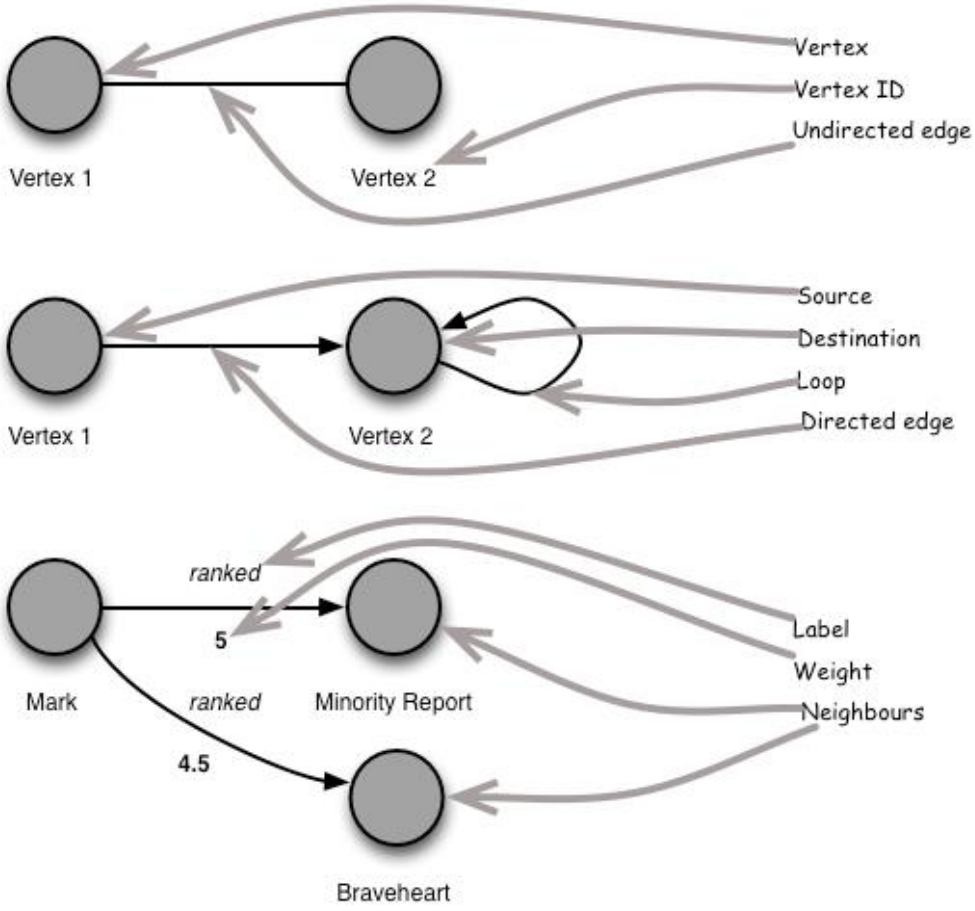
Πανεπιστήμιο Πατρών

Προχωρημένα Θέματα σε Κατανεμημένα Συστήματα

Graph Processing & Giraph

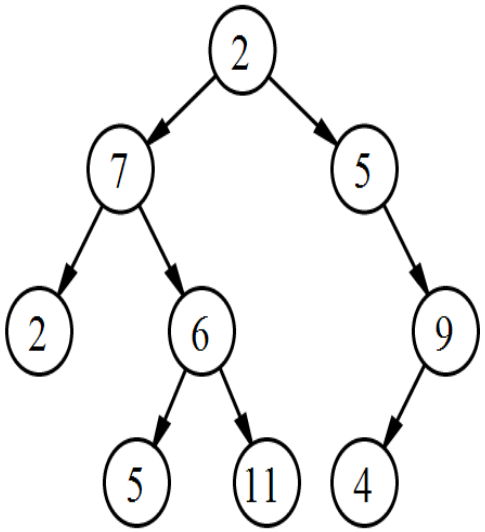
Graph Processing

Graphs are easy.

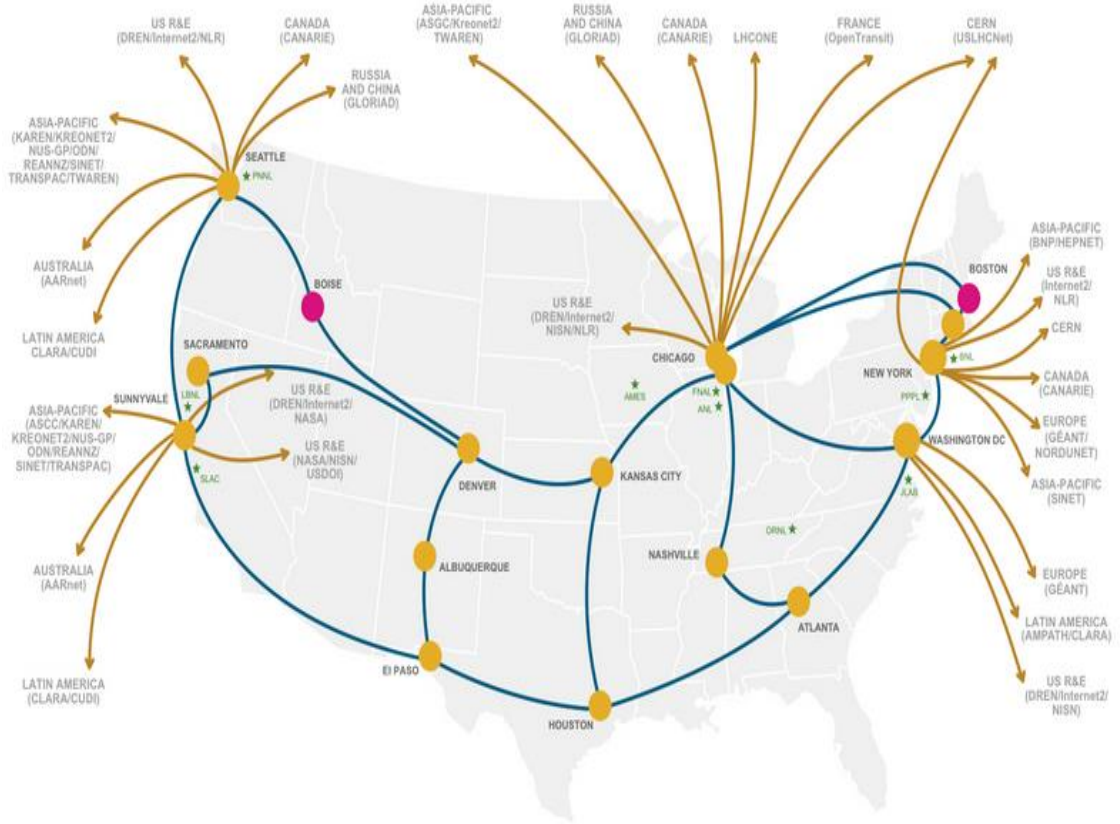


What is a graph?

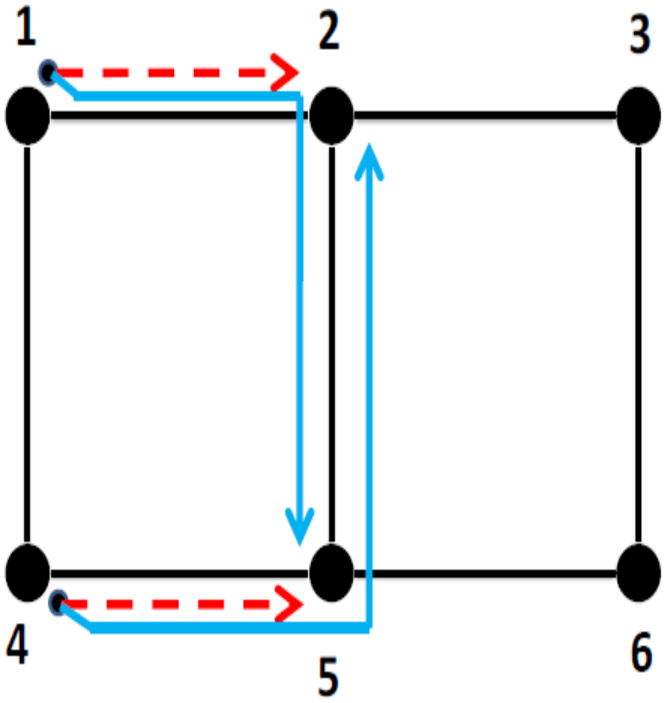
$$G = (V, E)$$



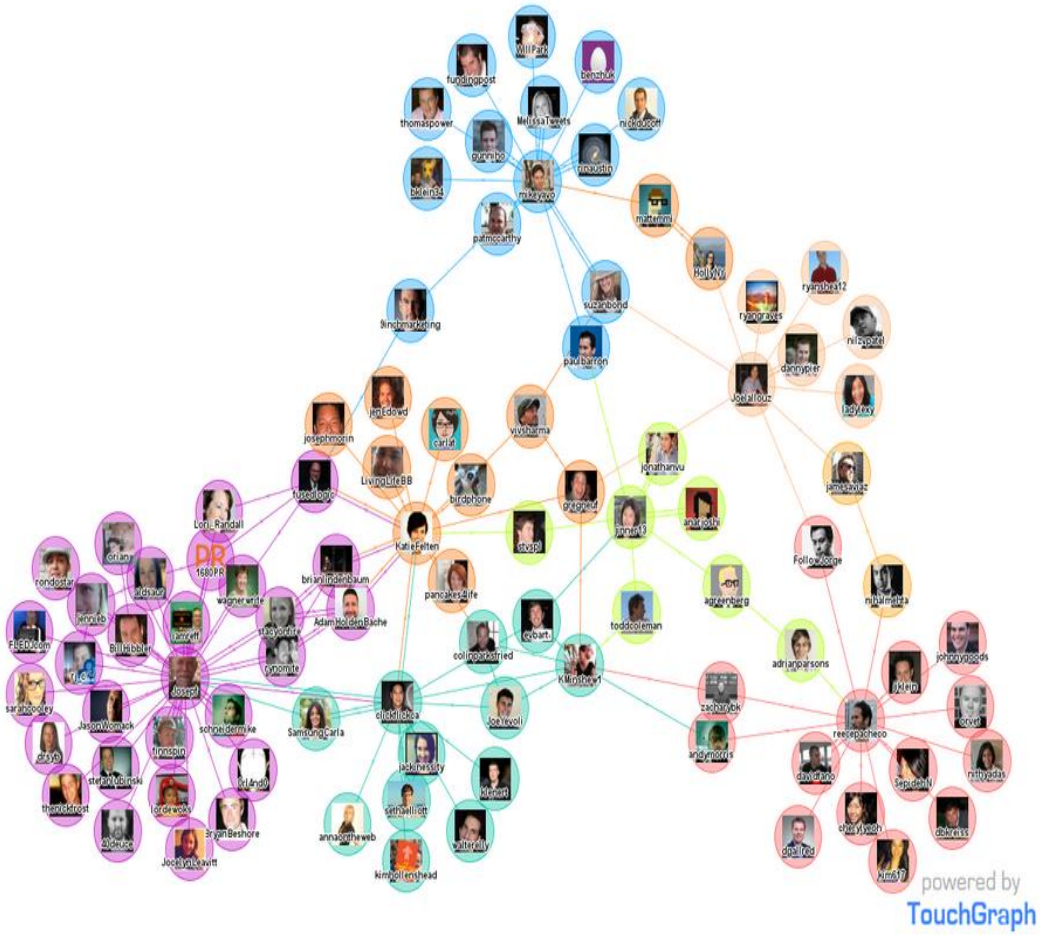
Binary Tree



Graph Problems

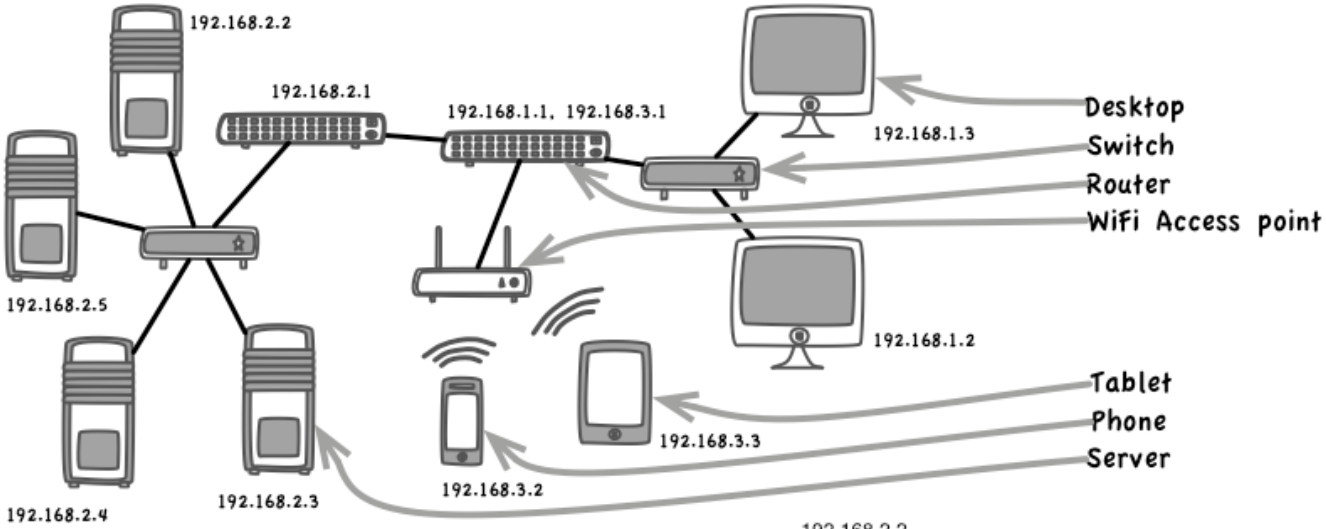


Network Routing



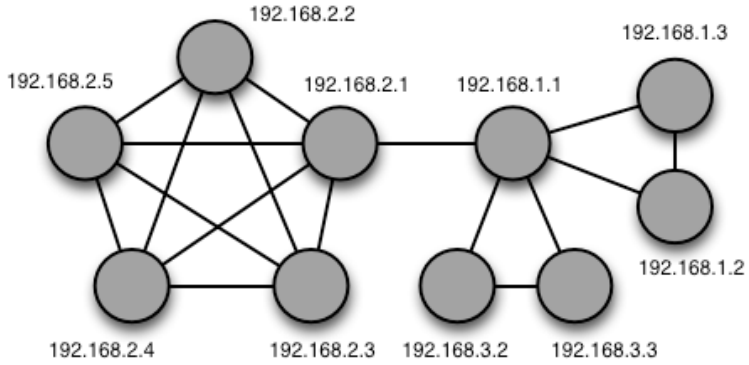
Social Network Connections

Example: A computer network

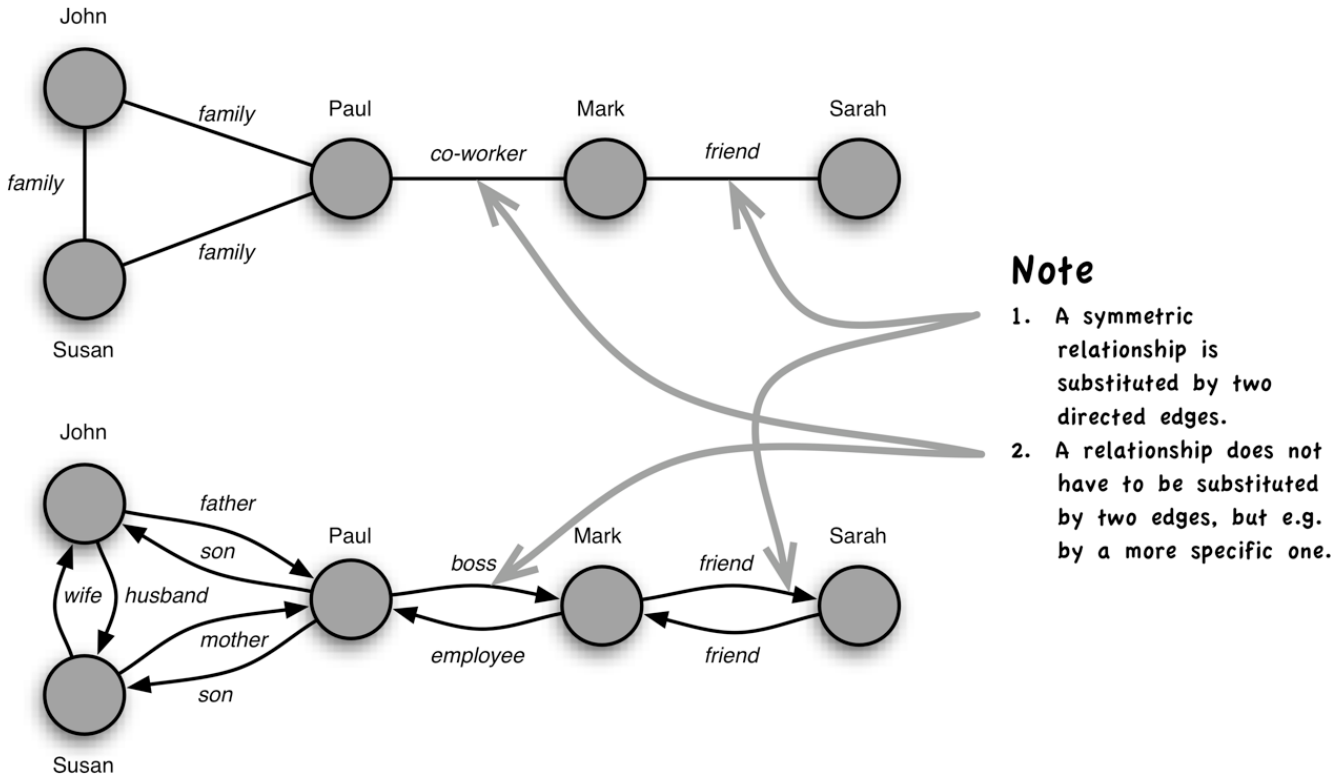


Note

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.

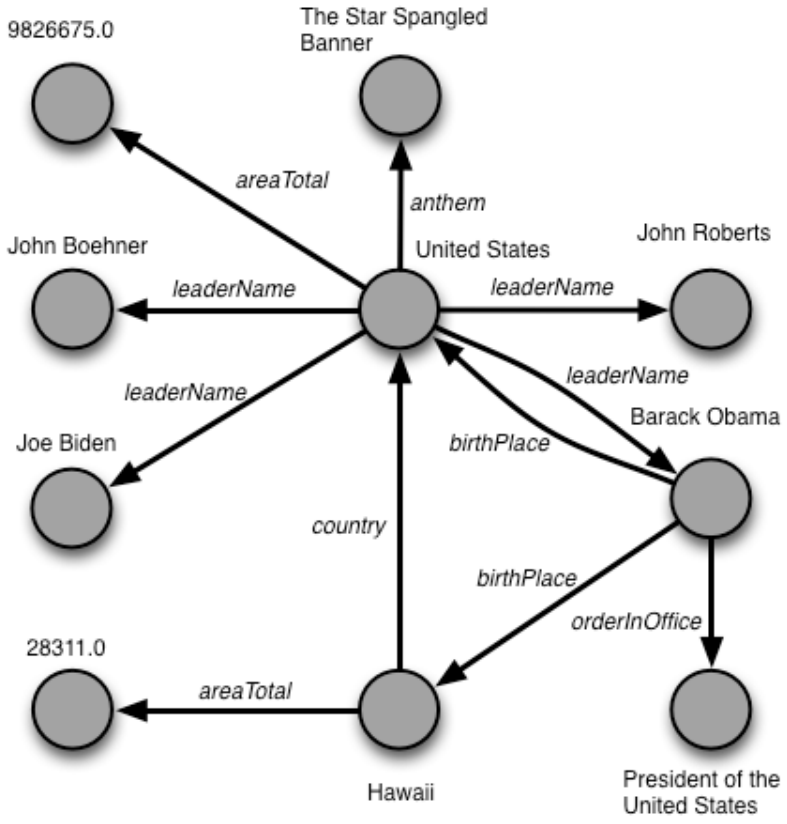


Example: A social network

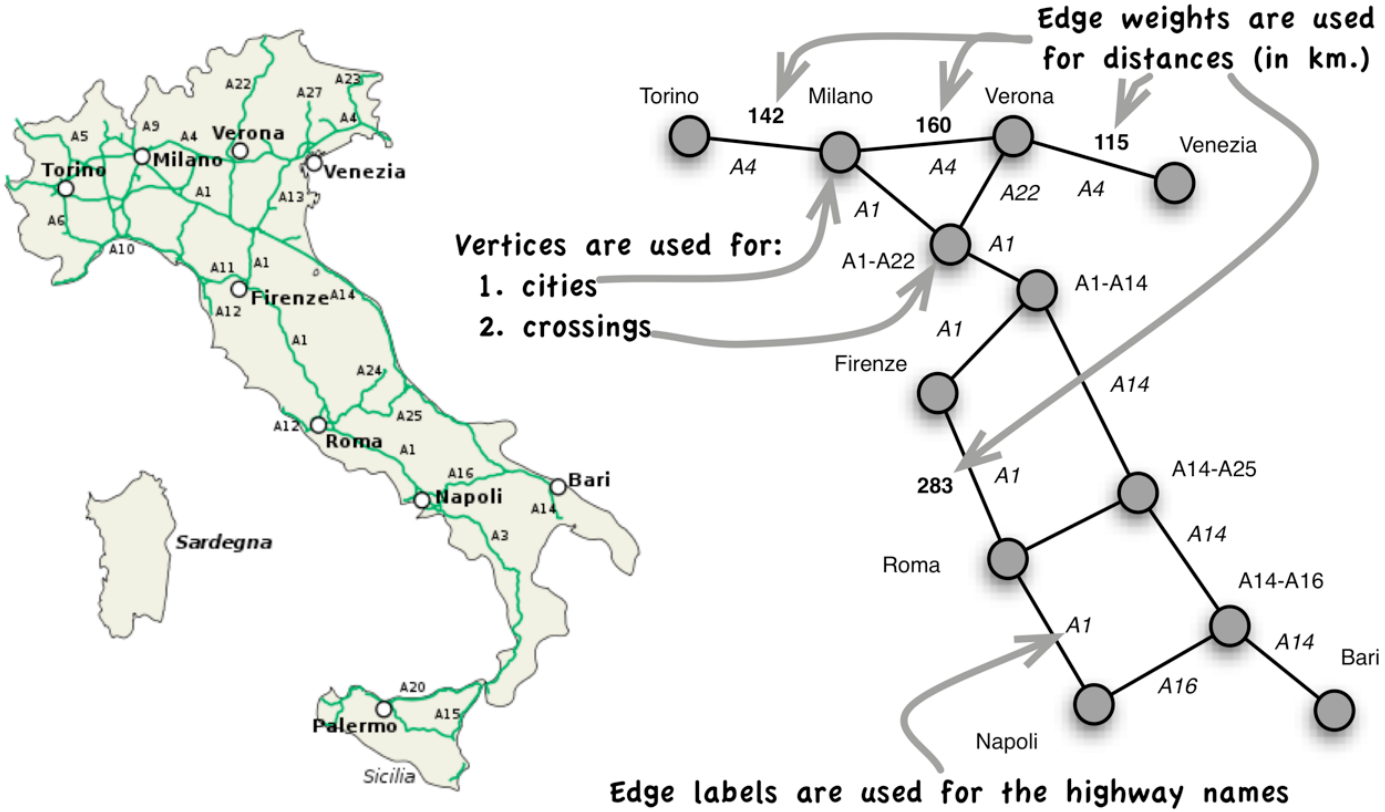


Example: A semantic network

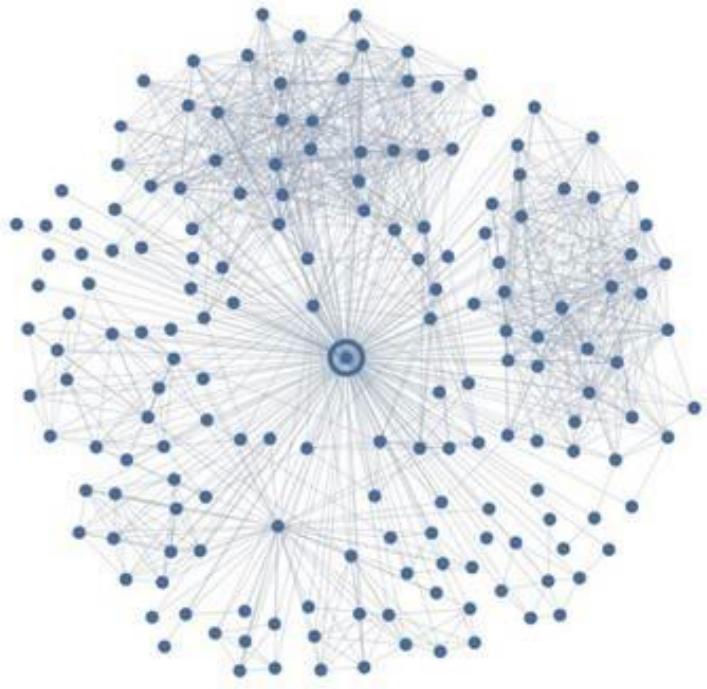
Subject	Predicate	Object
United States	areaTotal	9826675.0
United States	anthem	The Star Spangled Banner
United States	leaderName	Barack Obama
United States	leaderName	Joe Biden
United States	leaderName	John Boehner
United States	leaderName	John Roberts
Barack Obama	birthPlace	United States
Barack Obama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

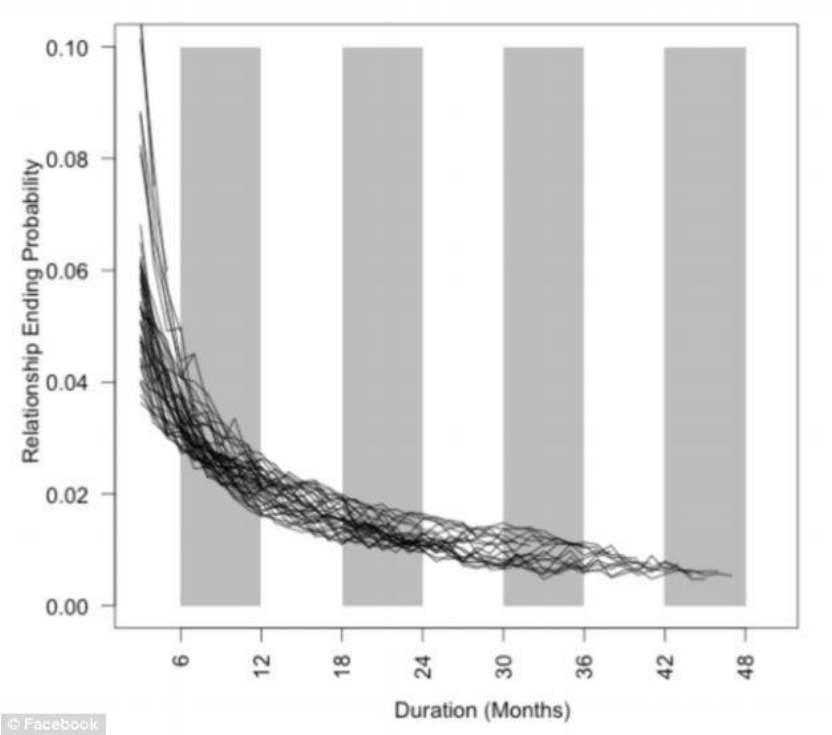


Example: A map



Predicting break ups





Graph approach

Aggregation approach

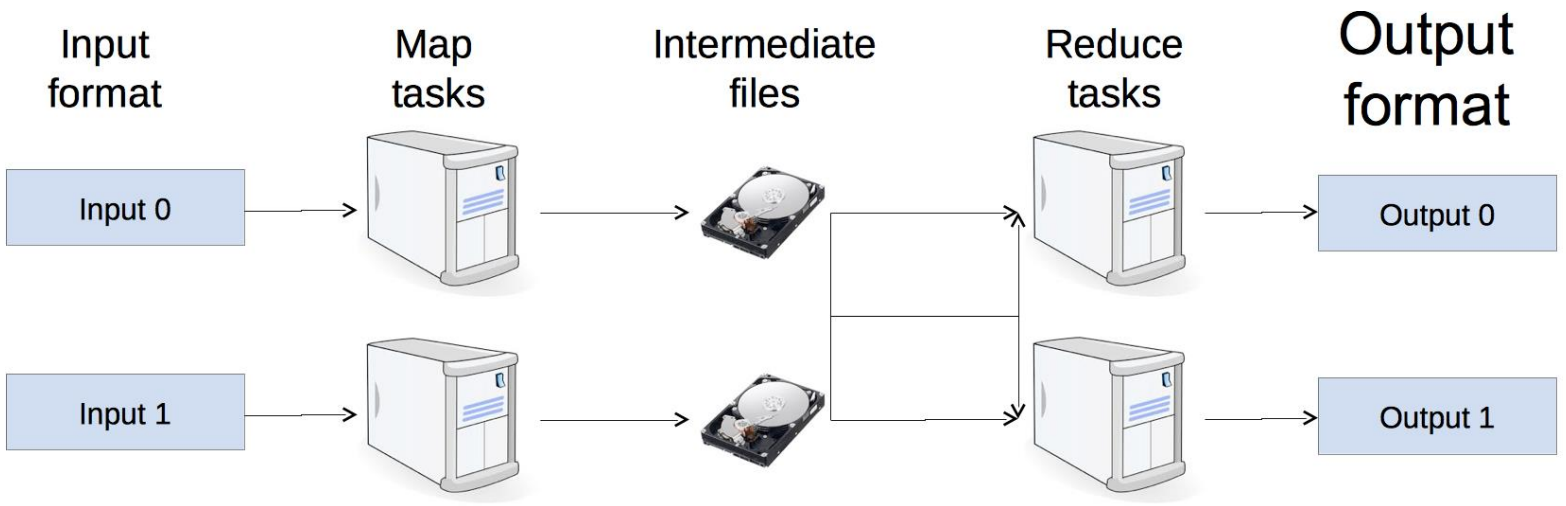
Graphs are nasty.

- Each vertex **depends** on its neighbors, **recursively**.
- **Recursive** problems are nicely solved **iteratively**.

PageRank in MapReduce

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

MapReduce dataflow



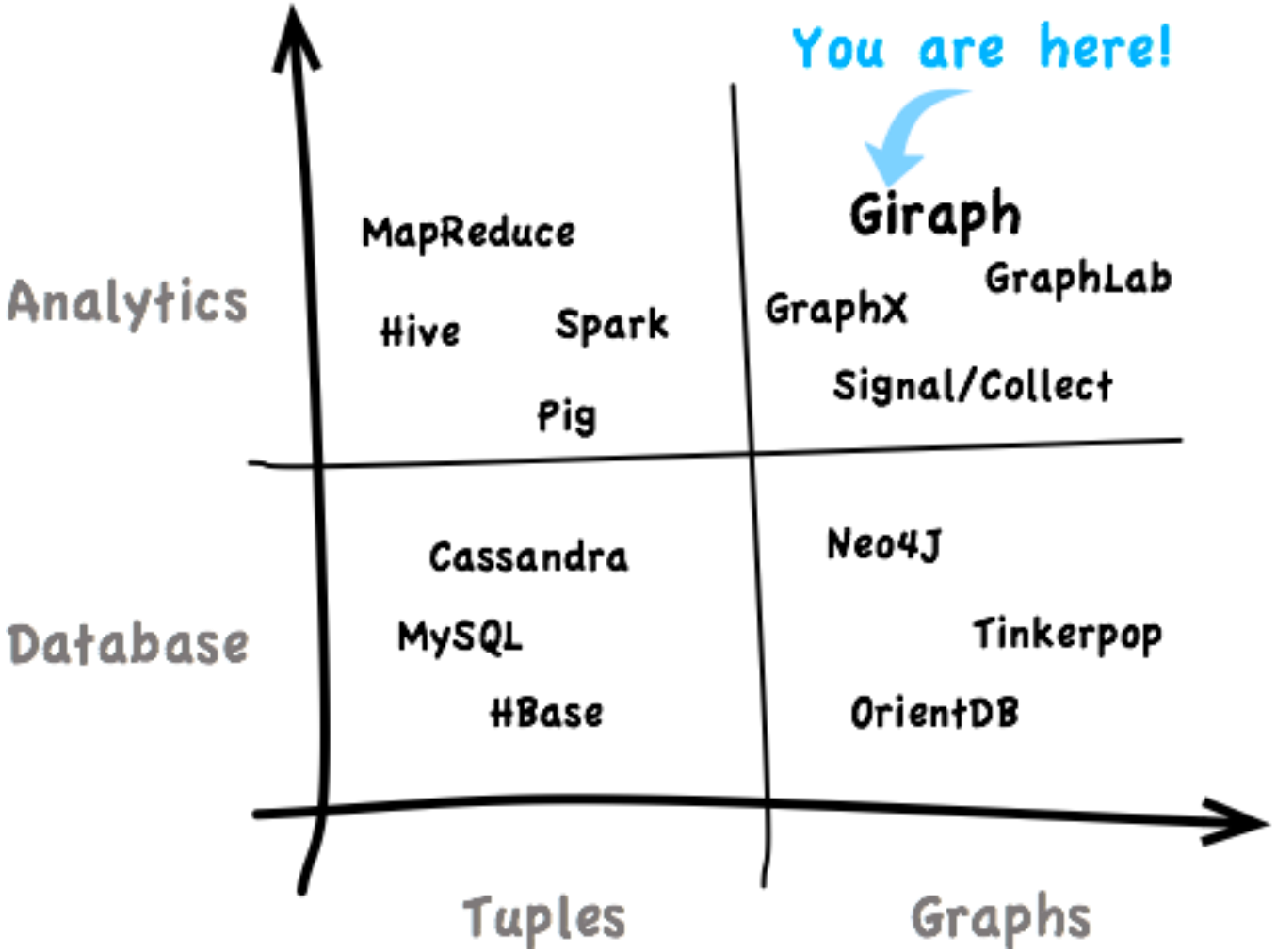
Drawbacks

- Each job is executed **N** times
  - Job **bootstrap**
- Mappers send PR values and **structure**
- Extensive **I/O** at input, shuffle & sort, output **in each iteration**!

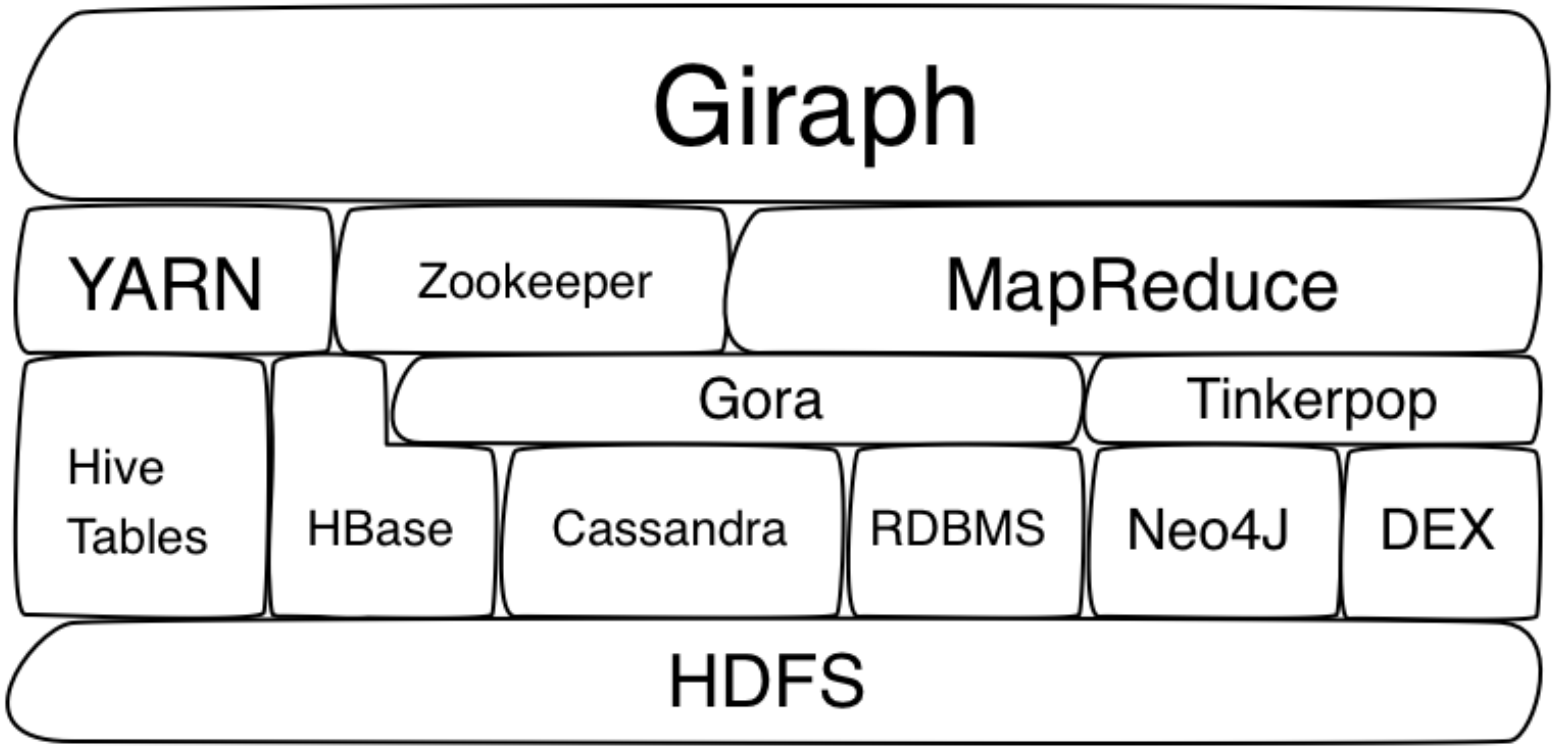
Apache Giraph

Apache Giraph

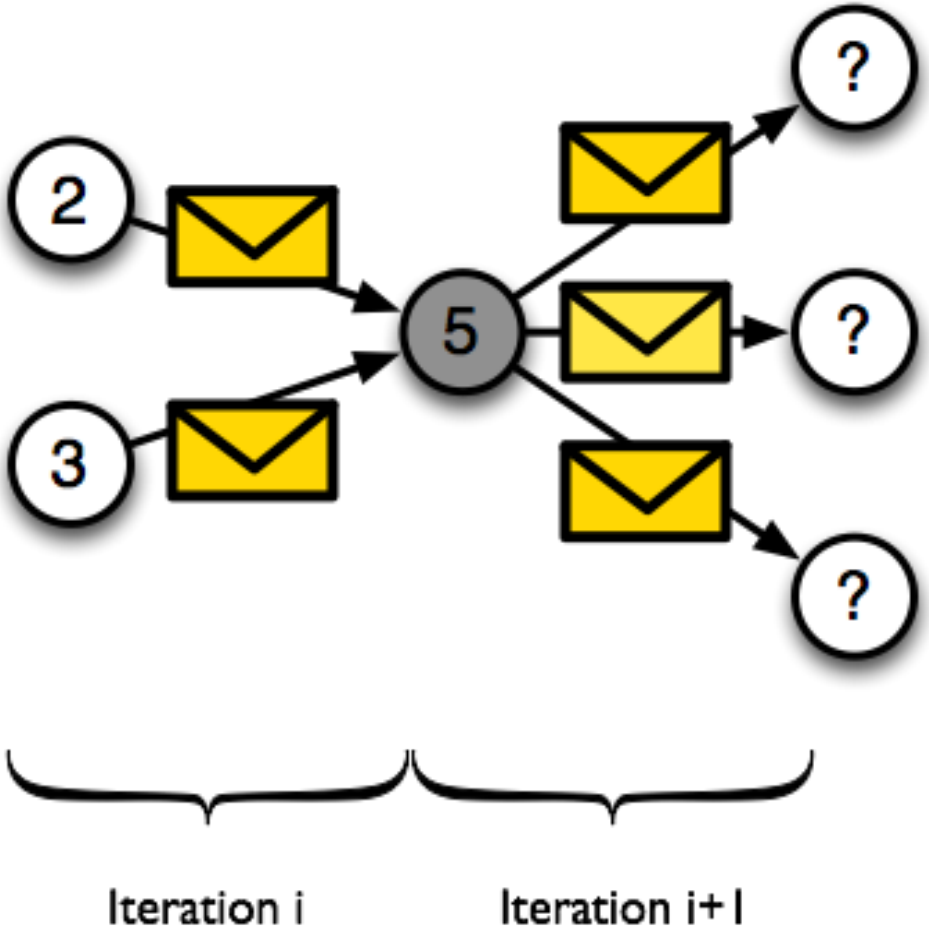
- Inspired by Google Pregel (2010)
- Donated to the Apache Software Foundation (ASF) by *Yahoo!* in 2011
- Top-level project in 2012
- 1.0 release in January 2013
- 1.1 release in November 2014


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

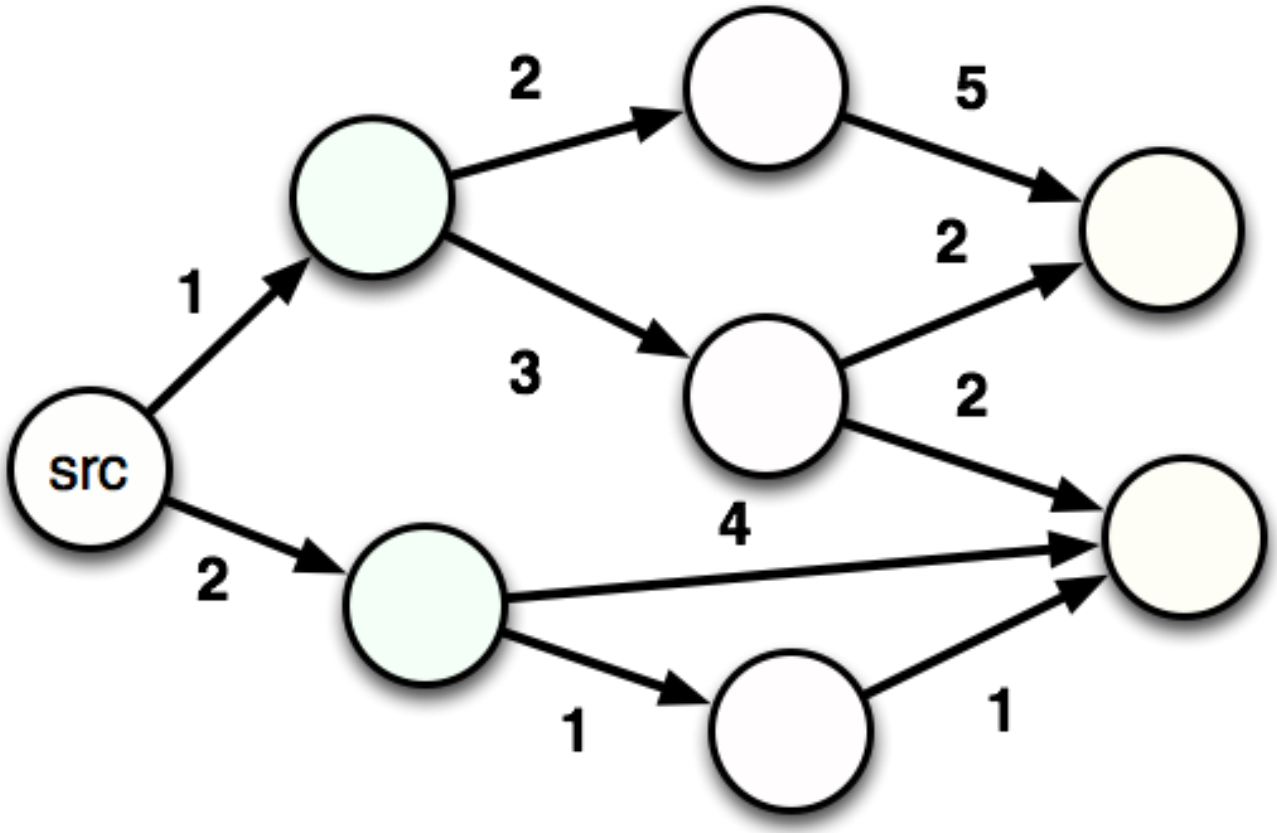
Plays well with Hadoop



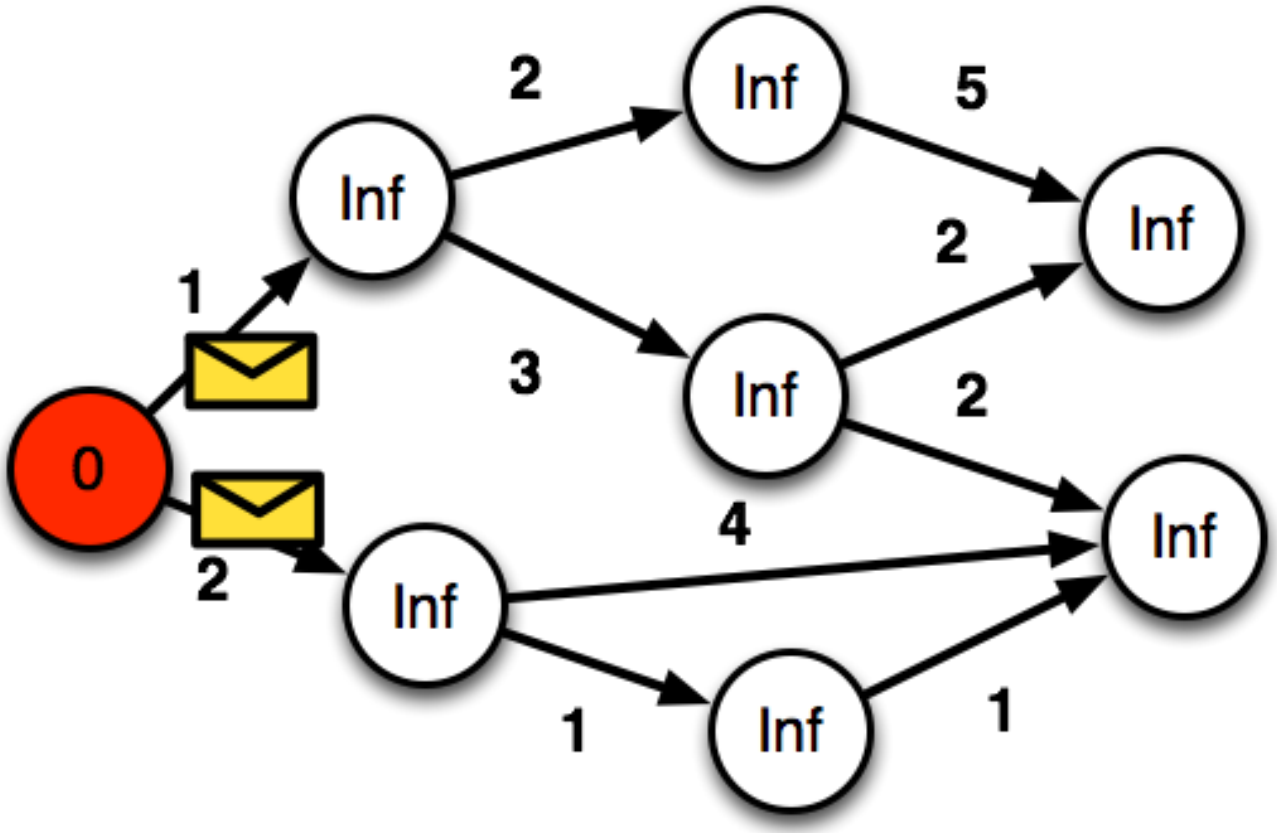
Vertex-centric API



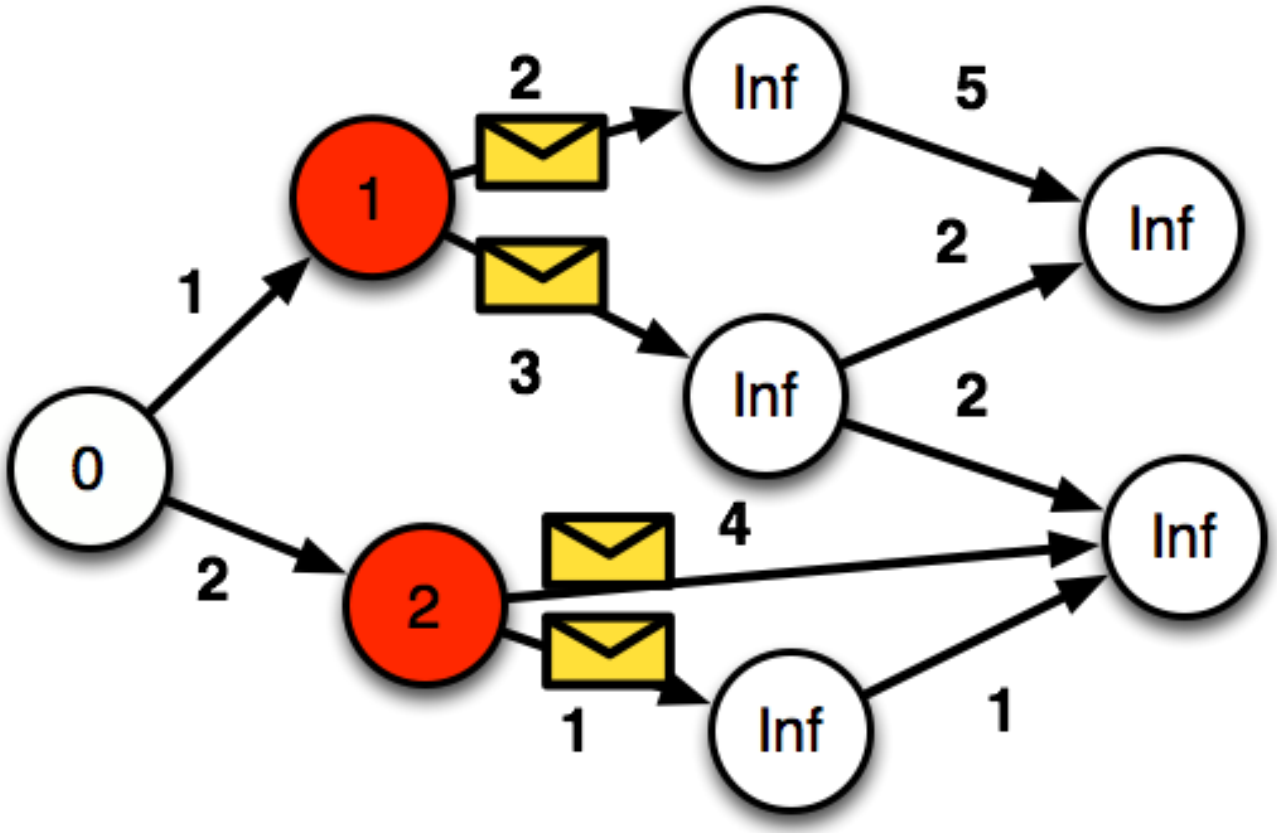
Shortest Paths



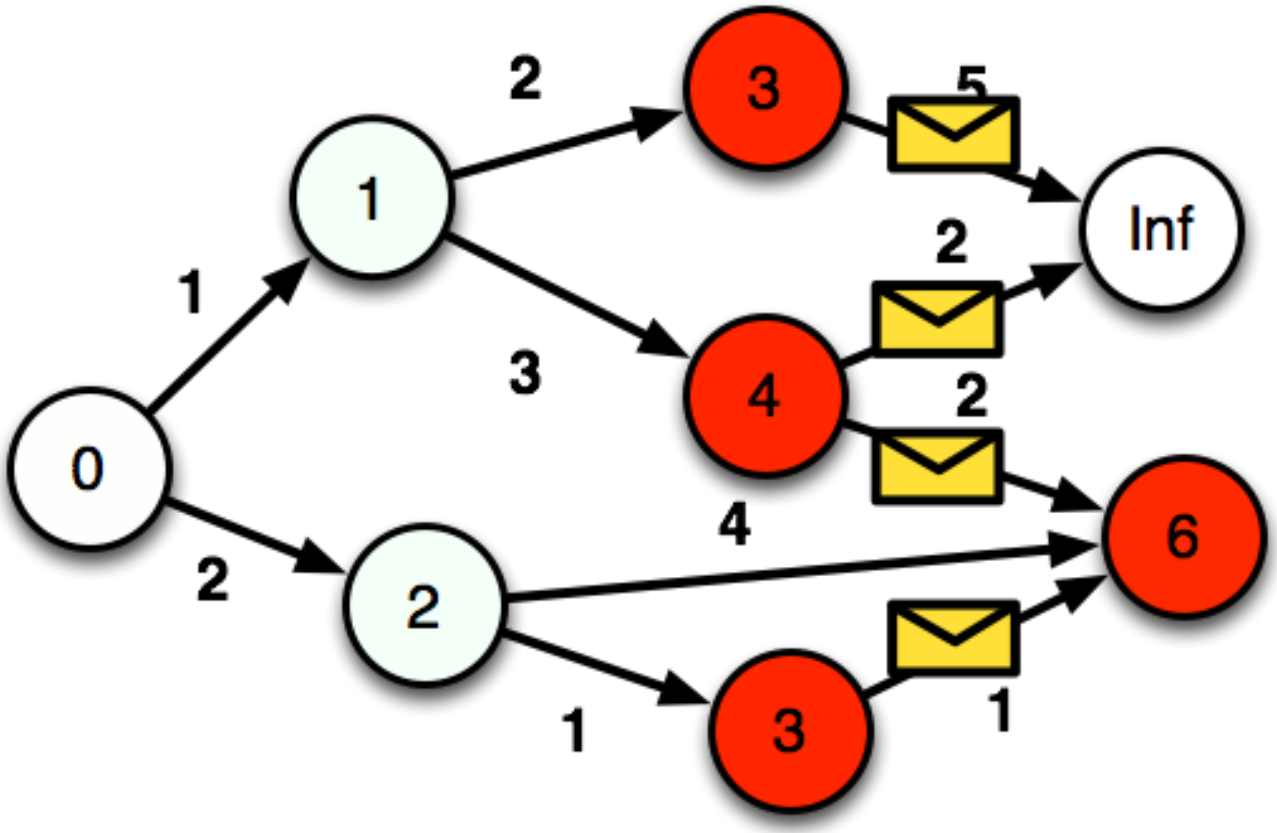
Shortest Paths



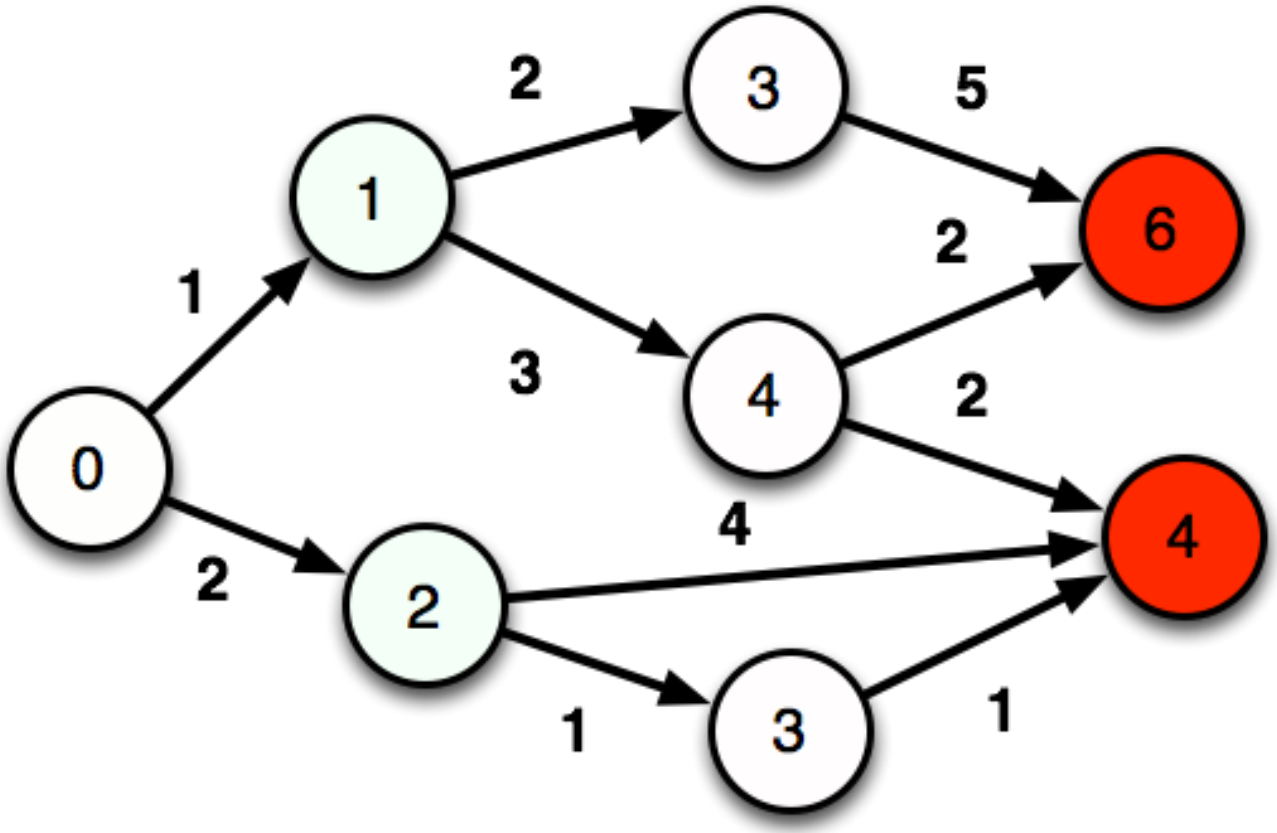
Shortest Paths



Shortest Paths



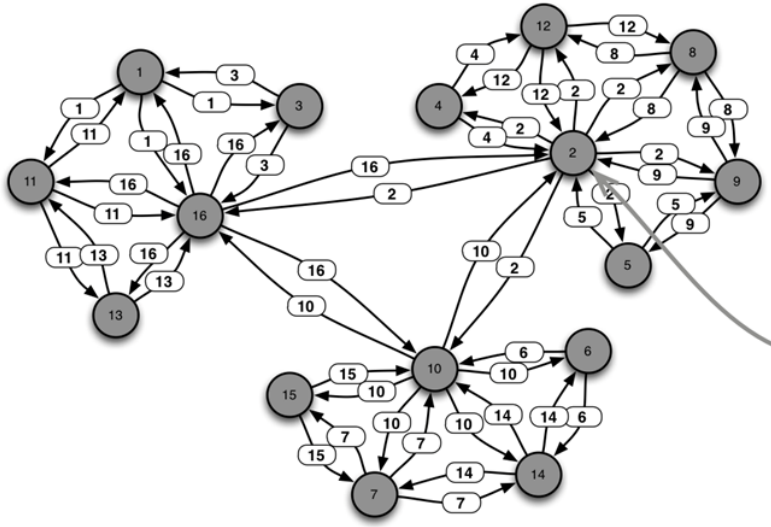
Shortest Paths



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()
```

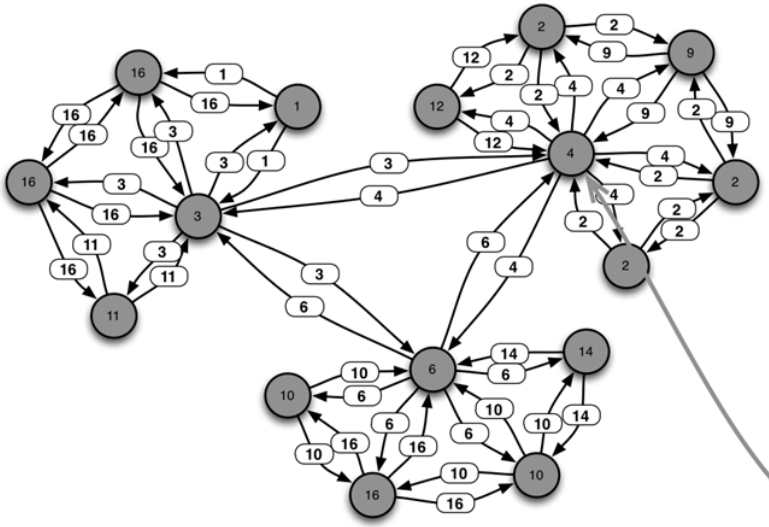
Finding Communities [1/2]



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.

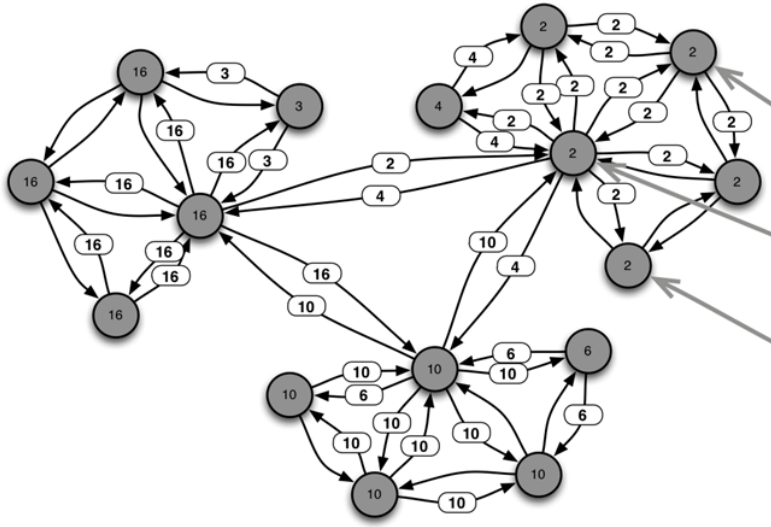


Superstep 1:

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

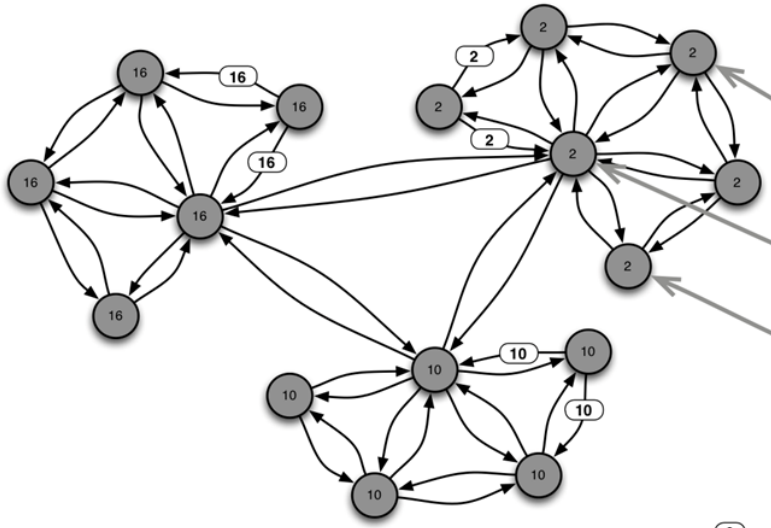
Finding Communities [2/2]



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:

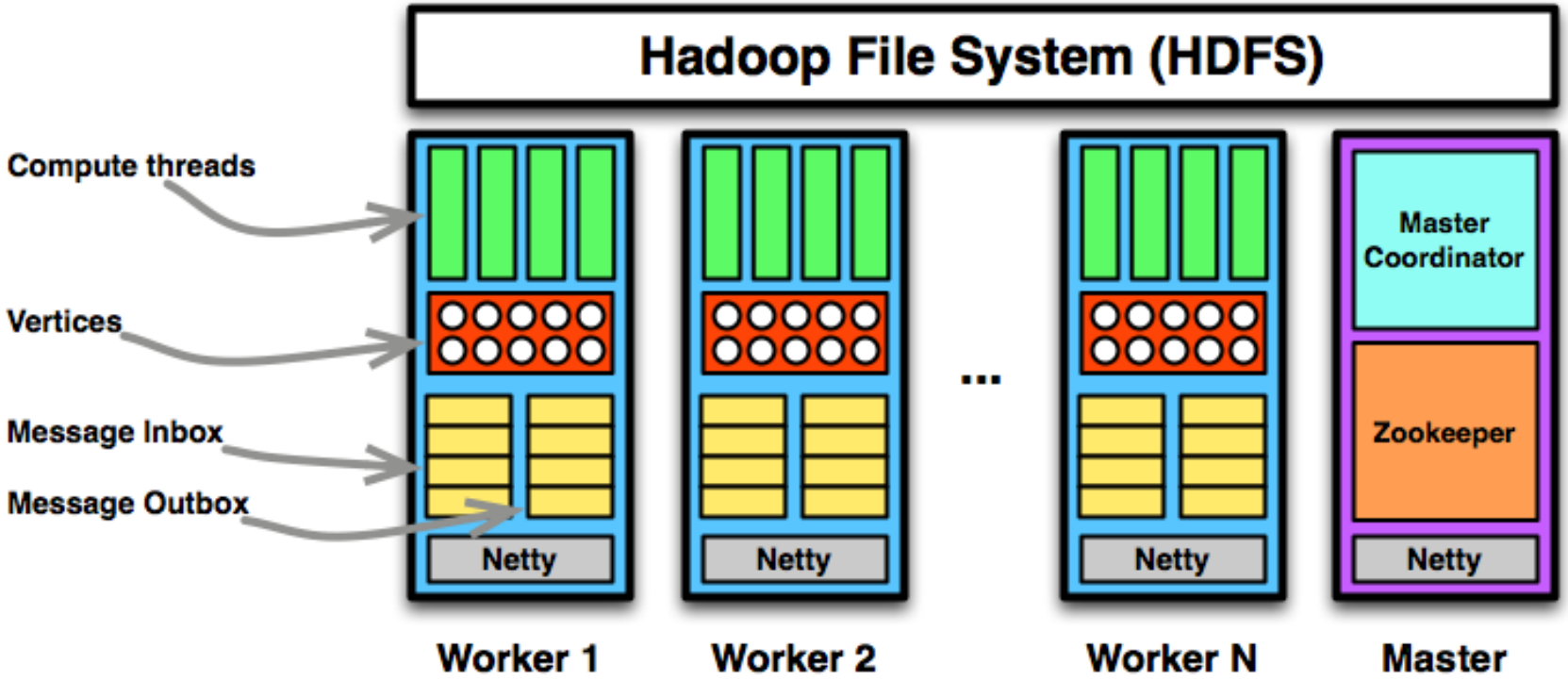
1. Remaining vertices receive the labels from the neighbours.
2. They update their edge values.
3. They compute their new labels.
4. They send their new value to the endpoints of their edges.

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

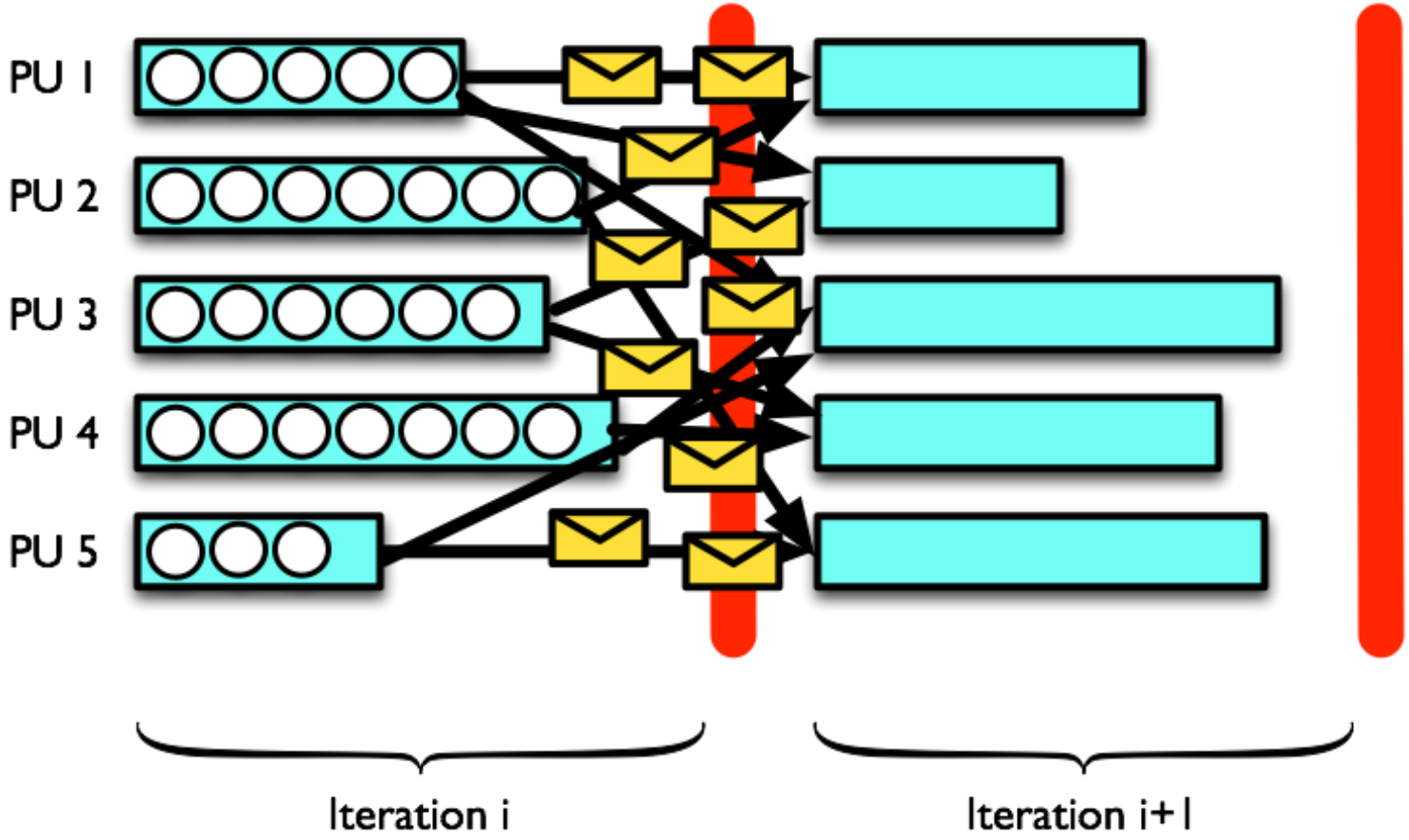
6 Label as a message

6 Active vertex with label

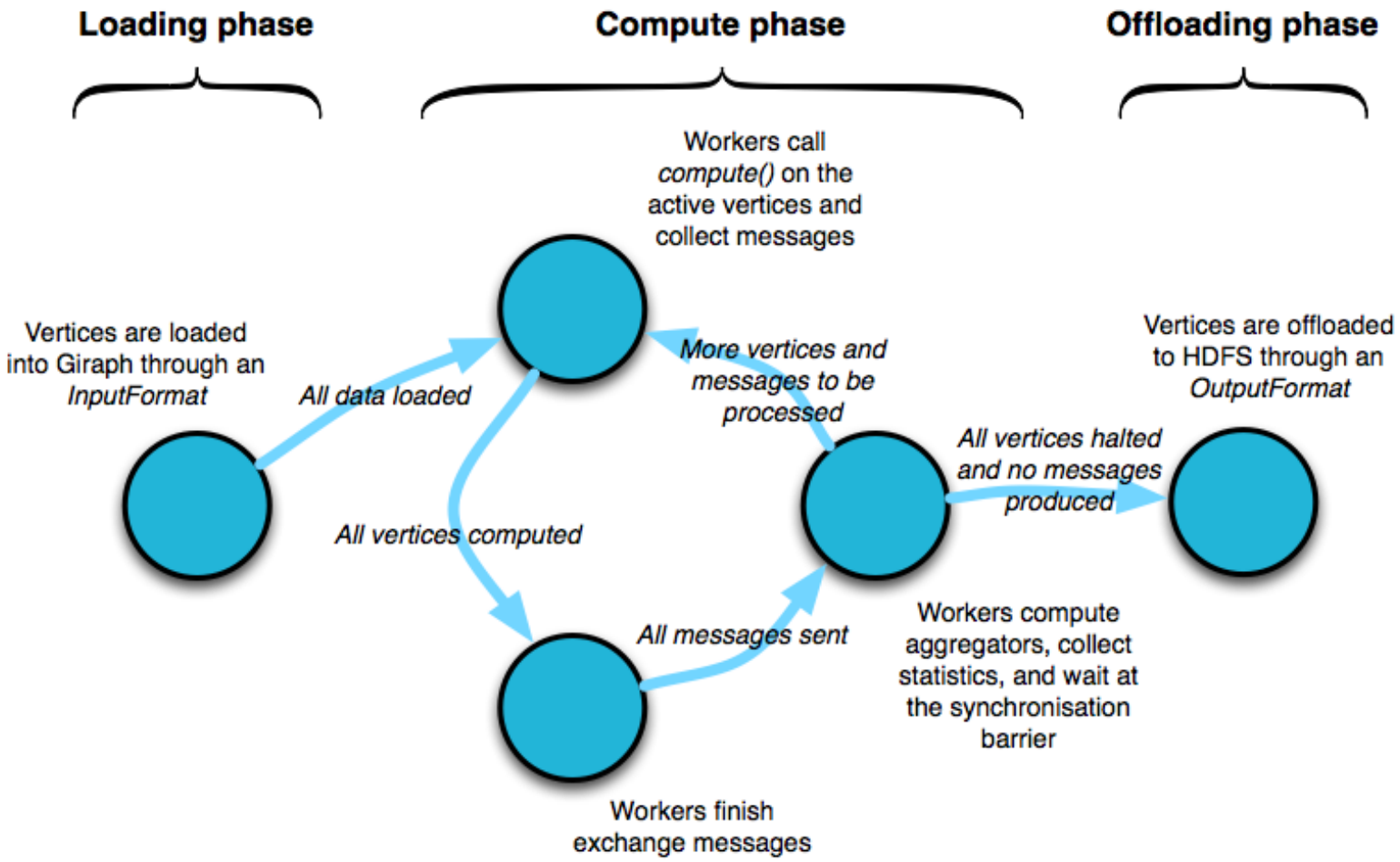
Architecture



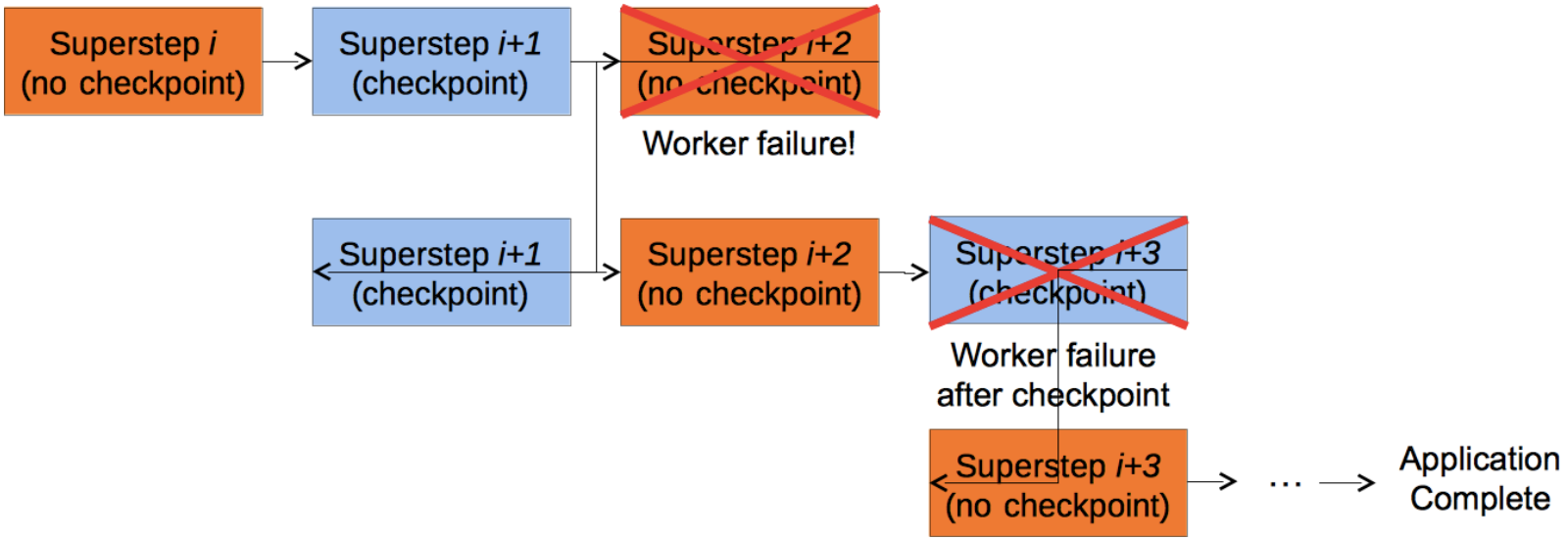
Bulk Synchronous Parallel & Giraph



Giraph job lifetime

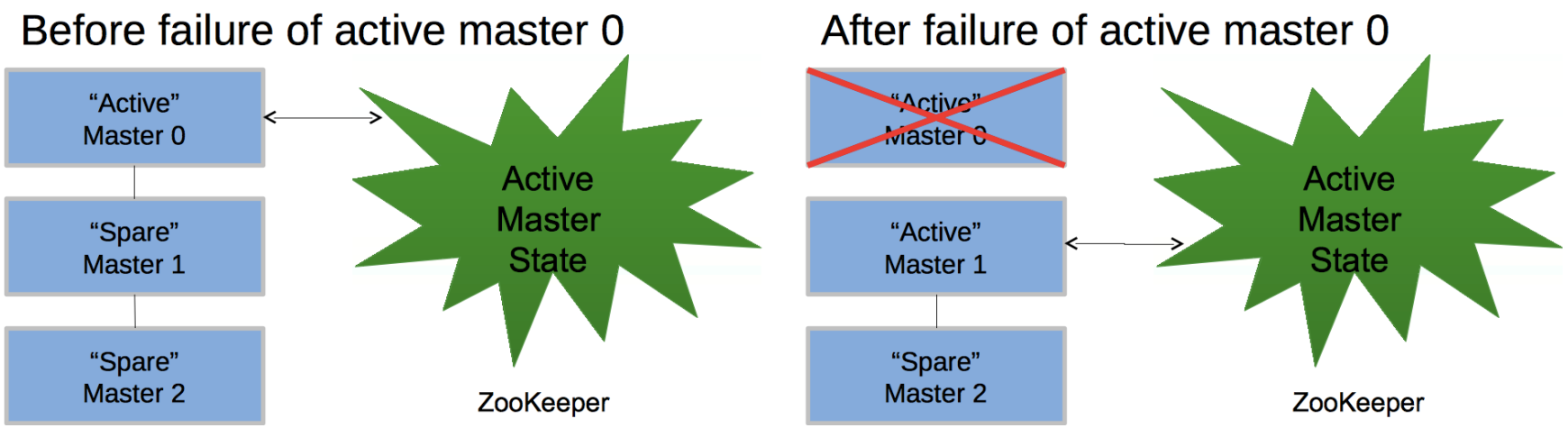


Checkpointing

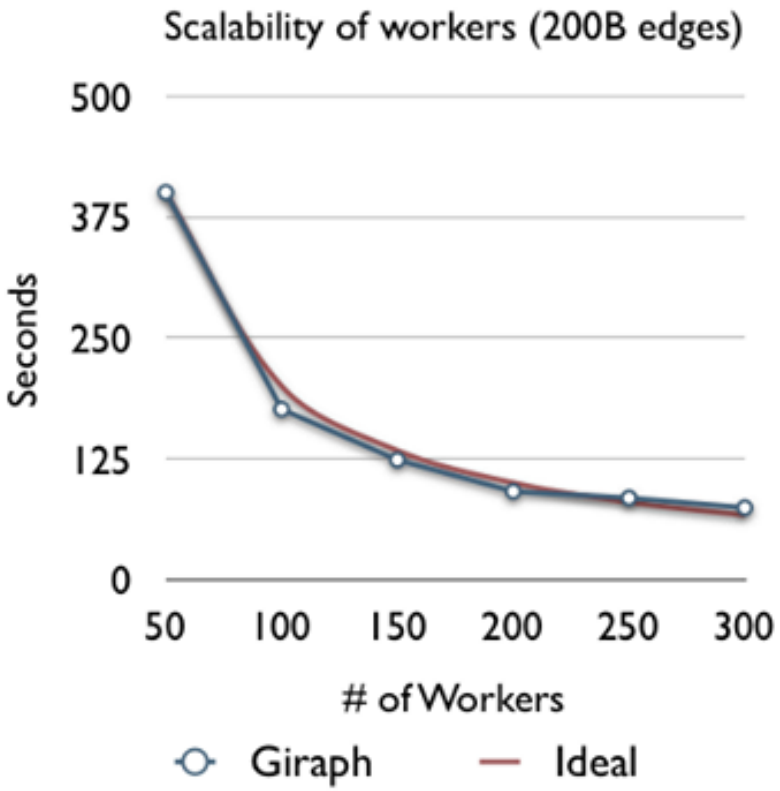


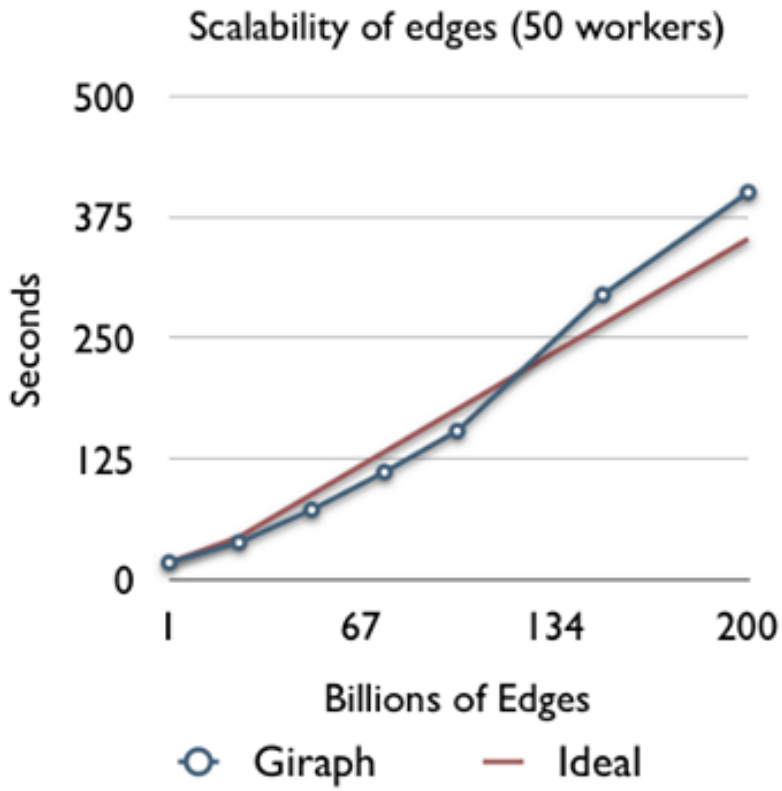
No SPoFs (*)

(*) SPoF = **S**ingle **P**oint **o**f **F**ailure



Giraph scales



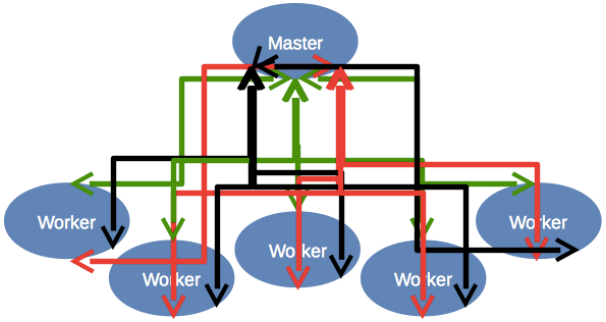


ref: https://www.facebook.com/notes/facebook-engineering/scaling-apache-giraph-to-a-trillion-edges/10151617006153920

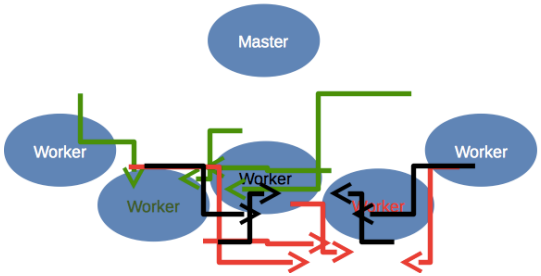
Giraph is fast

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

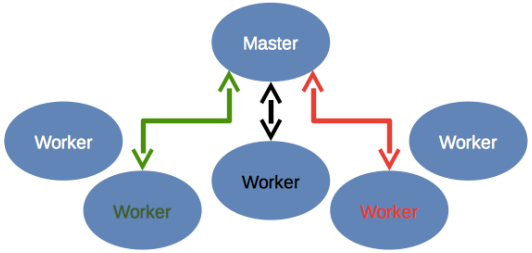
Aggregators



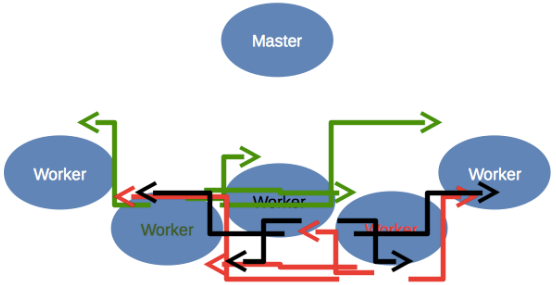
Workers own aggregators



Aggregator owners communicate with Master



Aggregator owners distribute values



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