Smals
ICT for society

Not Only SQL

# NoSQL : hype ou innovation ?

Grégory Ogonowski / Recherches
Octobre 2011

# Sommaire

- Introduction
- Théorème CAP
- NoSQL (principes, mécanismes, démos,...)
- Ce que nous avons constaté
- Recommandations
- Conclusion

# Introduction

- Bases de données relationnelles :
  - Solutions mûres et maîtrisées
  - Robustes
  - Ont fait leurs preuves

- Pourquoi s'embêter avec autre chose ???

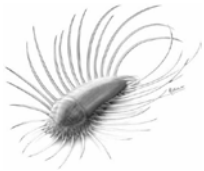
| UID | NOM | Prénom |
|---|---|---|
| 1 | Jagger | Mick |
| 2 | Harrison | George |
| 3 | Starr | Ringo |

| UID | GID |
|---|---|
| 1 | 2 |
| 2 | 1 |
| 3 | 1 |

| GID | Groupe |
|---|---|
| 1 | Beatles |
| 2 | Rolling Stones |

# Introduction : impact du Web 2.0

1970 1990 2000 2010



Primum Adabasolite

Relationalus databasauris

IBM LOTUS Dominosaurus

Armageddon 2.0

Columnus NoSQLausis

Graphosis NoSQLausis

KeyValueraptor NoSQLausis

Documentaris NoSQLausis

!!! N'ont pas disparu et ne disparaîtront pas !!!

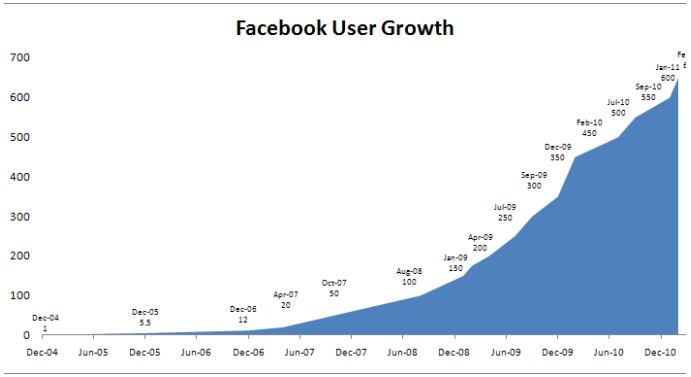
Smalsopithecus

# Introduction : impact du Web 2.0

- Énorme masse d'utilisateurs
- Gros volumes de données à gérer (hype Big Data)
- Contraintes différentes :
  - Scalabilité > Consistance
  - Disponibilité > Consistance

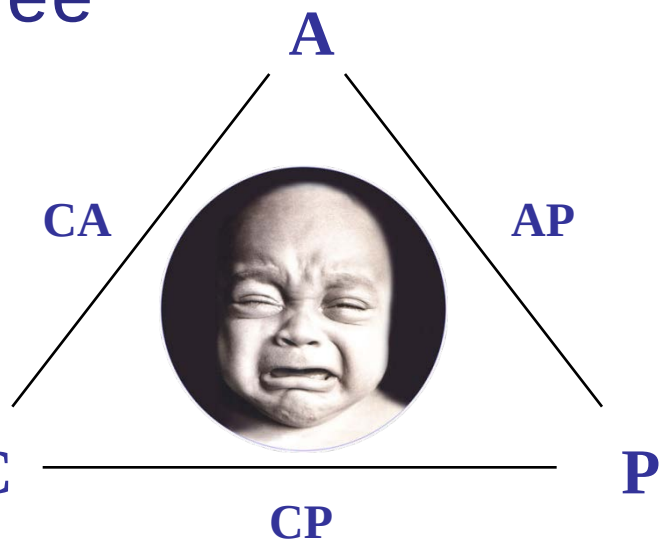
# Introduction : impact du Web 2.0

- Facebook:
  - Pic de 13 000 000 req db/sec
  - 690 milliards de pages vues/mois
  - 300 To de données… en cache mémoire
  - 25 To de logs par jour
  - 1 ingénieur/1,1 millions d'utilisateurs




# Théorème CAP

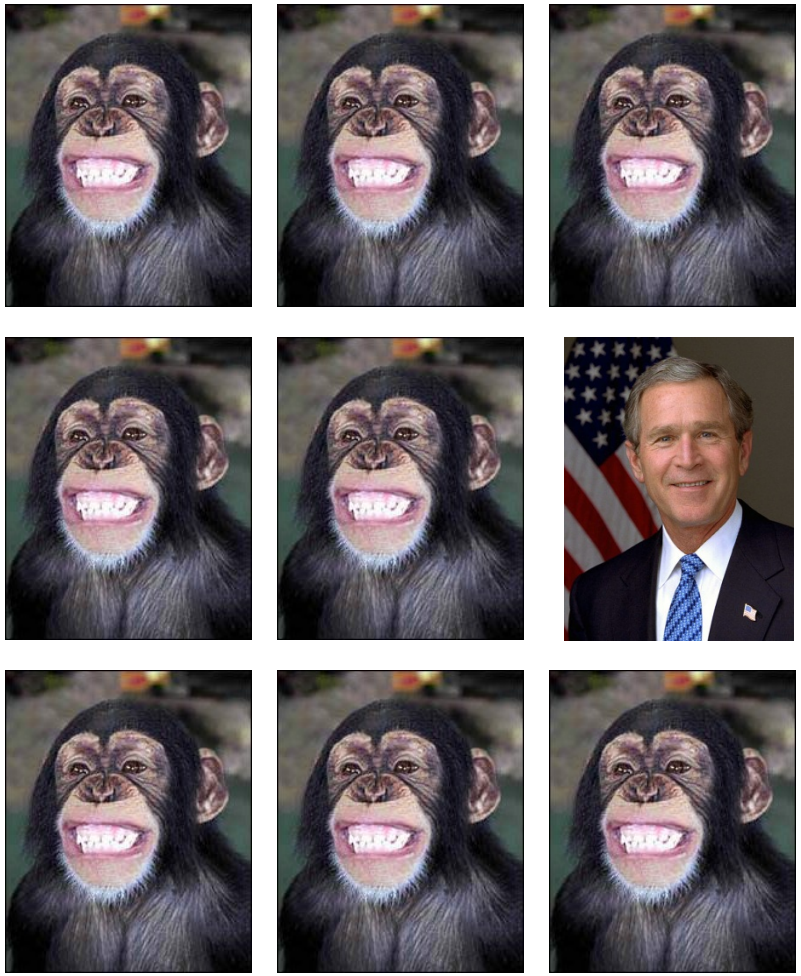
- **C**onsistency,
  **A**vailability,
  **P**artition tolerant
- Conjecture, démontrée en 1992 pour les systèmes distribués asynchrones




**Choisissez-en deux ➔**

# Théorème **C**AP : **C**onsistency

Tous les utilisateurs ont toujours la même vue sur toutes les données



# Théorème C**A**P : **A**vailability

Est-il toujours possible de lire des données et d'en écrire ?



# Théorème CA**P** : **P**artition tolerant

L'application peut-elle survivre à un partitionnement du réseau ?


# Théorème CAP

C : OK

A – P : **R**edondants ?

➔ CAP theorem =?= C**R**AP theorem

# Théorème CAP

Autre approche : PACELC

*"if there is a partition (P) how does the system tradeoff between availability and consistency (A and C); else (E) when the system is running as normal in the absence of partitions, how does the system tradeoff between latency (L) and consistency (C) ? "*

Ex : PA/EL

# Théorème CAP VS PACELC

VS

CAP

PACELC

**Simple**
**Flou**

**Complexe**
**Clair**

© http://buzzworld.fr/

**Deux manières différentes pour exprimer la même chose.**

# NoSQL : C'est quoi ?

- Catégorie de **B**ases de **D**onnées

- NoSQL = ~~No SQL~~ Not only SQL

(**S**tructured **Q**uery **L**anguage)

Ex : SELECT champ1, champ2 FROM table;

# NoSQL : C'est quoi ?

- Souvent
  - très scalables
  - non-relationnelles
  - très spécifiques
  - sans schéma

# NoSQL : ACID VS BASE

CAP = On ne peut pas tout avoir

- ACID
  - **A**tomicity
  - **C**onsistency
  - **I**solation
  - **D**urability
- BASE
  - **B**asically **A**vailable
  - **S**oft state
  - **E**ventual consistency

Souvent choisi pour NoSQL

# NoSQL : Qui utilise ça ???

- Facebook, Twitter, Google, bit.ly, Springer, Amazon, digg, IBM, LinkedIn, Rackspace, sourceforge, ...
- Globalement : des sites à très forte consultation
- Mais aussi de nombreux sites Web en tant que cache !

# NoSQL : Et ça marche ?

- Oui mais...

# NoSQL : Et ça marche ?

- Oui mais...
- ... pas une solution miracle
- Plusieurs mécanismes (souvent simples) pour améliorer la scalabilité et/ou la haute disponibilité

# NoSQL : Quelques mécanismes

Partitionnement

Réplication (+ Quorum)

# NoSQL : Quelques mécanismes
# Partitionnement

- Découpage des données en vue de les distribuer

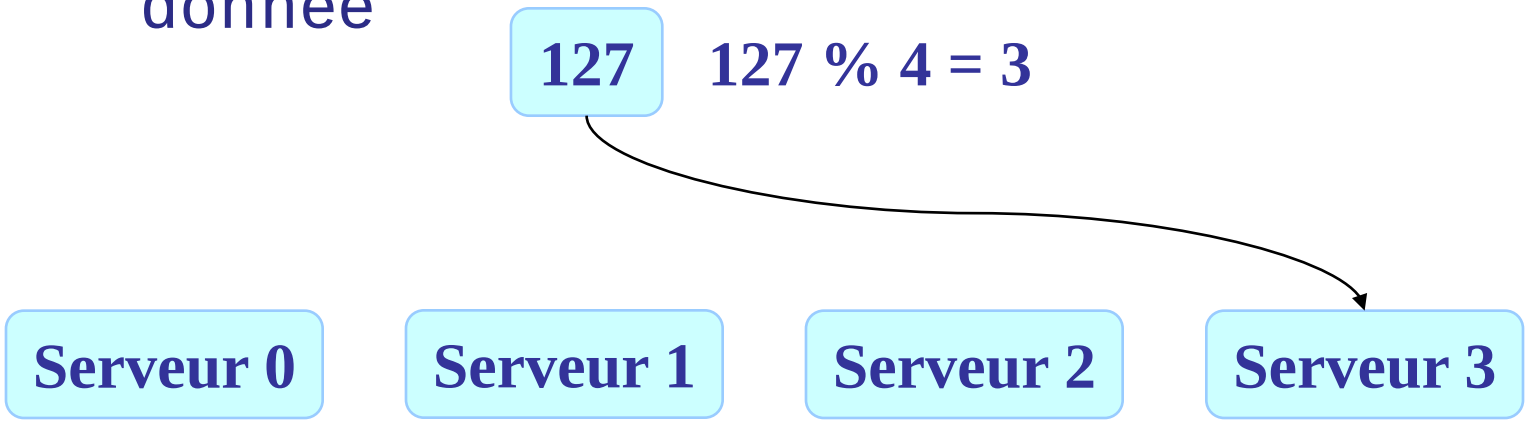
- Pas nouveau
- Mais… différentes manières de procéder

# NoSQL : Quelques mécanismes
# Réplication

- Au moins deux instances de chaque donnée
- Pas nouveau
- Mais… différentes manières de procéder


# NoSQL : Quelques mécanismes
# Partitionnement

- Traditionnellement : un "simple" modulo sur la clé pour déterminer l'emplacement d'une donnée
- Chaque nœud sait où se trouve chaque donnée



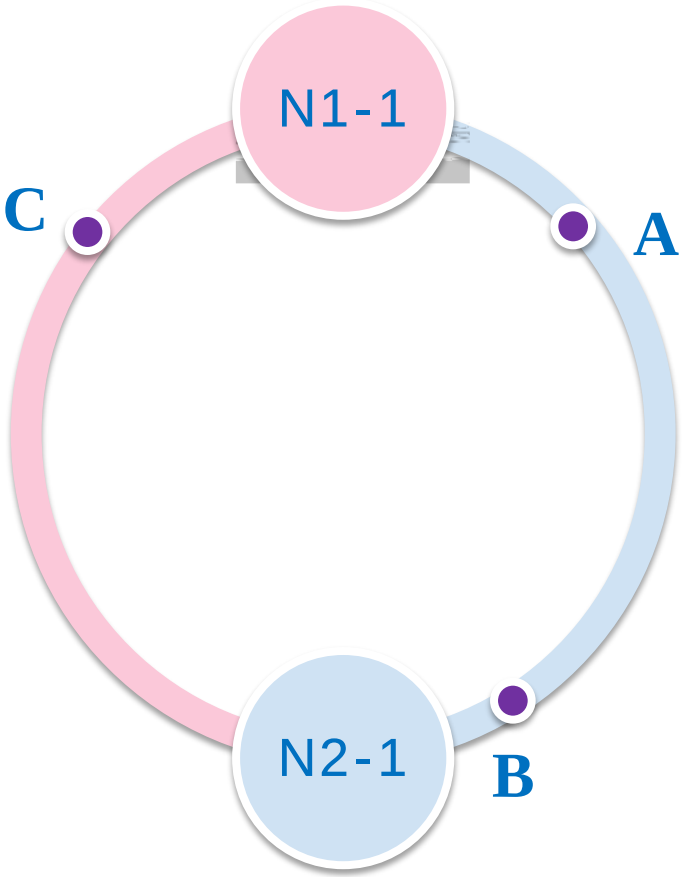

# NoSQL : Quelques mécanismes
# Partitionnement

- En cas d'ajout ou de retrait d'un serveur, besoin de tout recalculer ➔ pas évident à rendre scalable

| | | | | | |
|---|---|---|---|---|---|
| **Modulo 4** | **Serveur 0** | **Serveur 1** | **Serveur 2** | **Serveur 3** | |
| | **352** | **-** | **- 934** | **- 327, 395, 471** | |
| **Modulo 3** | **Serveur 0** | **Serveur 1** | **Serveur 2** | | |
| | **327, 471** | **- 352, 934** | **- 395** | | |
| **Modulo 5** | **Serveur 0** | **Serveur 1** | **Serveur 2** | **Serveur 3** | **Serveur 4** |
| | **395** | **- 471** | **- 327, 352** | **-** | **- 934** |

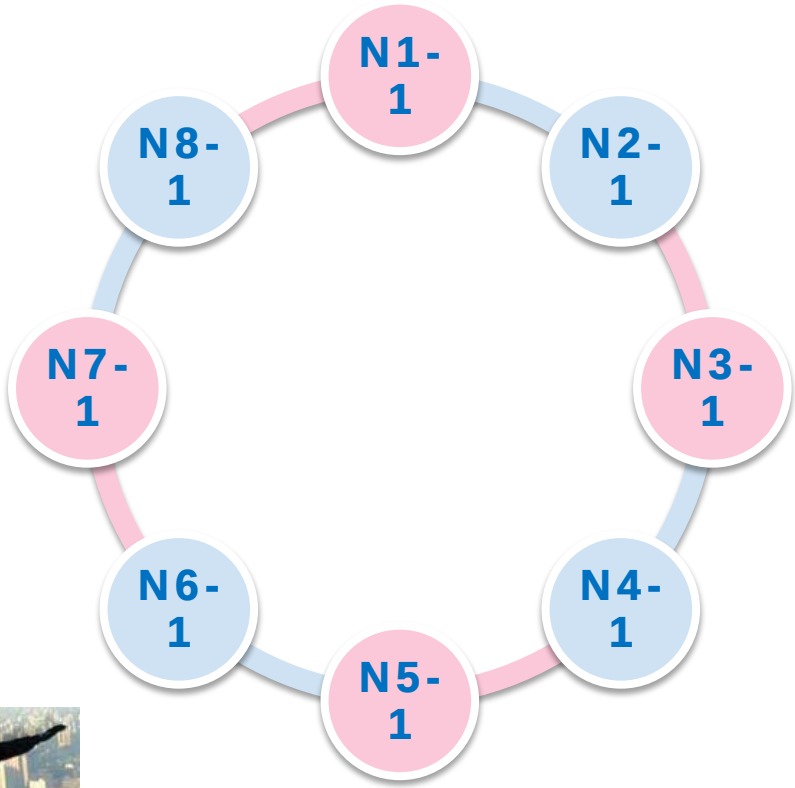
# NoSQL : Quelques mécanismes
# Partitionnement

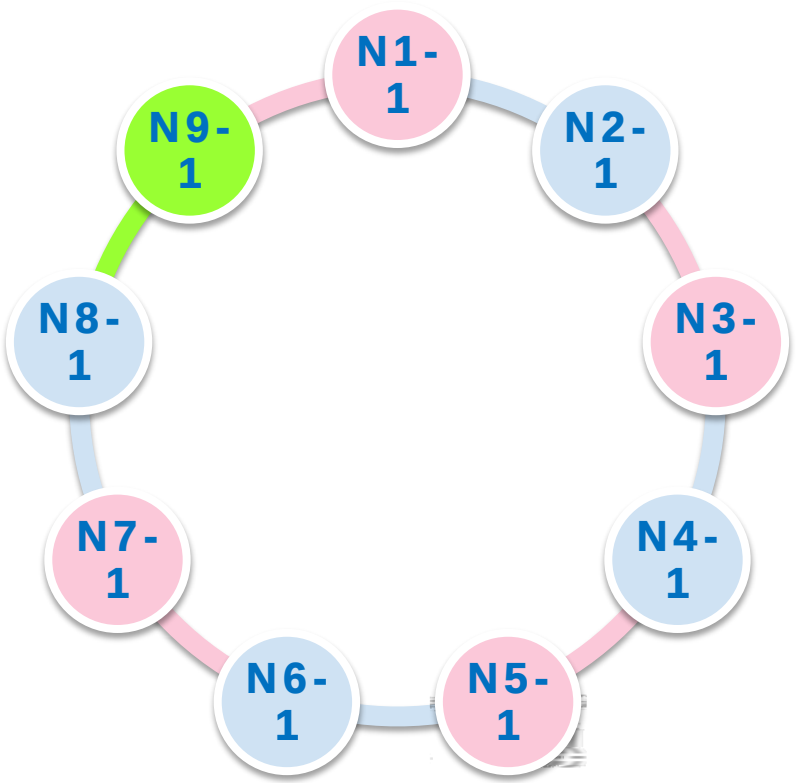
- Consistent hashing




# NoSQL : Quelques mécanismes
# Partitionnement

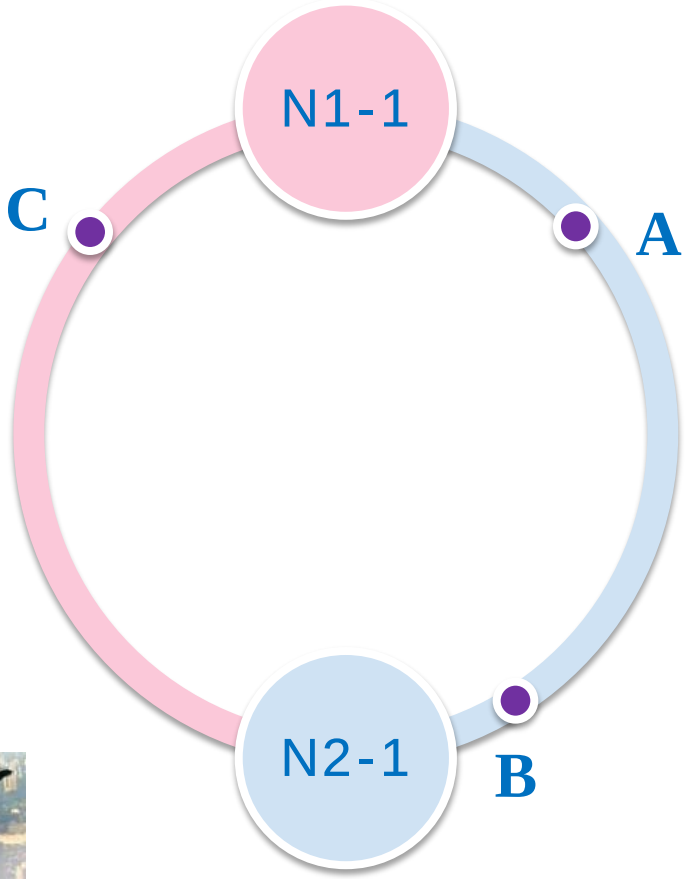
- Ajout d'un noeud : ajustements minimes

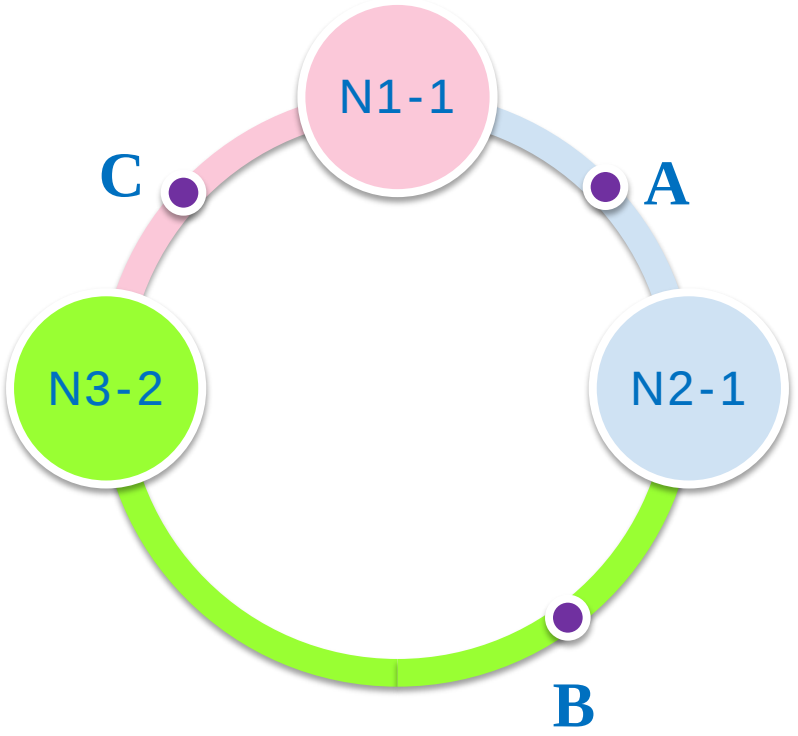



# NoSQL : Quelques mécanismes
# Partitionnement

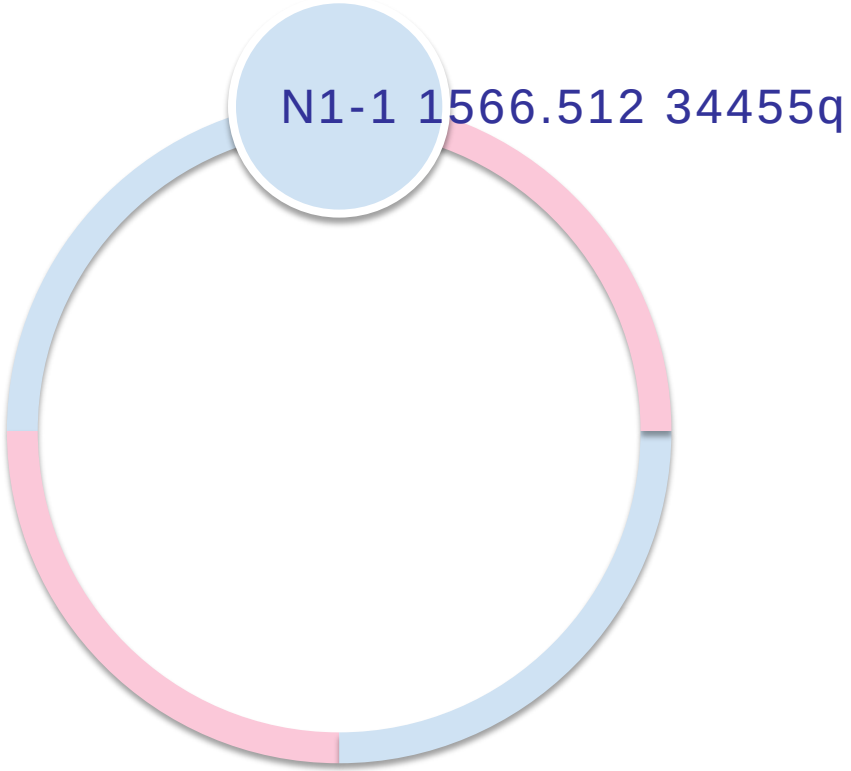
- Pondération des noeuds




# NoSQL : Quelques mécanismes
# Partitionnement

- Consistent hashing + réplication
- Relativement simple
- Chaque nœud sait où se trouve chaque donnée

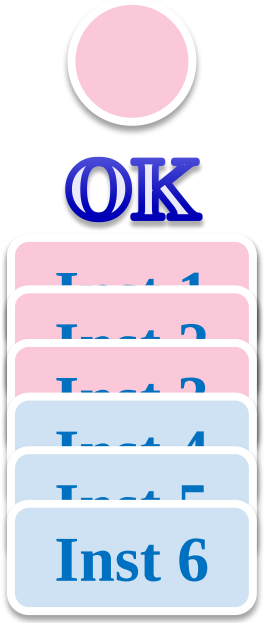



# NoSQL : Quelques mécanismes
# Quorum (finalement consistant)

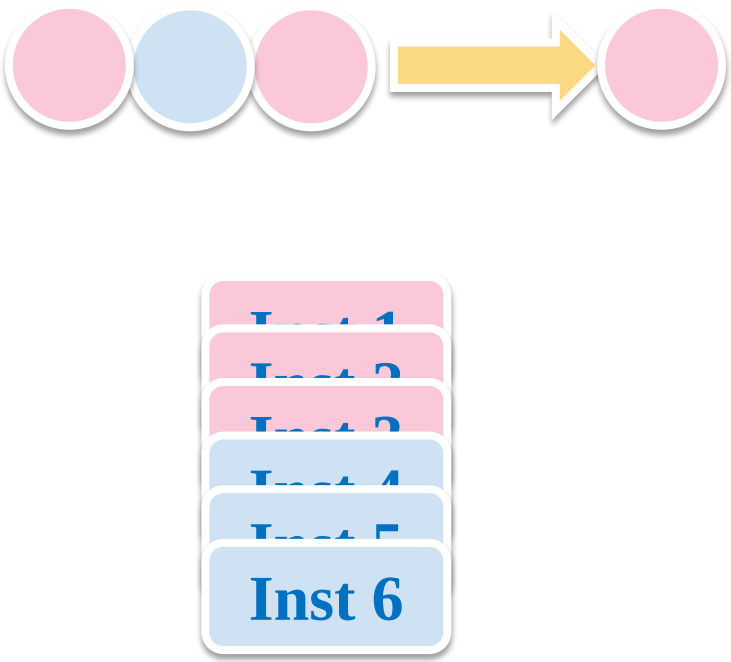
- Seuils de lecture (SL) et d'écriture (SE)
- Un petit exemple ? ☺
- Hypothèse : 6 copies de chaque donnée (#C = 6)
- Let's go…


# NoSQL : Quelques mécanismes
# Quorum (éventuellement consistant)

SE = 3




SL = 2



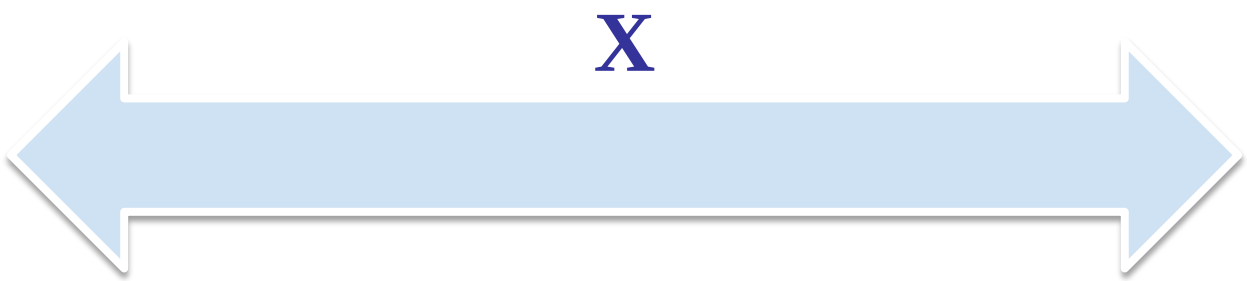

# NoSQL : Quelques mécanismes
# Quorum (finalement consistant)

- Données finalement consistantes
- Paramétrisation possible

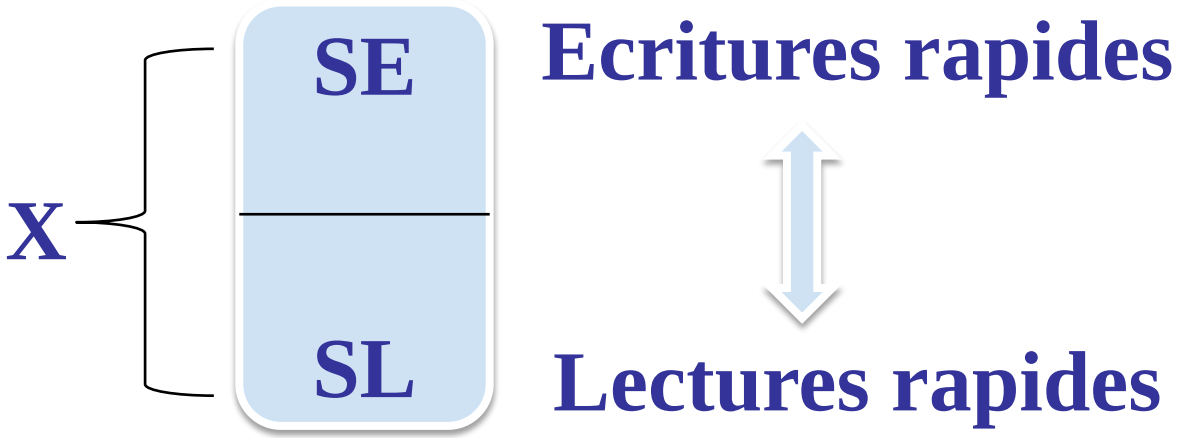
**Si (SL + SE > #C) alors**
**données consistantes**
**Sinon**
**données finalement consistantes**

**Si (SL < SE) alors**
**lectures rapides**
**Sinon si (SL > SE) alors**
**écritures rapides**
**Sinon**
**même vitesse pour écriture et lecture**

# NoSQL : Quelques mécanismes
# Quorum (finalement consistant)

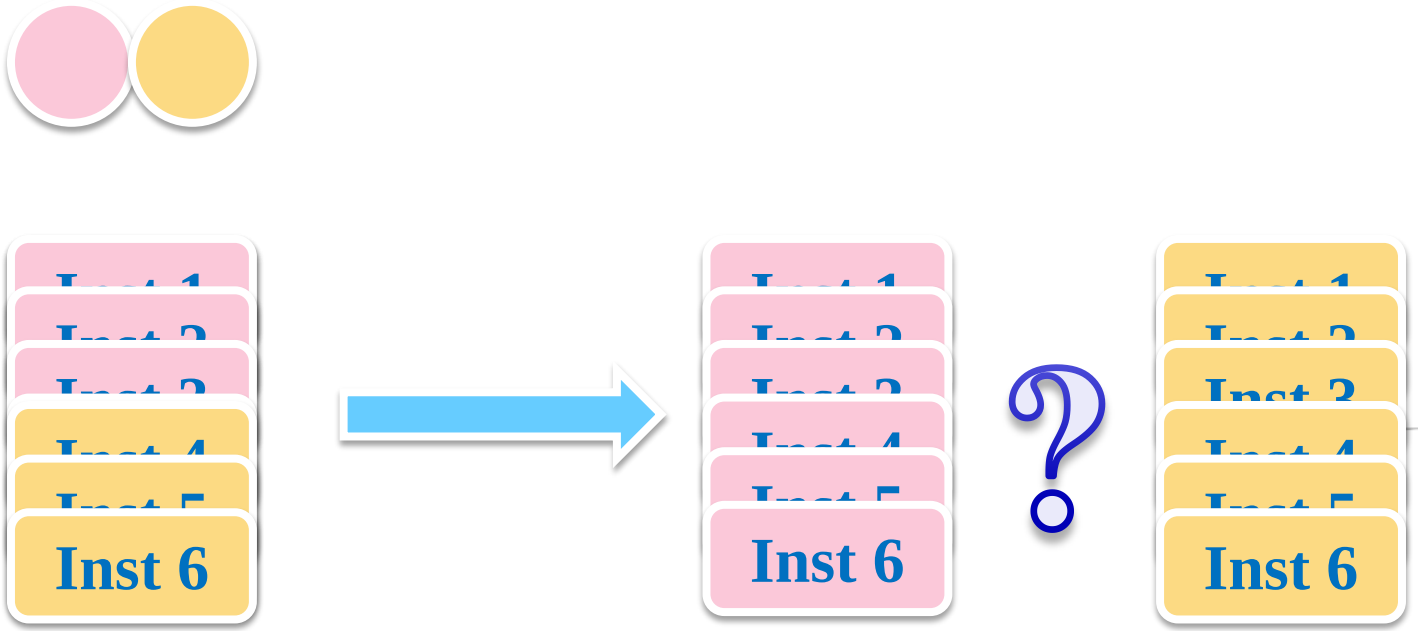



**Performances** **Consistance**




# NoSQL : Quelques mécanismes
# Quorum (finalement consistant)

## Plus complexe en pratique

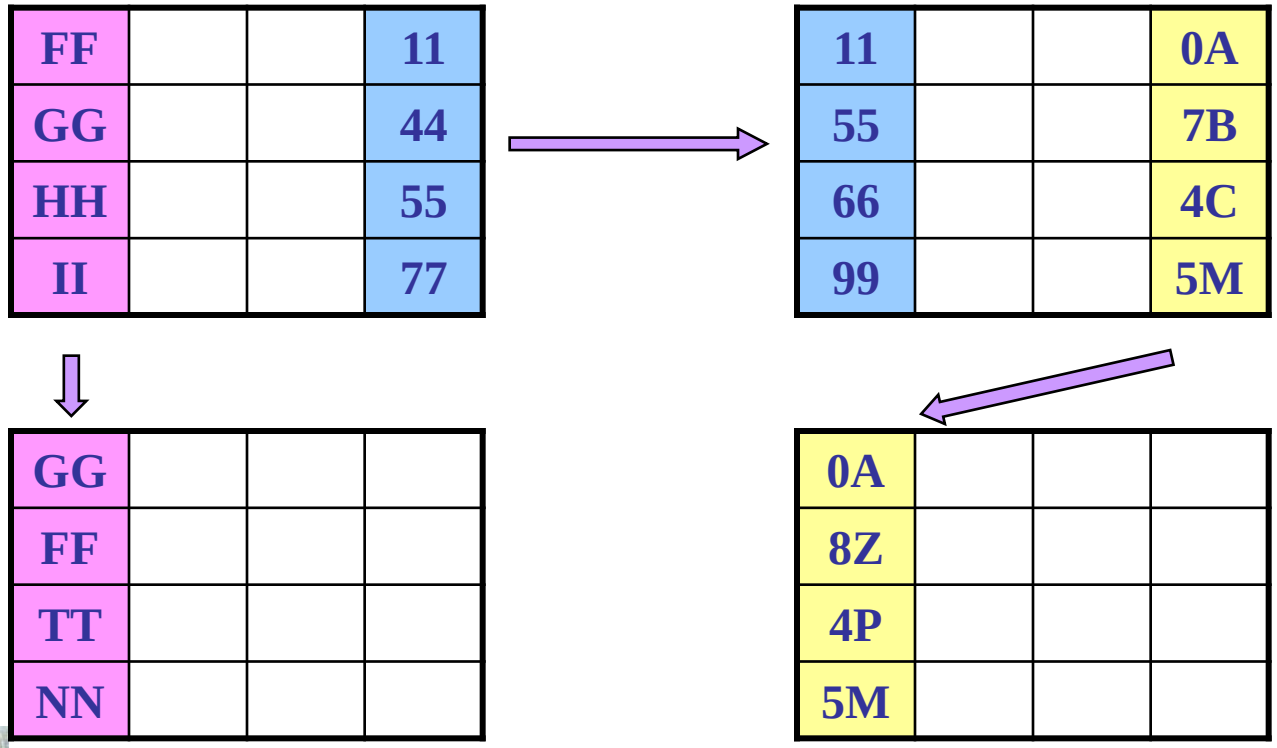



# NoSQL : Quelques mécanismes
# Quorum (finalement consistant)

- Vector clocks
- ...
- Laissons cela en annexe

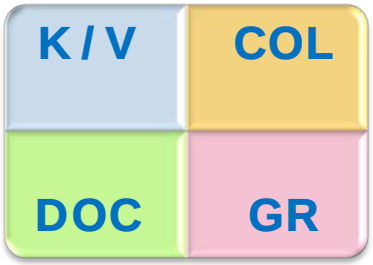
# NoSQL : Bases de données traditionnelles

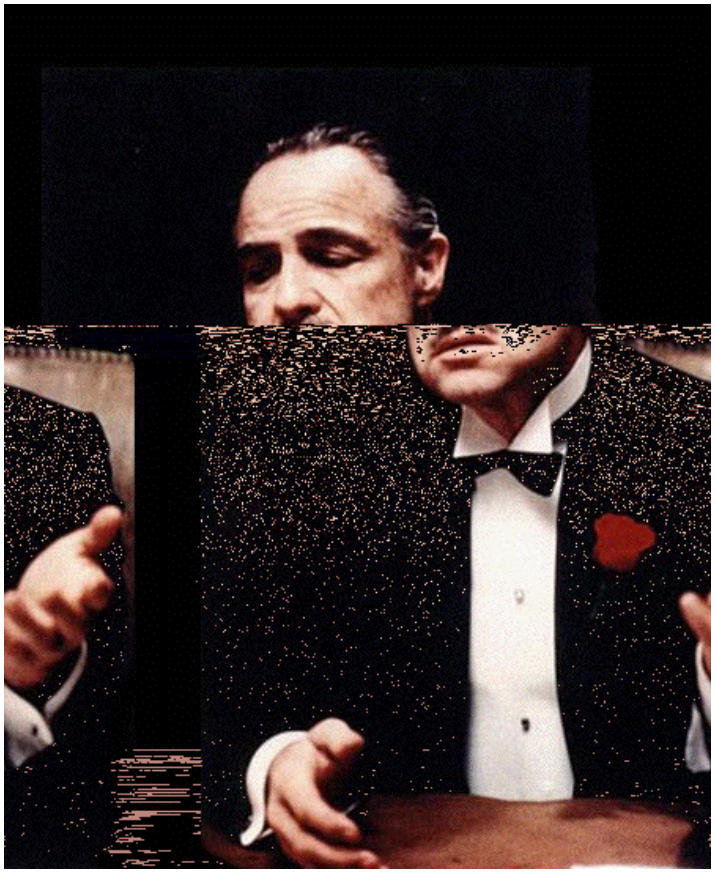
## Grands tableaux avec relations




# NoSQL : une grande famille

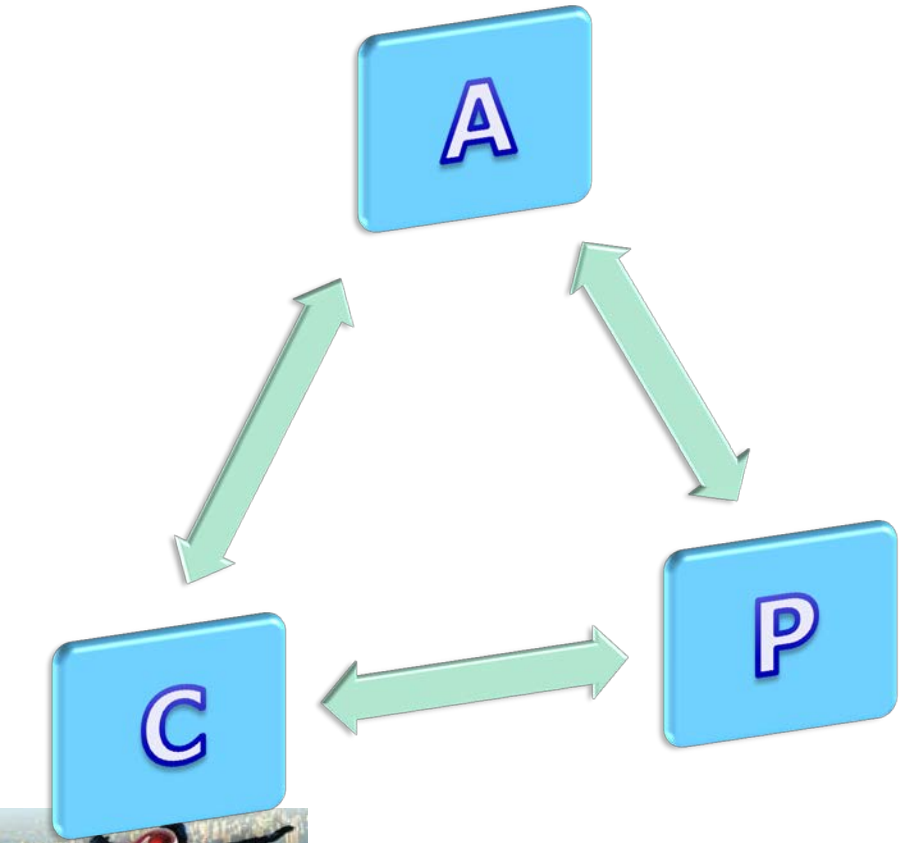
- Clé / Valeur
  - (K/V)
- Orientées colonnes
  - (COL)
- Orientées documents
  - (DOC)
- Orientées graphes
  - (GR)



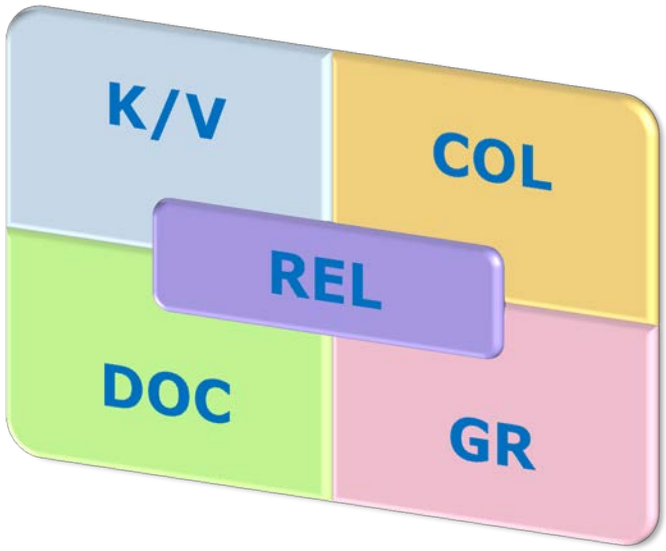



© http://www.paramount.com/

# NoSQL : caractérisation

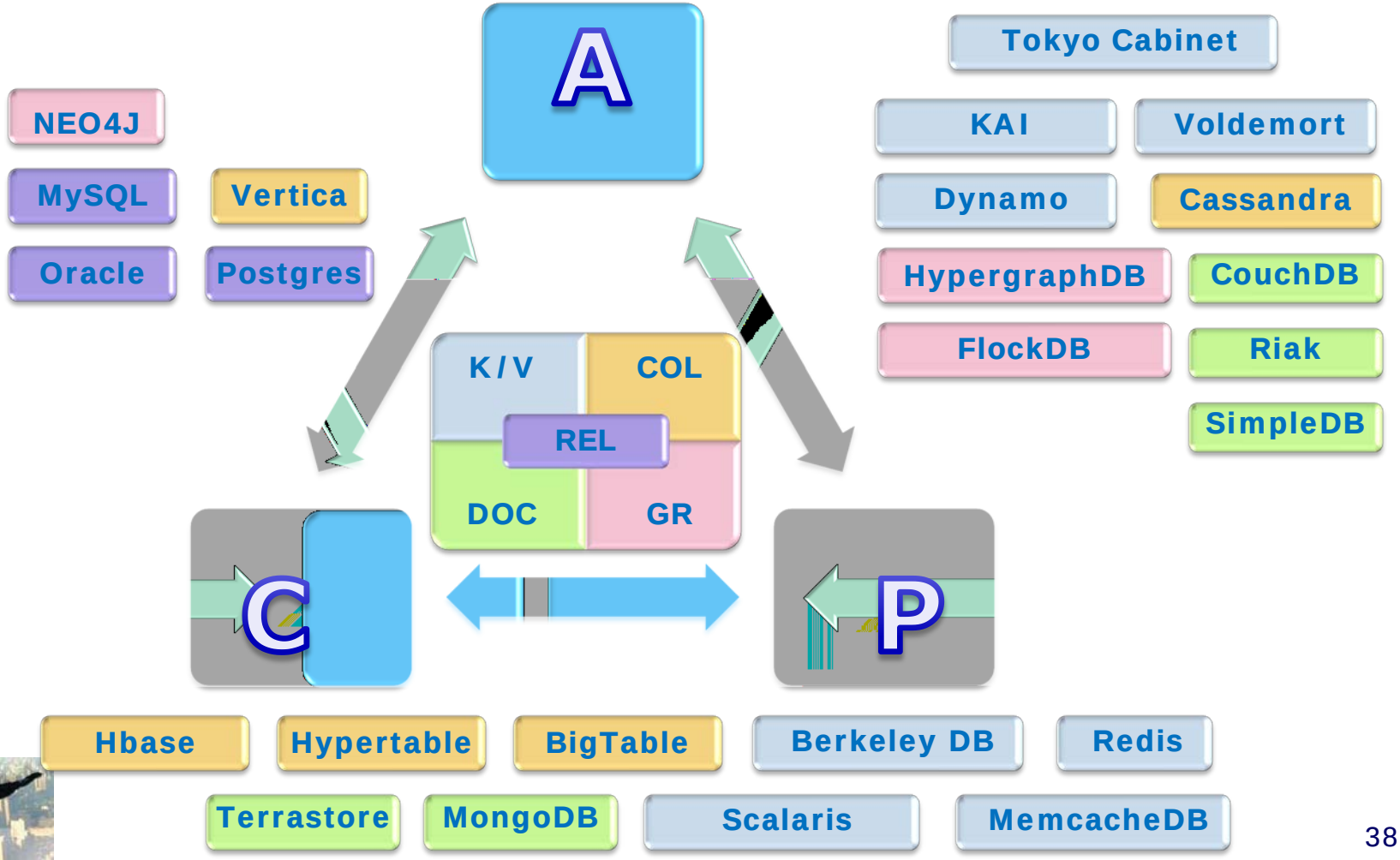






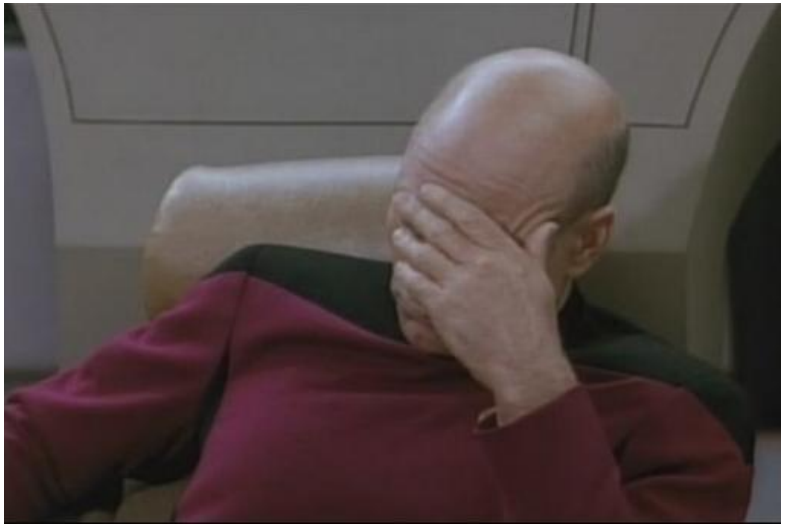

# NoSQL : caractérisation

http://blog.nahurst.com/visual-guide-to-nosql-systems



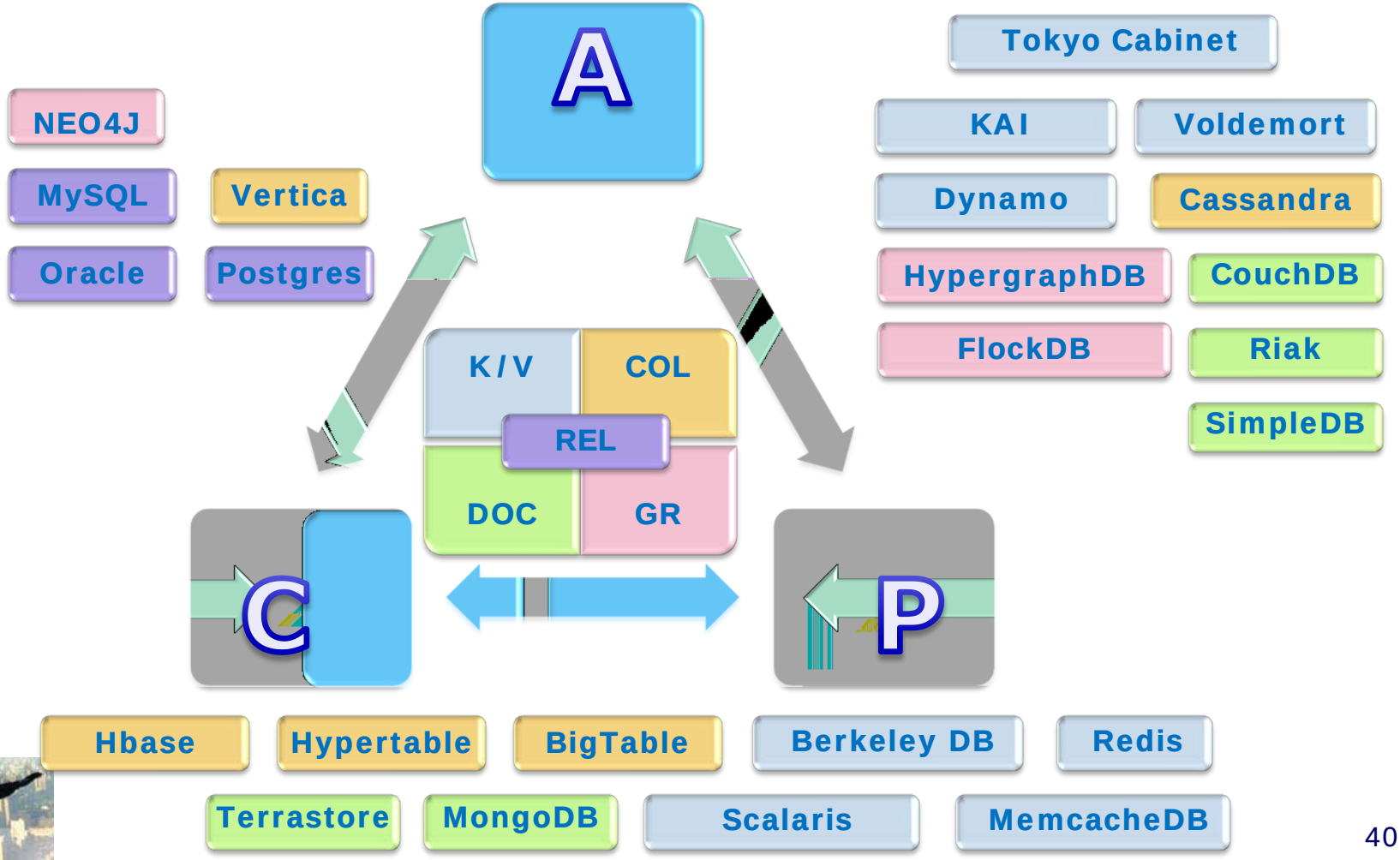

# NoSQL : caractérisation

- CA – AP – CP
- K/V – COL – DOC – GR
- 12 types de DB…
- … diverses architectures et modes de fonctionnement par type

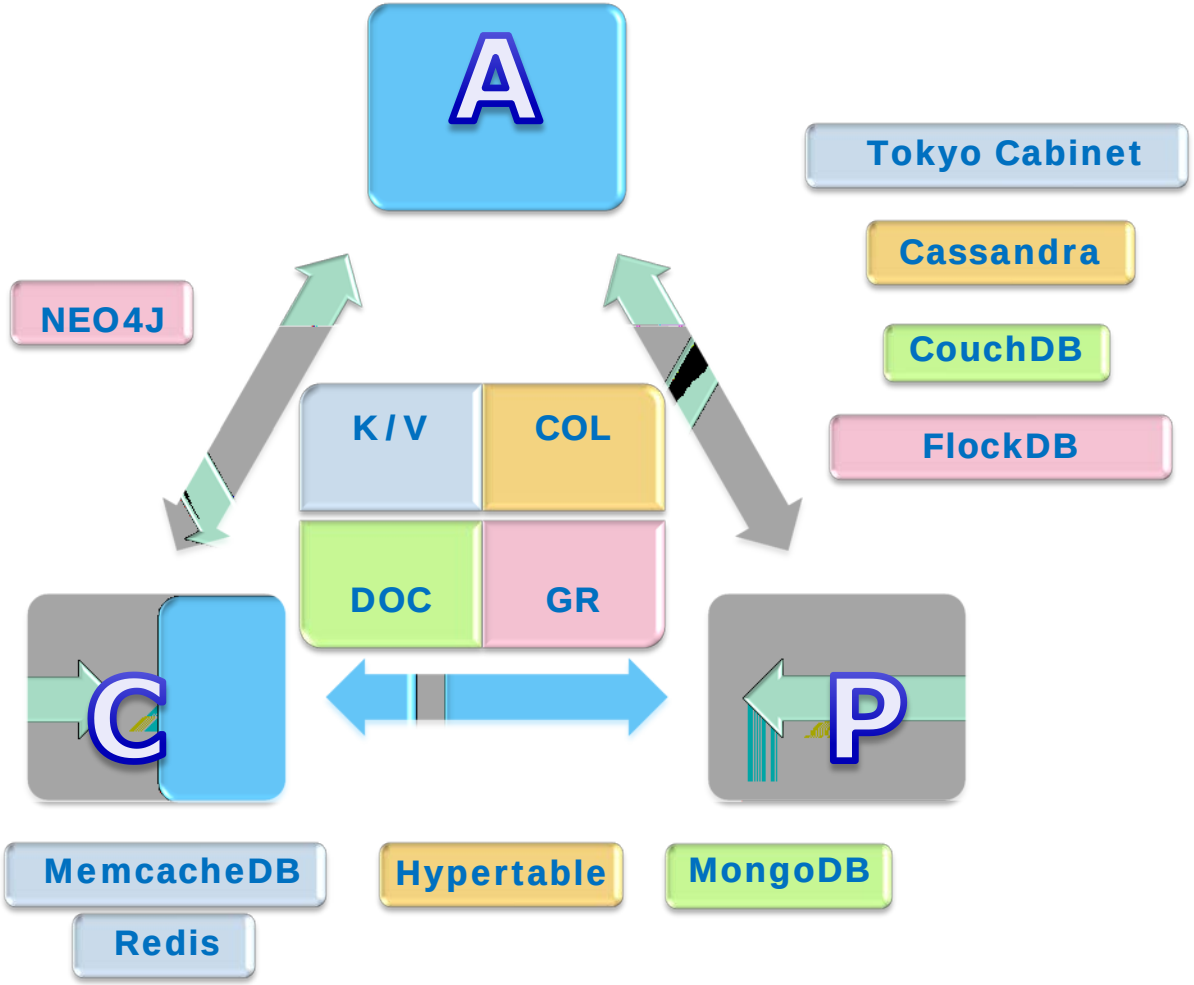


© http://www.paramount.com/

# NoSQL : caractérisation

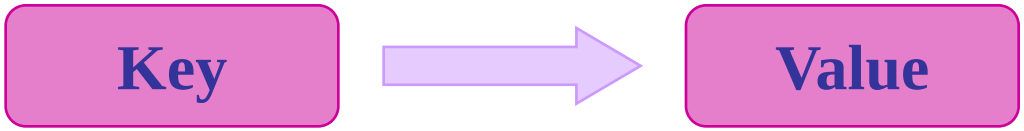



# NoSQL : caractérisation




# NoSQL : Clé-Valeur

Grandes tables de hachages




# NoSQL : Clé-Valeur

- Caractéristiques :
  - Simples
  - Interface pauvre (peu de commandes)
  - Très performantes
- Remarques :
  - Partitionnement des données pas toujours pris en charge par la DB

# NoSQL : Clé-Valeur

- Pour que faire :
  - Cache d'un site web
  - Stockage de données élémentaires

**NoSQL K/V**

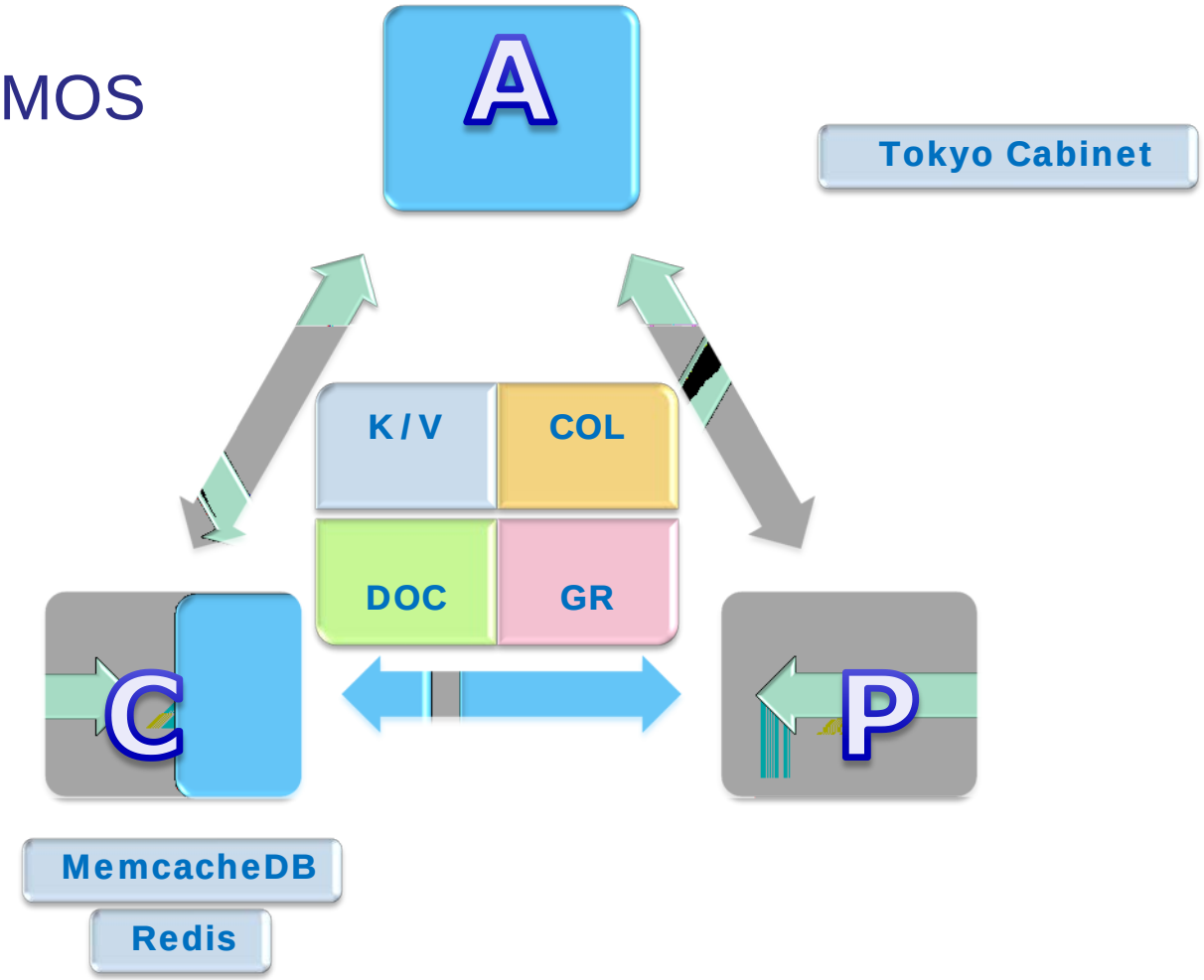
| Clé | Valeur |
|---|---|
| A | B |
| C | D |

**RDBMS**

| Clé | Valeur |
|---|---|
| A | B |
| C | D |

# NoSQL : Clé-Valeur

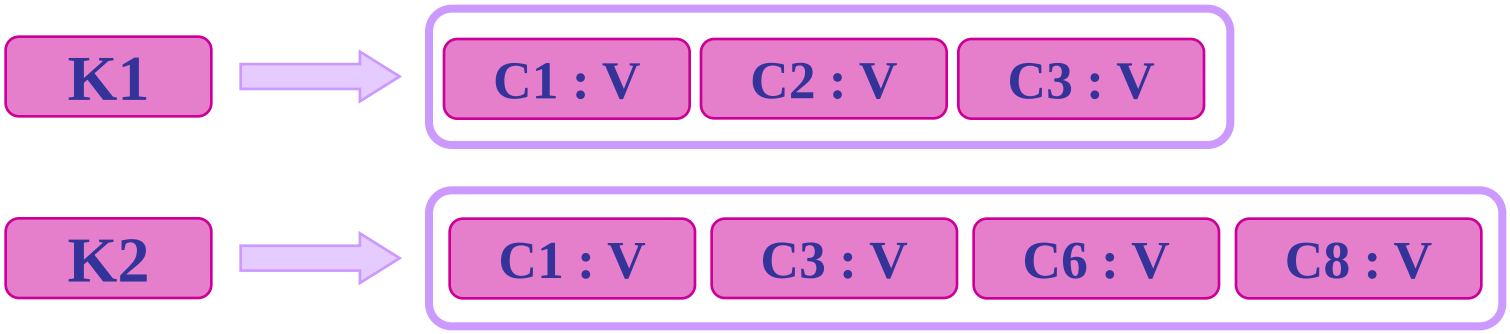
- DEMOS



# NoSQL : Solutions orientées colonnes

## Longues listes variées

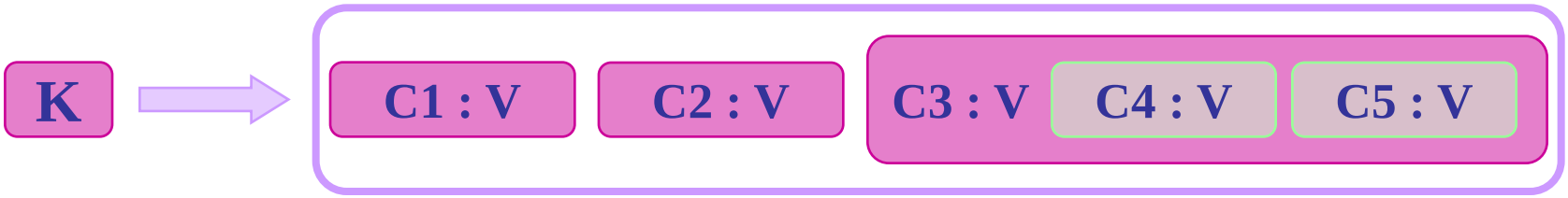
K = Key
C = Column
V = Value




# NoSQL : Solutions orientées colonnes

Parfois possible d'avoir un niveau de hiérarchie supplémentaire (super colonnes)

| | | |
|---|---|---|
| K | = | Key |
| C | = | Column |
| V | = | Value |



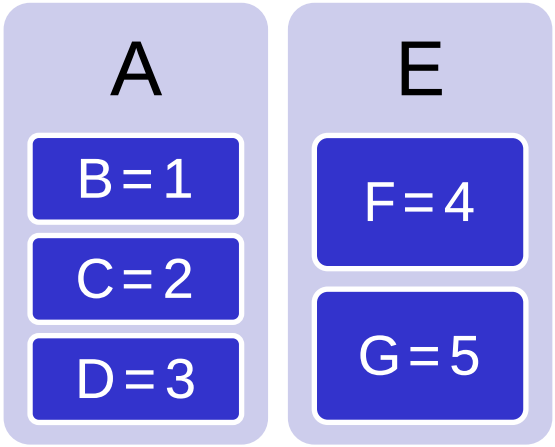

# NoSQL : Solutions orientées colonnes

- Caractéristiques :
  - Gestion d'un très grand nombre de colonnes (plusieurs millions)
  - Requêtes minimalistes
- Remarques :
  - Absolument pas prévu pour le relationnel ➔ difficile à exploiter pour les gens habitués aux RDBMS

# NoSQL : Solutions orientées colonnes

- Pour que faire :
  - Stockage de données chronologiques (ex : log système, evolution d'un dossier médical)
  - Relations one-to-many (ex : gérer des droits d'accès)

**NoSQL Col**

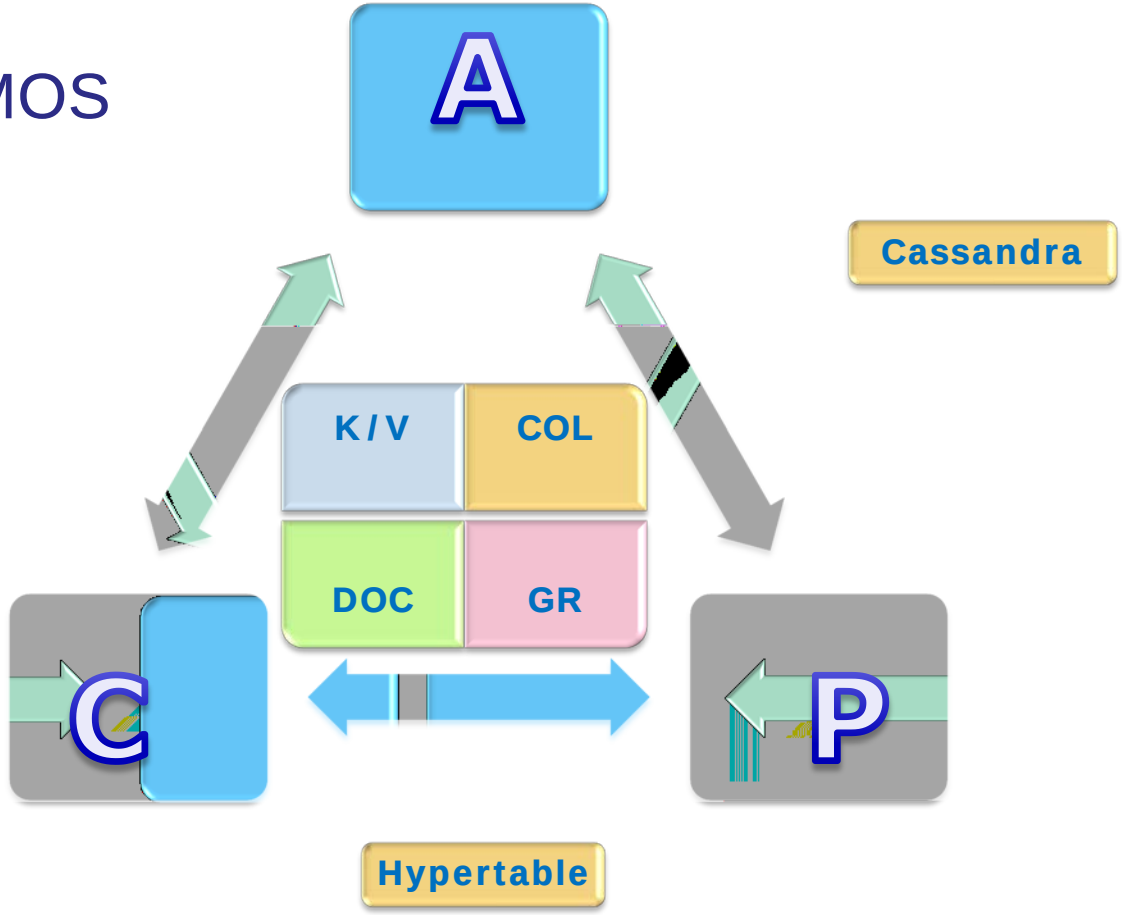



**RDBMS**

| Node | Father | Value |
|---|---|---|
| A | NULL | NULL |
| E | NULL | NULL |
| B | A | 1 |
| C | A | 2 |
| D | A | 3 |
| F | E | 4 |
| G | E | 5 |

# NoSQL : Solutions orientées colonnes

- DEMOS



# Pause

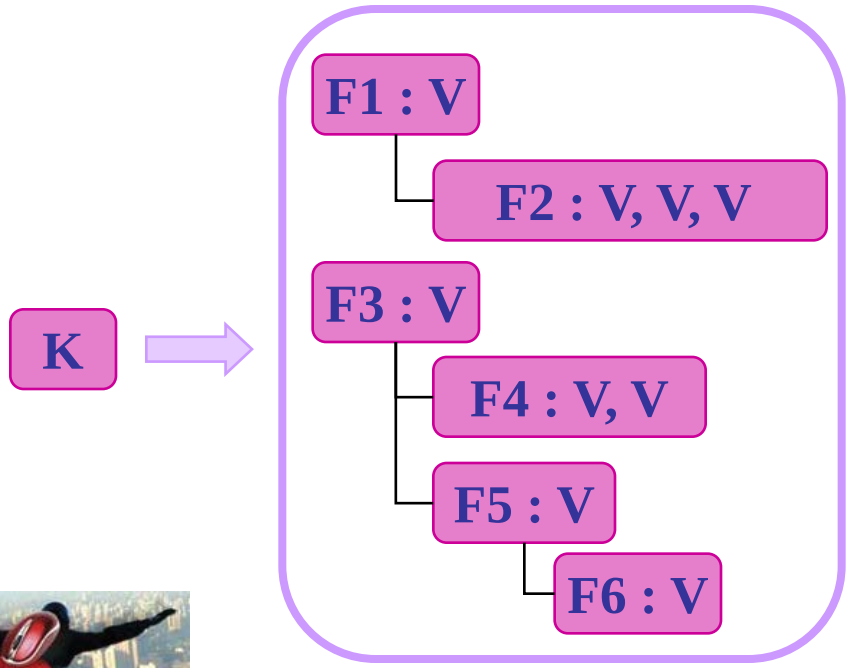
```
A := #persons_before_pause
B := #persons_after_pause
IF A > B THEN
        presentation_quality := bad
ELSE
        presentation_quality := normal
        IF B > A THEN
                attenders_punctuality := bad
```

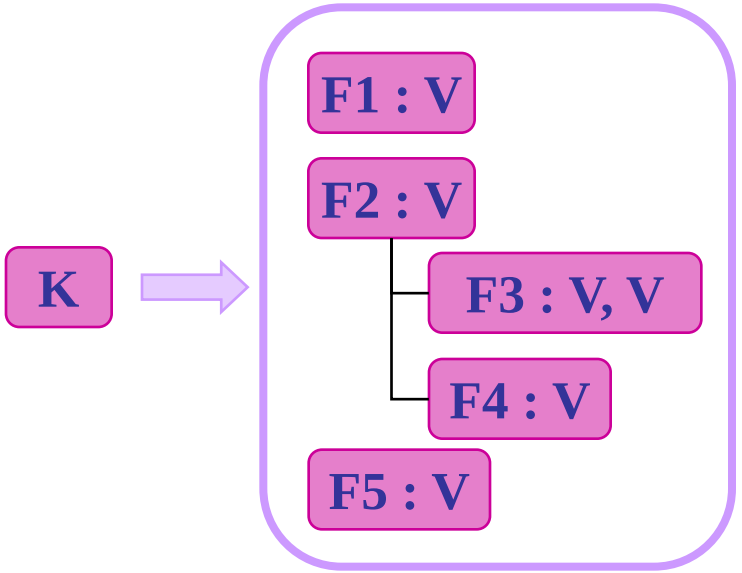
INSERT "Coffee" INTO "Hall";

# NoSQL : Solutions orientées documents

Documents structurés sans schémas prédéfinis

F = Field
V = Value
K = Key



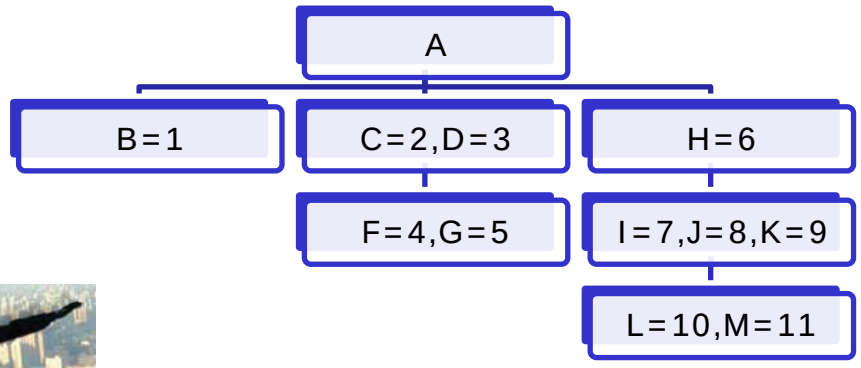

# NoSQL : Solutions orientées documents

- Caractéristiques :
  - Très flexibles
  - Syntaxe parfois très riche
  - Bien adapté au monde du Web

# NoSQL : Solutions orientées documents

- Pour que faire :
  - Gérer des données avec des structures complexes et variables (DMFA, profils utilisateurs complexes)
  - Cache de données structurées (manipulation de formulaires)

**NoSQL doc**



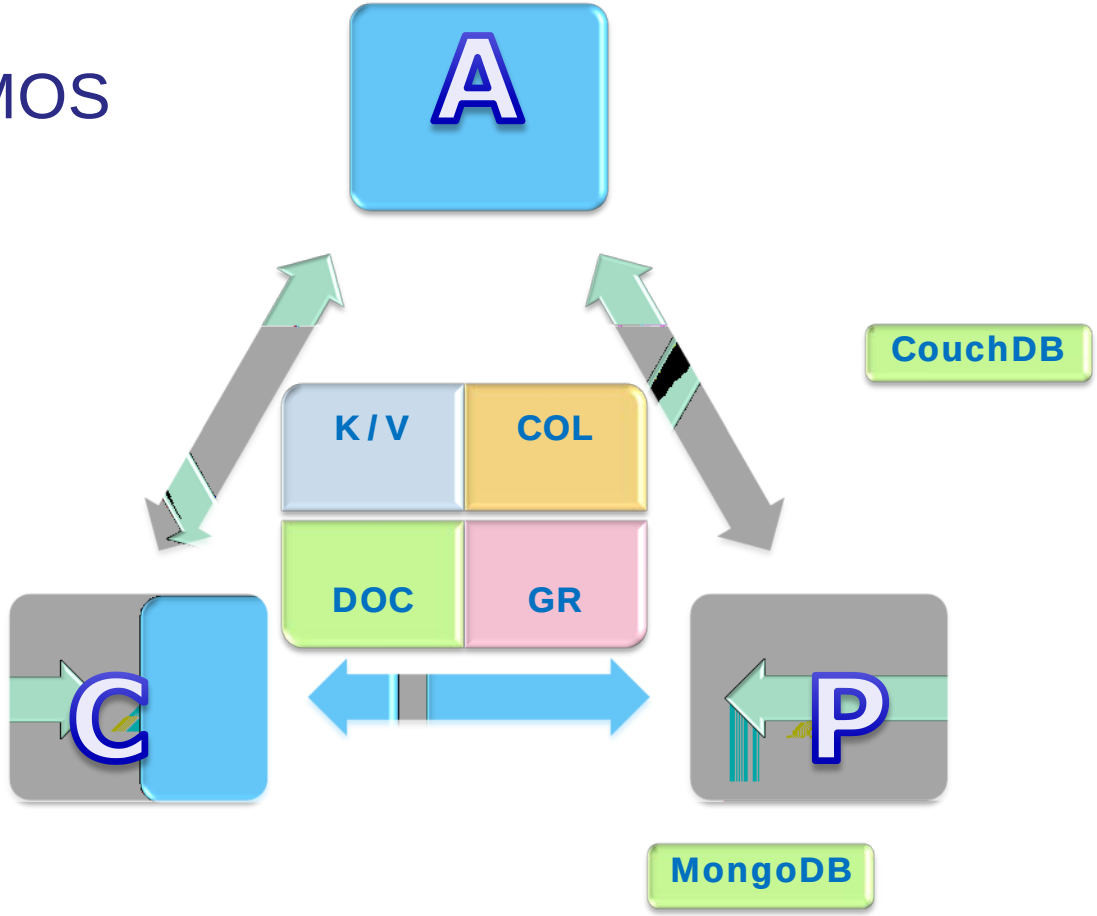

| Node | Father | Value |
|---|---|---|
| A | NULL | NULL |
| B | A | 1 |
| CD | A | NULL |
| C | NULL | 2 |
| D | NULL | 3 |
| FG | CD | NULL |
| F | NULL | 4 |
| G | NULL | 5 |
| H | A | NULL |
| IJK | H | NULL |
| I | NULL | 7 |
| J | NULL | 8 |
| K | NULL | 9 |
| LM | IJK | NULL |
| L | NULL | 10 |
| M | NULL | 11 |

**RDBMS**

| Node | MemberOf |
|---|---|
| C | CD |
| D | CD |
| F | FG |
| G | FG |
| I | IJK |
| J | IJK |
| K | IJK |
| L | LM |
| M | LM |

# NoSQL : Solutions orientées documents

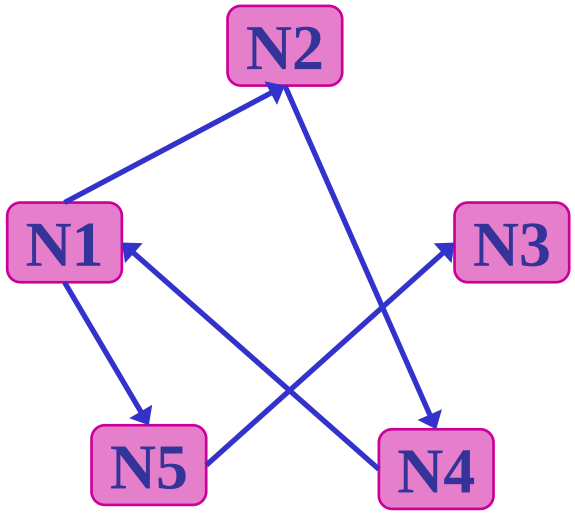
- DEMOS



# NoSQL : Solutions orientées graphes

N = Noeud

## Graphes orientés



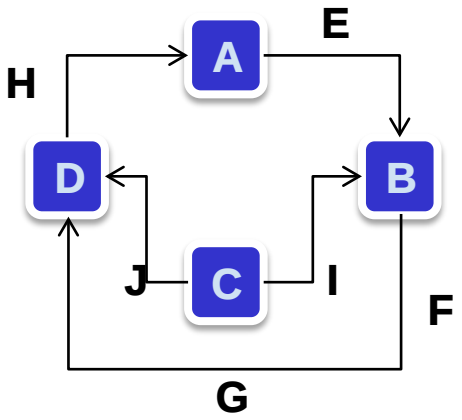

# NoSQL : Solutions orientées graphes

- Caractéristiques
  - Parfois optimisées pour la résolution de problèmes complexes et non pour les performances
- Remarque
  - Peu de solutions intéressantes à ce jour

# NoSQL : Solutions orientées graphes

- Pour que faire :
  - Gérer des relations complexes et dynamiques (ontologies, relations entre personnes, administrations, ...)

**NoSQL graph**




**RDBMS**

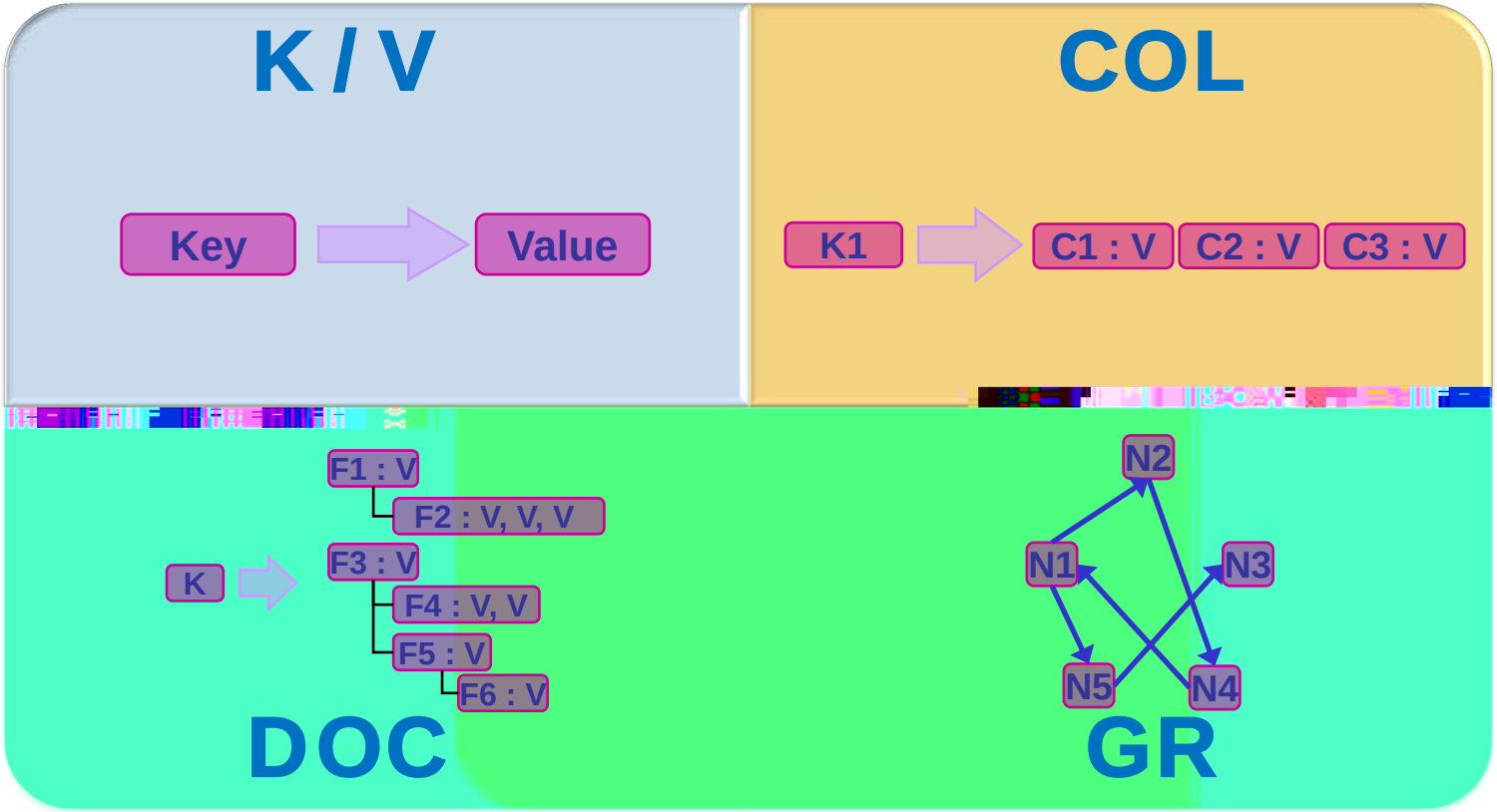
| Src | Dst | Rel |
|---|---|---|
| A | B | E |
| B | D | G |
| C | B | I |
| C | D | J |
| D | A | H |

# NoSQL : Orientées graphes

© http://www.foxmovies.com/

- DEMOS

# NoSQL : Récapitulatif




# NoSQL : Récapitulatif

## K / V

**Cache site Web**
**Stockage données élémentaires**

## COL

**Données chronologiques (log système, dossier médical)**
**Relations One-to-Many**

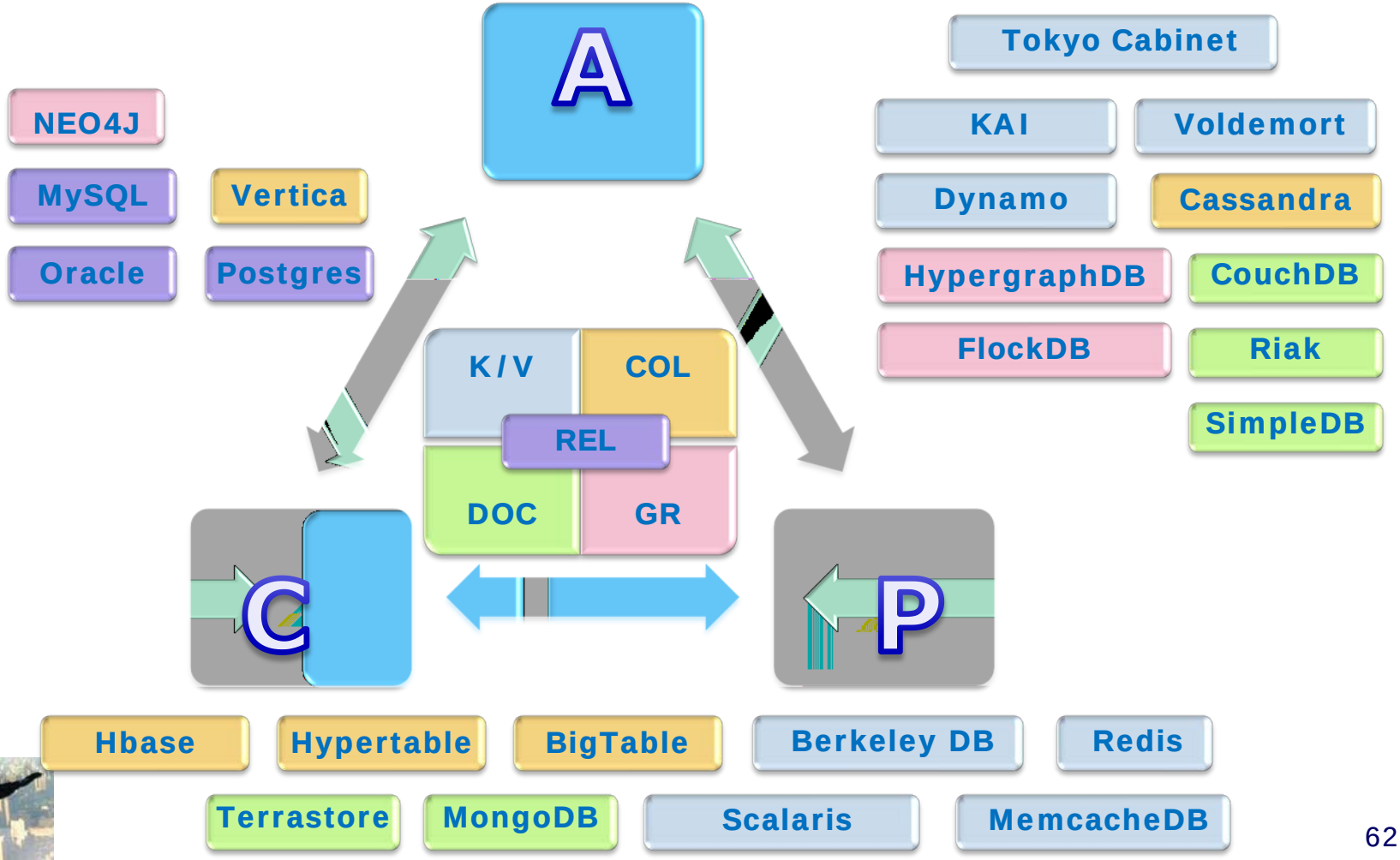
**Documents complexes (DMFA, profils utilisateurs, cache de données structurées )**

## DOC

**Gérer des relations complexes et dynamiques (ontologies, relations personnes/administrations)**

## GR

# NoSQL : Récapitulatif




# Ce que nous avons constaté

- Outils très spécifiques et efficaces dans leur domaine
- Maturité très variable d'une solution à l'autre
- Installation et prise en main aisées
- Peu ou pas d'outils tiers (incompatible avec l'existant ?)

# Ce que nous avons constaté

- Se méfier des benchmarks :
  - Solutions très spécifiques
  - Nécessité de réaliser des benchmarks exploitant les spécificités des bases de données testées
  - Topologie réseau propre à chaque DB

# Ce que nous avons constaté

- Ne pas négliger la documentation
- Limitations parfois inattendues :
  - 4Mo pour
    un objet MongoDB,
  - 1Mo pour
    un object Memcached
  - Pas de redéfinition
    de colonnes
    dans Cassandra
  - ...

# Ce que nous avons constaté

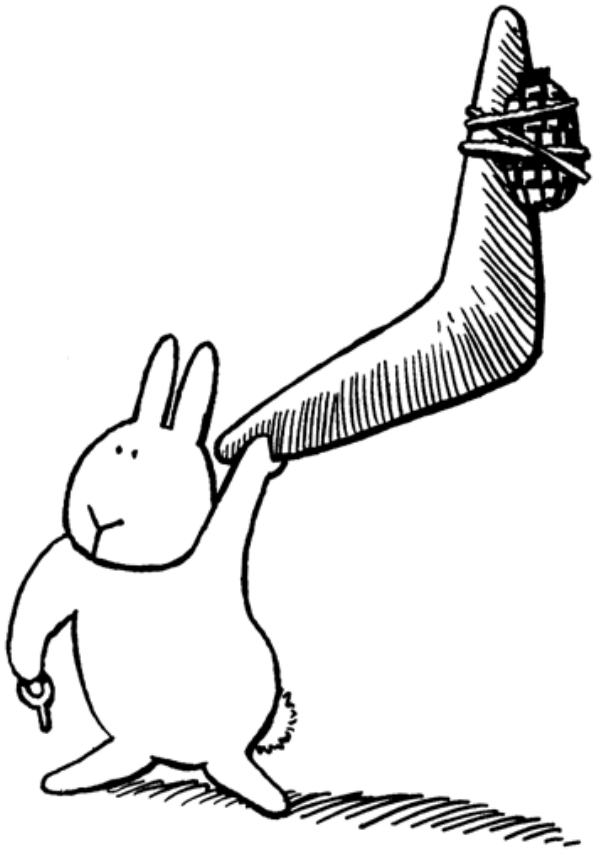
- Réception auprès des utilisateurs

- Incompréhension

- Réticence

# Recommandations

- NoSQL n'est pas un objectif en soi
- Ne répond pas à tous les problèmes
- A utiliser uniquement si les SGBDR ne conviennent pas :
  - Trop lents
  - Données difficiles à modéliser
  - Données difficiles à exploiter
  - Format des données variable

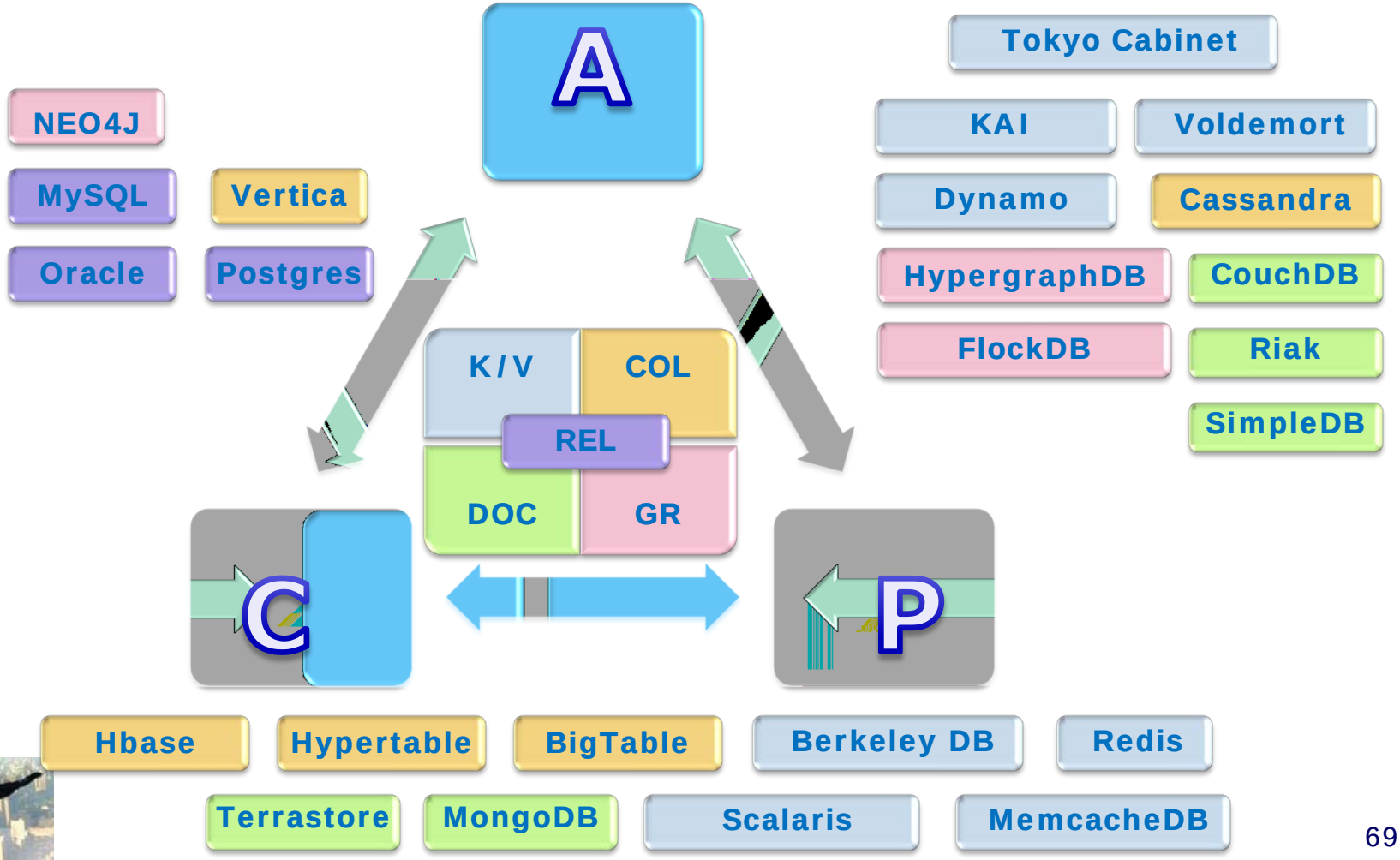




# Recommandations

- Adapter la BD au format des données et non le contraire
- NE PAS PENSER RELATIONNEL (surtout lors de la définition du modèle de données)
- Peut être utilisé conjointement à un RDBMS

# Recommandations (Choix d'une solution)




# Conclusion

- Bases de données NoSQL : une alternative sérieuse au modèle relationnel ?
- Alternative : non ! (permettent de résoudre des problèmes différents)
- Sérieux : oui ! (même Oracle va proposer sa solution NoSQL)
- A utiliser si les bases de données relationnelles se montrent inefficaces/trop complexes.

# Conclusion

- Beaucoup de solutions, mais très spécifiques
- Ne pas négliger la formation et la documentation
- Ne pas toujours penser relationnel
- Ne pas toujours penser relationnel !

# Conclusion

- NoSQL : hype ou innovation ?
- Hype et innovation
- Potentiel réel et non négligeable
- Téléchargez, jouez un peu, faites-vous une idée avant d'essayer ou de dire non ;-)

# Questions ?

SELECT "Questions" FROM "Public";

DELETE FROM "Public";