GraphX:
Unifying Table and Graph Analytics

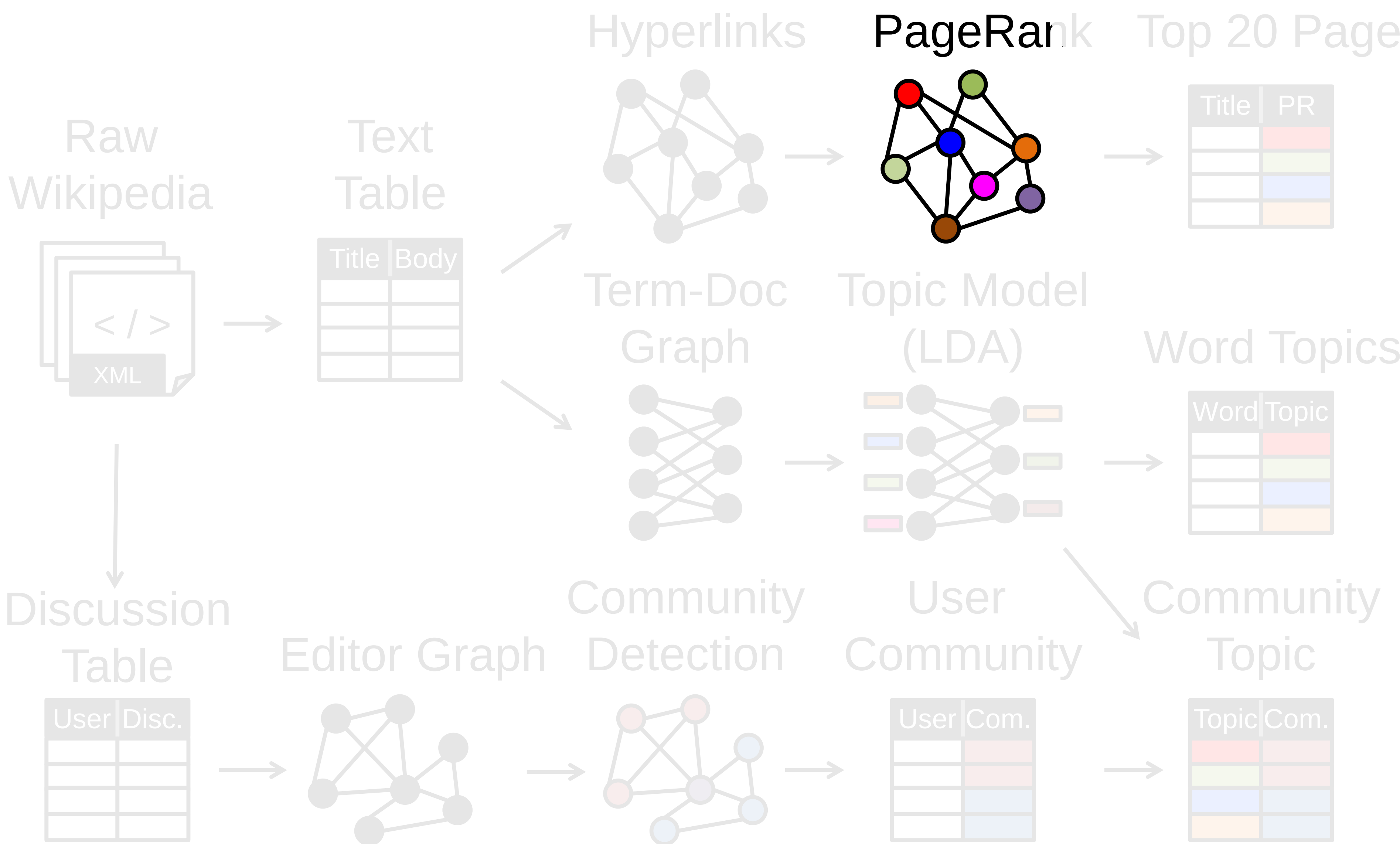
Presented by Joseph Gonzalez

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

http://www.istc-cc.cmu.edu/

Intel Science & Technology Center for Cloud Computing

Graphs are Central to Analytics



PageRank: Identifying Leaders

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

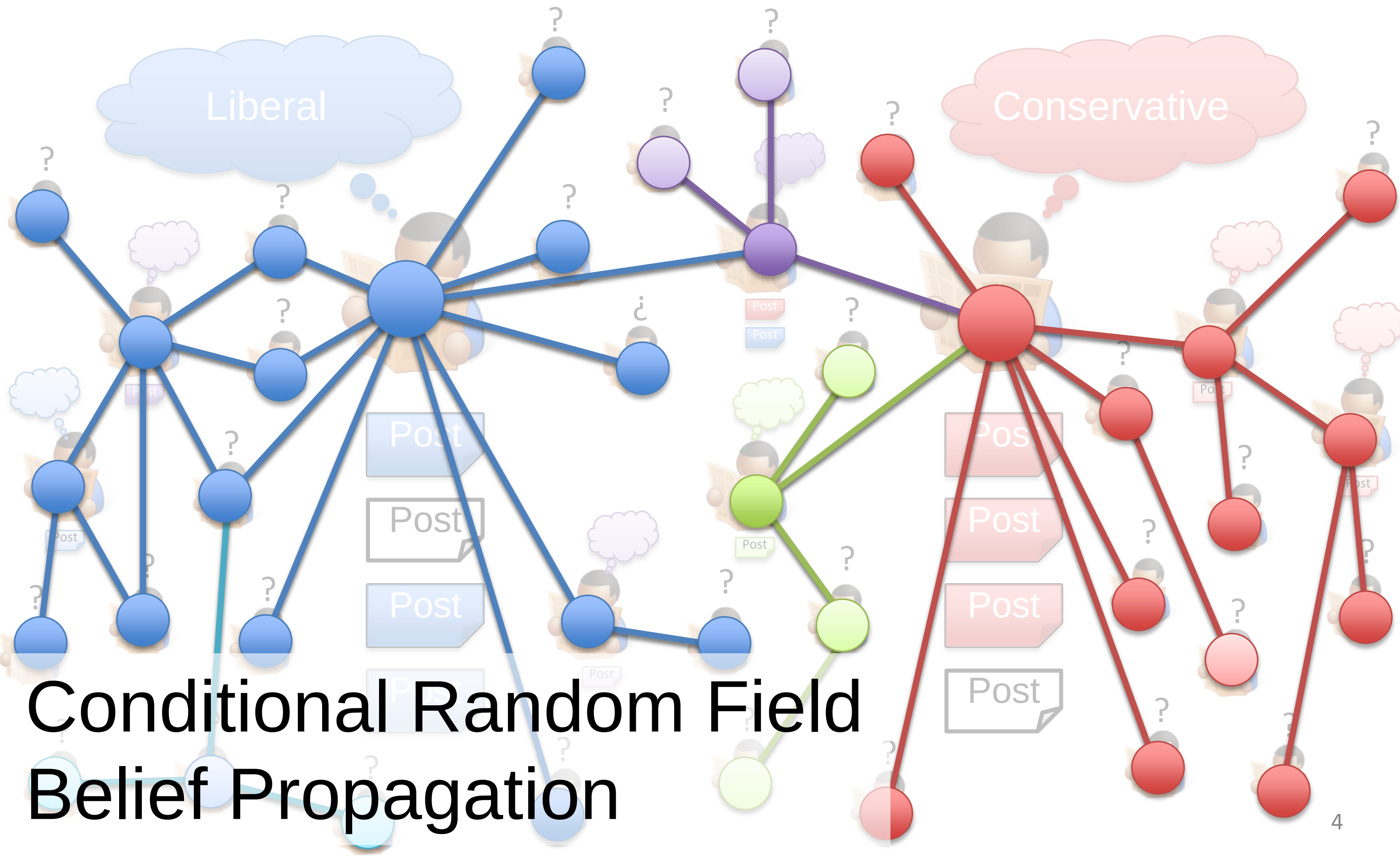
Rank of user *i*

Weighted sum of neighbors' ranks

Update ranks in parallel

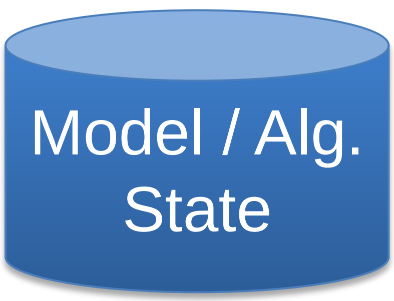
Iterate until convergence

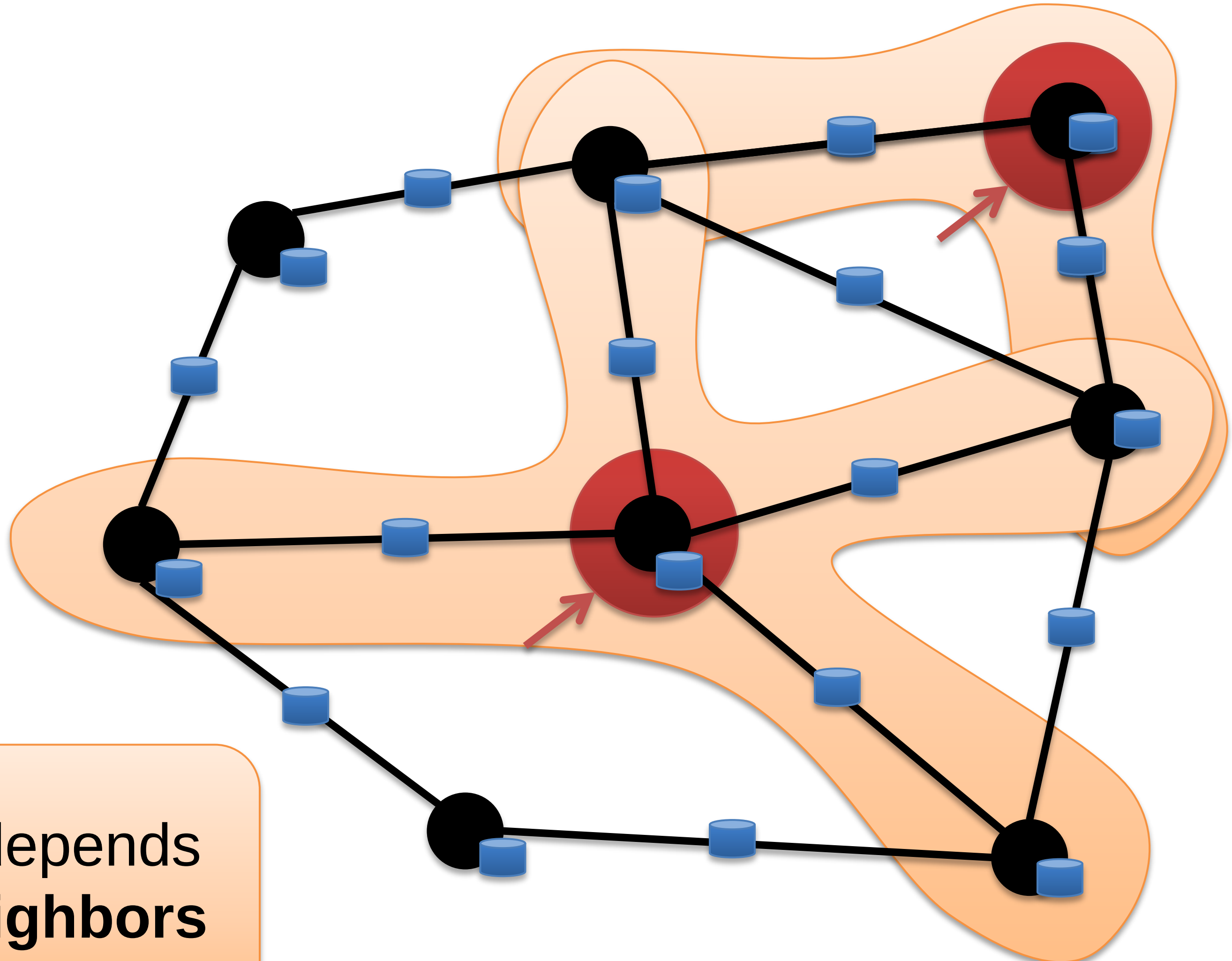
Predicting User Behavior



Conditional Random Field
Belief Propagation

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

MACHINE LEARNING

SOCIAL NETWORK ANALYSIS

GRAPH ALGORITHMS

Graph-Parallel Systems

Pregel
oogle

APACHE
GIRAPH

GraphLab

Expose specialized APIs to simplify graph programming.

“Think like a Vertex.”

- Pregel [SIGMOD’10]

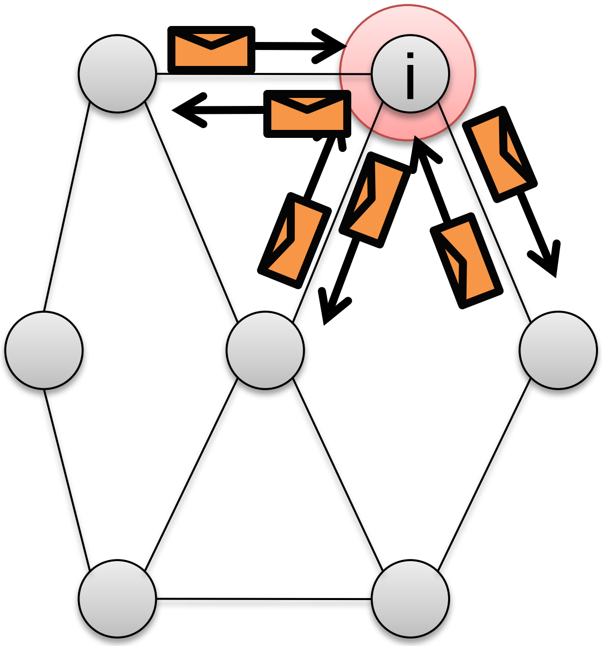
The Pregel (Push) Abstraction

Vertex-Programs interact by sending **messages**.

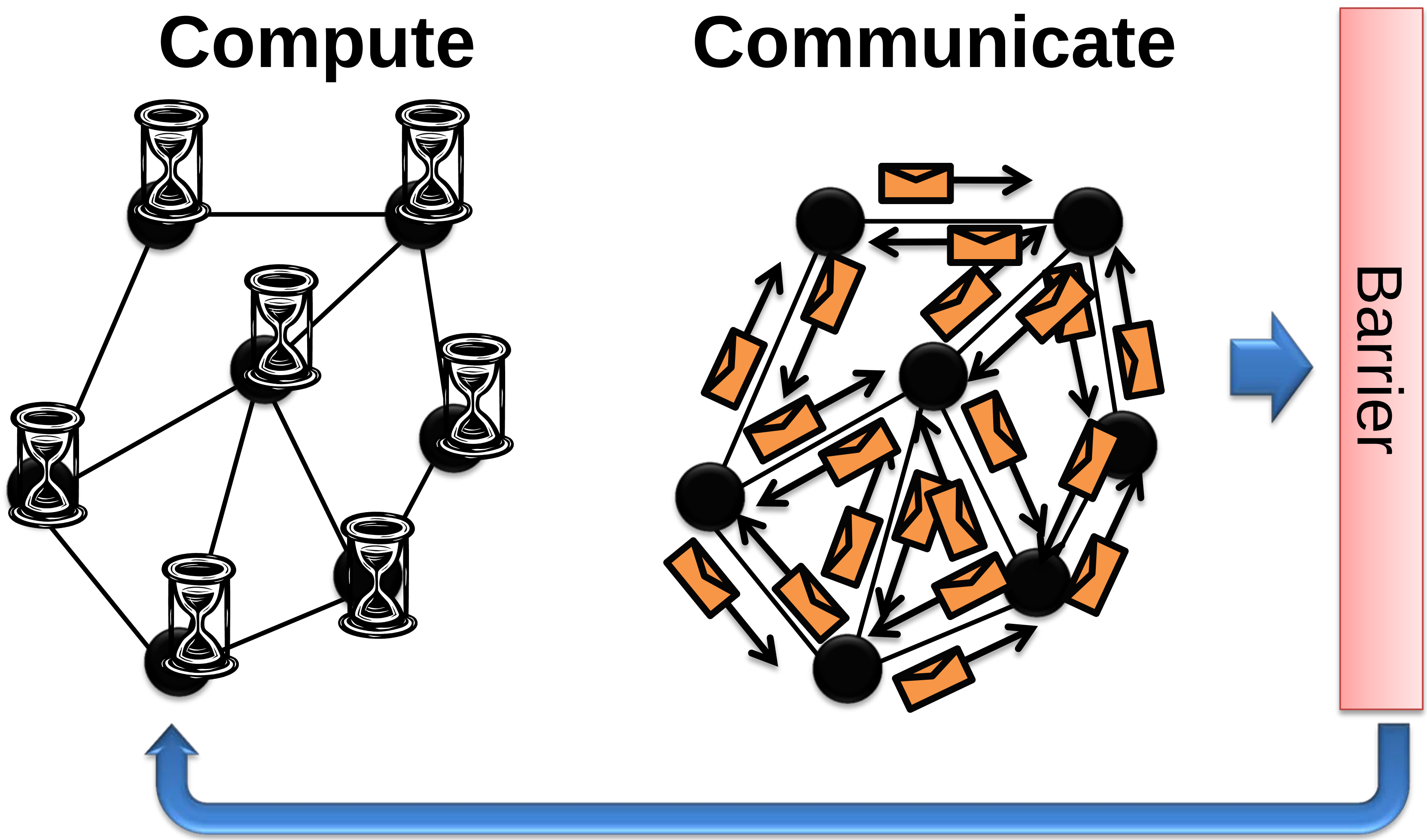
```
Pregel_PageRank(i, messages) :
  // Receive all the messages
  total = 0
  foreach( msg in messages) :
    total = total + msg

  // Update the rank of this vertex
  R[i] = 0.15 + total

  // Send new messages to neighbors
  foreach(j in out_neighbors[i]) :
    Send  msg(R[i]) to vertex j
```



Iterative Bulk Synchronous Execution



Graph-Parallel Systems

Pregel
oogle

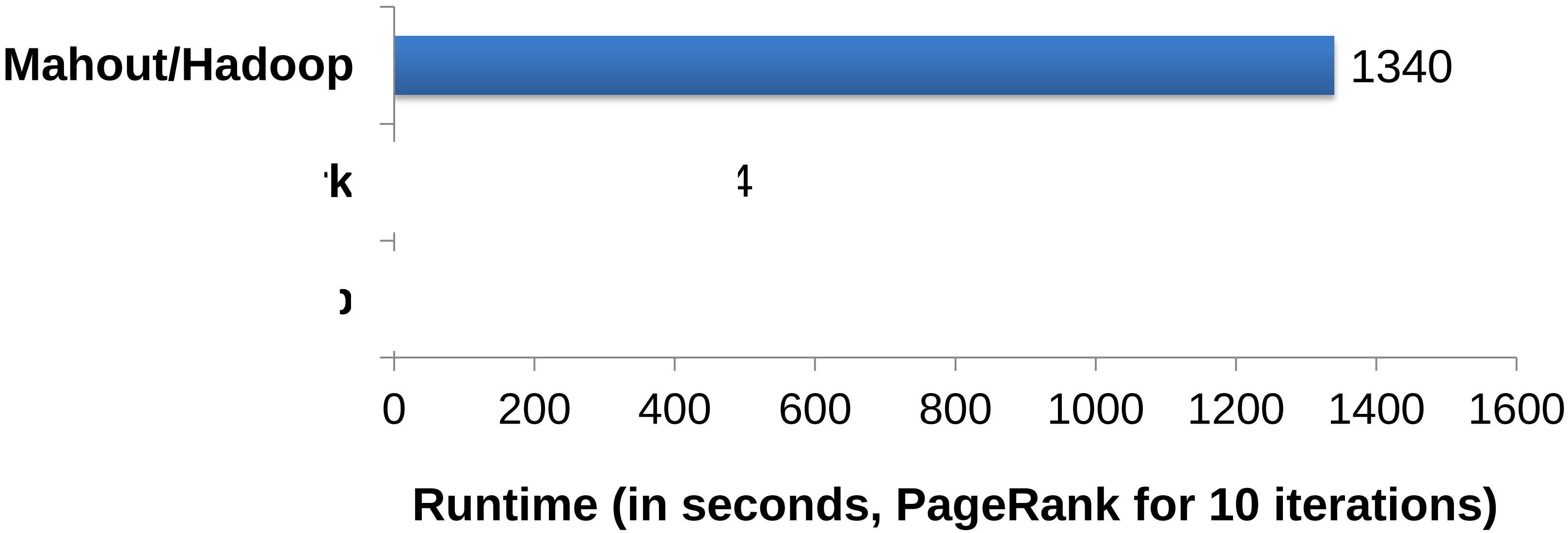
APACHE
GIRAPH

GraphLab

Expose specialized APIs to simplify graph programming.

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

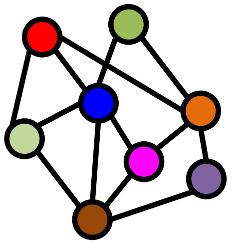
PageRank on the Live-Journal Graph



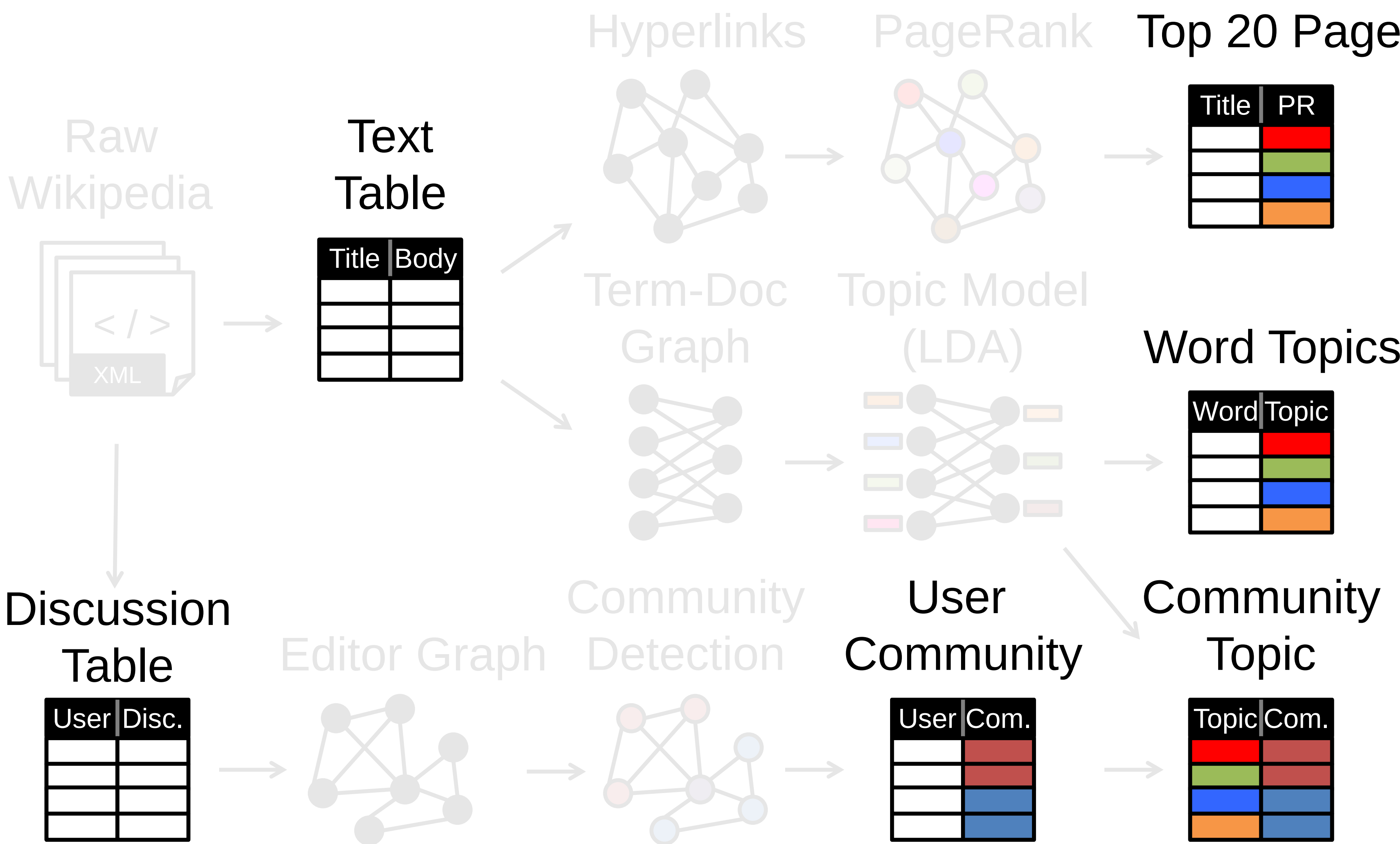
Spark is *4x faster* than Hadoop

GraphLab is *16x faster* than Spark

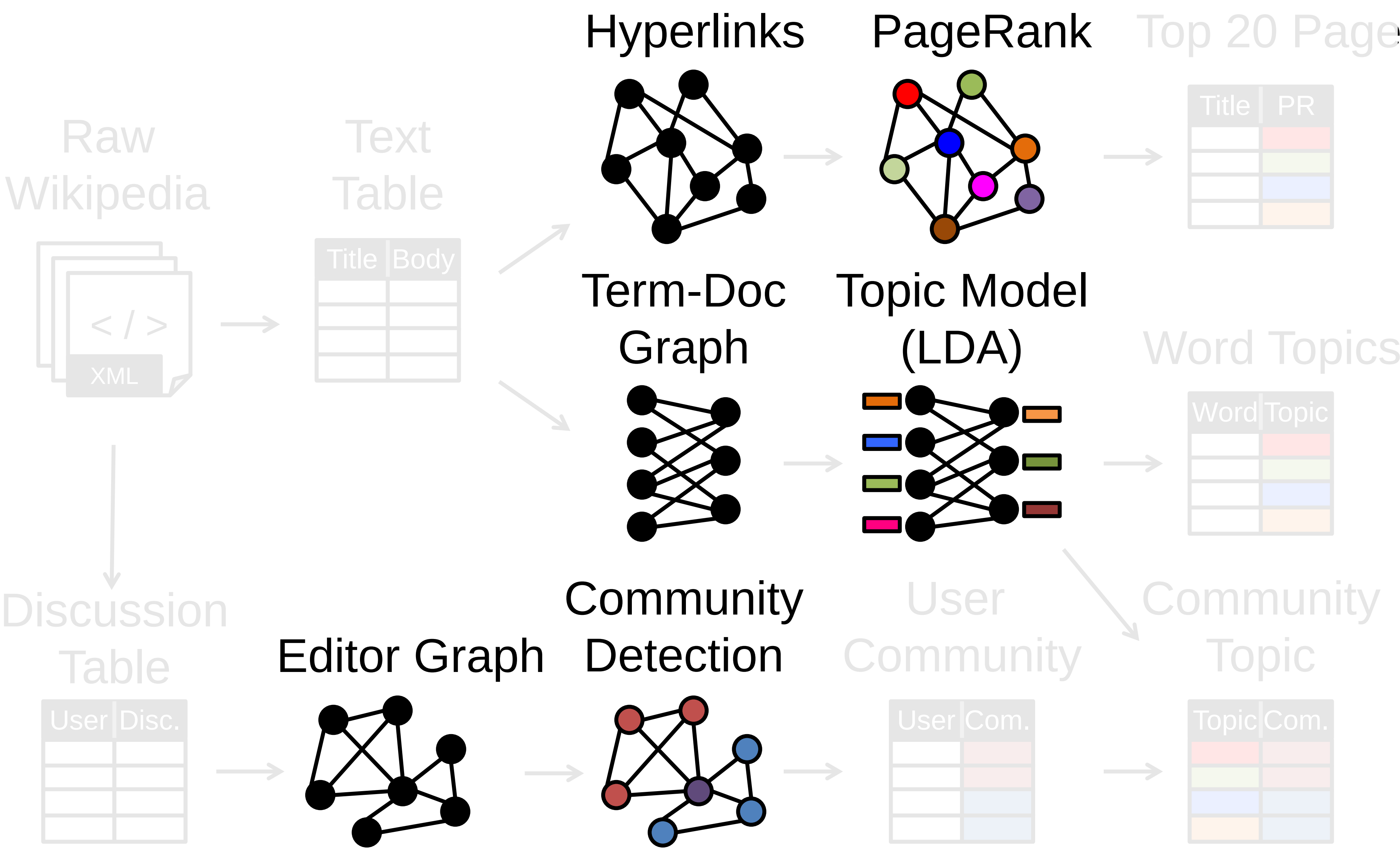
PageRan



Tables


Raw Wikipedia
< / >
XML
Text Table
Title
Body
Hyperlinks
PageRank
Top 20 Page
Title
PR
Term-Doc Graph
Topic Model (LDA)
Word Topics
Word
Topic
Discussion Table
User
Disc.
Editor Graph
Community Detection
User Community
User
Com.
Community Topic
Topic
Com.

Graphs

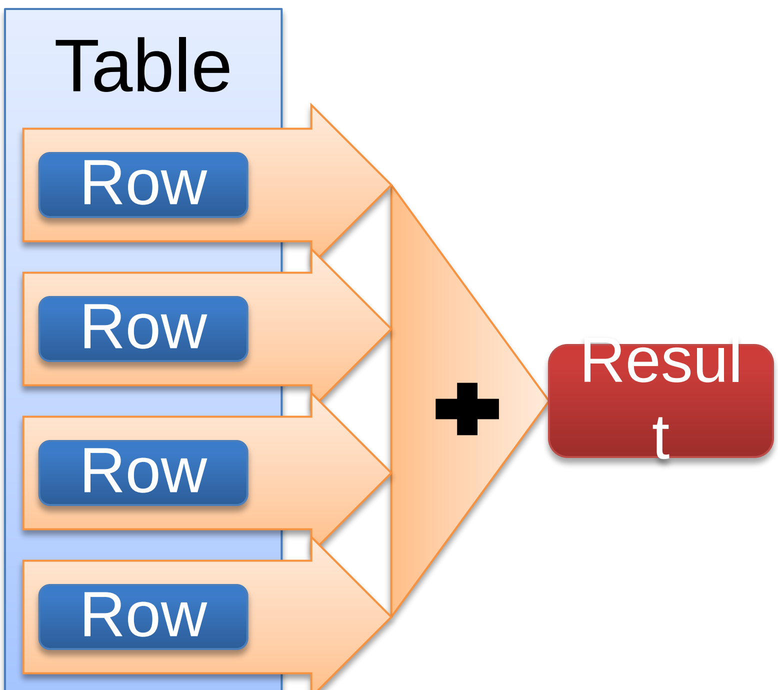


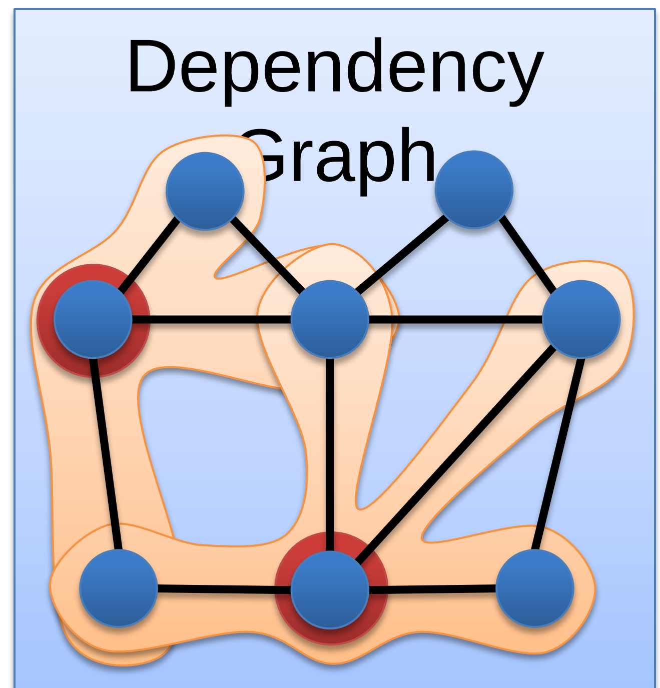
Raw Wikipedia
XML
< / >
Text Table
Title
Body
Hyperlinks
PageRank
Top 20 Page
Title
PR
Term-Doc Graph
Topic Model (LDA)
Word Topics
Word
Topic
Discussion Table
User
Disc.
Editor Graph
Community Detection
User Community
User
Com.
Community Topic
Topic
Com.

Separate Systems to Support Each View

Table View

Graph View





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

Difficult to Program and Use

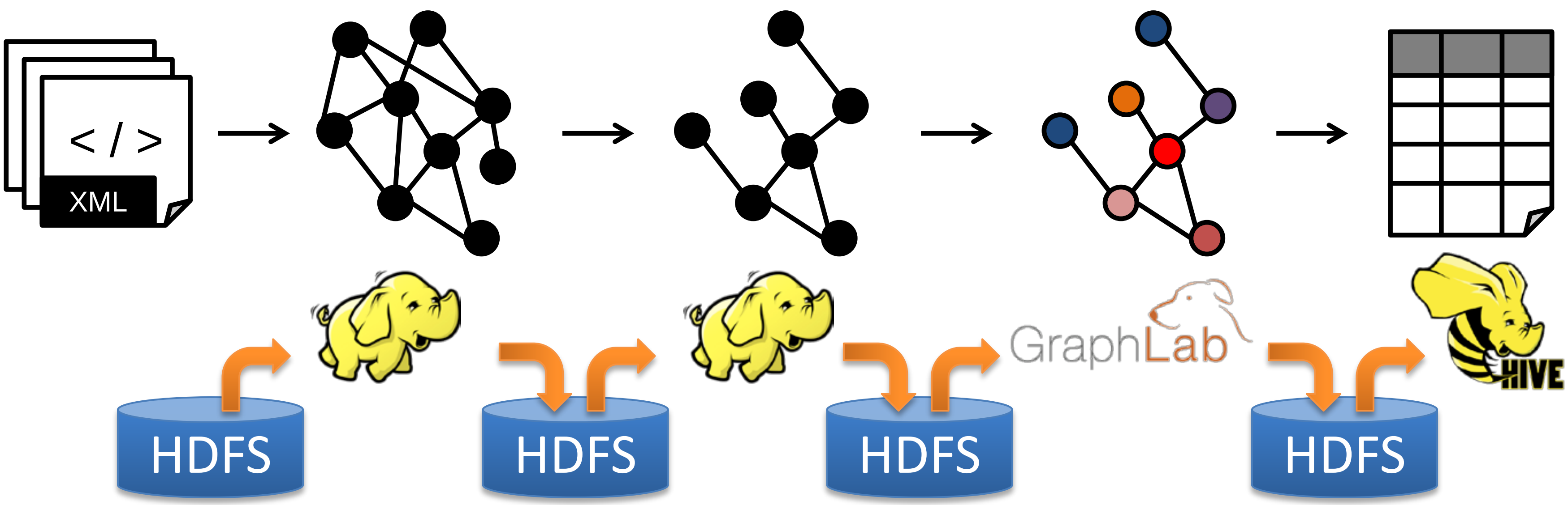
Users must ***Learn***, ***Deploy***, and ***Manage*** multiple systems

HIVE Spark APACHE GIRAPH GraphLab

Leads to brittle and often complex interfaces

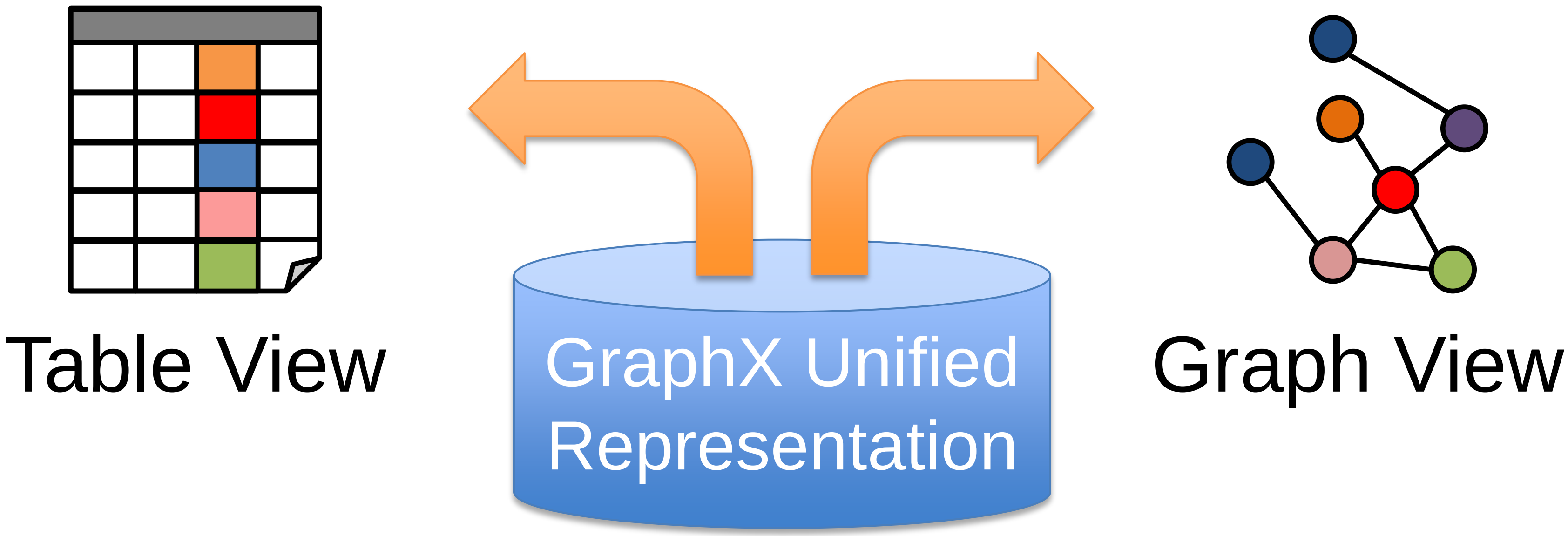
Inefficient

Extensive data movement and duplication across the network and file system



Limited reuse internal data-structures across stages

GraphX Solution: Tables and Graphs are *views* of the *same physical* data



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

Graphs → Relational Algebra

1. Encode graphs as distributed tables

2. Express graph computation in relational algebra

3. Recast graph systems optimizations as:
    1. Distributed join optimization
    2. Incremental materialized maintenance

Integrate Graph and Table data processing systems.

Achieve performance parity with specialized systems.

Distributed Graphs as Distributed Tables

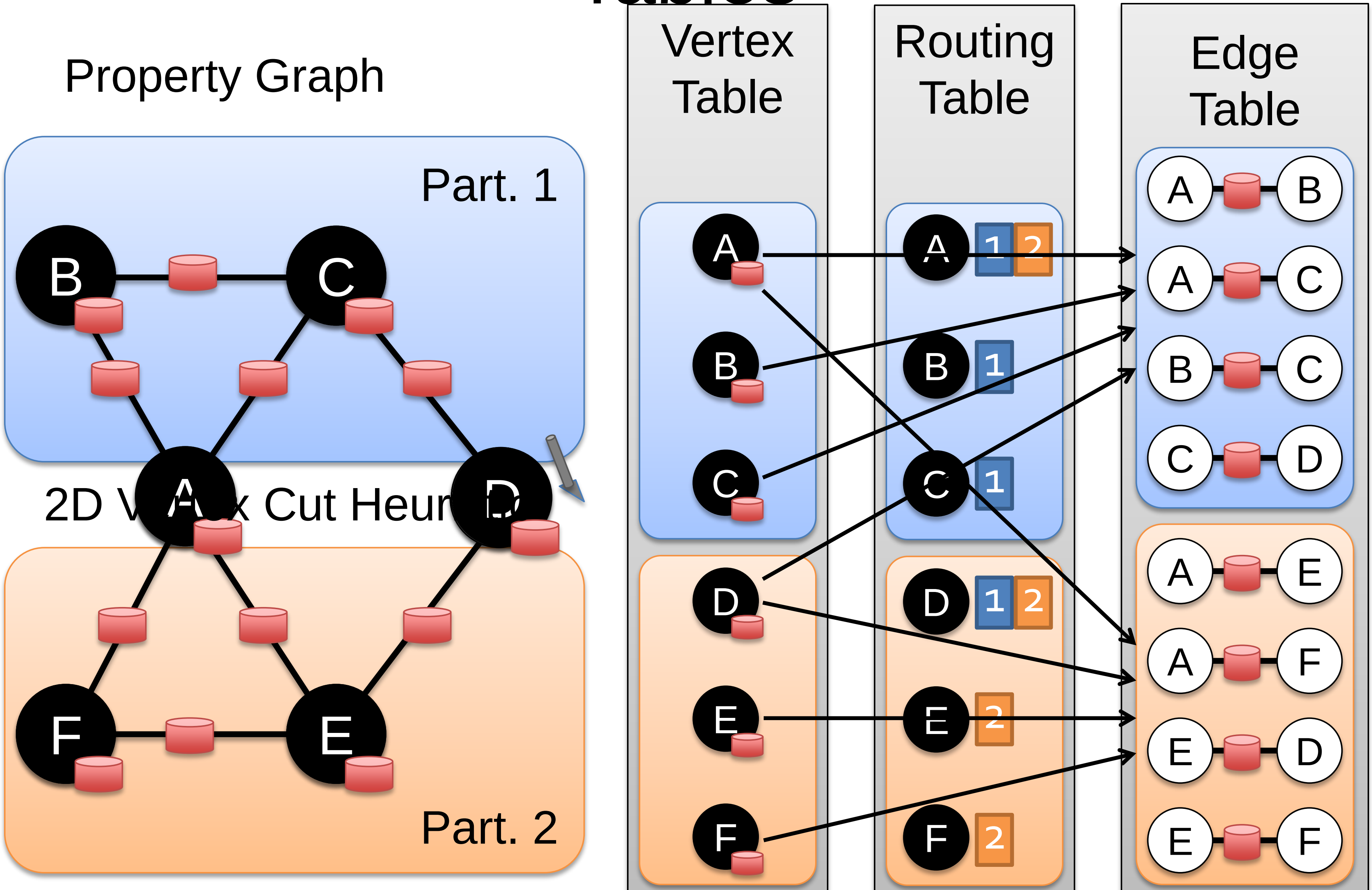


Table Operators

Table 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`	`...`

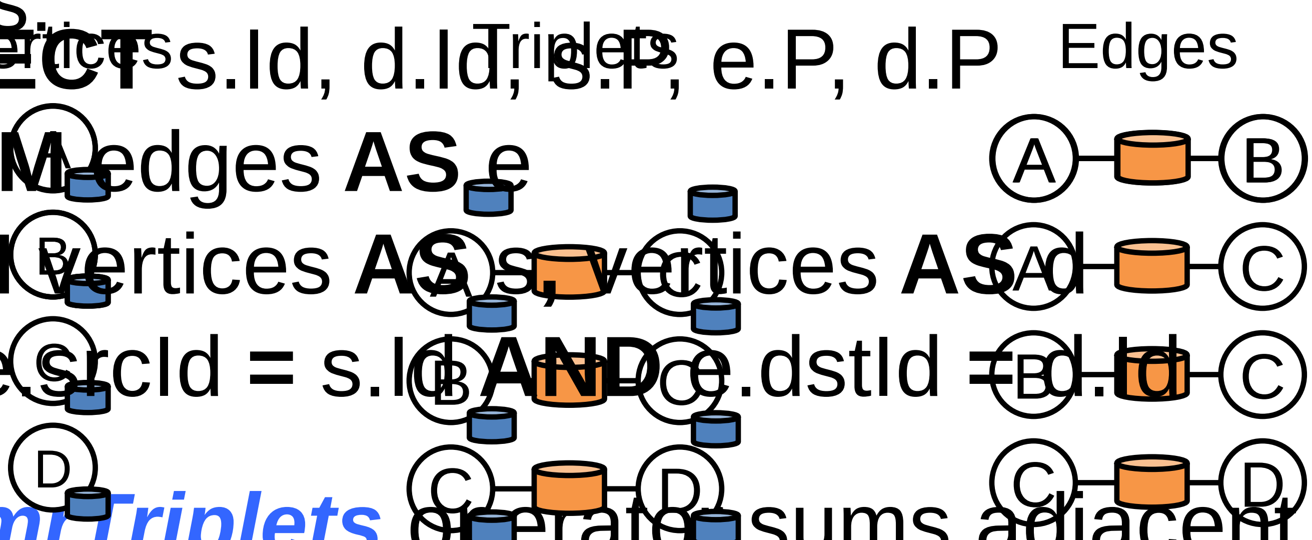
Graph Operators

```
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:

SELECT s.Id, d.Id, s.P, e.P, d.P
FROM edges **AS** e
JOIN vertices **AS** s, vertices **AS** d
ON e.srcId = s.Id **AND** e.dstId = d.Id



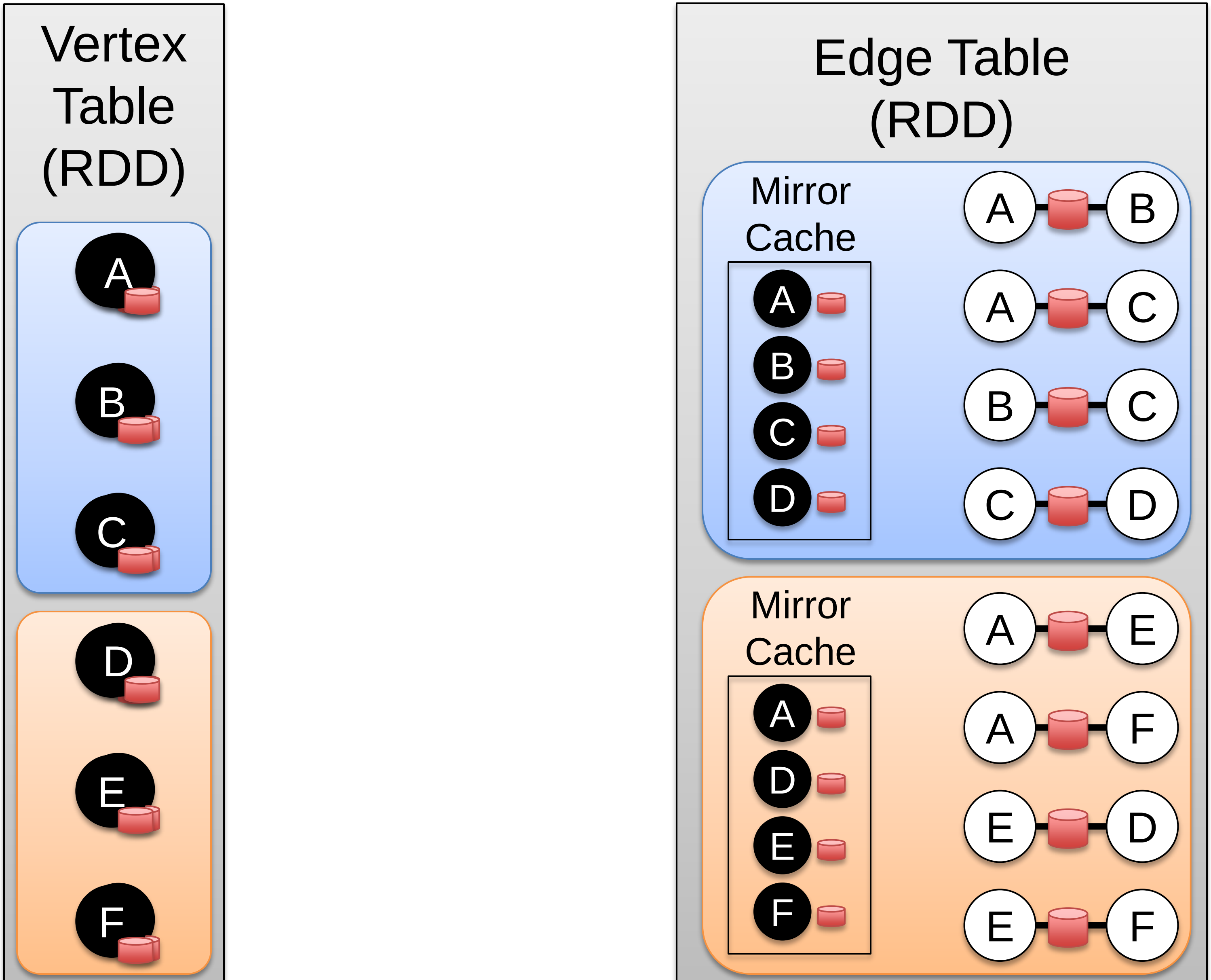
The ***mrTriplets*** operator sums adjacent triplets.

SELECT t.dstId, *reduce*(*map*(t)) **AS** sum
FROM triplets **AS** t **GROUPBY** t.dstId

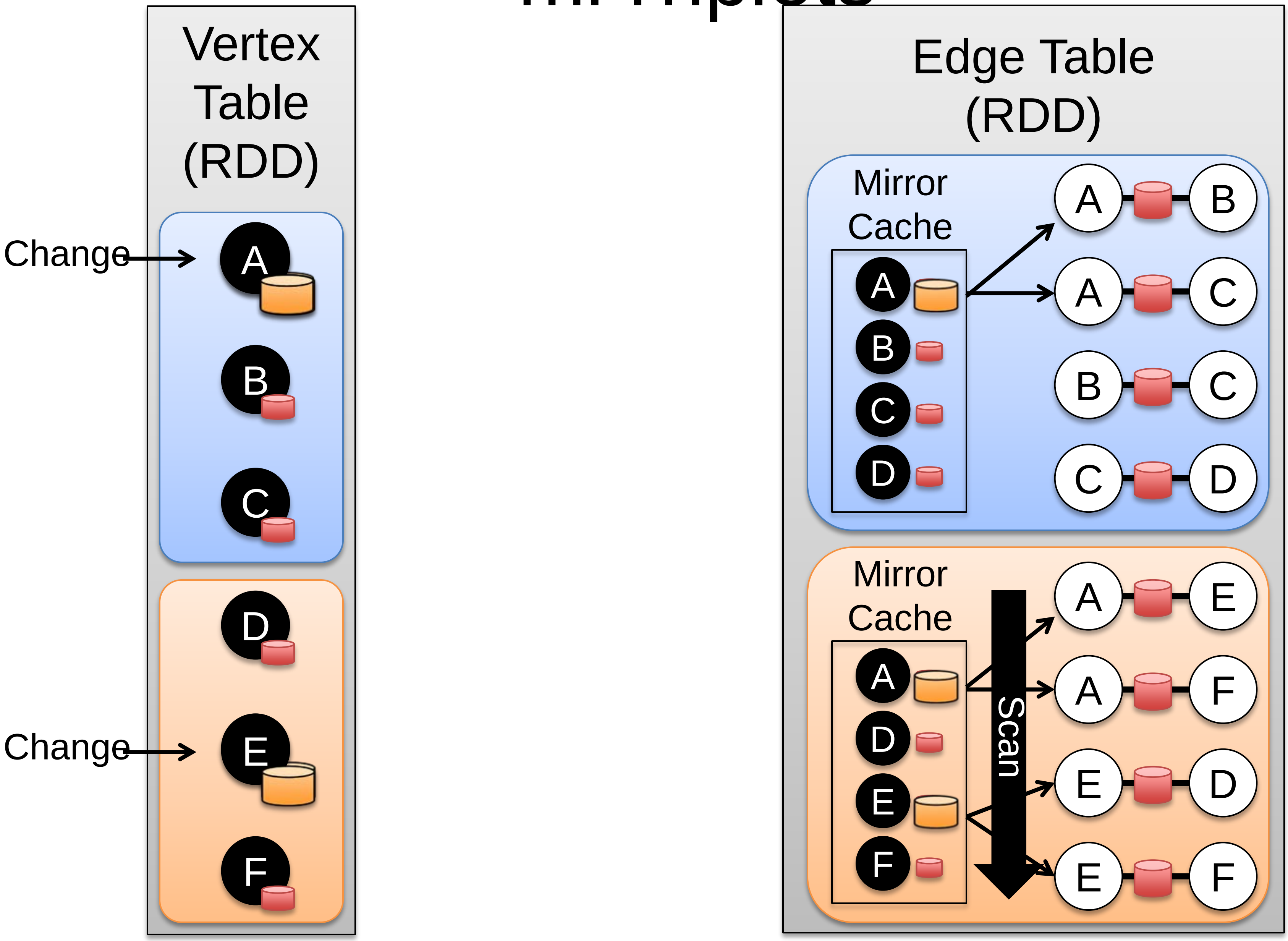
We express *enhanced* Pregel and GraphLab
abstractions using the GraphX operators
in less than 50 lines of code!

SYSTEM DESIGN

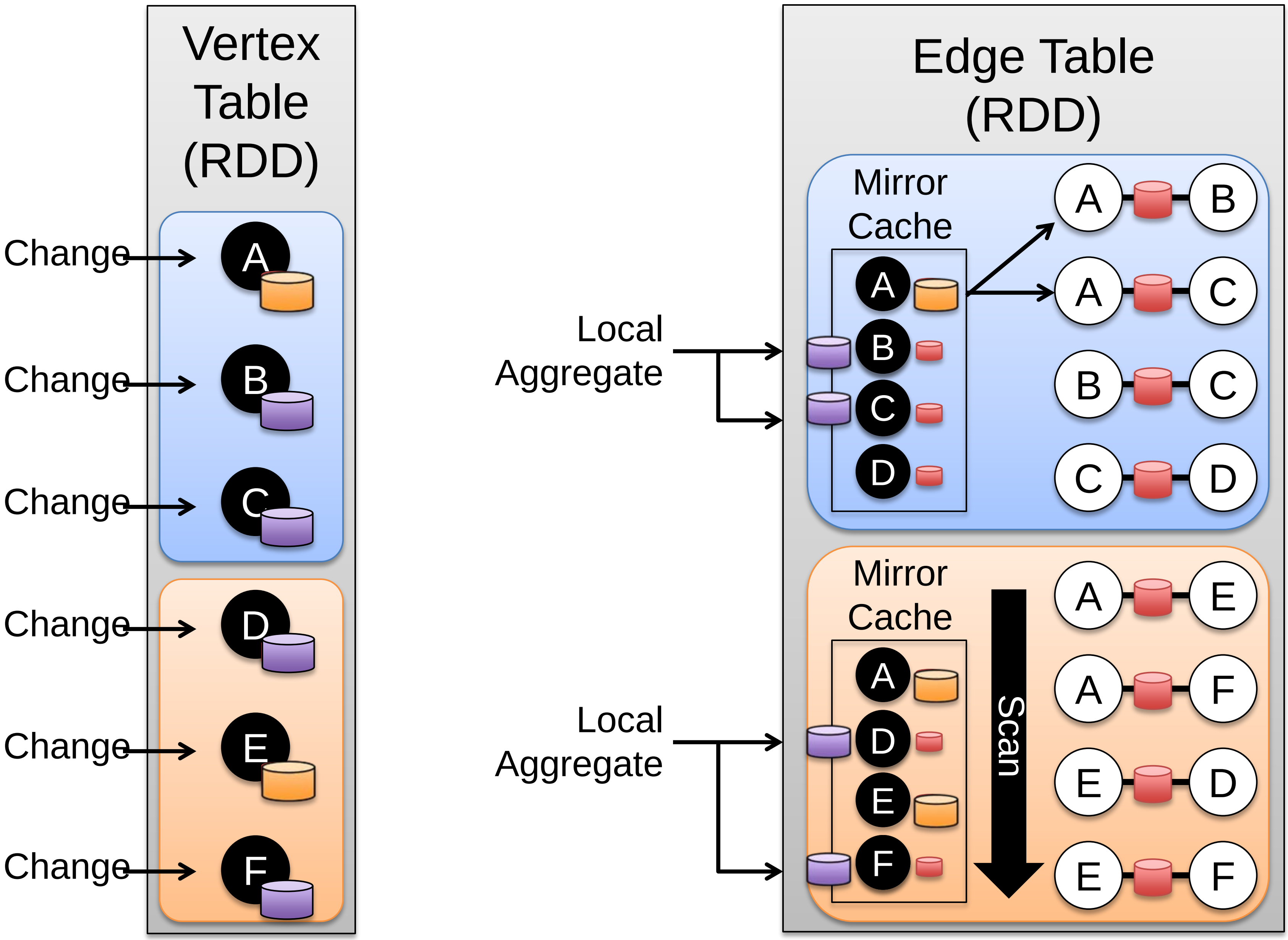
Caching for Iterative mrTriplets



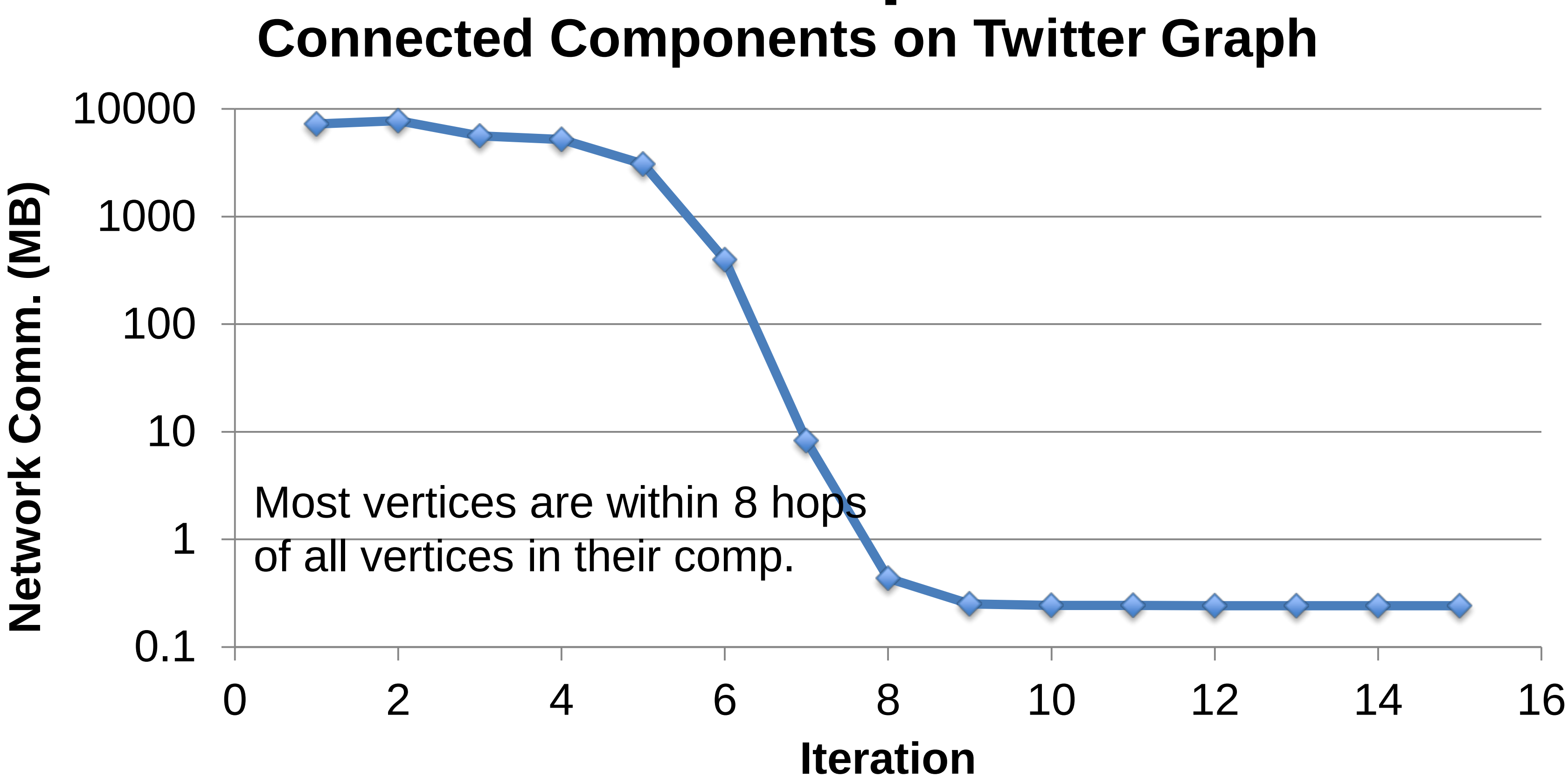
Incremental Updates for Iterative mrTriplets



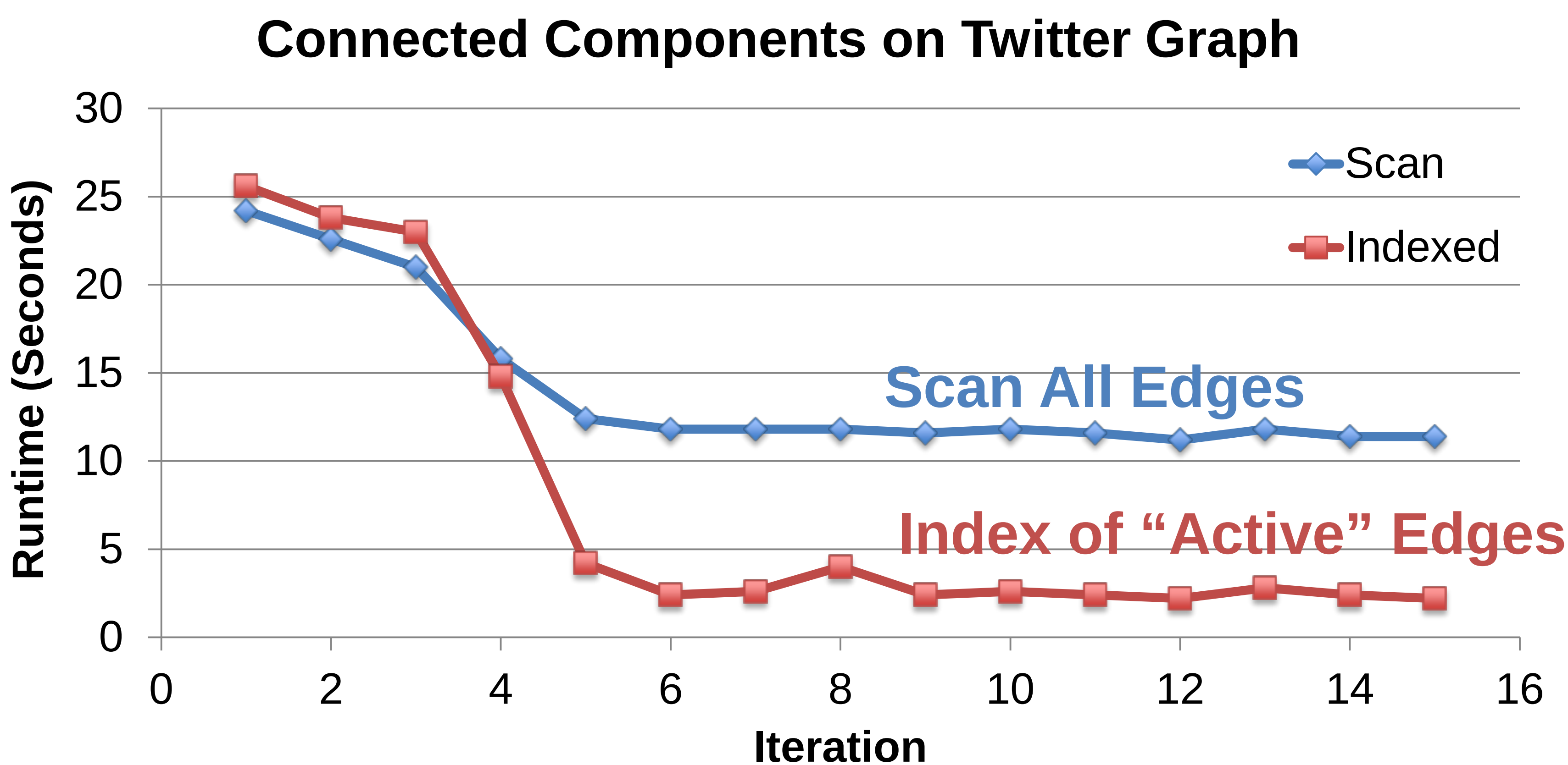
Aggregation for Iterative mrTriplets



Reduction in Communication Due to Cached Updates



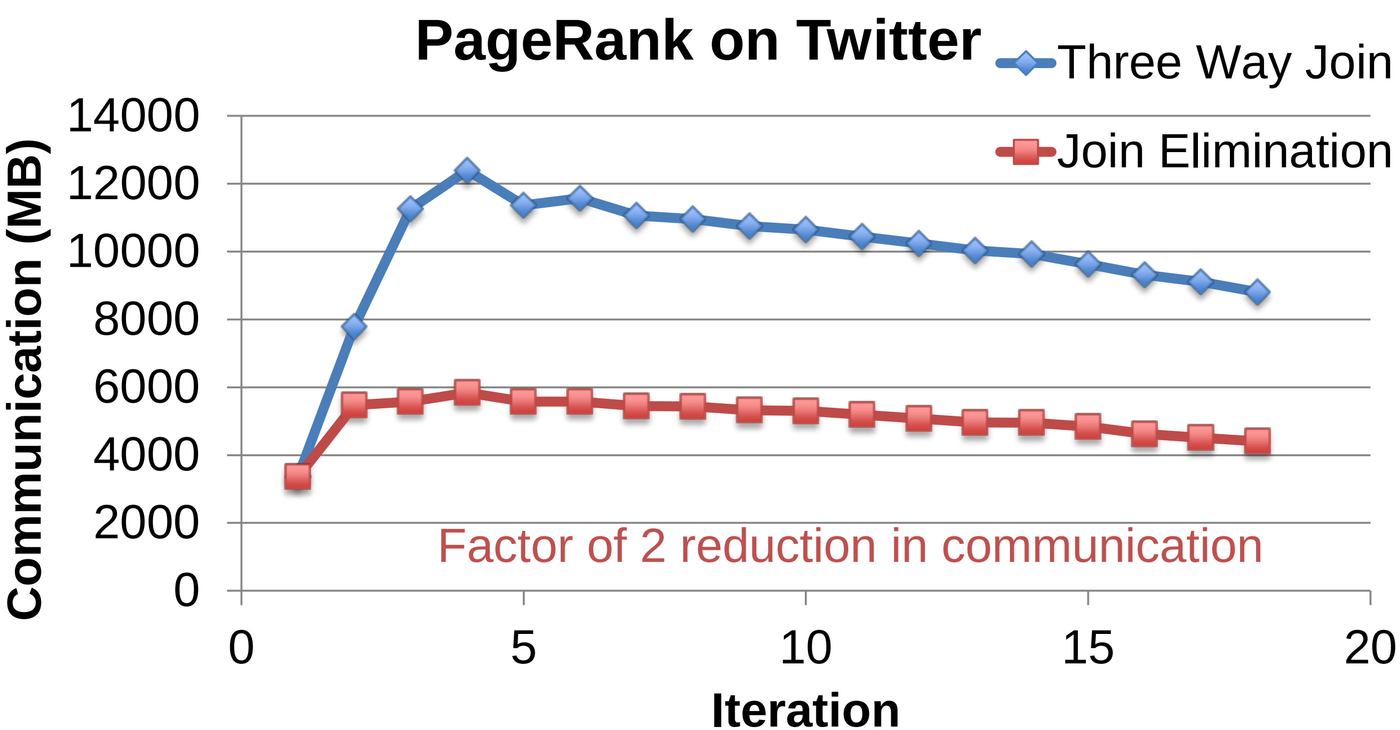
Benefit of Indexing *Active* Edges



Join Elimination

Identify and bypass joins for unused triplet fields

sendMsg(i→j, R[i], R[j], E[i,j]):
  // Compute single message
  return msg(R[i]/E[i,j])



Additional Query Optimizations

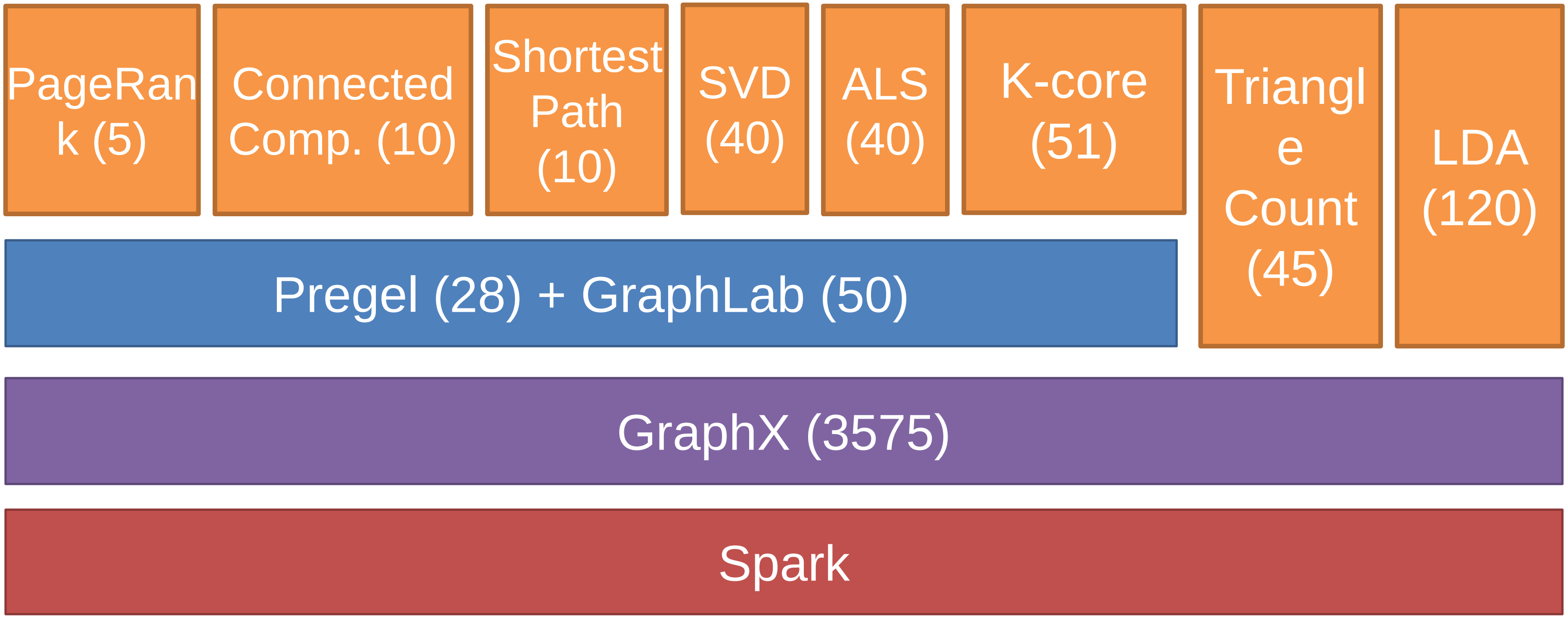
Indexing and Bitmaps:

- » To accelerate joins across graphs
- » To efficiently construct sub-graphs

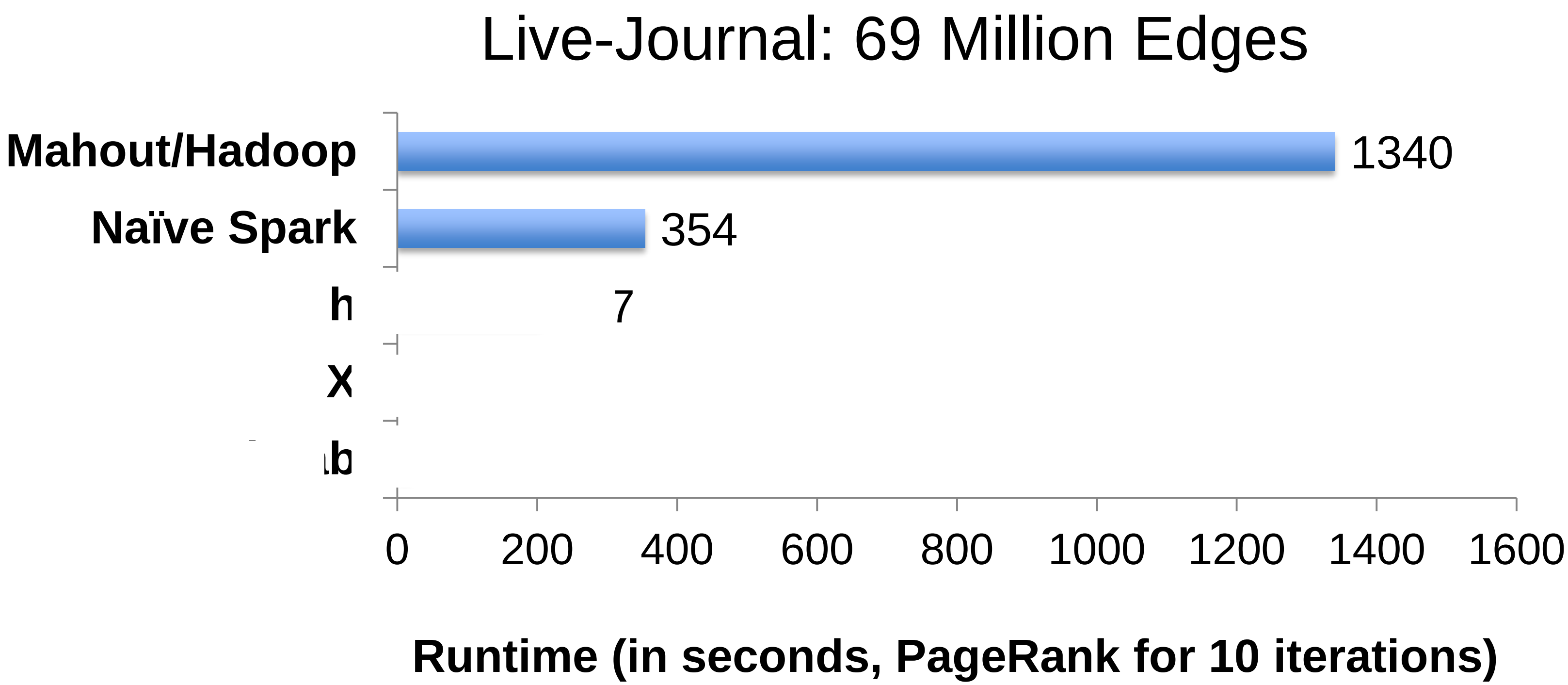
Substantial Index and Data Reuse:

- » Reuse routing tables across graphs and sub-graphs
- » Reuse edge adjacency information and indices

The GraphX Stack (Lines of Code)



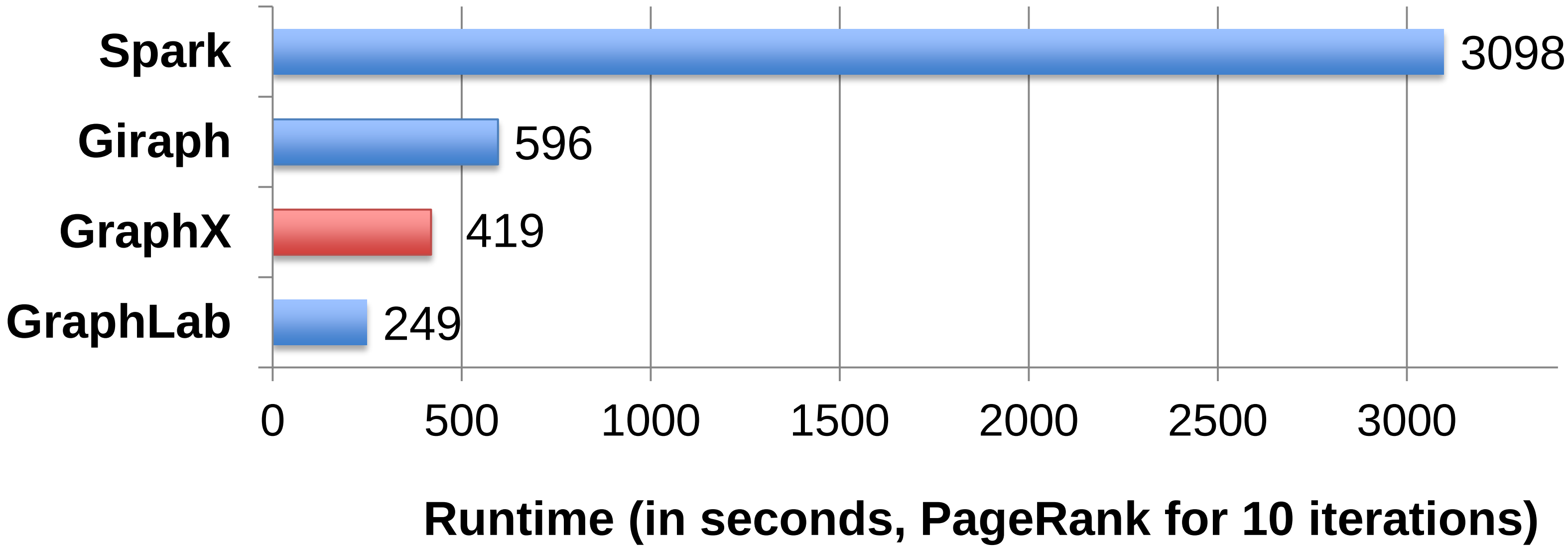
Performance Comparisons



GraphX is roughly 3x slower than GraphLab

GraphX scales to larger graphs

Twitter Graph: 1.5 Billion Edges



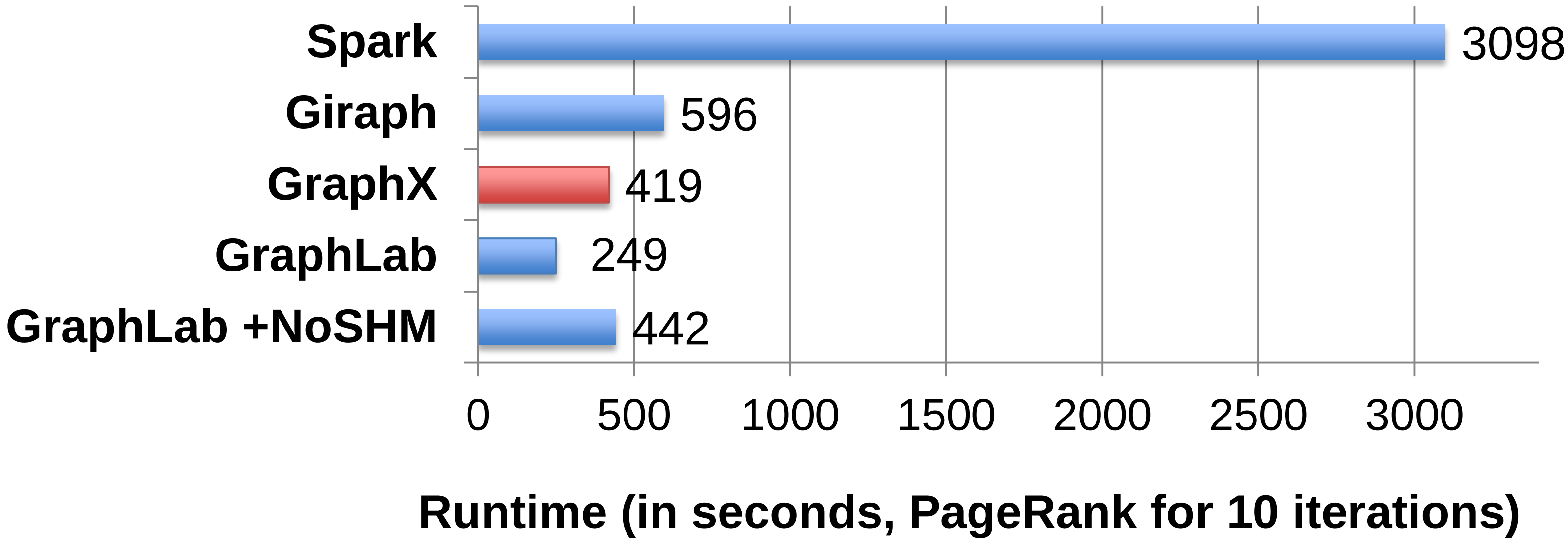
GraphX is roughly 2x slower than GraphLab

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

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

GraphX scales to larger graphs

Twitter Graph: 1.5 Billion Edges



GraphX is roughly 2x slower than GraphLab

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

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

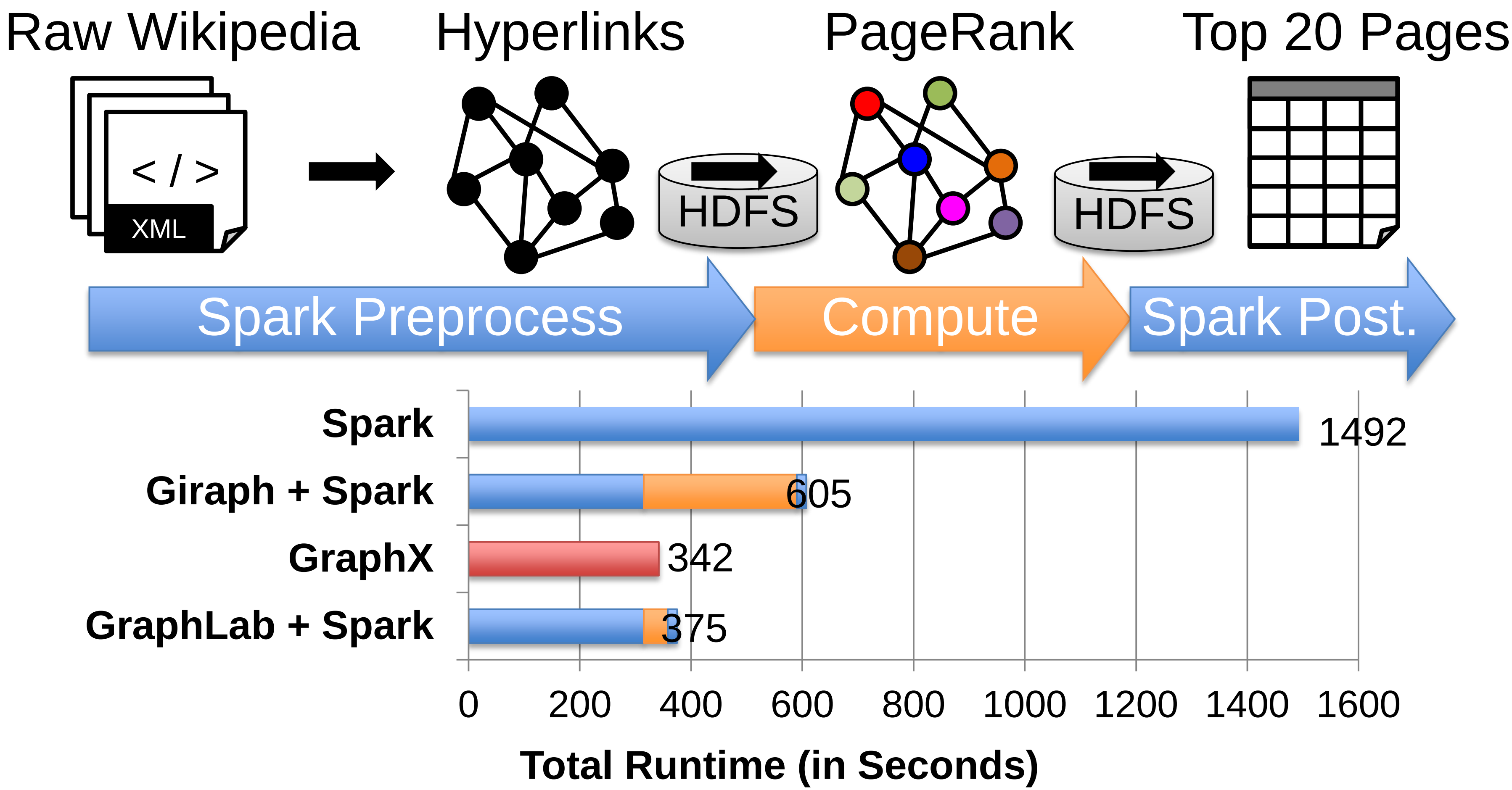
PageRank is just one stage….

What about a pipeline?

Example Analytics Pipeline

```
// Load raw data tables
val articles = sc.textFile("hdfs://wiki.xml").map(parserV)
val links = articles.flatMap(xmlLinkParser)
// Build the graph from tables
val graph = new Graph(articles, links)
// Run PageRank Algorithm
val pr = graph.PageRank(tol = 1.0e-5)
// Extract and print the top 20 articles
val topArticles = articles.join(pr).top(20).collect
for ((article, pageRank) <- topArticles) {
  println(article.title + '\t' + pageRank)
}
```

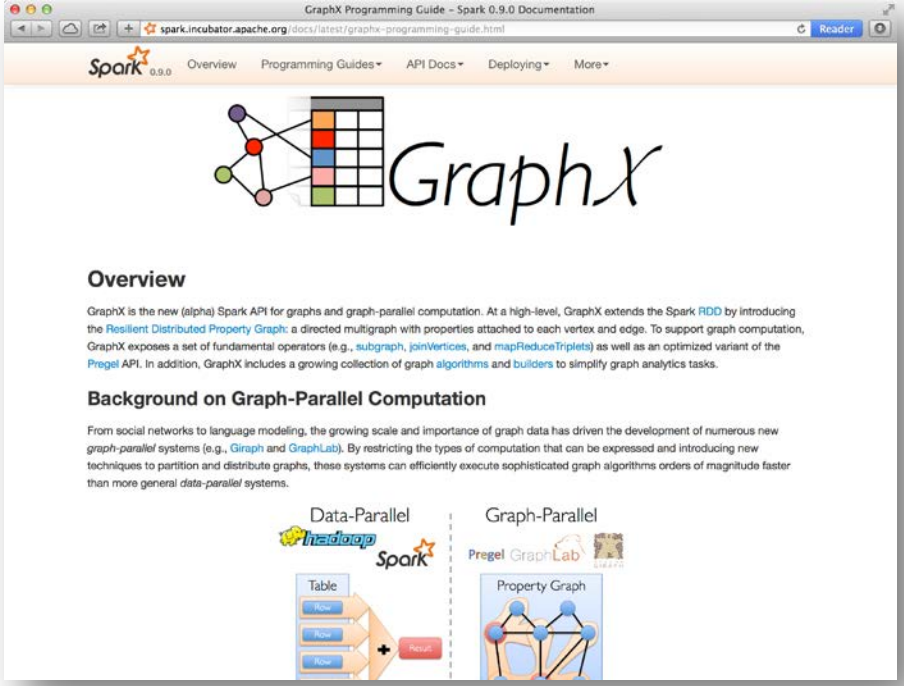
A Small Pipeline in GraphX



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

Status

Part of Apache Spark

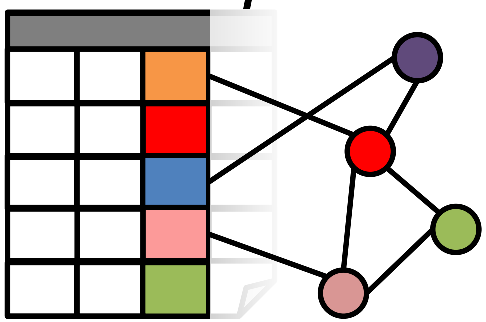


In production at Alibaba Taobao

GraphX: Unified Analytics

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

Thanks You

http://amplab.cs.berkeley.edu/projects/gra
phx/

jegonzal@eecs.berkeley.edu

Reynold Xin

Ankur Dave

Daniel Crankshaw

Michael Franklin

Ion Stoica