ITOUG
ITALIAN ORACLE USER GROUP

When Machine Learning meets Graph Databases

Gianni Ceresa

Managing Director of DATAlysis GmbH (Switzerland)

Working with Business Analytics, EPM tools and “data” for more than 10 years

Oracle ACE Director

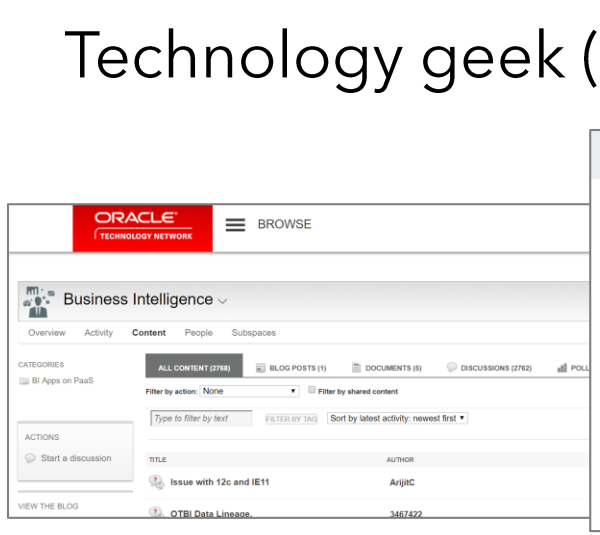
Part-time blogger on **gianniceresa.com**

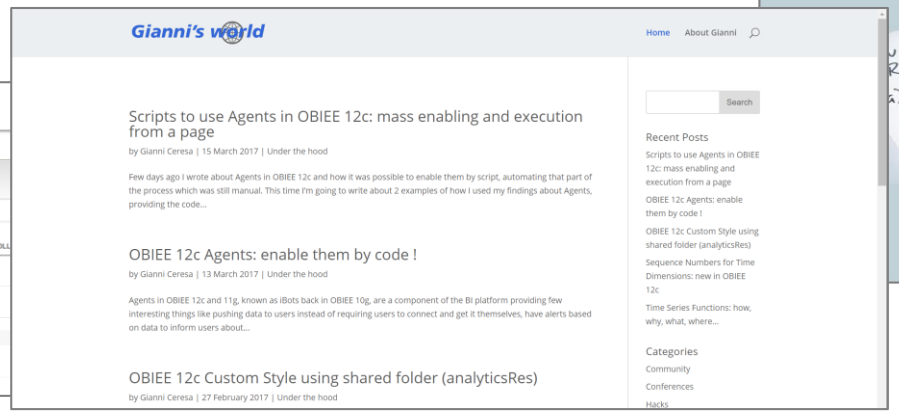
Full-time IRC (freenode | #obihackers) resident

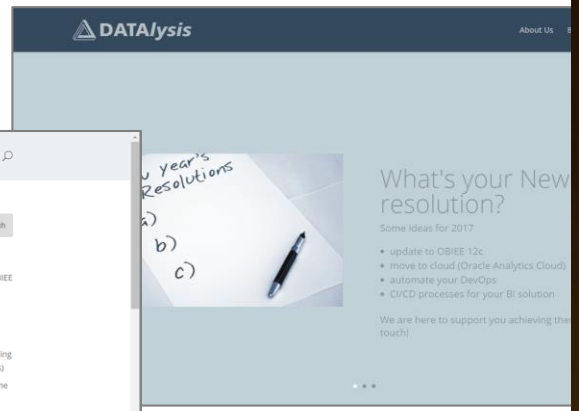
Same group on Telegram **http://telegram.me/obihackers**

ODC (ex OTN) forums addict

Technology geek (or just geek in general)







DATAlysis

450+ Technical Experts Helping Peers Globally

ORACLE ACE PROGRAM


ORACLE ACE Director


ORACLE ACE


ORACLE ACE Associate

---

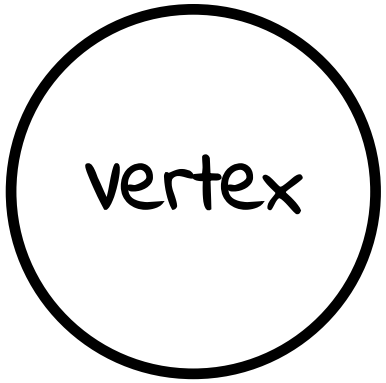
bit.ly/OracleACEProgram

Nominate yourself or someone you know: **acenomination.oracle.com**

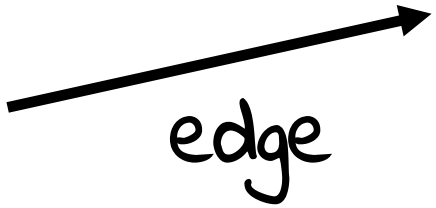
ORACLE

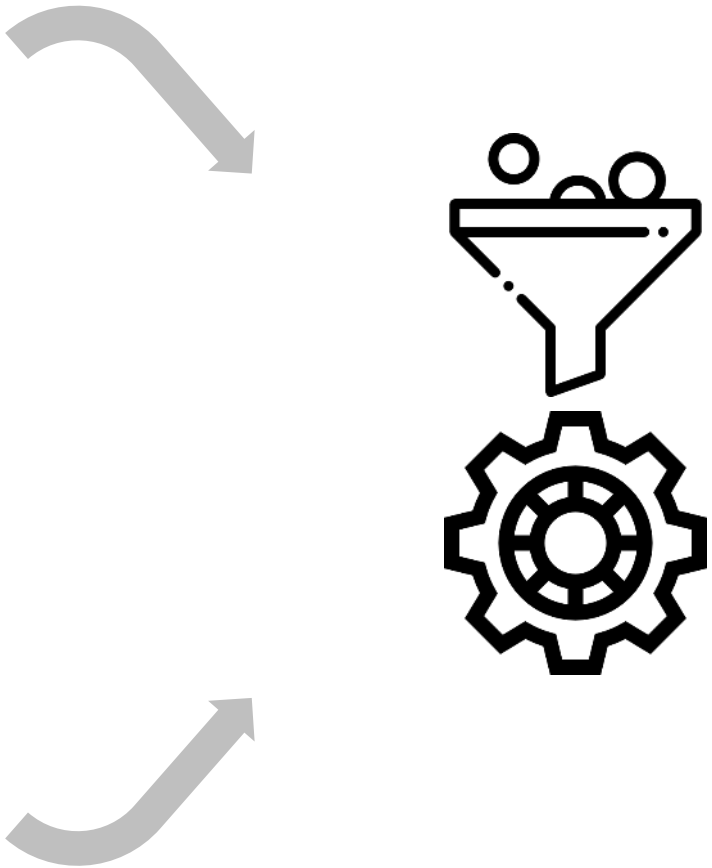
Graph Database: what's that?

Property Graph Database – What's that?



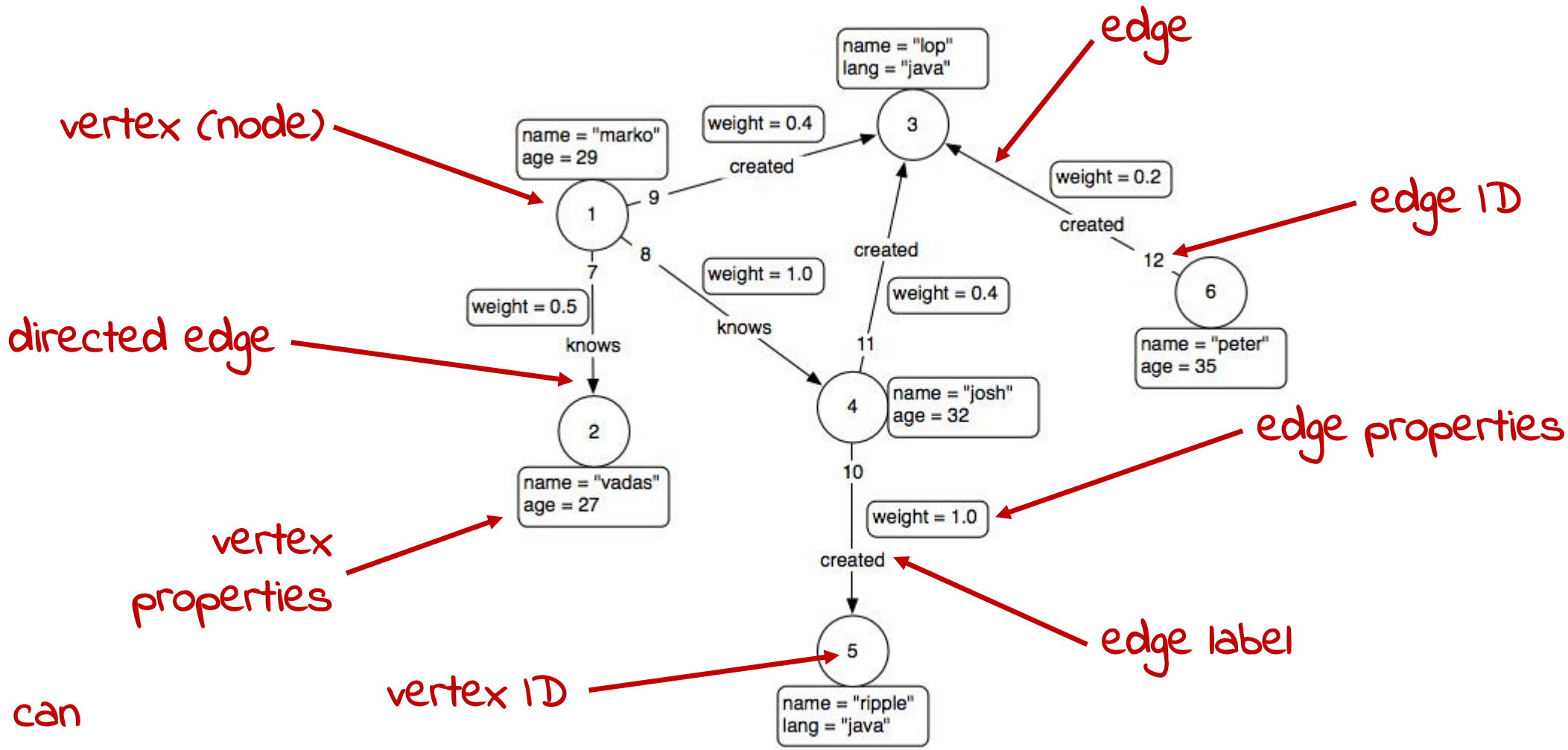
(also called node)





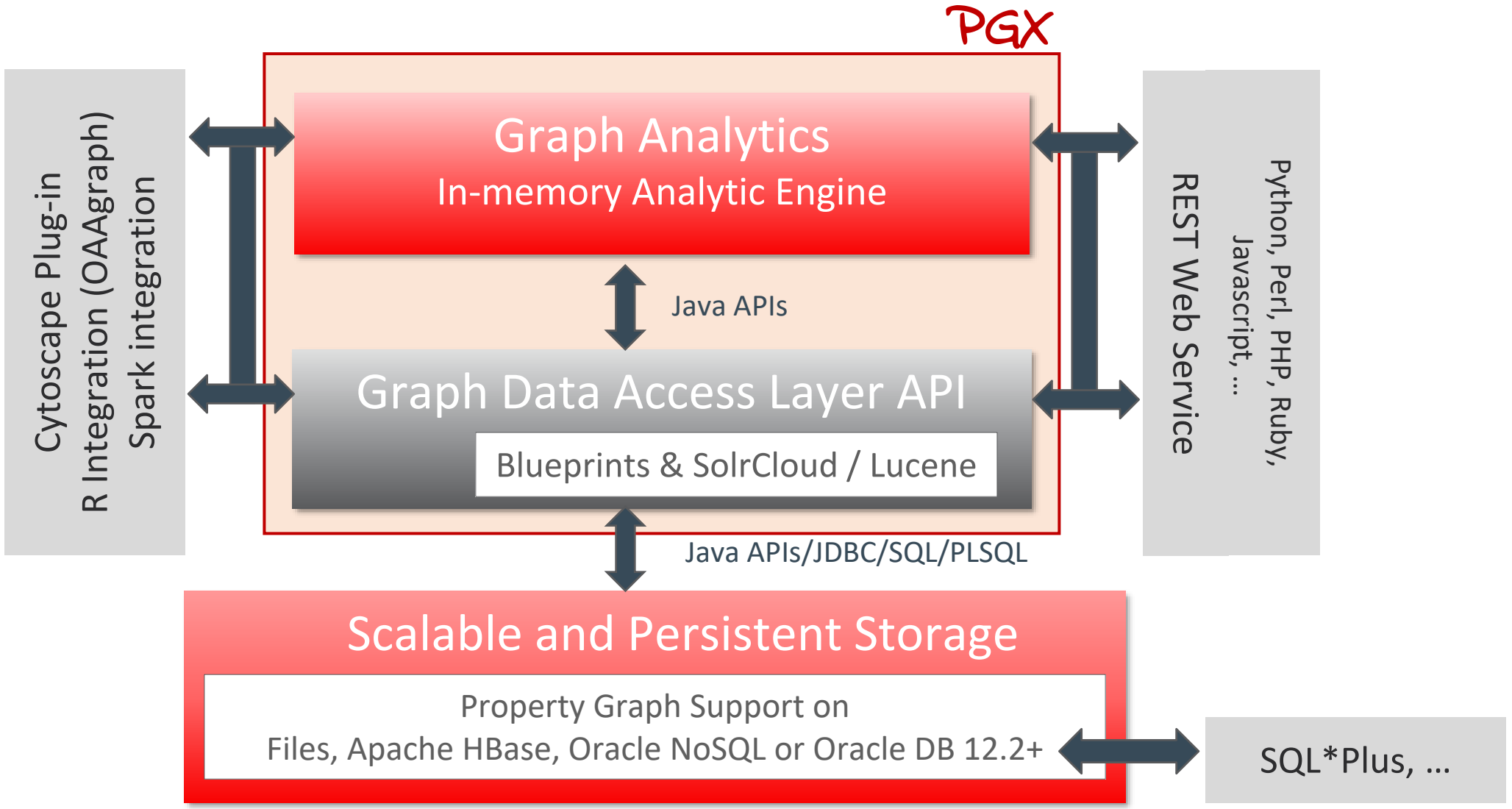
A Property Graph

Property Graph Database – What's that?

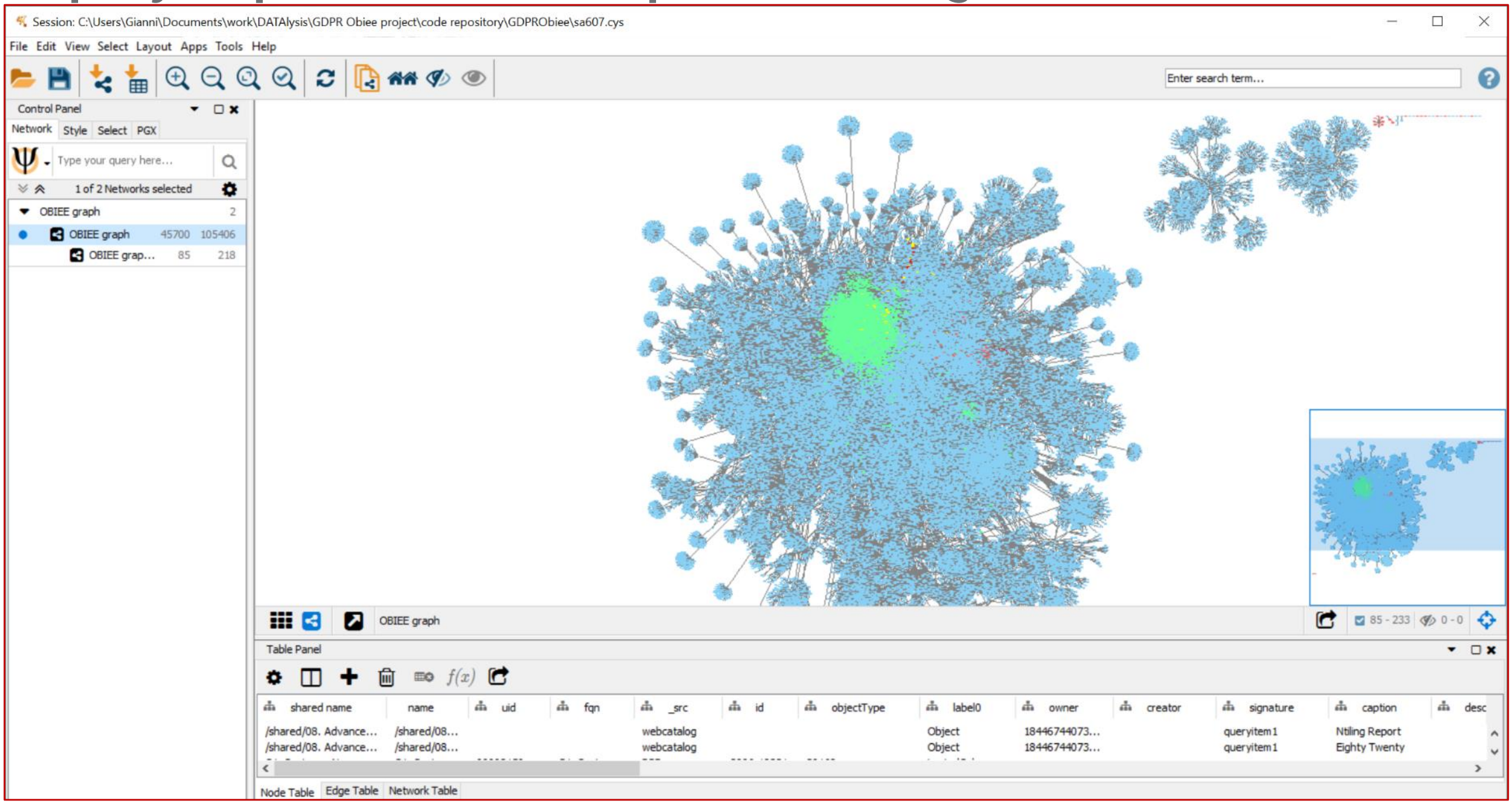


Oracle Property Graph

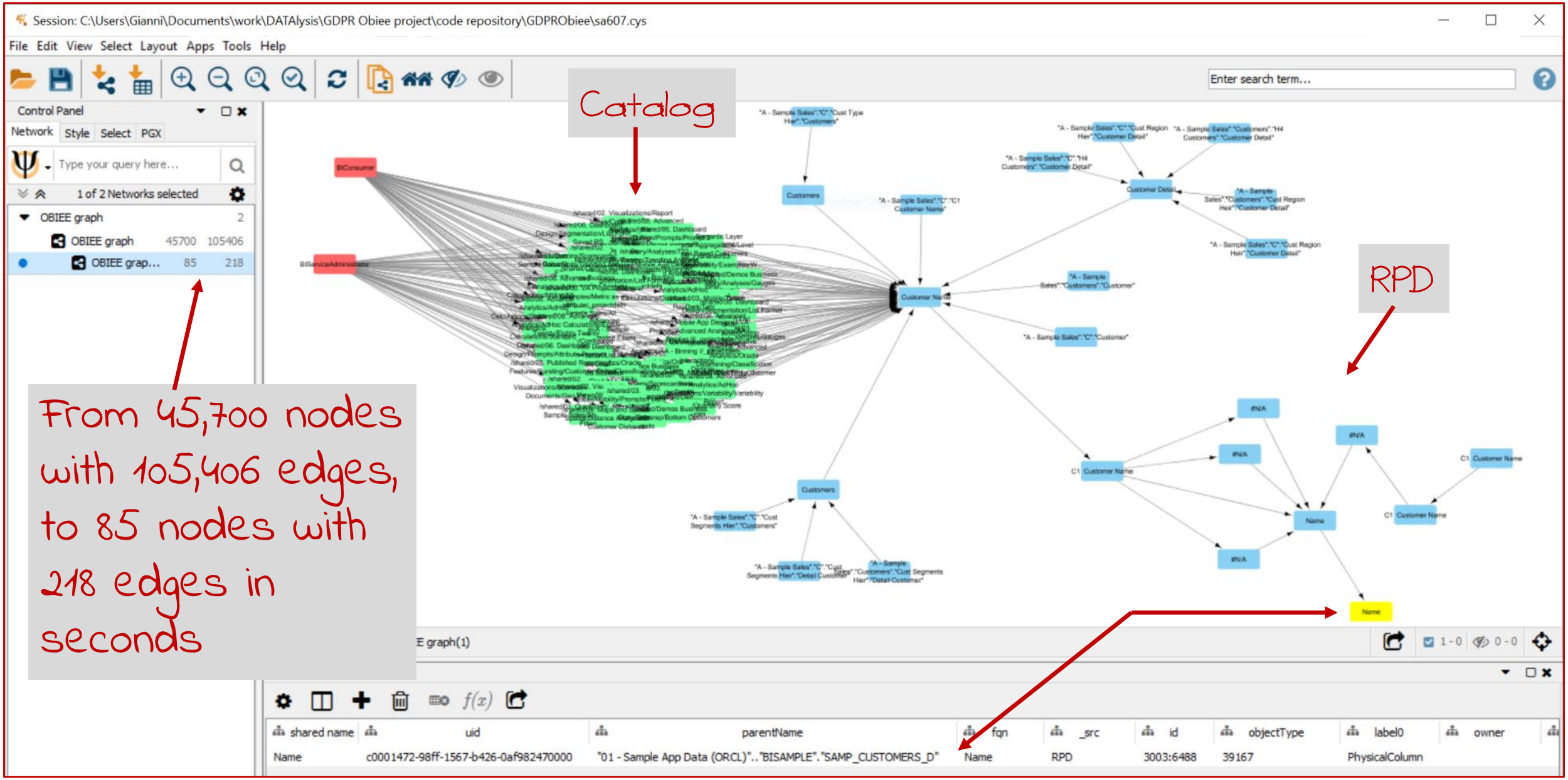
Oracle PGX (Parallel Graph AnalyticX) architecture



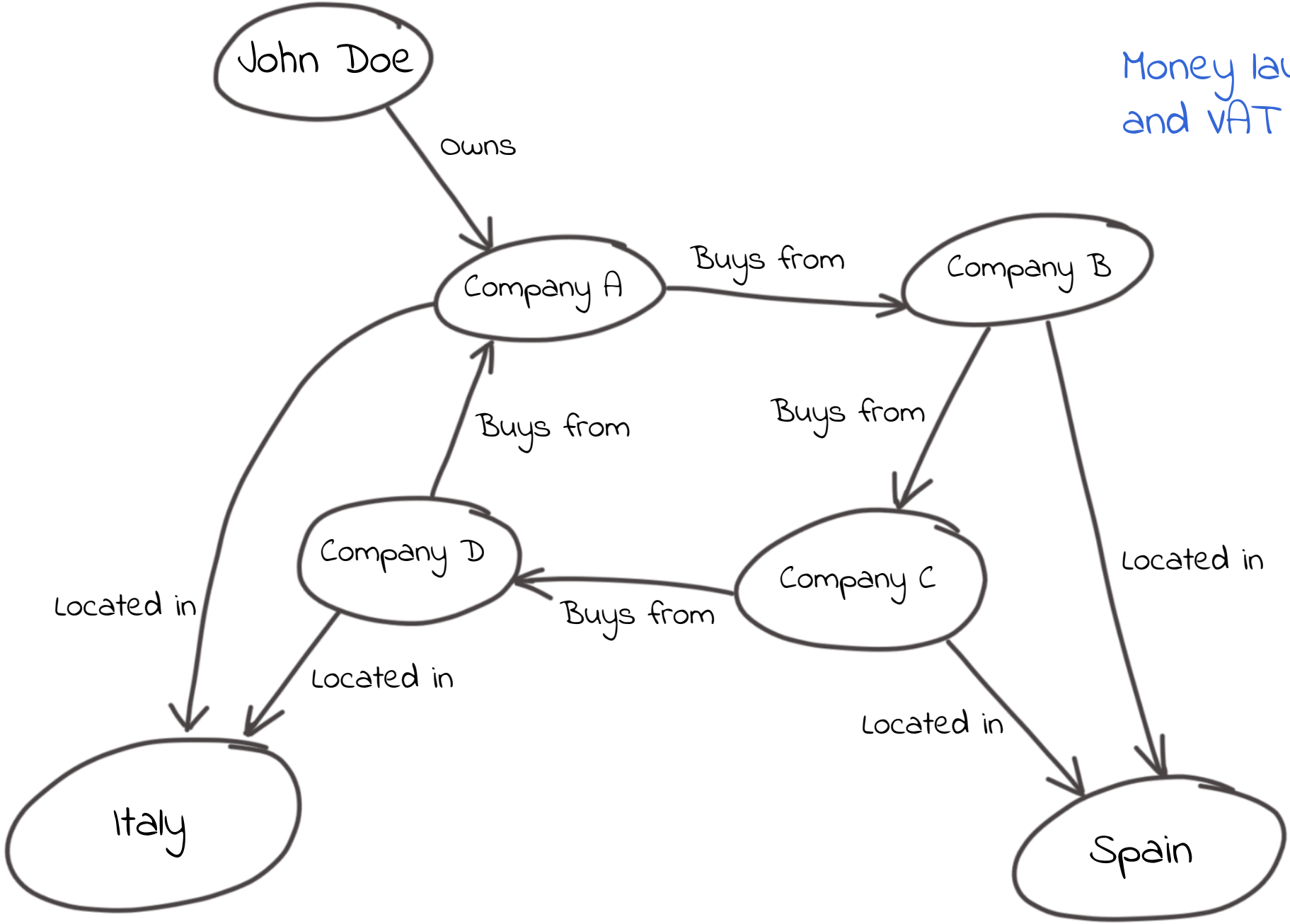
Property Graph Database – Examples: data lineage



Property Graph Database – Examples: data lineage



Property Graph Database – Examples: detecting frauds



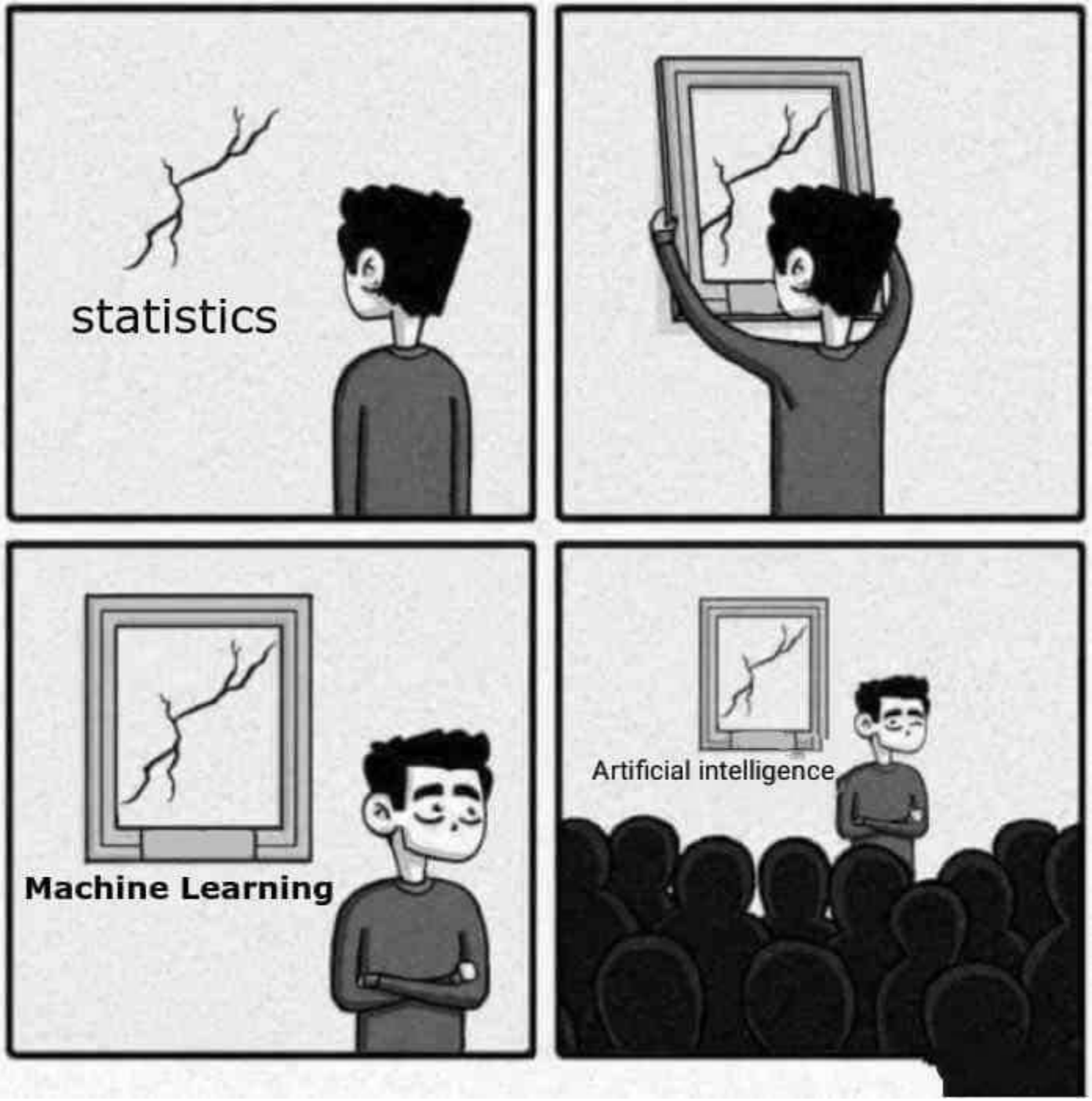
Money laundering and VAT frauds

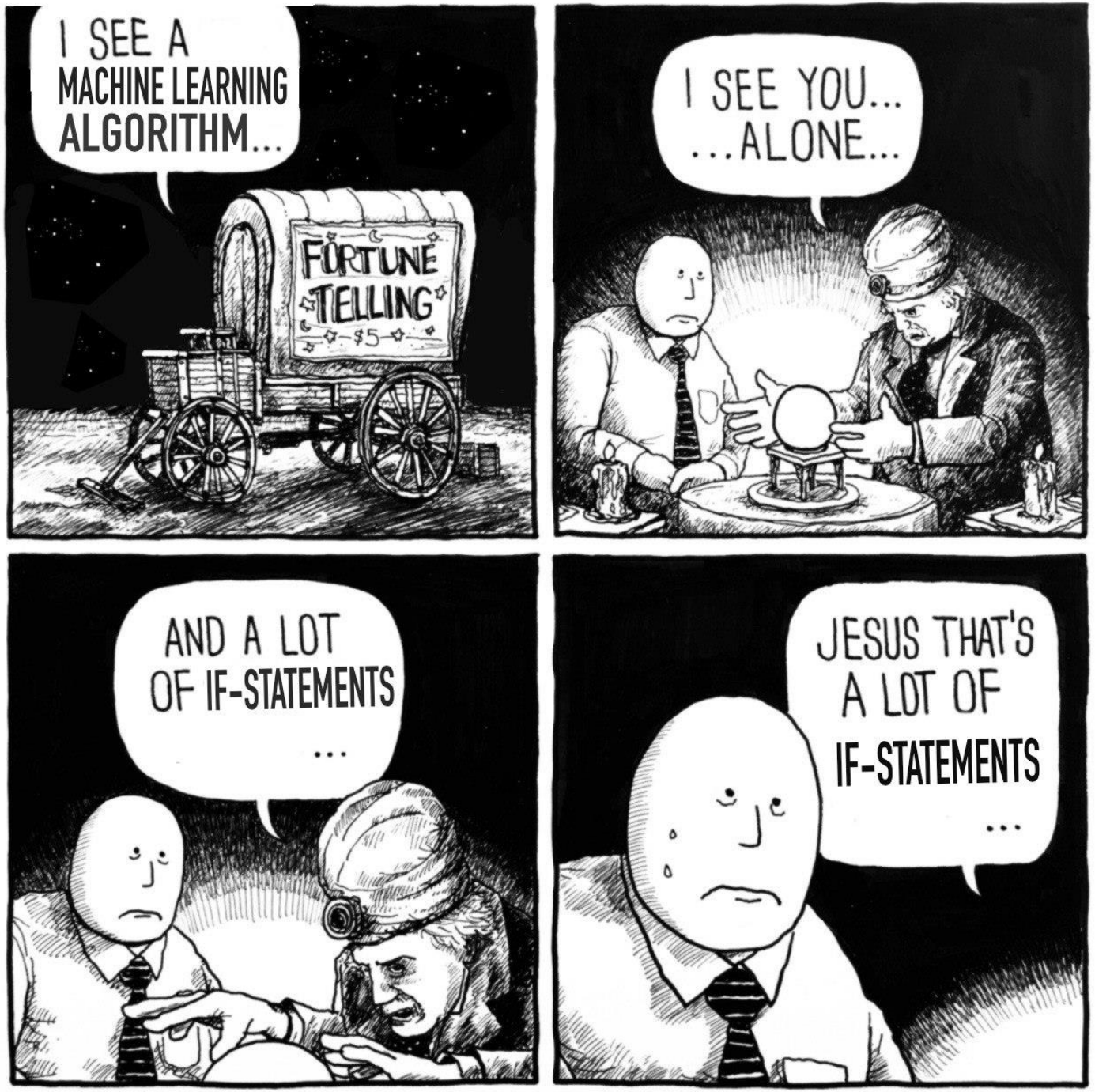
Graph Database – Examples: shortest path

Examples of graphs and graphs analytics can be seen when traveling from a location A to a location B :

Finding shortest path between 2 nodes of a graph

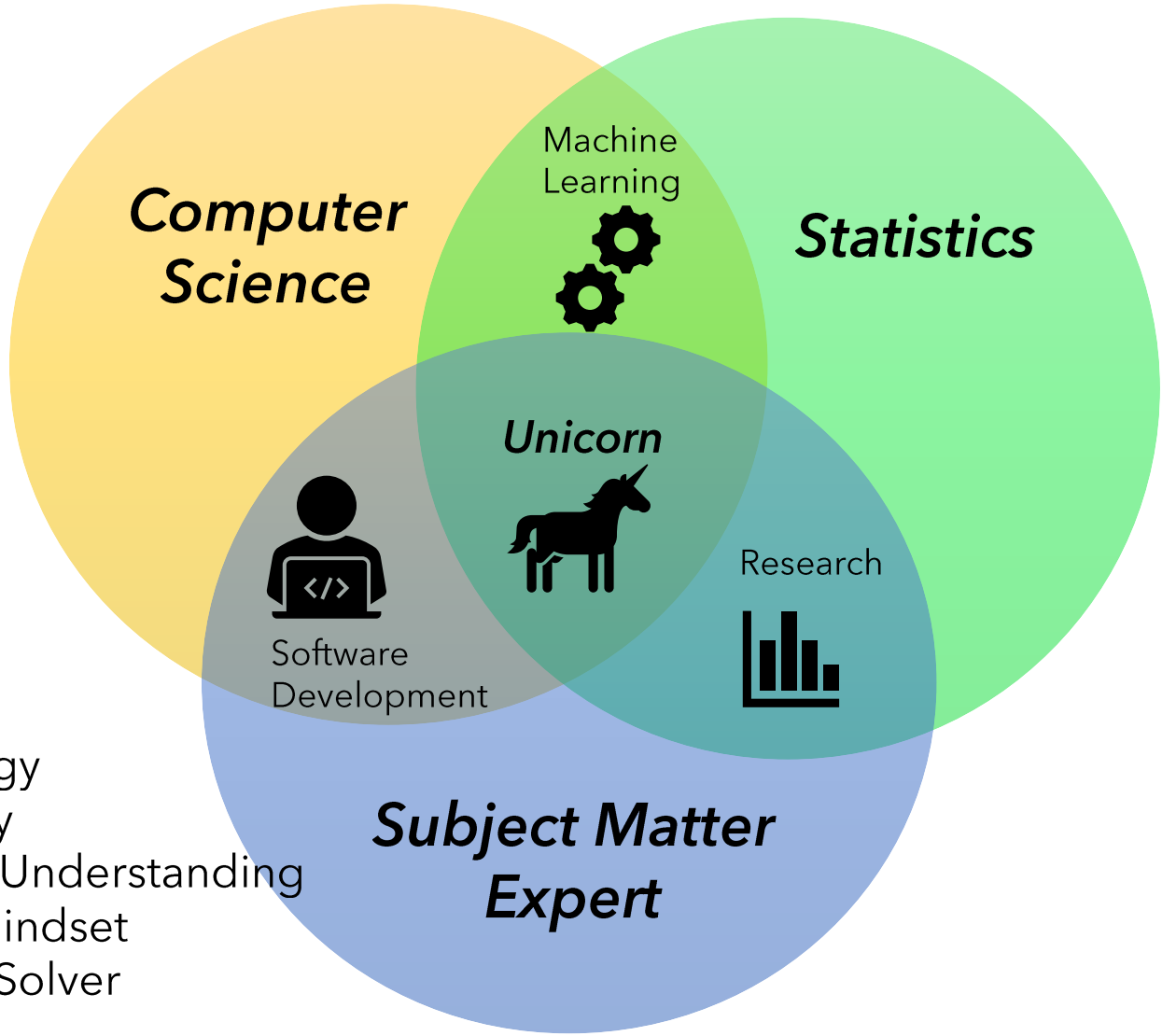
Back to the topic: Machine Learning




I SEE A MACHINE LEARNING ALGORITHM...
FORTUNE TELLING
$5
I SEE YOU...
...ALONE...
AND A LOT OF IF-STATEMENTS
...
JESUS THAT'S A LOT OF IF-STATEMENTS
...

Machine Learning

- R and/or Python
- SQL and NoSQL
- Hadoop / Spark
- Parallel Database
- Functional/ Object Orientated Programming



- Statistics
- Algebra
- Probabilities
- Bayesian

- Psychology
- Sociology
- Business Understanding
- Hacker Mindset
- Problem Solver
- Strategic

Simplifying ... a lot

Lot of linear algebra

- Matrices
- Vectors

Bunch of algorithms

Two definitions:

- Arthur Samuel (1959). Machine Learning: Field of study that gives computers the ability to learn without being explicitly programmed.

- Tom Mitchell (1998) Well-posed Learning Problem: A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.

It's all about the features

Machine Learning can only be as good as what you feed it with!

For example, estimating the price of a house:

How much?

by Francesco Tisiot (34)

It's all about the features

Machine Learning can only be as good as what you feed it with!

For example, estimating the price of a house:

How much?

Colour
yellow

Estimated price: N/A

It's all about the features

Machine Learning can only be as good as what you feed it with!

For example, estimating the price of a house:

How much?

Colour	*Size*	*Year*	*Location*
yellow	120 m2	1975	Bern

Estimated price: N/A

It's all about the features

Machine Learning can only be as good as what you feed it with!

For example, estimating the price of a house:

1'000 more columns of features

How much?

Colour	*Size*	*Year*	*Location*	*…*	*Public transport*	*Groceries*
yellow	120 m2	1975	Bern	…	100m	250m

Estimated price: N/A

Machine Learning isn't Machine Guessing

It's all about the features … and training

For example: estimating the price of a house:

Training

Colour	Size	Year	Location	…	Public transport	Groceries	Price
green	100 m2	1970	Bern	…	150m	200m	800K CHF
blue	180 m2	2010	Zürich	…	500m	1000m	2.4M CHF
red	75 m2	1950	Basel	…	50m	50m	500K CHF
…	(100'000 rows of houses with a price)						

How much?

Colour	Size	Year	Location	…	Public transport	Groceries
yellow	120 m2	1975	Bern	…	100m	250m

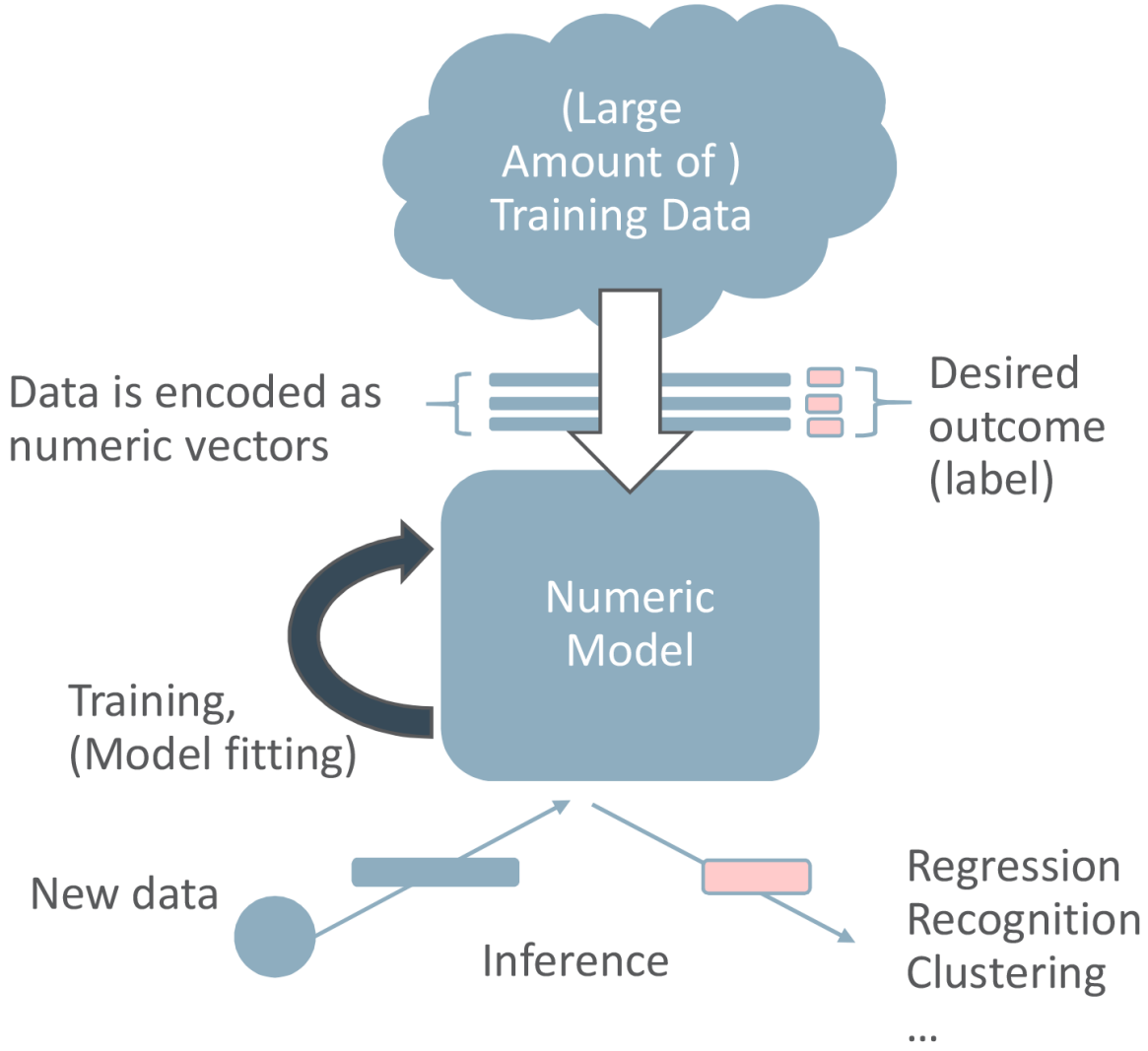
Estimated price: 1'217'542 CHF

It's all about the features … and training

Training? Is training really mandatory in Machine Learning?

A common global understanding of ML is that a model needs to be trained on some data before to be applied to some new data and get the extra insight we want the "machine" to find.

This isn't always the case …

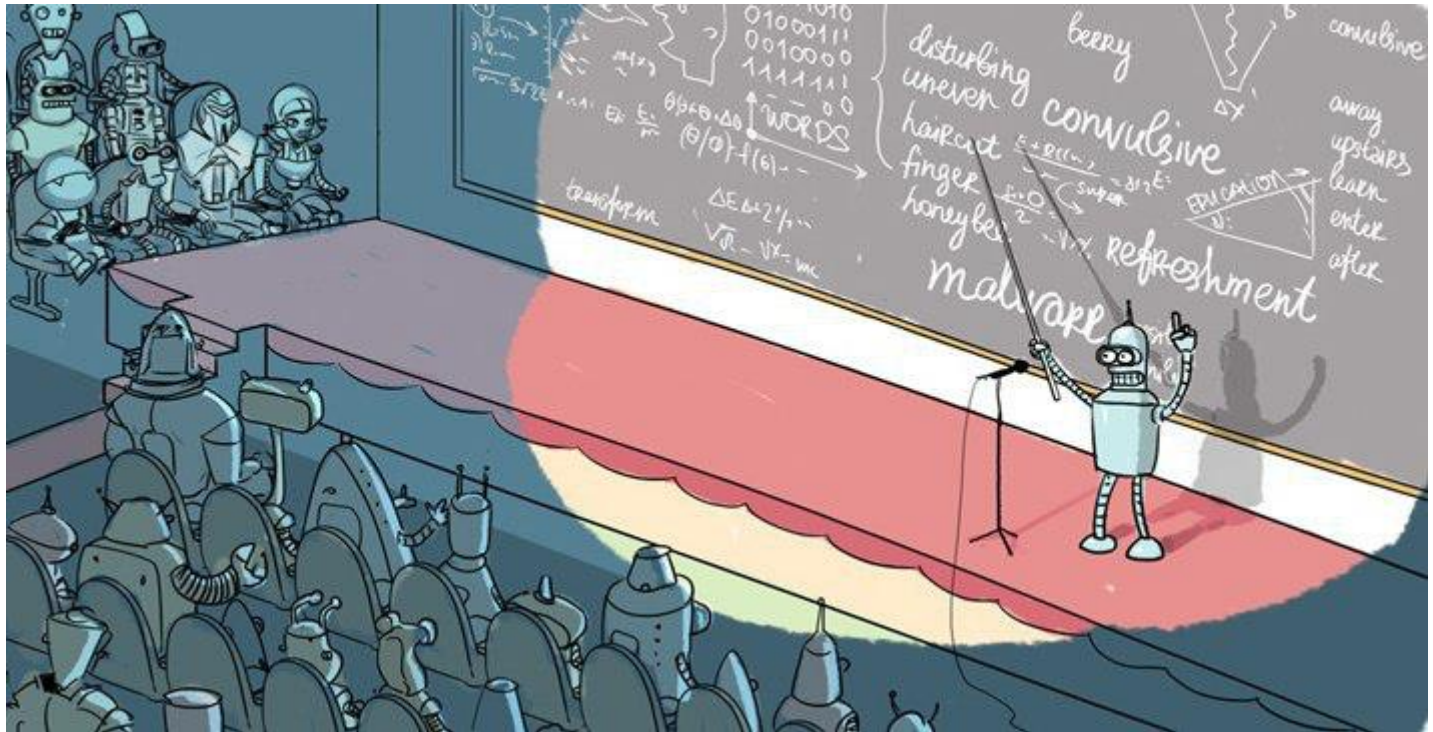


Finally back, for real, to the topic: Machine Learning & Graph Databases

Machine Learning with Graph databases

ML can be split into 3 main buckets when related to Graphs

- Graphs algorithms generating new insights
- Export features from a graph into a “classical” ML pipeline
- Bring ML into the graph

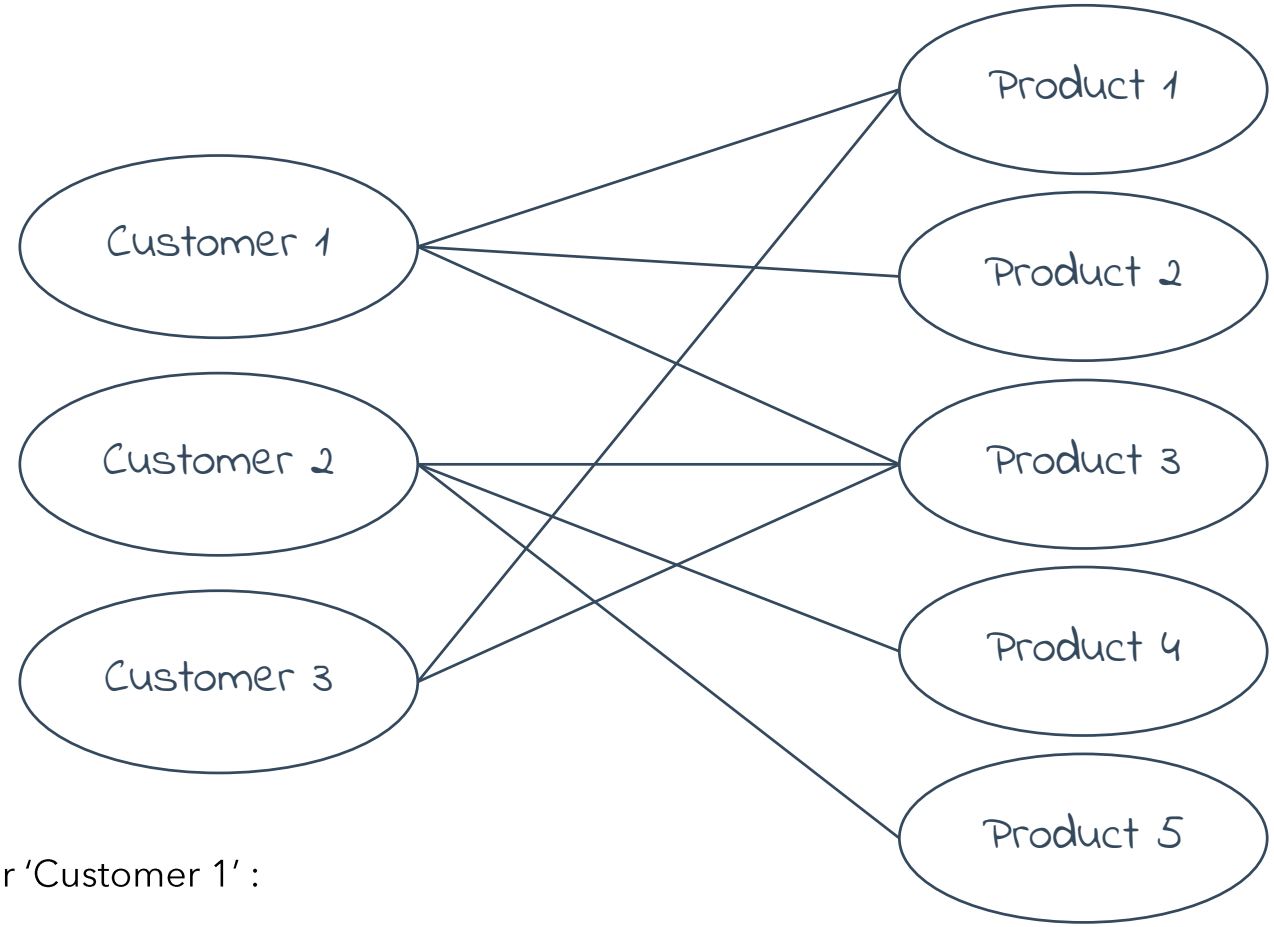


Graphs algorithms generating new insights

Example: finding similar customers based on what products they bought (clustering)

Personalized PageRank:

The Personalized PageRank allows to select a particular vertex or a set of vertices from the given graph in order to give them a greater importance when computing the ranking score, which will have as result a personalized PageRank score and reveal relevant (or similar) vertices to the ones chosen at the beginning.



Personalized PageRank for 'Customer 1' :
Customer 1 : 0.3401
Customer 3 : 0.1082
Customer 2 : 0.0920

Customer 1 is more similar to Customer 3 than Customer 2

Graphs algorithms generating new insights

Oracle PGX comes with a "set" of algorithms available out of the box

- 59 (in PGX 19.1.0 counting all the sub-versions)

It's possible to write new algorithm

- Using Green-Marl and "compiling" them in PGX
- Using Java

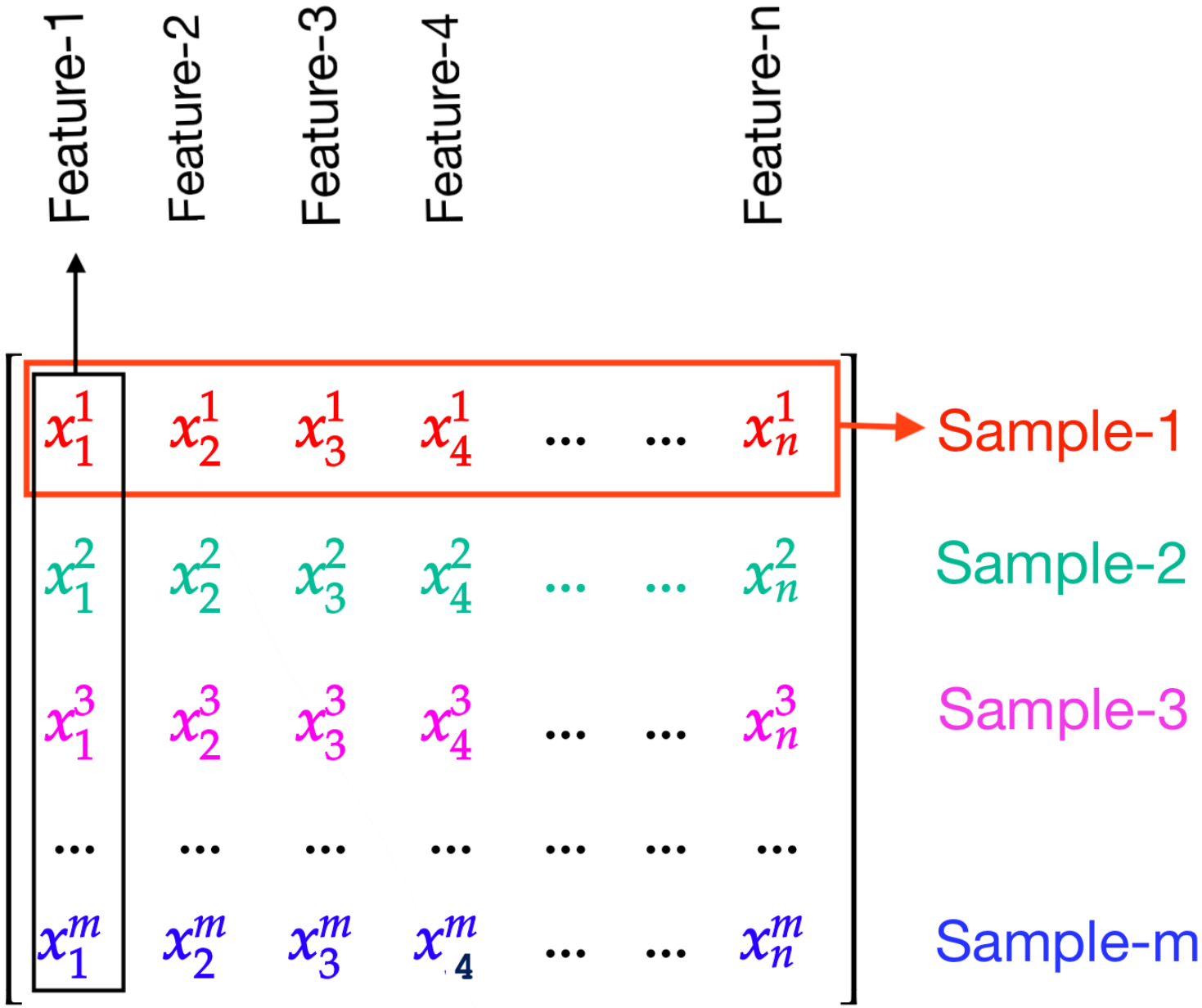
Built-In Algorithms

PGX includes a wide selection of optimized graph algorithms that can be invoked through the Analyst. The following table provides an overview of the available algorithms, grouped by category.

Category	Algorithms
Classic graph algorithms	Prim's Algorithm
Community detection	Conductance Minimization (Soman and Narang Algorithm), Infomap, Label Propagation
Connected components	Strongly Connected Components, Weakly Connected Components (WCC)
Link predition	WTF (Whom To Follow) Algorithm
Matrix factorization	Matrix Factorization
Other	Graph Traversal Algorithms
Path finding	Bellman-Ford Algorithms, Bidirectional Dijkstra Algorithms, Dijkstra Algorithms, Fattest Path, Hop Distance Algorithms
Ranking and walking	Closeness Centrality Algorithms, Degree Centrality Algorithms, Eigenvector Centrality, Hyperlink-Induced Topic Search (HITS), PageRank Algorithms, Random Walk with Restart, Stochastic Approach for Link-Structure Analysis (SALSA) Algorithms, Vertex Betweenness Centrality Algorithms
Structure evaluation	Adamic-Adar index, Conductance, Cycle Detection Algorithms, Degree Distribution Algorithms, Eccentricity Algorithms, K-Core, Local Clustering Coefficient (LCC), Modularity, Partition Conductance, Reachability Algorithms, Topological Ordering Algorithms, Triangle Counting

https://docs.oracle.com/cd/E56133_01/latest/reference/analytics/builtins.html

Export features from a graph



Export features from a graph

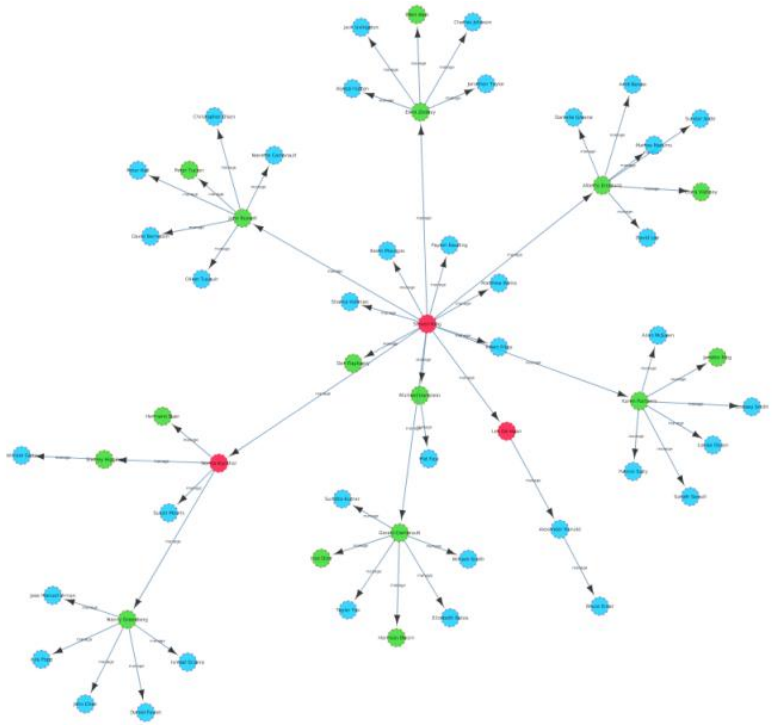
Graphs can be queried, results exported as “datasets” (rows and columns)

- Run graphs algorithms to generate “features”
  - PageRank
  - Number of in-out edges
  - etc.
- Query the graph to select all the features you need, including the newly generated properties of the algorithms
- Export (as CSV for example) and feed it into your standard ML pipeline

The power of classical ML increased by extra features representing the graph
(some components of the structure around data, enriching the basic features)

Bring ML into the graph

To bring ML into the graph, the graph needs to be made compatible with ML techniques



?

$$A = [a_{ij}] = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \square & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix}_{m \times n}$$

Bring ML into the graph

Oracle PGX has, at the moment (PGX 19.3.1), 2 built-in algorithms for ML in the graph:

- DeepWalk (Vertex embeddings)
- Pg2vec (Graph embeddings)

https://docs.oracle.com/cd/E56133_01/latest/prog-guides/mllib/index.html

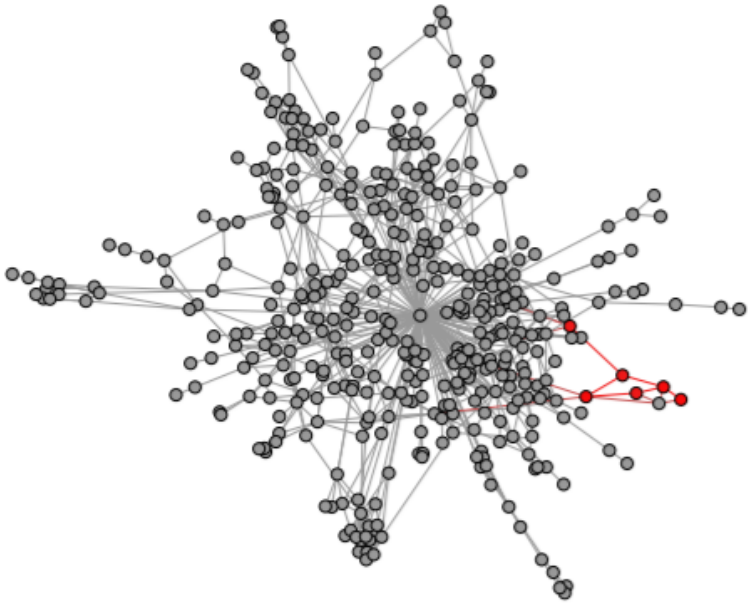
DeepWalk

Algorithm 1 DeepWalk(G, w, d, γ, t)

Input: graph $G(V, E)$
window size w
embedding size d
walks per vertex γ
walk length t

Algorithm 2 SkipGram(Φ, $\mathcal{W}_{v_i}$, w)

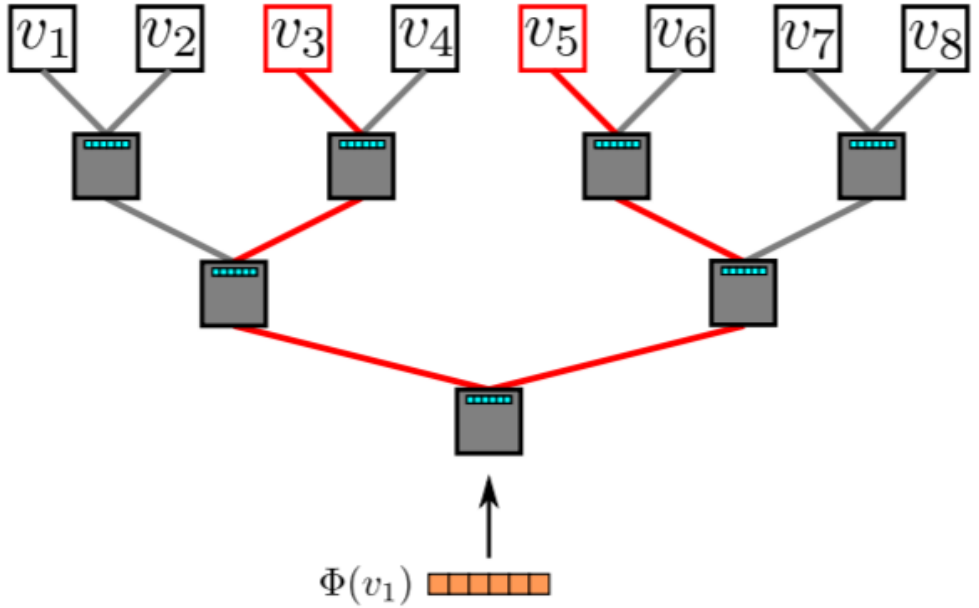
1: **for each** $v_j \in \mathcal{W}_{v_i}$ **do**
2: **for each** $u_k \in \mathcal{W}_{v_i}[j - w : j + w]$ **do**
3: $J(\Phi) = -\log \Pr(u_k \mid \Phi(v_i))$



(a) Random walk generation.

$$\mathcal{W}_{v_4} = \begin{matrix} 4 \\ \\ u_k \\ \\ \\ \\ \\ \\ \\ \end{matrix} \begin{bmatrix} 3 \\ 1 \\ 5 \\ 1 \\ \vdots \end{bmatrix} v_j \longrightarrow \Phi$$

(b) Representation mapping.



(c) Hierarchical Softmax.

DeepWalk (Random Walks + Word2vec)

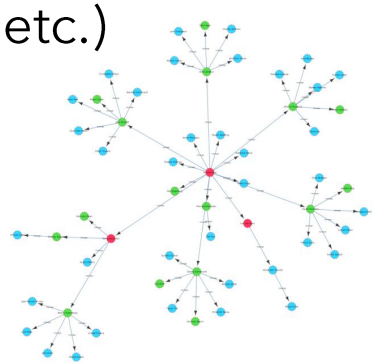
From a node:

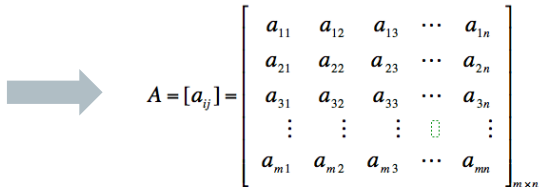
- Do random walks
- Build “sentences of words” by putting together the visited nodes
- Each node will have many “sentences of words”

Apply Word2vec:

- Transform the sentences into a vector for each node
  (applying NLP techniques, distance between words, NN etc.)

Result: each node has a vector representation



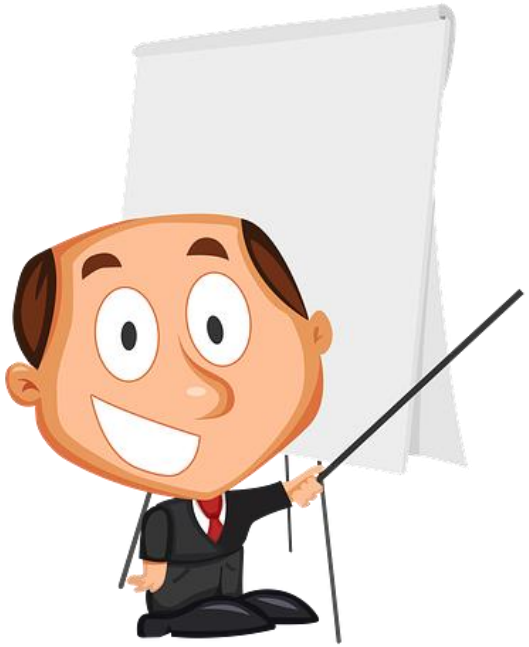


(It’s way more complicated than that, there are many theoretical and practical papers explaining these algorithms)

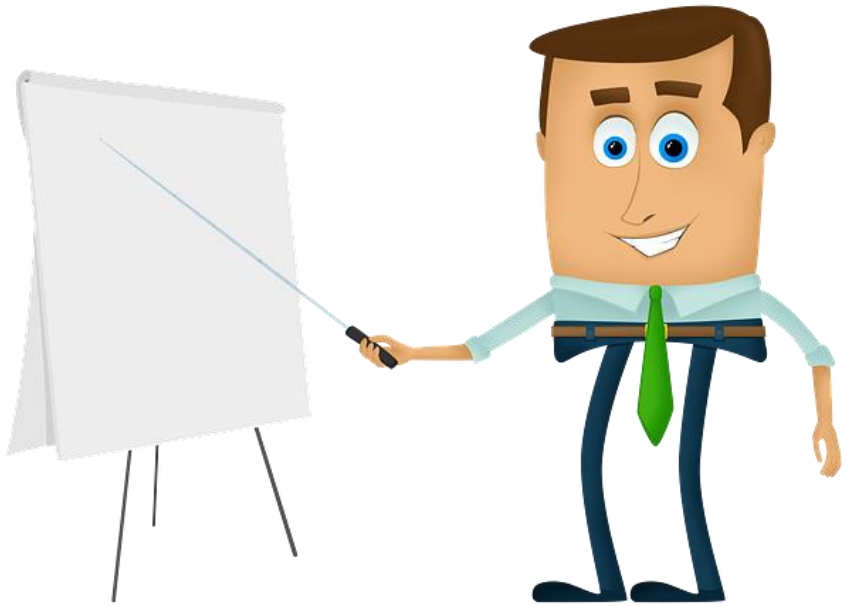
Let’s imagine...

Let's imagine...

Director



Teacher A

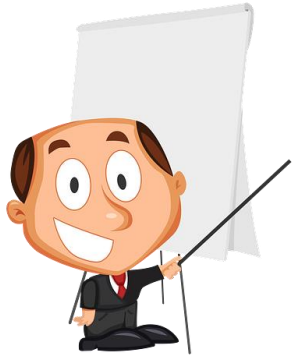


Teacher B

Let's imagine...

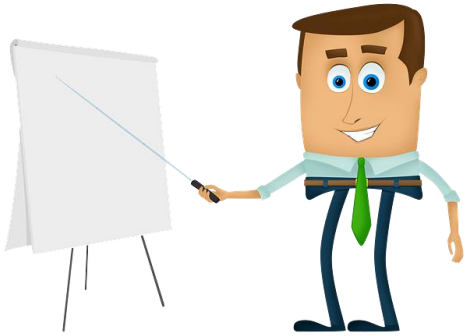
Director

Teacher A



Students

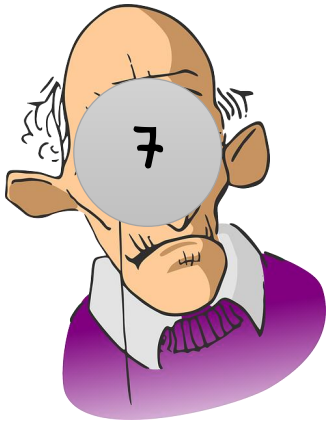
Teacher B



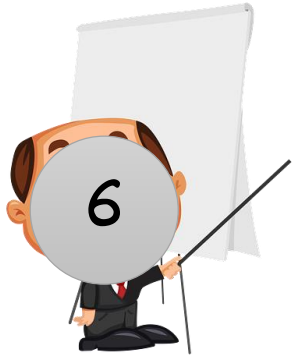
Students

Interactions' graph

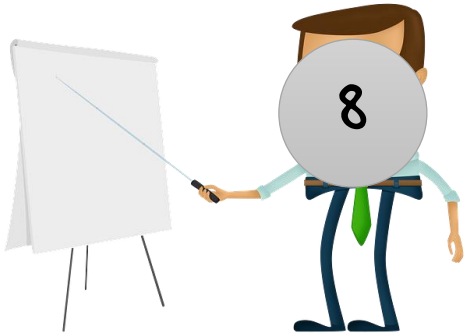
Director

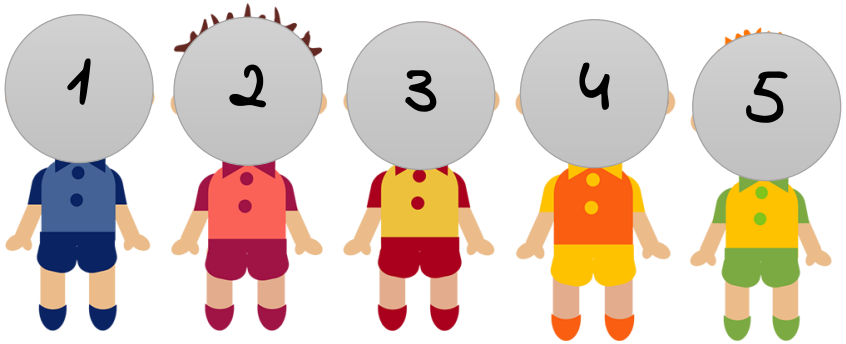


Teacher A

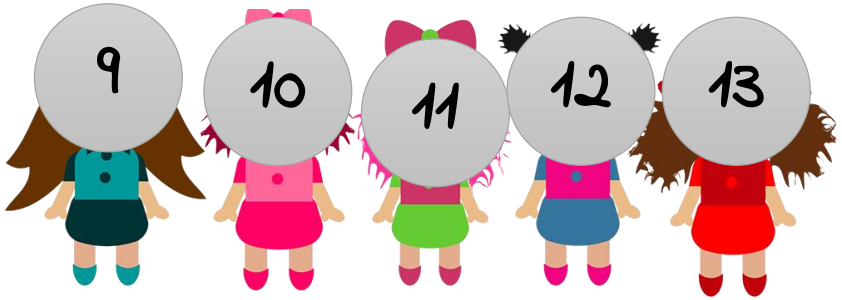


Teacher B



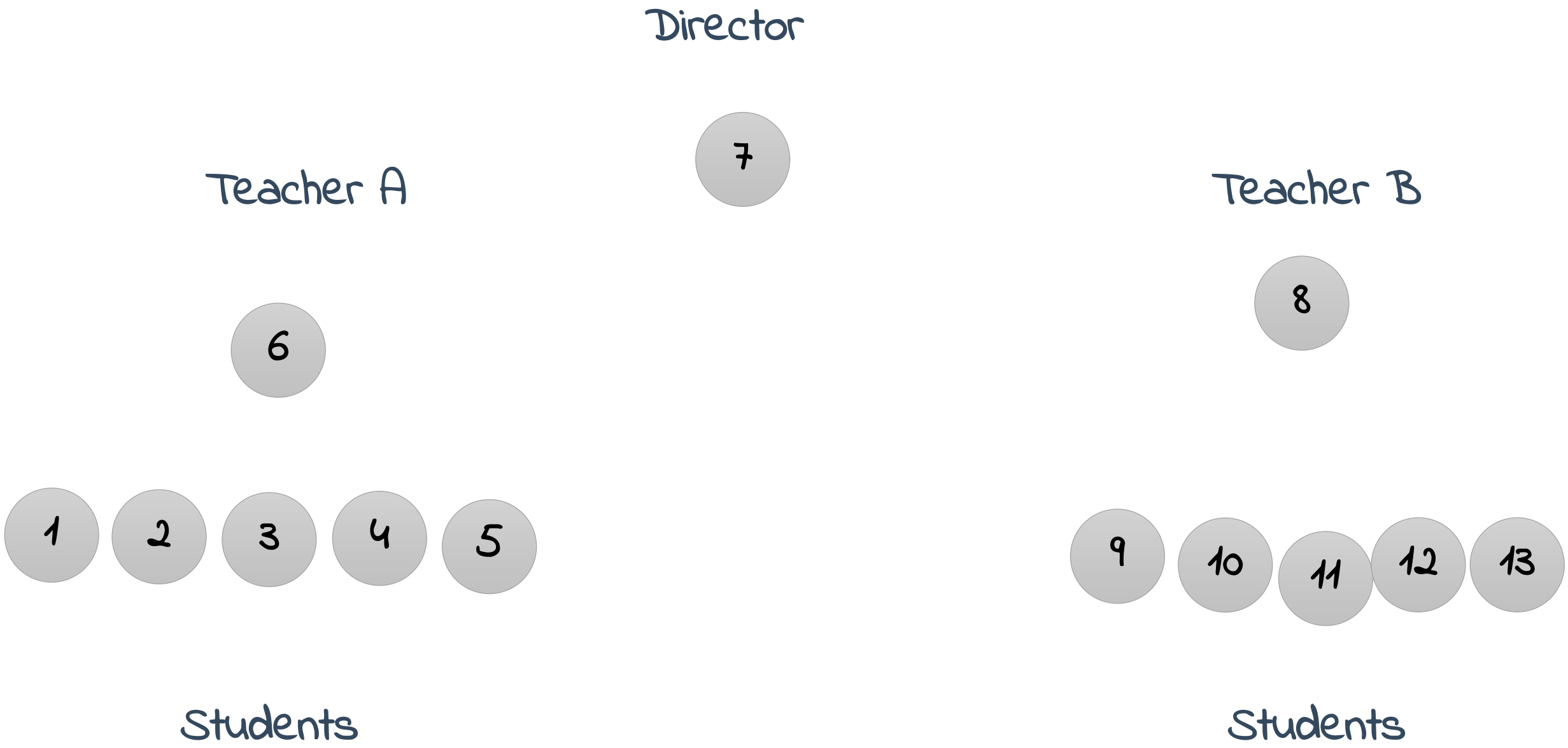


Students

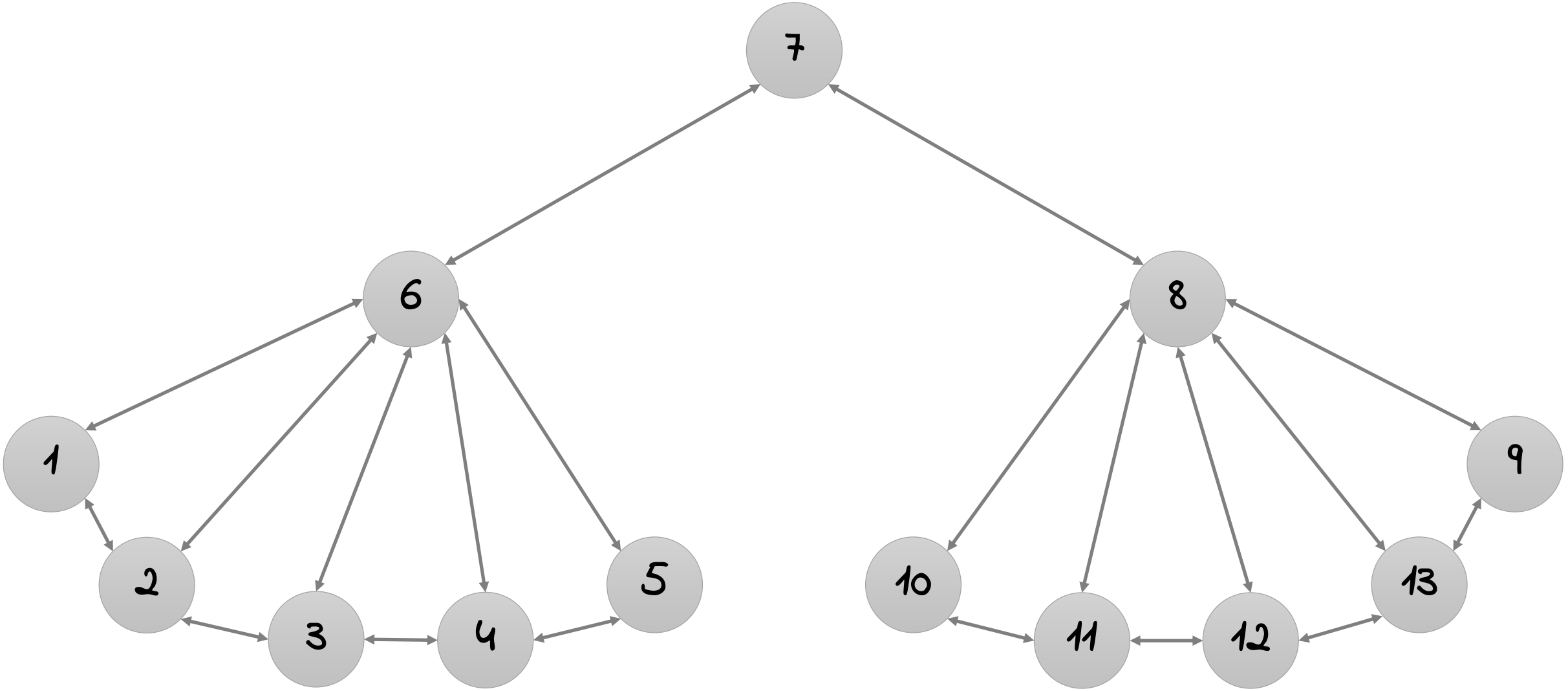


Students

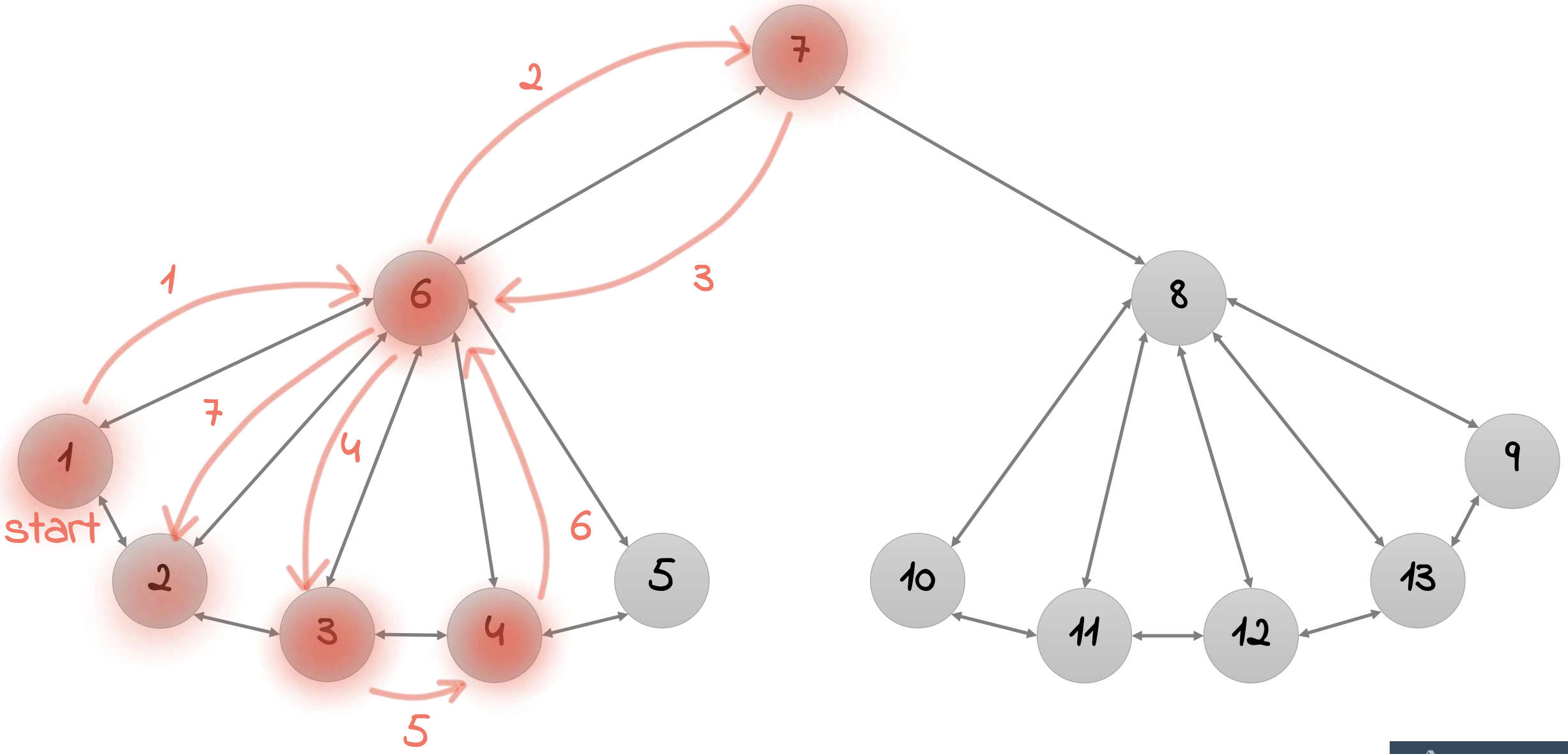
Interactions' graph



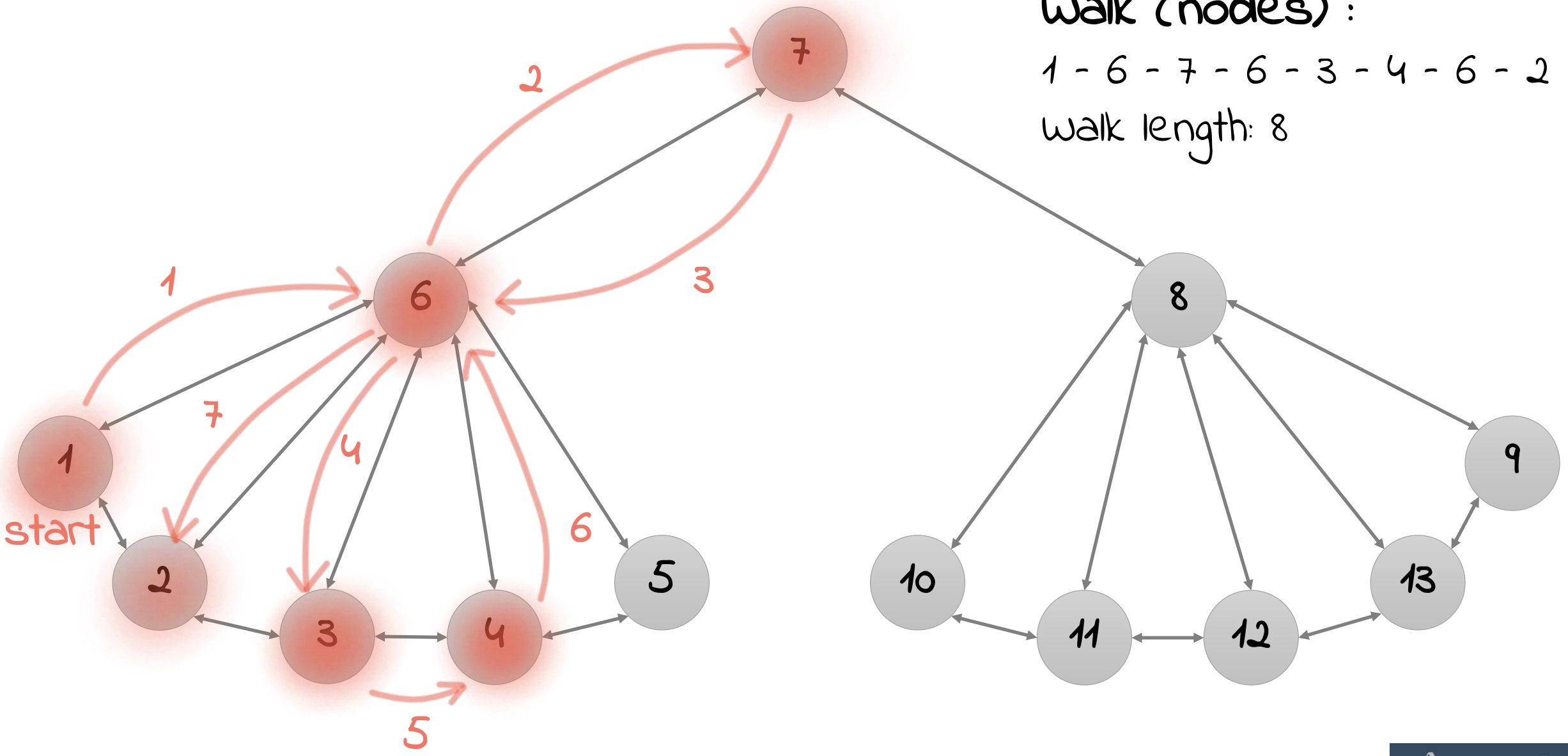
Interactions' graph



Interactions' graph – DeepWalk: step 1, walking randomly ...



Interactions' graph – DeepWalk: step 1, walking randomly ...



Interactions' graph – DeepWalk: step 1, walking randomly …

Some "walking" parameters:

- number of walks per node
- walk length

Result?

A list of "sentences"

- size: ("number of nodes" * "number of walks per node") x "walk length"
- sentences of words: the nodes can be identified by numbers or words, doesn't matter

Interactions' graph – DeepWalk: step 1, walking randomly …

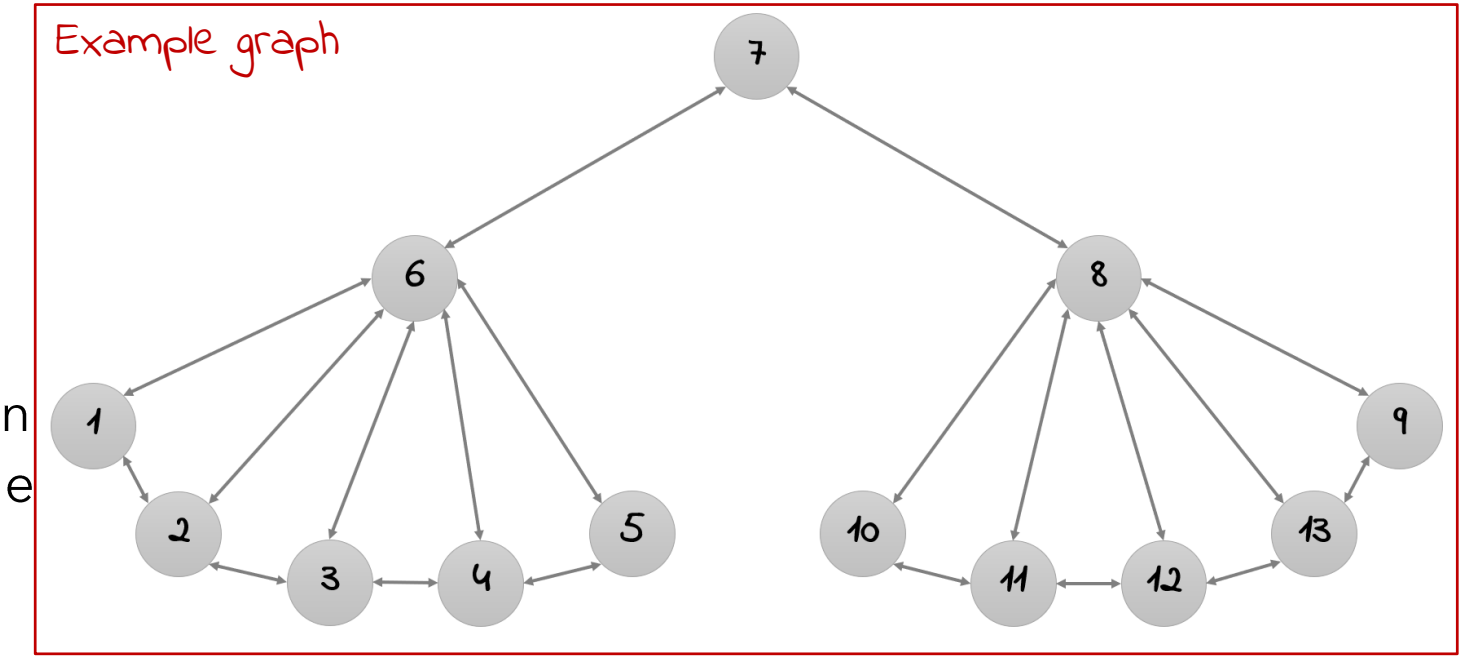
Some "walking" parameters:

- number of walks per node
- walk length

Result?

A list of "sentences"

- size: ("number of nodes" * "n
- sentences of words: the node



In our example:

- *the graph has 13 nodes*
- *with "number of walks per node" = 80, "walk length" = 40*

The result is a list of 1'040 sentences of 40 words each

Interactions' graph – DeepWalk: step 2, turn words into vectors

(in our example)
We have a list of 1'040 sentences of 40 words each, how does that become a vector of features for each node?

Using word embedding, a NLP technique

- NLP: Natural Language Processing
- Word embedding: capturing the context of a word in a document, semantic and syntactic similarity, relation with other words etc.

Word2vec model:

- a word embedding technique generating a vector representation of every single unique word
- The dimensionality of the embedding is a parameter to the model

1) words to dictionary (for a numeric, integer indices, representation of words)
2) feed the "magic" of the model (based on a neural network, a "simple" 3 layers one)
3) the result is a kind of lookup table that maps from integer indices (which stand for specific words) to dense vectors (their embeddings)

Interactions' graph – DeepWalk: step 2, turn words into vectors

Word2vec looks at which words are nearby to which other, using a neural network

A Word2vec "key" parameter:

- window size: how many words behind and ahead to consider

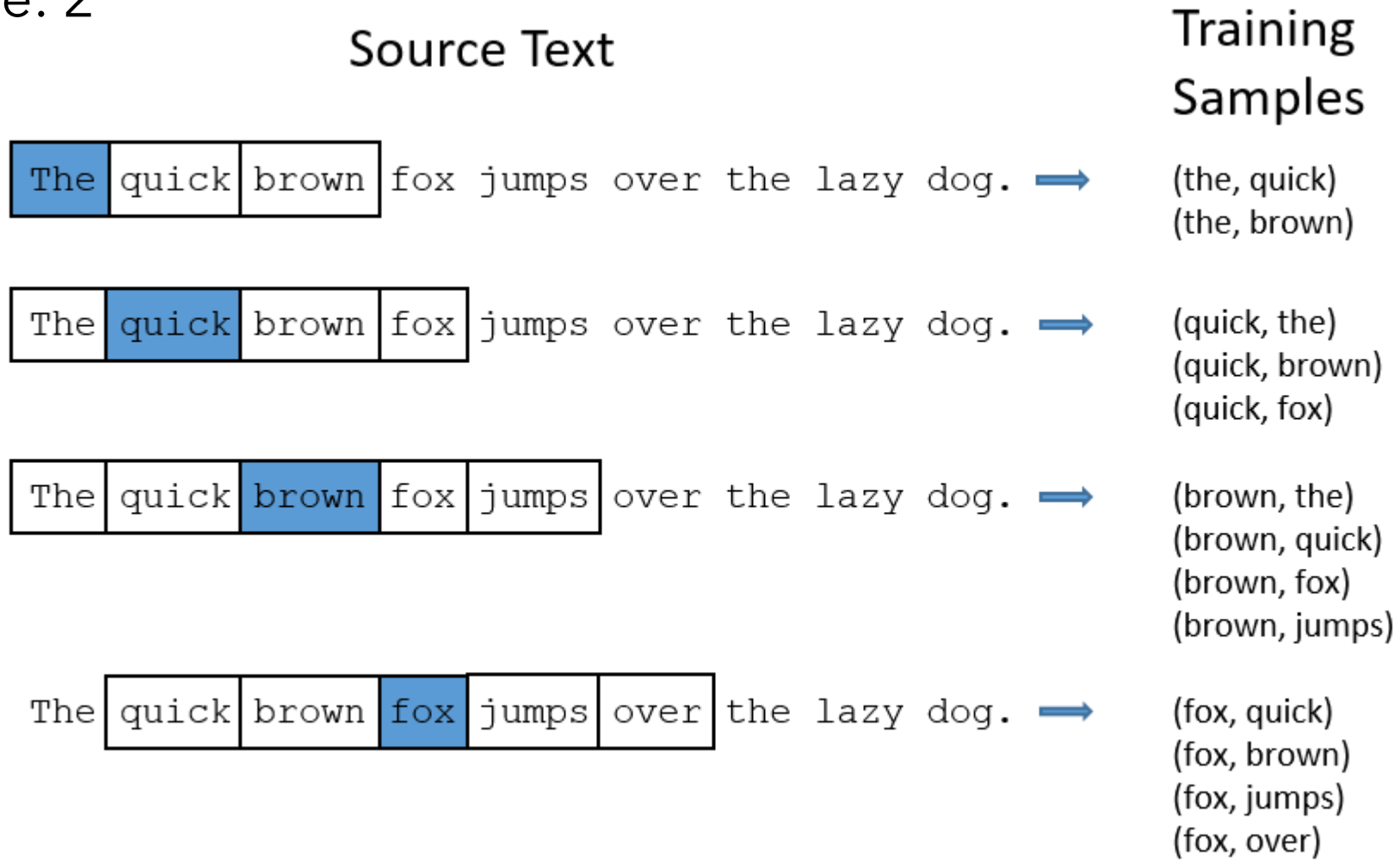
There are many other parameters which influence the neural network and the model itself, things like batch size, negative sampling, number of epochs, sampling rate etc.

All the settings of the Oracle PGX implementation of DeepWalk:
https://docs.oracle.com/cd/E56133_01/latest/javadocs/oracle/pgx/api/beta/mllib/DeepWalkModelBuilder.html

Interactions' graph – DeepWalk: step 2, turn words into vectors

sentence: "The quick brown fox jumps over the lazy dog"

window size: 2



Interactions' graph – DeepWalk: step 2, turn words into vectors

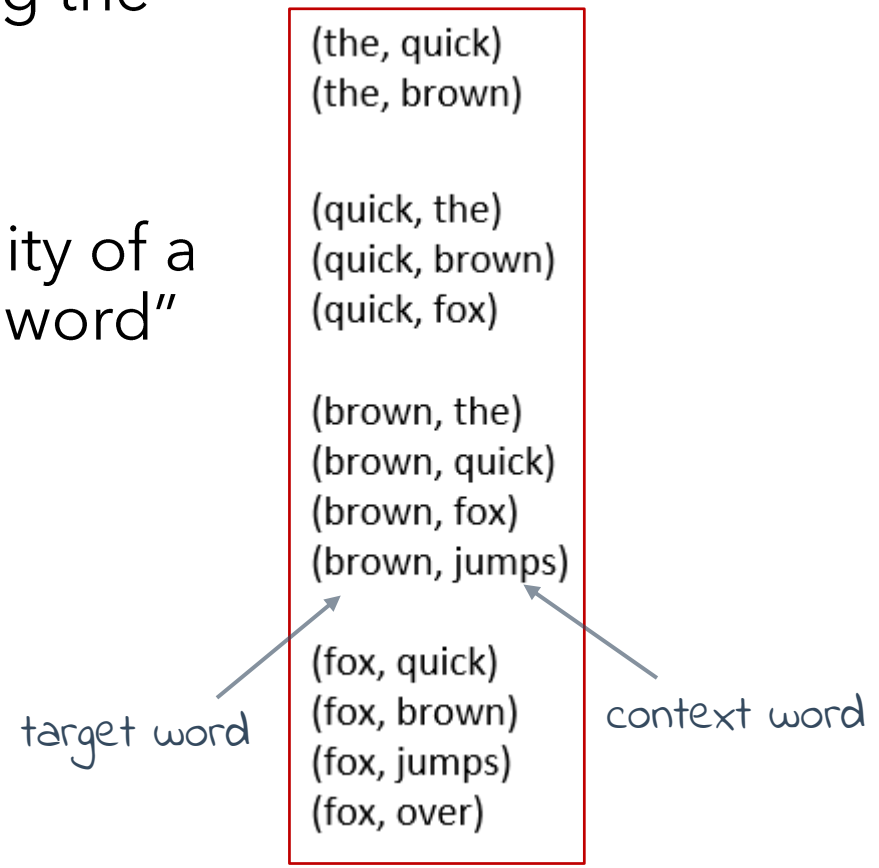
Word2vec uses the pairs of words obtained by applying the "windowing"

The objective of the model is to maximise the probability of a "context word" to be predicted as context for a "target word"

Result:

vertex 1 = $[v_1, v_2, v_3, \dots, v_n]$
vertex 2 = $[v_1, v_2, v_3, \dots, v_n]$
vertex 3 = $[v_1, v_2, v_3, \dots, v_n]$
vertex 4 = $[v_1, v_2, v_3, \dots, v_n]$
vertex 5 = $[v_1, v_2, v_3, \dots, v_n]$
…

n = layer size (by default 200 for Deepwalk in PGX)



(the details of the word2vec implementation are of out of the scope of this presentation and would take too long to cover)

DeepWalk … for real!

DEMO

DeepWalk

Using DeepWalk every node of the graph has a vector representation

- Putting together all the nodes we get a matrix
  (1 row = 1 node, 1 column = 1 component of the vector)
  - Size of the vectors is a parameter of DeepWalk, by default it is 200
- This can be used as source to apply “classical” machine learning algorithms
- Or directly in PGX by calling the “computeSimilars” method on the model

DeepWalk

DeepWalk on the DBpedia graph as an example (with 8’637’721 vertices and 165’049’964 edges) :

```
pgx> var similars = model.computeSimilars("Albert_Einstein", 10)
pgx> similars.print()
+-------------------------------------+
| dstVertex          | similarity     |
+-------------------------------------+
| Albert_Einstein    | 1.0000001192092896 |
| Physics            | 0.8664291501045227 |
| Werner_Heisenberg  | 0.8625140190124512 |
| Richard_Feynman    | 0.8496938943862915 |
| List_of_physicists | 0.8415523767471313 |
| Physicist          | 0.8384397625923157 |
| Max_Planck         | 0.8370327353477478 |
| Niels_Bohr         | 0.8340970873832703 |
| Quantum_mechanics  | 0.8331197500228882 |
| Special_relativity | 0.8280861973762512 |
+-------------------------------------+
```

DeepWalk

Using DeepWalk every node of the graph has a vector representation

- Putting together all the nodes we get a matrix
  (1 row = 1 node, 1 column = 1 component of the vector)
  - Size of the vectors is a parameter of DeepWalk, by default it is 200
- This can be used as source to apply "classical" machine learning algorithms
- Or directly in PGX by calling the "computeSimilars" method on the model

But …

What about properties? In a Property Graph every node can have a dynamic set of properties. Edges can have labels and properties too.

The "original" definition of DeepWalk doesn't care at all about that.

DeepWalk focus on the structure rather than the content…

Paragraph2Vec

Similar to DeepWalk, but on a sub-graph level instead of node

- Each sub-graph will be considered as a paragraph
- Generate random walks on each sub-graphs

Oracle improved Paragraph2Vec with their own “secret sauce” :

- Consider multiple properties (rather than a single one)
- When generated traces, consider edges (instead of vertices) as words
- Attach global properties of each sub-graph (like the size etc.)

Conclusion

- PGX is actively developed by Oracle Labs
  - Comes with great tutorials and documentation
- Has ~60 graphs algorithms out of the box
- ML algorithms are being implemented (DeepWalk, Oracle-custom-pg2vec)
  - Beta since PGX 19.1.0 (still beta at the moment)

The best results are achieved by a mix of techniques:
Use the graph to generate features and feed the whole result set to a ML Framework (TensorFlow, scikit-learn etc.) At least for now...