GraphX

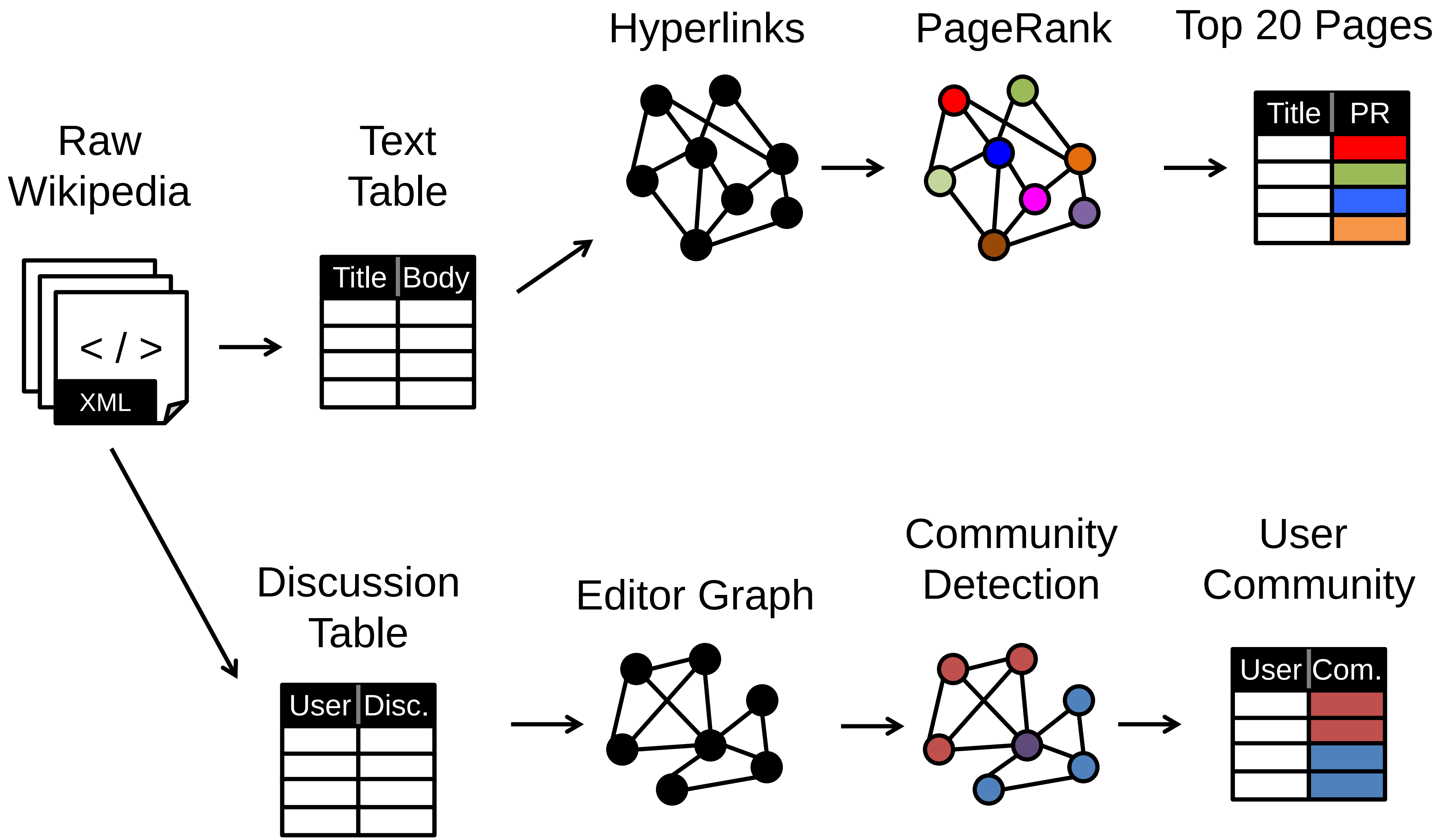
Unifying Data-Parallel and Graph-Parallel Analytics

Cloud Computing Research Topic

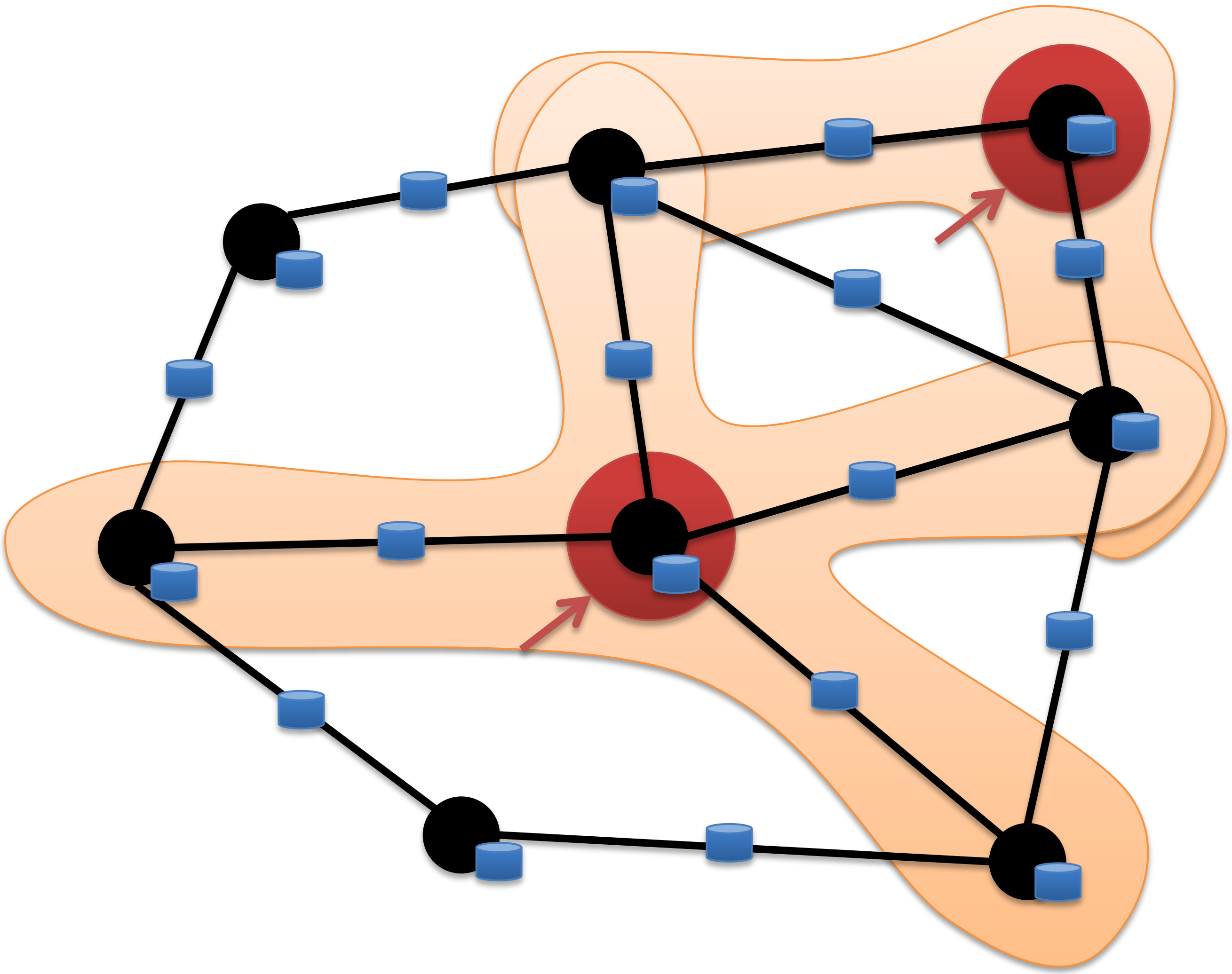
Fatemeh Shafiee
Kashif Rabbani

May 2018

Graphs are Central to Analytics



The Graph-Parallel Pattern



Computation depends only on the **neighbors**

Graph-Parallel Algorithms

- **Classification :** Neural Networks
- **Community Detection :** Triangle-Counting
- **Semi-supervised ML :** Graph SSL
- **Collaborative Filtering :** Alternating Least Squares
- **Graph Analytics :** PageRank, Shortest Path, Graph Coloring
- **Structured Prediction :**Max-Product, Linear Programs

Current Graph-Parallel Systems

Pregel

APACHE GIRAPH

GraphLab

Separate Systems to Support Each View

Table View

hadoop

Spark

Graph View

Pregel

GraphLab

APACHE GIRAPH

What are the Problems?

- Having separate systems for each view is **difficult to use** and **inefficient**

- Difficult to **Program** and Use

- Users must Learn, **Deploy**, and **Manage** multiple systems

- **Extensive** data movement and **duplication** across the network and file system

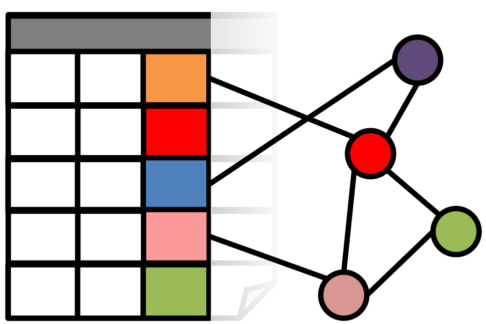
Solution: GraphX

New API

Blurs the distinction between Tables and Graphs

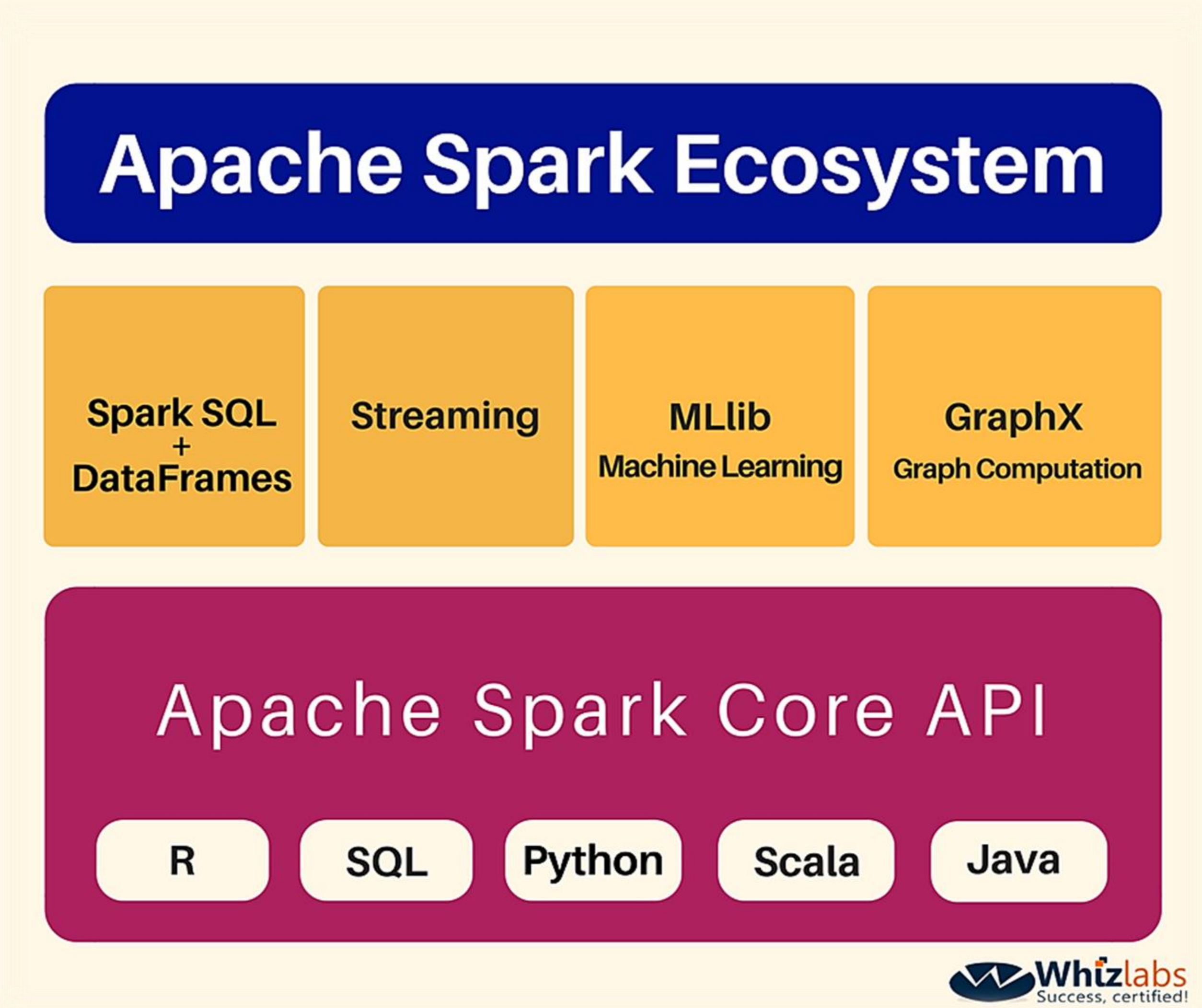
New System

Combines Data-Parallel Graph-Parallel Systems

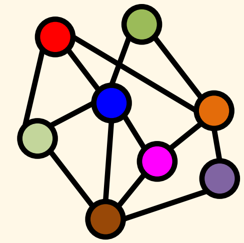


Enabling users to easily and efficiently express the entire graph analytics pipeline

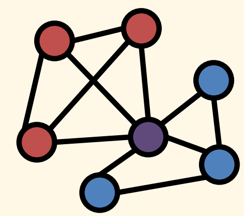
What is GraphX?



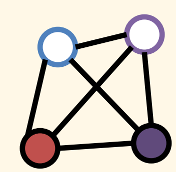
PageRank



Community Detection

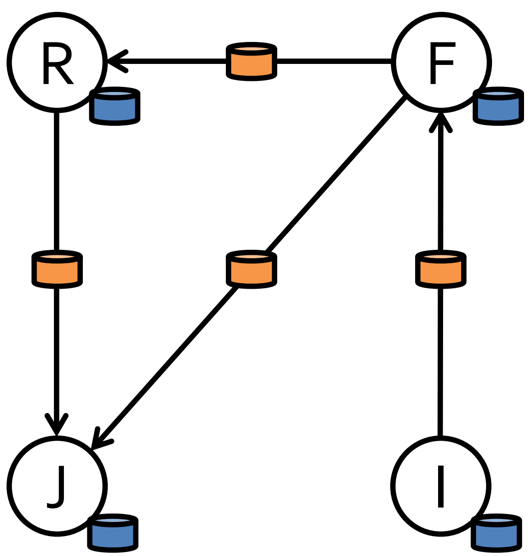


Triangle Count



Property Graph

Property Graph



Vertex Property Table

Id	Property (V)
Rxin	(Stu., Berk.)
Jegonzal	(PstDoc, Berk.)
Franklin	(Prof., Berk)
Istoica	(Prof., Berk)

Edge Property Table

SrcId	DstId	Property (E)
rxin	jegonzal	Friend
franklin	rxin	Advisor
istoica	franklin	Coworker
franklin	jegonzal	PI

RDD Operators

RDD operators are inherited from Spark:

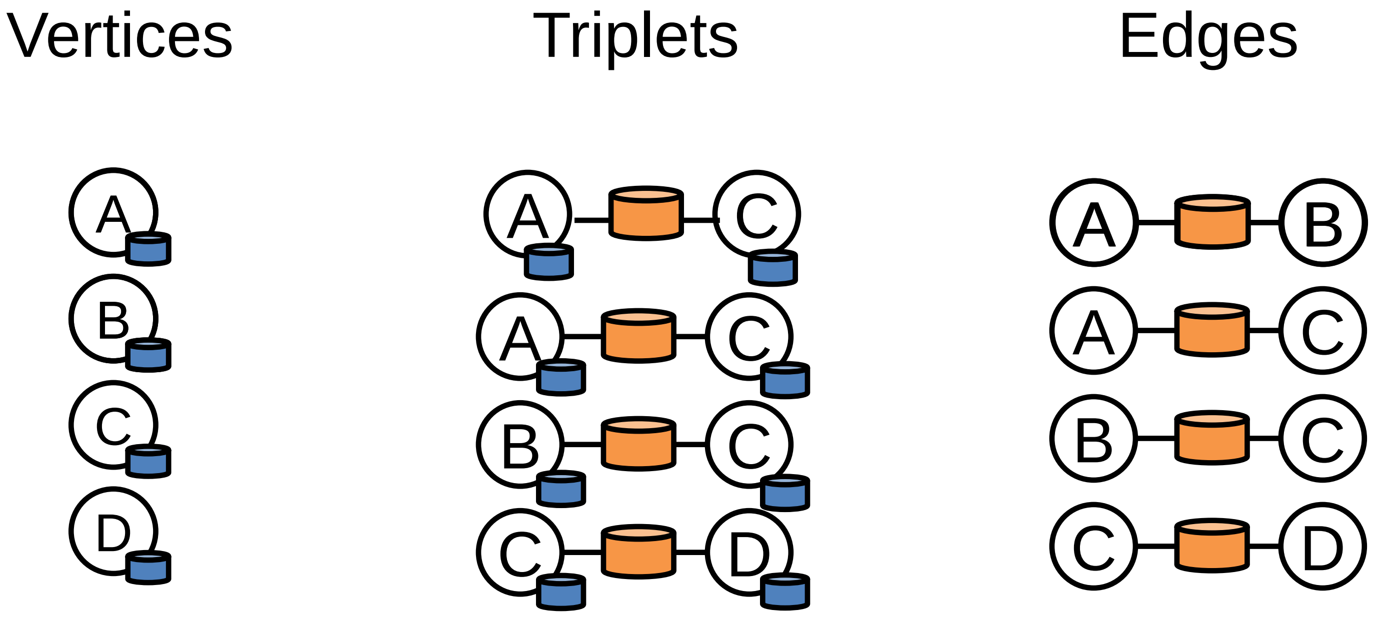
map	reduce	sample
filter	count	take
groupBy	fold	first
sort	reduceByKey	partitionBy
union	groupByKey	mapWith
join	cogroup	pipe
leftOuterJoin	cross	save
rightOuterJoin	zip	**...**

Graph Operators

```
class Graph [ V, E ] {
   def Graph(vertices: Table[ (Id, V) ],
             edges: Table[ (Id, Id, E) ])
   // Table Views -----------------
   def vertices: Table[ (Id, V) ]
   def edges: Table[ (Id, Id, E) ]
   def triplets: Table [ ((Id, V), (Id, V), E) ]
   // Transformations ------------------------------
   def reverse: Graph[V, E]
   def subgraph(pV: (Id, V) => Boolean,
                pE: Edge[V,E] => Boolean): Graph[V,E]
   def mapV(m: (Id, V) => T ): Graph[T,E]
   def mapE(m: Edge[V,E] => T ): Graph[V,T]
   // Joins ----------------------------------------
   def joinV(tbl: Table [(Id, T)]): Graph[(V, T), E ]
   def joinE(tbl: Table [(Id, Id, T)]): Graph[V, (E, T)]
   // Computation ----------------------------------
   def mrTriplets(mapF: (Edge[V,E]) => List[(Id, T)],
                  reduceF: (T, T) => T): Graph[T, E]
}
```

Triplets Join Vertices and Edges

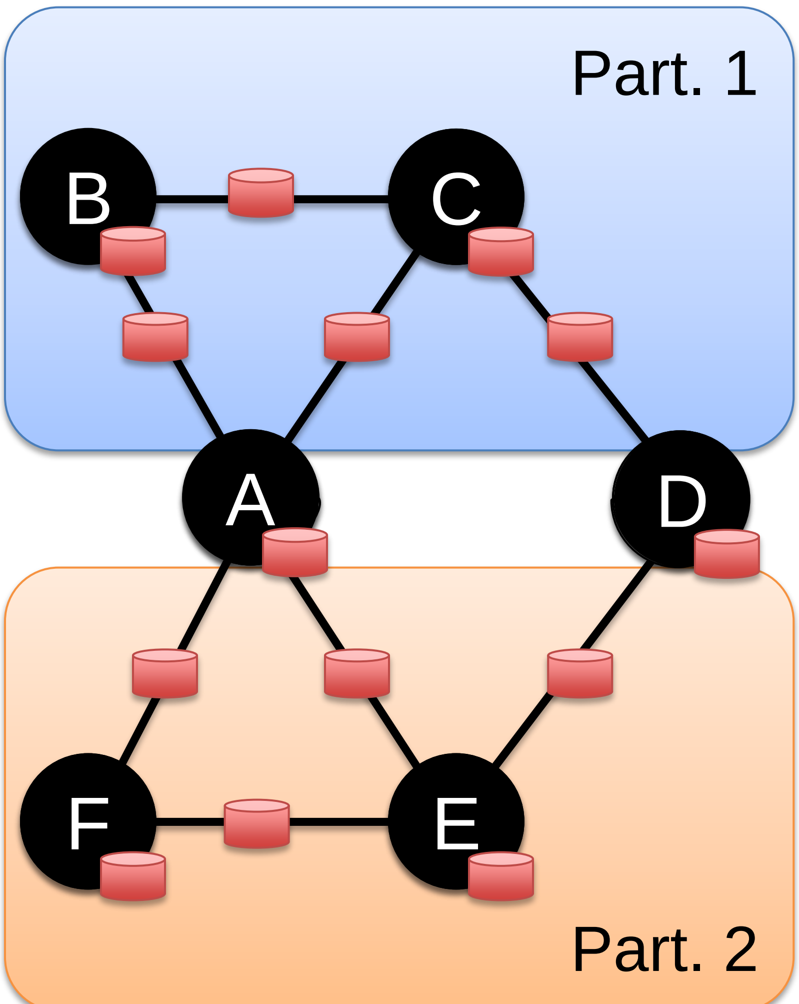
The ***triplets*** operator joins vertices and edges

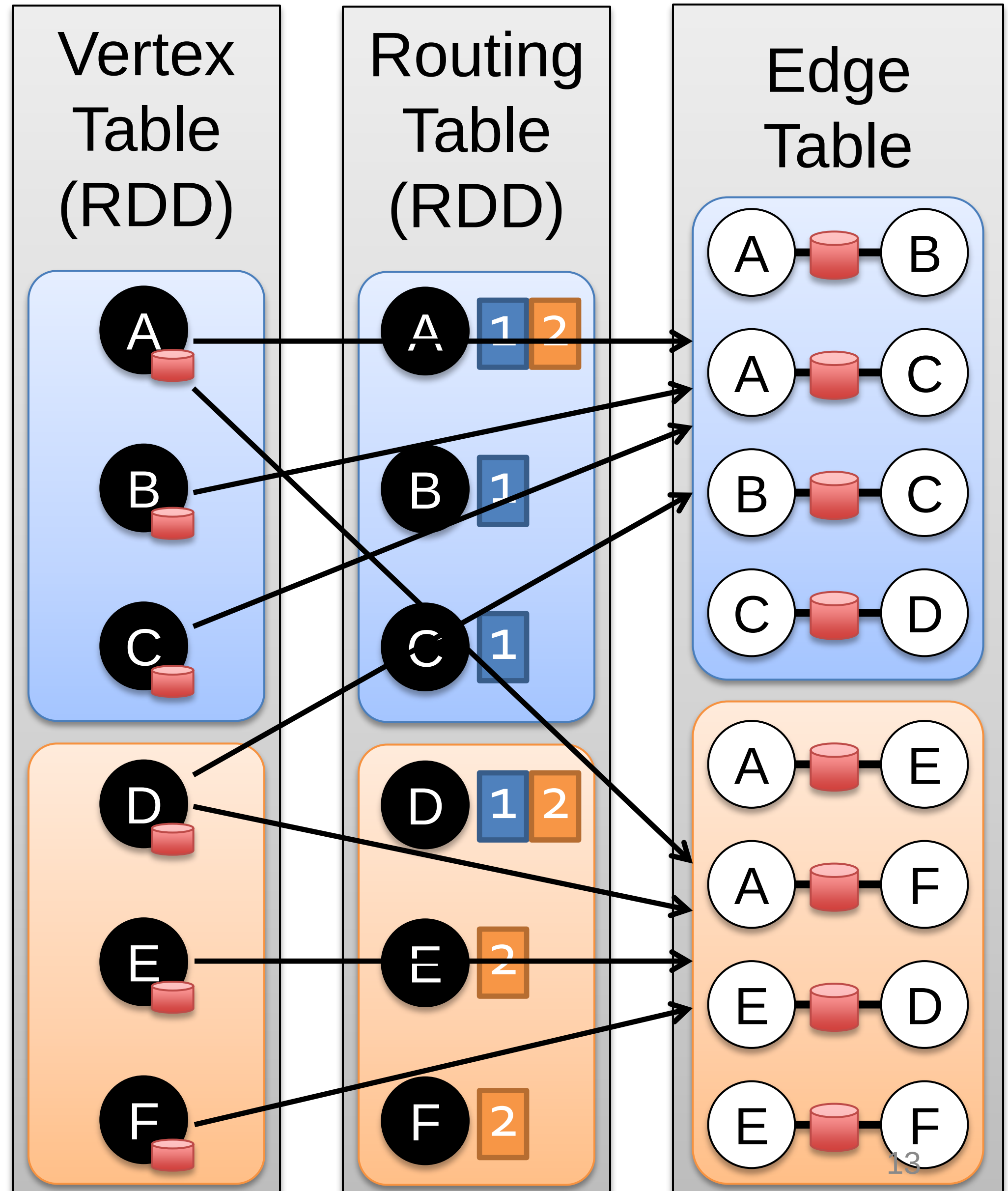


GraphX System Design

Distributed Graphs as Tables (RDDs)

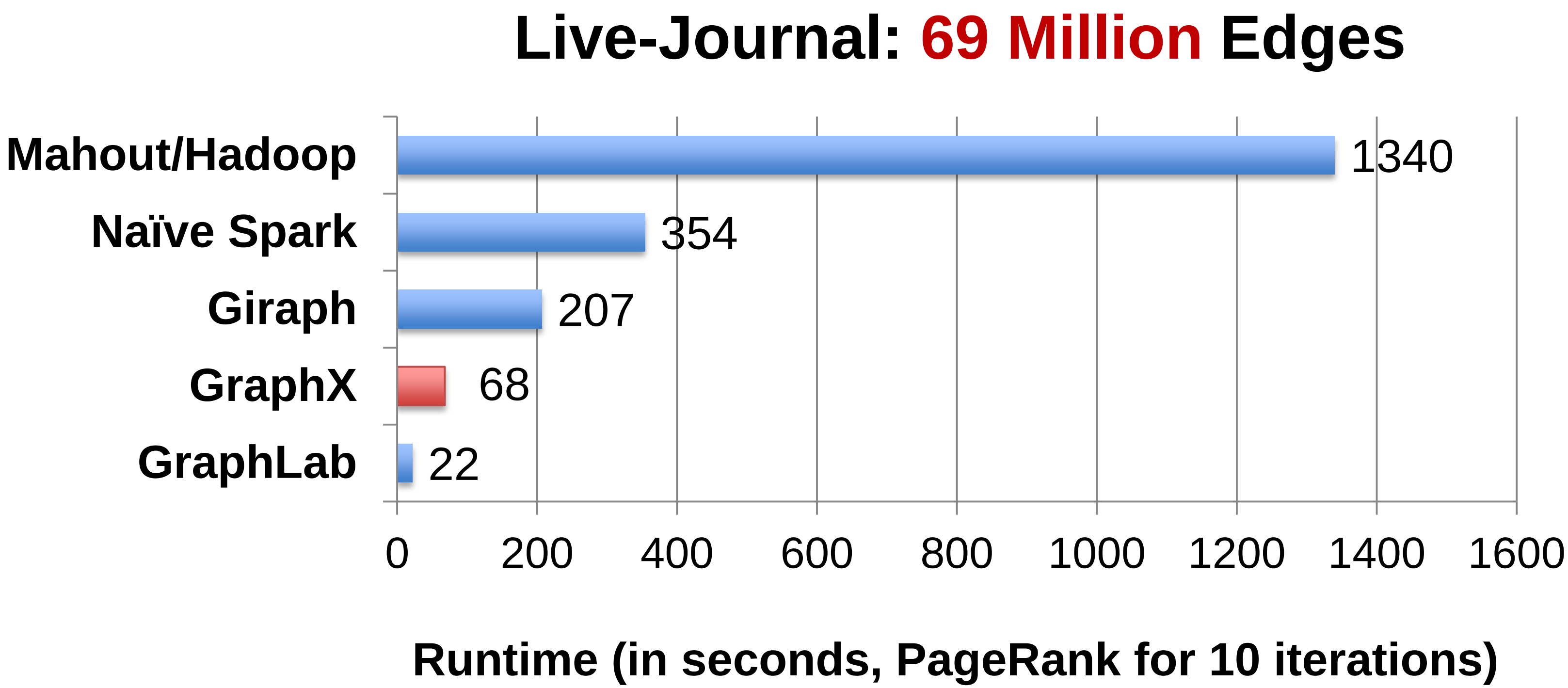
Property Graph





Ref: https://amplab.cs.berkeley.edu/

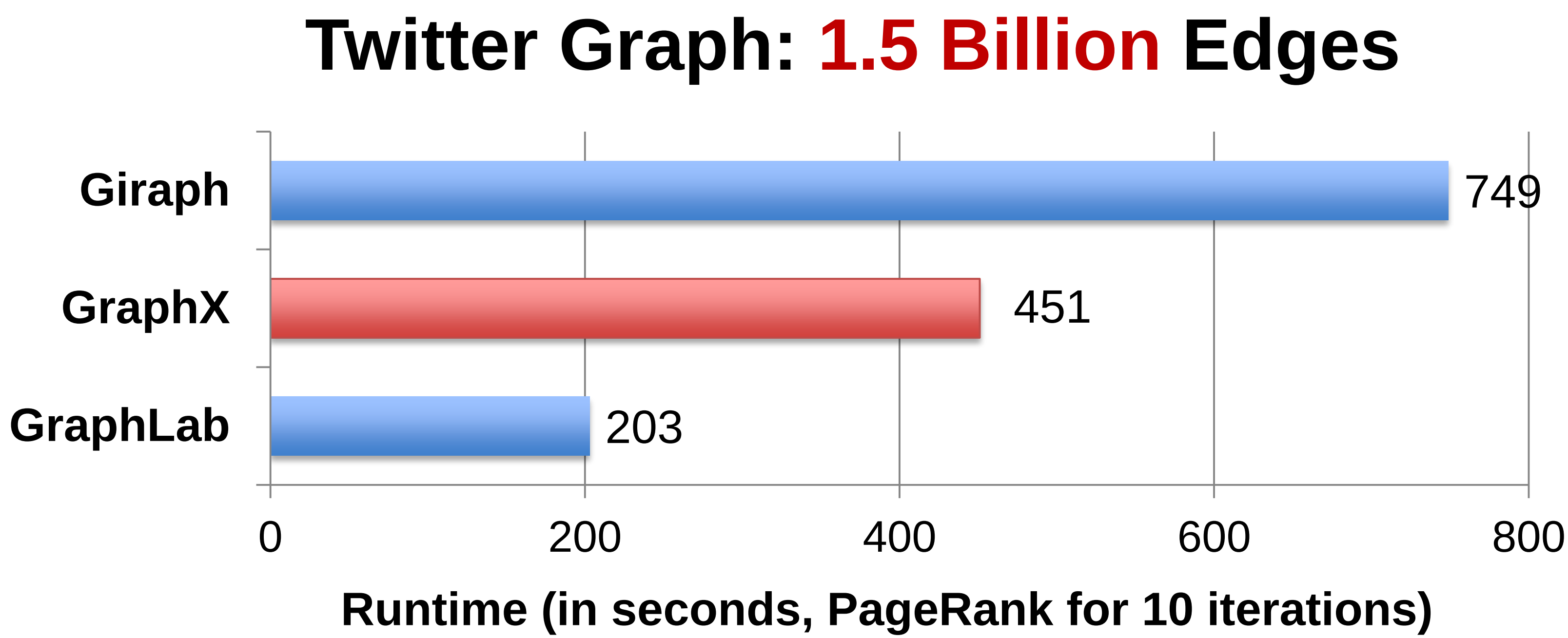
Performance Comparisons



GraphX is roughly 3x slower than GraphLab

Ref: https://amplab.cs.berkeley.edu/

GraphX scales to larger graphs



GraphX is roughly **2x slower** than GraphLab

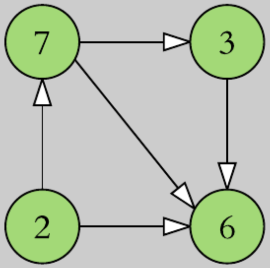
Scala + Java overhead: Lambdas, GC time, etc

No shared memory parallelism

Ref: https://amplab.cs.berkeley.edu/

Finding Min Value in a Graph

Initial Graph:

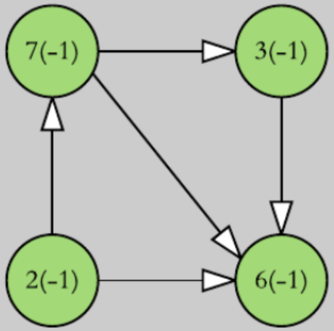


define **f** to be:

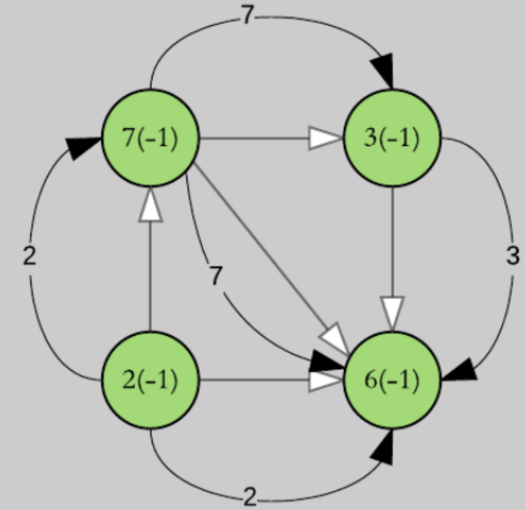
```
val originalValue = value

val value = (messages :+ value).min

if (originalValue == value)
  inactive()
else
  neighbours.foreach(sendMessage)
```

Superstep 0:

Initialise the graph with `-1` picked as the `originalValue`



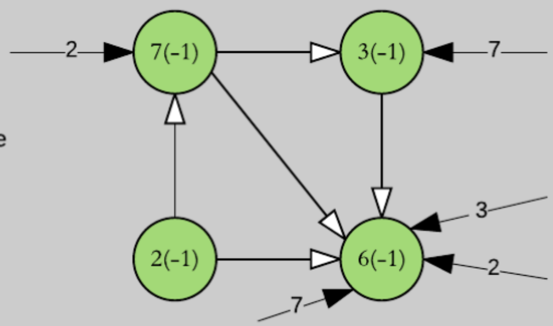
All active vertices send its value as message to its neighbouring vertices



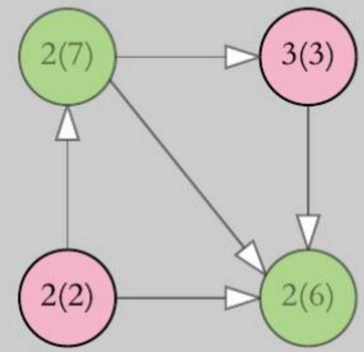
Finding Min Value in a Graph

Superstep 1:

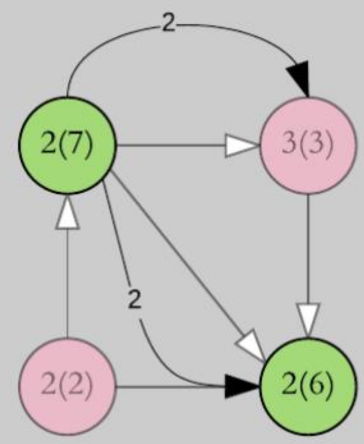
Receive messages from the previous superstep



Mutate the value of the vertices and make vertices with `originalValue == value` inactive



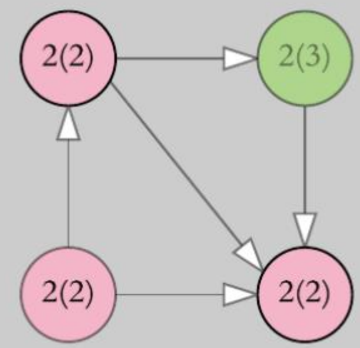
For vertices that are still active, send its value as message to its neighbouring vertices



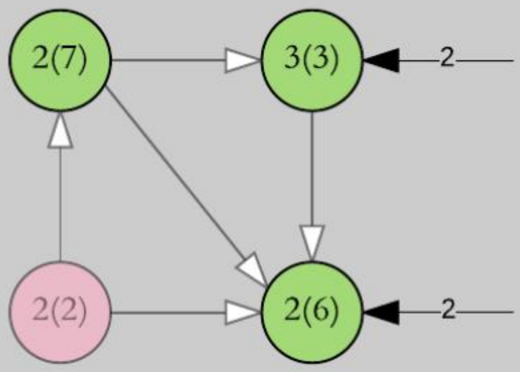
Finding Min Value in a Graph

Superstep 2:

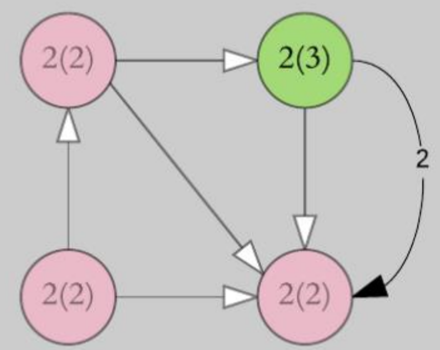
Mutate the value of the vertices and make vertices with *`originalValue == value`* inactive



Receive message from the previous superstep



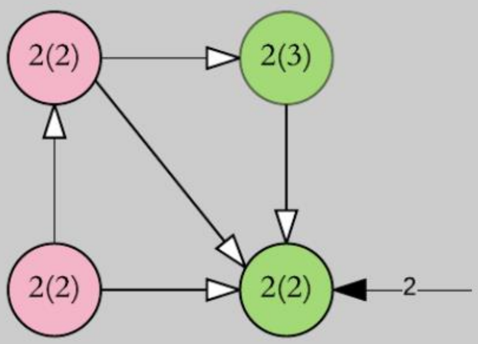
For vertices that are still active, send its value as message to its neighbouring vertices



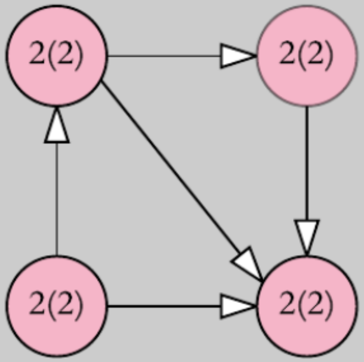
Finding Min Value in a Graph

Superstep 3:

Receive message from the previous superstep



Mutate the value of the vertices and make vertices with `originalValue == value` inactive



Interesting Use Cases

- Building a Graph of **US Businesses**
- Using Spark GraphX to **detect communities at Alibaba**
- Spark, GraphX, and Blockchains: Building a **Behavioral Analytics Platform for Fraud, and Finance**
- **Multi-Label Graph Analysis and Computations** Using GraphX

GraphX

Thank you