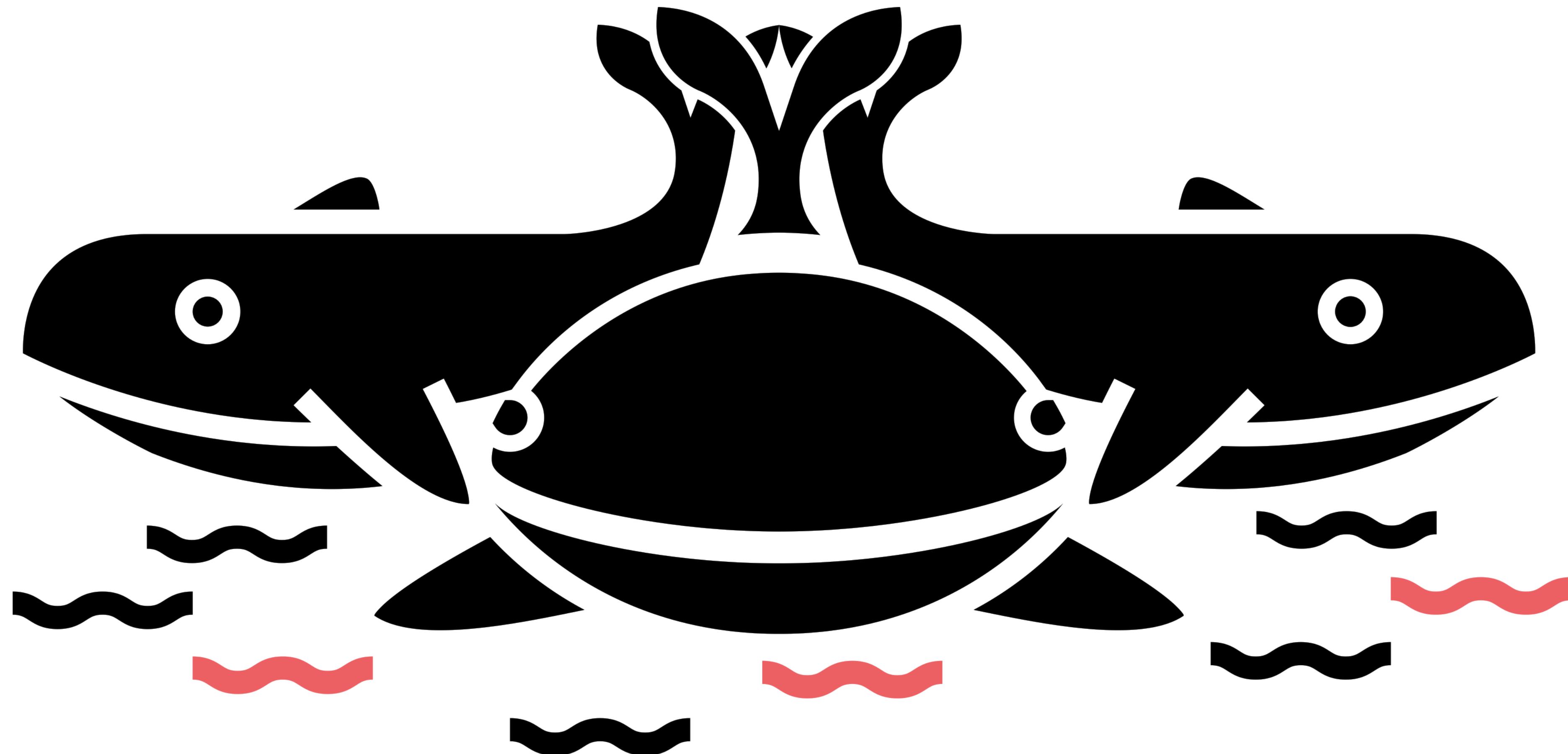


**Yandex**

# World of MapReduce



Combiner

Partitioner

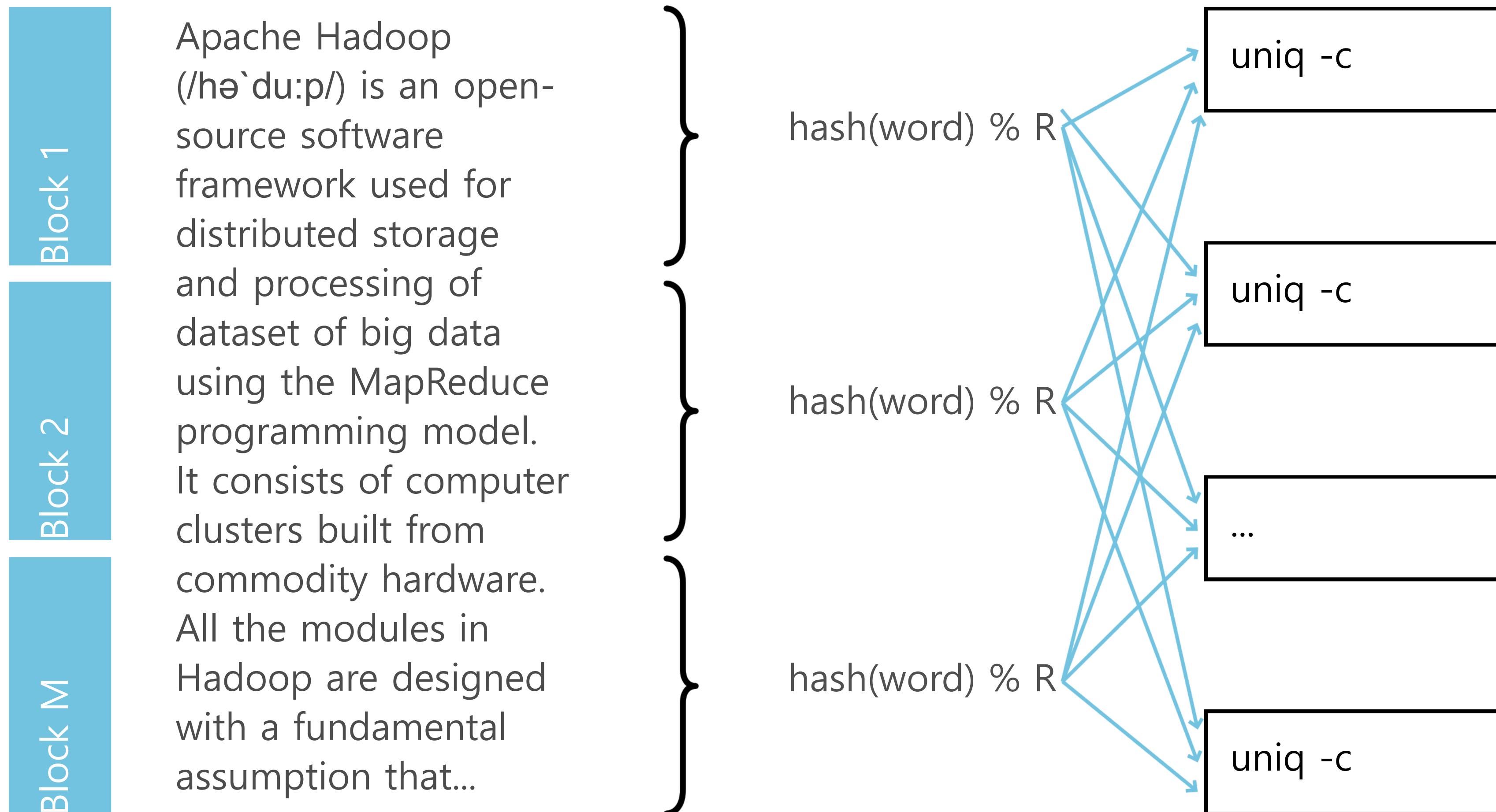
Comparator

# MapReduce

Combiner

# WordCount

```
cat wikipedia.dump | tr ' ' '\n' | sort | uniq -c
```



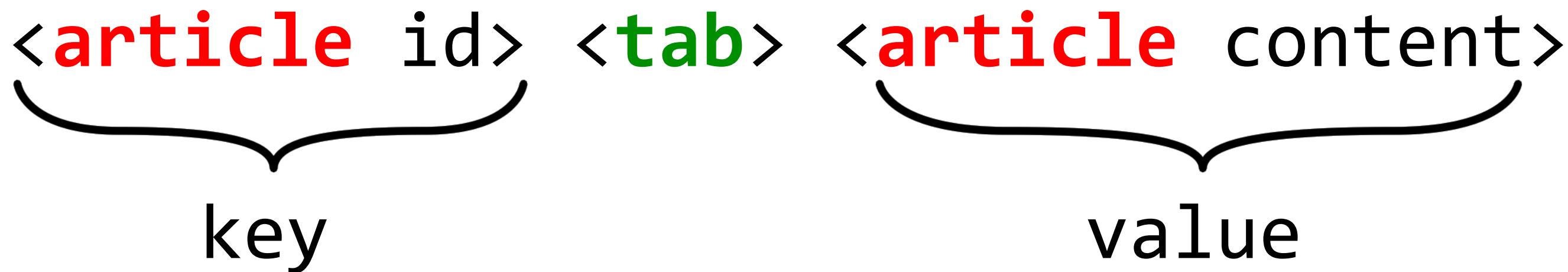
wikipedia.dump -> map () -> word

shuffle & sort

reduce()



WIKIPEDIA  
The Free Encyclopedia



**input:** word word a word b c d word d e...

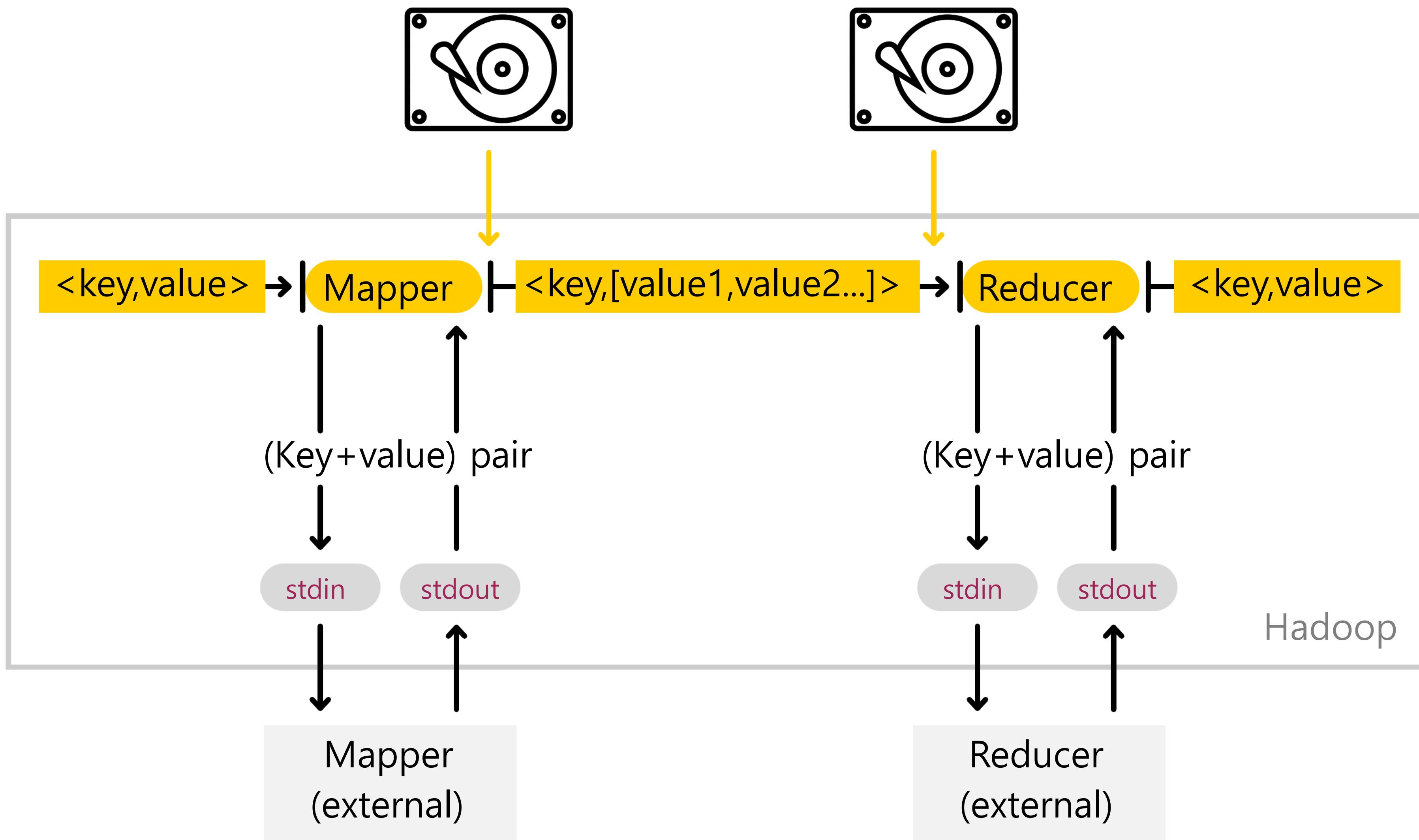
Mapper (Python): reporter\_mapper.py

```
from __future__ import print_function
import sys
```

```
for line in sys.stdin:
    article_id, content = line.split("\t", 1)
    words = content.split()
    for word in words:
        print(word, 1, sep="\t")
```

**output:** (word, 1), (word, 1), (a, 1), ...

## Shuffle & Sort



Combiner가 같은 key의 값을 합칠 수 있다.

**output:** (word, 1), (word, 1), (a, 1)

**output:** (word, 2), (a, 1), ...

**input:** word word a word b c d word d e...

Mapper (Python): reporter\_mapper.py    Mapper에서 Combiner역할까지 구현한 예시

```
from __future__ import print_function  
from collections import Counter  
import sys
```

```
for line in sys.stdin:  
    article_id, content = line.split("\t", 1)  
    words = content.split()  
    counts = Counter(words)  
    for word, word_count in counts.items():  
        print(word, word_count, sep="\t")
```

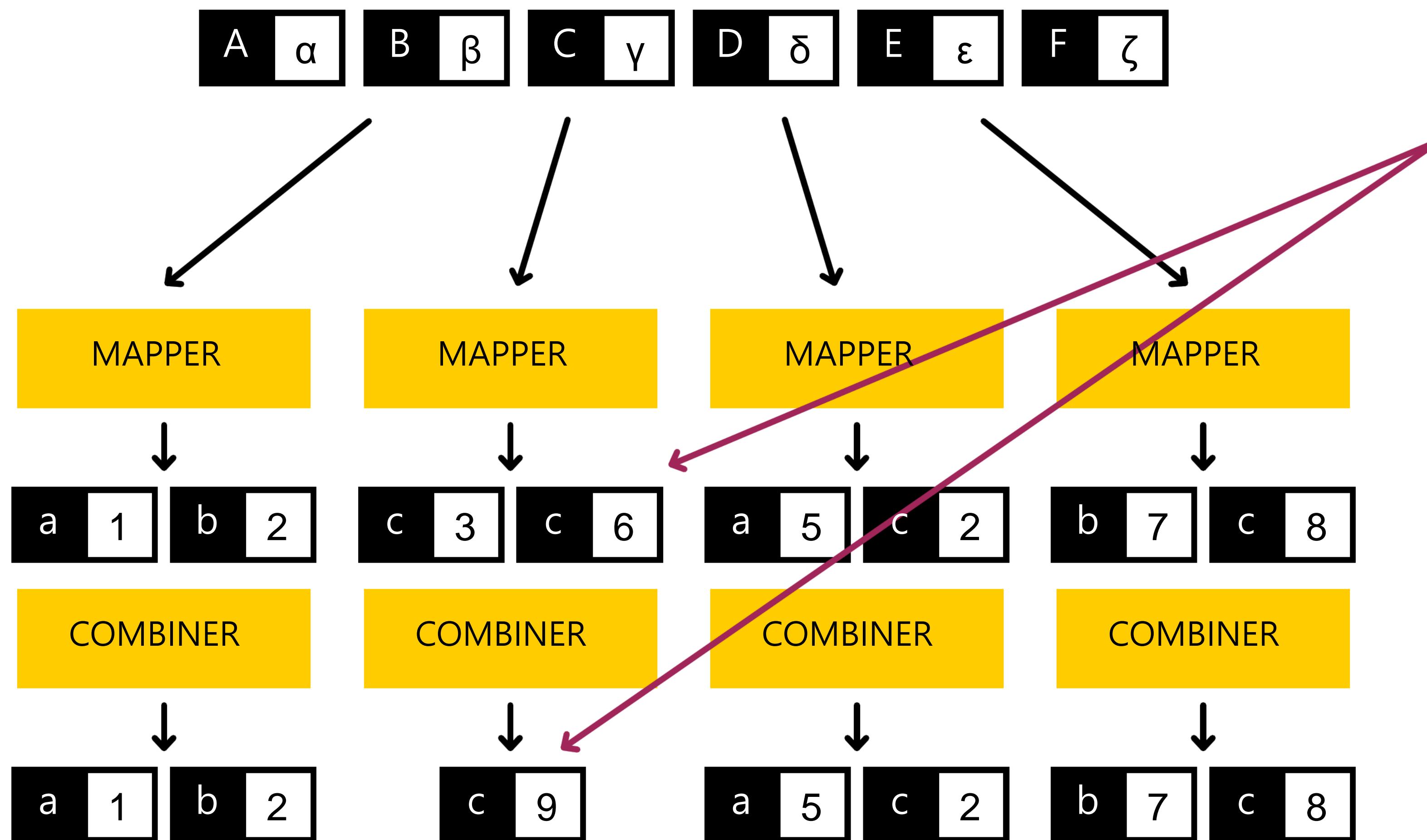


Python의 Counter collections를 활용  
각 단어마다 합계를 자동으로 계산

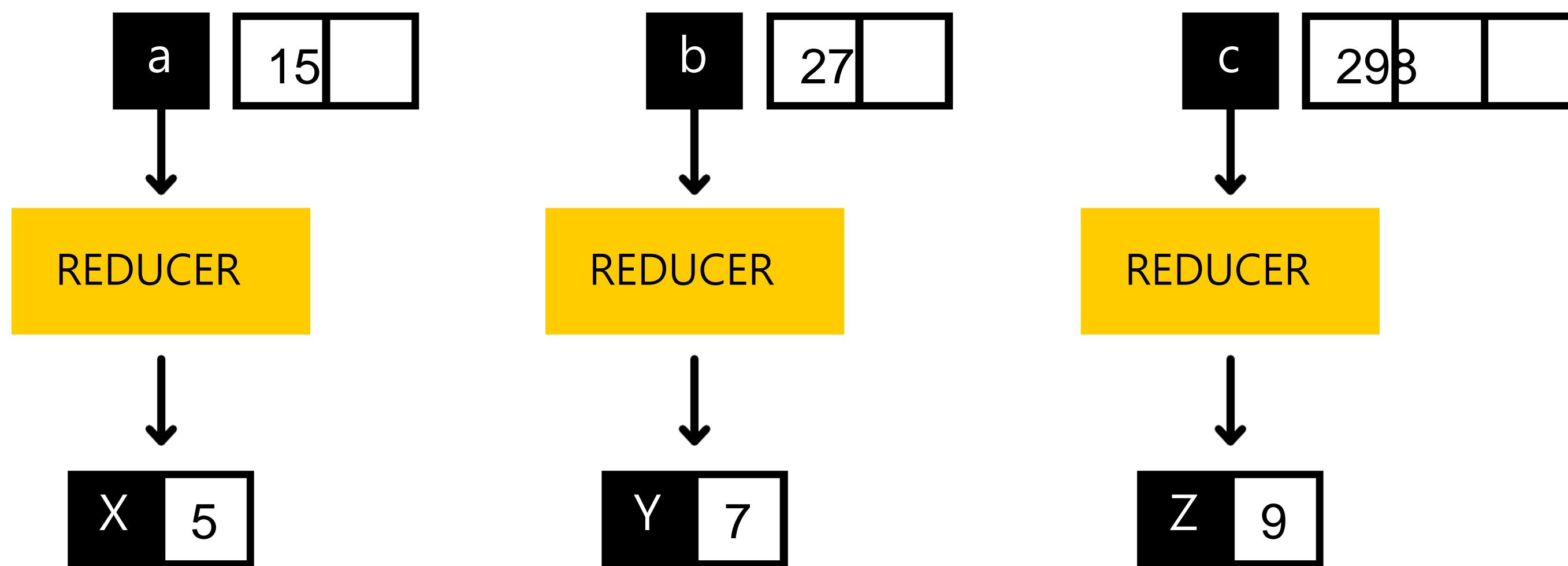
**output:** (a, 1), (b, 1), (c, 1), (d, 2), (e, 1), (word, 4), ...

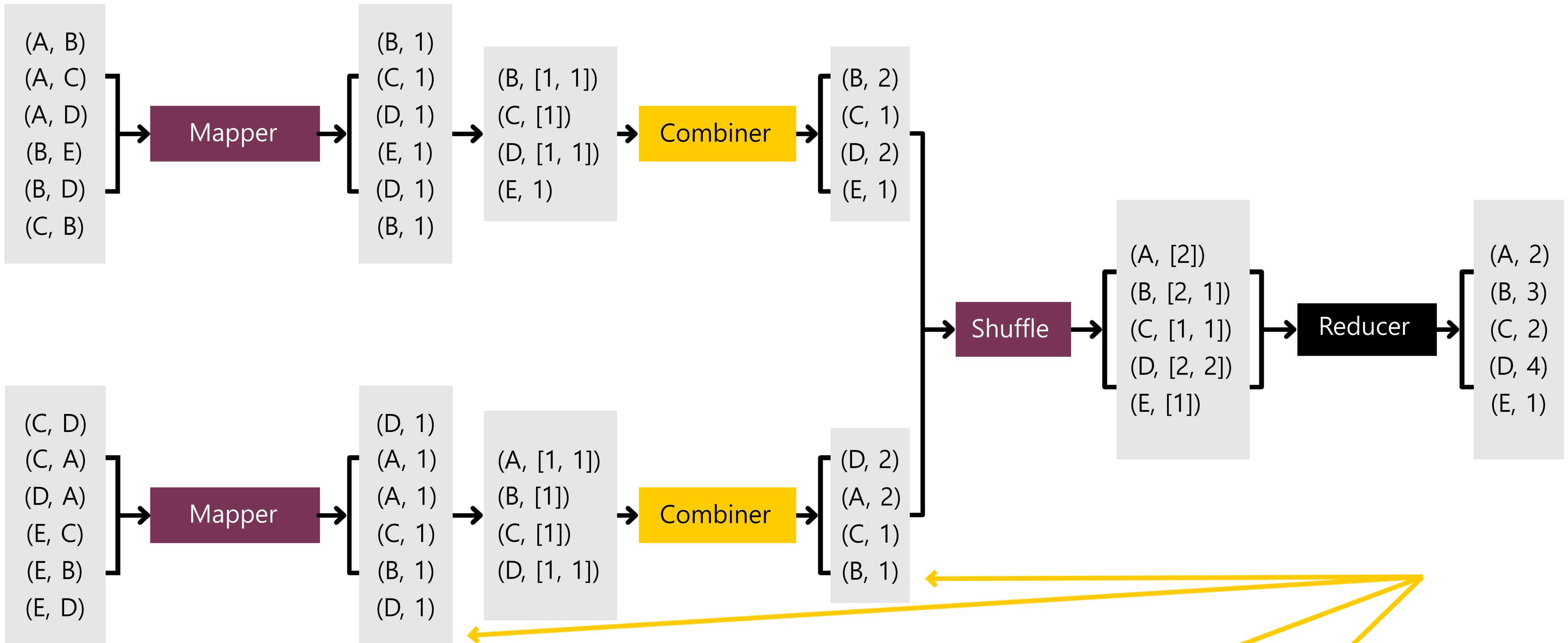
Combiner는 I/O operation을 줄이고 네트워크량을 줄여 전체적인 시간 감소가 가능

|                               | without Combiner | with Combiner |
|-------------------------------|------------------|---------------|
| Wall time (sec)               | <b>935</b>       | <b>528</b>    |
| CPU time (sec)                | <b>9790</b>      | <b>6584</b>   |
| Local FS Read (MB)            | <b>3006</b>      | <b>1324</b>   |
| Local FS Write (MB)           | <b>4527</b>      | <b>1963</b>   |
| Peek Map phys. memory (MB)    | <b>526</b>       | <b>606</b>    |
| Peek Map virt. memory (MB)    | <b>2131</b>      | <b>2144</b>   |
| Peek Reduce phys. memory (MB) | <b>2744</b>      | <b>631</b>    |
| Peek Reduce virt. memory (MB) | <b>3196</b>      | <b>3194</b>   |



Shuffle and Sort: aggregate values by keys





- read:  $[(k_{in}, v_{in}), \dots]$
- map:  $(k_{in}, v_{in}) \rightarrow [(k_{interm}, v_{interm}), \dots]$
- combiner:  $(k_{interm}, [(v_{interm}, \dots)]) \rightarrow [(k_{interm}, v_{interm}), \dots]$
- Shuffle & Sort: sort and group by  $k_{interm}$
- reduce:  $(k_{interm}, [(v_{interm}, \dots)]) \rightarrow [(k_{out}, v_{out}), \dots]$

```
yarn jar $HADOOP_STREAMING_JAR \
    -files mapper.py,reducer.py \
    -mapper 'python mapper.py' \
    -reducer 'python reducer.py' \
    -combiner 'python reducer.py' \
    -numReduceTasks 2 \
    -input gribodov.txt \
    -output word_count
```

위 예제는 combiner와 reducer의 역할이 같다.  
( 같은 key값의 value들을 합치는 것 )

## Map-Reduce Framework

```
Map input records=2681
Map output records=182
Map output bytes=1218
Map output materialized bytes=955
Input split bytes=126
> Combine input records=182
Combine output records=99
Reduce input groups=99
Reduce shuffle bytes=955
Reduce input records=99
Reduce output records=99
```



WIKIPEDIA  
The Free Encyclopedia

<article id> <tab> <article content>  
key value

**input:** word word a word b c d word d e...

mean

각 단어의 평균 사용 개수를  
구하는 문제



**output:** (word, 3.5), (a, 0.5), ...

**input:** word word a word b c d word d e...

Mapper (Python): mapper.py

```
from __future__ import print_function
from collections import Counter
import sys

for line in sys.stdin:
    article_id, content = line.split("\t", 1)
    words = content.split()
    counts = Counter(words)
    for word, word_count in counts.items():
        print(word, word_count, sep="\t")
        print(word, 1, word_count, sep="\t")
```

Mapper는 article을 count하기 위한 변수도 출력해야 한다.

~~output:~~ (a, 1), (d, 2), (word, 4), ...

**output:** (a, (1, 1)), (d, (1, 2)), (word, (1, 4)), ...

## Mapper (Python): combiner.py

```
from __future__ import print_function
import sys

current_word = None
word_count, article_count = 0, 0

for line in sys.stdin:
    word, articles, counts = line.split("\t", 2)
    articles, counts = int(articles), int(counts)
    if word == current_word:
        word_count += counts
        article_count += articles
    else:
        if current_word:
            assert len(current_word.rstrip()) > 0
            print(current_word, word_count, article_count, sep="\t")
        current_word = word
        word_count = counts
        article_count = articles

if current_word:
    print(current_word, word_count, article_count, sep="\t")
```

Combiner는 article의 합계와 단어의 합계를 구한다.

(a, (1, 1))  
(a, (1, 3))  
(a, (1, 5))  
(d, (1, 2))  
(d, (1, 3))  
(d, (1, 3))  
...  
...



(a, (3, 9))  
(d, (3, 8))  
...

## Mapper (Python): reducer.py

```
from __future__ import print_function
import sys

current_word = None
word_count, article_count = 0, 0

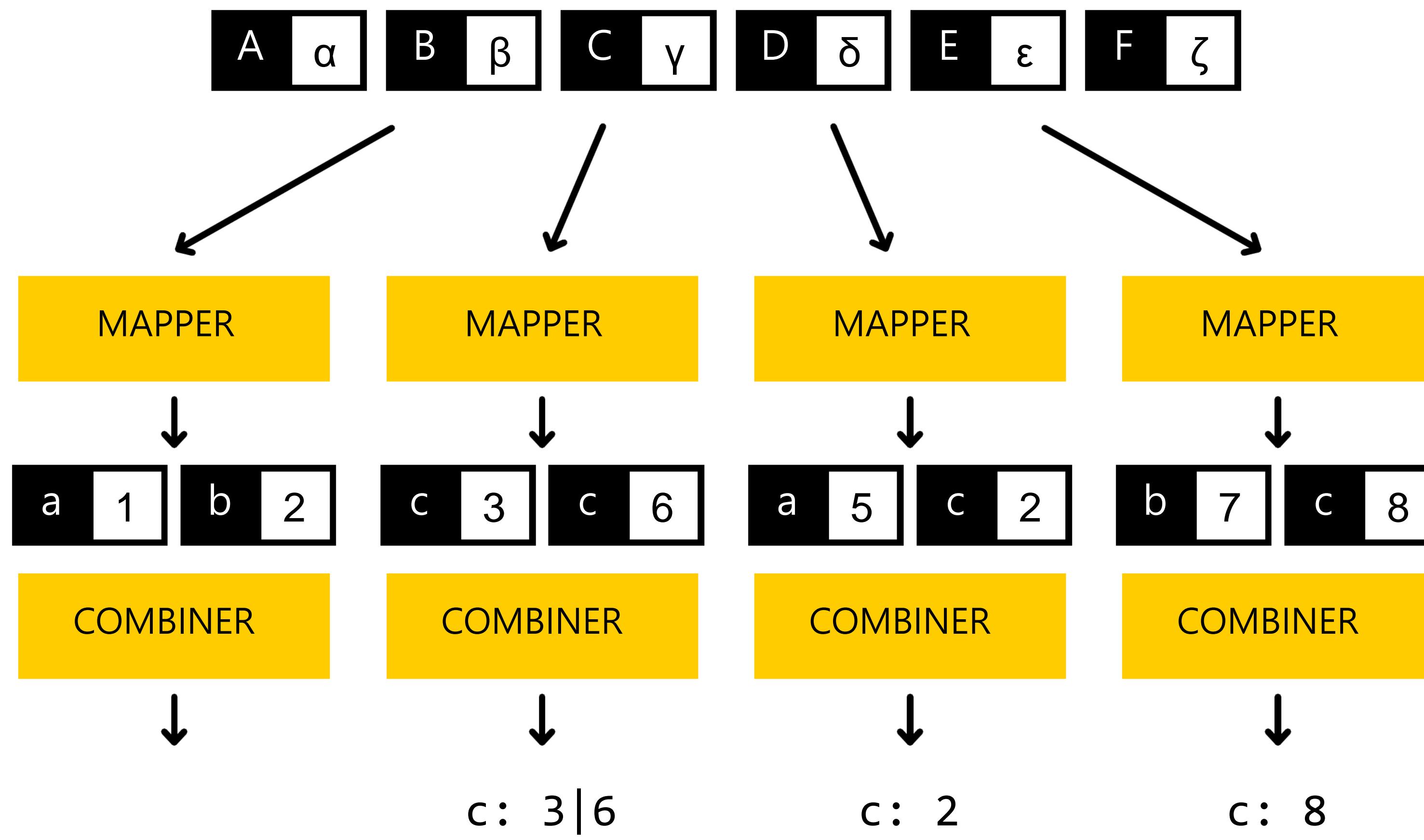
for line in sys.stdin:
    word, articles, counts = line.split("\t", 2)
    articles, counts = int(articles), int(counts)

    if word == current_word:
        word_count += counts
        article_count += articles
    else:
        if current_word:
            assert len(current_word.rstrip()) > 0
            print(current_word, word_count, article_count, sep="\t")
        current_word = word
        word_count = counts
        article_count = articles

if current_word:
    print(current_word, word_count, article_count, sep="\t")
```

Reducer는 Combiner와 코드가 같다.

## MEDIAN



c: 4.75

# Summary

- > Based on Mapper and Reducer you **can derive** MapReducer Combiner signature
- > You know **how to call** MapReduce Combiner in streaming applications
- > You know **how to write** MapReduce Combiner streaming scripts
- > You **can identify situations** where you do not need to use Combiner

# MapReduce

## Partitioner

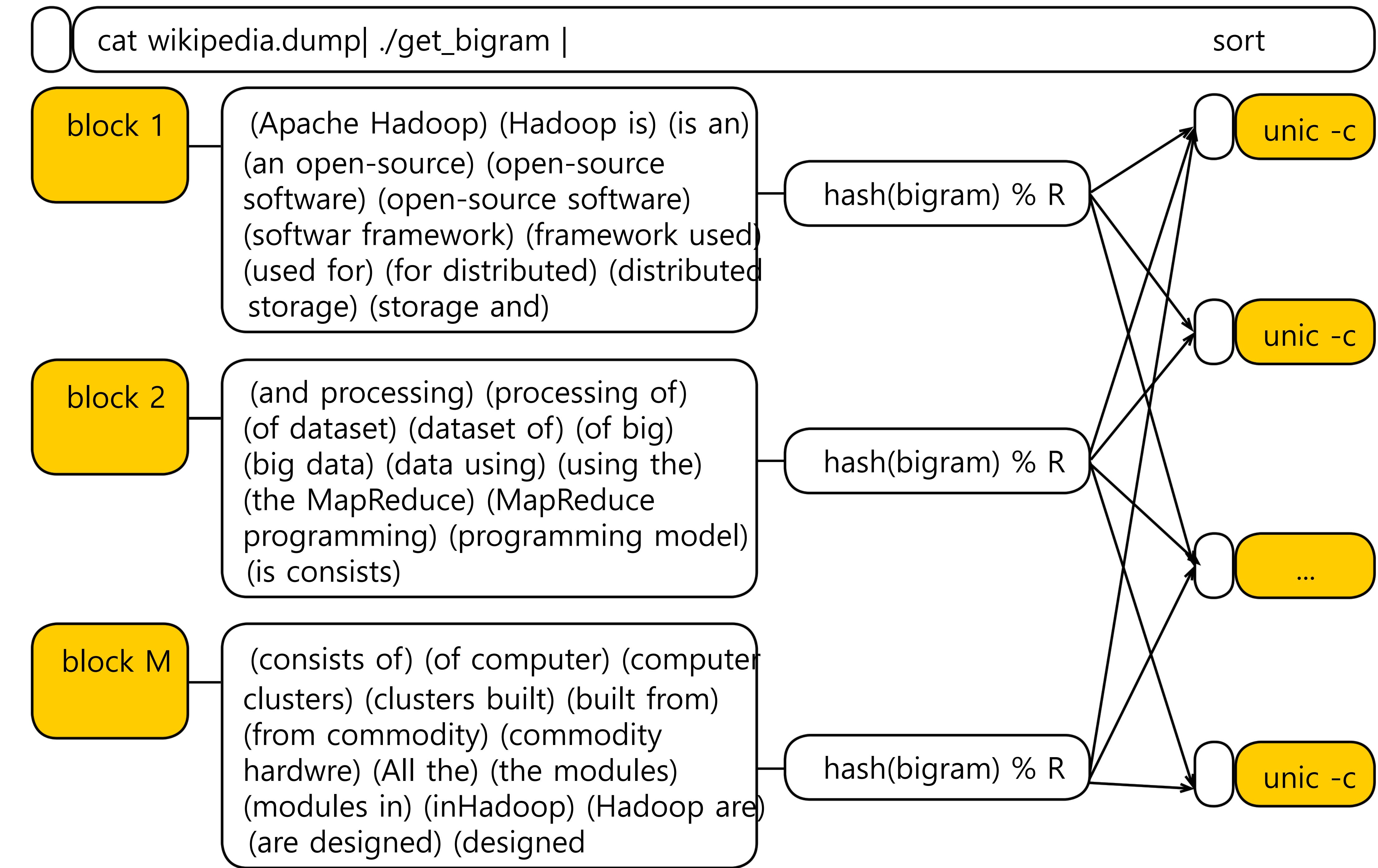
# Collocations

단어들의 결합이 의미를 갖는 것 => 하나의 단어로 인식해야 한다.

| natural English | unnatural English          | New and York      |
|-----------------|----------------------------|-------------------|
| the fast train  | the <del>quick</del> train |                   |
| fast food       | quick food                 |                   |
| a quick shower  | a fast shower              |                   |
| a quick meal    | a fast meal                | United and States |

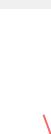
# WordCount

bigram : 두 개의 단어가 하나의 의미를 갖는 것



## input

```
$ head -c 100 wikipedia_sample.txt  
12 Anarchism Anarchism is often defined as a political  
philosophy which holds the state to ...
```



## Mapper (Python): bigram\_mapper.py

```
from __future__ import print_function  
import re  
import sys  
for line in sys.stdin:  
    article_id, content = line.split("₩t", 1)  
    words = re.split("₩₩+", content)  
    for index in range(len(words) - 1):  
        print(words[index], words[index + 1], 1, sep="₩t")
```

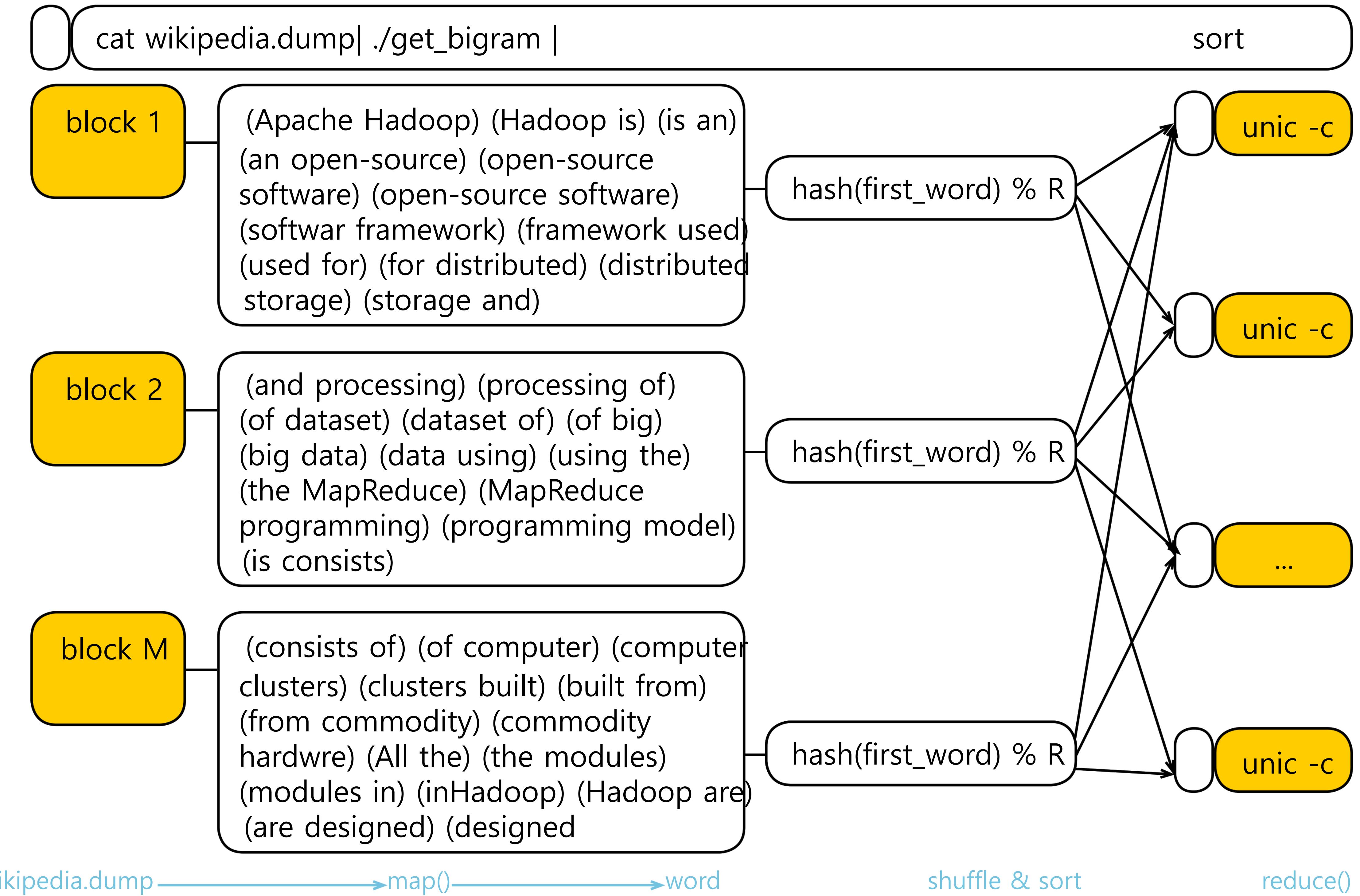


## output

```
$ head word_count/part-00000  
Anarchism Anarchism 1  
Anarchism is 1  
is often 1
```

문제점 : 첫번째 단어만 Key로 인식하기 때문에 두번째 단어가 제대로 정렬되는지 보장할 수 없다.

# WordCount



## Mapper (Python): inmemroy\_bigram\_reducer.py

```
from __future__ import print_function
from collections import Counter
import sys

current_word = None
bigram_count = Counter()

for line in sys.stdin:
    first_word, second_word, counts = line.split(" ", 2)
    counts = Counter({second_word: int(counts)})
    if first_word == current_word:
        bigram_count += counts
    else:
        if current_word:
            for second_word, bigram_count in bigram_count.items():
                print(current_word, second_word,
```

e.g.)

Anarchism Anarchism 1  
Anarchism is 1  
Anarchism Anarchism 1  
Anarchism test 1

문제점 : 첫번째 단어만 Key로 인식하기 때문에 두번째 단어가 제대로 정렬되는지 보장할 수 없다.

따라서, reducer에서 두번째 단어를 메모리에 저장하면서 값을 업데이트 해야 한다.

```
yarn --config $HADOOP_EMPTY_CONFIG jar $HADOOP_STREAMING_JAR \
  -files bigram_mapper.py,inmemory_bigram_reducer.py \
  -mapper 'python bigram_mapper.py' \
  -reducer 'python inmemory_bigram_reducer.py' \
  -numReduceTasks 5 \
  -input wikipedia_sample.txt \
  -output word_count \
```

```
$ grep $'^new\\t' word_count/* | sort -nrk3,3 | head -4
word_count/part-00001:new york 112
word_count/part-00001:new world 15
word_count/part-00001:new jewrsey 12
word_count/part-00001:new constitution 12
$ grep '$\\tyork\\t' word_count/* | sort -nrk3,3 | head
word_count/part-00001:new york 112
word_count/part-00004:sergeant york 1
```

key 정렬 문제 해결방법 :

두 개의 단어를 Shuffle&Sort를 위한 Key로 인식하도록 config를 설정할 수 있다.

=> Combiner가 메모리 사용없이 연산을 제대로 수행할 수 있도록 할 수 있다.

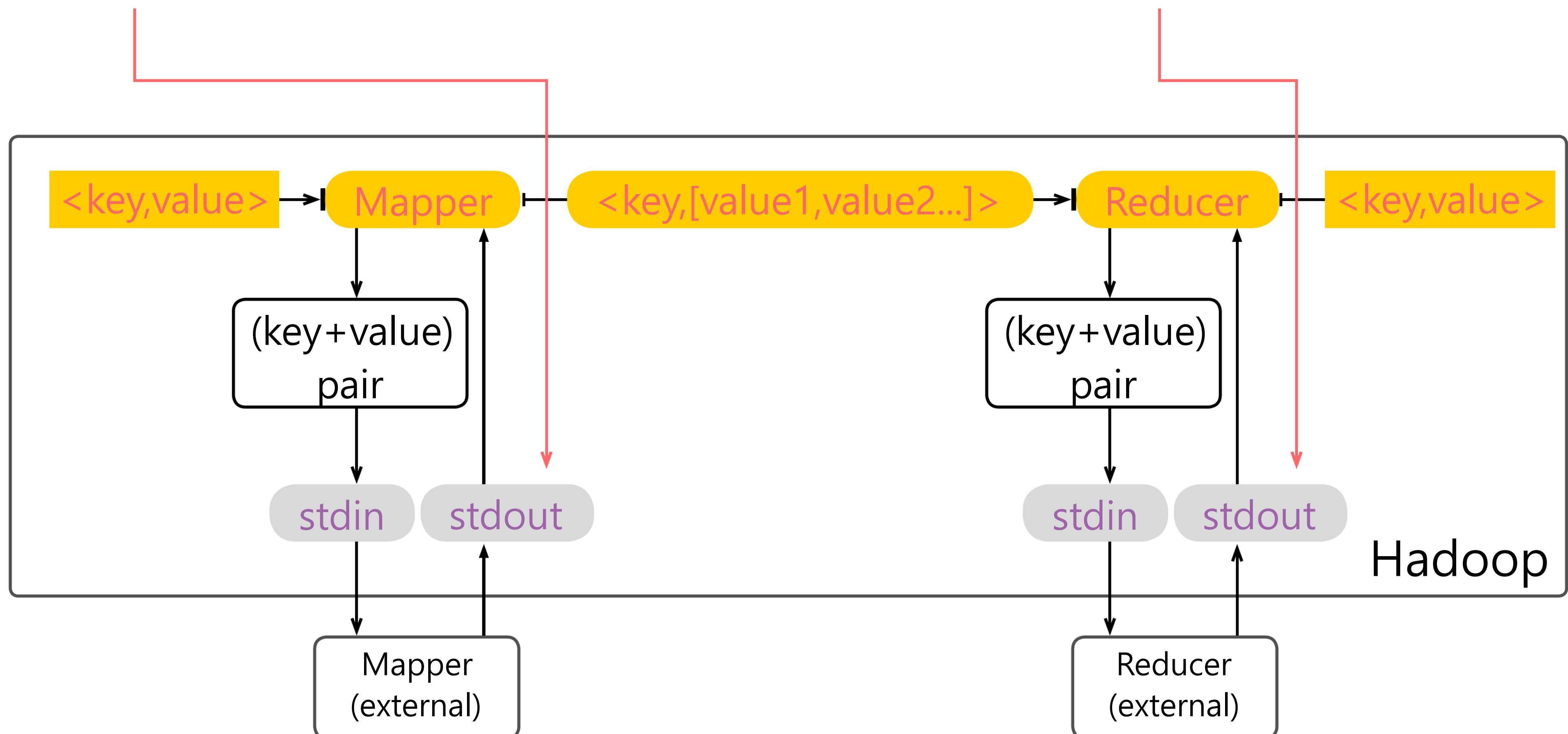
```
yarn --config $HADOOP_EMPTY_CONFIG jar $HADOOP_STREAMING_JAR \
  -D stream.num.map.output.key.fields=2 \
  -files bigram_mapper.py,bigram_reducer.py \
  -mapper 'python bigram_mapper.py' \
  -reducer 'python bigram_reducer.py' \
  -numReduceTasks 5 \
  -input wikipedia_sample.txt \
  -output word_count \
```

word\_count/part-00001:new york 112  
word\_count/part-00001:new world 15  
word\_count/part-00002:new constitution 12  
word\_count/part-00000:new jersey 12

separator는 input data에서 key, value 구분 문자를 의미한다.

- D stream.map.output.field.separator=.
- D stream.num. map.output.key.fields=1

- D stream.reduce.output.field.separator=.
- D stream.num. reduce.output.key.fields=2



IPv4 address (example): 69.89.31.226

## input

```
$ cat subnet.txt  
1.2.3.4  
2.3.4.5  
3.4.5.6  
4.5.6.7
```

```
yarn --config $HADOOP_EMPTY_CONFIG jar $HADOOP_STREAMING_JAR \  
-D stream.map.output.field.separator=. \  
-D stream.num.map.output.key.fields=1 \  
-D stream.reduce.output.field.separator=. \  
-D stream.num.reduce.output.key.fields=2 \  
-files identity_mr.py \  
-mapper 'python identity_mr.py' \  
-reducer 'python identity_mr.py' \  
-numReduceTasks 2 \  
-input subnet.txt \  
-output subnet_out
```

## output

```
cat subnet_out/*  
2 3.4 5  
4 5.6 7  
1 2.3 4  
3 4.5 6
```

## input

```
$ cat subnet.txt  
1.2.3.4  
2.3.4.5  
3.4.5.6  
4.5.6.7
```

```
yarn --config $HADOOP_EMPTY_CONFIG jar $HADOOP_STREAMING_JAR \  
-D stream.map.output.field.separator=. \  
-D stream.num.map.output.key.fields=1 \  
-----  
-D stream.reduce.output.field.separator=. \  
-D stream.num.reduce.output.key.fields=2 \  
-----  
-files identity_mr.py \  
-mapper 'python identity_mr.py' \  
-reducer 'python identity_mr.py' \  
-numReduceTasks 2 \  
-input subnet.txt \  
-output subnet_out
```

## output

```
cat subnet_out/*  
2 3.4 5  
4 5.6 7  
1 2.3 4  
3 4.5 6
```

partition.keypartitioner : Partition을 나누는 기준 설정

-k : key값이라는 의미

1, 2 : 첫번째 단어에서 두번째 단어까지

man sort | grep KEYDEF 방식과 같음

e.g.) 1.2, 1.2 : 첫번째 단어에서 2번째 글자를 정렬key로 선택한다

```
yarn --config $HADOOP_EMPTY_CONFIG jar $HADOOP_STREAMING_JAR \
-D map.output.key.field.separator=. \
-D mapreduce.partition.keypartitioner.options=-k1.2,1.2 \
-files identity_mr.py \
-mapper 'python identity_mr.py' \
-reducer 'python identity_mr.py' \
-numReduceTasks 2 \
-input subnet.txt \
-output subnet_out \
-partitioner org.apache.hadoop.mapred.lib.KeyFieldBasedPartitioner
```

| input             | output            |
|-------------------|-------------------|
| \$ cat subnet.txt | cat subnet_out/*  |
| 1a.2.3.4          | 3 <b>b</b> .4.5.6 |
| 2a.3.4.5          | 4 <b>b</b> .5.6.7 |
|                   | 1 <b>a</b> .2.3.4 |
|                   | 2 <b>a</b> .3.4.5 |

참고 : Partition의 개수 = reducer의 개수(자동)

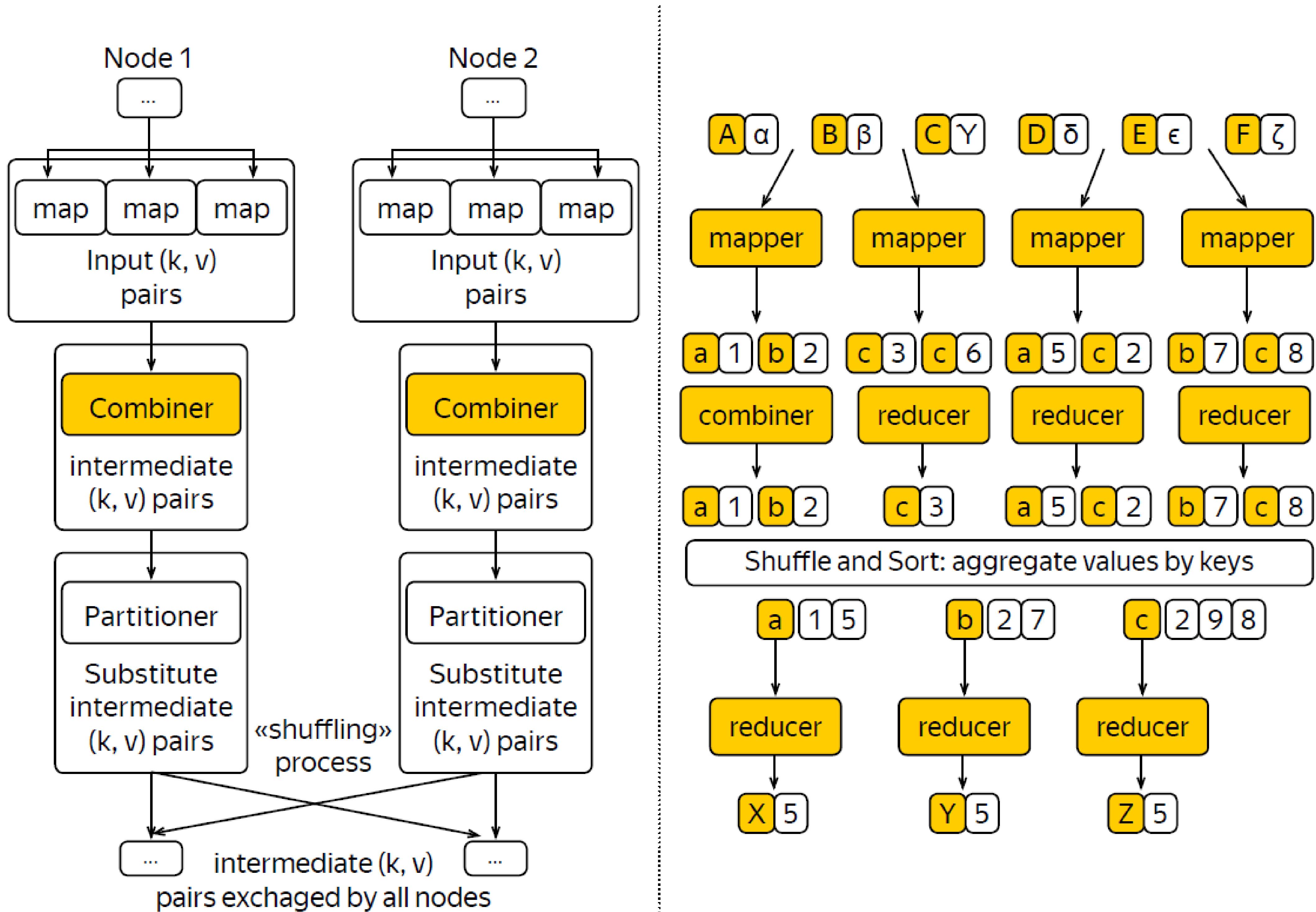
mapper에서는 각 reducer로 보낼 데이터를 partition으로 구분한다

각 Partition에서 Comparator 기준으로 다시 정렬된다.

0 Apr 1 14:59 \_SUCCESS

20 Apr 1 14:59 part-00001

20 Apr 1 14:59 part-00000



# Summary

- » You know **how to use** Partitioner in MapReduce application
- » You know **how to count** bigrams with Hadoop MapReduce and **distribute** load over reducers

# MapReduce

## Comparator

Comparator는 partition 내에서 정렬 Key를 정의하는 것

이 과정을 Secondary Sort라고도 한다.

## input

1.4.3.4  
2.4.4.5  
3.2.5.6  
4.1.6.7  
5.12.6.7

다중 정렬도 가능

r : reverse

```
yarn --config $HADOOP_EMPTY_CONFIG jar $HADOOP_STREAMING_JAR \
-D mapreduce.job.output.key.comparator.class=org.apache.hadoop.mapreduce.lib.partition.KeyFieldBasedComparator \
-D mapreduce.map.output.key.field.separator= \
-D mapreduce.partition.keycomparator.options="-k2,2 -k3,3r" \
-files identity_mr.py \
-mapper 'python identity_mr.py' \
-reducer 'python identity_mr.py' \
-numReduceTasks 1 \
-input subnet.txt \
-output subnet_out
```

## output

4.1.6.7  
5.12.6.7  
3.2.5.6  
2.4.4.5  
1.4.3.4

TEXT값으로 정렬된 상태

1 12 2 4 4

6 6 5 4 3

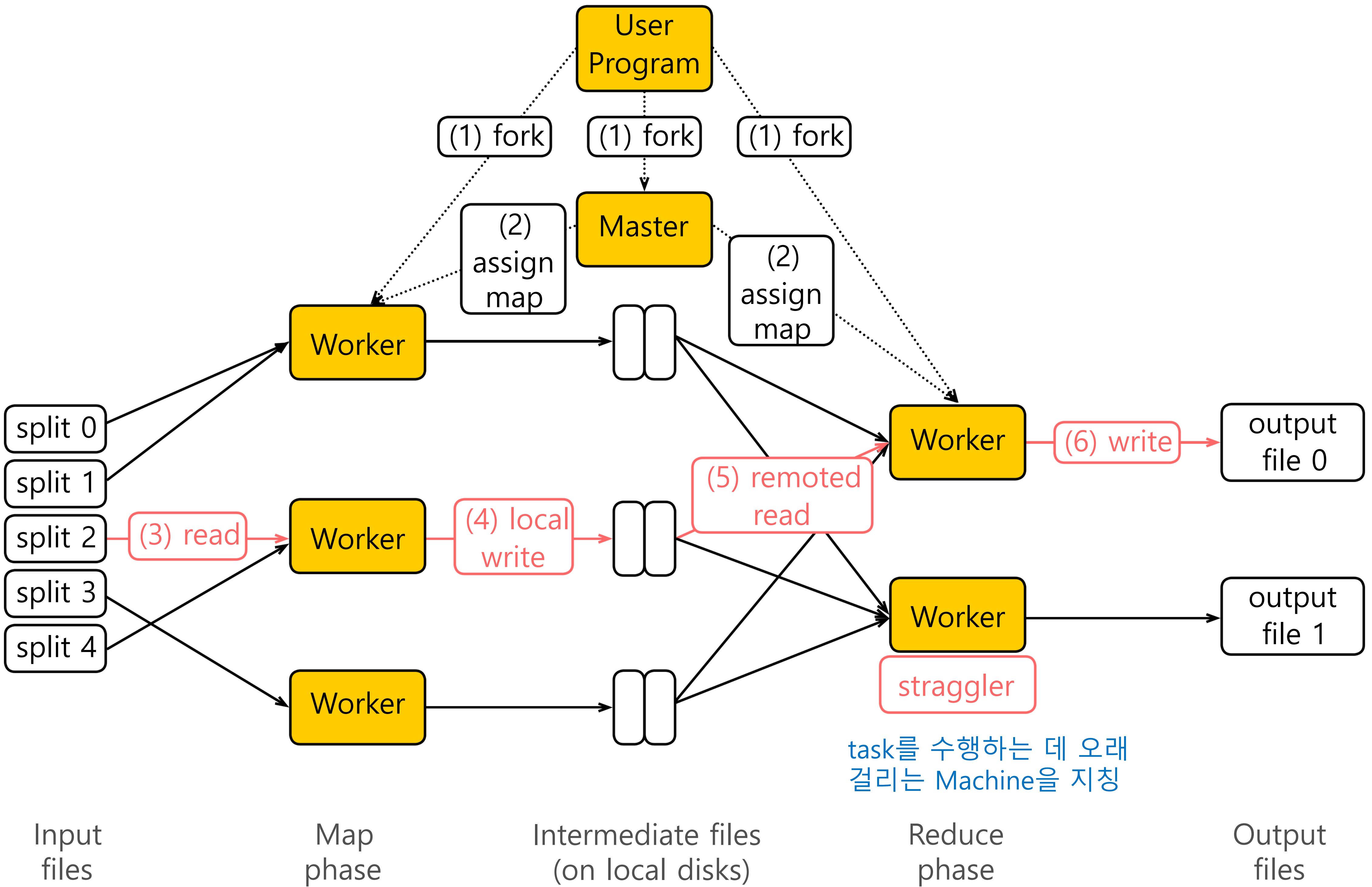
# MapReduce

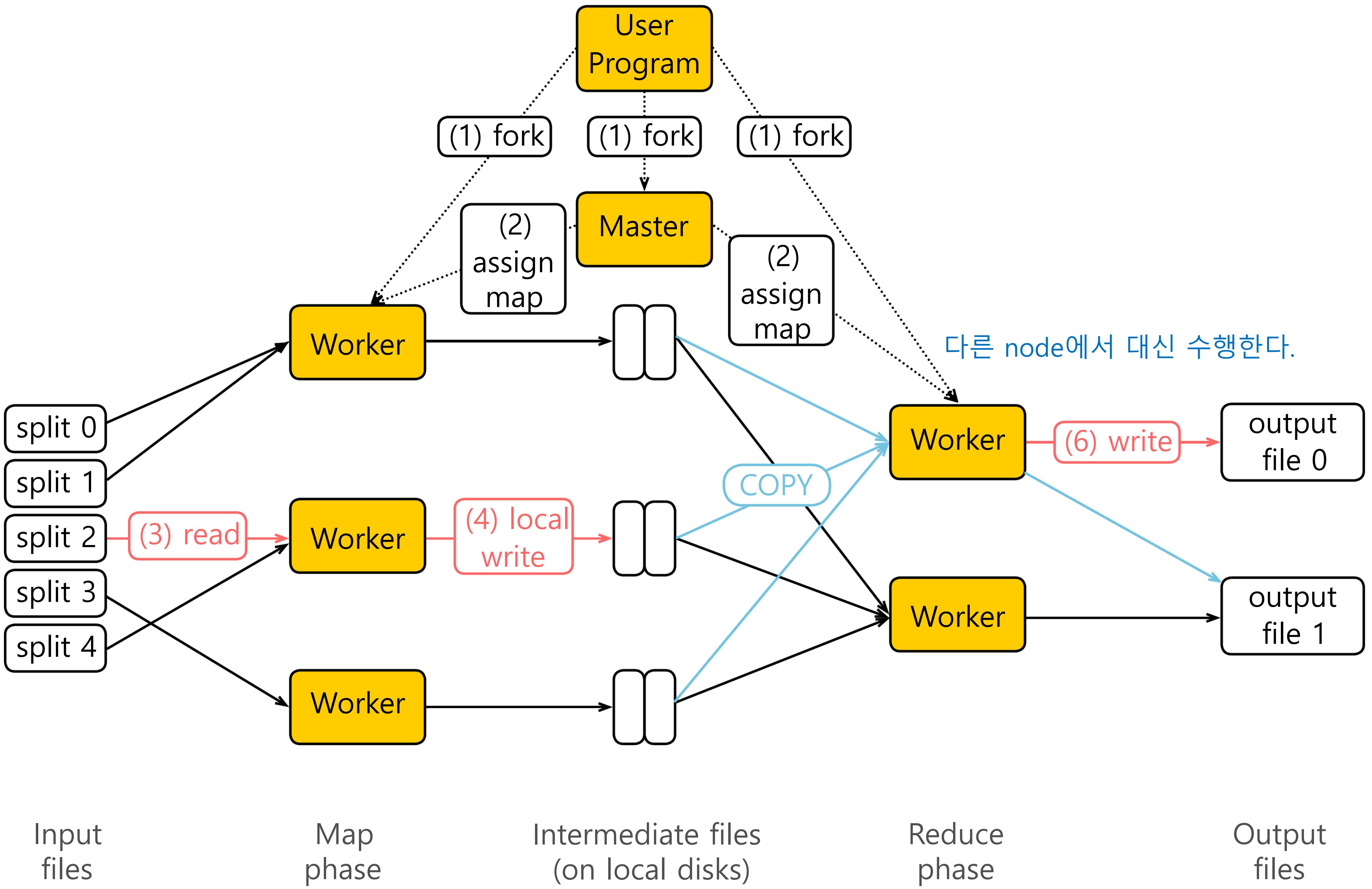
## Speculative Execution / Backup Tasks

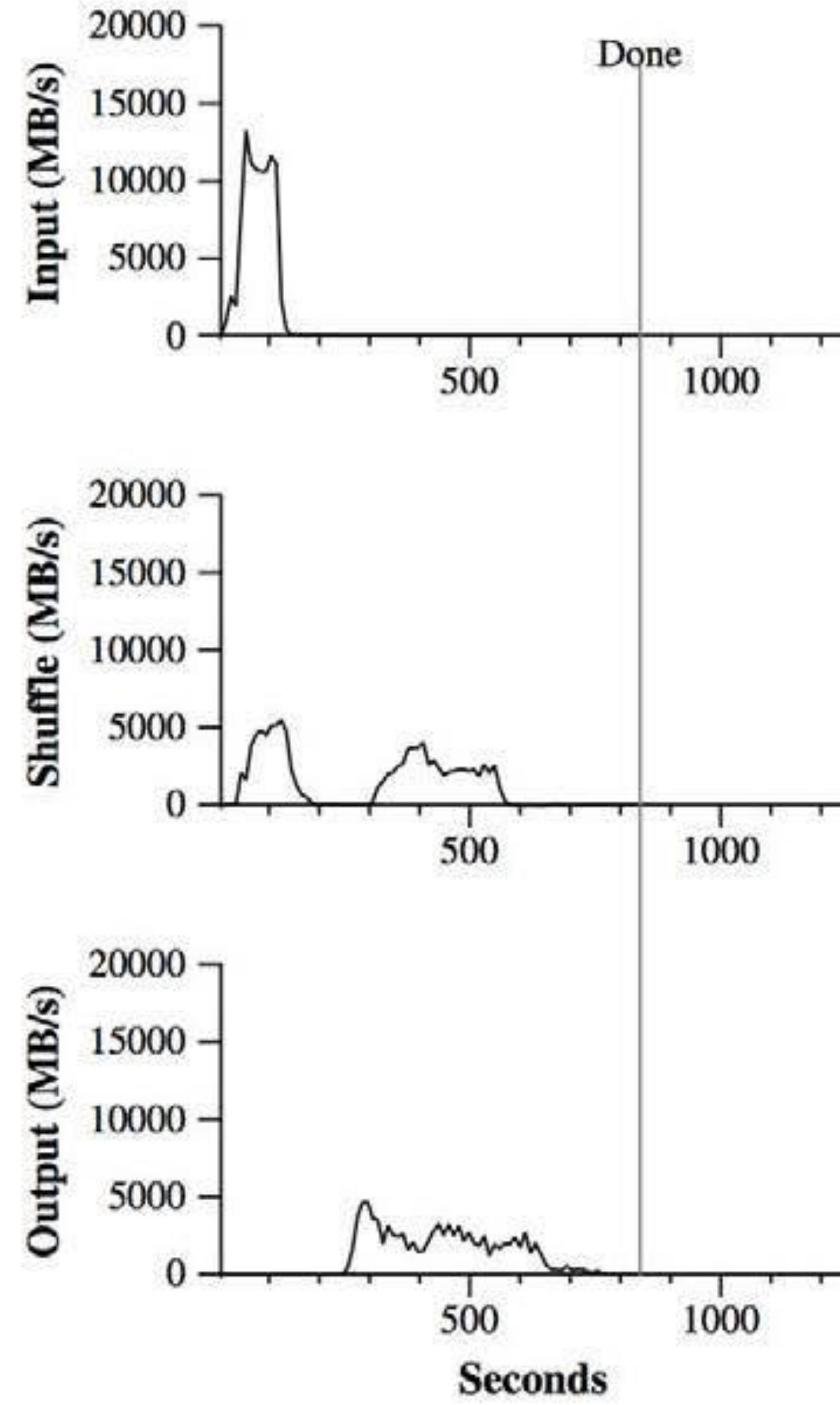
추측에 근거한 이라는 뜻

Task가 너무 오래 걸린다고 판단하면 다른 node에서 task를 대신 수행하도록 하는 것

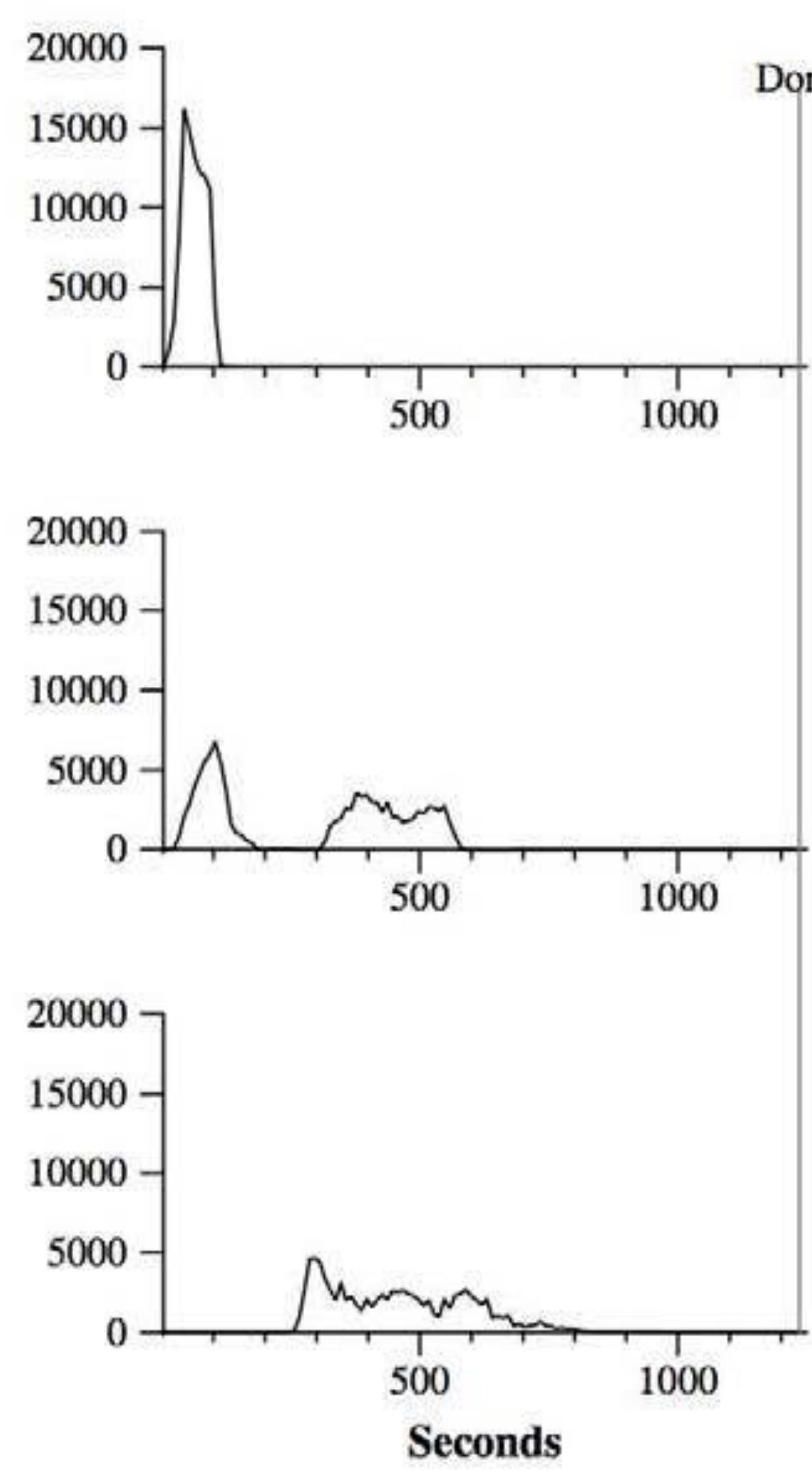
MapReduce Application이 끝나갈 때쯤 speculation을 실행



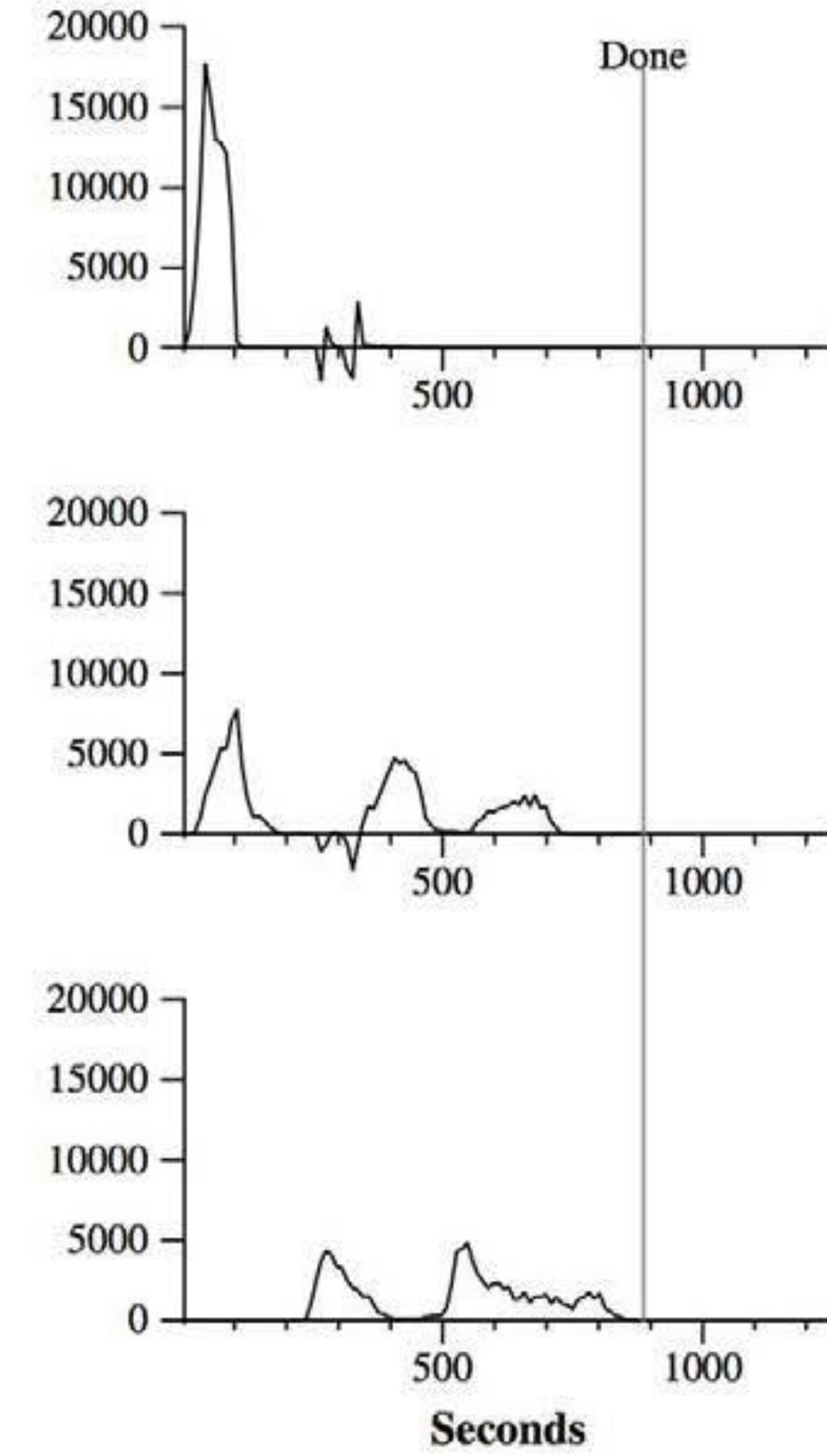




(a) Normal execution



(b) No backup tasks



(c) 200 tasks killed

Figure 3: Data transfer rates over time for different executions of the sort program

"MapReduce: Simplified Data Processing on Large Clusters"  
by Jeffrey Dean and Sanjay Ghemawat

```
yarn jar $MR_STREAMING_JAR \
-D mapreduce.map.speculative="true" \
-D mapreduce.job.speculative.speculative-cap-running-tasks=0.99 \
-D mapreduce.job.speculative.speculative-cap-total-tasks=0.99 \
-D mapreduce.job.speculative.retry-after-no-speculate=10 \
-files flaky_mapper.py \
-mapper 'python flaky_mapper.py' \
-output flaky_mr \
-numReduceTasks 0 \
-input /data/wiki/en_articles_part
```

```
>> mapreduce.map.speculative (default:"true")  
  
>> mapreduce.reduce.speculative (default:"true")  
  
>> mapreduce.job.speculative.speculative-cap-running-tasks  
  >>(default: 0.1) 퍼센트  
  
>> mapreduce.job.speculative.speculative-cap-total-tasks  
  >>(default: 0.01) 퍼센트  
  
>> mapreduce.job.speculative.retry-after-no-speculate  
  >>(default: 1000, ms)  
  
>> mapreduce.job.speculative.retry-after-speculate  
  >>(default: 15000, ms)
```



## Attempts for task\_1490709912648\_0465\_m\_000001

Logged in as: dr.who

| Show 20 entries                       |           |  |   |      |                               |                               |              |   |  | Search: |
|---------------------------------------|-----------|--|---|------|-------------------------------|-------------------------------|--------------|---|--|---------|
| Attempt                               | State     | Status   | Node                                    | Logs | Start Time                    | Finish Time                   | Elapsed Time | Note  |  |         |
| attempt_1490709912648_0465_m_000001_0 | FAILED    |  | /default/virtual-node3.atp-fvt.org:8042 | logs | Sun Apr 2 17:04:10 +0300 2017 | Sun Apr 2 17:04:15 +0300 2017 | 5sec         | Error: java.lang.RuntimeException: PipeMapRed.waitOutputThreads(): subprocess failed with code 1 at ... |  |         |
| attempt_1490709912648_0465_m_000001_1 | FAILED    |  | /default/virtual-node1.atp-fvt.org:8042 | logs | Sun Apr 2 17:04:17 +0300 2017 | Sun Apr 2 17:04:21 +0300 2017 | 3sec         | Error: java.lang.RuntimeException: PipeMapRed.waitOutputThreads(): subprocess failed with code 1 at ... |  |         |
| attempt_1490709912648_0465_m_000001_2 | KILLED    |  | /default/virtual-node1.atp-fvt.org:8042 | logs | Sun Apr 2 17:04:22 +0300 2017 | Sun Apr 2 17:04:26 +0300 2017 | 4sec         | Speculation: attempt_1490709912648_0465_m_000001_3 succeeded first!                                     |  |         |
| attempt_1490709912648_0465_m_000001_3 | SUCCEEDED | hdfs://virtual-master.atp-fvt.org:8020/data/wiki/en_articles_part/articles-part:0+38430992 | /default/virtual-node1.atp-fvt.org:8042 | logs | Sun Apr 2 17:04:22 +0300 2017 | Sun Apr 2 17:04:26 +0300 2017 | 4sec         |   |  |         |

Showing 1 to 4 of 4 entries      First Previous 1 Next Last

attempt\_1490709912648\_0465\_m\_000001\_2

KILLED

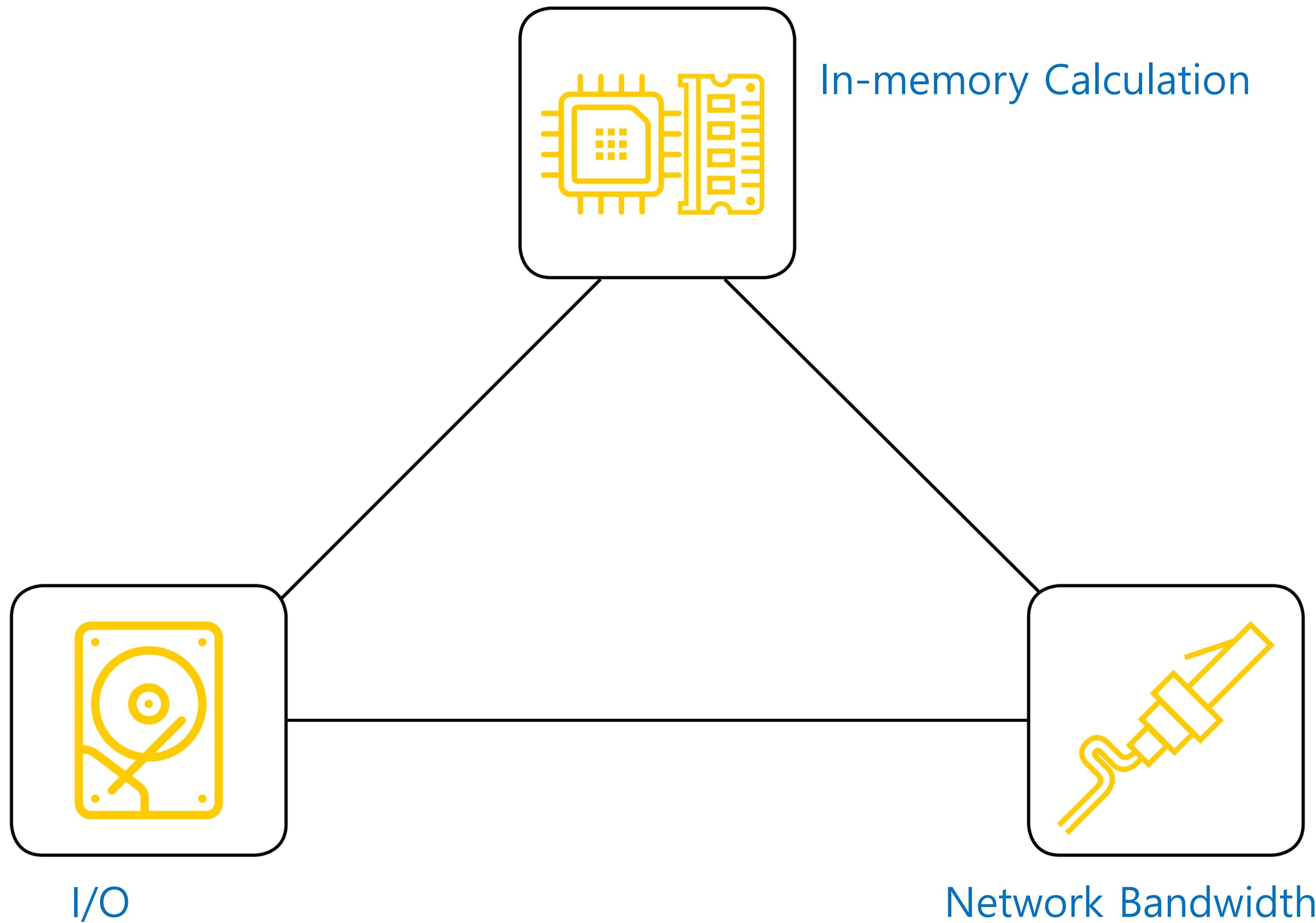
Speculation:  
attempt\_1490709912648\_0465\_m\_000001\_3  
succeeded first!

# MapReduce

## Compression

# Compression Tradeoff

Compression이 미치는 영향



Splittable : 압축파일을 여러 mapper에게 나눠 풀 수 있다. <-> 전체 파일 압축을 하나의 mapper에서 풀어야 한다.  
 (임의의 위치에서 파일을 읽을 수 있음) (처음부터 파일을 읽어야 한다.)

| Compression Format                              | Splittable | Comments   |
|---|------------|--|
| .deflate<br>.gz (gzip)                          | NO         | Uses DEFLATE algorithm                               |
| .bz2 (bzip)                                     | YES        | more effective than gzip, but slower compression     |
| .lzo<br>일반적인 하둡 시나리오에 적합 (Write once read many) | YES*       | decompress faster than gzip, compress less efficient |
| .snappy   | NO         | faster than LZO for decompression                    |

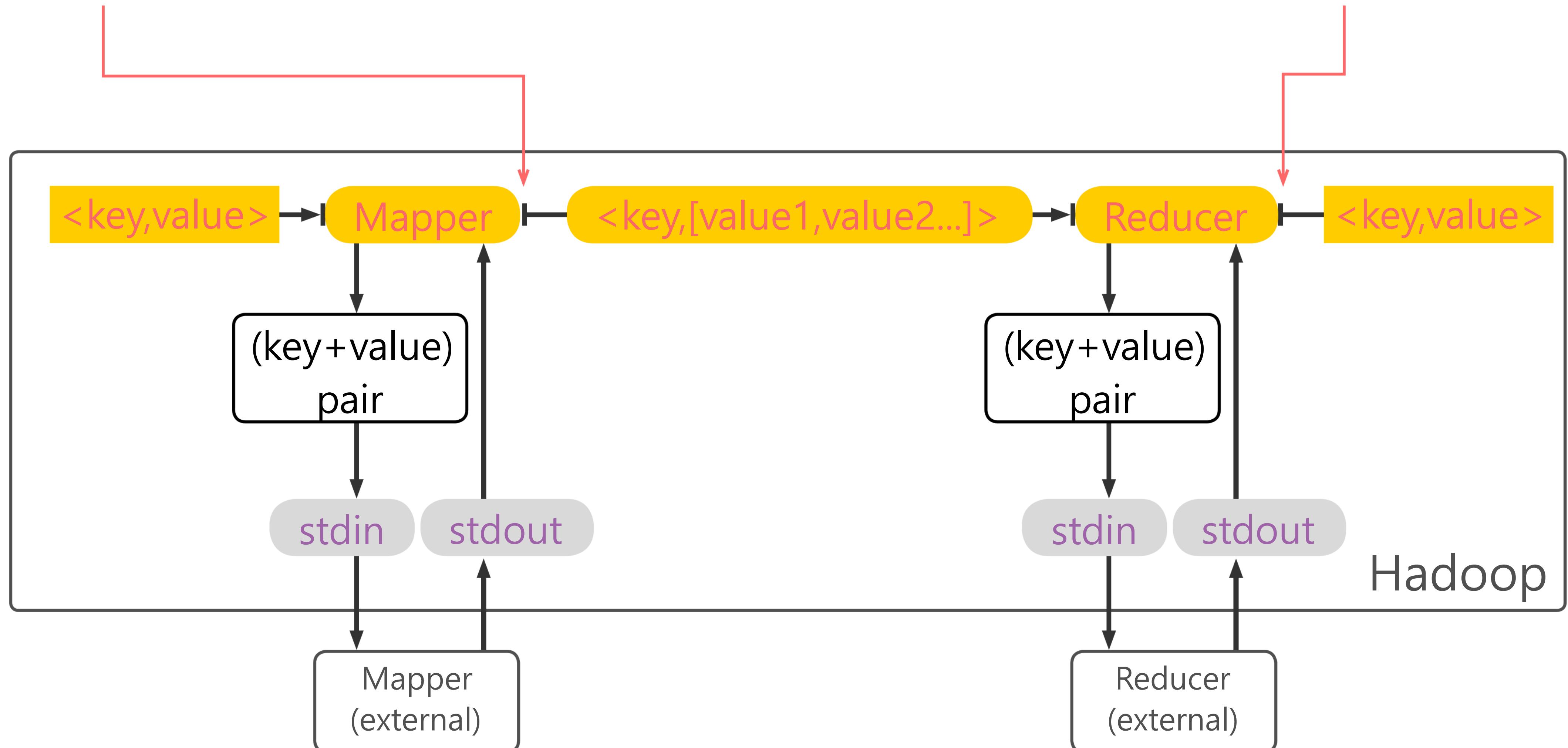
flags: -1(optimised for speed)...-9(optimised for space)

## Kinds of compression

| Compression format | Hadoop CompressionCodec                    |
|--------------------|--|
| DEFLATE            | org.apache.hadoop.io.compress.DefaultCodec |
| gzip               | org.apache.hadoop.io.compress.GzipCodec    |
| bzip2              | org.apache.hadoop.io.compress.BZip2Codec   |
| LZO                | com.hadoop.compression.lzo.LzopCodec       |
| LZ4                | org.apache.hadoop.io.compress.Lz4Codec     |
| Snappy             | org.apache.hadoop.io.compress.SnappyCodec  |

- › Block-level compression
  - › used in SequenceFiles, RCFiles, Parquet
  - › applied within a block of data
- › File-level compression
  - › applied to the file as a whole
  - › hinders an ability to navigate through file

-D mapreduce.compress.map.output=true  
-D mapreduce.map.output.compression.codec=...  
-D mapreduce.output.compress=true  
-D mapreduce.output.compression.codec=...

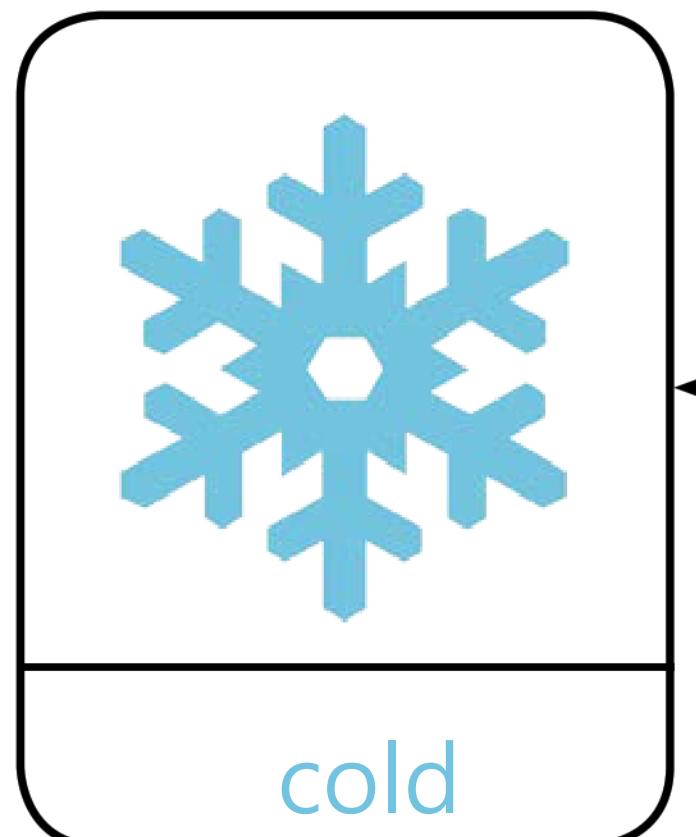


intermediate data 또는 output data 둘 다 적용 가능

# Summary

자주 접근하지 않는 데이터

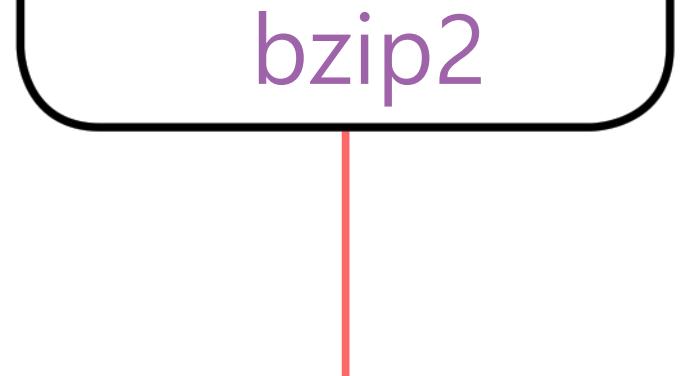
compression이 좋음



cold

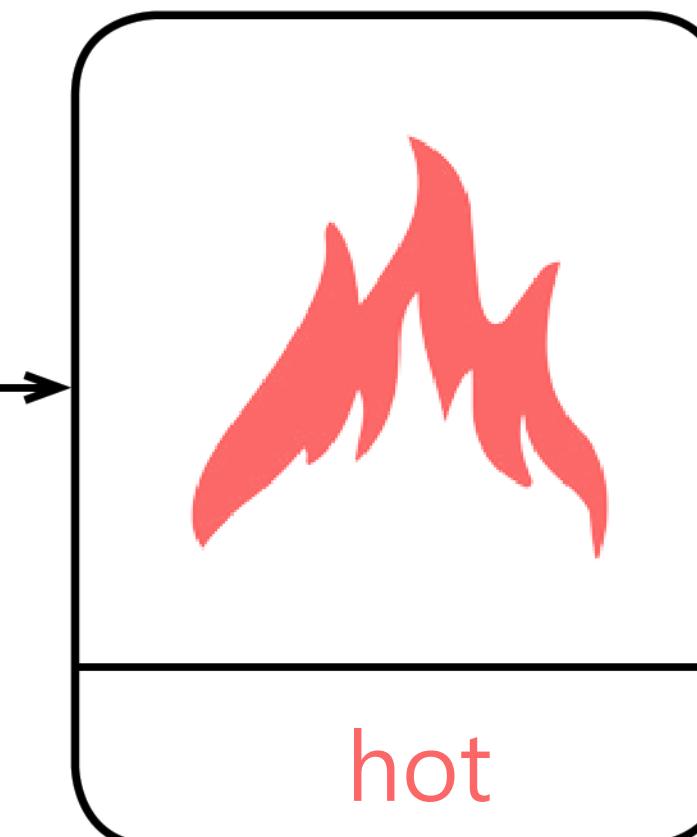


bzip2

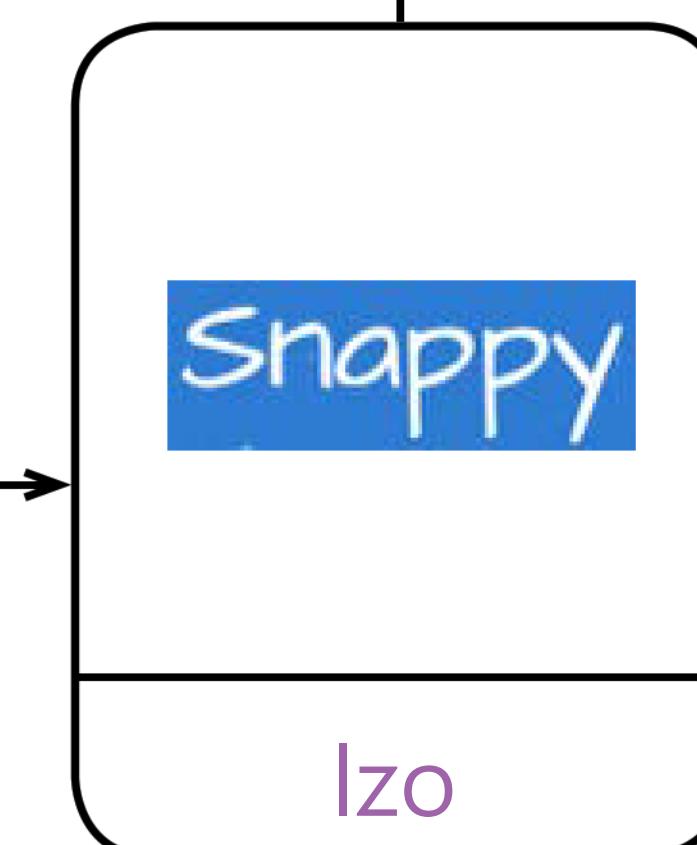


자주 읽는 데이터

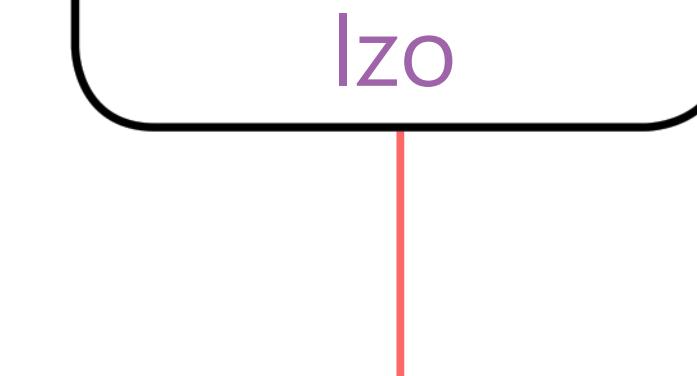
decompression이 빠름



hot



Izo

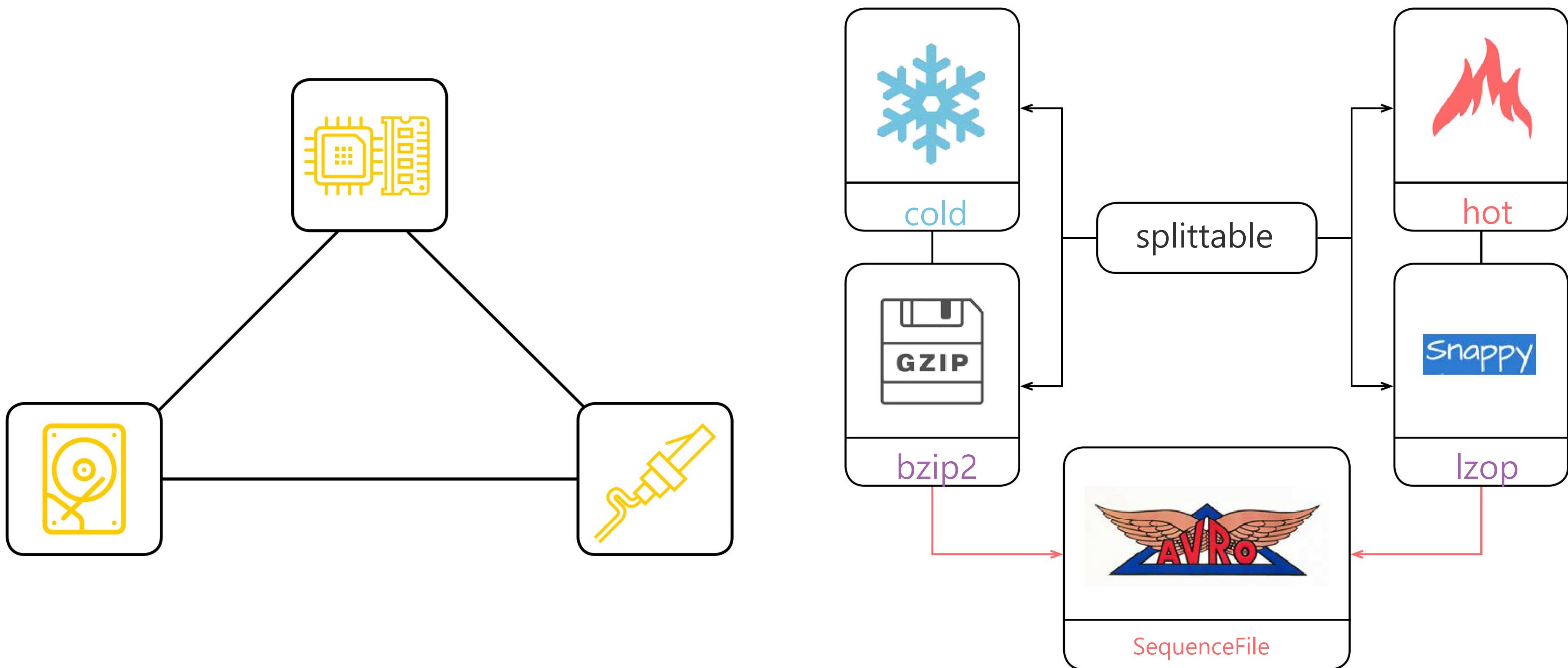


SequenceFile

gzip과 snappy는 file-level compression이 불가능 / block-level compression만 가능하다. (splittable 문제 )

# Summary

- » you know where and how to use compression to
- » optimise MapReduce Application



# Summary

- » you know how to use speculative execution to speed-up MapReduce application
- » see more configuration options in mapred-default.xml

**BigDATAteam**