The Streaming API: Motivation

- **Many organizations have developers skilled in languages other than Java, such as**
  - Ruby
  - Python
  - Perl
- **The Streaming API allows developers to use any language they wish to write Mappers and Reducers**
  - As long as the language can read from standard input and write to standard output

The Streaming API: Advantages and Disadvantages

- **Advantages of the Streaming API:**
  - No need for non-Java coders to learn Java
  - Fast development time
  - Ability to use existing code libraries
- **Disadvantages of the Streaming API:**
  - Performance
  - Primarily suited for handling data that can be represented as text
  - Streaming jobs can use excessive amounts of RAM or fork excessive numbers of processes
  - Although Mappers and Reducers can be written using the Streaming API, Partitioners, InputFormats etc. must still be written in Java

How Streaming Works

- **To implement streaming, write separate Mapper and Reducer programs in the language(s) of your choice**
  - They will receive input via `stdin`
  - They should write their output to `stdout`
- **If `TextInputFormat` (the default) is used, the streaming Mapper just receives each line from the file on `stdin`**
  - No key is passed
- **Mapper and Reducer output should be sent to `stdout` as**
  - `key` *`[tab]`* `value` *`[newline]`*
- **Separators other than tab can be specified**

Streaming: Example Mapper

- **Example streaming wordcount Mapper:**

```
#!/usr/bin/env perl
while (<>) {                    # Read lines from stdin
  chomp;                        # Get rid of the trailing newline
  (@words) = split /\W+/;       # Create an array of words
  foreach $w (@words) {         # Loop through the array
    print "$w\t1\n";            # Print out the key and value
  }
}
```

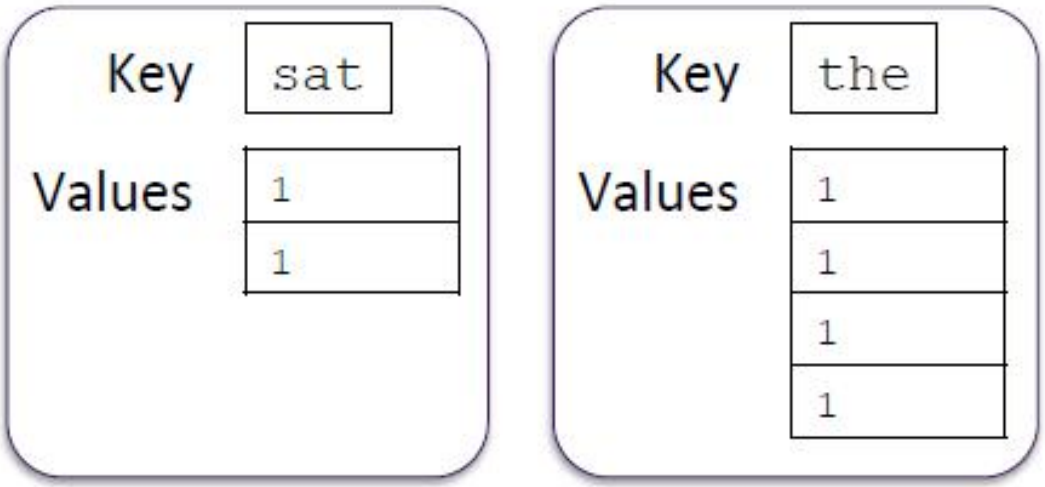
Python MapReduce Code:

```
mapper.py
#!/usr/bin/python
import sys
#Word Count Example
# input comes from standard input STDIN
for line in sys.stdin:
line = line.strip() #remove leading and trailing whitespaces
words = line.split() #split the line into words and returns as a list
for word in words:
#write the results to standard output STDOUT
print'%s    %s' % (word,1) #Emit the word
```

Streaming Reducers: Caution

- **Recall that in Java, all the values associated with a key are passed to the Reducer as an `Iterable`**
- **Using Hadoop Streaming, the Reducer receives its input as one key/value pair per line**
- **Your code will have to keep track of the key so that it can detect when values from a new key start**

Java: Iterable Objects



Streaming: stdin

```
sat     1
sat     1
the     1
the     1
the     1
the     1
```

Streaming: Example Reducer

- **Example streaming wordcount Reducer:**

```
#!/usr/bin/env perl
$sum = 0;
$last = "";
while (<>) {                          # read lines from stdin
  ($key, $value) = split /\t/;        # obtain the key and value
  $last = $key if $last eq "";        # first time through
  if ($last ne $key) {                # has key has changed?
    print "$last\t$sum\n";            # if so output last key/value
    $last = $key;                     # start with the new key
    $sum = 0;                         # reset sum for the new key
  }
  $sum += $value;                     # add value to tally sum for key
}
print "$key\t$sum\n";                 # print the final pair
```

reducer.py

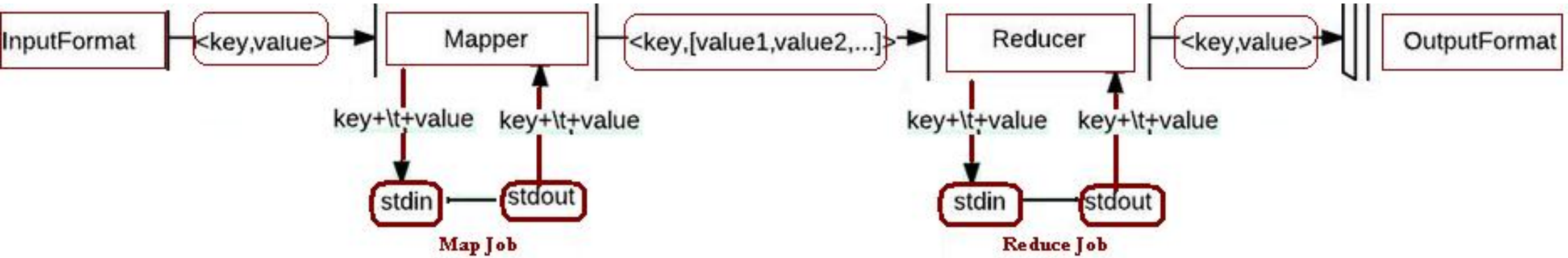
```
#!/usr/bin/python
import sys
from operator import itemgetter
# using a dictionary to map words to their counts
current_word = None
current_count = 0
word = None
# input comes from STDIN
for line in sys.stdin:
line = line.strip()
word,count = line.split('    ',1)
try:
count = int(count)
except ValueError:
continue
if current_word == word:
current_count += count
else:
if current_word:
print '%s   %s' % (current_word, current_count)
current_count = count
current_word = word
if current_word == word:
print '%s   %s' % (current_word,current_count)
```

Launching a Streaming Job

- **To launch a Streaming job, use e.g.:**

```
$ hadoop jar /usr/lib/hadoop-0.20-mapreduce/contrib/\
  streaming/hadoop-streaming*.jar \
  -input myInputDirs \
  -output myOutputDir \
  -mapper myMapScript.pl \
  -reducer myReduceScript.pl \
  -file mycode/myMapScript.pl \
  -file mycode/myReduceScript.pl
```

- **Many other command-line options are available**
  - See the documentation for full details
- **Note that system commands can be used as a Streaming Mapper or Reducer**
  - For example: `awk`, `grep`, `sed`, or `wc`


InputFormat
<key,value>
Mapper
<key,[value1,value2,...]>
Reducer
<key,value>
OutputFormat
key+\t+value
key+\t+value
key+\t+value
key+\t+value
stdin
stdout
stdin
stdout
Map Job
Reduce Job

```
$ bin/hadoop jar contrib/streaming-hadoop-0.18.0-streaming.jar  \
 -mapper     streamMapProgram \
 -reducer    streamReduceProgram \
 -input      dfs/path \
 -output     dfs/path
```