ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

Big Data Systems for Graphs

Yannis Kotidis

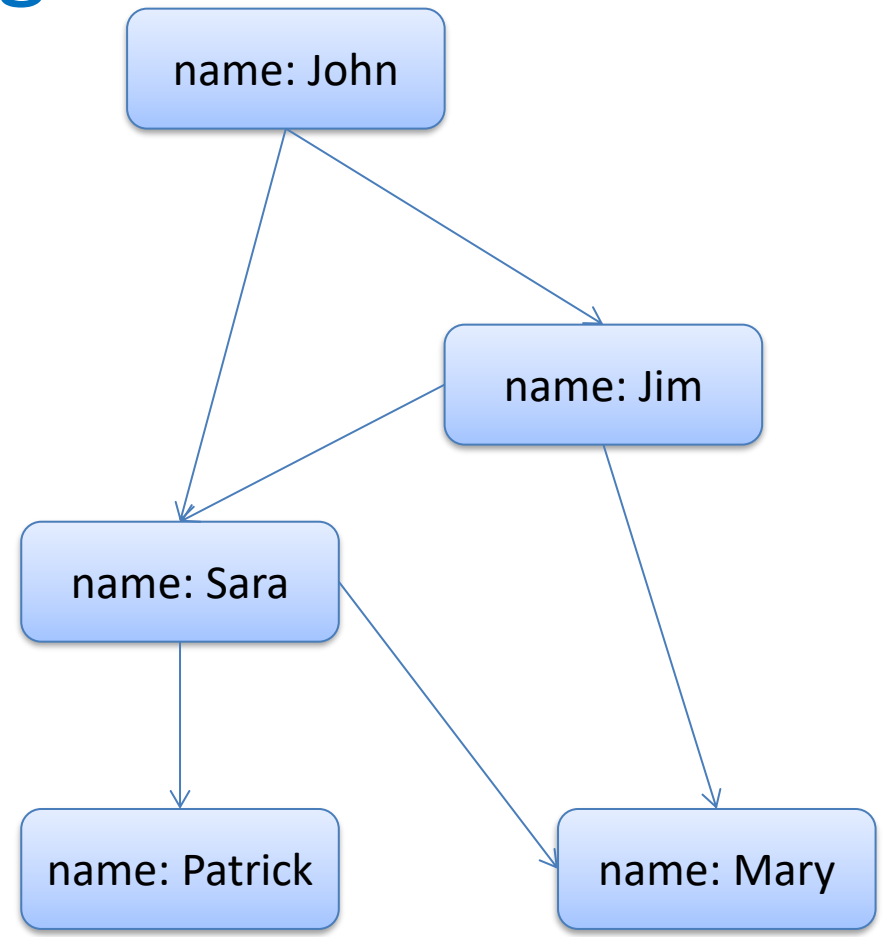
http://pages.cs.aueb.gr/~kotidis/

Processing Graph Data with Apache Spark

- **GraphX** (RDD-based)
  - Built on Spark's Resilient Distributed Datasets (RDDs)
  - Provides a property graph abstraction (Graph[VD, ED])
  - Includes built-in algorithms (PageRank, Connected Components, Triangle Counting)
- **GraphFrames** (DataFrame-based)
  - Built on Spark SQL's DataFrame/Dataset API
  - Enables graph queries using familiar DataFrame operations
  - Integrates with Spark SQL, MLlib, and supports motif finding
- Both support the **Pregel API** for iterative, message-passing algorithms
  - Inspired by Google's Pregel bulk-synchronous parallel model
  - Useful for iterative algorithms like PageRank, shortest paths, community detection
  - Provides a flexible way to implement custom graph algorithms

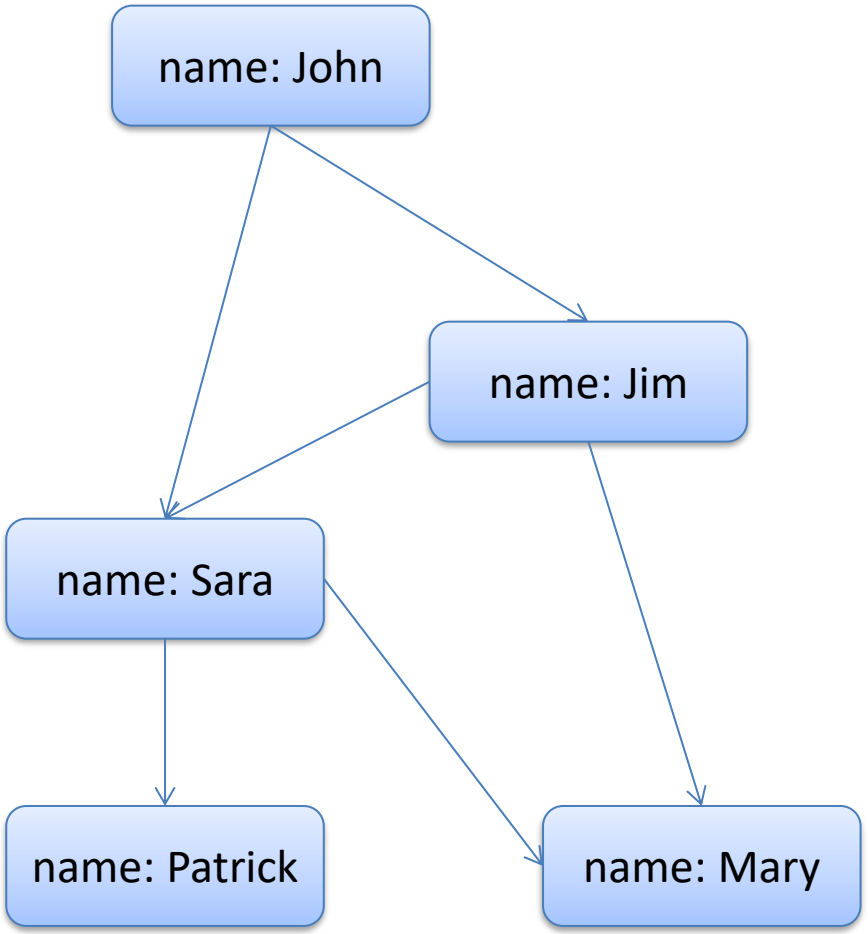
Friend suggestions example: Define nodes using a DataFrame

```
val v =
   spark.sqlContext.create
   DataFrame(List(
  ("john", "John", 29),
  ("sara", "Sara", 22),
  ("jim", "Jim", 42),
  ("patrick", "Patrick",19),
  ("mary", "Mary", 31)
)).toDF("id", "name",
   "age")
```



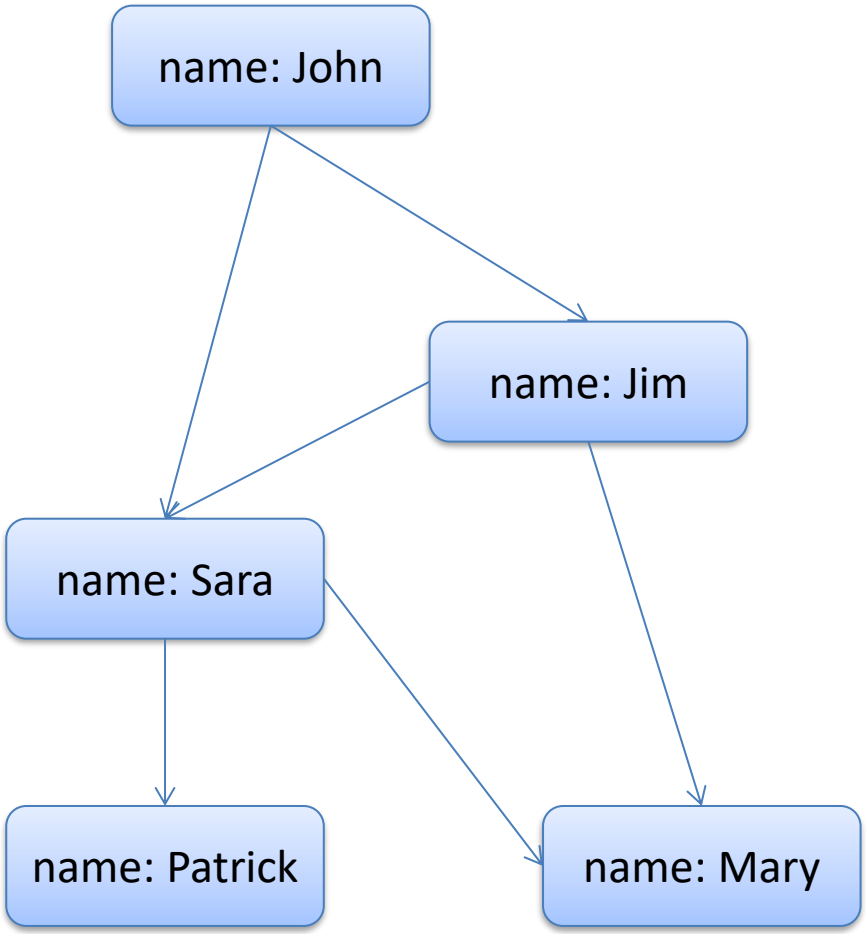
Now Define Edges

```
val e =
    spark.sqlContext.createData
    Frame(List(
  ("john", "sara", "knows"),
  ("john", "jim", "knows"),
  ("jim", "sara", "knows"),
  ("jim","mary","knows"),
  ("sara", "patrick", "knows"),
  ("sara", "mary", "knows")
)).toDF("src", "dst",
    "relationship")
```



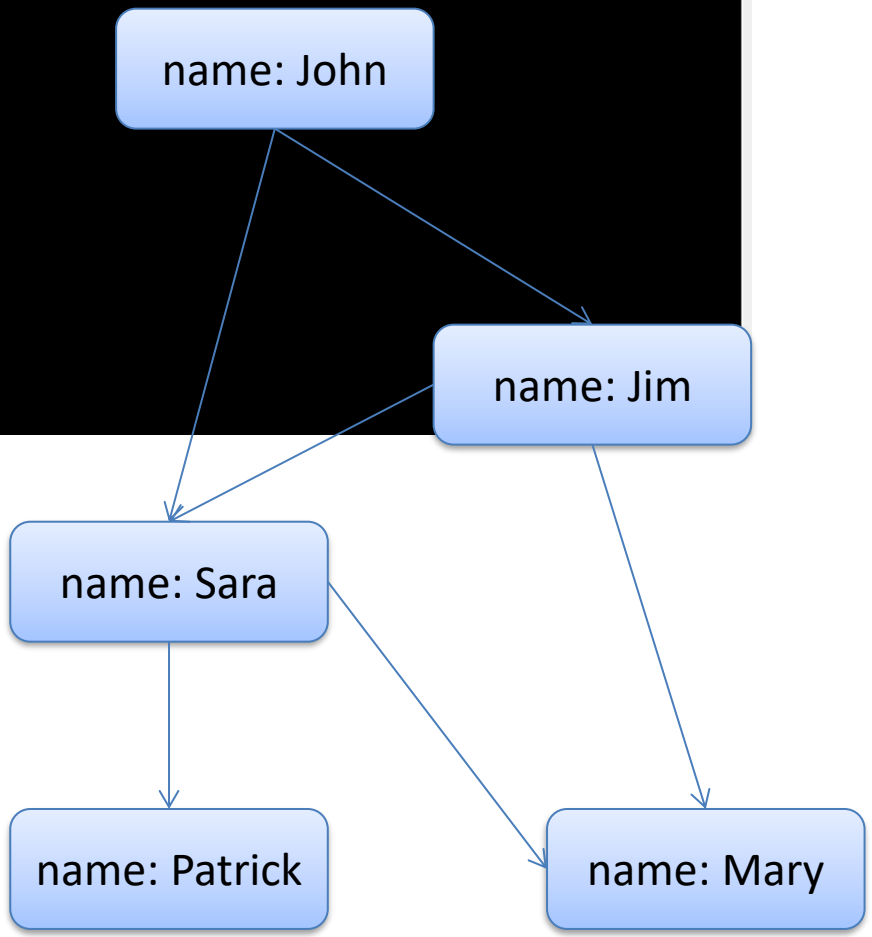
Create GraphFrame, run Motif

```
val g = GraphFrame(v, e)


g.find(
"(x)-[]->(f); (f)-[]->(fof);
   !(x)-[]->(fof)").
select("x","fof").groupBy("x
    ","fof").count.orderBy("c
    ount").show()
```



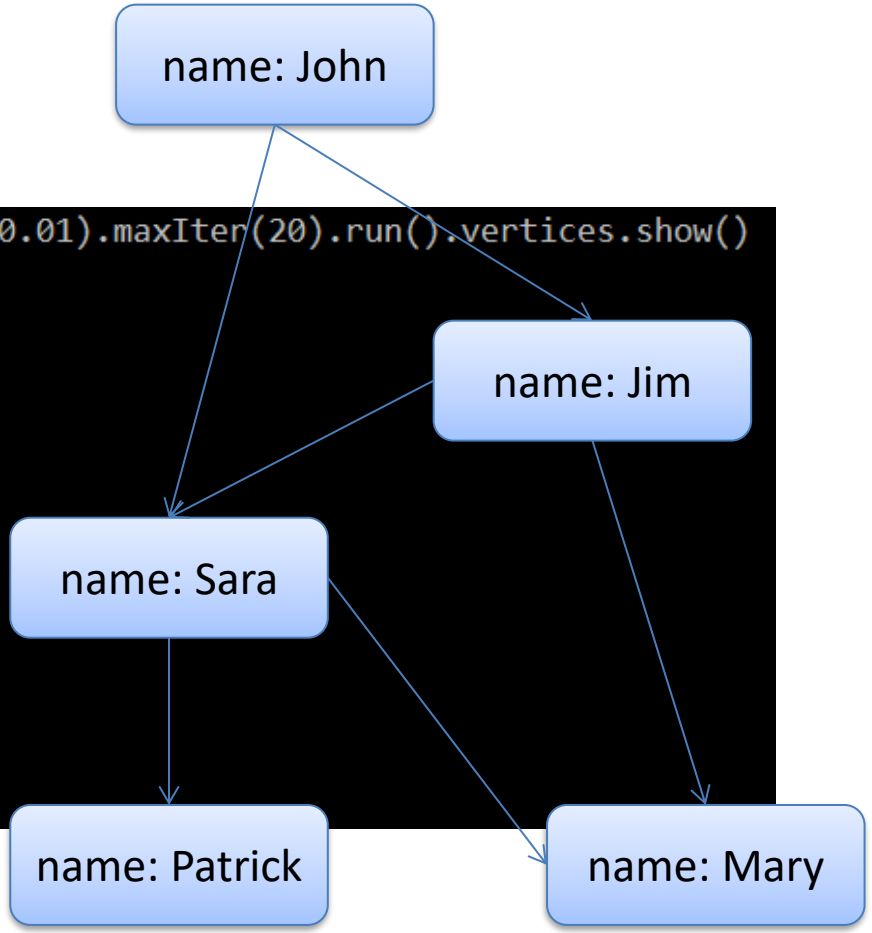
Result

```
scala> g.find("(x)-[]->(f); (f)-[]->(fof); !(x)-[]->(fof)").select("x","fof").groupBy("x","fof")
.count.orderBy("count").show()
+---------------+-------------------+-----+
|              x|                fof|count|
+---------------+-------------------+-----+
|  [jim, Jim, 42]|[patrick, Patrick...|    1|
|[john, John, 29]|[patrick, Patrick...|    1|
|[john, John, 29]|    [mary, Mary, 31]|    2|
+---------------+-------------------+-----+


scala>
```



PageRank Example

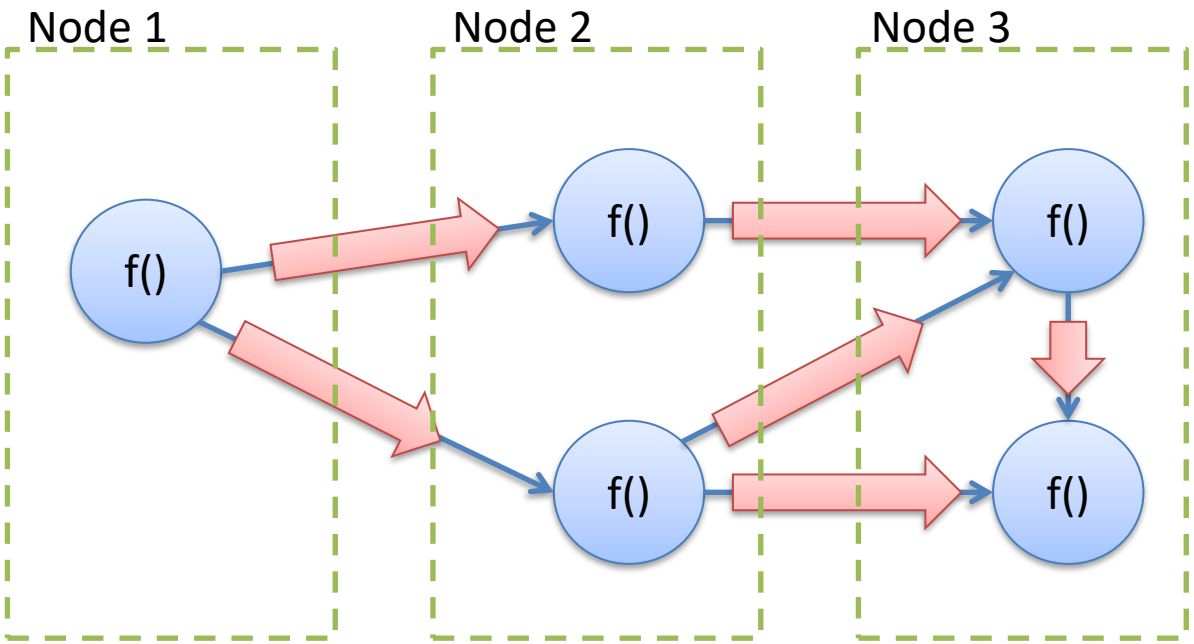
```
scala> val results = g.pageRank.resetProbability(0.01).maxIter(20).run().vertices.show()
+-------+-------+---+------------------+
|     id|   name|age|          pagerank|
+-------+-------+---+------------------+
|   mary|   Mary| 31|1.4698147724378927|
|   john|   John| 29|0.5163835727128357|
|   sara|   Sara| 22|1.1541301946025058|
|    jim|    Jim| 42|0.7719934412056895|
|patrick|Patrick| 19|1.0876780190410762|
+-------+-------+---+------------------+

results: Unit = ()

scala>
```



Think like a vertex!

- Many graph algorithms (e.g., PageRank, BFS, shortest paths) can be executed efficiently using parallel computation
- Vertex-Centric Programming Model: express the algorithm from the perspective of each graph node
  - **Parallel execution**: all vertices perform the same computation simultaneously
  - **Message passing**: vertices exchange messages with their neighbors at every step
  - **Synchronization**: computation proceeds in iterative supersteps with a global barrier
  - **Convergence**: iterations continue until results stabilize or no messages remain

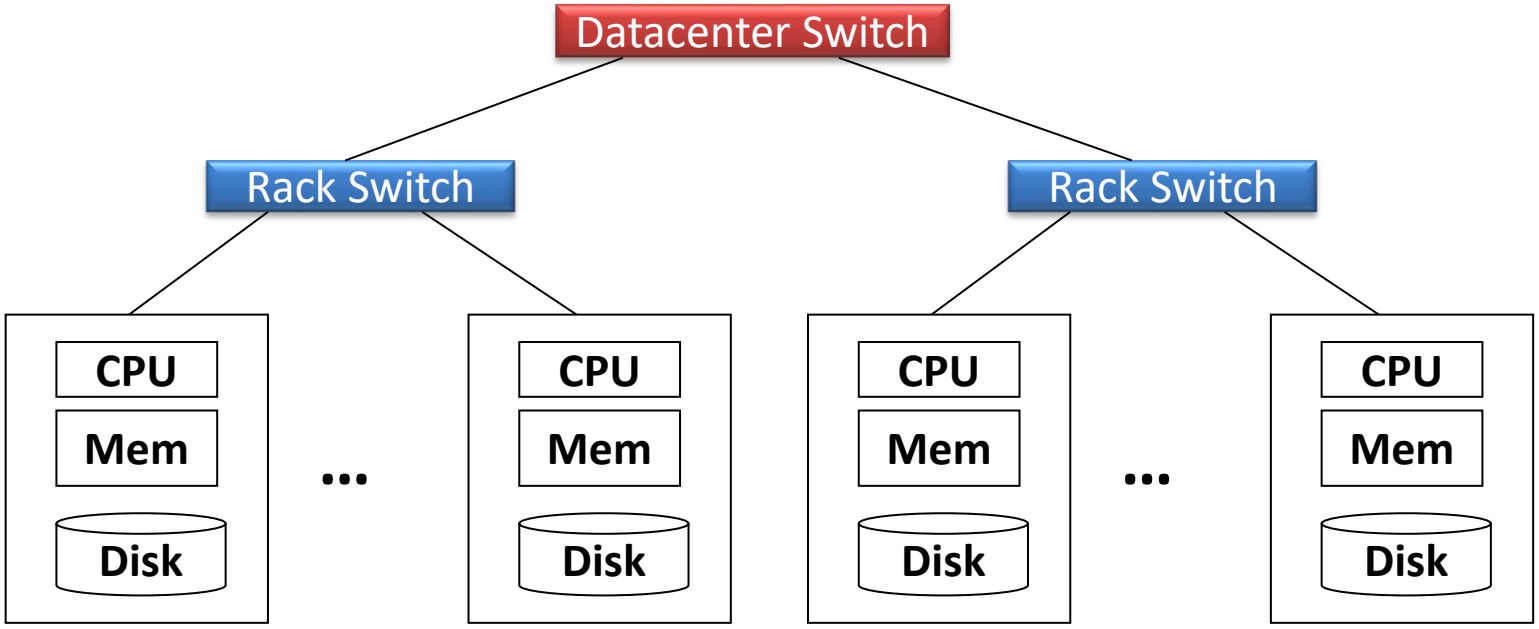
Distributed Graph Processing



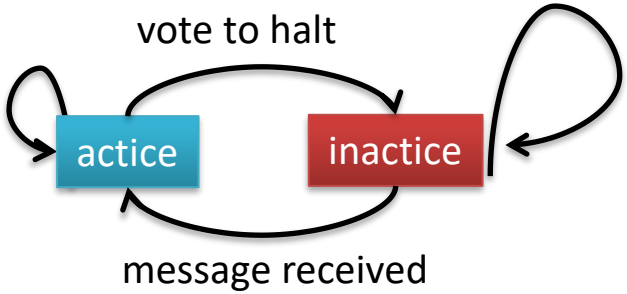
- Supper step: run user-defined code f()
- Synchronization: message exchange

Pregel

- Introduced by Google in their 2010 paper "*Pregel: A System for Large-Scale Graph Processing.*"
  - Google did not open-source Pregel but several open-source systems reimplemented the model (Apache Giraph, Apache Hama, etc)
- Designed to run on Google's cluster architecture
  - Each cluster consists of thousands of commodity PCs organized into racks with high intra-rack bandwidth

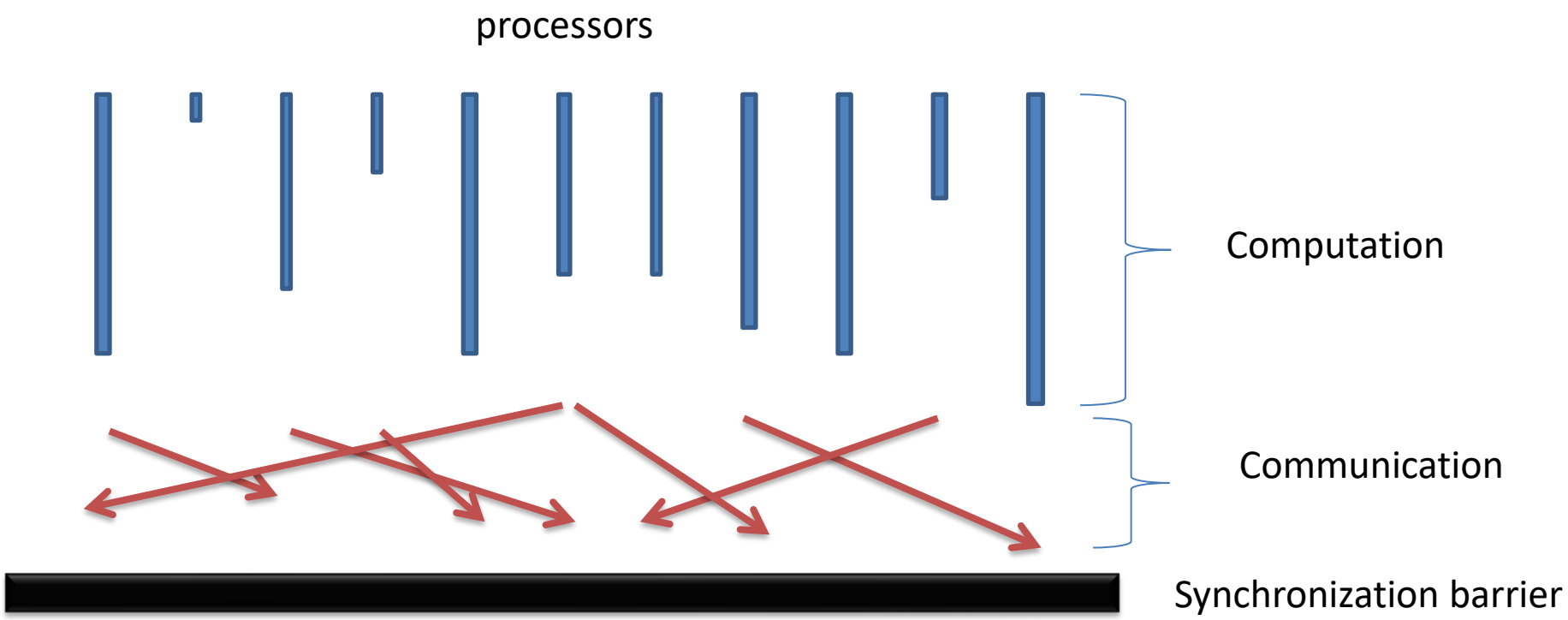


Pregel Superstep Model



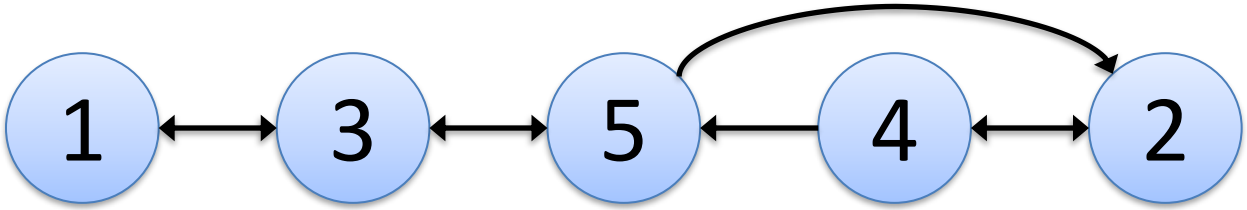
- Parallel Vertex Execution
  - Every vertex runs compute() simultaneously
  - Processes all messages from the previous superstep
- Inside compute() a vertex may
  - Update state (its own value or outgoing edges)
  - Send messages to other vertices
  - Modify topology (add/remove vertices or edges)
  - Vote to halt when it has no more work
- The Pregel job completes when
  - All vertices are inactive and no messages are in transit

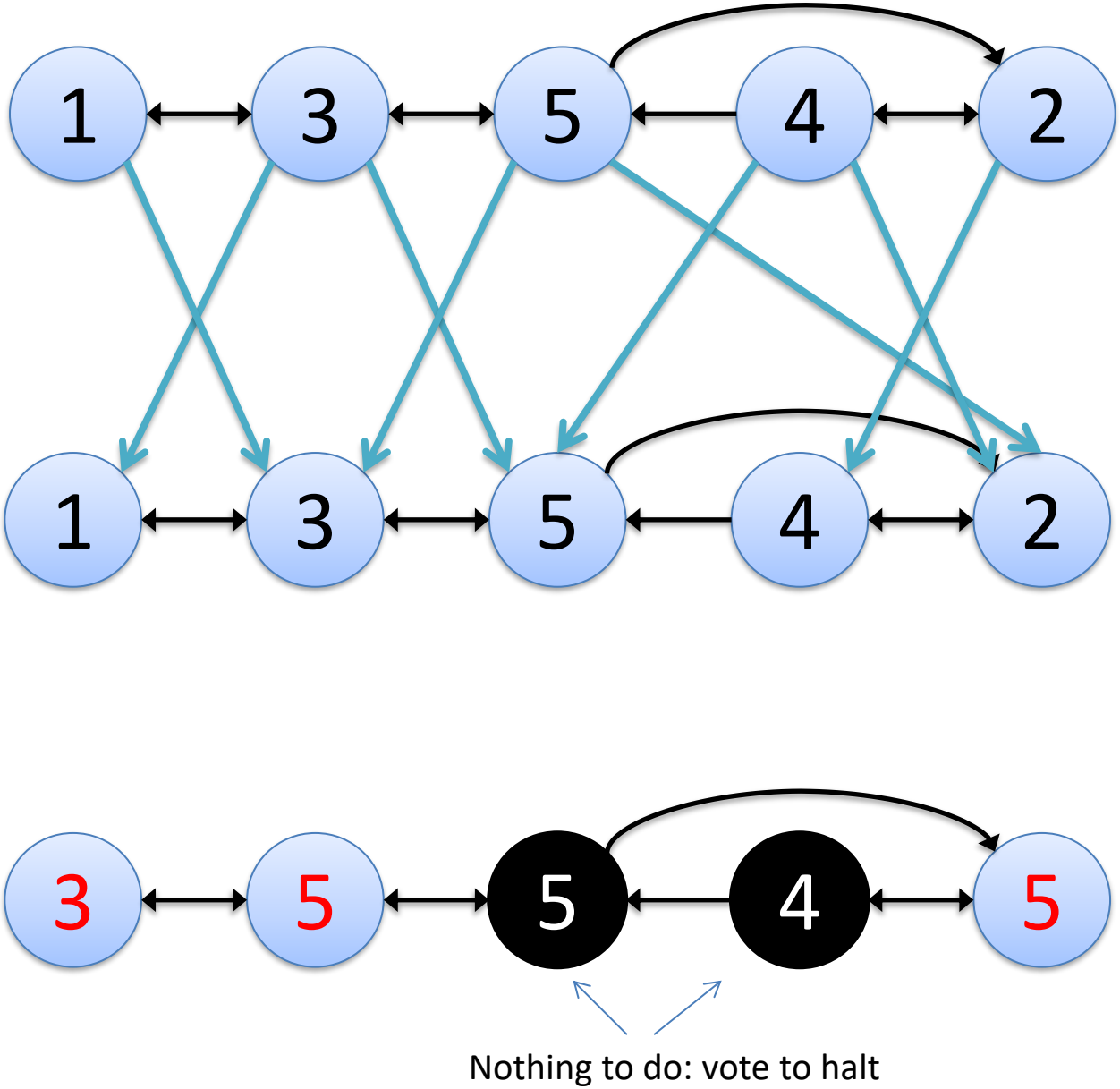
Bulk Synchronous Parallel Computing (Leslie Gabriel Valiant)



Toy problem

- Find the maximum value in a strongly connected graph component
  - Strongly connected: there is a directed path between any two vertices u, v

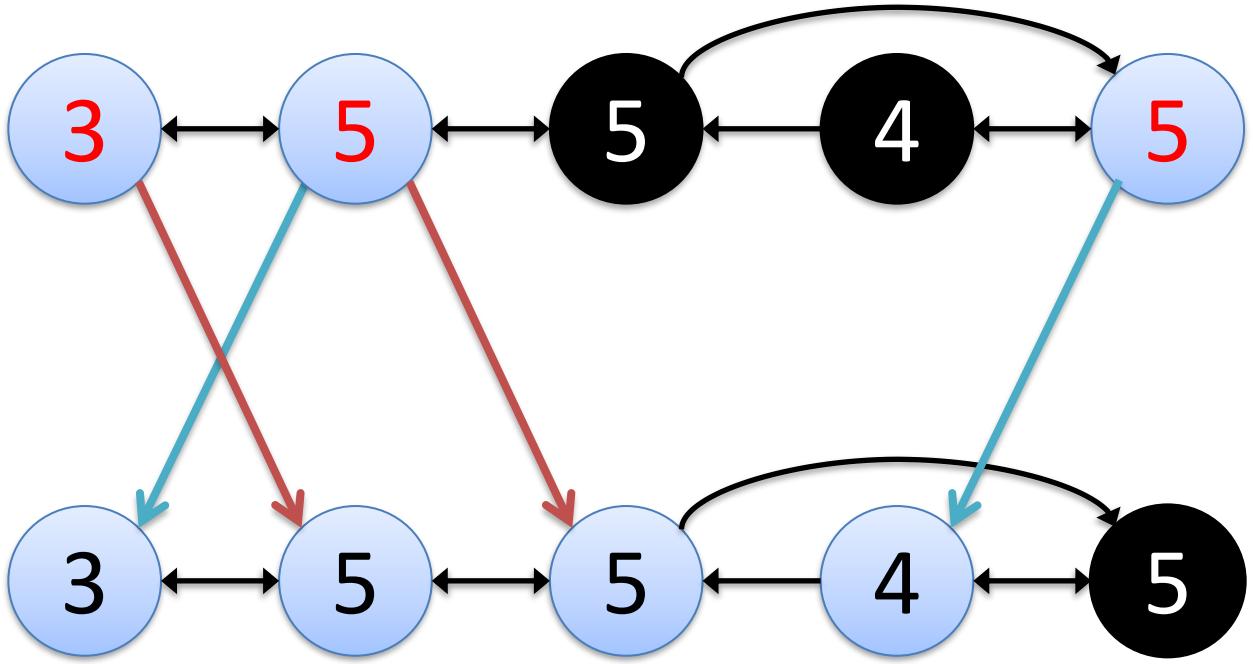




Super Step 1:
Transmit weights
to neighbors

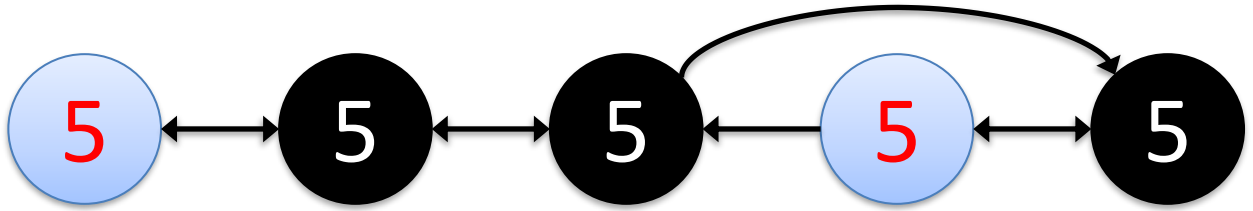
Start of
Super Step 2:
Read messages

Super Step 2:
Update weights,
if necessary

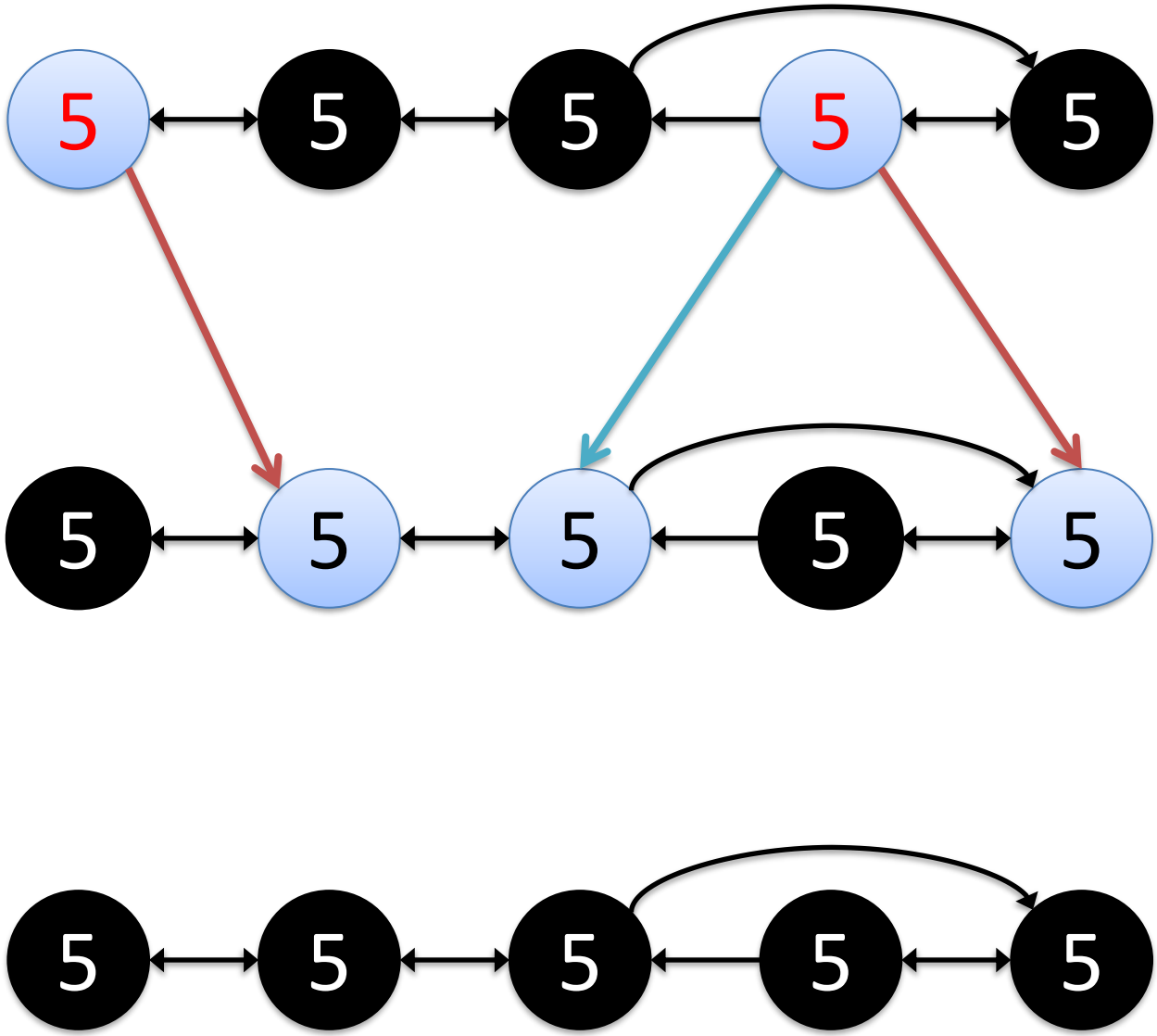


Super Step 2:
Transmit new weights, if necessary

Super Step 3:
Read messages

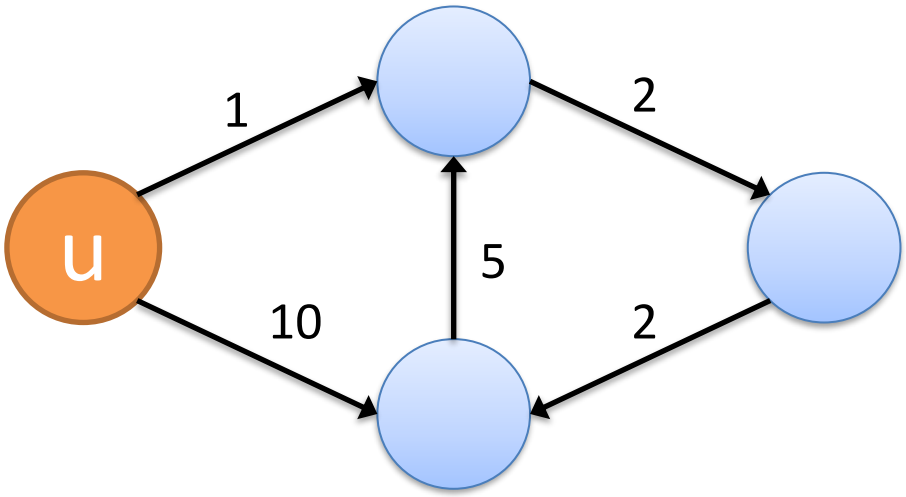


Super Step 3:
Update weights


5
5
5
5
5
5
5
5
5
5
5
5
5
5
5

Single Source Shortest Path

- Find shortest path from a source node u to all nodes
- Solution
  - Single CPU machine: Dijkstra's algorithm

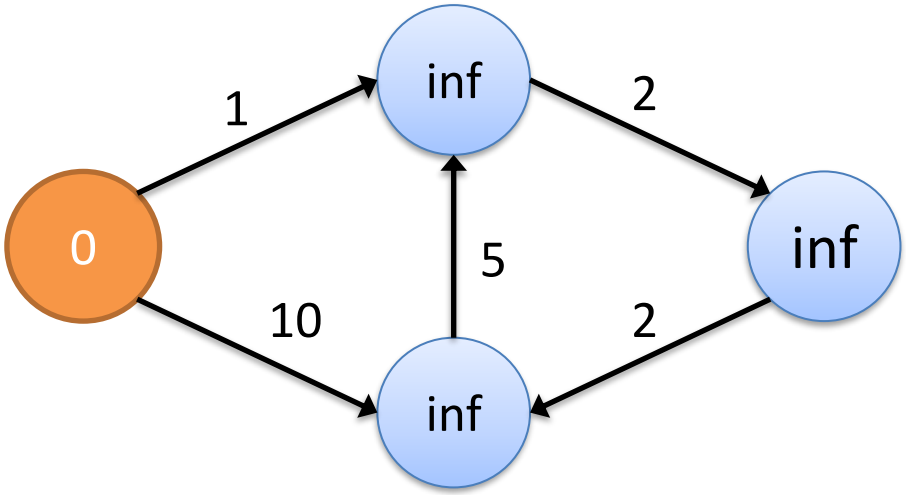


Dijkstra's algorithm Overview

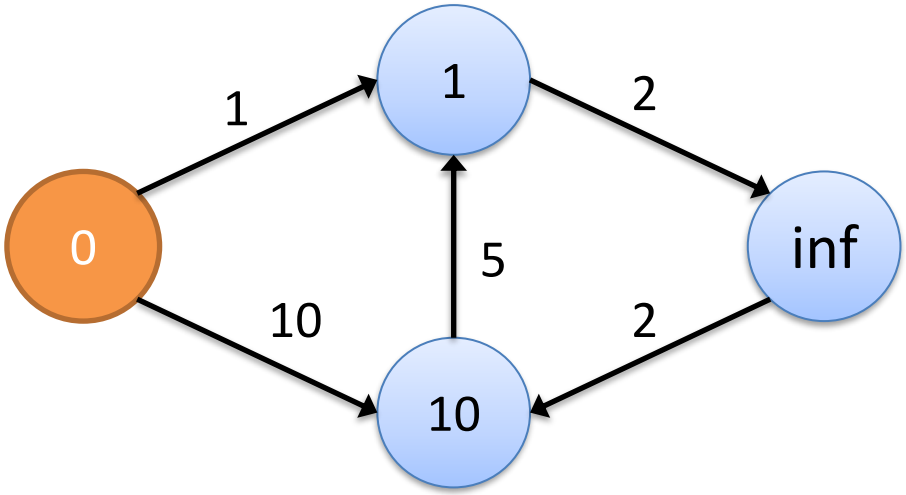
- Maintain distances of nodes from source (initially infinite, except source) in a priority queue
- At each step
  - Remove from queue node with minimum distance
  - Update shortest paths of adjacent nodes

Example: initialize queue

Q={0,inf,inf,inf}

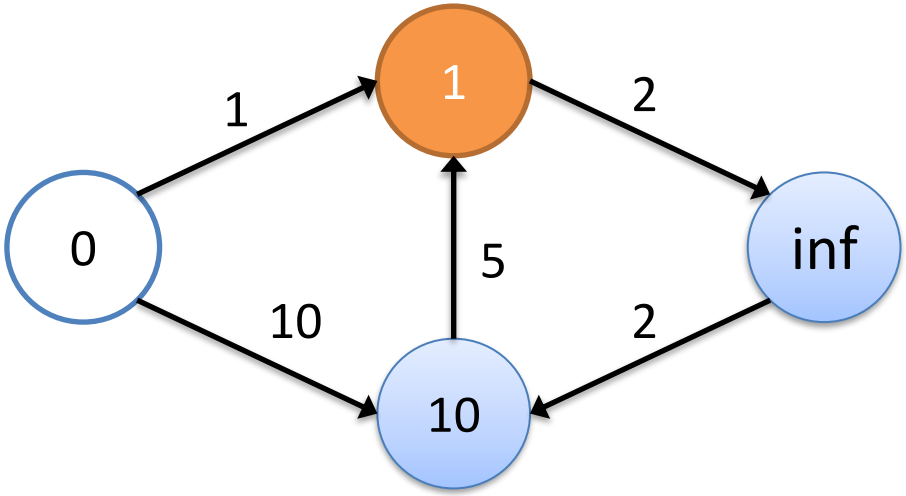


Update distances of adjacent nodes

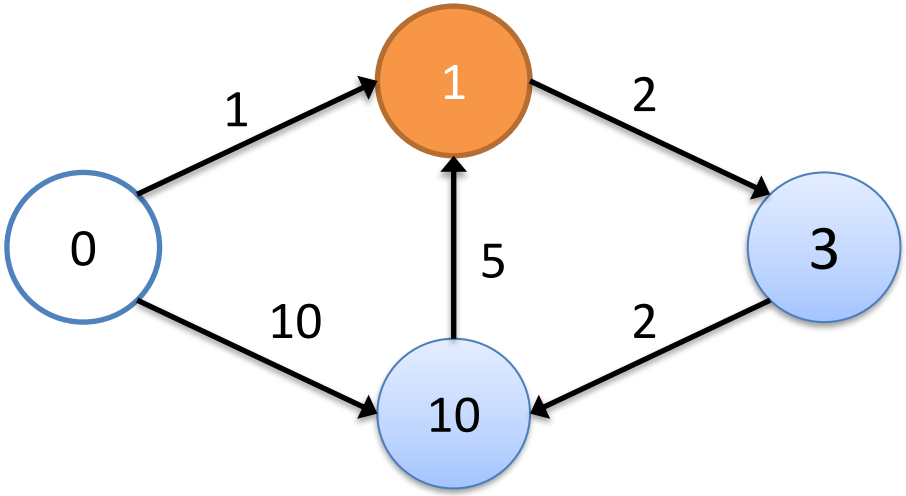


Pop next node from queue

Q={1,10,inf}

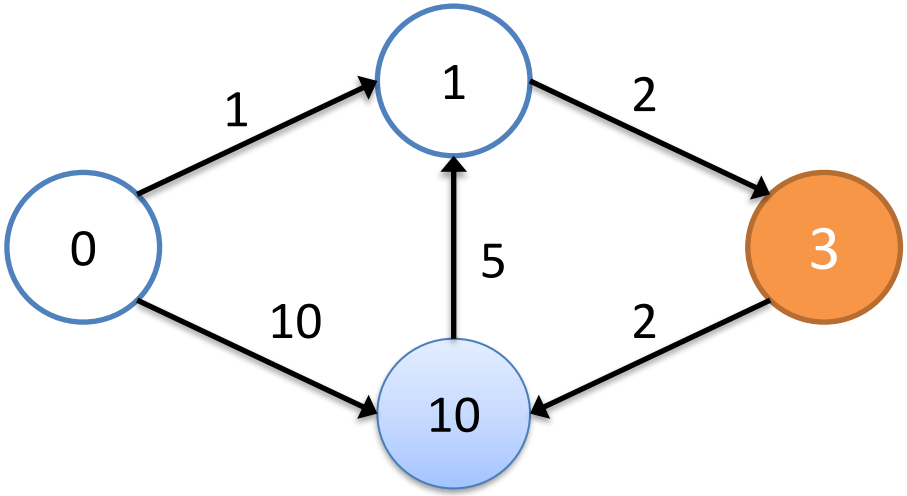


Update distances

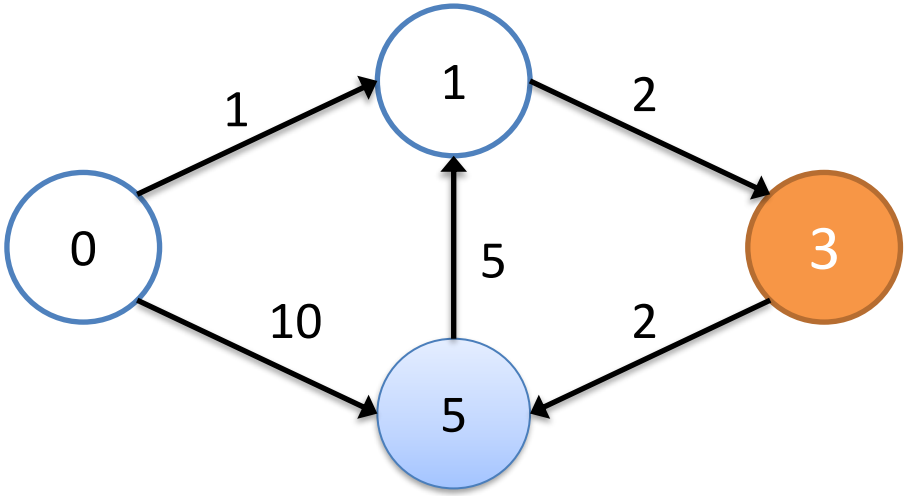


Pop next node from queue

Q={3,10}

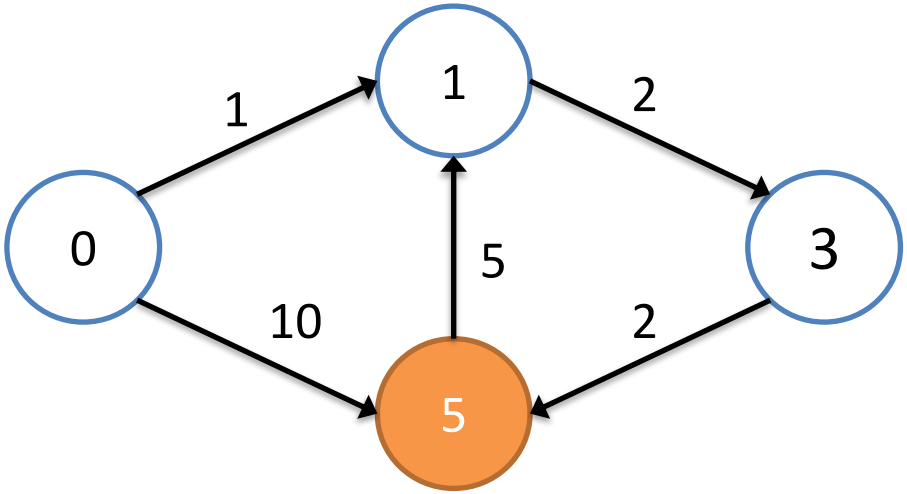


Update distances

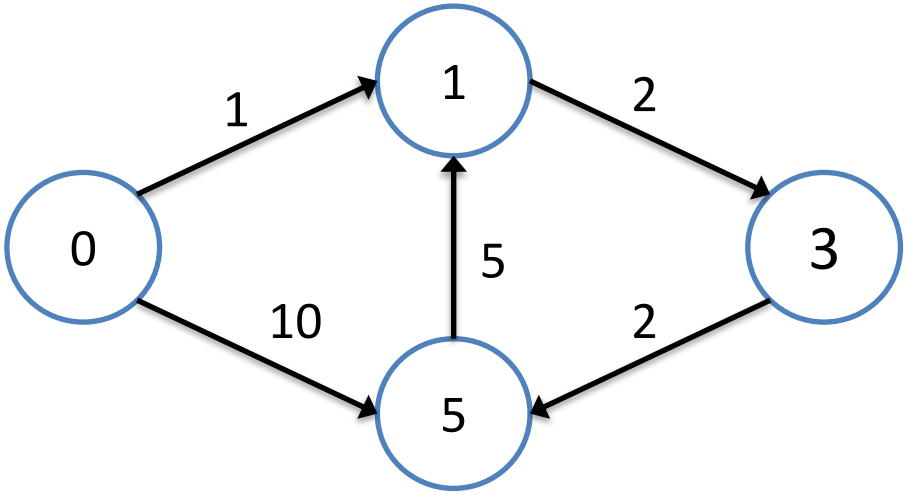


Pop last node, finished!

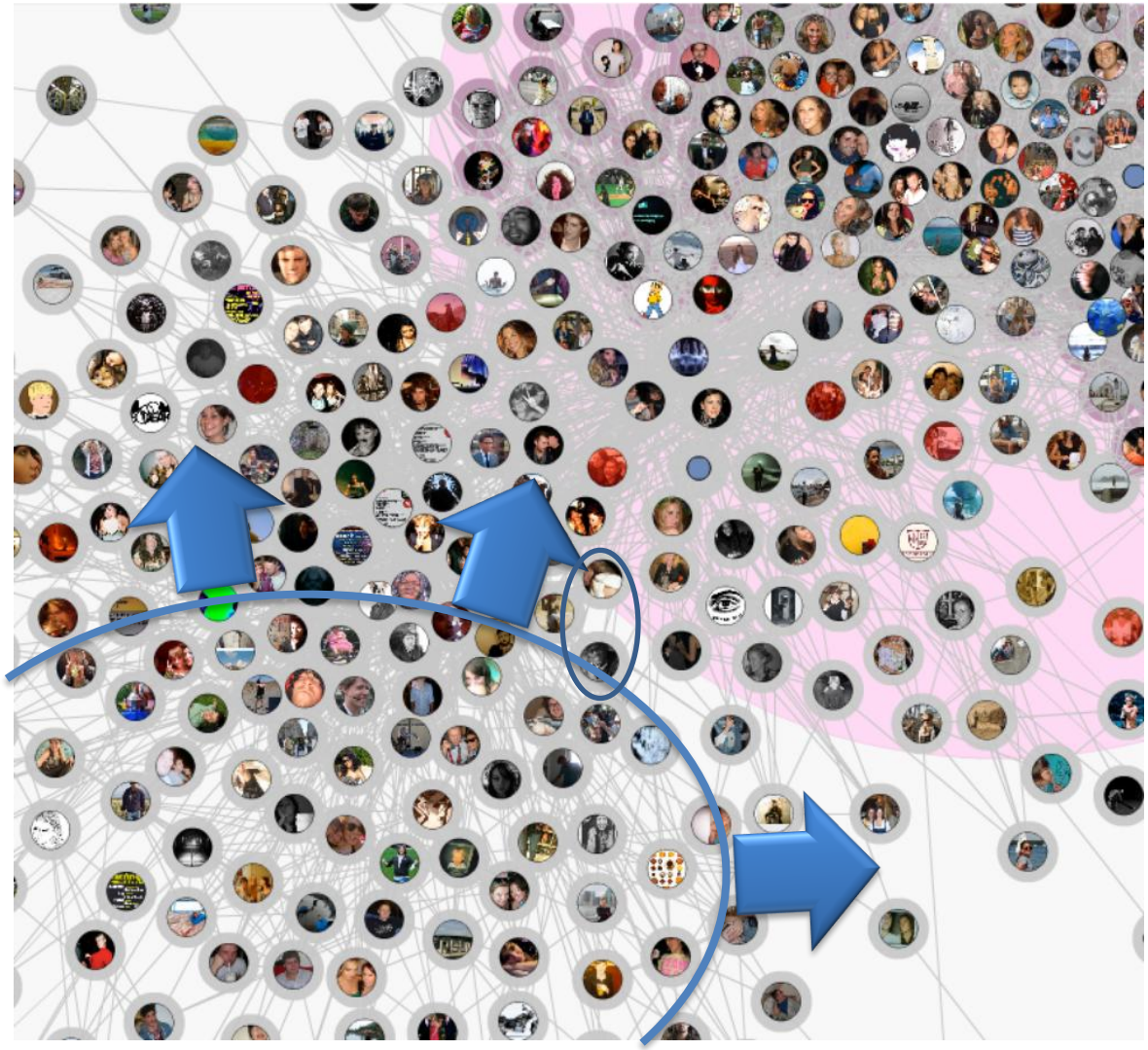
Q={5}



Computed distances

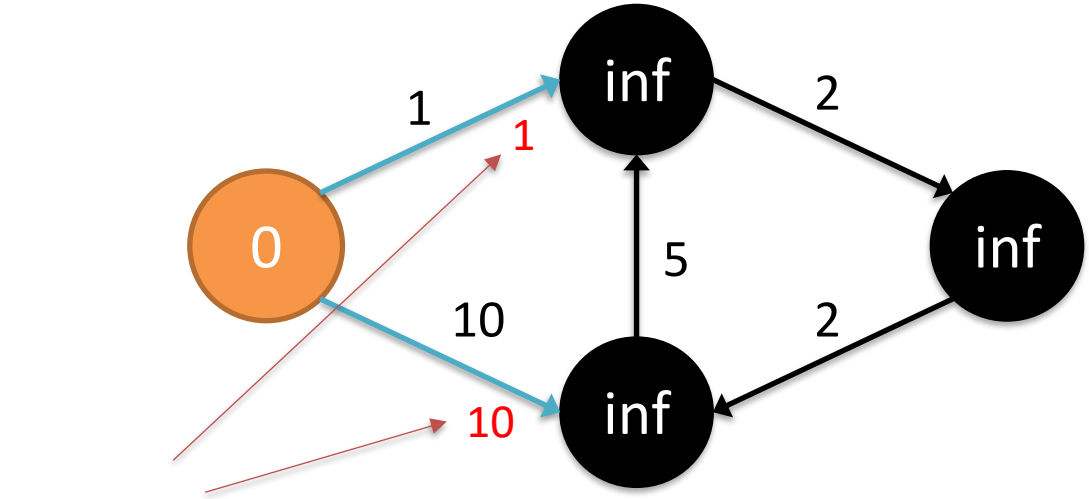


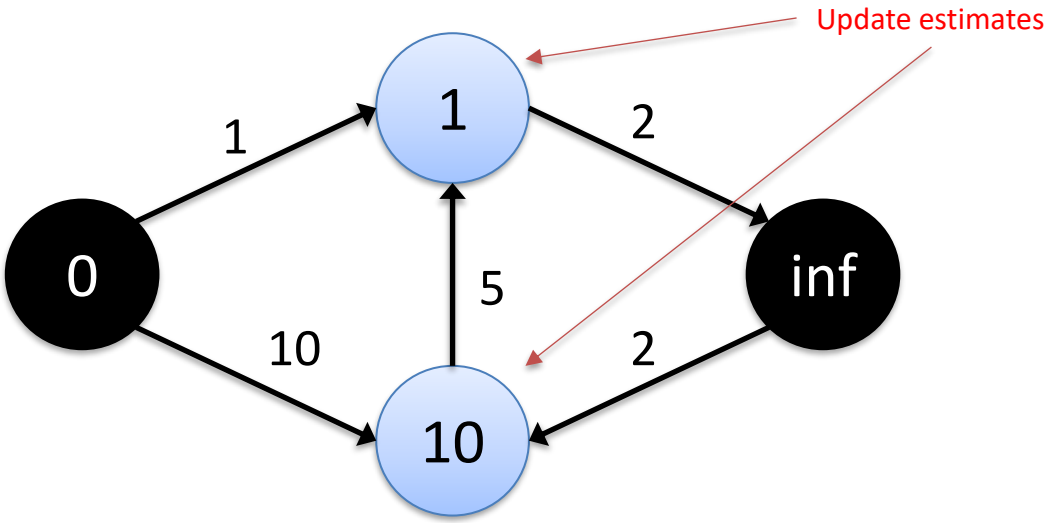
Dijkstra on a billion nodes graph

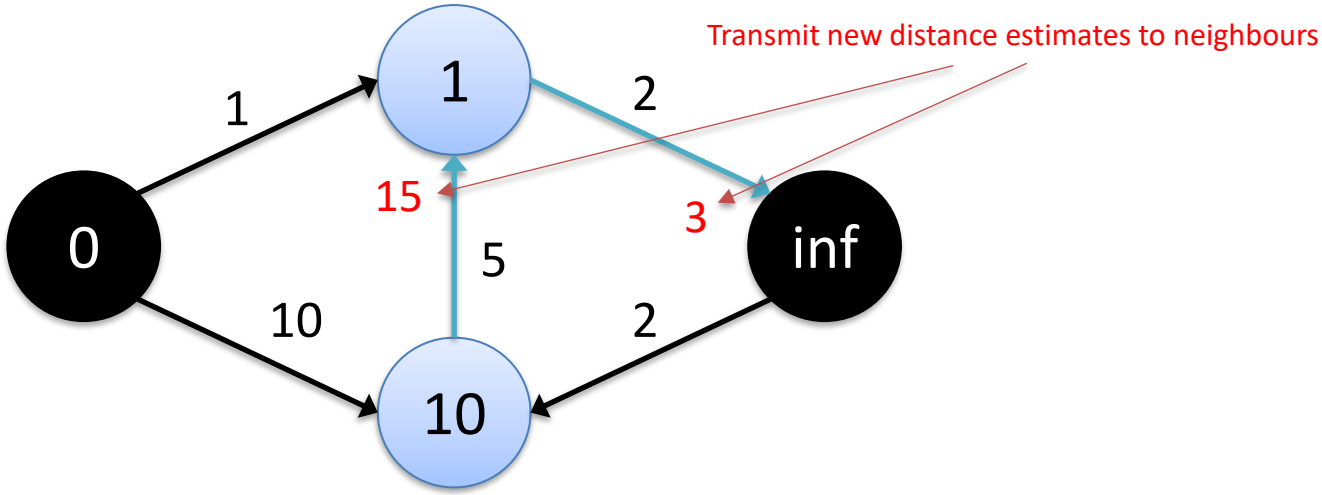


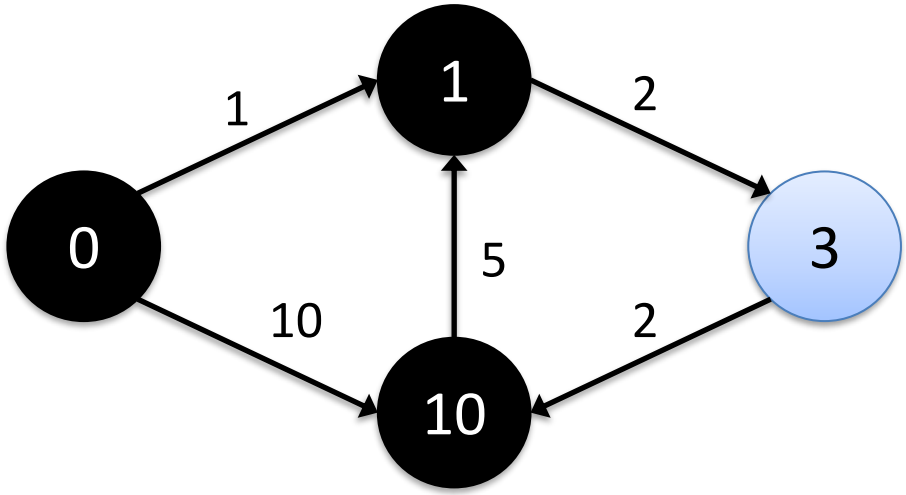
Parallel Breadth-First Search (PBFS)

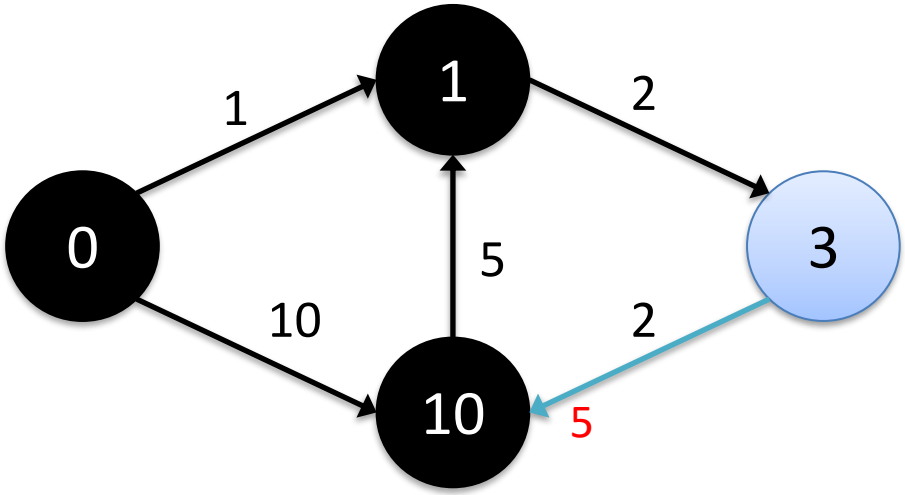
- Each node maintains current distance estimate
- Upon receive of a message from neighbors update estimate
  - If newly computed distance is shorter, inform neighbors

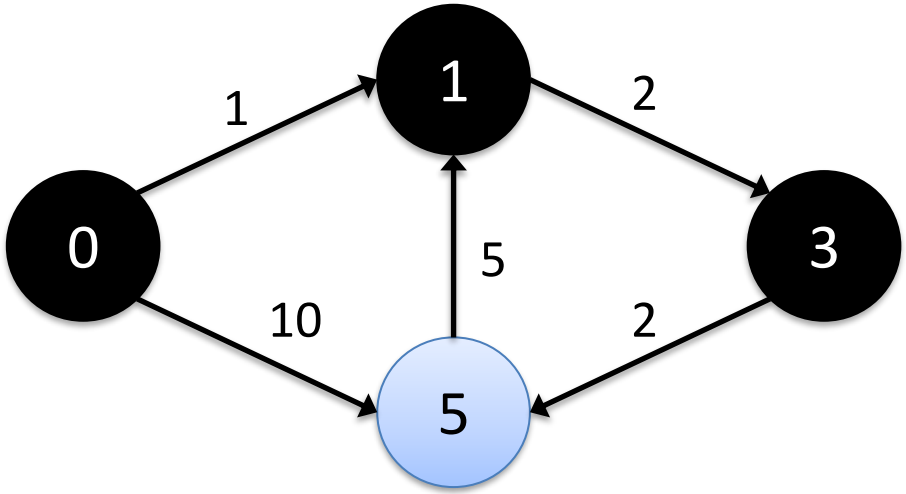

inf
1
2
1
0
inf
5
10
2
inf
10
Transmit distance estimates to neighbours

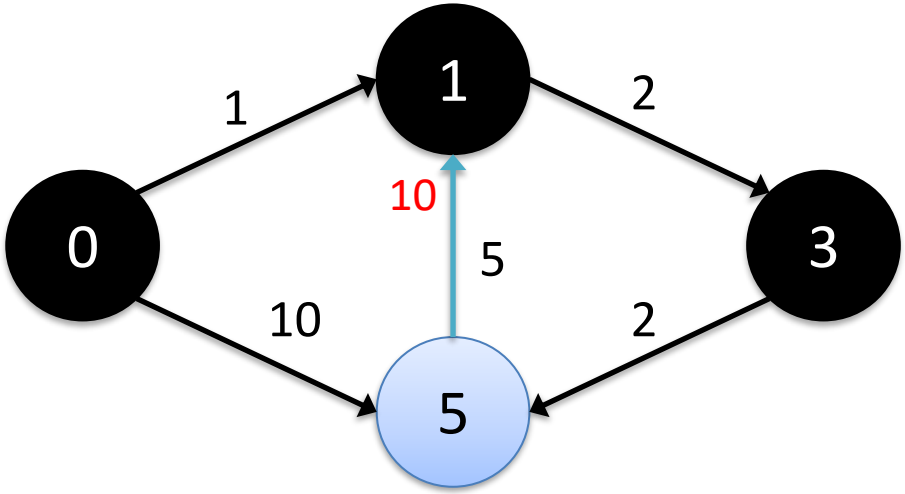

Update estimates
1
1
2
0
inf
5
10
2
10

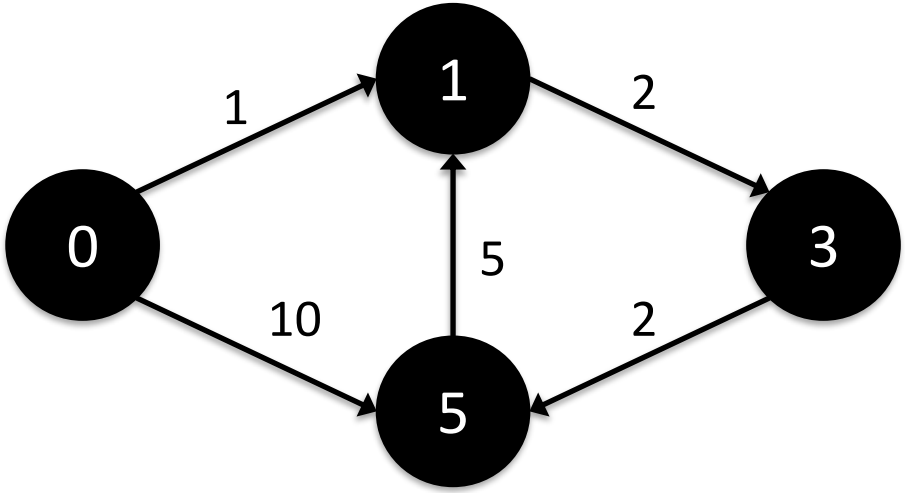

Transmit new distance estimates to neighbours
1
2
15
3
0
5
inf
1
10
2
10


1
1
2
0
5
3
10
2
10


1
2
0
3
5
10
2
10
5


1
1
2
0
3
5
10
2
5


1
2
10
0
5
3
10
2
5


1
2
0
5
3
10
2
5

PBFS vs Dijkstra

 PBFS: More (redundant) computations of distances until true shortest path is found

BUT

 Many parallel calculations per clock tick. No need of a global priority query, only local state maintained at each node

Shortest Path Code

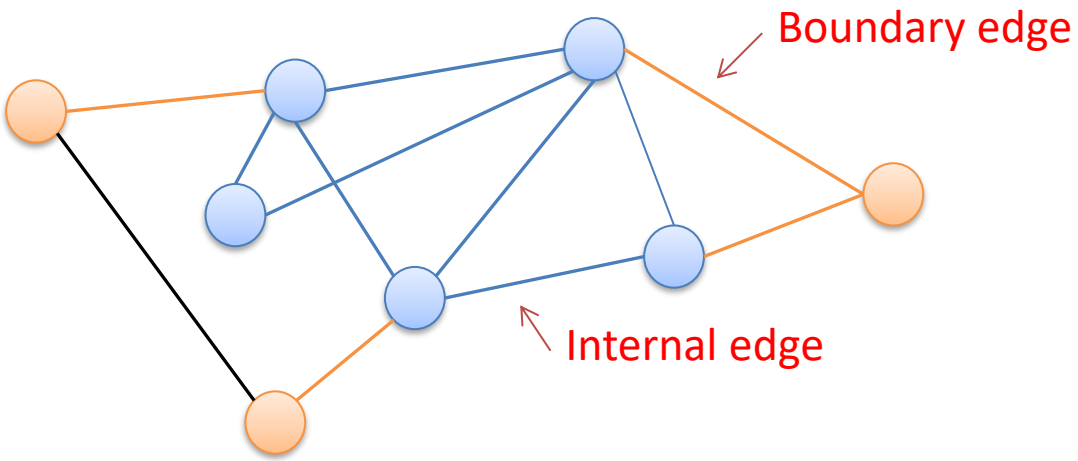
```
class ShortestPathVertex
    : public Vertex<int, int, int> {
  void Compute(MessageIterator* msgs) {
    int mindist = IsSource(vertex_id()) ? 0 : INF;
    for (; !msgs->Done(); msgs->Next())
      mindist = min(mindist, msgs->Value());
    if (mindist < GetValue()) {
      *MutableValue() = mindist;
      OutEdgeIterator iter = GetOutEdgeIterator();
      for (; !iter.Done(); iter.Next())
        SendMessageTo(iter.Target(),
                      mindist + iter.GetValue());
    }
    VoteToHalt();
  }
};
```

PageRank Code

```
class PageRankVertex
    : public Vertex<double, void, double> {
 public:
  virtual void Compute(MessageIterator* msgs) {
    if (superstep() >= 1) {
      double sum = 0;
      for (; !msgs->Done(); msgs->Next())
        sum += msgs->Value();
      *MutableValue() =
          0.15 / NumVertices() + 0.85 * sum;
    }

    if (superstep() < 30) {
      const int64 n = GetOutEdgeIterator().size();
      SendMessageToAllNeighbors(GetValue() / n);
    } else {
      VoteToHalt();
    }
  }
};
```

Semi-clustering in a social graph

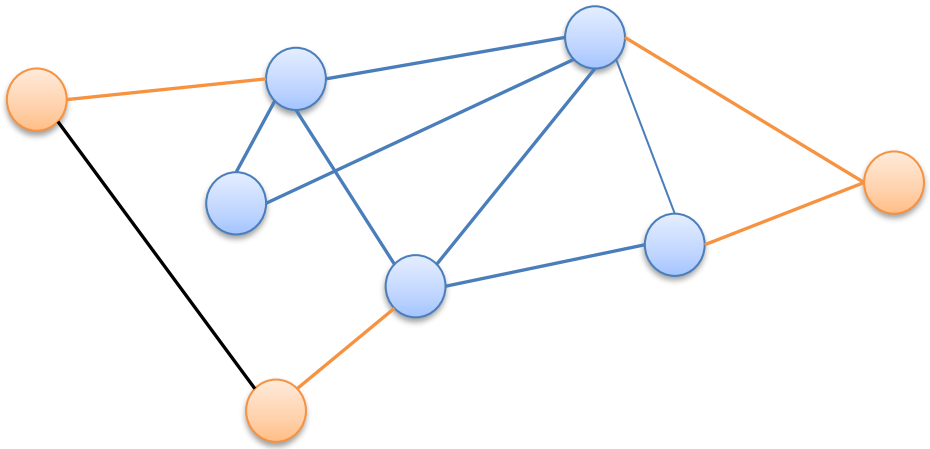
- A semi-cluster in a social graph is a group of people who interact frequently with each other and less frequently with others.
  - A person may belong to multiple semi-clusters



Evaluation of Semi-clusters

- I_c: sum of weights of internal edges
- B_c: sum of weights of boundary edges
- V_c: size of semi-cluster
- F_b: boundary edge score factor (0..1)

$$S_c = \frac{I_c - f_B B_c}{V_c(V_c - 1)/2}$$



$I_c = 7$ $B_c = 4$ $V_c = 5$

Computing Semi-clusters in Pregel

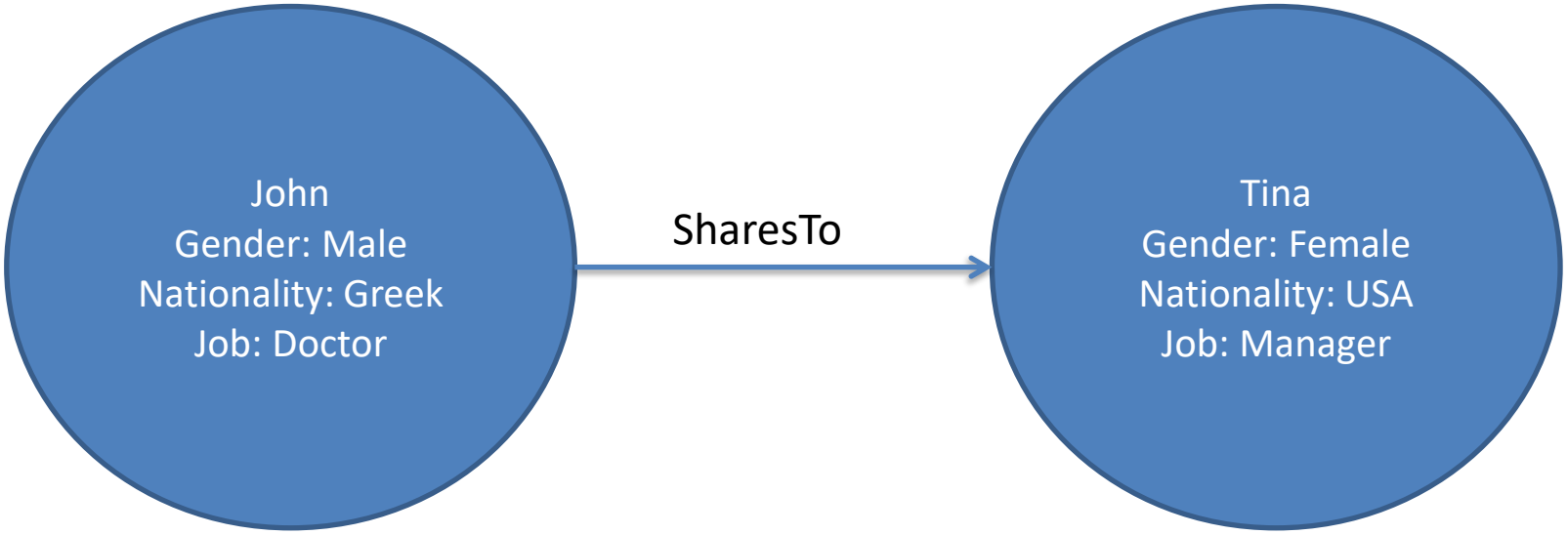
- Each vertex maintains a list containing at most C_{max} semi-clusters, sorted by score.
- In super-step 0 each node creates its own cluster and informs neighbors.
- In subsequent super-steps a vertex V iterates over the semi-clusters sent to it on the previous super-step.
  - If a semi-cluster does not already contain V and is not full then V is added to that cluster
  - The best k semi-clusters (sorted by their scores) are sent to neighbors
  - Node keeps a list of semi-clusters that contain V (itself)
- Stop if no new semi-clusters are formed of after a set of iterations

OLAP on Graph Datasets

BLENDING GRAPHS AND CUBES

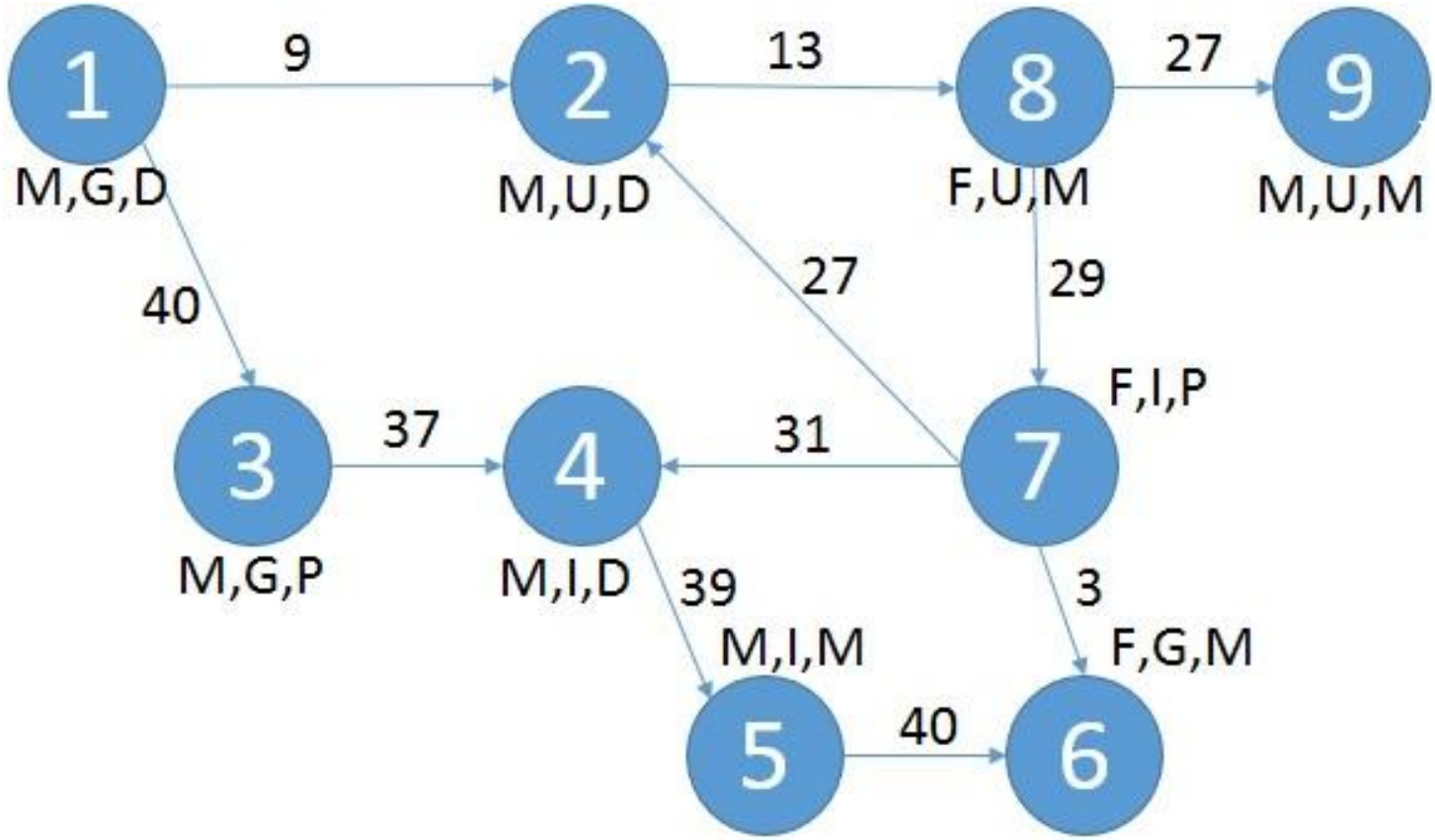
Interconnected Datasets

- Social network example
  - Users with three attributes (Gender, Nationality, Profession)

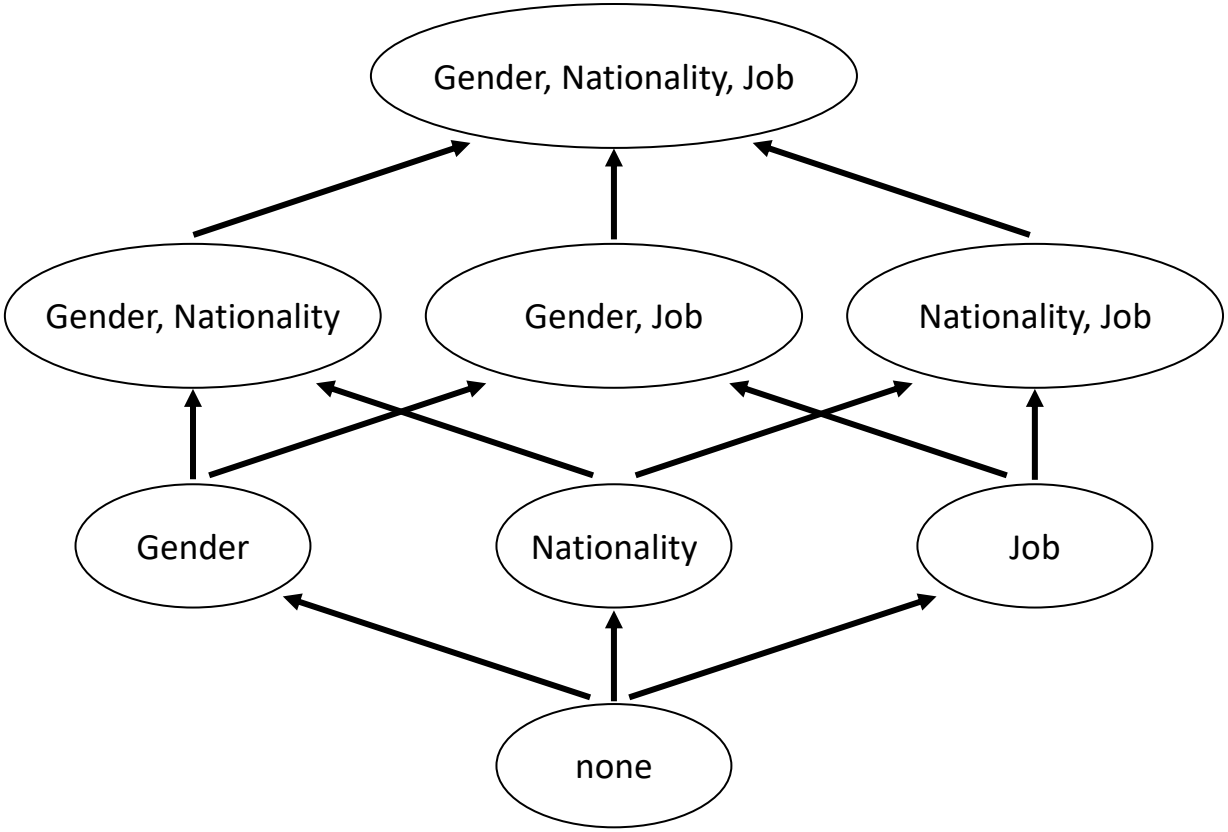


Toy Data

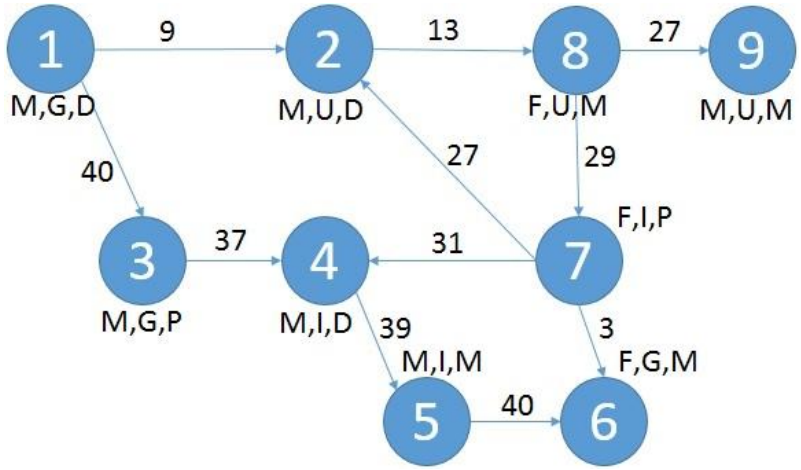
- Gender: **M**ale, **F**emale
- Nationality: **G**reek, **U**SA, **I**talian
- Job: **D**octor, **M**anager, **P**rofessor

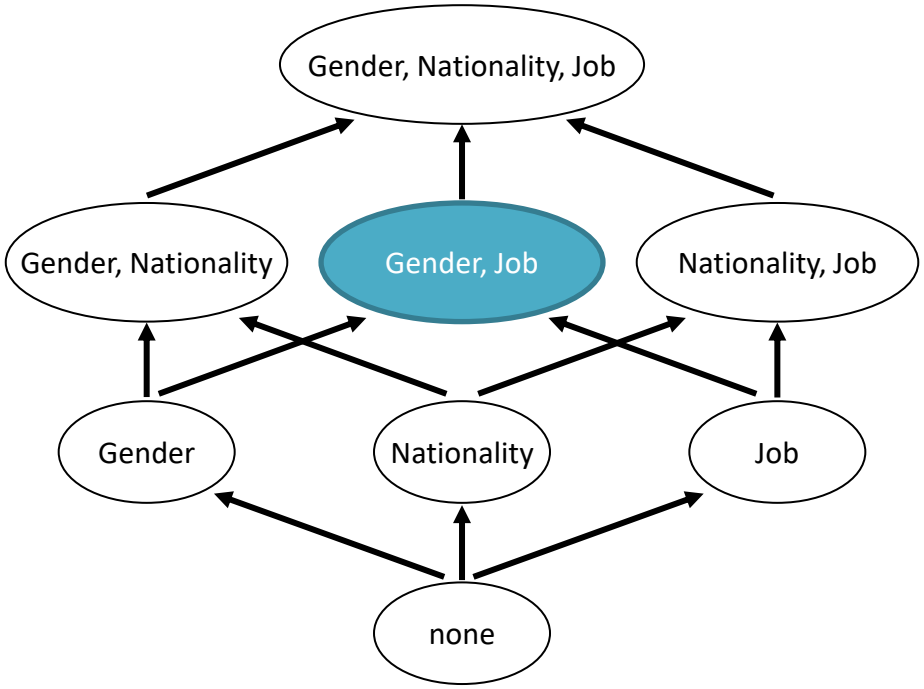


Data Cube on Nodes' Attributes



Data Cube Example



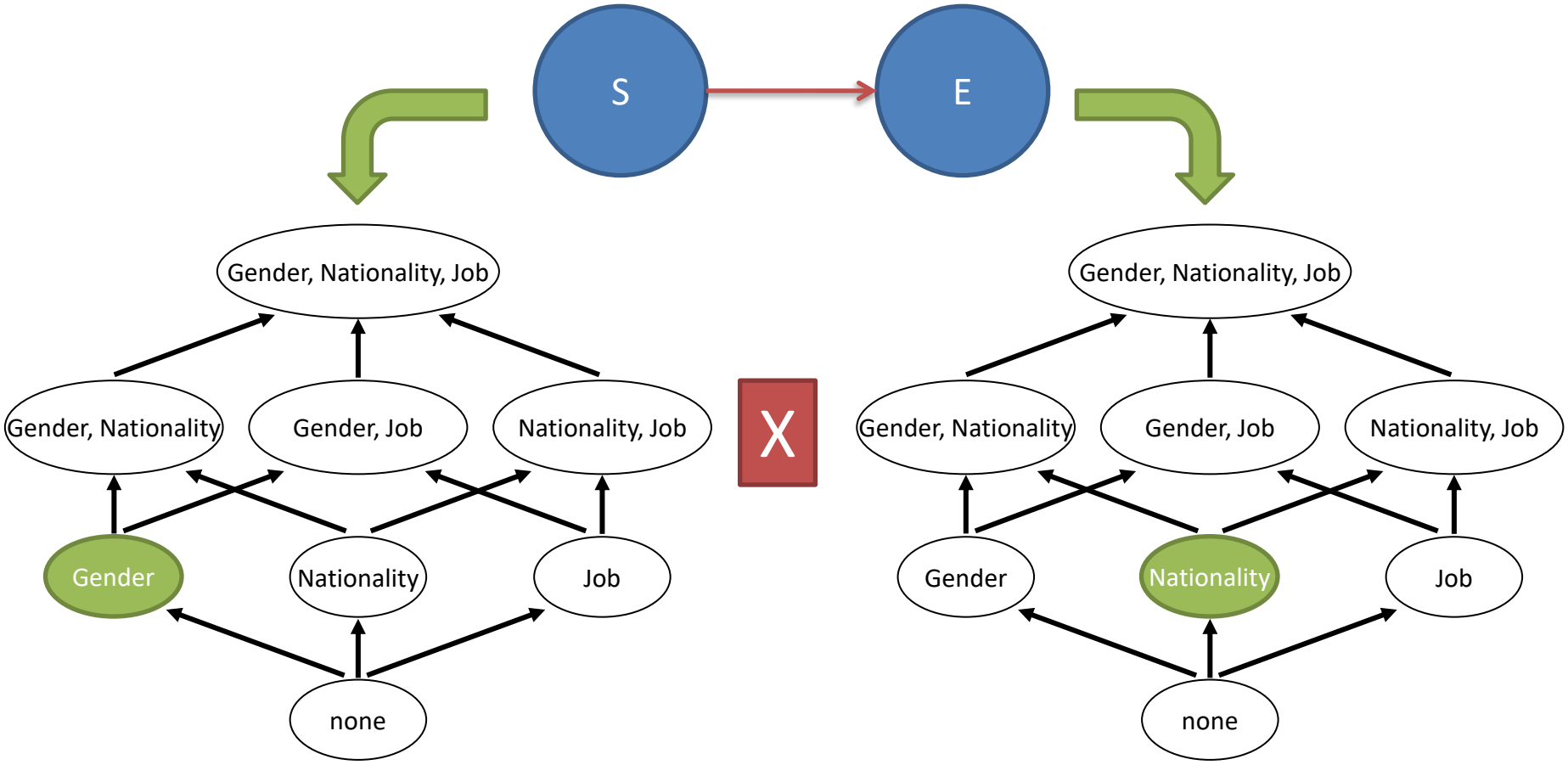


(Gender,Job) Cuboid

Gender	Job	Count
Male	Doctor	3
Male	Manager	2
Female	Manager	2
Male	Professor	1
Female	Professor	1

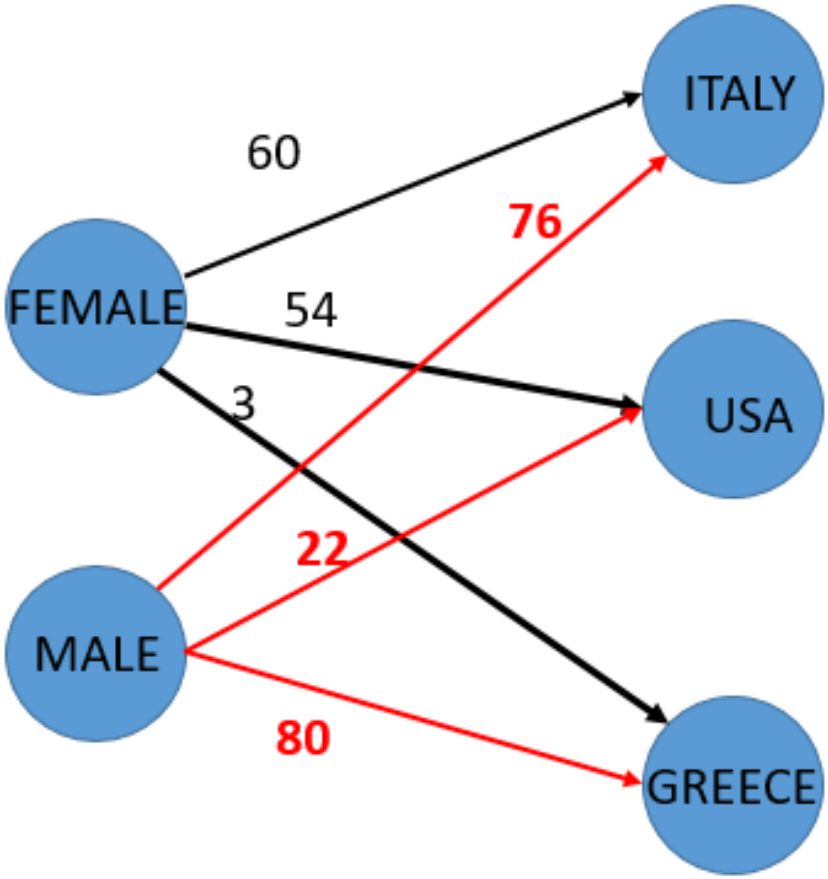
What is lost?

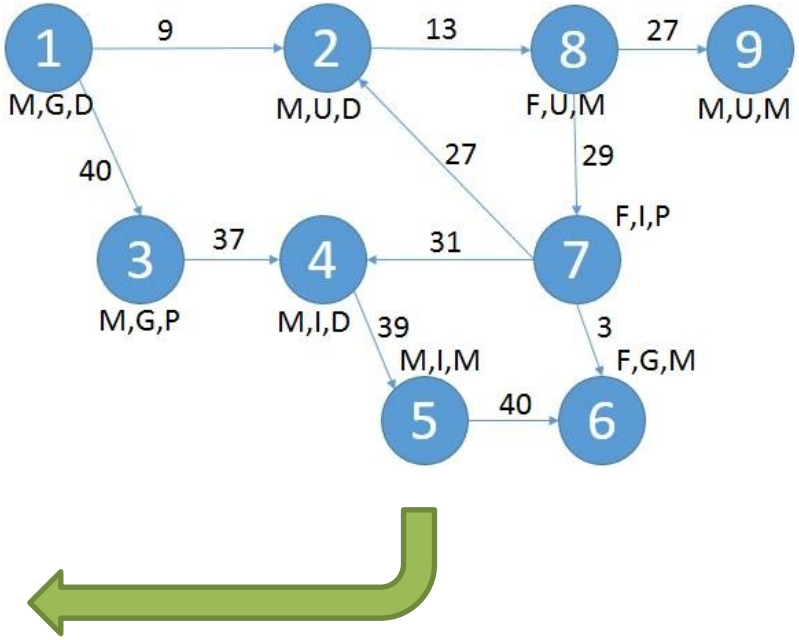
Graph Cube



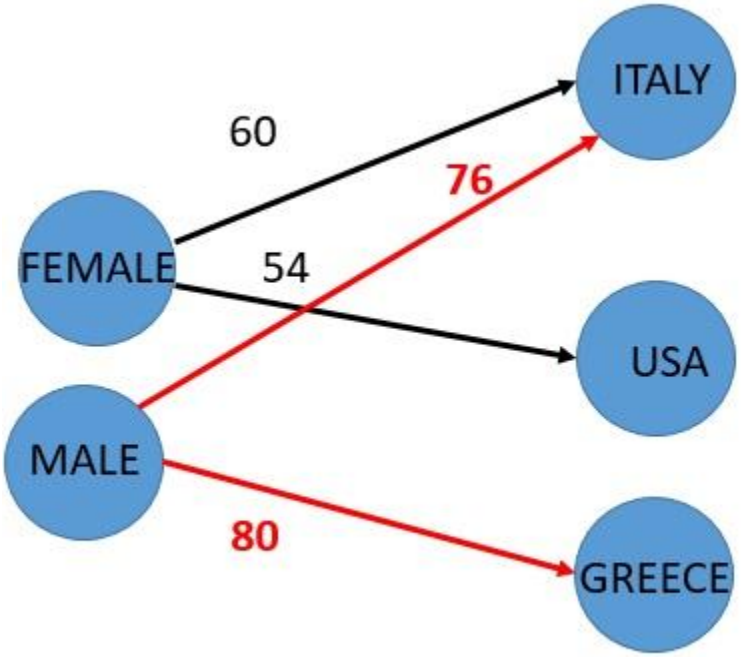
Example: (Gender) x (Nationality)

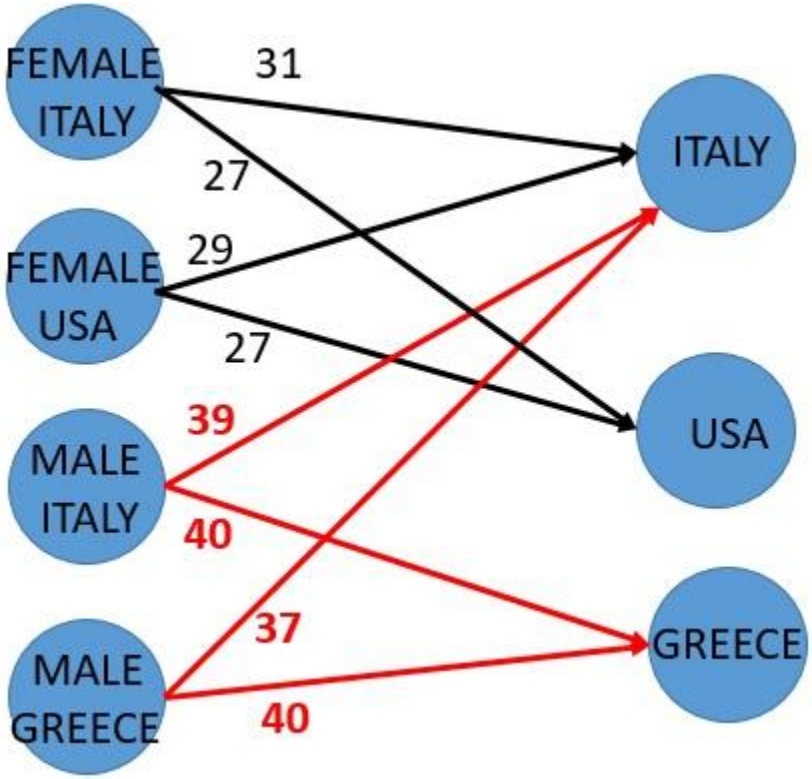
(Gender - Nationality) Cuboid





Drill-Down (original data)

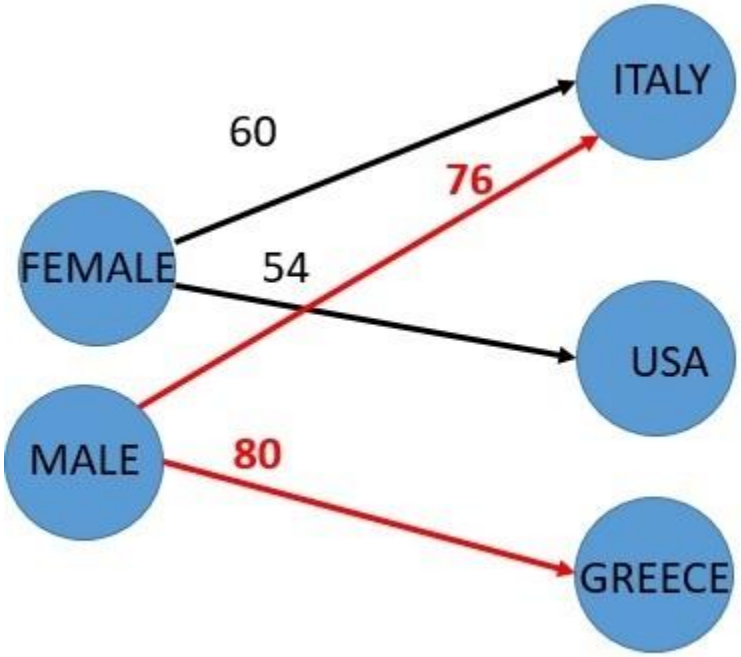


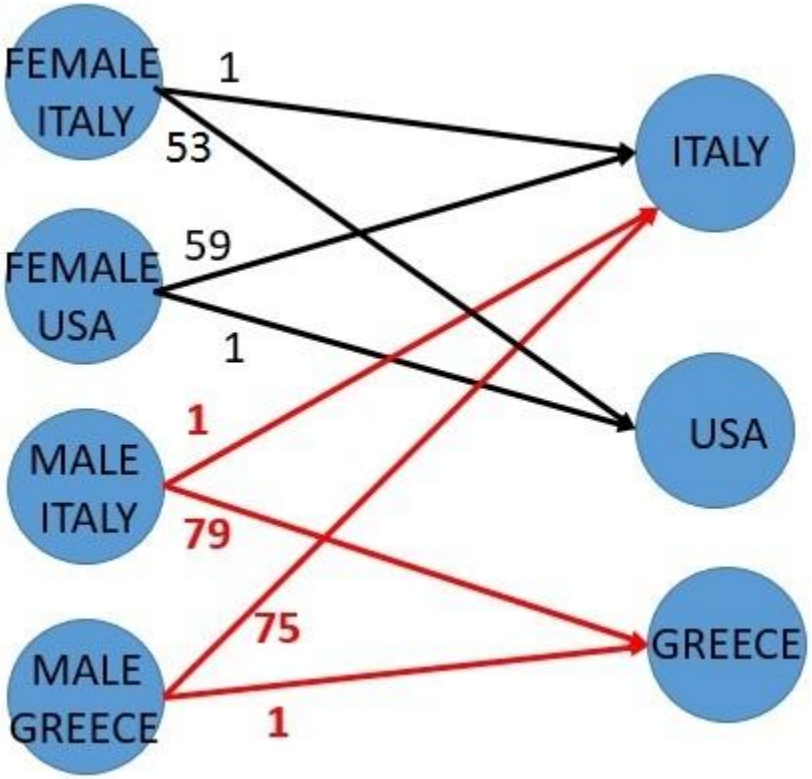


(Gender – Nationality)

(Gender, Nationality – Nationality)

Drill-Down (alternative data)





Bibliography (aueb+others)/Links

- Sergey Brin, Lawrence Page: The Anatomy of a Large-Scale Hypertextual Web Search Engine. Computer Networks 30(1-7): 107-117 (1998).
- Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. SIGMOD Conference 2010: 135-146.
- D. Bleco, Y. Kotidis. Graph Analytics on Massive Collections of Small Graphs. In Proceedings of the 17th International Conference on Extending Database Technology (EDBT), Athens, Greece, March, 2014.
- Peixiang Zhao, Xiaolei Li, Dong Xin, and Jiawei Han. Graph Cube: On Warehousing and OLAP MultidimensionalNetworks. In Proc. of 2011 ACM SIGMOD Int. Conf. on Management of Data, Athens, Greece, June 2011.
- D. Bleco, Y. Kotidis. Finding the Needle in a Haystack: Entropy Guided Exploration of Very Large Graph Cubes. In Proceedings of the International Workshop on Big Data Visual Exploration and Analytics (BigVis), Vienna, Austria, March 2018.
- I. Filippidou, Y. Kotidis. Effective and Efficient Graph Augmentation in Large Graphs. In Proceedings of the 2016 IEEE International Conference on Big Data (IEEE BigData 2016), Washington D.C., Dec 2016.
- V. Spyropoulos, Y. Kotidis. Digree: Building A Distributed Graph Processing Engine out of Single-node Graph Database Installations. ACM Sigmod Record, Volume 46(4), pages 22-27, December 2017.
- Y. Filippidou, Y. Kotidis. Online and On-demand Partitioning of Streaming Graphs. In Proceedings of the 2015 IEEE International Conference on Big Data (IEEE BigData 2015), Santa Clara, CA, Oct-Nov, 2015.
- http://giraph.apache.org/
- http://neo4j.com/
- http://www.sparsity-technologies.com/