Big data

- *Big data*, *large data cloud*.
- Rozwiązania nastawione na zastosowanie w wielkoskalowych serwisach, np. webowych.
- Stosowane przez Google, Facebook, itd.

Architektura rozproszonych magazynów danych

Przetwarzanie danych

Dane strukturalne

Rozproszony magazyn

Transport/routing

Architektura rozproszonych magazynów danych

	Google	Apache Hadoop	Sector
Przetwarzanie danych	MapReduce	MapReduce	Sphere UDF
Dane strukturalne	BigTable	HBase/Hive	Space
Rozproszony magazyn	GFS	HDFS	SDFS
Transport/routing	—	—	UDT

Apache Hadoop

- Składniki systemu:
  - Distributed File System – rozproszony magazyn,
  - Map/Reduce – rozproszone przetwarzanie.
- Open-source, napisany w języku Java.
- Cross-platform.

HDFS: cechy

- Służy do przechowywania dużych *plików*.
- Fizycznie pliki przechowywane jako bloki o rozmiarze 64 lub 128 MB.
- Każdy blok replikowany na wiele węzłow.
- Automatyczna replikacja, dynamiczna zmiana współczynnika replikacji dla poszczególnych pików.

HDFS: dostęp

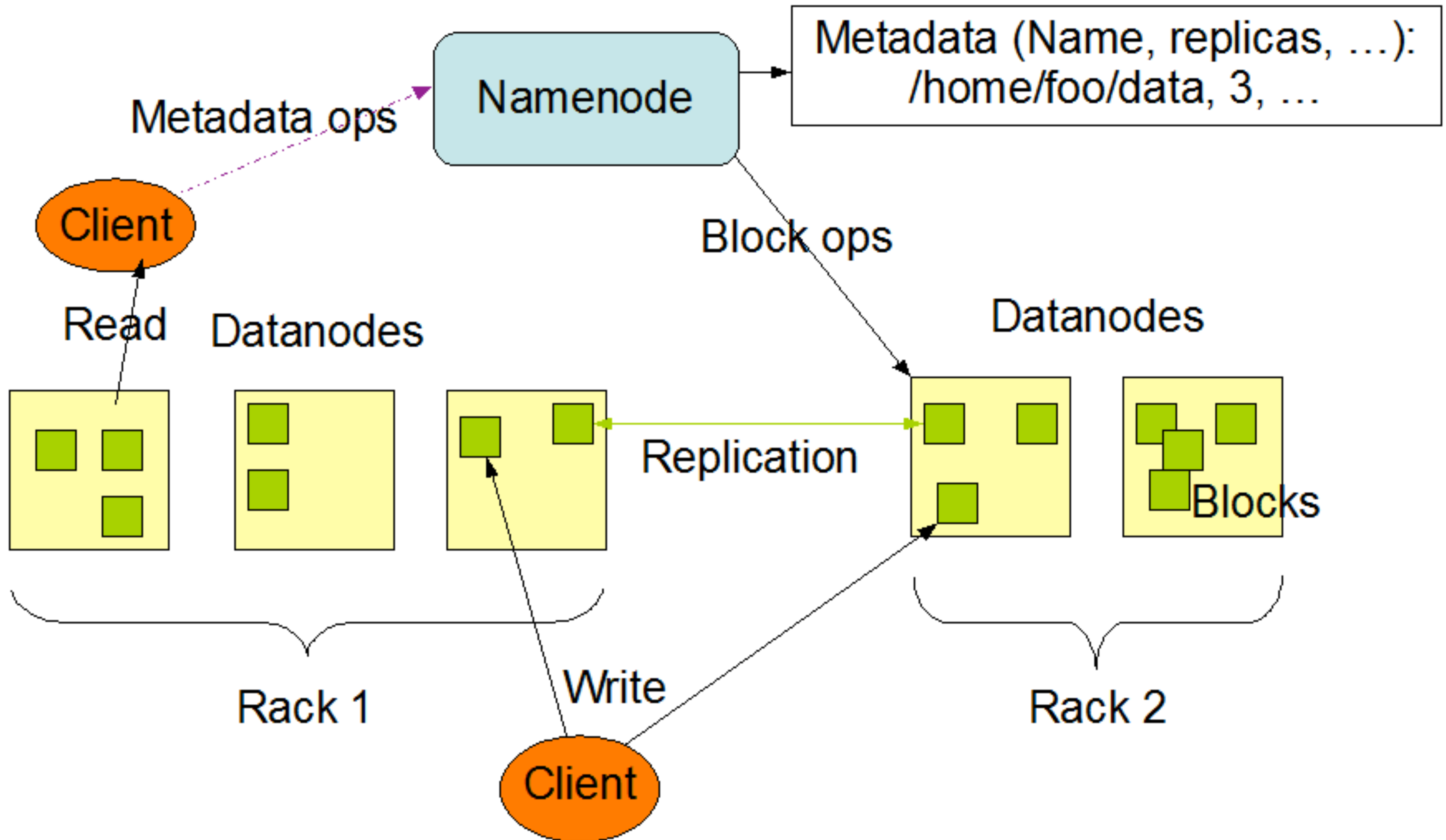
- Przy pomocy interfejsu command-line:

```
bin/hadoop fs -put my-file hdfs://node1:50070/foo/bar
```

- API Java lub C:

```
Path p = new Path("hdfs://node1:50070/foo/bar");
FileSystem fs = p.getFileSystem(conf);
DataOutputStream file = fs.create(p);
file.writeUTF("hello\n");
file.close();
```

Architektura HDFS



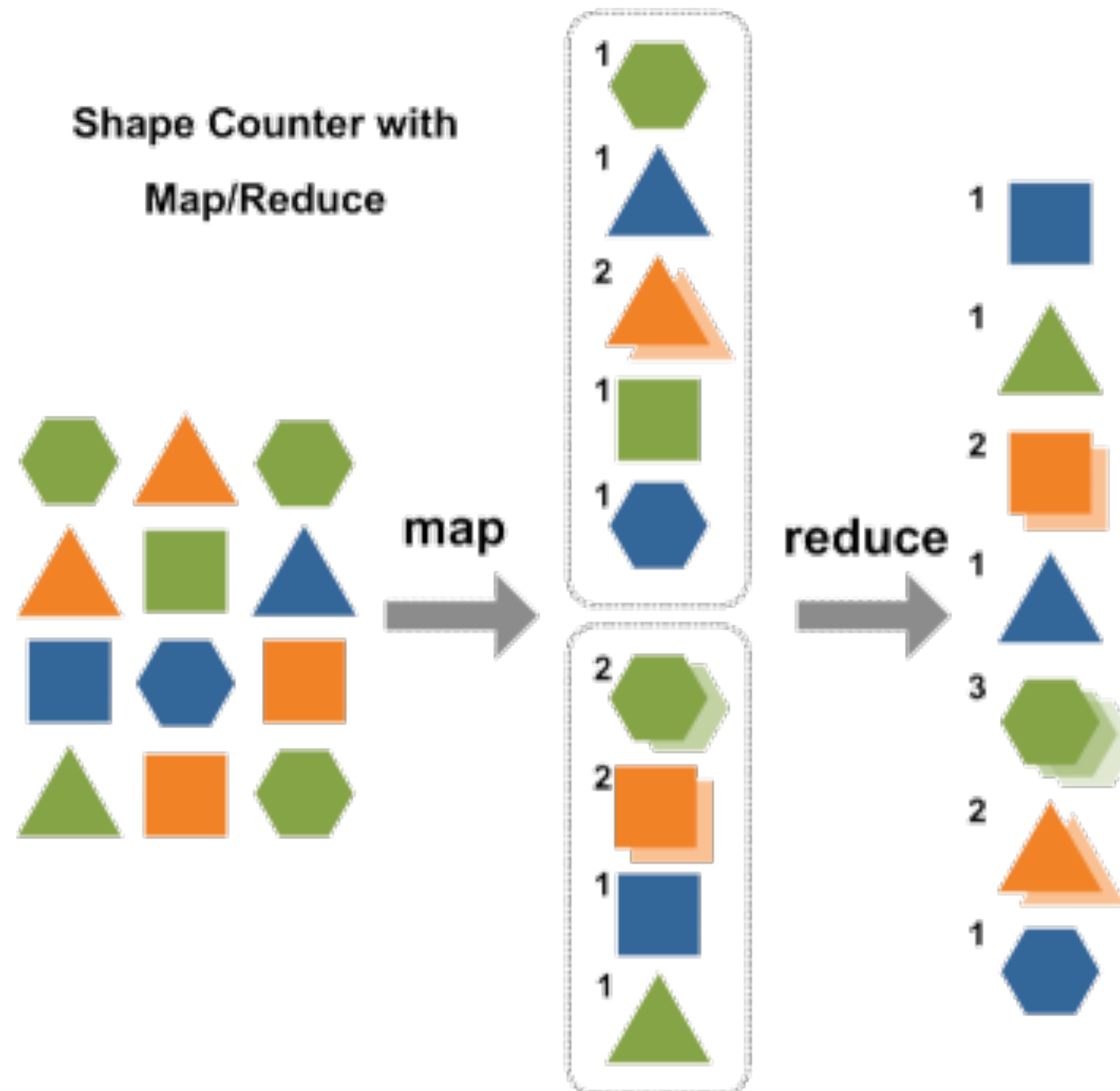
MapReduce

- Framework do łatwego tworzenia programów przetwarzających duże (kilka TB) zbiory danych.
- Przetwarzanie musi być odporne na usterki pomimo wykorzystywania „niepewnych” węzłów.
- Zwiększenie wydajności dzięki:
    - przetwarzaniu strumieniowemu (brak wyszukiwań),
    - tworzeniu potoków (*pipelining*).

MapReduce

- Dzieli dane wejściowe na niezależne „kawałki”, przetwarzane równolegle przez zadania *map*.
- Wyjście z zadań *map* jest sortowane przez framework i podawane na wejście zadań *reduce*.
- Framework zajmuje się szeregowaniem zadań, monitorowaniem i ponownym uruchamianiem w przypadku błędu.

MapReduce



http://www.gridgain.com/images/mapreduce_small.png

- Typowo, dane są przetwarzane na węźle, który je przechowuje – zwiększenie przepustowości.
- Jeden *JobTracker* na węźle *master* i po jednym *TaskTracker* na węzłach *slave*.
- W minimalnej konfiguracji, aplikacja dostarcza funkcji *map* i *reduce* poprzez implementację odpowiednich interfejsów Java.

MapReduce – nie tylko Java

- *Hadoop Streaming* pozwala na uruchamianie dowolnych programów (narzędzia lub skrypty shellowe) jako zadań *map* i *reduce*.
- *Hadoop Pipes* dostarcza API C++ (nie opartego na JNI) do implementacji aplikacji MapReduce.

MapReduce – I/O

- MapReduce przetwarza wyłącznie pary <*klucz, wartość*>.
- Klasy *klucz* i *wartość* muszą być serializowalne, a więc implemetować interfejs *Writable*.
- Klasa *klucz* musi implementować interfejs *WritableComparable* (do sortowania).

(input) `<k1, v1>` -> **map** -> `<k2, v2>` -> **combine** ->
-> `<k2, v2>` -> **reduce** -> `<k3, v3>` (output)

Map/Reduce: cechy

- Małe elementarne zadania *map* i *reduce*: lepsze równoważenie obciążenia, szybszy „powrót” po błędzie/porażce.
- Automatyczne ponowne uruchamianie: niektóre węzły są zawsze powolne lub niestabilne.
- Standardowy scenariusz: te same maszyny realizują przechowywanie i przetwarzanie.
- Optymalizacja lokalna: zadania *map* są w miarę możliwości alokowane do maszyn przechowujących ich dane wejściowe.

MapReduce: interfejsy

- **Mapper** – przetwarza pary wejściowe <k, v> na pary pośrednie: *map(WritableComparable, Writable, OutputCollector, Reporter)*
- **Reducer** – redukuje zbiór wartości pośrednich o tym samym kluczu do mniejszej liczby wartości: *reduce(WritableComparable, Iterator, OutputCollector, Reporter)*

MapReduce: interfejsy

- **Partitioner** – wprowadza podział kluczy pośrednich wartości. Domyślnie używana jest funkcja haszująca.
- **Reporter** – pozwala na raportowanie postępów przez funkcje *map* i *reduce*.
- **OutputCollector** – zbiera wyniki (pośrednie lub końcowe) zadań.

MapReduce: przykład

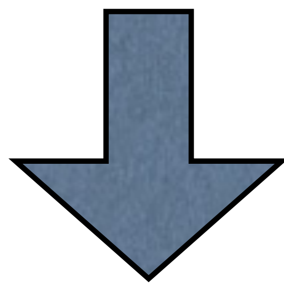
- Program zliczający częstotliwość występowania słów w plikach tekstowych.
- Wejście: katalog z plikami tekstowymi.
- Wyjście: plik tekstowy w formacie:
  słowo1 częstotliwość1
  słowo2 częstotliwość2
  ...

- Zadanie *map*: podziel kolejne linijki na słowa (przy pomocy *StringTokenizer*) i wygeneruj pary <*słowo*, *1*>.

```
public static class Map extends MapReduceBase implements
Mapper<LongWritable, Text, Text, IntWritable> {
  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  public void map(LongWritable key, Text value,
OutputCollector<Text, IntWritable> output, Reporter reporter) throws
IOException {
    String line = value.toString();
    StringTokenizer tokenizer = new StringTokenizer(line);
    while (tokenizer.hasMoreTokens()) {
      word.set(tokenizer.nextToken());
      output.collect(word, one);
    }
  }
}
```

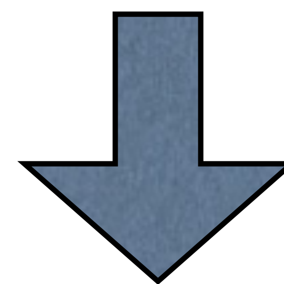
- Działanie zadania *map*:

`Hello World Bye World`



```
< Hello, 1>
< World, 1>
< Bye, 1>
< World, 1>
```

`Hello Hadoop Goodbye Hadoop`



```
< Hello, 1>
< Hadoop, 1>
< Goodbye, 1>
< Hadoop, 1>
```

- Zadanie *reduce*: sumuje wartości dla identycznych kluczy.

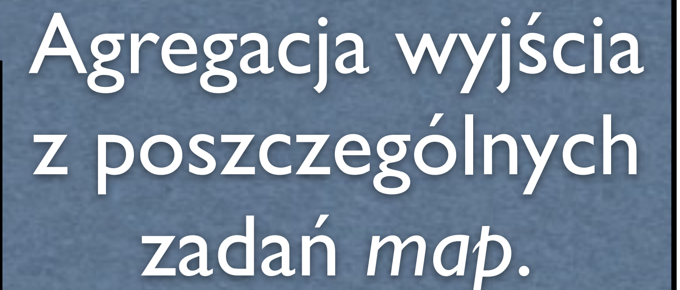
```
public static class Reduce extends MapReduceBase implements
Reducer<Text, IntWritable, Text, IntWritable> {
  public void reduce(Text key, Iterator<IntWritable> values,
OutputCollector<Text, IntWritable> output, Reporter reporter)
throws IOException {
    int sum = 0;
    while (values.hasNext()) {
      sum += values.next().get();
    }
    output.collect(key, new IntWritable(sum));
  }
}
```

- Całość jest „poskładana” w funkcji *main*:

```
public static void main(String[] args) throws Exception {
  JobConf conf = new JobConf(WordCount.class);
  conf.setJobName("wordcount");

  conf.setOutputKeyClass(Text.class);
  conf.setOutputValueClass(IntWritable.class);

  conf.setMapperClass(Map.class);
  conf.setCombinerClass(Reduce.class);
  conf.setReducerClass(Reduce.class);

  conf.setInputFormat(TextInputFormat.class);
  conf.setOutputFormat(TextOutputFormat.class);

  FileInputFormat.setInputPaths(conf, new Path(args[0]));
  FileOutputFormat.setOutputPath(conf, new Path(args[1]));

  JobClient.runJob(conf);
}
```

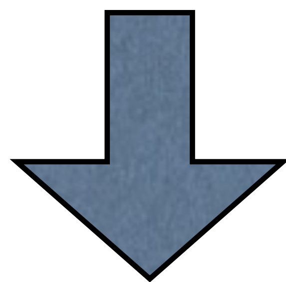
- Całość jest „poskładana” w funkcji *main*:

```
public static void main(String[] args) throws Exception {
  JobConf conf = new JobConf(WordCount.class);
  conf.setJobName("wordcount");

  conf.setOutputKeyClass(Text.class);
  conf.setOutputValueClass(IntWritable.class);

  conf.setMapperClass(Map.class);
  conf.setCombinerClass(Reduce.class);
  conf.setReducerClass(Reduce.class);

  conf.setInputFormat(TextInputFormat.class);
  conf.setOutputFormat(TextOutputFormat.class);

  FileInputFormat.setInputPaths(conf, new Path(args[0]));
  FileOutputFormat.setOutputPath(conf, new Path(args[1]));

  JobClient.runJob(conf);
}
```



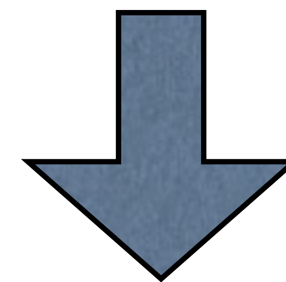
- Ponieważ określiliśmy klasę *Reducer* jako *combiner*, wyjście z poszczególnych *map* będzie wstępnie zagregowane:

```
< Hello, 1>
< World, 1>
< Bye, 1>
< World, 1>
```



```
< Bye, 1>
< Hello, 1>
< World, 2>
```

```
< Hello, 1>
< Hadoop, 1>
< Goodbye, 1>
< Hadoop, 1>
```



```
< Goodbye, 1>
< Hadoop, 2>
< Hello, 1>
```

HBase

- Baza danych oparta o Hadoop.
- *„Use it when you need random, realtime read/write access to your Big Data.”*
- Tryby pracy:
  - *Standalone* – w lokalnym systemie plików,
  - *Distributed* – korzysta z HDFS.

Hive

- Hurtownia danych dla Hadoop.
- Wykorzysuje MapReduce do przetwarzania, HDFS jako magazyn.
- Rozwijana przez Facebook.

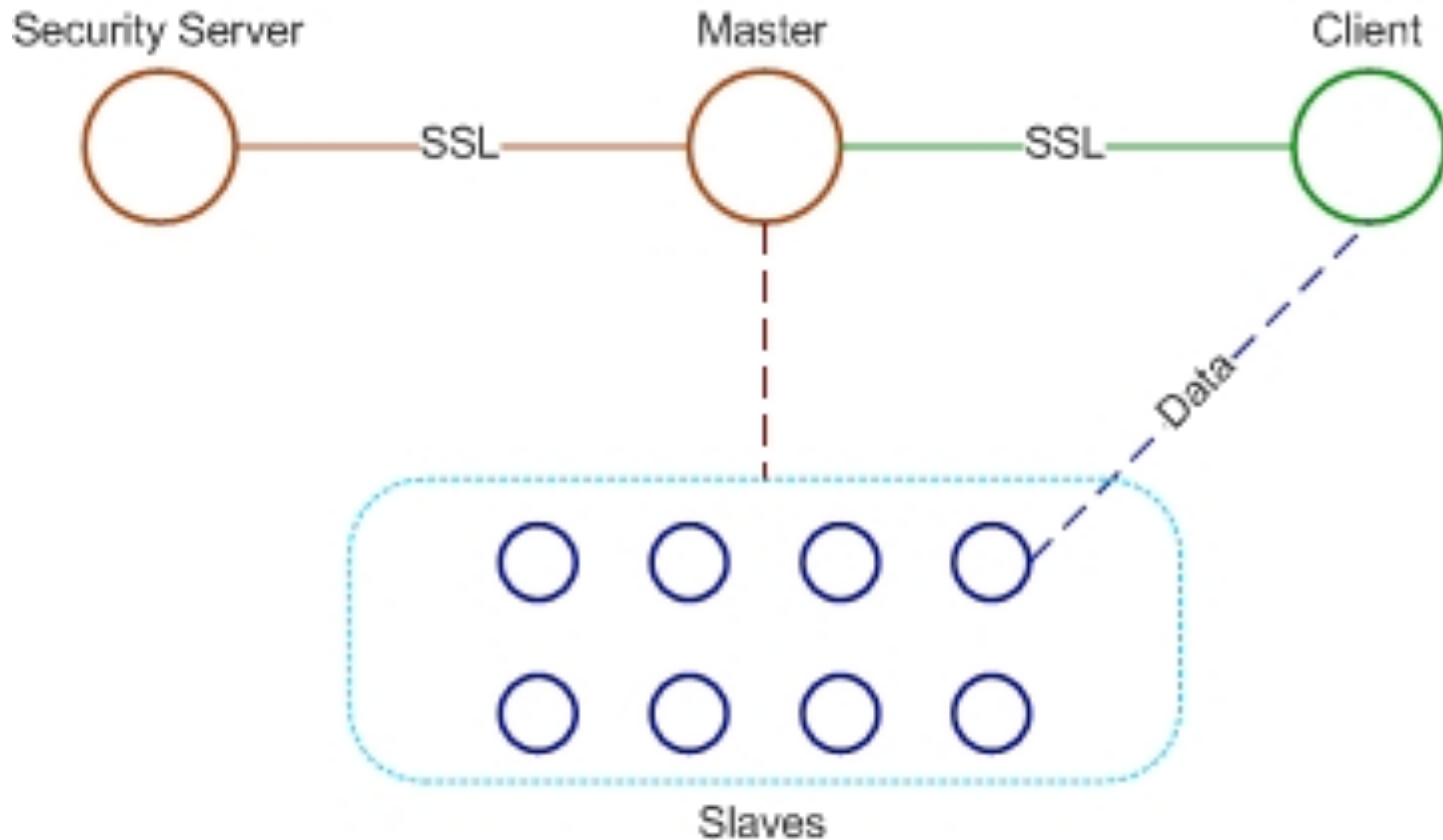
Architektura rozproszonych magazynów danych

	Google	Apache Hadoop	Sector
Przetwarzanie danych	MapReduce	MapReduce	Sphere UDF
Dane strukturalne	BigTable	HBase/Hive	Space
Rozproszony magazyn	GFS	HDFS	SDFS
Transport/routing	—	—	UDT

Sector/Sphere

- Rozwiązanie podobne do Hadoop pod względem koncepcji.
- Napisane w C++, a nie w Javie – szybsze przetwarzanie.
- Dodaje własny protokół transportowy – UDT (Hadoop – tylko TCP).

Sector/Sphere



Sector vs. Hadoop

- Nieco stronnicze porównanie...
- Benchmark Sector/Sphere vs. Hadoop
- Ale:
  - wydajność przetwarzania to nie wszystko,
  - Sporo zależy od profilu użycia.

Sector vs. Hadoop

- Wsparcie dla Java.
- Hadoop ma bogatsze API.
- Jakość dokumentacji.
- Czasami TCP działa lepiej niż UDT...