Data-Intensive Distributed Computing

CS 451/651 431/631 (Winter 2018)

Part 8: Analyzing Graphs, Redux (1/2)

March 20, 2018

Jimmy Lin
David R. Cheriton School of Computer Science
University of Waterloo

These slides are available at http://lintool.github.io/bigdata-2018w/

Graph Algorithms, again?

(srsly?)

What makes graphs hard? Fun!

Irregular structure

Fun with data structures!

Irregular data access patterns

Fun with architectures!

Iterations

Fun with optimizations!

Characteristics of Graph Algorithms

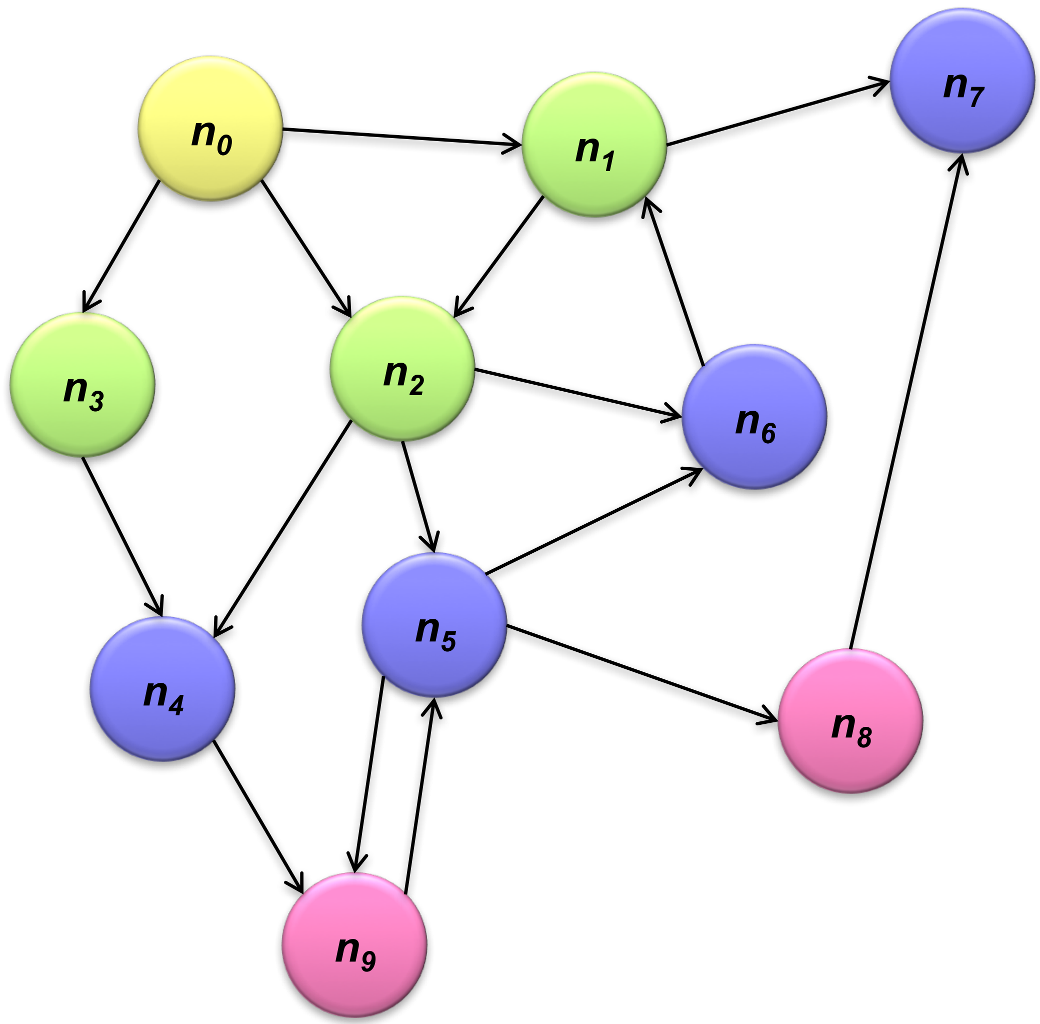
Parallel graph traversals

Local computations

Message passing along graph edges

Iterations

Visualizing Parallel BFS

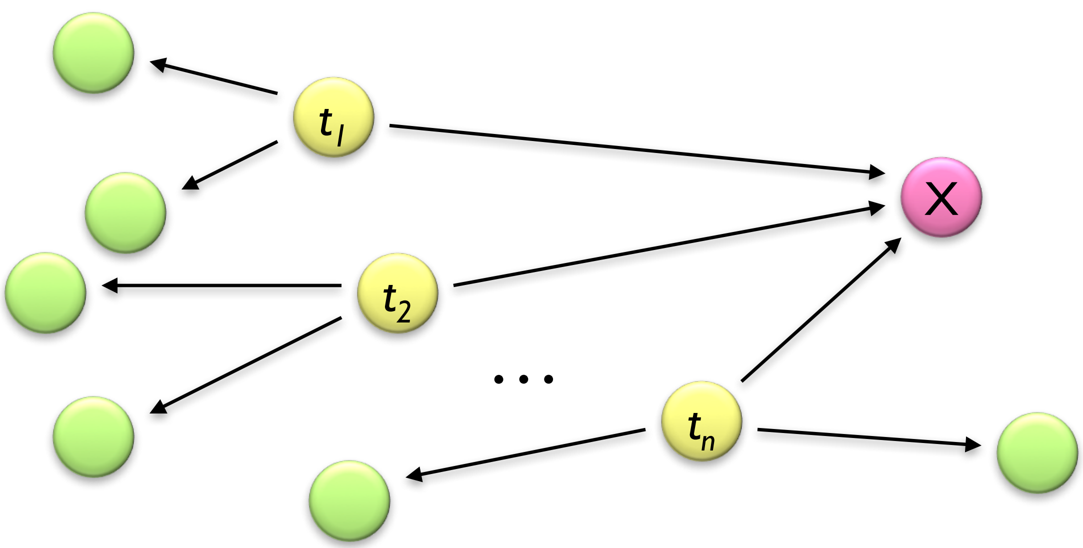


PageRank: Defined

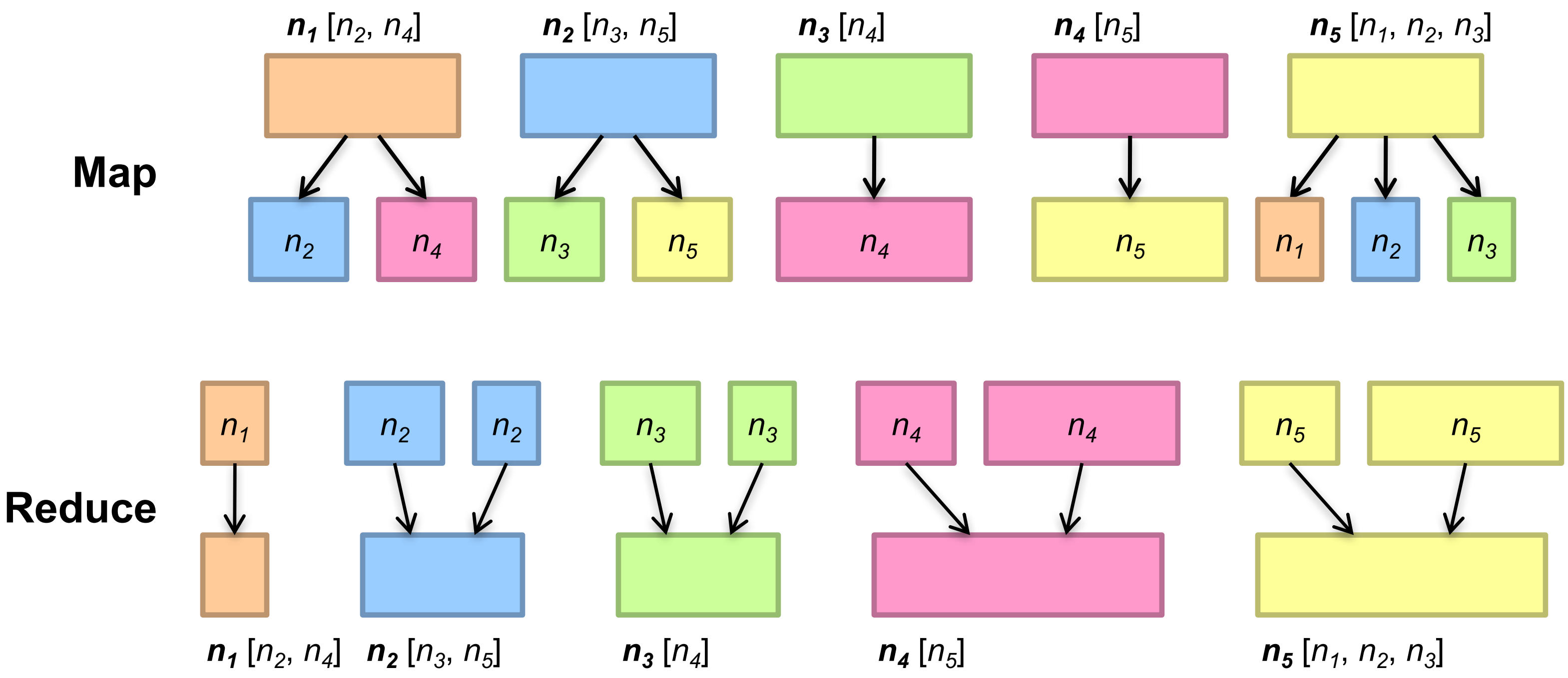
Given page x with inlinks $t_1 \ldots t_n$, where

- $C(t)$ is the out-degree of t
- α is probability of random jump
- N is the total number of nodes in the graph

$$PR(x) = \alpha \left(\frac{1}{N} \right) + (1 - \alpha) \sum_{i=1}^{n} \frac{PR(t_i)}{C(t_i)}$$



PageRank in MapReduce



PageRank vs. BFS

	PageRank	BFS
Map	PR/N	d+1
Reduce	sum	min

Characteristics of Graph Algorithms

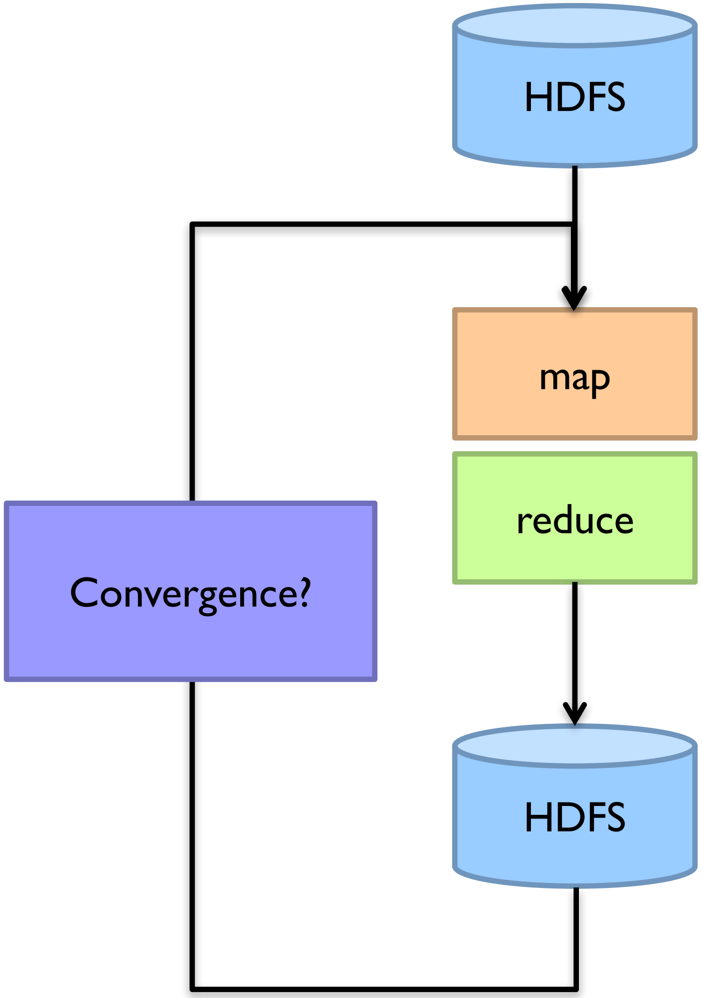
Parallel graph traversals

Local computations

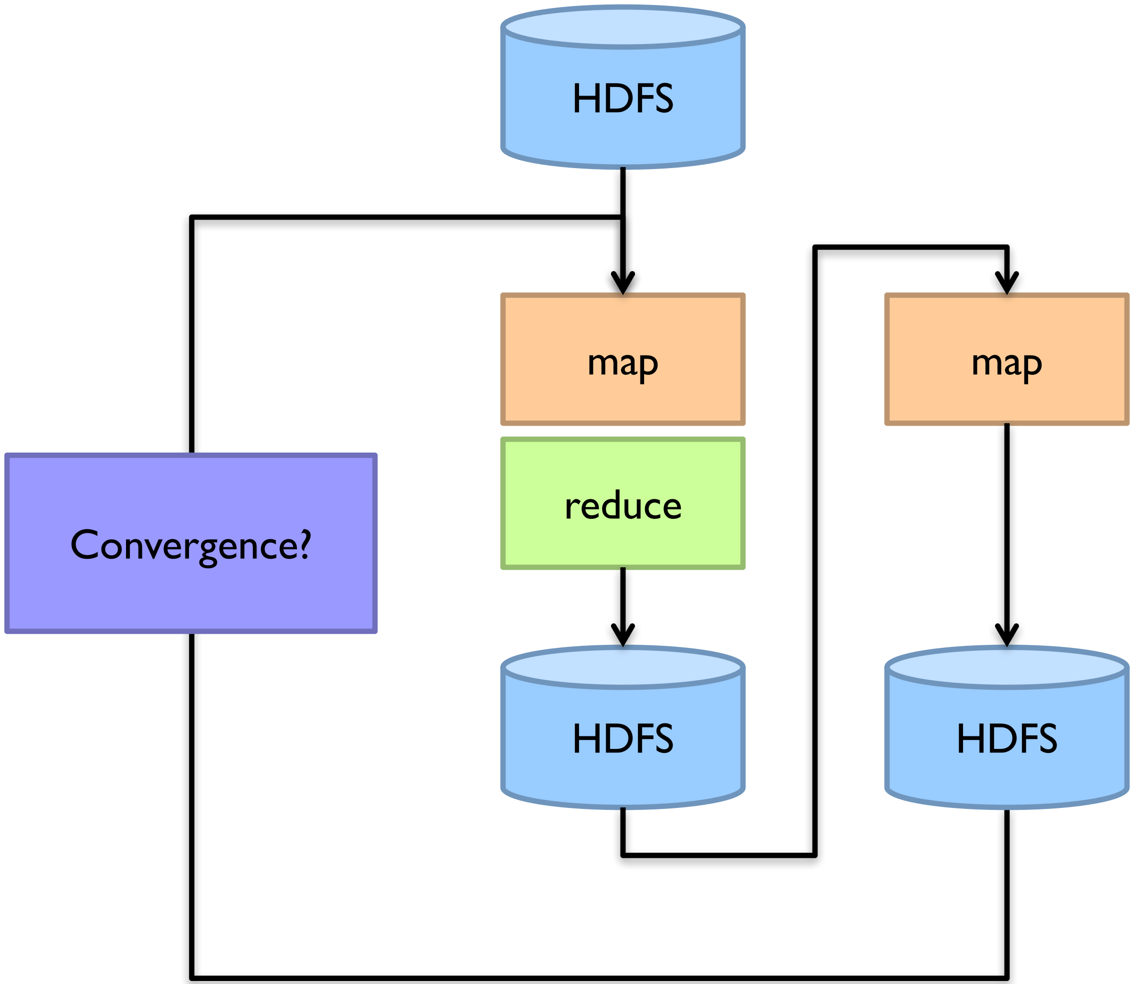
Message passing along graph edges

Iterations

BFS



PageRank



MapReduce Sucks

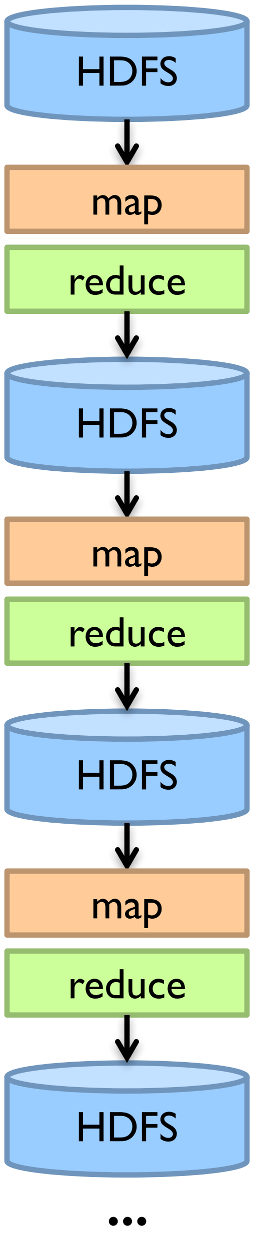
Hadoop task startup time

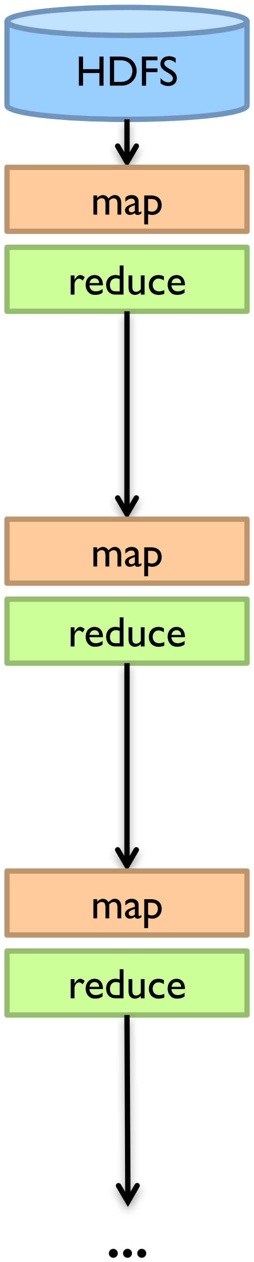
Stragglers

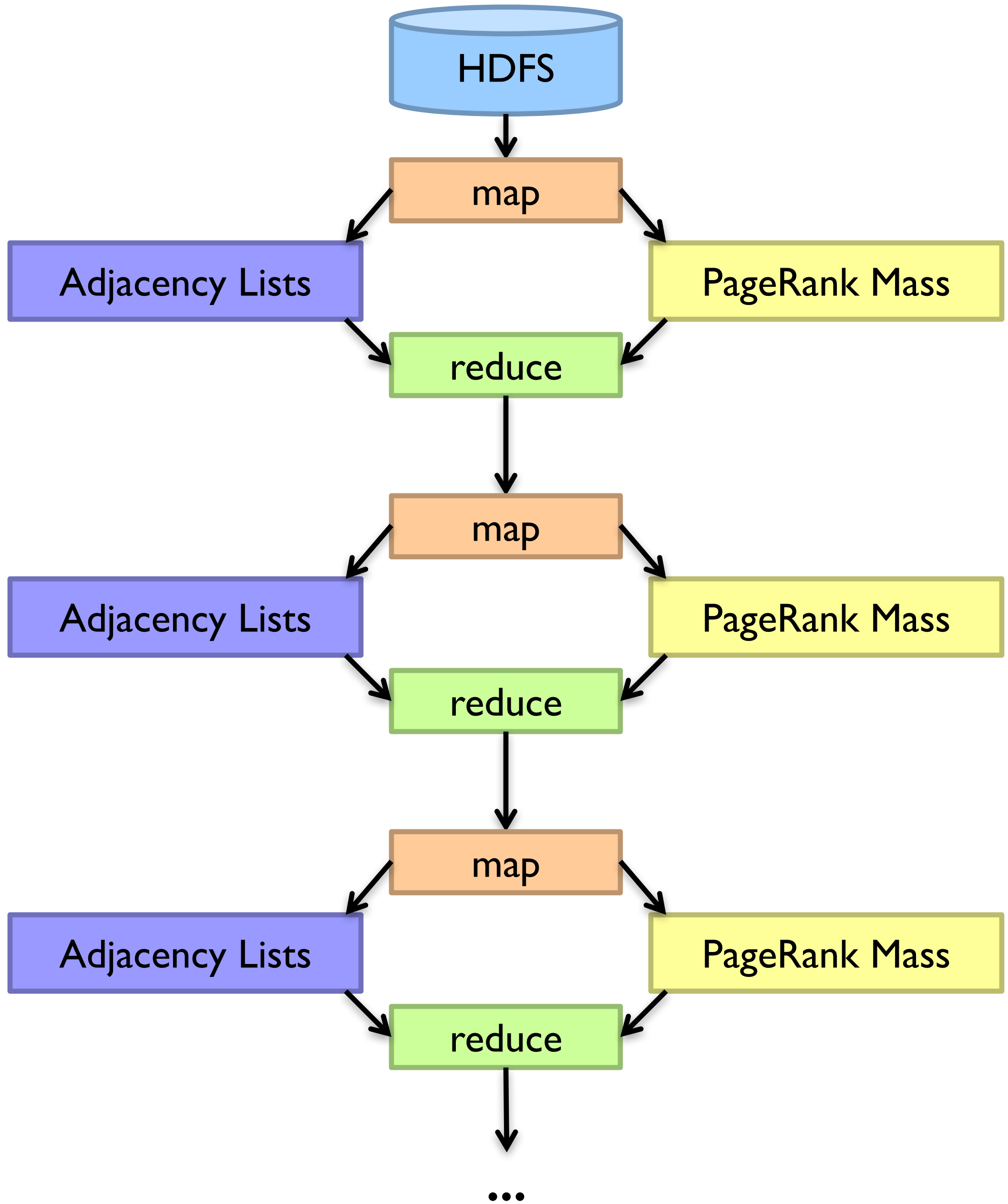
Needless graph shuffling

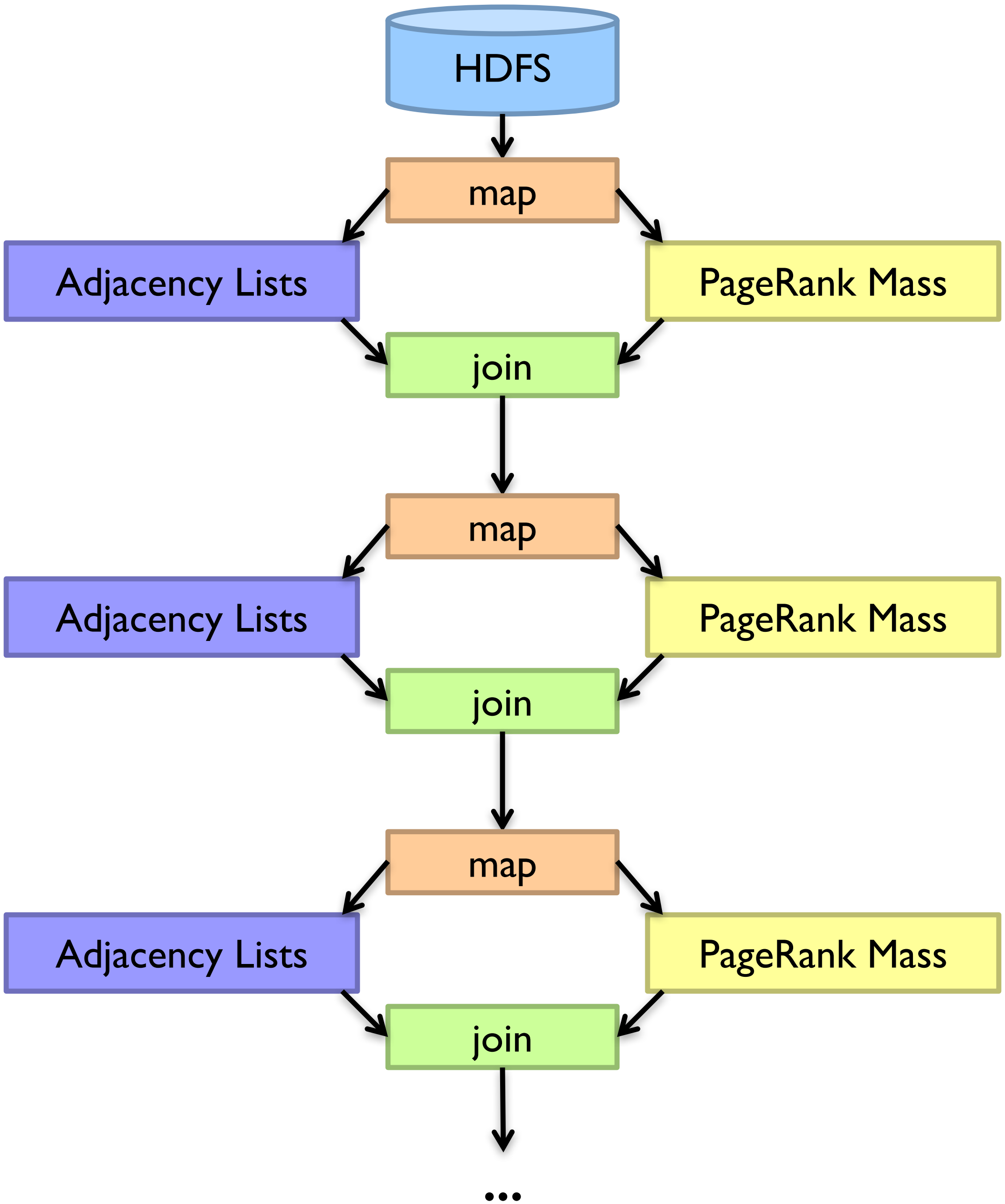
Checkpointing at each iteration

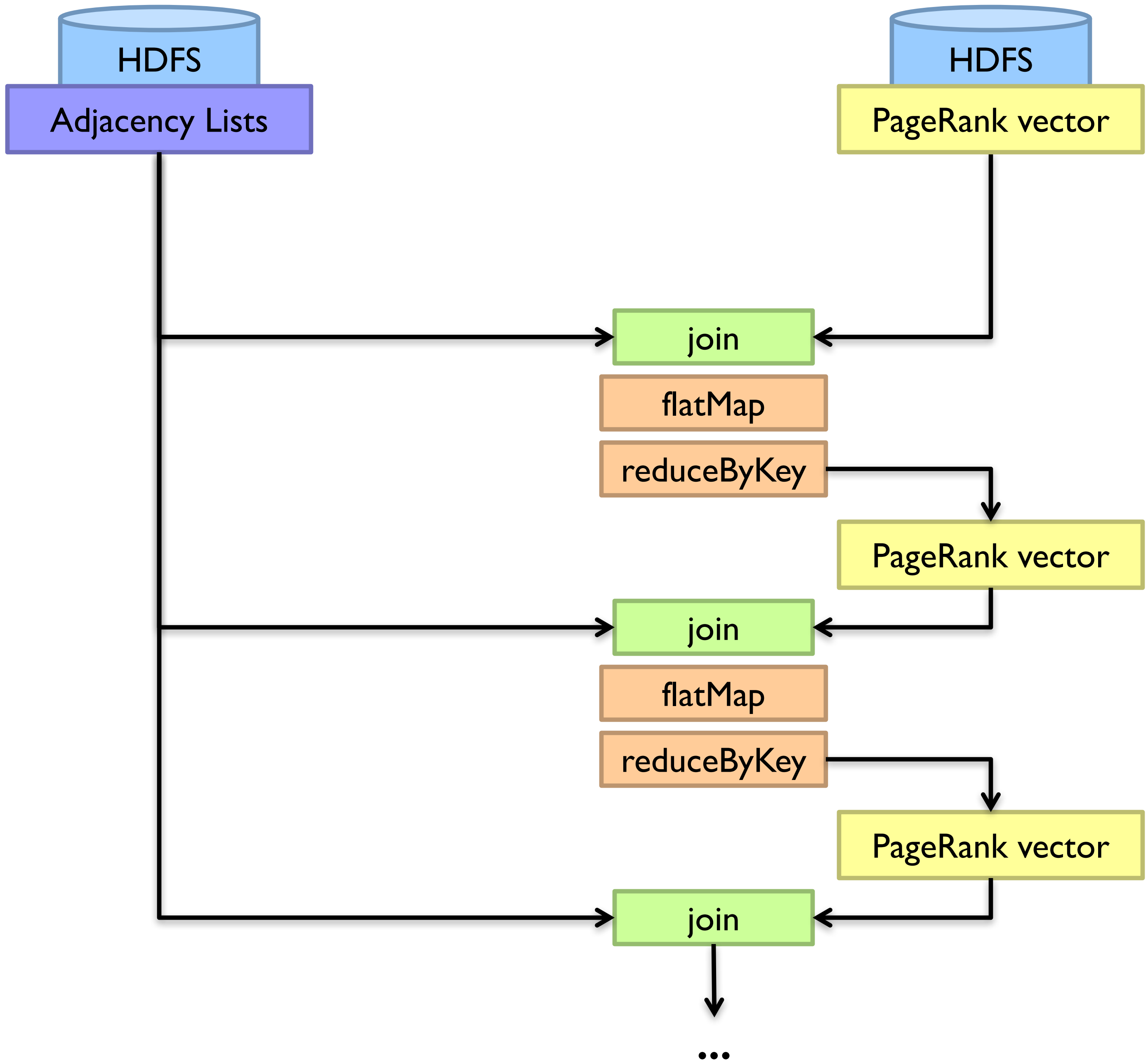
Let's Spark!

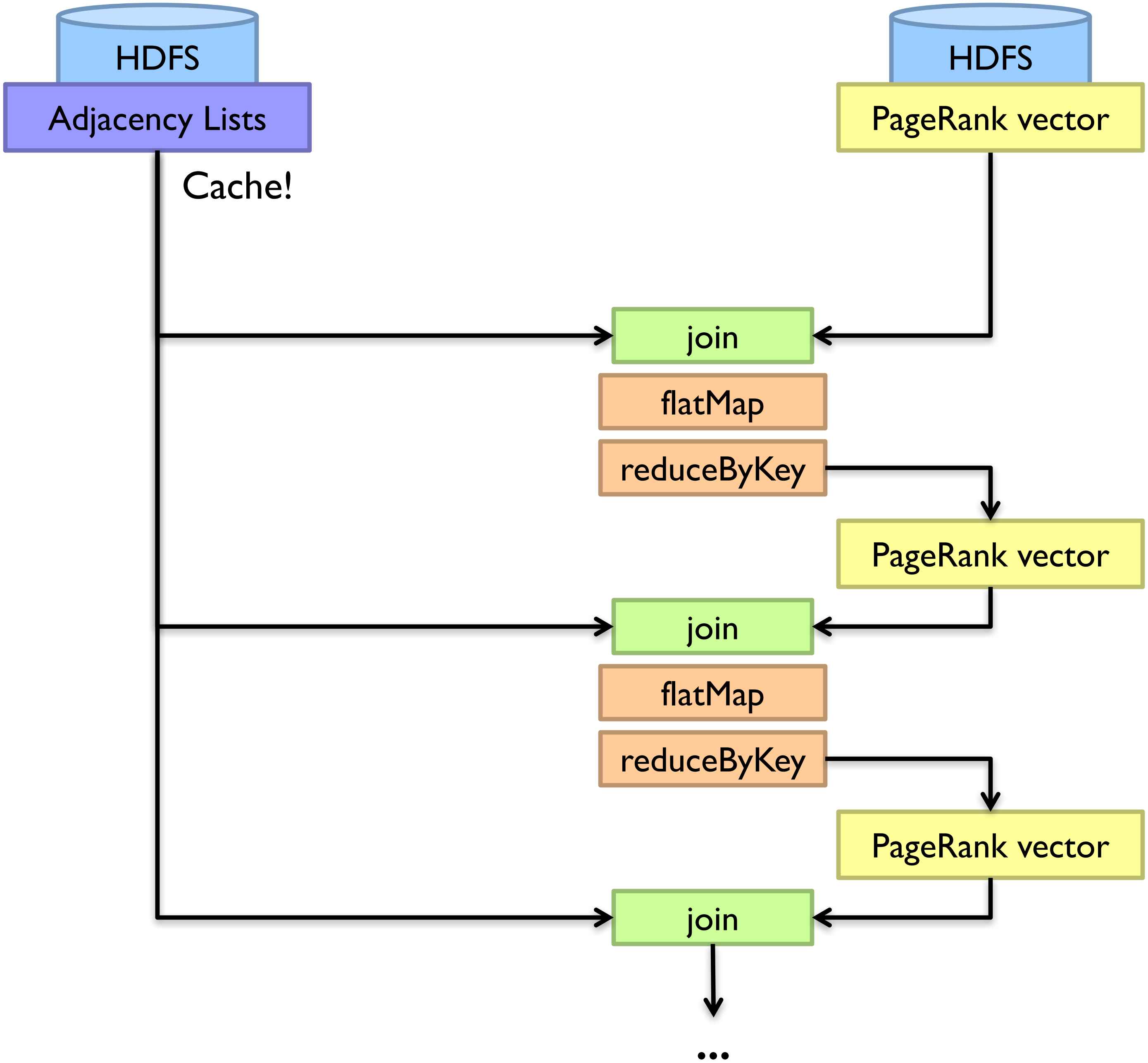



HDFS
map
reduce
map
reduce
map
reduce
...

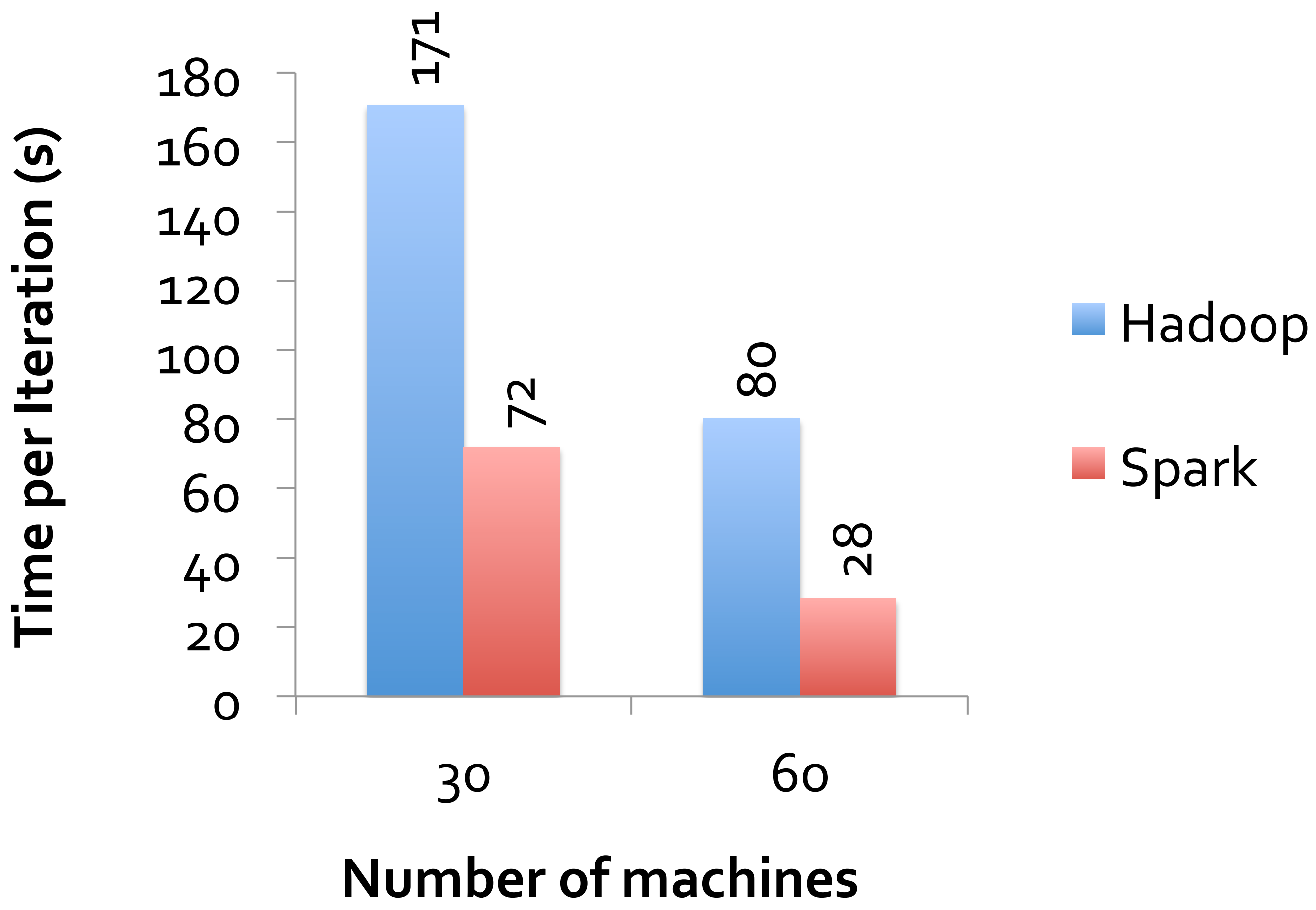

HDFS
map
Adjacency Lists
PageRank Mass
reduce
map
Adjacency Lists
PageRank Mass
reduce
map
Adjacency Lists
PageRank Mass
reduce
…


HDFS
map
Adjacency Lists
PageRank Mass
join
map
Adjacency Lists
PageRank Mass
join
map
Adjacency Lists
PageRank Mass
join
…


HDFS
Adjacency Lists
HDFS
PageRank vector
join
flatMap
reduceByKey
PageRank vector
join
flatMap
reduceByKey
PageRank vector
join
…


HDFS
Adjacency Lists
Cache!
HDFS
PageRank vector
join
flatMap
reduceByKey
PageRank vector
join
flatMap
reduceByKey
PageRank vector
join
…

MapReduce vs. Spark



Source: http://ampcamp.berkeley.edu/wp-content/uploads/2012/06/matei-zaharia-part-2-amp-camp-2012-standalone-programs.pdf

Characteristics of Graph Algorithms

Parallel graph traversals

Local computations

Message passing along graph edges

Iterations

Even faster?

Big Data Processing in a Nutshell

Let's be smarter about this!

Partition

Replicate

Reduce cross-partition communication

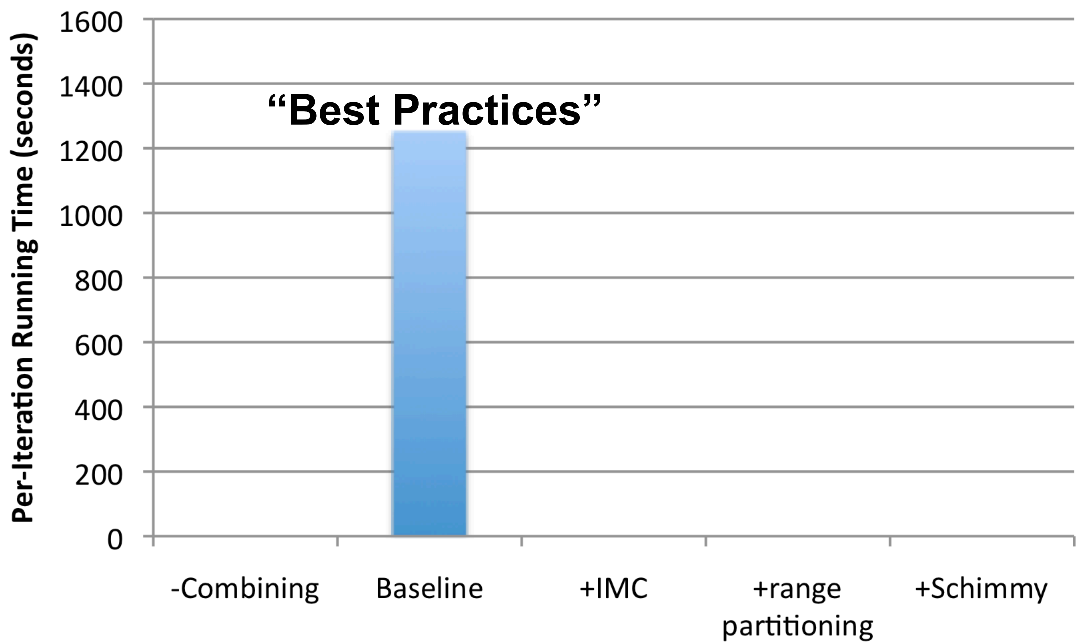
Simple Partitioning Techniques

Hash partitioning

Range partitioning on some underlying linearization

Web pages: lexicographic sort of domain-reversed URLs

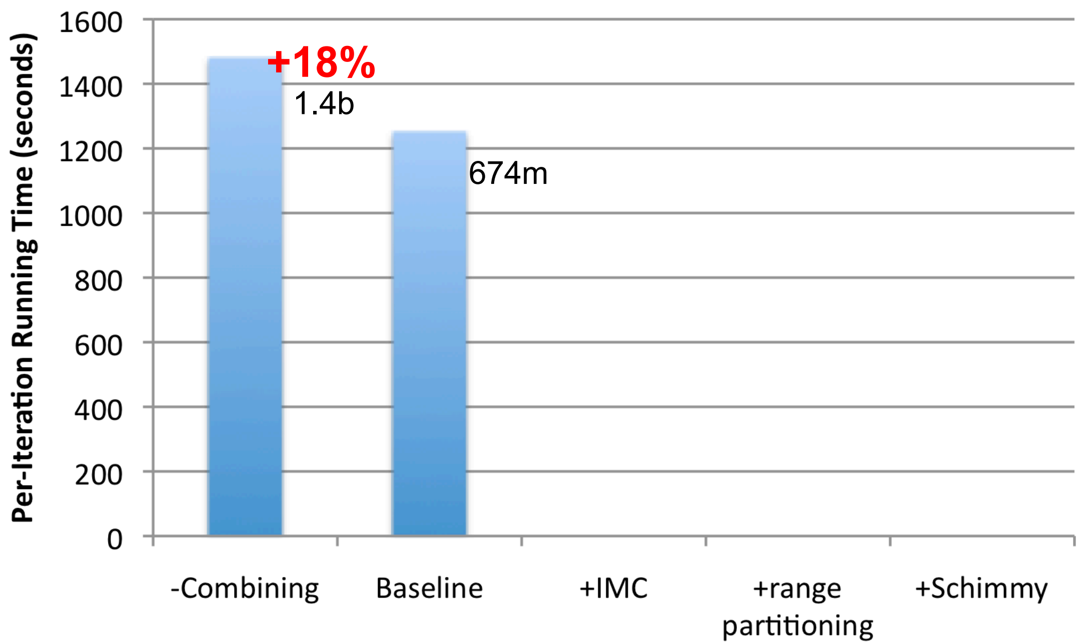
How much difference does it make?



PageRank over webgraph
(40m vertices, 1.4b edges)

Lin and Schatz. (2010) Design Patterns for Efficient Graph Algorithms in MapReduce.

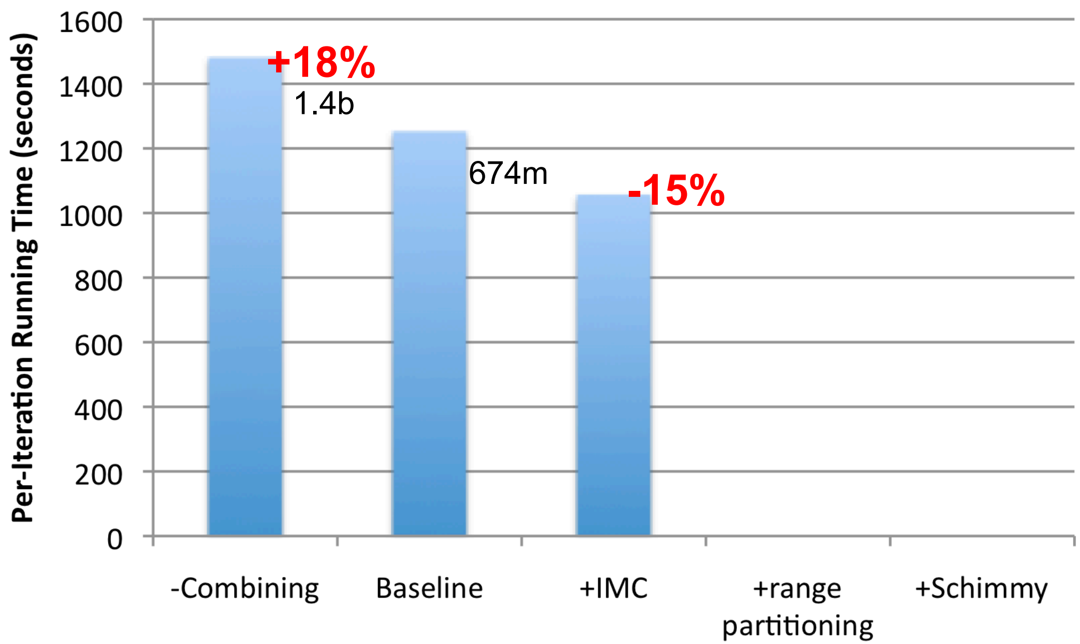
How much difference does it make?



PageRank over webgraph
(40m vertices, 1.4b edges)

Lin and Schatz. (2010) Design Patterns for Efficient Graph Algorithms in MapReduce.

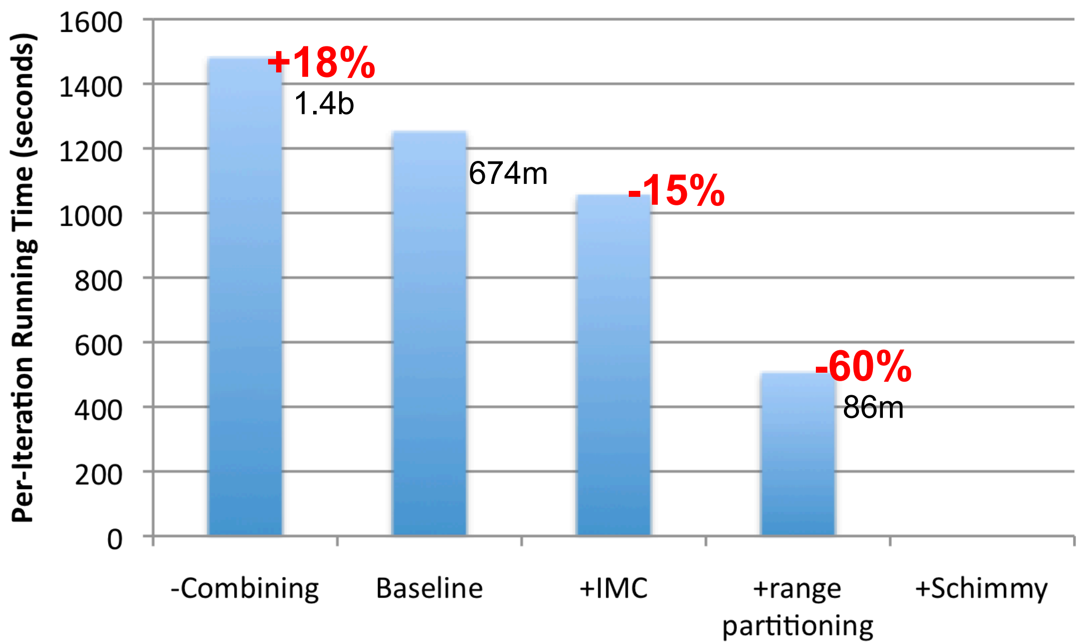
How much difference does it make?



PageRank over webgraph
(40m vertices, 1.4b edges)

Lin and Schatz. (2010) Design Patterns for Efficient Graph Algorithms in MapReduce.

How much difference does it make?



PageRank over webgraph
(40m vertices, 1.4b edges)

Lin and Schatz. (2010) Design Patterns for Efficient Graph Algorithms in MapReduce.

Schimmy Design Pattern

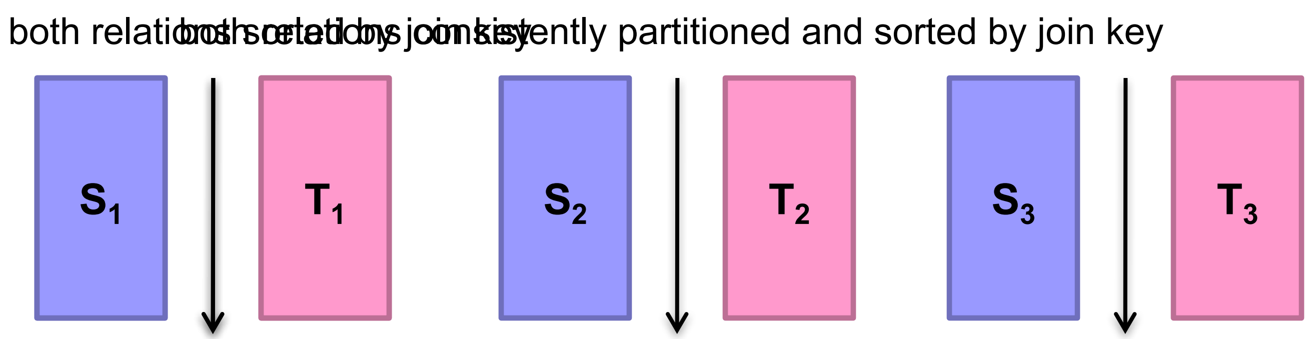
Basic implementation contains two dataflows:

Messages (actual computations)

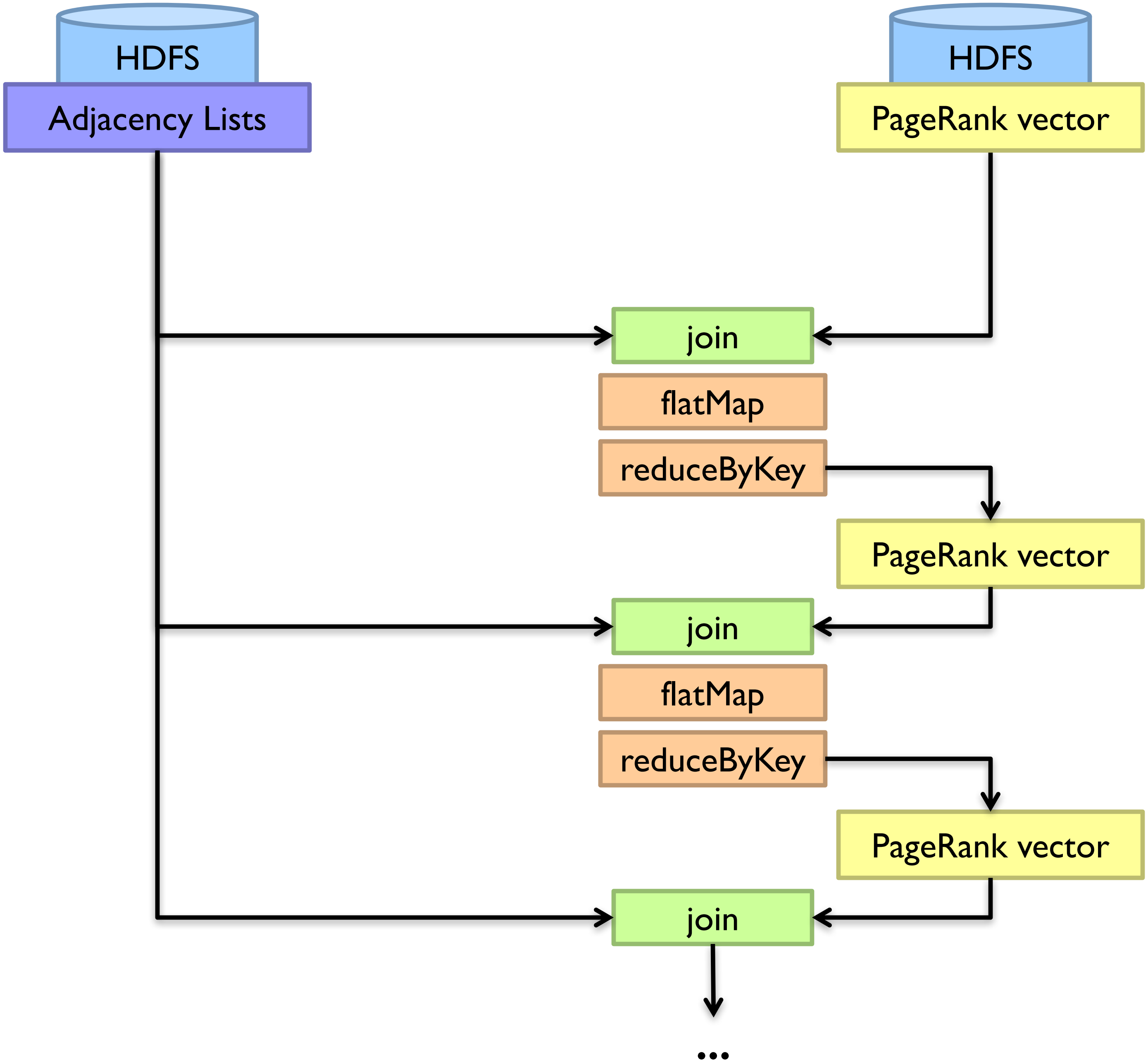
Graph structure ("bookkeeping")

Schimmy: separate the two dataflows, shuffle only the messages

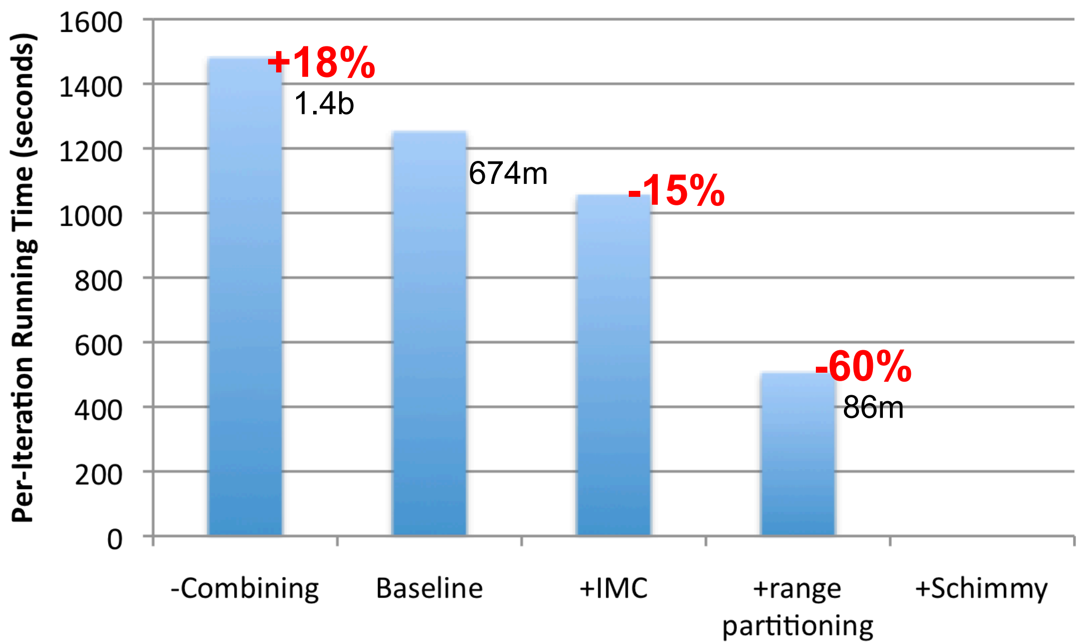
Basic idea: merge join between graph structure and messages



Lin and Schatz. (2010) Design Patterns for Efficient Graph Algorithms in MapReduce.


HDFS
Adjacency Lists
HDFS
PageRank vector
join
flatMap
reduceByKey
PageRank vector
join
flatMap
reduceByKey
PageRank vector
join
…

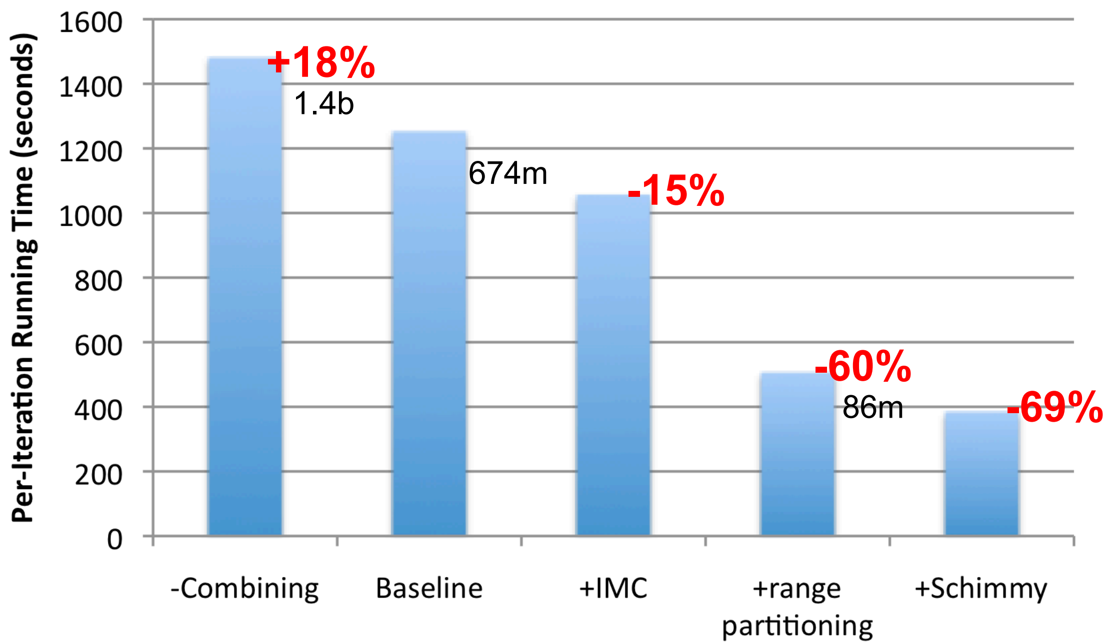
How much difference does it make?



PageRank over webgraph
(40m vertices, 1.4b edges)

Lin and Schatz. (2010) Design Patterns for Efficient Graph Algorithms in MapReduce.

How much difference does it make?



PageRank over webgraph
(40m vertices, 1.4b edges)

Lin and Schatz. (2010) Design Patterns for Efficient Graph Algorithms in MapReduce.

Simple Partitioning Techniques

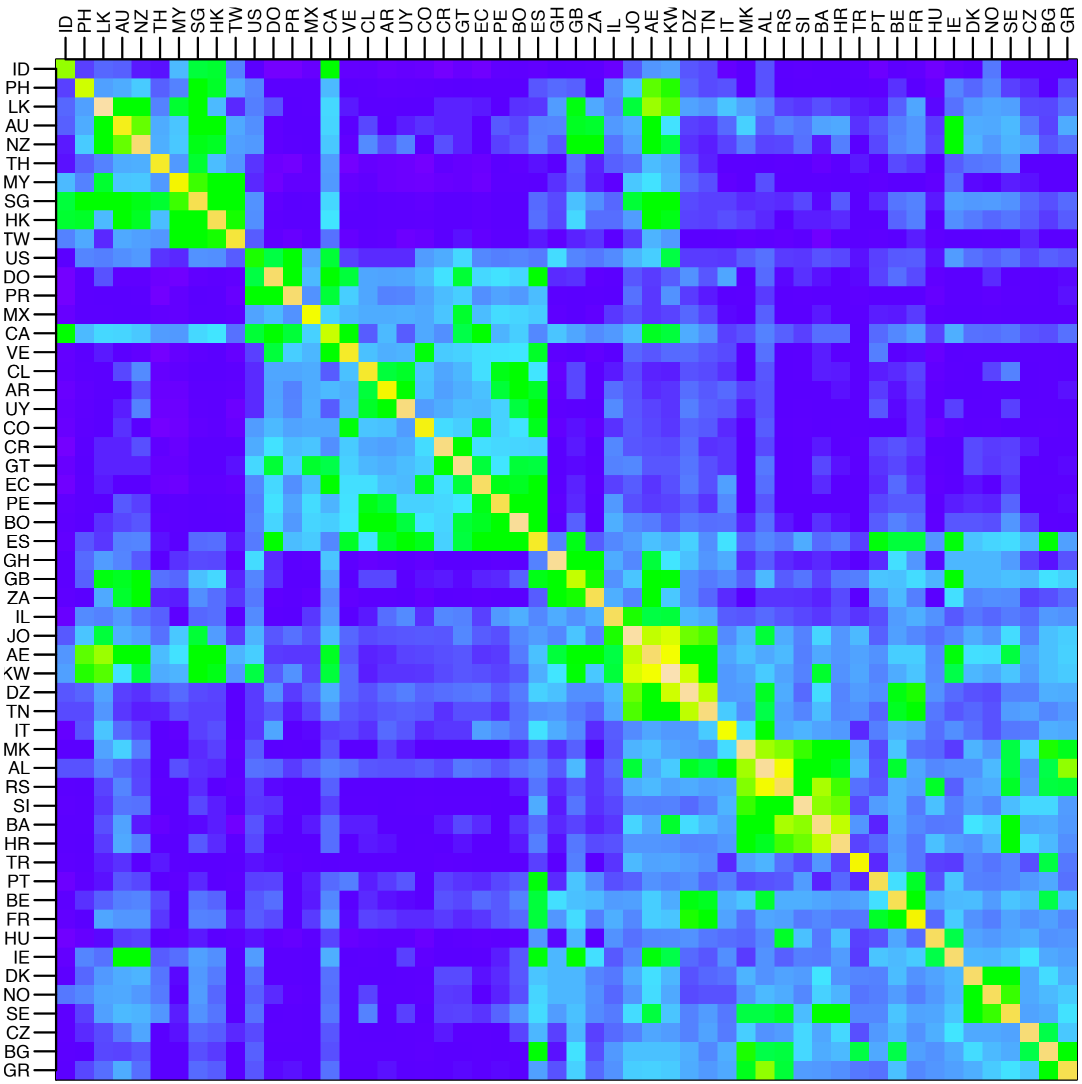
Hash partitioning

Range partitioning on some underlying linearization

Web pages: lexicographic sort of domain-reversed URLs

Social networks: sort by demographic characteristics

Country Structure in Facebook



Analysis of 721 million active users (May 2011)

54 countries w/ >1m active users, >50% penetration

Ugander et al. (2011) The Anatomy of the Facebook Social Graph.

Simple Partitioning Techniques

Hash partitioning

Range partitioning on some underlying linearization

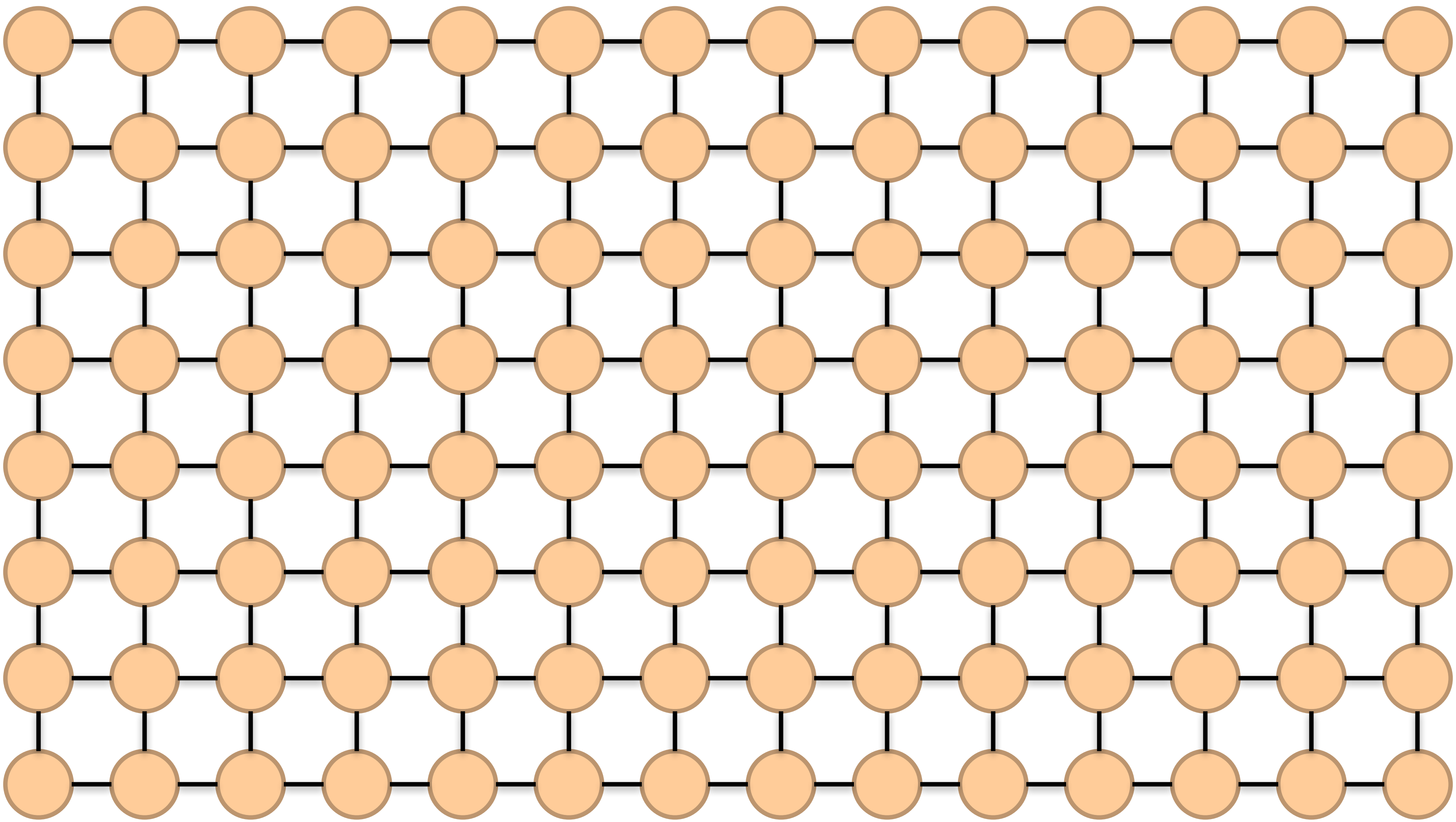
Web pages: lexicographic sort of domain-reversed URLs

Social networks: sort by demographic characteristics

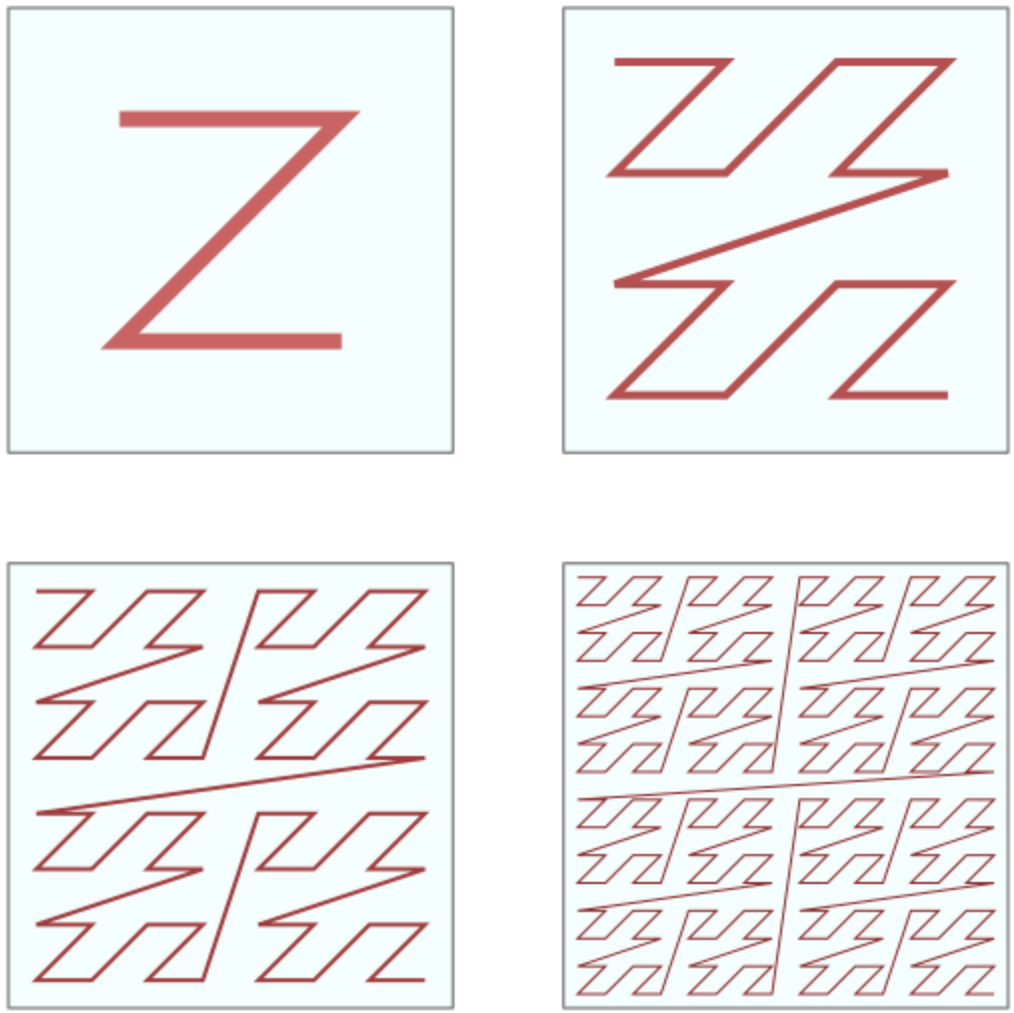
Geo data: space-filling curves

Aside: Partitioning Geo-data

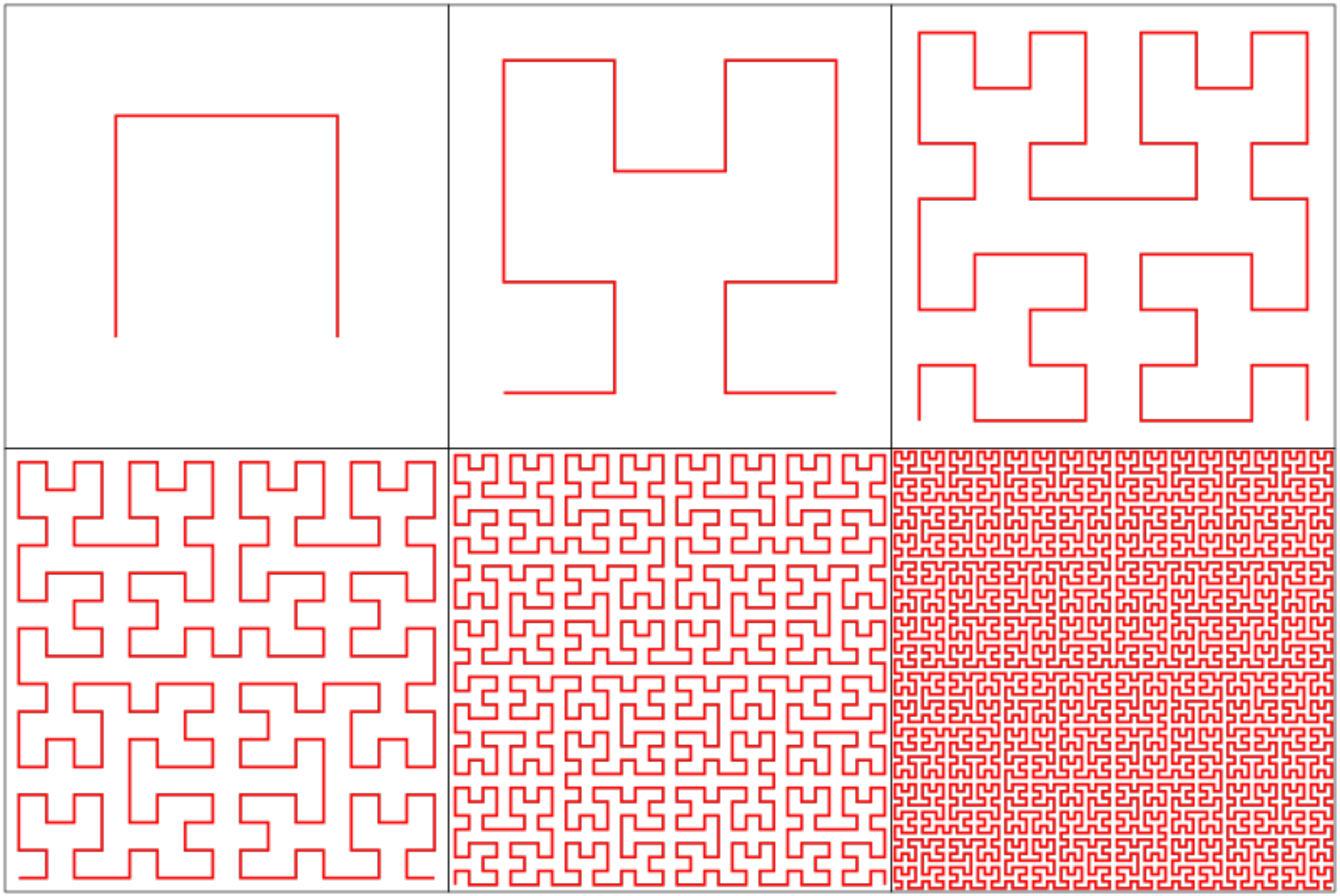
Geo-data = regular graph



Space-filling curves: Z-Order Curves



Space-filling curves: Hilbert Curves



Simple Partitioning Techniques

Hash partitioning

Range partitioning on some underlying linearization

Web pages: lexicographic sort of domain-reversed URLs

Social networks: sort by demographic characteristics

Geo data: space-filling curves

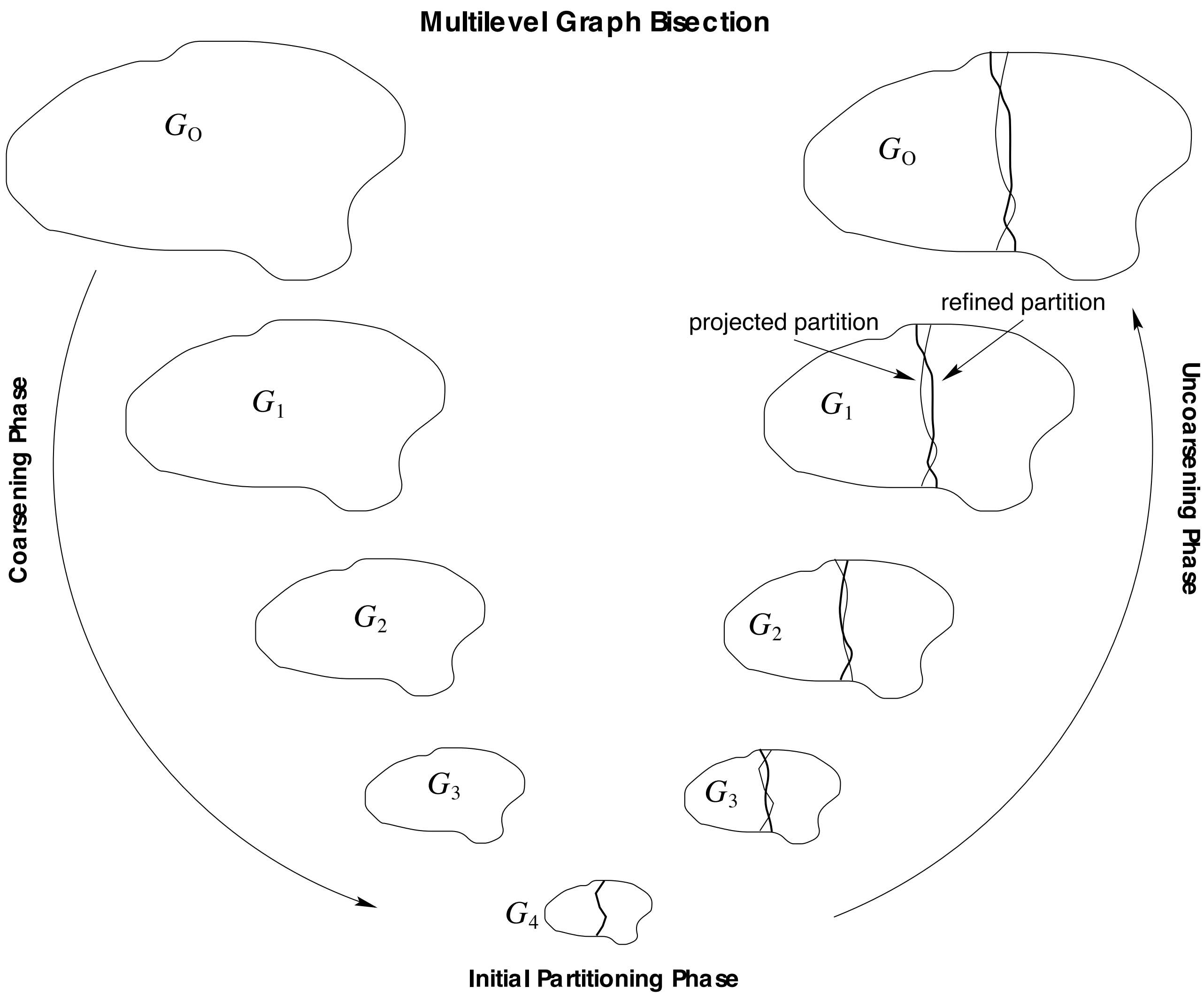
But what about graphs in general?

Source: http://www.flickr.com/photos/fusedforces/4324320625/

General-Purpose Graph Partitioning

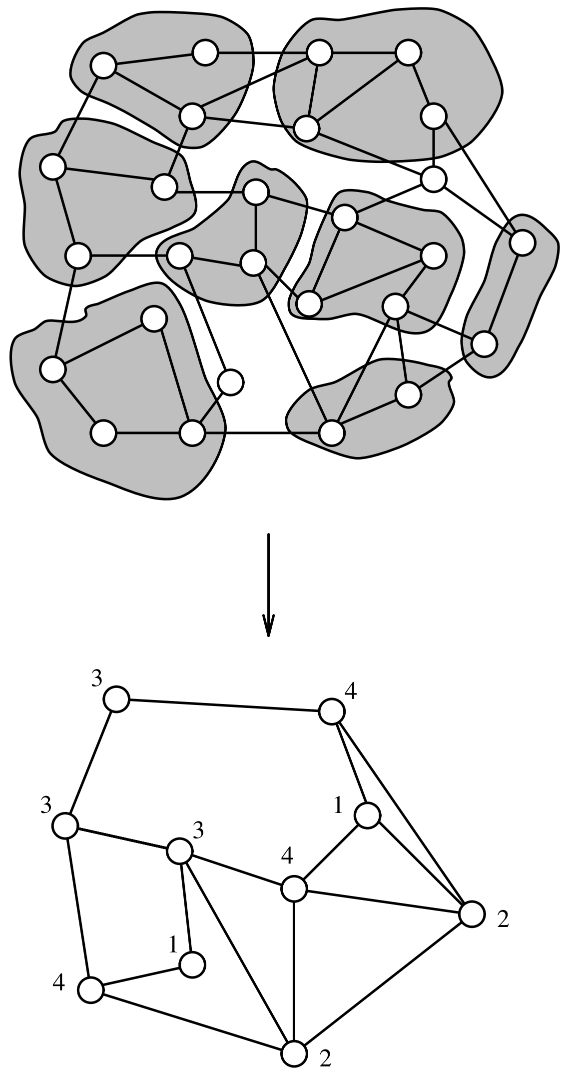
Graph coarsening
Recursive bisection

General-Purpose Graph Partitioning



Karypis and Kumar. (1998) A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs.

Graph Coarsening



Karypis and Kumar. (1998) A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs.

Chicken-and-Egg

To coarsen the graph you need to identify dense local regions
To identify dense local regions quickly you to need traverse local edges
But to traverse local edges efficiently you need the local structure!

To efficiently partition the graph, you need to already know what the partitions are!
Industry solution?

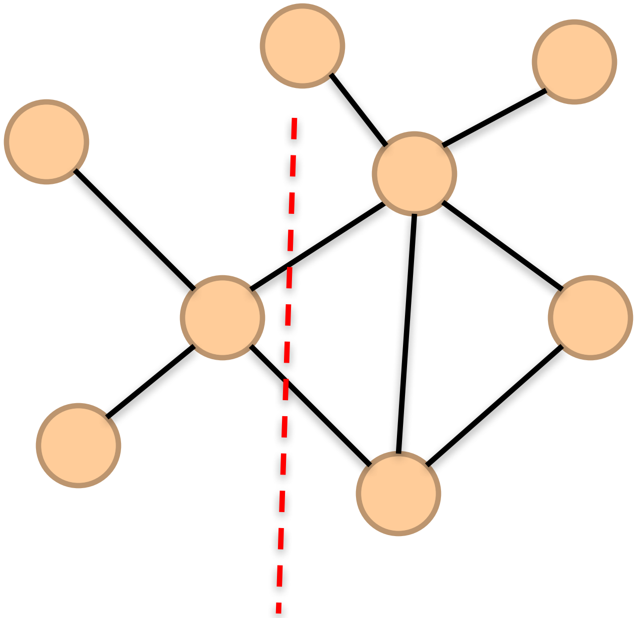
Big Data Processing in a Nutshell

Partition

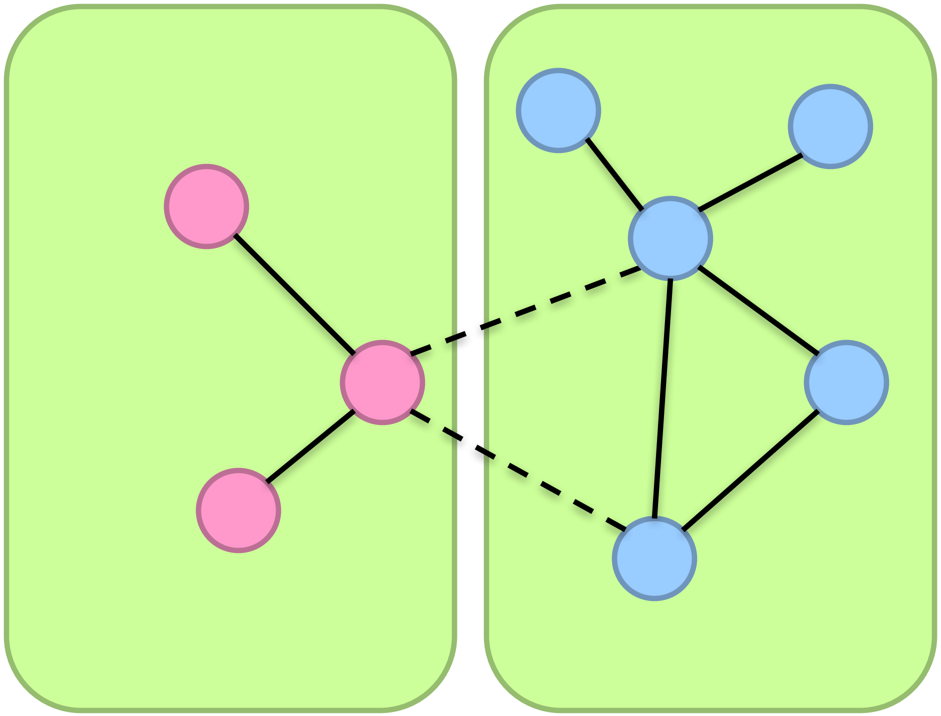
Replicate

Reduce cross-partition communication

Partition



Partition



What's the fundamental issue?

Characteristics of Graph Algorithms

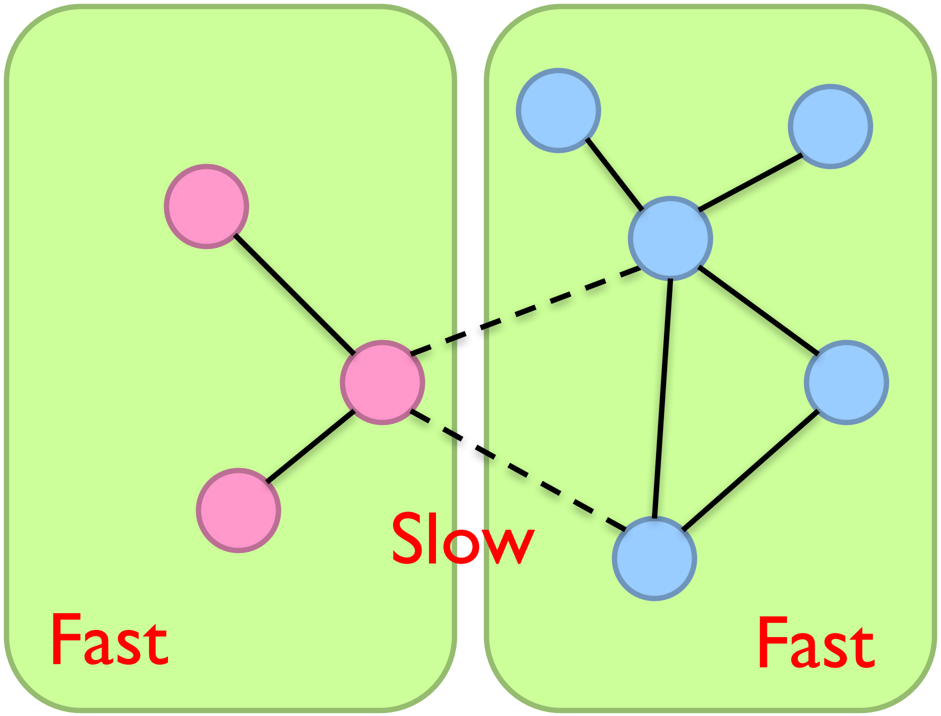
Parallel graph traversals

Local computations

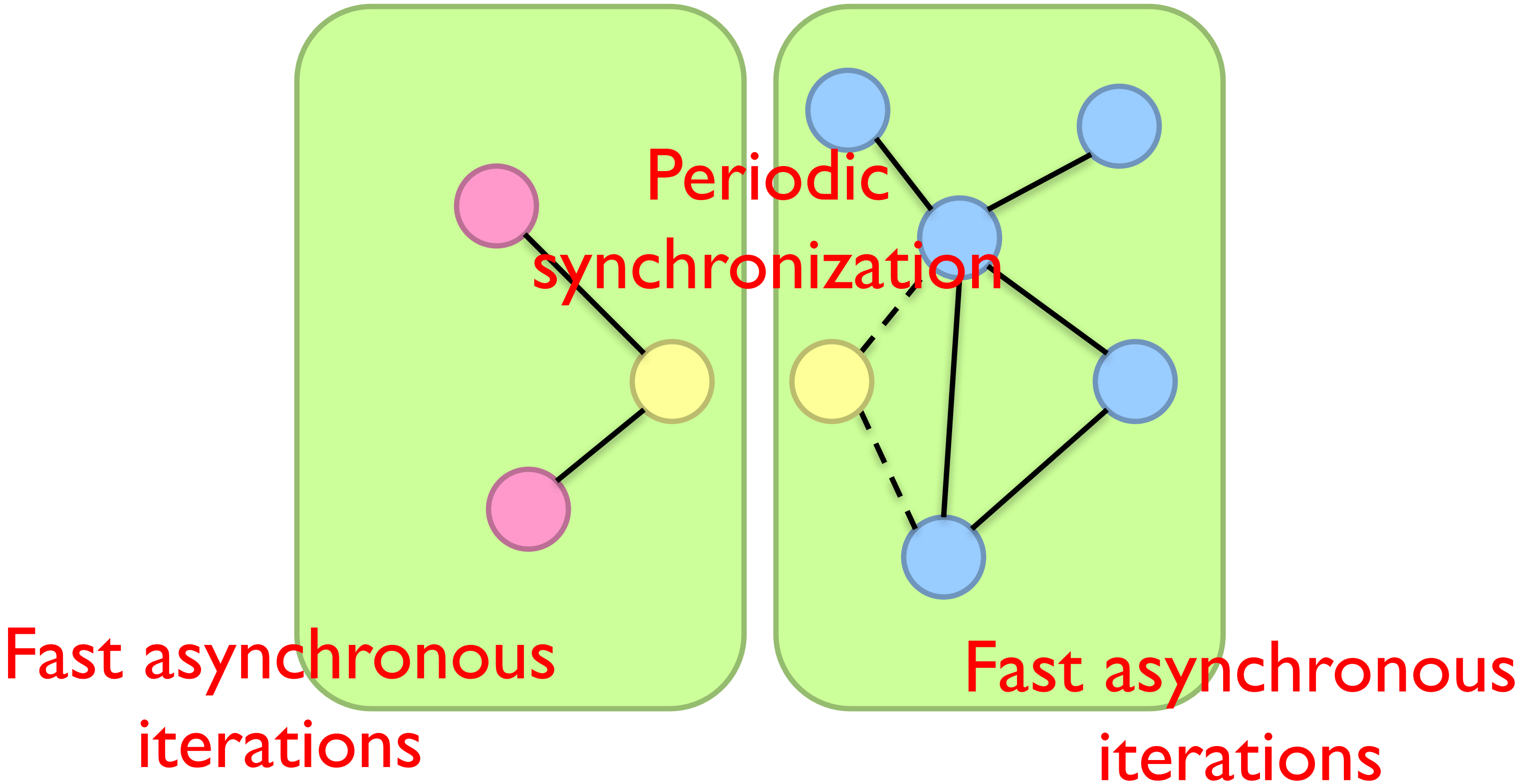
Message passing along graph edges

Iterations

Partition

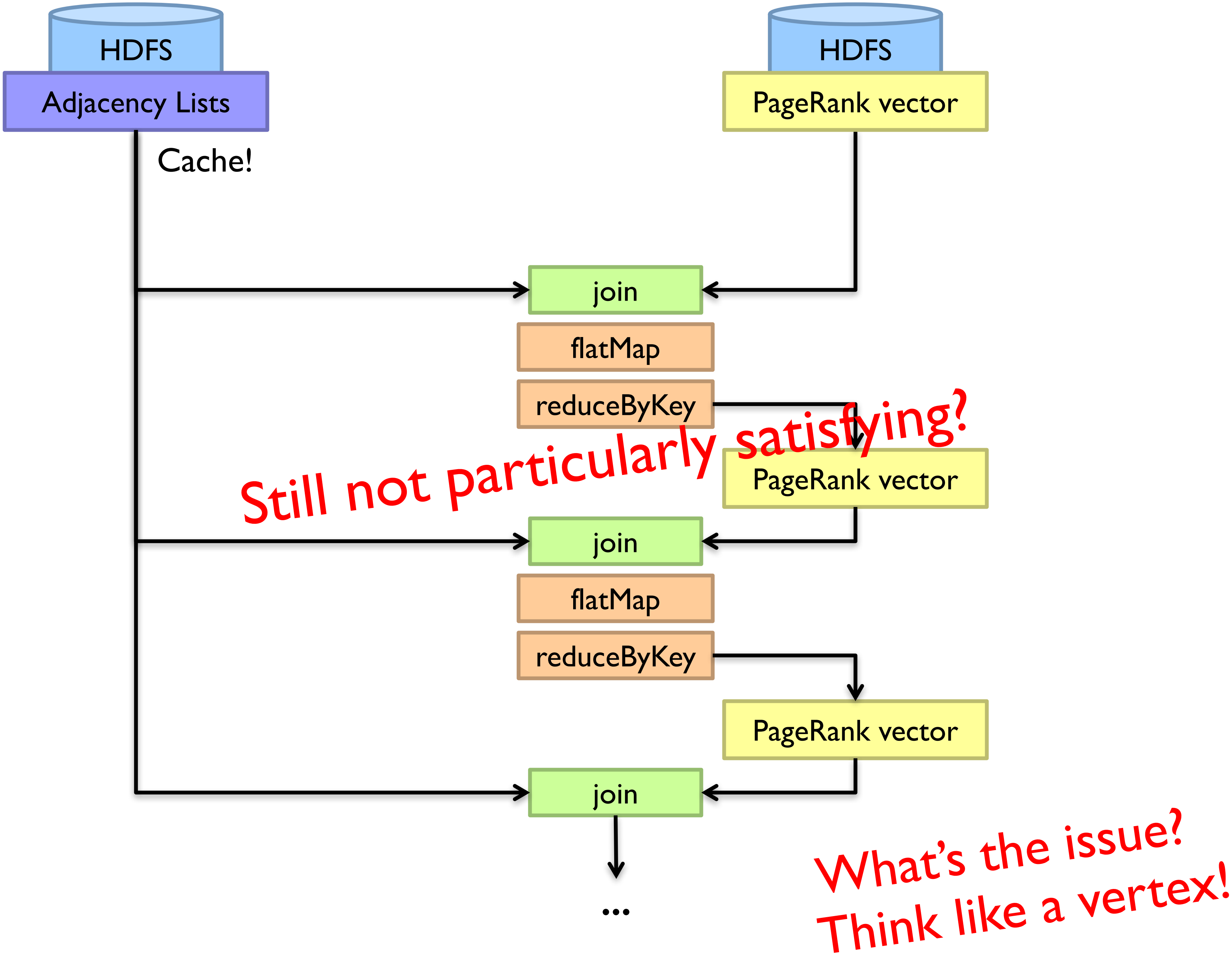


State-of-the-Art Distributed Graph Algorithms



Graph Processing Frameworks

Source: Wikipedia (Waste container)


HDFS
Adjacency Lists
Cache!
HDFS
PageRank vector
join
flatMap
reduceByKey
PageRank vector
join
flatMap
reduceByKey
PageRank vector
join
…
Still not particularly satisfying?
What's the issue?
Think like a vertex!

Pregel: Computational Model

Based on Bulk Synchronous Parallel (BSP)

Computational units encoded in a directed graph

Computation proceeds in a series of supersteps

Message passing architecture

Each vertex, at each superstep:

Receives messages directed at it from previous superstep

Executes a user-defined function (modifying state)

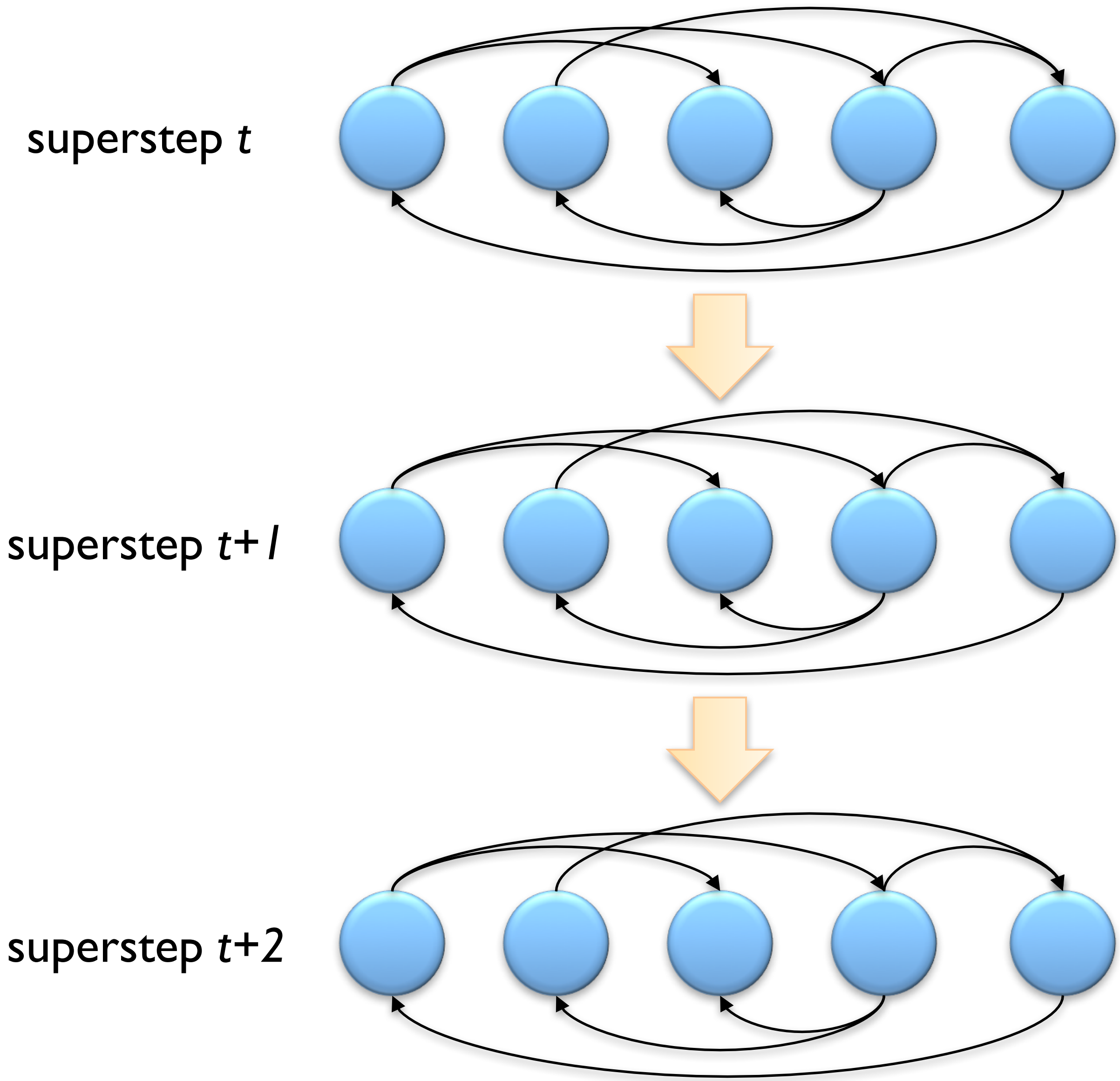
Emits messages to other vertices (for the next superstep)

Termination:

A vertex can choose to deactivate itself

Is "woken up" if new messages received

Computation halts when all vertices are inactive



Source: Malewicz et al. (2010) Pregel: A System for Large-Scale Graph Processing. SIGMOD.

Pregel: Implementation

Master-Worker architecture

Vertices are hash partitioned (by default) and assigned to workers

Everything happens in memory

Processing cycle:

Master tells all workers to advance a single superstep

Worker delivers messages from previous superstep, executing vertex computation

Messages sent asynchronously (in batches)

Worker notifies master of number of active vertices

Fault tolerance

Checkpointing

Heartbeat/revert

Pregel: SSSP

```
class ShortestPathVertex : public Vertex<int, int, int> {
  void Compute(MessageIterator* msgs) {
    int mindist = IsSource(vertex_id()) ? 0 : INF;
    for (; !msgs->Done(); msgs->Next())
      mindist = min(mindist, msgs->Value());
    if (mindist < GetValue()) {
      *MutableValue() = mindist;
      OutEdgeIterator iter = GetOutEdgeIterator();
      for (; !iter.Done(); iter.Next())
        SendMessageTo(iter.Target(),
                      mindist + iter.GetValue());
    }
    VoteToHalt();
  }
};
```

Source: Malewicz et al. (2010) Pregel: A System for Large-Scale Graph Processing. SIGMOD.

Pregel: PageRank

```
class PageRankVertex : public Vertex<double, void, double> {
public:
  virtual void Compute(MessageIterator* msgs) {
    if (superstep() >= 1) {
      double sum = 0;
      for (; !msgs->Done(); msgs->Next())
        sum += msgs->Value();
      *MutableValue() = 0.15 / NumVertices() + 0.85 * sum;
    }

    if (superstep() < 30) {
      const int64 n = GetOutEdgeIterator().size();
      SendMessageToAllNeighbors(GetValue() / n);
    } else {
      VoteToHalt();
    }
  }
};
```

Source: Malewicz et al. (2010) Pregel: A System for Large-Scale Graph Processing. SIGMOD.

Pregel: Combiners

```
class MinIntCombiner : public Combiner<int> {
  virtual void Combine(MessageIterator* msgs) {

  int mindist = INF;
  for (; !msgs->Done(); msgs->Next())
    mindist = min(mindist, msgs->Value());
    Output("combined_source", mindist);
  }

};
```

Source: Malewicz et al. (2010) Pregel: A System for Large-Scale Graph Processing. SIGMOD.

APACHE

GIRAPH

Giraph Architecture

Master – Application coordinator

Synchronizes supersteps

Assigns partitions to workers before superstep begins

Workers – Computation & messaging

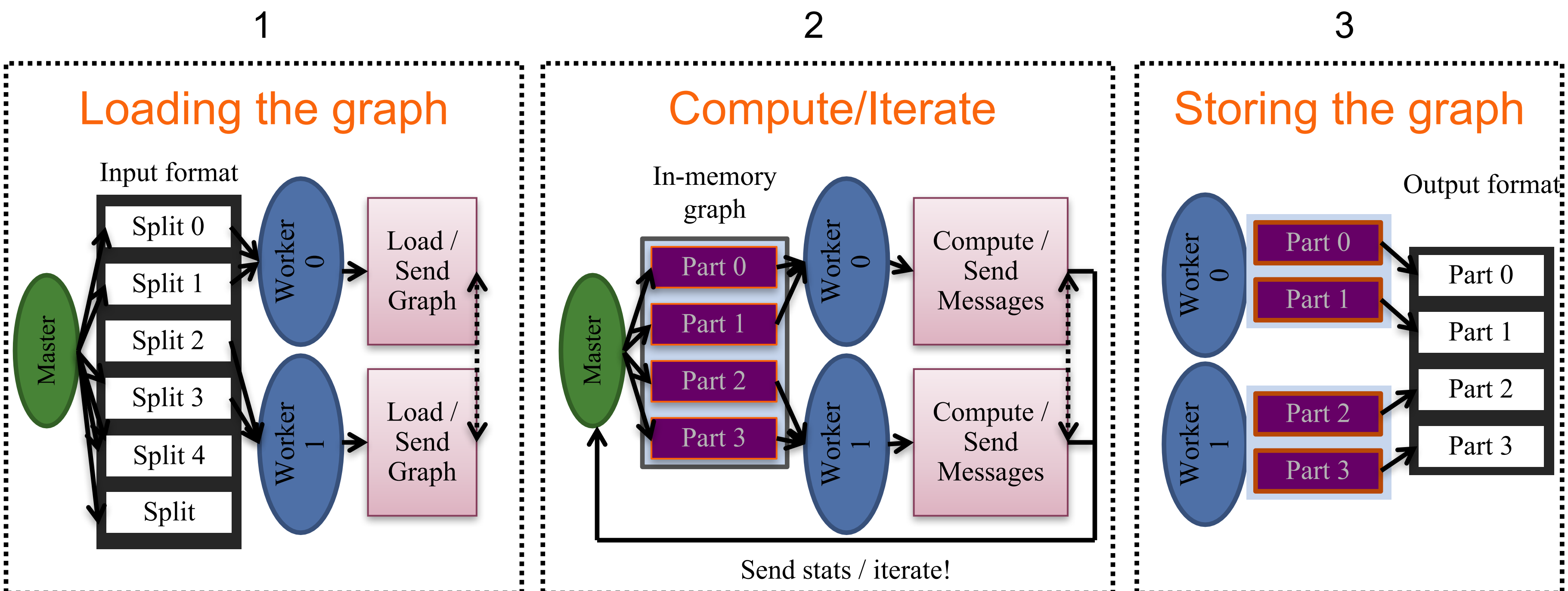
Handle I/O – reading and writing the graph

Computation/messaging of assigned partitions

ZooKeeper

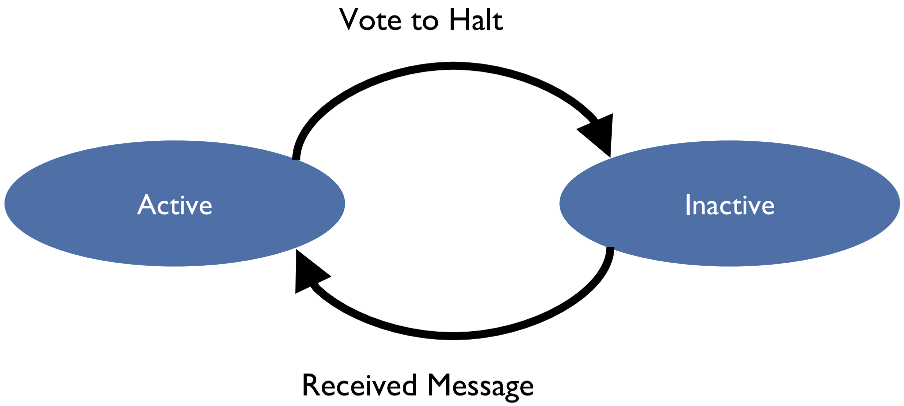
Maintains global application state

Giraph Dataflow

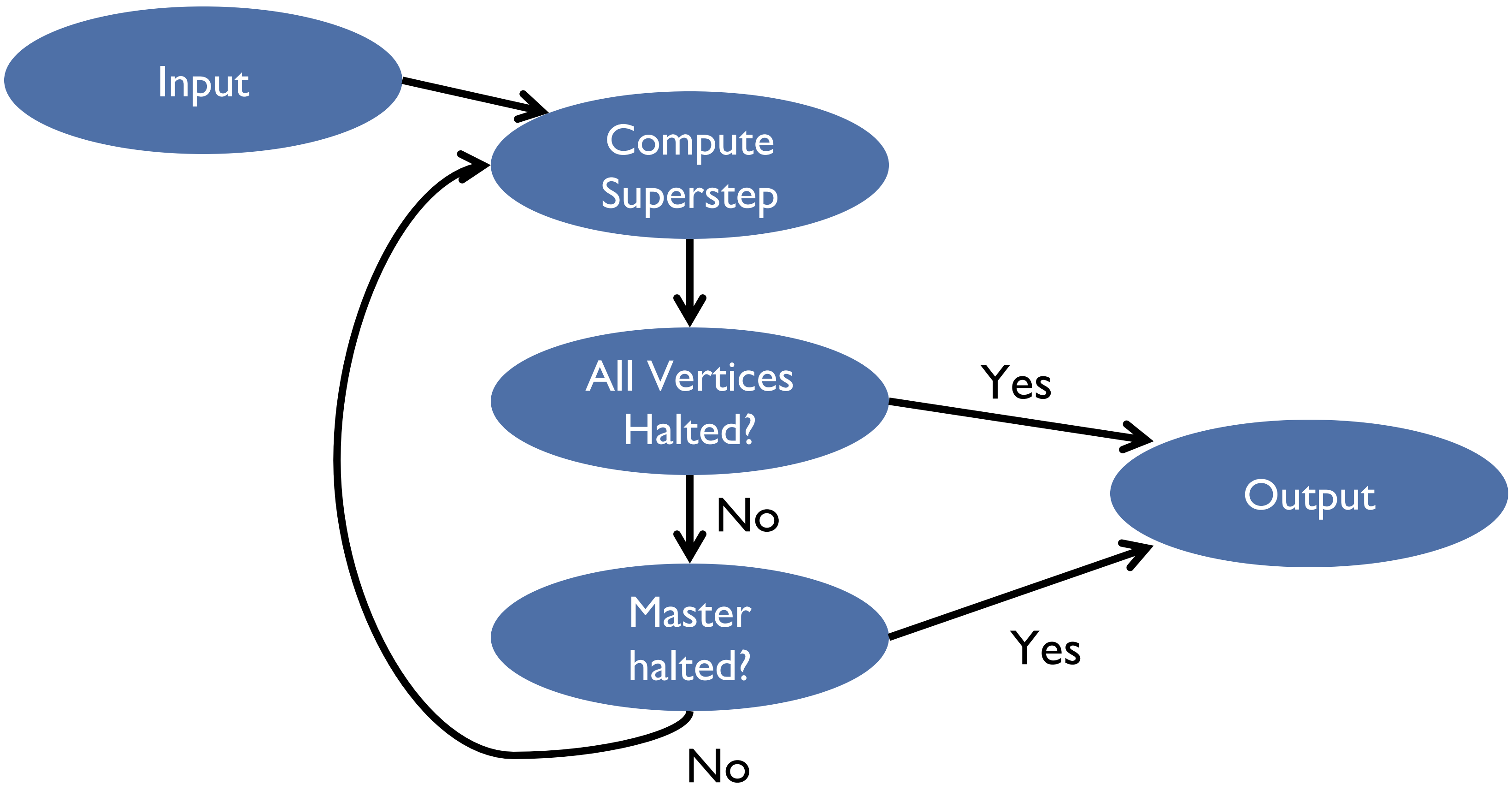


Giraph Lifecycle

Vertex Lifecycle



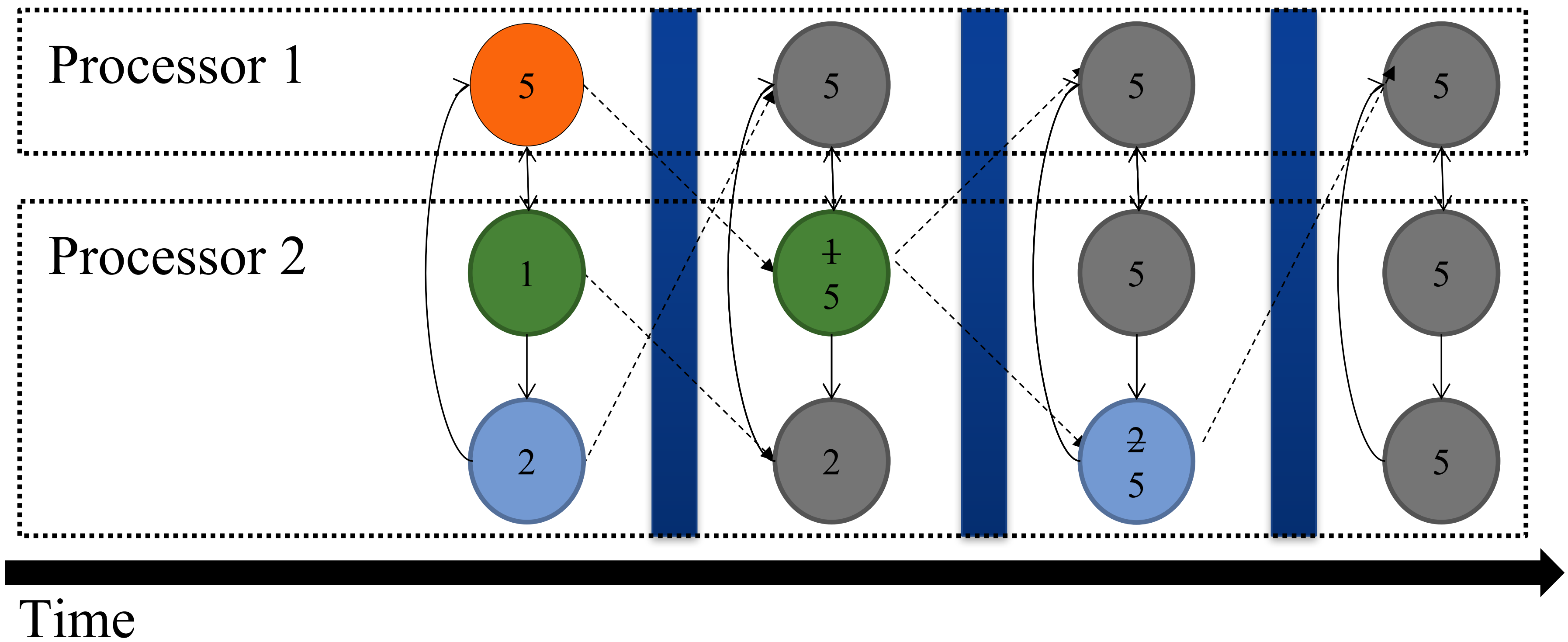
Giraph Lifecycle

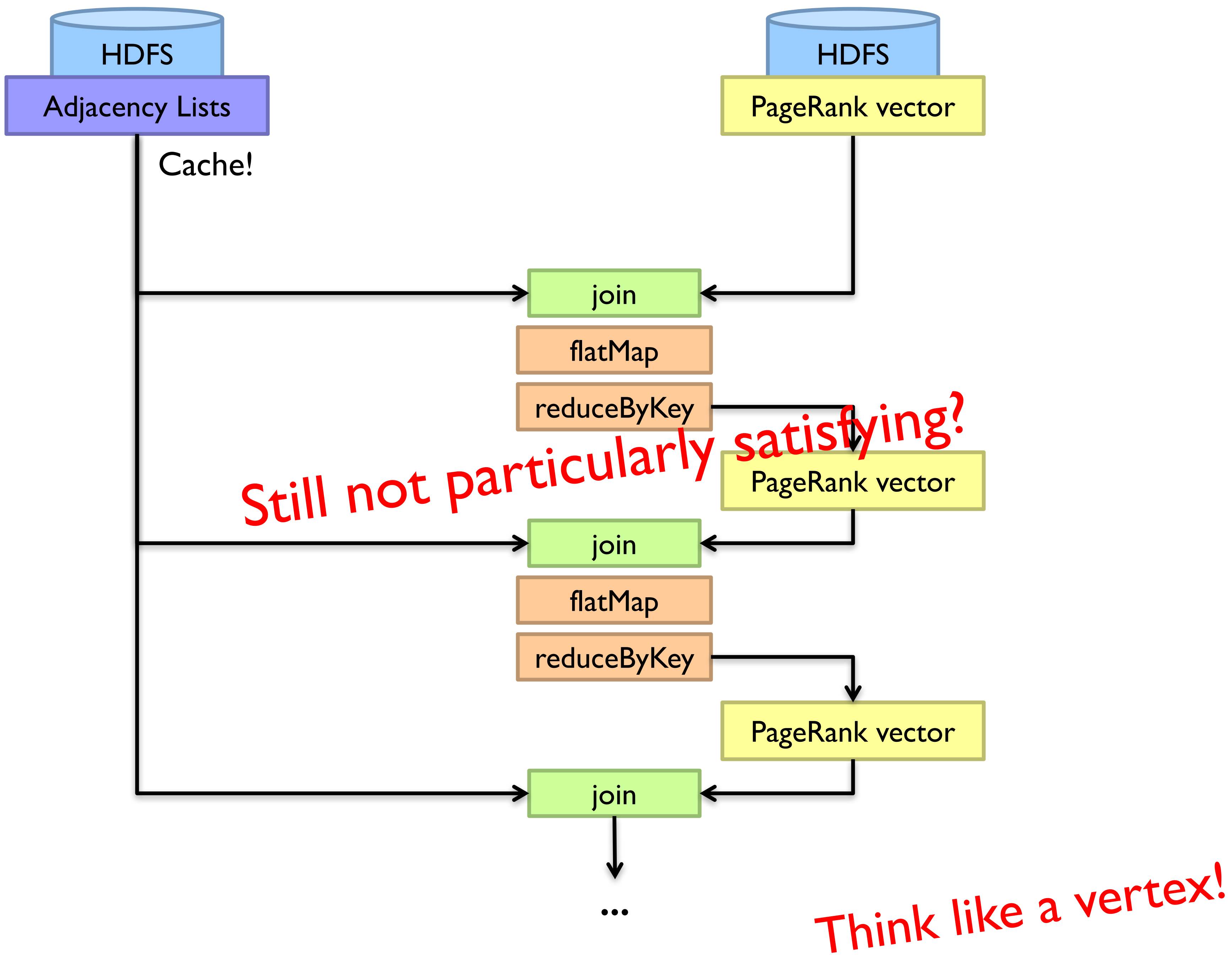


Giraph Example

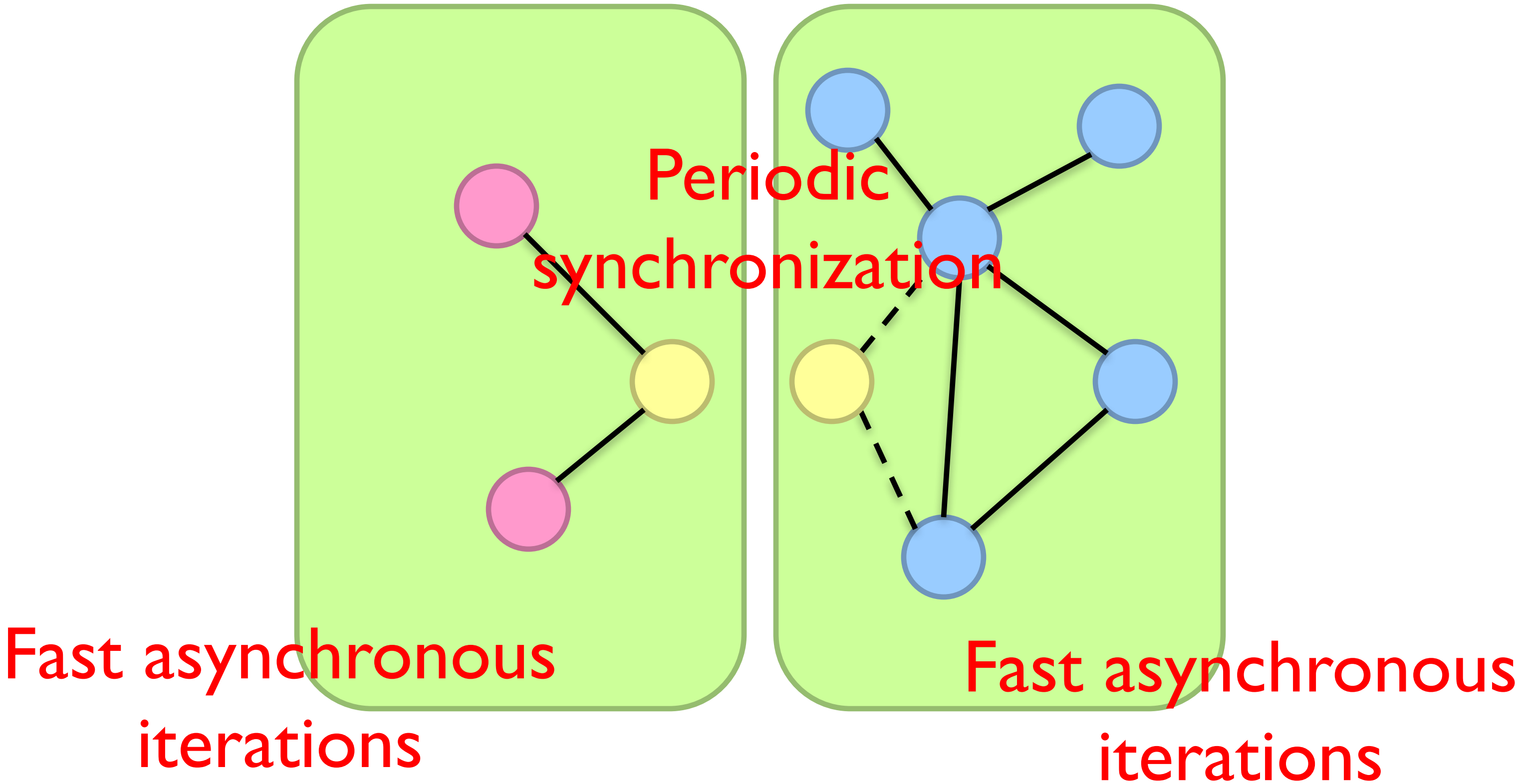
```java
public class MaxComputation extends BasicComputation<IntWritable, IntWritable,
    NullWritable, IntWritable> {
  @Override
  public void compute(Vertex<IntWritable, IntWritable, NullWritable> vertex,
      Iterable<IntWritable> messages) throws IOException
  {
    boolean changed = false;
    for (IntWritable message : messages) {
      if (vertex.getValue().get() < message.get()) {
        vertex.setValue(message);
        changed = true;
      }
    }
    if (getSuperstep() == 0 || changed) {
      sendMessageToAllEdges(vertex, vertex.getValue());
    }
    vertex.voteToHalt();
  }
}
```

Execution Trace




HDFS
Adjacency Lists
HDFS
PageRank vector
Cache!
join
flatMap
reduceByKey
PageRank vector
join
flatMap
reduceByKey
PageRank vector
join
…
Still not particularly satisfying?
Think like a vertex!

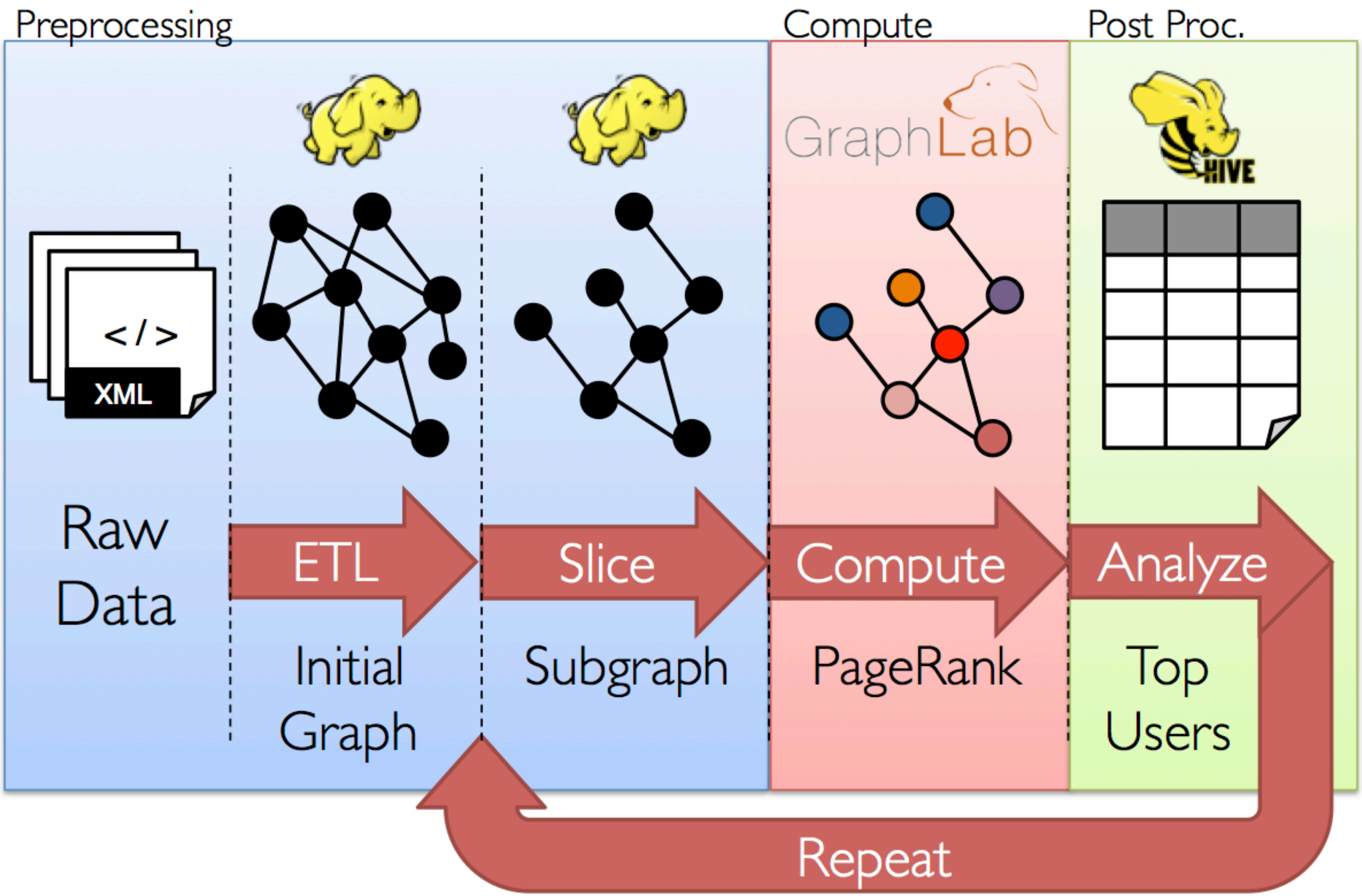
State-of-the-Art Distributed Graph Algorithms



Graph Processing Frameworks

Source: Wikipedia (Waste container)

GraphX: Motivation



GraphX = Spark for Graphs

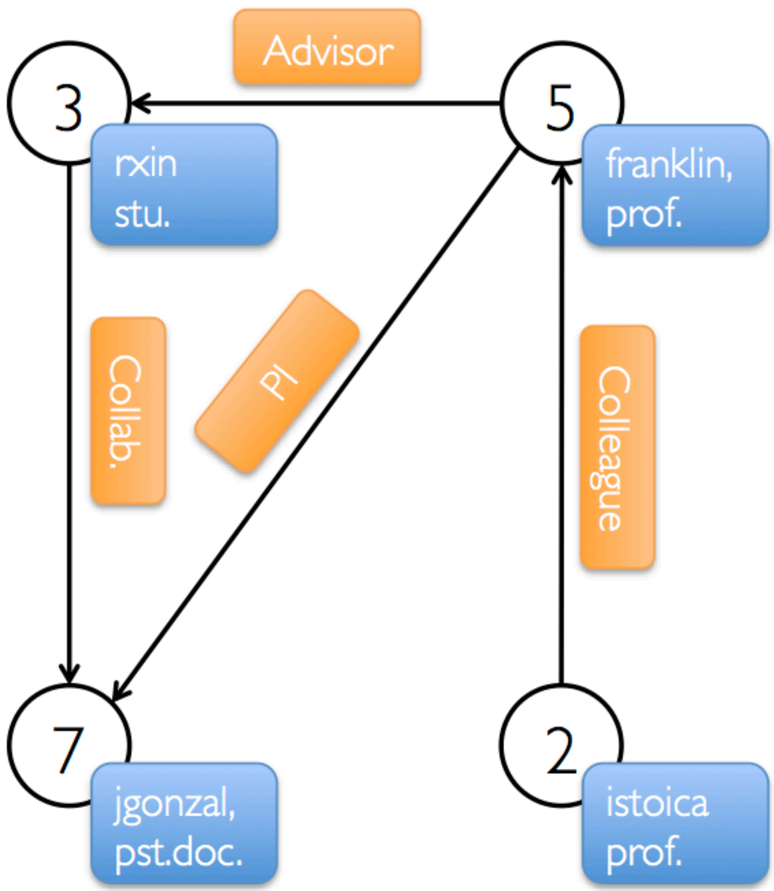
Integration of record-oriented and graph-oriented processing

Extends RDDs to Resilient Distributed Property Graphs

```
class Graph[VD, ED] {
  val vertices: VertexRDD[VD]
  val edges: EdgeRDD[ED]
}
```

Property Graph: Example

Property Graph



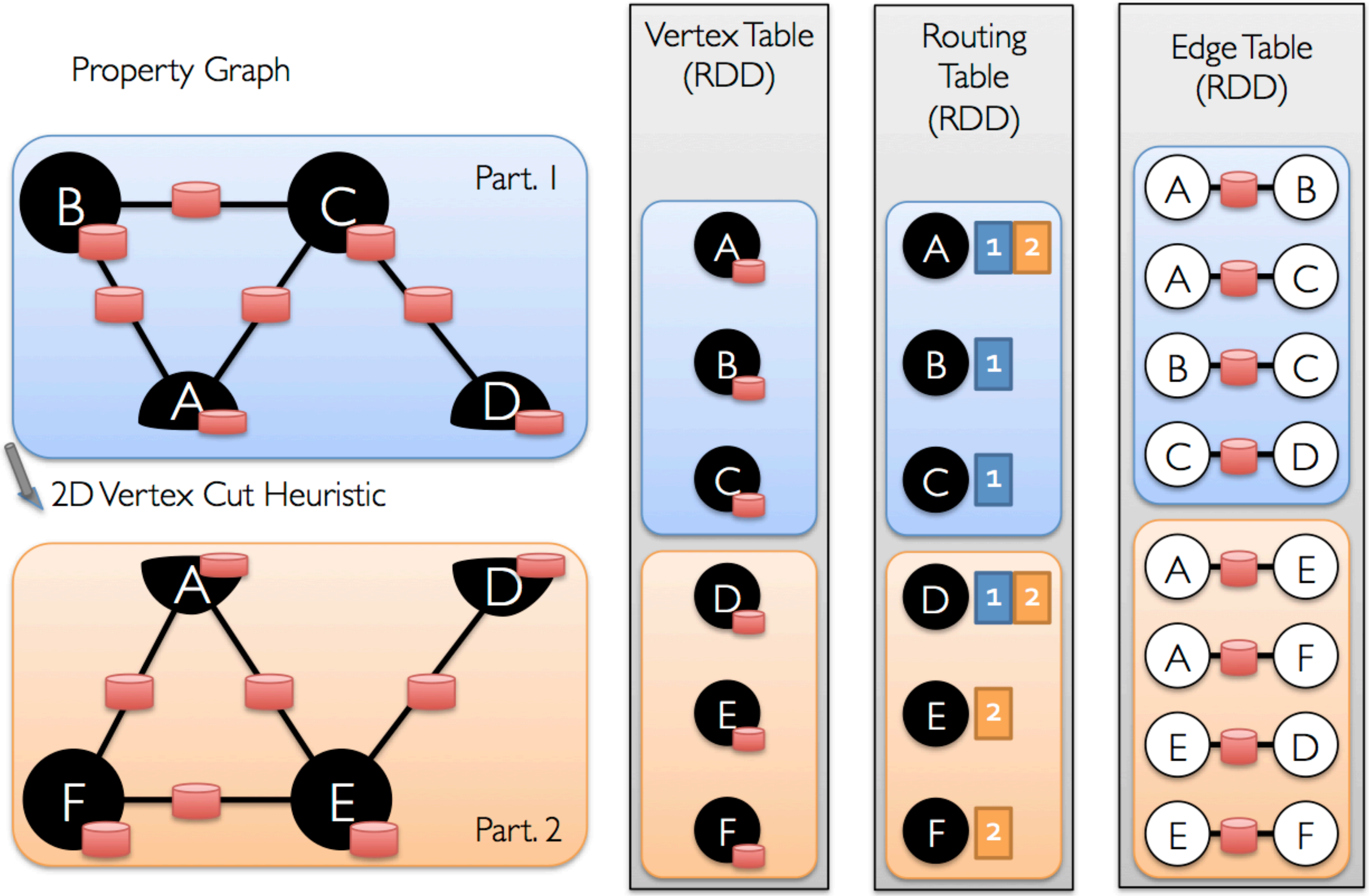
Vertex Table

Id	Property (V)
3	(rxin, student)
7	(jgonzal, postdoc)
5	(franklin, professor)
2	(istoica, professor)

Edge Table

SrcId	DstId	Property (E)
3	7	Collaborator
5	3	Advisor
2	5	Colleague
5	7	PI

Underneath the Covers



GraphX Operators

"collection" view

```
val vertices: VertexRDD[VD]
val edges: EdgeRDD[ED]
val triplets: RDD[EdgeTriplet[VD, ED]]
```

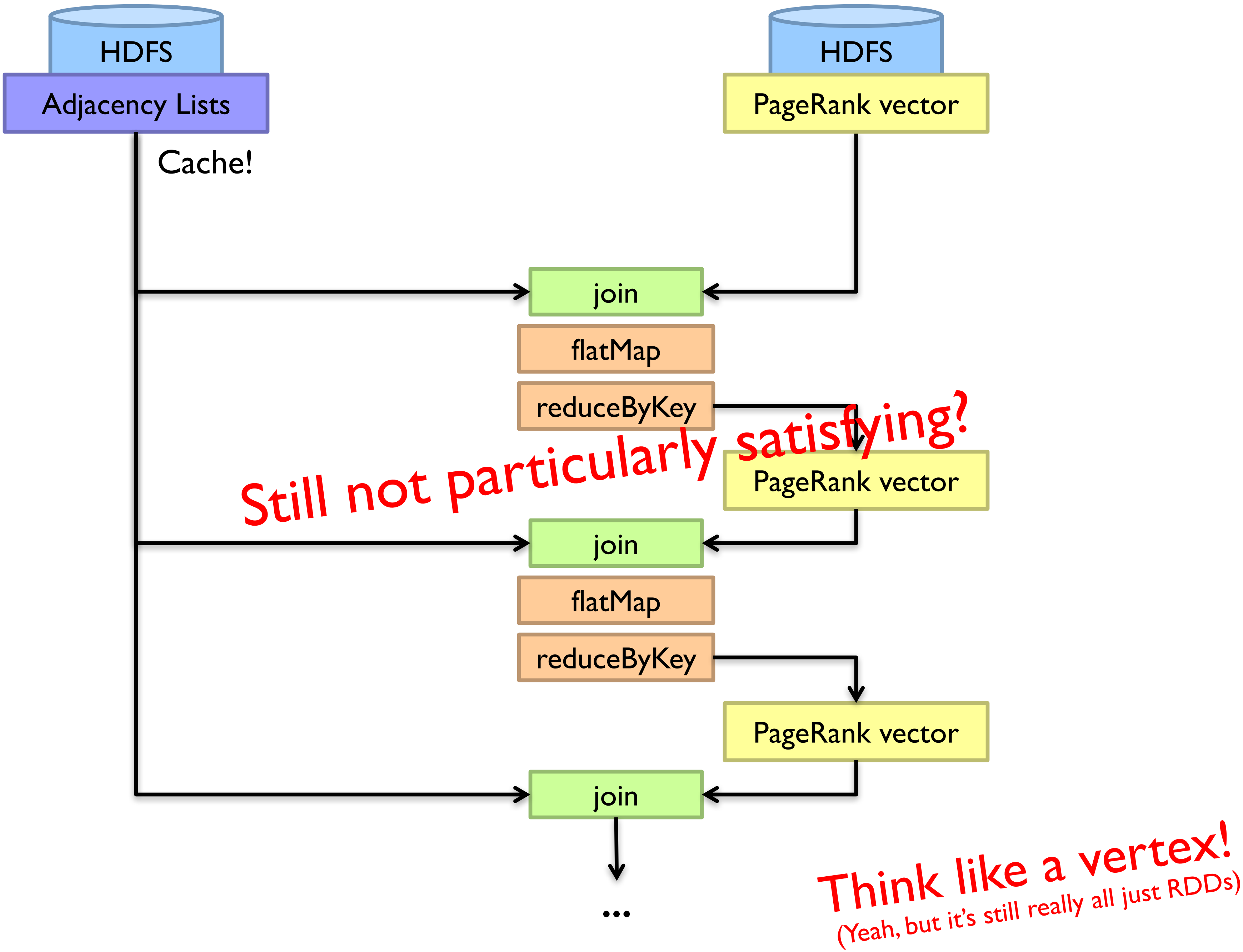
Transform vertices and edges

```
mapVertices
mapEdges
mapTriplets
```

Join vertices with external table

Aggregate messages within local neighborhood

Pregel programs


HDFS
Adjacency Lists
Cache!
HDFS
PageRank vector
join
flatMap
reduceByKey
PageRank vector
join
flatMap
reduceByKey
PageRank vector
join
…
Still not particularly satisfying?
Think like a vertex!
(Yeah, but it's still really all just RDDs)

Source: Wikipedia (Japanese rock garden)