

# Graph & Network Analytics

The power of relations

Vandy BERTEN

Section Recherche

6 juin 2017

# Table des matières

---

Définir un réseau

Caractériser un réseau

Visualiser un réseau

Manipuler un réseau

Interroger un réseau





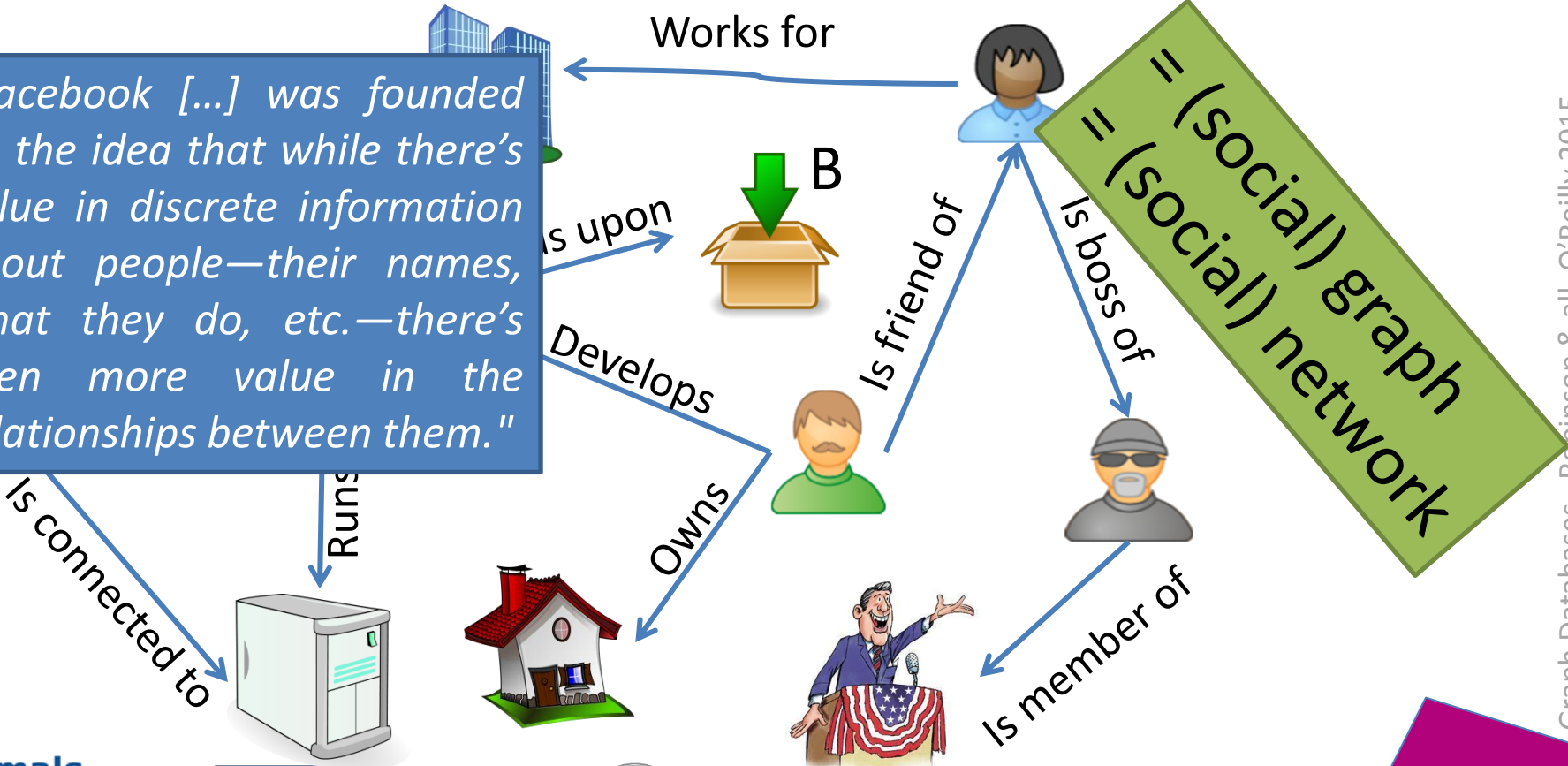
# Définir un réseau

Définir un réseau

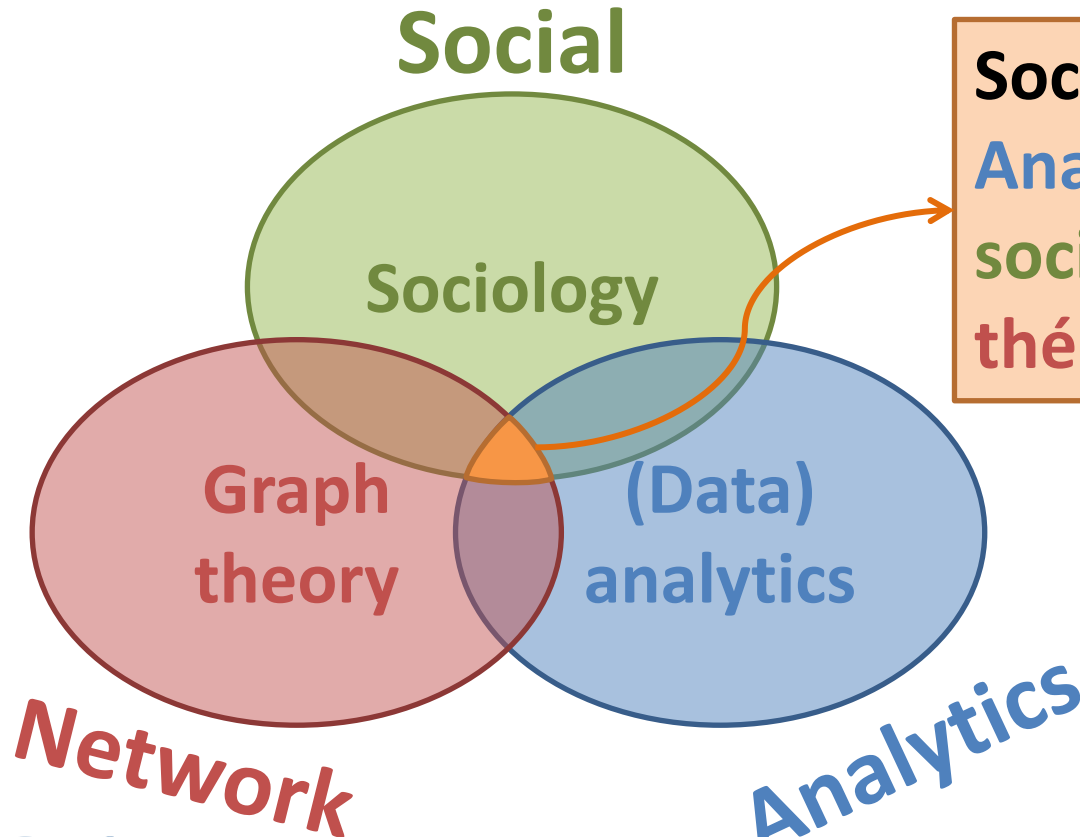
# MOTIVATIONS

# Motivations

"Facebook [...] was founded on the idea that while there's value in discrete information about people—their names, what they do, etc.—there's even more value in the relationships between them."



# Social Network Analytics

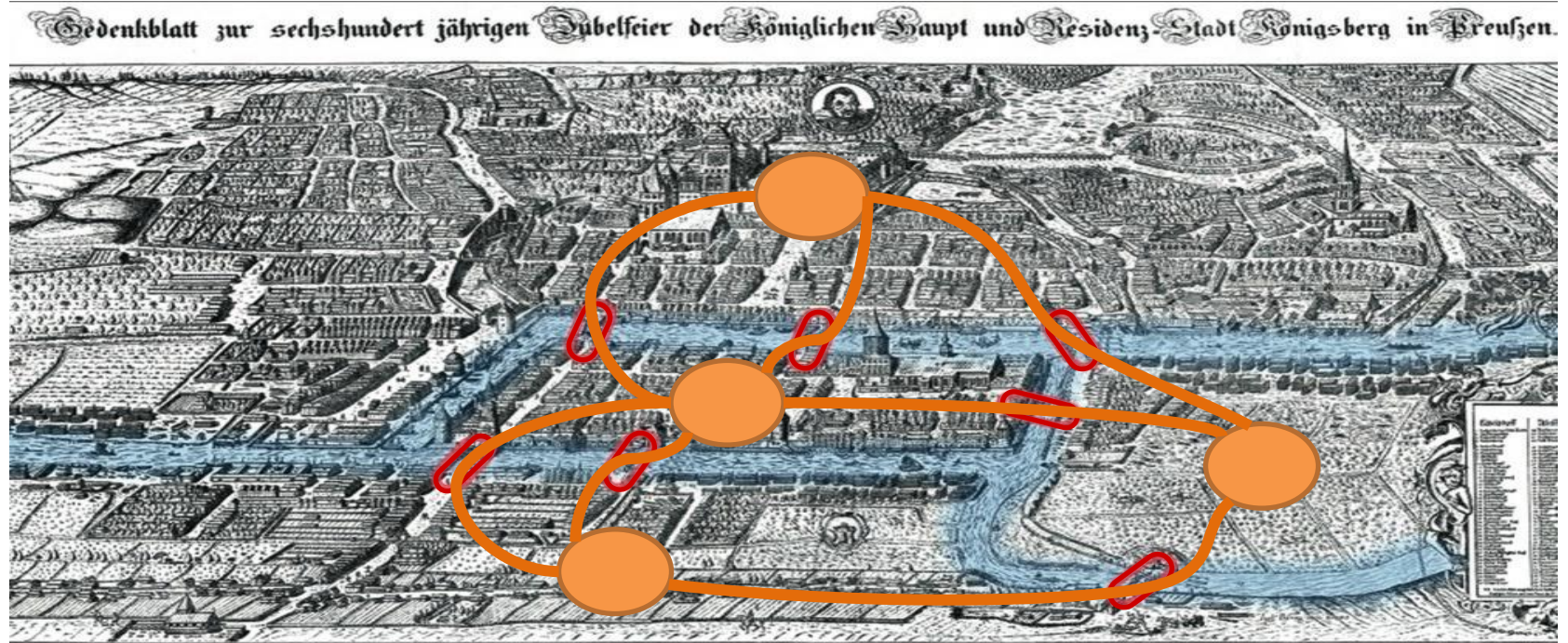


**Social Network Analytics :**  
**Analyse** des **structures sociales** au travers de la **théorie des graphes**

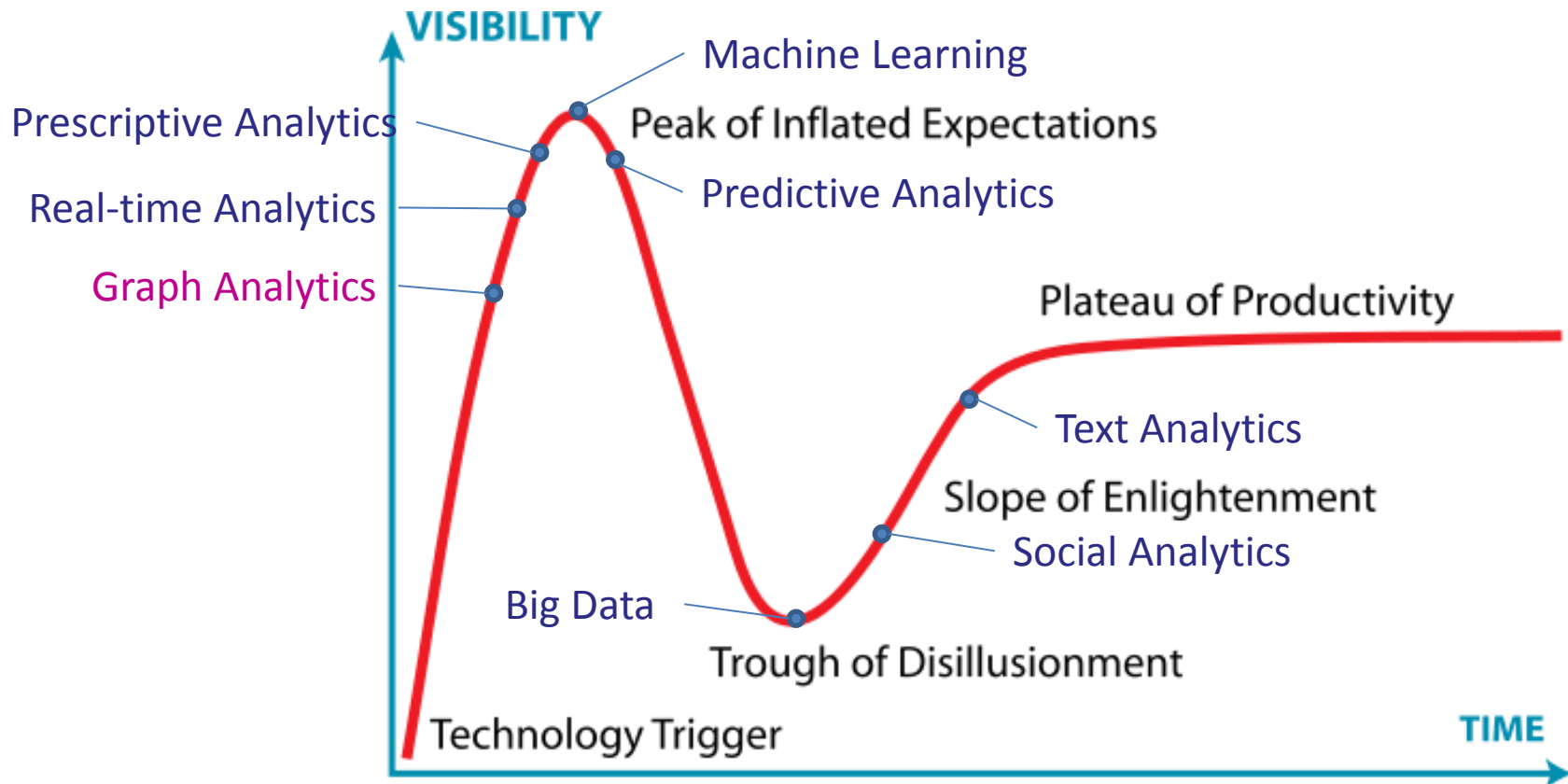
# (Social) Graph/Network Analytics is not...



# 7 ponts de Königsberg (Euler, 1736)



# Gartner : BI and Analytics/Data Science



# Gartner : applications



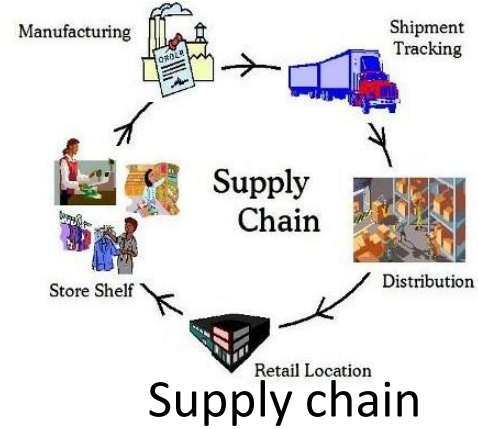
Route optimization



Epidemiology



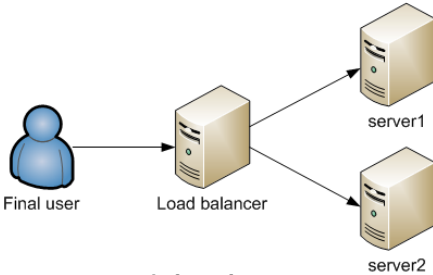
Intelligence



Supply chain



Fraud detection



Load balancing



Market basket analysis



Genome

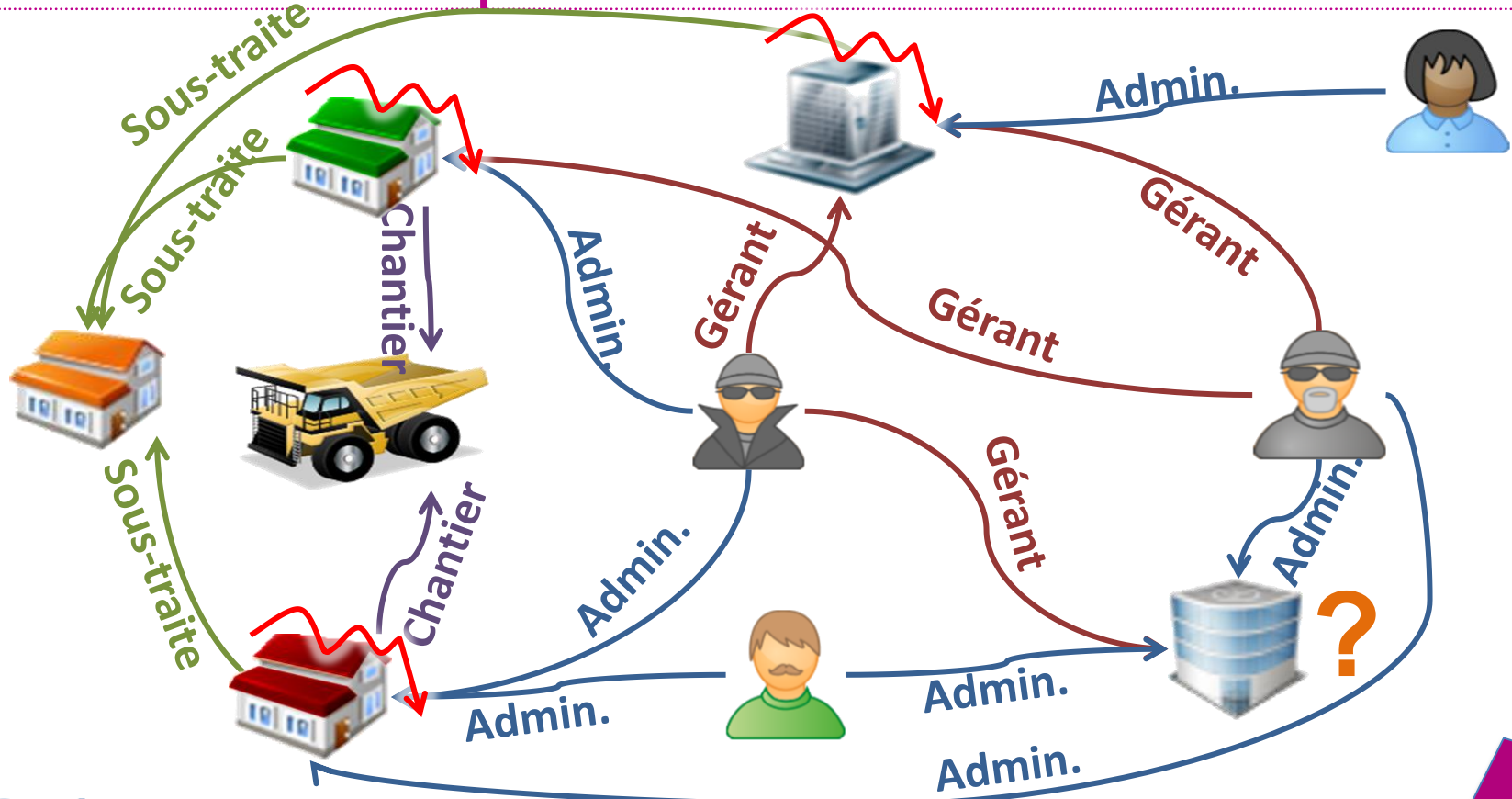


Money laundering

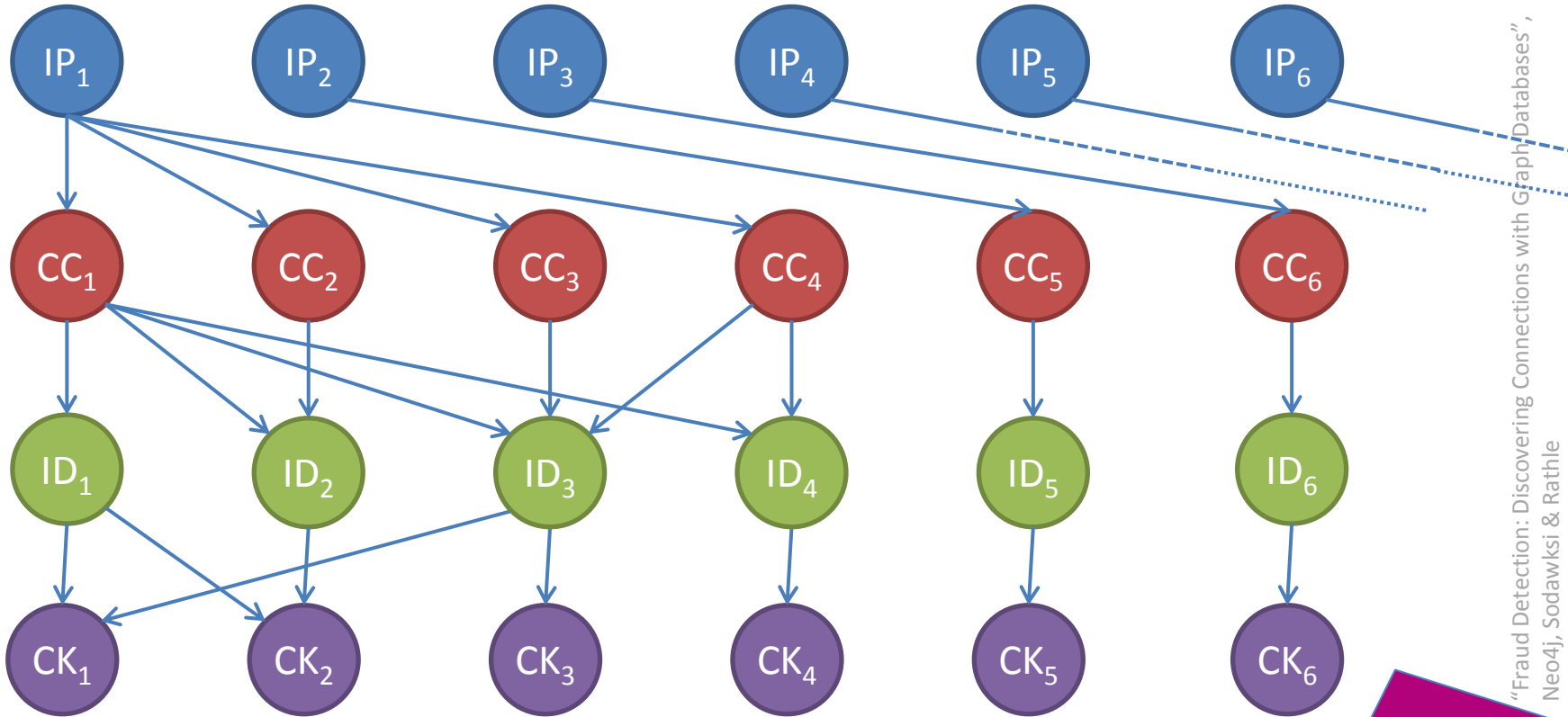
Définir un réseau

# EXEMPLES

# Spider construction



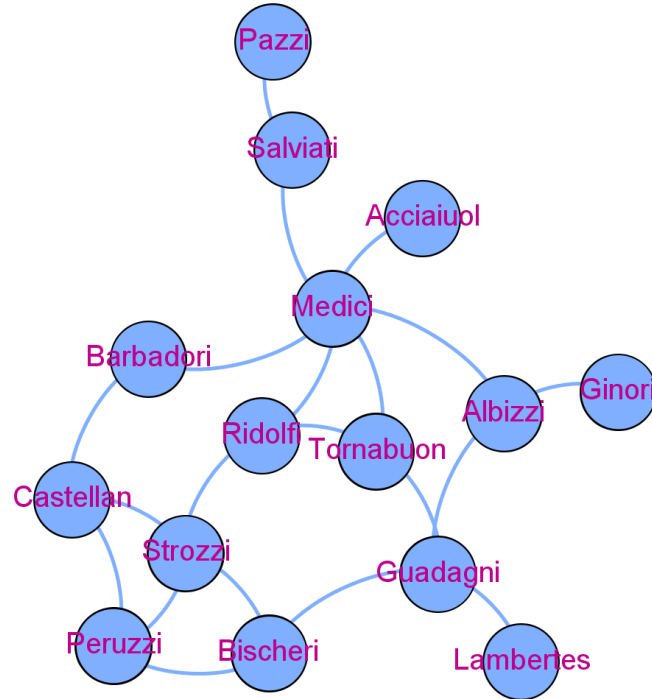
# e-commerce



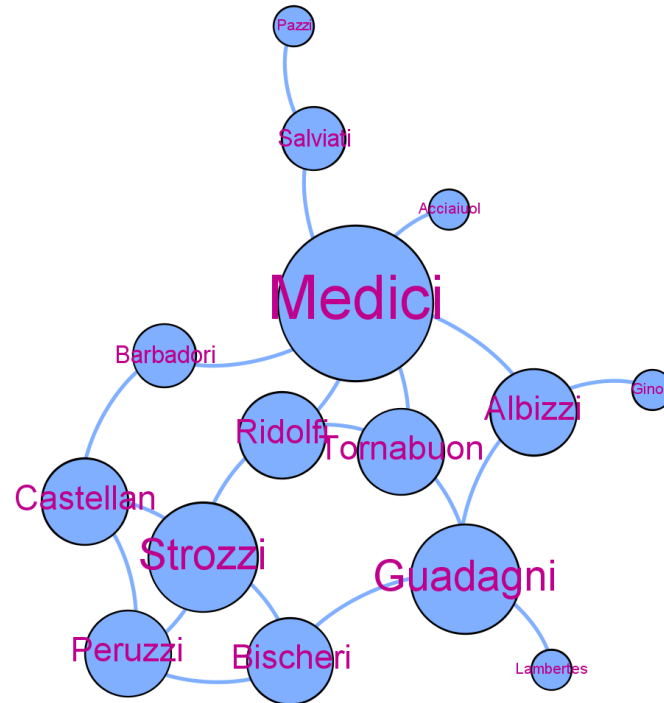
“Fraud Detection: Discovering Connections with Graph Databases”,  
Neo4j, Sodawski & Rathle



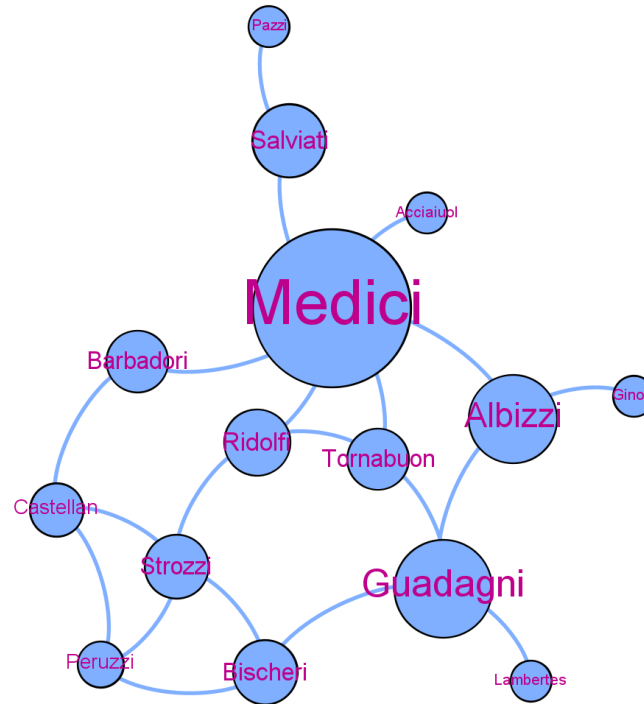
# Mariages à Florence



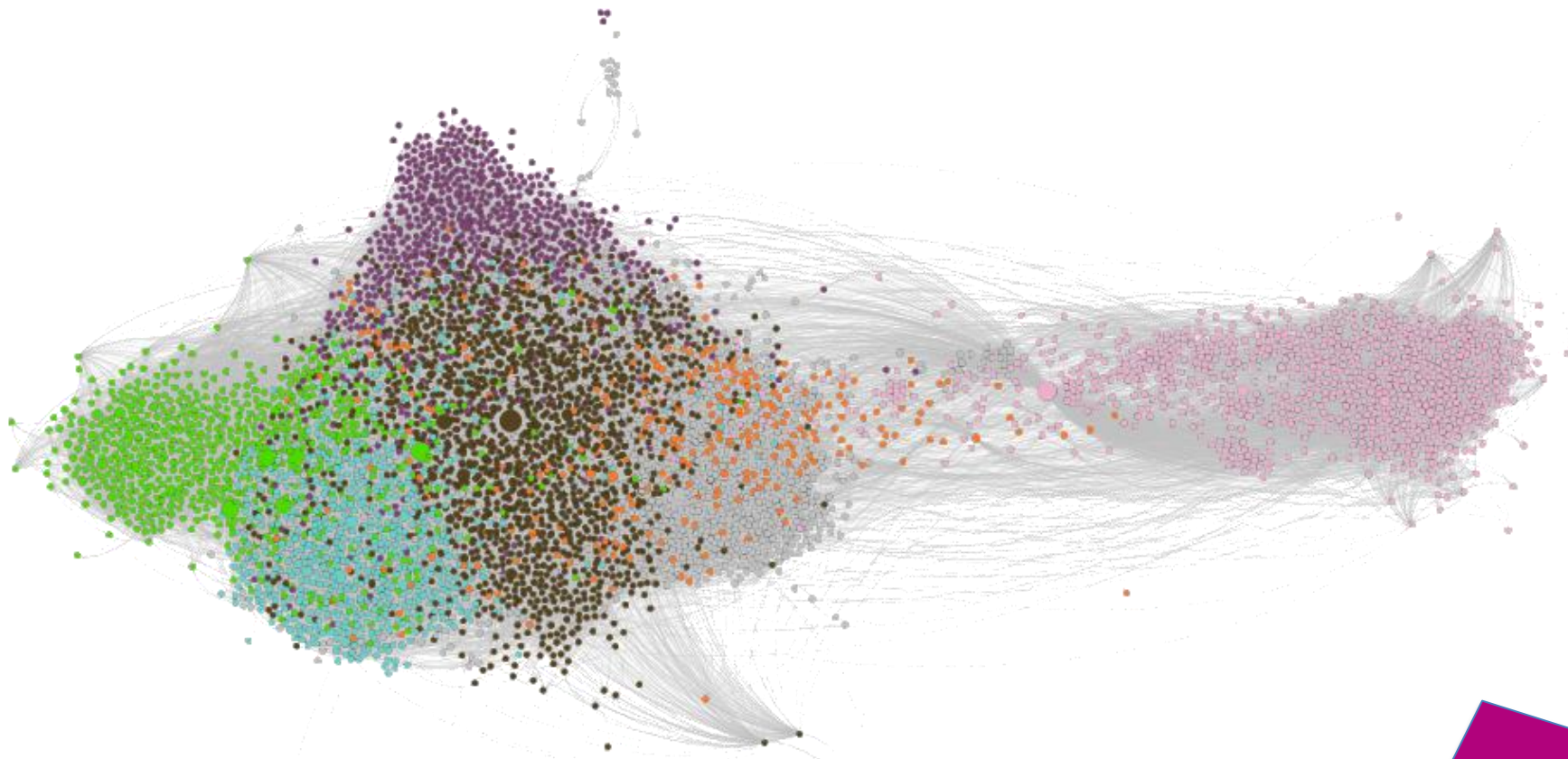
# Mariages à Florence



# Mariages à Florence



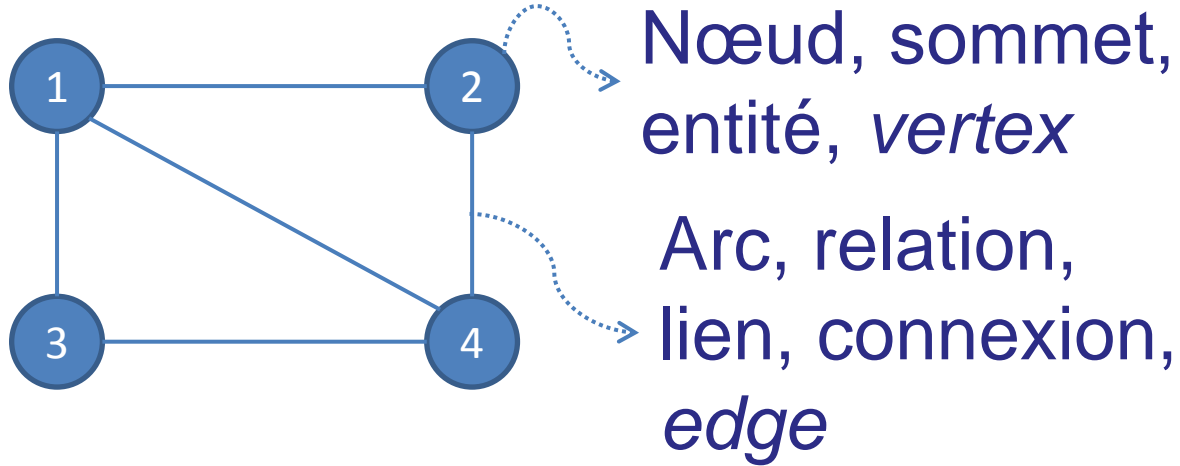
# Dépendance de bibliothèques Python



Définir un réseau

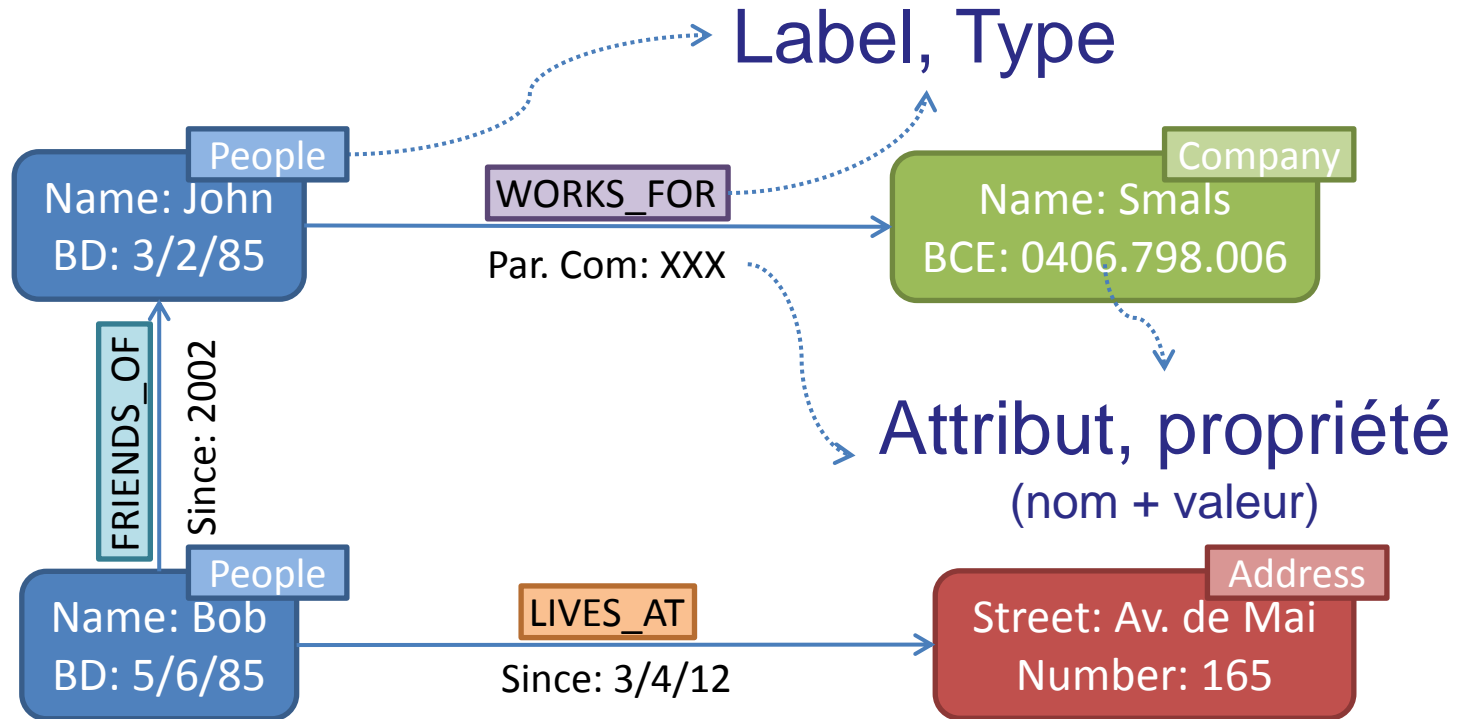
# DÉFINITIONS

# Nœuds, arcs



$$G = (V, E)$$

# Label, attribut



# Réseau dirigé/non-dirigé

---

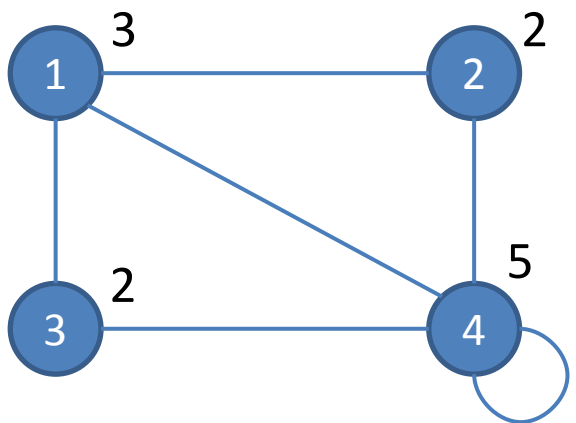


Non dirigé (*undirected*) :  
collègue, ami Facebook



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

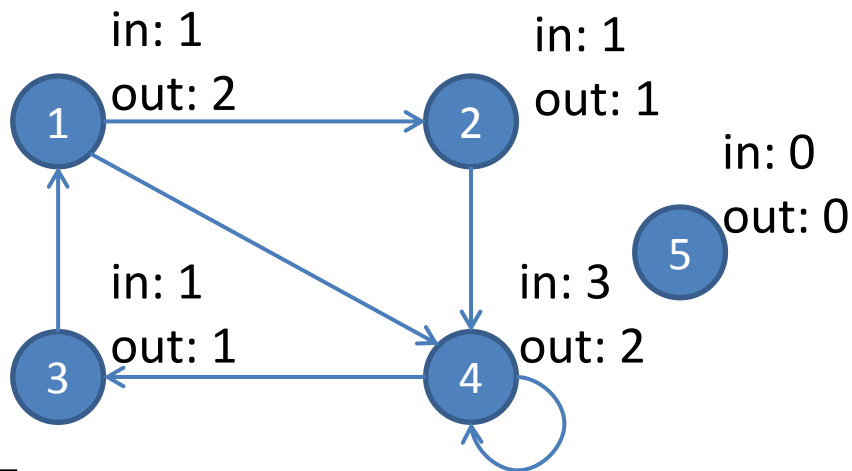
# Degré



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

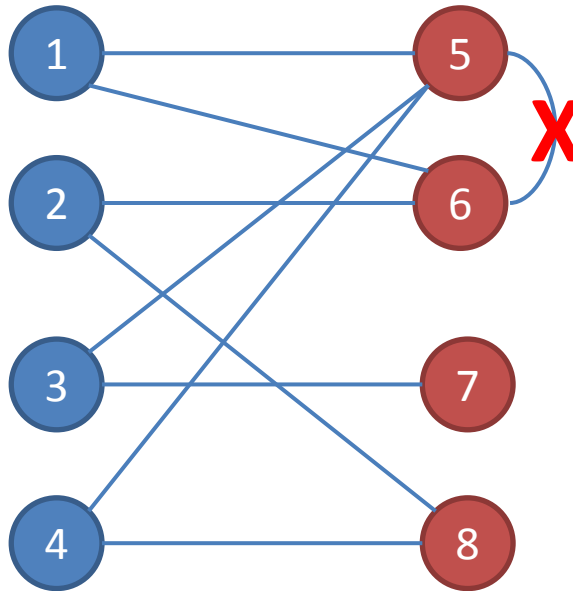


$$\sum_{v \in V} \text{deg}(v) = 2 |E|$$



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

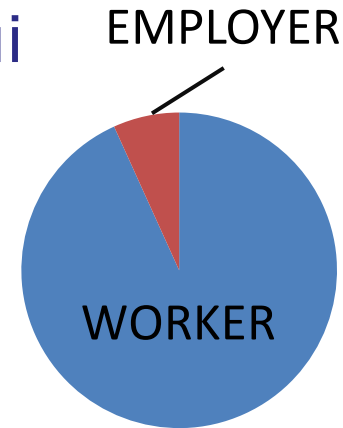
# Graphe biparti (*bipartite graph*)



# Exemple : Dimona



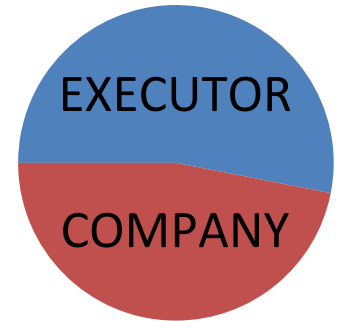
- Déclarations Dimona de 2007 à aujourd'hui
- 7 048 178 nœuds, 20 640 727 relations
- Graphe biparti
- Degré d'un nœud :
  - EMPLOYER : nombre de ses employés
  - WORKER : nombre de ses employeurs



# 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)



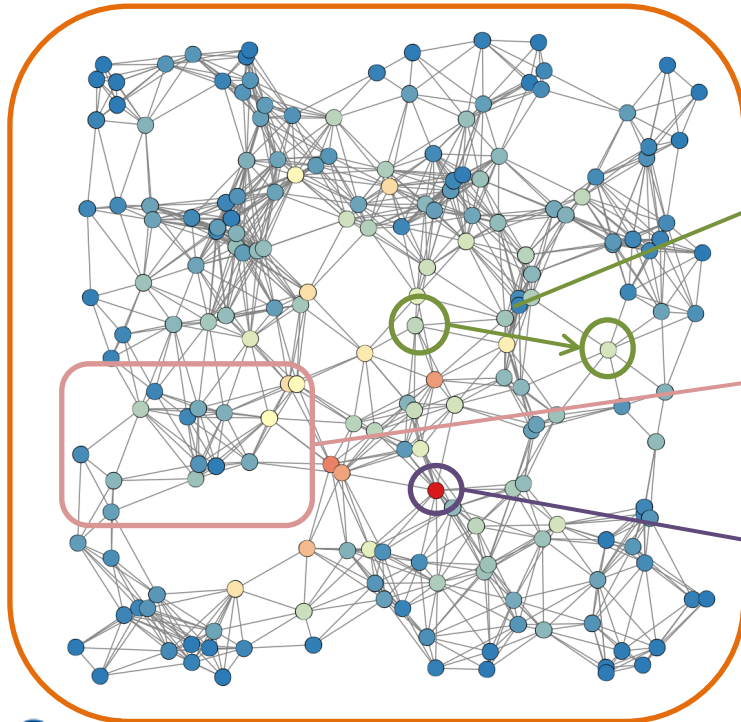


# Caractériser un réseau

# Caractériser un réseau

Le réseau dans son ensemble  
(Métriques générales)

On peut caractériser :



La relation/distance/similarité  
entre deux nœuds

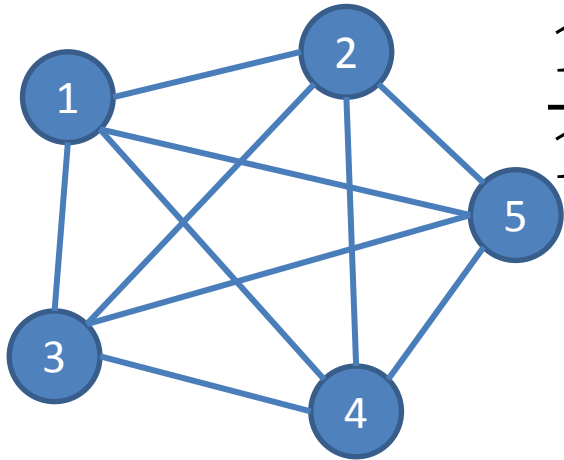
Un groupe de nœuds (Clustering)

Un nœud en particulier (Centralité)

Caractériser un réseau

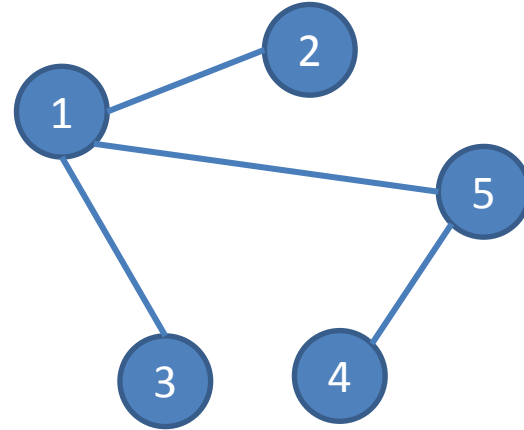
# MÉTRIQUES GÉNÉRALES

# Densité



$$\frac{10}{10} = 1$$

vs.

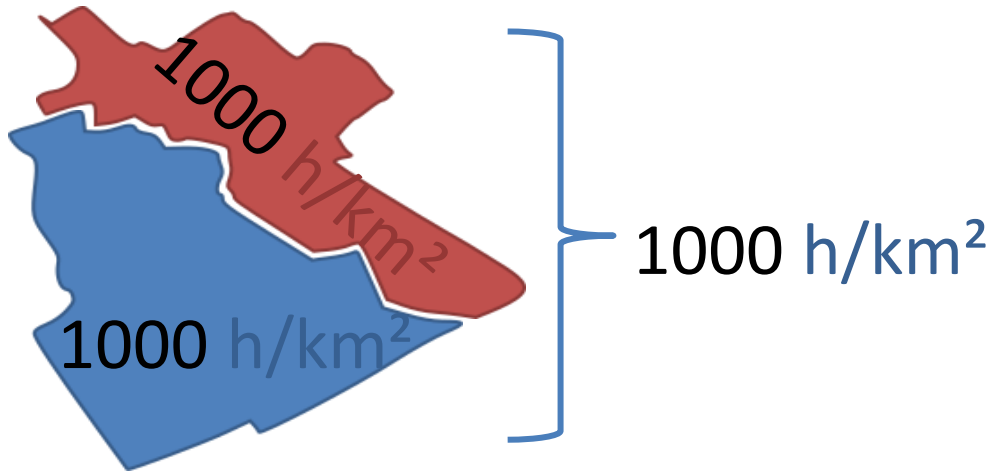


$$\frac{4}{10} = 0,4$$

$$\text{Densité} = \frac{\text{Nombre d'arcs}}{\text{Nombres d'arcs possibles}}$$

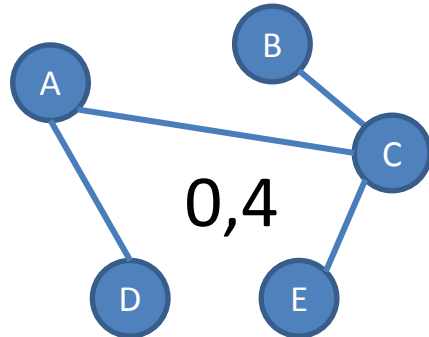
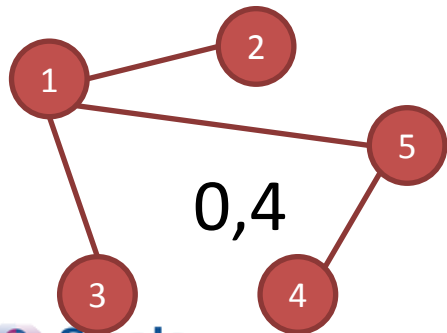
$$= \frac{|E|}{(|V|-1)+(|V|-2)+\dots+1} = \frac{2|E|}{|V| \times (|V|-1)}$$

# Densité ≠ Densité !



Difficile de comparer 2 graphes de tailles différentes !

Définition alternative :  
$$\frac{\text{Nombre d'arcs}}{\text{Nombres de noeuds}}$$



$$\frac{2 \times (4 + 4)}{10 \times 9} = 0,177 \dots$$

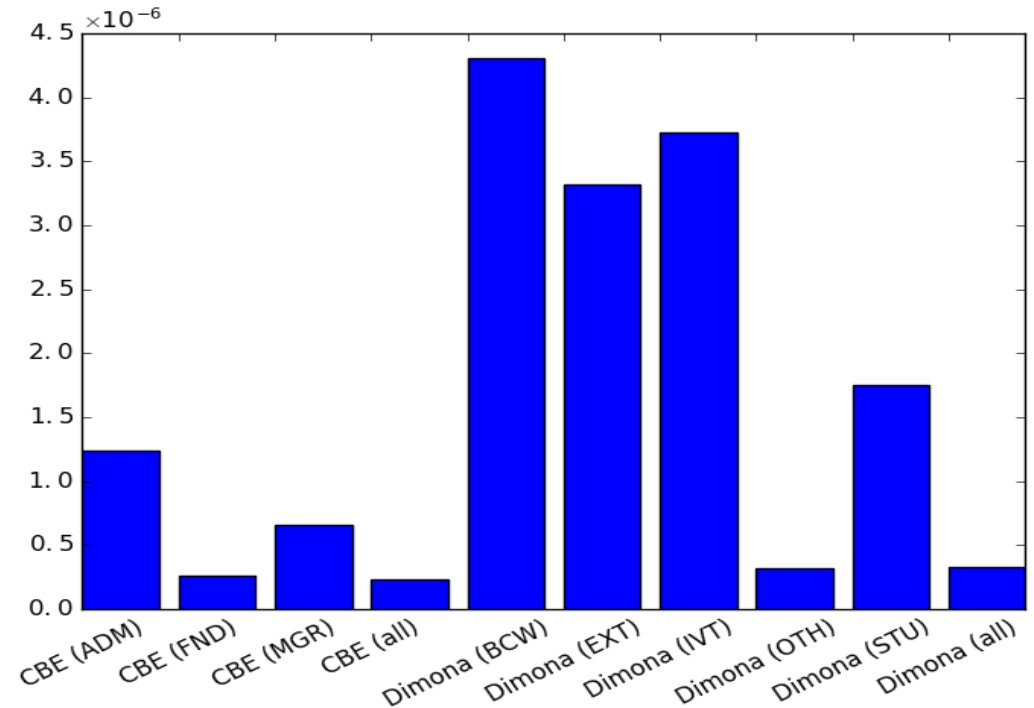
# Densité

## Dimona:

CODE	# nœuds	# arcs
ALL	6 474 928	13 722 678
BCW	332 705	476 003
IVT	209 166	163 105
OTH	5 587 049	9 950 130
STU	1 062 334	1 976 673

## BCE:

CODE	# nœuds	# arcs
ALL	3 572 876	2 933 450
ADM	923 485	1 055 779
MGR	1 087 647	781 269
FND	1 928 211	964 697



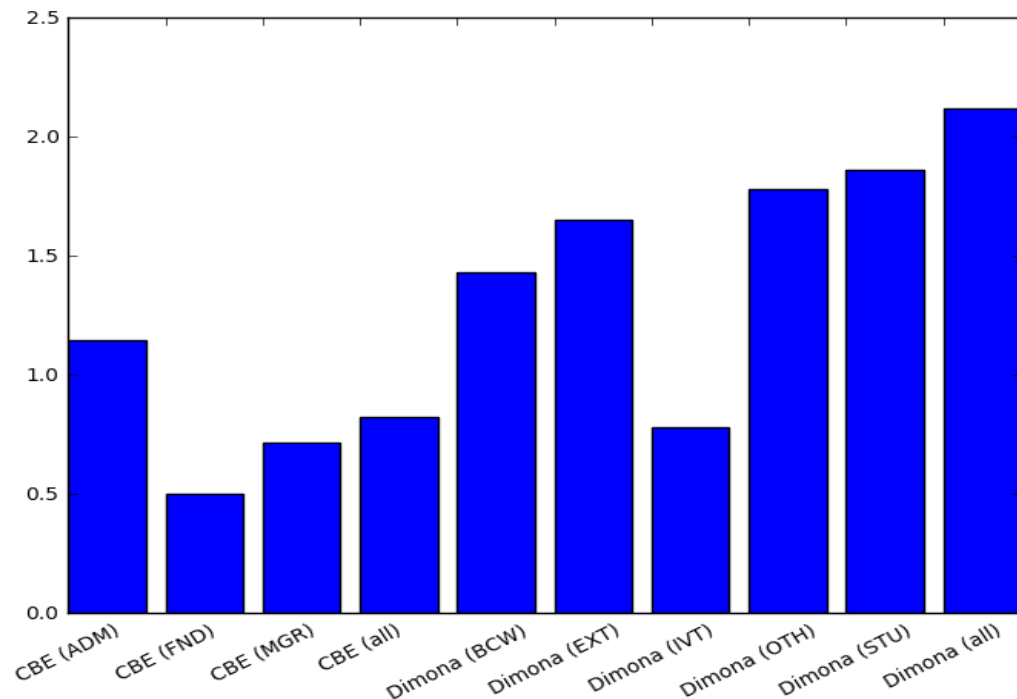
# Densité alternative

## Dimona:

CODE	# nœuds	# arcs
ALL	6 474 928	13 722 678
BCW	332 705	476 003
IVT	209 166	163 105
OTH	5 587 049	9 950 130
STU	1 062 334	1 976 673

## BCE:

CODE	# nœuds	# arcs
ALL	3 572 876	2 933 450
ADM	923 485	1 055 779
MGR	1 087 647	781 269
FND	1 928 211	964 697



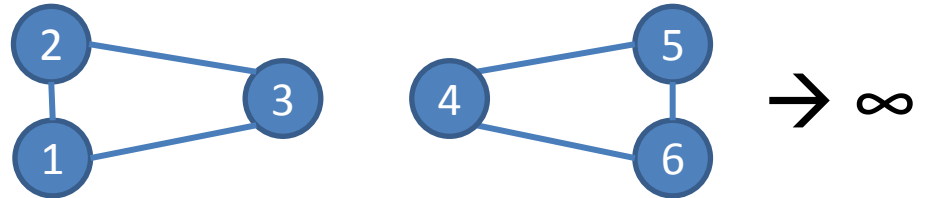
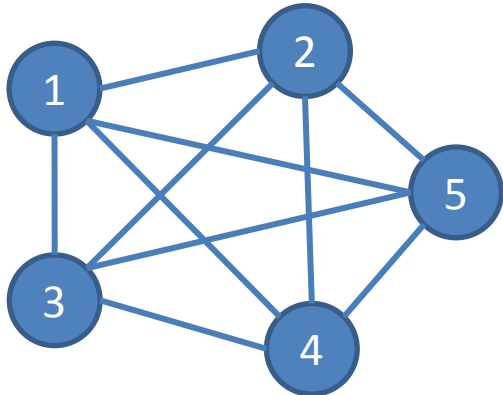
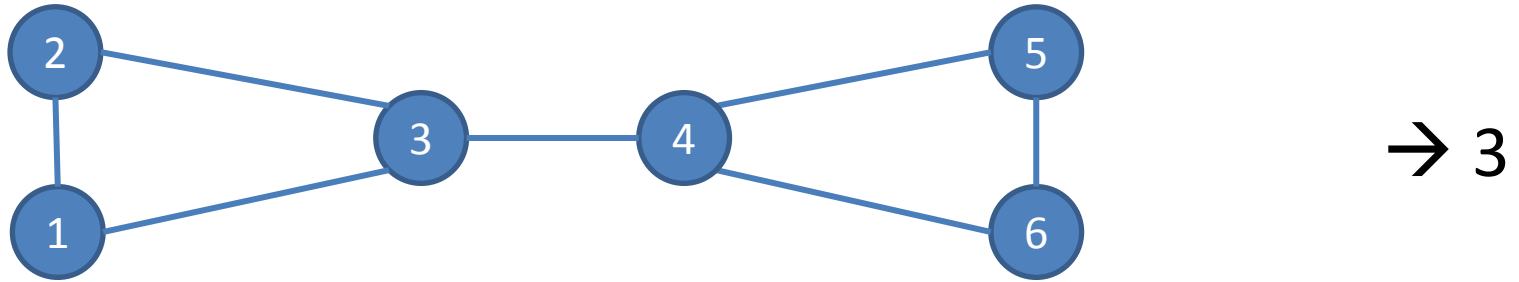
# Densité

---

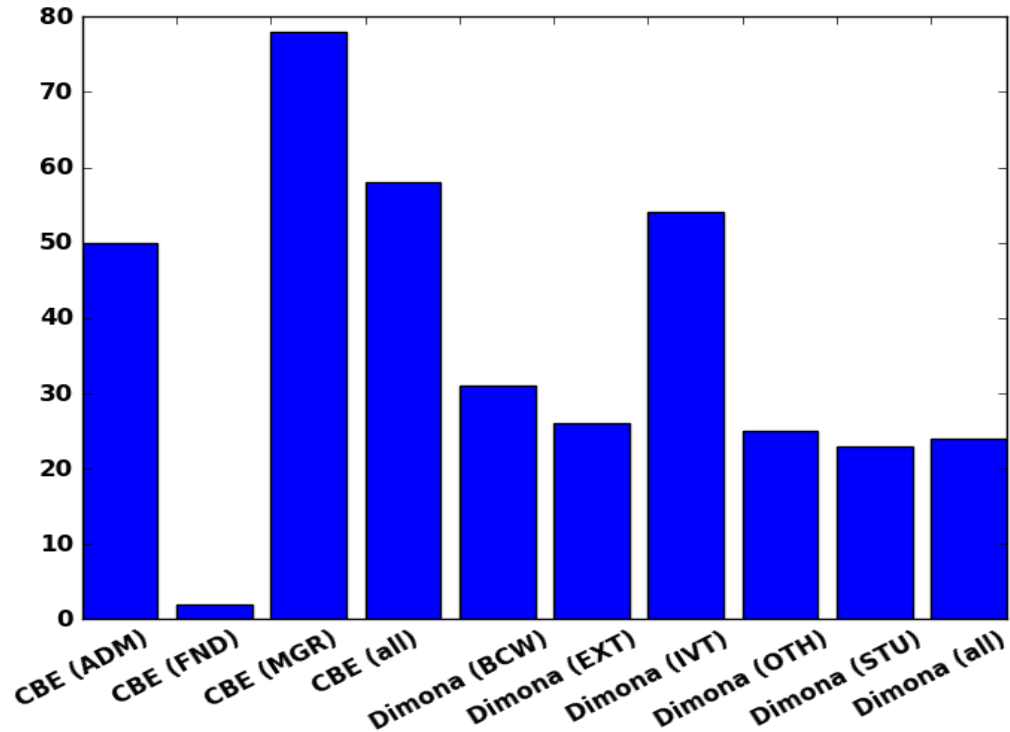
- Peut se calculer sur une **partie d'un graphe** (cluster, communauté, ...)
- Donne une idée de l'**intensité des relations**/contacts entre les nœuds
- Définition 2 : nombre **moyen** d'arcs par nœuds

# Diamètre

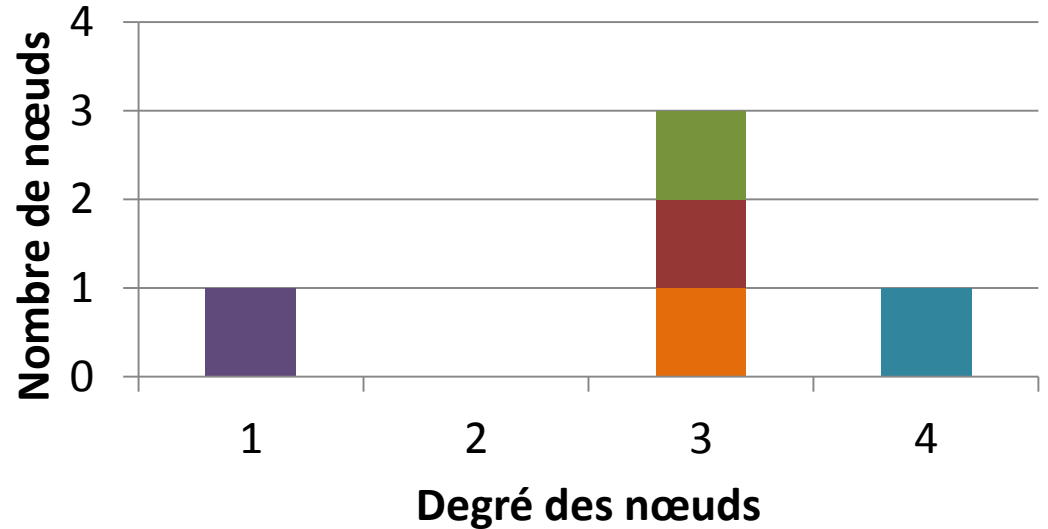
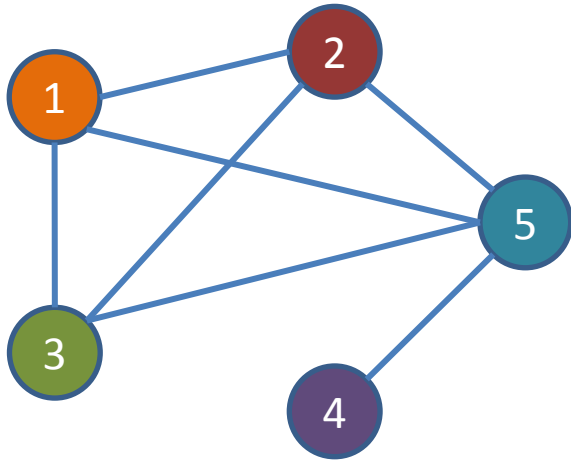
Diamètre = plus long « plus court chemin »



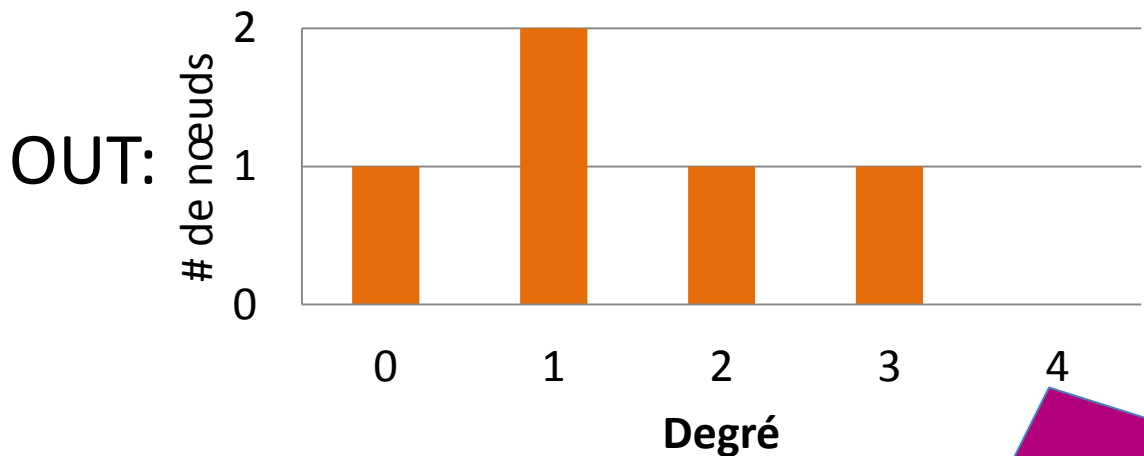
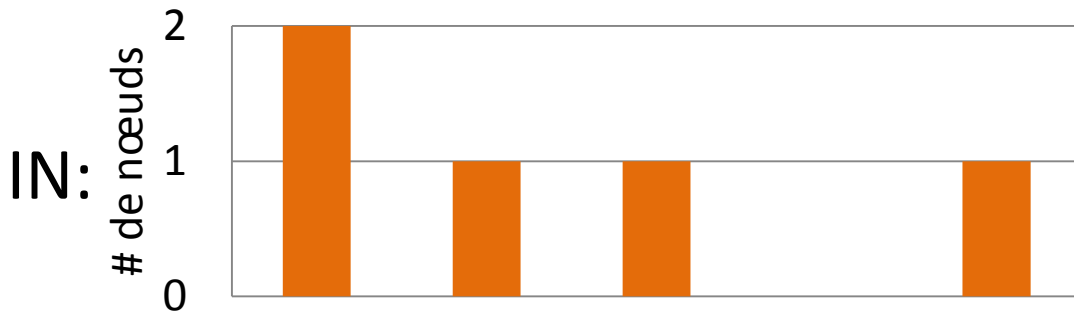
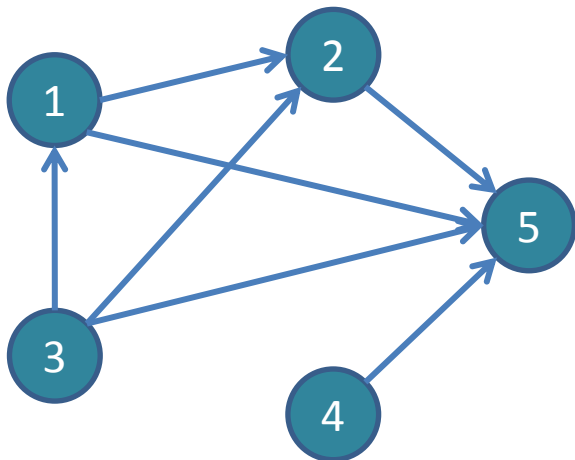
# Diamètre



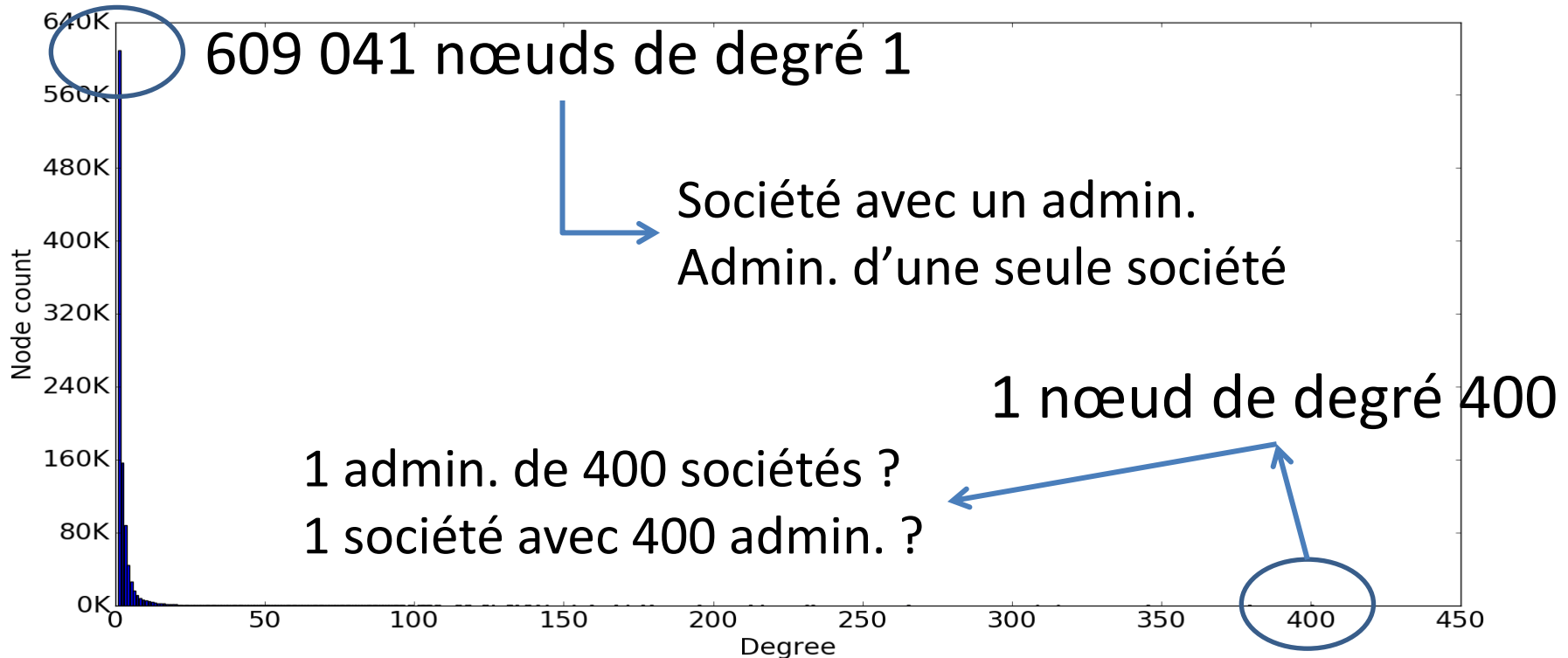
# Distribution de degrés



# Distribution de degrés (IN vs OUT)

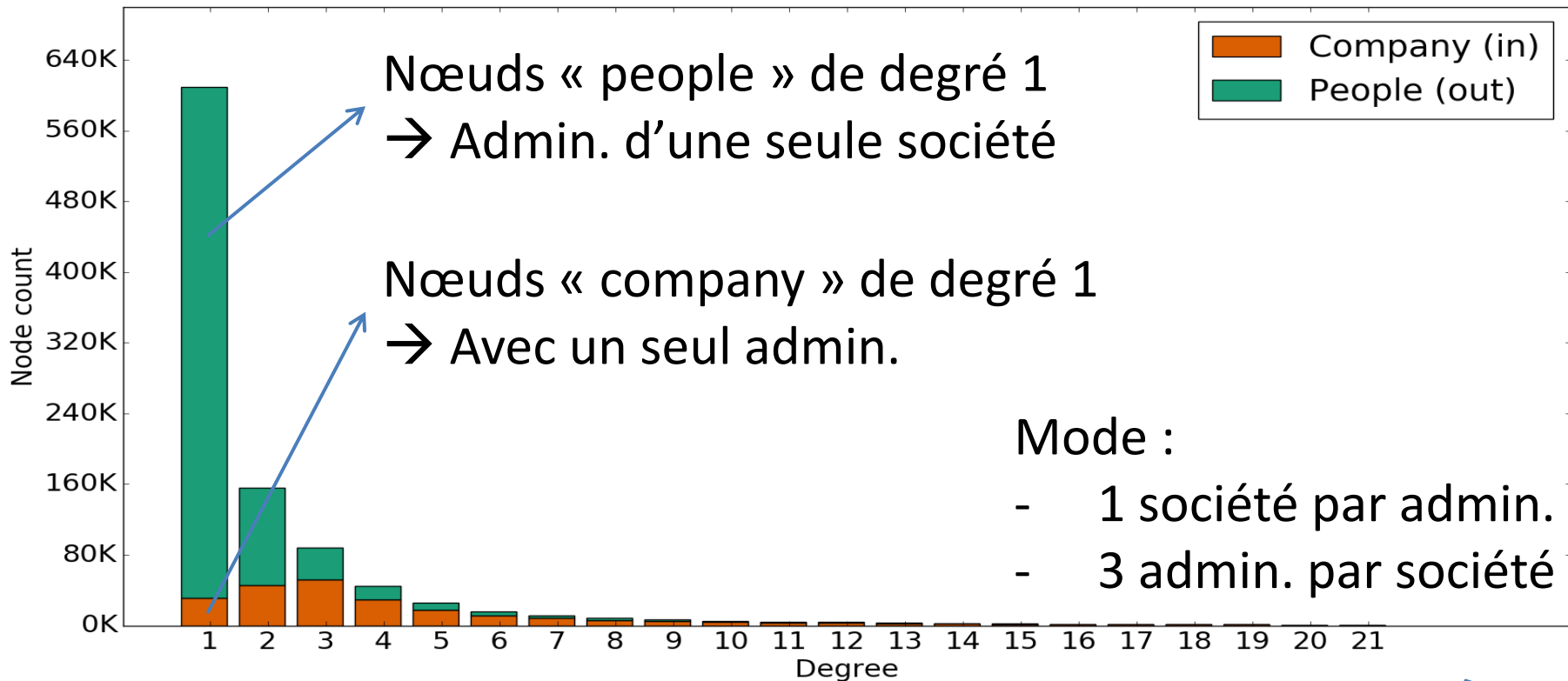


# Distribution de degrés : BCE (Adm)



Sur une période de 10 ans

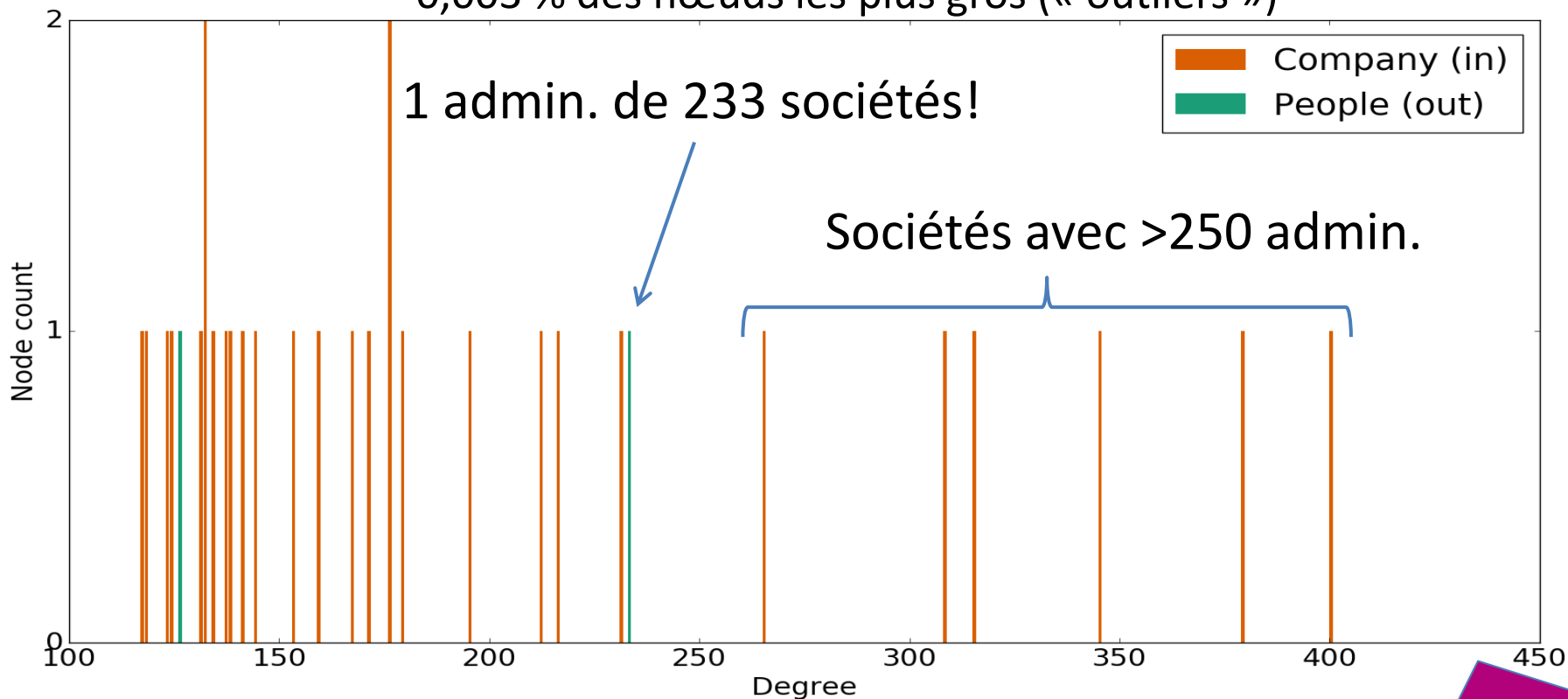
# Distribution de degrés : BCE (Adm)



Suppression de 0,5% des nœuds les plus gros

# Distribution de degrés : BCE (Adm)

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



# Métriques générales

---

- Densité : intensité des relations
- Diamètre : étalement du réseau
- Distribution de degré : « forme » du réseau

Caractériser un réseau

# CENTRALITÉ

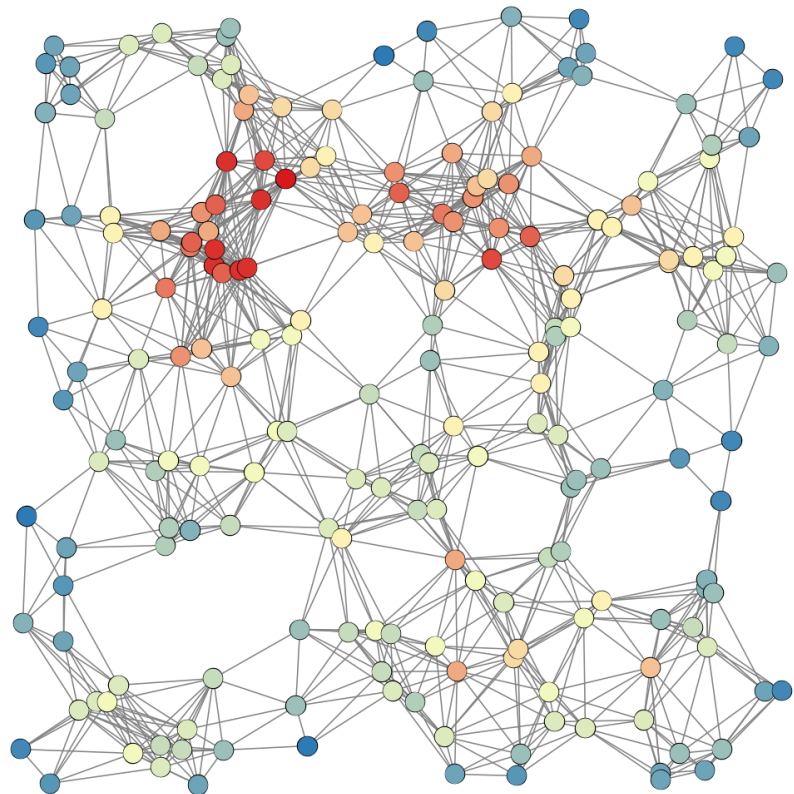
# Centralité

---

- **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...)

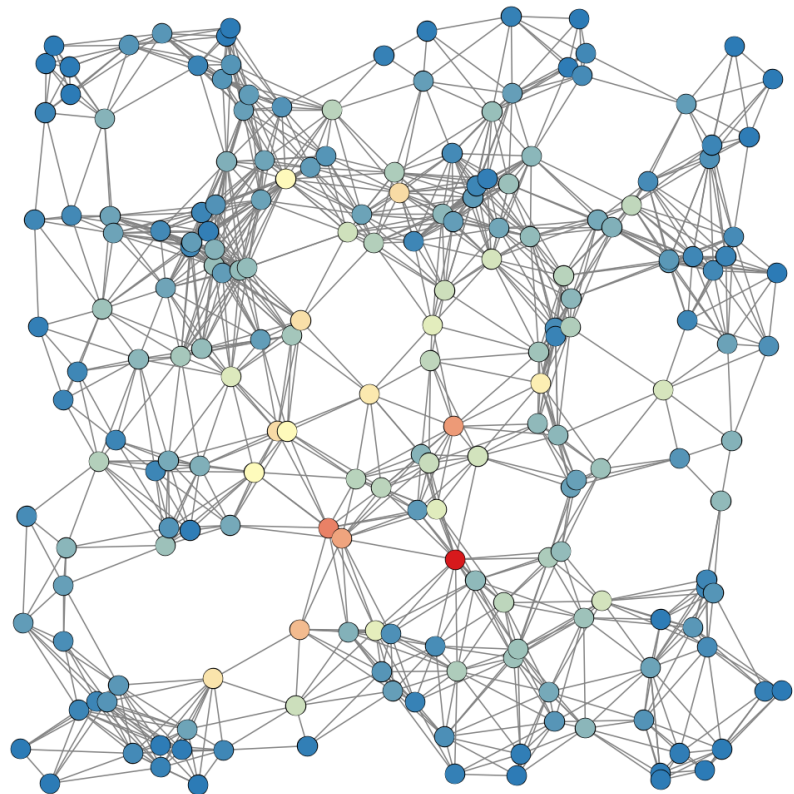
# Centralité de degré

Centralité :  
degré du  
nœud



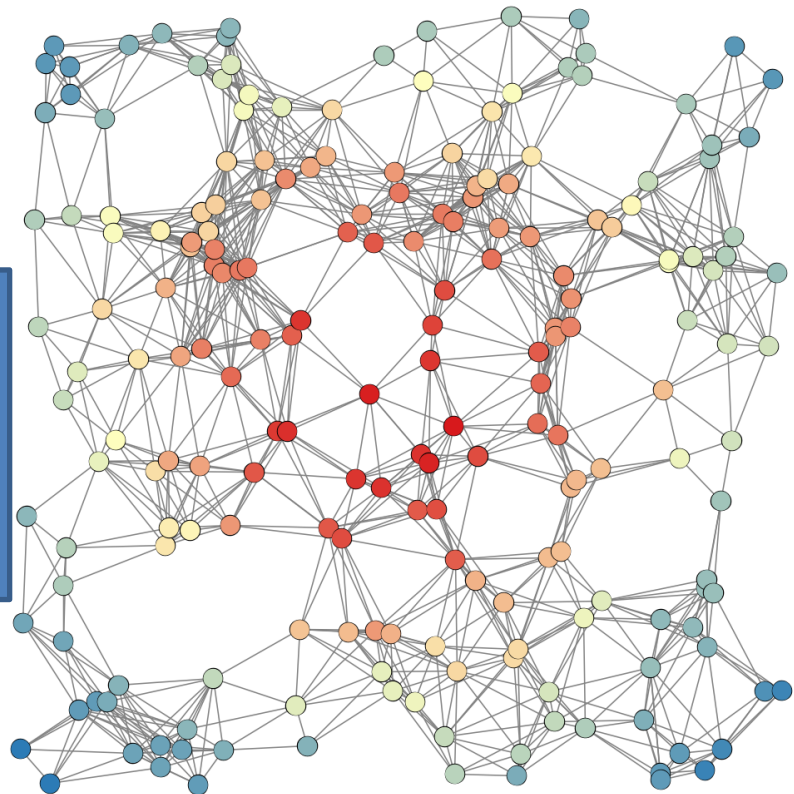
# Centralité d'intermédiarité (betweenness)

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

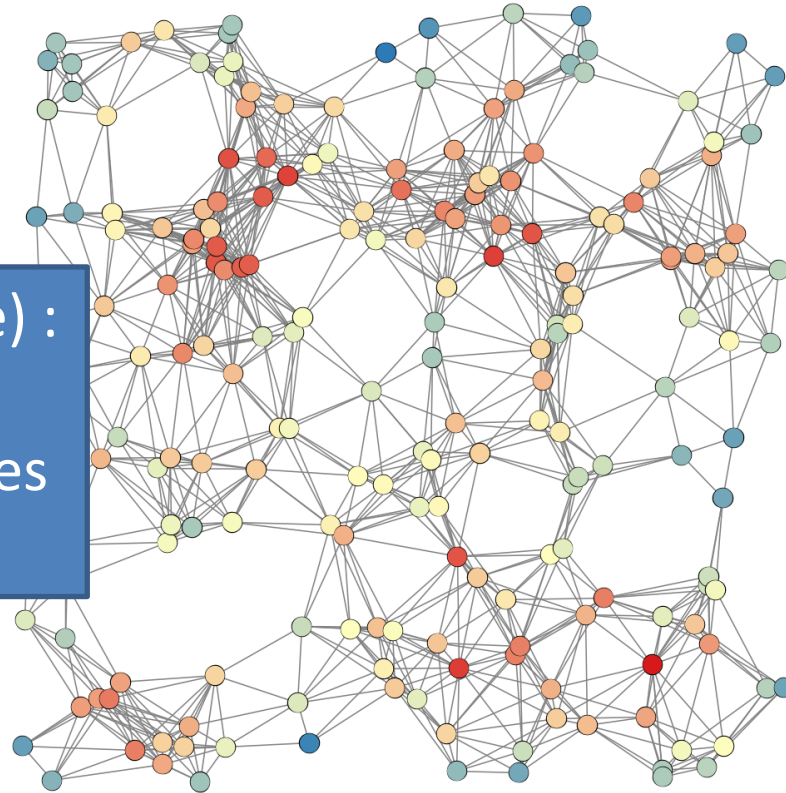


# Centralité de proximité (closeness)

Centralité :  
Distance  
moyenne de tous  
les autres nœuds



# 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

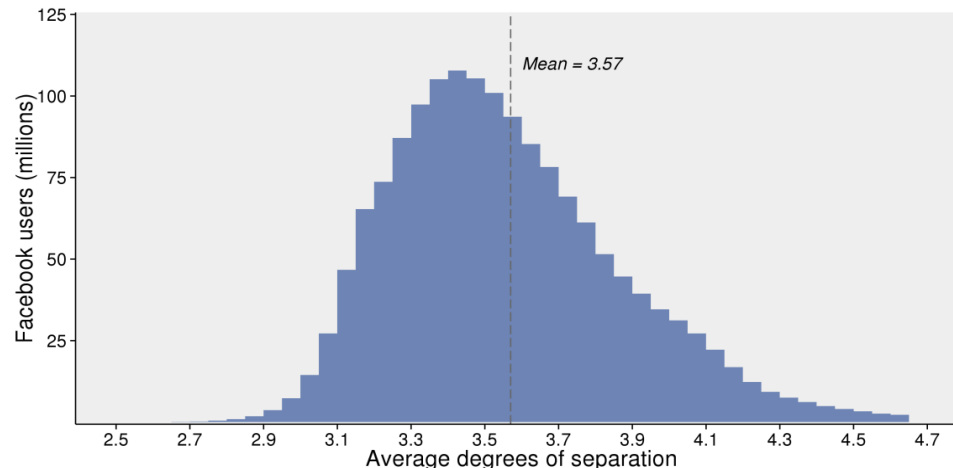
# Plus court chemin/Distance

---

- **Dijkstra** : Calcul du « plus court chemin » entre deux nœuds
- Chaque lien possède un « **poids** » (longueur, coût...)
- **Distance** =  $\sum$  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

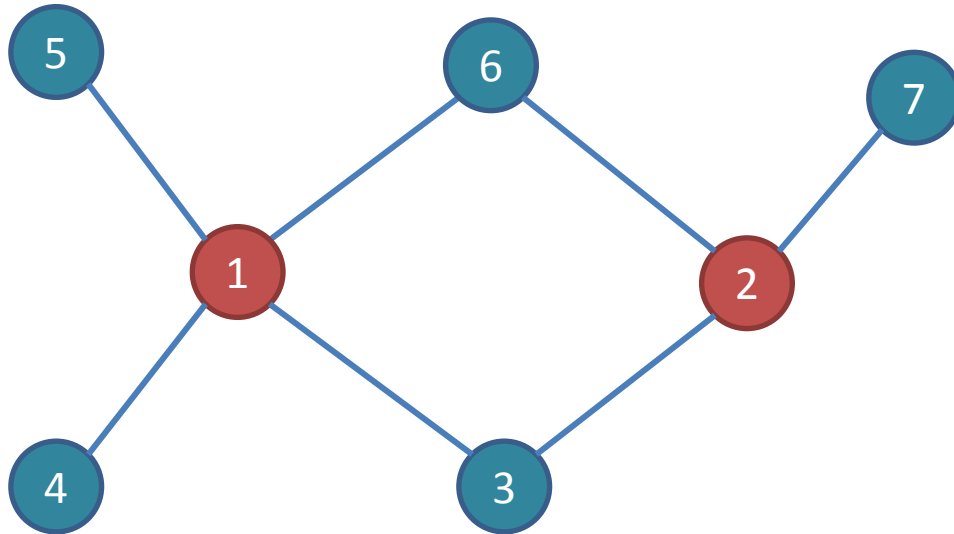
# Six degrees of separation

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



# Similarité de Jaccard

Similarité de Jaccard :



$$\frac{\# \text{ voisins communs}}{\# \text{ voisins totaux}}$$

$$2/5 = 0.4$$

# Distance

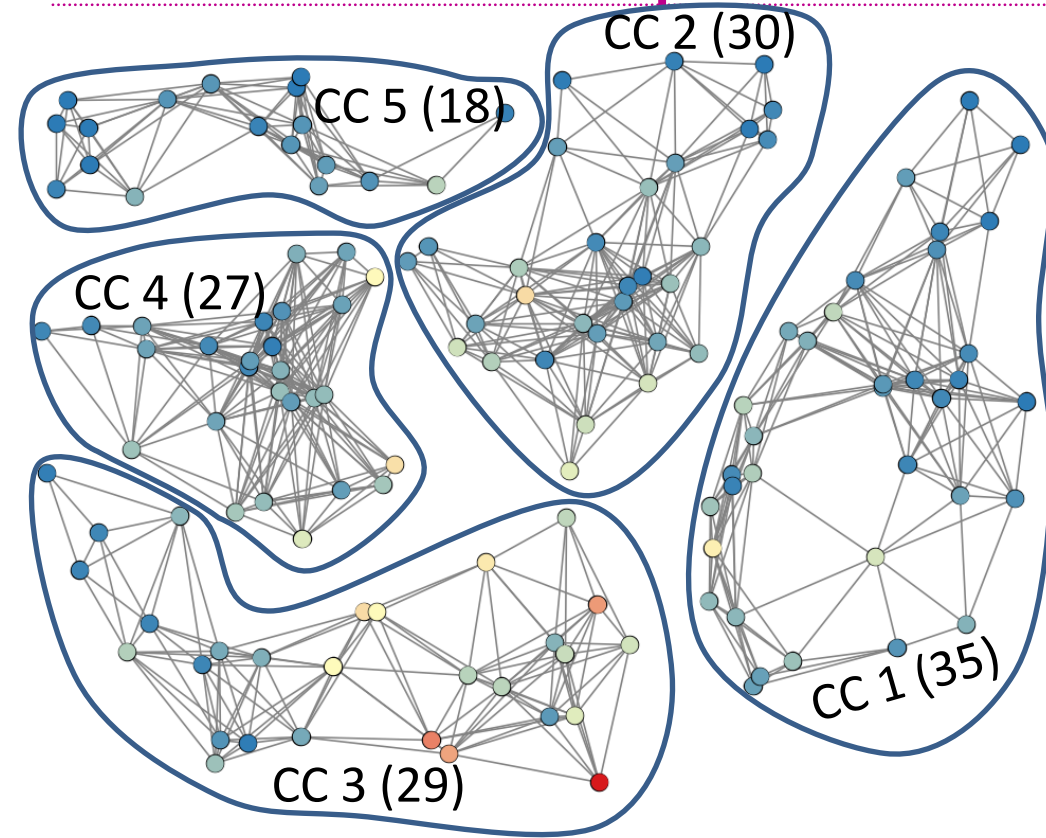
---

- Plus court chemin
- Similarité de Jaccard
  
- Hors présentation :
  - 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

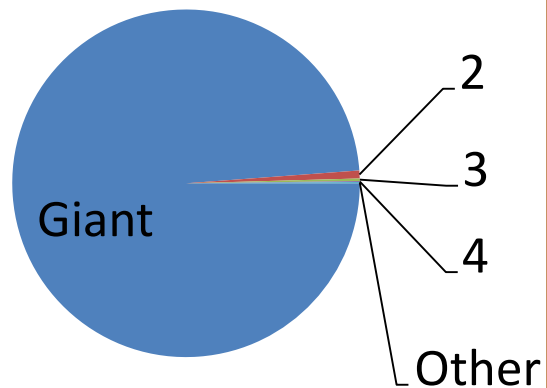
# Composante connexe

## 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%)

# Composante géante

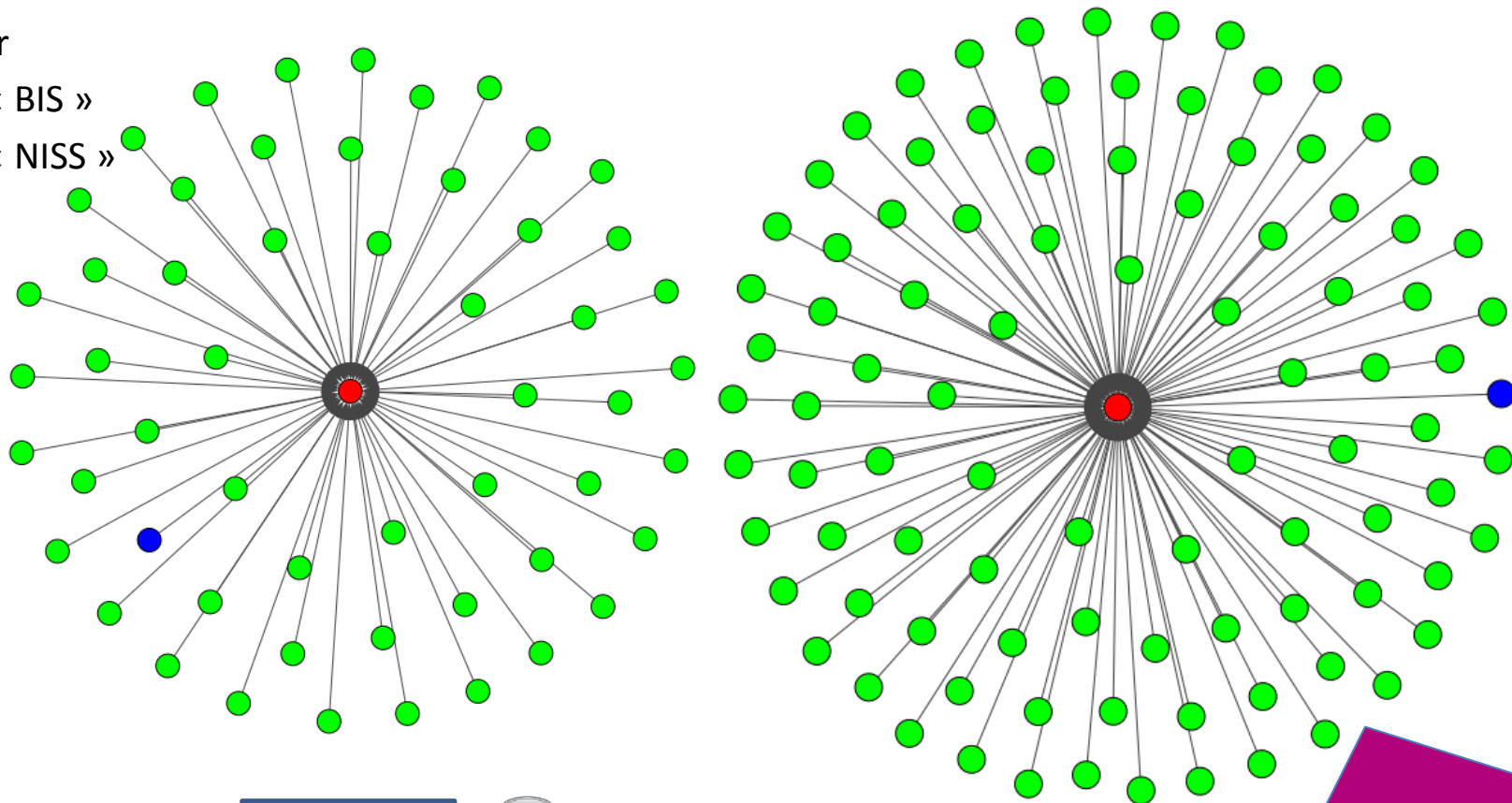
- Diamètre : 20
- Médiane : distance = 6



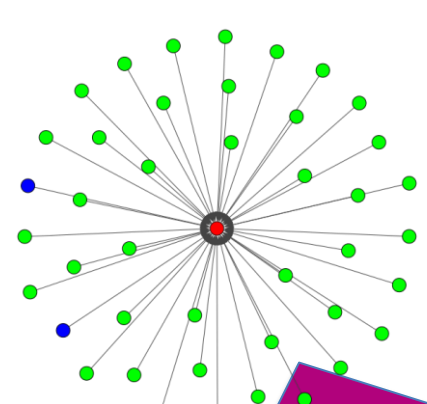
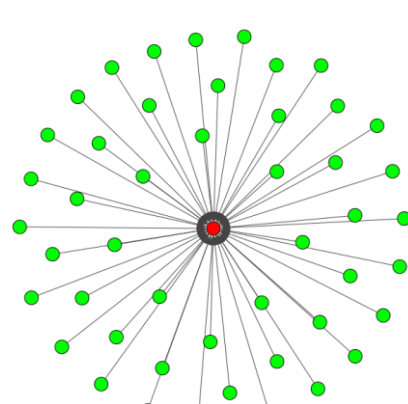
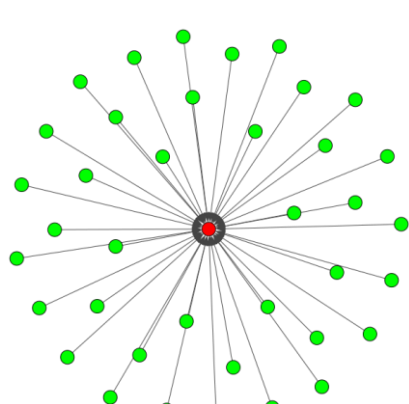
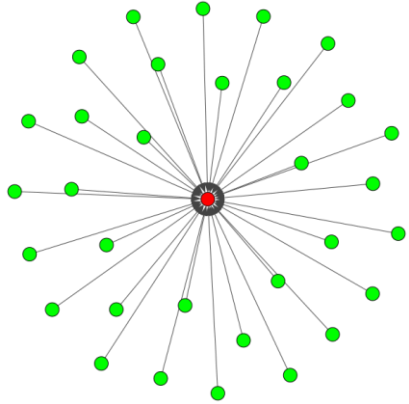
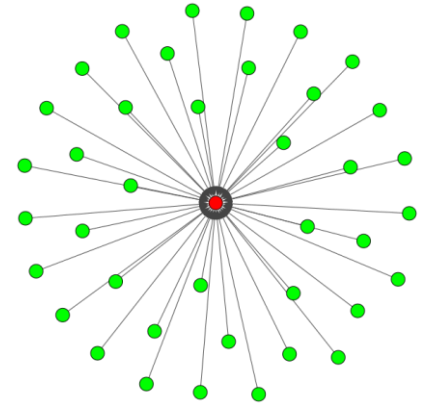
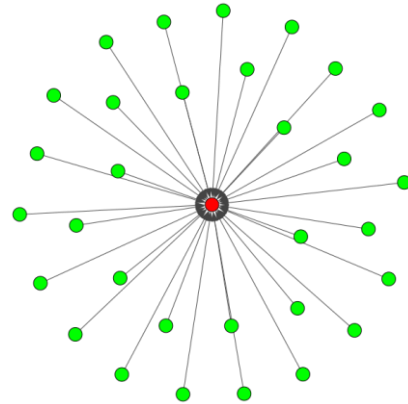
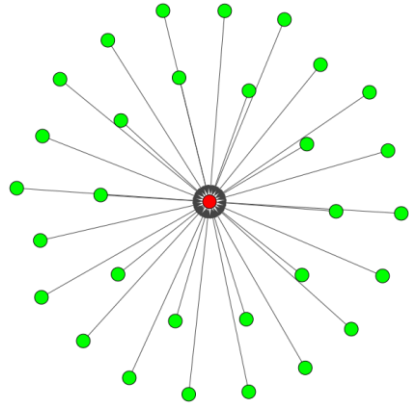
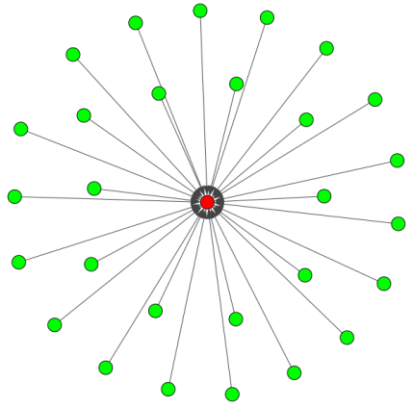
- → 50 % des « couples » sont « collègue de collègue de collègue »
- Percentile 95 : distance = 8 (+1 « de collègue »)
- Il y a des « super-connecteurs » : tous les enseignants sont collègues !

# Composante connexe : sub-giants

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

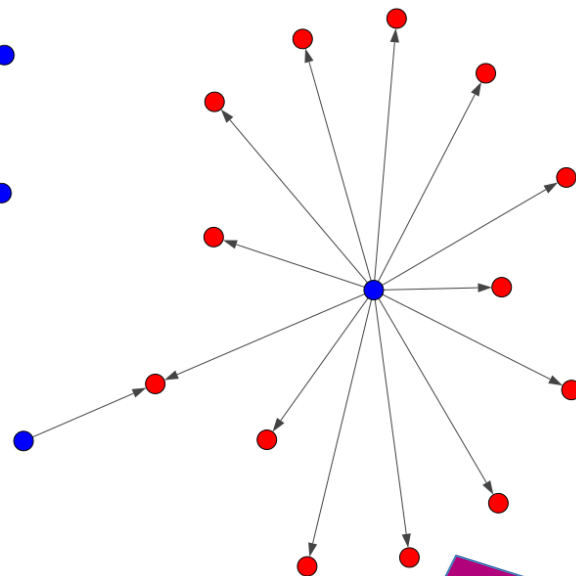
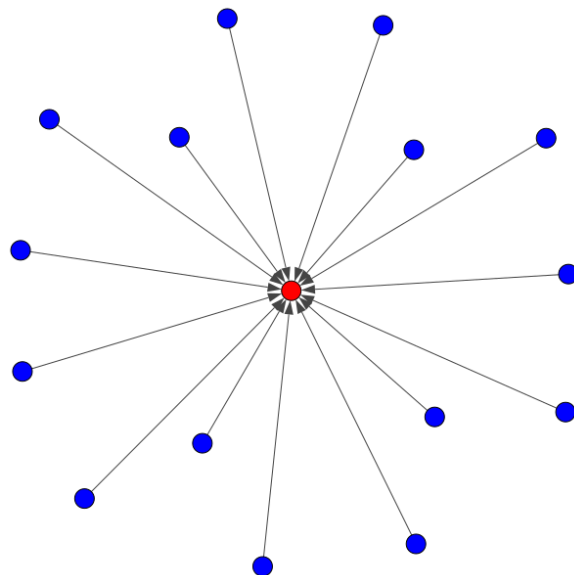
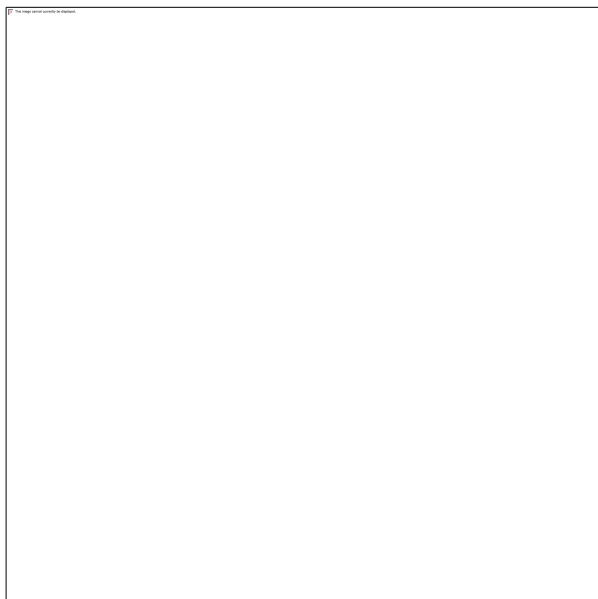


# Composante connexe : sub-giants

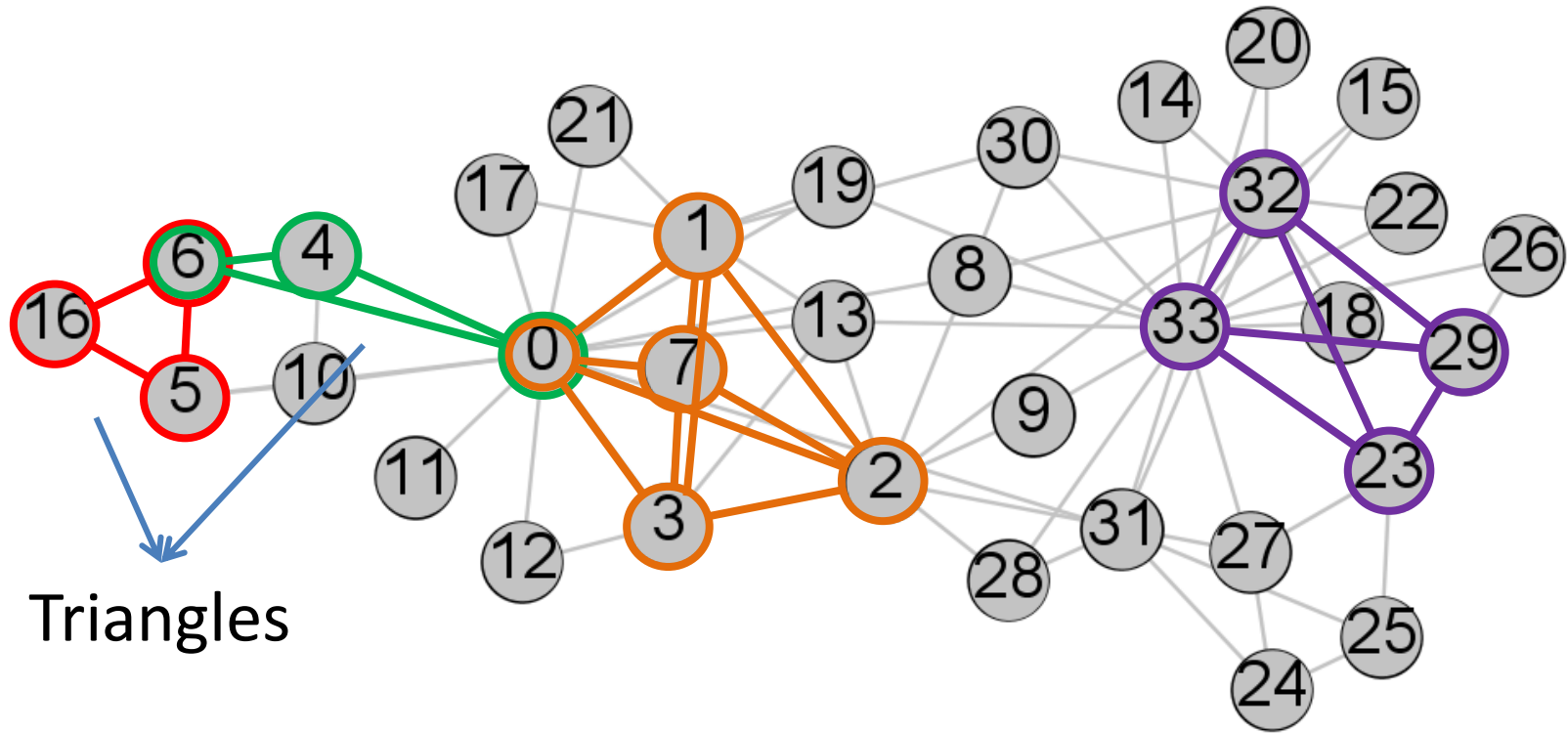


# Composante connexe : sub-giants

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



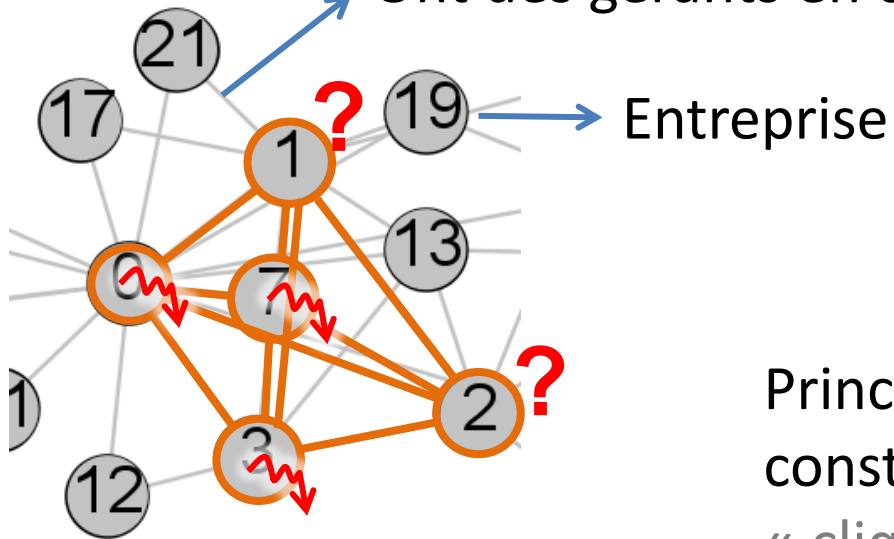
# Cliques



Triangles

# Cliques

Ont des gérants en commun

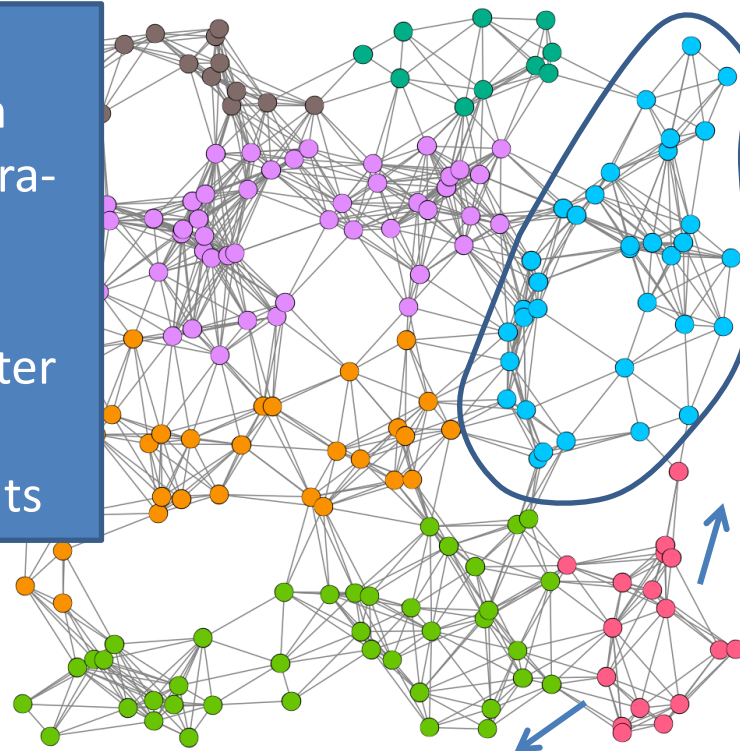
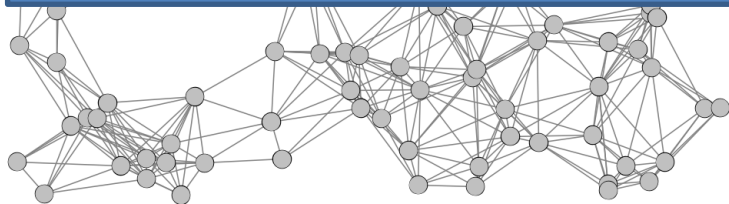


Principe utilisé pour détecter les « spider constructions » (avec une définition de « clique » un peu plus souple)

# Communautés

Coefficient de modularité :

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



Communauté

Groupe  
Module  
Cluster

Intra-cluster

Inter-cluster

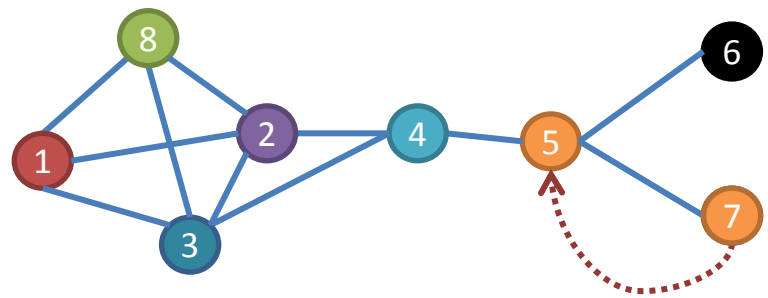
# Communautés : intérêts

---

- 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)

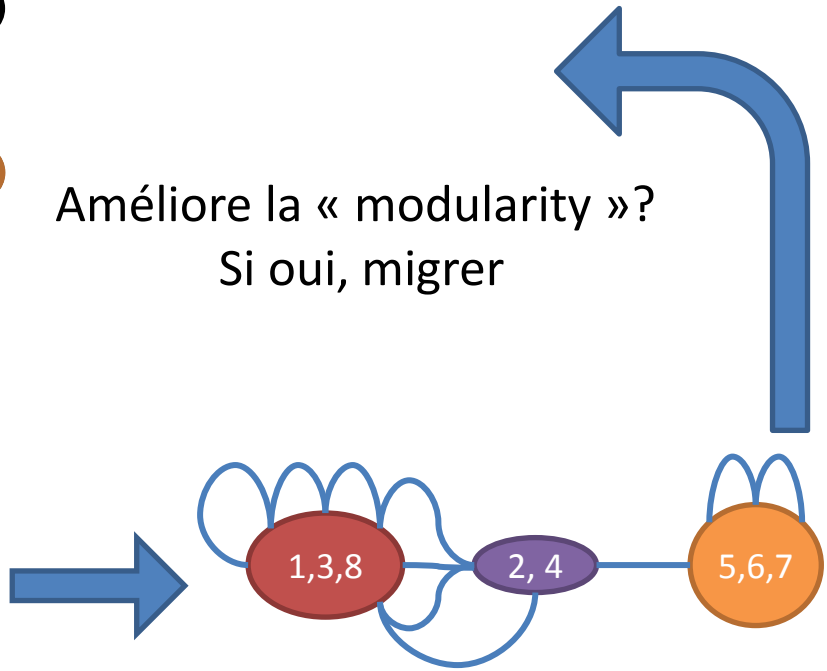
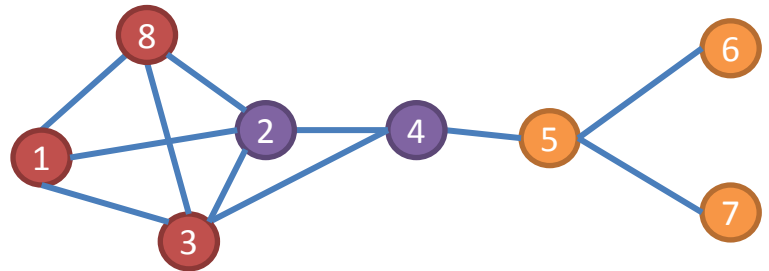
# Algorithme de Louvain (UCL, 2008)

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 »
- **Cliques** : groupes « totalement connectés »
- **Communautés** : groupes « sociaux »

Hors présentation : notion de « fortement » ou « faiblement » connecté si on tient compte des réseaux dirigés

Caractériser un réseau

# HOMOPHILIE

# Homophilie / Assortativité

---

- **Homophilie / Assortativité** : formalisation du « **qui se ressemble s'assemble** »
- Réseau « homophile » : les **nœuds similaires** sont **mieux liés** que les nœuds différents
- En détection de fraude/criminalité : plus de chances de trouver des fraudeurs dans le **voisinage** d'un fraudeur que n'importe où

# Questions ?

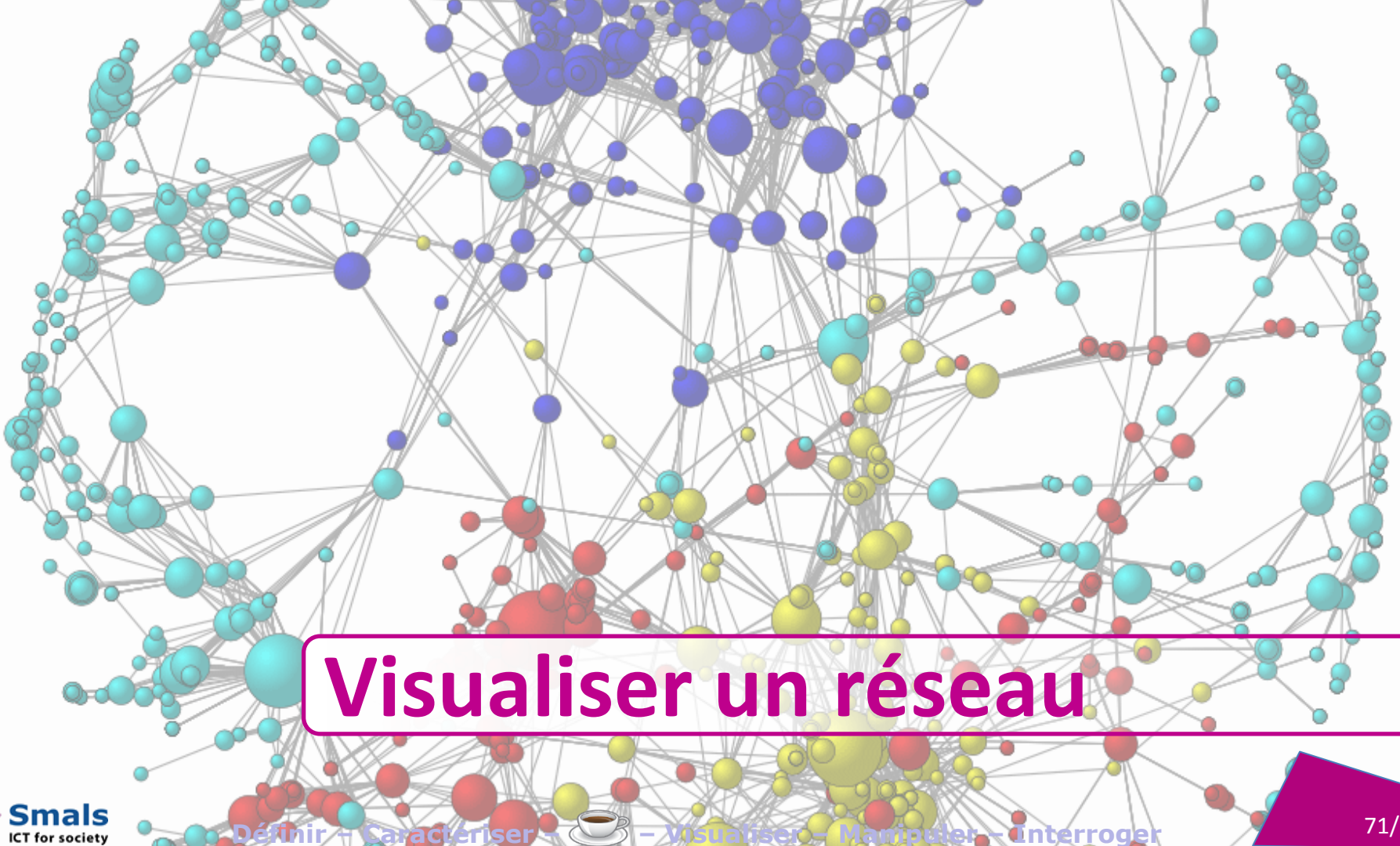
---



# Pause !

---

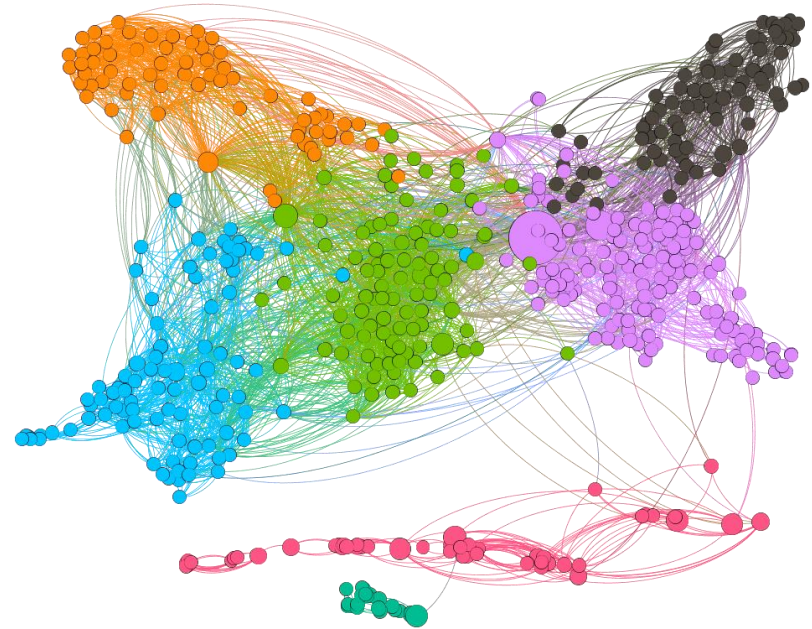
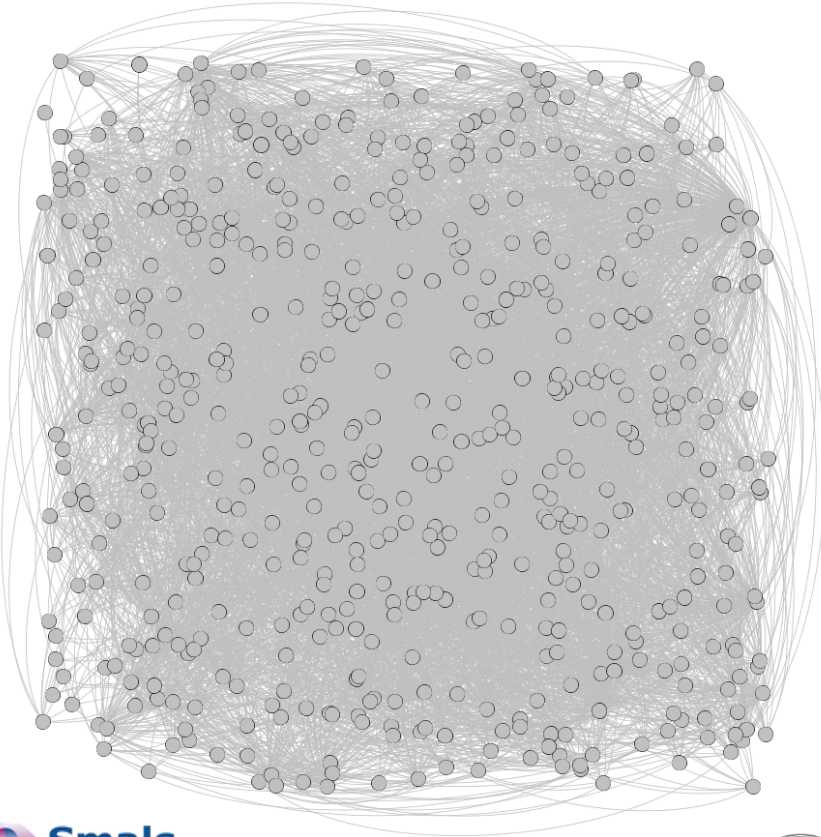




# Visualiser un réseau



# Visualiser

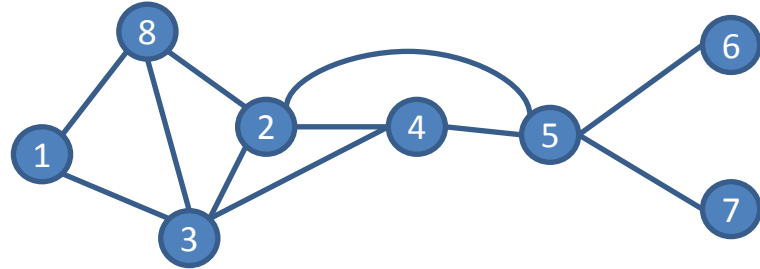


Visualiser un réseau

# 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

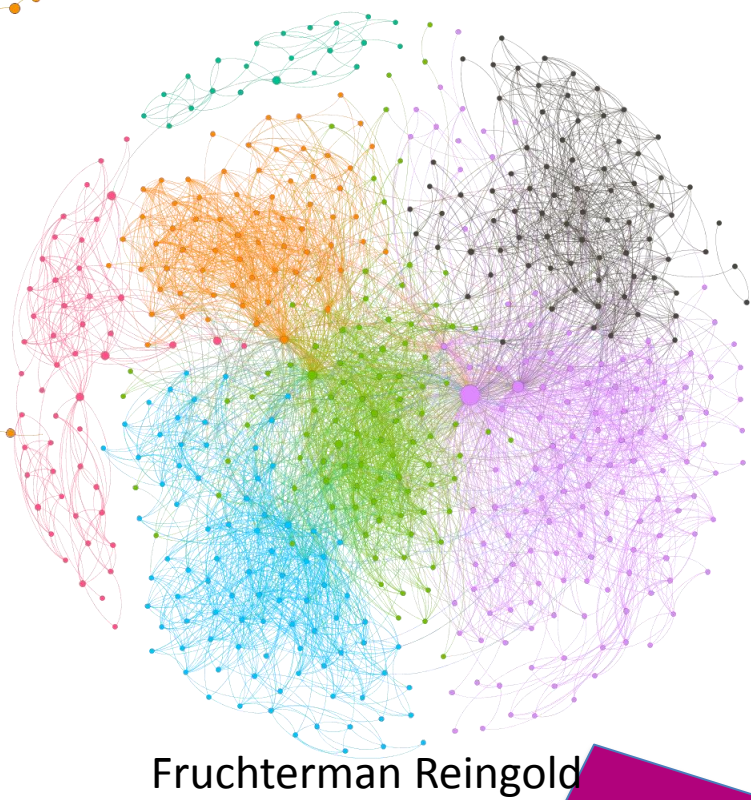
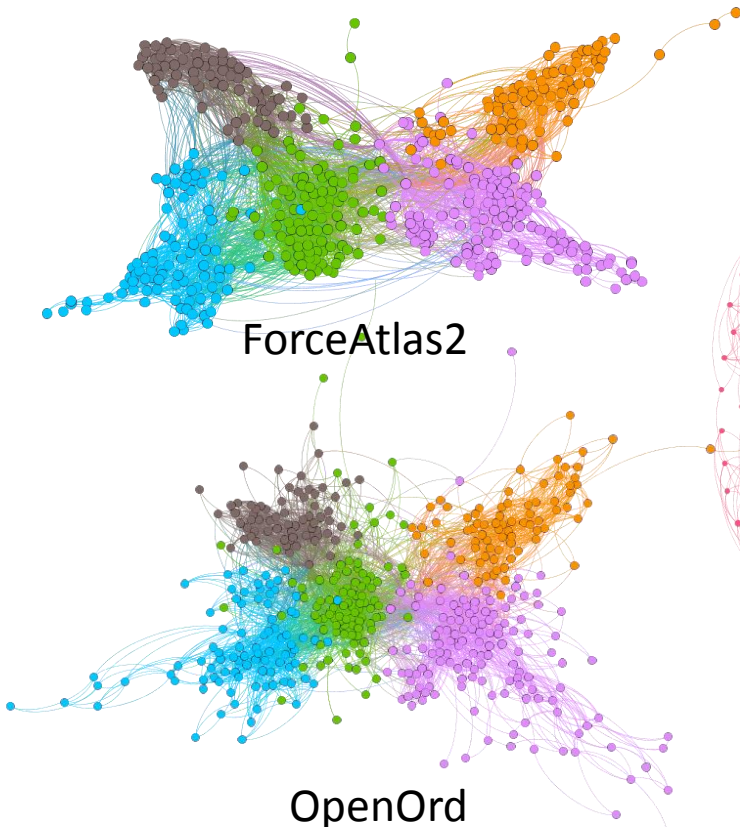
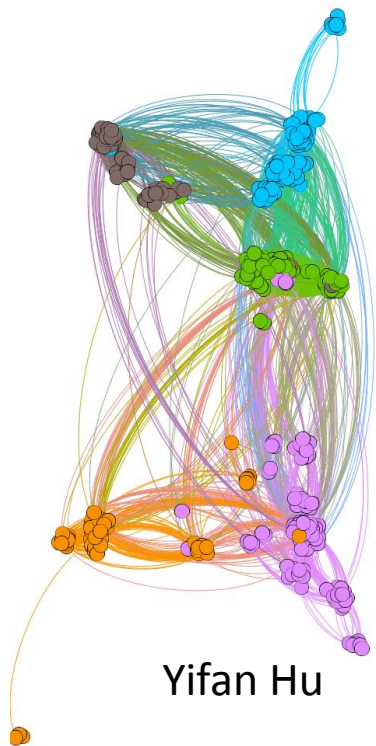


# Spatialisation

---

- **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

---

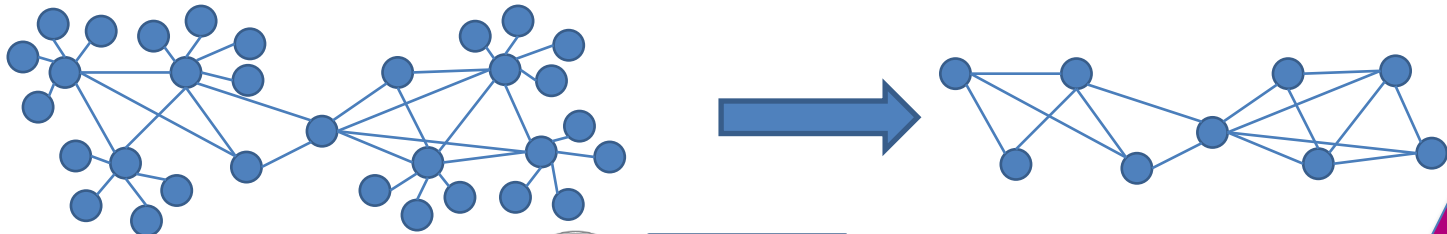
- Composantes connexes  
→ Méthode naturelle, pas de perte
- Communautés  
→ On perd les relations inter-communauté

# Filterer 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 »

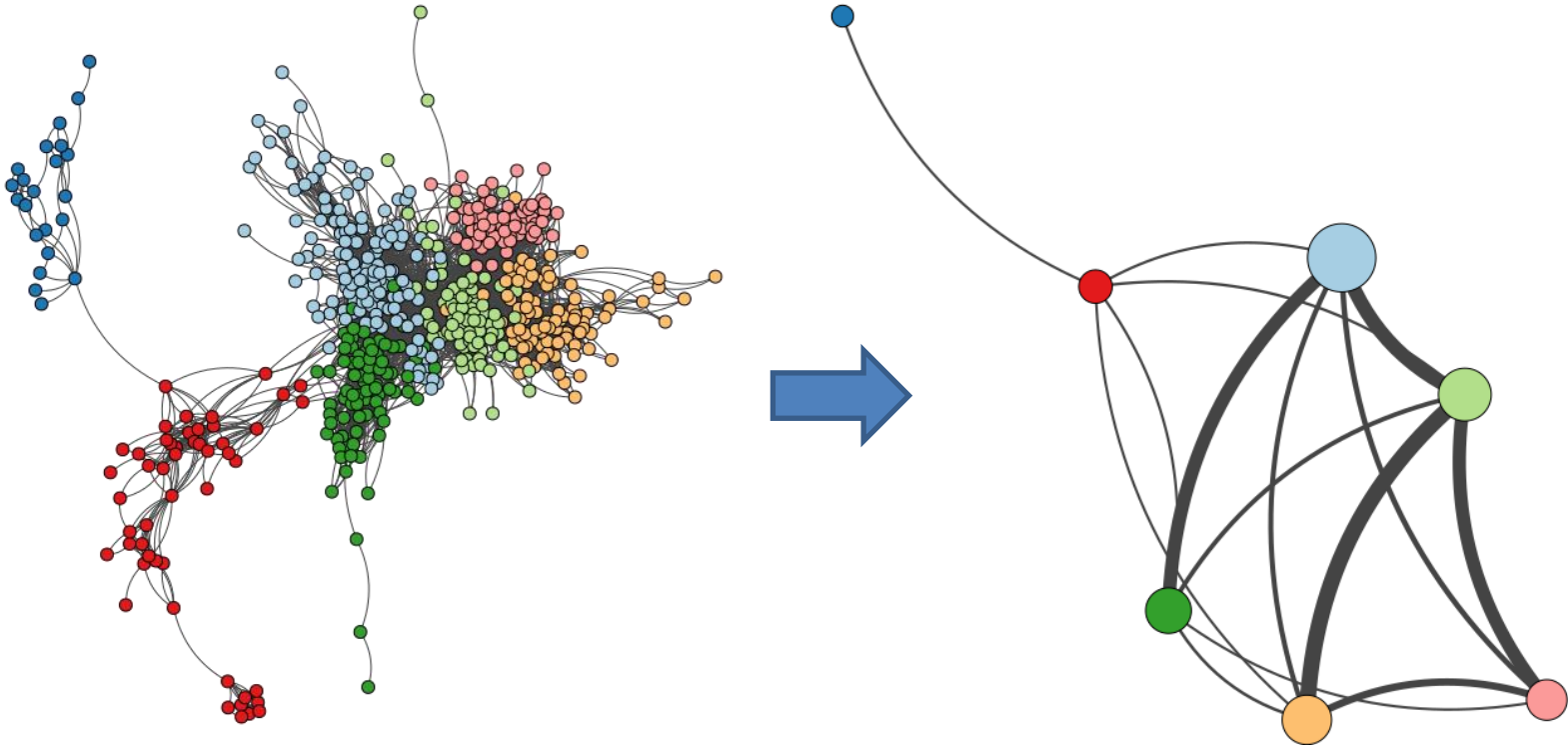


# Fusionner des nœuds

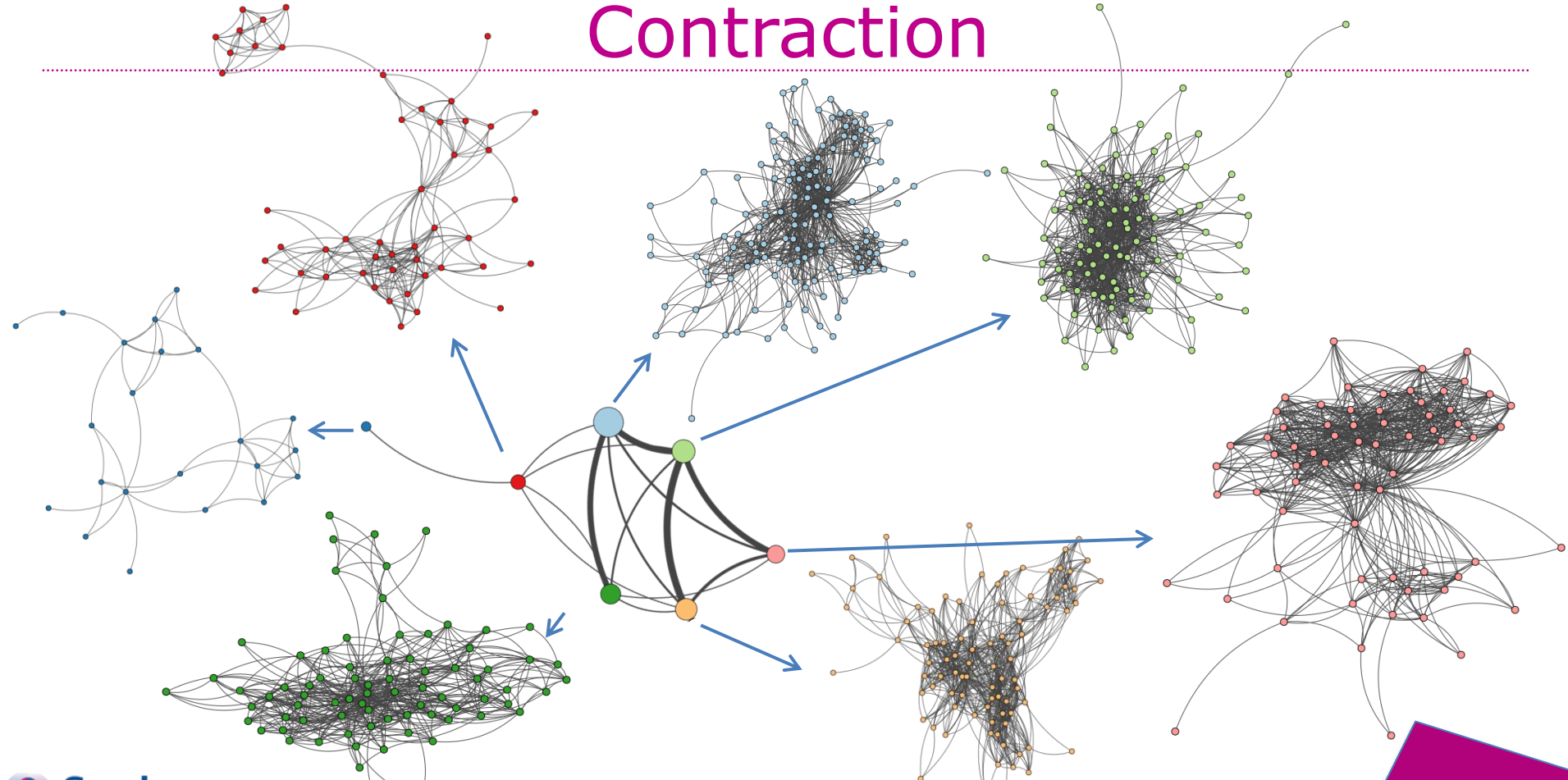
---

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

# Contraction



# Contraction

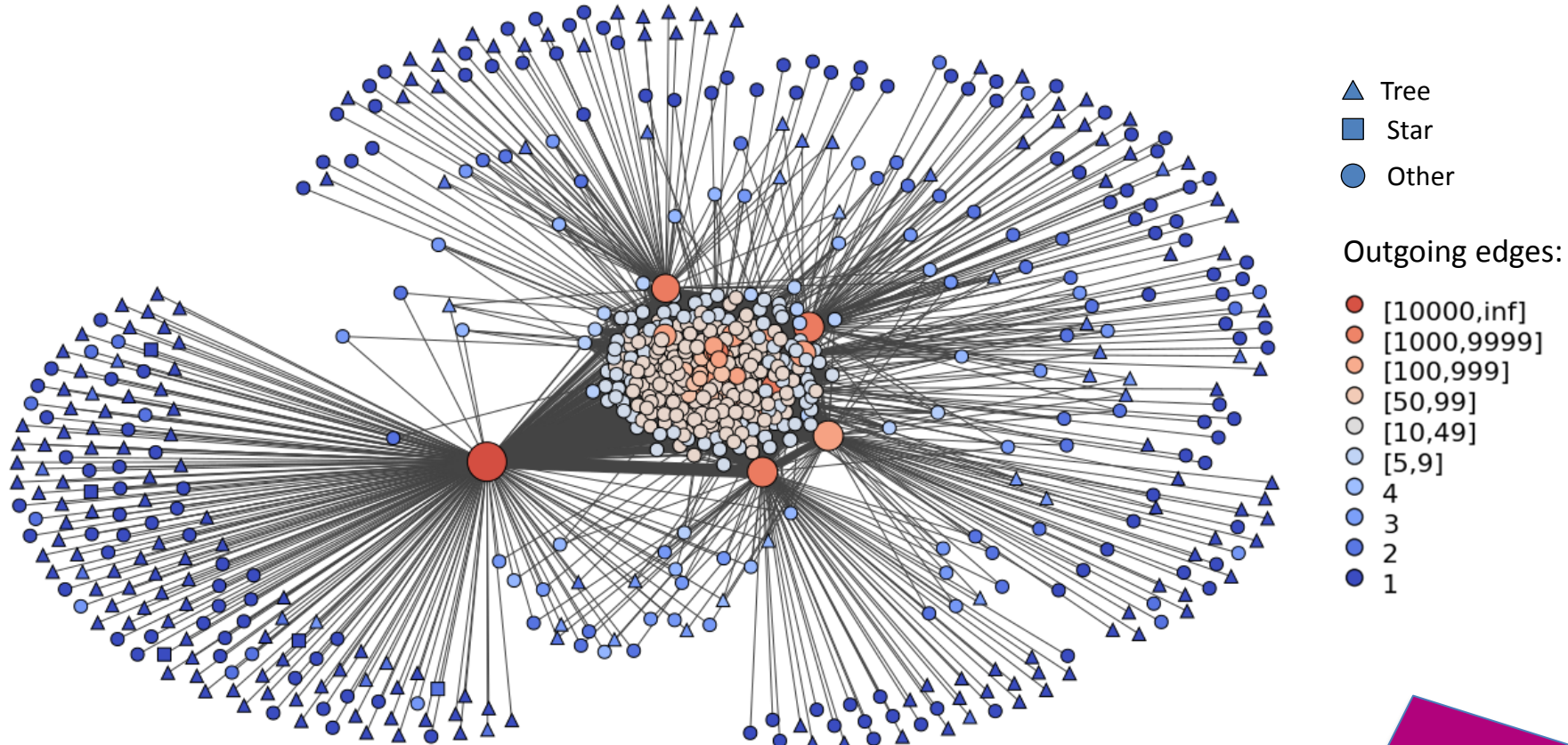


# Contraction : BCE (Administrateurs)

---

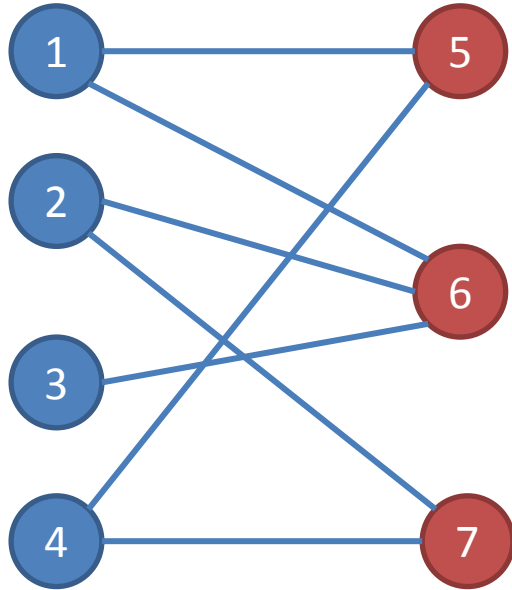
- Administrateurs BCE :
  - Composante géante = 655.158 nœuds (sur  $\sim 10^6$ )
- On applique Louvain :
  - 743 communautés ( $\sim 10''$  en mono-thread)

# Contraction : BCE (Administrateurs)

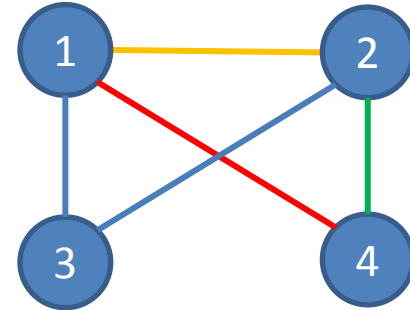


# Projection bipartite

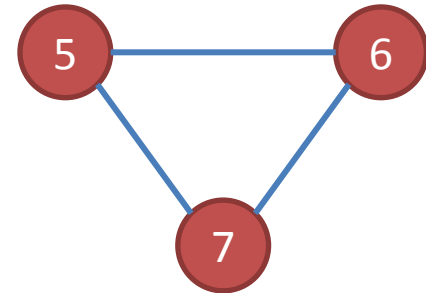
Travailleur      Employeurs



Collègues



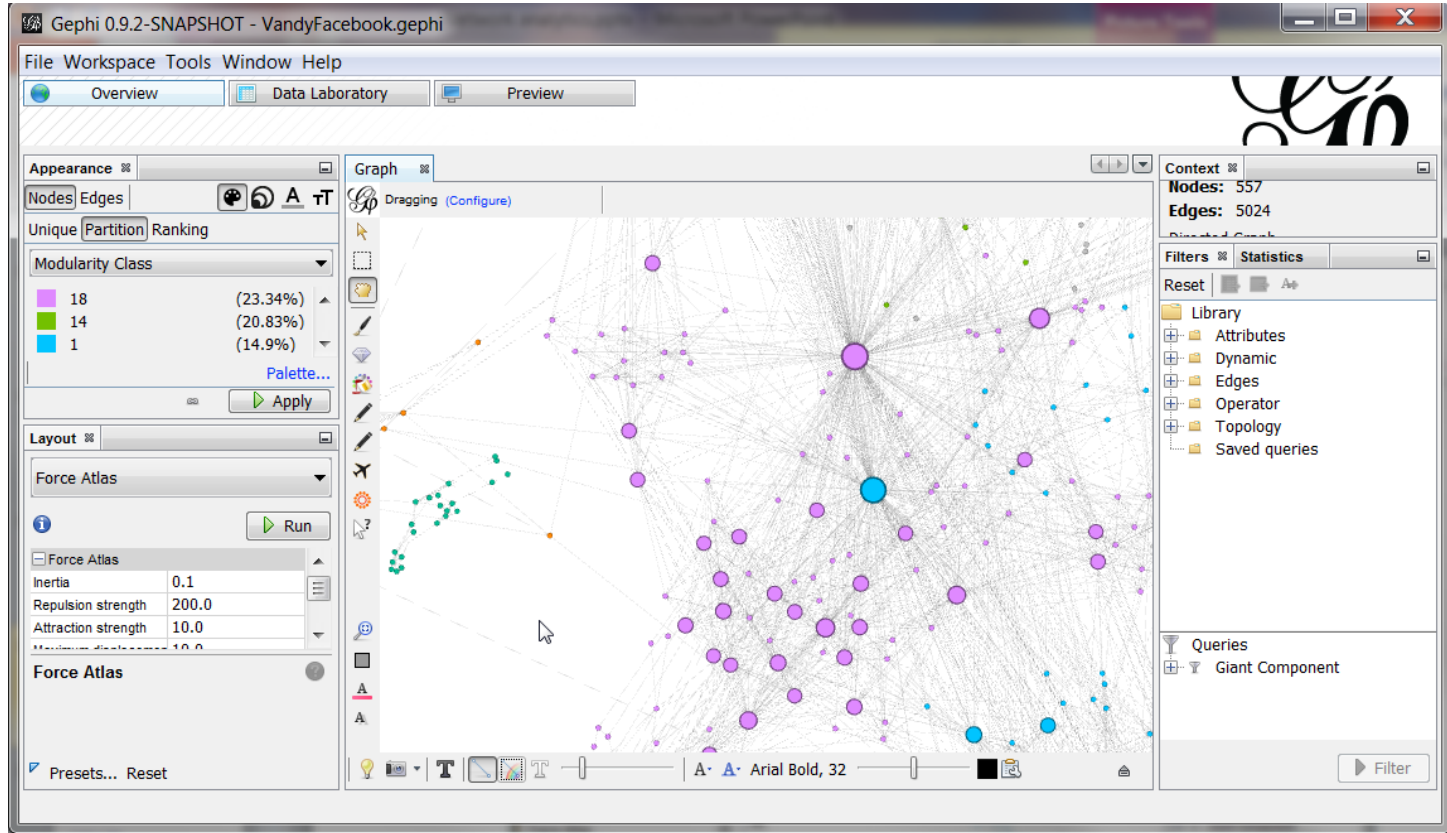
Travailleurs communs



Visualiser un réseau

# OUTIL DESKTOP : GEPHI

# Gephi



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>
```

Visualiser un réseau

# OUTIL WEB : LINKURIOUS



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

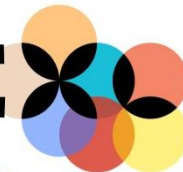


- SAS Network Analytics €
- IBM i2 €



**MALTEGO**

**NODE**



# 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) €



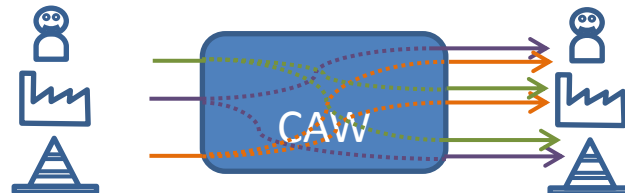
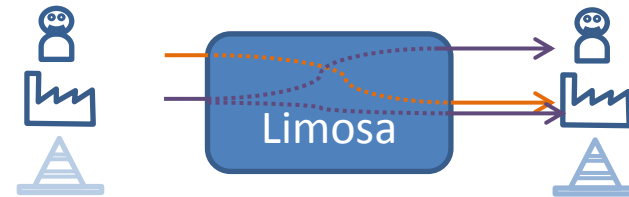
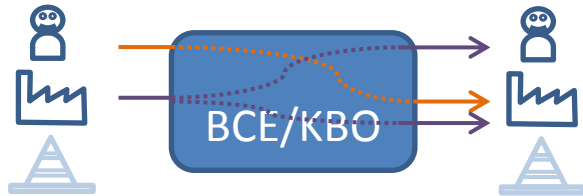
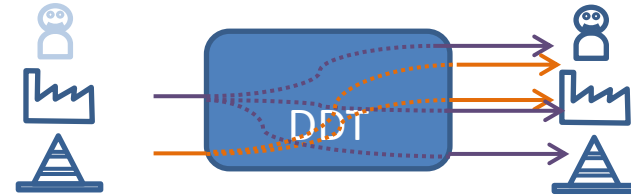
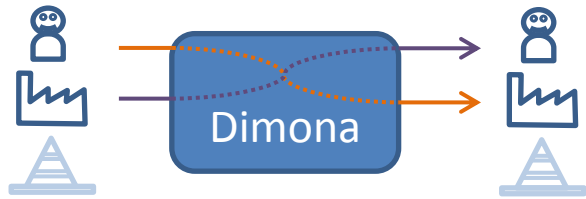
Visualiser un réseau

# USE CASE : SOCDUMPV2

# SocDumpv2 : Contexte

- Pour différents services d'inspection, on veut **visualiser le réseau** d'un groupe de personnes ou entreprises (suspectées de *dumping social* ou autre fraude)
- À partir de données de (principalement) :
  - **Dimona** : relations employeurs – employés (en Belgique)
  - **BCE/KBO** : relations sociétés – mandataires
  - **Limosa** : Travailleurs étrangers travaillant en Belgique
  - **DDT** : Sociétés – Chantiers – Sous-traitants
  - **CAW** : Chantiers – Travailleurs – Sociétés

# SocDumpv2 : modules



# SocDumpv2 : modules

---

- Chaque module produit 2 types d'outputs :
  - Des nœuds : personnes, entreprises, POW
  - Des relations
- Les nœuds d'outputs de chaque module peuvent servir d'input pour un autre
- Exemple :

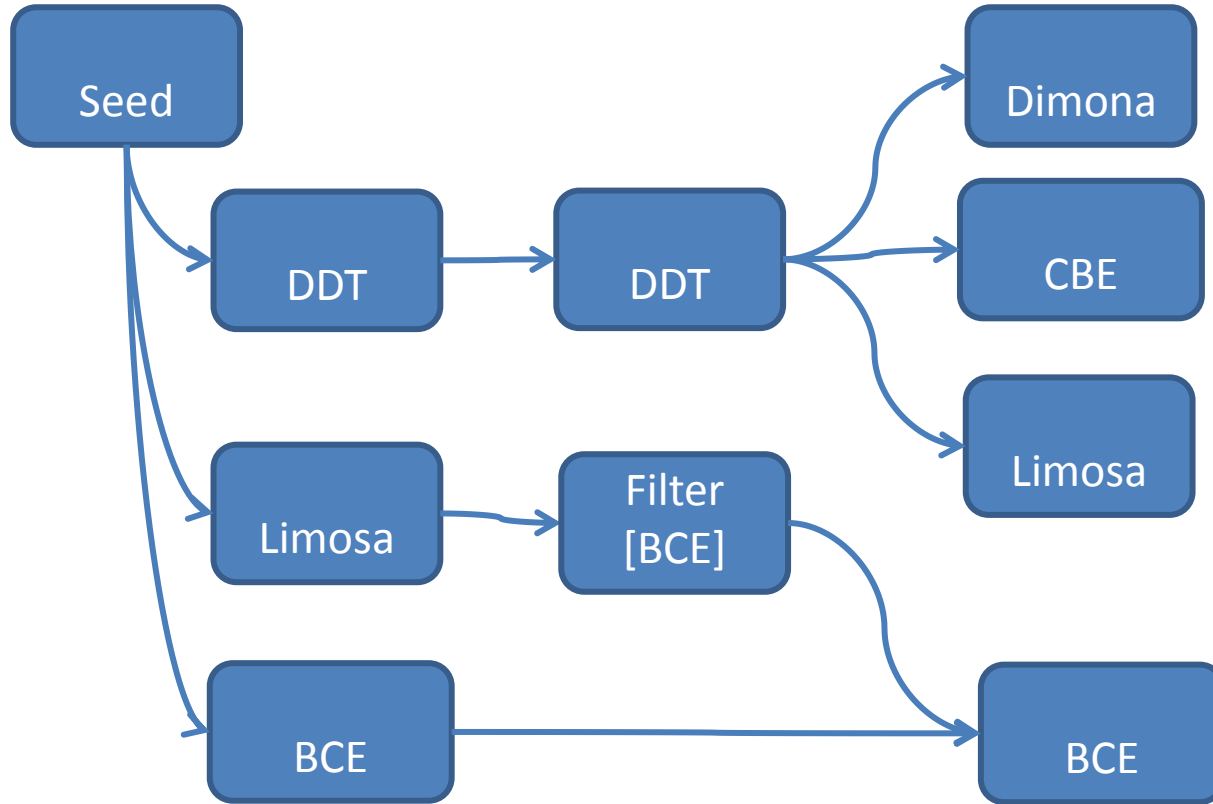
NISS –[BCE]-> Entreprises –[Dimona]-> Employés

# SocDumpv2 : qualité

---

- La **qualité** n'est pas toujours au rendez-vous !
- Une entreprise peut être désignée soit par un identifiant « **source authentique** » (numéro BCE, matricule...), soit par une un **numéro interne** à l'application (principalement DDT, Limosa)
- Idem pour les personnes : numéro NISS, ou numéro interne

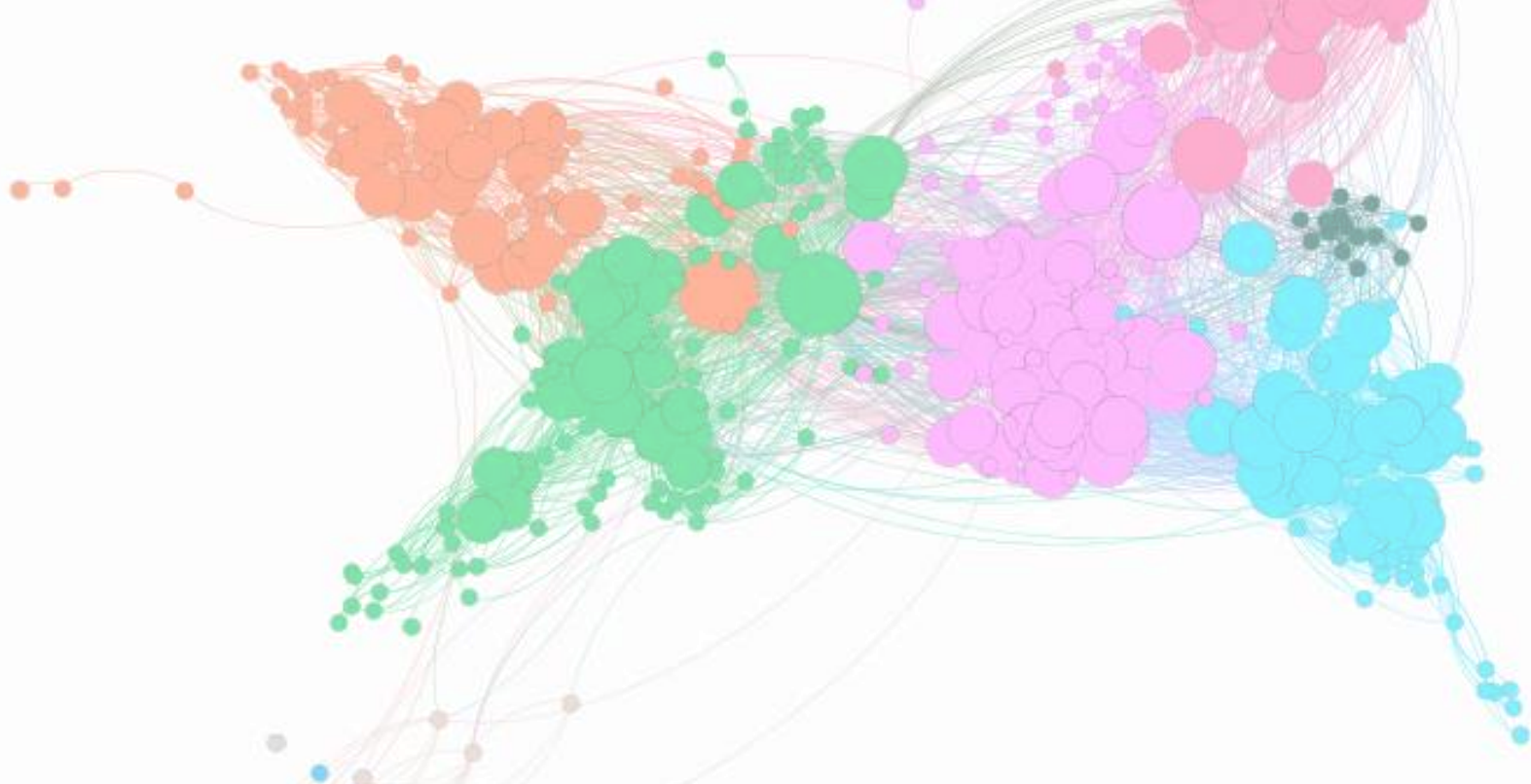
# SocDumpv2 : Flux



# SocDumpv2

---

- Après la recherche des données, on va :
  - Collecter les **adresses** des entreprises
  - **Enrichir** les nœuds (nom des entreprises et des personnes, dates de naissance)
  - Détecter et (éventuellement) fusionner les **doublons**
  - Calculer une série de **métriques** (distance, centralité...)
- On **visualise** ensuite le résultat : Gephi, TouchGraph...



# Manipuler un réseau

Manipuler un réseau

# IGRAPH (PYTHON/R)

# iGraph

---

- iGraph : librairie C de création, manipulation et analyse de graphes
- Interfacé avec R, Python, Mathematica
- Open Source, GNU GPL v2
- Très populaire dans la communauté scientifique
  
- + : standard, multi-langage, natif, très riche
- - : mono-cpu, installation pas toujours aisée

# iGraph : fonctions

---

- **Génération** : de nombreux modèles
- **Analyse** : centralité, chemins, communauté, recherche de pattern/motifs, comparaison isomorphique...
- **Modification** : projection bipartite, extraction
- **Visualisation** : layout, génération d'image
- **Import/Export**

# iGraph, Python vs R.

---

## Python

```
g = ig.Graph()
g.add_vertices(["a","b","c"])
g.add_edges([("a", "b"),
             ("a", "c")])

g.vs["label"] = ["x","y","z"]

g.vs[2]["name"] = "C"
```

## R

```
g <-graph(edges =
          c("a", "b", "a", "c"))

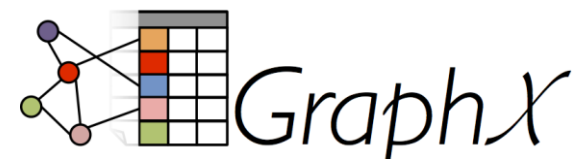
V(g)$label = c("x", "y", "z")

V(g)$name[2] = "C"
```

Manipuler un réseau

# BIG DATA: GIRAPH & GRAPHX

# GraphX



<http://spark.apache.org/graphx/>

- Librairie de manipulation de graphes pour Spark (framework open source de calcul distribué/Big Data)
- Spark :
  - Souvent utilisé sur Hadoop (pour HDFS)
  - En Scala, Java ou Python
  - GraphX : Uniquement en Scala
- Basé sur les RDDs (non-modifiables)
- + : Permet la manipulation de très gros graphes
- - : Librairie très limitée (à ce stade)



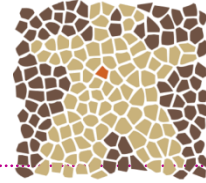
# GraphX

```
// Create an RDD for the vertices
val users: RDD[(VertexId, (String, String))] =
sc.parallelize(Array((3L, ("rxin", "student")),
                    (7L, ("jgonzal", "postdoc")),
                    (5L, ("franklin", "prof")))))

// Create an RDD for edges
val relationships: RDD[Edge[String]] =
sc.parallelize(Array(Edge(3L, 7L, "collab"),
                    Edge(5L, 3L, "advisor"),
                    Edge(5L, 7L, "pi")))

Graph val graph = Graph(users, relationships, defaultUser)
```

# Giraph



A P A C H E  
G I R A P H

<http://giraph.apache.org>

- "*Apache Giraph* is an iterative graph processing system built for high scalability."
- Extension de **Pregel** (Google)
- Utilisé par **Facebook**
- Basé sur **MapReduce** (Hadoop)

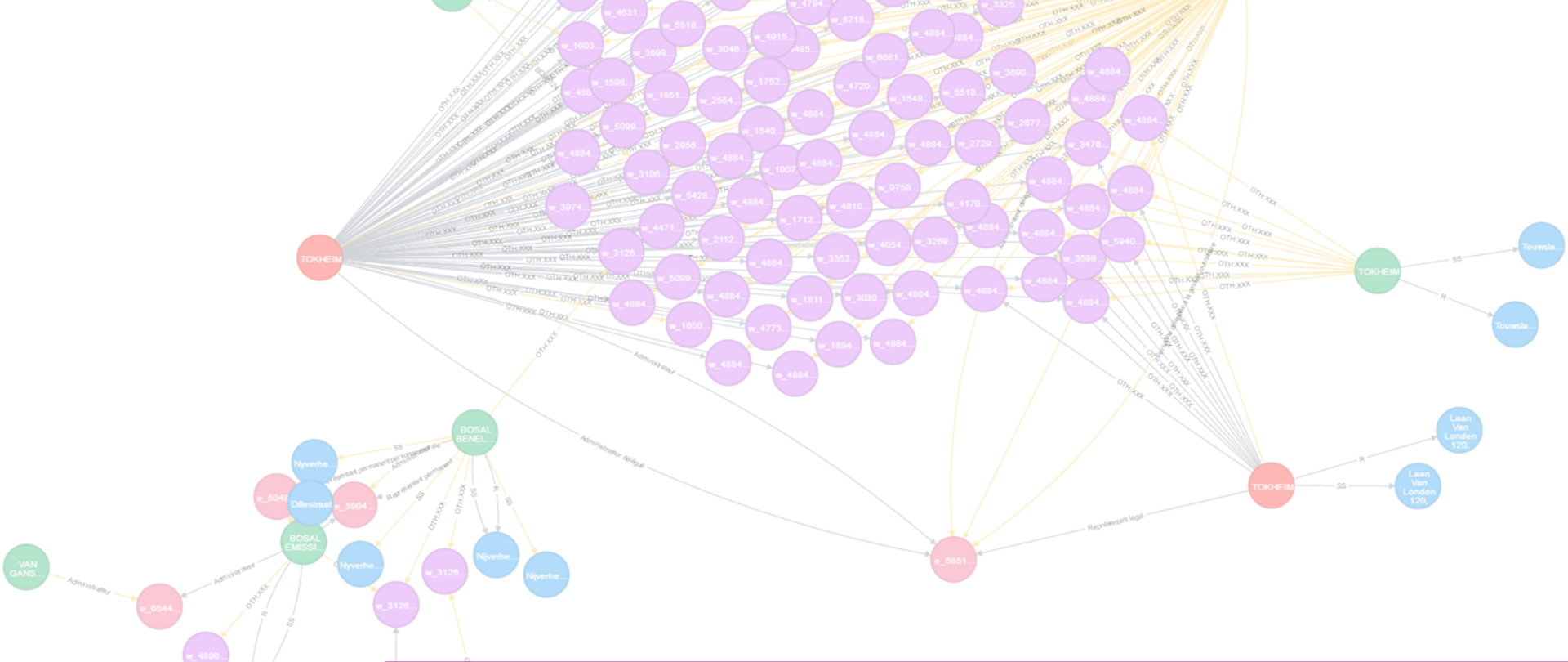
# GraphX vs Giraph

---

- GraphX créé pour être plus performant que Giraph (moins de passage par le disque)
- Benchmark par GraphX largement en sa faveur...
- Mais des benchmarks de Facebook montrent le contraire

<https://code.facebook.com/posts/319004238457019/>

<https://amplab.cs.berkeley.edu/wp-content/uploads/2014/09/graphx.pdf>



# Interroger un réseau



Interroger un réseau

# GRAPH DATABASES

# RDBMS et relations

Workers
ID
Name
Employer_ID

Companies
ID
Name



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 : Relational DataBase Management System

Définir – Caractériser – Visualiser – Manipuler – Interroger

# RDBMS et relations

Workers
ID
Name

Works_for
Worker_ID
Company_ID
From_date

Companies
ID
Name



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
```

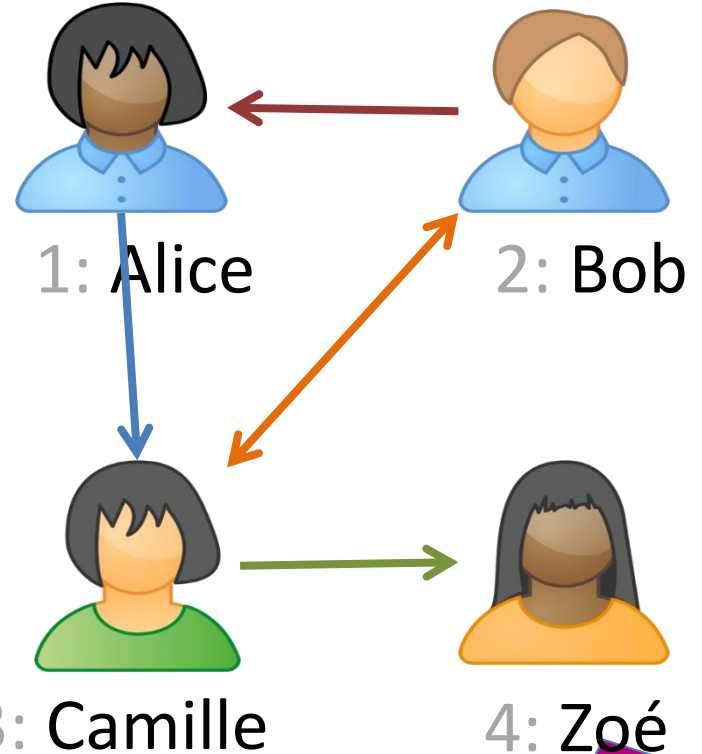
```
ON Likes.Liked_ID = p1.ID
```

```
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
```

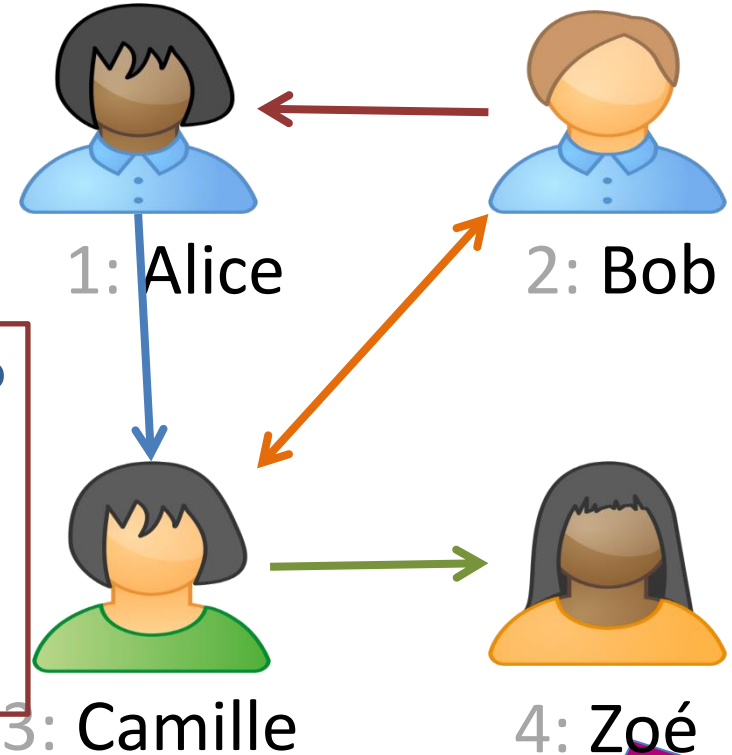
```
ON l1.Liker_ID = l2.Liked_ID
```

```
JOIN People p2
```

```
ON l2.Liker_ID = p2.ID
```

```
WHERE p2.Name = "Bob"
```

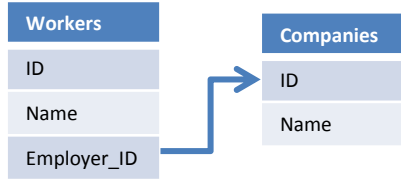
Qui Bob (aime)<sup>2</sup> ?



# RDBMS et relations : limitations

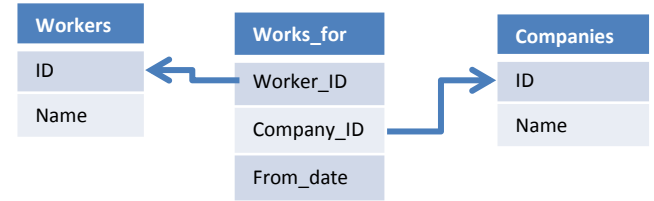
## Relations 1-n

Foreign key:



## Relations m-n (ou 1-n avec attributs)

Join table:



- Détourne le rôle d'un **attribut**
- Pas de type « Key »
- Contrainte d'intégrité possible (externe à la table)
- Structure de la relation à préciser à chaque requête
- Join : Lourd en écriture et en exécution
- Détourne le rôle d'une **table**
- Pas distinguable des autres

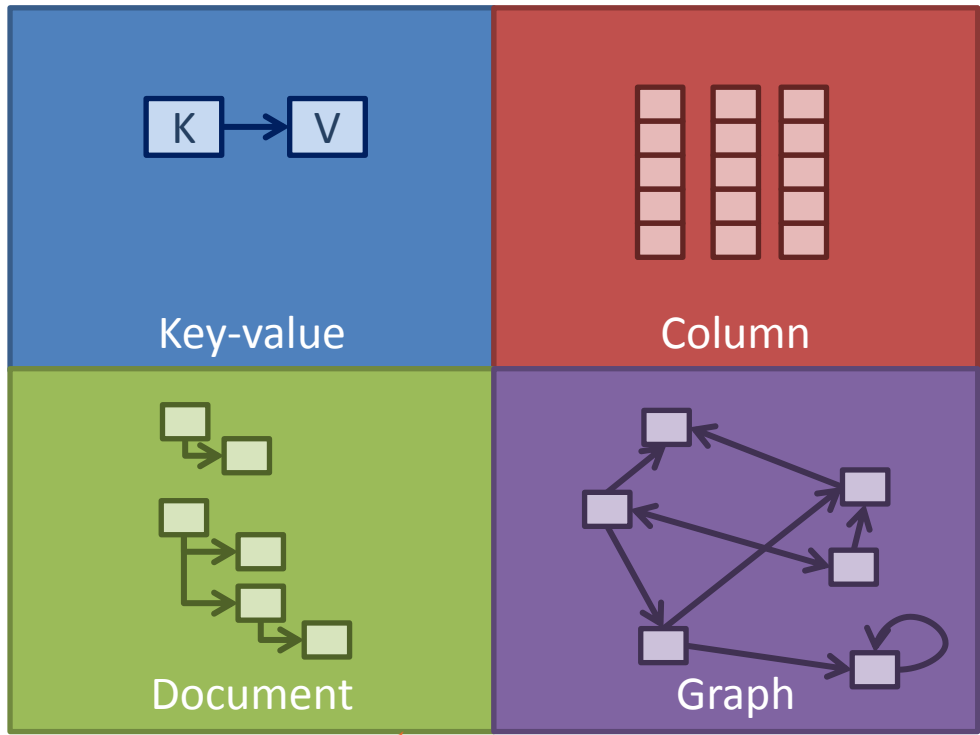
# Solutions NoSQL



*cassandra*



ORACLE  
NOSQL DATABASE



Google  
BigTable

APACHE  
HBASE

AllegroGraph



TITAN

neo4j

InfiniteGraph  
Powered by Objectivity

mongoDB

OrientDB

# 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** »

```
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"
```

Moteur efficace pour le parcours des relations

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 IName")
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 IName")
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 IName")
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 IName")
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 IName")
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 IName")
GROUP BY directReportees
UNION
SELECT L2Reportees.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 IName")
GROUP BY directReportees
) AS T
WHERE
GR
UN
) A
GR
UN
(SE
FR
JO
ON
JO
ON
W
)
```

```
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 IName")
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 IName")
GROUP BY directReportees
UNION
SELECT L2Reportees.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 IName")
GROUP BY directReportees
) AS T
WHERE
GR
UN
) A
GR
UN
(SE
FR
JO
ON
JO
ON
W
)
```

## 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."*

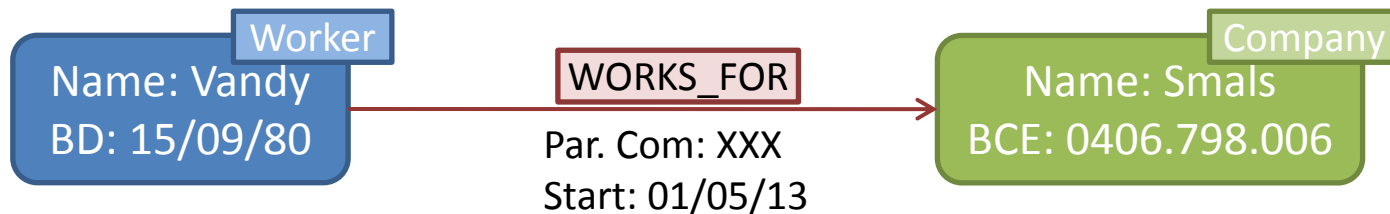


<https://neo4j.com/news/graph-databases-provide-crystal-ball-online-recommendations-wal-mart-ebay/>

Interroger un réseau

**NEO4J**

# Requête Neo4j



Nœud de type « Worker »

Nœud de type « Company »

Relation de type « WORKS\_FOR »

```
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

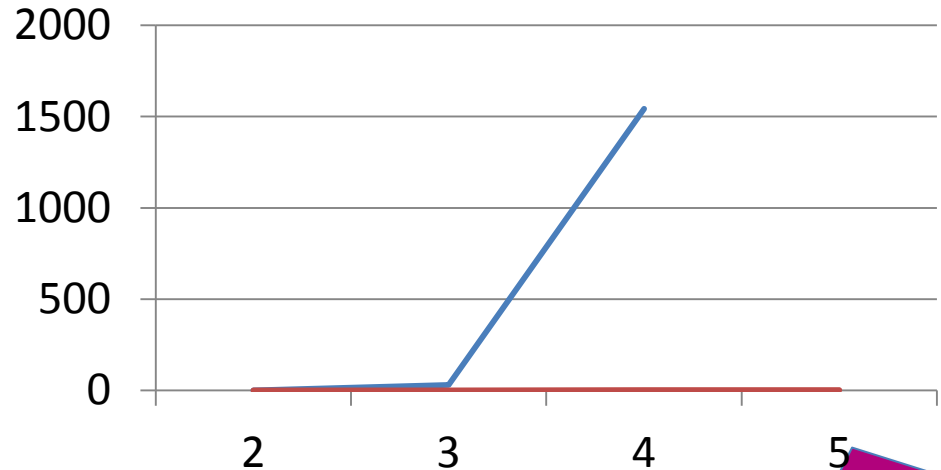
	RDBMS	Graph DB
schema		
WHERE ...	Search in Companies (with index): $O(\log([\text{Table size}]))$	Search in nodes (with index) : $O(\log([\text{DB size}]))$
Relation	JOIN : - Search in Works_for : $O(\log([\text{Table size}]))$ - Search in Workers : $O(\log([\text{Table size}]) \times [\text{nb res}])$	Follow pointers in node : $O([\text{nb of employees}])$

# Performance

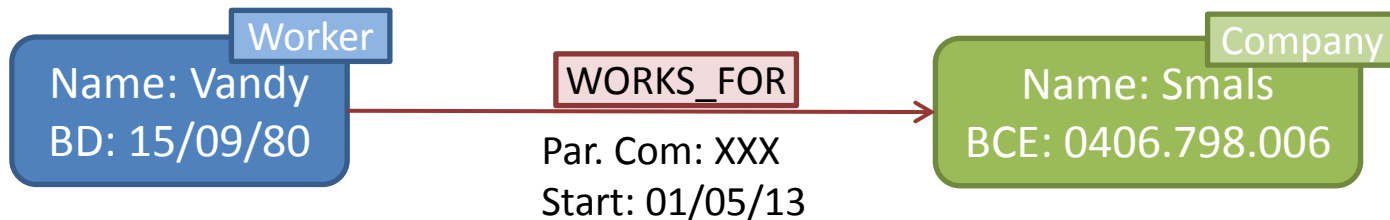
- Réseau social de  $10^6$  de nœuds
- $\sim 50$  voisins par nœuds
- Recherche des « amis d'amis »

Scénario très favorable aux graph DB !

Depth	MySQL	Neo4j	# results
2	0,02	0,01	2 500
3	30,27	0,17	110 000
4	1543,51	1,36	600 000
5	Unfinished	2,13	800 000



# Neo4j : Création des données



```
CREATE (:Person {Name: "Vandy", BD: "15/09/80"})
```

```
CREATE (:Company {Name: "Smals", BCE: "0406..."})
```

```
MATCH (p:Person {Name: "Vandy"}),  
(c:Company {Name: "Smals"})
```

```
CREATE (p) -[:WORKS_FOR {Start:...}] -> (c)
```

OU:

```
CREATE (:Person {Name: "Vandy", BD: "15/09/80"})  
-[:WORKS_FOR {Start:...}] -> (:Company {Name: "...", ...})
```

# Neo4j : importation de données

---

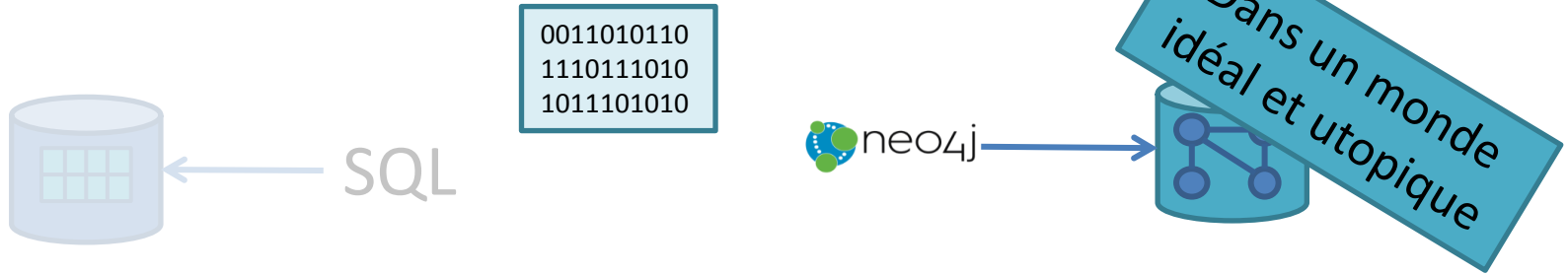
- Dans la pratique : importation de fichiers CSV (à préparer !!)
- Via une commande *Cypher* (**LOAD CSV FROM...**)
  - Facile pour des petits fichiers
  - Lent à l'exécution
- En *command line*
  - Beaucoup plus rapide
  - Mal documenté

Interroger un réseau

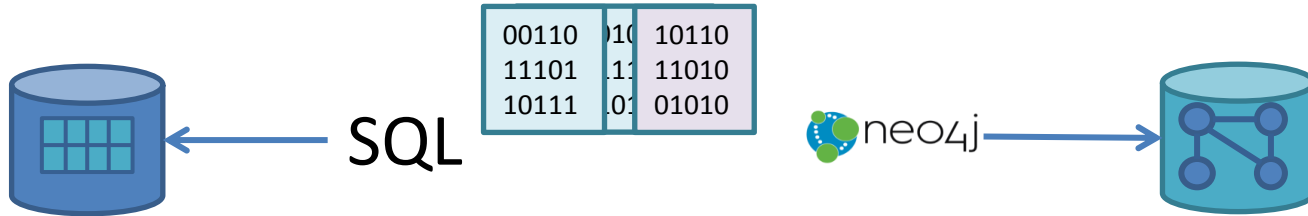
# GRAPHDB : LA SOLUTION ?

# RDBMS ~~et~~ GraphDB ?

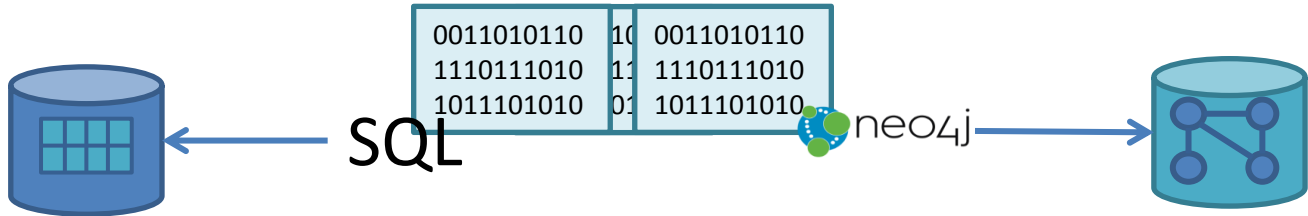
1



2



3



# 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...
- Neo4J: ACID, schema-less

# 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)

Interroger un réseau

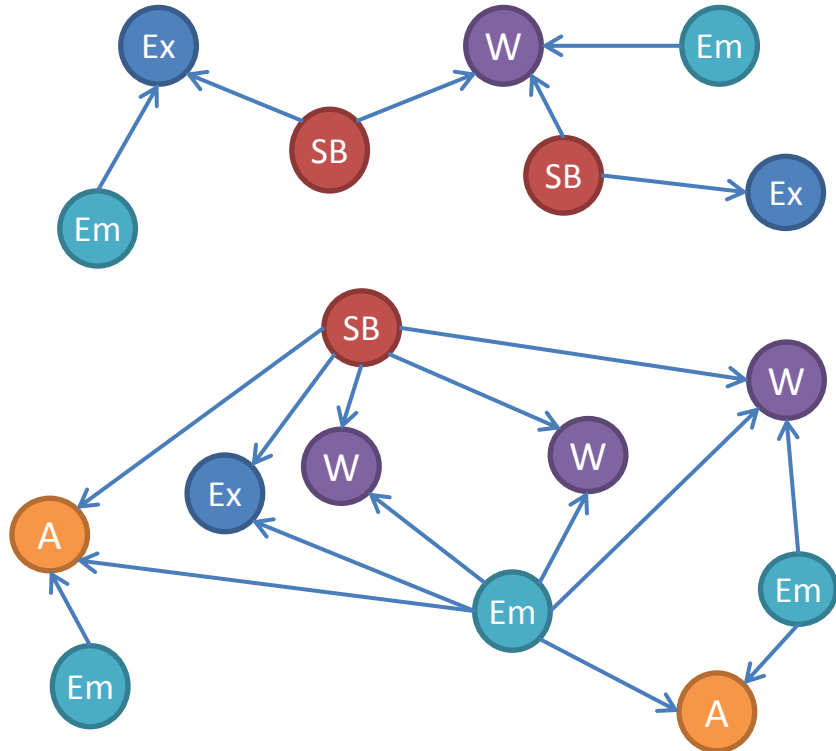
# USE CASE : SUPERBONUS

# 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

# Data collection



**DmfA: SuperBonus requestors**  
11078 employers

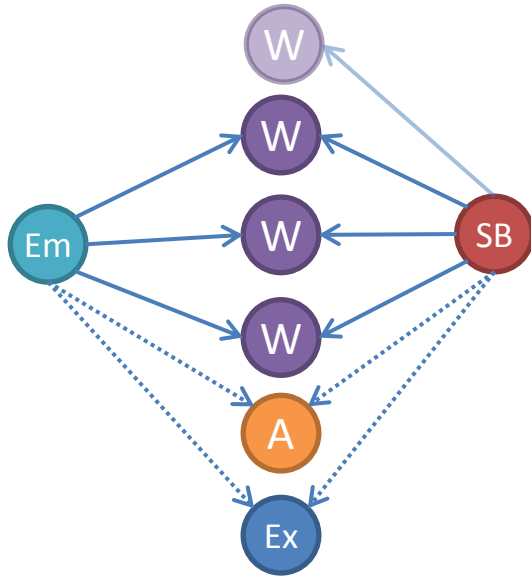
**Dimona: Worker in Periods 2015-2016**  
27056 workers

**Dimona: Employers in Periods 2015**  
15416 employers

**BCE: Executors + other companies**

**BCE: Addresses (3296 addr.) + cotenant**

# Full transfert



$\text{COUNT}(W) > \text{Em.Worker\_Cnt\_2015} * 80\%$

$\text{COUNT}(W) > \text{SB.Worker\_Cnt\_2016} * 80\%$

```
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
```

 Worker

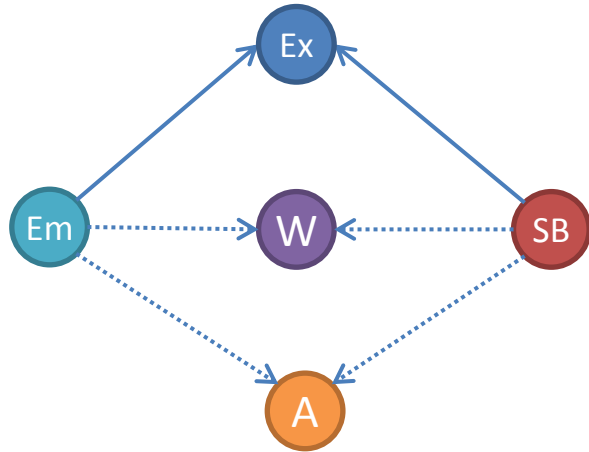
 Employer without SB

 Employer with SB

 Address

 Executor

# Shared admin and (worker or address)



$\text{COUNT}(W) > 0$  **OR**  $\text{COUNT}(A) > 0$

```
MATCH (sb:SB) -- (e:Executor) -- (comp:Company)
WHERE sb <> comp
OPTIONAL MATCH (sb) -- (w:Worker) -- (comp)
OPTIONAL MATCH (sb) -- (a:Address) -- (comp)
WITH sb, comp,
    COUNT(DISTINCT w) AS Workers_count,
    COUNT(DISTINCT a) AS Addresses_count
WHERE (Workers_count>0 OR Addresses_count>0)
RETURN sb, comp, ...
```

 Worker

 Employer without SB

 Employer with SB

 Address

 Executor

# Conclusions

# En résumé

---

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

**Manipuler** (coding) : Envir. classiques ou Big Data

**Visualiser** : Desktop, Javascript, Cloud

**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-scientists/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) : librairie JS / outils « à la » Linkurio.us
  - Statique : Gephi/TouchGraph, igraph → Excel, PDF...

# Références : [www.smalsresearch.be](http://www.smalsresearch.be)

---

- Blogs :
  - <https://www.smalsresearch.be/un-fraudeur-ne-fraude-jamais-seul/>
  - <https://www.smalsresearch.be/un-fraudeur-ne-fraude-jamais-seul-partie-2/>
  - <https://www.smalsresearch.be/bases-de-donnees-relationnelles-adequates-pour-des-relations/>
  - <https://www.smalsresearch.be/graph-db-vs-rdbms/>
- Autres :
  - NoSQL - Hype ou Innovation, Grégory Ogonowski, <https://www.smalsresearch.be/publications/document/?docid=59>

# 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](mailto:vandy.berten@smals.be)



More on Smals Research :  
Website : [www.smalsresearch.be](http://www.smalsresearch.be)  
Blog : [www.smalsresearch.be/blog](http://www.smalsresearch.be/blog)  
Twitter : [@SmalsResearch](https://twitter.com/SmalsResearch)

