

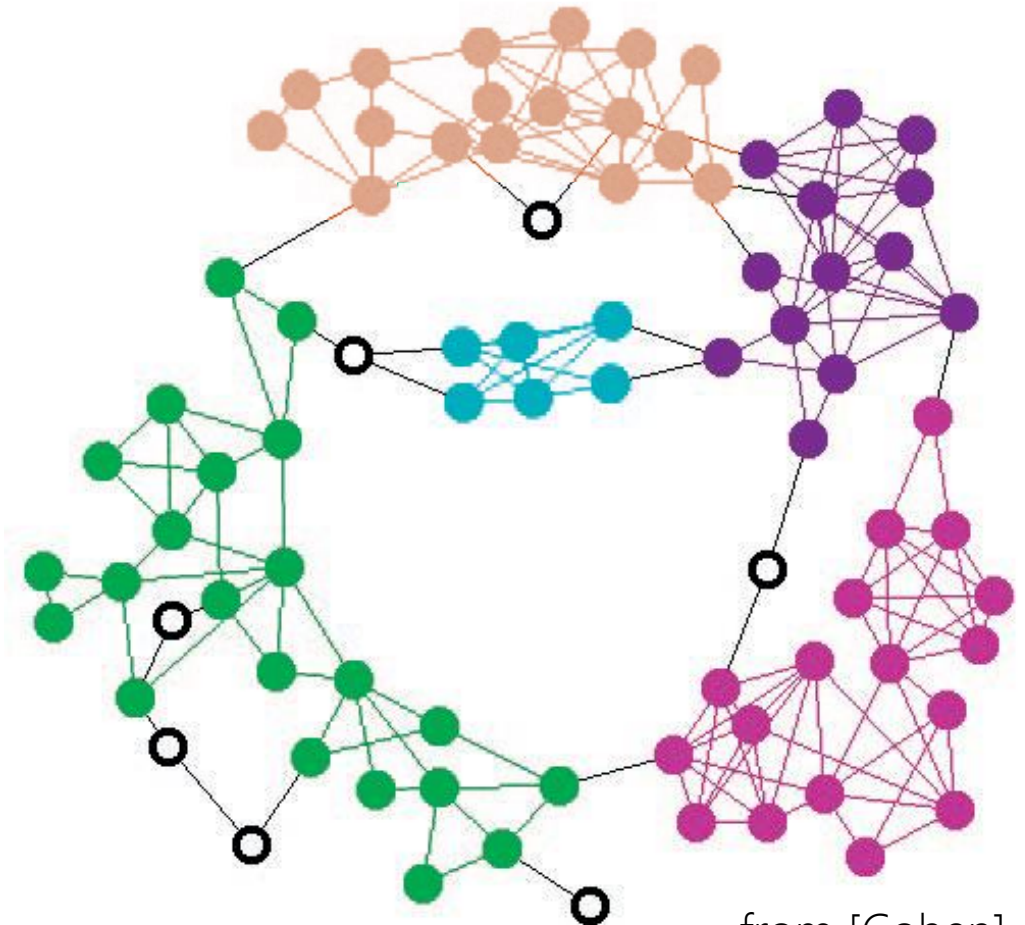
# Graph Mining with Spark

Distributed Big Data Analytics Seminar – Tim Draeger, Ricarda Schöler

# Problem

## Finding highly connected sub-graphs

- Why is this important?
  - Social Media graphs: groups of friends/family/co-workers
  - Website interlinking
- Why is this difficult?
  - Possible solution set size:  $2^{|V|}$ 
    - Exponential run time for naive approach
  - Often millions of vertices
    - ☹️



# The Data

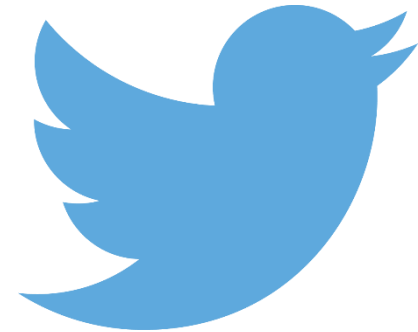
## Wikipedia

- Directed graph of English Wikipedia page interlinks from 2007
- ~1.9 million vertices, ~40 million edges, 1 GB size on disc



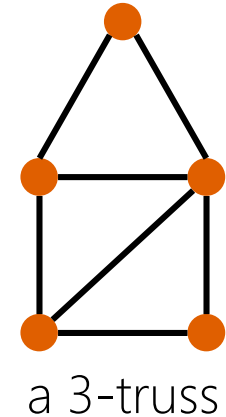
## Twitter

- Directed graph of anonymous Twitter follower/following data from 2009
- ~41 million vertices, ~1.6 billion edges, 25.5 GB size on disc



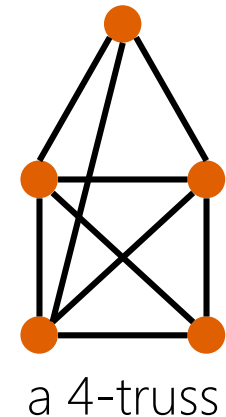
## k-Truss

- **Definition:** a maximal subgraph so that every edge is part of at least  $k-2$  triangles
- Indicates a high connectivity between its nodes
- Can be seen as a relaxation of the clique problem (= fully connected subgraph)



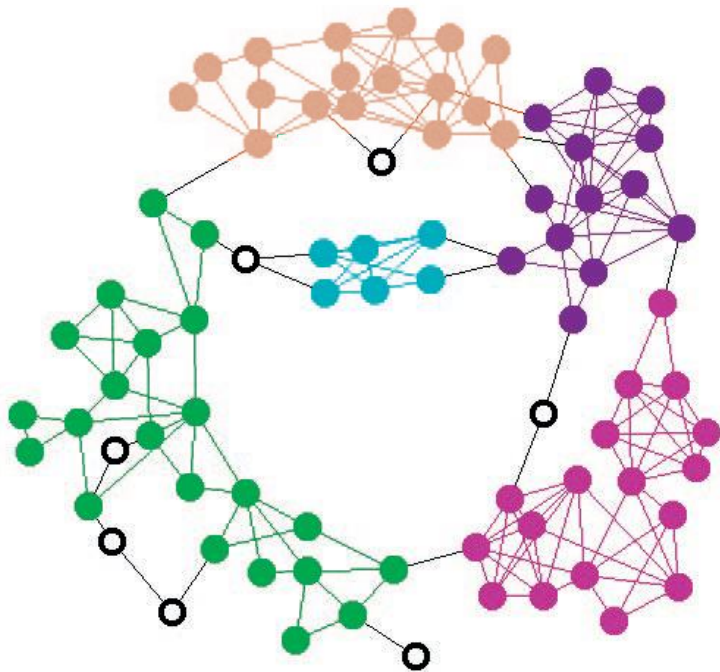
## How?

- Find all triangles in the graph
- Recursively remove all edges that are in  $< k$  triangles
- Return sets of nodes that are still connected

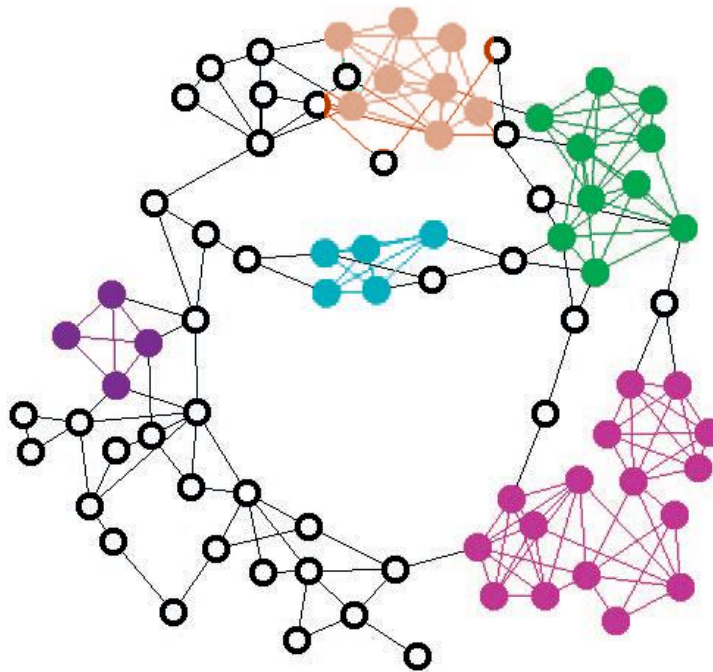


# k-Trusses

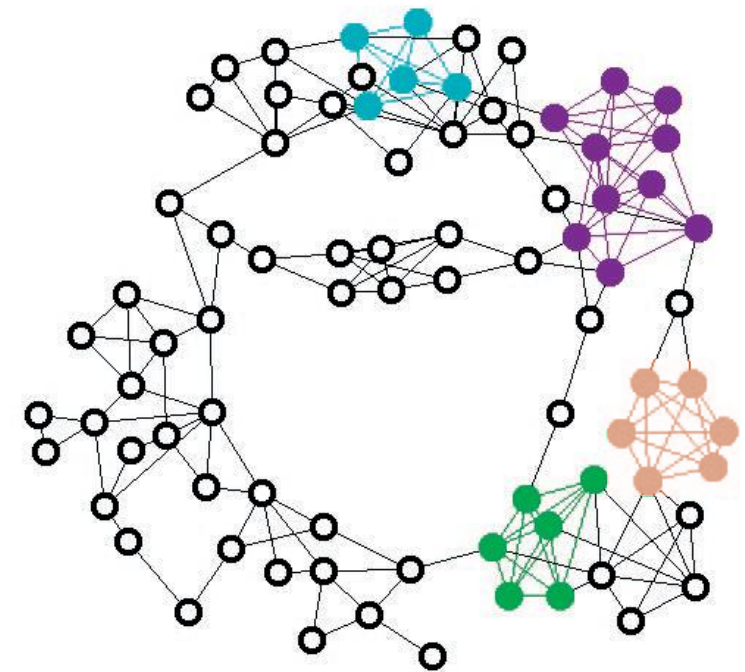
$k = 3$



$k = 4$



$k = 5$



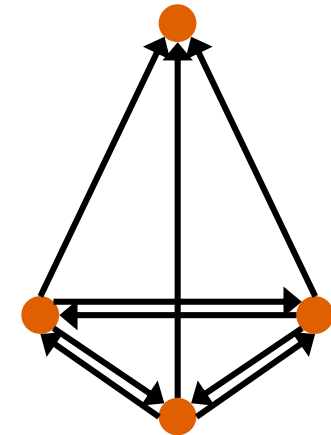
from [Cohen]



# Direction?

## Two possibilities

- Any direction
  - Accept trusses where either direction of an edge exists
- Both directions
  - Accept trusses only when both directions of an edge exist



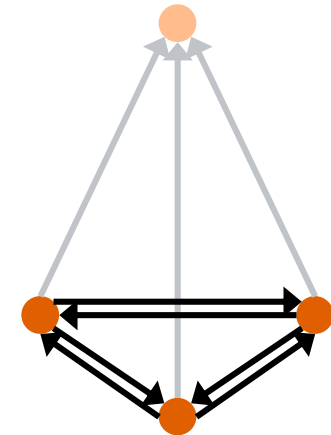
# Direction?

## Two possibilities

- Any direction
  - Accept trusses where either direction of an edge exists
- Both directions
  - Accept trusses only when both directions of an edge exist

## Decision

- Bidirectional edges only
  - Only nodes that actually interact with one another should form a truss
- Can use pre-processing step to create graph with bidirectional edges only



# Implementation

1. Create new graph from bidirectional edges only
  2. Set  $k$  = arbitrary value, subgraphs = (fullGraph)
  3. Find all  $k$ -trusses for each subgraph *after [Cohen]*
  4. If none exist:
    5. Reduce  $k$ , go to 3.
  6. Set subgraphs =(truss1, truss2, ...)
  7. Increase  $k$ , go to 3.
- (Increase or reduce  $k$  according to a binary search strategy)

Abort if  $k$  has already been seen before



# Evaluation – starting k & #cores

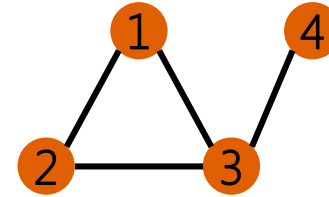
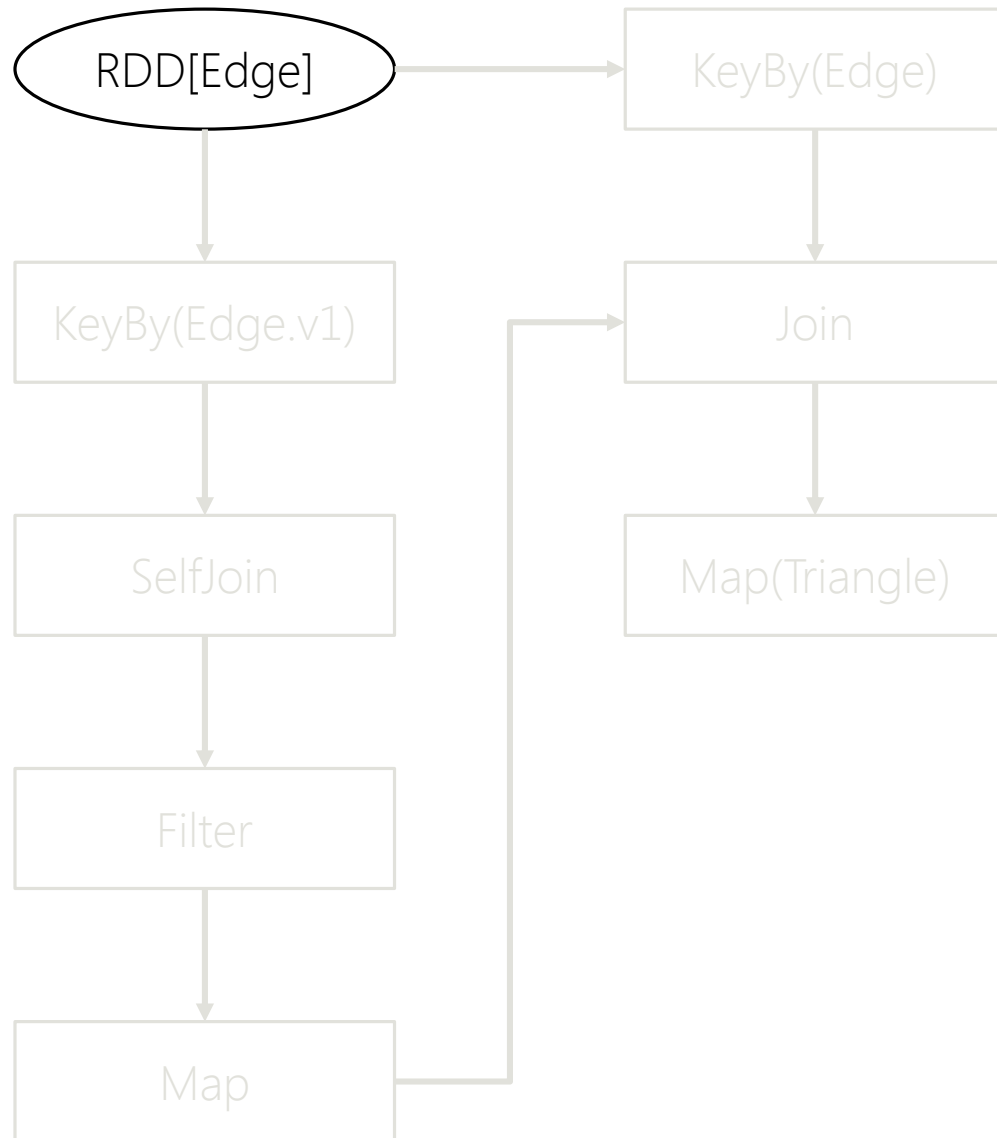
## Starting value for k (full run – maxTrussSize = 28)

- k = 10 – 17 minutes (k values tried: 10, 20, 40, 30, 25, 27, 28, 29, 28)
- k = 20 – 11 minutes (k values tried: 20, 40, 30, 25, 27, 28, 29, 28)
- k = 28 – 10 minutes (k values tried: 28, 56, 42, 35, 31, 29)
- k = 40 – 20 minutes (k values tried: 40, 21, 30, 25, 27, 28, 29, 28)

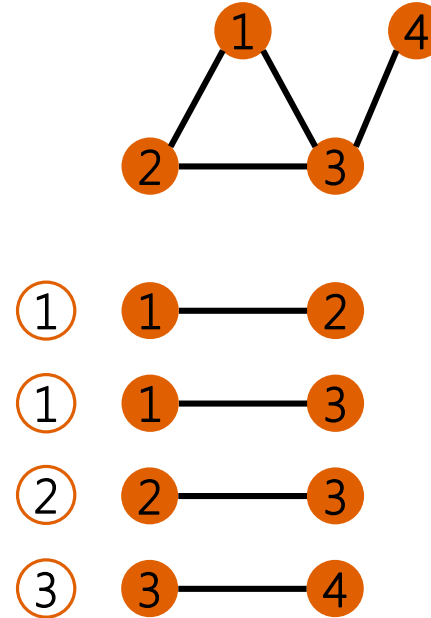
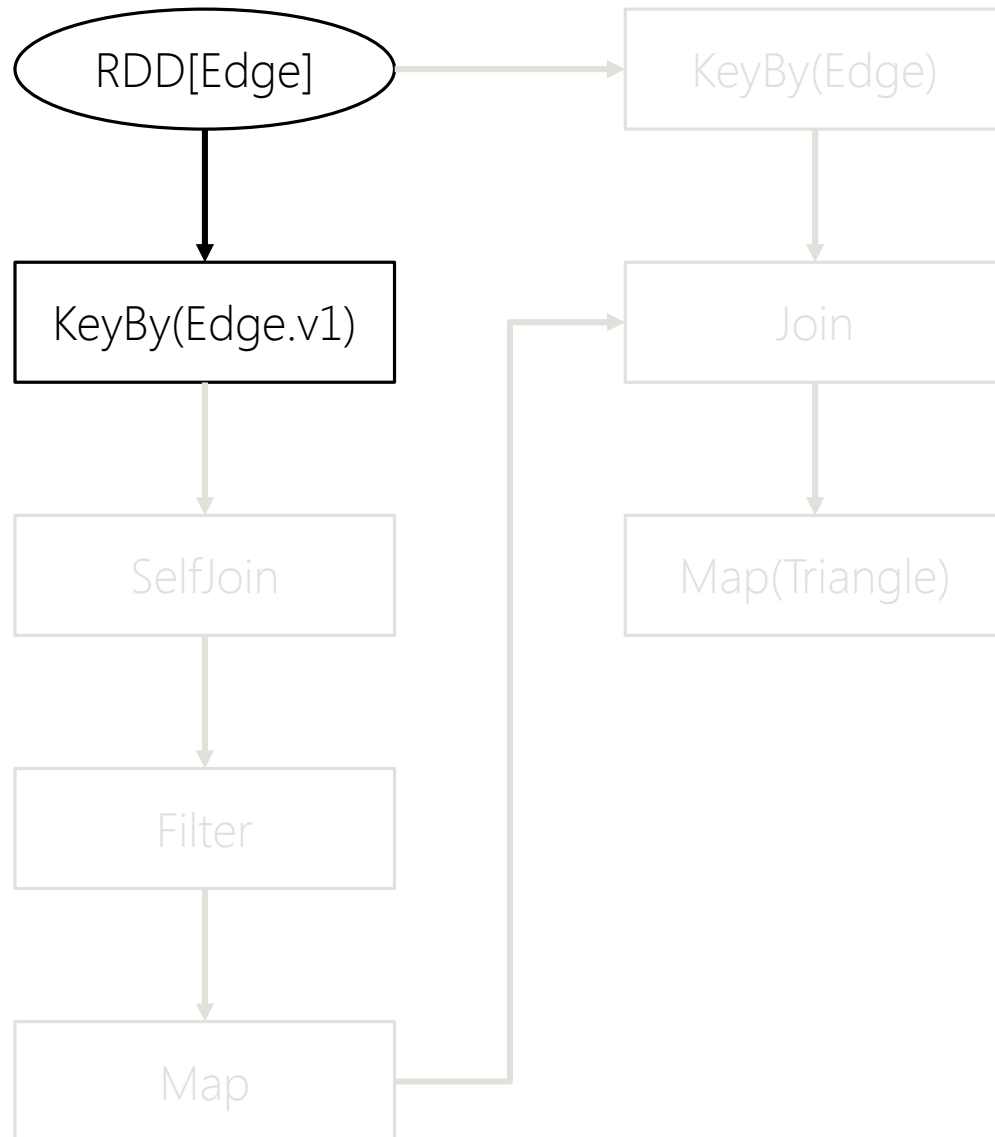
## Number of cores used (triangle calculation)

- 1 core x 5 machines – 16 minutes
- 1 core x 10 machines – 8.3 minutes
- 2 cores x 10 machines – 6.3 minutes

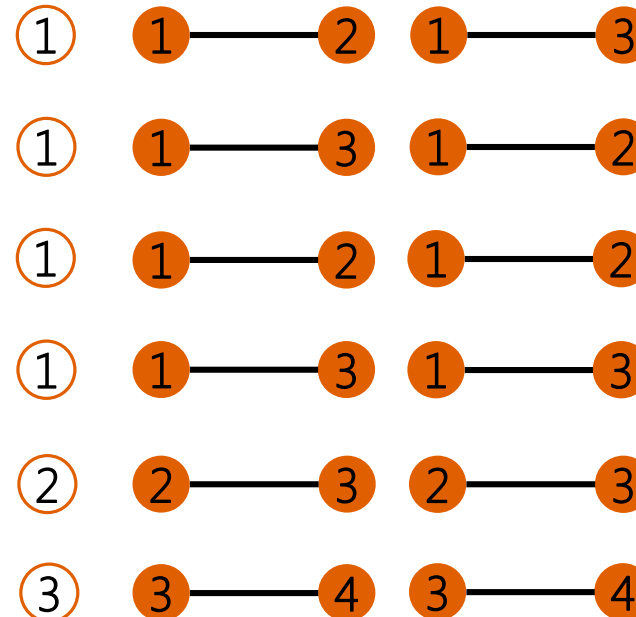
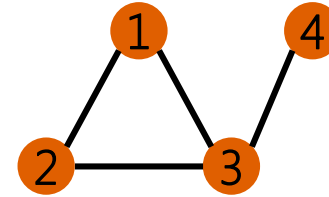
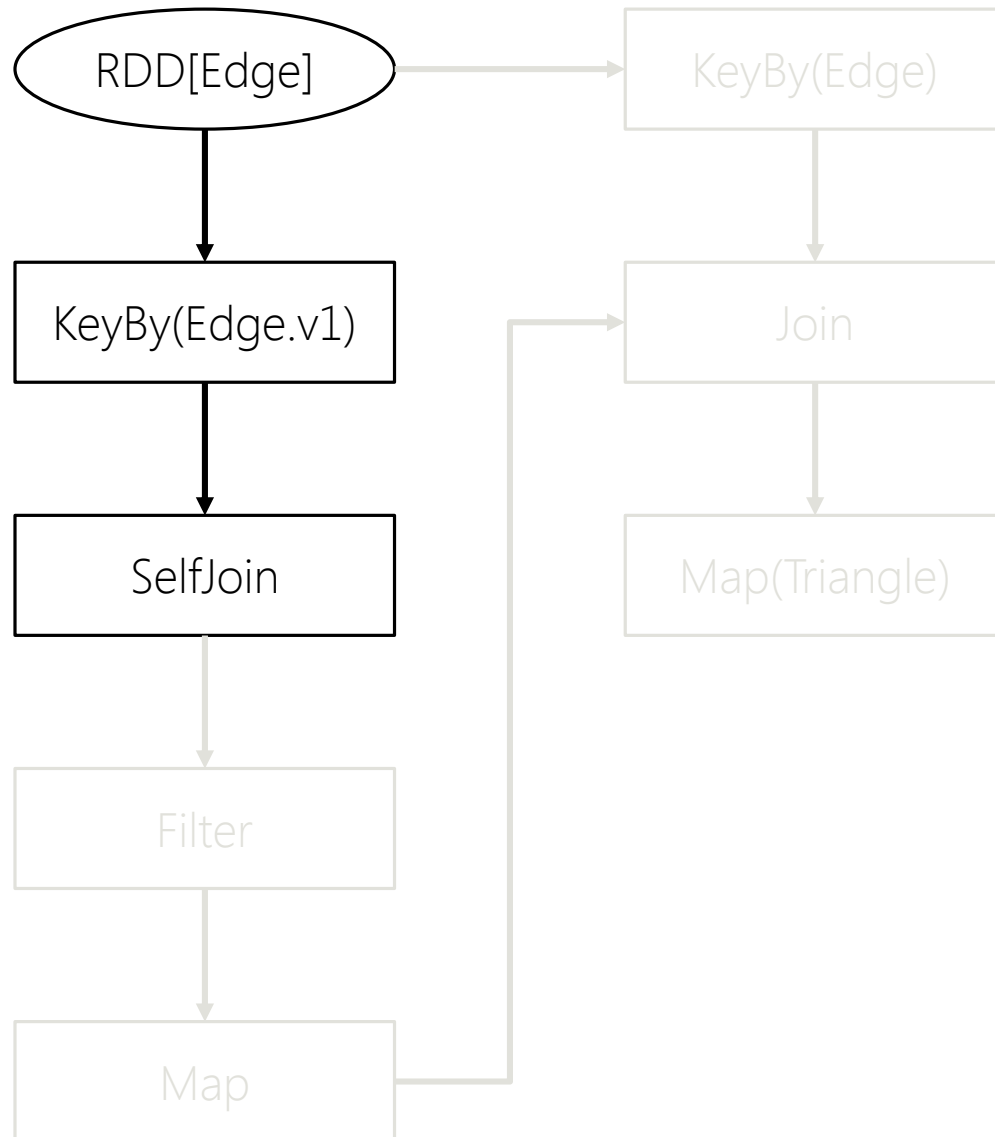
# Implementation – Triangle Generation



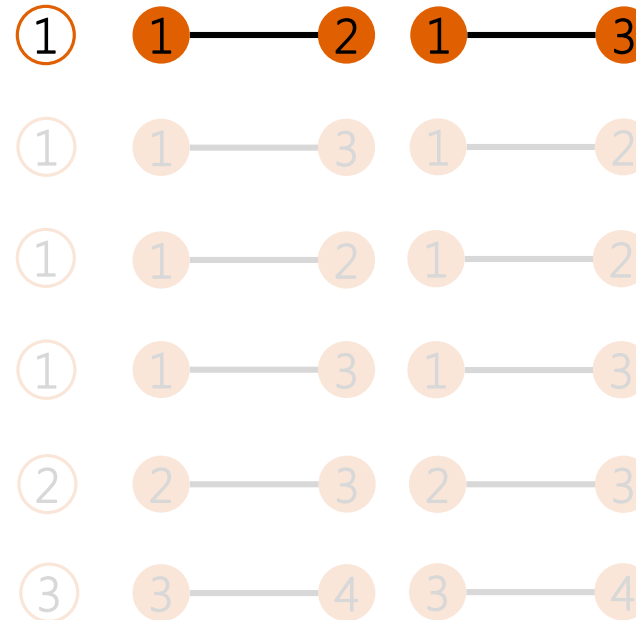
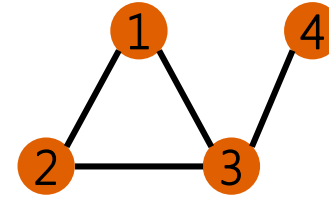
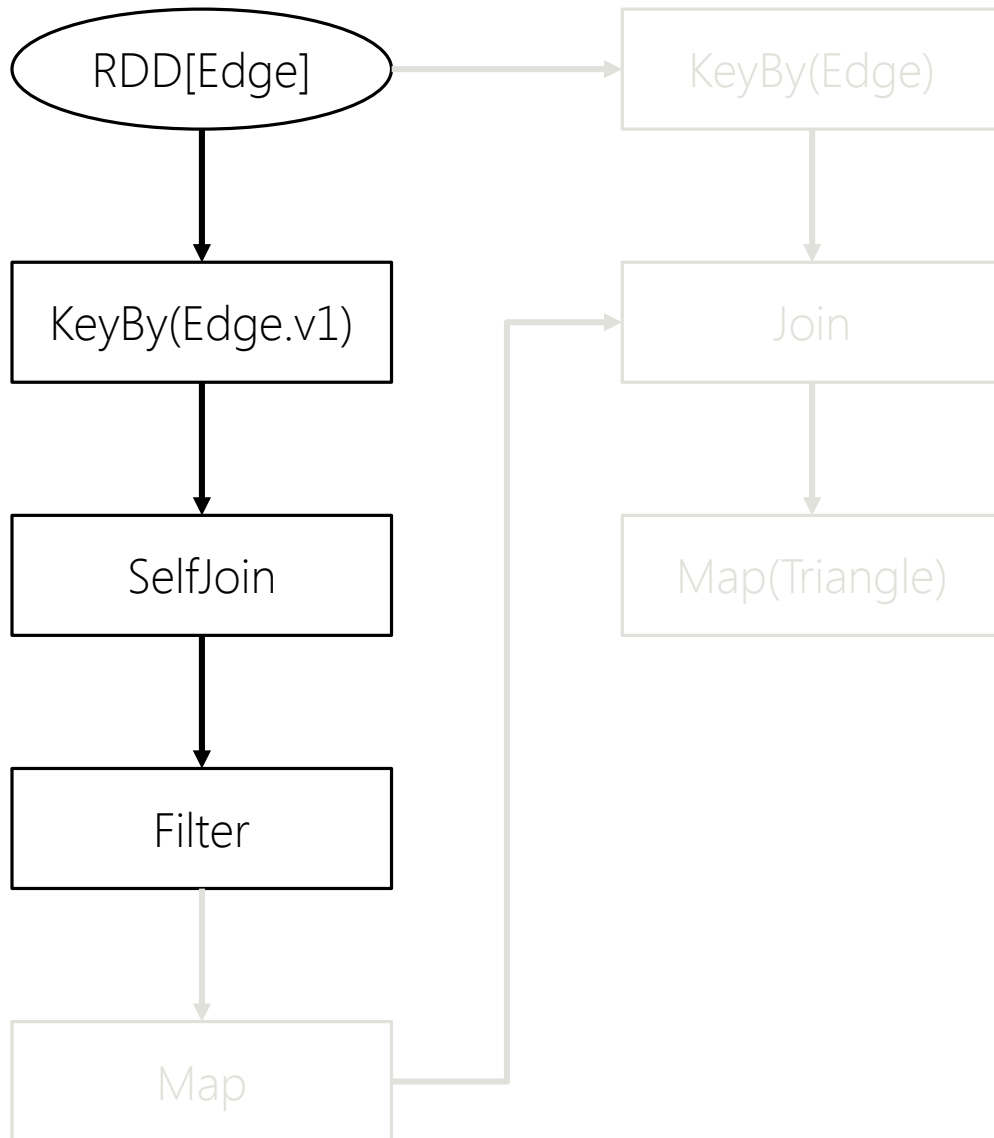
# Implementation – Triangle Generation



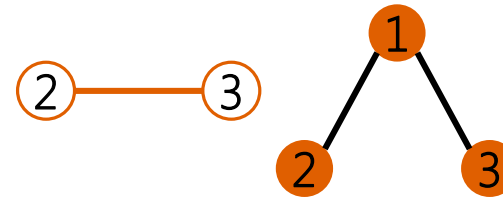
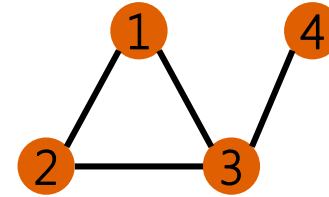
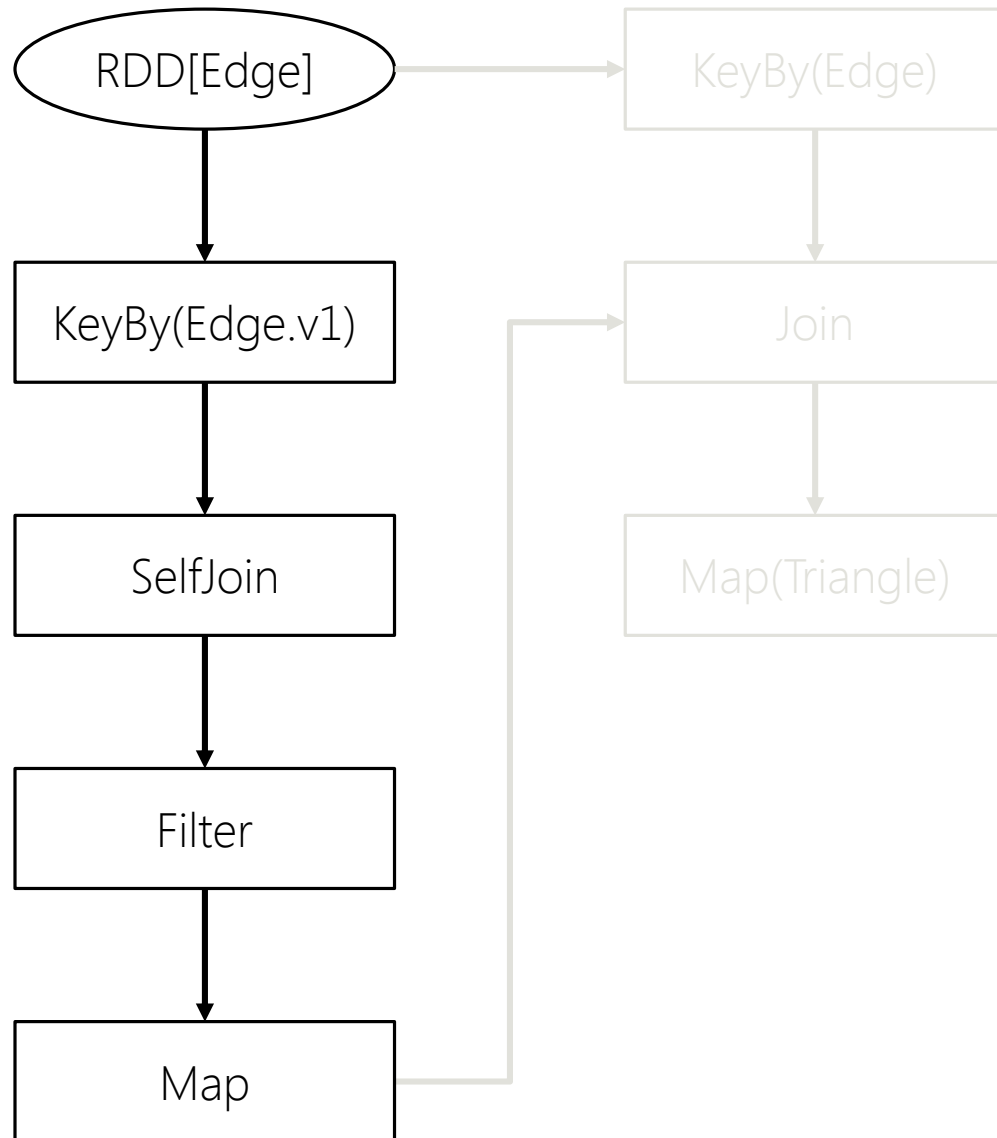
# Implementation – Triangle Generation



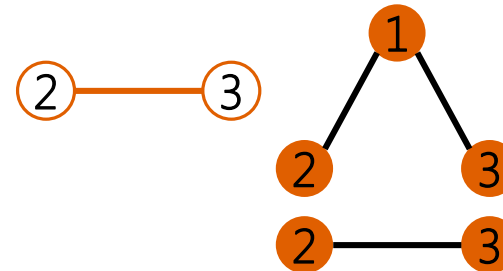
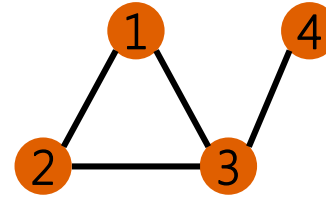
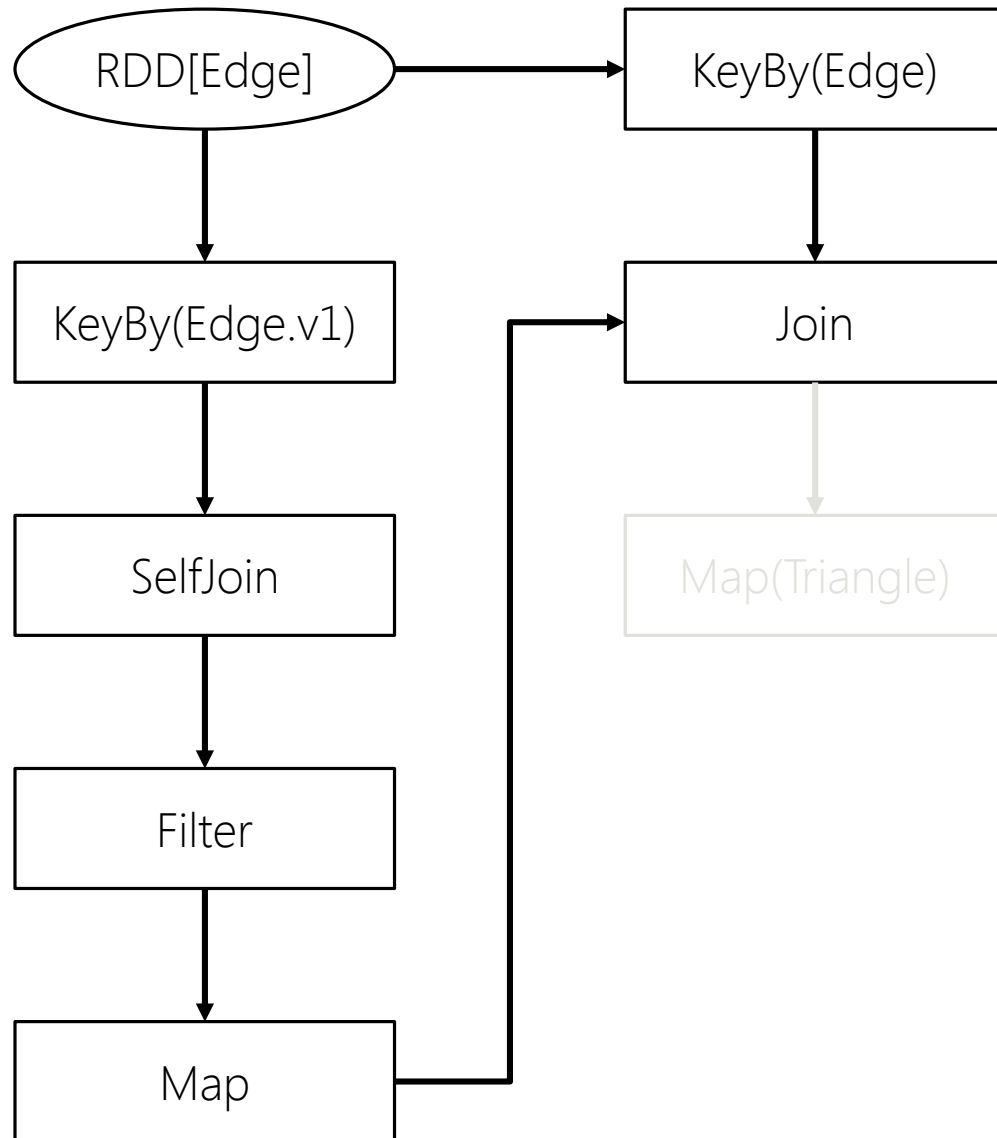
# Implementation – Triangle Generation



# Implementation – Triangle Generation

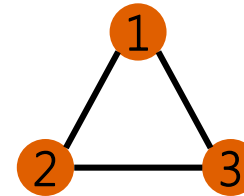
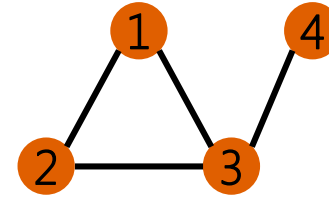
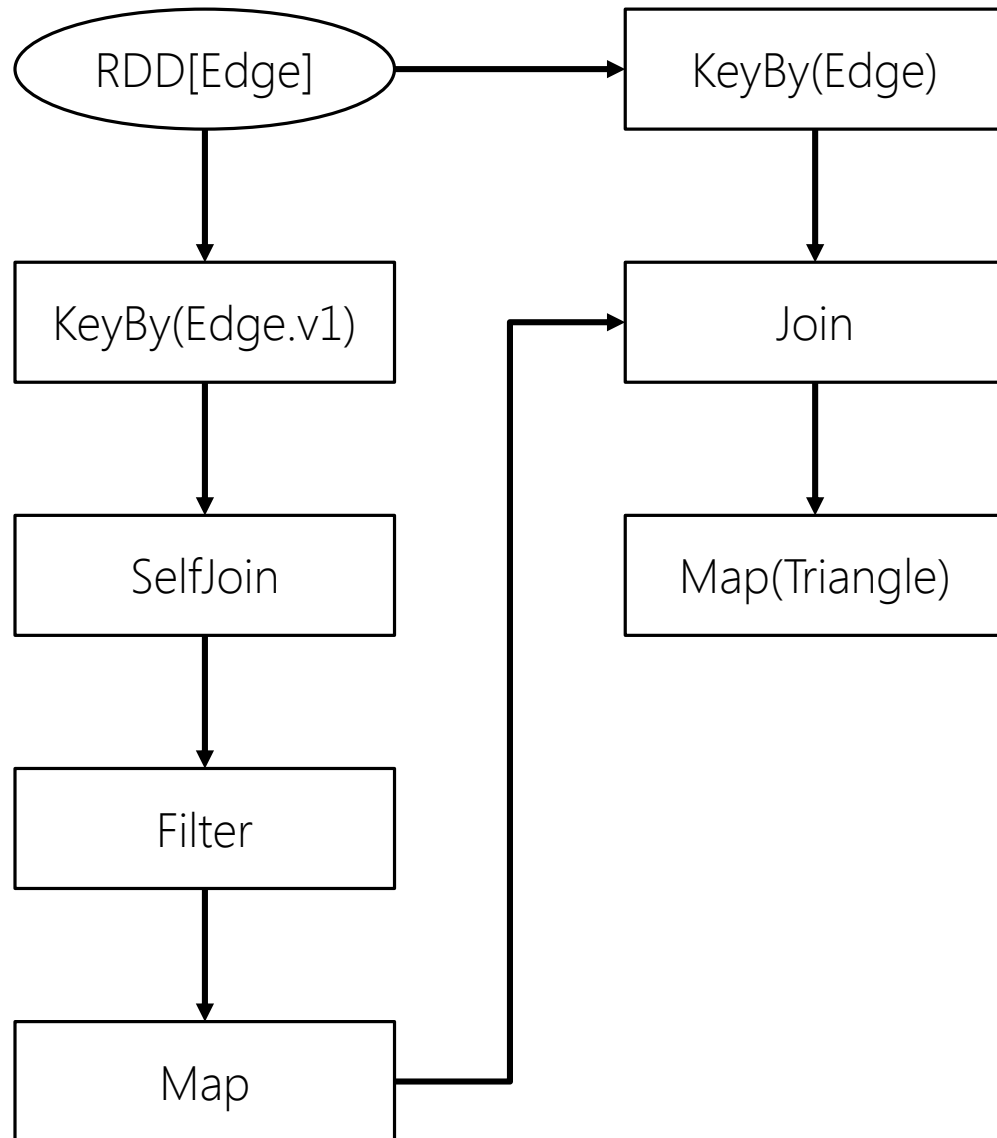


# Implementation – Triangle Generation

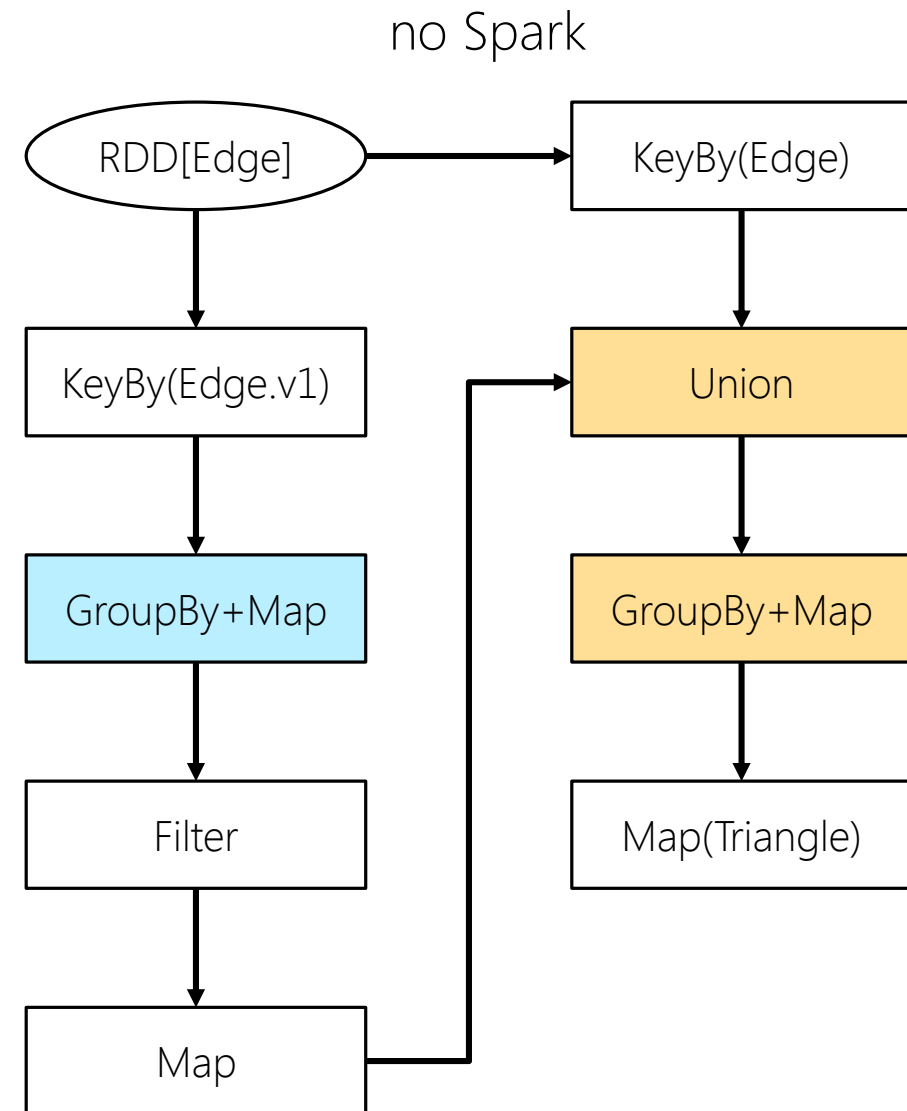
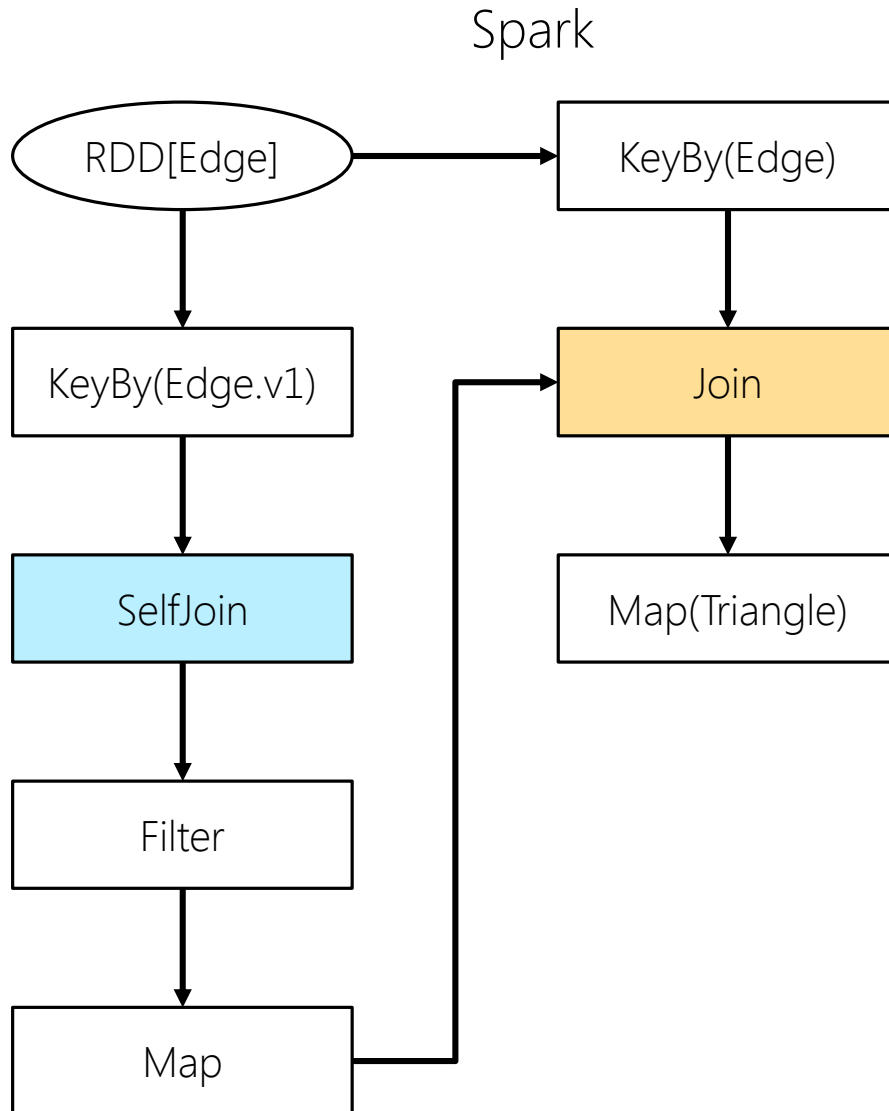




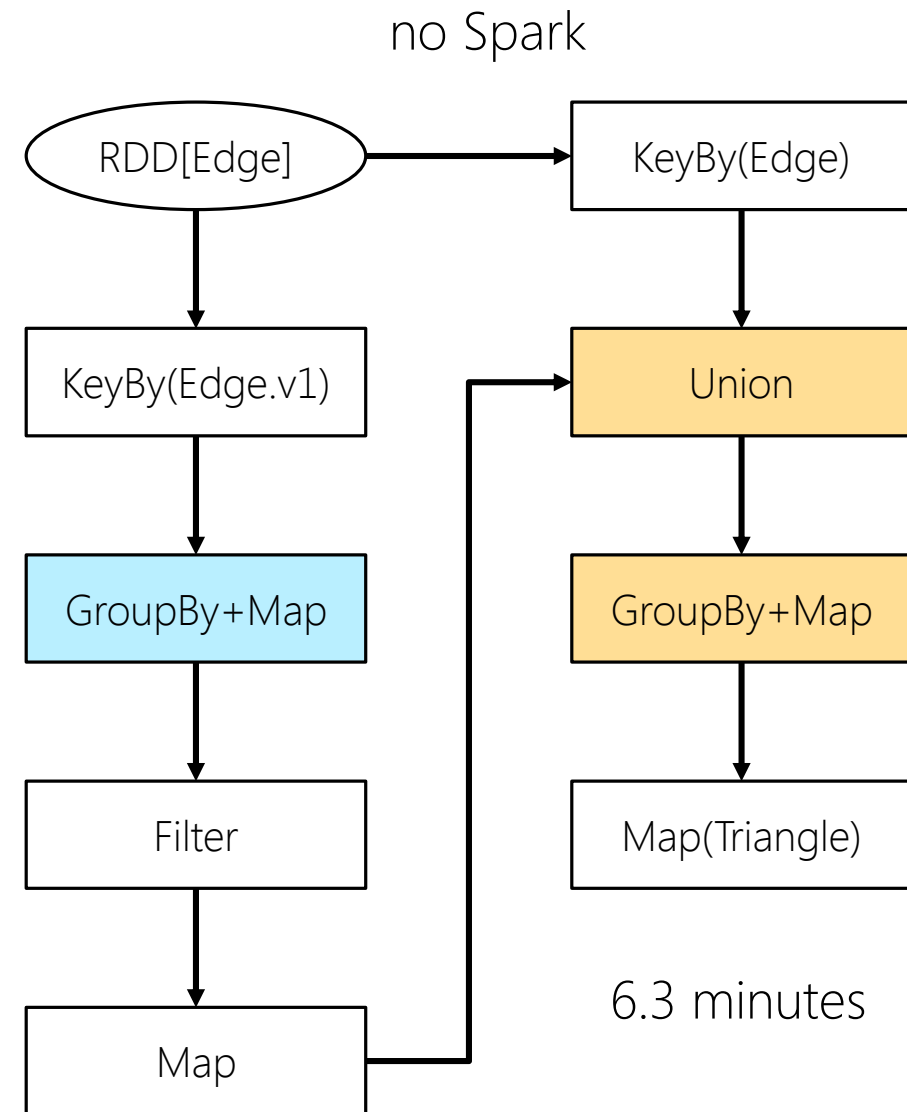
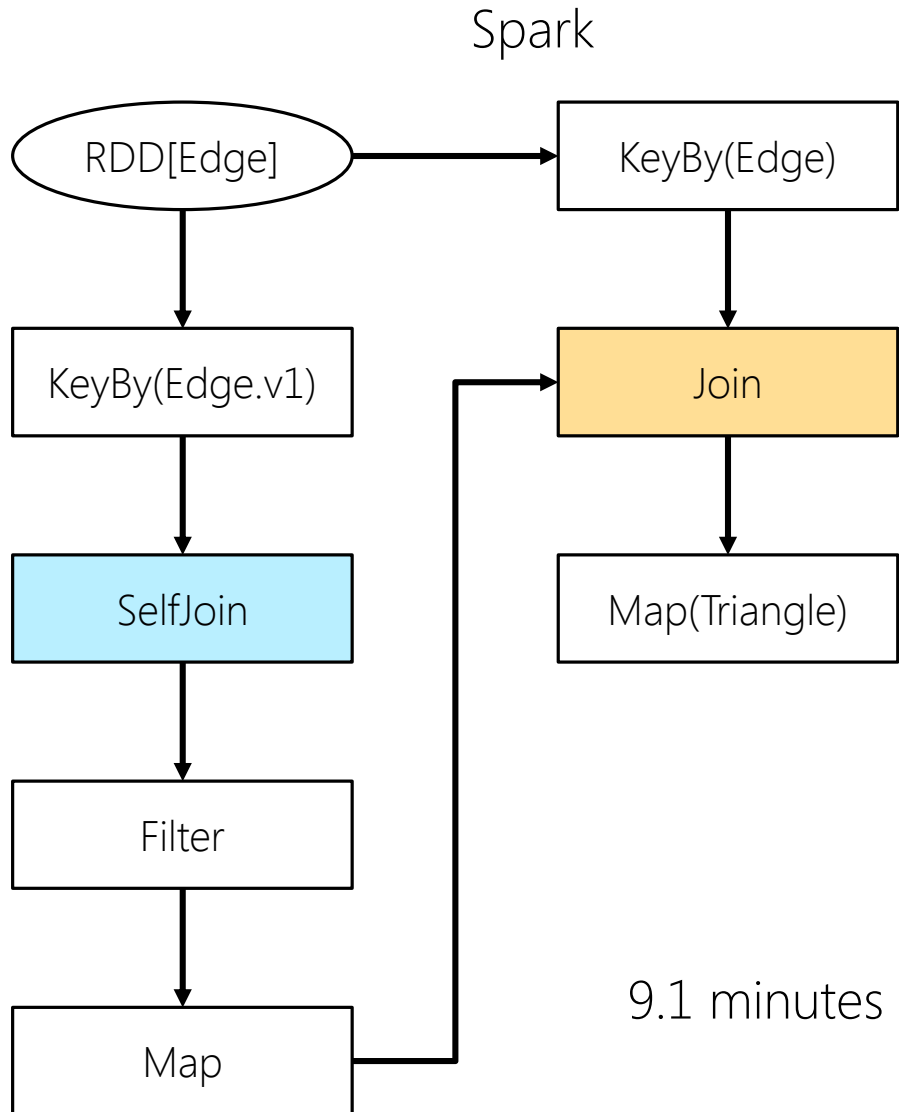
# Implementation – Triangle Generation



# Evaluation – Triangle Generation

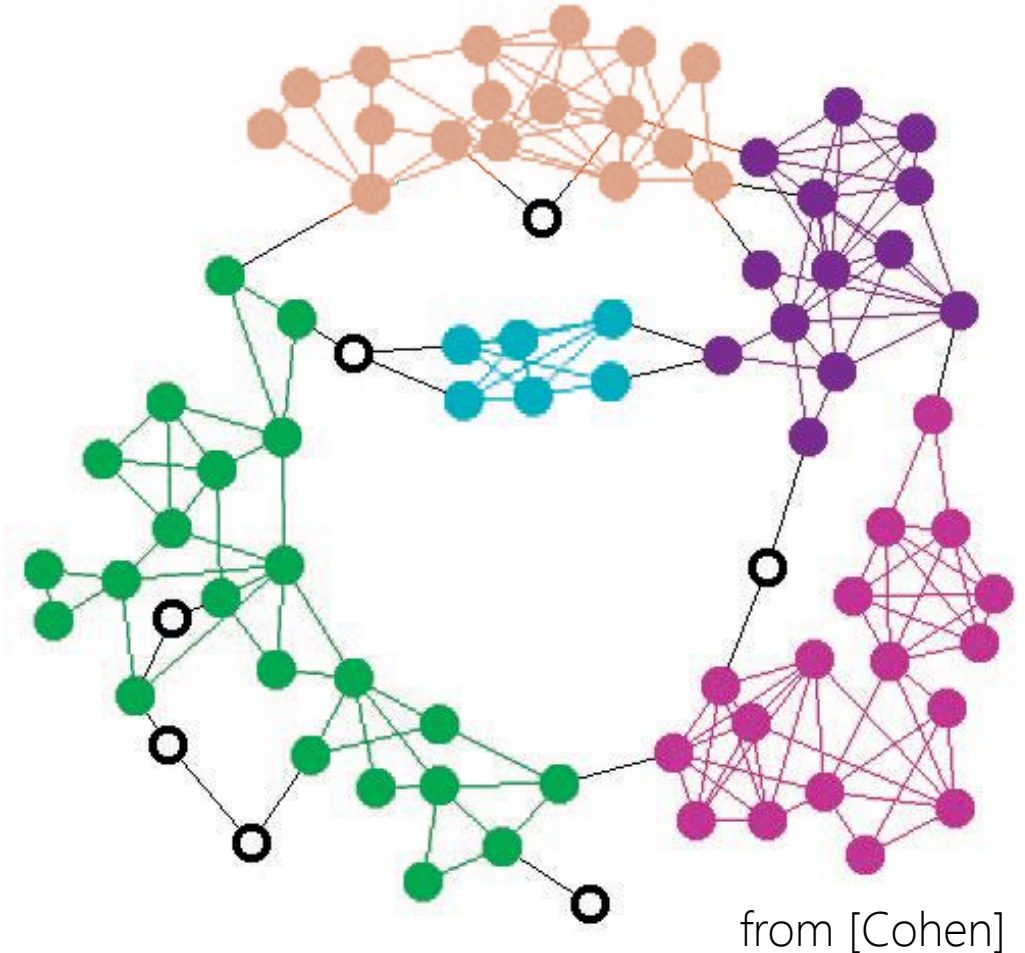


# Evaluation – Triangle Generation



# Conclusions

- Distributed calculation
  - choice of starting k is very influential on run time
  - great scaling with distribution over multiple machines
  - Spark-specific functions don't seem to perform very well
- Future Work
  - Better usage of Spark-specific functions
  - Distributed clique calculation
  - Usage of directed graphs instead



# References

Image on title slide: <http://polkadotimpressions.com/2013/01/18/facebook-graph-search-3/>

[Bron, Kerbosh]: Bron, Coen, and Joep Kerbosch. 'Algorithm 457: finding all cliques of an undirected graph.' *Communications of the ACM* 16, no. 9 (1973): 575-577.

[Cohen]: Jonathan Cohen, 'Graph Twiddling in a MapReduce World'. in *Computing in Science and Engineering* 11(4): 29-41 (2009)

[Xiang et al]: Xiang, J, Guo, C & Aboulnaga, A 2013, 'Scalable maximum clique computation using MapReduce'. in *Proceedings - International Conference on Data Engineering.*, 6544815, pp. 74-85, 29th International Conference on Data Engineering, ICDE 2013