

Хранение и Обработка Больших Объёмов Данных

Антон Горохов

старший разработчик, Яндекс

anton.gorokhov@gmail.com

План лекции

I. Файловая система HDFS

- 1) Устройство
- 2) Интерфейс

II. MapReduce задача

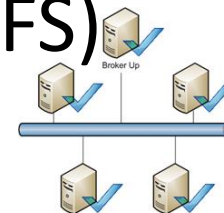
- 1) Combiner
- 2) Comparator
- 3) Partitioner
- 4) Настройки задачи

Требования к хранилищу

- Большой объем
- Последовательный доступ (write once, read many)
- Устойчивость к отказам оборудования

Где можно хранить

- HDD, RAID массивы
 - на одном сервере — ненадежно
- Сетевые хранилища (NAS)
 - не предназначены для обработки
- Распределенные файловые системы (DFS)
- Облачные хранилища (cloud storage)
 - расходы на пересылку
 - конфиденциальность



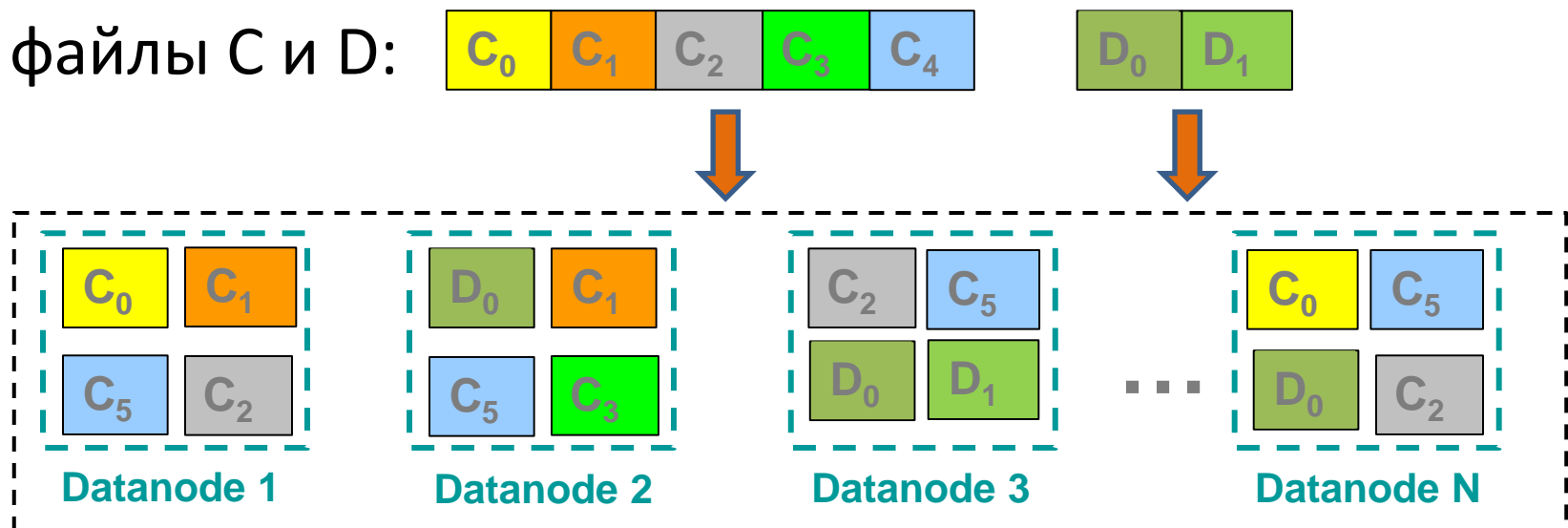
Специфика BigData: большие файлы

Разбиваем файлы на крупные блоки:

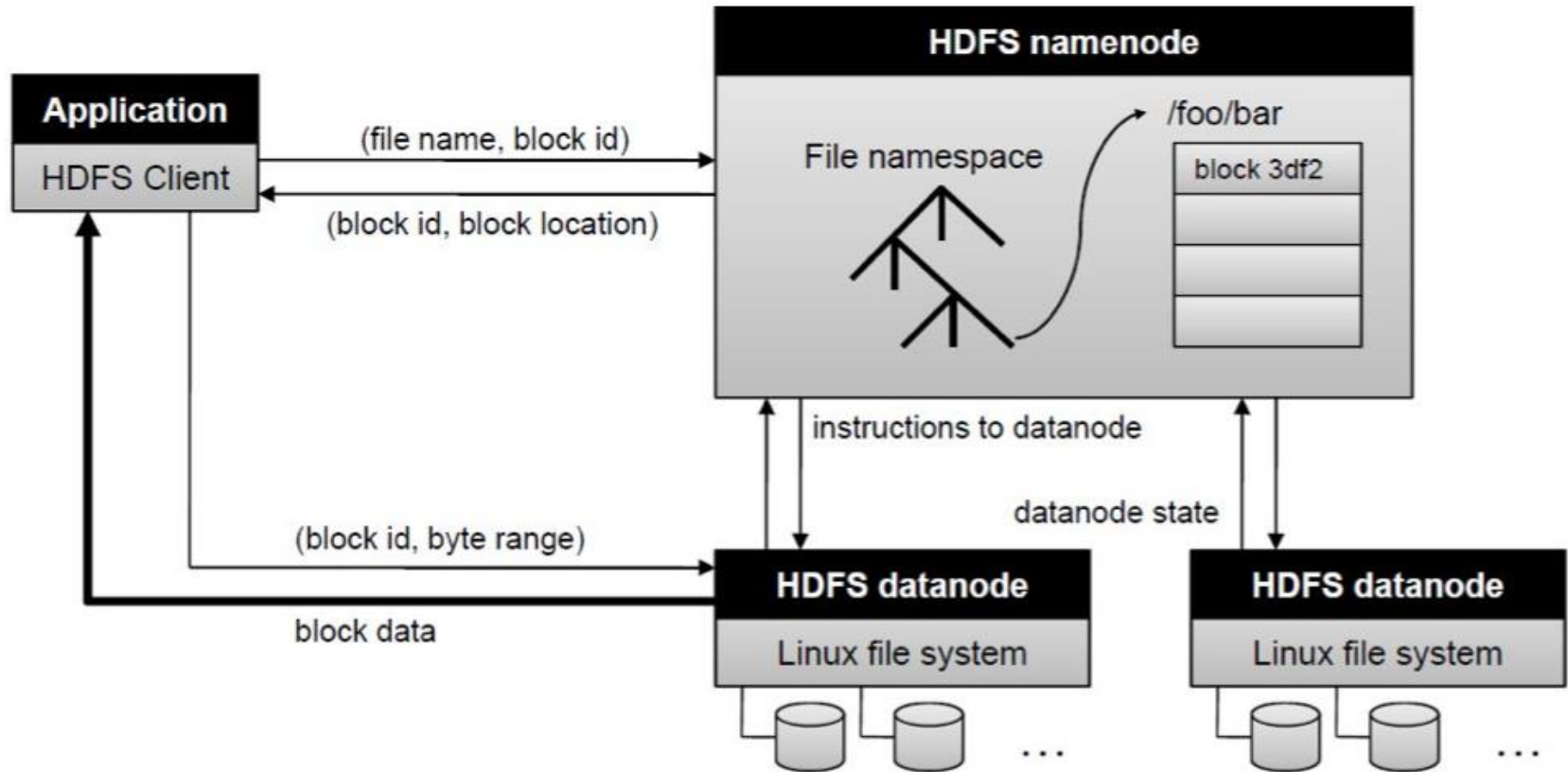
- время поиска << время чтения
- равномерное заполнение кластера
- равномерная загрузка кластера при обработке

HDFS – Hadoop Distributed File System

- Иерархическая структура, управляется **namenode**
- Файлы разбиваются на блоки (64 Mb)
- Блоки хранятся на узлах **datanodes** (3 реплики)
 - несколько реплик каждого блока (обычно 3)
 - учет топологии сети (не храним все в одной стойке)



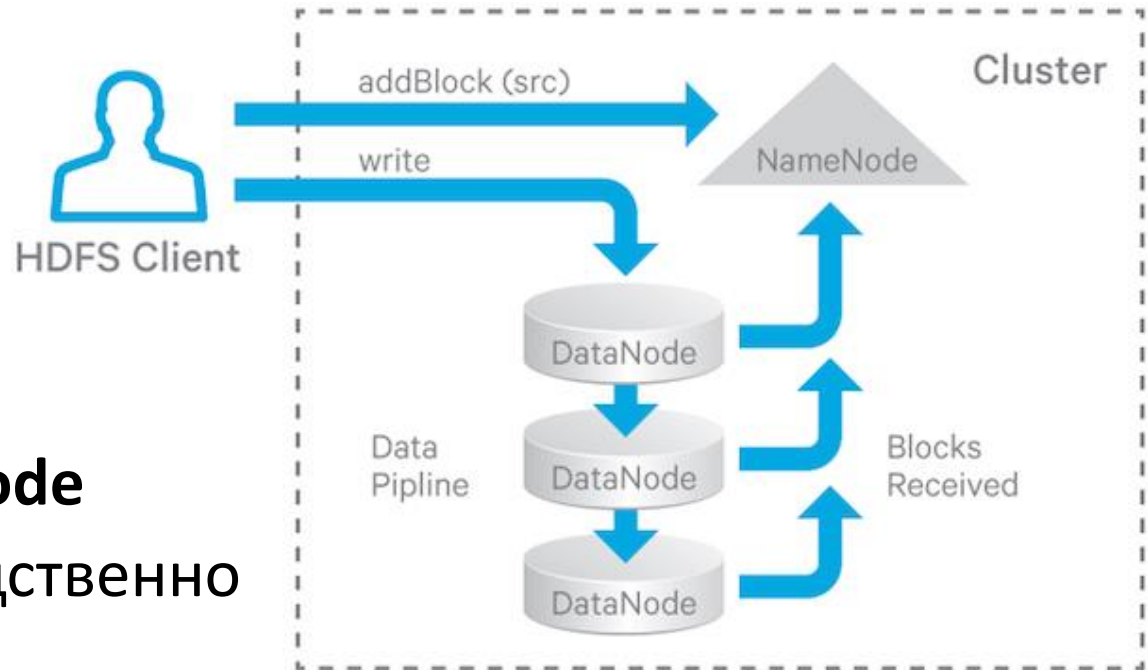
HDFS – Hadoop Distributed File System



Чтение:

- Клиент запрашивает у namenode реплики блоков
- Чтение **непосредственно** с ближайшей реплики

Запись в HDFS



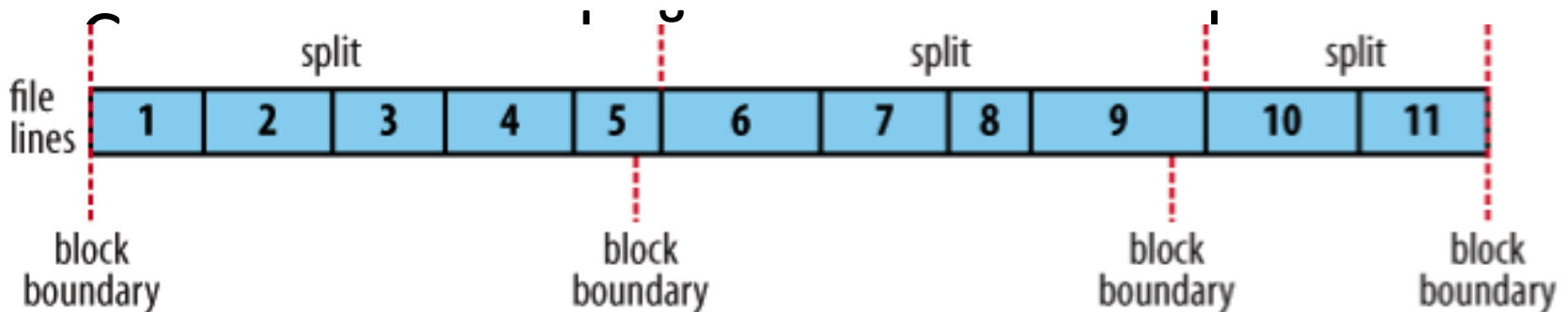
- Запрос к **namenode**
- Запись непосредственно на **datanode**
- Datanode передает блок дальше и сообщает о записи на namenode
- ... и так для каждого блока

Недостатки HDFS

- Большое время отклика (латентность)
- Не любит много мелких ($\ll \text{block_size}$) файлов
 - namenode хранит базу блоков в памяти
- Нет поддержки конкурентной записи
- Добавление (append) файла не стабильно
- Единая точка отказа – namenode
 - secondary namenode – резервный сервер, синхронизирует базу с namenode, ручное переключение

HDFS: splits

- Разделение файла на блоки не должно испортить его формат
 - текстовый файл состоит из строк
 - bz2 – из сжатых блоков
 - gz – тоже, но границы блока неразличимы



- Обычно: 1 сплит – одна тар-задача

План лекции

I. Файловая система HDFS

- 1) Устройство
- 2) Интерфейс

II. MapReduce задача

- 1) Combiner
- 2) Comparator
- 3) Partitioner
- 4) Настройки задачи

Интерфейс командной строки

```
$ hadoop fs -help
```

```
$ hadoop fs -ls /path
```

```
$ hadoop fs -put local remote
```

```
$ hadoop fs -get remote local
```

```
$ hadoop fs -cat remote
```

Программный интерфейс

SequenceFile – бинарный формат, key-value

```
FileSystem fs = FileSystem.get(conf);

Path file = new Path("/tmp", "test.txt");

Text word = new Text("test");
LongWritable count = new LongWritable(17);
SequenceFile.Writer writer = SequenceFile.createWriter(fs, conf, file,
    Text.class, LongWritable.class, CompressionType.NONE);

writer.append(word, count);
writer.close();

SequenceFile.Reader reader = new SequenceFile.Reader(fs, inFile, jobConf);
reader.next(word, count);
reader.close();
```

План лекции

I. Файловая система HDFS

- 1) Устройство
- 2) Интерфейс

II. MapReduce задача

- 1) Combiner
- 2) Comparator
- 3) Partitioner
- 4) Настройки задачи

Идея MapReduce

- *чтение $(k1, v1)$*
- *map: $(k1, v1) \rightarrow [(k2, v2)]$*
- *сортировка и группировка по ключу $k2$*
- *reduce: $(k2, [v2]) \rightarrow (k3, v3)$*

Задача: grep

$(docid, content) \rightarrow (docid, content^*)$

*- content содержит искомую фразу

map: $(docid, content) \rightarrow (docid, content^*)$

reduce: отсутствует

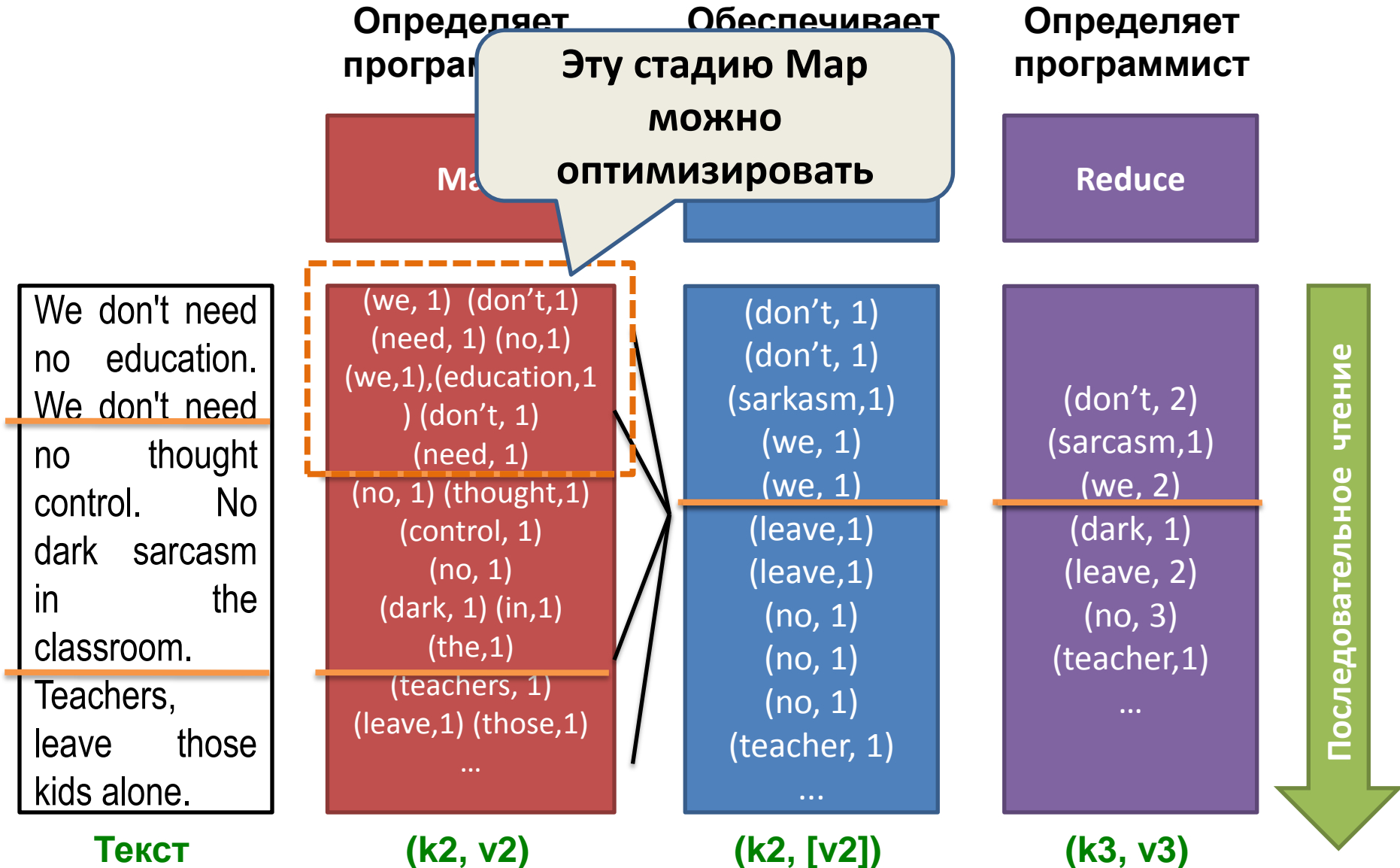
Задача: group by

$(\text{key}, \text{record}) \rightarrow (\text{key}, [\text{record}])$

map: тривиальный

reduce: тривиальный

Задача: подсчет слов



Hadoop: combiner

Определяет
программист

Map

(we, 1)
(don't, 1)
(need, 1)
(no, 1)
(we, 1)
(education, 1)
(don't, 1)
(need, 1)

Обеспечивает
Hadoop

Sort

(don't, 1)
(don't, 1)
(education, 1)
(need, 1)
(need, 1)
(no, 1)
(we, 1)
(we, 1)

Определяет
программист

Combine

(don't, 2)
(education, 1)
(need, 2)
(no, 1)
(we, 2)

Shuffle,
Reduce

Здесь combine = sum, т.е. combine = reduce

Hadoop: combiner

- Hadoop сортирует результат работы Map
 - объем относительно невелик
- Combine агрегирует отсортированный фрагмент
- Выгода:
 - Меньше данных пересылать на Reduce
 - ... и меньше там обрабатывать
- **NB:** combine не меняет типы $\langle k_2, v_2 \rangle$
 - т.к. применяется 0 и более раз

Hadoop: combiner

- reduce = sum
combine = ?
- reduce = avg
combine = ?
- reduce = median
combine = ?

Hadoop: combiner

- reduce = sum
combine = sum
- reduce = avg
combine = (sum, count)
- reduce = median
combine = -

План лекции

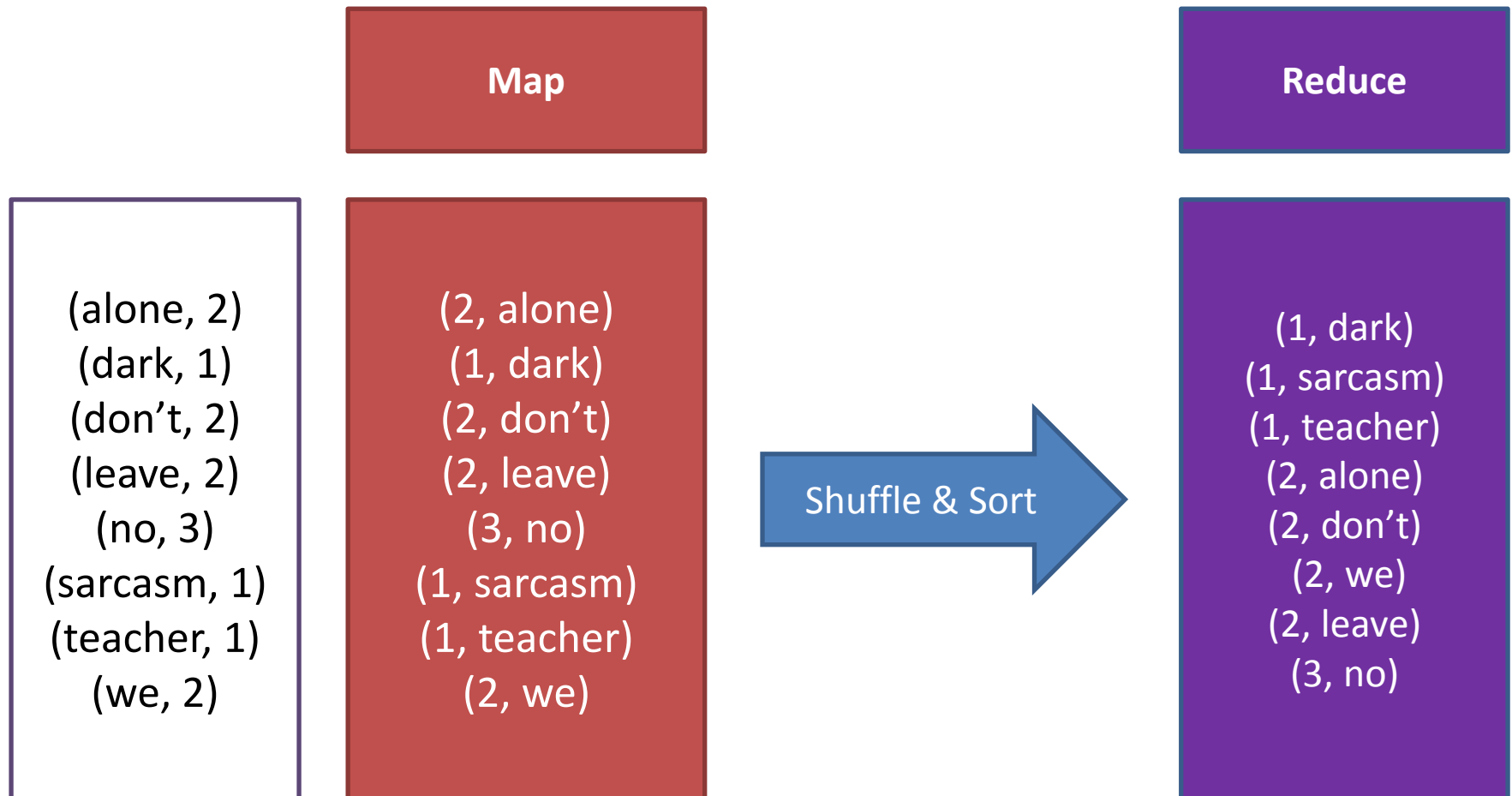
I. Файловая система HDFS

- 1) Устройство
- 2) Интерфейс

II. MapReduce задача

- 1) Combiner
- 2) Comparator
- 3) Partitioner
- 4) Настройки задачи

Задача: сортировка по количеству

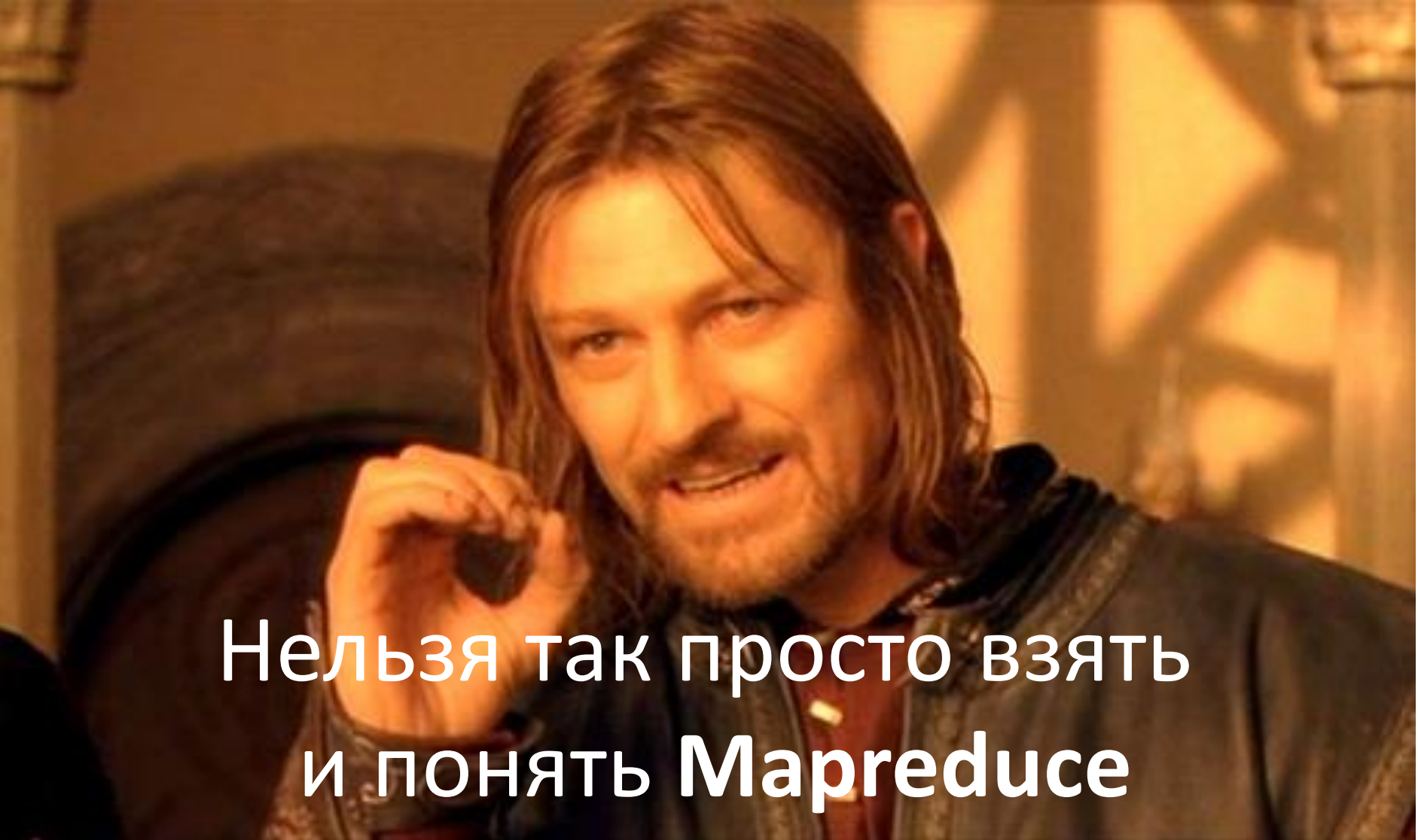


Что делать, если нужна сортировка в обратном порядке?

Hadoop: comparator

- Возможно задавать функцию сравнения
 - сравниваются всегда только ключи
- Сравнение зависит от:
 - типа данных, метод compareTo()
 - компаратора, переданного в `job.setSortComparatorClass()`

25й слайд



Нельзя так просто взять
и понять **Mapreduce**

Типы данных

- implements WritableComparable, Writable
 - Text, BooleanWritable, IntWritable,
 - LongWritable, DoubleWritable,
 - NullWritable – если тип отсутствует
- implements Writable
 - ArrayWritable, MapWritable,...
- Writable – Hadoop умеет сериализовывать
- WritableComparable – умеет сравнивать

**Соответствуют
Java типам:**
String, boolean,
int, long,
double

Форматы входных файлов

- implements InputFormat<K,V>
- TextInputFormat
 <LongWritable, Text> = (byte_offset, line)
- KeyValueTextInputFormat
 <Text, Text>
 Текстовый файл со строками вида: key <tab> value
- SequenceFileInputFormat<K,V>
 — бинарный формат

Форматы выходных файлов

- implements `OutputFormat<K,V>`
- `TextOutputFormat<K,V>`
текстовый файл со строками вида: key <tab> value
- `SequenceFileOutputFormat<K,V>`
бинарный формат

План лекции

I. Файловая система HDFS

- 1) Устройство
- 2) Интерфейс

II. MapReduce задача

- 1) Combiner
- 2) Comparator
- 3) Partitioner
- 4) Настройки задачи

Задача: обратный индекс

Input: (docid, content)

Output: (term, [docid,...])

term – слово из content

map: (docid, content) -> [(term, docid),...]

reduce: (term, [docid,...])

combine = reduce

Задача: обратный индекс (2)

Input: docid, content

Output: term, [(docid, tf), ...]

term – слово из content

tf – term frequency, т.е. сколько раз term встретился в документе

map: docid, content -> (term, docid), 1, ...

reduce: (term, docid), sum, ...

т.е. key=(term, docid), но **каждый term должен попасть на один редьюсер**

Hadoop: partitioner

- Контролирует вычисления номера редьюсера по ключу k_2
- По умолчанию HashPartitioner
 - тип ключа k_2 должен определять `hashCode()`
 - `hashCode() % R`
- `abstract class Partitioner<K, V>`
 - определить метод: `int getPartition(k, v, R)`

План лекции

I. Файловая система HDFS

- 1) Устройство
- 2) Интерфейс

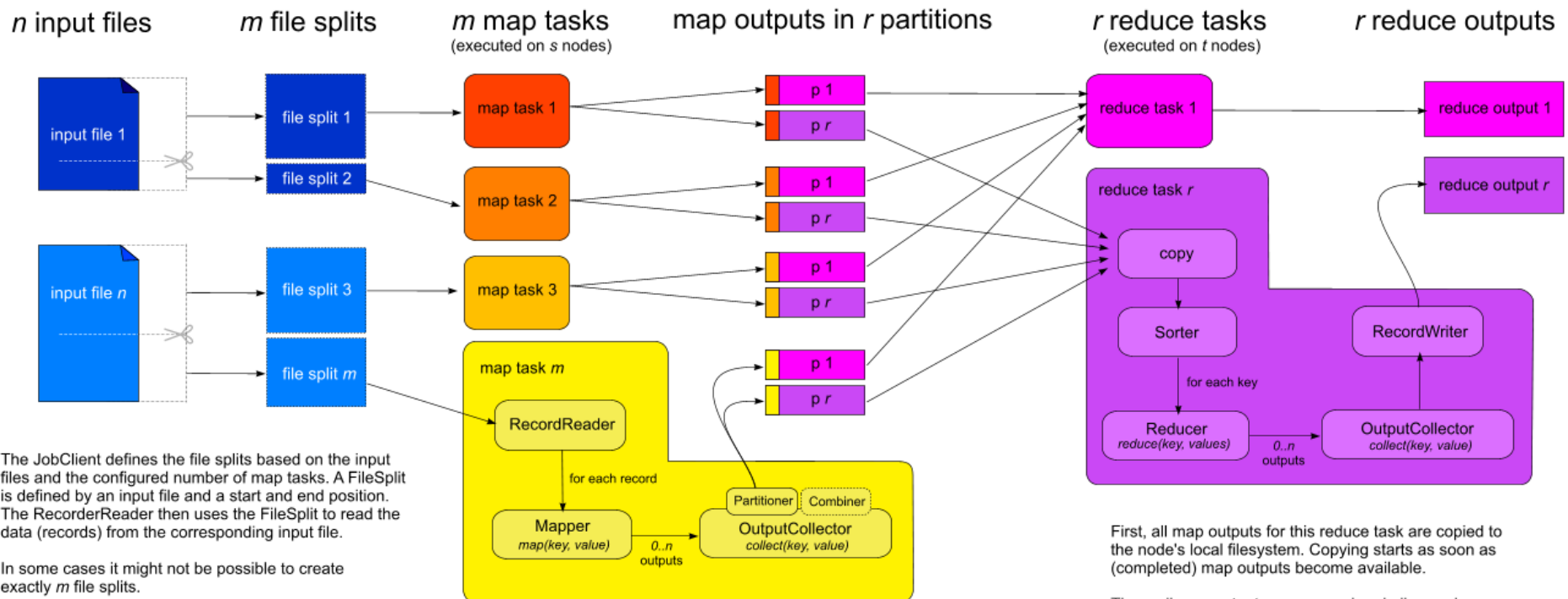
II. MapReduce задача

- 1) Combiner
- 2) Comparator
- 3) Partitioner
- 4) Настройки задачи

Схема работы Hadoop

map

reduce



The JobClient defines the file splits based on the input files and the configured number of map tasks. A FileSplit is defined by an input file and a start and end position. The RecordReader then uses the FileSplit to read the data (records) from the corresponding input file.

In some cases it might not be possible to create exactly *m* file splits.

Each record (key-value pair) read by the RecordReader is passed to the Mapper's map function.

The map input key and map output key(s) need not be the same. Each call to the map function may result in zero, one or even multiple output records.

A Partitioner defines *r* partitions (typically based on the output key) for writing the map output records.

Optionally, for optimization, a Combiner can be used to combine output records with the same key before they

First, all map outputs for this reduce task are copied to the node's local filesystem. Copying starts as soon as (completed) map outputs become available.

Then, all map outputs are merged and all records are sorted by their key. This way, records with the same key are placed sequentially.

Each unique key and all corresponding values are passed to the Reducer's reduce function. The reduce function calls the OutputCollector to collect outputs which are then written to the reduce output file by the RecordWriter.

Число задач

- Maps
 - Определяется числом блоков во входных файлах, размером блока, параметром `mapred.min.split.size`, реализацией `InputFormat`
`split_size = max(min_split_size=0, min(max_split_size=Long.MAX, dfs_block_size=128MB))`
- Reduces
 - По умолчанию 1 (в локальном режиме всегда 1)
 - Опция `-D mapred.reduce.tasks=N` (в Java: `job.setNumReduce(int)`)
 - 0, если reduce не нужен
 - Обычно подбирается опытным путем

Настройки задачи

```
@Override public int run(String[] args) throws Exception {  
    Job job = JobBuilder.parseInputAndOutput(this, getConf(), args);
```

```
    job.setInputFormatClass(TextInputFormat.class);  
    job.setOutputFormatClass(TextOutputFormat.class);
```

форматы

```
    job.setMapperClass(Mapper.class);  
    job.setMapOutputKeyClass(LongWritable.class);  
    job.setMapOutputValueClass(Text.class);
```

mapper + типы*

```
    job.setPartitionerClass(HashPartitioner.class);  
    job.setSortComparatorClass(RawComparator.class);
```

partitioner
comparator для reduce

```
    job.setNumReduceTasks(1);  
    job.setReducerClass(Reducer.class);  
    job.setCombinerClass(Reducer.class);
```

reducer
combiner

```
    job.setOutputKeyClass(LongWritable.class);  
    job.setOutputValueClass(Text.class);
```

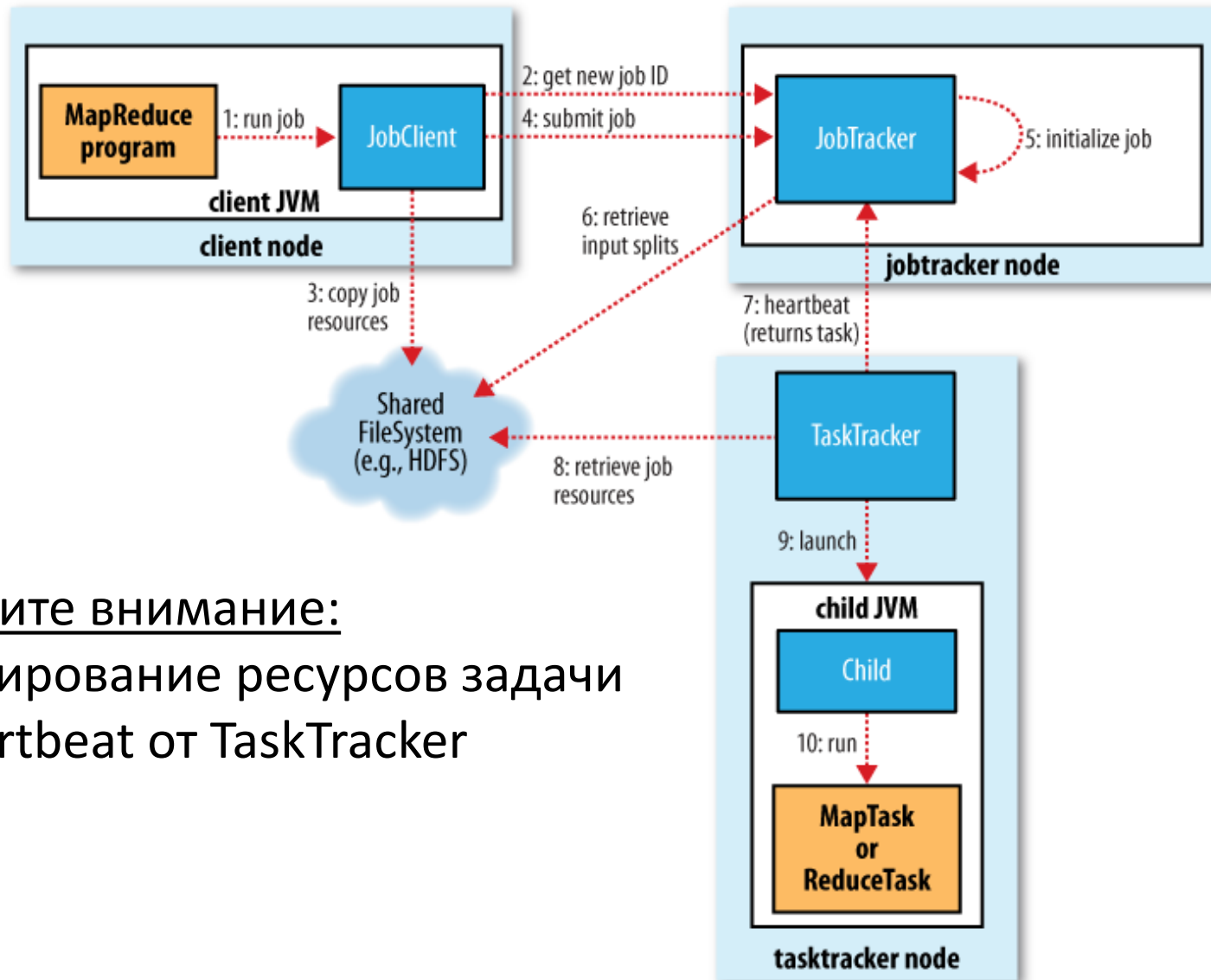
типы*

```
    return job.waitForCompletion(true) ? 0 : 1;
```

```
}
```

** в Java по классу нельзя определить тип в шаблоне*

Процесс запуска задачи



Обратите внимание:

- Копирование ресурсов задачи
- Heartbeat от TaskTracker

Управление задачей

Старт:

```
$ hadoop jar <JarFile> [<MainClass>] <args>
```

- будет выведен JobId – id задачи

Стоп:

```
$ mapred job -kill <JobId>
```

NB: по Ctrl+C задача не убивается

Термины

- Job – задача на кластере
- Task – map и reduce стадии
 - Map – столько, сколько splits во входном файле
 - Reduce – столько, сколько указал программист
- Attempt – попытка выполнения конкретного task
 - Несколько попыток на каждый task в случае неудачи
 - Speculative execution – запуск несколько attempts одного task: кто быстрее (можно запретить)

Вопросы?

I. Файловая система HDFS

- 1) Устройство
- 2) Интерфейс

II. MapReduce задача

- 1) Combiner
- 2) Comparator
- 3) Partitioner
- 4) Настройки задачи