Apache Spark GraphX

By **Roy Levin**

http://researcher.watson.ibm.com/researcher/view.php?person=il-ROYL

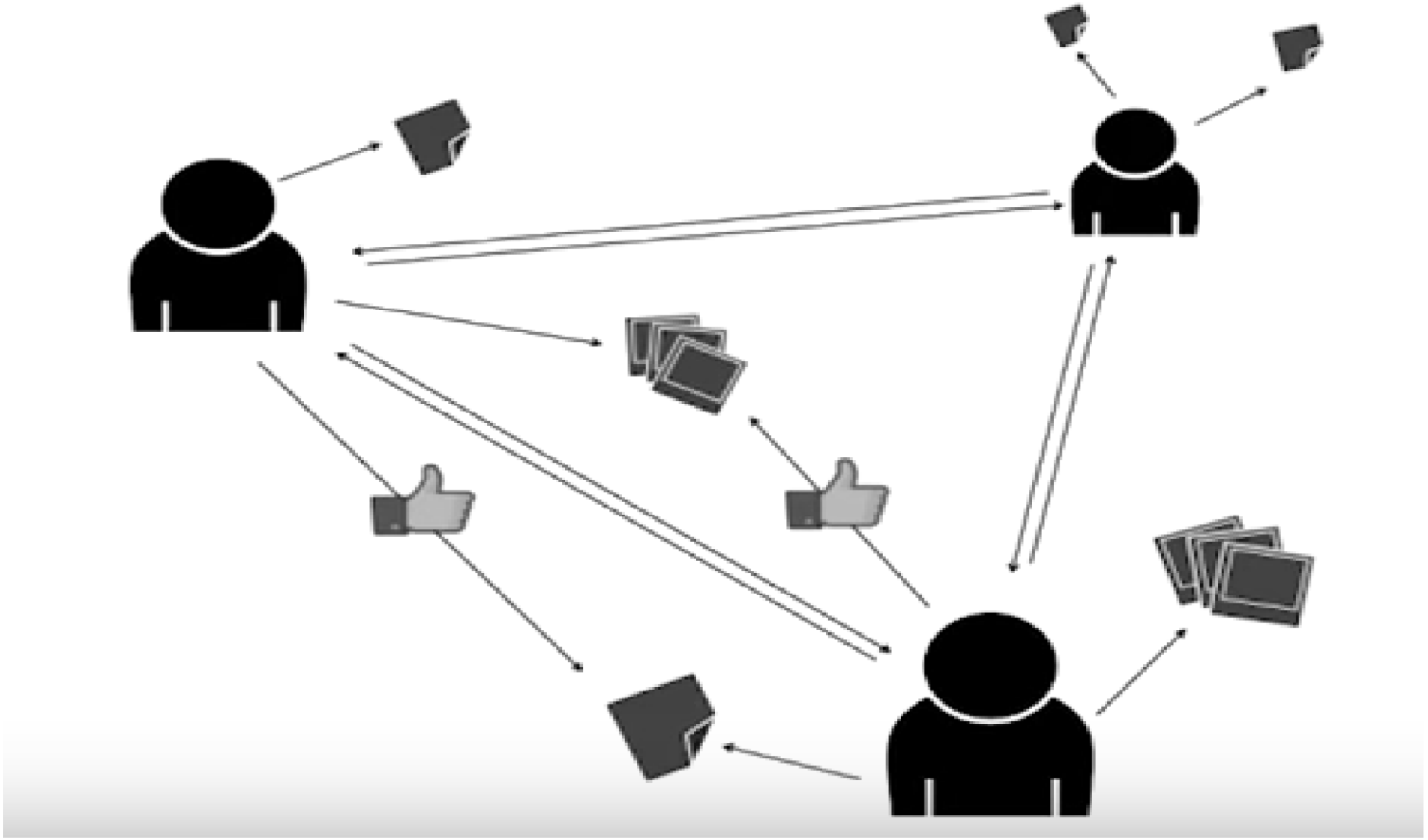
Agenda

- **An Intro to Graph Processing**
- Why GraphX?
- How (to use) GraphX?
- How GraphX Works

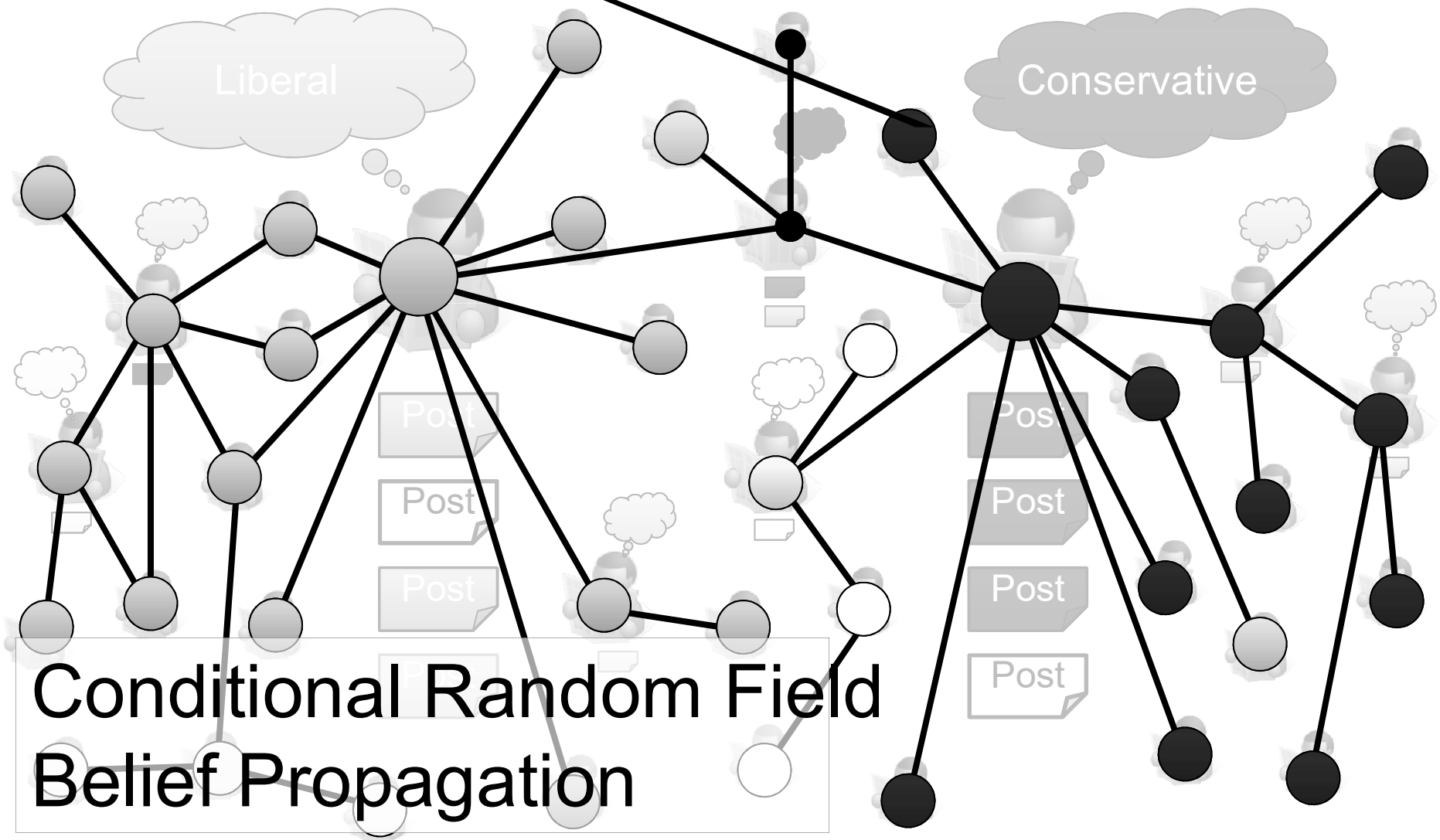
Graphs are Essential to Data Mining and Machine Learning

- Identify influential people and information
- Find communities
- Understand people’s shared interests
- Model complex data dependencies

Example: Social Networks

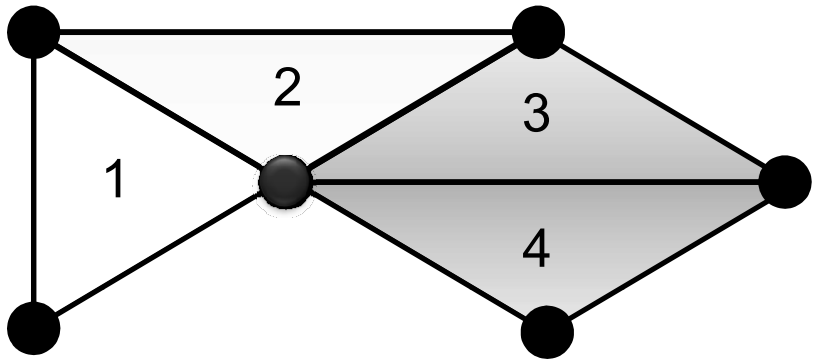


a la gd a d a



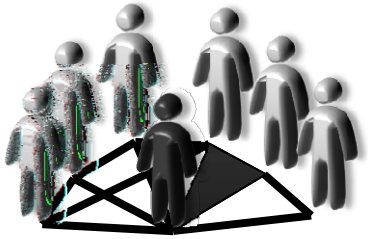
Conditional Random Field
Belief Propagation

Triangle Counting

- Count triangles passing through each vertex
- 

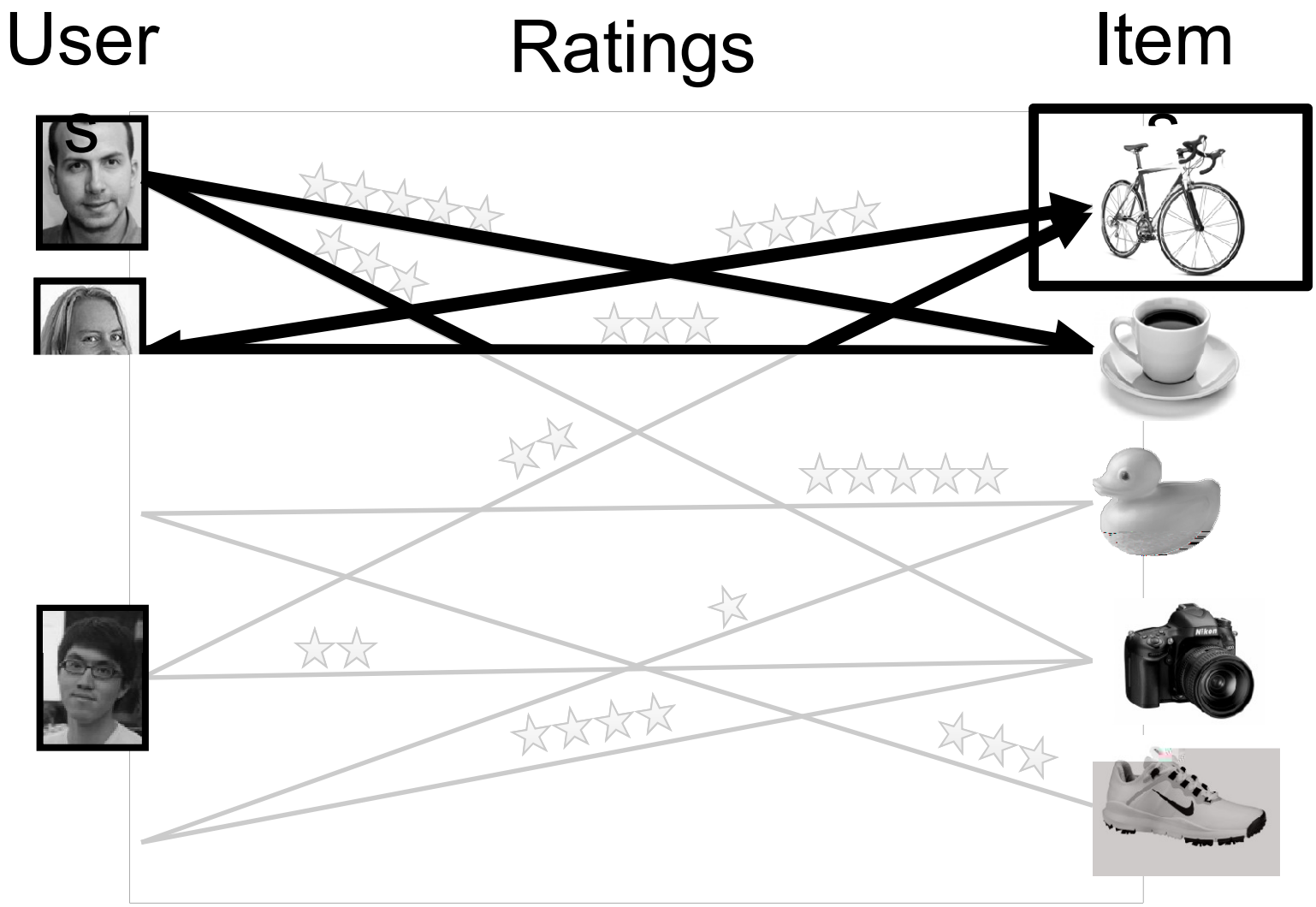
- Measures **cohesiveness** of local community

Fewer Triangles
Weaker Community



More Triangles
Stronger Community

Collaborative Filtering



Pregel

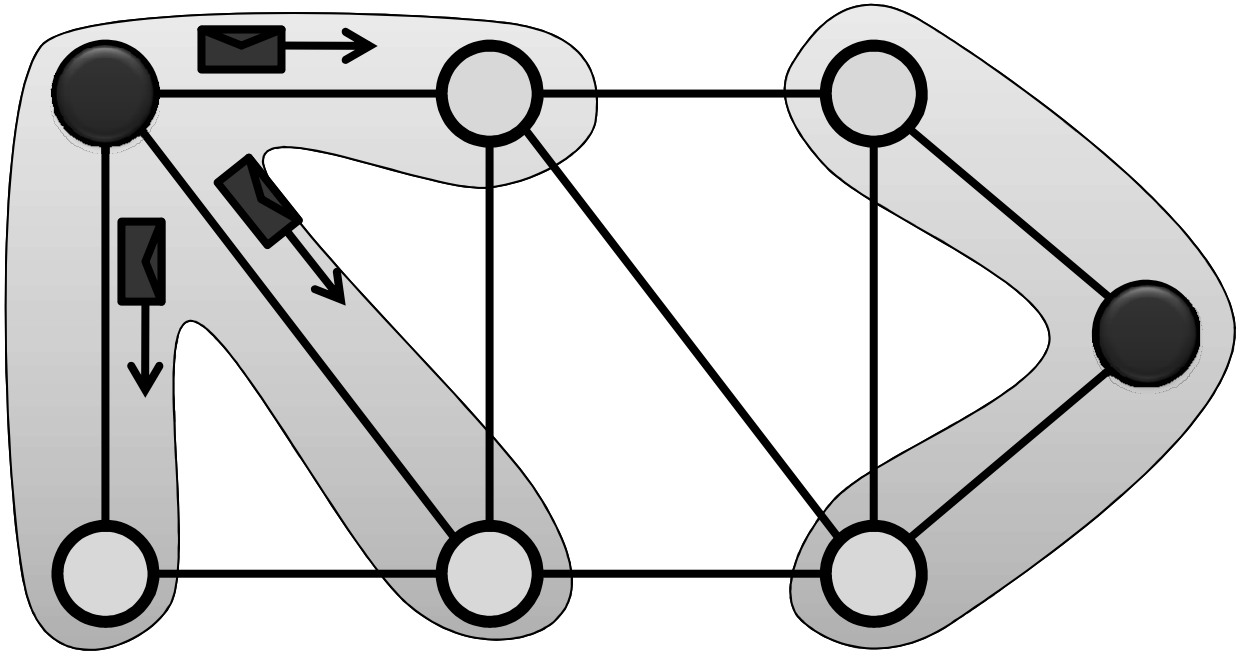
- Vertex-centric
- Runs in sequence of iterations called **super-steps**
- A vertex in super-step *a* can:
  - Read messages sent to it in super-step *a*
  - Send messages to other vertices (they will receive the messages in step *a*)
  - Modify its current state

The **Graph-Parallel** Abstraction

A user-defined **Vertex-Program** runs on each vertex

Graph constrains **interaction** along edges

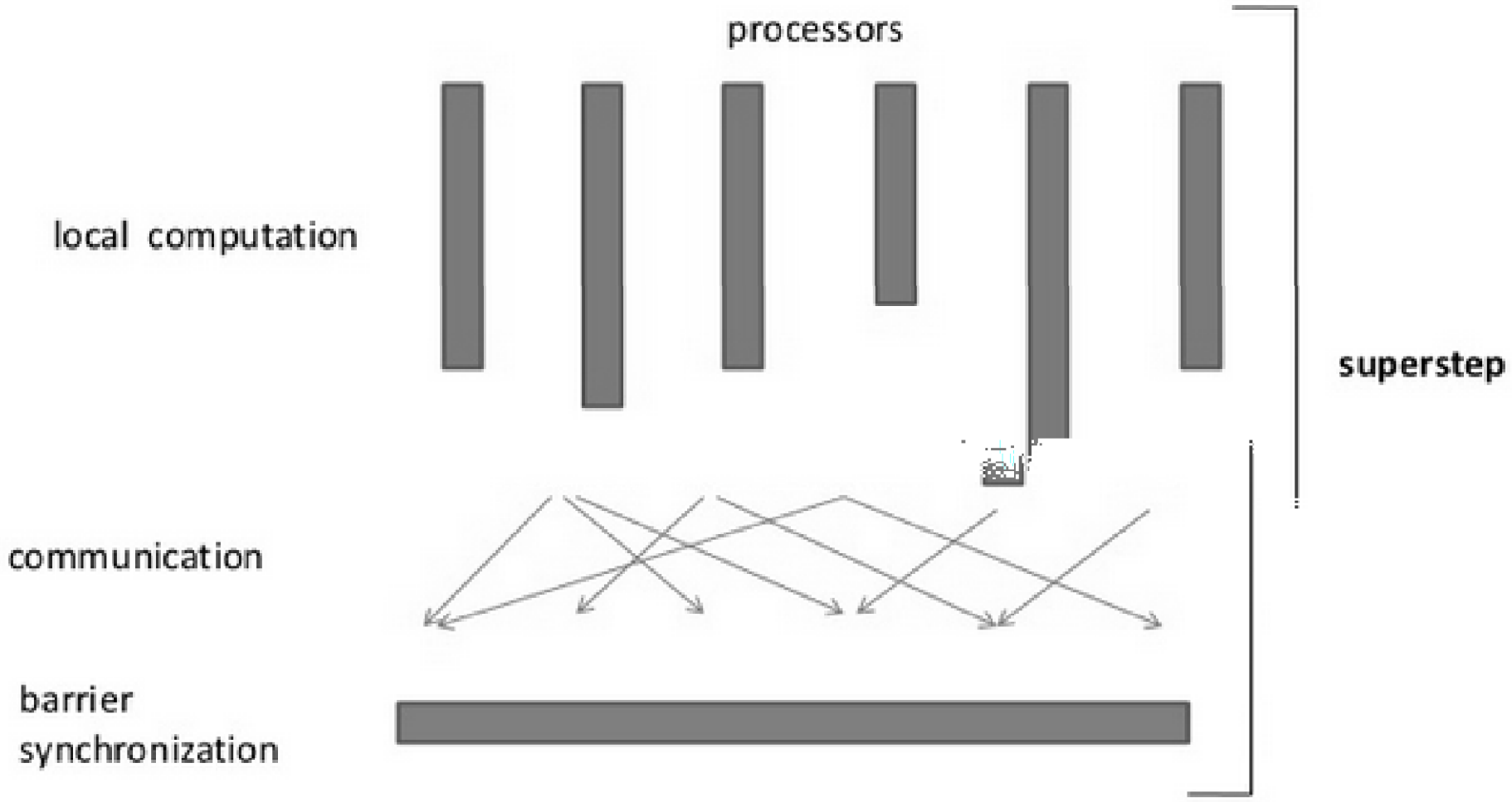
Using **messages** (e.g. **Pregel** [PODC'09, SIGMOD'10])



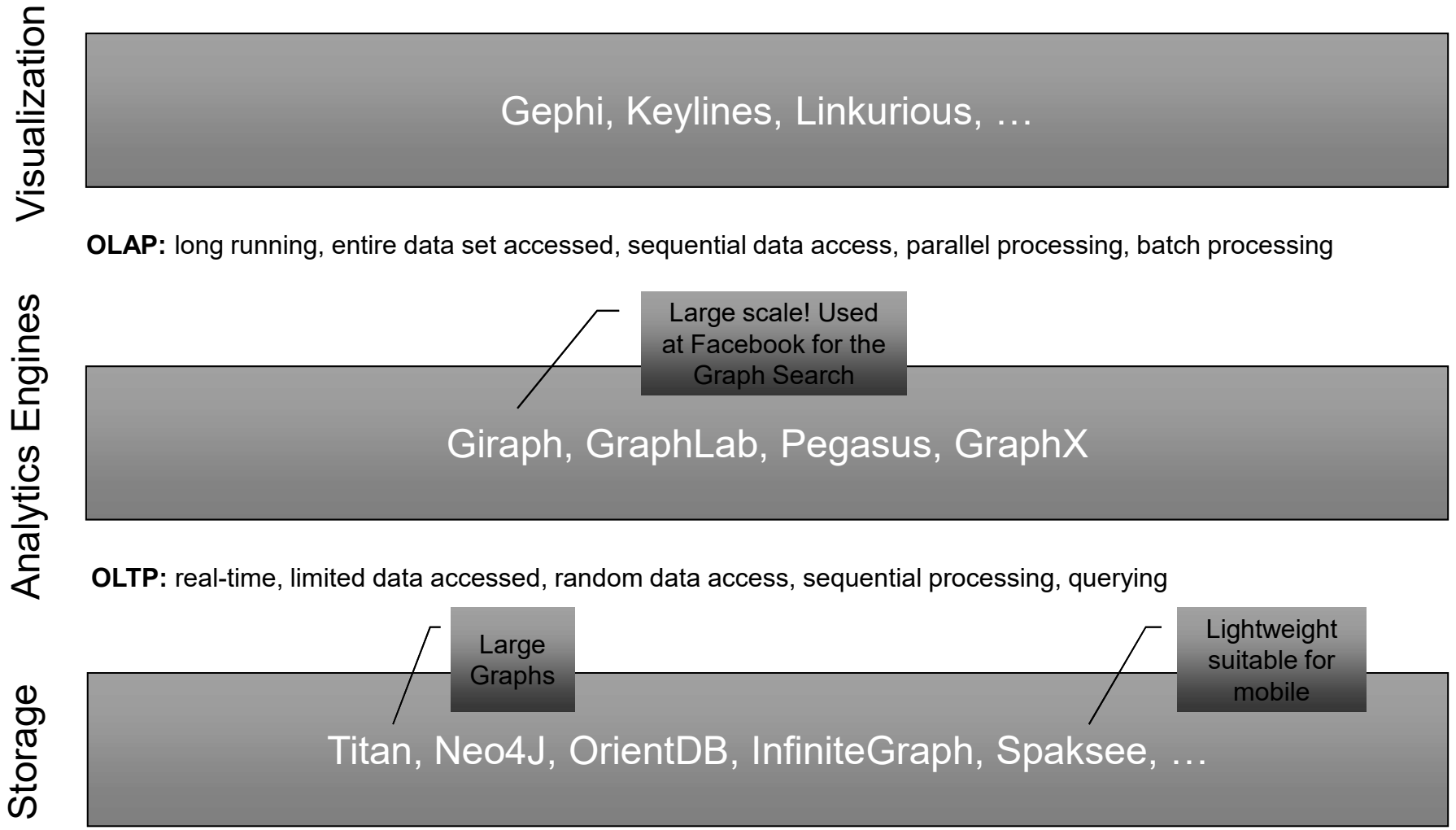
Parallelism: run multiple vertex programs simultaneously

Computing PageRank the **Pregel** way

Bulk Synchronous Parallel



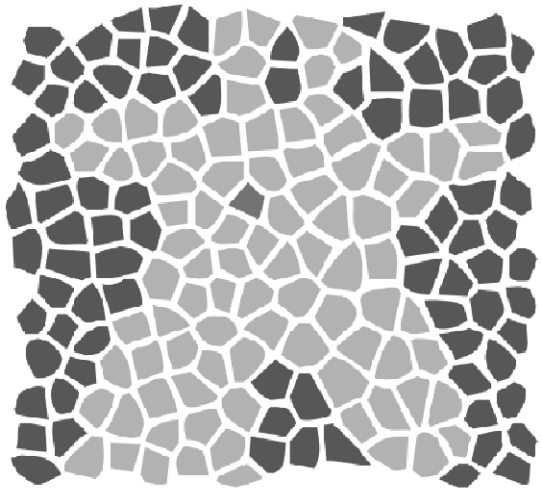
Graph Technology Components

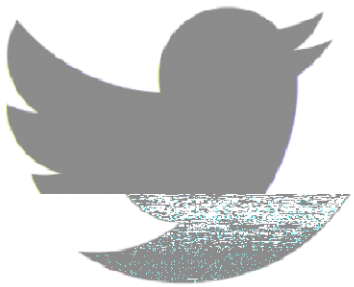


Agenda

- An Intro to Graph Processing
- **Why GraphX?**
- How (to use) GraphX?
- How GraphX Works

Specialized Graph Systems

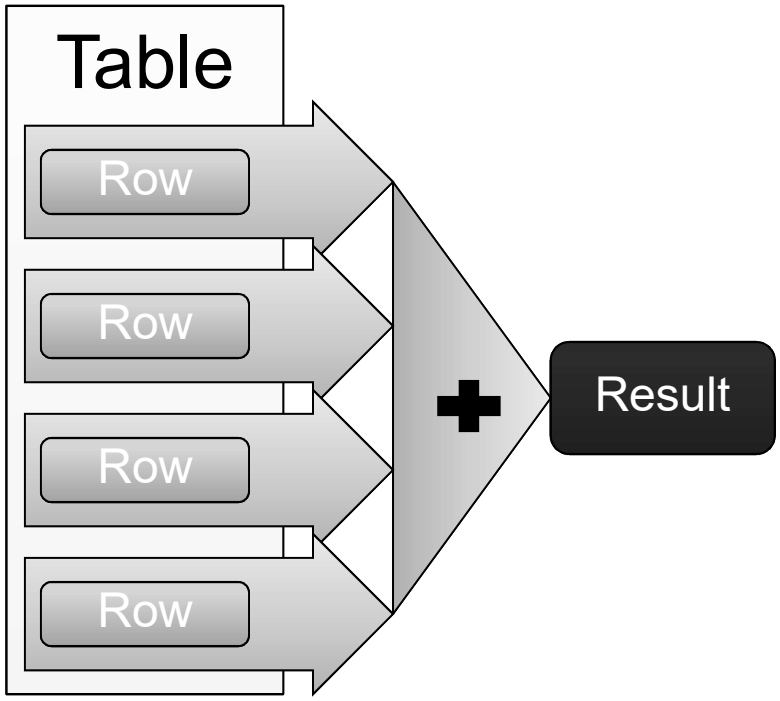




Google Pregel

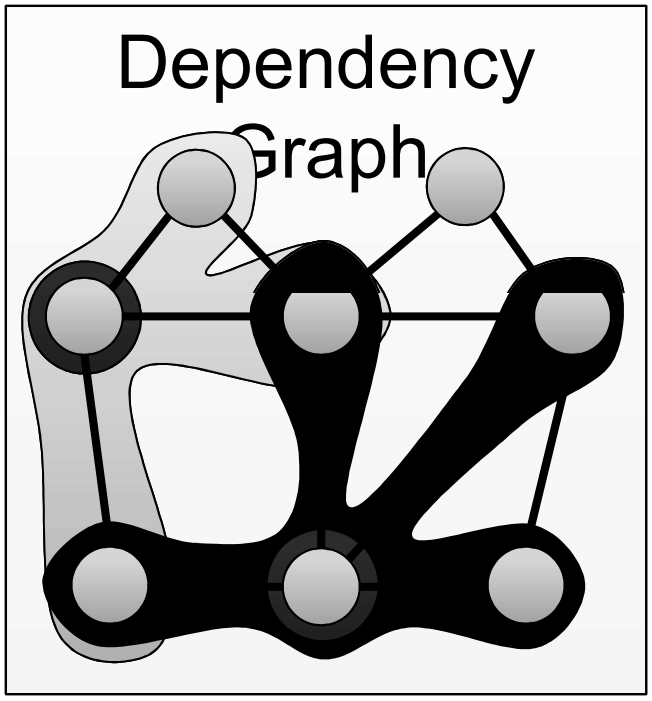
A Disconnect Between Data and Graph Processing

Data-Parallel



hadoop

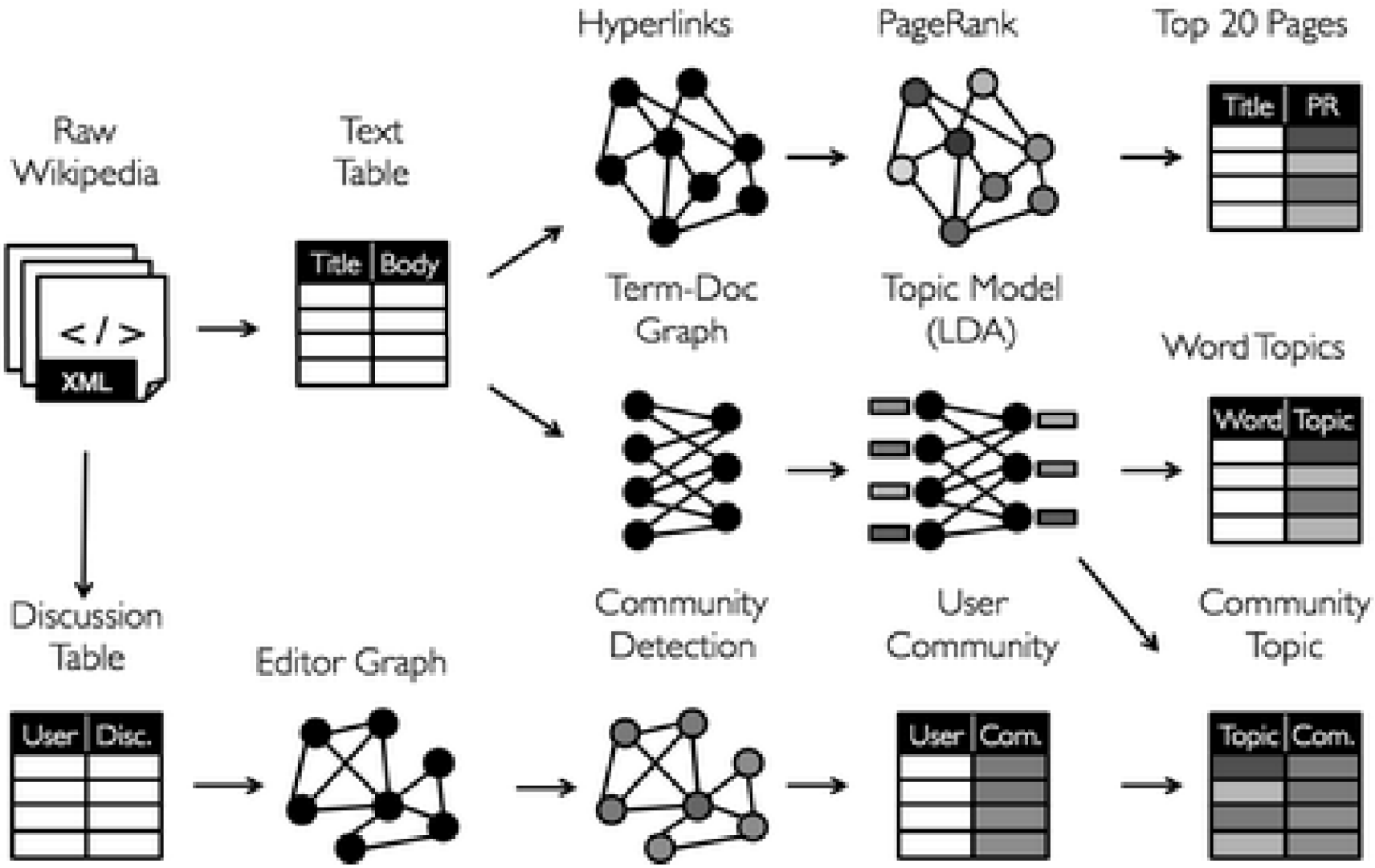
Graph-Parallel



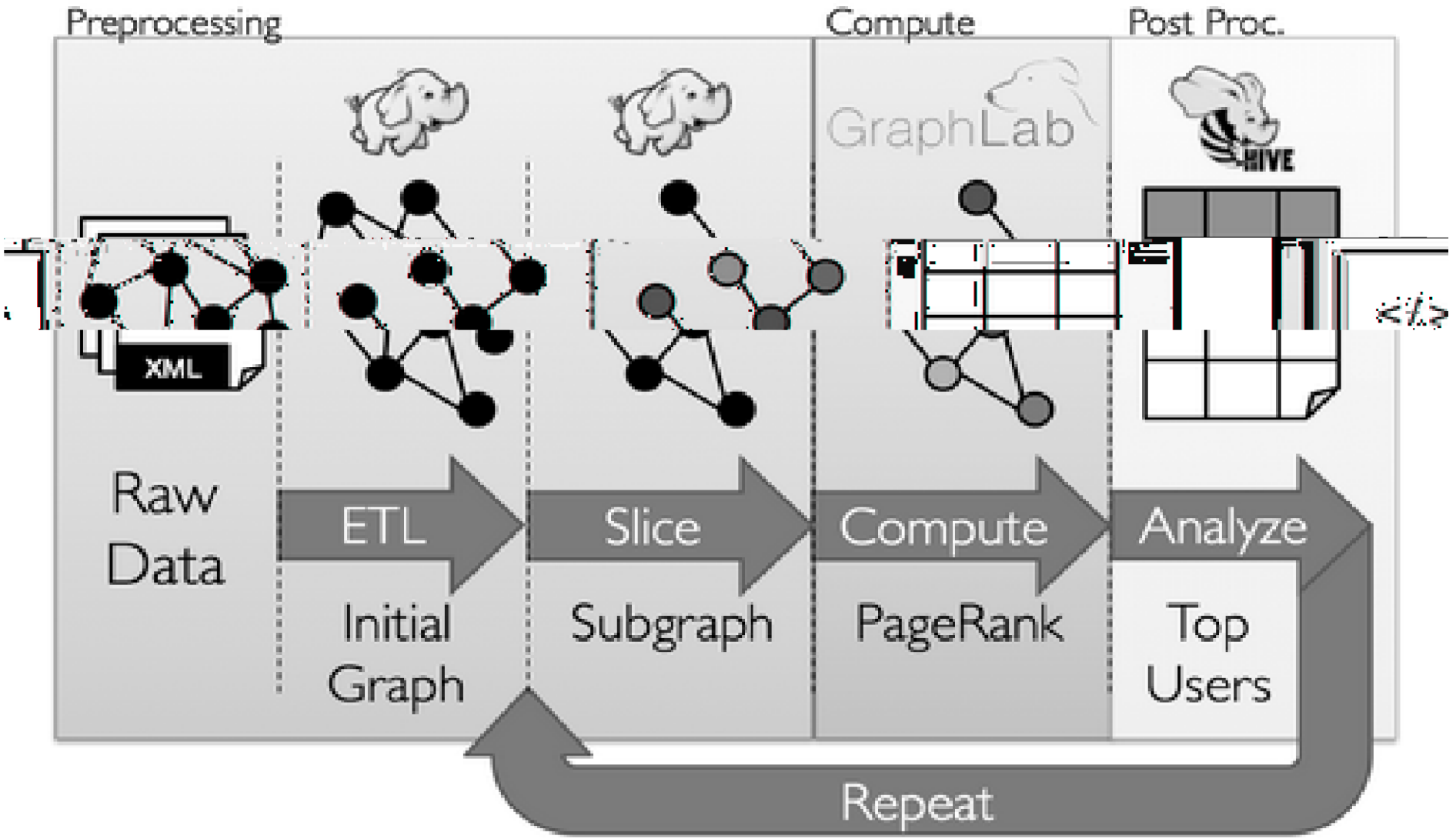
Pregel

GraphLab

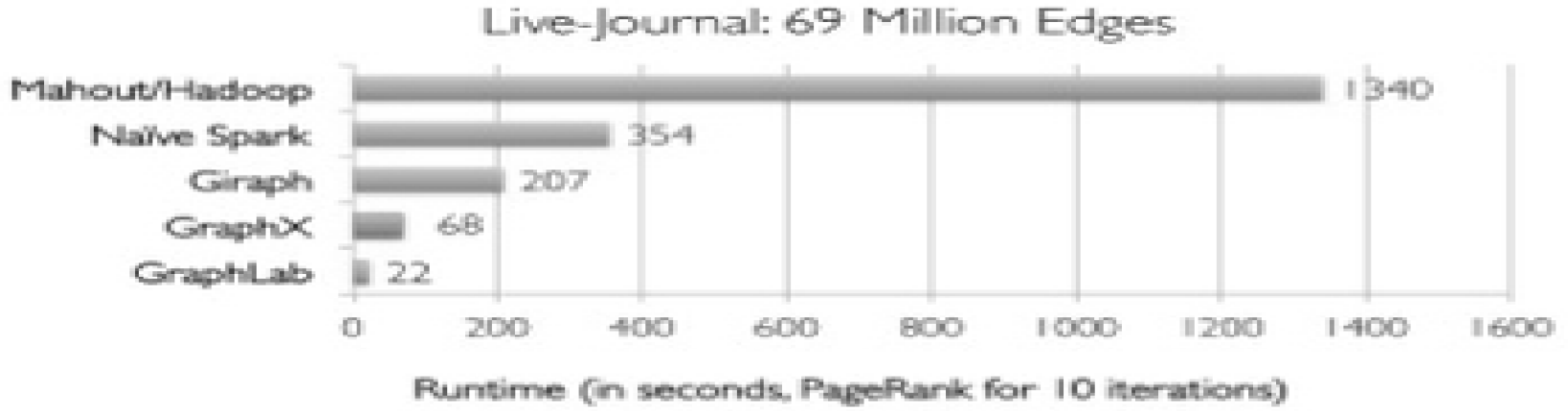
Typical Graph Processing Pipeline

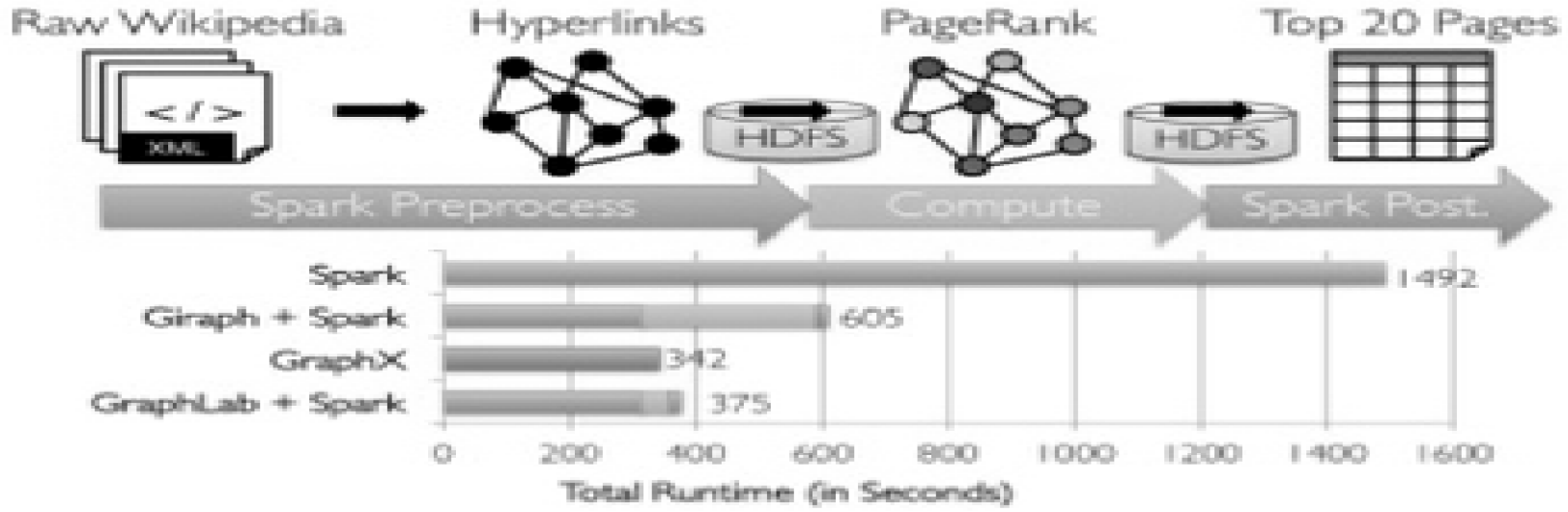


An Example of this Pipeline



Runtime Comparisons





Agenda

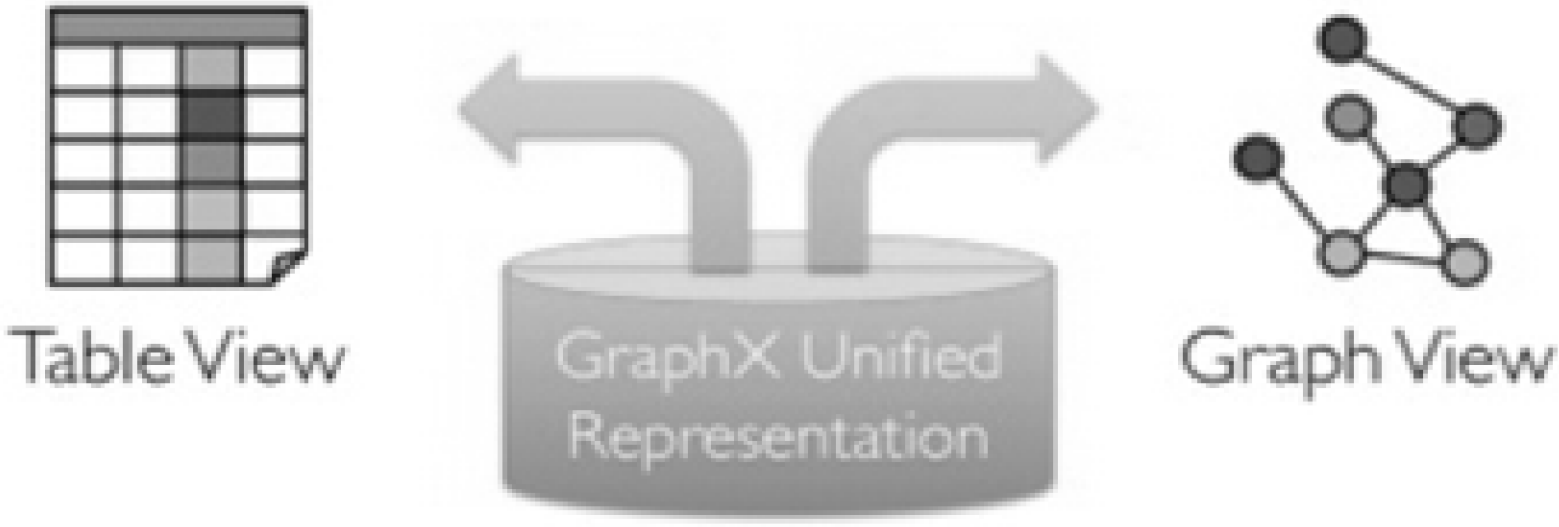
- An Intro to Graph Processing
- Why GraphX?
- **How (to use) GraphX?**
- How GraphX Works

GraphX Design Goals

- Blur distinction between tables & graphs
- Unify data & graph parallel processing
- Implement on top of Spark

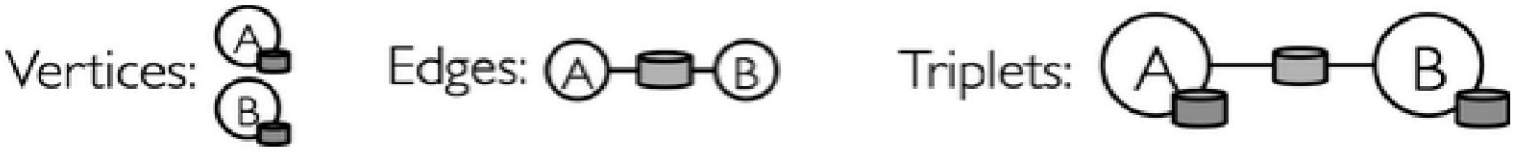
GraphX Key Idea

- Tables & Graphs are composable views of the same physical data
- View support operations that exploit semantics of the view to achieve efficient execution



GraphX Structure

- VertexRDD maps IDs to vertex content
- EdgeRDD are of the form (ID1, ID2, ET)
- Triplets are a combination of Vertex & Edge RDDs



*** Some graph frameworks assume that the edges of a single vertex can fit in a single machine --- GraphX does not make this assumption!**

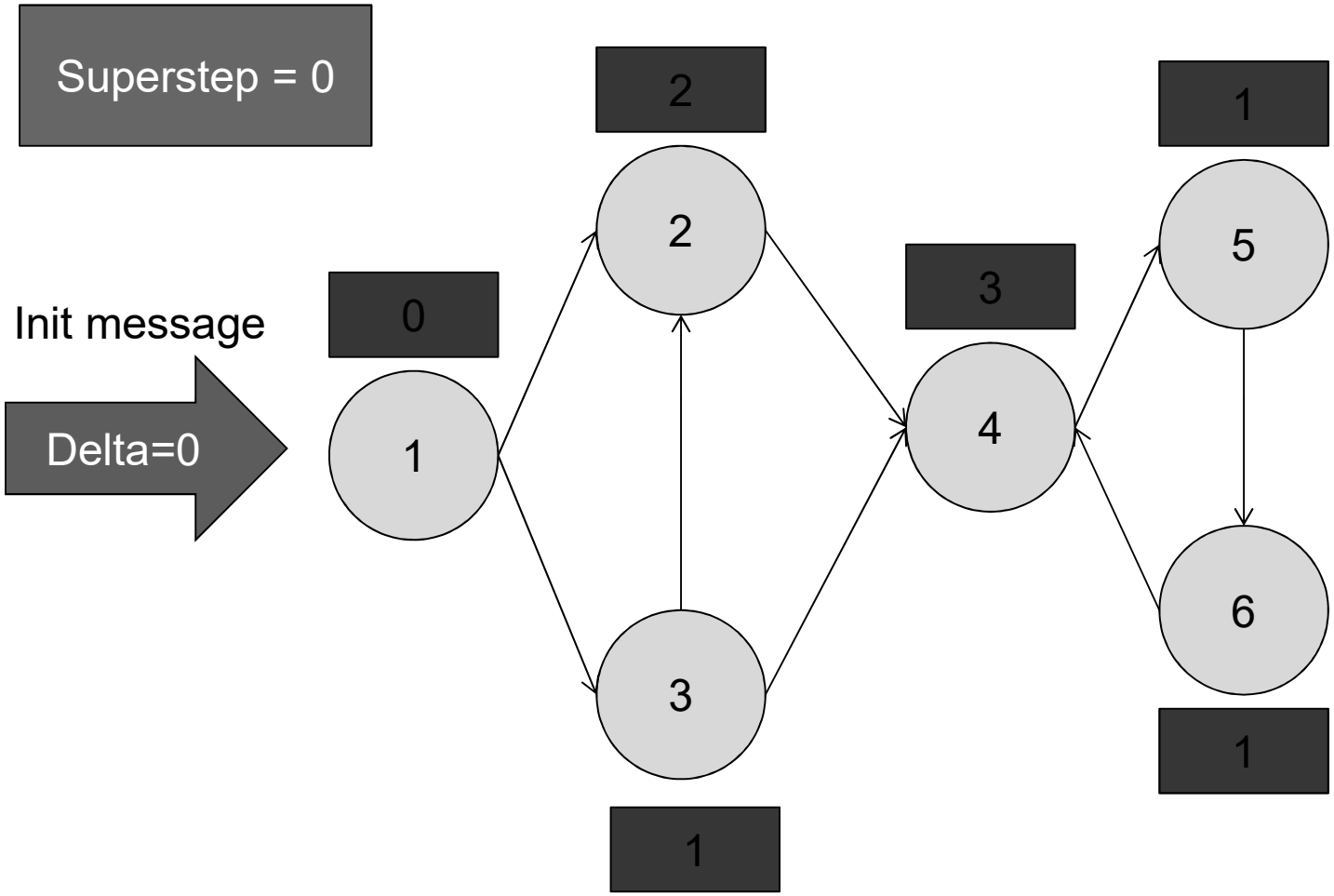
GraphX Structure

```
// Vertex collection
class VertexRDD[VD] extends RDD[(VertexId, VD)]

// Edge collection
class EdgeRDD[ED] extends RDD[Edge[ED]]
case class Edge[ED](srcId: VertexId = 0, dstId: VertexId = 0,
                    attr: ED = null.asInstanceOf[ED])

// Edge Triple
class EdgeTriplet[VD, ED] extends Edge[ED]
```

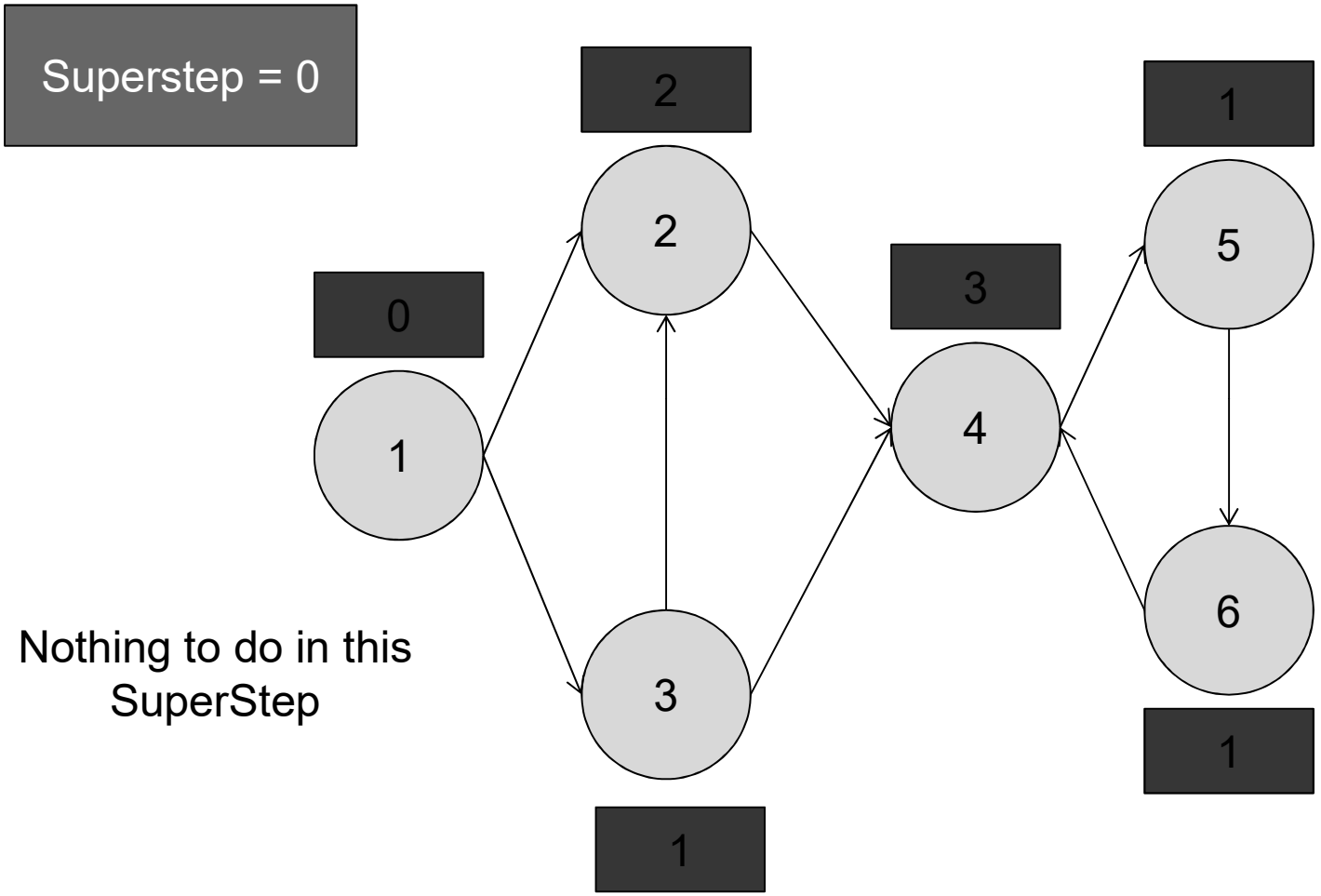
GraphX Example

```
val sc: SparkContext

// Create an RDD for the vertices
val users: VertexRDD[(String, String)] = sc.parallelize(
    Array((3L, ("rxin", "student")), (7L, ("jgonzal", "postdoc")),
          (5L, ("franklin", "prof")), (2L, ("istoica", "prof"))))

// Create an RDD for edges
val relationships: EdgeRDD[String] = sc.parallelize(
    Array(Edge(3L, 7L, "collab"),    Edge(5L, 3L, "advisor"),
          Edge(2L, 5L, "colleague"), Edge(5L, 7L, "pi")))

// Define a default user in case there are relationship with missing user
val defaultUser = ("John Doe", "Missing")

// Build the initial Graph
val userGraph: Graph[(String, String), String] =
    Graph(users, relationships, defaultUser)
```

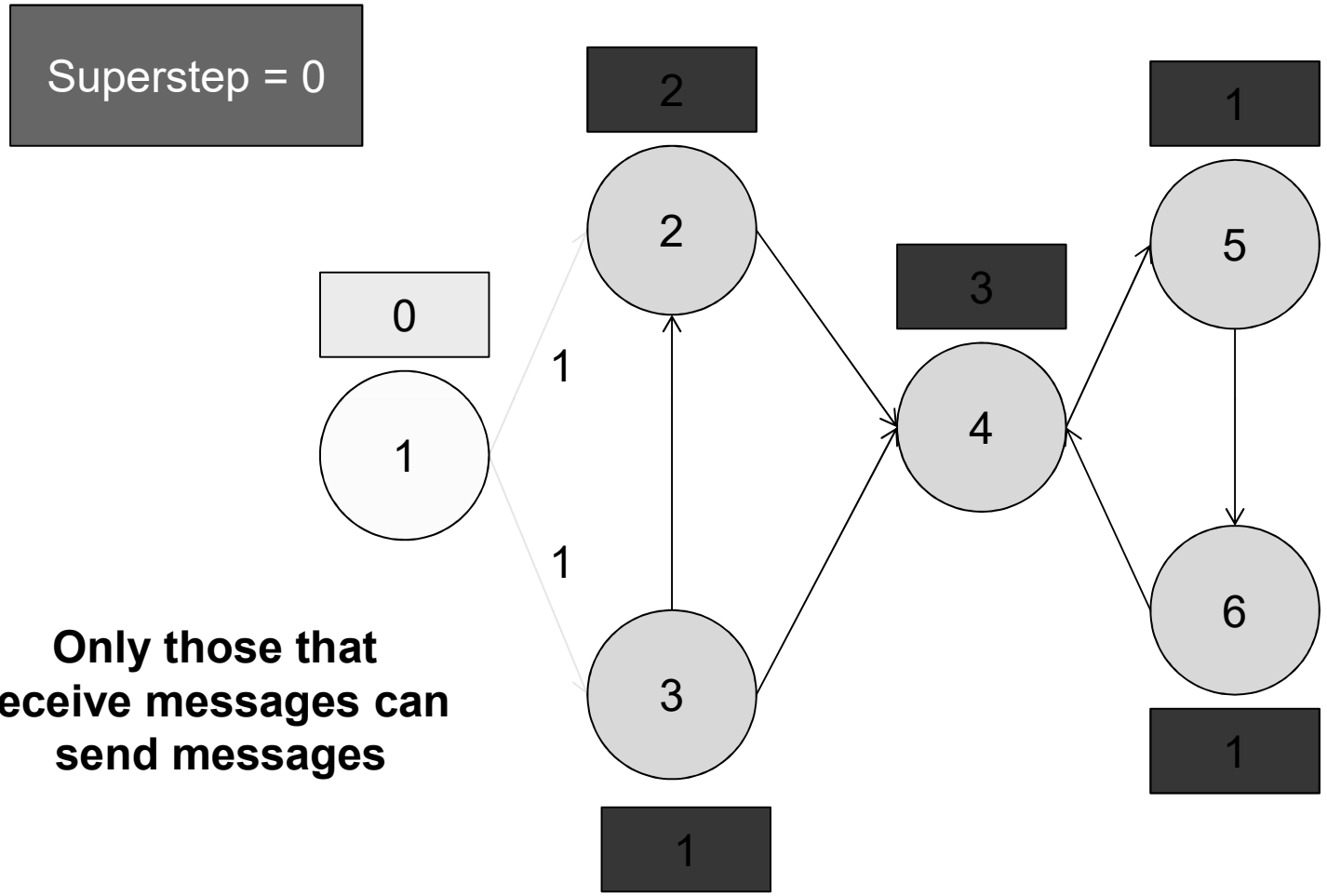
Is DAG Pregel Example



Is DAG Pregel: **vprog**

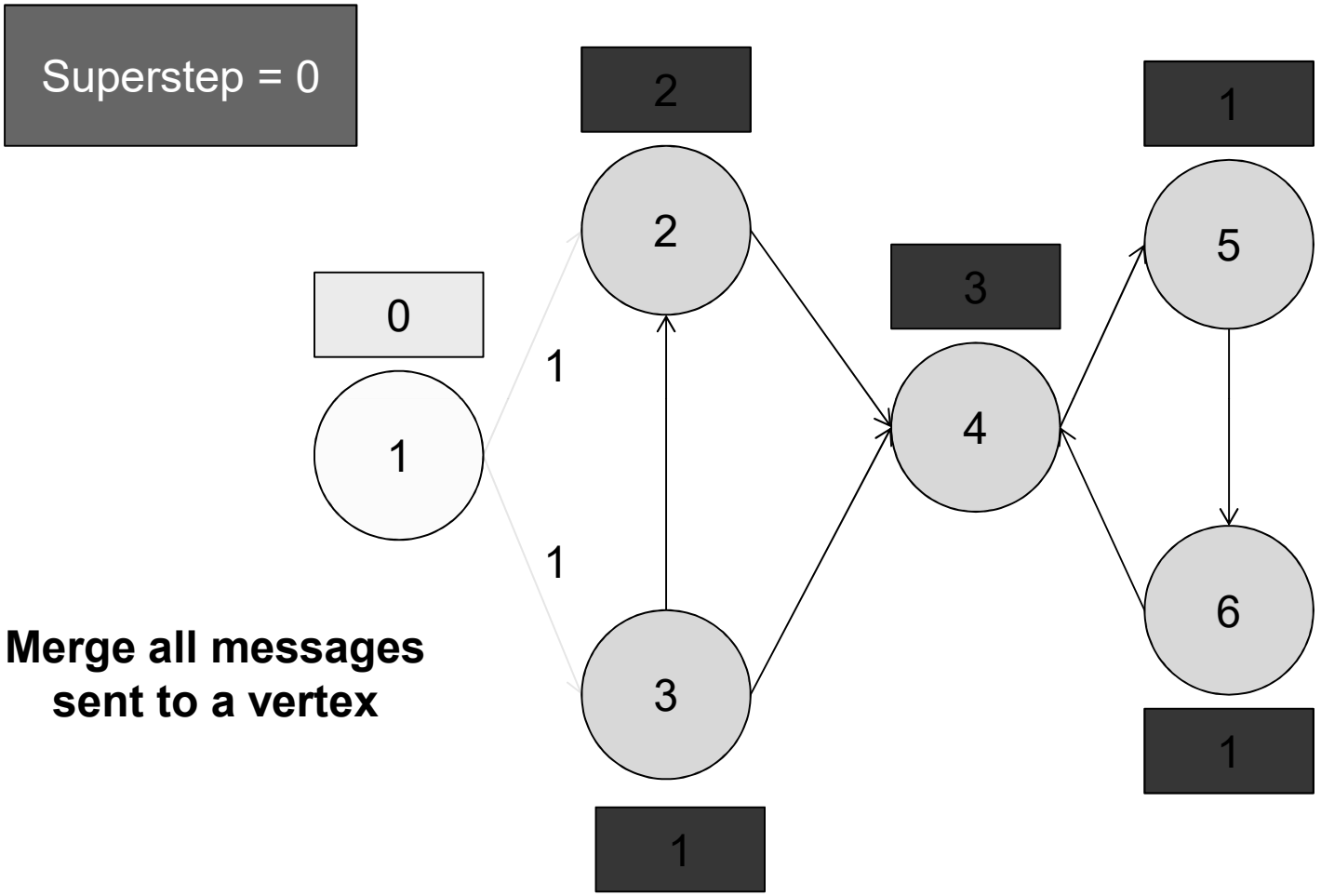


Is DAG Pregel: **sendMsg**

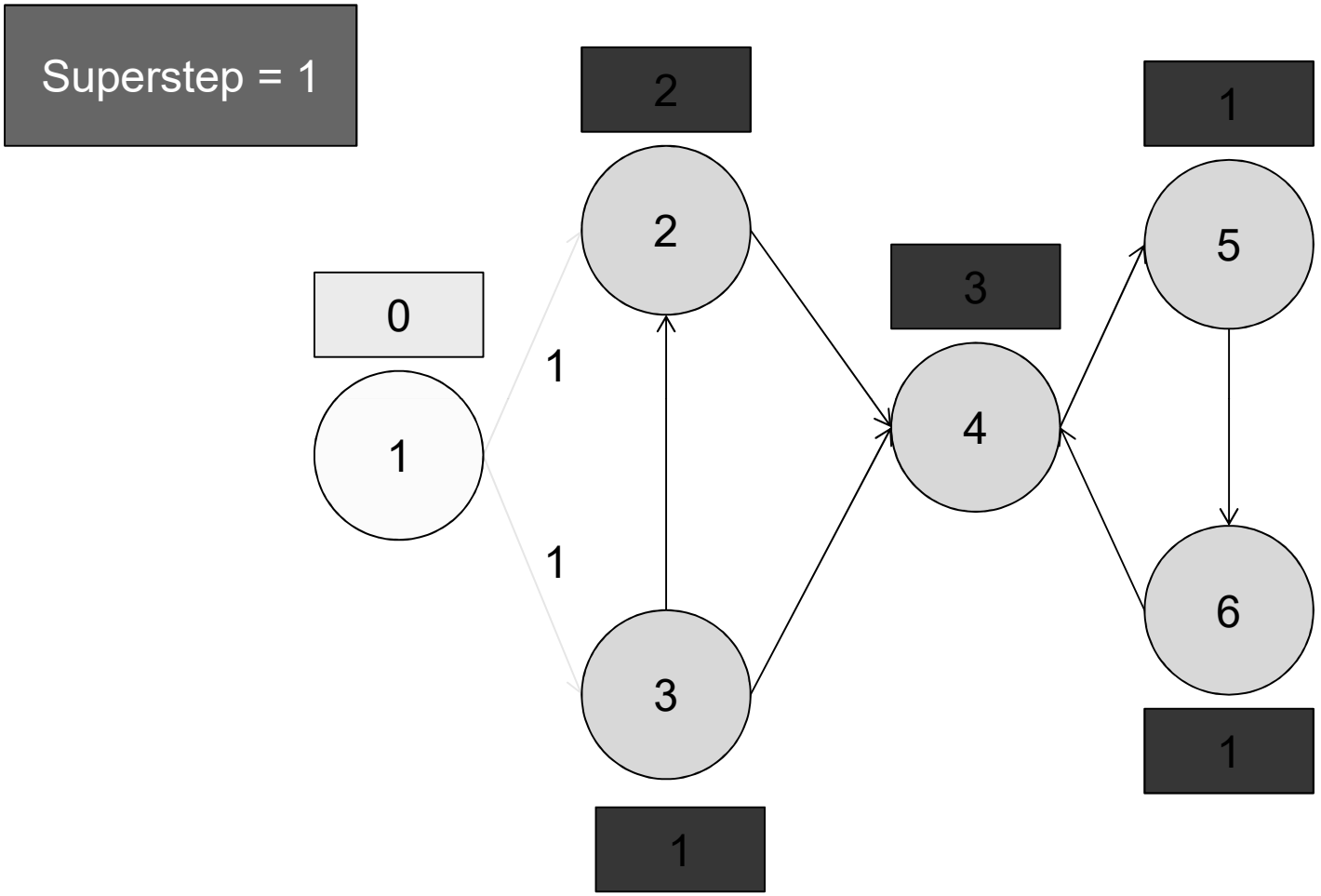


Only those that receive messages can send messages

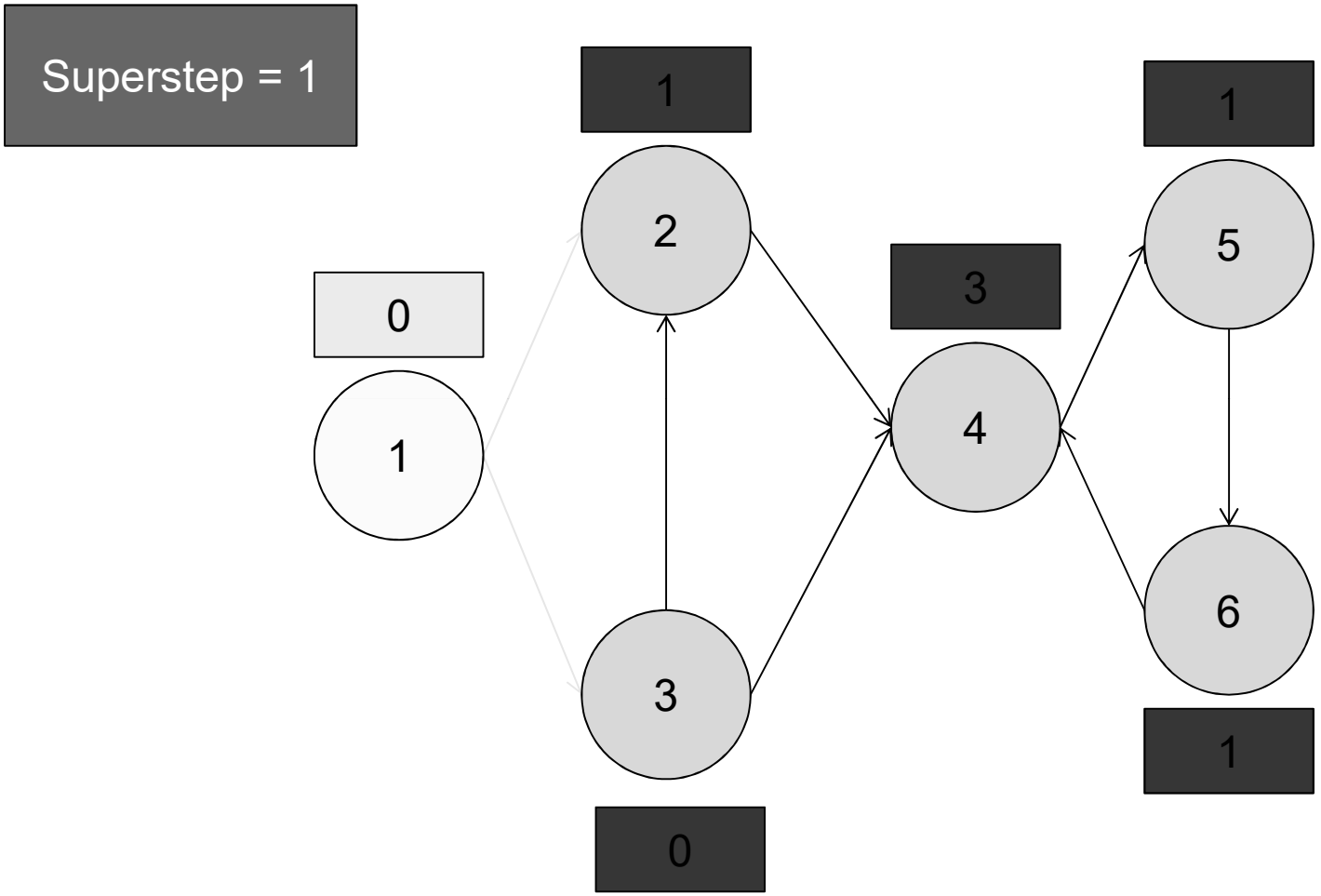
Is DAG Pregel: **mergeMsg**



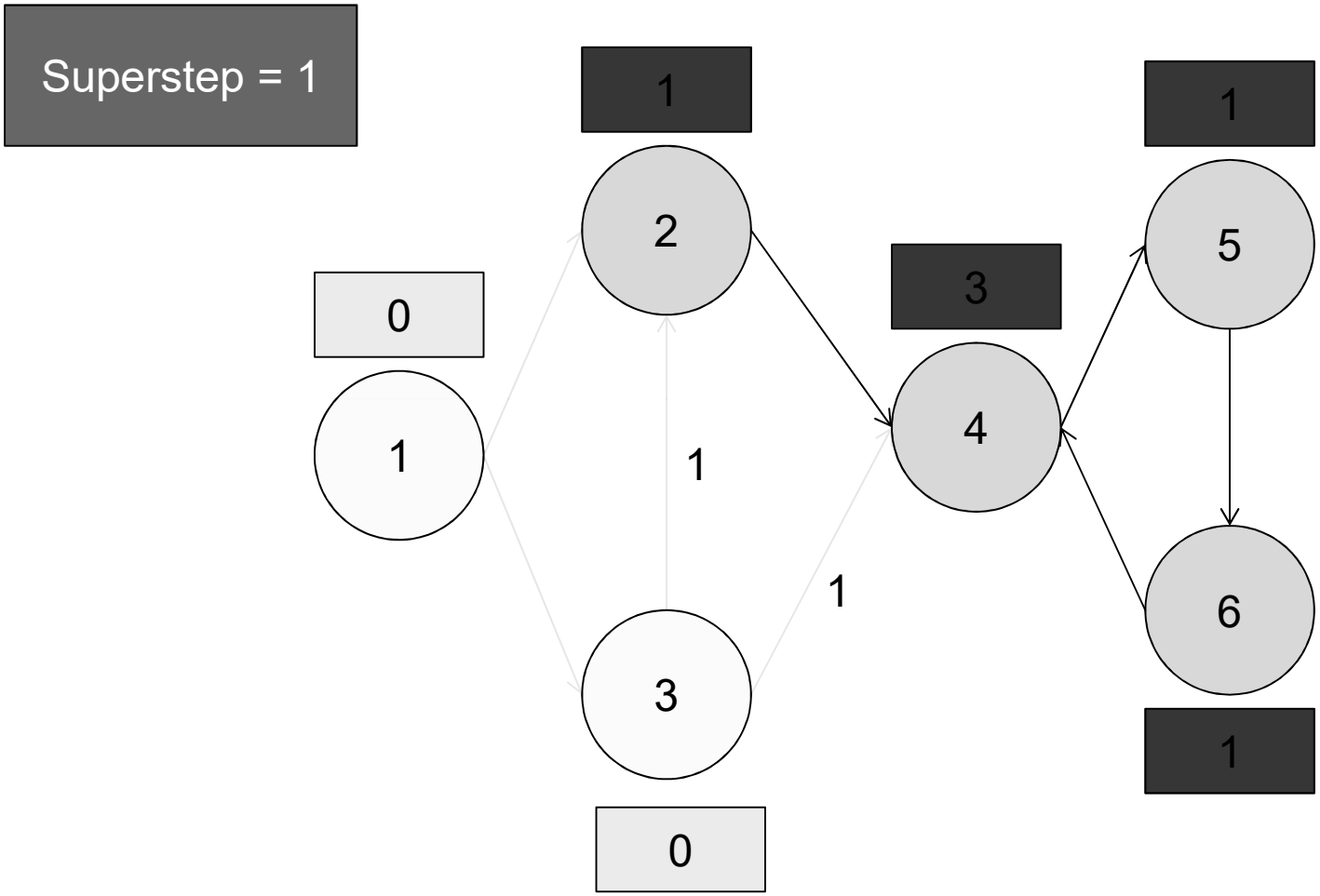
Is DAG Pregel: **vprog**



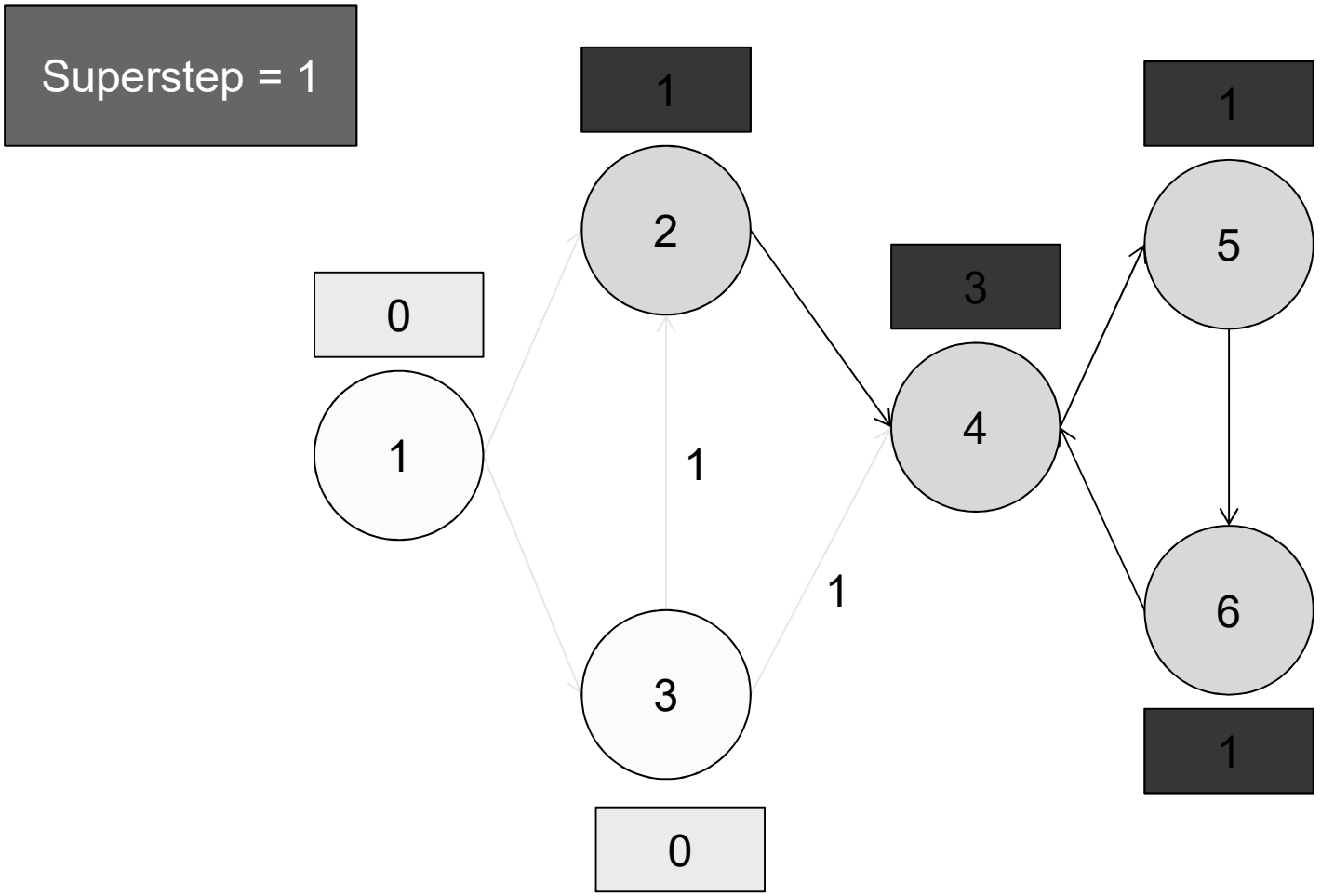
Is DAG Pregel: **vprog**



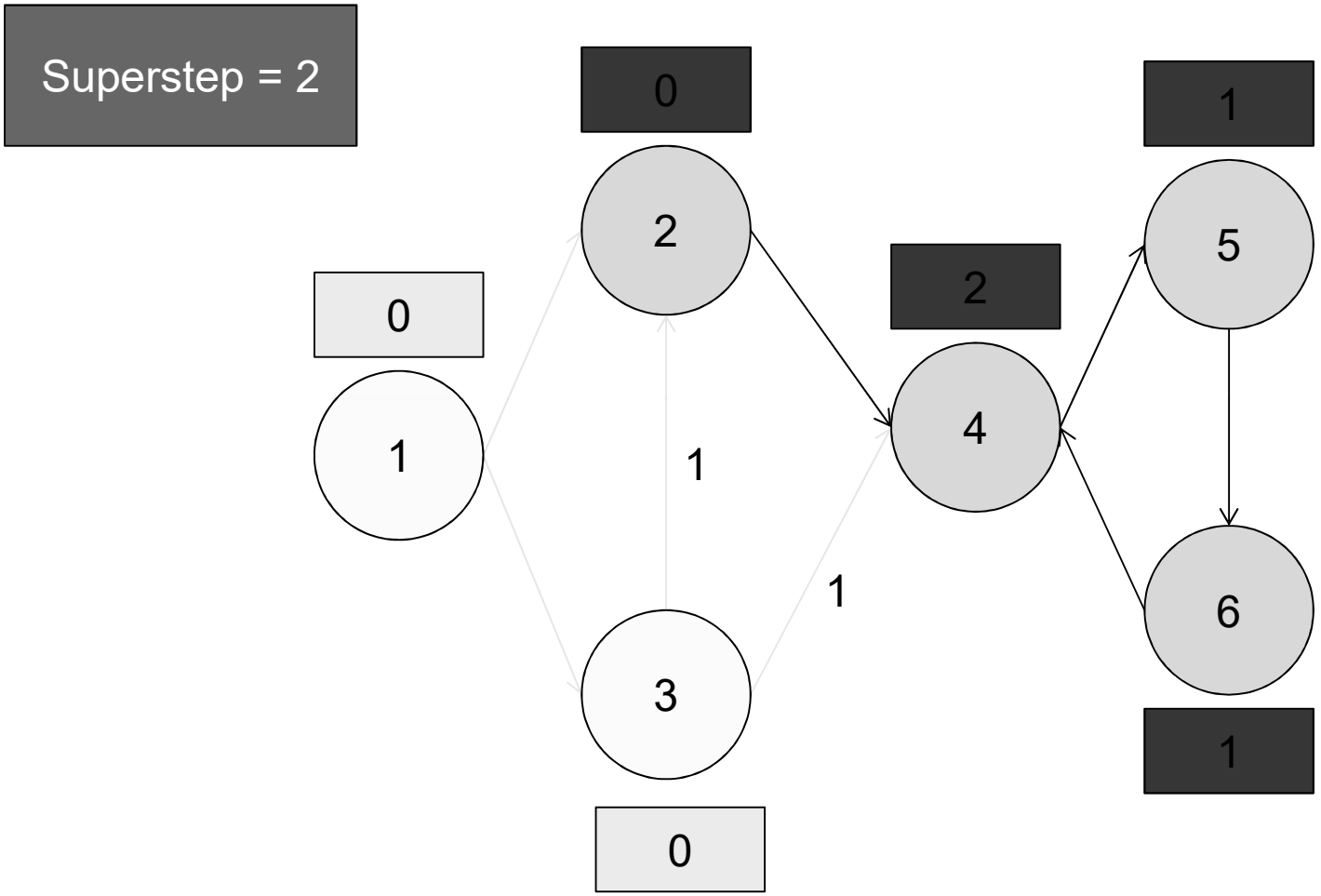
Is DAG Pregel: **sendMsg**



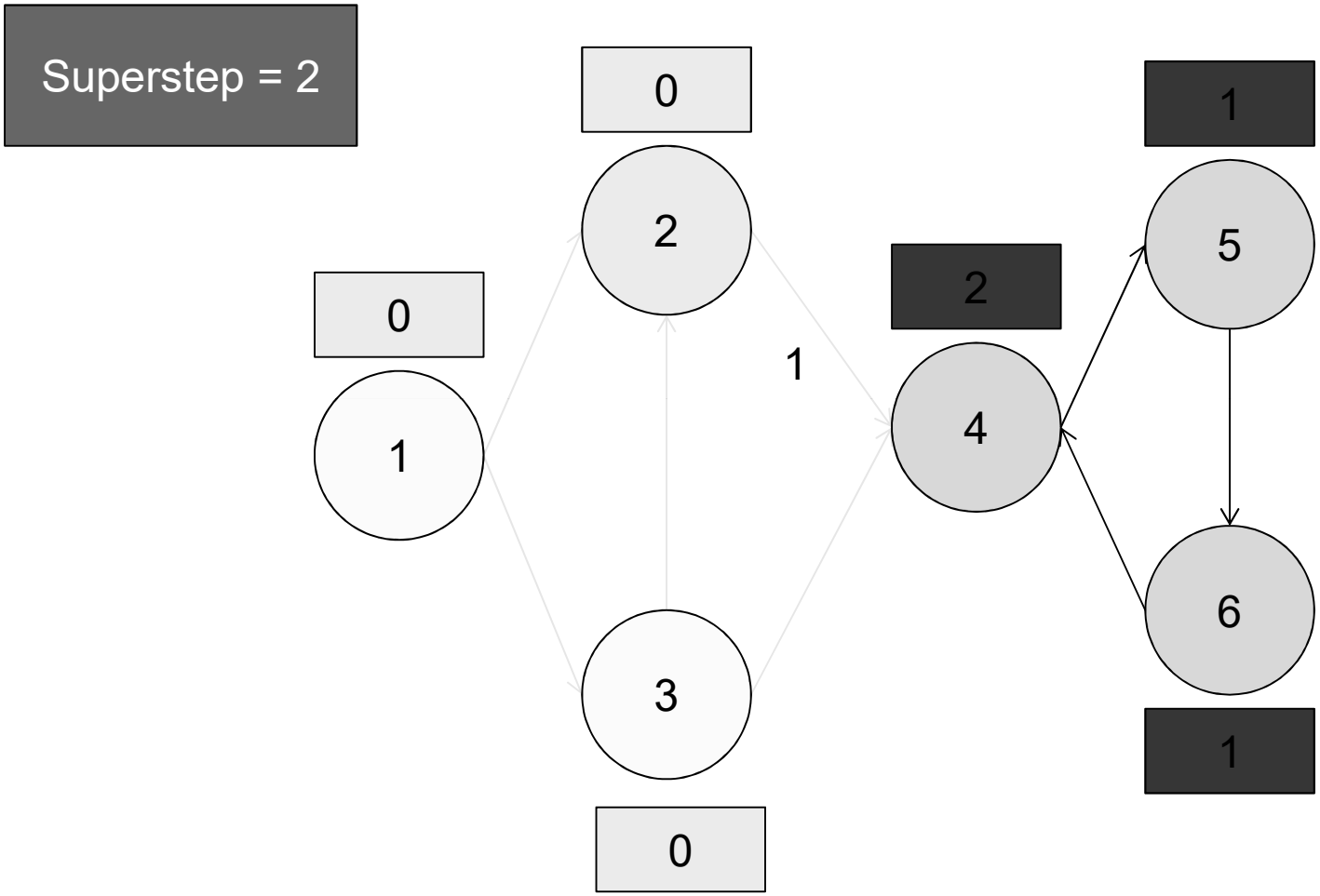
Is DAG Pregel: **mergeMsg**



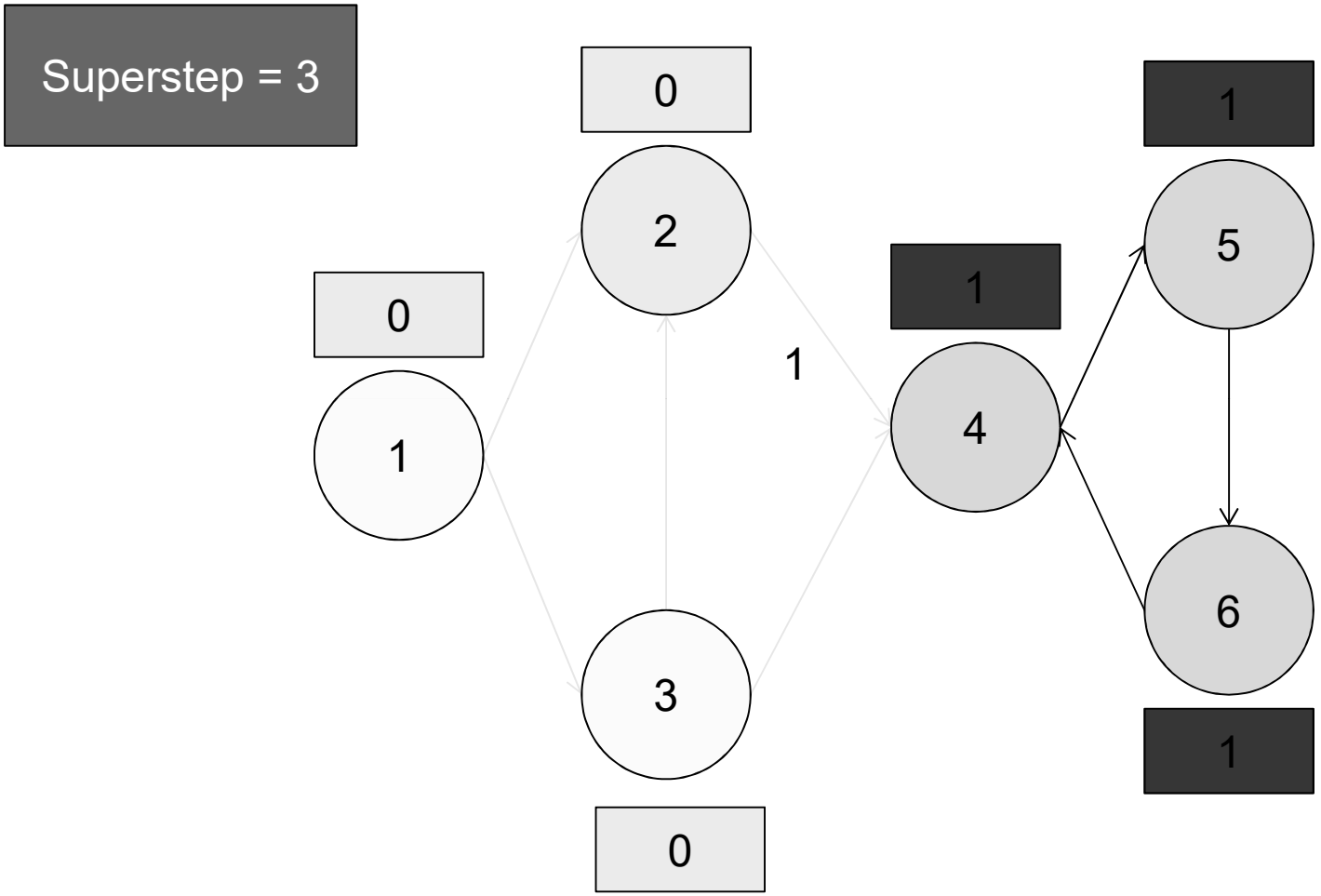
Is DAG Pregel: **vprog**



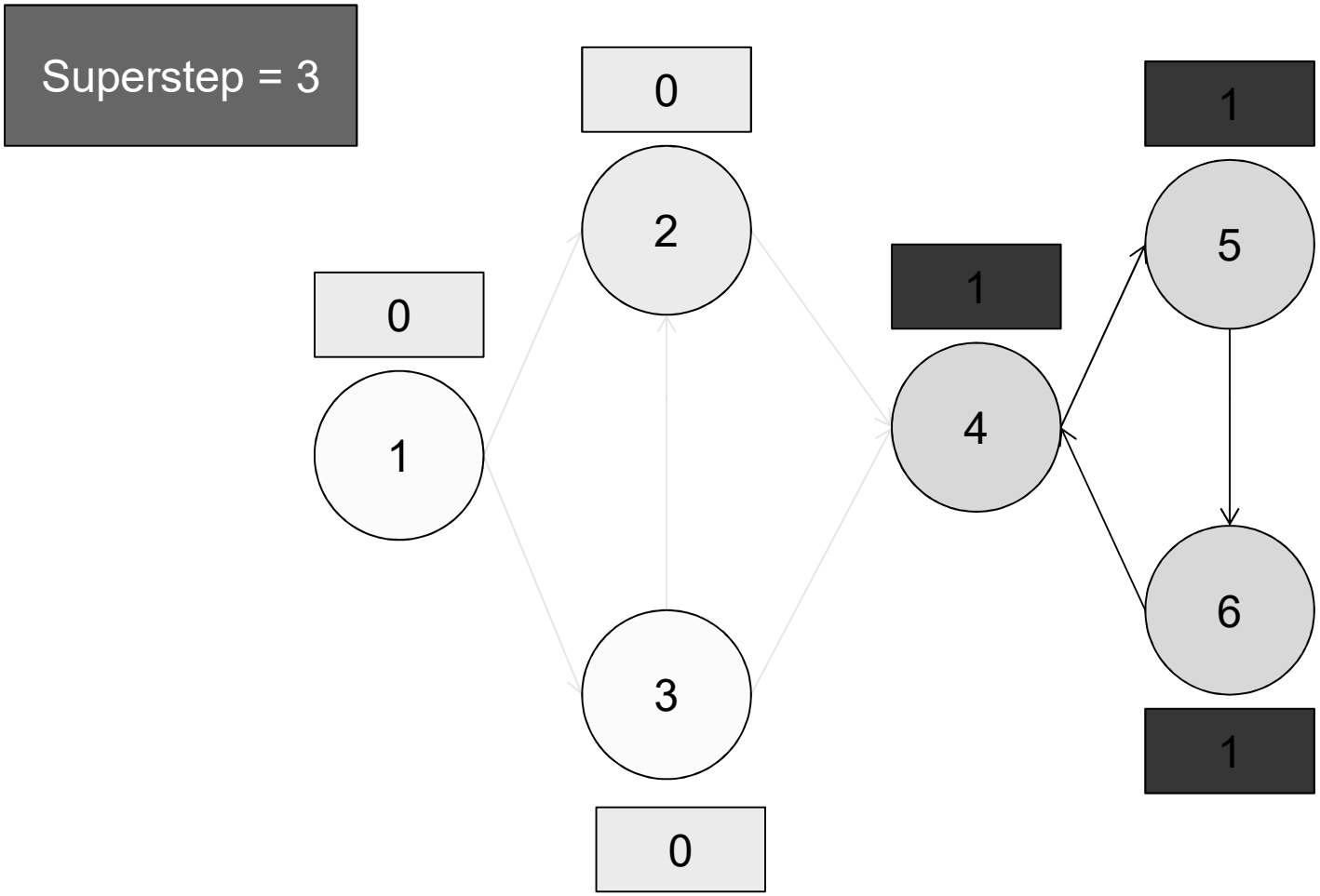
Is DAG Pregel: **sendMsg**



Is DAG Pregel: **vprog**



Is DAG Pregel: **sendMsg**



Final (non-pregel) Step

- Is a DAG iff all vertex incounts are zero
- This requires something like:
  - **filter(_.deg != 0).isEmpty**

Looking at the Code

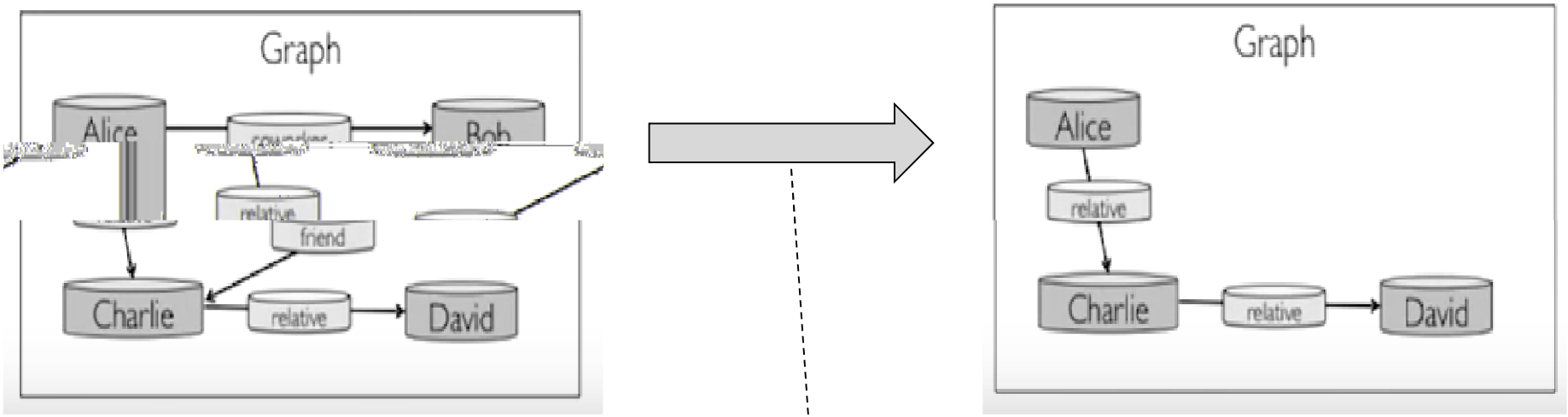
```
def isDag(): Boolean = {
  val dag = g.outerJoinVertices(g.inDegrees){
    case(vid, vt, oInDeg) => oInDeg.getOrElse(0)
  }

  import IsDagUtil._

  val resG = dag.pregel(
      initialMsg = -1,
      maxIterations = Int.MaxValue,
      activeDirection = EdgeDirection.Out
  )(vprog, sendMsg, mergeMsg)

  resG.vertices.filter(_._2 > 0).isEmpty
}
```

GraphX APIs

GraphX APIs

```
def mapVertices[VD2](map: (VertexId, VD) ⇒ VD2): Graph[VD2, ED]
def mapEdges[ED2](map: (Edge[ED]) ⇒ ED2): Graph[VD, ED2]
def mapTriplets[ED2](map: (EdgeTriplet[VD, ED]) ⇒ ED2): Graph[VD, ED2]


def reverse: Graph[VD, ED]
def subgraph(
    epred: (EdgeTriplet[VD, ED]) ⇒ Boolean,
    vpred: (VertexId, VD) ⇒ Boolean): Graph[VD, ED]
def mask[VD2, ED2](other: Graph[VD2, ED2]): Graph[VD, ED] // subgraph based on other
def groupEdges(merge: (ED, ED) ⇒ ED): Graph[VD, ED] // merge parallel edges
```

Example: subgraph



graph.subgraph(vpred = (_, name) => name != "Bob")

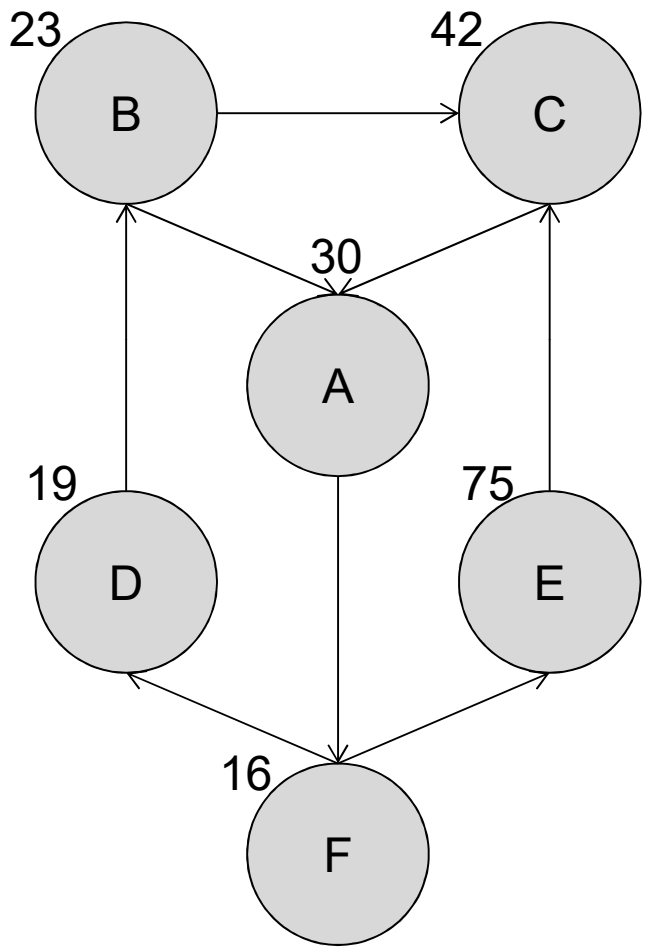
GraphX APIs

```
def outerJoinVertices[U, VD2](
    other: RDD[(VertexId, U)])
    (mapFunc: (VertexId, VD, Option[U]) ⇒ VD2): Graph[VD2, ED]
def aggregateMessages[A](
    sendMsg: (EdgeContext[VD, ED, A]) ⇒ Unit,
    mergeMsg: (A, A) ⇒ A): VertexRDD[A]
```

Age Of Oldest Follower

```
val oldestFollowersAge: VertexRDD[Int] =
  graph.aggregateMessages(
        ctx => ctx.sendToDst(ctx.srcAttr),
        (c1, c2) => Math.max(c1, c2)
  )
```

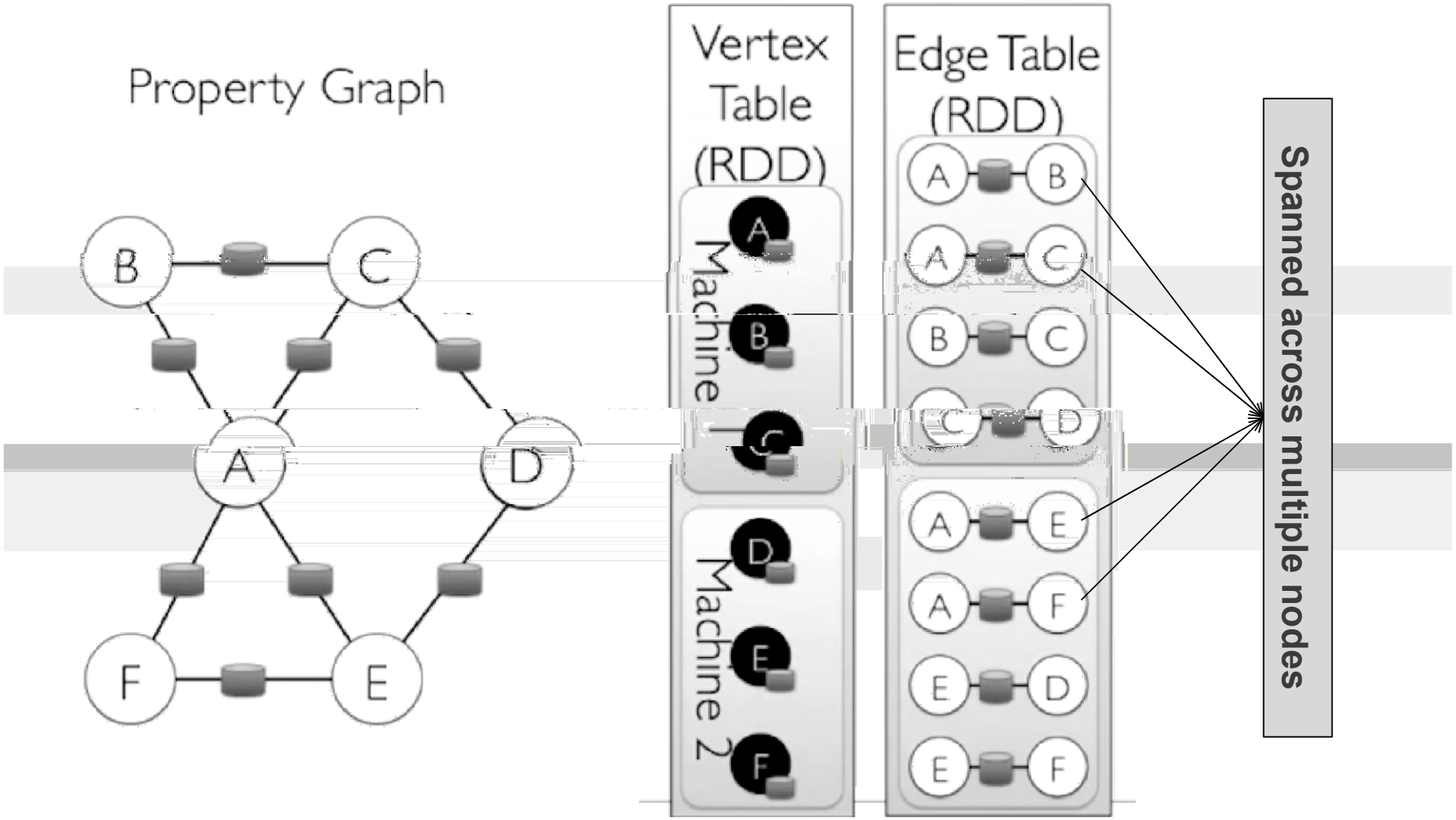
```
class EdgeContext[VD, ED, A] {
        def attr: ED

        def srcAttr: VD
        def dstAttr: VD

        def srcId: VertexId
        def dstId: VertexId

        def sendToDst(msg: A): Unit
        def sendToSrc(msg: A): Unit
}
```



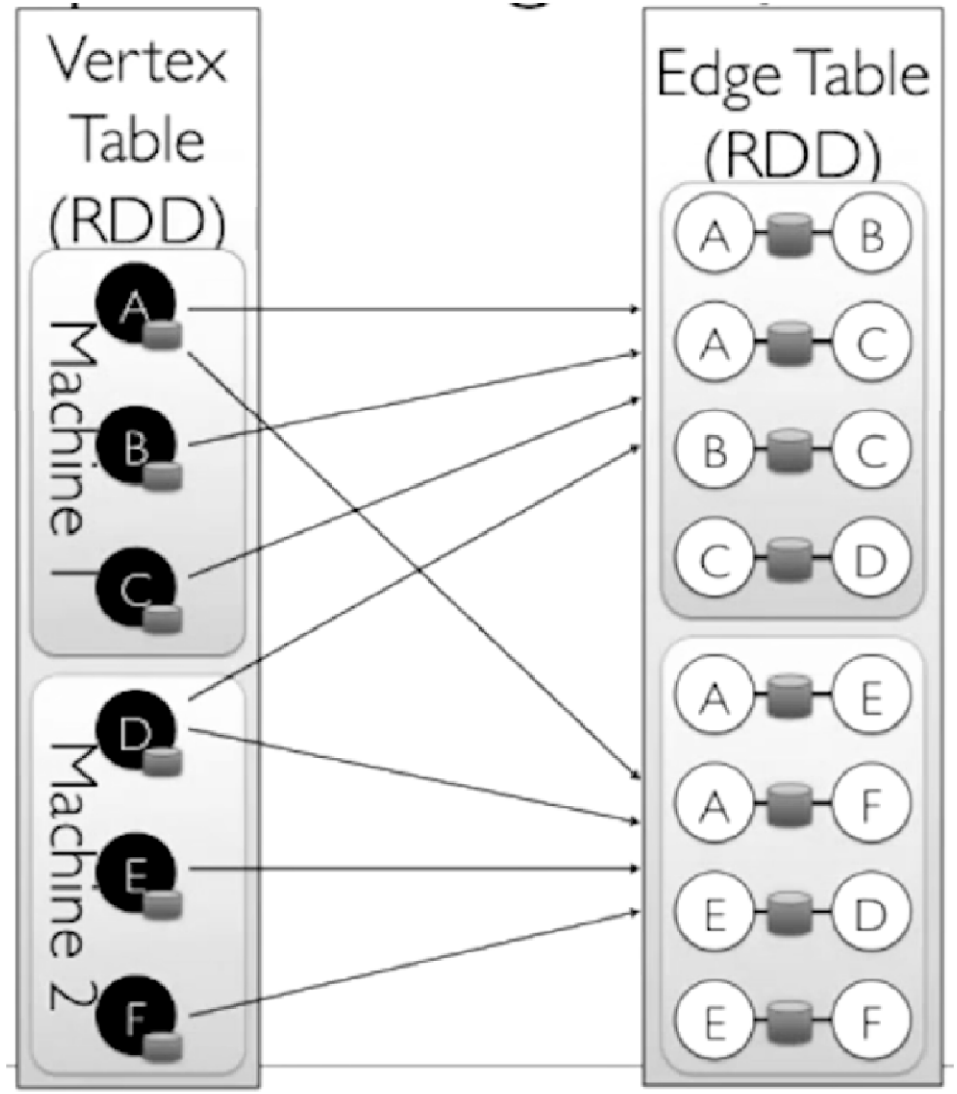
Agenda

- An Intro to Graph Processing
- Why GraphX?
- How (to use) GraphX?
- **How GraphX Works**

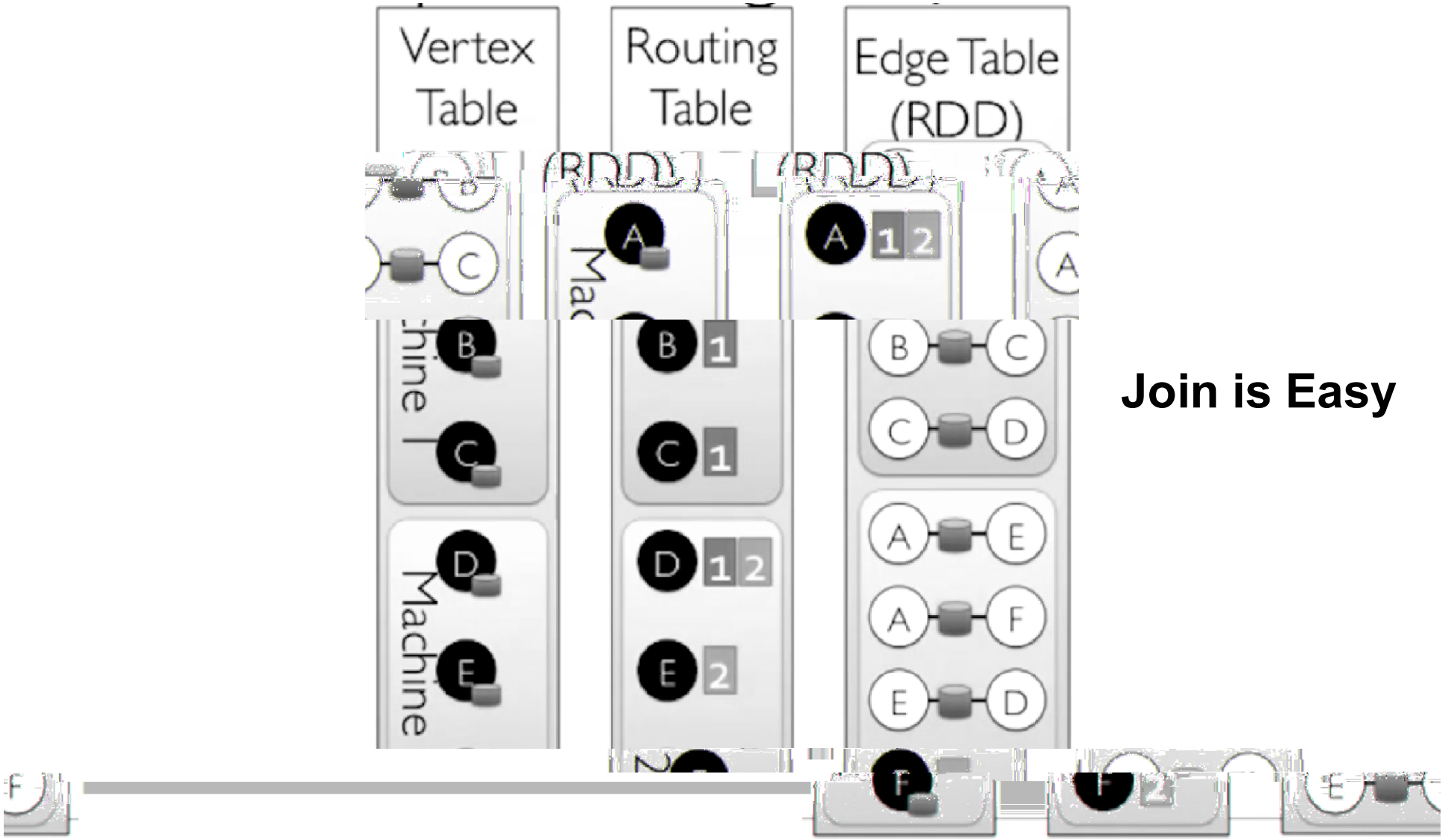
A Graph as-a Table



Triplets Under the Hood

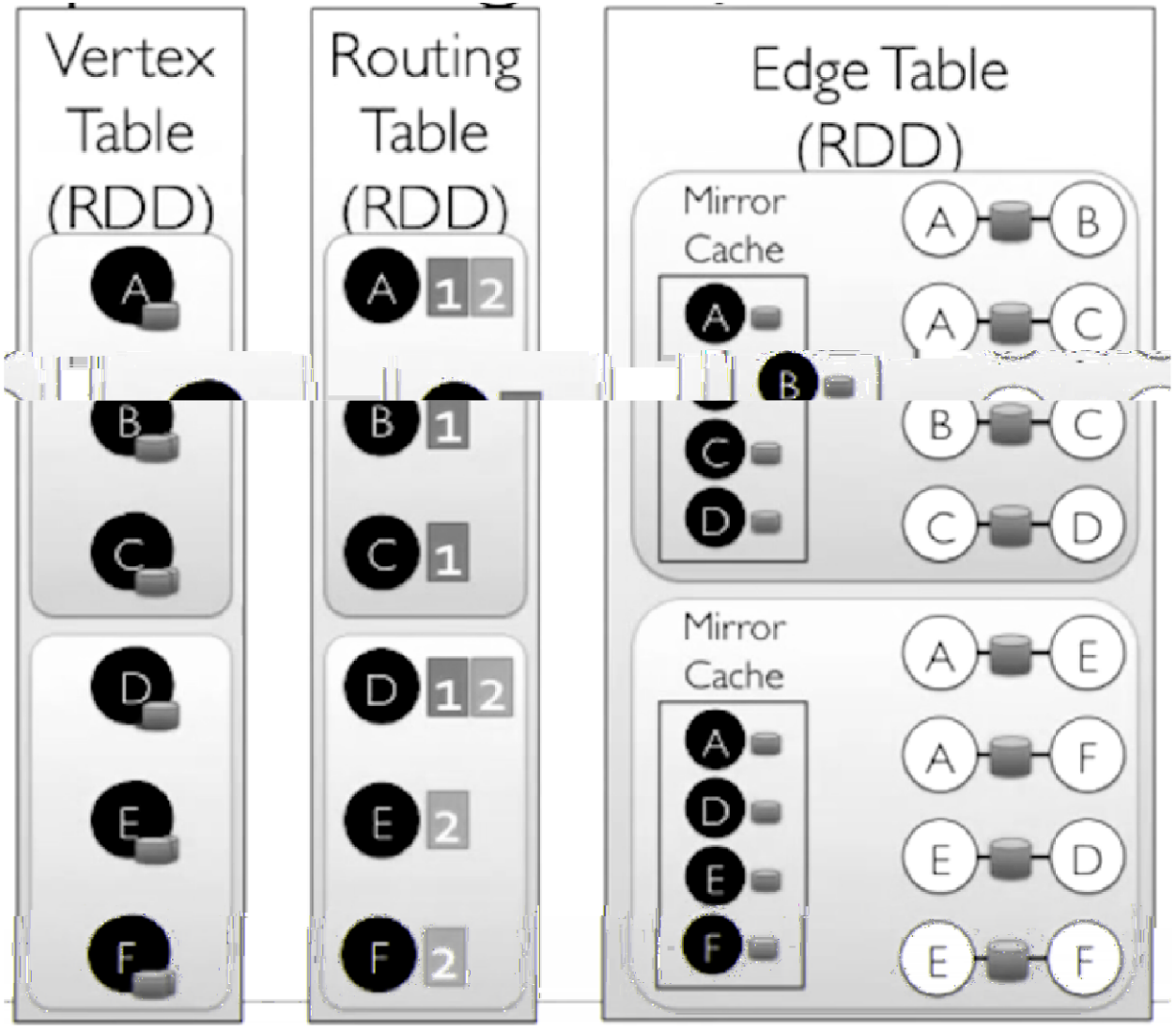


Triplets Under the Hood



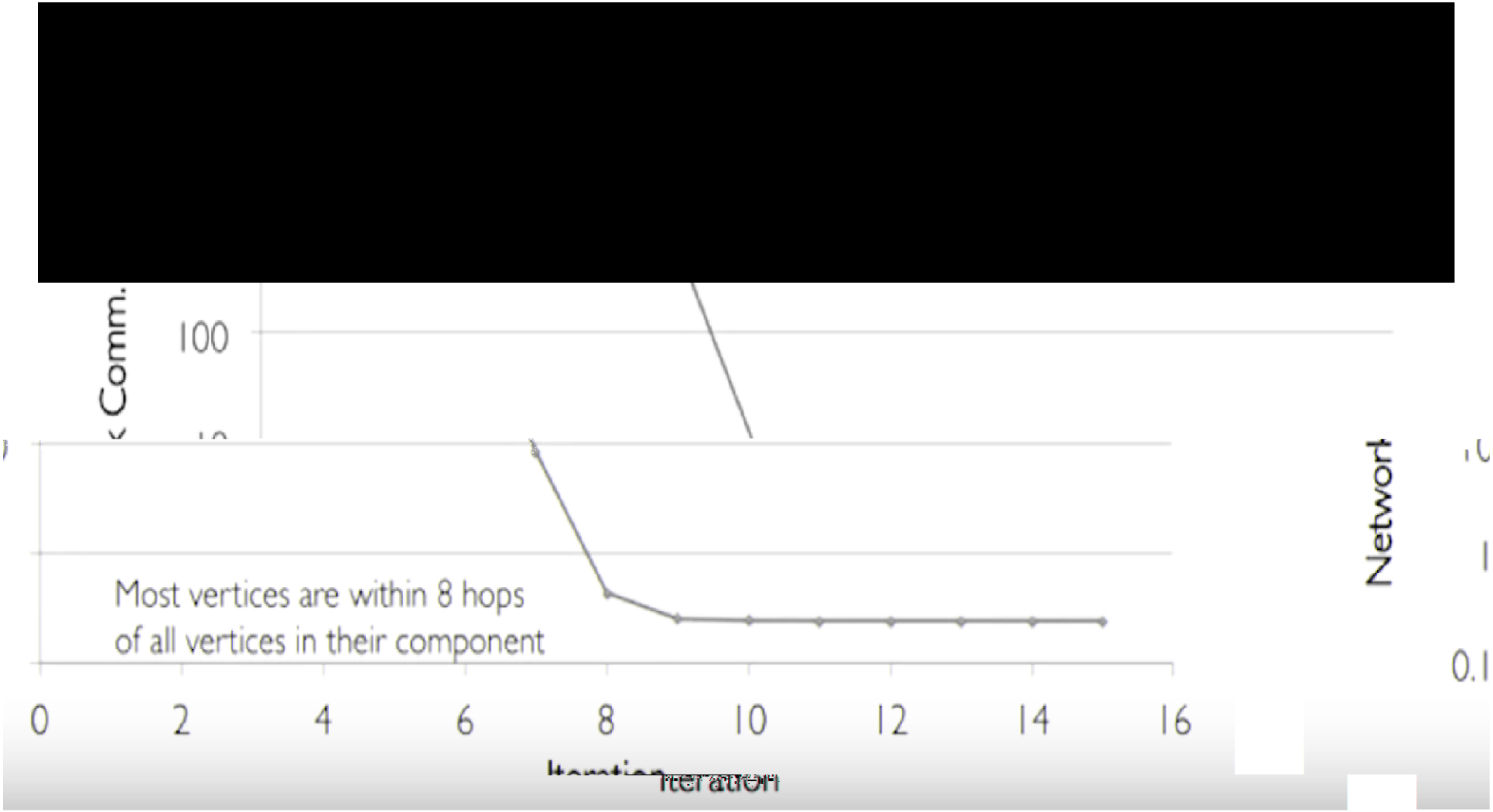
Join is Easy

Triplets Under the Hood



Send only vertex properties that changed

Reduction in Communication Due to Caching



Graph Partitioning

Graph Partitioning

- **Goal:** create a balanced graph partition

- **Challenges:**
  - Problem is NPH
  - Heuristics exist but they are complex and there is a chicken & egg problem here

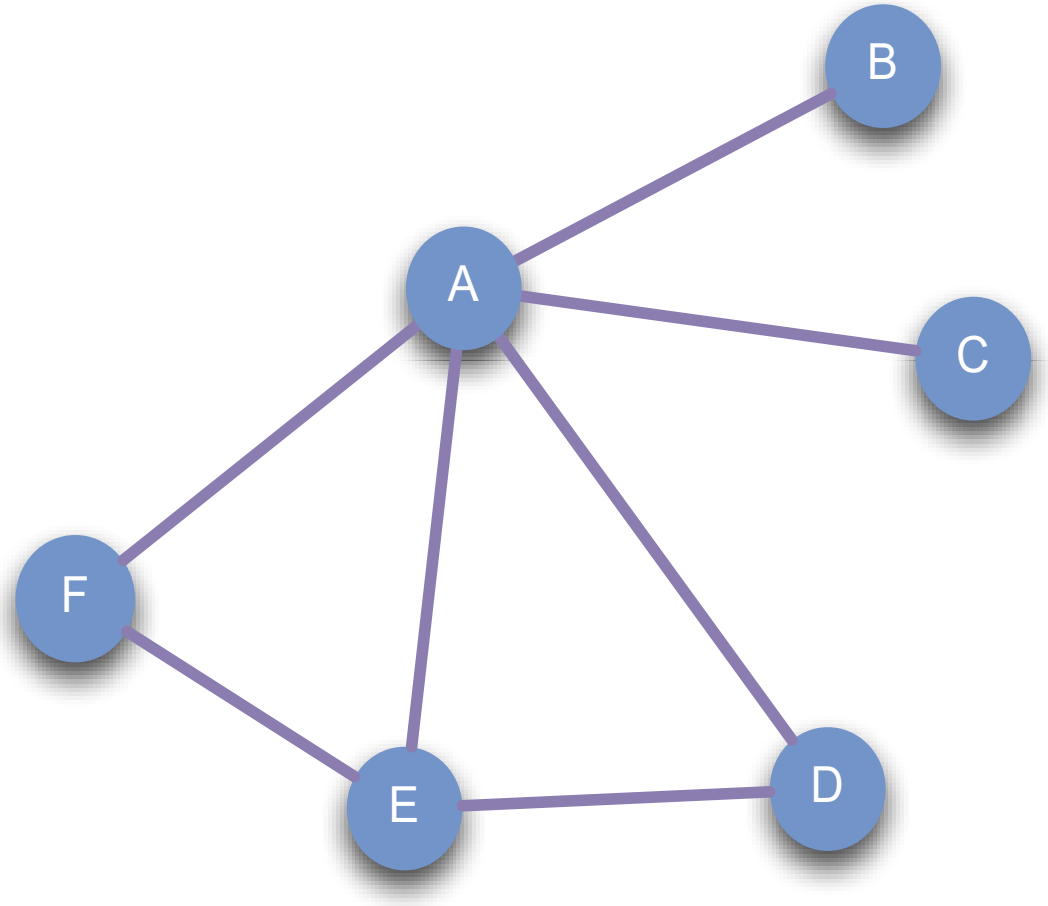
GraphX Partitioning

- Controls edge partitioning
- Four built in partitioning strategies
- Can also write custom partitioning

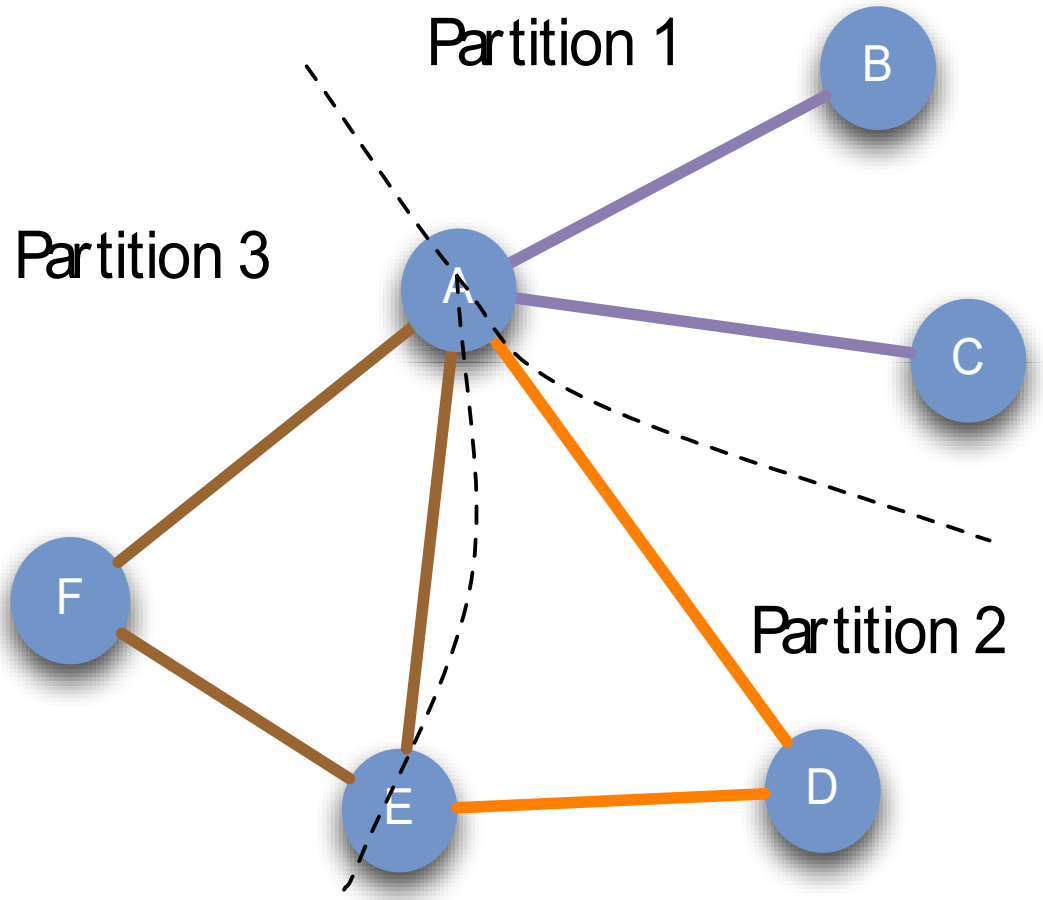
- Partitioning can be based on vertex/edge

g.partitionBy(PartitionStrategy.**EdgePartition2D, numPartitions)**

Vertex Cut Partitioning



Vertex Cut Partitioning



Support Partitioning Strategies

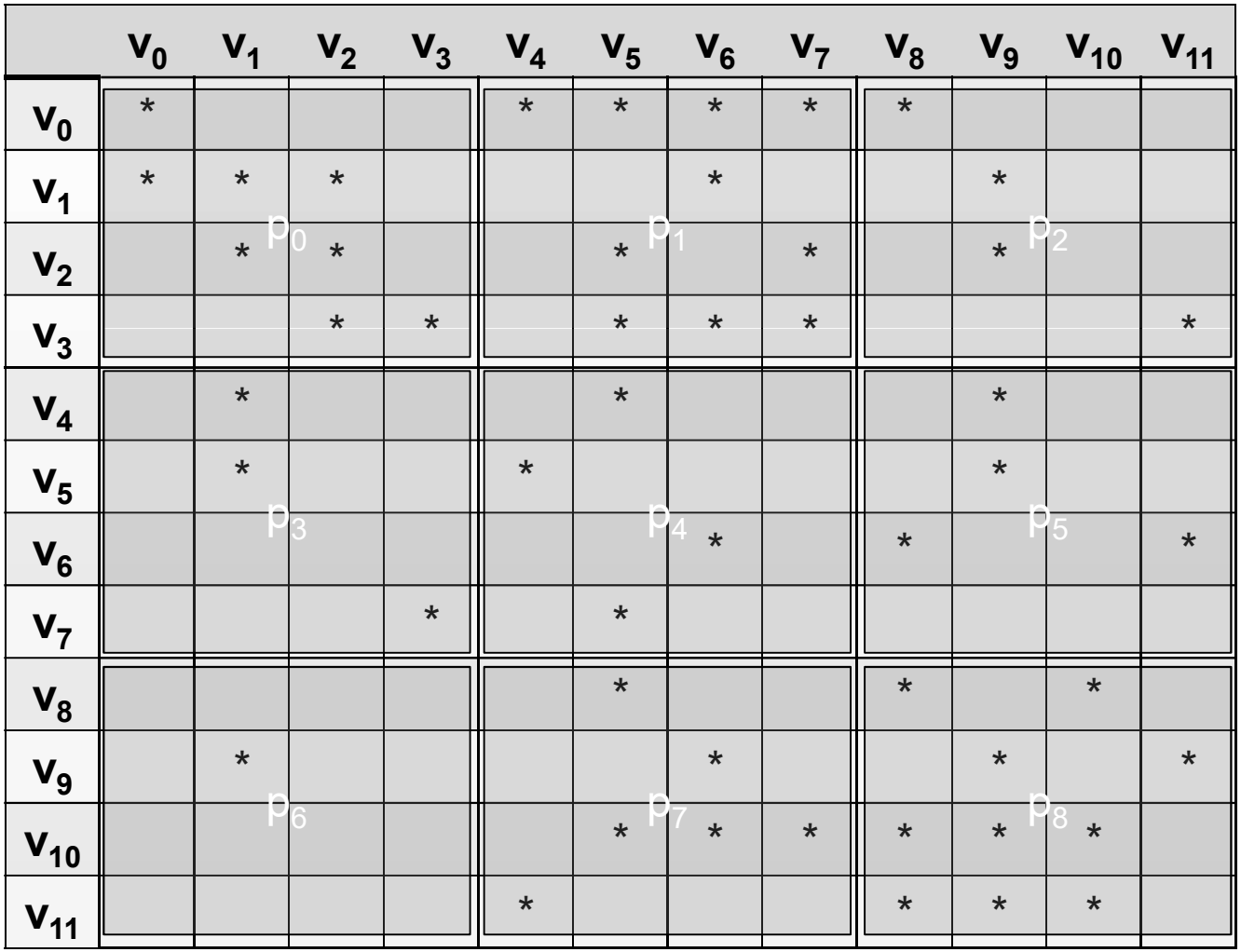
```
trait PartitionStrategy {
  def getPartition(src: VertexId, dst: VertexId, numParts: PartitionID): PartitionID
}
```

- **RandomVertexCut** – select edge partition based on **(src, dst)** hash
- **CanonicalRandomVertexCut –** same as above only canonical
- **EdgePartition1D** – partitioning based only on source vertex id (bad for hypernodes)

Partition As Sparse Matrix EdgePartition2D

	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}
v_0	*				*	*	*	*	*			
v_1	*	*	*				*			*		
v_2		*	*			*		*		*		
v_3			*	*		*	*	*				*
v_4		*				*				*		
v_5		*			*					*		
v_6							*		*			*
v_7				*		*						
v_8						*			*		*	
v_9		*					*			*		*
v_{10}						*	*	*	*	*	*	
v_{11}					*				*	*	*	

Partition As Sparse Matrix EdgePartition2D



Summary

1. Graph-parallel processing on Spark
2. Unify tables/graphs processing
3. Work being done to improve performance brining it closer to specialized engines such as GraphLab

References

- http://ampcamp.berkeley.edu/amp-camp-one-berkeley-2012
- http://www.slideshare.net/payberah/graph-processing-powergraph-and-graphx
- http://www.slideshare.net/payberah/pregel-43904273
- http://ampcamp.berkeley.edu/big-data-mini-course/graph-analytics-with-graphx.html
- http://spark.apache.org/docs/latest/graphx-programming-guide.html
- http://www.slideshare.net/SigmoidHR/graph-x-pregel
- https://www.youtube.com/watch?v=Y7hq5MudV9M
- http://note.yuhc.me/2015/03/graphx-partition-strategy/