Introduction to Map/Reduce

Christoph

bases on **Yahoo! Hadoop Tutorial** (Module 4)
http://public.yahoo.com/gogate/hadoop-tutorial/html/module4.html

Agenda

- What is Map/Reduce?
- The Building Blocks: mapping and reducing
- An example: Word Count
- Hadoop specifics
- Some Tips

What is Map/Reduce?

- Programming model
- designed for processing large volumes of data in parallel
- divides the work into a set of independent tasks
- influenced by *functional programming* constructs
  - programming paradigm that treats computation as the evaluation of mathematical functions and avoids state and mutable data
  - no changes of states
  - output value of a function depends only on the arguments that are input to the function

What is Map/Reduce?

- Designed to process **large volumes of data** in a parallel fashion in a distributed system
- Model would not scale if components were allowed to share data arbitrarily
  - communication overhead
  - reliability
- Instead, all data is immutable
- Changes and communication can be done by generating output

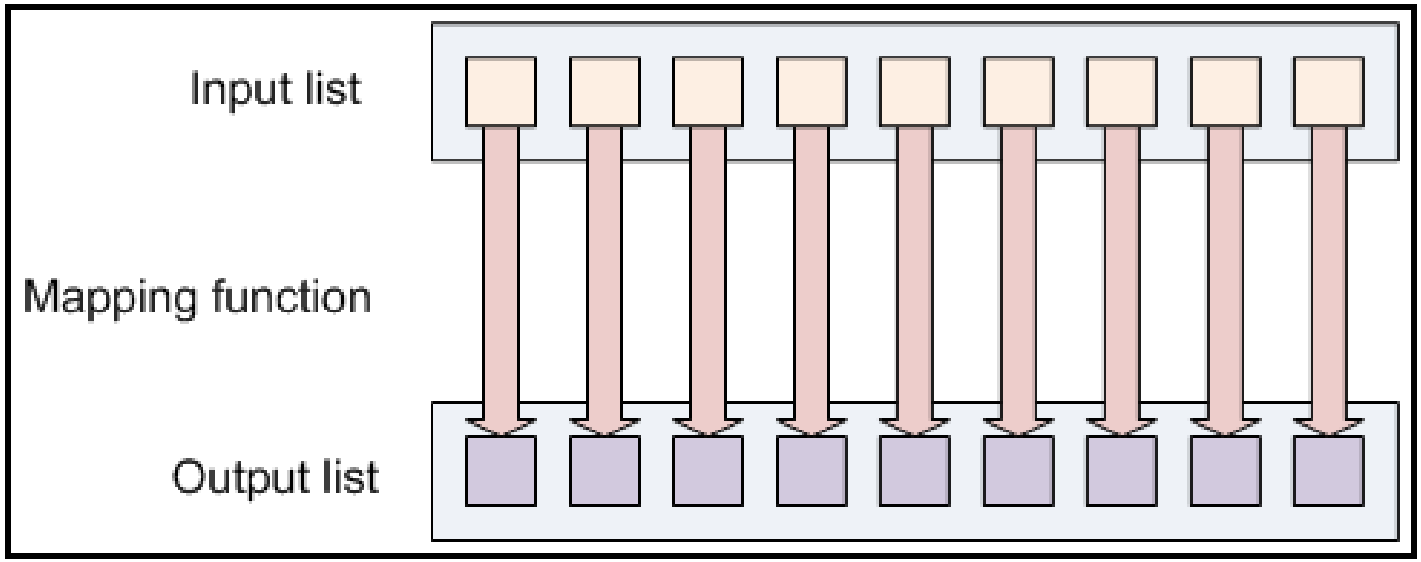
Building Blocks

- Map/Reduce programs transform lists of input data elements into lists of output data elements
- Twice
  - Map
  - Reduce

Building Blocks: Map

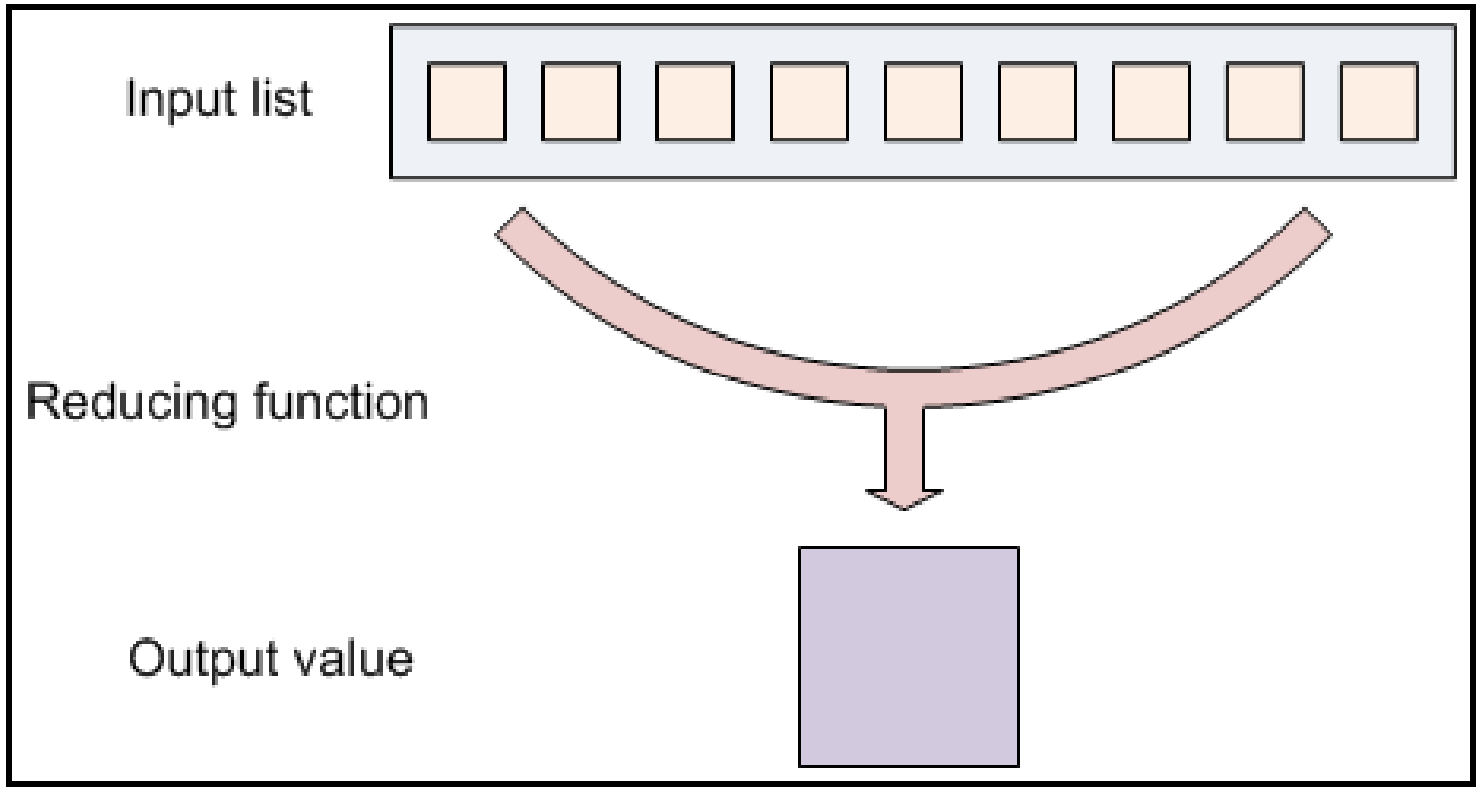
- list of input data elements is provided, one at a time
- Mapper transforms each element individually to an output data element
  - Hadoop: one or more output data elements

- Suppose you had a function `toUpper(str)` which returns an uppercase version of its input …



Building Blocks: Reduce

- lets you aggregate values
- receives an iterator of input (output in the first place) values
- then combines these values together, returning a single output value
  - Hadoop: one or more output data elements



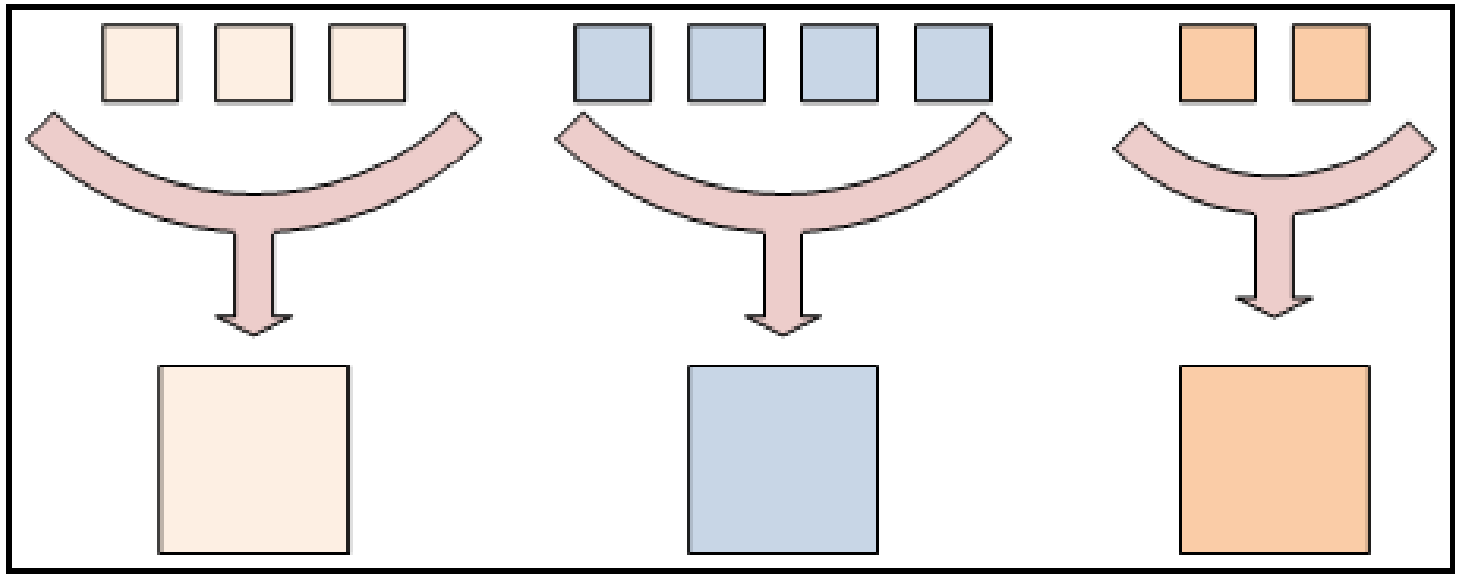
Building Blocks: Map/Reduce

- Map/Reduce program has two components: one that implements the *mapper*, and another that implements the *reducer*
- no value stands on its own: every value has a *key*
- Keys identify related values
- mapping and reducing functions receive (key, value) pairs
- mapping and reducing functions output (key, value) pairs

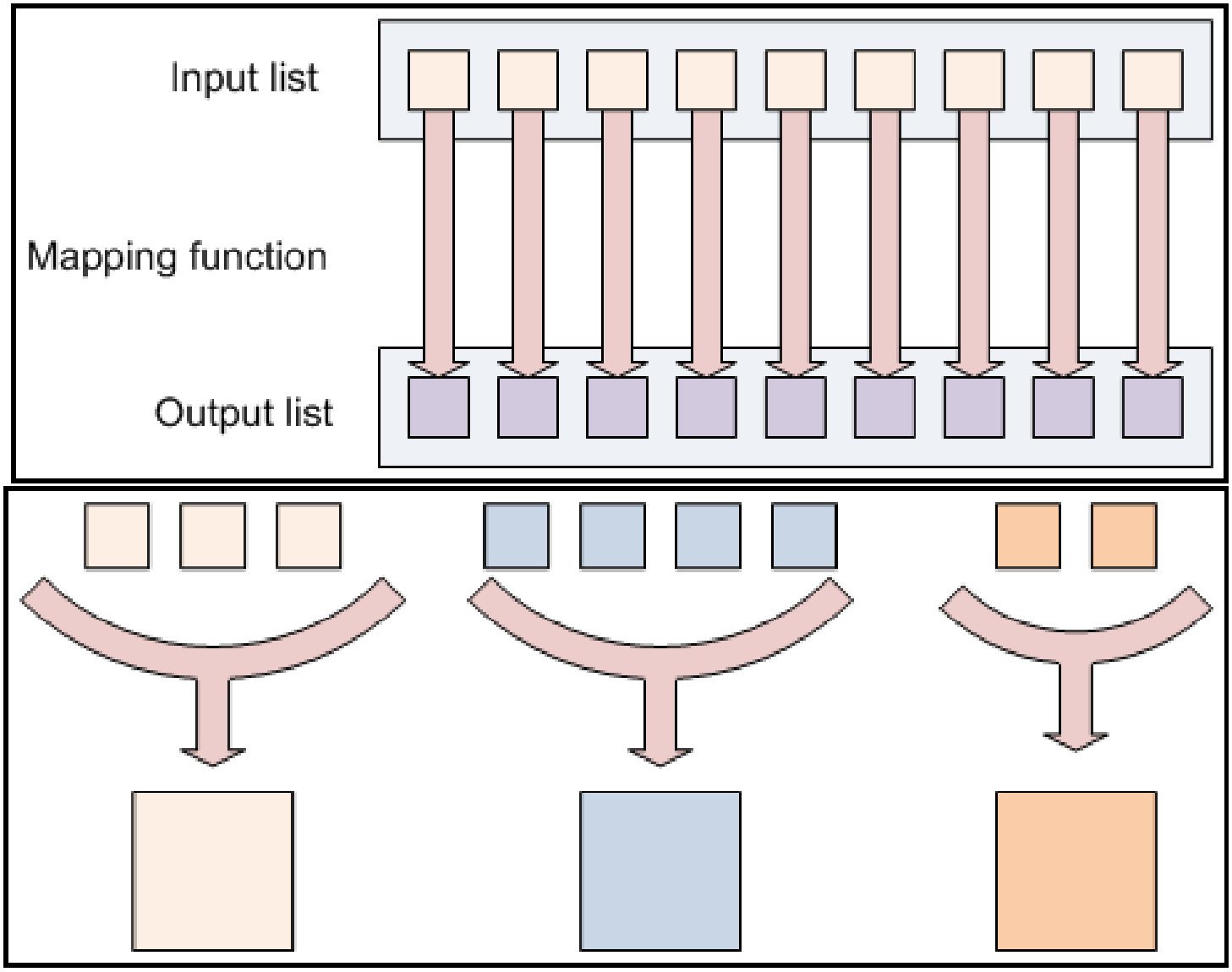
- Example: log of time-coded speedometer readings from multiple cars
  - **AAA-123 65mph, 12:00pm**
  - **ZZZ-789 50mph, 12:02pm**
  - **AAA-123 40mph, 12:05pm**
  - **CCC-56 25mph, 12:15pm**

Building Blocks: Map/Reduce

- **Keys divide the reduce space**
- reducing function turns a large list of values into one (or a few) output value(s)
- all of the output values are not reduced together
- values *with the same key* are presented to a single reducer
- performed independently of any reduce operations occurring on other lists of values (with different keys)



Building Blocks: Overview



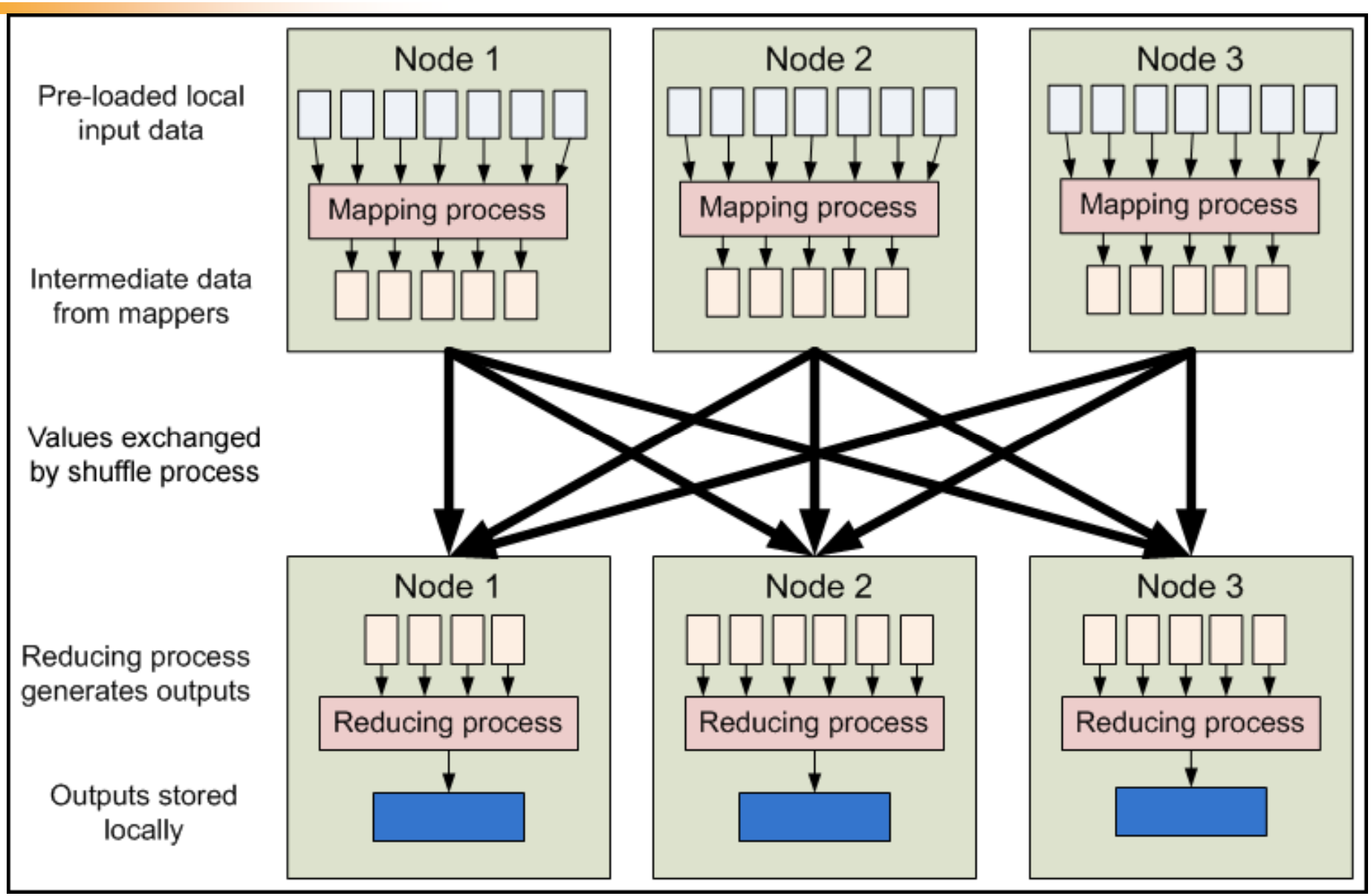
Example: Word Count

- how many times different words appear in a set of files
- **foo.txt:** Sweet, this is the foo file
- **bar.txt:** This is the bar file
- Expected output: sweet (1) this (2) is (2) the (2) foo (1) bar (1) file (2)

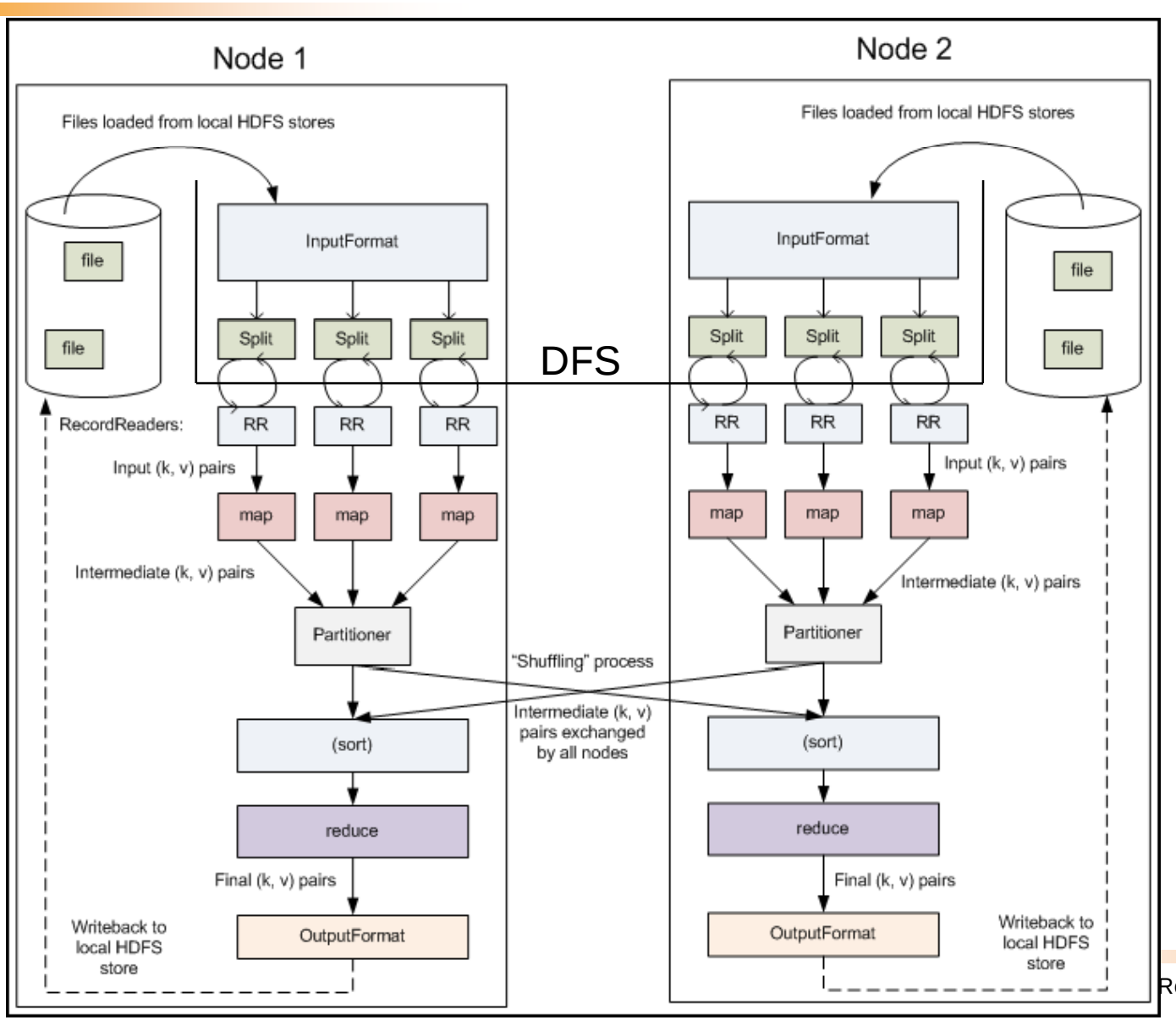
```
mapper (filename, file-contents):
  for each word in file-contents:
    emit (word, 1)

reducer (word, values):
  sum = 0
  for each value in values:
    sum = sum + value
  emit (word, sum)
```

- several instances of the mapper are created
- each instance receives different input file
- mappers output (word, 1) pairs → reducers
- several instances of the reducer are instantiated
- each reducer is responsible for processing the list of values associated with a different key (word)
- list of values will be a list of 1's
- reducer sums up those ones into a final count
- reducer emits the final (word, count)
- Note: default input format used by Hadoop presents each line of an input file as a separate input to the mapper function, not the entire file at a time; key is byte offset in file

Hadoop specifics: Data Flow (High Level)



Hadoop specifics: Data Flow (Low Level)

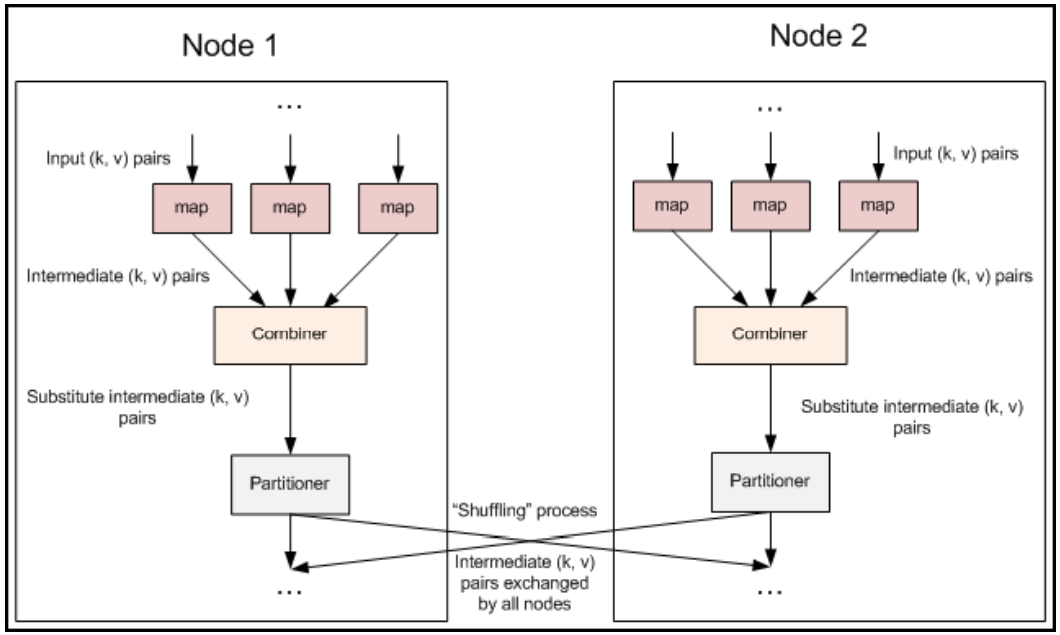


Hadoop specifics: Combiner

- remember Word Count example; picture one specific node

`map → (word_1, 1) (word_2, 1) (word_1, 1) (word_3, 1) …`

`reduce → (word_1, 2) (word_2, x) (word_3, y) …`



`map → (word_1, 1) (word_2, 1) (word_1, 1) (word_3, 1) …`

`combine → (word_1, u) (word_2, v) (word_3, w) … (local)`

`reduce → (word_1, x) (word_2, y) (word_3, z) … (global)`

Hadoop specifics: Driver Method

- initializes the job and instructs the Hadoop platform to execute your code

```
public void run(String inputPath, String outputPath) throws
                                                    Exception {
   JobConf conf = new JobConf(WordCount.class);
   conf.setJobName("wordcount");
   // the keys are words (strings)
   conf.setOutputKeyClass(Text.class);
   // the values are counts (ints)
   conf.setOutputValueClass(IntWritable.class);
   conf.setMapperClass(YourMapper.class);
   conf.setReducerClass(YourReducer.class);
   FileInputFormat.addInputPath(conf, new Path(inputPath));
   FileOutputFormat.setOutputPath(conf, new Path(outputPath));
   JobClient.runJob(conf);
}
```

Tips: Chaning Jobs, Debugging

- Not every problem can be solved with one Map/Reduce program
- Fewer are those which can be solved with a single job
- Many problems can be solved with Map/Reduce, by implementing several Map/Reduce steps which run in series:
  - Map1 -> Reduce1 -> Map2 -> Reduce2 -> Map3...

- Log files: *hadoop/logs/*.log*
- Named according to components: *namenode*, *datanode*, *jobtracker*, or *tasktracker*
- for individual programs ***tasktracker*** logs will be the most relevant
  - Exception thrown by user prog is recorded here
- Two more files in *hadoop/logs/userlogs/*: *stsout*, *stderr*
- on a multi-node Hadoop cluster logs are not centrally aggregated
- Run Hadoop locally!

Questions ?