Graph Mining with Spark and Flink

Distributed Big Data Analytics Seminar – Tim Draeger, Ricarda Schüler

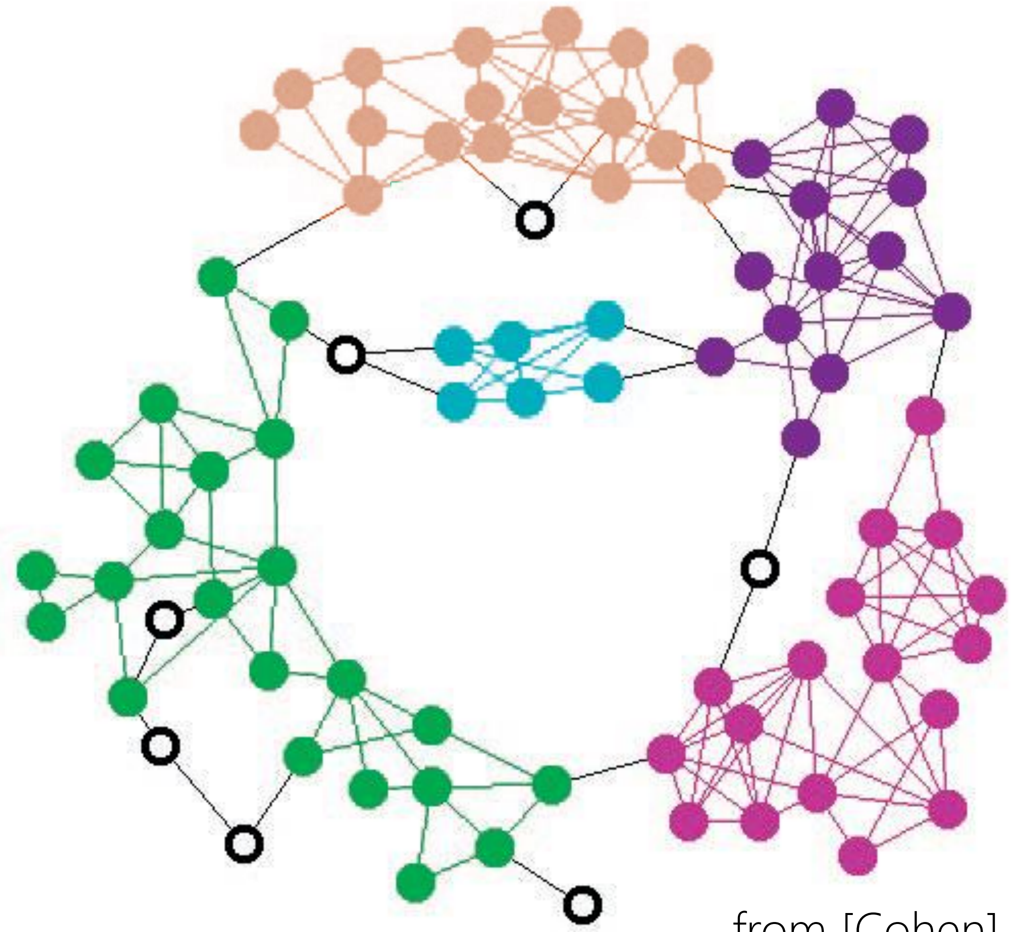
HPI Hasso Plattner Institut

Problem

Finding highly connected sub-graphs

- Why is this important?
  - Social media graphs: groups of friends/family/co-workers
  - Website interliking

- Why is this difficult?
  - Possible solution set size: $2^{|V|}$
  - → Exponential runtime for naive approach
  - Often millions of vertices
  - → ☹



from [Cohen]

The Data

Wikipedia

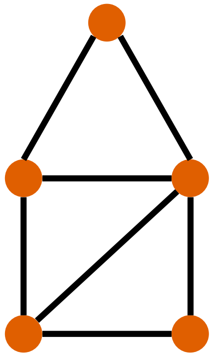
- Directed graph of English Wikipedia page interlinks from 2007
- ~1.9 million vertices, ~40 million edges, 1 GB size on disc

- Bidirectional version with 3.4 million edges, 54 MB

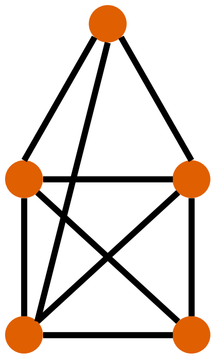
k-Trusses

k-Truss

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



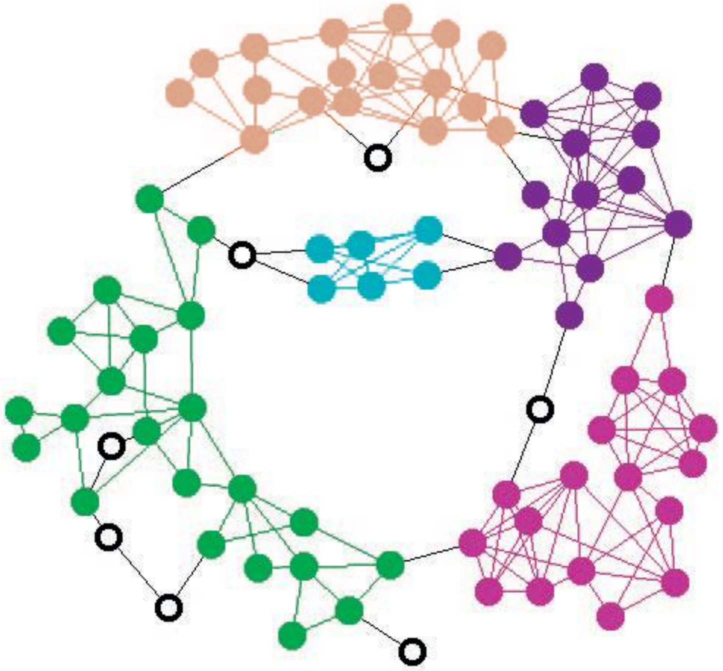
a 3-truss



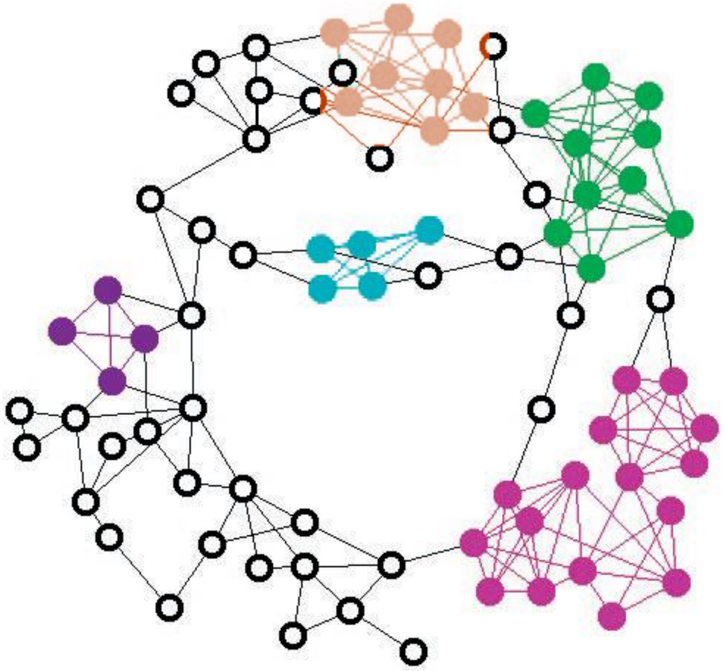
a 4-truss

k-Trusses

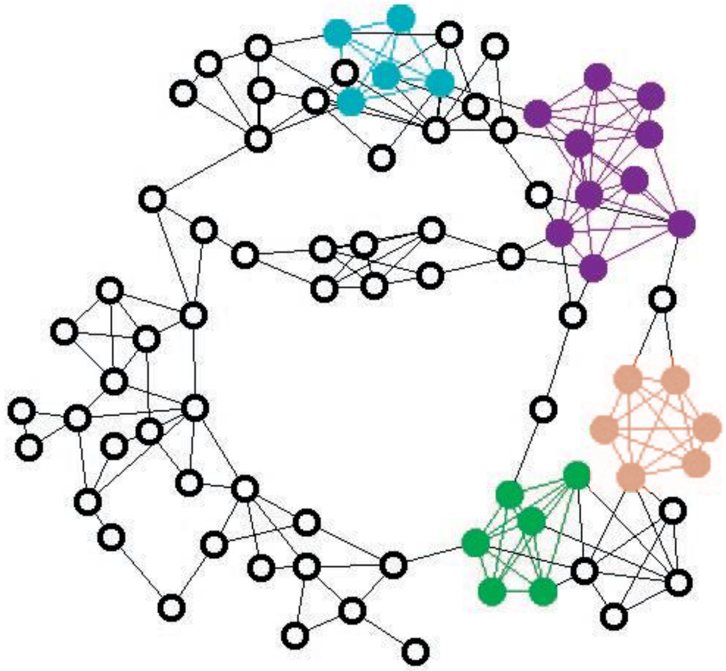
k = 3



k = 4



k = 5

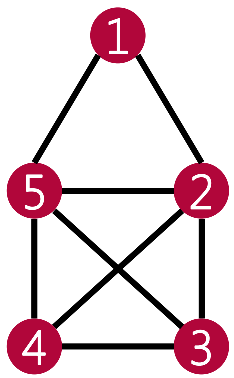


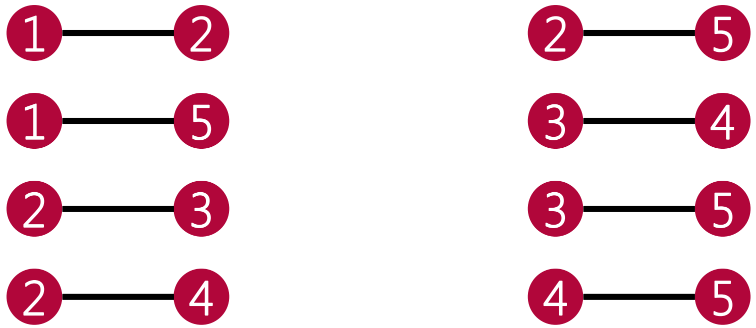
from [Cohen]

Implementation – Find Truss

Read graph

Find 4-truss





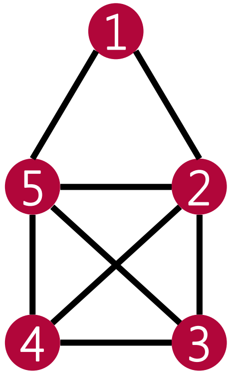
Implementation – Find Truss

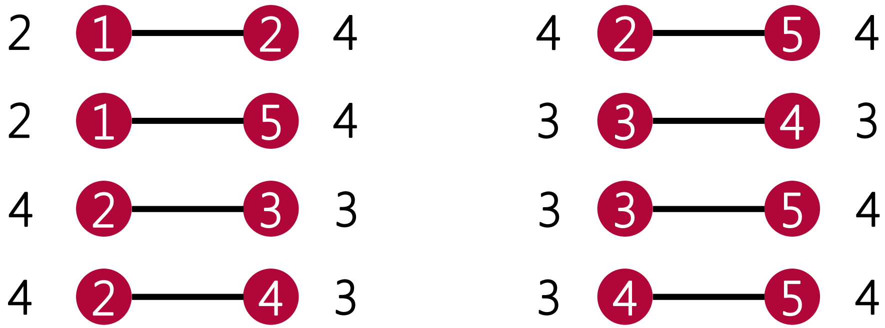
Read graph

↓

Add degrees

Find 4-truss





Implementation – Find Truss

Read graph

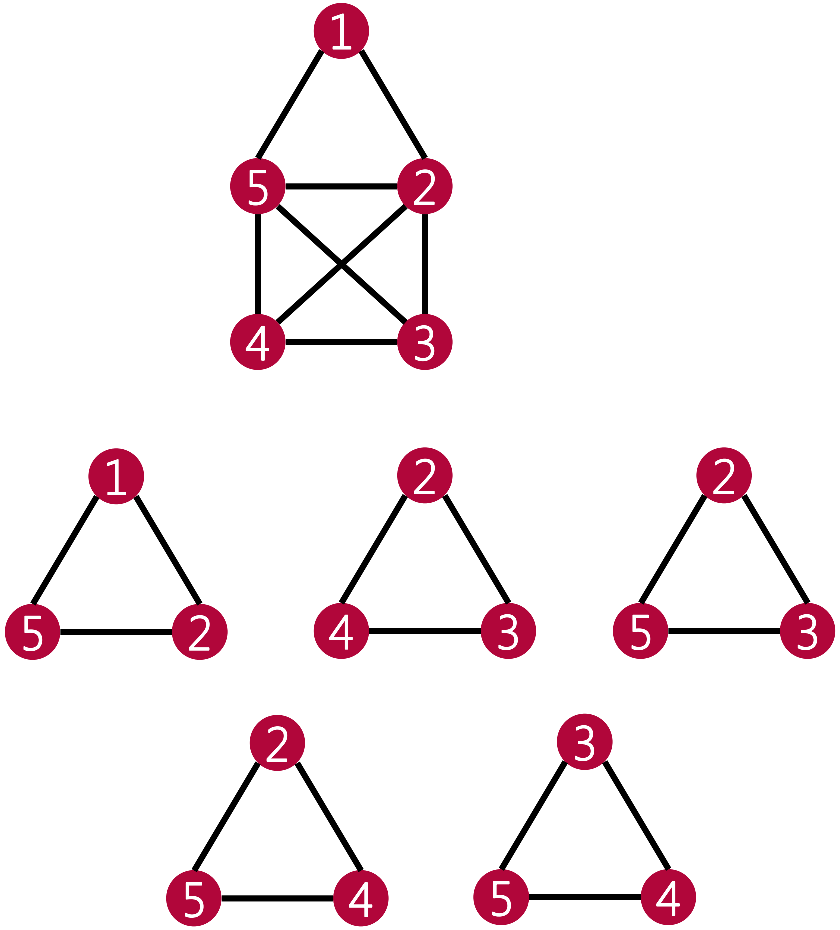
↓

Add degrees

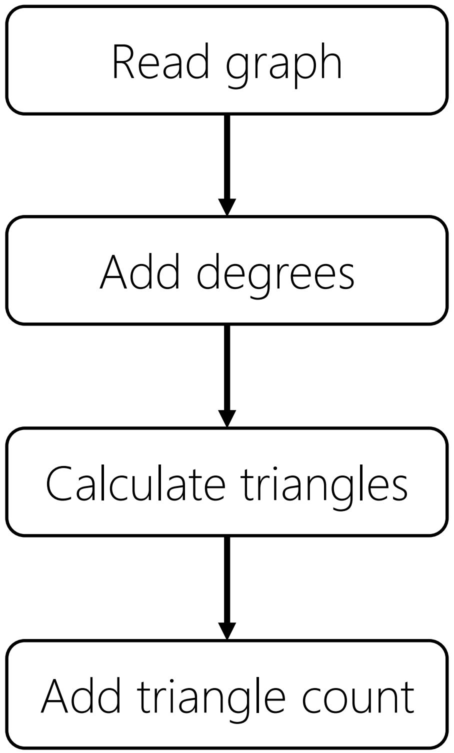
↓

Calculate triangles

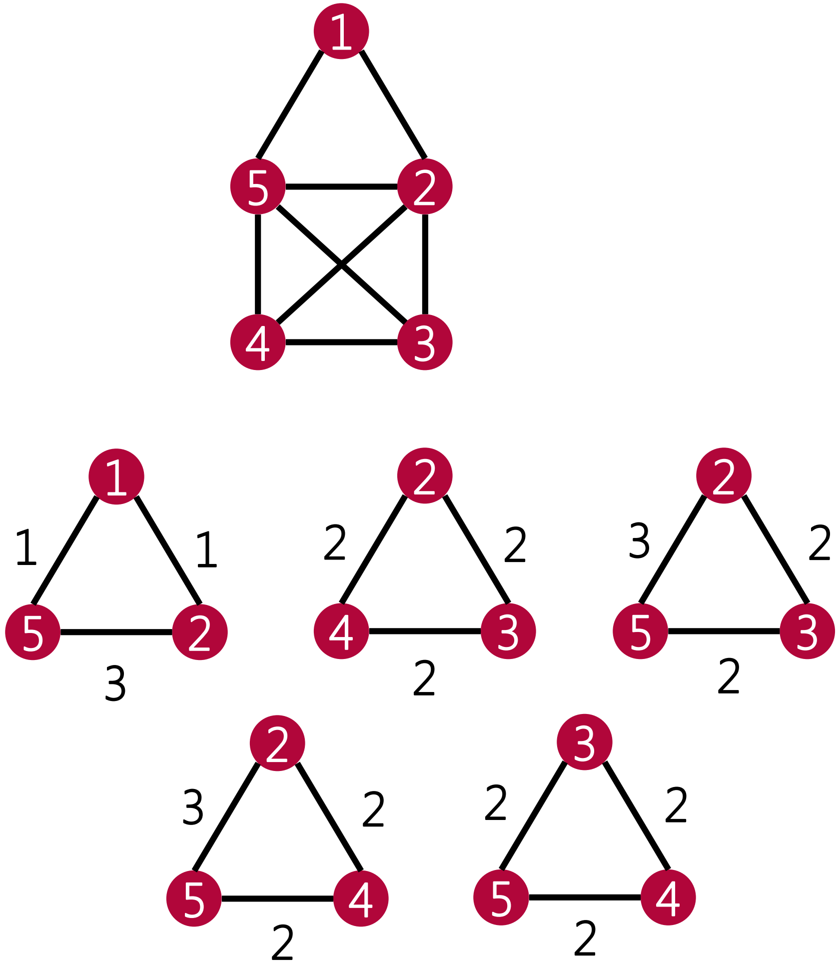
Find 4-truss



Implementation – Find Truss



Find 4-truss



Implementation – Find Truss

Read graph

↓

Add degrees

↓

Calculate triangles

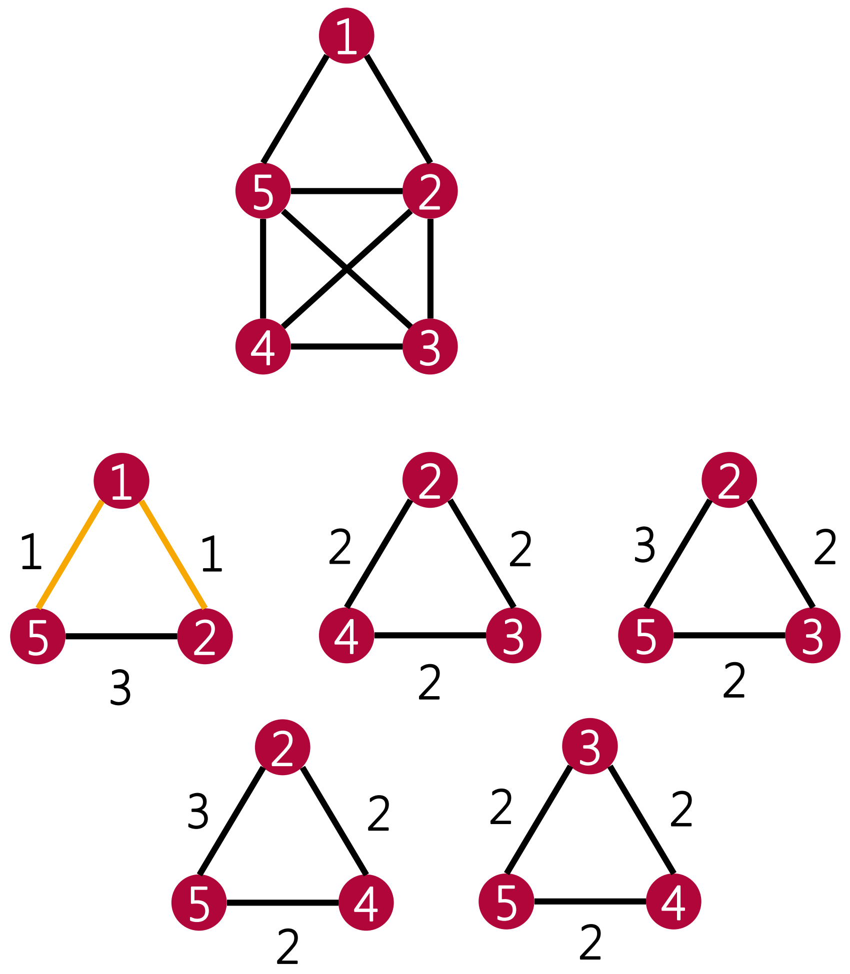
↓

Add triangle count

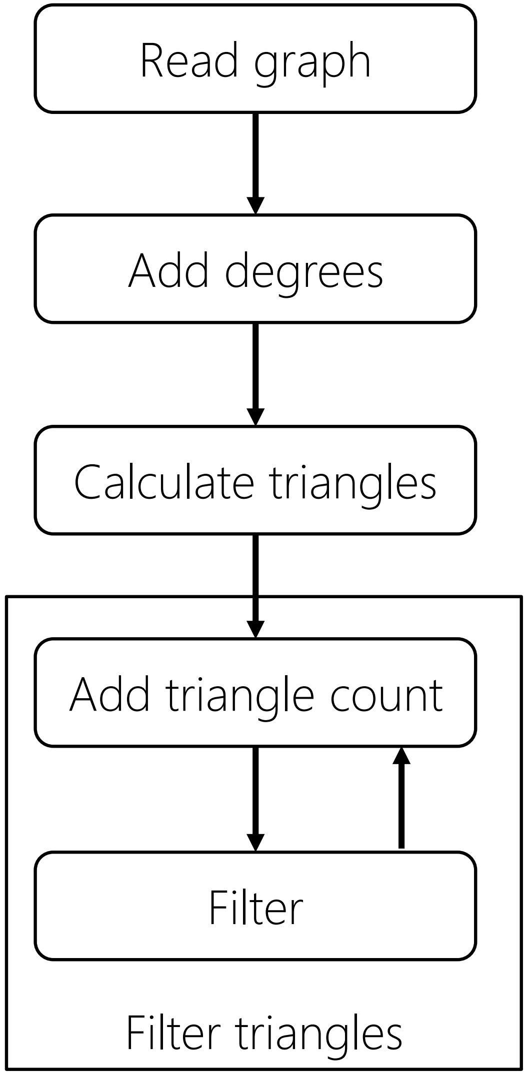
↓

Filter

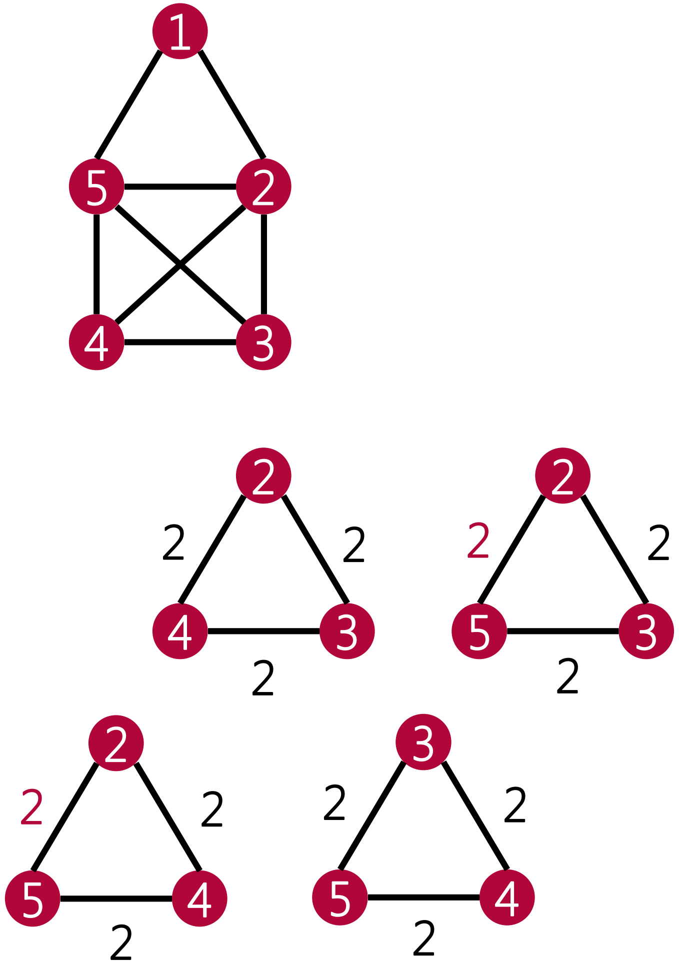
Find 4-truss



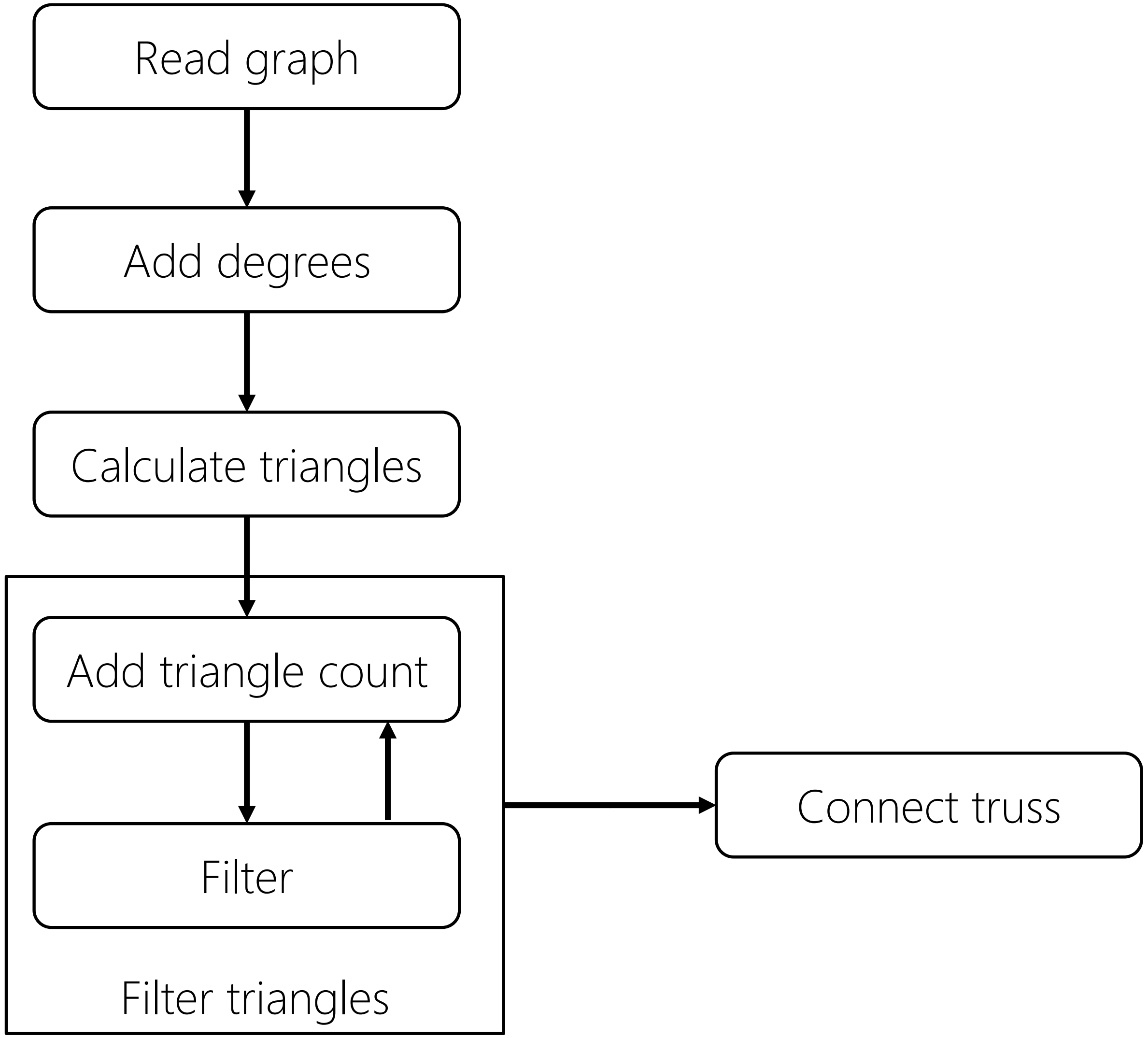
Implementation – Find Truss



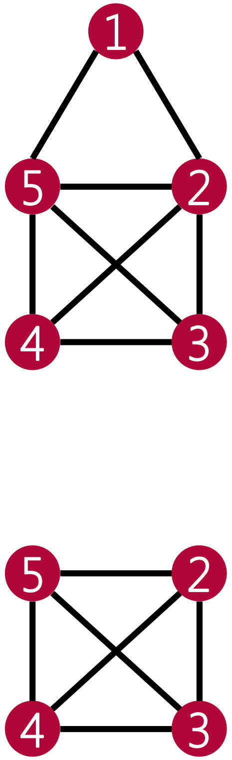
Find 4-truss



Implementation – Find Truss

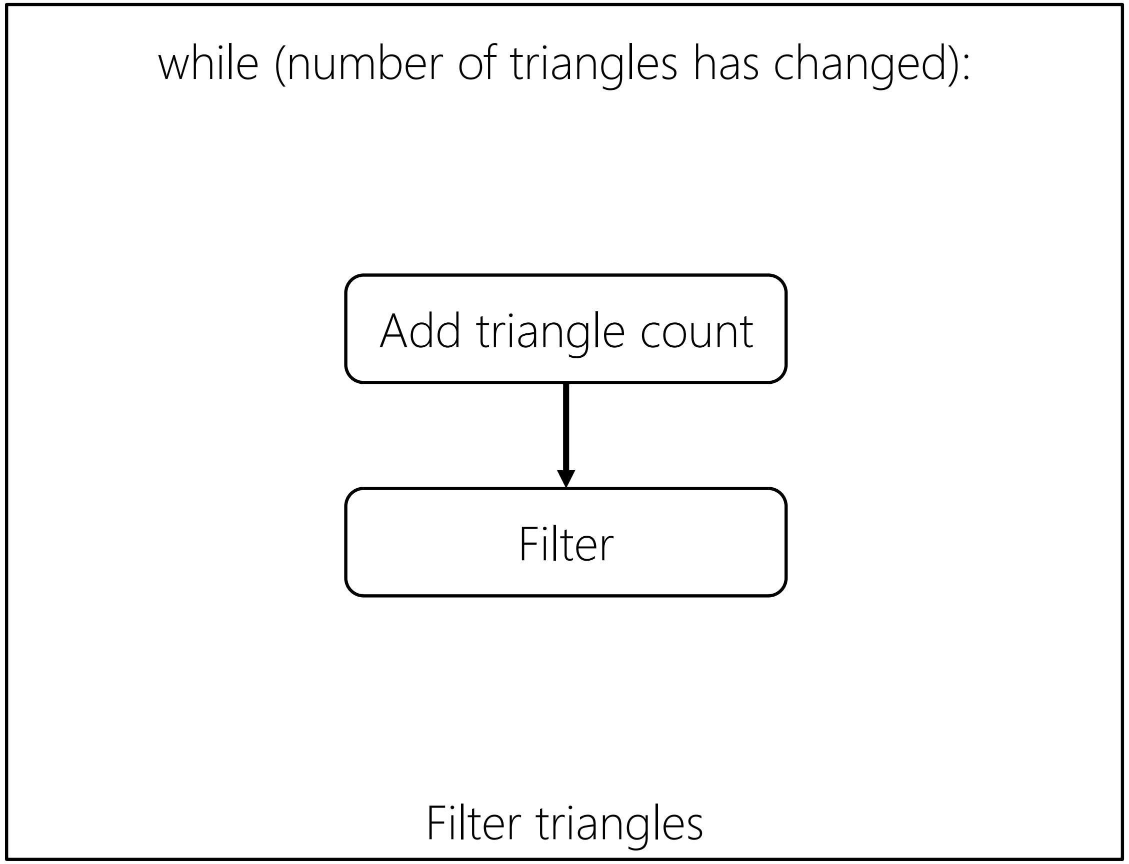


Find 4-truss

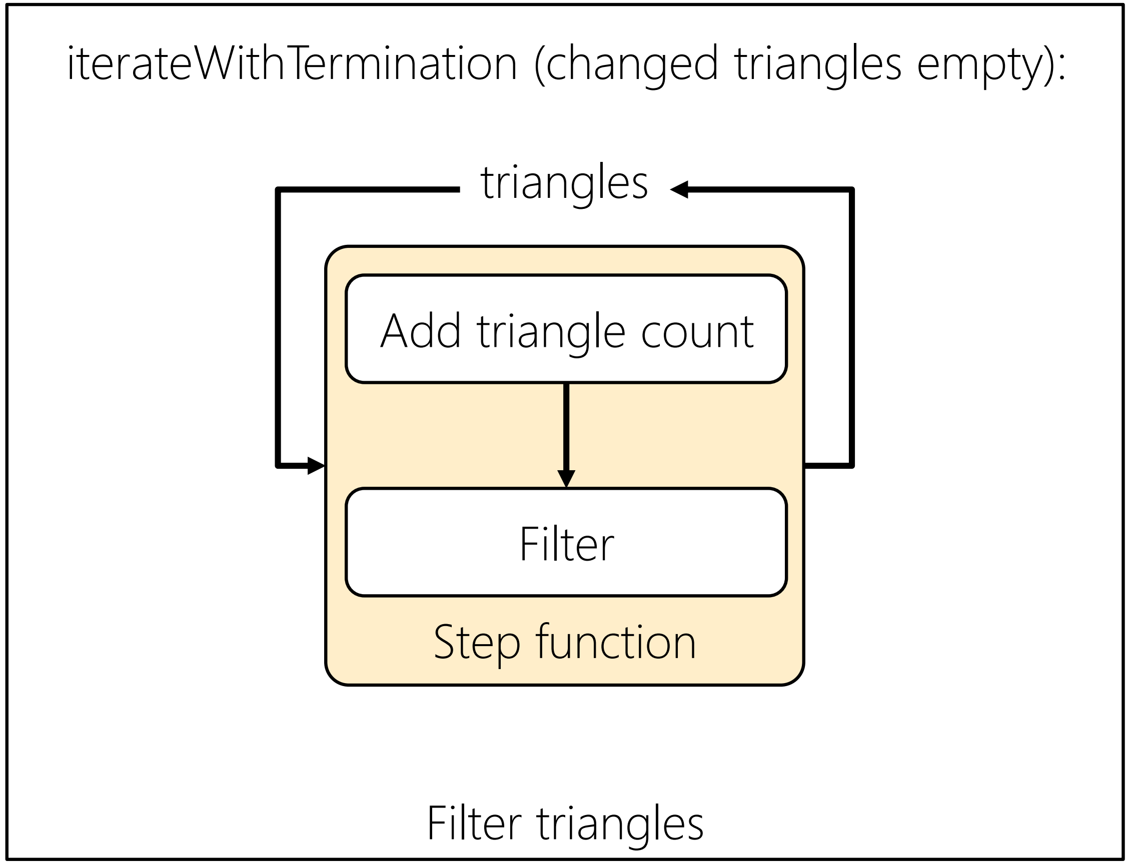


Implementation – Filter Triangles

Spark



Flink



Implementation – Max Truss

Finding the most dense truss in a graph

- Find a k-truss with user-defined k
- If at least one truss was found
  - set new k' > k
  - Search for new k'-trusses in the found trusses
- Else
  - Set new k' < k
  - Search for new k'-trusses in previous graph
- Repeat until a truss is found at k but none at k+1

- k is increased and decreased according to a binary search strategy

Implementation – Max Truss

Finding the most dense truss in a graph

- Find a k-truss with user-defined k
- If at least one truss was found
  - set new k' > k
  - Search for new k'-trusses in the found trusses
- Else
  - Set new k' < k
  - Search for new k'-trusses in previous graph
- Repeat until a truss is found at k but none at k+1

- k is increased and decreased according to a binary search strategy

True Max Truss at k = 28
Initial k = 20

k = 20	Truss found
k = 40	No truss found
k = 30	No truss found
k = 25	Truss found
k = 27	Truss found
k = 28	Truss found
k = 29	No truss found

Done. k = 28

Implementation – Max Truss

Finding the most dense truss in a graph – Flink issues

- Find a k-truss with user-defined k
- If at least one truss was found
  - set new k' > k
  - Search for new k'-trusses in the found trusses
- Else
  - Set new k' < k
  - Search for new k'-trusses in previous graph
- Repeat until a truss is found at k but none at k+1 Flink Iteration?

- k is increased and decreased according to a binary search strategy

Implementation – Max Truss

Finding the most dense truss in a graph – Flink issues

- Find a k-truss with user-defined k
- If at least one truss was found
  - set new k' > k
  - Search for new k'-trusses in the found trusses
- Else
  - Set new k' < k
  - Search for new k'-trusses in previous graph
- Repeat until a truss is found at k but none at k+1

Flink Iteration? → No nested iterations!

→ while loop

- k is increased and decreased according to a binary search strategy

Implementation – Max Truss

Finding the most dense truss in a graph – Flink issues

- Find a k-truss with user-defined k
- If at least one truss was found requires count → Data Sink
  - set new k' > k
  - Search for new k'-trusses in the found trusses
- Else
  - Set new k' < k
  - Search for new k'-trusses in previous graph
- Repeat until a truss is found at k but none at k+1

- k is increased and decreased according to a binary search strategy

Implementation – Max Truss

Finding the most dense truss in a graph – Flink issues

- Find a k-truss with user-defined k
- If at least one truss was found   requires count → Data Sink
  - set new k' > k
  - Search for new k'-trusses in the found trusses
- Else
  - Set new k' < k
  - Search for new k'-trusses in previous graph
- Repeat until a truss is found at k but none at k+1

Solution 1:
- Accept that we lose previous result
- Recalculate with new k on initial graph

Solution 2:
- Write previous result to disk
- Read at the start of every new iteration

- k is increased and decreased according to a binary search strategy

Implementation – Max Truss

Finding the most dense truss in a graph – Flink issues

- Find a k-truss with user-defined k
- If at least one truss was found   requires count → Data Sink
  - set new k' > k
  - Search for new k'-trusses in the found trusses
- Else
  - Set new k' < k
  - Search for new k'-trusses in previous graph
- Repeat until a truss is found at k but none at k+1

Solution 1:
- Accept that we lose previous result
- Recalculate with new k on initial graph

Solution 2:
- Write previous result to disk
- Read at the start of every new iteration

Speedup factor = ~3

- k is increased and decreased according to a binary search strategy

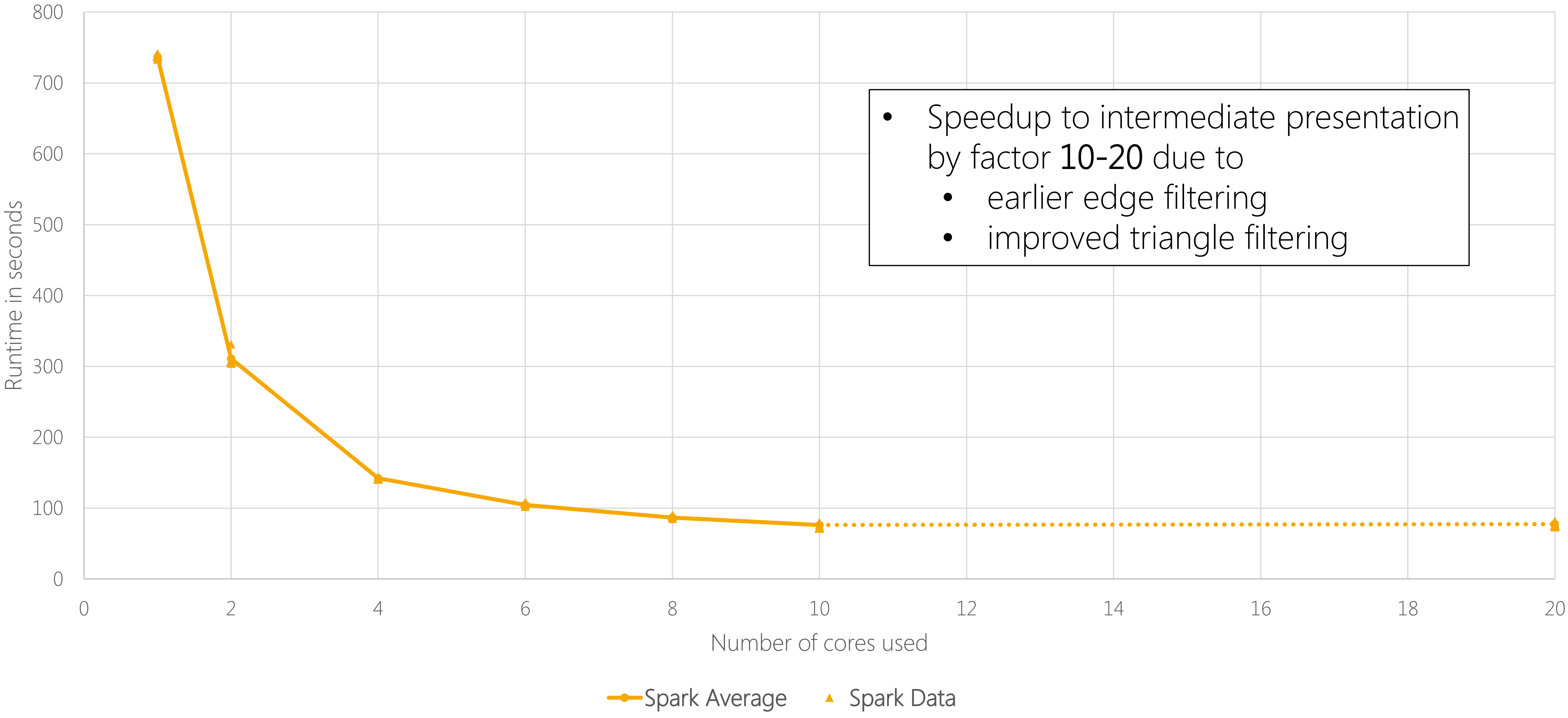
Evaluation – Conditions

HPI IS chair cluster

- Master: 1 Dell PowerEdge R310
  - 4(8)x2.66 GHz
  - 8 GB DDR3 RAM
- Slaves: 10 Dell OptiPlex 780
  - 2x2.6 GHz
  - using 4 GB DDR3 RAM

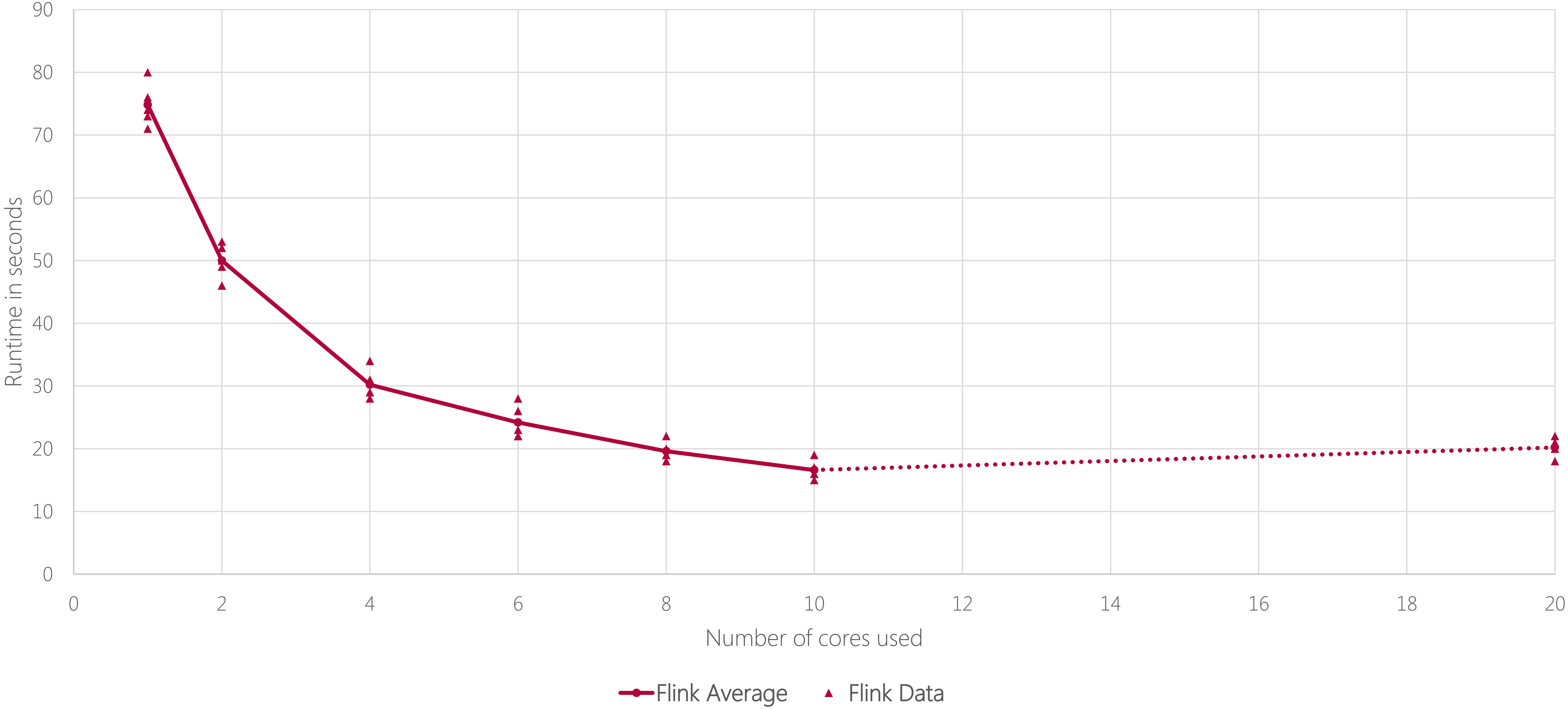
- Using only 1 core per slave
- k value of 20
- Bidirectional Wikipedia data set
- Averaged 5 measurements
  (Unless otherwise noted)

Evaluation – Spark scaling by #cores

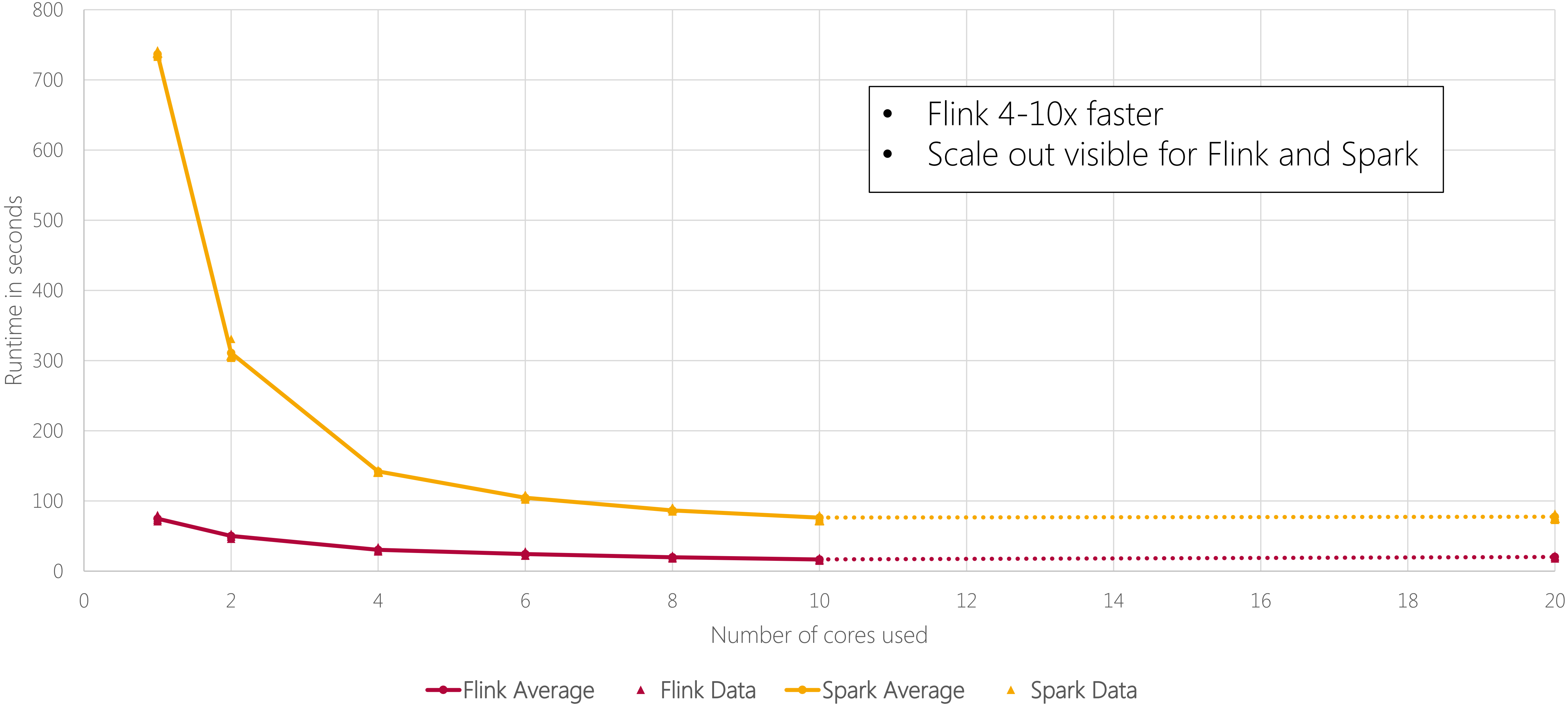


- Speedup to intermediate presentation by factor **10-20** due to
  - earlier edge filtering
  - improved triangle filtering

Evaluation – Flink scaling by #cores

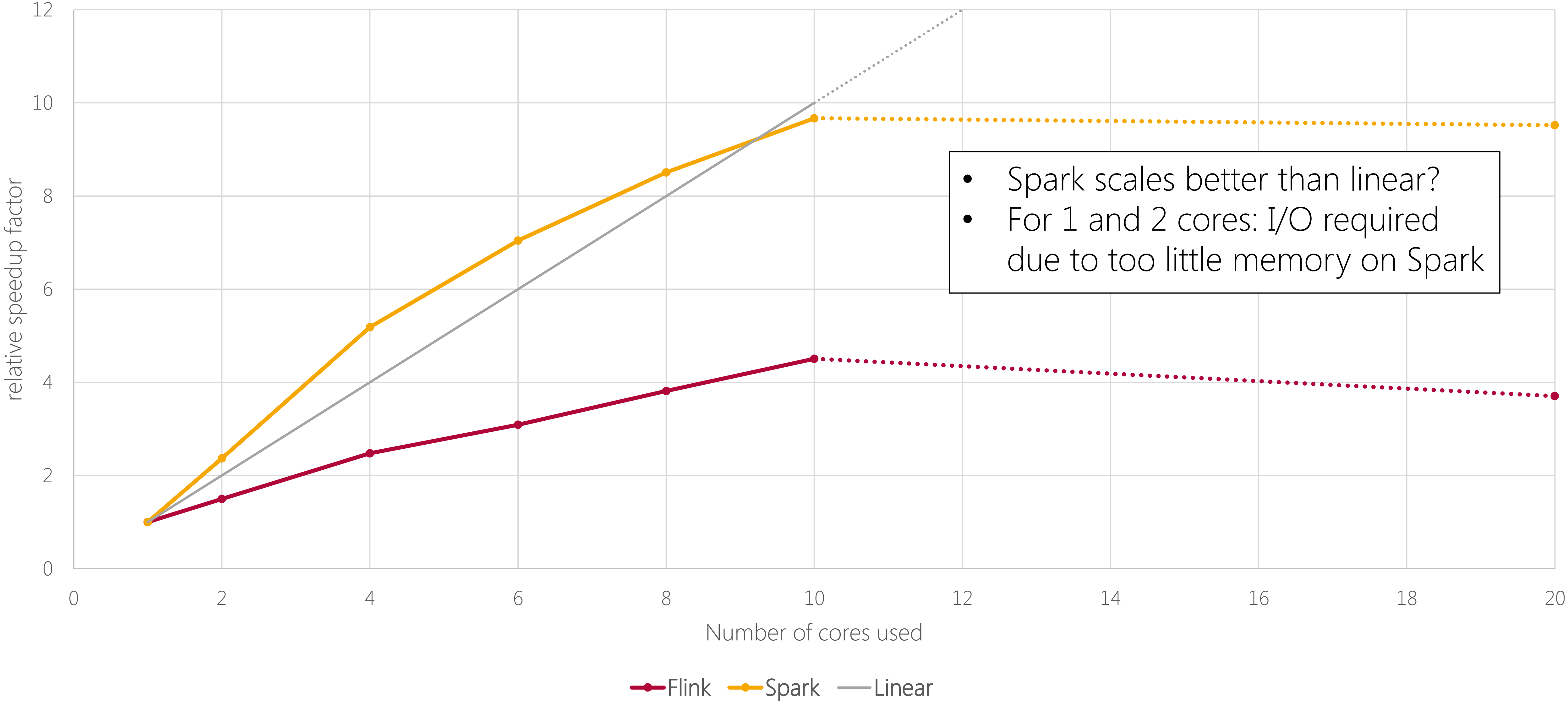


Evaluation – Flink vs Spark



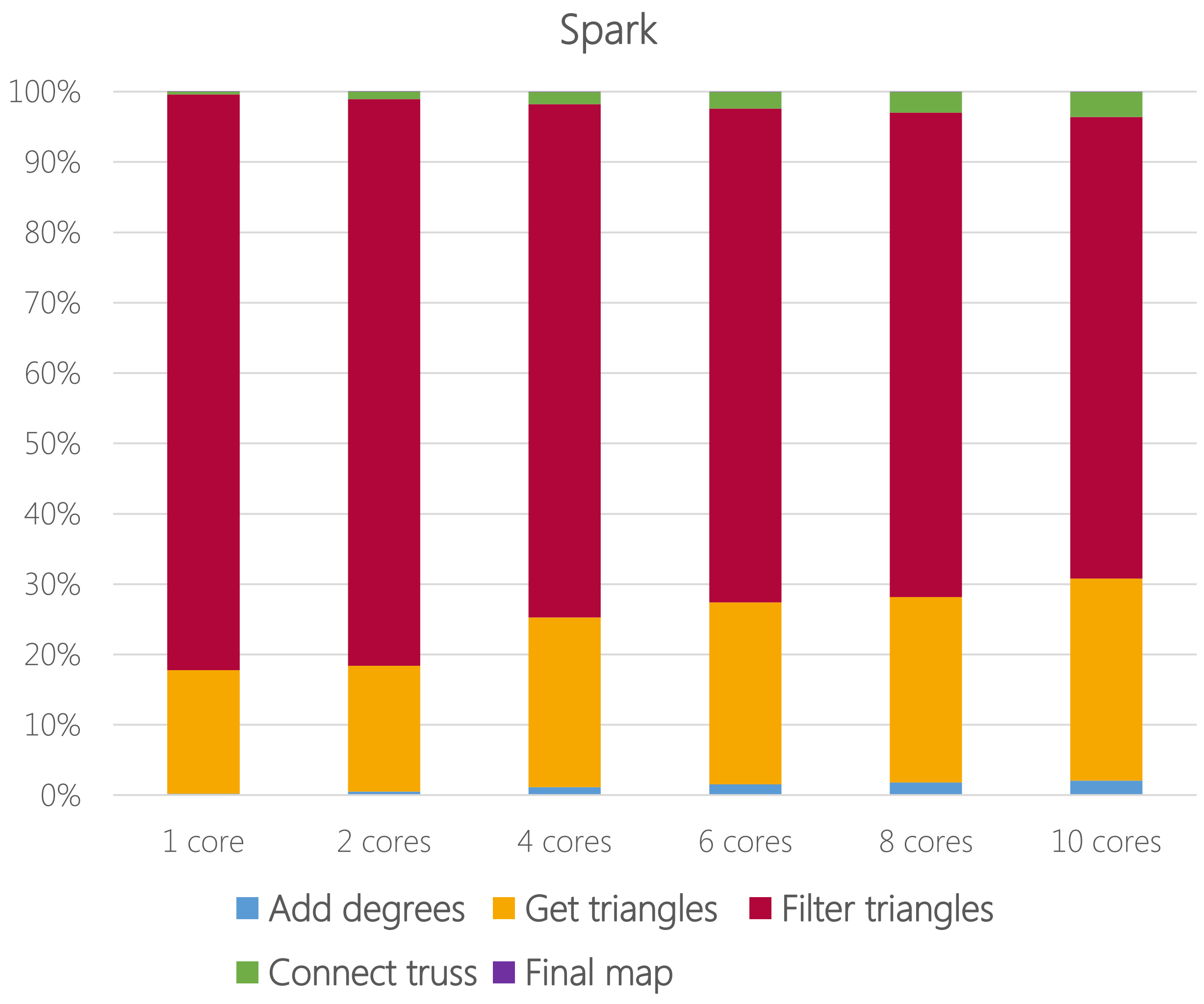
- Flink 4-10x faster
- Scale out visible for Flink and Spark

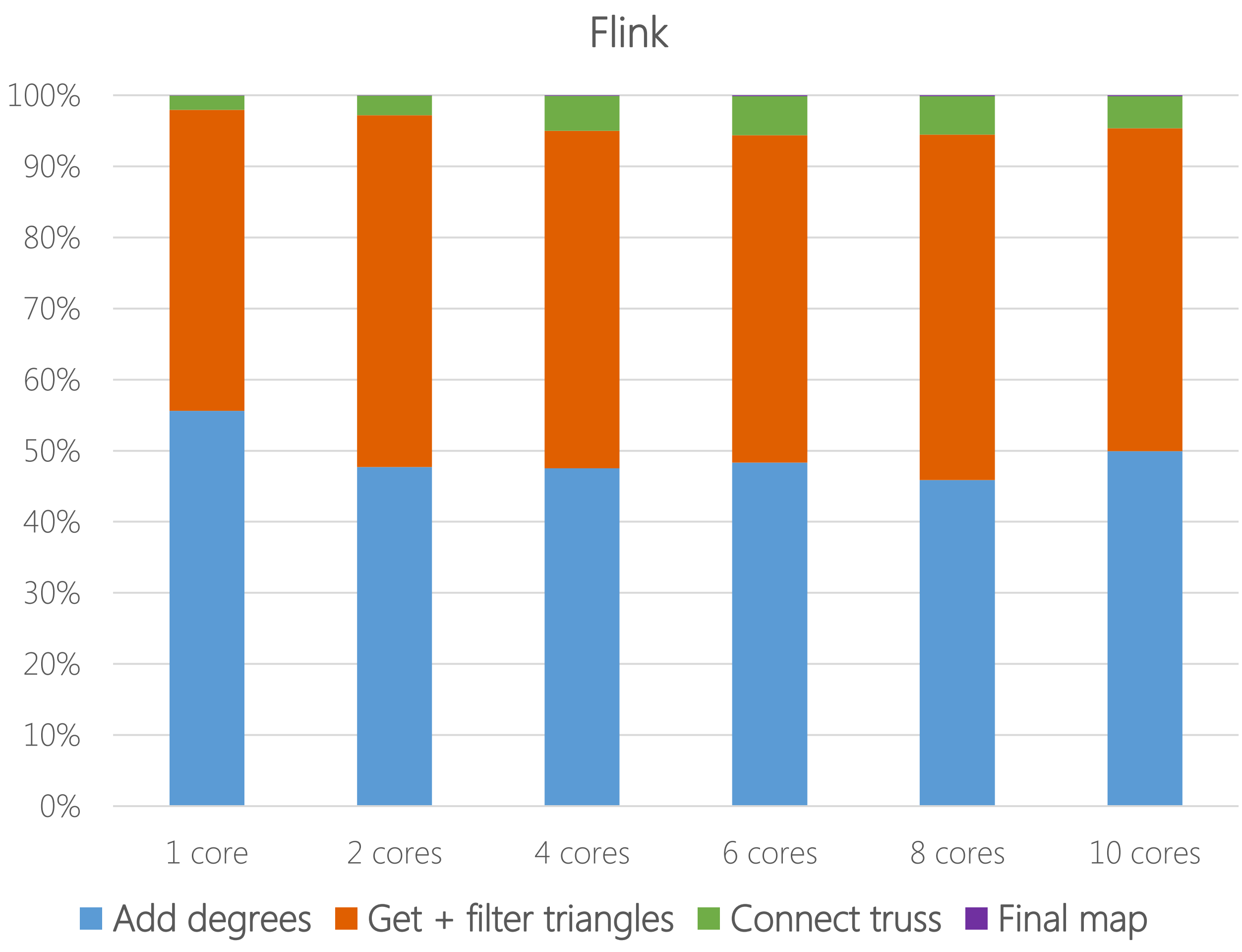
Evaluation – Relative Speedup



- Spark scales better than linear?
- For 1 and 2 cores: I/O required due to too little memory on Spark

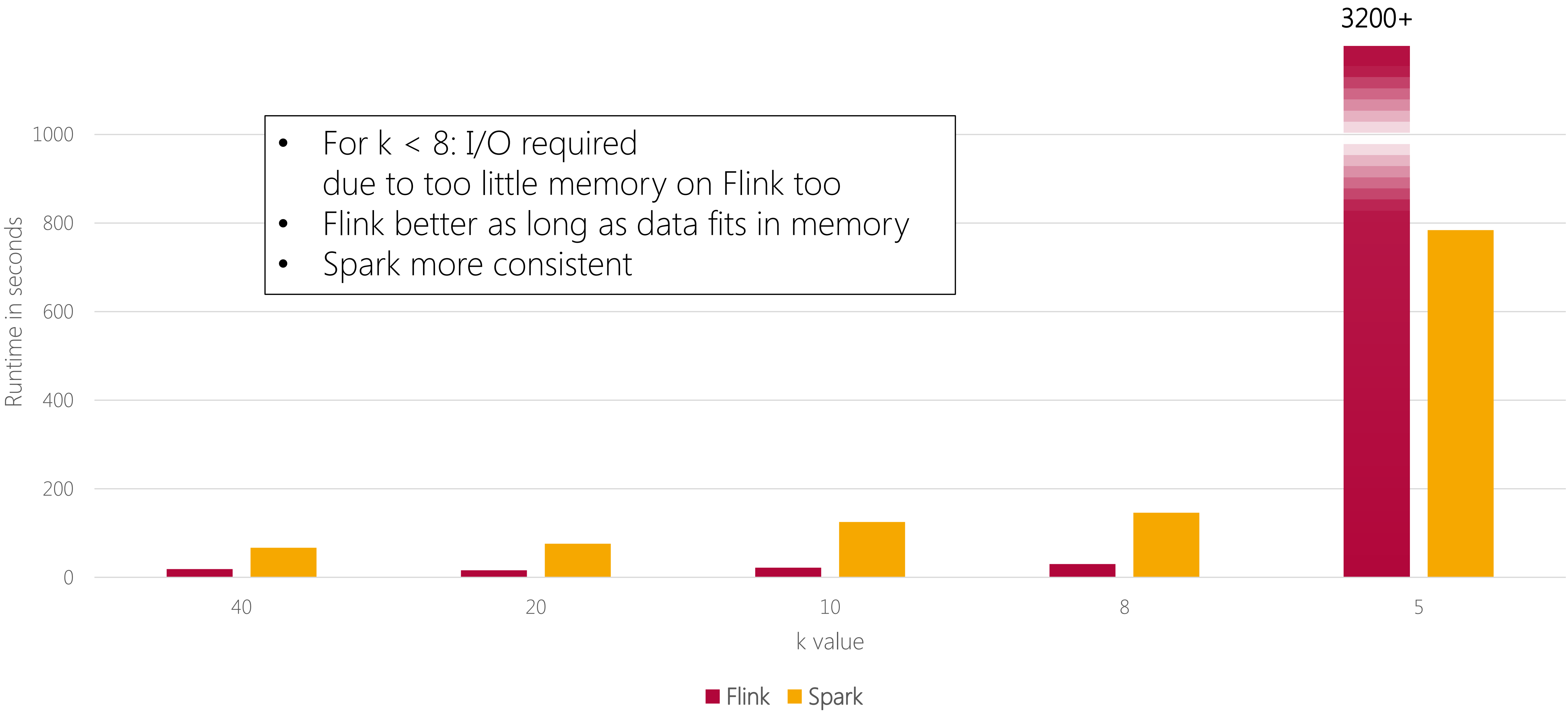
Evaluation – Program Parts



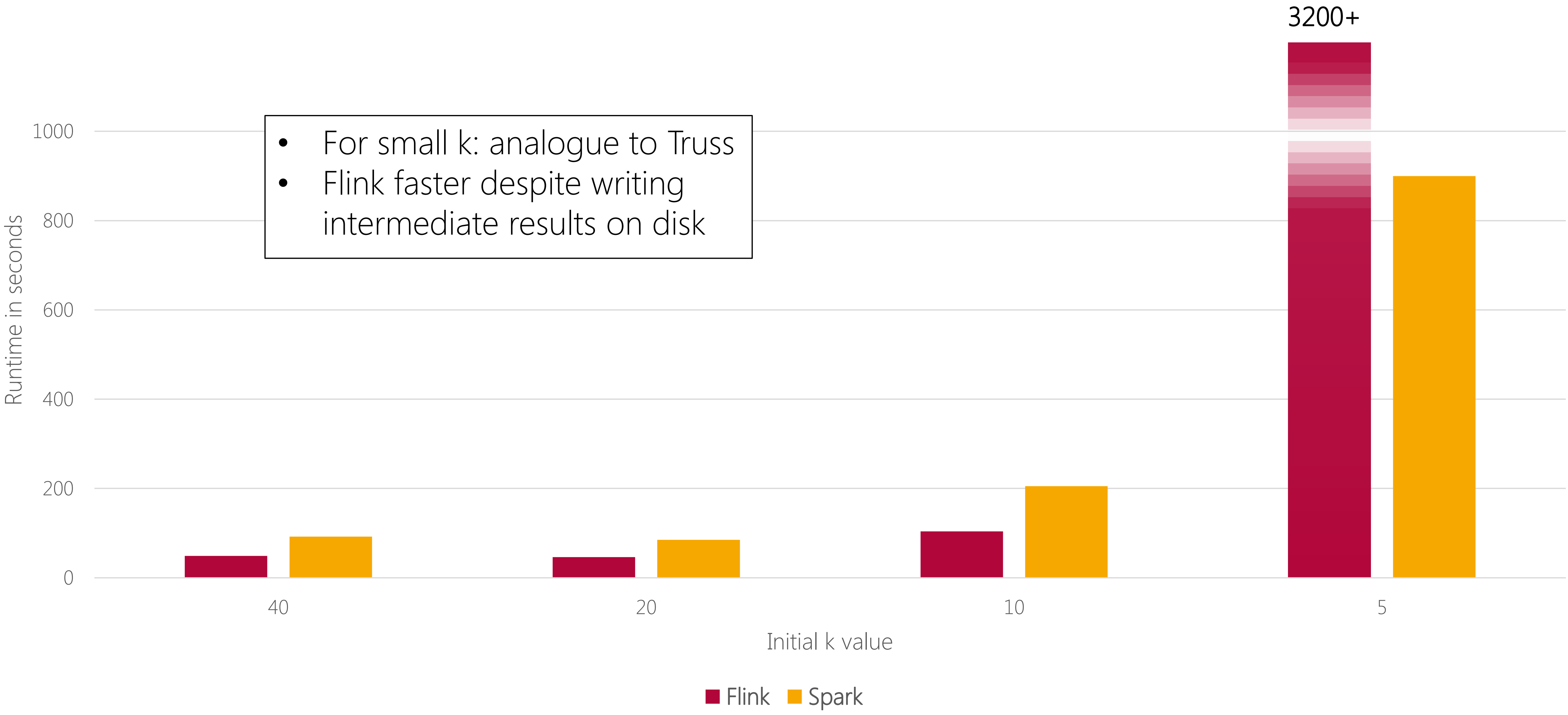


Evaluation – Truss

- For k < 8: I/O required
  due to too little memory on Flink too
- Flink better as long as data fits in memory
- Spark more consistent



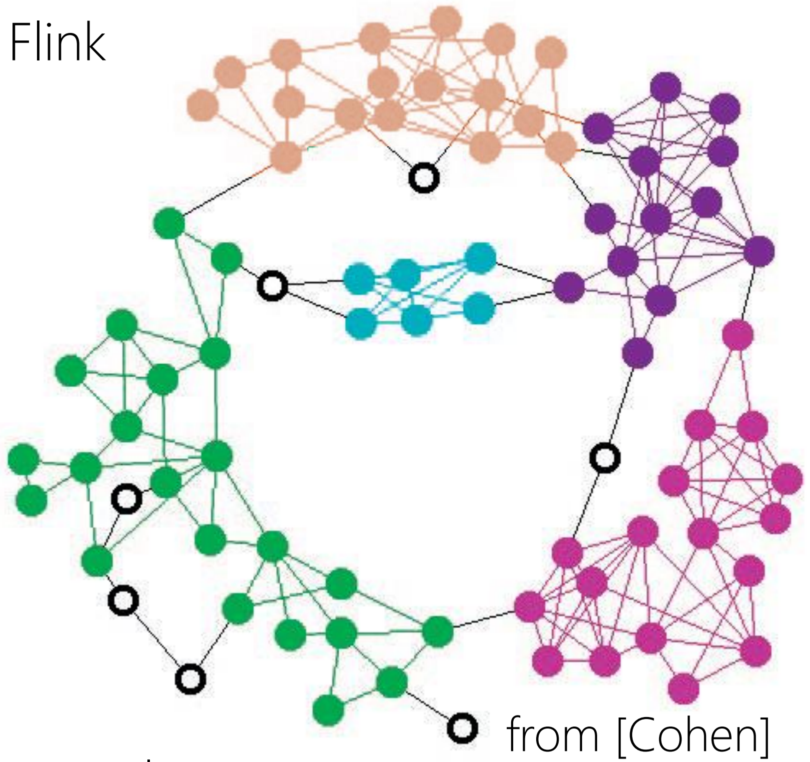
Evaluation – Maximal Truss



- For small k: analogue to Truss
- Flink faster despite writing intermediate results on disk

Conclusions

- Distributed Calculation
  - Great scaling with distribution over multiple machines for Spark and Flink
- Flink
  - Can hold more data in main memory due to different serialization
  - As long as the data fits in main memory, Flink is also faster
  - Speedup due to more efficient iterations
  - Could be improved even more by handling nested iterations
- Spark
  - Can deal much better with full main memory due to improved user control
  - More consistent and thus predictable performance



from [Cohen]

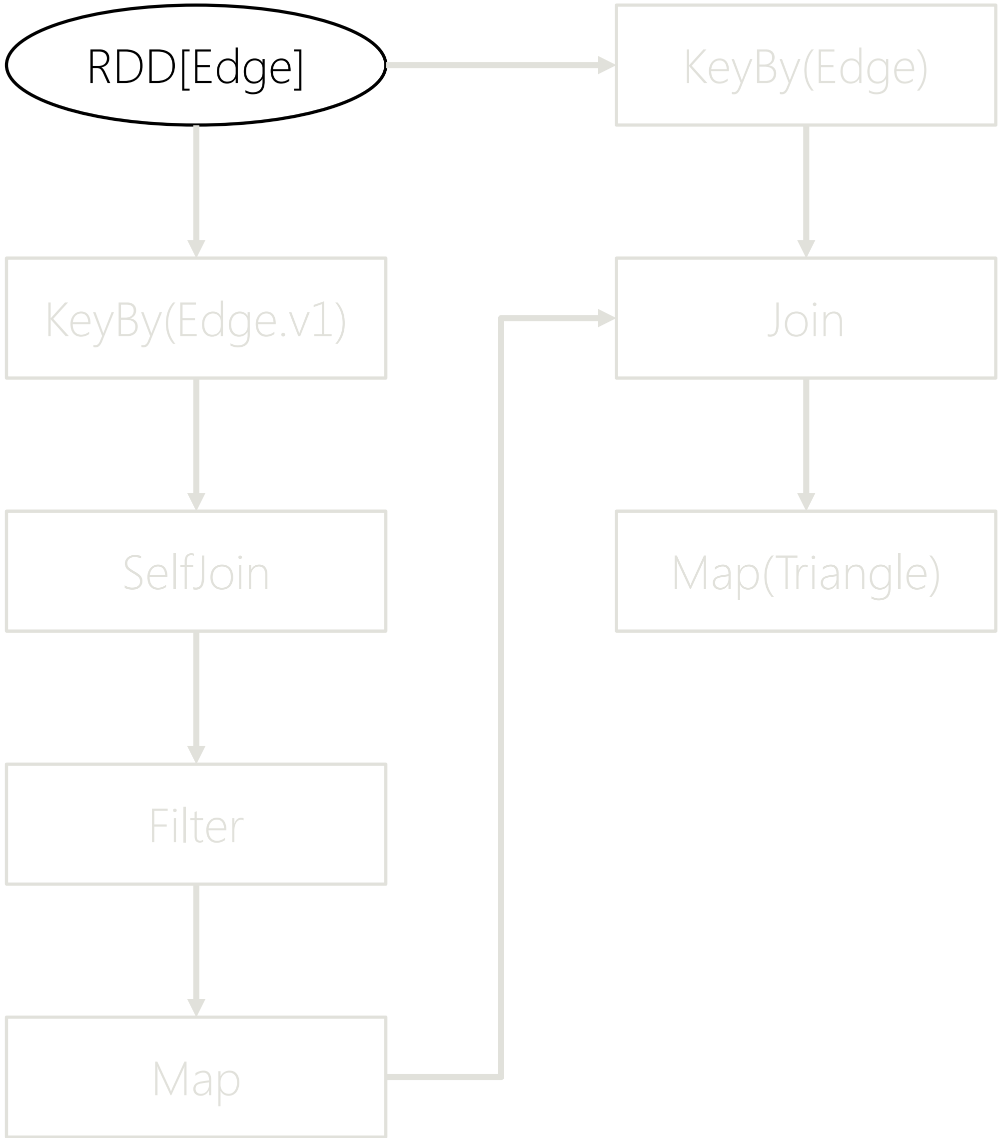
References

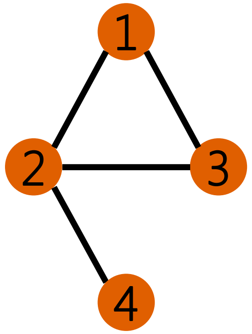
Image on title slide: http://polkadotimpressions.com/2013/01/18/facebook-graph-search-3/

[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.

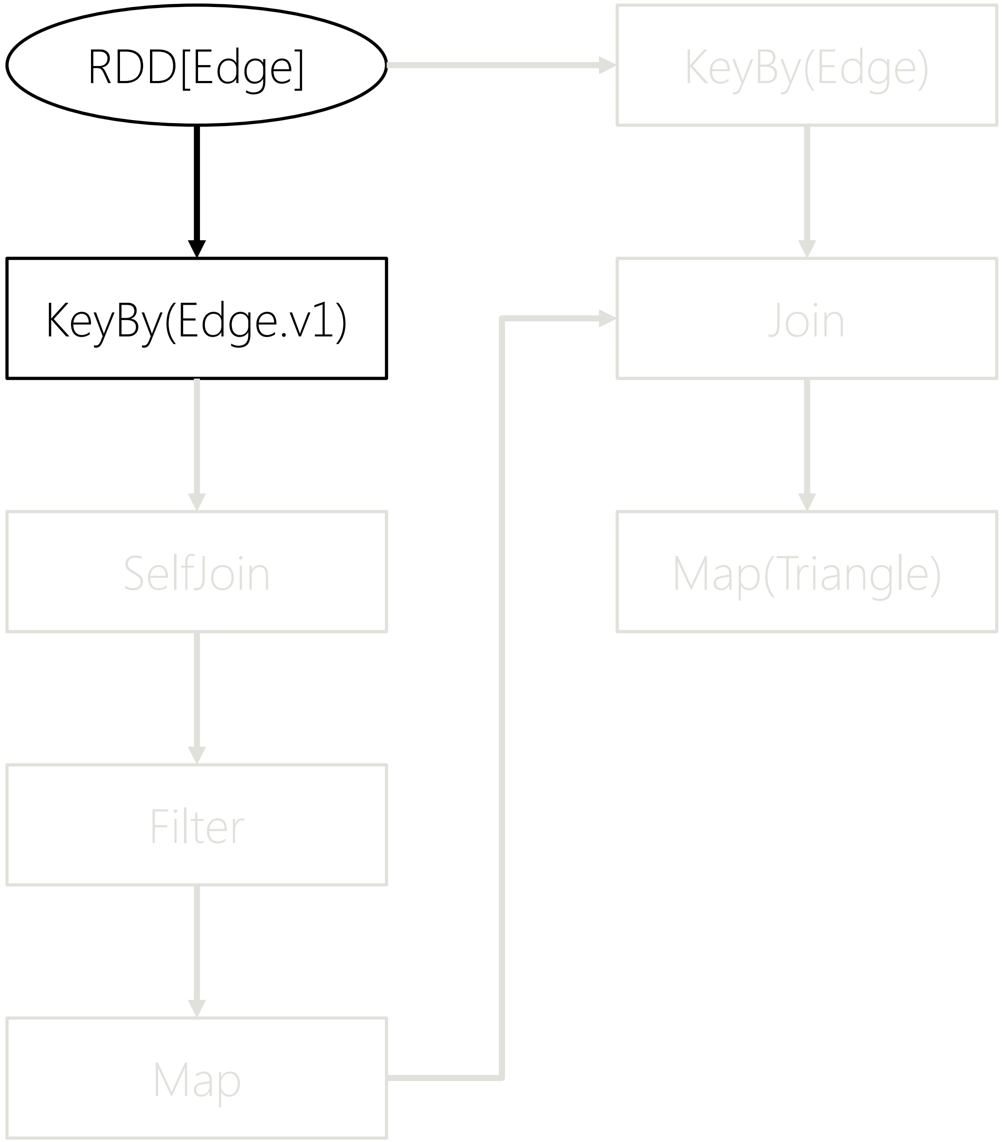
[Cohen]: Jonathan Cohen, 'Graph Twiddling in a MapReduce World'. in *Computing in Science and Engineering* 11(4): 29-41 (2009)

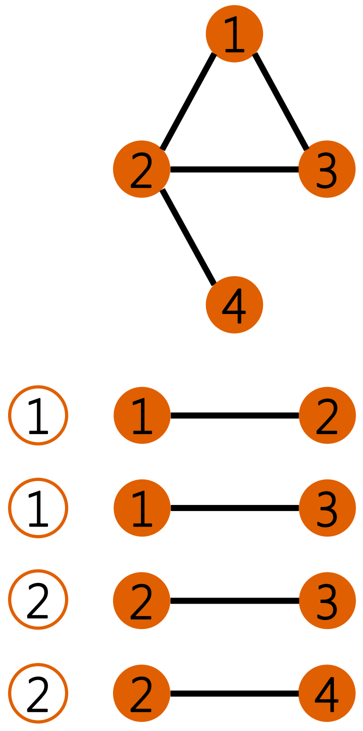
Implementation – Triangle Generation



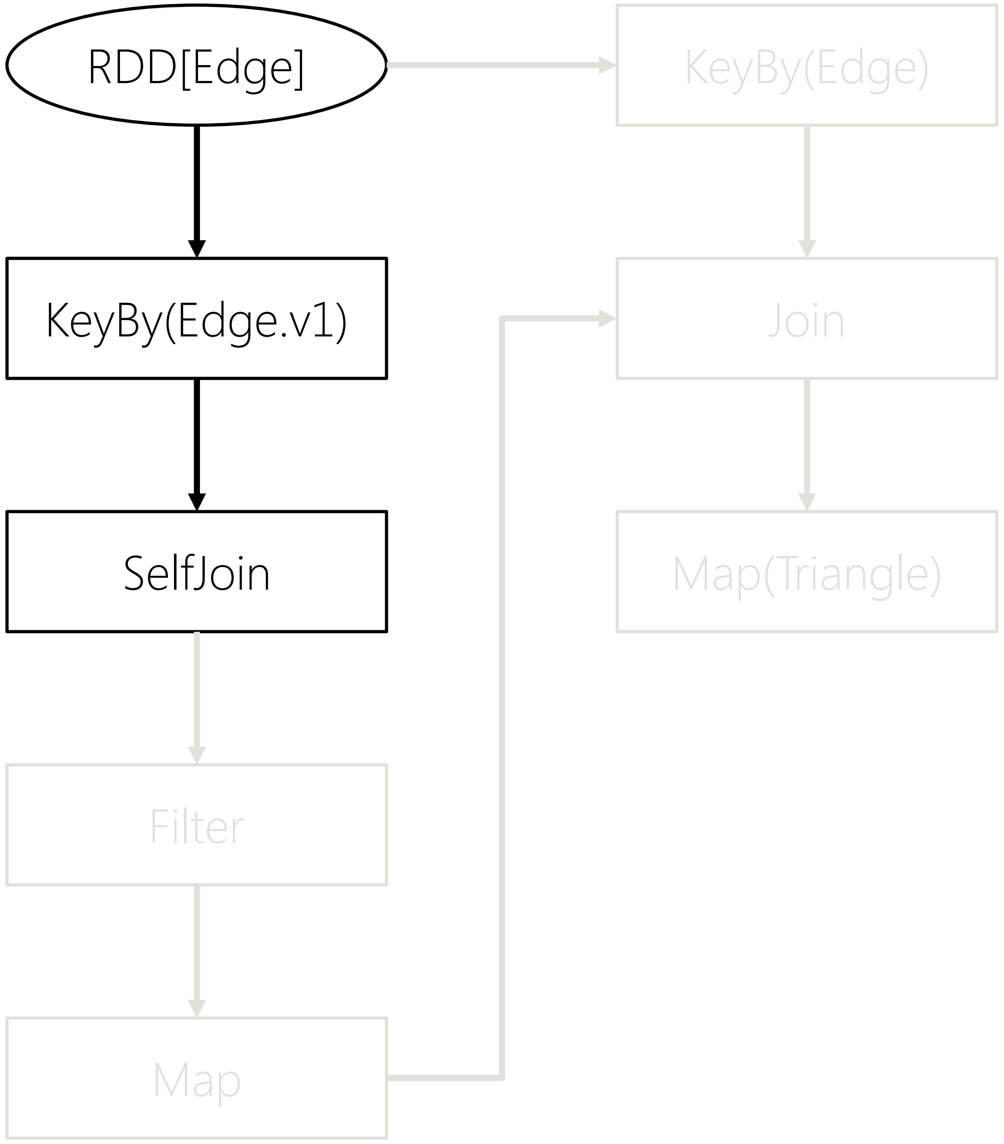


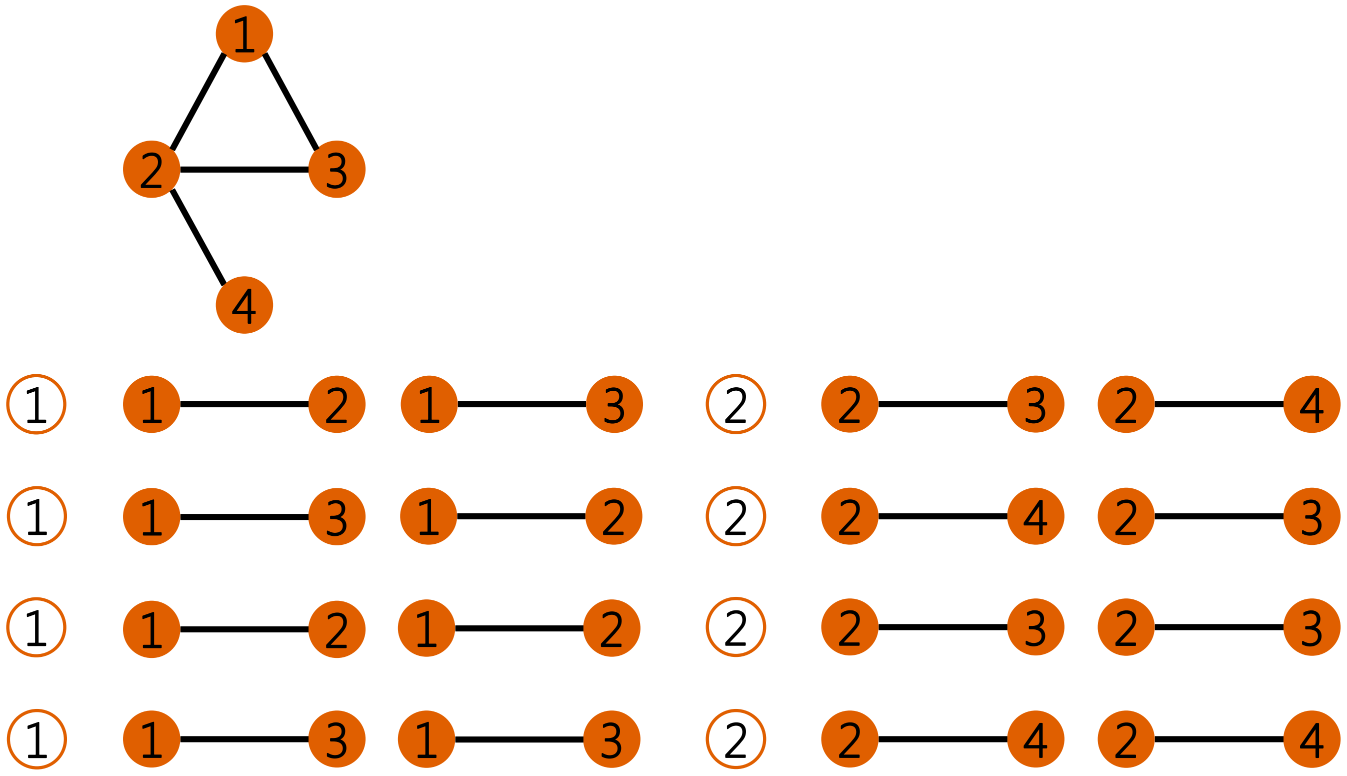
Implementation – Triangle Generation



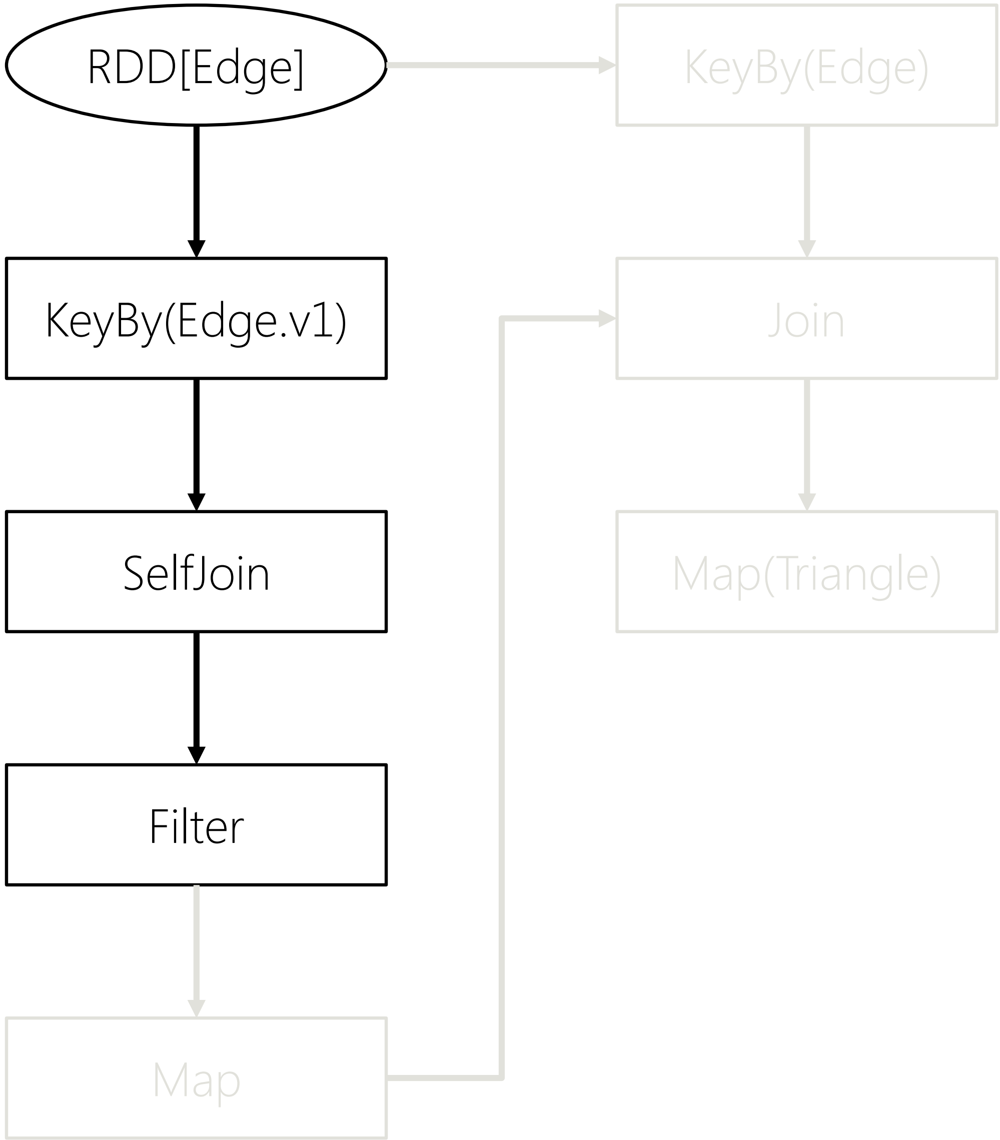


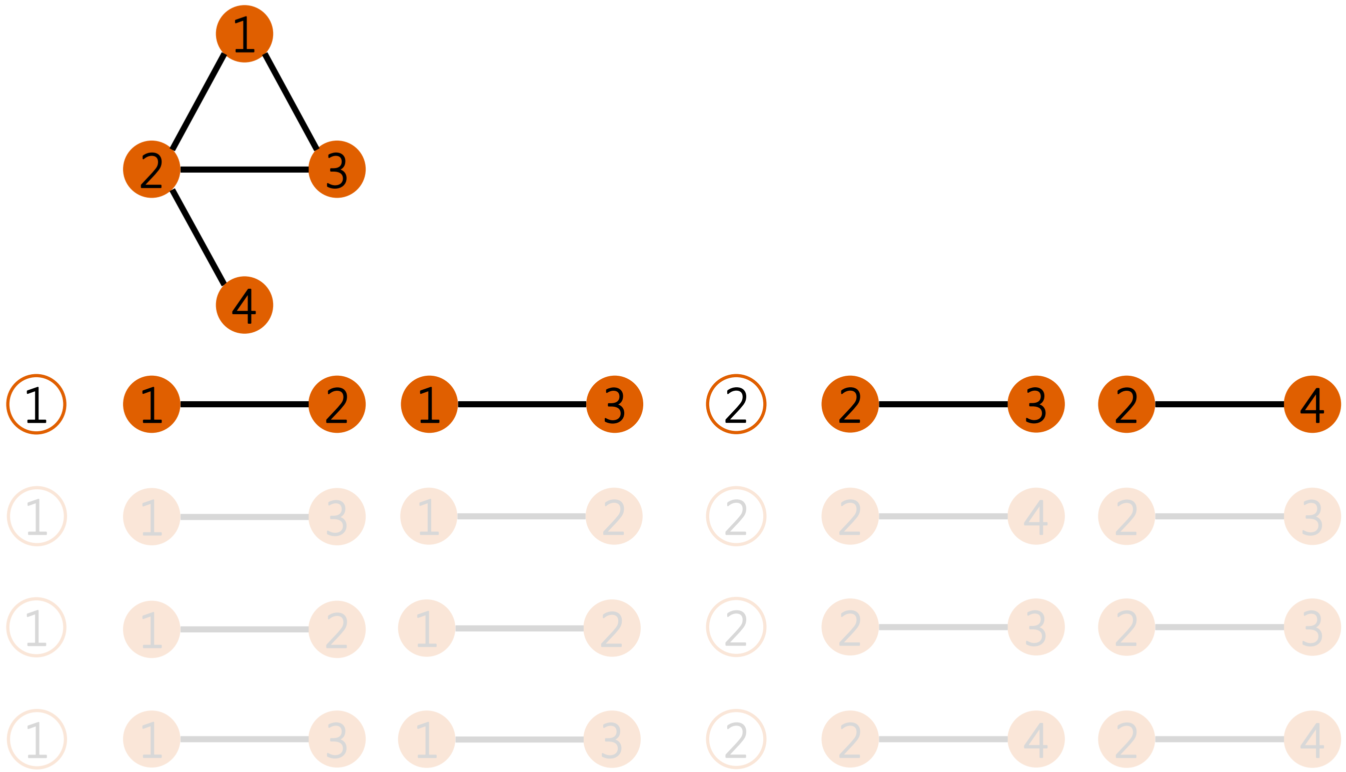
Implementation – Triangle Generation



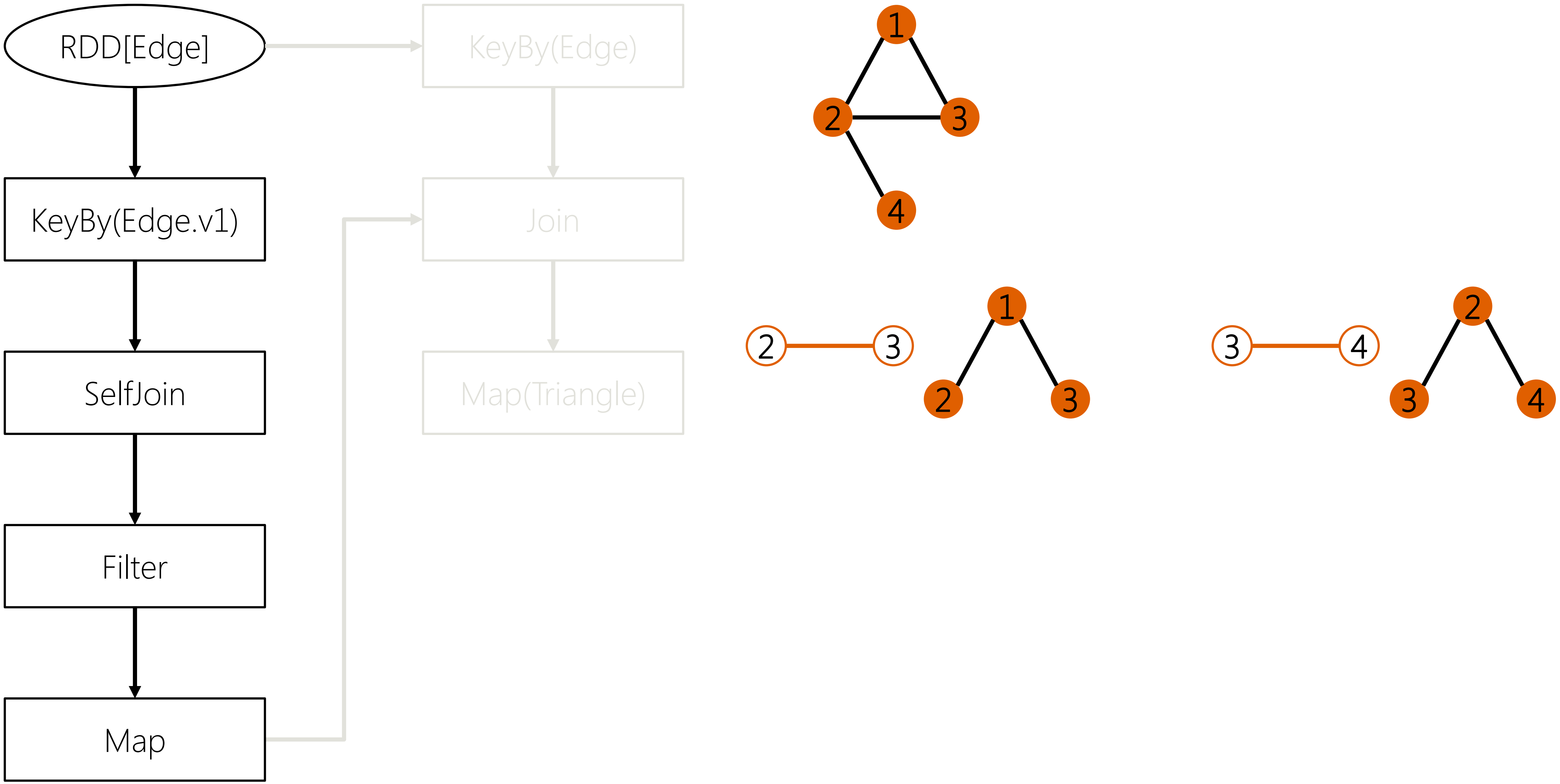


Implementation – Triangle Generation

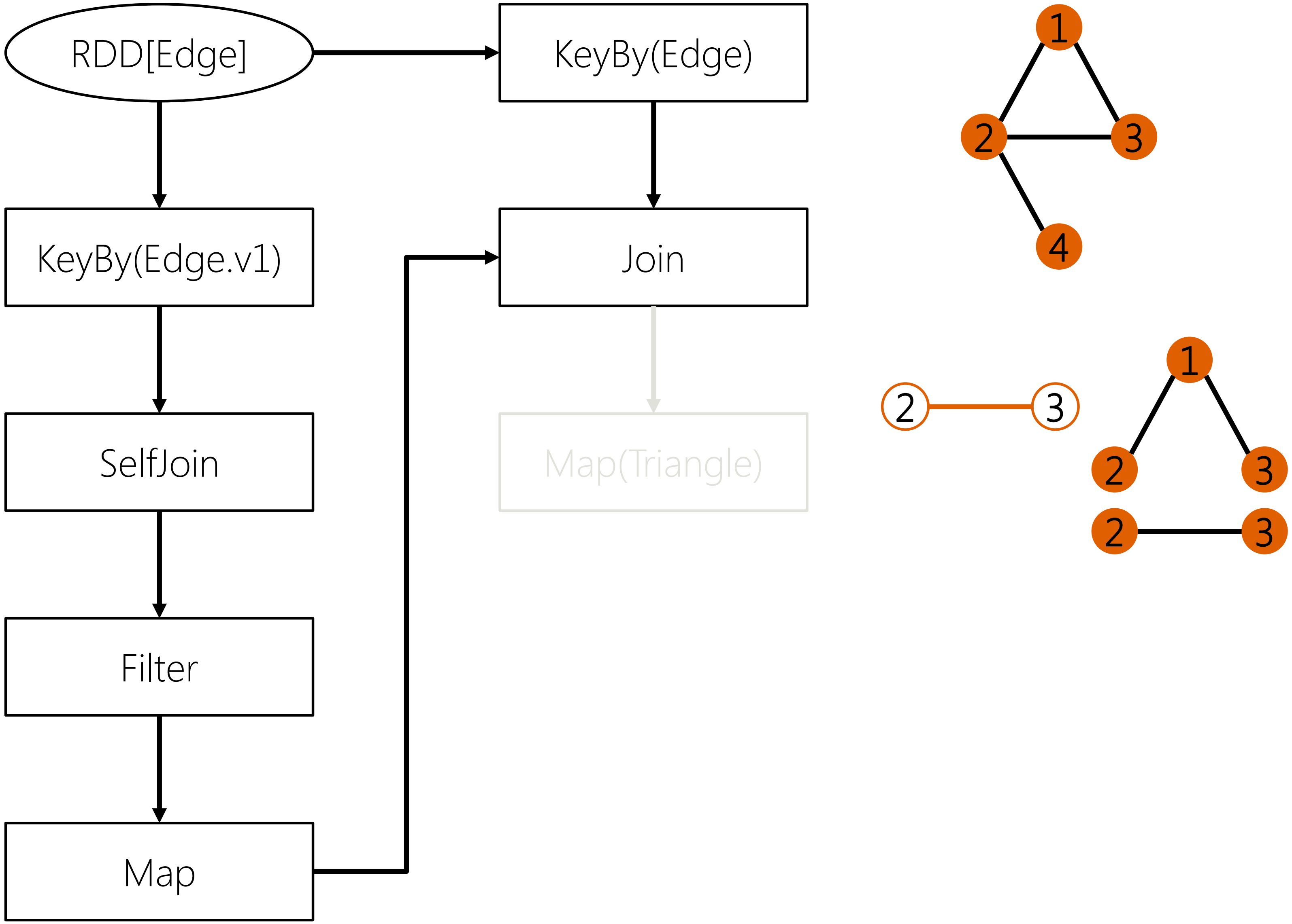




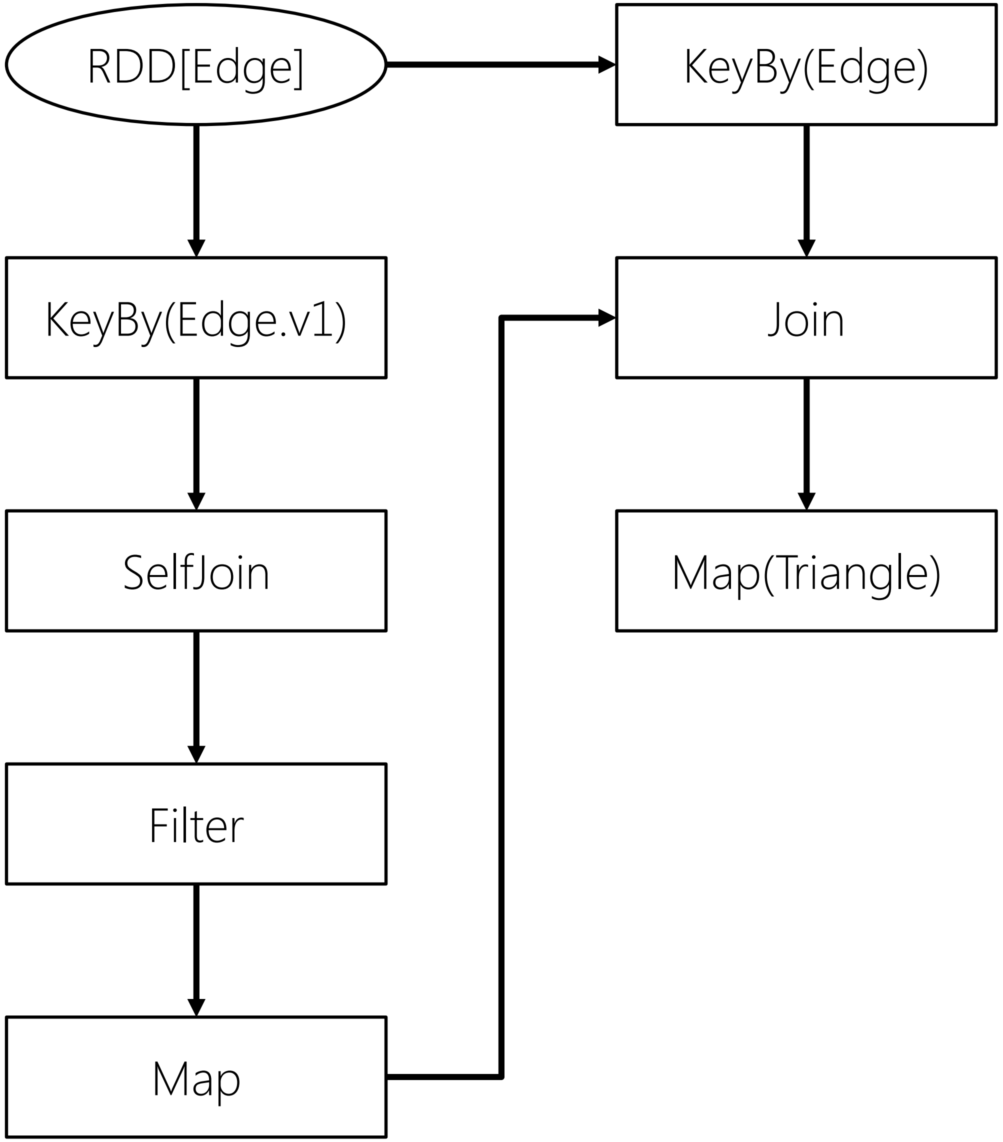
Implementation – Triangle Generation

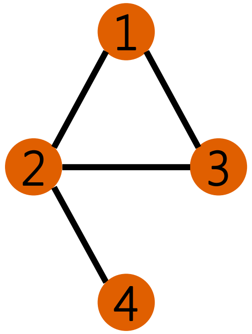


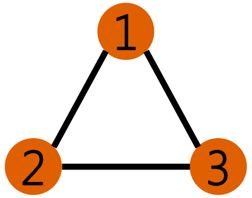
Implementation – Triangle Generation



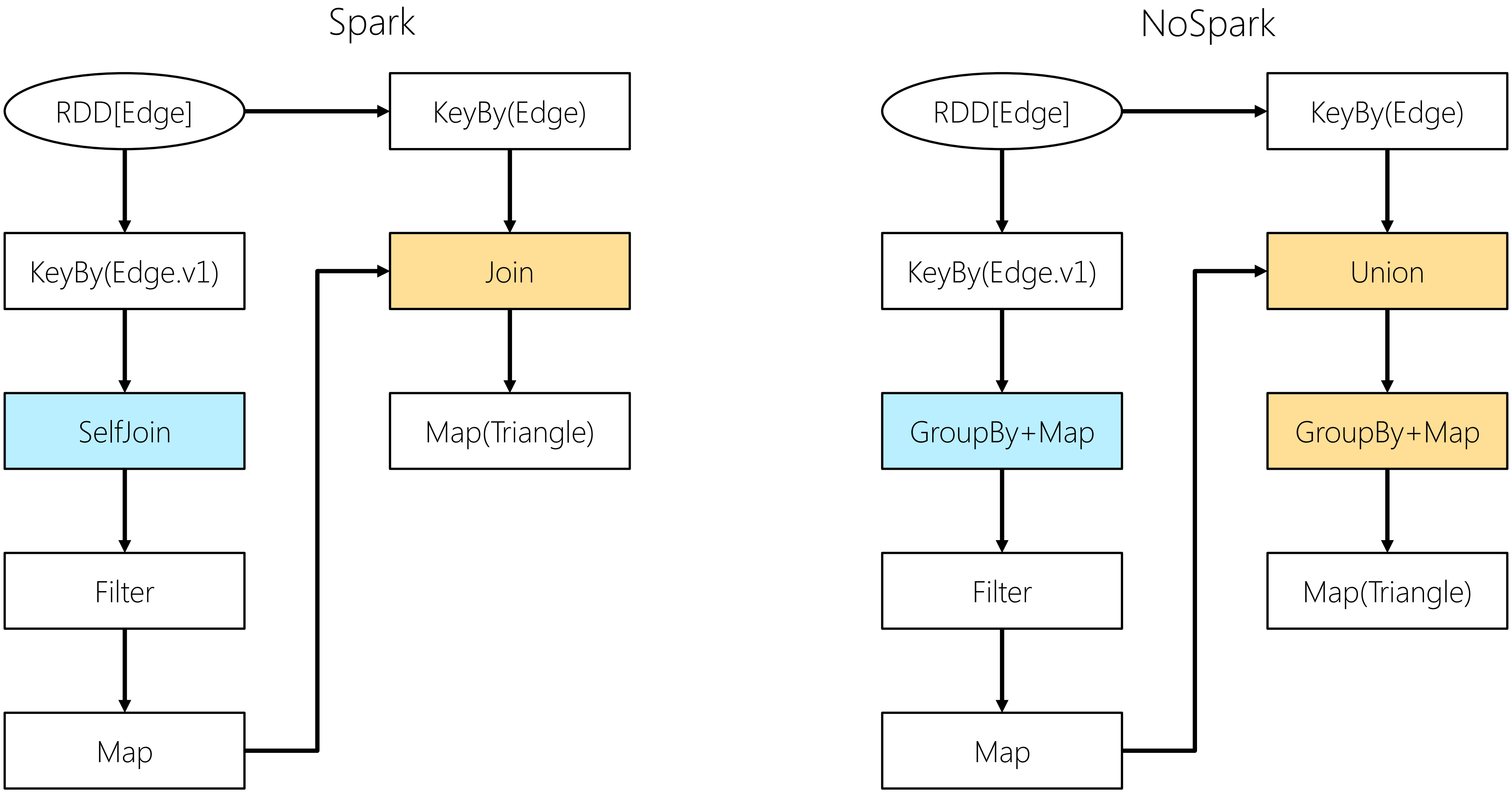
Implementation – Triangle Generation







Evaluation – Triangle Generation



Evaluation – Triangle Generation

By number of cores used

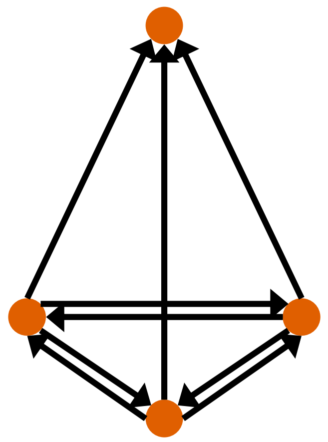
- 1 core x 5 machines
  - Spark 14 minutes
  - NoSpark 12 minutes
- 1 core x 10 machines
  - Spark 6.8 minutes
  - NoSpark 5.7 minutes
- 2 cores x 10 machines
  - Spark 6.5 minutes
  - NoSpark 4.1 minutes

using 4GB RAM

Direction?

Two possibilities

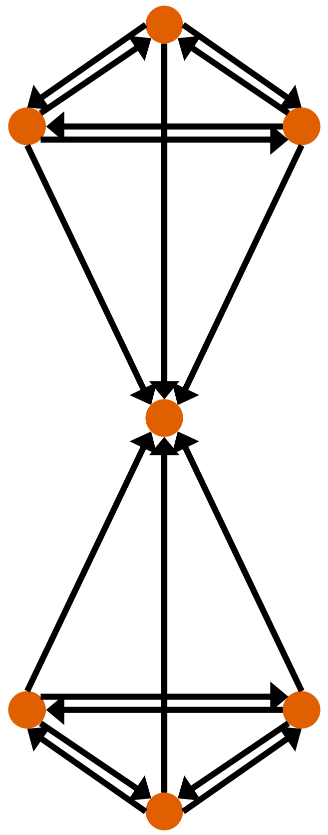
- Any direction
  - Accept trusses where either direction of an edge exists
- Both directions
  - Accept trusses only when both directions of an edge exist



Direction?

Two possibilities

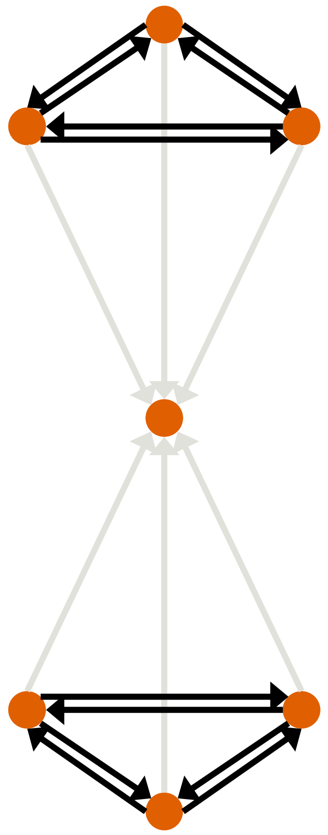
- Any direction
  - Accept trusses where either direction of an edge exists
- Both directions
  - Accept trusses only when both directions of an edge exist



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

Finding the maximum Truss

1. Create new graph from bidirectional edges only
2. Set k = arbitrary value, subraphs = (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.

Abort if k has already been seen before

(Increase or reduce k according to a binary search strategy)

Evaluation – starting k

Real 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, 28
- k = 40 – 20 minutes
  - k values tried: 40, 21, 30, 25, 27, 28, 29, 28

Using 10 machines, 20 cores, 4GB RAM