Graph & Network Analytics

The power of relations

Vandy BERTEN
Smals - Section Recherche
20 février 2019 - UMONS

Smals
ICT for society

Définir – Caractériser – Visualiser –  – Interroger

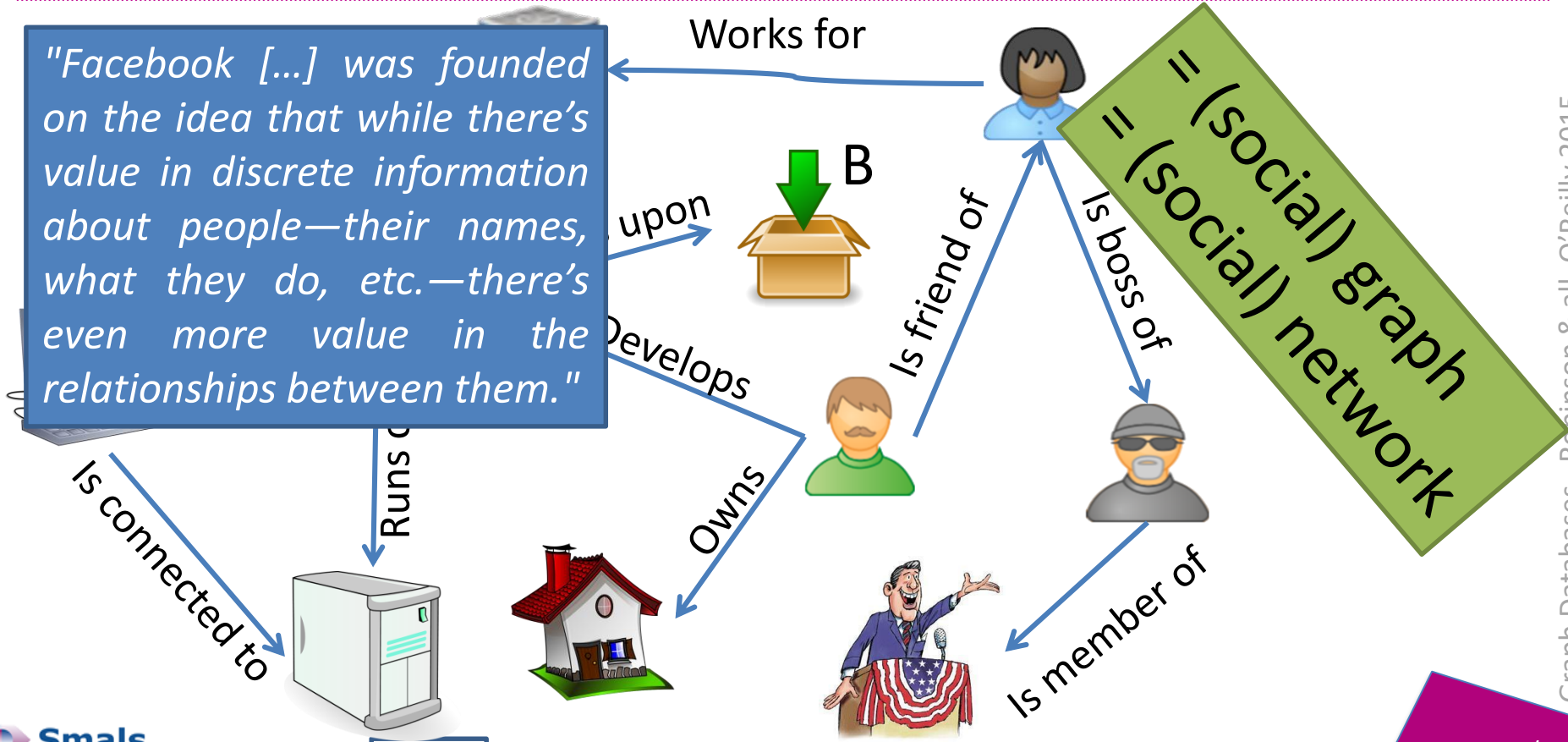
Table des matières

Définir un réseau

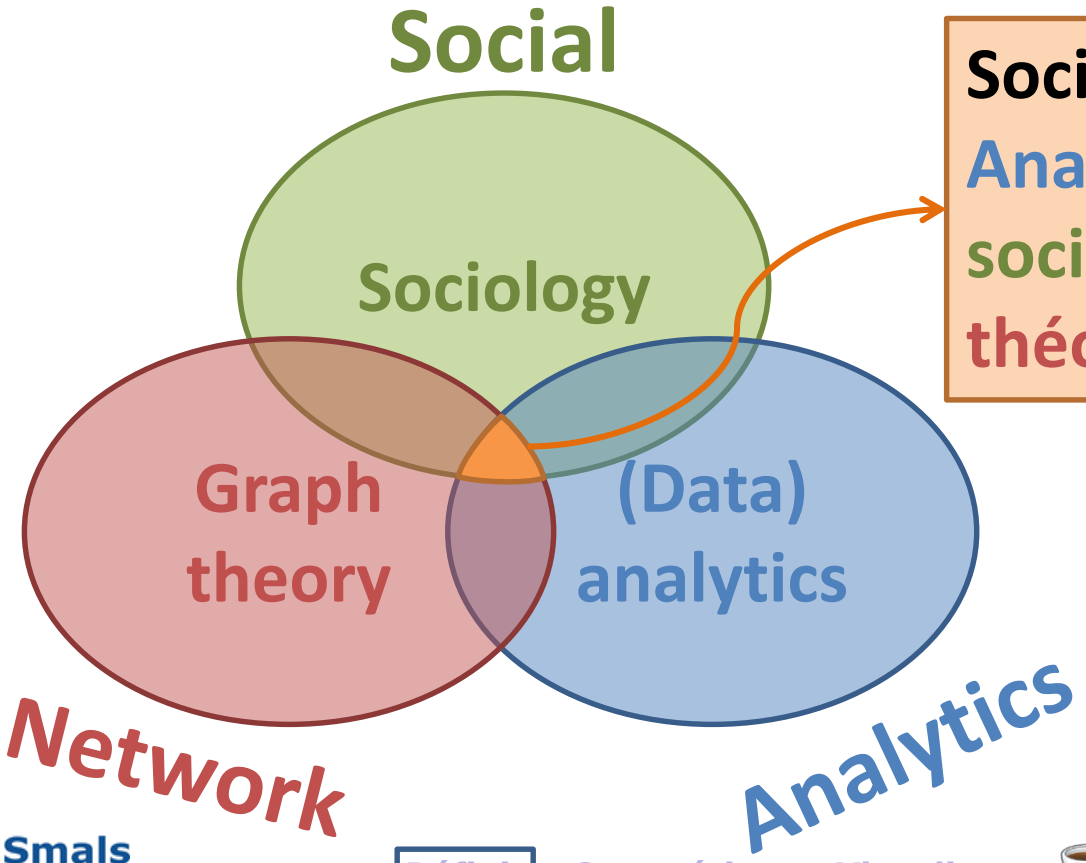
Définir un réseau

MOTIVATIONS

Motivations

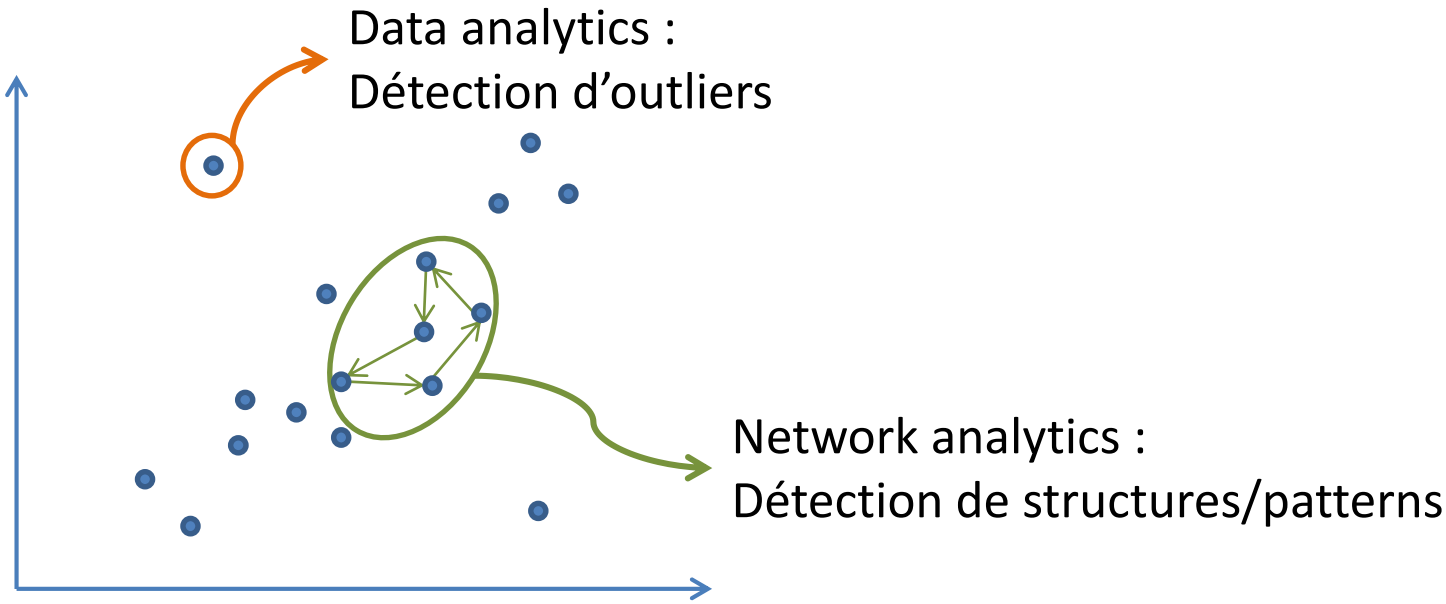


Social Network Analytics



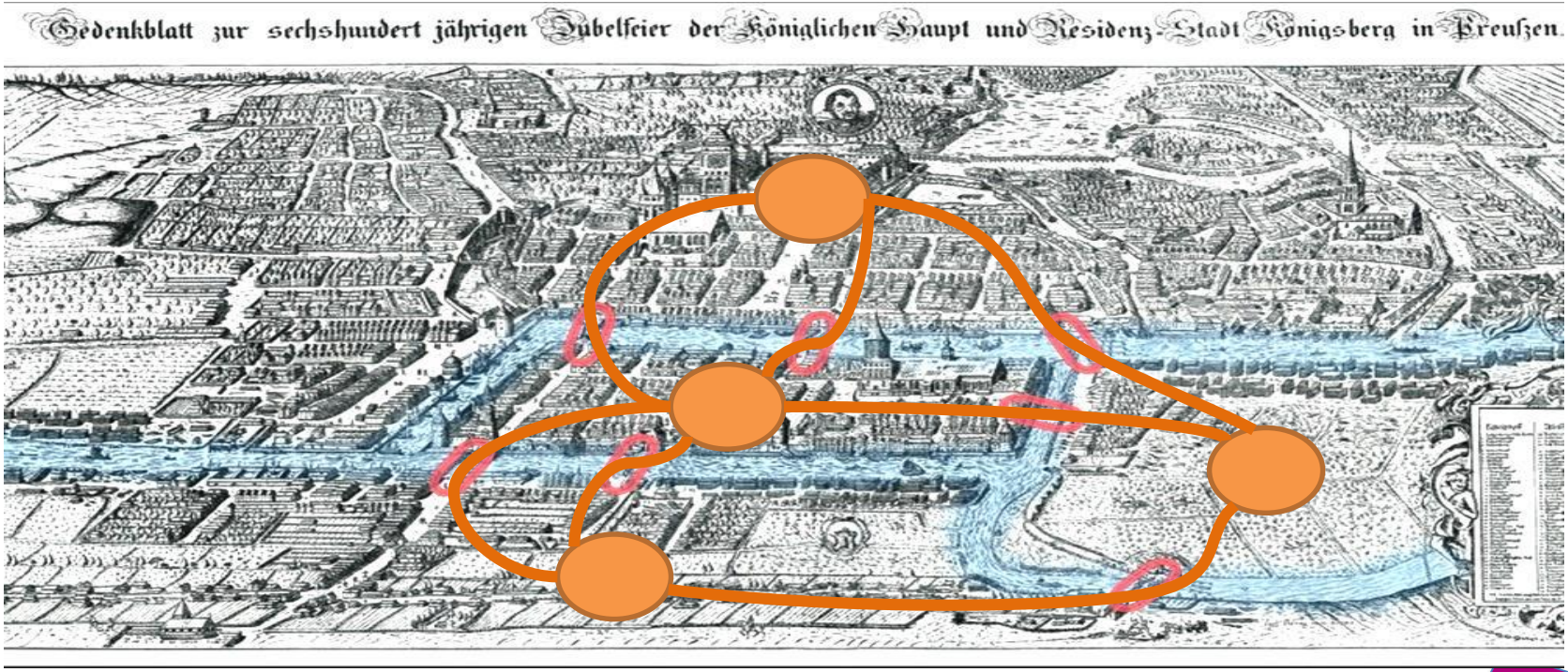
Social Network Analytics :
Analyse des **structures sociales** au travers de la **théorie des graphes**

Avantages de Network Analytics



(Social) Graph/Network Analytics is not…

7 ponts de Königsberg (Euler, 1736)



Gartner : applications

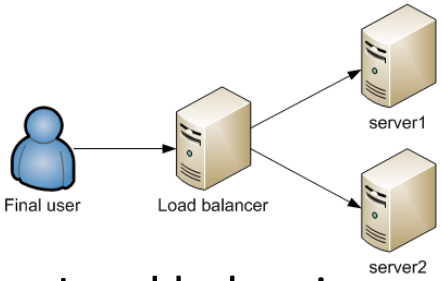
Route optimization

Epidemiology

Intelligence

Supply chain

Fraud detection



Load balancing

Market basket analysis

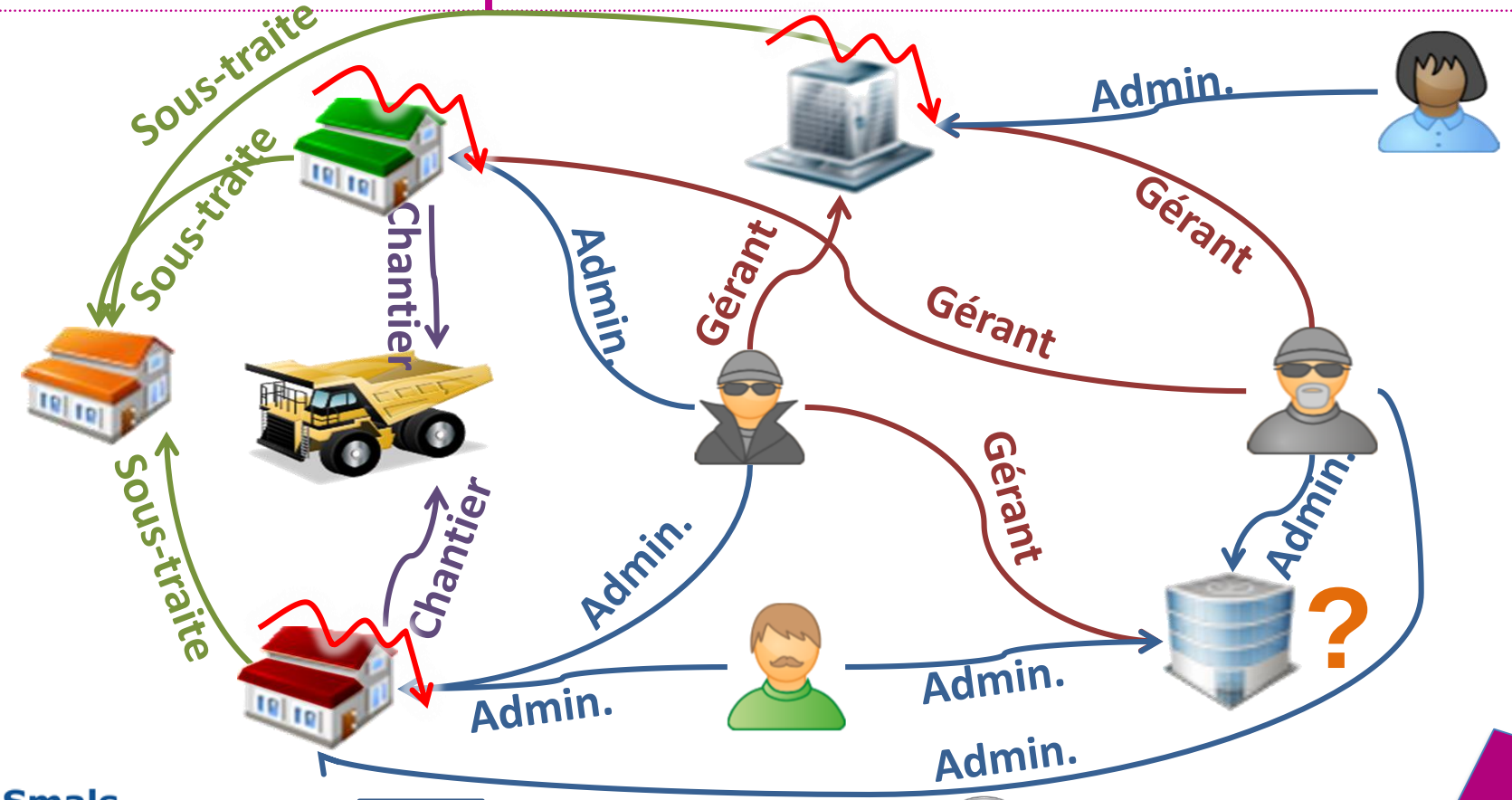
Genome

Money laundering

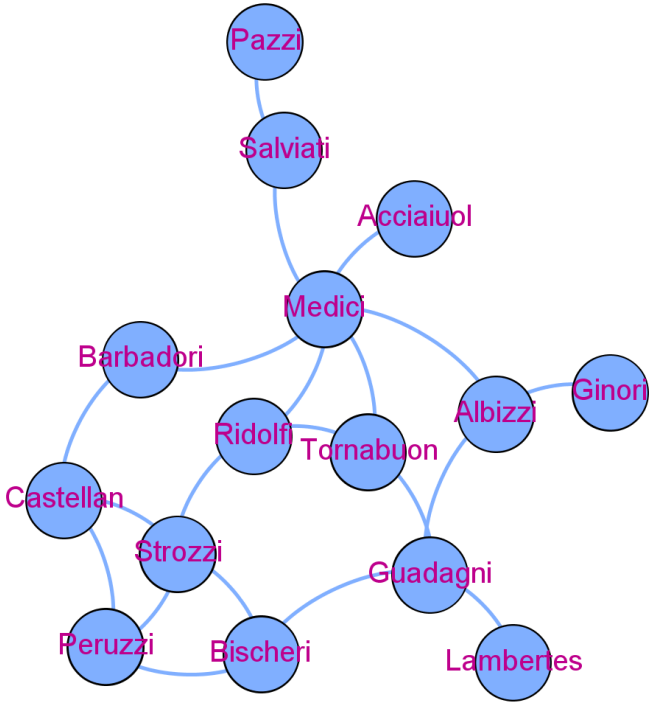
Définir un réseau

EXEMPLES

Spider construction

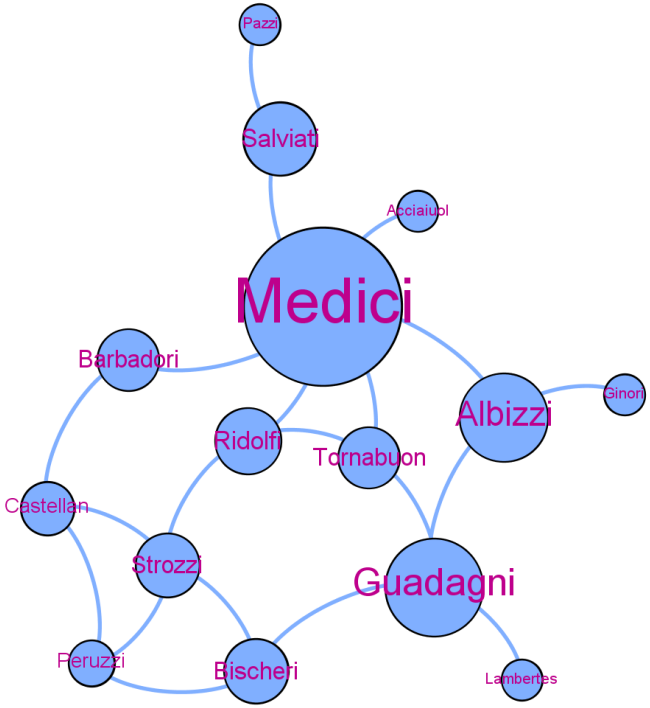


Mariages à Florence



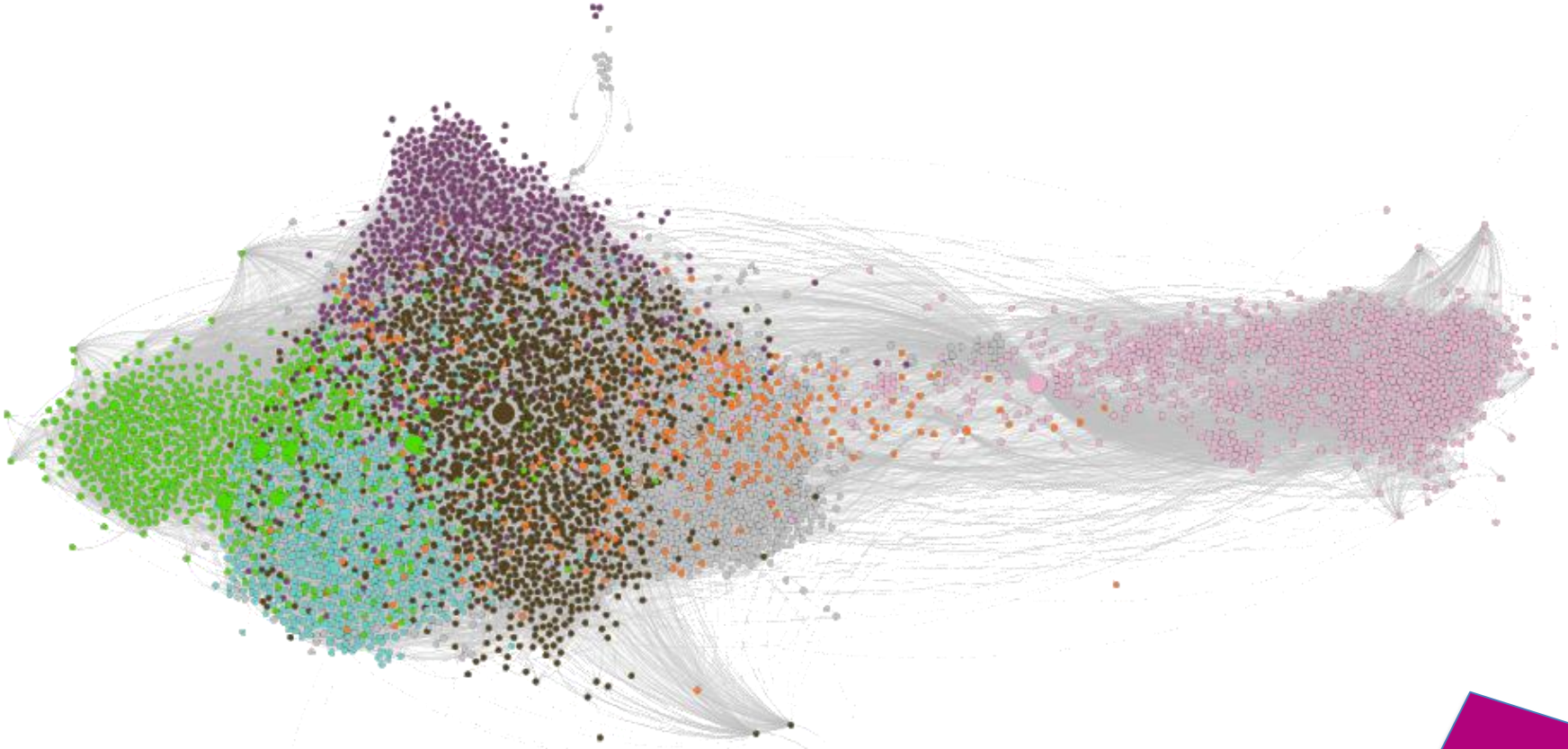
Padgett and Ansell, Robust Action and the Rise of the Medici, 1993

Mariages à Florence



Padgett and Ansell, Robust Action and the Rise of the Medici, 1993

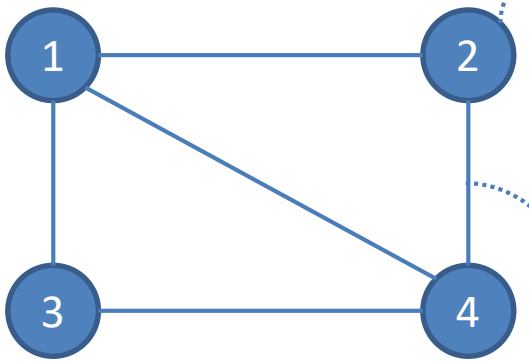
Dépendance de librairies Python



Définir un réseau

DÉFINITIONS

Nœuds, arcs

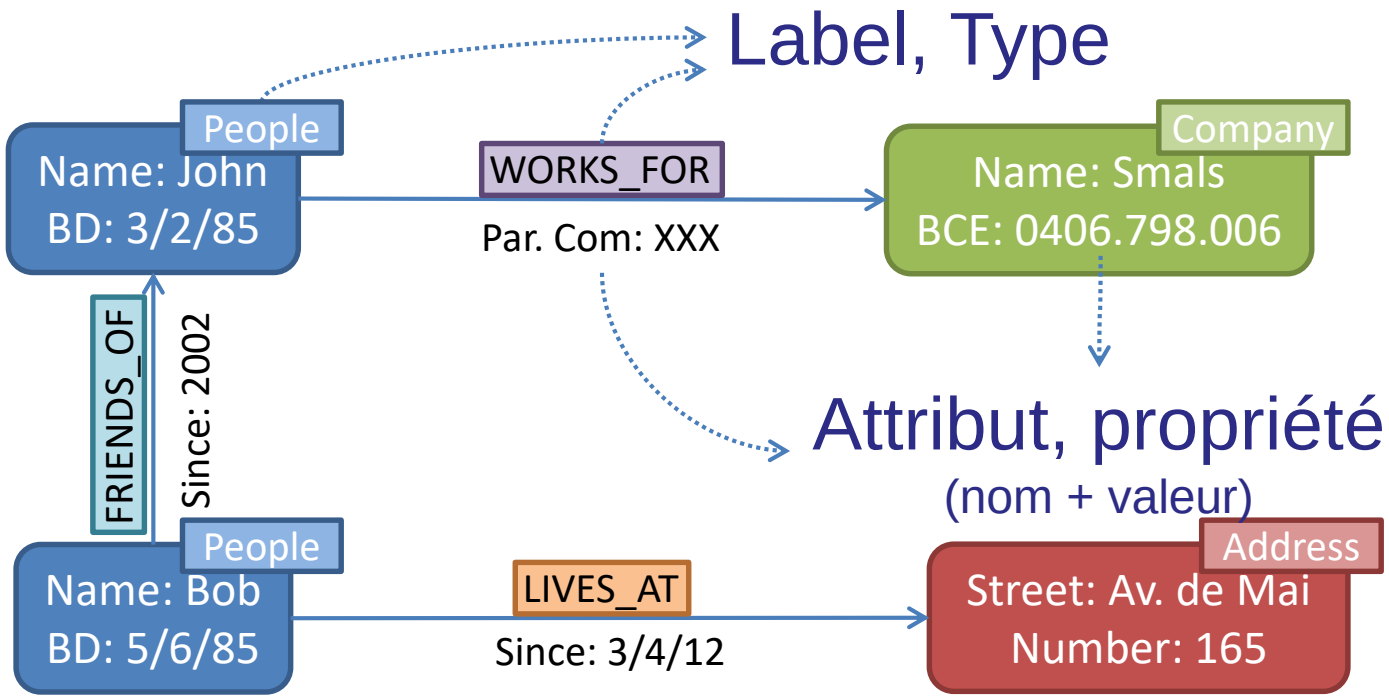


Nœud, sommet, entité, *vertex*

Arc, relation, lien, connexion, *edge*

$$G = (V, E)$$

Labeled Property Graph

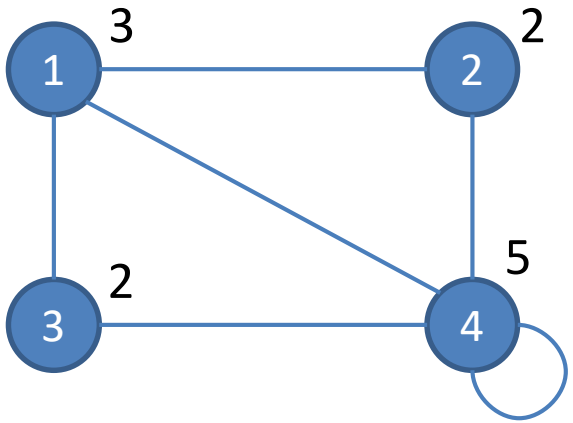


Réseau dirigé/non-dirigé

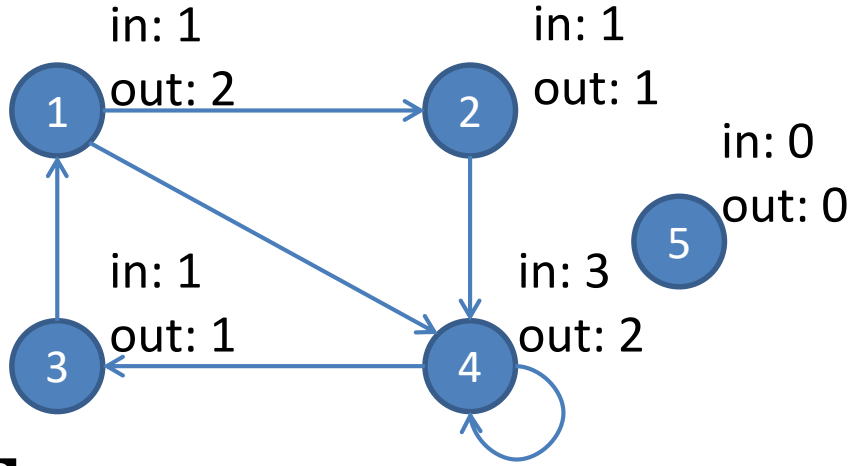
Non dirigé (*undirected*) :
collègue, ami Facebook

Dirigé (*directed*) :
travaille pour, dépend de, follower Twitter

Degré



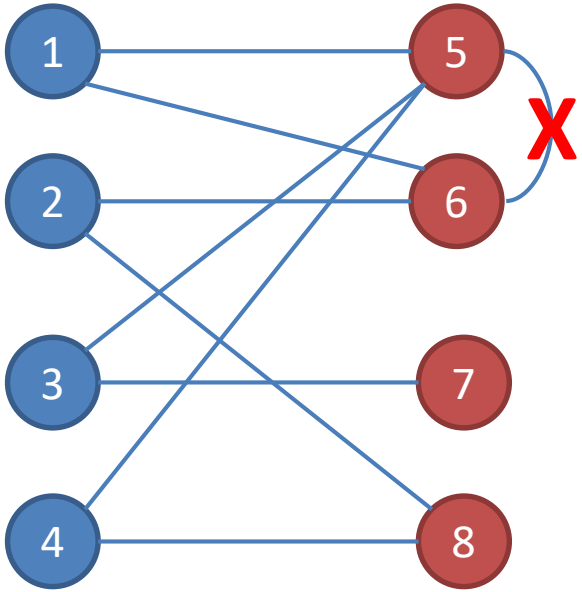
$$\forall v \in V \deg(v) = \deg^+(v) + \deg^-(v)$$



$$\sum_{v \in V} \deg(v) = 2\,|E|$$

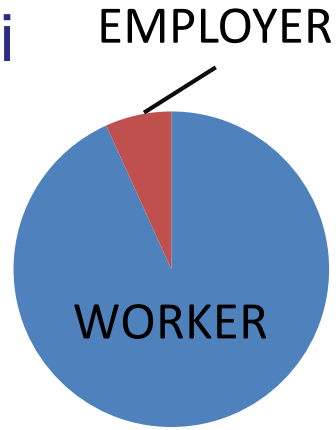
$$\sum_{v \in V} deg^+(v) = \sum_{v \in V} deg^-(v)$$

Graphe biparti *(bipartite graph)*



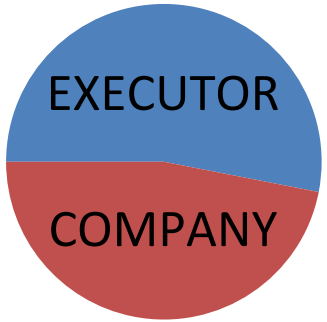
Exemple : Dimona

- Déclarations Dimona de 2007 à aujourd'hui
- 7 048 178 nœuds, 20 640 727 relations
- Graphe biparti
- Degré d'un nœud :
  - EMPLOYER : nombre de ses employés
  - WORKER : nombre de ses employeurs



Exemple : BCE/KBO

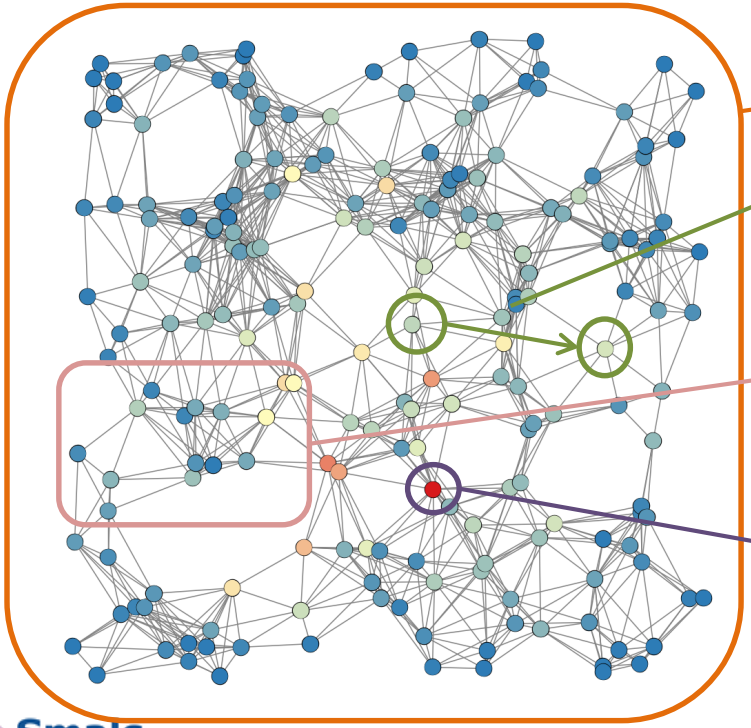
- Fonctions BCE de 2007 à aujourd'hui
- 3 572 876 nœuds, 3 244 227 relations
- Graphe biparti
- Degré d'un nœud :
  - EXECUTOR : nombre de compagnies où il officie
  - COMPANY : nombre de ses « executors » (=mandataires)



Caractériser un réseau

Caractériser un réseau

On peut caractériser :



Le réseau dans son ensemble
(Métriques générales)

La relation/distance/similarité
entre deux nœuds

Un groupe de nœuds (Clustering)

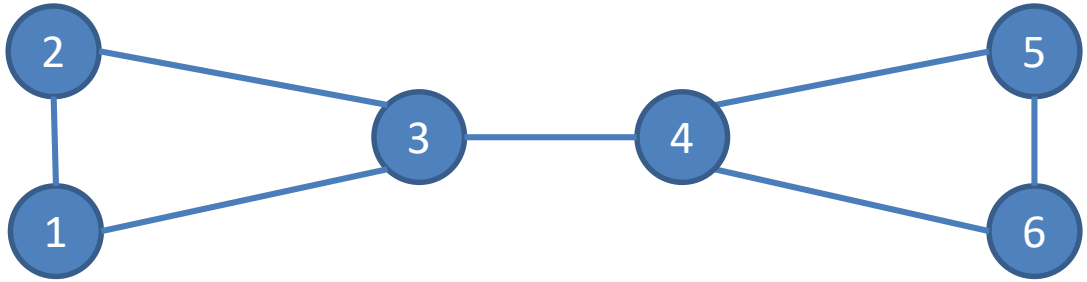
Un nœud en particulier (Centralité)

Caractériser un réseau

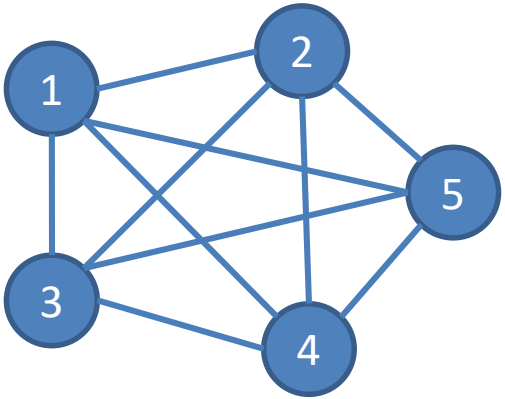
MÉTRIQUES GÉNÉRALES

Diamètre

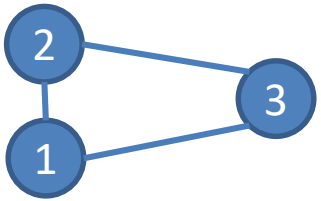
- Diamètre = plus long « plus court chemin »

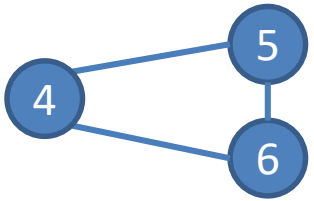


→ 3



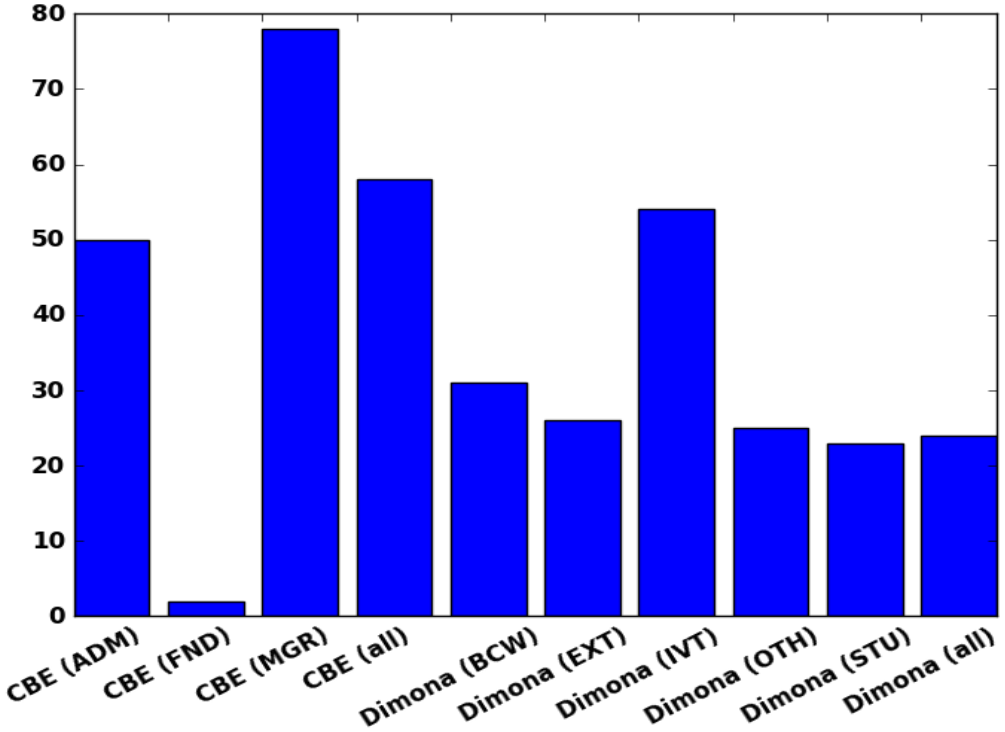
→ 1



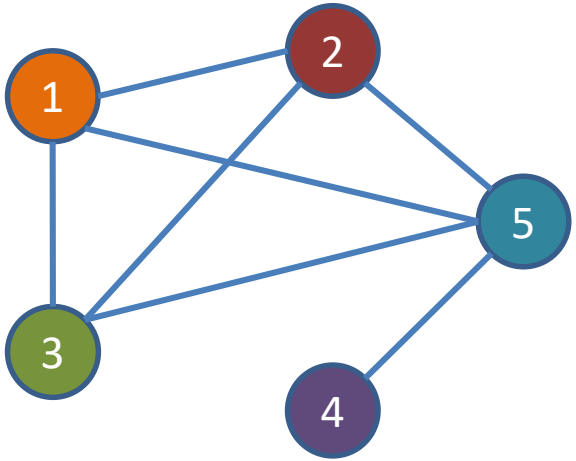


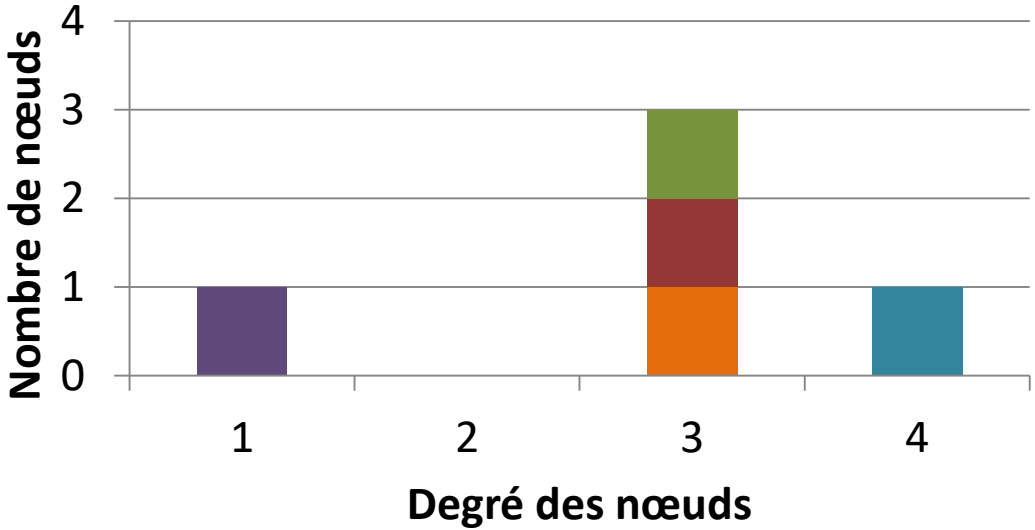
→ ∞

Diamètre

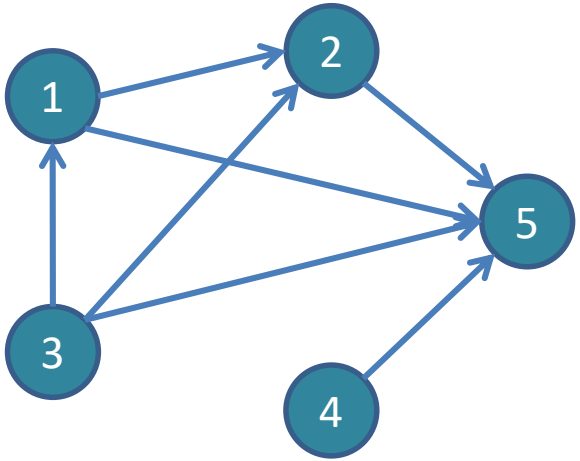

80
70
60
50
40
30
20
10
0
CBE (ADM)
CBE (FND)
CBE (MGR)
CBE (all)
Dimona (BCW)
Dimona (EXT)
Dimona (IVT)
Dimona (OTH)
Dimona (STU)
Dimona (all)

Distribution de degrés



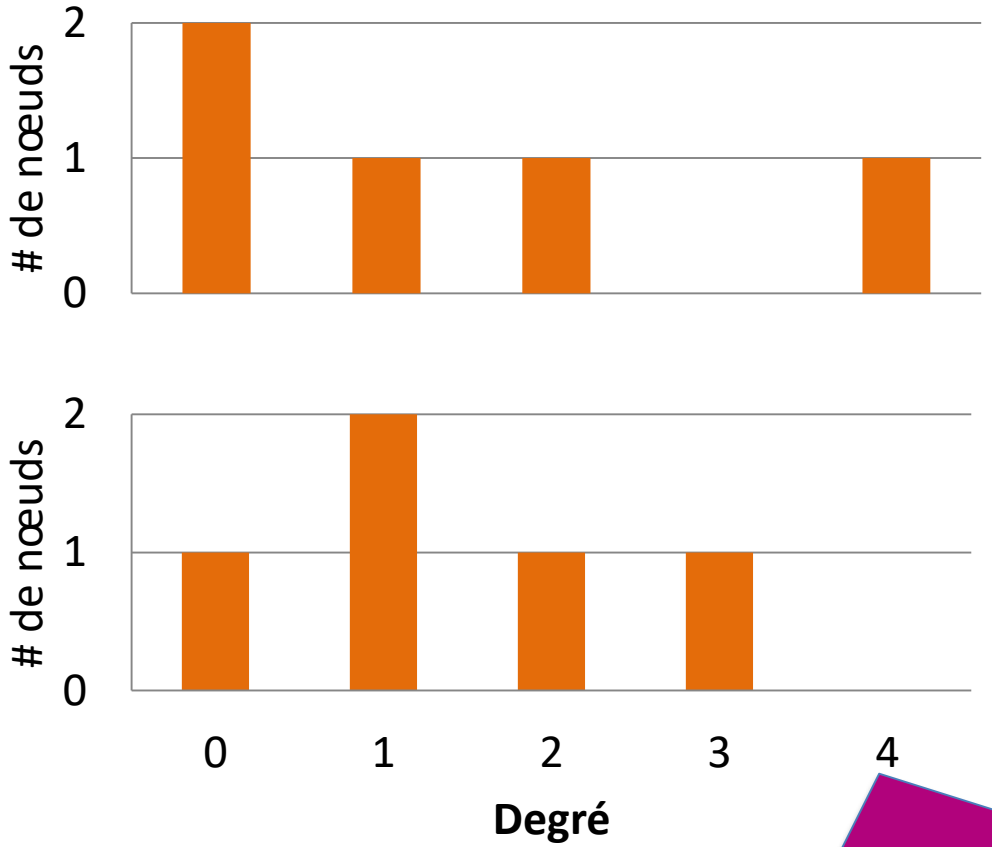


Distribution de degrés (IN vs OUT)

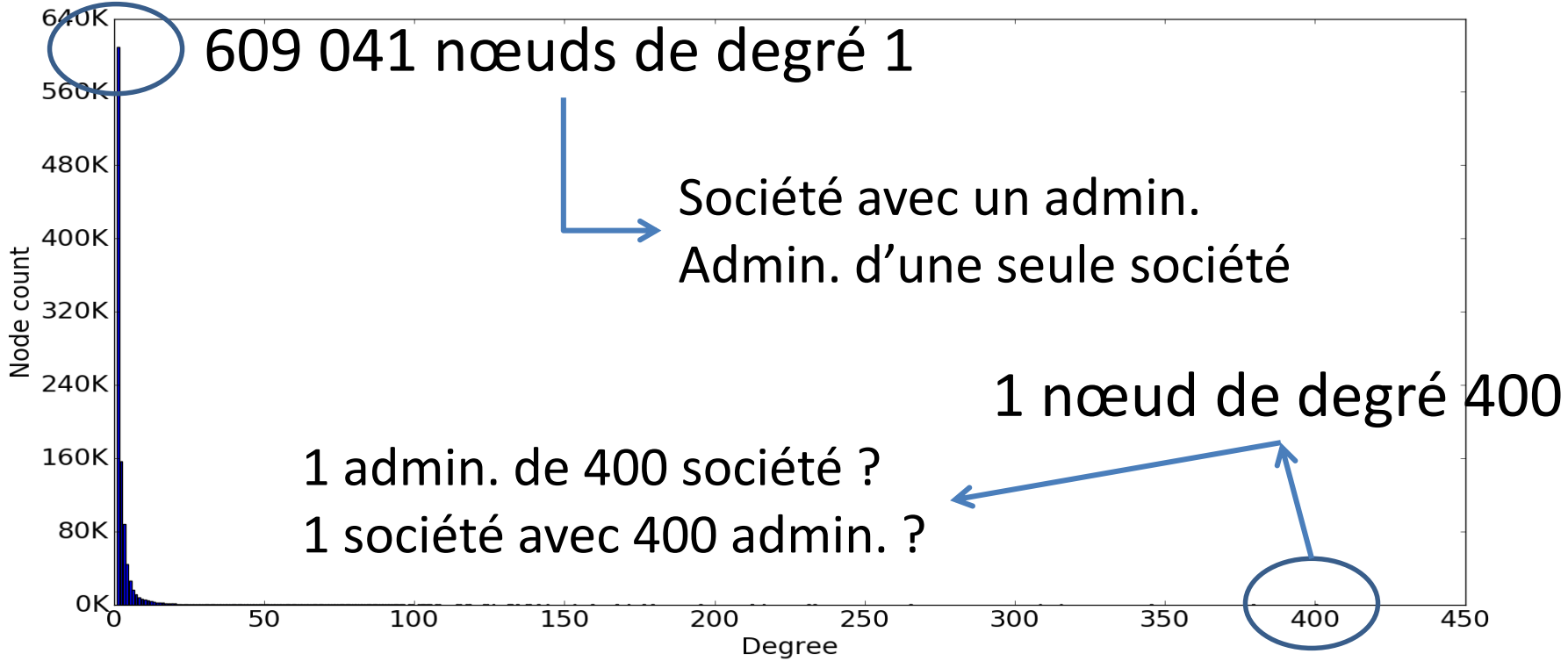


IN:

OUT:

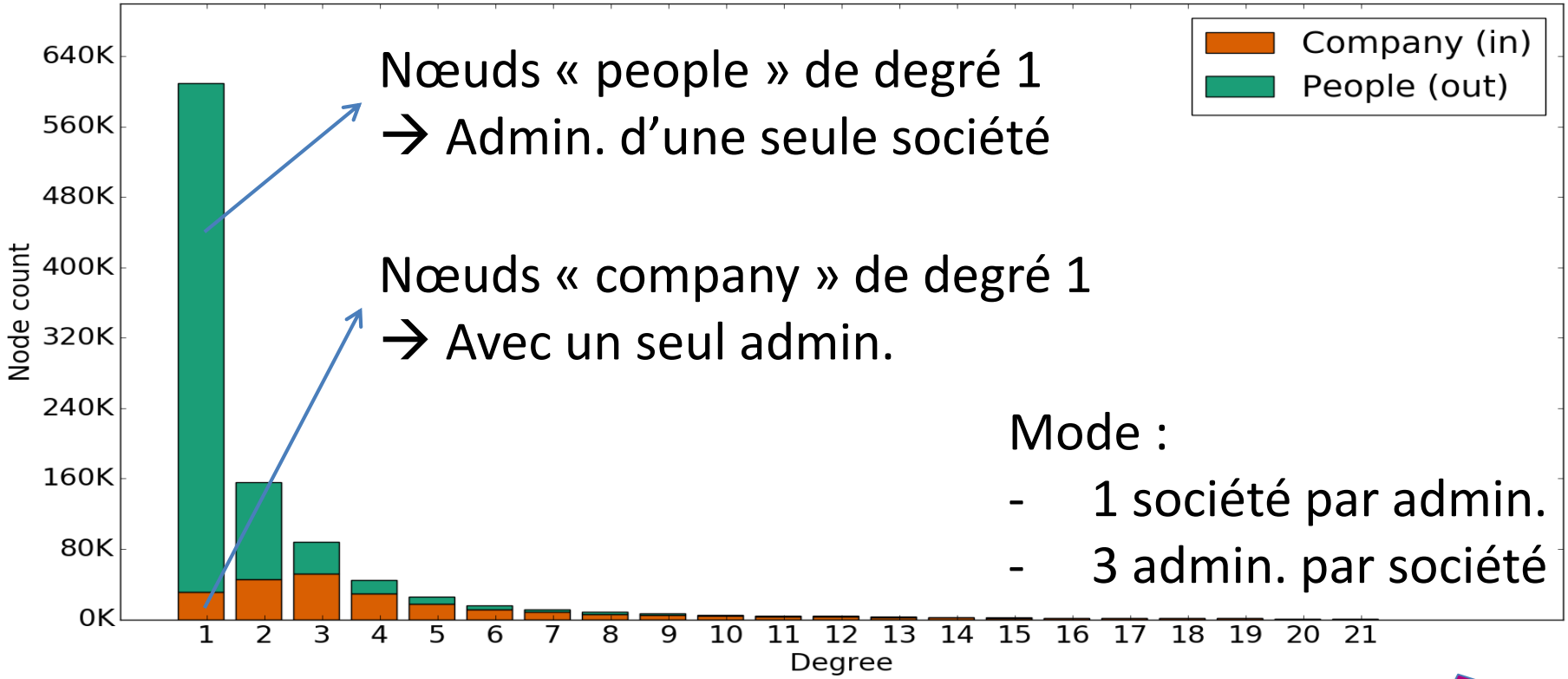


Distribution de degrés : BCE (Adm)



Sur une période de 10 ans

Distribution de degrés : BCE (Adm)



Nœuds « people » de degré 1
→ Admin. d'une seule société

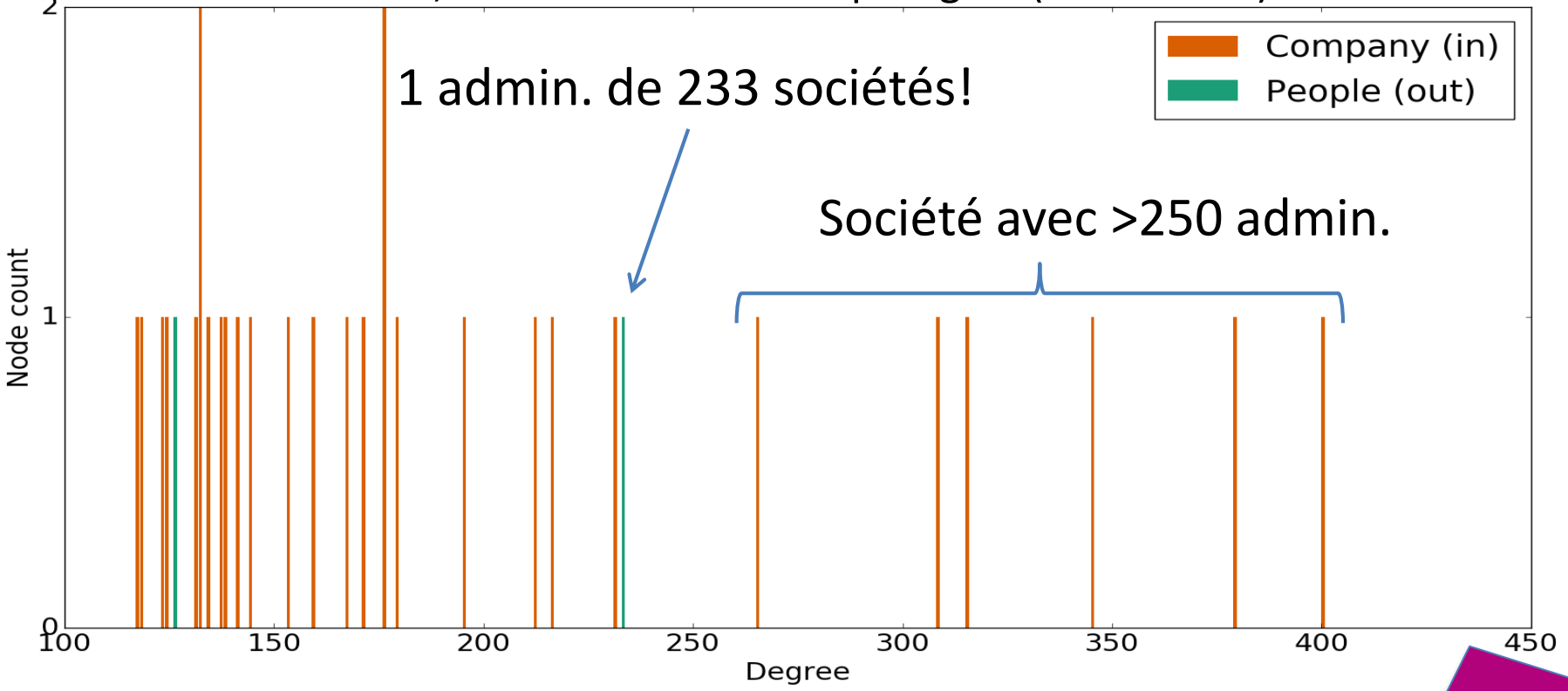
Nœuds « company » de degré 1
→ Avec un seul admin.

Mode :
- 1 société par admin.
- 3 admin. par société

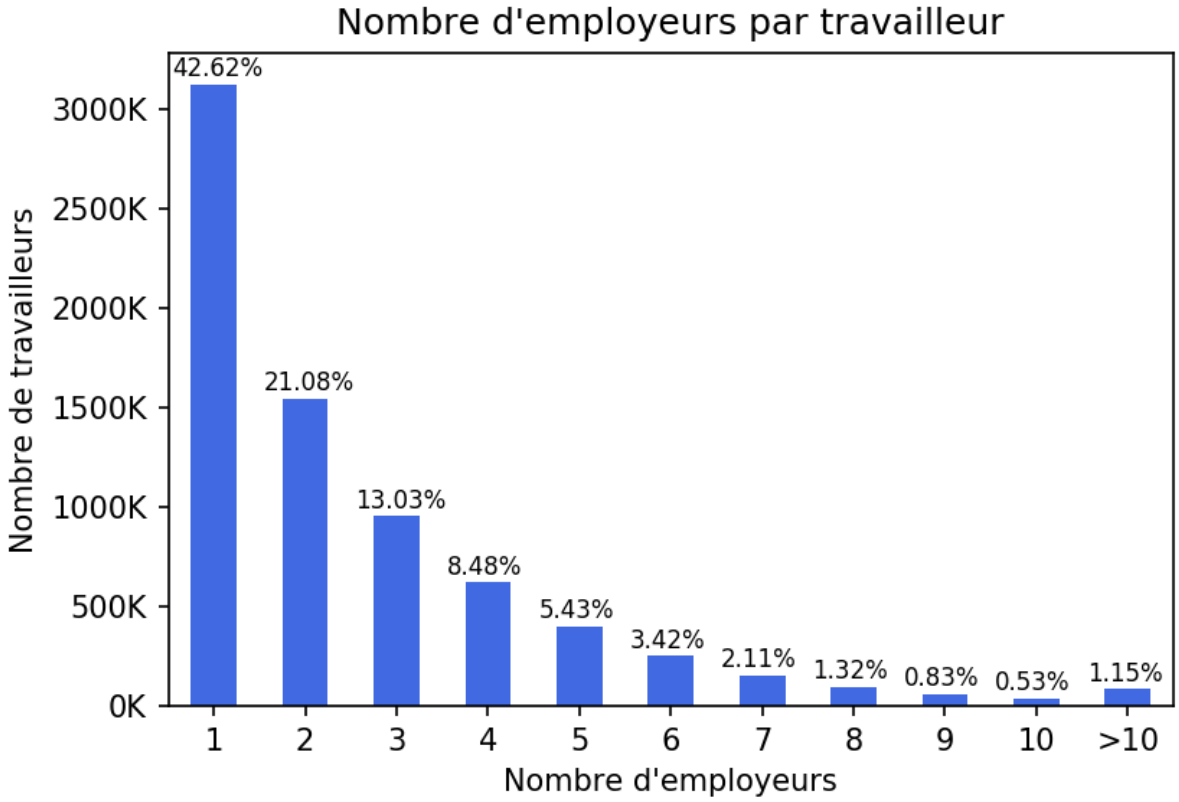
Suppression de 0,5% des nœuds les plus gros

Distribution de degrés : BCE (Adm)

0,003 % des nœuds les plus gros (« outliers »)



Distribution de degrés : Dimona



- 2003-2017
- Max: 110 employeurs !
- 63 : > 50 employeurs

Métriques générales

- Diamètre : étalement du réseau
- Distribution de degré : « forme » du réseau

- Hors présentation :
  - Densité : intensité des relations

Caractériser un réseau

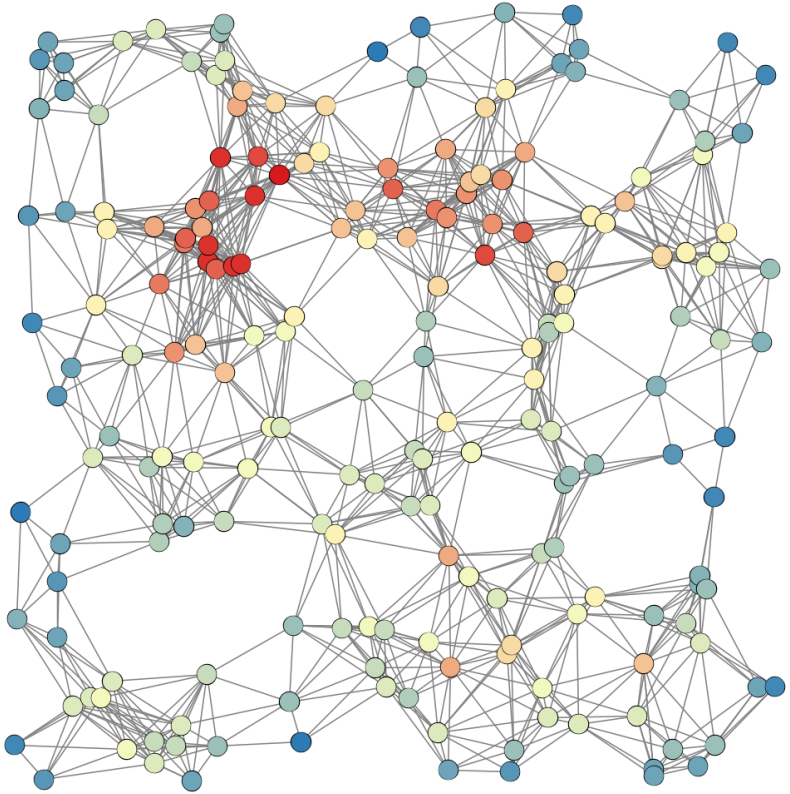
CENTRALITÉ

Centralité

- Centralité d'un nœud = importance d'un nœud au sein du réseau
- Peut se baser sur des caractéristiques :
  - Du nœud isolé (Degree centrality)
  - Du nœud et de son voisinage immédiat (K-core)
  - De l'ensemble du réseau (Betweenness, Page-rank…)

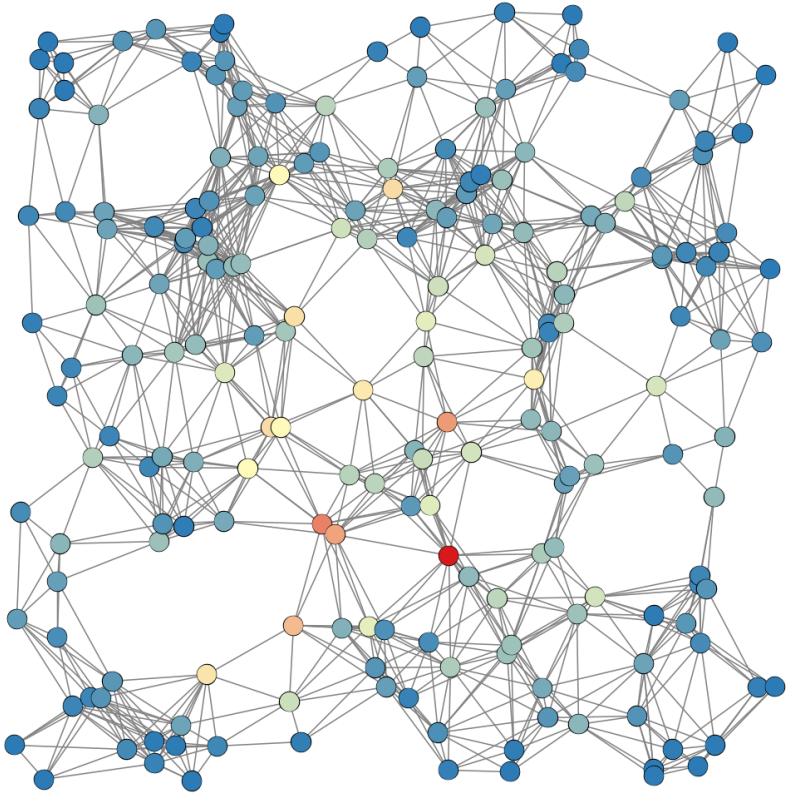
Centralité de degré

Centralité : degré du nœud



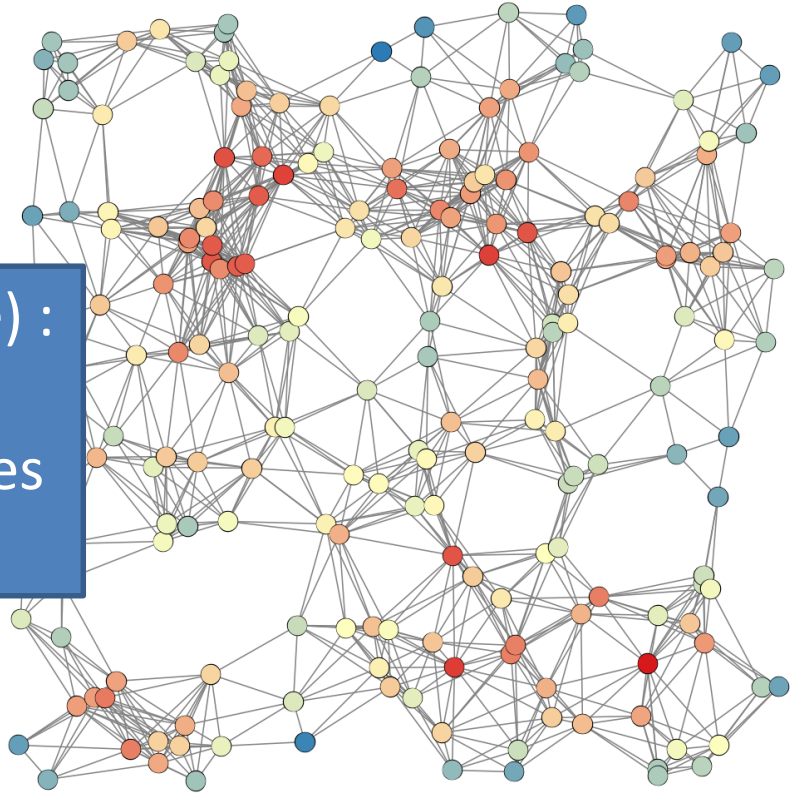
Centralité d'intermédiarité (betweenness)

Centralité : Nombre de « plus courts chemins » passant par le nœud



Page-rank

Page-rank (Google) : Diffusion de son importance vers ses voisins



Centralité

- Degré : nombre de voisins
- Betweenness : importance en tant que relai, connecteur, intermédiaire
- Page-rank : importance par diffusion

- Quoi utiliser ? Dépend très fort du business case

- Hors présentation :
  - K-coreness : nombre de voisins de même k-coreness
  - Closeness : centralité « topologique »

Caractériser un réseau

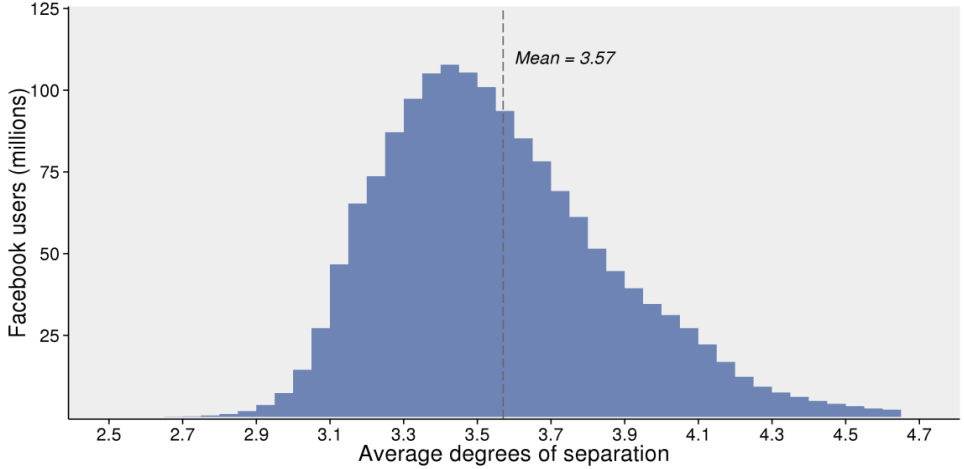
DISTANCE/SIMILARITÉ

Plus court chemin/Distance

- Dijkstra : Calcul du « plus court chemin » entre deux nœuds
- Chaque lien possède un « poids » (longueur, coût…)
- Distance = ∑ poids (=nbr de liens si non pondéré)
- Variante « A* » : pour les réseaux géographiques (utilisé par les GPS)
- Peut ne pas être unique. Le nombre de « plus courts chemins » est aussi une métrique de proximité

Six degree of separation

- Théorie : distance entre chaque personne sur terre ≤ 6 (via 5 connaissances intermédiaires)
- « Degré de séparation » = plus petite distance
- Facebook : moyenne des moyennes : 4,57



https://research.fb.com/three-and-a-half-degrees-of-separation/

Distance

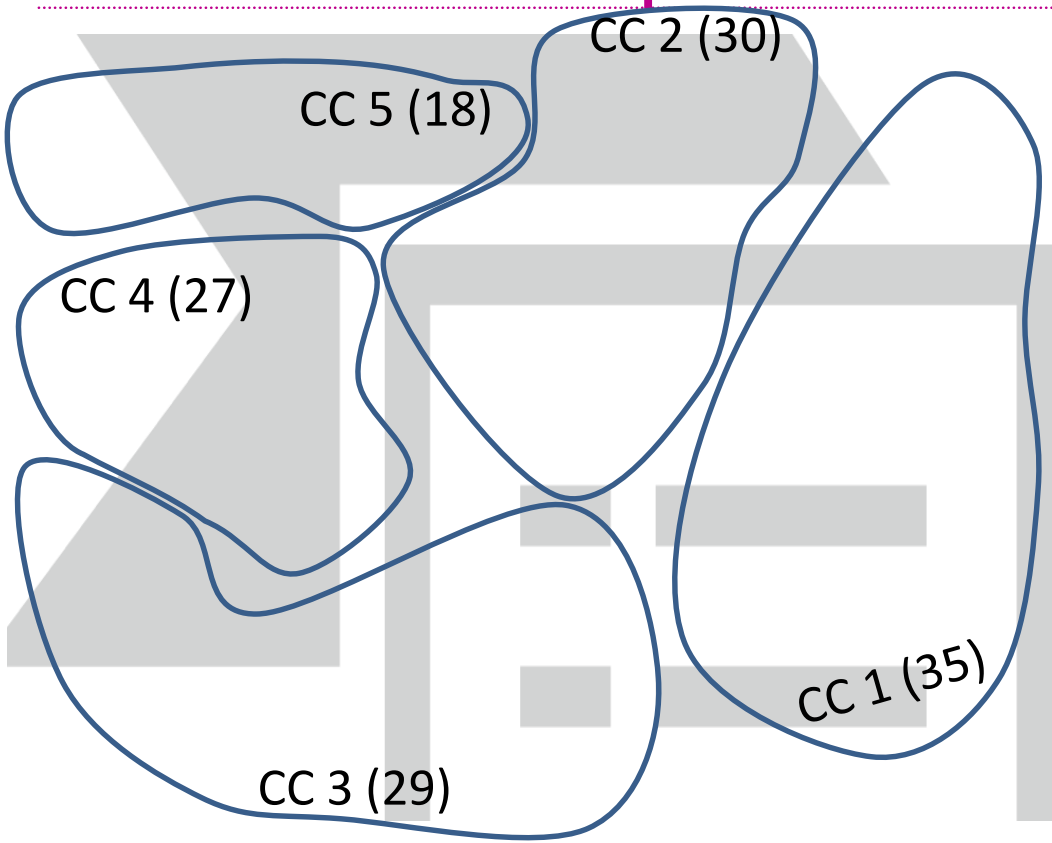
- Plus court chemin

- Hors présentation :
  - Similarité de Jaccard
  - Connectivité
  - Nombre de plus courts chemins
  - Dépendance sociale (social neighbors)

Caractériser un réseau

CLUSTERING

Composantes connexes



Composante connexe : ensemble (maximal) de nœuds avec un chemin entre chaque paire de nœuds

Composante géante *(giant component)* : la plus grande CC

Beaucoup d'analyses peuvent (doivent) se faire isolément sur chaque composante

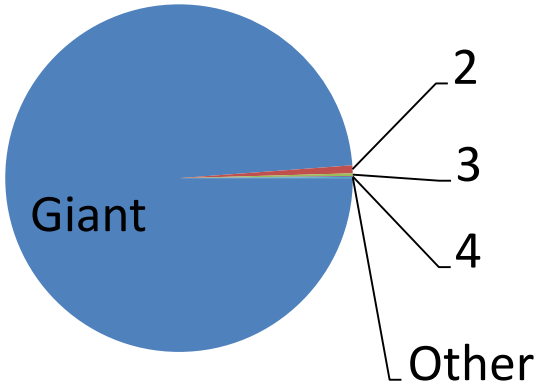
Composante connexe

Dimona (all)

	Taille de la CC	Nbre de CC
1	2	25 929
2	3	5 638
3	4	1 584
4	5	614
…	…	…
35	48	1
36	55	1
37	96	1
38	6 965 132	1

25 929 situations avec :

- Un employé sans aucun autre* employeur
- Dans une entreprise sans autre* employé



Giant component : 6,53 millions d'employés (99,3%) sont « collègues de (ex-)collègues de … » via 430K employeurs (92%)

* Dans la période considérée

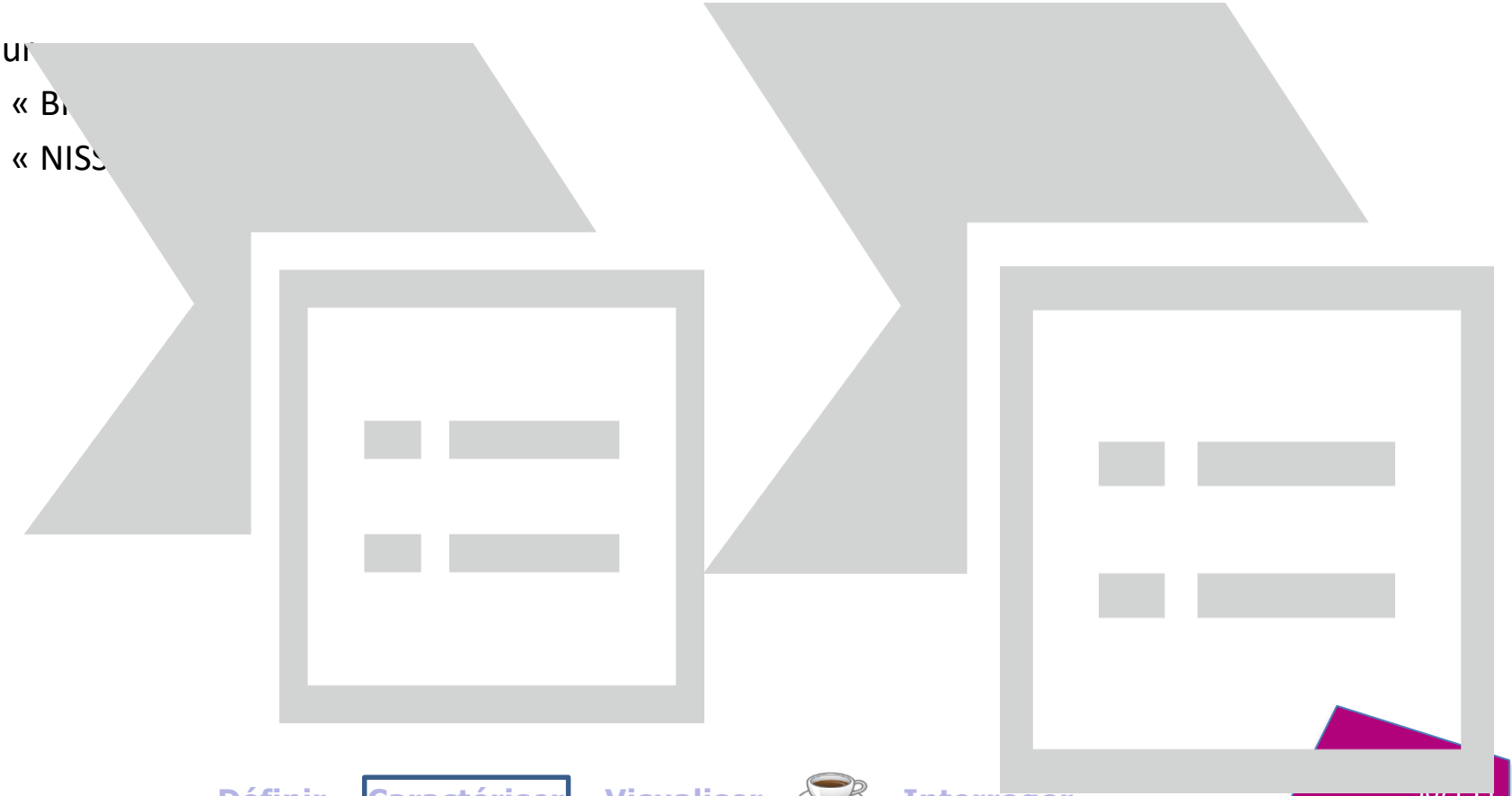
Composante géante

- Diamètre : 20
- Médiane : distance = 6

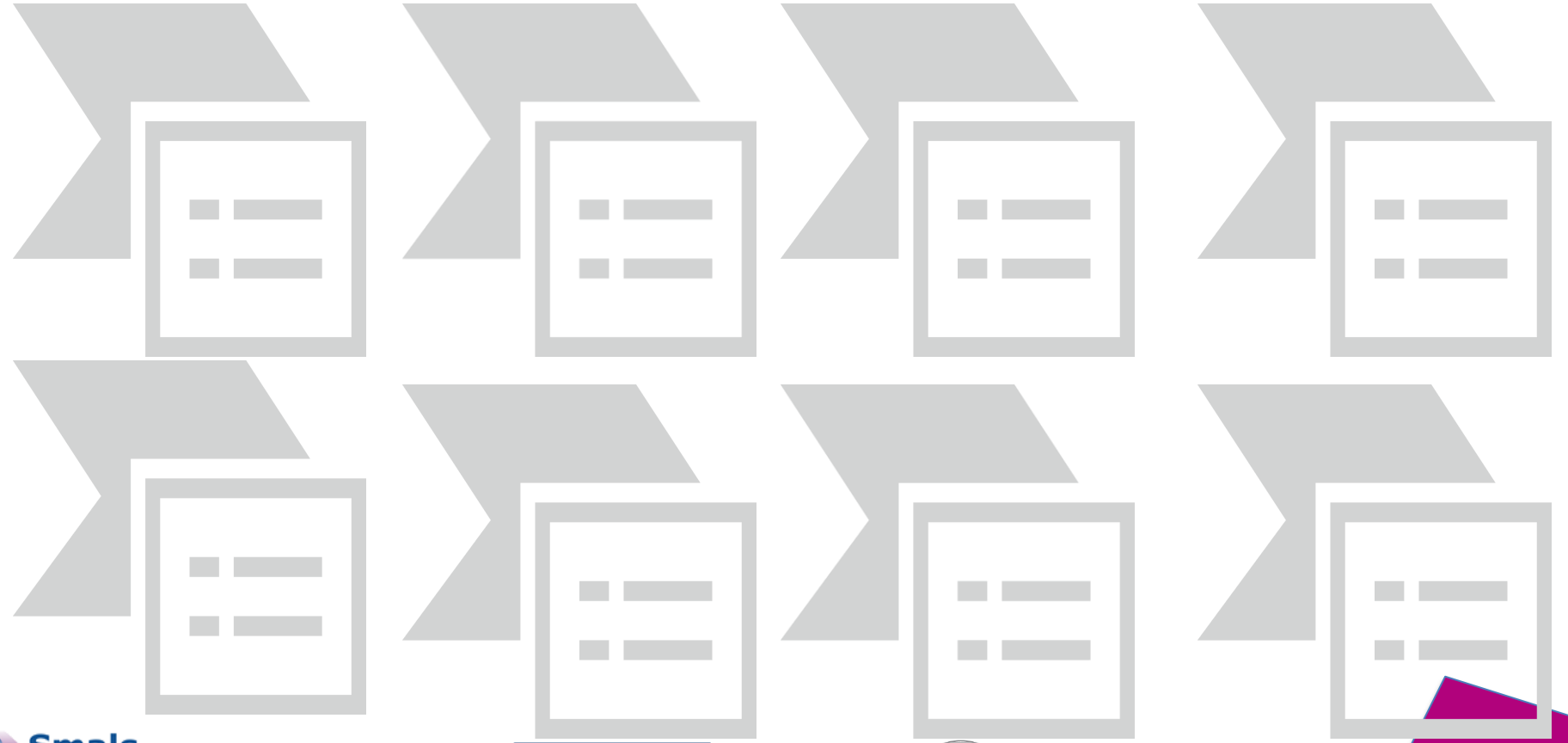
- → 50 % des « couples » sont « collègue de collègue de collègue »
- Percentile 95 : distance = 8 (+1 « de collègue »)
- Il y a des « super-connecteurs » : tous les enseignants sont collègues !

Composante connexe : sub-giants

- Employeur
- Employé « B
- Employé « NIS

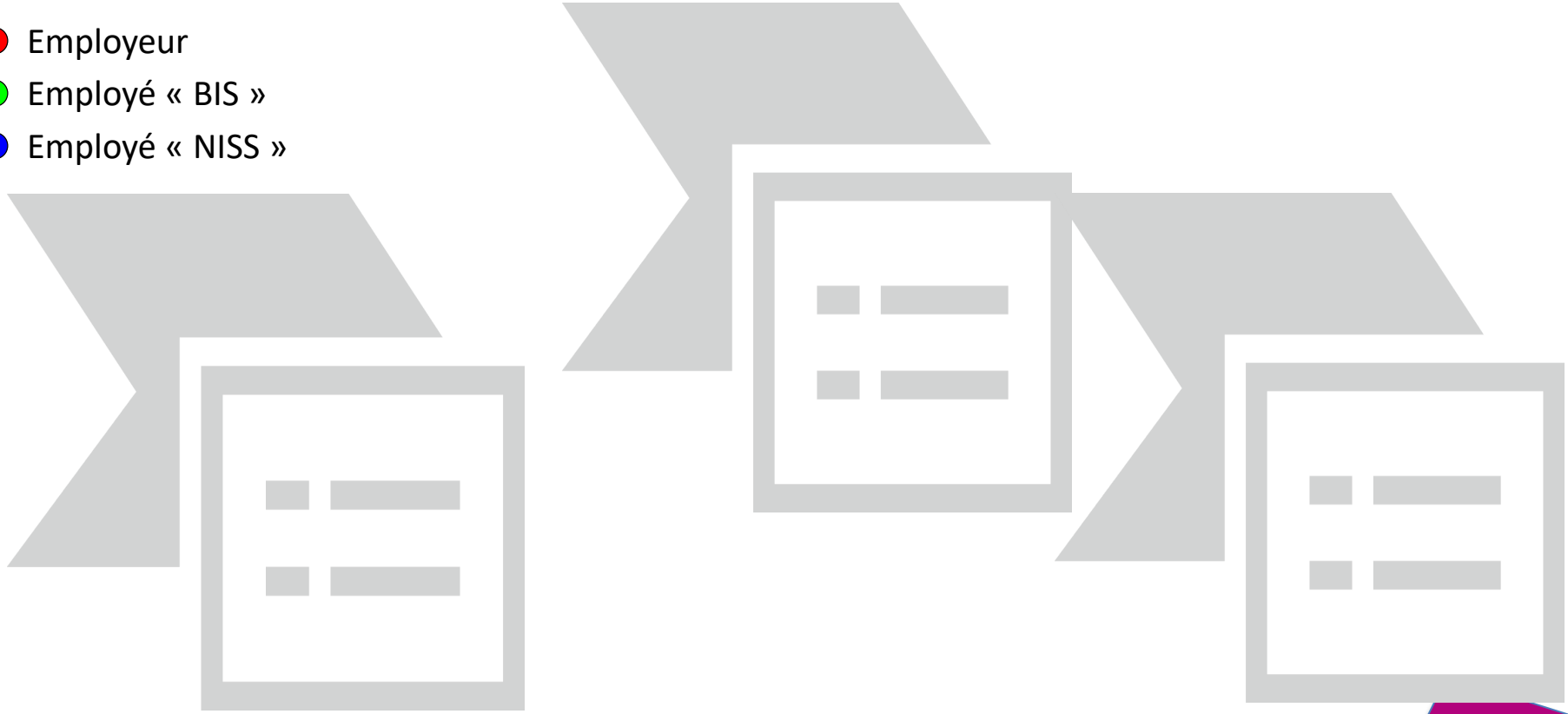


Composante connexe : sub-giants



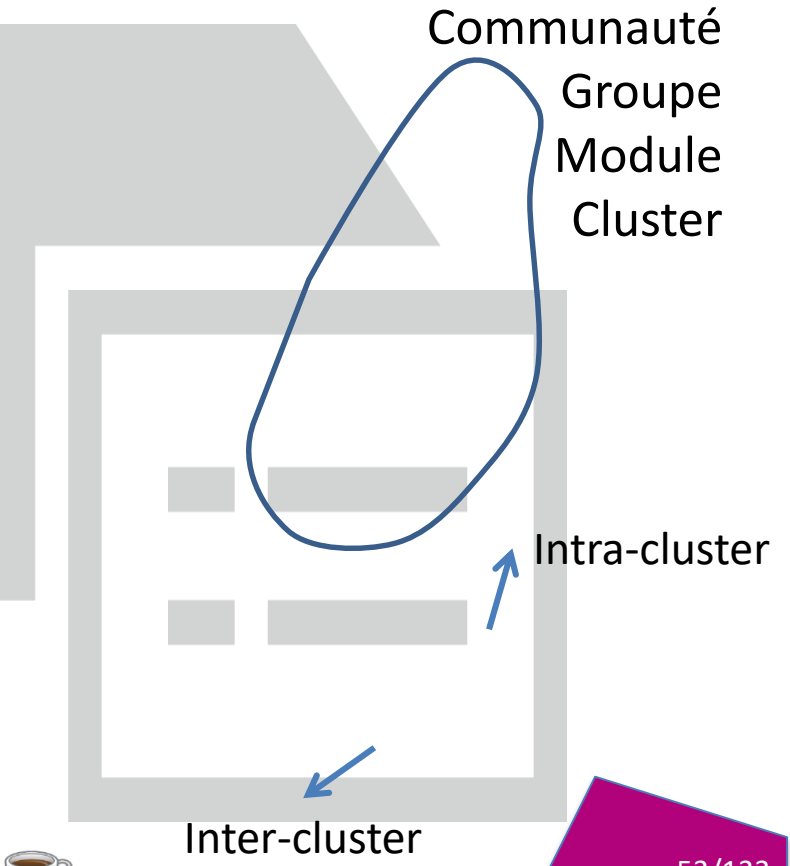
Composante connexe : sub-giants

- Employeur
- Employé « BIS »
- Employé « NISS »



Communautés

Coefficient de modularité :
- Mesure de la qualité d'une partition
- [-1, 1], liens inter-cluster vs liens intra-cluster
- Bonne partition (coef. ≈ 1):
  liens intra-cluster >> liens inter-cluster
- Optimisation = NP-hard
- Basé sur les liens, pas sur les attributs

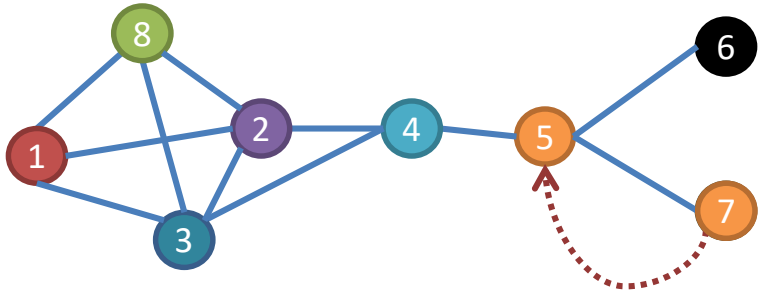


Communautés : intérêts

- Différentes communautés ont souvent des propriétés différentes de l'ensemble
- Permet l'analyse des communautés une par une
- Permet l'identification des « *missing links* » et des « *fake links* »
- Inférer des caractéristiques (attributs)
- Vue d'ensemble avec un « graphe de clusters » (+drill-down)

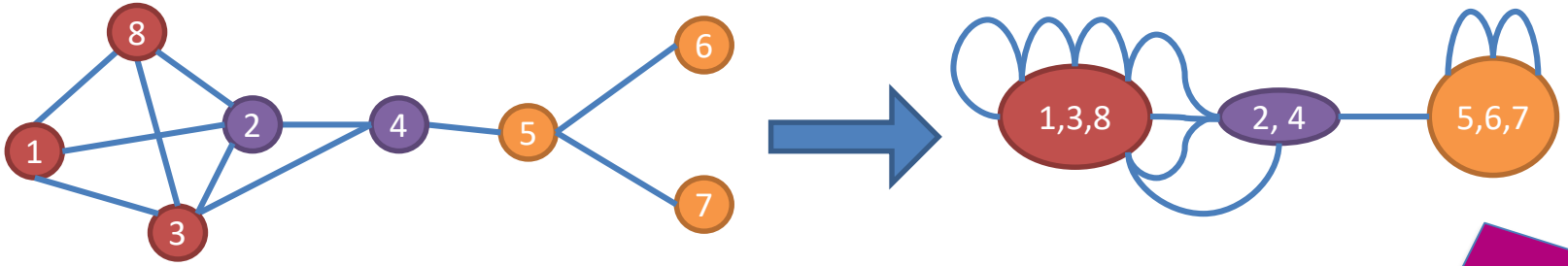
Algorithme de Louvain (UCL, 2008)

Phase 1



Améliore la « modularity »?
Si oui, migrer

Phase 2



Fast unfolding of communities in large networks, V. Blondel, J.-L. Guillaume, R. Lambiotte, E. Lefebvre, 2008

Clustering

- Composantes connexes : groupes « déconnectés »
- Communautés : groupes « sociaux »

Hors présentation :

- Cliques
- Notion de « fortement » ou « faiblement » connecté (réseaux dirigés)

Visualiser un réseau

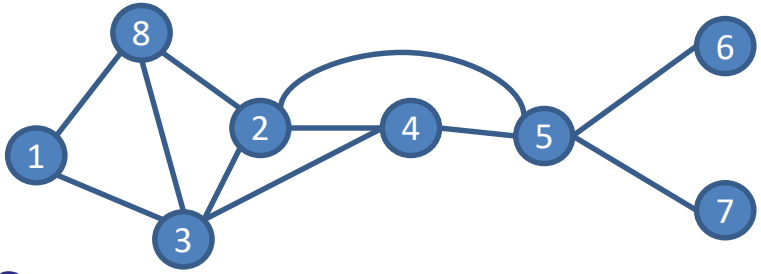
Visualiser

Visualiser un réseau

SPATIALISATION (LAYOUT)

Spatialisation

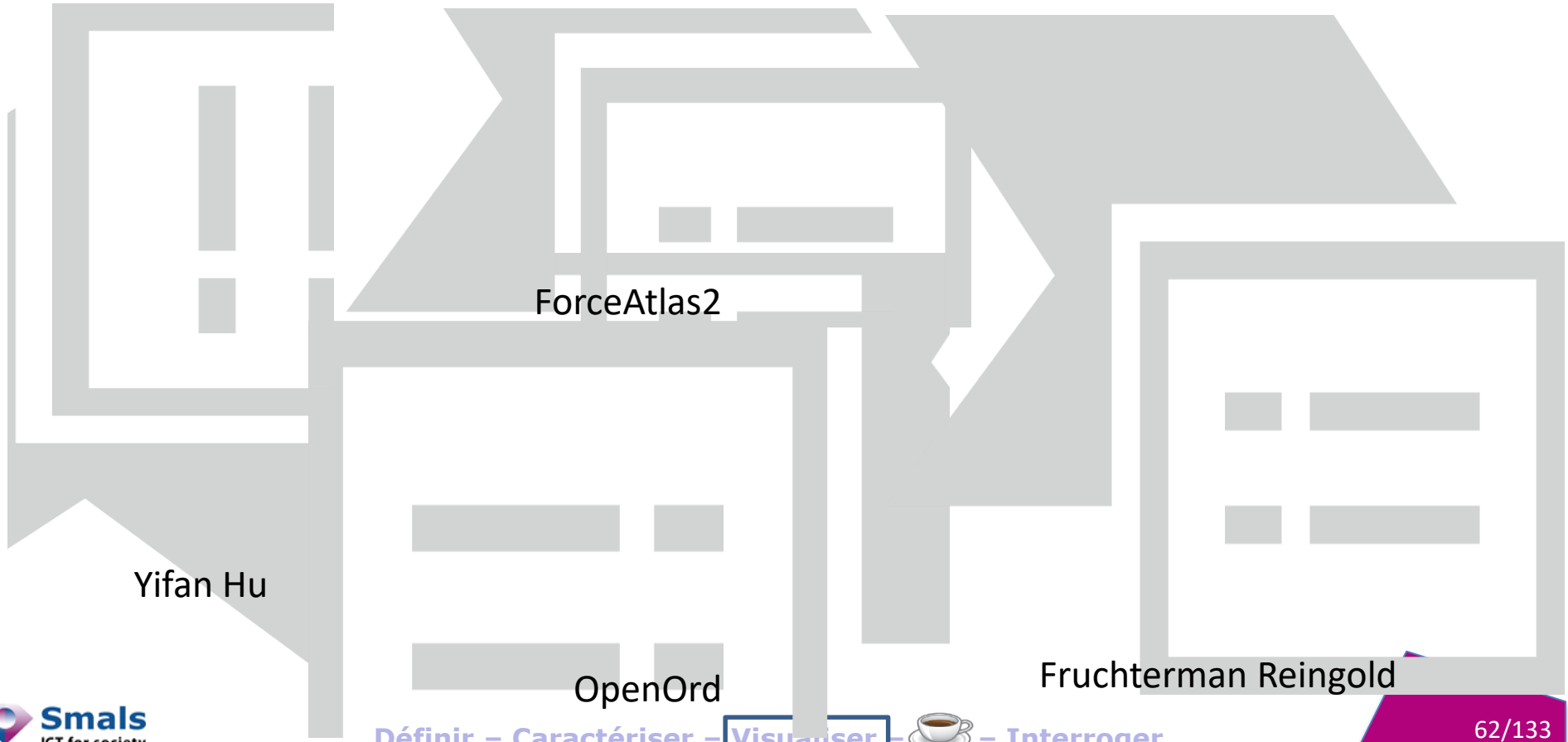
- Cas simple : graphe planaire = qui peut se dessiner sans croisement
- En général :
  - Minimiser les croisements
  - Rapprocher les nœuds connectés
  - Éloigner les nœuds non-connectés



Spatialisation

- Force-directed :
  - Force attractive (spring) entre deux nœuds liés
  - Force répulsive entre tous les nœuds
- Spectral : Utilisation des vecteurs/valeurs propres (*eigenvectors/values*)
- Autres : Hiérarchiques, circulaires, linéaires…

Spatialisation



Visualiser un réseau

SIMPLIFICATION

Simplification

- Au-delà de quelques centaines de nœuds, visualiser un graphe est illusoire
- Plusieurs méthodes permettent une approche
  - Découper le graphe
  - Supprimer des nœuds
  - Fusionner des nœuds

Découper le graphe

- Composantes connexes
  → Méthode naturelle, pas de perte

- Communautés
  → On perd les relations inter-communauté

Filtrer des nœuds

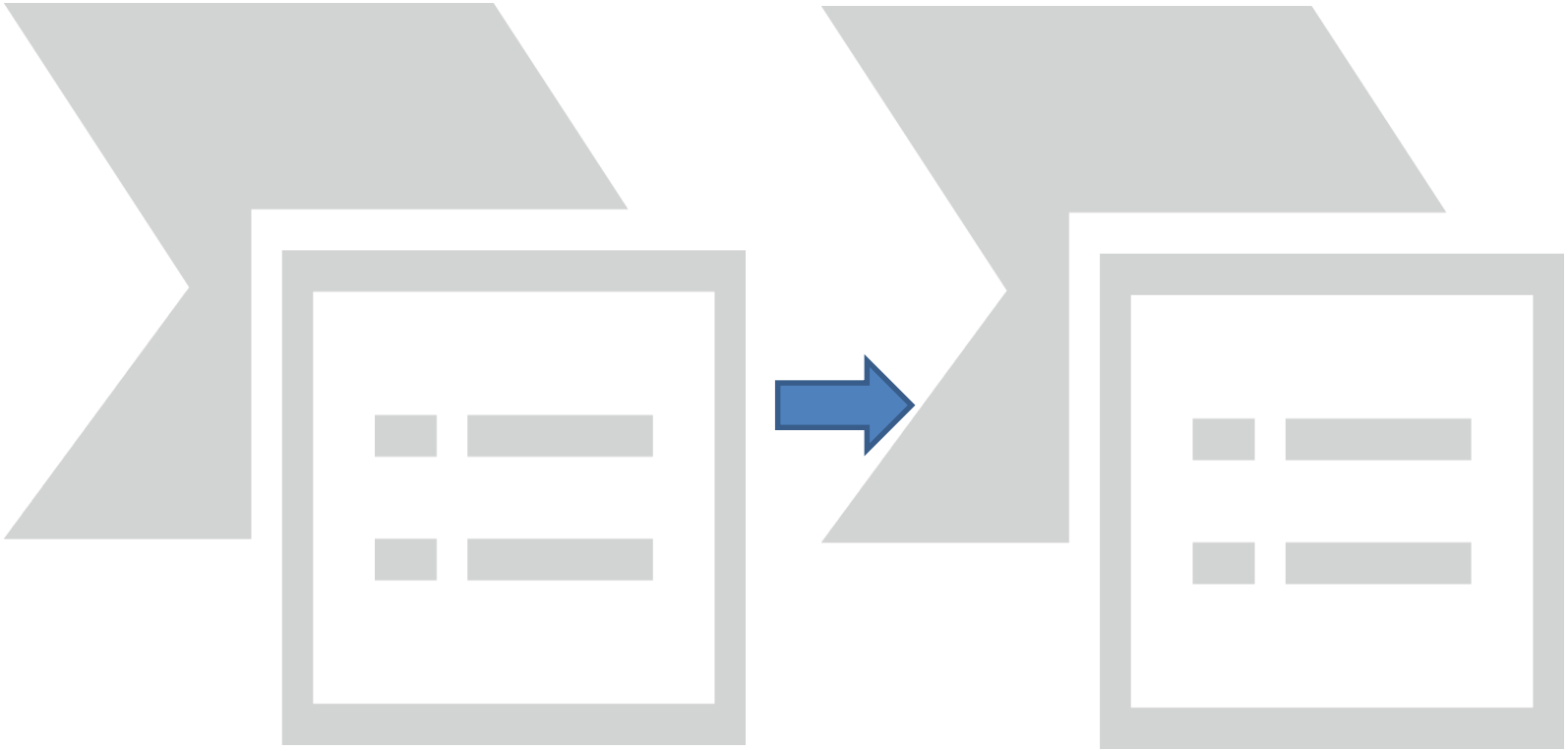
- Supprimer les super-connecteurs (degré élevé)
  → augmente le nombre de composantes connexes

- Supprimer les feuilles (degré/K-core faible)
  → focus sur le « backbone/core »

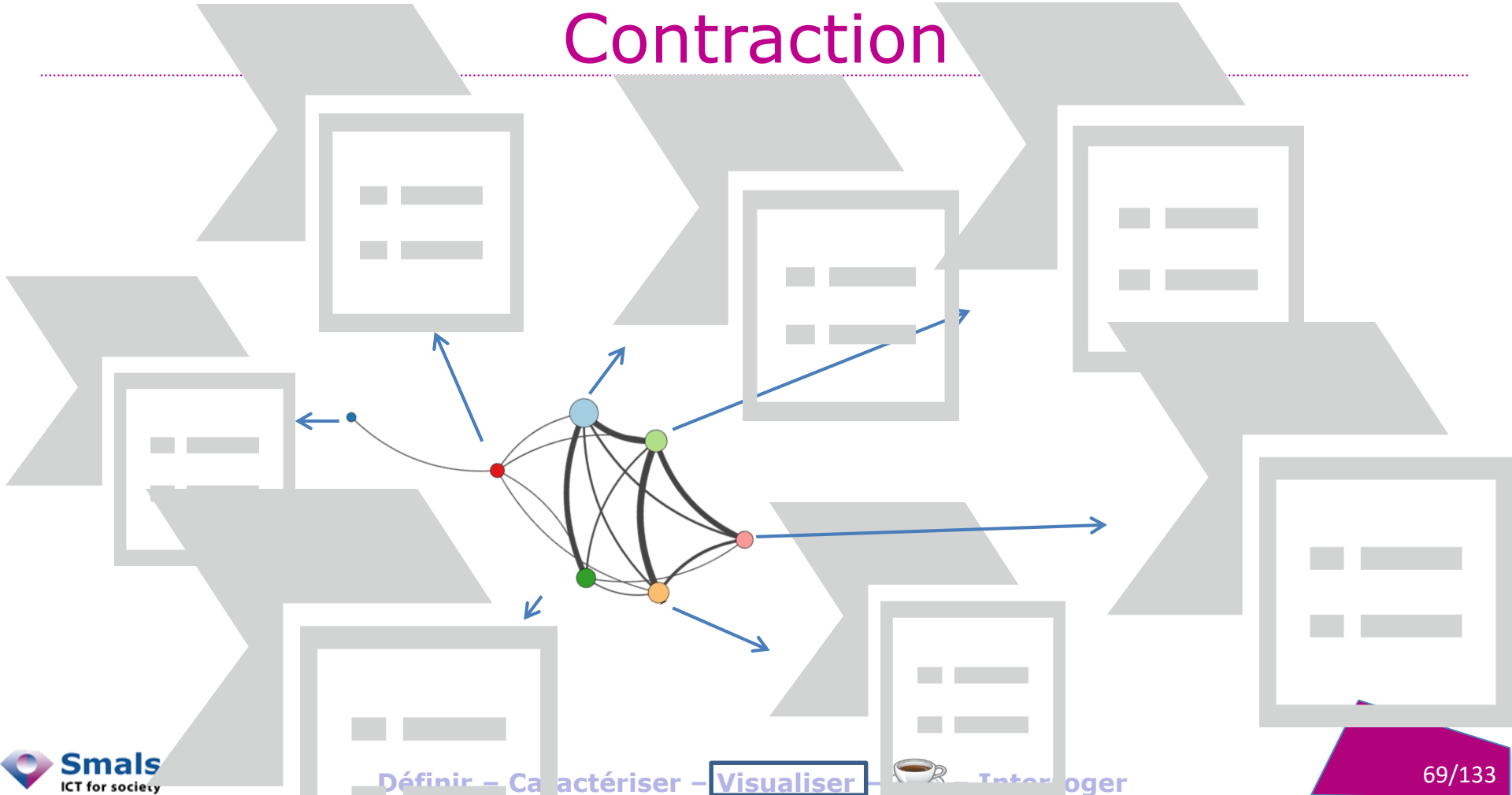
Fusionner des nœuds

- Contraction : fusionner des groupes
- Projection bipartite : conversion d'un graphe biparti en 2 graphes unipartis.

Contraction



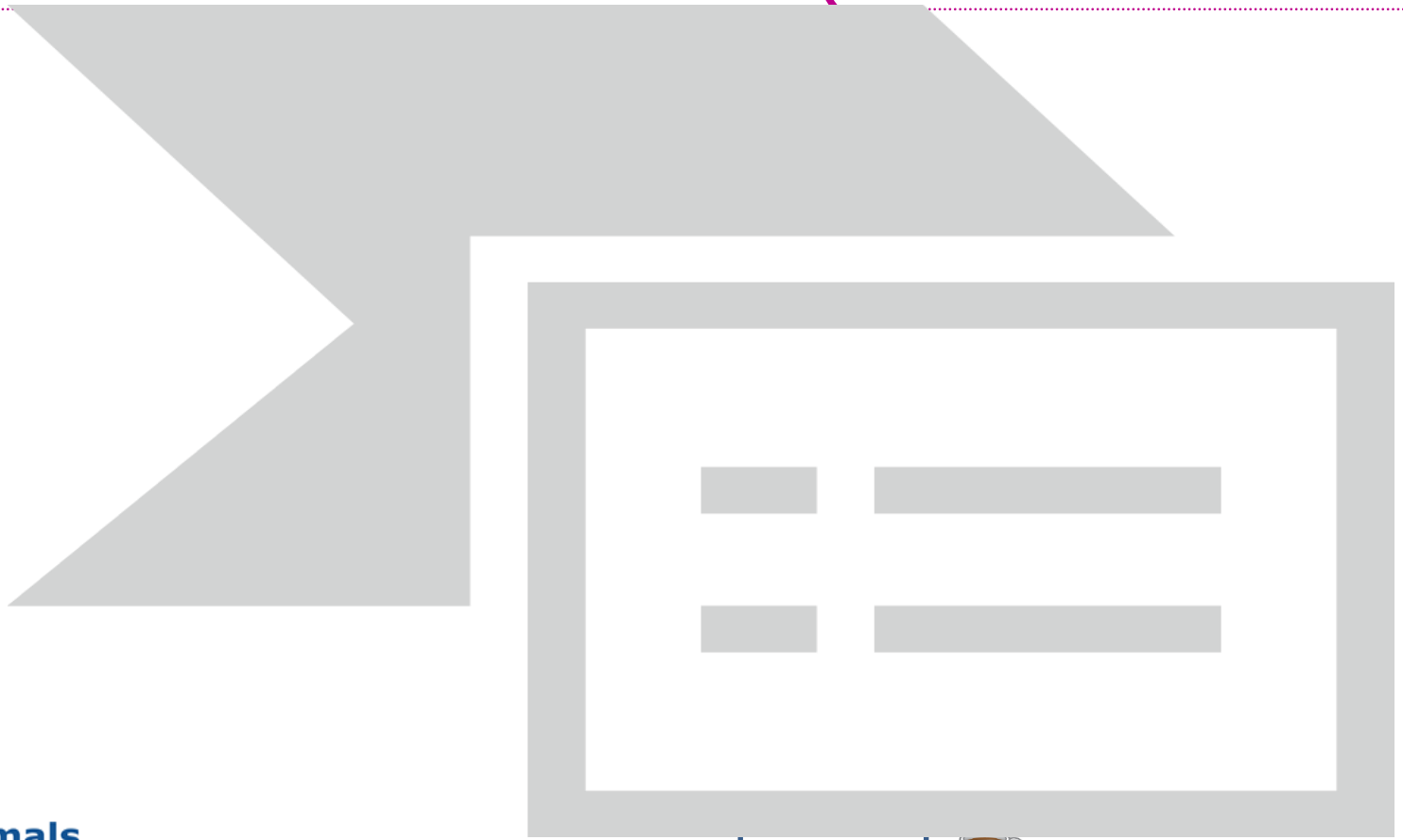
Contraction



Contraction : BCE (Administrateurs)

- Administrateurs BCE : Composante géante = 655.158 nœuds (sur ~10^6)
- On applique Louvain : 743 communautés (~10” en mono-thread)

Contraction : BCE (Administrateurs)

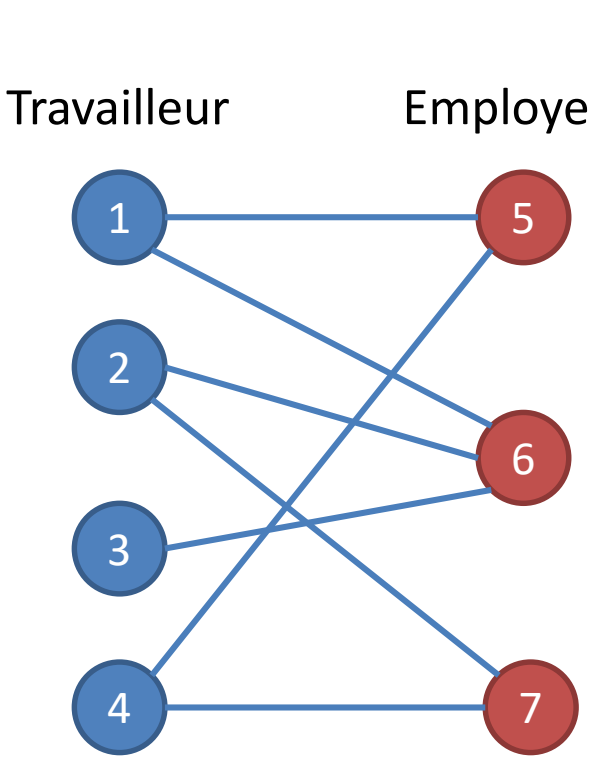


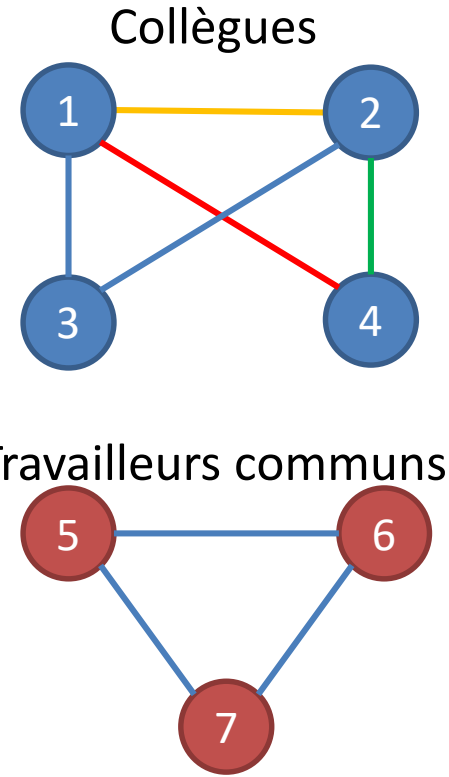
Tree
Star
Other

Outgoing edges:

[10000,inf]
[1000,9999]
[100,999]
[50,99]
[10,49]
[5,9]
4
3
2
1

Projection bipartite

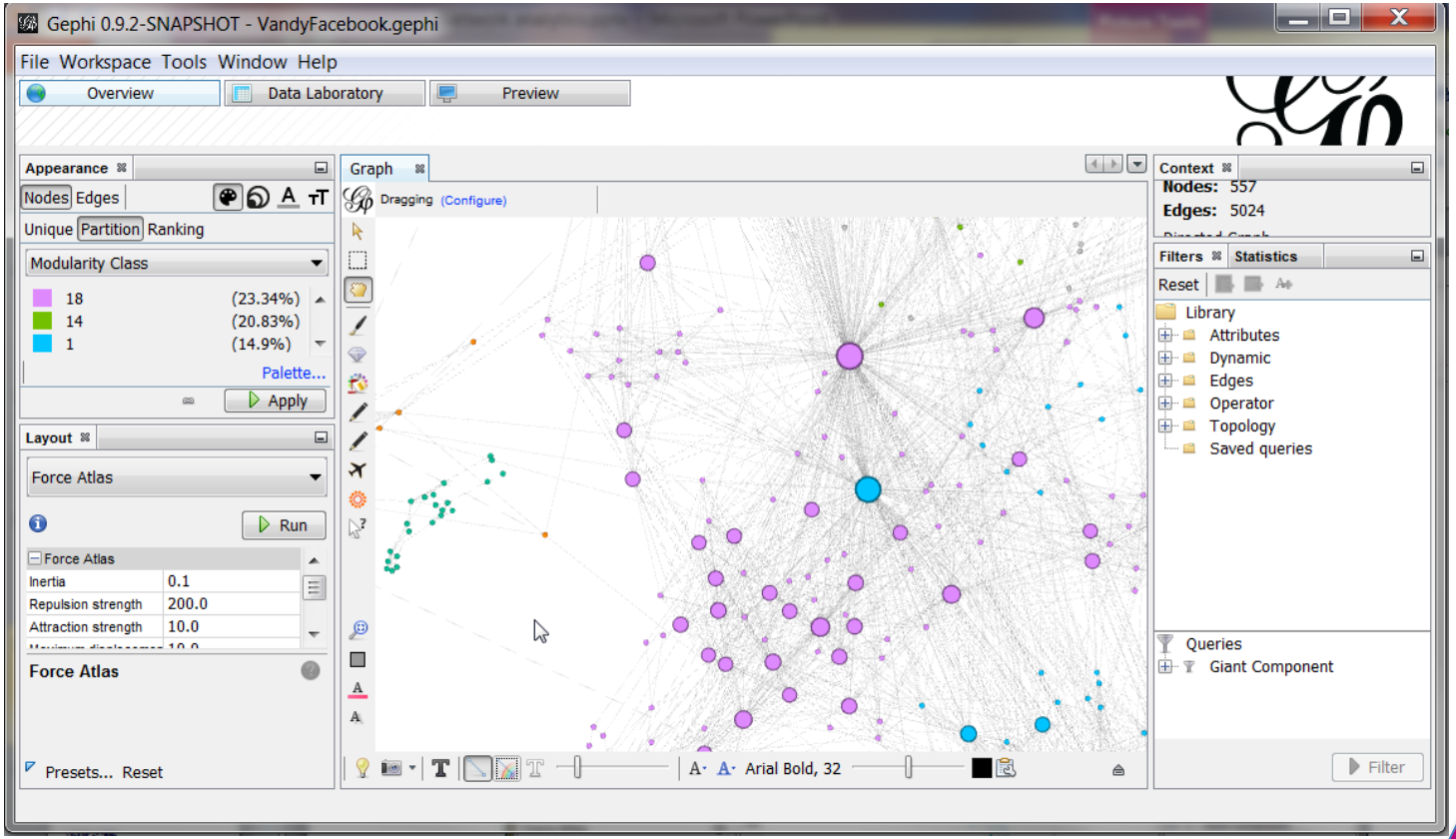




Visualiser un réseau

OUTIL DESKTOP : GEPHI

Gephi


Gephi 0.9.2-SNAPSHOT - VandyFacebook.gephi
File Workspace Tools Window Help
Overview
Data Laboratory
Preview
Appearance
Nodes Edges
Unique Partition Ranking
Modularity Class
18 (23.34%)
14 (20.83%)
1 (14.9%)
Palette...
Apply
Layout
Force Atlas
Run
Force Atlas
Inertia 0.1
Repulsion strength 200.0
Attraction strength 10.0
Force Atlas
Presets... Reset
Graph
Dragging (Configure)
Arial Bold, 32
Context
Nodes: 557
Edges: 5024
Filters
Statistics
Reset
Library
Attributes
Dynamic
Edges
Operator
Topology
Saved queries
Queries
Giant Component
Filter

Visualiser un réseau

LIBRAIRIE JS: VIS.JS

vis.js

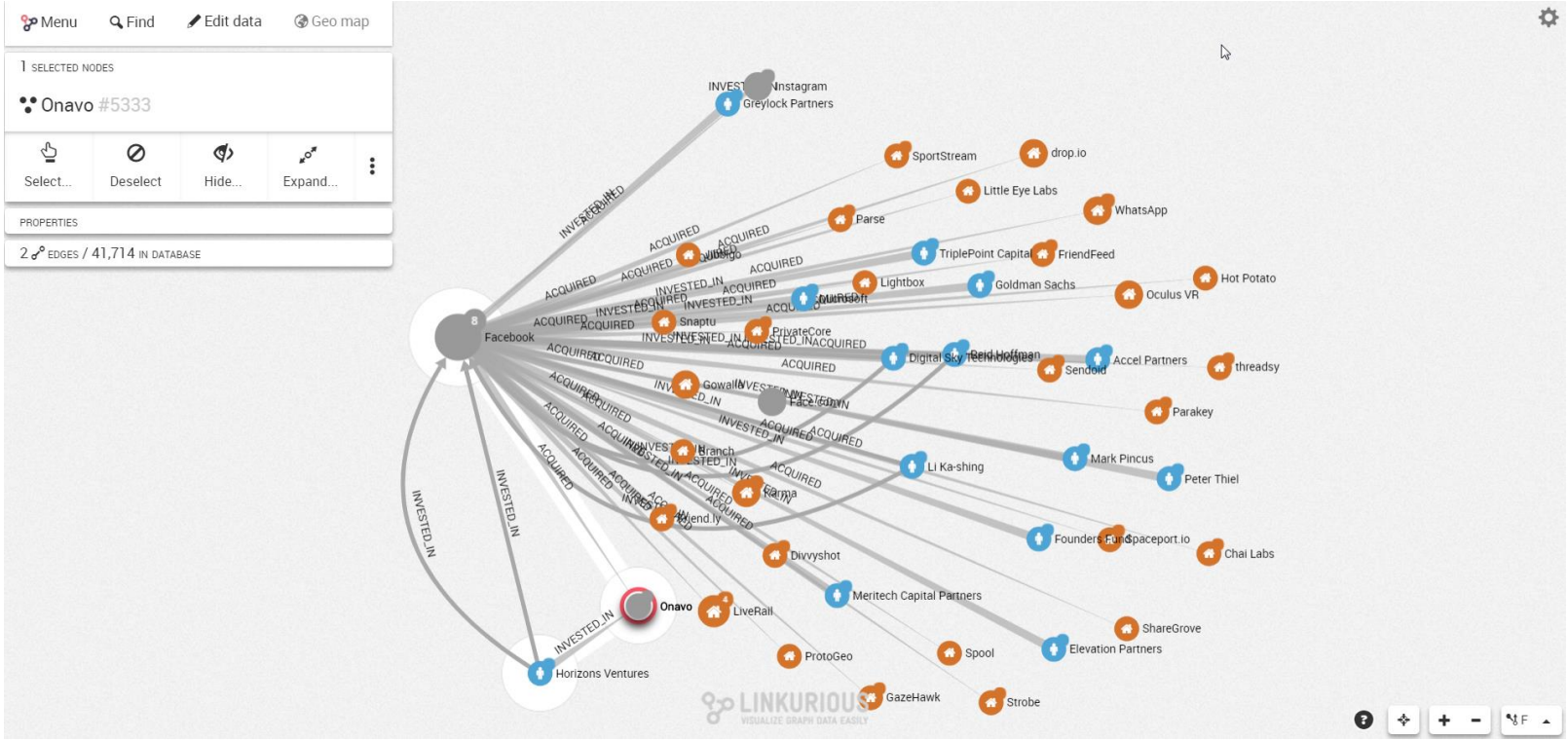
```
<div id="graph"></div>
<script type="text/javascript">
var nodes = new vis.DataSet([
    {id: 1, label: 'Node 1'},
    {id: 2, label: 'Node 2'},
    {id: 3, label: 'Node 3'},
]);
var edges = new vis.DataSet([
    {from: 1, to: 3},
    {from: 1, to: 2},
    {from: 3, to: 3}
]);
var container =
document.getElementById('graph');
var data = {
  nodes: nodes,
  edges: edges
};
var options = {};
var network =
  new vis.Network(container,
                  data,
                  options);
</script>
```

http://visjs.org/examples/network/basicUsage.html

Visualiser un réseau

OUTIL WEB : LINKURIOUS

Linkurio.us


Menu
Find
Edit data
Geo map
1 SELECTED NODES
Onavo #5333
Select...
Deselect
Hide...
Expand...
PROPERTIES
2 EDGES / 41,714 IN DATABASE
Facebook
Instagram
Greylock Partners
SportStream
drop.io
Little Eye Labs
Parse
WhatsApp
TriplePoint Capital
FriendFeed
Lightbox
Goldman Sachs
Hot Potato
Oculus VR
Snaptu
PrivateCore
Digital Sky Technologies
Reid Hoffman
Accel Partners
Sendoid
threadsy
Parakey
Gowalla
Branch
Mark Pincus
Li Ka-shing
Peter Thiel
Qwend.ly
Founders Fund
Spaceport.io
Chai Labs
Divvyshot
Meritech Capital Partners
Onavo
LiveRail
ShareGrove
Elevation Partners
ProtoGeo
Spool
Horizons Ventures
GazeHawk
Strobe
ACQUIRED
INVESTED_IN
LINKURIOUS
VISUALIZE GRAPH DATA EASILY

Visualiser un réseau

AUTRES OUTILS

Outils desktop

- Gephi https://gephi.org OS
- TouchGraph http://www.touchgraph.com €
- Cytoscape http://www.cytoscape.org OS
- Maltego https://www.paterva.com
- NodeXL https://nodexl.codeplex.com OS

- SAS Network Analytics €
- IBM i2 €

Outils Cloud

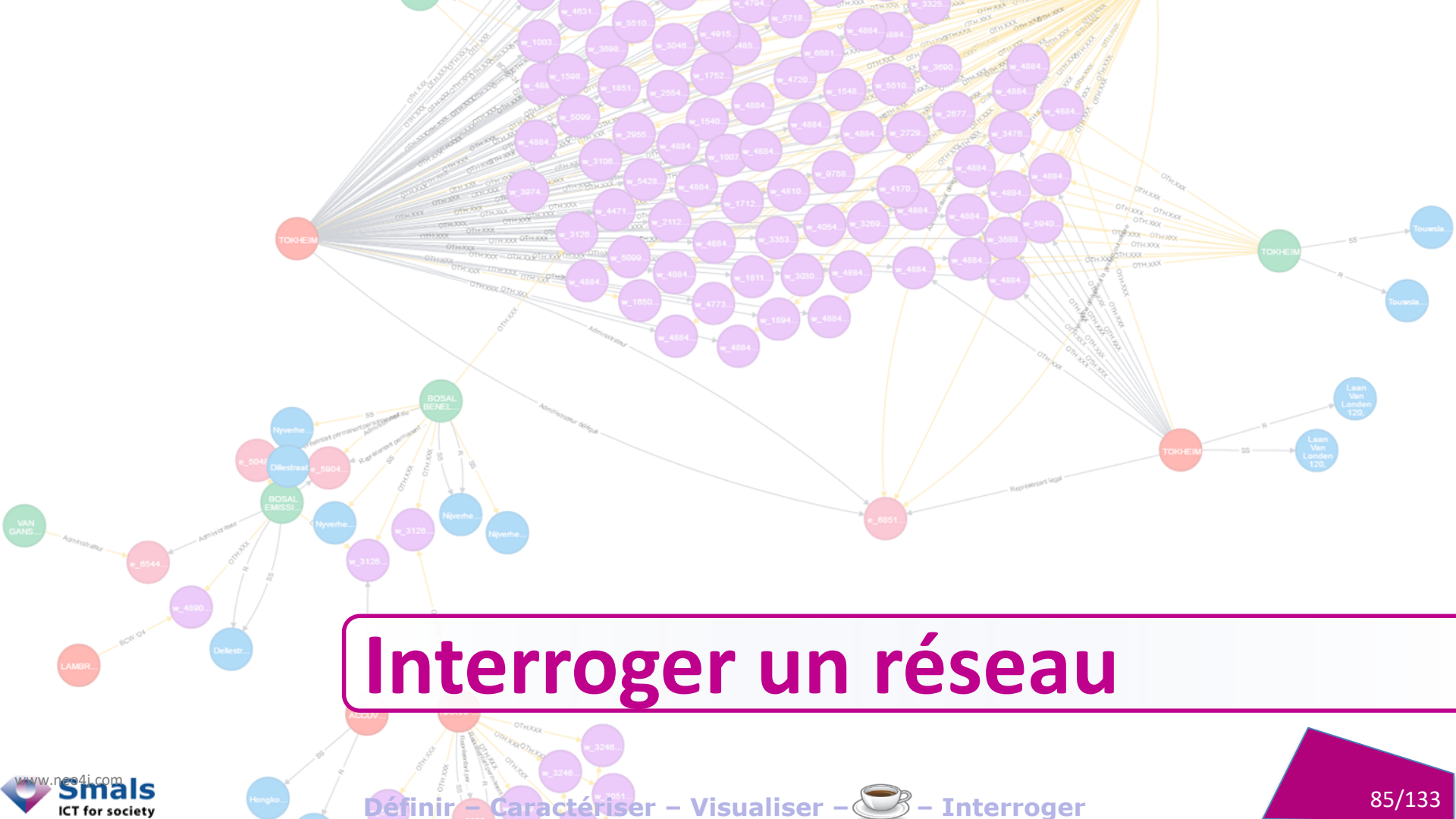
- Linkurio.us https://linkurio.us €
- Keylines https://cambridge-intelligence.com/keylines €

Librairies web

- Vis.js http://visjs.org OS
- Cytoscape.js http://js.cytoscape.org OS
- Sigma.js http://sigmajs.org OS
- D3.js https://d3js.org OS
- OGMA (Linkurio.us SDK) €

Questions ?

Pause !

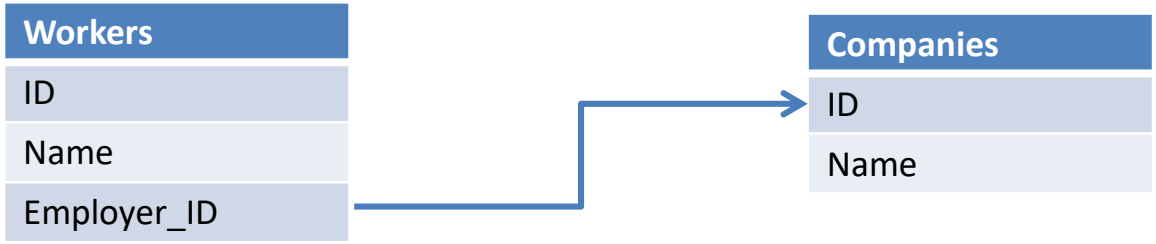


Interroger un réseau

Interroger un réseau

GRAPH DATABASES

RDBMS et relations



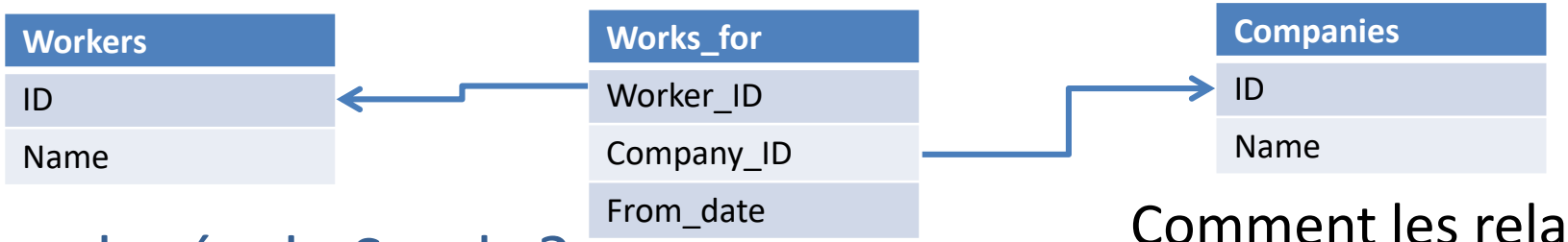
Comment les relations sont implémentées

Employés de Smals ?

Ce qui intéresse le développeur

```
SELECT  Workers.Name
FROM    Workers
JOIN    Companies
  ON    Workers.Employer_ID = Companies.ID
WHERE   Companies.Name = 'Smals'
```

RDBMS et relations



Employés de Smals ?

Comment les relations sont implémentées

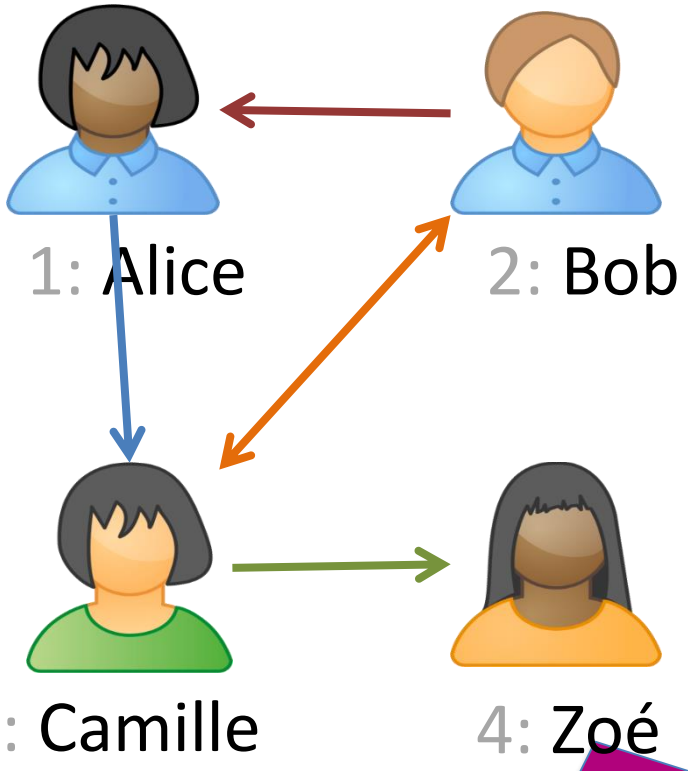
Ce qui intéresse le développeur

```
SELECT  Workers.Name
FROM    Workers
JOIN    Works_for
  ON    Workers.ID = Works_for.Worker_ID
JOIN    Companies
  ON    Works_for.Company_ID = Companies.ID
WHERE   Companies.Name = 'Smals'
```

RDBMS et relations

ID	Name
1	Alice
2	Bob
3	Camille
4	Zoé

Liker_ID	Liked_ID
1	3
2	1
2	3
3	2
3	4




```
SELECT  p1.Name
FROM    People p1
JOIN    Likes
  ON    Likes.Liked_ID = p1.ID
JOIN    People p2
  ON    Likes.Liker_ID = p2.ID
WHERE   p2.Name = "Bob"
```

Qui Bob aime ?

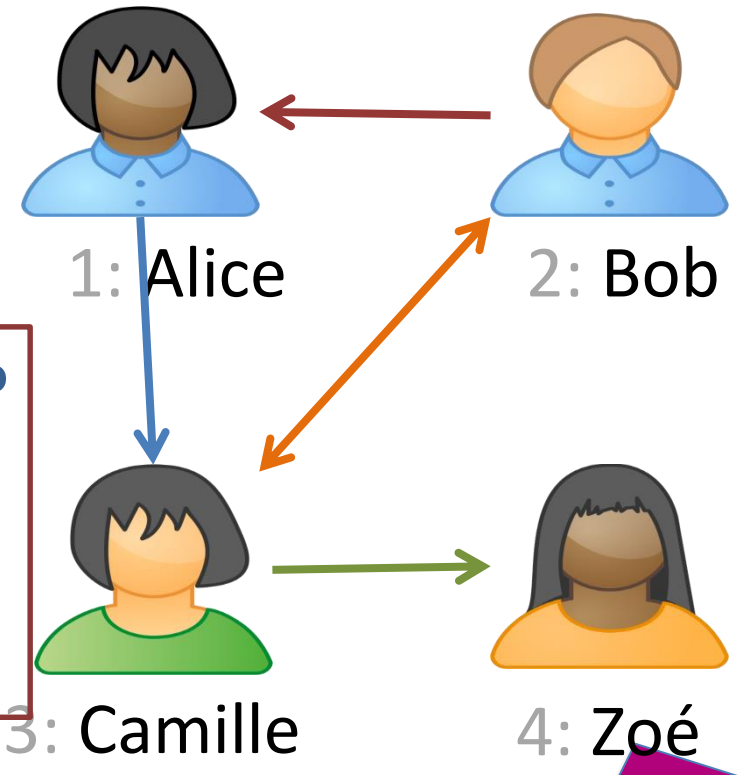
RDBMS et relations

ID	Name
1	Alice
2	Bob
3	Camille
4	Zoé

Liker_ID	Liked_ID
1	3
2	1
2	3
3	2
3	4

```
SELECT  p1.Name
FROM    People p1
JOIN    Likes l1
   ON   l1.Liked_ID = p1.ID
JOIN    Likes l2
   ON   l1.Liker_ID = l2.Liked_ID
JOIN    People p2
   ON   l2.Liker_ID = p2.ID
WHERE   p2.Name = "Bob"
```

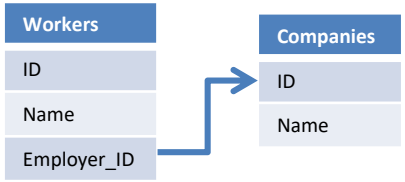
Qui Bob (aime)² ?



RDBMS et relations : limitations

Relations 1-n

Foreign key:

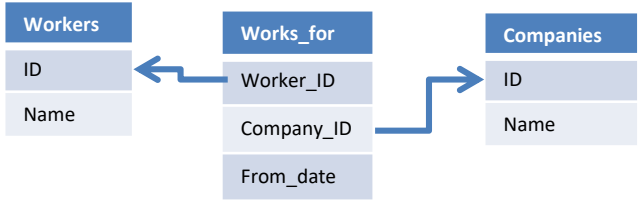


- Détourne le rôle d'un attribut
- Pas de type « Key »

Relations m-n

(ou 1-n avec attributs)

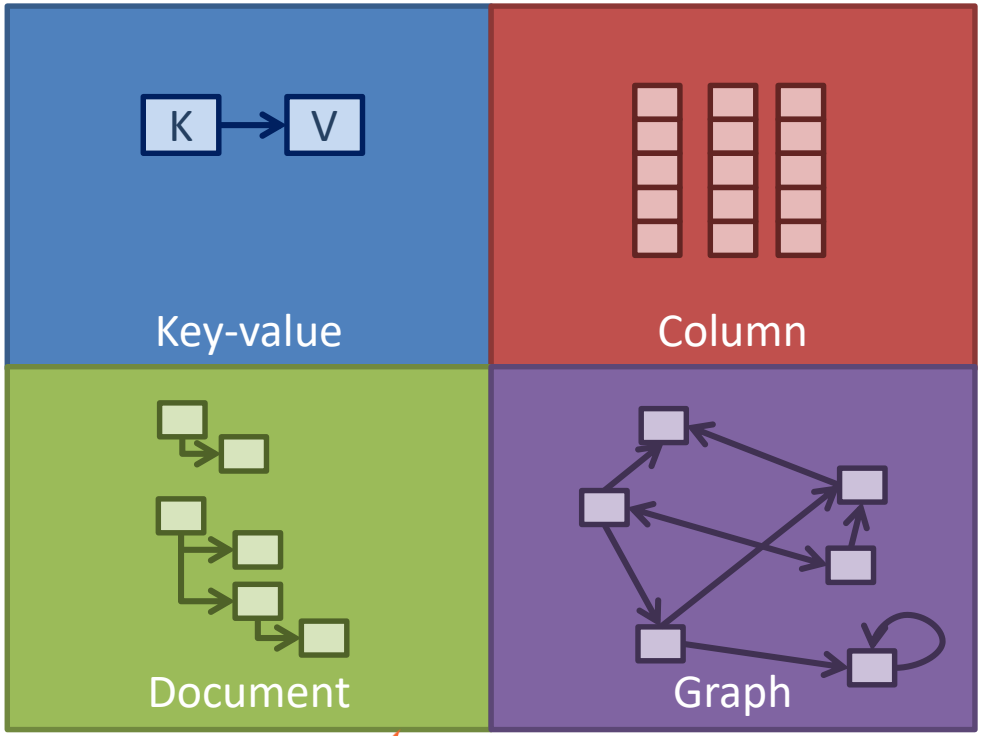
Join table:



- Détourne le rôle d'une table
- Pas distinguable des autres

- Contrainte d'intégrité possible (externe à la table)
- Structure de la relation à préciser à chaque requête
- Join : Lourd en écriture et en exécution

Solutions NoSQL



GraphDB : objectifs

Objectifs des *bases de données orientées graphes* :

Querying language qui sépare la définition **des relations** (à la création) et la recherche de « **matching** »

Moteur efficace pour le parcours des **relations**

```
SELECT p1.Name
FROM   People p1
JOIN   Likes l1
   ON  l1.Liked_ID = p1.ID
JOIN   Likes l2
   ON  l1.Liker_ID = l2.Liked_ID
JOIN   People p2
   ON  l2.Liker_ID = p2.ID
WHERE  p2.Name = "Bob"
```

Cypher (Neo4j)

```
MATCH
   (:People {Name:"Bob"})
   -[:Likes*2]->
   (p:People)
RETURN p.Name
```

GraphDB: moins de code

```
 (SELECT T.directReportees AS directReportees, sum(T.count) AS count
FROM (
SELECT manager.pid AS directReportees, 0 AS count
  FROM person_reportee manager
  WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
UNION
  SELECT manager.pid AS directReportees, count(manager.directly_manages) AS count
FROM person_reportee manager
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
GROUP BY directReportees
UNION
SELECT manager.pid AS directReportees, count(reportee.directly_manages) AS count
FROM person_reportee manager
JOIN person_reportee reportee
ON manager.directly_manages = reportee.pid
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
GROUP BY directReportees
UNION
SELECT manager.pid AS directReportees, count(L2Reportees.directly_manages) AS count
FROM person_reportee manager
JOIN person_reportee L1Reportees
ON manager.directly_manages = L1Reportees.pid
JOIN person_reportee L2Reportees
ON L1Reportees.directly_manages = L2Reportees.pid
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
GROUP BY directReportees
) AS T
GROUP BY directReportees)
UNION
(SELECT T.directReportees AS directReportees, sum(T.count) AS count
FROM (
SELECT manager.directly_manages AS directReportees, 0 AS count
FROM person_reportee manager
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
UNION
SELECT reportee.pid AS directReportees, count(reportee.directly_manages) AS count
FROM person_reportee manager
JOIN person_reportee reportee
ON manager.directly_manages = reportee.pid
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
GROUP BY directReportees
UNION
```

```
SELECT depth1Reportees.pid AS directReportees,
count(depth2Reportees.directly_manages) AS count
FROM person_reportee manager
JOIN person_reportee L1Reportees
ON manager.directly_manages = L1Reportees.pid
JOIN person_reportee L2Reportees
ON L1Reportees.directly_manages = L2Reportees.pid
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
GROUP BY directReportees
) AS T
GROUP BY directReportees)
UNION
(SELECT T.directReportees AS directReportees, sum(T.count) AS count
  FROM(
  SELECT reportee.directly_manages AS directReportees, 0 AS count
FROM person_reportee manager
JOIN person_reportee reportee
ON manager.directly_manages = reportee.pid
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
GROUP BY directReportees
UNION
SELECT L2Reportees.pid AS directReportees, count(L2Reportees.directly_manages)
AS count
FROM person_reportee manager
JOIN person_reportee L1Reportees
```

```
MATCH
   (boss)-[:MANAGES*0..3]->(sub),
   (sub)-[:MANAGES*1..3]->(report)
WHERE boss.name = « John Doe"
RETURN sub.name AS Subordinate,
       count(report) AS Total
```

GraphDB : moins de code

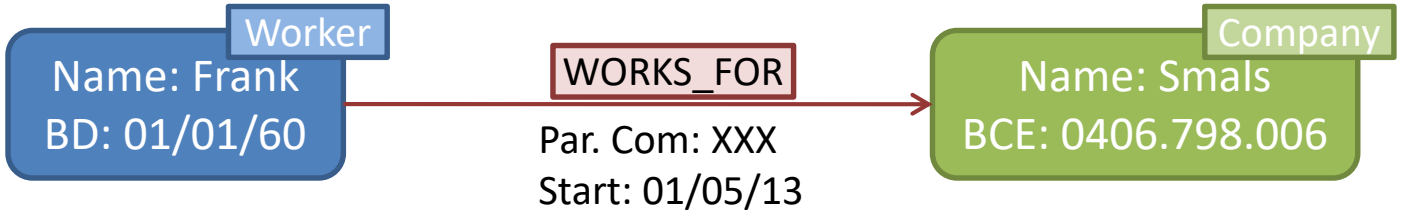
"We found Neo4j to be literally thousands of times faster than our prior MySQL solution, with queries that require 10-100 times less code. Today, Neo4j provides eBay with functionality that was previously impossible."

https://neo4j.com/news/graph-databases-provide-crystal-ball-online-recommendations-wal-mart-ebay/

Interroger un réseau

NEO4J

Requête Neo4j



Nœud de type « Worker »

Relation de type « WORKS_FOR »

Nœud de type « Company »

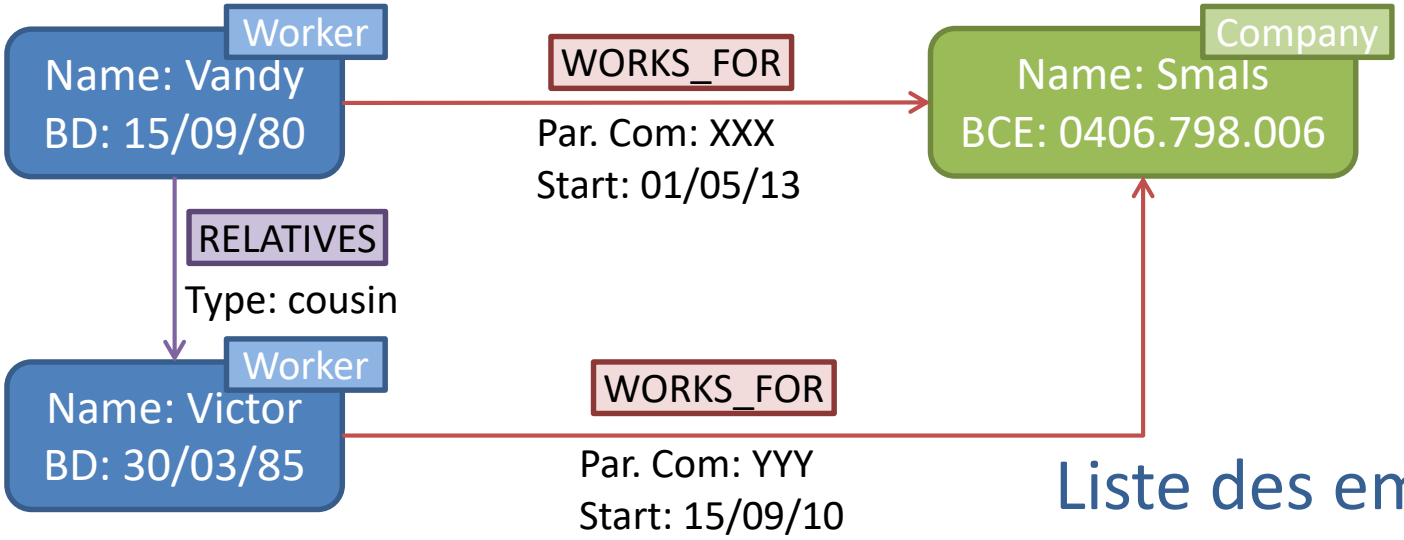
```
MATCH   (w:Worker)  -[r:WORKS_FOR]->  (c:Company)
WHERE   c.Name = "Smals"
RETURN  w.Name, w.BD, r.Start
```

Variante :

```
MATCH
  (w:Worker)
  -[r:WORKS_FOR]->
  (c:Company {Name:"Smals"})
RETURN …
```

Liste des employés de Smals

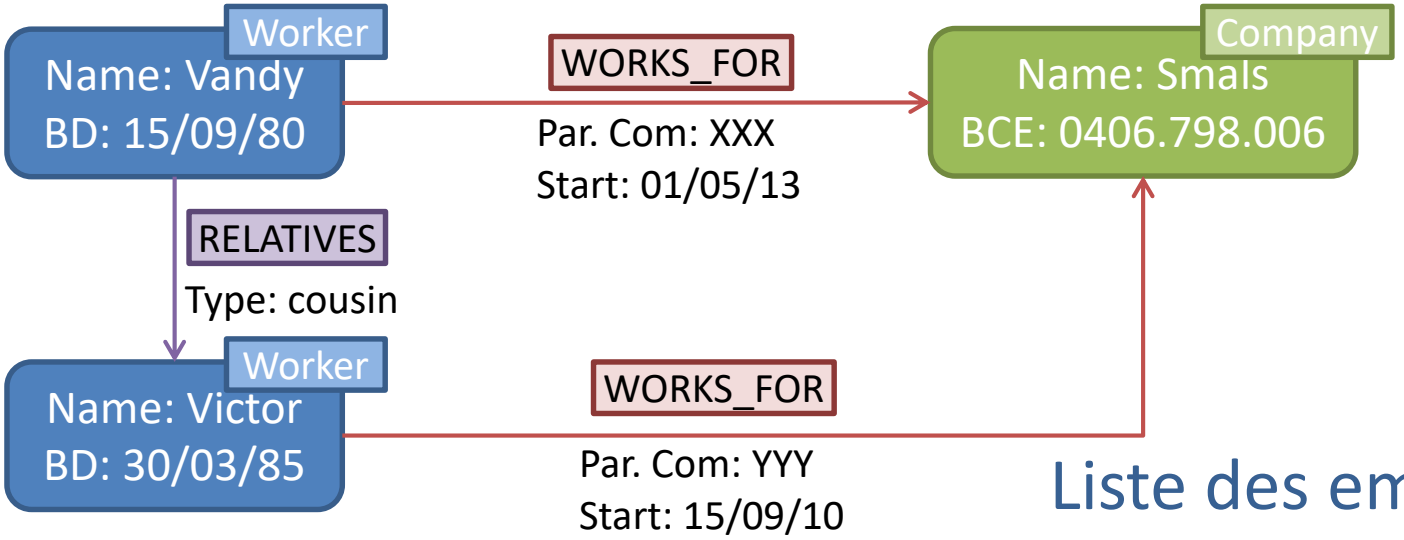
Requête Neo4j



Liste des employés de Smals apparentés

```
MATCH
    (w1:Worker)-[:WORKS_FOR]->(c:Company {Name:"Smals"}),
    (w2:Worker)-[:WORKS_FOR]->(c),
    (w1)-[:RELATIVES]->(w2)
RETURN w1.Name, w2.Name
```

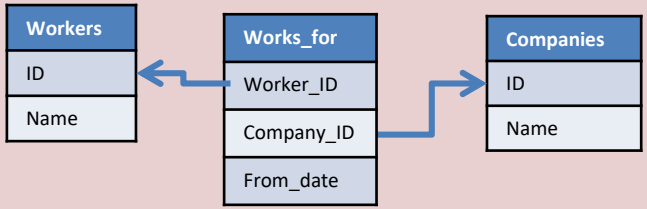
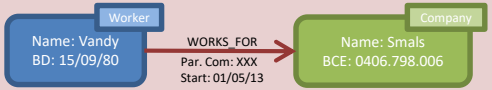
Requête Neo4j



Liste des employés de Smals apparentés

```
MATCH
    (w1:Worker)-[:WORKS_FOR]->(c:Company {Name:"Smals"})
     <-[:WORKS_FOR]-(w2:Worker)-[:RELATIVES]->(w1)
RETURN w1.Name, w2.Name
```

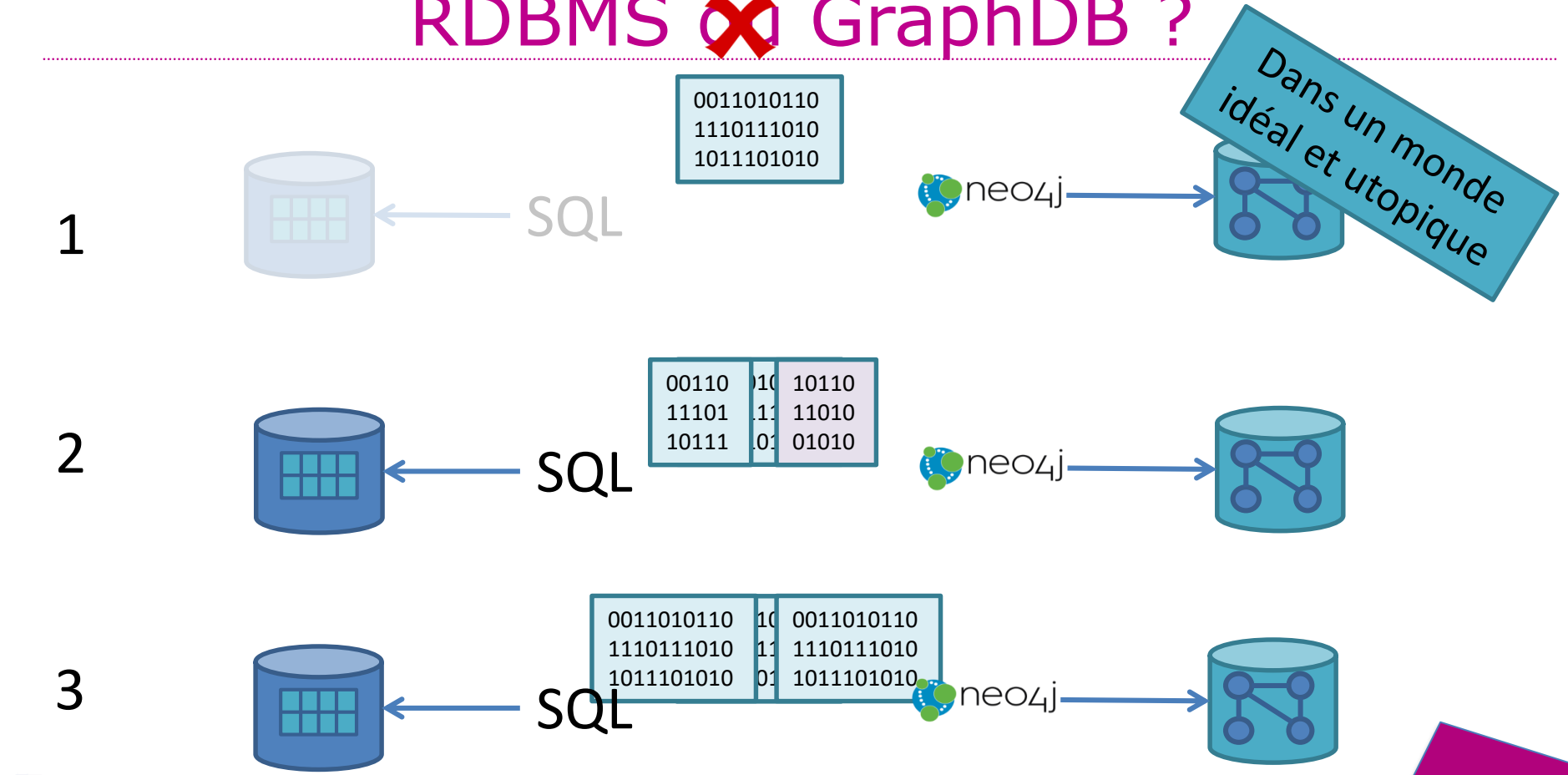
Fonctionnement

	RDBMS	Graph DB
schema		
WHERE …	Search in Companies (with index): O(log([Table size])	Search in nodes (with index) : O(log([DB size])
Relation	JOIN : - Search in Works_for : O(log([Table size])) - Search in Workers : O(log([Table size]) x [nb res])	Follow pointers in node : O([nb of employees])

Interroger un réseau

GRAPHDB : LA SOLUTION ?

RDBMS et GraphDB ?



GraphDB : Avantages

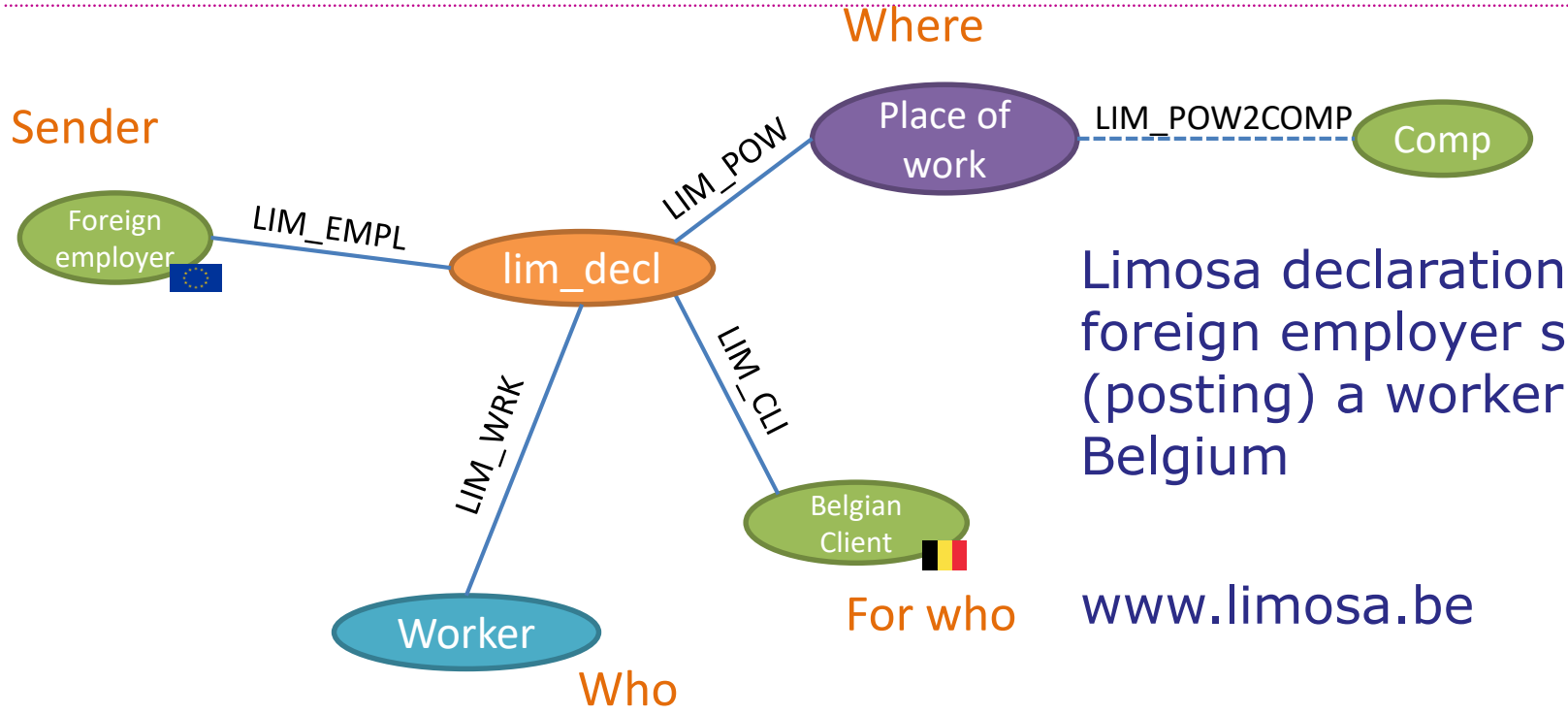
- Langage plus proche du modèle
- Pas de clés à gérer, ni de contrainte d'intégrité
- Pas de structures de relations à chaque requête
- Très efficace pour le parcours de relations
- Beaucoup de champs d'application : recommandation, fraude, infra, MDM, KM…

GraphDB : limitations

- Pas la maturité des RDBMS (robustesse, haute dispo., communauté…)
- Pas de standard
- Survie à l'effet de mode ? (cf OO DB)
- Pas optimal pour des recherches, agrégation, batch process, transactionnel
- Pas une alternative, plutôt un complément (ne résout pas les mêmes problèmes)

USE CASE: SOCIAL DUMPING

Limosa: detachment declaration



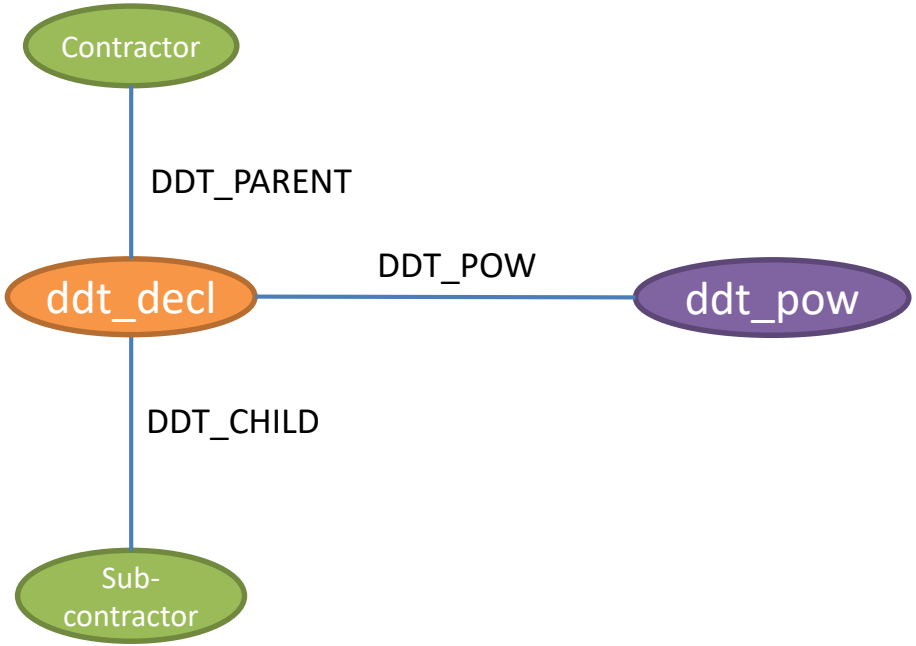
Limosa declaration: A foreign employer sending (posting) a worker in Belgium

www.limosa.be

Limosa : Landenoverschrijdend Informatiesysteem ten behoeve van MigratieOnderzoek bij de Sociale Administratie

Syst. d'infor. transfrontalier en vue de la recherche en matière de migration auprès de l'adm. sociale

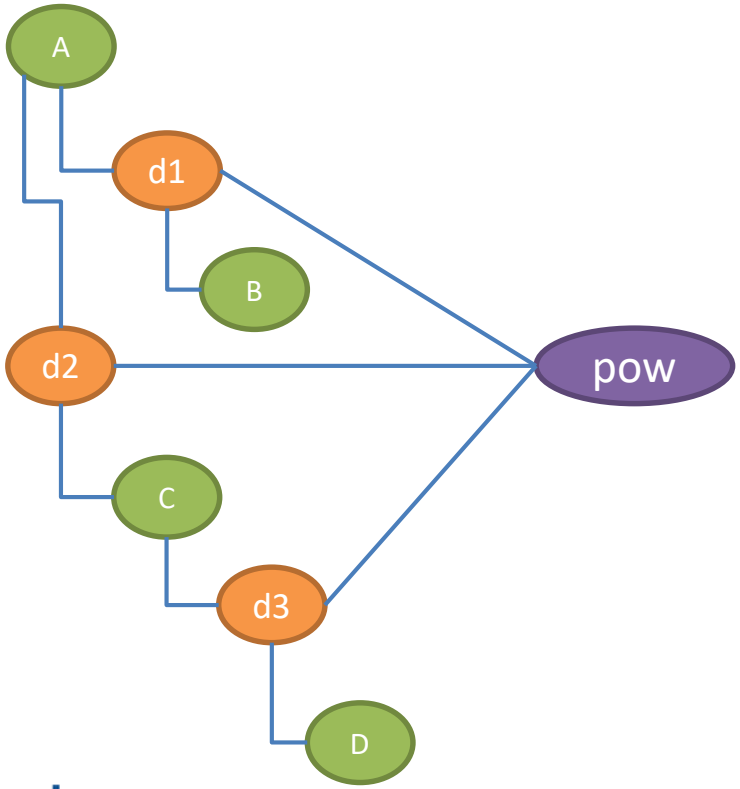
DDT: sub-contractors



DDT: Déclaration De Travaux

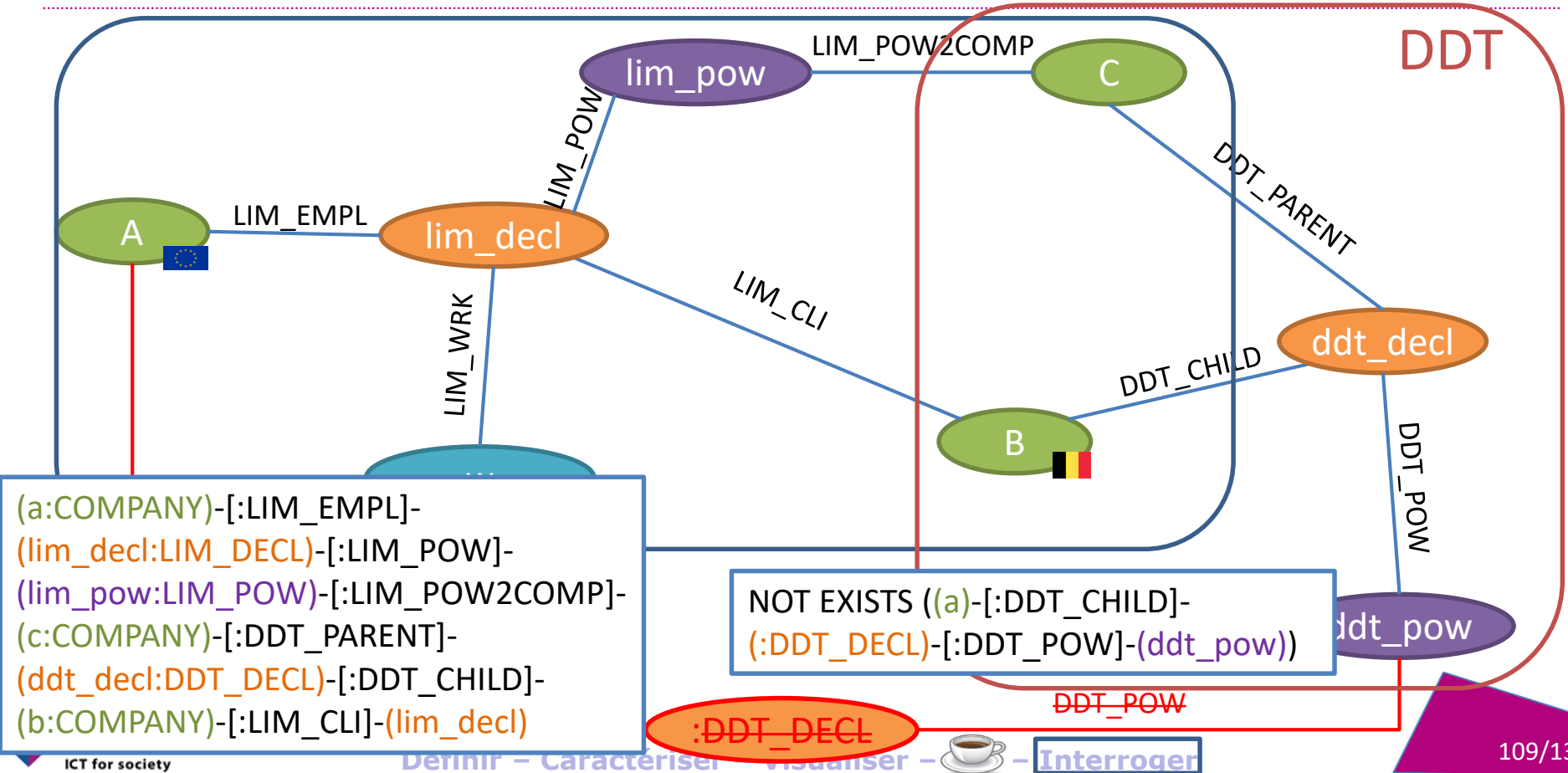
Aangifte van werken

DDT: sub-contractors



On a specific worksite (PoW), we have a chain/tree of sub-contractors

Full schema


DDT
lim_pow
LIM_POW2COMP
C
LIM_POW
DDT_PARENT
A
LIM_EMPL
lim_decl
LIM_CLI
LIM_WRK
ddt_decl
DDT_CHILD
B
DDT_POW
(a:COMPANY)-[:LIM_EMPL]-
(lim_decl:LIM_DECL)-[:LIM_POW]-
(lim_pow:LIM_POW)-[:LIM_POW2COMP]-
(c:COMPANY)-[:DDT_PARENT]-
(ddt_decl:DDT_DECL)-[:DDT_CHILD]-
(b:COMPANY)-[:LIM_CLI]-(lim_decl)
NOT EXISTS ((a)-[:DDT_CHILD]-
(:DDT_DECL)-[:DDT_POW]-(ddt_pow))
ddt_pow
DDT_POW
:DDT_DECL

Full query

```
MATCH
 (a:COMPANY)-[:LIM_EMPL]-(lim_decl:LIM_DECL)-[:LIM_POW]-
 (lim_pow:LIM_POW)-[:LIM_POW2COMP]-(c:COMPANY)-[:DDT_PARENT]-
 (ddt_decl:DDT_DECL)-[:DDT_CHILD]-(b:COMPANY)-[:LIM_CLI]-(lim_decl),
 (ddt_decl)-[:DDT_POW]-(ddt_pow:DDT_POW)
WHERE
 NOT EXISTS ((a)-[:DDT_CHILD]-(:DDT_DECL)-[:DDT_POW]-(ddt_pow)) …
RETURN
 DISTINCT a, b, c,
 COUNT(DISTINCT lim_decl),
 COUNT(DISTINCT ddt_decl)
ORDER BY COUNT(DISTINCT lim_decl) DESC
```

Implementation

- We copied in Neo4j 17M nodes, 45M relationships :
  - 800.000 companies, 1.650.000 (foreign) workers
  - 13 millions declarations
  - 185.000 duplication groups
  - 250.000 streets
  - …
- 9 node labels, 15 relationship types
- All companies triplets matching 1st pattern: ~10 seconds (325 results)

In SQL/RDBMS ?

- In SQL/RDBMS,
  - Any node and any relationship is a record : ~65 M records
  - Any type/label corresponds to a table : 24 tables
- Any (n:Type1)-[:REL1]-(m:Type2) corresponds to :

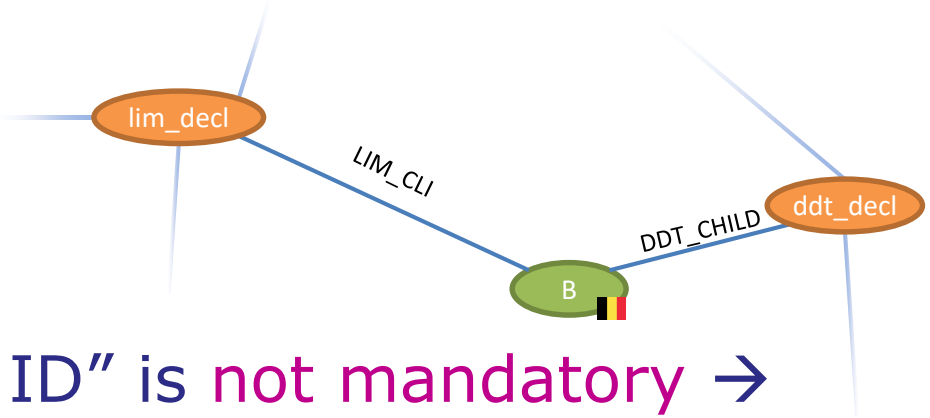
```
FROM Type1
JOIN REL1  ON Type1.key = REL1.type1_key
JOIN Type2 ON Type2.key = REL1.type2_key
```

→ Almost 20 joins!

- → Query far more complex to write
- → Much longer to run

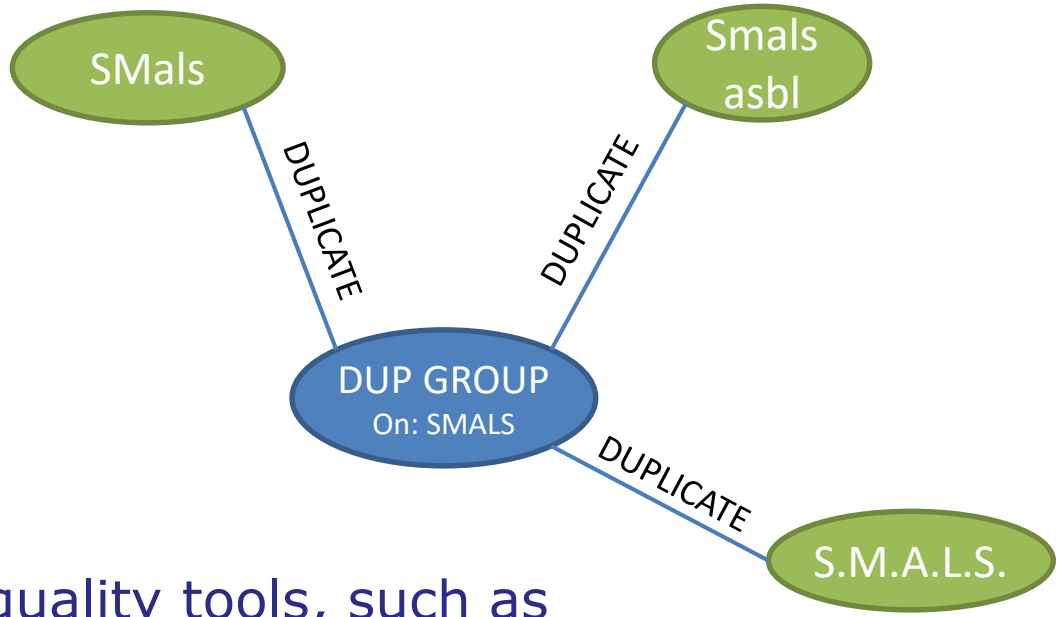
Data quality problem

- This only works if « B » has been declared with the same ID in both systems !



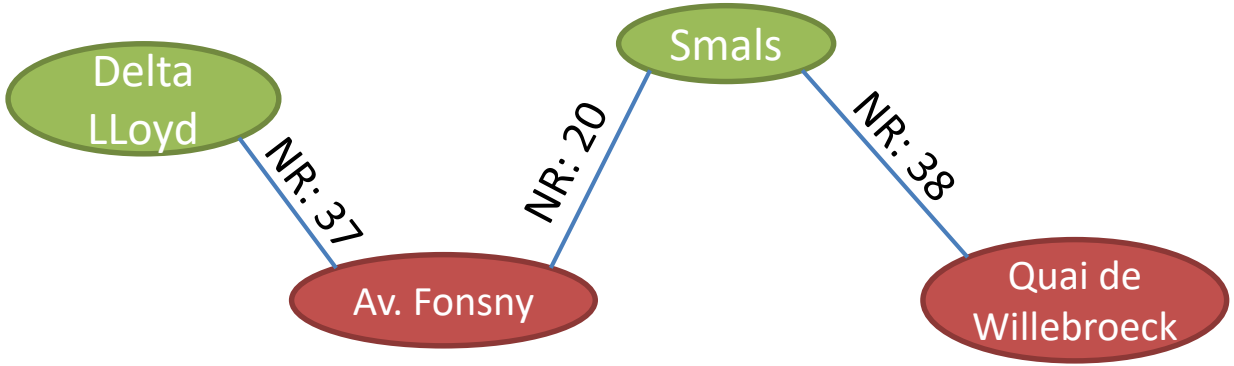
- Sometimes, the “official ID” is not mandatory → name + address is OK
- We have to deal with this!

Duplicates



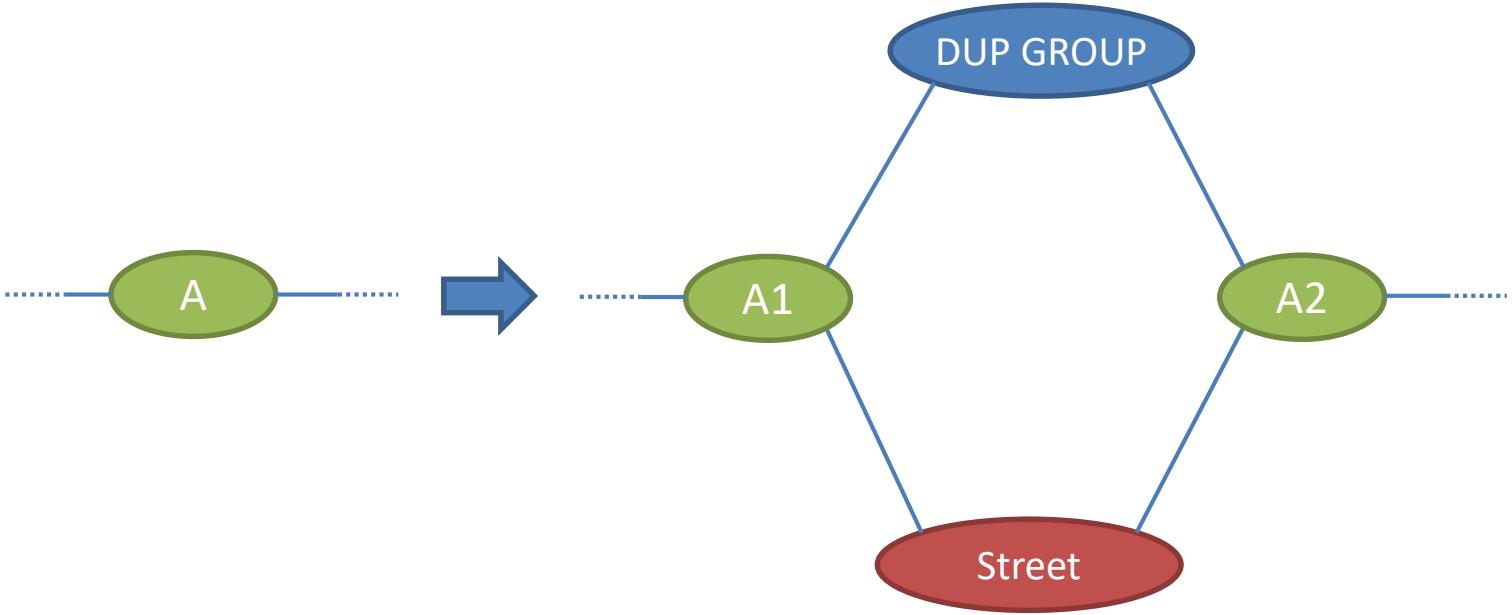
Thanks to data quality tools, such as Trillium (IntoDQ), OpenRefine…

Addresses

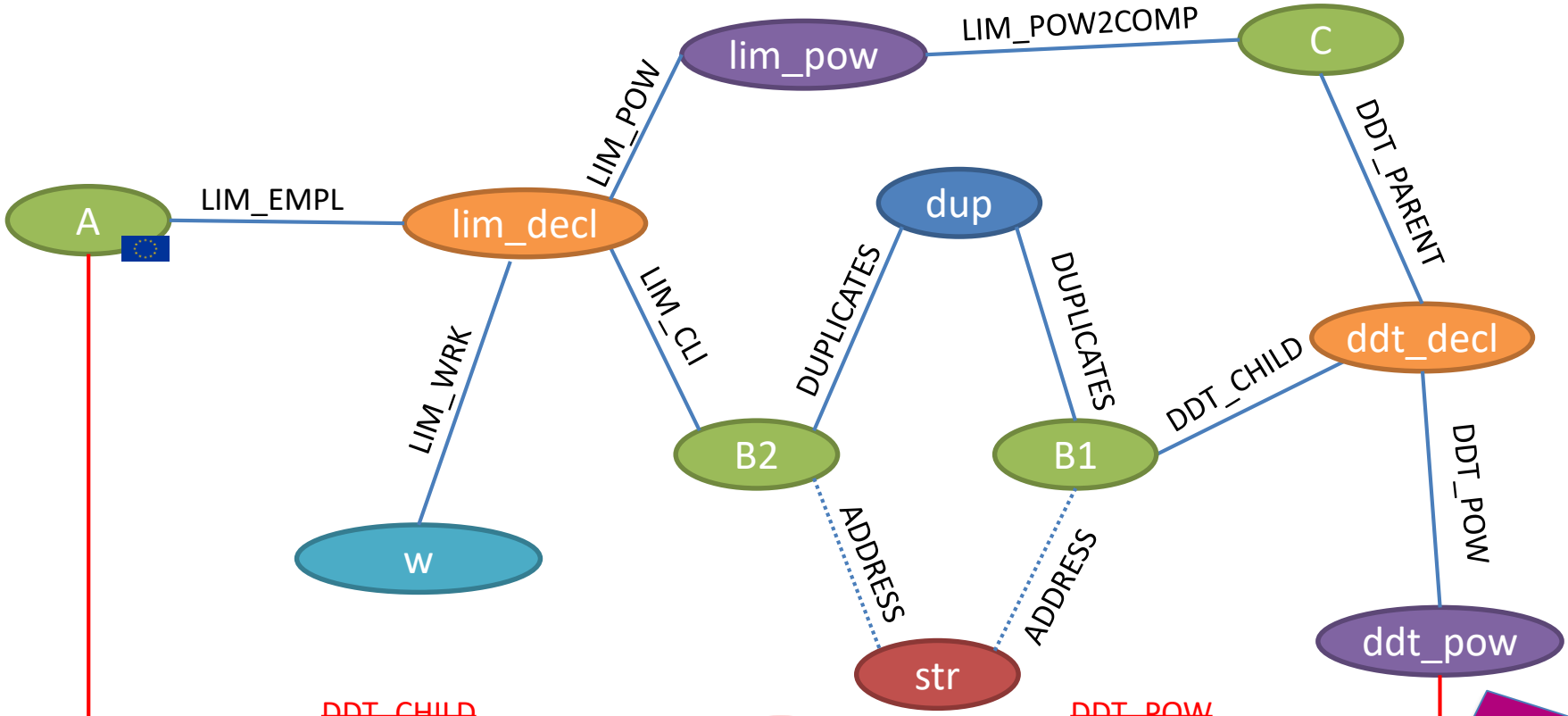


A standardization is needed! Thanks to data quality tools

Duplicates combinations



Schema with B duplicated



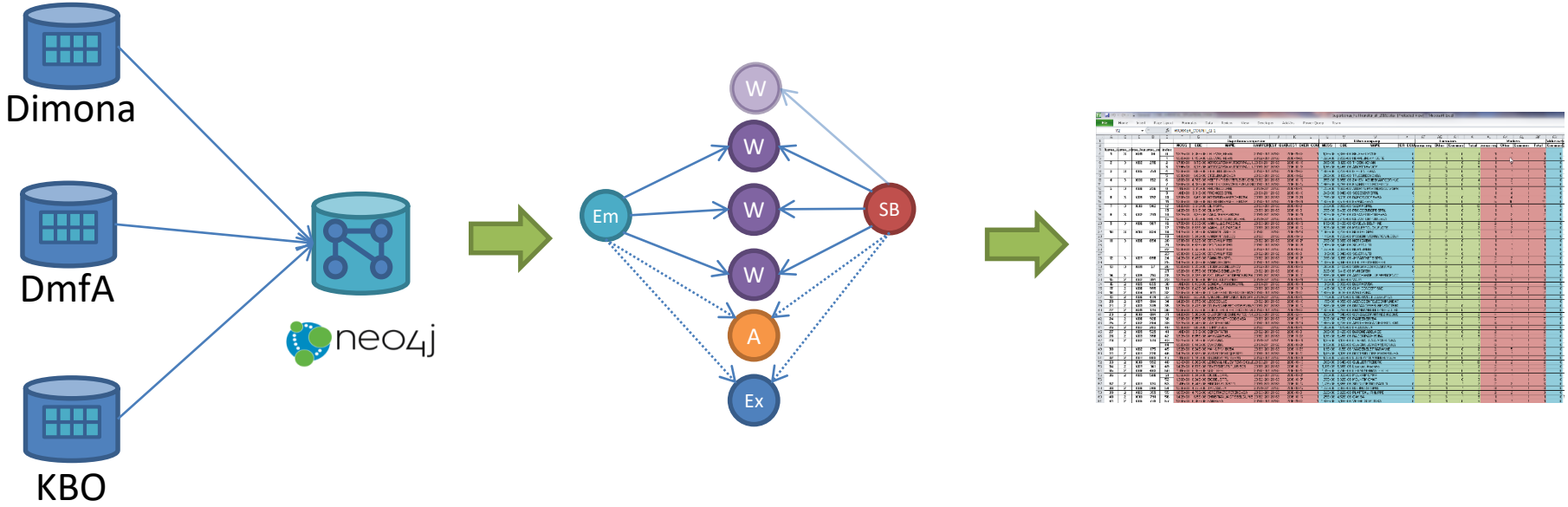
Interroger un réseau

USE CASE : SUPERBONUS

Superbonus : contexte

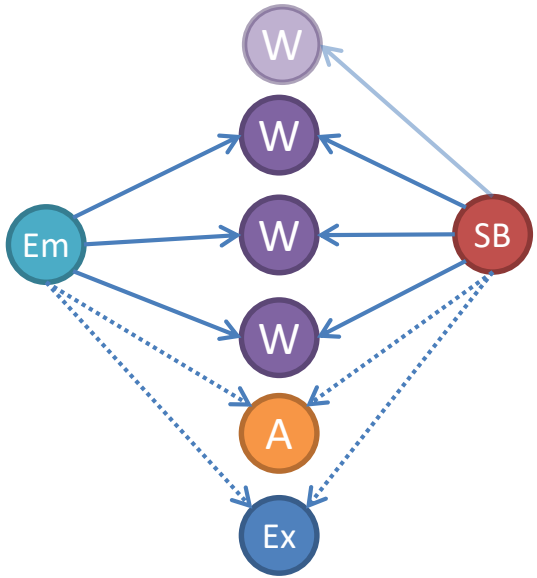
- Avec le TaxShift (2016), des avantages sont donnés aux nouveaux employeurs
- Beaucoup d'employeurs existants se recréent pour profiter de ces avantages
- Le « Network Analytics » permet d'identifier beaucoup de ces cas

Superbonus



W Worker Em Employer without Superbonus SB Employer with Superbonus A Address Ex Executor

Full transfert



COUNT(W) > Em.Worker_Cnt_2015 * 80%
COUNT(W) > SB.Worker_Cnt_2016 * 80%

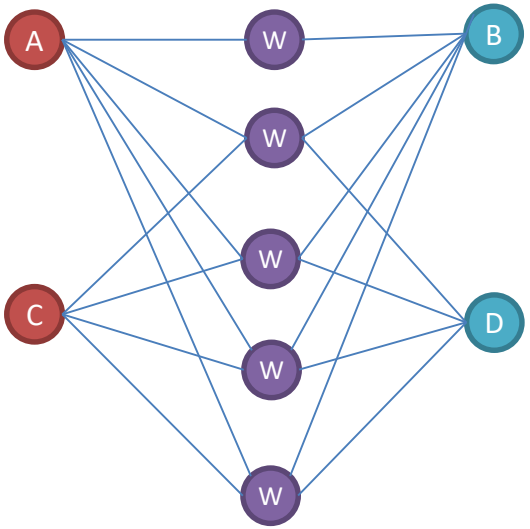
```
MATCH   (c_sb:SB)--(w:Worker)--(c_old:Company)
WHERE   c_sb <> c_old
OPTIONAL MATCH (c_sb)--(e:Executor)--(c_old)
OPTIONAL MATCH (c_sb)--(a:Address)--(c_old)
WITH    c_sb, c_old,
        COUNT(DISTINCT w) AS w_count
WHERE
   w_count > c_sb.WORKER_CNT_16 * 0.8 AND
   w_count > c_old.WORKER_CNT_15 * 0.8
RETURN c_sb, c_old, w_cnt
```

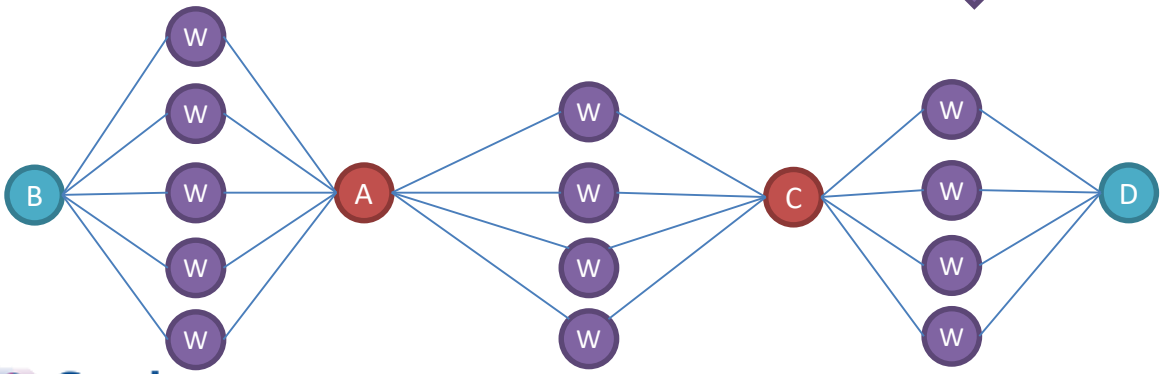
W Worker Em Employer without SB SB Employer with SB A Address Ex Executor

Table vs visualisation

SB req.	Other	Common workers
A	B	5
A	C	4
C	D	4

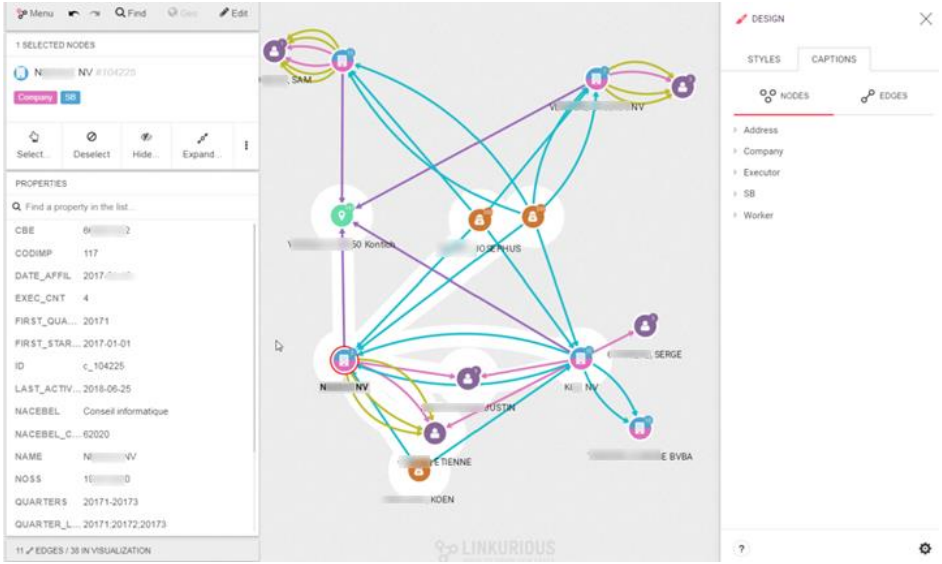
Une table …
Deux schémas

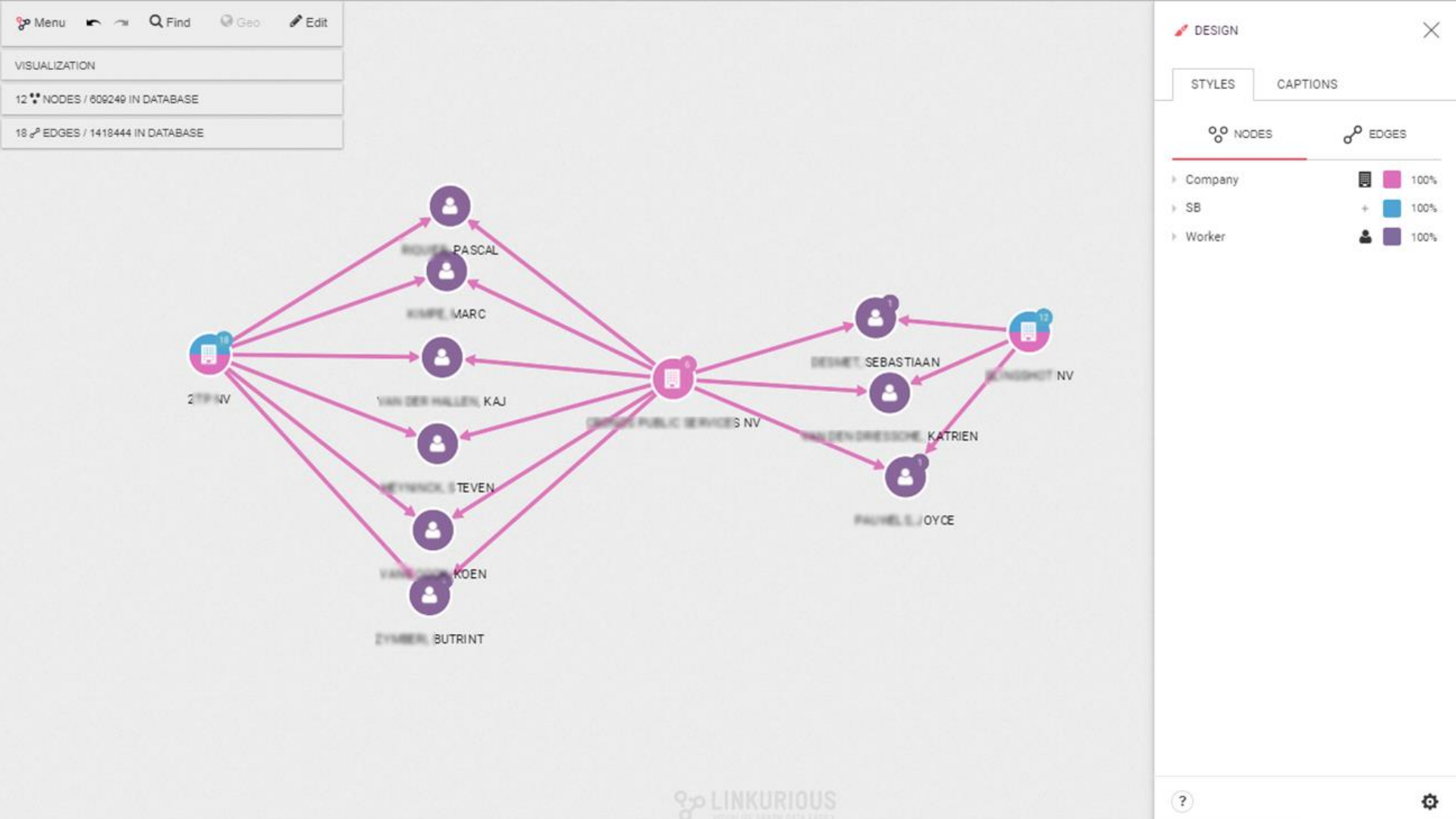


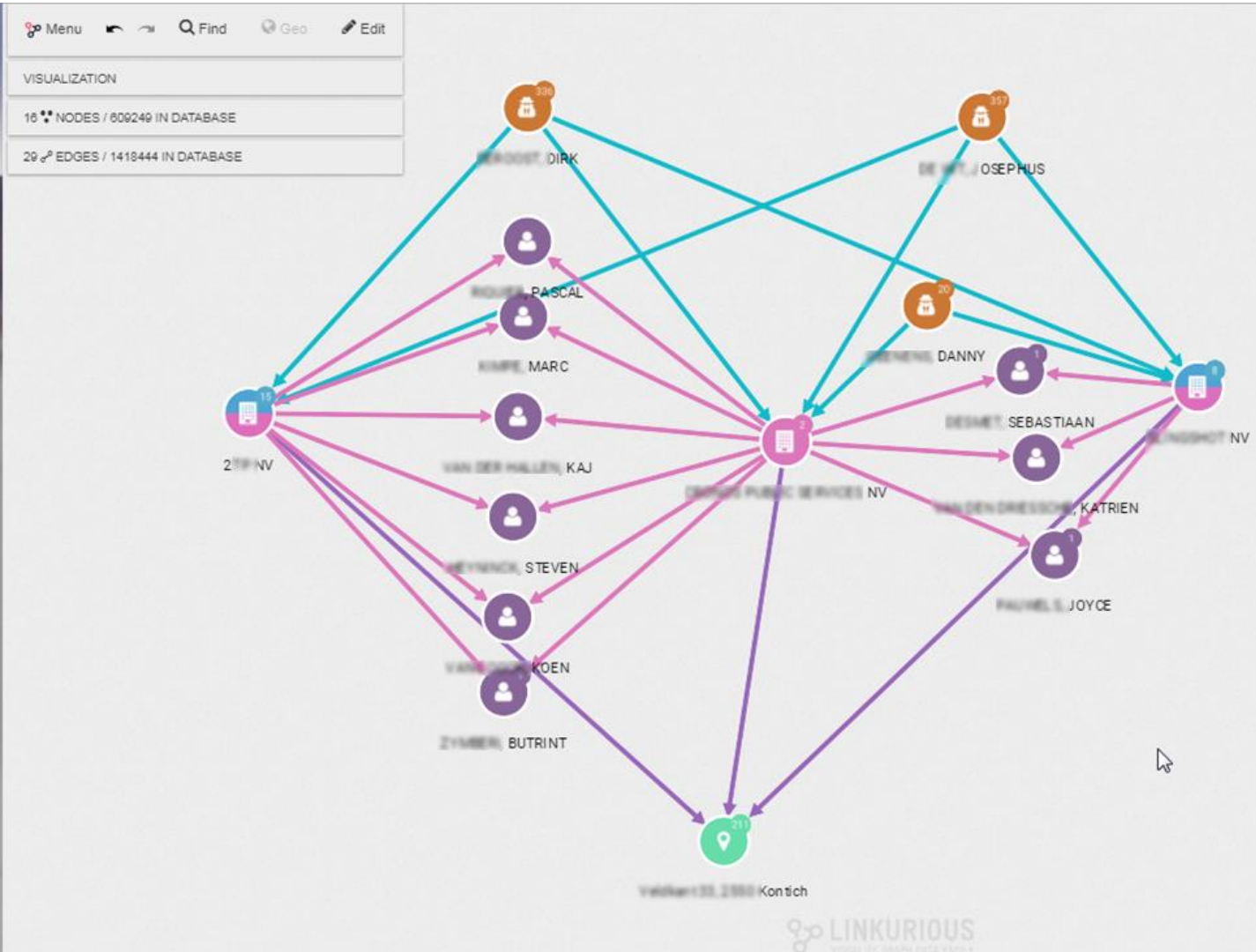


Linkurious

- Outil de visualisation d'une graph database
- Utilisable par des end users




Menu
Find
Geo
Edit
VISUALIZATION
12 NODES / 609249 IN DATABASE
18 EDGES / 1418444 IN DATABASE
PASCAL
MARC
KAJ
STEVEN
KOEN
BUTRINT
SEBASTIAAN
NV
KATRIEN
OYCE
DESIGN
STYLES
CAPTIONS
NODES
EDGES
Company
100%
SB
100%
Worker
100%
LINKURIOUS



DESIGN

STYLES | CAPTIONS

NODES | EDGES

- Address 100%
- Company 100%
- Executor 100%
- SB 100%
- Worker 100%

Conclusions

En résumé

Caractériser (théorie) : tout un réseau, un nœud (centralité), deux nœuds (distance), un groupe de nœuds (clustering)

Visualiser : Desktop, Javascript, Cloud

Manipuler (coding) : Envir. classiques ou Big Data

Interroger (graph DB) : focus sur les relations

Application : fraude, gestion des infras/logicielle, recommandation, intelligence, KM…

Conclusions

- De la place pour les graph analytics/graph DB dans nos institutions
- Pas uniquement pour la fraude !
- Pour les data-scientits/BI :
  - nouvelle façon de réfléchir, centré sur les relations
- Pour les DBA/développeurs :
  - nouvelle famille de DB, complémentaires
- Pour les analystes/business :
  - inspiration pour de nouveaux use cases

Conclusions : expérience personnelle

- Préparation des données :
  - R/Python + igraph/graphx
- Première analyse :
  - Gephi/TouchGraph
- Production :
  - GraphDB (Neo4j…)
- Pour le client :
  - Interactif (futur) : libraire JS / outils « à la » Linkurio.us
  - Statique : Gephi/TouchGraph, igraph → Excel, PDF…

Références : www.smalsresearch.be

- https://www.smalsresearch.be/author/berten/
- Un fraudeur ne fraude jamais seul (blog 1, blog 2)
- Graph DB vs RDBMS (blog 1, blog 2)
- Gérer les doublons dans une Graph DB (blog)
- Le marché du travail salarié en Belgique : une analyse réseau (blog 1, blog 2, blog 3)
- 7 raisons d'utiliser une Graph Database (blog)

Références

- **Graph Databases**, *Robinson & all*, O'Reilly 2015
- **Fraud Detection: Discovering Connections with Graph Databases**, *Neo4j*, Sodawksi & Rathle
- **Fraud Analytics Using Descriptive, Predictive, and Social Network Techniques: A Guide to Data Science for Fraud Detection**, *Bart Baesens, Veronique Van Vlasselaer, Wouter Verbeke*, Willey, 2015

Vandy Berten
02/787.57.32
vandy.berten@smals.be

More on Smals Research :
Website : www.smalsresearch.be
Blog : www.smalsresearch.be/blog
Twitter : @SmalsResearch