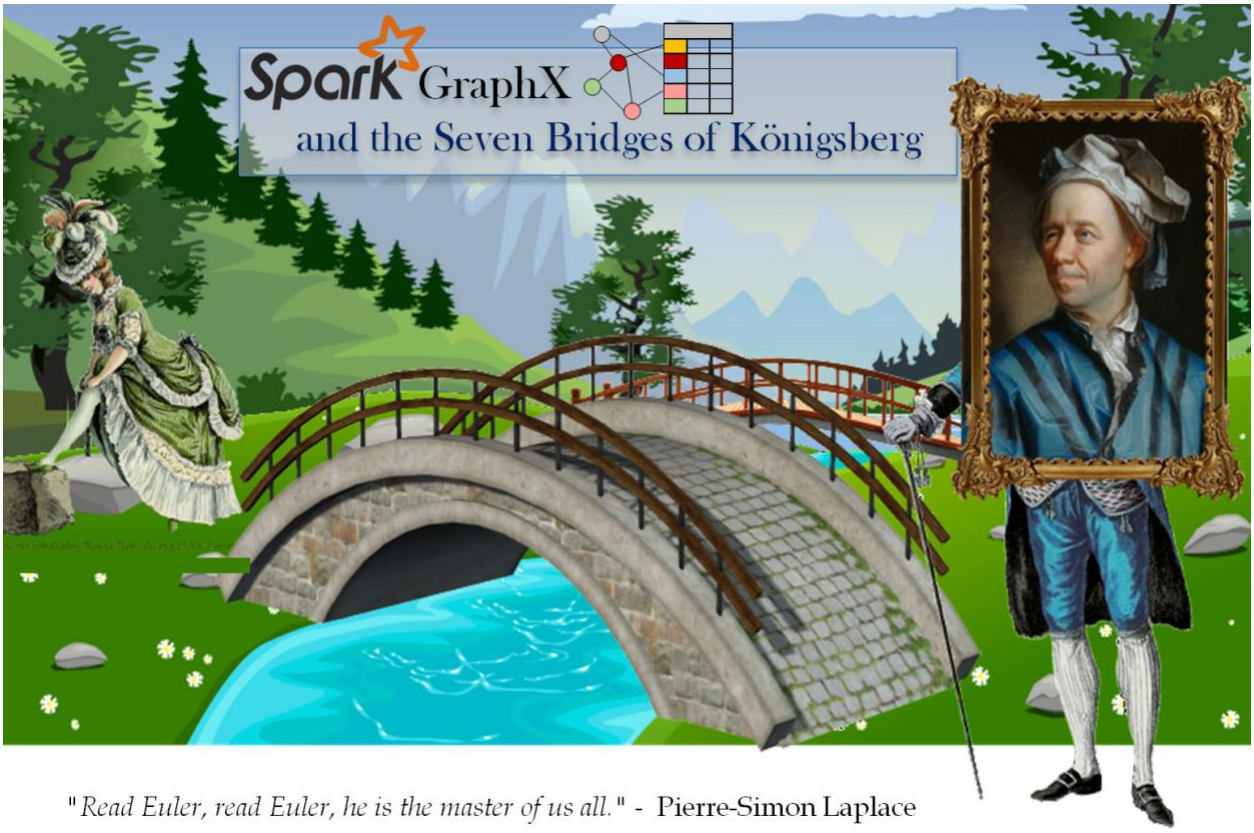
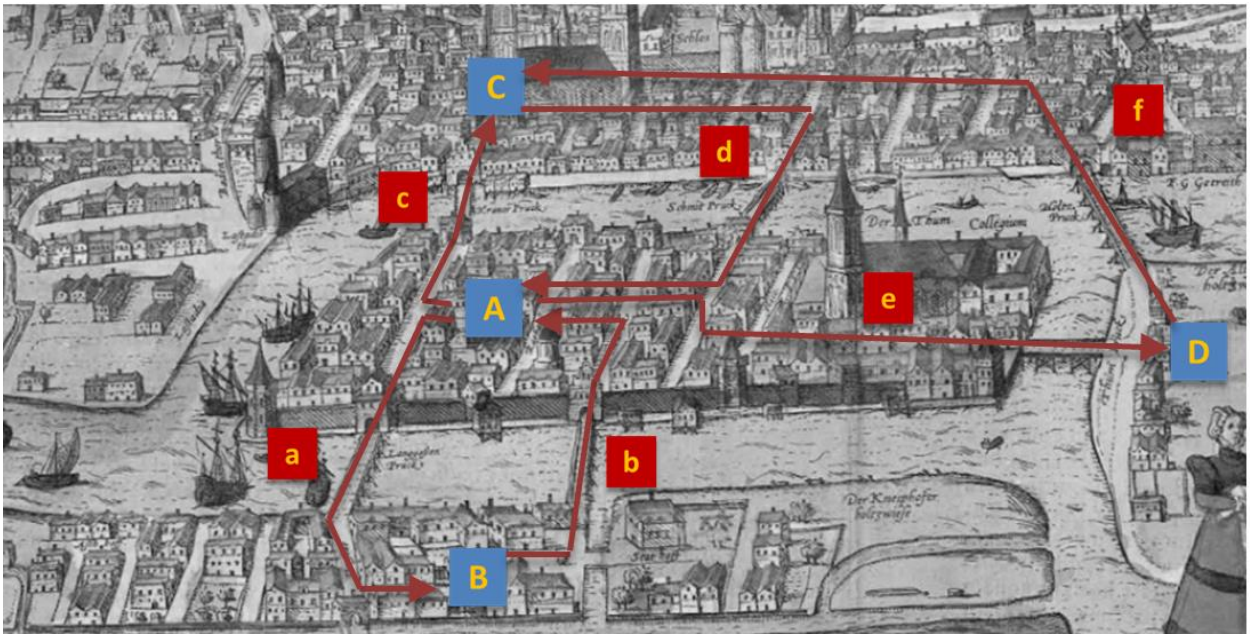


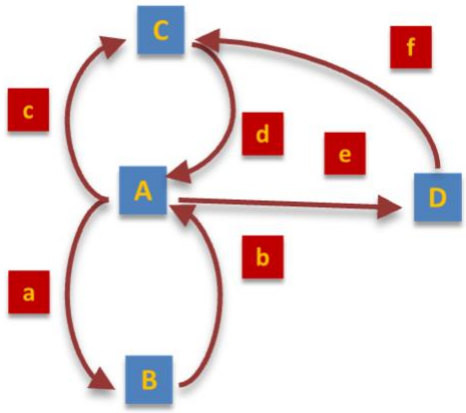
Königsberg, 1736

Leonard Euler, a Swiss mathematician made many contributions to the fields of mathematics, topology, mechanics, fluid dynamics, astronomy and even music theory. He introduced much of the mathematical terminology and notation we use today. If you use the notation f(x) you're using something Leonard Euler came up with to apply a function to an object.

One day while Euler was in the city of Konigsberg (Kaliningrad today) defined the following mathematical problem: is it possible to follow a path that crosses each bridge (seven of them) exactly once and return to the starting point?



Each land mass is a vertex (blue) and each bridge crossing is an edge (red). What he found is that there is no circuit without crossing a bridge twice (ex. C to A); the mathematical rigor and analysis he used to prove this conclusion formed the basis for graph theorem and topology. And we can represent this problem in tables of vertices and edges.

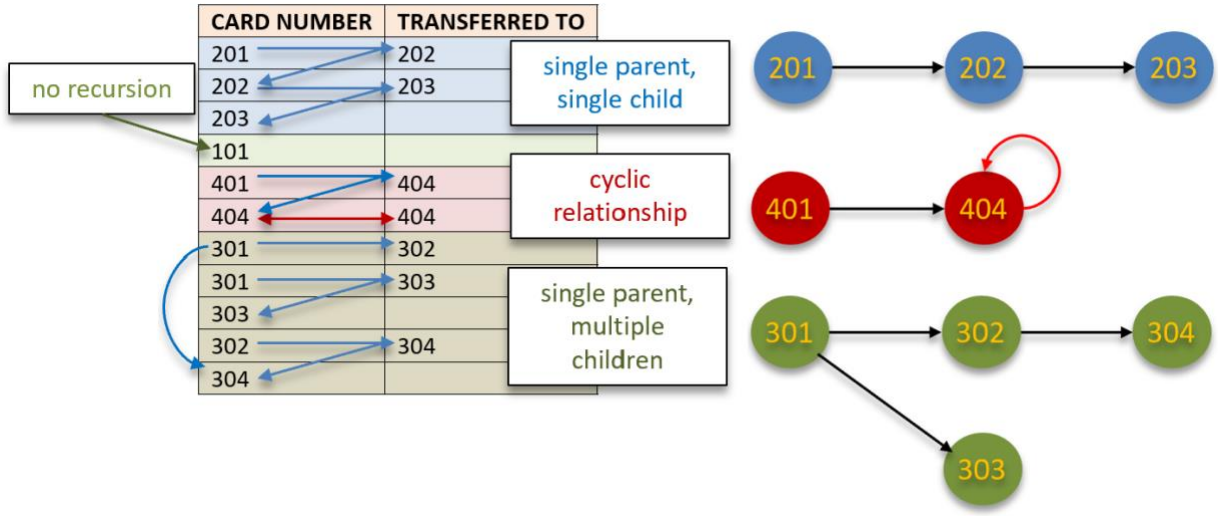


Directed Acyclic Graph (DAG)

Technical Debt

Graphs can be acyclic or cyclic, directed or non-directed and they form trees and hierarchies. Graphs can be used to visually represent relationships on social media platforms or for criminal investigations as we saw in the Panama papers. They make frequent appearances in distributed computing systems of nodes and clusters, and they can be used to traverse hierarchies and apply computations to each node like in PageRank.

Here we will use a graph to resolve technical debt; a cards database designed to track the issuing of new and replacement cards but contains no concept of an account. Replacement cards are issued when cards are reported as lost or stolen or if a customer changes product. Let's see what this looks like.



Card movement as a table and as a direct acyclic graph

For an existing card a replacement card is issued and the value for "TRANSFERRED TO" column is populated with the new card number the existing card has transferred to. The active card is identified by not have a transferred to value. In tabular format each record is independent of the next and certainly by looking at card '201' you are unaware that it is related to card '203'. The number of times a card can be transferred is theoretically limitless and the technical debt here is; how do I know card 203 is related to card 201 without having to traverse the hierarchy every time I want to know?
The cards database is missing an account number to essentially "hold" these cards together. The obvious candidate here is the first card that was issued. This card number will never change in this relationship, it is the hierarchy's anchor. The last card issued is the active card, and this will change whenever a transfer is initiated.

Solution on a Page (SQL)

In a tabular database with ANSI SQL 1999 compliance the use of a recursive common table (CTE) expression can be used to traverse each related card in the table. An anchor is set (active card has no transferred to value) and with the complete history of the card available in a single table the active card is then shared across related cards.

```
With CARD_RECURSION
as (Select CARD_NUMBER
         , TRANSFERRED_TO
         , 1 as CARD_STEP
         , CARD_NUMBER as ACTIVE_CARD
    From CARDS_TABLE
```

```
    Where TRANSFERRED_TO IS NULL
      Union All
    Select r.CARD_NUMBER
         , r.TRANSFERRED_TO
         , l.CARD_STEP + 1 as CARD_STEP
         , l.ACTIVE_CARD
    From CARDS_TABLE r
    Inner Join CARD_RECURSION l
    on r.TRANSFERRED_TO = l.CARD_NUMBER)
```

CARD NUMBER	TRANSFERRED TO	CARD STEP	ACTIVE CARD
201	202	3	203
202	203	2	203
203		1	203
101		1	101
301	302	3	304
301	303	2	303
303		1	303
302	304	2	304
304		1	304

After “sharing” the active card across cards an additional step is needed to traverse those cards and assign the grandparent of the hierarchy, the consistent card value that will never change because it was the first card assigned to a customer; this will be the account number.

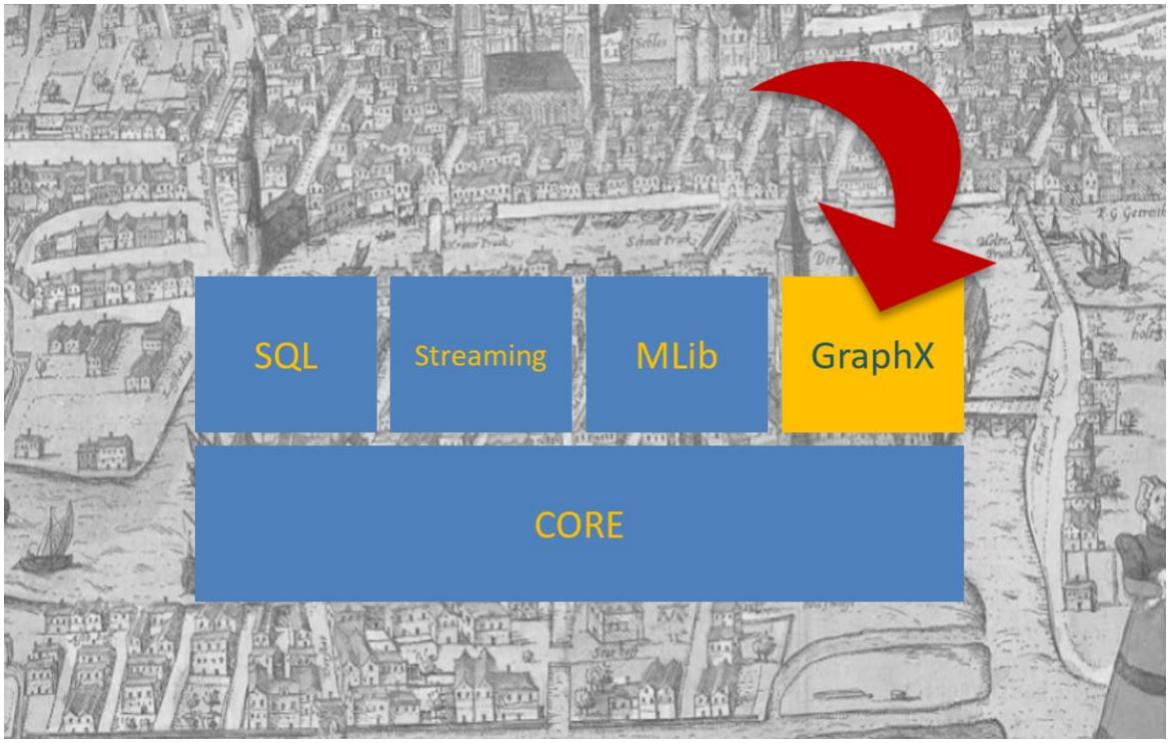
```
Select CARD_NUMBER
     , TRANSFERRED_TO
     , CARD_STEP
     , ACTIVE_CARD
     , First_Value(CARD_NUMBER) Over (Partition By ACTIVE_CARD Order By CARD_STEP
Desc) as ACCOUNT_NUMBER
From CARD_RECURSION
```

CARD NUMBER	TRANSFERRED TO	CARD STEP	ACTIVE CARD	ACCOUNT NUMBER
201	202	3	203	201
202	203	2	203	201
203		1	203	201
101		1	101	101
301	302	3	304	301
301	303	2	303	301
303		1	303	301
302	304	2	304	301
304		1	304	301

For distributed systems this is a little more tricky but solvable in Apache Spark GraphX.

Solution on a Page (Graph)

Apache Spark is a unified analytics platform with components for querying structured data through SQL, machine learning, streaming analytics and graph. The focus for this article is the use of the Pregel API under GraphX.



Apache Spark did not (at the time of writing) support recursive common table expressions in SQL. The challenge for the data engineers was to find another way of implementing this recursion using another means in Apache Spark. Hence the data engineers engaged the Pregel API and it executed the following.

Superstep 0,

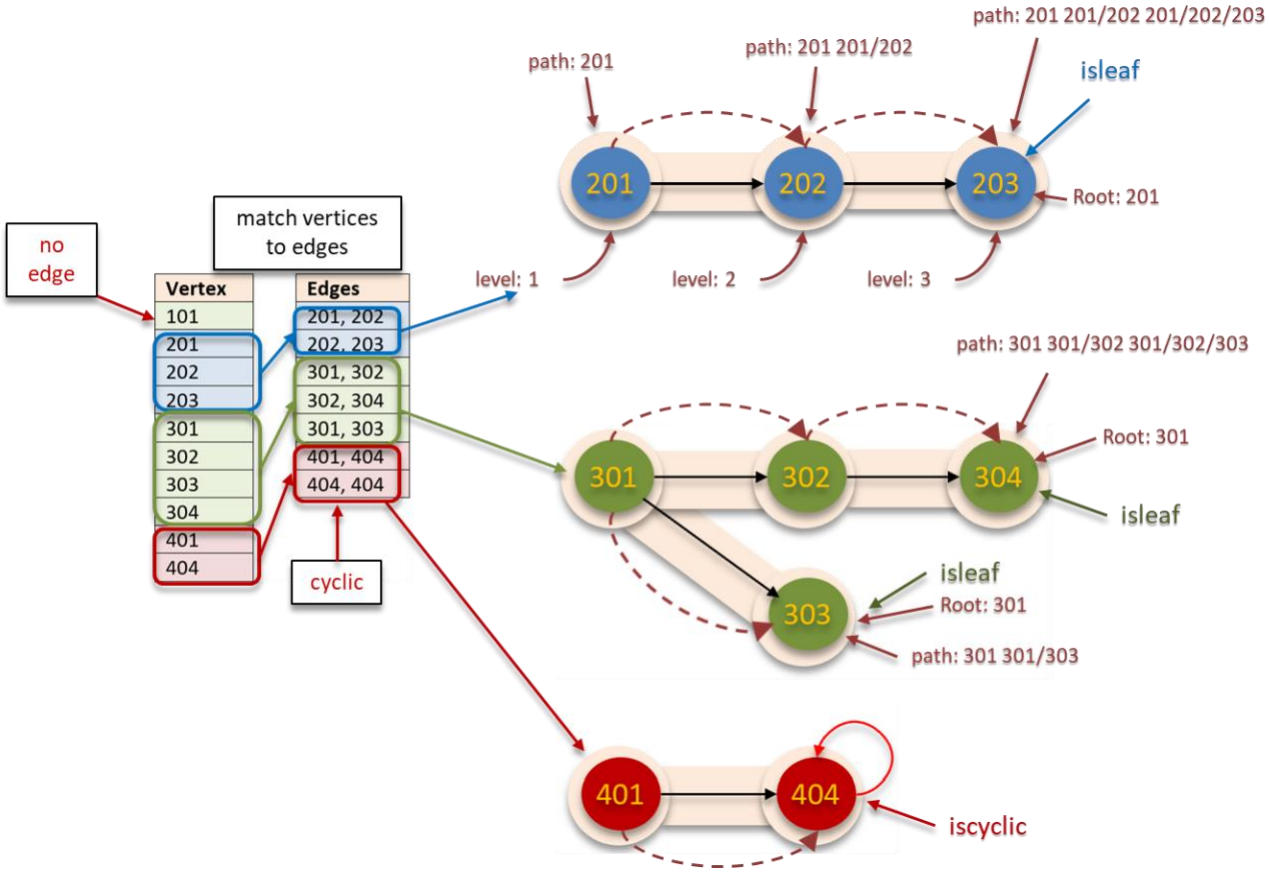
- list the vertices and edges…

Vertex
101
201
202
203
301
302
303
304
401
404

Edges
201, 202
202, 203
301, 302
302, 304
301, 303
401, 404
404, 404

A unique list of vertices is listed, and a list of edges are produced as an array; then…

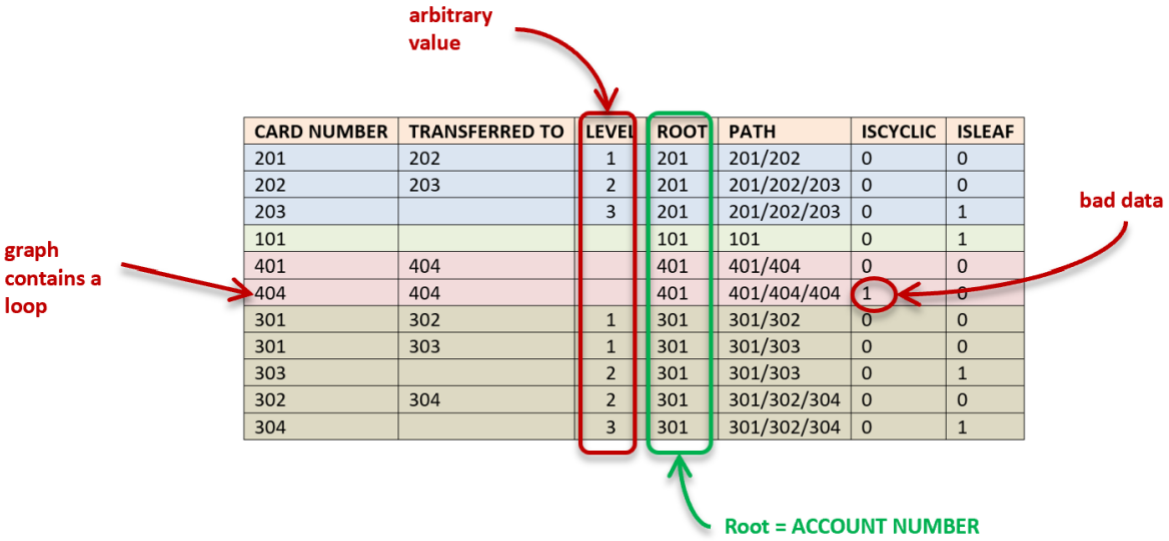
- …Pregel will scatter (load parallel graphs) …



Vertices are matched to the edges; vertices without edges are not graphed. As each vertex is stepped through in the superstep they are assigned values for:

- root (set as the first vertex in the hierarchy)
- isleaf (the lowest vertex in a tree)
- path (concatenation of vertices in the Eulerian path)
- iscyclic (if a loop exists in the hierarchy)

- …and persist to disk.



CARD NUMBER	TRANSFERRED TO	LEVEL	ROOT	PATH	ISCYCLIC	ISLEAF
201	202	1	201	201/202	0	0
202	203	2	201	201/202/203	0	0
203		3	201	201/202/203	0	1
101			101	101	0	1
401	404		401	401/404	0	0
404	404		401	401/404/404	1	0
301	302	1	301	301/302	0	0
301	303	1	301	301/303	0	0
303		2	301	301/303	0	1
302	304	2	301	301/302/304	0	0
304		3	301	301/302/304	0	1

Although the level differs from the card steps it does not matter to the result. We need to know what is the root node of the tree and that is the account number. It will never change; it will always be the root for the adjacent cards.
“101” had no adjacent vertex and thus is an account already.
“401” ended in a loop, this is an example of bad data from the source.

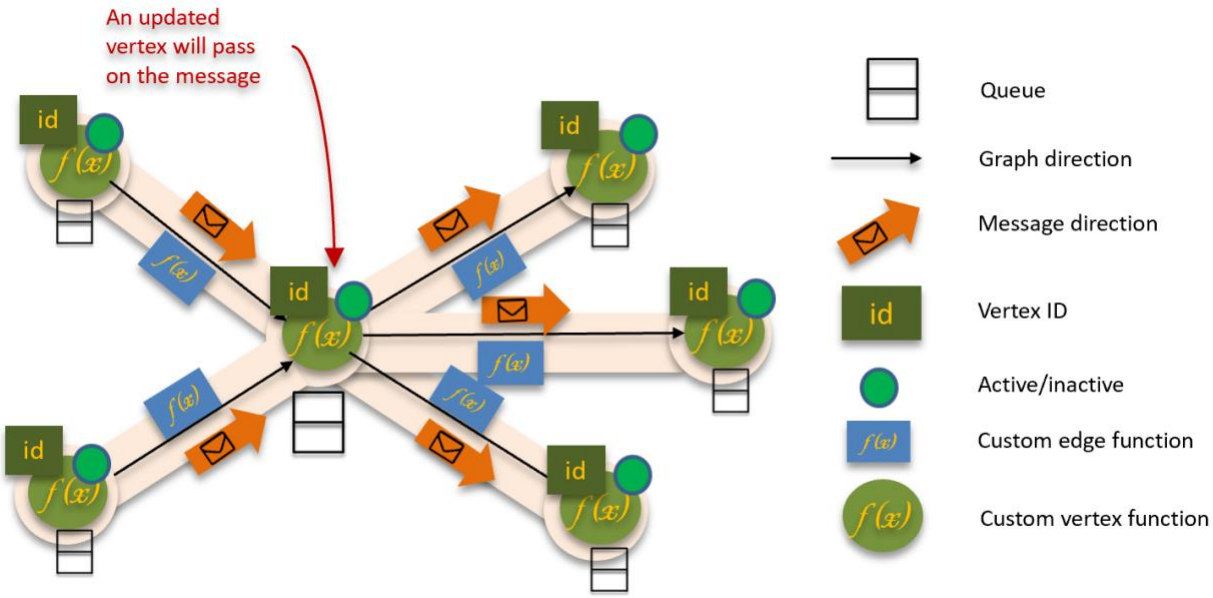
Code snippet:

```
?? insert code ??
```

In a single superstep with default values generated by using Pregel API we have already determined the account number for the cards! In the next section we’ll explain what happened in this graph and introduce an additional requirement for our use case and solve it with both a recursive CTE and see how the same is implemented in the Pregel API.

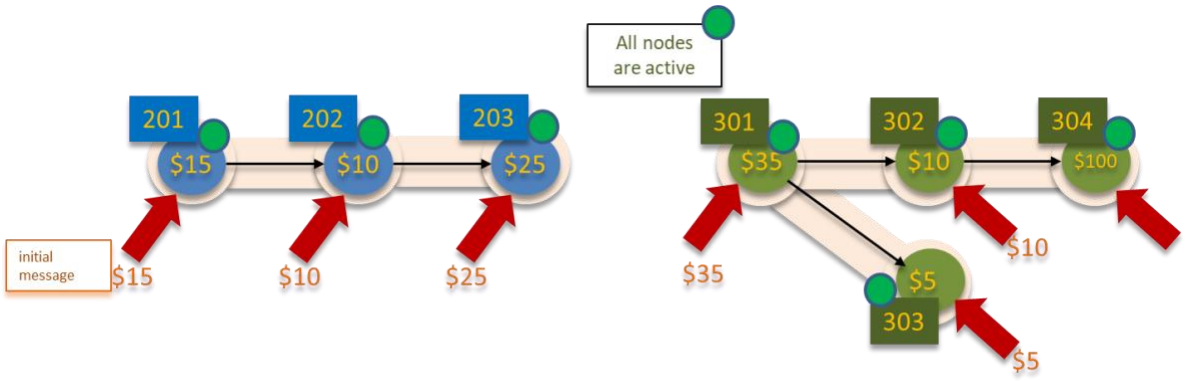
“Think like a Vertex”

Pregel is a message passing vertex centric API, a custom user defined function calculates a value and Pregel passes that result as a message to its nearest neighbouring vertex. Each vertex will have its own value and receive the neighbour’s value to process in its own copy of the same user defined function. The value held at a vertex is mutable and if the value does indeed change then in the next superstep that vertex will pass on the new value in a new message while it in turn could be receiving a message from another nearest neighbor.
In each superstep a vertex that receives a message remains active. If a vertex receives no message in a superstep then it will *vote* to halt and become inactive. If an inactive vertex receives a message that causes it to update its value then it will be active again. The supersteps are iterative and will continue until all vertices are inactive. Let’s take a look at what this message passing system looks like.

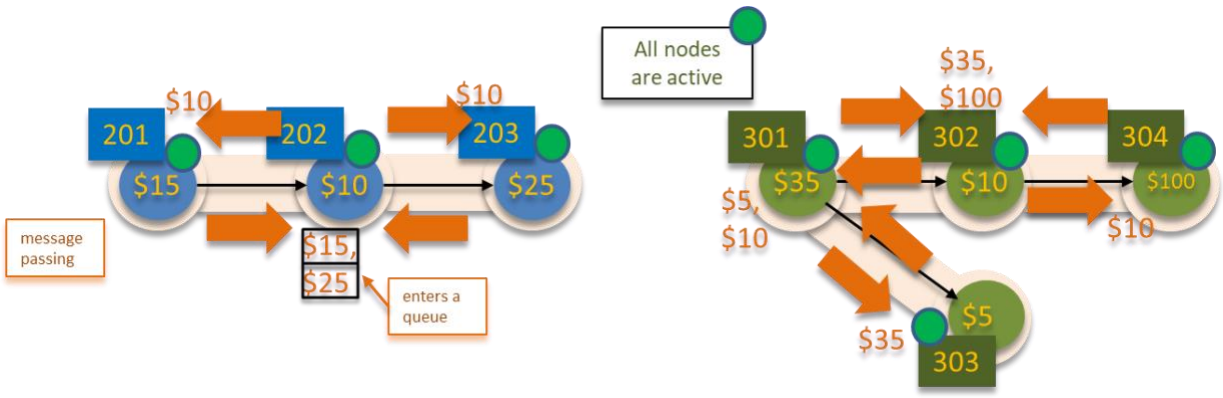


Each vertex and edge can contain a custom function. If a vertex receives multiple messages in a superstep then the messages will be queued to be processed by the vertex's user defined custom function. Let's change our use case a bit and use Pregel to assign the highest balance in each collection of cards, the highest balance reached for an account.

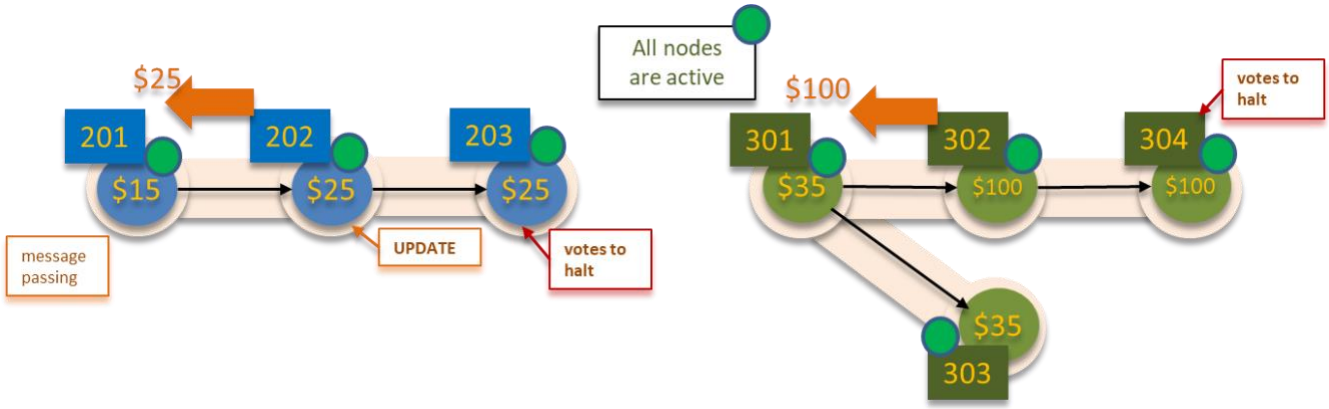
- **Superstep 0**; initial messages



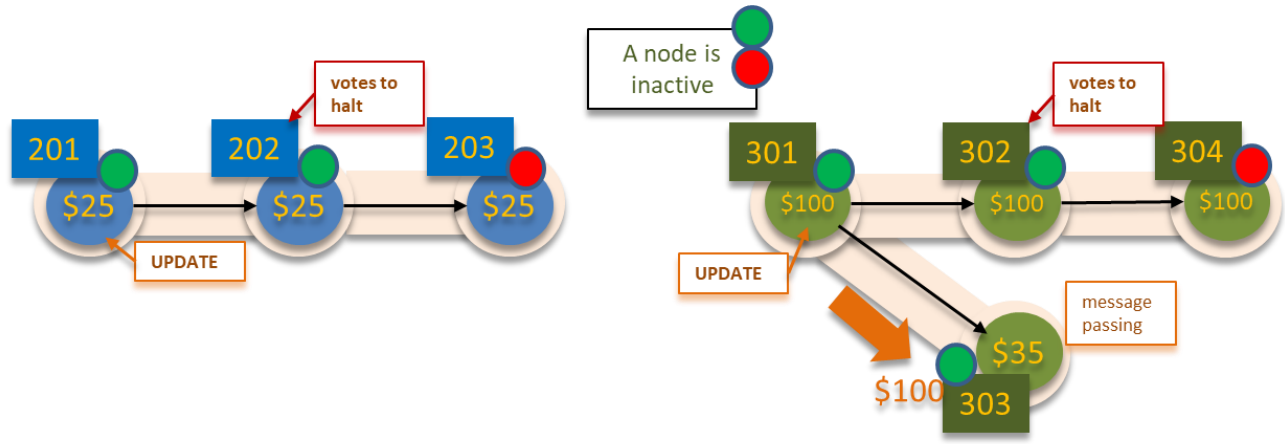
- **Superstep 1**; iterate, all nodes pass on their value to their nearest neighbours



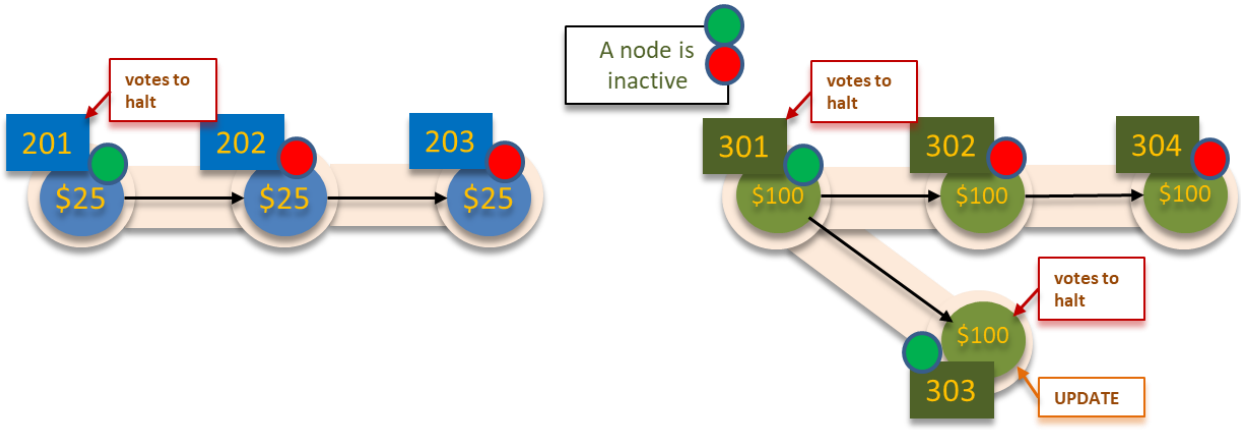
- **Superstep 2**; vertex value mutates if the custom function dictates so. A vertex that has not received a message in a superstep votes to halt (become inactive)



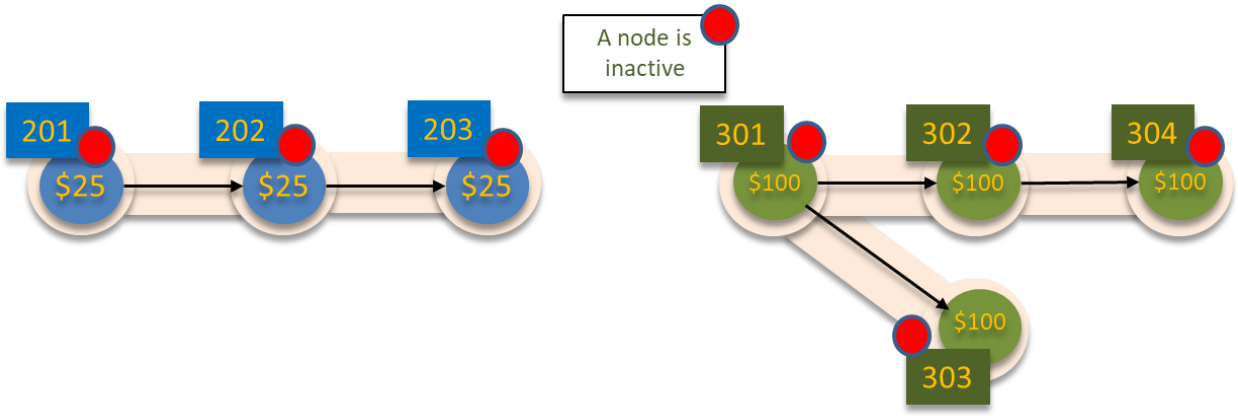
- **Superstep 3**; a vertex with its value changed will pass on the new value. An inactive vertex can become active again if it receives a new value.



- **Superstep 4**; the messages will travel until all values have been updated. As more vertices stop receiving messages the more vertices vote to halt.



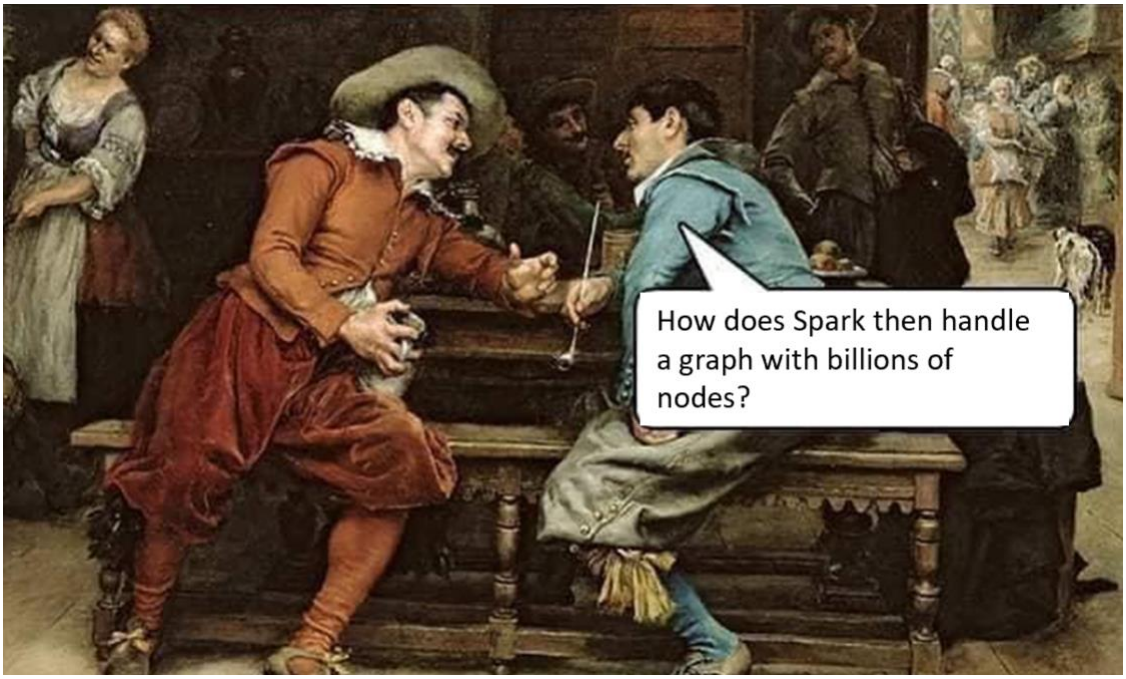
With no further messages being sent the entire algorithm has voted to halt and thus the algorithm has completed processing



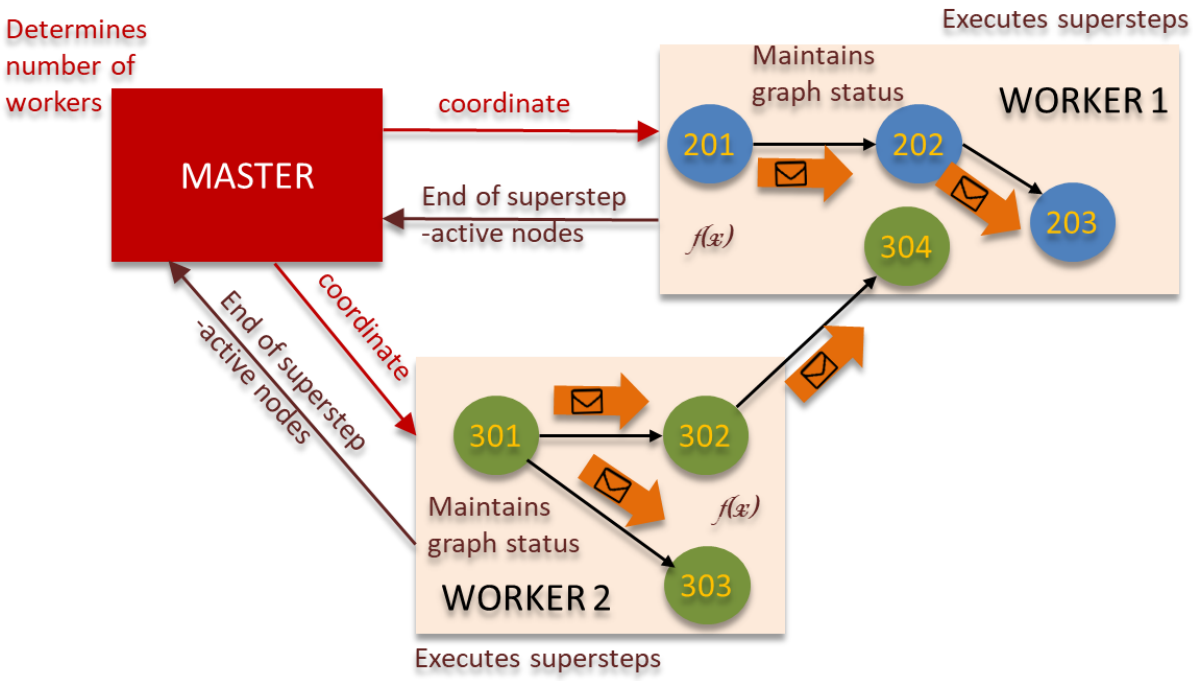
Code snippet:

?? insert code snippet ??

How Apache Spark divides the work



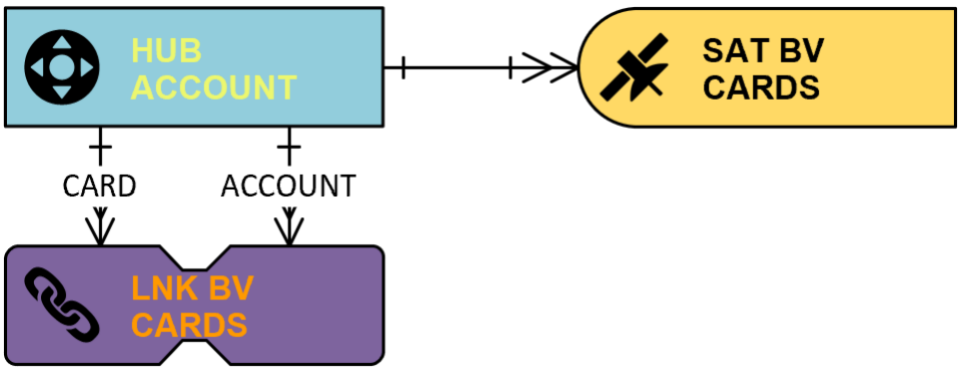
Spark's master will not execute any portion of the graph however it will coordinate the work on your behalf. A very large graph can span multiple worker nodes and the worker nodes feedback graph status updates to the master node. Fault tolerance is managed internally; the master pings the workers and failed nodes are replaced with the work reassigned to new works. A single graph can span multiple work nodes.



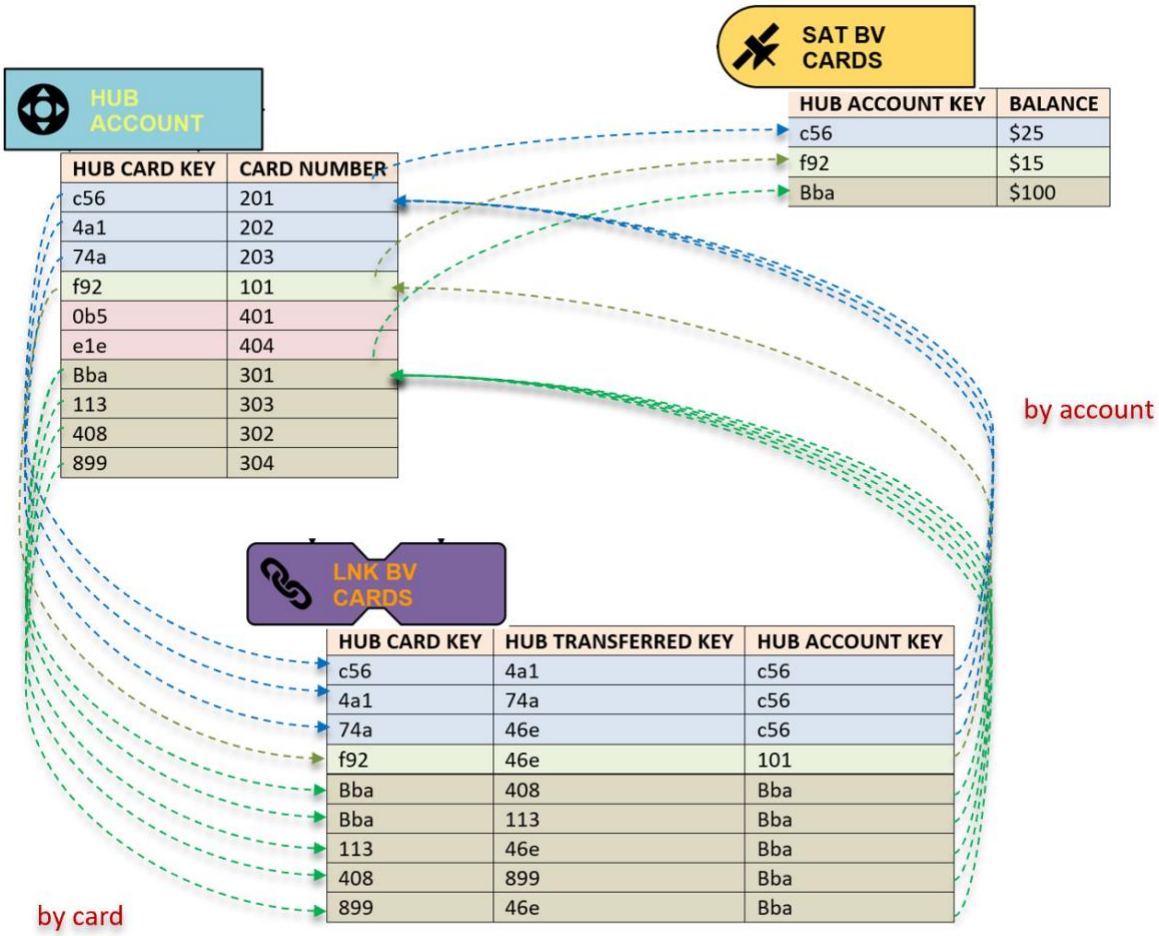
When there are no more active nodes the algorithm stops.

Load to Business Vault

The relationship is derived and not supplied from source; this business rule is then persisted in a business vault link table allowing querying for any card able to return the account number and all related card information throughout the account's history.



Any new transfer is added to the business vault link and any new card is added to the account hub. Each card will then have an entry in the business vault satellite. For any card you can then retrieve the max balance in the latest business vault satellite entry. Intricately the data loaded into this structure will look like this:



Notice that cards that did not transfer are assigned the zero key; a data vault 2.0 concept.

```
SELECT h_card.CARD_NUMBER
     , h_account.CARD_NUMBER AS account_number
     , sat.balance
FROM hub_account h_card
INNER JOIN lnk_bv_account lnk
   ON h_card.hub_card_key = lnk.hub_card_key
INNER JOIN hub_account h_account
   ON lnk.hub_account_key = h_account.hub_card_key
INNER JOIN sat_bv_account sat
   ON lnk.hub_account_key = sat.hub_account_key
```

Although the use case here is simple, Pregel has been used for far larger computational queries. The custom function at each vertex is a simple one, such as that used in PageRank. And from the closing image below, you can see where the name Pregel originated from!

City of Königsberg

Written and business problem by Patrick Cuba.
Designed and engineered by Nithin Kotha

Reference "Pregel: A System for Large-Scale Graph Processing" - Grzegorz Malewicz et al.
https://kowshik.github.io/JPregel/pregel_paper.pdf

END

Tables used for creating the diagrams:

CARD NUMBER	TRANSFERRED TO	CARD STEP	ACTIVE CARD	ACCOUNT NUMBER
201	202	3	203	201
202	203	2	203	201
203		1	203	201
101		1	101	101
401	404			
404	404			
301	302	3	304	301
301	303	2	303	301
303		1	303	301
302	304	2	304	301
304		1	304	301

CARD NUMBER	TRANSFERRED TO	CARD STEP	ACTIVE CARD	ACCOUNT NUMBER

CARD NUMBER	TRANSFERRED TO	CARD STEP	ACTIVE CARD	ACCOUNT NUMBER
201	202	3	203	201
202	203	2	203	201
203		1	203	201
101		1	101	101
301	302	3	304	301
301	303	2	303	301
303		1	303	301
302	304	2	304	301
304		1	304	301

CARD NUMBER	TRANSFERRED TO	LEVEL	ROOT	PATH	ISCYCLIC	ISLEAF
201	202	1	201	201/202	0	0
202	203	2	201	201/202/203	0	0
203		3	201	201/202/203	0	1
101			101	101	0	1
401	404		401	401/404	0	0
404	404		401	401/404/404	1	0
301	302	1	301	301/302	0	0
301	303	1	301	301/303	0	0
303		2	301	301/303	0	1
302	304	2	301	301/302/304	0	0
304		3	301	301/302/304	0	1

CARD NUMBER	TRANSFERRED TO	ACCOUNT NUMBER
201	202	201
202	203	201
203		201
101		101
301	302	301
301	303	301
303		301
302	304	301
304		301

CARD NUMBER
201
202
203
101
301
301

CARD NUMBER
303
302
304

HUB

HUB CARD KEY	CARD NUMBER
c56	201
4a1	202
74a	203
f92	101
0b5	401
e1e	404
Bba	301
113	303
408	302
899	304

LINK

HUB CARD KEY	HUB TRANSFERRED KEY	HUB ACCOUNT KEY
c56	4a1	c56
4a1	74a	c56
74a	46e	c56
f92	46e	101
Bba	408	Bba
Bba	113	Bba
113	46e	Bba
408	899	Bba
899	46e	Bba

SAT

HUB ACCOUNT KEY	BALANCE
c56	$25
f92	$15
Bba	$100