GraphX intro

@adrianulbona

What is GraphX?

- **graph library** for **Spark**
- graph abstraction used: **property graphs**
- **Scala**
- **Pregel**-ish
- **scalable?**

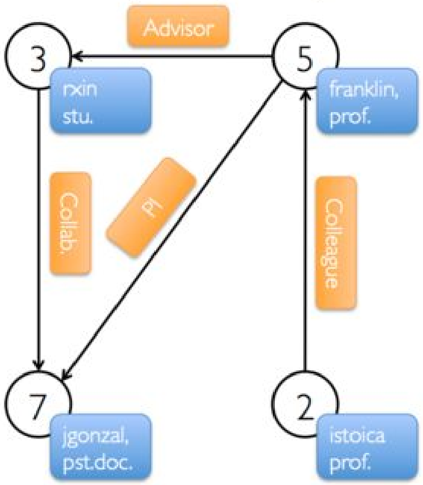
What is GraphX?

- **graph library** for **Spark**
- graph abstraction used: **property graphs**
- **Scala**
- **Pregel**-ish
- ~~**scalable**~~**? kind of scalable**

How it works? (1)

- well, property graphs are simple enough to use **RDDs** behind the curtain

Property Graph



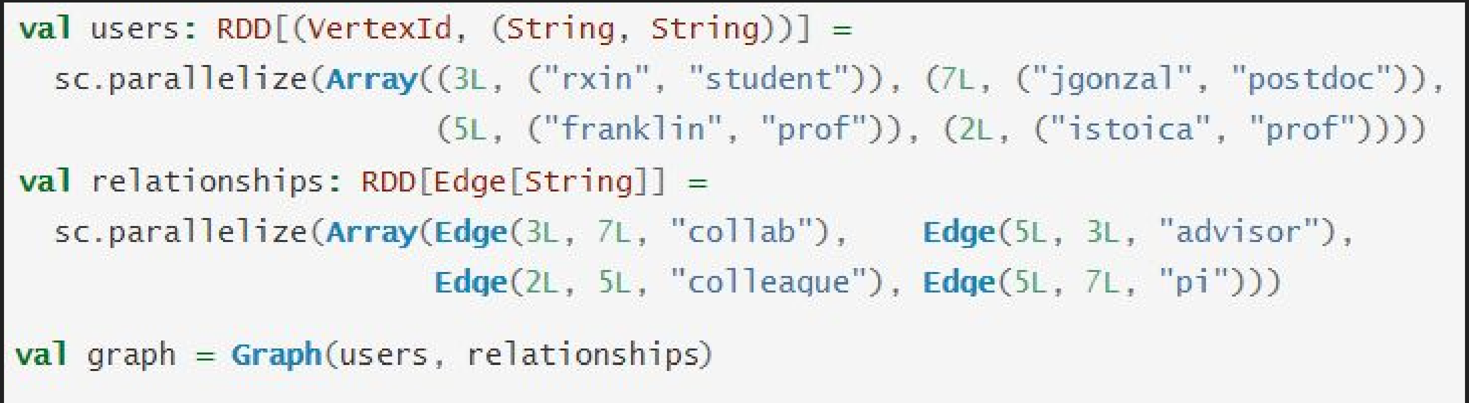
Vertex Table

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

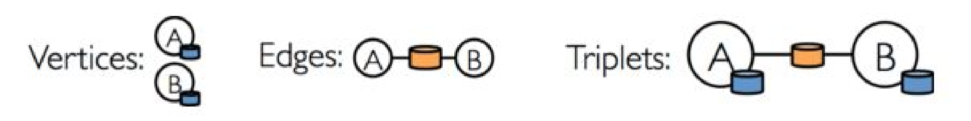
How it works? (2)

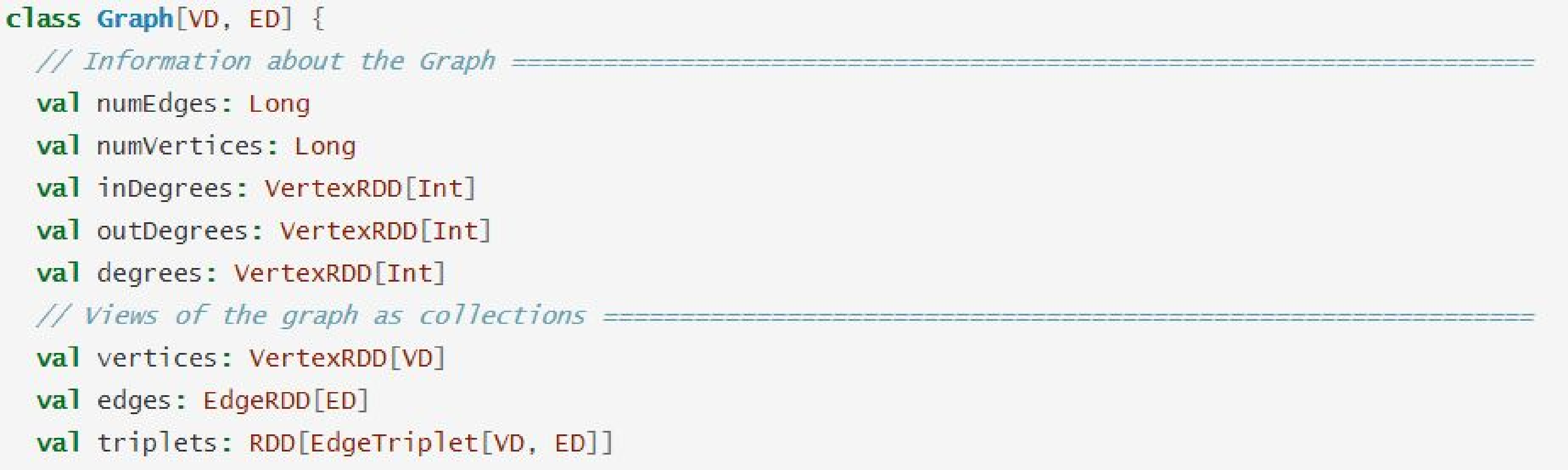


```scala
val users: RDD[(VertexId, (String, String))] =
  sc.parallelize(Array((3L, ("rxin", "student")), (7L, ("jgonzal", "postdoc")),
                       (5L, ("franklin", "prof")), (2L, ("istoica", "prof"))))
val relationships: RDD[Edge[String]] =
  sc.parallelize(Array(Edge(3L, 7L, "collab"),    Edge(5L, 3L, "advisor"),
                       Edge(2L, 5L, "colleague"), Edge(5L, 7L, "pi")))

val graph = Graph(users, relationships)
```

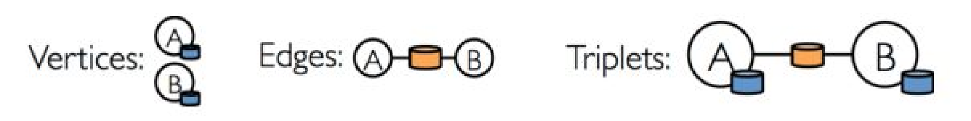
And now? (1)

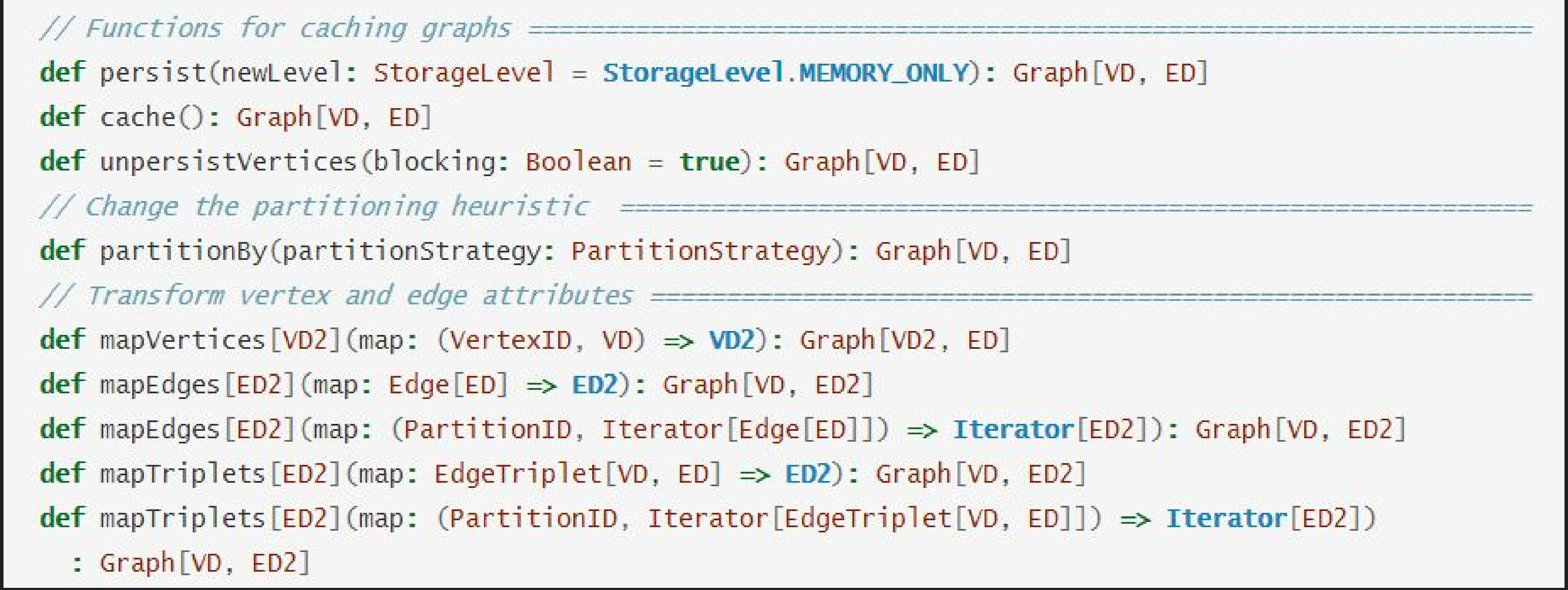




```
class Graph[VD, ED] {
  // Information about the Graph ===================================================================
  val numEdges: Long
  val numVertices: Long
  val inDegrees: VertexRDD[Int]
  val outDegrees: VertexRDD[Int]
  val degrees: VertexRDD[Int]
  // Views of the graph as collections =============================================================
  val vertices: VertexRDD[VD]
  val edges: EdgeRDD[ED]
  val triplets: RDD[EdgeTriplet[VD, ED]]
```

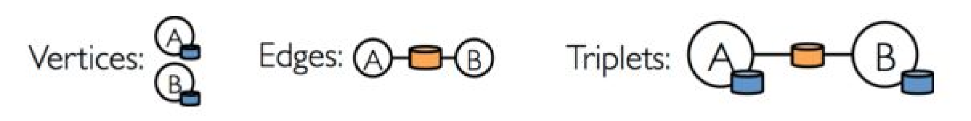
And now? (2)

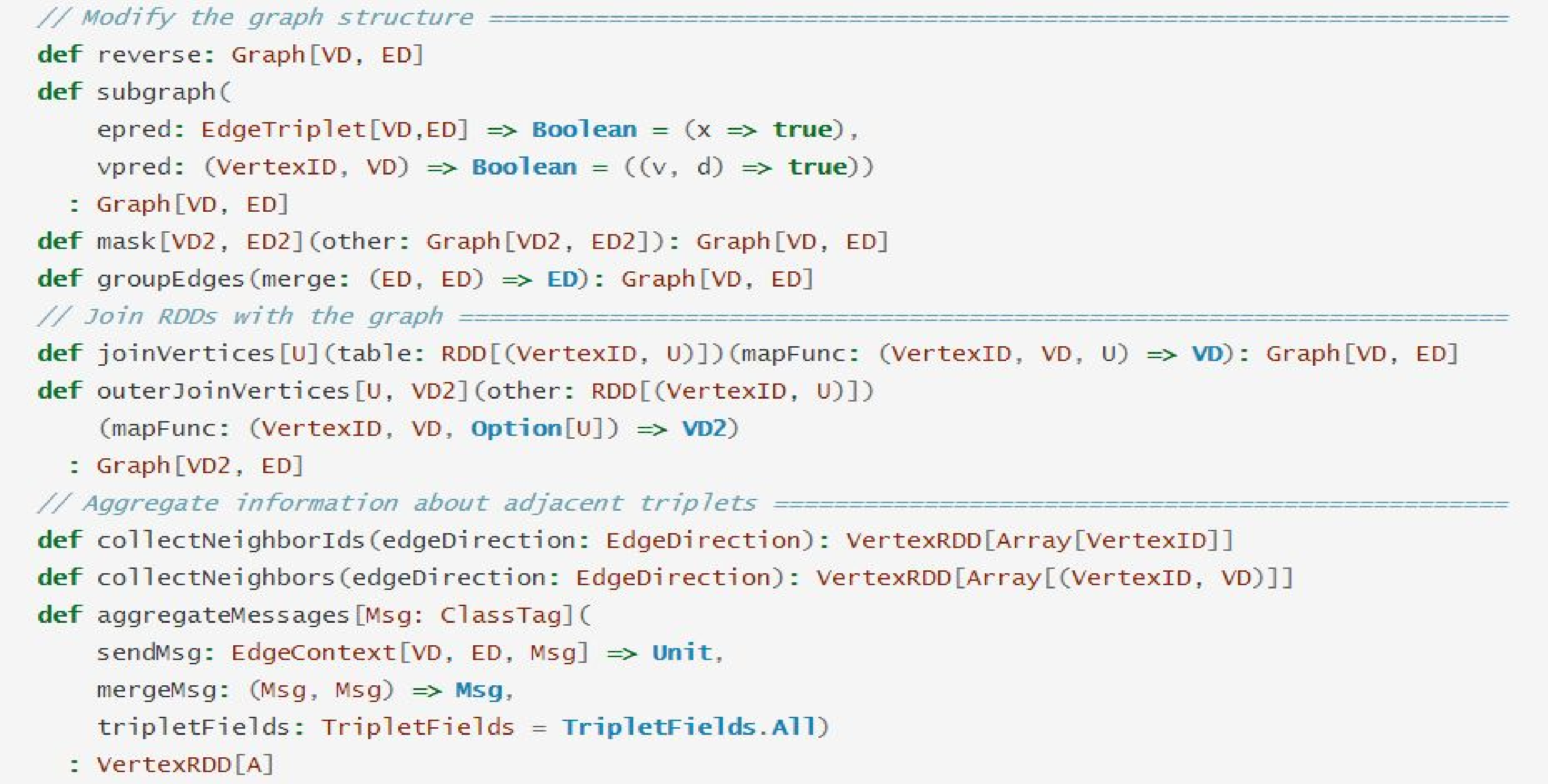




```
// Functions for caching graphs ===================================================================
def persist(newLevel: StorageLevel = StorageLevel.MEMORY_ONLY): Graph[VD, ED]
def cache(): Graph[VD, ED]
def unpersistVertices(blocking: Boolean = true): Graph[VD, ED]
// Change the partitioning heuristic  =============================================================
def partitionBy(partitionStrategy: PartitionStrategy): Graph[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]
def mapTriplets[ED2](map: (PartitionID, Iterator[EdgeTriplet[VD, ED]]) => Iterator[ED2])
  : Graph[VD, ED2]
```

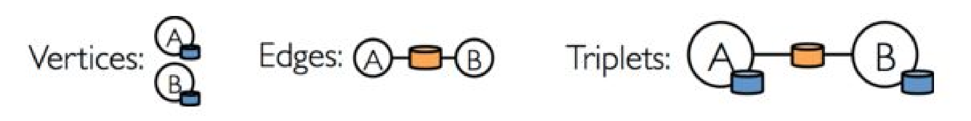
And now? (3)

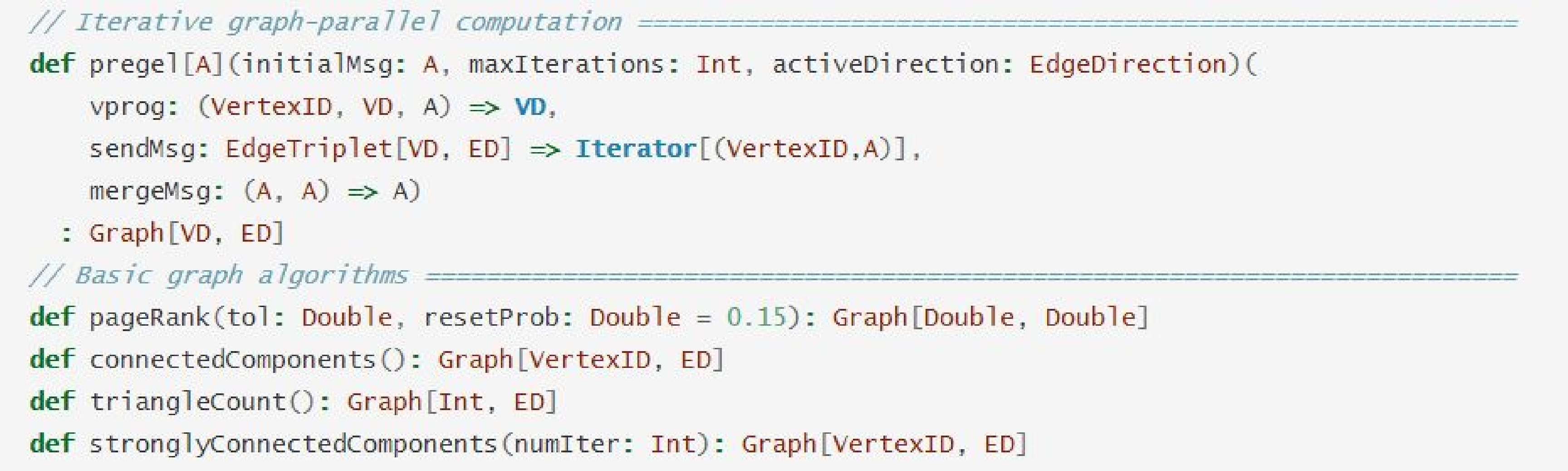




```
// Modify the graph structure ===============================================================
def reverse: Graph[VD, ED]
def subgraph(
    epred: EdgeTriplet[VD,ED] => Boolean = (x => true),
    vpred: (VertexID, VD) => Boolean = ((v, d) => true))
  : Graph[VD, ED]
def mask[VD2, ED2](other: Graph[VD2, ED2]): Graph[VD, ED]
def groupEdges(merge: (ED, ED) => ED): Graph[VD, ED]
// 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 collectNeighborIds(edgeDirection: EdgeDirection): VertexRDD[Array[VertexID]]
def collectNeighbors(edgeDirection: EdgeDirection): VertexRDD[Array[(VertexID, VD)]]
def aggregateMessages[Msg: ClassTag](
    sendMsg: EdgeContext[VD, ED, Msg] => Unit,
    mergeMsg: (Msg, Msg) => Msg,
    tripletFields: TripletFields = TripletFields.All)
  : VertexRDD[A]
```

And now? (4)

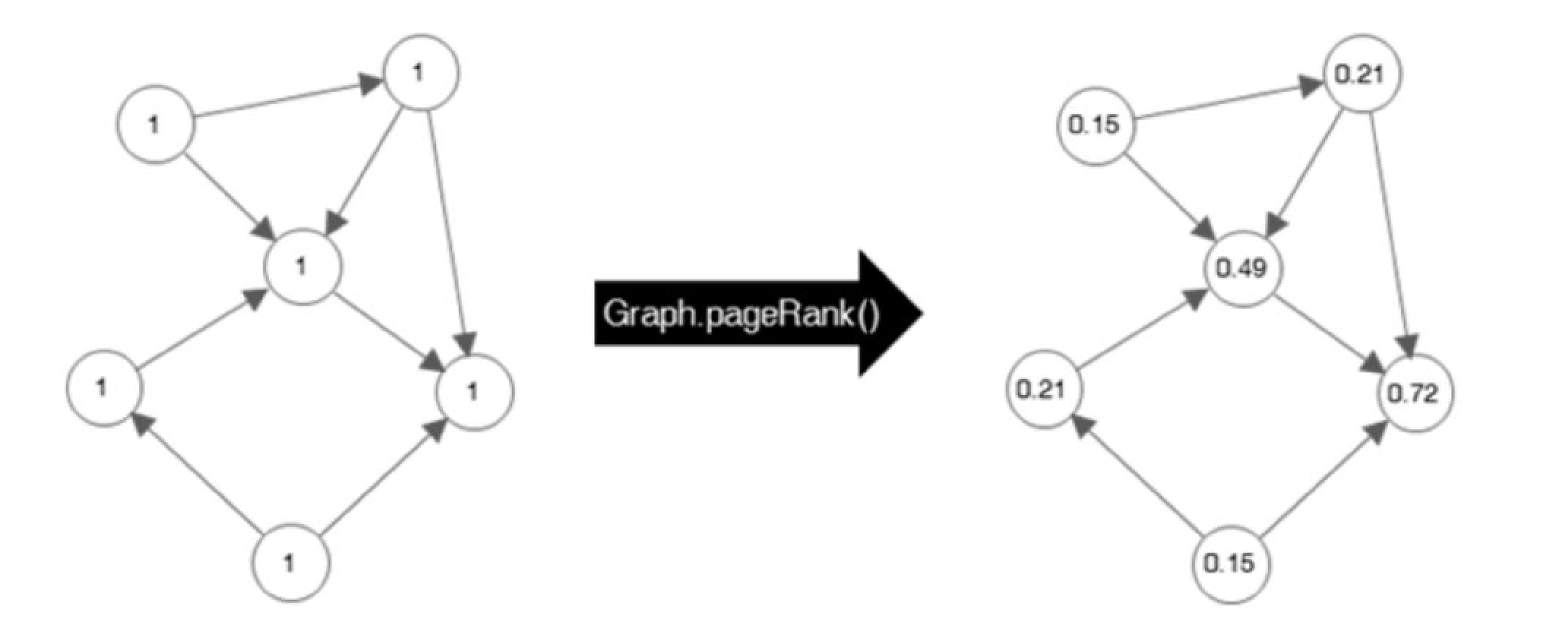




```scala
// Iterative graph-parallel computation ==========================================================
def pregel[A](initialMsg: A, maxIterations: Int, activeDirection: EdgeDirection)(
    vprog: (VertexID, VD, A) => VD,
    sendMsg: EdgeTriplet[VD, ED] => Iterator[(VertexID,A)],
    mergeMsg: (A, A) => A)
  : Graph[VD, ED]
// 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]
```

PageRank

$$\textit{PageRank of site} = \sum \frac{\textit{PageRank of inbound link}}{\textit{Number of links on that page}}$$



Demo

Where can I find out more about GraphX?

https://github.com/adrianulbona/graphx-example

http://spark.apache.org/docs/latest/graphx-programming-guide.html

https://kowshik.github.io/JPregel/pregel_paper.pdf

https://www.manning.com/books/spark-graphx-in-action