

LSML #1

Hadoop HDFS, MapReduce, YARN

Важное

$$O_{\text{итоговая}} = 0.125 * (O_{\text{мд31}} + O_{\text{мд32}} + O_{\text{мд33}} + O_{\text{мд34}}) + 0.5 * O_{\text{дз}}$$

[http://wiki.cs.hse.ru/Машинное обучение на больших данных 2021](http://wiki.cs.hse.ru/Машинное_обучение_на_больших_данных_2021)

350\$ в Azure, страхового запаса почти нет

Темы

1. Hadoop HDFS, MapReduce, YARN
2. Apache Spark
3. Онлайн-обучение и линейные модели
4. Градиентный бустинг для больших данных
5. Рекомендательные системы
6. Распараллеливание обучения нейросетей
7. Ускорение работы нейросетей
8. Трюки с хэшированием (bloom filter, min sketch, etc.)
9. ???



Большие данные (Big Data)

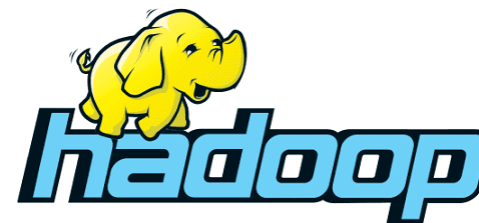
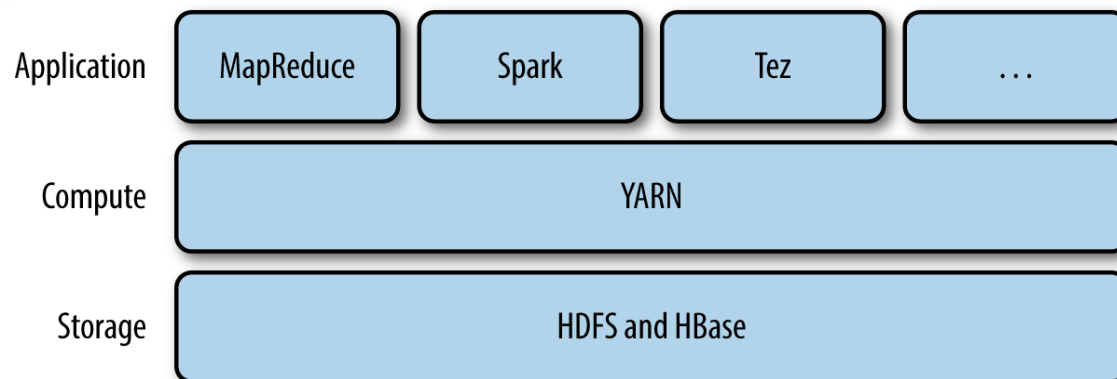
Большие данные отличаются 3 важными свойствами (**3V**):

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 дополнительной 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 структурированы, 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
<i>Количество серверов</i>	Один мощный	Много средних
<i>Объем данных</i>	Маленький	Большой
<i>Скорость запросов</i>	Моментальный ответ	Ответ с задержкой
<i>Формат данных</i>	Таблицы (структурированные)	Файлы (неструктурированные)
<i>Требования ACID</i>	Выполняются	Не требуются

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)

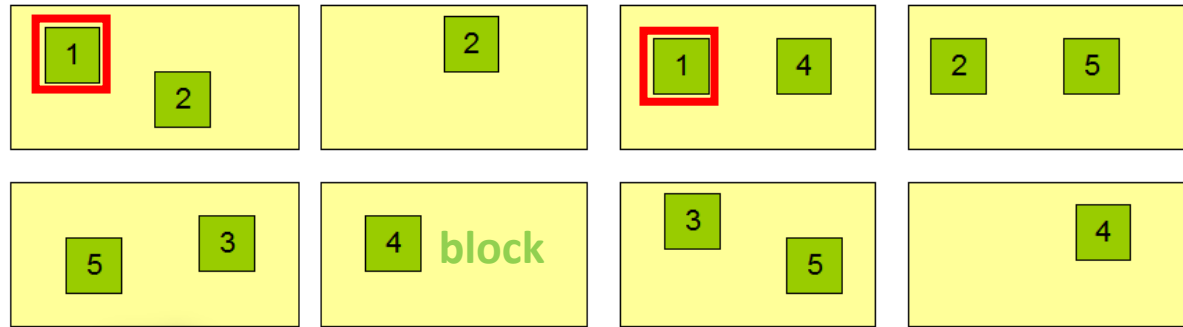
Соответствие **имя файла** → **блоки** хранится в памяти специальной машины (**Name Node**)

HDFS

Block Replication

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



node

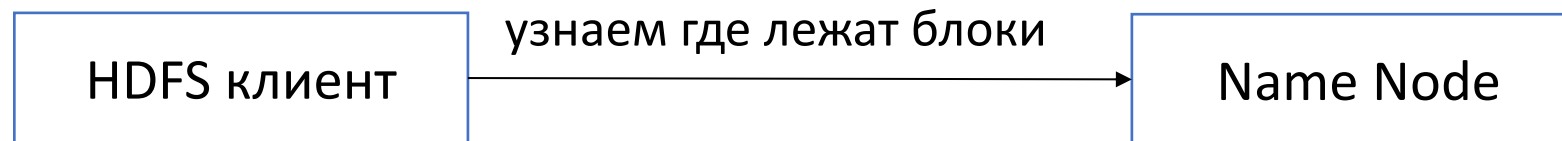
Зачем нужна репликация блоков

- 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



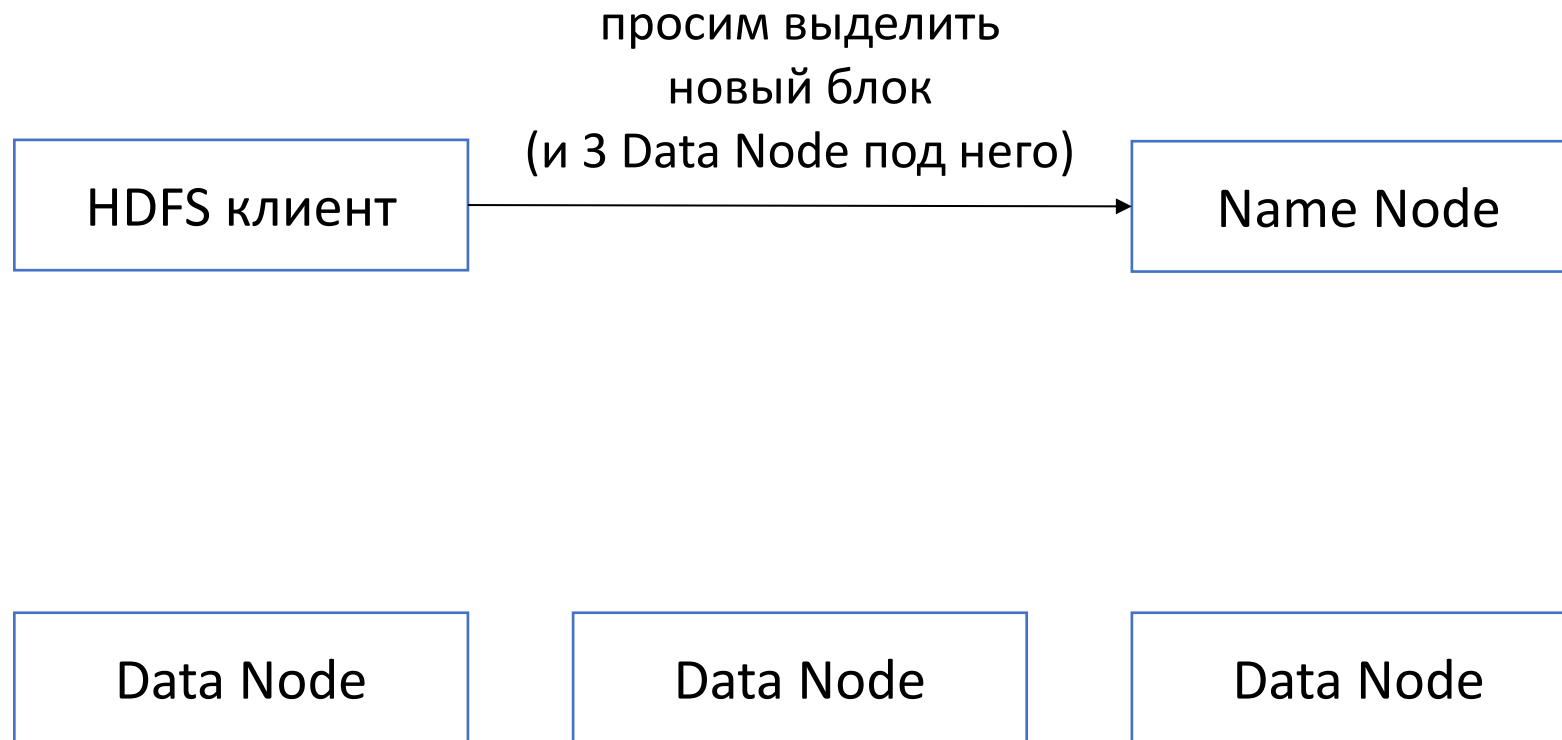
Чтение из HDFS



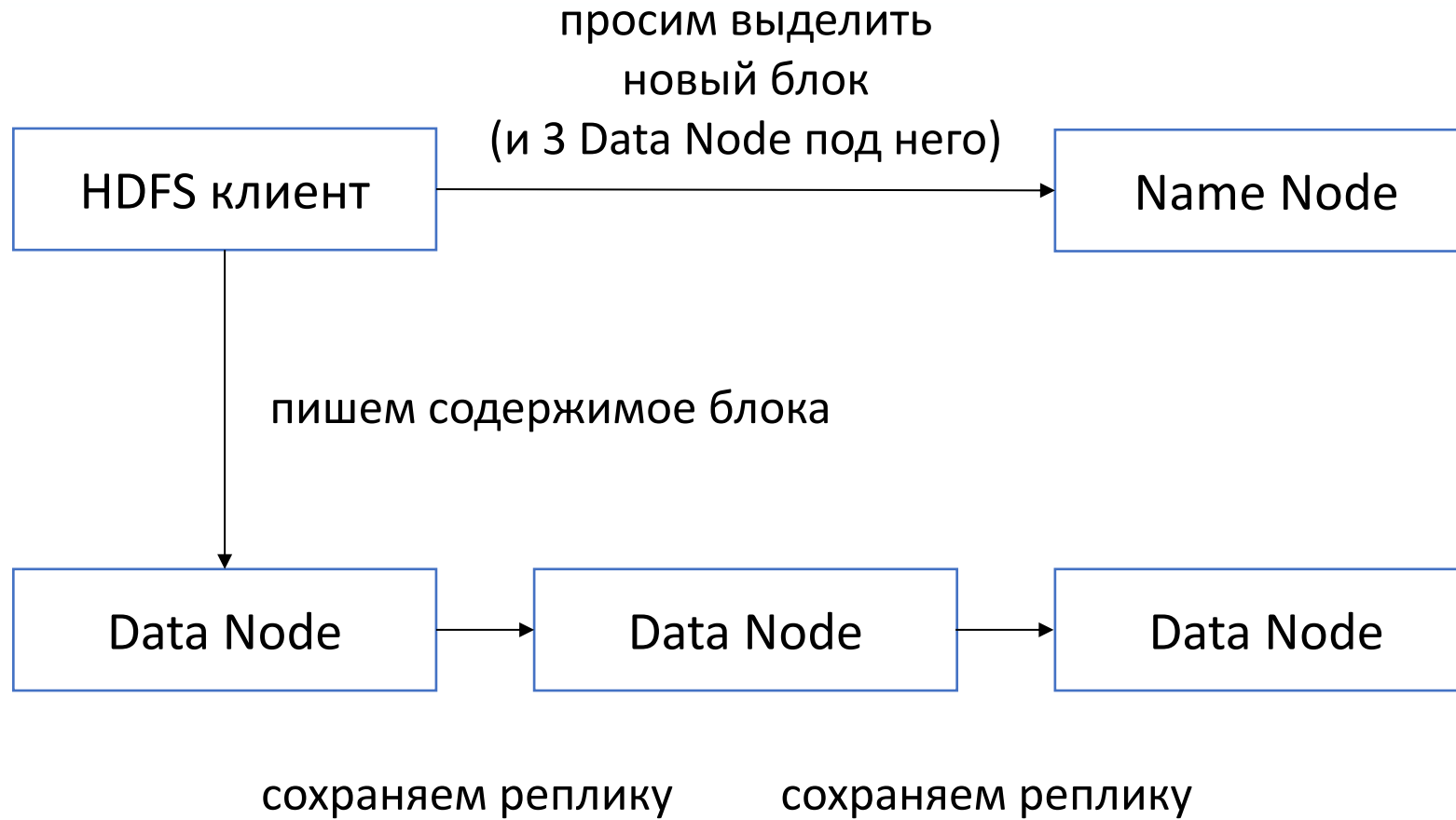
Запись в HDFS



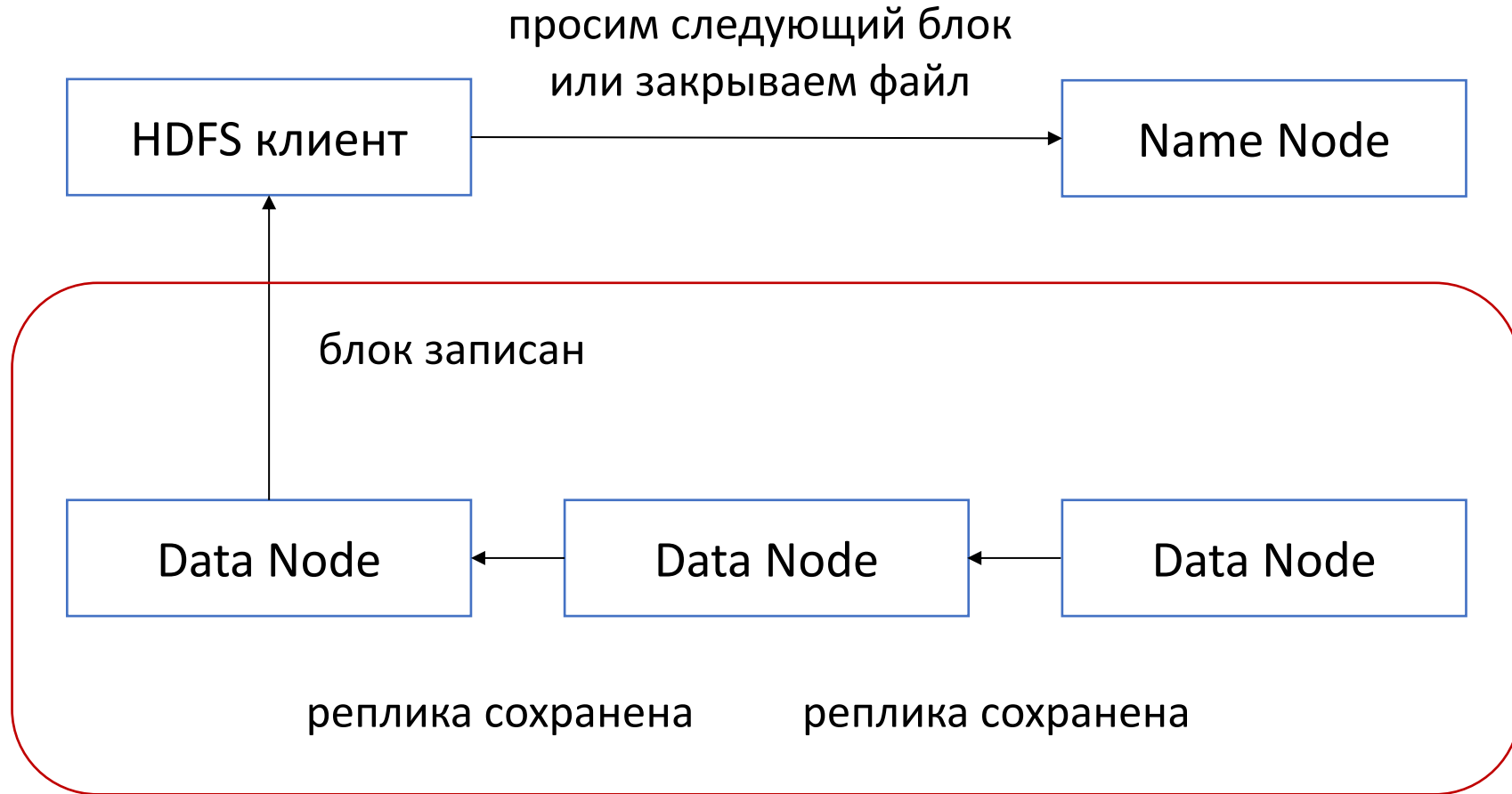
Запись в HDFS



Запись в HDFS



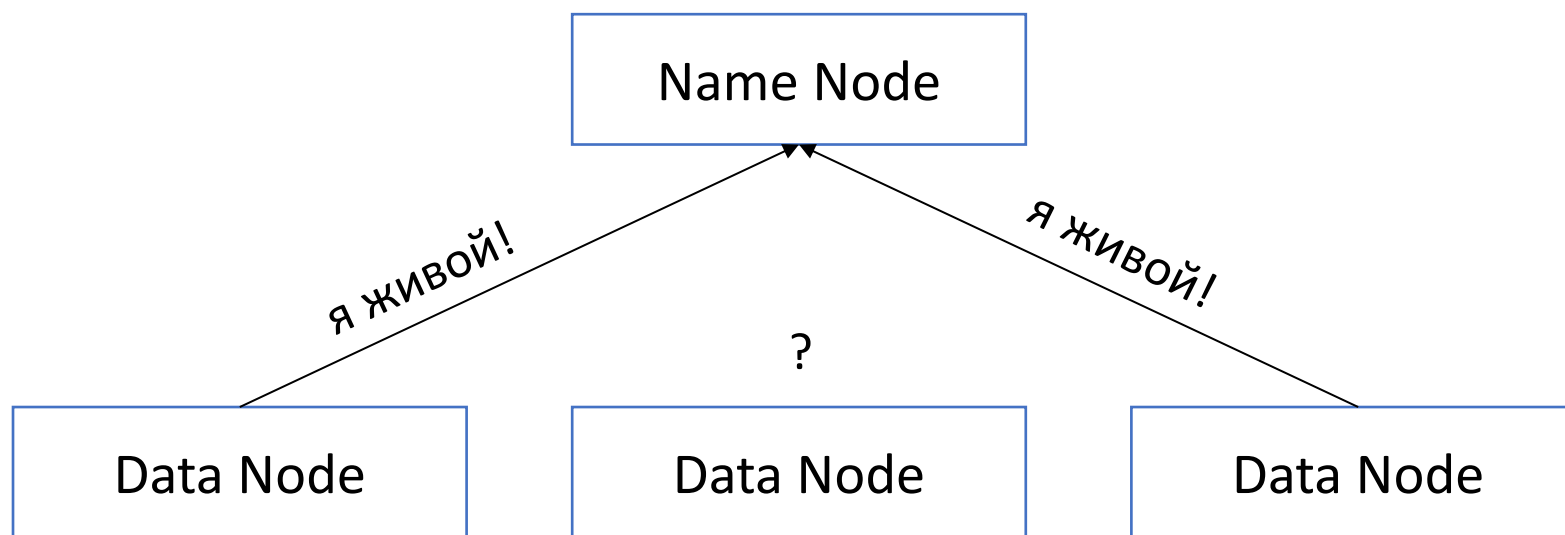
Запись в HDFS



ждемся ответа асинхронно, можем сразу
начинать писать следующий блок

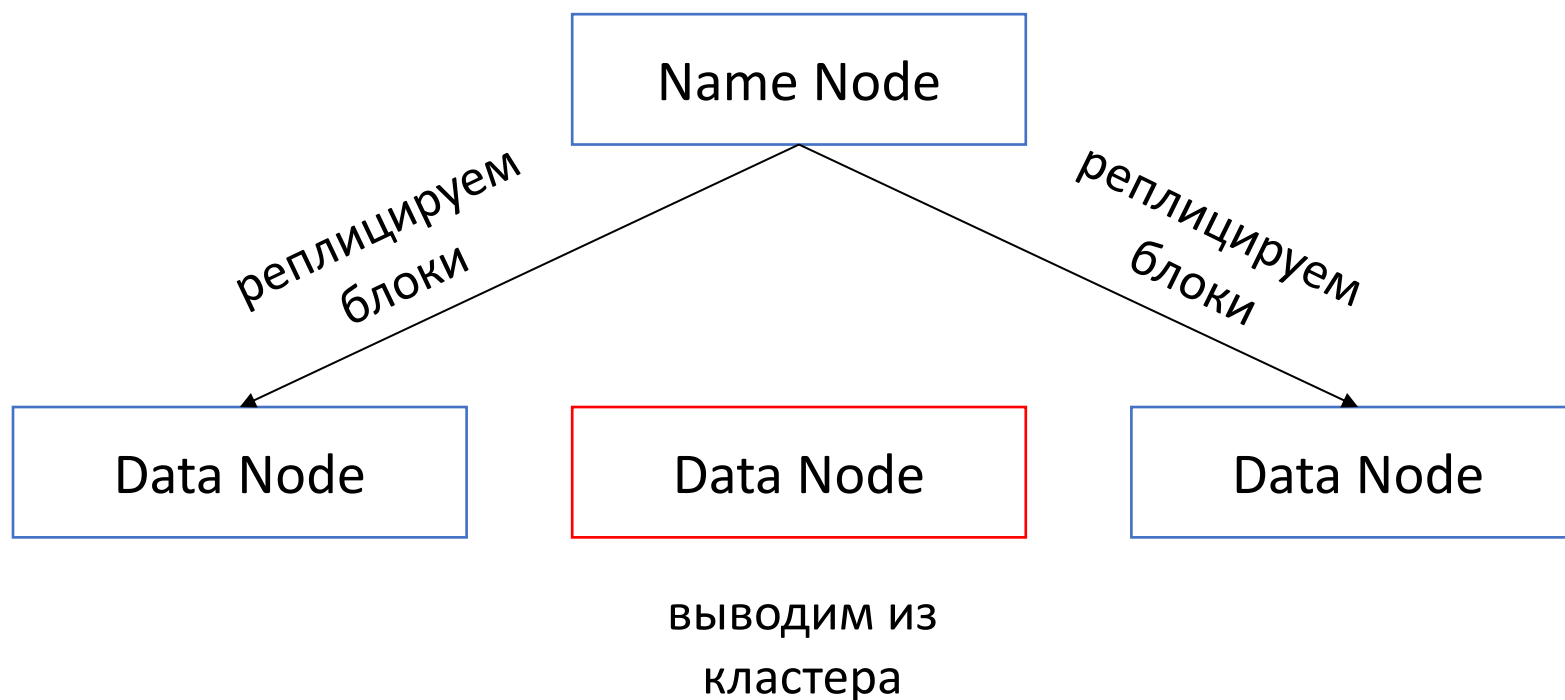
Устойчивость HDFS

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.



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).

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

Например, задача фильтрации строк файла идеально масштабируется. Все строки независимо проверяются предикатом, чем больше машин, тем быстрее будет работать.

Задача подсчета частот слов (Google)

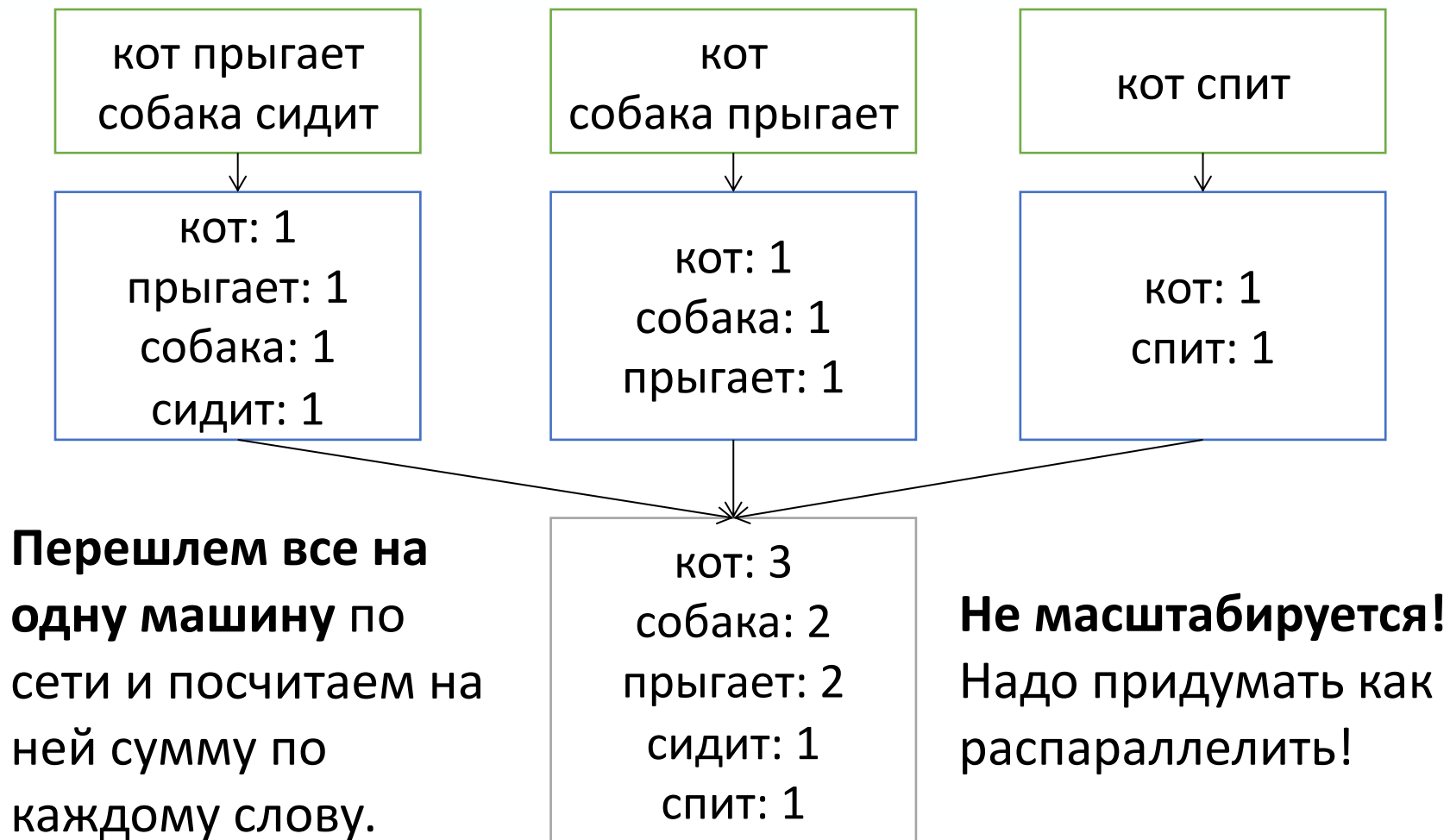
В файле в HDFS сохранены тексты со всего Интернета

Нам интересно узнать частоты всех слов (счетчики для tf-idf)

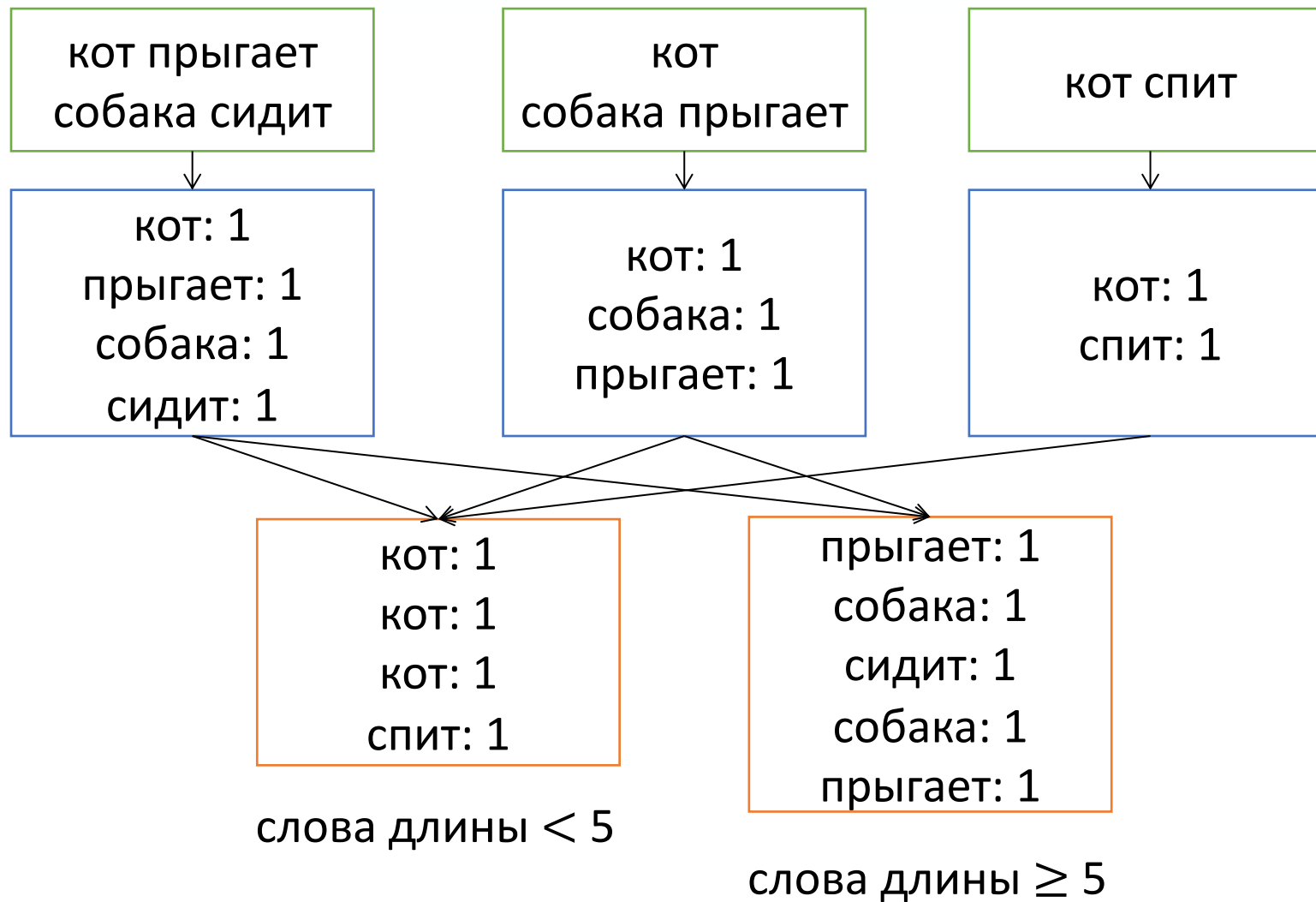
Как будем решать:

- для каждого блока посчитаем частоты слов в нем (идеально масштабируется)
- сложим частоты по всем блокам

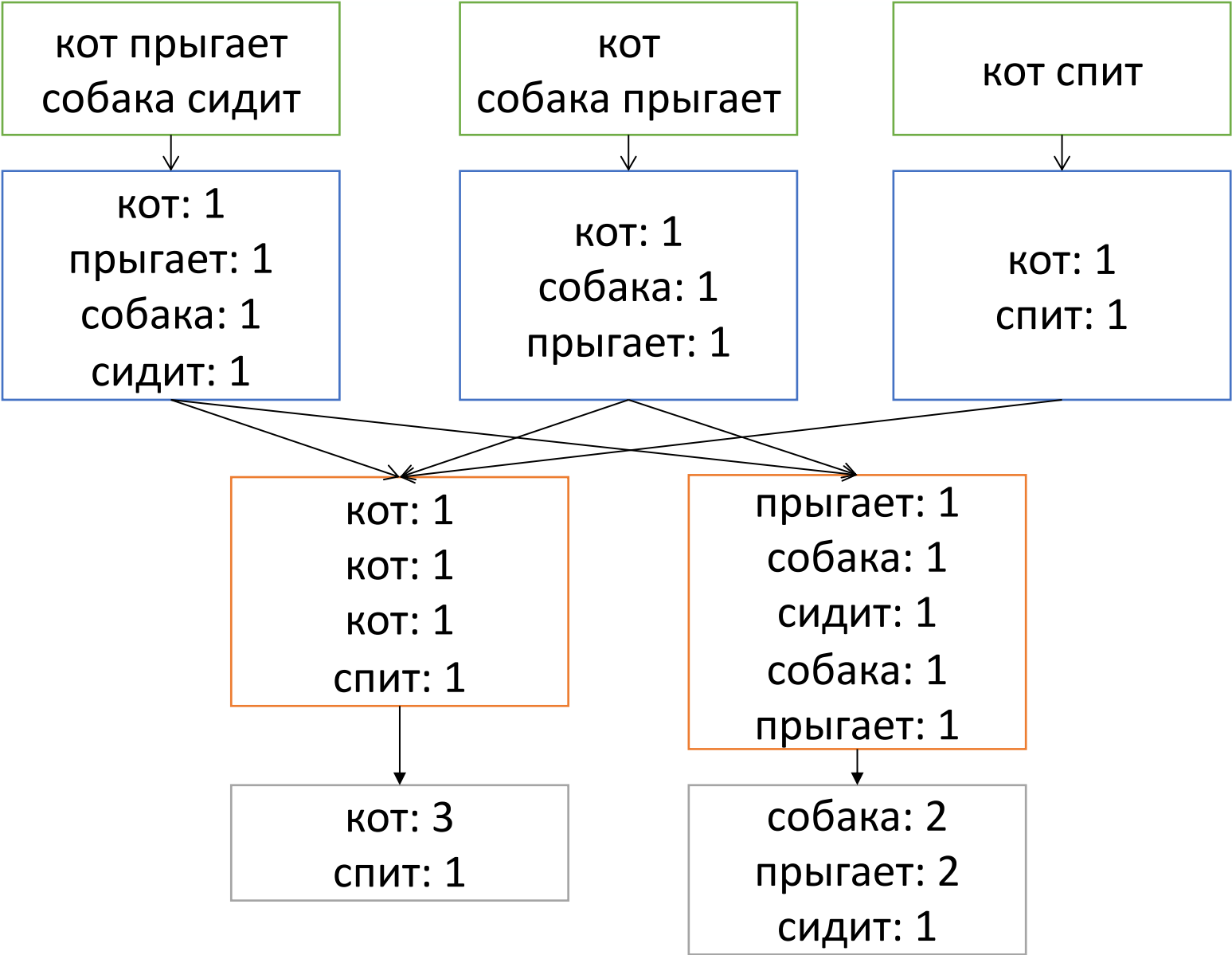
Наивный подход



Делим работу пополам по условию



Независимо обрабатываем половинки



Обобщим условие разделения работы

Предположим, мы хотим поделить работу на **N** частей.

Работу будем делить по значениям **hash(word) % N**, которое показывает на какую машину отправить слово.

Для хорошей хэш-функции (hash) все значения равновероятны.

Пример хэш-функции (*полиномиальная*):

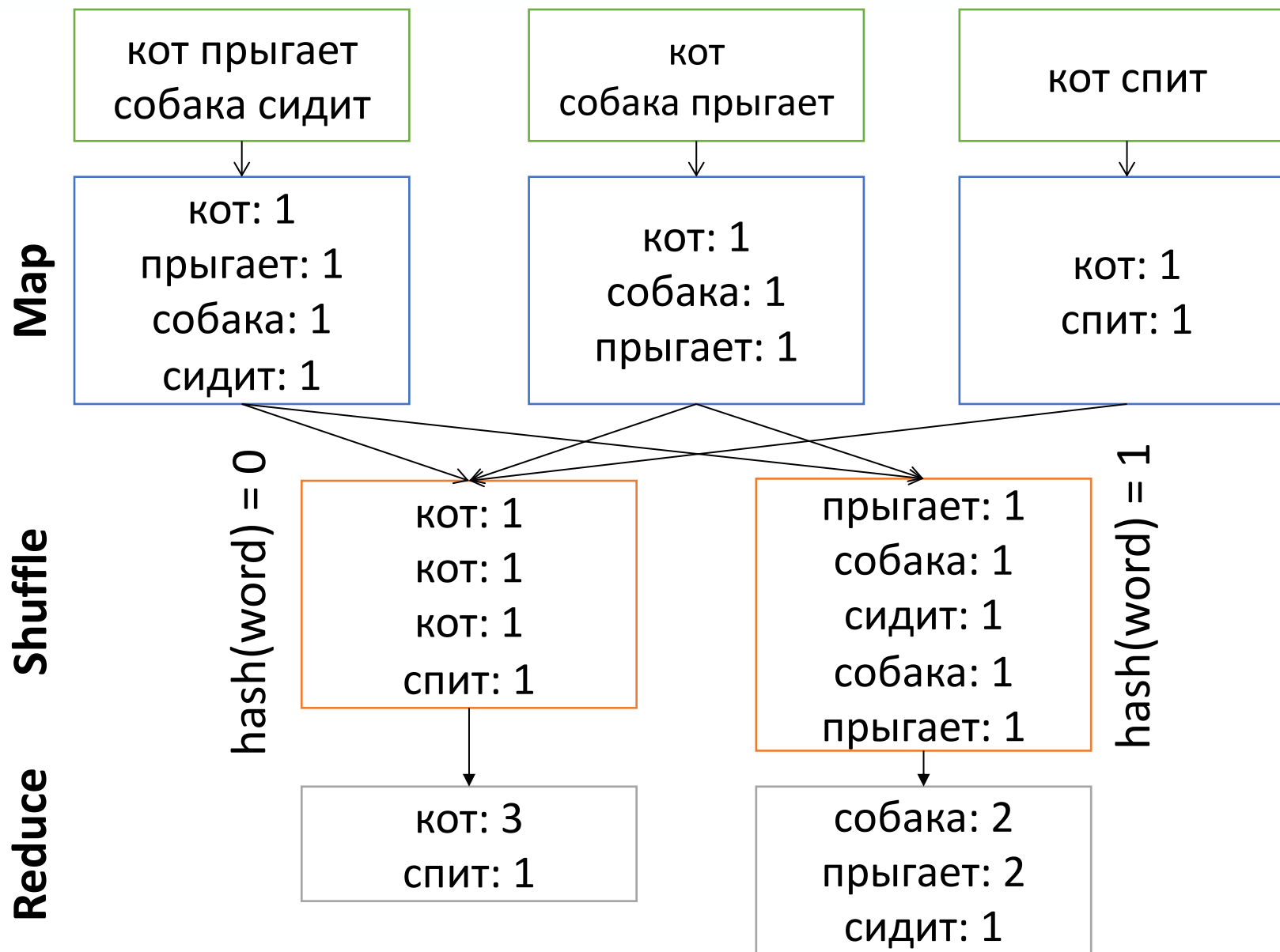
$$\text{hash}(s) = s[0] + s[1]p^1 + \dots + s[n]p^n$$

s – строка

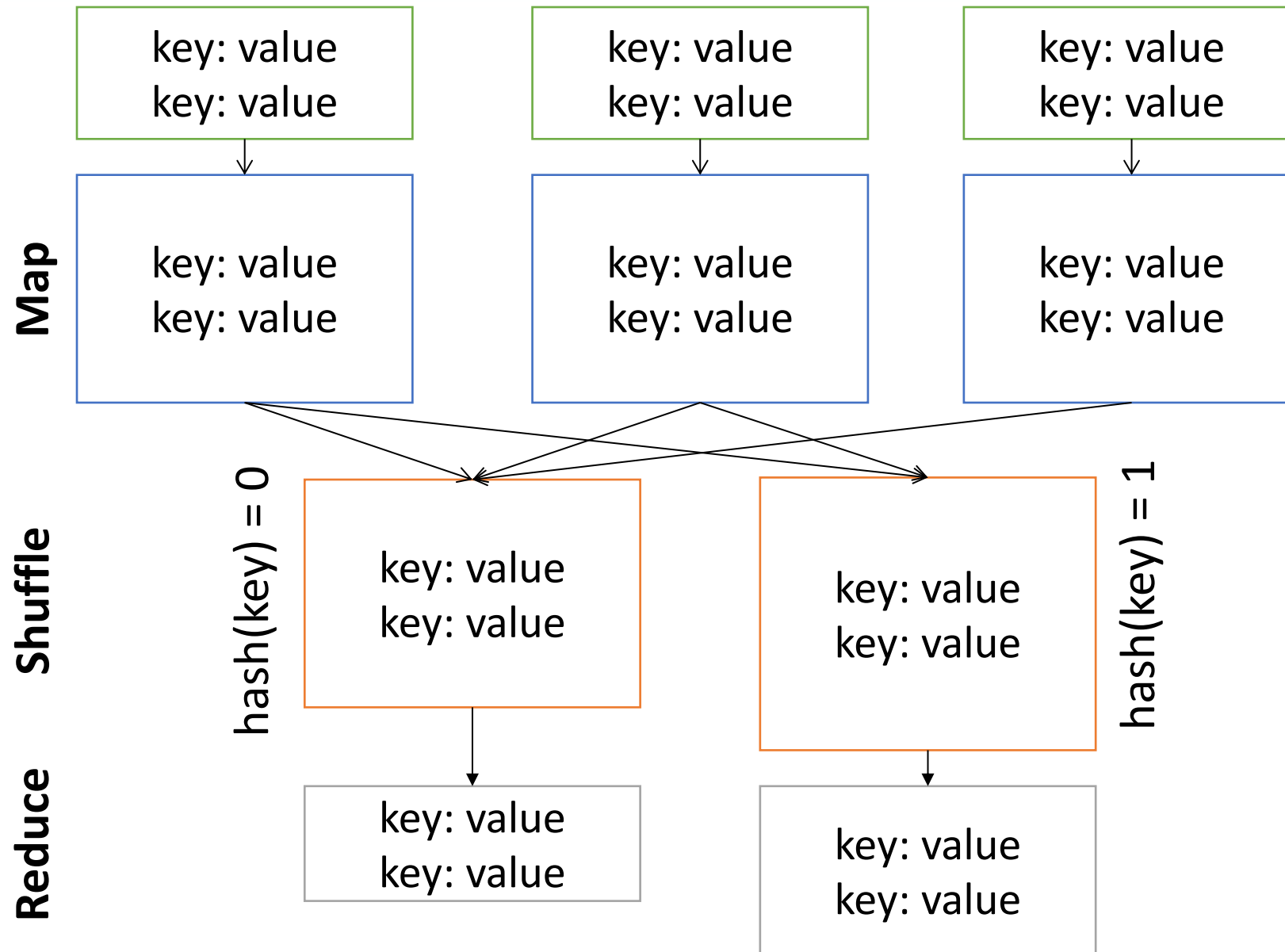
p – фиксированное простое число

$s[i]$ – код символа

Обобщим схему до MapReduce



Обобщим схему до MapReduce



Парадигма MapReduce

Map:

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

(номер строки, "кот спит") \rightarrow [("кот", 1), ("спит", 1)]

Shuffle:

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

Каждая часть **сортируется по key** (независимо)

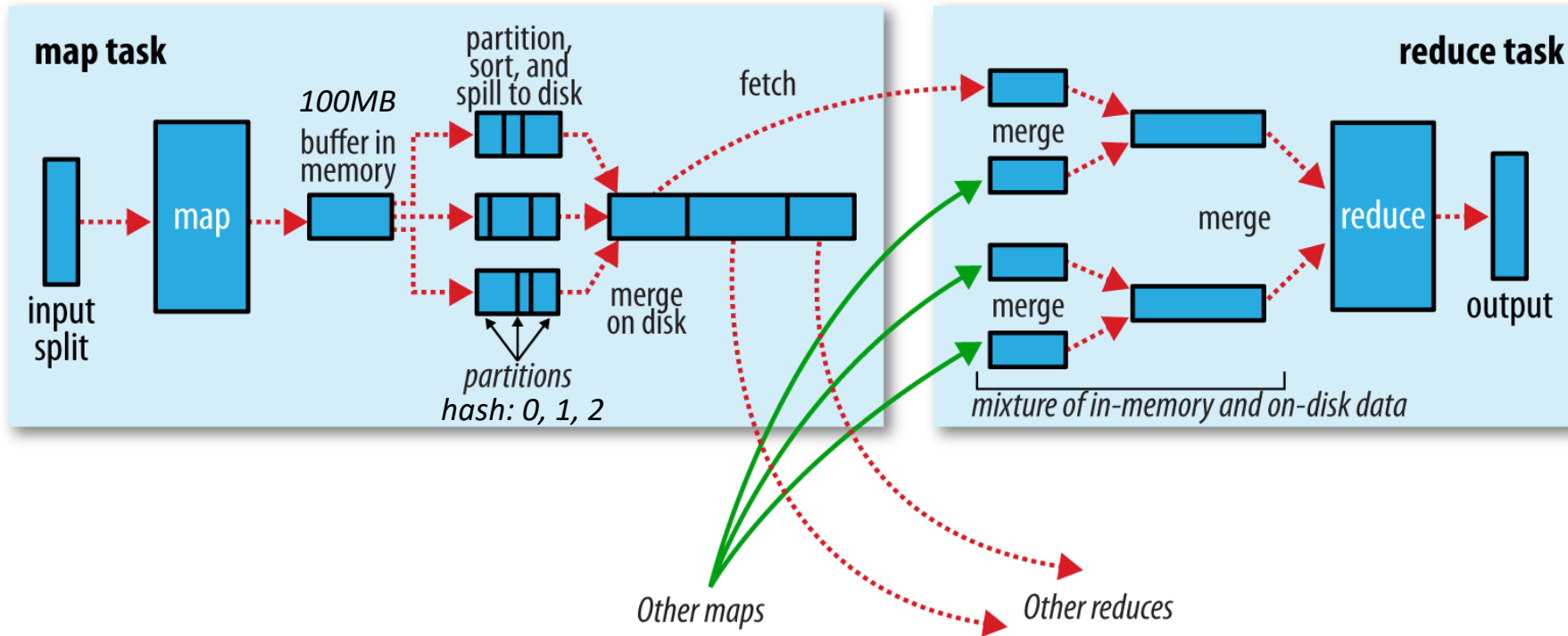
Reduce:

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

("кот", (1, 1, 1)) \rightarrow [("кот", 3)]

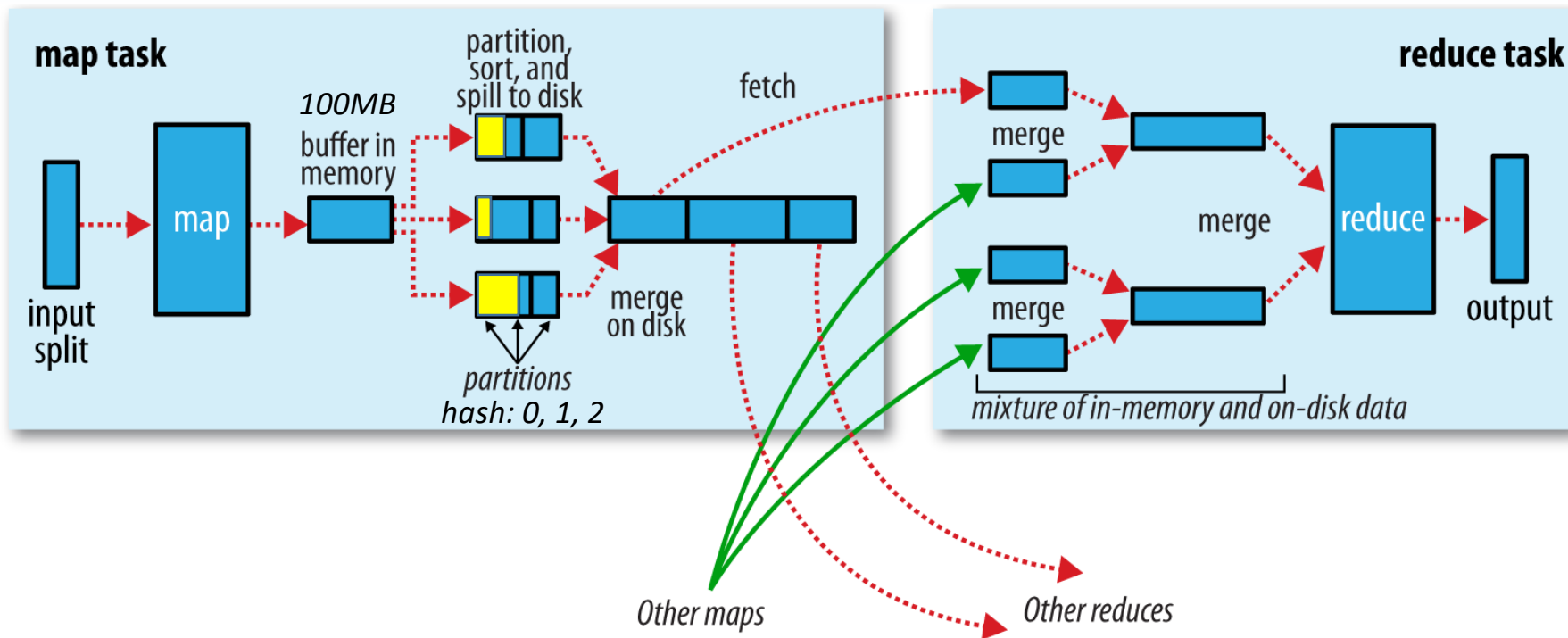
Удивительно, но довольно много вычислений ложатся на эту простую схему

Сортировку на шаге Shuffle можно ускорить



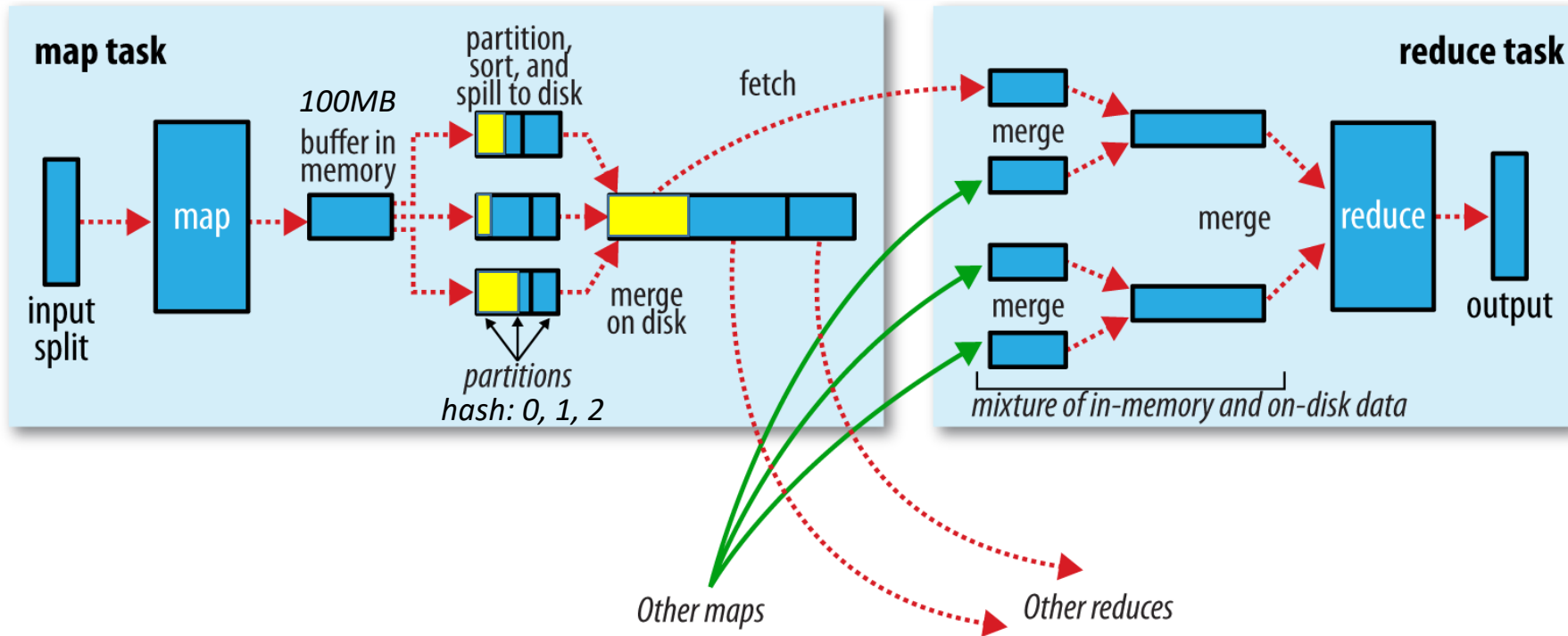
- Результат map шага можно отсортировать локально
- После можно использовать сортировку слиянием (merge sort) на шаге Shuffle за линейное время

Сортировку на шаге Shuffle можно ускорить



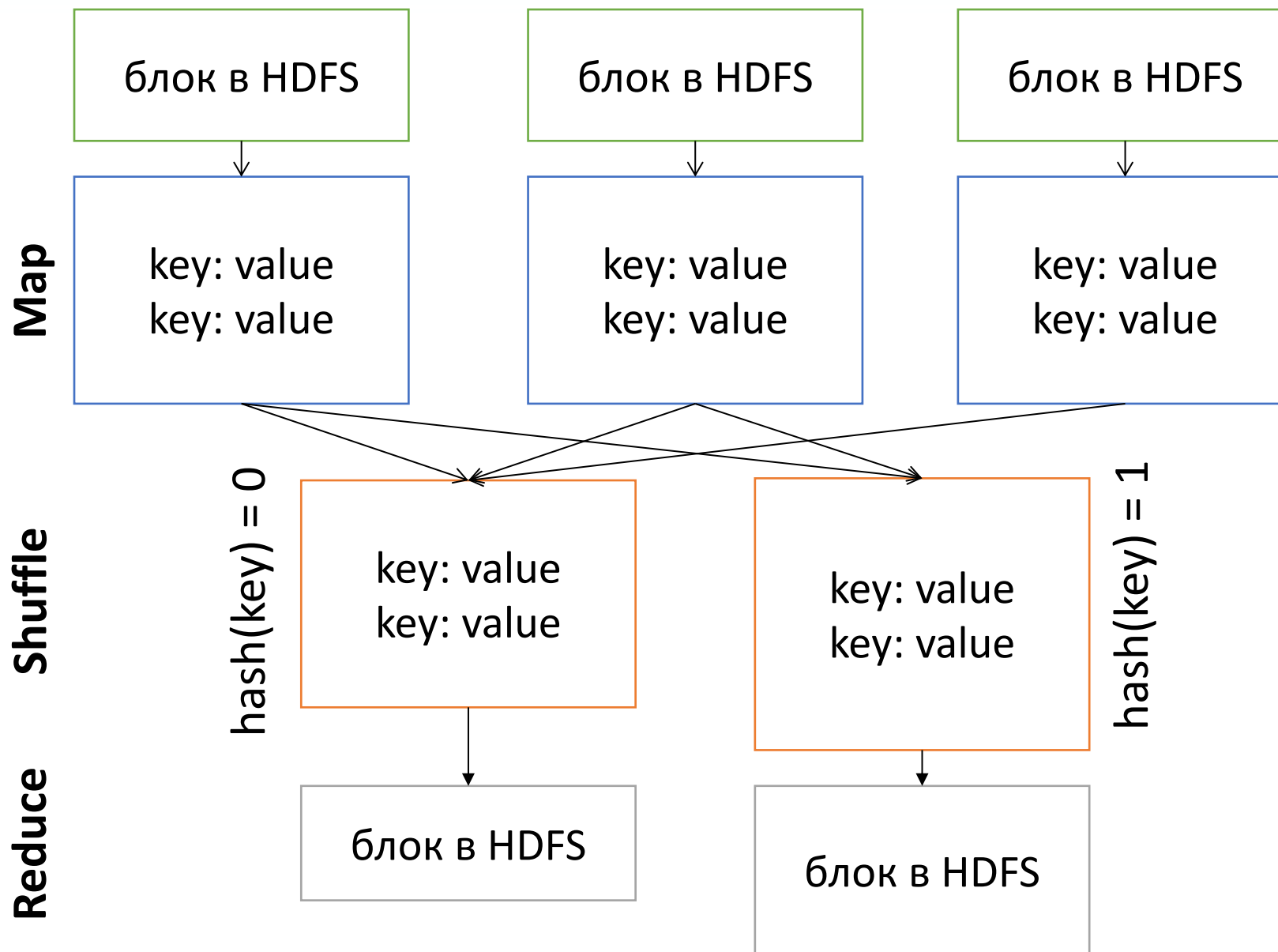
- Результат map шага можно отсортировать локально
- После можно использовать сортировку слиянием (merge sort) на шаге Shuffle за линейное время

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

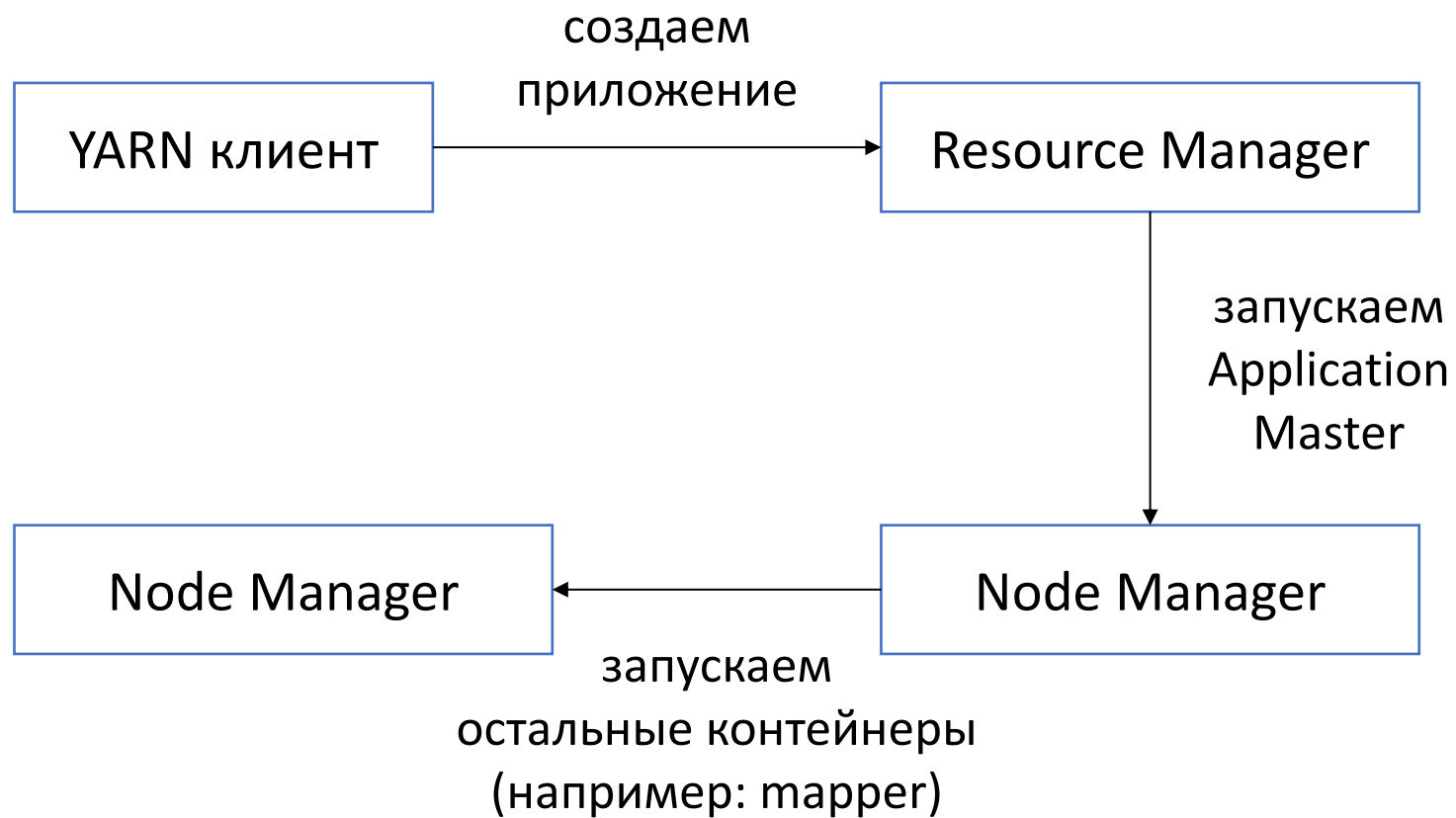
Работа с HDFS



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

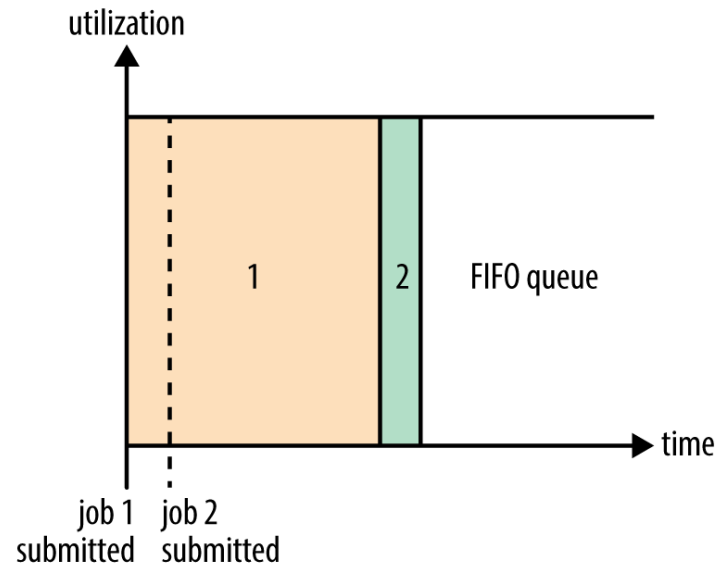
Работа с YARN



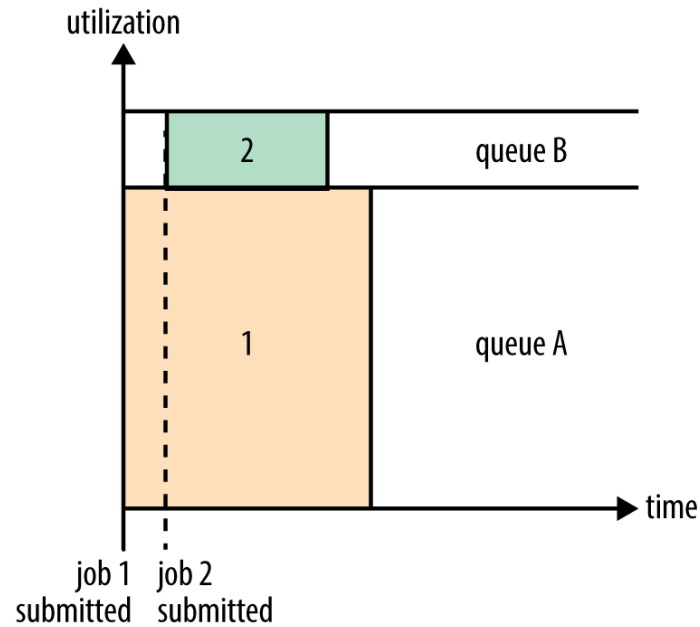
MapReduce работает поверх YARN

Планировщик в YARN

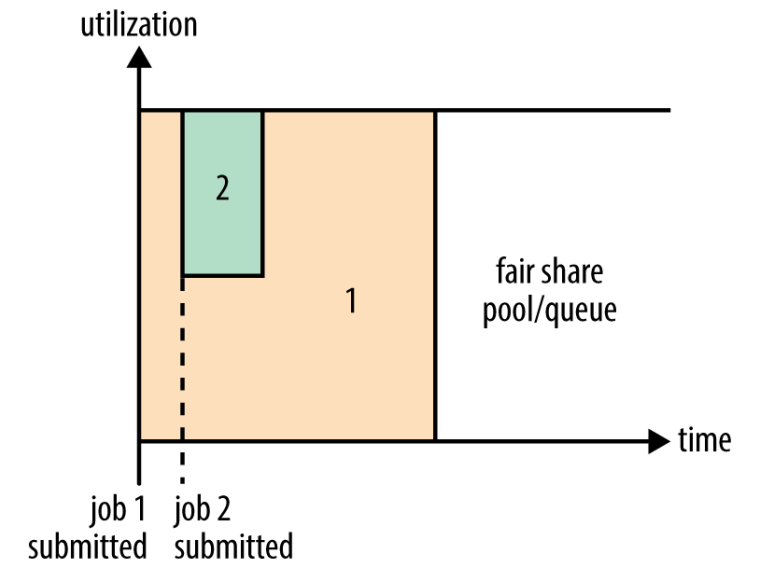
i. FIFO Scheduler



ii. Capacity Scheduler



iii. Fair Scheduler



MapReduce пример

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

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

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

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

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

Возьмем логи прослушиваний в Spotify.

Хотим для каждого альбома найти самый популярный трек.

M: $\#, (\text{user}, \text{track}, \text{album}) \rightarrow (\text{album}, \text{track}), 1$

R: $(\text{album}, \text{track}), (1, 1, \dots) \rightarrow (\text{album}, \text{track}), \text{count}$

MapReduce пример

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

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

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

Возьмем логи прослушиваний в Spotify.

Хотим для каждого альбома найти самый популярный трек.

M: $\#, (\text{user}, \text{track}, \text{album}) \rightarrow (\text{album}, \text{track}), 1$

R: $(\text{album}, \text{track}), (1, 1, \dots) \rightarrow (\text{album}, \text{track}), \text{count}$

M: $(\text{album}, \text{track}), \text{count} \rightarrow \text{album}, (\text{track}, \text{count})$

R: $\text{album}, \text{tracks} \rightarrow \text{album}, \text{most popular track}$

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
- YARN – менеджер ресурсов (на нем будет работать и Spark)