www.cnrs.fr

MongoDB

Une introduction

Sommaire

Historique

Historique

- **Initialement développé par la société 10gen en 2007**
  - 10gen a été rebaptisé en 2013 MongoDB Inc.
  - MongoDB provient de l'anglais « Humongous » (« énorme »)
  - Acquisition de la société WiredTiger en décembre 2014 (moteur de stockage)
- **SGBD NoSQL open source (licence AGPL V3.0)**
  - Ecrit en C++
- **Stocke ses données au format BSON (Binary JSON)**
  - Créé dans le but de fournir un compromis entre les bases de données clés/valeurs et les bases relationnelles classiques
- **Utilisé par des grands comptes comme Cisco, Google, Orange, Bouygues ou encore le CERN**
- **Dernière version stable : 3.4.3 en avril 2017**
- **Beaucoup d'interfaces avec les langages de programmation**
  - Java, PHP, … et Python !

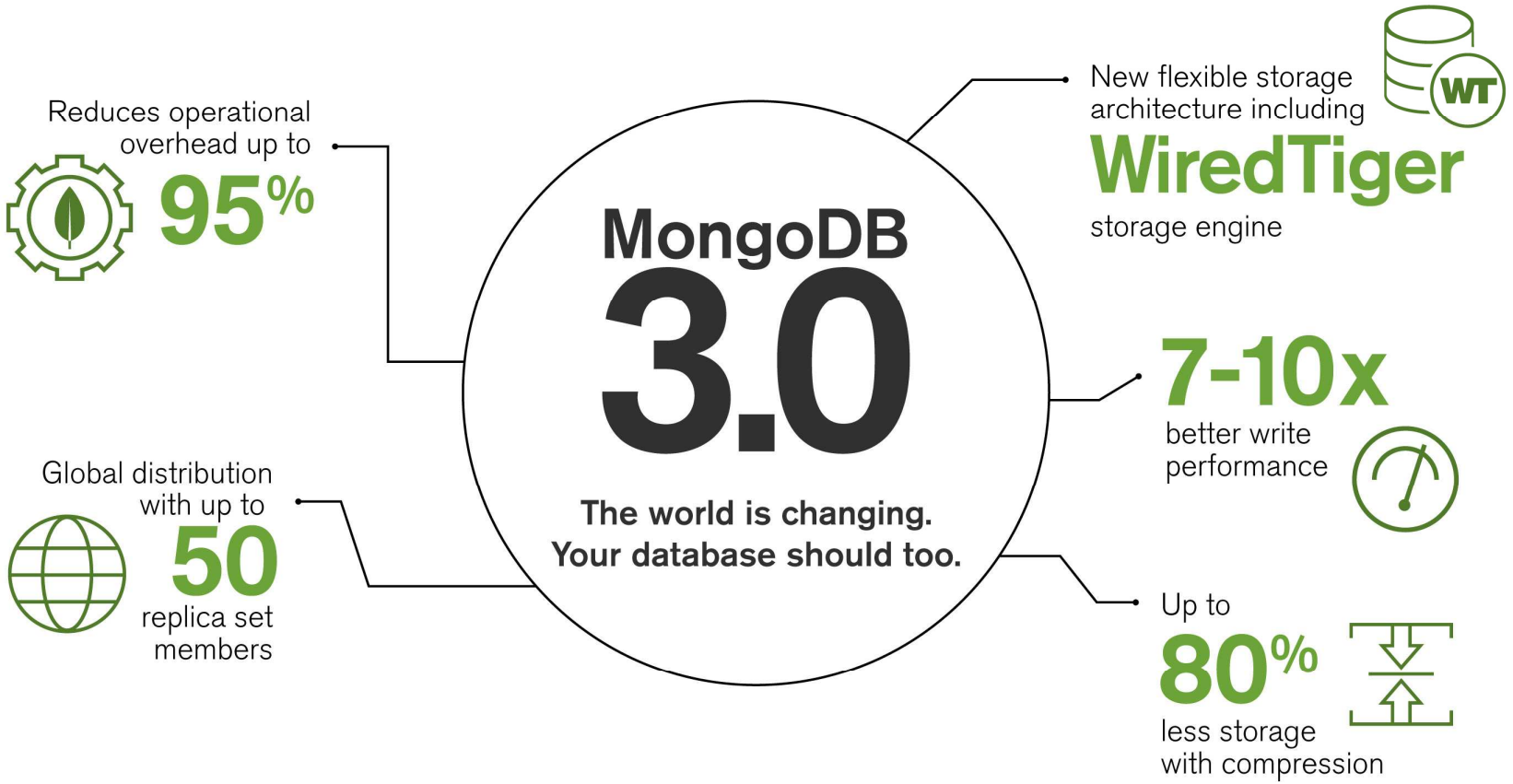
Historique

Nouveautés de la version 3

- Un nouveau moteur de stockage WiredTiger
- Verrous au niveau des collections (MMAPv1, WiredTiger) ou des documents (uniquement WiredTiger) plutôt qu'au niveau global (database)
- Compression des données stockées (WiredTiger)
- Un quorum élargi à 50 « Replica Set » (au lieu de 12)
- Un nouveau mécanisme d'authentification SCRAM-SHA-1 (remplace Mongo-CR)
- Amélioration des traces (catégorisation)
- Performances accrues pour les index

Historique

Synthèse des apports de la version 3

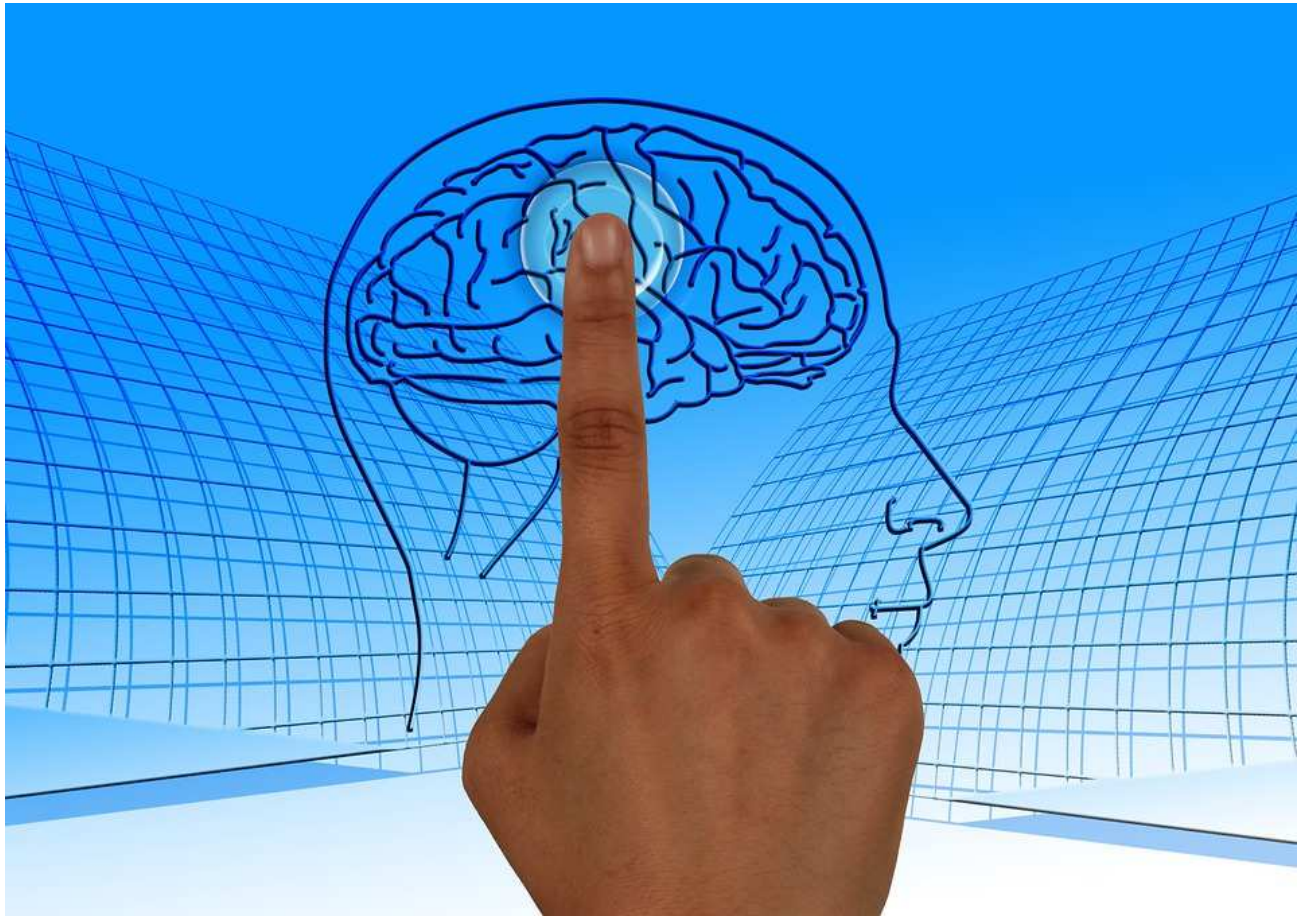


Source : http://blog.ippon.fr/wp-content/uploads/2015/02/FINAL-3.0Launch-Infographic-v7-1.jpg

Historique

- **Quelques remarques :**
  - Forte concurrence dans le secteur des bases NoSQL
  - Comme toutes les sociétés basant leurs modèles sur « l'Opensource », MongoDB Inc. complète son offre avec des outils spécifiques (interface d'administration Compass par exemple) ou du support … payants
  - MongoDB Inc a une forte stratégie d'innovation :
    - Beaucoup de versions se succèdent
    - Le contour fonctionnel du produit évolue fortement (et intègre même des notions de graphes par exemple, ou de la BI, des éléments renforcés en termes de sécurité ainsi que des connexions vers des environnements Spark)
    - Offre de services Cloud

Quelques rappels



Quelques rappels

- **Les données**
  - le volume augmente (plus ou moins rapidement)
  - la nature des données varie dans le temps ……
- **Les données numériques créées dans le monde seraient de :**
  - 1,2 zettaoctets par an en 2010 à 1,8 zettaoctets en 2011,
  - 2,8 zettaoctets en 2012
  - et s'élèveront à 40 zettaoctets en 2020 (source Wikipédia)
- **Le radiotélescope “Square Kilometre Array” par exemple, produira 50 teraoctets de données traitées par jour à partir d’un rythme de 7 000 teraoctets de donnée brutes par seconde (source : http://ercim-news.ercim.eu/en89/special/managing-large-data-volumes-from-scientific-facilities)**

Quelques rappels

- **Que ce soit en physique des particules (LHC), en astronomie (par exemple Sloan Digital Sky Survey : www.sdss.org) ou en biologie, les volumes de données générés sont considérables**
  - Selon Cisco, 966 exa-octets par an de transfert de données sur le réseau en 2014 (1 Exa-octet en 2004) (pour mémoire 1 exa-octet : 1018 octets et 1 peta-octet : 1015 octets)
  - 1,790 milliards à 2,230 milliards de personnes connectées en 2012
  - quelques chiffres généraux ici : http://netforbeginners.about.com/od/weirdwebculture/f/How-Big-Is-the-Internet.htm et là http://hmi.ucsd.edu/pdf/HMI_2009_ConsumerReport_Dec9_2009.pdf
  - Les échelles sont désormais l'exa-octet (1018) et le zéta-octet (1021)
- **Le nouveau défi est de pouvoir traiter ces volumétries**
- **Par exemple : recherche d'informations ou recherche de corrélations statistiques entre données**

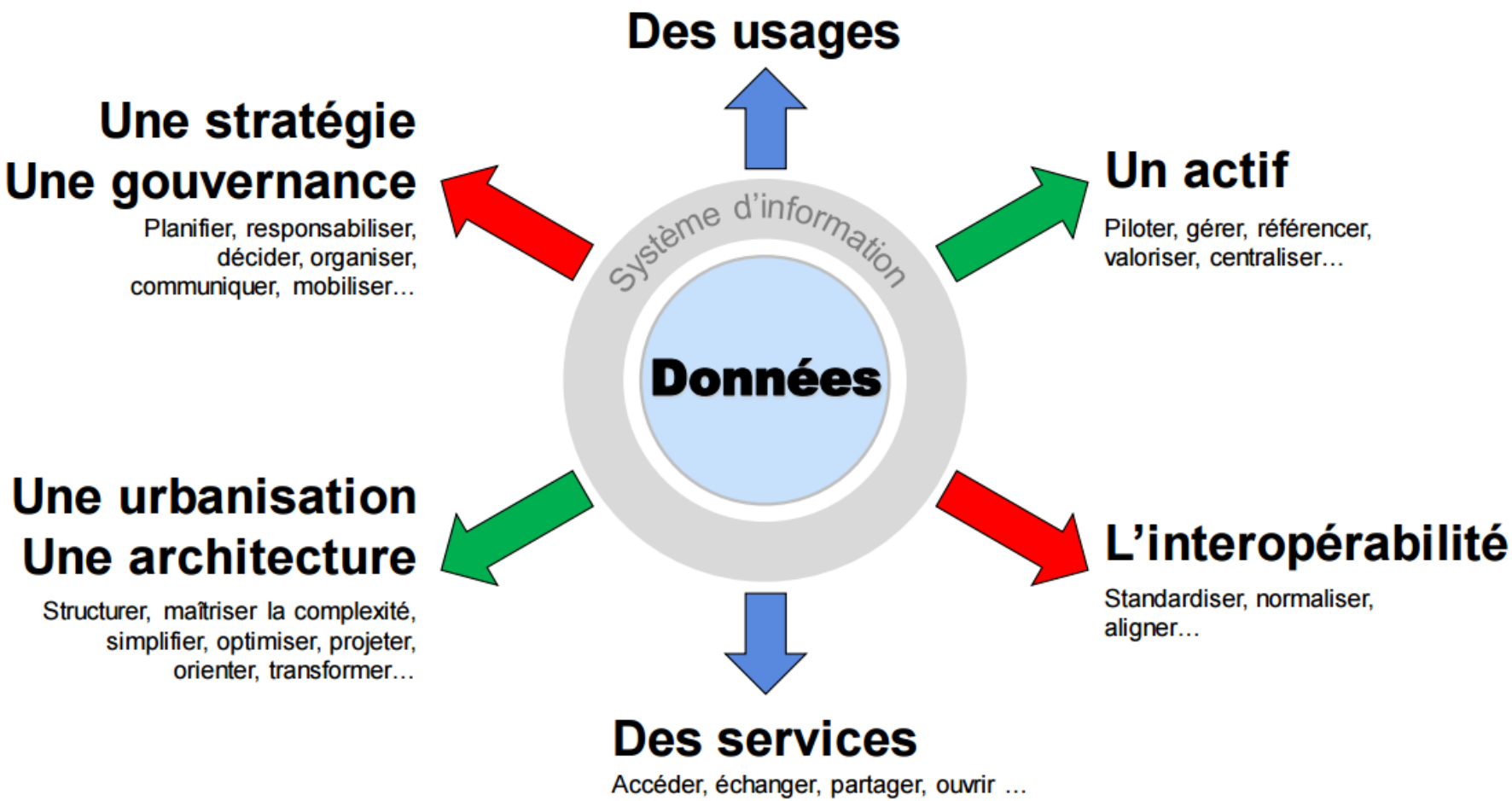
Quelques rappels

- **Les données**
  - le volume augmente (plus ou moins rapidement)
  - la nature des données varie dans le temps ……
- **Donc, rapidement, si rien n'est fait …**

Source : http://sharkoliv.blogg.org/mulet-bien-charge-a115760130

Quelques rappels

La gouvernance des données vue par la DINSIC



Quelques rappels

- **Etablir un Data Management Plan est fondamental !**

Cycle de vie des données
(Jones S, 2015)

Quelques rappels

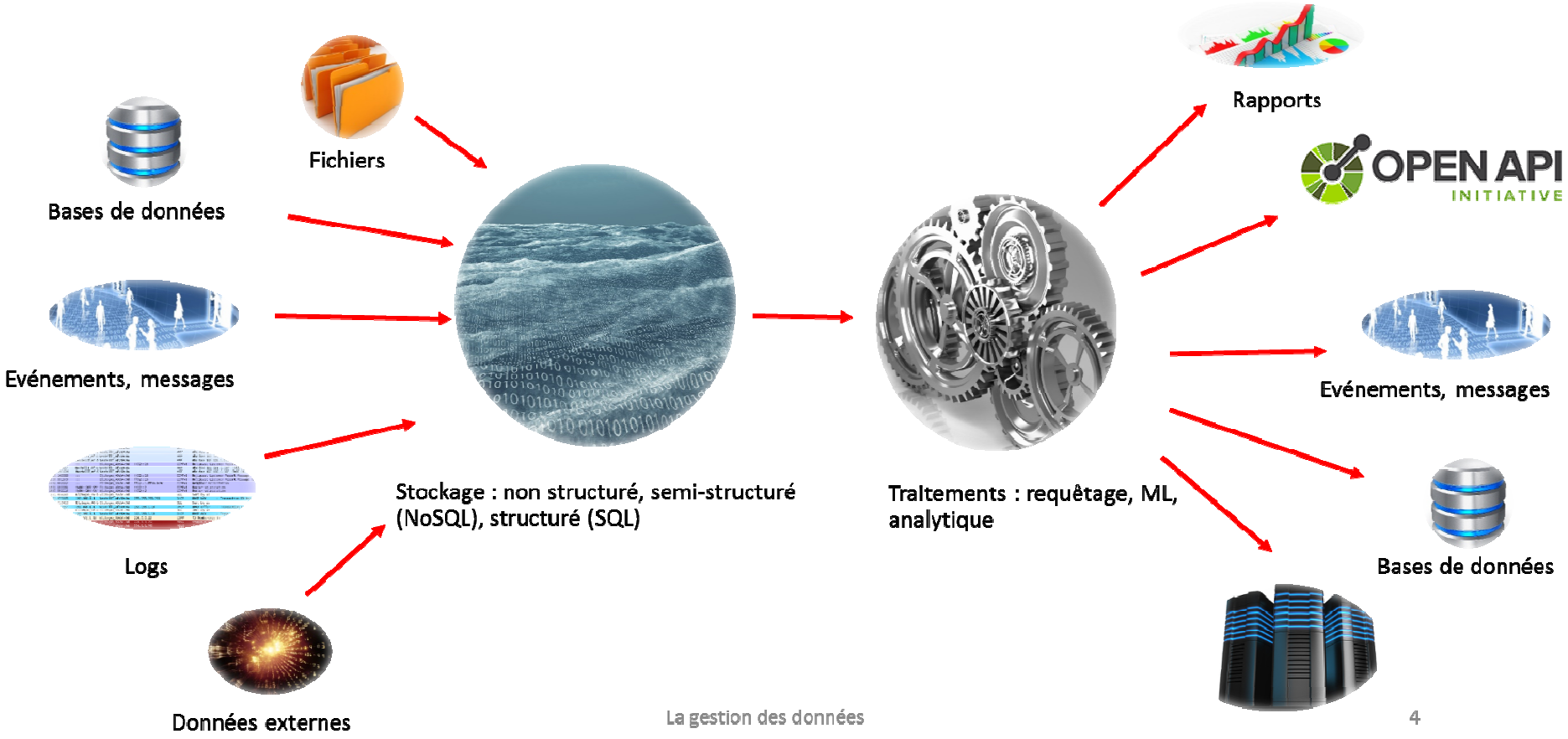
- **Structure générale d'un DMP**
  - Données administratives (description formelle qui, quoi, où, etc.)
  - Responsabilités et ressources (Qui est responsable, quels outils, etc.)
  - Collecte/création des données
  - Documentation et métadonnées
  - Stockage, sauvegarde et sécurité des données
  - Ethique et cadre légal
  - Partage des données
  - Sélection et archivage des données

D'après Cartier et al (2015), Deboin (2014), Digital curation Centre (2013), European Commission (2013) [Traduction française]

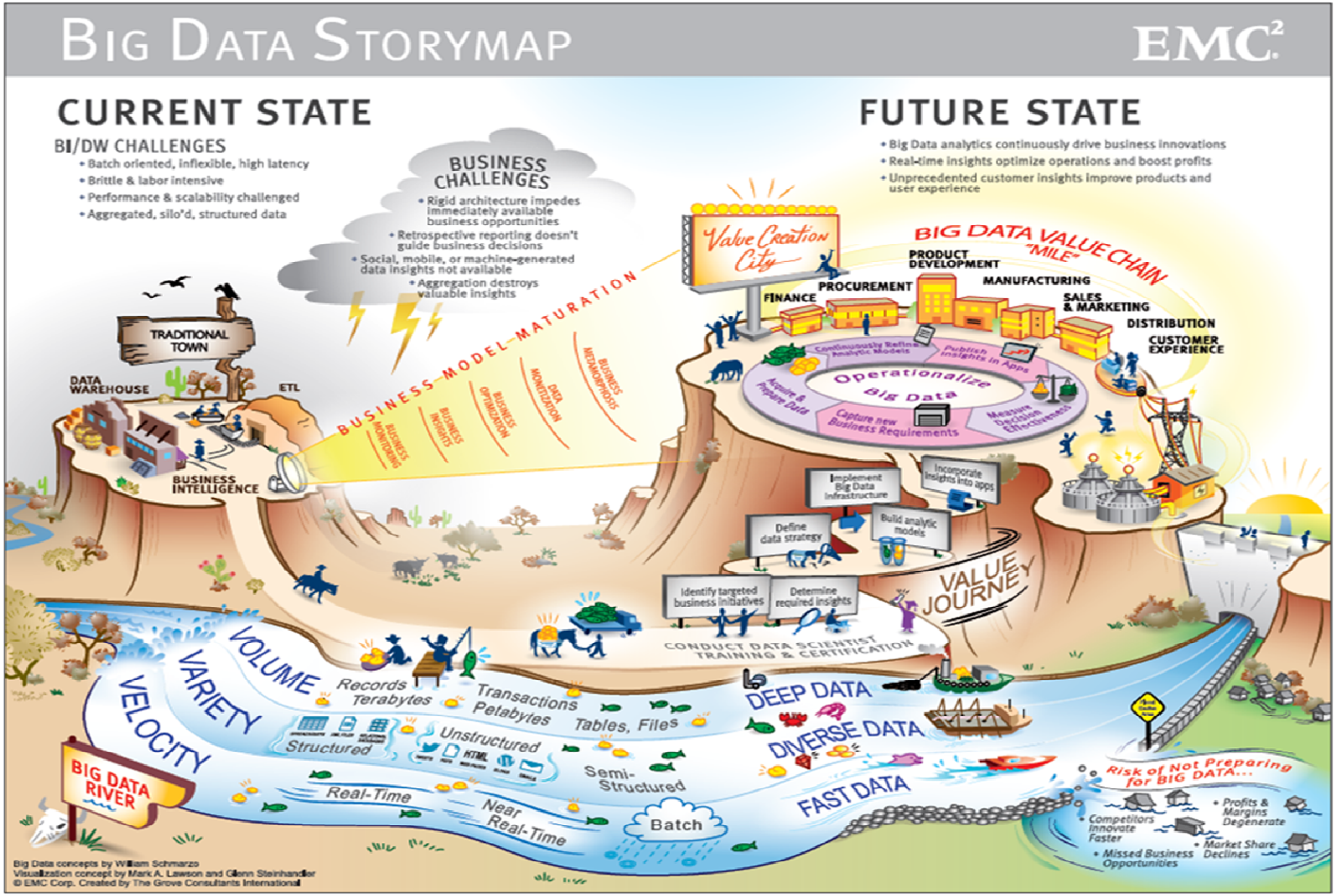
- **Un très bon document sur le site rBDD :**
  http://rbdd.cnrs.fr/IMG/pdf/jrbdd_atelierdmp_20151102.pdf?140/846e9cf5966f7af17e4806020cdcfadbf384edc7

Quelques rappels

Le « Data Lake »



Quelques rappels



Quelques rappels

- Big Data ou analytique avancée ?
  - Règle générale : attention aux sirènes du marketing …

Ici, traité avec beaucoup d'humour !

Quelques rappels

Big Data ou analytique avancée ?

- Les données sont désormais un actif à exploiter
  - « Make Me More Money », les quatre M du Big Data
- Pour faire cela, il faut :
  - Stocker des données
  - Exploiter les données
- Mais ces composantes sont valables à la fois pour l'analytique et le Big Data …
- Où se situe alors la différence ?

Source : http://sharkoliv.blogg.org/mulet-bien-charge-a115760130

Quelques rappels

Big Data ou analytique avancée ?

- La différence se situe sur les ordres de grandeur :
  - Du nombre de paramètres, p
  - De la taille des échantillons (au sens statistique), n
- Et donc :
  - En analytique : p = 1 à 10 ou à peu près et n = 1000 à 10000
  - En Big Data : p = potentiellement plusieurs centaines voire milliers et n = plusieurs centaines de milliers voire millions ou milliards. La valeur de p est en générale petite par rapport à n
- Le nouveau défi est de pouvoir traiter un nombre important de paramètres (ou caractéristiques) p
- Les méthodes statistiques traditionnelles ne sont plus réellement adaptées dans certains cas (p et n très grand)
  - Emergence de la science des données et du Machine Learning
  - Mise au point d'algorithmes ou d'heuristiques, pas toujours prouvés, mais efficaces

Quelques rappels

- **Big Data ou analytique avancée ?**
  - La traduction automatique est un exemple de succès de la science des données
    - Thématique très longtemps peu évolutive du fait de tentatives de formalisation du langage
      - Peu de résultats probants
    - Avancées spectaculaires de la part de Google en utilisant des paires « texte d'origine » //« texte traduit » à grand échelle (n très grand)
      - La traduction est « statistique » … mais avec une grande précision !
  - La reconnaissance d'images fait aussi partie des succès de la science des données

Quelques rappels

- **Big Data ou analytique avancée ?**
  - En conclusion, analytique et Big Data se différencient :
    - Par les valeurs de p et n
      - Si n * p est grand, c'est du Big Data
    - Par les méthodes utilisées, statistiques « classiques » dans un cas et algorithmes ou heuristiques ad hoc dans l'autre cas
    - Au niveau des volumétries de données, en pratique :
      - > 10 To
    - Avec des caractéristiques particulières : les 3 V
  - Définition officielle : journal officiel du 22 août 2014
    - Préconise d'utiliser en français le mot mégadonnées
    - Définition : données structurées ou non dont le très grand volume requiert des outils d'analyse adaptés

Quelques rappels

Quelques mises en garde

- Le paradoxe ici est que, sans théorie, des résultats sont obtenus
- Il est possible de traduire sans maitriser une quelconque théorie du langage !
- Mais cela a des limites :
  - Il faut prêter attention aux données et à leurs traitements
    - Attention aux biais …
  - Les traitements sur les données ne sont pas toujours d'une grande rigueur et le Machine Learning comprend une grand part « d'inventivité » : créations de variables, etc.
- Dans tous les cas, il faut être vigilant sur le contexte méthodologique et conserver un regard critique sur la démarche et les résultats obtenus
- Mais la profusion des données, traitées avec précaution, va permettre de guider des choix et de formuler des hypothèses sur des problèmes complexes (impact fort sur les processus de décision)

Quelques rappels

Datascience et Machine Learning

- Le « Data Scientist » est l'expert de la donnée. Il travaille avec des outils de fouille de données, de prototypage, etc. Son objectif est de visualiser, expliquer et mettre en évidence des corrélations entre les données (beaucoup de statistiques en perspectives …)

- L'expert en Machine Learning cherche à construire des systèmes pouvant « apprendre » à partir des données, ou du moins faire des prédictions avec des probabilités élevées ou jugées comme suffisante par rapport à l'objectif

Quelques rappels

Indice de maturité du modèle Big Data

- Il existe des indices de maturité du Big Data dans une organisation
- Ces indices varient en fonction du secteur d'activité, du pays, etc.
- Des nombreuses études existent sur le sujet
- Quelques indices (à adapter au contexte) :
  - Volume des données, stratégie de stockage (silos, « dark datas only », etc.),
  - Métiers de la Donnée (BI, autre) : carence dans les métiers de l'analytique ?
  - Existence d'outils
  - Services autour de la Donnée, transversalité de la question
  - Stratégie autour de la Donnée, implication managériale, existence d'un ROI sur le sujet
  - Degré de sensibilité à la sécurité des données
  - Etc.

Quelques rappels

Pour résumer

- L'analytique et le Big Data sont distincts
  - Le nombre de paramètres et la taille de la population les distinguent
- La prise en compte du Big Data est une nécessité
  - Pour la création de valeur
  - Pour faire face aux évolutions
- Le Big Data implique :
  - Une nouvelle approche des données
  - De nouvelles technologies
  - Et une implication au plus haut niveau de l'organisation !
- Le temps des SI est révolu, le temps de l'entreprise numérique est déjà là !

Quelques rappels

Les évolutions vues par le Gartner

Source : http://www.gartner.com/newsroom/id/3114217

Quelques rappels

- Des questions ou remarques ?

Quel(s) modèle(s) ?

- **Les solutions basées sur le modèle relationnel ne sont plus adaptées**
  - Temps de réponse trop longs, surtout sur les jointures qu'il est difficile de paralléliser
  - Modèle de conception impliquant des propriétés ne correspondant pas aux exigences : jointures, transactionnel, etc.
- **Deux options sont envisageables**
  - Scalabilité verticale : on augmente la capacité du serveur jusqu'à sa limite maximale … mais cela peut ne pas suffire
  - Scalabilité horizontale : on augmente le nombre de serveurs et la charge est répartie
- **Et, dans certains cas, le modèle relationnel doit être abandonné**
  - La notion de 'formes normales' doit être revue et les données 'dénormalisées' pour augmenter les performances et/ou mieux correspondre au besoin
- **Cela implique**
  - La suppression des jointures et la duplication de données
  - Cette duplication de données peut être plus ou moins importantes
    - Selon la variabilité des données
    - Les optimisations voulues en lectures/écritures

Quel(s) modèle(s) ?

- **Partitionnement horizontal sur plusieurs nœuds**

- **Distribution avec maître**
  - Réplication Maître-Esclave
  - Sharding
    - L'éclatement est géré automatiquement par le système
    - L'éclatement est obligatoirement distribué
    - La répartition de charge est possible

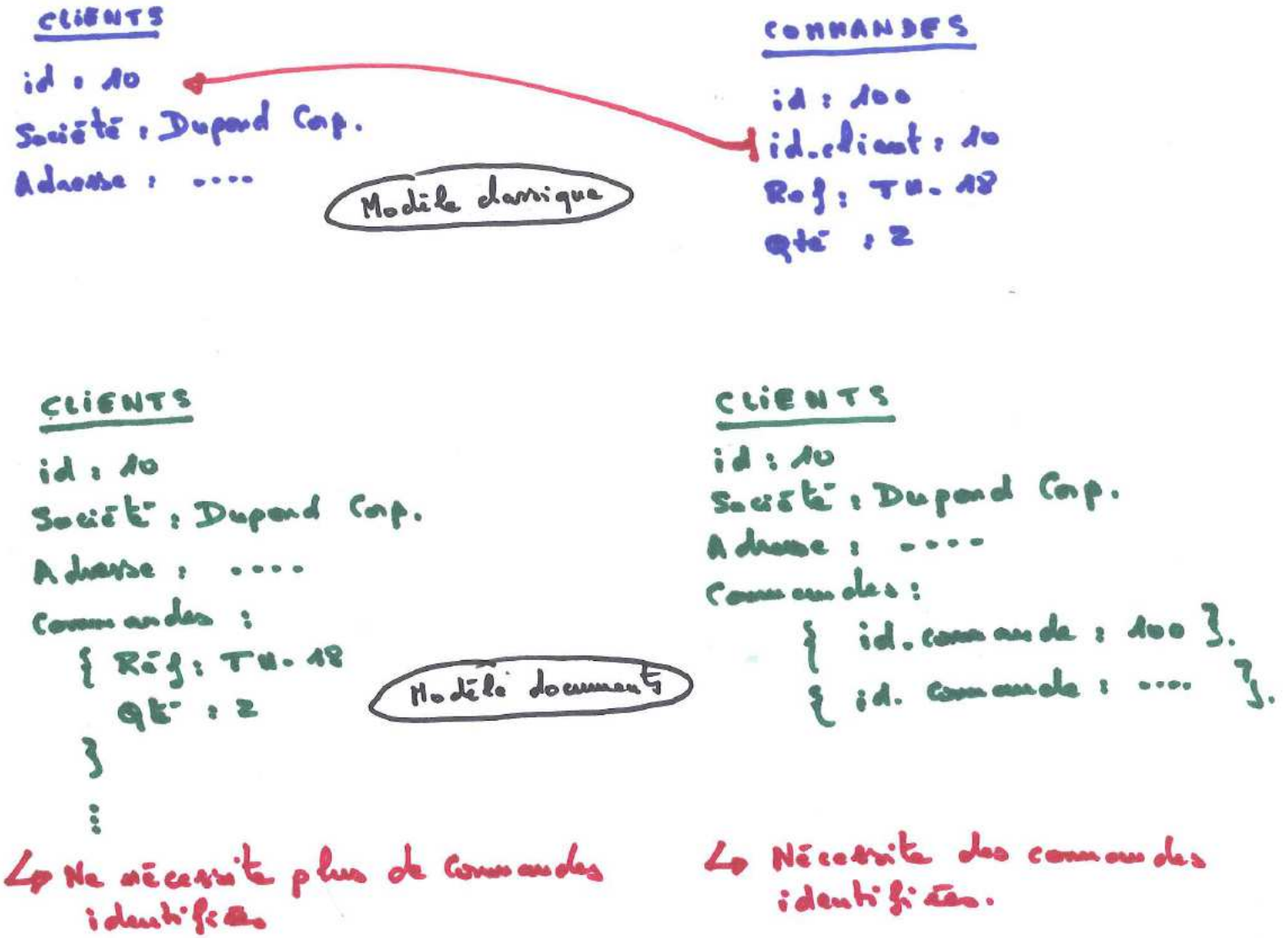
- **Distribution sans maître**
  - Réplication par bavardage
  - Répartition par distribution des clés

Quel(s) modèle(s) pour MongoDB ?

- **3 cas de figure sont envisageables**
  - A partir de 2 documents, un seul document « résultant » est construit
    - Il fusionne les 2 documents, en incluant l'ensemble des attributs
  - A partir de 2 documents, 2 documents subsistent
    - Un document se trouve augmenté des attributs ayant le plus faible taux de variabilité
    - Les autres attributs restent dans un document à part entière
  - A partir de 2 documents, 2 documents subsistent
    - Seuls les identifiants se retrouvent dans les documents où la mise en place d'une relation a un sens
    - Cette solution est proche d'une solution relationnelle classique
- **Il n'y a pas de solution privilégiée**
  - La solution retenue dépend avant tout du contexte
  - NoSQL (Not Only SQL) permet de s'affranchir des contraintes du modèle relationnel selon le besoin
  - **Attention** : SQL peut rester une excellente solution !

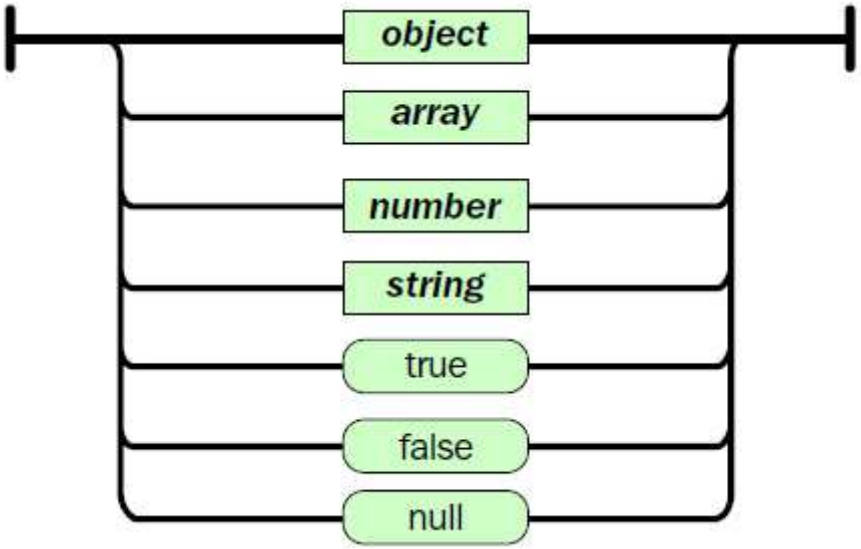
Quel(s) modèle(s) pour MongoDB ?

Illustration



Types de données

JSON, le format des documents



Types de données

JSON, le format des documents

- Un exemple :

Format clé/valeur

Début d'un document

```
{
"firstname" : "Mickael",
"lastname" : "Barroux",
"birth_date" : { "$date" : 616651200000 },
"address" : {
      "street" : "5, Main Street",
      "state" : "California",
      "city" : "Los Angeles",
      "country" : "US"
},
"phone_numbers" : [
      { "phone_number" : "0102030405" },
      { "phone_number" : "0203040506" }
]
}
```

Document dans un document

Début d'une liste

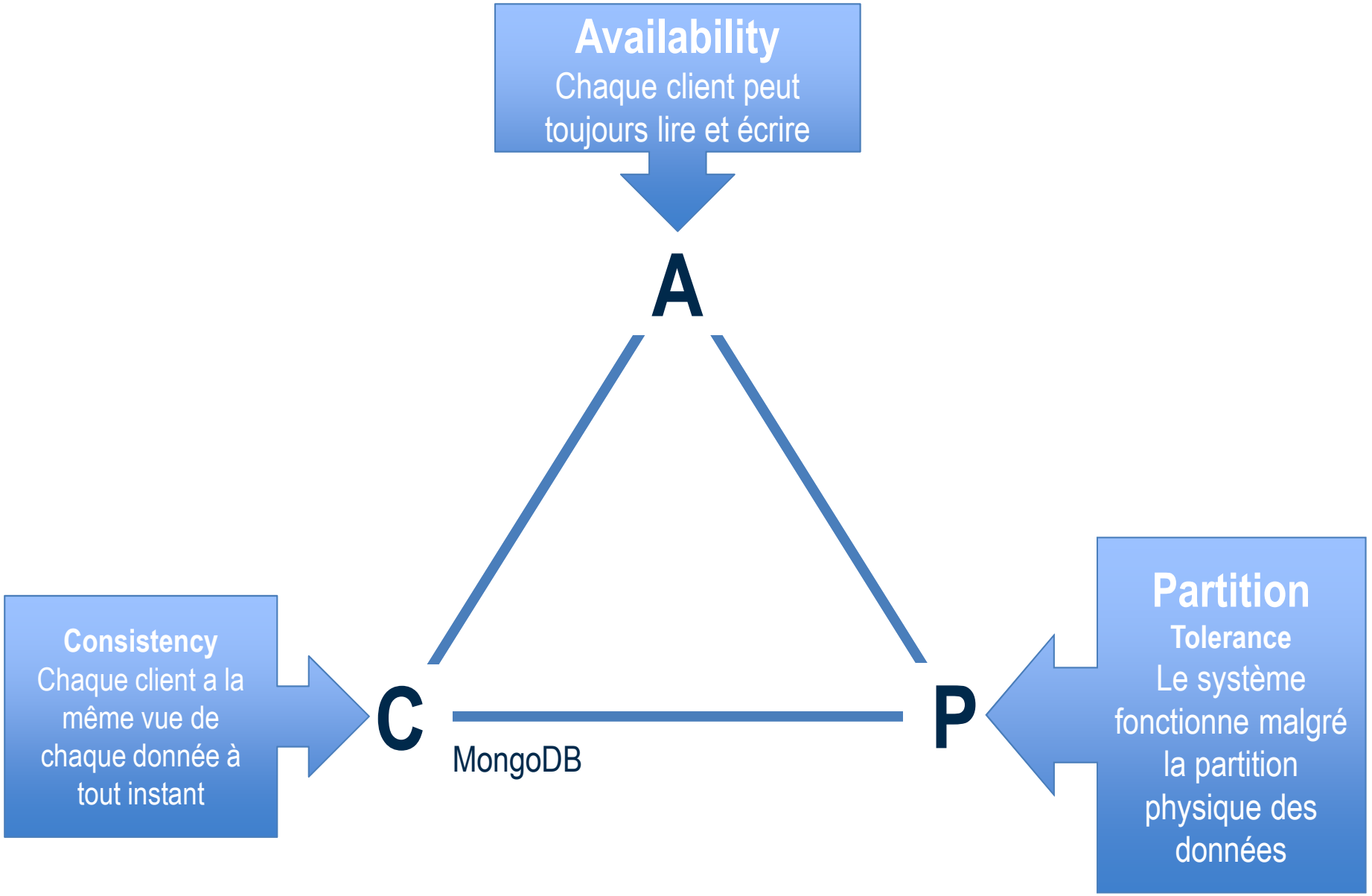
Fin d'une liste

Fin d'un document

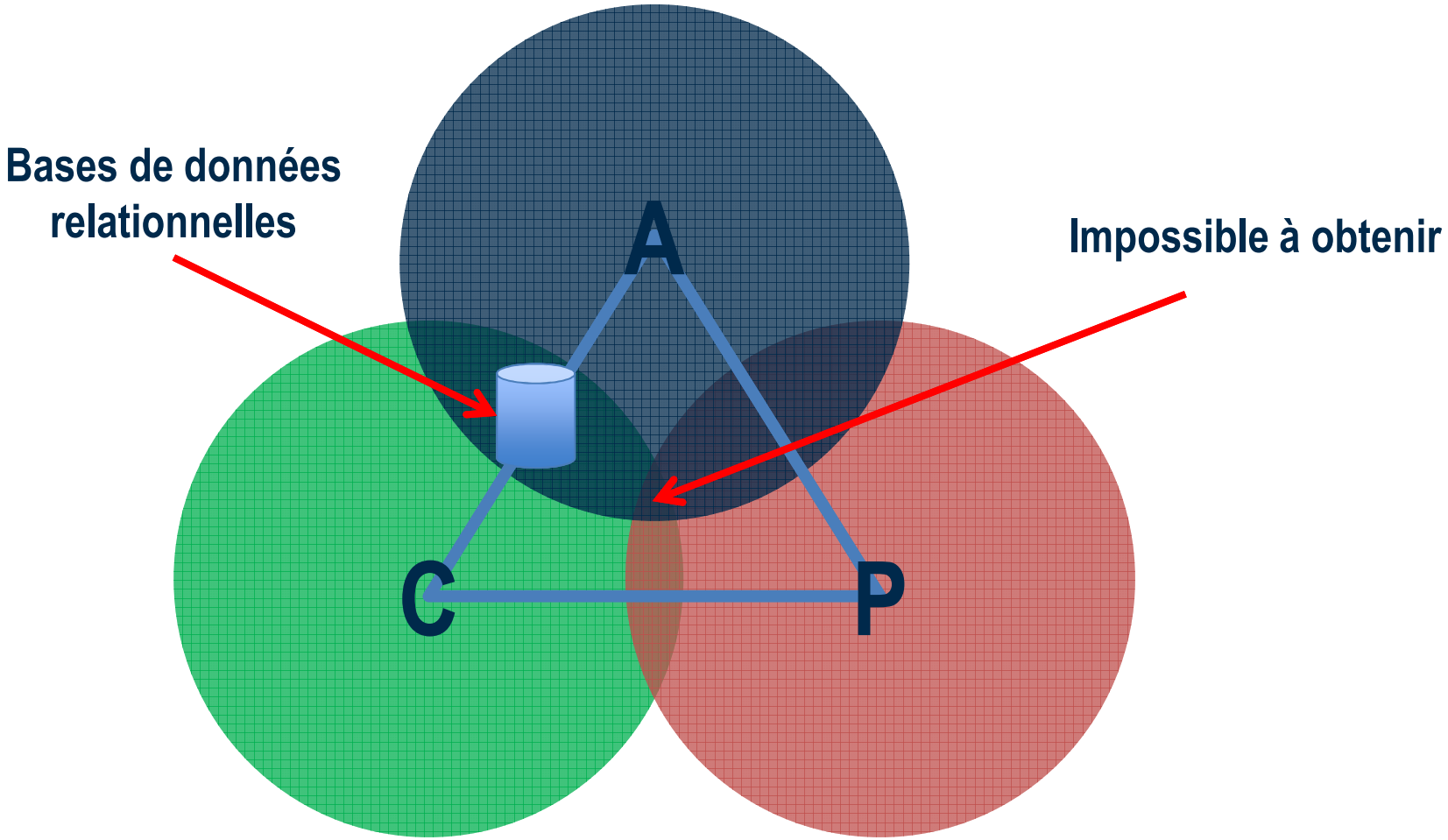
Types de données

- **JSON : JavaScript Object Notation**
  - C'est le format utilisé pour la formalisation des documents ainsi que pour l'échange de documents
    - Dans ce dernier, c'est une variante, BSON (version binaire de JSON), qui est utilisée
    - Cette version est plus compacte et surtout portable entre systèmes
  - Respect de la RFC 4627

Positionnement CAP



Positionnement CAP



Généralités sur MongoDB

Les avantages

- **Permet de travailler sur des données de taille importante dont le schéma de description n'est pas stable**
  - L'ajout d'un nouveau champ est instantané et n'affecte pas l'existant
  - Pas de schéma, donc pas de normalisation et pas de jointures
    - Plus de souplesse
- **Support du format JSON de bout en bout**
  - Un objet peut facilement être sérialisé au format JSON et stocké tel quel
- **Langage de requêtes puissant**
- **Répond aux besoins de performance**
  - Scaling « horizontal » simple et efficace (sharding)
  - Haute disponibilité
    - *Mise en place de "replica sets" simple et rapide*
    - *Récupération des données depuis un "replica" secondaire* sûre et performante
- **MongoDB possède des fonctions d'indexation géospatiale**

Source : http://sharkoliv.blogg.org/mulet-bien-charge-a115760130

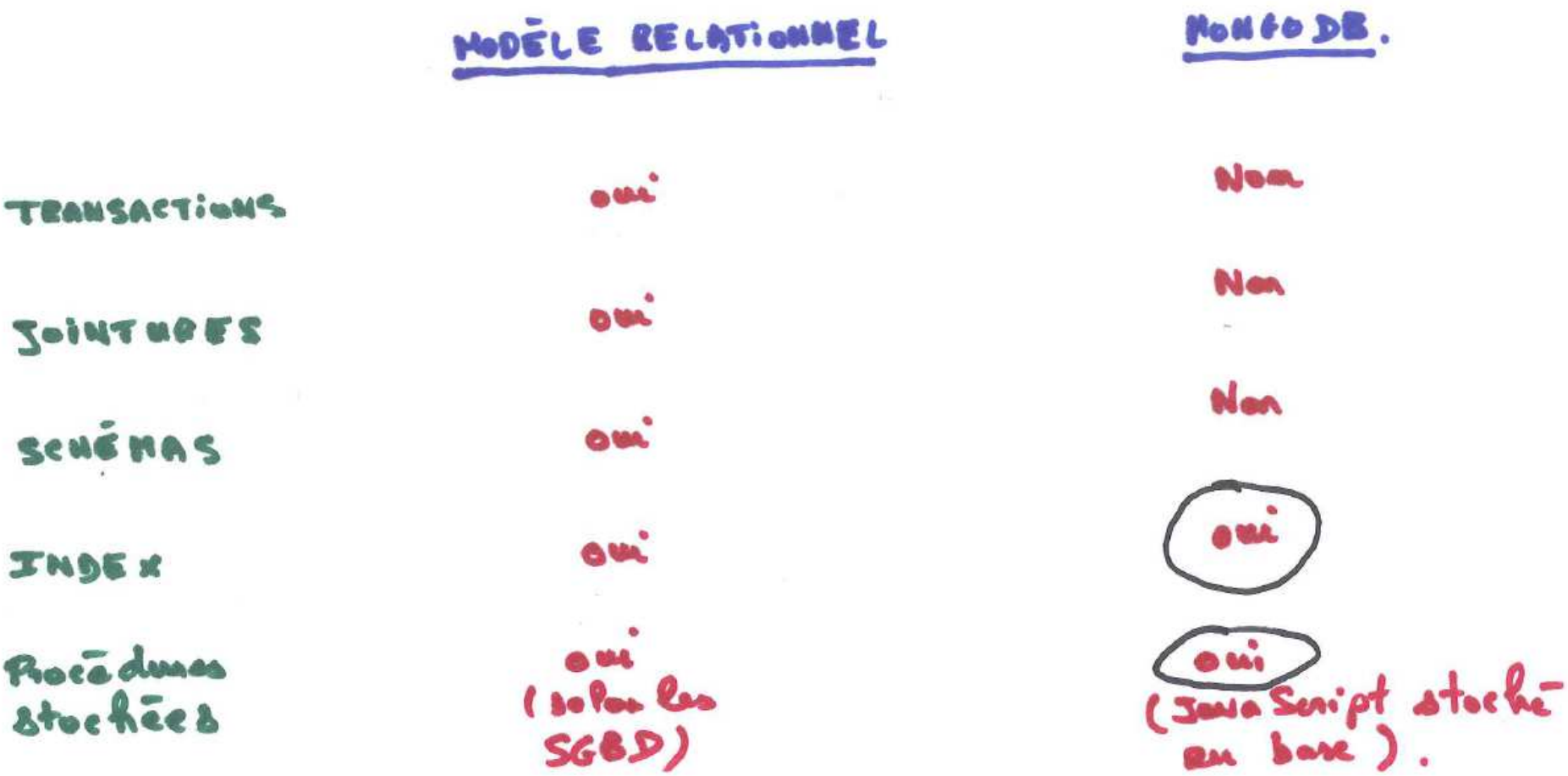
Les inconvénients

- **Pas de gestion du mode transactionnel**
  - Il faut faire appel à un gestionnaire de transactions si nécessaire (Java Transaction API, etc.)
- **Pas du tout de support du langage SQL**
- **Pas de schémas donc peu de vérifications (contraintes d'intégrité)**
- **Les mécanismes d'authentification, d'autorisation et de cryptage ne sont pas activés par défaut**

Source : http://sharkoliv.blogg.org/mulet-bien-charge-a115760130

Relationnel vs MongoDB

Ce que fait (et ne fait pas) MongoDB



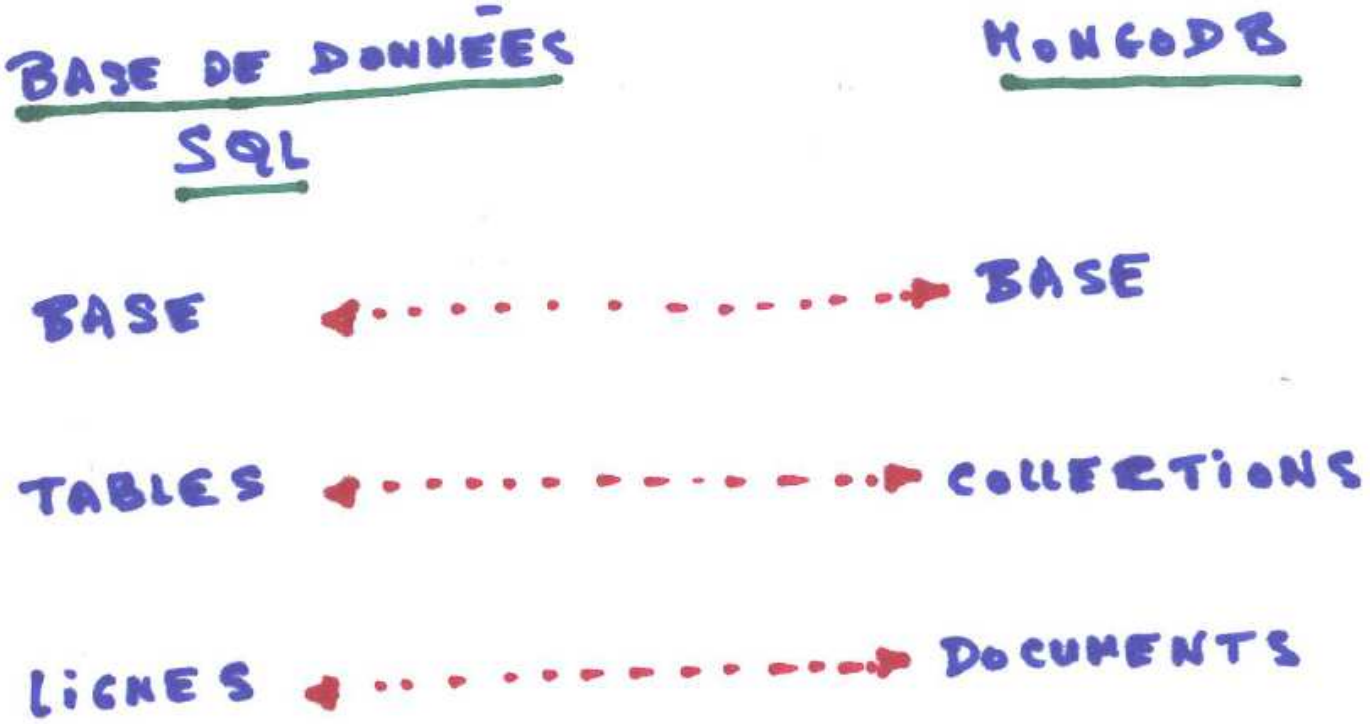
MongoDB est-il ACID ?

- **Propriétés ACID**
  - **Atomicité** : opérations atomiques au niveau du document
  - **Cohérence** : si la lecture se fait sur le maître (par défaut) alors la cohérence est forte sinon elle est faible (mais gain sur le fait de répartir)
  - **Isolation** : non car pas de transaction … mais verrous possibles au niveau des collections voire même des documents avec 3.0 et WiredTiger
  - **Durabilité**: journalisation des données et sauvegarde du journal toutes les 100ms. Mise en place de réplication possible

- **Globalement, MongoDB possède certaines propriétés ACID mais pas toutes**
  - C'est un constat attendu : la performance a un coût
  - Pour le reste, c'est à la discrétion du développeur
  - … Et en fonction du besoin exprimé : on ne peut pas tout avoir

Quelques liens

- Equivalences terminologiques



Types de données

Base de données

- Ensemble de collections
- Une instance MongoDB gère plusieurs bases de données

Collection

- Ensemble de documents
- Equivalent aux tables dans les SGBDR

Document

- Unité de base de stockage
- Equivalent aux lignes dans les SGBDR
- Ensemble de couples clé-valeur au format JSON
- Chaque document possède une attribut unique : `_id` dans une collection

Un point sur l'identifiant

- **« _id » est créé automatiquement par MongoDB**
- **Cet identifiant est, par défaut, construit de la manière suivante :**
  - 12 octets au format BSON
    - 4 octets : nombre de secondes depuis le début de l'ère Unix (01/01/1970)
    - 3 octets : identifiant machine
    - 2 octets : identifiant du processus
    - 3 octets : compteur démarrant à partir d'une valeur aléatoire
- **Cette identifiant peut être modifié mais son unicité doit être garantie !**

Installation et première mise en œuvre

Installation

Ajout du dépôt officiel pour Ubuntu (et Debian 8.x)

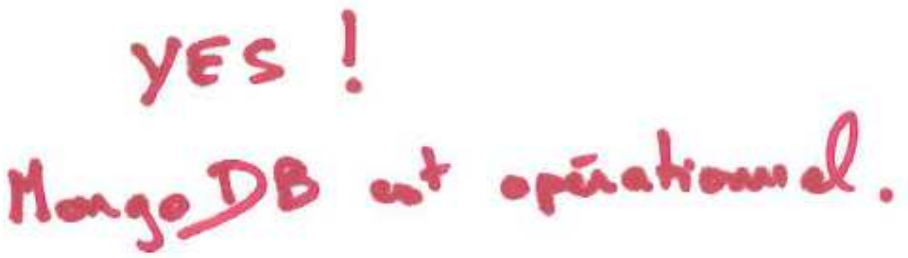
- `sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 7F0CEB10`
- `echo "deb http://repo.mongodb.org/apt/debian wheezy/mongodb-org/3.0 main" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.0.list`
- `sudo apt-get update`

Installation depuis le dépôt

- `sudo apt-get install -y mongodb-org`

Lancement

- `sudo service mongod start`
- `cat /var/log/mongodb/mongod.log`



Connection

- `mongo --shell --host <adresse du host>`

Le moteur de stockage

Point particulier sur le choix du moteur de stockage

- paramètres à positionner au lancement de MongoDB pour :
  - choisir le moteur de stockage,
  - la taille du cache,
  - la compression.

Les paramètres :

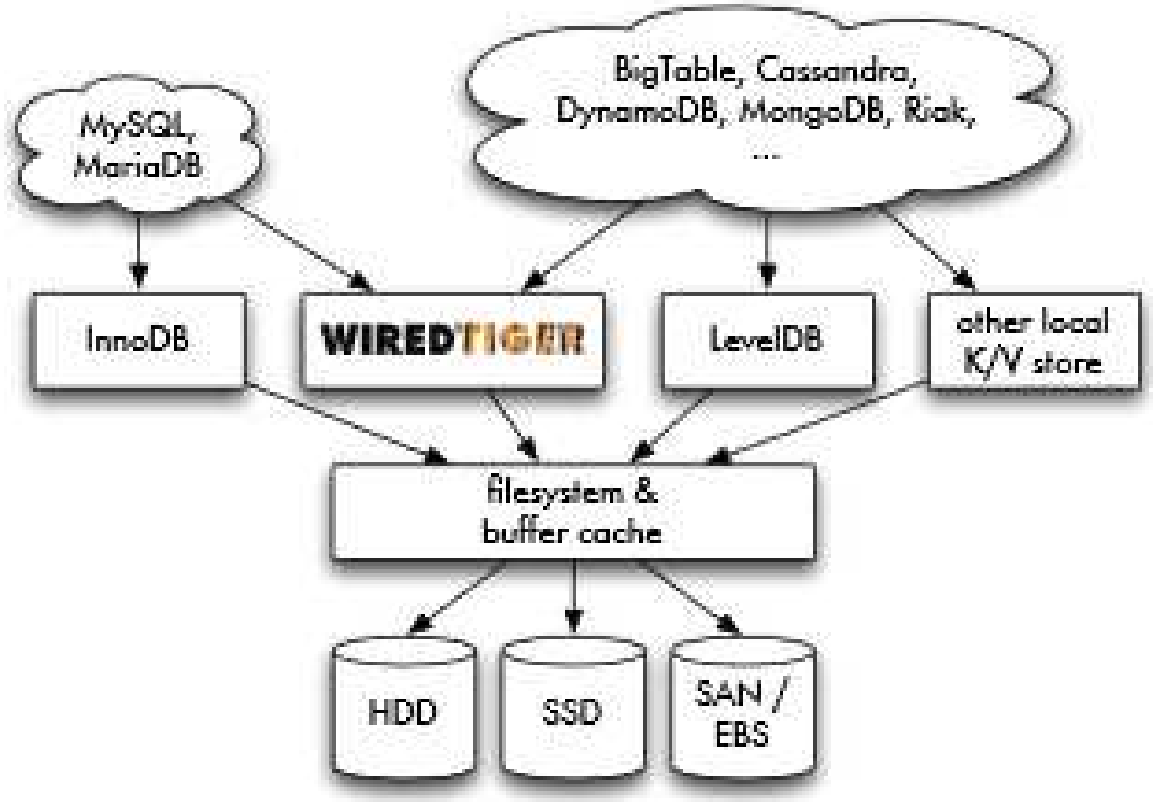
- –storageEngine wiredTiger|MMAPv1
- –wiredTigerCacheSizeGB int
- –wiredTigerCollectionBlockCompressor none|snappy|zlib
- –wiredTigerJournalCompressor none|snappy|zlib
- Pour plus d'information : https://docs.mongodb.org/v3.0/reference/configuration-options/#storage.wiredTiger

Le moteur de stockage

- **Remarques sur les moteurs de stockage :**
  - MMAPv1 est le moteur initial
  - WiredTiger a été racheté en 2014 (https://github.com/wiredtiger/wiredtiger)
    - WiredTiger, développait un moteur de stockage Open Source en mode clé/valeur qui « associe les performances des bases de données in-memory à la capacité illimitée des bases reposant sur du Flash ou des disques durs »
    - Développé par les créateurs de BerkeleydB
    - Adapté aux architectures fortement multi-threadées
    - Utilisation de BTREE et de compression des données sur disque
    - Index compressés en mémoire
    - Adapté aux caches processeurs importants
    - Fragmentation mémoire réduite (par rapport à MMAPv1) et gestion de cet aspect en tâche de fond
    - Fonctionnalités MVCC (Multiversion concurrency control)
      - rapidité et la prévisibilité lors des traitements d'écriture intensives simultanés
    - 7 à 10 fois plus rapide que le moteur d'origine MMAPv1
    - Utilisé aussi dans d'autres configurations, avec OpenLDAP par exemple
  - WiredTiger est le moteur par défaut depuis la version 3.2 de MongoDB

Le moteur de stockage

Remarques sur les moteurs de stockage :



Source : http://source.wiredtiger.com/pdfs/WiredTiger-1page.pdf

Installation

Vérifications et tests de quelques commandes

- La liste des bases existantes :
  - `show dbs`
- Créer une base
  - Pour utiliser ou créer une base ‘test’, il suffit de faire : `use test`
- Connaitre la base active :
  - `db`
- Obtenir la liste des utilisateurs :
  - `show users`
- Les collections :
  - Voir les collections : `show collections`
  - Insérer dans une collection : `db.savants.insert({nom:"Galilee", annee_nais: 1564 })`
  - Vérifier le résultat : `db.savants.find()`
- Détruire une collection dans une base (après le ‘use <db>’ adéquat !!) :
  - `db.savants.remove({})`
- Simplement sortir :
  - `exit`

Premiers tests

Sur un exemple concret

```
user1@nosql:~$ mongo
MongoDB shell version: 3.0.7
connecting to: test
> show dbs
admin         0.078GB
experiments   0.078GB
local         0.078GB
test          0.078GB
> db
test
> show collections
system.indexes
> db.savants.insert({nom:"Galilée",annee_naiss:1564})
WriteResult({ "nInserted" : 1 })
> show collections
savants
system.indexes
> db.savants.find()
{ "_id" : ObjectId("56663f2658aa5abaa4f3555e"), "nom" : "Galilée", "annee_naiss
> db.savants.remove({})
WriteResult({ "nRemoved" : 1 })
> db.savants.find()
> show collections
savants
system.indexes
> db.savants.drop()
true
> show collections
system.indexes
> exit
bye
user1@nosql:~$
```

connexion en mode client

Bases existantes

création collection et document

Destruction document et collection.

Cool ....

Administration et développements

Administration

- L'administration s'effectue en 'standard' via un shell écrit en Javascript
- Des outils graphiques sont disponibles
  - Robomongo (www.robomongo.org) est un outil très utile au quotidien
  - Mais il en existe d'autres, gratuits ou non (MongoChef par exemple)

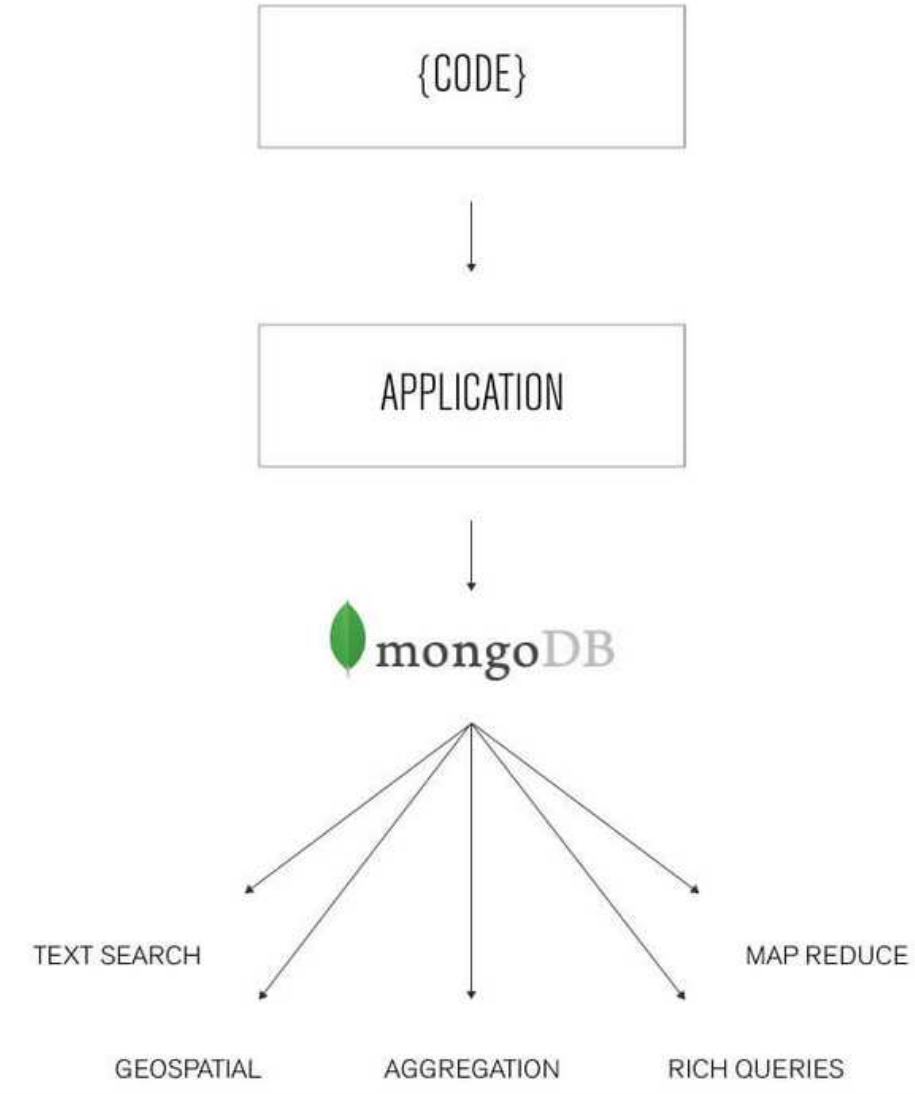
Installation de la librairie PyMongo pour le développement

- Permet l'utilisation de Python
- Commande : `pip install pymongo`

Rappel des principes

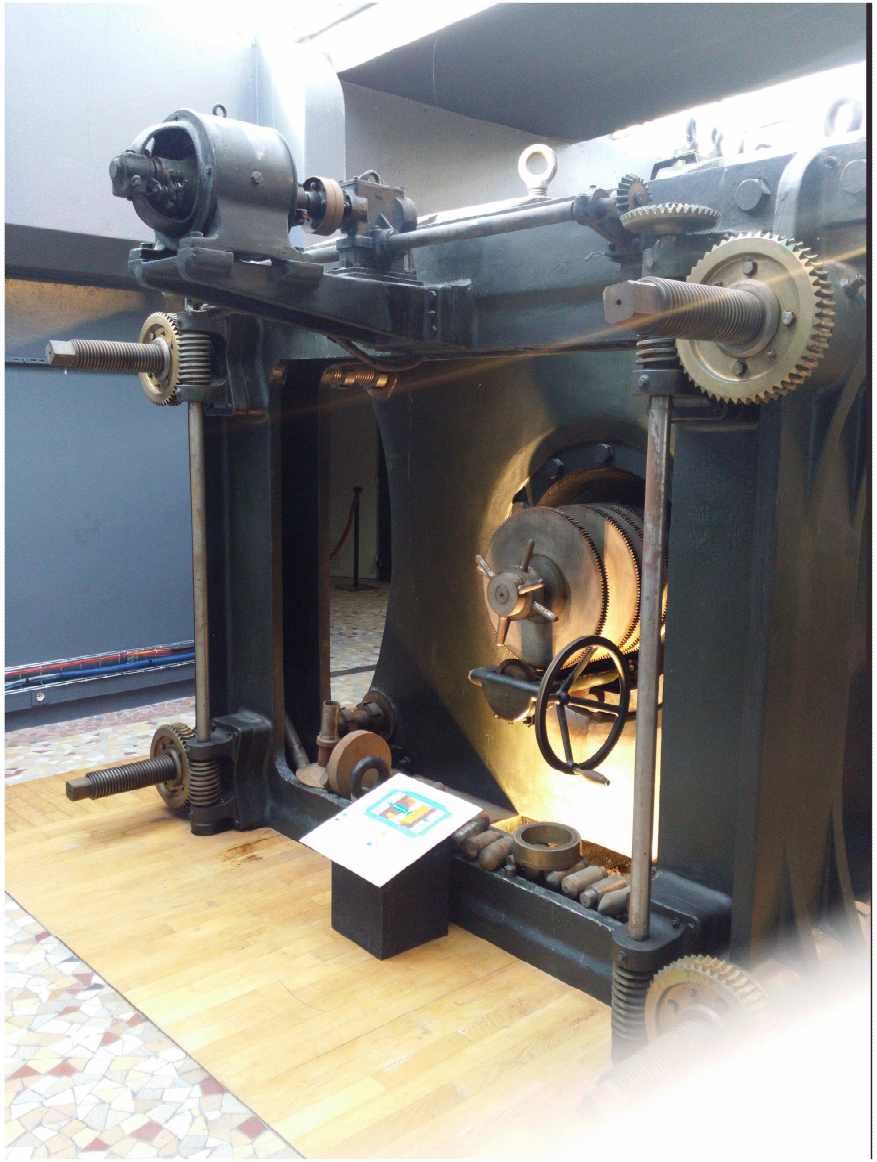
- Dans toute la suite, le travail va s'effectuer sur des documents et des collections de documents
- Le contenu d'un document, l'existence d'une ou plusieurs collections seront dépendants du contexte
- Des tests de performances devront, à chaque fois, guider le choix du modèle adéquat
- Le plus compliqué pour un administrateur de bases de données orienté « relationnel » est d'accepter de perdre certains réflexes de « bon sens » dans un modèle classique

Rappel des principes



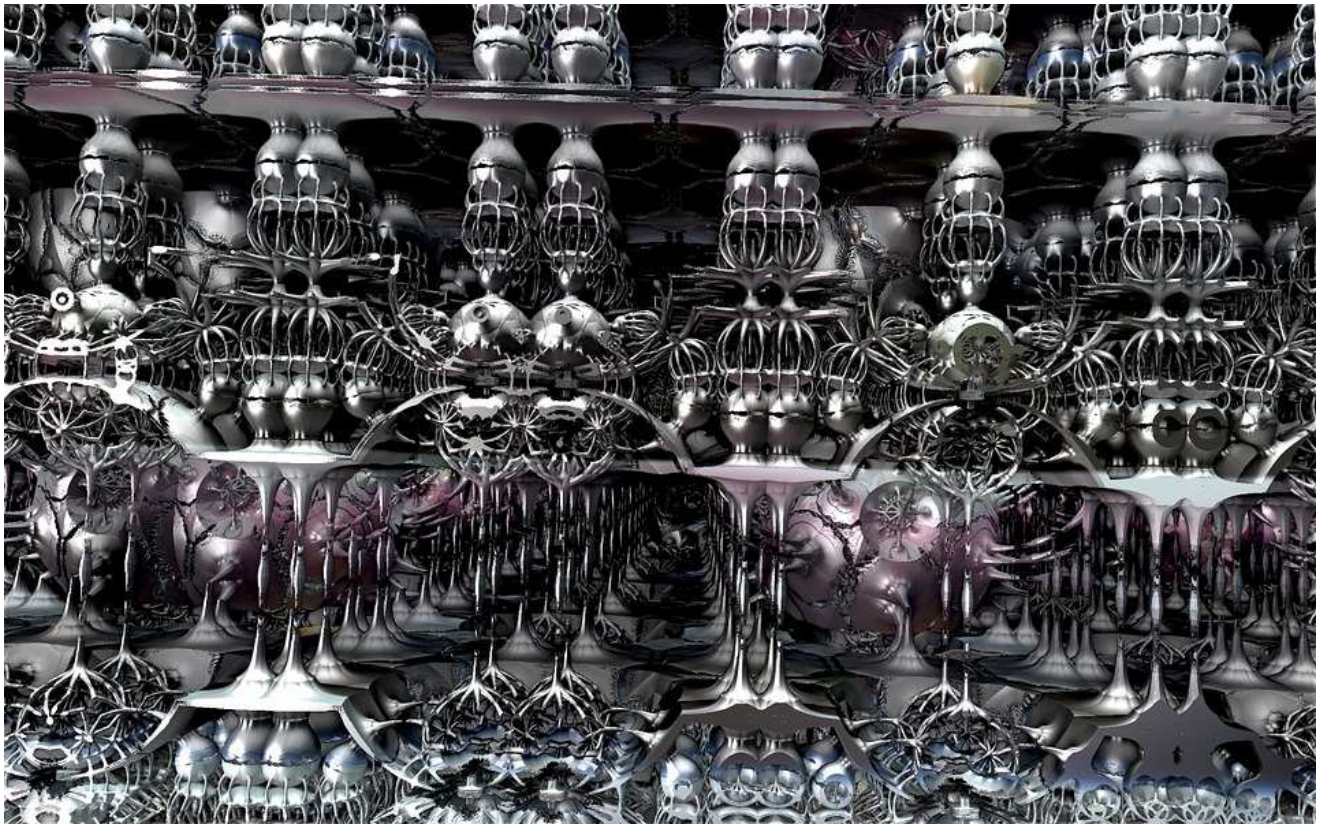
Source : https://www.mongodb.com/mongodb-architecture

Manips en cours …



Electro-aimant Meudon – Nov. 2015

Replicas et sharding



Replicas

- **Les objectifs de la mise en œuvre des réplicas et du sharding est de permettre d'augmenter la disponibilité et de favoriser le « passage à l'échelle » de manière horizontale**
  - Ajouts d'équipements et non augmentation de la puissance d'une machine
- **« Replica Set » et « sharding » sont liés**
- **Cela consiste à faire du partitionnement**
  - Une base de données devient une entité logique répartie sur n serveurs (extensibilité « horizontale »)
  - La répartition est transparente pour les clients via l'usage de routeurs de requêtes ('query routers')
  - La répartition se fait via une « shardkey » et un « shard » est soit un serveur unique soit un 'replica set'

Replicas

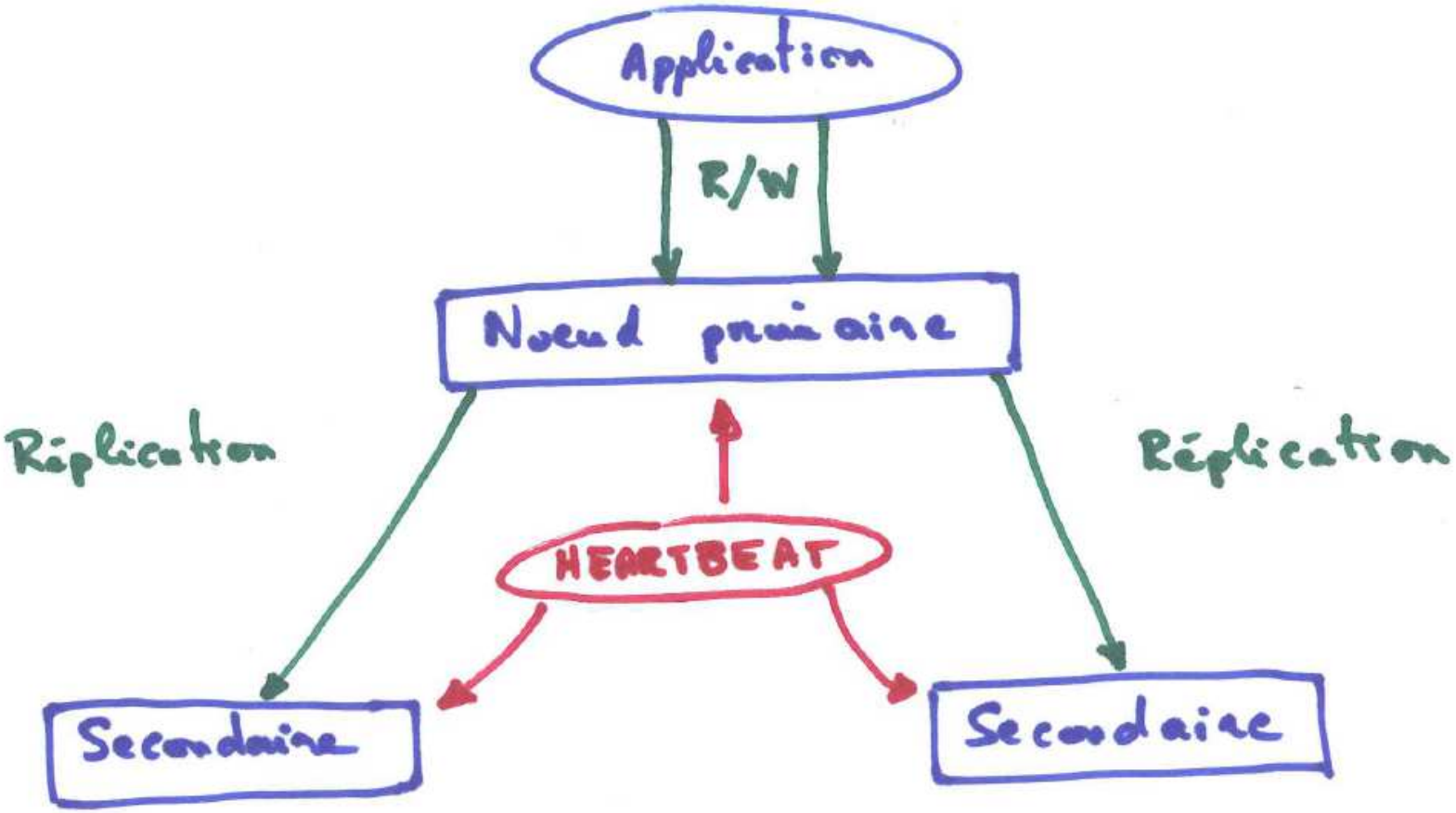
- **Un replica set contient :**
  - 1 primaire
  - Et jusqu'à 12 nœuds
- **Les replicas sont utiles pour :**
  - Augmenter la disponibilité des données
  - Augmenter le niveau de parallélisme des données
- **Cela nécessite de pouvoir écrire les données sur plusieurs serveurs**
  - Ces serveurs deviennent un groupe d'instances qui contiennent les mêmes données (« replica set »)
- **Mise en place d'un « heartbeat » toutes les 2 secondes et élection d'un nouveau primaire si défaillance**
- **La réplication entre primaire et secondaire(s) est transparente**
- **Le primaire reçoit toutes les requêtes (R ou W)**

Replicas

- **Les secondaires recopient les données en asynchrone à partir d'un fichier de log**
  - Cela leur permet aussi de se resynchroniser en cas de rupture de connexion
- **L'élection se fait avec un mécanisme de vote à la majorité**
  - La priorité est positionnée à la configuration (0 : non prioritaire, ensuite 1, 2, etc.) ou donnée au nœud qui possède les dernières mises à jour. Si problème (parité, trop de nœuds perdus), possibilité de créer des 'arbiter' qui ne font que participer au vote et ne possèdent pas de données
- **Il existe plusieurs type de réplicas**
  - 'Primary': détenteur des données. C'est là que les données sont écrites
  - 'Secondary': réplication du 'Primary'. Lecture des données uniquement. Peut devenir 'Primary' si besoin.
  - 'Priority 0': idem 'Secondary'. Peut devenir 'Primary'.
  - 'Hidden': idem 'Secondary' mais non visible. Sert de backup.
  - 'Arbiter': utilisé lors du mécanisme d'élection du primaire.

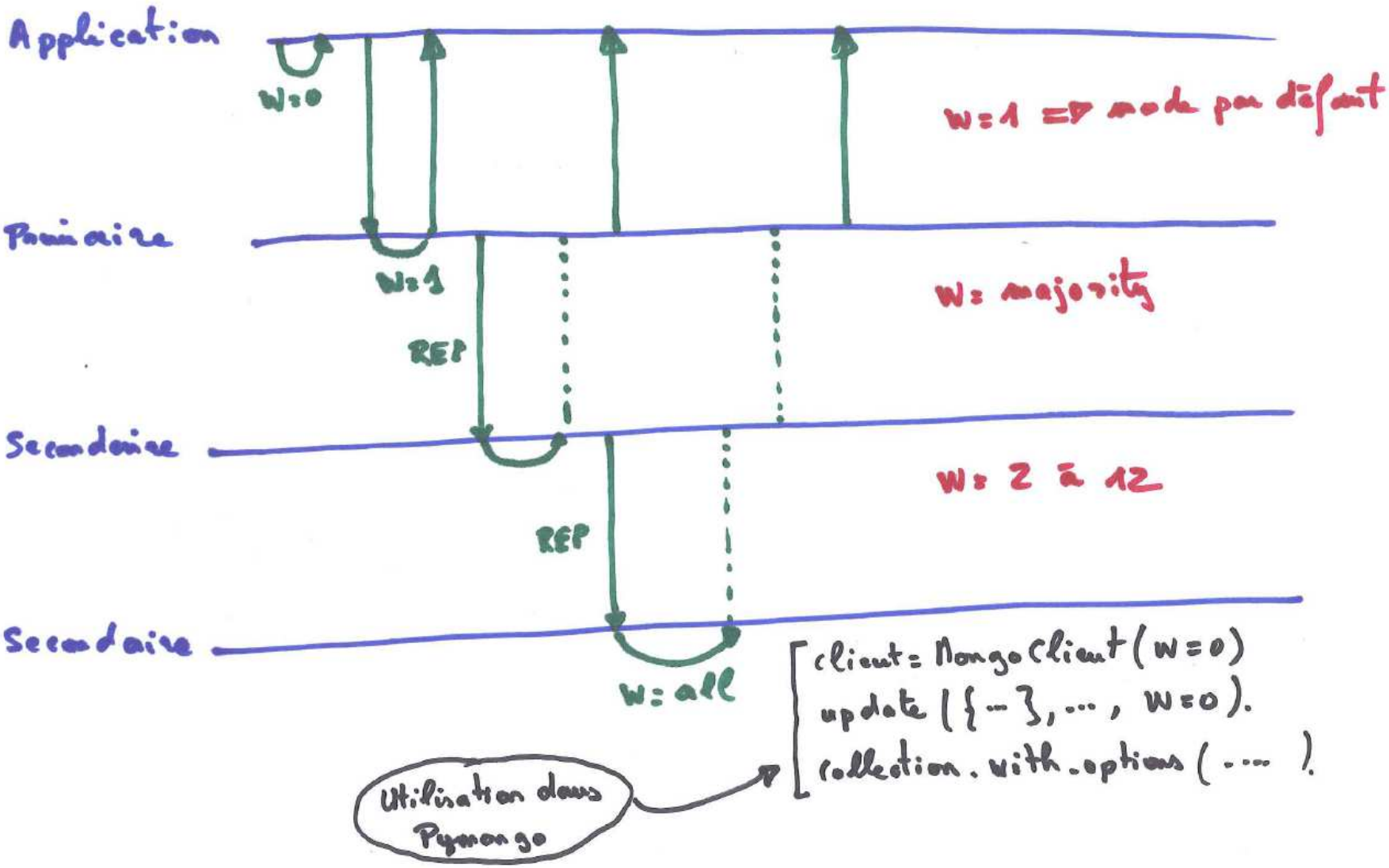
Replicas

- **Schéma récapitulatif**



Replicas

- Comment se passe une écriture dans ce cas (« write concern »):

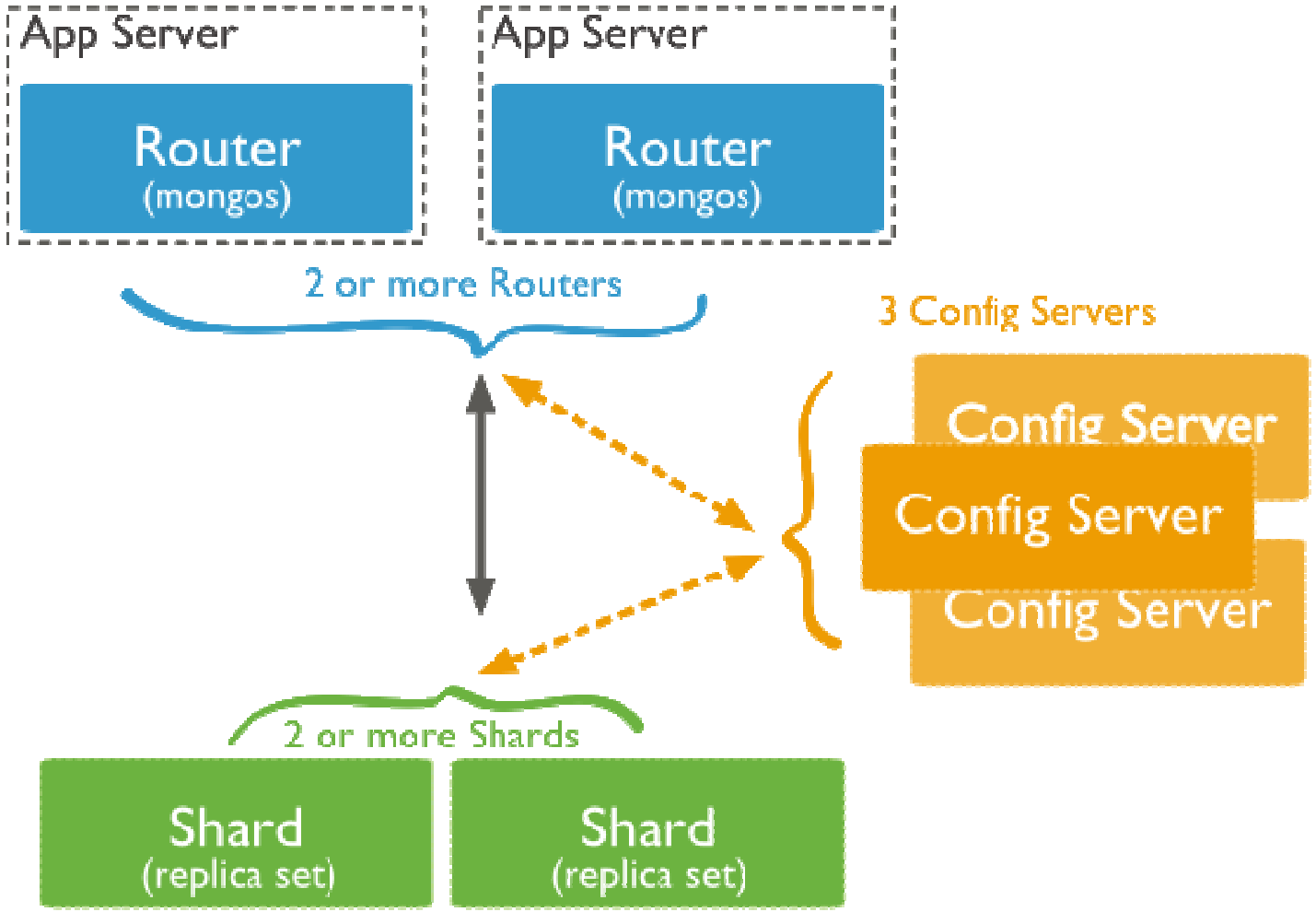


Sharding

- **Le sharding permet l'ajout de serveurs**
- **Le sharding n'a d'intérêt que pour un volume de données au moins supérieur à 300 Go**
- **Les collections sont distribuées selon une « shardkey »**
- **La « shardkey » est elle-même découpée en morceaux (« chunks ») et répartie sur les shards**
- **3 modes de répartition sont envisageables :**
  - « range based » : découpage de l'index en autant de morceaux qu'il y a de shards
  - « hash based » : utilisation d'une fonction de hachage
    - Distribution non uniforme
  - « tag aware » : via des tags fixés par l'administrateur
- **La répartition des chunks entre shards est automatique**
- **Un chunk ne peut faire que 64 Mo au maximum**
  - il est ensuite scindé en deux

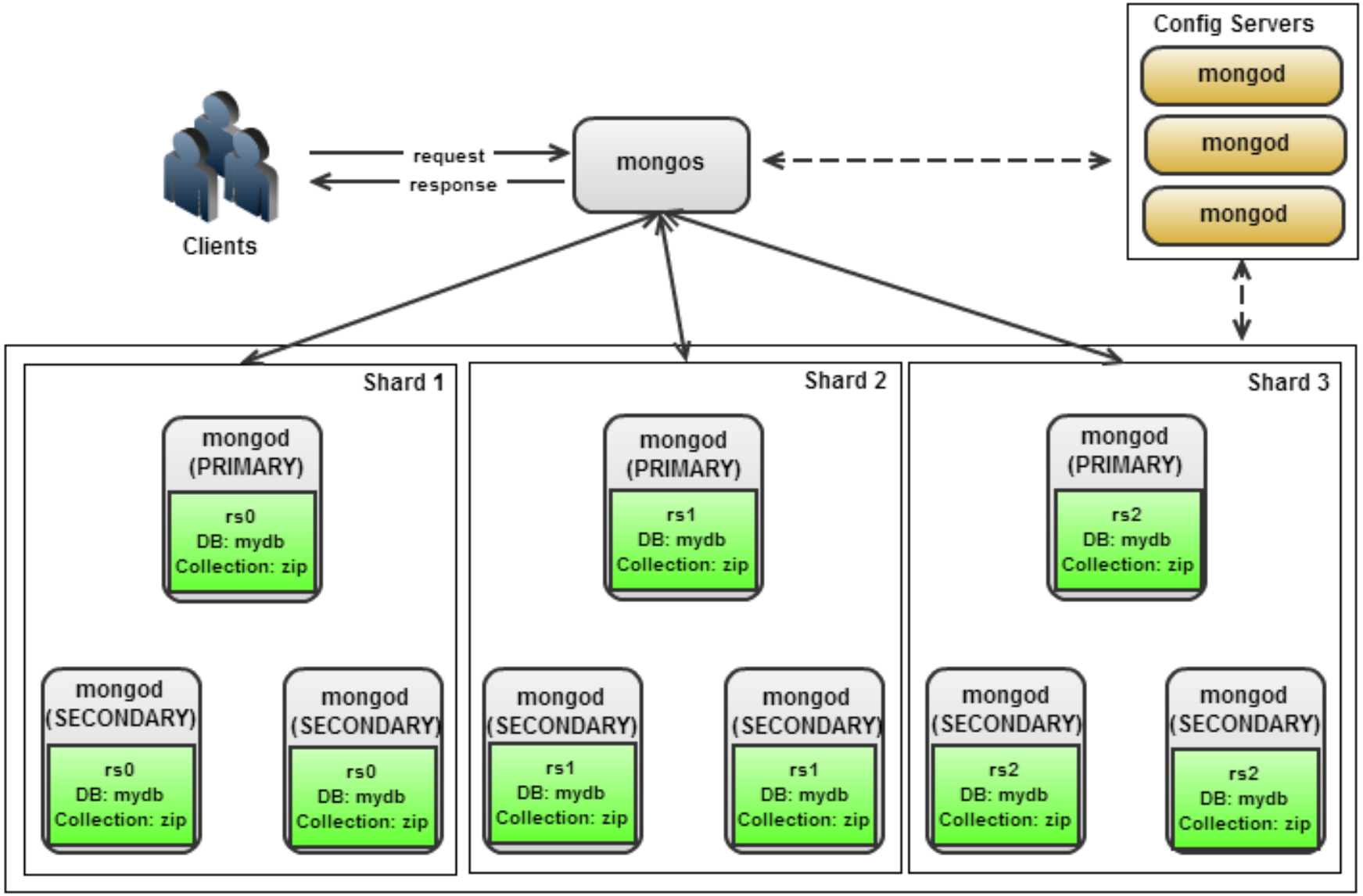
Replicas et sharding

- **Comment se passe une écriture dans ce cas :**



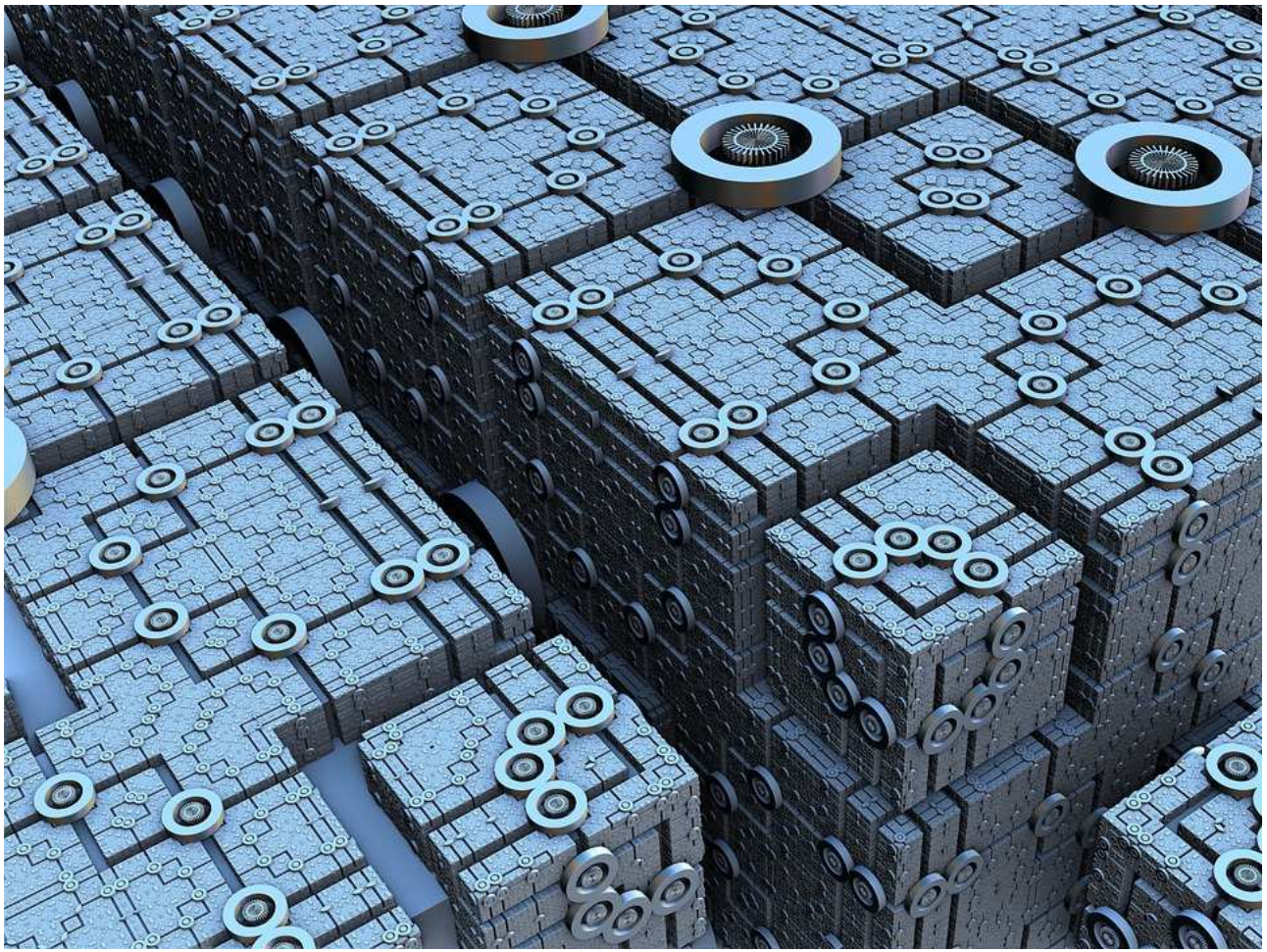
Source : http://blog.ippon.fr/wp-content/uploads/2014/10/sharded-cluster-production-architecture.png

Replicas et sharding



Source : http://dba.stackexchange.com/questions/82551/data-distribution-in-mongos-with-shards-or-replica-sets

GridFS



GridFS

- **GridFS est un “filesystem” conçu au-dessus de MongoDB**
- **GridFS permet de répondre à des problématiques telles que :**
  - Le stockage de fichiers en lien avec une BD : souvent, ces fichiers sont stockés dans un « filesystem à côté »
  - Cela permet de gérer les aspects CRUD de ces fichiers liés à une BD
  - Cela permet aussi de mettre en place des sauvegardes “homogènes”
- **GridFS résout ces questions en intégrant ces fichiers dans les BD**
  - Ils sont alors gérés comme une BD Mongo classique
- **GridFS découpe les fichiers en “chunks” de 256 Ko, “chunks” stockés selon la configuration de la base (sharding …)**
- **Par défaut, la taille maximale des documents MongoDB est de 16 Mo**
  - Pour des documents de taille supérieure, l’emploi de GridFS est donc recommandé

GridFS

- **Exemple d'utilisation :**
  - Chargement
    - Vérification de l'état en cours : mongofiles –d bigfiles list
    - Ajout du fichier : mongofiles –d bigfiles put datarandom.json
    - Vérification de l'état après l'ajout : mongofiles –d bigfiles list
  - Lecture
    - mongofiles -d bigfiles get datarandom.json
  - Lister
    - mongofiles –d bigfiles list
  - Détruire
    - mongofiles -d bigfiles delete datarandom.json
  - GridFS utilise 2 collections pour stocker les données ('show collections' après 'use bigfiles') :
    - fs.chunks
    - fs.files

GridFS

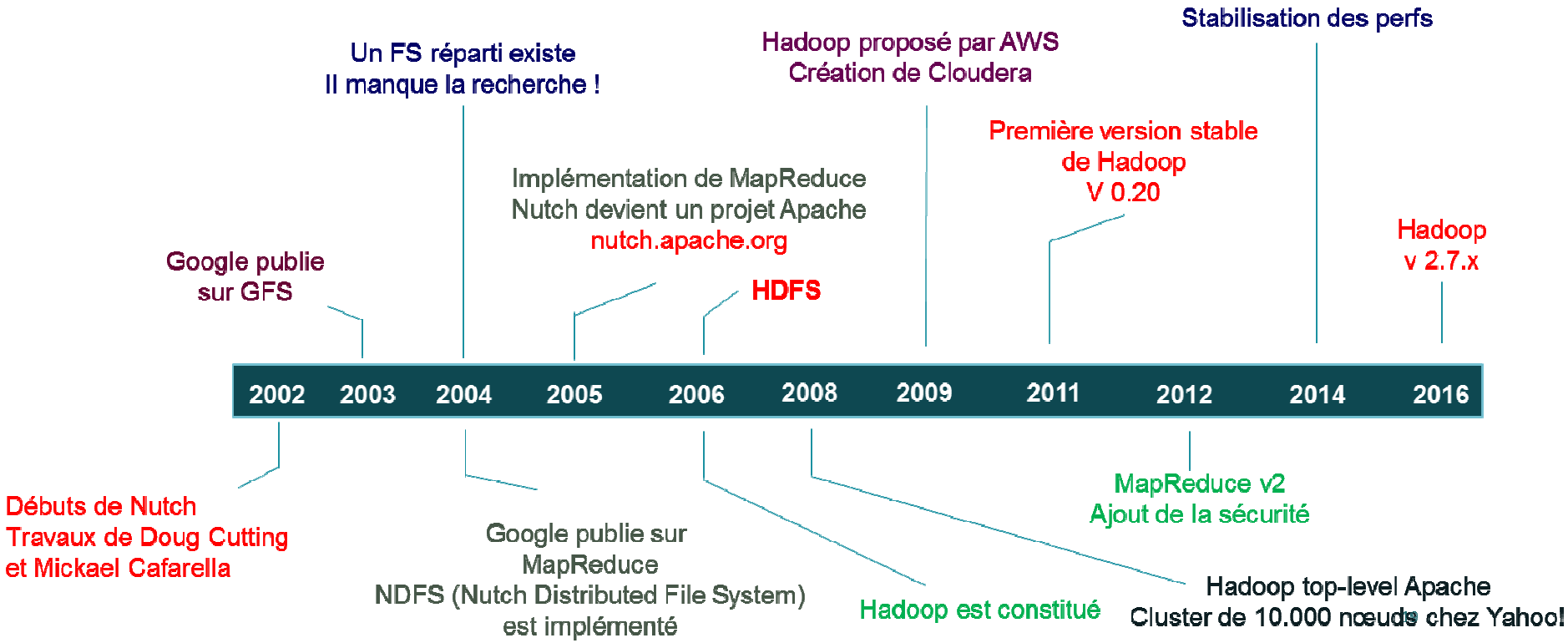
- **Il est alors possible d'exécuter les commandes classiques pour inspecter les modalités de stockage :**
  - db.fs.files.findOne()
  - db.fs.chunks.getIndexes()

Map/Reduce

Map/Reduce ?

Hadoop : les origines

Hadoop : petit nom d'une peluche en forme d'éléphant du fils de Doug Cutting !



Quel est le problème ?

- **Fonctionnement d'un code logiciel classique :**
  - Le code s'exécute sur un processeur et traite des données en entrée et fournit des données en sortie
  - Les données sont lues sur le support physique :
    - Une commande de lecture est envoyée au système
    - Cette commande de lecture est gérée par des interruptions pour assurer l'optimisation du partage du processeur
    - Le processeur n'attends pas la fin de la lecture et a la capacité d'exécuter un autre processus !
    - Le retour au code initial se fait sur interruption, lorsque les données sont disponibles
  - Mais cela implique qu'il y a respect du principe de localité des flux d'E/S
    - Les systèmes d'exploitation ne prennent en général pas en compte nativement le fait que les données puissent être réparties
  - Comment alors combiner la masse des données, la nécessité de réaliser des accès parallèles pour des questions de performance et l'augmentation du nombre de processeurs ?

Quel est le problème ?

- **Quelques conséquences :**
  - Il va falloir utiliser dans certains cas des algorithmes d'allocation de ressources parallèles adaptés
  - Un des points les plus problématiques concerne la gestion des verrous :
    - l'introduction du parallélisme nécessite la mise en place de verrous de plus en plus nombreux pour gérer les accès en parallèle
    - Un verrou de niveau fichier n'est pas envisageable dans un contexte d'un grand nombre de processus en parallèle : cela reviendrait à un traitement en série sur une ressource unique
    - Mais il y a des contraintes :
      - Peu de verrous : diminution de la capacité de parallélisme
      - Trop de verrous : fragmentation des tâches et donc du rendement des processeurs qui passent leur temps en commutation de contexte
  - La question de la granularité des verrous devient essentielle

Quel est le problème ?

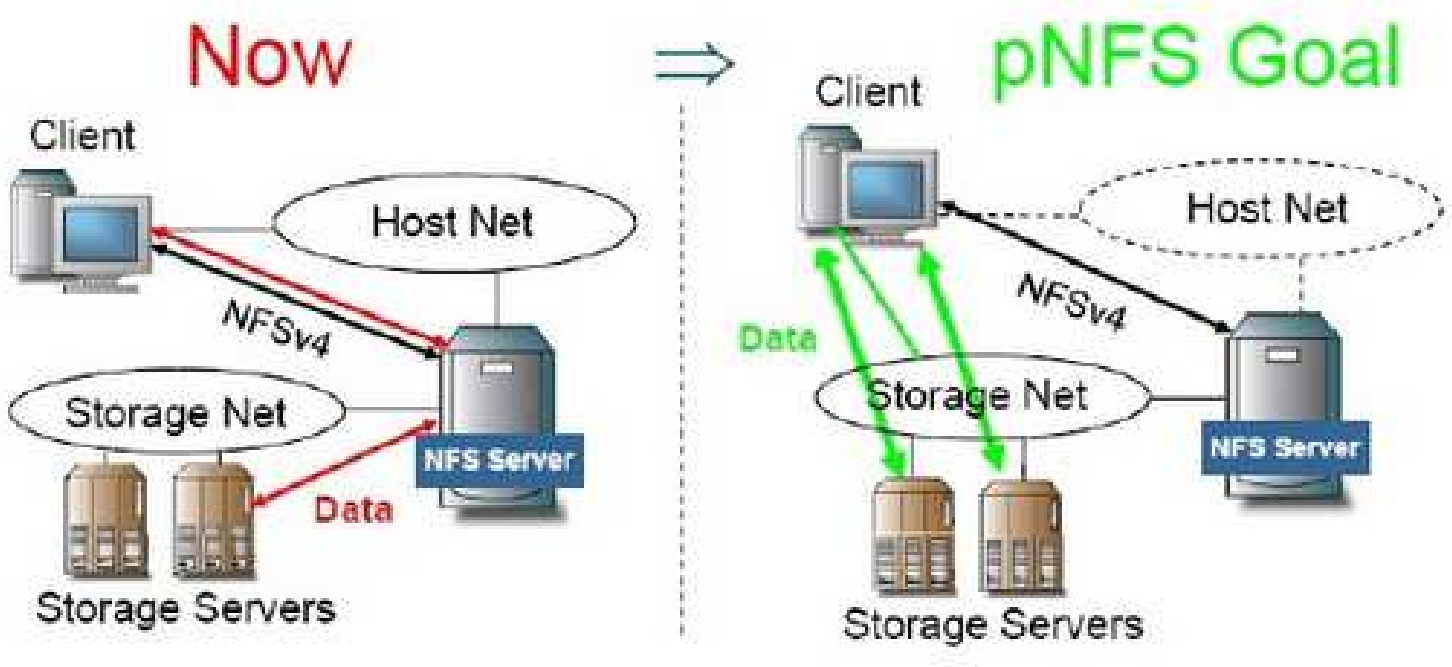
- **Et pourtant, encore une fois, le parallélisme pour l'accès aux données est un besoin critique :**
  - La capacité des disques augmentent mais le taux de transfert ne suit pas (environ 30% en 10 ans)
  - Il n'y a pas d'autre solution que de paralléliser les accès
- **Cela implique que, sur des volumétries importantes, il faut penser à déployer un système de fichiers adapté**
  - Emergence des PFS (Parallel File System) pour paralléliser le transport de la donnée
  - Nécessité de dissocier, dans certains cas, le stockage de la donnée de son indexation
  - Gérer la répartition/duplication des données et garantir la cohérence globale

Quel est le problème ?

- **Il existe plusieurs types de systèmes de fichiers distribués avec des caractéristiques différentes :**
  - Client/serveur (RPC) : HDFS, etc.
    - Héritage de NFS
  - Peer-to-peer : Cassandra, etc.
  - Stockage objet ou bloc
  - Mode de hachage (répartition) des données
  - Etc.
- **Beaucoup d'implémentations dans chaque catégorie**
  - Beaucoup de recherches et de concurrence industrielle
  - Difficile d'avoir des repères car le domaine évolue rapidement

Quel est le problème ?

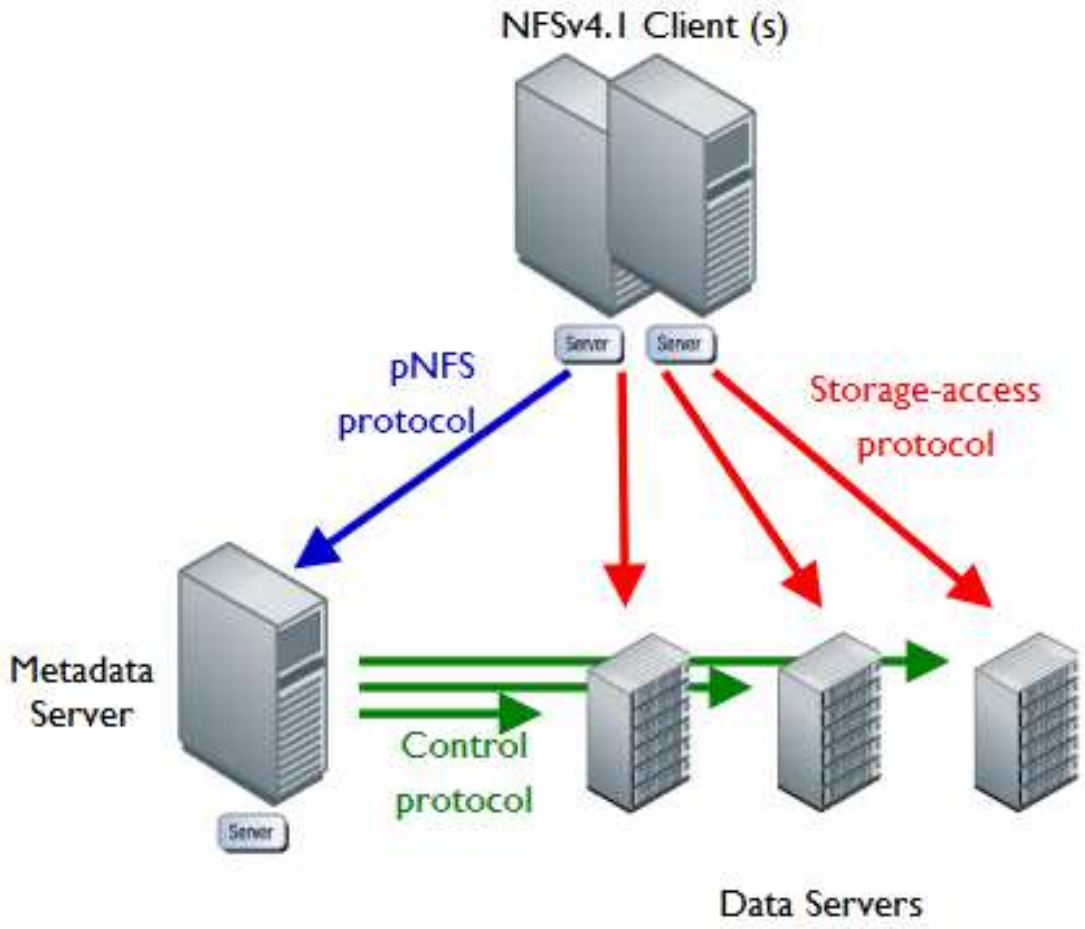
- pNFS (parallel NFS) / NFS v4.1 & v4.2



http://www.slideshare.net/schubertzhang/pnfs-introduction

Quel est le problème ?

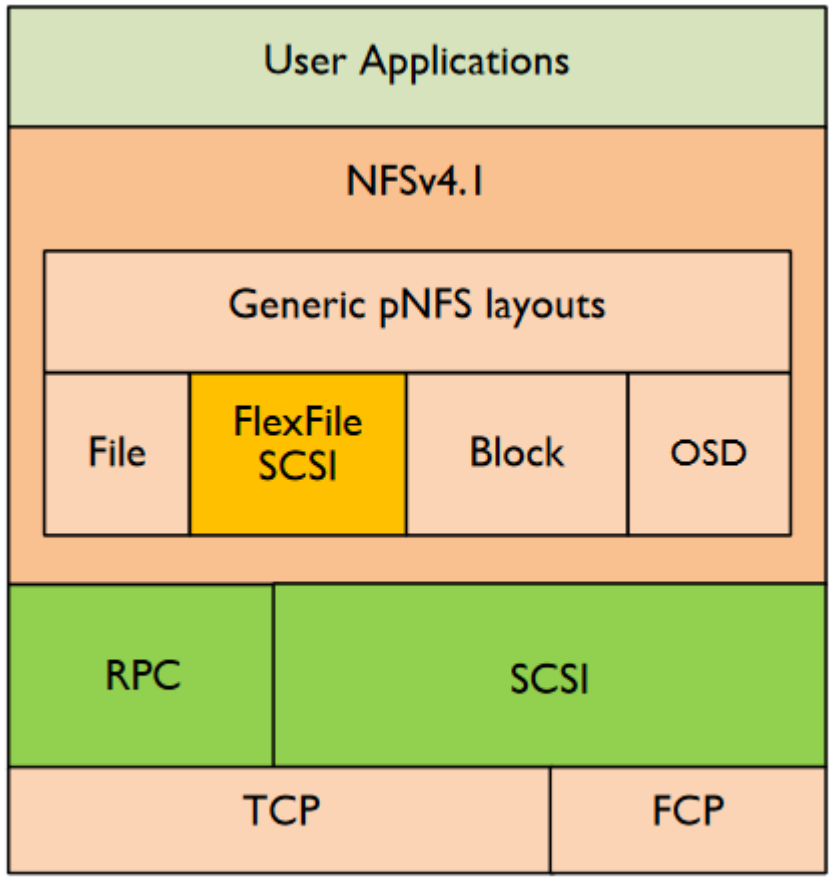
- pNFS (parallel NFS) / NFS v4.1 & v4.2



http://www.snia.org/sites/default/files/ESF/FINAL_SNIA_An_Overview_of_NFSv4-4_20Oct2015.pdf

Quel est le problème ?

- pNFS (parallel NFS) / NFS v4.1 & v4.2



OSD: Object based Storage Device

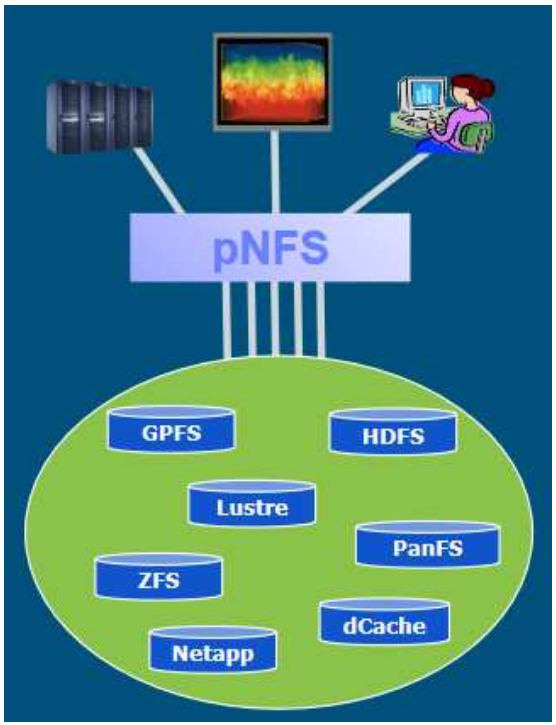
http://www.snia.org/sites/default/files/ESF/FINAL_SNIA_An_Overview_of_NFSv4-4_20Oct2015.pdf

Quel est le problème ?

- **pNFS (parallel NFS) / NFS v4.1 & v4.2**
  - RFC 5661 - describes NFS version 4 minor version 1, including features retained from the base protocol and protocol extensions made subsequently
  - RFC 5662 - contains the machine readable XDR definitions for the protocol.
  - RFC 5663 - provides a specification of a block based layout type definition to be used with the NFSv4.1 protocol. As such, this is a companion specification to NFS version 4 Minor Version 1
  - RFC 5664- provides a specification of an object based layout type definition to be used with the NFSv4.1 protocol. As such, this is a companion specification to NFS version 4 Minor Version 1
  - Etc.
- **Voir http://datatracker.ietf.org/wg/nfsv4 pour suivre les évolutions**

Quel est le problème ?

- **pNFS n'est qu'un exemple de système de fichiers**
  - Il en existe beaucoup d'autres se positionnant à des niveaux divers
  - CEPH, Lustre, etc.
- **Mais pNFS permet de fournir une couche d'abstraction supplémentaire et est intégré dans un processus de normalisation**



http://researcher.watson.ibm.com/researcher/files/us-dhildeb/hildebrand-fast16-tutorial.pdf

Quel est le problème ?

- **Chaque type de système de fichiers a donc ses avantages et ses inconvénients**
- **Les performances actuelles semblent être de l'ordre de la centaine voire de plusieurs centaines de Go/s**
  - Mais il est difficile d'avoir des chiffres
  - Tout dépend aussi des tests réalisés et des conditions de ces tests
    - Petits ou gros fichiers
    - Architecture matérielle retenue pour réaliser les tests
    - Nombre de clients simultanés
    - Etc.

Quel est le problème ?

- **Tout cela implique que, selon le domaine d'application, il faudra faire des POC et des tests associés**
- **Mais, même avec simplement une centaine de Go/s pour les accès disques, cela induit un principe fort :**
  - Il est plus facile d'obtenir des performances en matière d'accès aux données que pour ce qui concerne leur transfert dans un réseau, même performant
  - Il faut donc éviter au maximum les transferts de données via le réseau et privilégier les traitements en local
    - Exception faite des mécanismes de réplication par exemple mais qui sont en général faits au « fil de l'eau »
- **A retenir : la tendance est de plus en plus de rapprocher les traitements des données**
  - C'est la base en Big Data (cf Hadoop par exemple)

Quel est le problème ?

- **Les mécanismes physiques et logiciels permettent de plus en plus d'utiliser des matériels à bas coûts et d'assurer la sécurisation par la duplication**
- **Mais cela reste complexe**
  - Synchronisation des données, maj réparties (maintien de la cohérence des données), verrous, etc.
  - Il existe de multiples solutions concurrentes et/ou complémentaires pour les systèmes de fichiers (et l'utilisation d'un service Cloud n'a pas été évoqué !)
- **Les traitements doivent être rapprochés des données**
- **Les données doivent être stockées de manière « élastique »**
  - Quant à leurs localisations
  - Quant à leurs formats (structurés, semi-structurés, etc.)
  - Quant à leur universalité d'accès (initiative NFS v4.1 & v4.2)
- **Le NoSQL est d'un grand apport dans le domaine**

Map/Reduce ?

- **Développement Open Source en Java**
- **Représente 10% des développements de la fondation Apache**
- **Des contributeurs importants : Yahoo!, Facebook, eBay, Microsoft, Twitter, Cloudera, Hortonworks, etc.**
- **Des solutions concurrentes et/ou complémentaires, plus traditionnelles :**
  - SAN et autres armoires de stockage
  - Des appliances dédiées : TeraData, Scality, etc.
- **Mais ces solutions plus « traditionnelles » (et adaptées à leurs contextes d'utilisation) posent la question du passage à l'échelle à moindre coût**
  - Comment augmenter la capacité de stockage de manière linéaire et pour un coût maitrisé ?
- **C'est là que Hadoop propose une solution**

Map/Reduce ?

Hadoop = HDFS + MapReduce

Hadoop : la fiabilité

- **Prenons un exemple avec 200 machines (ou nœuds) :**
  - Chaque nœud à une fiabilité de 99%
  - A un instant donné, la probabilité de survenue d'une panne est de :
    - 1 – 0.99200 soit environ 86,6 % !!!!

- **Hadoop va donc devoir adresser cette dimension**
  - On parle de « design for failure »

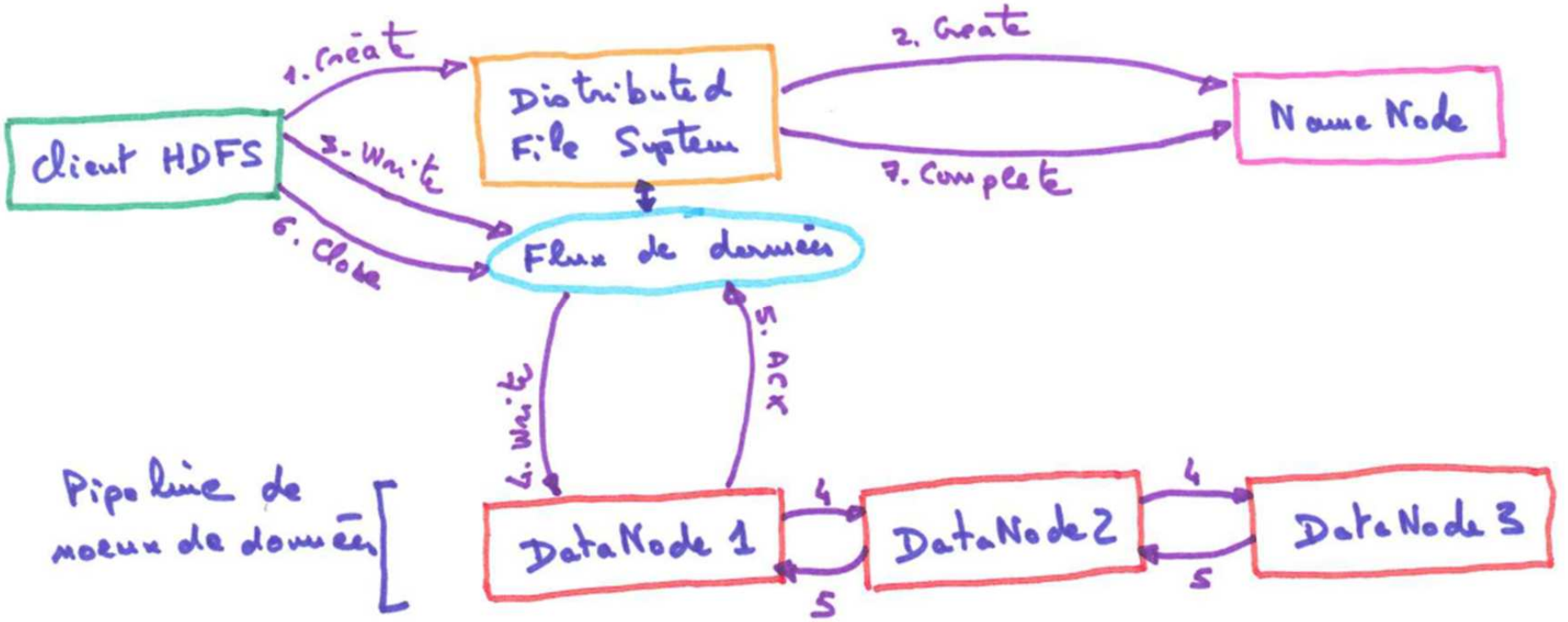
Hadoop : la fiabilité

- **Le problème de la fiabilité (« design for failure »)**
  - Géré par Hadoop via la réplication de portions de données
  - Il existe des données primaires dupliquées sur des secondaires
  - La réplication se fait si possible sur des nœuds éloignés

- **Données dupliquées ?**
  - Cela implique que la volumétrie de stockage disponible doit intégrer le facteur de réplication
  - Un facteur de réplica de 3 implique de tripler la capacité de stockage globale

Hadoop : la fiabilité

Le système de fichiers réparti HDFS



HDFS

- **On retrouve ici :**
  - Les datanodes : les nœuds de stockage physiques
    - Gèrent les blocs de données transmis
  - Le namenode : gère la liste des blocs pour un fichier donné
    - Permet de reconstituer un fichier dans un contexte de blocs répartis sur un primaire et des secondaires
    - Le namenode est un composant critique
      - S'il disparait, il est impossible de retrouver les données
  - Le client, au choix :
    - Un code logiciel effectuant une opération d'écriture
    - Des commandes systèmes, à l'image des commandes Unix

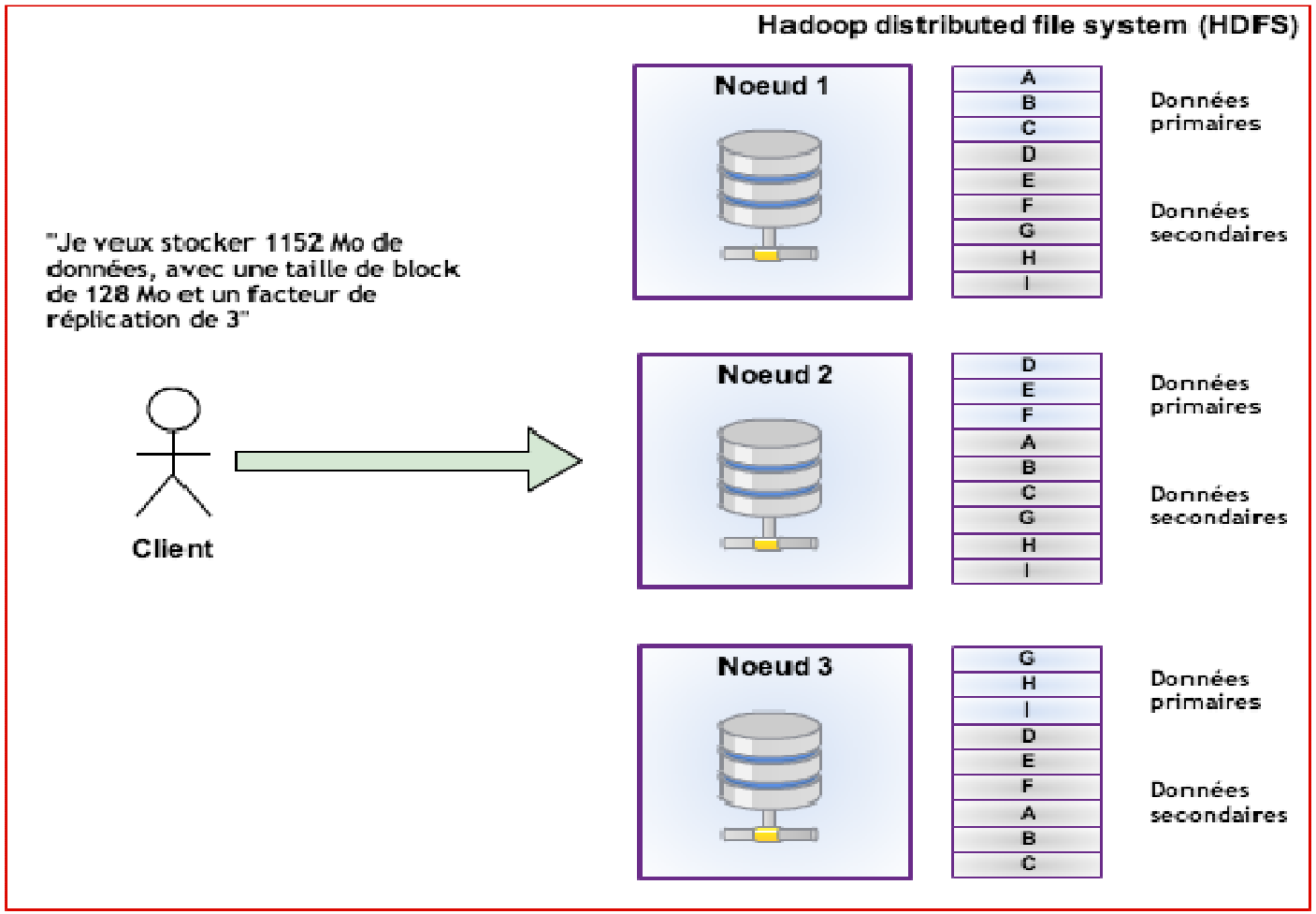
HDFS

Les contraintes

- HDFS est essentiellement conçu pour fonctionner en mode batch
- Les écritures ne se font pas de manière aléatoire dans un fichier
  - C'est une évolution prévue à terme
- HDFS procède plutôt par destructions de fichiers et/ou ajouts en fin de fichiers
- HDFS est très performant pour les écritures uniques et les lectures multiples
  - C'est la base du Big Data : « on prend tout, on ne détruit pas et on traite ensuite »
- Les petits fichiers ne sont pas toujours adéquats pour HDFS
  - Mieux vaut regrouper les données (concaténation par exemple)

Hadoop : la fiabilité

HDFS et répartition des données



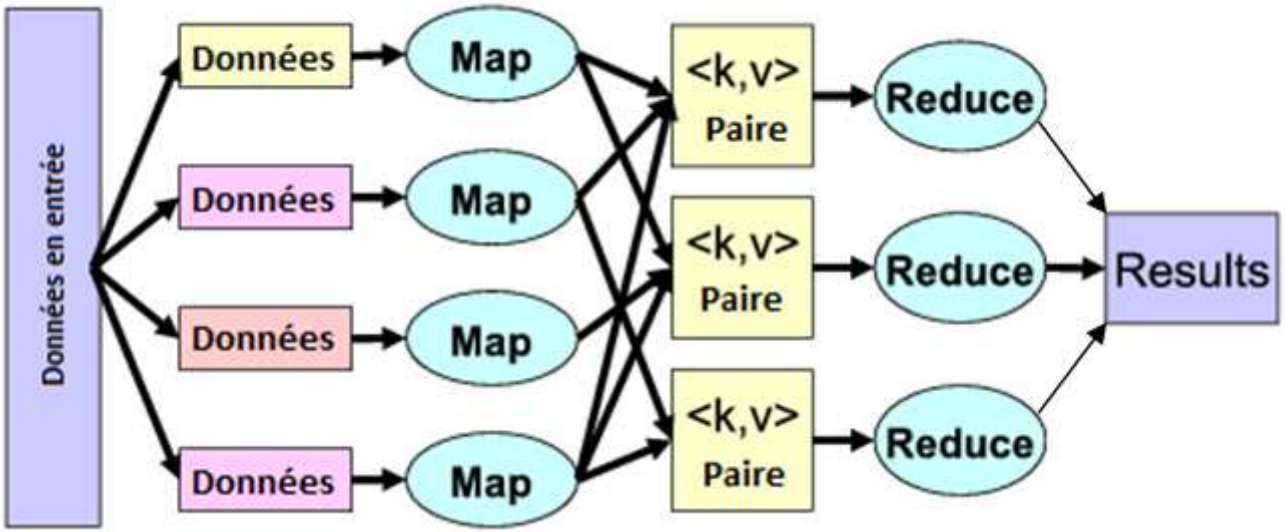
Répartition des données

Source : http://blog.ippon.fr/2013/05/14/big-data-la-jungle-des-differentes-distributions-open-source-hadoop/

Map/Reduce ?

Principes

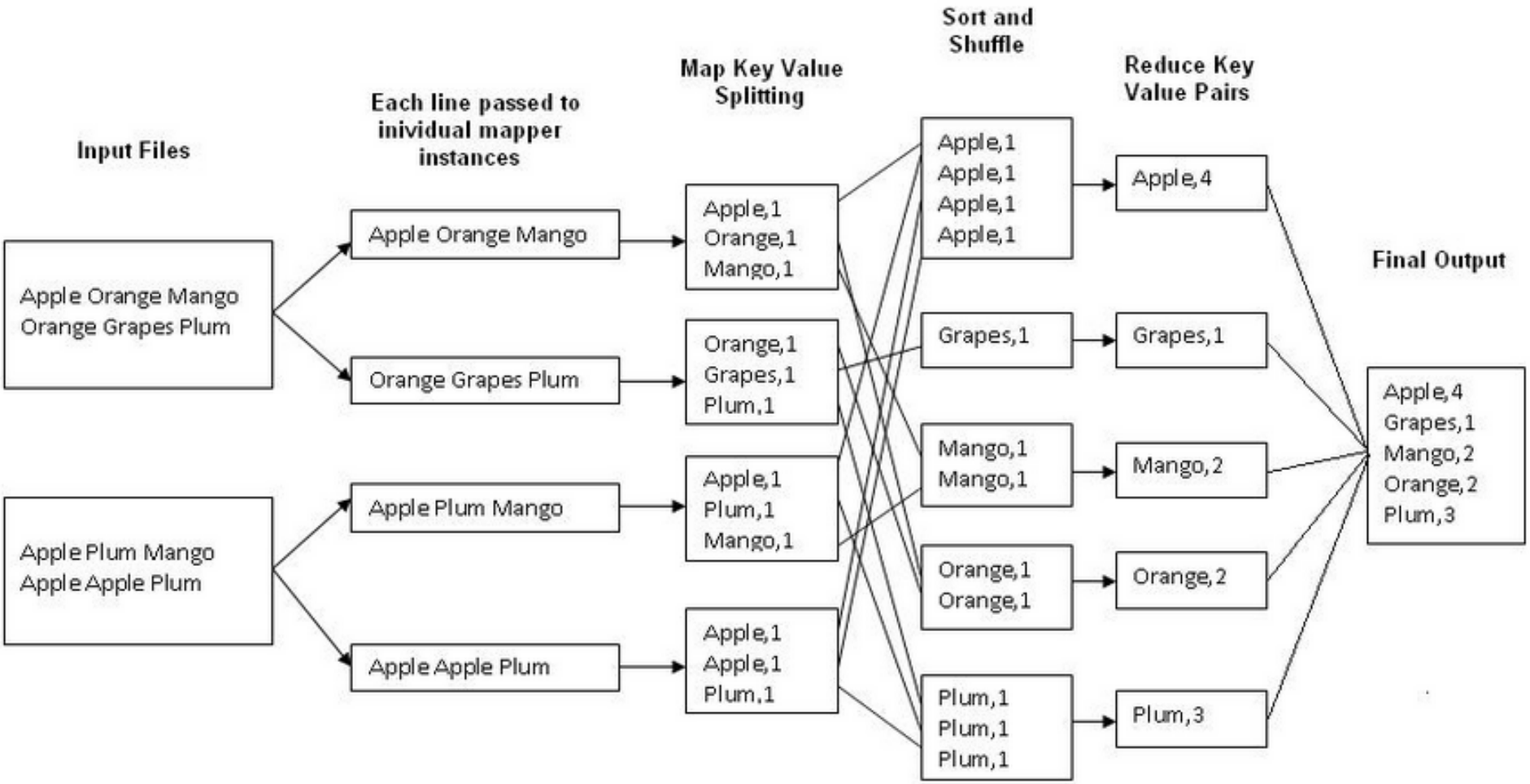
- Inventé par Google
- Permet de traiter des données en les distribuant dans un cluster
  - Hadoop
  - Apache Spark



Source : https://fr.wikipedia.org/wiki/MapReduce

Map/Reduce ?

Le célèbre exemple du “wordcount”

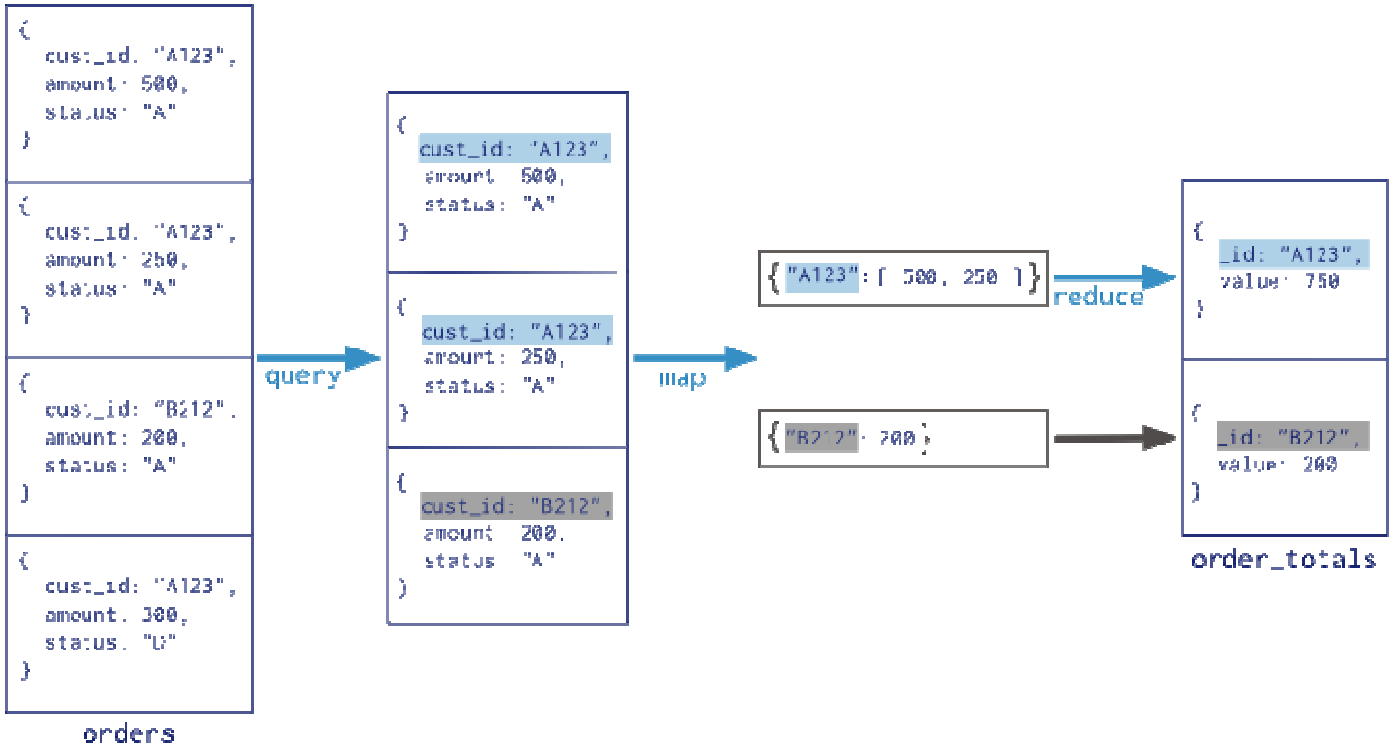


Source : http://javax4u.blogspot.fr/2012/11/hadoop.html

Map/Reduce ?

Principes

```
Collection
db.orders.mapReduce(
        map      →  function() { emit( this.cust_id, this.amount ); },
        reduce   →  function(key, values) { return Array.sum( values ) },
                 →  {
        query    →    query: { status: "A" },
        output   →    out: "order_totals"
                 →  }
                 )
```



Source : https://docs.mongodb.org/v3.0/_images/map-reduce.png

Map/Reduce

Requête en plusieurs étapes

- Définir une fonction de « map »

```
var mapFunction = function () {emit(this.year, 1);};
```

- Définir une fonction de « reduce »
  - Prend deux arguments:
    - l'identifiant du groupe auquel elle s'applique
    - la liste (un tableau en JavaScript) des valeurs produites par la fonction de « map ».

```
var reduceFunction = function (key, values) {return
Array.sum(values);};
```

- Définir un filtre (optionnel)

```
var queryParam = {query : {}, out : "result_set"};
```

- Lancer la requête

  ⓘ la sortie des fonctions « map » et du « reduce » doit être un document (pas un tableau)

```
db.publis.mapReduce(mapFunction, reduceFunction,
queryParam);
```

- Consulter le résultat

```
db.result_set.find();
```

Map/Reduce

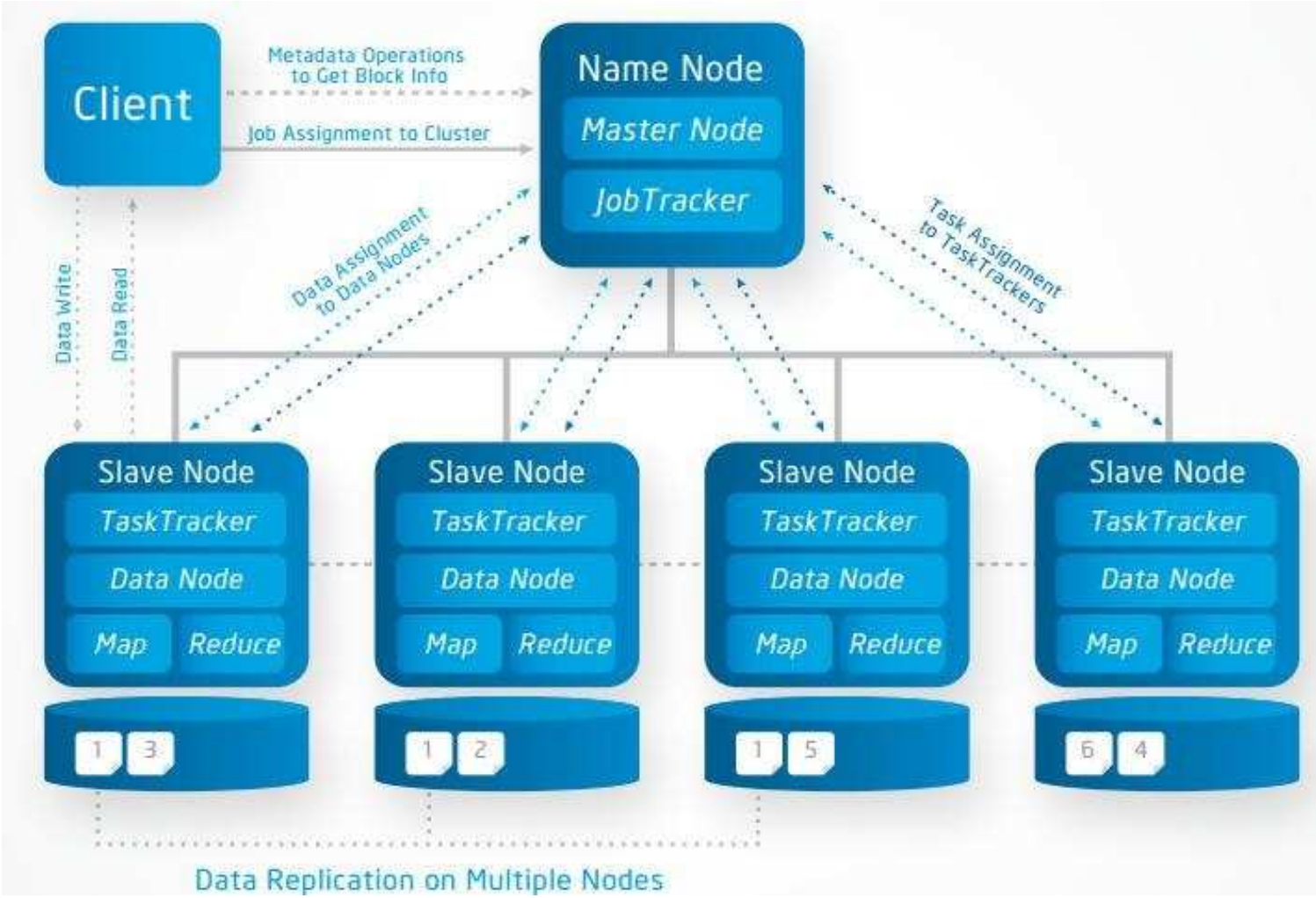
- A noter qu'il existe de larges discussions sur les avantages et inconvénients de Map/Reduce sur « aggregate »
- Par ailleurs, de nombreuses implémentations se font concurrence : Hadoop et Apache Spark notamment
- Références sur le sujet :
  - http://blog.mongodb.org/post/62900213496/qaing-new-code-with-mms-mapreduce-vs
  - https://sysdig.com/mongodb-showdown-aggregate-vs-map-reduce/

Les performances avec Spark

	Hadoop MR Record	Spark Record	Spark 1 PB
Data Size	102.5 TB	100 TB	1000 TB
Elapsed Time	72 mins	23 mins	234 mins
# Nodes	2100	206	190
# Cores	50400 physical	6592 virtualized	6080 virtualized
Cluster disk throughput	3150 GB/s (est.)	618 GB/s	570 GB/s
Sort Benchmark Daytona Rules	Yes	Yes	No
Network	dedicated data center, 10Gbps	virtualized (EC2) 10Gbps network	virtualized (EC2) 10Gbps network
Sort rate	1.42 TB/min	4.27 TB/min	4.27 TB/min
Sort rate/node	0.67 GB/min	20.7 GB/min	22.5 GB/min

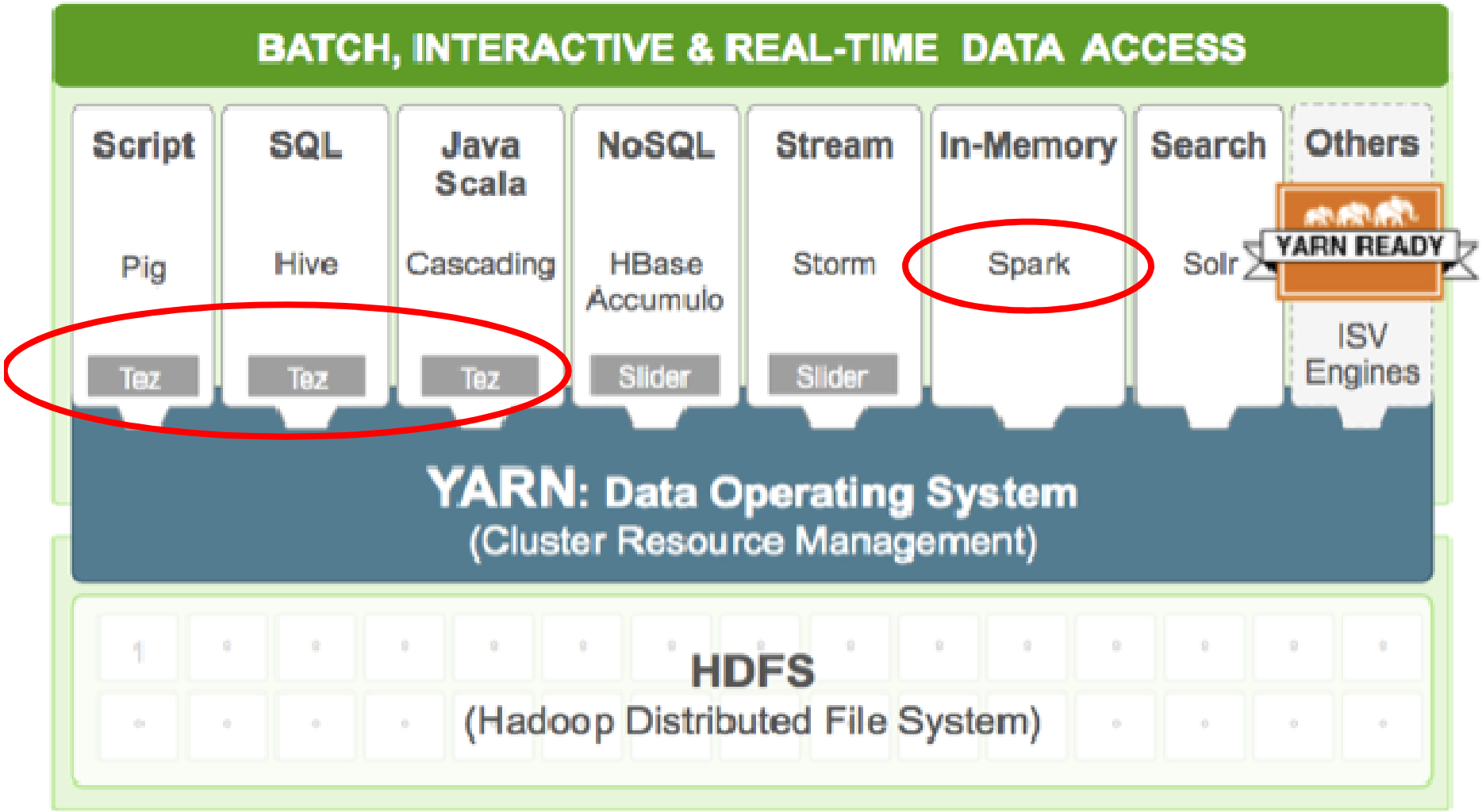
Hadoop : évolutions

- Hadoop : HDFS + MapReduce
- Le code est co-localisé avec les données

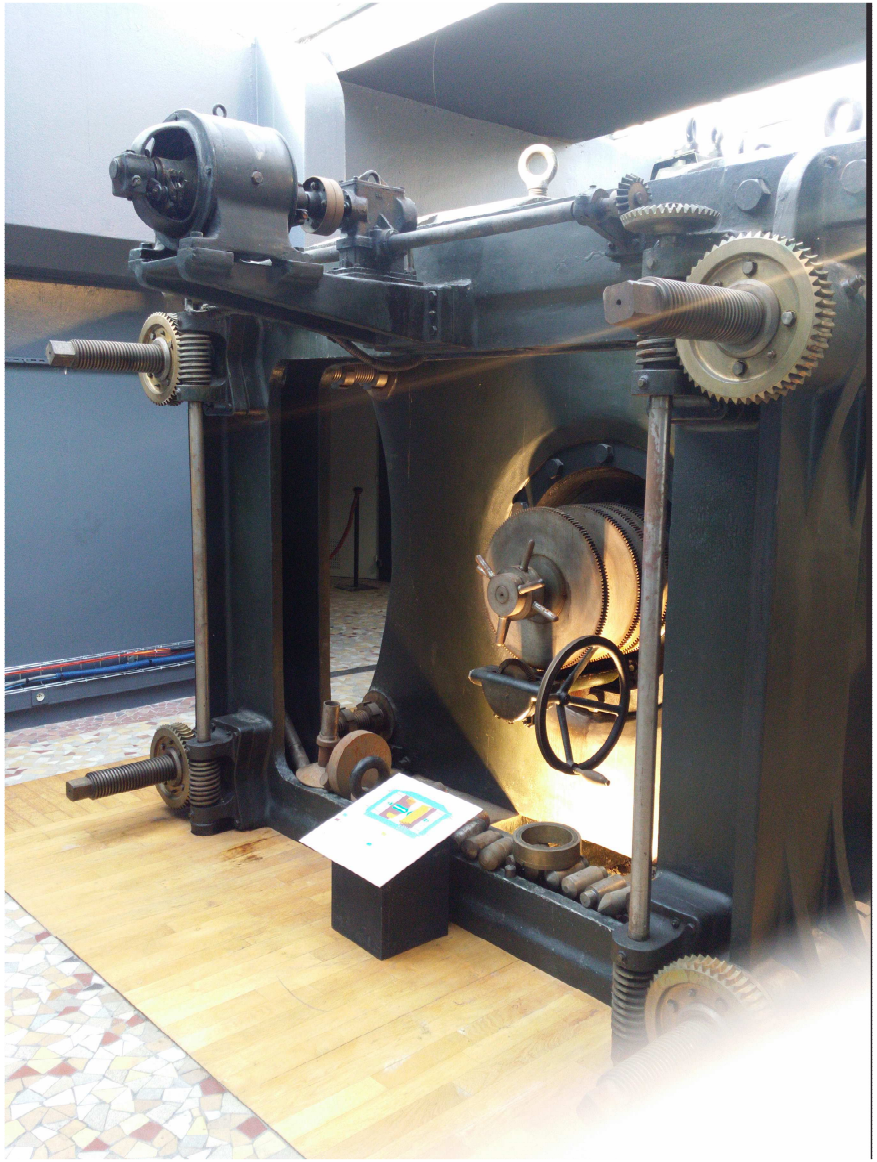


Hadoop : évolutions

Hadoop 2.0 : Tez et Spark

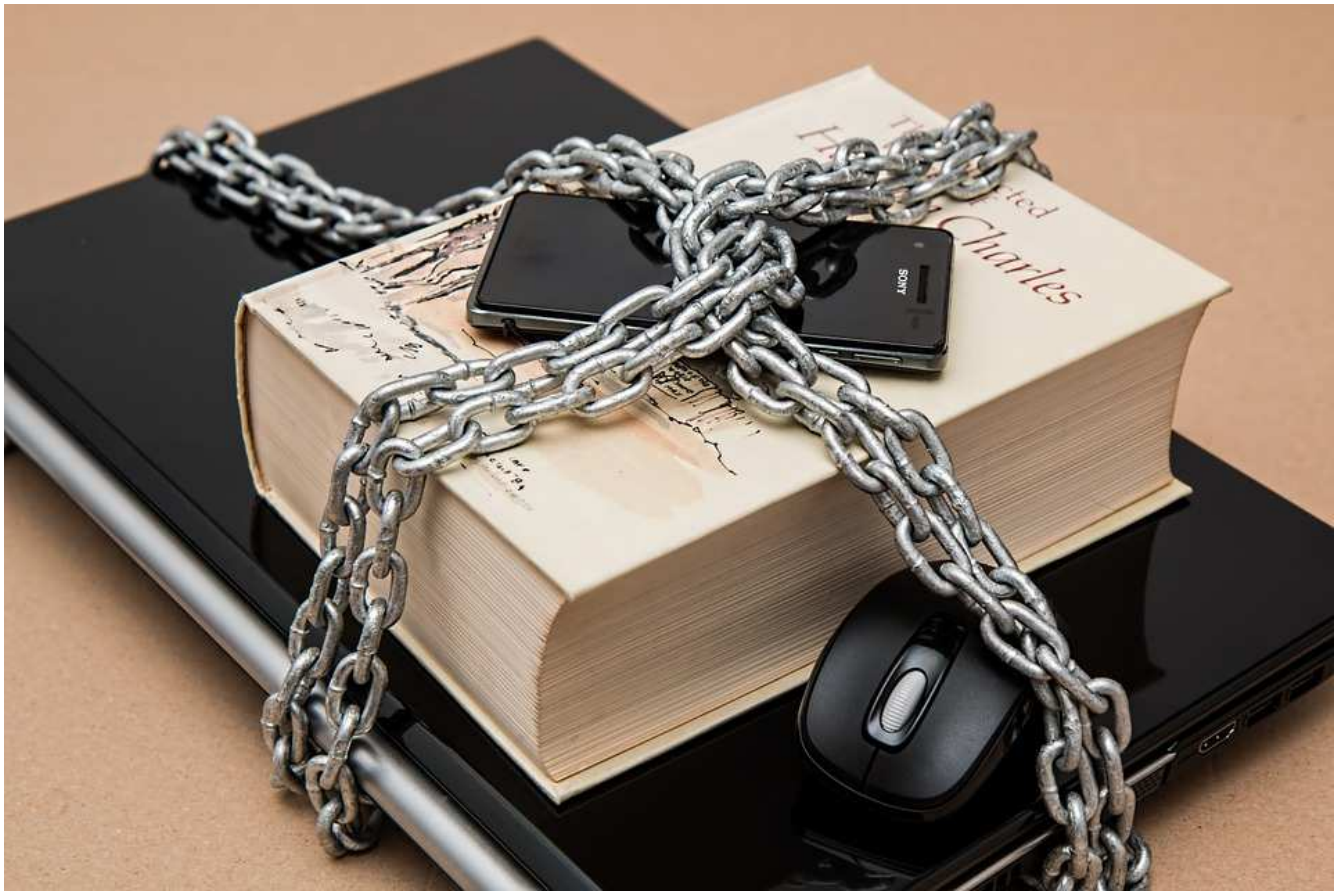


Manips en cours …



Electro-aimant Meudon – Nov. 2015

La sécurité



La sécurité

- La sécurité n'est pas la préoccupation première sur MongoDB
- Le principe est de considérer que seules les applications autorisées ont accès aux serveurs de données
- Ce principe peut paraitre très minimaliste mais peut se discuter dans un contexte où la haute performance être la règle
- Il est donc nécessaire de sécuriser l'infrastructure Mongo par des moyens complémentaires, si possible externes, qui n'entravent pas les performances
- Malgré tout, il est possible de créer une authentification de base sur MongoDB
  - Création d'utilisateurs
  - Mots de passe associés
- Le problème est que, selon les versions, cette fonctionnalité ne s'applique pas aux 'replica set' et au 'sharding' ce qui en limite la portée

La sécurité

Quelques commandes de base :

- Pour activer la gestion des comptes, il faut déjà activer cela au lancement du serveur MongoDB avec l'option '- -auth'
  - use testauth
  - db.createUser({user : 'opo',pwd:'opo', roles: ["readWrite", "dbAdmin"]
  - La liste des rôles prédéfinis se trouve ici : https://docs.mongodb.org/v3.0/reference/built-in-roles/#built-in-roles
  - Pour se connecter, il suffit ensuite de faire :
    - mongo –u opo –p opo - -authenticationDatabase testauth
  - Des rôles peuvent aussi être définis : voir documentation (https://docs.mongodb.org/v3.0/tutorial/manage-users-and-roles/)
  - A ce stade, les opérations habituelles sont possibles sur la base 'testauth'

Des questions ?