Computing the Mean: Version 1

- Mean(1, 2, 3, 4, 5) = (1+2+3+4+5) / 5 = 3

  - Mean(Mean(1, 2) = (1+2) /2 = 1.5;
  - Mean(3, 4, 5)) = (3+4+5) / 3 = 4
  -

Computing the Mean

- Can we use the reducer as a combiner?

Algorithm 3.4 Compute the mean of values associated with the same key

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

class REDUCER
  method REDUCE(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**
      $sum \leftarrow sum + r$
      $cnt \leftarrow cnt + 1$
    $r_{avg} \leftarrow sum/cnt$
    EMIT(string t, integer r_{avg})

Computing the Mean: Version 2

Does this work?

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

```
class Combiner
    method Combine(string t, integers [r1, r2, ...])
        sum ← 0
        cnt ← 0
        for all integer r ∈ integers [r1, r2, ...] do
            sum ← sum + r
            cnt ← cnt + 1
        Emit(string t, pair (sum, cnt))        ▷ Separate sum and count
```

Computing the Mean: Version 2

Does this work?

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

class Combiner
    method Combine(string t, integers [r1, r2, ...])
        sum ← 0
        cnt ← 0
        for all integer r ∈ integers [r1, r2, ...] do
            sum ← sum + r
            cnt ← cnt + 1
        Emit(string t, pair (sum, cnt))                ▷ Separate sum and count

class Reducer
    method Reduce(string t, pairs [(s1, c1), (s2, c2) ...])
        sum ← 0
        cnt ← 0
        for all pair (s, c) ∈ pairs [(s1, c1), (s2, c2) ...] do
            sum ← sum + s
            cnt ← cnt + c
        r_avg ← sum/cnt
        Emit(string t, integer r_avg)
```

Computing the Mean:

- Fixed?

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})

Average Temperatures

```python
from mrjob.job import MRJob
class AvgTemperature(MRJob):
    def mapper(self, _, line):
      month, temperature = line.split()
      yield (month, (int(temperature),1))
```

```python
def combiner(self, month, temperatures):
  sum, count = 0, 0
  for tmp, c in temperatures:
    sum = sum + tmp
    count += c
  yield (month, (sum, count))
```

```python
  def reducer(self, month, temperatures):
    month, (avg, count) = self._reducer_combiner(month, temperatures)
    # (May, 28 Degrees)
    yield (month, avg)
```

```python
def _reducer_combiner(self, month, temperatures):
  sum, count = 0, 0
  for tmp, c in temperatures:
    sum = sum + tmp
    count += c
  avg = sum/count
  return (month, (avg, count))
```

Input

```
Jan -9
Jan -8
Feb 17
Feb -9
Mar 1
Apr 10
Apr 20
May 18
Mar 3
Jun 19
Jun 25
Apr 8
May 11
```

Output

```
!python avg2.py temp.txt

"Apr"   12.666666666666666
"Feb"   2.0
"Jan"   -4.25
"Jun"   22.0
"Mar"   2.0
"May"   14.5
```

Example: Analysis of 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 with many elements*
  - *We focus on the temperature element*
  - *Data files are organized by date and weather station*

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

Example: Analysis of Weather Dataset

- **Query:** What's the highest recorded global temperature for each year in the dataset?

Complete run for the century took **42 minutes** on a single EC2 High-CPU Extra Large Instance

To speed up the processing, we need to run parts of the program in **parallel**

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

Hadoop MapReduce

- To use MapReduce, we need to express out query as a MapReduce job

- MapReduce job
  - *Map function*
  - *Reduce function*

- Each function has key-value pairs as input and output
  - *Types of input and output are chosen by the programmer*

MapReduce Design of NCDC Example

- **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)

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

Input File

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

The output from the map function is processed by MapReduce framework

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

Sort and Group By

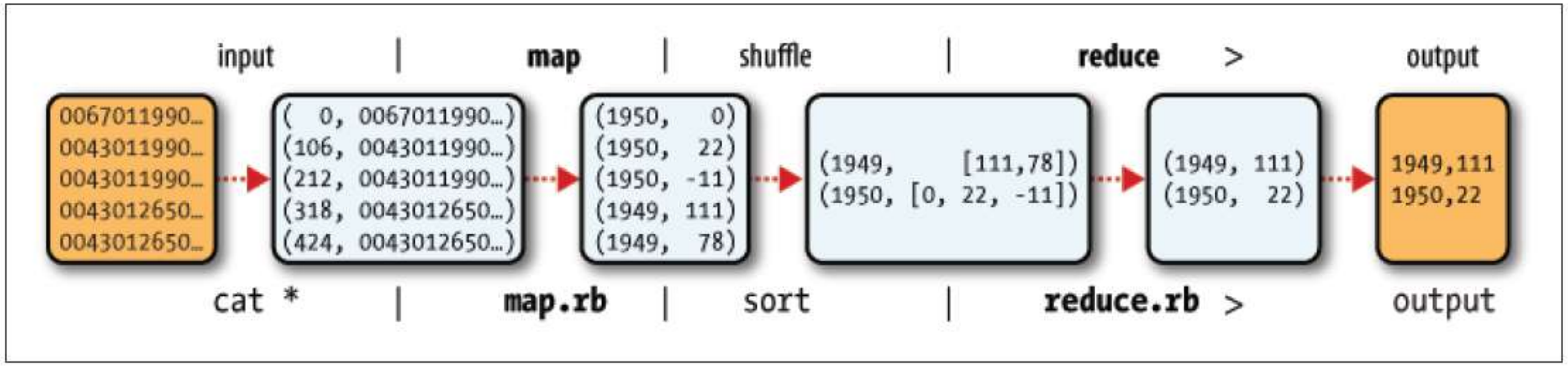
```
(1949, [111, 78])
(1950, [0, 22, -11])
```

- 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

The output from the map function is processed by MapReduce framework

```
(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)
```

Any improvement that you can suggest ?

Shuffle and Sort

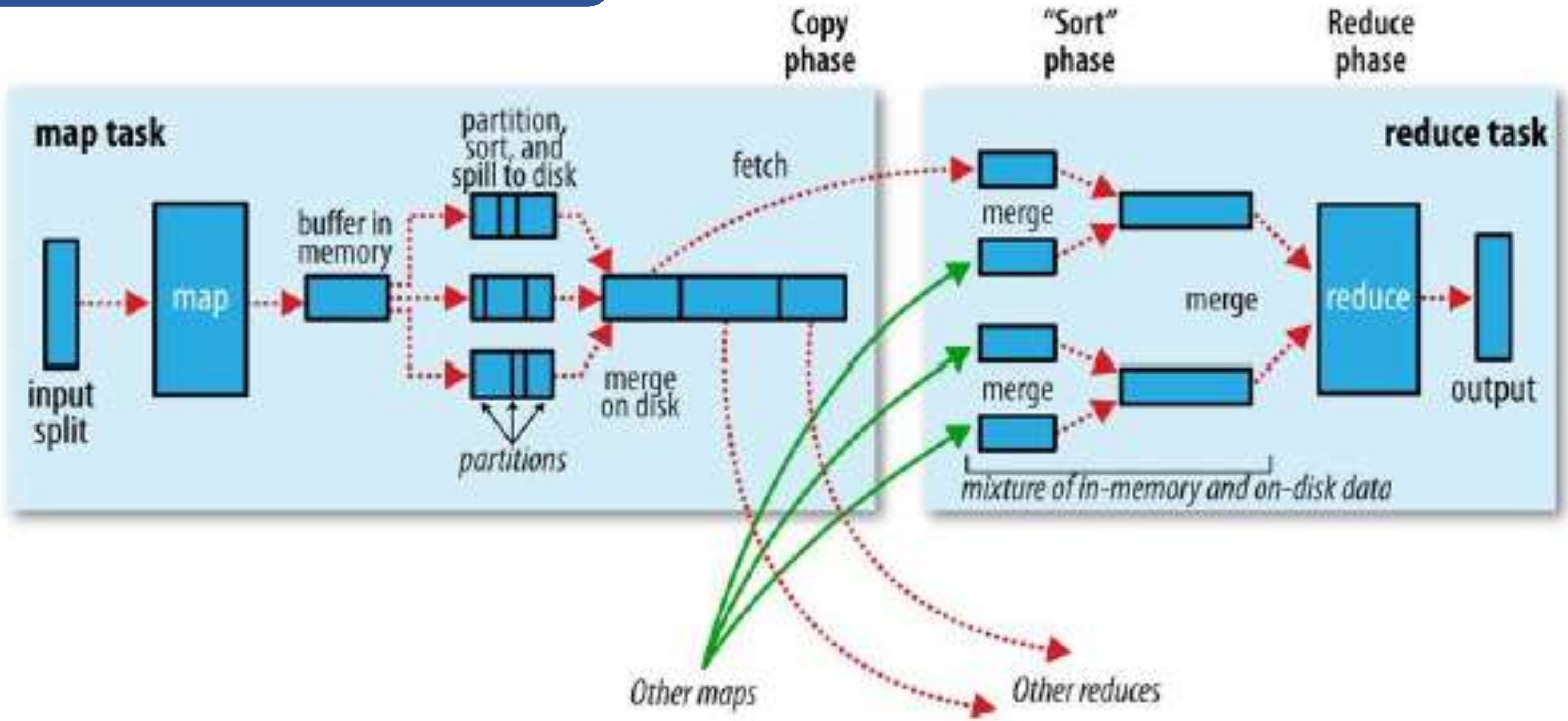


Figure 7-4. Shuffle and sort in MapReduce

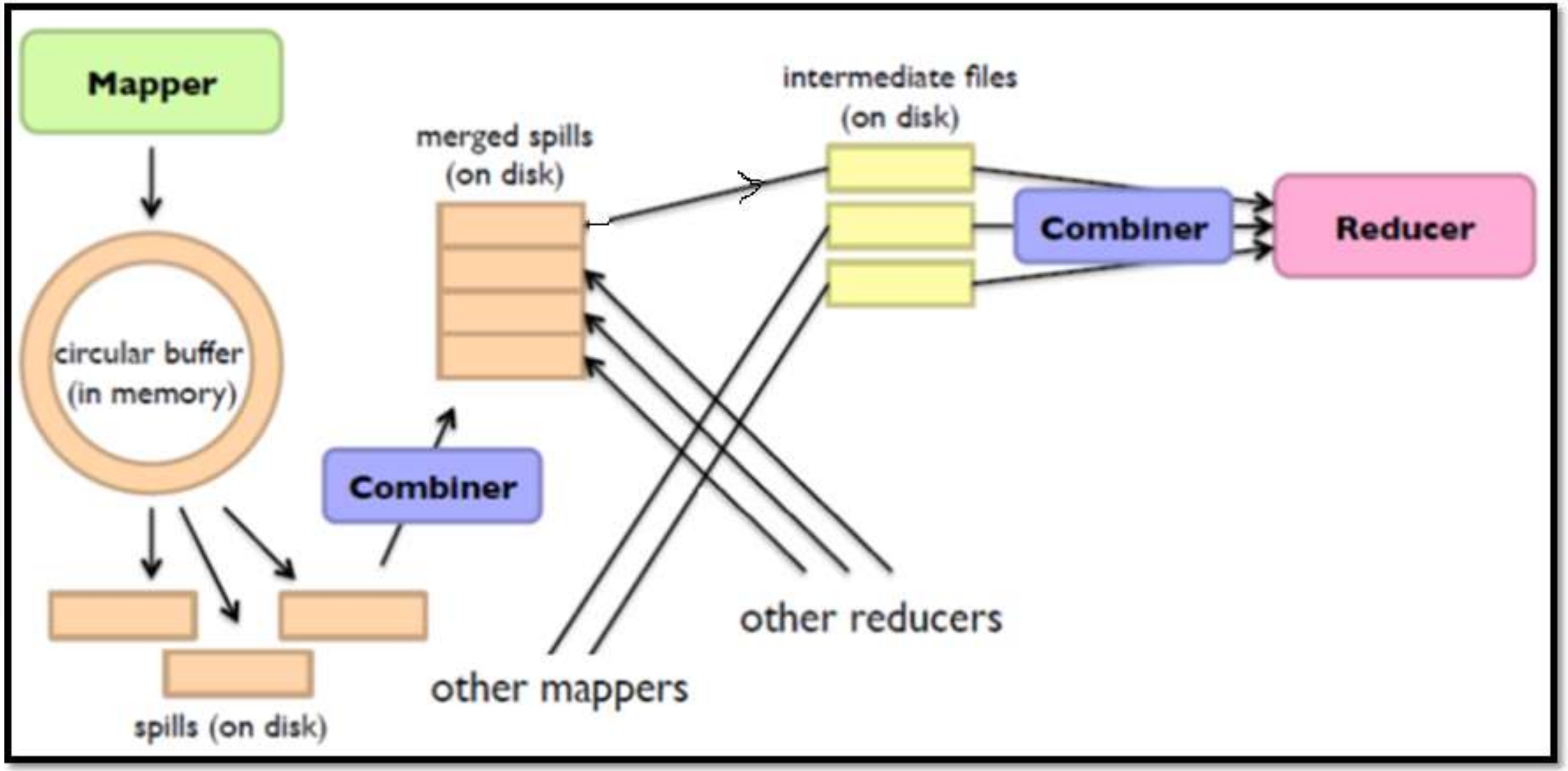
Shuffle and Sort

Map side

- Map outputs are buffered in memory in a **circular buffer**
- When buffer fills contents are "spilled" to disk
- **Spills merged** in a single, partitioned file (sorted within each partition): **combiner** runs during the merges

Reduce side

- Map outputs are copied to **reducer machine**
- **"Sort" is a multi-pass merge** of map outputs (happens in memory and on disk): **combiner** runs during the merges
- **Final merge pass** goes directly into reducer



MRJob

- A job is defined by a class that inherits from MRJob. This class contains methods that define the steps of your job.

- A “step” consists of a mapper, a combiner, and a reducer.
  - *All of these are optional, though you must have at least one.*
  - *So you could have a step that’s just a mapper, or just a combiner and a reducer.*
- When you only have one step, all you have to do is write methods called mapper(), combiner(), and reducer().

MRJob

- Most of the time, you'll need more than one step in your job.
- To define multiple steps, override [steps()]() to return a list of [MRSteps]().

```
from mrjob.job import MRJob
from mrjob.step import MRStep
import re

WORD_RE = re.compile(r"[\w']+")


class MRMostUsedWord(MRJob):

    def steps(self):
        return [
            MRStep(mapper=self.mapper_get_words,
                   combiner=self.combiner_count_words,
                   reducer=self.reducer_count_words),
            MRStep(reducer=self.reducer_find_max_word)
        ]
```

MRJob

- Most of the time, you'll need more than one step in your job.
- To define multiple steps, override steps() to return a list of MRSteps.

```
def mapper_get_words(self, _, line):
    # yield each word in the line
    for word in WORD_RE.findall(line):
        yield (word.lower(), 1)

def combiner_count_words(self, word, counts):
    # optimization: sum the words we've seen so far
    yield (word, sum(counts))
```

MRJob

- Most of the time, you'll need more than one step in your job.
- To define multiple steps, override steps() to return a list of MRSteps.

```
def mapper_get_words(self, _, line):
    # yield each word in the line
    for word in WORD_RE.findall(line):
        yield (word.lower(), 1)

def combiner_count_words(self, word, counts):
    # optimization: sum the words we've seen so far
    yield (word, sum(counts))

def reducer_count_words(self, word, counts):
    # send all (num_occurrences, word) pairs to the same reducer.
    # num_occurrences is so we can easily use Python's max() function.
    yield None, (sum(counts), word)
```

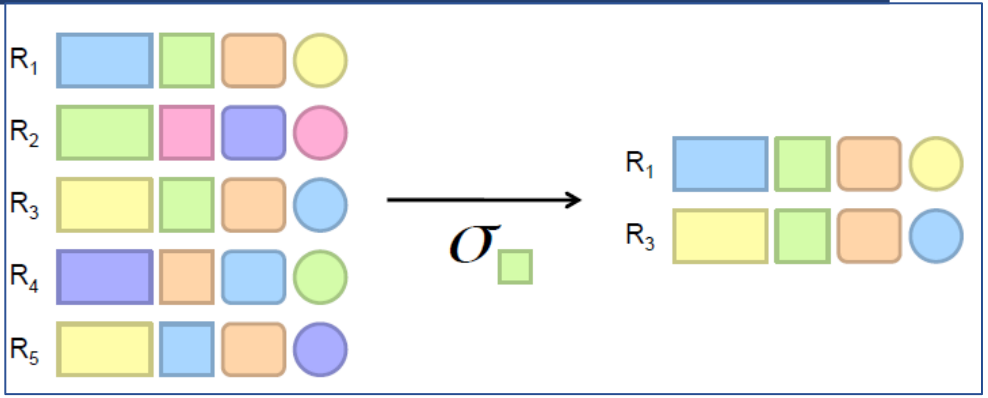
MRJob

- Most of the time, you'll need more than one step in your job.
- To define multiple steps, override steps() to return a list of MRSteps.

```
    # discard the key; it is just None
    def reducer_find_max_word(self, _, word_count_pairs):
        # each item of word_count_pairs is (count, word),
        # so yielding one results in key=counts, value=word
        yield max(word_count_pairs)


if __name__ == '__main__':
    MRMostUsedWord.run()
```

Operations

Find error msg from huge weblog.



Map Function:

- Filter and Emit error msg

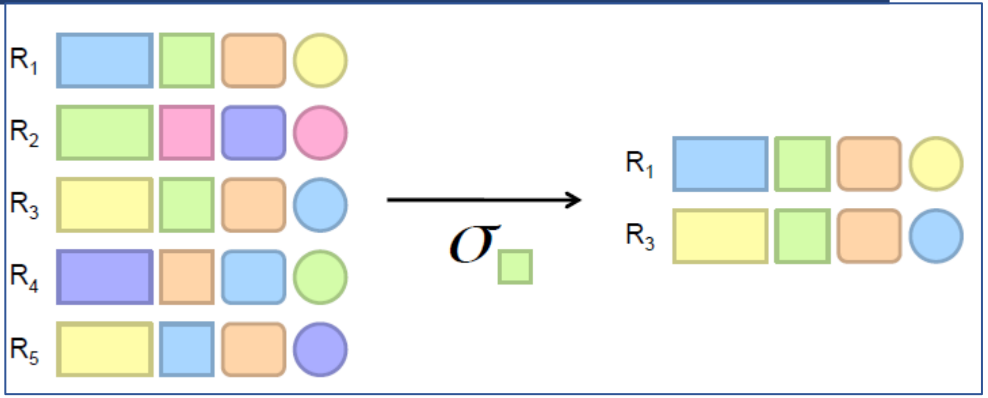
Reduce Function:

- No Reducer Necessary (unless you want to do something else)

Operations

Selection:

- *Select error msg from huge weblog.*



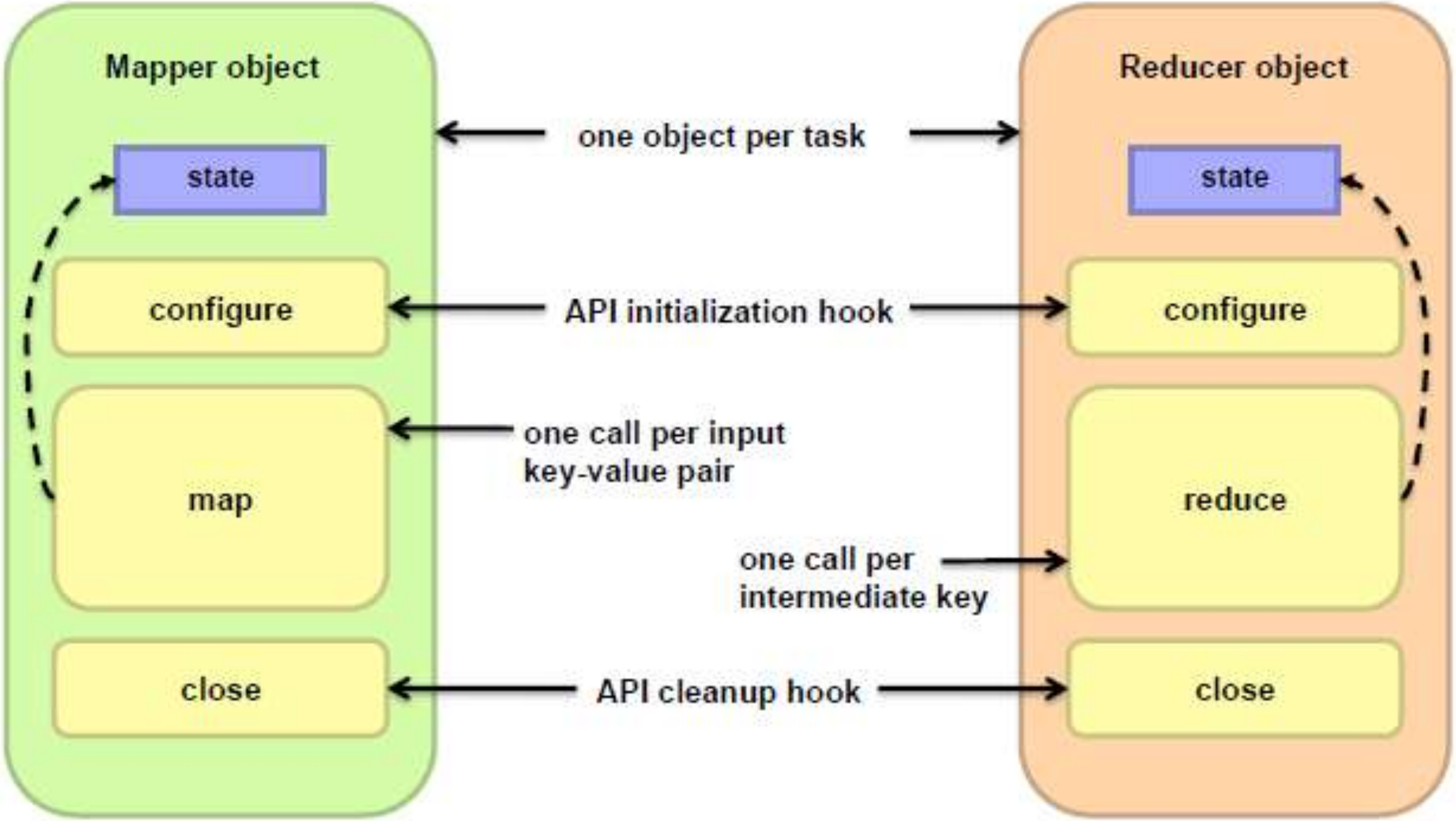
Map Function:

- Filter and Emit error msg

Reduce Function:

- No Reducer Necessary (unless you want to do something else)

Preserving State



Setup and teardown of tasks

- What if we need to load some kind of support file or a temporary file,
  - *Example :GREP we are searching for a particular pattern*

    - `mapper_init()`
    - `combiner_init()`
    - `reducer_init()`
    - `mapper_final()`
    - `combiner_final()`
    - `reducer_final()`

https://mrjob.readthedocs.io/en/latest/guides/writing-mrjobs.html#setup-and-teardown-of-tasks

Wordcount using init method

```
from mrjob.job import MRJob
from mrjob.step import MRStep

class MRWordFreqCount(MRJob):

    def init_get_words(self):
        self.words = {}

    def get_words(self, _, line):
        for word in WORD_RE.findall(line):
            word = word.lower()
            self.words.setdefault(word, 0)
            self.words[word] = self.words[word] + 1

    def final_get_words(self):
        for word, val in self.words.iteritems():
            yield word, val
```

Wordcount using init method

```
def sum_words(self, word, counts):
    yield word, sum(counts)

def steps(self):
    return [MRStep(mapper_init=self.init_get_words,
                   mapper=self.get_words,
                   mapper_final=self.final_get_words,
                   combiner=self.sum_words,
                   reducer=self.sum_words)]
```

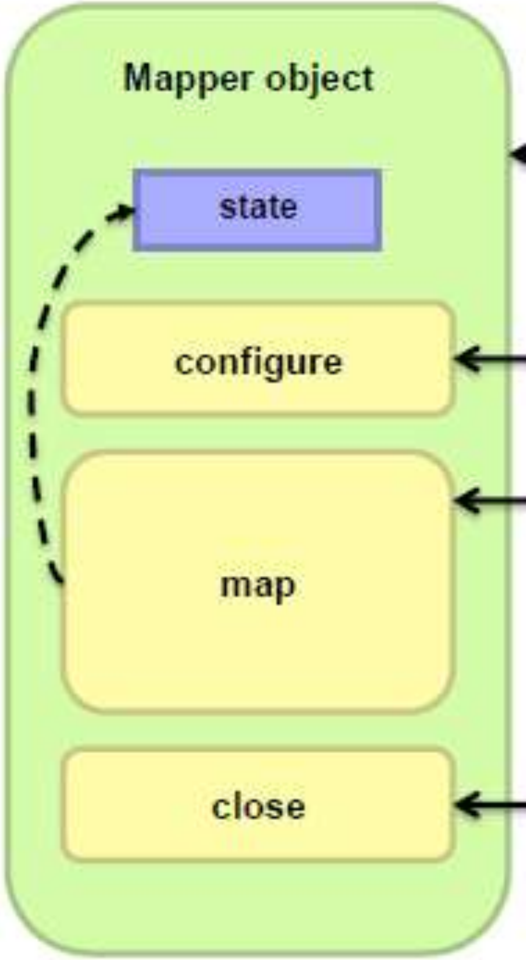
Wordcount using init method

```python
class MRWordFreqCount(MRJob):

    def init_get_words(self):
        self.words = {}

    def get_words(self, _, line):
        for word in WORD_RE.findall(line):
            word = word.lower()
            self.words.setdefault(word, 0)
            self.words[word] = self.words[word] + 1

    def final_get_words(self):
        for word, val in self.words.iteritems():
            yield word, val

    def sum_words(self, word, counts):
        yield word, sum(counts)

    def steps(self):
        return [MRStep(mapper_init=self.init_get_words,
                       mapper=self.get_words,
                       mapper_final=self.final_get_words,
                       combiner=self.sum_words,
                       reducer=self.sum_words)]
```

Word Count: Aggregate in Mapper

Algorithm 3.3 Word count mapper using the "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 documen
    method Close
        for all term t ∈ H do
            Emit(term t, count H{t})
```



Are combiners still needed?

Home Work

- Compute mean temperature of each year using Associative memory

Algorithm Design: Example

- **Term co-occurrence matrix for a text collection**
  - *M = N x N matrix (N = vocabulary size)*
  - ***Mij: number of times i and j co-occur** in some context*
    (for concreteness, let's say context = sentence)

- Why?
- Distributional profiles as a way of measuring semantic distance
- Semantic distance is useful for many language processing tasks

MapReduce: Large Counting Problems

- Term co-occurrence matrix for a text collection

  = specific instance of a large counting problem
  - *A large event space (number of terms)*
  - *A large number of observations (the collection itself)*
  - *Goal: keep track of interesting statistics about the events*

- **Basic approach**
  - *Mappers generate partial counts*
  - *Reducers aggregate partial counts*
- How do we aggregate partial counts efficiently?

Pairs Approch

- First Try: “Pairs”

- Each mapper takes a sentence:
  - *Generate all co-occurring term pairs*
  - *For all pairs, emit (a, b) ⟶ count*
- Reducers sum up counts associated with these pairs
- Use combiners!

Pairs: Pseudo-Code

Algorithm 3.8 Compute word co-occurrence ("pairs" approach)

```
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)        ▷ Emit count for each co-occurrence
```

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

“Pairs” Analysis

Advantages

- Easy to implement, easy to understand

Disadvantages

- Lots of pairs to sort and shuffle around (upper bound?)
- Not many opportunities for combiners to work

Try: "Stripes"

- Idea: group together pairs into an associative array

  (a, b) → 1
  (a, c) → 2
  (a, d) → 5 a → { b: 1, c: 2, d: 5, e: 3, f: 2 }
  (a, e) → 3
  (a, f) → 2

- Each mapper takes a sentence:
  - *Generate all co-occurring term pairs*
  - *For each term, emit a → { b: count$_b$, c: count$_c$, d: count$_d$ … }*
- Reducers perform element-wise sum of associative arrays

  a → { b: 1, d: 5, e: 3 }
  + a → { b: 1, c: 2, d: 2, f: 2 }
  a → { b: 2, c: 2, d: 7, e: 3, f: 2 }

Key: cleverly constructed data structure brings together partial results

Stripes: Pseudo-Code

What are the advantages of stripes?

Algorithm 3.9 Compute word co-occurrence ("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                ▷ Tally words co-occurring with w
            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)                          ▷ Element-wise sum
        Emit(term w, stripe Hf)
```

Stripes - Analysis

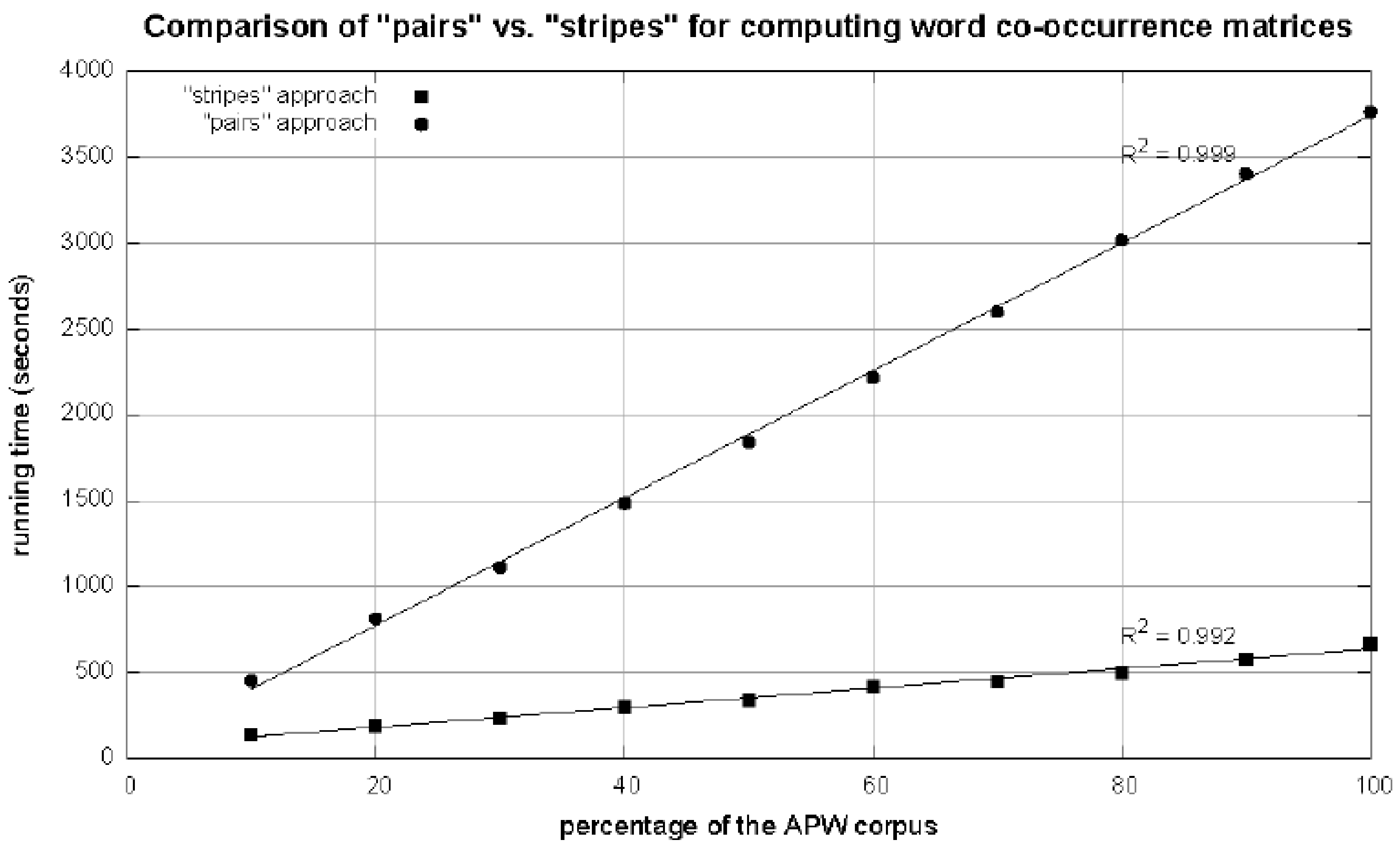
- **Advantages**
  - *Far less sorting and shuffling of key-value pairs*
  - *Can make better use of combiners*

- **Disadvantages**
  - *More difficult to implement*
  - *Underlying object more heavyweight*
  - *Fundamental limitation in terms of size of event space*

What about combiners?

- Both algorithms can benefit from the use of combiners,
  - *As the respective operations in their reducers (addition and element-wise sum of associative arrays) are both commutative and associative.*

- Are combiners equally effective in both pairs and stripes?



Cluster size: 38 cores
Data Source: Associated Press Worldstream (APW) of the English Gigaword Corpus (v3), which contains 2.27 million documents (1.8 GB compressed, 5.7 GB uncompressed)

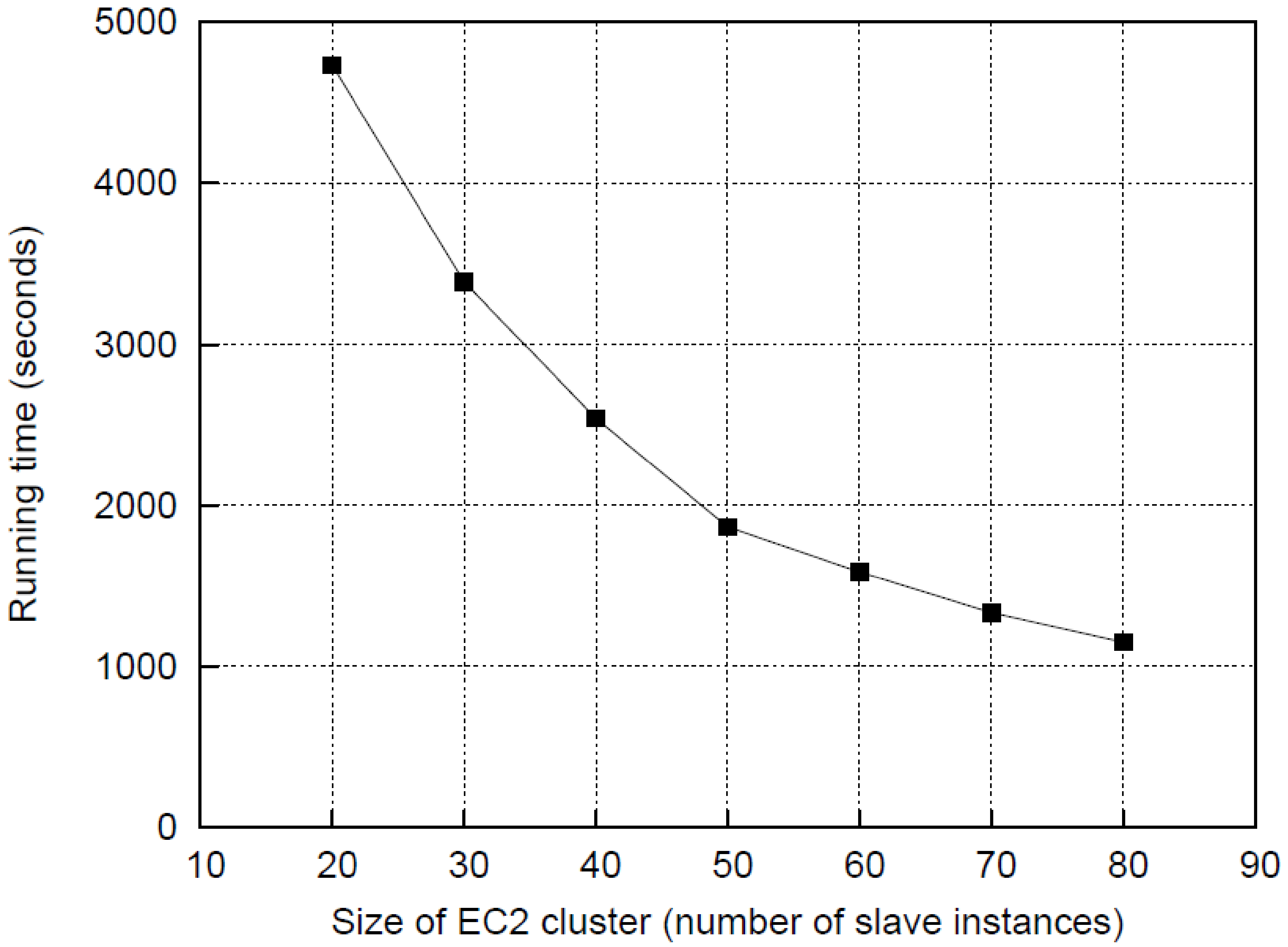


Figure 3.2: Running time of the stripes algorithm on the APW corpus with Hadoop clusters of different sizes from EC2

PAIRS VS STRIPES

- Pairs and stripes approaches represent endpoints along a continuum of possibilities.

- The pairs approach individually records each co-occurring event,
- The stripes approach records all co-occurring events with respect a conditioning event.

- A middle ground ...?