

Hadoop



План на неделю



Что такое большие данные



План на неделю



Что такое большие данные



Экосистема Hadoop для работы с большими данными



План на неделю



Что такое большие данные



Экосистема Hadoop для работы с большими данными



HDFS для хранения



План на неделю



Что такое большие данные



Экосистема Hadoop для работы с большими данными



HDFS для хранения



MapReduce для вычислений



Большие данные и Hadoop



Большие данные

→ Сегодня собирается огромное количество **данных об использовании** мобильных приложений, социальных сетей, интернет-магазинов и так далее



Большие данные

Журналистка попросила Tinder в рамках GDPR выслать всю имеющуюся на нее информацию, в ответ получила 800 страниц материалов, включая:

- Лайки в Facebook

Большие данные

Журналистка попросила Tinder в рамках GDPR выслать всю имеющуюся на нее информацию, в ответ получила 800 страниц материалов, включая:

- Лайки в Facebook
- Посты в Instagram

Большие данные

Журналистка попросила Tinder в рамках GDPR выслать всю имеющуюся на нее информацию, в ответ получила 800 страниц материалов, включая:

- Лайки в Facebook
- Посты в Instagram
- Образование

Большие данные

Журналистка попросила Tinder в рамках GDPR выслать всю имеющуюся на нее информацию, в ответ получила 800 страниц материалов, включая:

- Лайки в Facebook
- Посты в Instagram
- Образование
- Когда и как ведется переписка в Tinder

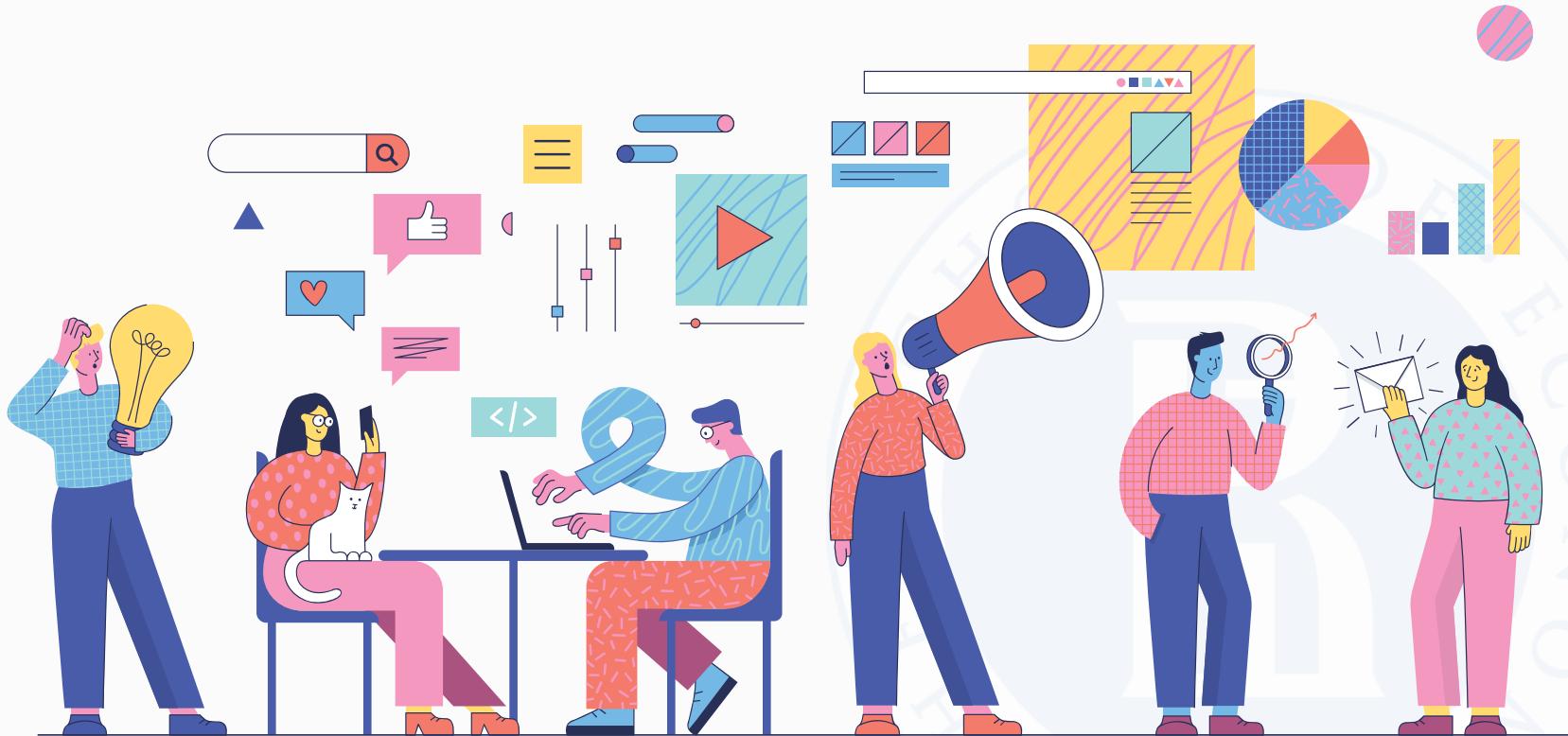
Большие данные

Журналистка попросила Tinder в рамках GDPR выслать всю имеющуюся на нее информацию, в ответ получила 800 страниц материалов, включая:

- Лайки в Facebook
- Посты в Instagram
- Образование
- Когда и как ведется переписка в Tinder
- Параметры людей, которых чаще лайкает в Tinder

Большие данные

→ Эти данные помогают компаниям улучшать сервис для пользователя: рекомендовать **персонально** фильмы, музыку, товары, других пользователей, и т. д.



Большие данные (Big Data)



Большие данные отличаются 3 важными свойствами (3V):

Большие данные (Big Data)



Большие данные отличаются 3 важными свойствами (3V):

- **Volume** — объем данных, который все время растет, так как активностей и пользователей все больше. В некоторые организации могут поступать 10 ТБ, в другие — 100 ПБ

Большие данные (Big Data)



Большие данные отличаются 3 важными свойствами (3V):

- **Volume** — объем данных, который все время растет, так как активностей и пользователей все больше. В некоторые организации могут поступать 10 ТБ, в другие — 100 ПБ
- **Velocity** — скорость генерации данных и, возможно, действий на их основе. Сегодня генерируются миллиарды постов в Facebook, поисков в Google, твитов в Twitter ежедневно

Большие данные (Big Data)

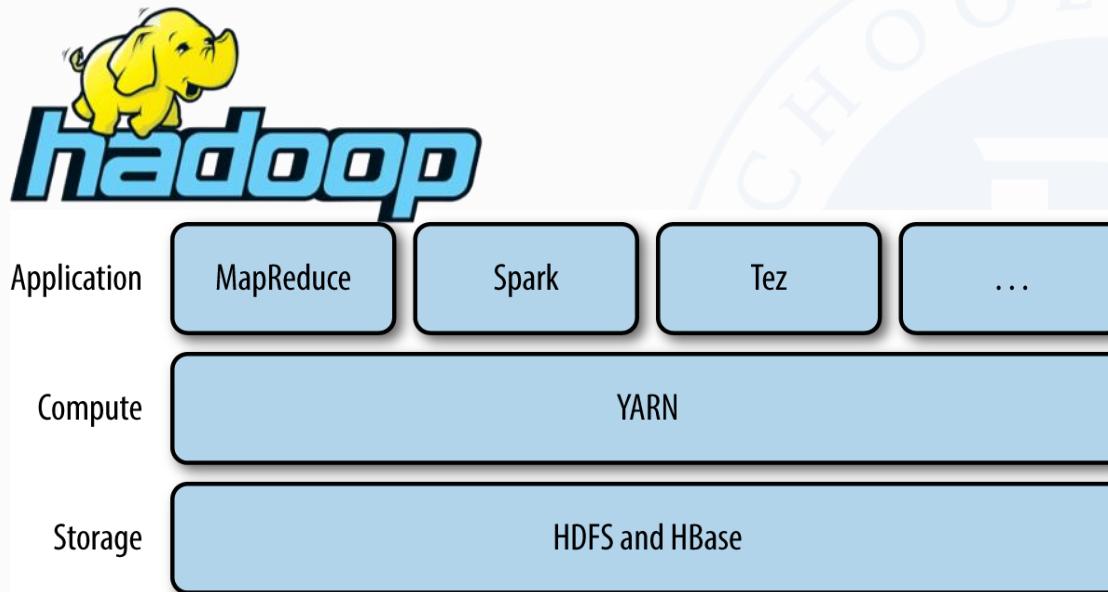


Большие данные отличаются 3 важными свойствами (3V):

- **Volume** — объем данных, который все время растет, так как активностей и пользователей все больше. В некоторые организации могут поступать 10 ТБ, в другие — 100 ПБ
- **Velocity** — скорость генерации данных и, возможно, действий на их основе. Сегодня генерируются миллиарды постов в Facebook, поисков в Google, твитов в Twitter ежедневно
- **Variety** — разнообразие данных. С появлением Big Data данные стали поступать в неструктурированном виде. Например, текст, аудио и видео требуют обработки

Экосистема Hadoop для Big Data

- HDFS — распределенная файловая система
- YARN — менеджер ресурсов кластера (CPU, RAM, и.т.д.)
- MapReduce — API для распределенных вычислений



Масштабируемость



Когда данных было меньше и они были более структурированными, их можно было положить в реляционную базу данных, заранее продумав схему данных и список возможных запросов к ним

Масштабируемость

- Когда данных было меньше и они были более структурированными, их можно было положить в реляционную базу данных, заранее продумав схему данных и список возможных запросов к ним
- Чтобы ускорить обработку в RDBMS, нужен сервер помощнее ([вертикальная масштабируемость](#))

Масштабируемость

- Когда данных было меньше и они были более структурированными, их можно было положить в реляционную базу данных, заранее продумав схему данных и список возможных запросов к ним
- Чтобы ускорить обработку в RDBMS, нужен сервер помощнее ([вертикальная масштабируемость](#))
- В Hadoop для ускорения обработки нужно больше средненьких серверов ([горизонтальная масштабируемость](#)), что сильно дешевле

RDBMS vs Hadoop

	RDBMS	Hadoop
Количество серверов	Один мощный	Много средних
Объем данных	Маленький	Большой
Скорость запросов	Моментальный ответ	Ответ с задержкой
Формат данных	Таблицы (структурированные)	Файлы (неструктурированные)
Требования ACID	Выполняются	Не требуются

Резюме



Большие данные: объем, скорость, разнообразие



Резюме



Большие данные: объем, скорость, разнообразие



Hadoop позволяет горизонтально масштабироваться

Резюме



Большие данные: объем, скорость, разнообразие



Hadoop позволяет горизонтально масштабироваться



Далее: файловая система HDFS

Файловая система HDFS

Hadoop Distributed File System (HDFS)

- Файлы разбиты на **блоки**, которые хранятся на разных машинах (**Data Nodes**)

Hadoop Distributed File System (HDFS)

- Файлы разбиты на **блоки**, которые хранятся на разных машинах (**Data Nodes**)
- Каждый блок **реплицируется** на нескольких **Data Nodes** (количество реплик конфигурируется, например, 3)

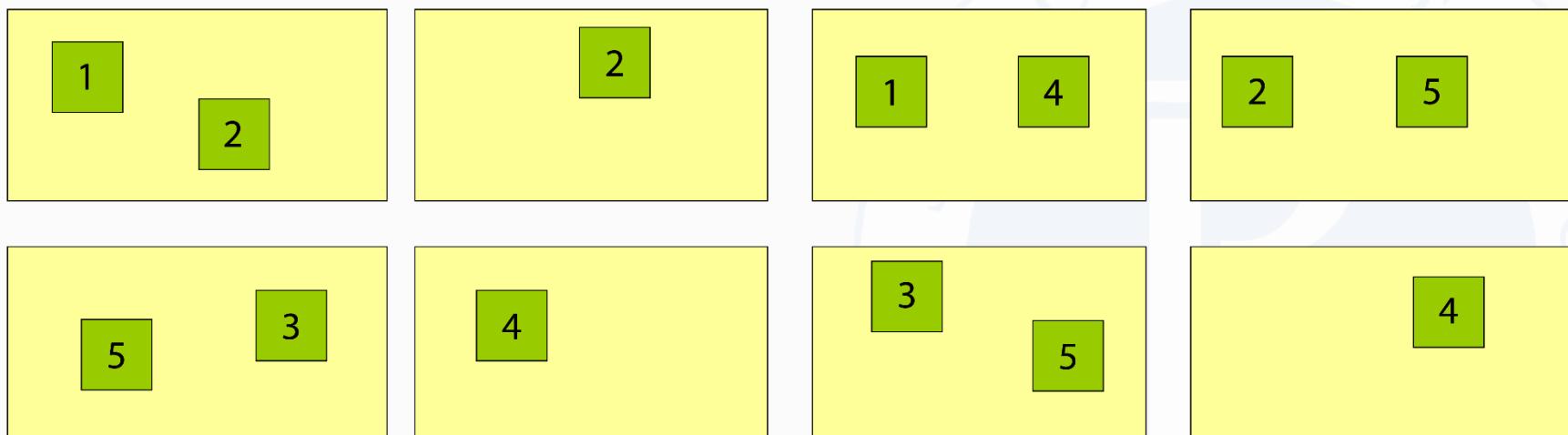
Hadoop Distributed File System (HDFS)

- Файлы разбиты на **блоки**, которые хранятся на разных машинах (**Data Nodes**)
- Каждый блок **реплицируется** на нескольких **Data Nodes** (количество реплик конфигурируется, например, 3)
- Соответствие **имя файла** → **блоки** хранится в памяти специальной машины (**Name Node**)

Репликация блоков

Namenode (Filename, numReplicas, block-ids, ...)
/users/sameerp/data/part-0, r:2, {1,3}, ...
/users/sameerp/data/part-1, r:3, {2,4,5}, ...

Datanodes



Зачем нужна репликация блоков

→ Предположим, что сервер ломается с вероятностью 0.001



Зачем нужна репликация блоков

- Предположим, что сервер ломается с вероятностью 0.001
- Какова вероятность что хотя бы 1 из 500 серверов сломается?

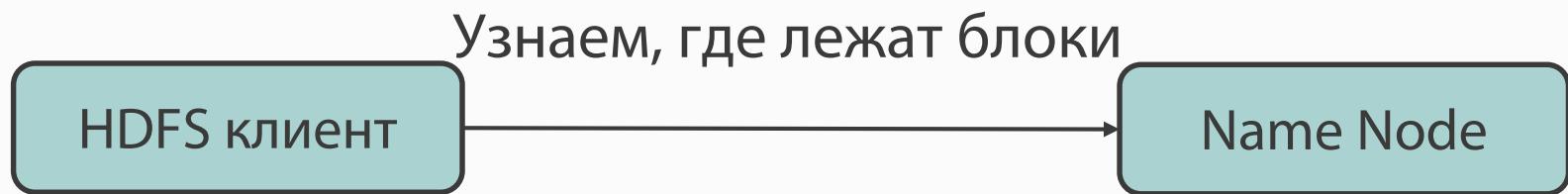


Зачем нужна репликация блоков

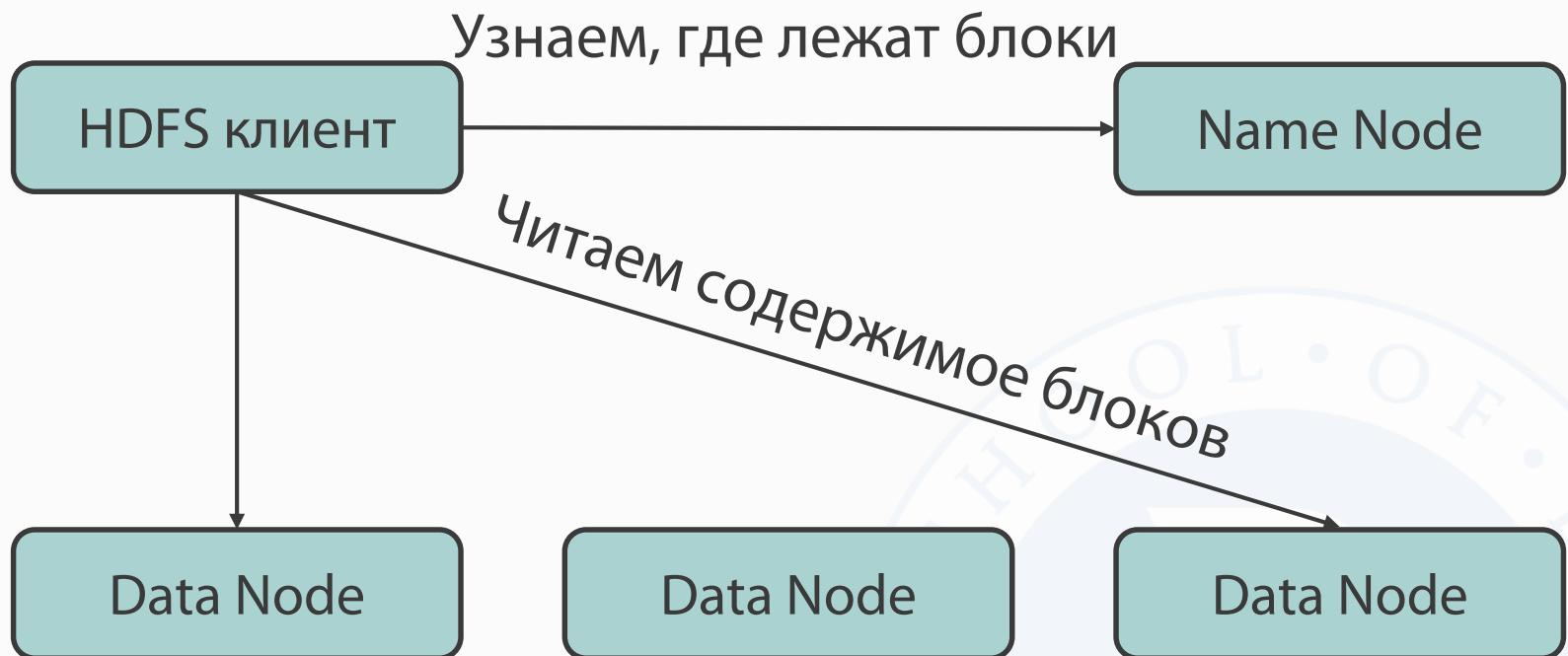
- Предположим, что сервер ломается с вероятностью 0.001
- Какова вероятность что хотя бы 1 из 500 серверов сломается?
- $1 - (1 - 0.001)^{500} \approx 0.4$



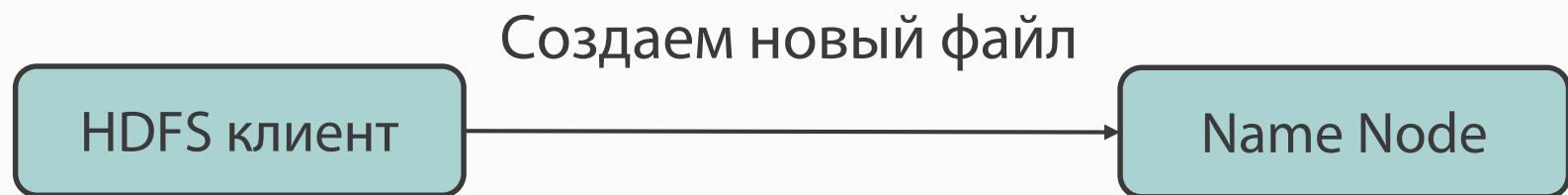
Чтение из HDFS



Чтение из HDFS

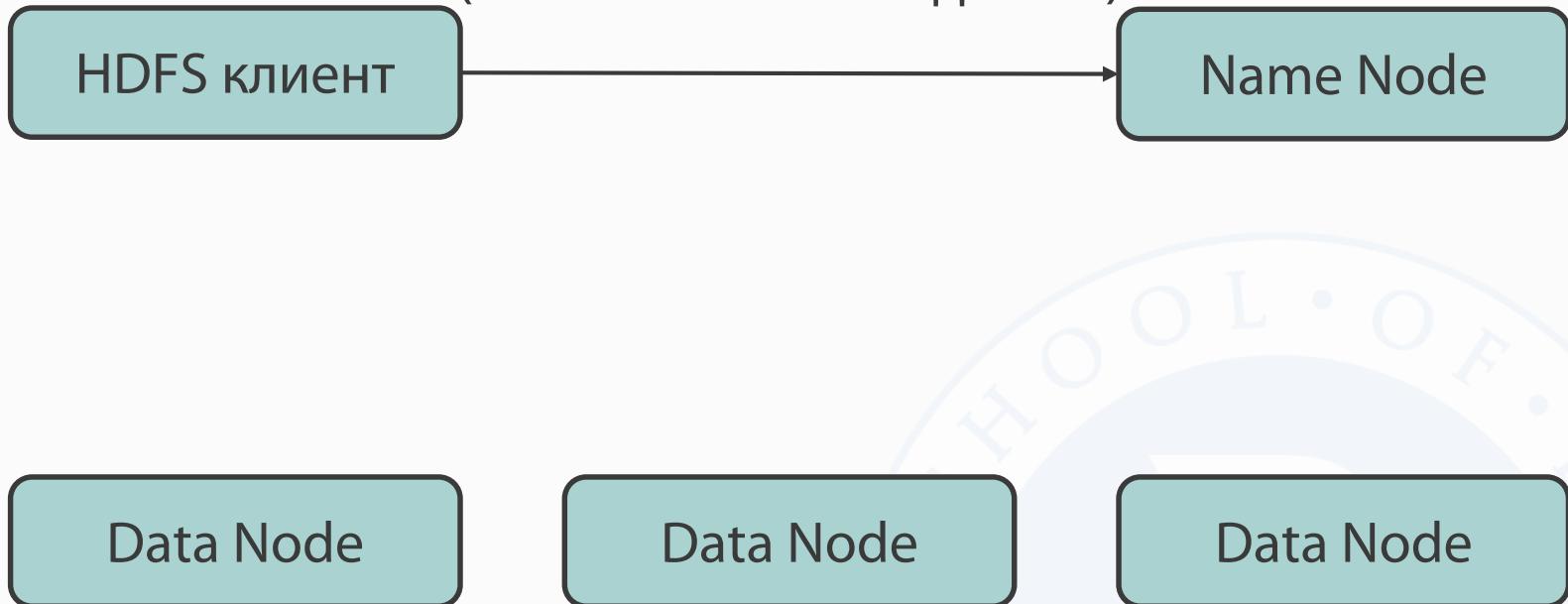


Запись в HDFS

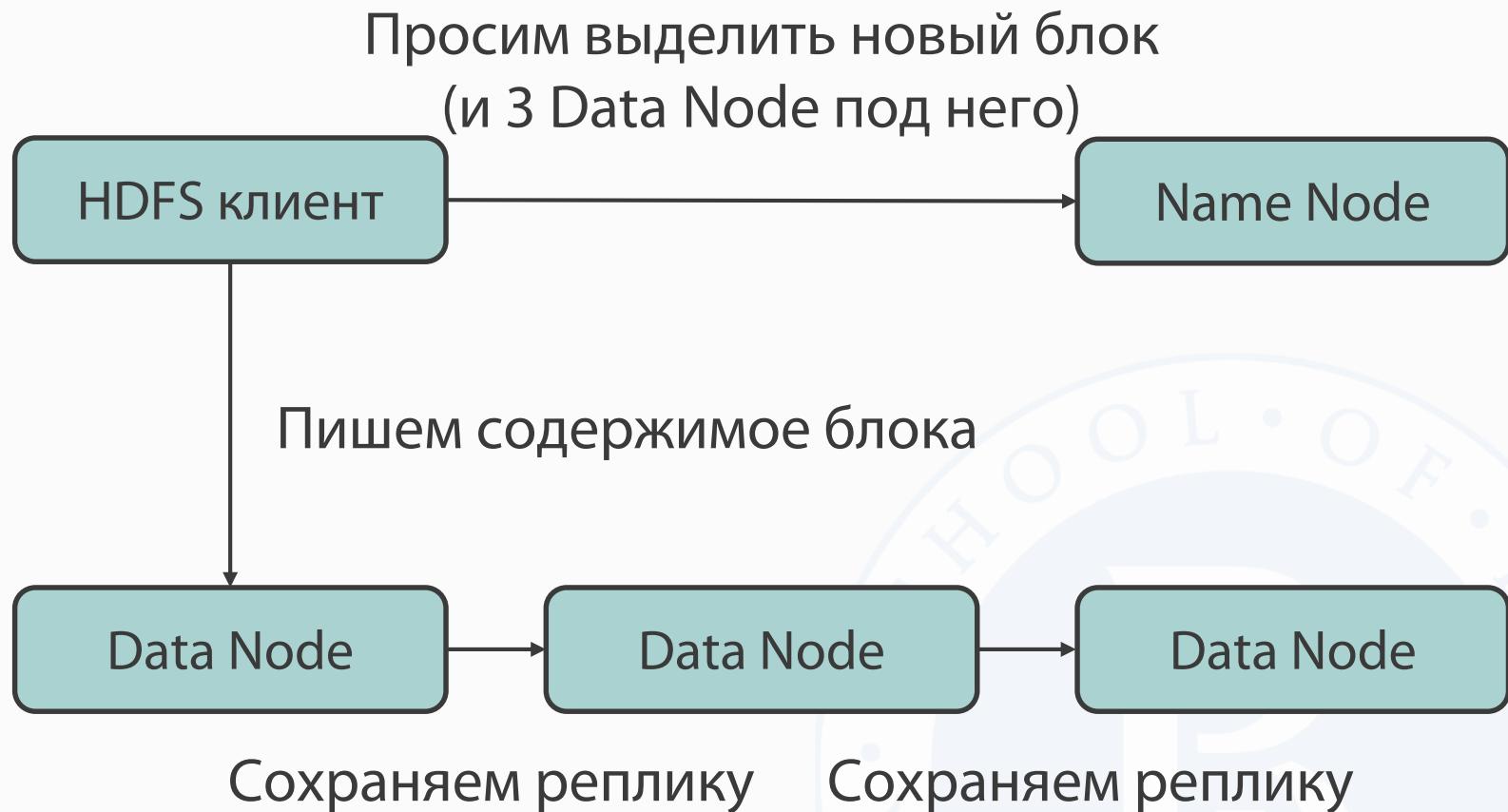


Запись в HDFS

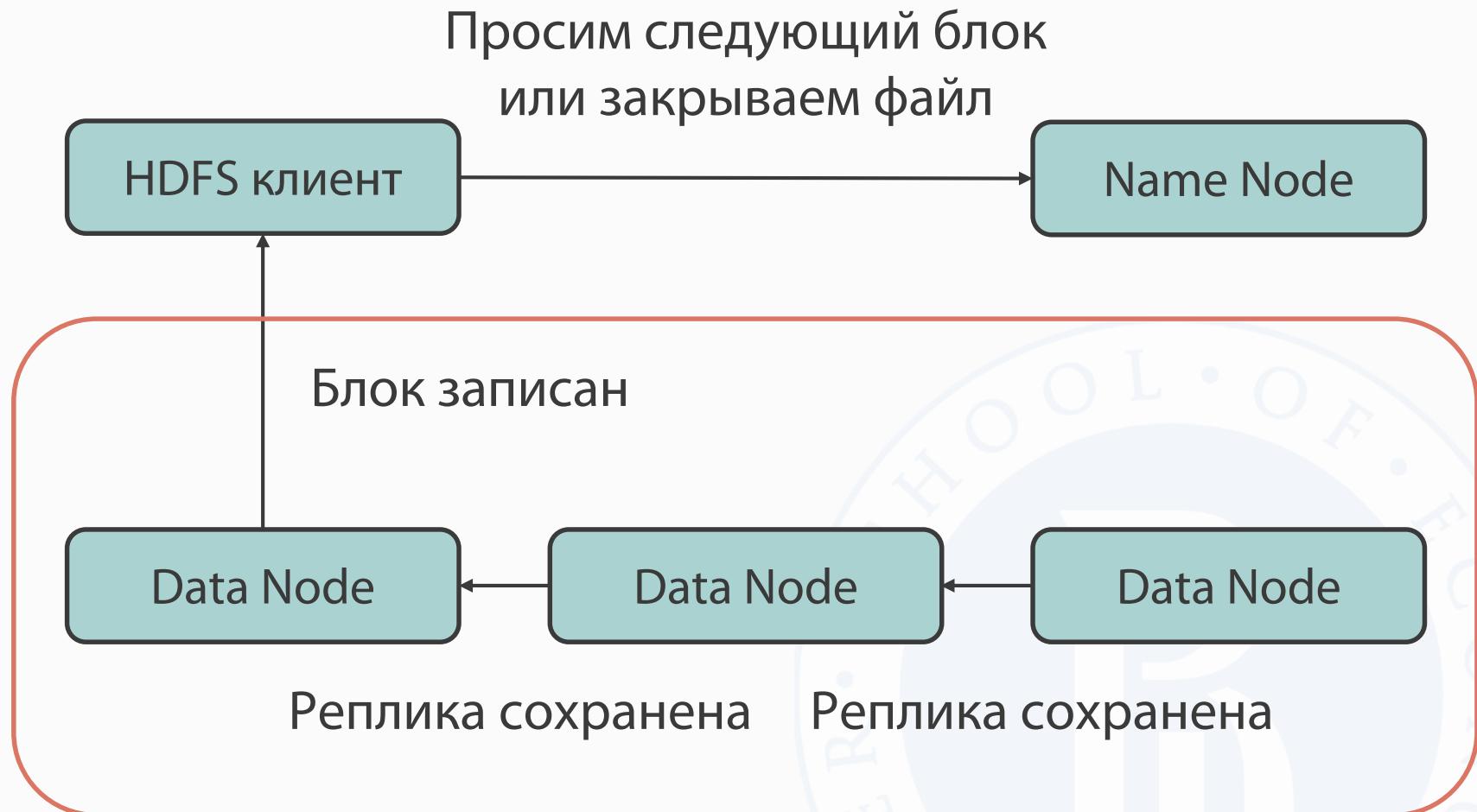
Просим выделить новый блок
(и 3 Data Node под него)



Запись в HDFS



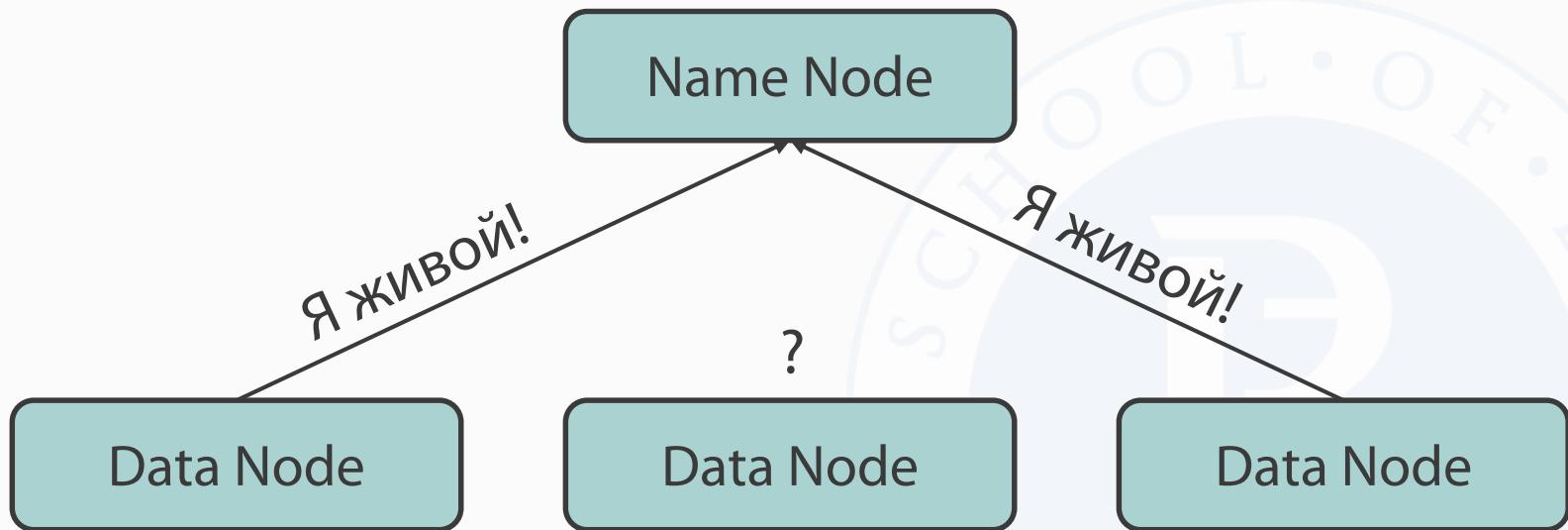
Запись в HDFS



Дождемся ответа асинхронно, можем сразу
начинать писать следующий блок

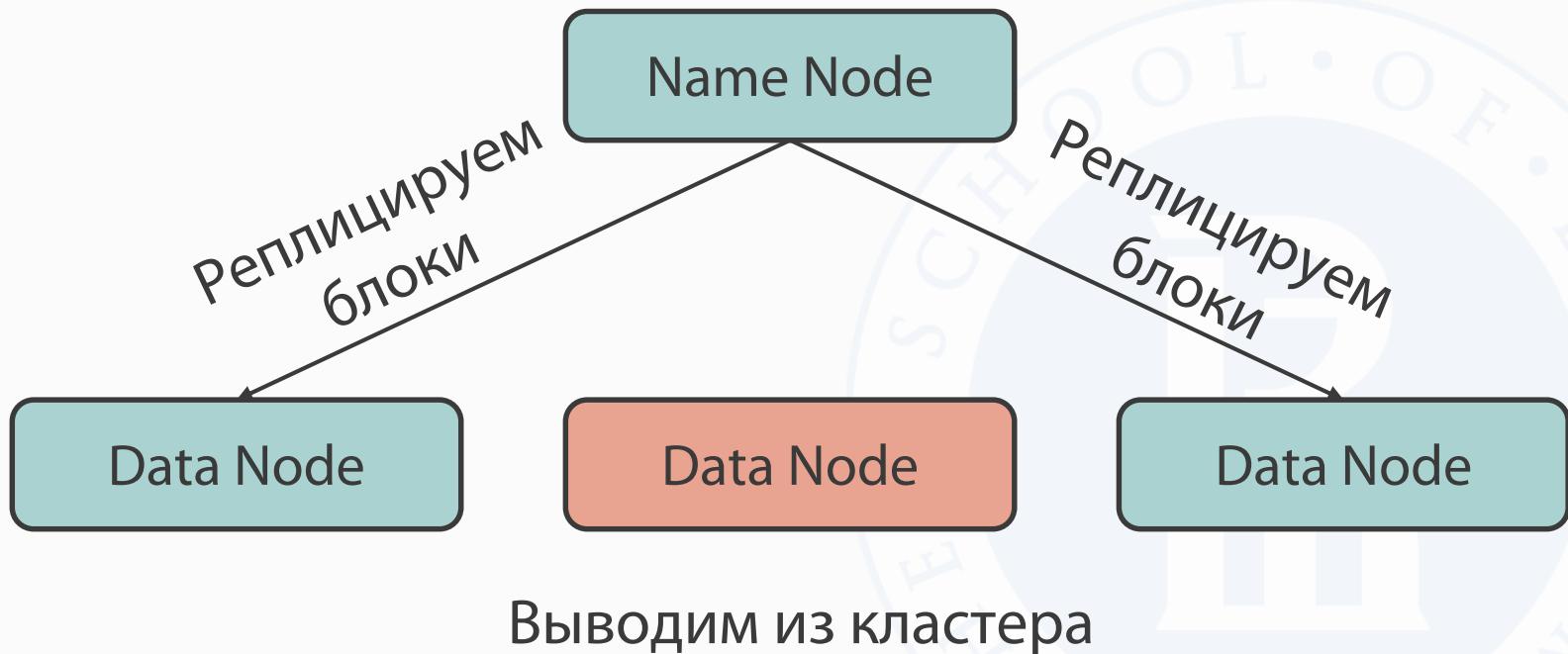
Устойчивость HDFS

→ Data Nodes сообщают Name Node, что они все еще живы (heartbeat). Если такого сообщения не пришло, Name Node может решить вывести машину из кластера (decommission) с репликацией ее блоков на другую машину



Устойчивость HDFS

→ Data Nodes сообщают Name Node, что они все еще живы (heartbeat). Если такого сообщения не пришло, Name Node может решить вывести машину из кластера (decommission) с репликацией ее блоков на другую машину



Резюме



HDFS хранит данные блоками на разных машинах

Резюме



HDFS хранит данные блоками на разных машинах



Для надежности хранения реализована прозрачная
репликация



Резюме



HDFS хранит данные блоками на разных машинах



Для надежности хранения реализована прозрачная
репликация



Далее: рассмотрим первую задачу на больших данных
от Google

Задача Word Count



Свойство локальности данных



Блоки огромного файла хранятся на разных машинах



Свойство локальности данных

- Блоки огромного файла хранятся на разных машинах
- Мы можем обрабатывать блоки **локально** на тех машинах, где они хранятся (быстрое чтение с локального диска)



Свойство локальности данных

- Если в задаче можно обрабатывать блоки независимо, то можно достигнуть идеальной масштабируемости (embarrassingly parallel)

Свойство локальности данных

- Если в задаче можно обрабатывать блоки независимо, то можно достигнуть идеальной масштабируемости (embarrassingly parallel)
- Например, задача фильтрации строк файла идеально масштабируется. Все строчки независимо проверяются предикатом; чем больше машин, тем быстрее будет работать

Задача подсчета частот слов (Google)

- В файле в HDFS сохранены тексты со всего Интернета

Задача подсчета частот слов (Google)

- В файле в HDFS сохранены тексты со всего Интернета
- Нам интересно узнать частоты всех слов
(счетчики для tf-idf)

Задача подсчета частот слов (Google)

- В файле в HDFS сохранены тексты со всего Интернета
- Нам интересно узнать частоты всех слов
(счетчики для tf-idf)
- ? Как будем решать?

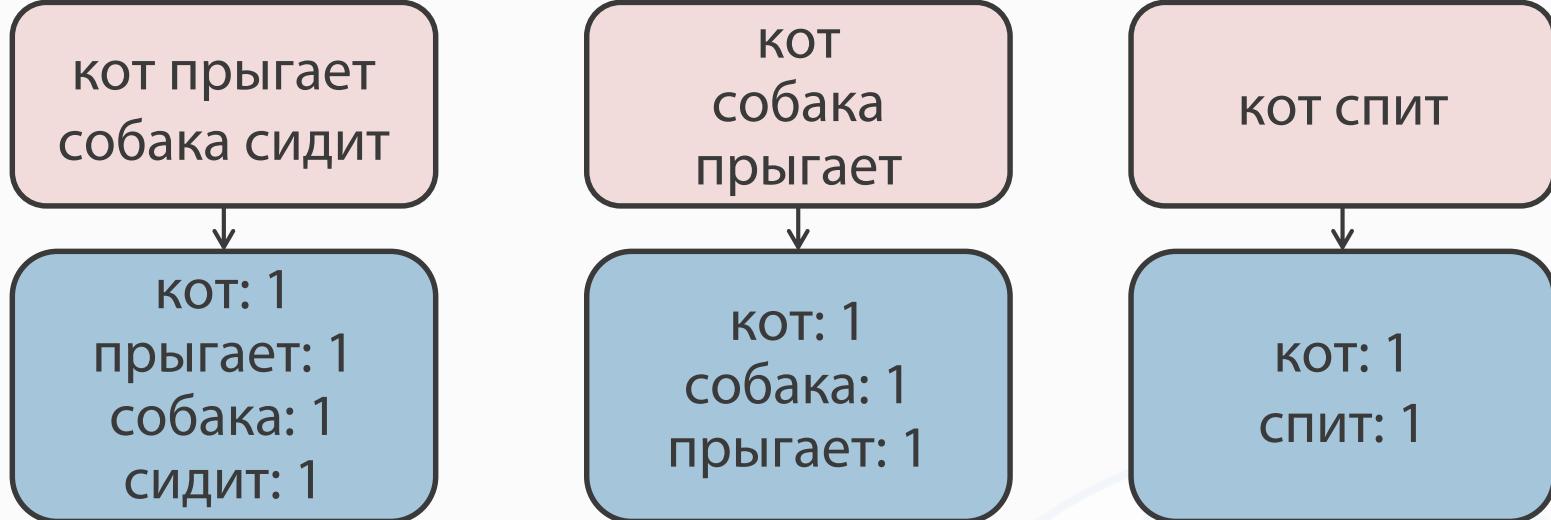
Задача подсчета частот слов (Google)

- В файле в HDFS сохранены тексты со всего Интернета
- Нам интересно узнать частоты всех слов
(счетчики для tf-idf)
- ❓ Как будем решать?
- ✓ Для каждого блока посчитаем частоты слов
в нем (идеально масштабируется)

Задача подсчета частот слов (Google)

- В файле в HDFS сохранены тексты со всего Интернета
- Нам интересно узнать частоты всех слов
(счетчики для tf-idf)
- ❓ Как будем решать?
- ✓ Для каждого блока посчитаем частоты слов
в нем (идеально масштабируется)
- ✓ Сложим частоты по всем блокам

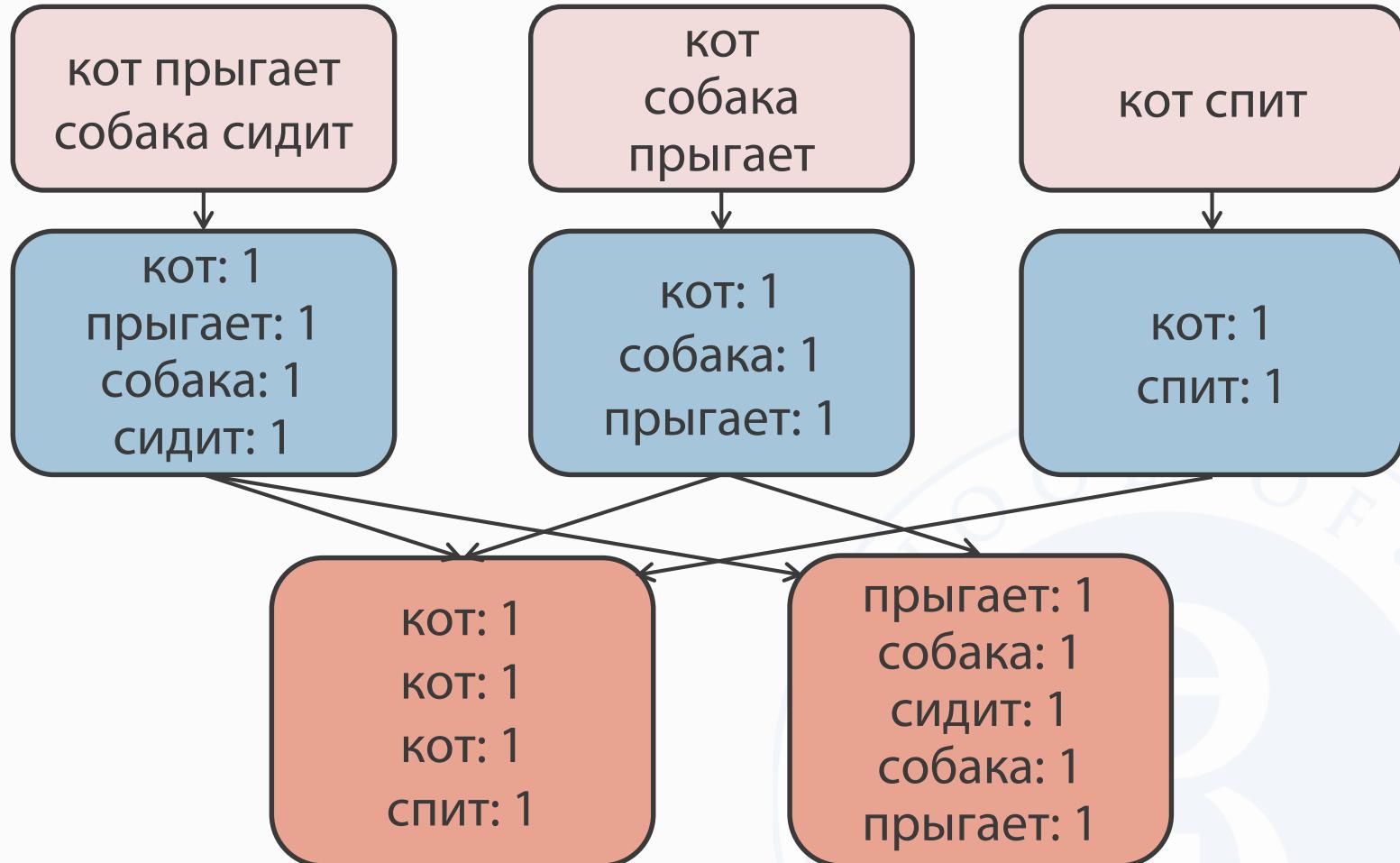
Независимо обрабатываем половинки



Перешлем все
на одну машину
по сети и посчитаем
на ней сумму
по каждому слову

Не масштабируется!
Надо придумать,
как распараллелить!

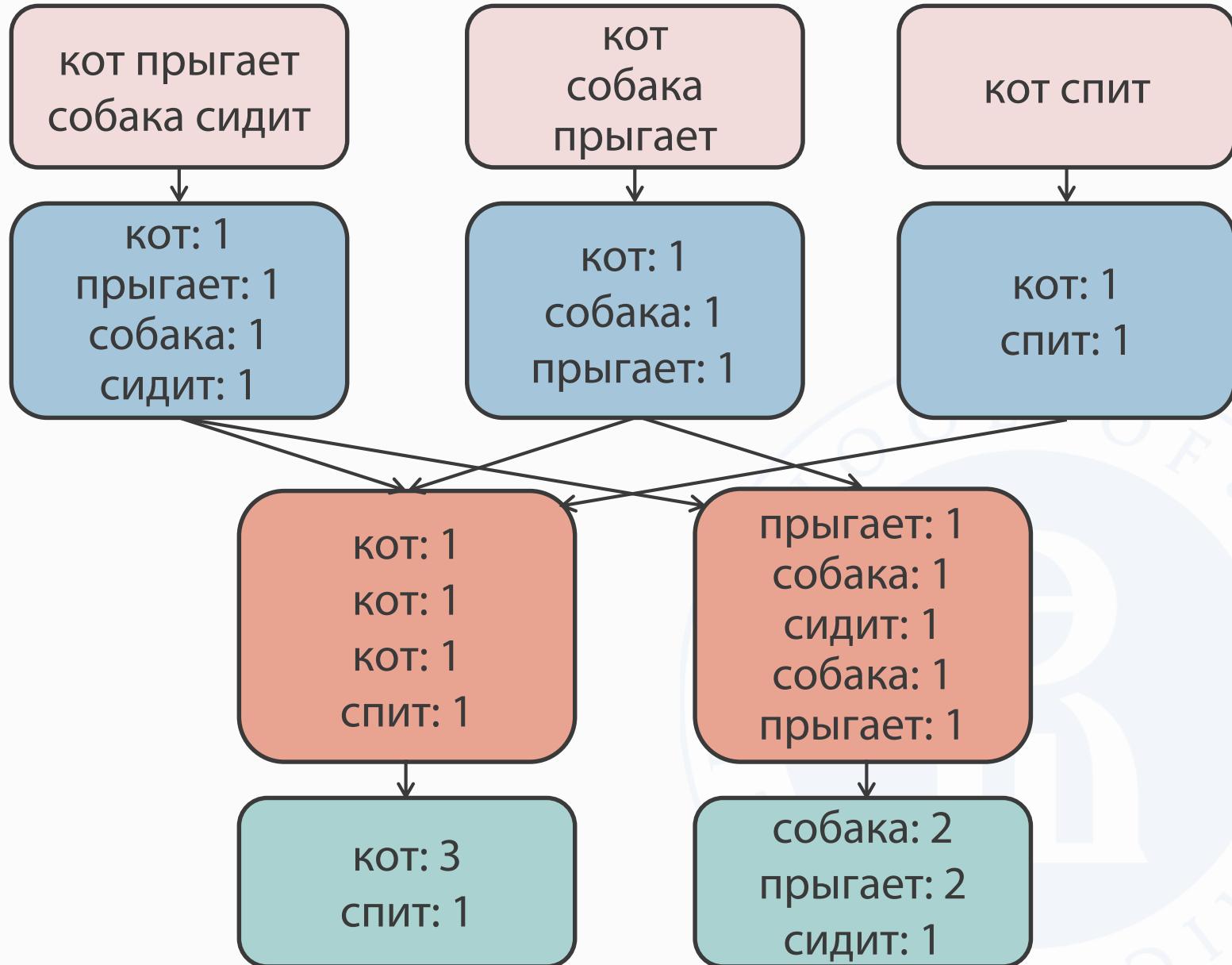
Независимо обрабатываем половинки



Слова длины < 5

Слова длины ≥ 5

Независимо обрабатываем половинки



Резюме



Посчитали частоты в каждом блоке



Резюме



Посчитали частоты в каждом блоке



Поделили дальнейшую работу по длине слова (≥ 5 , < 5)

Резюме



Посчитали частоты в каждом блоке



Поделили дальнейшую работу по длине слова (≥ 5 , < 5)



В каждой части сложили полученные частоты

Резюме



Посчитали частоты в каждом блоке



Поделили дальнейшую работу по длине слова ($\geq 5, < 5$)



В каждой части сложили полученные частоты



Далее: обобщим эту схему до MapReduce

MapReduce



Обобщим условие разделения работы



Предположим, мы хотим поделить работу на N частей

Обобщим условие разделения работы

- Предположим, мы хотим поделить работу на N частей
- Работу будем делить по значениям $\text{hash}(\text{word}) \% N$,
которое показывает, на какую машину отправить слово

Обобщим условие разделения работы

- Предположим, мы хотим поделить работу на N частей
- Работу будем делить по значениям $\text{hash}(\text{word}) \% N$, которое показывает, на какую машину отправить слово
- Для хорошей хэш-функции (hash) все значения равновероятны

Обобщим условие разделения работы

- Предположим, мы хотим поделить работу на N частей
- Работу будем делить по значениям $\text{hash}(\text{word}) \% N$, которое показывает, на какую машину отправить слово
- Для хорошей хэш-функции (hash) все значения равновероятны
- Пример хэш-функции (полиномиальной):

$$\text{hash}(s) = s[0] + s[1]p^1 + \dots + s[n]p^n$$

s – строка

p – фиксированное простое число

$s[i]$ – код символа

Независимо обрабатываем половинки

Map

кот прыгает
собака сидит

кот
собака
прыгает

кот спит

кот: 1
прыгает: 1
собака: 1
сидит: 1

кот: 1
собака: 1
прыгает: 1

кот: 1
спит: 1

Shuffle

$\text{hash}(\text{word}) = 0$

кот: 1
кот: 1
кот: 1
спит: 1

прыгает: 1
собака: 1
сидит: 1
собака: 1
прыгает: 1

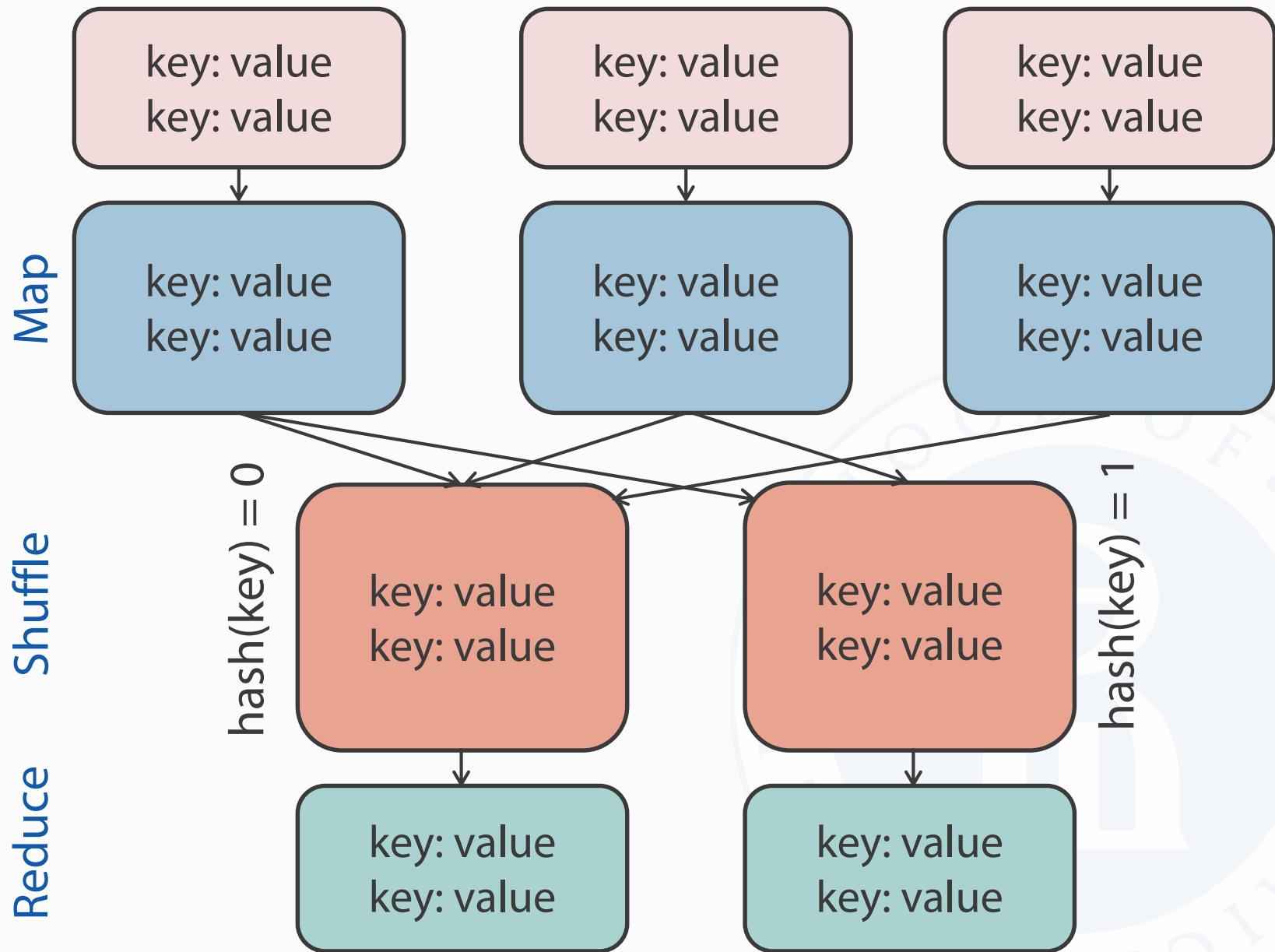
$\text{hash}(\text{word}) = 1$

собака: 2
прыгает: 2
сидит: 1

Reduce

кот: 3
спит: 1

Независимо обрабатываем половинки



Парадигма MapReduce



Map:

$(K1, V1) \rightarrow \text{List}(K2, V2)$

(номер строки, «кот спит») $\rightarrow [(\text{«кот»}, 1), (\text{«спит»}, 1)]$

Парадигма MapReduce

→ Map:

$(K_1, V_1) \rightarrow \text{List}(K_2, V_2)$

(номер строки, «кот спит») → [(«кот», 1), («спит», 1)]

→ Shuffle:

Ключи делятся по $\text{hash}(\text{key}) \% N$ на N частей

Каждая часть сортируется по key (независимо)

Парадигма MapReduce

→ Map:

$(K1, V1) \rightarrow \text{List}(K2, V2)$

(номер строки, «кот спит») $\rightarrow [(\text{«кот»}, 1), (\text{«спит»}, 1)]$

→ Shuffle:

Ключи делятся по $\text{hash}(\text{key}) \% N$ на N частей

Каждая часть сортируется по key (независимо)

→ Reduce:

$(K2, \text{List}(V2)) \rightarrow \text{List}(K3, V3)$

(«кот», (1, 1, 1)) $\rightarrow [(\text{«кот»}, 3)]$

Резюме



Работаем с парами (ключ, значение)



Резюме



Работаем с парами (ключ, значение)



Делить работу после шага Map можем на произвольное
число частей по хэшу от ключа

Резюме



Работаем с парами (ключ, значение)



Делить работу после шага Map можем на произвольное
число частей по хэшу от ключа



Удивительно, но довольно много вычислений ложатся
на эту простую схему

Резюме



Работаем с парами (ключ, значение)



Делить работу после шага Мар можем на произвольное число частей по хэшу от ключа



Удивительно, но довольно много вычислений ложатся на эту простую схему

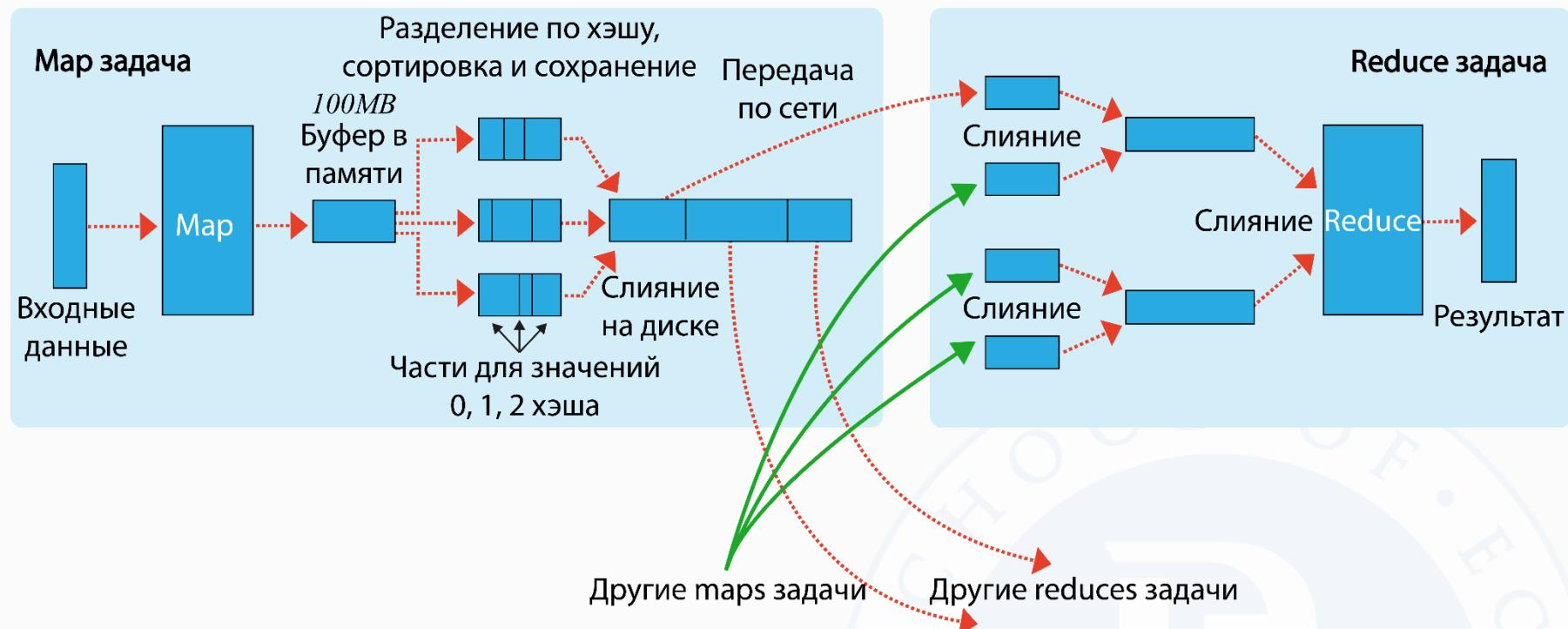


Далее: поговорим о деталях Hadoop MapReduce

Hadoop MapReduce

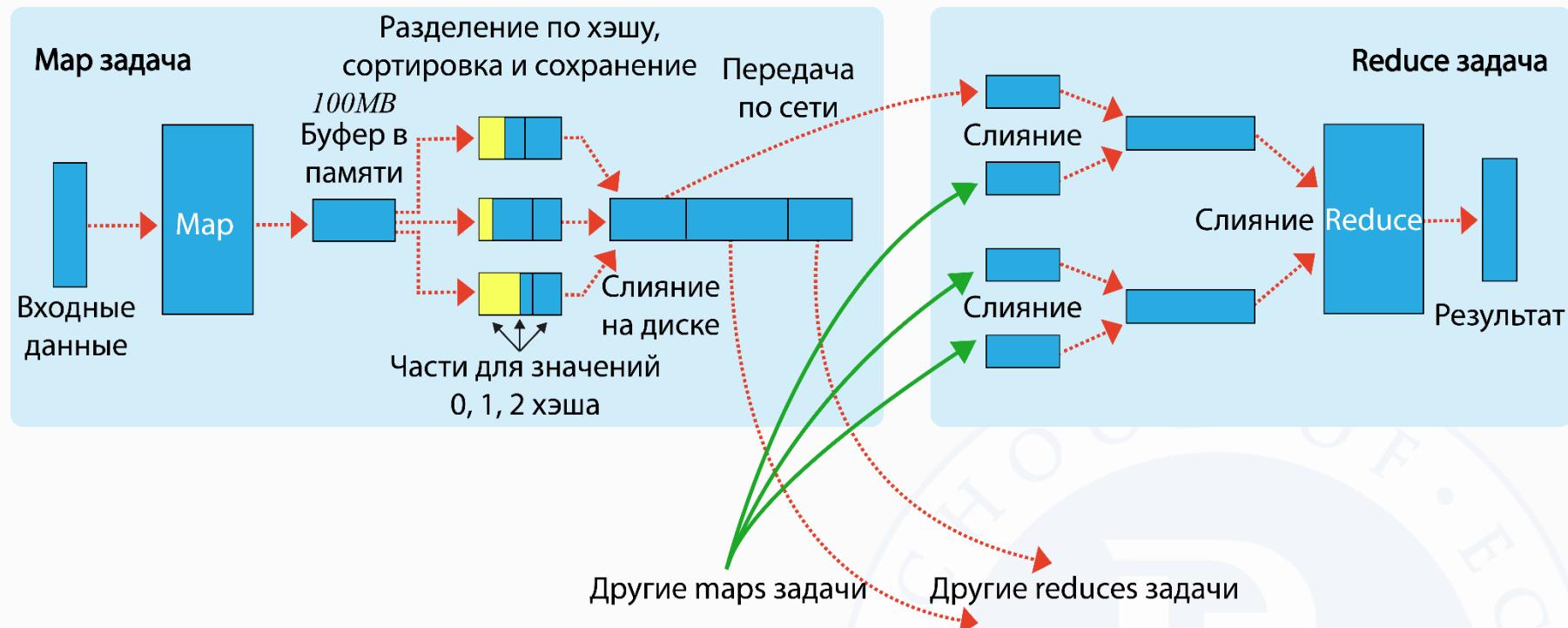


Сортировку на шаге Shuffle можно ускорить



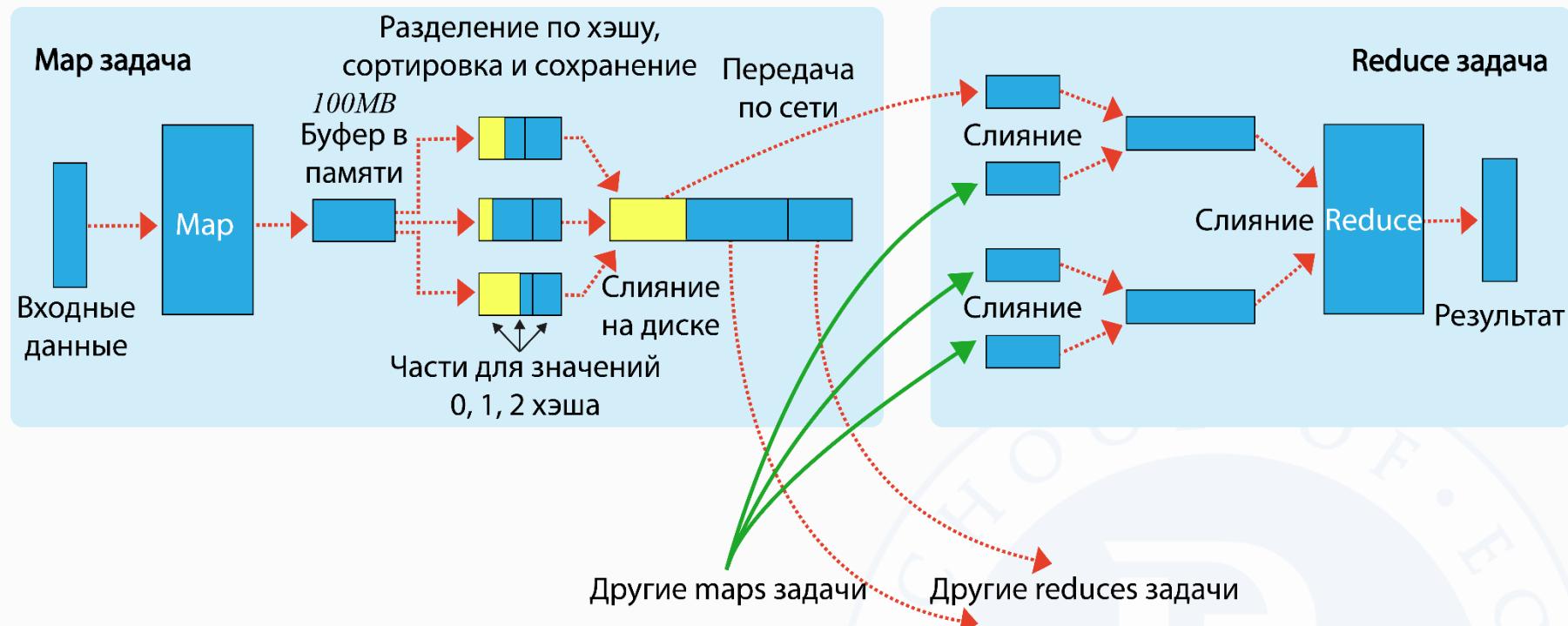
- Результат Мар задача можно отсортировать локально
- После можно использовать сортировку слиянием (merge sort) на шаге Shuffle за линейное время

Сортировку на шаге Shuffle можно ускорить



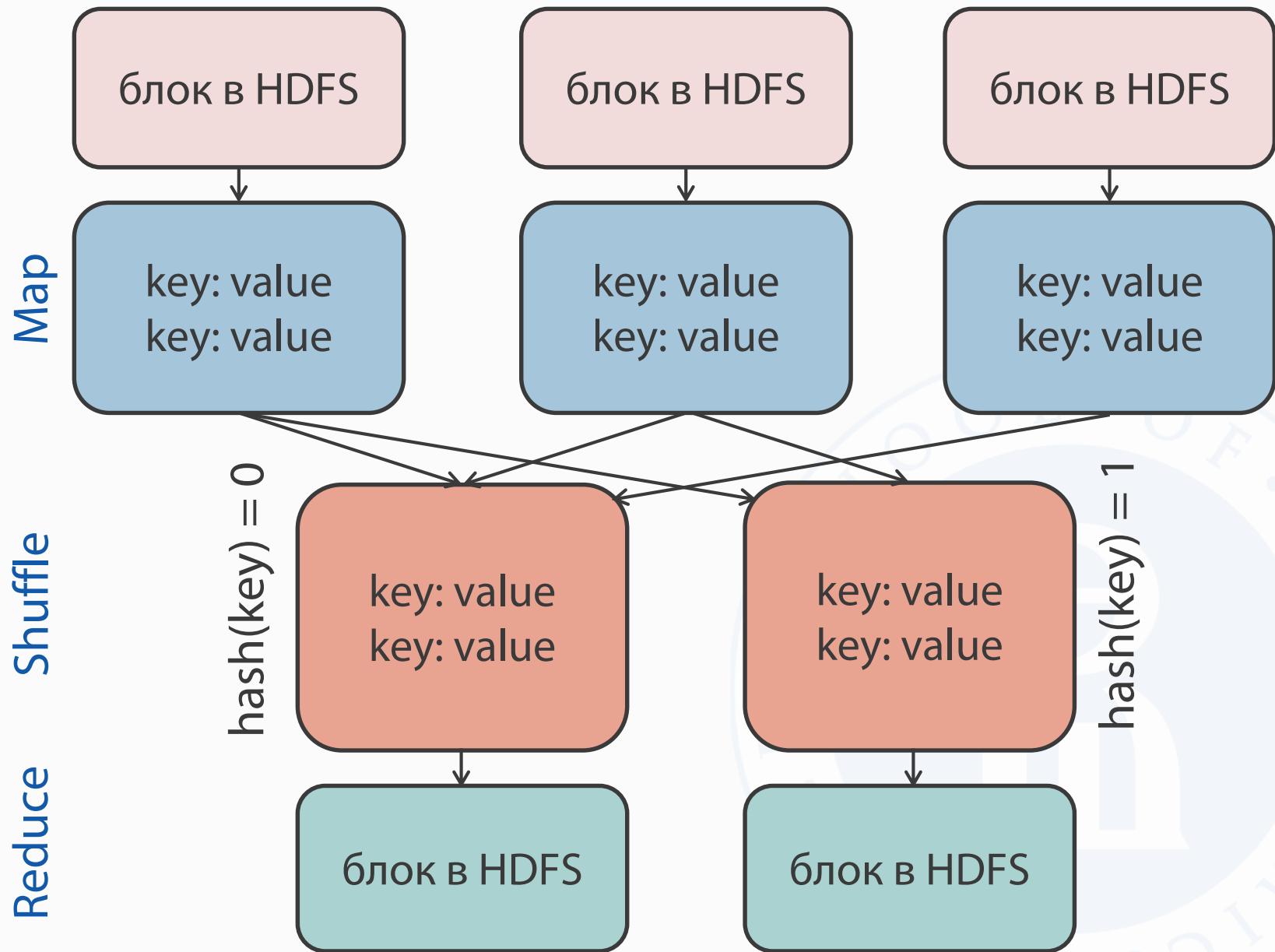
- Результат Мар задача можно отсортировать локально
- После можно использовать сортировку слиянием (merge sort) на шаге Shuffle за линейное время

Сортировку на шаге Shuffle можно ускорить



- Результат Map шага можно отсортировать локально
- После можно использовать сортировку слиянием (merge sort) на шаге Shuffle за линейное время

Независимо обрабатываем половинки



Надежность работы

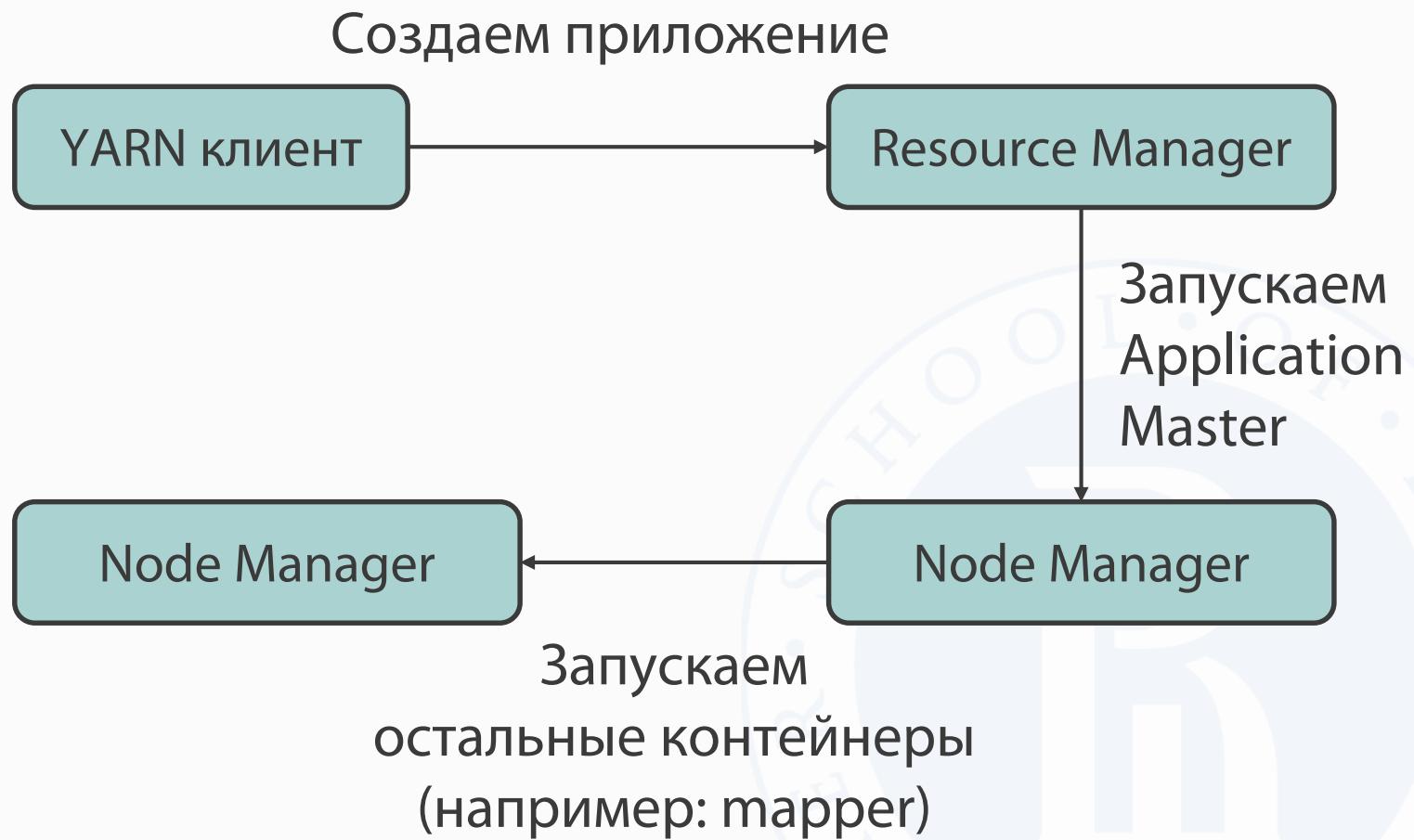


При потере маппера можем перезапустить задачу
только для его блоков

Надежность работы

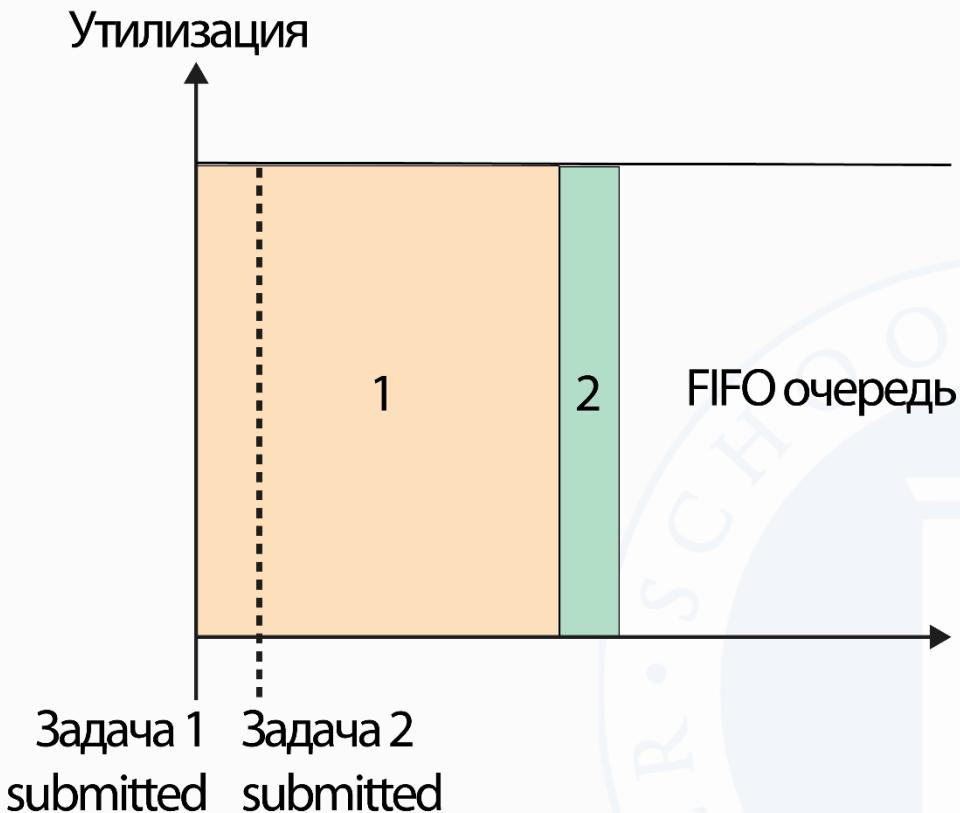
- При потере маппера можем перезапустить задачу только для его блоков
- При потере редьюсера заново собираем данные со всех мапперов только для его значения хэша

Работа с YARN



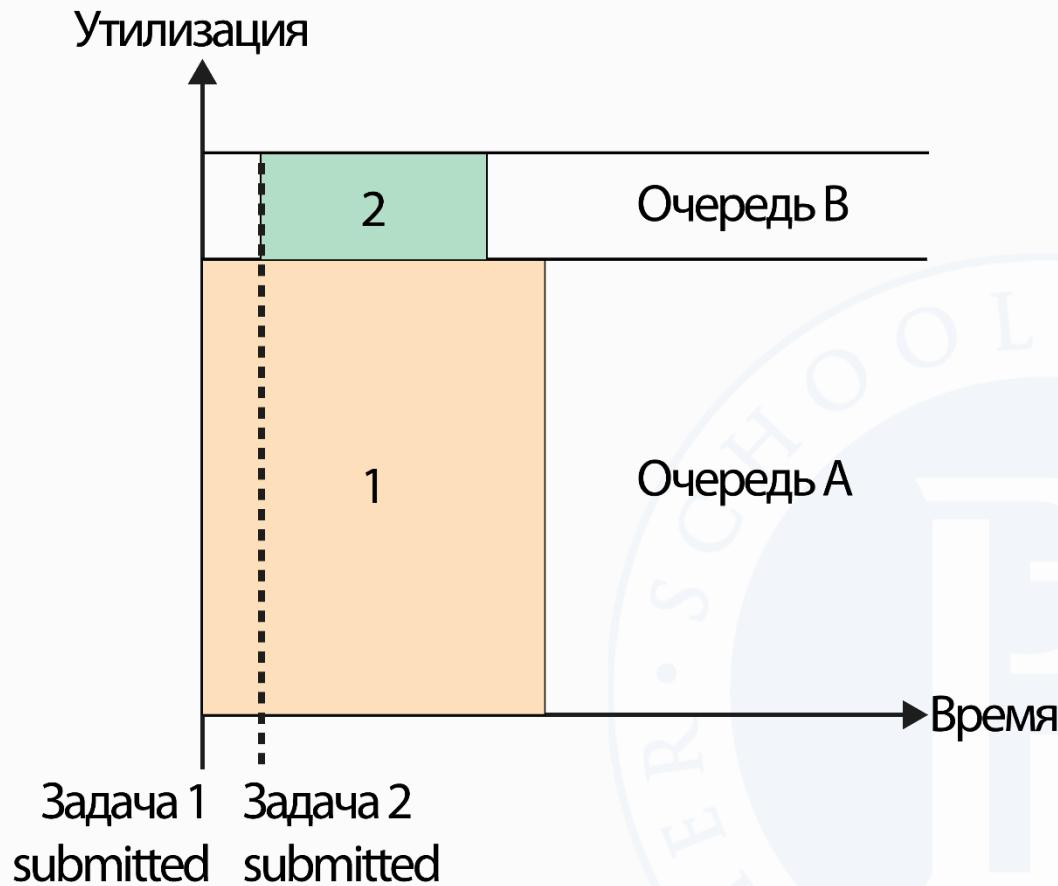
Планировщик в YARN

i. Планировщик FIFO



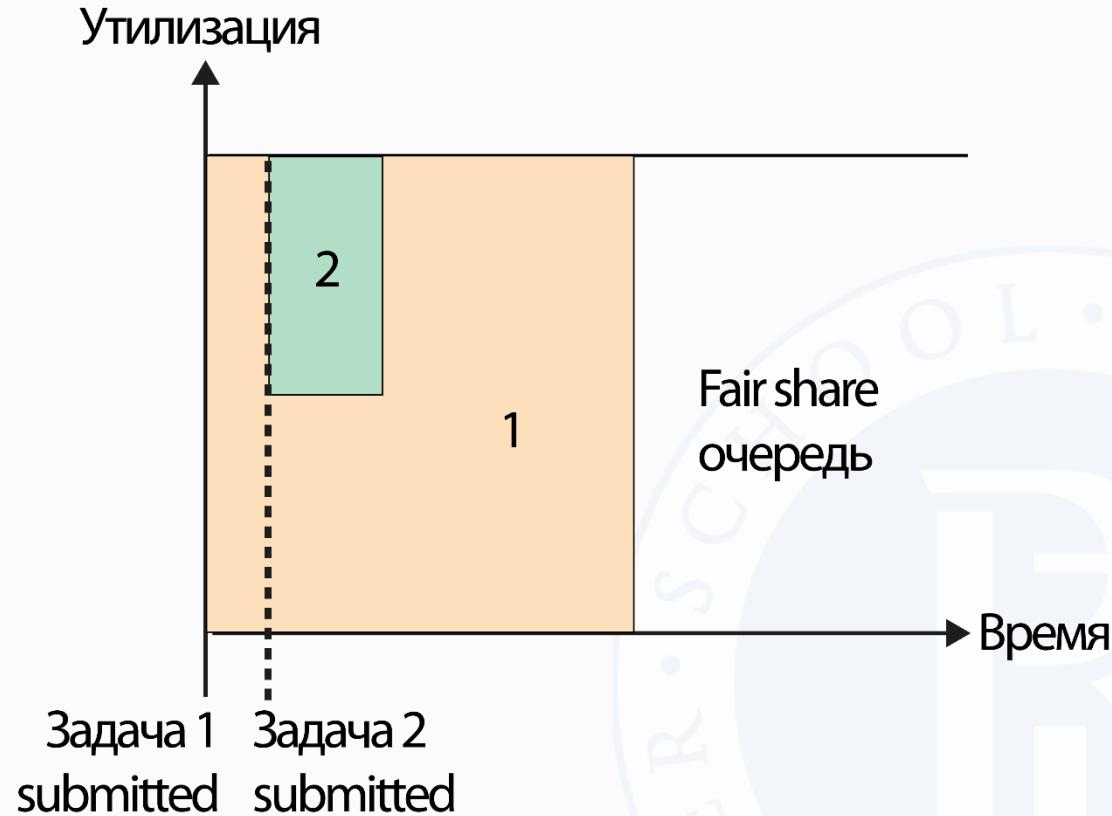
Планировщик в YARN

ii. Планировщик Capacity



Планировщик в YARN

iii. Планировщик Fair



Резюме



Сортировка данных в Hadoop очень эффективная

Резюме



Сортировка данных в Hadoop очень эффективная



Схема MapReduce устойчива к умирающим машинам

Резюме



Сортировка данных в Hadoop очень эффективная



Схема MapReduce устойчива к умирающим машинам



MapReduce работает поверх YARN

Резюме



Сортировка данных в Hadoop очень эффективная



Схема MapReduce устойчива к умирающим машинам



MapReduce работает поверх YARN



Далее: разберем пример реальной задачи

Пример реальной задачи



MapReduce пример

→ Map: $(K_1, V_1) \rightarrow \text{List}(K_2, V_2)$

→ Reduce: $(K_2, \text{List}(V_2)) \rightarrow \text{List}(K_3, V_3)$

UserId	TrackId	AlbumId
11123	4521	842
14322	3593	957
...

→ Возьмем логи прослушиваний в Spotify.

Хотим для каждого альбома найти самый популярный трек

MapReduce пример

→ Map: $(K_1, V_1) \rightarrow \text{List}(K_2, V_2)$

→ Reduce: $(K_2, \text{List}(V_2)) \rightarrow \text{List}(K_3, V_3)$

UserId	TrackId	AlbumId
11123	4521	842
14322	3593	957
...

→ Возьмем логи прослушиваний в Spotify.

Хотим для каждого альбома найти самый популярный трек

M: #, (user, track, album) → **(album, track)**, 1

R: **(album, track)**, (1,1,...) → (album, track), count

MapReduce пример

→ Map: $(K_1, V_1) \rightarrow \text{List}(K_2, V_2)$

→ Reduce: $(K_2, \text{List}(V_2)) \rightarrow \text{List}(K_3, V_3)$

UserId	TrackId	AlbumId
11123	4521	842
14322	3593	957
...

→ Возьмем логи прослушиваний в Spotify.

Хотим для каждого альбома найти самый популярный трек

M: #, (user, track, album) → **(album, track)**, 1

R: **(album, track)**, (1,1,...) → (album, track), count

M: (album, track), count → **album**, (track, count)

R: **album**, tracks → album, most popular track

Заключение



Hadoop создан для обработки больших данных и горизонтально масштабируется

Заключение



Hadoop создан для обработки больших данных и горизонтально масштабируется



HDFS — распределенная и надежная файловая система



Заключение



Hadoop создан для обработки больших данных и горизонтально масштабируется



HDFS — распределенная и надежная файловая система



MapReduce — распределенный и надежный способ обработки больших данных в HDFS