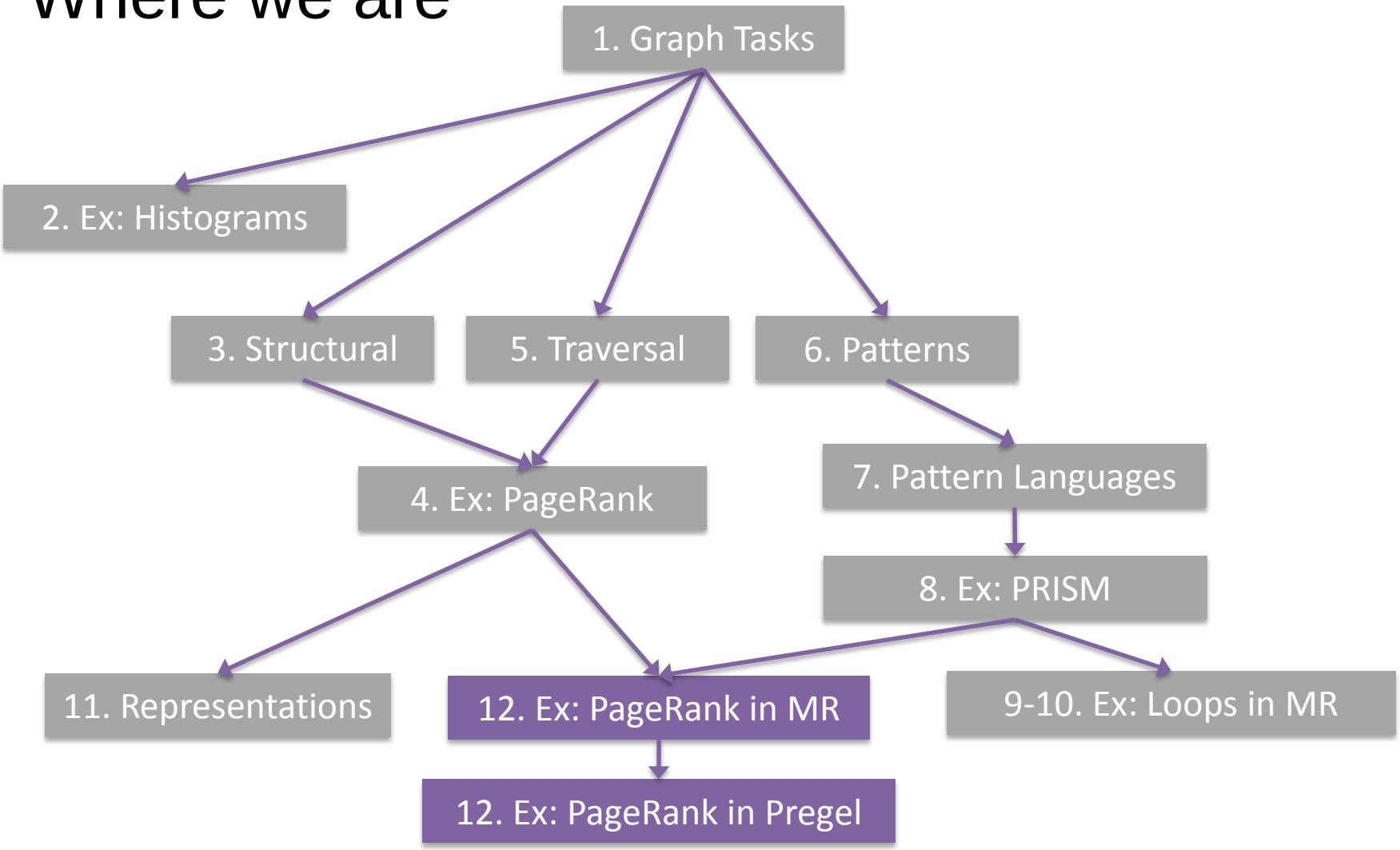
Where we are

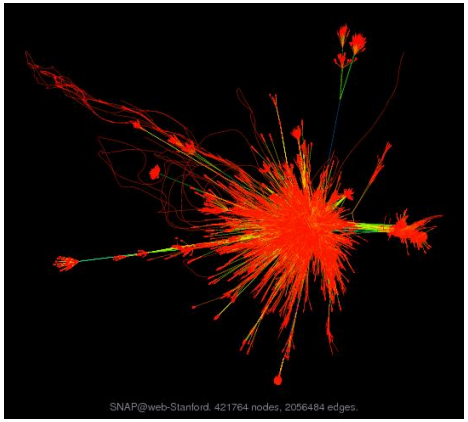

1. Graph Tasks
2. Ex: Histograms
3. Structural
5. Traversal
6. Patterns
4. Ex: PageRank
7. Pattern Languages
8. Ex: PRISM
11. Representations
12. Ex: PageRank in MR
9-10. Ex: Loops in MR
12. Ex: PageRank in Pregel

Big Graphs

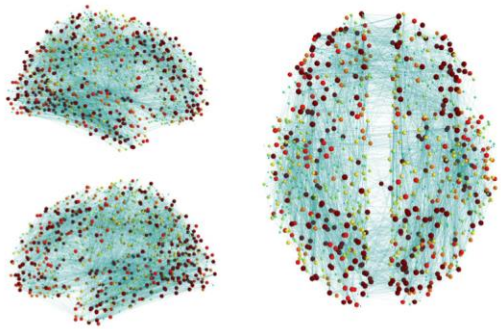
- Social scale
  - 1 billion vertices, 100 billion edges
- Web scale
  - 50 billion vertices, 1 trillion edges
- Brain scale
  - 100 billion vertices, 100 trillion edges

Paul Butler, Facebook, 2010

https://www.facebook.com/notes/facebook-engineering/visualizing-friendships/469716398919



Web graph from the SNAP database (http://snap.stanford.edu/data)



Gerhard et al, frontiers in neuroinformatics, 2011

material adapted from Paul Burkhardt, Chris Waring

MapReduce for PageRank

```
class Mapper
    method Map(id n, vertex N)
    p ← N.PAGERANK/|N.ADJACENCYLIST|
    EMIT(id n, vertex N)
    for all nodeid m in N.ADJACENCYLIST do
        EMIT(id m, value p)

class Reducer
    method REDUCE(id m, [p1, p2, …])
        M ← null, s ← 0
        for all p in [p1, p2, …] do
            if ISVERTEX(p) then
                M ← p
            else
                s ← s + p
        M.PAGERANK ← s * 0.85 + 0.15 / TOTALVERTICES
        EMIT(id m, vertex M)
```

Problems

- The entire state of the graph is shuffled on every iteration
- We only need to shuffle the new rank contributions, not the graph structure
- Further, we have to control the iteration outside of MapReduce

Malewicz et al. SIGMOD 10

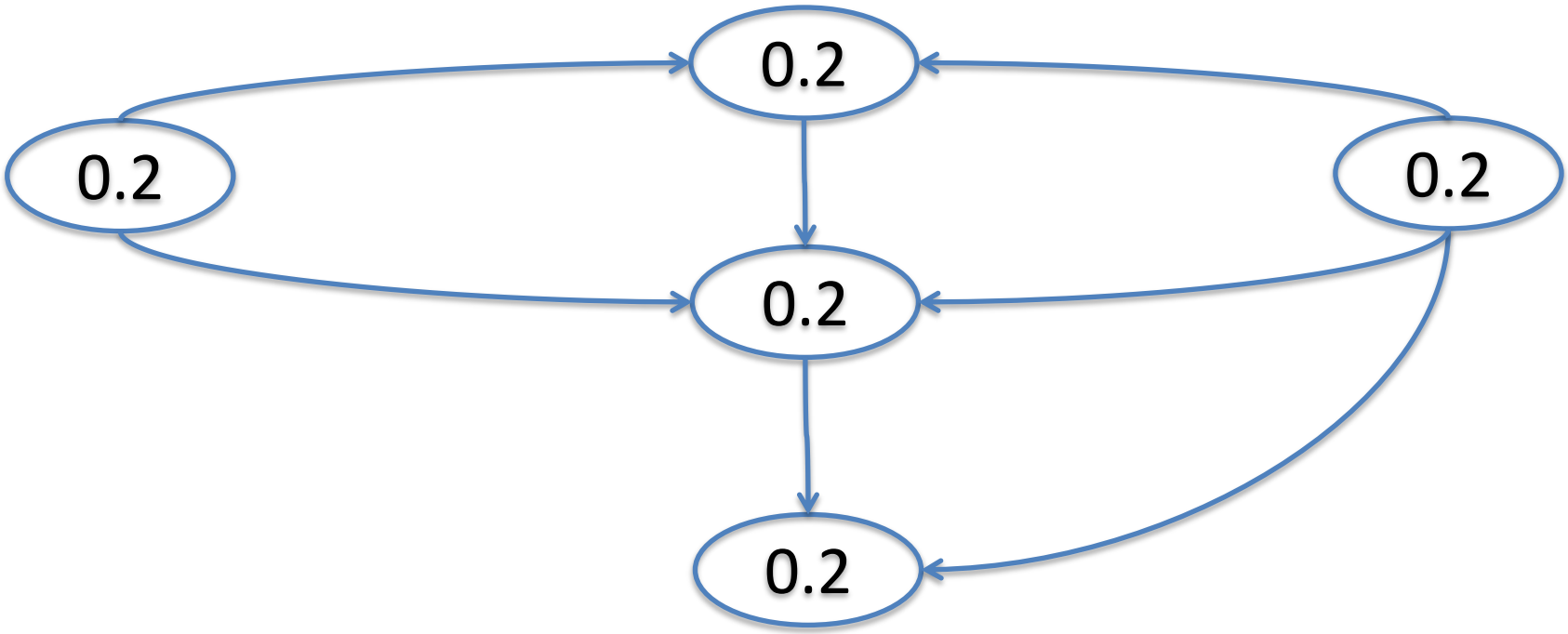
Pregel

- Originally from Google
  - Open source implementations
  - Apache Giraph, Stanford GPS, Jpregel, Hama
- Batch algorithms on large graphs

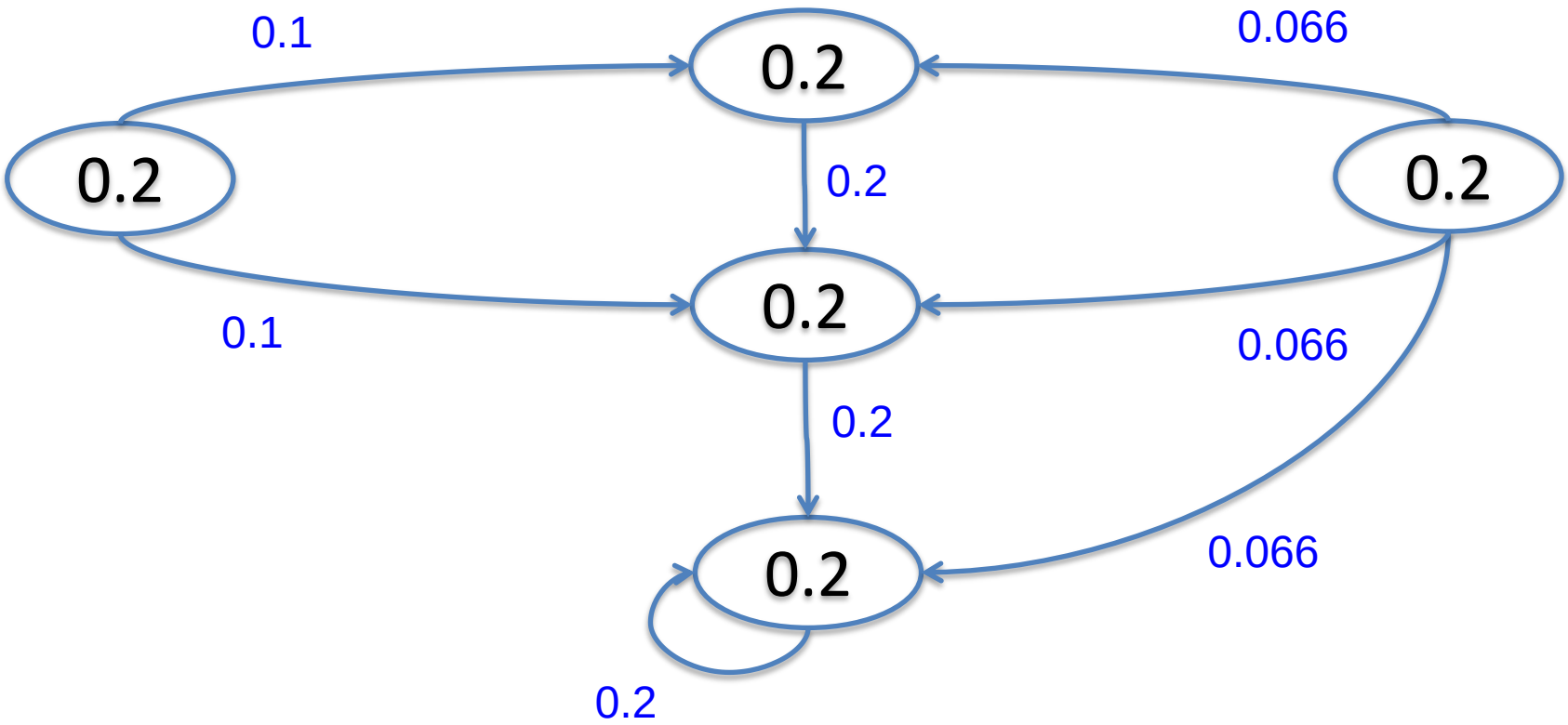
```
while any vertex is active or max iterations not reached:
    for each vertex:
        process messages from neighbors from previous iteration
        send messages to neighbors
        set active flag appropriately
```

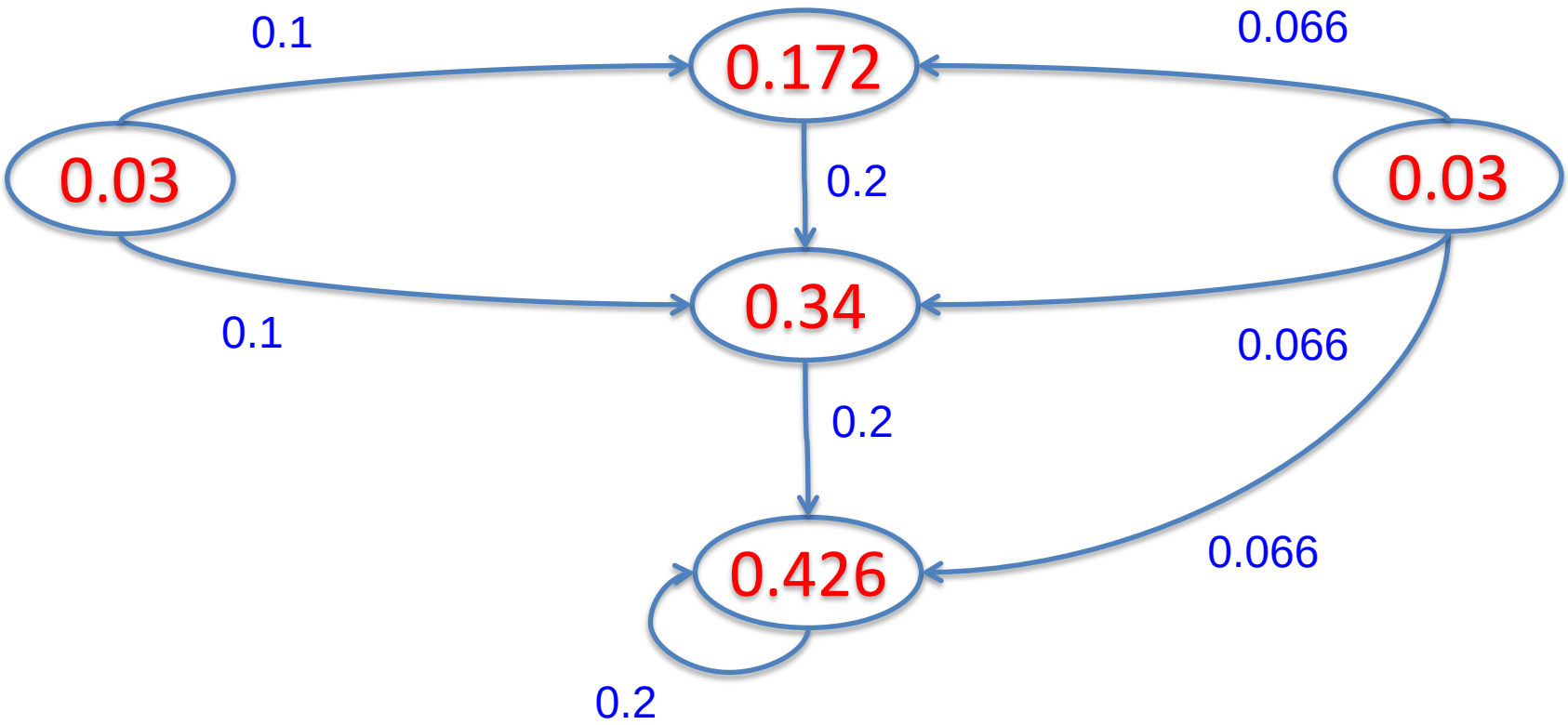
this loop is run in parallel

```
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();
        }
    }
};
```

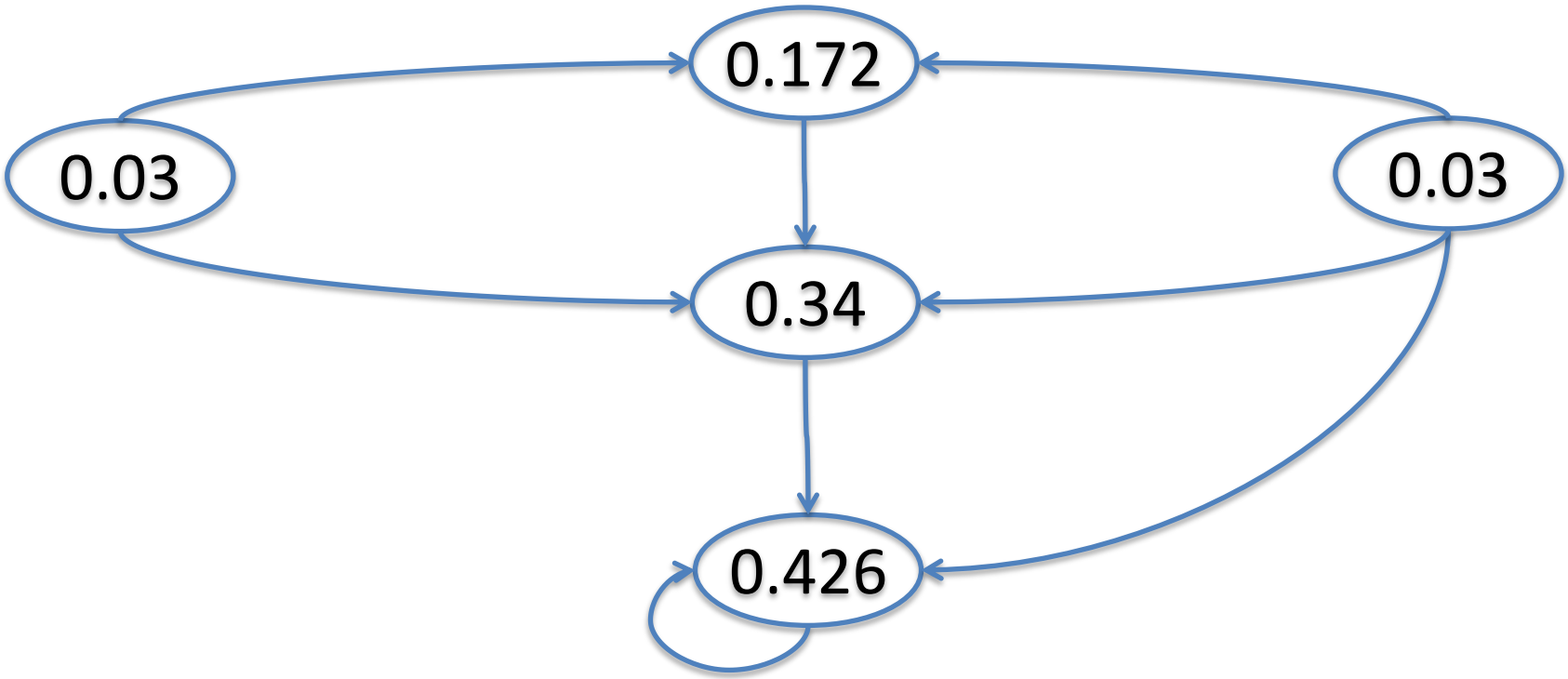


sum = sum(incoming values)
rank = 0.15 / 5 + 0.85 * sum

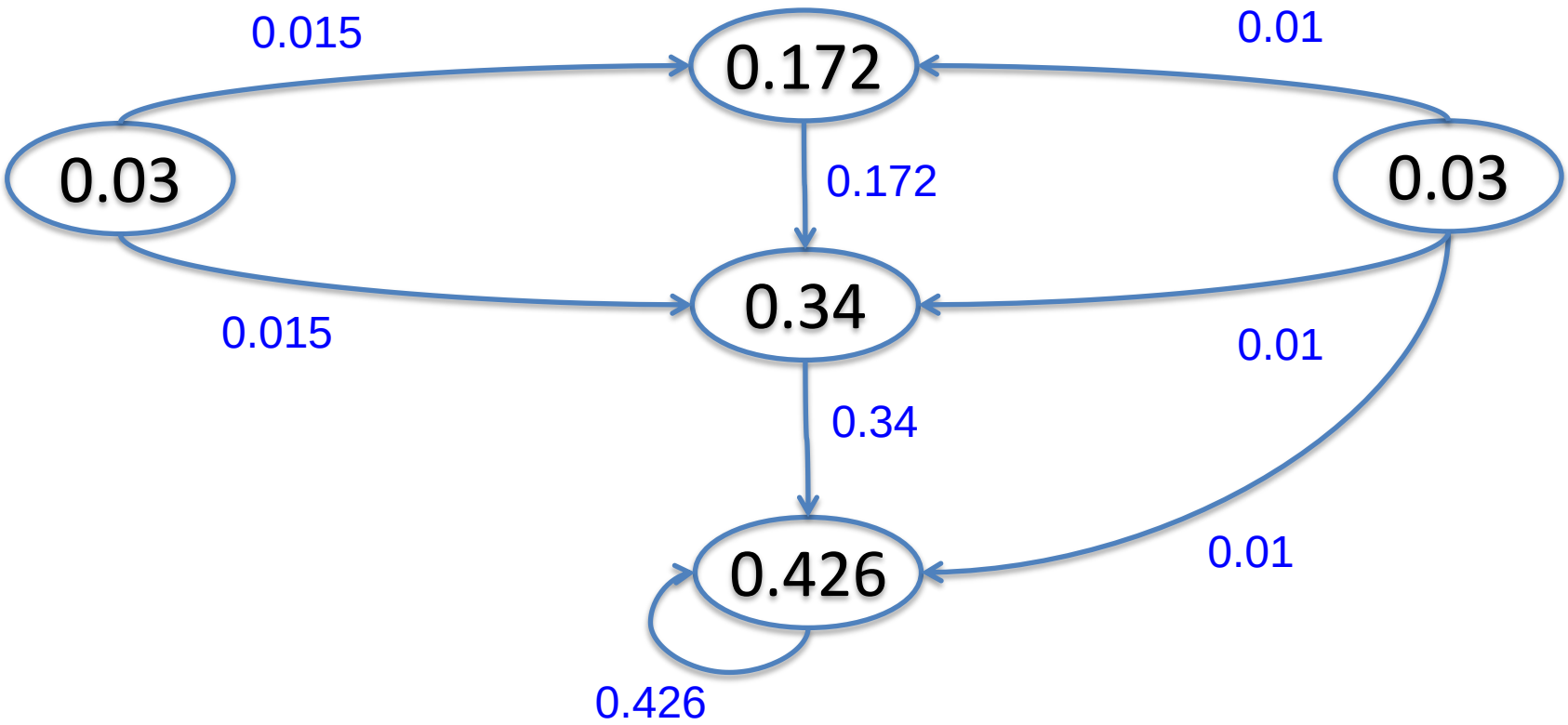

0.1
0.066
0.2
0.2
0.2
0.2
0.1
0.066
0.2
0.066
0.2
0.2



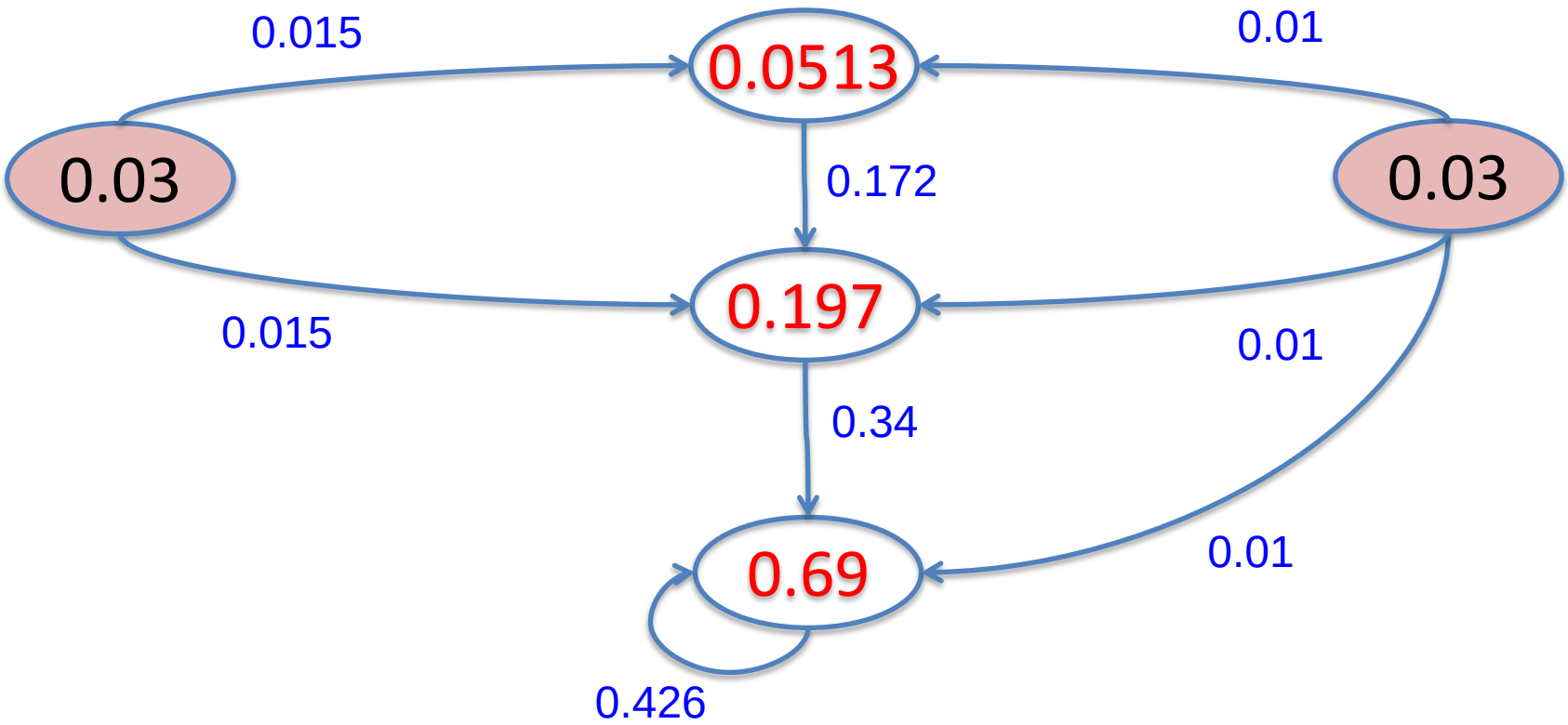
sum = sum(incoming values)
rank = 0.15 / 5 + 0.85 * sum



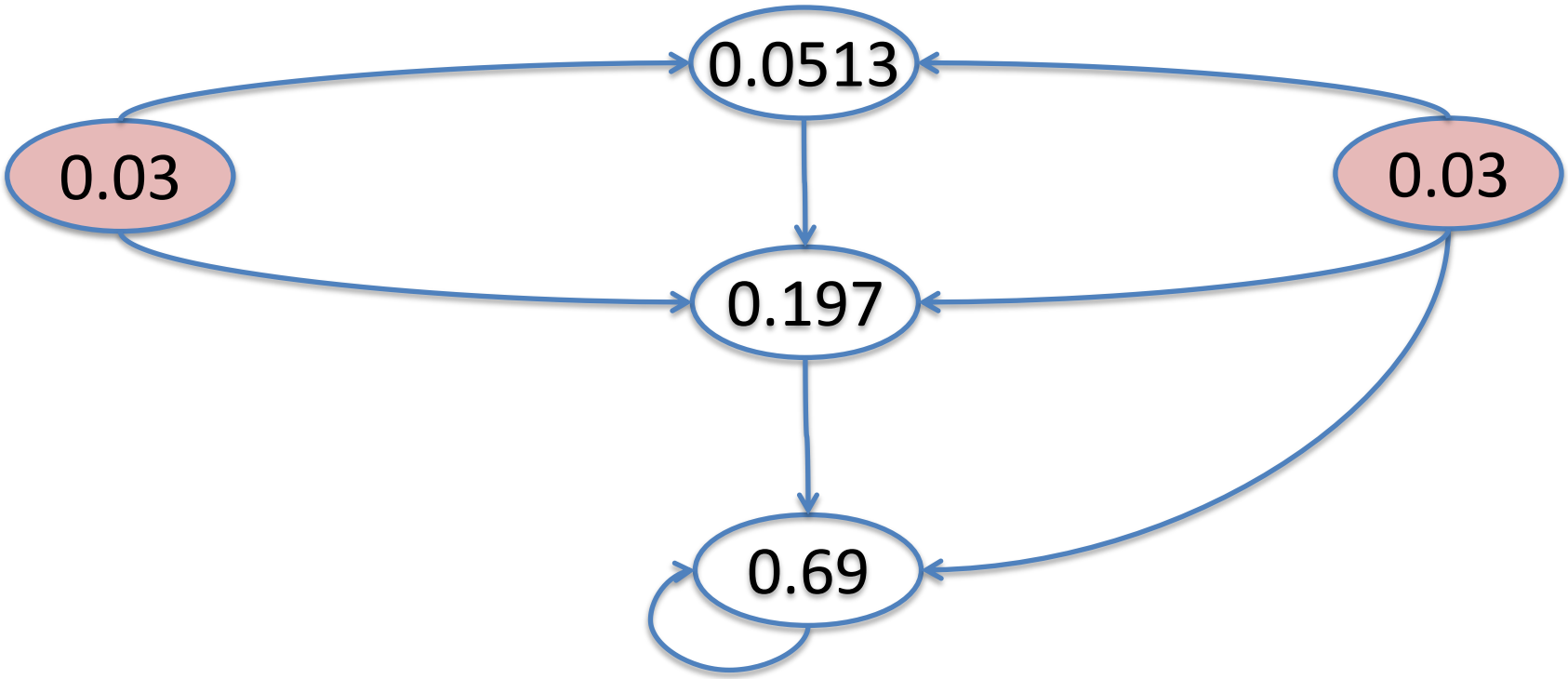
sum = sum(incoming values)
rank = 0.15 / 5 + 0.85 * sum



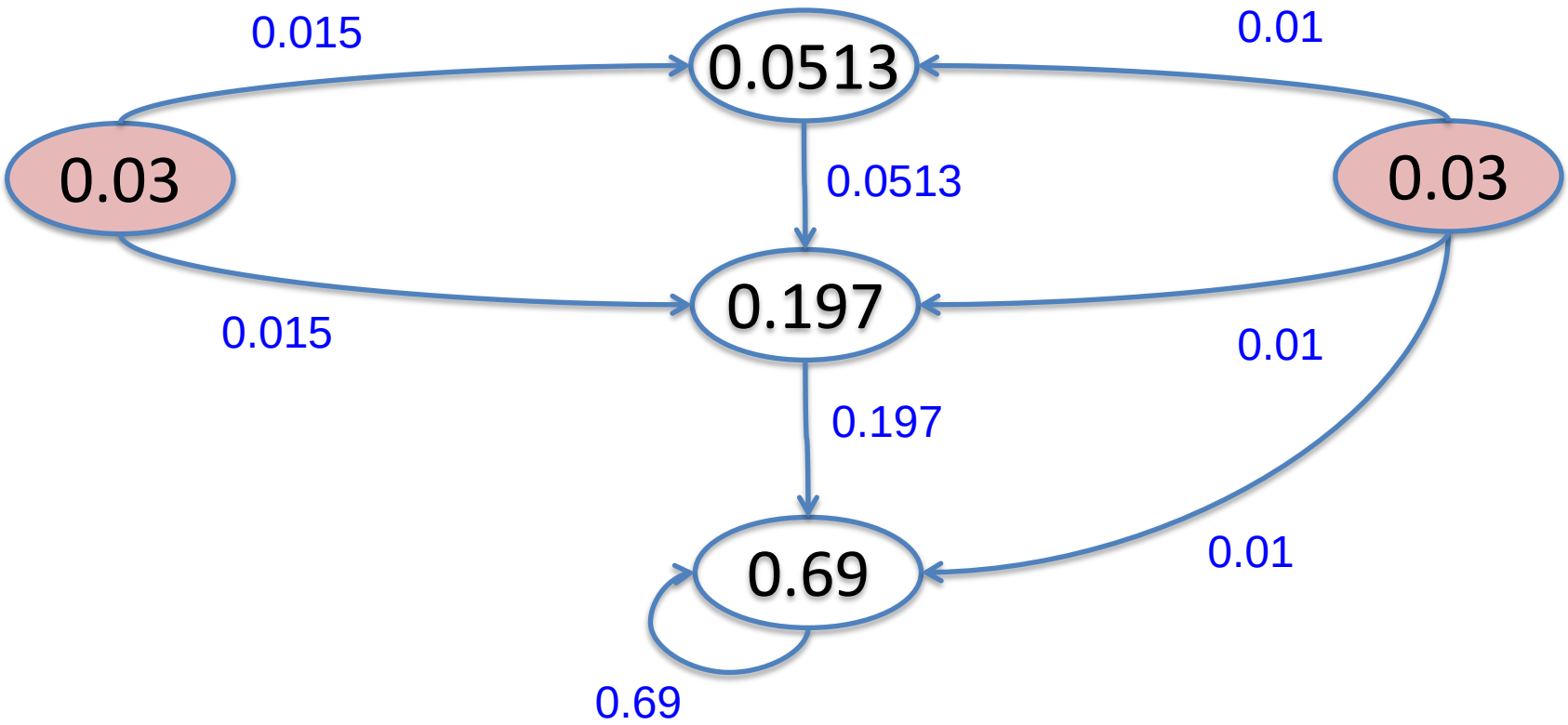
sum = sum(incoming values)
rank = 0.15 / 5 + 0.85 * sum



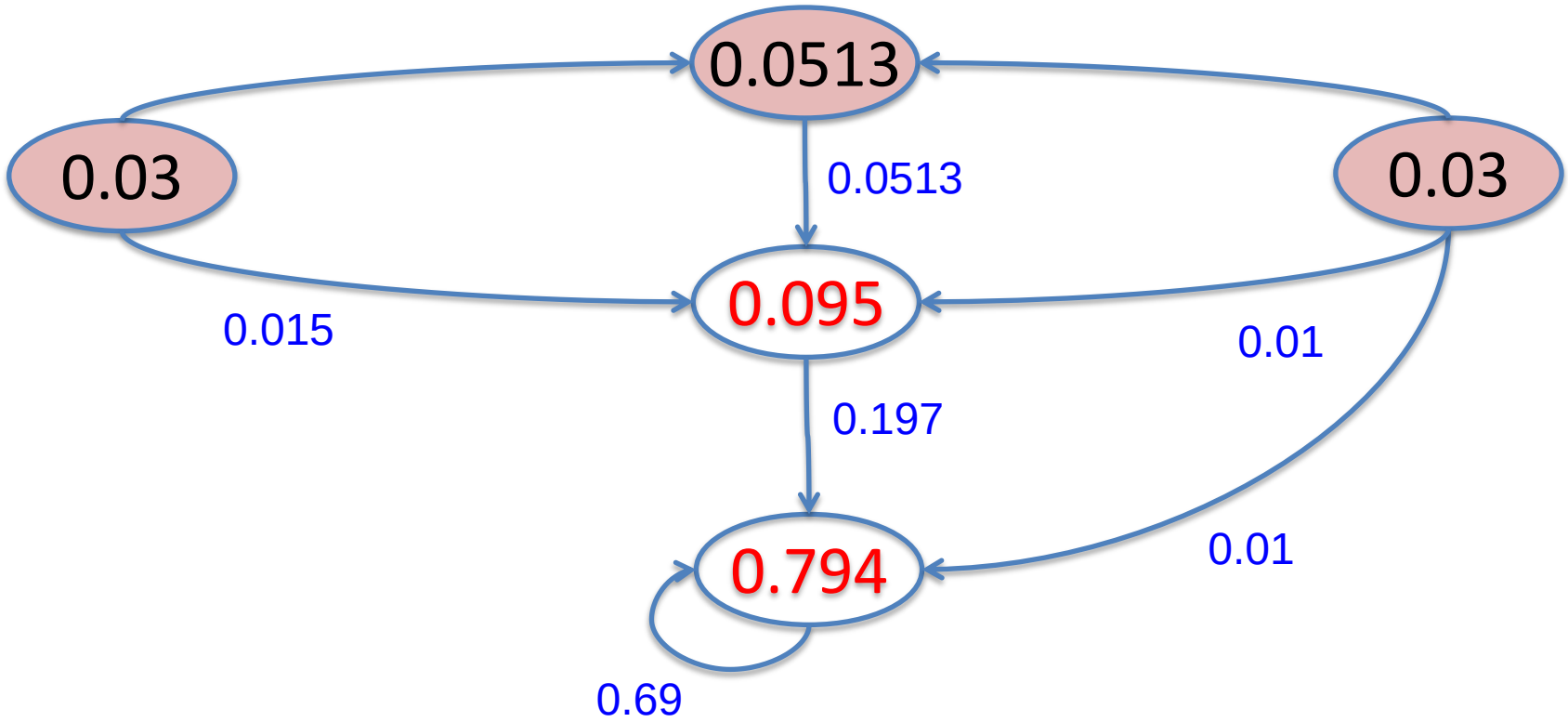
sum = sum(incoming values)
rank = 0.15 / 5 + 0.85 * sum



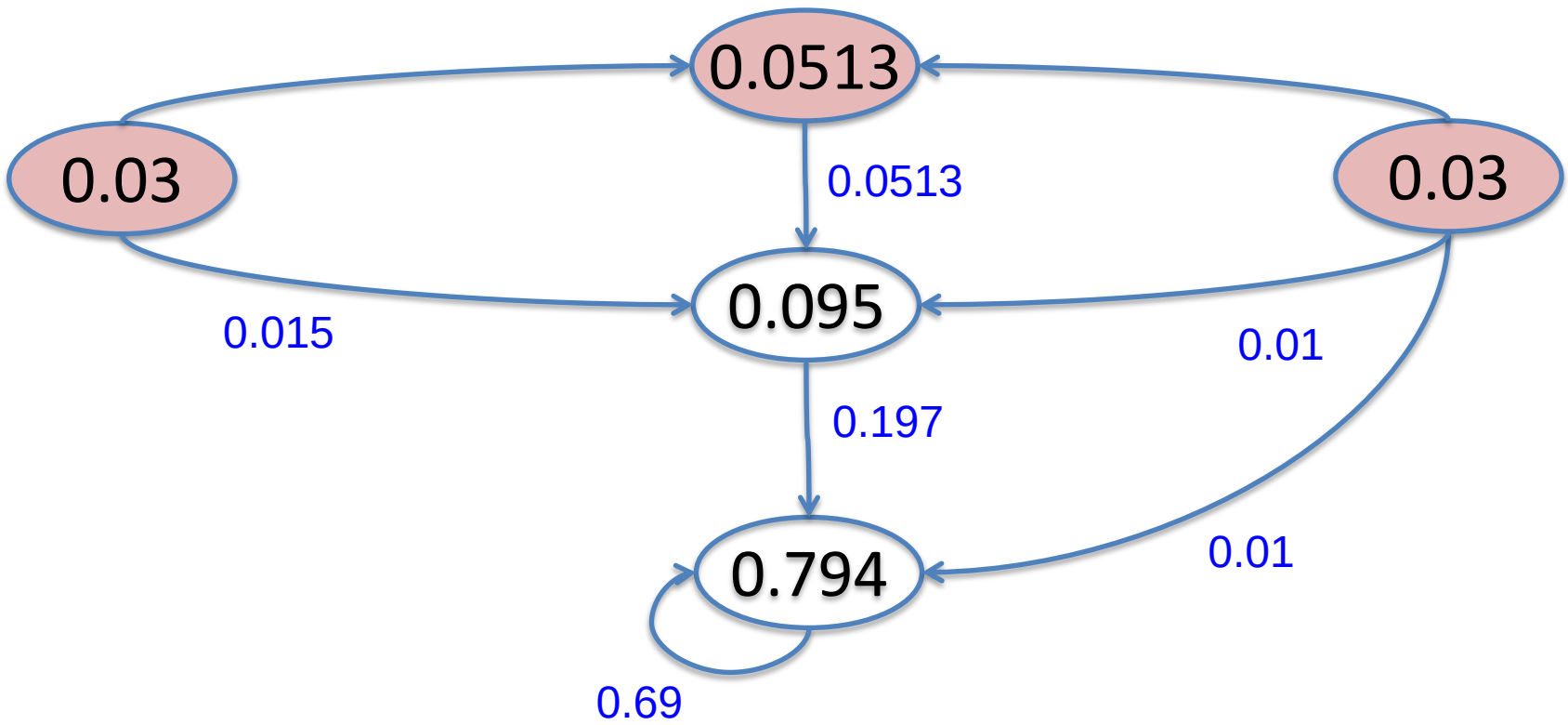
sum = sum(incoming values)
rank = 0.15 / 5 + 0.85 * sum



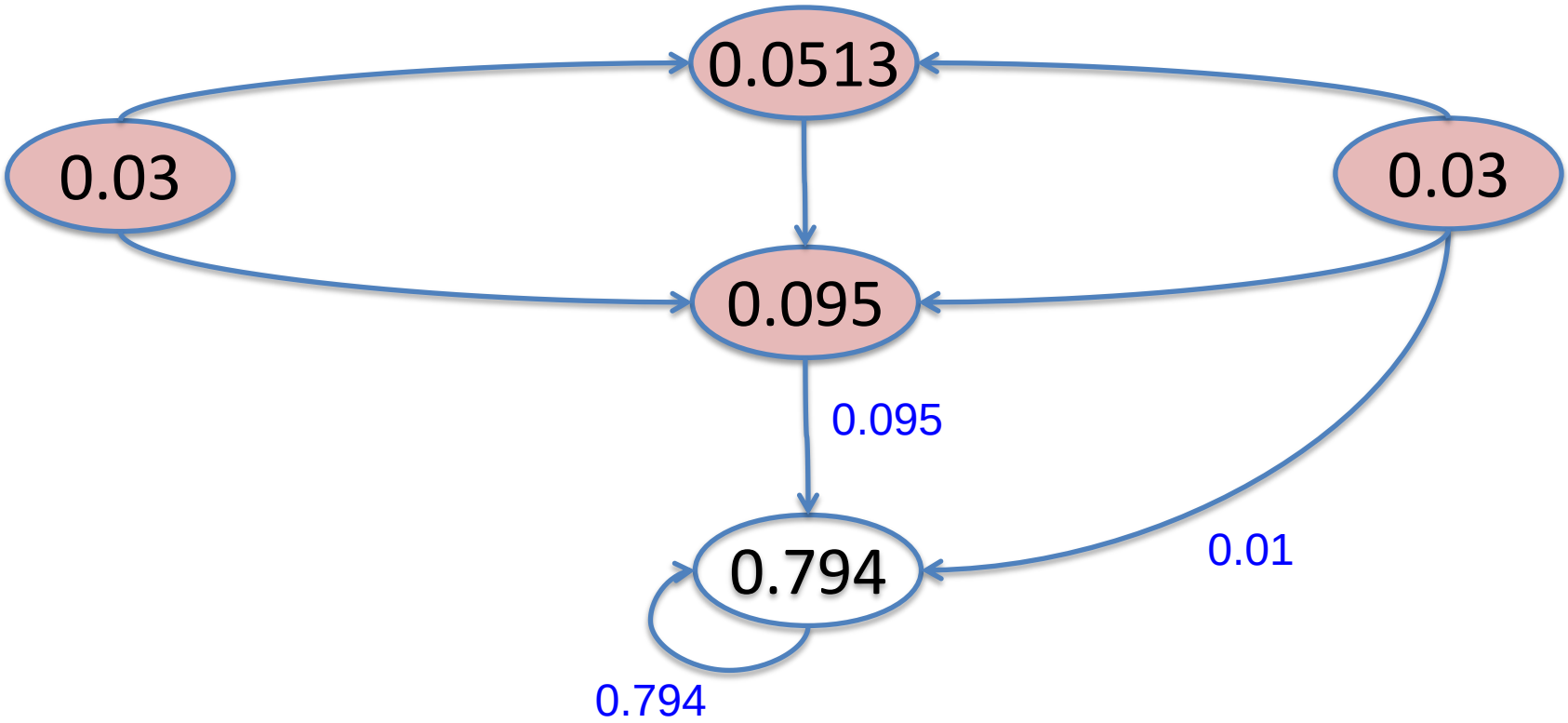
sum = sum(incoming values)
rank = 0.15 / 5 + 0.85 * sum



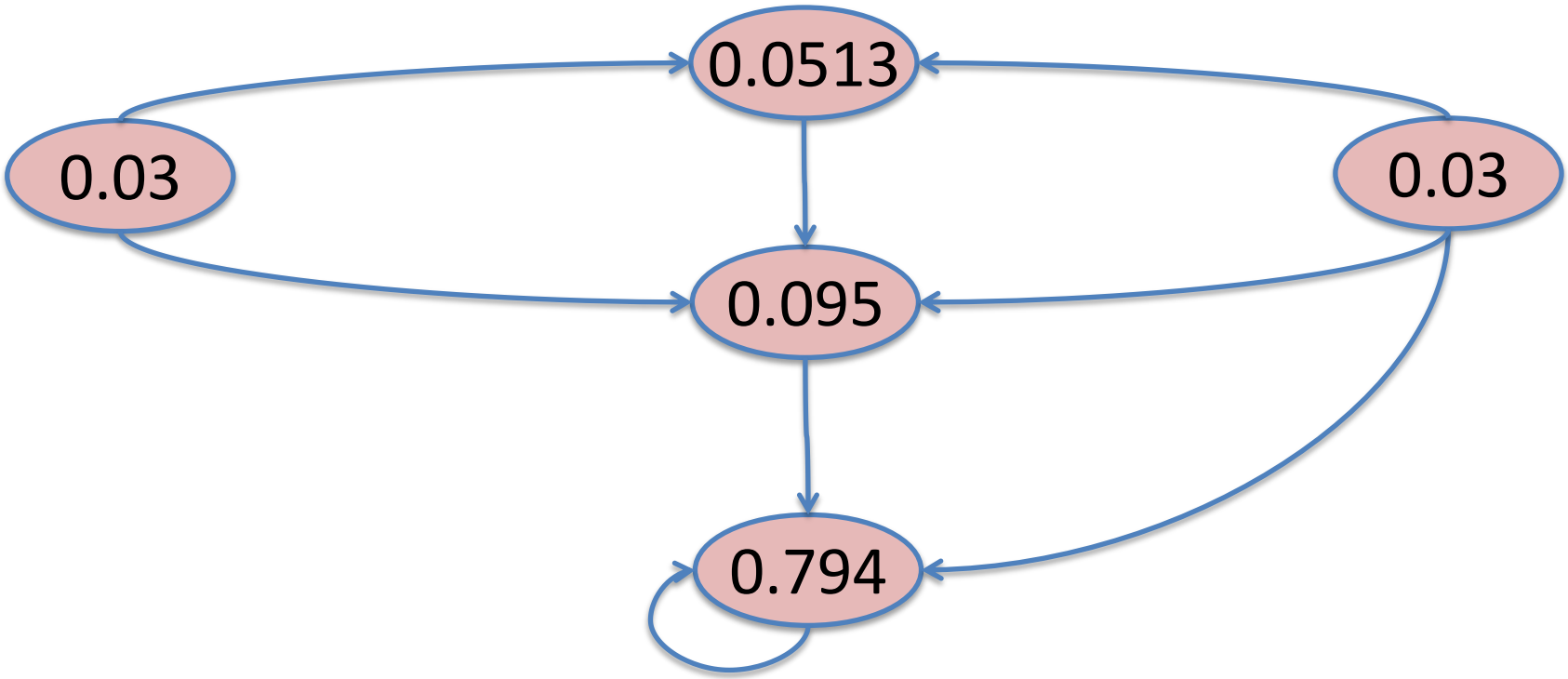
sum = sum(incoming values)
rank = 0.15 / 5 + 0.85 * sum



sum = sum(incoming values)
rank = 0.15 / 5 + 0.85 * sum



sum = sum(incoming values)
rank = 0.15 / 5 + 0.85 * sum



sum = sum(incoming values)
rank = 0.15 / 5 + 0.85 * sum