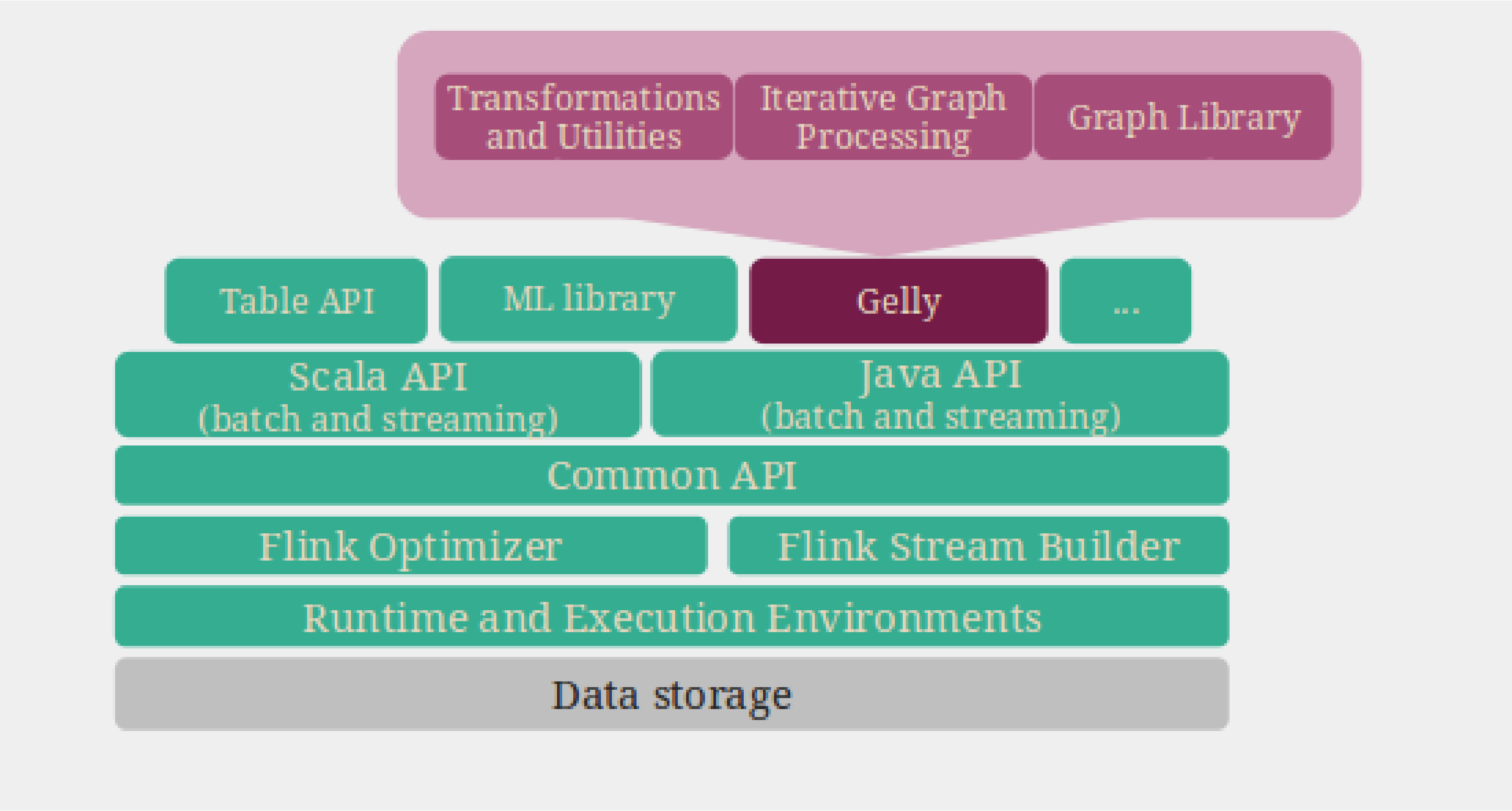
Large-Scale Graph Processing with Apache Flink

Andra Lungu
Flink committer
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



Transformations and Utilities

Iterative Graph Processing

Graph Library

Table API

ML library

Gelly

...

Scala API (batch and streaming)

Java API (batch and streaming)

Common API

Flink Optimizer

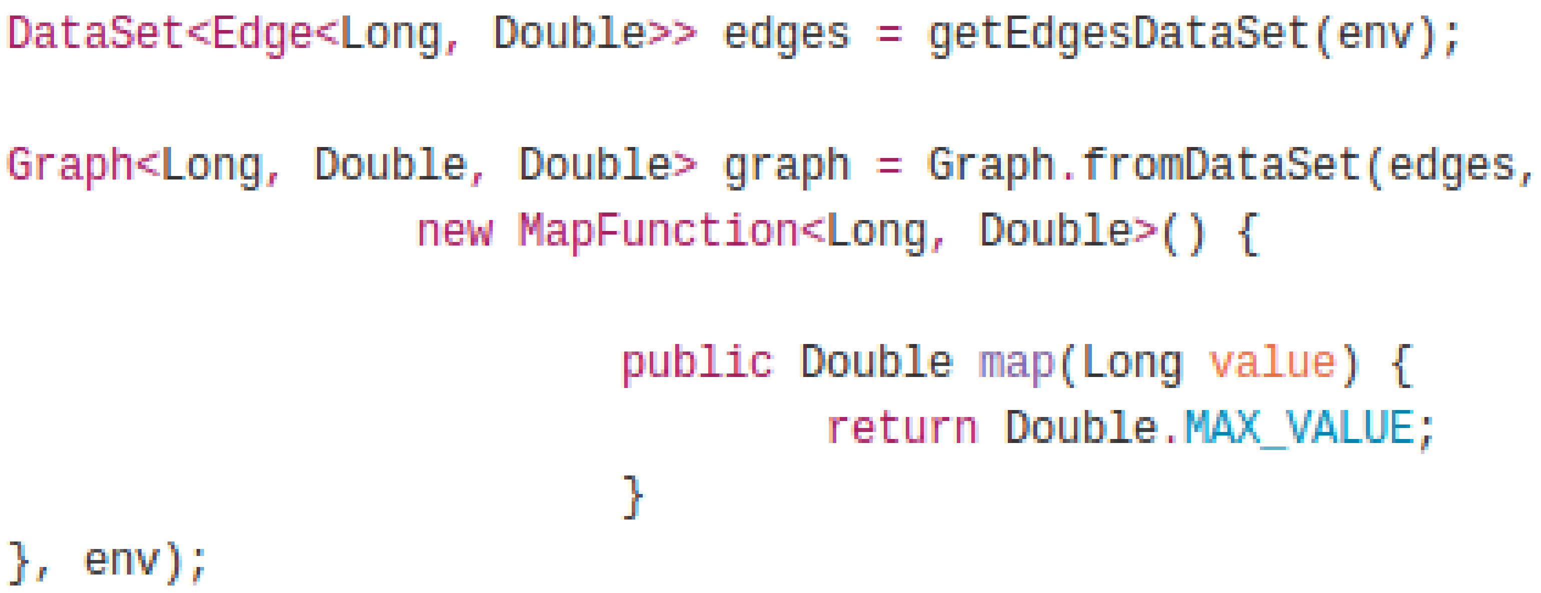
Flink Stream Builder

Runtime and Execution Environments

Data storage

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);
```

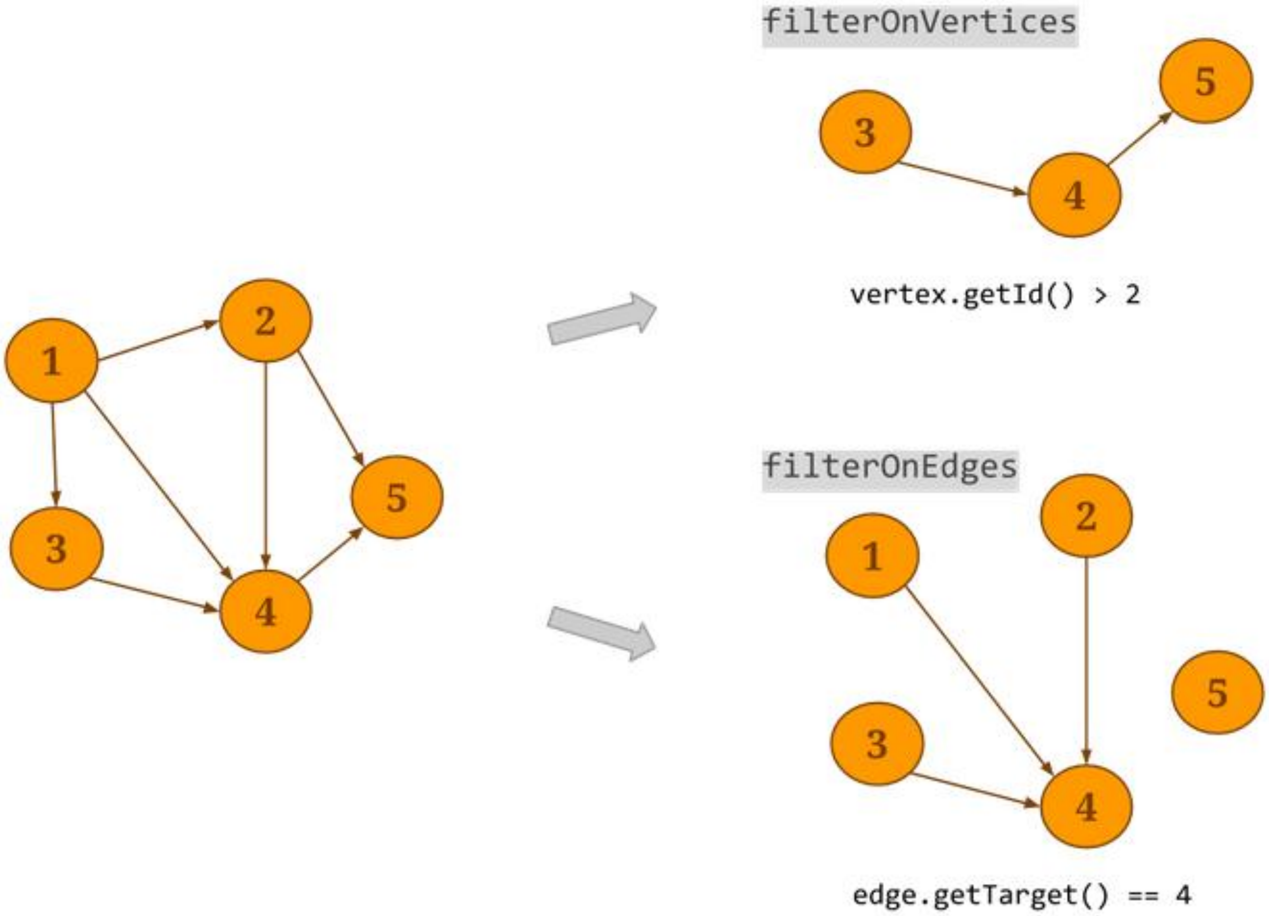
Graph Properties

- getVertices()
- getEdges()
- getVertexIds()
- getEdgeIds()
- inDegrees()
- outDegrees()
- getDegrees()
- numberOfVertices()
- numberOfEdges()
- 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



`filterOnVertices`

`vertex.getId() > 2`

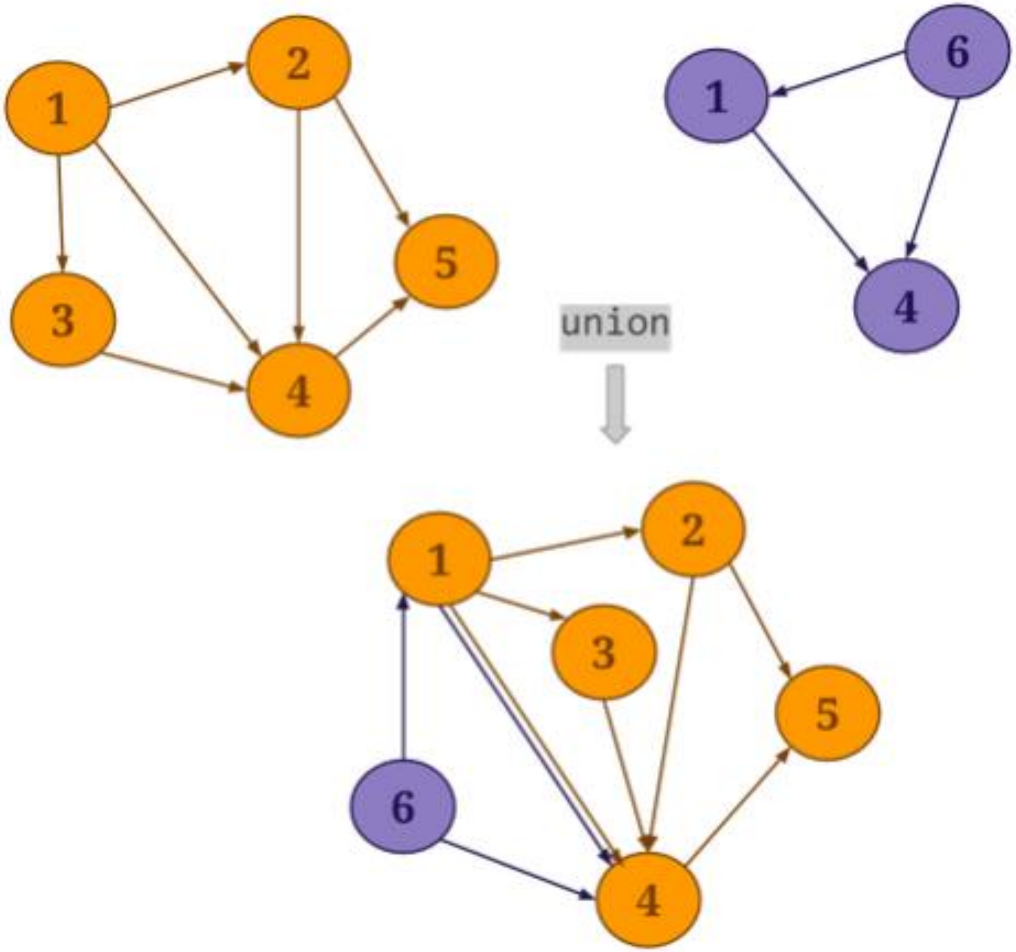
`filterOnEdges`

`edge.getTarget() == 4`

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

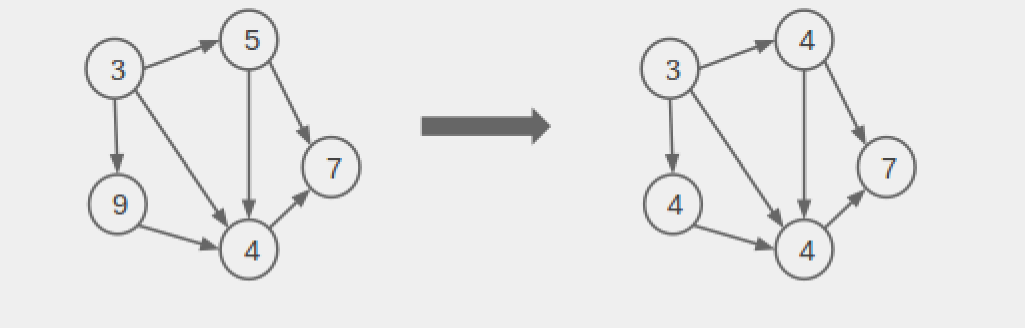


Graph Mutations

- **addVertex(final** Vertex<K, VV> vertex)
- **addVertices**(List<Vertex<K, VV>> verticesToAdd)
- **addEdge**(Vertex<K, VV> source, Vertex<K, VV> target, EV edgeValue)
- **addEdges**(List<Edge<K, EV>> newEdges)
- **removeVertex**(Vertex<K, VV> vertex)
- **removeVertices**(List<Vertex<K, VV>> verticesToBeRemoved)
- **removeEdge**(Edge<K, EV> edge)
- **removeEdges**(List<Edge<K, EV>> edgesToBeRemoved)

Neighborhood Methods

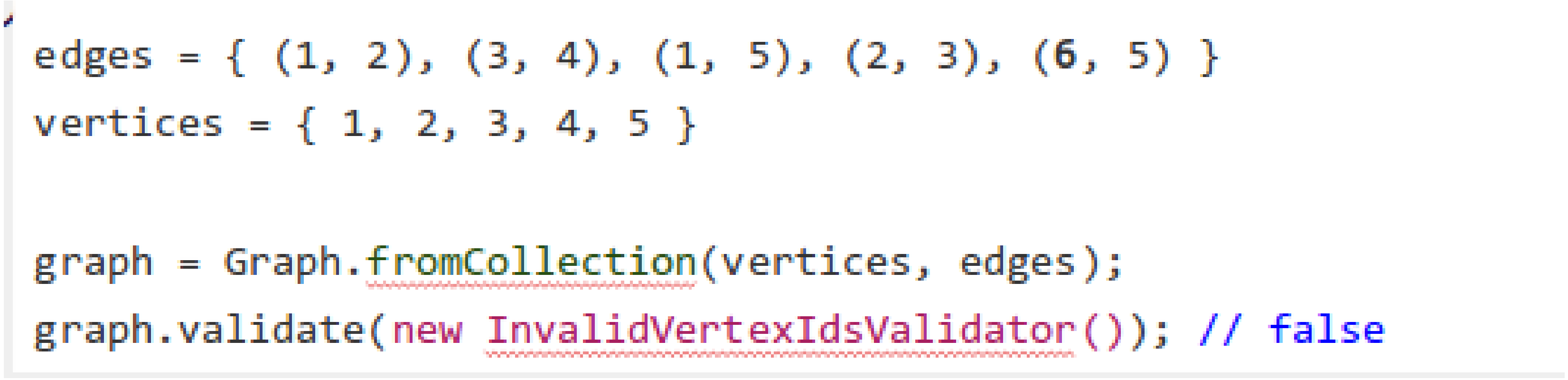
- **reduceOnNeighbors**(reduceNeighborsFunction, direction)



- reduceOnEdges
- groupReduceOnNeighbors; groupReduceOnEdges

Graph Validation

- Given criteria:
  - Edge IDs correspond to vertex IDs



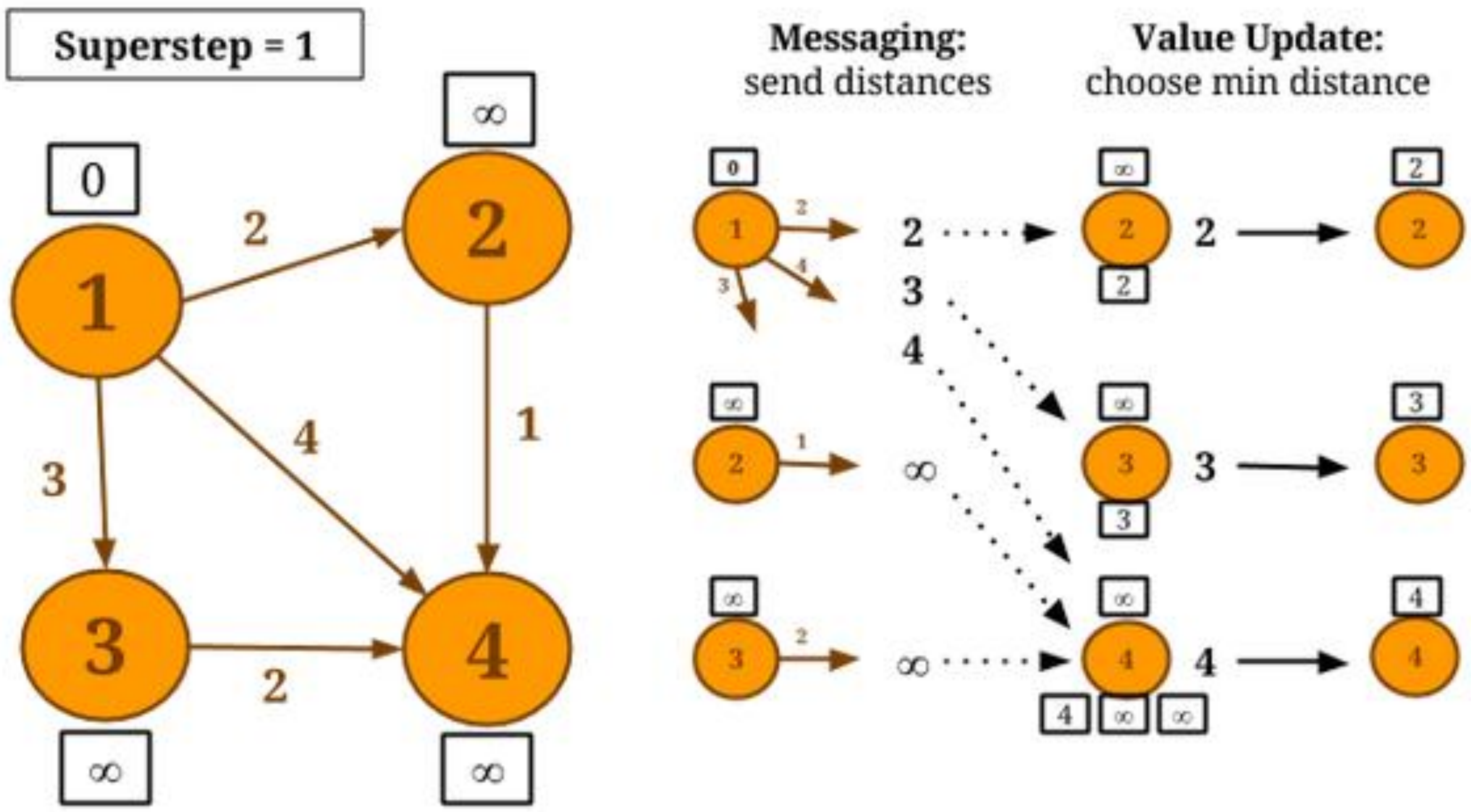
```
edges = { (1, 2), (3, 4), (1, 5), (2, 3), (6, 5) }
vertices = { 1, 2, 3, 4, 5 }

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

Vertex-centric Iterations

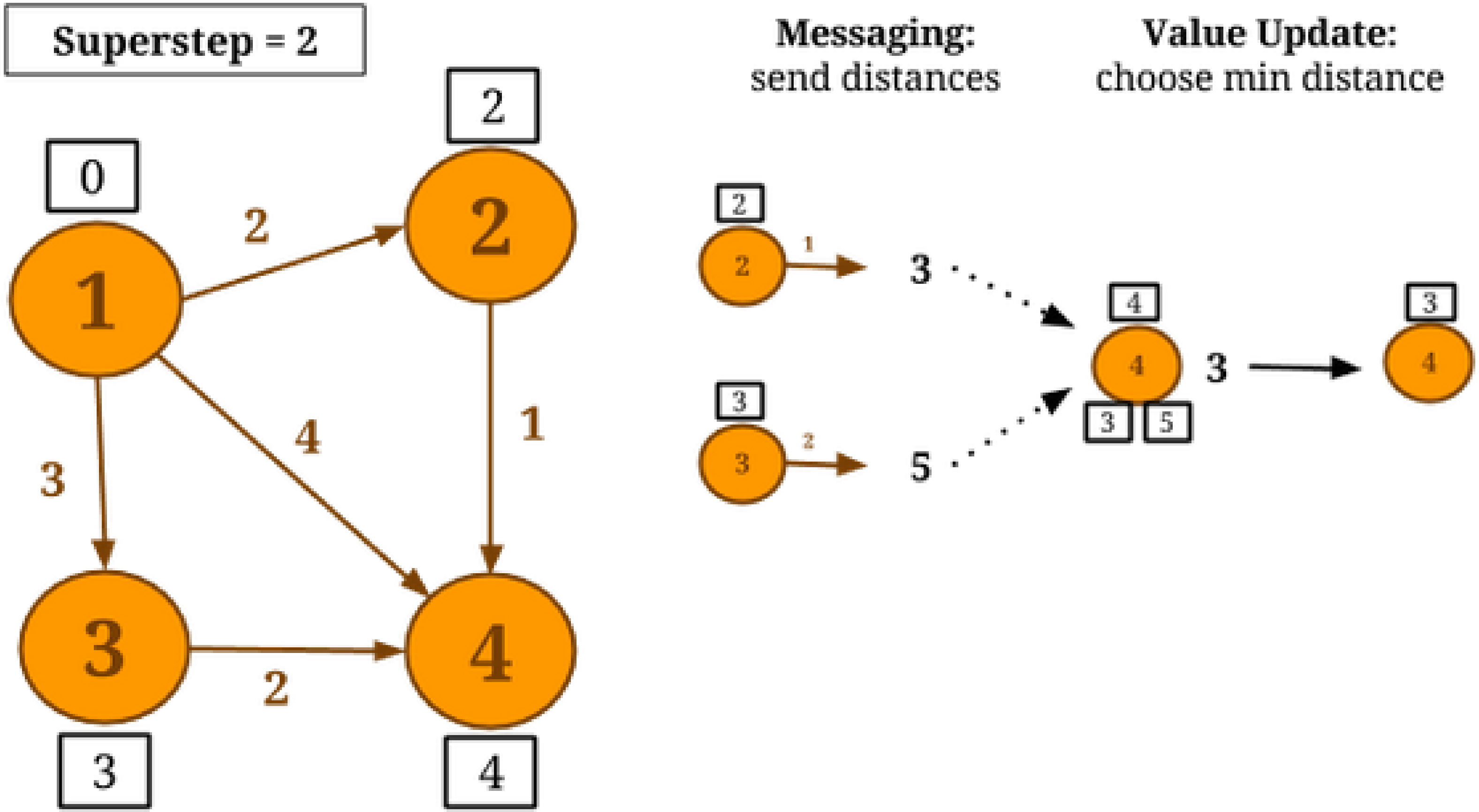
- Pregel [BSP] Execution Model
- UDFs:
  - Messaging Function
  - VertexUpdateFunction

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

Single Source Shortest Paths



SSSP – Second Superstep

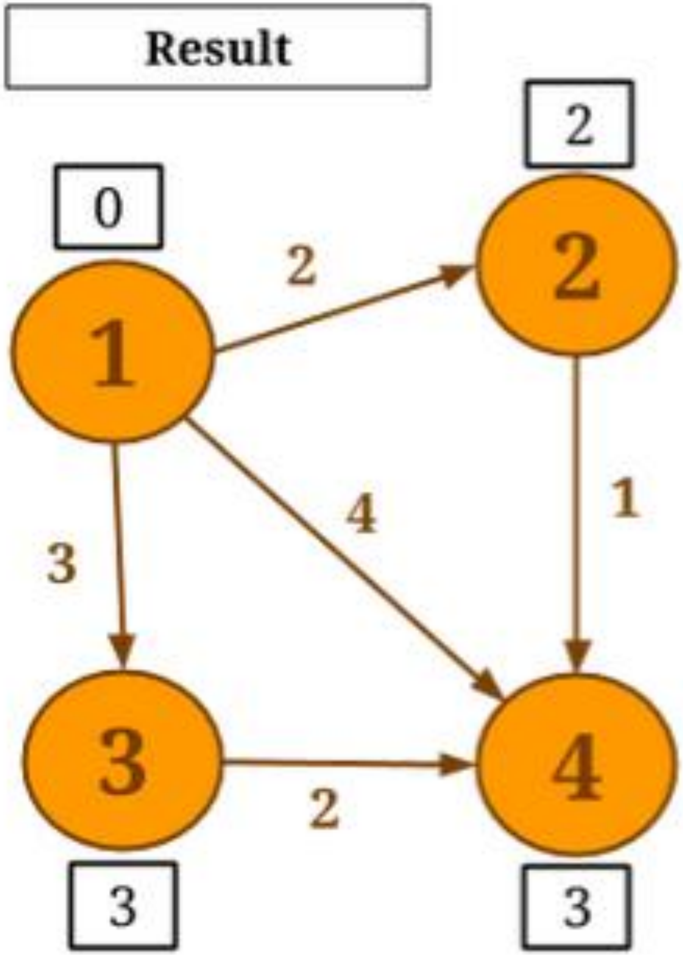


Superstep = 2

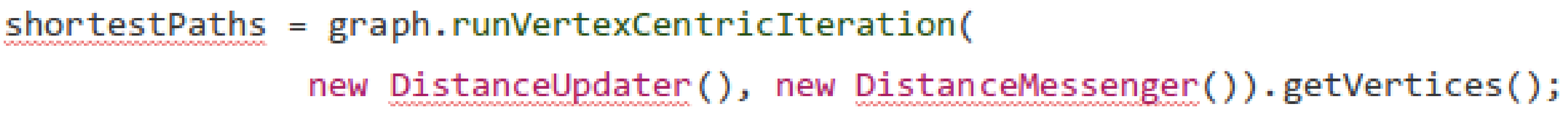
Messaging:
send distances

Value Update:
choose min distance

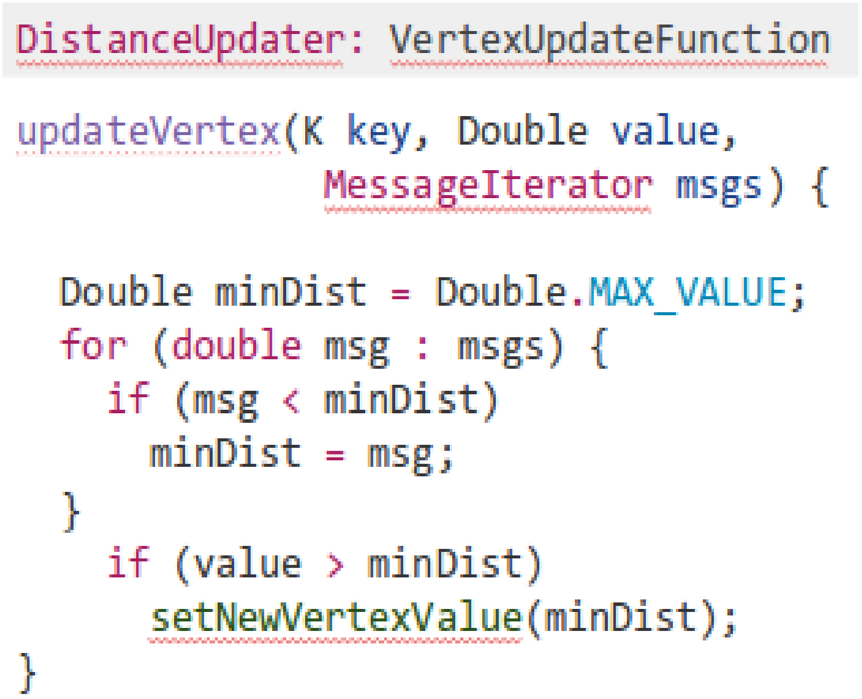
SSSP – Result



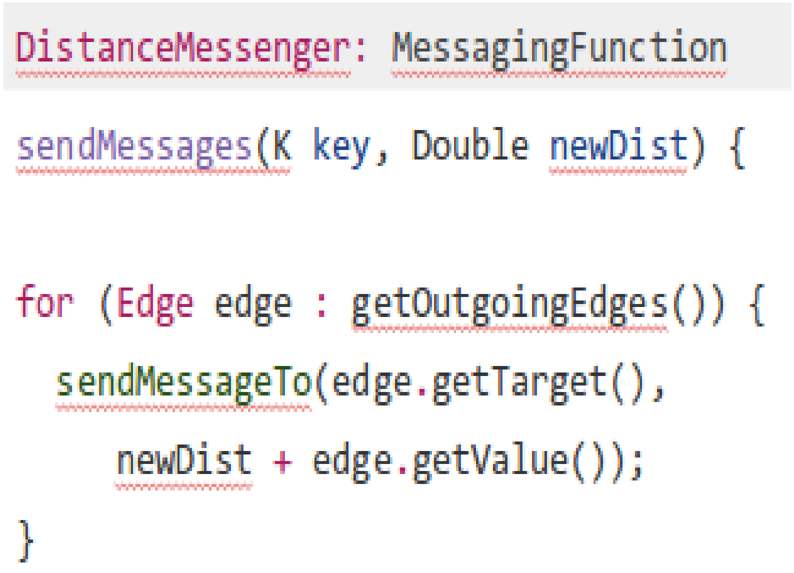
SSSP – code snippet



```
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);
}
```

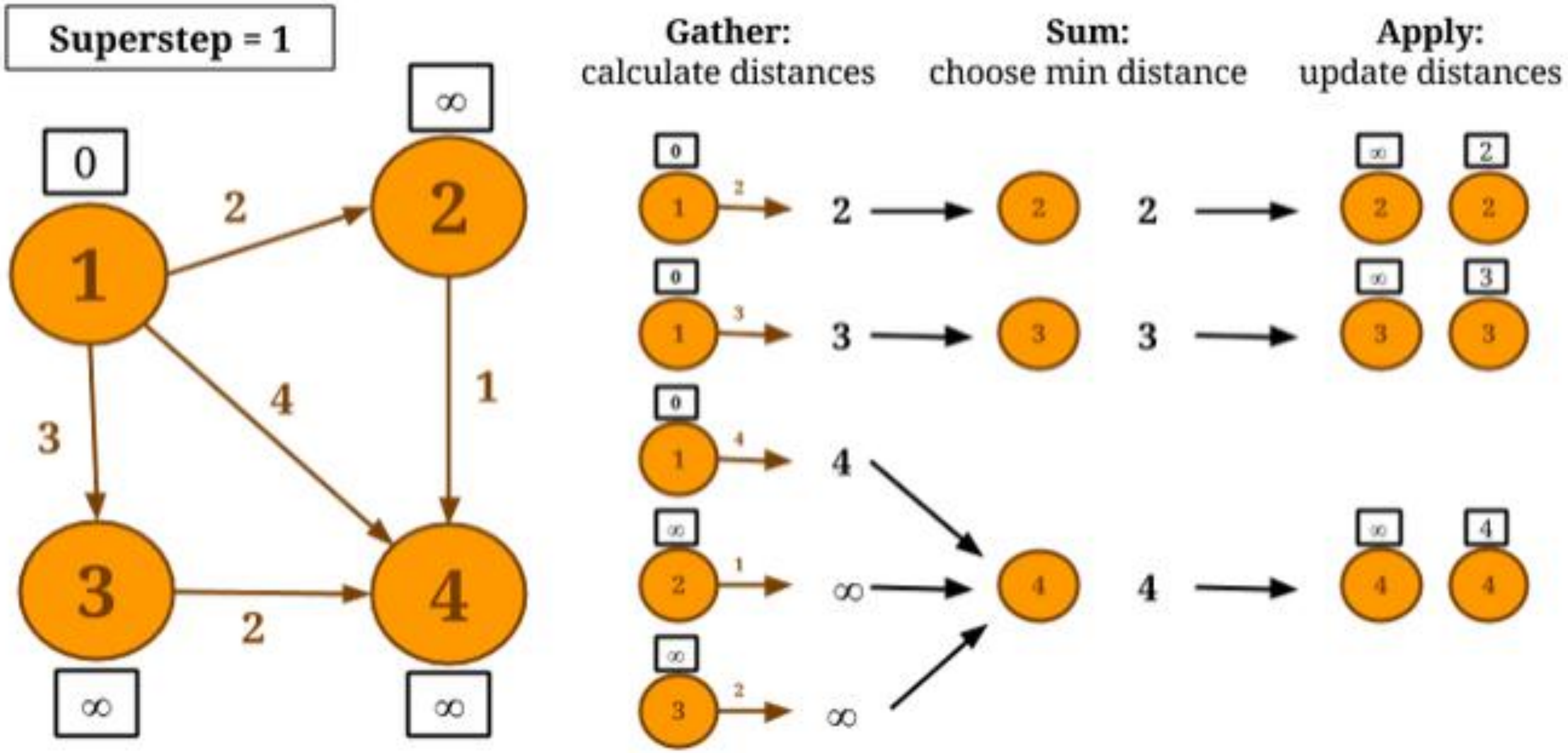


```
DistanceMessenger: MessagingFunction

sendMessages(K key, Double newDist) {


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

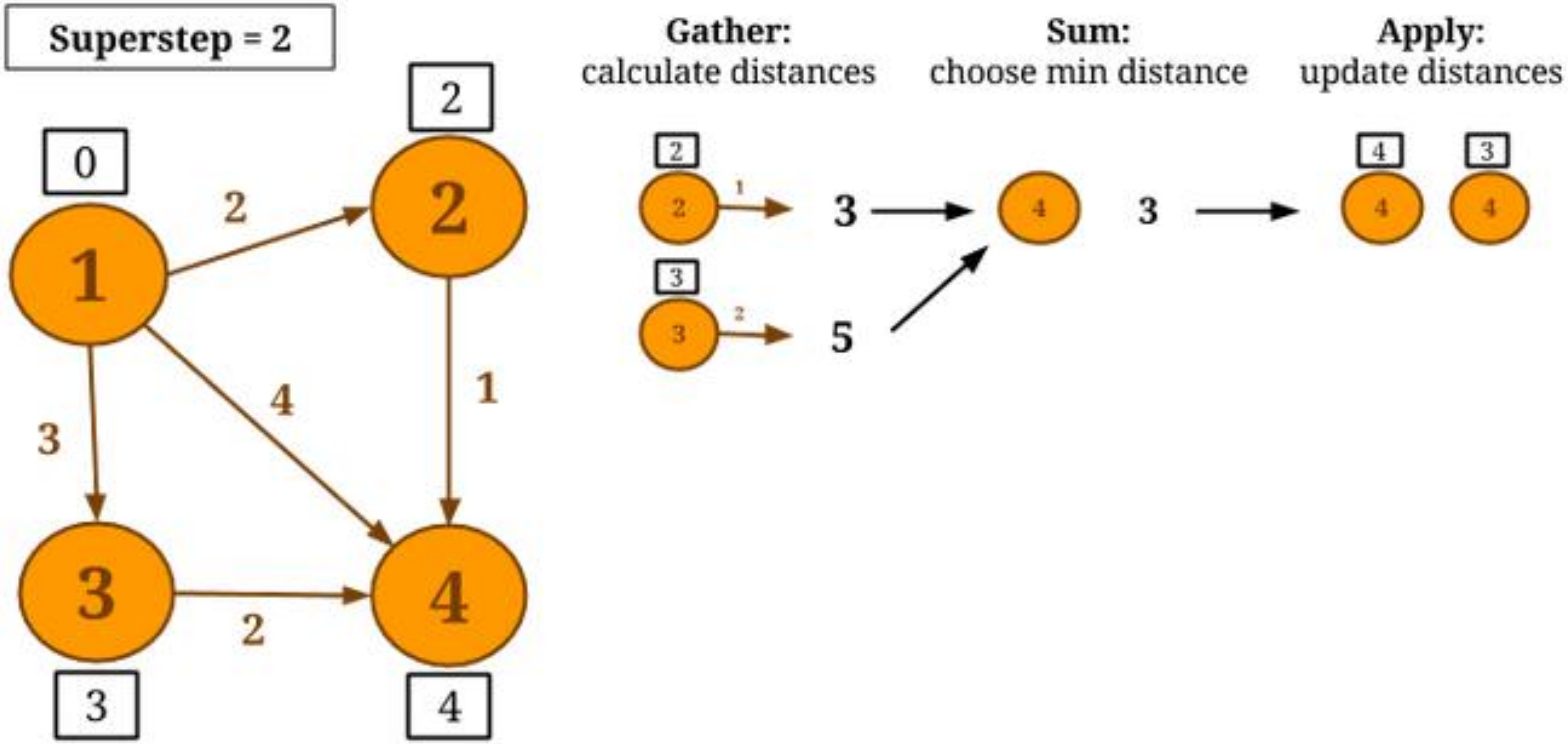
Gather-Sum-Apply Iterations

- UDFs:
  - GatherFunction
  - SumFunction
  - ApplyFunction
- Back to SSSP:
  - Gather: neighbor value + edge weight
  - Sum/Accumulate: choose min
  - Apply: compare computed min and update

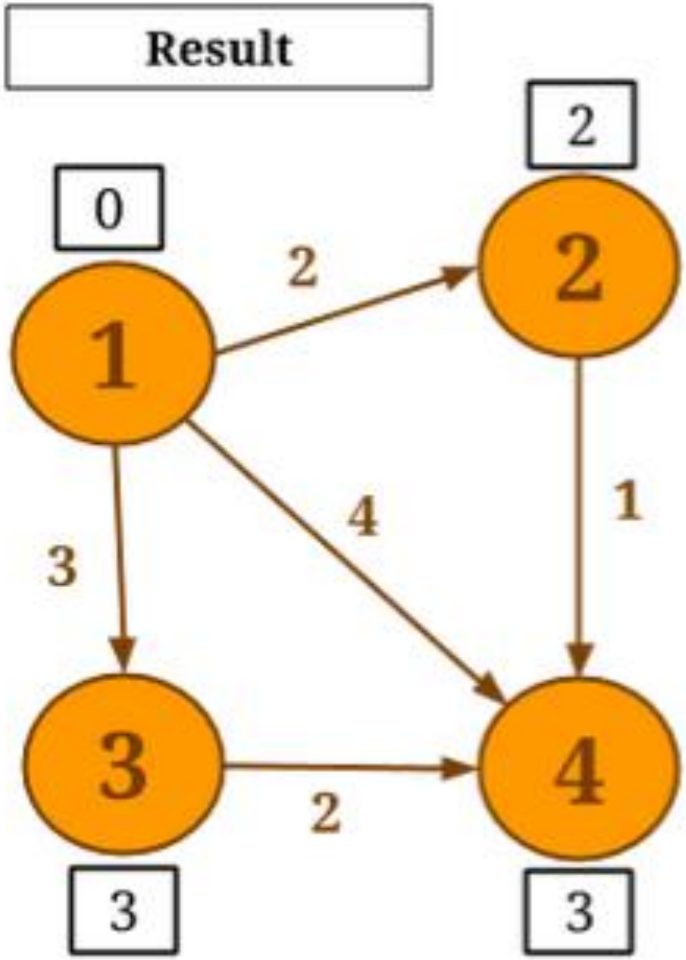
SSSP – Superstep 1



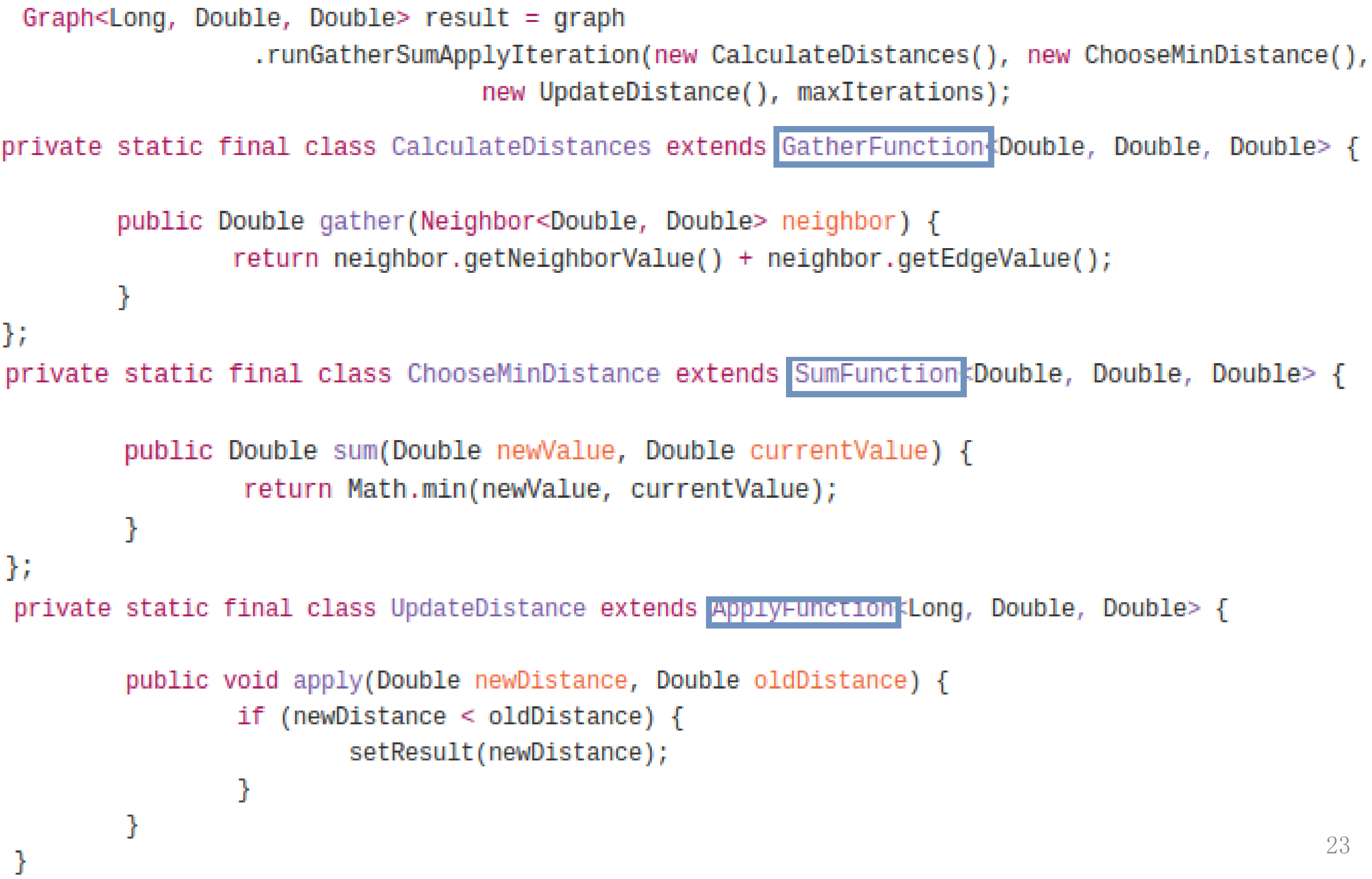
SSSP – Superstep 2



SSSP - Result



SSSP – code snippet



```
 Graph<Long, Double, Double> result = graph
                  .runGatherSumApplyIteration(new CalculateDistances(), new ChooseMinDistance(),
                                 new UpdateDistance(), maxIterations);
private static final class CalculateDistances extends GatherFunction<Double, Double, Double> {

        public Double gather(Neighbor<Double, Double> neighbor) {
                return neighbor.getNeighborValue() + neighbor.getEdgeValue();
        }
};
private static final class ChooseMinDistance extends SumFunction<Double, Double, Double> {

        public Double sum(Double newValue, Double currentValue) {
                return Math.min(newValue, currentValue);
        }
};
 private static final class UpdateDistance extends ApplyFunction<Long, Double, Double> {

        public void apply(Double newDistance, Double oldDistance) {
                if (newDistance < oldDistance) {
                        setResult(newDistance);
                }
        }
 }
```

Vertex-centric or GSA?

- Messaging = Gather + Sum
- Update = Apply

- Skewed graphs? – GSA (parallel gather)
- coGroup vs. reduce
- GSA gathers from immediate neighbors;
- Vertex-centric send to any vertex

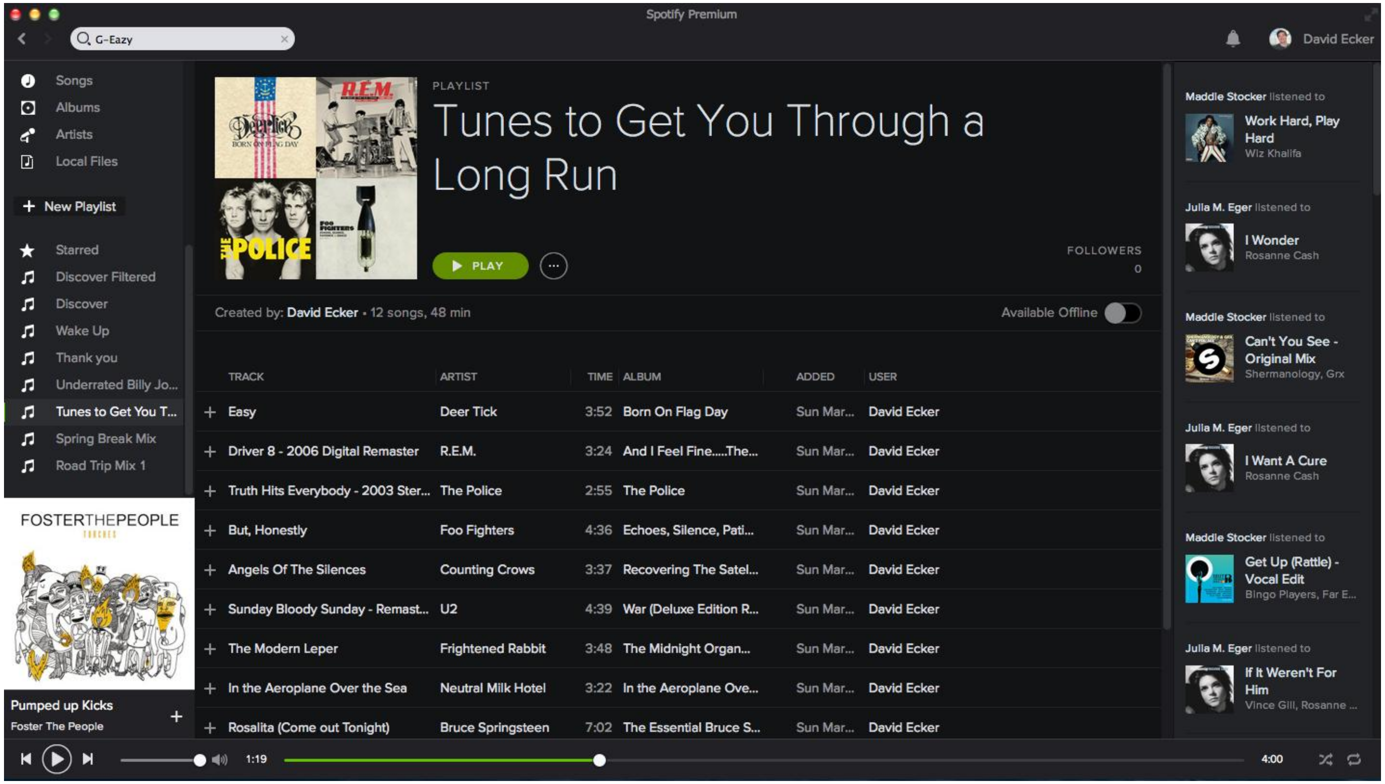
Library of Algorithms

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

Music Profiles Example

Input Data

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



PLAYLIST

Tunes to Get You Through a Long Run

Created by: David Ecker • 12 songs, 48 min

TRACK	ARTIST	TIME	ALBUM	ADDED	USER
Easy	Deer Tick	3:52	Born On Flag Day	Sun Mar...	David Ecker
Driver 8 - 2006 Digital Remaster	R.E.M.	3:24	And I Feel Fine....The...	Sun Mar...	David Ecker
Truth Hits Everybody - 2003 Ster...	The Police	2:55	The Police	Sun Mar...	David Ecker
But, Honestly	Foo Fighters	4:36	Echoes, Silence, Pati...	Sun Mar...	David Ecker
Angels Of The Silences	Counting Crows	3:37	Recovering The Satel...	Sun Mar...	David Ecker
Sunday Bloody Sunday - Remast...	U2	4:39	War (Deluxe Edition R...	Sun Mar...	David Ecker
The Modern Leper	Frightened Rabbit	3:48	The Midnight Organ...	Sun Mar...	David Ecker
In the Aeroplane Over the Sea	Neutral Milk Hotel	3:22	In the Aeroplane Ove...	Sun Mar...	David Ecker
Rosalita (Come out Tonight)	Bruce Springsteen	7:02	The Essential Bruce S...	Sun Mar...	David Ecker

Maddie Stocker listened to
Work Hard, Play Hard
Wiz Khalifa

Julia M. Eger listened to
I Wonder
Rosanne Cash

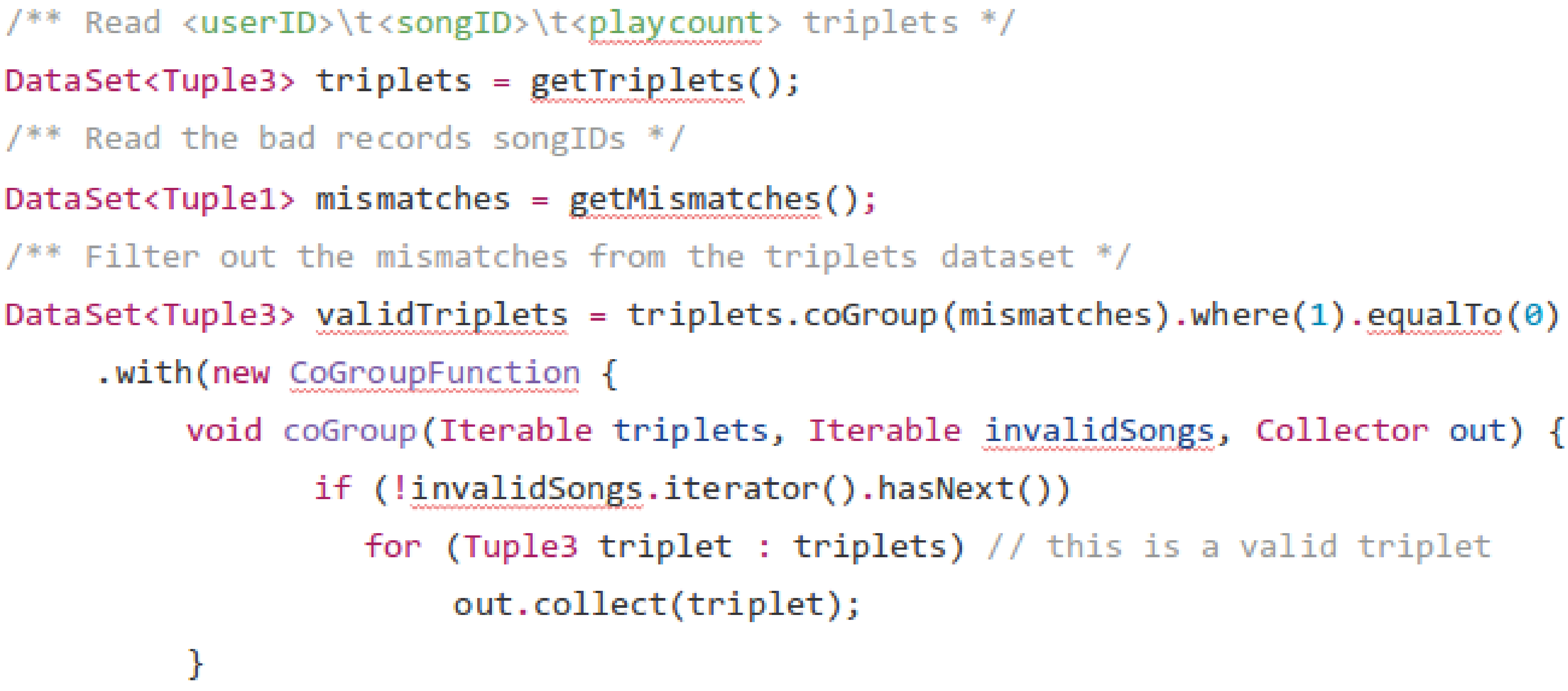
Maddie Stocker listened to
Can't You See - Original Mix
Shermanology, Grx

Julia M. Eger listened to
I Want A Cure
Rosanne Cash

Maddie Stocker listened to
Get Up (Rattle) - Vocal Edit
Bingo Players, Far E...

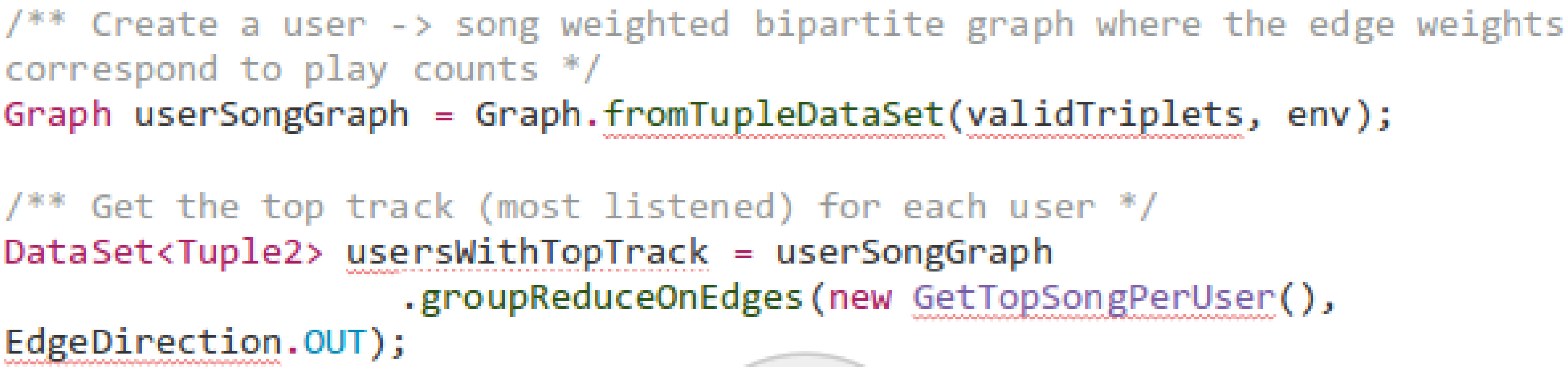
Julia M. Eger listened to
If It Weren't For Him
Vince Gill, Rosanne ...

Filter out Bad Records

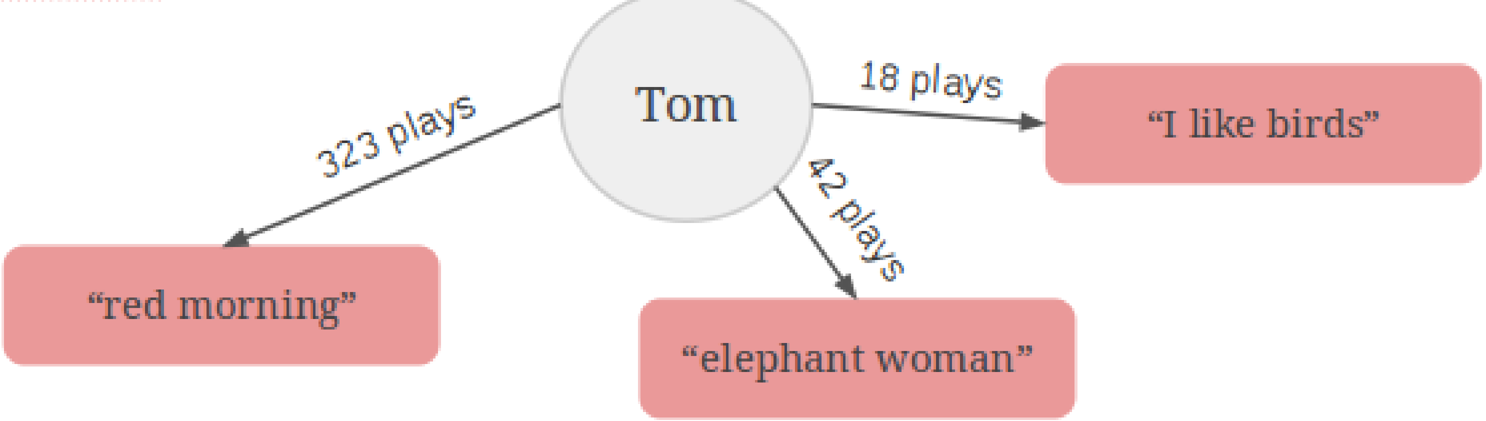


```
/** Read <userID>\t<songID>\t<playcount> triplets */
DataSet<Tuple3> triplets = getTriplets();
/** Read the bad records songIDs */
DataSet<Tuple1> mismatches = getMismatches();
/** Filter out the mismatches from the triplets dataset */
DataSet<Tuple3> validTriplets = triplets.coGroup(mismatches).where(1).equalTo(0)
     .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);
         }
```

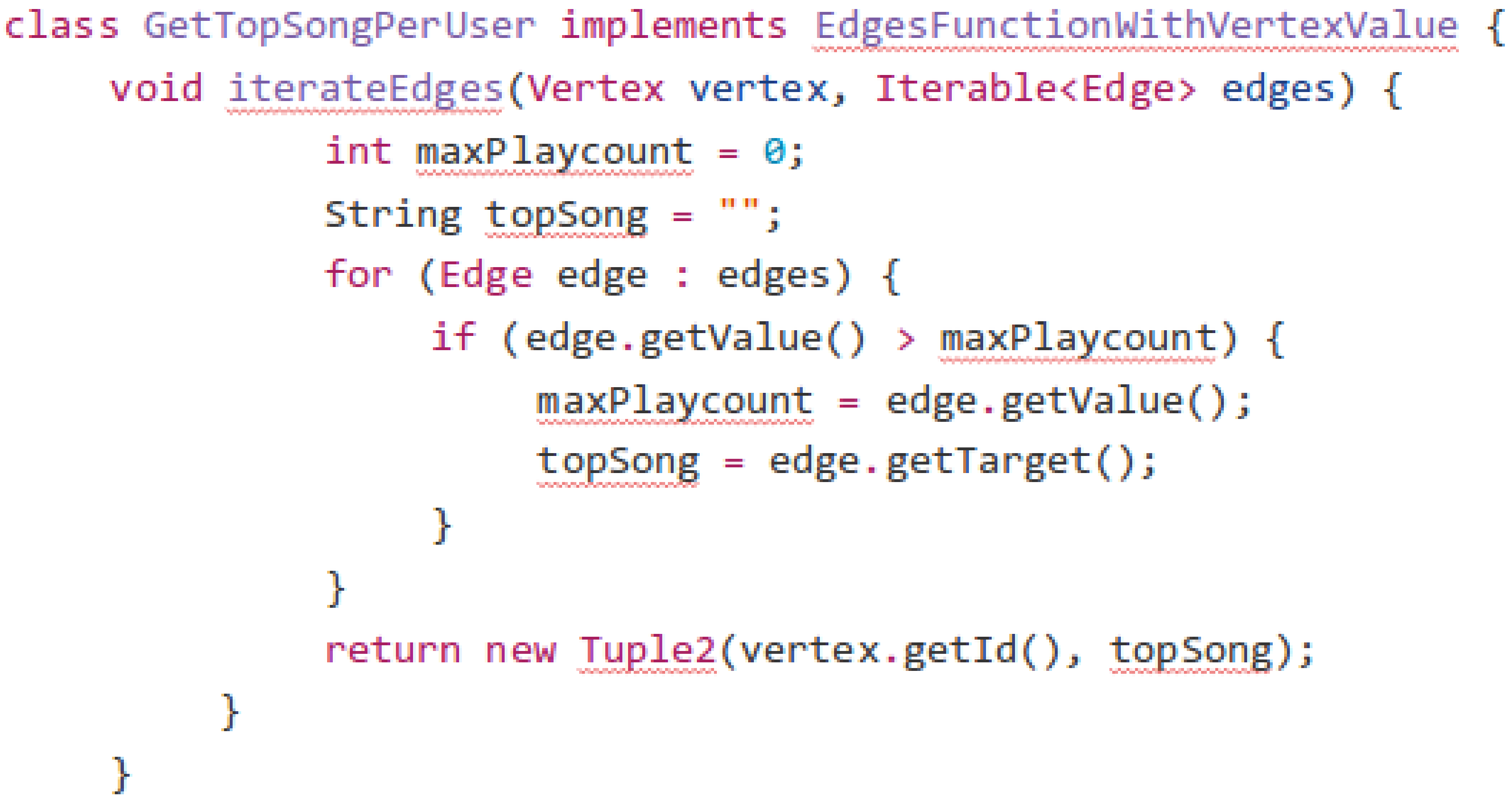
Compute Top Songs/User



```
/** 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);
```

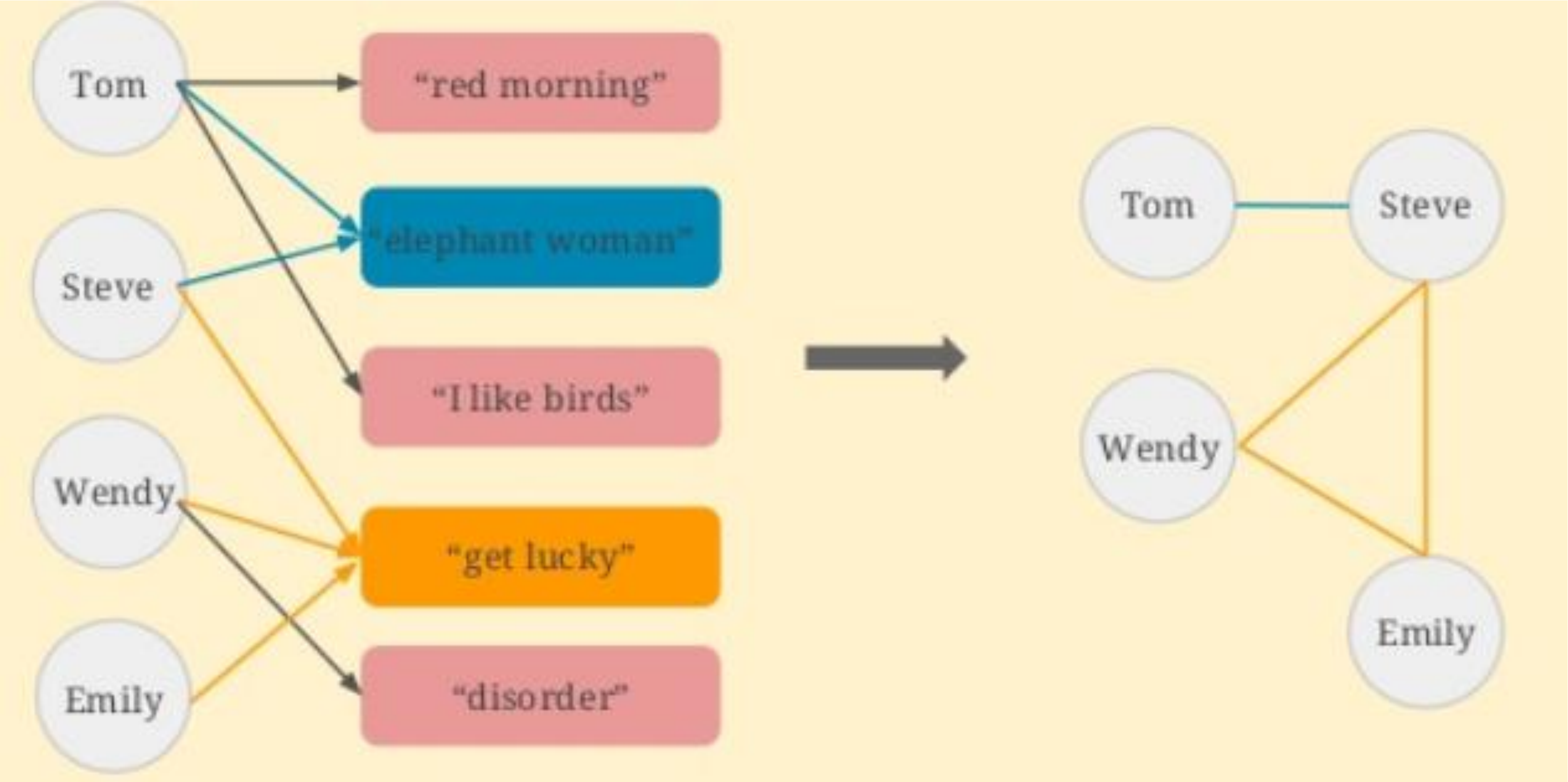


Compute Top Songs/User

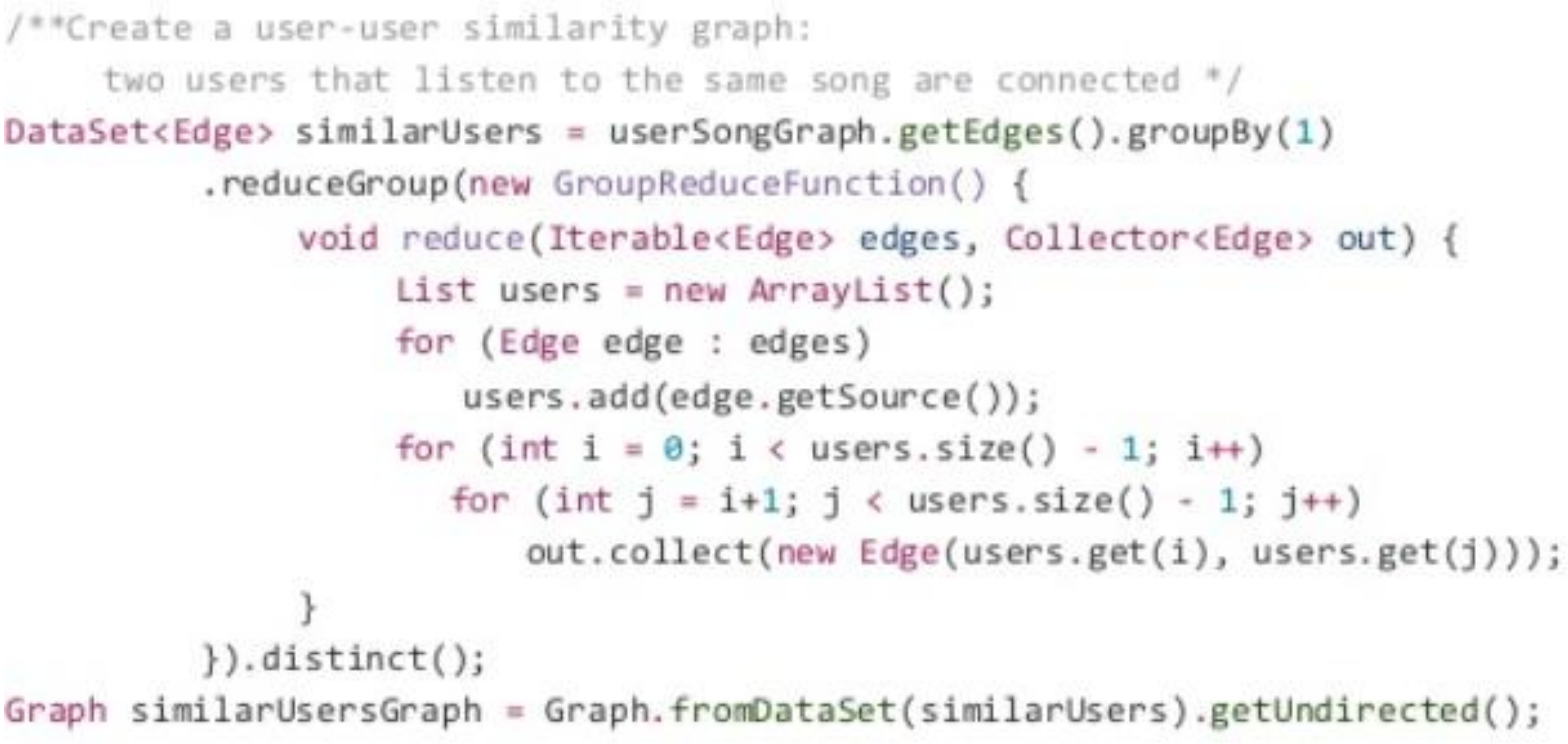


```
class GetTopSongPerUser implements EdgesFunctionWithVertexValue {
     void iterateEdges(Vertex vertex, Iterable<Edge> edges) {
              int maxPlaycount = 0;
              String topSong = "";
              for (Edge edge : edges) {
                   if (edge.getValue() > maxPlaycount) {
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

Create a user-user Graph

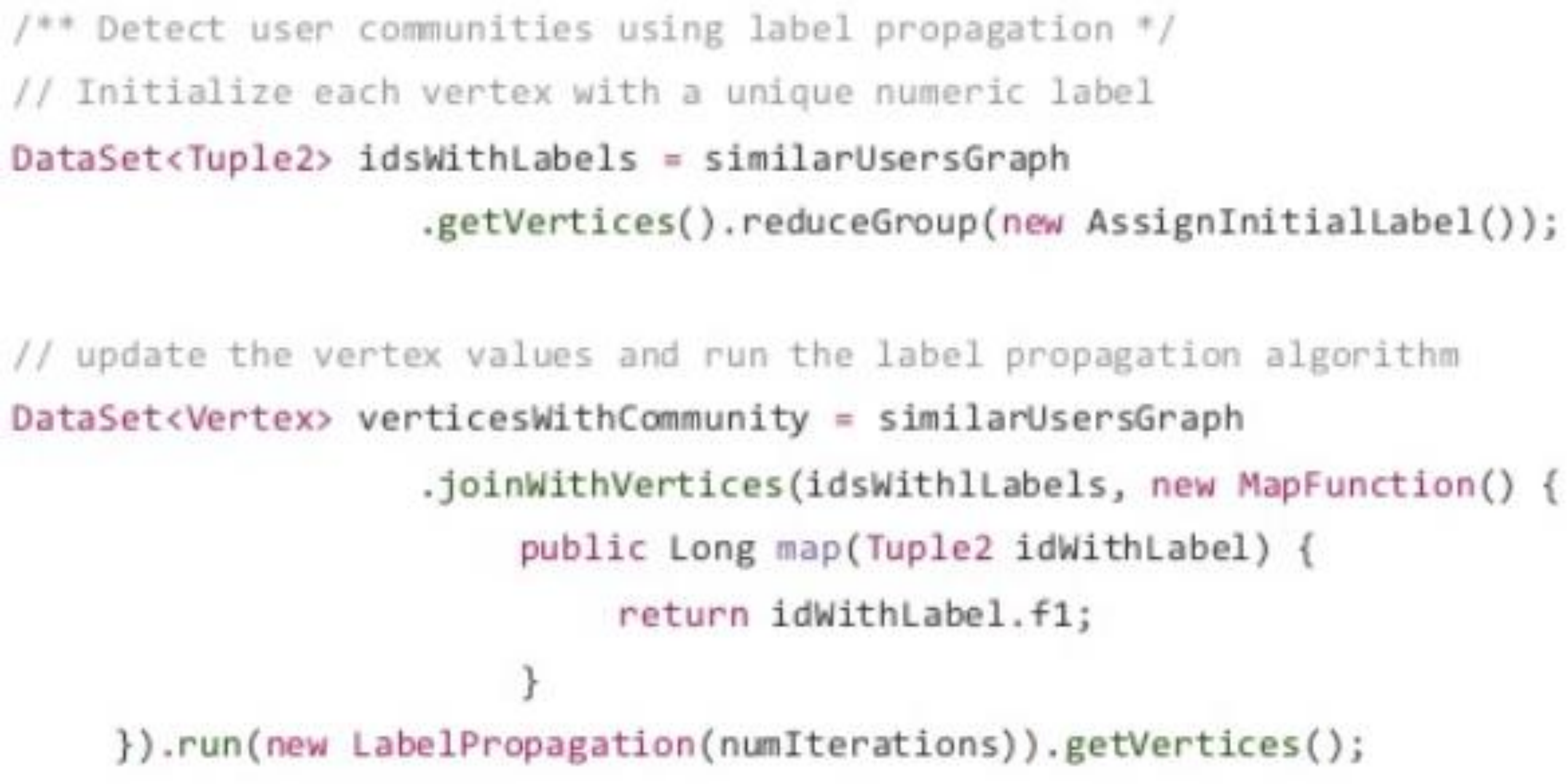


Create a user-user Graph



```
/**Create a user-user similarity graph:
    two users that listen to the same song are connected */
DataSet<Edge> similarUsers = userSongGraph.getEdges().groupBy(1)
        .reduceGroup(new GroupReduceFunction() {
            void reduce(Iterable<Edge> edges, Collector<Edge> out) {
                List users = new ArrayList();
                for (Edge edge : edges)
                    users.add(edge.getSource());
                for (int i = 0; i < users.size() - 1; i++)
                    for (int j = i+1; j < users.size() - 1; j++)
                        out.collect(new Edge(users.get(i), users.get(j)));
            }
        }).distinct();
Graph similarUsersGraph = Graph.fromDataSet(similarUsers).getUndirected();
```

Cluster Similar Users



```
/** Detect user communities using label propagation */
// Initialize each vertex with a unique numeric label
DataSet<Tuple2> idsWithLabels = similarUsersGraph
                   .getVertices().reduceGroup(new AssignInitialLabel());

// update the vertex values and run the label propagation algorithm
DataSet<Vertex> verticesWithCommunity = similarUsersGraph
                   .joinWithVertices(idsWithlLabels, new MapFunction() {
                       public Long map(Tuple2 idWithLabel) {
                            return idWithLabel.f1;
                       }
     }).run(new LabelPropagation(numIterations)).getVertices();
```

Coming up Next

- Gelly Blog Post
- Scala API
- More Library Methods
- Flink Streaming Integration
- Graph Partitioning Techniques
- Specialized Operators for Highly Skewed Graphs
- Bipartite Graph Support

Curious? Gelly Roadmap

flink.apache.org

@ApacheFlink

user@flink.apache.org

dev@flink.apache.org