GraphX: *Unifying Data-Parallel and Graph-Parallel Analytics*

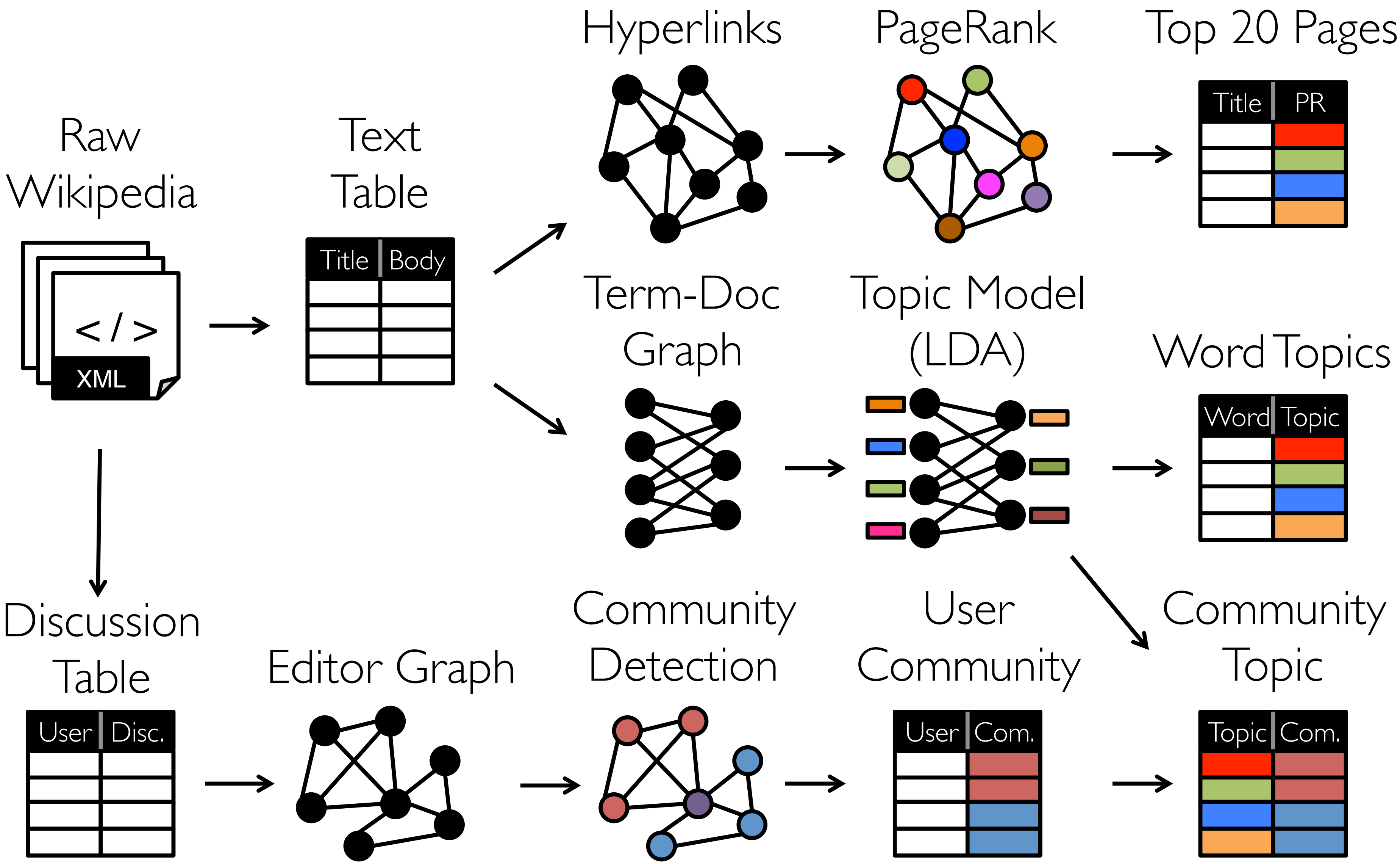
Presented by Joseph Gonzalez

Joint work with Reynold Xin, Daniel Crankshaw, Ankur Dave, Michael Franklin, and Ion Stoica

Strata 2014

*These slides are best viewed in PowerPoint with animation.

Graphs are Central to Analytics



PageRank: Identifying Leaders

$$R[i] = 0.15 + \sum_{j \in \mathrm{Nbrs}(i)} w_{ji} R[j]$$

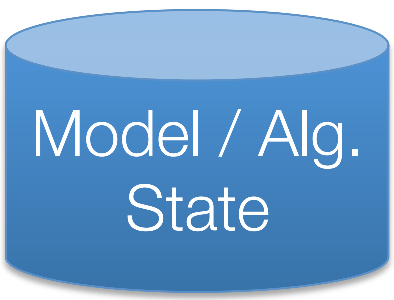
Rank of user *i*

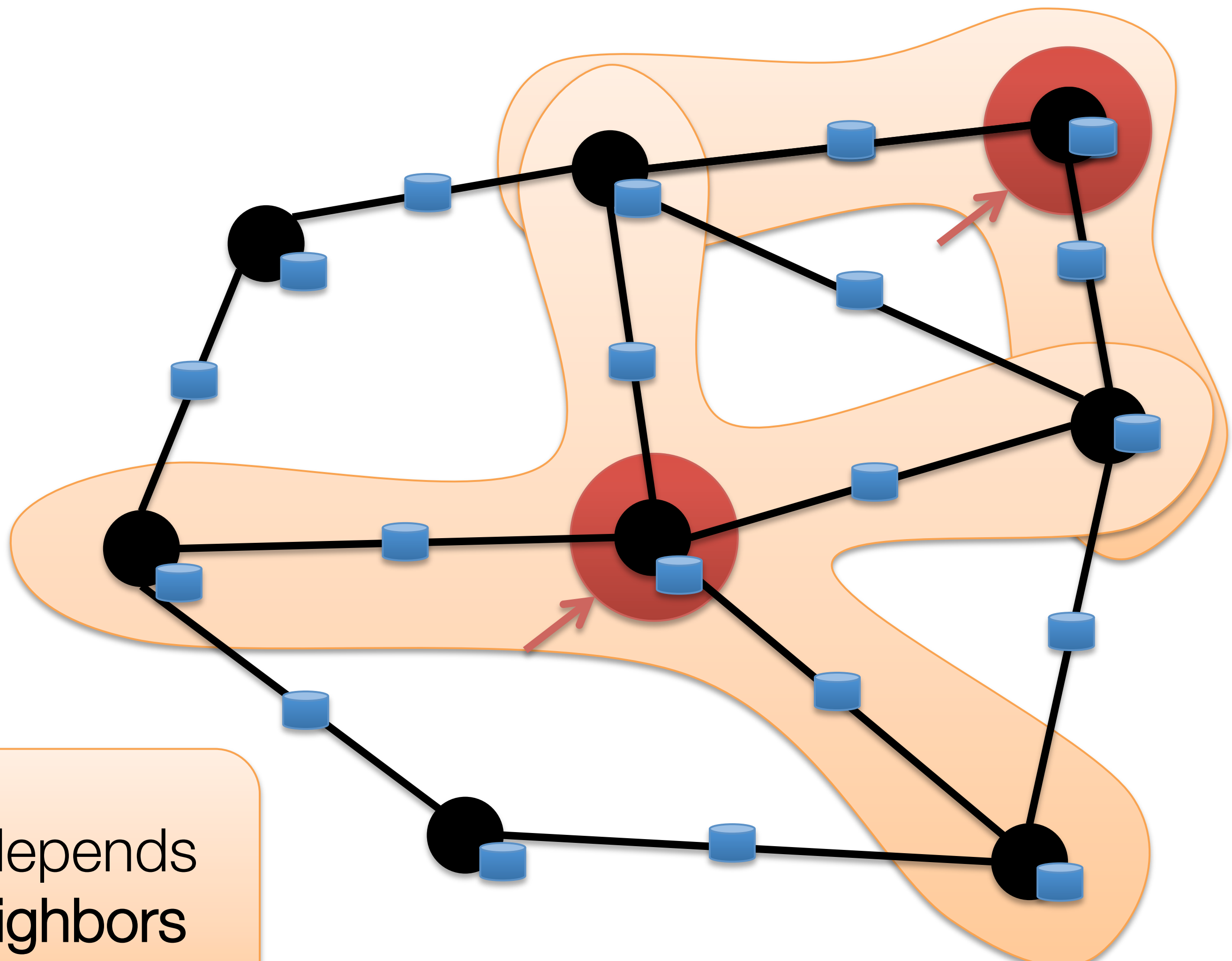
Weighted sum of neighbors' ranks

Update ranks in parallel

Iterate until convergence

The Graph-Parallel Pattern





Computation depends only on the **neighbors**

Many Graph-Parallel Algorithms

- Collaborative Filtering
  - Alternating Least Squares
  - Stochastic Gradient Descent
  - Tensor Factorization
- Structured Prediction
  - Loopy Belief Propagation
  - Max-Product Linear Programs
  - Gibbs Sampling
- Semi-supervised ML
  - Graph SSL
  - CoEM
- Community Detection
  - Triangle-Counting
  - K-core Decomposition
  - K-Truss
- Graph Analytics
  - PageRank
  - Personalized PageRank
  - Shortest Path
  - Graph Coloring
- Classification
  - Neural Networks

Graph-Parallel Systems

Pregel oogle

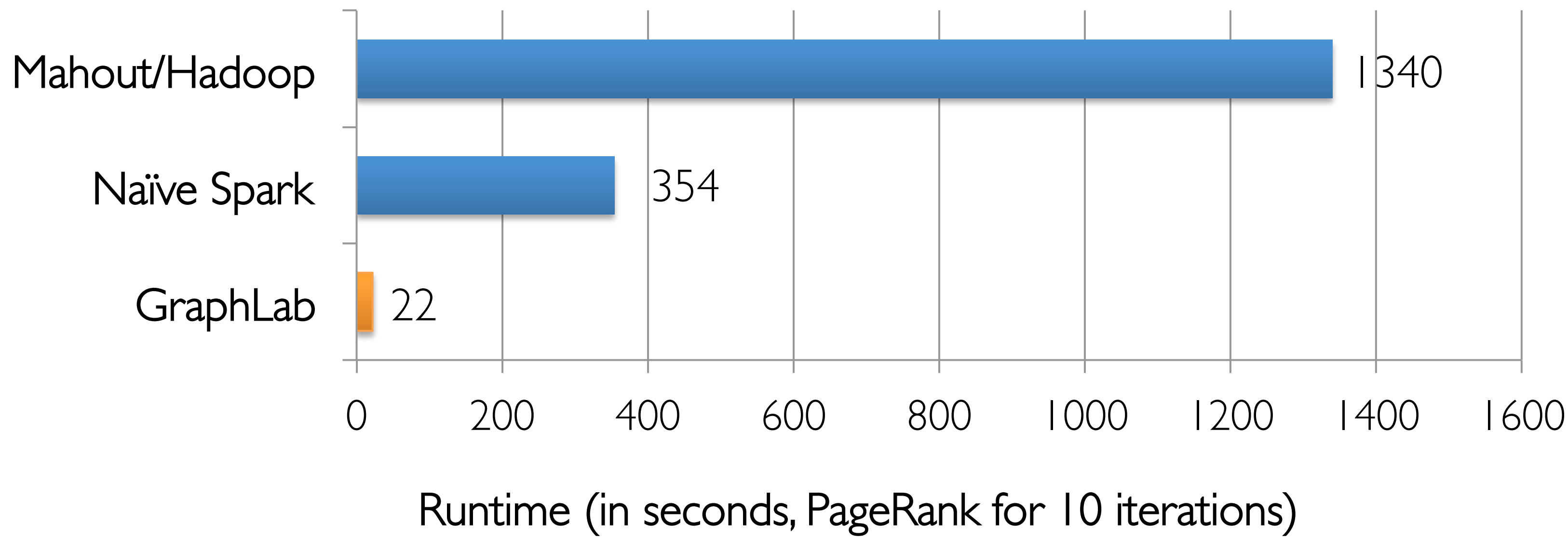
APACHE GIRAPH

GraphLab

Expose specialized APIs to simplify graph programming.

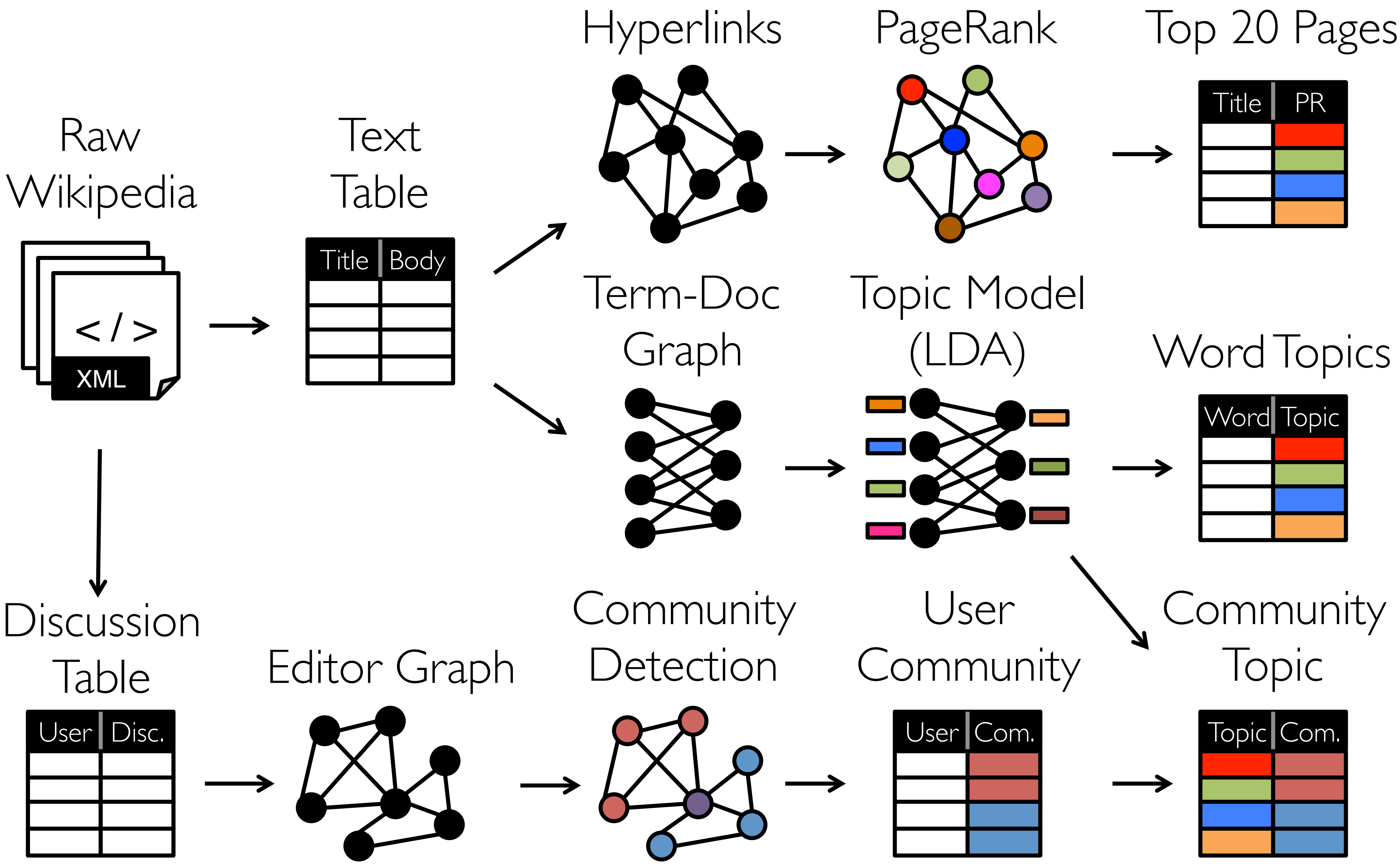
Exploit graph structure to achieve orders-of-magnitude performance gains over more general data-parallel systems.

PageRank on the Live-Journal Graph



GraphLab is *60x faster* than Hadoop
GraphLab is *16x faster* than Spark

Graphs are Central to Analytics

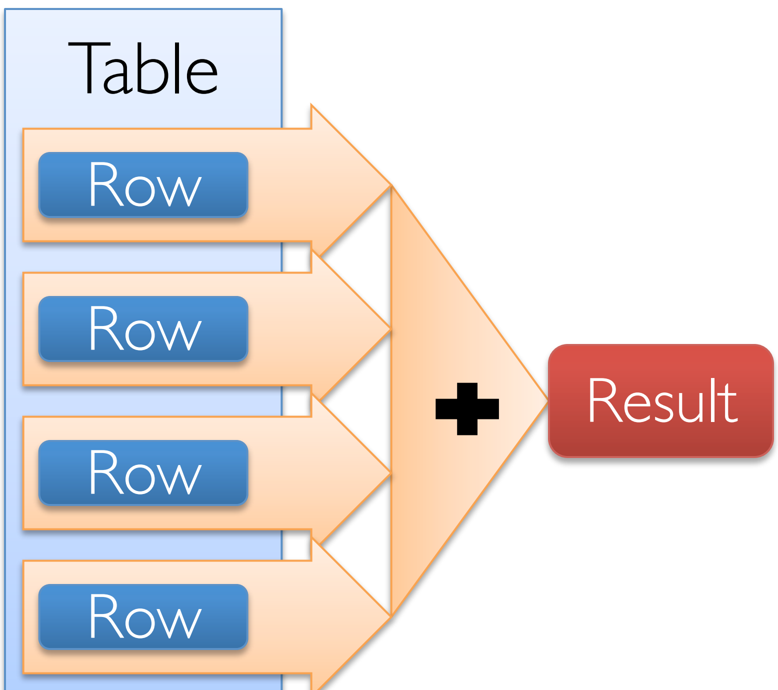


Separate Systems to Support Each View

Table View

hadoop

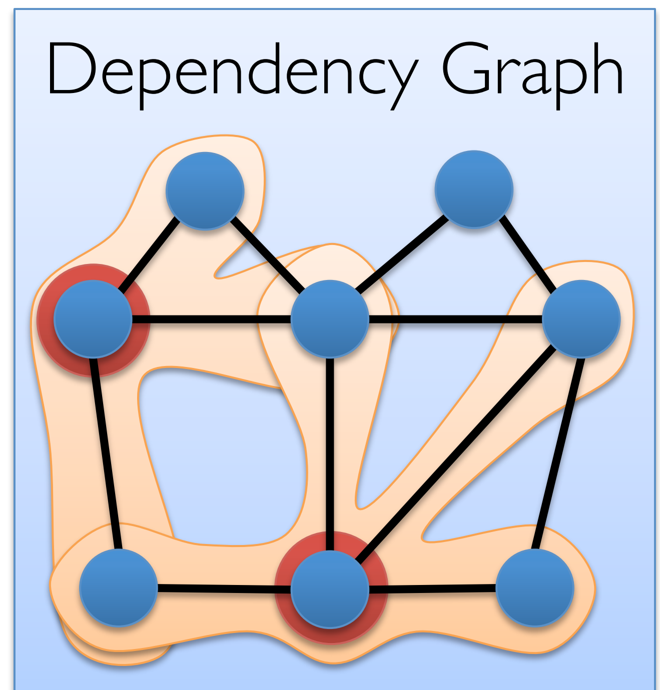
Spark



Graph View

Pregel GraphLab

APACHE GIRAPH



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

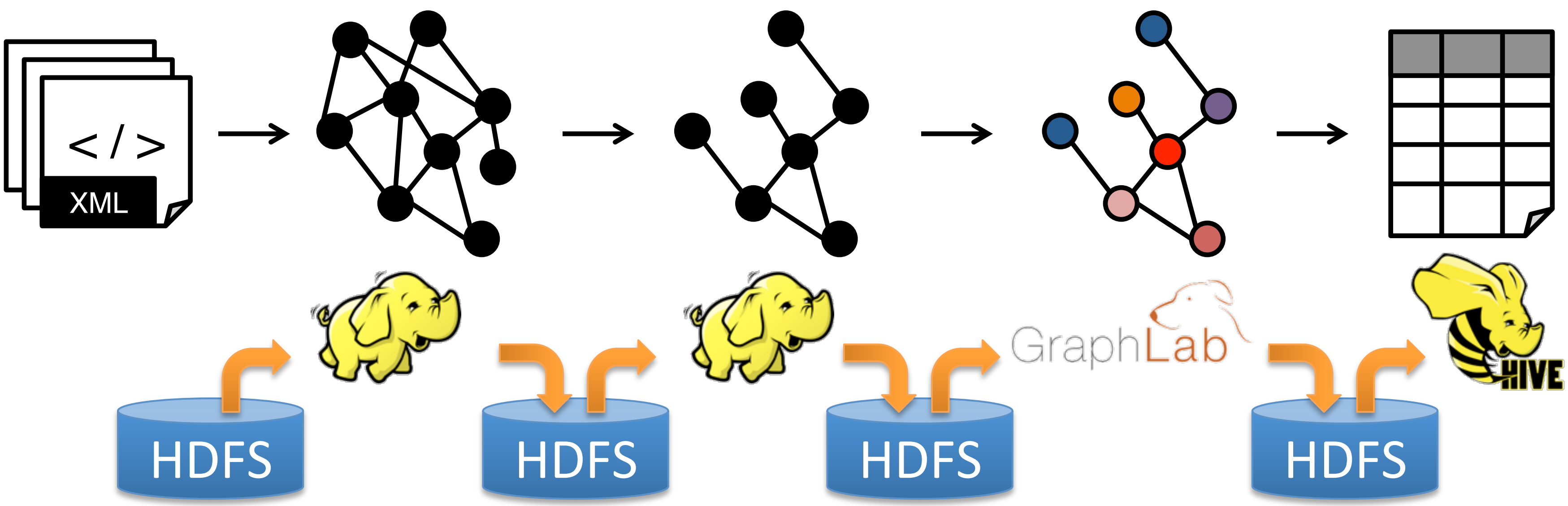
Difficult to Program and Use

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

Leads to brittle and often complex interfaces

Inefficient

Extensive data movement and duplication across the network and file system

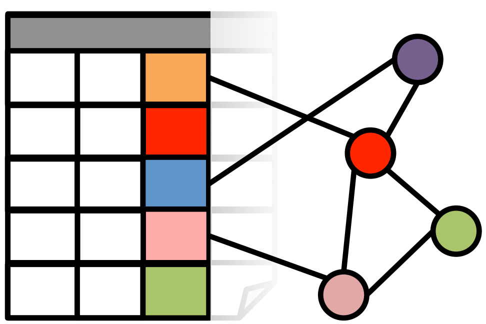


Limited reuse internal data-structures across stages

Solution: The GraphX Unified Approach

New API

Blurs the distinction between Tables and Graphs

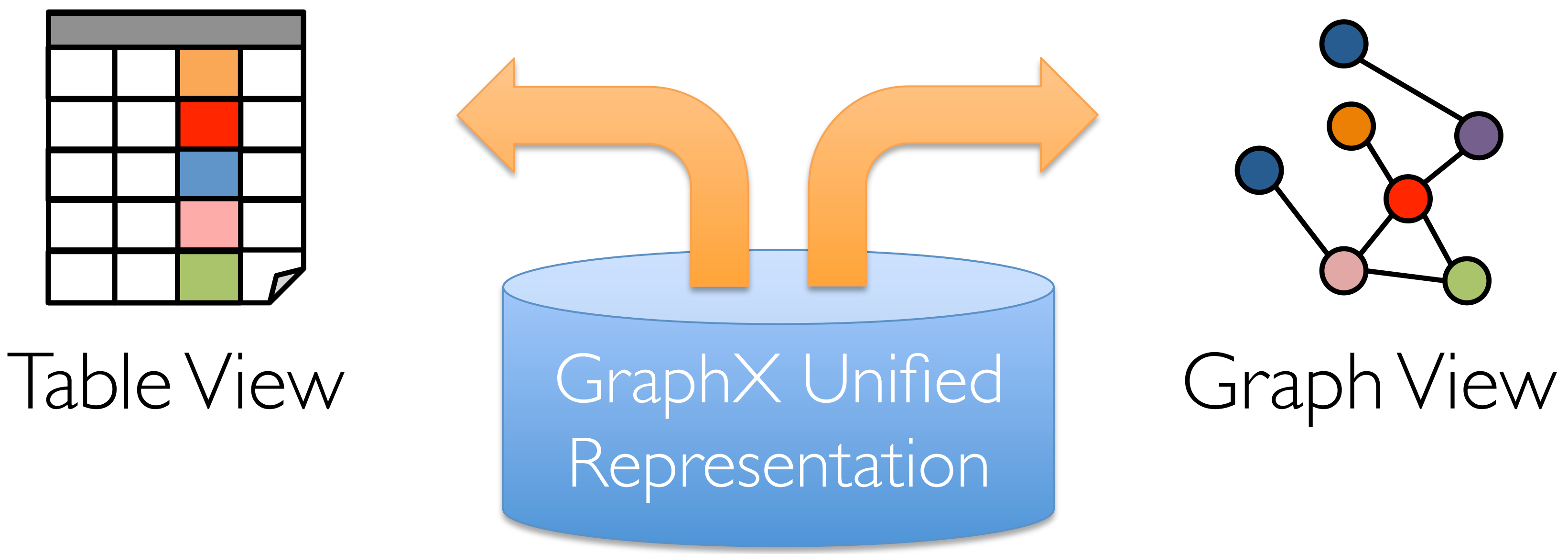


New System

Combines Data-Parallel Graph-Parallel Systems

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

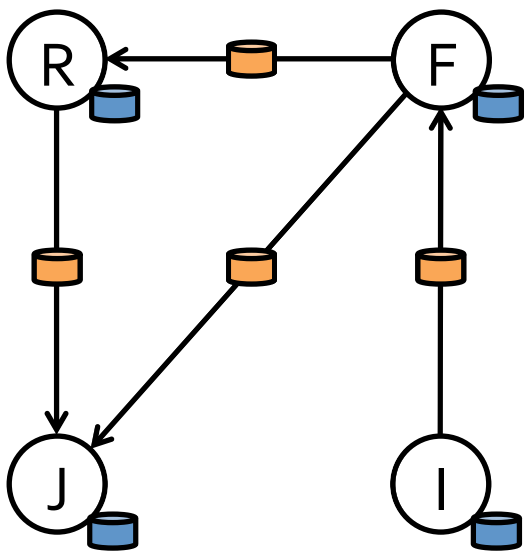
Tables and Graphs are composable *views* of the *same physical* data



Each view has its own operators that exploit the semantics of the view to achieve efficient execution

View a Graph as a Table

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

Table Operators

Table (RDD) operators are inherited from Spark:

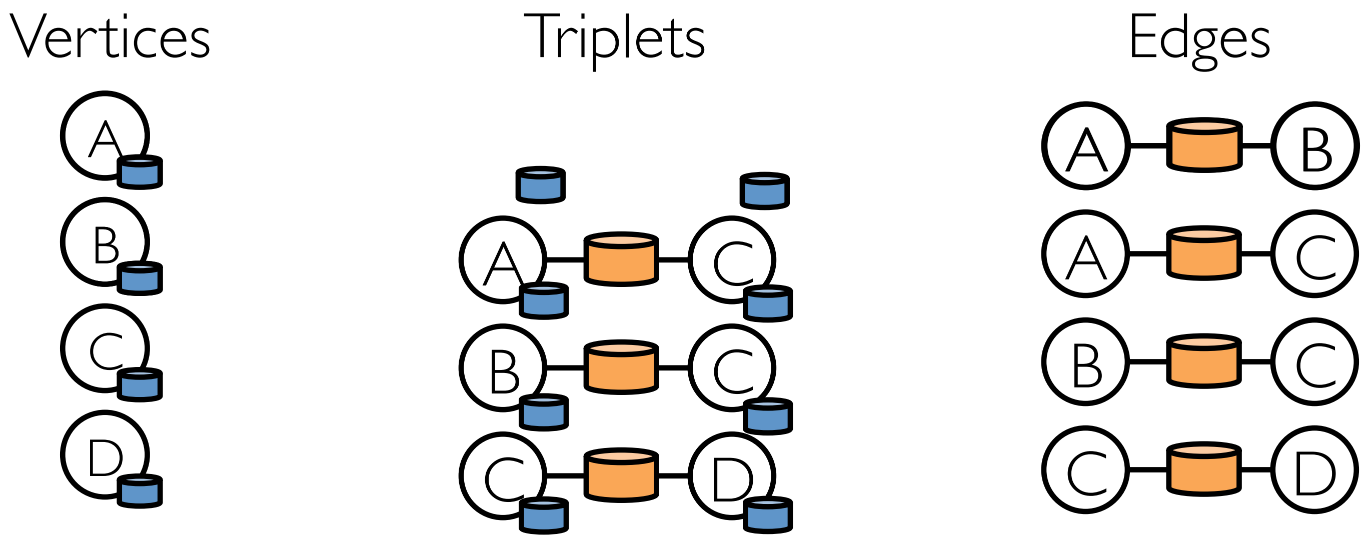
`map`	`reduce`	`sample`
`filter`	`count`	`take`
`groupBy`	`fold`	`first`
`sort`	`reduceByKey`	`partitionBy`
`union`	`groupByKey`	`mapWith`
`join`	`cogroup`	`pipe`
`leftOuterJoin`	`cross`	`save`
`rightOuterJoin`	`zip`	`...`

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:

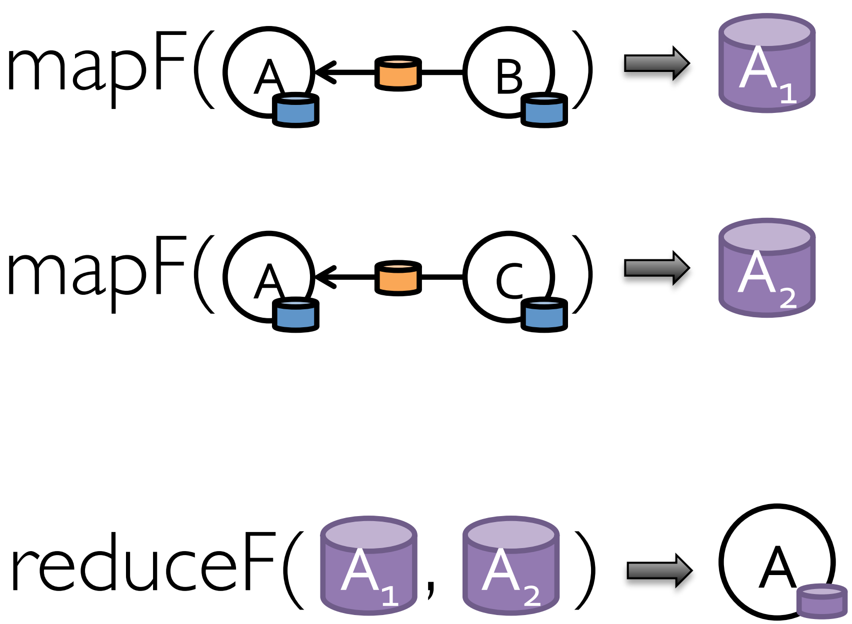


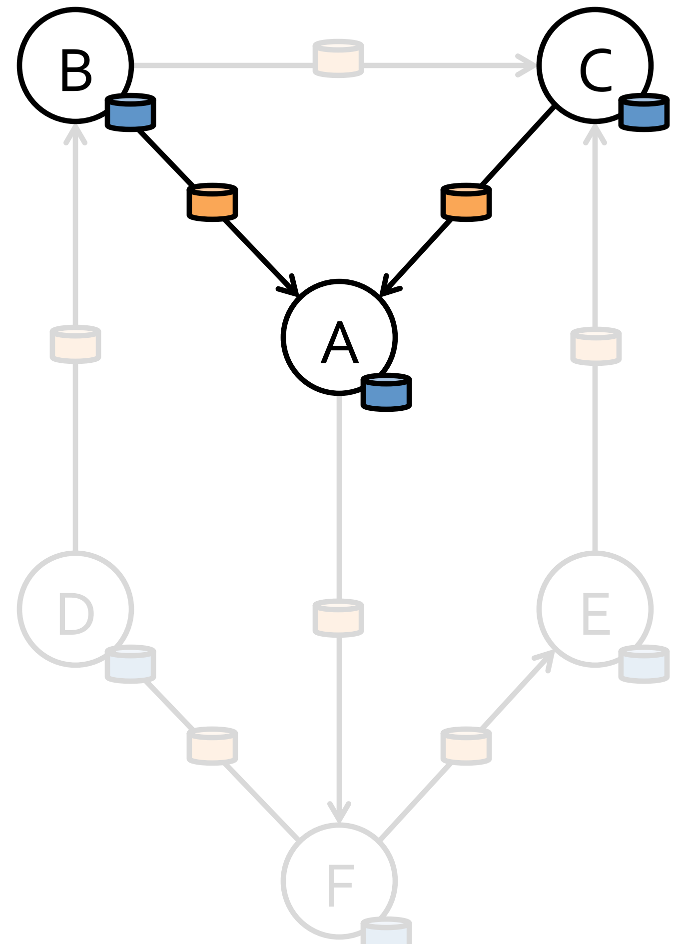
The *mrTriplets* operator sums adjacent triplets.

SELECT t.dstId, *reduceUDF*(*mapUDF*(t)) **AS** sum
FROM triplets **AS** t **GROUPBY** t.dstId

Map Reduce Triplets

Map-Reduce for each vertex

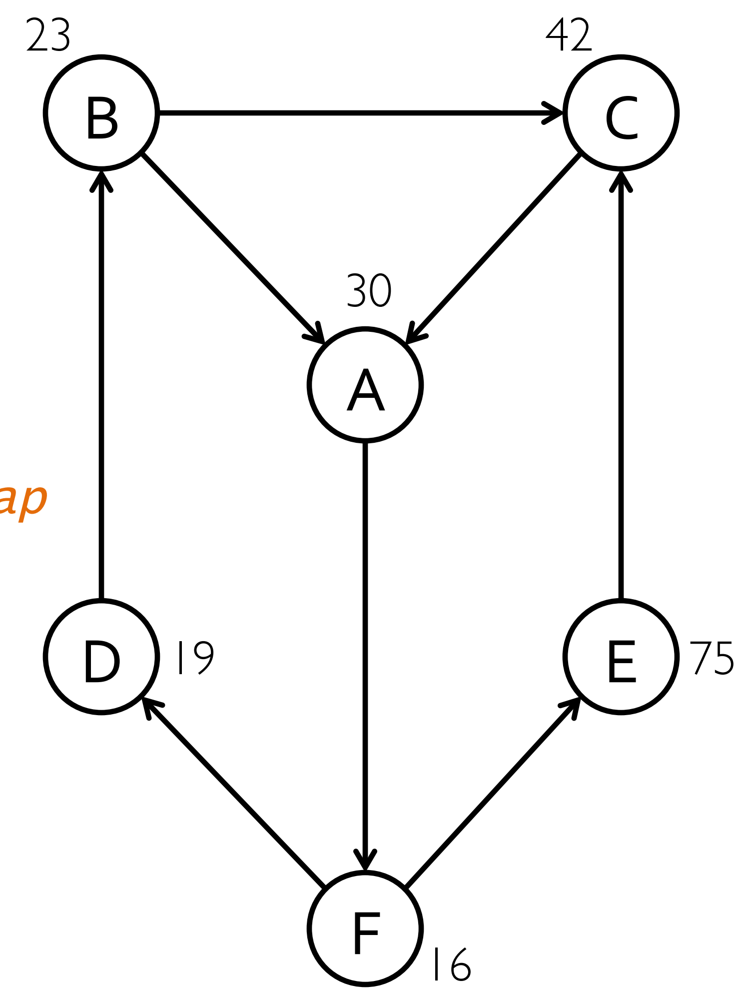




Example: Oldest Follower

What is the age of the oldest follower for each user?

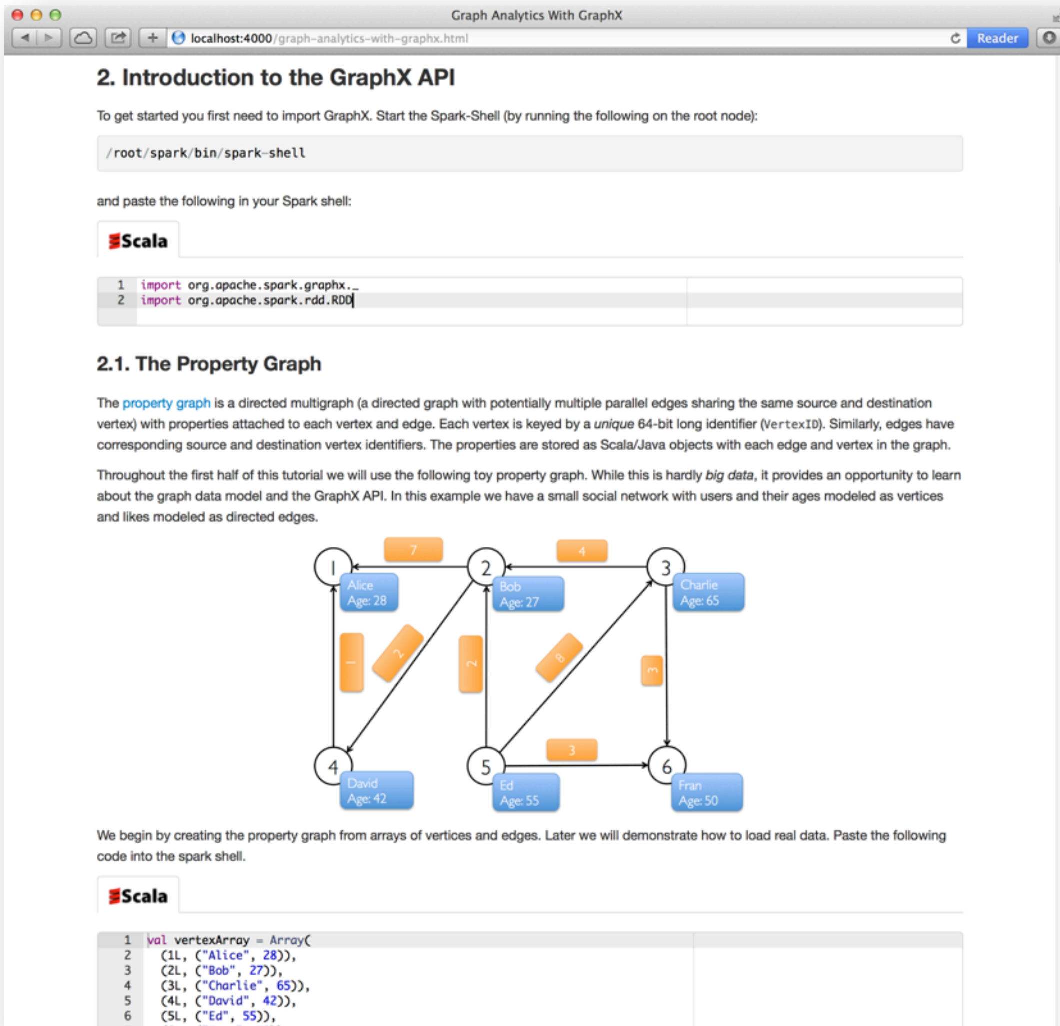
```
val oldestFollowerAge = graph
  .mrTriplets(
    e=> (e.dst.id, e.src.age),//Map
    (a,b)=> max(a, b) //Reduce
  )
  .vertices
```



We express the Pregel and GraphLab abstractions using the GraphX operators in less than 50 lines of code!

By composing these operators we can construct entire graph-analytics pipelines.

DIY Demo this Afternoon


Graph Analytics With GraphX

localhost:4000/graph-analytics-with-graphx.html

Reader

2. Introduction to the GraphX API

To get started you first need to import GraphX. Start the Spark-Shell (by running the following on the root node):

```
/root/spark/bin/spark-shell
```

and paste the following in your Spark shell:

Scala

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

2.1. The Property Graph

The property graph is a directed multigraph (a directed graph with potentially multiple parallel edges sharing the same source and destination vertex) with properties attached to each vertex and edge. Each vertex is keyed by a *unique* 64-bit long identifier (`VertexID`). Similarly, edges have corresponding source and destination vertex identifiers. The properties are stored as Scala/Java objects with each edge and vertex in the graph.

Throughout the first half of this tutorial we will use the following toy property graph. While this is hardly *big data*, it provides an opportunity to learn about the graph data model and the GraphX API. In this example we have a small social network with users and their ages modeled as vertices and likes modeled as directed edges.

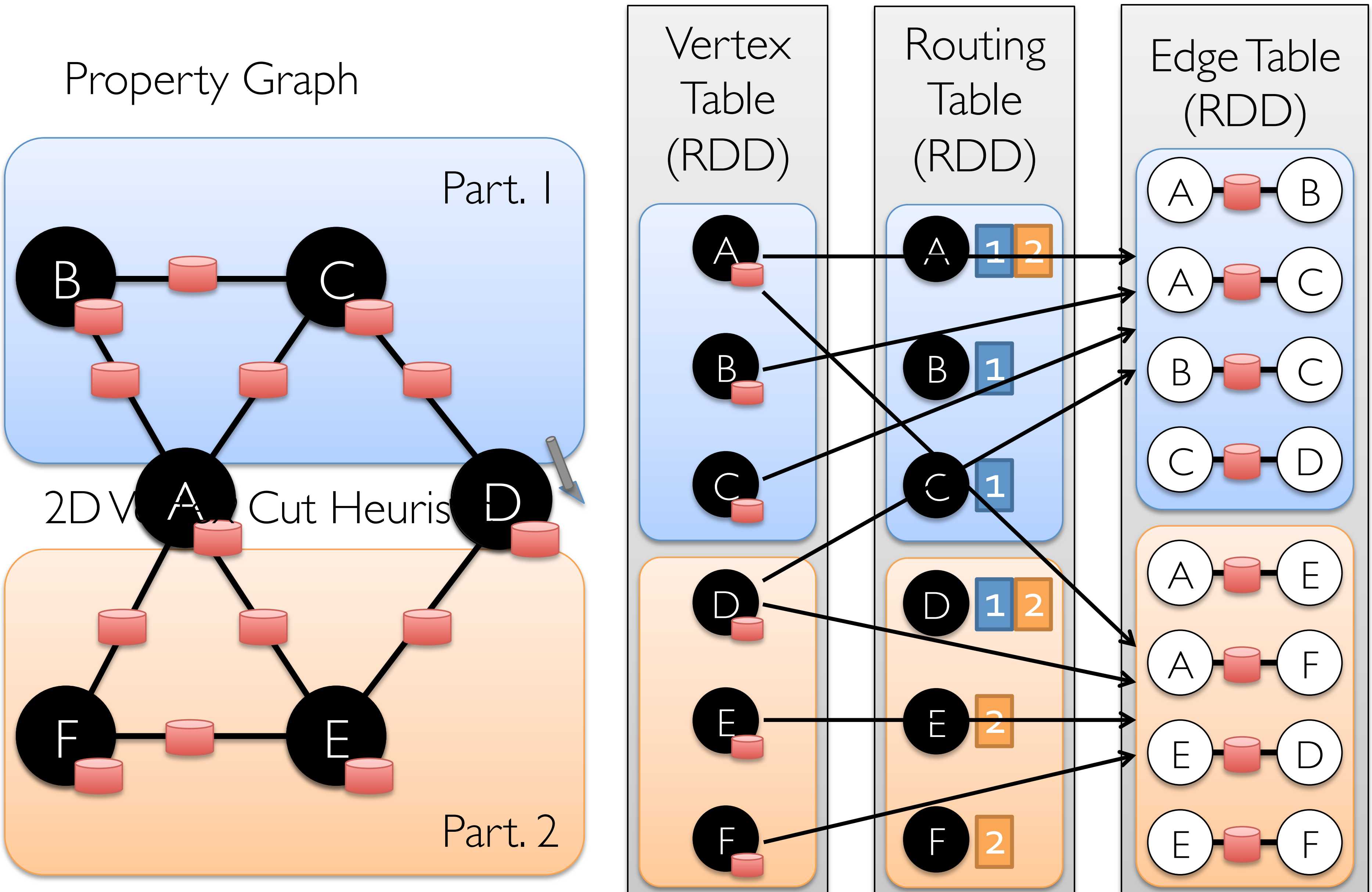
We begin by creating the property graph from arrays of vertices and edges. Later we will demonstrate how to load real data. Paste the following code into the spark shell.

Scala

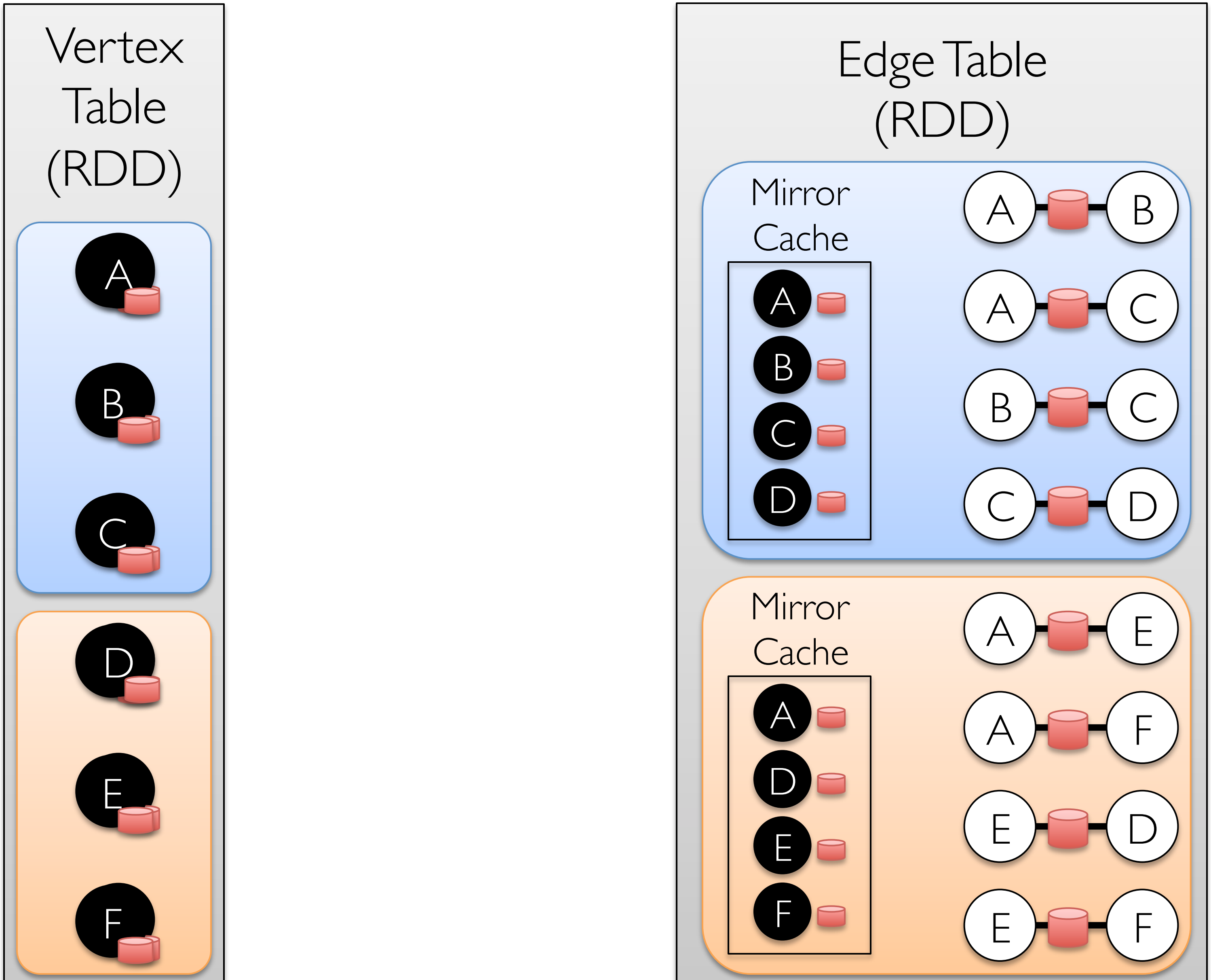
```
val vertexArray = Array(
  (1L, ("Alice", 28)),
  (2L, ("Bob", 27)),
  (3L, ("Charlie", 65)),
  (4L, ("David", 42)),
  (5L, ("Ed", 55)),
  (6L, ("Fran", 50))
```

GraphX System Design

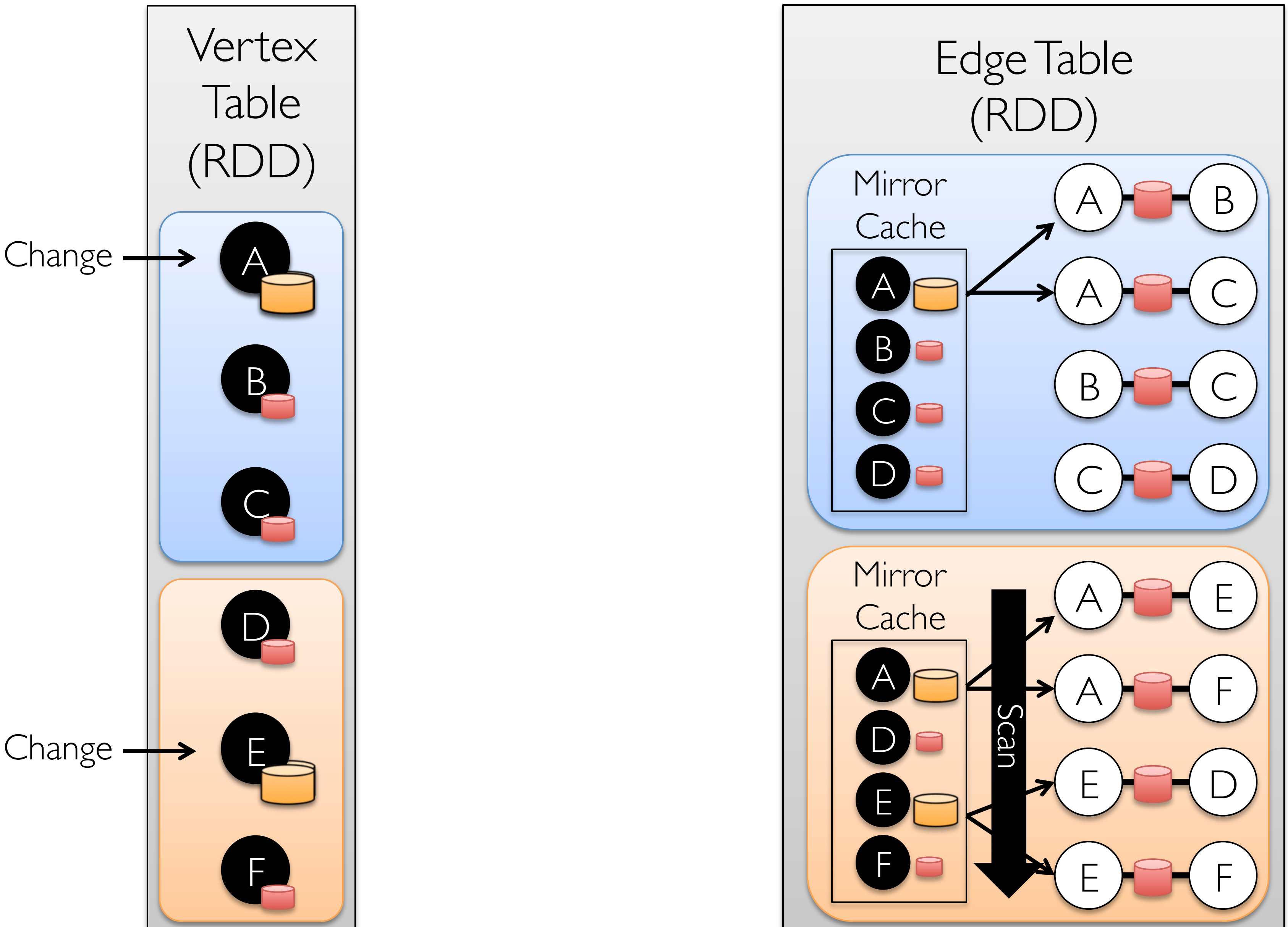
Distributed Graphs as Tables (RDDs)



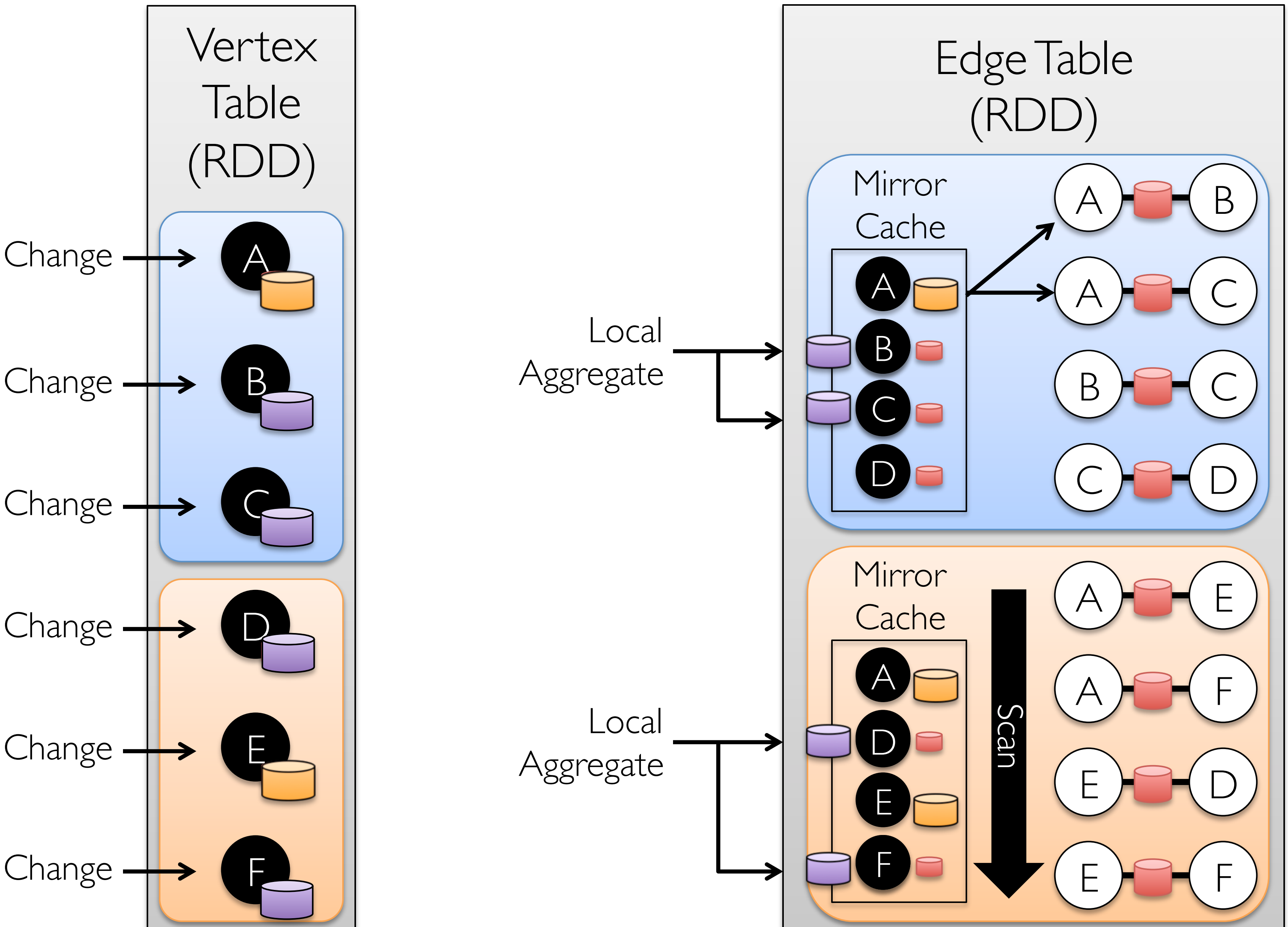
Caching for Iterative mrTriplets



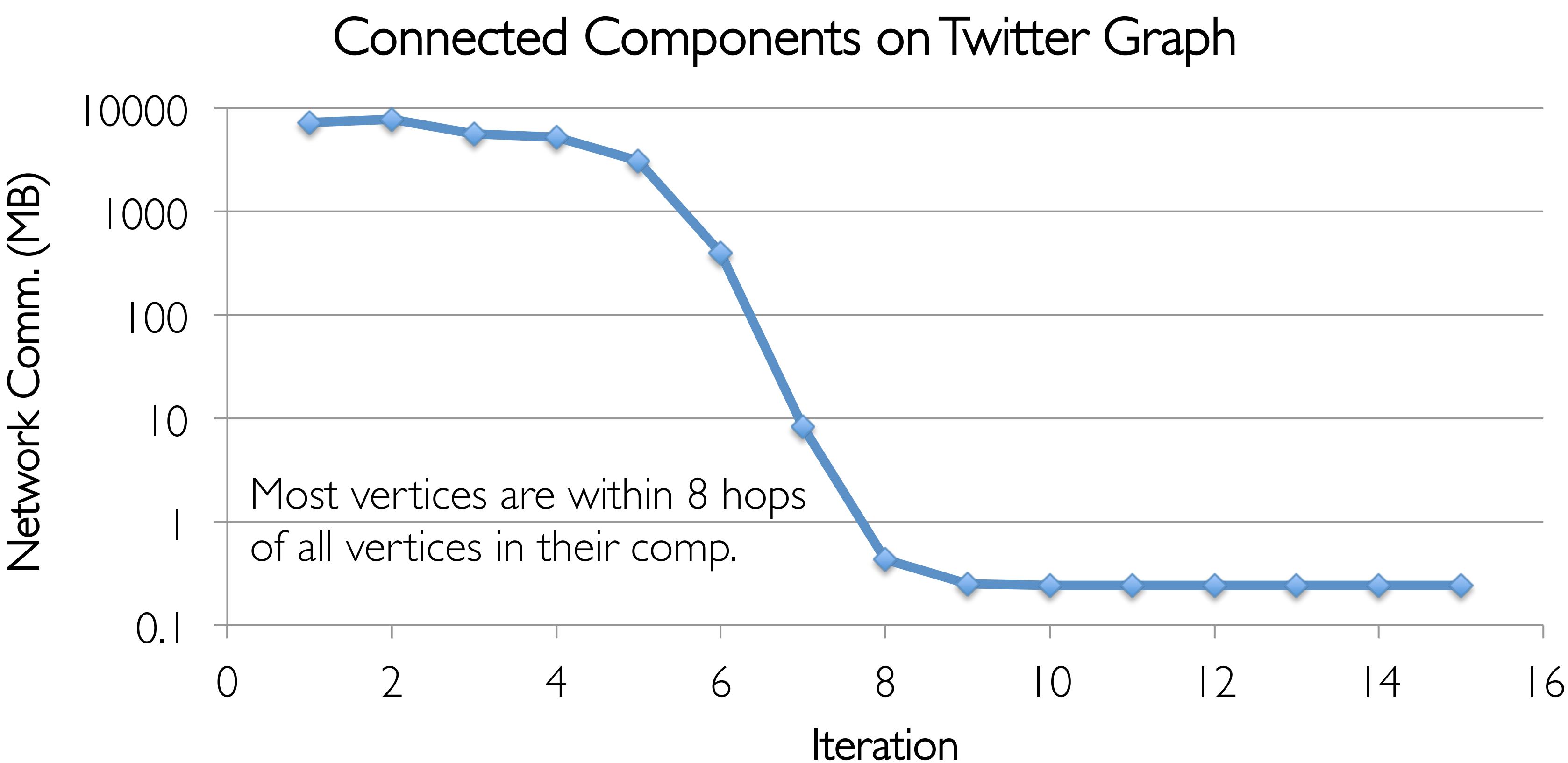
Incremental Updates for Iterative mrTriplets



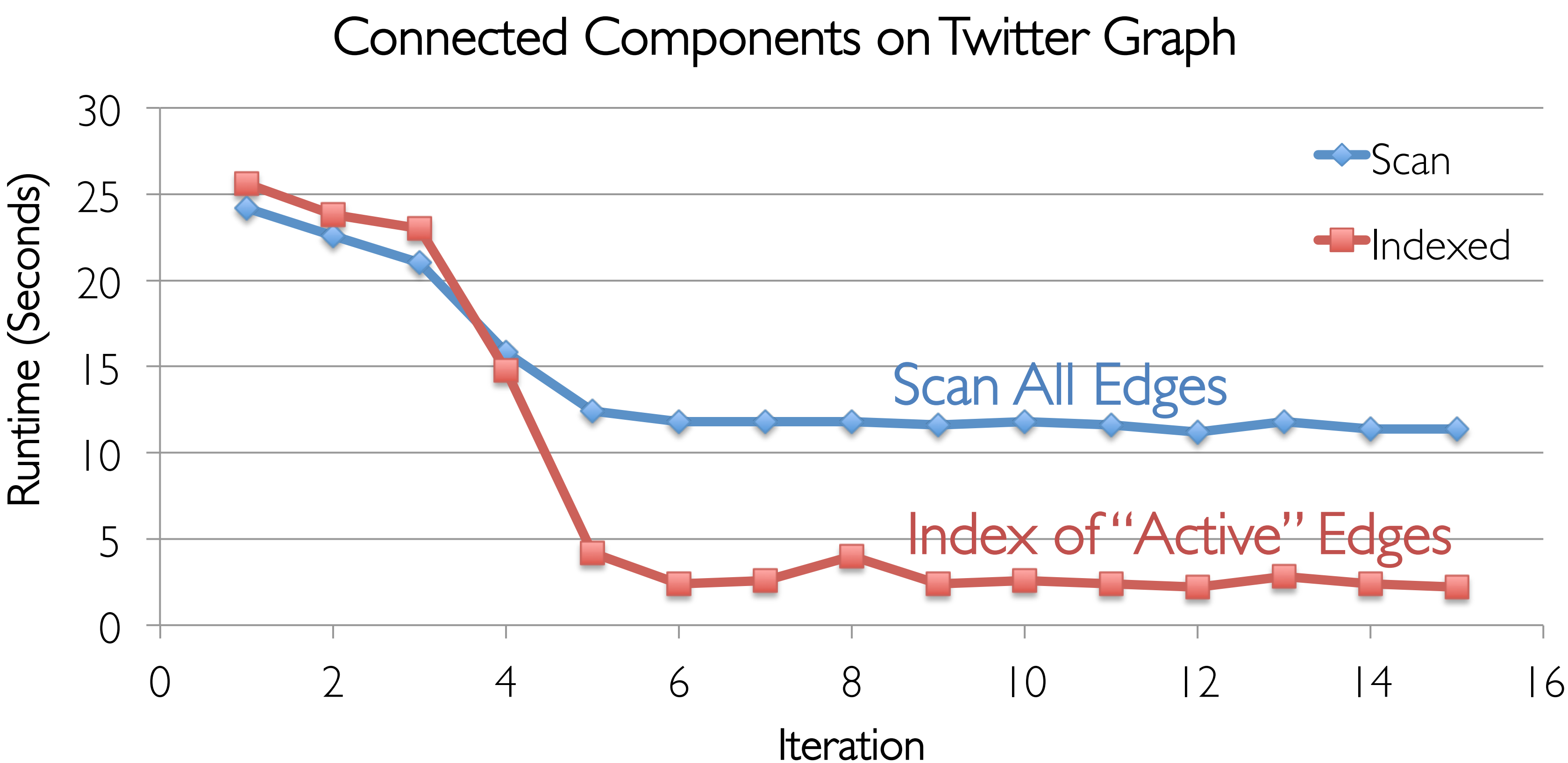
Aggregation for Iterative mrTriplets



Reduction in Communication Due to Cached Updates



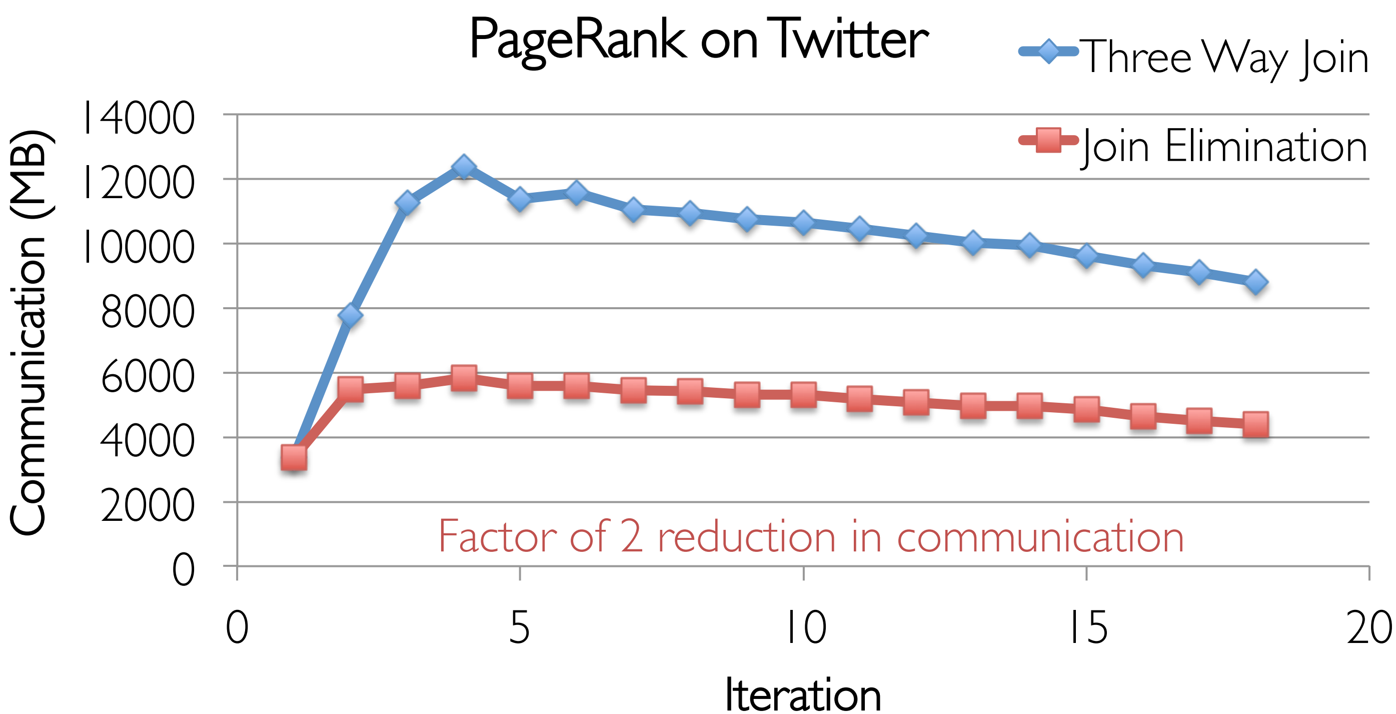
Benefit of Indexing *Active* Edges



Join Elimination

Identify and bypass joins for unused triplets fields

» *Example:* PageRank only accesses source attribute



Additional Query Optimizations

Indexing and Bitmaps:

» To accelerate joins across graphs

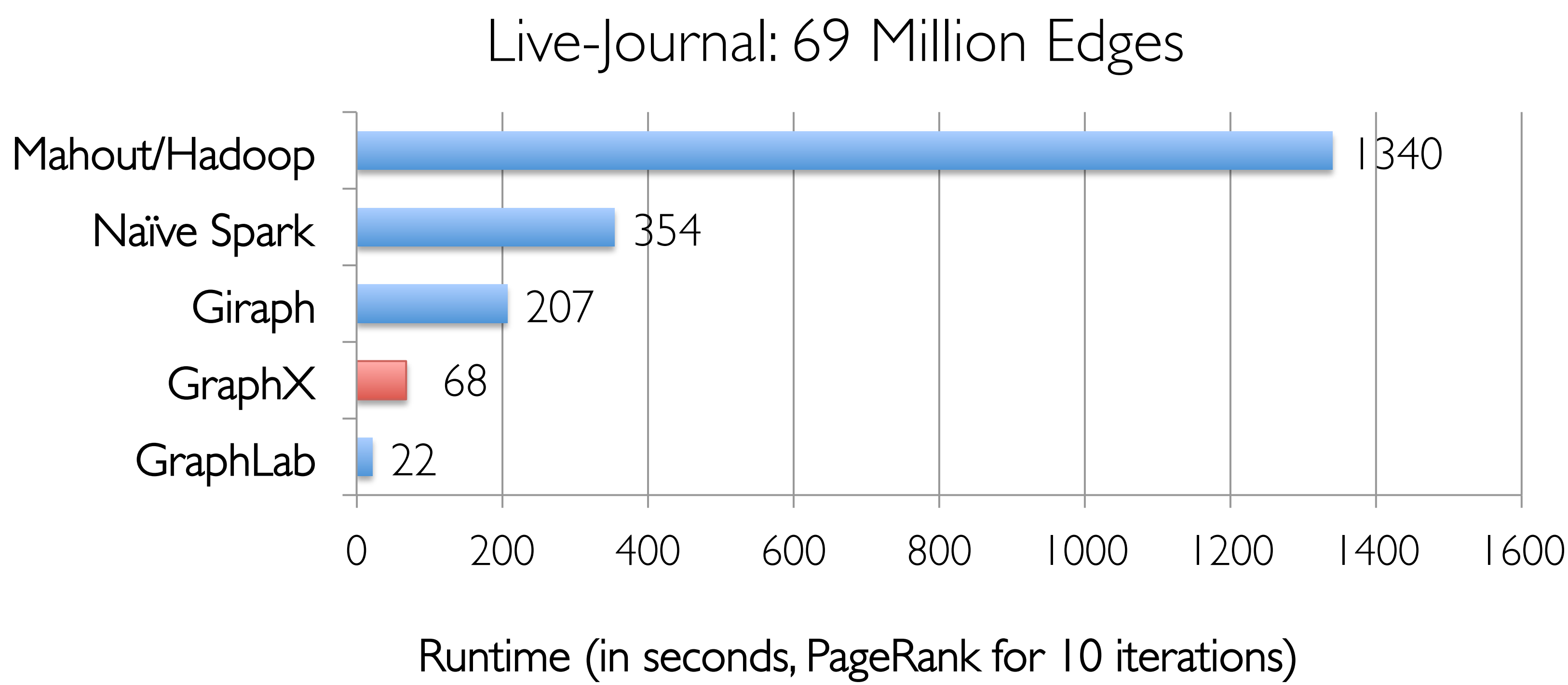
» To efficiently construct sub-graphs

Substantial Index and Data Reuse:

» Reuse routing tables across graphs and sub-graphs

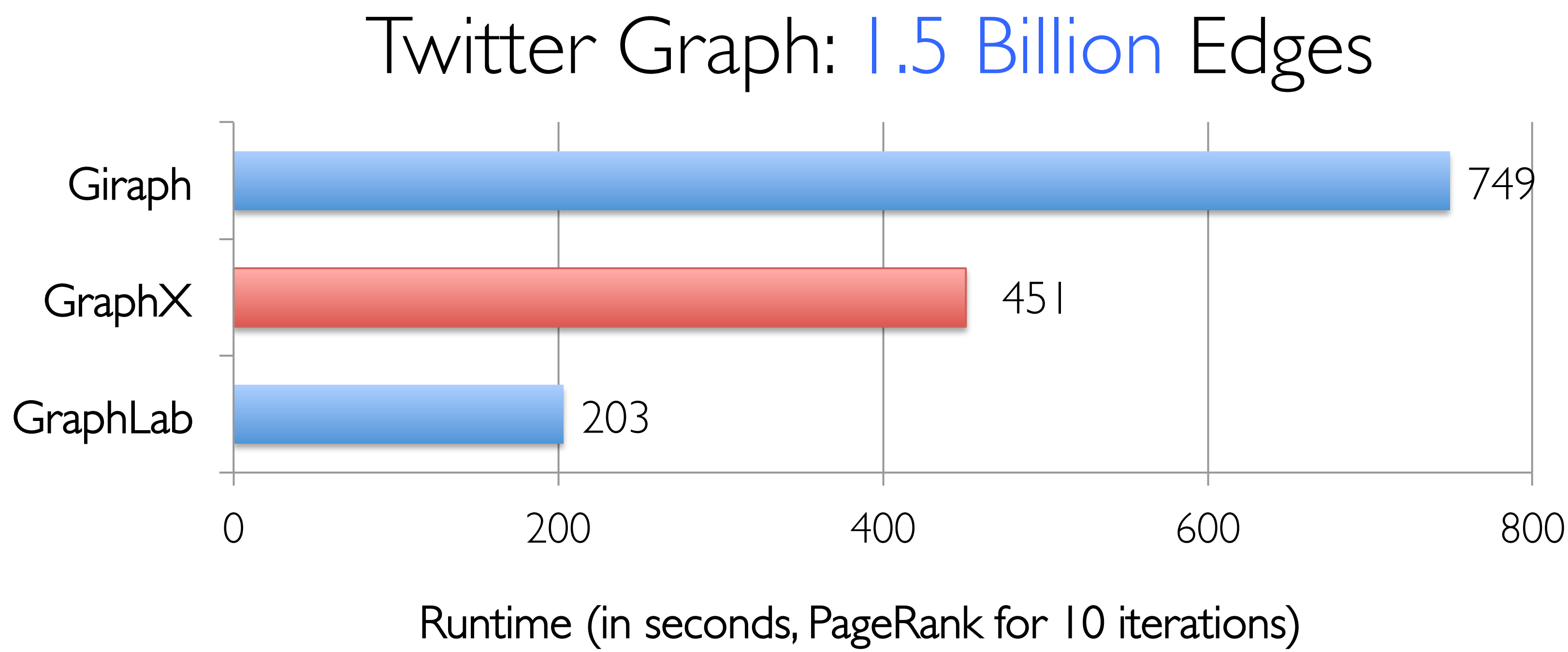
» Reuse edge adjacency information and indices

Performance Comparisons



GraphX is roughly 3x slower than GraphLab

GraphX scales to larger graphs



GraphX is roughly 2x slower than GraphLab

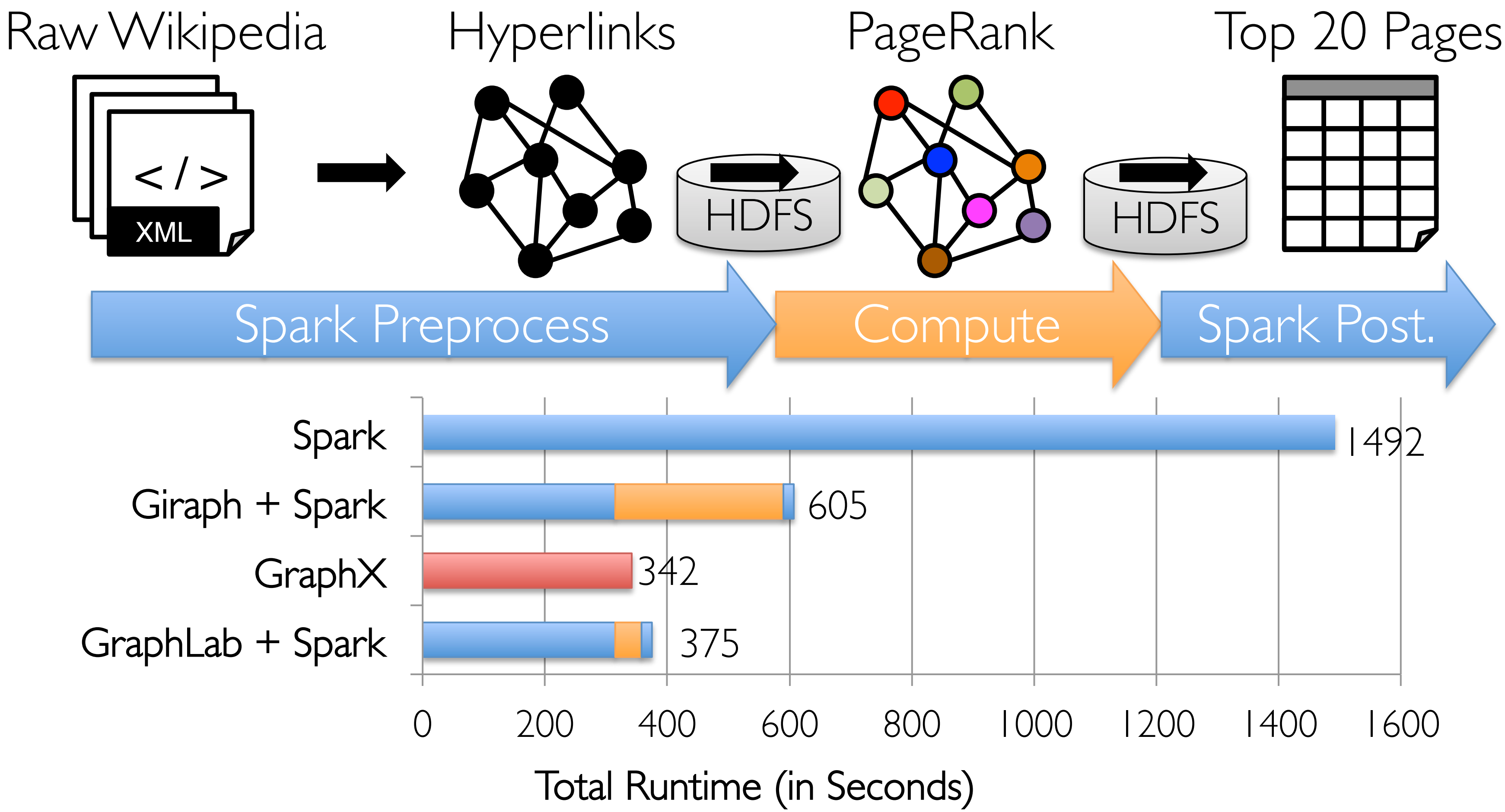
» Scala + Java overhead: Lambdas, GC time, …

» No shared memory parallelism: 2x increase in comm.

PageRank is just one stage….

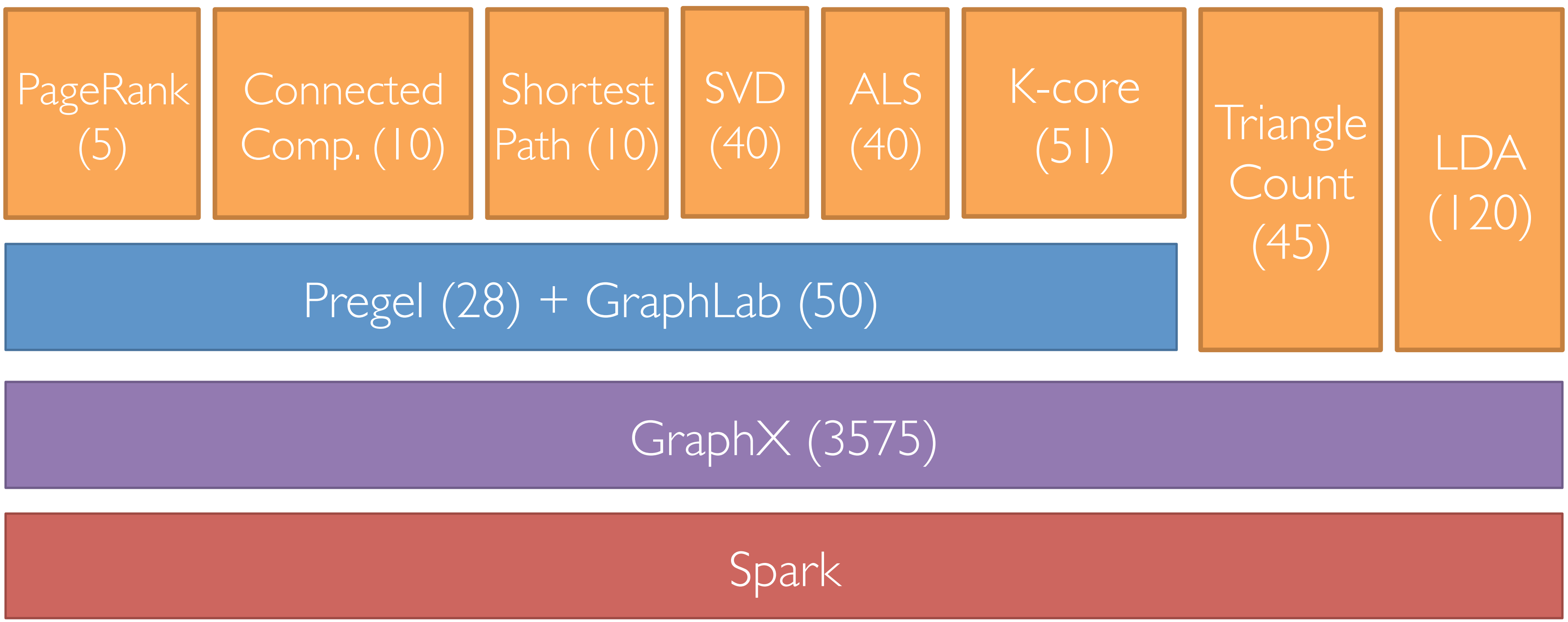
What about a pipeline?

A Small Pipeline in GraphX



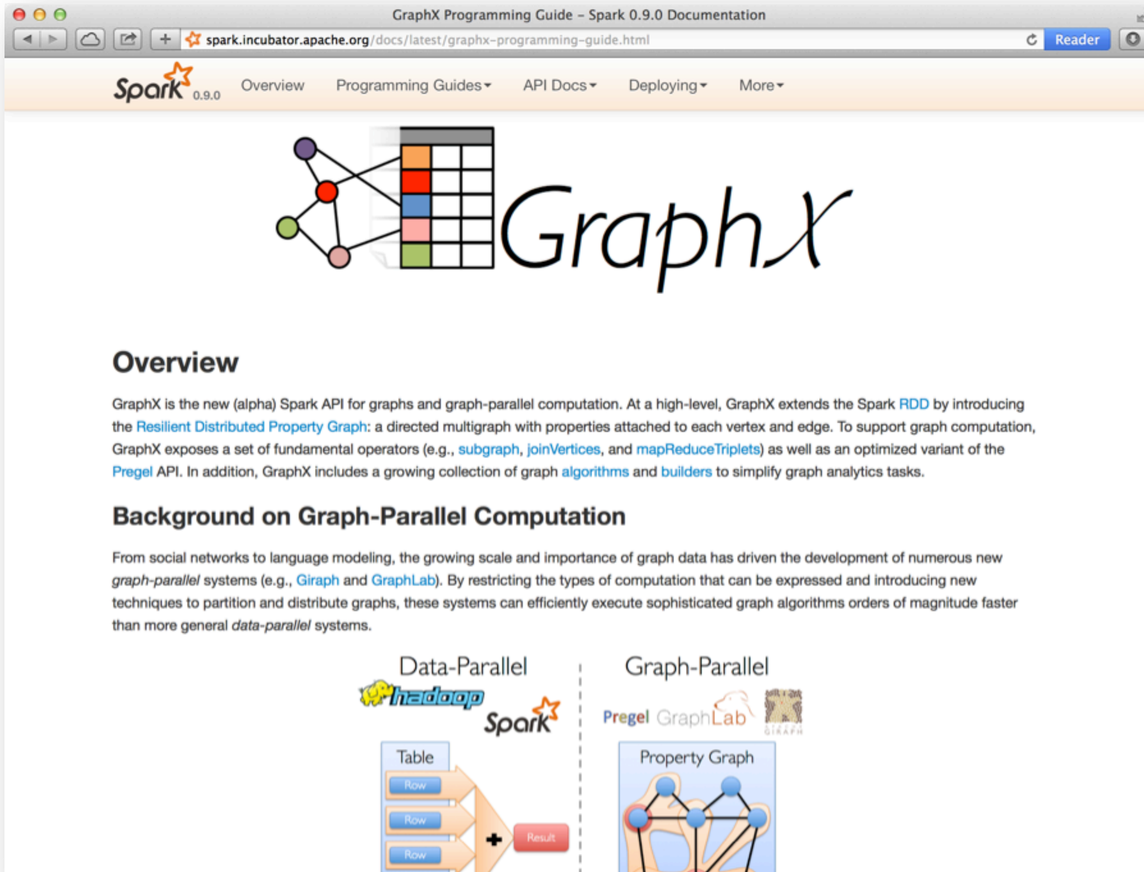
Timed end-to-end GraphX is *faster* than GraphLab

The GraphX Stack
(Lines of Code)



Status

Alpha release as part of Spark 0.9



Seeking collaborators and feedback

Conclusion and Observations

Domain specific views: *Tables* and *Graphs*

» tables and graphs are first-class composable objects

» specialized operators which exploit view semantics

Single system that efficiently spans the pipeline

» minimize data movement and duplication

» eliminates need to learn and manage multiple systems

Graphs through the lens of database systems

» Graph-Parallel Pattern → Triplet joins in relational alg.

» Graph Systems → Distributed join optimizations

Active Research

Static Data → Dynamic Data

» Apply GraphX unified approach to time evolving data

» Model and analyze relationships over time

Serving Graph Structured Data

» Allow external systems to interact with GraphX

» Unify distributed graph databases with relational database technology

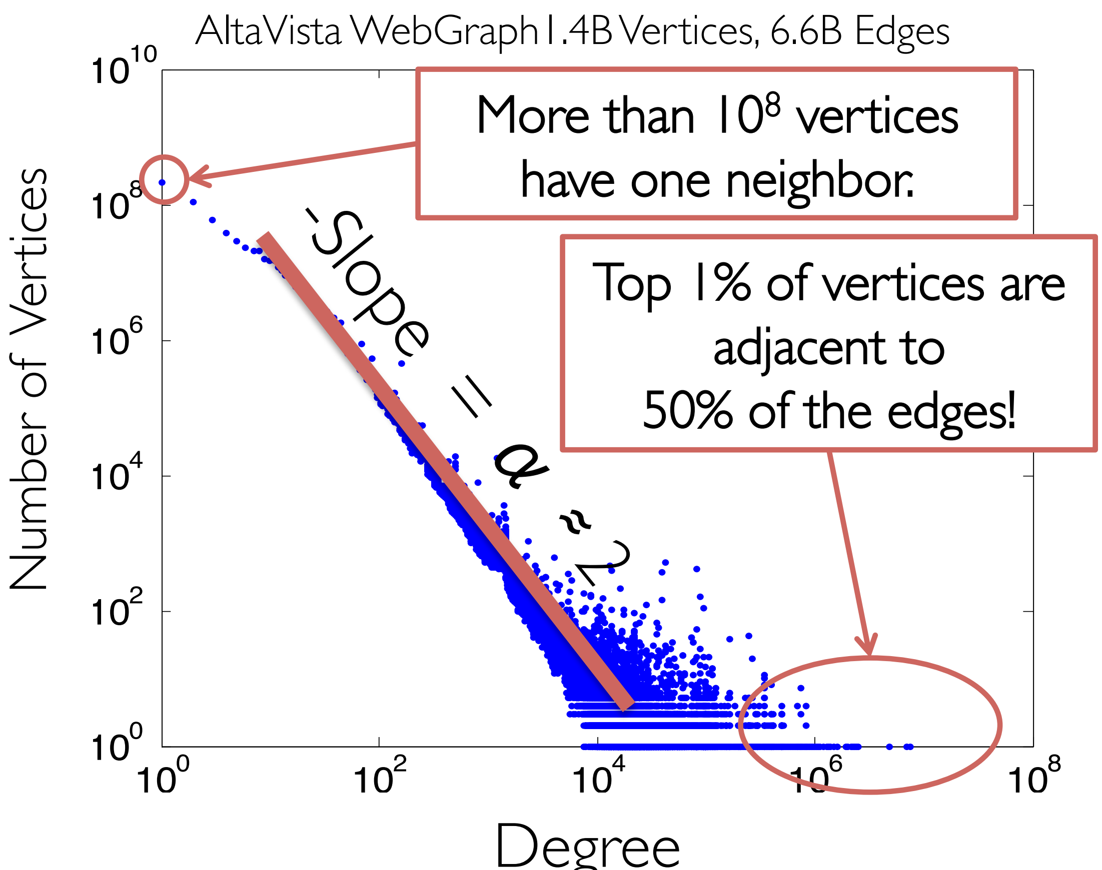
Thanks!

http://amplab.github.io/graphx/

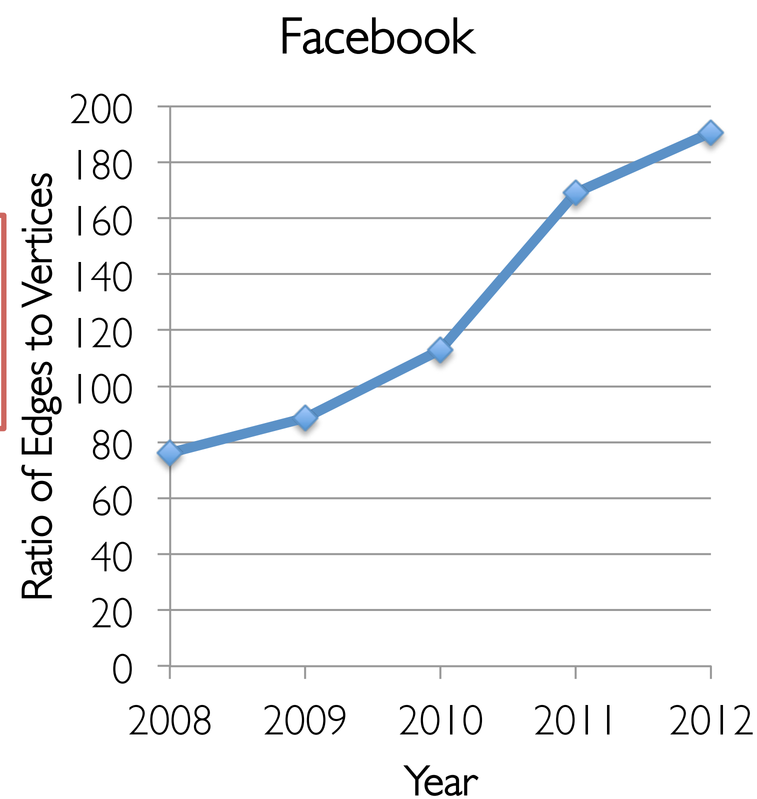
ankurd@eecs.berkeley.edu
crankshaw@eecs.berkeley.edu
rxin@eecs.berkeley.edu
jegonzal@eecs.berkeley.edu

Graph Property 1
Real-World Graphs

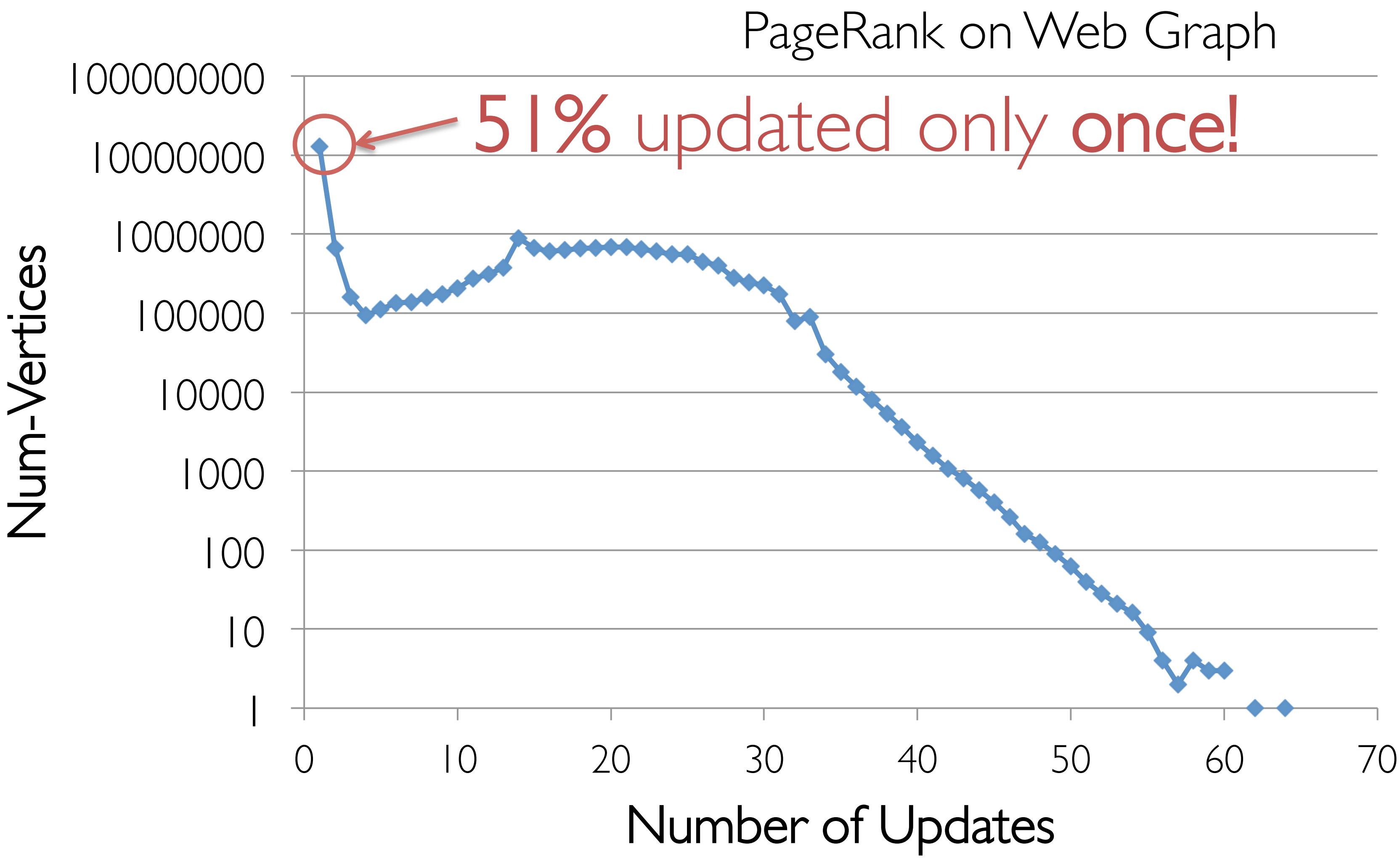
Power-Law Degree Distribution



Edges >> Vertices



Graph Property 2
Active Vertices



Graphs are Essential to Data Mining and Machine Learning

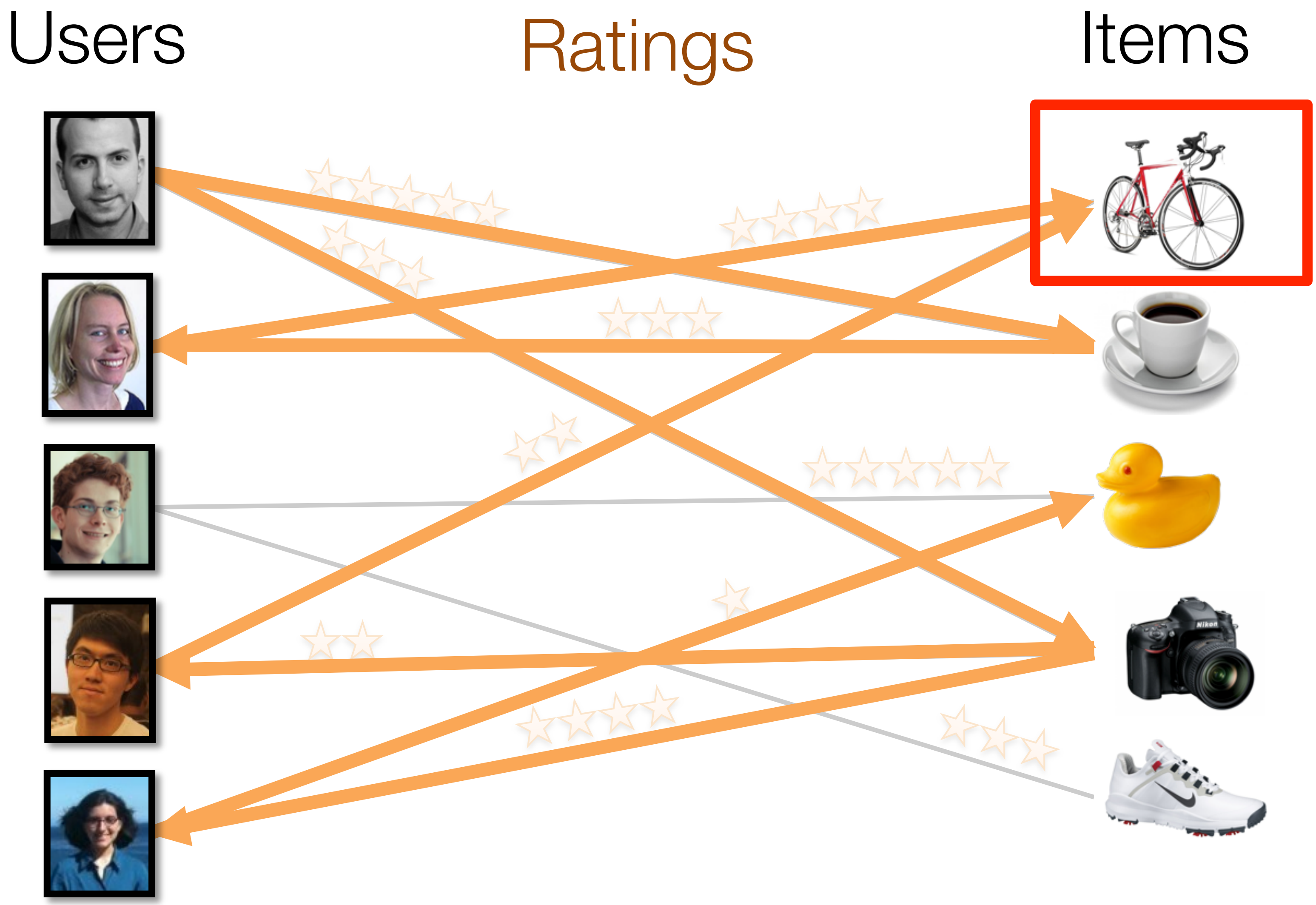
Identify influential people and information

Find communities

Understand people's shared interests

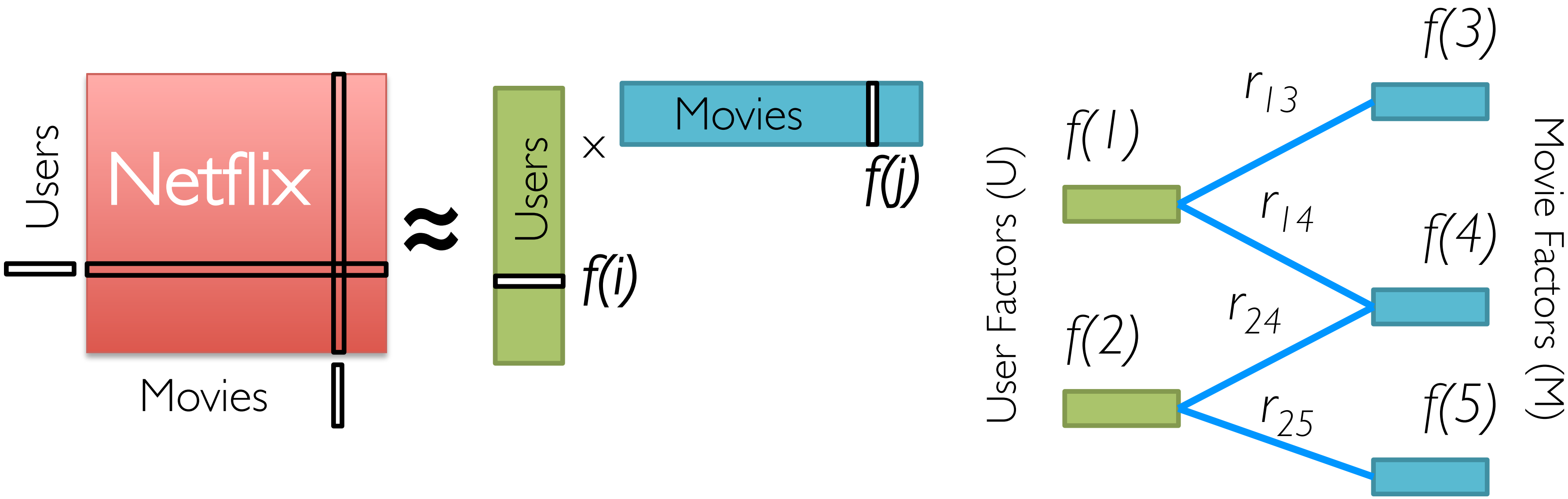
Model complex data dependencies

Recommending Products



Recommending Products

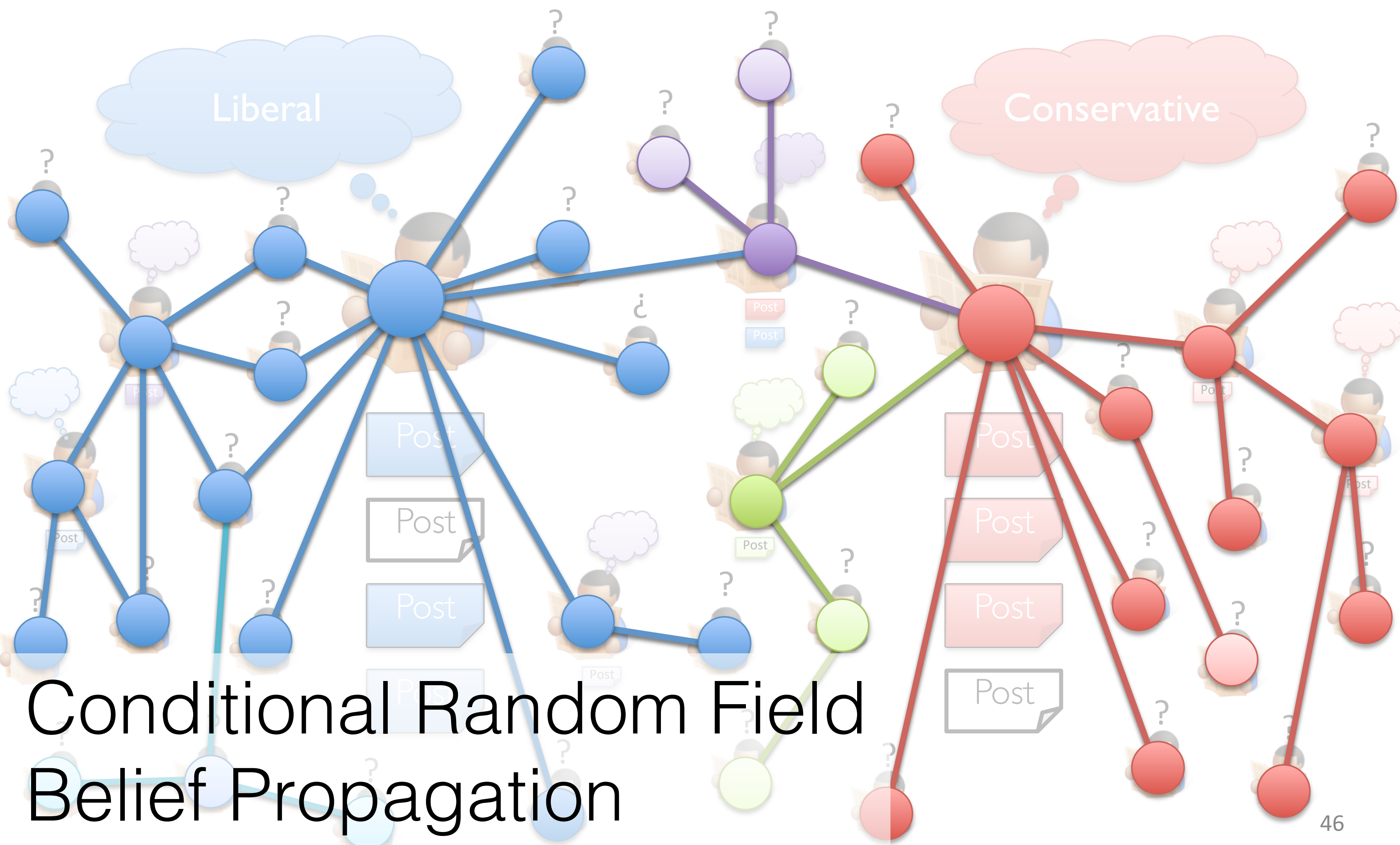
Low-Rank Matrix Factorization:



Iterate:

$$f[i] = \arg\min_{w \in \mathbb{R}^d} \sum_{j \in \mathrm{Nbrs}(i)} \left(r_{ij} - w^T f[j]\right)^2 + \lambda ||w||_2^2$$

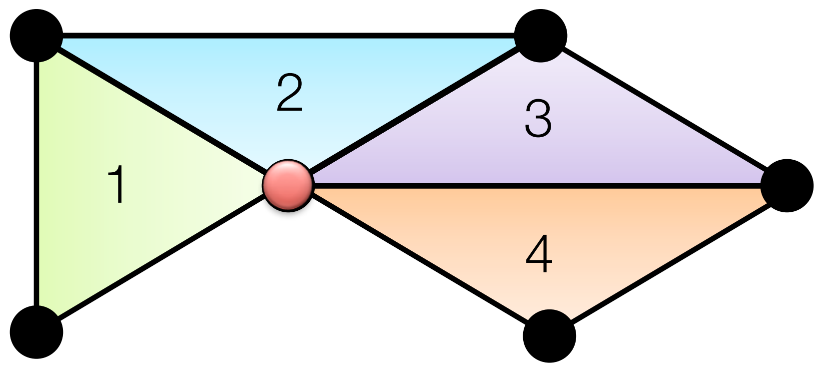
Predicting User Behavior



Conditional Random Field
Belief Propagation

Finding Communities

Count triangles passing through each vertex:



Measures "cohesiveness" of local community

Fewer Triangles
Weaker Community

More Triangles
Stronger Community

Example Graph Analytics Pipeline