(BIG)GRAPHS IN THE CLOUD

BIG DATA PROCESSING

Félix Cuadrado

felix.cuadrado@qmul.ac.uk

Queen Mary University of London

School of Electronic Engineering and Computer Science

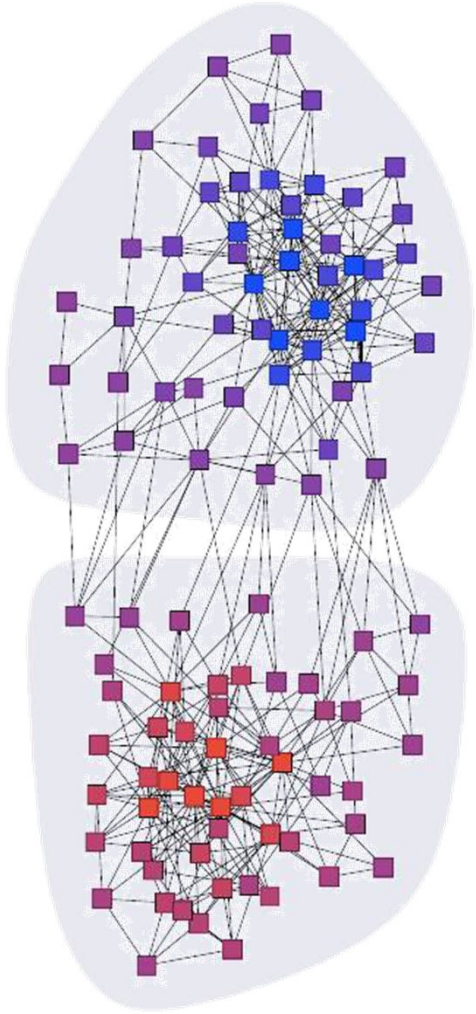
Queen Mary
University of London

Contents

- **A world of graphs**
- Storing Big Graphs
- Processing Big Graphs

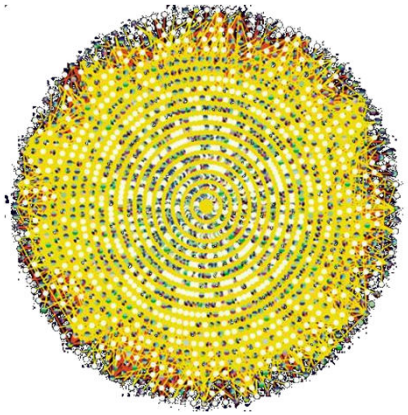
Graph Definition

- A graph G = (V,E), where
  - V represents the set of vertices (nodes)
  - E represents the set of edges (links)
  - Both vertices and edges may contain additional information
- Different types of graphs:
  - Directed vs. undirected edges
  - Presence or absence of cycles

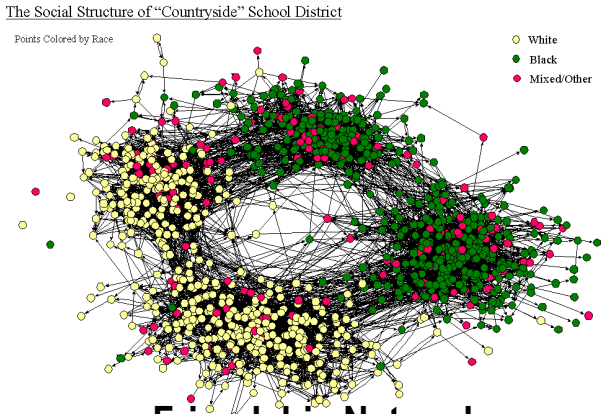


Graphs are ubiquitous

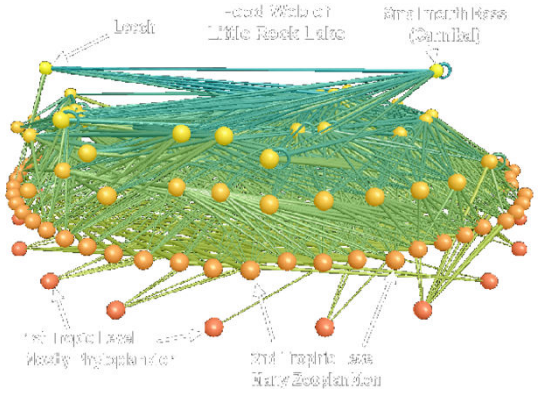
Internet Map
[lumeta.com]



Protein Interactions
[genomebiology.com]

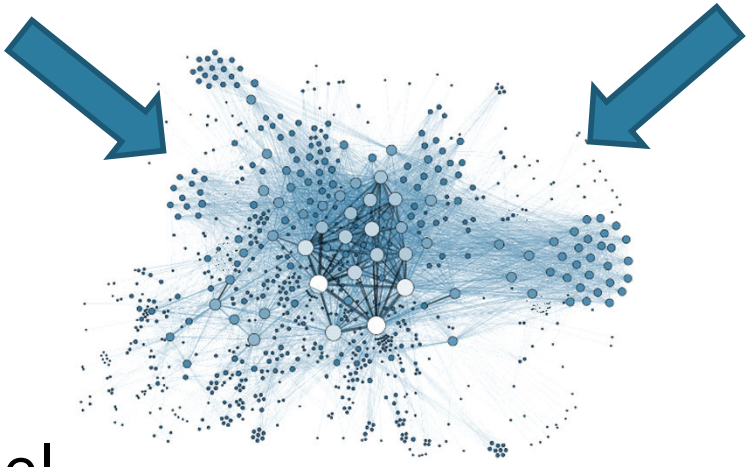


Friendship Network
[Moody '01]



Food Web [Martinez '91]

Modeling & tracking interactions

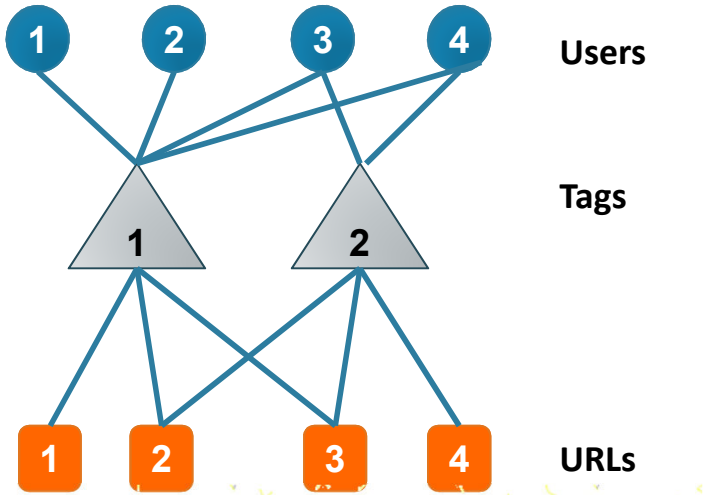


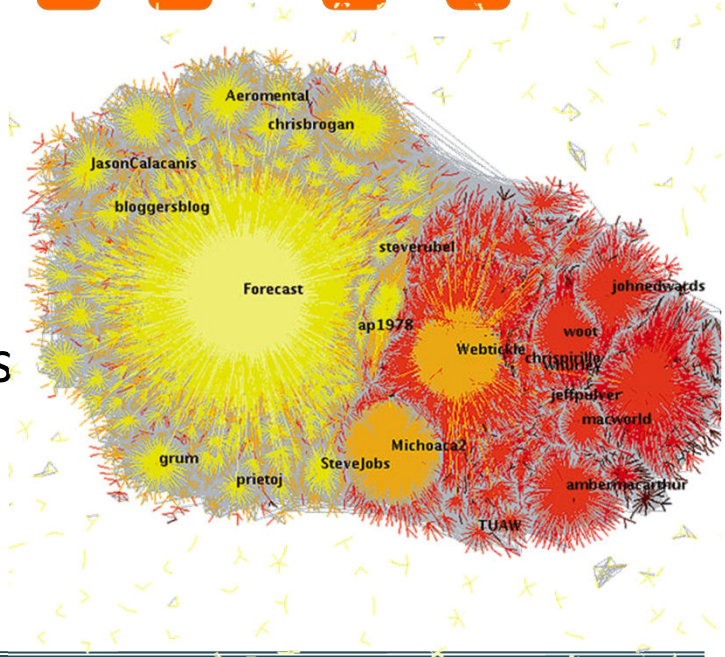
Graphs model interactions

Graph size grows with input data

Social graphs

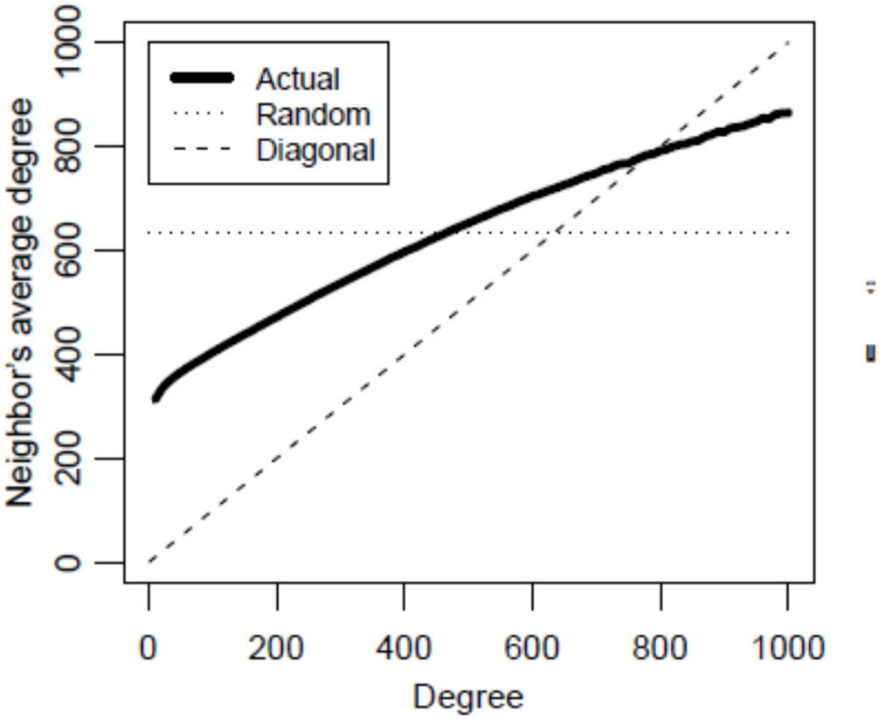
- Social media defines networks
  - Contacts
  - Messages
  - Tags
- Graph analysis allows to obtain valuable information
  - Identify leaders in a community
    - Measure influence
  - Identify “special” nodes and communities
    - Breaking up terrorist cells
    - Track spread of avian flu

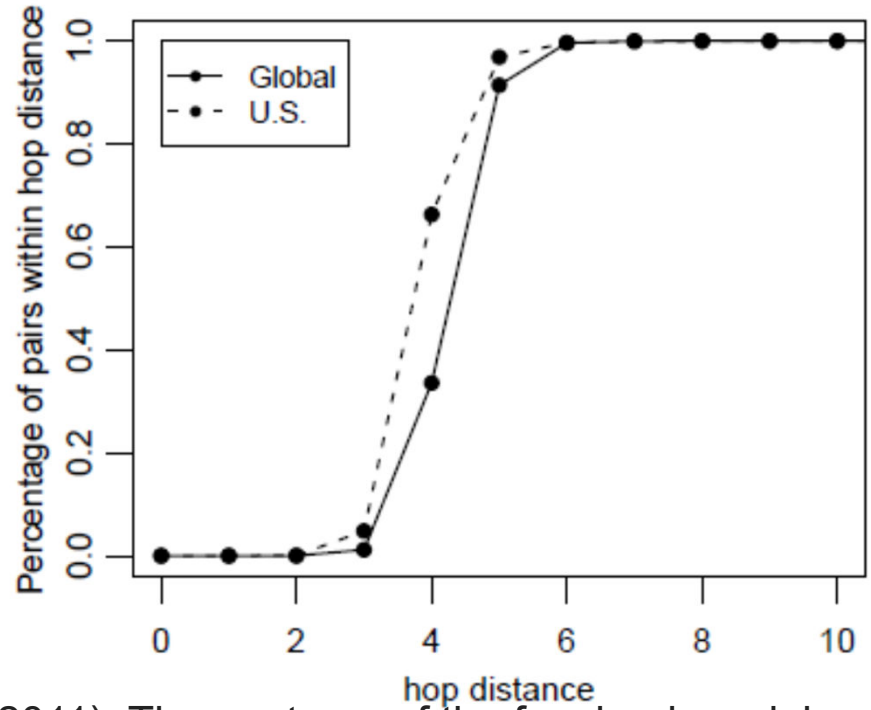




The Facebook Graph

- The power law in the degree popularity
- Replicated in many other human networks

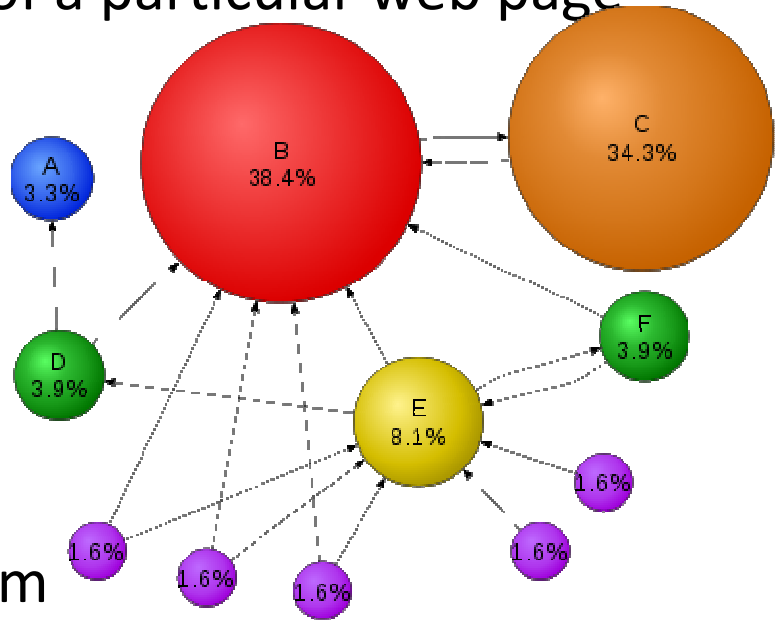




Ugander, J., Karrer, B., Backstrom, L., & Marlow, C. (2011). The anatomy of the facebook social graph. *arXiv preprint arXiv:1111.4503*.

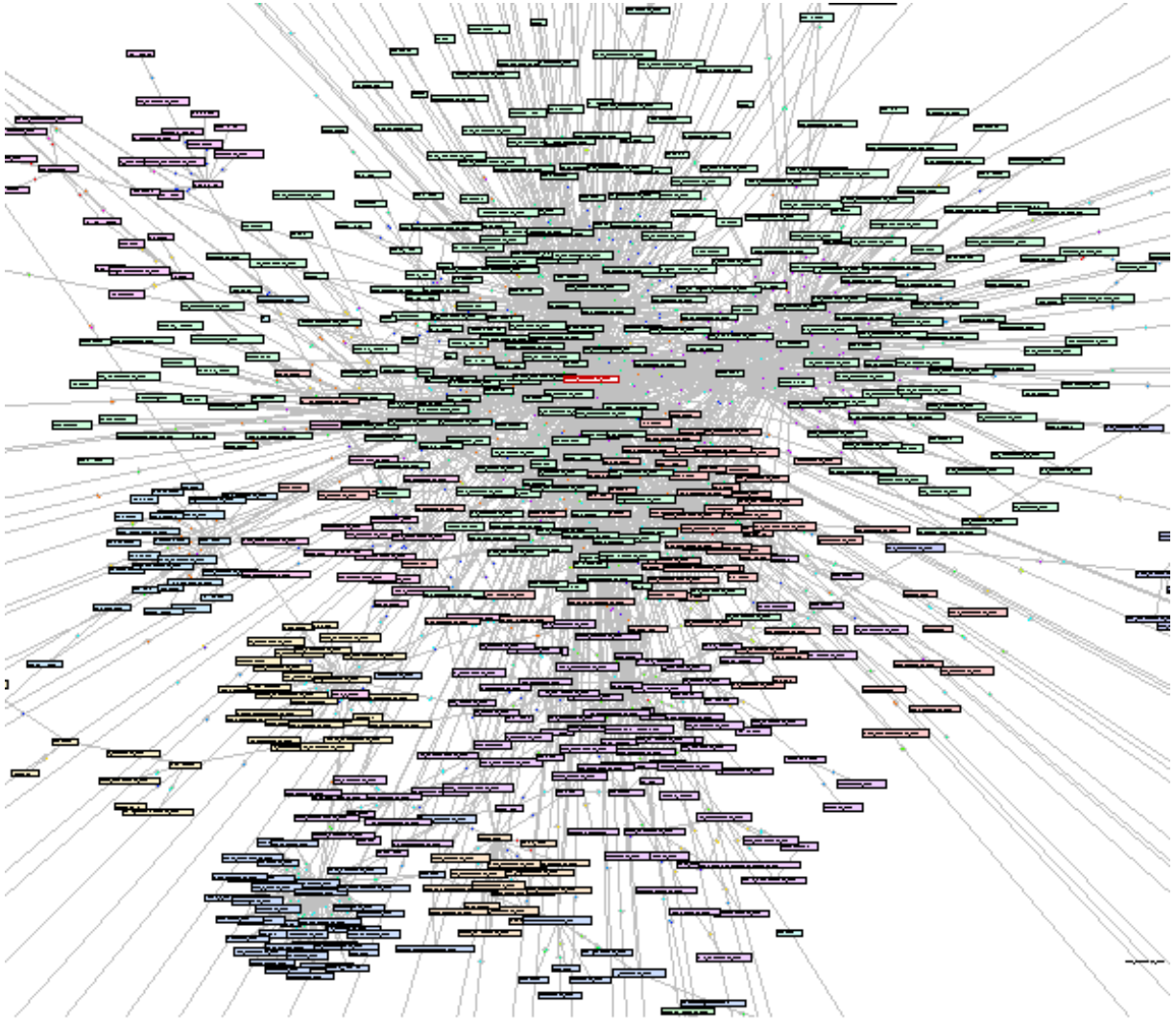
Google's PageRank

- PageRank is a link analysis algorithm
- The rank value indicates an importance of a particular web page
- A hyperlink to a page counts as a vote of support
- A page that is linked to by many pages with high PageRank receives a high rank itself
- A PageRank of 0.5 means there is a 50% chance that a person clicking on a random link will be directed to the document with the 0.5 PageRank



Page, L., Brin, S., Motwani, R., & Winograd, T. (1999). The PageRank citation ranking: bringing order to the web., WWW

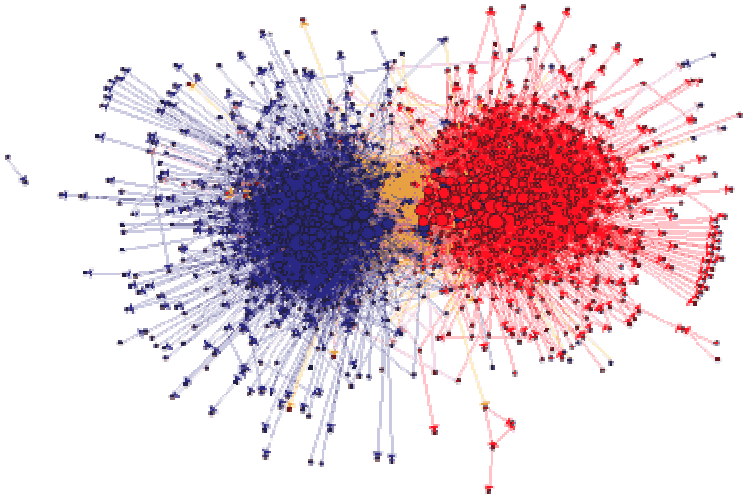
Community detection



H., Jeffrey. "Exploring Enron: Visualizing ANLP results." *Applied Natural Language Processing*. 2004.

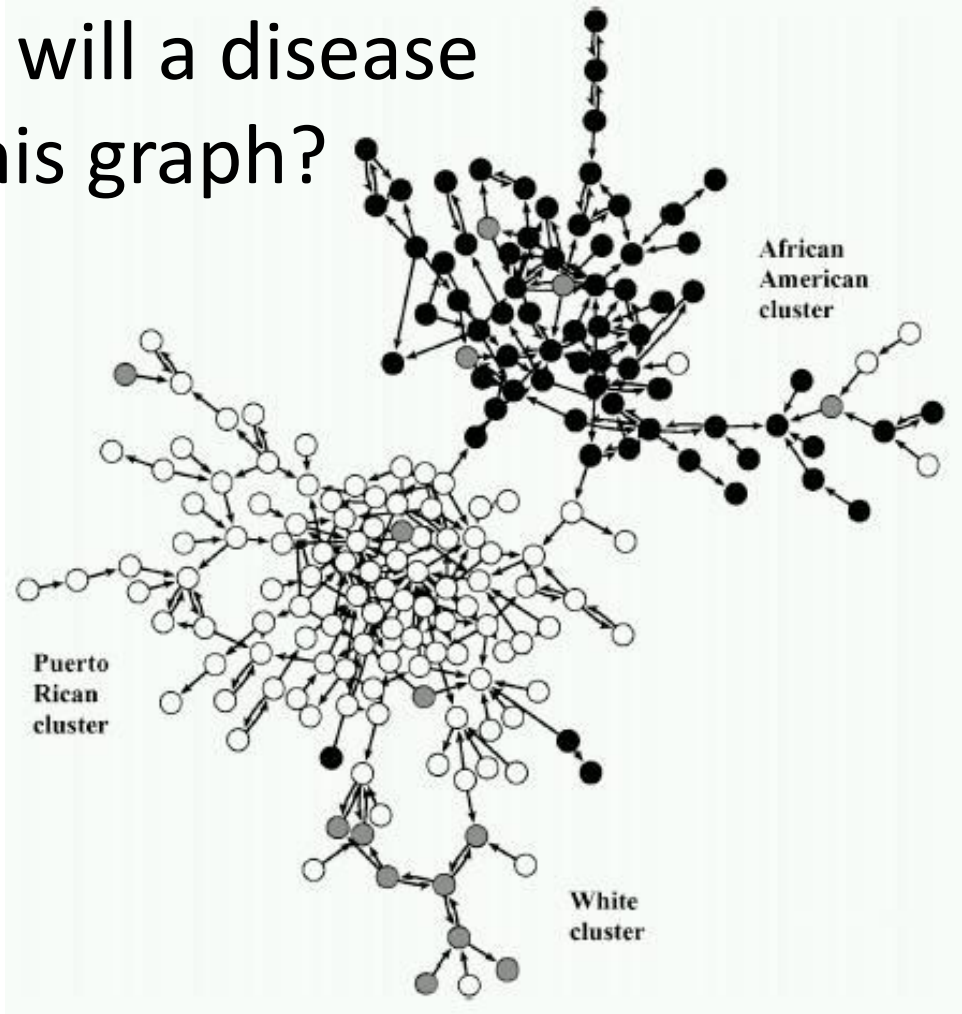
Bipartite graphs

- Nodes only can have edges to the other part
- “Stable marriage” problem
  - Monster.com, Match.com
- Web advertising, click prediction



Contagion / epidemic networks

- How quickly will a disease spread on this graph?



"Needle exchange" networks of drug users [Weeks et al. 2002]

Queen Mary University of London

Contents

- A world of graphs
- **Storing Big Graphs**
- Processing Big Graphs

Graph Management / Storage

- Traditional DBs, NoSQL DBs can store graphs
- But query languages do not support native queries on graph elements (checking for relations)
- We need query languages/abstractions suitable for finding relationship patters
- Rise of graph DBs
  - Neo4j
  - Titan

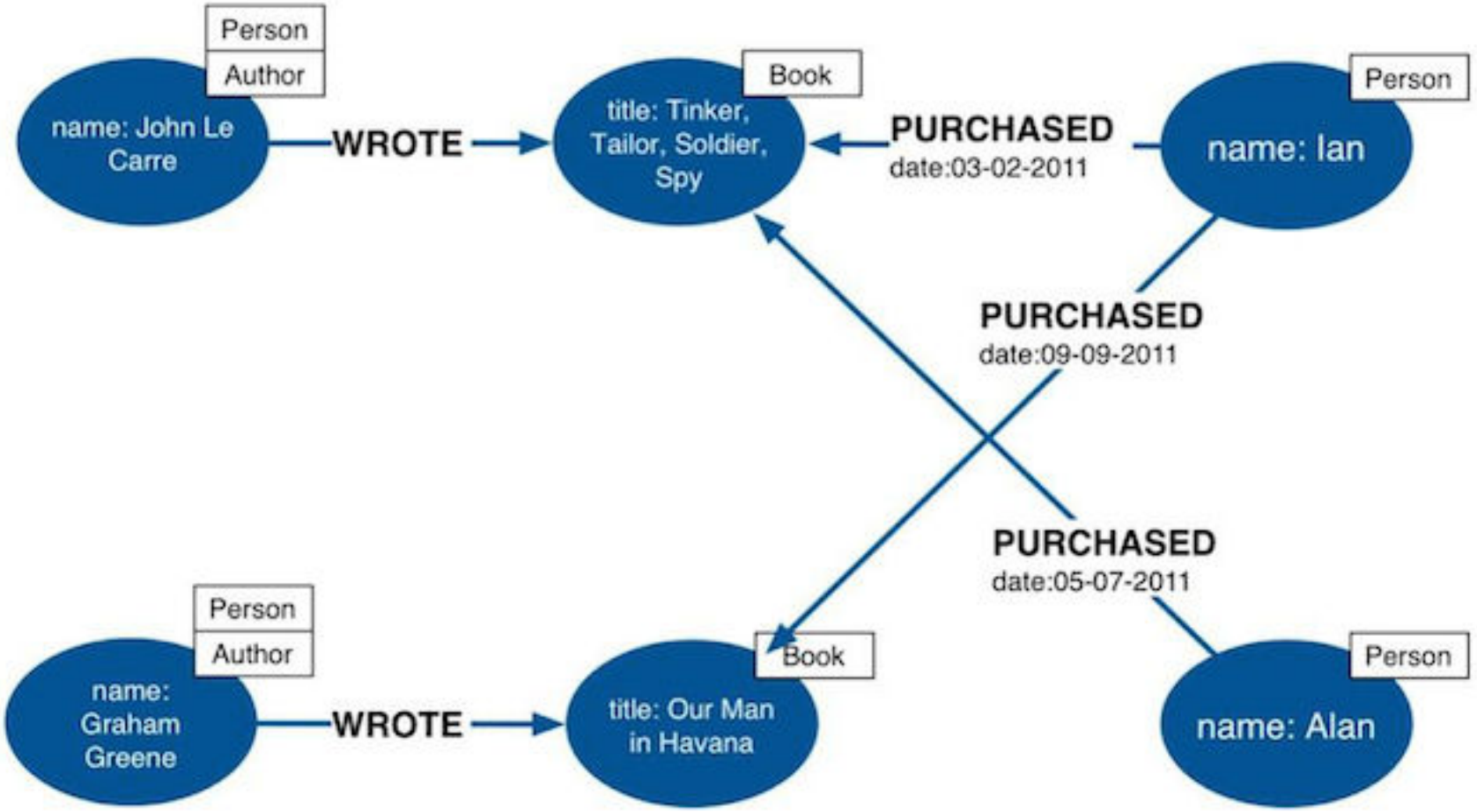
Graph Databases

- Database that uses graph structures with nodes, edges and properties to store data
- Provides index-free adjacency
  - Every node is a pointer to its adjacent element
  - Fast for following relationships
- Edges hold most of the important information and connect
  - nodes to other nodes
  - nodes to properties

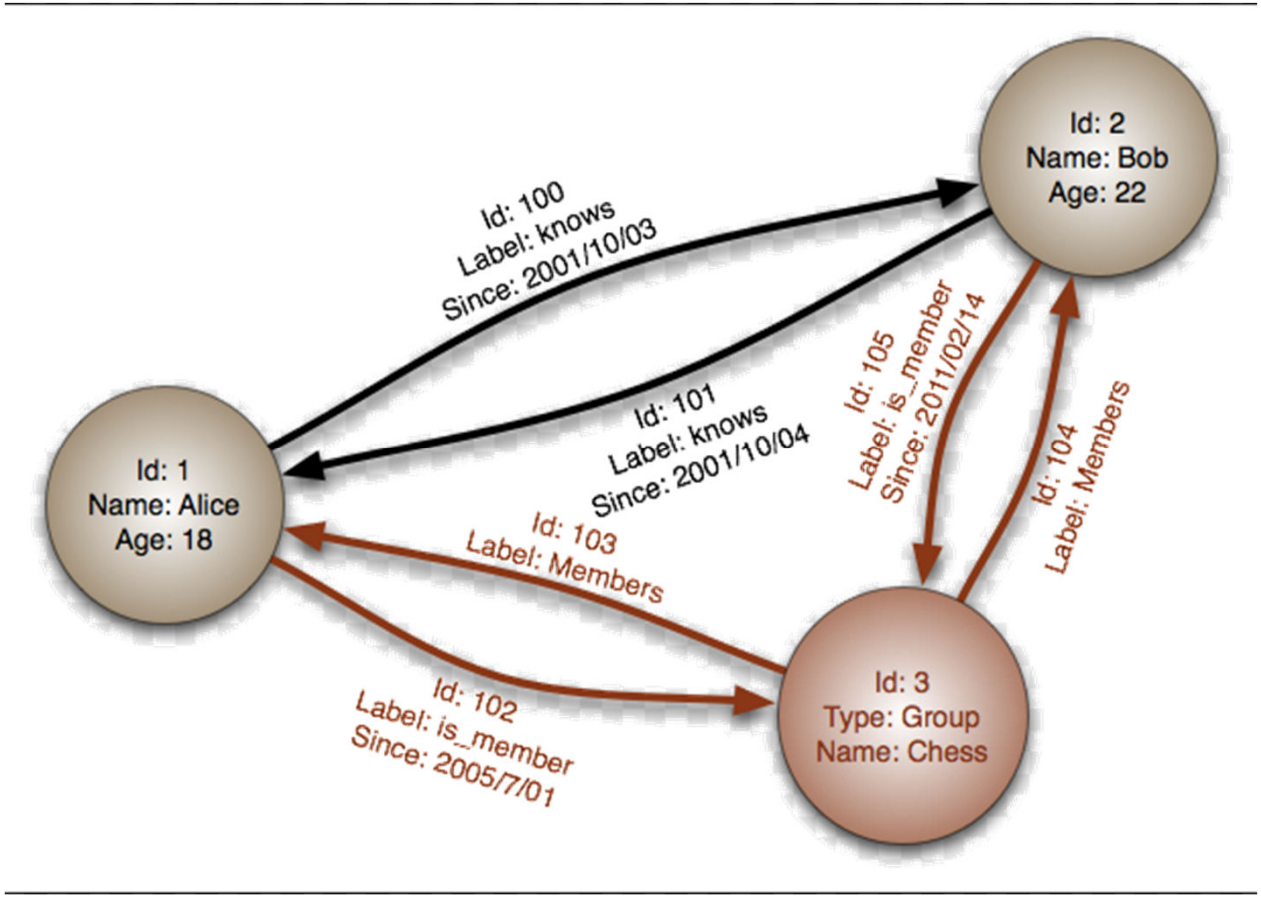
Neo4j

- Java-based graph database
- ACID – atomic, consistent, isolated and durable for logical units of work
- Property model: powerful schema-less way to model graph-based information
- Good performance for not massive datasets
- Cypher - Graph-specific query language
  - ASCII-art syntax. Define and match patterns

The property graph model



Property Graph



SQL: Modeling and Querying a Graph

Task: listing of all friends-of-a-friend-of-a-friend of mine (who is obviously not a friend of mine)

ACCOUNT_HOLDER	FRIEND_OF_HOLDER
Person001	Person002
Person001	Person006
Person002	Person004
Person004	Person005
Person006	Person007
Person007	Person008

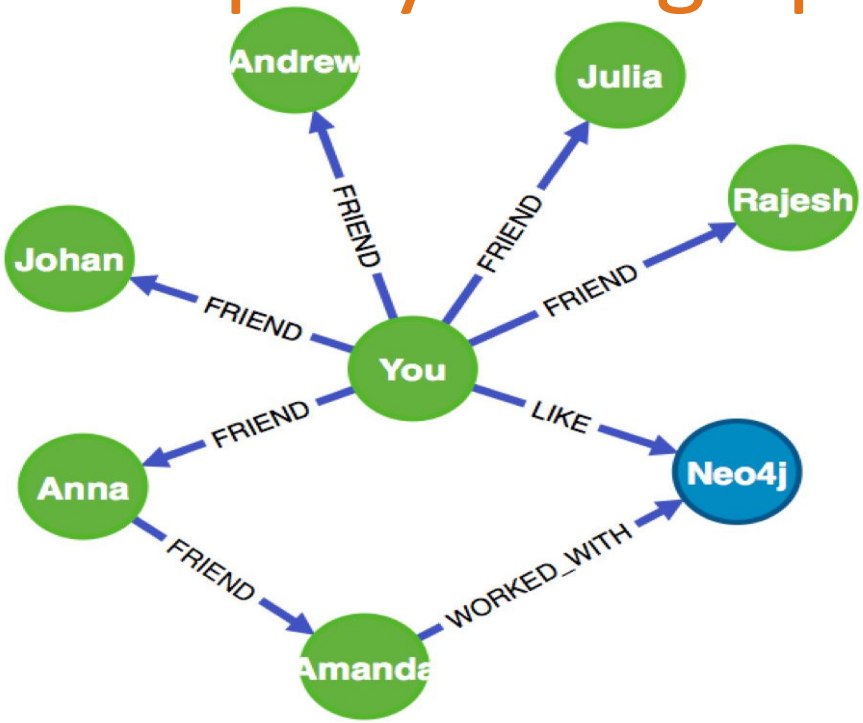
```
SQL: SELECT FRIEND_OF_HOLDER FROM TABLE_FRIENDS WHERE FRIEND_OF_HOLDER IN
(SELECT FRIEND_OF_HOLDER FROM TABLE_FRIENDS WHERE ACCOUNT_HOLDER IN
(SELECT FRIEND_OF_HOLDER FROM TABLE_FRIENDS WHERE ACCOUNT_HOLDER IN
(SELECT FRIEND_OF_HOLDER FROM TABLE_FRIENDS WHERE ACCOUNT_HOLDER = 'Person001')))
AND FRIEND_OF_HOLDER NOT IN (SELECT FRIEND_OF_HOLDER FROM TABLE_FRIENDS WHERE
ACCOUNT_HOLDER = 'Person001'); FRIEND_OF_HOLDER
```

Cypher query language

- Query Language for Neo4j
  - Becoming standard through OpenCypher initative
- Match queries, returning all the graph elements who satisfy all the
- Expressive, does not require joining or other SQL

```
MATCH [a:Person {ID: “Person001”}]-[:IS_A_FRIEND_OF*3]->[b:Person]
WHERE NOT (a)-[:IS_A_FRIEND_OF]-(b)
RETURN DISTINCT b.id;
```

Sample Cypher query on a graph




```
MATCH (you {name:"You"})
MATCH (expert)-[:WORKED_WITH]->(db:Database {name:"Neo4j"})
MATCH path = shortestPath( (you)-[:FRIEND*..5]-(expert) )
RETURN db,expert,path
```

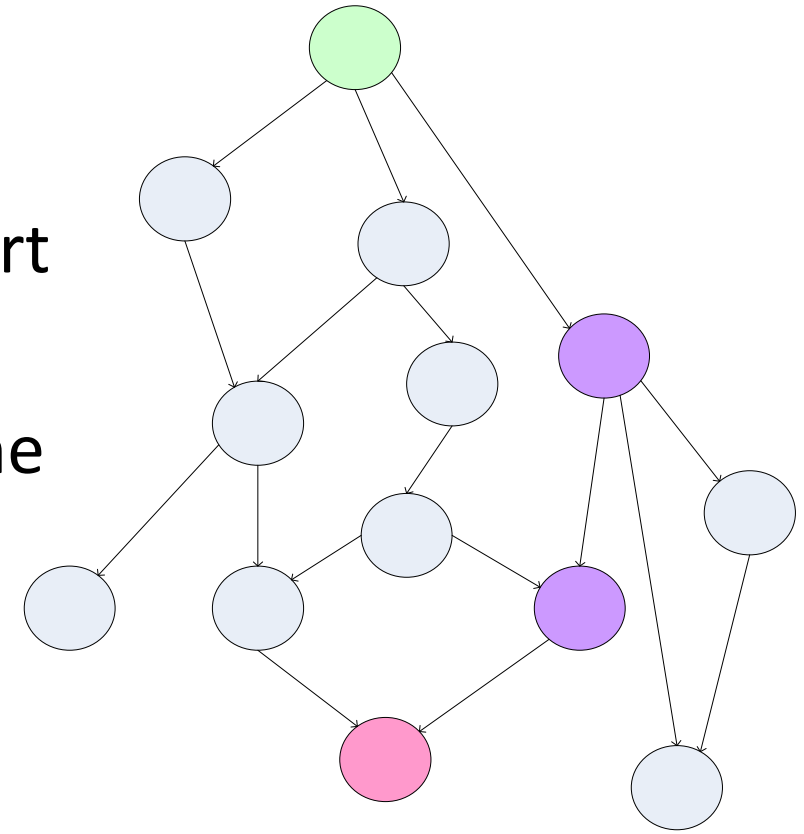
Queen Mary University of London

Contents

- A world of graphs
- Storing Big Graphs
- **Processing Big Graphs**

Sample graph problem: Finding the Shortest Path

- SSSP: The Single-Source shortest path problem.
- Common graph search application: finding the shortest path from a start node to one or more target nodes
- Commonly done on a single machine with *Dijkstra's Algorithm*



Dijkstra's algorithm

```
dist[s] ←o                          (distance to source vertex is zero)
for all v ∈ V–{s}
      do dist[v] ←∞                 (set all other distances to infinity)
S←∅                                 (S, the set of visited vertices is initially empty)
Q←V                                 (Q, the queue initially contains all
vertices)
while Q ≠∅                          (while the queue is not empty)
do  u ← mindistance(Q,dist)         (select the element of Q with the min.
                                    distance)
    S←S∪{u}                         (add u to list of visited vertices)
     for all v ∈ neighbors[u]
          do if  dist[v] > dist[u] + w(u, v)          (if new shortest path found)
                then    d[v] ←d[u] + w(u, v) (set new value of shortest path)
                    (if desired, add traceback code)
return dist
```

Graph representation

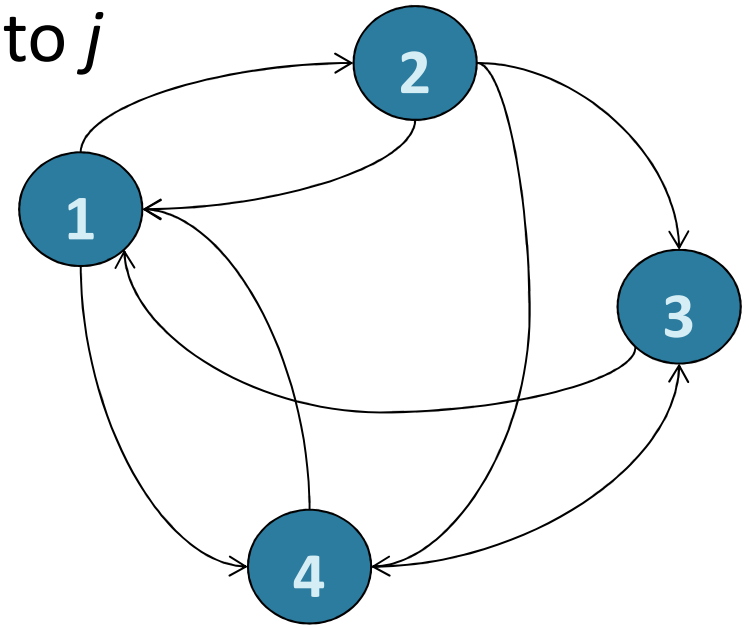
- Algorithms need to keep the information about how graph is represented
  - Structure, and attributes of edges and vertices
  - Both for serialization, and for loading in memory
  - Example: edge list
    - 1 4
    - 2 3
    - 5 2
    - 6 1
    - 1 4

Graph Representation: Adjacency Matrices

Represent a graph as an *n* x *n* square matrix *M*

- $n = |V|$
- $M_{ij} = 1$ means a link from node *i* to *j*

	1	2	3	4
1	0	1	0	1
2	1	0	1	1
3	1	0	0	0
4	1	0	1	0



Adjacency Matrices: Critique

- Advantages:
  - Naturally encapsulates iteration over nodes
  - Rows and columns correspond to inlinks and outlinks
- Disadvantages:
  - Sparse graphs. A graph $G = (V,E)$ is sparse if $|E|$ is much smaller than $|V|^2$.
  - Lots of zeros for sparse matrices
  - Lots of wasted space

Graph Traversal in MapReduce

- Approach: Parallel processing of each vertex
  - Each Map/Reduce function has access to limited info
    - One node and its links
- **Iterate** executions of a MapReduce job
  - Map: compute something on each node. Potentially send information to that/other nodes that is aggregated in Reducers.
  - Reducers compute something on each node
  - The output of the reducers in iteration #n becomes the input of the mappers in iteration #n+1

Equal Weight SSPS

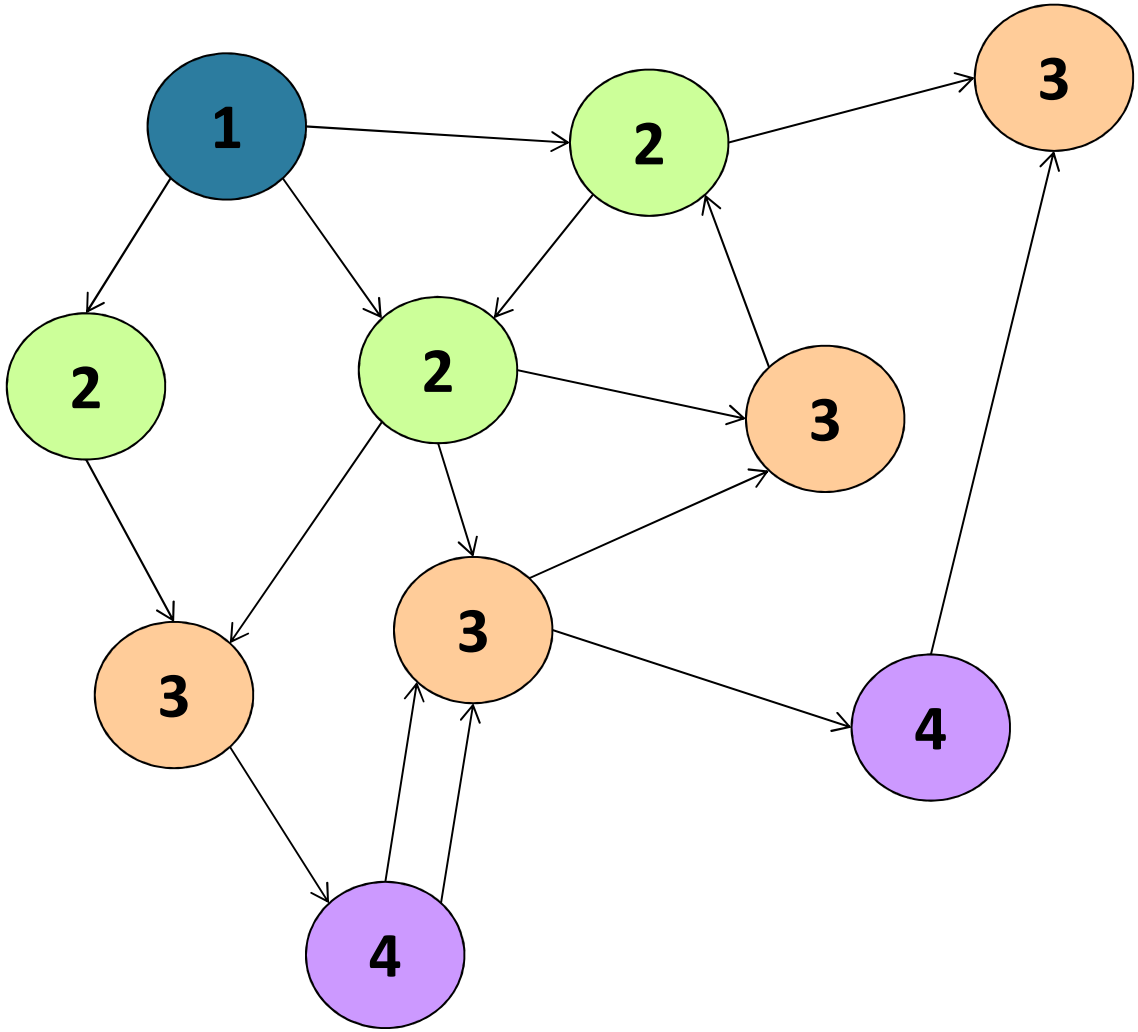
- Problem: from an origin node, find the shortest distance to every other node of the graph, with all links having the same distance
  - Also known as the Kevin Bacon number problem
- Small world phenomenom
  - The world is tied together by a network of personal relationships.
  - There's a popular hypothesis called *Six Degrees of Separation.*

Finding the Shortest Path: Intuition

- Breadth-First Search (BFS) algorithm
- We can define the solution to this problem inductively:
  - distanceTo(startNode) = 0
  - For all nodes n directly reachable from startNode,
    distanceTo(n) = 1
  - For all nodes n reachable from some other set of nodes S,

    distanceTo(n) = 1 + min(distanceTo(m), m $\in$ S)

Visualizing Parallel BFS



MapReduce graph processing performance

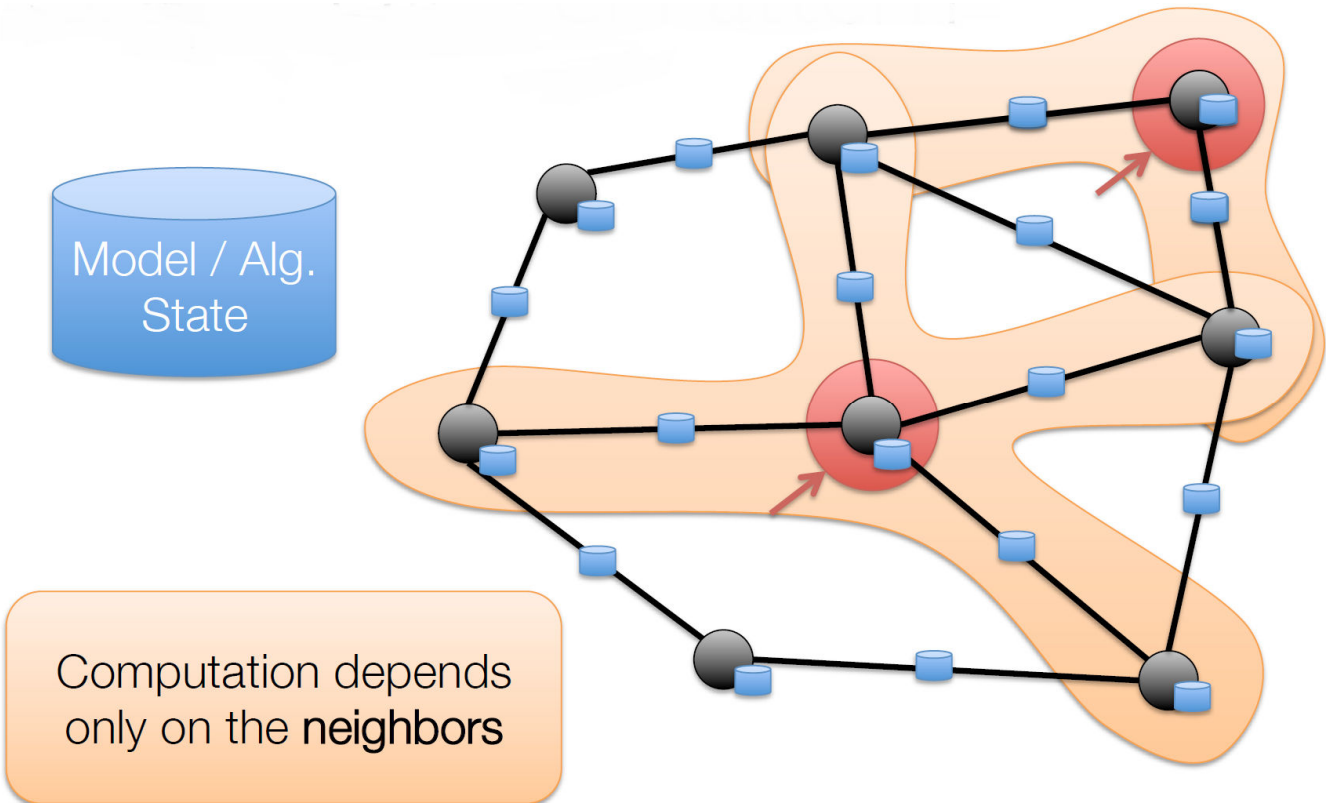
- Iterative algorithms involve HDFS writing in each step
  - Resending the graph structure in each iteration is **VERY inefficient**
- One Map task per node, and sending of messages to other nodes depending on connections results in significant communications cost.
- In-memory systems are a much better fit for this type of computation
  - Graph-specific in-memory sytsems have developed recently

Google's Pregel model: Think like a vertex [4]

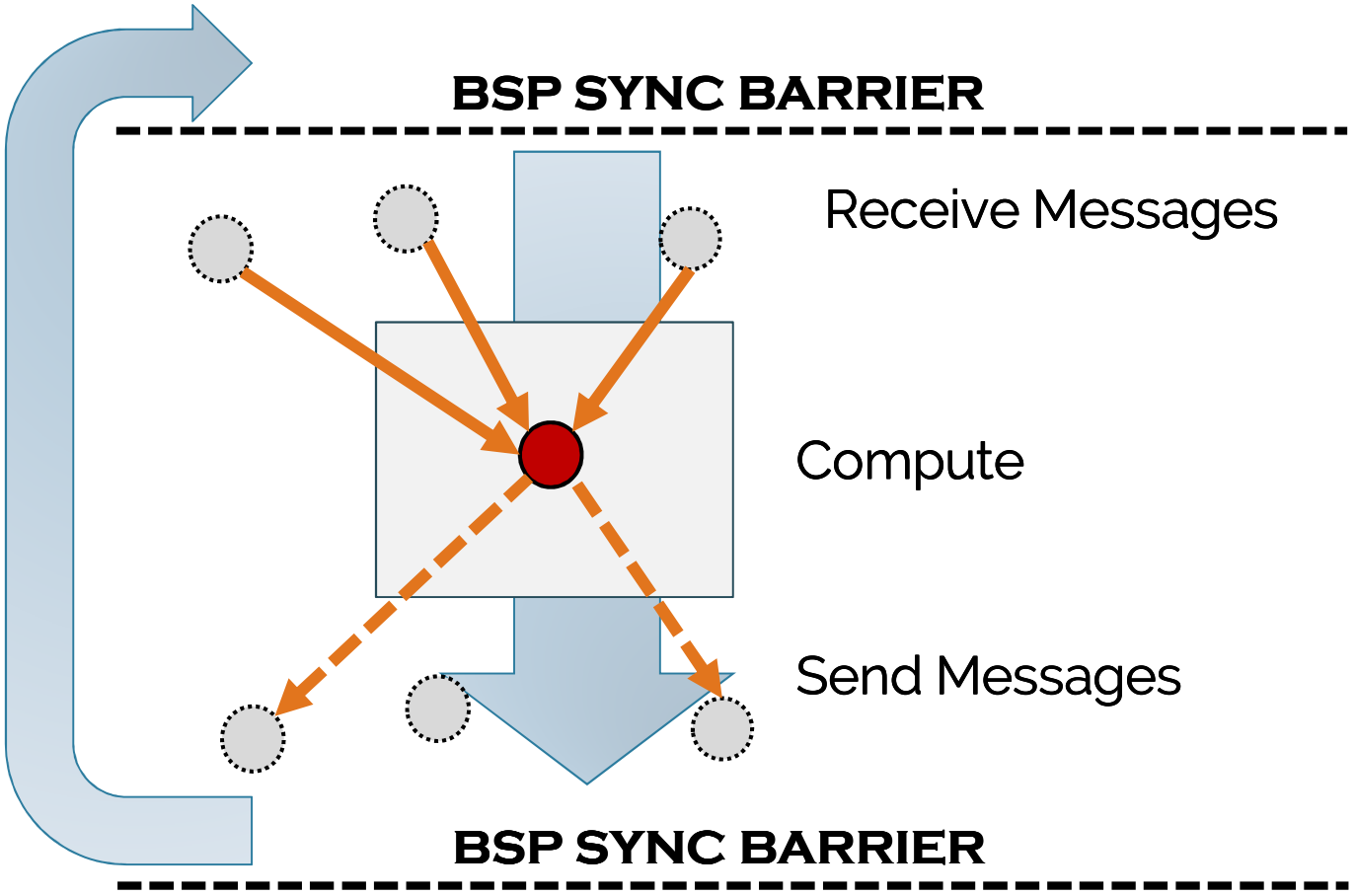
- Computation is iterative
  - Every iteration, a function that is executed at each vertex
    - Vertices can send messages to its neighbours
    - Messages arrive in the next iteration
- Computation is executed in parallel
  - Each vertex is independent from the rest in the same step
  - Messages are the synchronization mechanism

[4] Malewicz, G., Austern, M. H., Bik, A. J., Dehnert, J. C., Horn, I., Leiser, N., & Czajkowski, G. (2010, June). Pregel: a system for large-scale graph processing. In *Proceedings of the ACM SIGMOD*

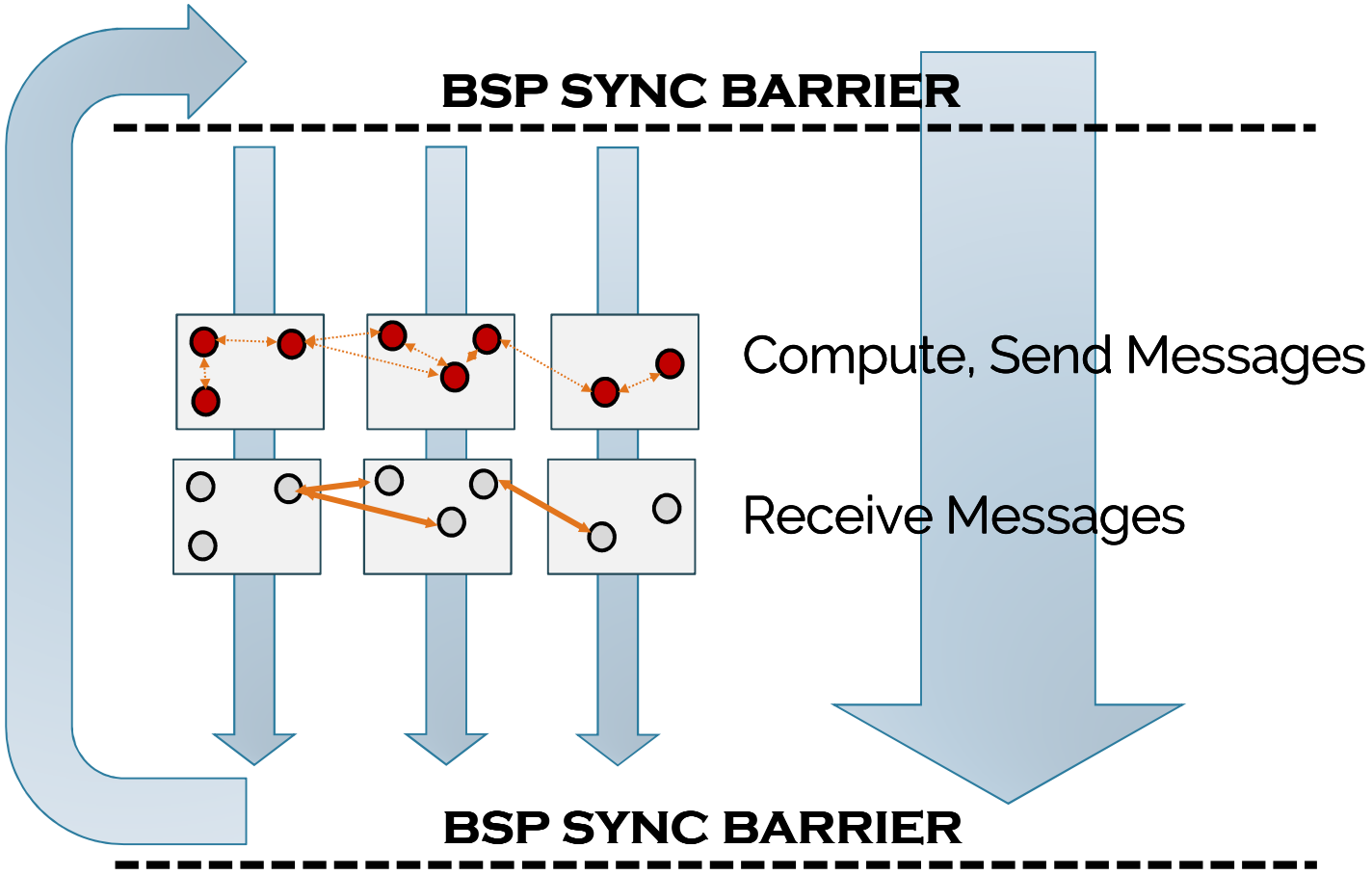
Pregel: The world of a vertex



Think like a vertex



Pregel's node-centric processing model



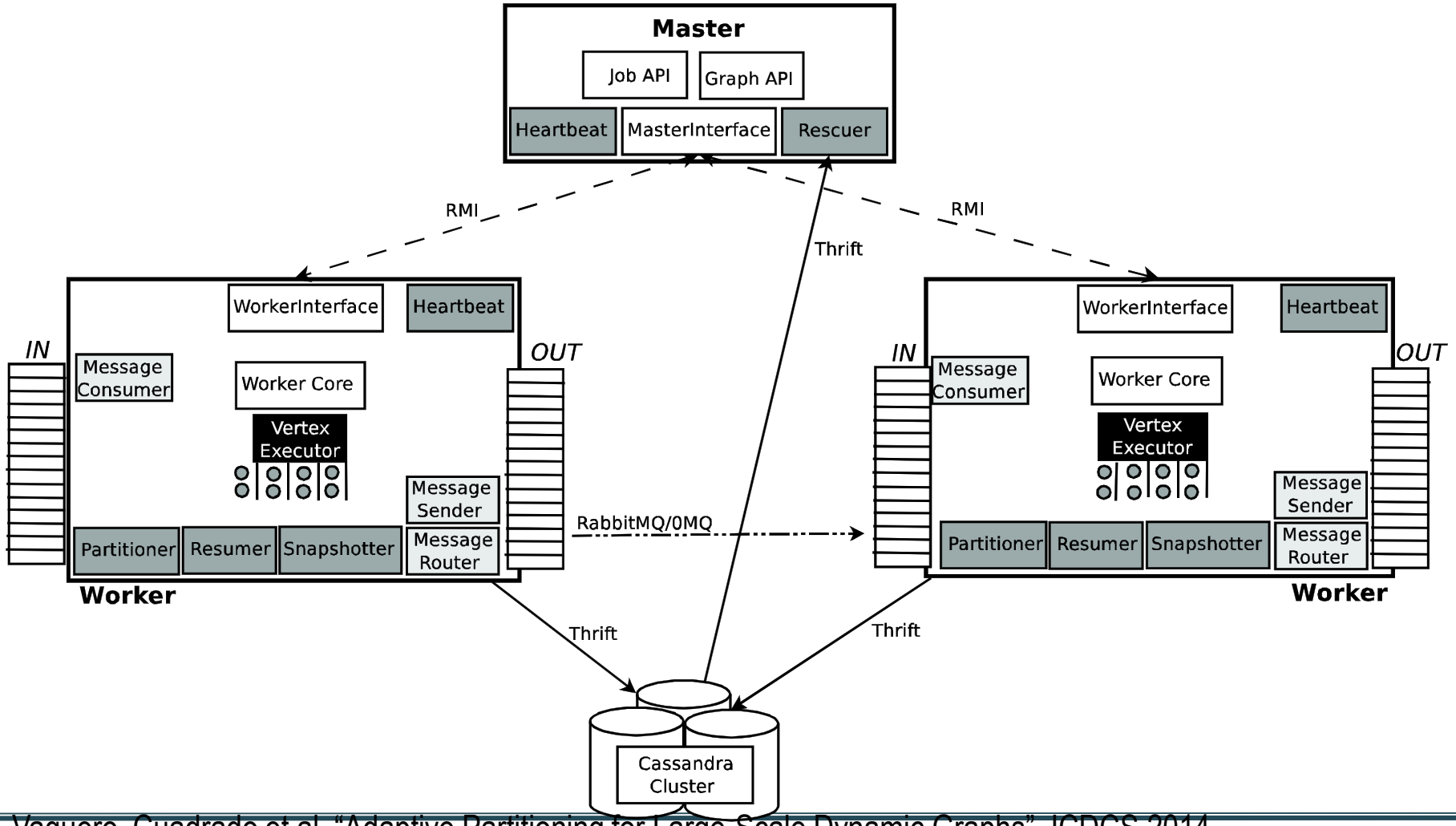
Pregel-style graph processing systems

- Pregel
  - Original Google paper
- Apache Giraph
  - Java-based
- Apache Spark GraphX
  - Extension of Spark with
- …Ongoing research efforts in this space

Example: PageRank in Apache Giraph

```
public void compute(Vertex <LongWritable, DoubleWritable, FloatWritable> vertex,
                                                    Iterable messages) throws IOException{
    if (getSuperstep() >=1 ) {
       double sum = 0;
       for (DoubleWritable message : messages) {
          sum += message.get();
       }
       DoubleWritable vertexValue =
             new DoubleWritable((0.15f / getTotalNumVertices()) + 0.85f * sum);
       vertex.setValue(vertexValue);
    }
    if (getSuperstep() < MAX_SUPERSTEPS) {
       long edges = vertex.getNumEdges();
       sendMessageToAllEdges(vertex,
                 new DoubleWritable(vertex.getValue().get() / edges));
    } else {
       vertex.voteToHalt();
    }
}
```

GraHPa: Dynamic distributed graph processing



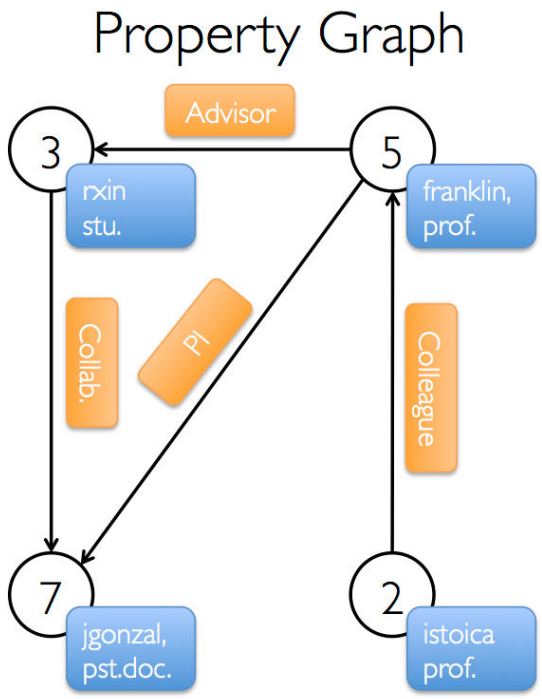
Vaquero, Cuadrado et.al, "Adaptive Partitioning for Large-Scale Dynamic Graphs", ICDCS 2014

Spark GraphX

- Spark library for graph processing
- Provides specialized RDDs for representing graph structure, as well as its information (property graphs
- Provides methods for creating graph, transforming them, implementing multiple common graph metrics and algorithms

GraphX Property Graphs

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



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

```
val userGraph: Graph[(String, String), String]
```

GraphX RDD

- Holds graph data and provides methods
- VertexRDD[VertexId, VertexData]
  - Vertice IDs have to be longs
    - String to hash conversion
  - Holds vertex properties
- EdgeRDD [EdgeData]
  - Holds source, destination, edge properties
  - Technically a directed multigraph
- Triplets
  - Join of source vertex, destination vertex and edge

Graph creation in GraphX

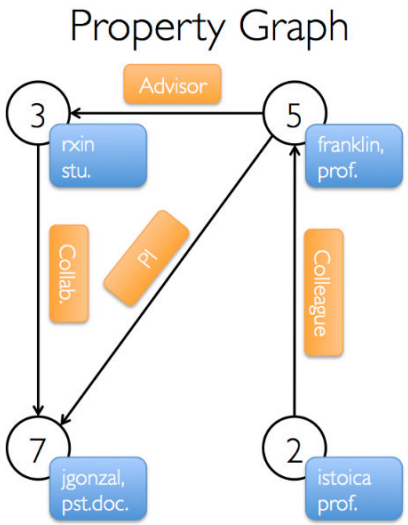
- Graphs can be created from a file, format must be a list of edges in text format. Values will be long numbers
  - GraphLoader.edgeListFile("hdfs://……")
- From RDDs with the right information about nodes and edges.
  - Graph.fromEdges(<<EdgeRDD>>)
  - Graph.fromEdgeTuples(<<RDD[(VertexId, VertexId)] >>)
- Possible to specify how many partitions, how the graph is partitioned

GraphX predefined methods

- A graph RDD has multiple convenience methods that provide access to its information and implement relevant operations
- Graph.vertices, graph.edges, graph.triples
  - Access to RDDs with the property information
- Graph.degrees
  - Provides a tuple with (vertexId, degree of each vertex)
- Graph.connectedComponents
  - Obtains each of the connected components of the graph

Transforming edges and vertices

```
graph.vertices.map(v => v._2._2).collect
//returns (student, postdoc, professor, professor)
 graph.edges.filter ( e => e._3.equals("PI").count
//returns 1
```



Vertex Table

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

Edge Table

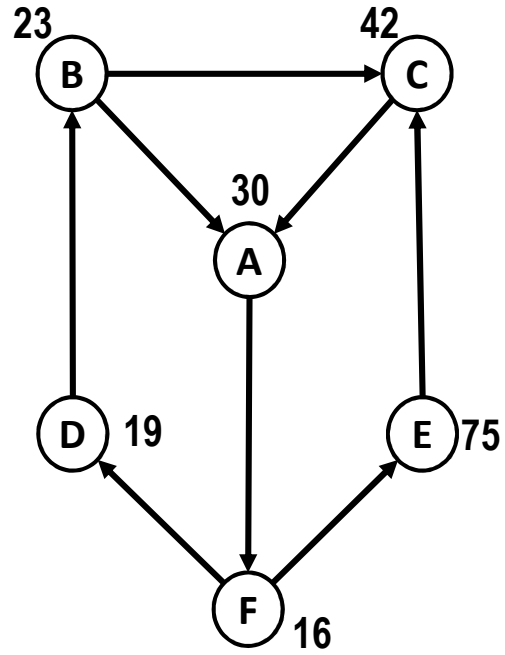
SrcId	DstId	Property (E)
3	7	Collaborator
5	3	Advisor
2	5	Colleague
5	7	PI

Graph aggregate computation

- aggregateMessages transformation allows to send and process messages through every a message through each edge
  - sendMsg: EdgeContext[VD, ED, Msg] => Unit
    - Can send messages to either src or destination, using context
  - mergeMsg: (Msg, Msg) => Msg
    - All the receiving messages to a vertex are reduced into one
- Returns a tuple of (vertexId, results)

Age of the oldest follower of each node

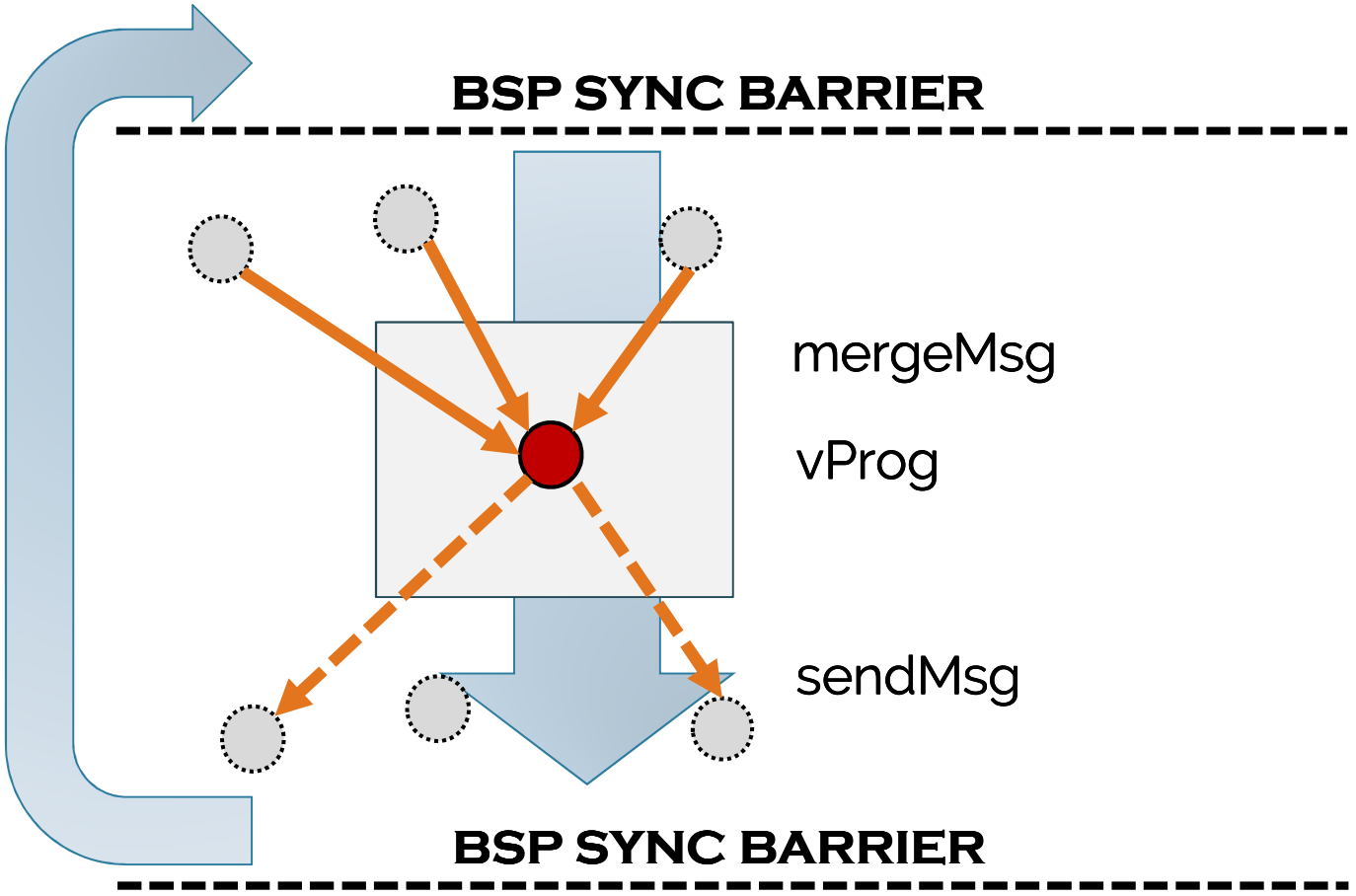
```
val olderFollowers: VertexRDD[(Int, Double)] =
  graph.aggregateMessages[(Int, Double)](
    // sendMessages
    edge=>  edge.sendToDst(edge.srcAttr) },
    //mergeMessages
    (a, b) => math.max(a,b)
)
```



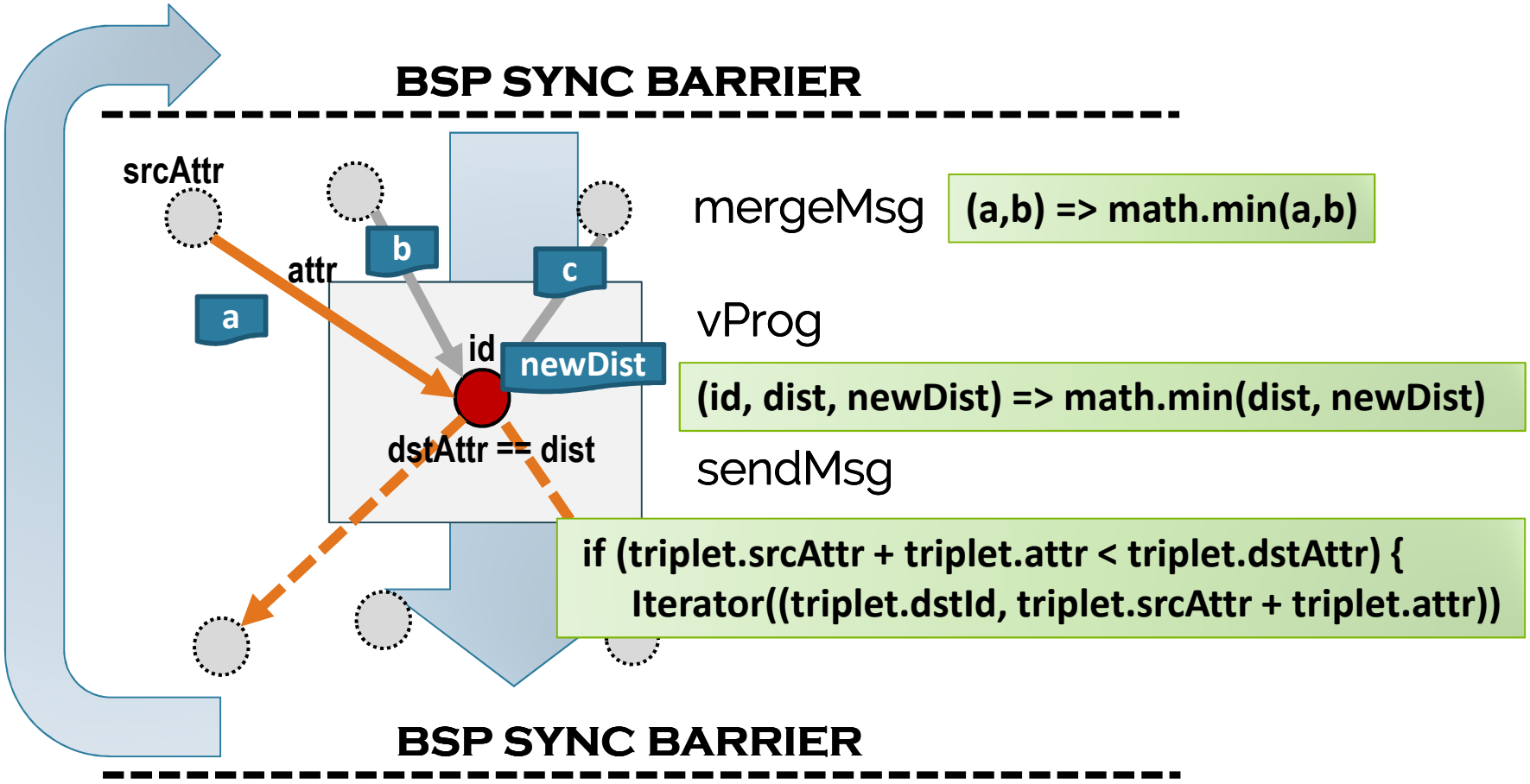
Pregel programming in GraphX

- GraphX provides a Pregel transformation for iterative graph traversals
- Need to specify number of iterations
- Three functions for implementing Pregel's model
  - vprog: (VertexID, VD, A) => VD
    - Updates vertex properties (VD)
  - sendMsg: EdgeTriplet[VD, ED] => Iterator[(VertexID,A)]
    - Sends messages to vertex neighbours
  - mergeMsg: (A, A) => A)
    - Combines all incoming messages into a single one
- Returns new graph with updated vertex values

GraphX Pregel model



GraphX Pregel model (2)



Breadth First Search in Pregel

```
val initialGraph = graph.mapVertices((id, _) => if (id == sourceId) 0.0 else
Double.PositiveInfinity)
val sssp = initialGraph.pregel(Double.PositiveInfinity)(
 (id, dist, newDist) => math.min(dist, newDist), // Vertex Program
 triplet => {  // Send Message
  if (triplet.srcAttr + triplet.attr < triplet.dstAttr) {
   Iterator((triplet.dstId, triplet.srcAttr + triplet.attr))
  } else {
   Iterator.empty
  }
 },
 (a,b) => math.min(a,b) // Merge Message
 )
```