# GraphX

# GRAPHX

GraphX es un nuevo componente en Spark para gráficos y cálculos de gráficos paralelos.

# GRAPH X

•Un GRAPH  es un gráfico múltiple dirigido con atributo definido por el usuario adjunto a cada vértice y borde

classGraph[VD, ED]

PASOS PARA CREARLO

1.CREAR UN  **VertexRDD–RDD[(Long, VD)]**

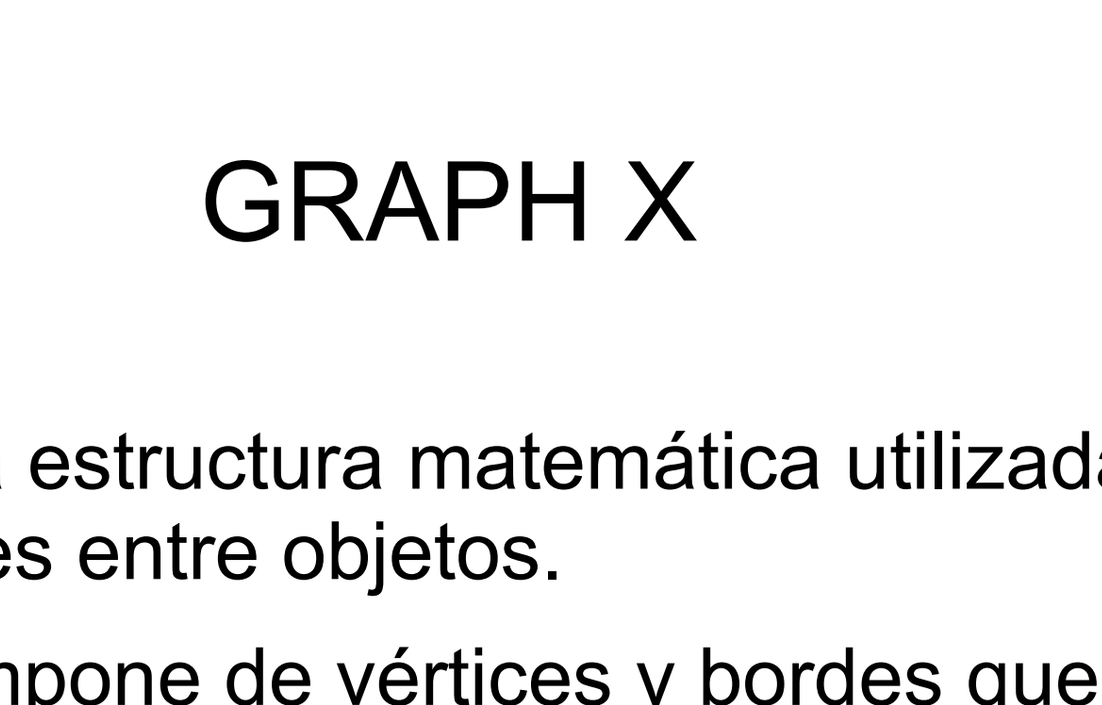
2.CREAR LOS  **EdgeRDD–RDD[Edge[ED]]**

3.CREAR UN GRAFICO USANDO LOS VERTICES Y LAS ARISTAS

•**val graph = Graph(vertexRDD, edgeRDD, error VD)**

# GRAPH X

- Un gráfico es una estructura matemática utilizada para modelar relaciones entre objetos.
- Un gráfico se compone de vértices y bordes que los conectan.
- Los vértices son los objetos y las aristas son las relaciones entre ellos.
-

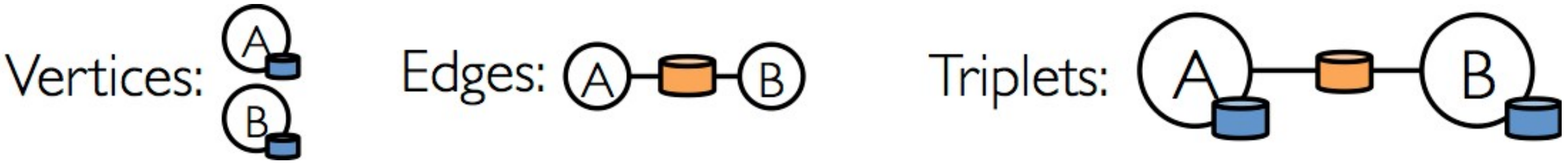


# GRAPH X

- Además de las vistas de vértices y bordes del gráfico de propiedades, GraphX también expone una vista de triplete.

- La vista de triplete une lógicamente las propiedades de vértice y borde que producen un RDD[EdgeTriplet[VD, ED]] que contiene instancias de la clase EdgeTriplet. Esta unión se puede expresar gráficamente como:

# GRAPH X




The `EdgeTriplet` class extends the `Edge` class by adding the `srcAttr`
respectively.

Use the `graph.triplets` view to display who likes who. The output sh

```
Bob likes Alice
Bob likes David
Charlie likes Bob
Charlie likes Fran
David likes Alice
Ed likes Bob
Ed likes Charlie
Ed likes Fran
```

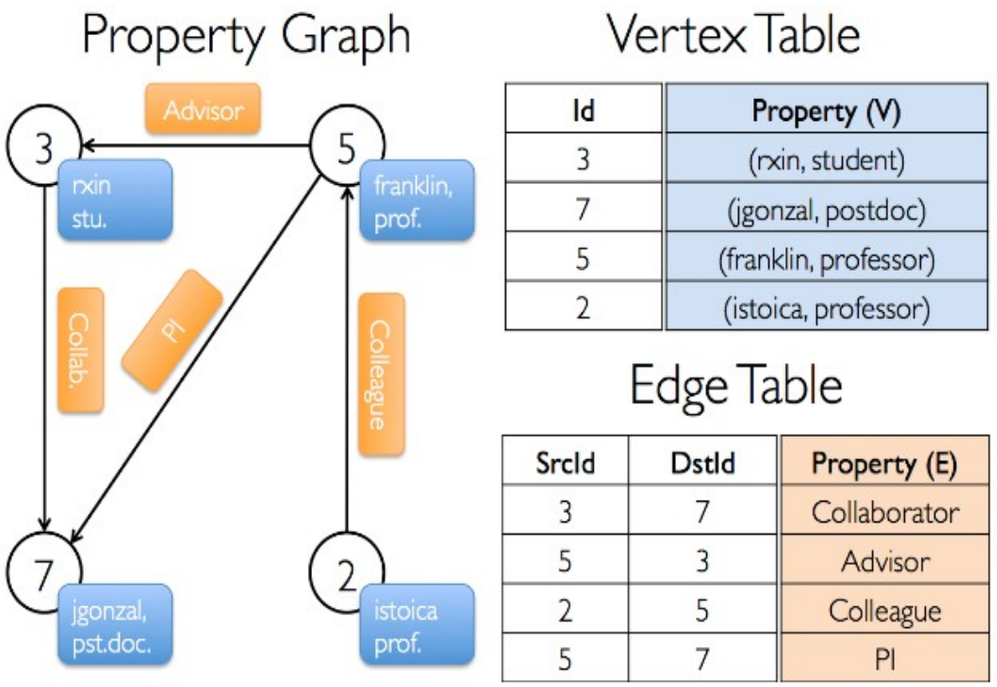
Here is a partial solution:

# GRAPH X

- **Un gráfico dirigido** es un gráfico donde los bordes tienen una dirección asociada a ellos. Un ejemplo de un gráfico dirigido es un seguidor de Twitter. El usuario Bob puede seguir al usuario Carol sin dar a entender que el usuario Carol sigue al usuario Bob.

- **Un gráfico normal** es un gráfico donde cada vértice tiene el mismo número de bordes. Un ejemplo de un gráfico regular son los amigos de Facebook. Si Bob es amigo de Carol, entonces Carol también es amiga de Bob.

# GRAPH X




Vertex Table

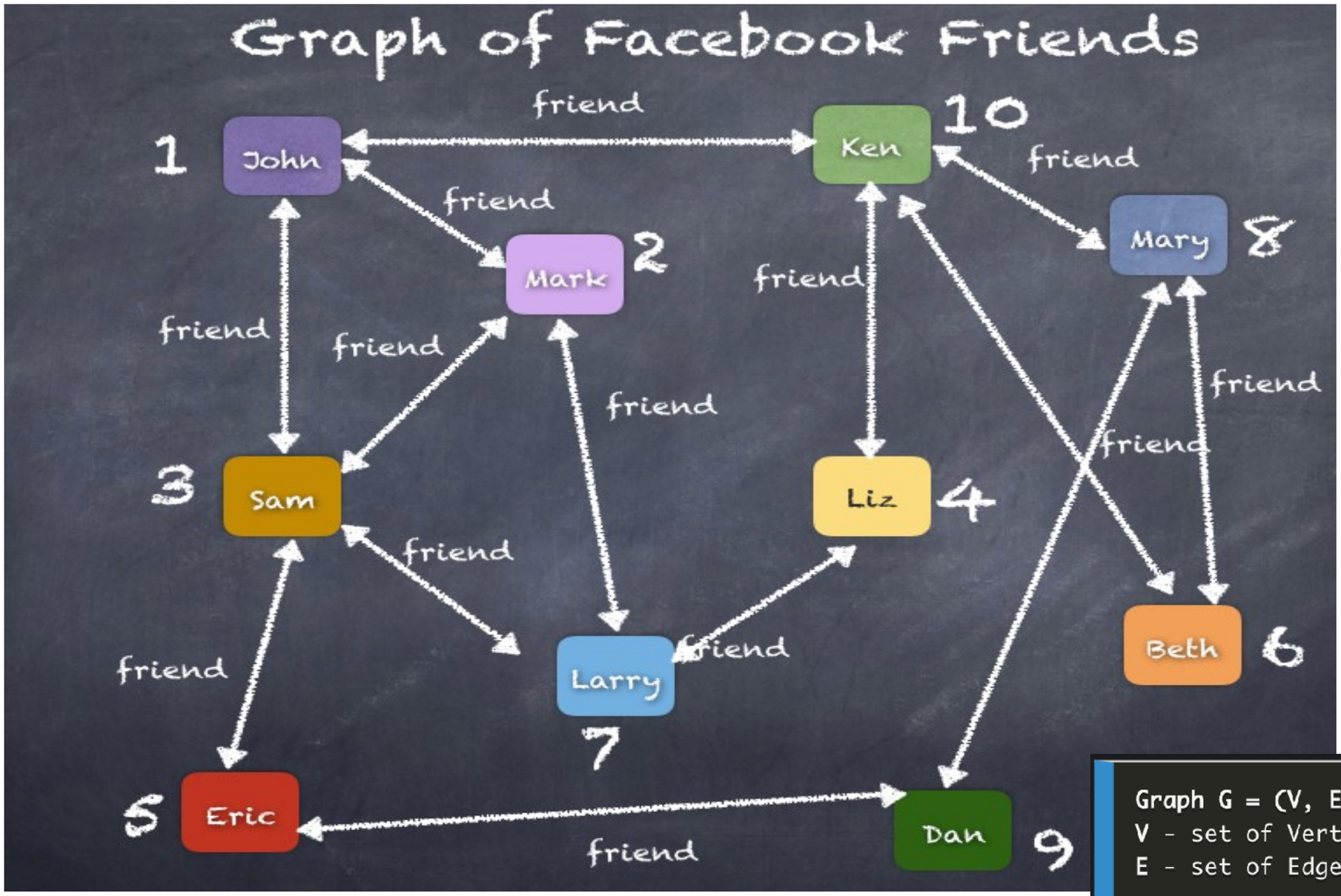
| Id | Property (V) |
|---|---|
| 3 | (rxin, student) |
| 7 | (jgonzal, postdoc) |
| 5 | (franklin, professor) |
| 2 | (istoica, professor) |

Edge Table

| SrcId | DstId | Property (E) |
|---|---|---|
| 3 | 7 | Collaborator |
| 5 | 3 | Advisor |
| 2 | 5 | Colleague |
| 5 | 7 | PI |

e resulting graph would have the type signature:

# GRAPH X. CASO REAL




```
Graph G = (V, E)
V - set of Vertices
E - set of Edges
```

# GRAPHX

## LIBRERIAS QUE VAMOS A NECESITAR

```
import org.apache.spark._
import org.apache.spark.graphx._
import org.apache.spark.rdd.RDD

import org.apache.spark.graphx.GraphLoader
import org.apache.spark.graphx.GraphOps
```

# GRAPH X. OPERACIONES

```
/** Summary of the functionality in the property graph */
class Graph[VD, ED] {
  // Information about the Graph
  val numEdges: Long
  val numVertices: Long
  val inDegrees: VertexRDD[Int]
  val outDegrees: VertexRDD[Int]
  val degrees: VertexRDD[Int]

  // Views of the graph as collections
  val vertices: VertexRDD[VD]
  val edges: EdgeRDD[ED]
  val triplets: RDD[EdgeTriplet[VD, ED]]


  // Transform vertex and edge attributes
  def mapVertices[VD2](map: (VertexID, VD) => VD2): Graph[VD2, ED]
  def mapEdges[ED2](map: Edge[ED] => ED2): Graph[VD, ED2]
  def mapEdges[ED2](map: (PartitionID, Iterator[Edge[ED]]) => Iterator[ED2]): Graph[VD,
ED2]
  def mapTriplets[ED2](map: EdgeTriplet[VD, ED] => ED2): Graph[VD, ED2]
```

# GRAPH X.OPERACIONES

```
// Join RDDs with the graph
  def joinVertices[U](table: RDD[(VertexID, U)])(mapFunc: (VertexID, VD, U) =>
VD): Graph[VD, ED]
  def outerJoinVertices[U, VD2](other: RDD[(VertexID, U)])
      (mapFunc: (VertexID, VD, Option[U]) => VD2)
    : Graph[VD2, ED]

  // Aggregate information about adjacent triplets
  def collectNeighbors(edgeDirection: EdgeDirection):
VertexRDD[Array[(VertexID, VD)]]
  def mapReduceTriplets[A: ClassTag](
      mapFunc: EdgeTriplet[VD, ED] => Iterator[(VertexID, A)],
      reduceFunc: (A, A) => A)
    : VertexRDD[A]
  // Basic graph algorithms
  def pageRank(tol: Double, resetProb: Double = 0.15): Graph[Double, Double]
  def connectedComponents(): Graph[VertexID, ED]
  def triangleCount(): Graph[Int, ED]
  def stronglyConnectedComponents(numIter: Int): Graph[VertexID, ED]
}
```

# GRAPH X

Table 4.1 shows some of the useful fields available on `EdgeTriplet`.

**Table 4.1. Key fields provided by `EdgeTriplet`**

| Field | Description |
|---|---|
| Attr | Attribute data for the edge |
| srcId | Vertex id of the edge's source vertex |
| srcAttr | Attribute data for the edge's source vertex |
| dstId | Vertex id of the edge's destination vertex |
| dstAttr | Attribute data for the edge's destination vertex |

Sign in for more free preview time

val triplets: RDD[EdgeTriplet[VD, ED]]

GRAPHX

**LISTA DE LOS ALGORITMOS:**

PageRank
Triangle Counting
Collaborative Filtering
The Graph-Parallel Pattern

# GRAPH X

Collaborative Filtering
- » Alternating Least Squares
- » Stochastic Gradient Descent
- » Tensor Factorization

Community Detection
- » Triangle-Counting
- » K-core Decomposition
- » K-Truss

Structured Prediction
- » Loopy Belief Propagation
- » Max-Product Linear Programs
- » Gibbs Sampling

Graph Analytics
- » PageRank
- » Personalized PageRank
- » Shortest Path
- » Graph Coloring

Semi-supervised ML
- » Graph SSL
- » CoEM

Classification
- » Neural Networks