DATABRICKS

GRAPHX: UNIFIED GRAPH ANALYTICS ON SPARK

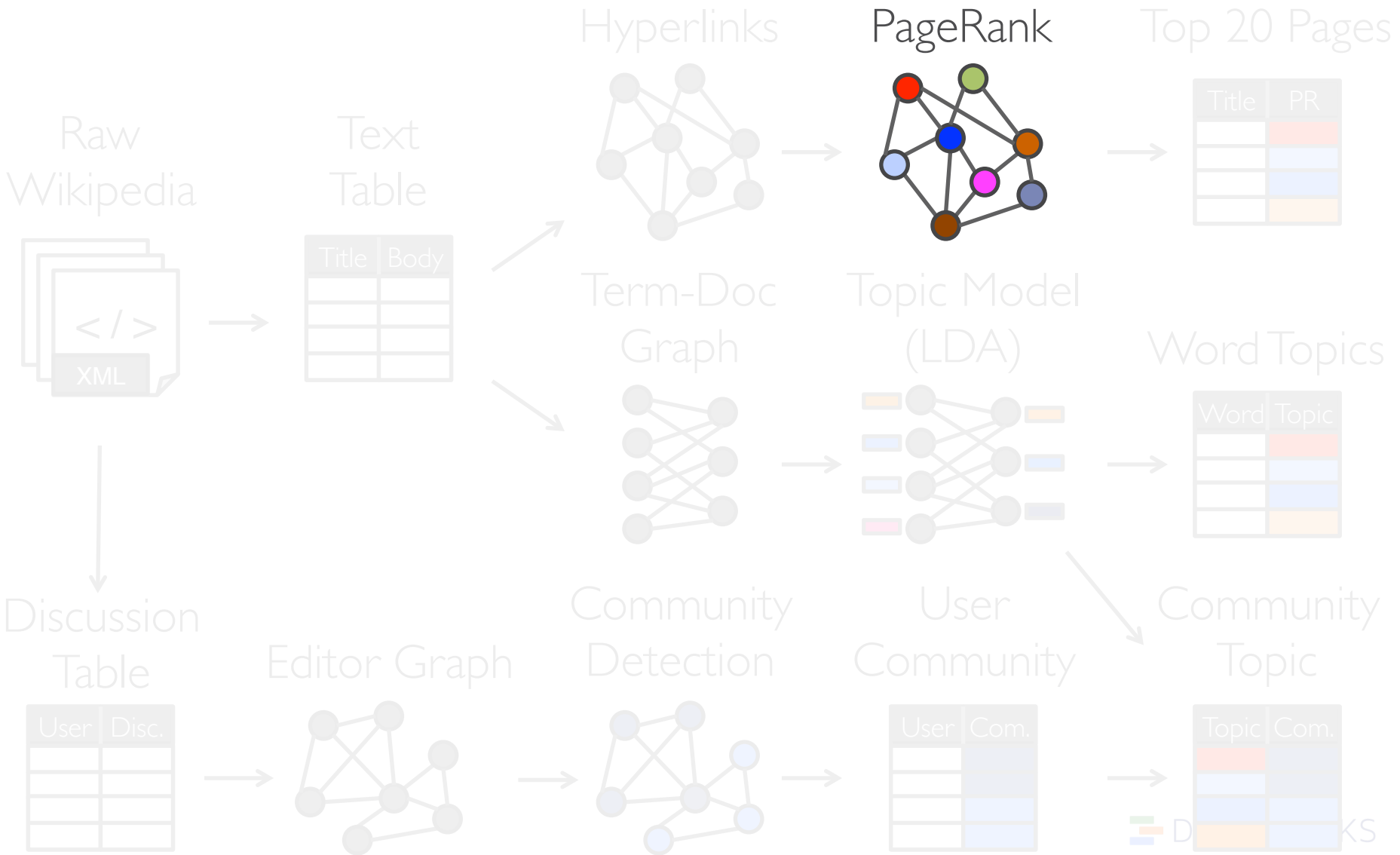
Joseph Gonzalez, Reynold Xin, **Ankur Dave**, Daniel Crankshaw,
Michael Franklin, and Ion Stoica

*These slides are best viewed in PowerPoint with animation.

1. Motivation for GraphX
2. GraphX Implementation
3. Future Directions

Motivation for GraphX

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*

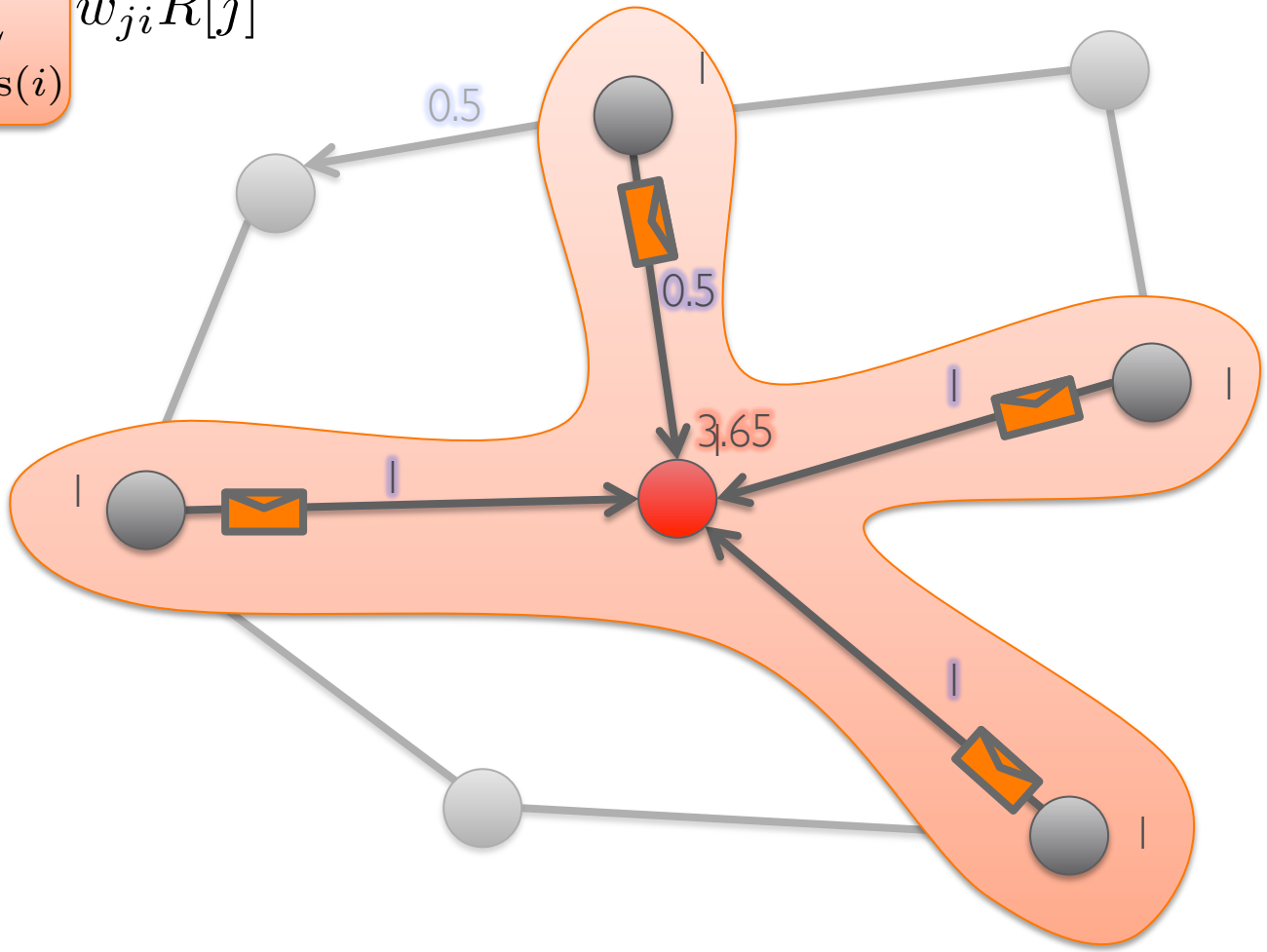
Weighted sum of neighbors' ranks

Iterate until convergence

Update ranks in parallel

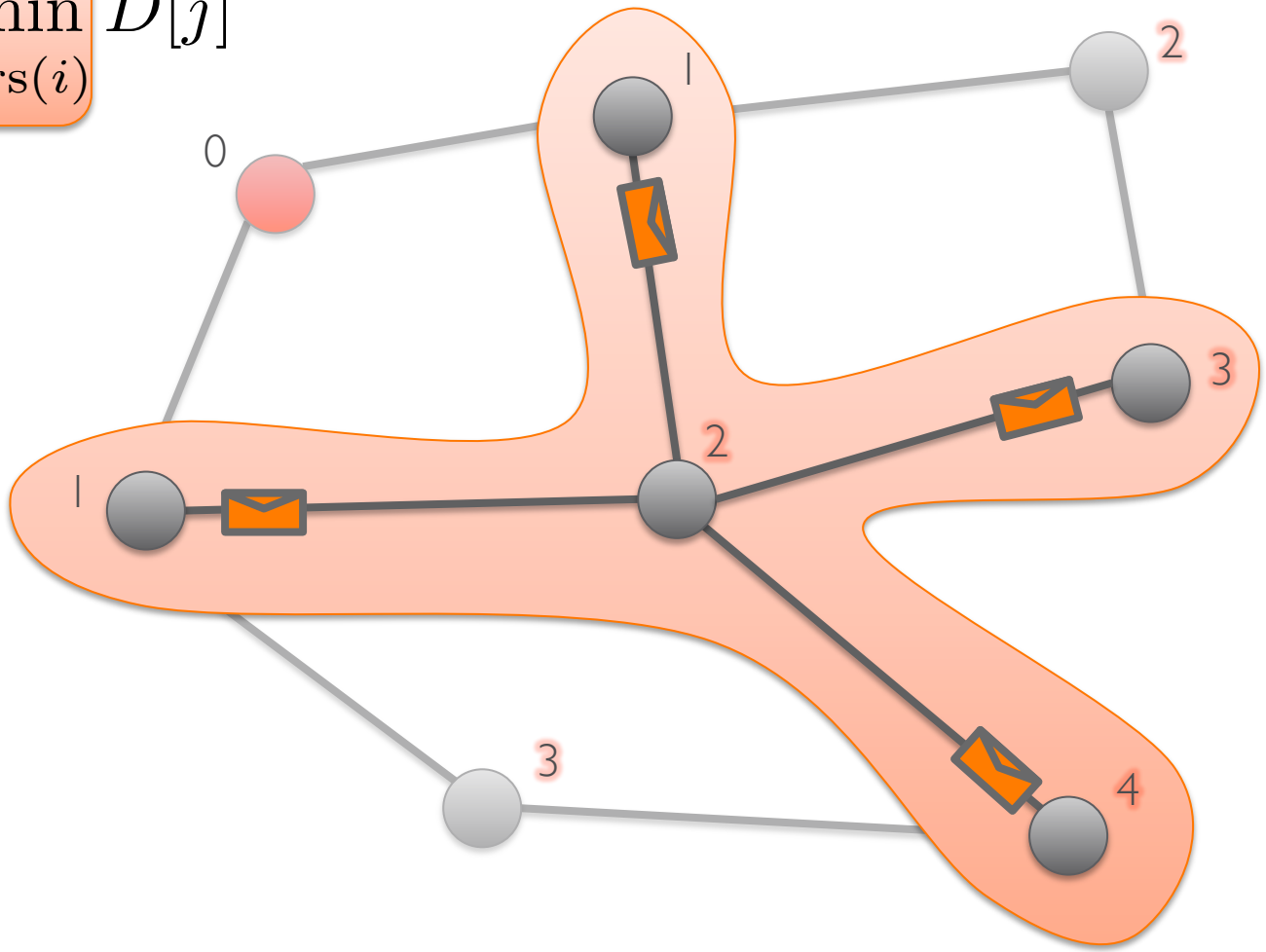
PageRank

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

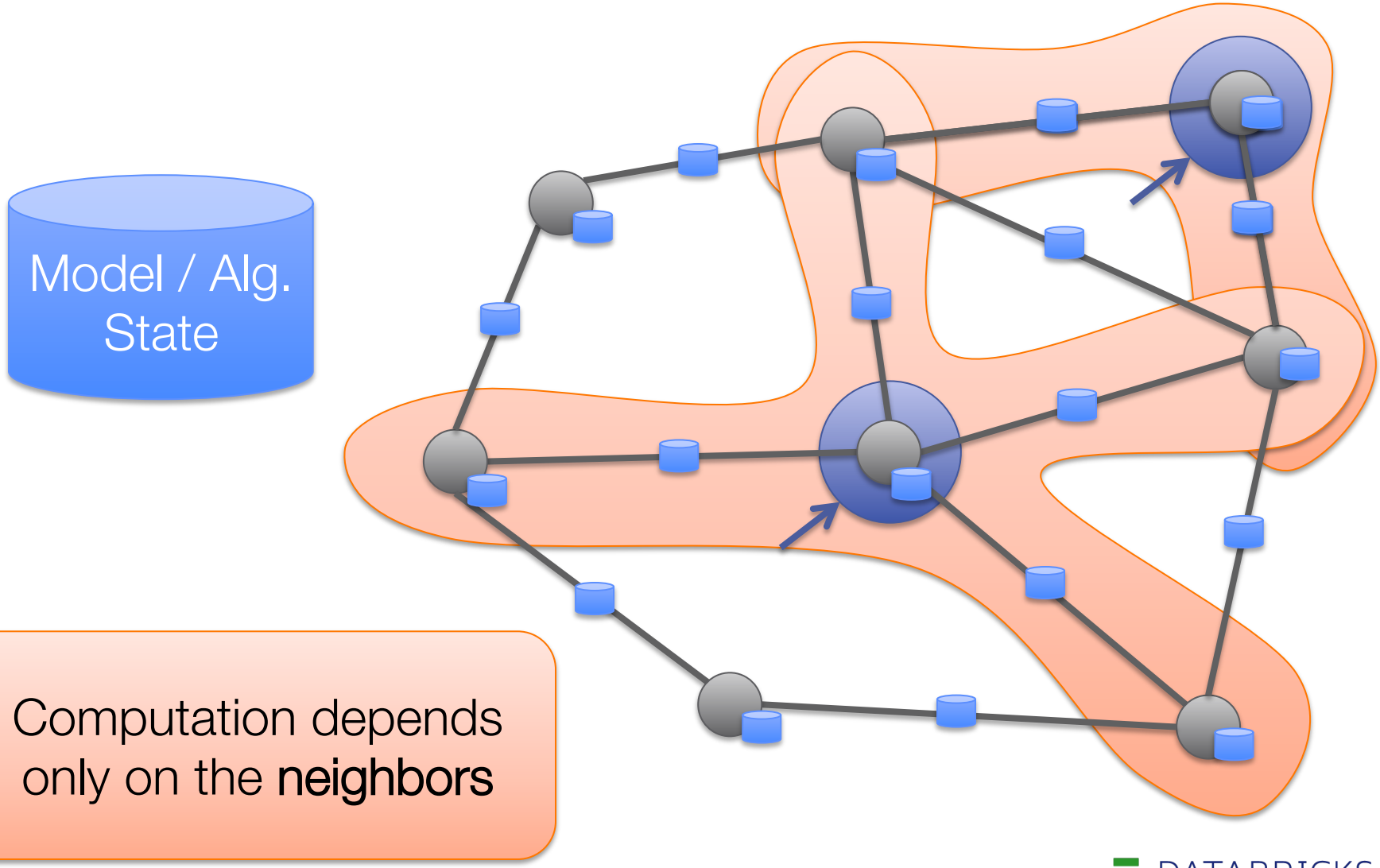


Single-Source Shortest Path

$$D[i] = 1 + \underset{j \in \mathrm{Nbrs}(i)}{\arg\min} D[j]$$



The Graph-Parallel Pattern



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

Expose specialized APIs to simplify graph programming.

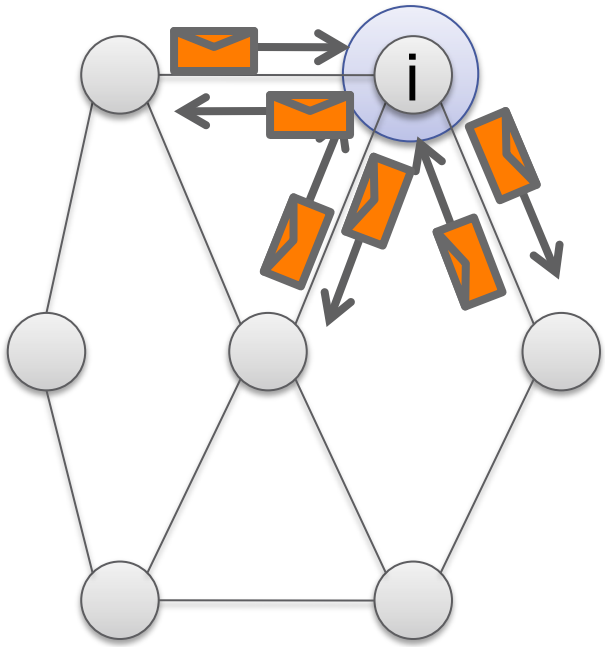
Exploit graph structure to achieve orders-of-magnitude performance gains over more general data-parallel systems.

DATABRICKS

Specialized API: Pregel

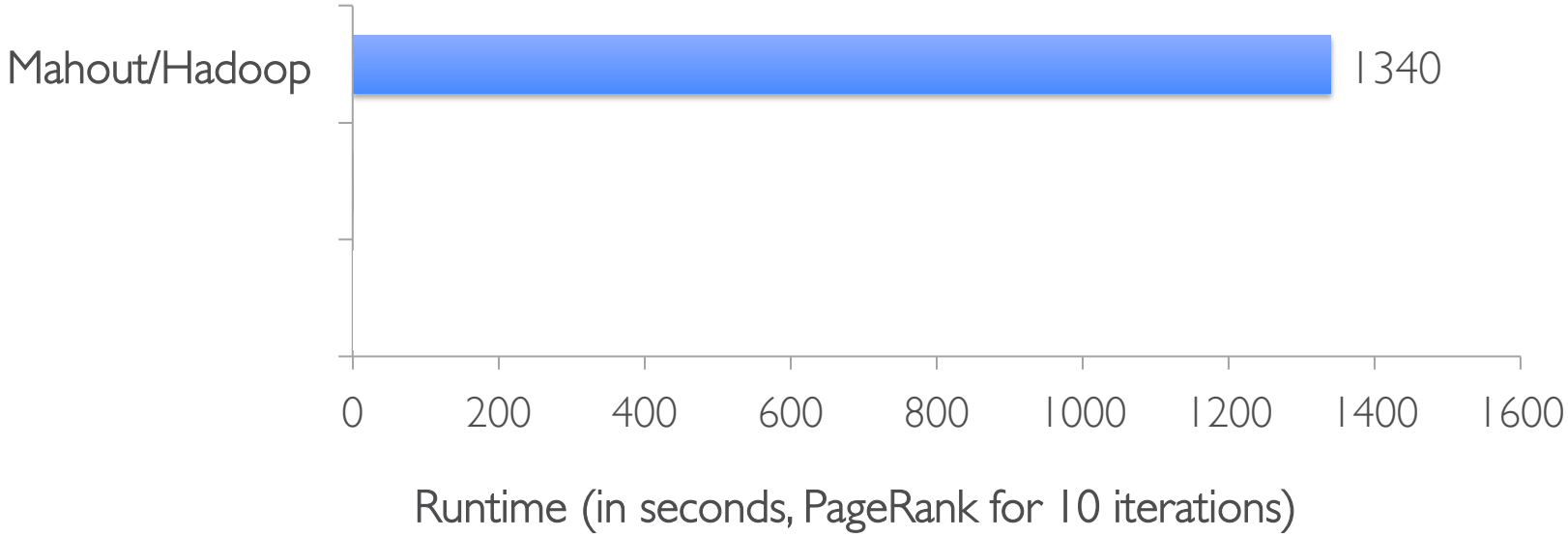
Vertex-Programs interact by sending **messages**.

```
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]

PageRank on LiveJournal Graph (69M edges)

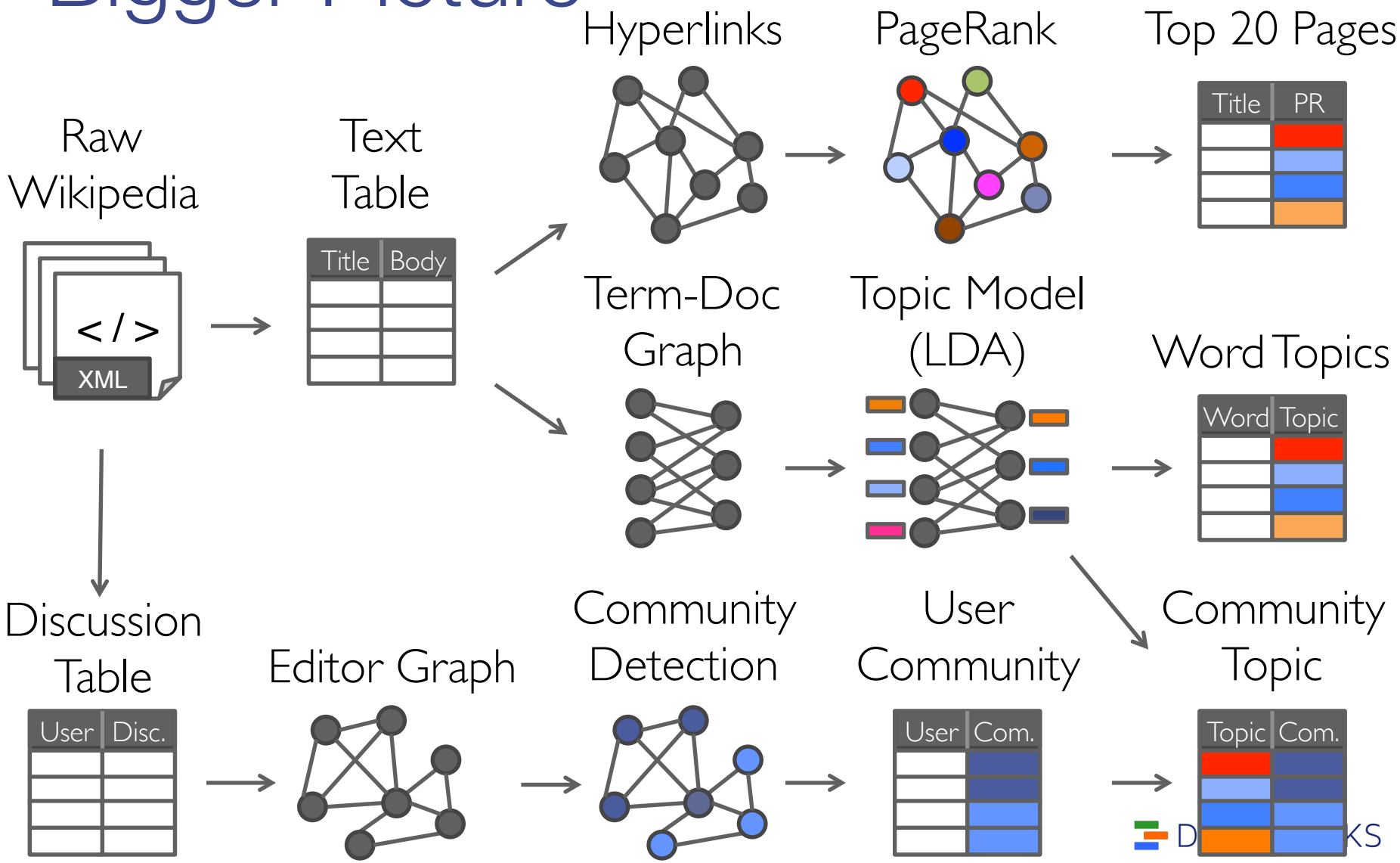


Spark is *4x faster* than Hadoop

GraphLab is *16x faster* than Spark

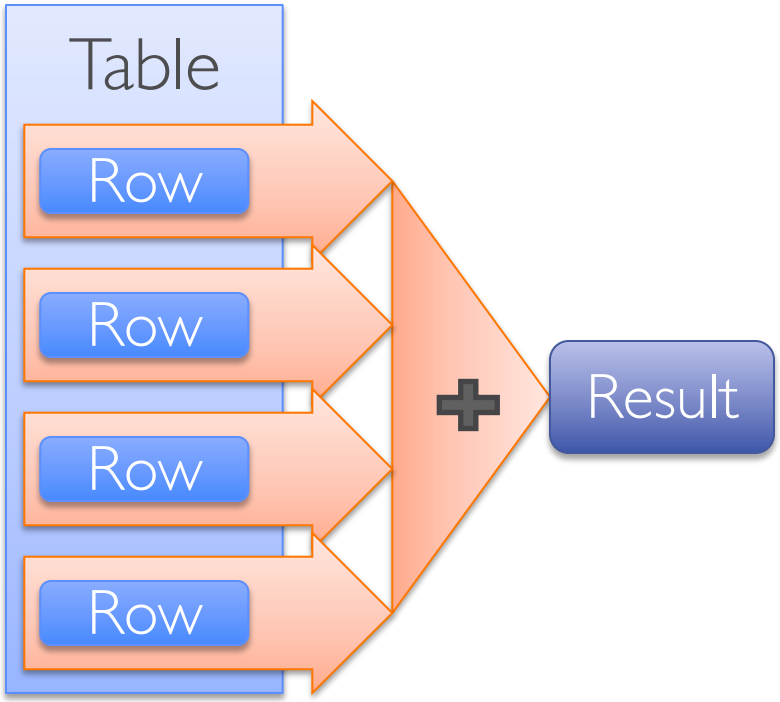
DATABRICKS

Specialized Systems Miss the Bigger Picture

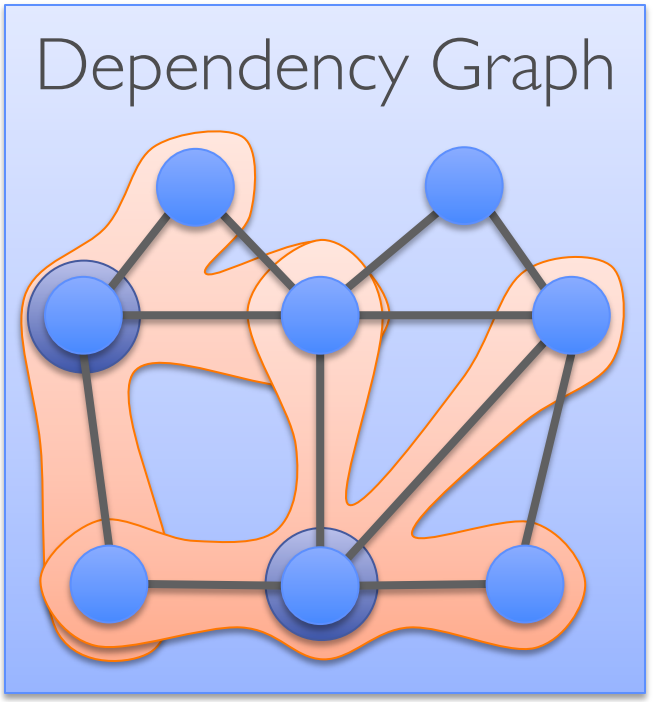


Separate Systems to Support Each View

Data-Parallel



Graph-Parallel



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

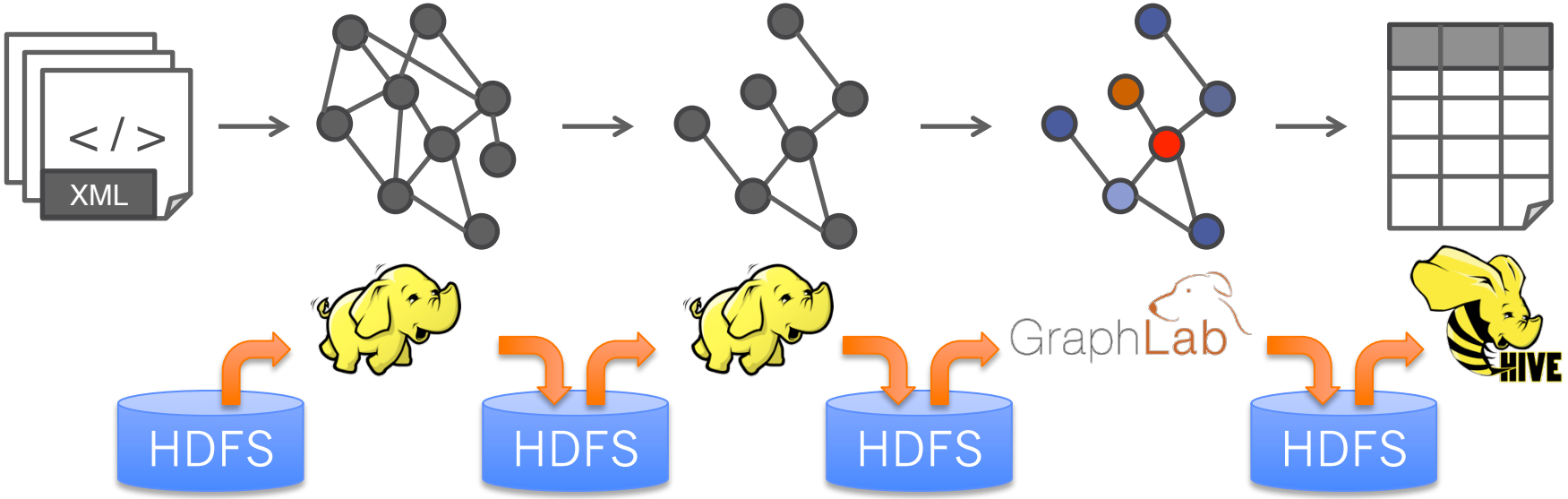
Difficult to Program and Use

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

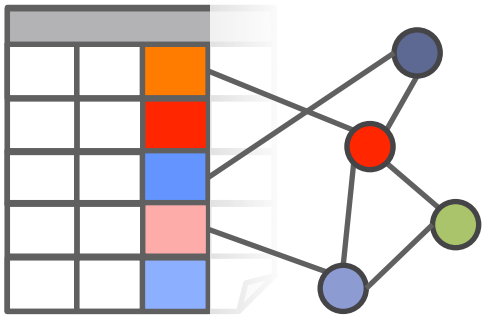


Limited reuse internal data-structures across stages

Solution: The GraphX Unified Approach

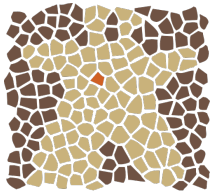
New API

Blurs the distinction between Tables and Graphs



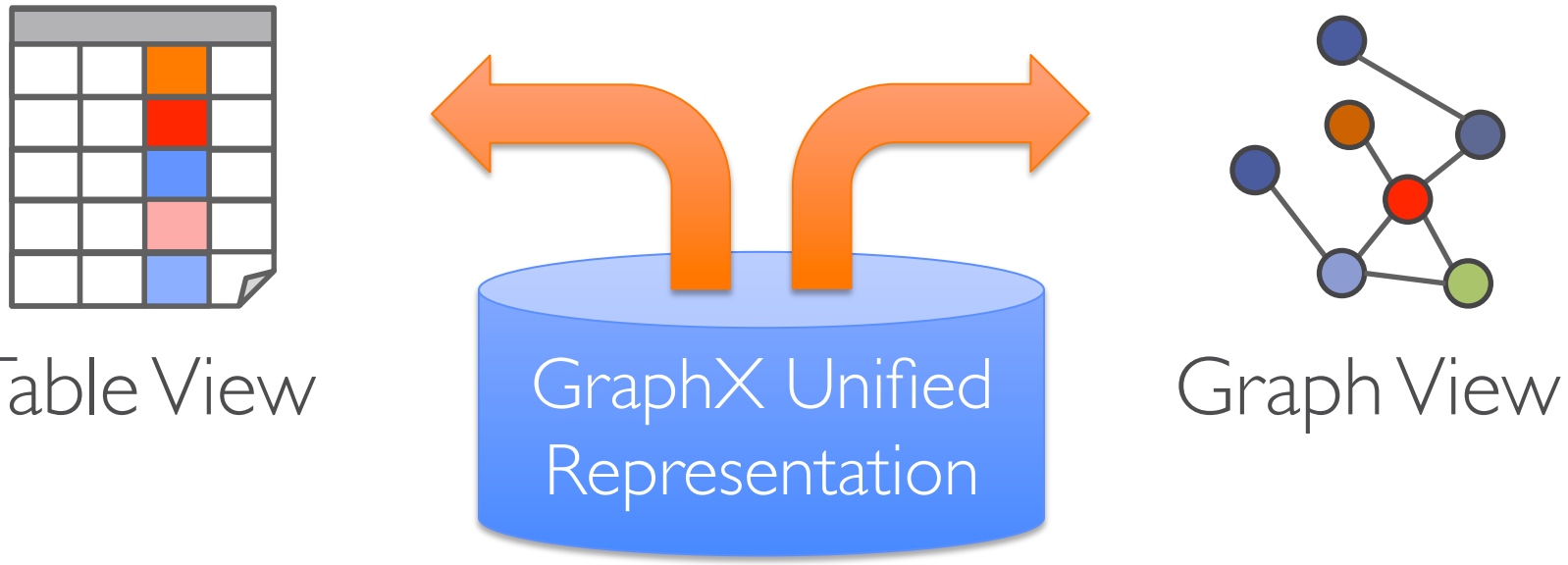
New Library

Embeds Graph-Parallel model in Spark



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

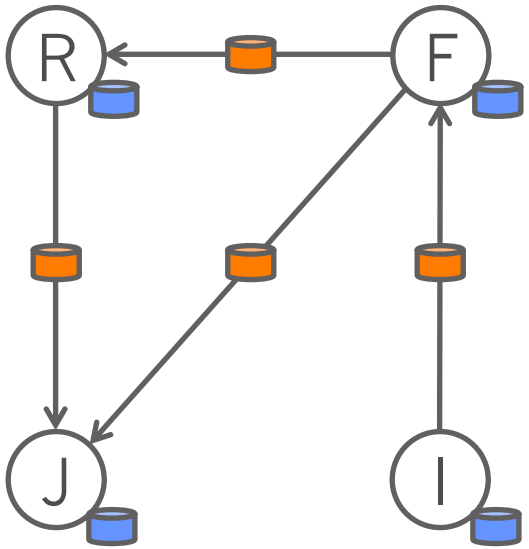
Tables and Graphs are composable views of the same physical data



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 Table

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

Edge Table

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

Table Operators

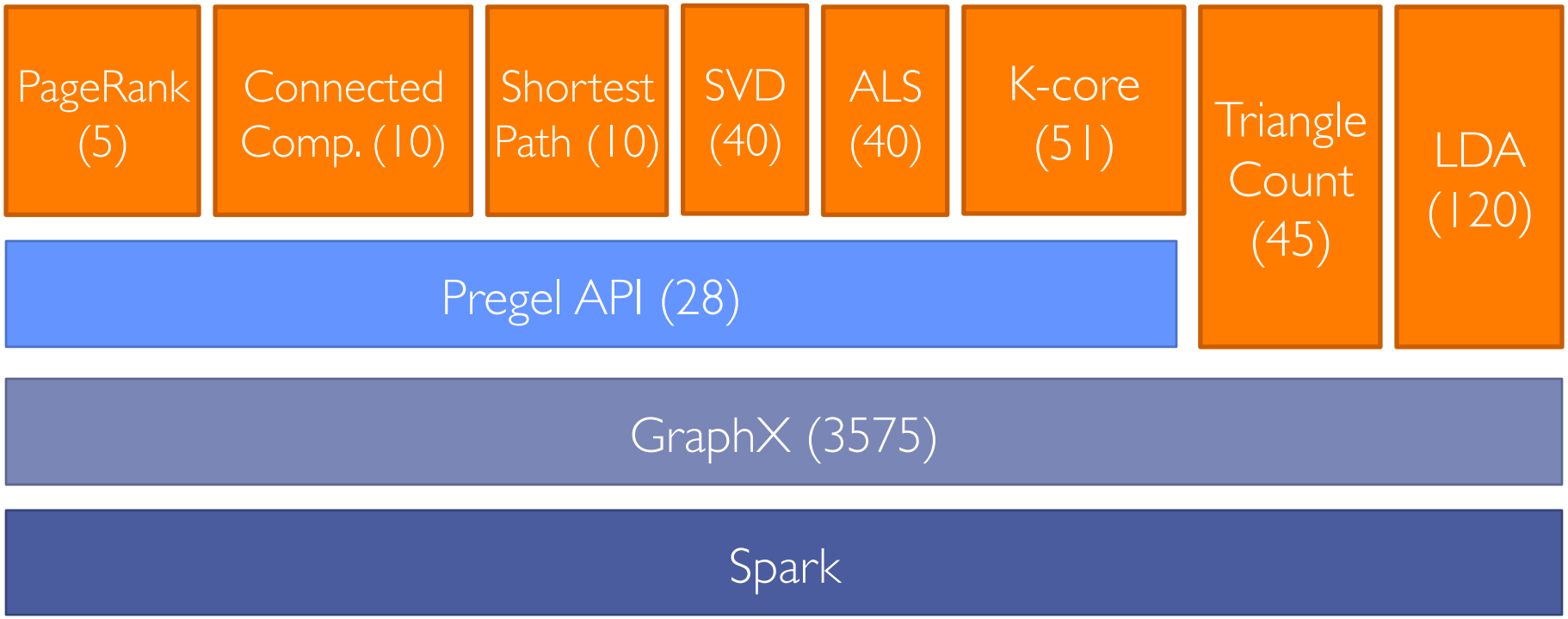
Table (RDD) operators are inherited from Spark:

```
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) ]
  // Computation ------------------------------
  def mrTriplets(mapF: (Edge[V,E]) => List[(Id, T)],
                 reduceF: (T, T) => T): Graph[T, E]

  // Convenience functions ----------------------------
  def mapV(m: (Id, V) => T ): Graph[T,E]
  def mapE(m: Edge[V,E] => T ): Graph[V,T]
  def joinV(tbl: Table [(Id, T)]): Graph[(V, T), E ]
  def joinE(tbl: Table [(Id, Id, T)]): Graph[V, (E, T)]
  def reverse: Graph[V, E]
  def subgraph(pV: (Id, V) => Boolean,
               pE: Edge[V,E] => Boolean): Graph[V,E]
}
```

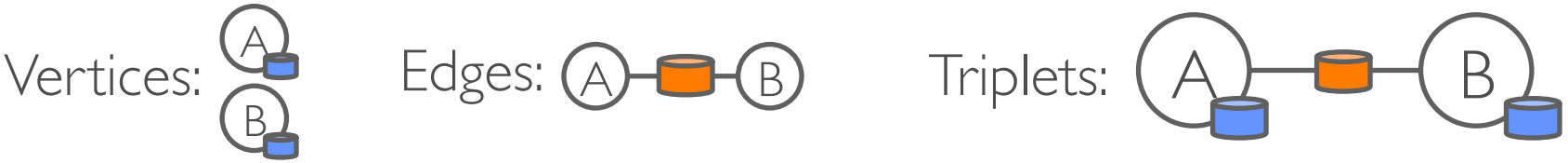
The GraphX Stack (Lines of Code)



Triplets Join Vertices and Edges

The *triplets* operator joins vertices and edges:

```
SELECT src.Id, dst.Id, src.attr, e.attr, dst.attr
FROM edges AS e JOIN vertices AS src, vertices AS dst
ON e.srcId = src.Id AND e.dstId = dst.Id
```

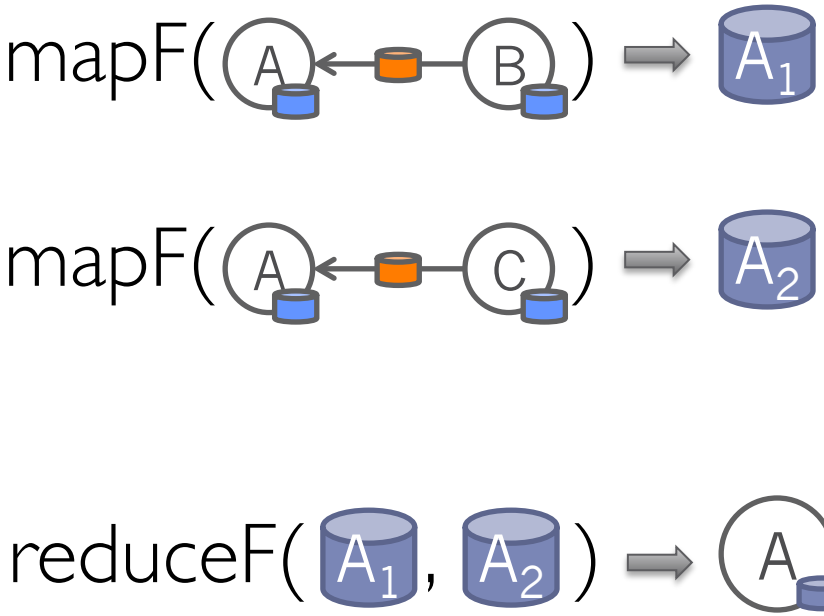


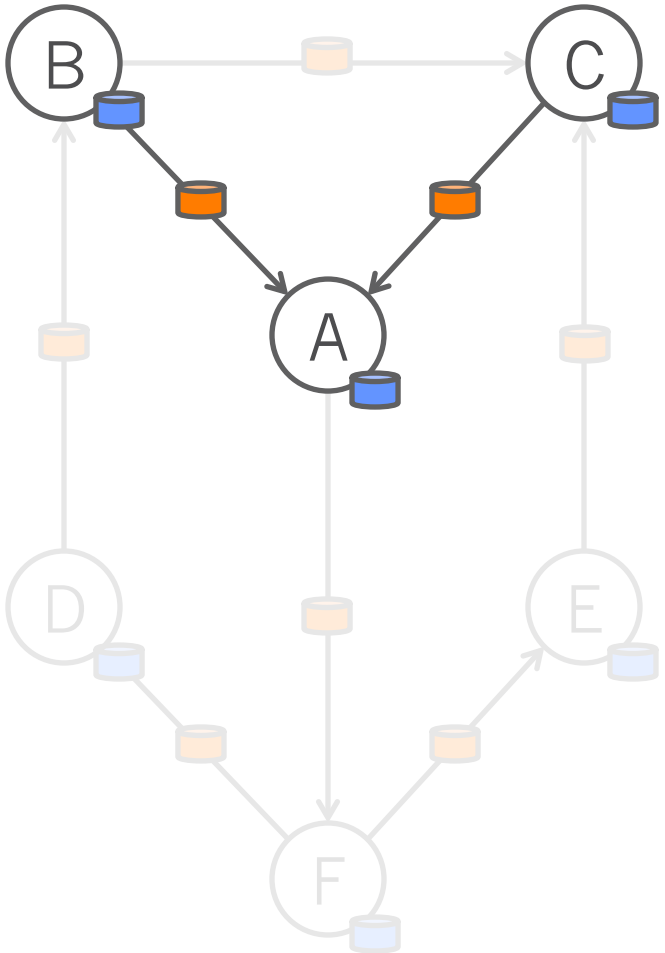
The *mrTriplets* operator sums adjacent triplets.

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

Map Reduce Triplets

Map-Reduce for each vertex

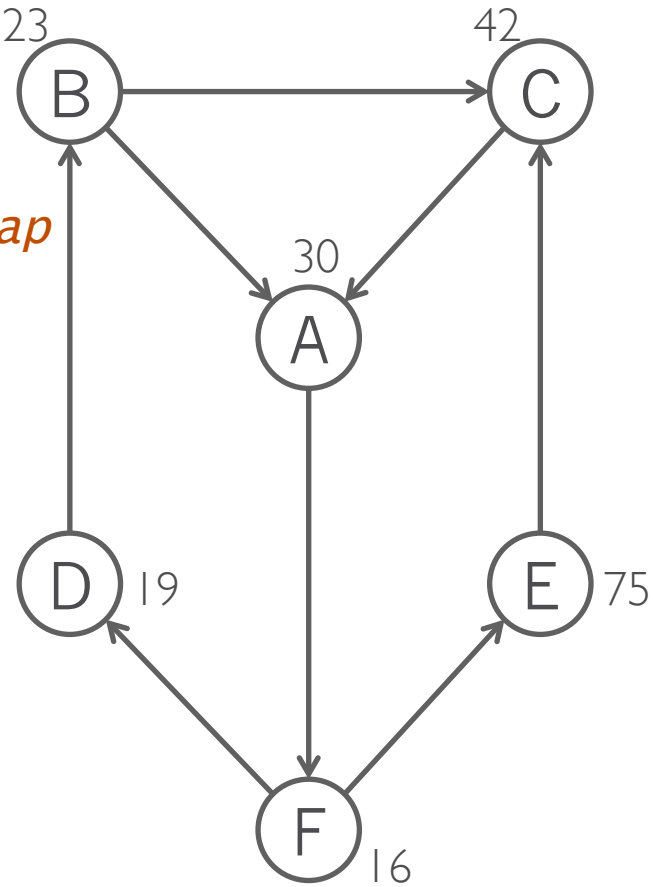




Example: Oldest Follower

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
```



ank in GraphX

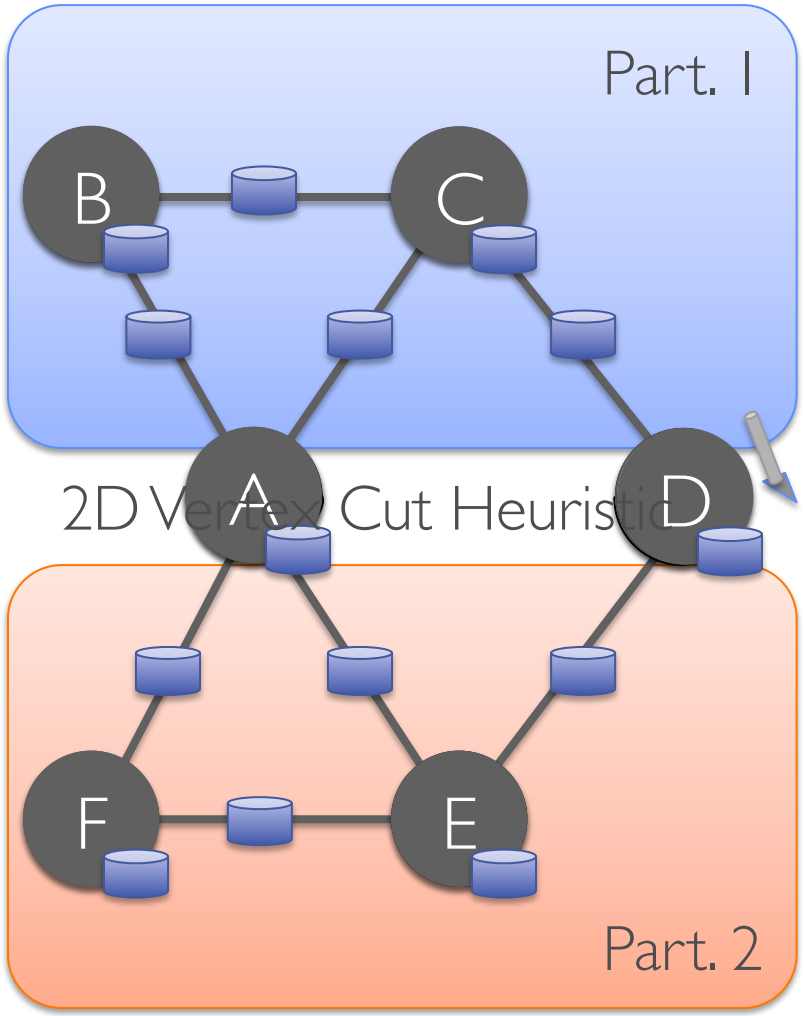
Graphs are first-class objects

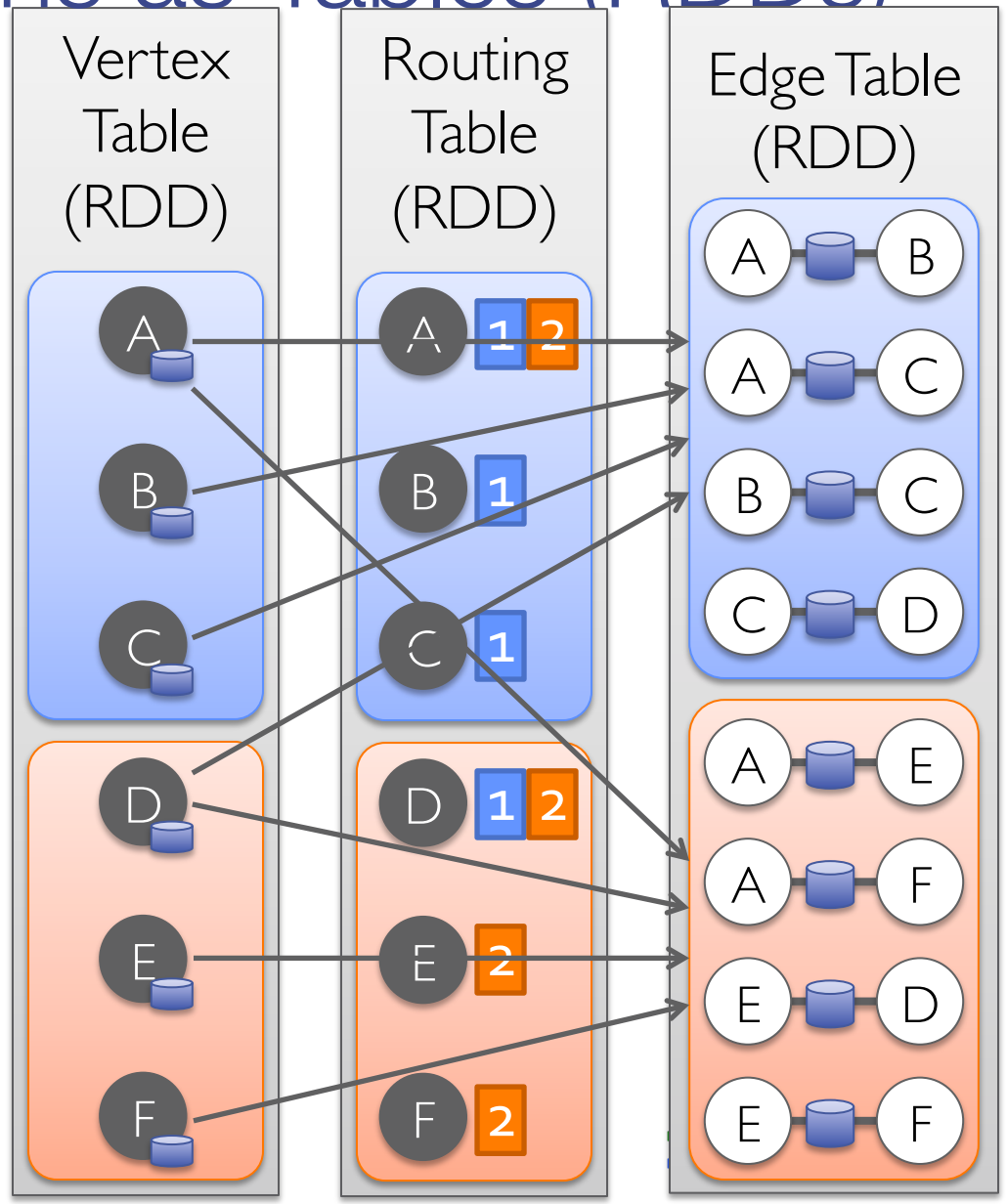
```
// Loa  and initialize the graph
val graph = GraphLoader.edgeListFile("hdfs://web.txt")
val prGraph = graph.joinVertices(graph.outDegrees)
// Implement and Run PageRank
val pageRank =
  prGraph.pregel(initialMessage = 0.0, iter = 10)(
    (oldV, msgSum) => 0.15 + 0.85 * msgSum,
    triplet => triplet.src.pr / triplet.src.deg,
    (msgA, msgB) => msgA + msgB)
// Get the top 20 pages
pageRank.vertices.top(20)(Ordering.by(_._2)).foreach(println)
```

Implementation

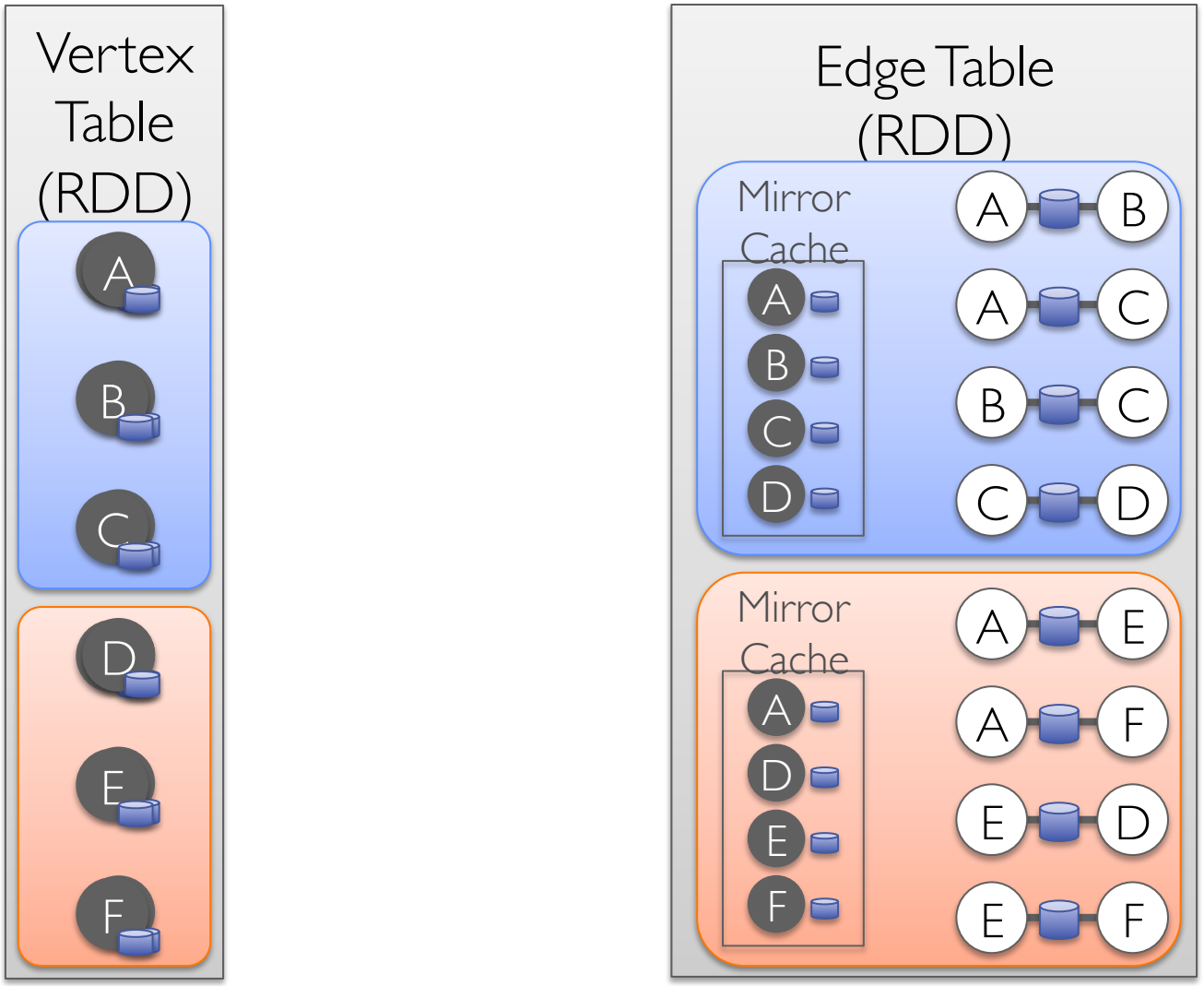
Distributed Graphs as Tables (RDDs)

Property Graph

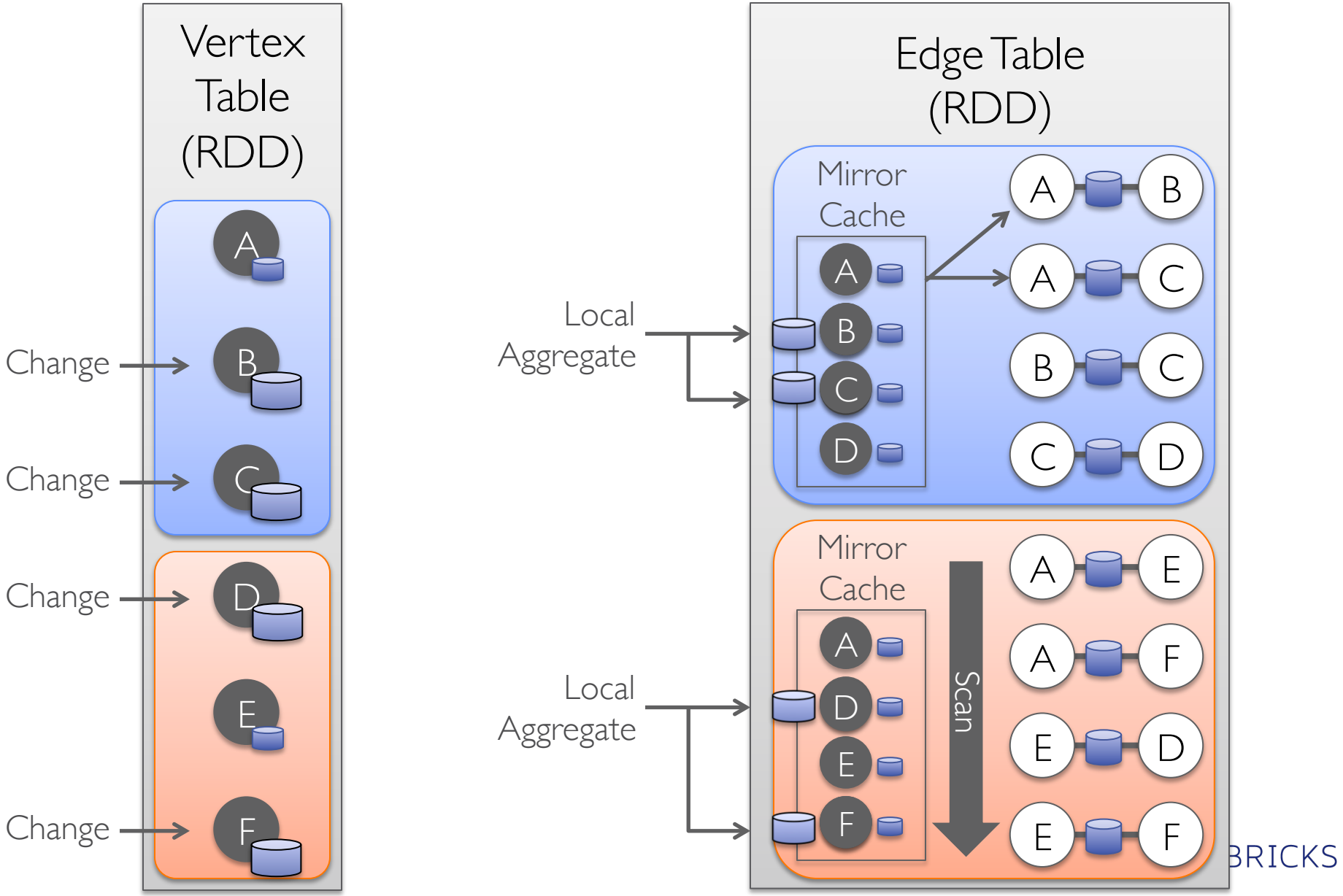




Caching for Iterative mrTriplets



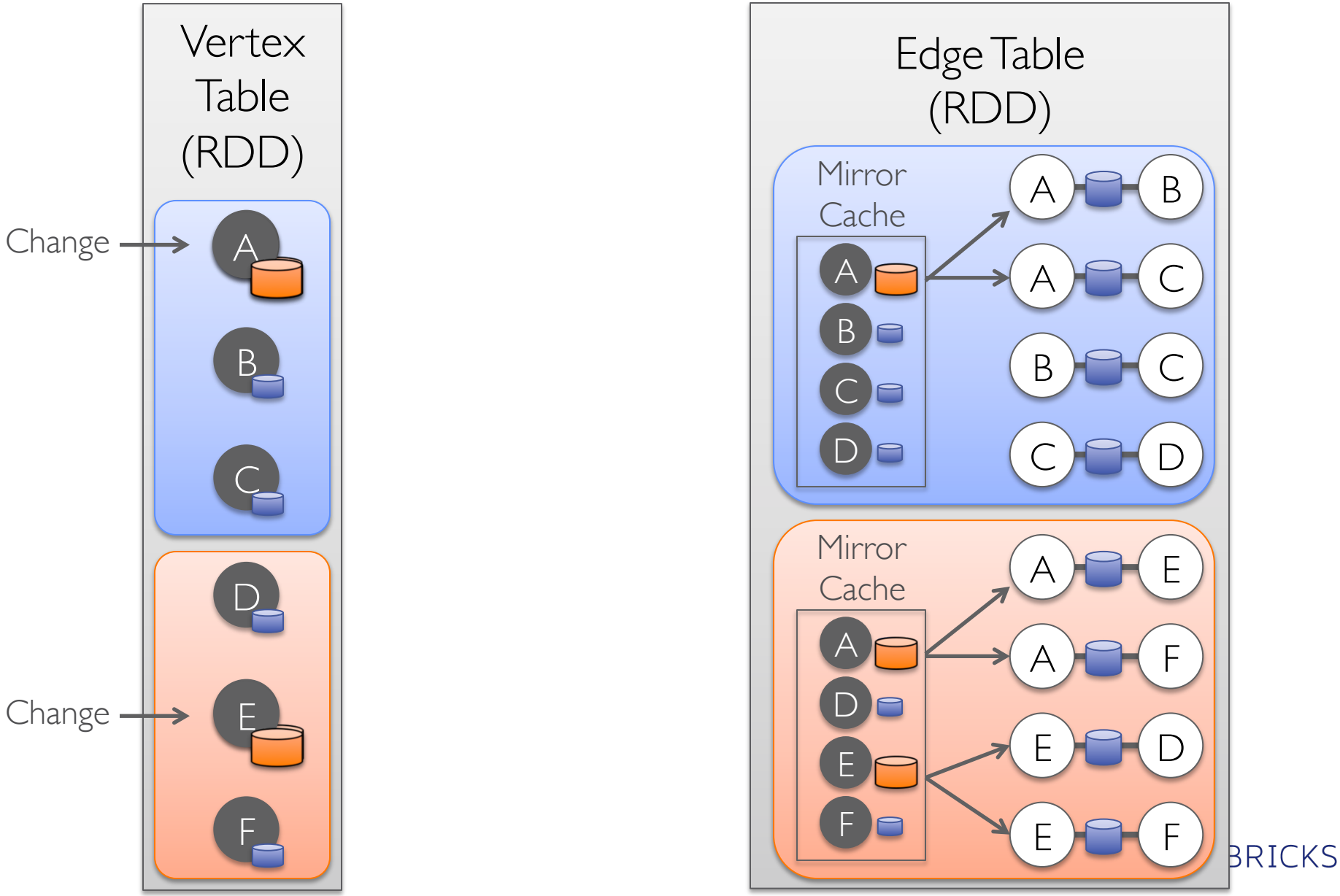
mrTriplets Execution



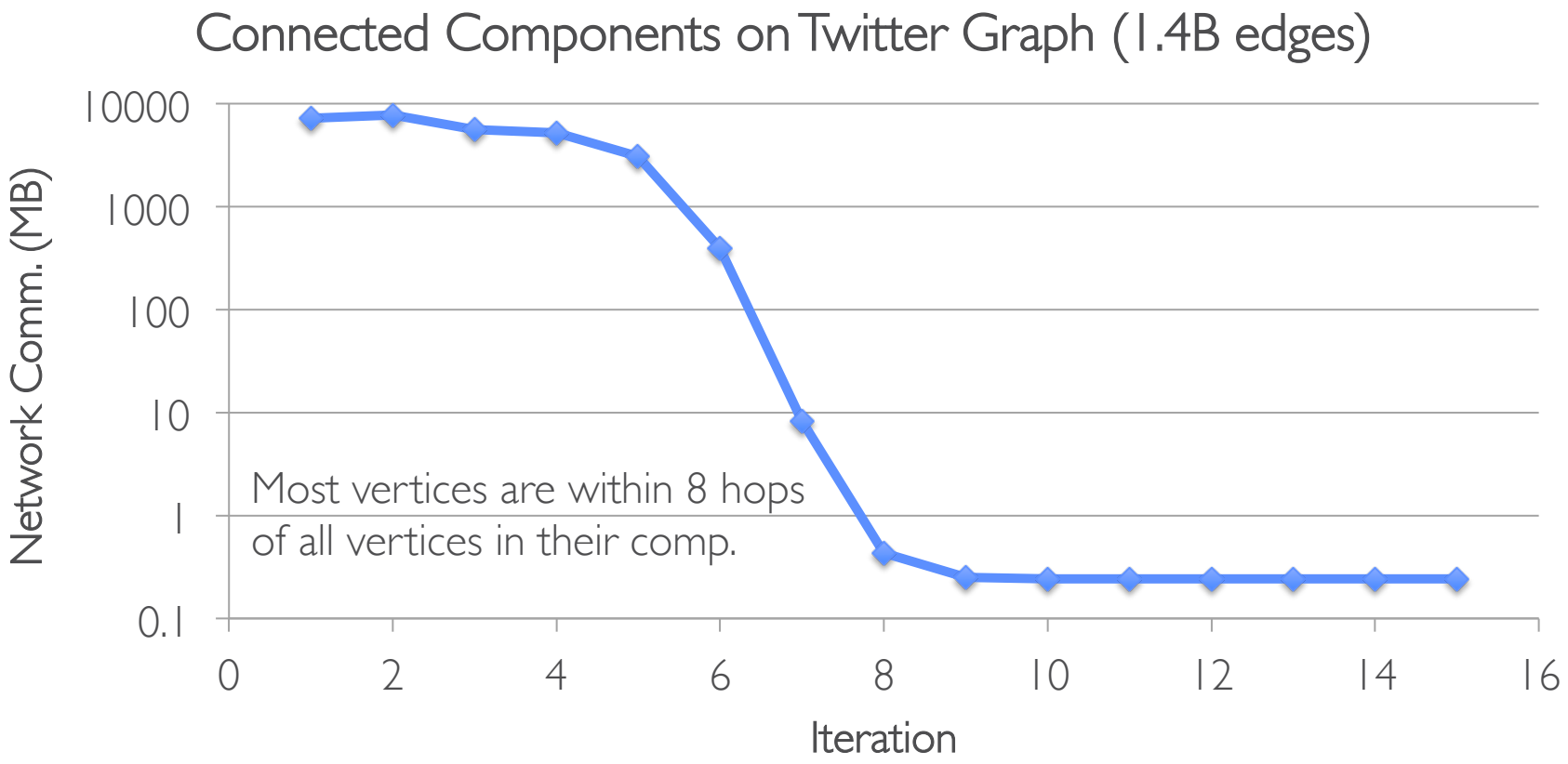
Optimizations

1. Incremental Updates to Mirror Caches
2. Join Elimination
3. Index Scanning for Active Sets
4. Local Vertex and Edge Indices
5. Index and Routing Table Reuse

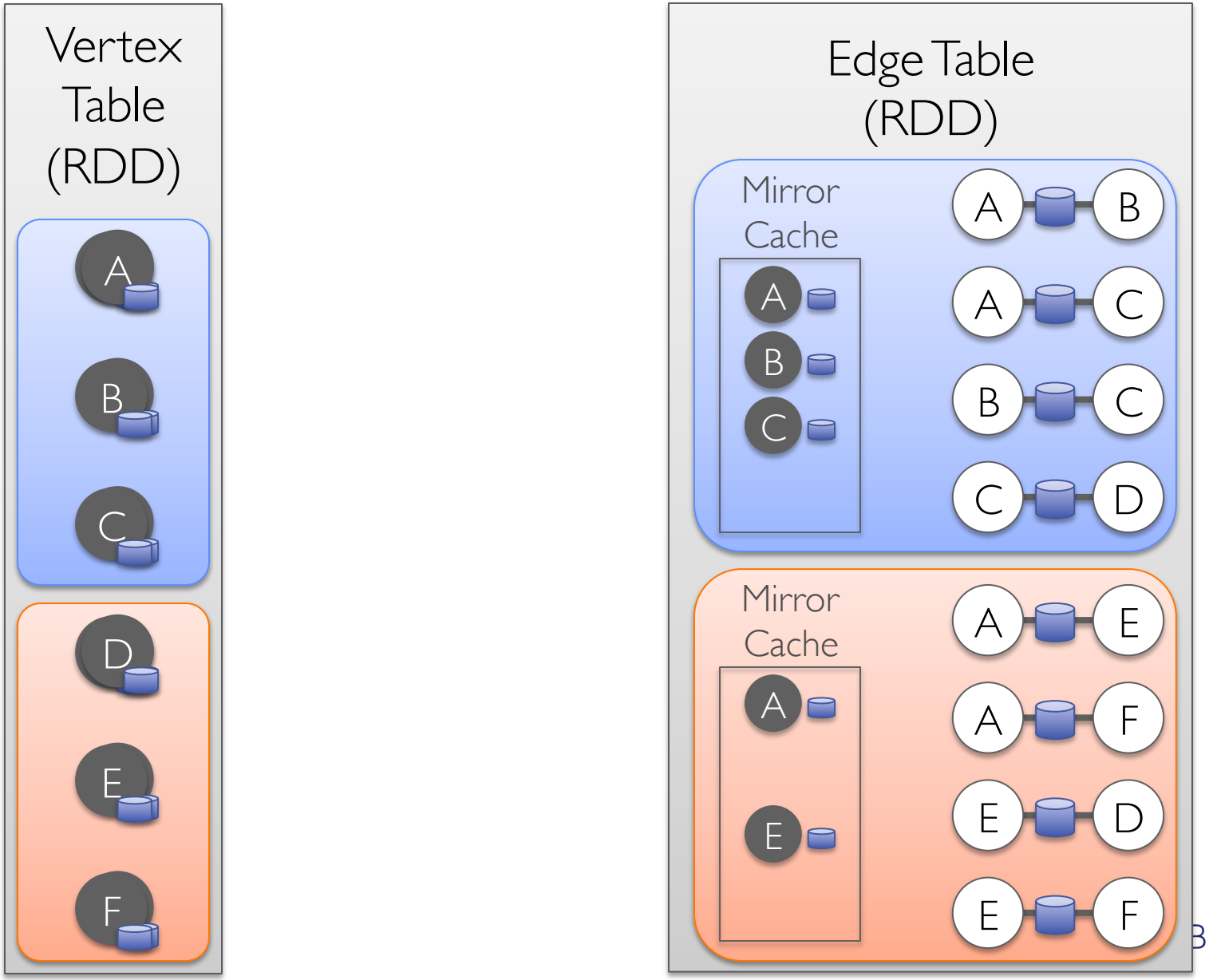
1. Incremental Updates for Mirror Caches



Reduction in Communication Due to Cached Updates



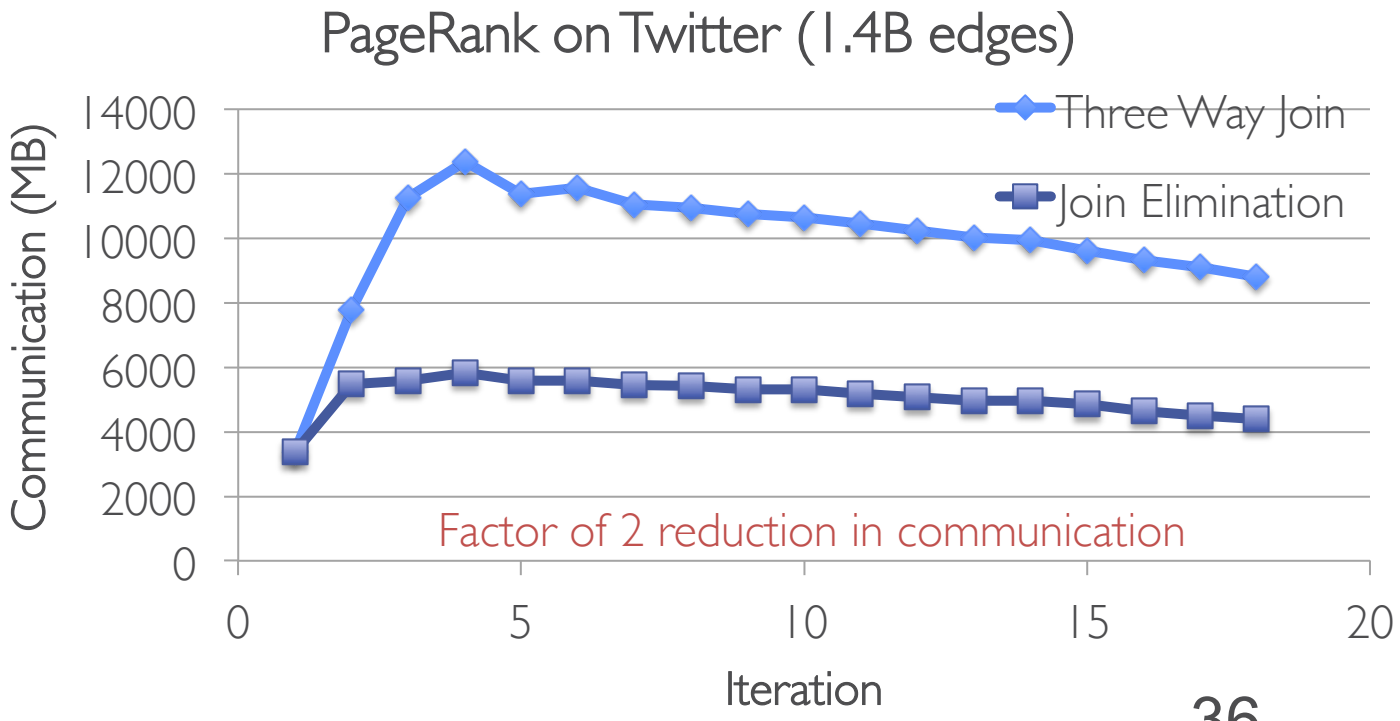
2. Join Elimination



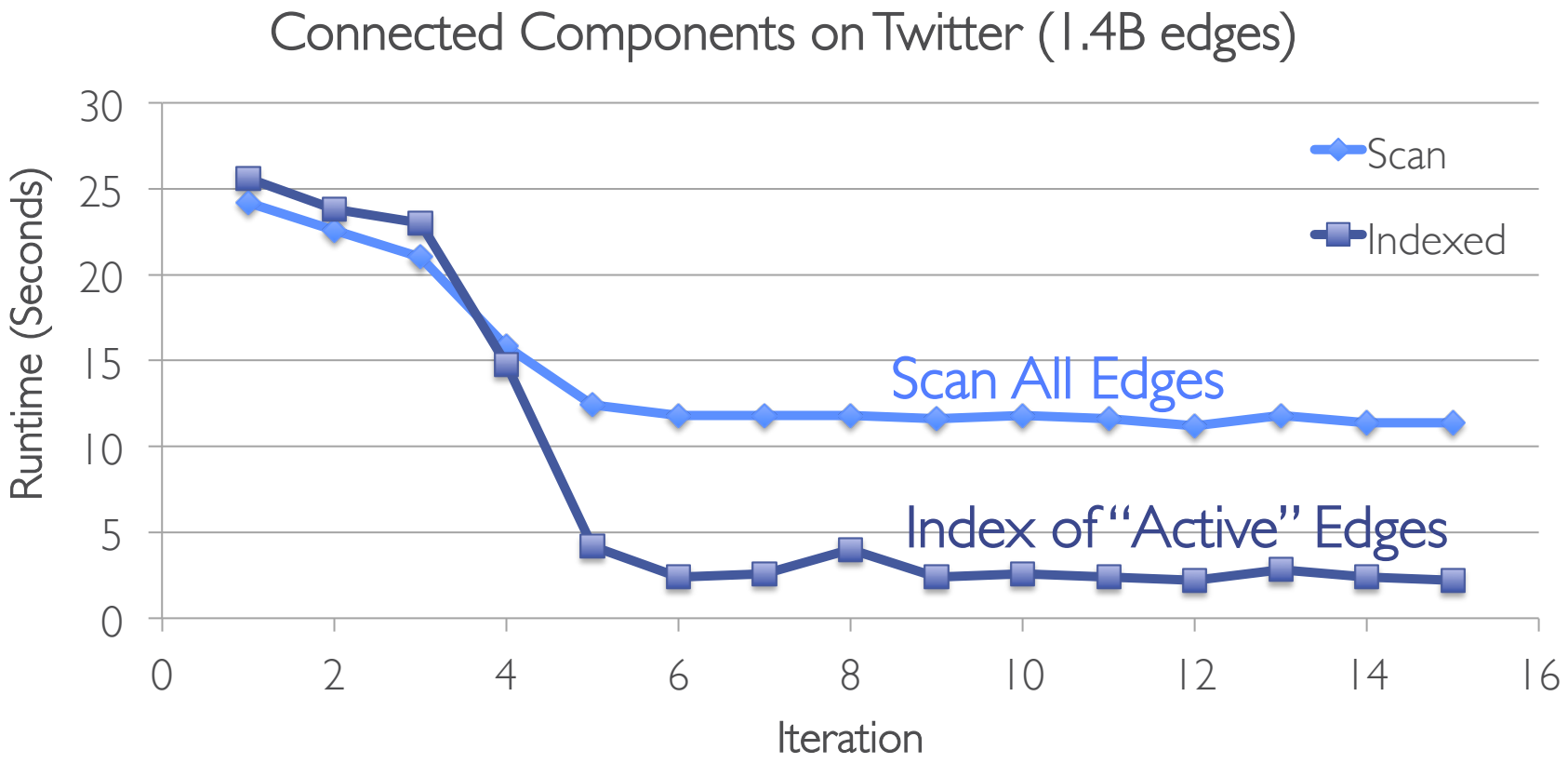
Reduction in Communication Due to Join Elimination

UDF byte code inspection for join elimination:

- Identify and bypass joins for unused triplets fields
- *Example:* PageRank only accesses source attribute



3. Index Scanning for Active Sets



DATABRICKS

Additional Query Optimizations

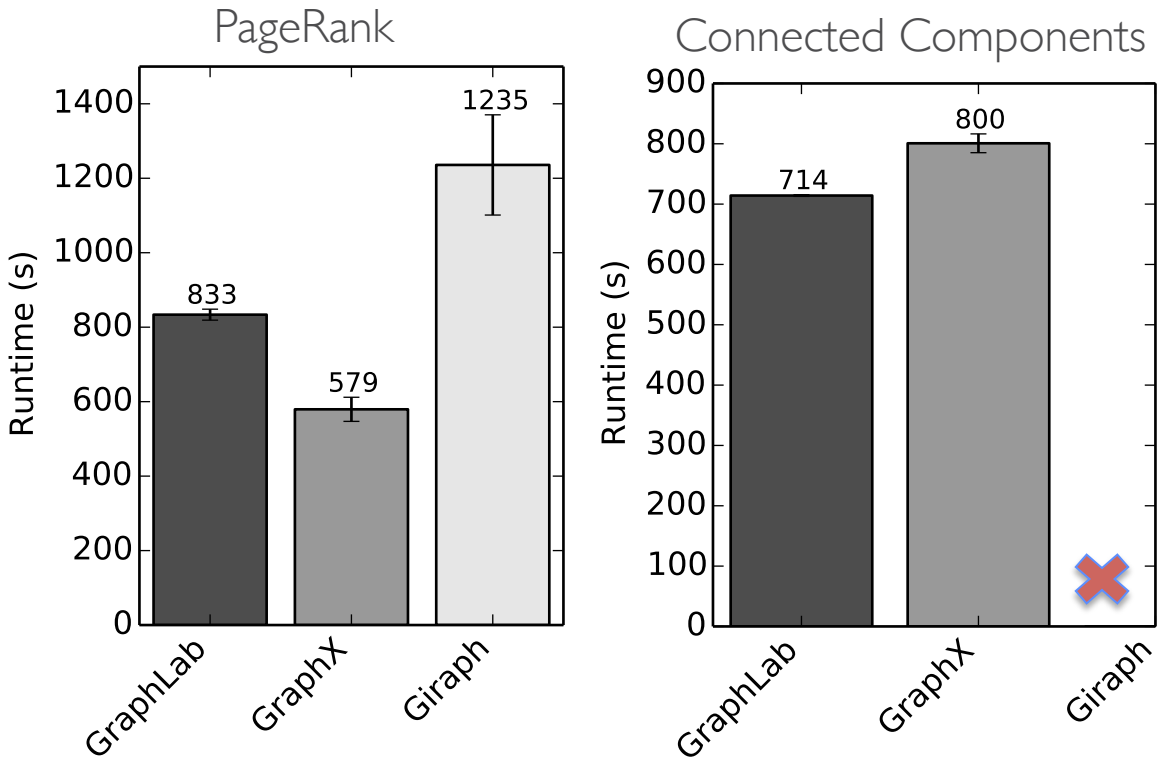
Indexing and Bitmaps:

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

Substantial Index and Data Reuse:

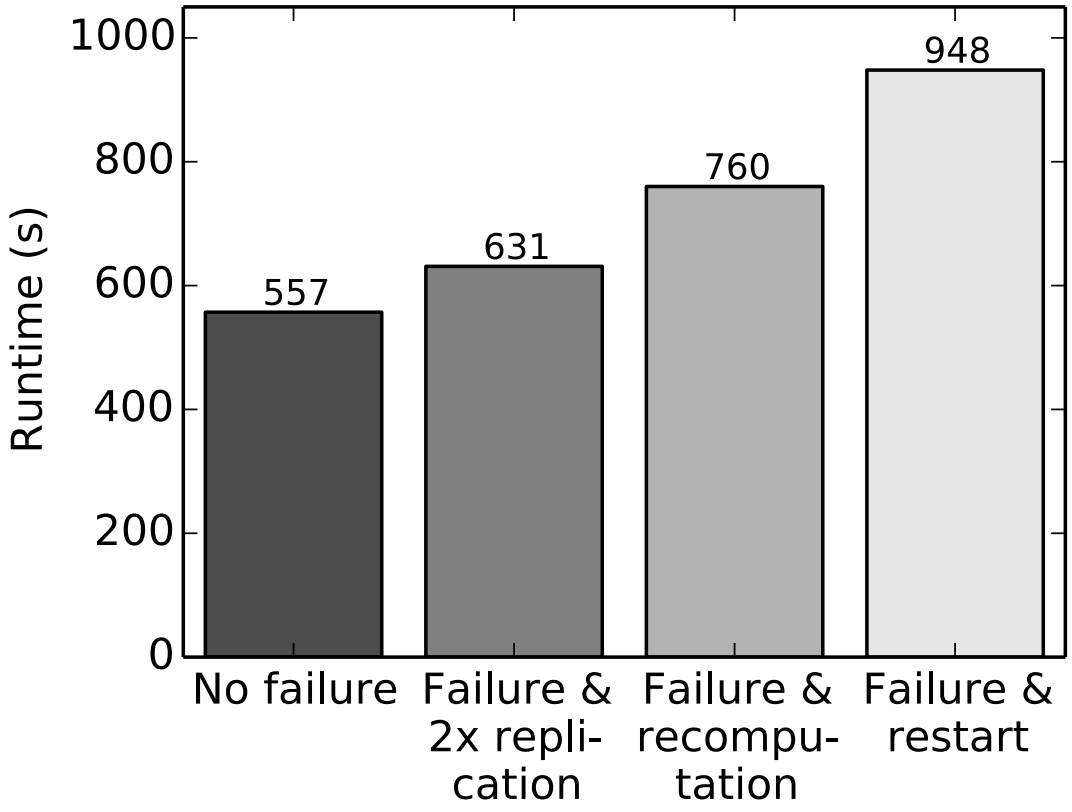
- *Split Hash Table*: reuse key sets across hash tables
- Reuse routing tables across graphs and sub-graphs
- Reuse edge adjacency information and indices

Multi-System Comparison



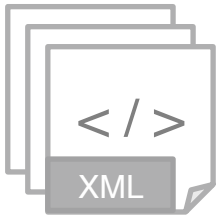
uk-2007-05 web graph (3.7B edges)

Fault Tolerance



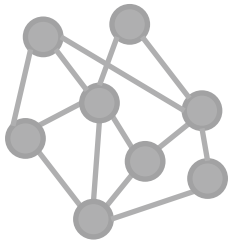
GraphX Training

Raw Wikipedia

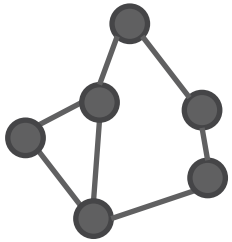


Text Table

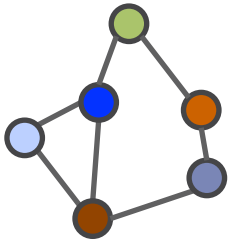
Hyperlinks



Berkeley subgraph



PageRank



Top 20 Pages

DATABRICKS

Demo

Future Directions

- Computation on time-varying graphs
- Graph serving
- Operating on compressed graphs

Thanks!

http://spark.apache.org/graphx

{jegonzal, rxin, ankurd, crankshaw}@eecs.berkeley.edu