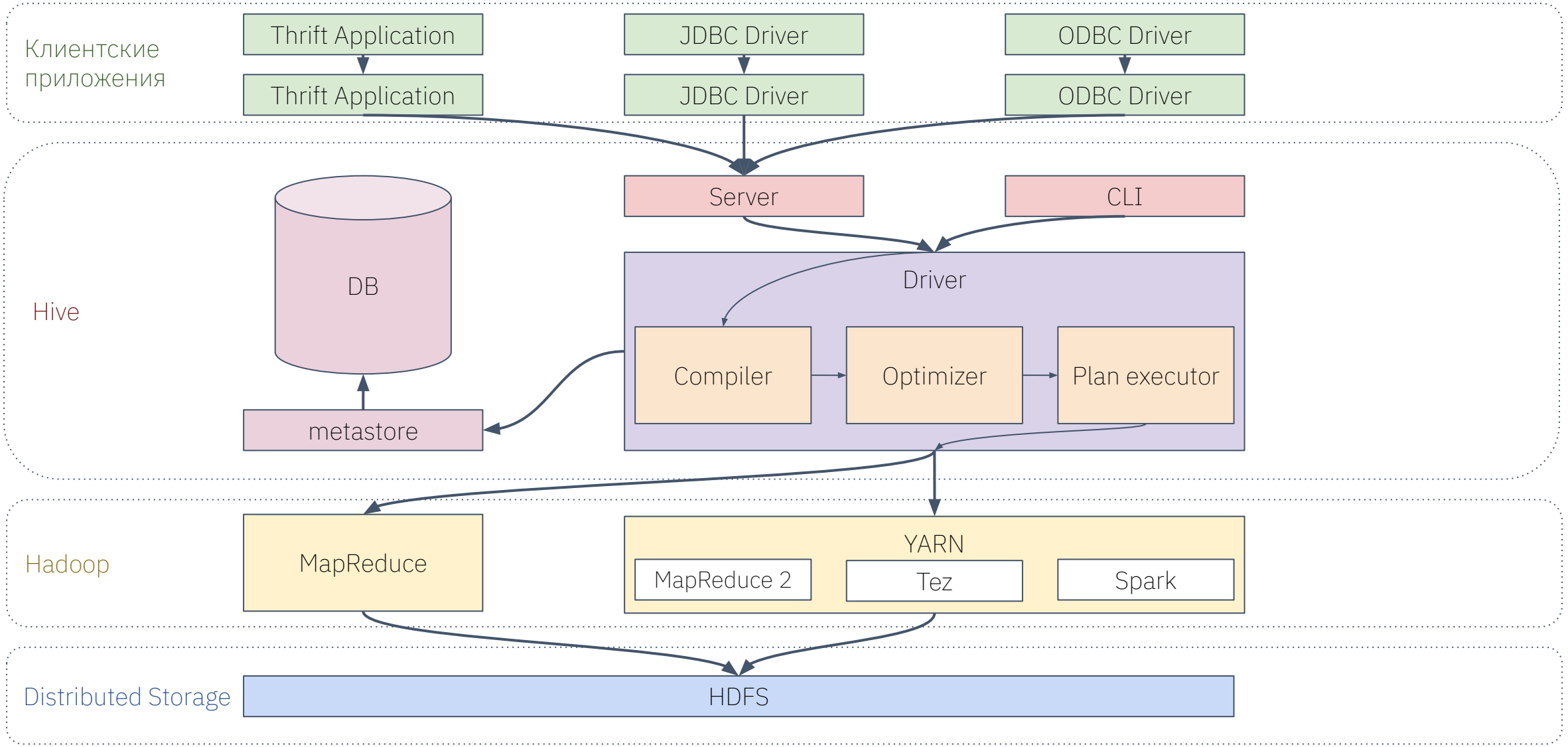
ДВИЖКИ HIVE


```
hive.execution.engine=mr
```

```
hive.execution.engine=spar
k
```

```
hive.execution.engine=tez
```

АРХИТЕКТУРА



МЕТАХРАНИЛИЩЕ

Репозиторий метаданных Hive. Состоит из двух компонентов:

- daemon
- база данных

По умолчанию в качестве базы данных используется Apache Derby, работающий в одном JVM процессе с Hive. В этом случае может быть открыт только один сеанс работы.

Для поддержки множества пользователей следует использовать любую JDBC-совместимую базу данных (MySQL, PostgreSQL...)

СХЕМА РАБОТЫ

СХЕМА РАБОТЫ

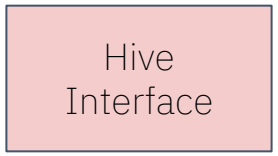


СХЕМА РАБОТЫ

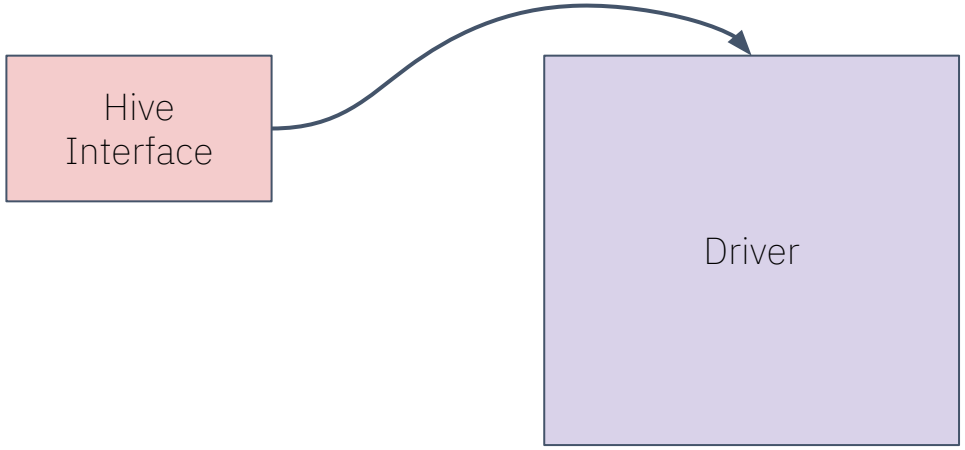


СХЕМА РАБОТЫ

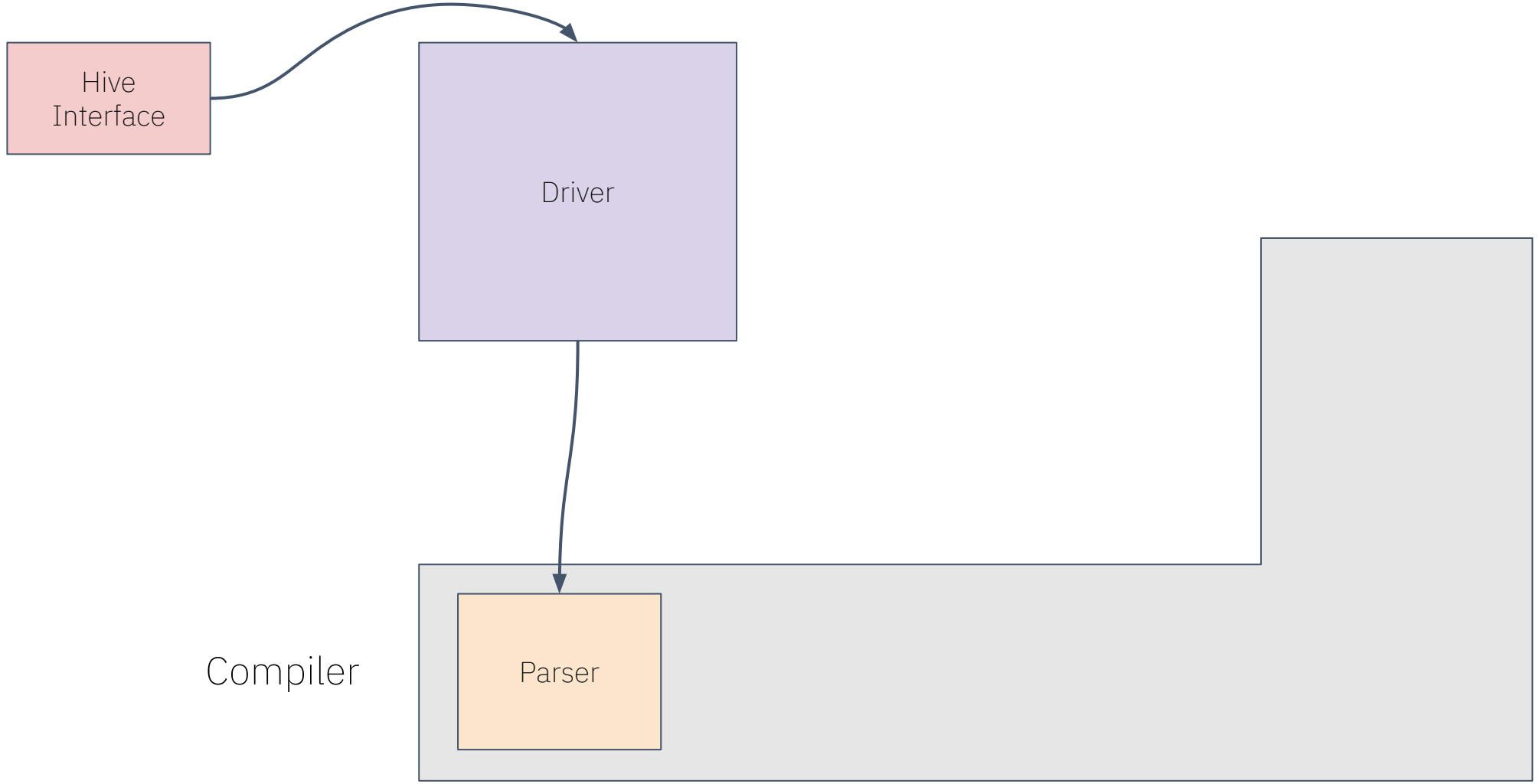


СХЕМА РАБОТЫ

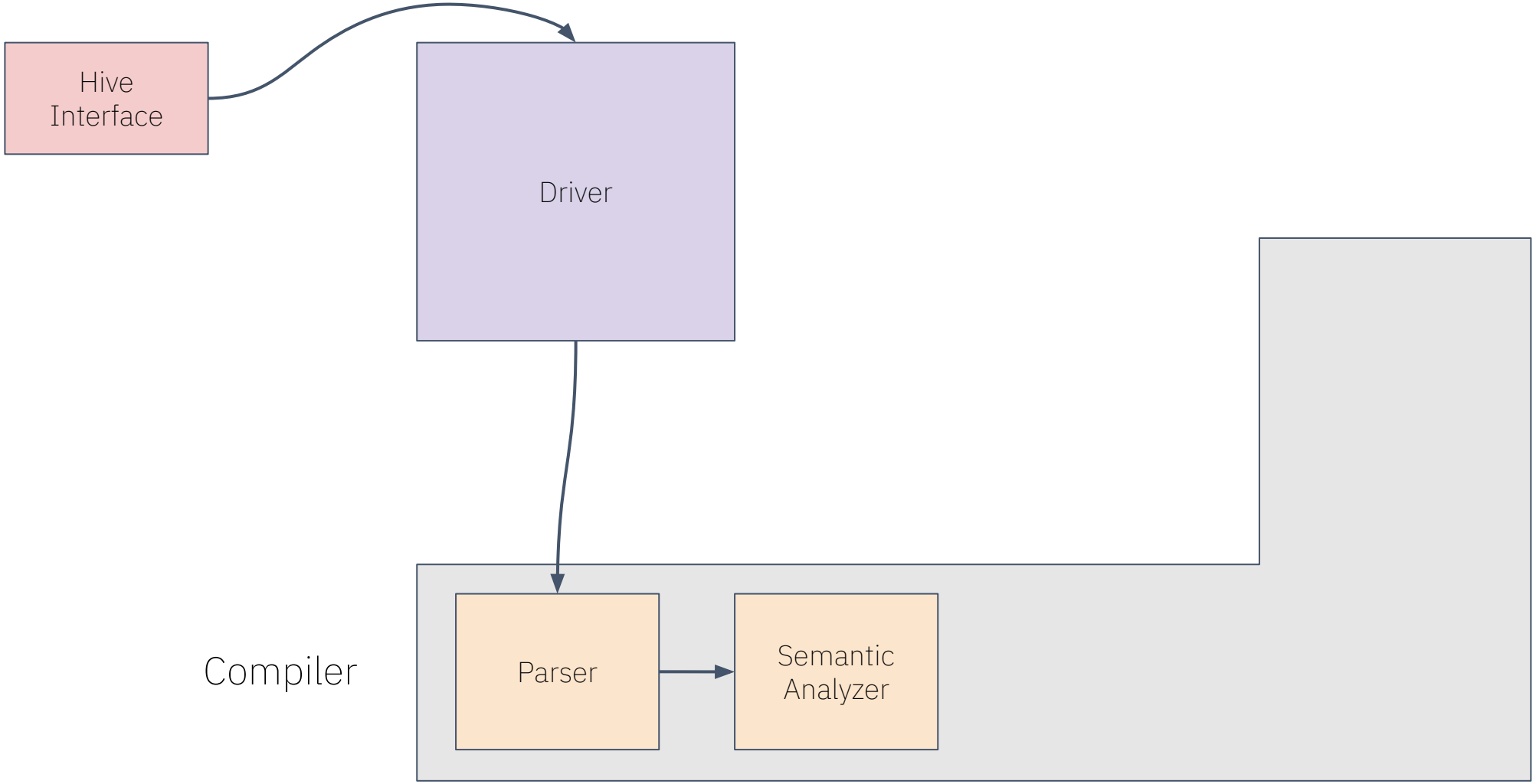


СХЕМА РАБОТЫ

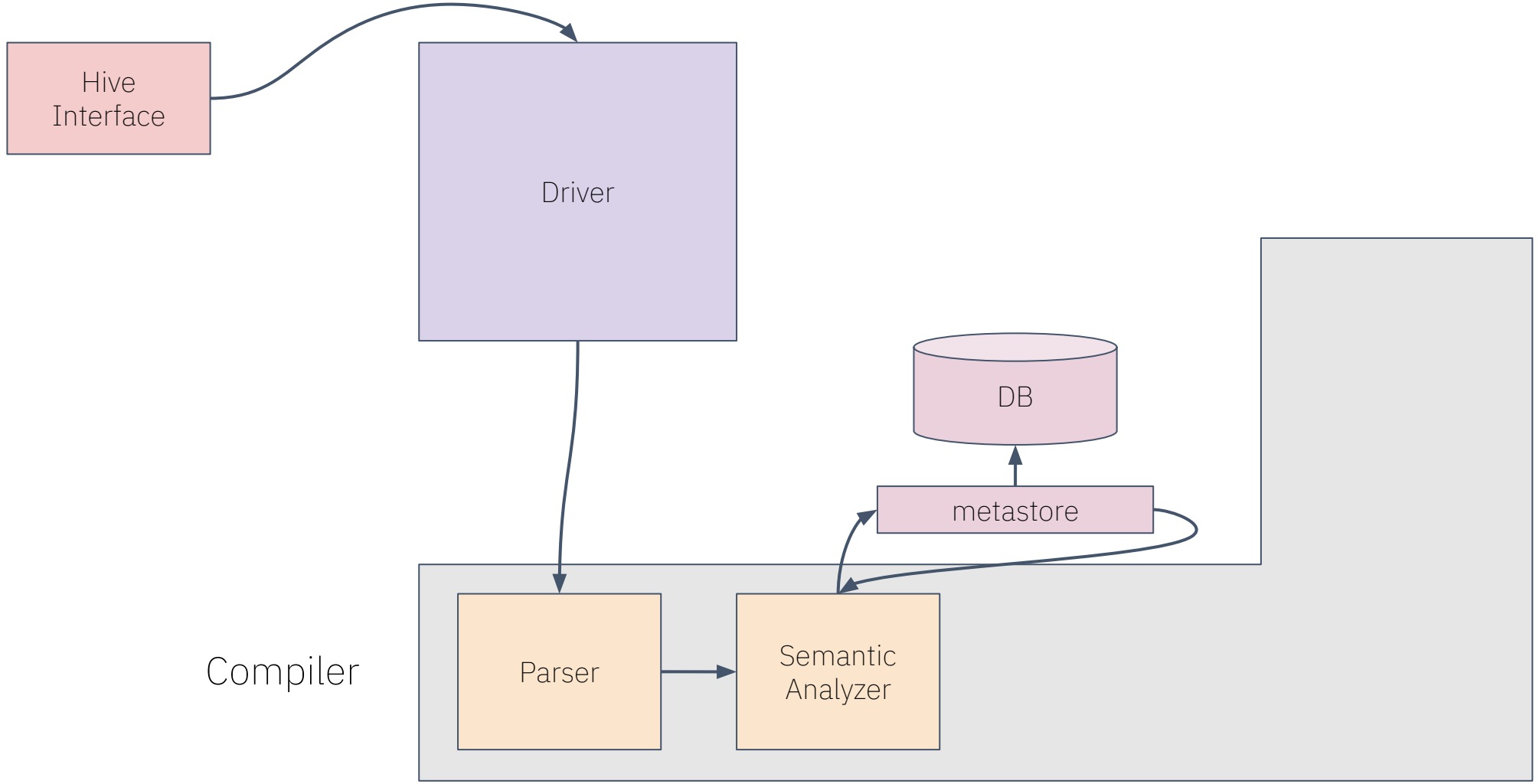


СХЕМА РАБОТЫ

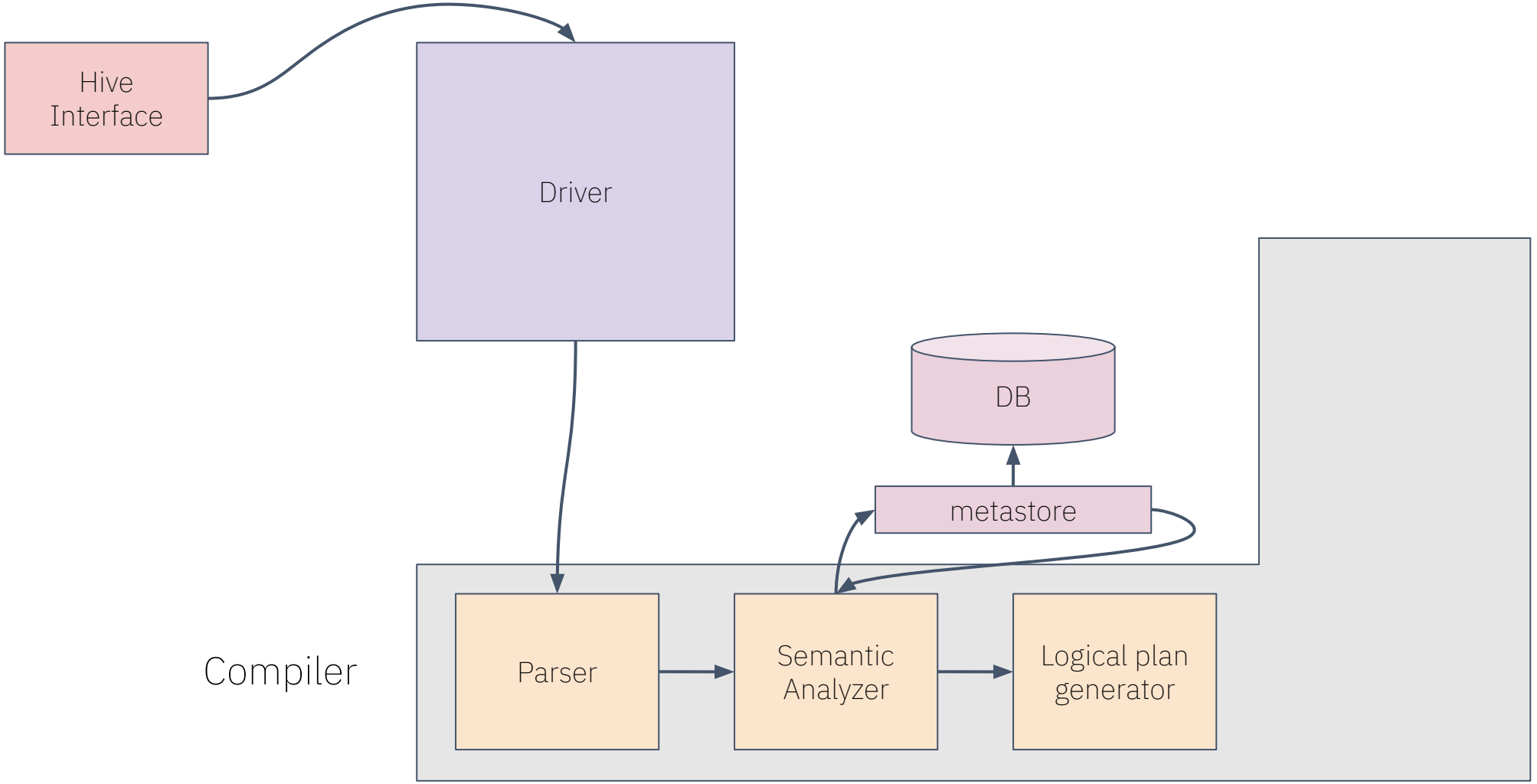


СХЕМА РАБОТЫ

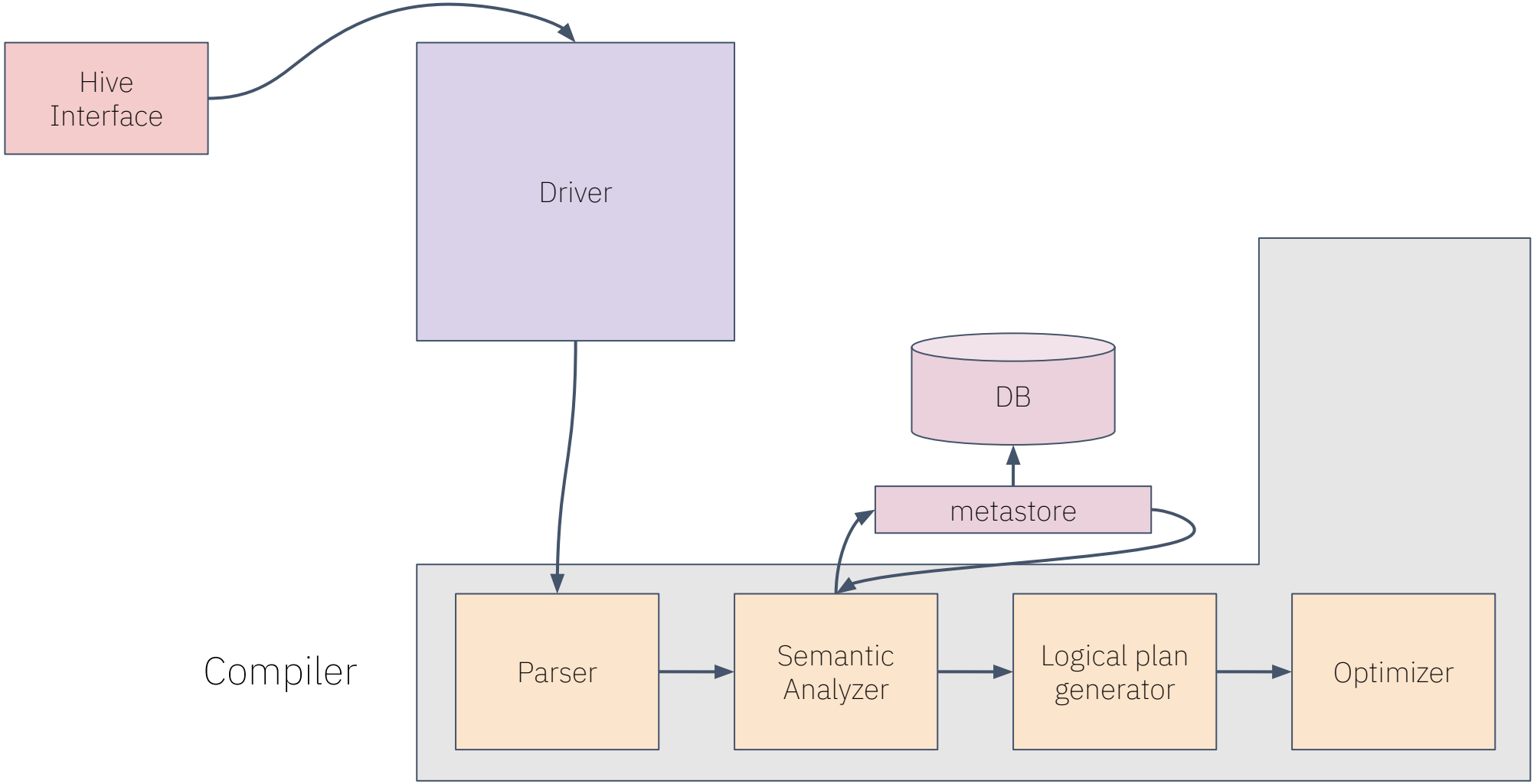


СХЕМА РАБОТЫ

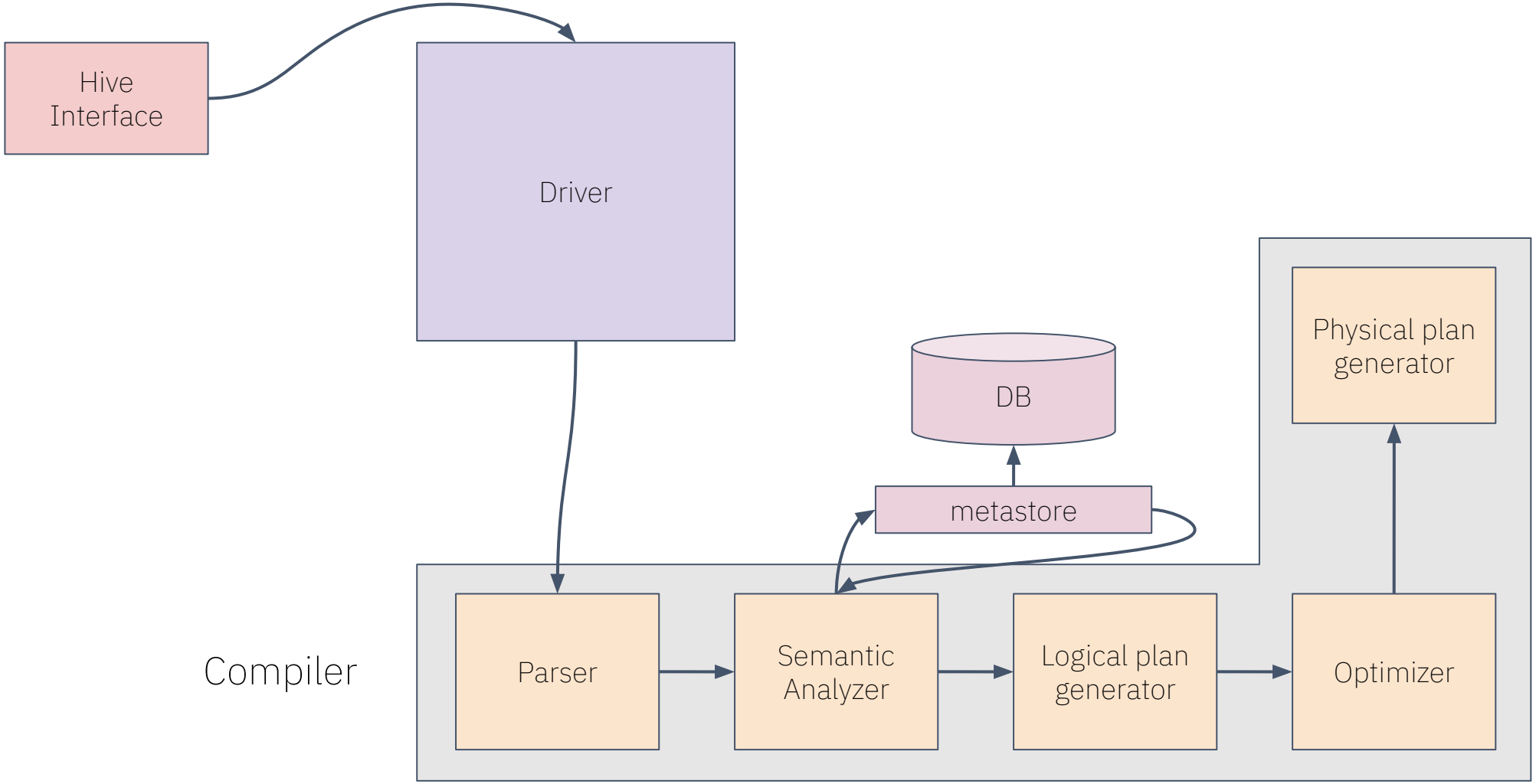


СХЕМА РАБОТЫ

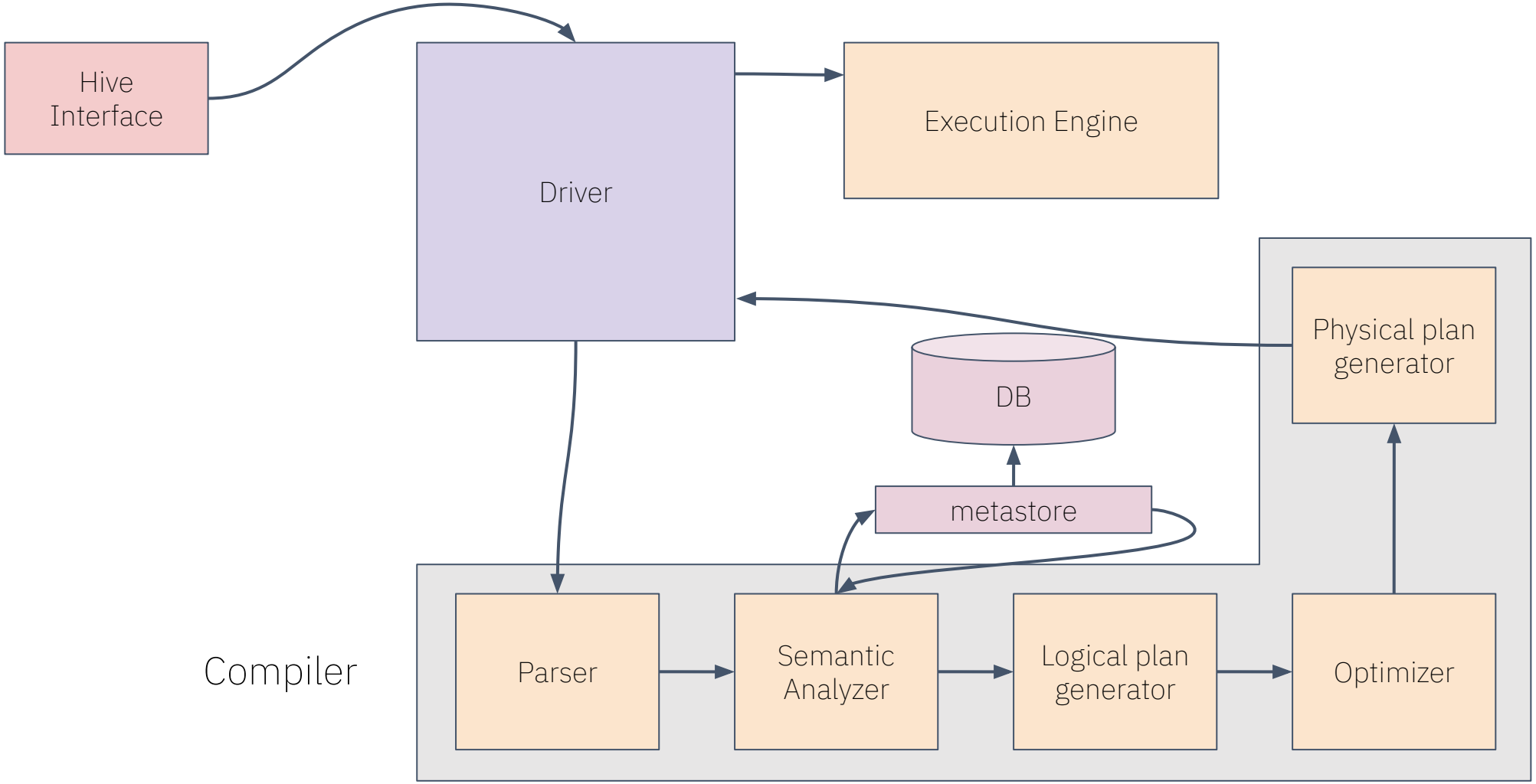


СХЕМА РАБОТЫ

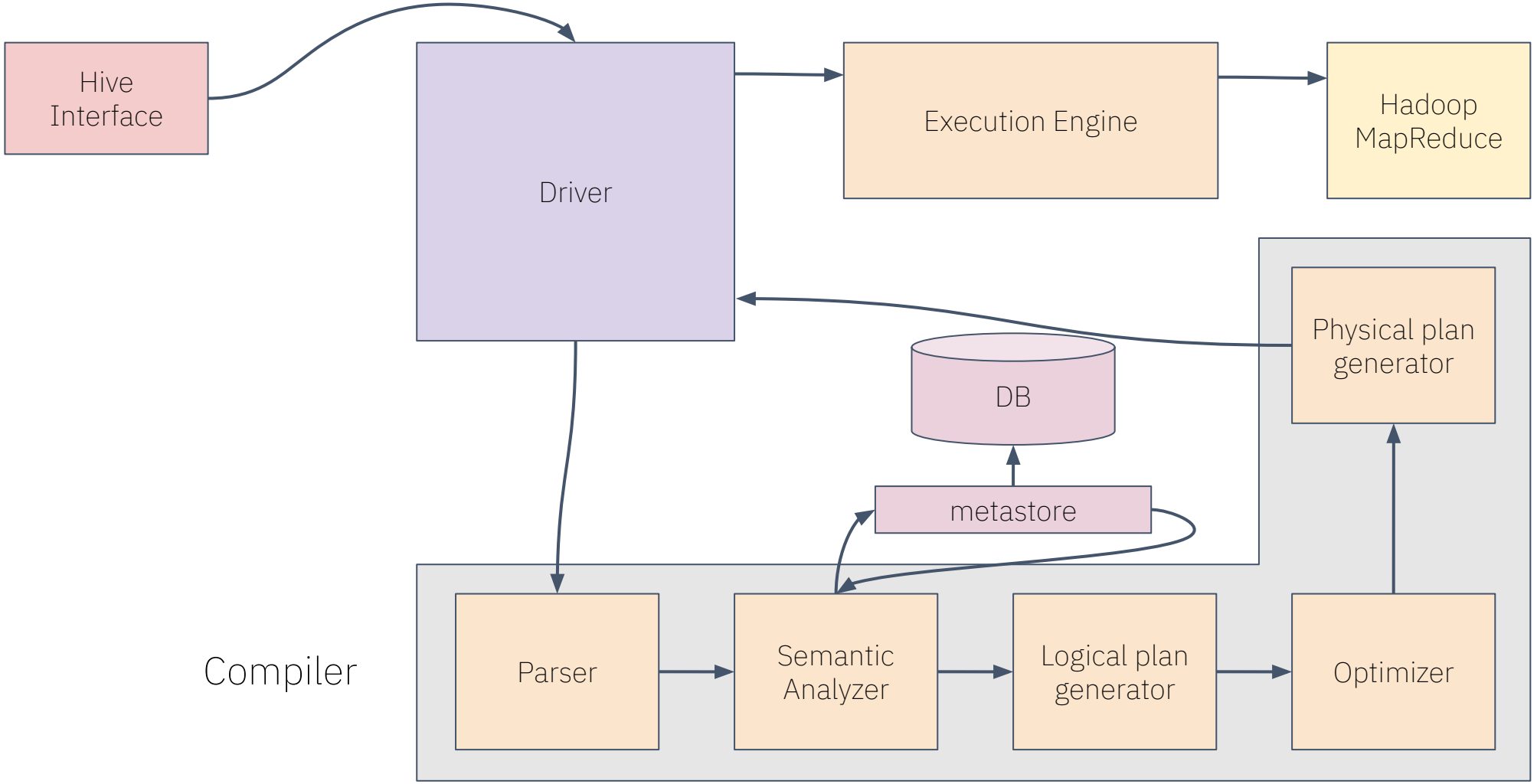
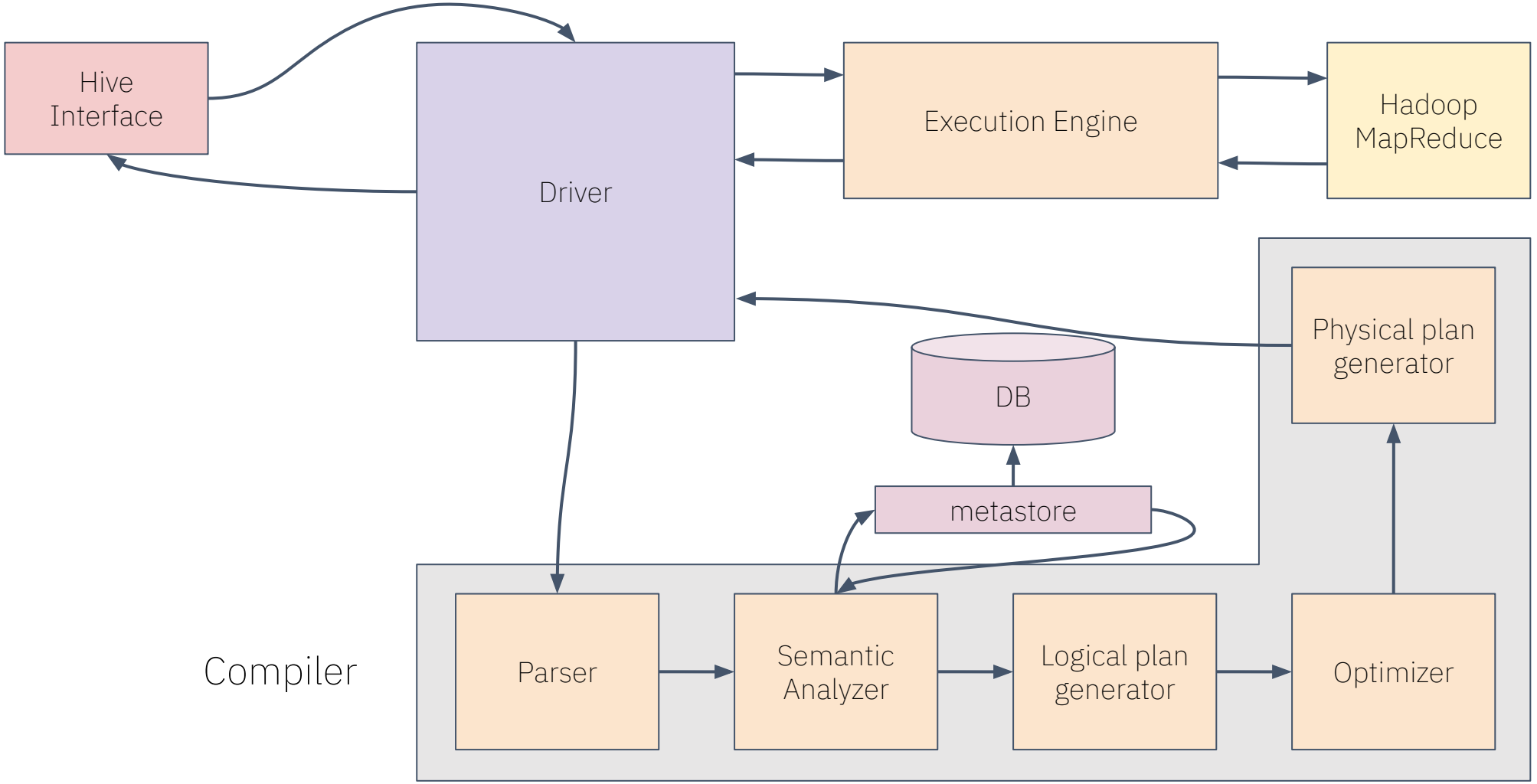
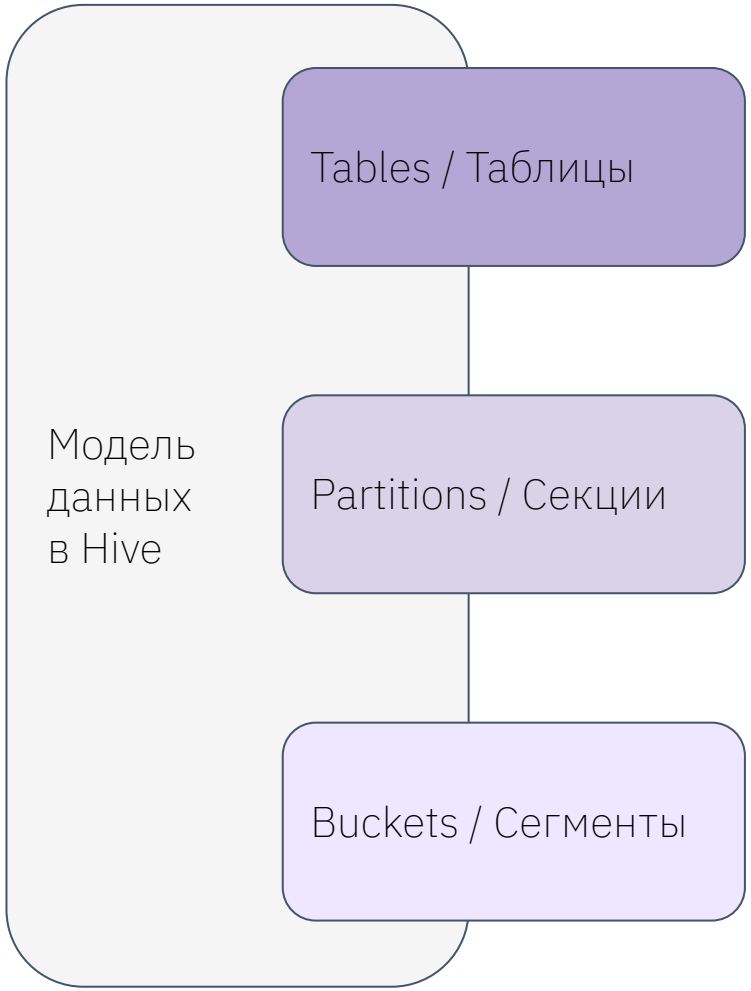


СХЕМА РАБОТЫ

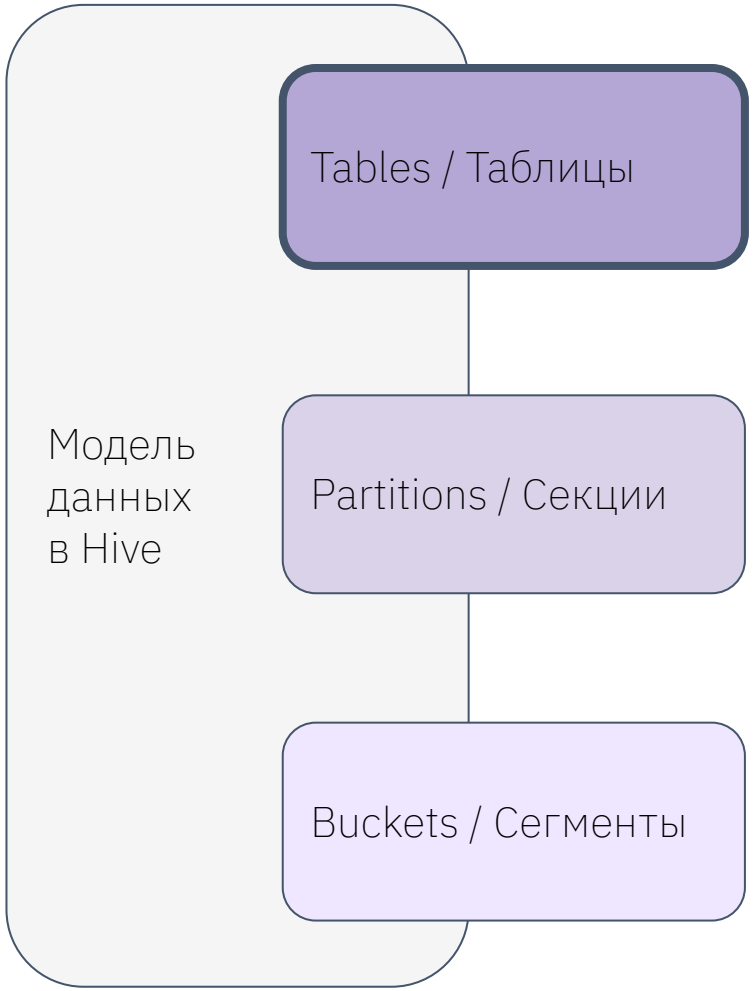


МОДЕЛЬ ДАННЫХ

МОДЕЛЬ ДАННЫХ



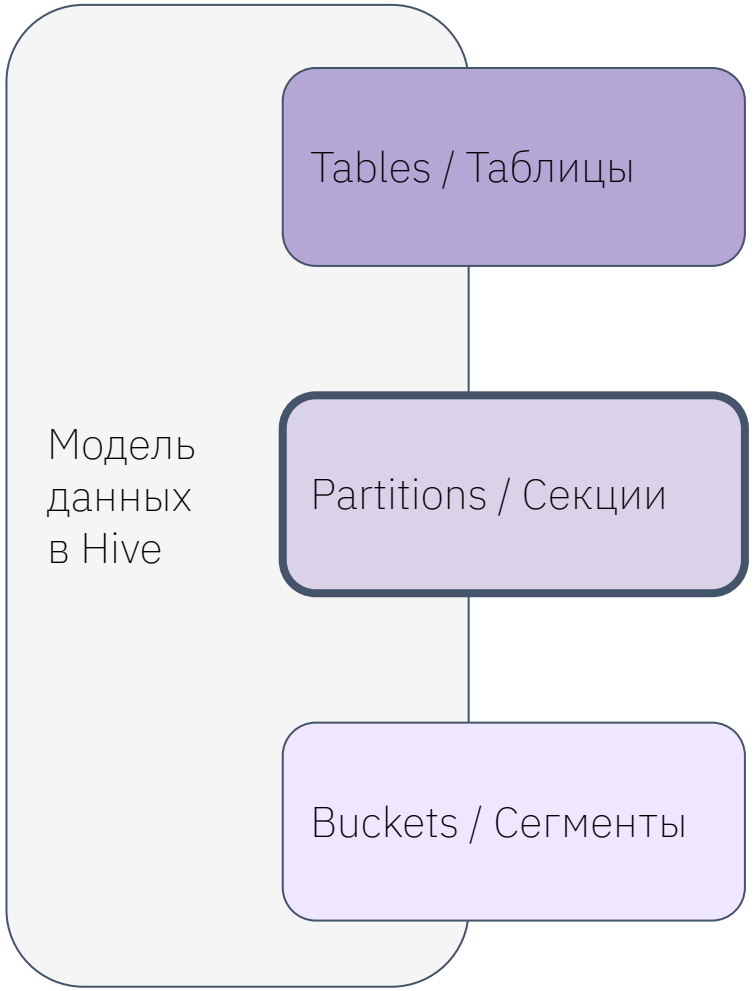
МОДЕЛЬ ДАННЫХ



Таблицы создаются таким же образом, как в классических реляционных базах данных. Являются физическим представлением данных, хранящихся в директории внутри файловой системы:

```
/user/hive/warehouse/mytable
```

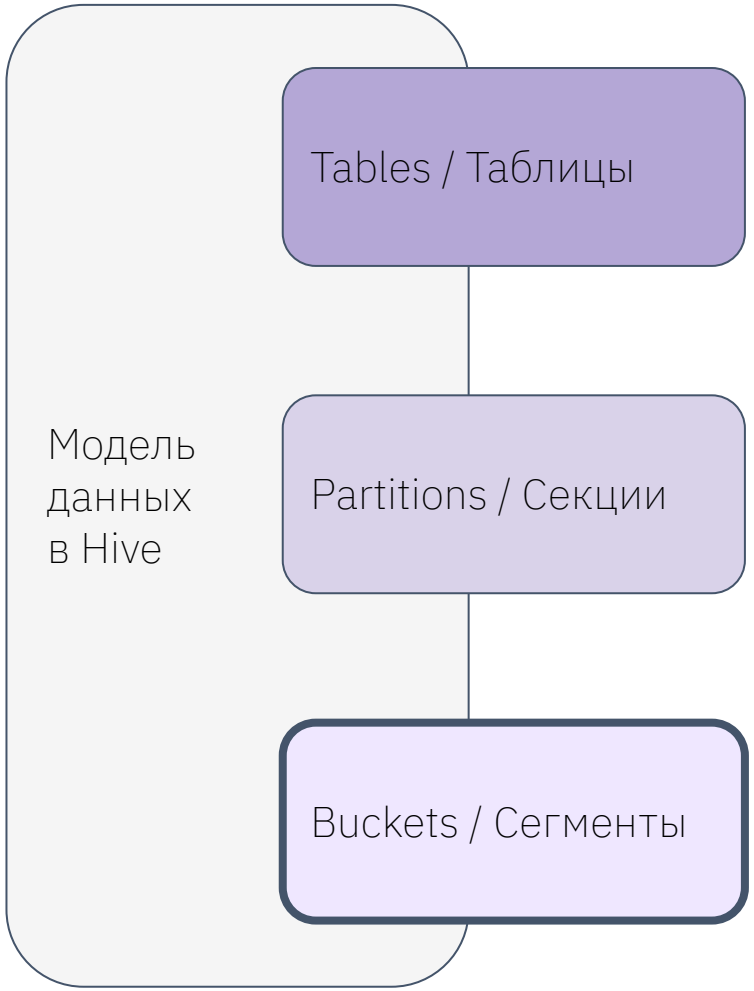
МОДЕЛЬ ДАННЫХ



Данные в таблицах разбитые по ключу называются партициями. Ключом является значение одной из колонок:

```
/user/hive/warehouse/mytable/City=Moscow
/user/hive/warehouse/mytable/City=Tambov
```

МОДЕЛЬ ДАННЫХ



Партиционированные данные могут быть также разбиты на бакеты. Оптимальным размером бакета является размер блока в HDFS:

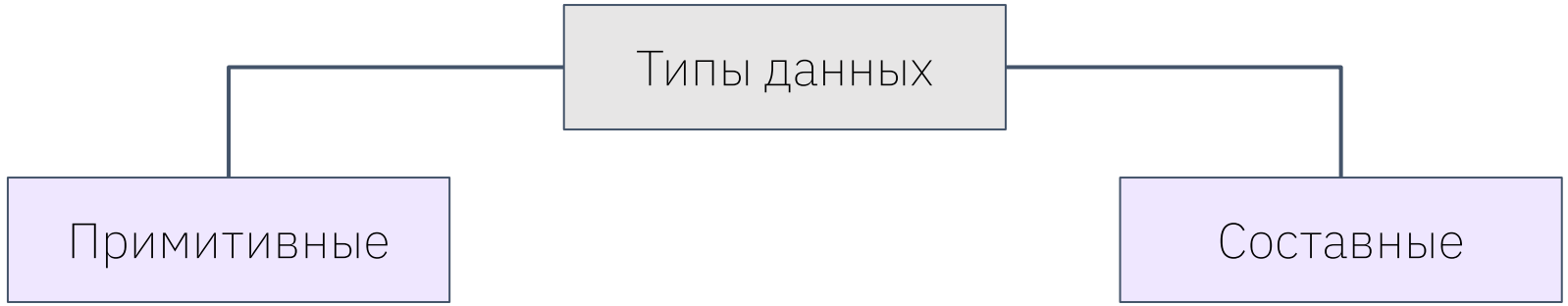
```
/user/hive/warehouse/mytable/City=Moscow/part-00000
/user/hive/warehouse/mytable/City=Moscow/part-00001
                        ...
/user/hive/warehouse/mytable/City=Tambov/part-00000
/user/hive/warehouse/mytable/City=Tambov/part-00002
/user/hive/warehouse/mytable/City=Tambov/part-00003
                        ...
```

ПЕРЕРЫВ

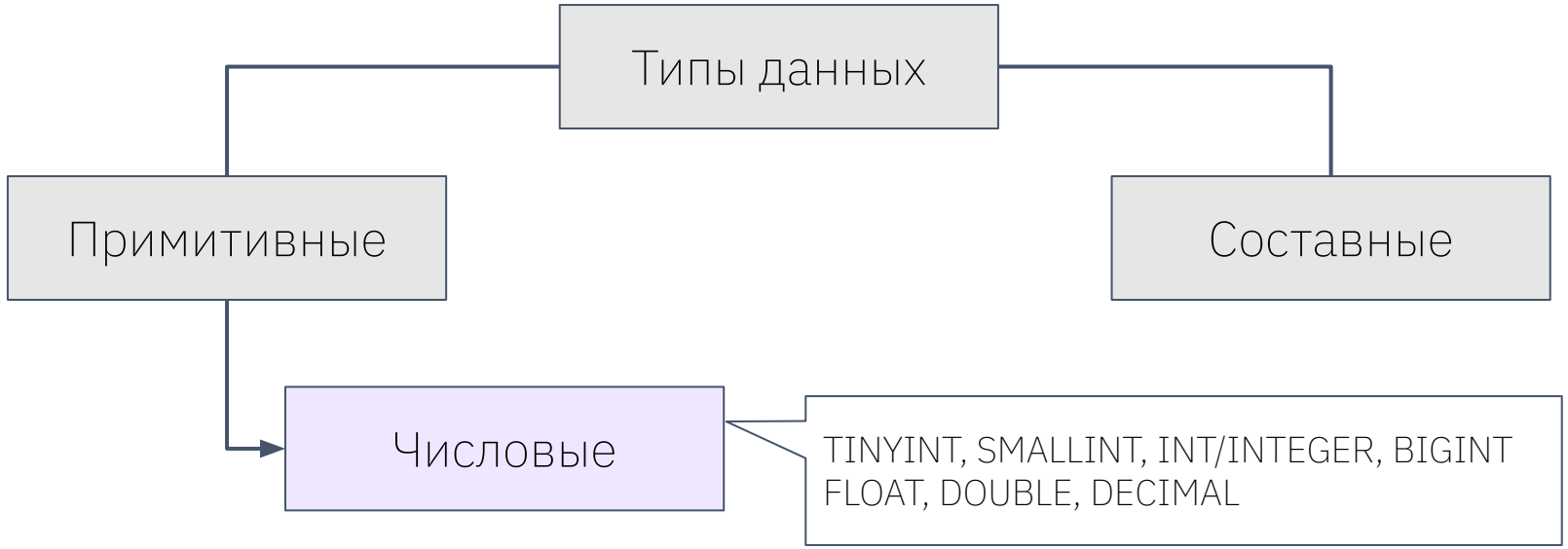

10:00

ТИПЫ ДАННЫХ

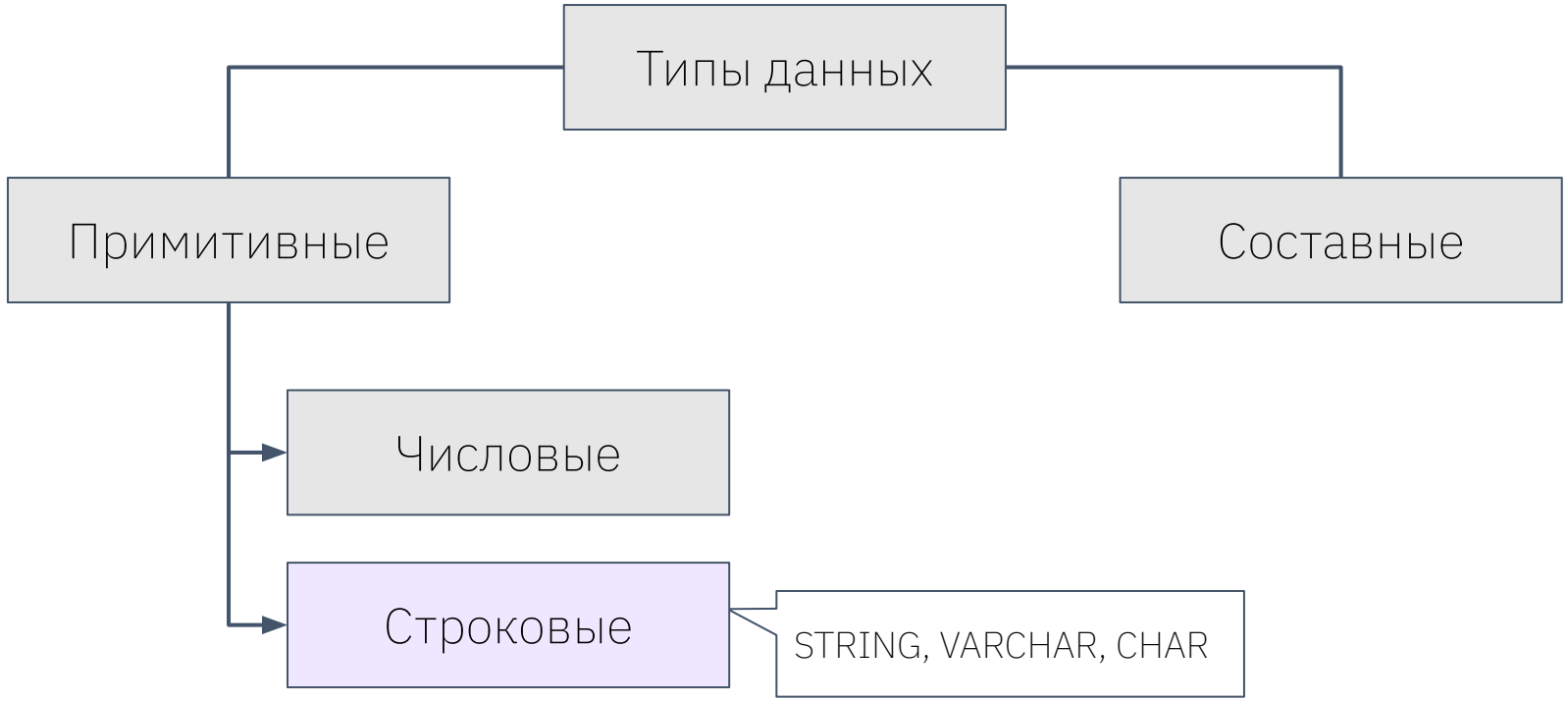
ТИПЫ ДАННЫХ



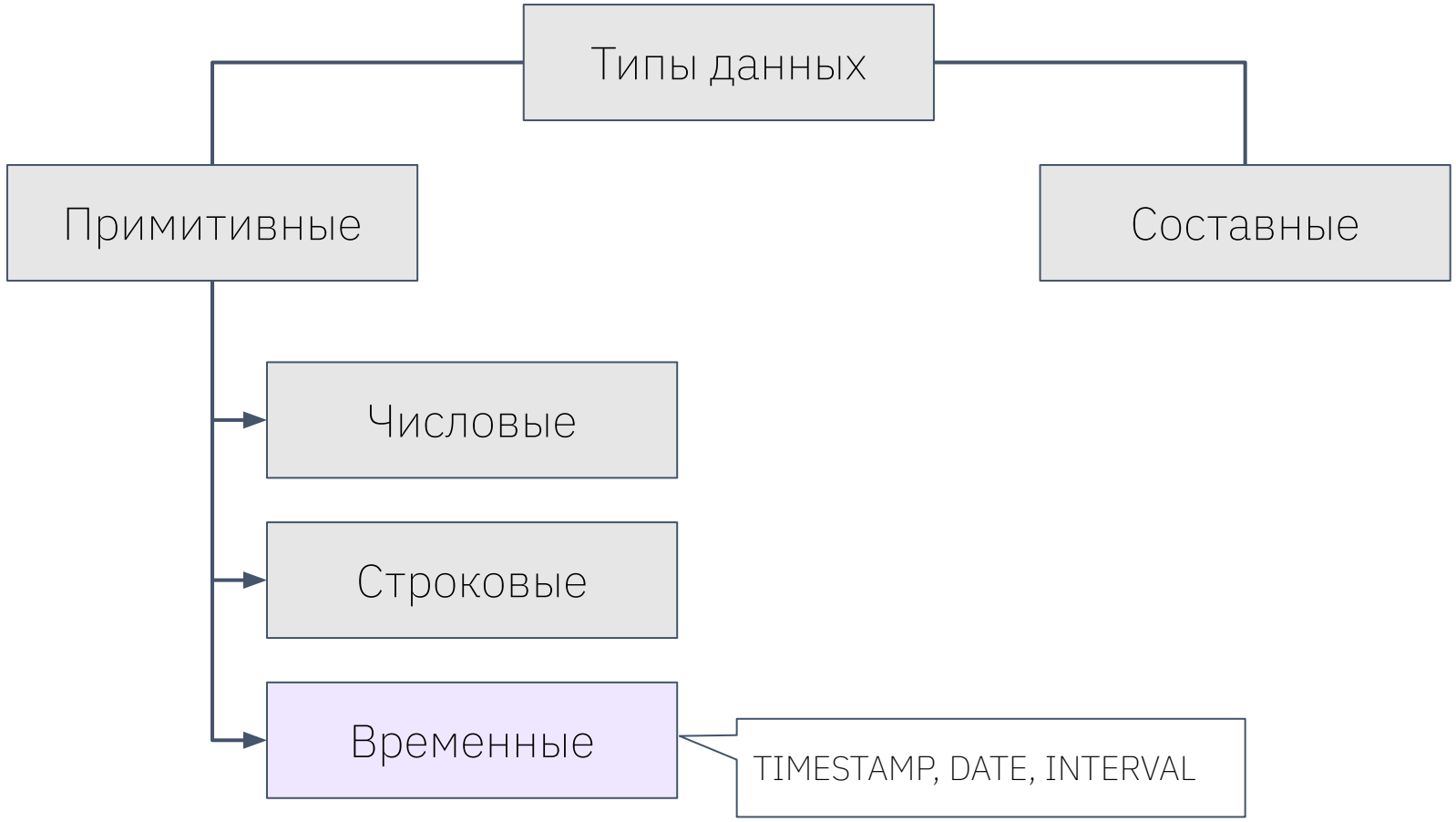
ТИПЫ ДАННЫХ


Типы данных
Примитивные
Составные
Числовые
TINYINT, SMALLINT, INT/INTEGER, BIGINT
FLOAT, DOUBLE, DECIMAL

ТИПЫ ДАННЫХ


Типы данных
Примитивные
Составные
Числовые
Строковые
STRING, VARCHAR, CHAR

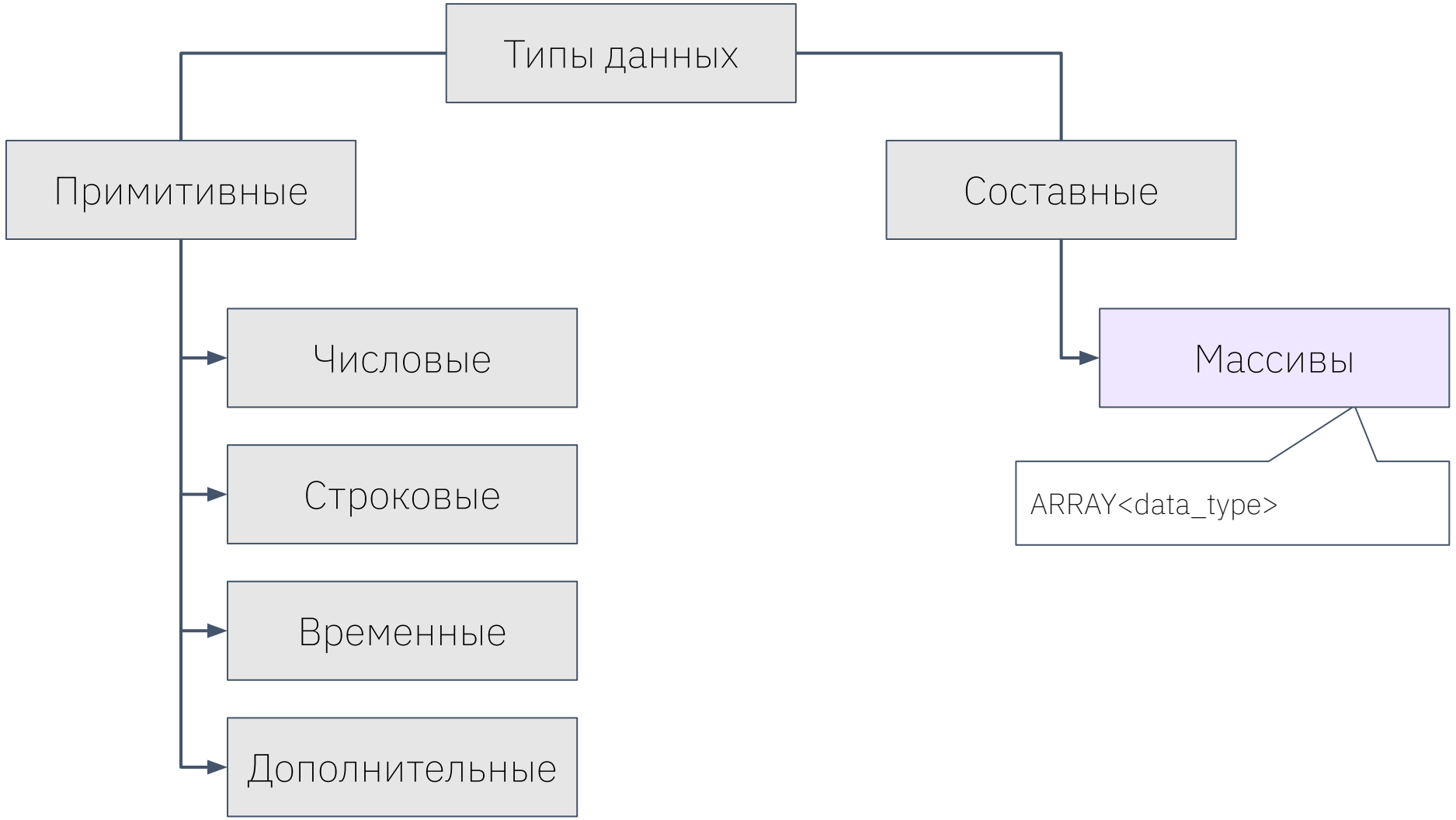
ТИПЫ ДАННЫХ


Типы данных
Примитивные
Составные
Числовые
Строковые
Временные
TIMESTAMP, DATE, INTERVAL

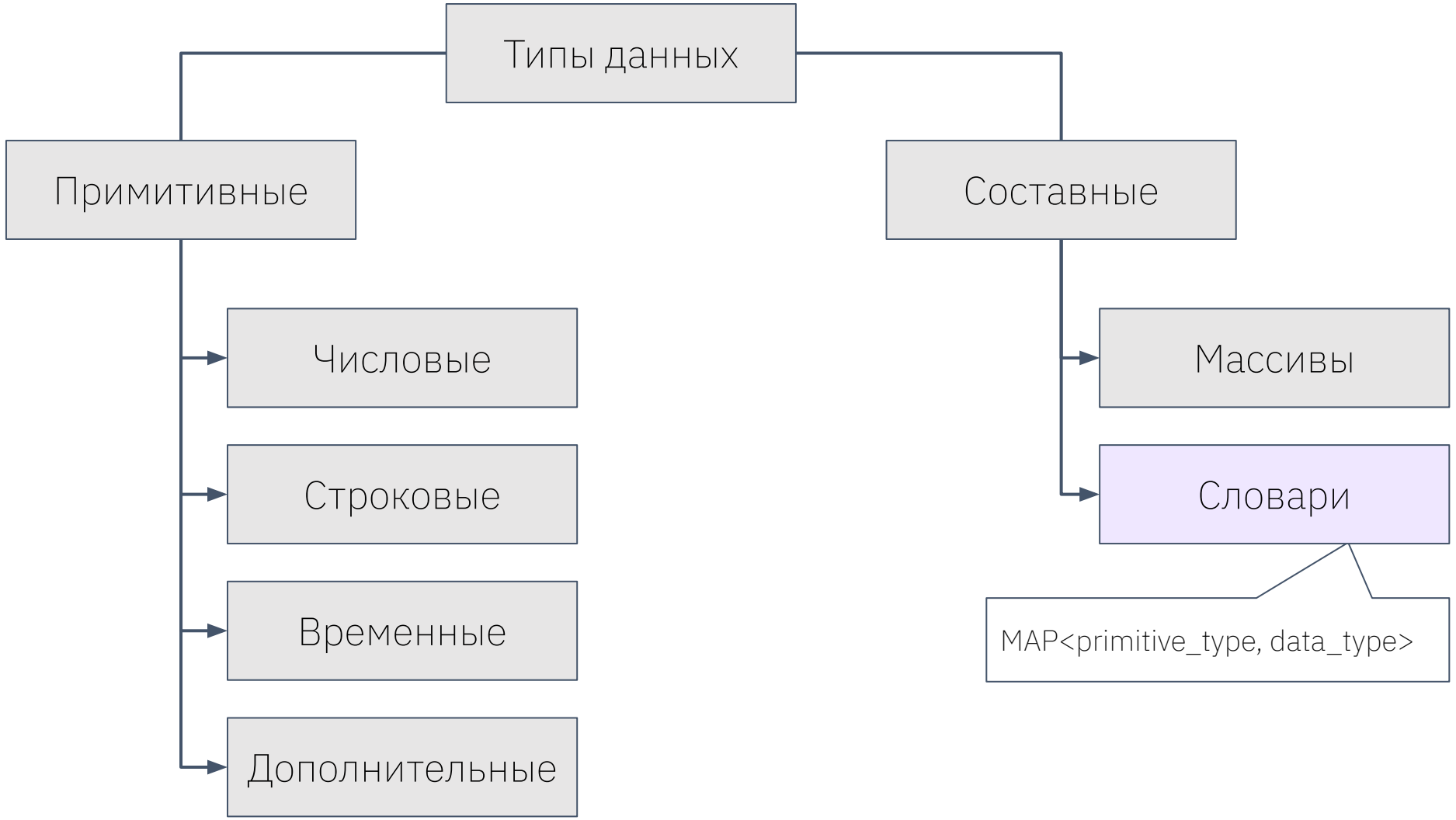
ТИПЫ ДАННЫХ


Типы данных
Примитивные
Составные
Числовые
Строковые
Временные
Дополнительные
BOOLEAN, BINARY

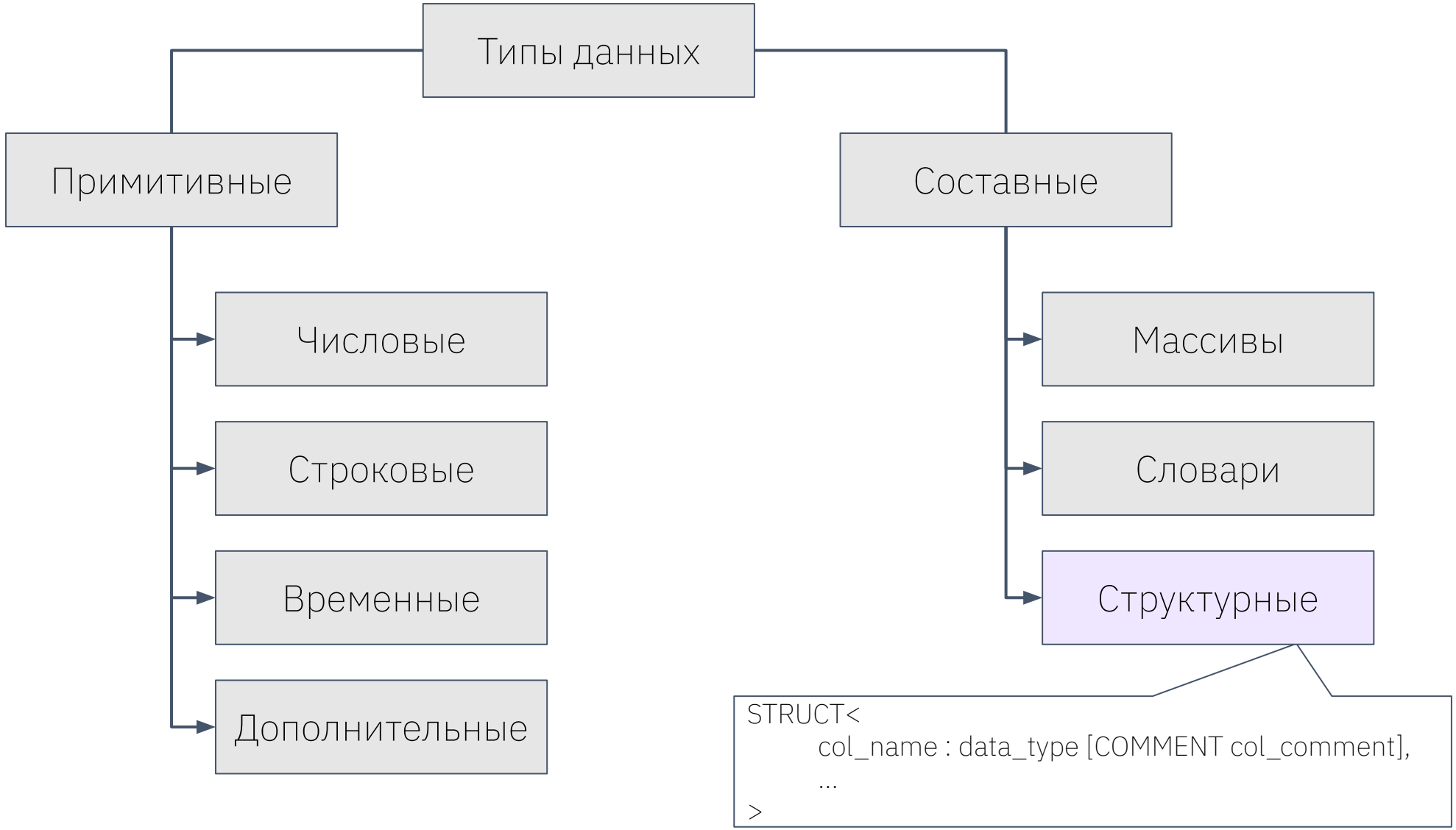
ТИПЫ ДАННЫХ


Типы данных
Примитивные
Составные
Числовые
Строковые
Временные
Дополнительные
Массивы
ARRAY<data_type>

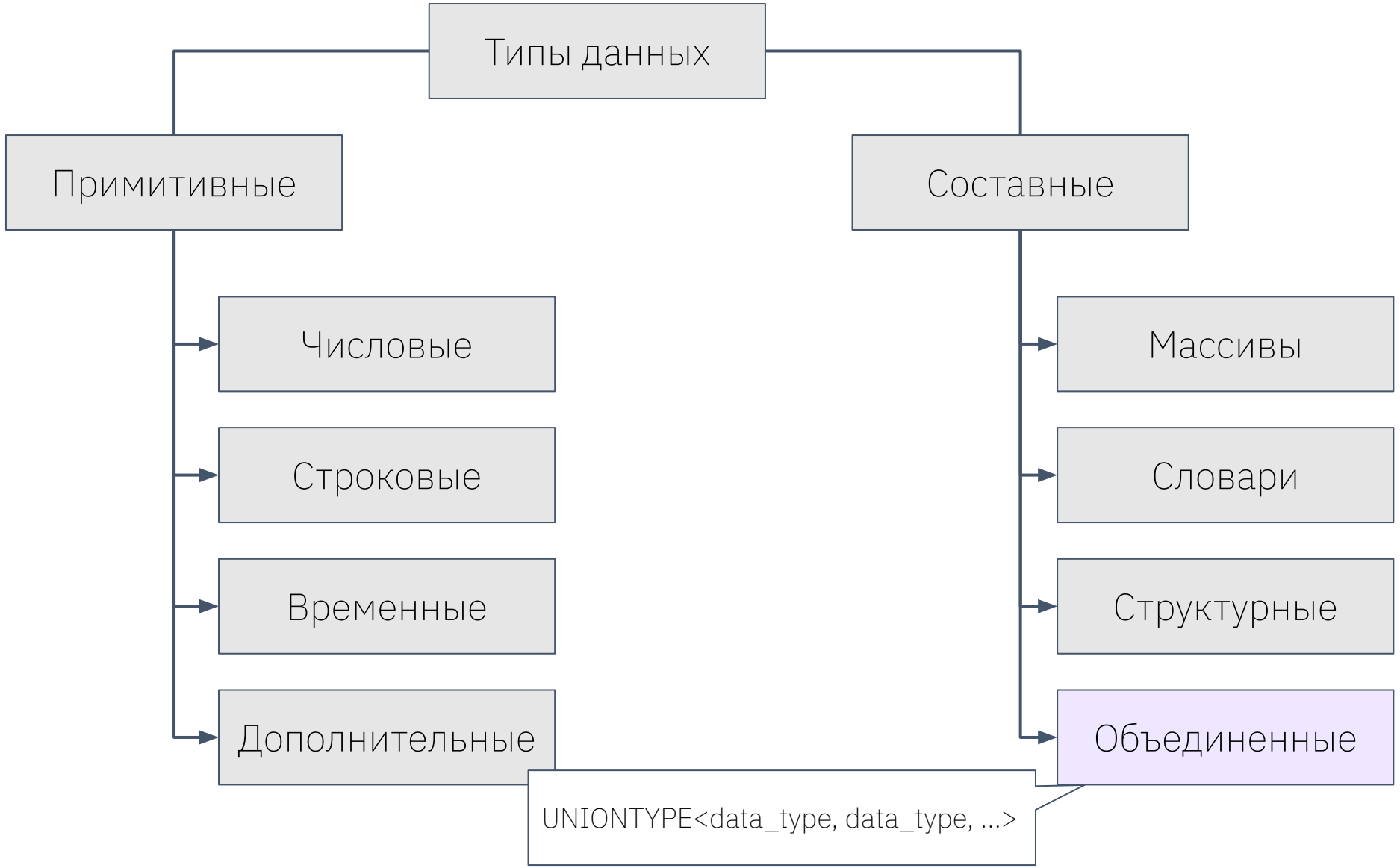
ТИПЫ ДАННЫХ


Типы данных
Примитивные
Составные
Числовые
Строковые
Временные
Дополнительные
Массивы
Словари
MAP<primitive_type, data_type>

ТИПЫ ДАННЫХ


Типы данных
Примитивные
Составные
Числовые
Строковые
Временные
Дополнительные
Массивы
Словари
Структурные
STRUCT<
col_name : data_type [COMMENT col_comment],
...
>

ТИПЫ ДАННЫХ


Типы данных
Примитивные
Составные
Числовые
Строковые
Временные
Дополнительные
Массивы
Словари
Структурные
Объединенные
UNIONTYPE<data_type, data_type, ...>

HQL

DDL

```
CREATE TABLE mytable(id INT, name STRING, age INT,  # объявляем схему
city STRING)
COMMENT 'This is a sample table'                    # комментарий для читаемости
PARTITIONED BY (city STRING)                        # партиционирование
ROW FORMAT DELIMITED                                # строки разделяются '\n'
FIELDS TERMINATED BY ','                            # поля разделяются запятой
STORED AS TEXTFILE;                                 # хранится в виде текстового файла
```

Таблица создается в специальной директории warehouse и полностью находится под управлением Hive

DDL

```
CREATE EXTERNAL TABLE my_external_table(id INT, name STRING, age INT,

city STRING)

LOCATION '/user/ivanov/mytable';
```

Таблица не создается в warehouse. У Hive сохраняется только ссылка на неё

DDL

```
DROP TABLE mytable;
```

Так как таблица находится под управлением Hive, то произойдёт полное удаление данных вместе с метадатой

```
DROP TABLE my_external_table;
```

Так как внешняя таблица **не** находится под управлением Hive, то произойдёт удаление только метадаты внутри Hive. Данные останутся нетронутыми

DML

```
LOAD DATA LOCAL INPATH '/home/ivanov/peoples.txt'

INTO TABLE mytable;
```

Файл peoples.txt находится в локальной файловой системе и будет скопирован в директорию warehouse

```
LOAD DATA INPATH '/user/ivanov/peoples.txt'

INTO TABLE mytable;
```

Файл peoples.txt находится в HDFS и будет скопирован в директорию warehouse

DML

```
INSERT OVERWRITE TABLE newtable

SELECT * FROM mytable;
```

Все данные из таблицы *mytable* будут скопированы в **существующую** таблицу *newtable*

```
CREATE TABLE newtable

AS SELECT * FROM mytable;
```

Будет **создана** новая таблица *newtable* с такой же схемой и данными как в таблице *mytable*

ЗАПРОСЫ

```
SHOW TABLES;
```

Отобразить список всех таблиц в хранилище

ЗАПРОСЫ

```
SELECT * FROM mytable;
```

Показать все данные таблицы

```
SELECT COUNT(DISTINCT city) FROM mytable;
```

Агрегация

```
SELECT COUNT(*) FROM mytable GROUP BY city;

SELECT * FROM mytable SORT BY id DESC;

FROM mytable SELECT * ORDER BY id ASC;
```

Агрегация и сортировка

ЗАПРОСЫ

```
SELECT p.*, o.*

FROM mytable p

JOIN orders p

ON (p.id = o.id);
```

Поддерживается inner, left, right, full outer виды слияний

Можно использовать множественное слияние

ФУНКЦИИ

Содержит большое количество встроенных функций: математические, статистические, строковые, даты/времени, условные, аггрегатные, преобразования типов, для работы с JSON, XML, составными типами данных. Полный список UDF функций можно посмотреть в официальной wiki или используя команды:

```
hive> SHOW FUNCTIONS;
hive> DESCRIBE FUNCTION lenght;
length (str | binary) - Returns the length of str or number of bytes in binary data
```

ФУНКЦИИ

Типы функций:

- UDF (User Defined Function) — применяются построчно
- UDAF (User Defined Aggregate Function) — используются совместно с GROUP BY оператором
- UDTF (User Defined Table Function) — применяются на таблицу целиком

Стандартную библиотеку функций можно расширить своими, добавив в classpath Hive

HIVE VS RDBMS

HIVE vs RDBMS

Apache Hive

- Проверяет схему данных при **чтении**, что ускоряет *добавление* данных и позволяет работать с *неизвестной* схемой данных

RDBMS

- Проверяет схему при **записи**, что ускоряет *чтение* данных и повышает уровень их *консистентности*

HIVE vs RDBMS

Apache Hive

- Проверяет схему данных при **чтении**, что ускоряет *добавление* данных и позволяет работать с *неизвестной* схемой данных
- Размер данных измеряется петабайтами

RDBMS

- Проверяет схему при **записи**, что ускоряет *чтение* данных и повышает уровень их *консистентности*
- Размер данных измеряется терабайтами

HIVE vs RDBMS

Apache Hive

- Проверяет схему данных при **чтении**, что ускоряет *добавление* данных и позволяет работать с *неизвестной* схемой данных
- Размер данных измеряется петабайтами
- Ориентирован на модель “один раз записал, много раз прочитал”

RDBMS

- Проверяет схему при **записи**, что ускоряет *чтение* данных и повышает уровень их *консистентности*
- Размер данных измеряется терабайтами
- Поддерживается инструментарий как записи данных, так и чтения

HIVE vs RDBMS

Apache Hive

- Проверяет схему данных при **чтении**, что ускоряет *добавление* данных и позволяет работать с *неизвестной* схемой данных
- Размер данных измеряется петабайтами
- Ориентирован на модель “один раз записал, много раз прочитал”
- Несмотря на поддержку SQL является хранилищем данных

RDBMS

- Проверяет схему при **записи**, что ускоряет *чтение* данных и повышает уровень их *консистентности*
- Размер данных измеряется терабайтами
- Поддерживается инструментарий как записи данных, так и чтения
- Традиционная база данных, основанная на реляционной модели данных

HIVE vs RDBMS

Apache Hive

- Проверяет схему данных при **чтении**, что ускоряет *добавление* данных и позволяет работать с *неизвестной* схемой данных
- Размер данных измеряется петабайтами
- Ориентирован на модель “один раз записал, много раз прочитал”
- Несмотря на поддержку SQL является хранилищем данных
- Легко масштабируется при расширении кластера

RDBMS

- Проверяет схему при **записи**, что ускоряет *чтение* данных и повышает уровень их *консистентности*
- Размер данных измеряется терабайтами
- Поддерживается инструментарий как записи данных, так и чтения
- Традиционная база данных, основанная на реляционной модели данных
- Для масштабирования требуются соответствующие навыки

ОСОБЕННОСТИ

ОСОБЕННОСТИ

Использование HiveQL позволяет специалистам, знающим SQL, сразу начать работу с данными на кластере

ОСОБЕННОСТИ

Использование HiveQL позволяет специалистам, знающим SQL, сразу начать работу с данными на кластере

Наличие таких абстракций, как “таблицы”, приближает пользовательский опыт к опыту работы с традиционными реляционными базами данных

ОСОБЕННОСТИ

Использование HiveQL позволяет специалистам, знающим SQL, сразу начать работу с данными на кластере

Наличие таких абстракций, как “таблицы”, приближает пользовательский опыт к опыту работы с традиционными реляционными базами данных

RPC, JDBC, ODBC интерфейсы позволяют разрабатывать приложения удобным для пользователей способом

ОСОБЕННОСТИ

Использование HiveQL позволяет специалистам, знающим SQL, сразу начать работу с данными на кластере

Наличие таких абстракций, как “таблицы”, приближает пользовательский опыт к опыту работы с традиционными реляционными базами данных

RPC, JDBC, ODBC интерфейсы позволяют разрабатывать приложения удобным для пользователей способом

Статическая типизация и богатый набор встроенных функций упрощает работу с данными и уменьшает вероятность появления ошибок

ПРАКТИКА

ПОДНИМАЕМ ЛОКАЛЬНЫЙ КЛАСТЕР

```
docker start -i gbhdp
```

ПОДГОТОВКА К УСТАНОВКЕ

Предварительно создаем служебные директории в HDFS:

```
$ hdfs dfs -mkdir -p /user/hive/warehouse
$ hdfs dfs -chmod a+w /user/hive/warehouse
```

УСТАНОВКА

Скачиваем и распаковываем дистрибутив:

```
$ wget https://apache-mirror.rbc.ru/pub/apache/hive/hive-2.3.9/apache-hive-2.3.9-bin.tar.gz
$ tar xzf apache-hive-2.3.9-bin.tar.gz
$ rm apache-hive-2.3.9-bin.tar.gz
$ mv apache-hive-2.3.9-bin hive
```

Задаем необходимые переменные окружения:

```
$ cd hive
$ export HIVE_HOME=`pwd`
$ export PATH=$PATH:$HIVE_HOME/bin
```

Инициализируем метастор:

```
$ schematool -dbType derby -initSchema
```

Проверяем работу:

```
$ hive -e 'show tables;'
OK
Time taken: 6.128 seconds
```

НАСТРОЙКА HIVE SERVER

Создадим файл ~/hive/conf/hive-site.xml и вставим:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
  <property>
    <name>hive.server2.enable.doAs</name>
    <value>FALSE</value>
    <description/>
  </property>
</configuration>
```

ЗАПУСК HIVE SERVER

Запускаем в фоне Hive Server:

```
$ hiveserver2 &> /dev/null &
```

Подключаемся через beeline cli:

```
$ beeline -u jdbc:hive2://localhost:10000
```

Проверяем работу:

```
0: jdbc:hive2://localhost:10000> show tables;
OK
+-----------+
| tab_name  |
+-----------+
+-----------+
No rows selected (0.404 seconds)
```

Выходим:

```
0: jdbc:hive2://localhost:10000> !q
Closing: 0: jdbc:hive2://localhost:10000
```

ИНТЕРАКТИВНАЯ ОБОЛОЧКА

Работа в Hive Shell сводится к исполнению команд на HiveQL

При передаче ключа *-f* можно исполнить заранее написанный скрипт:

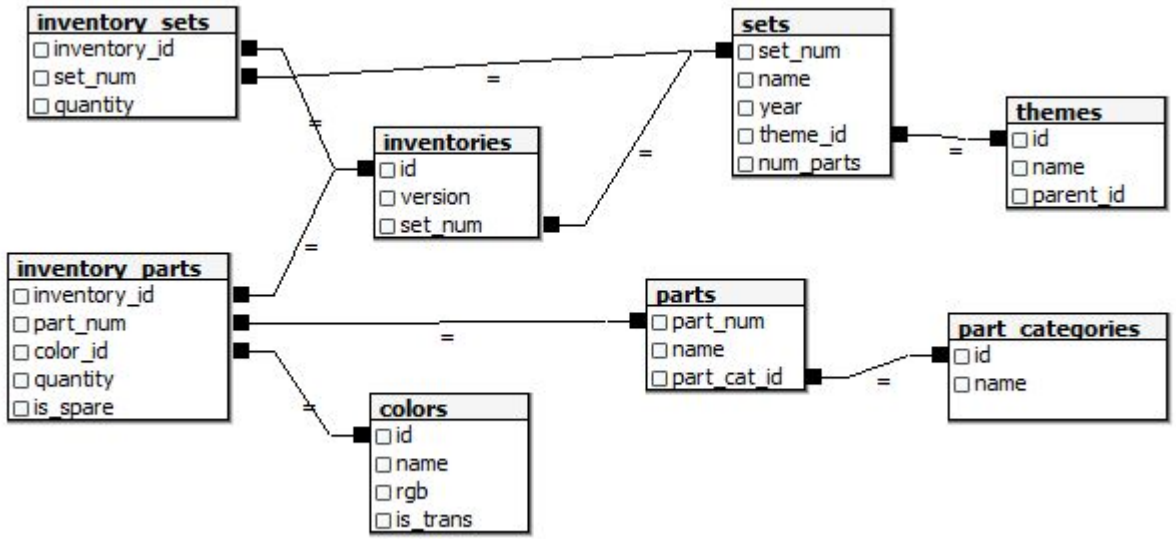
```
hive -f script.q
```

Либо через ключ *-e* выполнить произвольный запрос:

```
hive -e 'SELECT * FROM mytable'
```

ПРИМЕР РАБОТЫ

Поработаем с датасетом LEGO Database:



Загрузим данные themes и sets в Hive таблицы и найдем набор с самым большим количеством деталей

ПРИМЕР РАБОТЫ

Загружаем данные в контейнер:

```
$ docker cp sets.csv gbhdp:/home/hduser/

$ docker cp themes.csv gbhdp:/home/hduser/
```

Переключаемся на терминал с хадуп сессией и включаем интерактивную оболочку:

```
$ hive
```

ПРИМЕР РАБОТЫ

Создаем таблицу для themes.csv:

```
CREATE TABLE lego_themes(id INT, name STRING, parent_id INT)
    COMMENT 'Information on lego themes'
    ROW FORMAT DELIMITED
    FIELDS TERMINATED BY ','
    STORED AS TEXTFILE
    TBLPROPERTIES('skip.header.line.count'='1');
```

ПРИМЕР РАБОТЫ

Создаем таблицу для sets.csv:

```
CREATE TABLE lego_sets(set_num STRING, name STRING, year INT, theme_id INT,
num_parts INT)
    COMMENT 'Information on lego sets'
    ROW FORMAT DELIMITED
    FIELDS TERMINATED BY ','
    STORED AS TEXTFILE
    TBLPROPERTIES('skip.header.line.count'='1');
```

ПРИМЕР РАБОТЫ

```
hive> CREATE TABLE lego_themes(id INT, name STRING, parent_id INT)
    > COMMENT 'Information on lego themes'
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ','
    > STORED AS TEXTFILE;
OK
Time taken: 0.18 seconds
hive> CREATE TABLE lego_sets(set_num STRING, name STRING, year INT, theme_id INT, num_parts INT)
    > COMMENT 'Information on lego sets'
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ','
    > STORED AS TEXTFILE;
OK
Time taken: 0.086 seconds
hive> SHOW TABLES;
OK
lego_sets
lego_themes
Time taken: 0.028 seconds, Fetched: 2 row(s)
```

ПРИМЕР РАБОТЫ

Загружаем данные в таблицы:

```
LOAD DATA LOCAL INPATH '/home/hduser/themes.csv' INTO TABLE lego_themes;

LOAD DATA LOCAL INPATH '/home/hduser/sets.csv' INTO TABLE lego_sets;
```

```
hive> LOAD DATA LOCAL INPATH '/home/hduser/themes.csv' INTO TABLE lego_themes;
Loading data to table default.lego_themes
OK
Time taken: 6.916 seconds
hive> LOAD DATA LOCAL INPATH '/home/hduser/sets.csv' INTO TABLE lego_sets;
Loading data to table default.lego_sets
OK
Time taken: 0.326 seconds
```

ПРИМЕР РАБОТЫ

```
hive> SELECT * FROM lego_themes LIMIT 5;
OK
NULL    name    NULL
1       Technic NULL
2       Arctic Technic  1
3       Competition     1
4       Expert Builder  1
Time taken: 0.131 seconds, Fetched: 5 row(s)
hive> SELECT * FROM lego_sets LIMIT 5;
OK
set_num name    NULL    NULL    NULL
00-1    Weetabix Castle 1970    414     471
0011-2  Town Mini-Figures       1978    84      12
0011-3  Castle 2 for 1 Bonus Offer       1987    199     2
0012-1  Space Mini-Figures      1979    143     12
Time taken: 0.137 seconds, Fetched: 5 row(s)
```

ПРИМЕР РАБОТЫ

Набор с самым большим количеством деталей:

```
SELECT name, year, num_parts

FROM lego_sets

WHERE num_parts IN (SELECT Max(num_parts) FROM lego_sets);
```

ПРИМЕР РАБОТЫ

```
hive> SELECT name, year, num_parts FROM lego_sets WHERE num_parts IN (SELECT Max(num_parts) FROM lego_sets);
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a
leases.
Query ID = hduser_20210507211902_be32c787-3eec-4d71-8299-2882b66eea68
Total jobs = 3
Launching Job 1 out of 3
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Job running in-process (local Hadoop)
2021-05-07 21:19:03,772 Stage-2 map = 100%,  reduce = 100%
Ended Job = job_local2082942877_0002
Stage-5 is selected by condition resolver.
Stage-1 is filtered out by condition resolver.
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/hduser/hive/lib/log4j-slf4j-impl-2.6.2.jar!/org/slf4j/impl/StaticLogg
SLF4J: Found binding in [jar:file:/home/hduser/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.25.jar!/org/s
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
2021-05-07 21:19:14     Starting to launch local task to process map join;      maximum memory = 477626368
2021-05-07 21:19:14     Dump the side-table for tag: 1 with group count: 1 into file: file:/tmp/hduser/b7eb2b
9210616200196-1/-local-10005/HashTable-Stage-3/MapJoin-mapfile01--.hashtable
2021-05-07 21:19:14     Uploaded 1 File to: file:/tmp/hduser/b7eb2b89-3d02-45b1-8725-dfc0f2d3c944/hive_2021-0
ge-3/MapJoin-mapfile01--.hashtable (280 bytes)
2021-05-07 21:19:14     End of local task; Time Taken: 0.463 sec.
Execution completed successfully
MapredLocal task succeeded
Launching Job 3 out of 3
Number of reduce tasks is set to 0 since there's no reduce operator
Job running in-process (local Hadoop)
2021-05-07 21:19:17,029 Stage-3 map = 100%,  reduce = 0%
Ended Job = job_local232705884_0003
MapReduce Jobs Launched:
Stage-Stage-2:  HDFS Read: 5132484 HDFS Write: 2075420 SUCCESS
Stage-Stage-3:  HDFS Read: 3073756 HDFS Write: 1037710 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
Taj Mahal       2008    5922
Time taken: 14.916 seconds, Fetched: 1 row(s)
```

ПРИМЕР РАБОТЫ

Добавим к выдаче тему набора:

```
SELECT s.name, s.year, t.name AS theme, s.num_parts
FROM lego_sets s
    JOIN lego_themes t ON (s.theme_id = t.id)
WHERE s.num_parts IN (SELECT Max(num_parts) FROM lego_sets);
```

```
OK
Taj Mahal       2008    Sculptures      5922
Time taken: 24.762 seconds, Fetched: 1 row(s)
```

ОСТАНОВКА ЛОКАЛЬНОГО КЛАСТЕРА

```
exit
```

HIVE VS PIG VS IMPALA

Pig

Hive

Impala

Pig vs Hive & Impala

```
input_lines = LOAD '/tmp/word.txt' AS (line:chararray);
words = FOREACH input_lines GENERATE FLATTEN(TOKENIZE(line)) AS word;
filtered_words = FILTER words BY word MATCHES '\\w+';
word_groups = GROUP filtered_words BY word;
word_count = FOREACH word_groups GENERATE COUNT(filtered_words) AS count, group AS word;
ordered_word_count = ORDER word_count BY count DESC;
STORE ordered_word_count INTO '/tmp/results.txt';
```

```
DROP TABLE IF EXISTS docs;
CREATE TABLE docs (line STRING);
LOAD DATA INPATH 'input_file' OVERWRITE INTO TABLE docs;
CREATE TABLE word_counts AS
SELECT word, count(1) AS count FROM
 (SELECT explode(split(line, '\s')) AS word FROM docs) temp
GROUP BY word
ORDER BY word;
```

Impala

Impala daemon (impalad)

- Запущен на каждом узле кластера.
- Читает и пишет файлы данных
- Принимает запросы на выполнение от impala-shell, Hue, JDBC, или ODBC.
- Параллелизирует работу запроса и передает результаты центральному координатору.

Impala Statestore (statestored)

- Мониторит состояние узлов кластера. Обычно один на кластер.
- Рассылает информацию о доступности узлов по кластеру.

Impala Catalog Service (catalogd)

- Следит за метаданными о объектах в кластере.
- Обновляет и рассылает обновления метаданных.

ПРАКТИЧЕСКОЕ ЗАДАНИЕ

ПРАКТИЧЕСКОЕ ЗАДАНИЕ

Приложите составленные запросы и первые 10 строк результатов:

1. Установите Apache Hive в ваш контейнер с Apache Hadoop
2. Скачайте датасет lego-database и импортируйте его в Hive
3. Составьте запрос, который выведет имя набора (sets.name) с самым большим количеством деталей (sets.num_parts)
4. Составьте запрос, который выведет в каком году (sets.year) вышло больше всего наборов
5. Составьте запрос, который выведет общее количество деталей (inventory_parts.quantity) для каждого из цветов (colors.name)
6. * Измените Dockerfile так, чтобы вместе с Hadoop устанавливался и запускался Hive

Спасибо!

Каждый день вы становитесь лучше :)