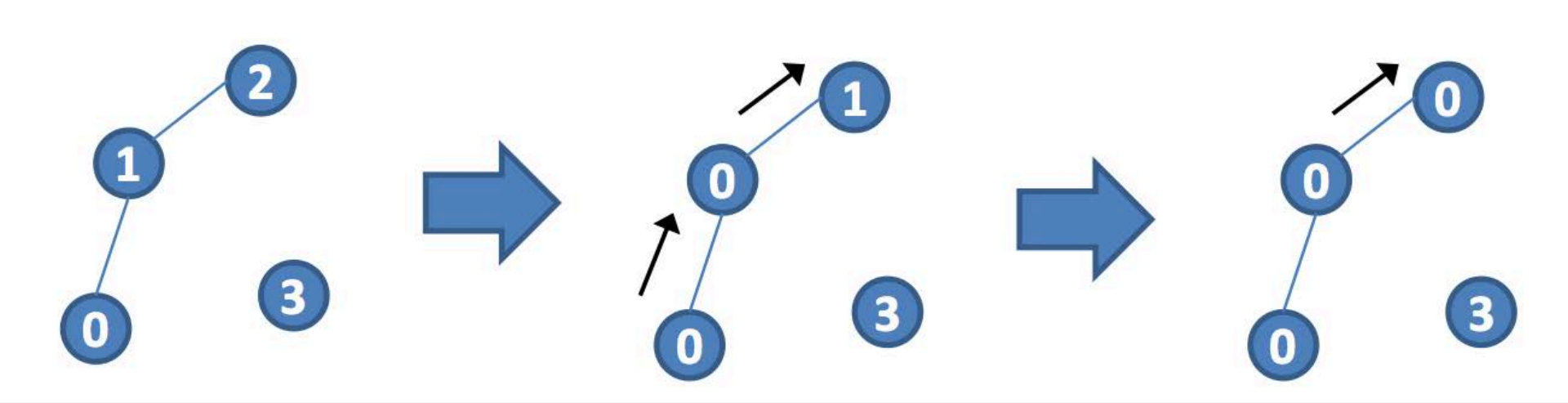
# CLOUD COMPUTING APPLICATIONS

## Giraph Example

Roy Campbell & Reza Farivar

# Giraph Ex: Connected Components of an Undirected Graph

- *Algorithm*: propagate smallest vertex label to neighbors until convergence




- In the end, all vertices of a component will have the same label

# Create a Custom Vertex

```
/*
 * Licensed to the Apache Software Foundation (ASF) under one
 * or more contributor license agreements.  See the NOTICE file
 * distributed with this work for additional information
 * regarding copyright ownership.  The ASF licenses this file
 * to you under the Apache License, Version 2.0 (the
 * "License"); you may not use this file except in compliance
 * with the License.  You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.apache.giraph.examples;

import org.apache.giraph.graph.IntIntNullIntVertex;
import org.apache.hadoop.io.IntWritable;

import java.io.IOException;
```

# Create a Custom Vertex

```
/**
 * Implementation of the HCC algorithm that identifies connected components and
 * assigns each vertex its "component identifier" (the smallest vertex id
 * in the component)
 *
 * The idea behind the algorithm is very simple: propagate the smallest
 * vertex id along the edges to all vertices of a connected component. The
 * number of supersteps necessary is equal to the length of the maximum
 * diameter of all components + 1
 *
 * The original Hadoop-based variant of this algorithm was proposed by Kang,
 * Charalampos, Tsourakakis and Faloutsos in
 * "PEGASUS: Mining Peta-Scale Graphs", 2010
 *
 * http://www.cs.cmu.edu/~ukang/papers/PegasusKAIS.pdf
 */
@Algorithm(
    name = "Connected components",
    description = "Finds connected components of the graph"
)
```

# Create a Custom Vertex

```java
public class ConnectedComponentsVertex extends IntIntNullIntVertex {
  /**
   * Propagates the smallest vertex id to all neighbors. Will always choose to
   * halt and only reactivate if a smaller id has been sent to it.
   *
   * @param messages Iterator of messages from the previous superstep.
   * @throws IOException
   */
  @Override
  public void compute(Iterable<IntWritable> messages) throws IOException {
    int currentComponent = getValue().get();

    // First superstep is special, because we can simply look at the neighbors
    if (getSuperstep() == 0) {
      for (IntWritable neighbor : getNeighbors()) {
        if (neighbor.get() < currentComponent) {
          currentComponent = neighbor.get();
        }
      }
      // Only need to send value if it is not the own id
      if (currentComponent != getValue().get()) {
        setValue(new IntWritable(currentComponent));
        for (IntWritable neighbor : getNeighbors()) {
          if (neighbor.get() > currentComponent) {
            sendMessage(new IntWritable(neighbor.get()), getValue());
          }
        }
      }

      voteToHalt();
      return;
    }

    boolean changed = false;
    // did we get a smaller id ?
    for (IntWritable message : messages) {
      int candidateComponent = message.get();
      if (candidateComponent < currentComponent) {
        currentComponent = candidateComponent;
        changed = true;
      }
    }

    // propagate new component id to the neighbors
    if (changed) {
      setValue(new IntWritable(currentComponent));
      sendMessageToAllEdges(getValue());
    }
    voteToHalt();
  }
}
```