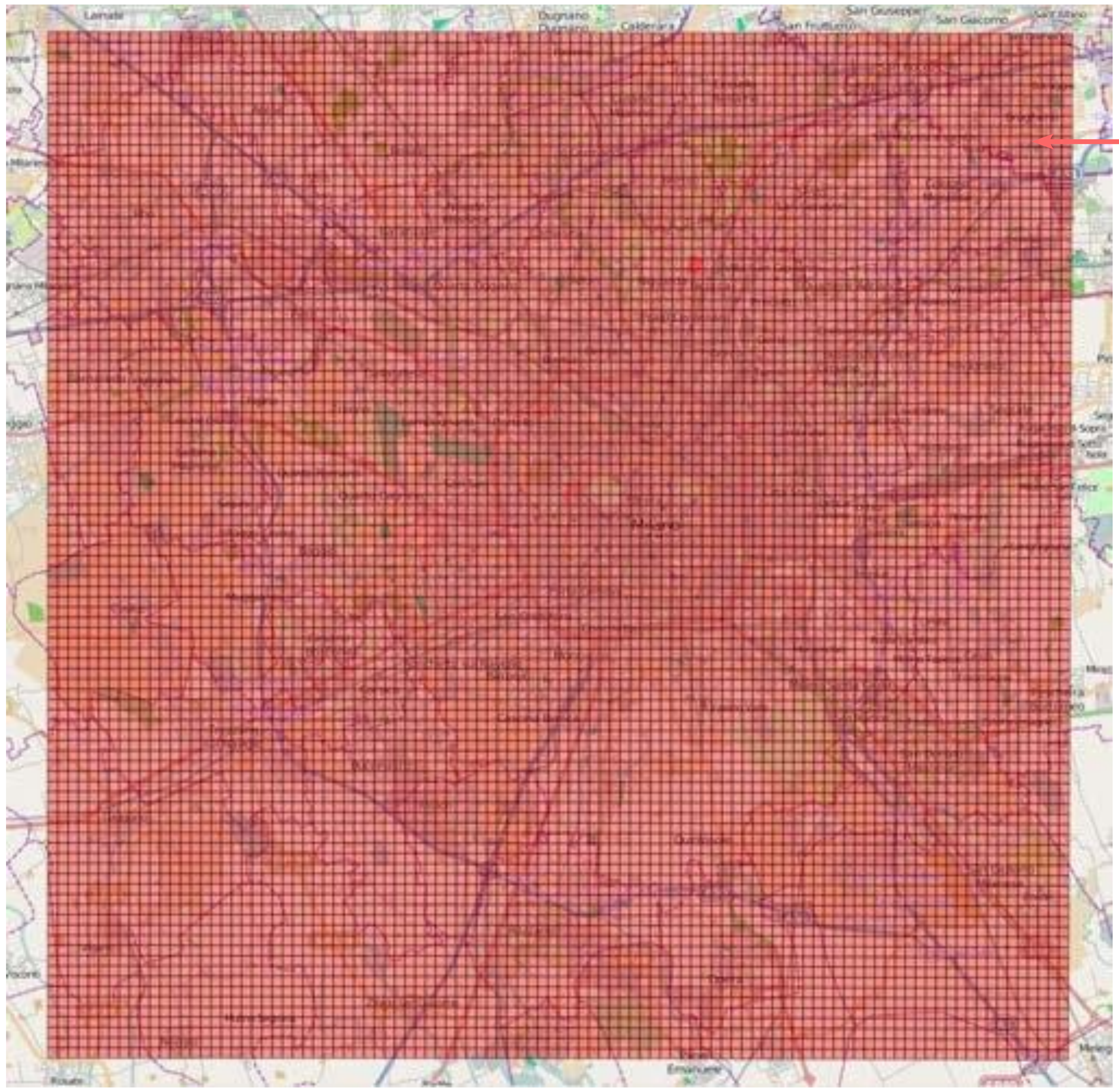
Yandex

# Telecommunications Analytics

Map and Reduce Side Joins

# Telecommunications - SMS, Call, Internet - MI



Milano Grid

› Square ID
›› Time Interval
›› Country Code
›› SMS-in Activity
›› SMS-out Activity
›› Call-in Activity
›› Call-out Activity
›› Internet Traffic Activity

Schema

해결할 문제 : Square ID를 South와 North 지역으로 구분해 SMS-in activity 합을 구한다.

# BIG data

로그 데이터

- ›› Square ID
- ›› Time Interval
- ›› Country Code
- ›› SMS-in Activity
- ›› SMS-out Activity
- ›› Call-in Activity
- ›› Call-out Activity
- ›› Internet Traffic Activity

지역을 구분하기 위한 Join이 필요

```
1 1383260400000 0 0.08136262351125882
1 1383260400000 39 0.14186425470242922
0.1567870050390246 0.16093793691701822
0.052274848528573205 11.028366381681026
1 1383261000000 0 0.13658782275823106
0.02730046487718618
1 1383261000000 33
0.026137424264286602
…
```

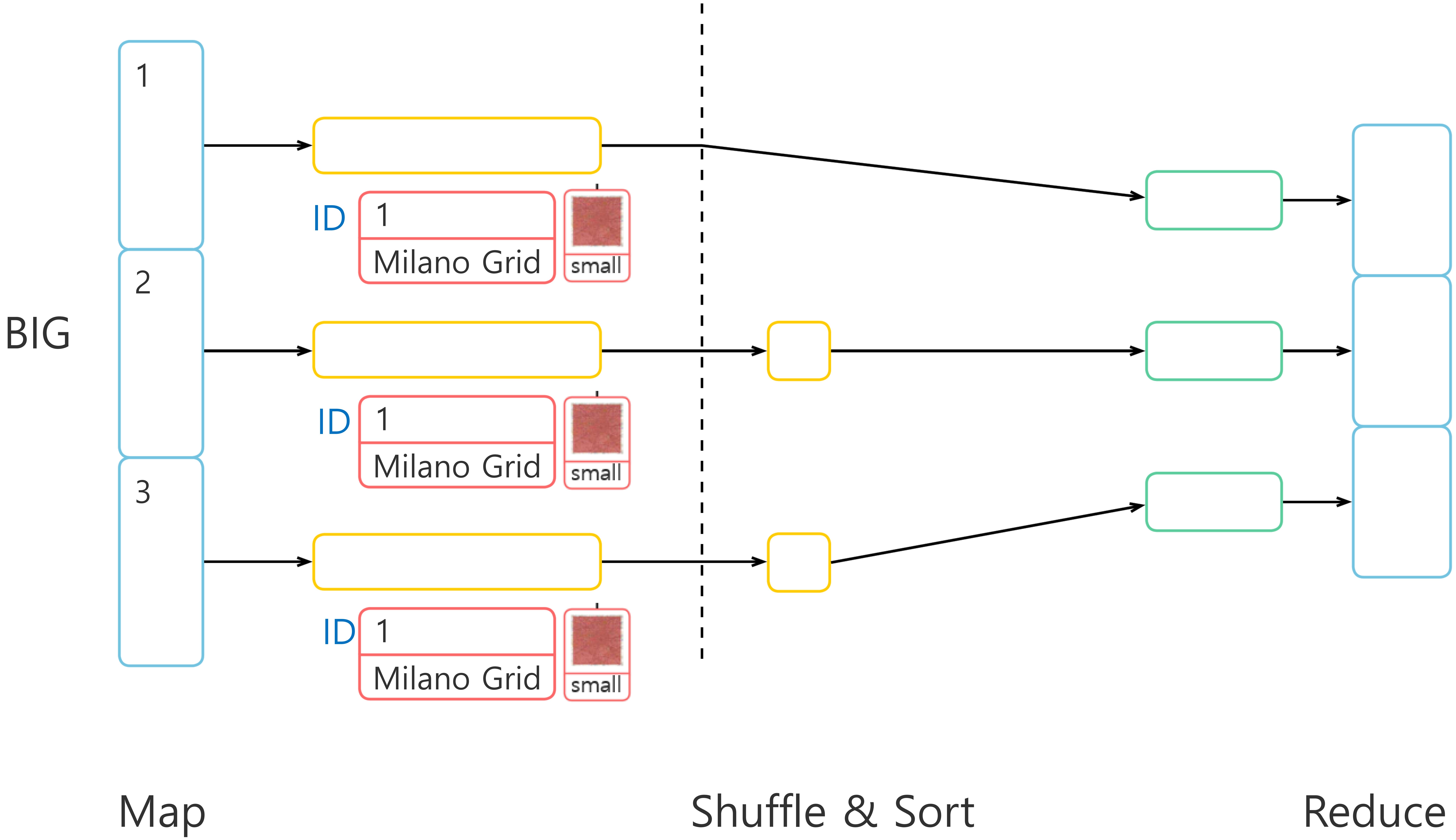
# small data

Square ID에 대한 위치정보

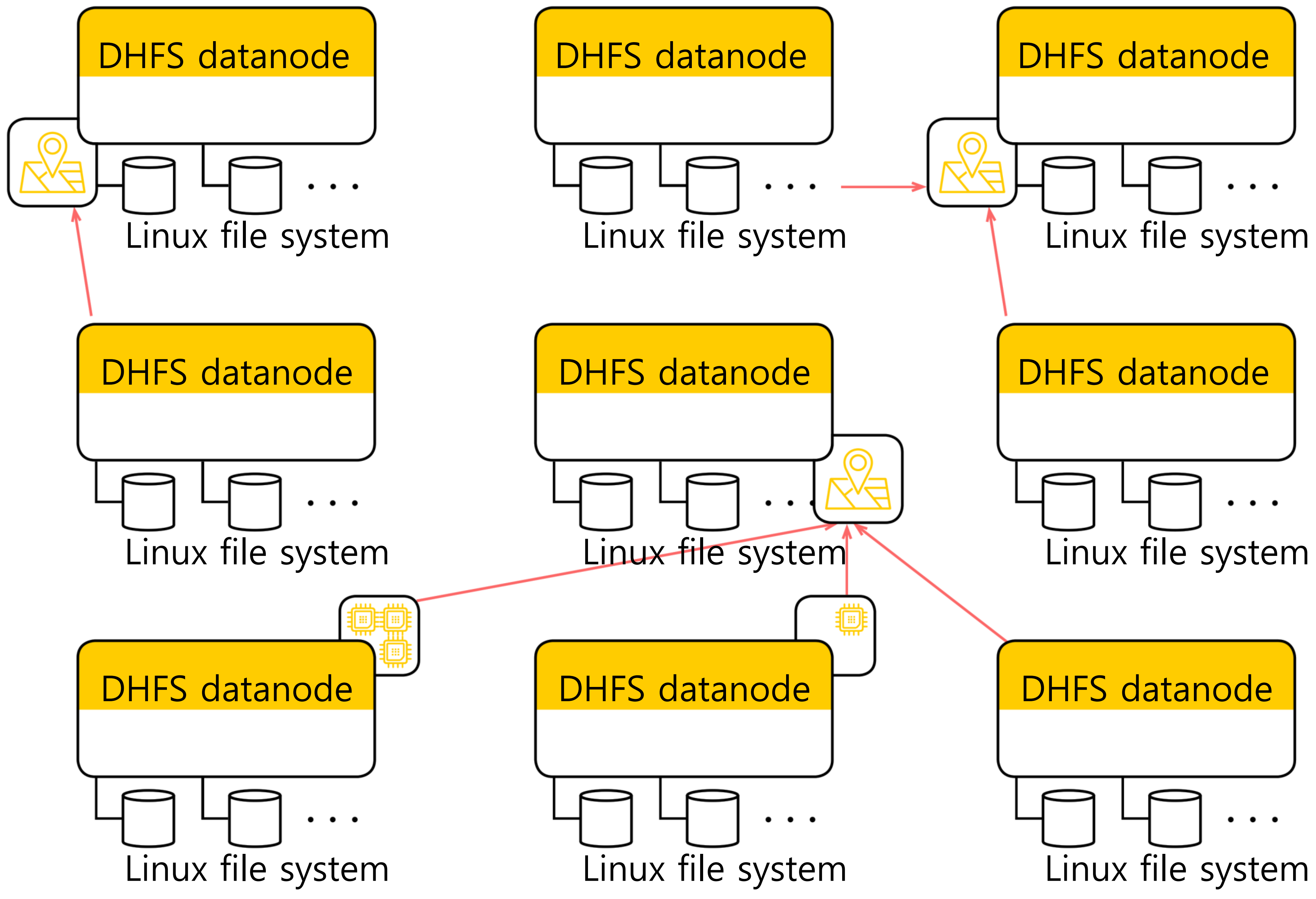
{‘type’: ‘Polygon’, ‘coordinates’:
[[[9.0114910478323, 45.35880131440966],
[9.014491488013135, 45.35880097314403],
[9.0144909480813, 45.35668565341486],
[9.011490619692509,
45.356685994655464], [9.0114910478323,
45.35880131440966]]]}
…

# Map-Side Join

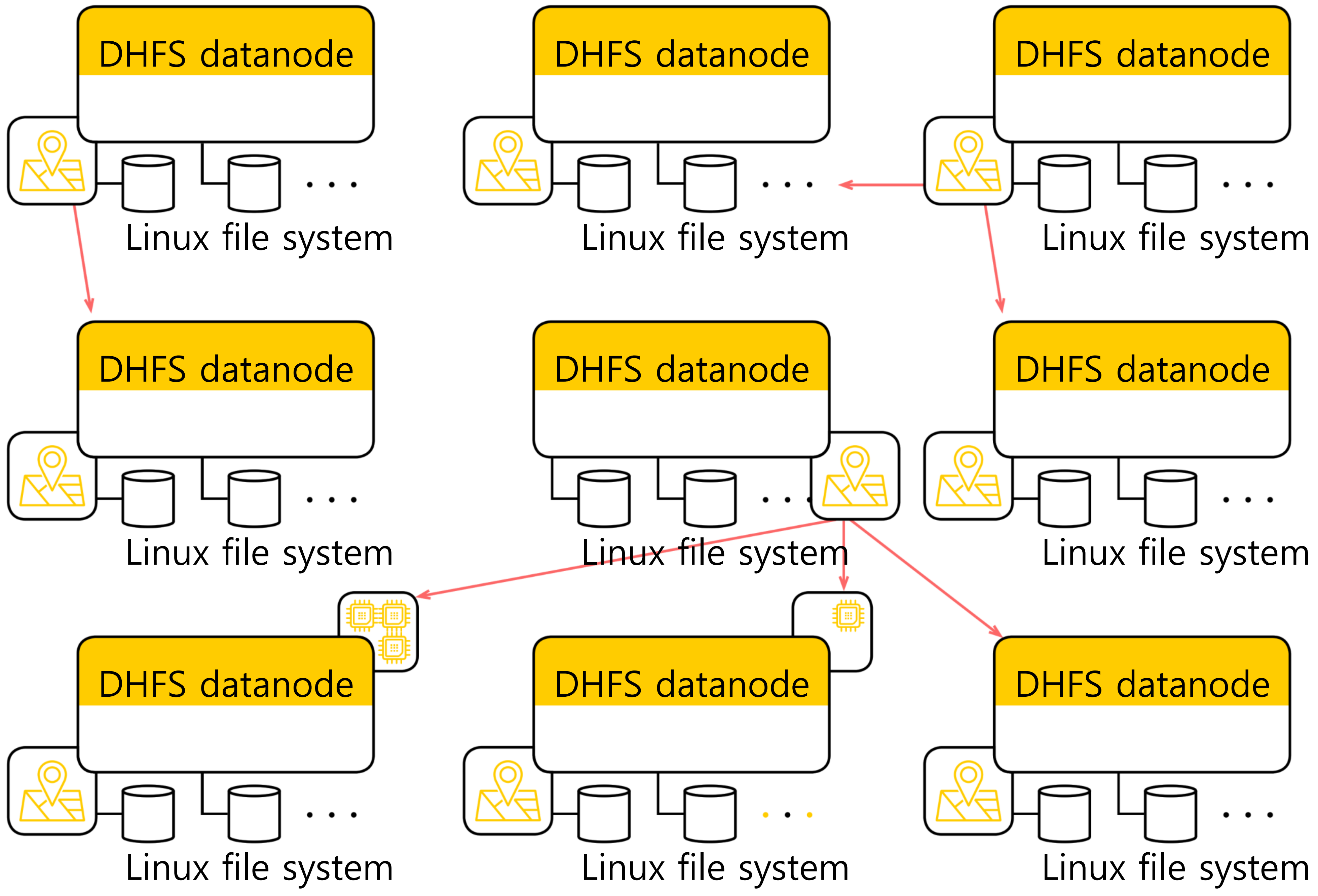
Mapper에서 데이터를 메모리로 읽어 join 하는 경우



# HDFS Read



# Distributed Cache



데이터가 HDFS에 저장되어 있는 경우, 3개의 replica에서
모든 mapper로 데이터를 전달해야 한다. => 비효율적

Distributed Cache를 활용하면 각 machine에 한 번만 네트워크로
보내면 mapper가 서로 공유하기 때문에 효율적이다.

```
yarn jar $HADOOP_STREAMING_JAR ₩
        -files read_from_hdfs_mapper.py,hdfs:///user/adral/milano-grid.geojson ₩
        -mapper 'python map_side_mapper.py' ₩
        -numReduceTasks 0 ₩
        -input /data/telecommunication ₩
        -output telecom-joins
```

```
$ hdfs dfs -cat telecom-joins/part-00000 | head
South   0.0813626235113
South   0.141864254702
South   0.136587822758
 ...
```

## map_side_mapper.py

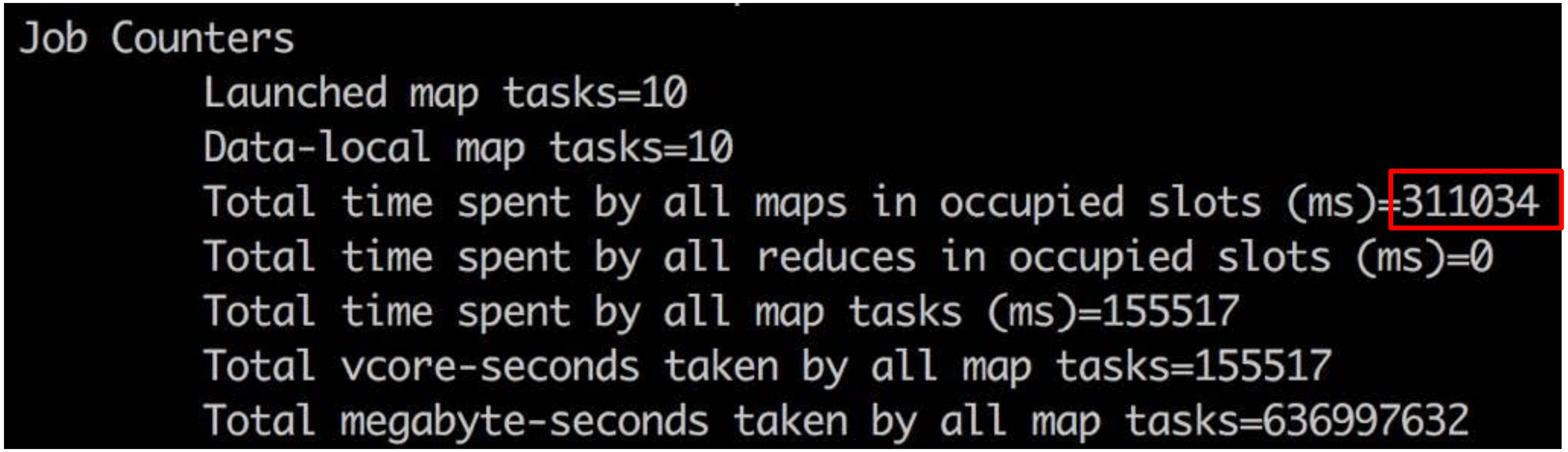
```
geojson = json.load(open("milano-grid.geojson"))
grid = load_grid(geojson)
for line in sys.stdin:
square_id,aggregate = line.split("₩t", 1)
square_id = int(square_id)
time_interval, country, sms_in, sms_out, call_in, call_out, internet = aggregate.split("₩t")
if sms_in:
sms_in = float(sms_in)
print(grid[square_id],sms_in,sep="₩t")
```

from load_grid function

load_grid에서 north or south 지역을 계산

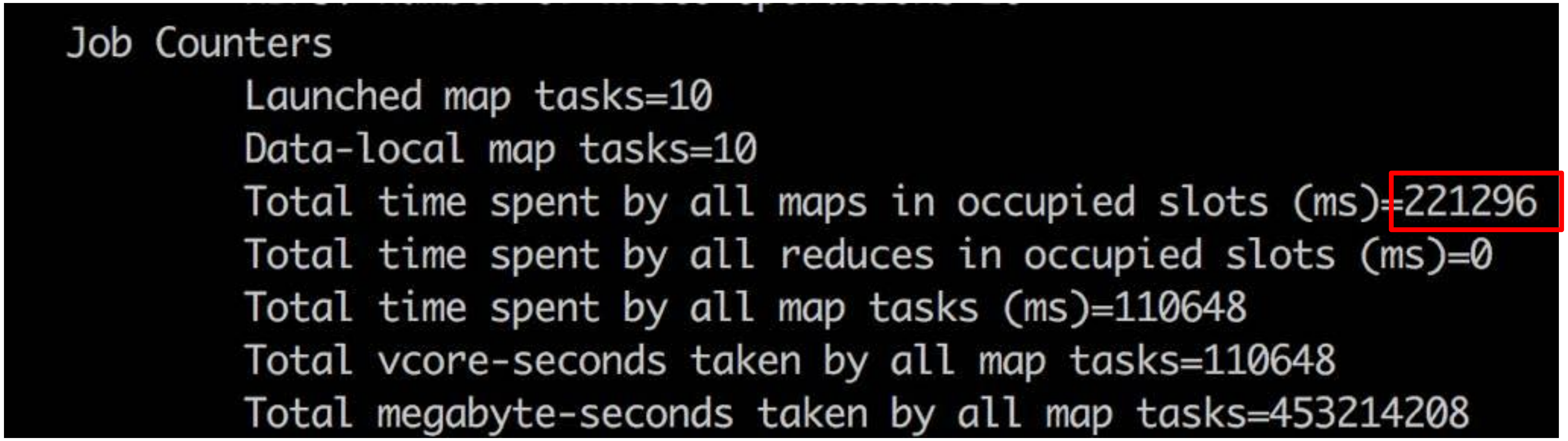
```
    grid[square_id] = "North" if (min_y + max_y) / 2 > middle_point else "South"
```

# HDFS read



```
Job Counters
        Launched map tasks=10
        Data-local map tasks=10
        Total time spent by all maps in occupied slots (ms)=311034
        Total time spent by all reduces in occupied slots (ms)=0
        Total time spent by all map tasks (ms)=155517
        Total vcore-seconds taken by all map tasks=155517
        Total megabyte-seconds taken by all map tasks=636997632
```

# local read; Distributed Cache

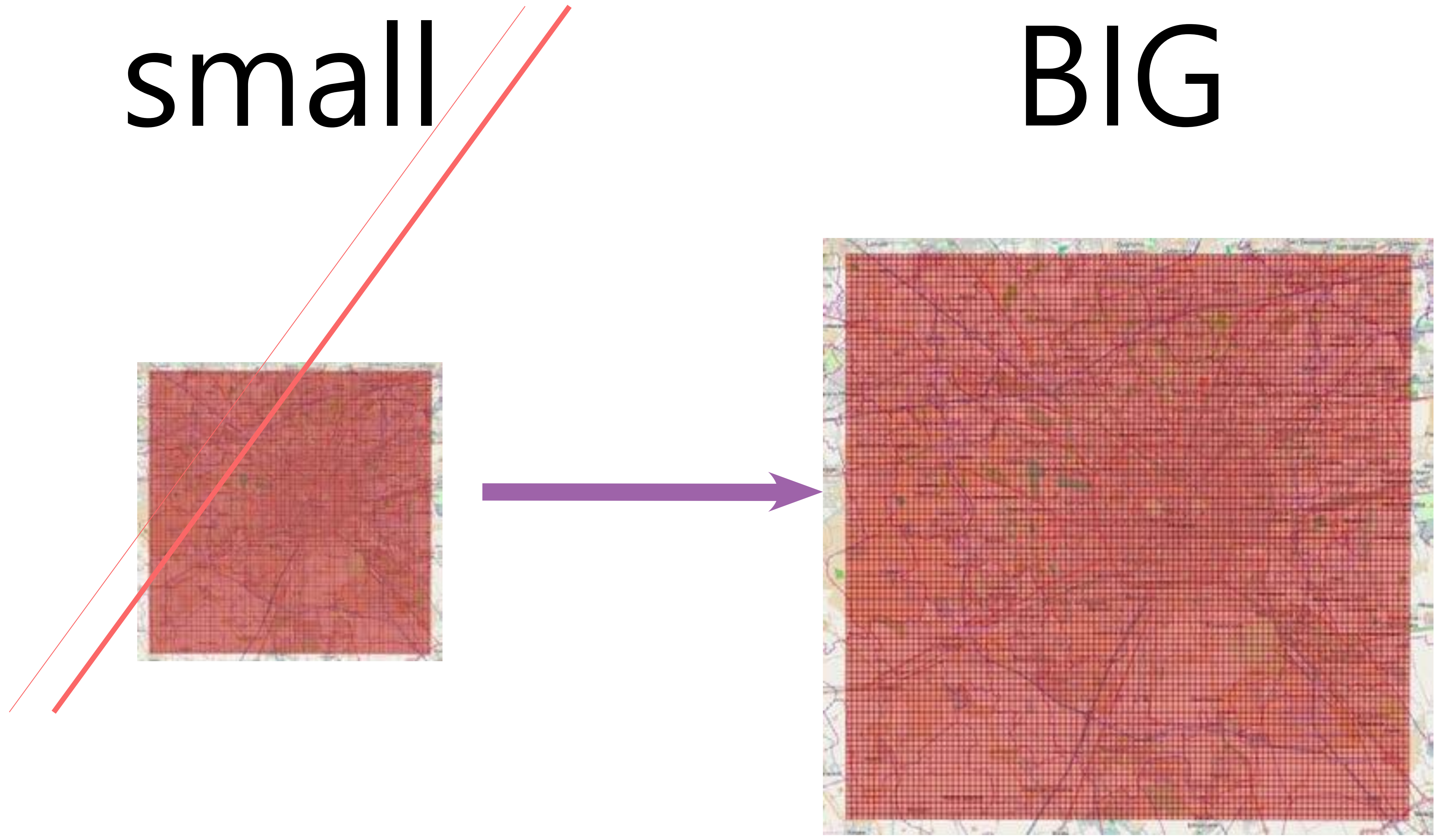


```
 Job Counters
        Launched map tasks=10
        Data-local map tasks=10
        Total time spent by all maps in occupied slots (ms)=221296
        Total time spent by all reduces in occupied slots (ms)=0
        Total time spent by all map tasks (ms)=110648
        Total vcore-seconds taken by all map tasks=110648
        Total megabyte-seconds taken by all map tasks=453214208
```

grid 데이터가 BIG 데이터인 경우, mapper 메모리에 데이터를 저장할 수 없어서 Map-Side Join은 사용불가

# BIG

›› Square ID
›› Time Interval
›› Country Code
›› SMS-in Activity
›› SMS-out Activity
›› Call-in Activity
›› Call-out Activity
›› Internet Traffic Activity



~~small~~

BIG

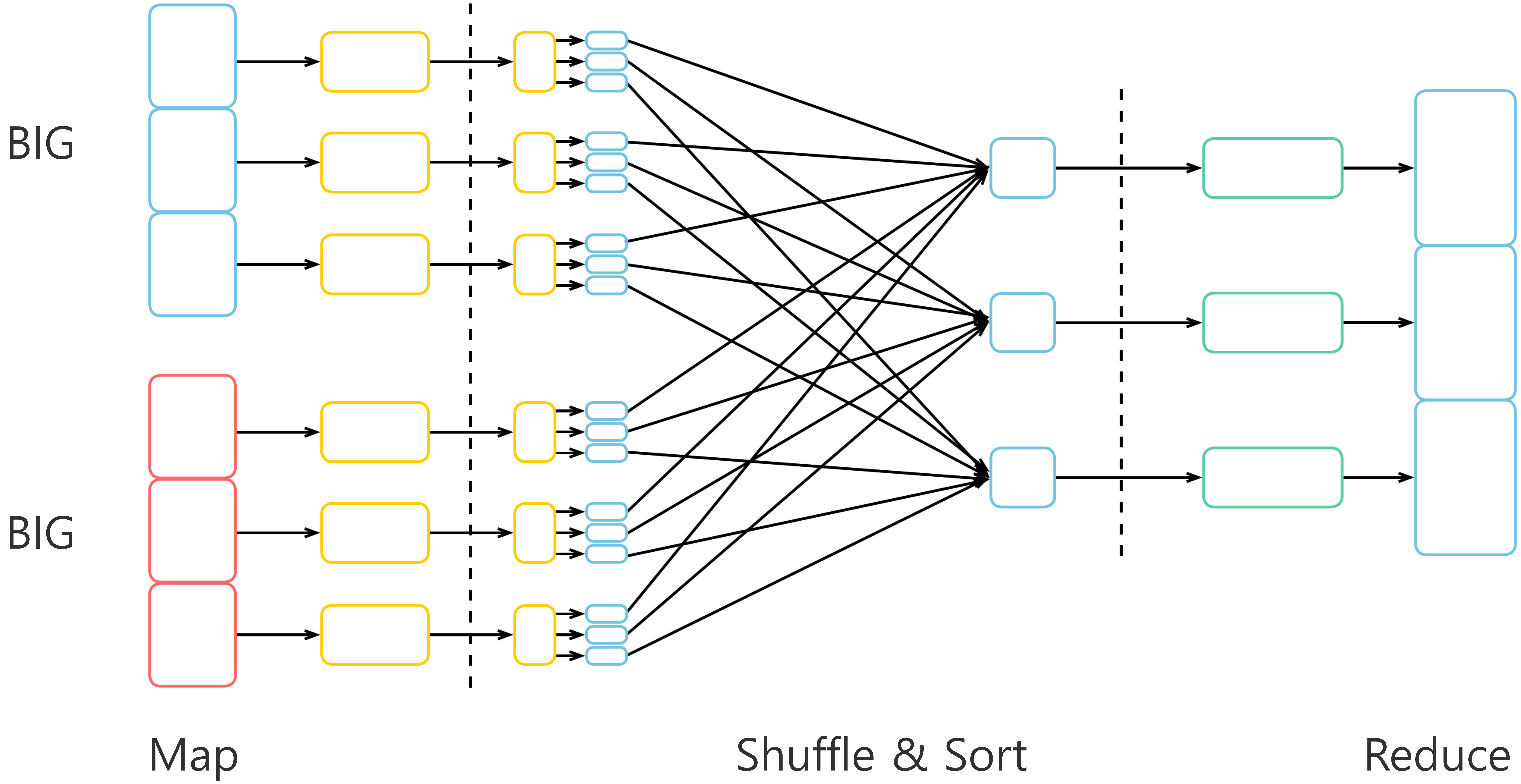
```
1 1383260400000 0 0.08136262351125882
1 1383260400000 39 0.1418642547024292
0.1567870050390246 0.16093793691701822
0.052274848528573205 11.028366381681026
1 1383261000000 0 0.13658782275823106
0.02730046487718618
1 1383261000000 33
0.026137424264286602
…
```

{‘type’: ‘Polygon’, ‘coordinates’:
[[[9.0114910478323, 45.35880131440966],
[9.014491488013135, 45.35880097314403],
[9.0144909480813, 45.35668565341486],
[9.011490619692509,
45.356685994655464], [9.0114910478323,
45.35880131440966]]]}
…

# Reduce-Side Join

Reducer에서 데이터를 join하는 경우

Mapper에서 선행 작업이 필요하다.



## reduce_side_mapper.py

```
if "geojson" in os.environ["mapreduce_map_input_file"]:
    geojson = json.load(sys.stdin)
    grid = load_grid(geojson)
    for grid_id, ce11_type in grid.items():
        print(grid_id, "grid", ce11_type, sep="₩t")
else
    for line in sys.stdin:
        square_id, aggregate = line.split("₩t", 1)
        square_id = int(square_id)
        time_interval, country, sms_in, sms_out, call_in, call_out, internet = aggregate.split("₩t"
        if sms_in:
            sms_in = float(sms_in)
        print(square_id, "logs", sms_in, sep="₩t")
```

os.environ["mapreduce_map_input_file"] : input 파일명

mapper에서 log 데이터와 geojson 데이터 2종류를 받는다.

구분해서 처리하기 위해 파일명으로 확인하고 output으로 grid 또는 logs 텍스트를 추가

# Mapper만 먼저 확인

참고 : gejson이 BIG 데이터일 경우, distributed cache를 활용하지 않고 input으로 입력한다.

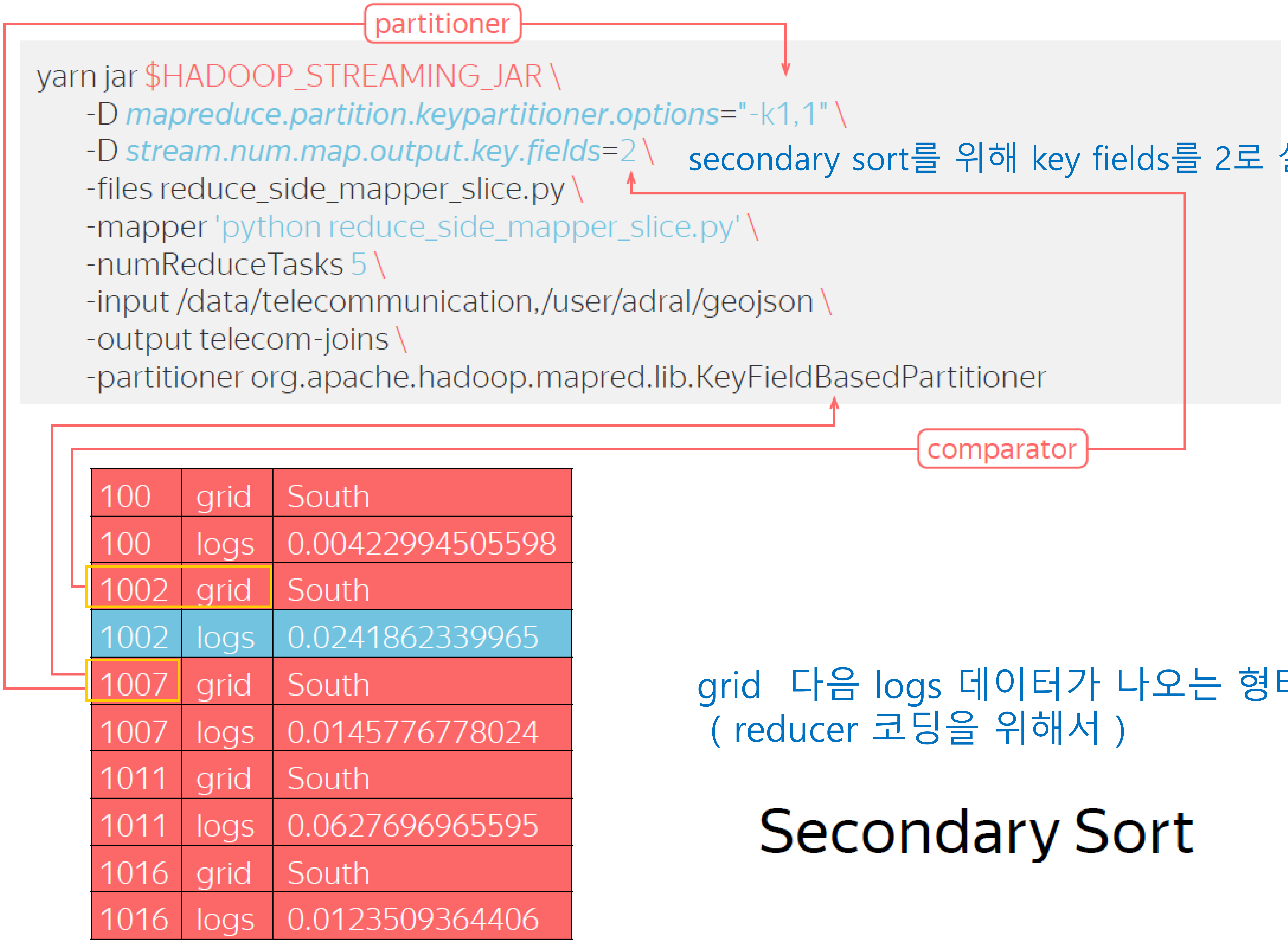
```
yarn jar $HADOOP_STREAMING_JAR ₩
             -files reduce_side_mapper.py ₩
             -mapper 'python reduce_side_mapper.py’ ₩
             -numReduceTasks 0 ₩
             -input /data/telecommunication, /user/adral/geojson ₩
             -output telecom-joins
```

```
$ hdfs dfs -text telecom-joins/part-00010 | head -3
1 grid South
2 grid South
3 grid South
```

string

```
$ hdfs dfs -text telecom-joins/part-00000 | head -3
1 logs 0.0813626235113
1 logs 0.141864254702
1 logs 0.136587822758
```

numeric



```
yarn jar $HADOOP_STREAMING_JAR \
    -D mapreduce.partition.keypartitioner.options="-k1,1" \
    -D stream.num.map.output.key.fields=2 \
    -files reduce_side_mapper_slice.py \
    -mapper 'python reduce_side_mapper_slice.py' \
    -numReduceTasks 5 \
    -input /data/telecommunication,/user/adral/geojson \
    -output telecom-joins \
    -partitioner org.apache.hadoop.mapred.lib.KeyFieldBasedPartitioner
```

secondary sort를 위해 key fields를 2로 설정

| | | |
|---|---|---|
| 100 | grid | South |
| 100 | logs | 0.00422994505598 |
| 1002 | grid | South |
| 1002 | logs | 0.0241862339965 |
| 1007 | grid | South |
| 1007 | logs | 0.0145776778024 |
| 1011 | grid | South |
| 1011 | logs | 0.0627696965595 |
| 1016 | grid | South |
| 1016 | logs | 0.0123509364406 |

grid 다음 logs 데이터가 나오는 형태로 정렬
( reducer 코딩을 위해서 )

# Secondary Sort

## reduce_side_reducer.py

grid를 먼저 읽고 logs의 value를 더한다.

```
from __future__ import print_function
import sys

current_grid = None
grid_load = 0
grid_location = None

for line in sys.stdin:
  grid_id, label, value = line.strip("₩n").split("₩t", 2)
  if label == "grid":
    if current_grid:
      print(current_grid, grid_location, grid_load, sep="₩t")
    current_grid = grid_id
    grid_load = 0
    grid_location = value
  else:
    counts = float(value)
    grid_load += counts

if current_grid != None:
  print(current_grid, grid_location, grid_load, sep="₩t")
```

```
yarn jar $HADOOP_STREAMING_JAR ₩
    -D mapreduce.partition.keypartitioner.options="-k1,1" ₩
    -D stream.num.map.output.key.fields=2 ₩
    -files reduce_side_mapper.py, reduce_side_reducer.py ₩
    -mapper ‘python reduce_side_mapper.py' ₩
    -reducer ‘python reduce_side_reducer.py’ ₩
    -numReduceTasks 5 ₩
    -input /data/telecommunication,/user/adral/geojson ₩
    -output telecom-joins ₩
    -partitioner org.apache.hadoop.mapred.lib.KeyFieldBasedPartitioner
```

| | | |
|---|---|---|
| 100 | South | 55.723185988 |
| 1002 | South | 26.6296384356 |
| 1007 | South | 25.216401618 |
| 1011 | South | 33.9120375534 |
| 1016 | South | 29.0697003186 |
| 1020 | South | 27.635637365 |
| 1025 | South | 21.7622321062 |
| 1034 | South | 93.4964559323 |
| 1039 | South | 106.557543309 |
| 1043 | South | 130.640809358 |

# Summary

›› you know how and when to use Map-Side Join

›› you know how and when to use Reduce-Side Join

›› you know how and when to use Secondary Sort

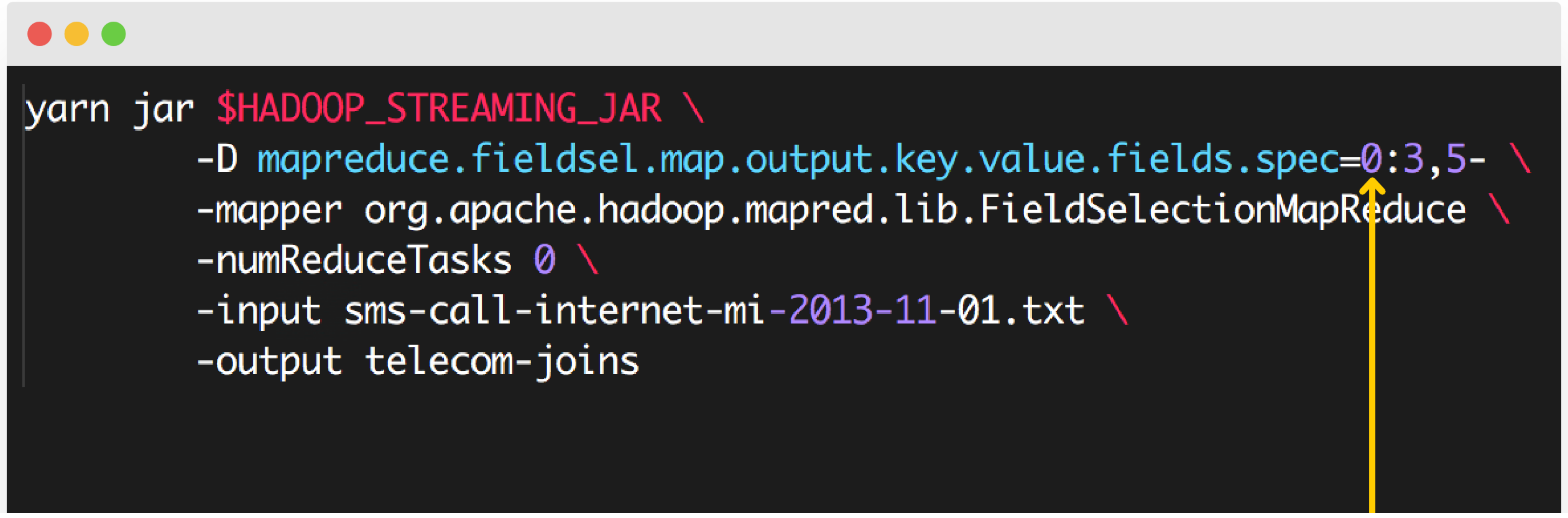
# Telecommunications Analytics

Job Chaining

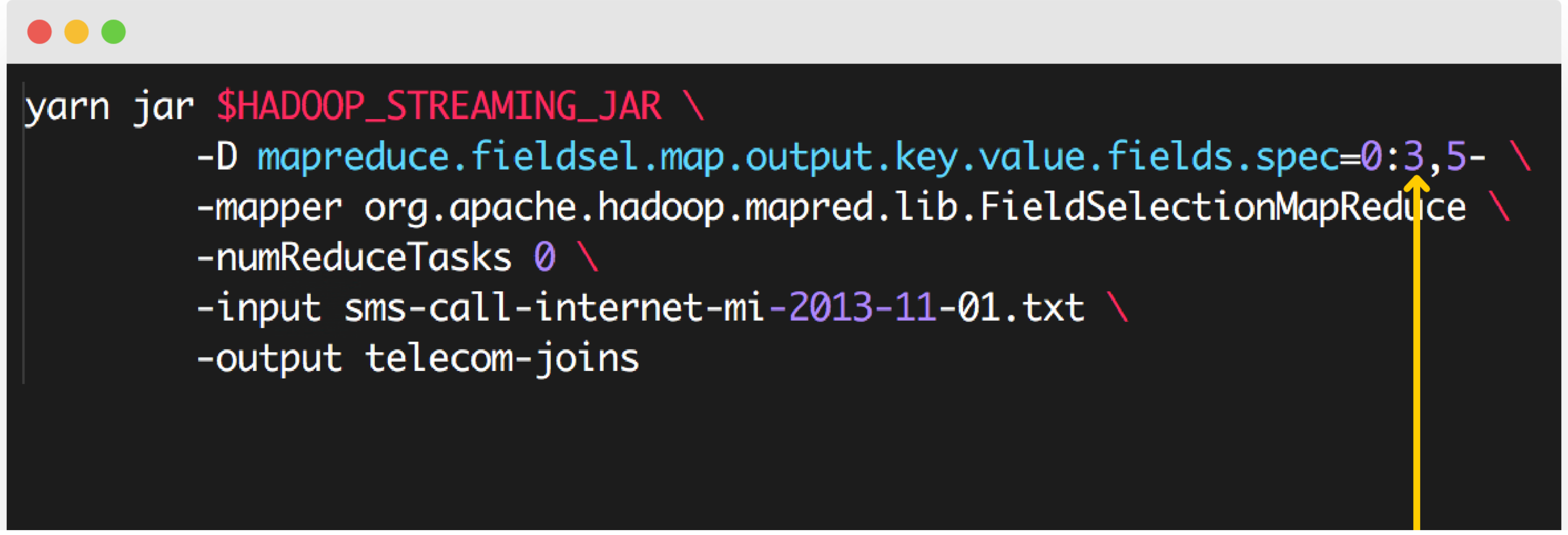
# KeyFieldSelection   필요한 데이터만 Key, Value를 간단히 추출하는 법

-D mapreduce.fieldsel.map.output.key.value.fields.spec=

-mapper org.apache.Hadoop.mapred.lib.FieldSelectionMapReduce



```
yarn jar $HADOOP_STREAMING_JAR \
        -D mapreduce.fieldsel.map.output.key.value.fields.spec=0:3,5- \
        -mapper org.apache.hadoop.mapred.lib.FieldSelectionMapReduce \
        -numReduceTasks 0 \
        -input sms-call-internet-mi-2013-11-01.txt \
        -output telecom-joins
```



```
yarn jar $HADOOP_STREAMING_JAR \
        -D mapreduce.fieldsel.map.output.key.value.fields.spec=0:3,5- \
        -mapper org.apache.hadoop.mapred.lib.FieldSelectionMapReduce \
        -numReduceTasks 0 \
        -input sms-call-internet-mi-2013-11-01.txt \
        -output telecom-joins
```

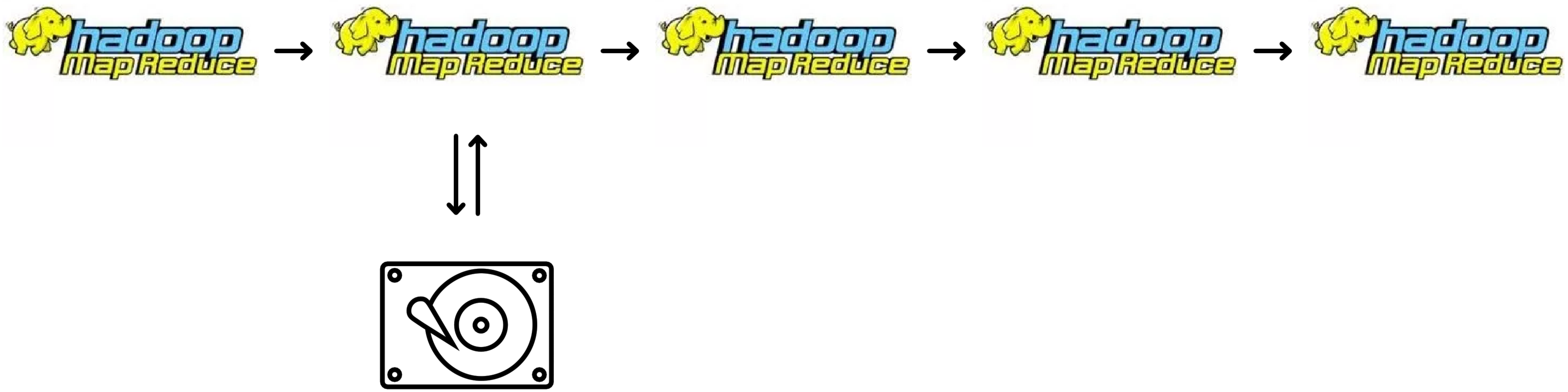
key index: 0 (Square ID)

value index: 3,5-

```
$ head sms-call-internet-mi-2013-11-01.txt
11383260400000  0   0.08136262351125882        4번째는 지워짐
1 1383260400000  39  0.14186425470242922  0.1567870050390246 0.16093793691701822
0.052274848528573205 11.028366381681026
1 1383261000000  0 0.13658782275823106    0.02730046487718618
...

$ head telecom-joins/part-00000
1  0.08136262351125882
1  0.14186425470242922  0.16093793691701822 0.052274848528573205  11.028366381681026
1 0.13658782275823106   0.02730046487718618
1    0.026137424264286602
 ...
```

# Job Chaining(파이프라인) 구성



## Job Chaining 활용 프레임워크

Java

PYDOOP

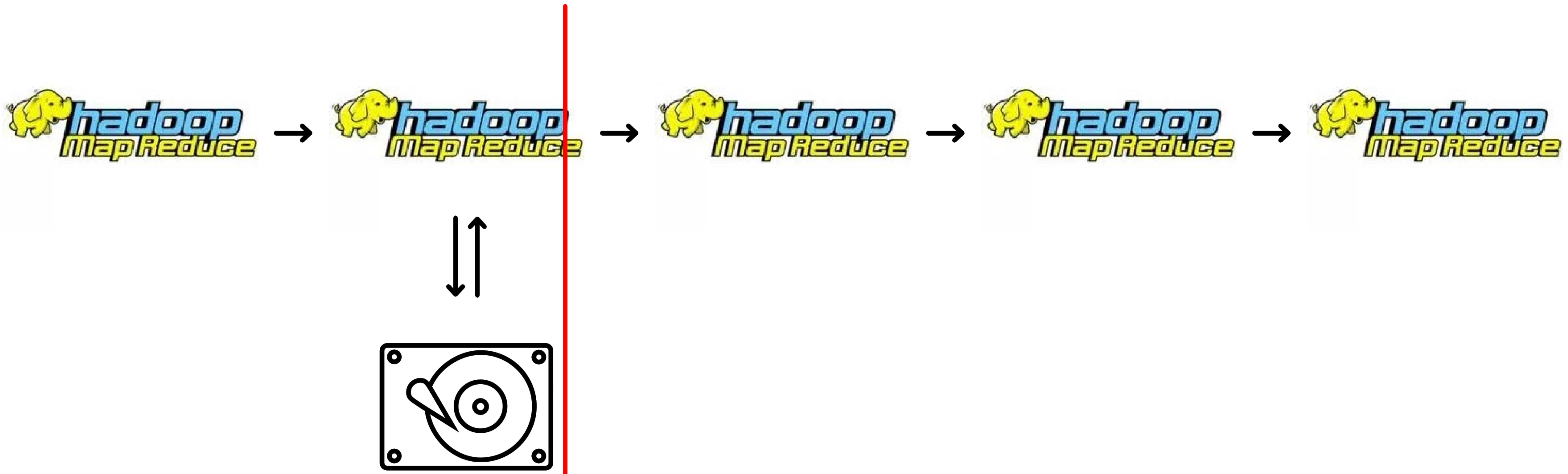
HADOOPY

MrJob

# Job Chaining(파이프라인) 구성 예시

FieldSelection

MapSide Join



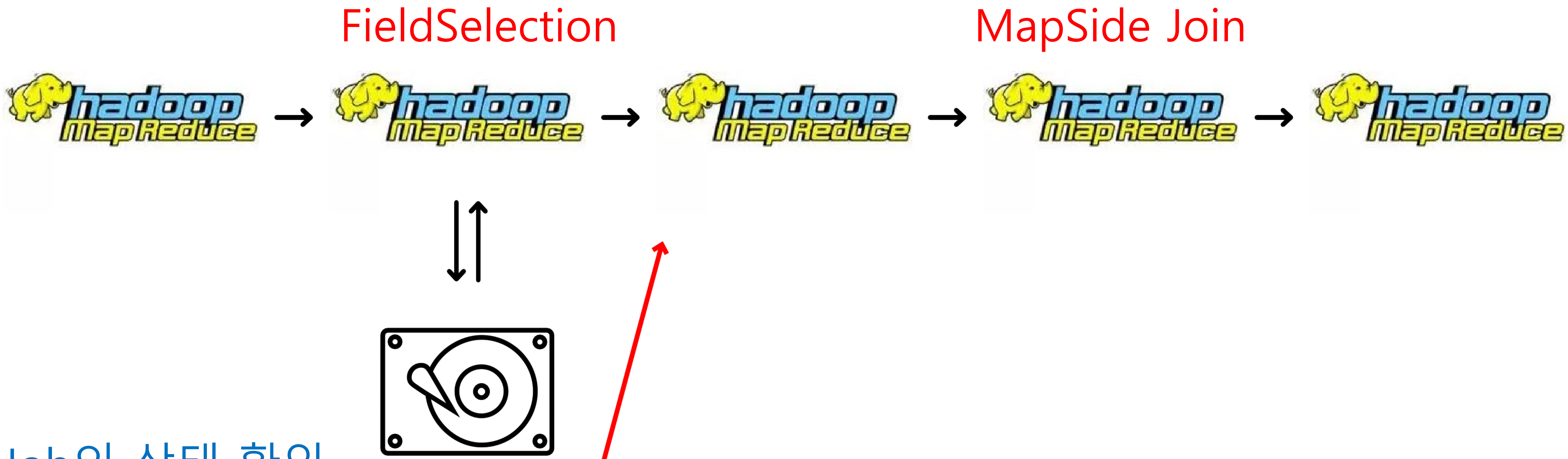
Job의 상태 확인

Job이 정상적으로 종료되면 return code가 0이다.

```
yarn jar …FieldSelectionMapReduce…
application_return_code=$?

if [ $application_return_code != "0" ]
then
    echo "FieldSelection phase was NOT successful"
    exit $application_return_code
fi
```

# Job Chaining(파이프라인) 구성 예시



## Job의 상태 확인

## 현재 application ID 확인방법

```
$ yarn application -list
         Application-Id     Application-Name     Application-Type        User
Queue              State          Final-State        Progress                 Tracking-URL

application_1491639197451_0559 select age from (select a.Age as age,...
1000(Stage-1)      MAPREDUCE      s201701 root.users.s201701              RUNNING
UNDEFINED          90.74% http://virtual-node1.atp-fivt.org:45272
…
application_1491639197451_0565streamjob7453411855907589438.jar
MAPREDUCE      adral     root.users.adral              RUNNING            UNDEFINED
5%  http://virtual-node3.atp-fivt.org:46751
…
```

# Job의 상태 확인 방법 1

```
$ yarn application -status application_1491639197451_0565
```

```
Application Report :
Application-Id : application_1491639197451_0565
Application-Name : streamjob7453411855907589438.jar
Application-Type : MAPREDUCE
User : adral
Queue : root.users.adral
Start-Time : 1492272089050
Finish-Time : 1492272181541
Progress : 100%
State : FINISHED
Final-State : SUCCEEDED
Tracking-URL : http://virtual-master.atp-fivt.org:19888/jobhistory/job/
job_1491639197451_0565
RPC Port : 42533
AM Host : virtual-node3.atp-fivt.org
Aggregate Resource Allocation : 544105 MB-seconds, 176 vcore-seconds
Log Aggregation Status : SUCCEEDED
Diagnostics:
```

## Job의 상태 확인 방법 2

```
$ hdfs dfs -test -e telecom-joins/field-selection/_SUCCESS
```

-e : exits

OUTPUT 폴더의 /_SUCCESS file은 job이 잘 끝났음을 의미한다.

```
$ hdfs dfs -ls telecom-joins/field-selection
```

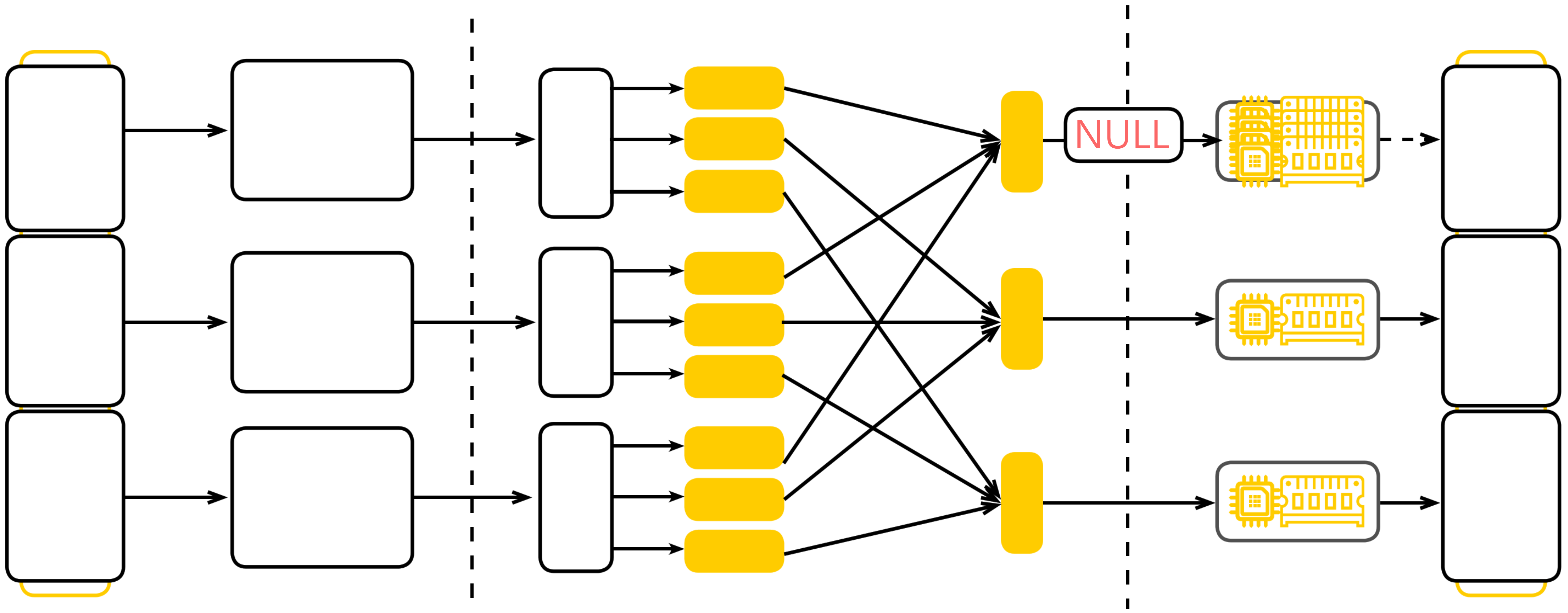
```
Found 11 items
-rw-r--r-- 3 ...       0 2017-04-15 18:30 telecom-joins/field-selection/_SUCCESS
-rw-r--r-- 3 ... 8516829 2017-04-15 18:28 telecom-joins/field-selection/part-00000
-rw-r--r-- 3 ... 8555052 2017-04-15 18:29 telecom-joins/field-selection/part-00001
-rw-r--r-- 3 ... 8380546 2017-04-15 18:29 telecom-joins/field-selection/part-00002
-rw-r--r-- 3 ... 8546802 2017-04-15 18:29 telecom-joins/field-selection/part-00003
-rw-r--r-- 3 ... 8293343 2017-04-15 18:29 telecom-joins/field-selection/part-00004
-rw-r--r-- 3 ... 8348375 2017-04-15 18:29 telecom-joins/field-selection/part-00005
...
```

# Telecommunications Analytics

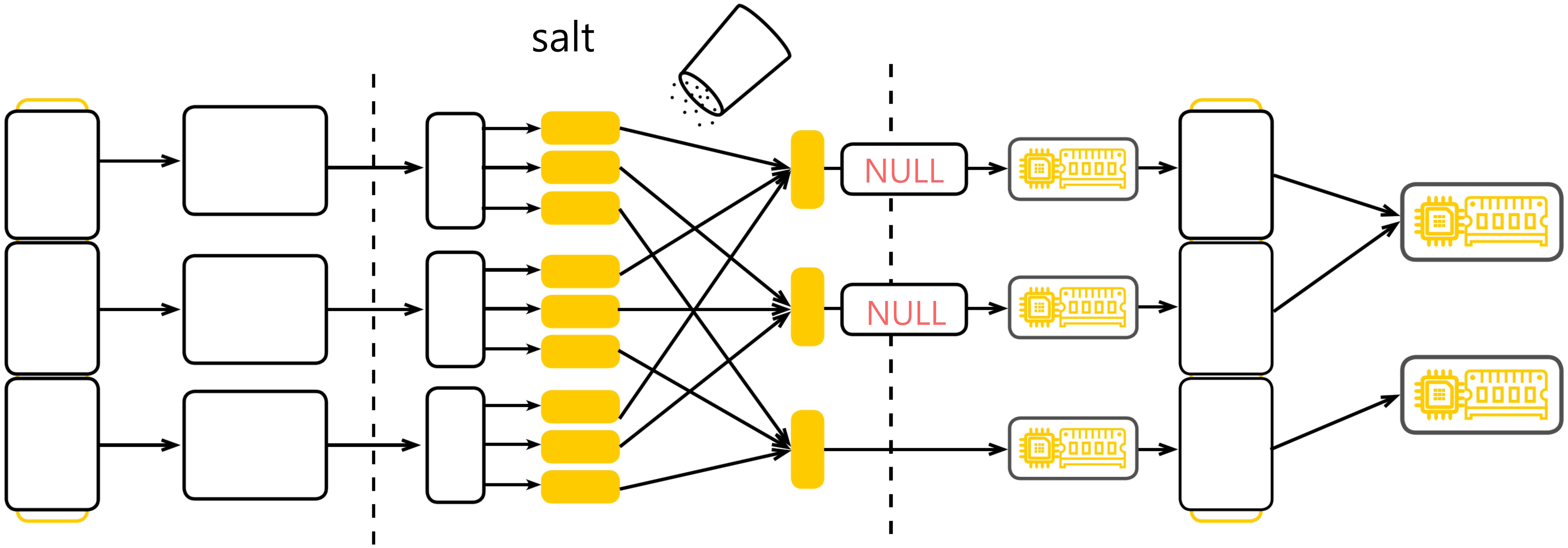
Data Skew, Salting

Data Skew란 특정 key에 대한 value가 너무 많을 때를 말한다.

=> 특정 reducer에 데이터가 많이 몰려 JOB 수행이 오래 걸린다.

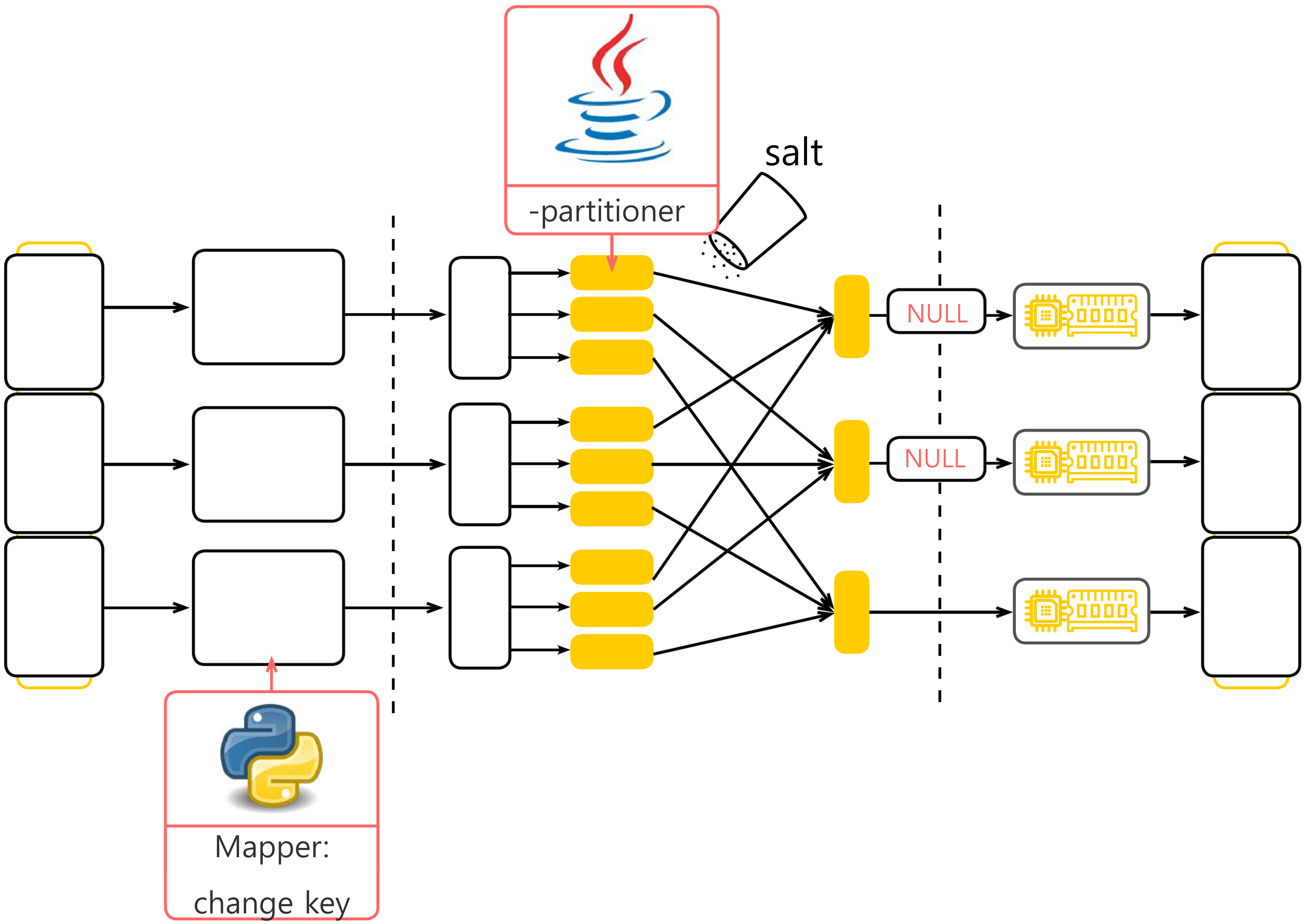


Salting은 같은 Key의 데이터를 임의로 여러 reducer로 분산시켜 작업하는 것

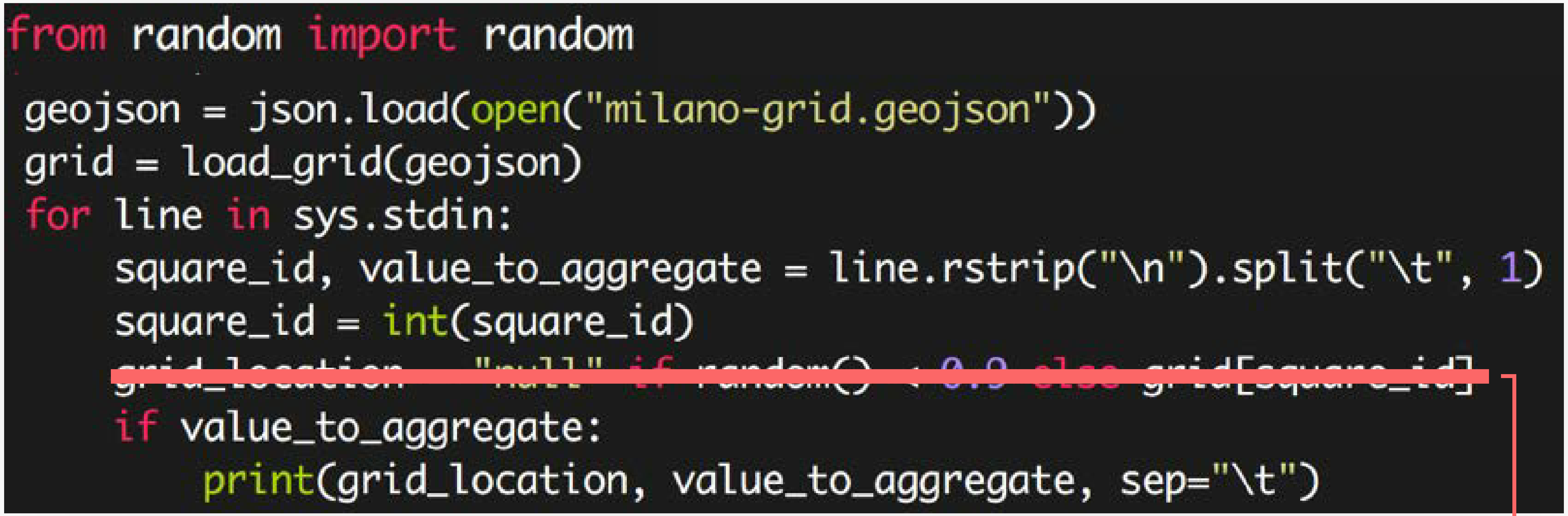


Salting 방법 2가지

- Java로 partitioner 코드를 수정하는 방법
- Mapper에서 Key를 임의로 수정하는 방법



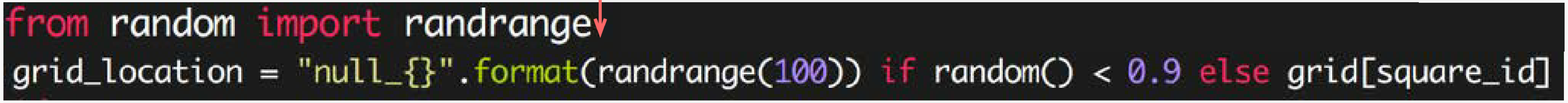
## Mapper



```
from random import random

geojson = json.load(open("milano-grid.geojson"))
grid = load_grid(geojson)
for line in sys.stdin:
    square_id, value_to_aggregate = line.rstrip("\n").split("\t", 1)
    square_id = int(square_id)
    ~~grid_location = "null" if random() < 0.9 else grid[square_id]~~
    if value_to_aggregate:
        print(grid_location, value_to_aggregate, sep="\t")
```

참고 : 테스트를 위해 10%의 데이터를 모두 Null 키를 만든다.

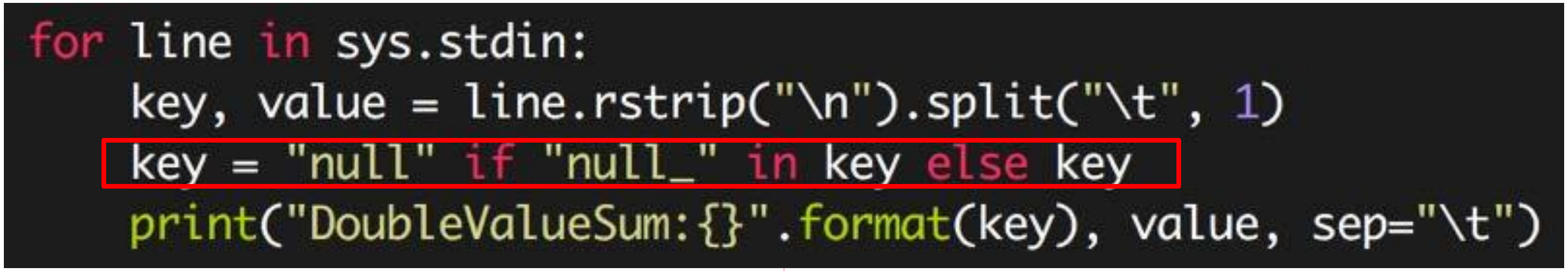


```
from random import randrange
grid_location = "null_{}".format(randrange(100)) if random() < 0.9 else grid[square_id]
```

null_+임의값 : Null 데이터를 분배하기 위해 Null key에 임의의 값을 더한다.

```
…
null_58 40989.56529872355
null_67 40775.58025775422
null_76 42430.98650098723
…
```

Job Chaining을 활용해 새로운 Job의 mapper로 null key를 원래대로 만든다.



```
for line in sys.stdin:
    key, value = line.rstrip("\n").split("\t", 1)
    key = "null" if "null_" in key else key
    print("DoubleValueSum:{}".format(key), value, sep="\t")
```

-reducer aggregate

```
South 164302.58197312435
null 4145425.004916422
North 296659.74407499237
```