

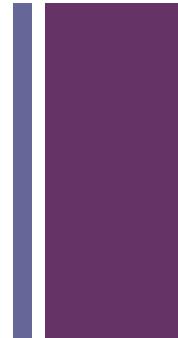
+



Initiation NodeJS, Express  
& MongoDB

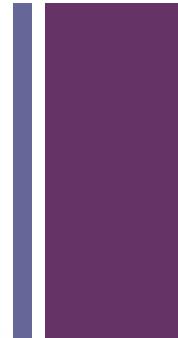
+

# Formateur : Qui suis-je?



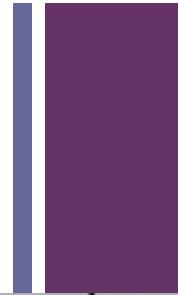
+

Et vous?



+

Et vos attentes pour ces 5 jours?





# Ce que nous allons aborder

(nous y rajouterons les attentes exprimées si elles ne sont pas abordées ici)

- Installer et configurer un serveur NodeJS
- Mettre en œuvre les concepts de la programmation événementielle et asynchrone
- Mettre en place un Framework Web (Geddy, Express)
- Apprendre à créer des WebServices REST
- Manipuler l'API de NodeJS
- Gérer la persistance dans une base de données NoSQL à la main
- Gérer la persistance dans une base de données NoSQL avec des langages classiques : NodeJS, Java, etc.
- Comprendre le principe derrière les BDD NoSQL
- Installer et administrer MongoDB
- Maintenir et optimiser MongoDB

+

## Du JavaScript côté serveur

# Genèse du JavaScript côté serveur et de Node.js

- JavaScript a toujours été l'un de mes langages préférés. Il est souvent considéré comme un langage complexe à maintenir, probablement à cause des possibilités syntaxiques qu'il offre (*closures, notation JSON...*), mais en réalité il ne demande peut-être qu'un peu plus de rigueur que d'autres langages.
- Depuis sa création, JavaScript a naturellement été associé aux sites et applications web, domaine pour lequel il a été créé. Il a permis de rendre les pages web plus dynamiques, en commençant par des animations très *flashy* dans les années 1990/2000 (ce que l'on appelait le DHTML), pour en arriver vers les applications les plus complexes que nous connaissons tous aujourd'hui et qui se substituent aux applications de bureau : webmail avec Gmail, cartographie avec Google Maps, etc.

# Le moteur Google V8 utilisé côté serveur

- Mais si JavaScript a été créé plus spécialement pour être exécuté dans un navigateur, il a été standardisé et permet théoriquement d'être utilisé pour tout type d'application, et notamment des programmes autonomes, c'est-à-dire ne nécessitant pas de navigateur web pour s'exécuter. C'est sur cet usage que se base Node.js.
- Node.js se constitue d'un programme (appelé *node*) permettant d'interpréter du code JavaScript, traditionnellement en ligne de commande. Il est également accompagné d'outils supplémentaires, dont le plus intéressant s'appelle *npm*, pour *Node package manager*.

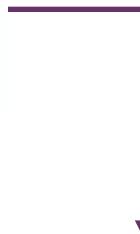
# Le moteur Google V8 utilisé côté serveur

- Node.js utilise le moteur d'exécution ultrarapide V8 de Google Chrome. Ce moteur V8 avait fait beaucoup parler de lui à la sortie de Google Chrome, car c'est un outil open source créé par Google qui analyse et exécute du code JavaScript très rapidement.
- Jusqu'à la sortie de Chrome, la plupart des navigateurs lisaien le code JavaScript de façon peu efficace : le code était lu et interprété au fur et à mesure. Le navigateur mettait beaucoup de temps à lire le JavaScript et à le transformer en code machine compréhensible pour le processeur.
- Le moteur V8 de Google Chrome, qui est réutilisé ici par Node.js, fonctionne complètement différent. Très optimisé, il fait ce qu'on appelle de la compilation JIT (Just In Time). Il transforme le code JavaScript très rapidement en code machine et l'optimise même.

# Pourquoi utiliser la programmation événementielle ?

- Comme JavaScript est un langage conçu autour de la notion d'évènement, Node.js a pu mettre en place une architecture de code entièrement non bloquante. Différence entre un code bloquant et un code non bloquant?

Télécharger un fichier  
Afficher le fichier  
Faire autre chose



Télécharger un fichier  
Dès que c'est terminé, afficher le fichier  
Faire autre chose

+

## Rappels JavaScript : les bases, callbacks, closures, notion de scope...

Je pars du principe que vous avez déjà utilisé JavaScript...



+

## Premiers pas en NodeJS



# Installation du serveur Node.js et création d'un serveur Web en quelques lignes

- Node est un outil multiplateforme. Il est possible très facilement de l'utiliser sous Linux, Windows, OS X, etc. Son installation est très simple !

Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, npm, is the largest ecosystem of open source libraries in the world.

Important security release for 8.x, please update now!

Download for macOS (x64)

**6.11.4 LTS**

Recommended For Most Users

**8.6.0 Current**

Latest Features

[Other Downloads](#) | [Changelog](#) | [API Docs](#)

[Other Downloads](#) | [Changelog](#) | [API Docs](#)

Or have a look at the [LTS schedule](#).

Sign up for [Node.js Everywhere](#), the official Node.js Weekly Newsletter.

# Installation du serveur NodeJS et création d'un serveur Web en quelques lignes

Exemple :

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World\n');
}).listen(8888, '127.0.0.1');
console.log('Server running at http://127.0.0.1:8888/');
```

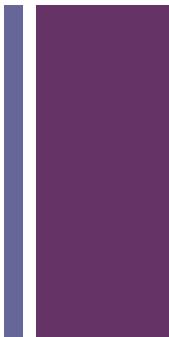
Exécution :

```
$ node example.js
Server running at http://127.0.0.1:8888/
```

Vous pouvez recopier ce code et le tester !



# L'approche modulaire de Node.js



Pour organiser votre programme, vous pouvez définir des modules :

```
var tools = require('./tools.js');
console.log( 'Aire = ' + tools.circleArea(4));
```

Le fichier `tools.js` :

```
var PI = Math.PI;

module.exports.circleArea = function (r) {
    return PI * r * r;
}

module.exports.rectangleArea = function (w, h) {
    return w*h;
}
```

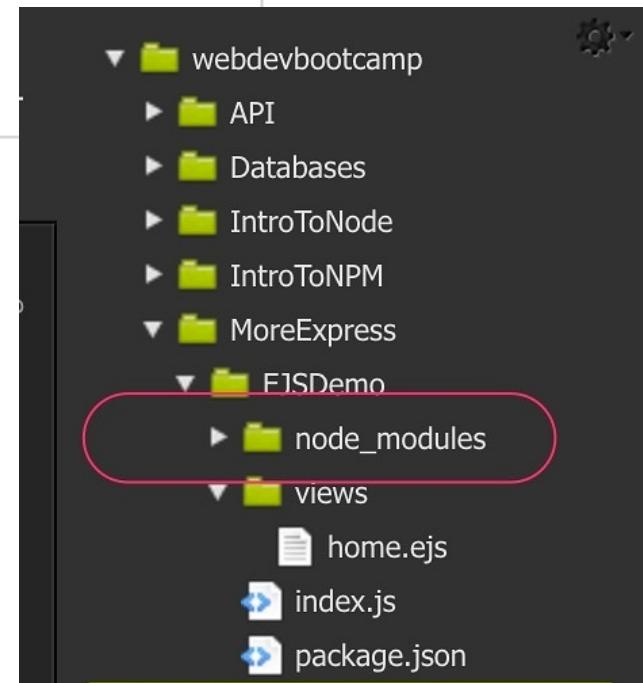
Vous pouvez recopier ce code et le tester !



# Le gestionnaire d'extensions NPM

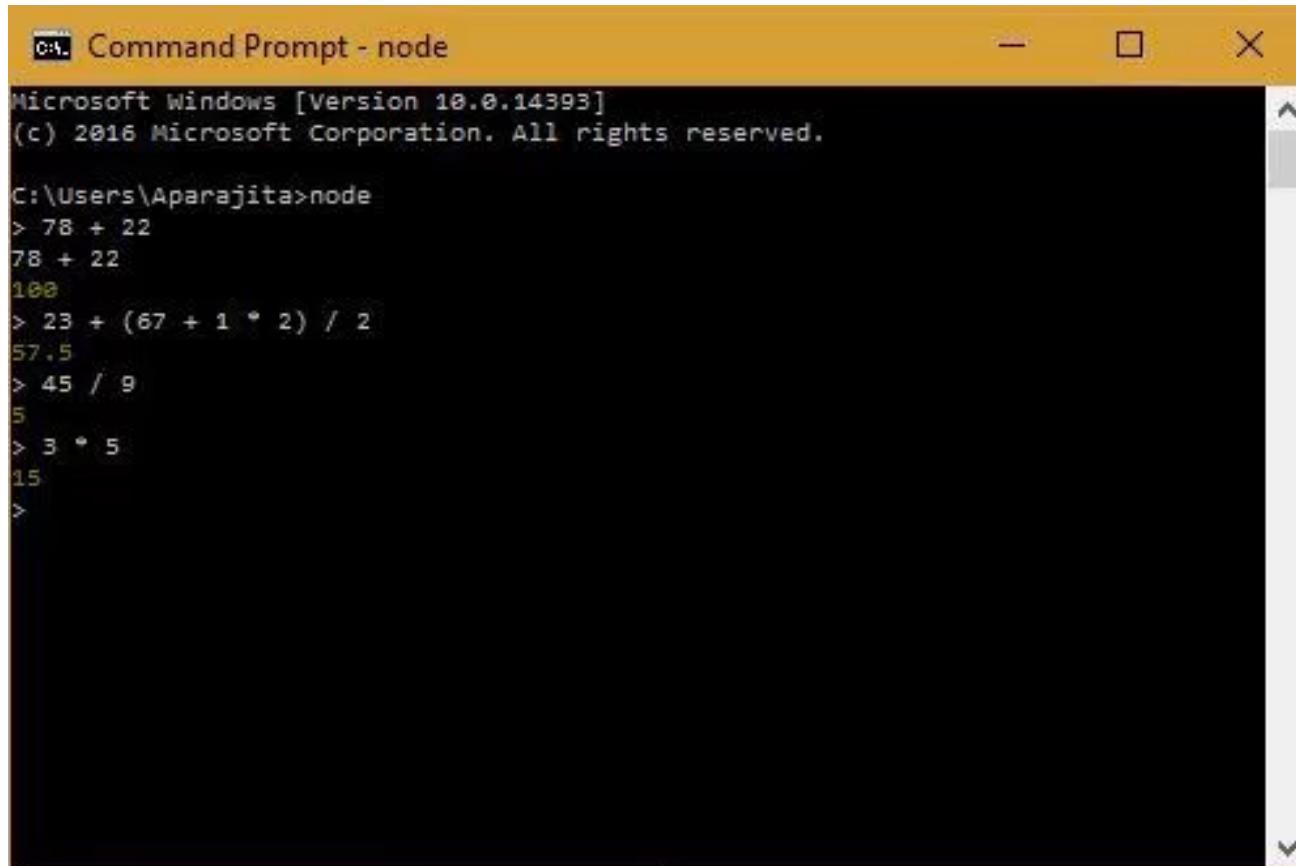
Pour télécharger et installer des modules, vous pouvez utiliser npm :

```
$ npm install express
npm http GET https://registry.npmjs.org/express
...
...
```



```
var express = require('express');
var app = express();
app.get('/t.txt', function(req, res){ res.send('t'); });
app.listen(8080);
```

# Utilisation de Node.js en REPL (Read-Eval-Print-Loop)



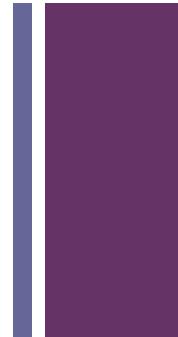
The screenshot shows a Windows Command Prompt window with a yellow title bar. The title bar reads "Command Prompt - node". The window itself has a black background and contains the following text:

```
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\Aparajita>node
> 78 + 22
78 + 22
100
> 23 + (67 + 1 * 2) / 2
57.5
> 45 / 9
5
> 3 * 5
15
>
```

# Utilisation de Node.js en REPL (Read-Eval-Print-Loop)

```
deepakverma@deepakverma: ~
deepakverma@deepakverma:~$ node
> var evon=1
undefined
> evon
1
> var findnerd = 12
undefined
> findnerd*evon
12
> findnerd+evon
13
> var master = function(num){
... console.log("Please check the number" + num);}
undefined
> master()
Please check the numberundefined
> master(6)
Please check the number6
undefined
> █
```



## EXERCICE

**Travaux pratiques réalisés :** *Usage de l'utilitaire NPM.*  
*Développer une première application Web affichant «Hello World».*

+

## **Les fondamentaux de NodeJS & Express (framework web)**

+

# Quel intérêt de développer en asynchrone ?



Rapidité, Rapidité, Rapidité !!!

# Gestion basique des requêtes/réponses HTTP & HTTPS

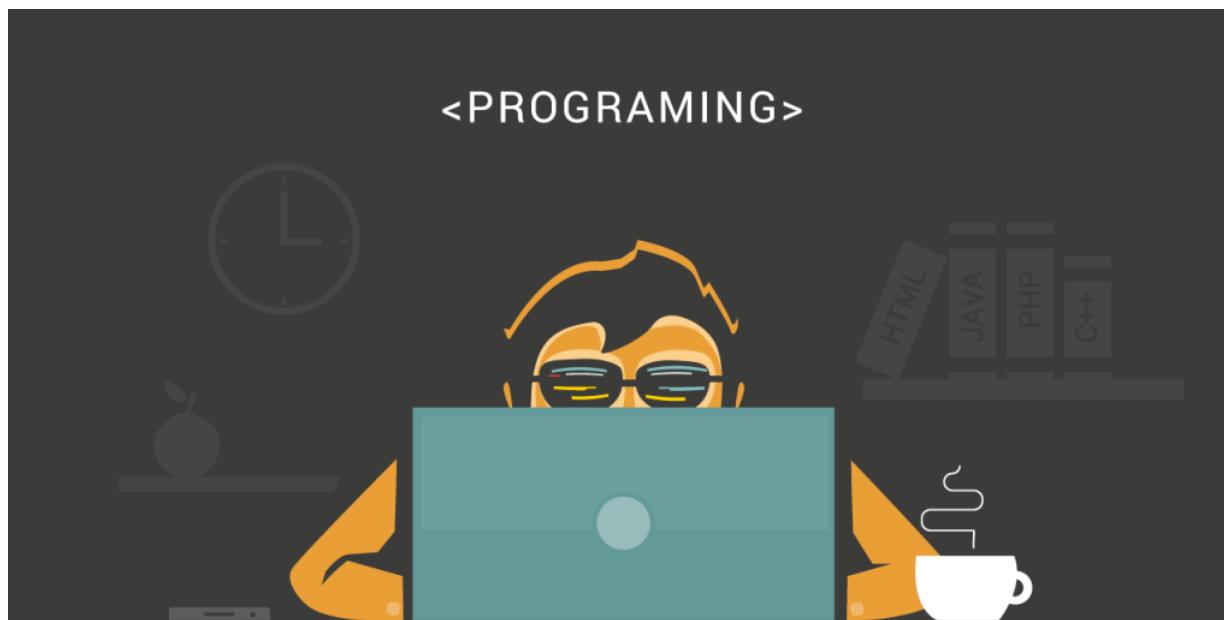
```
var http = require('http');
var url = require('url');
var fs = require('fs');

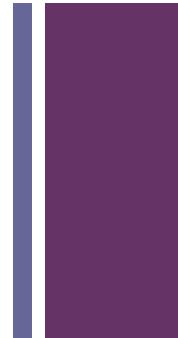
var server = http.createServer(function (req, res) {
  var url_parts = url.parse(req.url, true);
  var name = url_parts.query.name;
  if (name) {
    console.log('Name: ' +name);
    res.writeHead(200, {'Content-Type': 'application/json'});
    res.end(JSON.stringify({message: 'Hello ' +name + '!'}));
  } else {
    console.log('No name!');
    res.writeHead(200, {'Content-Type': 'text/html'});
    fs.readFile('hello02.html',function (err,data) {
      res.end(data);
    });
  }
}).listen(1337, '127.0.0.1');

console.log('Server running at http://127.0.0.1:1337/');
```

**Vous pouvez recopier ce code et le tester !**

Un développeur doit être capable de trouver les ressources nécessaires pour résoudre toutes les problématiques qui lui sont présentées. Vous allez avoir une série d'exercices pour maîtriser divers aspects de NodeJS. Je serai à votre disposition de même que Google, StackOverflow et la documentation officielle de NodeJS (<https://nodejs.org/dist/latest-v8.x/docs/api/>)





## EXERCICE

**Travaux pratiques :** *Création et lecture d'un fichier texte.*  
*Lecture d'une ressource en ligne. Apprendre à créer des modules JS. Création d'un serveur et gestion de plusieurs pages (contact, about, home, etc.). Calculatrice. Etc.*

# Les concepts fondamentaux d'Express

- Lorsque l'on crée un site ou une application web, il est nécessaire que celui /celle-ci soit bien architecturé(e). C'est justement le but d'un framework. À l'instar d'autres frameworks comme Symfony et Ruby On Rails, Express permet d'organiser votre application web en fonction des requêtes qu'elle est censée recevoir. Vous l'aurez compris, contrairement aux autres frameworks cités précédemment, Express reste dans la philosophie de Node.js en étant léger et très simple à mettre en œuvre.
- Contrairement à *http* ou *url* que nous avons utilisés précédemment, Express (qui se constitue principalement du module *express*) ne fait pas partie de la distribution de base de Node.js. Pour l'inclure à notre application, il faut l'inclure !

```
> npm install express
npm http GET https://registry.npmjs.org/express
```



# Configuration d'Express et construction d'un squelette d'application

```
var express =require('express');
var fs = require('fs');
var app = express();

app.get('/',function(req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  fs.readFile('express01.html', function (err,data) {
    res.end(data);
  });
});

app.get('/hello/:name', function(req,res) {
  res.writeHead(200, {'Content-Type': 'application/json'});
  res.end(JSON.stringify({message: 'Hello ' + req.params.name + '!'}));
});

app.listen(8080);
```

Le code n'est-il pas plus « propre » qu'avec une succession de IF?



## Le rendu de vues avec EJS

- Si vous avez déjà développé un site web, vous connaissez sans doute le principe des templates. Le but est d'écrire la vue (traditionnellement du HTML) séparément du code métier. Dans l'idéal, un webdesigner ne connaissant pas du tout le code métier (qu'il soit JavaScript, PHP, Ruby. . . ) doit être capable d'écrire le template.

# Le rendu de vues avec EJS

```
var express = require('express');
var app = express();
app.engine('.html', require('ejs').__express);
app.set('views', __dirname + '/views');
app.set('view engine', 'html');

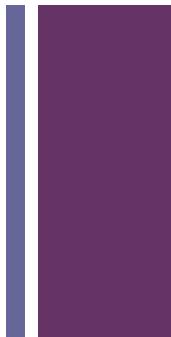
var users = [{name:"joe",age:21},{name:"bob",age:22}];
app.get('/', function(req, res){
  res.render('users', {
    users: users,
  });
});

app.listen(8080);
```

Vous pouvez recopier ce code et le tester !



# Le rendu de vues avec EJS



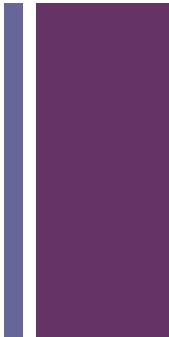
Le fichier users.html :

```
<!doctype html>
<html>
<head>
  <title>Example</title>
</head>
<body>
  Users :
  <ul>
    <% for (i in users) { %>
    <li><%= users[i].name %> : <%= users[i].age %></li>
    <% } %>
  </ul>
</body>
</html>
```

**Vous pouvez recopier ce code et le tester !**



# La gestion de formulaires



## La partie HTML

```
<form method="post" action="/">
  <input type="text" name="user[name]">
  <input type="text" name="user[email]">
  <input type="submit" value="Submit">
</form>
```

## La partie NodeJS

```
app.use(express.bodyParser());

app.post('/', function(request, response){
  console.log(request.body.user.name);
  console.log(request.body.user.email);
});
```

**Vous pouvez recopier ce code et le tester !**

# La gestion des upload de fichiers

## Formidable

build status [passing](#)



*File Upload using Formidable*

### Purpose

A node.js module for parsing form data, especially file uploads.

### Current status

This module was developed for [Transloadit](#), a service focused on uploading and encoding images and videos. It has been battle-tested against hundreds of GB of file uploads from a large variety of clients and is considered production-ready.

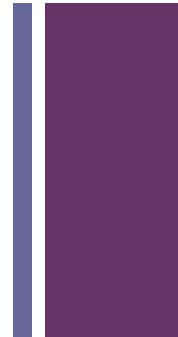
### Features

- Fast (~500mb/sec), non-buffering multipart parser
- Automatically writing file uploads to disk
- Low memory footprint
- Graceful error handling
- Very high test coverage

### Installation

Via [npm](#):

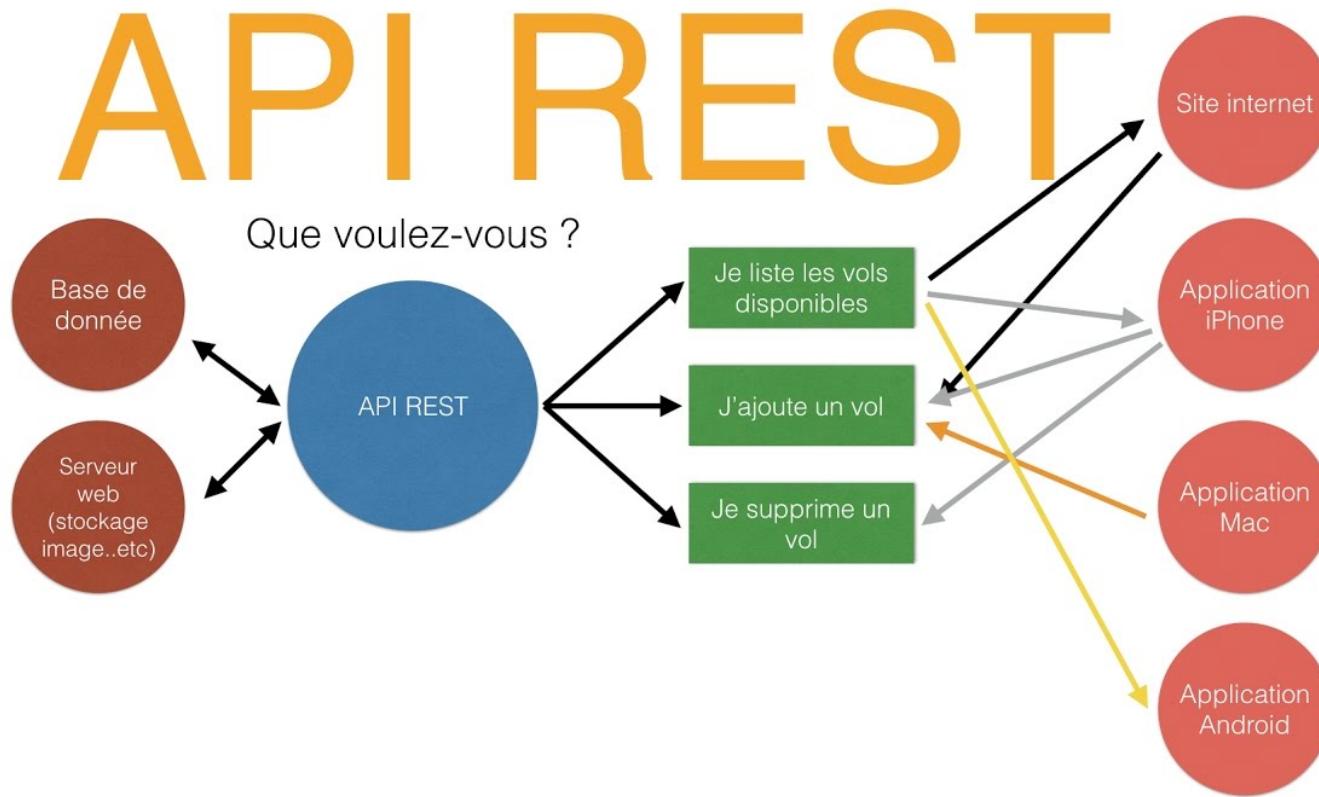
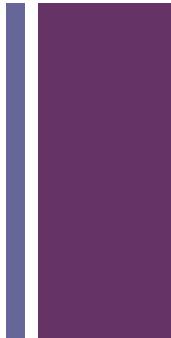
```
npm install formidable@latest
```

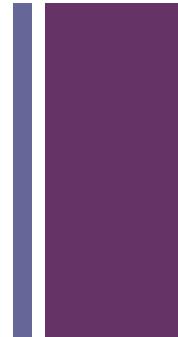


## EXERCICE

**Travaux pratiques :** *Intégrer votre formulaire dans NodeJS grâce à Express et préparer le formulaire (code analyse des données après Submit) qui va recueillir les informations du formulaire*

# Mise en place d'une API REST





# EXERCICE

**Travaux pratiques : Mise en place d'une API REST**

+

**Développement d'une application  
complète**



## Exercice avec NodeJS & Mongo



Réaliser un moteur de recherche de photos qui exploite l'API de Yahoo Flickr (<https://www.flickr.com/services/api/> ), NodeJS et (en fin de semaine uniquement) MongoDB.

Ce moteur de recherche doit contenir un champ de recherche, des champs de filtrage (taille de l'image, date minimum d'upload de la photo, date maximale d'upload de la photo, tri grâce à divers paramètres, recherche NSFW, tags supplémentaires, appartient ou non à une galerie, etc.) et une zone qui permet de voir les photos (URLs ou photos complètes).

En sélectionnant sur une photo, on doit pouvoir obtenir les données (nom, etc.) de son auteur, les autres photos de l'auteur, les commentaires de la photo, la position géographique et tout autre élément que vous pourrez récupérer.

*Les données doivent être sauvegardées dans une base MongoDB pour être réutilisées facilement et rapidement.*

+

# **Introduction au NoSQL & Fonctionnement**

# Les bases relationnelles et SQL ?

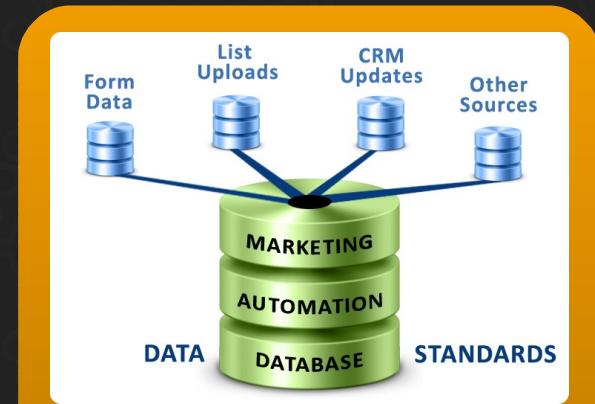
- Modèle prédominant pour stocker de l'information depuis + de 30 ans !



Nécessité de nombreuses lectures / écritures en simultané. Les RDBMS proposent un système de gestion de transaction efficace permettant d'éviter le pire !



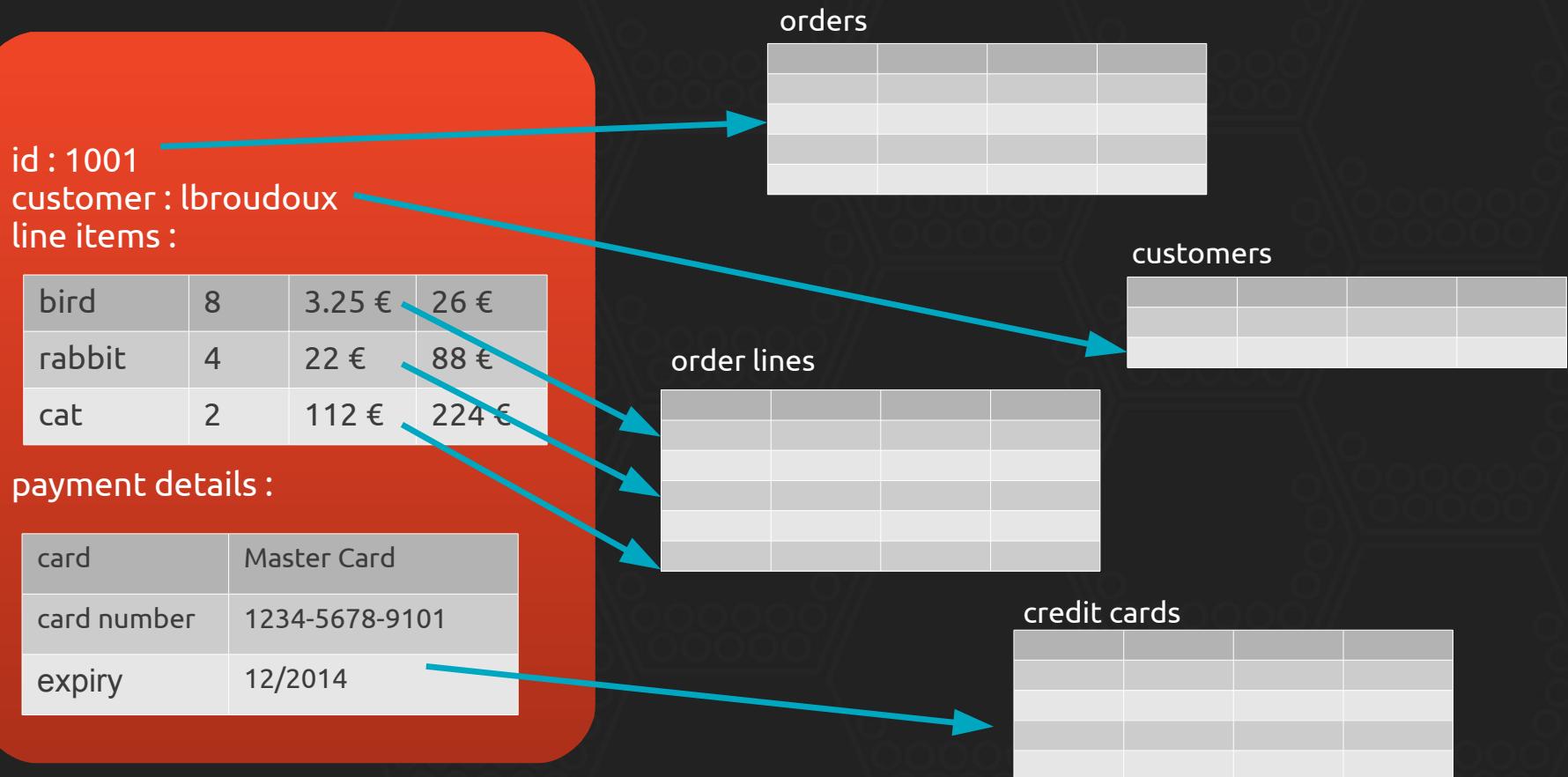
Ecosystème riche et collaboration indispensable. L'écriture puis les lectures dans un RDBMS est un modèle fréquent d'intégration applicative.



Les RDBMS ont imposé un paradigme de modélisation et un langage de requête (SQL) standards globalement partagés entre tous les vendeurs.

# Les limites des bases relationnelles (i)

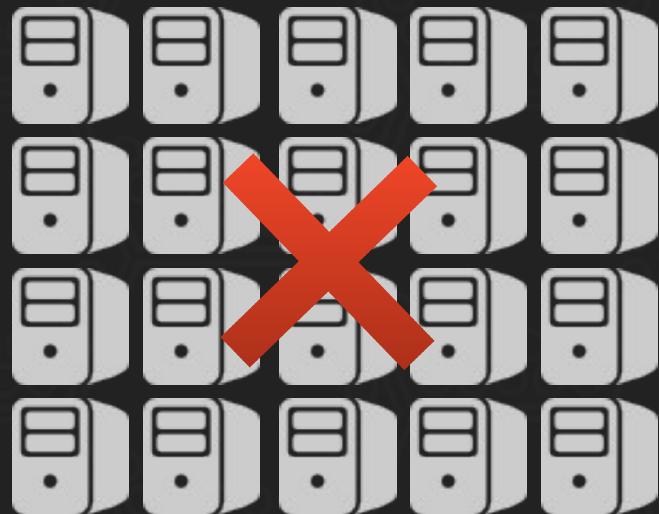
## *Impedance mismatch*



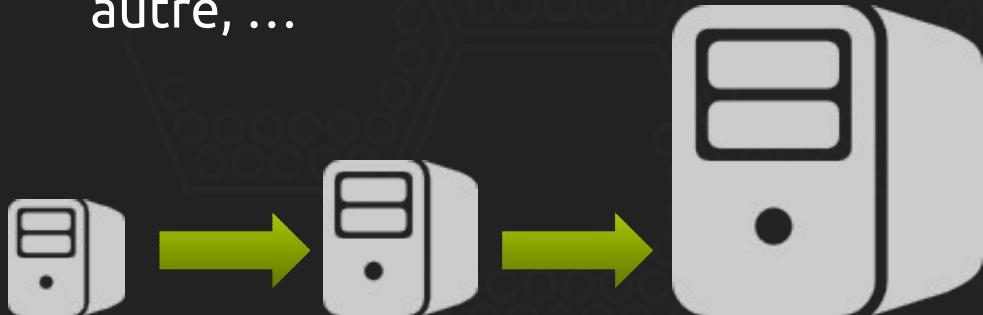
# Les limites des bases relationnelles (ii)

## Scalabilité

Le modèle de consistance des RDBMS empêche l'utilisation de plusieurs machines pour répartir la charge (au moins en écriture)



Pour augmenter la performance d'accès à la base, pas d'autres moyens que d'acheter un plus gros serveur, puis un autre, puis un autre, ...



# Eléments de contexte

- Plusieurs tendances forment le terreau d'un changement
  - La SOA (Service Oriented Architecture)
    - Intégration applicative maintenant basée sur la notion de service
  - Le cloud et les besoins en très haute disponibilité
    - Prédominance des *clusters*
    - Approche *commodity hardware*
  - La réalité économique
    - Coût des machines très haute performance
    - Facturation « au cœur » par les vendeurs

# NoSQL ?



NoSQL s'écarte du modèle de données relationnel pour généralement proposer un modèle par agrégat.

Terme apparu en 2009 avec plusieurs interprétations possibles :

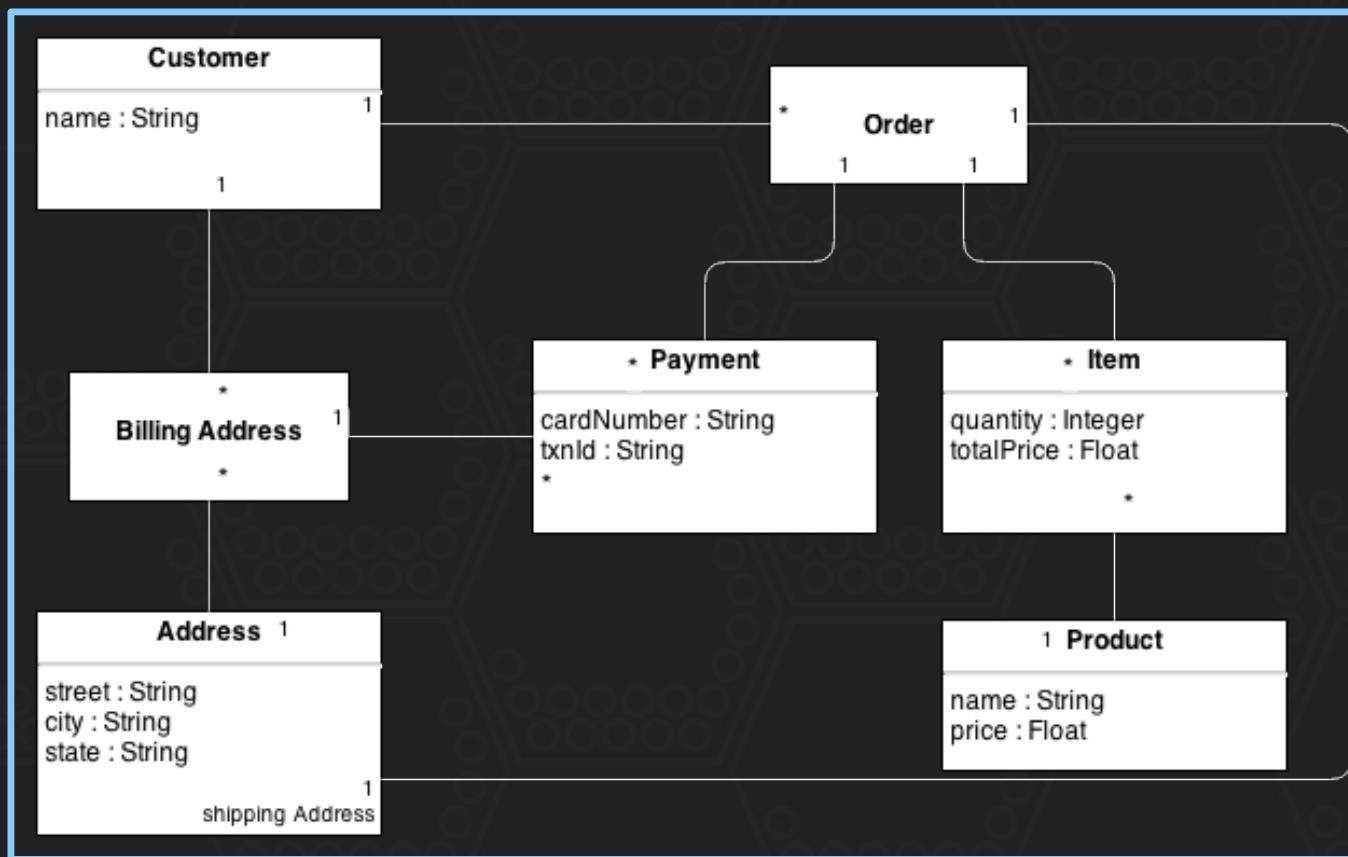
- No SQL
- Not Only SQL

Pas de définition formelle mais des caractéristiques communes partagées par les bases dites NoSQL :

- N'utilise pas de modèle relationnel et donc pas le langage SQL
- Conçu pour être exécutée dans un cluster,
- Tendance à être Open Source,
- Généralement sans schéma et donc permettant de stocker n'importe quelle donnée dans n'importe quelle « ligne »

# Relations & Agrégats (i)

- Soit un modèle relationnel exemple, exprimé en UML, en utilisant des associations



# Relations & Agrégats (ii)

- Soit sa projection sur le modèle relationnel physique
  - Normalisation, pas de répétition, agnostique à l'usage

Customers

Id	Name
123	Broudoux

Orders

Id	CustomerId	ShippingAddressId
99	123	66

Products

Id	Name	Price
27	Rabbit	3.25

BillingAddress

Id	CustomerId	AddressId
55	123	66

Items

Id	OrderId	ProductId	Quantity	Price
991	99	27	3	9.75

Address

Id	City	Street
66	Parigne Le Polin	

Payments

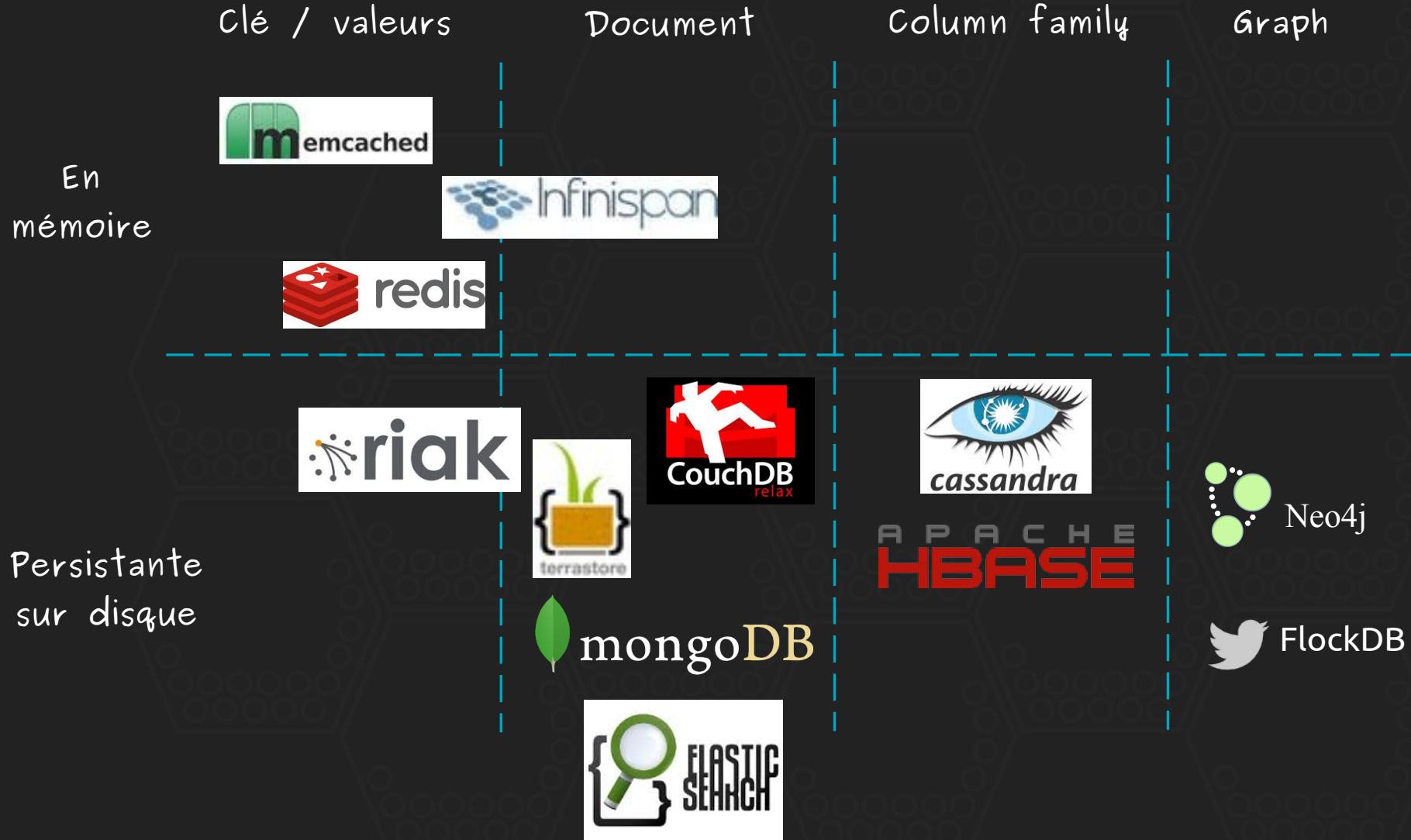
Id	OrderId	CardNumber	TxnId	BillingAddressId
991	99	1234-4567-8910	a23ef75cd65b78	66

# Relations & Agrégats (iv)

```
1 // in customers
2 {
3     id: 123,
4     name: "Broudoux",
5     billingAddress: [{city: "Parigne le Polin"}]
6 }
7
8 // in orders
9 {
10     id: 990123,
11     customerId: 123,
12     items:[
13         productId: 27,
14         price: 3.25,
15         name: "Rabbit",
16         quantity: 3,
17         totalPrice: 9.75
18     ],
19     shippingAddress: {city: "Parigne le Polin"},
20     payment: [
21         cardNumber: "1234-4567-8910",
22         txnId: "a23ef75cd65b78",
23         billingAddress: {city: "Parigne le Polin"}
24     ]
25 }
```

- Soit sa projection en JSON (notation communément utilisé dans le monde NoSQL)
  - Dénormalisation
  - 2 agrégats principaux
  - Relations entre agrégats arbitraires et assurées applicativement

# Panorama des solutions



# Bases clés / valeurs (i)



C'est quoi ?

- Grossièrement : une simple hash table où tous les accès se font en utilisant la clé primaire
- Seulement 3 opérations possibles :
  - Put : donner une valeur à une clé
  - Get : récupérer la valeur d'une clé
  - Delete : effacer la clé et sa valeur
- Support de structures basiques (list, hash, set)
- Parfois, notion de *bucket* ou couplage à un moteur d'indexation (ex : Riak)
- Très haute performance (*in-memory* possible)
- RESTful !

# Bases clés / valeurs (ii)



A utiliser pour ...

- De l'information très volatile
  - Session utilisateur, données d'un panier d'achat
- De l'information très peu volatile et accédée très fréquemment
  - Descriptions produits, paramétrage applicatif



A éviter pour ...

- Des données possédant des relations
  - Relations entre agrégats ou corrélation entre données de différents ensemble de clés
- Des opérations impliquant de multiples clés
- Des besoins de requêtage par les données

# Bases clés / valeurs (iii)



Comment ?

```
byte[] value = new String("My Value").getBytes();
Bucket bucket = client.fetchBucket("sessions").execute();
bucket.store(key, value).execute();
```

Ecriture d'une paire  
clé/valeur

```
Bucket bucket = client.fetchBucket("sessions").execute();
IRiakObject riakObject = bucket.fetch(key).execute();
byte[] bytes = riakObject.readValue();
String value = new String(bytes);
```

Lecture d'une valeur

```
Bucket bucket = client.fetchBucket("sessions").execute();
DomainBucket<UserSession> sessionBucket = DomainBucket.builder(bucket, UserSession.class).build();
```

Construction d'un objet domain complexe

# Bases document (i)



C'est quoi ?

- Bases stockant des documents qui peuvent être des arbres Xml, JSON, BSON, etc ...
  - Pensez à des bases clés-valeurs où le contenu est examinable
- Documents souvent regroupés par Collection
  - 2 documents de la même Collection ne possèdent pas nécessairement la même structure
- Requêtes possibles en utilisant des syntaxes analogues à Xpath, Xquery, Javascript, JXPath
- Parfois, fonctions d'agrégation : sum, count, group

# Bases document (ii)



A utiliser pour ...

- Données avec partie structurée et partie non structurée
  - Evénements des applicatifs (*sharding* possible par application)
- Données de publication variables
  - CMS, Blogging avec commentaires, contenu dynamique, etc ...
- Données de suivi temps réel ou analytiques



A éviter pour ...

- Opérations nécessitant consistance sur plusieurs agrégats
- Des structures d'agrégat très changeante avec des besoins de requêtage forts
  - Inconvénient du *schemaless*

# Bases document (iii)



Comment ?

```
Mongo mongo = new Mongo("localhost:271017");
mongo.slaveOk();

DBCollection orders = mongo.getDB("shopping").getCollection("orders");
BasicDBObject order = new BasicDBObject("customerId", 123)
    .append("name", "lbroudoux")
    .append("items", items);      // items is a List<BasicDBObject>
orders.insert(order);
```

Ecriture d'un  
document

```
DBCollection orders = mongo.getDB("shopping").getCollection("orders");
BasicDBObject query = new BasicDBObject();
query.put("name", "lbroudoux");
DBCursor cursor = orders.find(query).slaveOk();
```

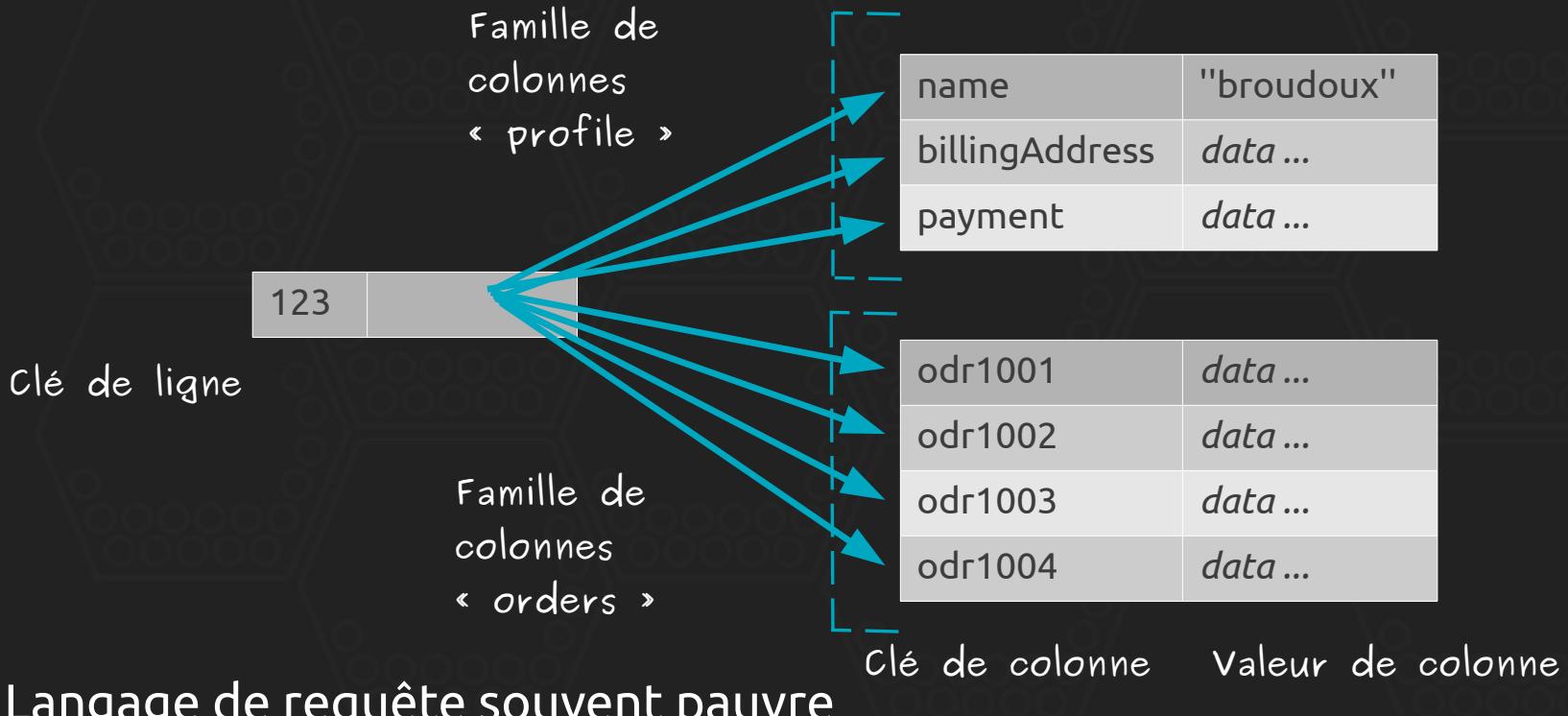
Requête : Query  
by example

# Bases *column-family* (i)



C'est quoi ?

- Données définies par une clé à laquelle peut correspondre plusieurs familles de colonnes étant elles même des maps de données



- Langage de requête souvent pauvre

# Bases *column-family* (ii)



A utiliser pour ...

- Données avec partie structurée et partie non structurée
  - Evénements des applicatifs (*sharding* possible par application)
- Données de publication variables
  - CMS, Blogging avec commentaires, contenu dynamique, etc ...
- Compteurs et analytiques
- Données avec TTL



A éviter pour ...

- Des besoins de requêtage complexes
- Des besoins de calcul d'agrégation simples (nécessité de passer systématiquement par Map-Reduce aujourd'hui)



# Bases *column-family* (iii)



Comment ?

```
Session session = cluster.connect();

session.execute("CREATE KEYSPACE simplex WITH replication " +
    "= {'class':'SimpleStrategy', 'replication_factor':3};");
session.execute(
    "CREATE TABLE simplex.songs (" +
        "id uuid PRIMARY KEY," +
        "title text," +
        "album text," +
        "artist text," +
        "tags set<text>," +
        "data blob" +
    ");");

session.execute(
    "INSERT INTO simplex.songs (id, title, album, artist, tags) " +
    "VALUES (" +
        "756716f7-2e54-4715-9f00-91dcbea6cf50," +
        "'La Petite Tonkinoise'," +
        "'Bye Bye Blackbird'," +
        "'Joséphine Baker'," +
        "{'jazz', '2013'}" +
    ");");
```

Création d'un schéma en CQL

# Bases *graph* (i)



C'est quoi ?

- Stockage sous forme d'entités (nodes) et d'associations (edges)
- Les nodes possèdent des propriétés (penser à un objet)
- Les edges possèdent un type (likes, author)
- Optimisées pour traverser le graph rapidement dans n'importe quel sens
  - « Quelles sont les personnes employées par X dont les amis aiment le film Y ? »
- Modèle de distribution constraint : souvent pas de *sharding* automatique, seulement réPLICATION
- Langages de requête « exotiques » : Gremlin, Cypher
- Parfois, complété par un moteur d'indexation

# Bases *graph* (ii)



A utiliser pour ...

- Les moteurs de recommandations
  - « Les autres clients ayant acheté ce produit ont aussi acheté ... »
- Les données naturellement connectées
  - Réseaux sociaux
- Les services basés sur la localisation ou le calcul d'itinéraires



A éviter pour ...

- Les cas où de nombreux nœuds doivent être mis à jour

# Bases *graph* (iii)



Comment ?

```
Node laurent = graphDB.createNode();
laurent.setProperty("name", "Laurent");
Node julien = graphDB.createNode();
julien.setProperty("name", "Julien");

laurent.createRelationshipTo(julien, COLLEAGUE);
julien.createRelationshipTo(laurent, COLLEAGUE);
laurent.createRelationshipTo(julien, FORMER_MANAGER);
```

Création de nœuds  
et de relations

```
Index<Node> nodeIndex = graphDB.index().forNodes("nodes");
Node node = nodeIndex.get("name", "Julien").getSingle();

allRelationships = node.getRelationships();    // 2
incomingRelations = node.getRelationships(Direction.INCOMING);    // 1

formerManagerRelations = node.getRelationships(Direction.BOTH, FORMER_MANAGER);    // 1
```

Recherche simple et parcours des relations

# Bases *graph* (iv)



Comment ?

```
Traverser colleaguesTraverser = node.traverse(  
    Order.BREADTH_FIRST,                                // En largeur, puis en profondeur  
    StopEvaluator.END_OF_GRAPH,                          // Arret a la toute fin du graph  
    ReturnableEvaluator.ALL_BUT_START_NODE,            // On exclu le noeud de depart du resultat  
    EdgeTypes.COLLEAGUE,                               // On suit les relations COLLEAGUE  
    Direction.OUTGOING);                            // Relations sortantes uniquement
```

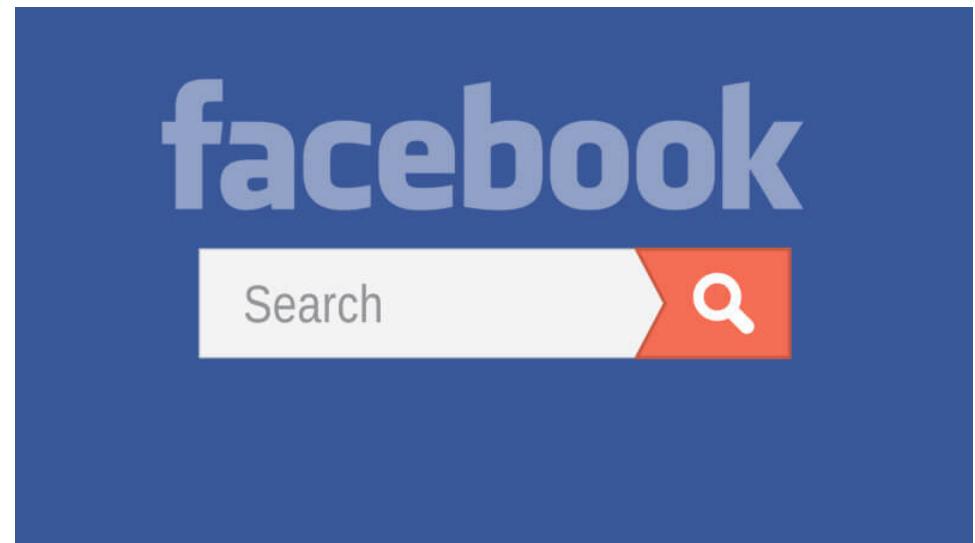
Recherche de tous les nœuds liés, transitivement

```
Node robert = nodeIndex.get("name", "Robert").getSingle();  
  
PathFinder<Path> finder = GraphAlgoFactory.allPaths(          // Aussi: shortestPath()  
    Traversal.expanderForTypes(COLLEAGUE, Direction.OUTGOING), // Specification de la traversée  
    MAX_DEPTH);                                              // Limite de recherche, nombre de hops  
Iterable<Path> paths = finder.findAllPaths(julien, robert);
```

Recherche des chemins possibles entre 2 nœuds



# Présentation du mouvement NoSQL : exemples concrets d'utilisation



## Customers Who Bought This Item Also Bought



[30 Rock: Seasons 1-3](#)  
DVD ~ Tracy Morgan  
 (7)  
\$60.49



[Desperate Housewives: The Complete Seasons 1-5](#)  
DVD ~ Teri Hatcher  
 (2)  
\$179.99



[Scrubs: The Complete Seasons 1-8](#)  
DVD ~ Zach Braff  
 (2)  
\$148.49

+

**MongoDB?**

# Introduction à MongoDB

- MongoDB se distingue des systèmes de gestion de base de données traditionnels par le type d'objet qu'il peut stocker et sa manière de les stocker. Ici pas de tables et de relations (comme des clés étrangères). Vous stockez des objets constitués de propriétés, dont les valeurs peuvent être d'autres objets.
- Pour installer MongoDB., vous pouvez utiliser le gestionnaire de paquets de votre système, ou bien télécharger le programme sur le site de MongoDB (<http://www.mongodb.org/>). Une fois installé, vous pouvez lancer le shell de MongoDB par la commande mongo.



# Installation de MongoDB

<https://www.mongodb.com/>

The screenshot shows the MongoDB download page with the 'Community Server' tab highlighted in green. The top navigation bar includes links for Atlas, Community Server, Enterprise Server, Ops Manager, Compass, and Connector for BI. Below the navigation is a green banner with the text 'Current Stable Release (3.4.10)'. To the right of the banner are links for 'Current Release', 'Previous Releases', and 'Development Releases'. Under the banner, there are download links for Windows, Linux, and OSX. A dropdown menu for the OSX version is open, showing 'OS X 10.7+ 64-bit w/SSL x64'. A note below the download links states: 'This version of MongoDB does not restrict network access by default. Please review the [security checklist](#) for instructions on how to prevent unauthorized access.' Further down, there's a section for 'Package Manager' with a link to 'Instructions for installing with Homebrew'. At the bottom, there's a 'Binary' section with a 'DOWNLOAD (tgz)' button and a download link: [https://fastdl.mongodb.org/osx/mongodb-osx-ssl-x86\\_64-3.4.10.tgz](https://fastdl.mongodb.org/osx/mongodb-osx-ssl-x86_64-3.4.10.tgz).

Atlas      Community Server      Enterprise Server      Ops Manager      Compass      Connector for BI

Current Stable Release (3.4.10)

10/31/2017: [Release Notes](#) | [Changelog](#)  
Download Source: [tgz](#) | [zip](#)

Current Release | Previous Releases | Development Releases

Windows      Linux      OSX

Version:  
OS X 10.7+ 64-bit w/SSL x64

This version of MongoDB does not restrict network access by default. Please review the [security checklist](#) for instructions on how to prevent unauthorized access.

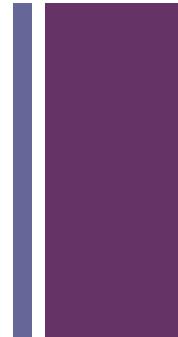
Package Manager:  
[Instructions for installing with Homebrew](#)

Binary: [Installation Instructions](#) | [All Version Binaries](#)

DOWNLOAD (tgz)      [https://fastdl.mongodb.org/osx/mongodb-osx-ssl-x86\\_64-3.4.10.tgz](https://fastdl.mongodb.org/osx/mongodb-osx-ssl-x86_64-3.4.10.tgz)



# Connexion à MongoDB



```
mongod --config /usr/local/etc/mongod.conf
```

```
mongo
```



# Les premières commandes à connaître : le B-A-BA.

`help`

`show dbs`

`show collections`

`use <db_name>`



## Les premières commandes à connaître : INSERT

```
db.restaurants.insert(  
  {  
    "name" : "Cookies Corner",  
    "address" : {  
      "street" : "1970 2nd St",  
      "zipcode" : 20001,}  
  }  
);
```



## Les premières commandes à connaître : FIND

```
db.restaurants.find({name: "Cookies Corner"})
```



## Les premières commandes à connaître : REMOVE

```
db.restaurants.remove({});
```

+

## **Travailler avec les documents**

# JSON (*Javascript Object Notation*)

# JSON (2002)

- Un format **textuel** et léger de représentation de données **structurée**
  - **Lisible** par l'homme
  - Généré et analysé **trivialement** par la machine
- **Inspiré** largement par le **javascript** (sous-ensemble de ECMA-262) tout en restant indépendant du langage : un sous-ensemble minimal & portable de javascript

# JSON (RFC 4627 en 2006)

- Beaucoup de langages supportés : objective C, C, C++, java ect...
- Beaucoup de langues : français, anglais, chinois, japonnais...
- Il n'existe qu'une seule version de JSON : aucun révision prévue

# Usage

- JSON est utilisé
  - En tant que format d'échange entre un client (typiquement un navigateur, possiblement embarqué) et un serveur (backend),
  - Non pas en tant que format de représentation de données
  - Par google, yahoo (services Web)
  - Dans le domaines des smartphones du fait de sa légéreté...

# JSON – Exemple

```
{ "menu": {  
    "id": "file",  
    "value": "File",  
    "popup": {  
        "menuitem": [  
            { "value": "New", "onclick":  
"CreateNewDoc ()"},  
            { "value": "Open", "onclick": "OpenDoc ()"},  
            { "value": "Close", "onclick":  
"CloseDoc ()"}  
        ] } } }
```

# Format – 2 types structurés

- JSON s'organise autours de **deux structures** (ou types structurés)
  - Object : une collection (non ordonnée) de paires (nom-valeur)
  - Array : un séquence (ordonnée) de valeurs
- Utilisées par n'importe quel langage moderne, ces 2 types permettent de représenter des structures, des tables (par exemple hash tables) ect...
- L'imbrication est permise mais aucune structure récursive n'est supportée

# Format – array

- Une séquence ordonnée de 0 à n **valeurs** encerclée par [ ]
  - `array = [ value * ( , value ) ]`
- Est utilisée lorsque les noms (qui sont uniques) sont des entiers séquentiels
- Exemple : [ 30 ,40, 50 ]

# Format - object

- Une collection non ordonnées de 0 à n paires
  - object = [ member \* ( , pair ) ]
  - Chaque paire est composée d'un **nom** et d'une **valeur**
    - pair = string : value
    - nomDeLaPaire1 : valeurDeLaPaire1 ,  
nomDelaPaire2 : valeurDeLaPaire2
- Est souvent utilisé lorsque les noms sont des strings
- Exemple : {"size": 50}

# Format - Exemple d'objects imbriqués

```
{ "Image": {  
    "Width": 800,  
    "Height": 600,  
    "Title": "View from 15th",  
    "Thumbnail": {  
        "Url": "http://ex.fr/img/48",  
        "Height": 125,  
        "Width": "100"},  
    "IDs": [116, 943, 234, 38793] } }
```

# Format - Exemple de tableau contenant 2 objets

```
[ { "Latitude": 37.7668,  
  "Longitude": -122.3959,  
  "Address": "",  
  "City": "SAN FRANCISCO",  
  "Country": "US" },  
  { "Latitude": 37.371991,  
  "Longitude": -122.026020,  
  "City": "SUNNYVALE",  
  "Country": "US" } ]10
```

# Format – types primitifs

- Aux 2 types structurés (array, object) s'ajoutent 4 types primitifs

string	Séquence de 0 à n caractères unicodes suivant les conventions du langage de programmation C, entouré par ""
number	Nombre décimale signé
boolean	true ou false
null	null

- Une valeur est de type structuré ou primitive

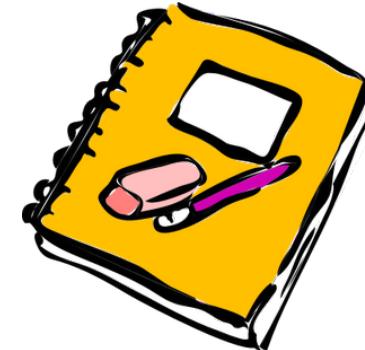
# Générateur & Parseur

- Le **générateur** doit créer un **texte conforme** à la grammaire JSON
- Le **parseur** transforme un texte en une autre représentation. Il analyse tout texte conforme à la grammaire JSON et possiblement des **extensions**, des formats non conformes
- Les limites (longueur, profondeur du texte, des nombres, des strings et de leur contenus) sont fixées au niveau de l'implémentation

# Conclusion

- JSON est utilisé en tant que format léger d'échange entre un client et un serveur
- JSON constitue une alternative à XML notamment lorsque les clients (smartphones) sont contraints
  - Est géré par les navigateurs actuels
  - Est intégré par les plateformes des services Web
- De multiples implémentations / languages (Unicode) sont disponibles

# Introduction à MongoDB : Exercice de modélisation



Ligue nationale de hockey (LNH) vous a missionné pour réaliser leur système de base de données avec les exigences suivantes :

- La LNH a de nombreuses équipes,
- Chaque équipe a un nom, une ville, un entraîneur, un capitaine, et un ensemble de joueurs,
- Chaque joueur appartient à une seule équipe,
- Chaque joueur a un nom, une position (comme l'aile gauche ou le gardien de but), un niveau de compétence et un ensemble de dossiers de blessures,
- Un capitaine d'équipe est également un joueur,
- Un jeu est joué entre deux équipes (appelé host\_team et guest\_team) et a une date (comme le 11 mai 1999) et une note (par exemple 4 à 2).

**Proposer un/des schémas JSON de documents pour modéliser ces bases de données dans MongoDB**



# Introduction to MongoDB

# What is MongoDB ?

- Scalable High-Performance Open-source, Document-orientated database.
- Built for Speed
- Rich Document based queries for Easy readability.
- Full Index Support for High Performance.
- Replication and Failover for High Availability.
- Auto Sharding for Easy Scalability.
- Map / Reduce for Aggregation.

# Why use MongoDB?

- SQL was invented in the 70's to store **data**.
- MongoDB stores **documents (or) objects**.
- Now-a-days, everyone works with **objects** (Python/Ruby/Java/etc.)
- And we need Databases to persist our **objects**. Then why not store **objects** directly ?
- Embedded documents and arrays reduce need for joins. **No Joins** and No-multi document **transactions**.



# What is MongoDB great for?

- RDBMS replacement for **Web Applications**.
- **Semi-structured Content Management**.
- **Real-time Analytics & High-Speed Logging**.
- Caching and **High Scalability**



## Not great for?

- Highly **Transactional** Applications.
- Problems requiring **SQL**.

# Impedance Mismatch

```
// your application code  
class Foo { int x; string [] tags;}
```

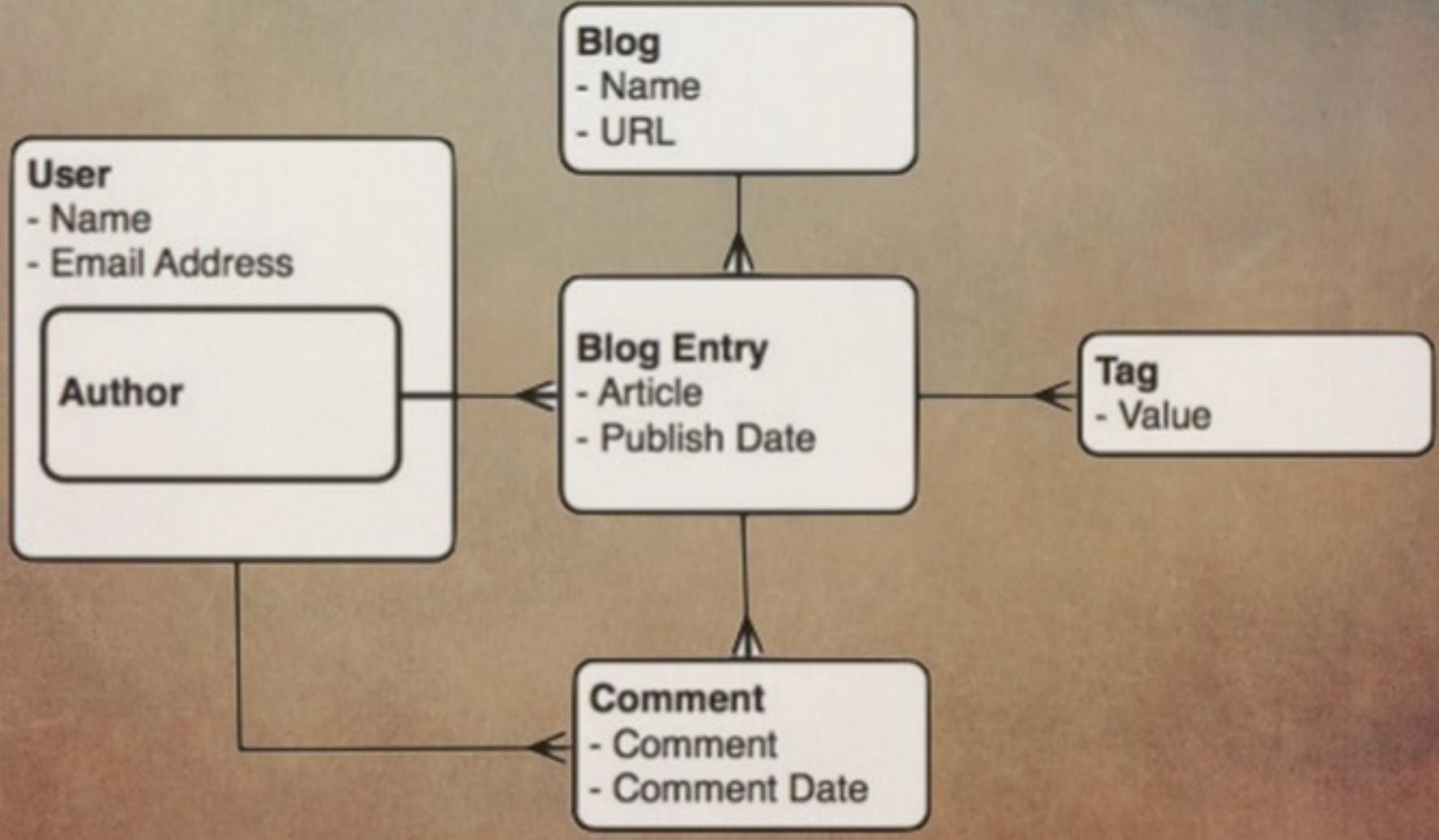
x	name	tagId	x
1	Abc	33	1
2	Xyz	34	2
		33	2
tagId	tag		
33	red		
34	blue		

## No Impedance Mismatch

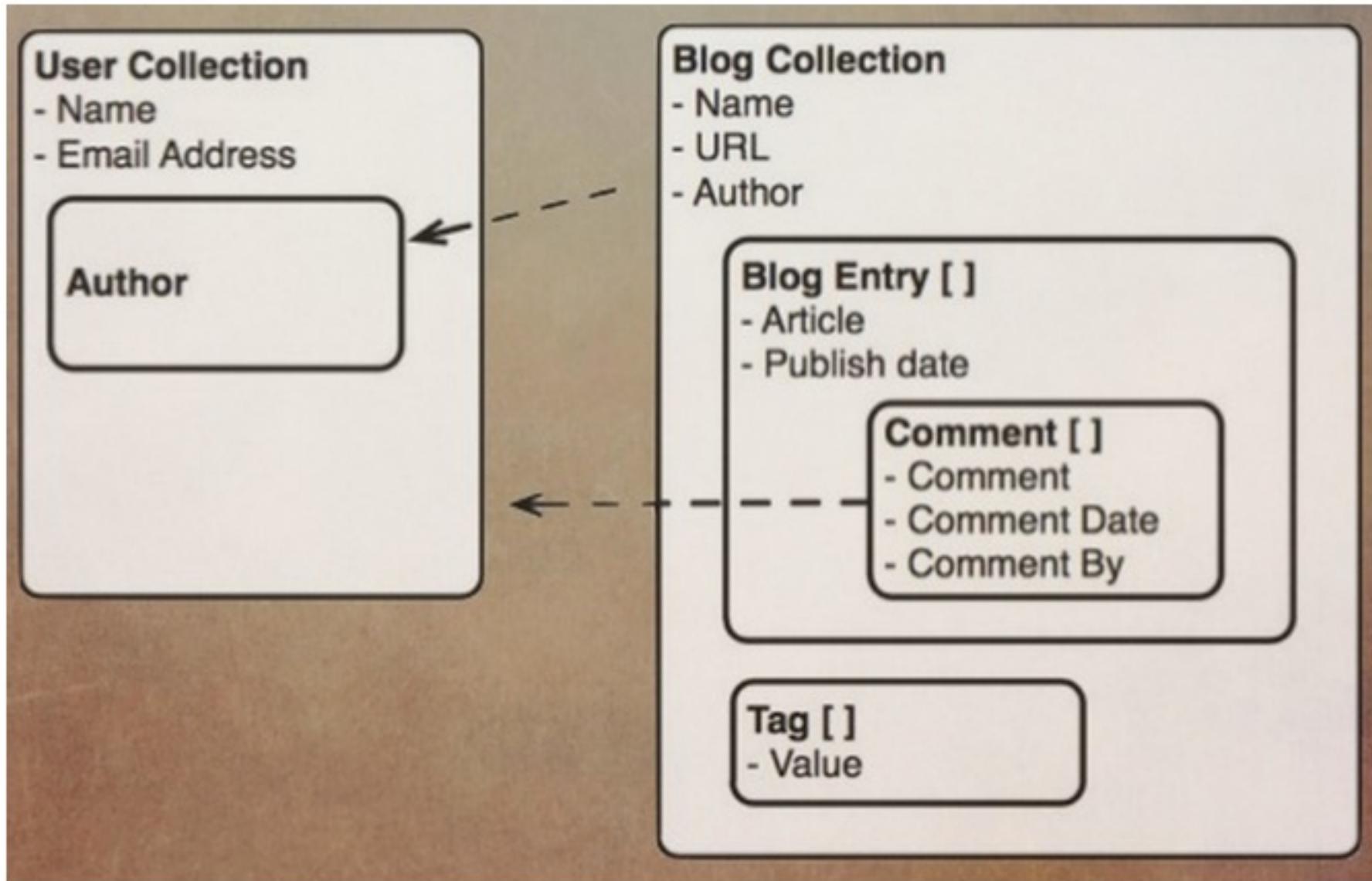
```
// your application code
class Foo { int x; string [] tags; }

// mongo document for Foo
{ x: 1, tags: ['abc','xyz'] }
```

# Blog in relational DB



# Blog post structure in document DB



## Blog post in JSON DB

```
{ _id : ObjectId("4c4ba5c0672c685e5e8aabf3"),
  author : "steve",
  date : "Sat Apr 24 2013 19:47:11",
  text : "About MongoDB...",
  tags : [ "tech", "databases" ],
  comments : [
    {
      author : "Fred",
      date : "Sat Apr 25 2013 20:51:03 GMT-0700",
      text : "Best Post Ever!"
    }
  ]
}
```

When I say  
**Database**



Think  
**Database**

- Made up of Multiple **Collections**.
- Created **on-the-fly** when referenced for the first time.

When I say  
**Collection**



Think  
**Table**

- Schema-less, and contains **Documents**.
- **Indexable** by one/more keys.
- Created **on-the-fly** when referenced for the first time.
- **Capped Collections:** Fixed size, older records get dropped after reaching the limit.

When I say  
**Document**



Think  
**Record/Row**

- Stored in a **Collection**.
- Have **\_id** key – works like Primary keys in MySQL.
- Supported Relationships – **Embedded (or) References**.
- Document storage in **BSON** (Binary form of JSON).

## Understanding the Document Model

```
var post = {  
    '_id': ObjectId('3432'),  
    'author': ObjectId('2311'),  
    'title': 'Introduction to MongoDB',  
    'body': 'MongoDB is an open sources.. ',  
    'timestamp': Date('01-04-12'),  
    'tags': ['MongoDB', 'NoSQL'],  
    'comments': [{  
        'author': ObjectId('5331'),  
        'date': Date('02-04-12'),  
        'text': 'Did you see.. ',  
        'upvotes': 7} ]  
}  
  
> db.posts.insert(post);
```



# The Problem

- You say:

```
db.foo.find({ x: 10 })
```

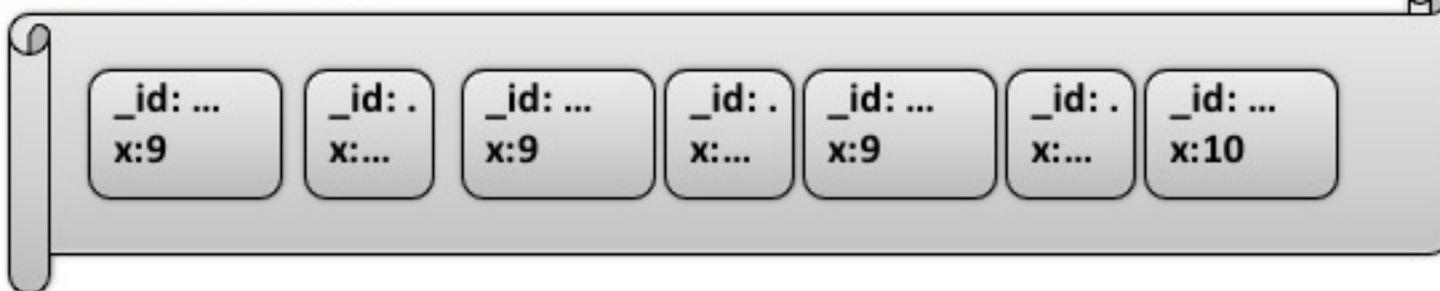
Ouch!

Reads EVERY document!

- The server does :(pseudo)

```
for each doc d in 'foo' {  
    if ( d.x == 10 ){  
        return d  
    }  
}
```

## Document Storage



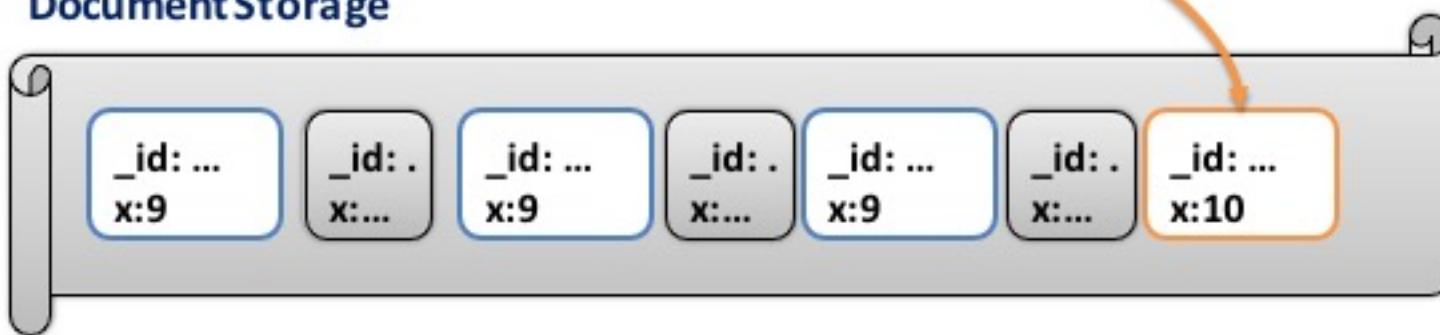
# The Solution

Index - field 'x' , collection 'foo'

Value	Doc Pointers
9	[171, 819, 2309]
10	[4376]

`db.foo.find({ x:10 })`

## Document Storage



## Create Index

```
db.foo.ensureIndex(keys, options)
```

Collection

Which fields?  
In what Order?  
Geo / Text

Name?  
Build now?  
Unique  
Sparse?  
TTL?  
Language?

## Secondary Indexes

Create Index on any field in the document

// 1 means ascending, -1 means descending

```
> db.posts.ensureIndex({'author': 1});
```

//Index Nested Documents

```
> db.posts.ensureIndex('comments.author': 1);
```

// Index on tags

```
> db.posts.ensureIndex({'tags': 1});
```

// Geo-spatial Index

```
> db.posts.ensureIndex({'author.location': '2d'});
```



## Find

```
// find posts which has 'MongoDB' tag.
```

```
> db.posts.find({tags: 'MongoDB'});
```

```
// find posts by author's comments.
```

```
> db.posts.find({'comments.author': 'Johnson'}).count();
```

```
// find posts written after 31st March.
```

```
> db.posts.find({'timestamp': {'$gte': Date('31-03-12')}});
```

```
// find posts written by authors around [22, 42]
```

```
> db.posts.find({'author.location': {'$near':[22, 42]}});
```

\$gt, \$lt, \$gte, \$lte, \$ne, \$all, \$in, \$nin...



# Find

```
db.foo.find(query, projection)
```

Which fields?

Which documents?

## Find: projection

```
> db.posts.find({}, {title:1})
```

```
{ "_id" : ObjectId("5654381f37f63ffc4ebf1964"),
  "title" : "NodeJS server" }
{ "_id" : ObjectId("5654385c37f63ffc4ebf1965"),
  "title" : "Introduction to MongoDB" }
```

Like

```
select title from posts
```

Empty projection like

```
select * from posts
```

# Find

## Find

- Query criteria
  - Single value field
  - Array field
  - Sub-document /dot notation

## Projection

- Filed inclusion and exclusion

## Cursor

- Sort
- Limit
- Skip

## Paging example

```
place1 = {  
    name : "10gen HQ",  
    address : "229 W 43rd St. 5th Floor",  
    city : "New York",  
    zip : "10036",  
    tags : [ "business", "awesome" ]  
}  
> db.places.insert(place1)  
per_page = 10;  
page_num = 3;  
  
places = db.places  
    .find({ "city" : "new york" })  
    .sort({ "ts" : -1 })  
    .skip((page_num - 1) * per_page)  
    .limit(per_page);
```

## Update: replace the document

```
> db.posts.update(  
  {"_id" : ObjectId("5654381f37f63ffc4ebf1964")},  
  {  
    title:"NodeJS server"  
});
```

This will **replace** the document by {title:"NodeJS server"}

## Update: change only the part of document

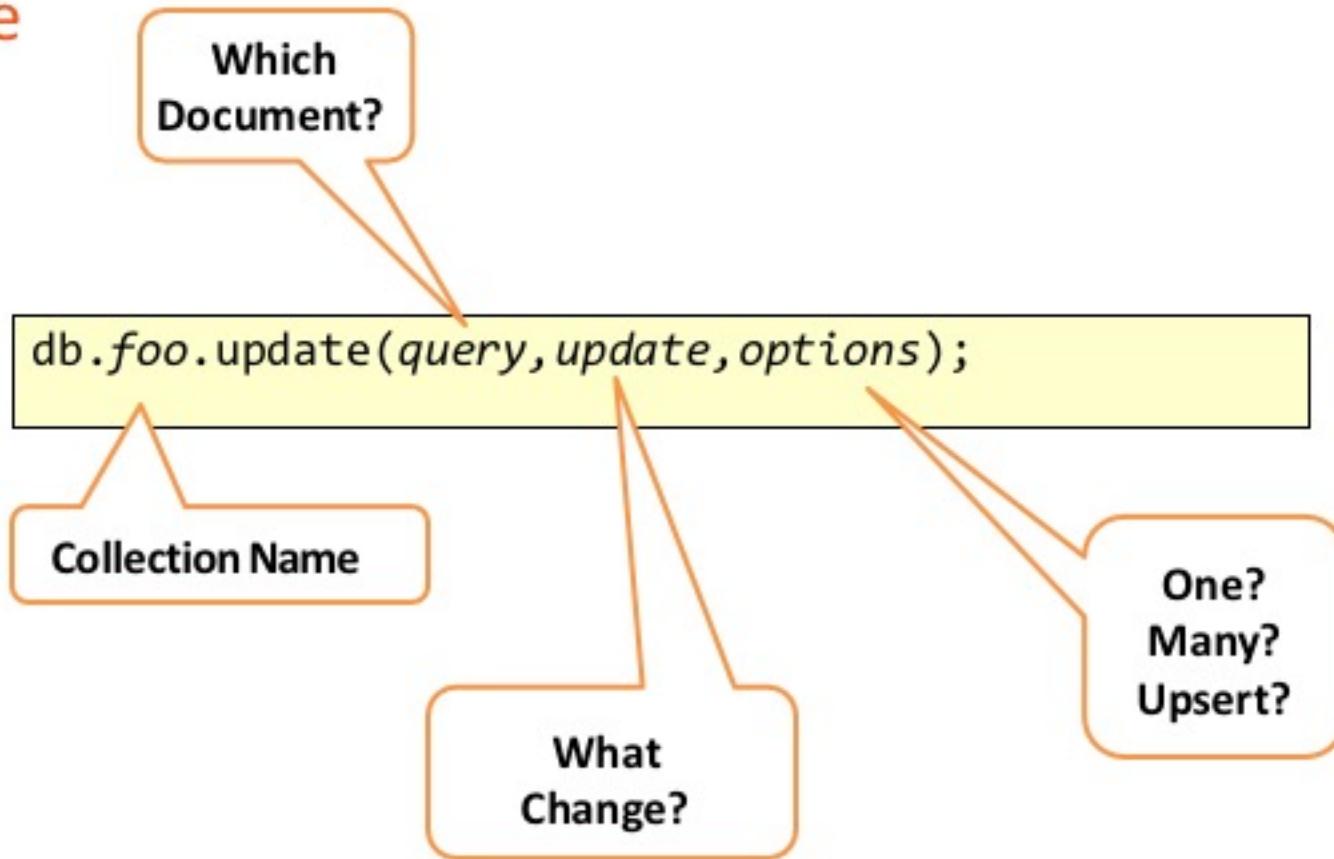
```
> db.posts.update(  
  {"_id" : ObjectId("5654381f37f63ffc4ebf1964")},  
  {  
    $addToSet: {tags:"JS"},  
    $set: {title:"NodeJS server"},  
    $unset: { comments: 1}  
  });
```

\$set, \$unset

\$push, \$pull, \$pop, \$addToSet

\$inc, \$decr, many more...

## Update



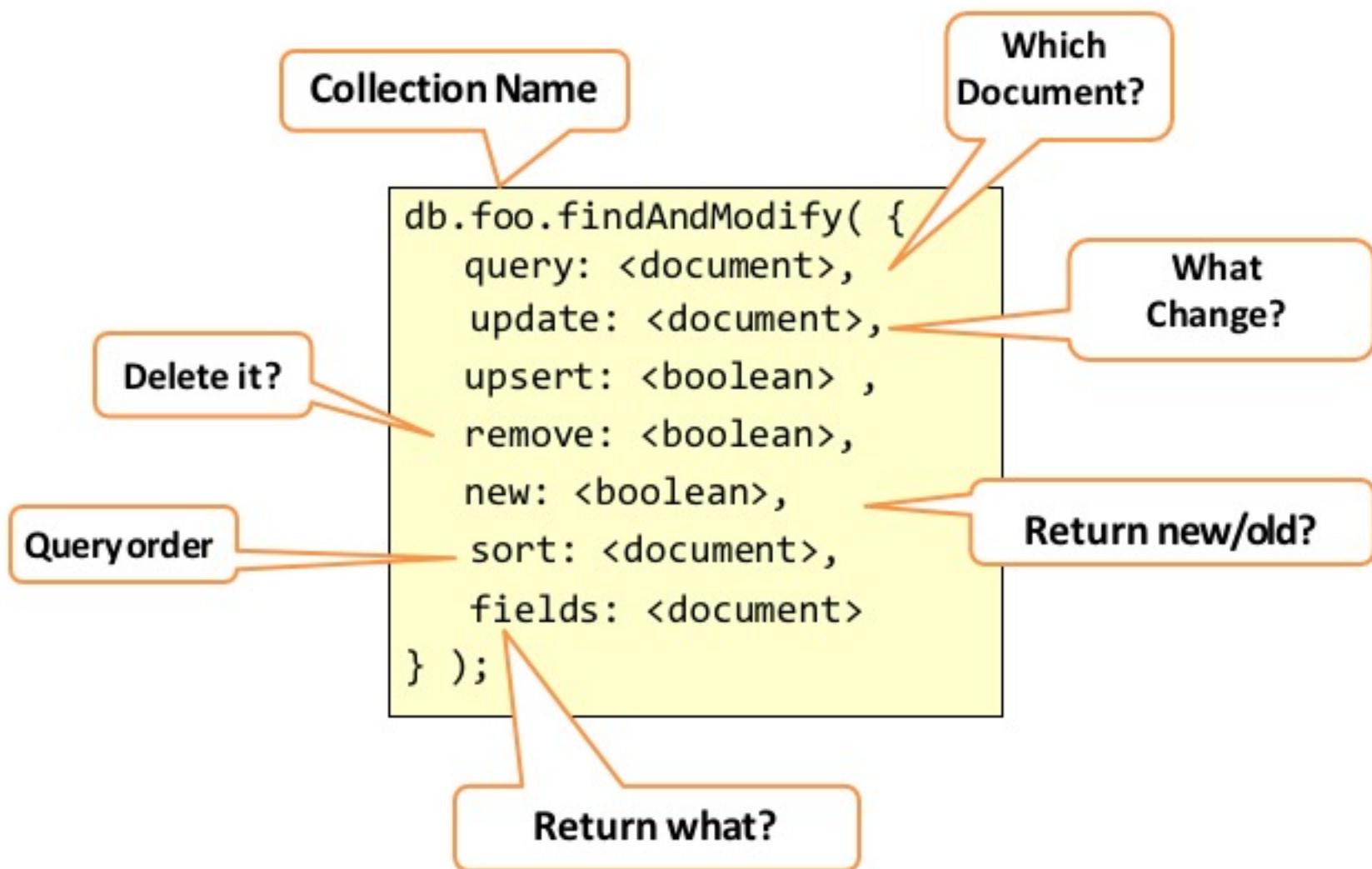
### Options:

**{multi: true}** – will change all found documents;

by default only first found will be updated

**{upsert: true}** – will insert document if it was not found

## Find And Modify

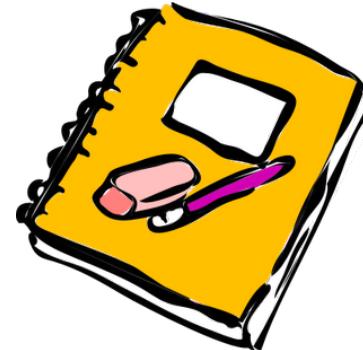


## Some Cool features

- Geo-spatial Indexes for Geo-spatial queries.  
\$near, \$within\_distance, Bound queries (circle, box)
- GridFS  
Stores Large Binary Files.
- Map/Reduce  
GROUP BY in SQL, map/reduce in MongoDB.



# Introduction à MongoDB : Exercice de modélisation

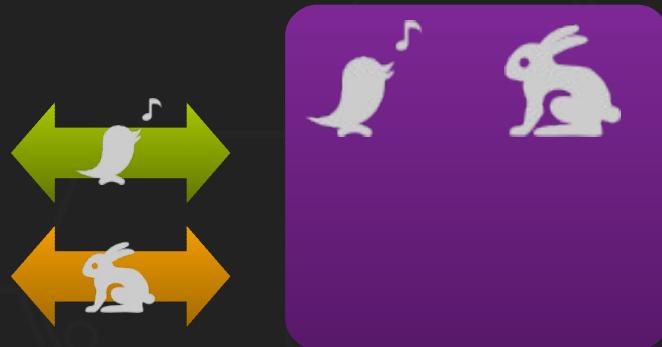


Ecrire les commandes MongoDB pour insérer les schémas JSON de l'exercice et insérer des “dummy data” pour tester la cohérence de nos schémas

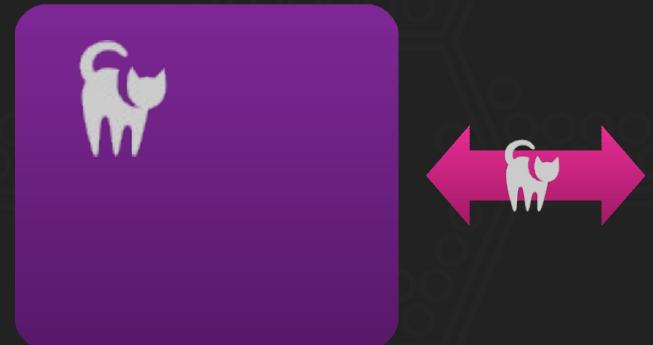
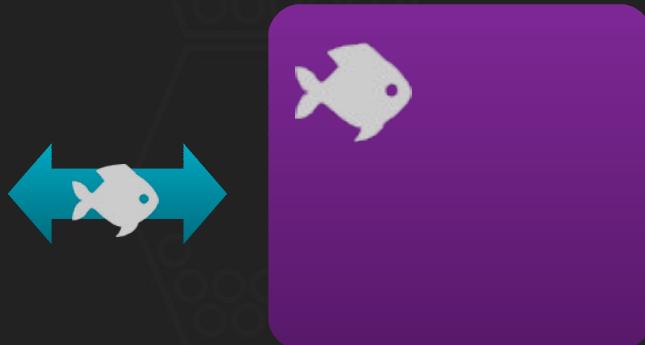
+

## **Sharding sous MongoDB**

# Sharding (i)



Chaque shard lit et écrit  
ses propres données



# Sharding (ii)



## Bénéfices

- Performance
  - En écriture car *scaling horizontal*
  - En lecture lorsque géolocalisation des données
- Espace disque des machines
- Résilience partielle voire localisée



## Préoccupations

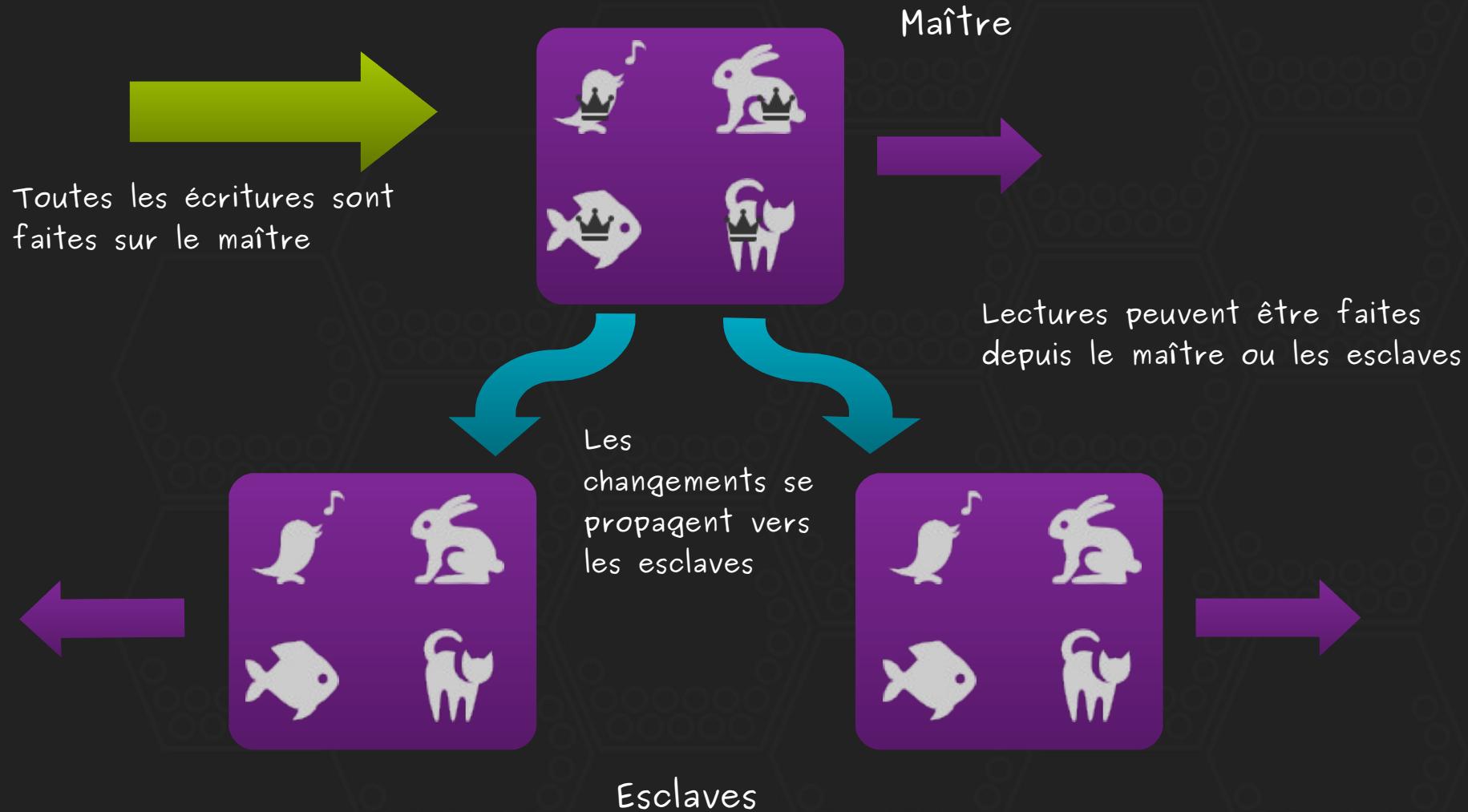
- Répartition équitable des données
- Localisation des accès communs - d'où les agrégats ;-)
- *Sharding as application logic* ?!

+

## RéPLICATION

# sous MongoDB

# RéPLICATION MAÎTRE / ESCLAVE (i)



# RéPLICATION MAÎTRE / ESCLAVE (ii)



## Bénéfices

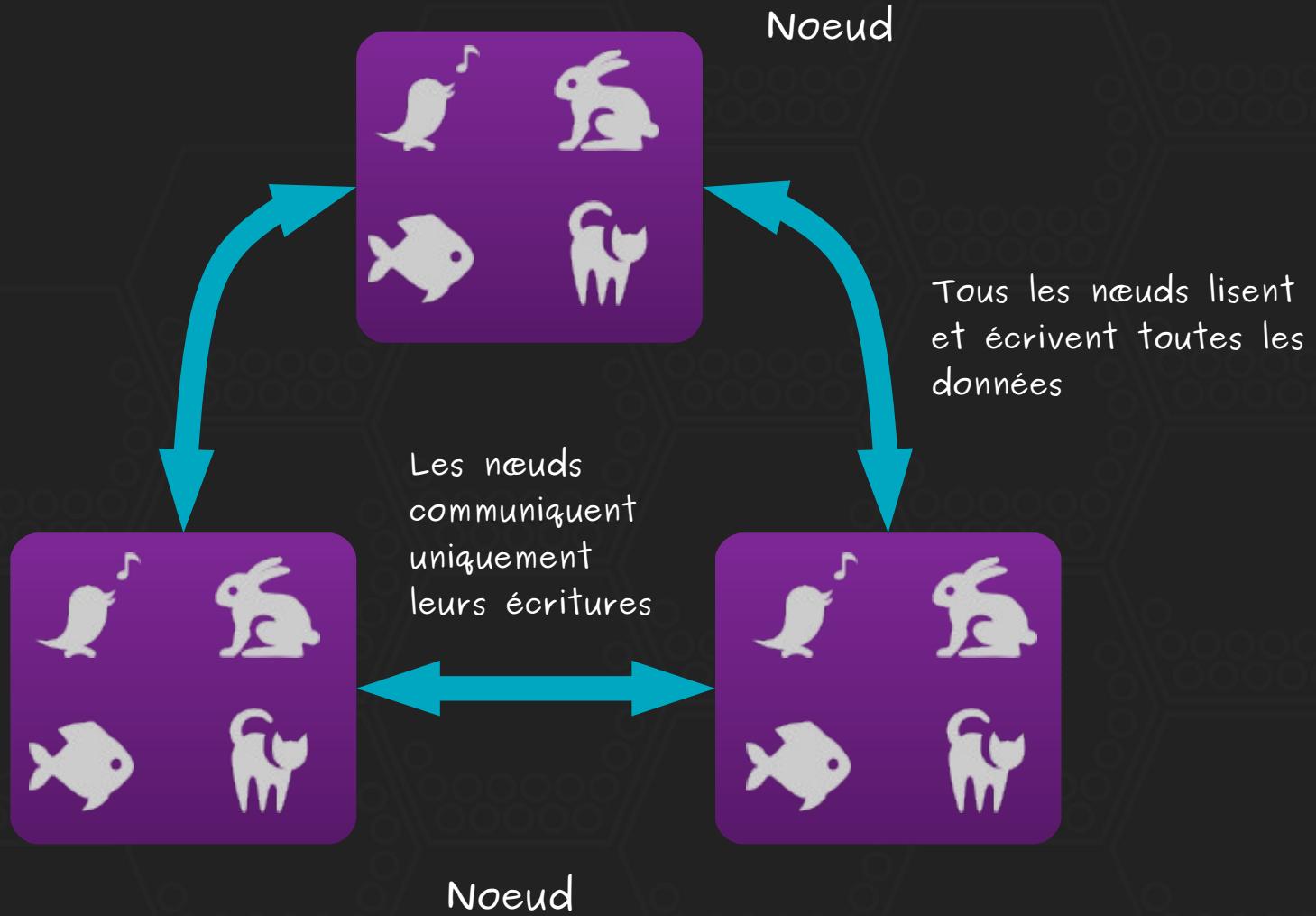
- Pour les applications avec beaucoup de lectures
  - Performance
  - Résilience
- Elasticité par le provisioning de nouveaux esclaves



## Préoccupations

- Inconsistance possible en lecture (*local read-write* à gérer par le driver)
- Résilience pour l'écriture fonction de la capacité de changement de rôle (maître est un *SPOF*)
- Algorithme de vote automatique et *split brain* !

# RéPLICATION Peer to Peer (i)



# RéPLICATION Peer to Peer (ii)



## Bénéfices

- Résilience en lecture comme en écriture
- Elasticité par le provisioning transparent
- Performances des lectures



## Préoccupations

- Inconsistances car écritures simultanées possibles
- Performances des écritures
  - Voir les quorums et *versions vectors* ...
- Volume de données à synchroniser et sens de synchro !

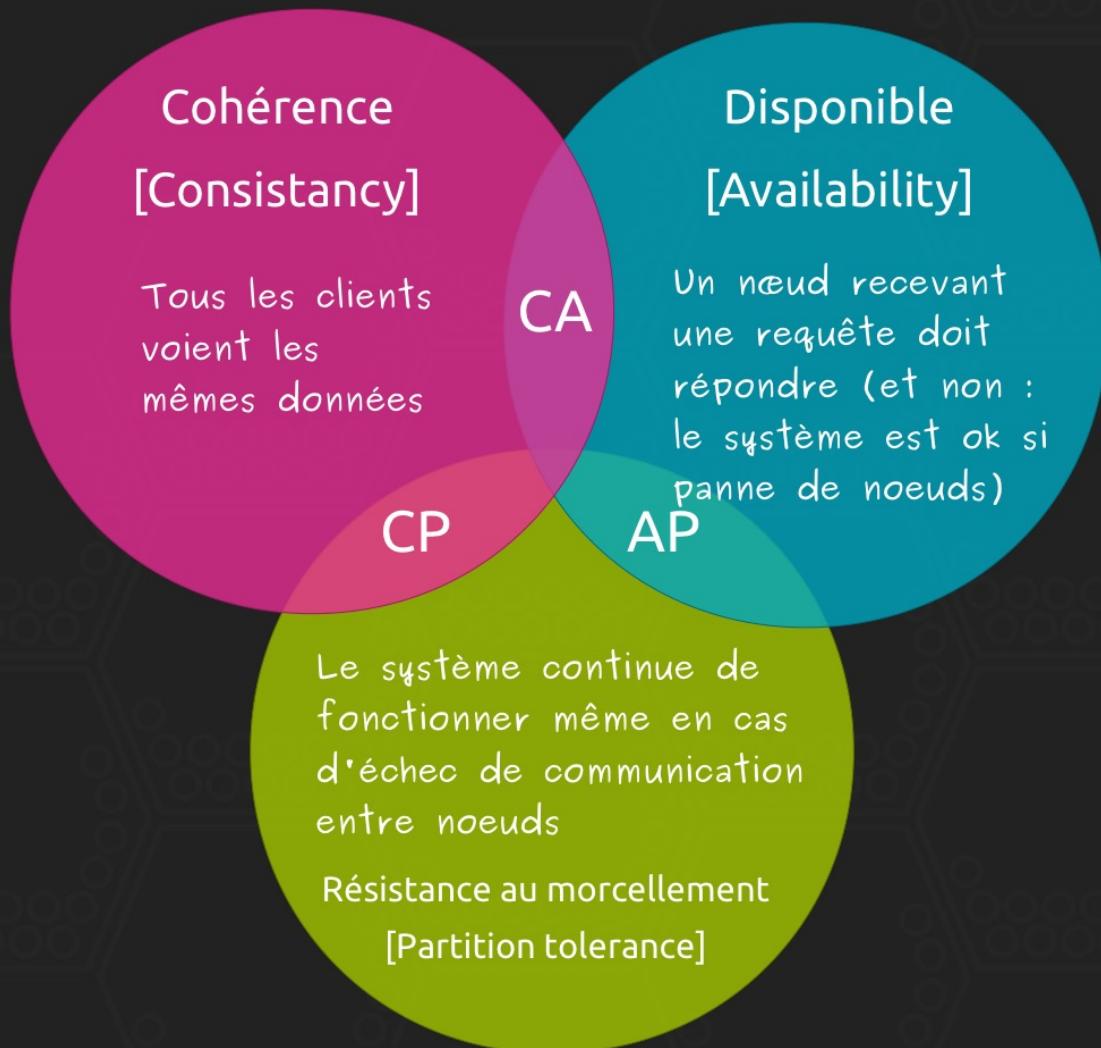
# Théorème CAP

« Il est impossible pour un système distribué de satisfaire plus de 2 des 3 propriétés suivantes : Cohérence, Disponibilité et Résistance au morcellement »

- Eric Brewer, 2000



# Théorème CAP



+

Mise en pratique des ReplicaSet et du  
Sharding

# Travaux Pratiques

(cf. MongoDB-ReplicaSet-Sharding.pdf)



+

## **Logiciels d'administration avec GUI**



# Mac OS X & Windows : Logiciel Robot 3T (RoboMongo)

The screenshot shows the RoboMongo 0.8.4 application window. On the left, the sidebar displays the database structure: 'MongoHQ FloodData (1)' containing 'FloodData' (with 'Collections (4)' including 'System', 'floodtweetsarchive', 'Indexes', and 'Functions (0)'), and 'Users (2)'. The main area has two tabs: 'floodtweetsarchive' and 'Logs'. The 'floodtweetsarchive' tab shows a query in the top panel:

```
db.floodtweetsarchive.find( { location: "London" } )  
db.floodtweetsarchive.find( { followers_count: { $gt: 100 } } )
```

The results of the query are displayed below, showing a single document with the following fields:

```
/* 0 */  
{  
    "topic" : "tweets/TrevorDolan",  
    "payload" : "RT @SurreySAR: @Volunteer_SFVS: In 5 days we have used 2100 rashers,1800 sausages",  
    "location" : "London",  
    "lang" : "en",  
    "sentiment" : {  
        "score" : 0,  
        "comparative" : 0,  
        "tokens" : [  
            "..."  
        ]  
    }  
}
```

The bottom panel shows a table with 9 rows, each representing an ObjectId and its corresponding 11-field document. The columns are 'Key' (ObjectId), 'Value' (document), and 'Type' (Object). The table is as follows:

Key	Value	Type
► (1) ObjectId("530201228...")	{ 11 fields }	Object
► (2) ObjectId("530201418...")	{ 11 fields }	Object
► (3) ObjectId("530201b58...")	{ 11 fields }	Object
► (4) ObjectId("530201b58...")	{ 11 fields }	Object
► (5) ObjectId("530202468...")	{ 11 fields }	Object
► (6) ObjectId("5302025f8...")	{ 11 fields }	Object
► (7) ObjectId("530202878...")	{ 11 fields }	Object
► (8) ObjectId("530202b78...")	{ 11 fields }	Object
► (9) ObjectId("530202b78...")	{ 11 fields }	Object

+

## **Gestion des utilisateurs sous MongoDB**



# Créer un admin de la base MongoDB

```
use admin

db.createUser(
    {
        user:"myadmin",
        pwd:"secret",
        roles:[{role:"root",db:"admin"}]
    }
)

exit
```

```
mongo -u myadmin -p --authenticationDatabase admin
```



# Créer un utilisateur sur une base de données précise

```
use mydb
```

```
db.createUser(  
  {  
    user: "mydbuser",  
    pwd: "mydbsecret",  
    roles: ["readWrite"]  
  }  
)
```

```
exit
```

```
db.auth('mydbuser','mydbsecret')  
db.getUsers()
```



## Supprimer un utilisateur

```
use mydb
db.dropUser('mydbuser')
```

+

**Sauvegarder & importer des  
données dans MongoDB**



## Les commandes à connaître

```
mongodump --out /data/backup/
```

```
mongorestore --port <port number> <path to the backup>
```

+

**Gérer la journalisation et les  
statistiques sur les collections**



# Activer/Gérer/Désactiver les journaux

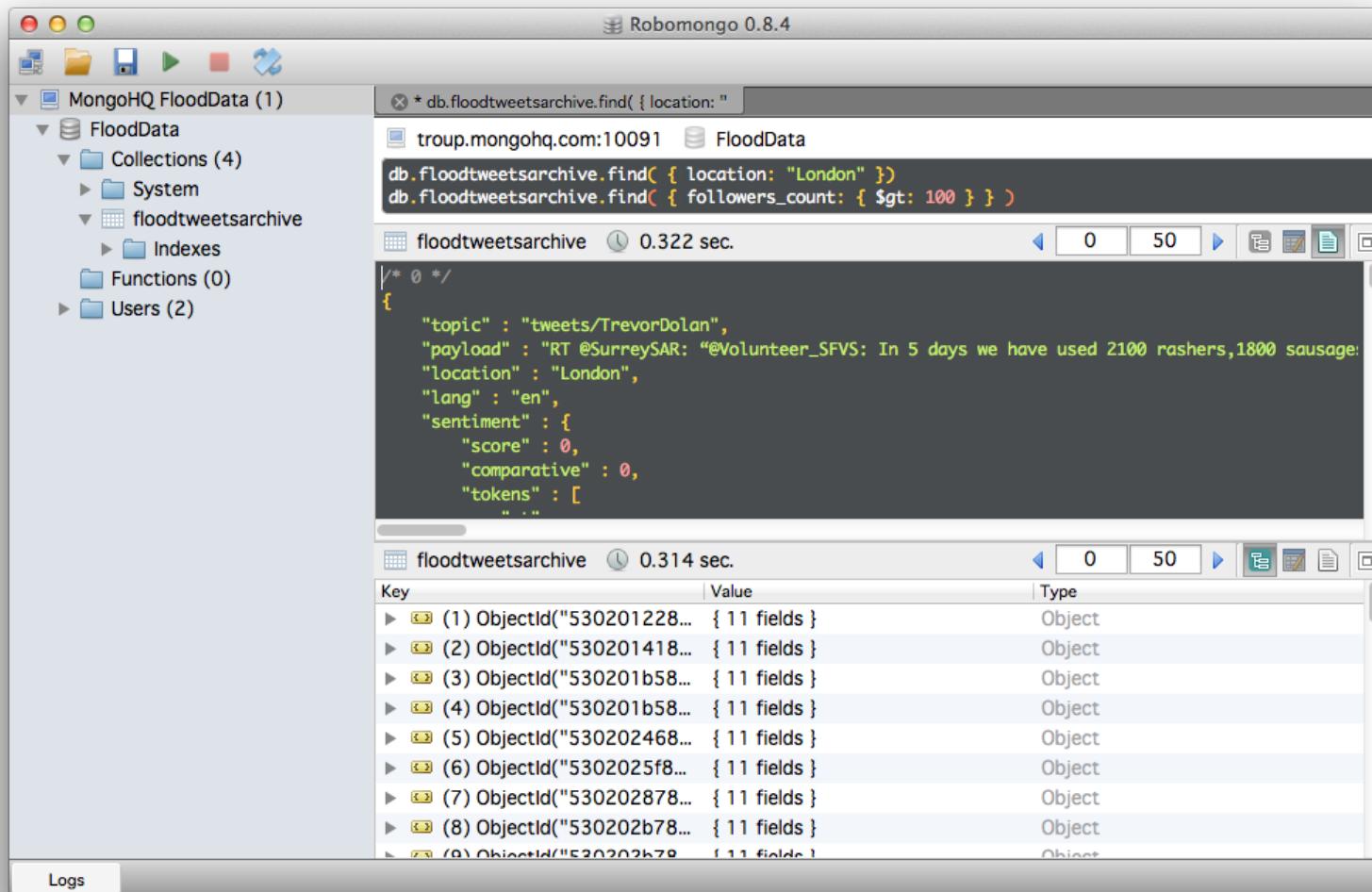
`/etc/mongod.conf`

```
# Disables write-ahead journaling  
#nojournal = true  
smallfiles = true
```

```
sudo service mongodb stop  
sudo service mongodb start
```

```
sudo rm /var/lib/mongodb/journal/prealloc.*
```

# Robot 3T (RoboMongo)



# mongostat

L'utilitaire mongostat fournit un aperçu rapide de l'état d'une instance de mongod ou mongos en cours d'exécution.

```
→ ~ mongostat
connected to: 127.0.0.1
insert query update delete getmore command flushes mapped vsize res faults locked db idx miss % qrlqw arlsw netIn netOut conn time
 *0   *0   *0   *0   0   1|0   0   80m 2.63g 33m 0 local:0.0% 0 0|0 0|0 62b 2k 2 09:56:07
 *0   *0   *0   *0   0   1|0   0   80m 2.63g 33m 0 local:0.0% 0 0|0 0|0 62b 2k 2 09:56:08
 *0   1   101   *0   1   1|0   0   10g 22.5g 37m 577 .:0.2% 0 0|0 0|0 57k 51k 2 09:56:09
 *0   *0   1089   *0   0   1|0   0   10g 22.5g 54m 3021 .:0.2% r_development:10.6% 0 0|0 0|0 1m 4m 2 09:56:10
 *0   *0   4472   *0   0   1|0   0   10g 22.5g 62m 1638 r_development:42.5% 0 0|0 0|1 3m 2k 2 09:56:11
 *0   *0   4289   *0   1   1|0   0   10g 22.5g 77m 2661 r_development:36.4% 0 0|0 0|0 1m 4m 2 09:56:12
 *0   *0   5007   *0   0   1|0   0   10g 22.5g 68m 40 r_development:42.0% 0 0|0 0|1 2m 2k 2 09:56:13
 *0   *0   4613   *0   1   1|0   0   10g 22.5g 83m 2328 r_development:38.5% 0 0|0 0|1 1m 4m 2 09:56:14
 *0   *0   5058   *0   0   1|0   0   10g 22.5g 74m 40 r_development:42.2% 0 0|0 0|1 2m 2k 2 09:56:15
 *0   *0   4974   *0   0   1|0   0   10g 22.5g 82m 40 r_development:41.4% 0 0|0 0|0 2m 2k 2 09:56:16
insert query update delete getmore command flushes mapped vsize res faults locked db idx miss % qrlqw arlsw netIn netOut conn time
 *0   *0   4703   *0   1   1|0   0   10g 22.5g 81m 2322 r_development:39.5% 0 0|0 0|0 1m 4m 2 09:56:17
 *0   *0   4983   *0   0   1|0   0   10g 22.5g 88m 38 r_development:41.2% 0 0|0 0|1 2m 2k 2 09:56:18
 *0   *0   4608   *0   1   1|0   0   10g 22.5g 87m 2322 r_development:39.2% 0 0|0 0|0 1m 4m 2 09:56:19
 *0   2   5067   *0   0   1|0   1   10g 22.5g 94m 40 r_development:43.2% 0 0|0 0|0 2m 2k 2 09:56:20
 *0   *0   4917   *0   0   1|0   0   10g 22.5g 102m 38 r_development:41.2% 0 0|0 0|0 2m 2k 2 09:56:21
 *0   *0   4584   *0   1   1|0   0   10g 22.5g 108m 2264 r_development:38.8% 0 0|0 0|1 1m 4m 2 09:56:22
 *0   *0   4478   *0   0   1|0   0   10g 22.5g 112m 44 r_development:37.6% 0 0|0 0|0 2m 2k 2 09:56:23
```



# mongotop

mongotop fournit une méthode pour suivre le temps que met une instance MongoDB pour la lecture et l'écriture des données.

## Mongotop

ns	total	read	write	
test.employee	4894ms	0ms	4894ms	
admin.system.roles	0ms	0ms	0ms	
admin.system.version	0ms	0ms	0ms	
bookmarks.counters	0ms	0ms	0ms	
bookmarks.links	0ms	0ms	0ms	
bookmarks.products	0ms	0ms	0ms	
bookmarks.system.indexes	0ms	0ms	0ms	
bookmarks.system.namespaces	0ms	0ms	0ms	
bookmarks.users	0ms	0ms	0ms	
demodb.system.indexes	0ms	0ms	0ms	

# MongoDB Nagios Plugin

(<https://github.com/mzupan/nagios-plugin-mongodb>)





## MongoDB Atlas

- Atlas prend en charge les tâches opérationnelles, telles que le provisioning, la configuration, les mises à jour, les sauvegardes ou la remise en service après incident. Mais également le monitoring !





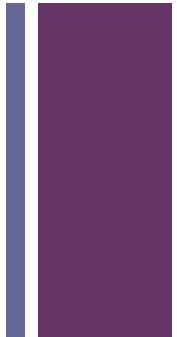
# MongoDB Cloud Manager

Une plateforme hébergée pour gérer MongoDB sur l'infrastructure de votre choix. Cloud Manager vous fait gagner du temps et de l'argent et vous aide à protéger votre expérience client en éliminant les approximations liées à l'exécution de MongoDB.



+

## D'autres besoins? D'autres outils?



<https://docs.mongodb.com/manual/administration/monitoring/>



Des questions? Je reste joignable sur [nzeka@nachosmedia.com](mailto:nzeka@nachosmedia.com) !