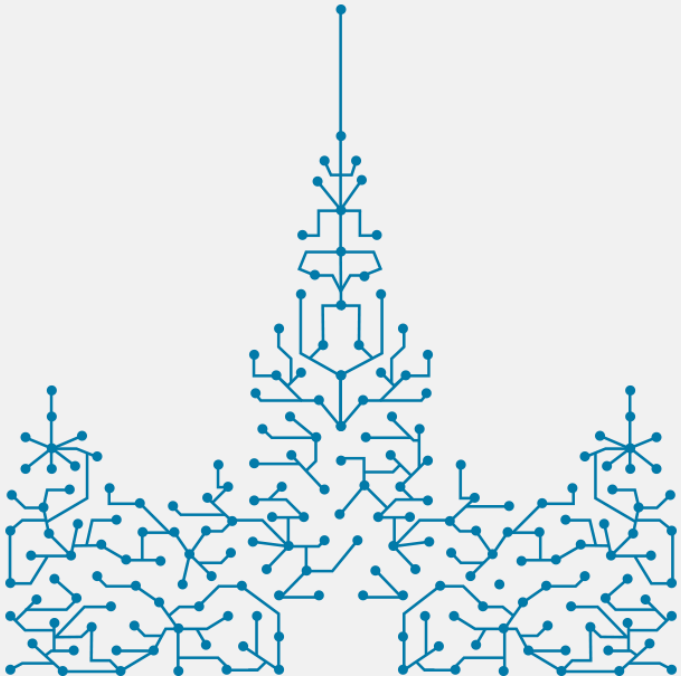
ТЕХНОСФЕРА

Лекция №4

MapReduce в Hadoop. Введение.

Чернов Евгений

Необходимость MapReduce

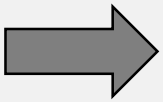


Задача

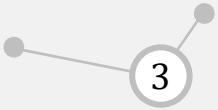
Сколько раз в файле text.txt встречается каждое слово?

text.txt:

```
aut Caesar aut nihil
aut aut
de mortuis aut bene aut nihil
```



```
aut 6
Caesar 1
nihil 2
de 1
mortuis 1
bene 1
```



Решение

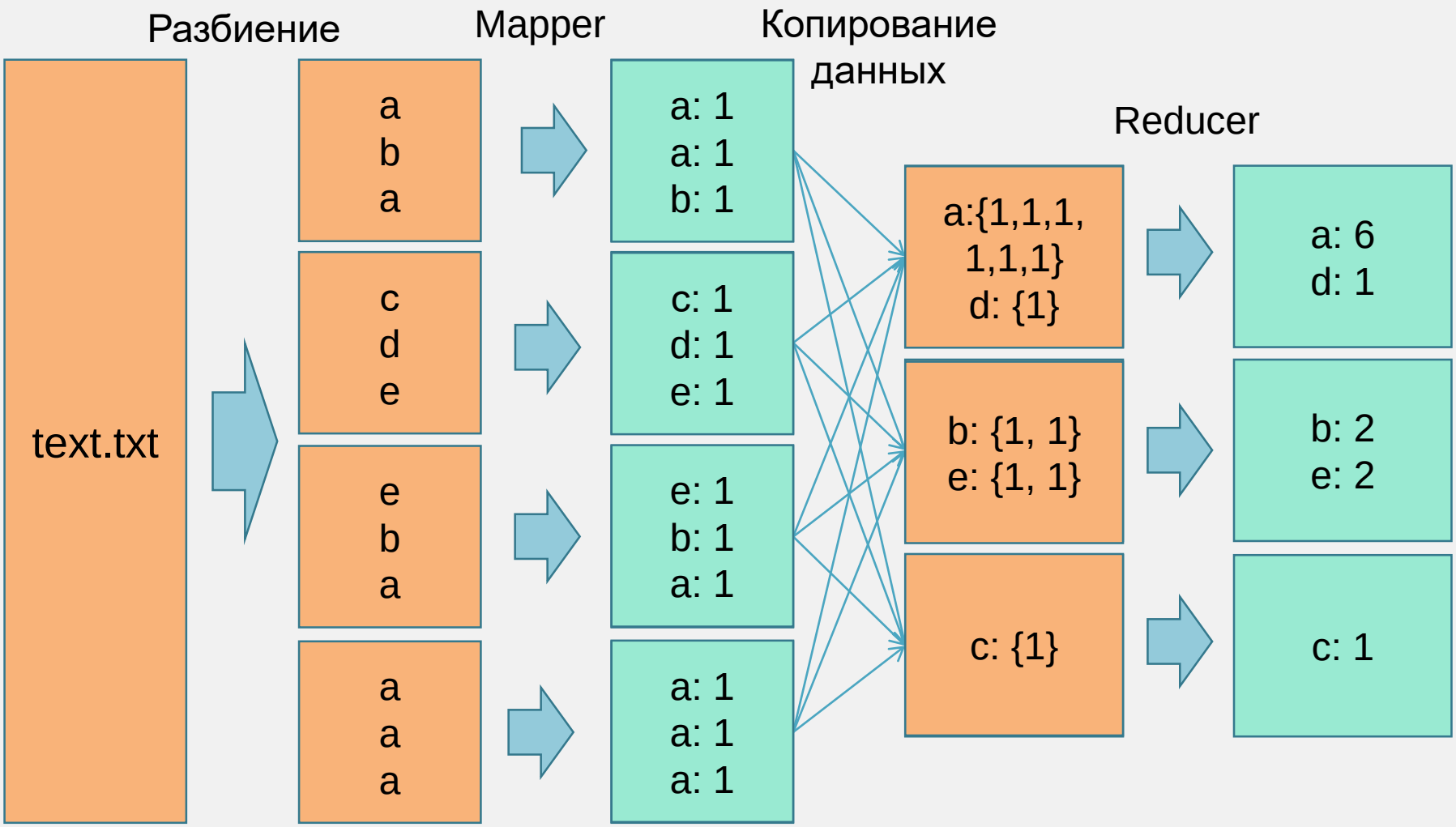
```
#include <iostream>
#include <string>
#include <unordered_map>
using namespace std;

int main() {
	string str;
	ifstream file("text.txt")
	unordered_map<string, int> map;
	while (file >> str)
			++map[str];

	for (auto word: map)
		cout << word.first << "\t" << word.second <<
endl;

	return 0;
}
```

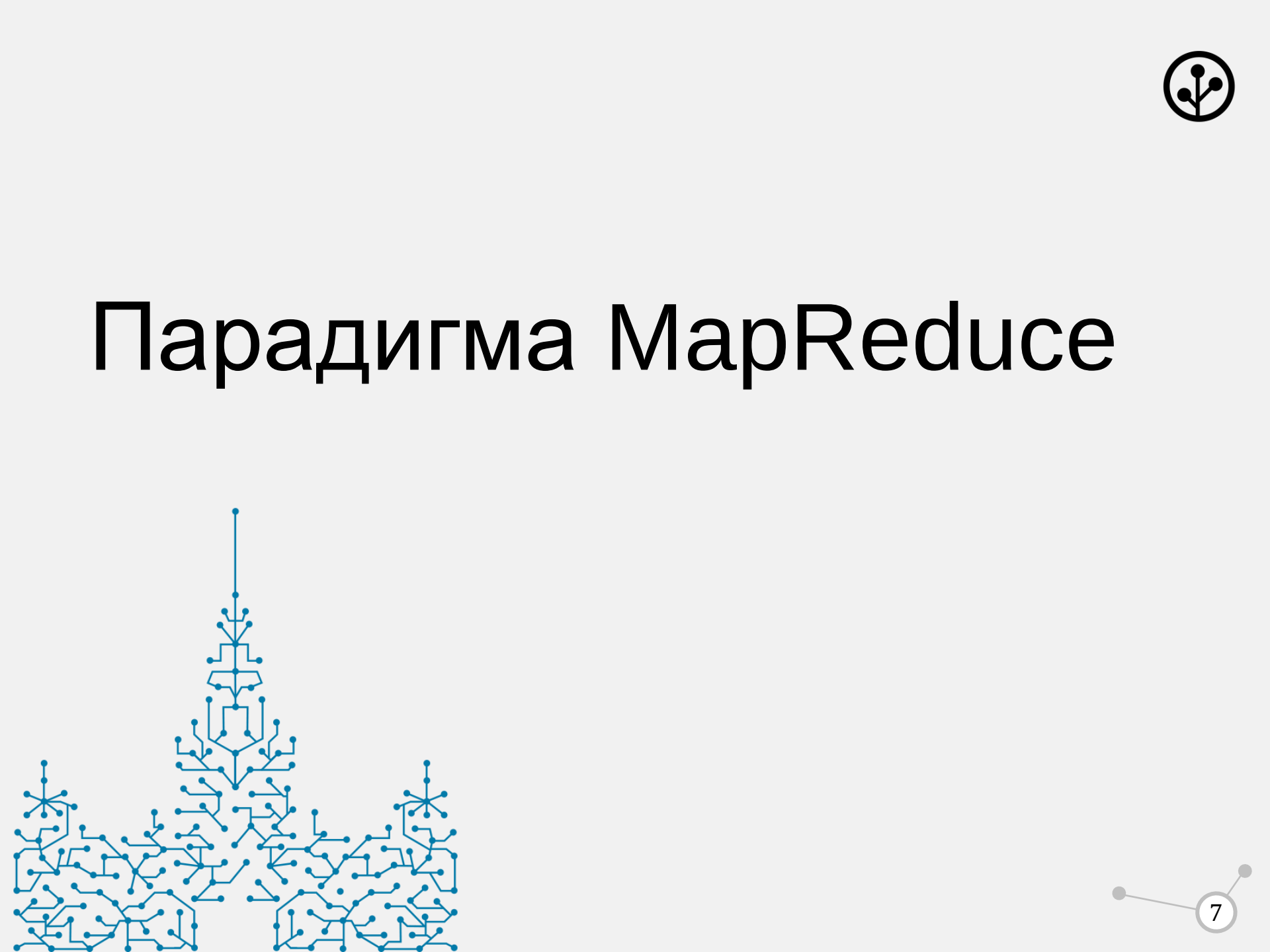
Проблемы однопоточной программы

Если вам нужно обработать файл, размером 1 Тб, то:

1. Займет очень много времени
2. Может не хватить памяти для хранения map

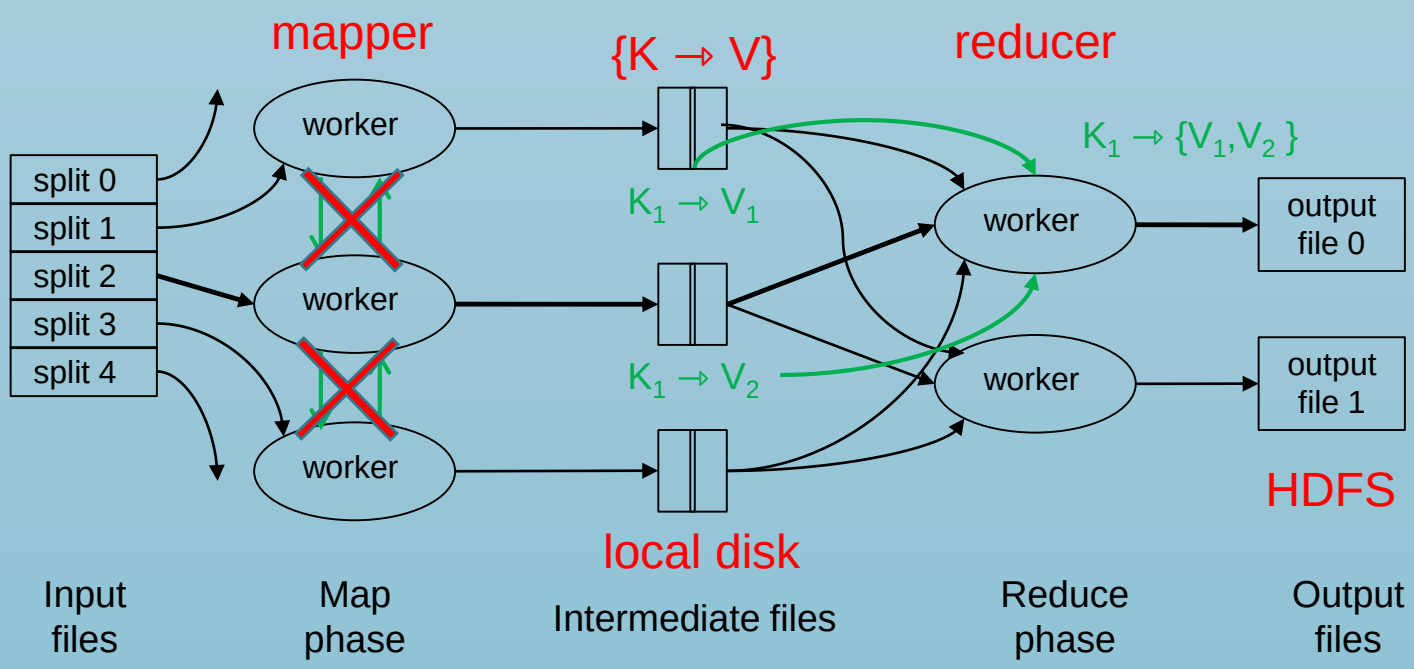
Подход Mapreduce



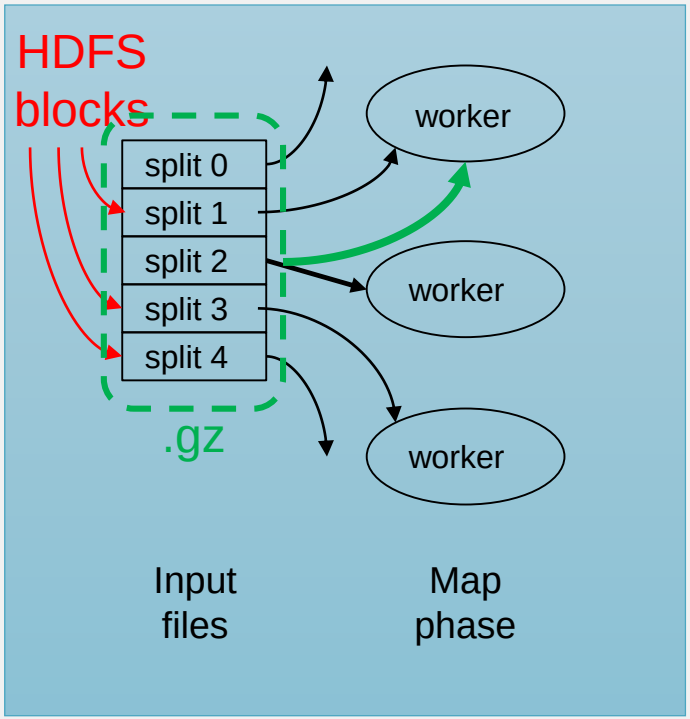


Парадигма MapReduce

Схема Map-Reduce

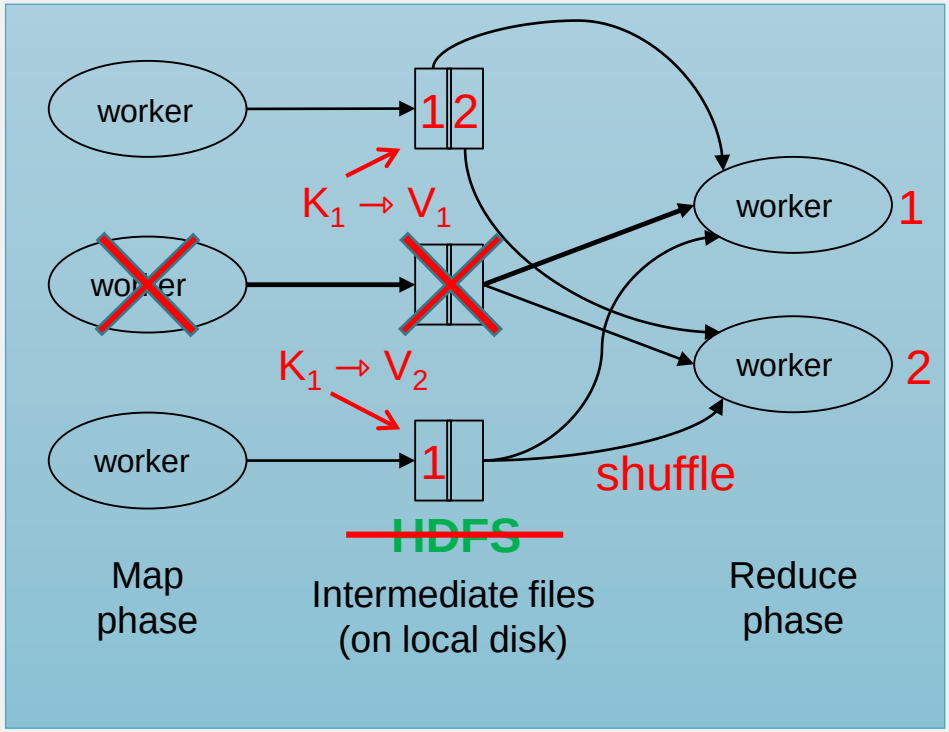


Входные данные



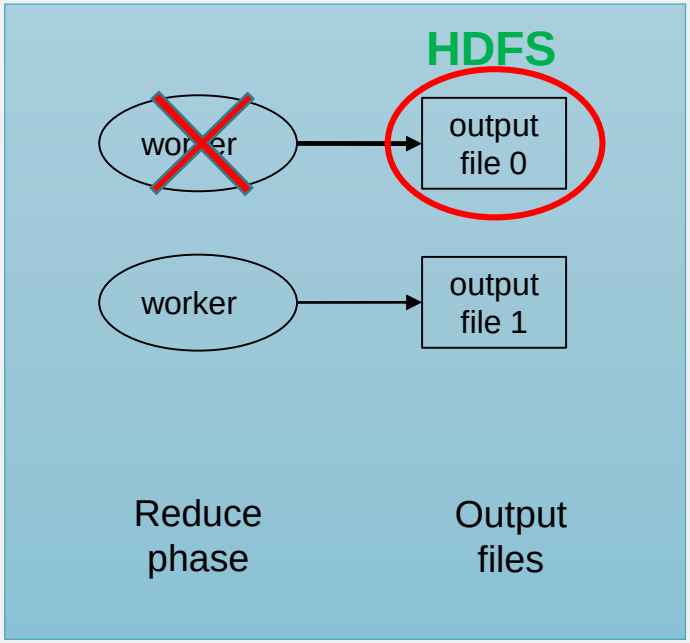
- Входные данные должны быть разделяемыми (**splitable**):
  - файл .gz не делется
  - Обычно split=block в hdfs
- Данные в каждом split должны быть независимыми
- Один worker обрабатывает один split (число worker = число split =? blocks)
- Worker обычно запускается там, где лежит его split (**data locality**)

Передача данных между Map и Reduce



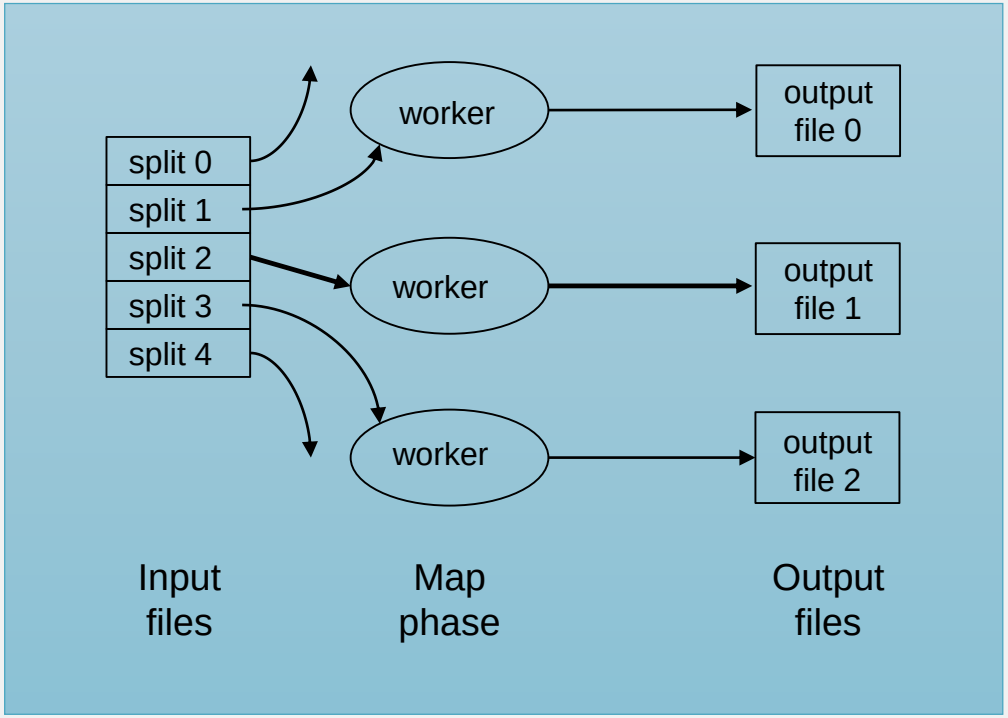
- Промежуточные данные пишутся на локальный диск
- Для каждого редьюсера маппер создает свой файл с данными
- Данные – это пара (Key, Value)
- Данные с одним Key попадают на один Reducer
- Редьюсеры начинают работать после завершения всех мапперов

Результат Map-Reduce задачи

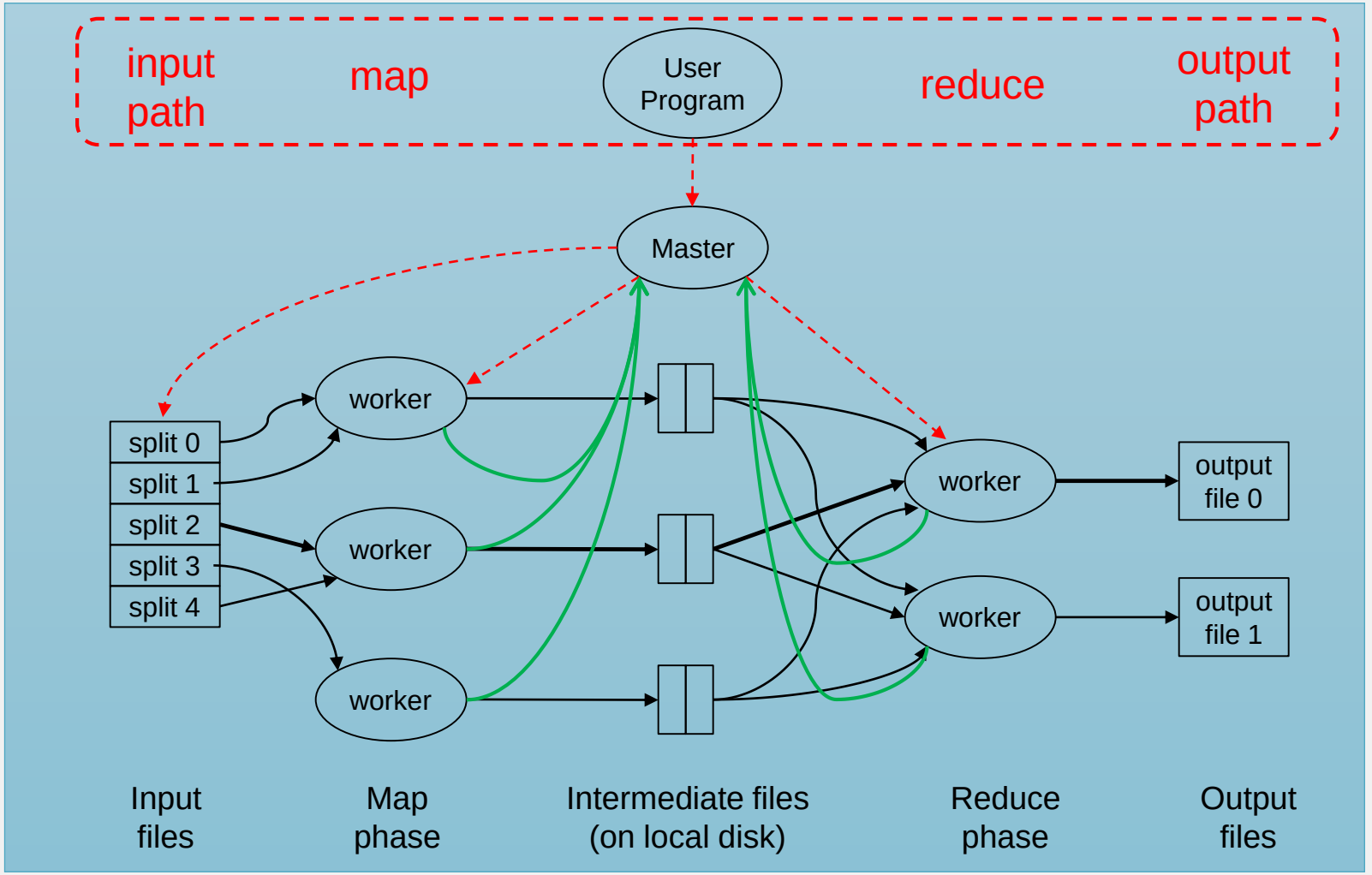


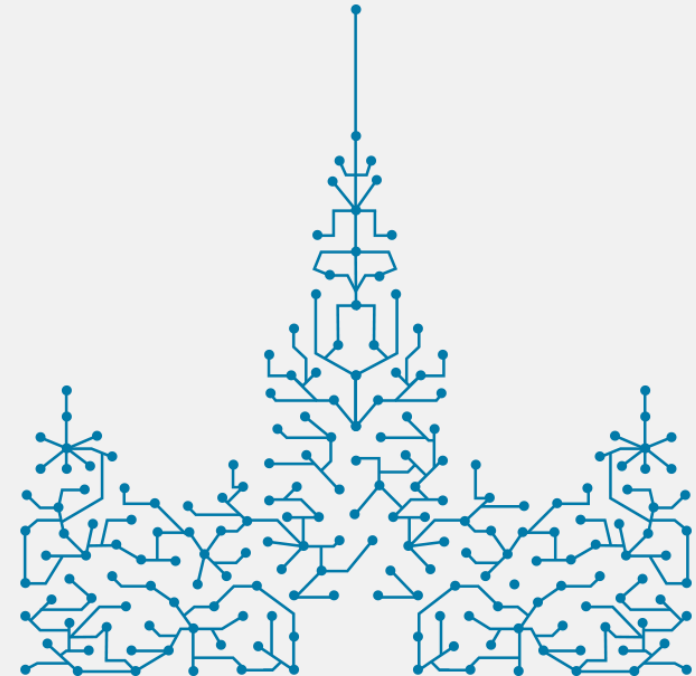
- Данные сохраняются в hdfs
- Каждый редьюсер пишет в свой файл
- Число редьюсеров задает пользователь
- Данные вида Key -> Value
- Формат данных определяется пользователем

Map-Reduce без Reduce



MapReduce workflow



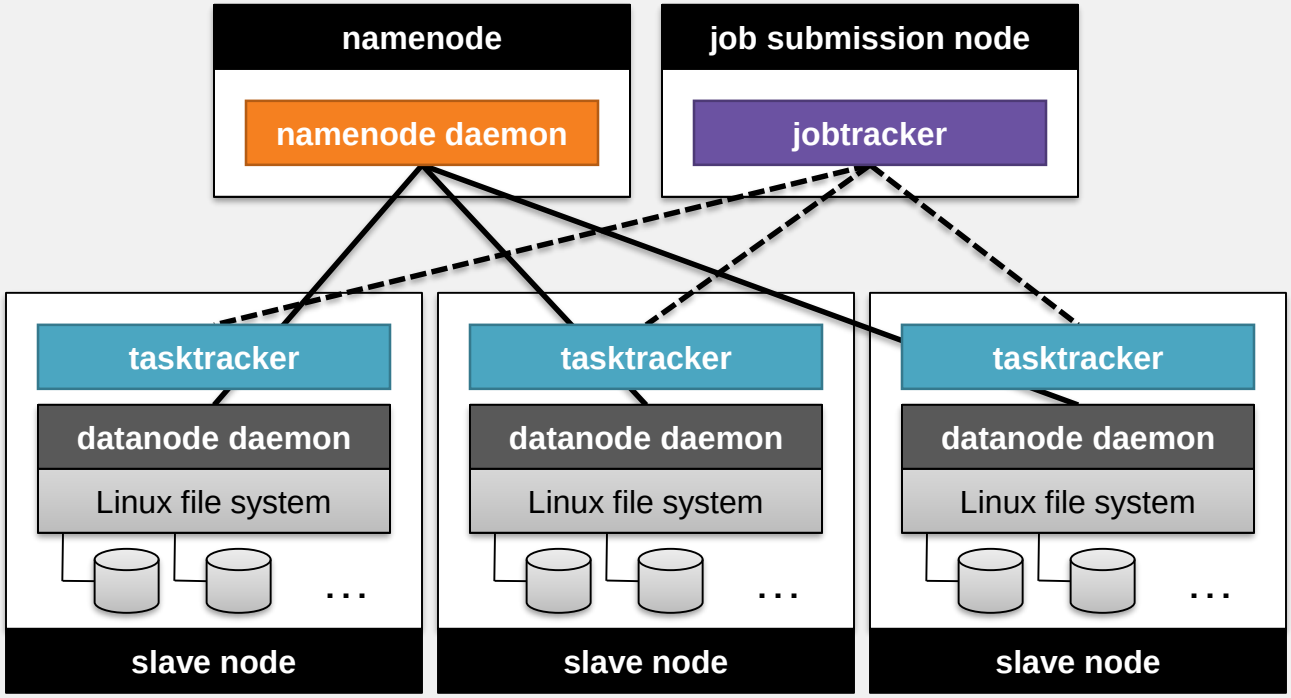


Фреймворк MapReduce

MapReduce Framework

- Управление запуском задач
- Управление “*data distribution*”
- Управление синхронизацией этапов MapReduce
- Обработка ошибок и отказов
- Все работает поверх HDFS

Hadoop MapReduce & HDFS



JobTracker

- Управляет запуском тасков и определяет, на каком ***TaskTracker*** таск будет запущен
- Управляет процессом работы MapReduce задач (*jobs*)
- Мониторит прогресс выполнения задач
- Перезапускает зафейленные или медленные таски

TaskTracker

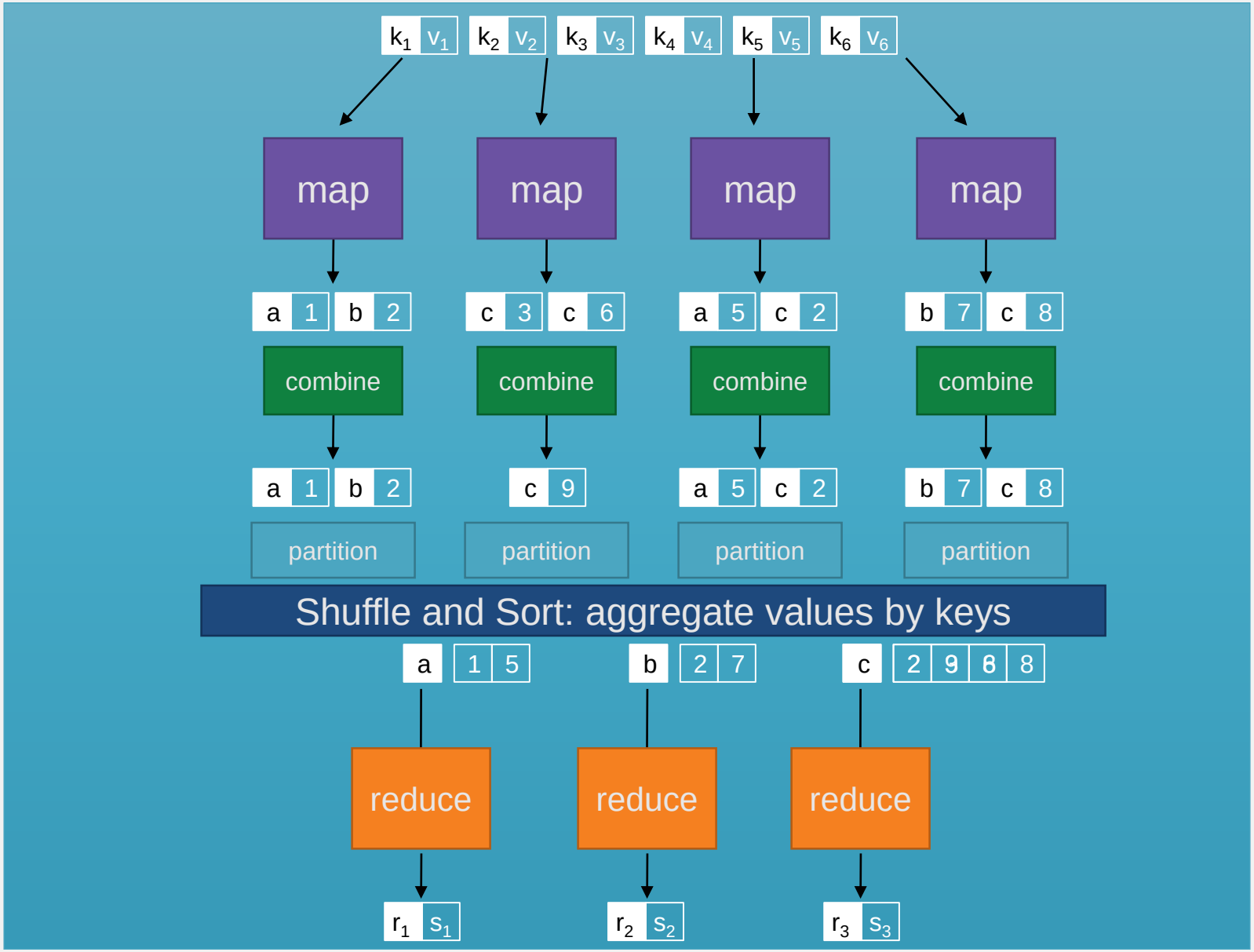
- Отвечает за работу всех worker на одном сервере
- Получает от **JobTracker** информацию о том, какой worker на каких данных нужно запустить
- Посылает в **JobTracker** статистику о прогрессе выполнения задачи (counters)
- Сообщает в **JobTracker** об удачном завершении или падении worker

Система слотов

- Для каждого TaskTracker определяется число слотов (slots)
- Таск запускается на одном слоте
- M мапперов + R редьюсеров = N слотов
- Для каждого слота определяется кол-во потребляемой ОЗУ

Пример:

- **100** серверов по **24** ядра
- **22** ядра доступны для MR задач
- Выделяем **14** под мапперы, **8** под редьюсеры
- Всего **2200** слота: **1400** для мапперов, **800** для редьюсеров


k1 v1
k2 v2
k3 v3
k4 v4
k5 v5
k6 v6
map
map
map
map
a 1 b 2
c 3 c 6
a 5 c 2
b 7 c 8
combine
combine
combine
combine
a 1 b 2
c 9
a 5 c 2
b 7 c 8
partition
partition
partition
partition
Shuffle and Sort: aggregate values by keys
a 1 5
b 2 7
c 2 9 8 8
reduce
reduce
reduce
r1 s1
r2 s2
r3 s3

Основные функции MapReduce

map (k1, v1) → list(k2, v2)

reduce (k2, list(v2) → list(k3, v3)

Опциональные функции MapReduce

partition (k2, number of partitions) → № of reducer

- Распределяет ключи по редьюсерам
- Часто просто хеш от key: *hash(k2) mod n*

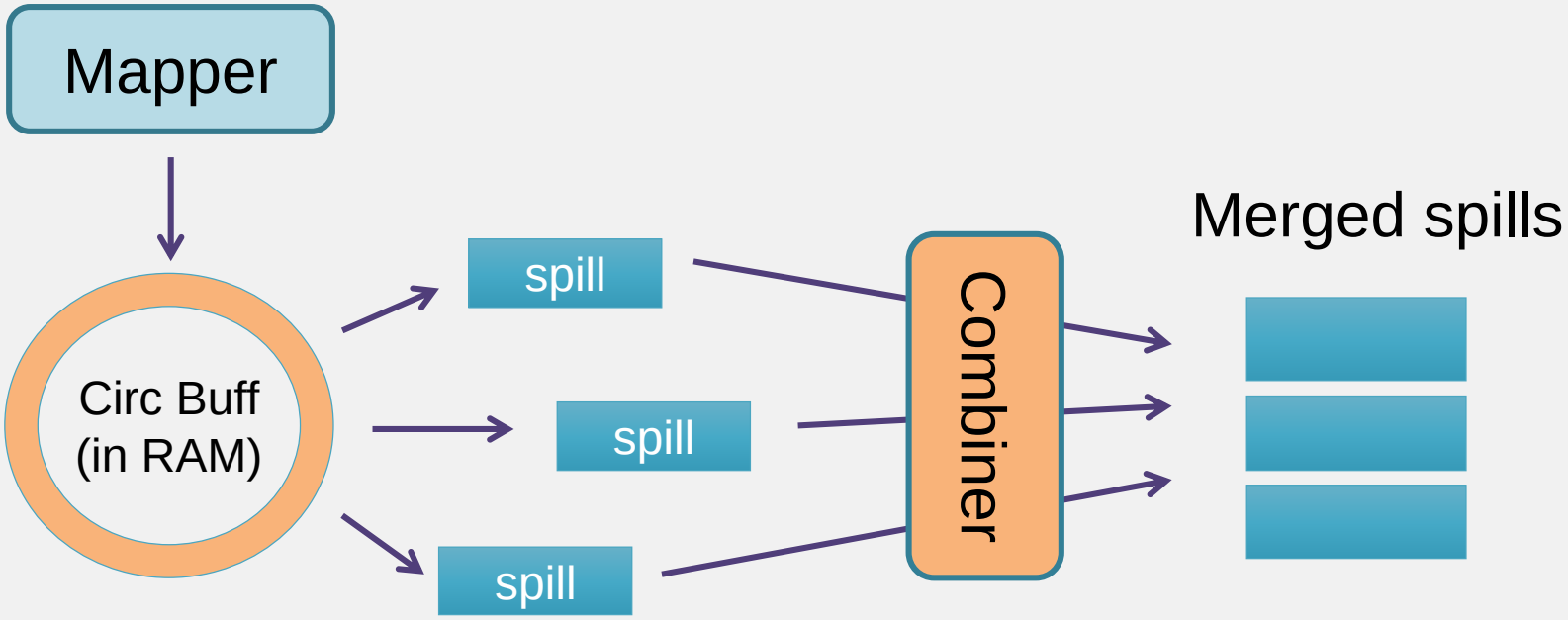
combine (k2, v2) → list(k2, v2)

- Мини-reducers которые выполняются после завершения фазы map
- Используется в качестве оптимизации для снижения сетевого трафика на reduce
- (!) Не должен менять тип ключа и значения

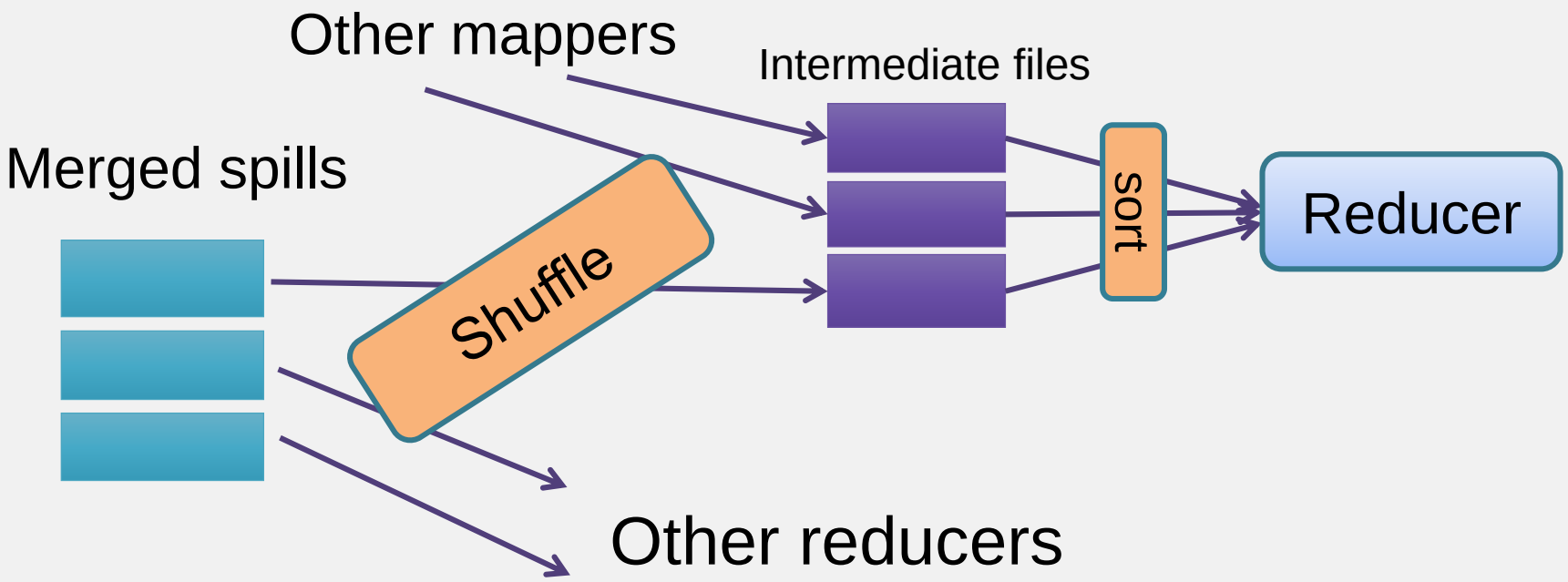
Shuffle и Sort в Hadoop

- На стороне ***Map***
  - Выходные данные буферизуются в памяти в циклическом буфере
  - Когда размер буфера достигает предела, данные “скидываются” (*spilled*) на диск
  - Затем все такие “сброшенные” части объединяются (*merge*) в один файл, разбитый на части
    - Внутри каждой части данные отсортированы
    - ***Combiner*** запускается во время процедуры объединения
- На стороне ***Reduce***
  - Выходные данные от мапперов копируются на машину, где будет запущен редьюсер
  - Процесс сортировки (*sort*) представляет собой многопроходный процесс объединения (*merge*) данных от мапперов
    - Это происходит в памяти и затем пишется на диск
  - Итоговый результат объединения отправляется непосредственно на редьюсер

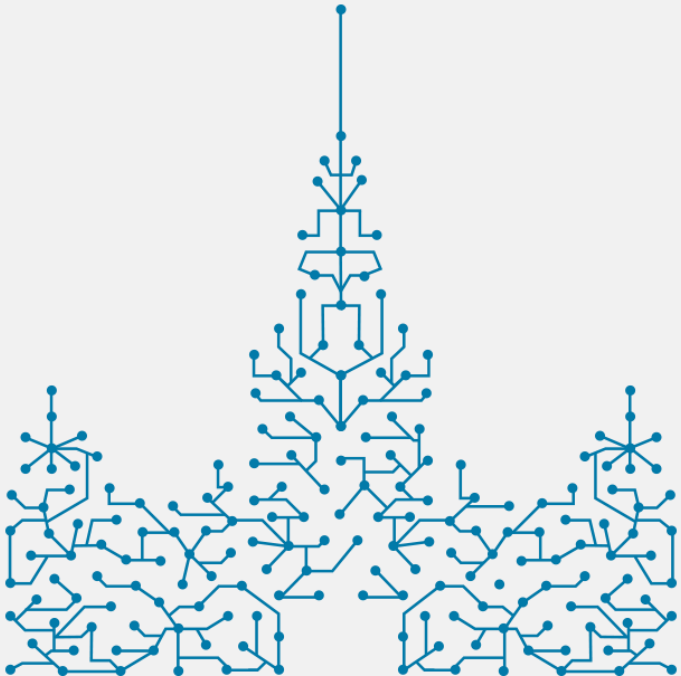
Shuffle и Sort в Hadoop



Shuffle и Sort в Hadoop



MapReduce API



Типы API

- *org.apache.hadoop.mapreduce*
  - Новое API, будем использовать в примерах
- *org.apache.hadoop.mapred*
  - Старое API, лучше не использовать

Класс Job

Содержит описание MapReduce задачи:

- *input/output* пути
- Формат *input/output* данных
- Указания классов для *mapper, reducer, combiner* и *partitioner*
- Типы значений пар *key/value*
- Количество редьюсеров

Класс Mapper

- *void setup(Mapper.Context context)*
  - Вызывается один раз при запуске таска
- *void map(K key, V value, Mapper.Context context)*
  - Вызывается для каждой пары key/value из *input split*
- *void cleanup(Mapper.Context context)*
  - Вызывается один раз при завершении таска

Класс Reducer/Combiner

- *void setup(Reducer.Context context)*
  - Вызывается один раз при запуске таска
- *void reduce(K key, Iterable<V> values, Reducer.Context context)*
  - Вызывается для каждого *key*
- *void cleanup(Reducer.Context context)*
  - Вызывается один раз при завершении таска

Класс Partitioner

int getPartition(K key, V value, int numPartitions)

- Возвращает номер reducer для ключа K

“Hello World”: Word Count

```
Map(String docid, String text):
    for each word w in text:
        Emit(w, 1);

Reduce(String term, Iterator<Int> values):
    int sum = 0;
    for each v in values:
        sum += v;
    Emit(term, sum);
```

WordCount: Configure Job

- Создание объекта Job:

```
Job job = Job.getInstance(getConf(), "WordCount");
```

- Определение jar для задачи:

```
job.setJarByClass(getClass());
```

WordCount: Configure Job

Определение input:

```
TextInputFormat.addInputPath(job, new Path(args[0]));
job.setInputFormatClass(TextInputFormat.class);
```

- в качестве пути может быть файл, директория, шаблон пути (*/path/to/dir/test_**)
- *TextInputFormat* читает входные данные как текстовый файл:
  - key – LongWritable
  - value – Text

WordCount: Configure Job

Определение output:

```
TextOutputFormat.setOutputPath(job, new Path(args[1]));
job.setOutputFormatClass(TextOutputFormat.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
```

Если типы для *map* и *reduce* отличаются, то:

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

WordCount: Configure Job

- Определение класса для **Mapper** и **Reducer**:

```
job.setMapperClass(WordCountMapper.class);
job.setReducerClass(WordCountReducer.class);
```

- Определение класса для **Combiner**:

```
job.setCombinerClass(WordCountReducer.class);
```

WordCount: запуск задачи

Запускает задачу и ждет ее окончания:

```
job.waitForCompletion(true);
```

- *true* в случае успеха, *false* в случае ошибки

WordCountJob

```
public class WordCountJob extends Configured implements Tool {
        @Override
        public int run(String[] args) throws Exception {
                Job job = Job.getInstance(getConf(), "WordCount");
                job.setJarByClass(getClass());

                TextInputFormat.addInputPath(job, new Path(args[0]));
                job.setInputFormatClass(TextInputFormat.class);

                job.setMapperClass(WordCountMapper.class);
                job.setReducerClass(WordCountReducer.class);
                job.setCombinerClass(WordCountReducer.class);

                TextOutputFormat.setOutputPath(job, new Path(args[1]));
                job.setOutputFormatClass(TextOutputFormat.class);
                job.setOutputKeyClass(Text.class);
                job.setOutputValueClass(IntWritable.class);

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

WordCountJob

```
public class WordCountJob extends Configured implements Tool {
        public static void main(String[] args) throws Exception {
                int exitCode = ToolRunner.run(
                        new WordCountJob(), args);
                System.exit(exitCode);
        }
}
```

WordCount: Mapper

- Наследник класса:

```
public class Mapper <KEYIN, VALUEIN, KEYOUT, VALUEOUT>
```

- Должен быть реализован метод map():

```
void map(KEYIN key, VALUEIN value, Context context){}
```

- Типы key / value (из org.apache.hadoop.io):
  - IntWritable
  - Text
  - ImmutableBytesWritable

WordCount: Mapper

```
public class WordCountMapper
        extends Mapper<LongWritable, Text, Text, IntWritable> {

    private final static IntWritable one = new IntWritable(1);
    private final Text word = new Text();

    @Override
    protected void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
        StringTokenizer tokenizer =
                                = new StringTokenizer(value.toString());

        while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            context.write(word, one);
        }
    }
}
```

WordCount: Reducer

- Наследник класса:

```
public class Reducer<KEYIN, VALUEIN, KEYOUT, VALUEOUT>
```

- Должен быть реализован метод reduce():

```
void reduce(KEYIN key, Iterable<VALUEIN> values,
            Context context){}
```

- Типы входных данных в Reducer должны совпадать с типами выходных данных Mapper

WordCount: Reducer

```
public class WordCountReducer
    extends Reducer<Text, IntWritable, Text, IntWritable> {

    @Override
    protected void reduce(Text key,
        Iterable<IntWritable> values, Context context)
            throws IOException, InterruptedException {

        int sum = 0;
        for (IntWritable value : values) {
            sum += value.get();
        }
        context.write(key, new IntWritable(sum));
    }
}
```

Reducer в качестве Combiner

- Меньше данных отправляется на ***Reducer***
- Типы key/value в output у ***Reducer*** и ***Mapper*** равны
- Фреймворк MapReduce не гарантирует вызов ***Combiner***
  - Нужно только с точки зрения оптимизации
  - Логика приложения не должна зависеть от вызова вызов ***Combiner***

Типы данных в Hadoop

- ***Writable***
  - Определяет протокол де-сериализации. Каждый тип в Hadoop должен быть ***Writable***
- ***WritableComprable***
  - Определяет порядок сортировки. Все ключи должны быть ***WritableComprable*** (но не значения!)
- ***Text, IntWritable, LongWritable*** и т.д.
  - Конкретные реализации для конкретных типов
- ***SequenceFiles***
  - Бинарно-закодированная последовательность пар *key/value*

Комплексные типы данных в Hadoop

Простой способ:

- Закодировать в ***Text***: (x, 10) = “x:10”
- Для раскодирования нужен специальный метод парсинга
- Просто, работает, но…

Сложный (правильный) способ:

- Определить реализацию своего типа ***Writable(Comprable)***
- Необходимо реализовать методы ***readFields, write, (compareTo)***
- Более производительное решение, но сложнее в реализации

Класс InputSplit

- ***Split*** – это набор логически организованных записей
  - Строки в файле
  - Строки в выборке из БД
- Каждый экземпляр *Mapper* обрабатывает один split
  - Функция *map(k, v)* вызывается для каждой записи из split
- Сплиты реализуются расширением класса ***InputSplit***:
  - FileSplit
  - TableSplit

Класс InputFormat

- Создает *input splits*
- Определяет, как читать каждый *split*

```
public abstract class InputFormat<K, V> {
    public abstract List<InputSplit> getSplits(JobContext context)
                        throws IOException, InterruptedException;

    public abstract RecordReader<K,V> createRecordReader(InputSplit
split, TaskAttemptContext context )
                        throws IOException, InterruptedException;
}
```

Класс InputFormat

Готовые классы-реализации ***InputFormat***:

- **TextInputFormat**
  - LongWritable / Text
- **NLineInputFormat**
  - *NLineInputFormat.setNumLinesPerSplit(job, 100);*
- **DBInputFormat**
- **TableInputFormat** (HBASE)
  - ImmutableBytesWritable / Result
- **SequenceFileInputFormat**

Выбор нужного формата:

```
job.setInputFormatClass(*InputFormat.class);
```

Класс OutputFormat

- Определяет формат выходных данных
- Реализация интерфейса класса ***OutputFormat***
  - Проверяет output для задачи
  - Создает реализацию ***RecordWriter***
  - Создает реализацию ***OutputCommitter***

Класс OutputFormat

Готовые классы-реализации ***OutputFormat***:

- *TextOutputFormat*
- *DBOutputFormat*
- *TableOutputFormat* (HBASE)
- *SequenceFileOutputFormat*
- *NullOutputFormat*

Выбор нужного формата:

```
job.setOutputFormatClass(*OutputFormat.class);
job.setOutputKeyClass(*Key.class);
job.setOutputValueClass(*Value.class);
```

Запуск задач в Hadoop

$ hadoop jar <jar> [mainClass] args...

Generic Option	Описание
-conf <conf_file.xml>	Добавляет свойства конфигурации из указанного файла в объект *Configuration*
-Dproperty=value	Устанавливает значение свойства конфигурации в объекте *Configuration*
-files <file,file,file>	Предоставляет возможность использовать указанные файлы в задаче MapReduce через *DistributedCache*
-libjars <f.jar, f2.jar>	Добавляет указанные jars к переменной CLASSPATH у тасков задачи и копирует их через *DistributedCache*

$ **hadoop jar** file.jar org.my.main.class **–files** dict.txt **–D** send.stat=true

Отладка задач в Hadoop

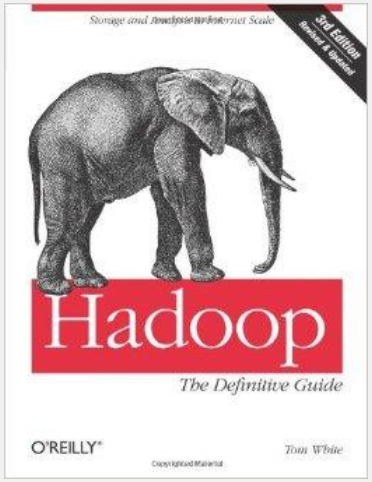
Логирование:

- ***System.out.println***
- Доступ к логам через веб-интерфейс
- Лучше использовать отдельный класс-логер (log4j)
- Аккуратней с количеством данных в логах

Использование счетчиков:

```
context.getCounter("GROUP", "NAME").increment(1);
```

Книги

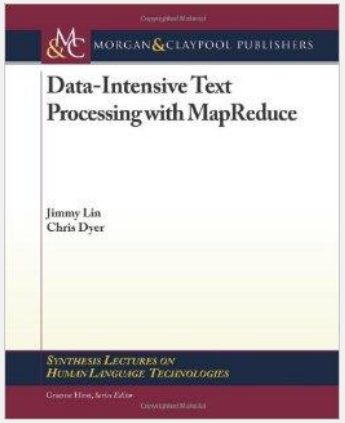


Hadoop: The Definitive Guide
Tom White (Author)
O'Reilly Media; 3rd Edition

Chapter 2: MapReduce
Chapter 5: Developing a MapReduce Application
Chapter 7: MapReduce Types and Formats

Data-Intensive Text Processing with MapReduce
Jimmy Lin and Chris Dyer (Authors) (April, 2010)

Chapter2: MapReduce Basics



ТЕХНОСФЕРА

Отмечайтесь и оставляйте отзыв

Спасибо за внимание!

Евгений Чернов

e.chernov@corp.mail.ru