Data-Intensive Distributed Computing

CS 431/631 451/651 (Fall 2019)

Part 8: Analyzing Graphs, Redux (1/2)

November 19, 2019

Ali Abedi

These slides are available at https://www.student.cs.uwaterloo.ca/~cs451

Characteristics of Graph Algorithms

Parallel graph traversals

Local computations

Message passing along graph edges

Iterations

Even faster?

Big Data Processing in a Nutshell

Let's be smarter about this!

Partition

Replicate

Reduce cross-partition communication

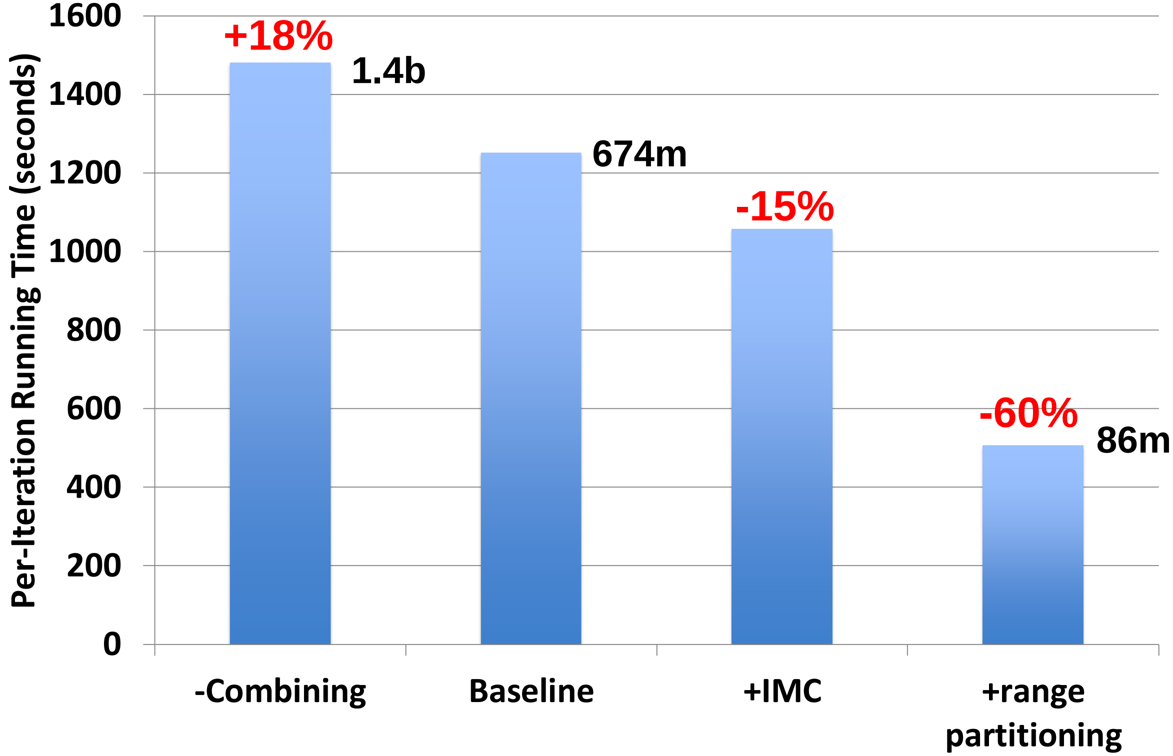
Simple Partitioning Techniques

Hash partitioning

Range partitioning on some underlying linearization

Web pages: lexicographic sort of domain-reversed URLs

How much difference does it make?



PageRank over webgraph
(40m vertices, 1.4b edges)

Lin and Schatz. (2010) Design Patterns for Efficient Graph Algorithms in MapReduce.

Simple Partitioning Techniques

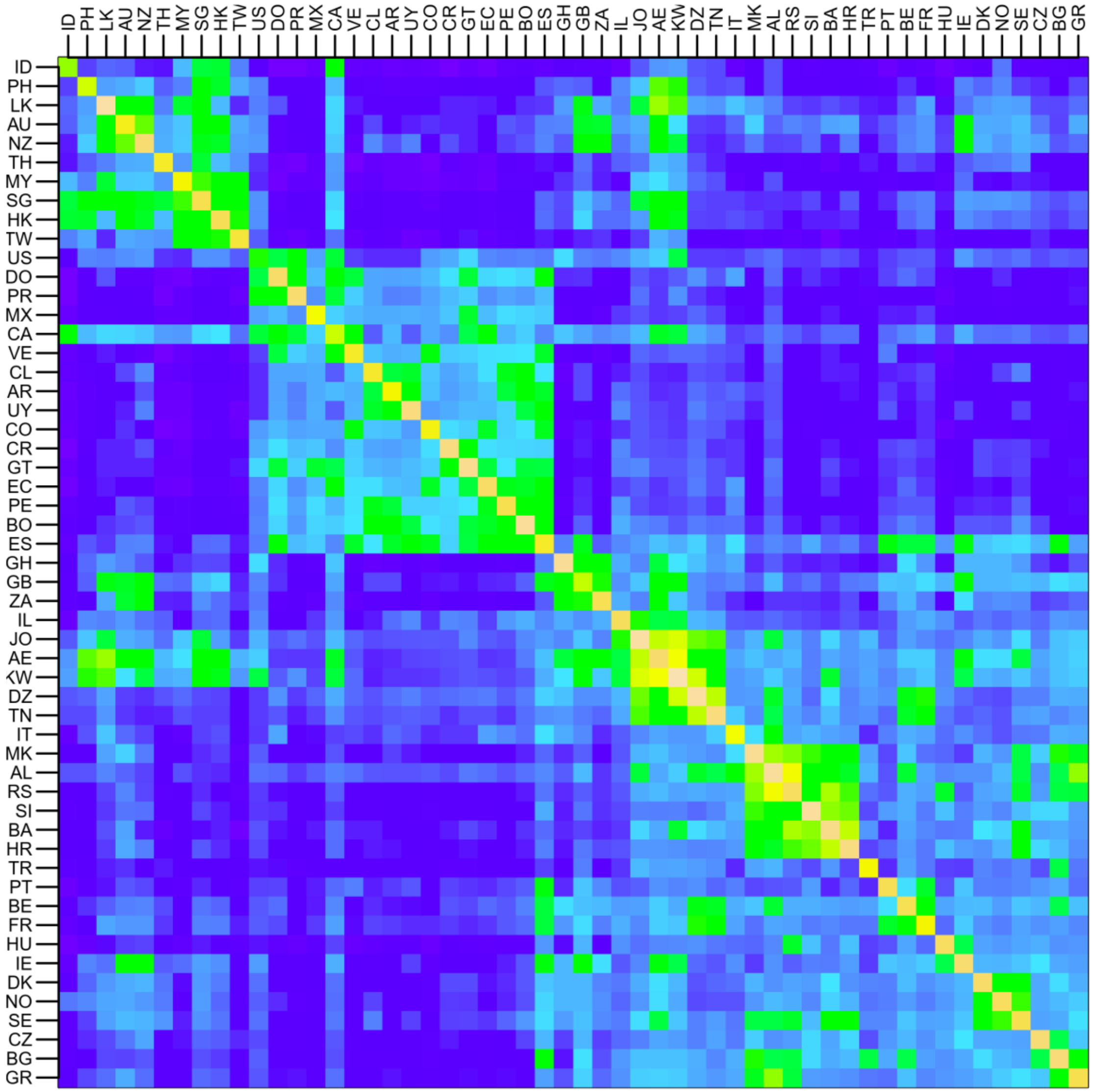
Hash partitioning

Range partitioning on some underlying linearization

Web pages: lexicographic sort of domain-reversed URLs

Social networks: sort by demographic characteristics

Country Structure in Facebook



Analysis of 721 million active users (May 2011)

54 countries w/ >1m active users, >50% penetration

Ugander et al. (2011) The Anatomy of the Facebook Social Graph.

Simple Partitioning Techniques

Hash partitioning

Range partitioning on some underlying linearization

Web pages: lexicographic sort of domain-reversed URLs

Social networks: sort by demographic characteristics

But what about graphs in general?

Source: http://www.flickr.com/photos/fusedforces/4324320625/

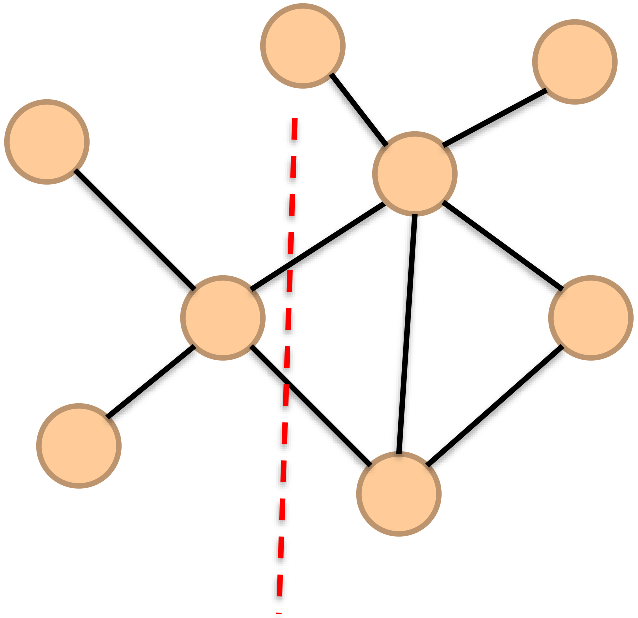
Big Data Processing in a Nutshell

Partition

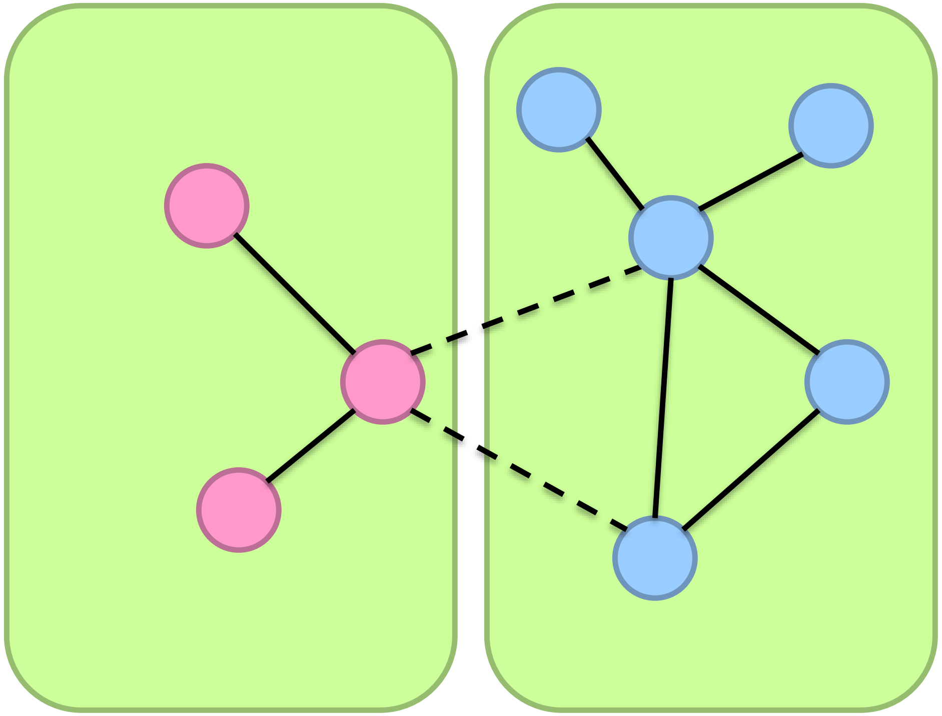
Replicate

Reduce cross-partition communication

Partition

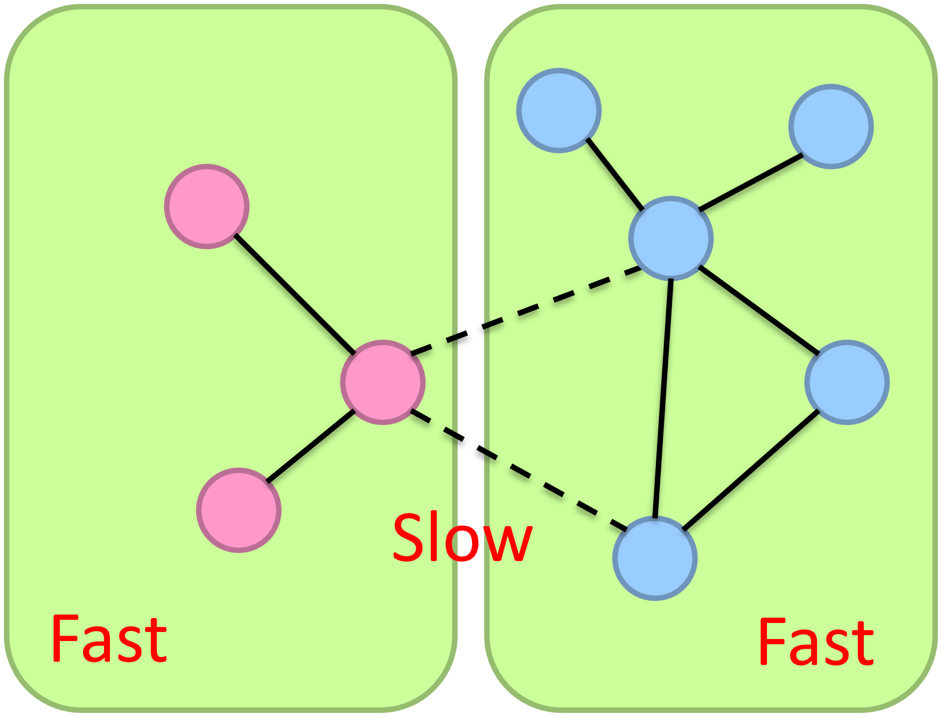


Partition

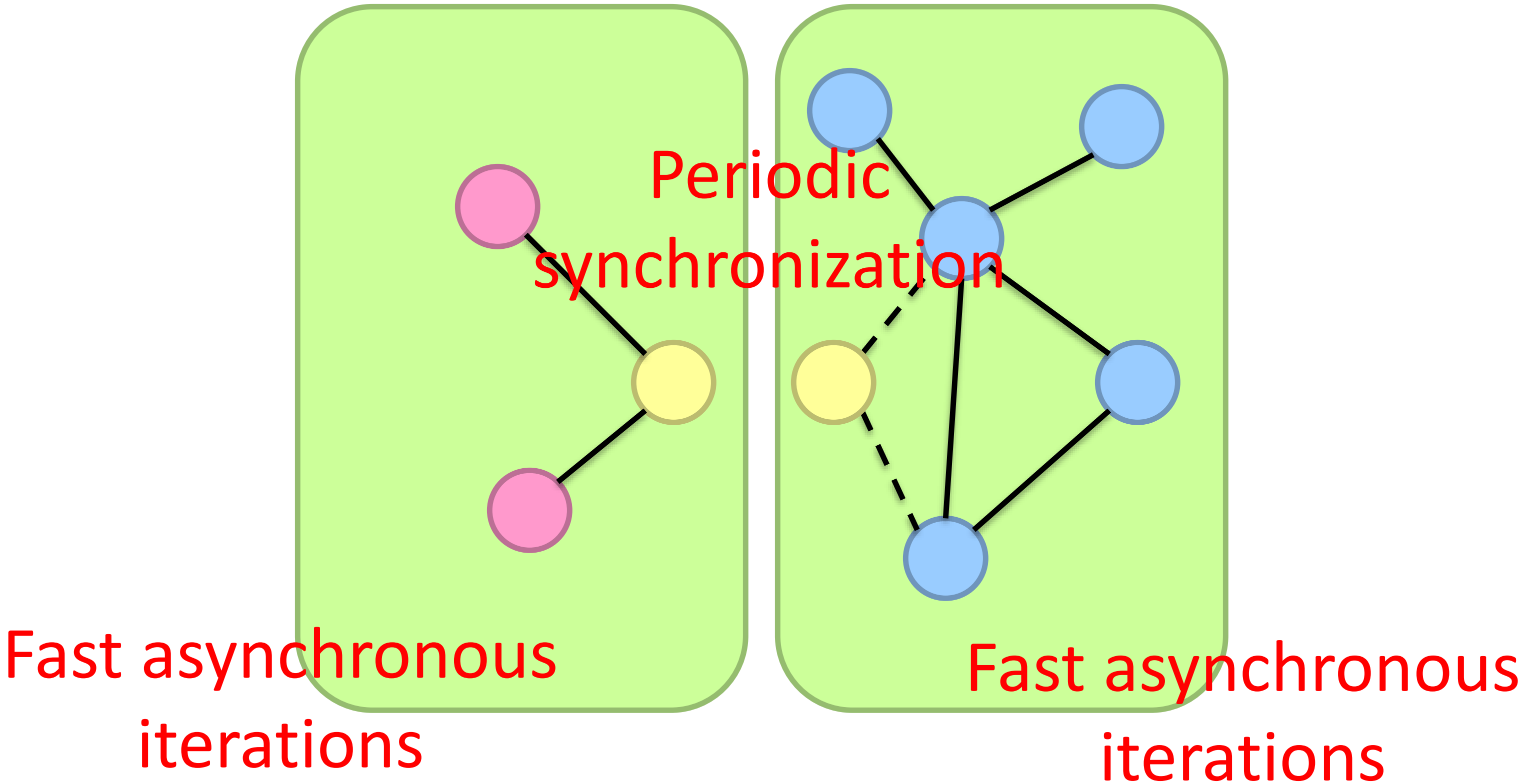


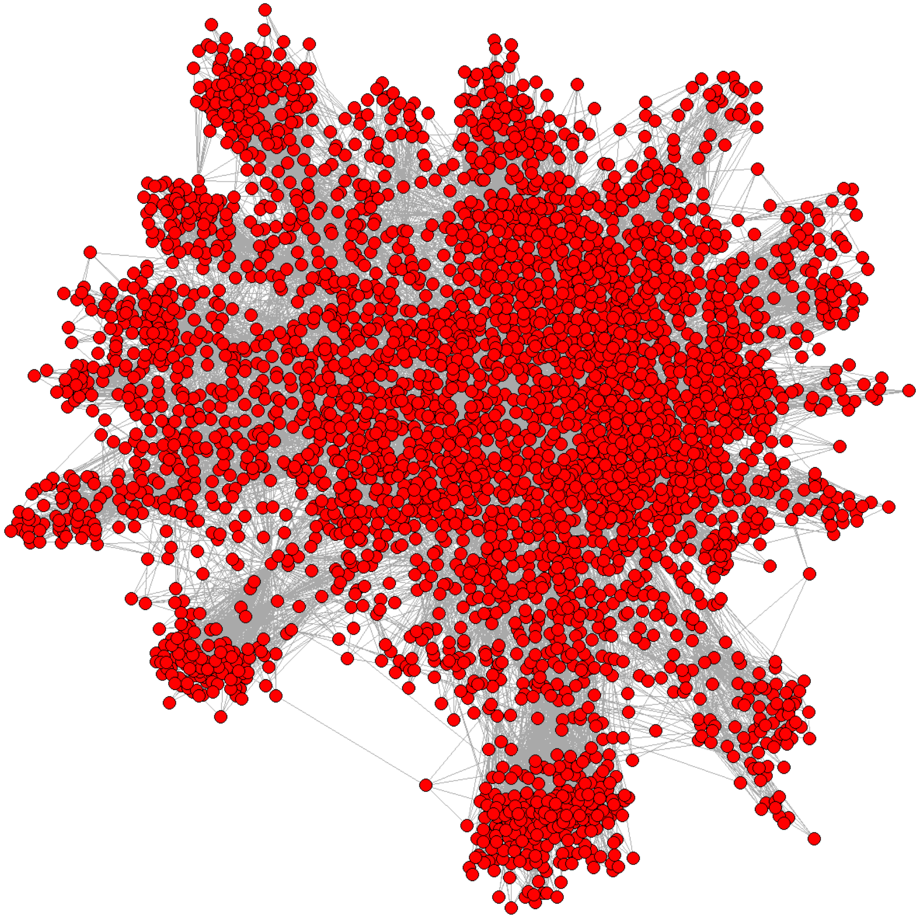
What’s the fundamental issue?

Partition



State-of-the-Art Distributed Graph Algorithms





Graph Processing Frameworks

Graph Processing Frameworks

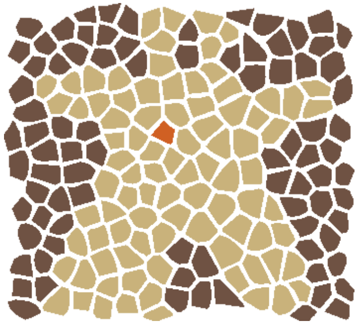
- Pregel
  - Google
- Apache Giraph
  - Based on Pregel
  - On Hadoop
- Spark GraphX

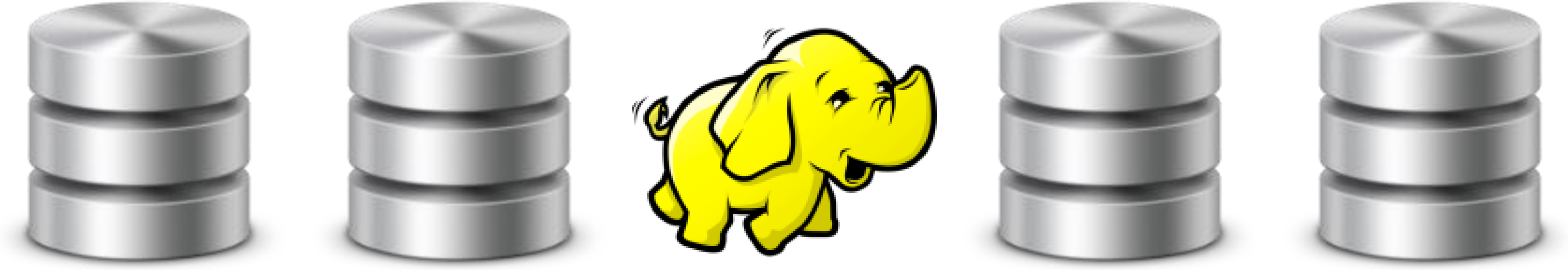
APACHE

GIRAPH

What is Apache Giraph

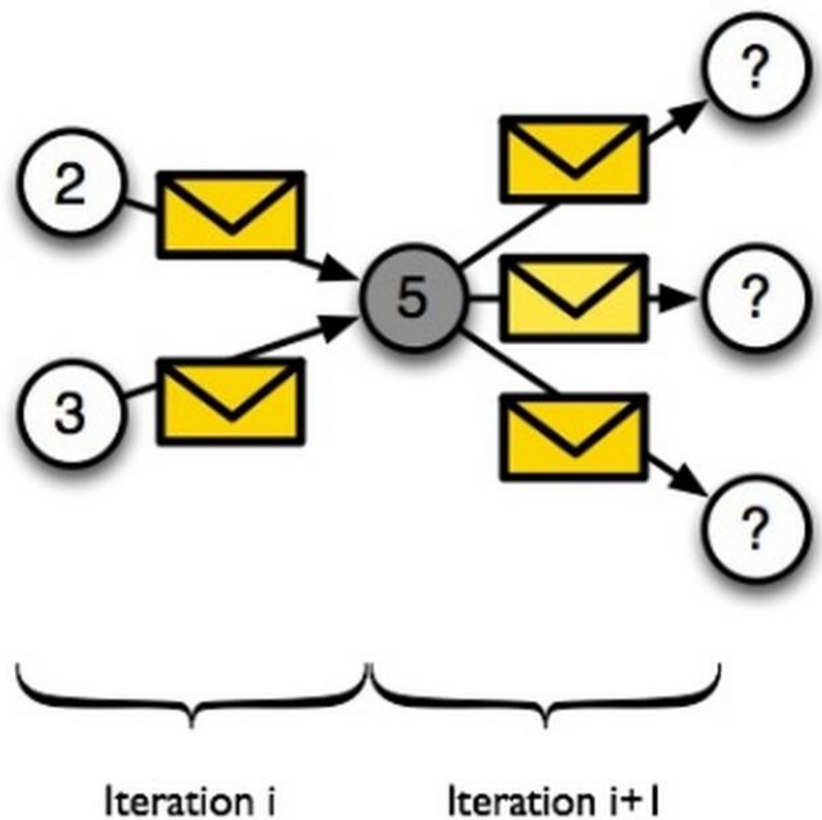
- Giraph performs iterative calculation on top of an existing Hadoop cluster





Bulk-Synchronous Parallel (BSP) Programming Model

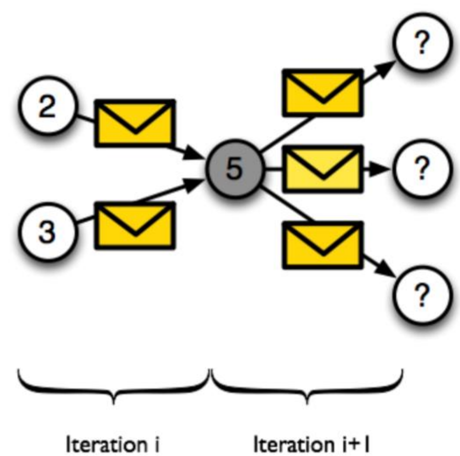
Vertex-centric model



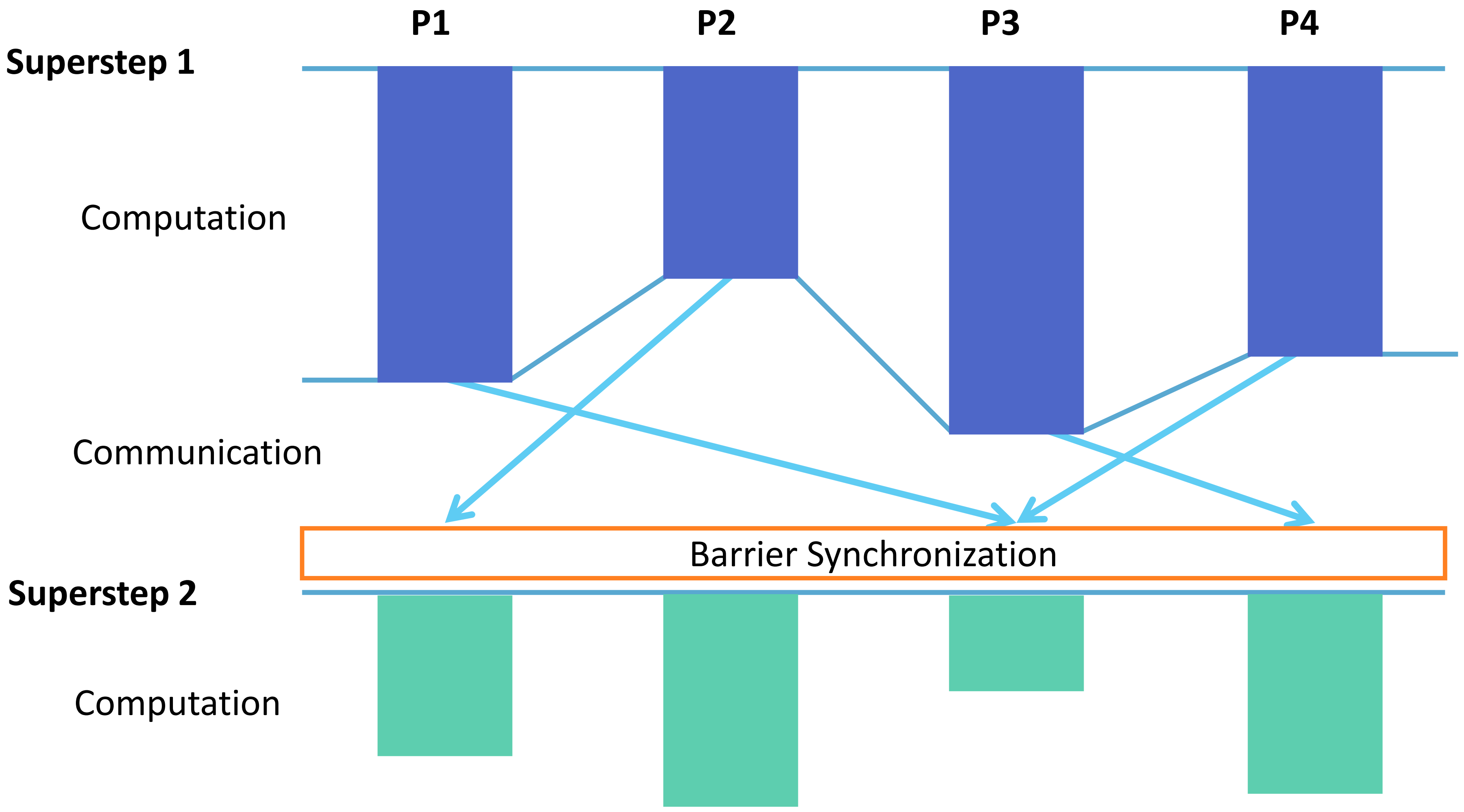
Vertex Centric Programming

- Vertex Centric Programming Model
  - Logic written from perspective on a single vertex.
  - Executed on all vertices.

- Vertices know about
  - Their own value(s)
  - Their outgoing edges

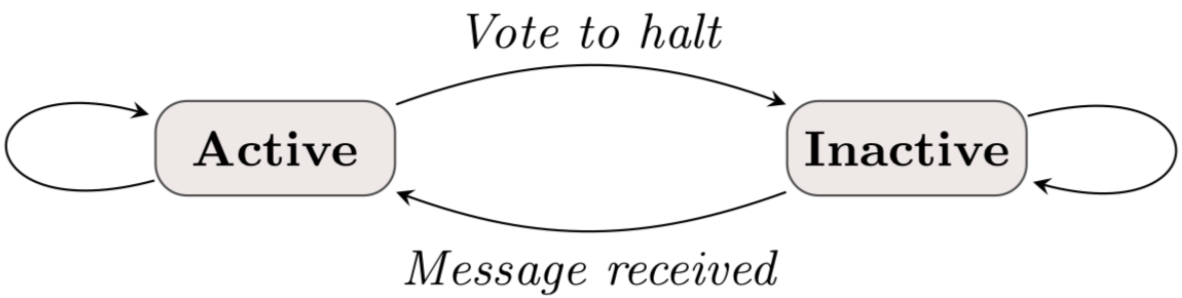


Bulk-Synchronous Parallel (BSP) Programming Model



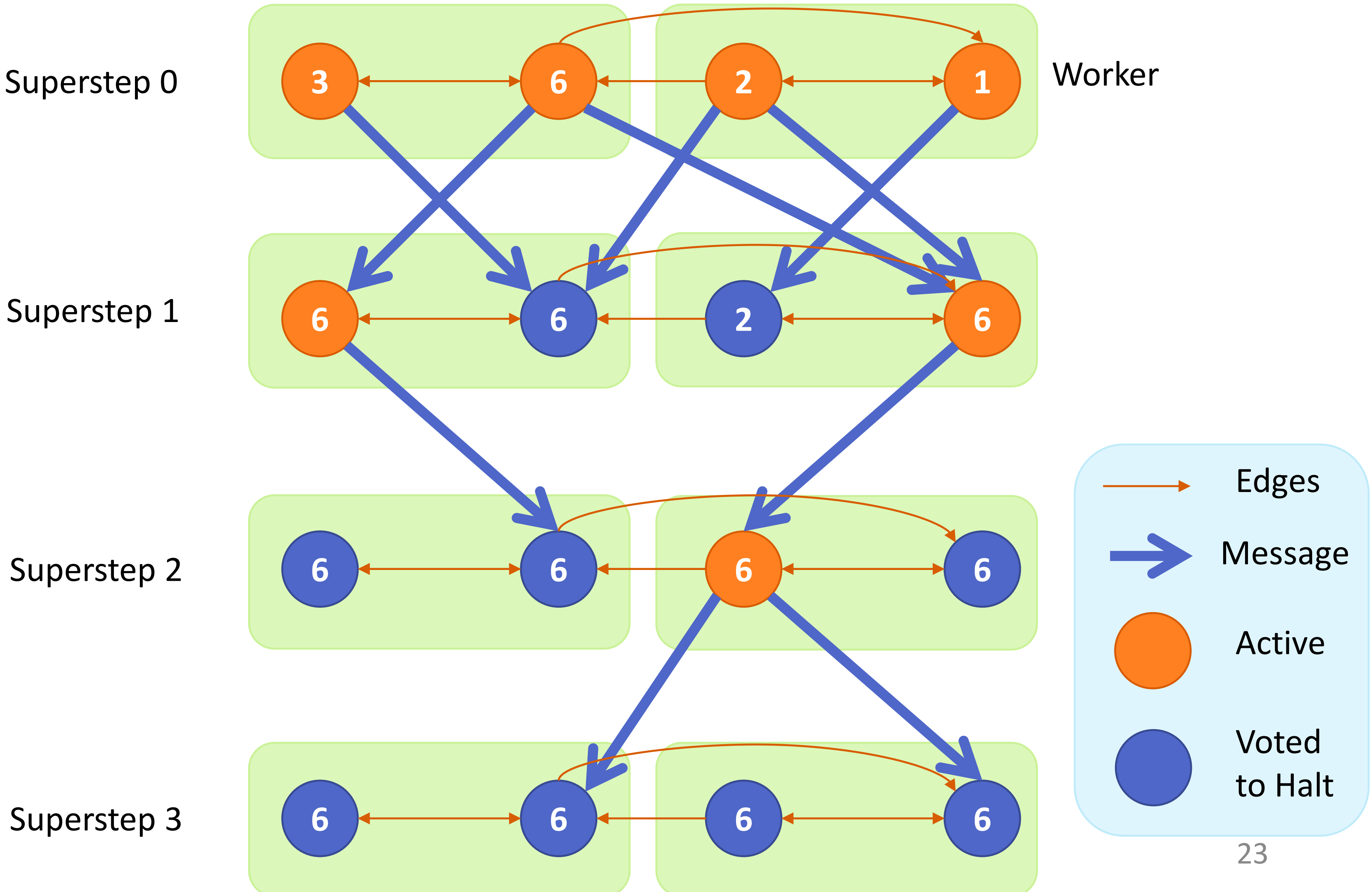
"Often *expensive* and should be *used as sparingly* as possible"

Vertex State Machine



- In superstep 0, every vertex is in the active state.
- A vertex deactivates itself by voting to halt.
- It can be reactivated by receiving an (external) message.
- Algorithm termination is based on every vertex voting to halt.

Finding Largest Value in a Graph



Advantages

- Makes distributed programming easy
  - No locks, semaphores, race conditions
  - Separates computing from communication phase
- Vertex-level parallelization
  - Bulk message passing for efficiency
- Stateful (in-memory)
  - Only messages & checkpoints hit disk

Giraph Architecture

Master – Application coordinator

Synchronizes supersteps

Assigns partitions to workers before superstep begins

Workers – Computation & messaging

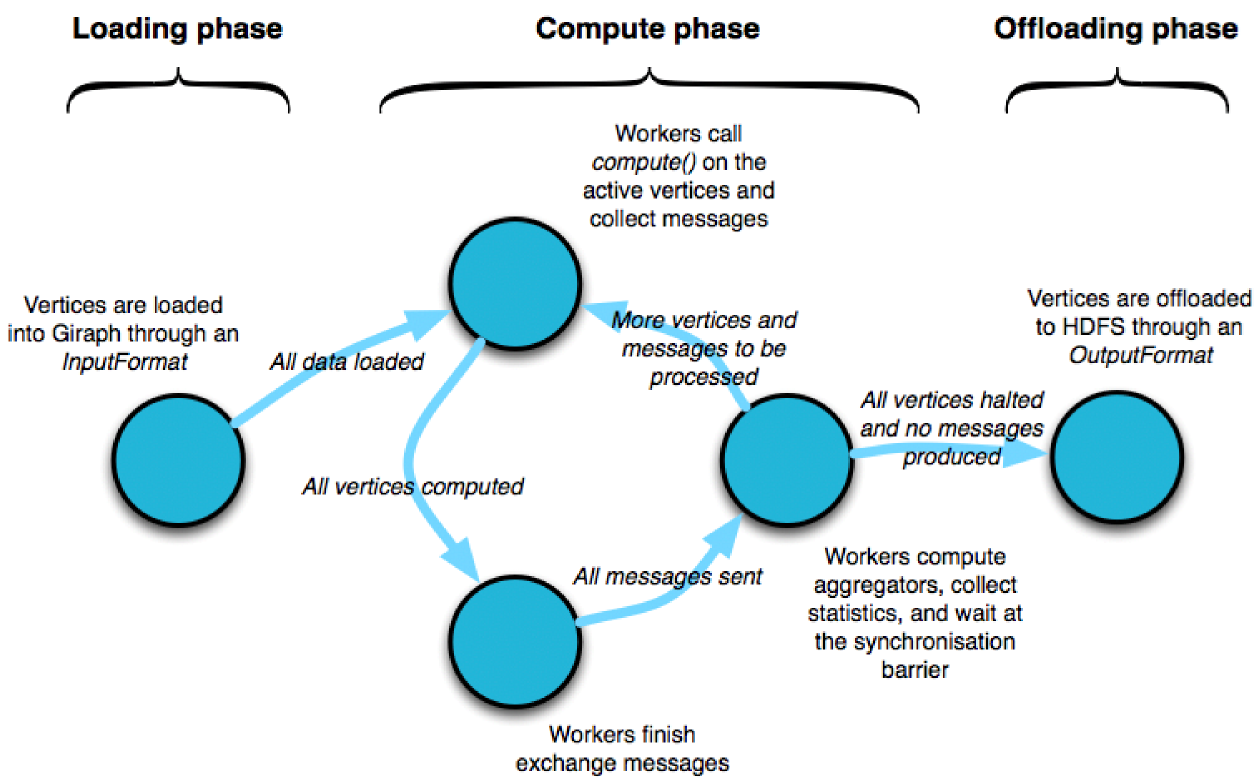
Handle I/O – reading and writing the graph

Computation/messaging of assigned partitions

ZooKeeper

Maintains global application state

Lifecycle of a Pregel Program



Apache Giraph, Claudio Martella, Hadoop Summit, Amsterdam, April 2014

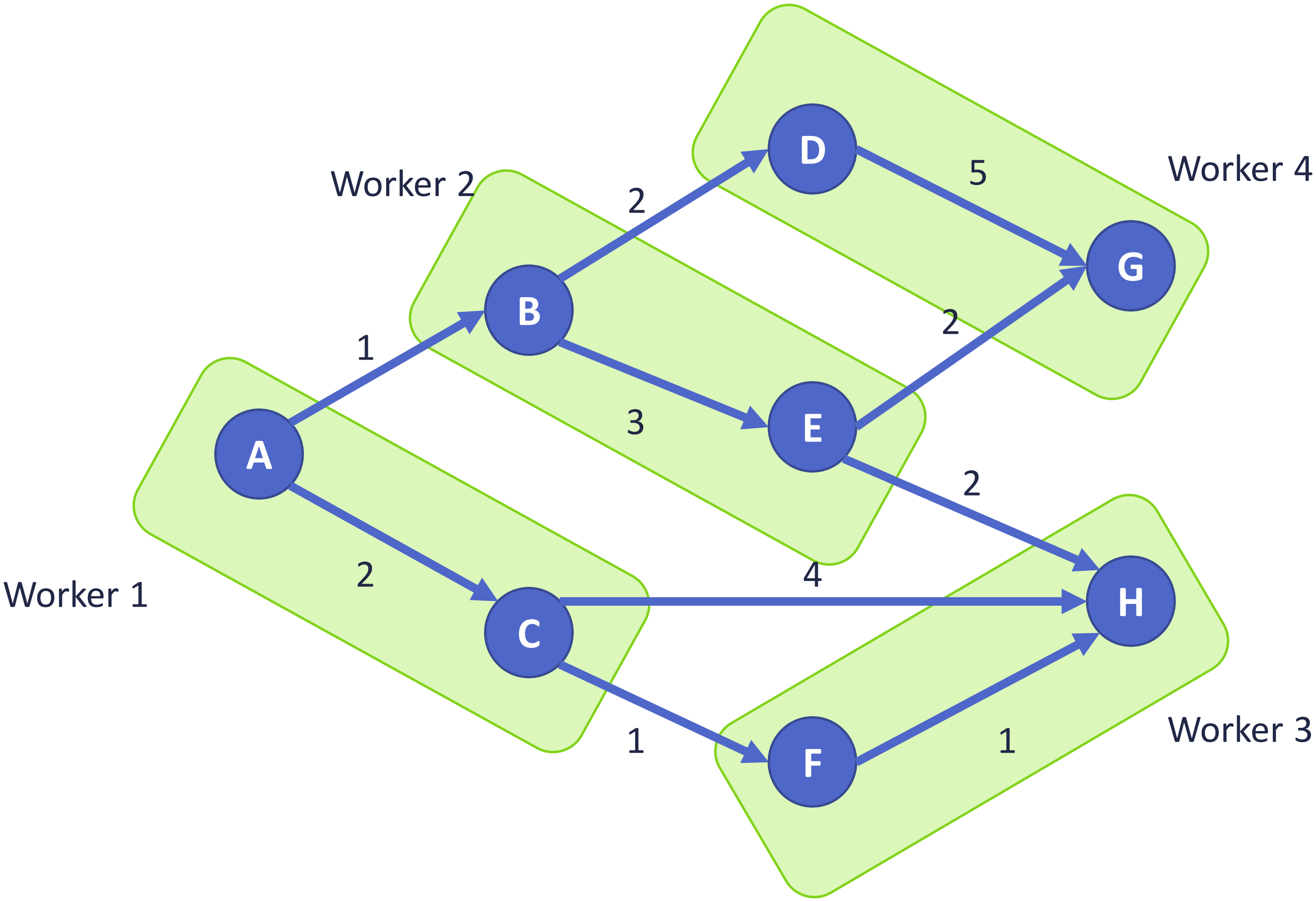
Applications

Shortest path

Example compute() method

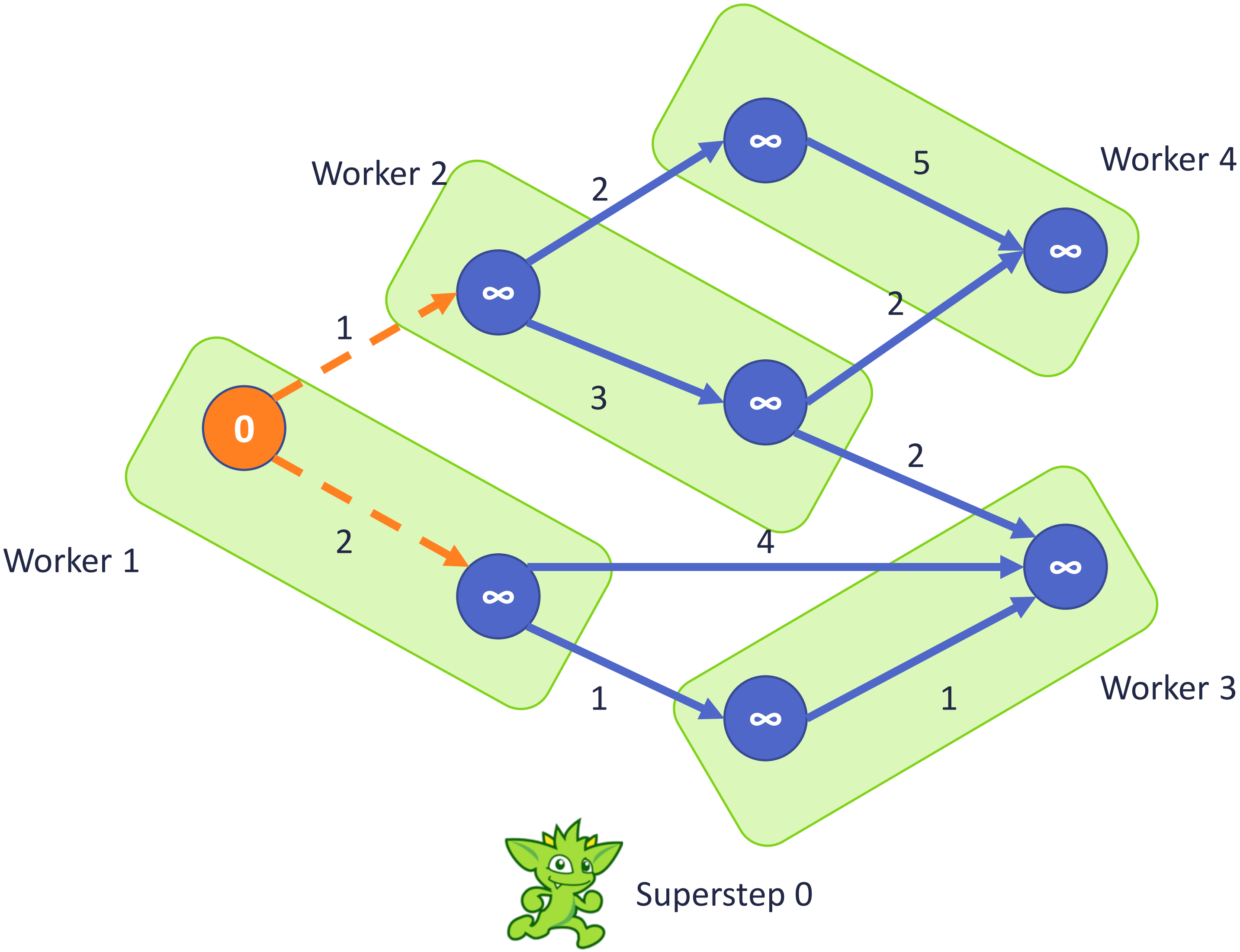
```
public void compute(Iterable<DoubleWritable> messages) {
  double minDist = Double.MAX_VALUE;
  for (DoubleWritable message : messages) {
    minDist = Math.min(minDist, message.get());
  }
  if (minDist < getValue().get()) {
    setValue(new DoubleWritable(minDist));
    for (Edge<LongWritable, FloatWritable> edge : getEdges()) {
      double distance = minDist + edge.getValue().get();
      sendMessage(edge.getTargetVertexId(), new DoubleWritable(distance));
    }
  }
  voteToHalt();
}
```

SSSP (1/6)

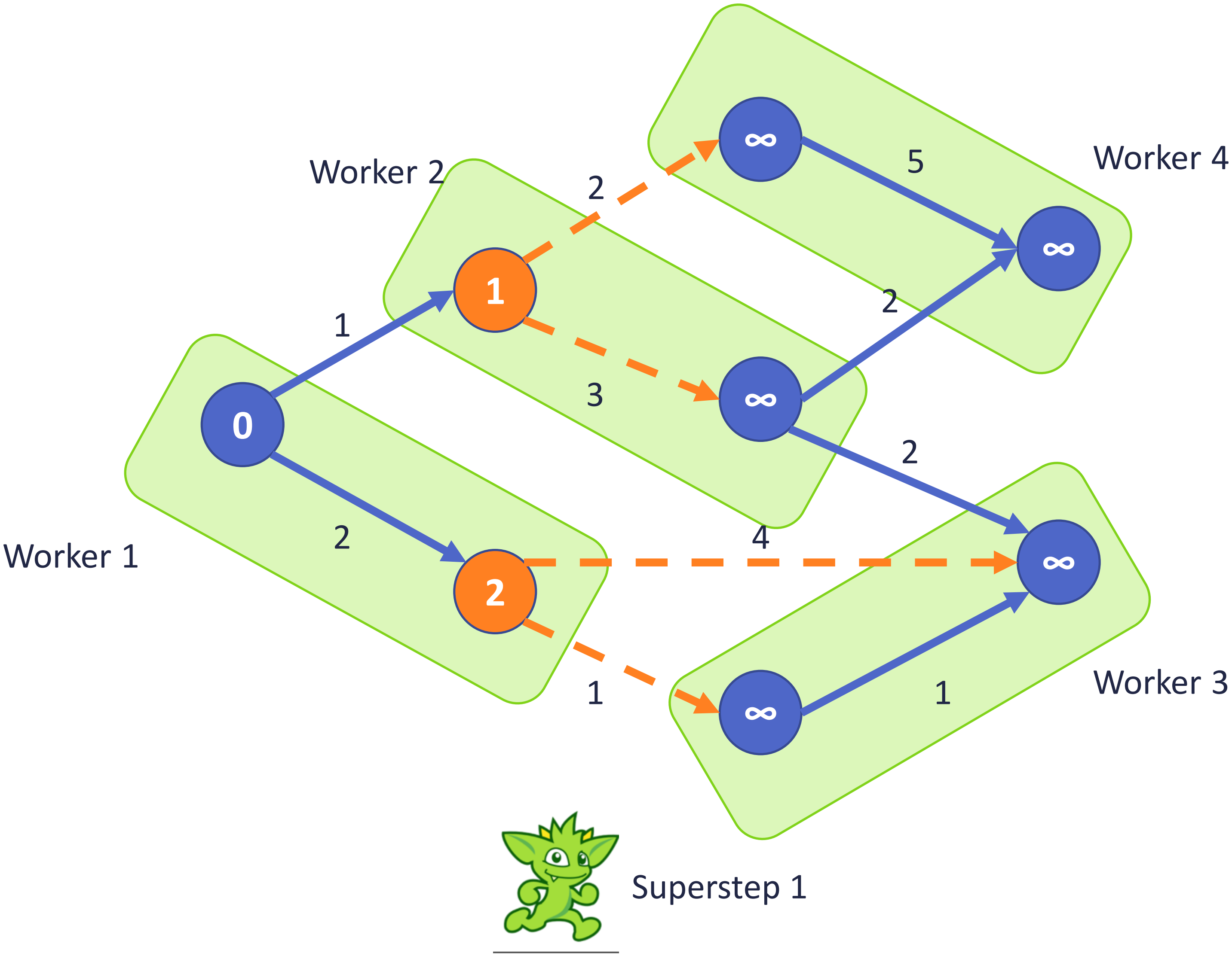


Input Graph

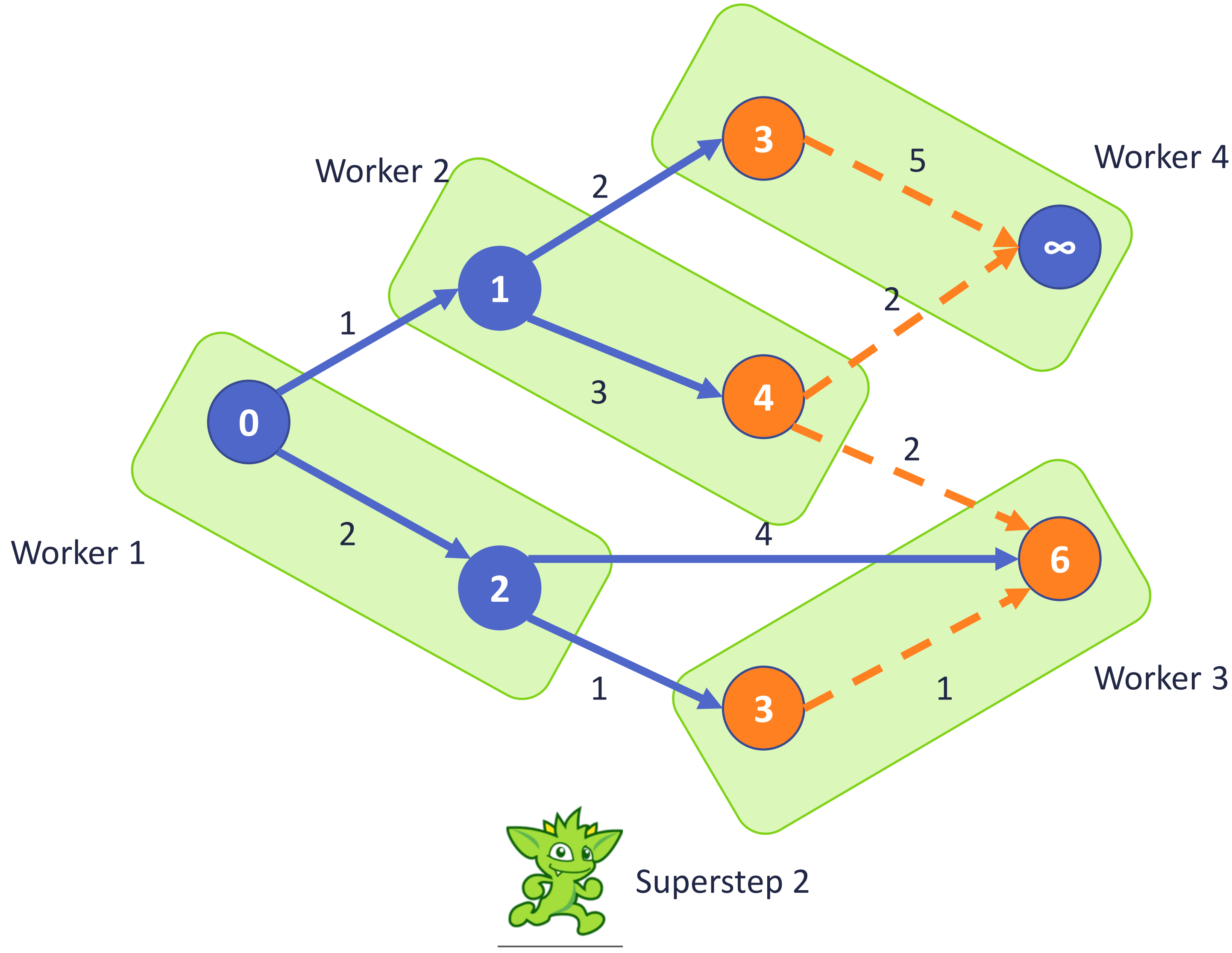
SSSP (2/6)



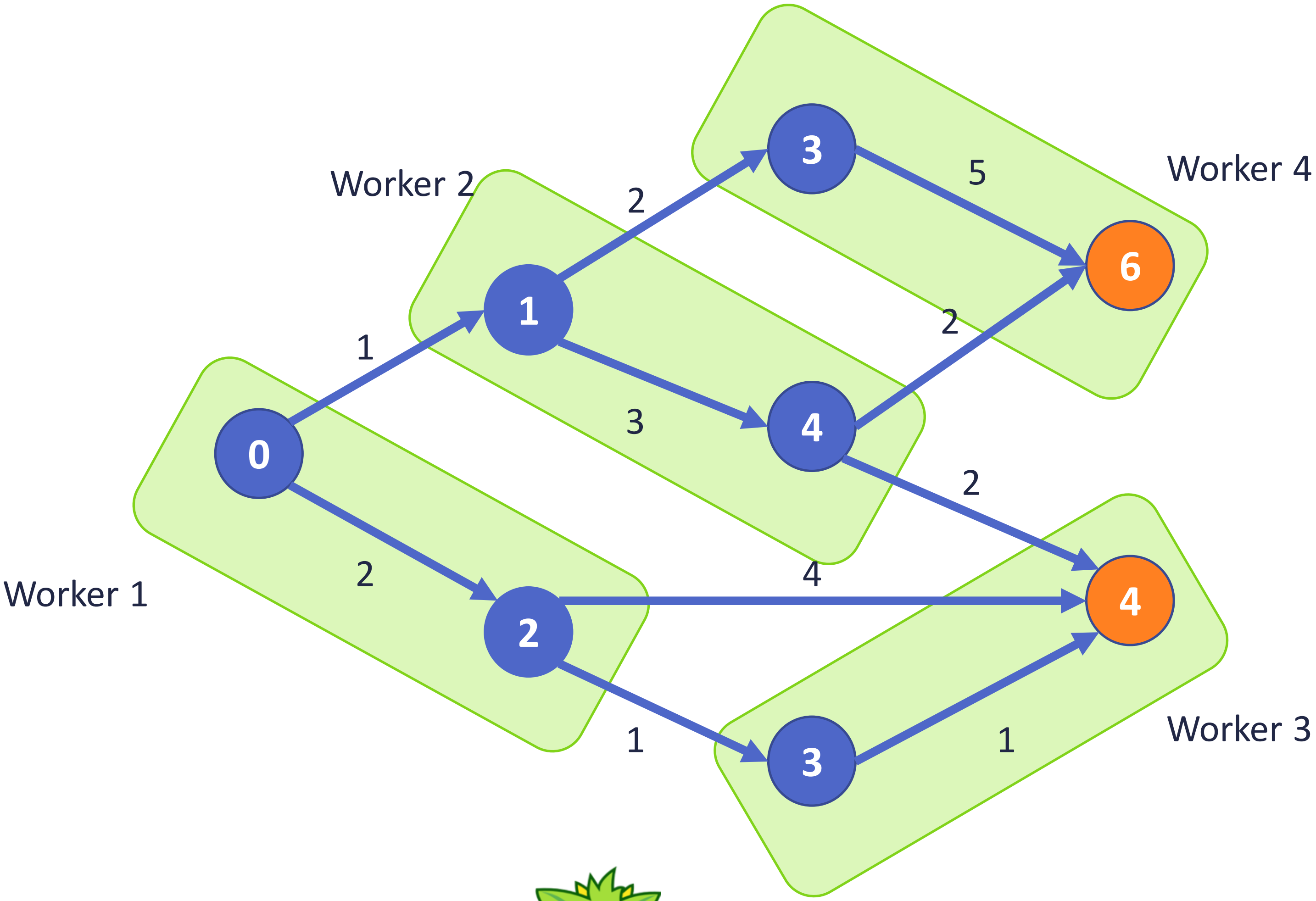
SSSP (3/6)


Worker 1
Worker 2
Worker 3
Worker 4
0
1
2
∞
∞
∞
∞
∞
1
2
2
3
5
2
2
4
1
1
Superstep 1

SSSP (4/6)

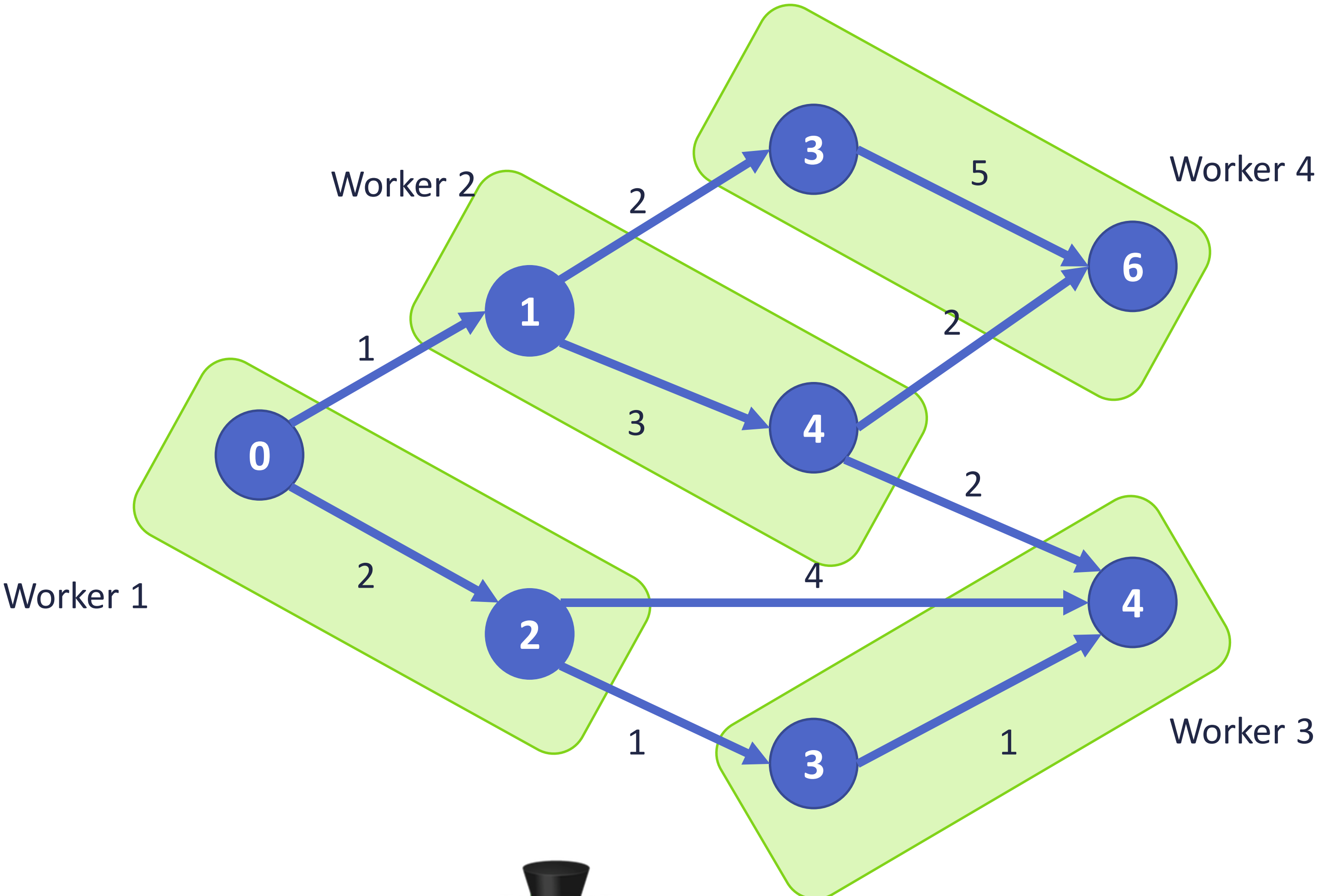

Worker 2
Worker 4
Worker 1
Worker 3
0
1
2
3
4
3
6
∞
1
2
2
3
4
1
5
2
2
1
Superstep 2

SSSP (5/6)



Superstep 3

SSSP (6/6)



Algorithm has converged

Finding the Max

```
public class MaxComputation extends BasicComputation<IntWritable, IntWritable,
    NullWritable, IntWritable> {
  @Override
  public void compute(Vertex<IntWritable, IntWritable, NullWritable> vertex,
      Iterable<IntWritable> messages) throws IOException
  {
    boolean changed = false;
    for (IntWritable message : messages) {
      if (vertex.getValue().get() < message.get()) {
        vertex.setValue(message);
        changed = true;
      }
    }
    if (getSuperstep() == 0 || changed) {
      sendMessageToAllEdges(vertex, vertex.getValue());
    }
    vertex.voteToHalt();
  }
}
```

Execution Trace