

# DWH

Лекция №5:

Hadoop. Основные компоненты. MapReduce

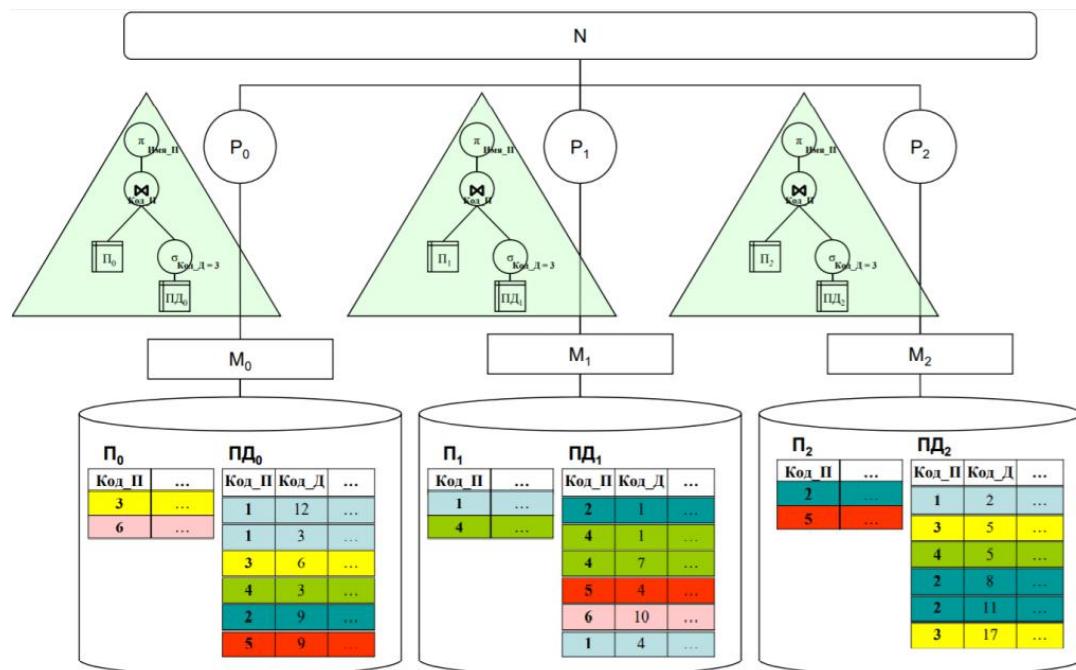


В прошлой  
лекции



# MPP системы

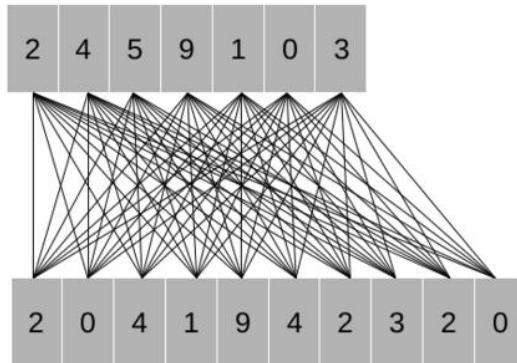
- Индексы
- Фрагментация
- Репликация
- Тиражирование запросов



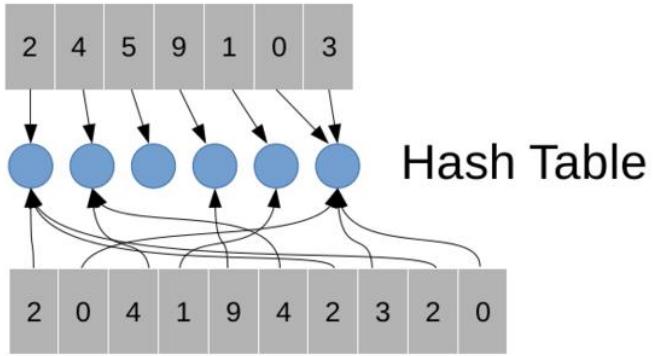
# Стратегии джойнов

- Nested Loops Join
- Hash Match Join
- Merge Join
- Broadcast join

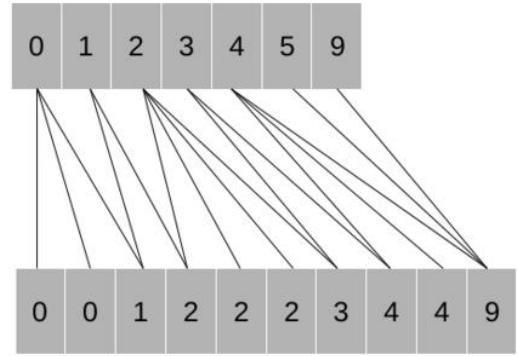
Nested Loops Join



Hash Join



Merge Join



# Введение и история



# Введение

Apache Hadoop – проект фонда Apache Software Foundation, свободно распространяемый набор утилит, библиотек и фреймворков для разработки и выполнения распределенных программ, работающих на кластерах из сотен и тысяч узлов.





# История

2003 г. – [The Google File System. Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. Google\\*](#).

2004 г. – [MapReduce: Simplified Data Processing on Large Clusters. Jeffrey Dean and Sanjay Ghemawat](#)

2005 г. – Дуг Каттинг и Майк Кафарелла, вдохновленные публикациями Google, начинают разработку Hadoop, которая первоначально состояла из двух модулей HDFS и MapReduce

2006 г. – Hadoop 0.1.0. В качестве эксперимента отсортировали 1.8 ТВ данных за 48 часов на 188 машинах.

...

2023 г. – Вышла версия Hadoop 3.3.6. На данный момент существует более 150 продуктов и систем связанных с Hadoop

\*Hadoop – имя игрушечного слонёнка, который принадлежал ребенку Каттинга

# Экосистема Hadoop

Основные компоненты

cloudera®

APACHE  Spark™

TEZ 

  
HORTONWORKS®

APACHE  HBASE



 ARENADATA

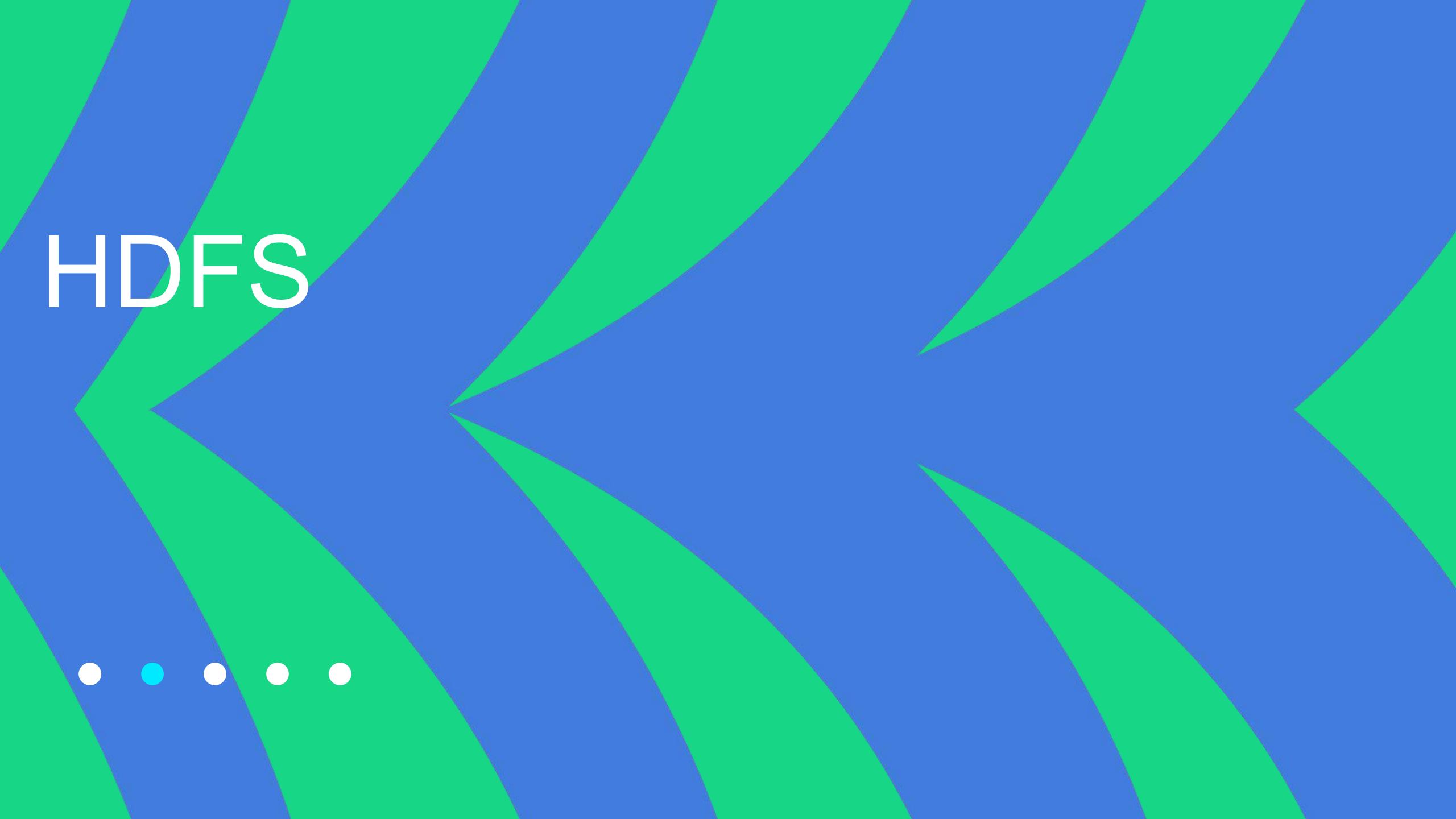






Apache  ORC™  
 Parquet





# HDFS

... . . . .

# HDFS

HDFS (Hadoop Distributed File System) – распределенная файловая система, предназначенная для хранения файлов больших размеров, поблочно распределенных между узлами вычислительного кластера.

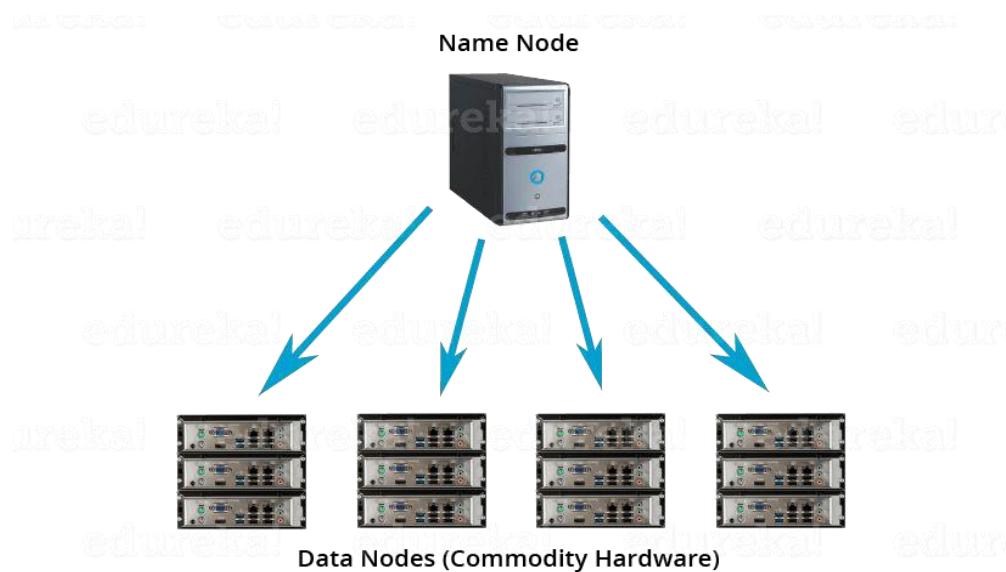
Особенности:

- Не любит маленькие файлы;
- Горизонтальная масштабируемость (относительно недорогая);
- Параллельная запись и чтение данных;
- Изменение файлов не предусмотрено, только добавление и перезапись;
- Надежность (предусмотрены механизмы репликации и устойчивости к отказам оборудования)

# Архитектура HDFS

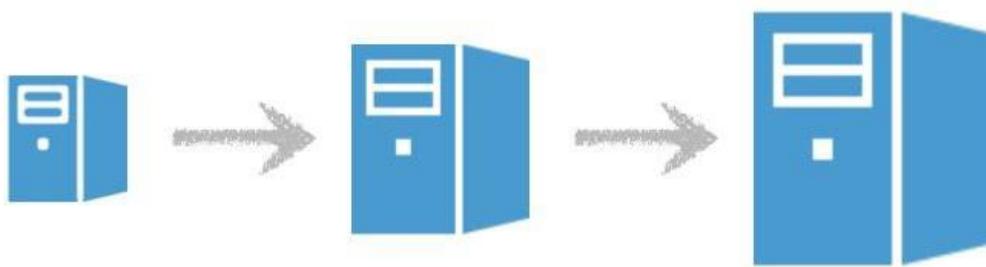
NameNode – сервер, управляющий пространством имен файловой системы (знает где и что лежит) и доступом клиентов к данным.

DataNodes – сервера, которые непосредственно хранят данные.



# HDFS. Масштабируемость

Относительно недорогая горизонтальная масштабируемость.



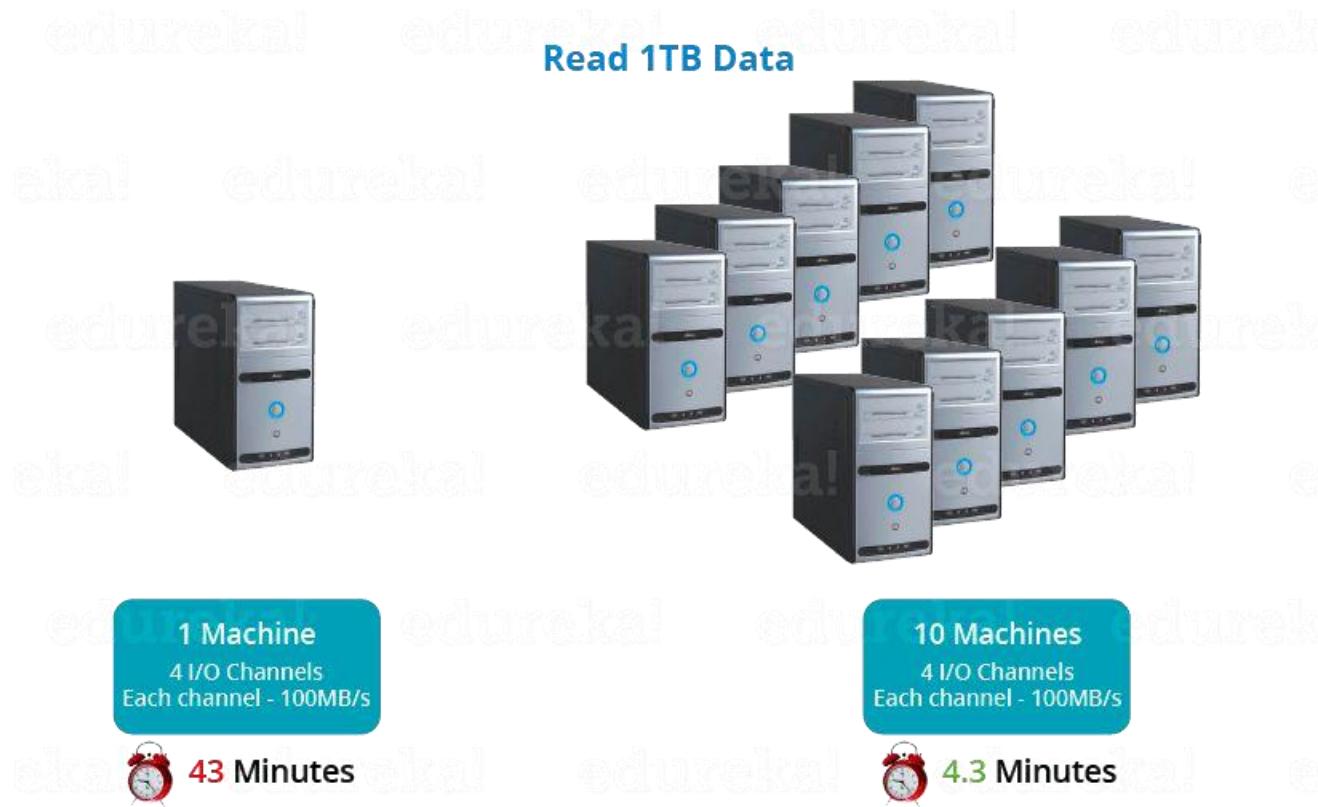
Scale Up



Scale Out

# HDFS. Параллельная запись и чтение данных

Запись и чтение данных производится во множестве потоков, что существенно ускоряет обработку



# HDFS. CLI-клиент

Команды для работы с файлами в hdfs очень схожи с командами в Linux:

- hdfs dfs -ls /path/to/data
- hdfs dfs -cat (-text) /path/to/data/file.txt
- hdfs dfs -rm -r -f /data/my\_data/2017/
- hdfs dfs -mkdir -p /path/to/new dirname
- hdfs dfs -copyToLocal (-get) /hdfs\_path /local\_path
- hdfs dfs -copyFromLocal (-put) /local\_path /hdfs\_path
- hdfs dfs -help

С полным списком можно ознакомиться по ссылке [FileSystemShell](#)

# HDFS. CLI-клиент. Вайлдкарды.

Следующая команда копирует все логи за 5 августа:

```
hdfs dfs -copyToLocal /data/logs/20180805_*.log  
/local_data/logs
```

Pattern	Description
?	Matches any single character
*	Matches zero or more characters
[abc]	Matches a single character from character set {a,b,c}.
[a-b]	Matches a single character from the character range {a...b}. Note that the “^” character must occur immediately to the right of the opening bracket.
[^a]	Matches a single character that is not from character set or range {a}. Note that the “^” character must occur immediately to the right of the opening bracket.
\c	Removes (escapes) any special meaning of character c.
{ab,cd}	Matches a string from the string set {ab, cd}.
{ab,c{de, fh}}	Matches a string from the string set {ab, cde, cfh}.

# HDFS. CLI-клиент

```
[v.alehin@ADH-cluster420359d37dd5-Master1-0 ~]$ hdfs dfs -text ~/example.txt.gz
1,foo
2,bar
3,baz
4,hello
5,world
[v.alehin@ADH-cluster420359d37dd5-Master1-0 ~]$ hdfs dfs -cat ~/example.txt.gz
♦
v♦eexample.txt3♦I♦♦♦2♦IJ,♦2♦U\&:♦99♦\♦:♦E9)\♦" [v.alehin@ADH-cluster420359d37dd5-Master1-0 ~]$
```

# HDFS. Блоки

При записи на hdfs файл делится на блоки. По умолчанию размер блока 128 МВ, но можно задать другой размер.

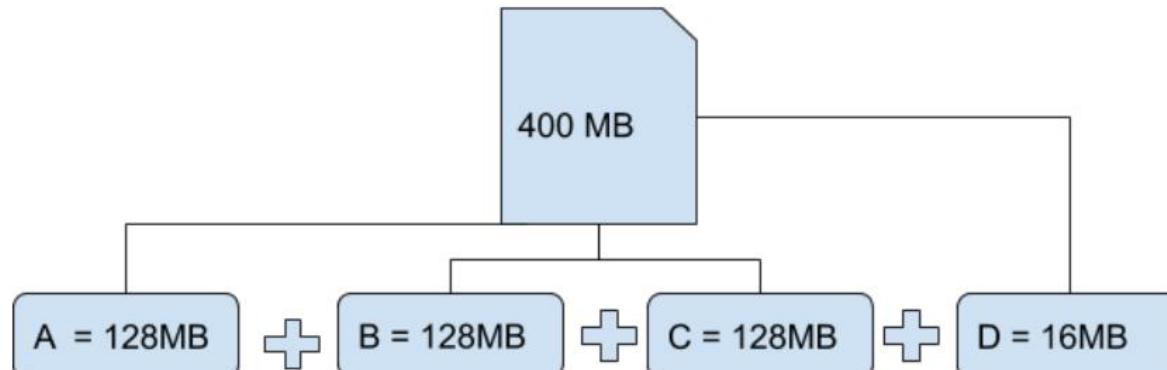
Установленный по умолчанию размер блока hdfs можно узнать командой:

```
hdfs getconf -confKey dfs.blocksize
```

Также при записи можно указать желаемый размер блока:

```
hdfs dfs -copyFromLocal -Ddfs.blocksize=134217728 ./example.txt
```

```
/tmp/test_data/
```



# HDFS. Репликация

HDFS обеспечивает надежный способ хранения данных. Копии блоков (реплики) хранятся на нескольких серверах. По умолчанию фактор репликации равен 3 (т.е. 3 копии).

Каждому файлу можно задать свой фактор репликации:

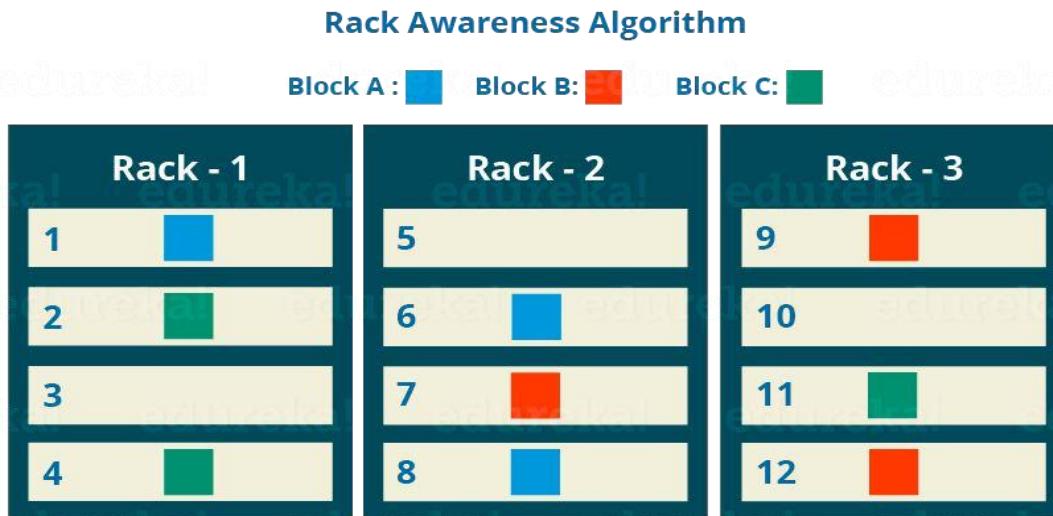
```
hdfs dfs -setrep 3 /tmp/test_data/example.txt
```

```
[v.alehin@ADH-cluster420359d37dd5-Master1-0 ~]$ hdfs dfs -ls -h ~/hadoop1
Found 3 items
-rw-r--r--  3 v.alehin hadoop      6.0 K 2024-03-19 11:27 /home/v.alehin/hadoop1/.DS_Store
drwxr-xr-x  - v.alehin hadoop          0 2024-03-19 11:27 /home/v.alehin/hadoop1/data1
drwxr-xr-x  - v.alehin hadoop          0 2024-03-19 11:27 /home/v.alehin/hadoop1/script1
[v.alehin@ADH-cluster420359d37dd5-Master1-0 ~]$ hdfs dfs -du ~/hadoop1
6148    18444  /home/v.alehin/hadoop1/.DS_Store
1992483  5977449  /home/v.alehin/hadoop1/data1
9779    29337  /home/v.alehin/hadoop1/script1
```

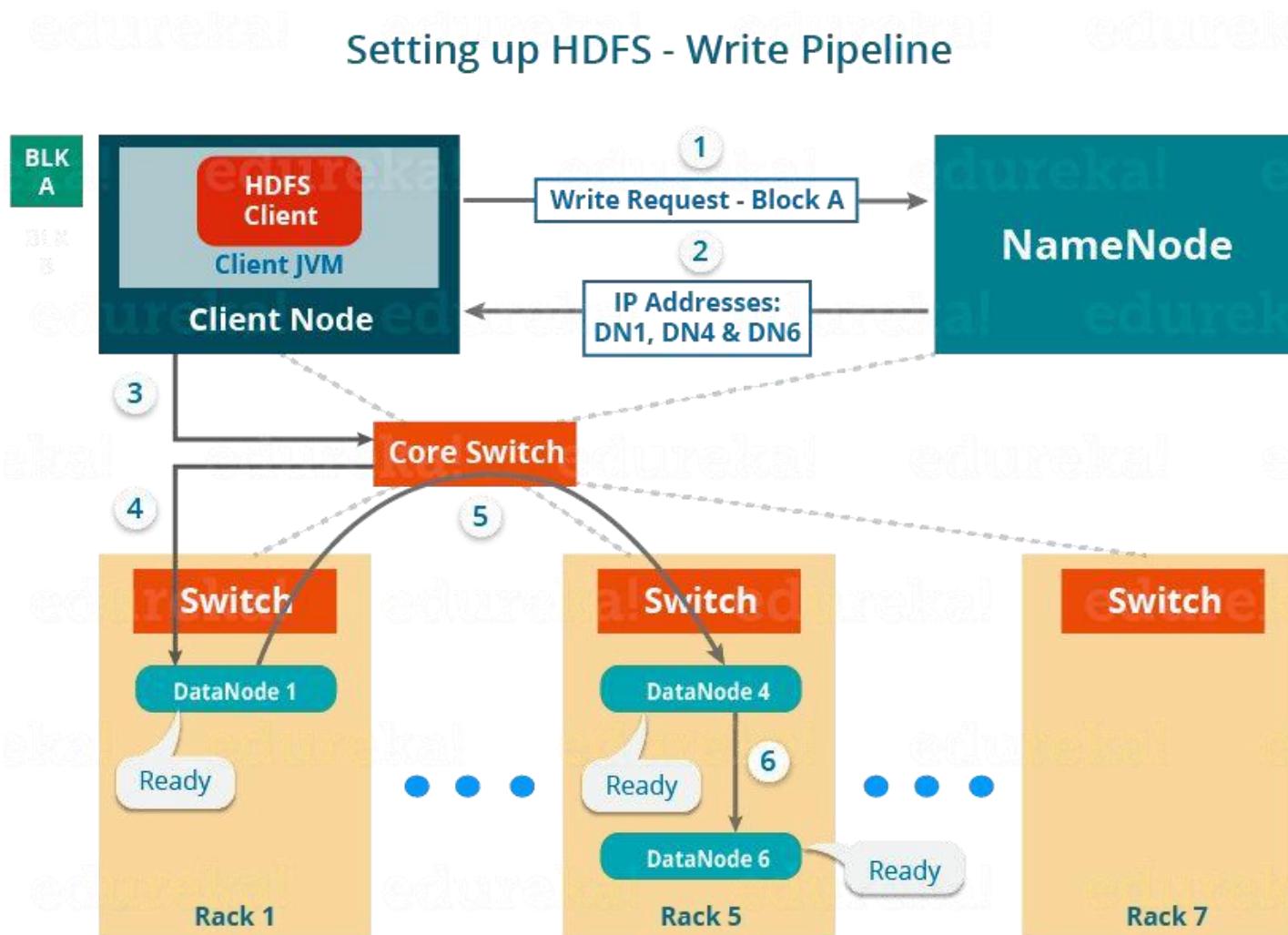
# HDFS. Репликация

Размещение блоков происходит следующим образом:

- первая реплика размещается на локальной ноде
- вторая реплика на другой ноде в этой же стойке
- третья реплика на произвольной ноде другой стойки
- остальные реплики размещаются произвольным способом

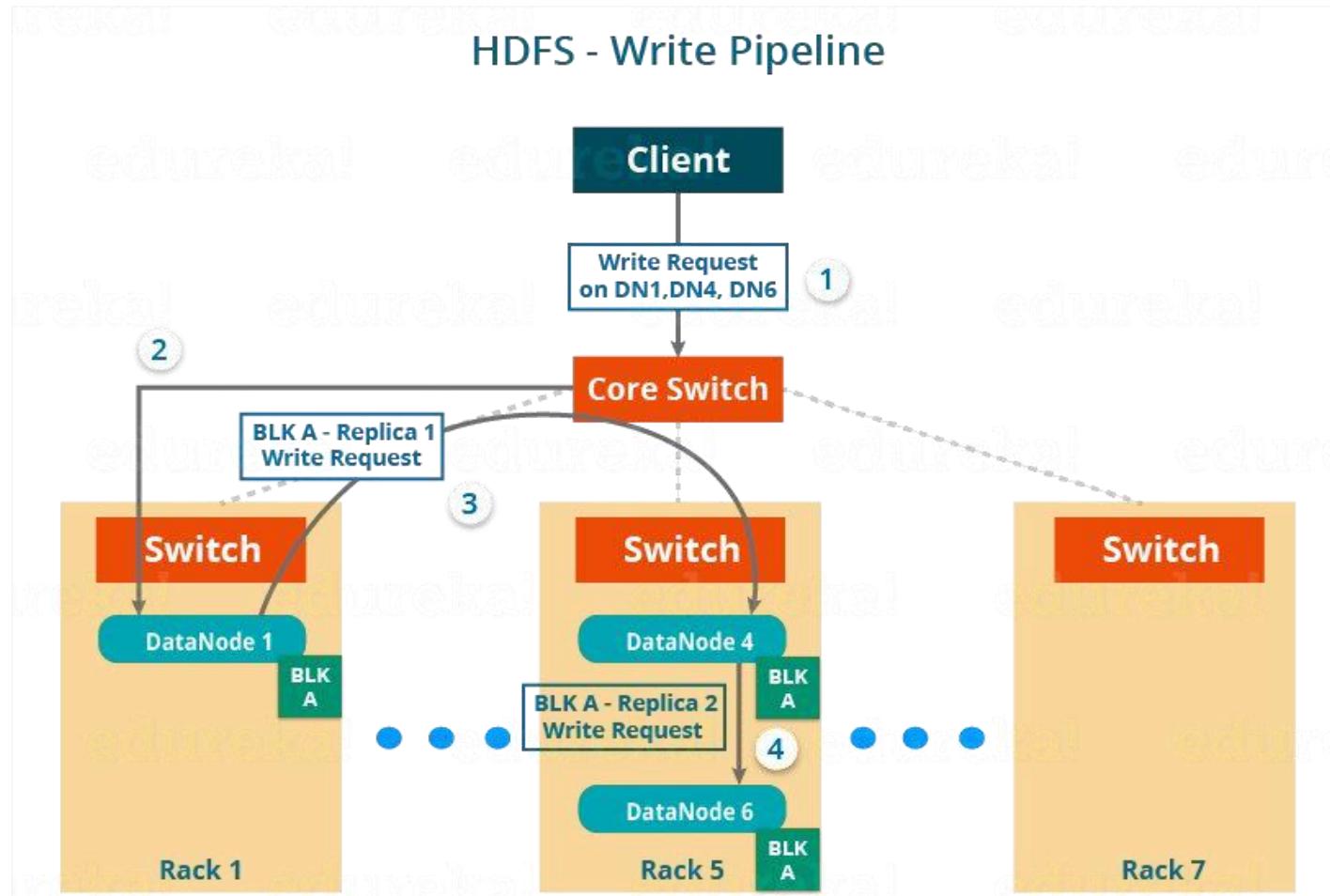


# HDFS. Запись данных



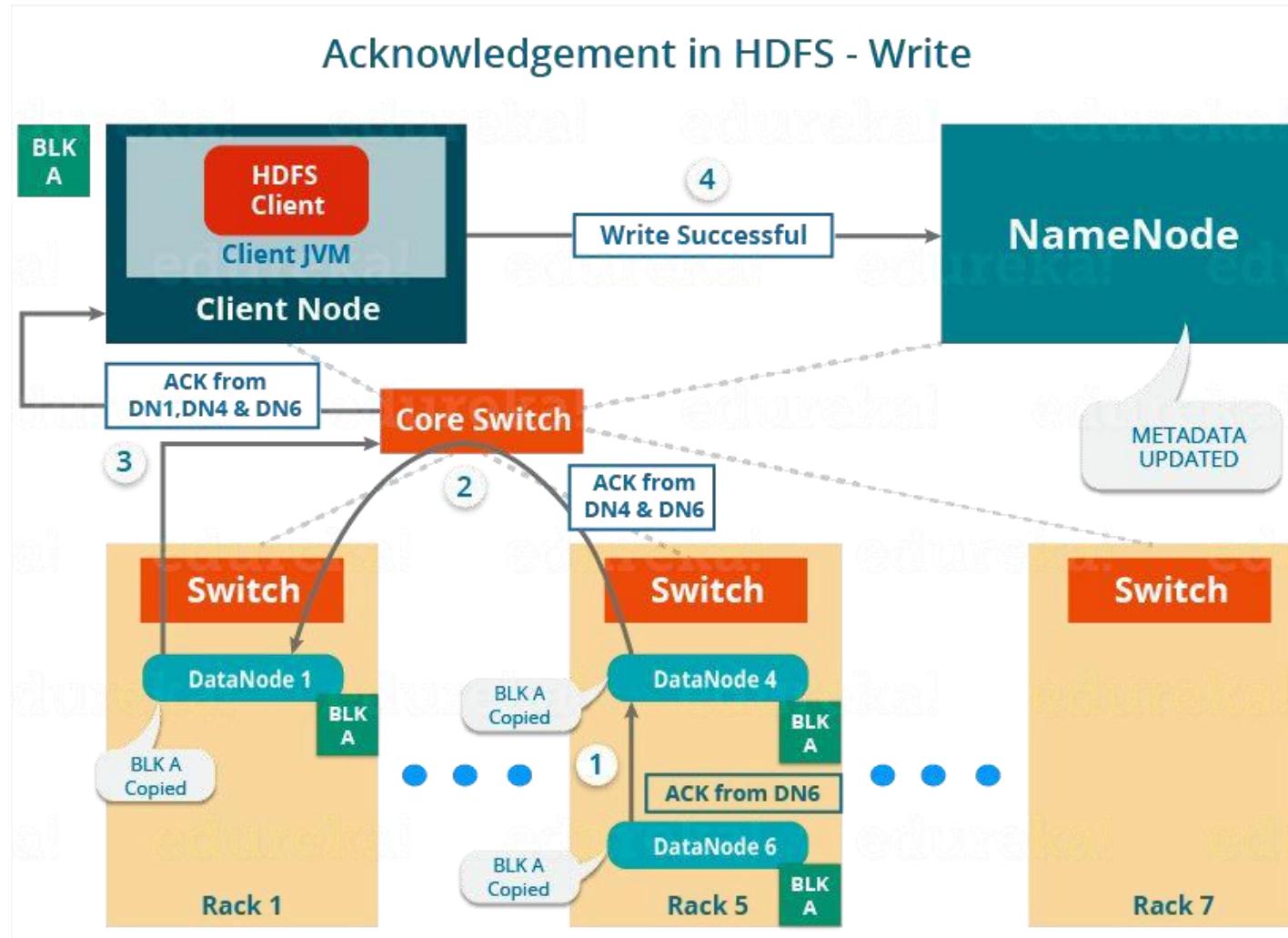
1 стадия – подготовка

# HDFS. Запись данных



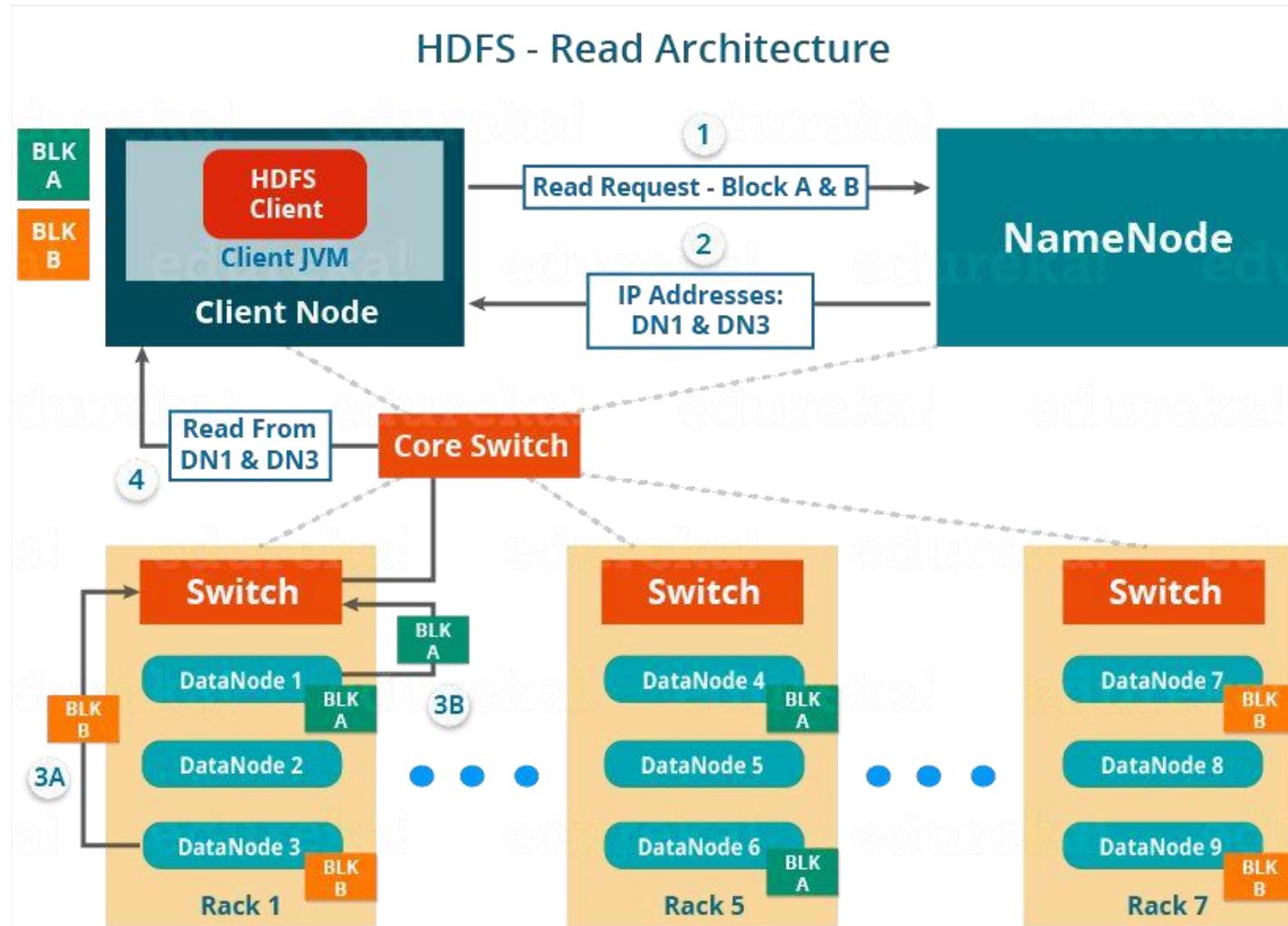
2 стадия – запись

# HDFS. Запись данных



3 стадия – завершение и обновление метаданных

# HDFS. Чтение данных



Выбирается ближайший к клиенту Datanode для чтения данных

# MapReduce

... ● ...

# MapReduce

Это модель распределенной обработки данных, предложенная компанией Google для обработки больших объемов данных на компьютерных кластерах.

Особенности:

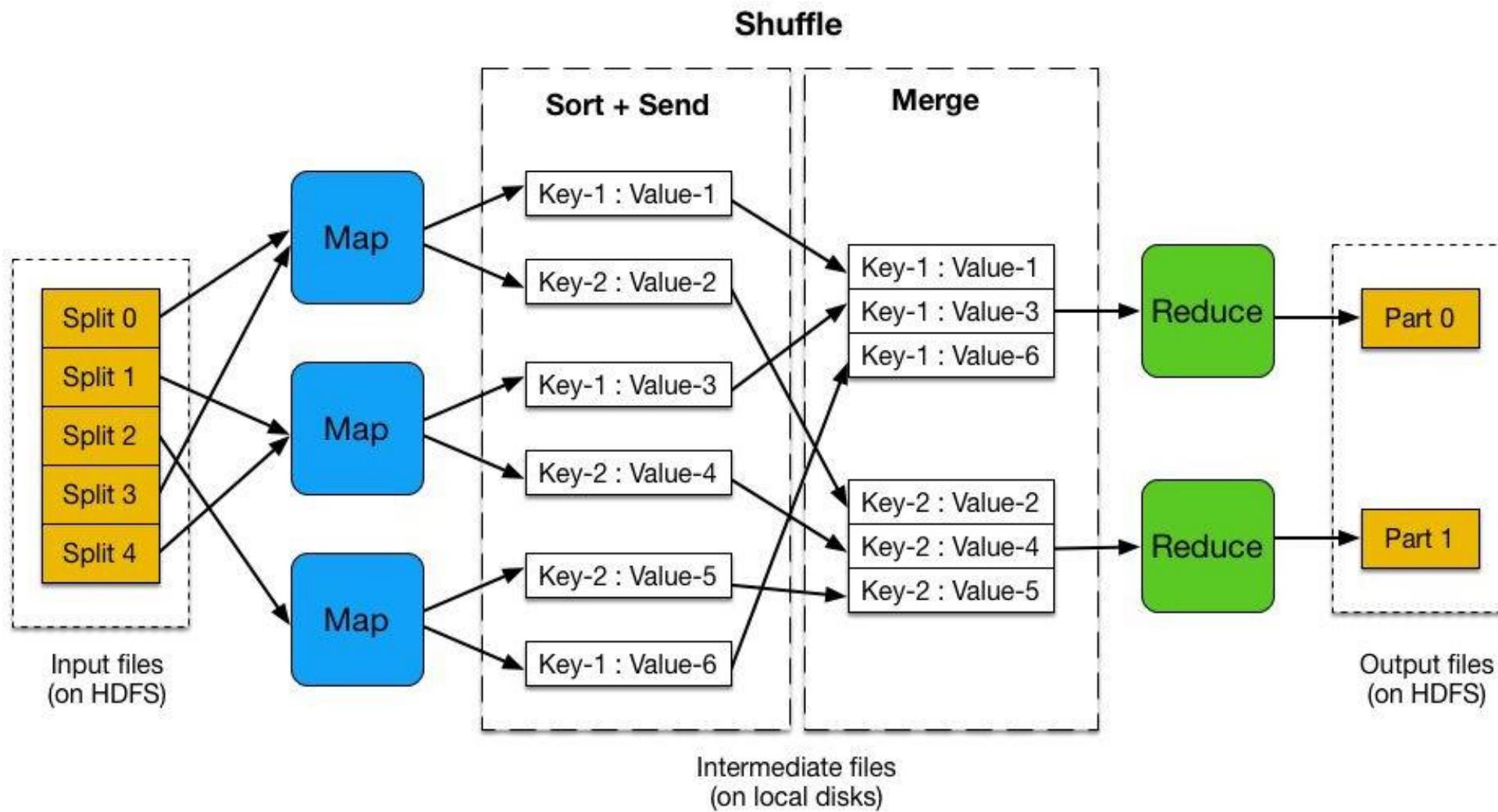
- Обрабатываем данные параллельно
- Данные обрабатываем там же где и храним (принцип локальности данных - доставляем код к данным)
- Сложный процесс обработки можно декомпозировать на несколько простых

# MapReduce. Стадии

Обработка данных происходит в 3 стадии:

1. Стадия Map
2. Стадия Shuffle
3. Стадия Reduce

# MapReduce

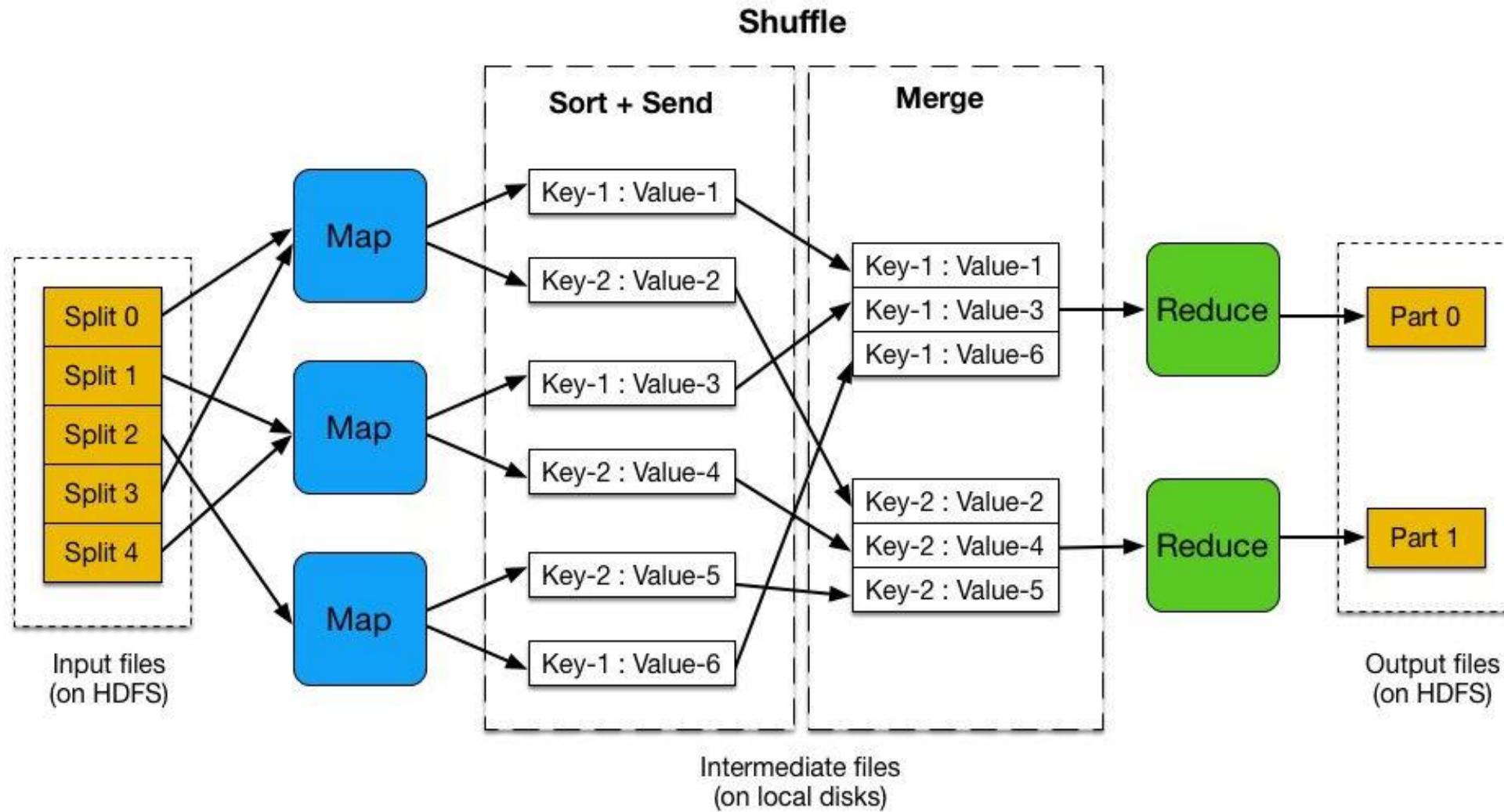


# MapReduce. Стадия map

На этой стадии данные обрабатываются при помощи функции `map()`, которую определяет пользователь. Работа этой стадии заключается в **предобработке** и фильтрации данных.

Функция `map()` выдаёт множество пар **ключ-значение**. Множество – т.е. может выдать только одну запись, может не выдать ничего, а может выдать несколько пар **ключ-значение**. Что будет находиться в ключе и в значении – решать пользователю, но ключ – очень важная вещь, так как данные **с одним ключом** в будущем попадут в один экземпляр функции `reduce`.

# MapReduce. Стадия map



# MapReduce. Стадия Shuffle

В этой стадии вывод функции map «разбирается по корзинам» – каждая корзина соответствует одному ключу вывода стадии map. В дальнейшем эти корзины послужат входом для reduce, т.е. все данные с одинаковыми ключами отправляются на один редьюсер.

На один редьюсер попадают все записи, у которых остаток от деления  $\text{hash}(\text{key})$  на количество редьюсеров совпадает:

номер редьюсера для пары  $(\text{key}, \text{value}) = \text{hash}(\text{key}) \bmod \text{num\_reducer}$

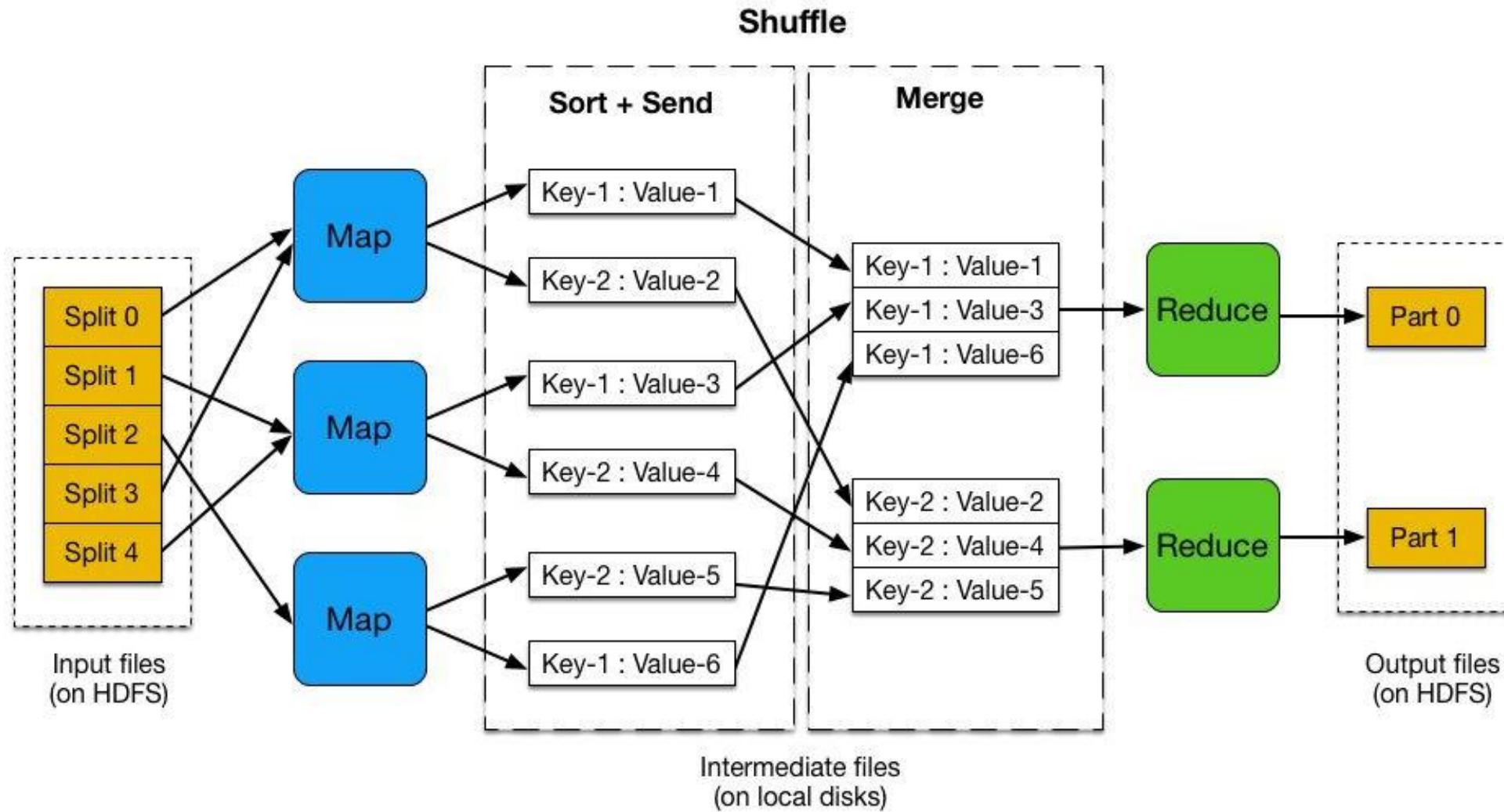
# MapReduce. Стадия Shuffle

Пример: Предположим у нас 2 редьюсера:

- номер редьюсера для пары ("car", 1) =  $\text{hash}(\text{"car"}) \bmod \text{num\_reducer} = 57686 \bmod 2 = 0$
- номер редьюсера для пары ("bear", 1) =  $\text{hash}(\text{"bear"}) \bmod \text{num\_reducer} = 123 \bmod 2 = 1$
- номер редьюсера для пары ("river", 1) =  $\text{hash}(\text{"river"}) \bmod \text{num\_reducer} = 23235 \bmod 2 = 1$
- номер редьюсера для пары ("car", 1) =  $\text{hash}(\text{"car"}) \bmod \text{num\_reducer} = 57686 \bmod 2 = 0$

Обычно самая “тяжелая” стадия выполнения.

# MapReduce. Стадия Shuffle



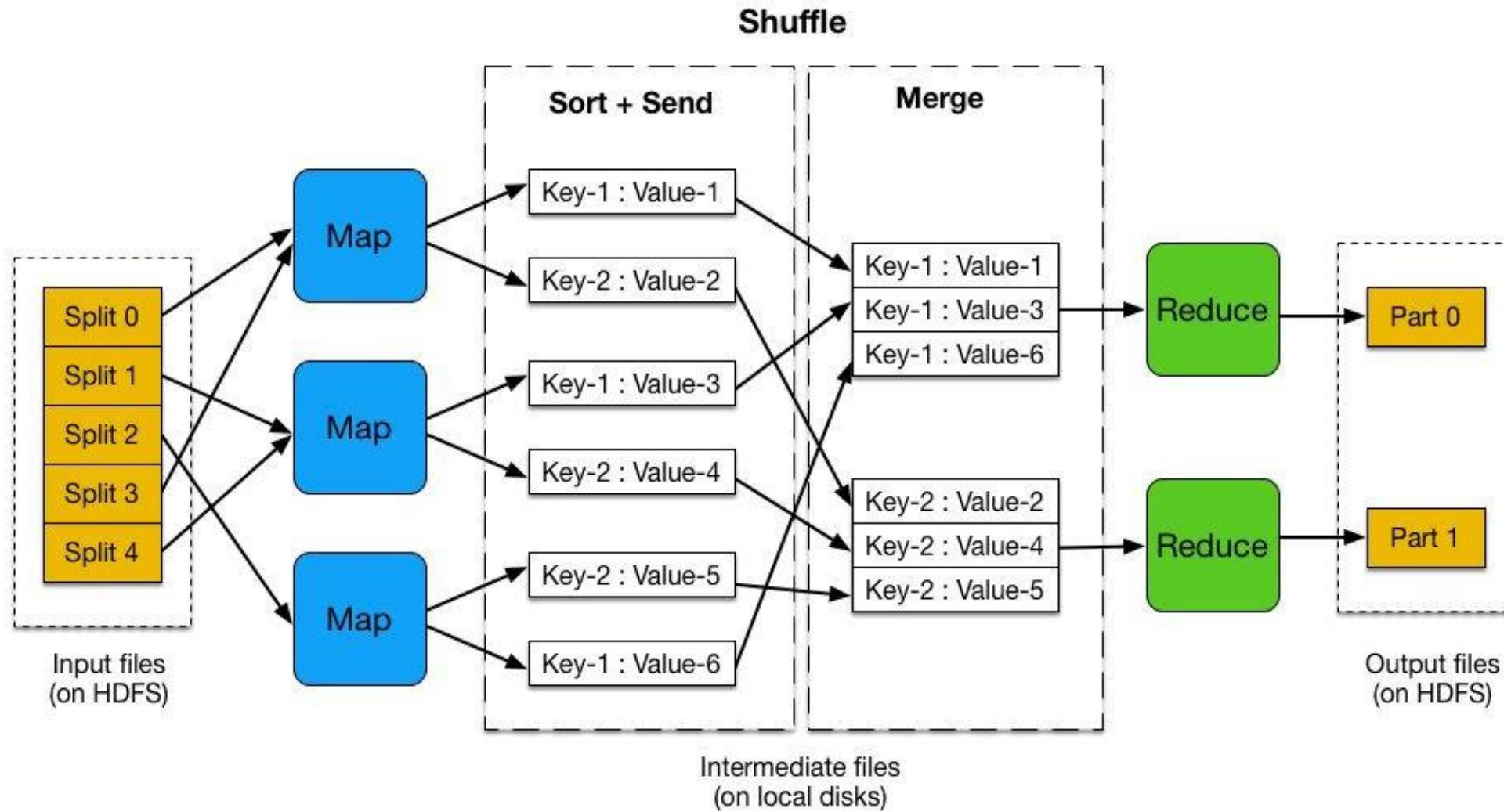
# MapReduce. Стадия Reduce

Каждая «корзина» со значениями, сформированная на стадии shuffle, попадает на вход функции `reduce()`. Функция `reduce` задаётся пользователем и вычисляет **финальный результат** для отдельной «корзины».

Множество всех значений, возвращаемых функцией `reduce()`, является финальным результатом MapReduce-задачи.

Количество выходных файлов при записи будет равно количеству редьюсеров (задается пользователем)

# MapReduce. Стадия Reduce



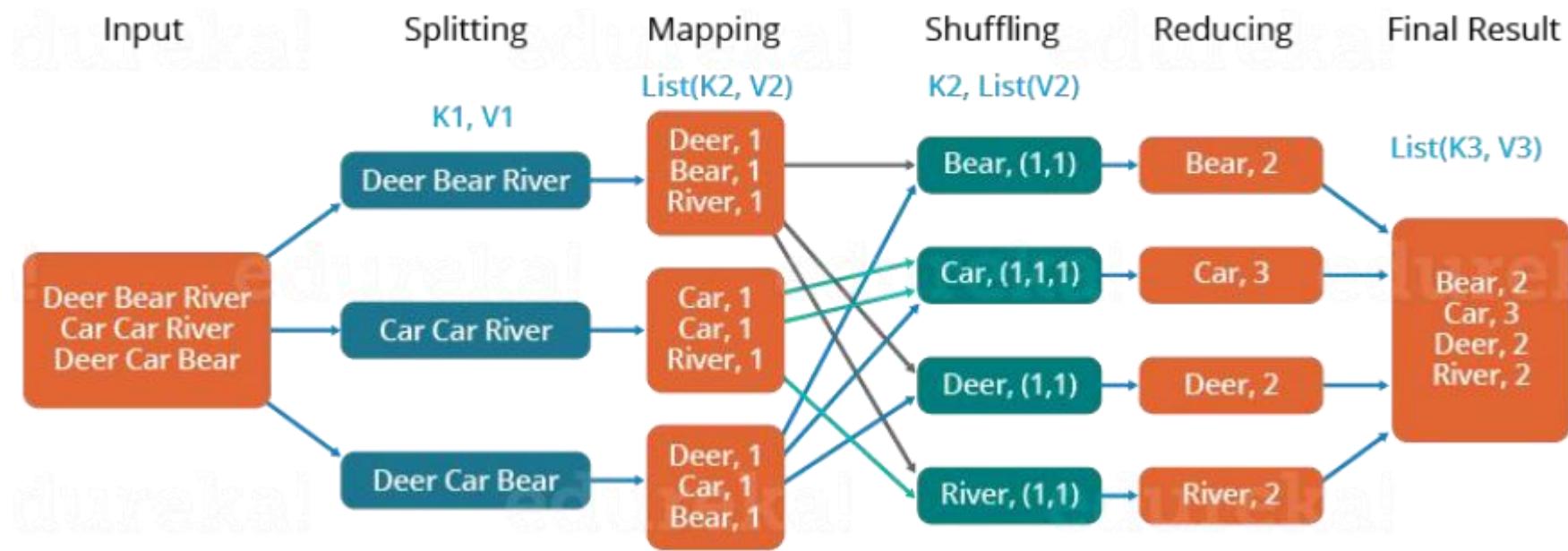
# MapReduce. Пример Word Count

Задача:

Имеется большой корпус документов. Для каждого слова, хотя бы один раз встречающегося в корпусе, подсчитать суммарное количество раз, которое оно встретилось в корпусе.

# MapReduce. Пример Word Count

The Overall MapReduce Word Count Process



```
1. def map(doc):  
2.     for word in doc:  
3.         yield word, 1
```

```
1. def reduce(word, values):  
2.     yield word, sum(values)
```

# MapReduce. Combine

В некоторых задачах можно облегчить стадию shuffle, используя combiner.

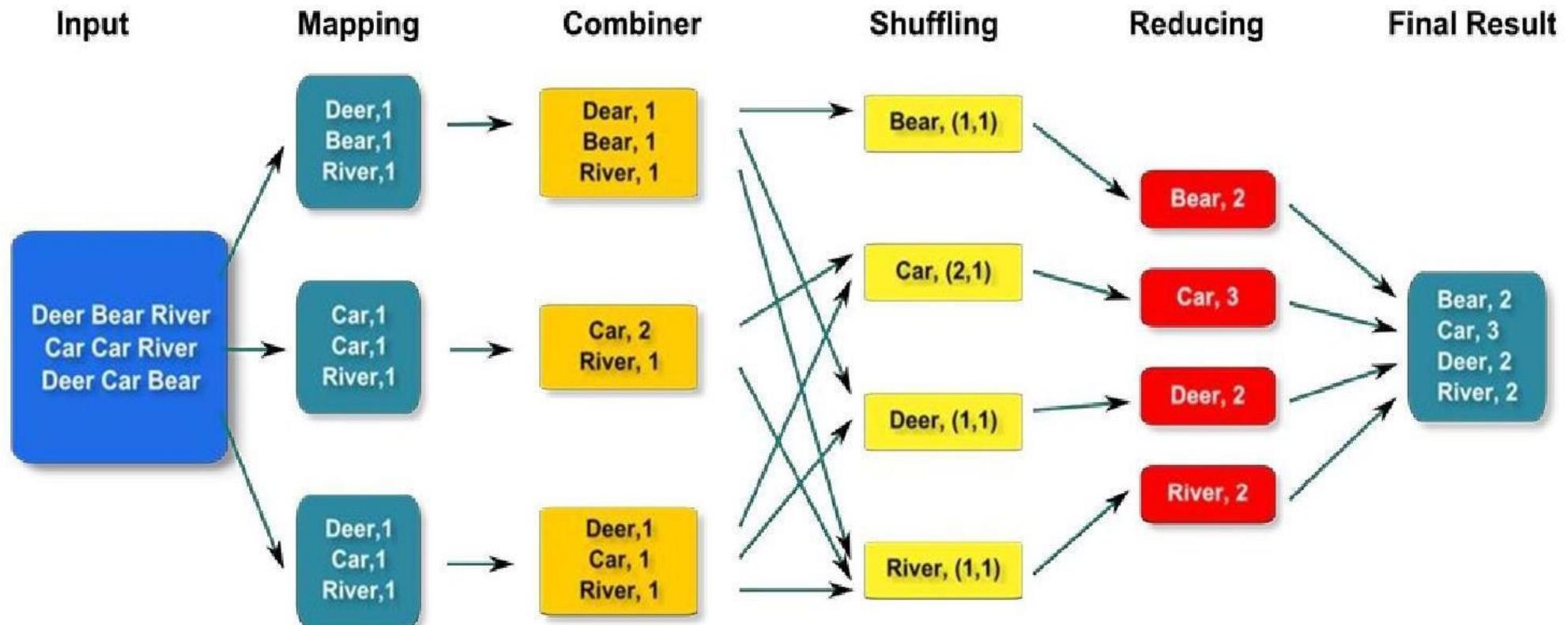
Combine – это функция очень похожая на reduce, она принимает на вход выход мапперов и выдаёт агрегированный результат для этих мапперов, тем самым сокращая передачу данных по сети на стадии shuffle.

Важное отличие от reduce – на в комбайнер попадают не все значения, соответствующие одному ключу, а те, что обрабатывались одним маппером.

# MapReduce. Combine

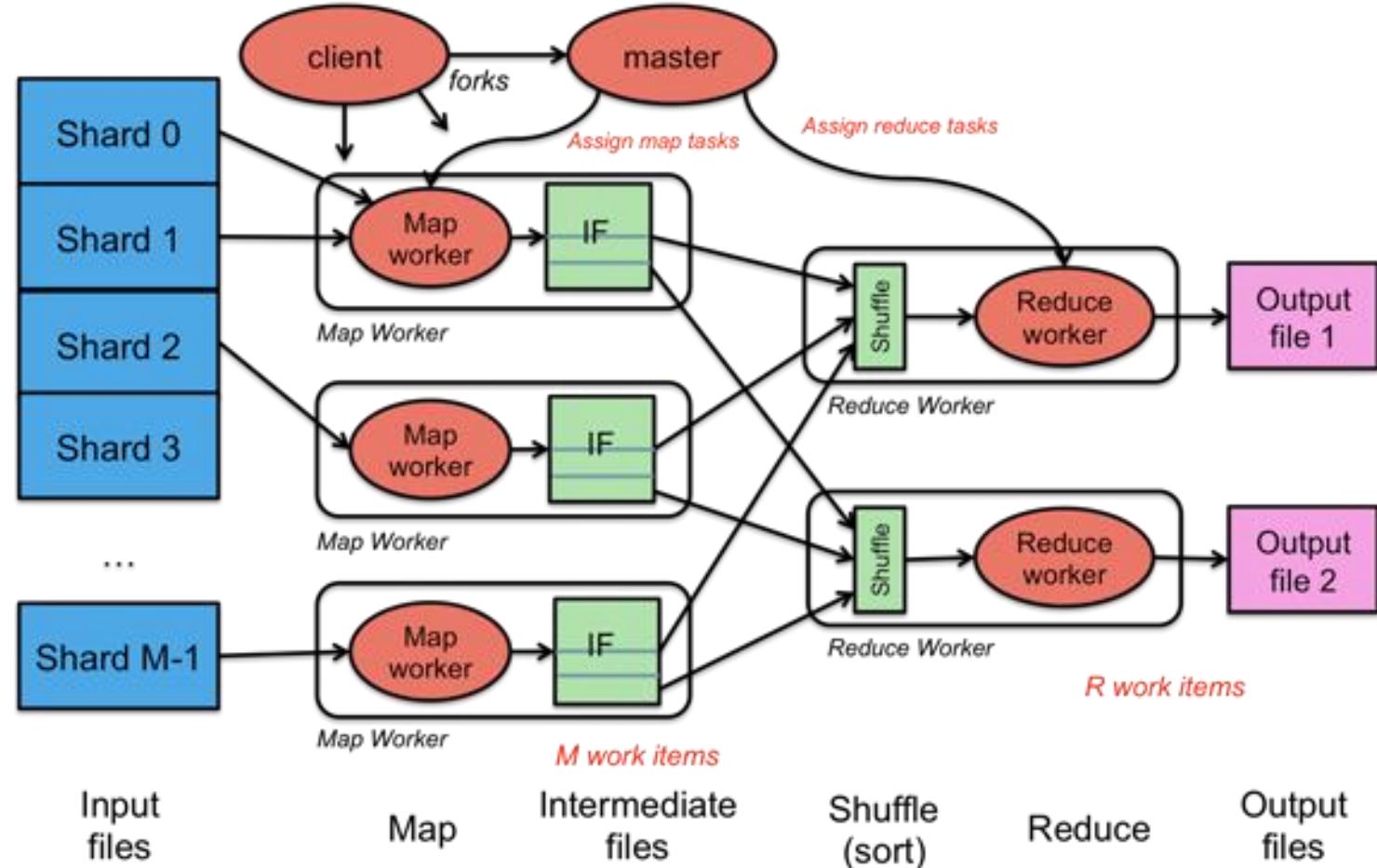
Пример использования combiner в задаче WordCount

```
1. def combine(word, values):  
2.     yield word, sum(values)
```



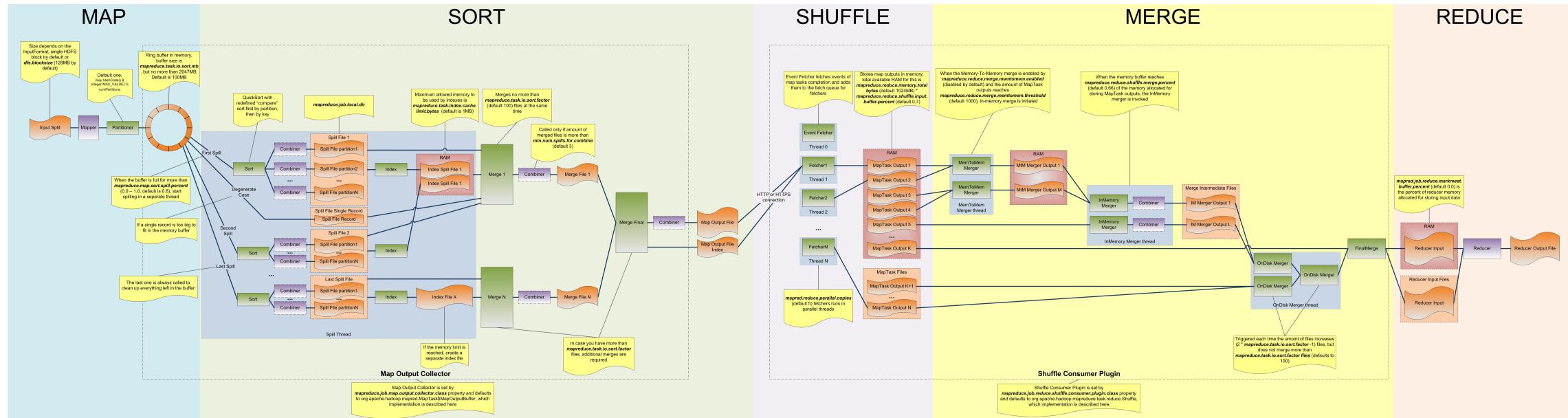
# MapReduce. И еще чуть-чуть деталей...

- Step 1. Split input
- Step 2. Fork processes
- Step 3. Map
- Step 4: Map worker: Partition
- Step 5: Reduce: Sort (Shuffle)
- Step 6: Reduce function
- Step 7: Saving results



# MapReduce. Больше деталей!

Конечно, стадий которые делает MapReduce гораздо больше:

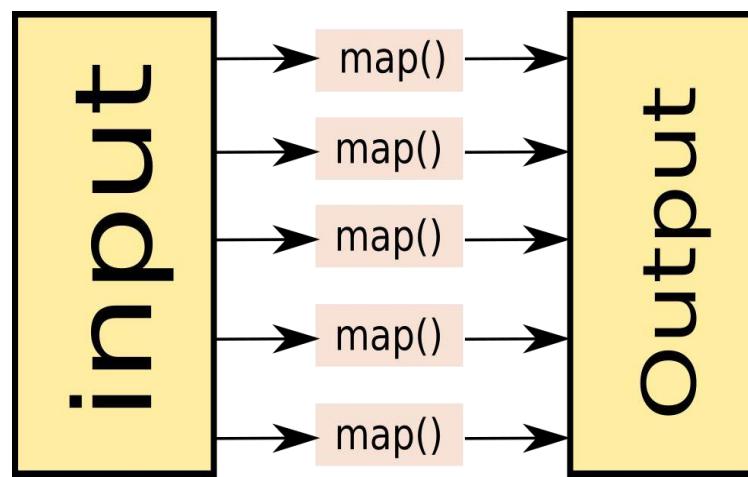


# MapReduce. Map Only Job

Существует ряд задач, в которых можно обойтись только стадией  
Мар.

Пример задачи: имеется множество логов (одна строка – одно  
событие). Произвести фильтрацию логов, оставив только логи с  
ошибками.

```
1. def map(log):  
2.     if "error" in log:  
3.         yield log
```



# MapReduce. Пример Топ-N

**Задача:** Вывести топ-N самых встречающихся слов в документе.

**MapReduce 1:**

Делаем WordCount как в примере ранее и сохраняем на диск.

**MapReduce 2:**

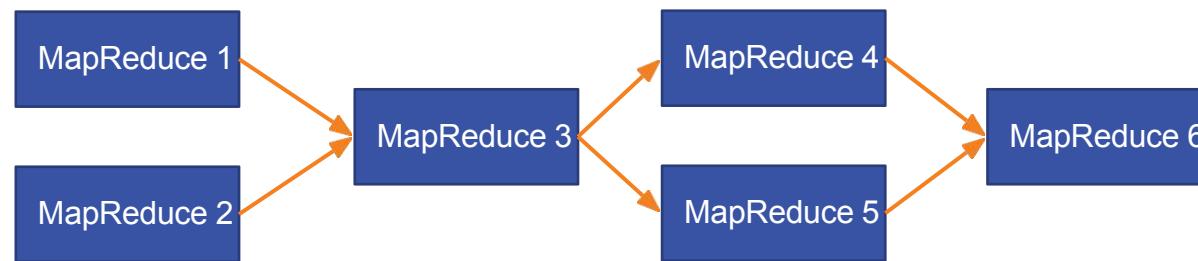
- Стадия Map: каждый маппер возвращает топ-N записей по своим данным.
- Стадия Reduce: делаем 1 редьюсер, который возвращает искомые топ-N записей.



# MapReduce. Цепочки задач

Как мы убедились выше, для решения многих задач одним MapReduce не обойтись.

Для того, чтобы выполнить последовательность MapReduce-задач, достаточно просто в качестве входных данных для второй задачи указать папку, которая была указана в качестве output для первой и запустить их по очереди.

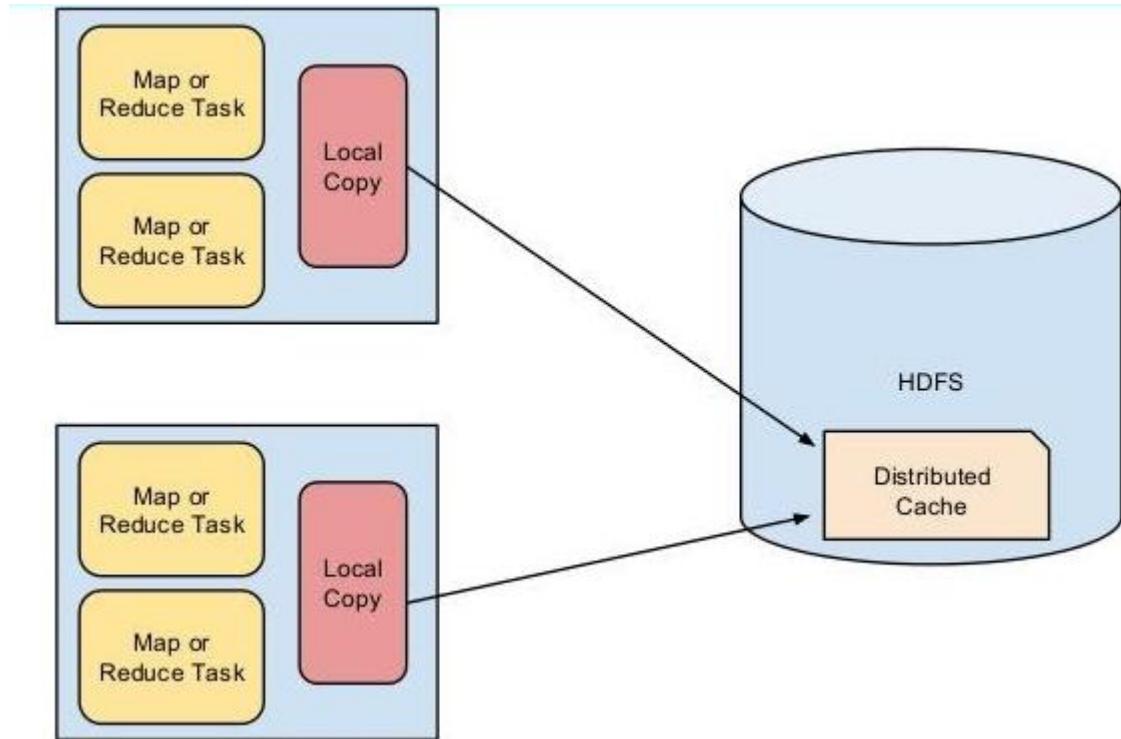


Для упрощения управления такими цепочками задач существуют отдельные инструменты, например airflow и luigi.

# MapReduce. Distributed Cache

Distributed Cache позволяет добавлять файлы [к окружению](#), в котором выполняется MapReduce-задача.

Можно добавлять файлы, хранящиеся на HDFS, локальные файлы, которыми потом можно пользоваться как будто они находятся в локальной директории.

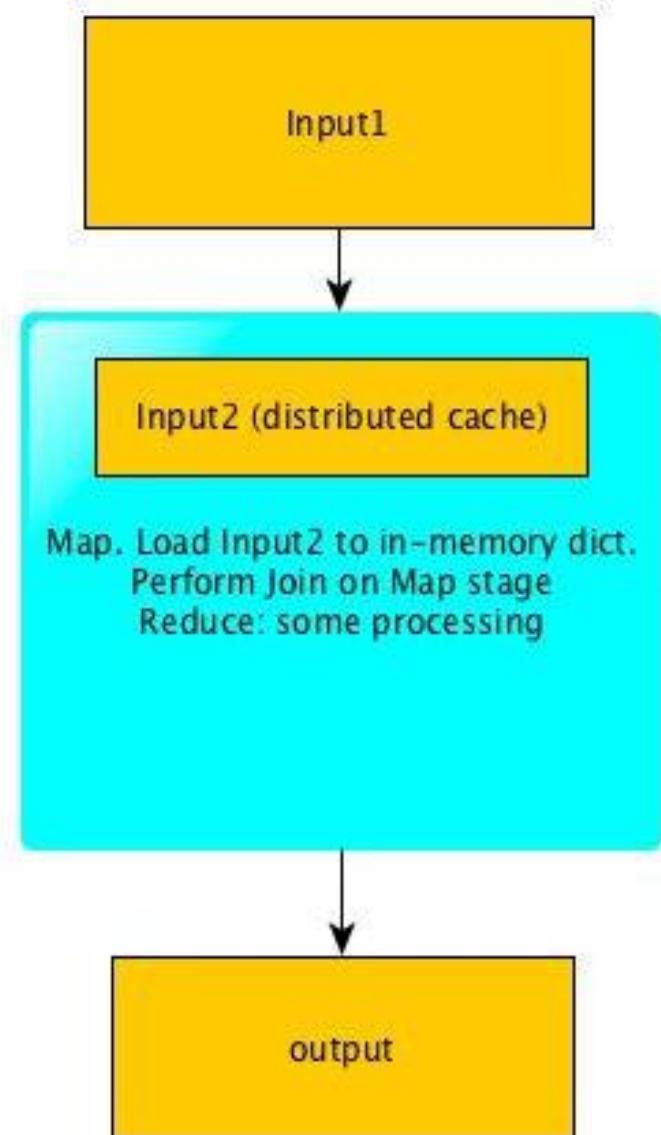


# MapReduce. Map Side Join

Если одна из соединяемых таблиц небольшая (влезает в память маппера), то мы можем загрузить эту таблицу в [Distributed Cache](#).

При инициализации маппера считать ее в память как словарь и сделать “локальный” join.

Таким образом задача соединения решается при помощи 1-го MapReduce, а, собственно, сам join вообще происходит внутри map.



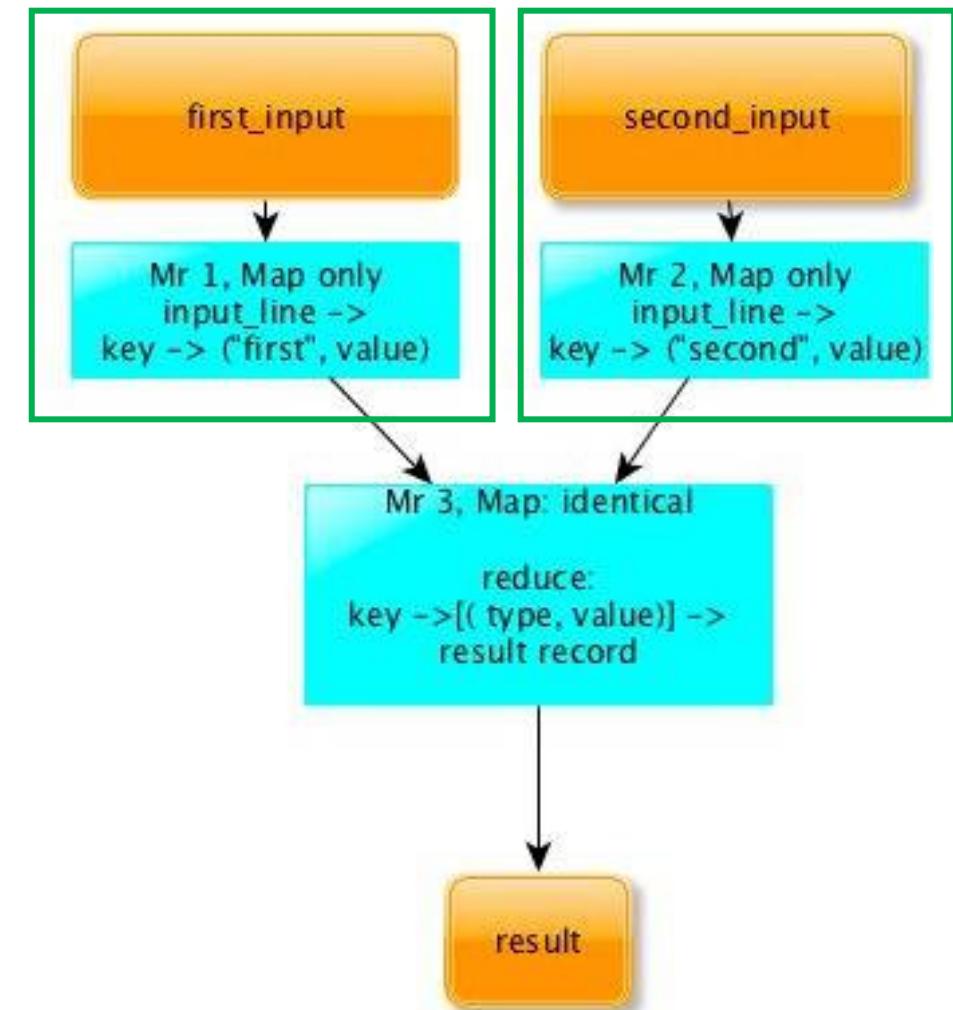
# MapReduce. Reduce Side Join (Stage 1)

На каждую из входных таблиц запускается отдельный MapReduce (Map only), выход которых:

`key -> (type, value)`

Где:

- `key` – это ключ, по которому нужно объединять таблицы,
- `type` – тип таблицы,
- `value` – это данные, привязанные к ключу

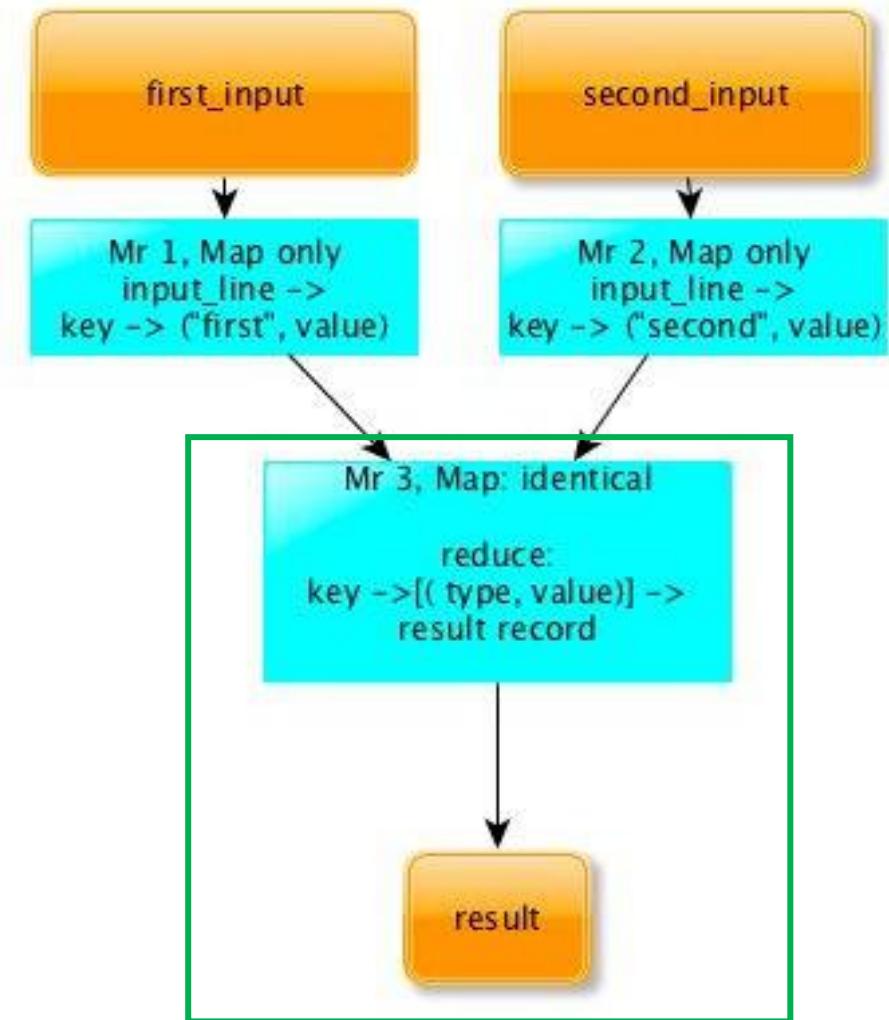


# MapReduce. Reduce Side Join (Stage 2)

Выходы обоих MapReduce подаются на вход 3-му MapReduce, который, собственно, и выполняет объединение. Маппер этой задачи просто копирует входные данные. Дальше shuffle раскладывает данные по ключам и подаёт на вход редьюсеру в виде:

`key -> [ (type, value) ]`

Важно, что в этот момент на редьюсер попадают записи из обеих таблиц и, таким образом, данных достаточно чтобы решить исходную задачу



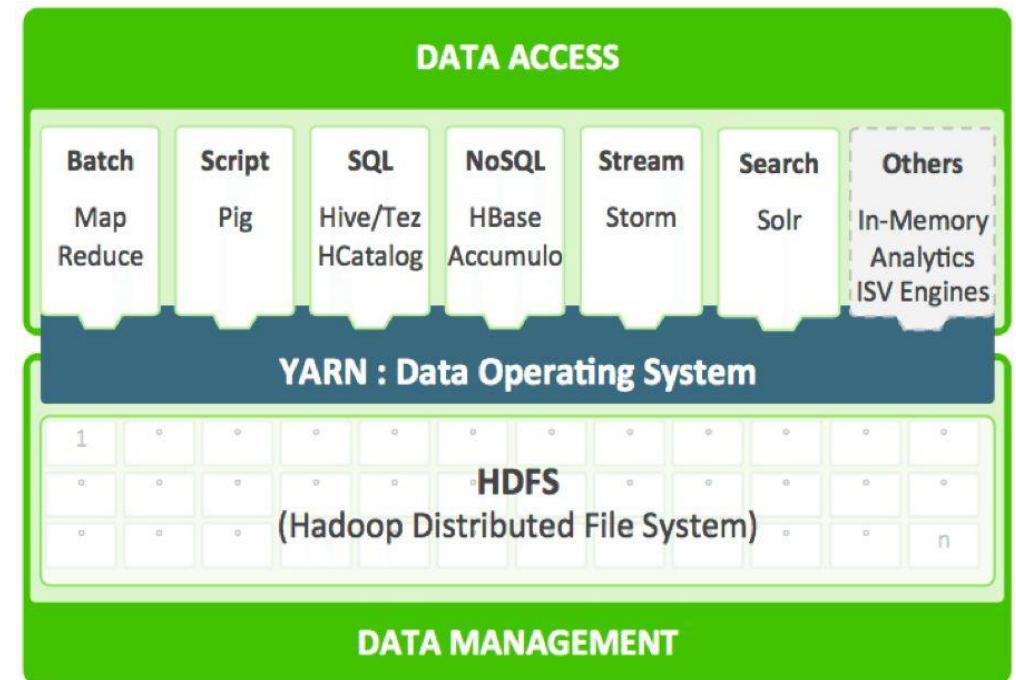
# YARN

... • • • •

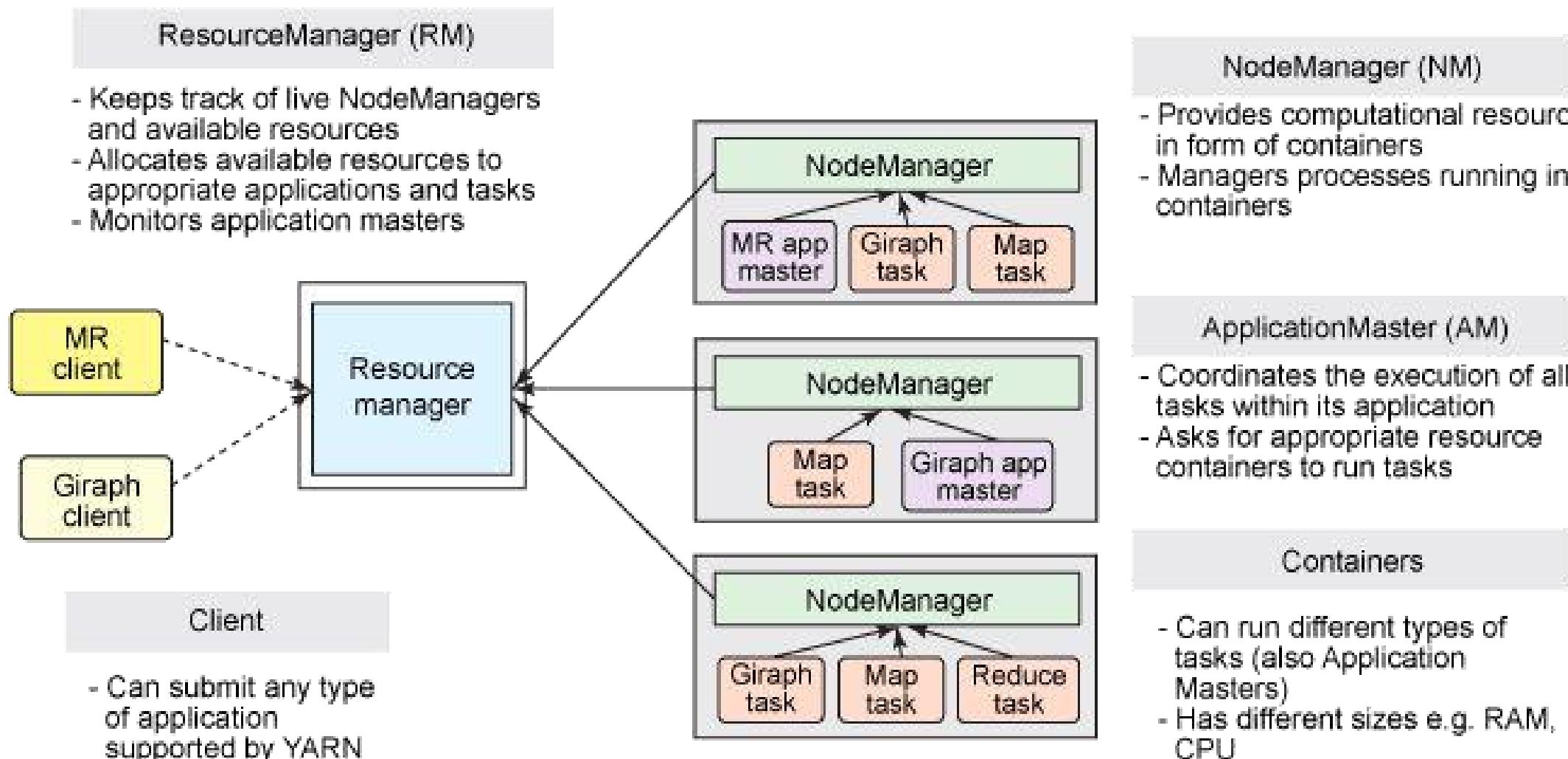
# YARN

YARN (Yet Another Resource Negotiator — «ещё один ресурсный посредник») – это программный фреймворк выполнения распределенных приложений.

- Распределение доступных ресурсов среди приложений
- Отслеживание статуса выполнения приложений



# YARN. Архитектура



# YARN. ResourceManager

Принцип схож с HDFS — есть мастер нода (Resource Manager), которая рулит всем подряд и есть менеджеры узлов (NodeManagers)

ResourceManager (RM) – глобальный менеджер ресурсов:

- Наблюдение за вычислительными узлами (живы ли они, сколько ресурсов доступно)
- Распределение ресурсов среди приложений
- Мониторинг приложений

# YARN. NodeManager

NodeManager (NM) – демоны, запущенные на вычислительном узле:

- Предоставление ресурсов в виде контейнеров (RAM+CPU)
- Отслеживание локальных ресурсов и отправка отчетов RM'у

NodeManager управляет абстрактными контейнерами – ресурсами узла, доступными для конкретного приложения.

# YARN. ApplicationMaster

ApplicationMaster (AM) – компонент, ответственный за планирование жизненного цикла,

координацию и отслеживание статуса выполнения приложения:

- Каждое приложение имеет свой экземпляр ApplicationMaster.
- Координирует (мониторинг) выполнение всех подзадач в рамках приложения.
- Ведет переговоры с RM о предоставлении ресурсов и работает с менеджерами узлов для выполнения и мониторинга подзадач

# YARN. Job Scheduling

В реальном мире ресурсы (кластера) сильно ограничены, и возникает вопрос их распределения между приложениями. Тут на помощь приходит [планировщик](#).

Scheduler (планировщик) – это компонент ResourceManager YARN. По сути это планировщик, ответственный за распределение ресурсов между затребовавшими их приложениями. Scheduler является «чистым» планировщиком: он не ведет мониторинга и не отслеживает статус приложений.

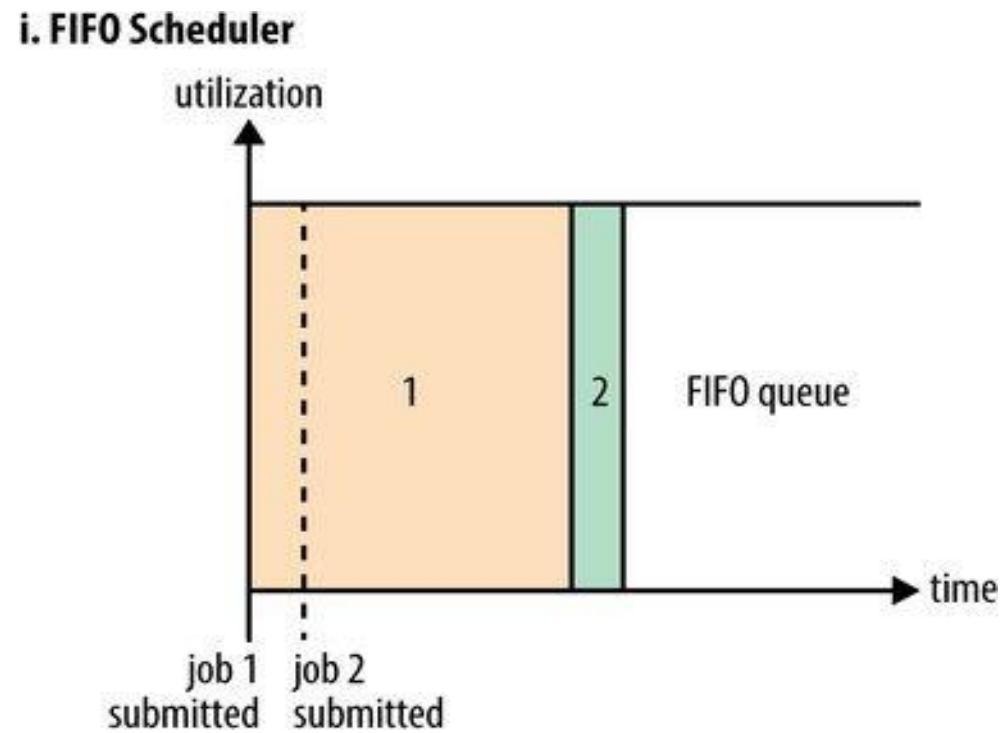
На выбор доступно 3 планировщика:

- FIFO
- Capacity Scheduler
- Fair Schedule

# YARN. FIFO Scheduler

FIFO (First In, First Out) – очередь с приоритетом, т.е. кто раньше пришел, тот раньше и отработает.

- Поддерживает приоритеты (`VERY_LOW`, `LOW`, `NORMAL`, `HIGH`, `VERY_HIGH`)
- Не подходит для общих многоцелевых кластеров (т.е. где много пользователей).

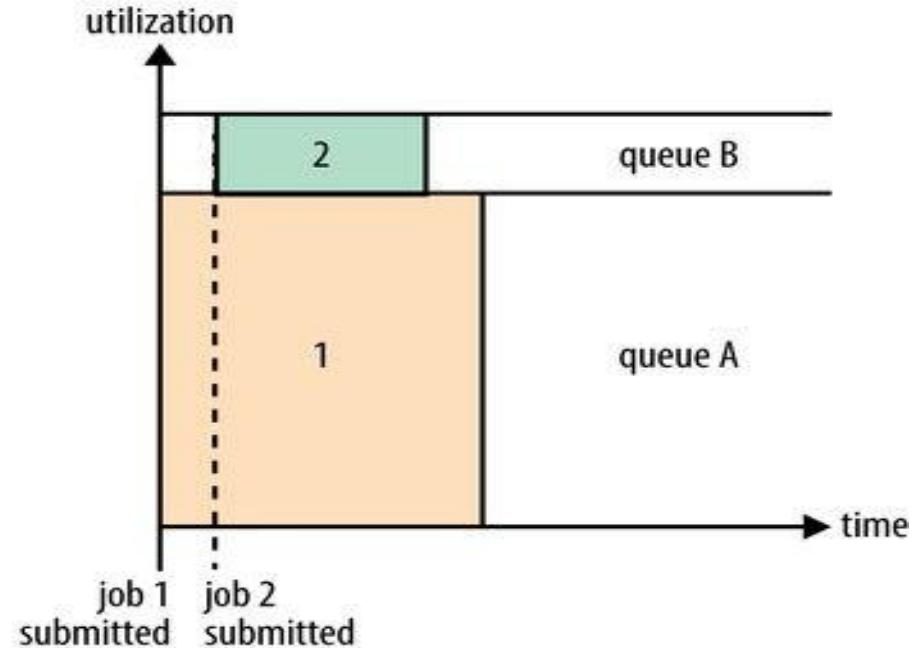


# YARN. Capacity Scheduler

## Capacity Scheduler

- Поддерживает очереди
- Гарантирует емкость ресурсов (capacity) в рамках очереди
- Подходит для общих, многоцелевых кластеров.
- Из-за гарантии емкости в кластере могут оставаться неиспользуемые ресурсы

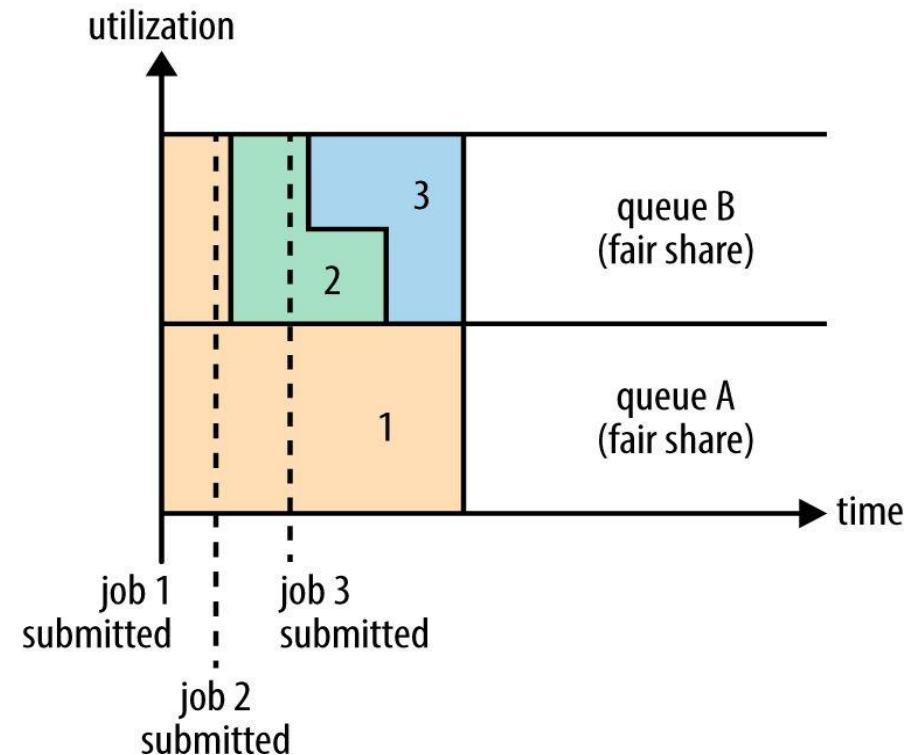
## ii. Capacity Scheduler



# YARN. Fair Scheduler

Fair Scheduler – справедливый планировщик.

- Поддерживает очереди
- Гарантирует честное распределение ресурсов в рамках очереди
- Подходит для общих, многоцелевых кластеров.
- Из-за честного распределения отнимаются ресурсы у работающих приложений, что в свою очередь несет дополнительные траты на пересчет тасков.



# YARN. WebUI

YARN Web UI – удобный инструмент, для мониторинга загрузки кластера.

Logged in as: dr.who

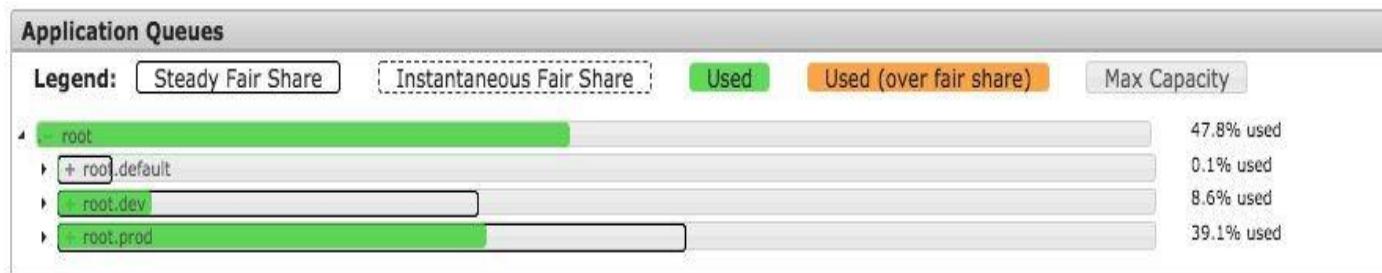
 All Applications

Cluster Metrics											
	Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	Vcores Used	Vcores Total	Vcores Reserved
1	0	1	0	2	3 GB	3 GB	0 B	2	2	0	

Cluster Nodes Metrics							
	Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes	Shutdown Nodes
1	0	0	0	0	0	0	0

Scheduler Metrics																	
Scheduler Type	Scheduling Resource Type			Minimum Allocation			Maximum Allocation			Maximum Cluster Application Priority							
Capacity Scheduler	[MEMORY]			<memory:256, vCores:1>			<memory:3072, vCores:1>			0							
Show 20 entries	Search:																
ID	User	Name	Application Type	Queue	Application Priority	StartTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU VCores	Allocated Memory MB	% of Queue	% of Cluster	Progress	Tracking UI	Blacklisted Nodes
application_1508428570624_0001	username	Spark shell	SPARK	default	0	Thu Oct 19 08:58:23 -0700 2017	N/A	RUNNING	UNDEFINED	2	2	3072	100.0	100.0	100.0	 ApplicationMaster	0

Showing 1 to 1 of 1 entries First Previous 1 Next Last



# Практика MapReduce

• • • •

# Подготовка

1. Подключаемся к кластеру:

```
ssh ВАШ_ЛОГИН@89.208.231.195
```

```
ssh v.alehin@89.208.231.195
```

2. Открываем еще одно окно и копируем файл hadoop1.zip на кластер:

```
scp /Путь/до/файла/hadoop1.zip ВАШ_ЛОГИН@89.208.231.195:/home/ВАШ_ЛОГИН/
```

```
scp /home/v.alehin/hadoop1.zip v.alehin@89.208.231.195:/home/v.alehin/
```

3. Распаковываем архив:

```
unzip hadoop1.zip
```

```
rm hadoop1.zip
```

```
ls hadoop1
```

4. Смотрим содержимое своей директории на hdfs:

```
hdfs dfs -ls /home/$USER
```

5. Создаем директорию на hdfs и копируем в нее данные:

```
hdfs dfs -mkdir /home/$USER/data
```

```
hdfs dfs -copyFromLocal ./hadoop1/data1 /home/$USER/data/
```

# Полезные команды

Для работы с hdfs:

<https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/FileSystemShell.html>

Для работы с linux:

- ls – просмотреть содержимое директории
- rm – удалить файл
- rm -r -f – удалить директорию (Опция -f принудительное удаление)
- unzip – распаковать zip-архив
- cat – вывести содержимое файла
- pwd – показывает путь к текущему каталогу
- cd /path/to/dir – перемещение к заданной директории
- cd .. – перемещение на уровень выше
- head – вывести первые 10 строк файла
- tail – вывести последние 10 строк файла
- grep – поиск в файлах
- man – наше всё (man man)

# Пример 1. Word Count

**Задача:** реализовать WordCount на MapReduce.

1. Проверить на данных `wctest.txt`
2. Проверить на данных `books`

# Пример 1. Word Count

Задача: реализовать WordCount на MapReduce.

1. Проверить на данных wctest.txt
2. Проверить на данных books

Решение:

- Маппер: script1/mr1\_wc\_mapper.py
- Редьюсер: script1/mr1\_wc\_reducer.py

Запускаем локально для проверки:

```
cat /home/$USER/hadoop1/data1/wctest.txt \
| python /home/$USER/hadoop1/script1/mr1_wc_mapper.py \
| sort -k1,1 \
| python /home/$USER/hadoop1/script1/mr1_wc_reducer.py
```

# Пример 1. Word Count

Запускаем на кластере:

```
mapred streaming \
-D mapred.reduce.tasks=1 \
-input /home/$USER/hadoop1/data1/wctest.txt \
-output /home/$USER/data/output1/mr1_wctest \
-mapper mr1_wc_mapper.py \
-reducer mr1_wc_reducer.py \
-file /home/$USER/hadoop1/script1/mr1_wc_mapper.py \
-file /home/$USER/hadoop1/script1/mr1_wc_reducer.py
```

# Пример 1. Word Count

Запускаем на кластере:

```
mapred streaming \
-D mapred.reduce.tasks=1 \ # Количество редьюсеров
-input ... \ # Путь до данных на hdfs
-output ... \ # Куда записать результат на hdfs
-mapper ... \ # Код маппера
-reducer ... \ # Код редьюсера
-file ...      # Файлы, которые понадобятся для работы
                # обычно файлы с кодом маппера и редьюсера
```

С полным списком параметров можно ознакомится по ссылке:

<https://hadoop.apache.org/docs/current/hadoop-streaming/HadoopStreaming.html>

# Пример 1. Word Count

Проверяем результаты работы:

```
hdfs dfs -ls /home/$USER/data/output1/mr1_wctest/  
hdfs dfs -cat /home/$USER/data/output1/mr1_wctest/*
```

Удаляем результаты работы прошлого запуска:

```
hdfs dfs -rm -r /home/$USER/data/output1/mr1_wctest
```

## Пример 2. Top N

**Задача:** Вывести топ- $N$  самых встречающихся слов в документах.

## Пример 2. Top N

**Задача:** Вывести топ- $N$  самых встречающихся слов в документах.

**Решение:**

MR1:

Делаем WordCount как в прошлом примере, сохраняем на диск.

MR2:

- Map: каждый маппер возвращает топ- $N$  записей по своим данным
- Reduce: делаем 1 редьюсер, который возвращает искомые топ- $N$  записей.

Код маппера и редьюсера **совпадают**:

`script1/mr2_topn_mapper_reducer.py`

## Пример 2. Top N

Запускаем на кластере:

```
mapred streaming \
-D mapred.reduce.tasks=1 \
-input /home/$USER/data/output1/mr1_wcbooks/* \
-output /home/$USER/data/output1/mr2_topn \
-mapper mr2_topn_mapper_reducer.py \
-reducer mr2_topn_mapper_reducer.py \
-file /home/$USER/hadoop1/script1/mr2_topn_mapper_reducer.py
```

Проверяем результаты работы:

```
hdfs dfs -ls /home/$USER/data/output1/mr2_topn
hdfs dfs -cat /home/$USER/data/output1/mr2_topn/*
```

## Пример 2. Top N

Следует обратить внимание:

- Код маппера и редьюсера **совпадают**
- **Один** редьюсер
- Используем результаты прошлого шага

/home/\$USER/data/output1/mr1\_wcbooks

# Пример 3. Inner Join

**Задача:**

Написать inner-join на MapReduce  
(ReduceSideJoin) для двух таблиц по полю  
`product_id`:

1. `shop_product.csv` содержит поля  
`product_id` и `description`
2. `shop_price.csv` содержит поля `product_id` и  
`price`

В результате должен получится файл из трех  
колонок:

`(product_id, description, price)`

**Содержимое файла результата:**

```
1 Картофель, кг 15.43
1 Картофель, кг 29.96
2 Капуста свежая, кг 26.03
3 Лук репчатый, кг 31.87
5 Свекла, кг 35.32
6 Морковь, кг 46.43
7 Яблоки, кг 111.79
7 Яблоки, кг 127.08
8 Апельсины, кг 84.82
```

Порядок строк не важен, разделитель – \t

# Пример 3. Inner Join

Решение

# Не пример 1. Inner Join

Решение данной задачи будет рассмотрено на следующем занятии, но при желании вы можете попытать свои силы и получить дополнительные баллы (10). Присылайте решение на портале до:

2024-03-26 17:59:59

Подсказки:

- Обратите внимание на разделители полей в исходных файлах
- Первые две MR-задачи имеют тип MapOnly (`-D mapred.reduce.tasks=0`)

Решение **должно** содержать архив с файлами с кодом:

- `mr3_join_mapper.py`, `mr3_join_reducer.py`
- `mr3_price_mapper.py`, `mr3_product_mapper.py` (исходники были в архиве `hadoop1.zip`)
- `README` с командами запуска на кластере и описанием решения.

Название архива: `mr_join_вашлогин.zip`

## Не пример 2. Агрегация

Также, предлагается решить задачу агрегации данных, для закрепления знаний о MapReduce.

Варианты задания будут выданы [индивидуально в посте](#). Требования к отправке решения аналогичны.

Решение должно содержать архив с файлами с кодом

- `mr4_agg_mapper.py`, `mr4_agg_reducer.py`
- `README` с командами запуска на кластере и описанием решения.

Название архива: `mr4_agg_вашлогин.zip`.

Спасибо за  
внимание!

