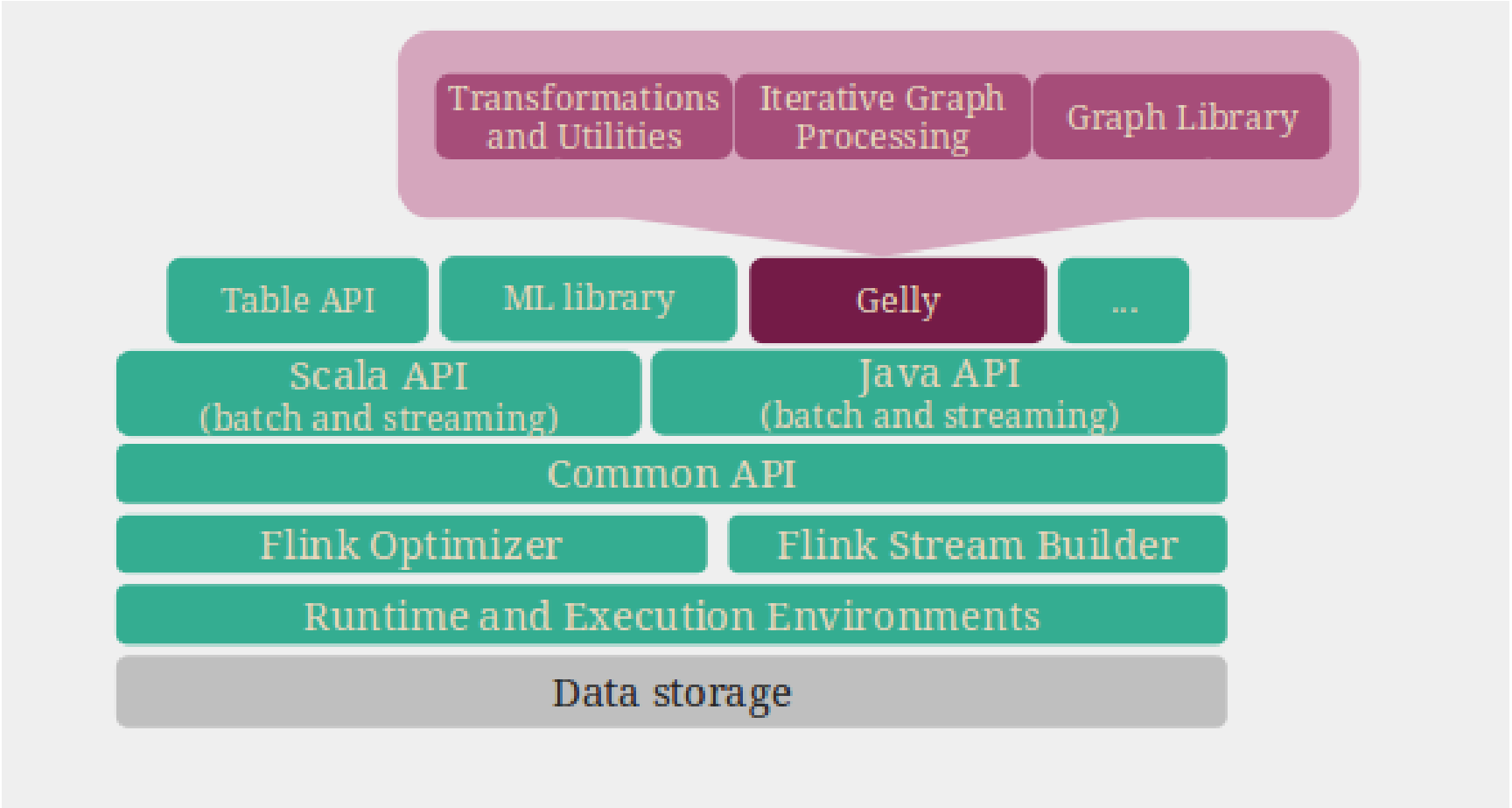
Large-Scale Graph Processing with Apache Flink

Andra Lungu
Flink contributor
andra.lungu
@campus.tu-berlin.de

What is Gelly?

- Large-scale graph processing API
- On top of Flink's Java API
- Official release: Flink 0.9
- Off-the shelf library methods
- Supports record and graph analysis applications; iterative algorithms

The Growing Flink Stack



How to use Gelly?

Graph Creation

```
DataSet<Edge<Long, Double>> edges = getEdgesDataSet(env);

Graph<Long, Double, Double> graph = Graph.fromDataSet(edges,
                new MapFunction<Long, Double>() {

                        public Double map(Long value) {
                                return Double.MAX_VALUE;
                        }
}, env);
```

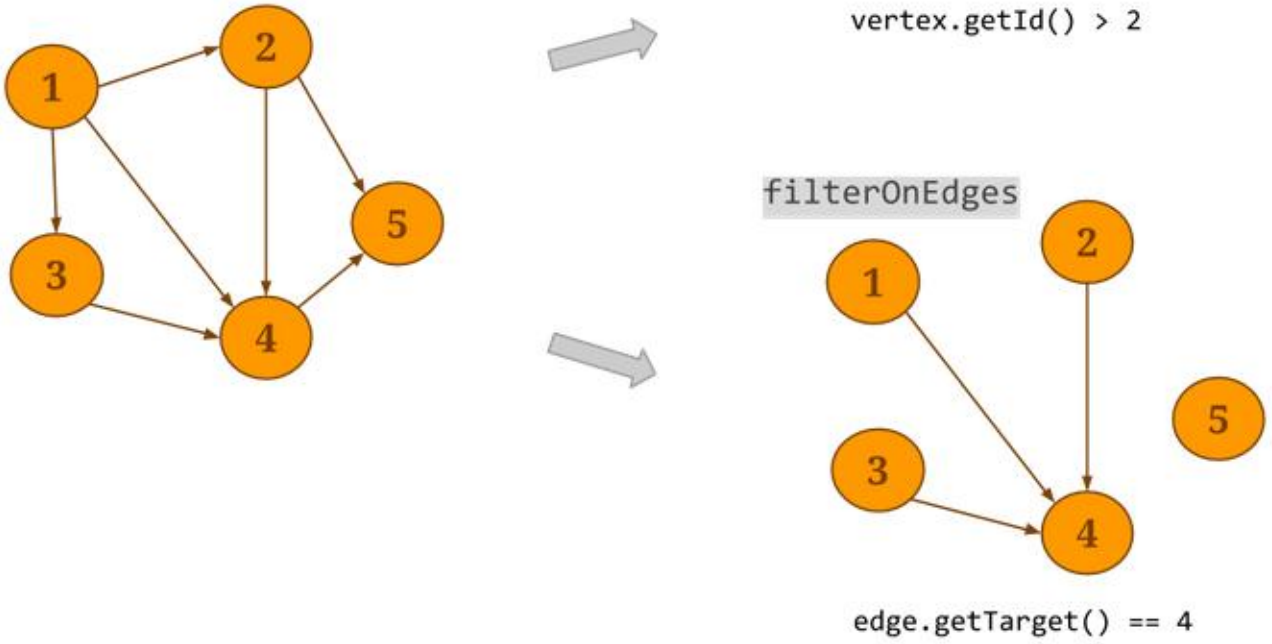
Graph Properties

- getVertices()
- getEdges()
- getVertexIds()
- getEdgeIds()
- inDegrees()
- outDegrees()
- getDegrees()
- numberOfVertices()
- numberOfEdges()
- isWeaklyConnected(int maxIterations)
- getTriplets()

Graph Transformations

- Map
  - **mapVertices**(final MapFunction<Vertex<K, VV>, NV> mapper)
  - **mapEdges**(final MapFunction<Edge<K, EV>, NV> mapper)
- Filter
  - **filterOnVertices**(FilterFunction<Vertex<K, VV>> vertexFilter)
  - **filterOnEdges**(FilterFunction<Edge<K, EV>> edgeFilter)
  - **subgraph**(FilterFunction<Vertex<K, VV>> vertexFilter, FilterFunction<Edge<K, EV>> edgeFilter)

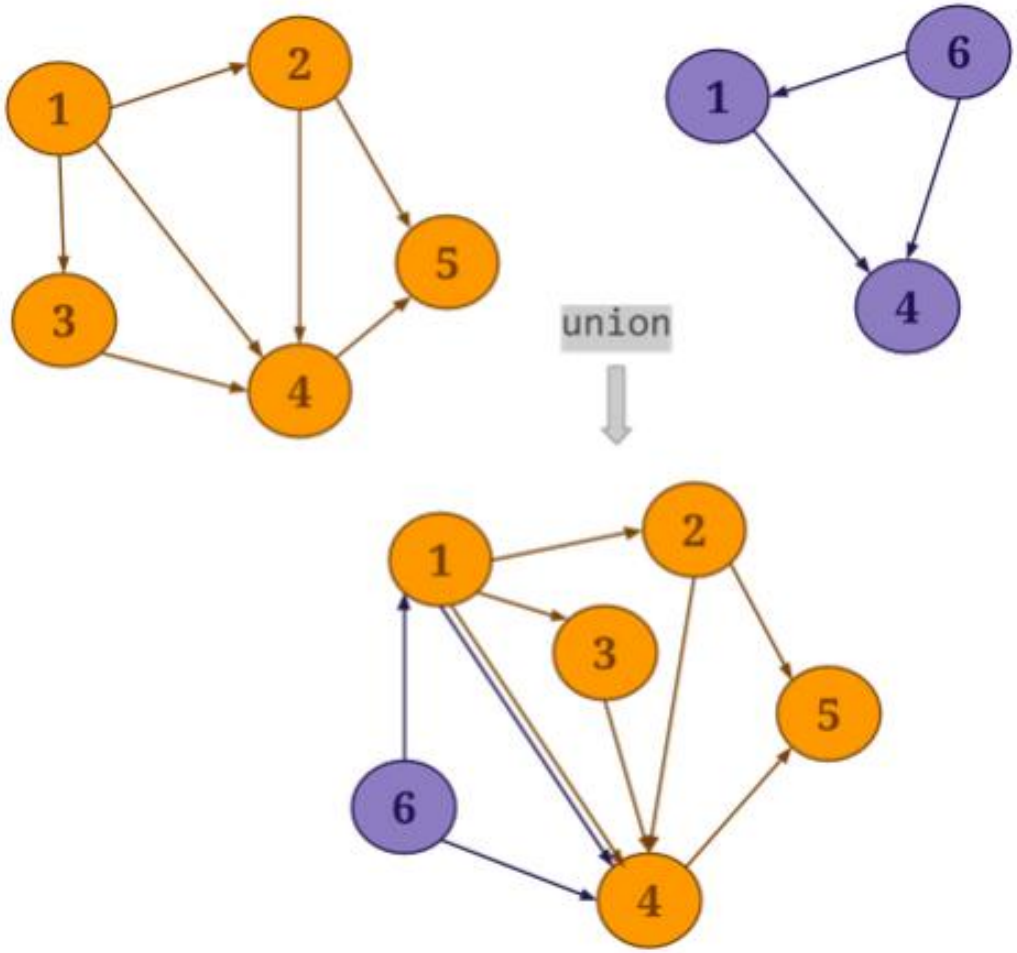
Filter Functions



Graph Transformations

- Join
  - **joinWithVertices**(DataSet<Tuple2<K, T>> inputDataSet, final MapFunction<Tuple2<VV, T>, VV> mapper)
  - **joinWithEdges**(DataSet<Tuple3<K, K, T>> inputDataSet, final MapFunction<Tuple2<EV, T>, EV> mapper)
  - **joinWithEdgesOnSource** / **joinWithEdgesOnTarget**
- Reverse
- Undirected

Union

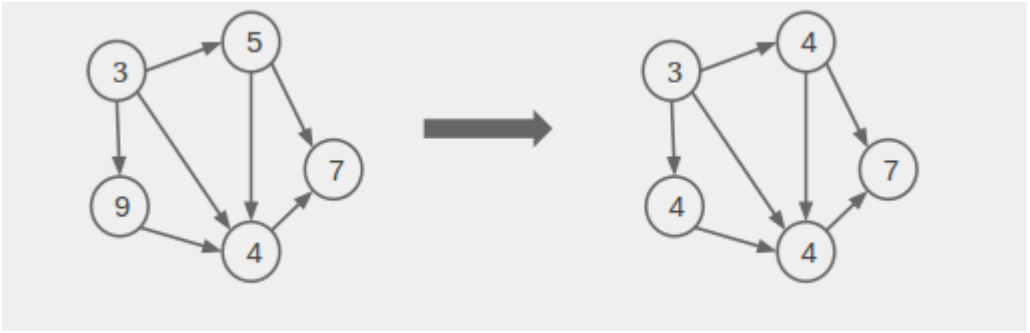


Graph Mutations

- **addVertex**(final Vertex<K, VV> vertex, List<Edge<K, EV>> edges)
- **addEdge**(Vertex<K, VV> source, Vertex<K, VV> target, EV edgeValue)
- **removeVertex**(Vertex<K, VV> vertex)
- **removeEdge**(Edge<K, EV> edge)

Neighborhood Methods

- **reduceOnNeighbors**(reduceNeighborsFunction, direction)



- reduceOnEdges
- groupReduceOnNeighbors; groupReduceOnEdges

Graph Validation

- Given criteria:
  - Edge IDs correspond to vertex IDs

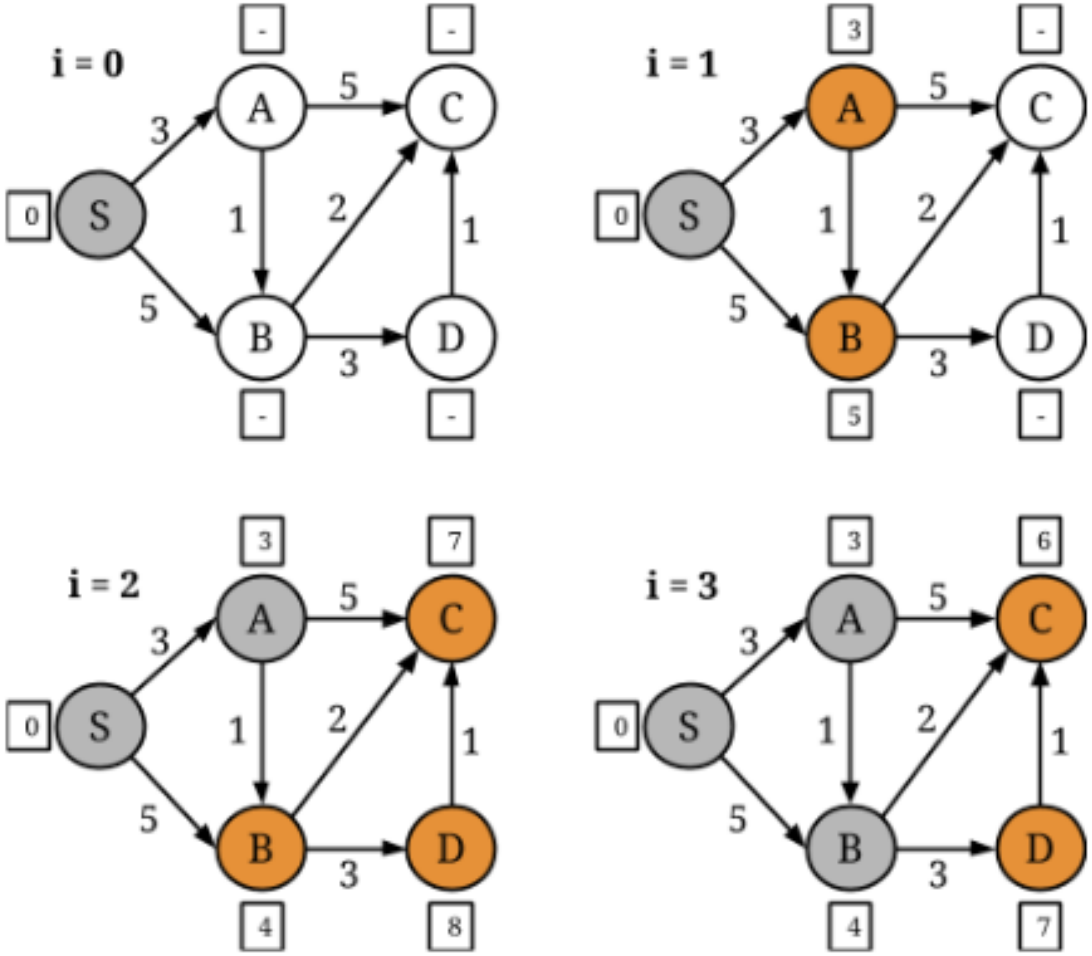
```
edges = { (1, 2), (3, 4), (1, 5), (2, 3), (6, 5) }
vertices = { 1, 2, 3, 4, 5 }

graph = Graph.fromCollection(vertices, edges);
graph.validate(new InvalidVertexIdsValidator()); // false
```

Vertex-centric Iterations

- Pregel [BSP] Execution Model
- UDFs:
  - Messaging Function
  - VertexUpdateFunction

- S-1: receive messages from neighbors
- S: update vertex values
- S+1: send new value to neighbors

Single Source Shortest Paths



SSSP – code snippet

```
shortestPaths = graph.runVertexCentricIteration(
              new DistanceUpdater(), new DistanceMessenger()).getVertices();
```

DistanceUpdater: VertexUpdateFunction

```
updateVertex(K key, Double value,
             MessageIterator msgs) {

  Double minDist = Double.MAX_VALUE;
  for (double msg : msgs) {
    if (msg < minDist)
      minDist = msg;
  }
    if (value > minDist)
      setNewVertexValue(minDist);
}
```

DistanceMessenger: MessagingFunction

```
sendMessages(K key, Double newDist) {

for (Edge edge : getOutgoingEdges()) {
  sendMessageTo(edge.getTarget(),
      newDist + edge.getValue());
}
```

Library of Algorithms

- Weakly Connected Components
- Community Detection
- Page Rank
- Single Source Shortest Paths
- Label Propagation

Gelly Hands-On

LiveJournal Social Network

- journals, individual blogs

Dataset statistics	
Nodes	4847571
Edges	68993773
Nodes in largest WCC	4843953 (0.999)
Edges in largest WCC	68983820 (1.000)
Nodes in largest SCC	3828682 (0.790)
Edges in largest SCC	65825429 (0.954)
Average clustering coefficient	0.2742
Number of triangles	285730264
Fraction of closed triangles	0.04266
Diameter (longest shortest path)	16
90-percentile effective diameter	6.5

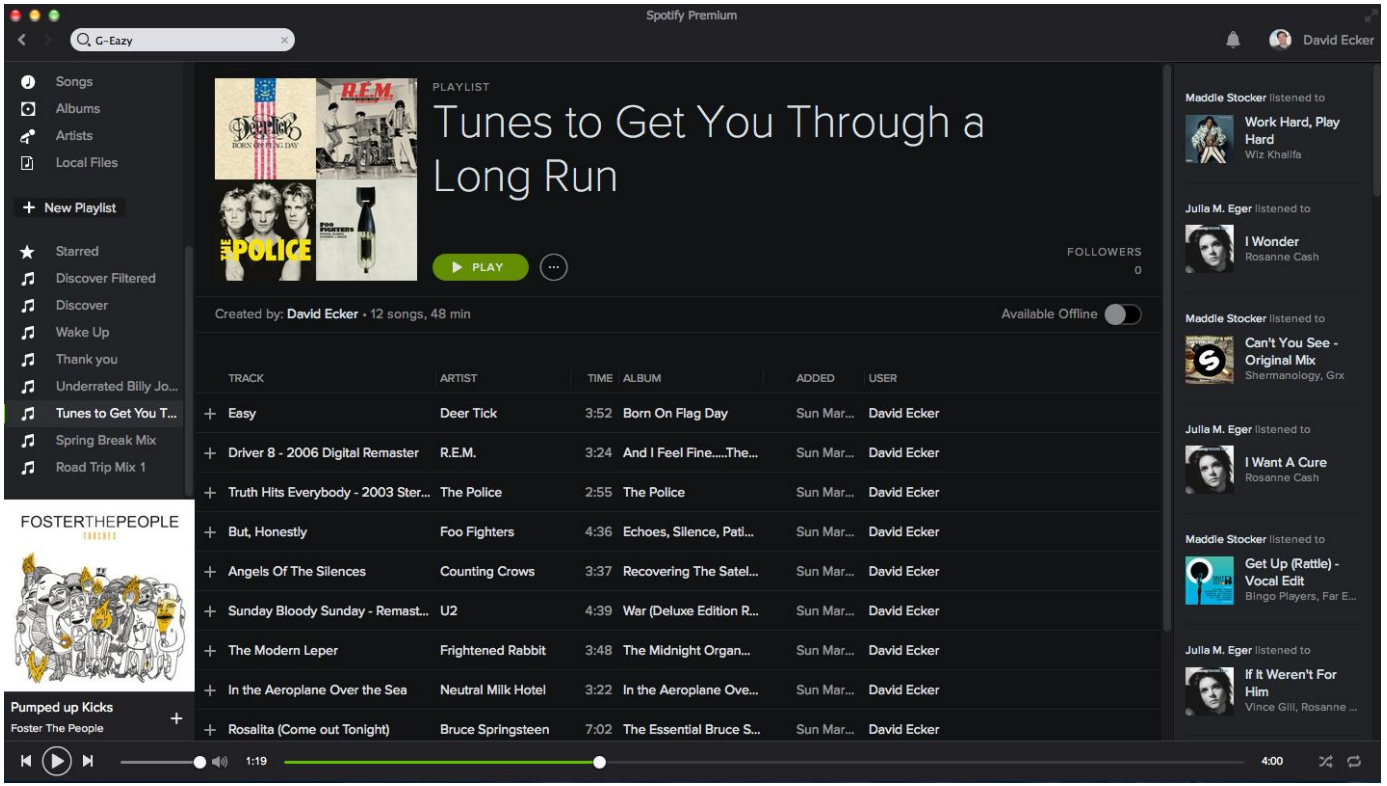
Agenda

- Create a graph [edge file]
- Number of vertices, number of edges, average in-degree
- Top 3 vertices with the most neighbors
- Jaccard Similarity Metric
- Page Rank

Music Profiles Example

Input Data

- <user-id, song-id, play-count>
- Set of bad records [IDs]

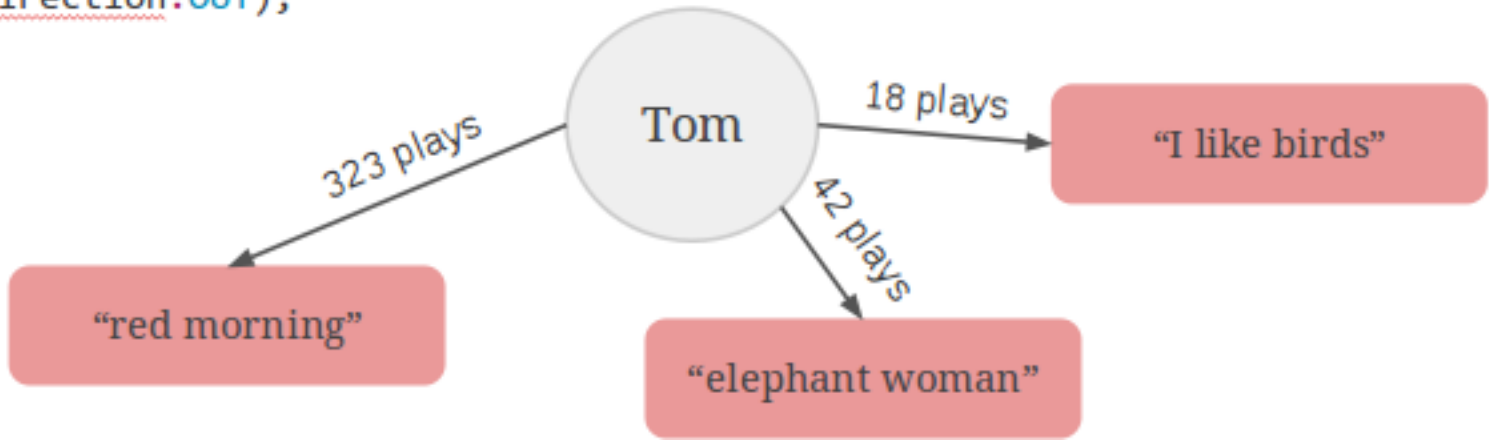


Filter out Bad Records

```
/** Read <userID>\t<songID>\t<playcount> triplets */
DataSet<Tuple3> triplets = getTriplets();
/** Read the bad records songIDs */
DataSet<Tuple1> mismatches = getMismatches();
/** Filter out the mismatches from the triplets dataset */
DataSet<Tuple3> validTriplets = triplets.coGroup(mismatches).where(1).equalTo(0)
    .with(new CoGroupFunction {
        void coGroup(Iterable triplets, Iterable invalidSongs, Collector out) {
            if (!invalidSongs.iterator().hasNext())
                for (Tuple3 triplet : triplets) // this is a valid triplet
                    out.collect(triplet);
        }
```

Compute Top Songs/User

```
/** Create a user -> song weighted bipartite graph where the edge weights
correspond to play counts */
Graph userSongGraph = Graph.fromTupleDataSet(validTriplets, env);

/** Get the top track (most listened) for each user */
DataSet<Tuple2> usersWithTopTrack = userSongGraph
                    .groupReduceOnEdges(new GetTopSongPerUser(),
EdgeDirection.OUT);
```



Compute Top Songs/User

```
class GetTopSongPerUser implements EdgesFunctionWithVertexValue {
    void iterateEdges(Vertex vertex, Iterable<Edge> edges) {
            int maxPlaycount = 0;
            String topSong = "";
            for (Edge edge : edges) {
                if (edge.getValue() > maxPlaycount) {
                    maxPlaycount = edge.getValue();
                    topSong = edge.getTarget();
                }
            }
            return new Tuple2(vertex.getId(), topSong);
        }
    }
```

Coming up Next

- Scala API
- More Library Methods
- Flink Streaming Integration
- Graph Partitioning Techniques
- Specialized Operators for Highly Skewed Graphs
- GSA Iteration [already merged]

flink.apache.org

@ApacheFlink

user@flink.apache.org

dev@flink.apache.org