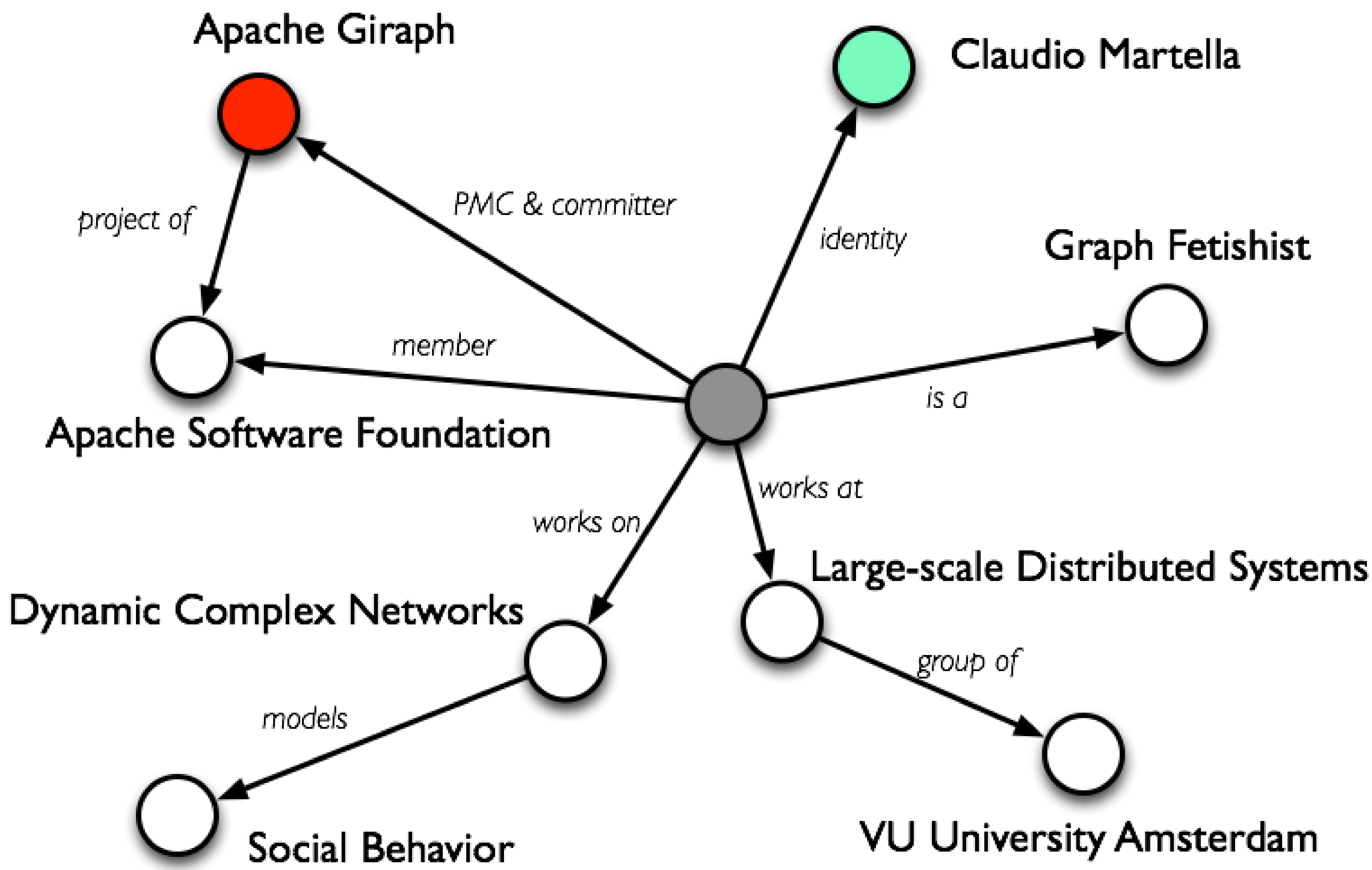
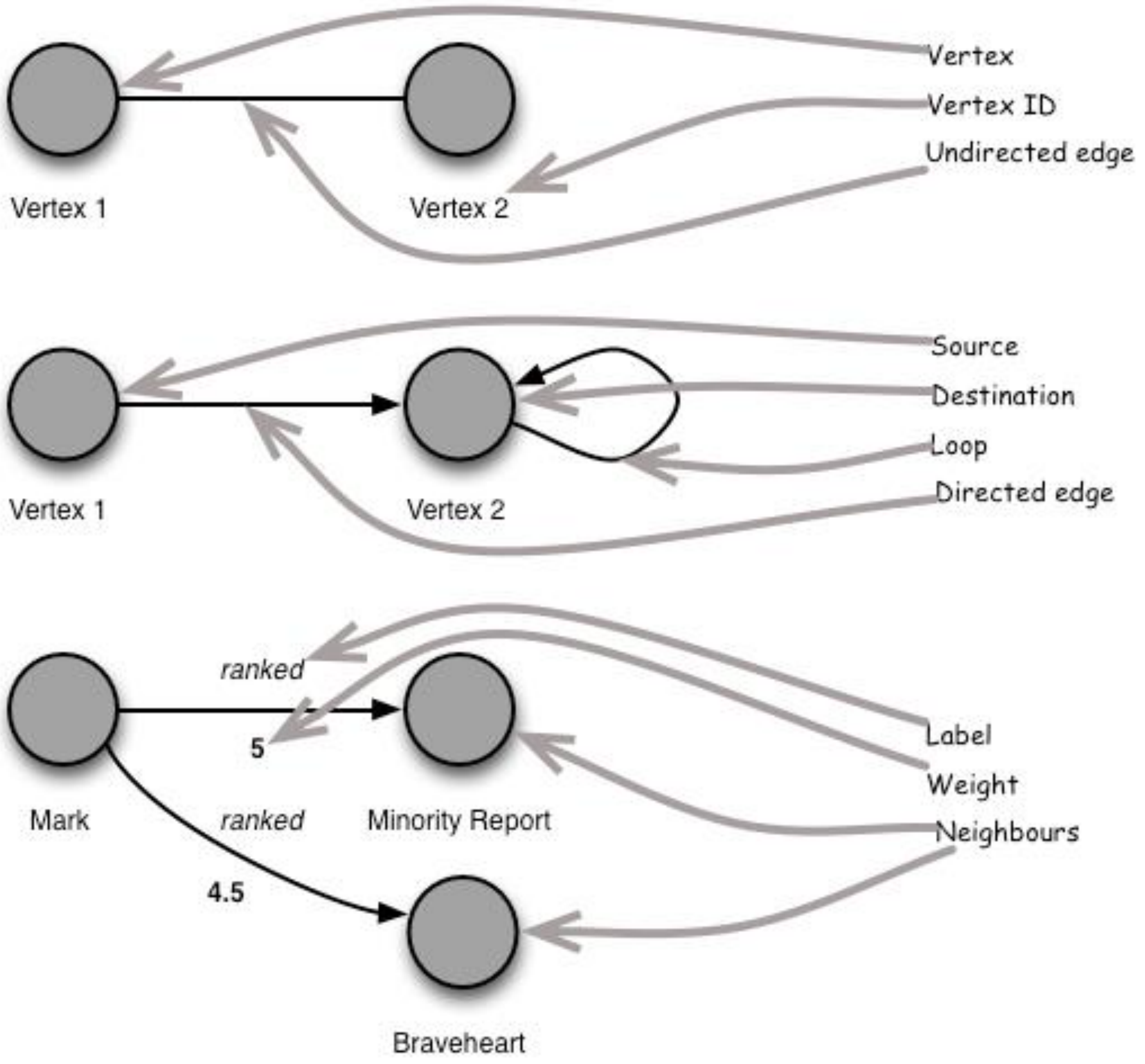
Apache Giraph

Large-scale Graph Processing on Hadoop

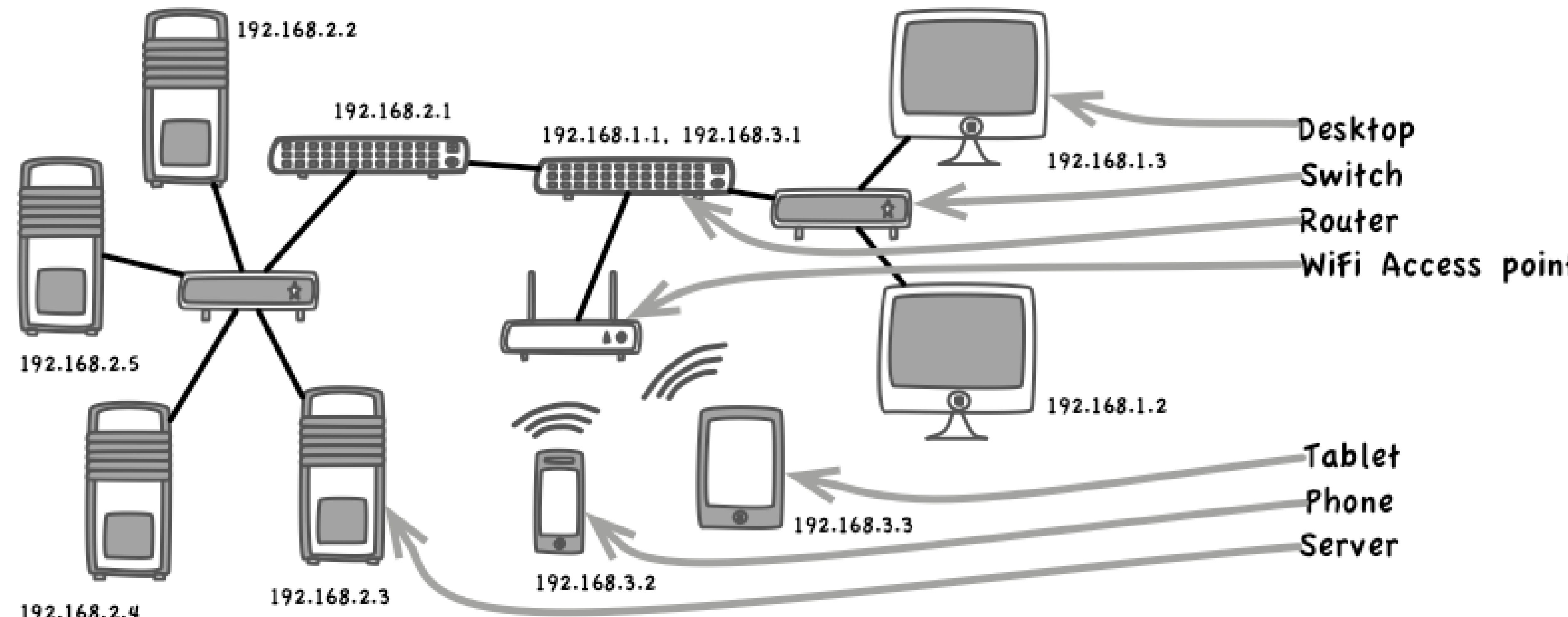
Claudio Martella
<claudio@apache.org> @claudiomartella


Apache Giraph
Claudio Martella
project of
PMC & committer
identity
Graph Fetishist
member
is a
Apache Software Foundation
works at
works on
Large-scale Distributed Systems
Dynamic Complex Networks
group of
models
Social Behavior
VU University Amsterdam

Graphs are simple

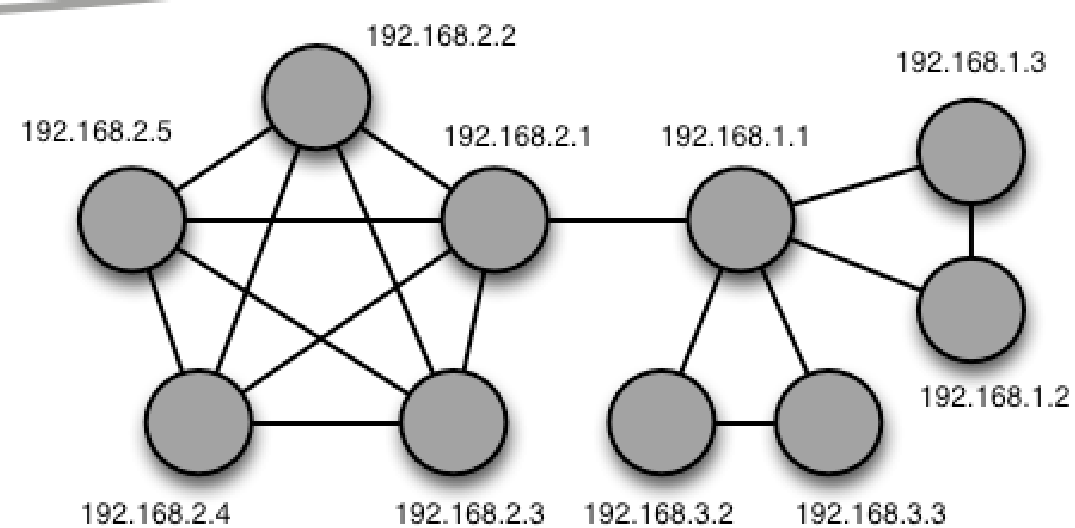


A computer network

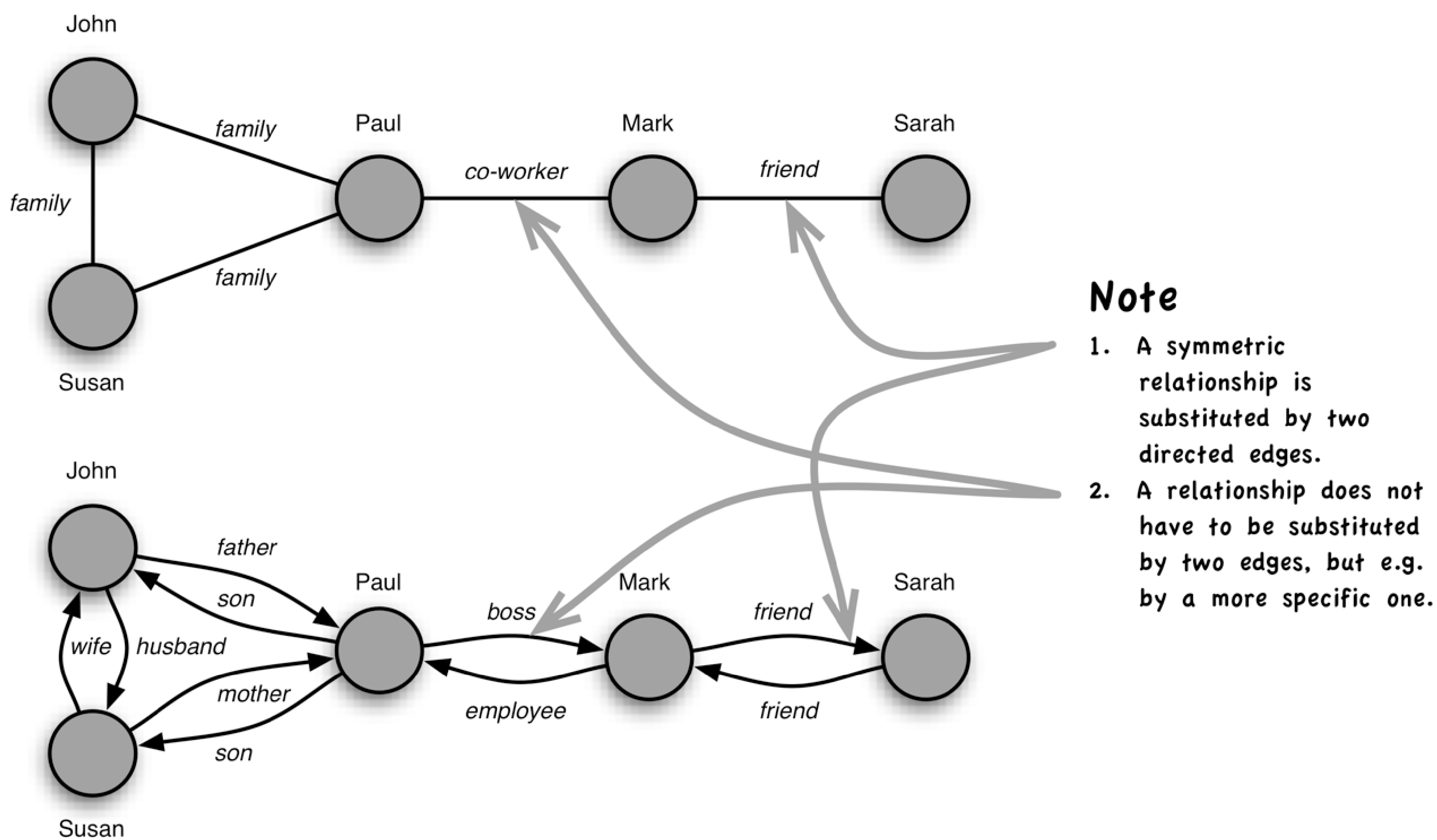


Note

1. There are three networks: servers, desktops and mobile.
2. They are connected through two routers/firewalls.
3. We ignored the switches and the access point in the graph.
4. Router 192.168.1.1 has two interfaces but we used one as vertex ID.

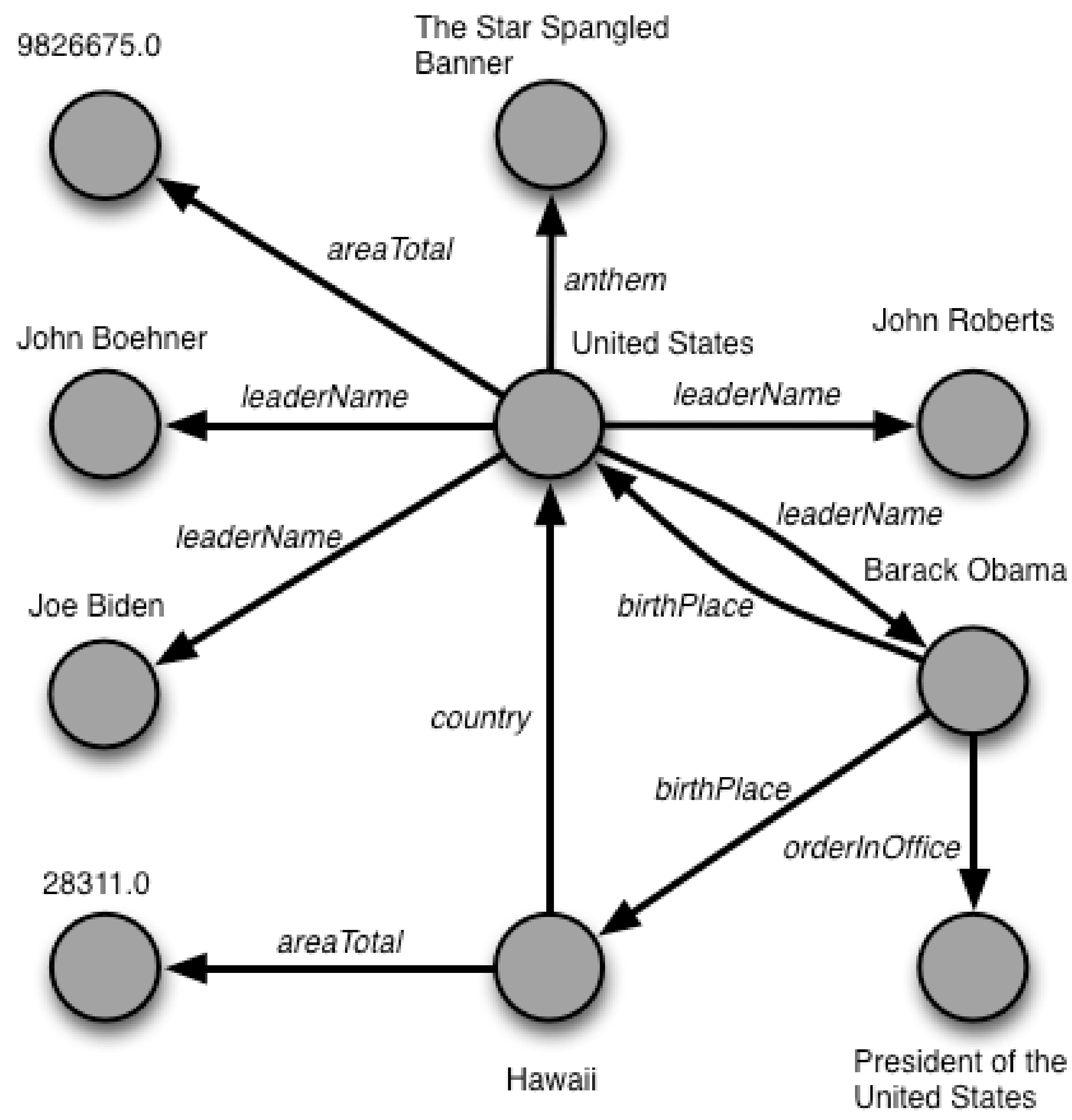


A social network



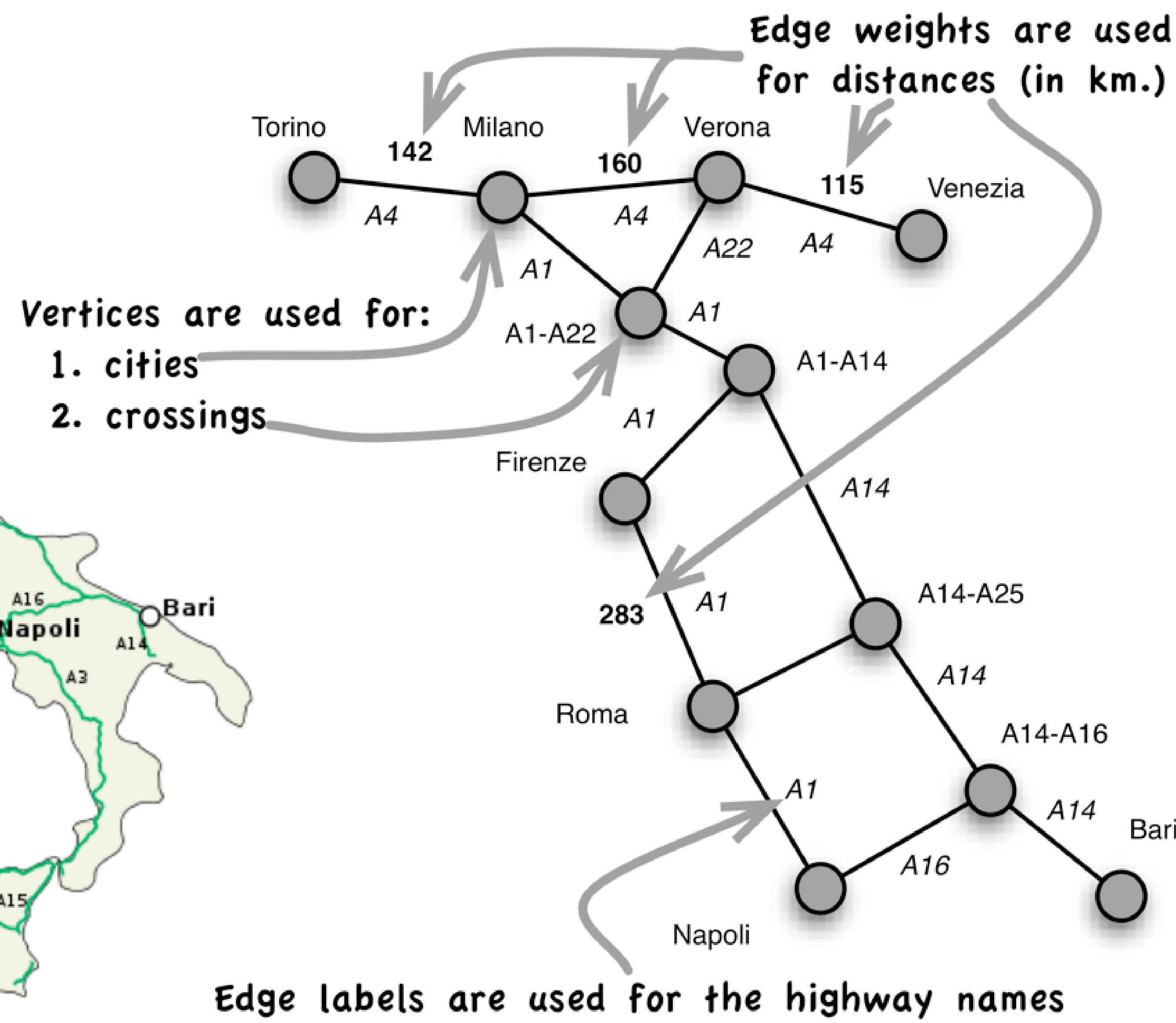
A semantic network

Subject	Predicate	Object
United States	areaTotal	9826675.0
United States	anthem	The Star Spangled Banner
United States	leaderName	Barack Obama
United States	leaderName	Joe Biden
United States	leaderName	John Boehner
United States	leaderName	John Roberts
Barack Obama	birthPlace	United States
Barack Obama	birthPlace	Hawaii
Barack Obama	orderInOffice	President of the United States
Hawaii	areaTotal	28311.0
Hawaii	country	United States



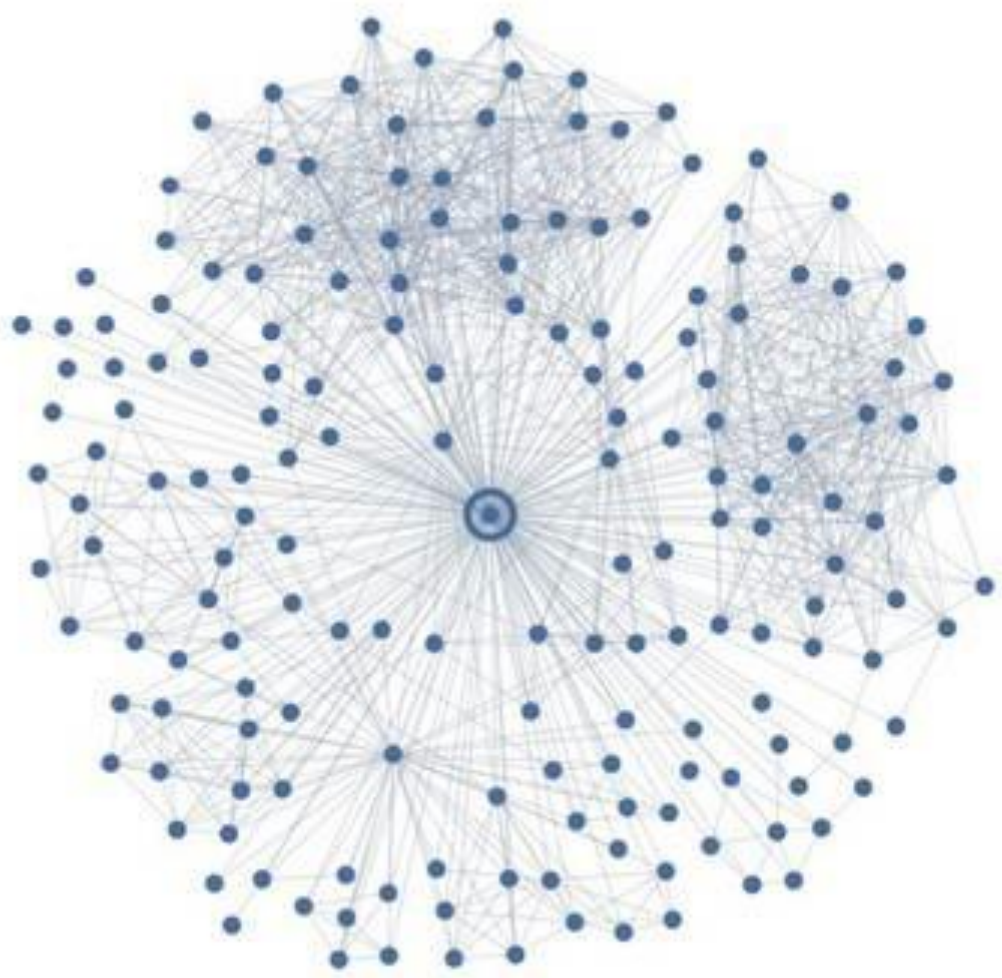
A map

A22
A27
A23
A4
A5
A9
A4
Verona
Torino
Milano
Venezia
A1
A13
A6
A10
A11
A1
Firenze
A14
A12
A1
A24
A25
A12
Roma
A1
A16
Bari
Napoli
A14
A3
Sardegna
A20
A15
Palermo
Sicilia

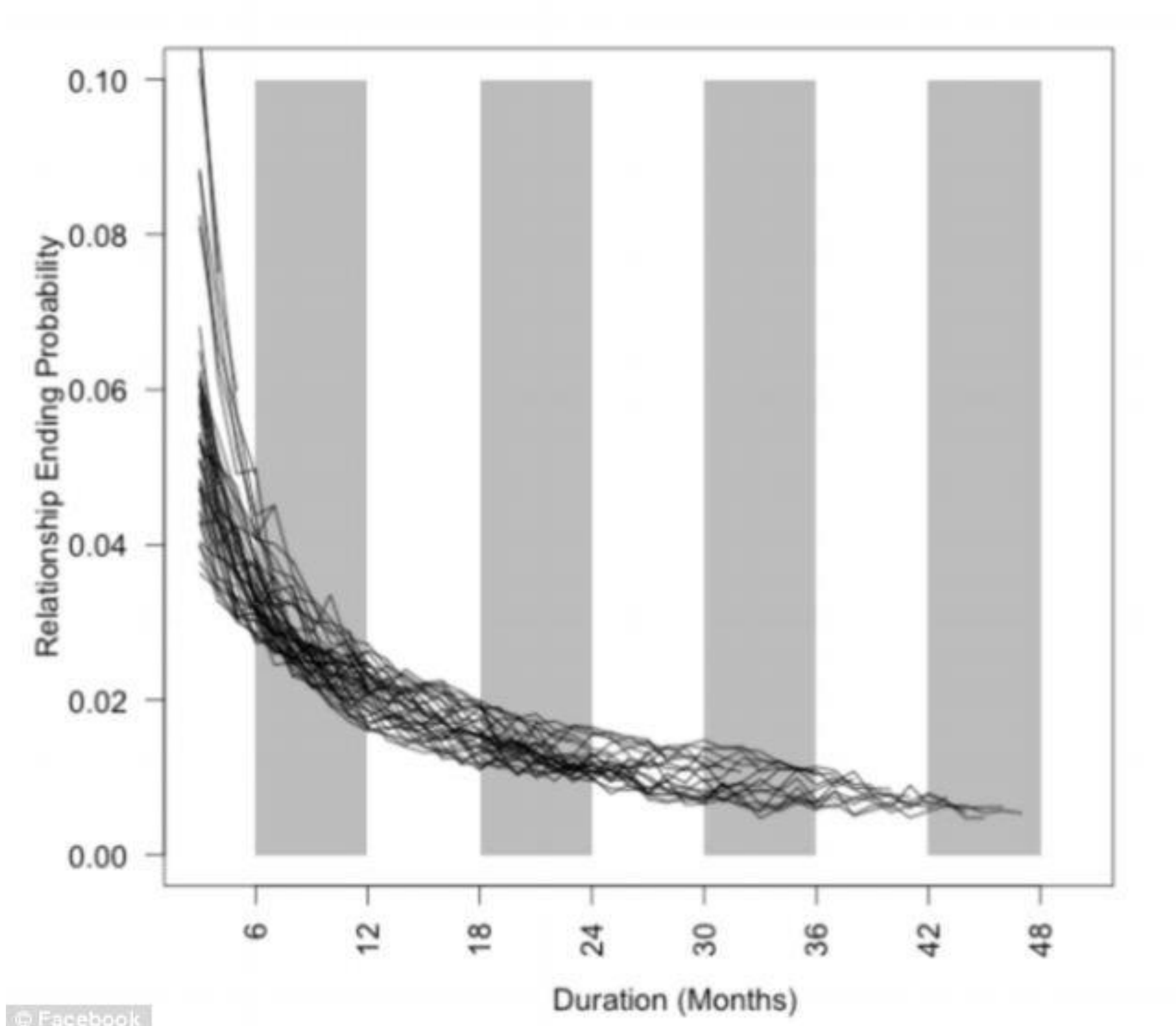


Edge weights are used for distances (in km.)
Torino
142
Milano
160
Verona
115
Venezia
A4
A4
A22
A4
A1
Vertices are used for:
1. cities
2. crossings
A1-A22
A1
A1-A14
A1
Firenze
A14
A14-A25
283
A1
A14
Roma
A14-A16
A1
A14
Bari
A16
Napoli
Edge labels are used for the highway names

Predicting break ups



Graph approach

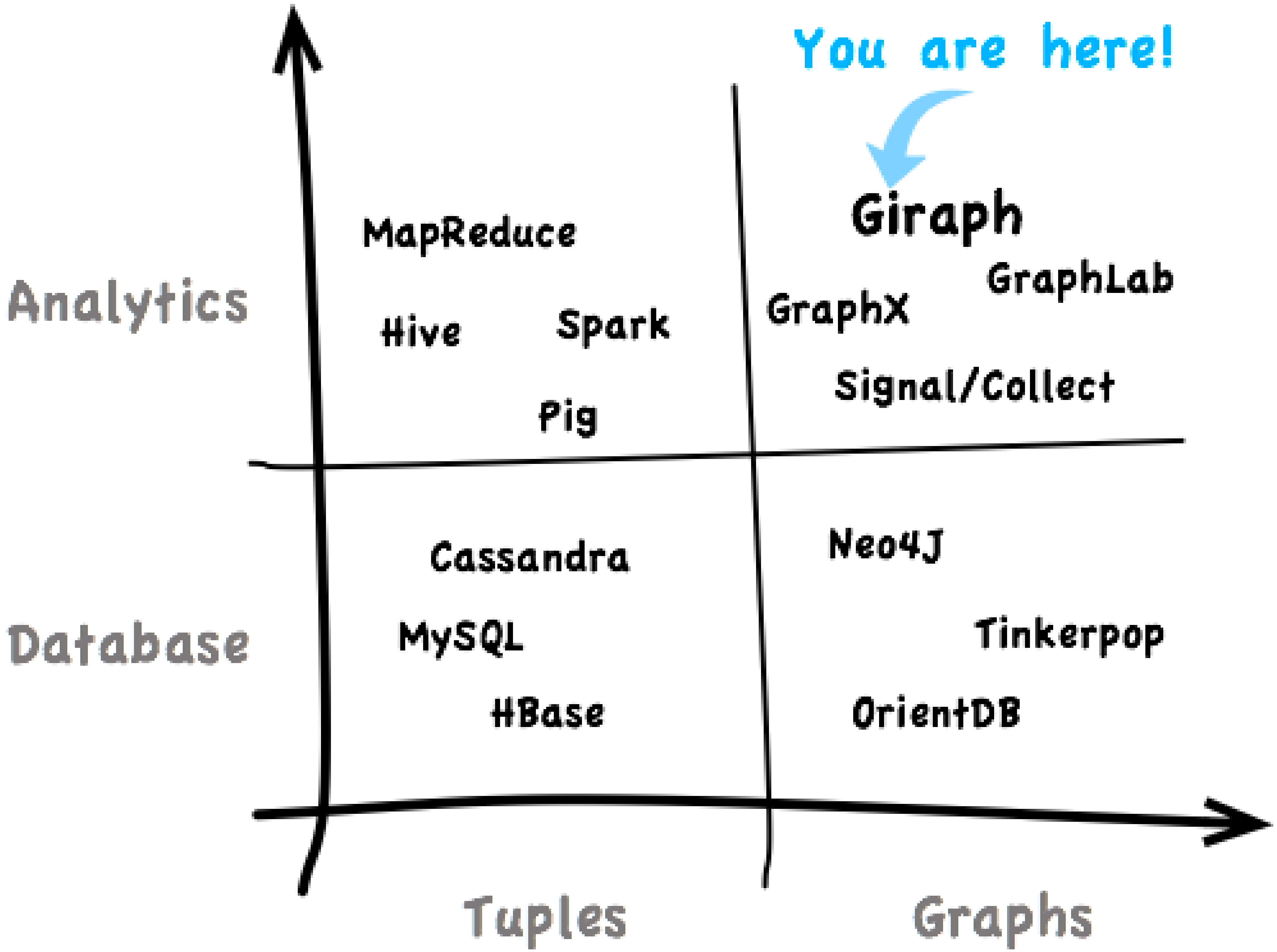


Aggregation approach

Graphs are nasty.

Each vertex depends on its neighbours, recursively.

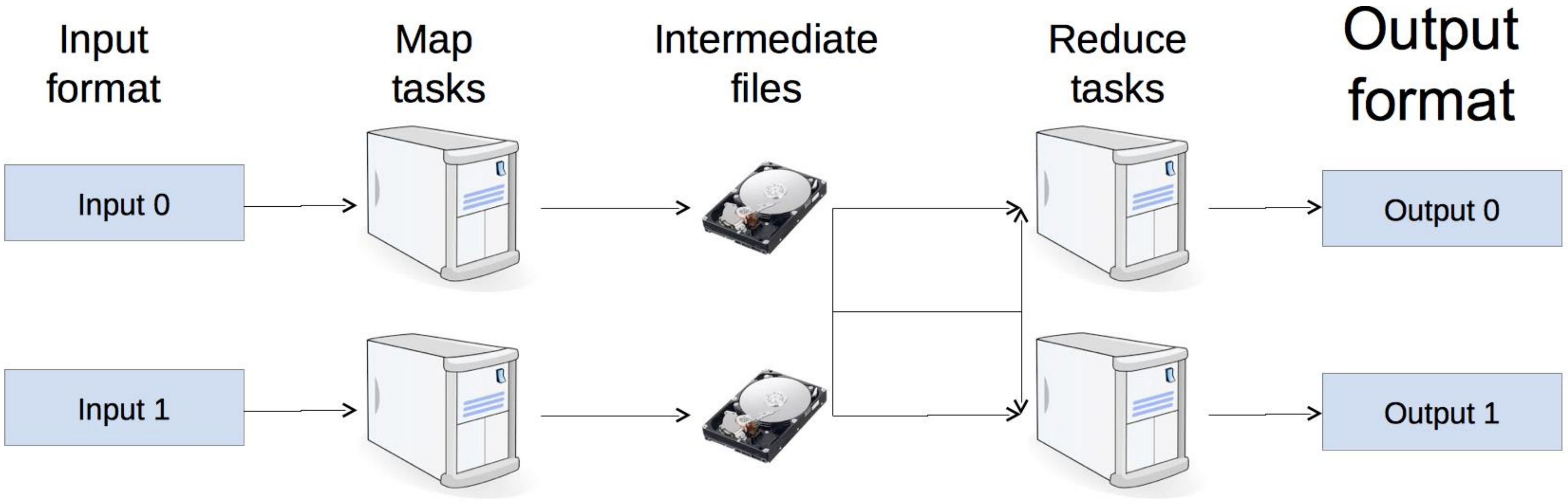
Recursive problems are nicely solved iteratively.


You are here!
Giraph
MapReduce
GraphLab
Analytics
GraphX
Hive
Spark
Signal/Collect
Pig
Cassandra
Neo4J
Database
MySQL
Tinkerpop
HBase
OrientDB
Tuples
Graphs

PageRank in MapReduce

- Record: < v_i, pr, [v_j, ..., v_k] >
- Mapper: emits < v_j, pr / #neighbours >
- Reducer: sums the partial values

MapReduce dataflow



Drawbacks

- Each job is executed N times
- Job bootstrap
- Mappers send PR values and structure
- Extensive IO at input, shuffle & sort, output

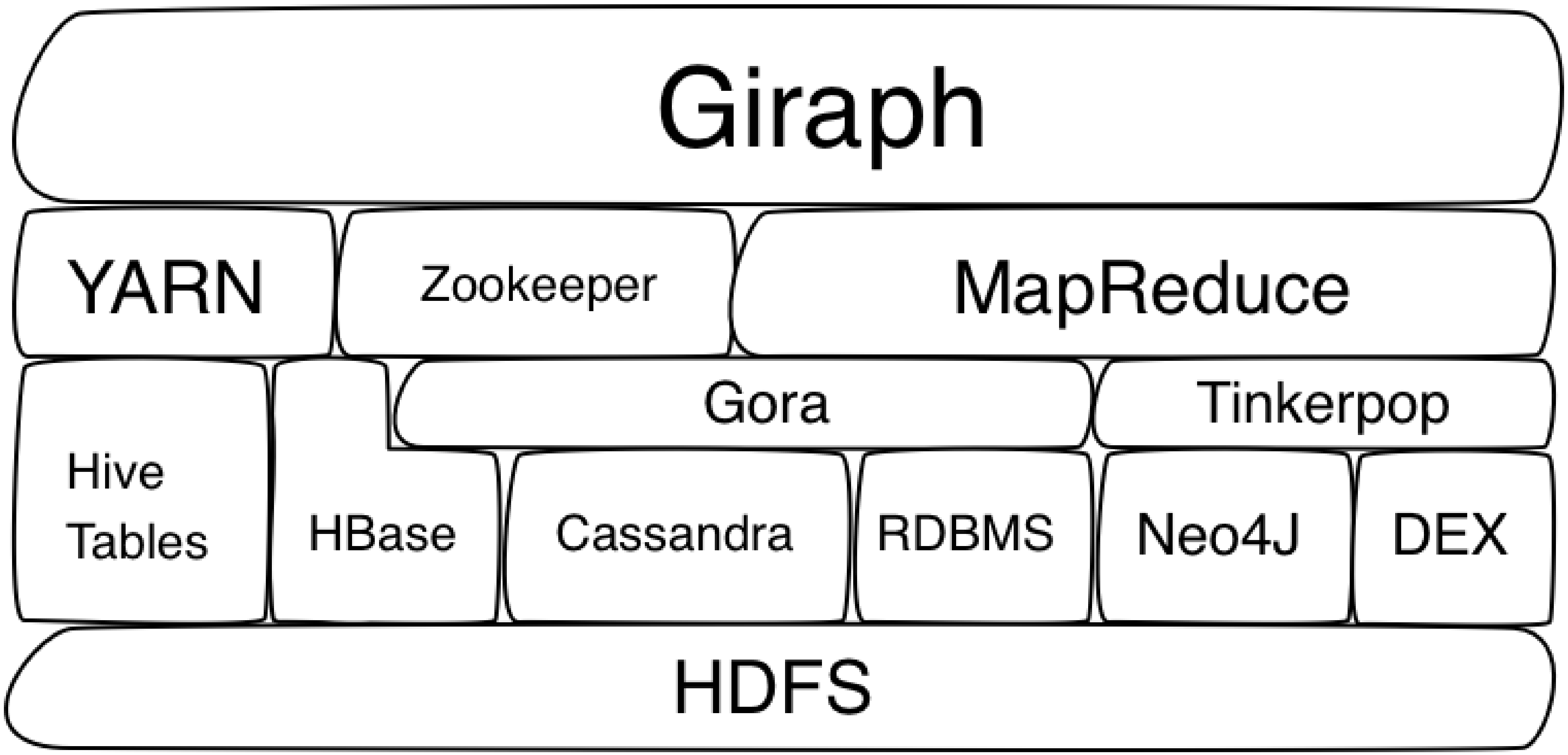
APACHE

GIRAPH

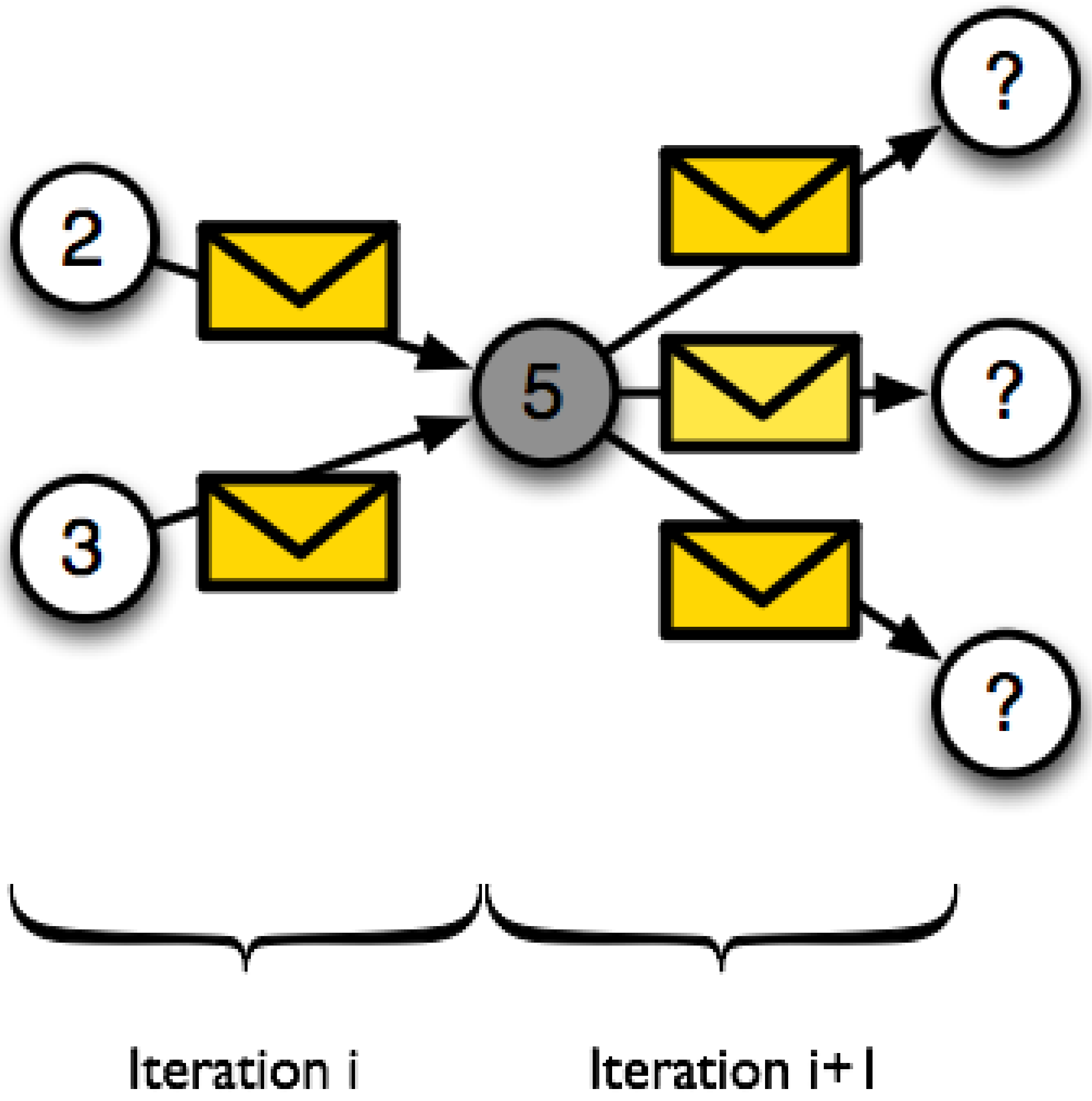
Timeline

- Inspired by Google Pregel (2010)
- Donated to ASF by Yahoo! in 2011
- Top-level project in 2012
- 1.0 release in January 2013
- 1.1 release in November 2014

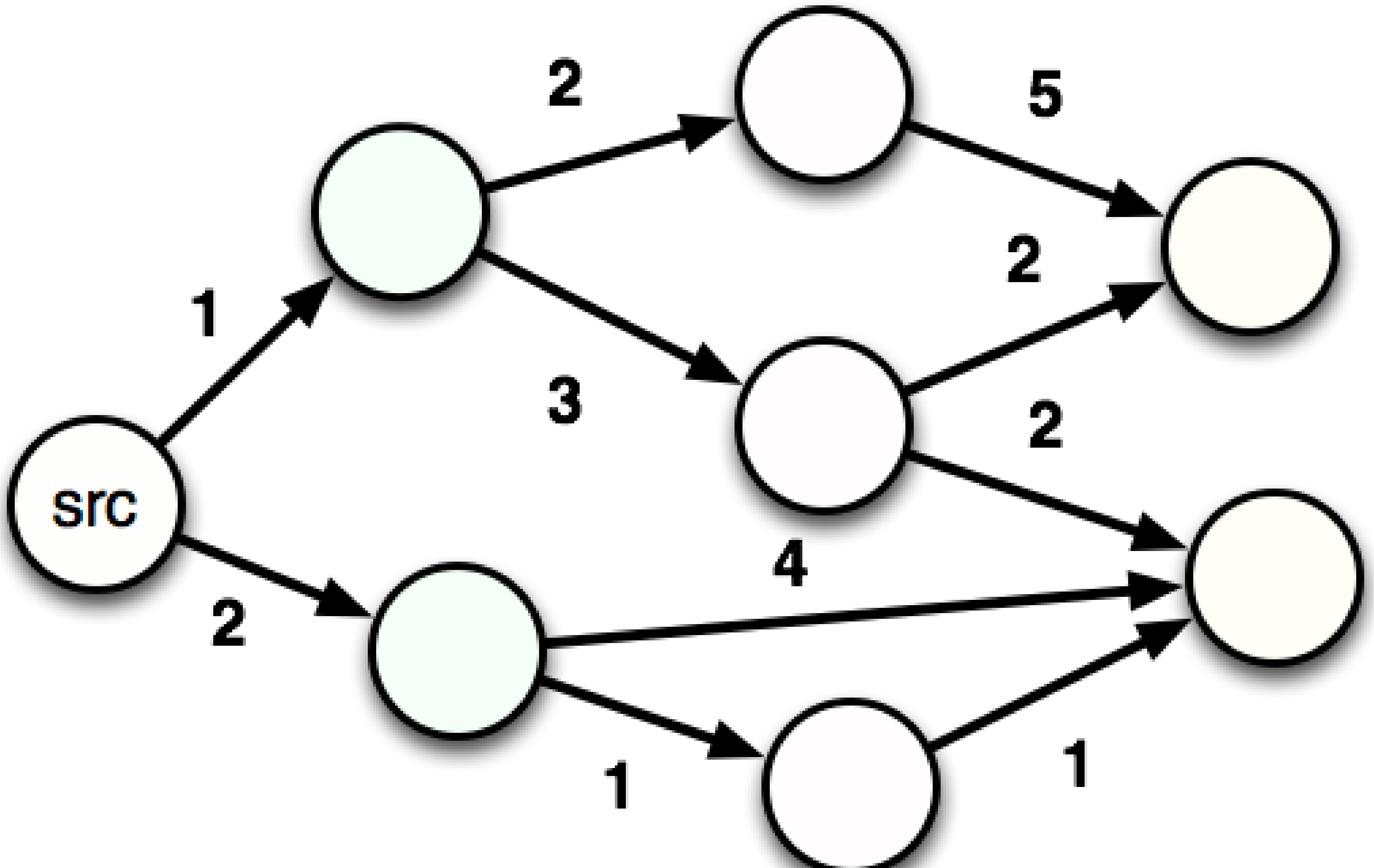
Plays well with Hadoop



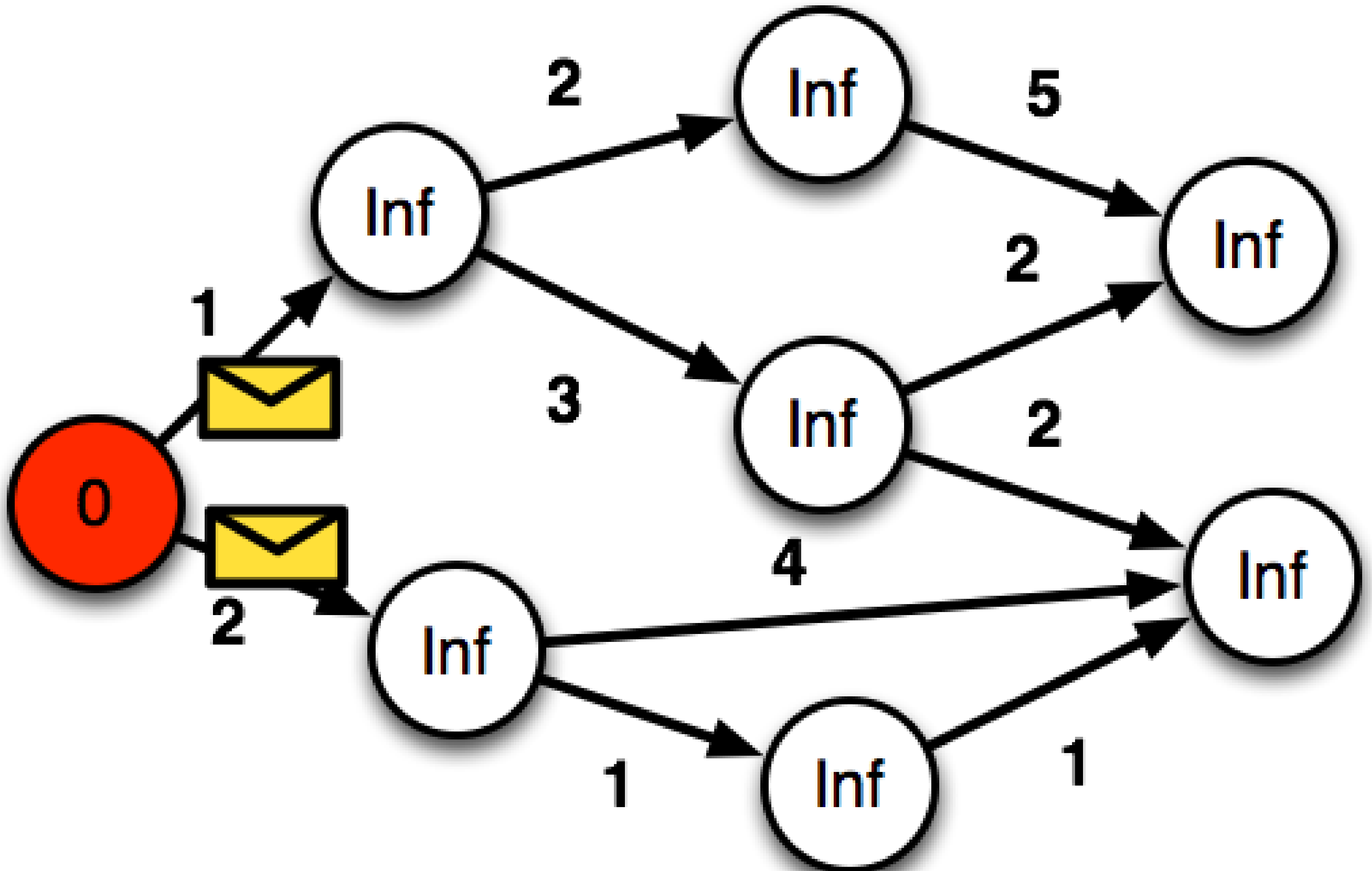
Vertex-centric API



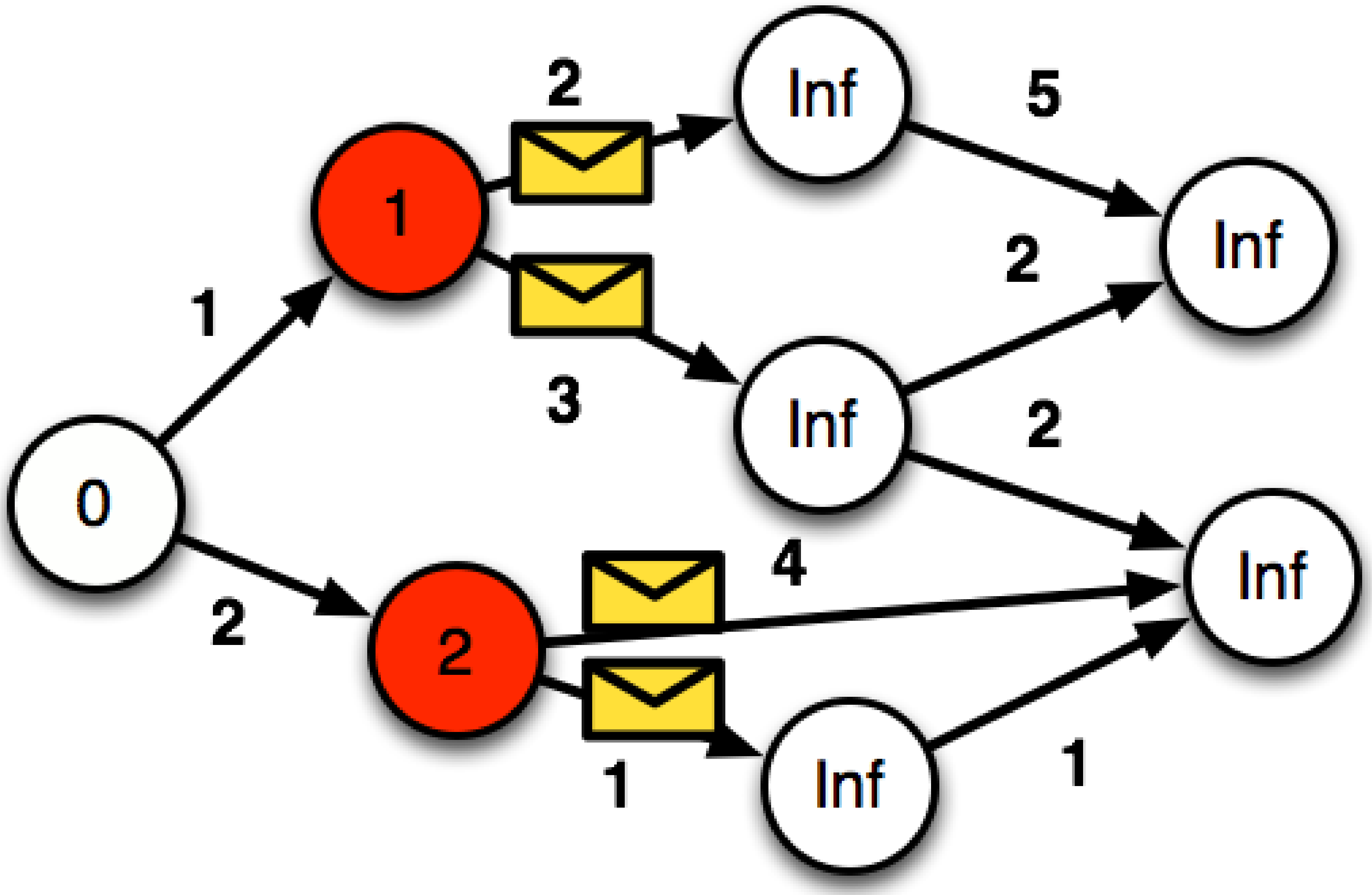
Shortest Paths



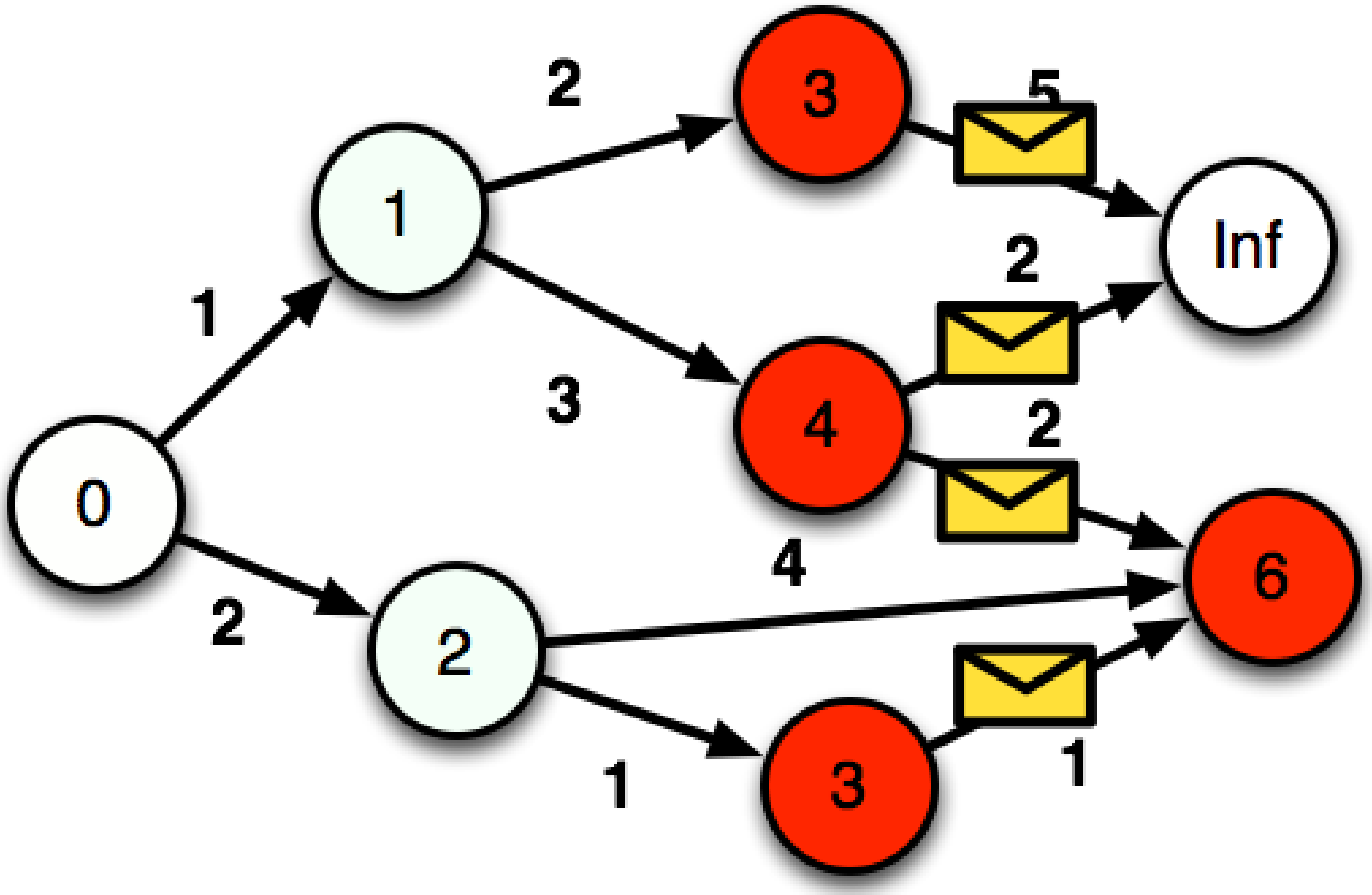
Shortest Paths



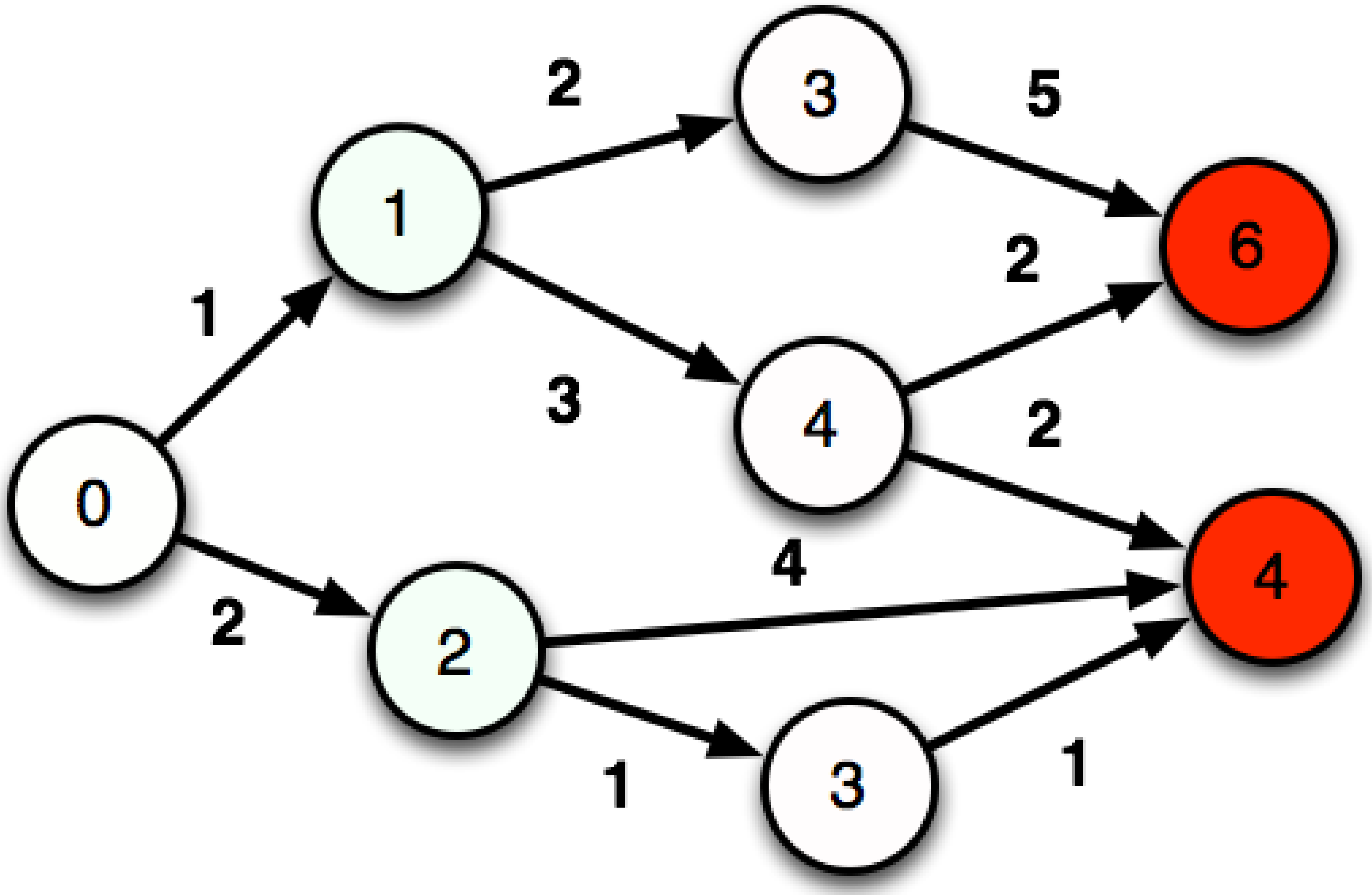
Shortest Paths



Shortest Paths

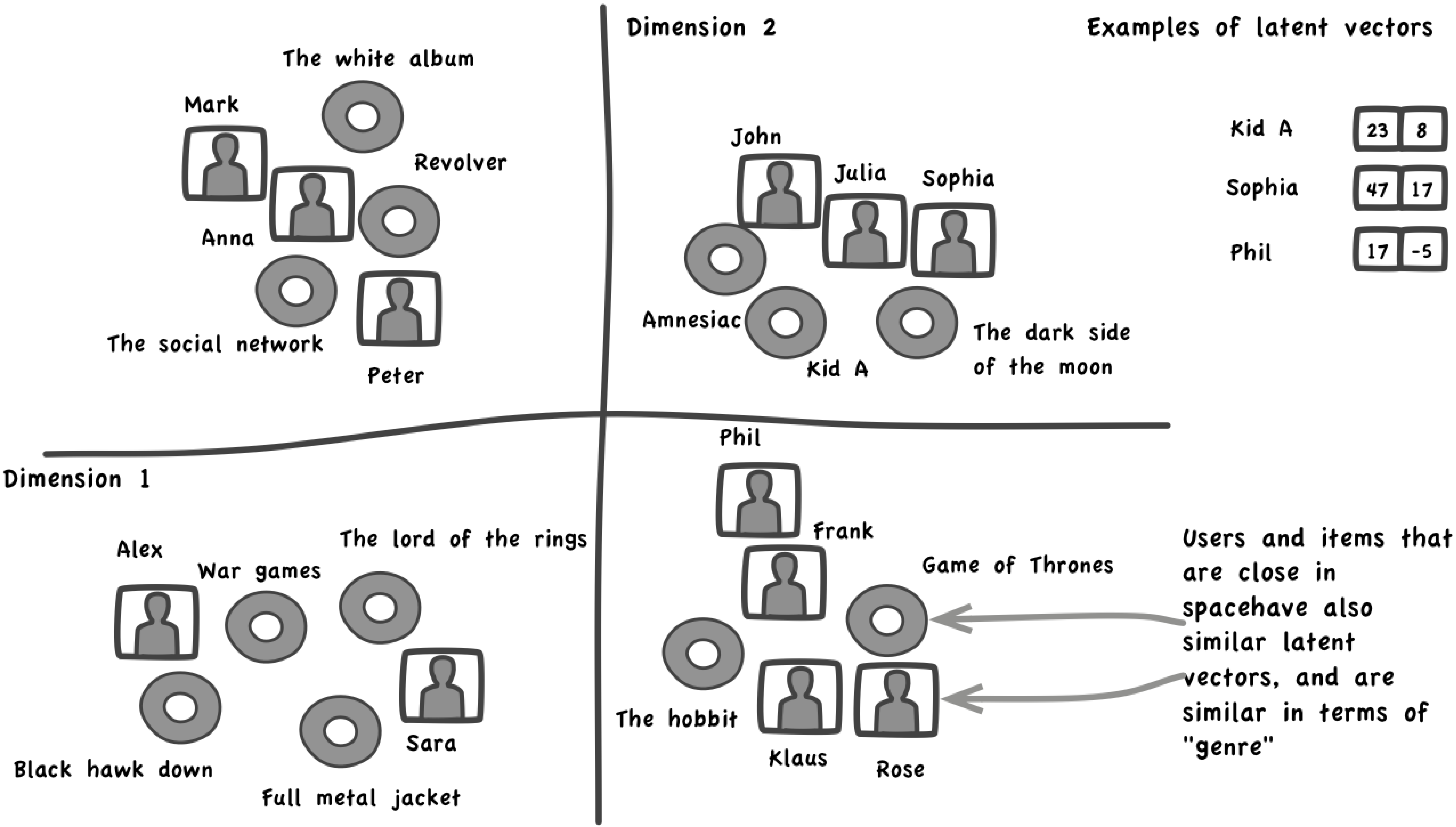


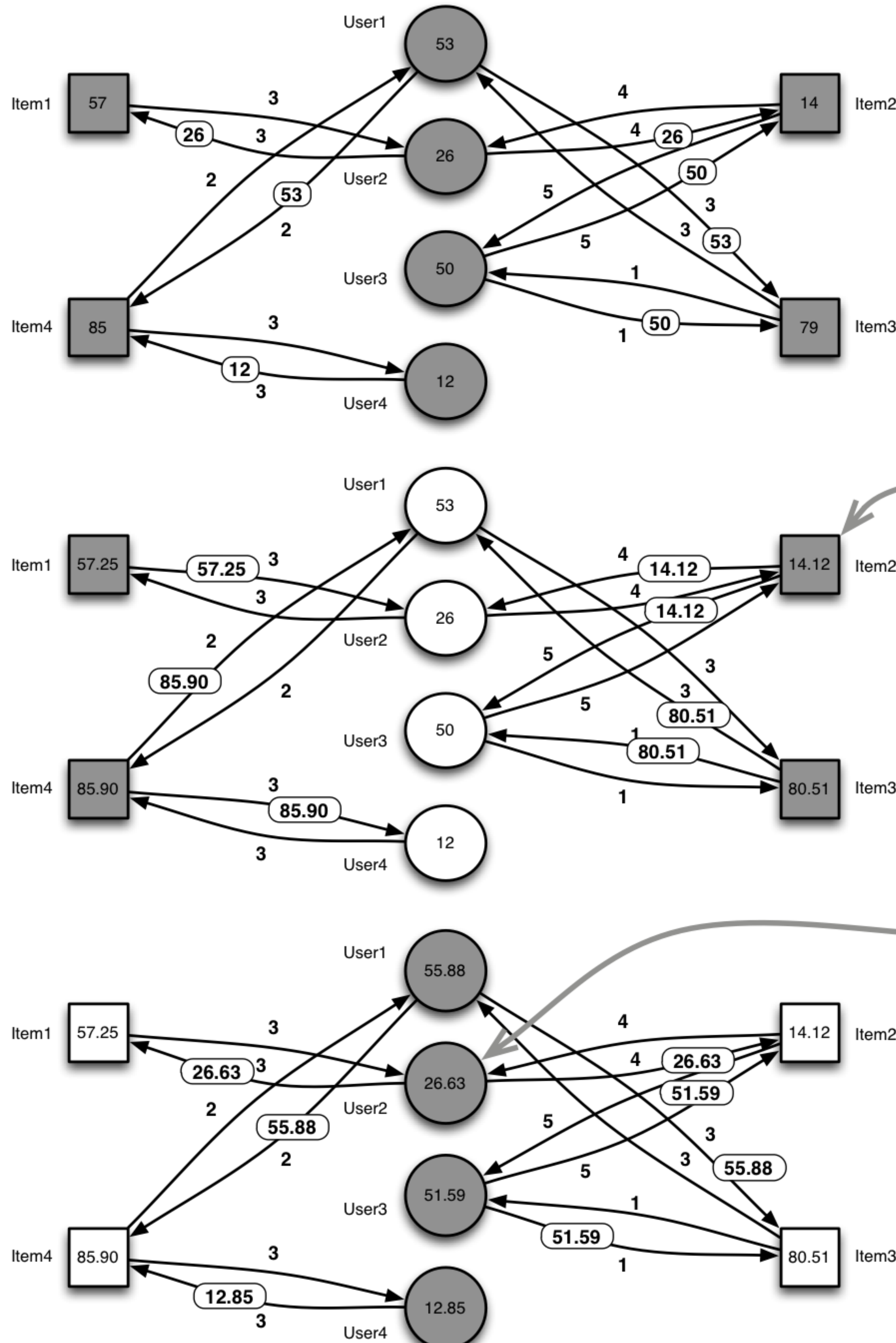
Shortest Paths



Code

```
def compute(vertex, messages):
    minValue = Inf     # float('Inf')
    for m in messages:
        minValue = min(minValue, m)
    if minValue < vertex.getValue():
        vertex.setValue(minValue)
        for edge in vertex.getEdges():
            message = minValue + edge.getValue()
            sendMessage(edge.getTargetId(), message)
    vertex.voteToHalt()
```


Dimension 2
Examples of latent vectors
The white album
Mark
Revolver
Anna
The social network
Peter
John
Julia
Sophia
Amnesiac
Kid A
The dark side of the moon
Kid A
23
8
Sophia
47
17
Phil
17
-5
Dimension 1
Alex
War games
The lord of the rings
Black hawk down
Full metal jacket
Sara
Phil
Frank
Game of Thrones
The hobbit
Klaus
Rose
Users and items that are close in spacehave also similar latent vectors, and are similar in terms of "genre"



Superstep 0:

1. All vertices initialise their value at random.
2. User vertices send their value to their endpoints.
3. All vertices vote to halt.

The item vector updates its latent vector based on the latent vectors coming from the users, and spreads it around.

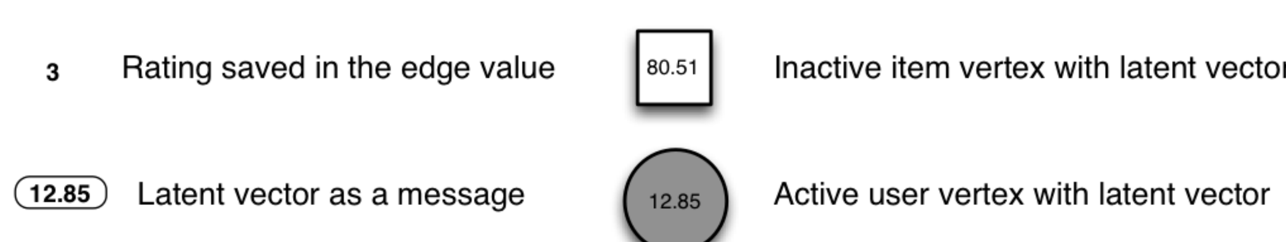
Superstep 1:

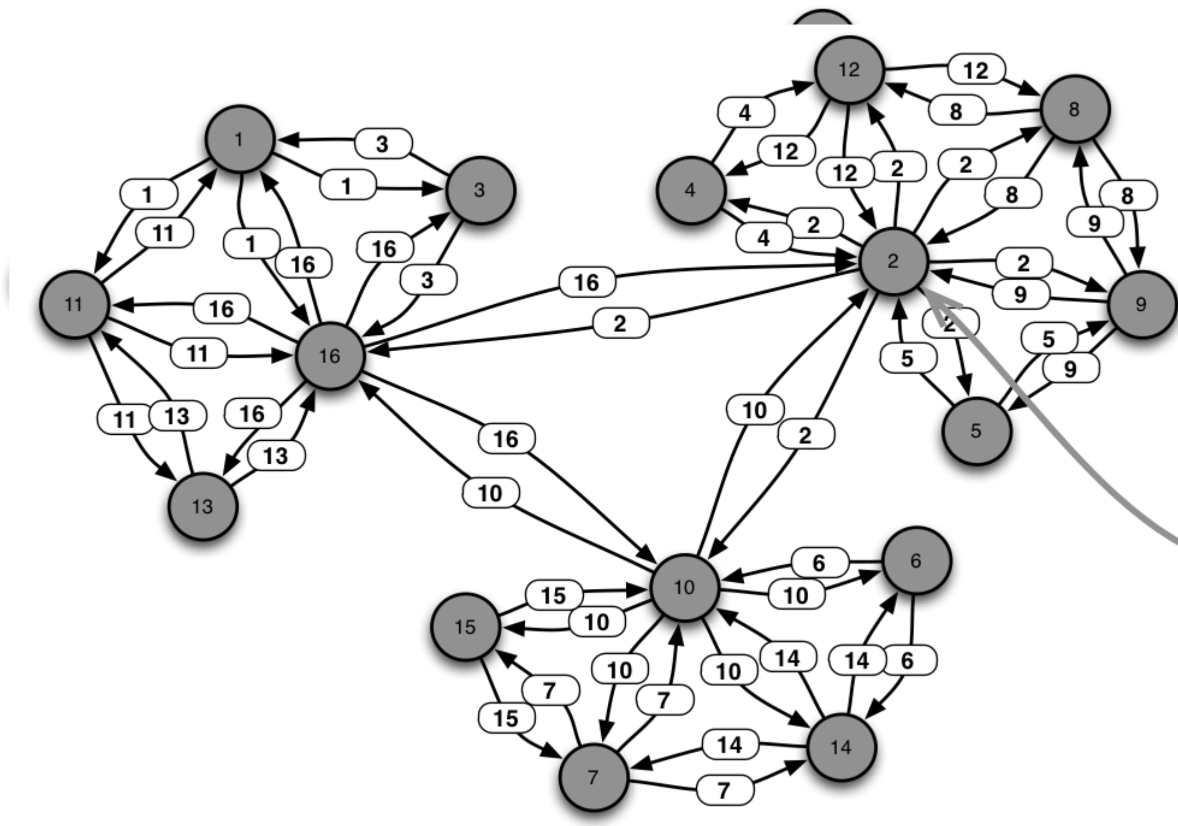
1. Item vertices are woken up.
2. Item vertices update their values according to SGD based on the messages, edge values and the current latent vector
3. Item vertices send their new value to their endpoints.
4. Item vertices vote to halt.

It's now the turn of the user vectors to update their latent vectors based on the items' latent vectors computed in the previous super step

Superstep 2:

1. User vertices are woken up.
2. User vertices update their values according to SGD based on the messages, edge values and the current latent vector
3. User vertices send their new value to their endpoints.
4. User vertices vote to halt.

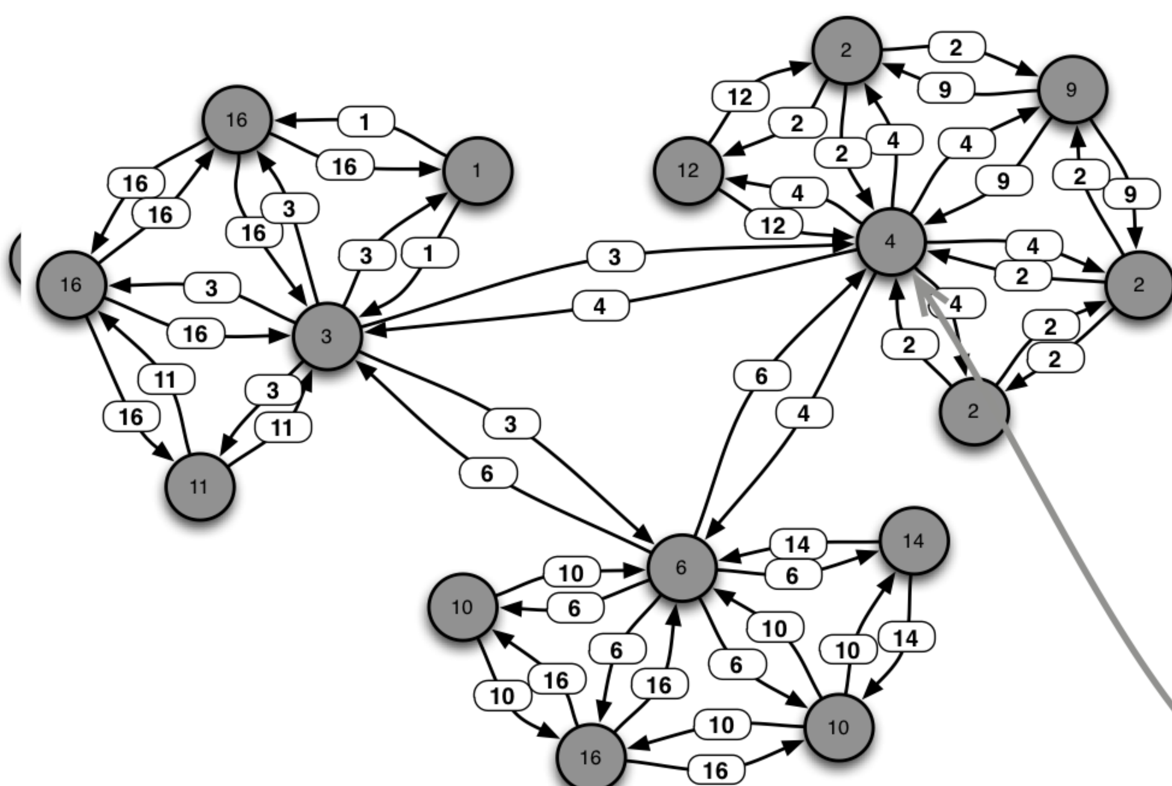




Superstep 0:

1. All vertices initialise their value to their ID.
2. All vertices send their value to the other endpoint.

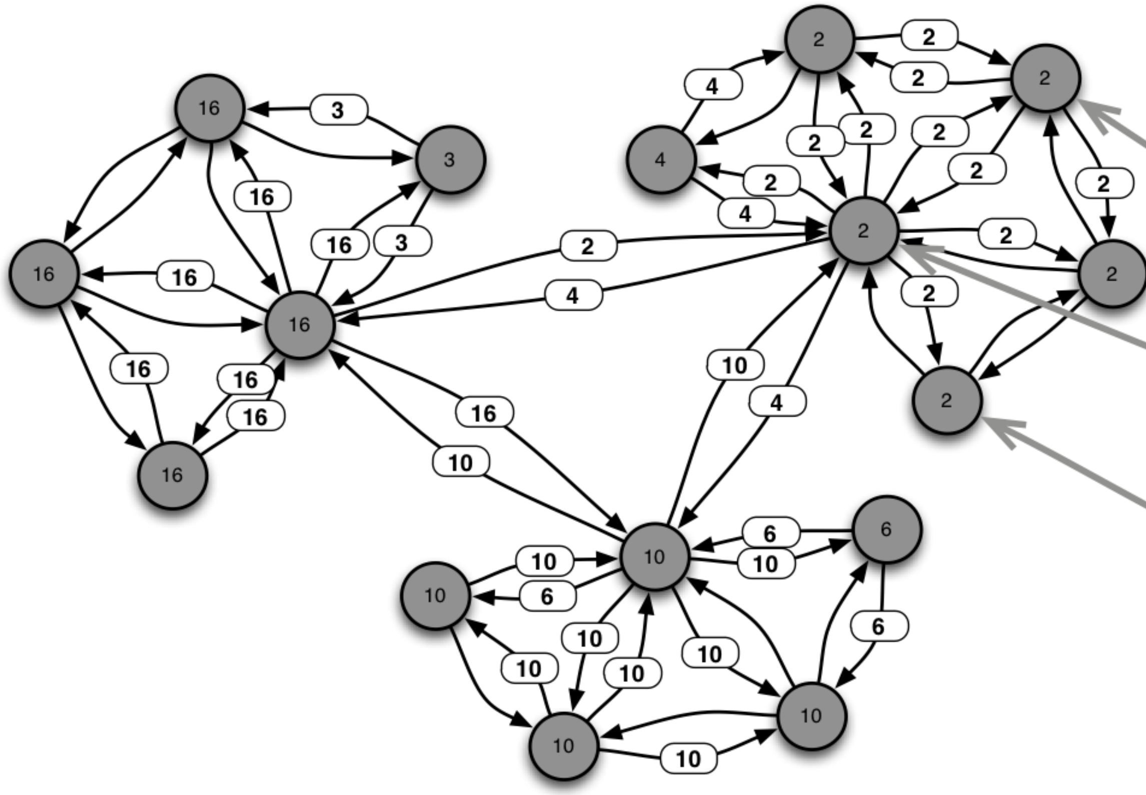
Every vertex propagates the ID through the messages.



Superstep 1:

1. Vertices receive the labels of their neighbours and compute frequency.
2. Edge values are initialised to these labels.
3. At this first superstep, all frequencies are 1 so vertices break ties randomly.
4. Vertices update their value and send it to the endpoints of their edges..

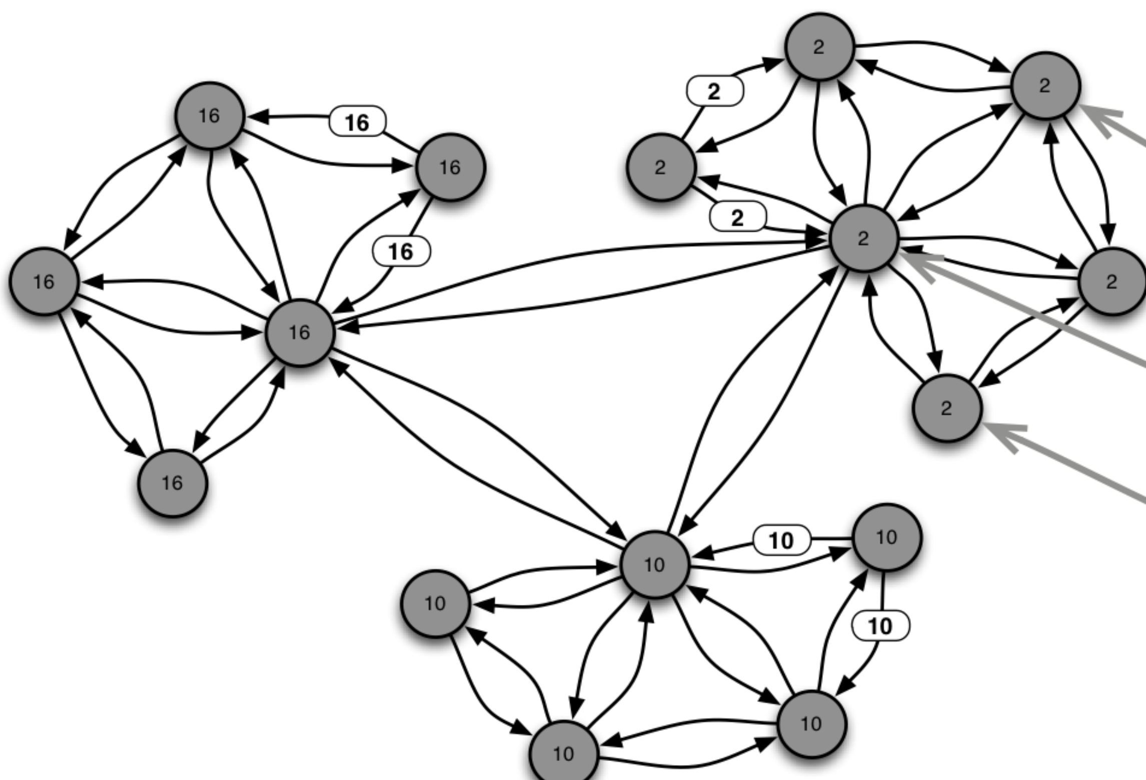
The central vertex acquires at random one of the lanes from the neighbours



Superstep 2:

1. Vertices receive the labels of their neighbours and compute the frequency.
2. Vertices update edge values
3. Vertices update their label based on the frequencies breaking ties randomly.
4. Vertices send their values to the endpoints of their edges.

The ID of the central vertex starts being propagated within the community



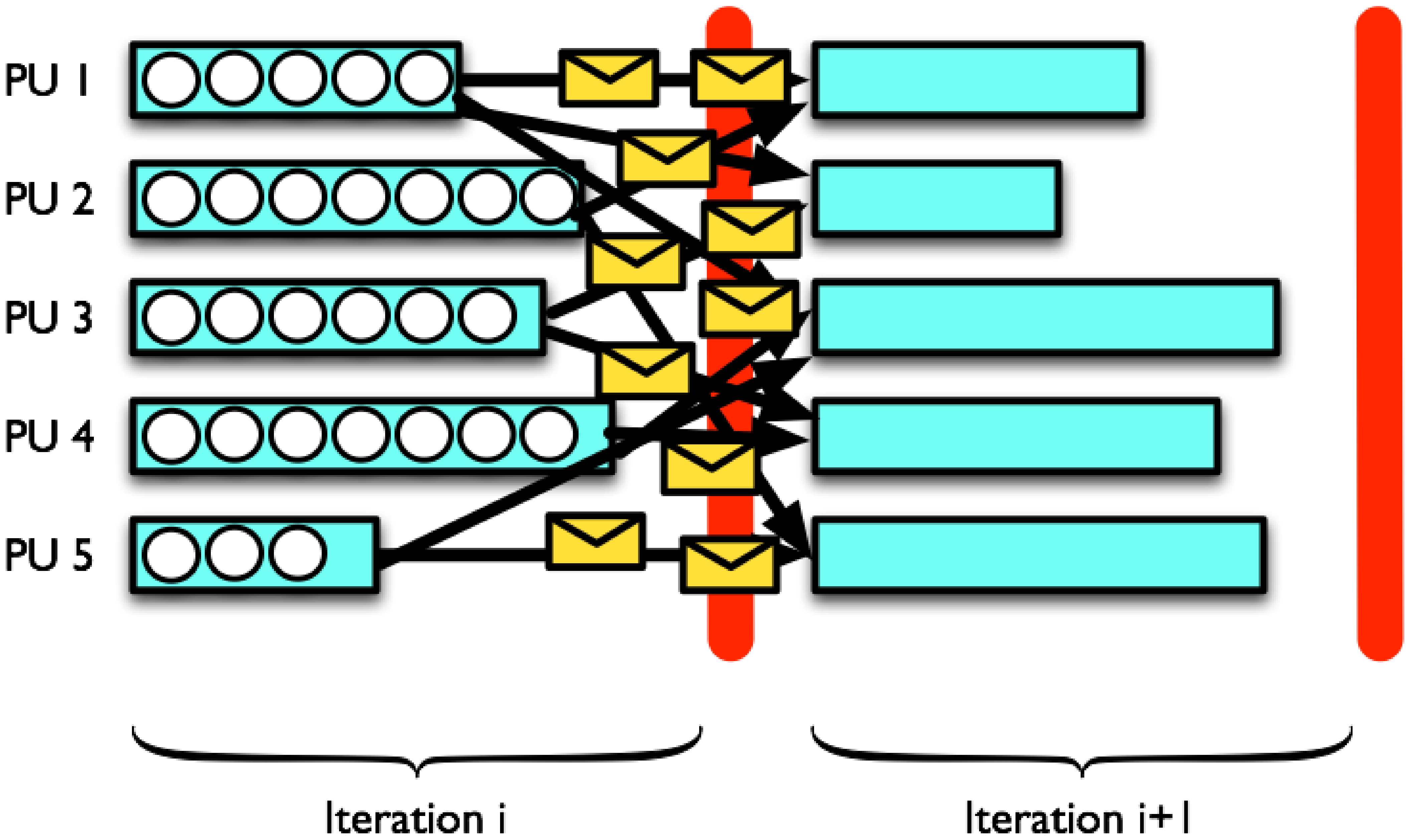
Superstep 4:

1. Remaining vertices receive the labels from the neighbours.
2. They update their edge values.
3. They compute their new labels.
4. They send their new value to the endpoints of their edges.

The whole community has now acquired the id of the central node.

6 Label as a message   6 Active vertex with label

BSP & Giraph



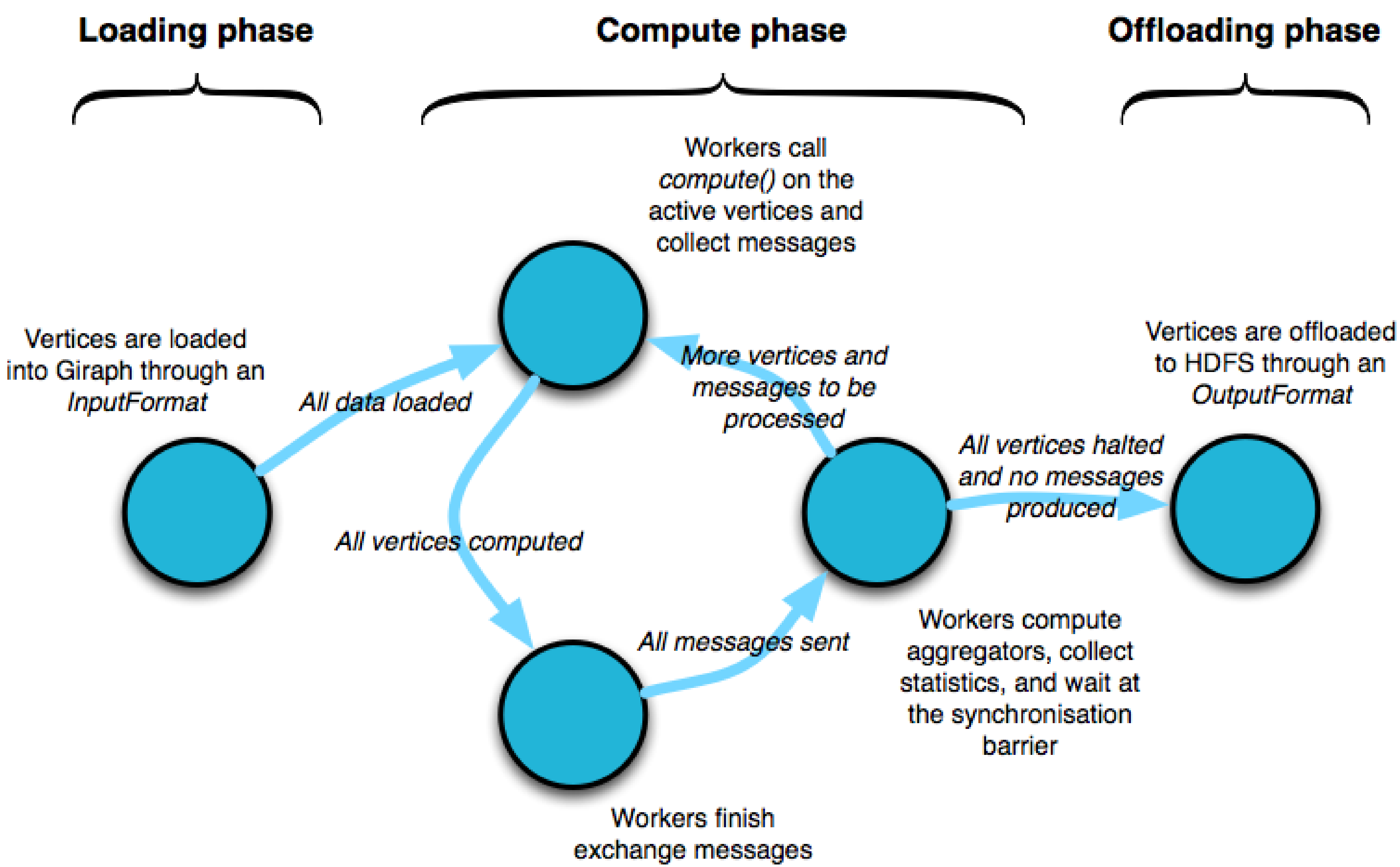
Advantages

- No locks: message-based communication
- No semaphores: global synchronization
- Iteration isolation: massively parallelizable

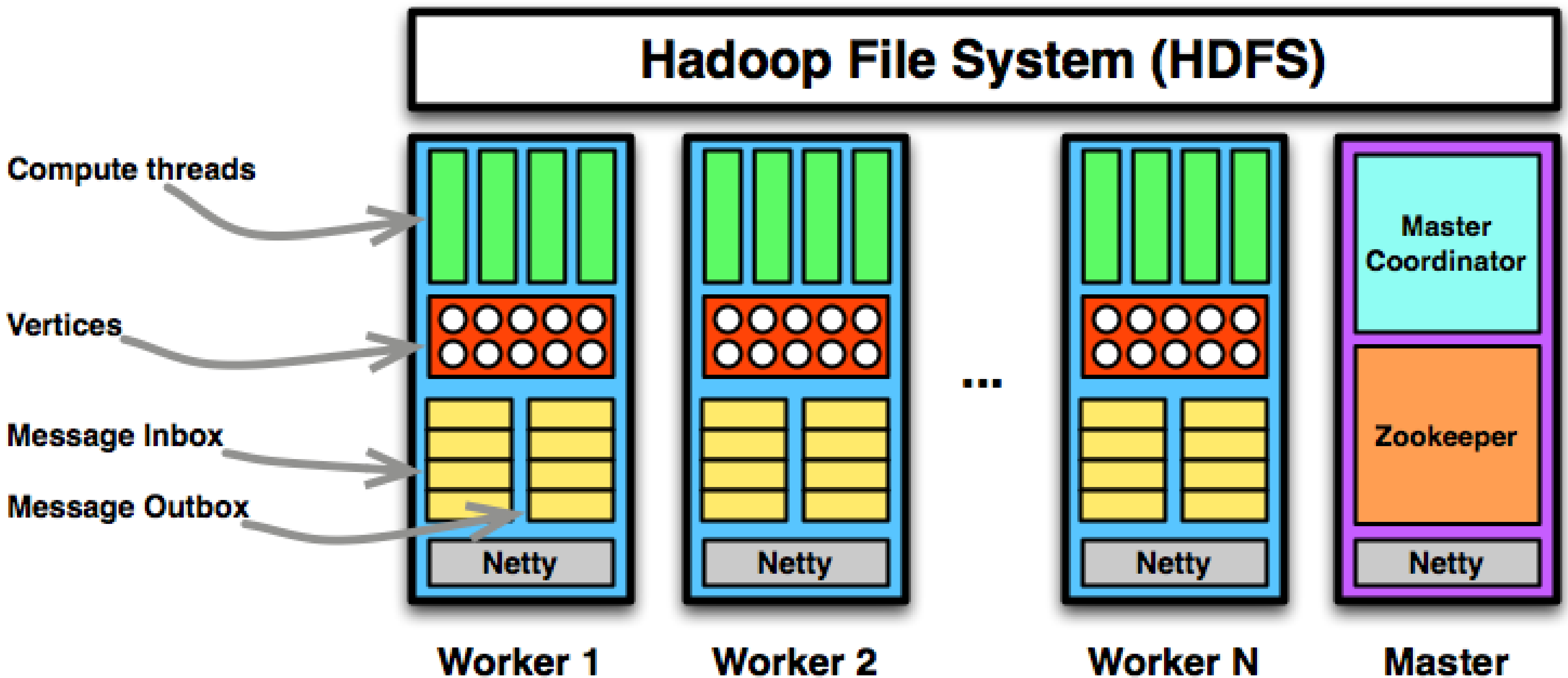
Designed for iterations

- Stateful (in-memory)
- Only intermediate values (messages) sent
- Hits the disk at input, output, checkpoint
- Can go out-of-core

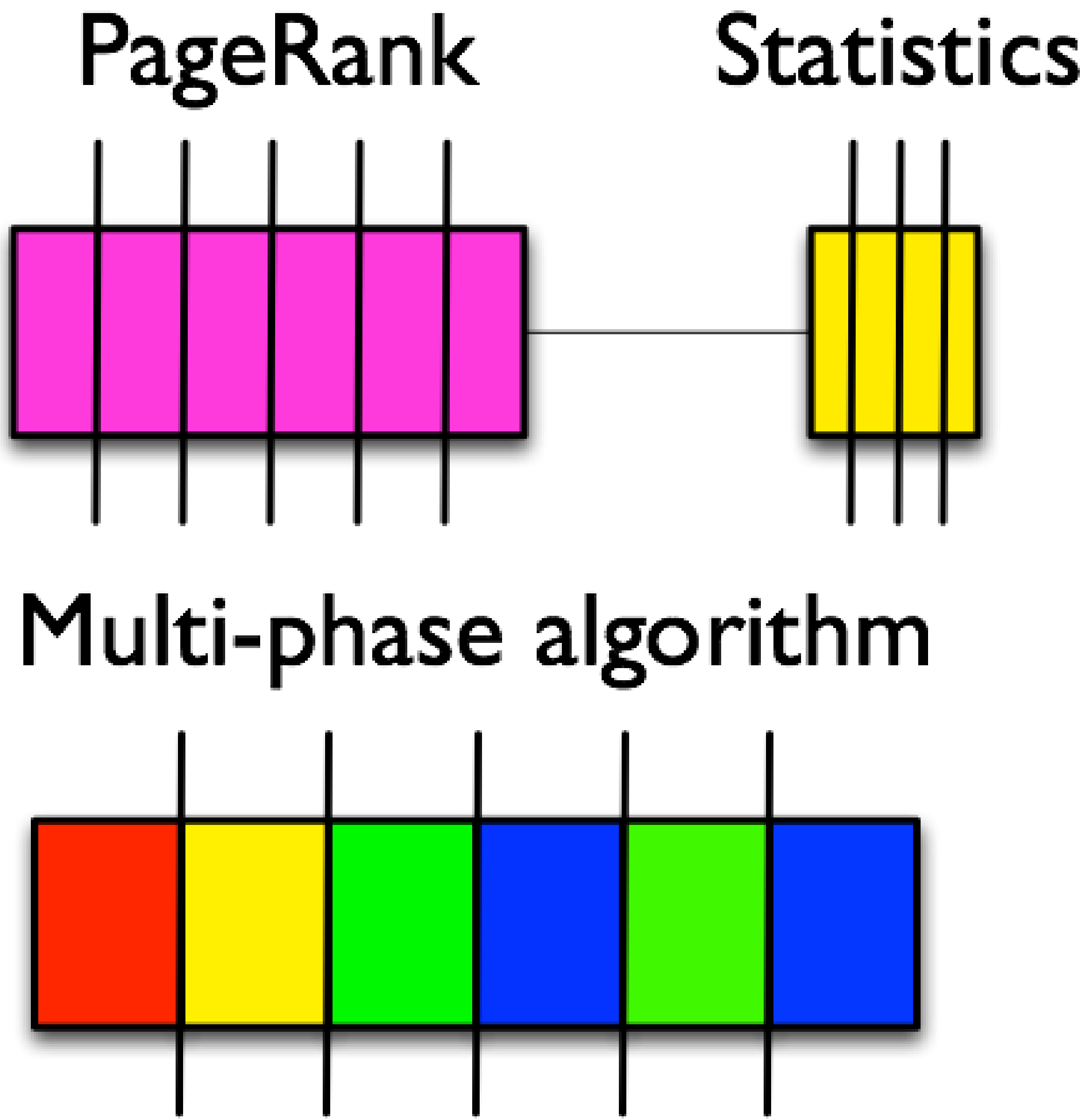
Giraph job lifetime



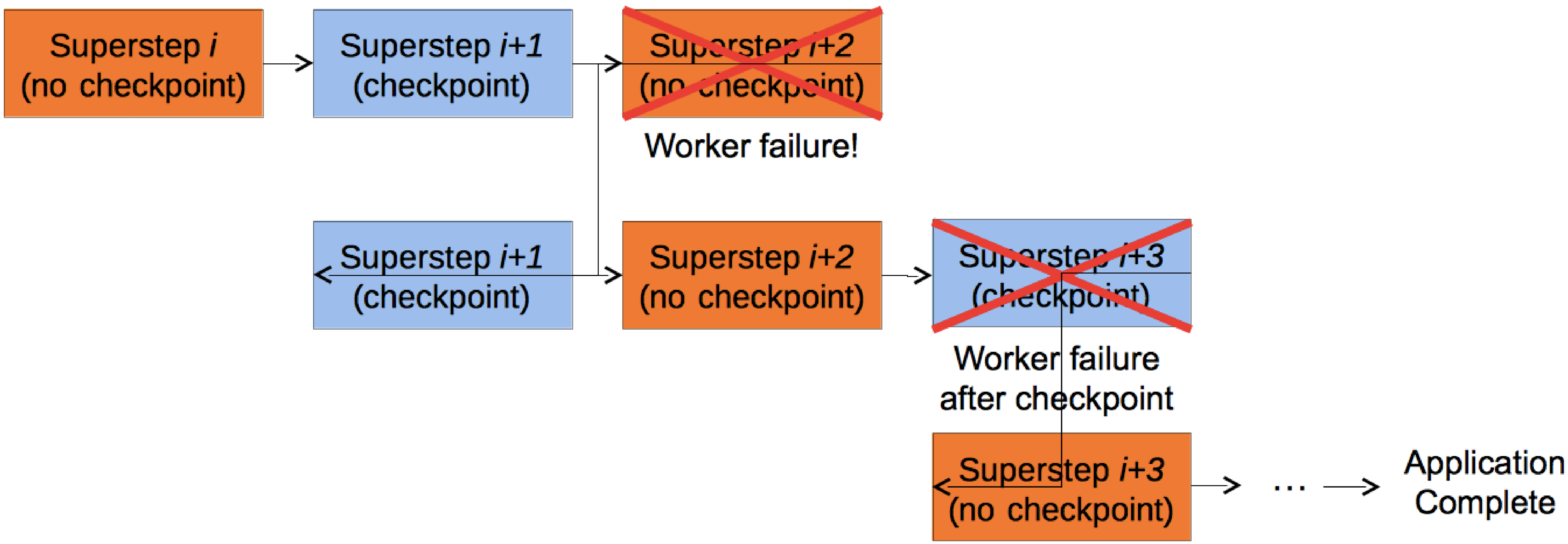
Architecture



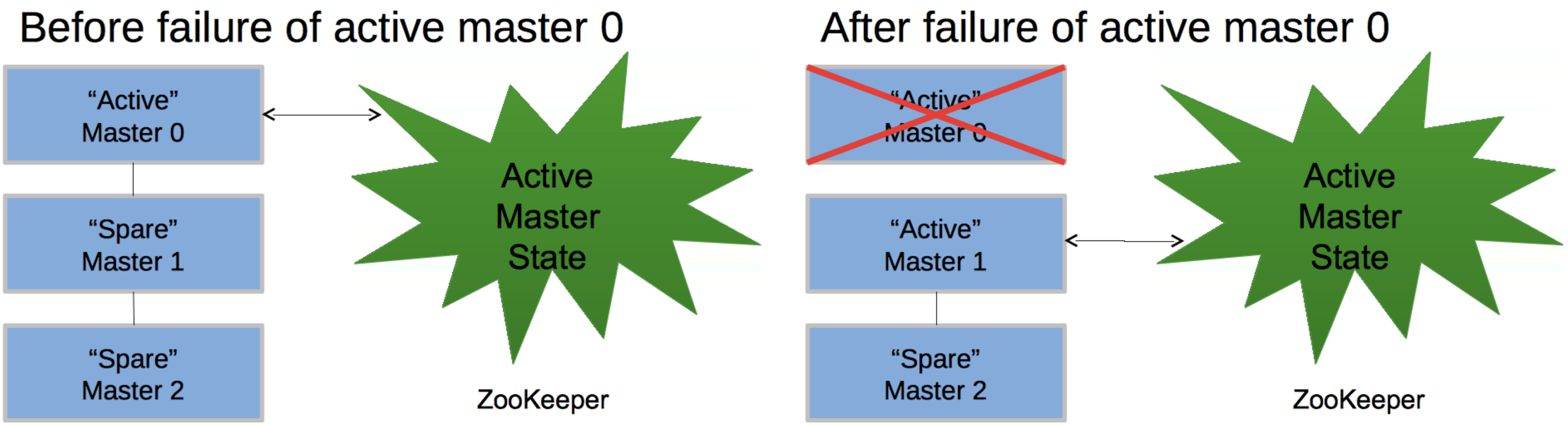
Composable API



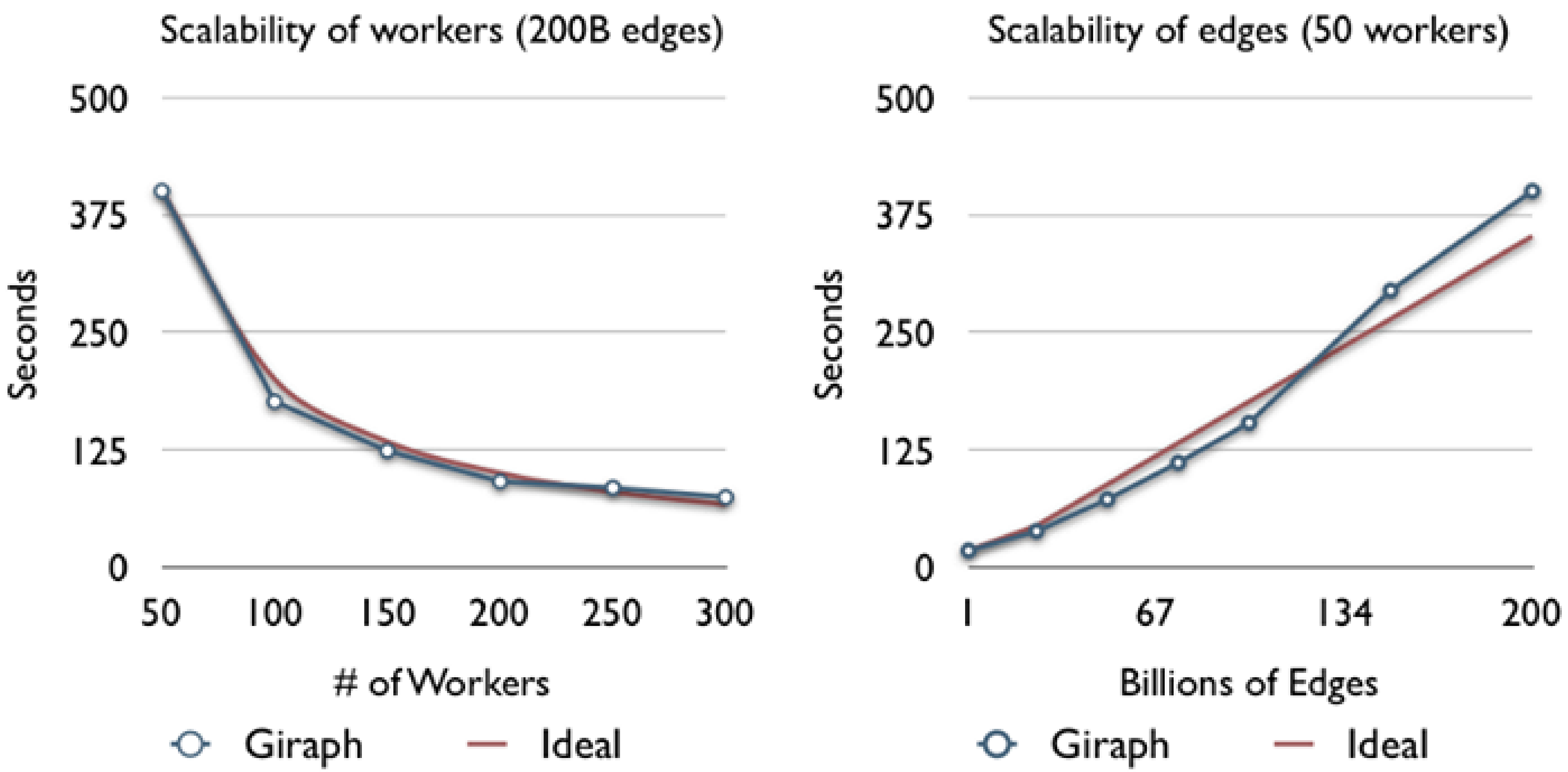
Checkpointing



No SPoFs


Before failure of active master 0
"Active"
Master 0
"Spare"
Master 1
"Spare"
Master 2
Active
Master
State
ZooKeeper
After failure of active master 0
"Active"
Master 0
"Active"
Master 1
"Spare"
Master 2
Active
Master
State
ZooKeeper

Giraph scales

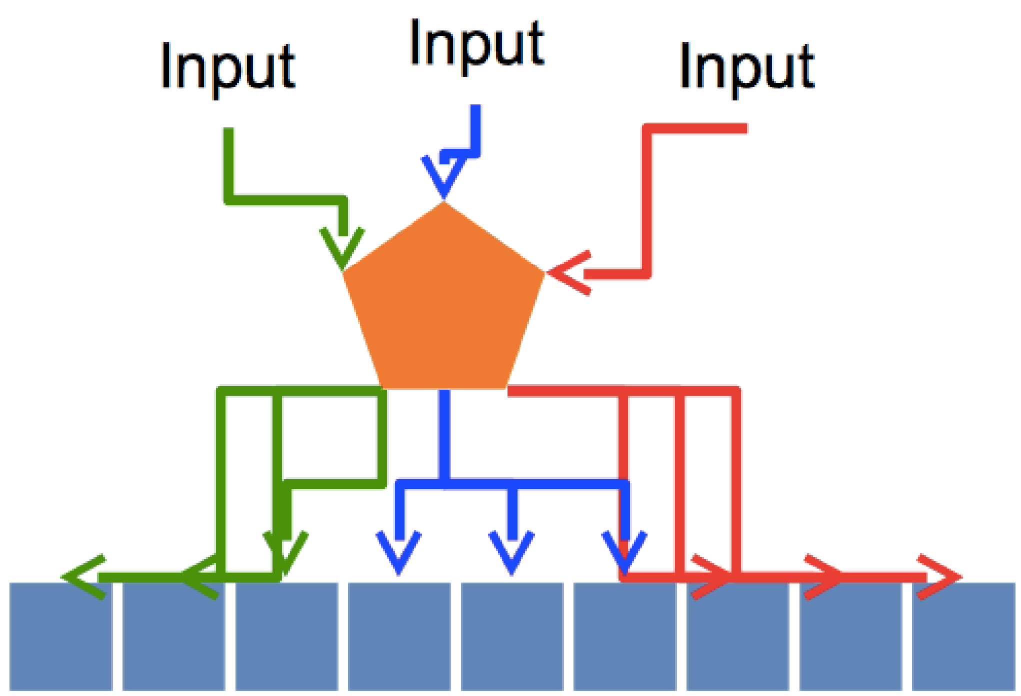


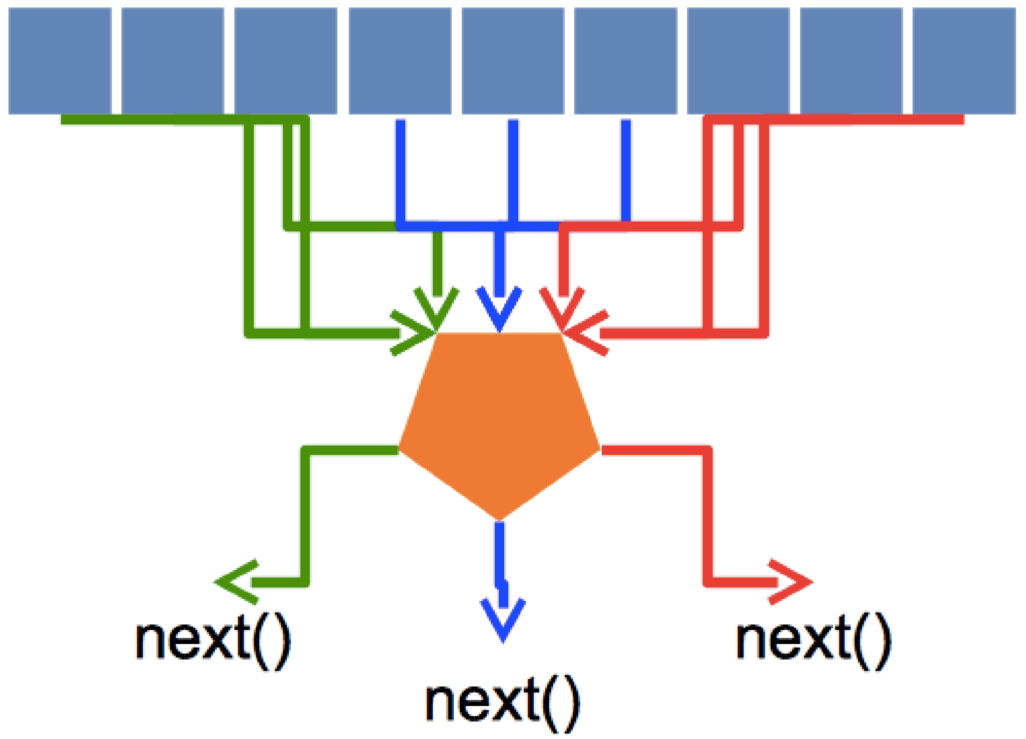
[ref: https://www.facebook.com/notes/facebook-engineering/scaling-apache-giraph-to-a-trillion-edges/10151617006153920](https://www.facebook.com/notes/facebook-engineering/scaling-apache-giraph-to-a-trillion-edges/10151617006153920)

Giraph is fast

- 100x over MR (Pr)
- jobs run within minutes
- given you have resources ;-)

Serialised objects

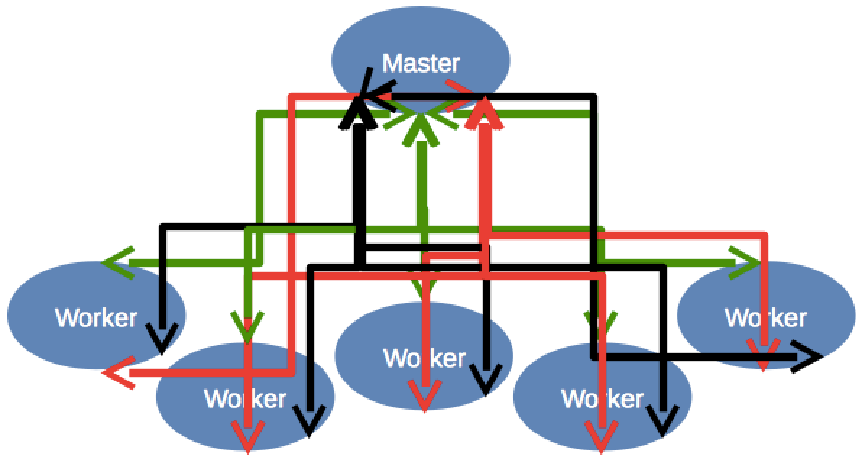




Primitive types

- Autoboxing is expensive
- Objects overhead (JVM)
- Use primitive types on your own
- Use primitive types-based libs (e.g. fastutils)

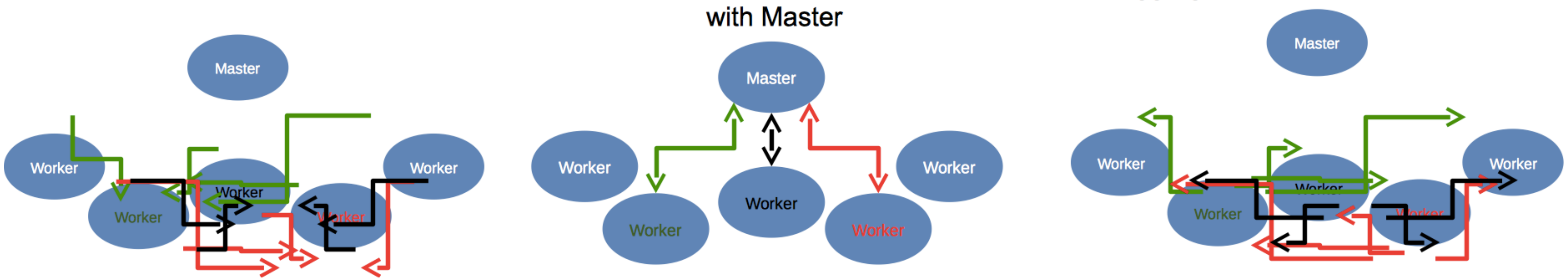
Sharded aggregators



Workers own aggregators

Aggregator owners communicate with Master

Aggregator owners distribute values



Okapi

- Apache Mahout for graphs
- Graph-based recommenders: ALS, SGD, SVD++, etc.
- Graph analytics: Graph partitioning, Community Detection, K-Core, etc.

Thank you

http://giraph.apache.org

<claudio@apache.org> @claudiomartella