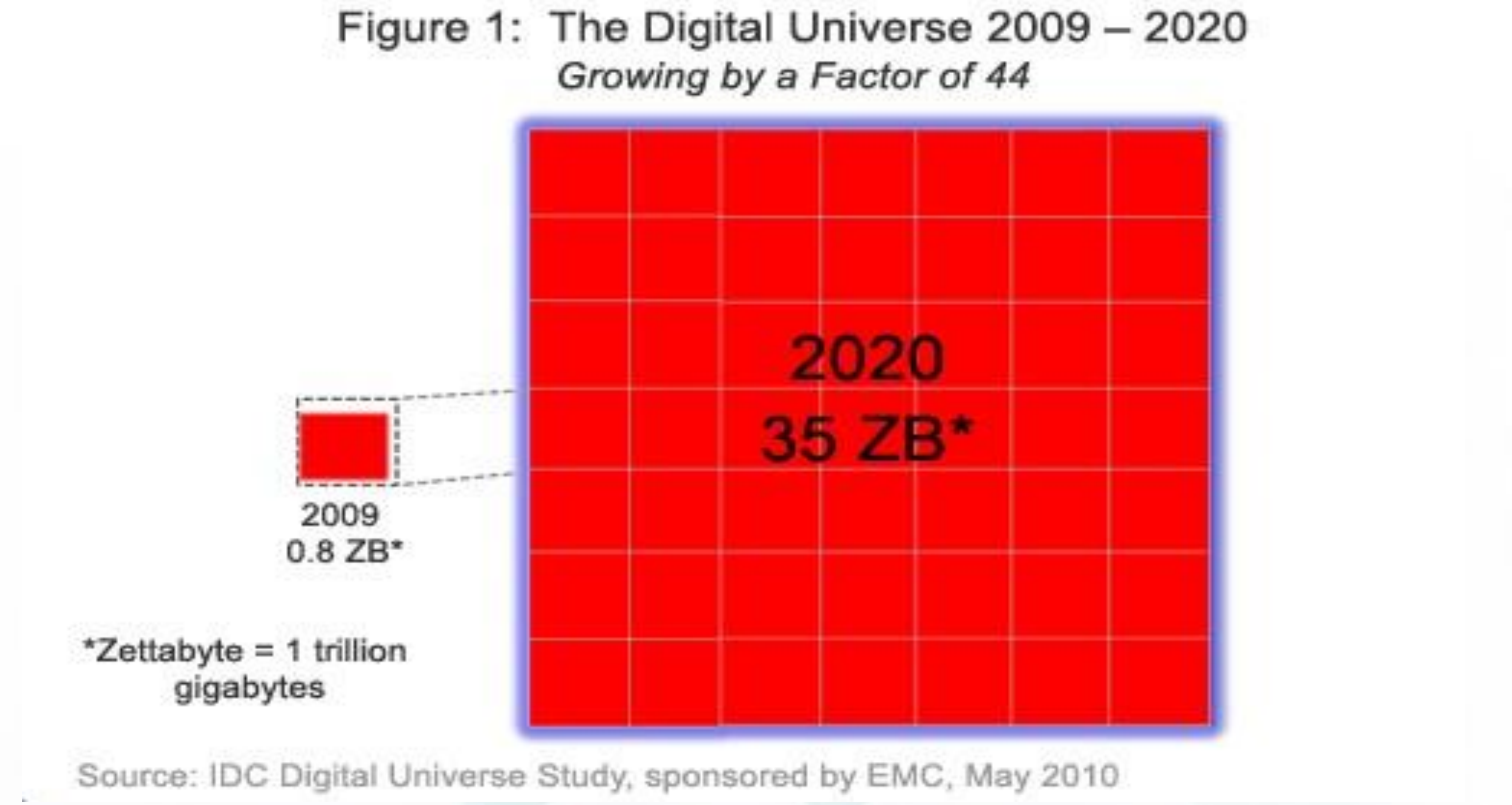
Meet Hadoop & MapReduce

Learning Objectives

- The Apache Hadoop Project
- Analyze a Weather Dataset
- Hadoop Map and Reduce
- Java MapReduce Programming
- Map and Reduce Tasks

The Digital Universe



- Digital Universe: the total amount of data stored in the world's computers
- Zettabyte: 10^{21} bytes >> Exabyte >> Petabyte >> Terabyte

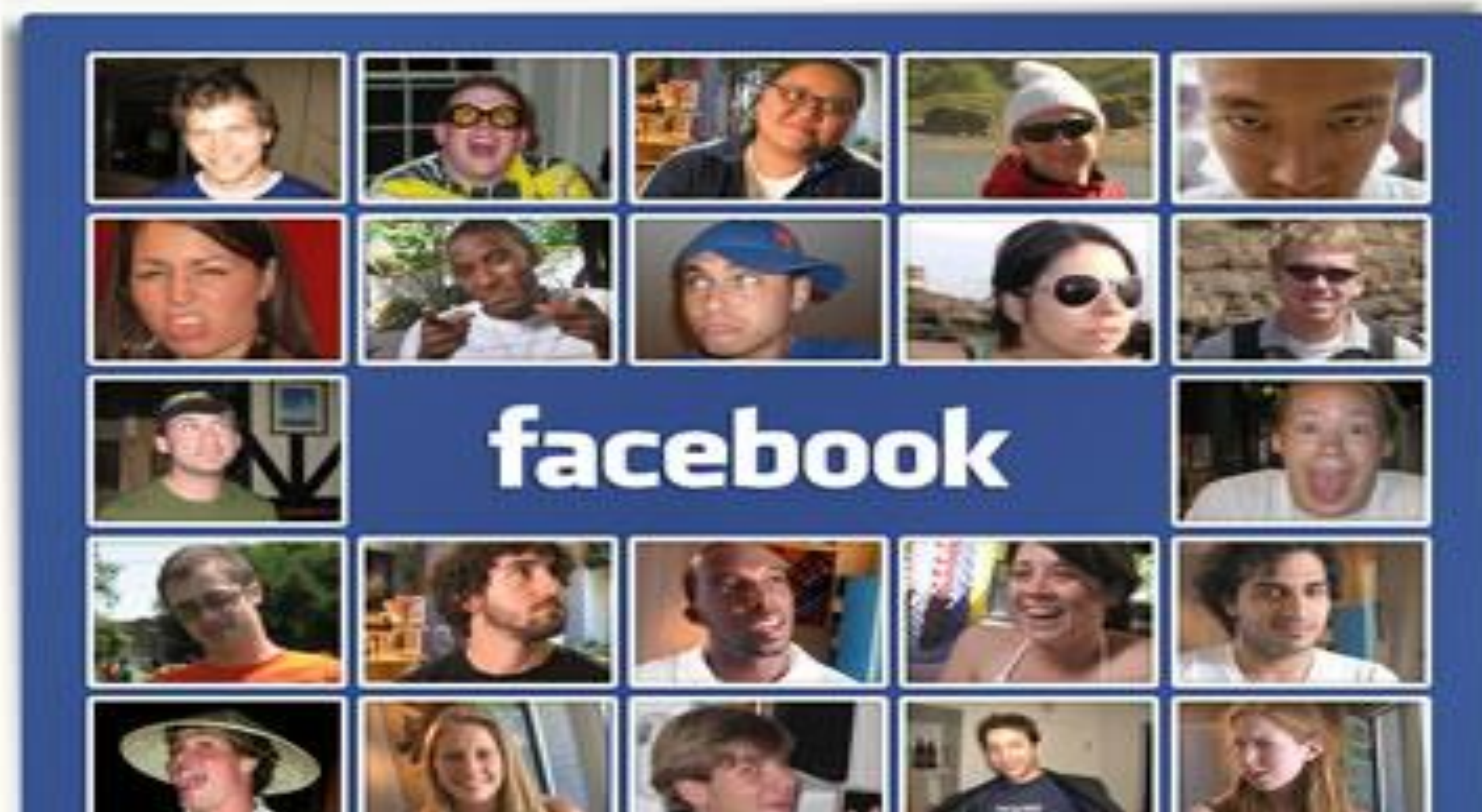
Hadoop – The Definitive Guide

Chapter 1 Meet Hadoop & Chapter 2 MapReduce

Flood of Data – 1

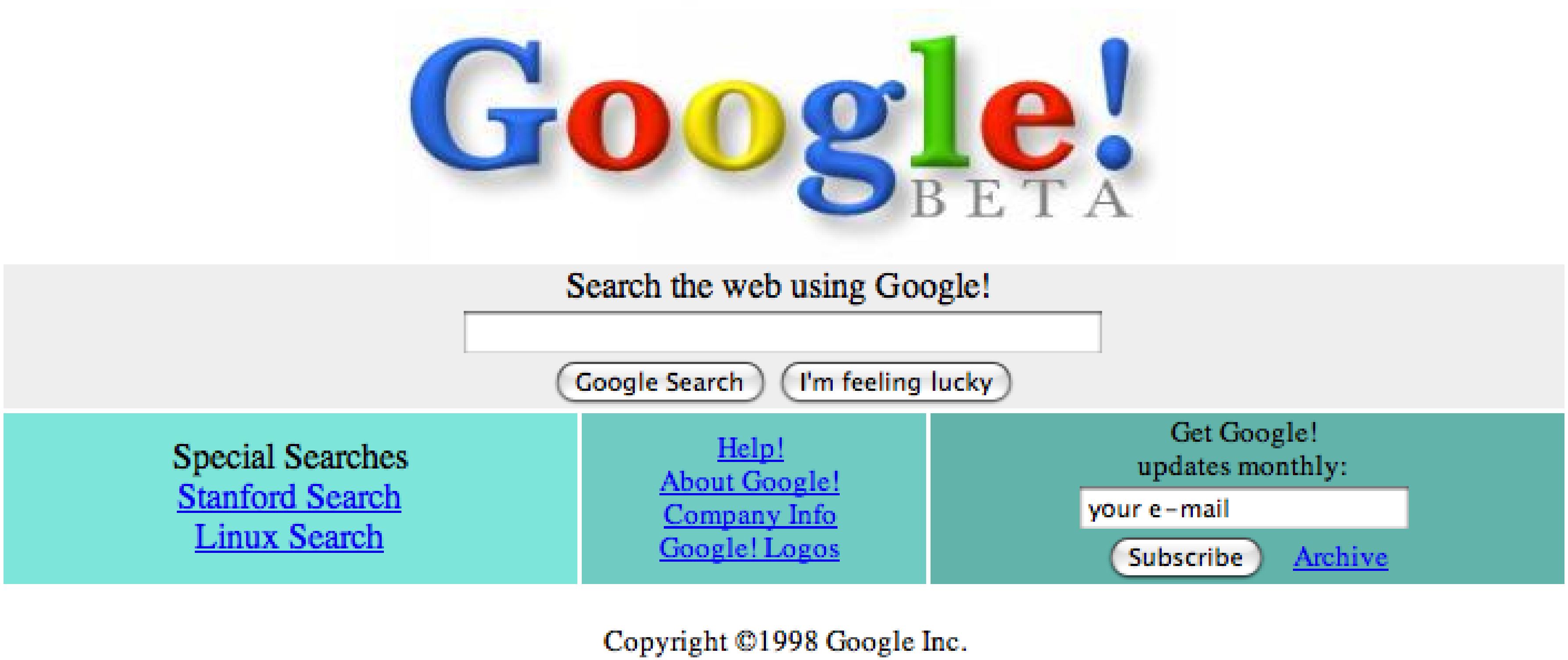
NYSE generates 1TB new trade data / day

Flood of Data – 2



Facebook hosts 10 billion photos (1 petabyte)

Flood of Data – 3



Google processed about 24 petabytes of data per day in 2009

Public Data

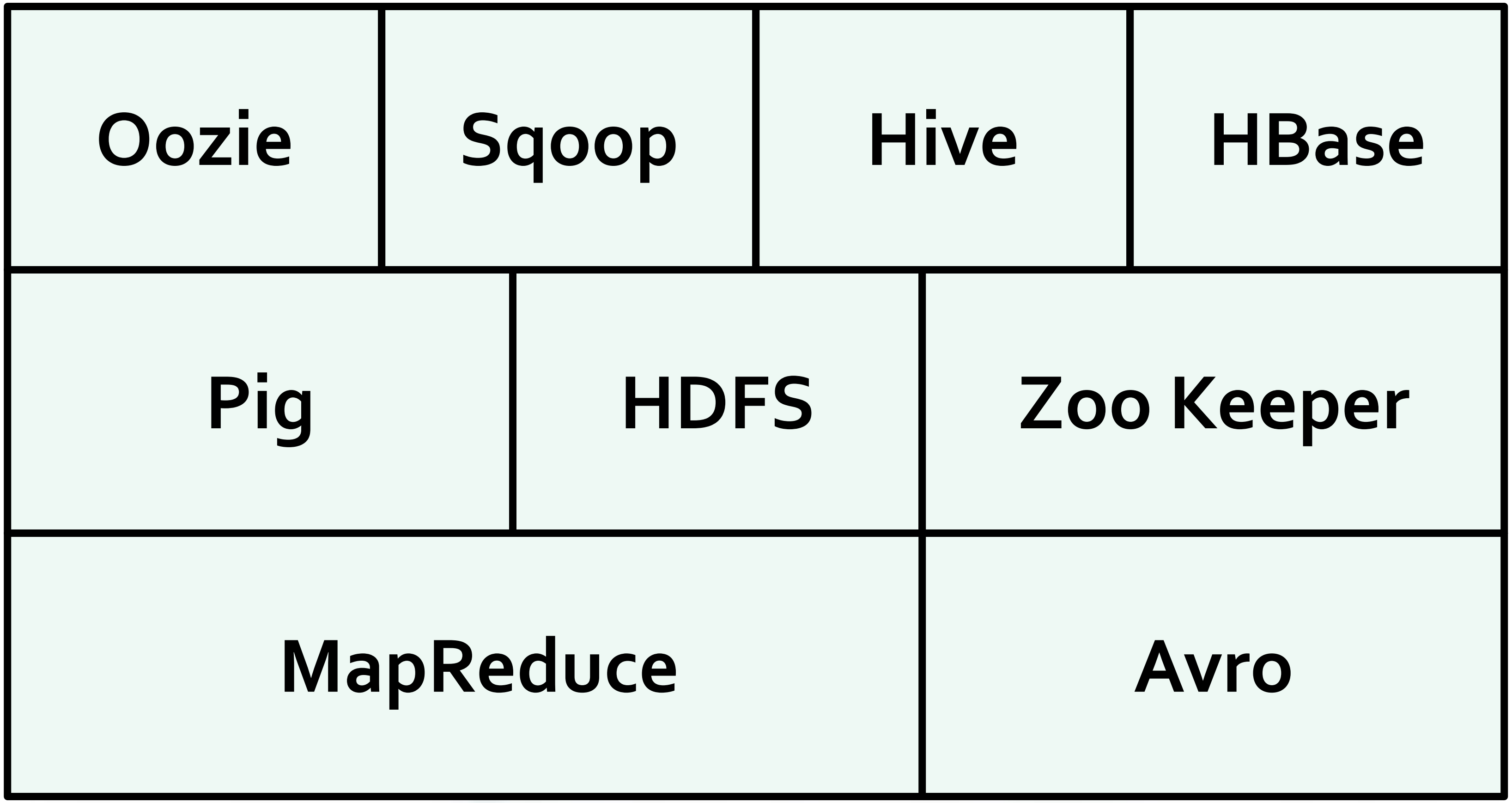
Available Public Data Sets on AWS

- Annotated Human Genome
- Public database of chemical structures
- Various census data and labor statistics

History of Hadoop

- Created by Doug Cutting
- Originated in Apache Nutch (2002)
  - Open source web search engine, a part of the Lucene project
- NDFS (Nutch Distributed File System, 2004)
- MapReduce (2005)
- Doug Cutting joins Yahoo! (Jan 2006)
- Official start of Apache Hadoop project (Feb 2006)
- Adoption of Hadoop on Yahoo! Grid team (Feb 2006)

The Apache Hadoop Project



MapReduce

- MapReduce is a programming model for parallel data processing.
- Hadoop can run MapReduce programs written in various languages: e.g. Java, Ruby, Python, C++.
- MapReduce comes into its own for large datasets.

Weather Dataset

- Data from NCDC (National Climatic Data Center)
  - A large volume of log data collected by weather sensors: e.g. temperature
- Data format
  - Line-oriented ASCII format
  - One line is one record
  - Each record has many elements
  - We focus on the temperature element
  - Data files are organized by date and weather station
  - There is a directory for each year from 1901 to 2001, each containing a gzipped file for each weather station with its readings for that year.

Analysis of Weather Dataset

- The link to the dataset: ftp://ftp.ncdc.noaa.gov/pub/data/noaa/
- What's the highest recorded global temperature for each year in the dataset?

Year Temperature

```
0067011990999991950051507004...9999999N9+00001+99999999999...
0043011990999991950051512004...9999999N9+00221+99999999999...
0043011990999991950051518004...9999999N9-00111+99999999999...
0043012650999991949032412004...0500001N9+01111+99999999999...
0043012650999991949032418004...0500001N9+00781+99999999999...
```

Contents of data files

```
% ls raw/1990 | head
010010-99999-1990.gz
010014-99999-1990.gz
010015-99999-1990.gz
010016-99999-1990.gz
010017-99999-1990.gz
010030-99999-1990.gz
010040-99999-1990.gz
010080-99999-1990.gz
010100-99999-1990.gz
010150-99999-1990.gz
```

List of data files

Analyzing the Data with Unix Tools

- Use *awk* for processing line-oriented data.
- Complete run for the century took 42 minutes on a single EC2 High-CPU Extra Large Instance.

```
#!/usr/bin/env bash
for year in all/*
do
  echo -ne `basename $year .gz`"\t"
  gunzip -c $year | \
    awk '{ temp = substr($0, 88, 5) + 0;
           q = substr($0, 93, 1);
           if (temp !=9999 && q ~ /[01459]/ && temp > max) max = temp }
         END { print max }'
done
```

```
% ./max_temperature.sh
1901    317
1902    244
1903    289
1904    256
1905    283
...
```

Parallelize This Work

- To speed up the processing, we need to run parts of the program in **parallel.**
- **Dividing** the work
  - Process different years in different process.
  - It is important to divide the work into even distribution , i.e., split the input into fixed-size chunks.
- **Combining** the results
  - Combination is more delicate when using fixed-size.
- But still we are limited by the processing capacity of a single machine.
  - Some datasets grow beyond the capacity of a single machine.

Issues for Parallel Processing

- To use multiple machines, we need to consider a variety of complex problems:
  - Coordination: Who runs the overall job?
  - Reliability: How do we deal with failed processes?
- **Hadoop** can take care of these issues.

Hadoop MapReduce

- To use MapReduce, we need to express out query as a MapReduce job.
- A MapReduce job includes two phases: the **Map** phase and the **Reduce** phase.
- Each phase has key-value pairs as input and output.
- Programmer specifies the Map and Reduce functions and the types of input and output for them.

MapReduce Design of NCDC Example - 1

Map phase

- Text input format of the dataset files
  - Key: offset of the line (unnecessary)
  - Value: each line of the files
- Pull out the year and the temperature
  - The map phase is simply data preparation phase
  - Drop bad records (filtering)

Input File

```
0067011990999991950051507004...9999999N9+00001+99999999999...
0043011990999991950051512004...9999999N9+00221+99999999999...
0043011990999991950051518004...9999999N9-00111+99999999999...
0043012650999991949032412004...0500001N9+01111+99999999999...
0043012650999991949032418004...0500001N9+00781+99999999999...
```

MapReduce Design of NCDC Example - 2

Input File

```
0067011990999991950051507004...9999999N9+00001+99999999999...
0043011990999991950051512004...9999999N9+00221+99999999999...
0043011990999991950051518004...9999999N9-00111+99999999999...
0043012650999991949032412004...0500001N9+01111+99999999999...
0043012650999991949032418004...0500001N9+00781+99999999999...
```

Input of Map Function (key, value)

```
(0, 0067011990999991950051507004...9999999N9+00001+99999999999...)
(106, 0043011990999991950051512004...9999999N9+00221+99999999999...)
(212, 0043011990999991950051518004...9999999N9-00111+99999999999...)
(318, 0043012650999991949032412004...0500001N9+01111+99999999999...)
(424, 0043012650999991949032418004...0500001N9+00781+99999999999...)
```

Map →

Output of Map Function (key, value)

```
(1950, 0)
(1950, 22)
(1950, -11)
(1949, 111)
(1949, 78)
```

MapReduce Design of NCDC Example - 3

- The output from the map function is processed by Reduce function.
  - Sorts and groups the key-value pairs by key

```
(1950, 0)
(1950, 22)
(1950, -11)
(1949, 111)
(1949, 78)
```

Sort and Group By →

```
(1949, [111, 78])
(1950, [0, 22, -11])
```

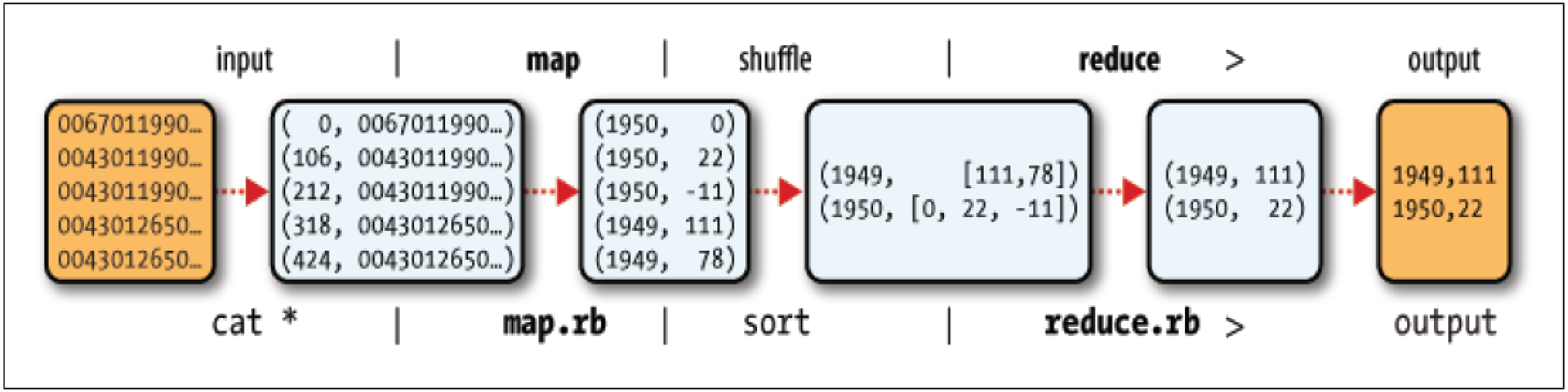
- Reduce function iterates through the list and pick up the maximum value

```
(1949, [111, 78])
(1950, [0, 22, -11])
```

Reduce →

```
(1949, 111)
(1950, 22)
```

MapReduce Design of NCDC Example - 4



Java Implementation: Map

- Map function implements the Mapper interface – generic type and four type parameters (input key, input value, output key, output value type).

```
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class MaxTemperatureMapper
  extends Mapper<LongWritable, Text, Text, IntWritable> {

  private static final int MISSING = 9999;

  @Override
  public void map(LongWritable key, Text value, Context context)
      throws IOException, InterruptedException {

    String line = value.toString();
    String year = line.substring(15, 19);
    int airTemperature;
    if (line.charAt(87) == '+') { // parseInt doesn't like leading plus signs
      airTemperature = Integer.parseInt(line.substring(88, 92));
    } else {
      airTemperature = Integer.parseInt(line.substring(87, 92));
    }
    String quality = line.substring(92, 93);
    if (airTemperature != MISSING && quality.matches("[01459]")) {
      context.write(new Text(year), new IntWritable(airTemperature));
    }
  }
}
```

Java Implementation: Reduce

- Reduce function implements the Reducer interface – generic type and four type parameters (input key, input value, output key, output value type)
- Input types of the reduce function must match the output types of the map function: Text and IntWritable.

```java
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class MaxTemperatureReducer
  extends Reducer<Text, IntWritable, Text, IntWritable> {

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

    int maxValue = Integer.MIN_VALUE;
    for (IntWritable value : values) {
      maxValue = Math.max(maxValue, value.get());
    }
    context.write(key, new IntWritable(maxValue));
  }
}
```

Java Implementation: Main - 1

```
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class MaxTemperature {

  public static void main(String[] args) throws Exception {
    if (args.length != 2) {
      System.err.println("Usage: MaxTemperature <input path> <output path>");
      System.exit(-1);
    }

    Job job = new Job();
    job.setJarByClass(MaxTemperature.class);
    job.setJobName("Max temperature");

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.setMapperClass(MaxTemperatureMapper.class);
    job.setReducerClass(MaxTemperatureReducer.class);

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

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

Java Implementation: Main - 2

- job.setJarByClass – code will be packaged into a JAR file; pass a class to the Job's setJarByClass() method; Hadoop will locate the JAR file containing this class and distribute around the cluster.
- addInputPath() & setOutputPath() – specify input and output paths; If the output directory exists before running the job, Hadoop will complain and not run the job.
- setMapperClass() & setReducerClass – specify map and reduce types.
- setOutputKeyClass() & setOutputValueClass() – control the output types for the map and the reduce functions, which are often the same.
- Wait ForCompletion() – submits the job and waits for it to finish.

A Test Run

Output Log

```
% export HADOOP_CLASSPATH=hadoop-examples.jar
% hadoop MaxTemperature input/ncdc/sample.txt output
12/02/04 11:50:41 WARN util.NativeCodeLoader: Unable to load native-hadoop library
for your platform... using builtin-java classes where applicable
12/02/04 11:50:41 WARN mapred.JobClient: Use GenericOptionsParser for parsing the
arguments. Applications should implement Tool for the same.
12/02/04 11:50:41 INFO input.FileInputFormat: Total input paths to process : 1
12/02/04 11:50:41 INFO mapred.JobClient: Running job: job_local_0001
12/02/04 11:50:41 INFO mapred.Task:  Using ResourceCalculatorPlugin : null
12/02/04 11:50:41 INFO mapred.MapTask: io.sort.mb = 100
12/02/04 11:50:42 INFO mapred.MapTask: data buffer = 79691776/99614720
12/02/04 11:50:42 INFO mapred.MapTask: record buffer = 262144/327680
12/02/04 11:50:42 INFO mapred.MapTask: Starting flush of map output
12/02/04 11:50:42 INFO mapred.MapTask: Finished spill 0
12/02/04 11:50:42 INFO mapred.Task: Task:attempt_local_0001_m_000000_0 is done. And i
s in the process of commiting
12/02/04 11:50:42 INFO mapred.JobClient:  map 0% reduce 0%
12/02/04 11:50:44 INFO mapred.LocalJobRunner:
12/02/04 11:50:44 INFO mapred.Task: Task 'attempt_local_0001_m_000000_0' done.
```

Final Output

```
% cat output/part-00000
1949    111
1950    22
```

Output Log (cont.)

```
12/02/04 11:50:44 INFO mapred.Task:  Using ResourceCalculatorPlugin : null
12/02/04 11:50:44 INFO mapred.LocalJobRunner:
12/02/04 11:50:44 INFO mapred.Merger: Merging 1 sorted segments
12/02/04 11:50:44 INFO mapred.Merger: Down to the last merge-pass, with 1 segments
left of total size: 57 bytes
12/02/04 11:50:44 INFO mapred.LocalJobRunner:
12/02/04 11:50:45 INFO mapred.Task: Task:attempt_local_0001_r_000000_0 is done. And
is in the process of commiting
12/02/04 11:50:45 INFO mapred.LocalJobRunner:
12/02/04 11:50:45 INFO mapred.Task: Task attempt_local_0001_r_000000_0 is allowed to
commit now
12/02/04 11:50:45 INFO output.FileOutputCommitter: Saved output of task 'attempt_local
_0001_r_000000_0' to output
12/02/04 11:50:45 INFO mapred.JobClient:  map 100% reduce 0%
12/02/04 11:50:47 INFO mapred.LocalJobRunner: reduce > reduce
12/02/04 11:50:47 INFO mapred.Task: Task 'attempt_local_0001_r_000000_0' done.
12/02/04 11:50:48 INFO mapred.JobClient:  map 100% reduce 100%
12/02/04 11:50:48 INFO mapred.JobClient: Job complete: job_local_0001
12/02/04 11:50:48 INFO mapred.JobClient: Counters: 17
12/02/04 11:50:48 INFO mapred.JobClient:   File Output Format Counters
12/02/04 11:50:48 INFO mapred.JobClient:     Bytes Written=29
12/02/04 11:50:48 INFO mapred.JobClient:   FileSystemCounters
12/02/04 11:50:48 INFO mapred.JobClient:     FILE_BYTES_READ=357503
12/02/04 11:50:48 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=425817
12/02/04 11:50:48 INFO mapred.JobClient:   File Input Format Counters
12/02/04 11:50:48 INFO mapred.JobClient:     Bytes Read=529
12/02/04 11:50:48 INFO mapred.JobClient:   Map-Reduce Framework
12/02/04 11:50:48 INFO mapred.JobClient:     Map output materialized bytes=61
12/02/04 11:50:48 INFO mapred.JobClient:     Map input records=5
12/02/04 11:50:48 INFO mapred.JobClient:     Reduce shuffle bytes=0
12/02/04 11:50:48 INFO mapred.JobClient:     Spilled Records=10
12/02/04 11:50:48 INFO mapred.JobClient:     Map output bytes=45
12/02/04 11:50:48 INFO mapred.JobClient:     Total committed heap usage (bytes)=36923
8016
12/02/04 11:50:48 INFO mapred.JobClient:     SPLIT_RAW_BYTES=129
12/02/04 11:50:48 INFO mapred.JobClient:     Combine input records=0
12/02/04 11:50:48 INFO mapred.JobClient:     Reduce input records=5
12/02/04 11:50:48 INFO mapred.JobClient:     Reduce input groups=2
12/02/04 11:50:48 INFO mapred.JobClient:     Combine output records=0
12/02/04 11:50:48 INFO mapred.JobClient:     Reduce output records=2
12/02/04 11:50:48 INFO mapred.JobClient:     Map output records=5
```

Jobtracker and Tasktracker

- To scale out, we need to store the data in a distributed filesystem, HDFS (Chap. 3).
- MapReduce job is divided into map tasks and reduce tasks.
- Two types of nodes: Jobtracker and Tasktracker
  - Jobtracker coordinates all the jobs on the system by scheduling tasks to run on tasktrackers.
  - If a task fails, the jobtracker can reschedule it on a different tasktracker.
  - Tasktracker runs tasks and send progress reports to the jobtracker.

Input Splits

- Divides input into fixed-size pieces – input splits
  - Hadoop creates one map task for each split.
  - Map task runs the user-defined map function for each record in the split.
- Size of splits
  - Small size is better for load-balancing: faster machine will be able to process more splits.
  - But if splits are too small, the overhead of managing the splits dominate the total execution time.
  - For most jobs, a good split size tends to be the size of a HDFS block, 64MB (default).

Data locality optimization

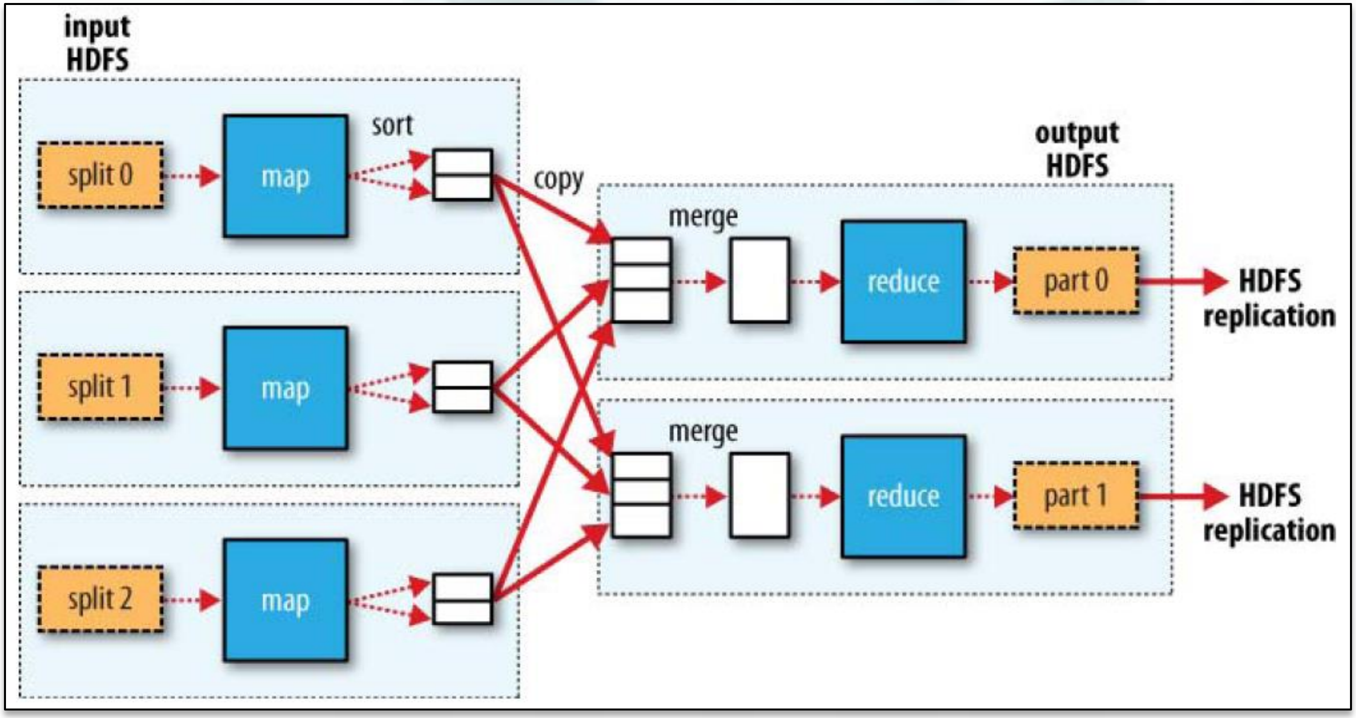
- Run the map task on a node where the input data resides in HDFS.
- This is the reason why the split size is the same as the block size
  - The largest size of the input that can be guaranteed to be stored on a single node.
  - If the split spanned two blocks, it would be unlikely that any HDFS node stored both blocks.

How Map Tasks Write Output

- Map tasks write their output to local disk (not to HDFS)
  - Map output is intermediate output.
  - Once the job is complete the map output can be thrown away.
  - So storing it in HDFS with replication, would be overkill.
  - If the node of map task fails, Hadoop will automatically rerun the map task on another node.

How Reduce Tasks Work – 1

- Reduce tasks don't have the advantage of data locality.
  - Input to a single reduce task is normally the output from all mappers;
  - Output of the reduce is stored in HDFS for reliability.



How Reduce Tasks Work – 2

- The number of reduce tasks is not governed by the size of the input, but is specified independently.
- When there are multiple reducers, the map tasks partition their output:
  - One partition for each reduce task.
  - The records for every key are all in a single partition.
  - Partitioning can be controlled by a user-defined partitioning function.

Combiner Function - 1

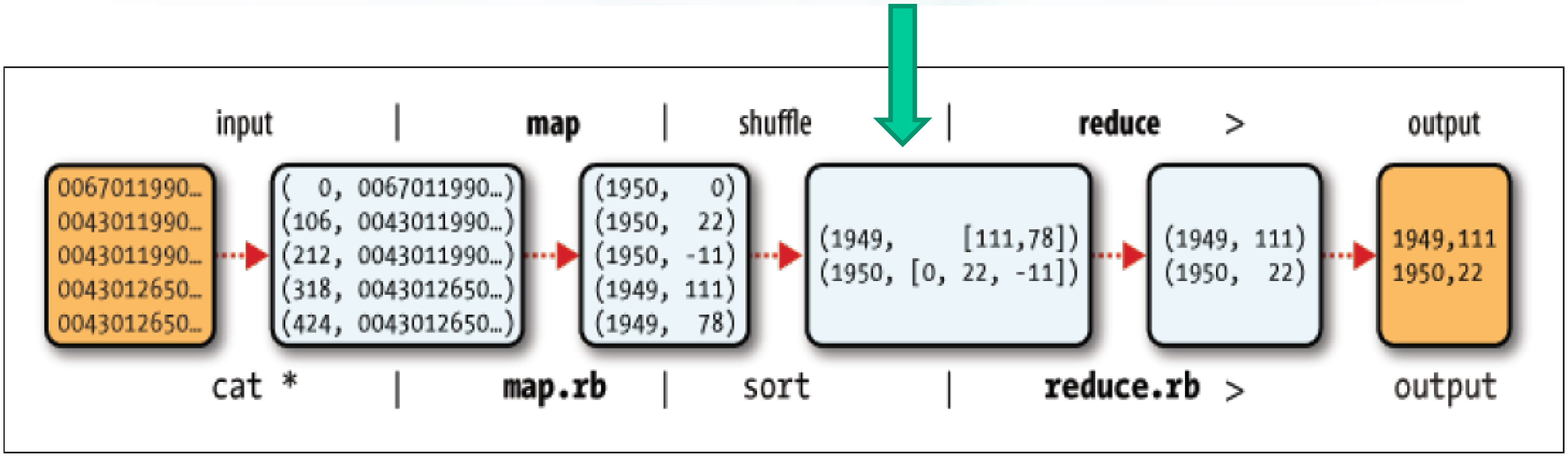
Combiner function

- Can be specified by users and is run on the map output.
- Forms the input to the reduce function.
- Minimizes the data transferred between map and reduce tasks.
- But Hadoop do not guarantee how many times it will call combiner function for a particular map output record.
  - It is just optimization.
  - The number of calling (even zero) does not affect the output of Reducers.

max(0, 20, 10, 25, 15) = max(max(0, 20, 10), max(25, 15)) = max(20, 25) = 25

Combiner Function - 2

Combiner
Function



Combiner Function - 3

- Combiner Function is defined using the Reducer class.
- Just set the combiner class on the Job.

```
public class MaxTemperatureWithCombiner {

  public static void main(String[] args) throws Exception {
    if (args.length != 2) {
      System.err.println("Usage: MaxTemperatureWithCombiner <input path> " +
          "<output path>");
      System.exit(-1);
    }

    Job job = new Job();
    job.setJarByClass(MaxTemperatureWithCombiner.class);
    job.setJobName("Max temperature");

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.setMapperClass(MaxTemperatureMapper.class);
    /*[*/job.setCombinerClass(MaxTemperatureReducer.class)/*]*/;
    job.setReducerClass(MaxTemperatureReducer.class);

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

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

Hadoop Streaming and Pipes

- Hadoop provides API for writing map and reduce functions in other languages (Ruby, Python,…).
- The other languages must be able to read standard input and write to standard output.
- Hadoop Streaming uses Unix standard streams as the interface between Hadoop and your program.
- Hadoop pipes is the name of the C++ interface to Hapdoop MapReduce.
  - Uses sockets as the channel over which the tasktracker interacts with the process running the C++ map or reduce funtion.