
# Lecture 8: Apache Spark examples

DAT470 / DIT065 Computational techniques for large-scale data

Matti Karppa

2024-04-22

# Estimating pi in Spark

```
import findspark
findspark.init()
```

Locates and sets up the Spark environment

```
import random
from pyspark import SparkContext
sc = SparkContext(master = 'local[16]')

NUM_SAMPLES = 100000000

def inside(p):
    x, y = random.random(), random.random()
    return x*x + y*y < 1

count = sc.parallelize(range(0,NUM_SAMPLES)) \
          .filter(inside).count()

print(f'Pi is roughly {4.0*count/NUM_SAMPLES}')
```

# Estimating pi in Spark

```
import findspark
findspark.init()
import random
from pyspark import SparkContext
sc = SparkContext(master = 'local[16]')

NUM_SAMPLES = 100000000

def inside(p):
    x, y = random.random(), random.random()
    return x*x + y*y < 1

count = sc.parallelize(range(0,NUM_SAMPLES)) \
          .filter(inside).count()

print(f'Pi is roughly {4.0*count/NUM_SAMPLES}')
```

Sets up the spark context: where and how the code is run

The `'local[16]'` bit is just boilerplate that means the code is run locally using 16 cores

# Estimating pi in Spark

UNIVERSITY OF GOTHENBURG
CHALMERS UNIVERSITY OF TECHNOLOGY

```
import findspark
findspark.init()
import random
from pyspark import SparkContext
sc = SparkContext(master = 'local[16]')

NUM_SAMPLES = 100000000
```

Sample *x* and *y*, and return whether it is within the unit circle (or the quarter thereof)
The input argument is ignored

```
def inside(p):
    x, y = random.random(), random.random()
    return x*x + y*y < 1

count = sc.parallelize(range(0,NUM_SAMPLES)) \
          .filter(inside).count()

print(f'Pi is roughly {4.0*count/NUM_SAMPLES}')
```

# Estimating pi in Spark

UNIVERSITY OF GOTHENBURG
CHALMERS UNIVERSITY OF TECHNOLOGY

```
import findspark
findspark.init()
import random
from pyspark import SparkContext
sc = SparkContext(master = 'local[16]')

NUM_SAMPLES = 100000000

def inside(p):
    x, y = random.random(), random.random()
    return x*x + y*y < 1

count = sc.parallelize(range(0,NUM_SAMPLES)) \
          .filter(inside).count()

print(f'Pi is roughly {4.0*count/NUM_SAMPLES}')
```

`range` returns an iterator which is used as a dummy argument for parallelization (`NUM_SAMPLES` arguments)

`filter` selects only those elements that satisfy the predicate

`count` works as if a reducer and counts the number of elements

**Note:** No array is created at any point

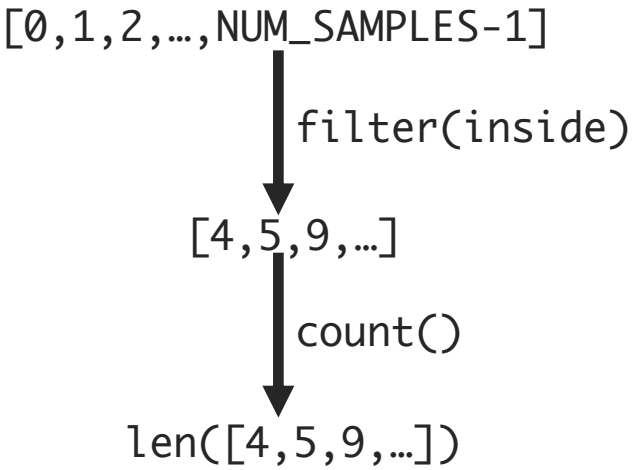
# The Spark programming model

```
sc.parallelize(range(0,NUM_SAMPLES)) \
  .filter(inside).count()
```

A simple corresponding example in serial Python

```
def odd(x):
    return x % 2
```

```
>>> a = [2,3,5,8,9]
>>> b = list(filter(odd,a))
>>> b
[3, 5, 9]
>>> len(b)
3
```




**Note:** arrays are not actually created for intermediate results

# The Spark programming model

```
>>> data = [0,1,2,3,4,5]
>>> distData = sc.parallelize(data)
>>> distData.reduce(lambda a, b: a + b)
15
```

- Spark programs are built around RDDs
- RDDs can be generated by the **driver program**, or they can be obtained from HDFS, or from files on disk, or from other RDDs
- RDDs distribute the data on the nodes, and contain the necessary information to reconstruct them: this makes them fault-tolerant, and effective
- Due to their distributed nature, RDDs can naturally be processed in parallel

# Finding duplicates in PySpark

```
import findspark
findspark.init()
from pyspark import SparkContext
sc = SparkContext(master = 'local[4]')

distFile = sc.textFile('test.data')

counts = distFile.map(lambda l: l.split(',')) \
                 .map(lambda t: (t[0],1)) \
                 .reduceByKey(lambda a,b: a+b) \
                 .filter(lambda t: t[1] > 1)
cc = counts.collect()
print(cc)
```

# Finding duplicates in PySpark

```
distFile = sc.textFile('test.data')
```

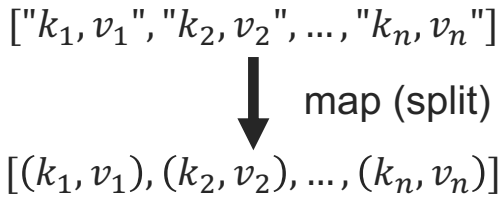
Construct an RDD from a file on the disk

$["k_1, v_1", "k_2, v_2", \ldots, "k_n, v_n"]$

# Finding duplicates in PySpark

Split each key,value string into a (key,value) tuple

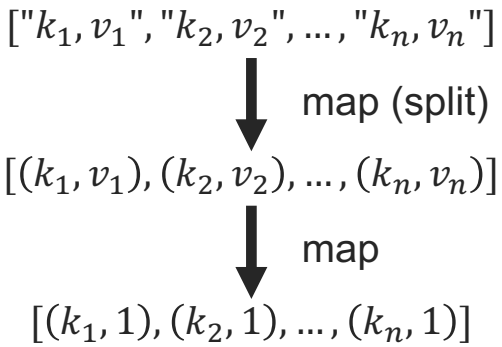
```
distFile.map(lambda l: l.split(',')) \
        .map(lambda t: (t[0],1)) \
        .reduceByKey(lambda a,b: a+b) \
        .filter(lambda t: t[1] > 1)
```



# Finding duplicates in PySpark

Convert the tuples into (key,1) tuples

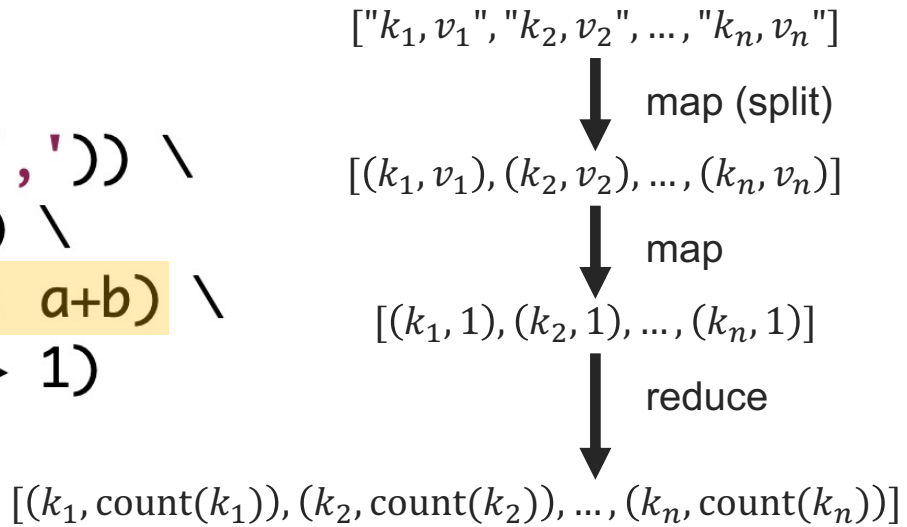
```
distFile.map(lambda l: l.split(',')) \
        .map(lambda t: (t[0],1)) \
        .reduceByKey(lambda a,b: a+b) \
        .filter(lambda t: t[1] > 1)
```



# Finding duplicates in PySpark

Collect tuples by key and reduce

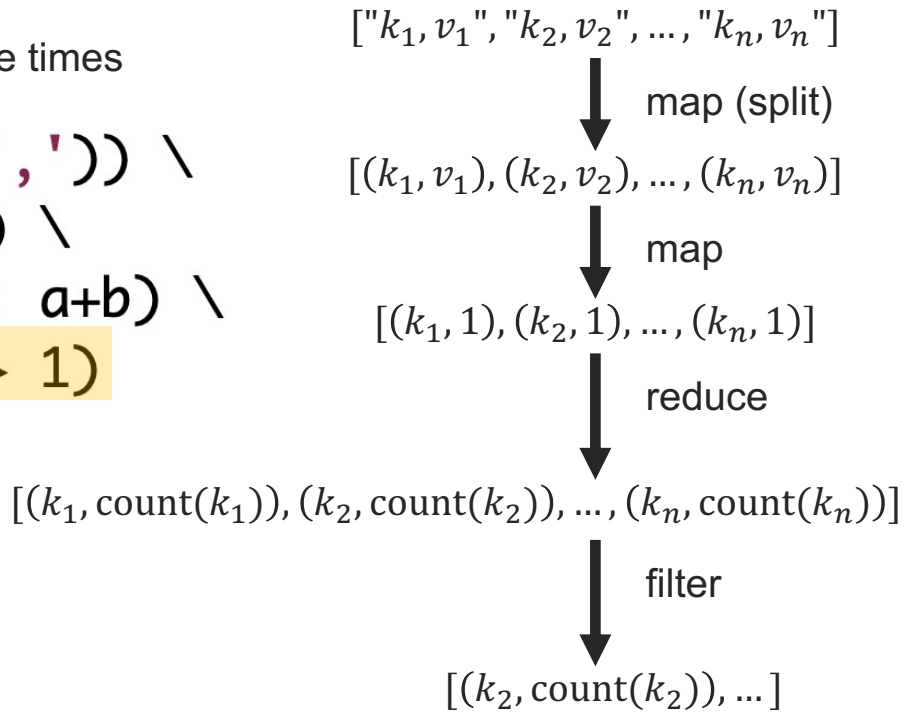
```
distFile.map(lambda l: l.split(',')) \
        .map(lambda t: (t[0],1)) \
        .reduceByKey(lambda a,b: a+b) \
        .filter(lambda t: t[1] > 1)
```



# Finding duplicates in PySpark

Filter only those tuples that occur multiple times

```
distFile.map(lambda l: l.split(',')) \
        .map(lambda t: (t[0],1)) \
        .reduceByKey(lambda a,b: a+b) \
        .filter(lambda t: t[1] > 1)
```



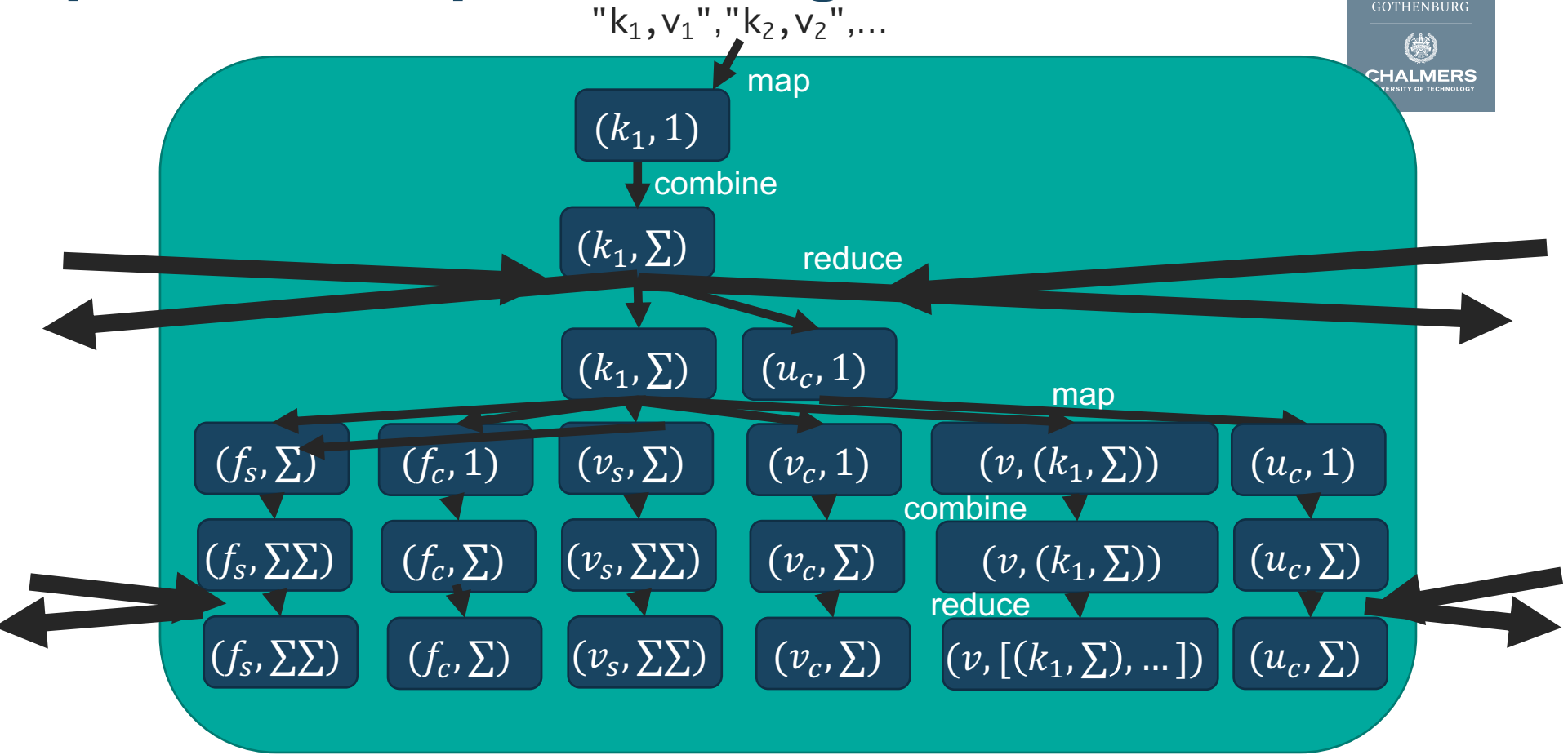
UNIVERSITY OF GOTHENBURG
CHALMERS UNIVERSITY OF TECHNOLOGY

# How do we get this?

```
There are 10 unique keys in the RDD of which 6 appear more than 20 times (sum = 517).
The following keys appear over 100 times (sum = 376).
[('REP', 104), ('LAREM', 272)]
```

- We need to count the number of unique keys
- We need to count the number of occurrences of each key
- We need to filter the number of frequently occurred keys
- We need to also filter the number of very frequently occurred keys
- …and we need to track the most frequent items and store them in a list

# A potential MapReduce algorithm



# Explanation of the algorithm

1. Map each input line into (key,1) tuples
2. Combine such tuples into (key,sum) tuples
3. Reduce such tuples into (key,sum) tuples, and additionally emit a $(u_c, 1)$ tuple for each key for counting unique keys
   - The special key $u_c$ tracks the count of unique keys
   - We now have information about the number of occurrence of each key as (key,sum) tuples
4. In the second step, map the tuples such that $(u_c, 1)$ tuples are preserved, and for other (key,sum) tuples, yield a $(f_c, 1)$ and a $(f_s, \text{sum})$ tuple if they exceed frequency threshold to count the number and sum of frequent elements, and $(v_c, 1)$, $(v_s, \text{sum})$, and $(v, (\text{key}, \text{sum}))$ tuples to count the number and sum of very frequent elements, plus record the actual key-sum pairs
5. Combine the values by summing up $(u_c, 1)$, $(f_c, 1)$, $(v_c, 1)$, $(f_s, \text{sum})$, and $(v_c, \text{sum})$ tuples, and joining the $(v, (\text{key}, \text{sum}))$ tuples as a list
6. Reduce the values by summing up $u_c$, $f_c$, $f_s$, $v_c$, $v_s$ tuples and joining the $v$ tuples as a list
7. We now have all we need to print the description

# Explanation of the algorithm

- Although the algorithm gets everything done, it is long, unwieldy, and ugly
- In such cases, data potentially needs to be duplicated to nodes, and messages passed through the interconnect because the nodes have no access to data
- Same values need to be potentially recomputed
- Spark provides a more flexible interface for such operations

# Potential PySpark implementation

```
import findspark
findspark.init()
from pyspark import SparkContext
sc = SparkContext(master = 'local[4]')
distFile = sc.textFile('test2.data')
keyTuples = distFile.map(lambda l: l.split(',')) \
                    .map(lambda t: (t[0],1))
countsPerKey = keyTuples.reduceByKey(lambda a,b: a+b)
unique_keys_count = countsPerKey.count()
FREQUENT = 20
VERY_FREQUENT = 100
frequent_keys = countsPerKey.filter(lambda t: t[1] > FREQUENT)
frequent_keys_count = frequent_keys.count()
frequent_keys_sum = frequent_keys.values().sum()
very_frequent_keys = frequent_keys.filter(lambda t: t[1] > VERY_FREQUENT)
very_frequent_keys_sum = very_frequent_keys.values().sum()
vfk = very_frequent_keys.collect()
print(f'There are {unique_keys_count} unique keys in the RDD'
      f' of which {frequent_keys_count} appear more than '
      f'{FREQUENT} times (sum = {frequent_keys_sum}).')
print(f'The following keys appear over {VERY_FREQUENT} times '
      f'(sum = {very_frequent_keys_sum}).')
print(vfk)
```

Distribute text file contents to nodes

# Potential PySpark implementation

```
import findspark
findspark.init()
from pyspark import SparkContext
sc = SparkContext(master = 'local[4]')
distFile = sc.textFile('test2.data')
keyTuples = distFile.map(lambda l: l.split(',')) \
                    .map(lambda t: (t[0],1))
countsPerKey = keyTuples.reduceByKey(lambda a,b: a+b)
unique_keys_count = countsPerKey.count()
FREQUENT = 20
VERY_FREQUENT = 100
frequent_keys = countsPerKey.filter(lambda t: t[1] > FREQUENT)
frequent_keys_count = frequent_keys.count()
frequent_keys_sum = frequent_keys.values().sum()
very_frequent_keys = frequent_keys.filter(lambda t: t[1] > VERY_FREQUENT)
very_frequent_keys_sum = very_frequent_keys.values().sum()
vfk = very_frequent_keys.collect()
print(f'There are {unique_keys_count} unique keys in the RDD'
      f' of which {frequent_keys_count} appear more than '
      f'{FREQUENT} times (sum = {frequent_keys_sum}).')
print(f'The following keys appear over {VERY_FREQUENT} times '
      f'(sum = {very_frequent_keys_sum}).')
print(vfk)
```

Distribute text file contents to nodes

Map text lines to (key,1) pairs

# Potential PySpark implementation

```
import findspark
findspark.init()
from pyspark import SparkContext
sc = SparkContext(master = 'local[4]')
distFile = sc.textFile('test2.data')
keyTuples = distFile.map(lambda l: l.split(',')) \
                    .map(lambda t: (t[0],1))
countsPerKey = keyTuples.reduceByKey(lambda a,b: a+b)
unique_keys_count = countsPerKey.count()
FREQUENT = 20
VERY_FREQUENT = 100
frequent_keys = countsPerKey.filter(lambda t: t[1] > FREQUENT)
frequent_keys_count = frequent_keys.count()
frequent_keys_sum = frequent_keys.values().sum()
very_frequent_keys = frequent_keys.filter(lambda t: t[1] > VERY_FREQUENT)
very_frequent_keys_sum = very_frequent_keys.values().sum()
vfk = very_frequent_keys.collect()
print(f'There are {unique_keys_count} unique keys in the RDD'
      f' of which {frequent_keys_count} appear more than '
      f'{FREQUENT} times (sum = {frequent_keys_sum}).')
print(f'The following keys appear over {VERY_FREQUENT} times '
      f'(sum = {very_frequent_keys_sum}).')
print(vfk)
```

Distribute text file contents to nodes

Map text lines to (key,1) pairs

Reduce counts by summing, so the RDD contains map from key to the total count

# Potential PySpark implementation

```
import findspark
findspark.init()
from pyspark import SparkContext
sc = SparkContext(master = 'local[4]')
distFile = sc.textFile('test2.data')
keyTuples = distFile.map(lambda l: l.split(',')) \
                    .map(lambda t: (t[0],1))
countsPerKey = keyTuples.reduceByKey(lambda a,b: a+b)
unique_keys_count = countsPerKey.count()
FREQUENT = 20
VERY_FREQUENT = 100
frequent_keys = countsPerKey.filter(lambda t: t[1] > FREQUENT)
frequent_keys_count = frequent_keys.count()
frequent_keys_sum = frequent_keys.values().sum()
very_frequent_keys = frequent_keys.filter(lambda t: t[1] > VERY_FREQUENT)
very_frequent_keys_sum = very_frequent_keys.values().sum()
vfk = very_frequent_keys.collect()
print(f'There are {unique_keys_count} unique keys in the RDD'
      f' of which {frequent_keys_count} appear more than '
      f'{FREQUENT} times (sum = {frequent_keys_sum}).')
print(f'The following keys appear over {VERY_FREQUENT} times '
      f'(sum = {very_frequent_keys_sum}).')
print(vfk)
```

Distribute text file contents to nodes

Map text lines to (key,1) pairs

Reduce counts by summing, so the RDD contains map from key to the total count

Count the number of keys

# Potential PySpark implementation

```
import findspark
findspark.init()
from pyspark import SparkContext
sc = SparkContext(master = 'local[4]')
distFile = sc.textFile('test2.data')
keyTuples = distFile.map(lambda l: l.split(',')) \
                    .map(lambda t: (t[0],1))
countsPerKey = keyTuples.reduceByKey(lambda a,b: a+b)
unique_keys_count = countsPerKey.count()
FREQUENT = 20
VERY_FREQUENT = 100
frequent_keys = countsPerKey.filter(lambda t: t[1] > FREQUENT)
frequent_keys_count = frequent_keys.count()
frequent_keys_sum = frequent_keys.values().sum()
very_frequent_keys = frequent_keys.filter(lambda t: t[1] > VERY_FREQUENT)
very_frequent_keys_sum = very_frequent_keys.values().sum()
vfk = very_frequent_keys.collect()
print(f'There are {unique_keys_count} unique keys in the RDD'
      f' of which {frequent_keys_count} appear more than '
      f'{FREQUENT} times (sum = {frequent_keys_sum}).')
print(f'The following keys appear over {VERY_FREQUENT} times '
      f'(sum = {very_frequent_keys_sum}).')
print(vfk)
```

Distribute text file contents to nodes

Map text lines to (key,1) pairs

Reduce counts by summing, so the RDD contains map from key to the total count

Count the number of keys

Reuse the RDD and filter the frequent keys

# Potential PySpark implementation

```
import findspark
findspark.init()
from pyspark import SparkContext
sc = SparkContext(master = 'local[4]')
distFile = sc.textFile('test2.data')
keyTuples = distFile.map(lambda l: l.split(',')) \
                    .map(lambda t: (t[0],1))
countsPerKey = keyTuples.reduceByKey(lambda a,b: a+b)
unique_keys_count = countsPerKey.count()
FREQUENT = 20
VERY_FREQUENT = 100
frequent_keys = countsPerKey.filter(lambda t: t[1] > FREQUENT)
frequent_keys_count = frequent_keys.count()
frequent_keys_sum = frequent_keys.values().sum()
very_frequent_keys = frequent_keys.filter(lambda t: t[1] > VERY_FREQUENT)
very_frequent_keys_sum = very_frequent_keys.values().sum()
vfk = very_frequent_keys.collect()
print(f'There are {unique_keys_count} unique keys in the RDD'
      f' of which {frequent_keys_count} appear more than '
      f'{FREQUENT} times (sum = {frequent_keys_sum}).')
print(f'The following keys appear over {VERY_FREQUENT} times '
      f'(sum = {very_frequent_keys_sum}).')
print(vfk)
```

Distribute text file contents to nodes

Map text lines to (key,1) pairs

Reduce counts by summing, so the RDD contains map from key to the total count

Count the number of keys

Reuse the RDD and filter the frequent keys

Use the same RDD for two different operations: count and sum

# Potential PySpark implementation

```
import findspark
findspark.init()
from pyspark import SparkContext
sc = SparkContext(master = 'local[4]')
distFile = sc.textFile('test2.data')
keyTuples = distFile.map(lambda l: l.split(',')) \
                    .map(lambda t: (t[0],1))
countsPerKey = keyTuples.reduceByKey(lambda a,b: a+b)
unique_keys_count = countsPerKey.count()
FREQUENT = 20
VERY_FREQUENT = 100
frequent_keys = countsPerKey.filter(lambda t: t[1] > FREQUENT)
frequent_keys_count = frequent_keys.count()
frequent_keys_sum = frequent_keys.values().sum()
very_frequent_keys = frequent_keys.filter(lambda t: t[1] > VERY_FREQUENT)
very_frequent_keys_sum = very_frequent_keys.values().sum()
vfk = very_frequent_keys.collect()
print(f'There are {unique_keys_count} unique keys in the RDD'
      f' of which {frequent_keys_count} appear more than '
      f'{FREQUENT} times (sum = {frequent_keys_sum}).')
print(f'The following keys appear over {VERY_FREQUENT} times '
      f'(sum = {very_frequent_keys_sum}).')
print(vfk)
```

Distribute text file contents to nodes

Map text lines to (key,1) pairs

Reduce counts by summing, so the RDD contains map from key to the total count

Count the number of keys

Reuse the RDD and filter the frequent keys

Use the same RDD for two different operations: count and sum

Do the same with very frequent keys

# Potential PySpark implementation

```
import findspark
findspark.init()
from pyspark import SparkContext
sc = SparkContext(master = 'local[4]')
distFile = sc.textFile('test2.data')
keyTuples = distFile.map(lambda l: l.split(',')) \
                    .map(lambda t: (t[0],1))
countsPerKey = keyTuples.reduceByKey(lambda a,b: a+b)
unique_keys_count = countsPerKey.count()
FREQUENT = 20
VERY_FREQUENT = 100
frequent_keys = countsPerKey.filter(lambda t: t[1] > FREQUENT)
frequent_keys_count = frequent_keys.count()
frequent_keys_sum = frequent_keys.values().sum()
very_frequent_keys = frequent_keys.filter(lambda t: t[1] > VERY_FREQUENT)
very_frequent_keys_sum = very_frequent_keys.values().sum()
vfk = very_frequent_keys.collect()
print(f'There are {unique_keys_count} unique keys in the RDD'
      f' of which {frequent_keys_count} appear more than '
      f'{FREQUENT} times (sum = {frequent_keys_sum}).')
print(f'The following keys appear over {VERY_FREQUENT} times '
      f'(sum = {very_frequent_keys_sum}).')
print(vfk)
```

Distribute text file contents to nodes

Map text lines to (key,1) pairs

Reduce counts by summing, so the RDD contains map from key to the total count

Count the number of keys

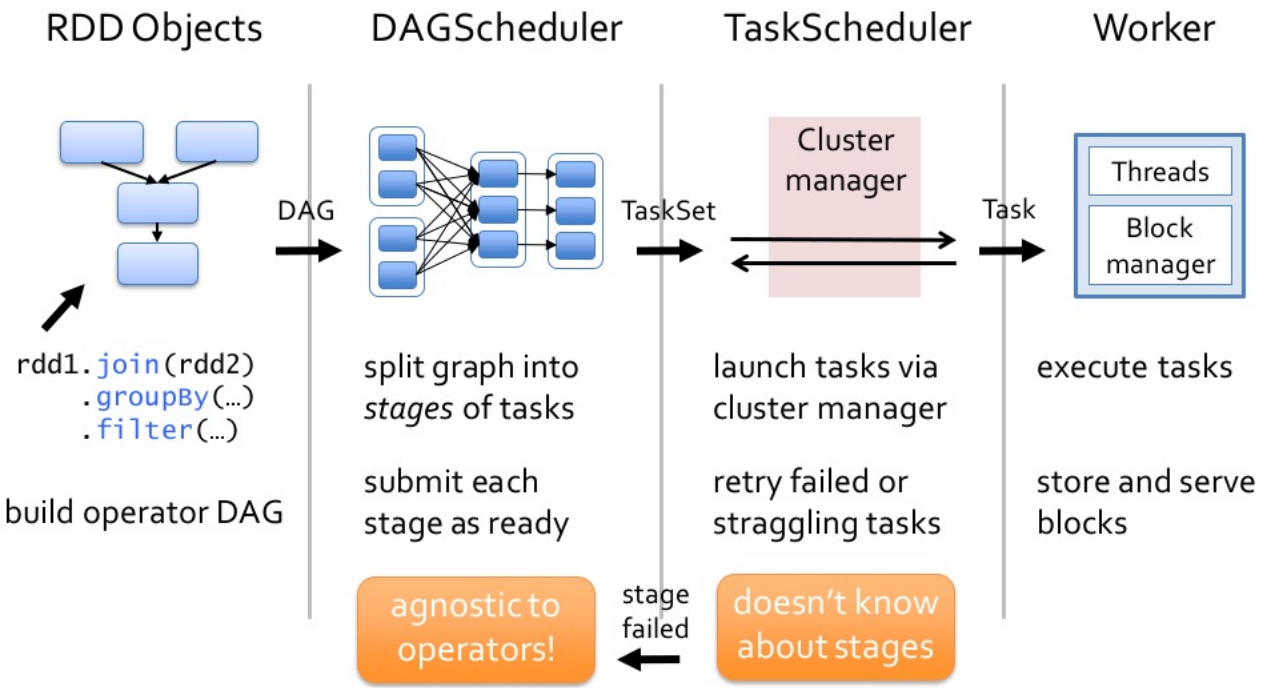
Reuse the RDD and filter the frequent keys

Use the same RDD for two different operations: count and sum

Do the same with very frequent keys

**Collect** the values: copy the values from the distributed filesystem onto the driver node

# Apache Spark under the hood



UNIVERSITY OF GOTHENBURG
CHALMERS UNIVERSITY OF TECHNOLOGY

# Apache Spark under the hood

- Program flow is managed as a Directed Acyclic Graph (DAG) where nodes are RDDs and edges are transformations
- The transformations are lazy and computation is triggered by actions
- When computation, or a **job**, is requested, the DAG is divided into **stages** of **tasks**; the DAG scheduler tries to optimize the operators in this graph and to exploit data locality, which can lead to many independent operations scheduled in the same stage
- The stages of tasks are passed to the task scheduler that launces them using the cluster manager; the task scheduler is unaware of dependencies among the stages
- Worker executes the task on the node; it only knows of the code passed to it