CSE 590: Special Topics Course
(Supercomputing)

Lecture 10
(MapReduce & Hadoop)

Rezaul A. Chowdhury
Department of Computer Science
SUNY Stony Brook
Spring 2016

MapReduce

MapReduce is

- a *programming model* for expressing distributed computations on massive datasets, and
- an *execution framework* for large-scale data processing on commodity clusters

Developed at Google in 2004 (Jeffrey Dean & Sanjay Ghemawat).

An open-source version called *Hadoop* was later developed at Yahoo.

Hadoop is now an Apache project.

Amazon Elastic MapReduce runs Hadoop on Amazon EC2.

MapReduce

MapReduce provides

- Simple API's, and
- Automatic
  - Parallelization
  - Data distribution
  - Load balancing
  - Fault tolerance

Big Ideas behind MapReduce

Scale Out Instead of Scaling Up: A large number of commodity low-end servers is preferred over a small number of high-end servers.

Be Ready to Tackle Failures: Failures are the norm at warehouse scale computing.

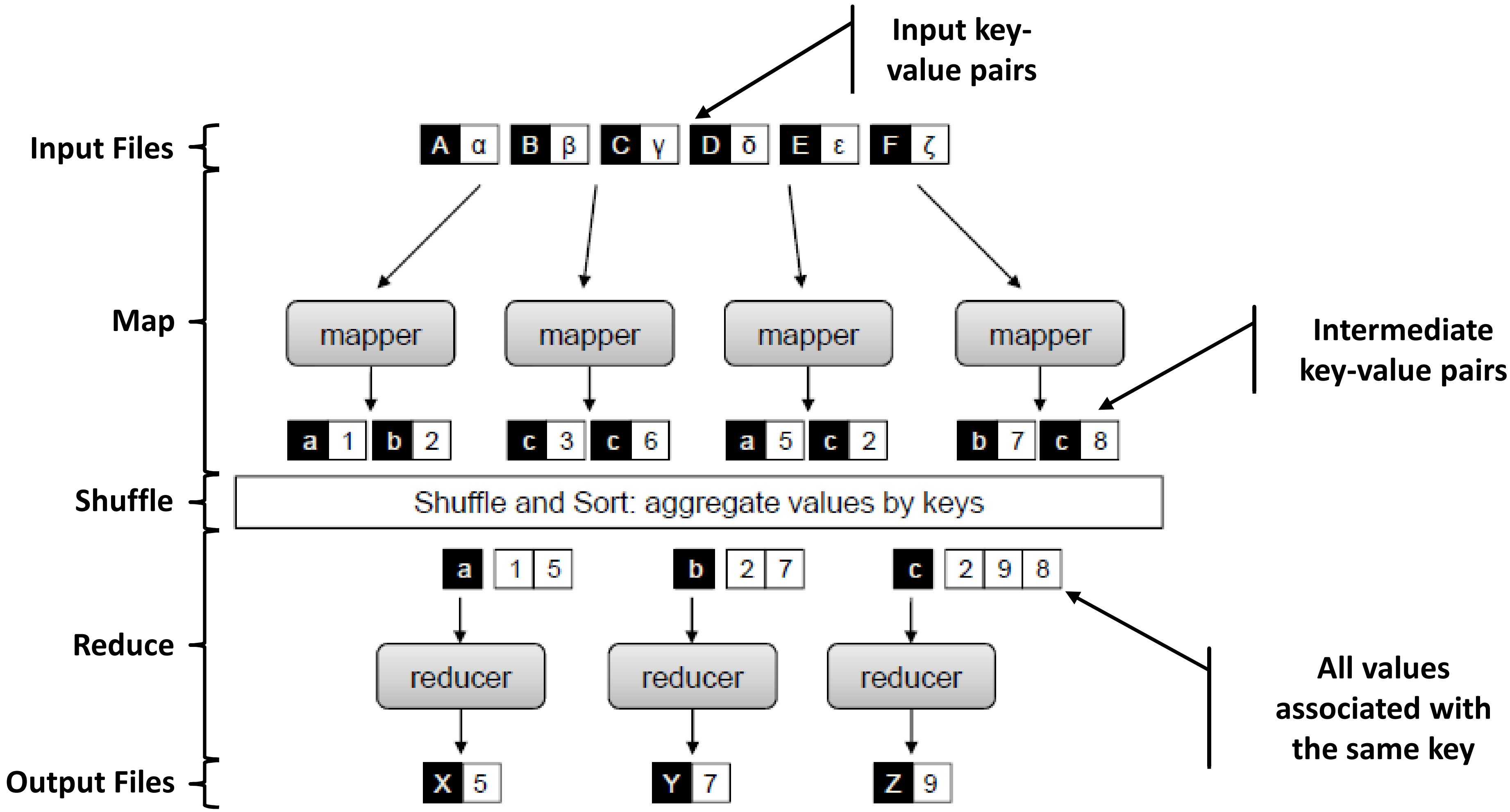
Move Code to the Data: Code transfer is much cheaper than transferring massive amounts of data.

Process Data Sequentially: Random accesses to data stored on disks are much costlier than sequential accesses.

Hide System-Level Details from Programmers: Provide a simple abstraction that is easy to reason about.

Seamless Scalability: A simple programming model to approach ideal scaling characteristics in many circumstances.

A Simplified View of MapReduce



Source: Lin & Dyer, “Data-Intensive Text Processing with MapReduce”

A Simple Word Count Example

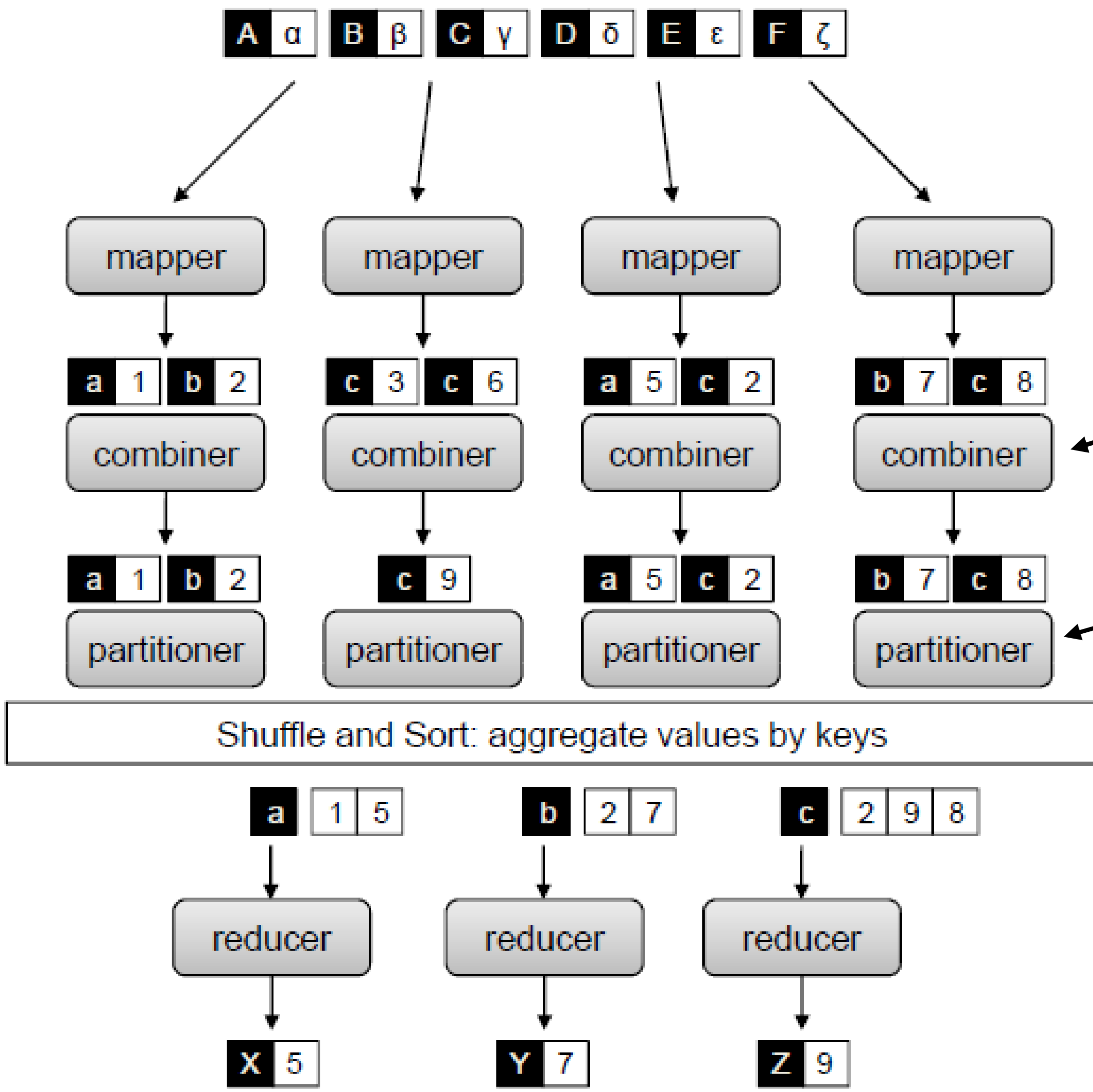
Count the number of occurrences of every word in a text collection.

class MAPPER
    method MAP(docid a, doc d)
        for all term $t \in$ doc d **do**
            EMIT(term t, count 1)

class REDUCER
    method REDUCE(term t, counts $[c_1, c_2, \ldots]$)
        $sum \leftarrow 0$
        for all count $c \in$ counts $[c_1, c_2, \ldots]$ **do**
            $sum \leftarrow sum + c$
        EMIT(term t, count sum)

Source: Lin & Dyer, “Data-Intensive Text Processing with MapReduce”

Combiner & Partitioner



A mini reducer in the map phase

Determines which key goes to which reducer

Source: Lin & Dyer, “Data-Intensive Text Processing with MapReduce”

Word Count with In-Mapper Combining

class MAPPER
    method MAP(docid a, doc d)
        $H \leftarrow$ new ASSOCIATIVEARRAY
        for all term $t \in$ doc d **do**
            $H\{t\} \leftarrow H\{t\} + 1$ ▷ Tally counts for entire document
        for all term $t \in H$ **do**
            EMIT(term t, count $H\{t\}$)

class REDUCER
    method REDUCE(term t, counts $[c_1, c_2, \ldots]$)
        $sum \leftarrow 0$
        for all count $c \in$ counts $[c_1, c_2, \ldots]$ **do**
            $sum \leftarrow sum + c$
        EMIT(term t, count sum)

Source: Lin & Dyer, “Data-Intensive Text Processing with MapReduce”

Word Count with Improved In-Mapper Combining

```
class Mapper
    method Initialize
        H ← new AssociativeArray
    method Map(docid a, doc d)
        for all term t ∈ doc d do
            H{t} ← H{t} + 1                    ▷ Tally counts across documents
    method Close
        for all term t ∈ H do
            Emit(term t, count H{t})
```

```
class Reducer
    method Reduce(term t, counts [c1, c2, . . .])
        sum ← 0
        for all count c ∈ counts [c1, c2, . . .] do
            sum ← sum + c
        Emit(term t, count sum)
```

Source: Lin & Dyer, “Data-Intensive Text Processing with MapReduce”

Compute Mean of Values Associated with Each Key

```
class Mapper
    method Map(string t, integer r)
        Emit(string t, integer r)

class Reducer
    method Reduce(string t, integers [r1, r2, ...])
        sum ← 0
        cnt ← 0
        for all integer r ∈ integers [r1, r2, ...] do
            sum ← sum + r
            cnt ← cnt + 1
        r_avg ← sum/cnt
        Emit(string t, integer r_avg)
```

Source: Lin & Dyer, “Data-Intensive Text Processing with MapReduce”

Mean of Values with a Separate Combiner

class Mapper
  method Map(string t, integer r)
    Emit(string t, integer r)

class Combiner
  method Combine(string t, integers $[r_1, r_2, \ldots]$)
    $sum \leftarrow 0$
    $cnt \leftarrow 0$
    for all integer $r \in$ integers $[r_1, r_2, \ldots]$ **do**
      $sum \leftarrow sum + r$
      $cnt \leftarrow cnt + 1$
    Emit(string t, pair (sum, cnt)) ▷ Separate sum and count

class Reducer
  method Reduce(string t, pairs $[(s_1, c_1), (s_2, c_2) \ldots]$)
    $sum \leftarrow 0$
    $cnt \leftarrow 0$
    for all pair $(s, c) \in$ pairs $[(s_1, c_1), (s_2, c_2) \ldots]$ **do**
      $sum \leftarrow sum + s$
      $cnt \leftarrow cnt + c$
    $r_{avg} \leftarrow sum/cnt$
    Emit(string t, integer r_{avg})

Source: Lin & Dyer, “Data-Intensive Text Processing with MapReduce”

Mean of Values with a Separate Combiner

class MAPPER
  method MAP(string t, integer r)
    EMIT(string t, pair $(r, 1)$)

class COMBINER
  method COMBINE(string t, pairs $[(s_1, c_1), (s_2, c_2) \ldots]$)
    $sum \leftarrow 0$
    $cnt \leftarrow 0$
    for all pair $(s, c) \in$ pairs $[(s_1, c_1), (s_2, c_2) \ldots]$ **do**
      $sum \leftarrow sum + s$
      $cnt \leftarrow cnt + c$
    EMIT(string t, pair (sum, cnt))

class REDUCER
  method REDUCE(string t, pairs $[(s_1, c_1), (s_2, c_2) \ldots]$)
    $sum \leftarrow 0$
    $cnt \leftarrow 0$
    for all pair $(s, c) \in$ pairs $[(s_1, c_1), (s_2, c_2) \ldots]$ **do**
      $sum \leftarrow sum + s$
      $cnt \leftarrow cnt + c$
    $r_{avg} \leftarrow sum/cnt$
    EMIT(string t, integer r_{avg})

Source: Lin & Dyer, “Data-Intensive Text Processing with MapReduce”

Mean of Values with an In-Mapper Combiner

class MAPPER
  method INITIALIZE
    $S \leftarrow$ new ASSOCIATIVEARRAY
    $C \leftarrow$ new ASSOCIATIVEARRAY
  method MAP(string t, integer r)
    $S\{t\} \leftarrow S\{t\} + r$
    $C\{t\} \leftarrow C\{t\} + 1$
  method CLOSE
    for all term $t \in S$ **do**
      EMIT(term t, pair $(S\{t\}, C\{t\})$)

class REDUCER
  method REDUCE(string t, pairs $[(s_1, c_1), (s_2, c_2) \ldots]$)
    $sum \leftarrow 0$
    $cnt \leftarrow 0$
    for all pair $(s, c) \in$ pairs $[(s_1, c_1), (s_2, c_2) \ldots]$ **do**
      $sum \leftarrow sum + s$
      $cnt \leftarrow cnt + c$
    $r_{avg} \leftarrow sum / cnt$
    EMIT(string t, integer r_{avg})

Source: Lin & Dyer, “Data-Intensive Text Processing with MapReduce”

Computing Word Co-occurrences

```
class Mapper
    method Map(docid a, doc d)
        for all term w ∈ doc d do
            for all term u ∈ Neighbors(w) do
                Emit(pair (w, u), count 1)

class Reducer
    method Reduce(pair p, counts [c1, c2, . . .])
        s ← 0
        for all count c ∈ counts [c1, c2, . . .] do
            s ← s + c
        Emit(pair p, count s)
```

Source: Lin & Dyer, “Data-Intensive Text Processing with MapReduce”

Word Co-occurrences (Stripes Approach)

```
class Mapper
    method Map(docid a, doc d)
        for all term w ∈ doc d do
            H ← new AssociativeArray
            for all term u ∈ Neighbors(w) do
                H{u} ← H{u} + 1
            Emit(Term w, Stripe H)
```

```
class Reducer
    method Reduce(term w, stripes [H1, H2, H3, ...])
        Hf ← new AssociativeArray
        for all stripe H ∈ stripes [H1, H2, H3, ...] do
            Sum(Hf, H)
        Emit(term w, stripe Hf)
```

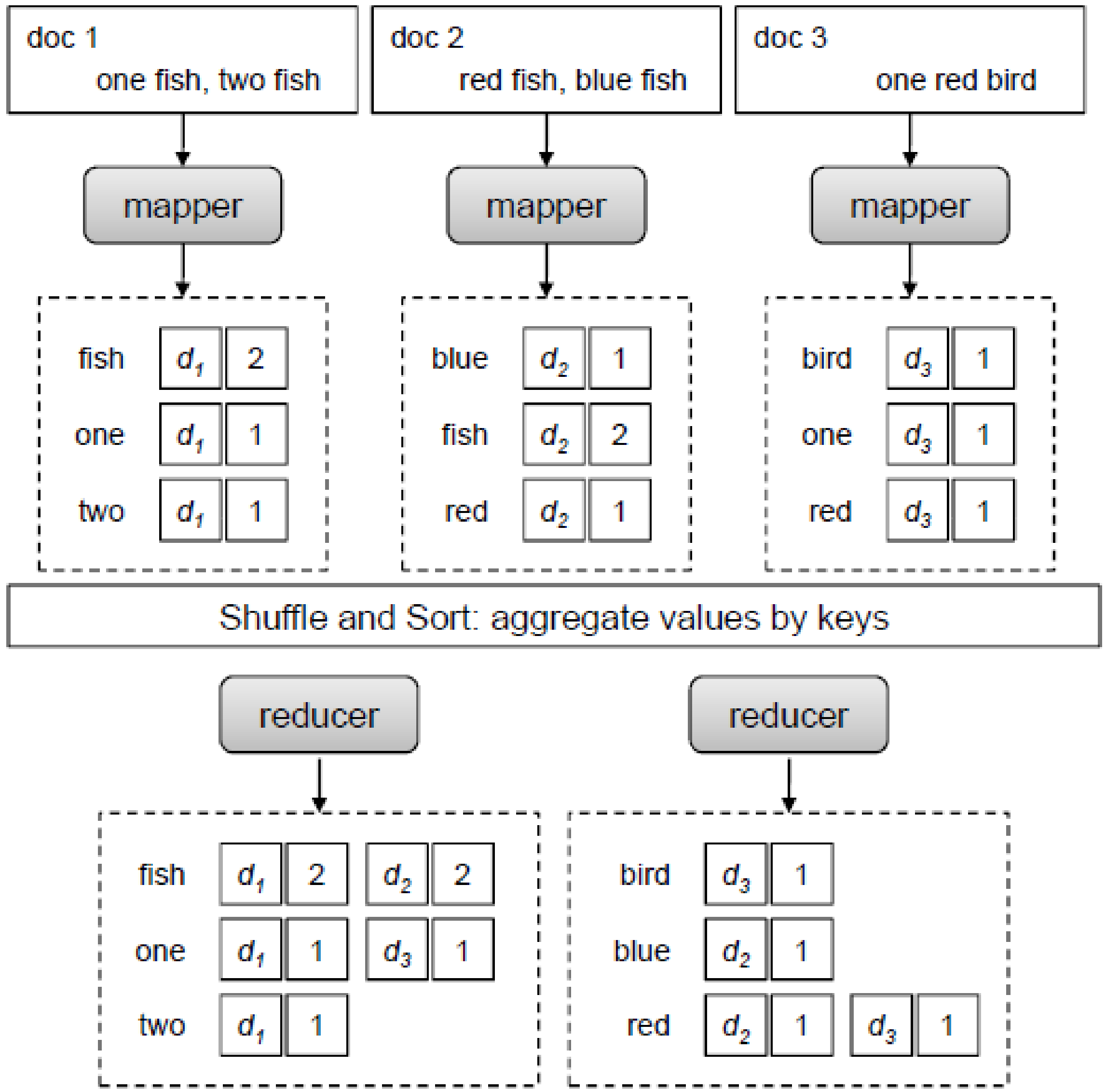
Source: Lin & Dyer, "Data-Intensive Text Processing with MapReduce"

Baseline Inverted Indexing for Text Retrieval

```
class Mapper
    procedure Map(docid n, doc d)
        H ← new AssociativeArray
        for all term t ∈ doc d do
            H{t} ← H{t} + 1
        for all term t ∈ H do
            Emit(term t, posting ⟨n, H{t}⟩)

class Reducer
    procedure Reduce(term t, postings [⟨n1, f1⟩, ⟨n2, f2⟩ ...])
        P ← new List
        for all posting ⟨a, f⟩ ∈ postings [⟨n1, f1⟩, ⟨n2, f2⟩ ...] do
            Append(P, ⟨a, f⟩)
        Sort(P)
        Emit(term t, postings P)
```

Source: Lin & Dyer, “Data-Intensive Text Processing with MapReduce”

Baseline Inverted Indexing for Text Retrieval



Source: Lin & Dyer, "Data-Intensive Text Processing with MapReduce"

Scalable Inverted Indexing for Text Retrieval

```
class Mapper
    method Map(docid n, doc d)
        H ← new AssociativeArray
        for all term t ∈ doc d do
            H{t} ← H{t} + 1
        for all term t ∈ H do
            Emit(tuple ⟨t, n⟩, tf H{t})

class Reducer
    method Initialize
        t_prev ← ∅
        P ← new PostingsList
    method Reduce(tuple ⟨t, n⟩, tf [f])
        if t ≠ t_prev ∧ t_prev ≠ ∅ then
            Emit(term t, postings P)
            P.Reset()
        P.Add(⟨n, f⟩)
        t_prev ← t
    method Close
        Emit(term t, postings P)
```

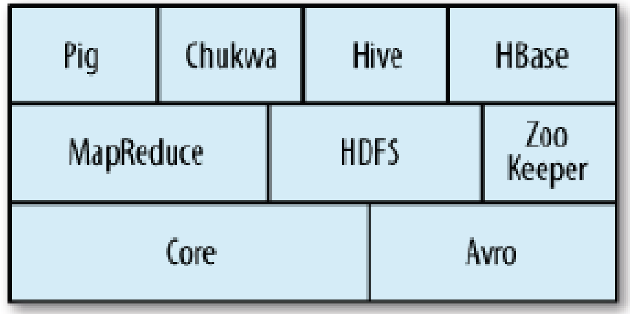
Source: Lin & Dyer, “Data-Intensive Text Processing with MapReduce”

Parallel Breadth-First Search

```
class Mapper
    method Map(nid n, node N)
        d ← N.Distance
        Emit(nid n, N)                                  ▷ Pass along graph structure
        for all nodeid m ∈ N.AdjacencyList do
            Emit(nid m, d + 1)                          ▷ Emit distances to reachable nodes

class Reducer
    method Reduce(nid m, [d1, d2, ...])
        dmin ← ∞
        M ← ∅
        for all d ∈ counts [d1, d2, ...] do
            if IsNode(d) then
                M ← d                                   ▷ Recover graph structure
            else if d < dmin then                       ▷ Look for shorter distance
                dmin ← d
        M.Distance ← dmin                               ▷ Update shortest distance
        Emit(nid m, node M)
```

Source: Lin & Dyer, "Data-Intensive Text Processing with MapReduce"

Hadoop Subprojects



Source: Tom White,
"Hadoop – The Definitive Guide"

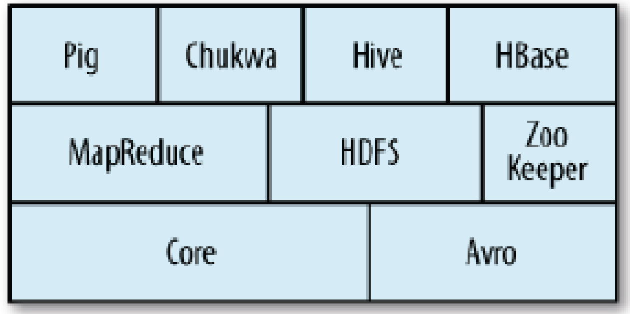
Core: A set of components and interfaces for distributed file systems and general I/O (serialization, Java RPC, persistent data structures).

Avro: A data serialization system for efficient, cross-language RPC, and persistent data storage.

MapReduce: A distributed data processing model and execution environment that runs on large clusters of commodity machines.

HDFS: A distributed filesystem that runs on large clusters of commodity machines.

Hadoop Subprojects



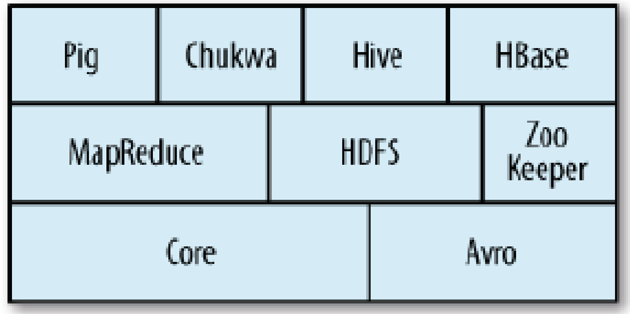
Source: Tom White,
"Hadoop – The Definitive Guide"

Pig: A data flow language and execution environment for exploring very large datasets. Pig runs on HDFS and MapReduce clusters.

HBase: A distributed, column-oriented database. HBase uses HDFS for its underlying storage, and supports both batch-style computations using MapReduce and point queries (random reads).

ZooKeeper: A distributed, highly available coordination service. ZooKeeper provides primitives such as distributed locks that can be used for building distributed applications.

Hadoop Subprojects



Source: Tom White,
"Hadoop – The Definitive Guide"

Hive: A distributed data warehouse. Hive manages data stored in HDFS and provides a query language based on SQL (and which is translated by the runtime engine to MapReduce jobs) for querying the data.

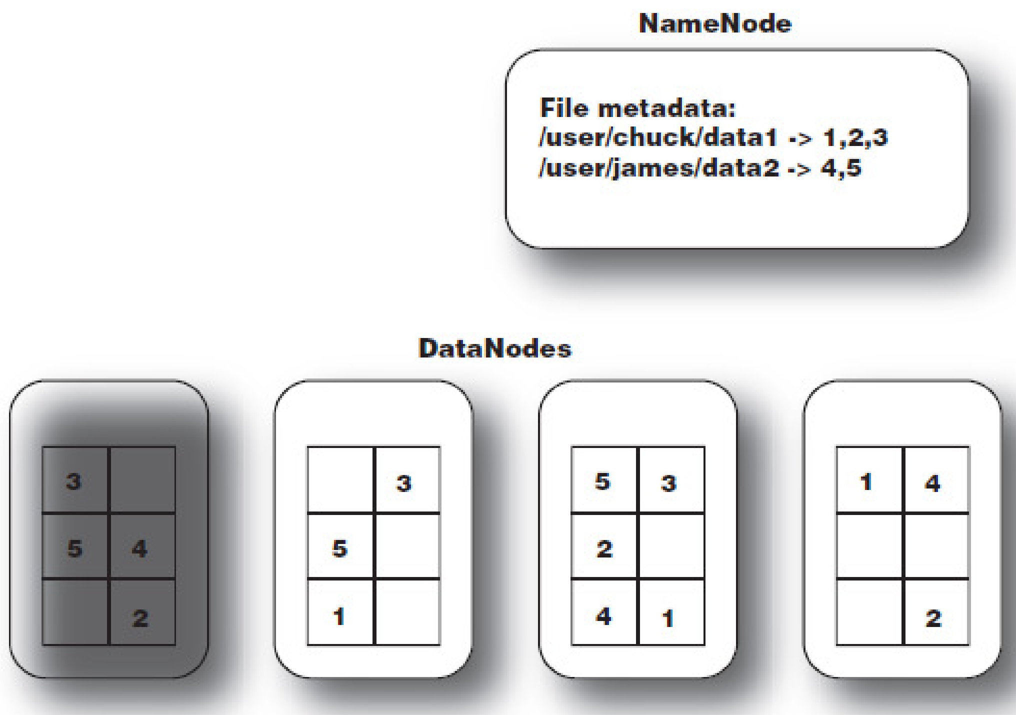
Chukwa: A distributed data collection and analysis system. Chukwa runs collectors that store data in HDFS, and it uses MapReduce to produce reports.

The Building Blocks of Hadoop

On a fully configured Hadoop cluster a set of daemons or resident programs run on the different servers in the network.

- NameNode
- DataNode
- Secondary NameNode
- JobTracker
- TaskTracker

The Building Blocks of Hadoop

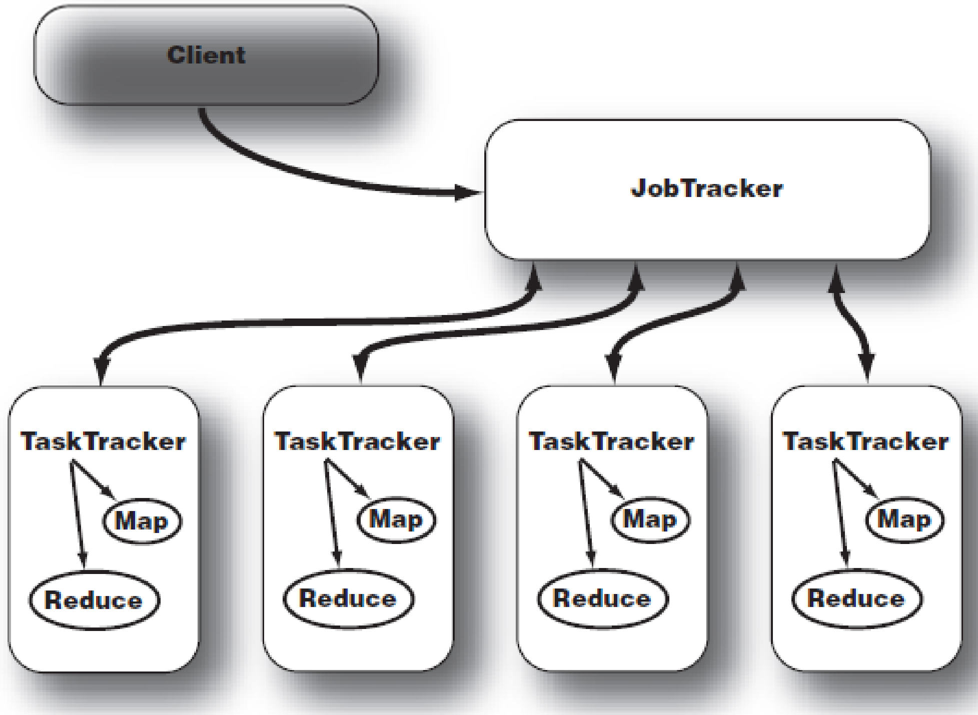


Source: Chuck Lam, "Hadoop in Action"

NameNode: The bookkeeper of HDFS: keeps track of how files are broken down into file blocks, which nodes store those blocks, and the overall health of the distributed filesystem.

DataNode: Each slave machine in the cluster hosts a DataNode daemon to perform the reading and writing of HDFS blocks to actual files on the local filesystem.

The Building Blocks of Hadoop

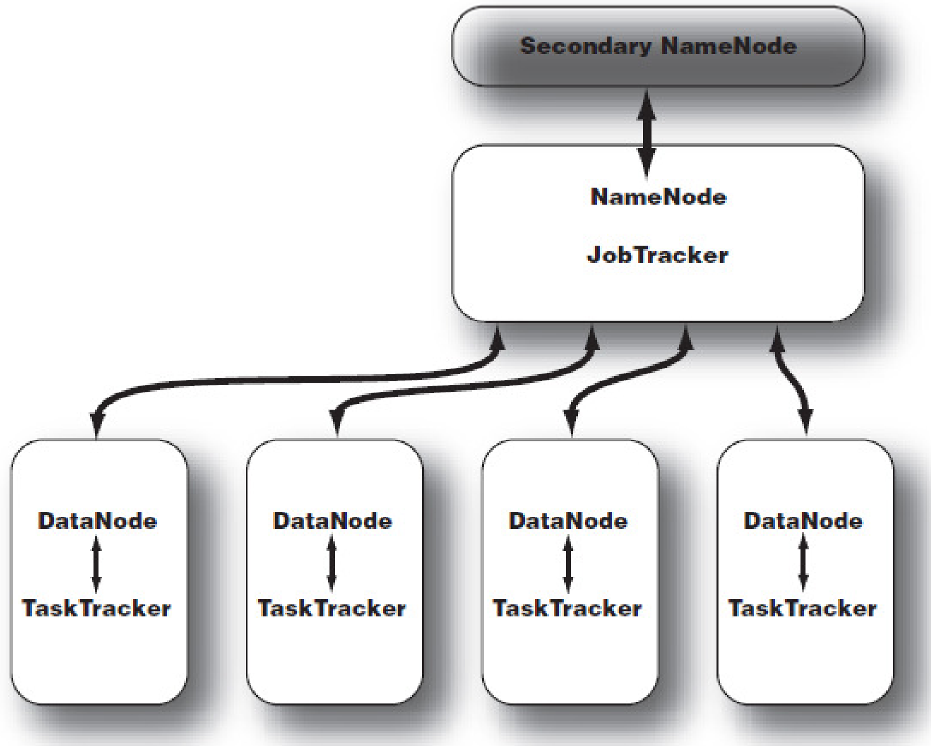


Source: Chuck Lam, "Hadoop in Action"

JobTracker: Determines the execution plan for a job by determining which files to process, assigns nodes to different tasks, and monitors all tasks as they're running. Should a task fail, the JobTracker will automatically relaunch the task, possibly on a different node.

TaskTracker: Manages the execution of individual (map or reduce) tasks on each slave node.

The Building Blocks of Hadoop



Source: Chuck Lam, "Hadoop in Action"

Secondary NameNode: It communicates with the NameNode to take periodic snapshots of the HDFS metadata. Does not keep track of any real-time changes to HDFS. Can be configured to work as the NameNode in the event of the failure of the original NameNode.

Hadoop Distributed File System (HDFS) Design

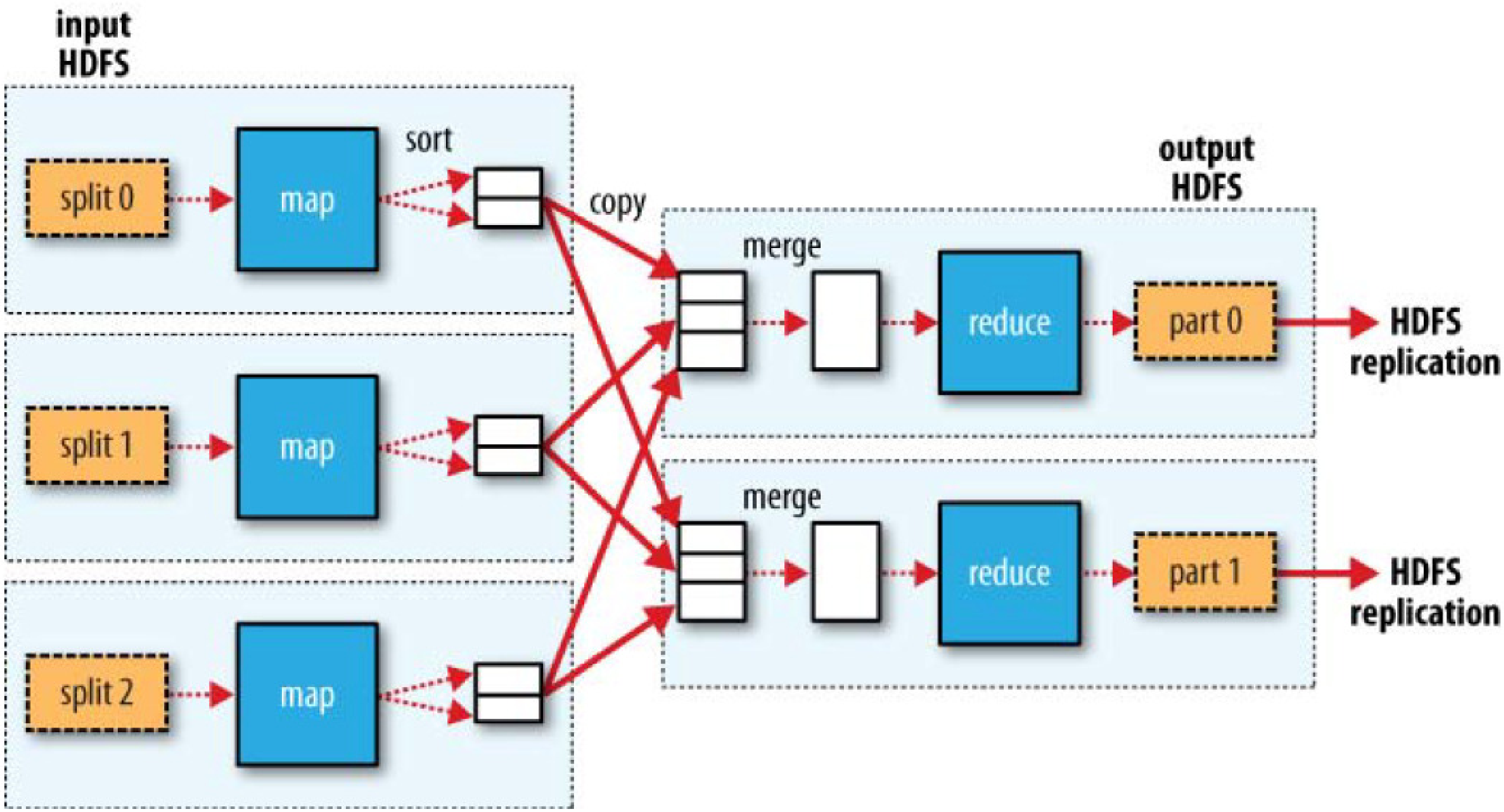
HDFS was designed for

- Very large files
- Streaming data access
- Commodity hardware

But not for

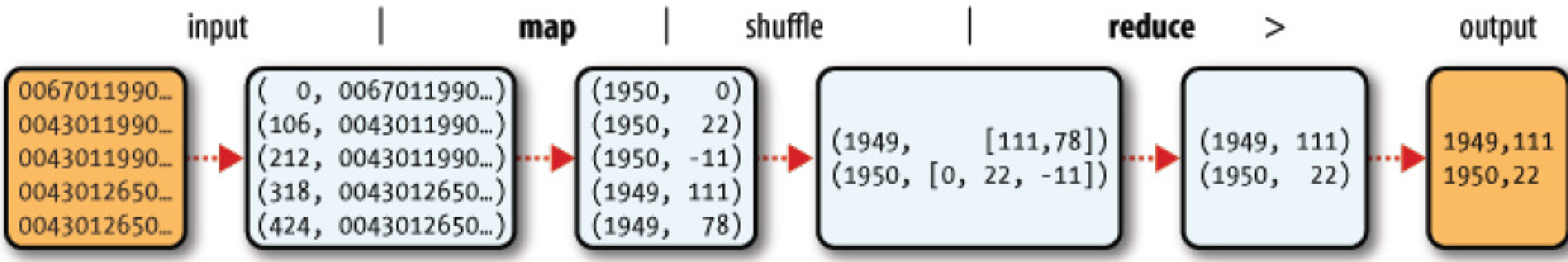
- Low latency access
- Lots of small files
- Multiple writes, arbitrary file modifications

Hadoop MapReduce



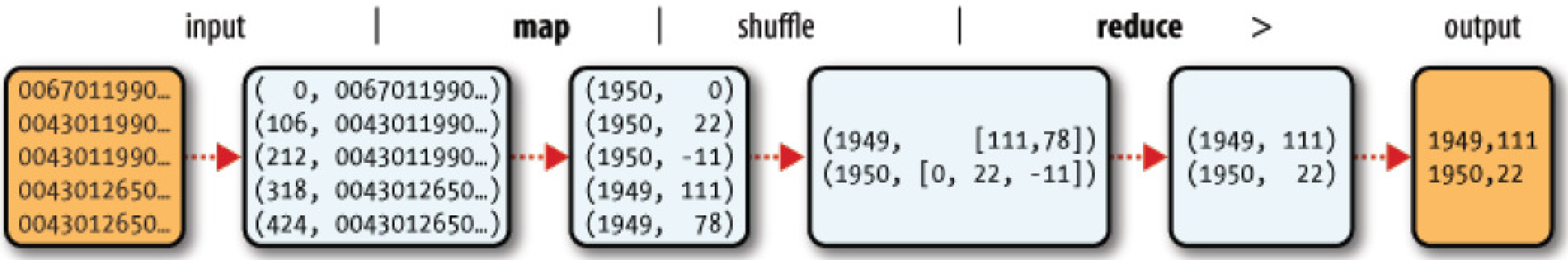
Source: Tom White,
"Hadoop – The Definitive Guide"

An Example: Mining Weather Data
Find Maximum Temperature Every Year



Source: Tom White,
"Hadoop – The Definitive Guide"

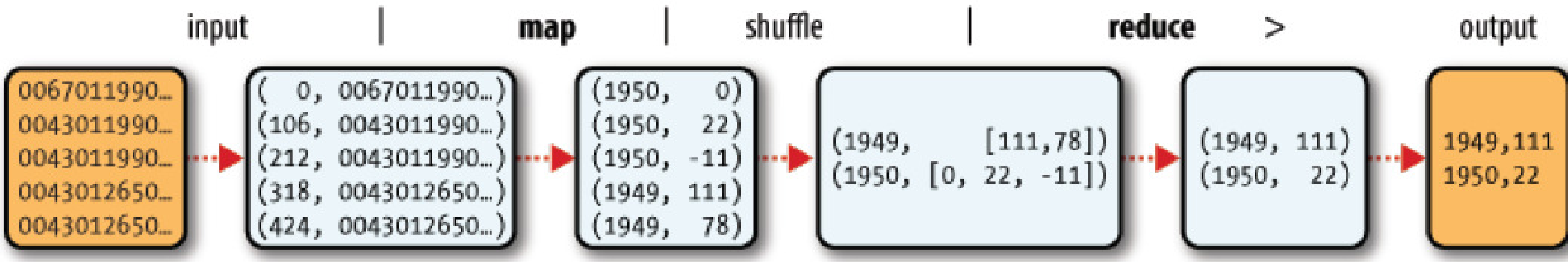
Maximum Temperature Every Year (Java)



Source: Tom White, “Hadoop – The Definitive Guide”

```
public class NewMaxTemperature {

  static class NewMaxTemperatureMapper
    extends Mapper<LongWritable, Text, Text, IntWritable> {

       … … …
   }

  static class NewMaxTemperatureReducer
    extends Reducer<Text, IntWritable, Text, IntWritable> {
       … … …
   }

  public static void main(String[] args) throws Exception {
       … … …
   }

}
```

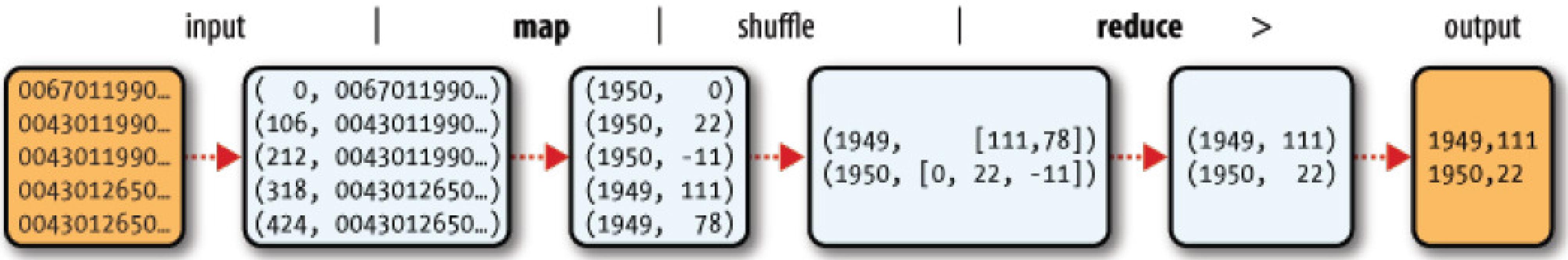
Maximum Temperature Every Year (Java)



Source: Tom White, “Hadoop – The Definitive Guide”

```
static class NewMaxTemperatureMapper
  extends Mapper<LongWritable, Text, Text, IntWritable> {

  private static final int MISSING = 9999;

  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));
    }
  }
}
```

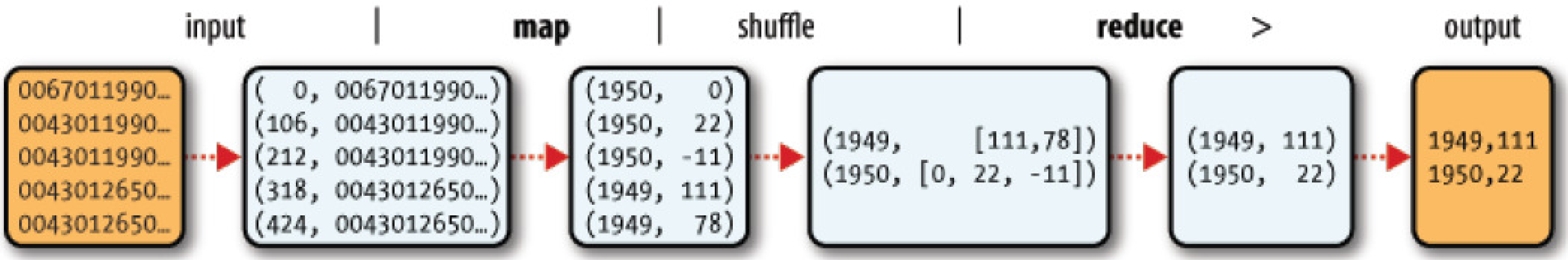
Maximum Temperature Every Year (Java)



Source: Tom White, “Hadoop – The Definitive Guide”

```
static class NewMaxTemperatureReducer
  extends Reducer<Text, IntWritable, Text, IntWritable> {

  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));
  }
}
```

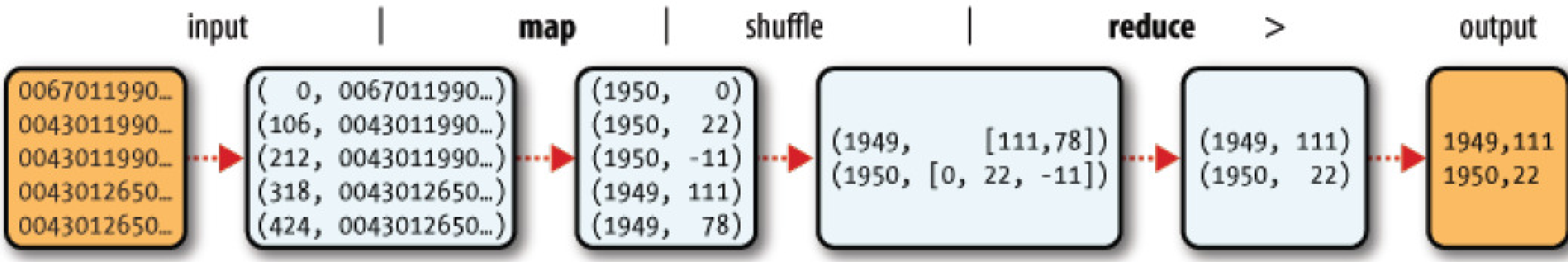
Maximum Temperature Every Year (Java)



Source: Tom White, “Hadoop – The Definitive Guide”

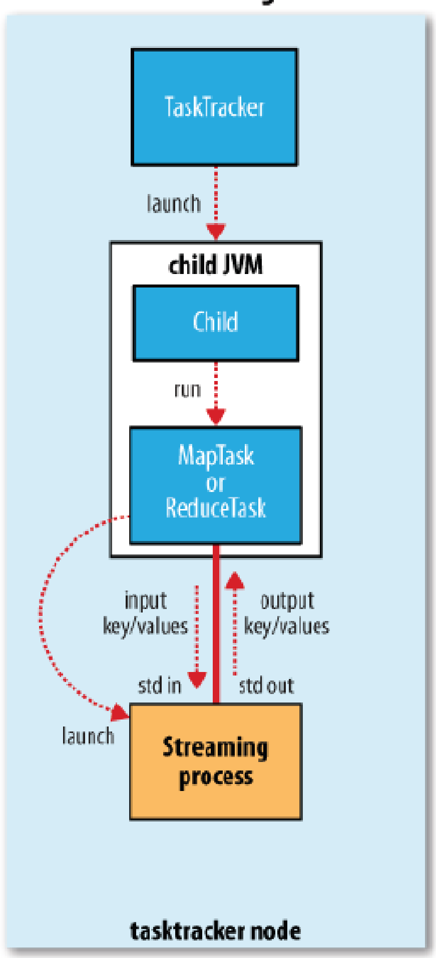
```java
public static void main(String[] args) throws Exception {
  if (args.length != 2) {
    System.err.println("Usage: NewMaxTemperature <input path> <output path>");
    System.exit(-1);
  }

  Job job = new Job();
  job.setJarByClass(NewMaxTemperature.class);

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

  job.setMapperClass(NewMaxTemperatureMapper.class);
  job.setReducerClass(NewMaxTemperatureReducer.class);

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

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

Maximum Temperature Every Year (Python)



Source: Tom White, “Hadoop – The Definitive Guide”

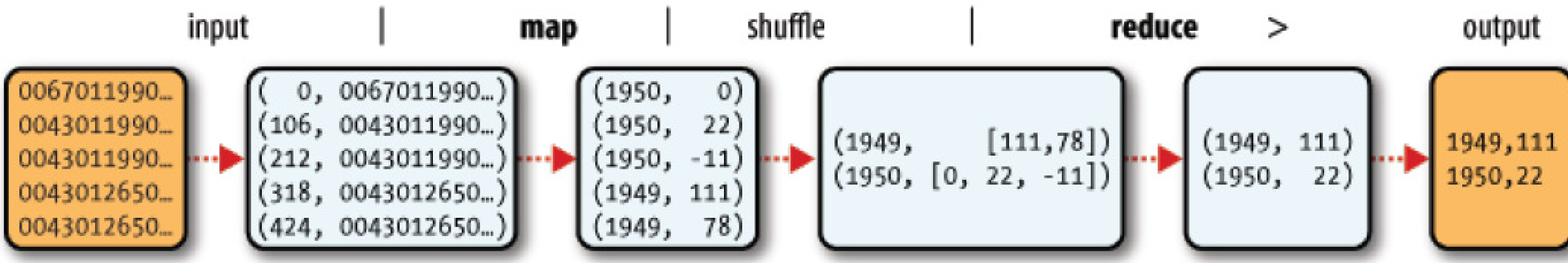
Streaming



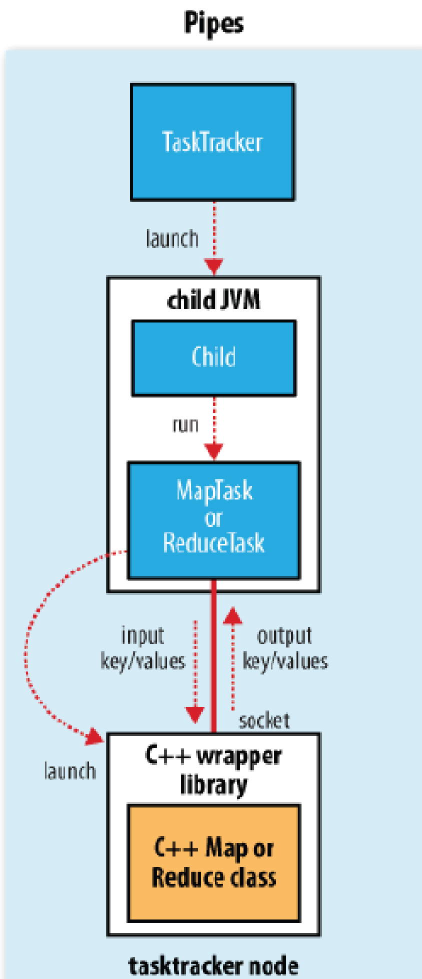

```python
#!/usr/bin/env python

import re
import sys

for line in sys.stdin:
  val = line.strip()
  (year, temp, q) = (val[15:19], val[87:92], val[92:93])
  if (temp != "+9999" and re.match("[01459]", q)):
    print "%s\t%s" % (year, temp)
```

```python
#!/usr/bin/env python

import sys

(last_key, max_val) = (None, 0)
for line in sys.stdin:
  (key, val) = line.strip().split("\t")
  if last_key and last_key != key:
    print "%s\t%s" % (last_key, max_val)
    (last_key, max_val) = (key, int(val))
  else:
    (last_key, max_val) = (key, max(max_val, int(val)))

if last_key:
  print "%s\t%s" % (last_key, max_val)
```

Maximum Temperature Every Year (C++)

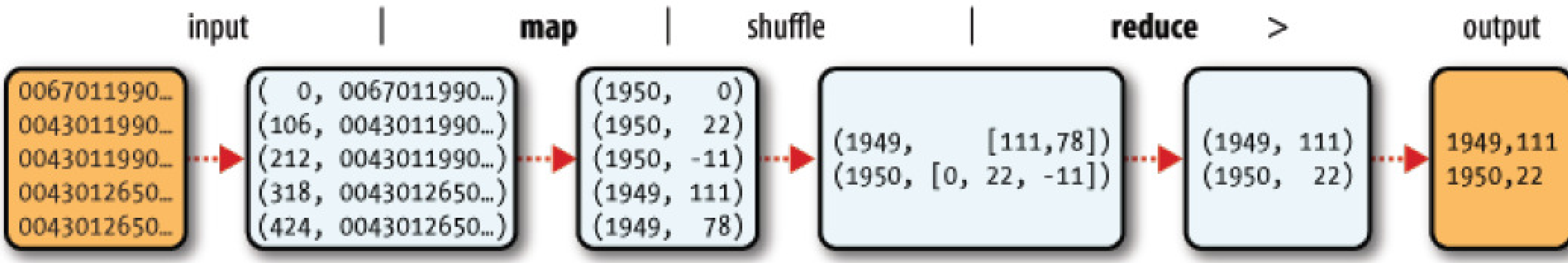


Source: Tom White, "Hadoop – The Definitive Guide"



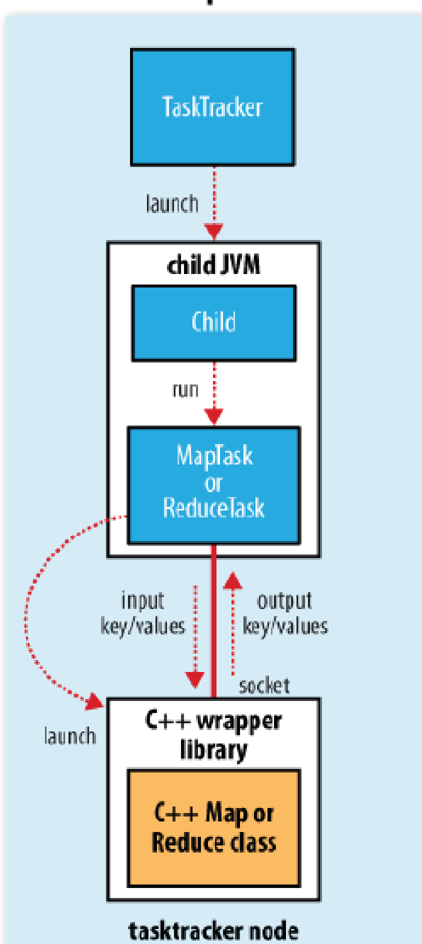

```cpp
#include "hadoop/Pipes.hh"
#include "hadoop/TemplateFactory.hh"
#include "hadoop/StringUtils.hh"

class MaxTemperatureMapper : public HadoopPipes::Mapper {
public:
  MaxTemperatureMapper(HadoopPipes::TaskContext& context) {
  }
  void map(HadoopPipes::MapContext& context) {
    std::string line = context.getInputValue();
    std::string year = line.substr(15, 4);
    std::string airTemperature = line.substr(87, 5);
    std::string q = line.substr(92, 1);
    if (airTemperature != "+9999" &&
        (q == "0" || q == "1" || q == "4" || q == "5" || q == "9")) {
      context.emit(year, airTemperature);
    }
  }
};
```

Maximum Temperature Every Year (C++)



Source: Tom White, “Hadoop – The Definitive Guide”

Pipes




```
class MapTemperatureReducer : public HadoopPipes::Reducer {
public:
  MapTemperatureReducer(HadoopPipes::TaskContext& context) {
  }
  void reduce(HadoopPipes::ReduceContext& context) {
    int maxValue = INT_MIN;
    while (context.nextValue()) {
      maxValue = std::max(maxValue, HadoopUtils::toInt(context.getInputValue()));
    }
    context.emit(context.getInputKey(), HadoopUtils::toString(maxValue));
  }
};


int main(int argc, char *argv[]) {
  return HadoopPipes::runTask(HadoopPipes::TemplateFactory<MaxTemperatureMapper,
                              MapTemperatureReducer>());
}
```