Telecommunications Analytics

Map and Reduce Side Joins

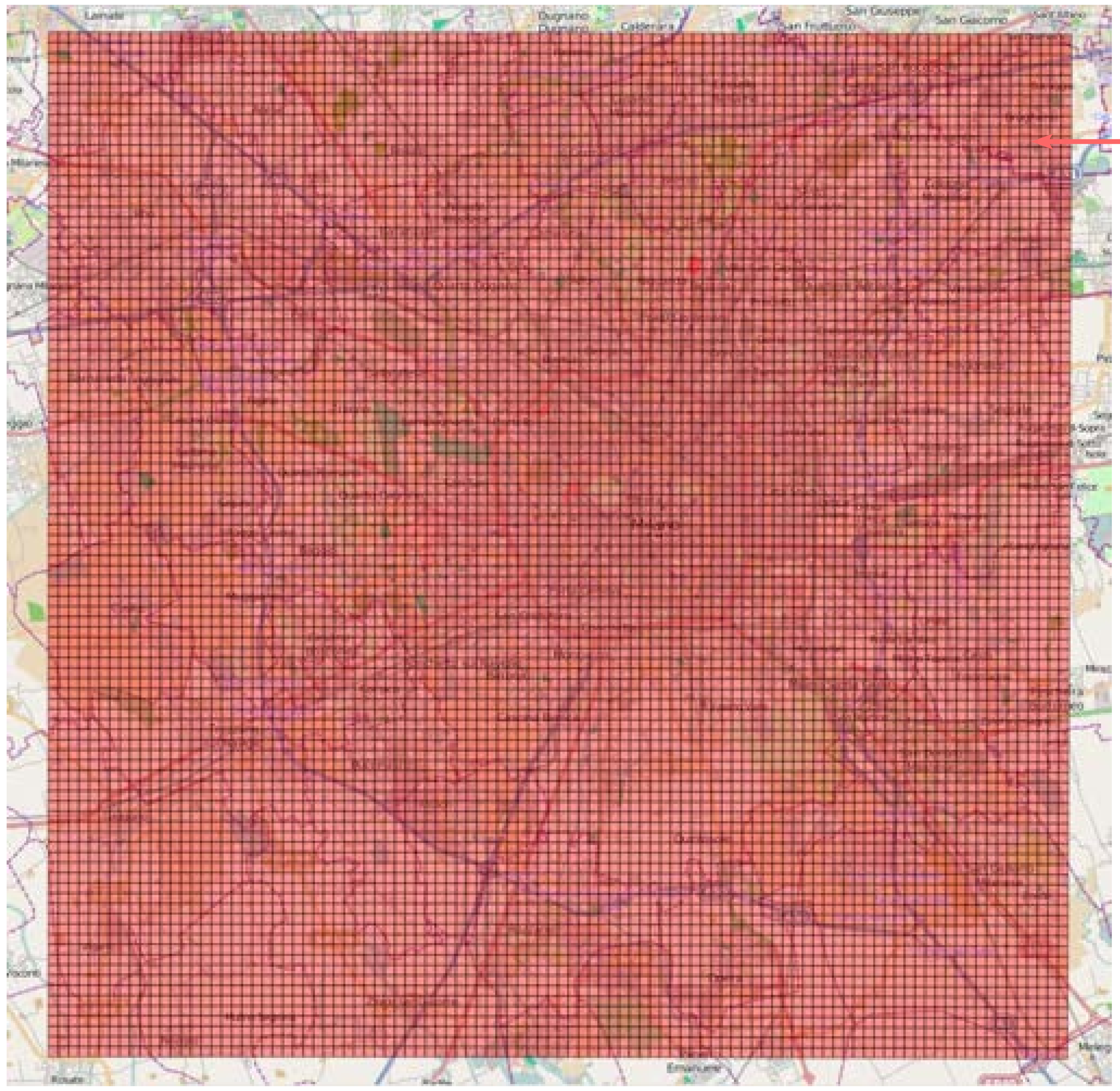
Telecommunications - SMS, Call, Internet - MI

Telecommunications - SMS, Call, Internet - MI

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

Schema

Telecommunications - SMS, Call, Internet - MI



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

Milano Grid

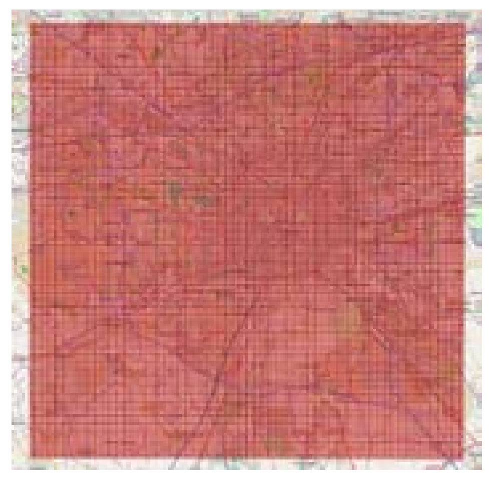
Schema

BIG

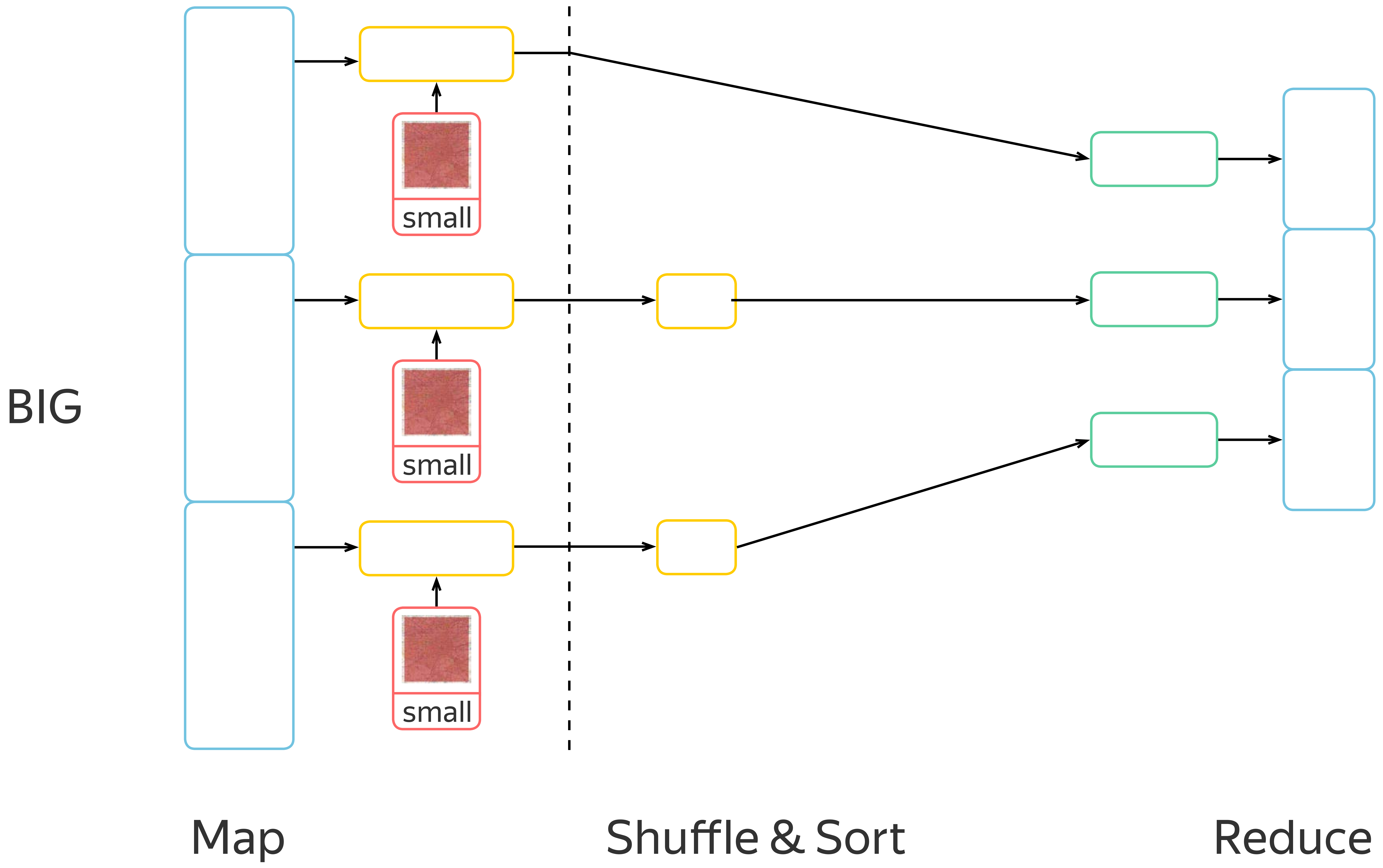
- Square ID
- Time Interval
- Country Code
- SMS-in Activity
- SMS-out Activity
- Call-in Activity
- Call-out Activity
- Internet Traffic Activity

```
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



{'type': 'Polygon', 'coordinates':
[[[9.0114910478323, 45.35880131440966],
[9.014491488013135, 45.35880097314403],
[9.0144909480813, 45.35668565341486],
[9.011490619692509,
45.356685994655464], [9.0114910478323,
45.35880131440966]]]}
...


BIG
small
small
small
Map
Shuffle & Sort
Reduce

```
def download_grid(hdfs_path):
        child_process = subprocess.Popen([
              "hdfs", "dfs", "-cat", hdfs_path
              ], stdout=subprocess.PIPE)
        out, err = child_process.communicate()
        geojson = json.loads(out)
        return geojson
```

```
def download_grid(hdfs_path):
        child_process = subprocess.Popen([
              "hdfs", "dfs", "-cat", hdfs_path
              ], stdout=subprocess.PIPE)
        out, err = child_process.communicate()
        geojson = json.loads(out)
        return geojson

geojson = download_grid("/user/adral/milane-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.sp1it("\t")
  if sms_in:
    sms_in = float(sms_in)
    print(grid[square_id], sms_in, sep="\t")
```

```
def download_grid(hdfs_path):
        child_process = subprocess.Popen([
              "hdfs", "dfs", "-cat", hdfs_path
              ], stdout=subprocess.PIPE)
        out, err = child_process.communicate()
        geojson = json.loads(out)
        return geojson

geojson = download_grid("/user/adral/milane-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.sp1it("\t")
  if sms_in:
    sms_in = float(sms_in)
    print(grid[square_id], sms_in, sep="\t")
```

```
def download_grid(hdfs_path):
        child_process = subprocess.Popen([
              "hdfs", "dfs", "-cat", hdfs_path
              ], stdout=subprocess.PIPE)
        out, err = child_process.communicate()
        geojson = json.loads(out)
        return geojson

geojson = download_grid("/user/adral/milane-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.sp1it("\t")
  if sms_in:
    sms_in = float(sms_in)
    print(grid[square_id], sms_in, sep="\t")
```

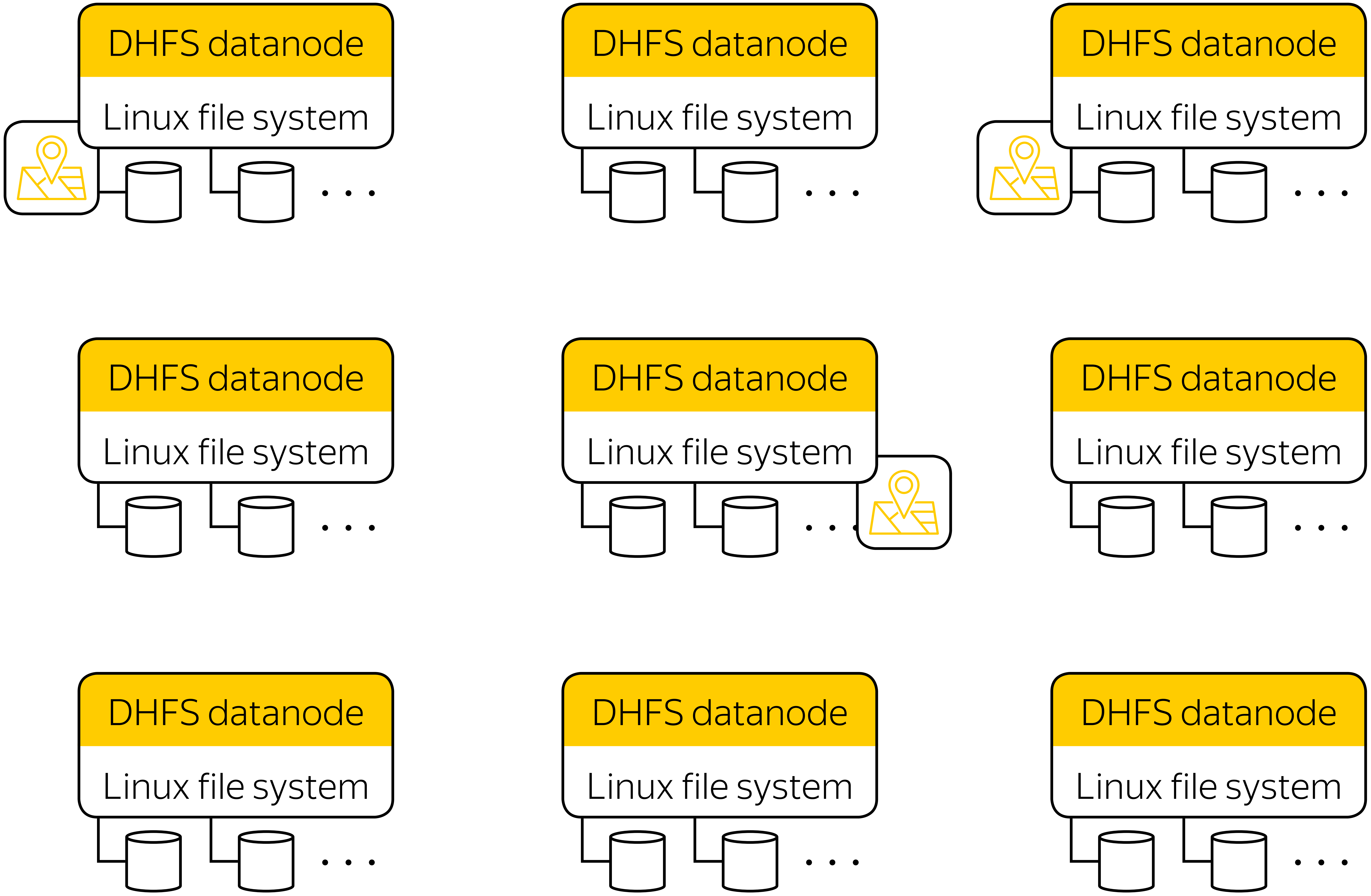
from load_grid function

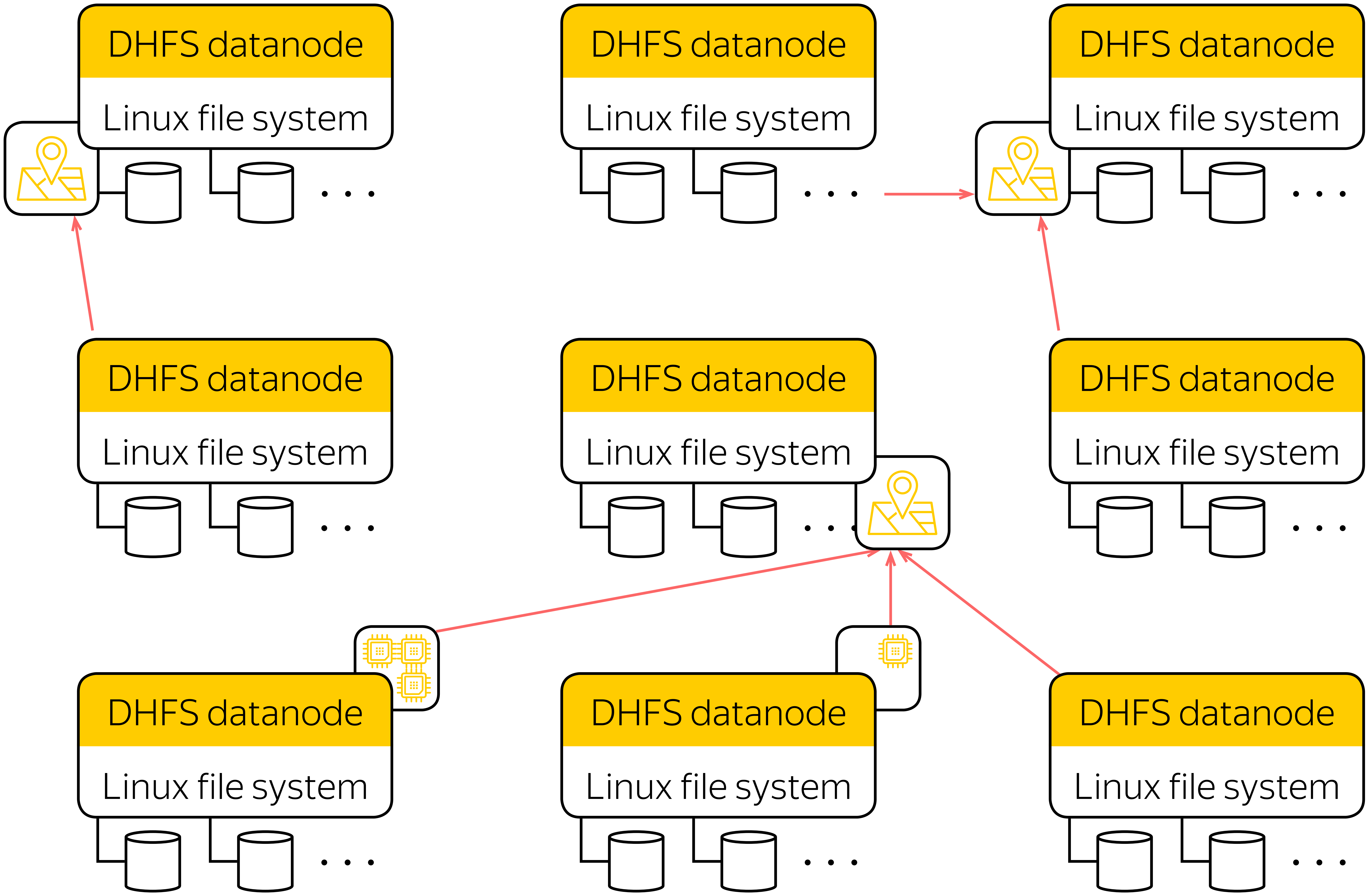
```
grid[square_id] = "North" if (min_y + max_y) / 2 > middle_point else "South"
```

```
yarn jar $HADOOP_STREAMING_JAR \
        -files read_from_hdfs_mapper.py \
        -mapper 'python read_From_hdfs_mapper.py' \
        -numReduceTasks 0 \
        -input /data/telecommunication \
        -output telecom-joins
```

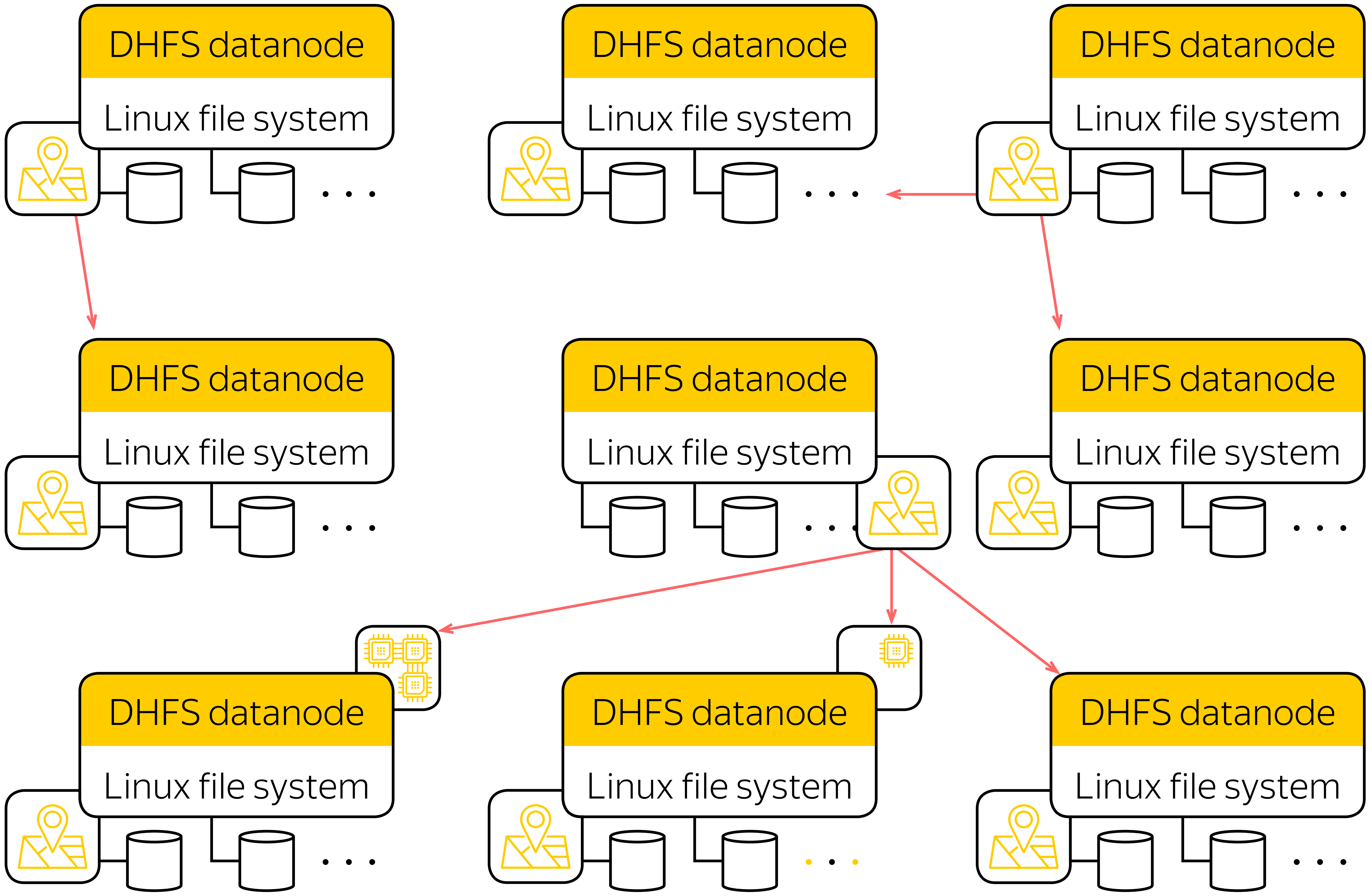
```
$ hdfs dfs -cat telecom-joins/part-00000 | head
South 0.0813626235113
South 0.141864254702
South 0.136587822758
South 0.278452077461
...
```

```
$ ./run_join.sh 10.03s user 0.32s system 30% cpu 33.444 total
```


DHFS datanode
Linux file system
DHFS datanode
Linux file system
DHFS datanode
Linux file system
DHFS datanode
Linux file system
DHFS datanode
Linux file system
DHFS datanode
Linux file system
DHFS datanode
Linux file system
DHFS datanode
Linux file system
DHFS datanode
Linux file system


DHFS datanode
Linux file system
DHFS datanode
Linux file system
DHFS datanode
Linux file system
DHFS datanode
Linux file system
DHFS datanode
Linux file system
DHFS datanode
Linux file system
DHFS datanode
Linux file system
DHFS datanode
Linux file system
DHFS datanode
Linux file system

Distributed Cache



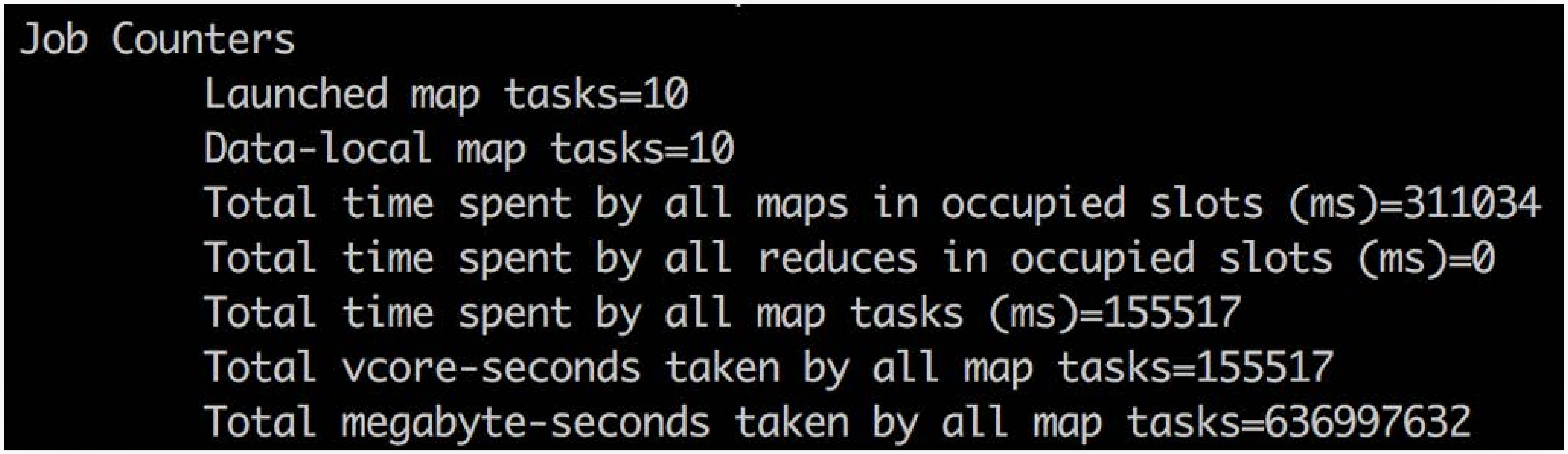
```
~~def download_grid(hdfs_path):~~
        ~~child_process = subprocess.Popen([~~
                ~~"hdfs", "dfs", "-cat",hdfs_path~~
                ~~], stdout=subprocess.PIPE)~~
        ~~out, err = child_process.communicate()~~
        ~~geojson = json.loads(out)~~
        return geojson

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.sp1it("\t")
    if sms_in:
        sms_in = float(sms_in)
        print(grid[square_id],sms_in,sep="\t")
```

HDFS data
Distributed
Cache

```
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
```

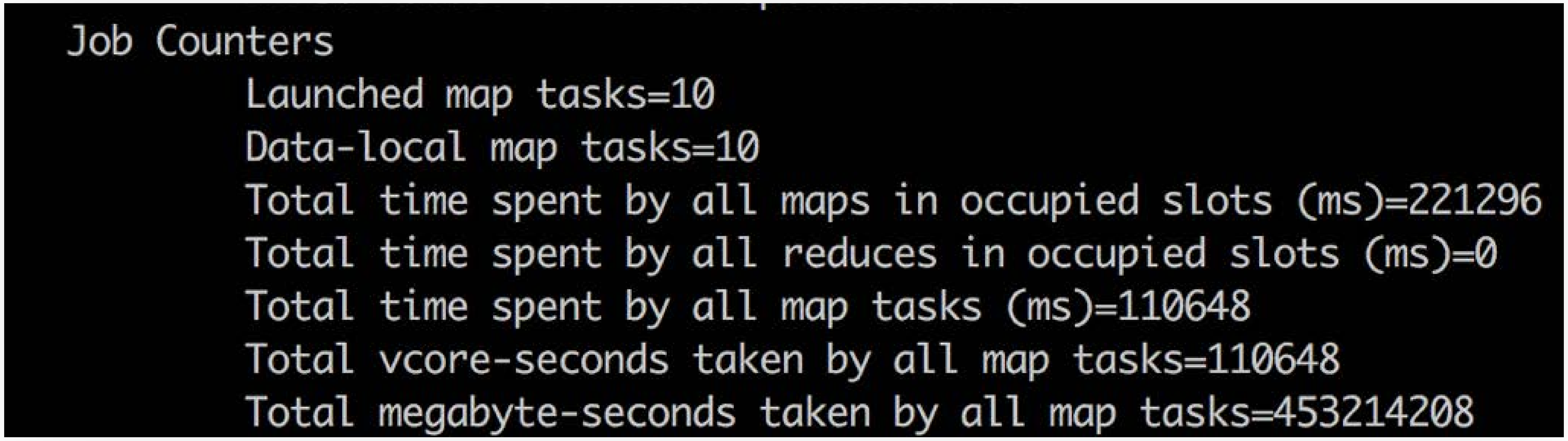
```
$ hdfs dfs -cat telecom-joins/part-00000 | head
South 0.0813626235113
South 0.141864254702
South 0.136587822758
South 0.278452077461
...

$ ./run_map_side_join.sh 9.90s user 0.37s system 34% cpu
29.383 total
```

HDFS read



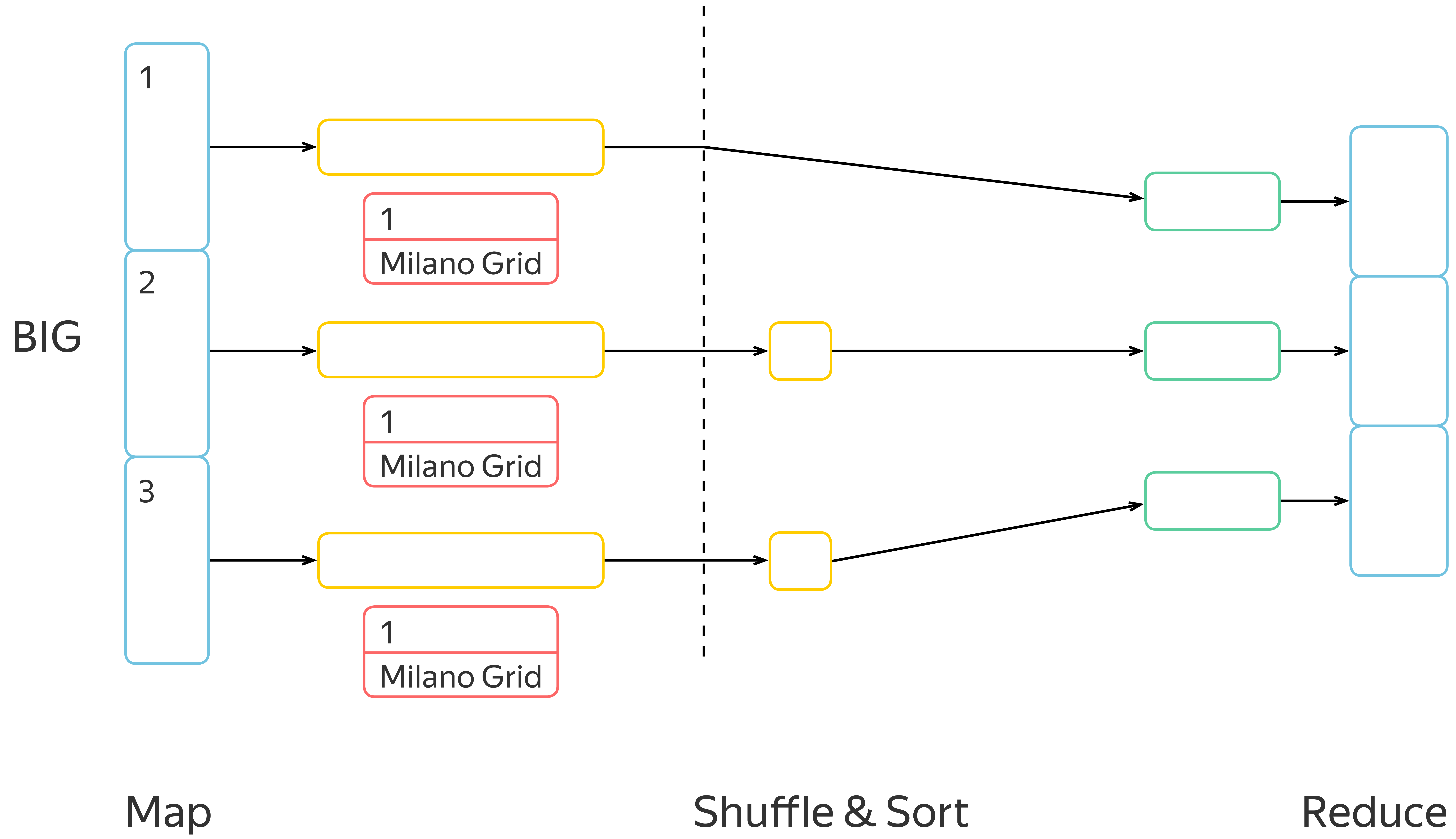
```
Job Counters
        Launched map tasks=10
        Data-local map tasks=10
        Total time spent by all maps in occupied slots (ms)=311034
        Total time spent by all reduces in occupied slots (ms)=0
        Total time spent by all map tasks (ms)=155517
        Total vcore-seconds taken by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

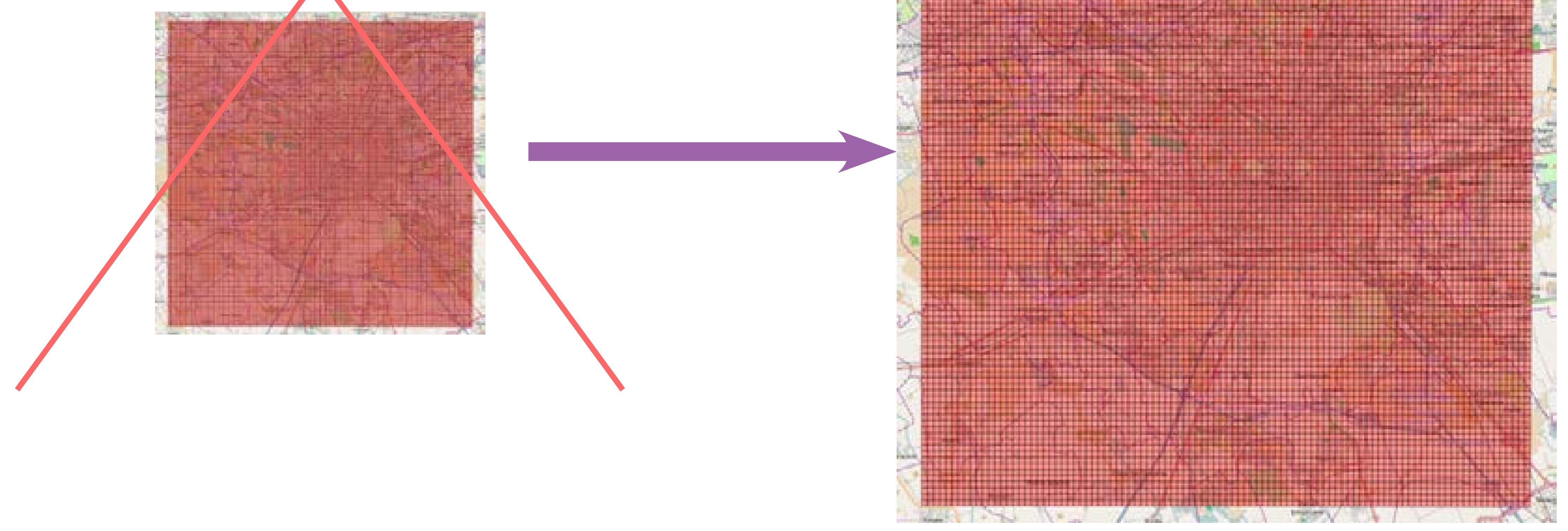
Map-Side Join



BIG

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

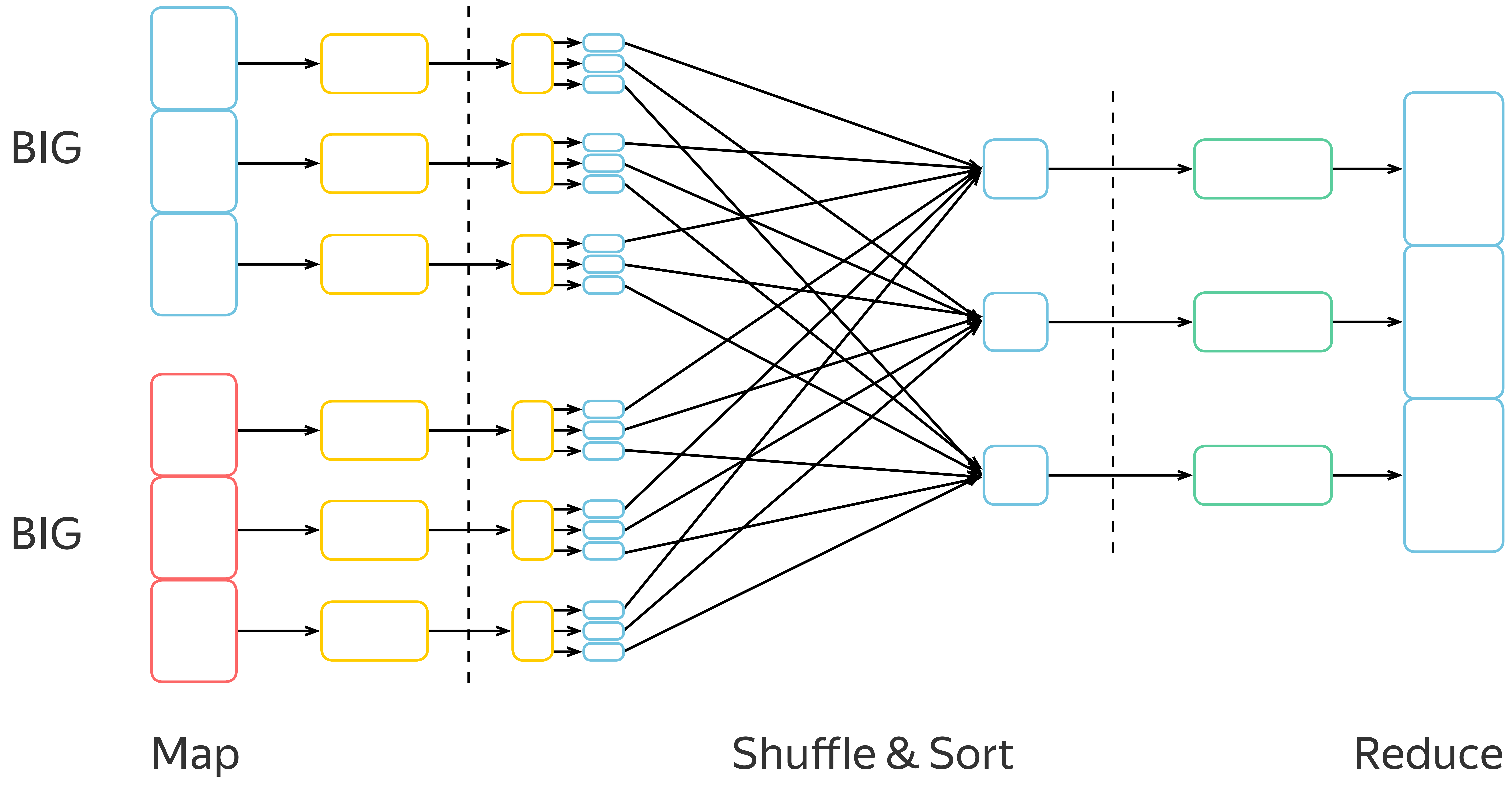
~~small~~ BIG



```
1 1383260400000 0 0.08136262351125882
1 1383260400000 39 0.14186425470242922
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



```
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")
```

```
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

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

```
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                                   string
2 grid South
3 grid South

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

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

1002	logs	0.0162920020569
1002	logs	0.0203572254966
1002	grid	South
1007	grid	South
1007	logs	0.0386839804552
1007	logs	0.0253373398645

[🟥, 🟦, 🟥, 🟥, 🟥]

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

1002	logs	0.0162920020569
1002	logs	0.0203572254966
1002	grid	South
1007	grid	South
1007	logs	0.0386839804552
1007	logs	0.0253373398645

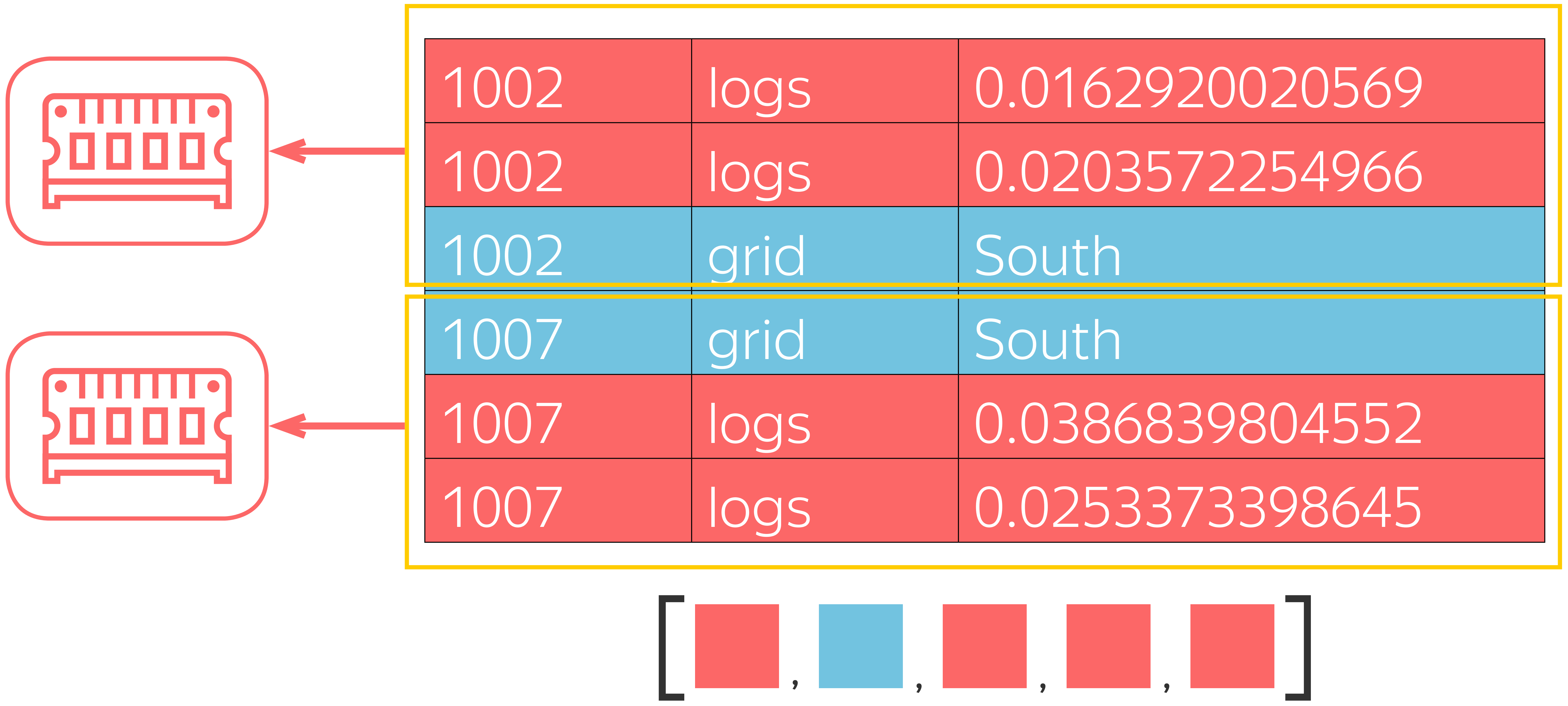
[🟥, 🟦, 🟥, 🟥, 🟥]

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

1002	logs	0.0162920020569
1002	logs	0.0203572254966
1002	grid	South
1007	grid	South
1007	logs	0.0386839804552
1007	logs	0.0253373398645

[🟥, 🟦, 🟥, 🟥, 🟥]

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

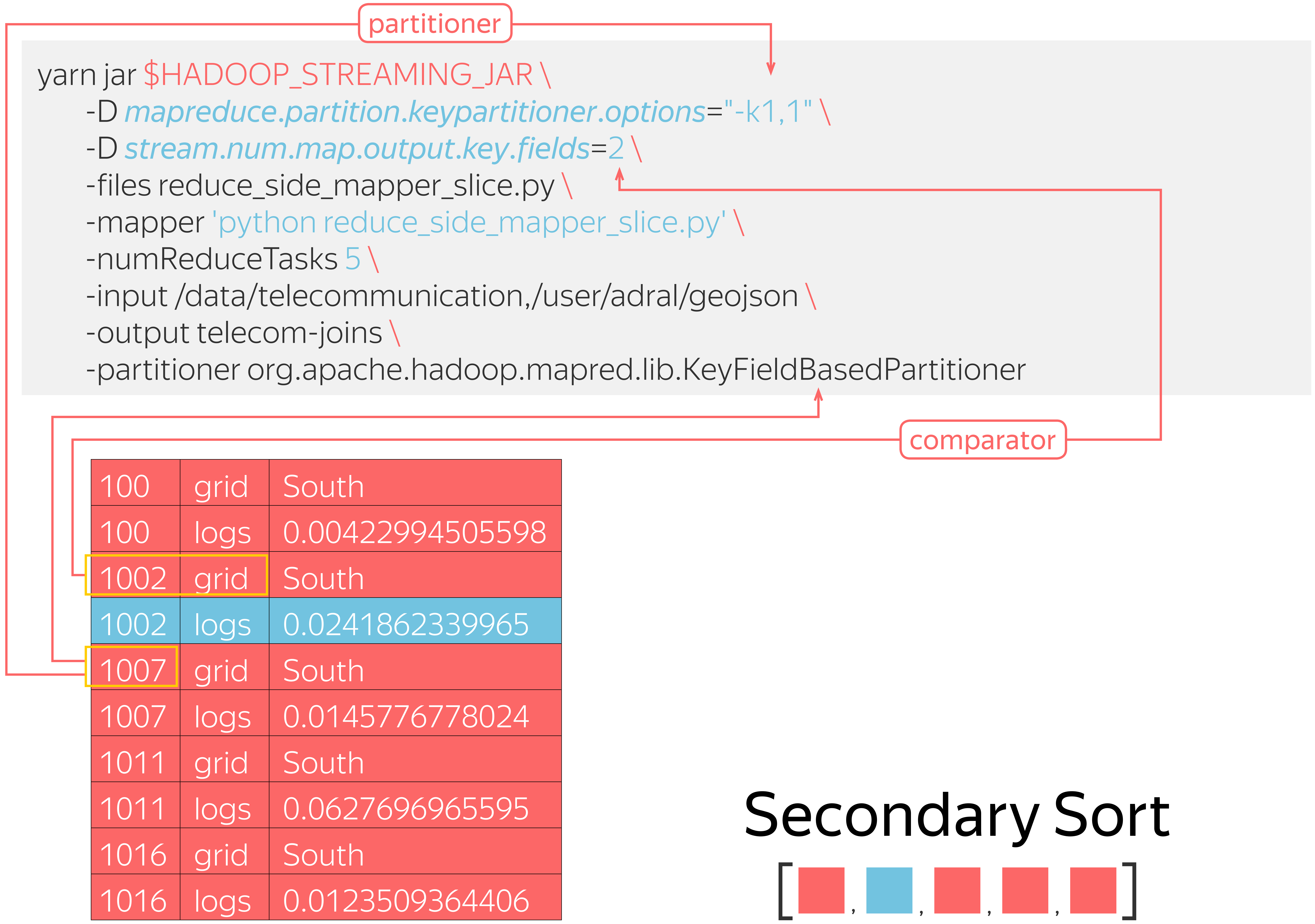


1002	logs	0.0162920020569
1002	logs	0.0203572254966
1002	grid	South
1007	grid	South
1007	logs	0.0386839804552
1007	logs	0.0253373398645

[🟥, 🟦, 🟥, 🟥, 🟥]

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

[🟥, 🟦, 🟥, 🟥, 🟥]

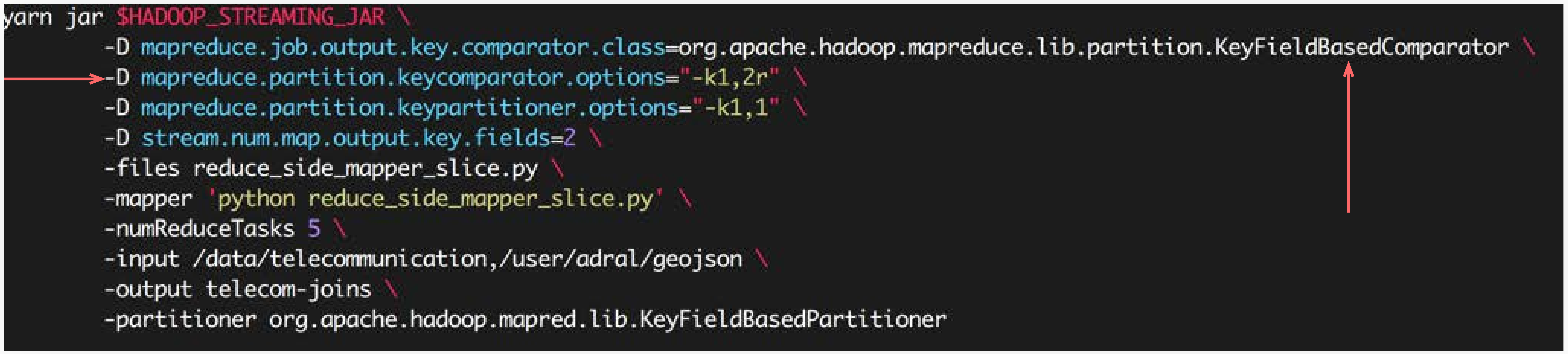


```
yarn jar $HADOOP_STREAMING_JAR \
    -D mapreduce.partition.keypartitioner.options="-k1,1" \
    -D stream.num.map.output.key.fields=2 \
    -files reduce_side_mapper_slice.py \
    -mapper 'python reduce_side_mapper_slice.py' \
    -numReduceTasks 5 \
    -input /data/telecommunication,/user/adral/geojson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

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

Secondary Sort

[🟥, 🟦, 🟥, 🟥, 🟥]



```
yarn jar $HADOOP_STREAMING_JAR \
        -D mapreduce.job.output.key.comparator.class=org.apache.hadoop.mapreduce.lib.partition.KeyFieldBasedComparator \
        -D mapreduce.partition.keycomparator.options="-k1,2r" \
        -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
```

9996	logs	0.149333295147
9996	grid	North
9991	logs	0.330465627227
9991	grid	North
9987	logs	0.0296826530265
9987	grid	North
9982	logs	0.262932749854
9982	grid	North
998	logs	0.0881801546604
998	grid	South

[🟥, 🟥, 🟥, 🟦, 🟥]

```
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

```python
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 != "grid":
    print(current_grid, grid_location, grid_load, sep="\t")
```

Summary

Summary

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

Summary

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

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