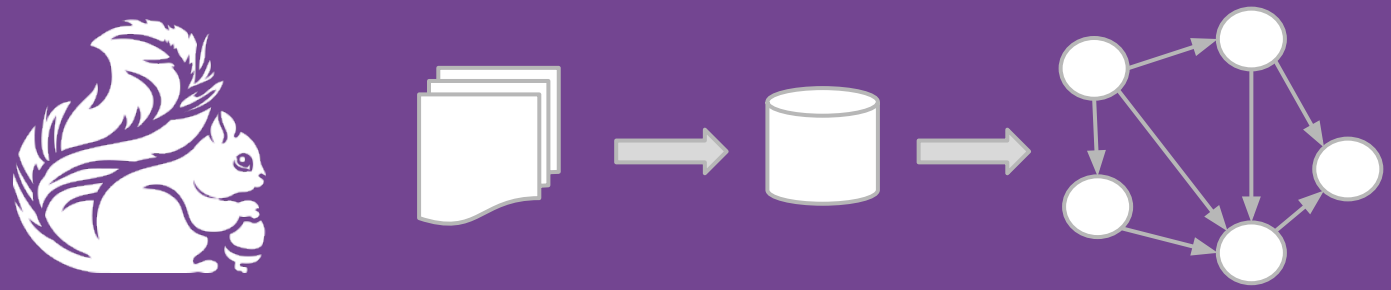
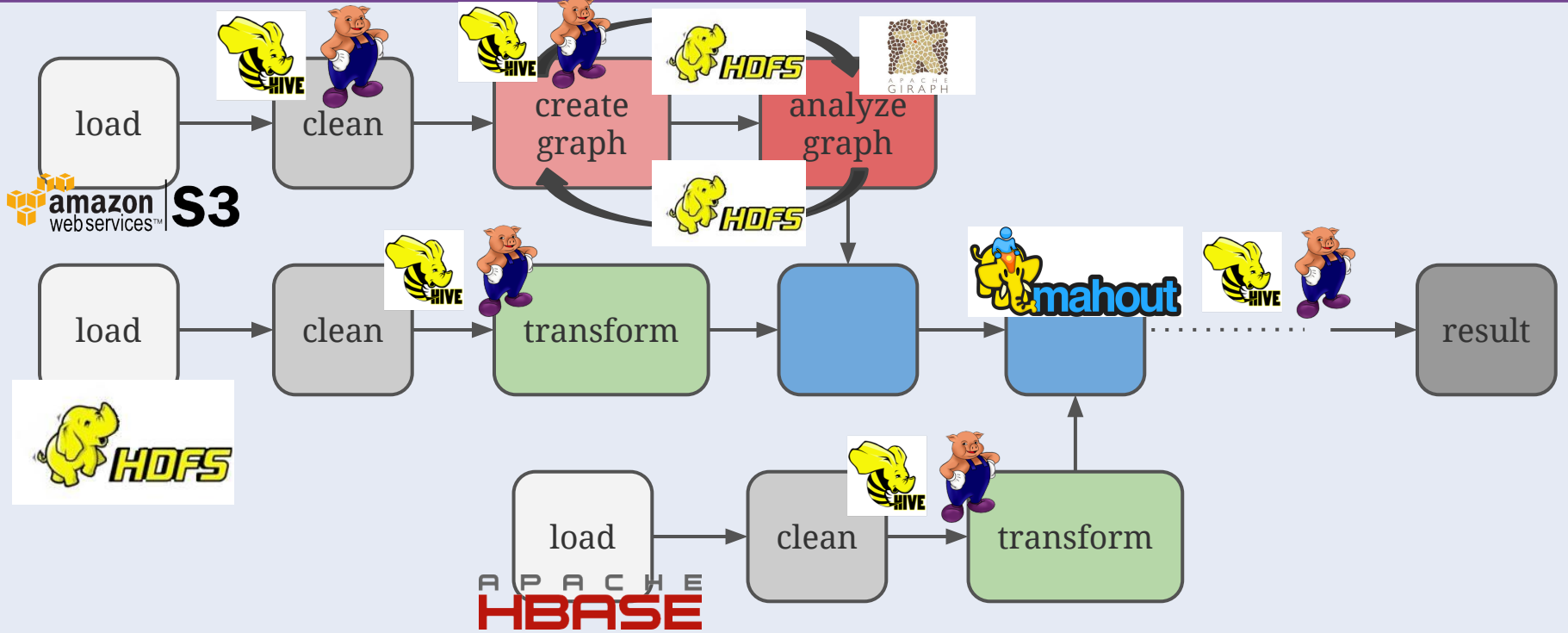
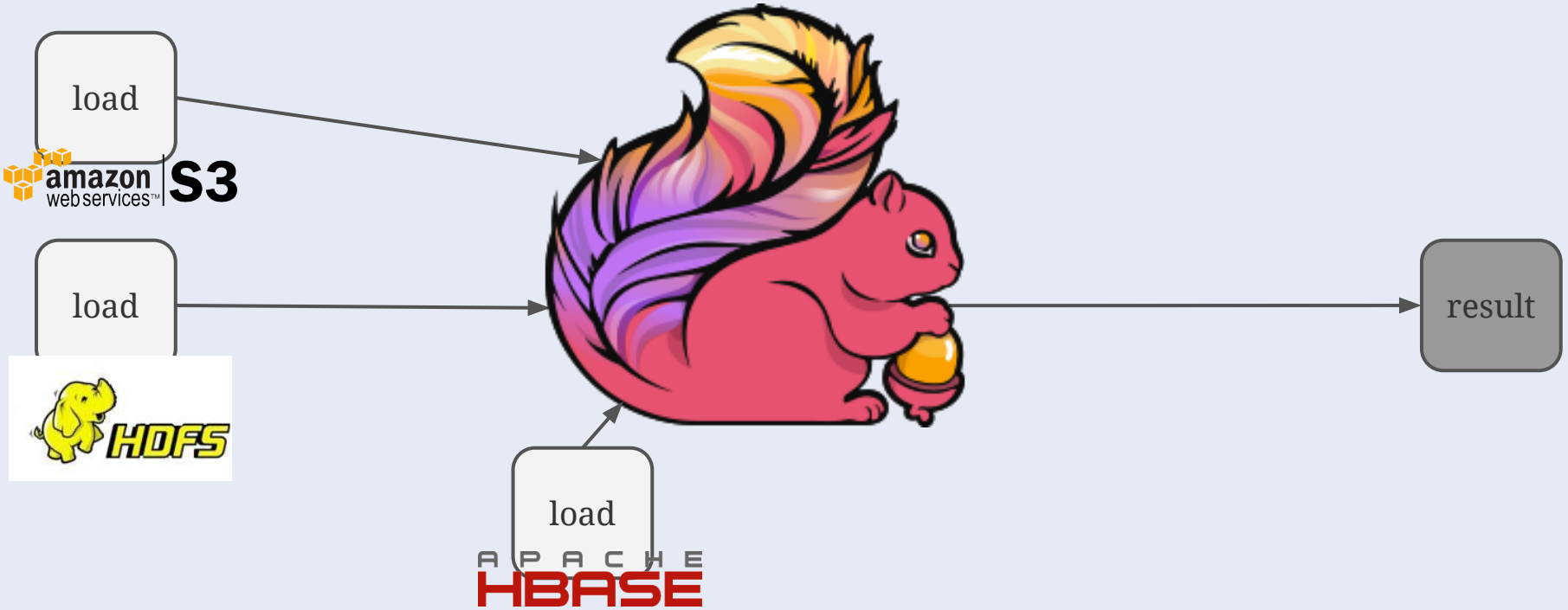
Why Graph Processing with Flink?



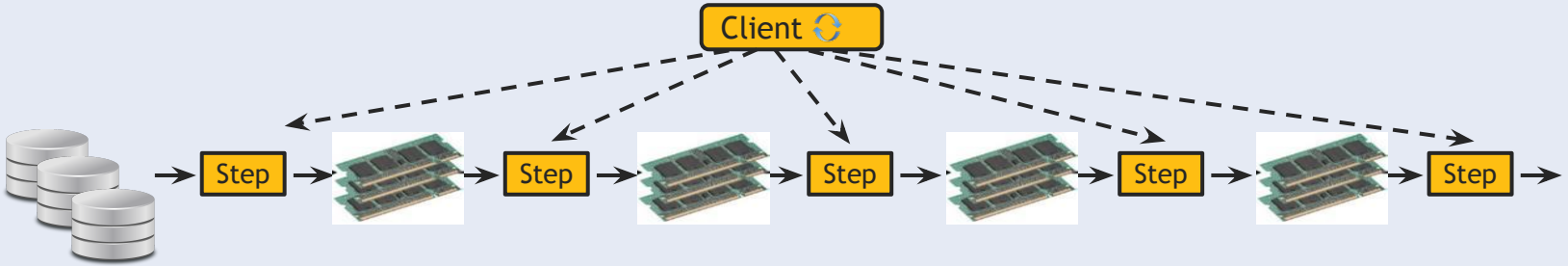
A data analysis pipeline



A more *user-friendly* pipeline



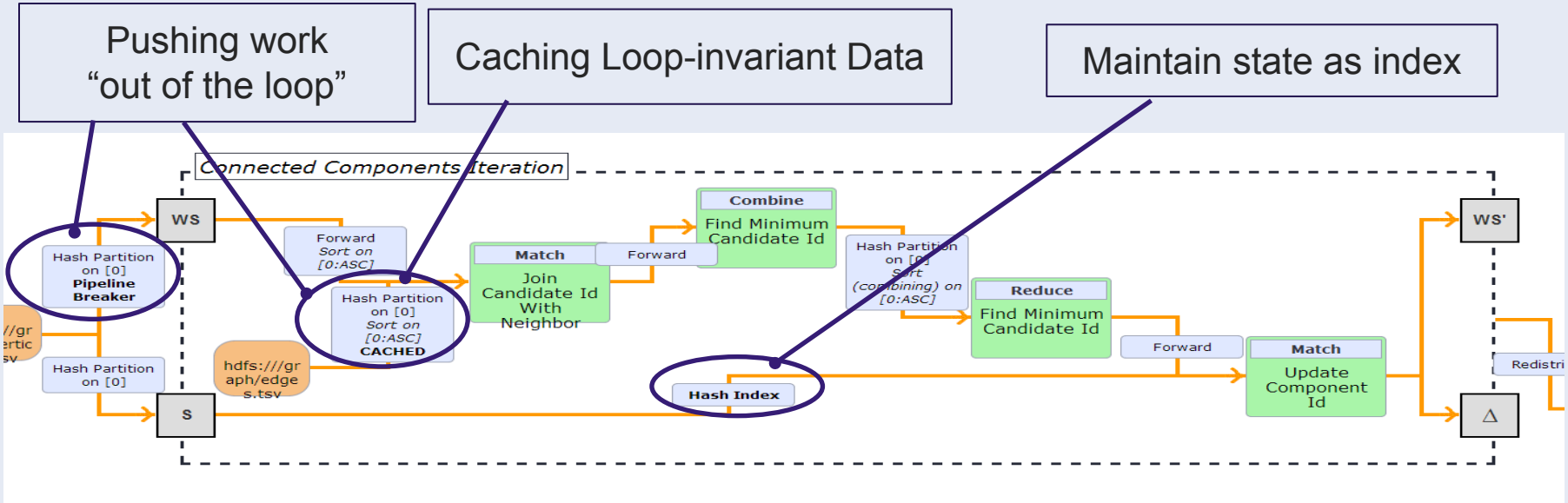
Iterate by unrolling



- for/while loop in client submits one job per iteration step
- Data reuse by caching in memory and/or disk

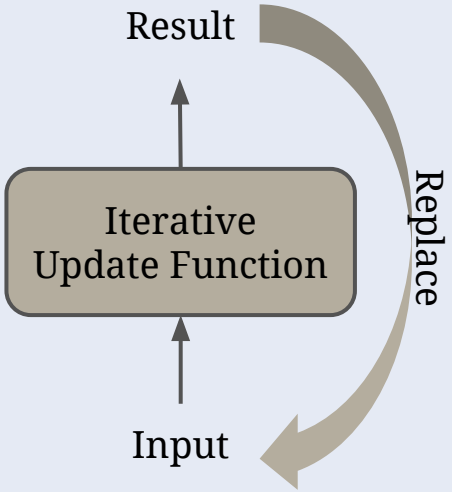
Native Iterations

- the runtime is aware of the iterative execution
- no scheduling overhead between iterations
- caching and state maintenance are handled automatically

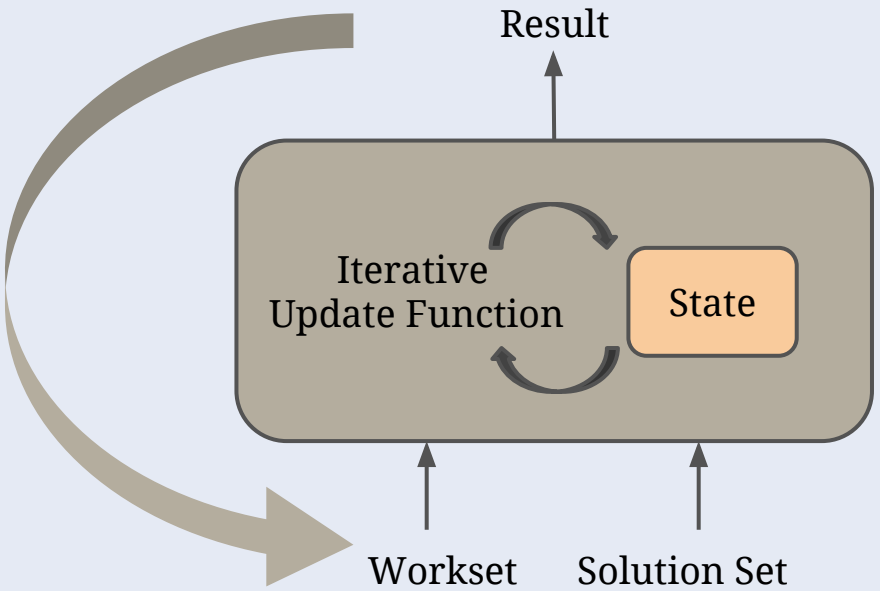


Flink Iteration Operators

Iterate

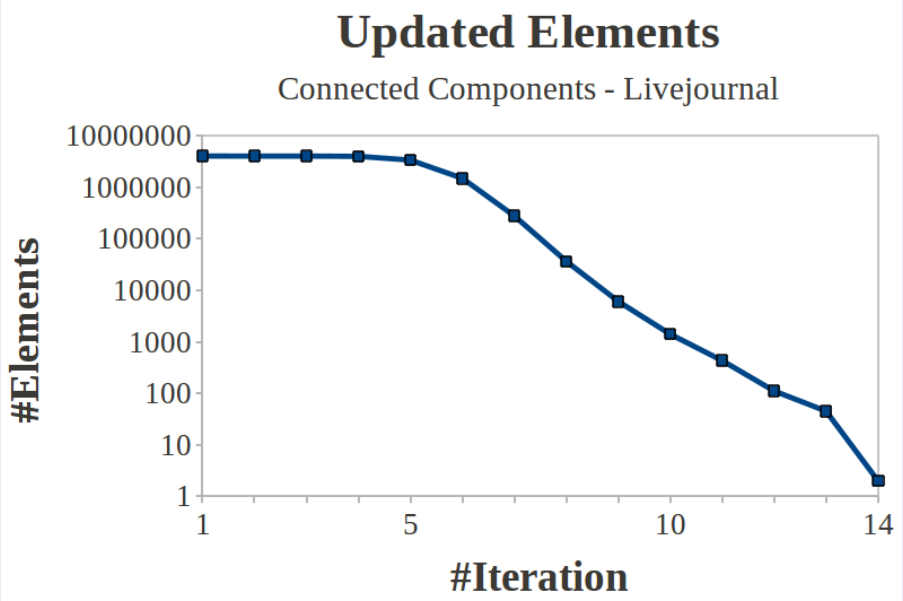


IterateDelta



Delta Iterations

Many iterative algorithms exhibit *sparse computational dependencies* and *asymmetric convergence*



Gelly

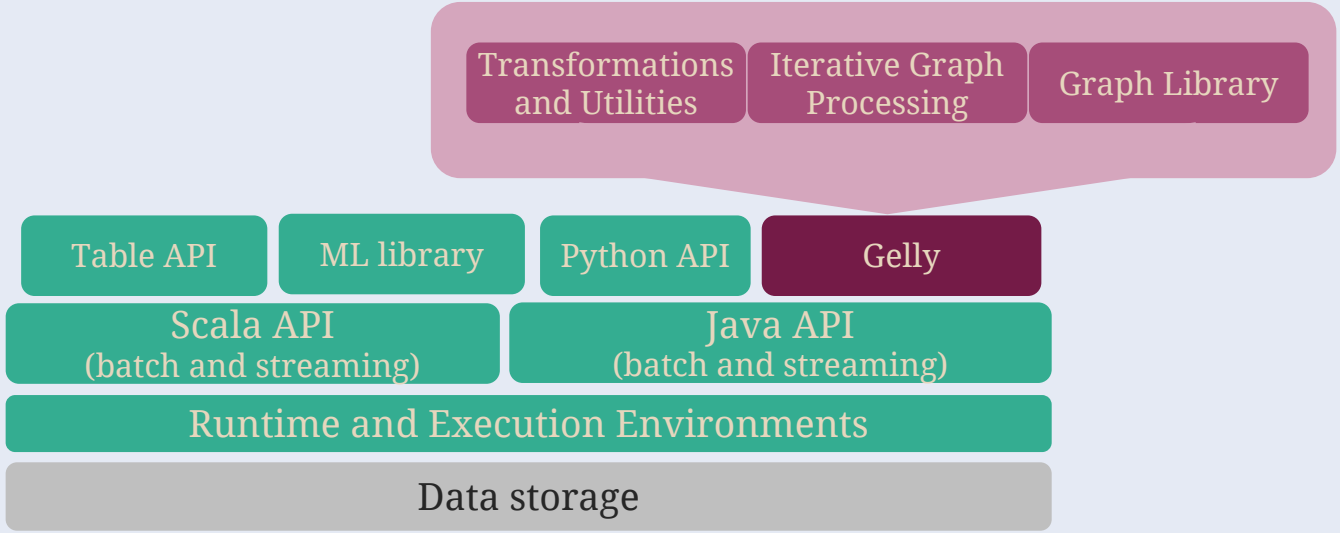
the Flink Graph API

Meet Gelly

- Java & Scala Graph APIs on top of Flink
- Ships already with Flink 0.9 (Beta)
- Can be seamlessly mixed with the DataSet Flink API

  → easily implement applications that use both *record-based* and *graph-based* analysis

Gelly in the Flink stack



Hello, Gelly!

```
// create a graph from a Dataset of Vertices and
a DataSet of Edges
Graph<String, Long, Double> graph =
        Graph.fromDataSet(vertices, edges, env);
```

Hello, Gelly!

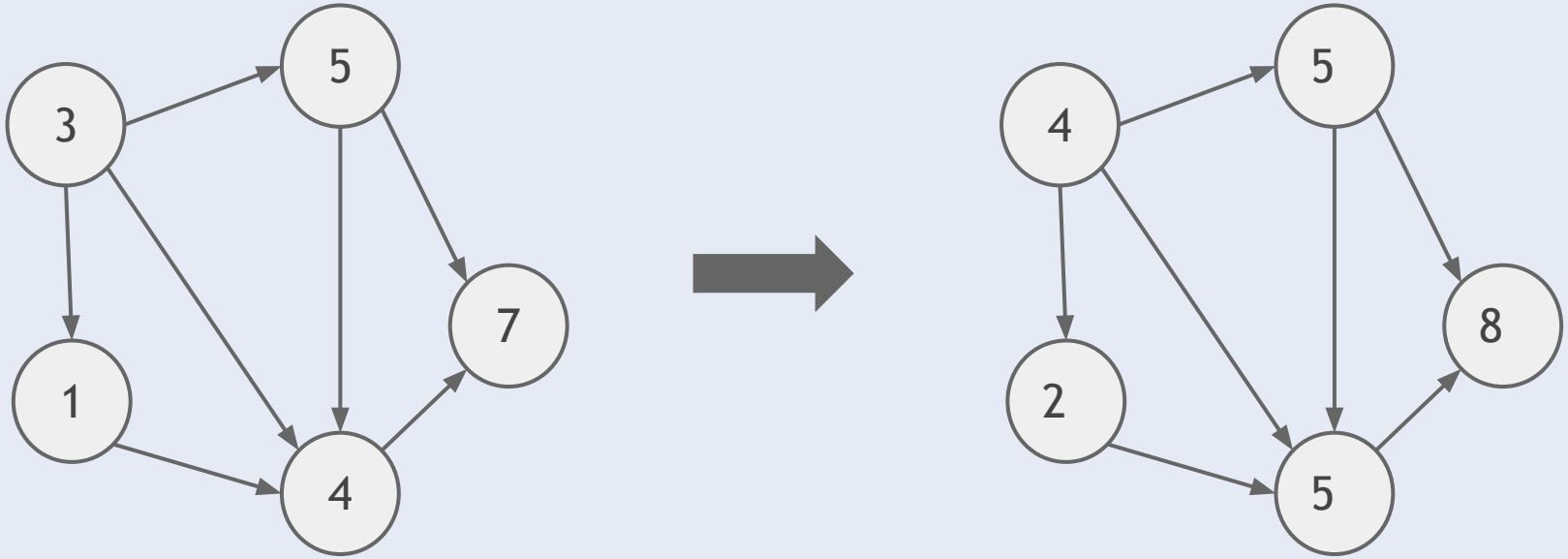
```
// create a graph from a Collection of Edges and
initialize the Vertex values
Graph<String, Long, Double> graph =
   Graph.fromCollection(edges,
      new MapFunction<String,Long>() {
         public Long map(String vertexId) {
            return 1l; }
      }, env);
```

Available Methods

- Graph Properties
  - getVertexIds
  - getEdgeIds
  - numberOfVertices
  - numberOfEdges
  - getDegrees
  - isWeaklyConnected
  - ...
- Transformations
  - map, filter, join
  - subgraph, union, difference
  - reverse, undirected
  - getTriplets
- Mutations
  - add vertex/edge
  - remove vertex/edge

Example: mapVertices

increment vertex values by 1



Example: mapVertices

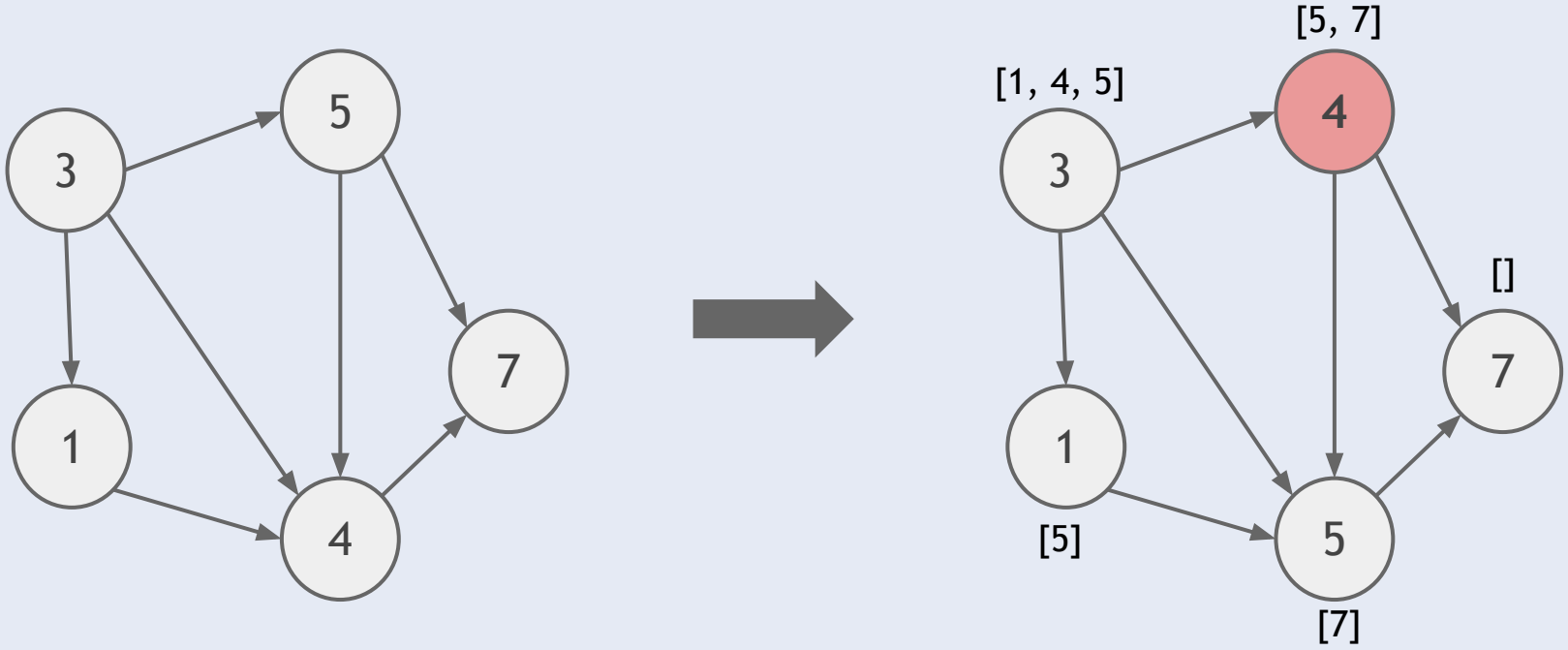
```
Graph<Long, Long, Long> updatedGraph =
   graph.mapVertices(
   new MapFunction<Vertex<Long, Long>, Long>() {
     public Long map(Vertex<Long, Long> value) {
         return value.getValue() + 1;
       }
   });
```

Neighborhood Methods

- Apply a reduce function to the 1st-hop neighborhood of each vertex in parallel

```
graph.reduceOnNeighbors(
        new MinValue(), EdgeDirection.OUT);
```

Neighborhood Methods



Iterative Graph Processing

Gelly offers iterative graph processing abstractions on top of Flink’s Delta iterations

- vertex-centric (similar to Pregel, Giraph)
- gather-sum-apply (similar to PowerGraph)

Vertex-centric Iterations

- **`MessagingFunction`**: what messages to send to other vertices
- **`VertexUpdateFunction`**: how to update a vertex value, based on the received messages

The workset contains only *active* vertices (received a msg in the previous superstep)

Vertex-centric SSSP

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

```
updateVertex(Vertex<K, Double> vertex,
             MessageIterator msgs) {

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

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

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

Gather-Sum-Apply Iterations

- **`Gather`**: how to ***transform*** each of the edges and neighbors of each vertex
- **`Sum`**: how to ***aggregate*** the partial values of `Gather` to a single value
- **`Apply`**: how to ***update*** the vertex value, based on the new aggregate and the current value

Gather-Sum-Apply SSSP

```
shortestPaths = graph.runGatherSumApplyIteration(new CalculateDistances(),
          new ChooseMinDistance(), new UpdateDistance()).getVertices();
```

CalculateDistances: **Gather**

```
gather(Neighbor<Double, Double> neighbor) {
                return neighbor.getNeighborValue() + neighbor.getEdgeValue();
}
```

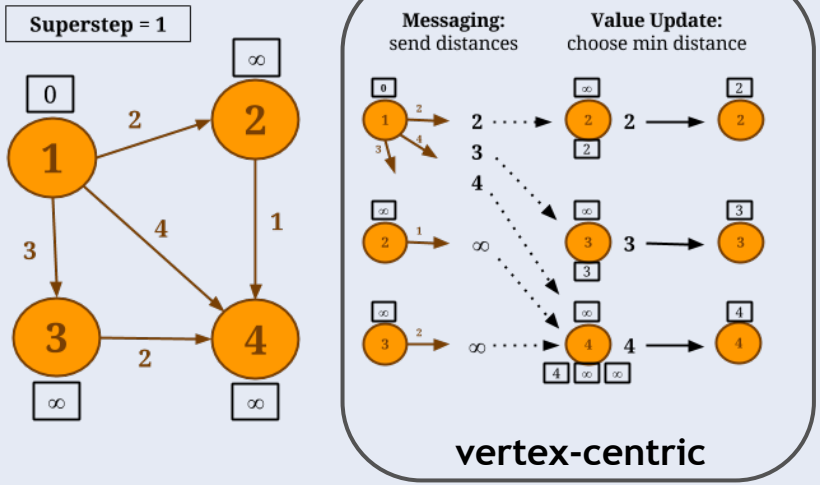
ChooseMinDistance: **Sum**

```
sum(Double newValue, Double currentValue) {
                return Math.min(newValue, currentValue);
}
```

ChooseMinDistance: **Apply**

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

Vertex-centric or GSA?



when all messages are needed for the update

when a vertex needs to send messages to non-neighbors

when message creation is independent & expensive (parallelized in Gather)

when the graph is skewed (~ 1.5x faster)

if update is associative-commutative

Library of Algorithms

- PageRank*
- Single Source Shortest Paths*
- Label Propagation
- Weakly Connected Components*
- Community Detection

Upcoming: Triangle Count, HITS, Affinity Propagation, Graph Summarization

```
graphWithRanks = inputGraph.run(new PageRank(maxIterations, dampingFactor));
```

*: both vertex-centric and GSA implementations

Feeling Gelly?

- Grab the Flink master: https://github.com/apache/flink
- Read the Gelly programming guide: https://ci.apache.org/projects/flink/flink-docs-master/libs/gelly_guide.html
- Follow the Gelly School tutorials (Beta): http://gellyschool.com
- Read our blog post: https://flink.apache.org/news/2015/08/24/introducing-flink-gelly.html
- Come to Flink Forward (October 12-13, Berlin): http://flink-forward.org/