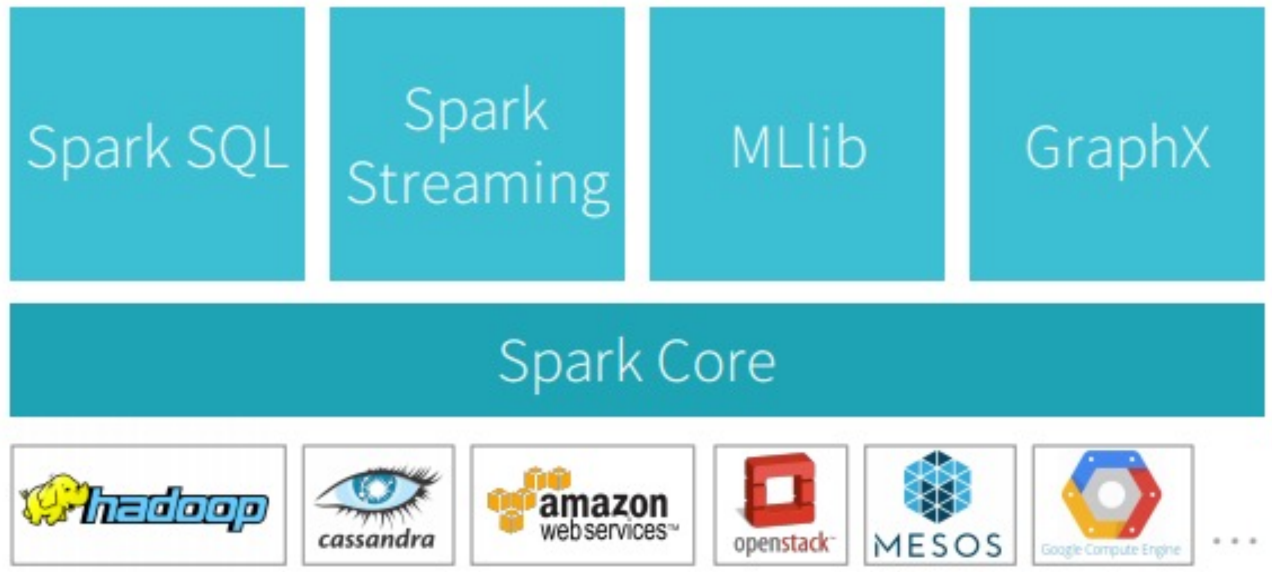
# CSCE 5300

# GraphX and GraphFrames

# Apache Spark Engine

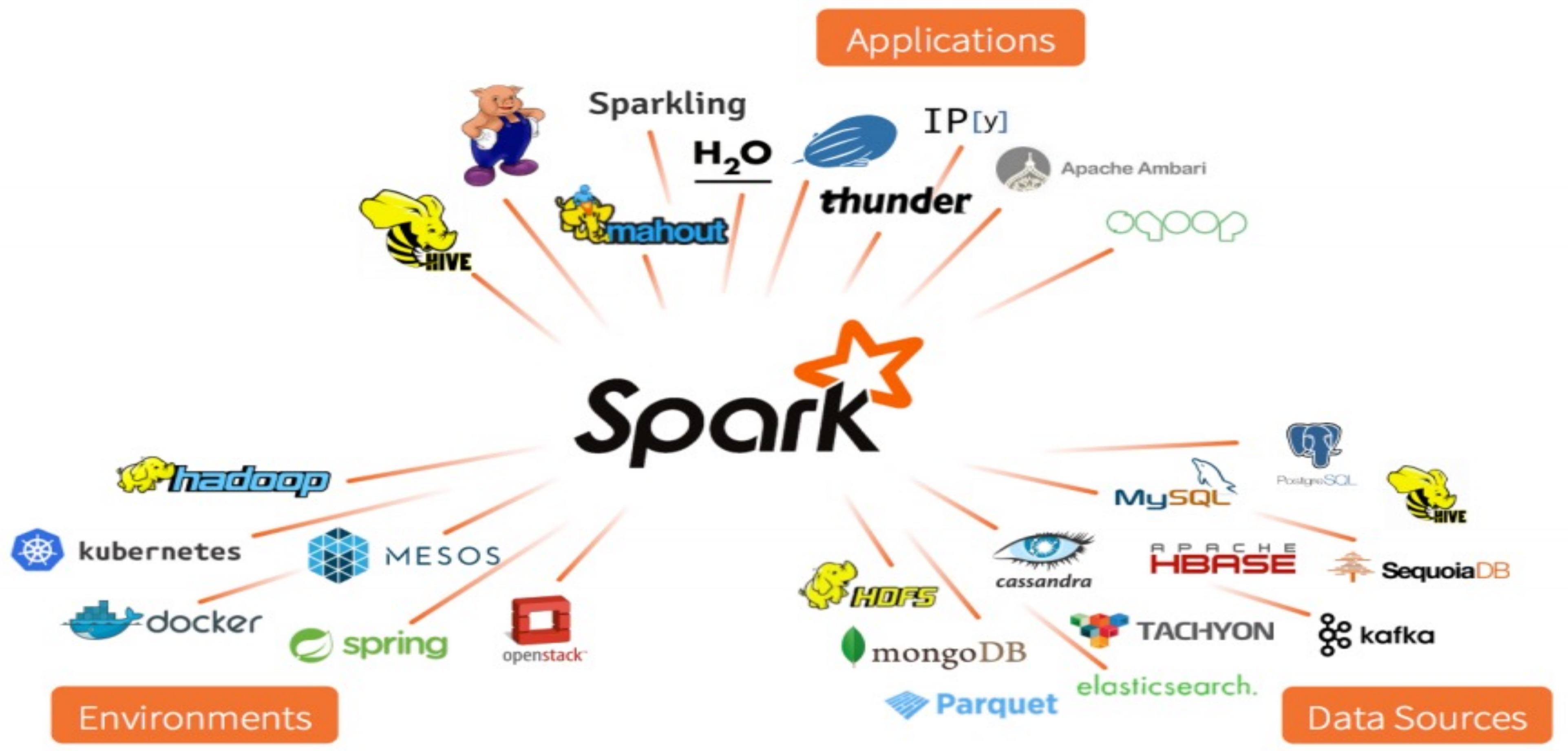


Scale out, fault tolerant

Python, Java, Scala, and R APIs

Standard libraries

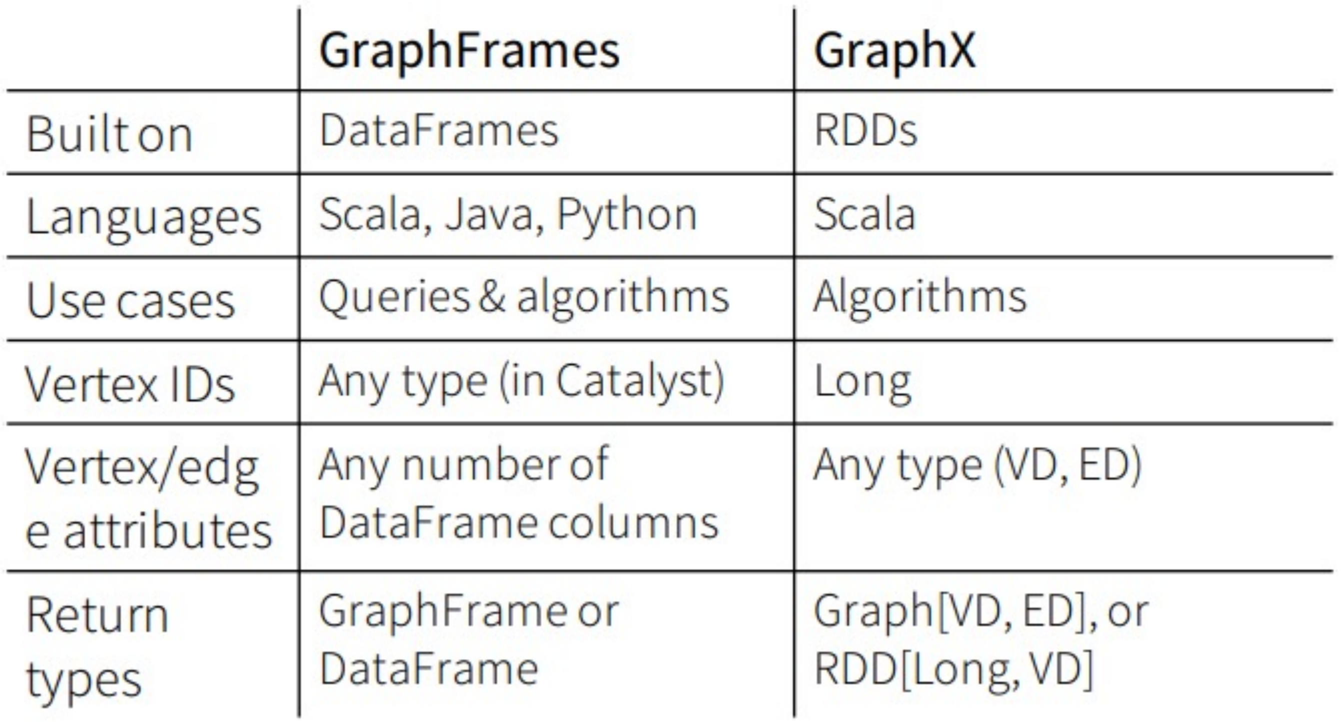
# Open source Eco-System



## Graphs

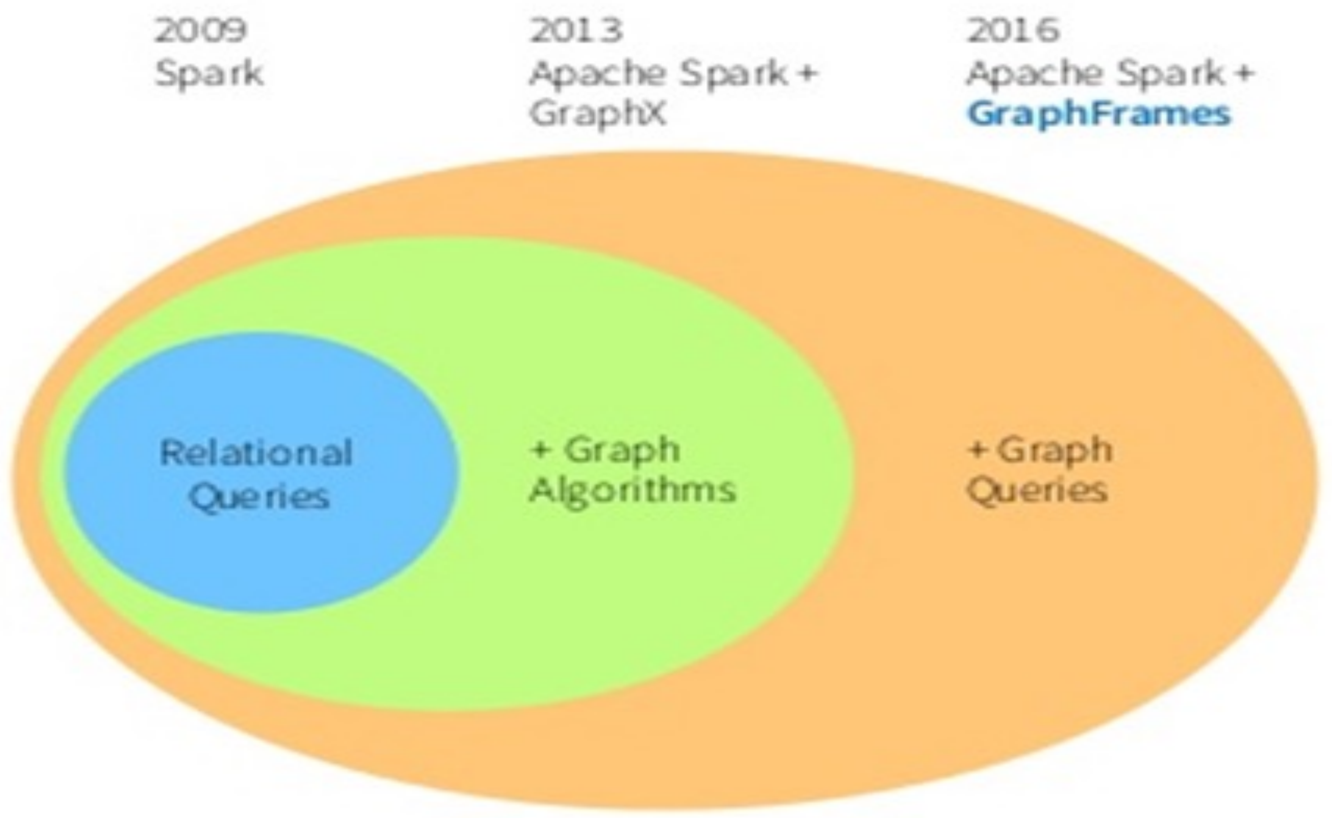
- *GraphX is to RDDs as GraphFrames are to DataFrames*
- GraphFrames represent graphs: vertices (e.g., users) and edges (e.g., relationships between users).
- GraphFrames are based upon Spark DataFrames
- GraphX are based upon RDDs

# GraphX vs GraphFrames



| | GraphFrames | GraphX |
|---|---|---|
| Built on | DataFrames | RDDs |
| Languages | Scala, Java, Python | Scala |
| Use cases | Queries & algorithms | Algorithms |
| Vertex IDs | Any type (in Catalyst) | Long |
| Vertex/edge attributes | Any number of DataFrame columns | Any type (VD, ED) |
| Return types | GraphFrame or DataFrame | Graph[VD, ED], or RDD[Long, VD] |

## Continued ..


2009
Spark
2013
Apache Spark +
GraphX
2016
Apache Spark +
GraphFrames
Relational
Queries
+ Graph
Algorithms
+ Graph
Queries

## GraphFrames Configuration

- Download Spark (Latest Version)

  https://spark.apache.org/downloads.html

From Command Line:

- Go to bin
  - Spark-shell –packages graphframes:graphframes:0.5.0-spark2.3-s_2.11

Add jars

  - From project structure add the downloaded jars in intelliJ

## Load Pyspark GraphFrames

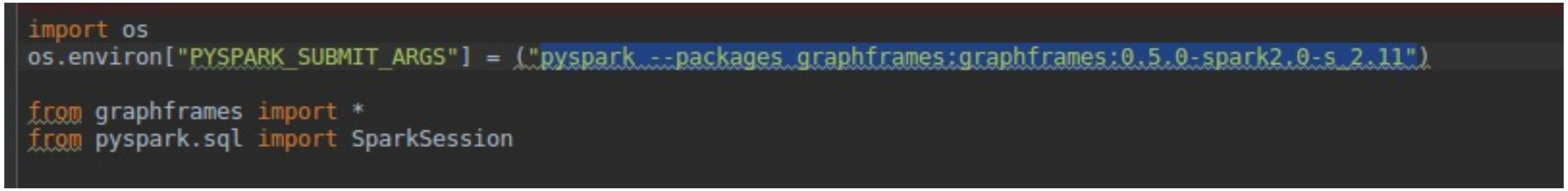
**From PyCharm:**

```
import os
os.environ["PYSPARK_SUBMIT_ARGS"] = (
"--packages graphframes:graphframes:0.5.0-spark2.0-s_2.11 pyspark-shell")
```

**From Command Line:**

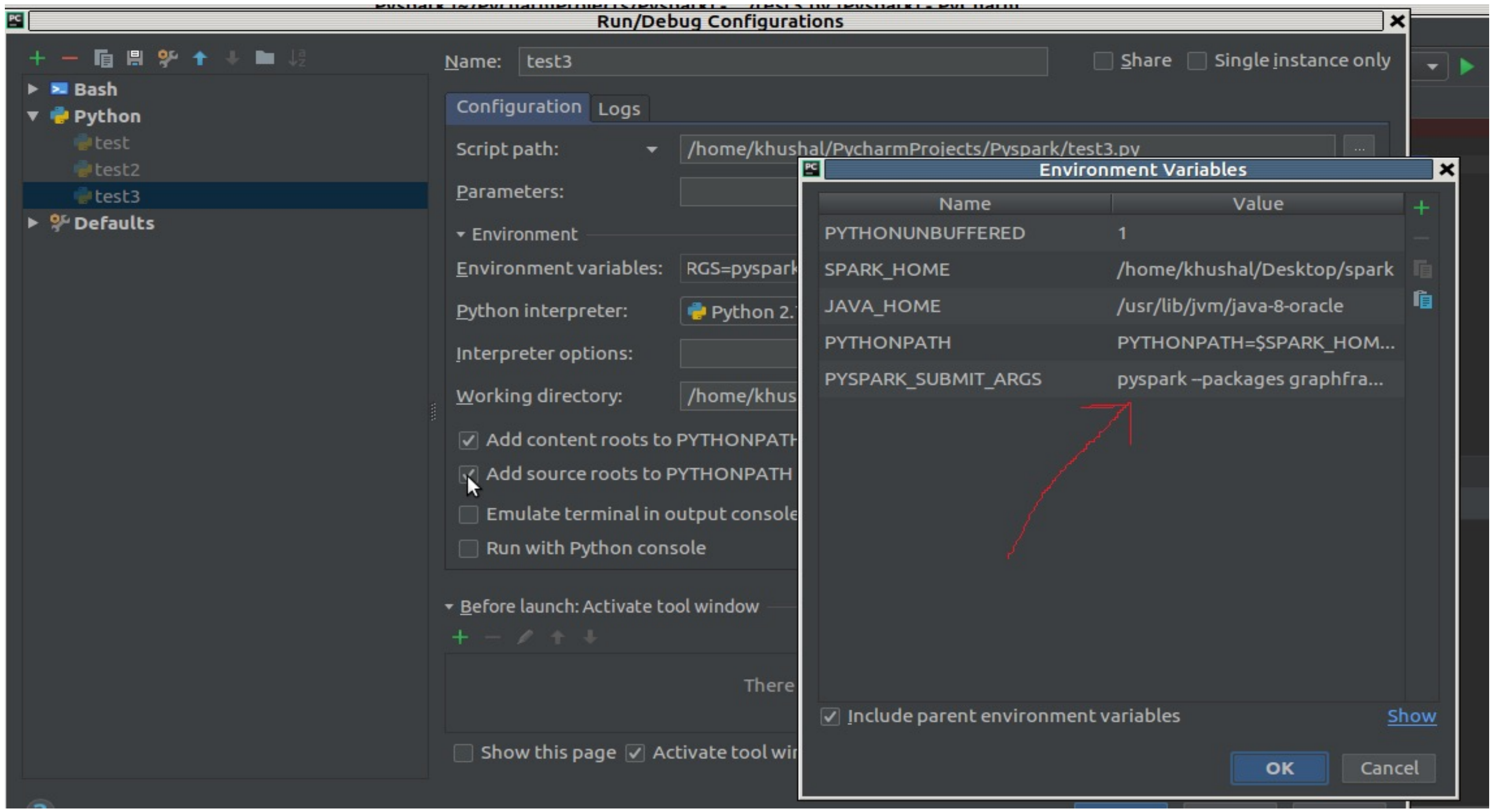
```
Pyspark --packages graphframes:graphframes:0.5.0-spark2.0-s_2.11 pyspark-shell
```

## Load in PyCharm or Edit Configuration Settings



```
import os
os.environ["PYSPARK_SUBMIT_ARGS"] = ("pyspark --packages graphframes:graphframes:0.5.0-spark2.0-s_2.11")

from graphframes import *
from pyspark.sql import SparkSession
```

# Continued ..



Run/Debug Configurations

Name: test3 — Share — Single instance only

Bash
Python
test
test2
test3
Defaults

Configuration | Logs

Script path: /home/khushal/PycharmProjects/Pyspark/test3.py
Parameters:
Environment
Environment variables: RGS=pyspark
Python interpreter: Python 2.
Interpreter options:
Working directory: /home/khus

☑ Add content roots to PYTHONPATH
☑ Add source roots to PYTHONPATH
☐ Emulate terminal in output console
☐ Run with Python console

Before launch: Activate tool window

☐ Show this page ☑ Activate tool wi

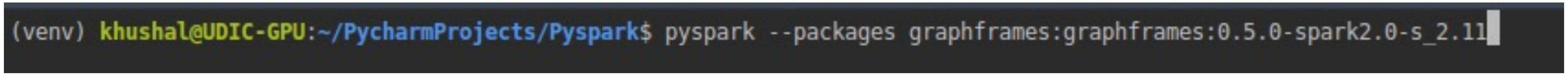
Environment Variables

| Name | Value |
|---|---|
| PYTHONUNBUFFERED | 1 |
| SPARK_HOME | /home/khushal/Desktop/spark |
| JAVA_HOME | /usr/lib/jvm/java-8-oracle |
| PYTHONPATH | PYTHONPATH=$SPARK_HOM... |
| PYSPARK_SUBMIT_ARGS | pyspark --packages graphfra... |

☑ Include parent environment variables — Show

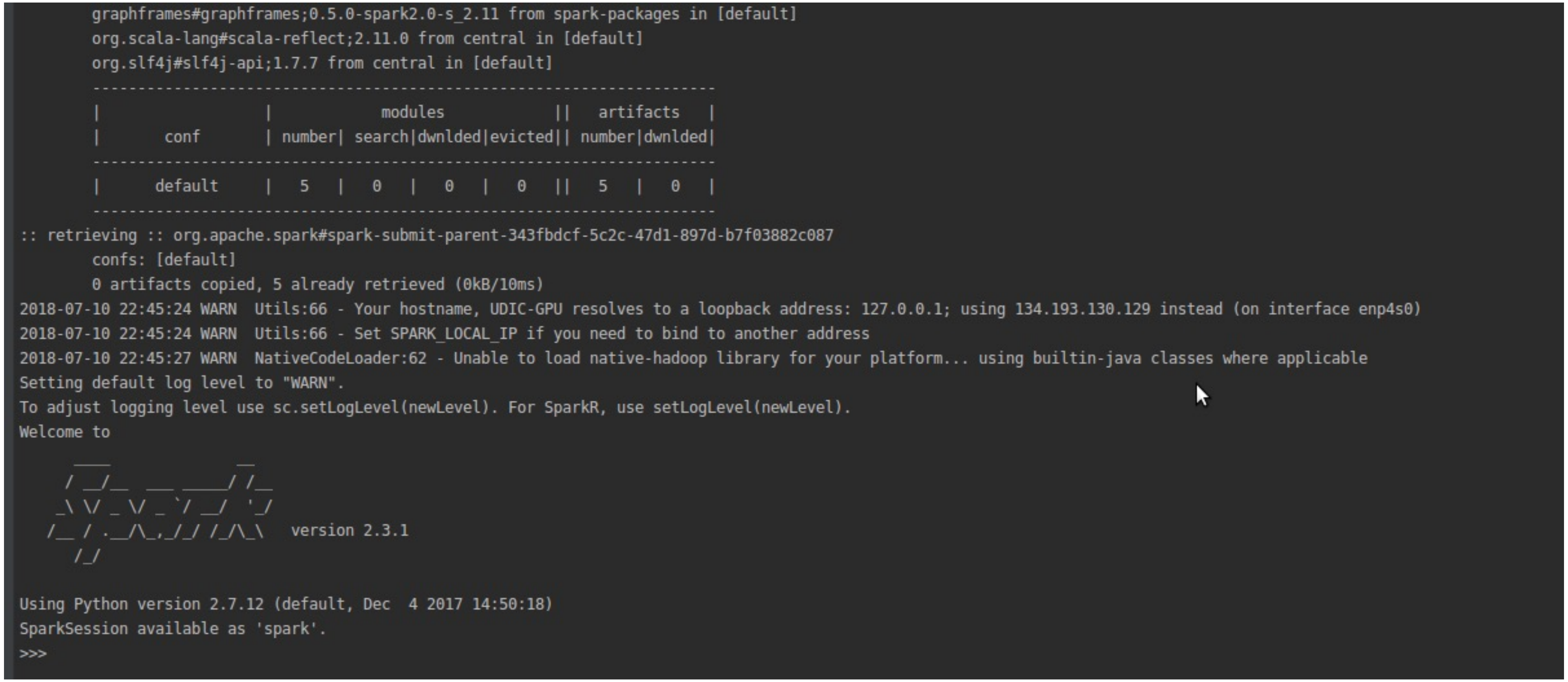
OK — Cancel

## Pyspark command line



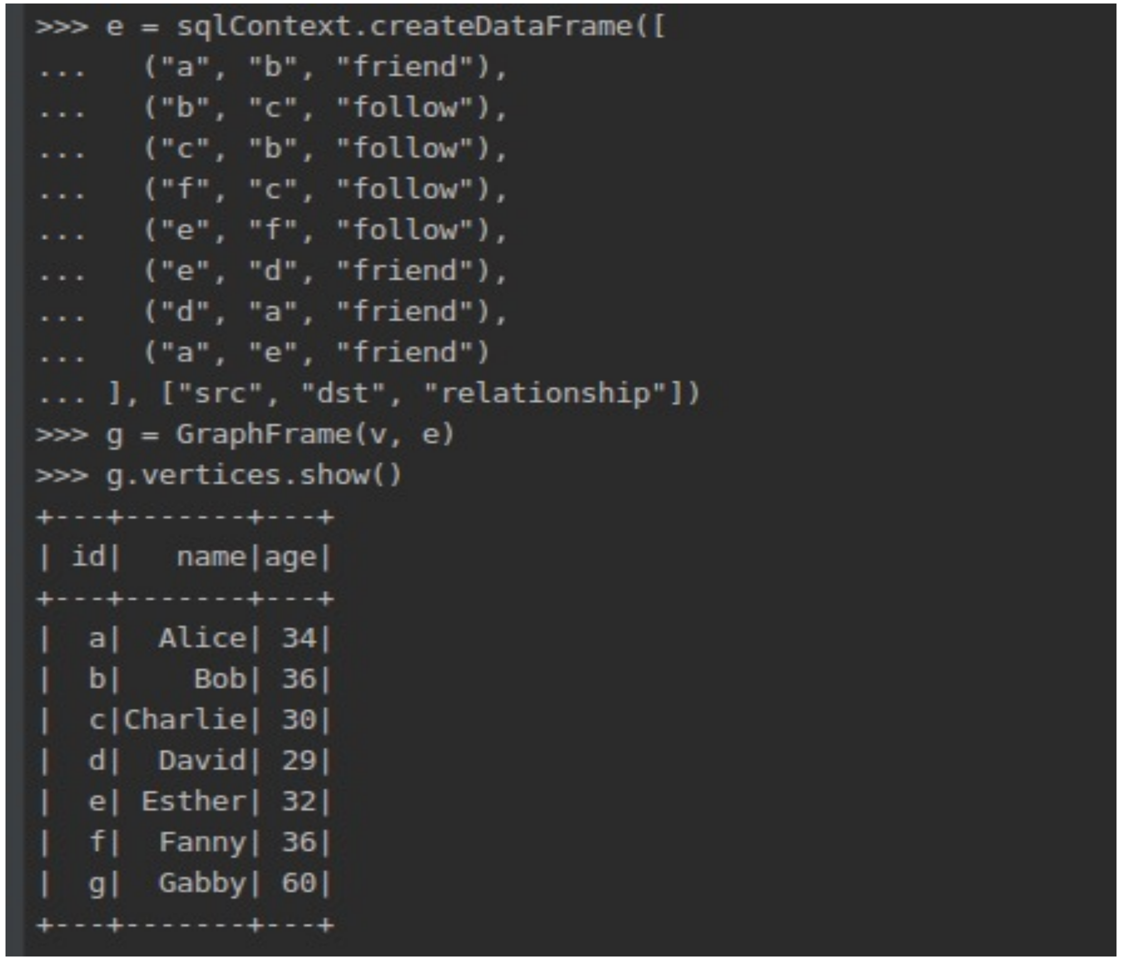
```
(venv) khushal@UDIC-GPU:~/PycharmProjects/Pyspark$ pyspark --packages graphframes:graphframes:0.5.0-spark2.0-s_2.11
```

# Pyspark Loaded with graphframes



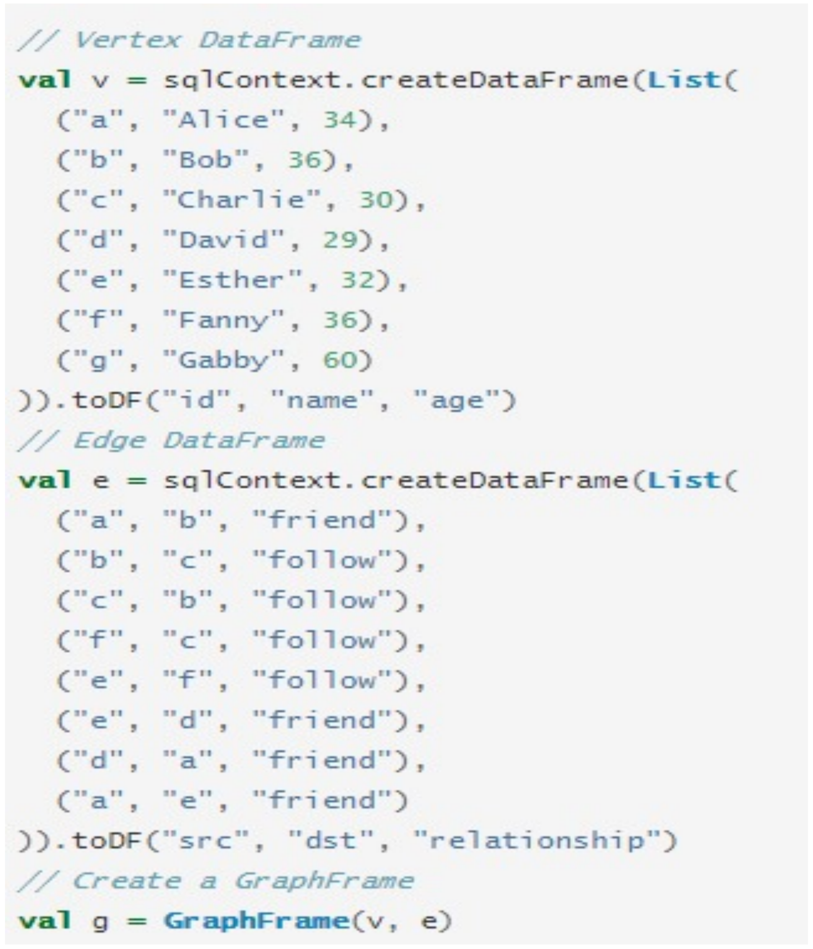
```
        graphframes#graphframes;0.5.0-spark2.0-s_2.11 from spark-packages in [default]
        org.scala-lang#scala-reflect;2.11.0 from central in [default]
        org.slf4j#slf4j-api;1.7.7 from central in [default]
        ---------------------------------------------------------------------
        |                  |            modules            ||   artifacts   |
        |       conf       | number| search|dwnlded|evicted|| number|dwnlded|
        ---------------------------------------------------------------------
        |      default     |   5   |   0   |   0   |   0   ||   5   |   0   |
        ---------------------------------------------------------------------
:: retrieving :: org.apache.spark#spark-submit-parent-343fbdcf-5c2c-47d1-897d-b7f03882c087
        confs: [default]
        0 artifacts copied, 5 already retrieved (0kB/10ms)
2018-07-10 22:45:24 WARN  Utils:66 - Your hostname, UDIC-GPU resolves to a loopback address: 127.0.0.1; using 134.193.130.129 instead (on interface enp4s0)
2018-07-10 22:45:24 WARN  Utils:66 - Set SPARK_LOCAL_IP if you need to bind to another address
2018-07-10 22:45:27 WARN  NativeCodeLoader:62 - Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.3.1
      /_/

Using Python version 2.7.12 (default, Dec  4 2017 14:50:18)
SparkSession available as 'spark'.
>>>
```

# Create Dataframe in Pyspark



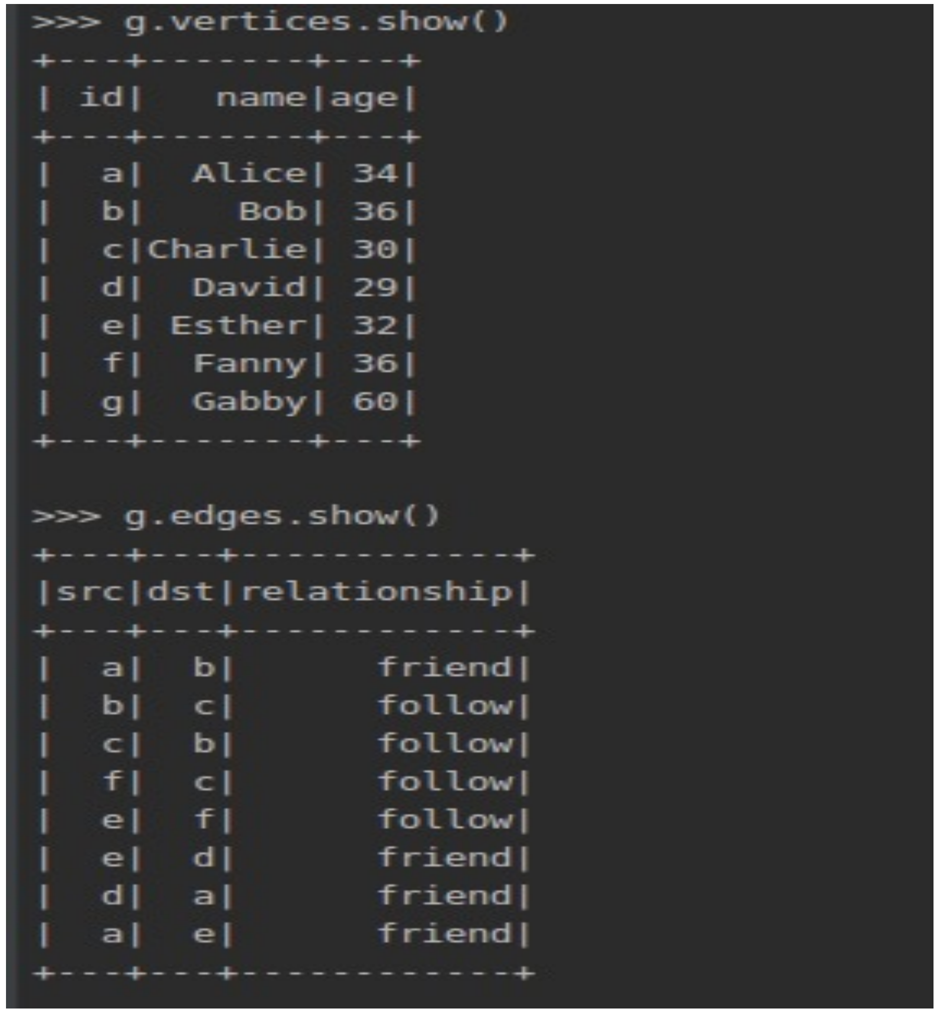
```
>>> e = sqlContext.createDataFrame([
...   ("a", "b", "friend"),
...   ("b", "c", "follow"),
...   ("c", "b", "follow"),
...   ("f", "c", "follow"),
...   ("e", "f", "follow"),
...   ("e", "d", "friend"),
...   ("d", "a", "friend"),
...   ("a", "e", "friend")
... ], ["src", "dst", "relationship"])
>>> g = GraphFrame(v, e)
>>> g.vertices.show()
+---+-------+---+
| id|   name|age|
+---+-------+---+
|  a|  Alice| 34|
|  b|    Bob| 36|
|  c|Charlie| 30|
|  d|  David| 29|
|  e| Esther| 32|
|  f|  Fanny| 36|
|  g|  Gabby| 60|
+---+-------+---+
```

## Create Dataframe in Scala



```
// Vertex DataFrame
val v = sqlContext.createDataFrame(List(
  ("a", "Alice", 34),
  ("b", "Bob", 36),
  ("c", "Charlie", 30),
  ("d", "David", 29),
  ("e", "Esther", 32),
  ("f", "Fanny", 36),
  ("g", "Gabby", 60)
)).toDF("id", "name", "age")
// Edge DataFrame
val e = sqlContext.createDataFrame(List(
  ("a", "b", "friend"),
  ("b", "c", "follow"),
  ("c", "b", "follow"),
  ("f", "c", "follow"),
  ("e", "f", "follow"),
  ("e", "d", "friend"),
  ("d", "a", "friend"),
  ("a", "e", "friend")
)).toDF("src", "dst", "relationship")
// Create a GraphFrame
val g = GraphFrame(v, e)
```

## GraphFrames Output

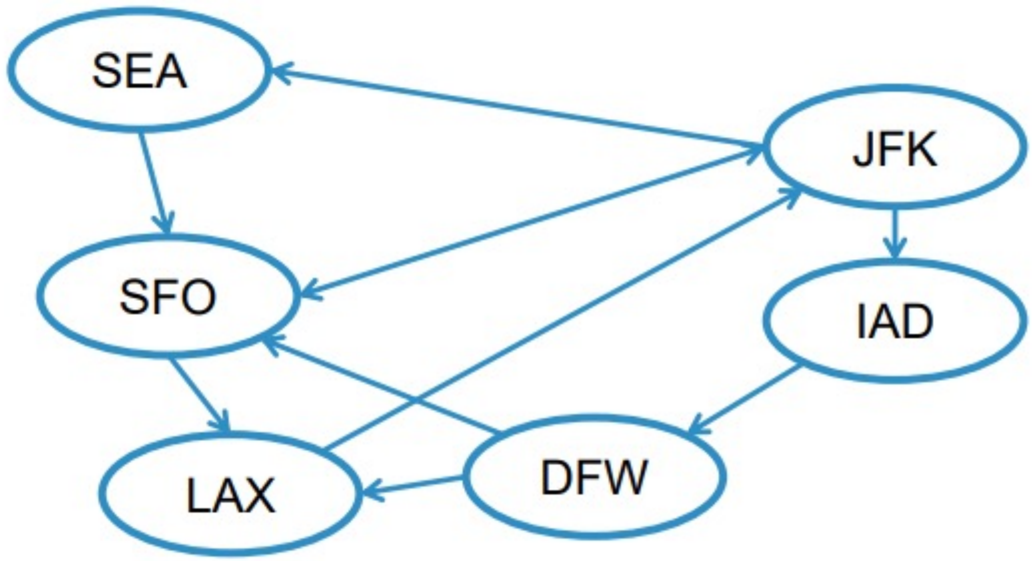


```
>>> g.vertices.show()
+---+-------+---+
| id|   name|age|
+---+-------+---+
|  a|  Alice| 34|
|  b|    Bob| 36|
|  c|Charlie| 30|
|  d|  David| 29|
|  e| Esther| 32|
|  f|  Fanny| 36|
|  g|  Gabby| 60|
+---+-------+---+

>>> g.edges.show()
+---+---+------------+
|src|dst|relationship|
+---+---+------------+
|  a|  b|      friend|
|  b|  c|      follow|
|  c|  b|      follow|
|  f|  c|      follow|
|  e|  f|      follow|
|  e|  d|      friend|
|  d|  a|      friend|
|  a|  e|      friend|
+---+---+------------+
```

## Motif Finding

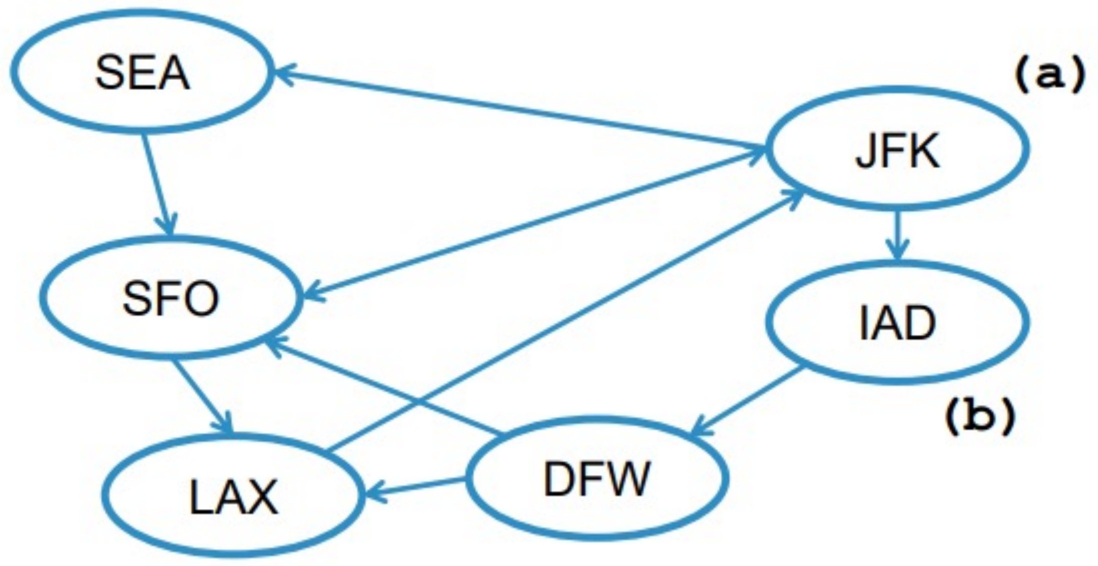
- Search for structural pattern in a graph

```
val paths: DataFrame =
  g.find("(a)-[e1]->(b);
              (b)-[e2]->(c);
             !(c)-[]->(a)")
```



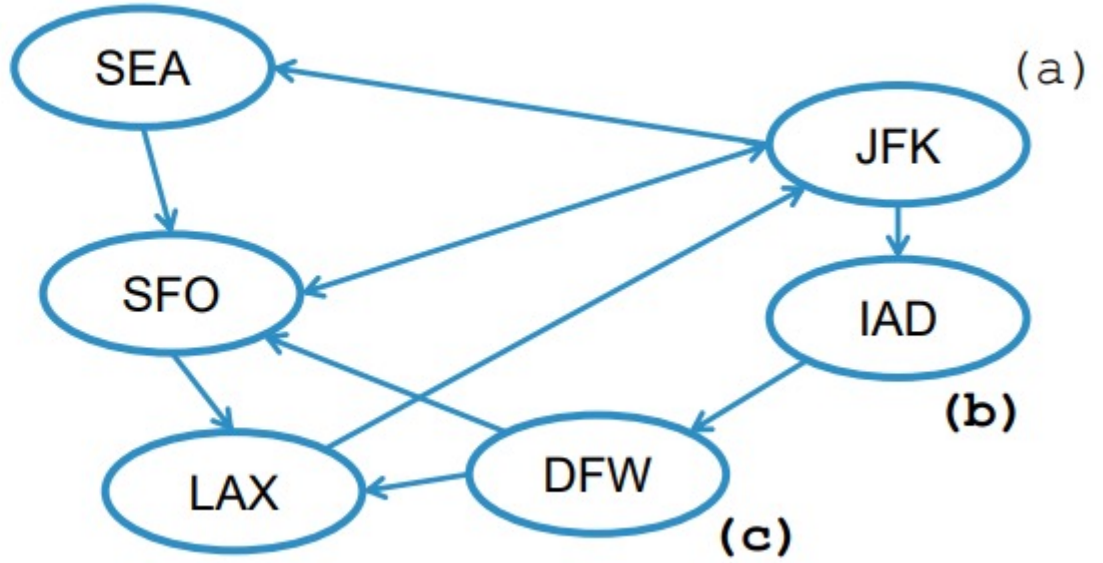
## Continued …

```
val paths: DataFrame =
  g.find("(a)-[e1]->(b);
          (b)-[e2]->(c);
         !(c)-[]->(a)")
```



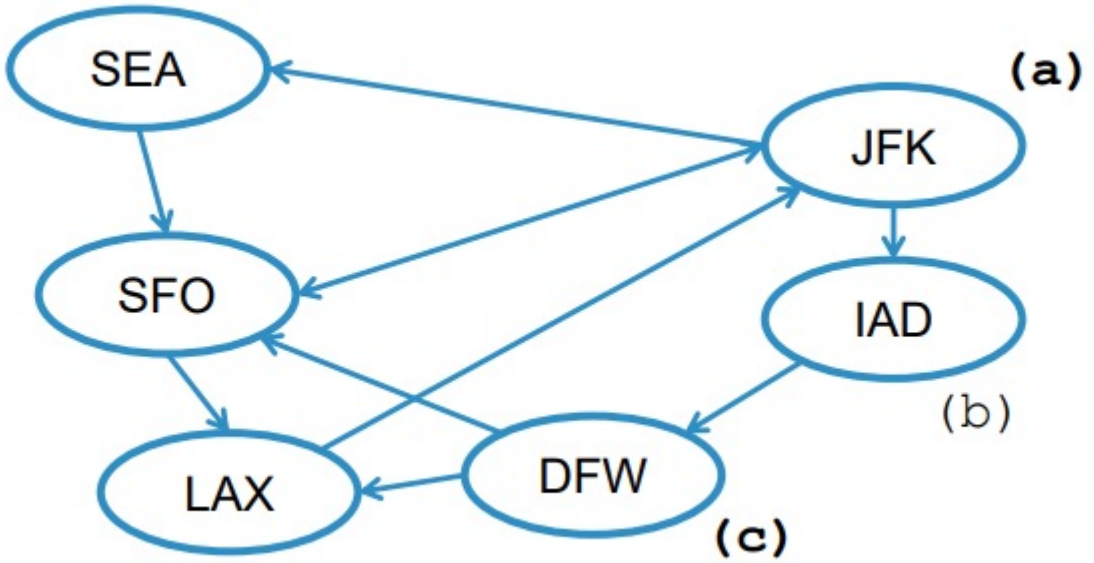
# Continued …

```
val paths: DataFrame =
  g.find("(a)-[e1]->(b);
          (b)-[e2]->(c);
         !(c)-[]->(a)")
```

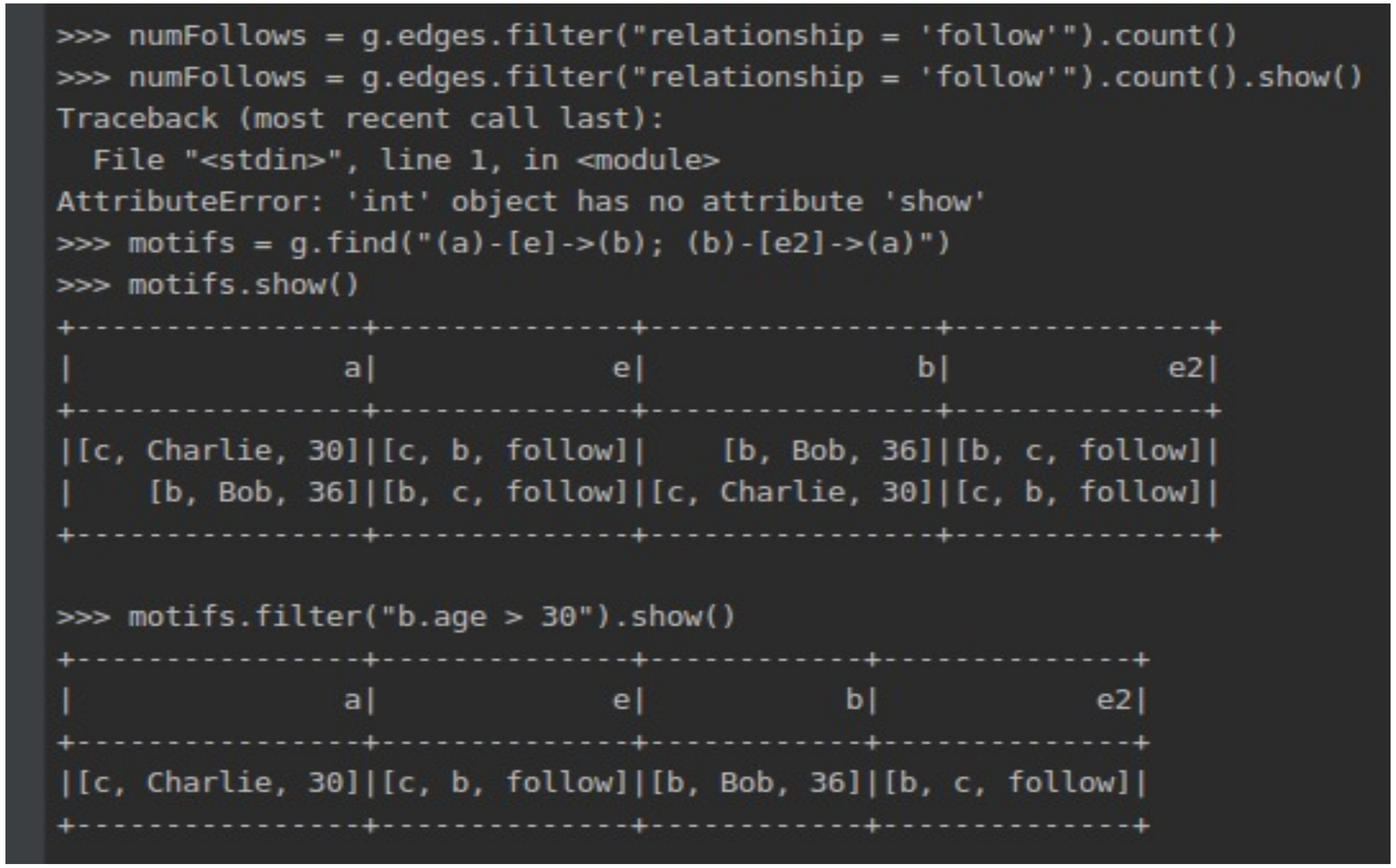


## Continued …

```
val paths: DataFrame =
  g.find("(a)-[e1]->(b);
          (b)-[e2]->(c);
        !(c)-[]->(a)")
```

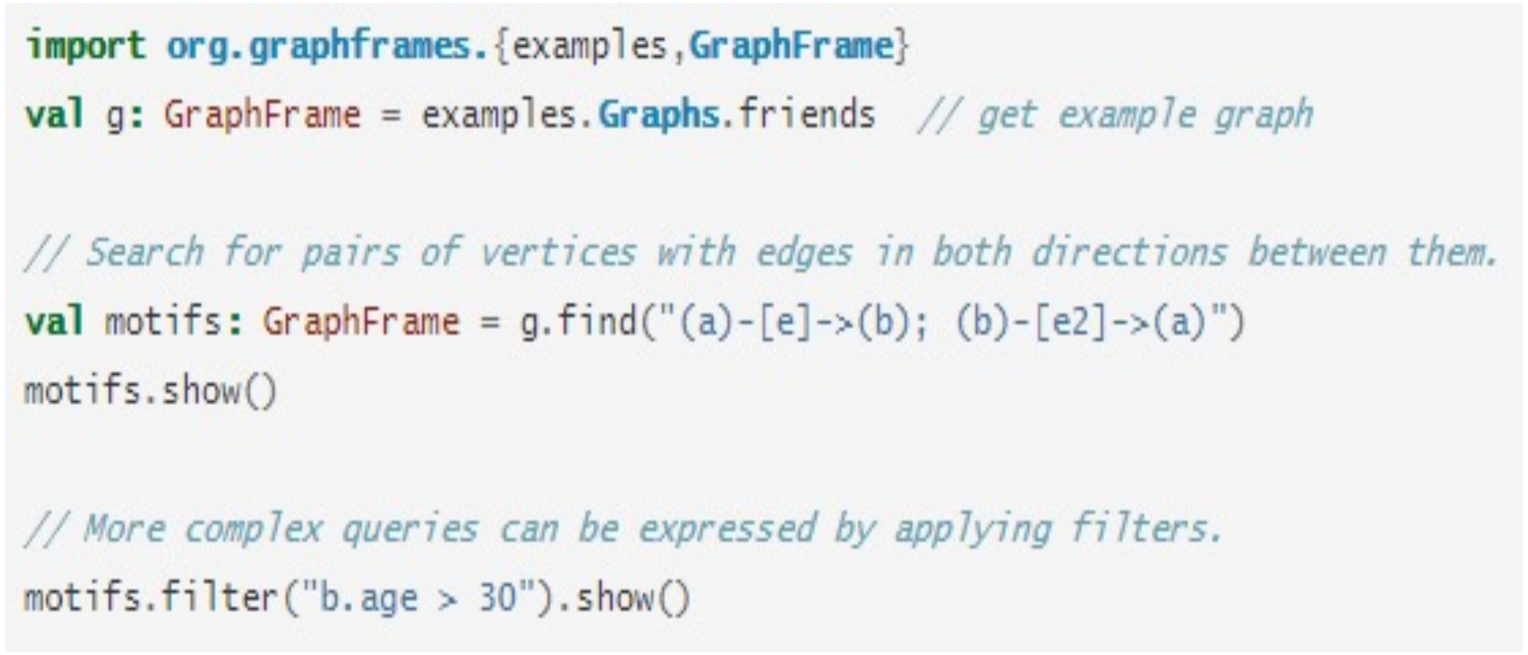


## Motifs filter by age (pyspark)



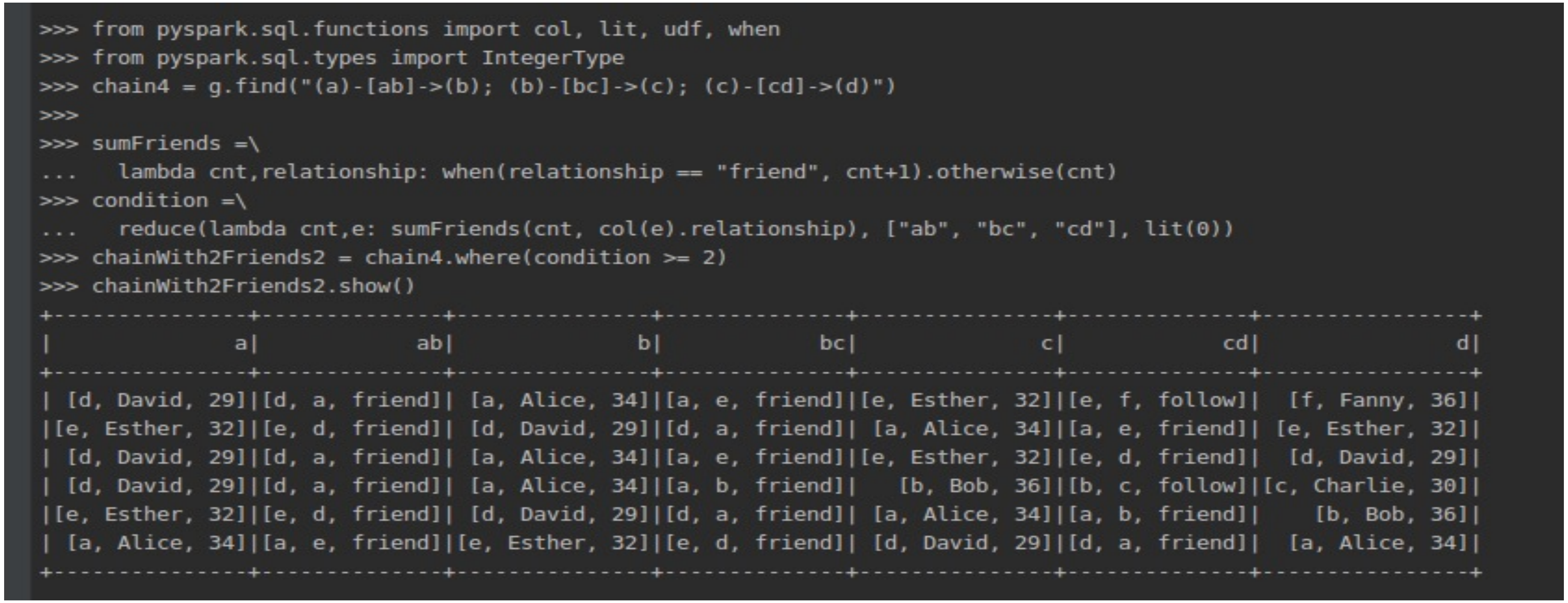
```
>>> numFollows = g.edges.filter("relationship = 'follow'").count()
>>> numFollows = g.edges.filter("relationship = 'follow'").count().show()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'int' object has no attribute 'show'
>>> motifs = g.find("(a)-[e]->(b); (b)-[e2]->(a)")
>>> motifs.show()
+----------------+--------------+----------------+--------------+
|               a|             e|               b|            e2|
+----------------+--------------+----------------+--------------+
|[c, Charlie, 30]|[c, b, follow]|    [b, Bob, 36]|[b, c, follow]|
|    [b, Bob, 36]|[b, c, follow]|[c, Charlie, 30]|[c, b, follow]|
+----------------+--------------+----------------+--------------+

>>> motifs.filter("b.age > 30").show()
+----------------+--------------+------------+--------------+
|               a|             e|           b|            e2|
+----------------+--------------+------------+--------------+
|[c, Charlie, 30]|[c, b, follow]|[b, Bob, 36]|[b, c, follow]|
+----------------+--------------+------------+--------------+
```

## Scala



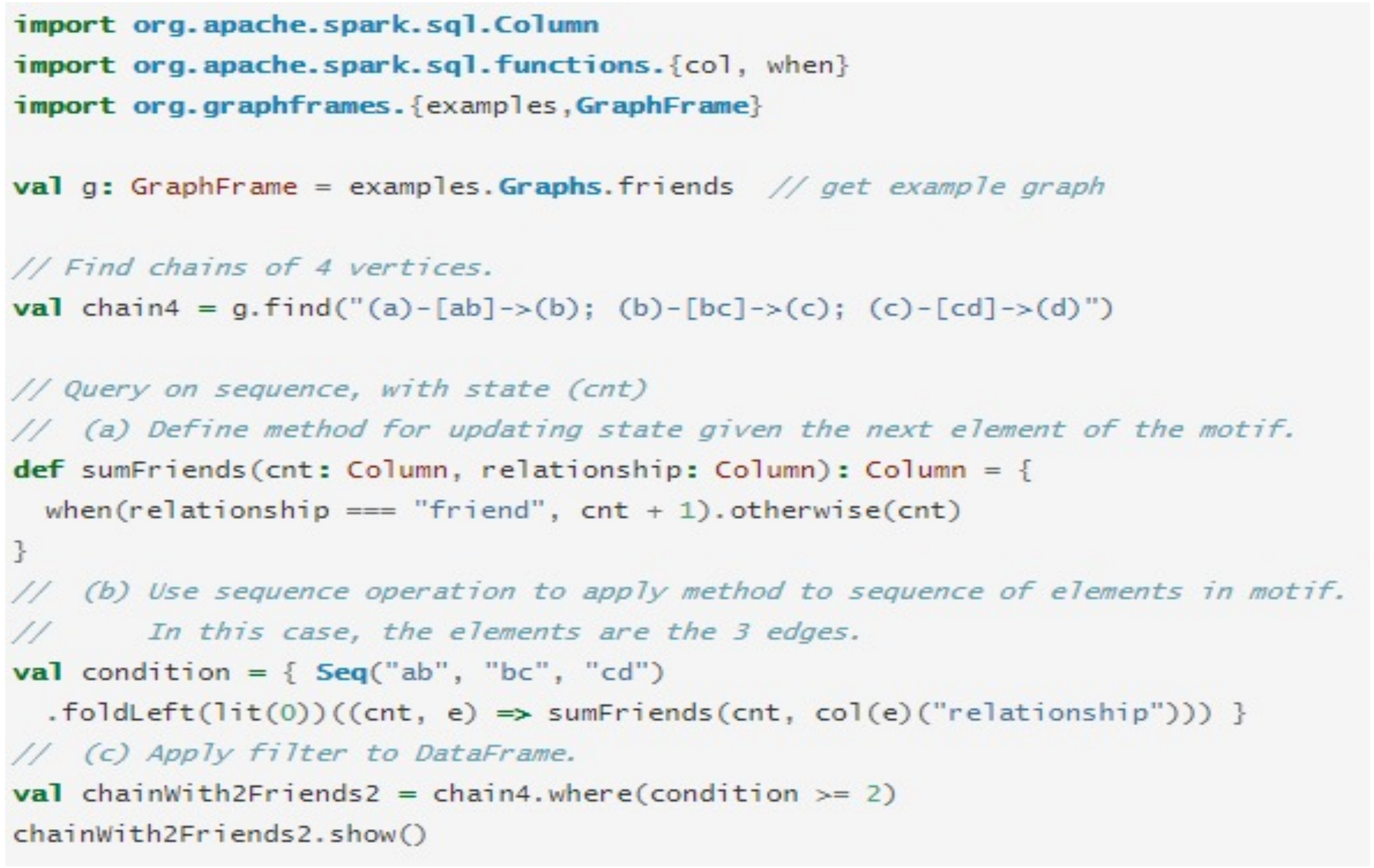
```
import org.graphframes.{examples,GraphFrame}
val g: GraphFrame = examples.Graphs.friends  // get example graph

// Search for pairs of vertices with edges in both directions between them.
val motifs: GraphFrame = g.find("(a)-[e]->(b); (b)-[e2]->(a)")
motifs.show()

// More complex queries can be expressed by applying filters.
motifs.filter("b.age > 30").show()
```

# Combination of Relationship (pyspark)



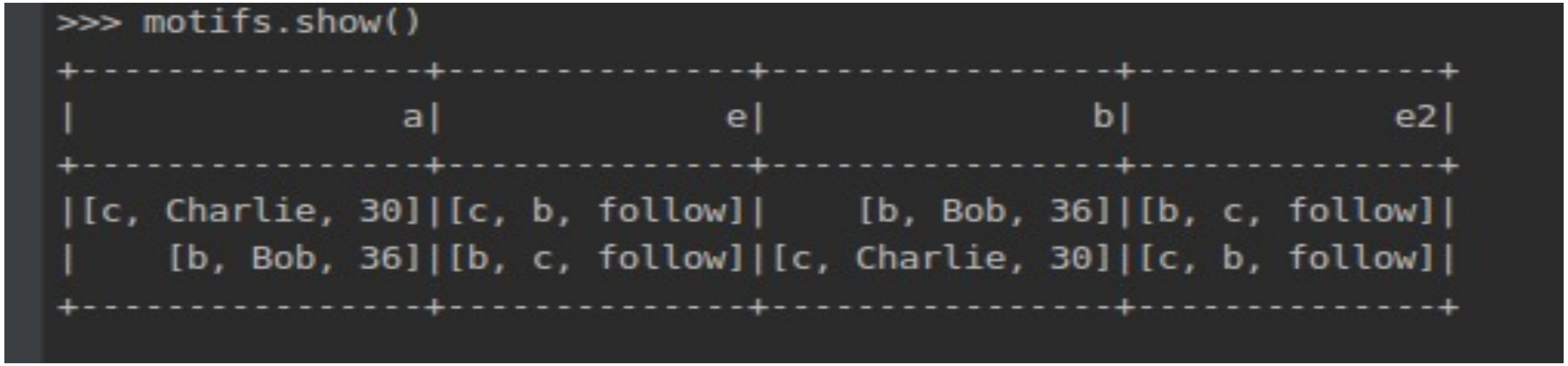
```
>>> from pyspark.sql.functions import col, lit, udf, when
>>> from pyspark.sql.types import IntegerType
>>> chain4 = g.find("(a)-[ab]->(b); (b)-[bc]->(c); (c)-[cd]->(d)")
>>>
>>> sumFriends =\
...   lambda cnt,relationship: when(relationship == "friend", cnt+1).otherwise(cnt)
>>> condition =\
...   reduce(lambda cnt,e: sumFriends(cnt, col(e).relationship), ["ab", "bc", "cd"], lit(0))
>>> chainWith2Friends2 = chain4.where(condition >= 2)
>>> chainWith2Friends2.show()
+--------------+-------------+--------------+-------------+--------------+-------------+---------------+
|             a|           ab|             b|           bc|             c|           cd|              d|
+--------------+-------------+--------------+-------------+--------------+-------------+---------------+
| [d, David, 29]|[d, a, friend]| [a, Alice, 34]|[a, e, friend]|[e, Esther, 32]|[e, f, follow]|  [f, Fanny, 36]|
|[e, Esther, 32]|[e, d, friend]| [d, David, 29]|[d, a, friend]| [a, Alice, 34]|[a, e, friend]| [e, Esther, 32]|
| [d, David, 29]|[d, a, friend]| [a, Alice, 34]|[a, e, friend]|[e, Esther, 32]|[e, d, friend]|  [d, David, 29]|
| [d, David, 29]|[d, a, friend]| [a, Alice, 34]|[a, b, friend]|   [b, Bob, 36]|[b, c, follow]|[c, Charlie, 30]|
|[e, Esther, 32]|[e, d, friend]| [d, David, 29]|[d, a, friend]| [a, Alice, 34]|[a, b, friend]|    [b, Bob, 36]|
| [a, Alice, 34]|[a, e, friend]|[e, Esther, 32]|[e, d, friend]| [d, David, 29]|[d, a, friend]|  [a, Alice, 34]|
+--------------+-------------+--------------+-------------+--------------+-------------+---------------+
```

## Scala



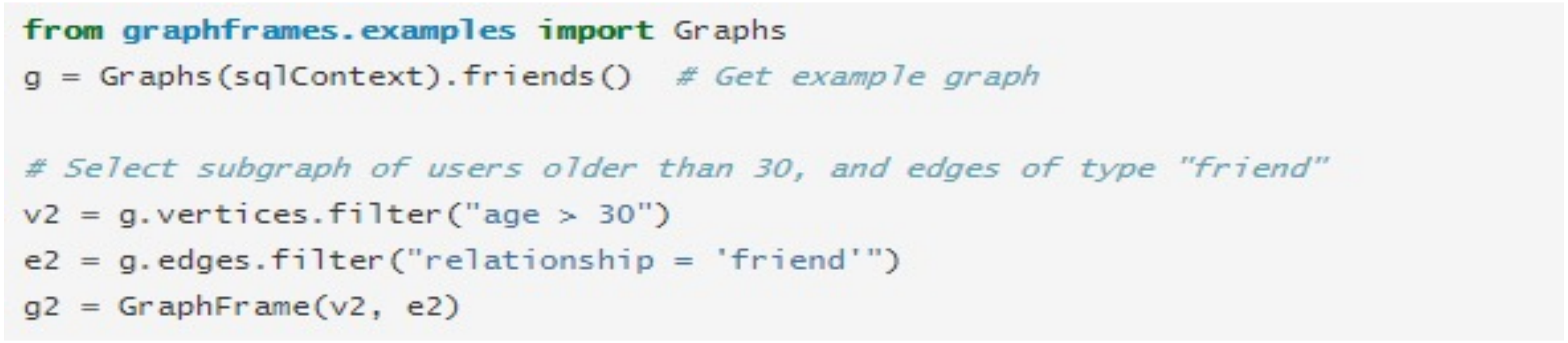
```scala
import org.apache.spark.sql.Column
import org.apache.spark.sql.functions.{col, when}
import org.graphframes.{examples,GraphFrame}

val g: GraphFrame = examples.Graphs.friends  // get example graph

// Find chains of 4 vertices.
val chain4 = g.find("(a)-[ab]->(b); (b)-[bc]->(c); (c)-[cd]->(d)")

// Query on sequence, with state (cnt)
//  (a) Define method for updating state given the next element of the motif.
def sumFriends(cnt: Column, relationship: Column): Column = {
  when(relationship === "friend", cnt + 1).otherwise(cnt)
}
//  (b) Use sequence operation to apply method to sequence of elements in motif.
//      In this case, the elements are the 3 edges.
val condition = { Seq("ab", "bc", "cd")
  .foldLeft(lit(0))((cnt, e) => sumFriends(cnt, col(e)("relationship"))) }
//  (c) Apply filter to DataFrame.
val chainWith2Friends2 = chain4.where(condition >= 2)
chainWith2Friends2.show()
```

## Motif show



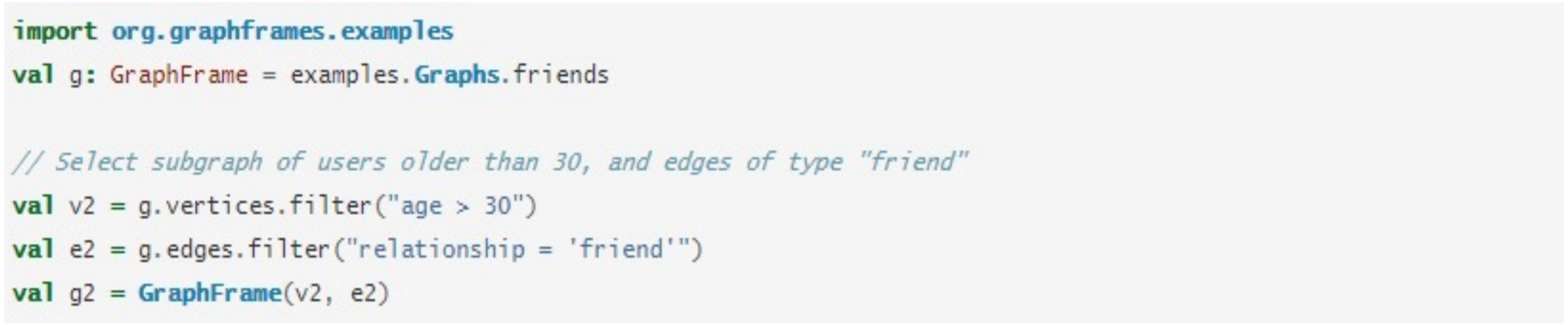
```
>>> motifs.show()
+---------------+--------------+----------------+--------------+
|              a|             e|               b|            e2|
+---------------+--------------+----------------+--------------+
|[c, Charlie, 30]|[c, b, follow]|    [b, Bob, 36]|[b, c, follow]|
|    [b, Bob, 36]|[b, c, follow]|[c, Charlie, 30]|[c, b, follow]|
+---------------+--------------+----------------+--------------+
```

## Create Subgraphs (Pyspark)



```
from graphframes.examples import Graphs
g = Graphs(sqlContext).friends()  # Get example graph

# Select subgraph of users older than 30, and edges of type "friend"
v2 = g.vertices.filter("age > 30")
e2 = g.edges.filter("relationship = 'friend'")
g2 = GraphFrame(v2, e2)
```

## Create Subgraphs (Scala)

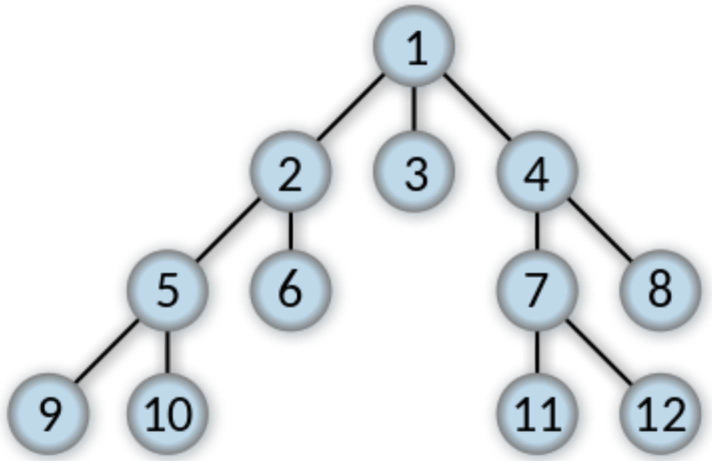


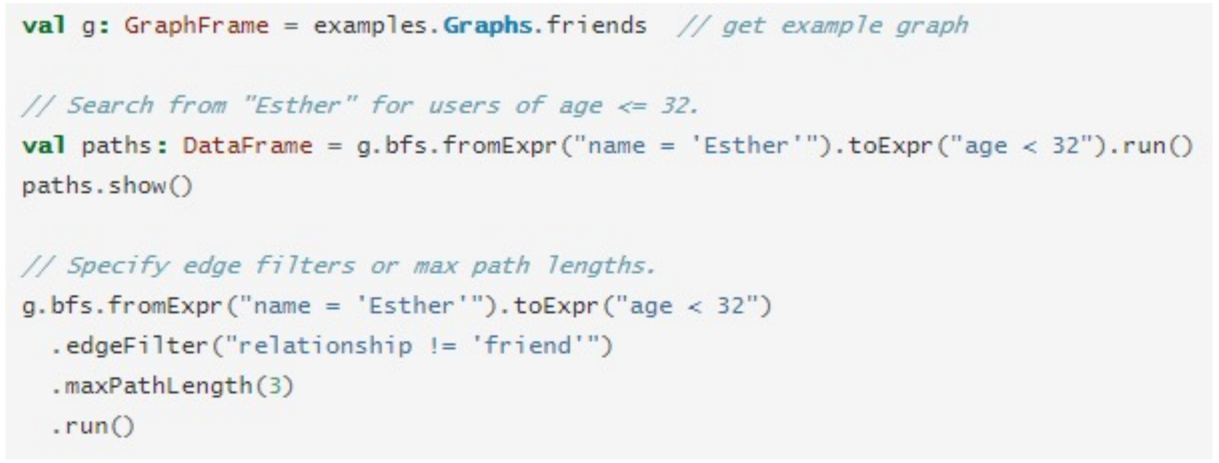
```scala
import org.graphframes.examples
val g: GraphFrame = examples.Graphs.friends

// Select subgraph of users older than 30, and edges of type "friend"
val v2 = g.vertices.filter("age > 30")
val e2 = g.edges.filter("relationship = 'friend'")
val g2 = GraphFrame(v2, e2)
```

## Graph Algorithms

## Breath-first-search (BFS) algorithm

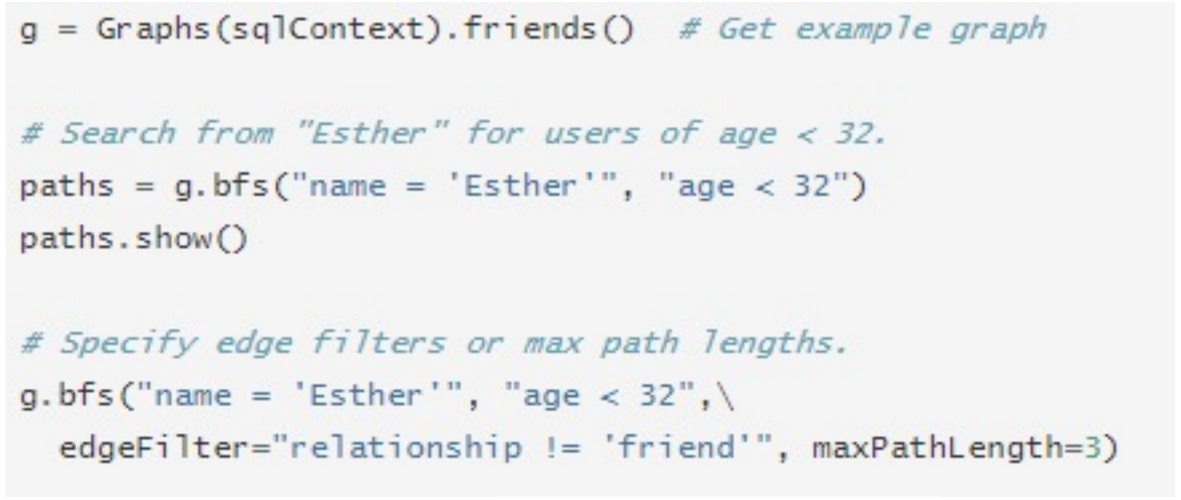
- Breadth-first search (BFS) finds the shortest path(s) from one vertex (or a set of vertices) to another vertex (or a set of vertices)
- The beginning and end vertices are specified as Spark DataFrame expressions





```scala
val g: GraphFrame = examples.Graphs.friends  // get example graph

// Search from "Esther" for users of age <= 32.
val paths: DataFrame = g.bfs.fromExpr("name = 'Esther'").toExpr("age < 32").run()
paths.show()

// Specify edge filters or max path lengths.
g.bfs.fromExpr("name = 'Esther'").toExpr("age < 32")
  .edgeFilter("relationship != 'friend'")
  .maxPathLength(3)
  .run()
```

Scala

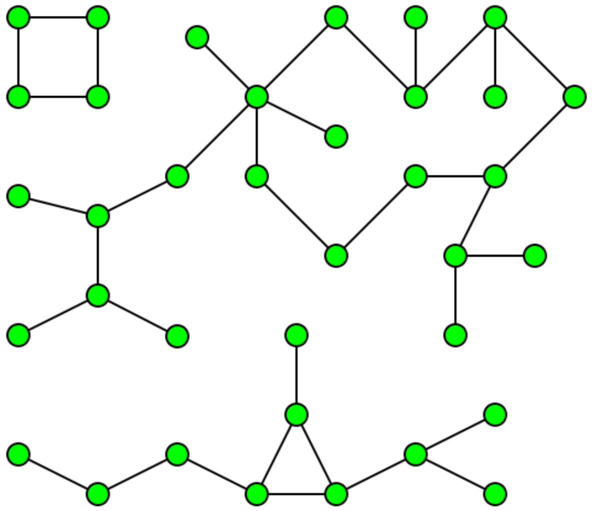


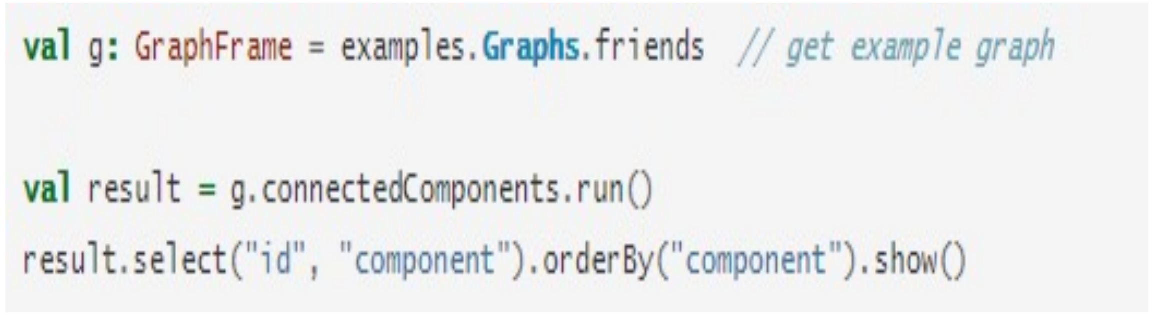
```python
g = Graphs(sqlContext).friends()  # Get example graph

# Search from "Esther" for users of age < 32.
paths = g.bfs("name = 'Esther'", "age < 32")
paths.show()

# Specify edge filters or max path lengths.
g.bfs("name = 'Esther'", "age < 32",\
  edgeFilter="relationship != 'friend'", maxPathLength=3)
```

python™

## Connected Components

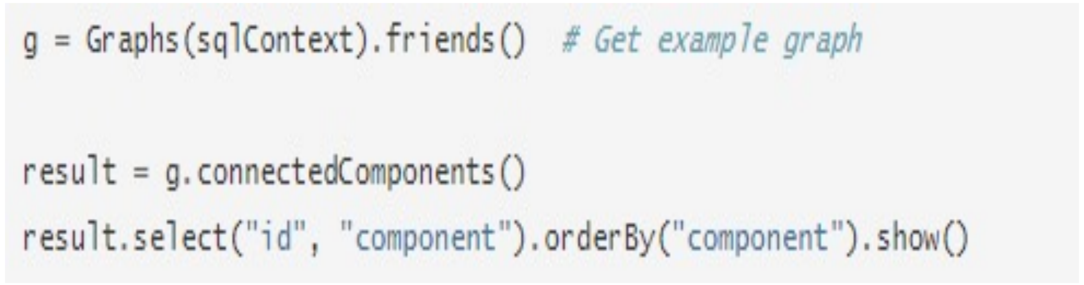
- Computes the connected component membership of each vertex and returns a graph with each vertex assigned a component ID.





```scala
val g: GraphFrame = examples.Graphs.friends  // get example graph

val result = g.connectedComponents.run()
result.select("id", "component").orderBy("component").show()
```

Scala

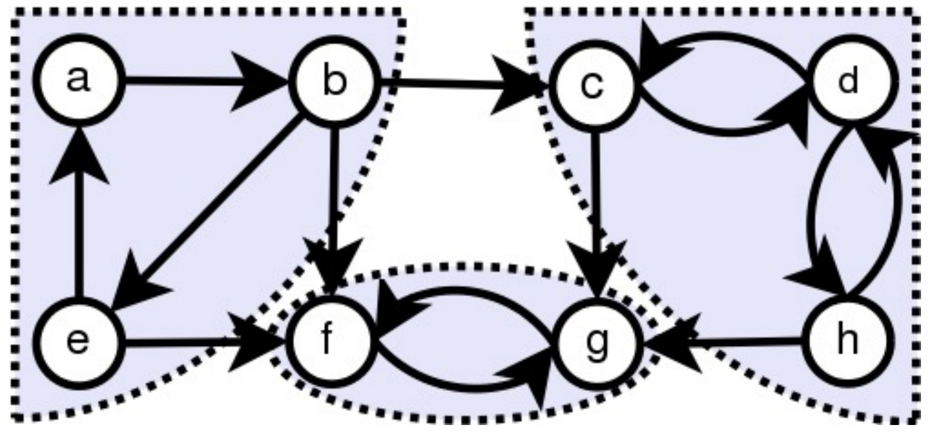


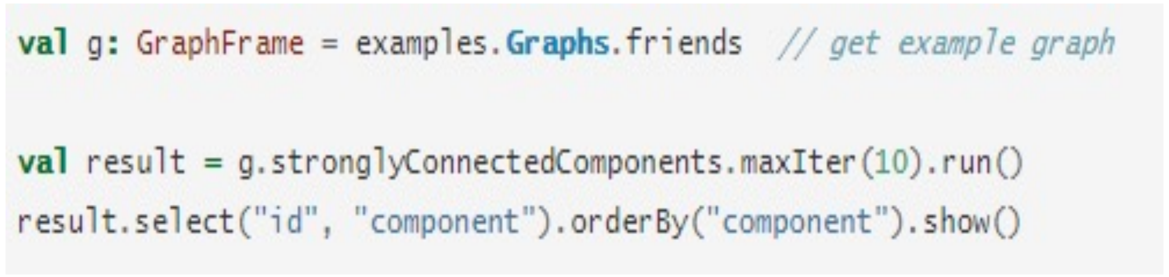
```python
g = Graphs(sqlContext).friends()  # Get example graph

result = g.connectedComponents()
result.select("id", "component").orderBy("component").show()
```

python™

## Strongly Connected Components

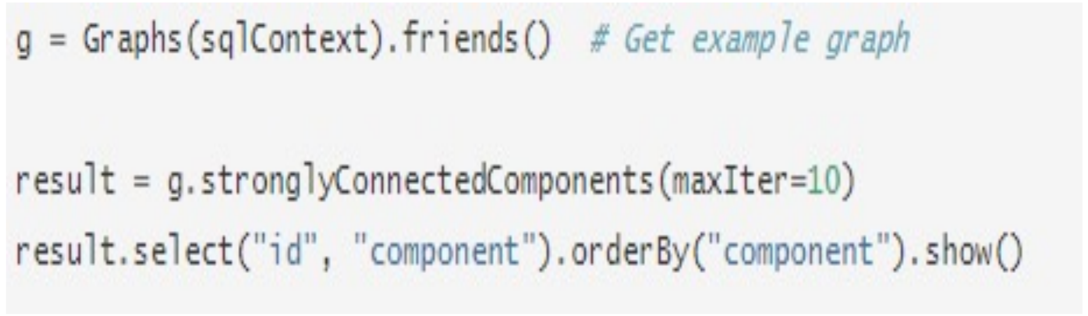
- Compute the strongly connected component (SCC) of each vertex and return a graph with each vertex assigned to the SCC containing that vertex.





```scala
val g: GraphFrame = examples.Graphs.friends  // get example graph

val result = g.stronglyConnectedComponents.maxIter(10).run()
result.select("id", "component").orderBy("component").show()
```

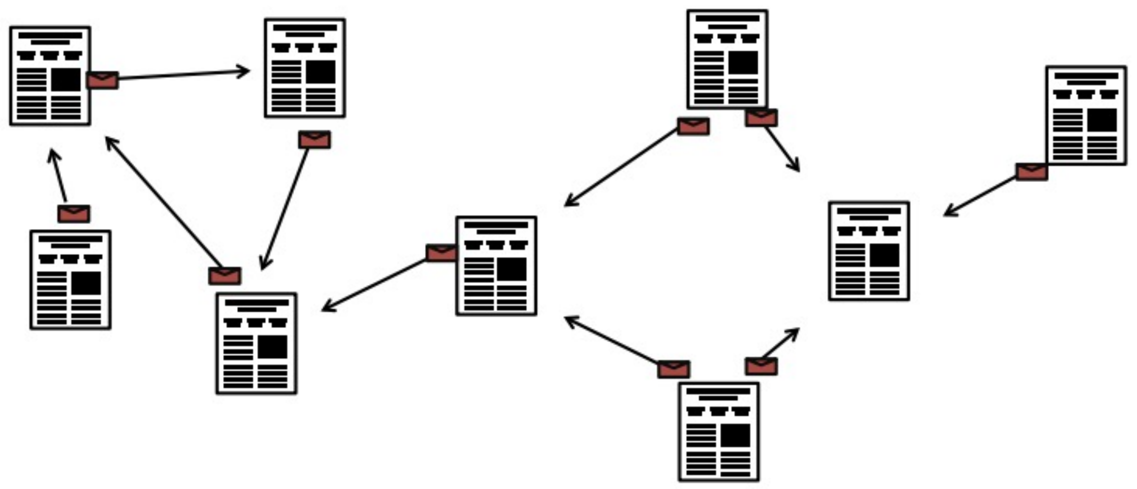
Scala



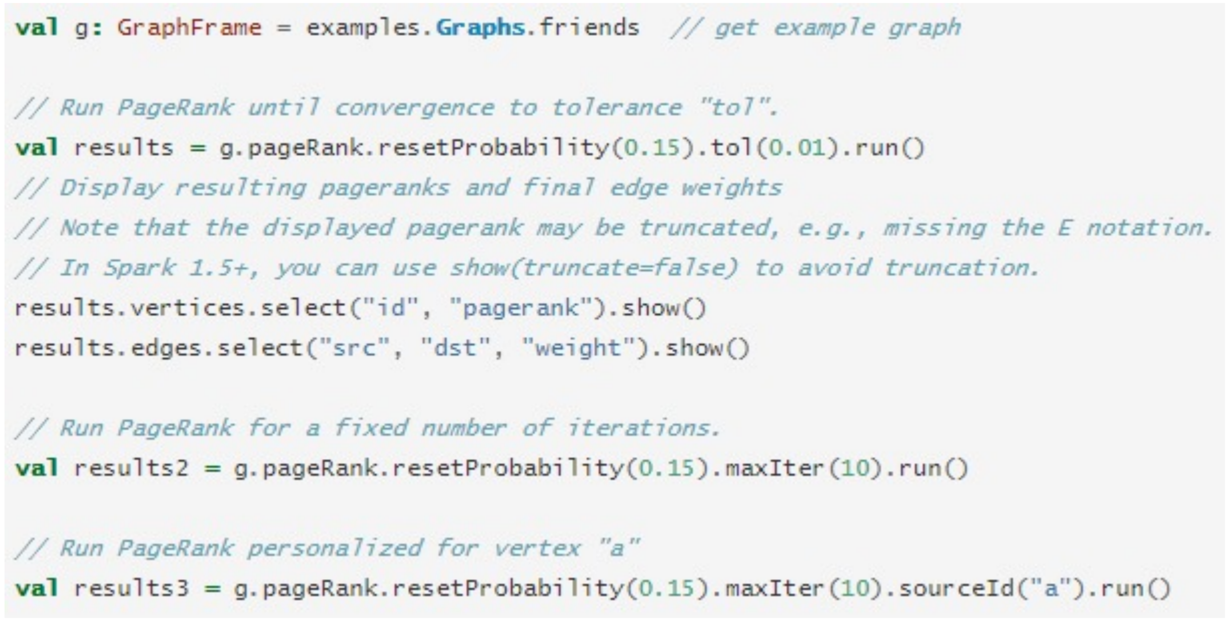
```python
g = Graphs(sqlContext).friends()  # Get example graph

result = g.stronglyConnectedComponents(maxIter=10)
result.select("id", "component").orderBy("component").show()
```

python™

## PageRank

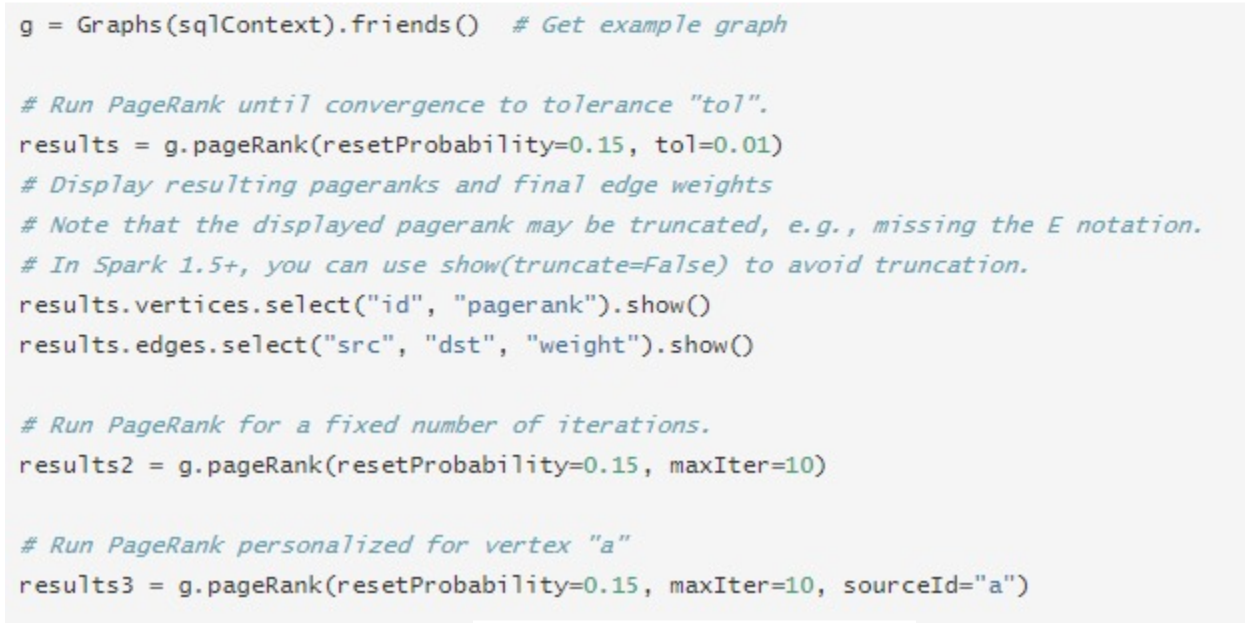


- PageRank (PR) is an algorithm used by Google Search to rank websites in their search engine results. PageRank was named after Larry Page
- PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important is the website



```scala
val g: GraphFrame = examples.Graphs.friends  // get example graph

// Run PageRank until convergence to tolerance "tol".
val results = g.pageRank.resetProbability(0.15).tol(0.01).run()
// Display resulting pageranks and final edge weights
// Note that the displayed pagerank may be truncated, e.g., missing the E notation.
// In Spark 1.5+, you can use show(truncate=false) to avoid truncation.
results.vertices.select("id", "pagerank").show()
results.edges.select("src", "dst", "weight").show()

// Run PageRank for a fixed number of iterations.
val results2 = g.pageRank.resetProbability(0.15).maxIter(10).run()

// Run PageRank personalized for vertex "a"
val results3 = g.pageRank.resetProbability(0.15).maxIter(10).sourceId("a").run()
```

Scala

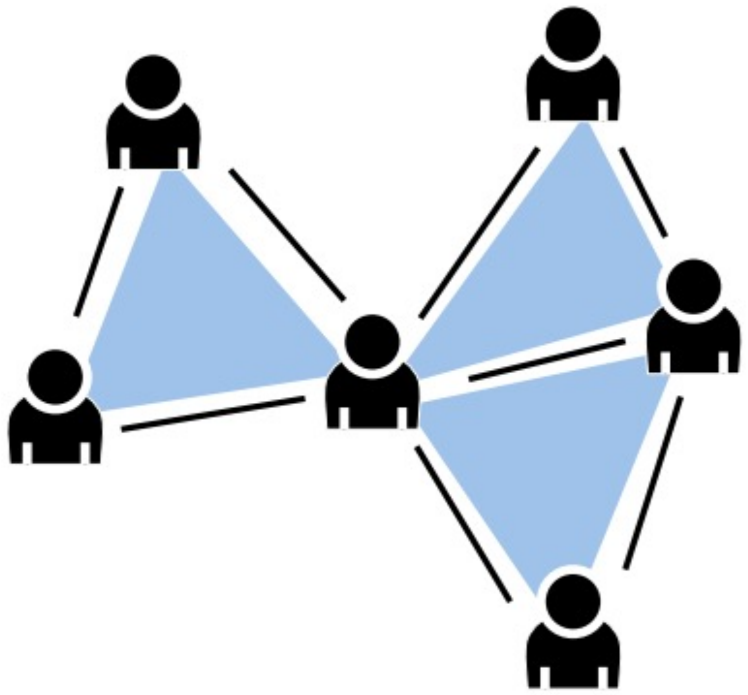


```python
g = Graphs(sqlContext).friends()  # Get example graph

# Run PageRank until convergence to tolerance "tol".
results = g.pageRank(resetProbability=0.15, tol=0.01)
# Display resulting pageranks and final edge weights
# Note that the displayed pagerank may be truncated, e.g., missing the E notation.
# In Spark 1.5+, you can use show(truncate=False) to avoid truncation.
results.vertices.select("id", "pagerank").show()
results.edges.select("src", "dst", "weight").show()

# Run PageRank for a fixed number of iterations.
results2 = g.pageRank(resetProbability=0.15, maxIter=10)

# Run PageRank personalized for vertex "a"
results3 = g.pageRank(resetProbability=0.15, maxIter=10, sourceId="a")
```

python™

## Triangle Counting

Computes the number of triangles passing through each vertex.



```scala
val g: GraphFrame = examples.Graphs.friends  // get example graph

val results = g.triangleCount.run()
results.select("id", "count").show()
```

Scala

```python
g = Graphs(sqlContext).friends()  # Get example graph

results = g.triangleCount()
results.select("id", "count").show()
```

python™

# Save and Loading Graphframe

```scala
val g: GraphFrame = examples.Graphs.friends  // get example graph

// Save vertices and edges as Parquet to some location.
g.vertices.write.parquet("hdfs://myLocation/vertices")
g.edges.write.parquet("hdfs://myLocation/edges")

// Load the vertices and edges back.
val sameV = sqlContext.read.parquet("hdfs://myLocation/vertices")
val sameE = sqlContext.read.parquet("hdfs://myLocation/edges")

// Create an identical GraphFrame.
val sameG = GraphFrame(sameV, sameE)
```

Scala

```python
g = Graphs(sqlContext).friends()  # Get example graph

# Save vertices and edges as Parquet to some location.
g.vertices.write.parquet("hdfs://myLocation/vertices")
g.edges.write.parquet("hdfs://myLocation/edges")

# Load the vertices and edges back.
sameV = sqlContext.read.parquet("hdfs://myLocation/vertices")
sameE = sqlContext.read.parquet("hdfs://myLocation/edges")

# Create an identical GraphFrame.
sameG = GraphFrame(sameV, sameE)
```

python™

## References

- https://spark.apache.org/docs/latest/graphx-programming-guide.html
- http://www.sparktutorials.net/Analyzing+Flight+Data%3A+A+Gentle+Introduction+to+GraphX+in+Sparkhttp://www.sparktutorials.net/Analyzing+Flight+Data%3A+A+Gentle+Introduction+to+GraphX+in+Spark
- https://mapr.com/blog/how-get-started-using-apache-spark-graphx-scala/
- https://www.edureka.co/blog/spark-graphx/
- https://graphframes.github.io/user-guide.html
- https://databricks.com/blog/2016/03/03/introducing-graphframes.html
- https://docs.databricks.com/spark/latest/graph-analysis/graphframes/user-guide-scala.html