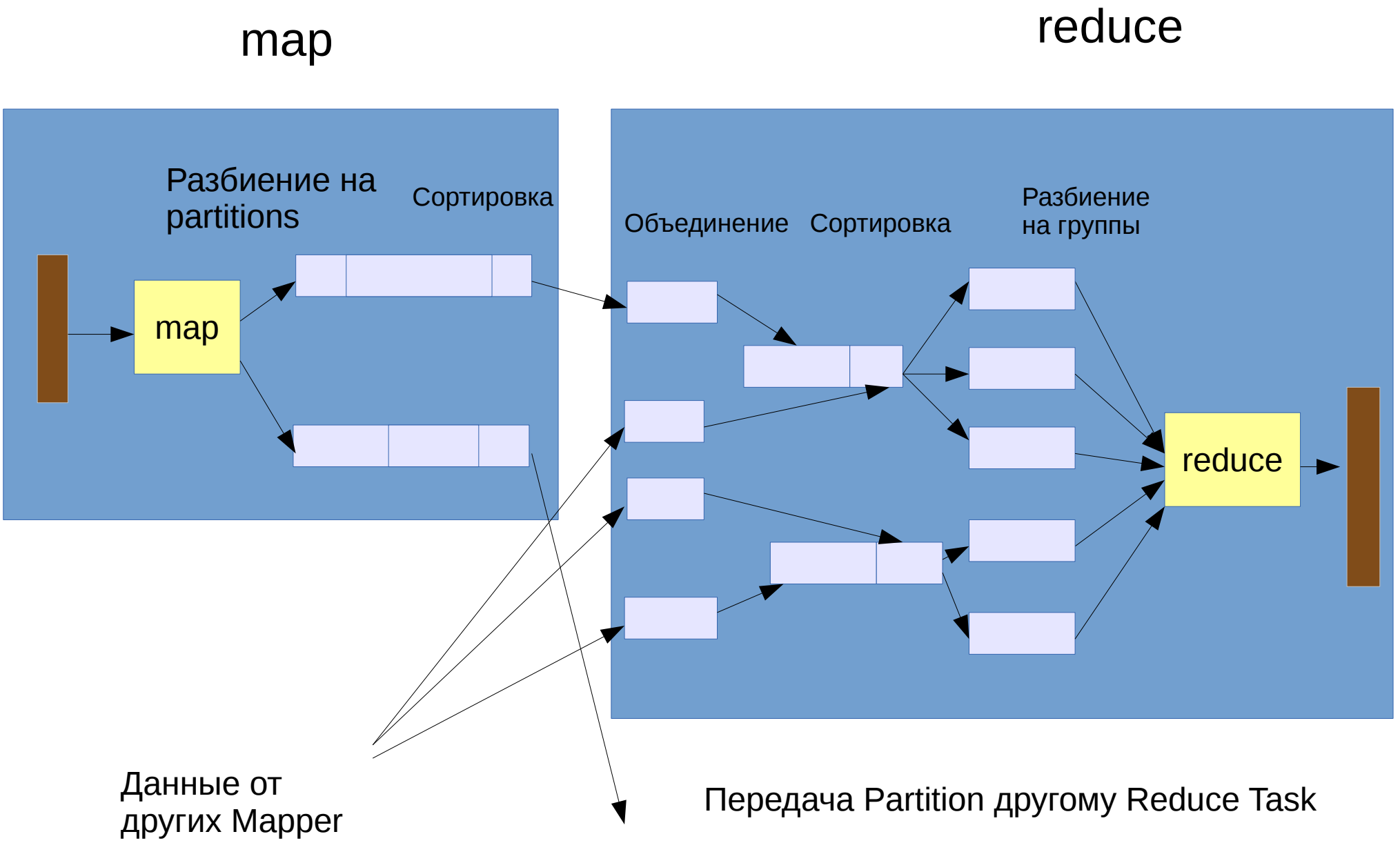
# Hadoop. MapReduce. Примеры

Возможности hadoop.mapreduce в реальных задачах

# Детали MapReduce



# Ход выполнения MAP

- Исходные данные для каждого Map task извлекаются с помощью InputFormat
- К исходным данным применяется функция MAP
- Результат на основании ключа и функции партиционирования разбивается на partitions (1 partition для каждого reduce task)
- Внутри partition производится сортировка по ключу
- Если задана функция combiner, то она применяется для каждого ключа и набора его значений
- Результат сохраняется на локальный жесткий диск компьютера где выполняется Map task

# Partitioner

- Позволяет вручную определить partition для записи сформированной функцией map

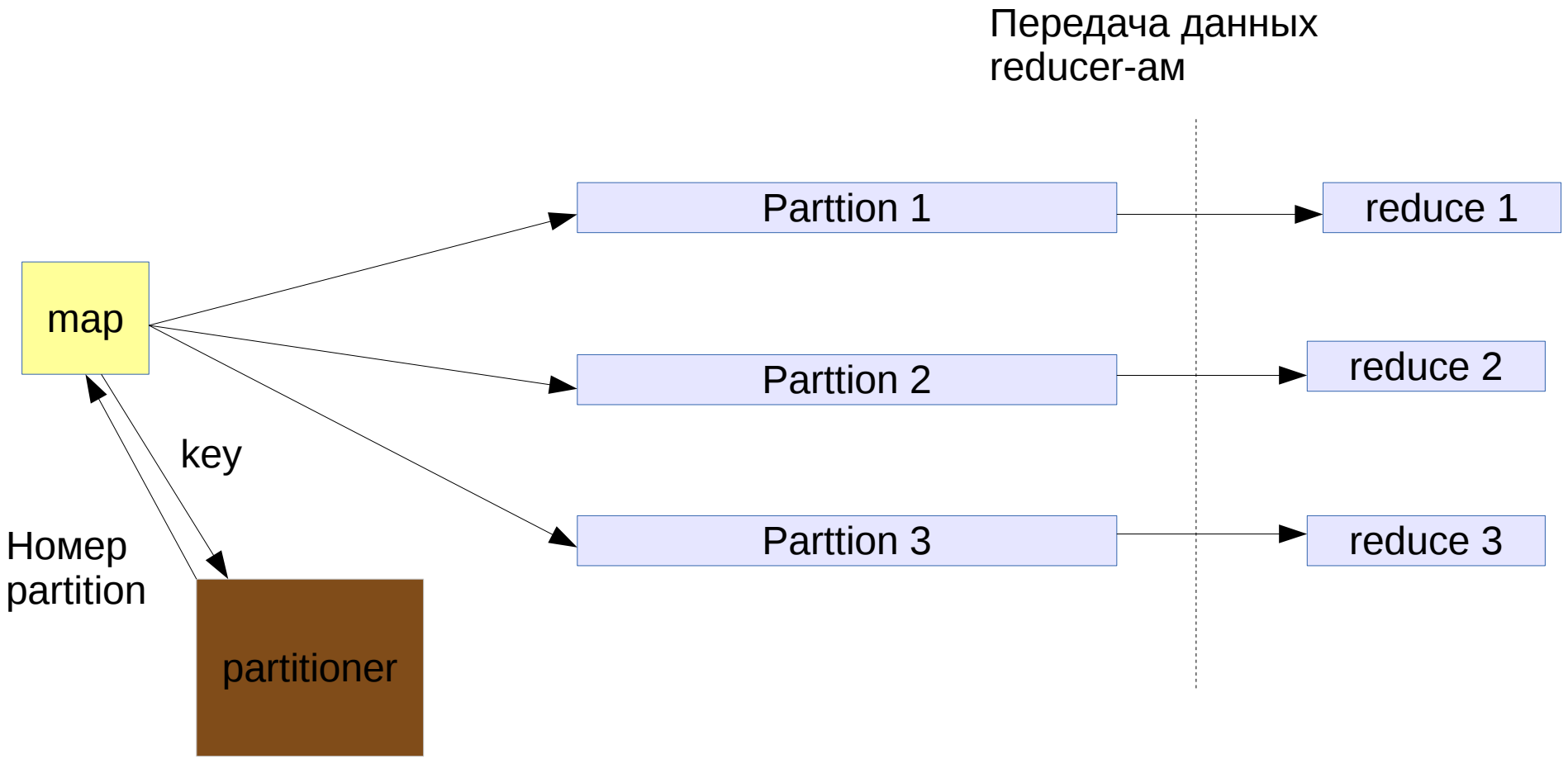
```
package org.apache.hadoop.mapreduce;
public abstract class Partitioner<KEY, VALUE> {
  public abstract int getPartition(KEY key, VALUE value, int numPartitions);
}
```

- Базовый partitioner работающий по умолчанию :

```
public class  HashPartitioner<K, V> extends Partitioner<K, V> {
  public int getPartition(K key, V value,  int numReduceTasks) {
    return (key.hashCode() & Integer.MAX_VALUE) % numReduceTasks;
  }
}
```

-

# Partitioner



# Ход выполнения Reduce

- Task содеиняется со всеми Map Task и скачивает себе свои partitions
- Выполняется процедура merge:

  -Все partitions объединяются.

  -Происходит разбиение на группы. Либо по ключу, либо используя заданную функцию группировки

  -Группы сортируются
- Для каждой группы применяется функция reduce
- Результат сохраняется в HADOOP с помощью OutputFormat

# Grouping Comparator

- Предназначен для разбития исходных данных reduce на группы
- Кажда группа подается на вход функции reduce независимо от других групп.
- Настраивается при создании job

  job.setGroupingComparatorClass(MyComparator.class);

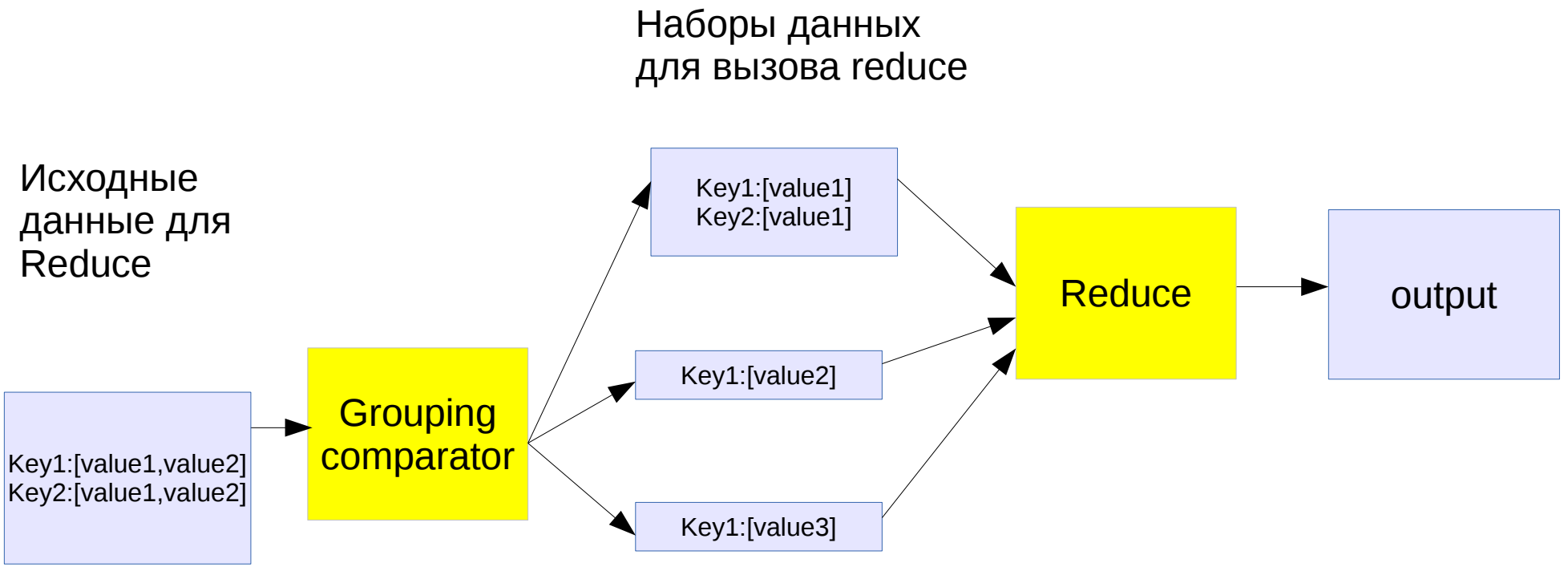
  –

  По умолчанию разбиение на гурппы не осуществляется – все входные данные reduce приходят в одном вызове
- Базовый класс

```
public class WritableComparator implements RawComparator {
...
 public int compare(WritableComparable a, WritableComparable b) {
    return a.compareTo(b);
 }}
```

# Grouping comparator

## позволяет изменить выборку для Redece



# Частичная сортировка (partial sort)

- Данные отсортированы по ключу только внутри одного файла output (для каждого reduce)
- Для ряда задач (напрмер для индексов) этого достаточно
- Partial sort осуществляется по умолчанию в MapReduce – дополнительный код не требуется.

# Полная сортировка

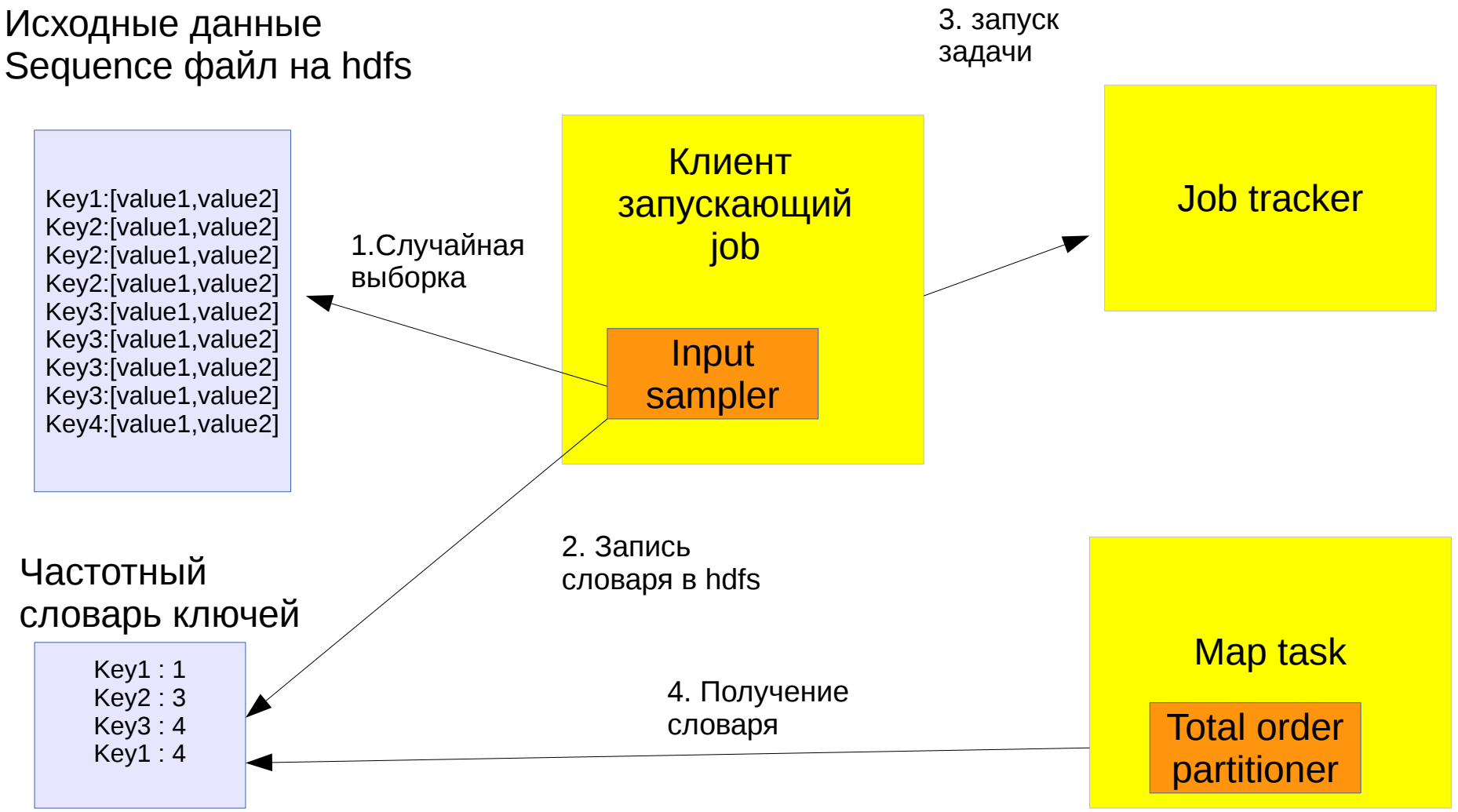
- К частичной сортировке добавляем контроль над разбиением вывода Map на partitions.
- 1 вариант - с помощью заранее определенного алгоритма (например весь возможный диапазон ключей делим на N)
- 2 вариант – используем Sampler

  перед запуском mapreduce исследуем множество ключей для оптимального разбиения на Partitions

# Немного про InputSampler

- Выполняется перед запуском job
- Читает исходные данные с большим интервалом и выбирает ключи
- Создает файл описыающий распределение ключей
- Требует исходных данных со значимыми ключами (например SequenceInputFormat)
- Ключи на выходе функции map должны совпадать с ключами входного файла
- TotalOrderPartitioner при инициализации считывает файл подготовленный InputSampler и опирается на него в разбиении данных на partitions

# Input sampler



# Пример полной сортировки

```
Job job = Job.getInstance();

job.setJarByClass(SortJob.class);

job.setJobName("Full sort");

job.setInputFormatClass(SequenceFileInputFormat.class
);

SequenceFileInputFormat.addInputPath(job, new
Path(args[0]));

FileOutputFormat.setOutputPath(job, new Path(args[1]));

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(ServiceCall.class);

job.setNumReduceTasks(2);

...
```

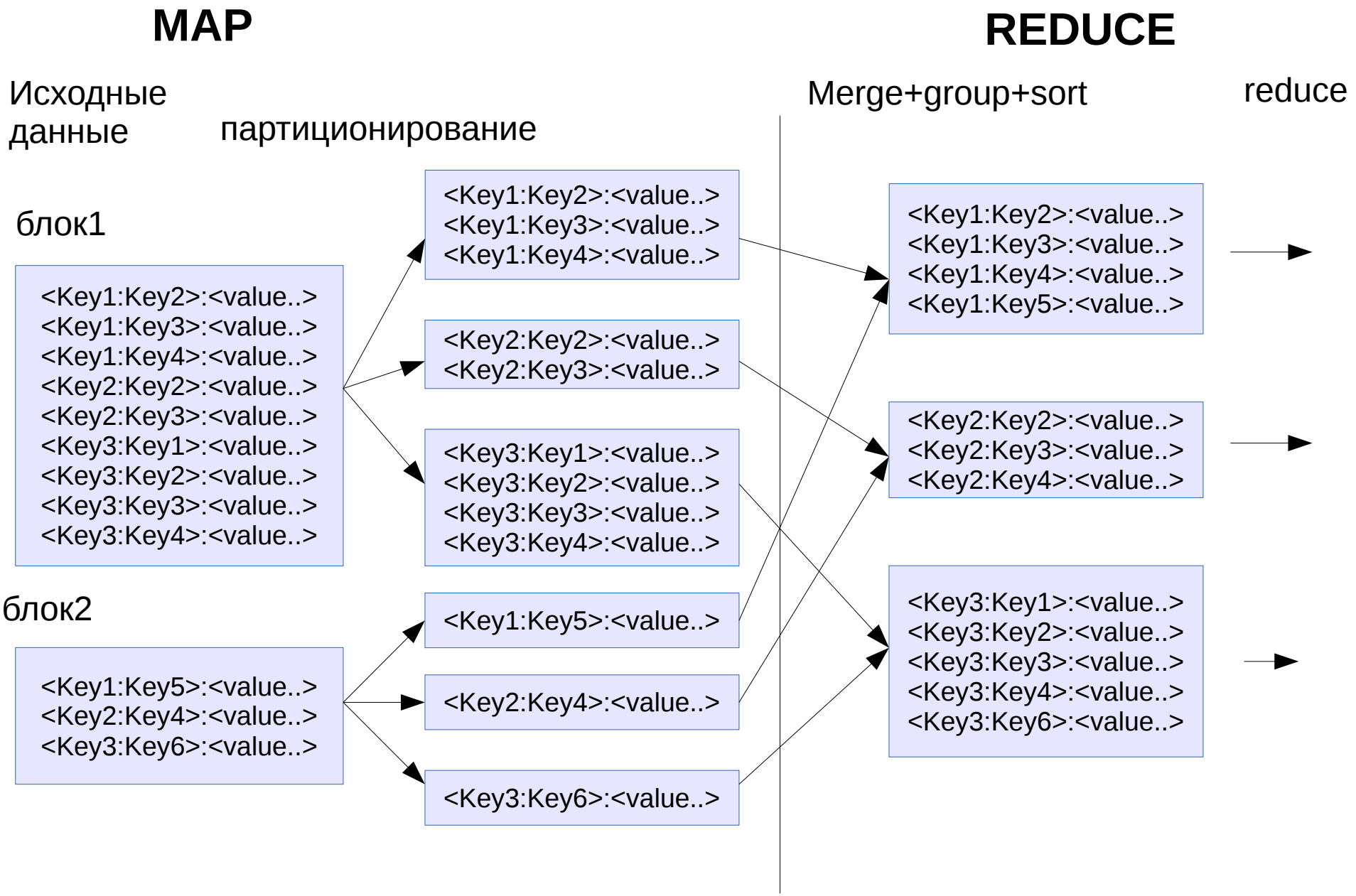
# Пример полной сортировки продолжение

```
...
job.setPartitionerClass(TotalOrderPartitioner.class);
InputSampler.Sampler<Text, ServiceCall> sampler = new
InputSampler.RandomSampler<Text, ServiceCall>(0.5, 10, 10);
Path input = FileInputFormat.getInputPaths(job)[0];
input = input.makeQualified(input.getFileSystem( job.getConfiguration() ));
Path partitionFile = new Path(input.getParent(), "_partitions");
TotalOrderPartitioner.setPartitionFile(job.getConfiguration(),partitionFile);
InputSampler.<Text, ServiceCall>writePartitionFile(job, sampler);
URI partitionUri = new URI(partitionFile.toString() + "#_partitions");
DistributedCache.addCacheFile(partitionUri, job.getConfiguration());
DistributedCache.createSymlink(job.getConfiguration());
System.exit(job.waitForCompletion(true) ? 0 : 1);
```

# Сортировка по ключу и значению

- Используется в случаях когда reduce ожидает определенной последовательности value.
- В качестве ключа используется составной ключ – key+value
- Партиционирование делается аналогично полной сортировке но только по первой части составного ключа (key) – используется ручной partitioner
- Группировка делается по первой части ключа ручным GroupingComparator

# Пример


MAP
REDUCE
Исходные данные
партиционирование
Merge+group+sort
reduce
блок1
<Key1:Key2>:<value..>
<Key1:Key3>:<value..>
<Key1:Key4>:<value..>
<Key2:Key2>:<value..>
<Key2:Key3>:<value..>
<Key3:Key1>:<value..>
<Key3:Key2>:<value..>
<Key3:Key3>:<value..>
<Key3:Key4>:<value..>
блок2
<Key1:Key5>:<value..>
<Key2:Key4>:<value..>
<Key3:Key6>:<value..>
<Key1:Key2>:<value..>
<Key1:Key3>:<value..>
<Key1:Key4>:<value..>
<Key1:Key5>:<value..>
<Key2:Key2>:<value..>
<Key2:Key3>:<value..>
<Key2:Key4>:<value..>
<Key3:Key1>:<value..>
<Key3:Key2>:<value..>
<Key3:Key3>:<value..>
<Key3:Key4>:<value..>
<Key3:Key6>:<value..>

# Пример

```
public static class SecondarySortCallMapper
            extends Mapper<Text, ServiceCall, TextPair, ServiceCall> {
        protected void map(Text key, ServiceCall call, Context context)
                throws IOException, InterruptedException {
            context.write(new TextPair(call.getSystemB(), call.getSystemA()),  call);
        }
    }

public static class FirstPartitioner extends Partitioner<TextPair, ServiceCall> {
        public int getPartition(TextPair key, ServiceCall value, int numPartitions) {
            return (key.getFirst().hashCode() & Integer.MAX_VALUE) % numPartitions;
        }
    }
```

# Пример comparator

```
public static class FirstComparator extends WritableComparator {
        protected FirstComparator() {
            super(TextPair.class, true);
        }
        @Override
        public int compare(WritableComparable a1, WritableComparable b1) {
            TextPair a = (TextPair) a1;
            TextPair b = (TextPair) b1;
            return a.getFirst().compareTo(b.getFirst());
        }
    }
```

# Пример job

- Job job = Job.getInstance();
- job.setJarByClass(SecondarySortJobSimple.class);
- job.setJobName("Secondary sort");
- //настраиваем формат данных
- job.setInputFormatClass(SequenceFileInputFormat.class);
- SequenceFileInputFormat.addInputPath(job, new Path(args[0]));
- FileOutputFormat.setOutputPath(job, new Path(args[1]));
- job.setNumReduceTasks(2);
- //настраиваем mapper, partitioner, GroupingComparatorClass
- job.setMapperClass(SecondarySortCallMapper.class);
- job.setPartitionerClass(FirstPartitioner.class);
- job.setGroupingComparatorClass(TextPair.FirstComparator.class);
- //настраиваем типы данных
- job.setMapOutputKeyClass(TextPair.class);
- job.setOutputKeyClass(TextPair.class);
- job.setOutputValueClass(ServiceCall.class);
- System.exit(job.waitForCompletion(true) ? 0 : 1);