CS 378 – Big Data Programming

Lecture 7

File Formats

Review

- Assignment 3 – InvertedIndex
- Questions/issues?
- Coding comments (from Assignment 2)
  - The perils of reusing objects

File Formats

- In assignments 1 and 2, we used
  - **`TextInputFormat`**
  - **`TextOutputFormat`**
- Key value pairs:
  - Input: **`LongWritable`**/**`Text`**
  - Output: **`Text`**/**`DoubleArrayWritable`**
- The input file is just lines of text
  - How does the **`LongWritable`** get generated?

File Formats

- Input formats provide an instance that extends Hadoop class **`RecordReader`**
- **`RecordReader`** methods
  - **`initialize(InputSplit, TaskAttemptContext)`**
  - **`nextKeyValue()`**
  - **`getCurrentKey()`**
  - **`getCurrentValue()`**
  - **`getProgress()`**
  - **`close()`**

File Formats

- What does **`TextInputFormat`** do?
  - Via its **`RecordReader`** implementer

- Identifies the next line of input
  - Text through the next newline
- Creates the **`Text`** object with this content
- Calculates the position of this line in the input split
- Creates the **`LongWritable`** with this number
- Reports progress via **`getProgress()`**

File Formats

- Key value pairs:
  - Output: **`Text`**/**`DoubleArrayWritable`**
- The output file is just lines of text
  - How does this text get generated?
- Similar to input formats, output is controlled by instances that extend **`RecordWriter`**
- **`RecordWriter`** methods
  - **`write(key, value)`**
  - **`close()`**

File Formats

- What does **`TextOutputFormat`** do?
  - Via its **`RecordWriter`** implementer

- Calls **`toString()`** on the key, writes this string
- Writes a tab character
- Calls **`toString()`** on the value, writes this string

- How do we control the format of our results for WordStatistics?

File Formats

- Suppose we wanted to use the output of WordCount as input to another map-reduce job
  - Maybe we collected word counts for each day's emails
  - Now we want to sum up multiple days

- One approach: Use **`TextInputFormat`**
  - Map input is **`LongWritable`**, **`Text`**
  - We'd have to parse the value in the Text object to separate the key and value (separated by a tab)

File Formats

- Another approach: implement a custom file format

- What do we need to do?
- In our custom file format class …
  - Define a **`RecordReader`** interface implementer to:
  - Grab one line of input from the input split
  - Find the key/value separator
  - Return the key and the value as **`Text`** objects

- Seems like a convenient class to have around

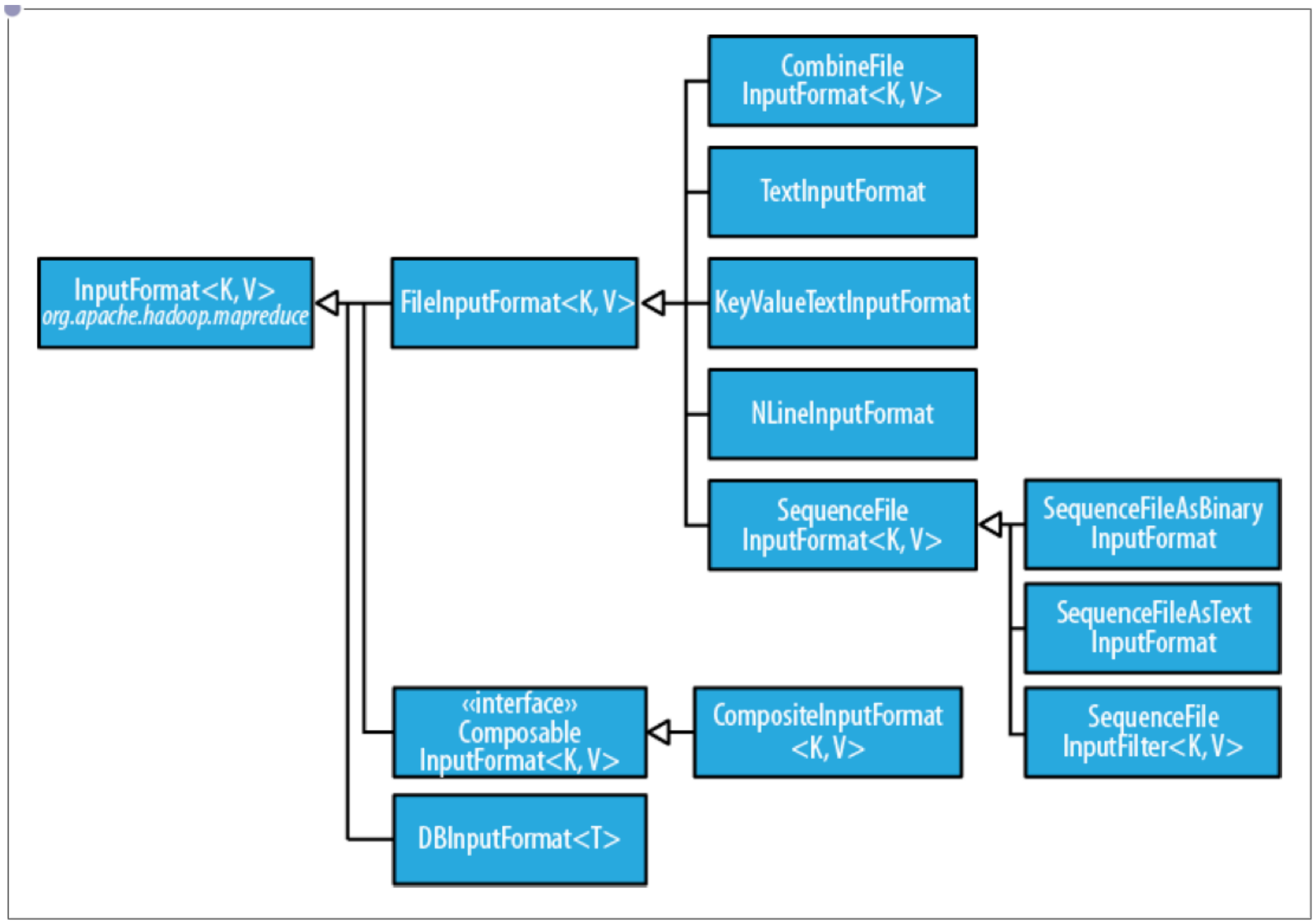
File Formats

- Hadoop provides exactly this class for us:
- **`KeyValueTextInputFormat`**
  - You can set the separator character (by default, tab)

- Other file formats and readers provided by Hadoop
  - Reading from a database
  - Each mapper receives exactly N lines
  - XML stream processing
  - Sequence files (binary)

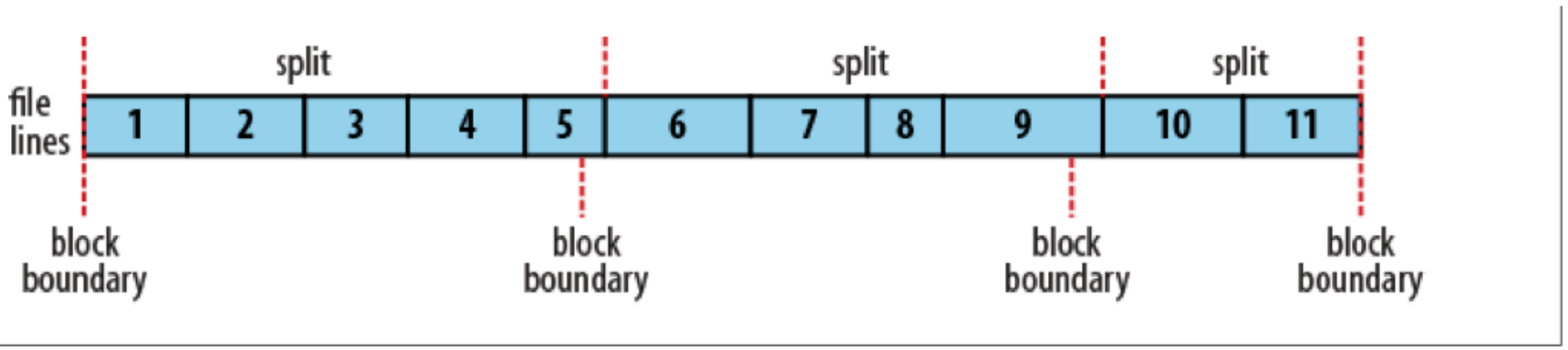
File Formats

Figure 7-2, Hadoop: The Definitive Guide 3rd Edition


InputFormat<K, V>
org.apache.hadoop.mapreduce
FileInputFormat<K, V>
CombineFile InputFormat<K, V>
TextInputFormat
KeyValueTextInputFormat
NLineInputFormat
SequenceFile InputFormat<K, V>
SequenceFileAsBinary InputFormat
SequenceFileAsText InputFormat
SequenceFile InputFilter<K, V>
«interface» Composable InputFormat<K, V>
CompositeInputFormat <K, V>
DBInputFormat<T>

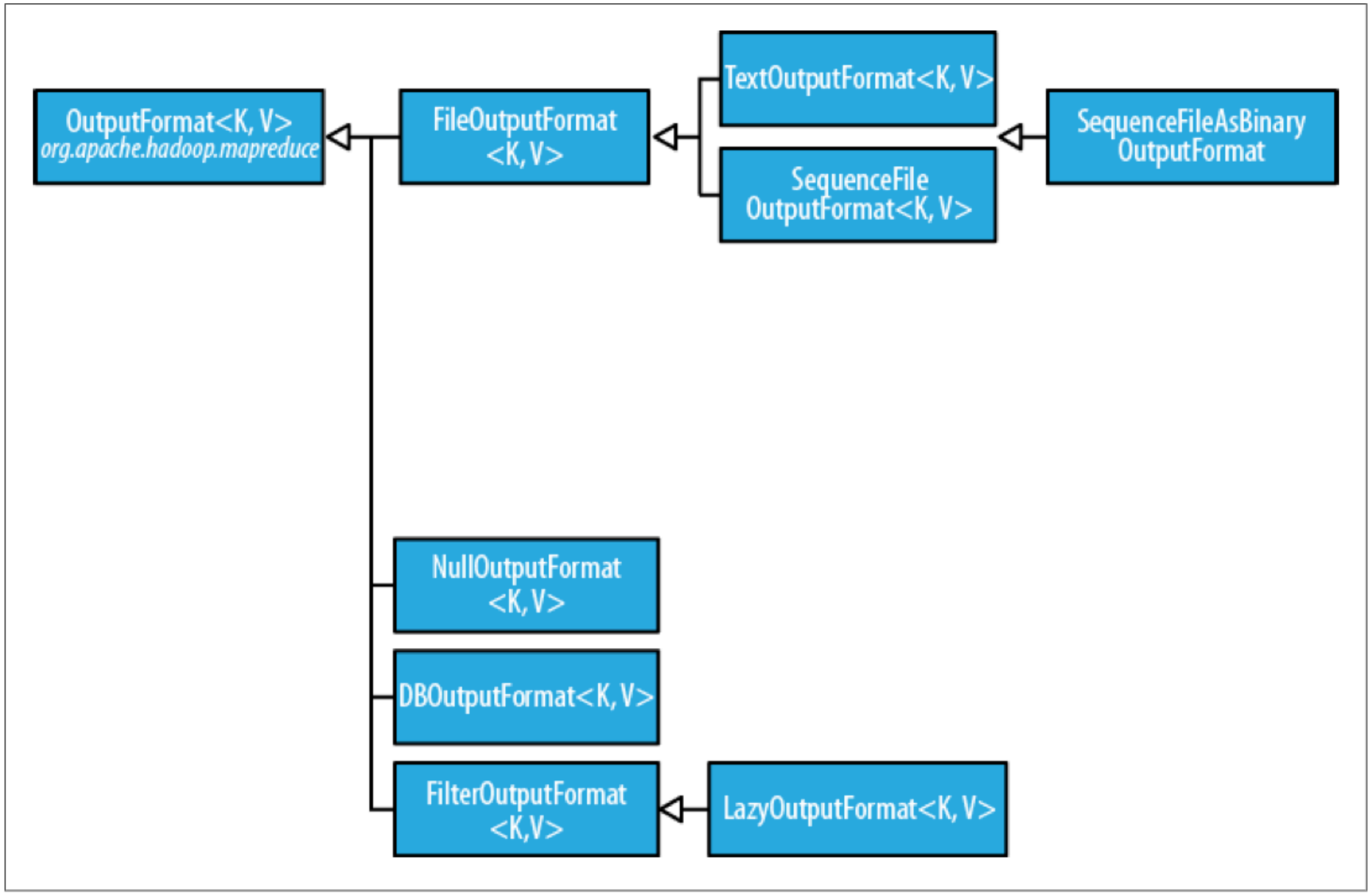
File Formats

Figure 7-3, Hadoop: The Definitive Guide 3rd Edition



File Formats

Figure 7-4, Hadoop: The Definitive Guide 3rd Edition


OutputFormat<K, V>
org.apache.hadoop.mapreduce
FileOutputFormat <K, V>
TextOutputFormat<K, V>
SequenceFile OutputFormat<K, V>
SequenceFileAsBinary OutputFormat
NullOutputFormat <K, V>
DBOutputFormat<K, V>
FilterOutputFormat <K,V>
LazyOutputFormat<K, V>