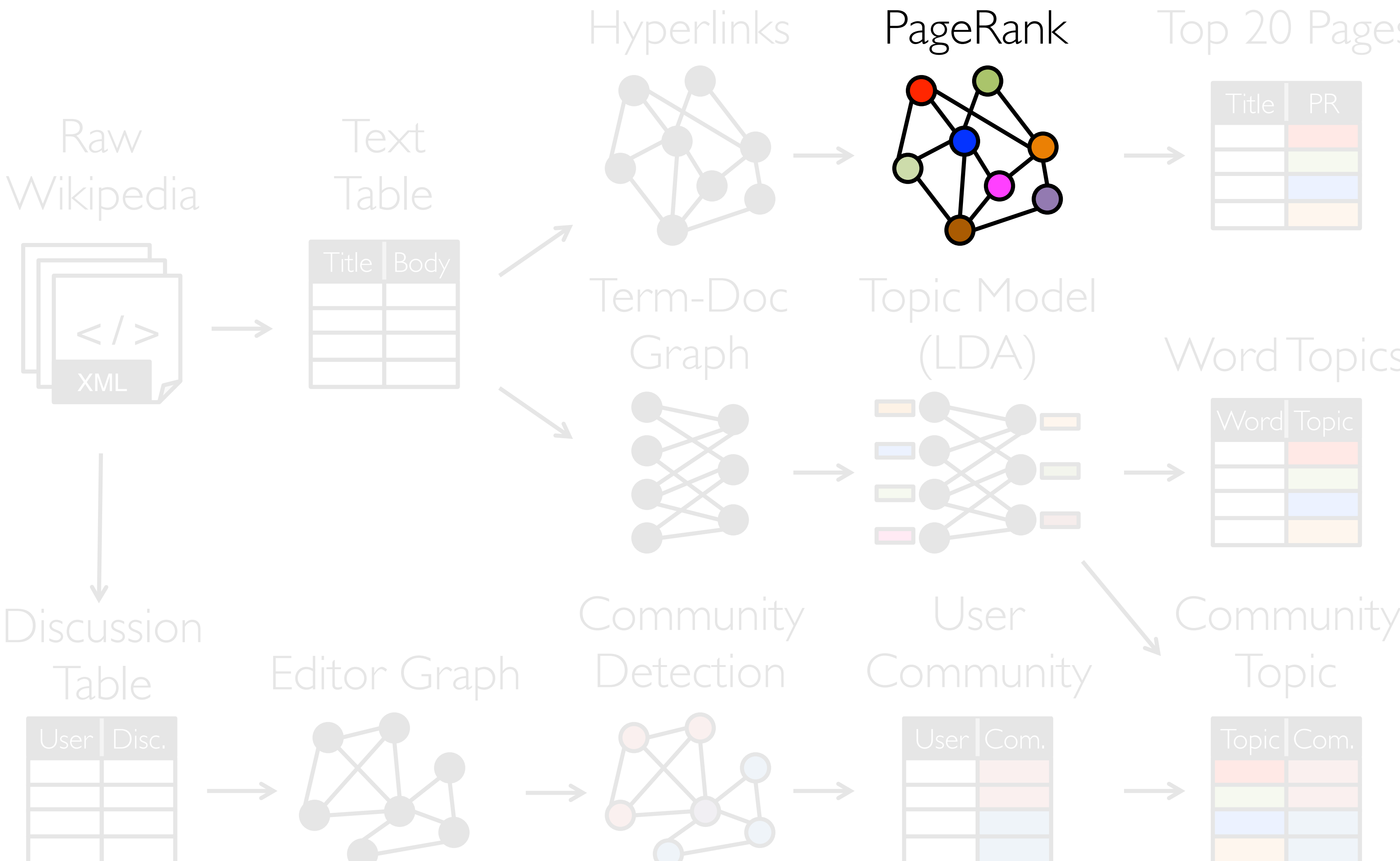
From *Graphs* to *Tables*: The Design of Scalable Systems for *Graph Analytics*

Joseph E. Gonzalez
Post-doc, UC Berkeley AMPLab
jegonzal@eecs.berkeley.edu
Co-founder, GraphLab Inc.
joseph@graphlab.com

WWW'14 Workshop on Big Graph Mining

*These slides are best viewed in PowerPoint with animation.

Graphs are Central to Analytics



PageRank: Identifying Leaders

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

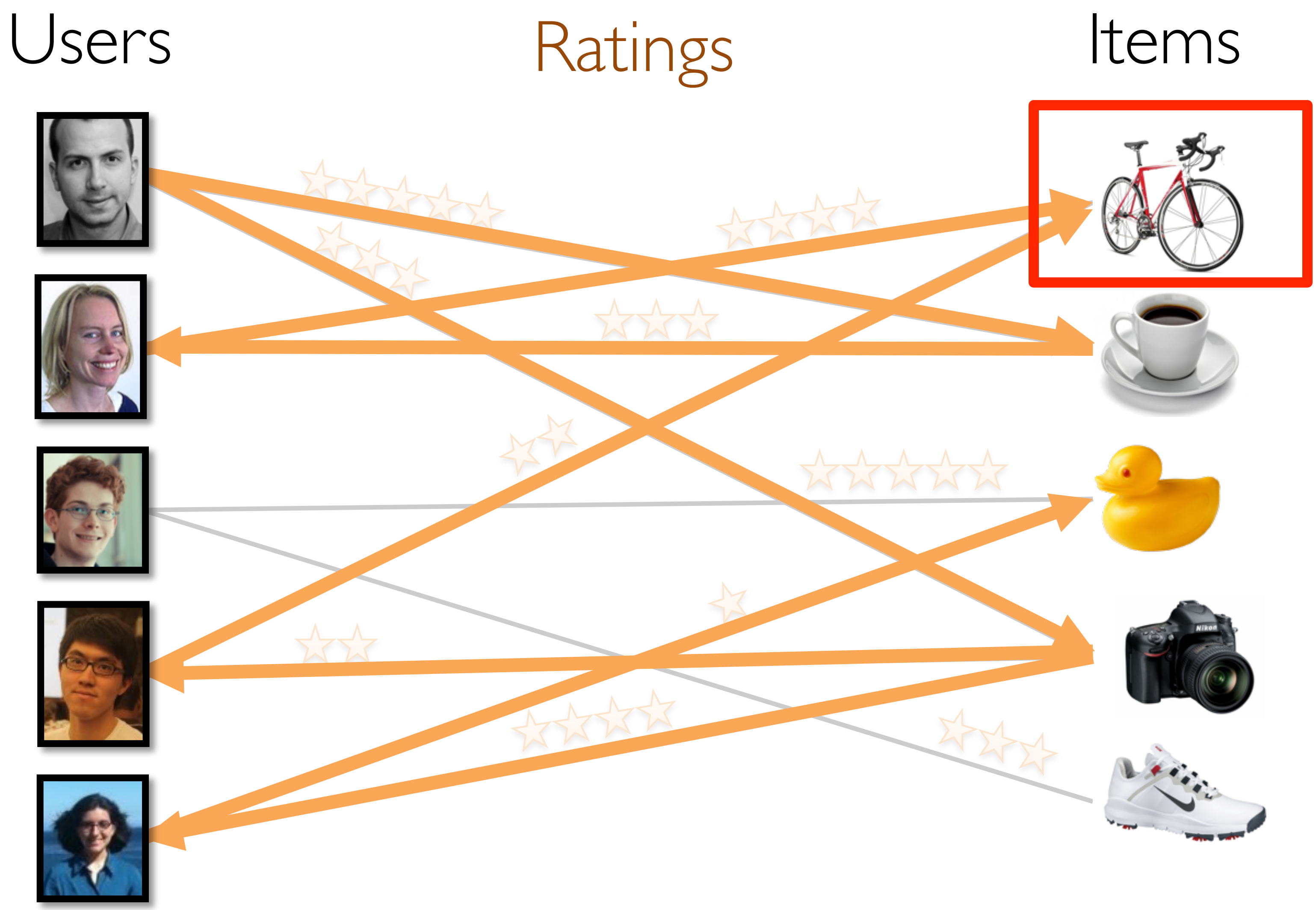
Rank of user *i*

Sum of neighbors

Update ranks in parallel

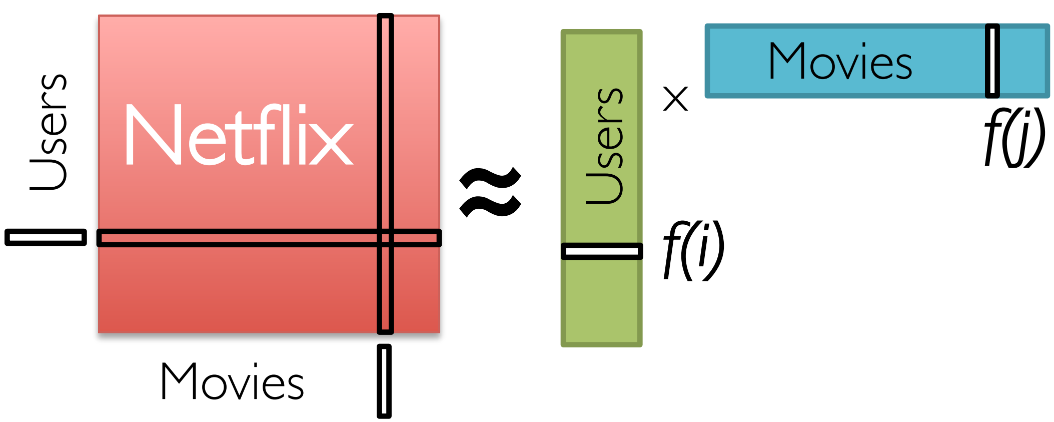
Iterate until convergence

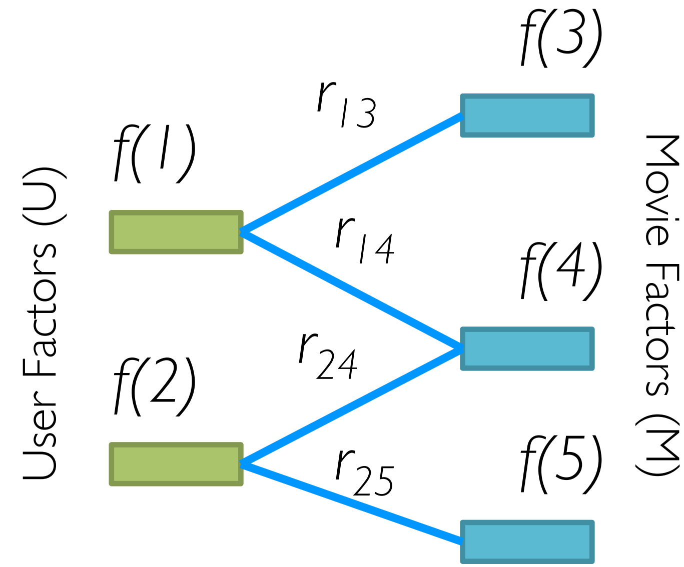
Recommending Products



Recommending Products

Low-Rank Matrix Factorization:

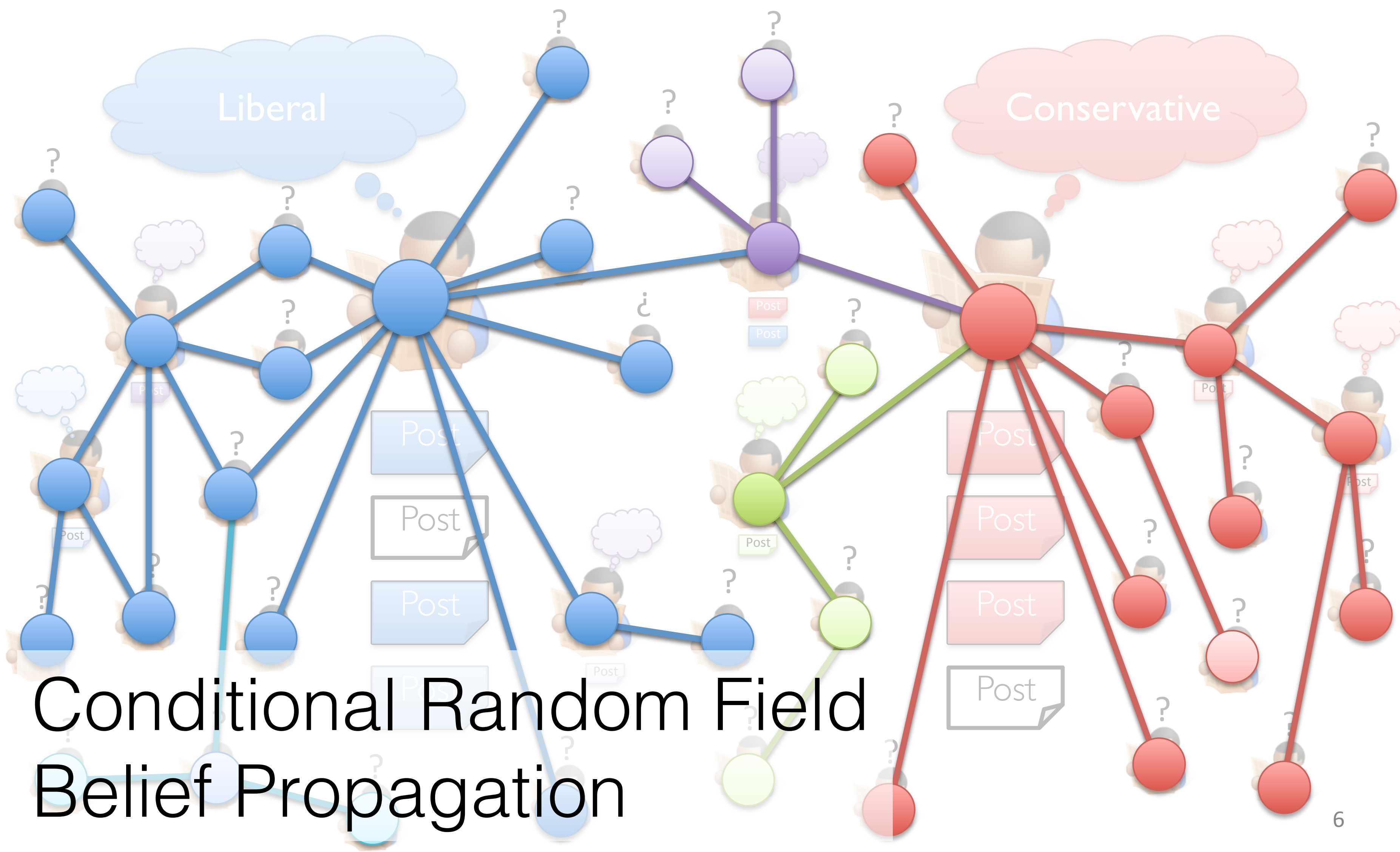




Iterate:

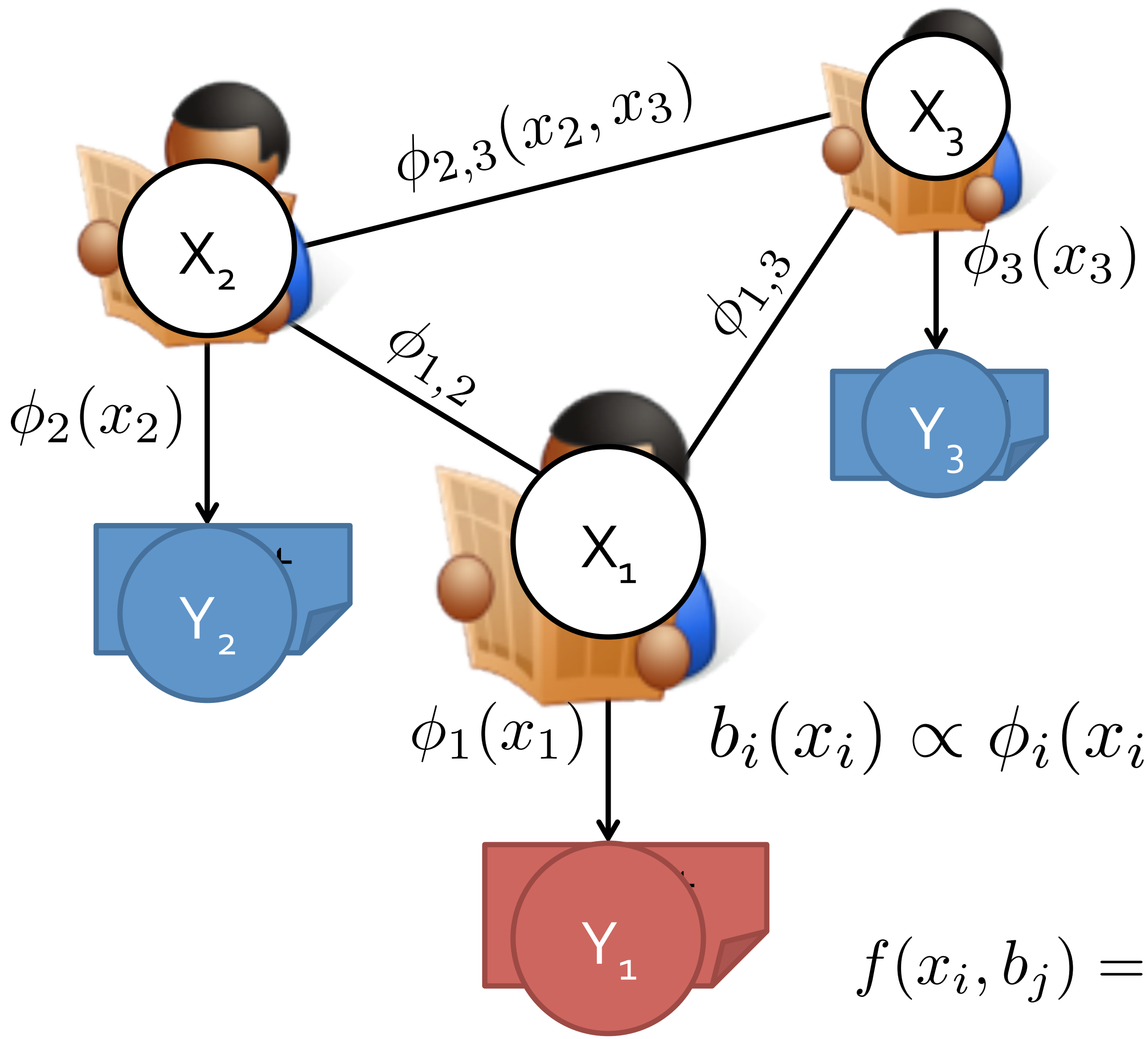
$$f[i] = \arg\min_{w \in \mathbb{R}^d} \sum_{j \in \mathrm{Nbrs}(i)} \left(r_{ij} - w^T f[j]\right)^2 + \lambda ||w||_2^2$$

Predicting User Behavior



Conditional Random Field
Belief Propagation

Mean Field Algorithm

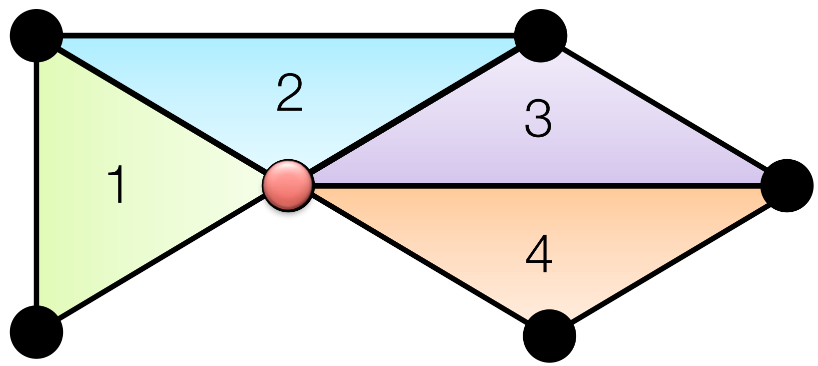


$$b_i(x_i) \propto \phi_i(x_i) \exp \left(\overbrace{\sum_{j \in N_i} f(x_i, b_j)}^{\text{Sum over Neighbors}} \right)$$

$$f(x_i, b_j) = \sum_{x_j} b_j(x_j) \log \phi_{i,j}(x_i, x_j)$$

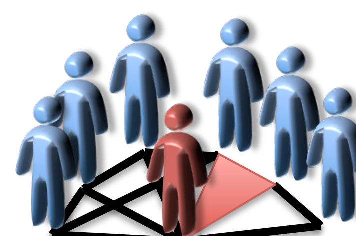
Finding Communities

Count triangles passing through each vertex:



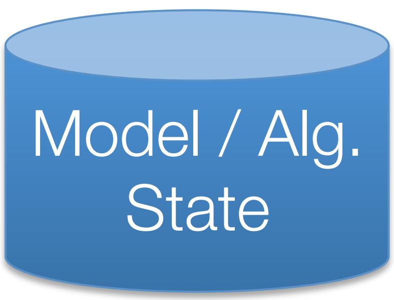
Measures "cohesiveness" of local community

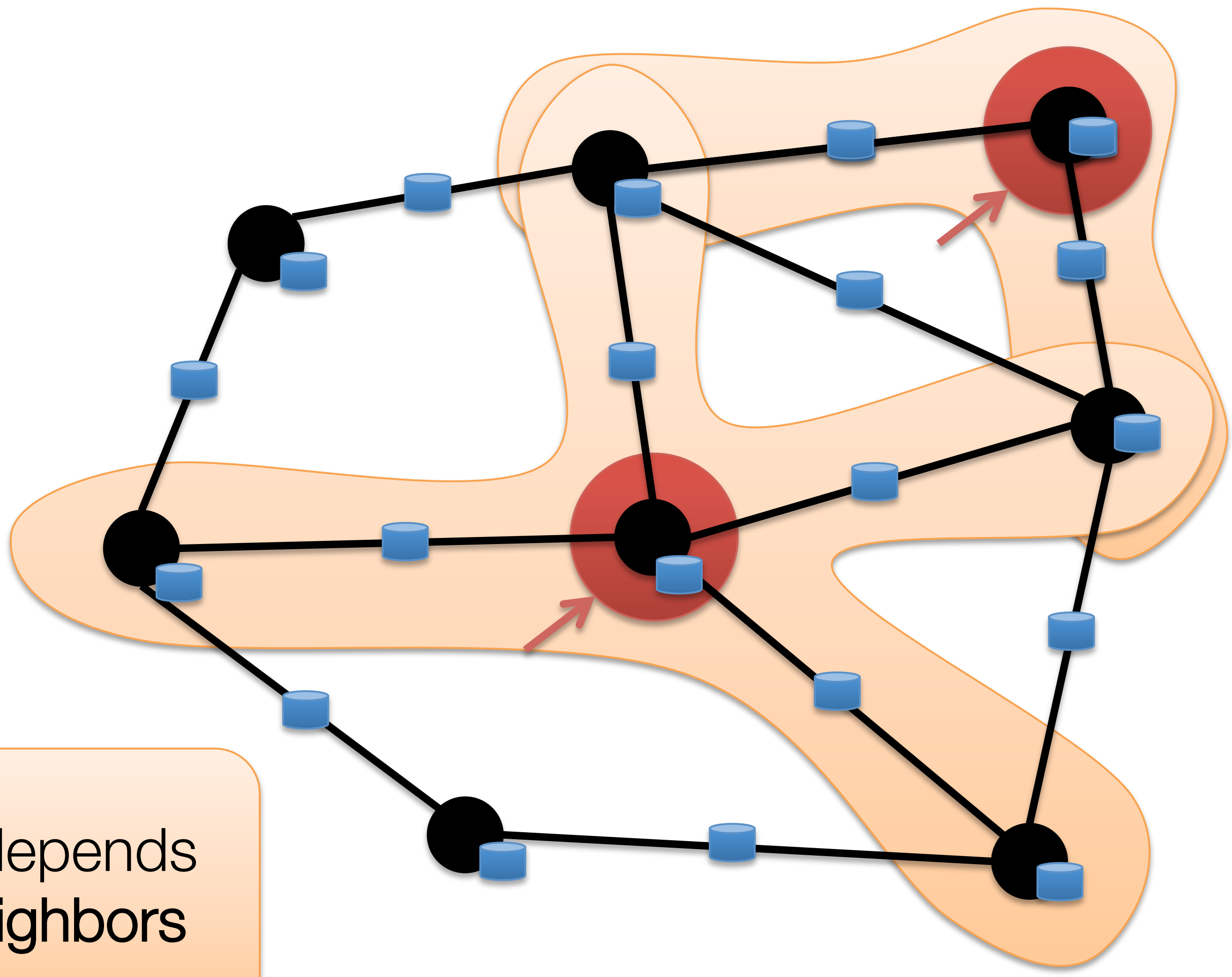
Fewer Triangles
Weaker Community



More Triangles
Stronger Community

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

APACHE GIRAPH

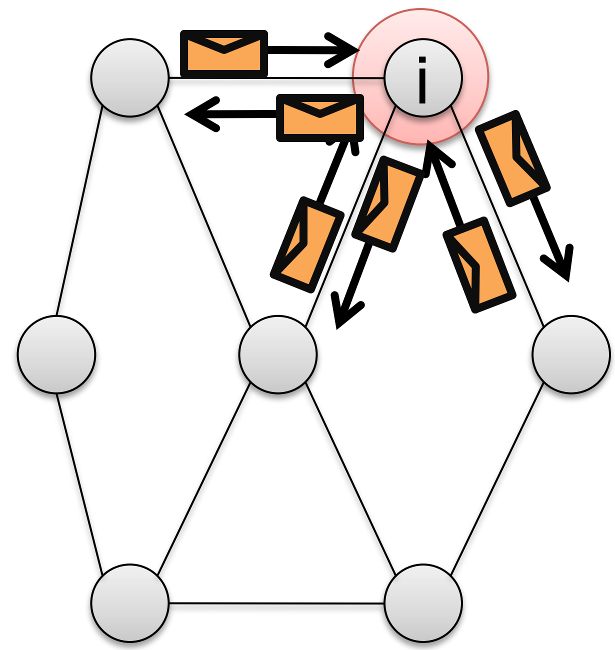
GraphLab

Expose specialized APIs to simplify graph programming.

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
```

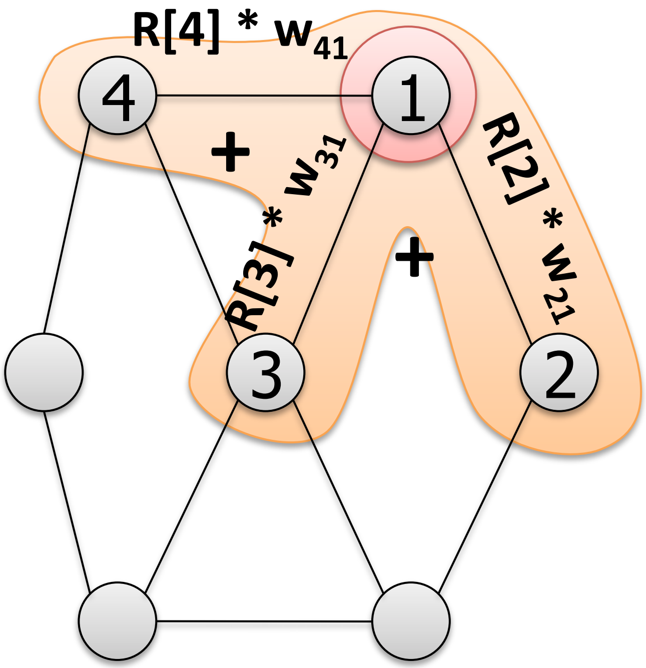


Malewicz et al. [PODC'09, SIGMOD'10]

The GraphLab (Pull) Abstraction

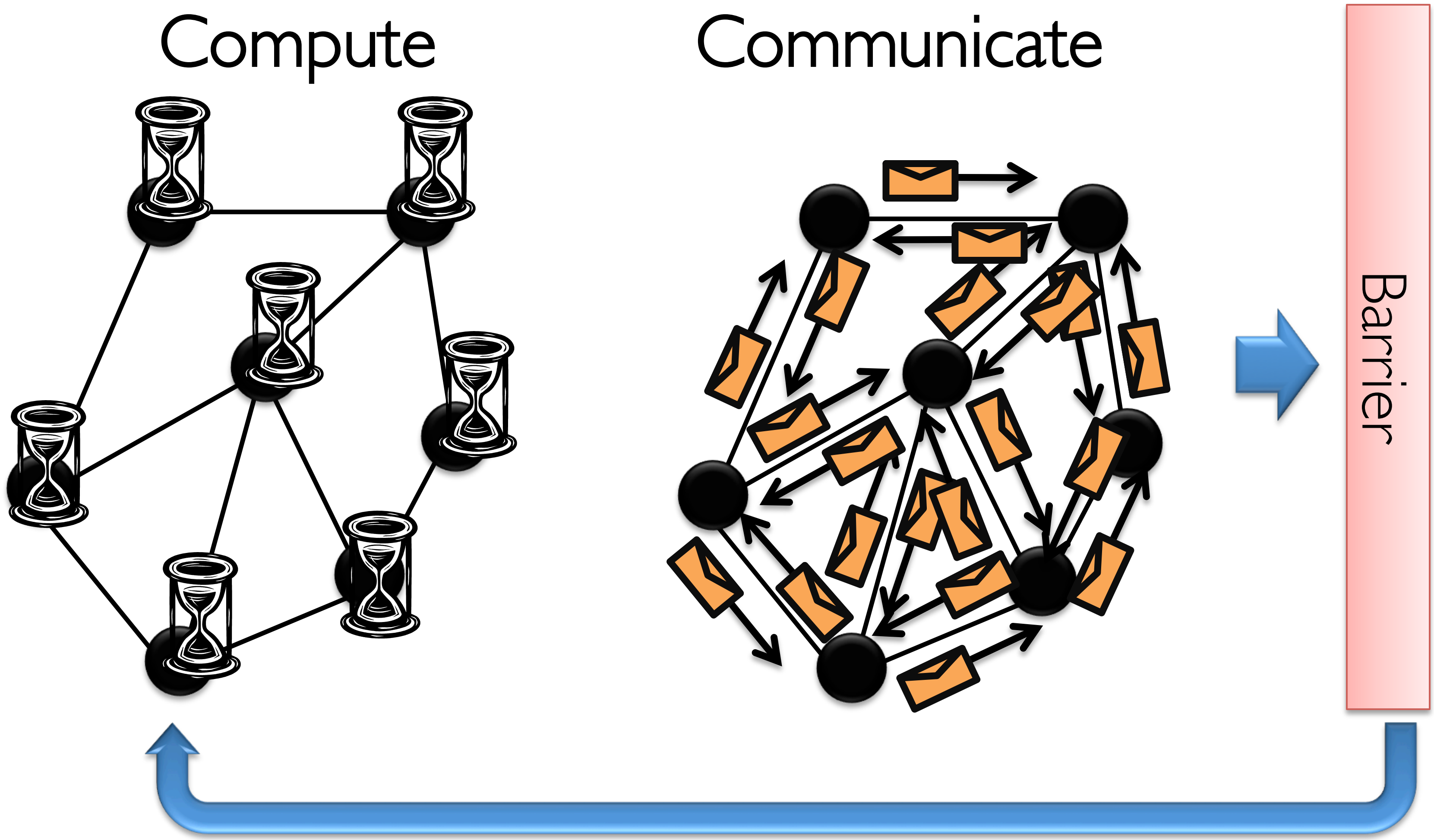
Vertex Programs directly **access** adjacent vertices and edges

```
GraphLab_PageRank(i)
  // Compute sum over neighbors
  total = 0
  foreach( j in neighbors(i)):
    total = total + R[j] * w_ji

  // Update the PageRank
  R[i] = 0.15 + total
```



Data movement is managed by the system and not the user.

Iterative Bulk Synchronous Execution



Graph-Parallel Systems

Pregel

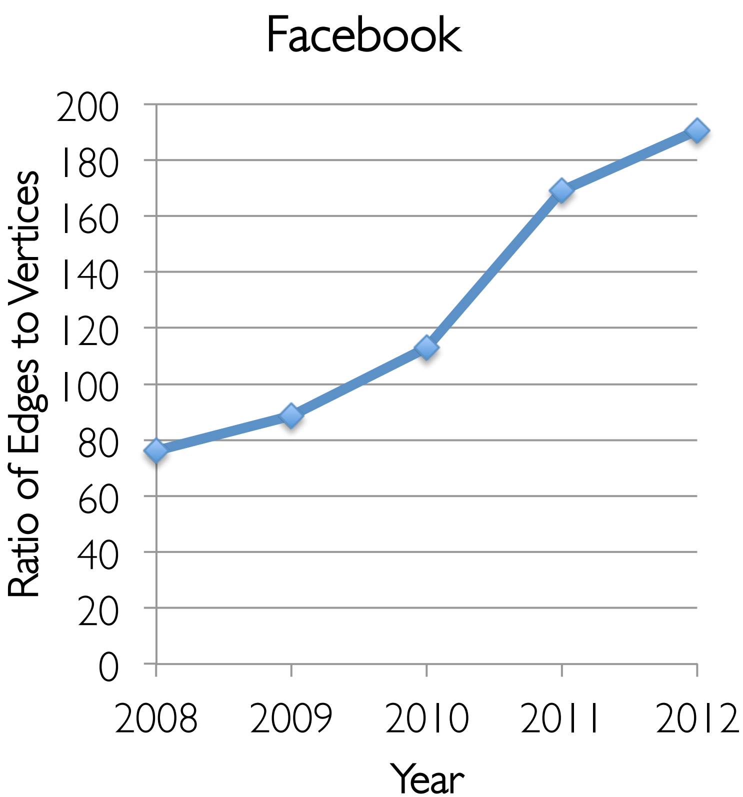
APACHE GIRAPH

GraphLab

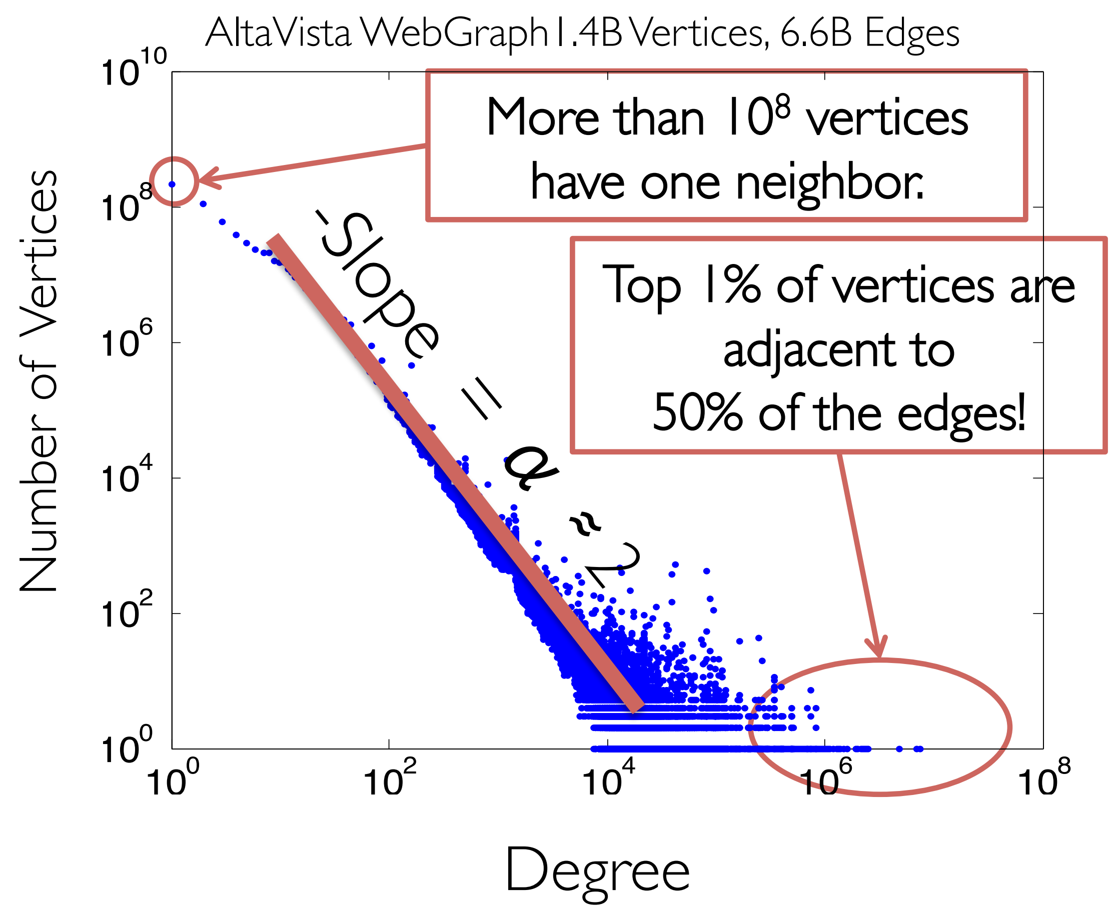
Exploit graph structure to achieve
orders-of-magnitude performance gains
over more general data-parallel systems.

Real-World Graphs

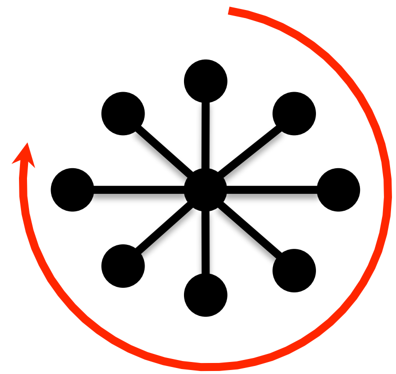
Edges >> Vertices



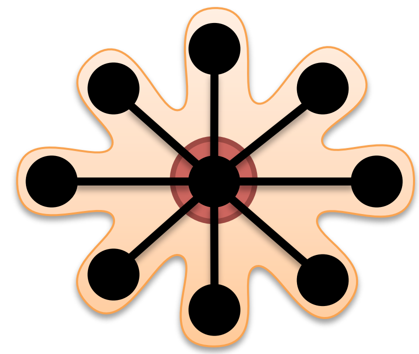
Power-Law Degree Distribution



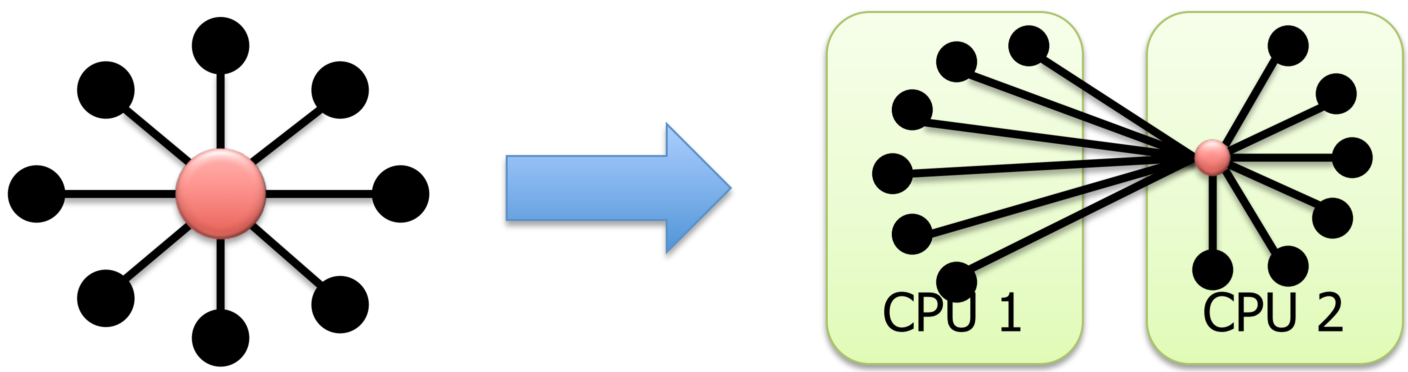
Challenges of **High-Degree** Vertices



Sequentially process edges



Touches a large fraction of graph

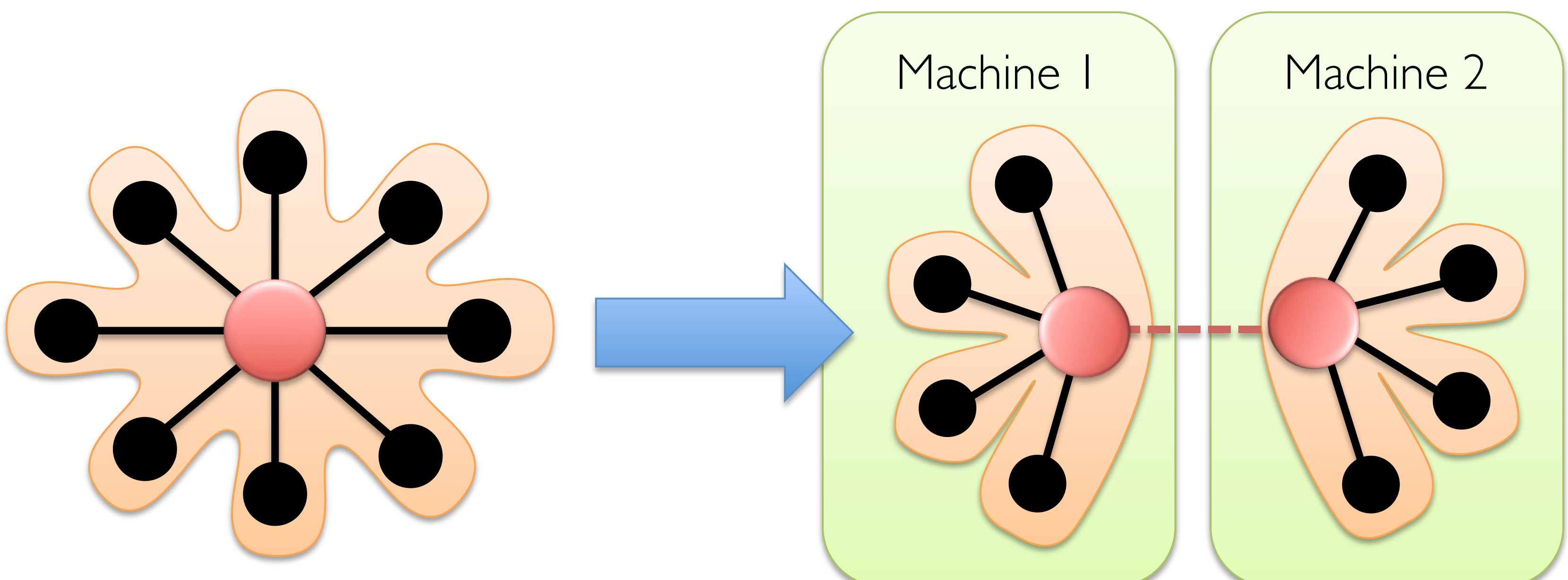


Provably Difficult to Partition

GraphLab

(PowerGraph, OSDI'12)

Program This | **Run on This**



Split **High-Degree** vertices

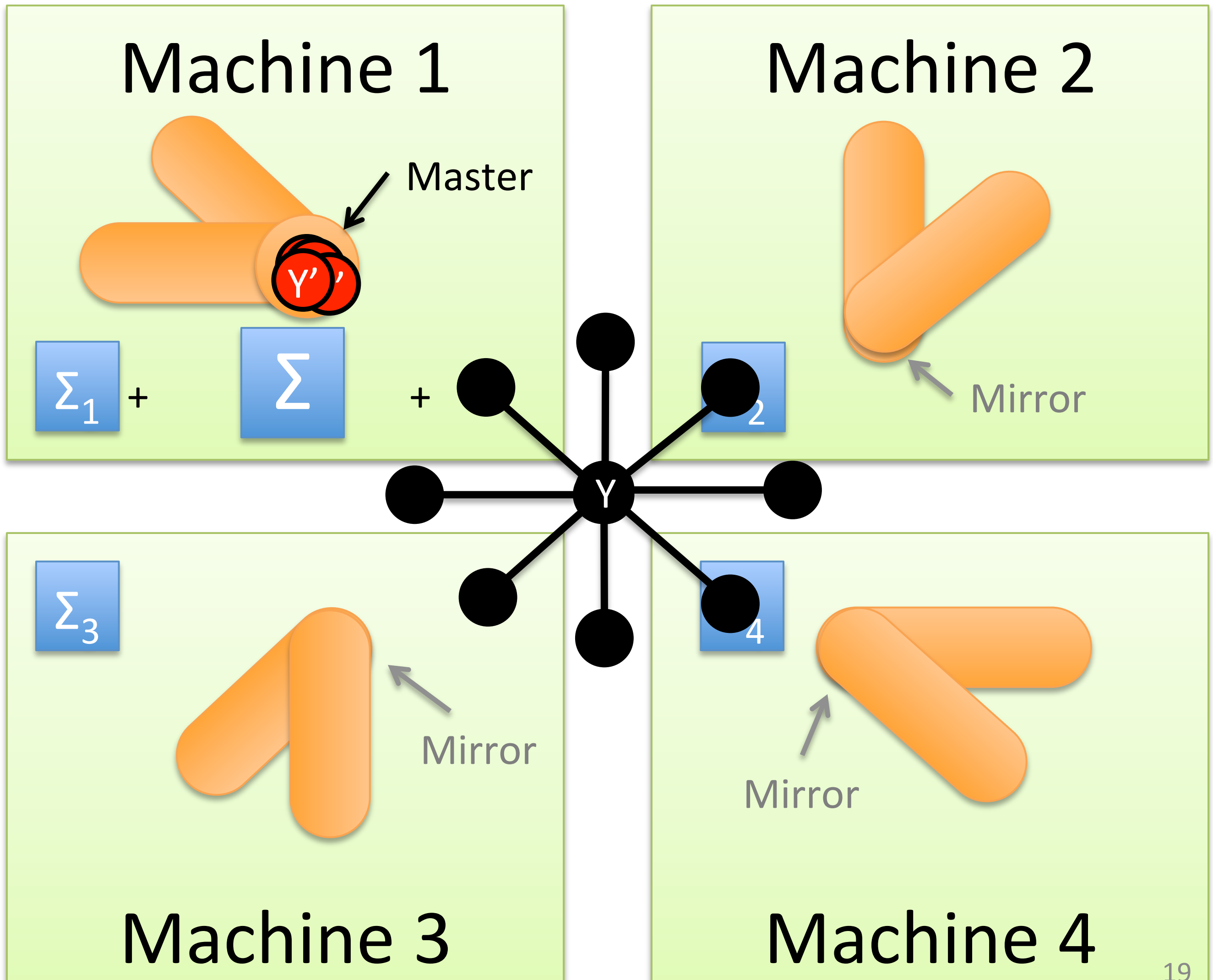
New Abstraction → *<u>Equivalence</u> on Split Vertices*

GAS Decomposition

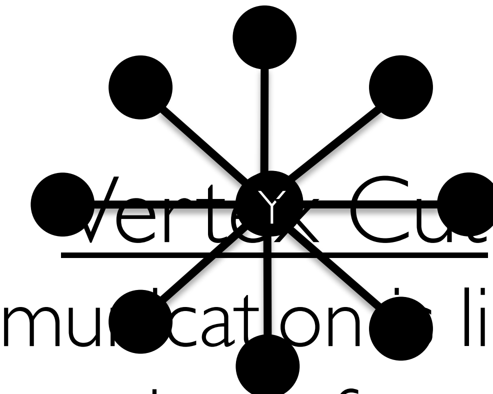
Gather

Apply

Scatter



Minimizing Communication in PowerGraph



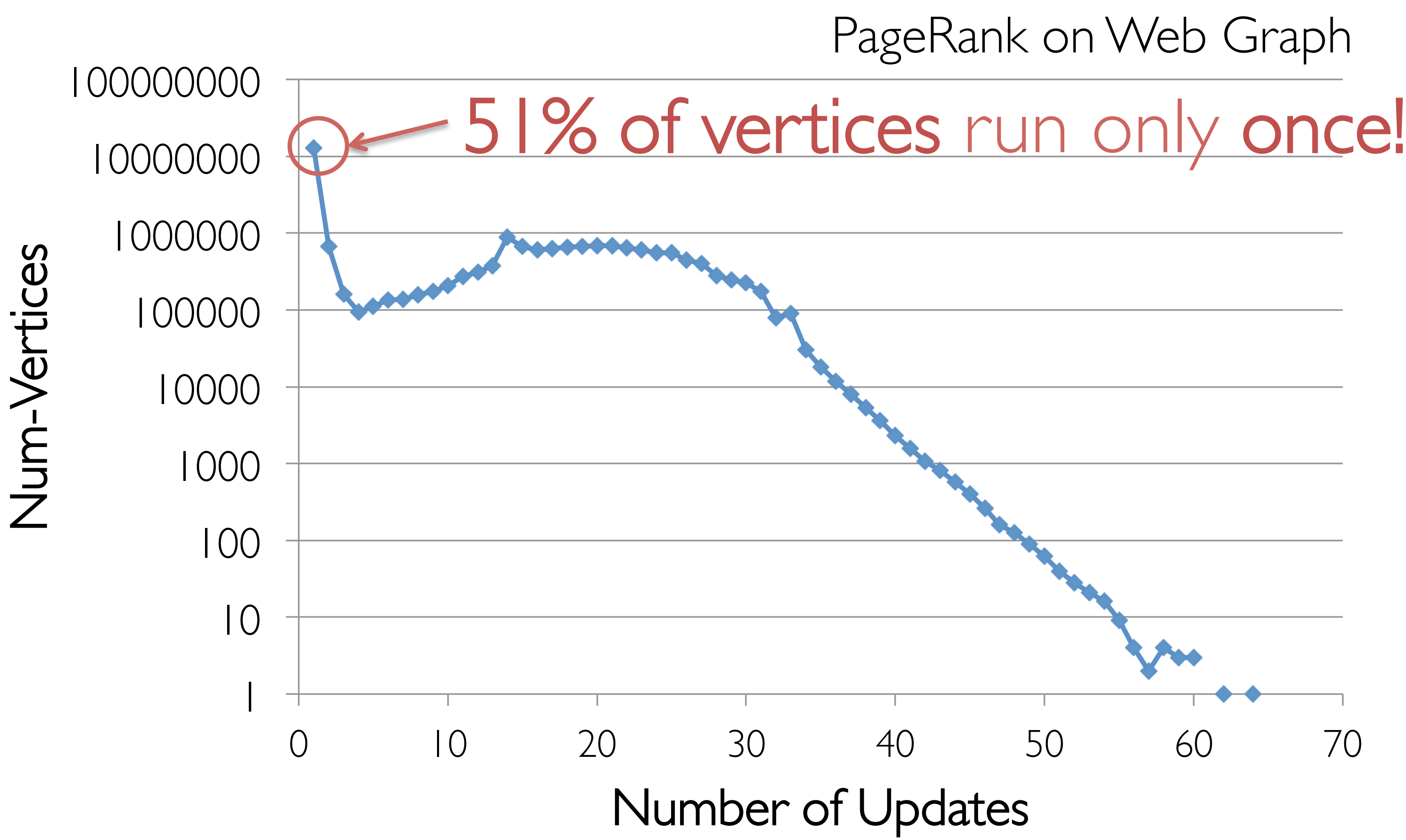
Vertex Cut

Communication is linear in the number of machines each vertex spans.

Total communication upper bound:

$$O\left(\#\text{vertices}\sqrt{\#\text{machines}}\right)$$

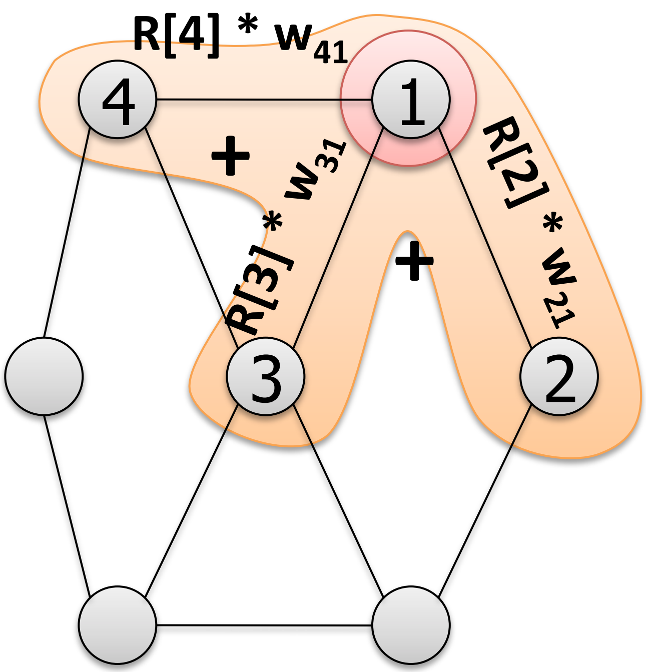
Shrinking Working Sets



The GraphLab (Pull) Abstraction

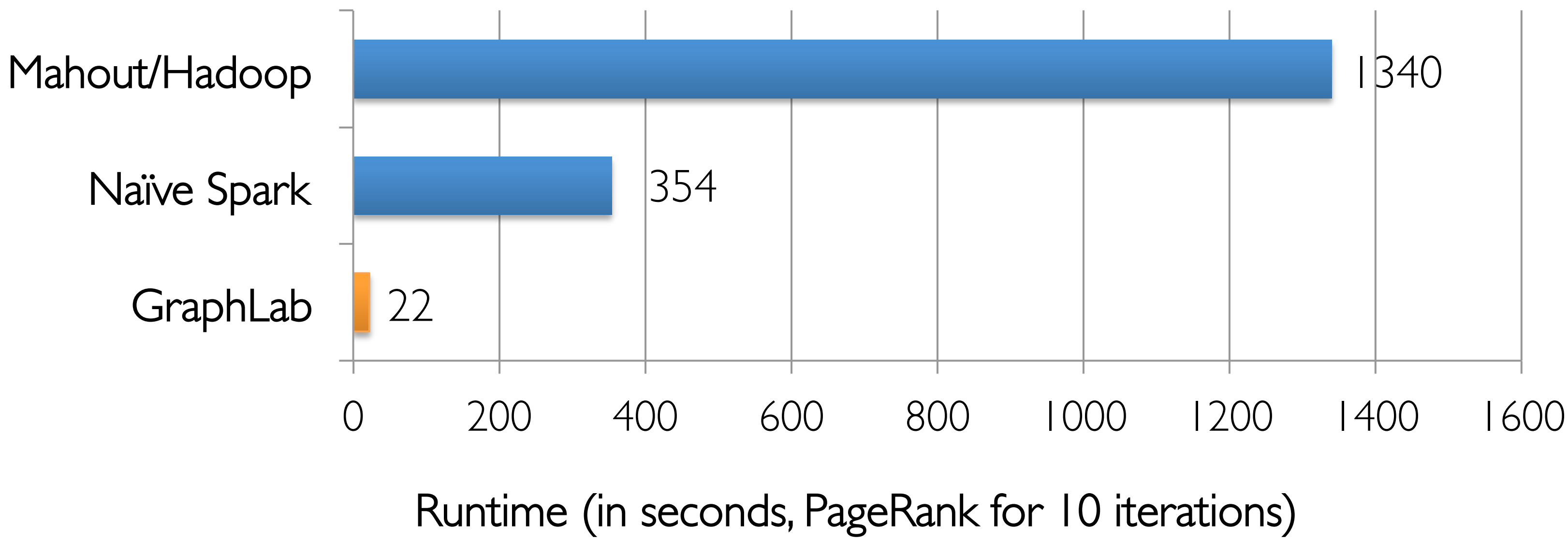
Vertex Programs directly **access** adjacent vertices and edges

```
GraphLab_PageRank(i)
  // Compute sum over neighbors
  total = 0
  foreach( j in neighbors(i)):
    total = total + R[j] * w_ji

  // Update the PageRank
  R[i] = 0.15 + total

  // Trigger neighbors to run again
  if R[i] not converged then
   signal nbrsOf(i) to be recomputed
```



Trigger computation *only* when necessary.

PageRank on the Live-Journal Graph

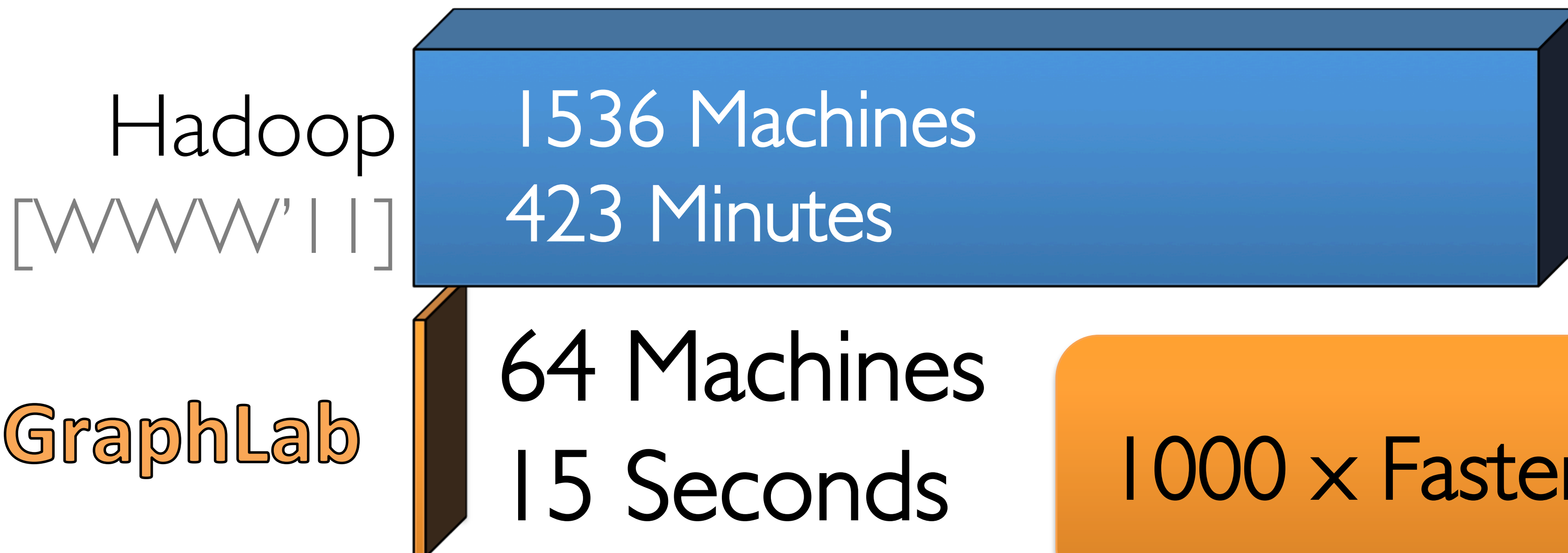


GraphLab is *60x faster* than Hadoop
GraphLab is *16x faster* than Spark

Triangle Counting on Twitter

40M Users, 1.4 Billion Links

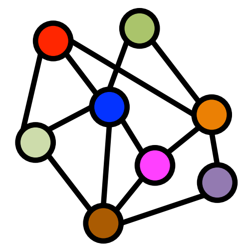
Counted: 34.8 Billion Triangles



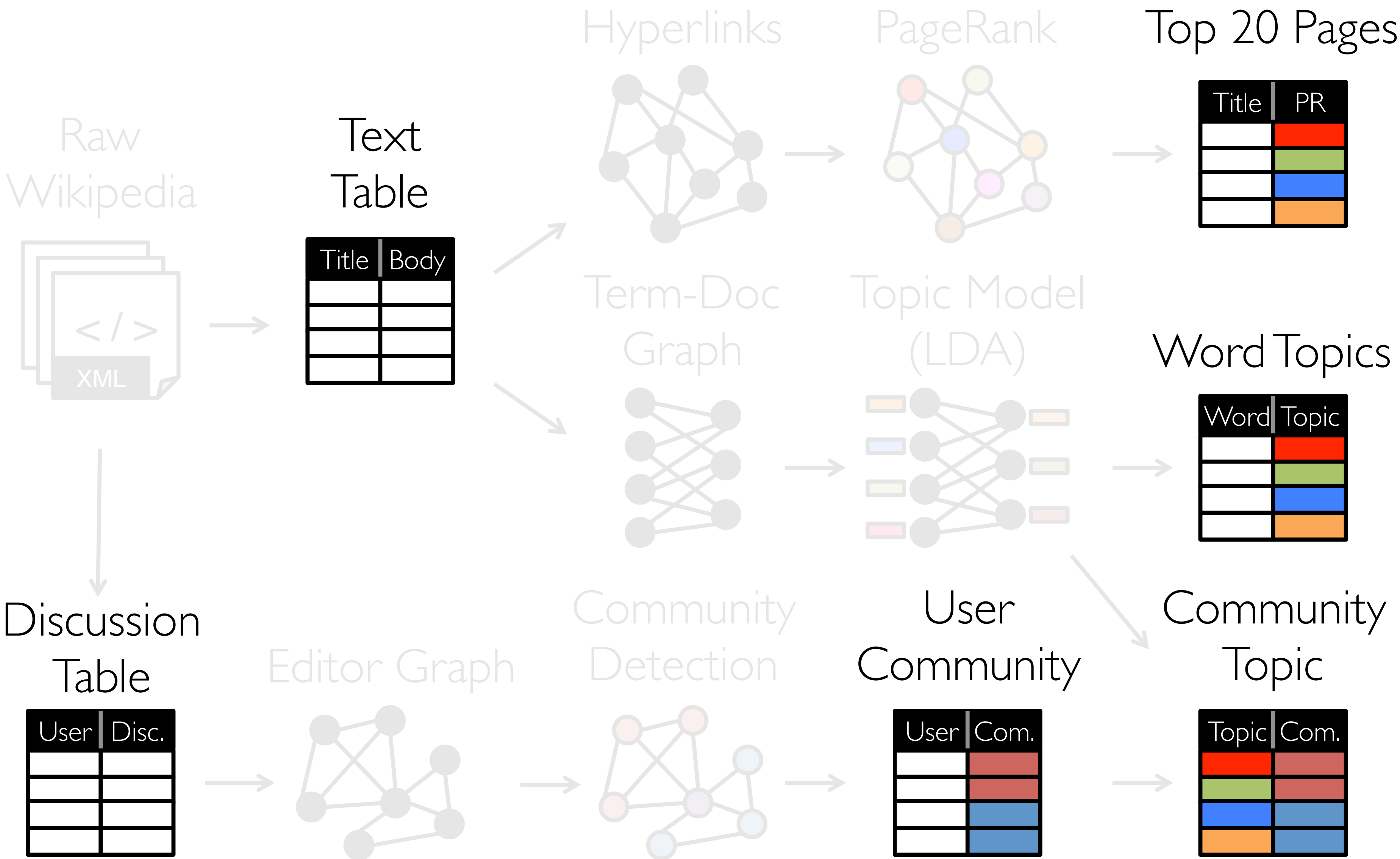
1000 x Faster

S. Suri and S. Vassilvitskii, "Counting triangles and the curse of the last reducer," WWW'11

PageRank

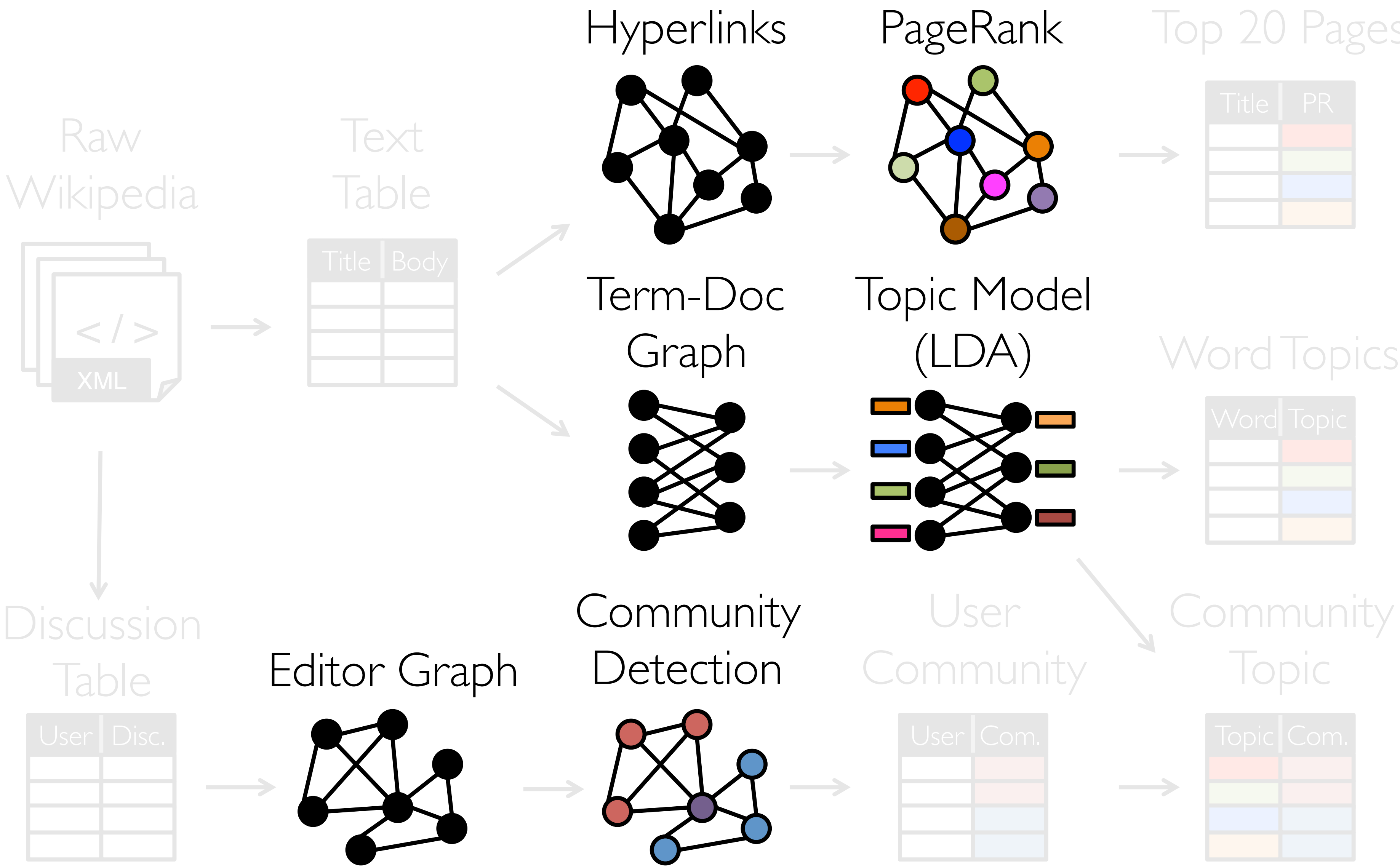


Tables



Raw Wikipedia
< / >
XML
Text Table
Title
Body
Hyperlinks
PageRank
Top 20 Pages
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

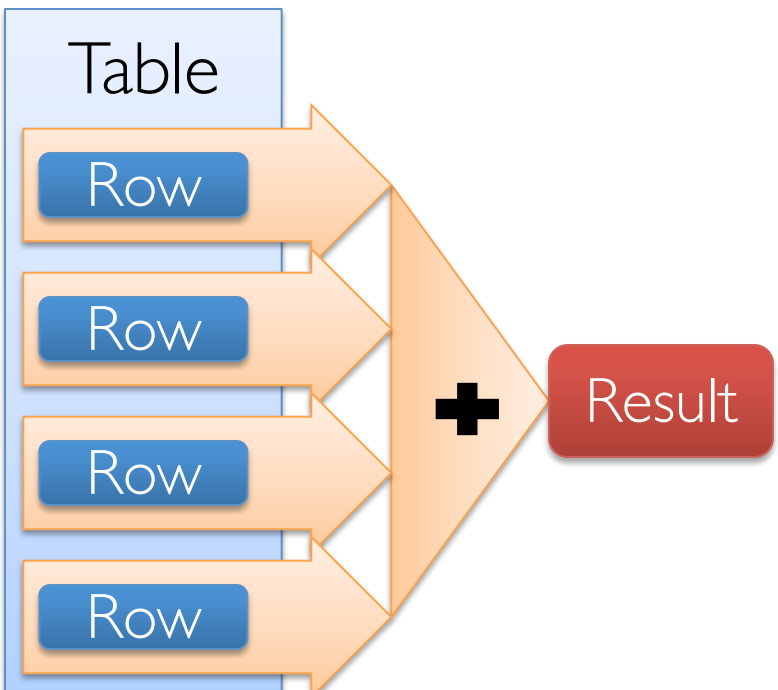

Raw Wikipedia
< / >
XML
Text Table
Title
Body
Hyperlinks
PageRank
Top 20 Pages
Title
PR
Term-Doc Graph
Topic Model (LDA)
Word Topics
Word
Topic
Discussion Table
User
Disc.
Editor Graph
Community Detection
User Community
User
Com.
Community Topic
Topic
Com.

Separate Systems to Support Each View

Table View

hadoop

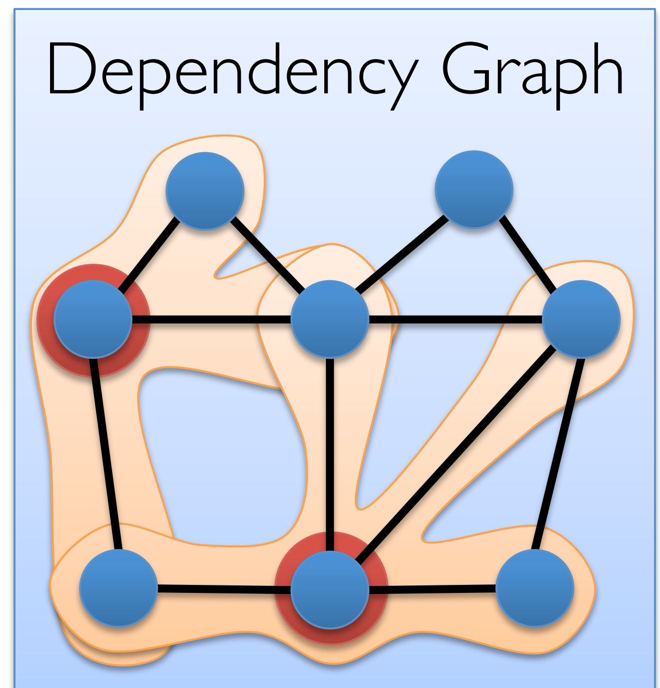
Spark



Graph View

Pregel GraphLab

APACHE GIRAPH



Separate systems for each view can be difficult to use and inefficient

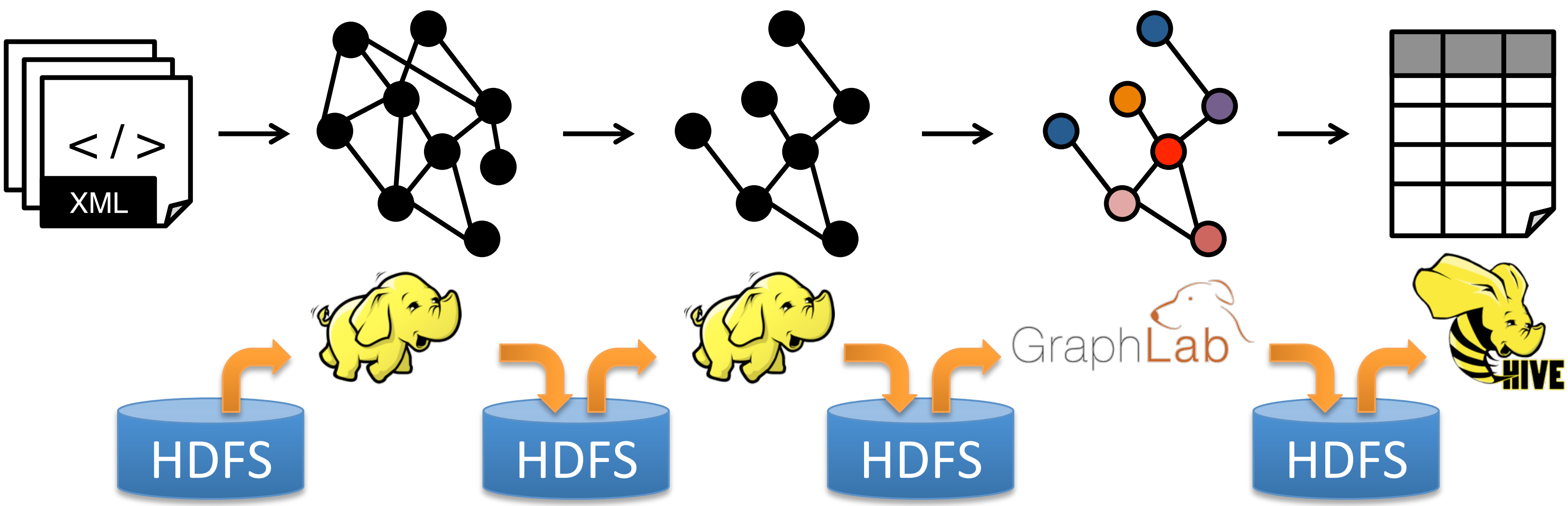
Difficult to Program and Use

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

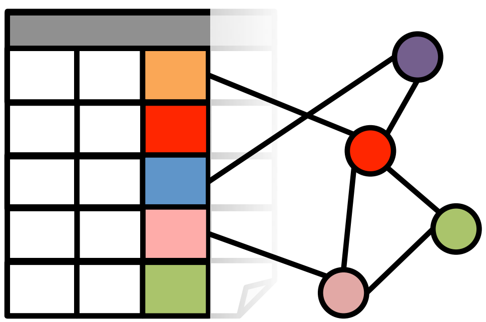


Limited reuse internal data-structures across stages

Solution: The GraphX Unified Approach

New API

Blurs the distinction between Tables and Graphs



New System

Combines Data-Parallel Graph-Parallel Systems

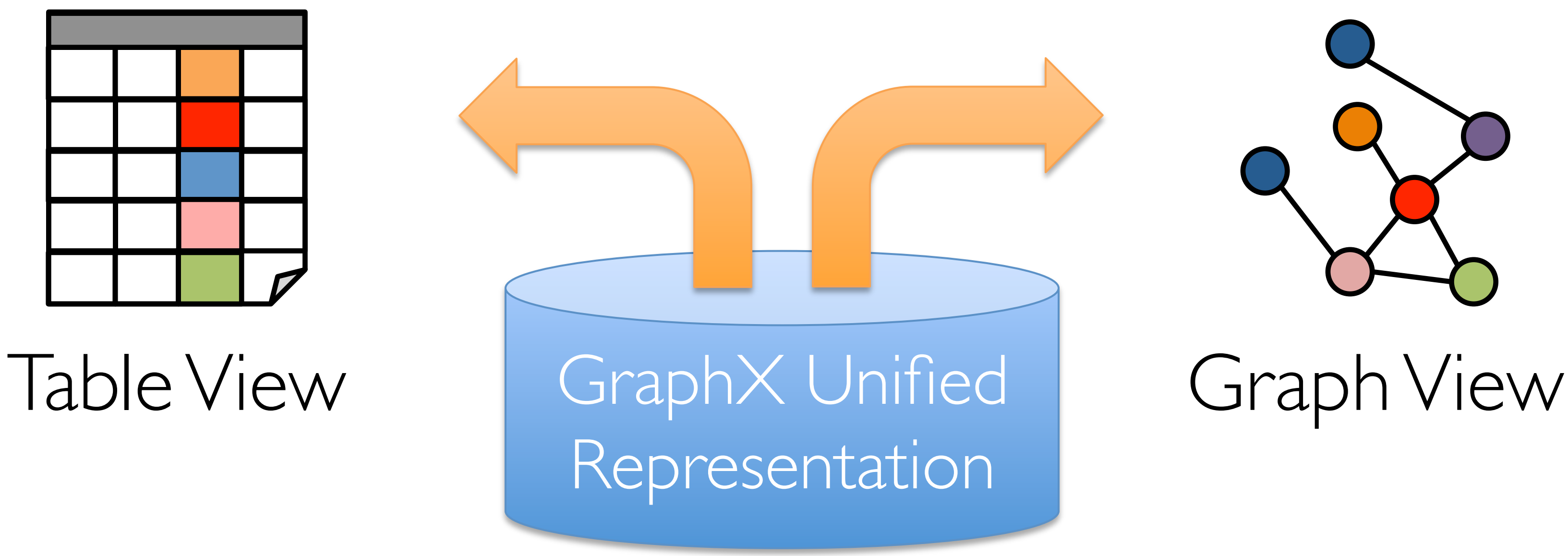
APACHE GIRAPH

Spark

GraphLab

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

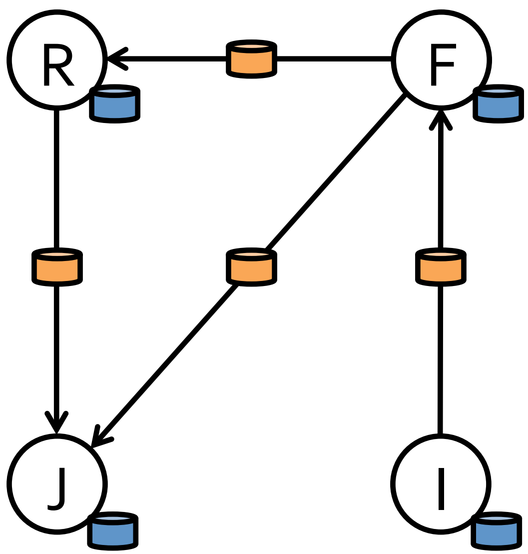
Tables and Graphs are composable *views* of the *same physical* data



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            ...
```

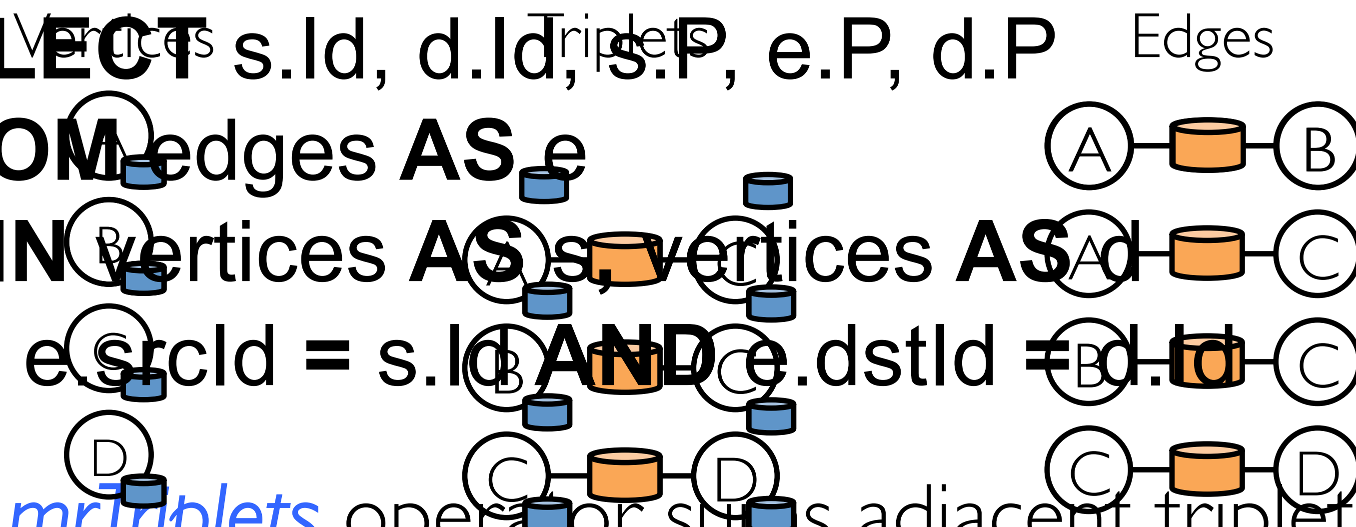
Graph Operators

```
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:

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!

Enhanced to Pregel in GraphX

```
pregelPR(i, messageSum ):
  // Receive all the messages
  total = 0          messageSum
  foreach( msg in messageList) :
    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]/E[i,j]) to vertex
```

```
combineMsg(a, b):
 // Compute sum of two messages
 return a + b
```

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

Require Message Combiners

Remove Message Computation from the Vertex Program

Implementing PageRank in GraphX

```
// Load and initialize the graph
val graph = GraphBuilder.text("hdfs://web.txt")
val prGraph = graph.joinVertices(graph.outDegrees)

// Implement and Run PageRank
val pageRank =
  prGraph.pregel(initialMessage = 0.0, iter = 10)(
    (oldV, msgSum) => 0.15 + 0.85 * msgSum,
    triplet => triplet.src.pr / triplet.src.deg,
    (msgA, msgB) => msgA + msgB)
```

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

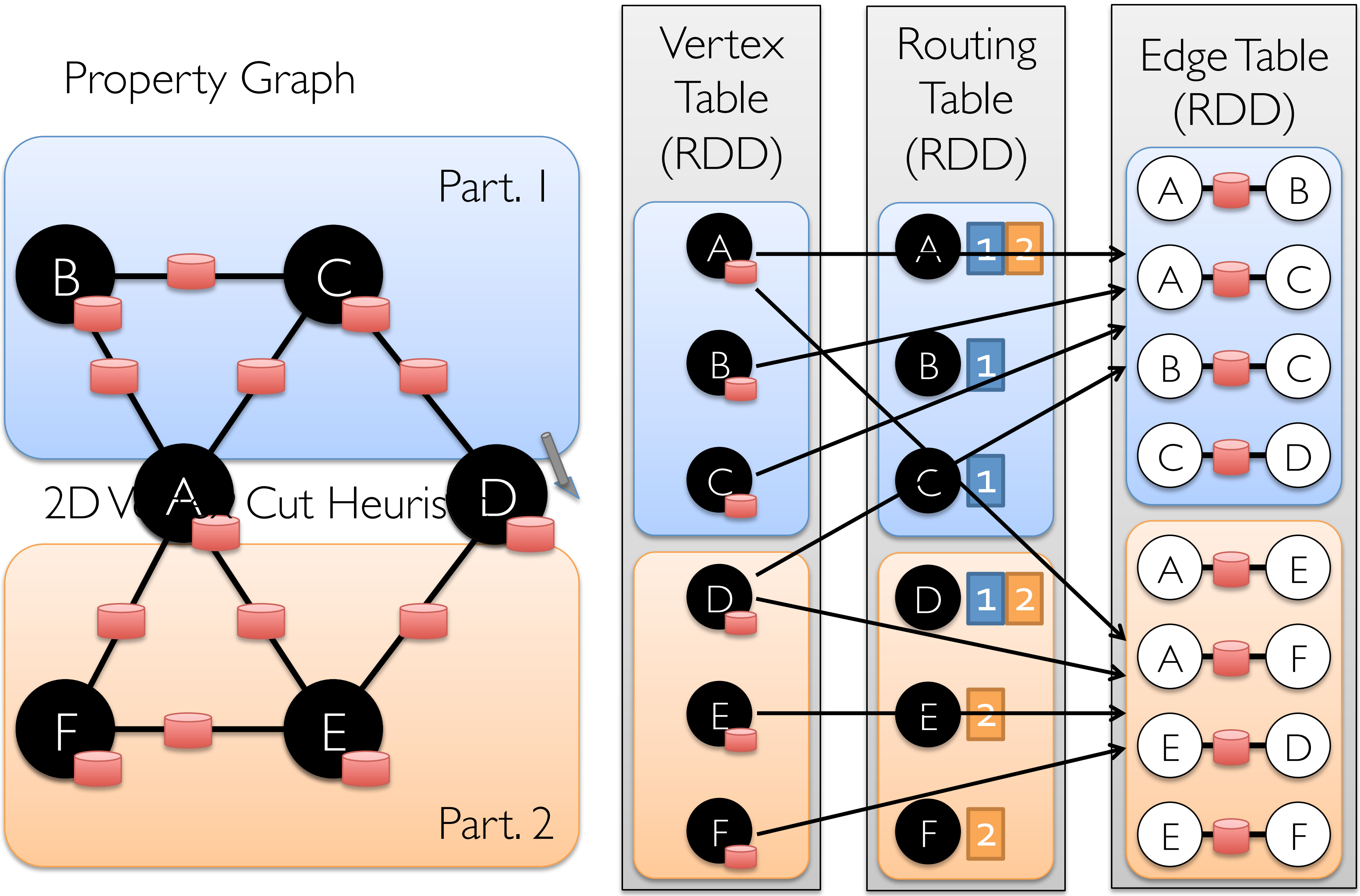
By composing these operators we can construct entire graph-analytics pipelines.

Example Analytics Pipeline

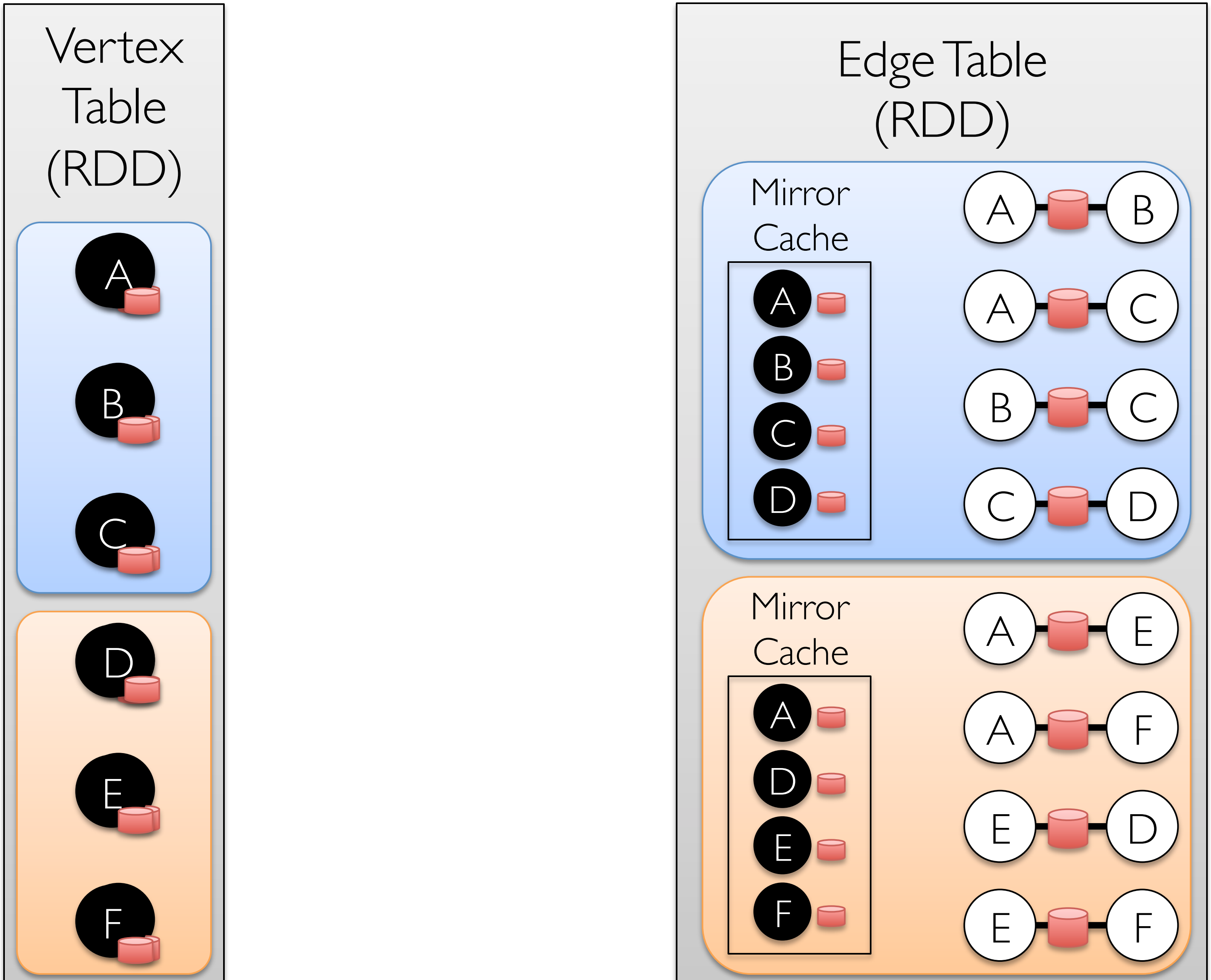
```
// Load raw data tables
val verts = sc.textFile(“hdfs://users.txt”).map(parserV)
val edges = sc.textFile(“hdfs://follow.txt”).map(parserE)
// Build the graph from tables and restrict to recent links
val graph = new Graph(verts, edges)
val recent = graph.subgraph(edge => edge.date > LAST_MONTH)
// Run PageRank Algorithm
val pr = graph.PageRank(tol = 1.0e-5)
// Extract and print the top 25 users
val topUsers = verts.join(pr).top(25).collect
topUsers.foreach(u => println(u.name + ‘\t’ + u.pr))
```

GraphX System Design

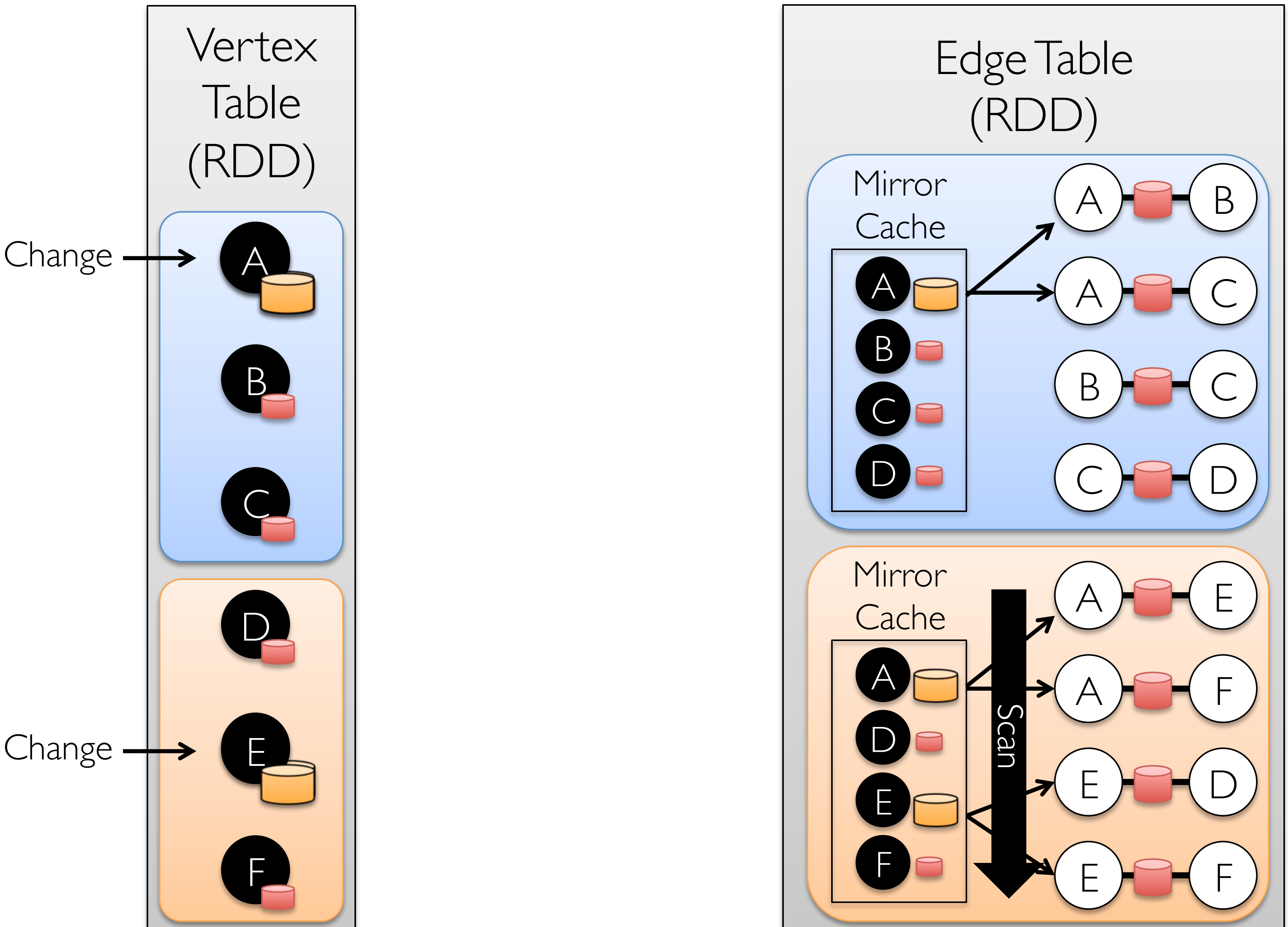
Distributed Graphs as Tables (RDDs)



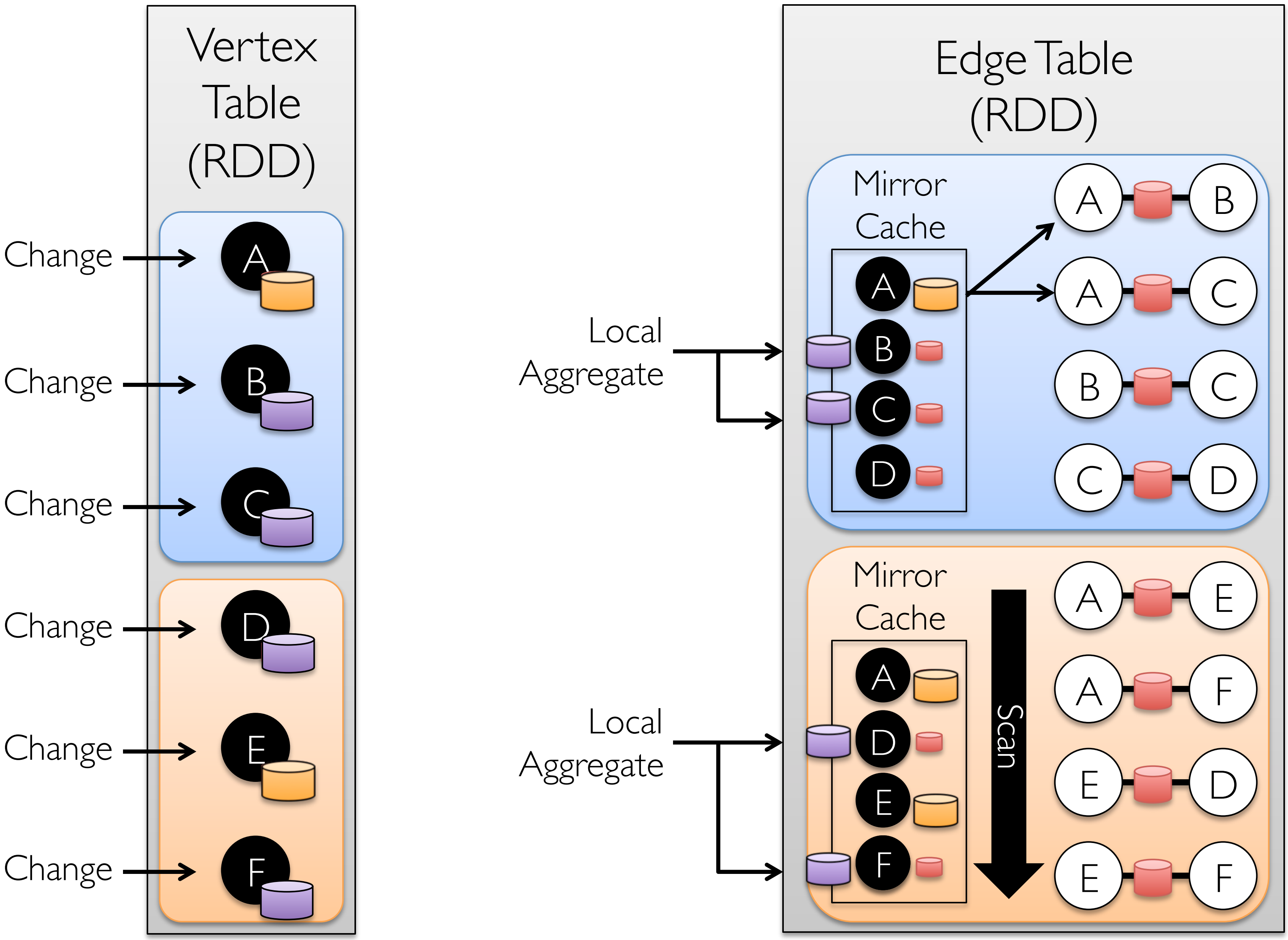
Caching for Iterative mrTriplets



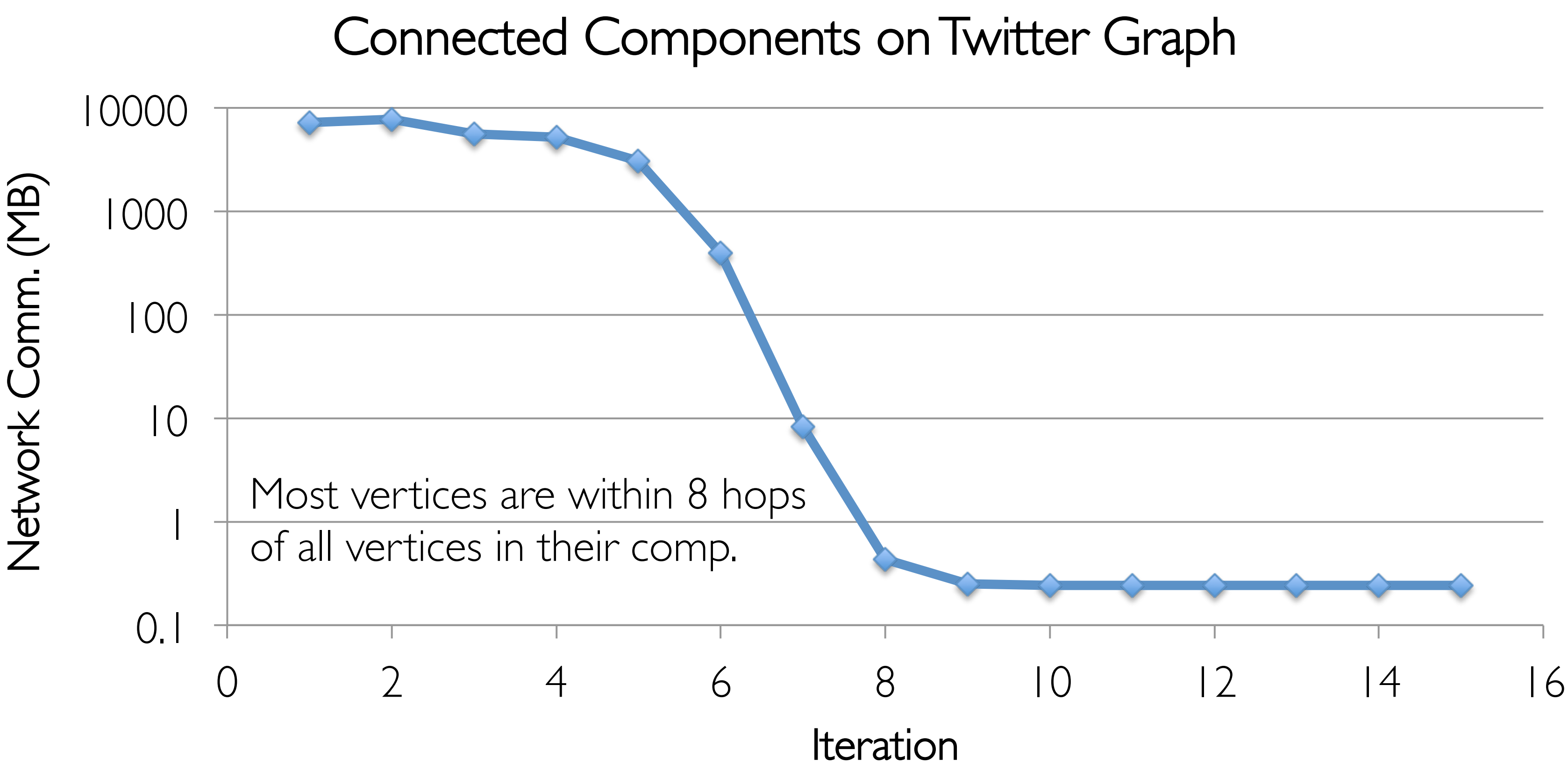
Incremental Updates for Iterative mrTriplets



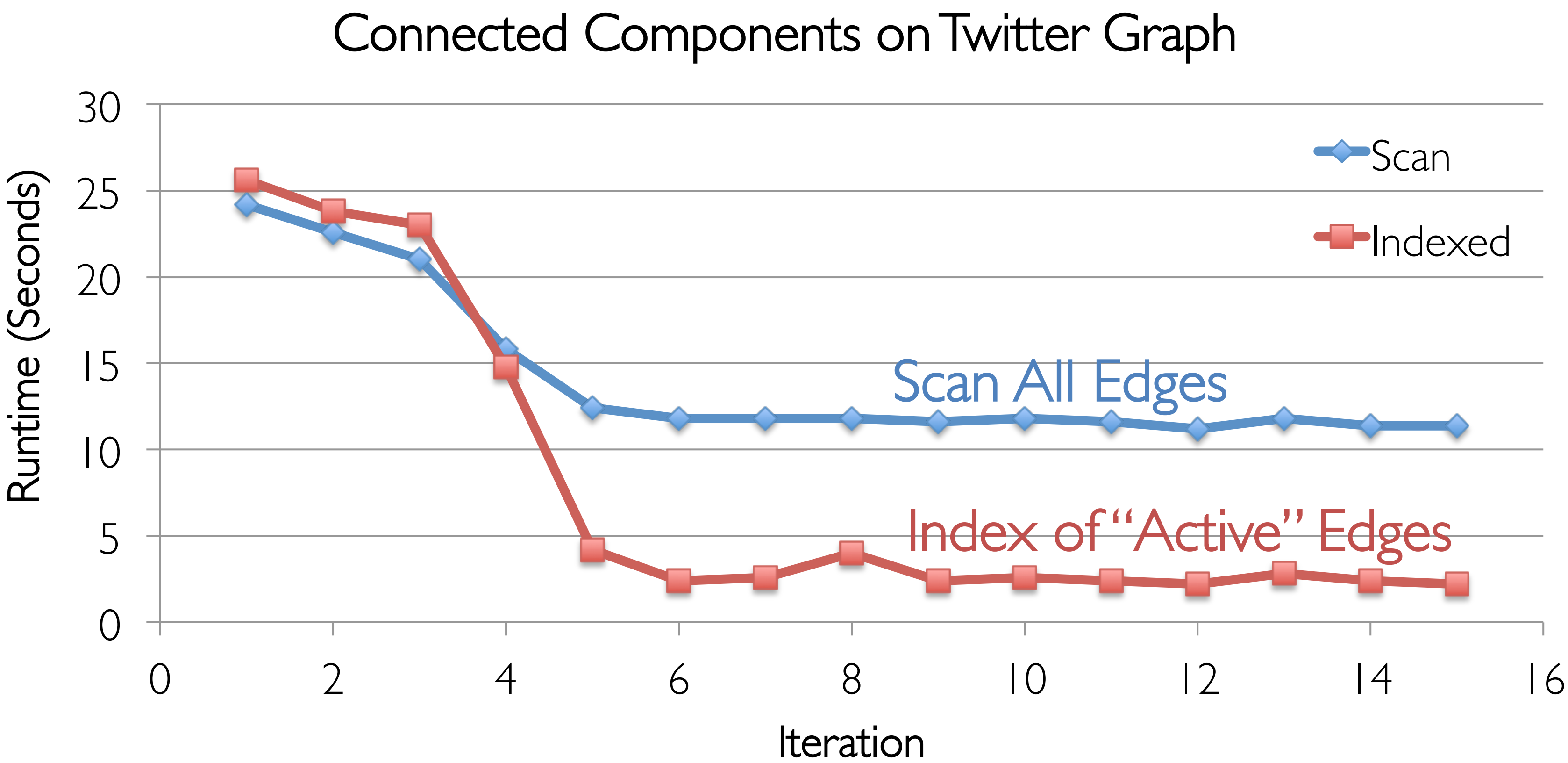
Aggregation for Iterative mrTriplets



Reduction in Communication Due to Cached Updates



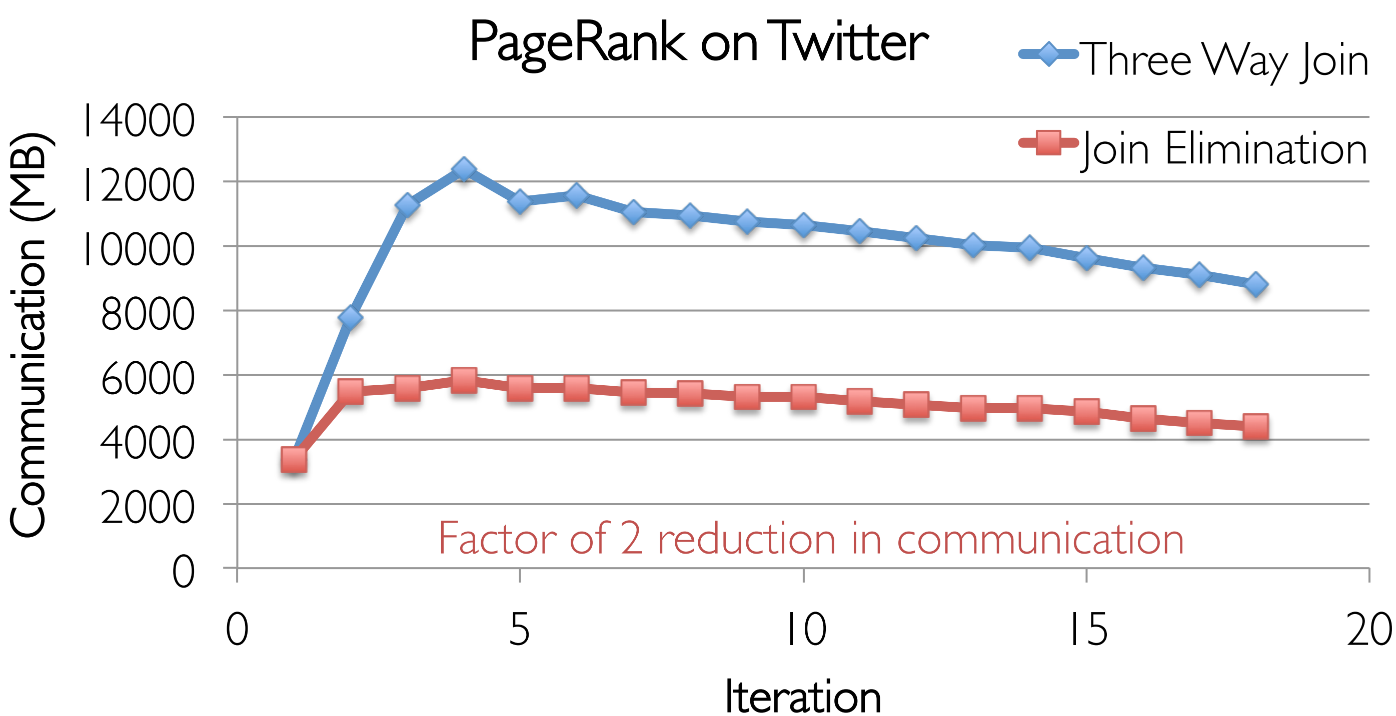
Benefit of Indexing *Active* Edges



Join Elimination

Identify and bypass joins for unused 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

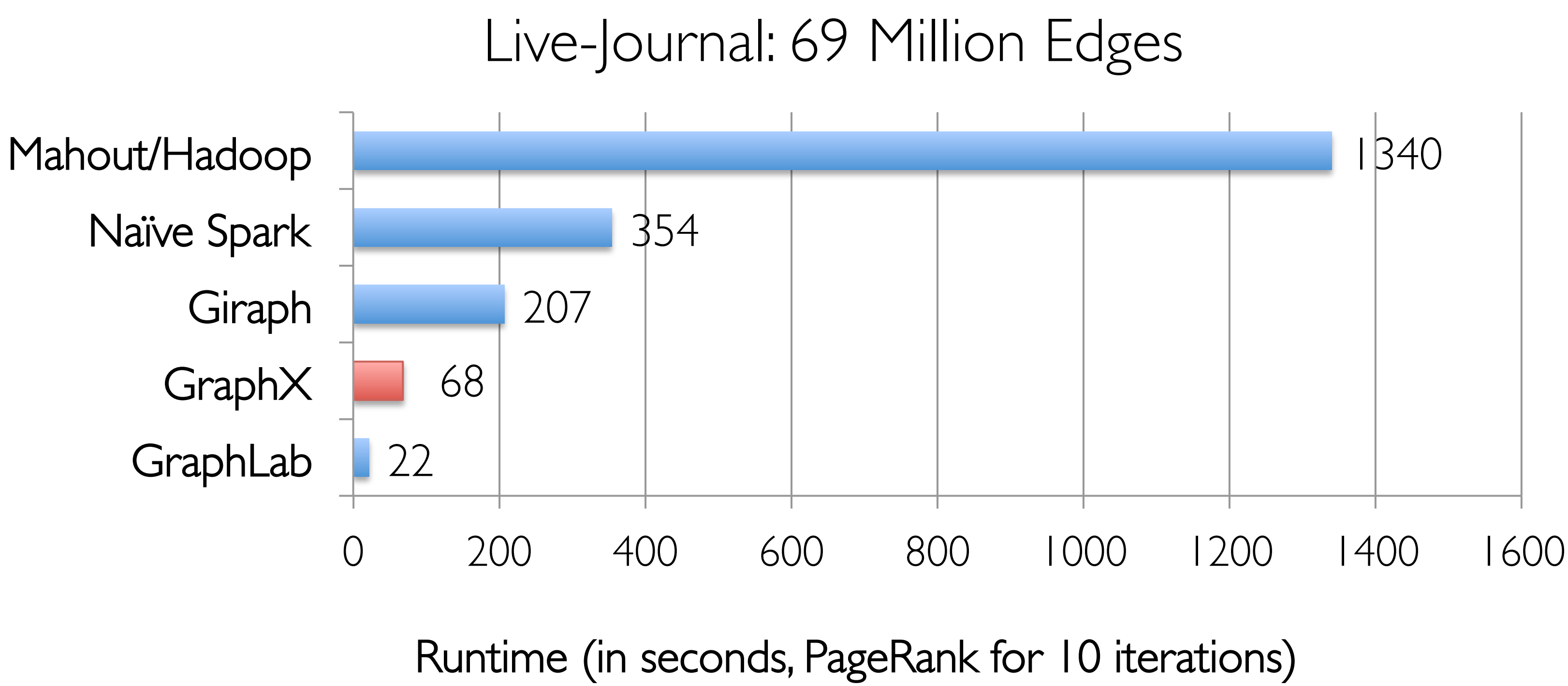
Indexing and Bitmaps:

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

Substantial Index and Data Reuse:

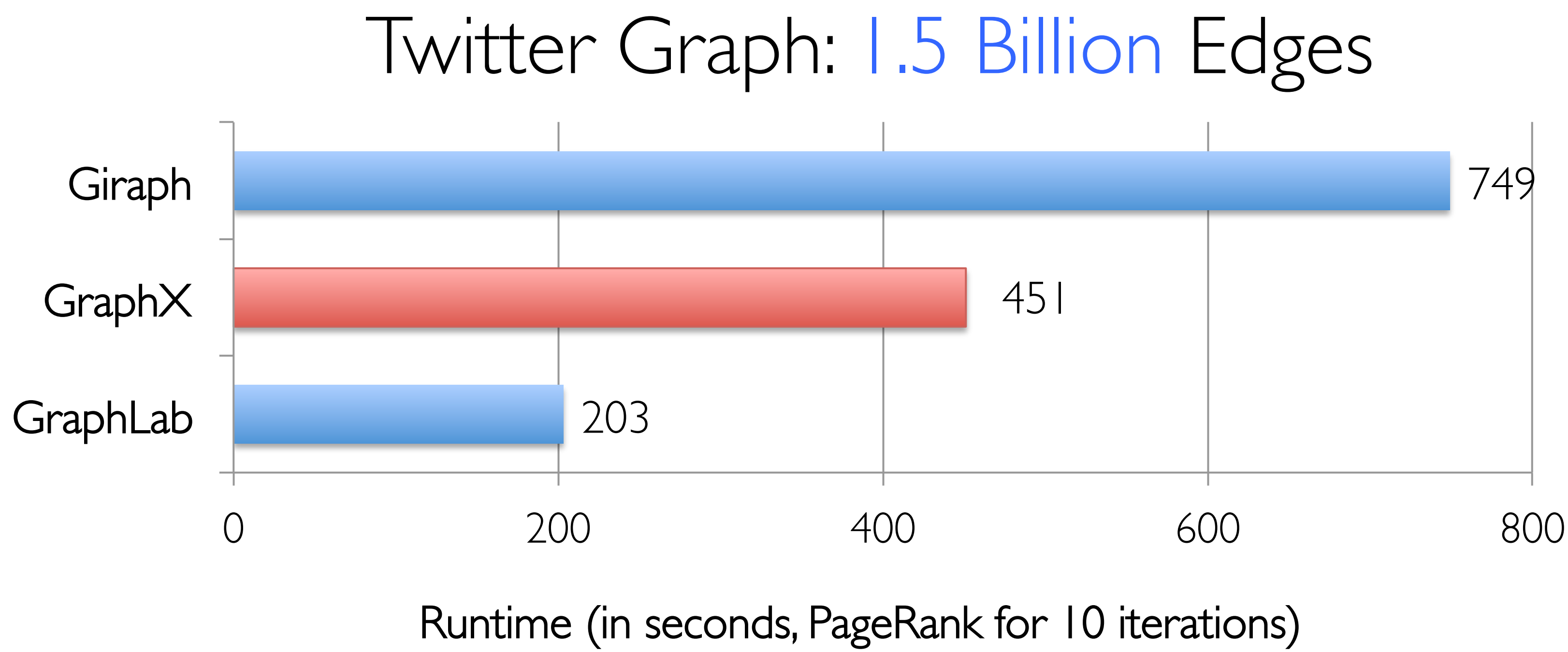
- » Reuse routing tables across graphs and sub-graphs
- » Reuse edge adjacency information and indices

Performance Comparisons



GraphX is roughly 3x slower than GraphLab

GraphX scales to larger graphs



GraphX is roughly 2x slower than GraphLab

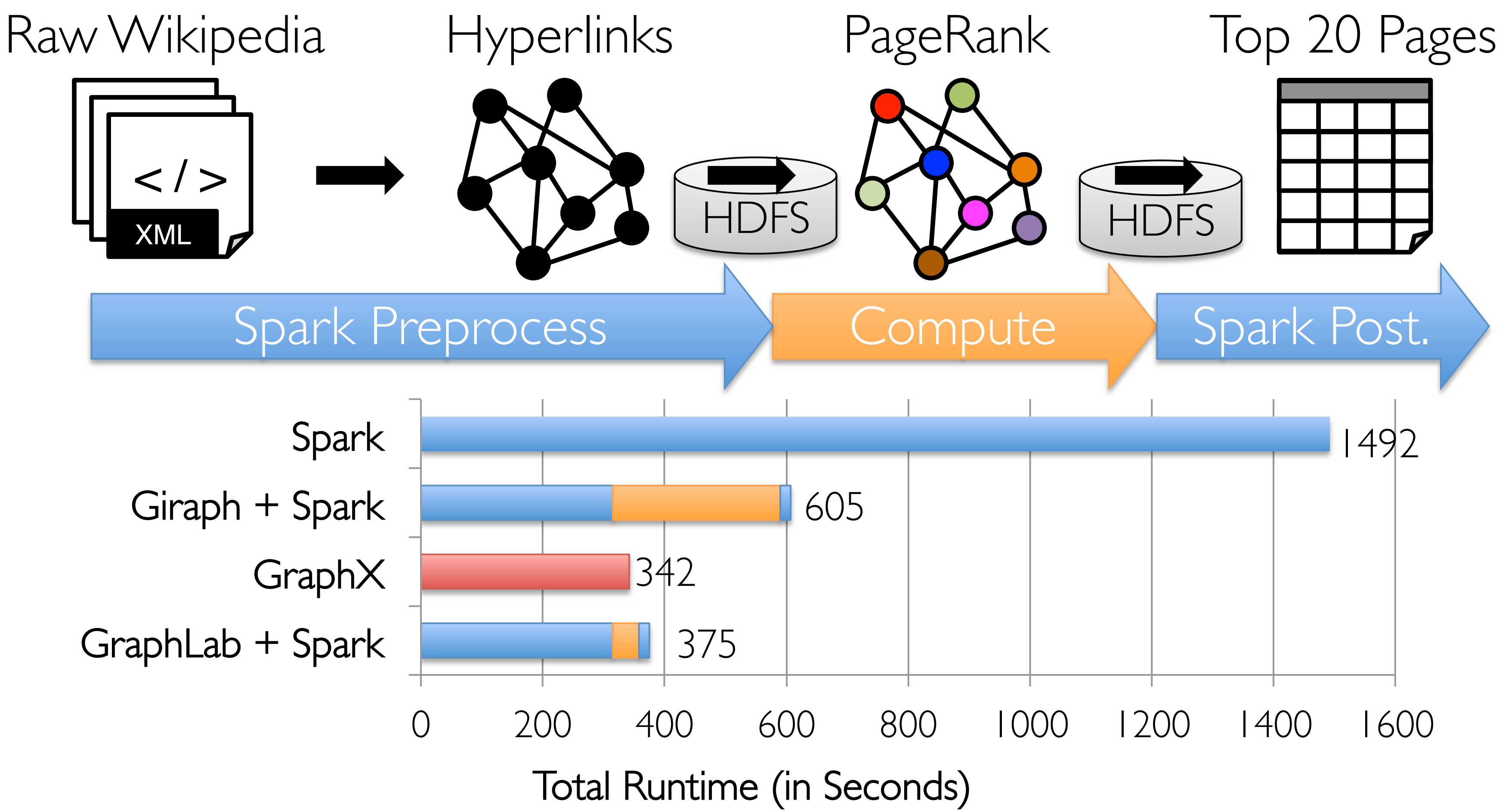
» Scala + Java overhead: Lambdas, GC time, …

» 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

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

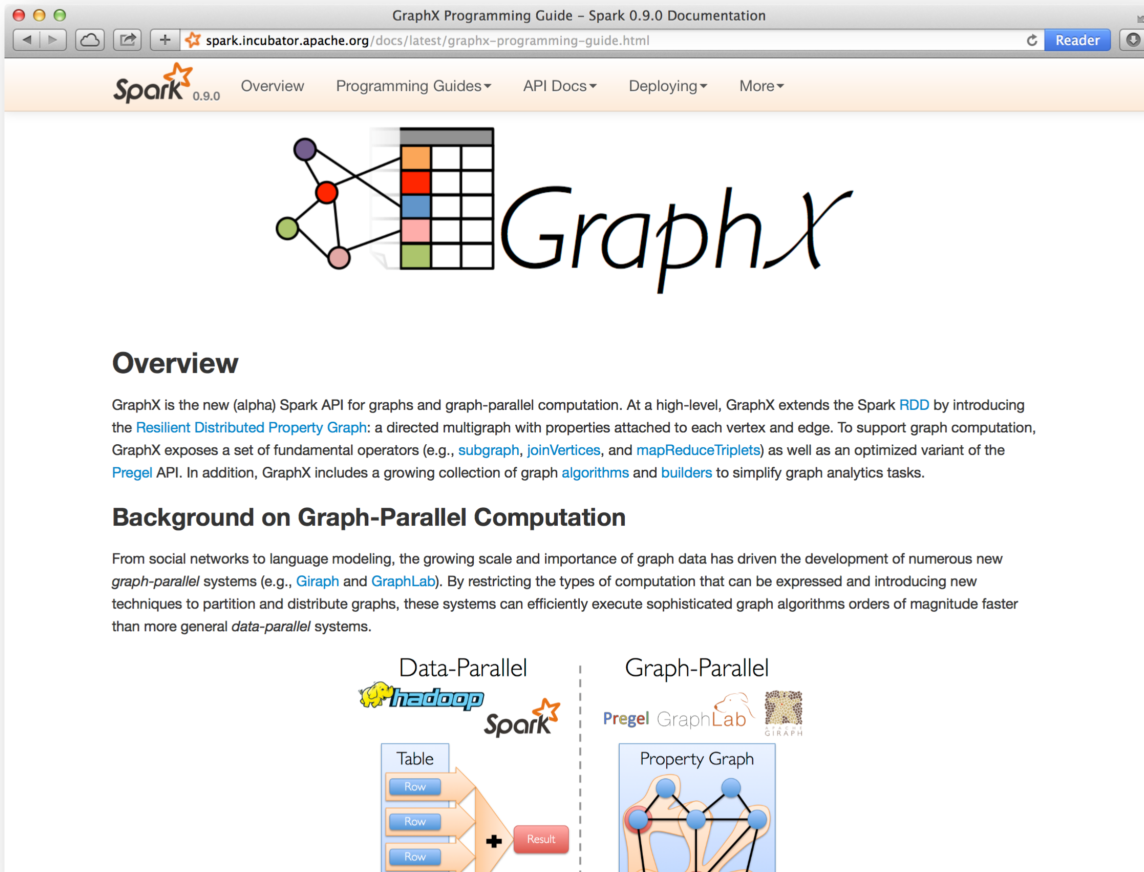
Graphs through the lens of database systems

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

» Graph Systems → Distributed join optimizations

Open Source Project

Alpha release as part of Spark 0.9



Overview

GraphX is the new (alpha) Spark API for graphs and graph-parallel computation. At a high-level, GraphX extends the Spark RDD by introducing the Resilient Distributed Property Graph: a directed multigraph with properties attached to each vertex and edge. To support graph computation, GraphX exposes a set of fundamental operators (e.g., subgraph, joinVertices, and mapReduceTriplets) as well as an optimized variant of the Pregel API. In addition, GraphX includes a growing collection of graph algorithms and builders to simplify graph analytics tasks.

Background on Graph-Parallel Computation

From social networks to language modeling, the growing scale and importance of graph data has driven the development of numerous new *graph-parallel* systems (e.g., Giraph and GraphLab). By restricting the types of computation that can be expressed and introducing new techniques to partition and distribute graphs, these systems can efficiently execute sophisticated graph algorithms orders of magnitude faster than more general *data-parallel* systems.

Active Research

Static Data → Dynamic Data

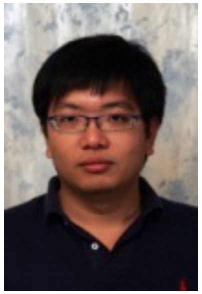
- » Apply GraphX unified approach to time evolving data
- » Materialized view maintenance for graphs

Serving Graph Structured Data

- » Allow external systems to interact with GraphX
- » Unify distributed graph databases with relational database technology

Collaborators

GraphLab:


Yucheng Low


Haijie Gu


Aapo Kyrola


Danny Bickson


Carlos Guestrin


Alex Smola


Guy Blelloch

GraphX:


Reynold Xin


Ankur Dave


Daniel Crankshaw


Michael Franklin


Ion Stoica

Thanks!

http://tinyurl.com/ampgraphx

jegonzal@eecs.berkeley.edu