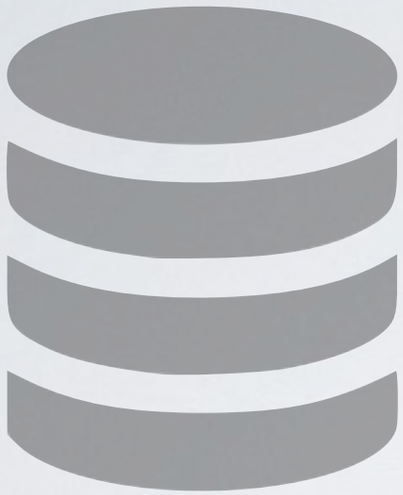
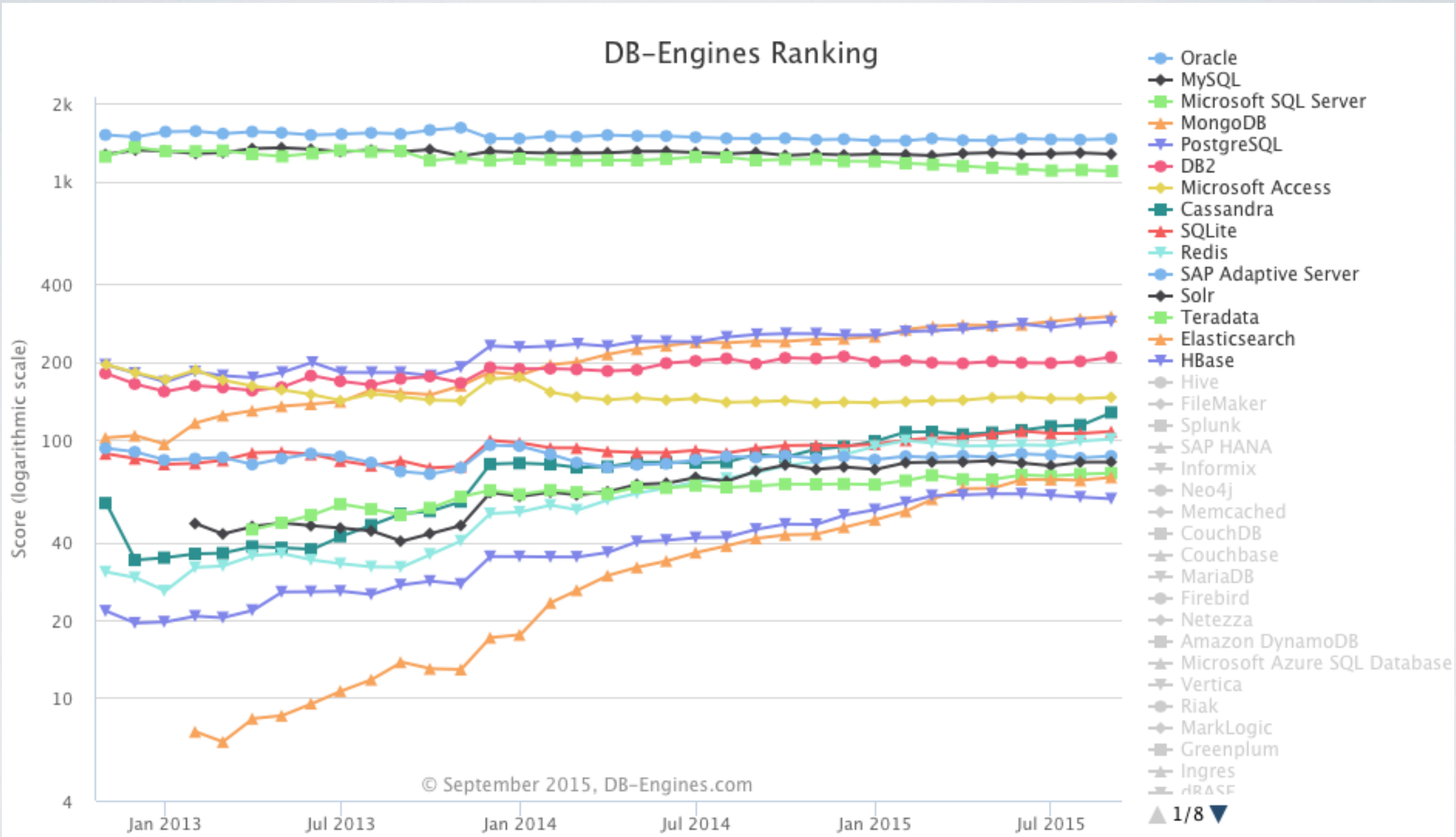


BASES DE DONNÉES

RELATIONNELLES & NOSQL

Cédric Villa - @zestprod - 18/09/2015

HISTOIRE


DB-Engines Ranking
Score (logarithmic scale)
2k
1k
400
200
100
40
20
10
4
Jan 2013
Jul 2013
Jan 2014
Jul 2014
Jan 2015
Jul 2015
© September 2015, DB-Engines.com
Oracle
MySQL
Microsoft SQL Server
MongoDB
PostgreSQL
DB2
Microsoft Access
Cassandra
SQLite
Redis
SAP Adaptive Server
Solr
Teradata
Elasticsearch
HBase
Hive
FileMaker
Splunk
SAP HANA
Informix
Neo4j
Memcached
CouchDB
Couchbase
MariaDB
Firebird
Netezza
Amazon DynamoDB
Microsoft Azure SQL Database
Vertica
Riak
MarkLogic
Greenplum
Ingres
1/8

DIFFERENCES STRUCTURELLES

BASES DE DONNÉES RELATIONNELLES

SCHÉMA
RELATIONNEL

Fortement normalisé

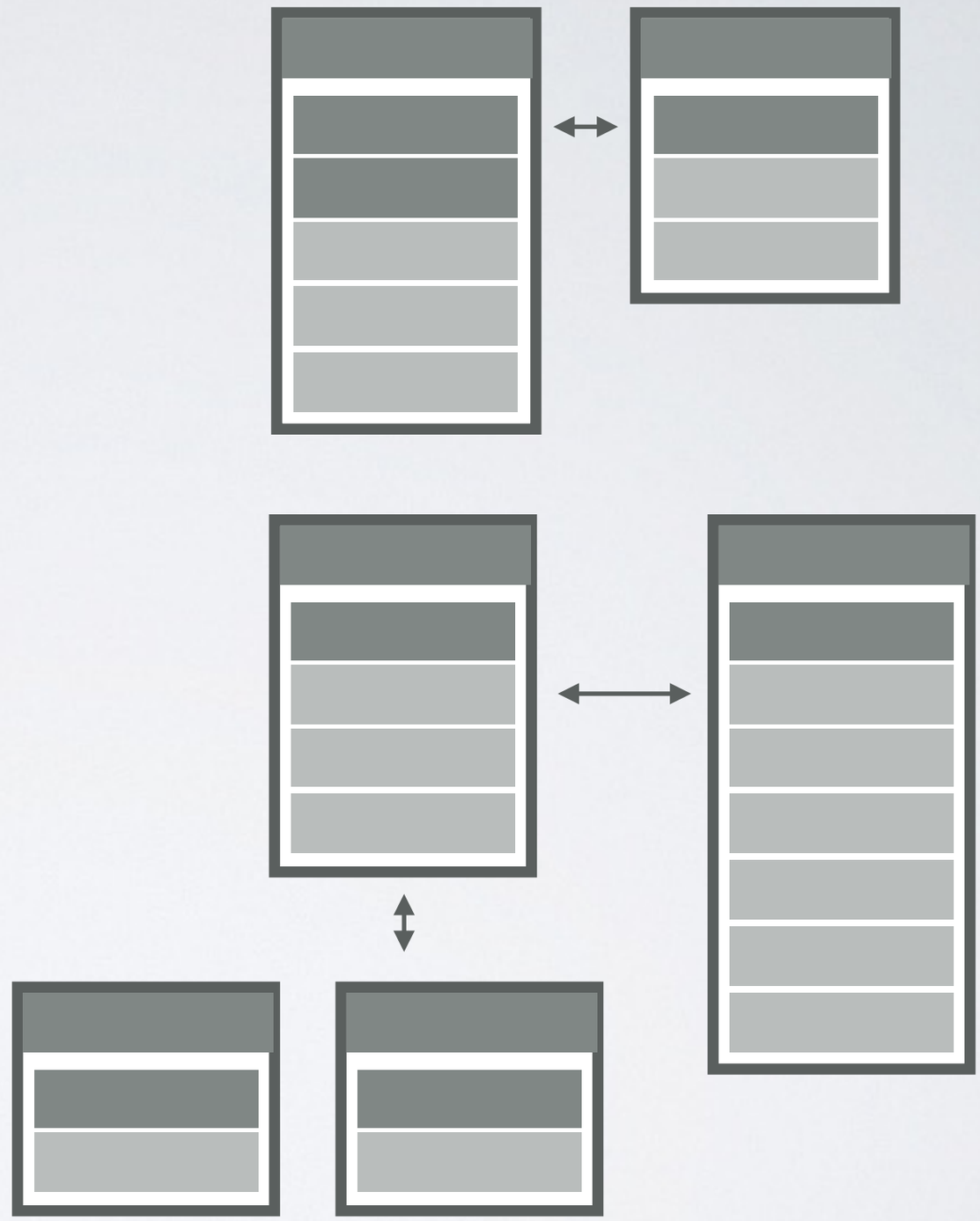


SCHÉMA
RELATIONNEL

Extraction précise

Métaphore
bibliothécaire pointilleux

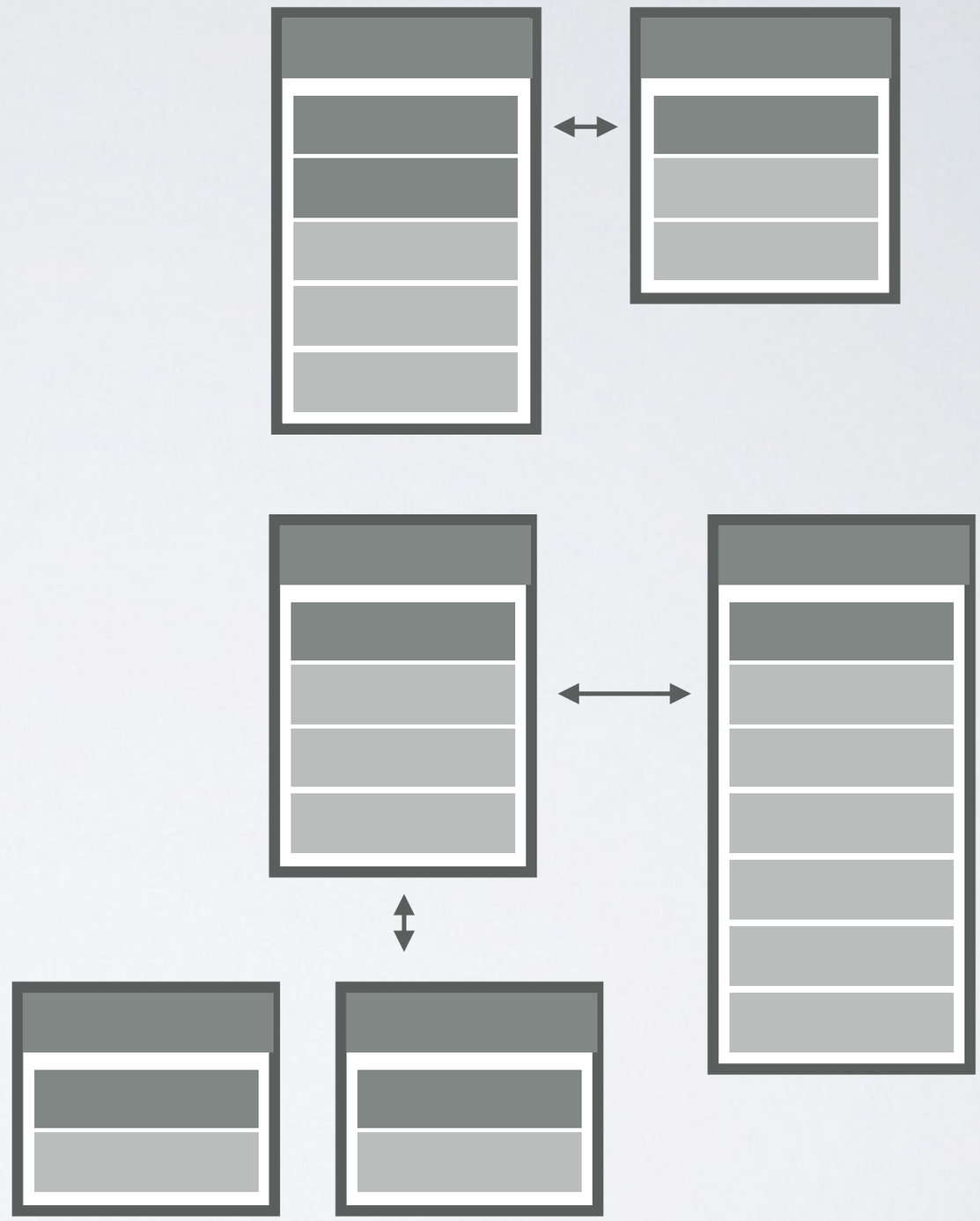
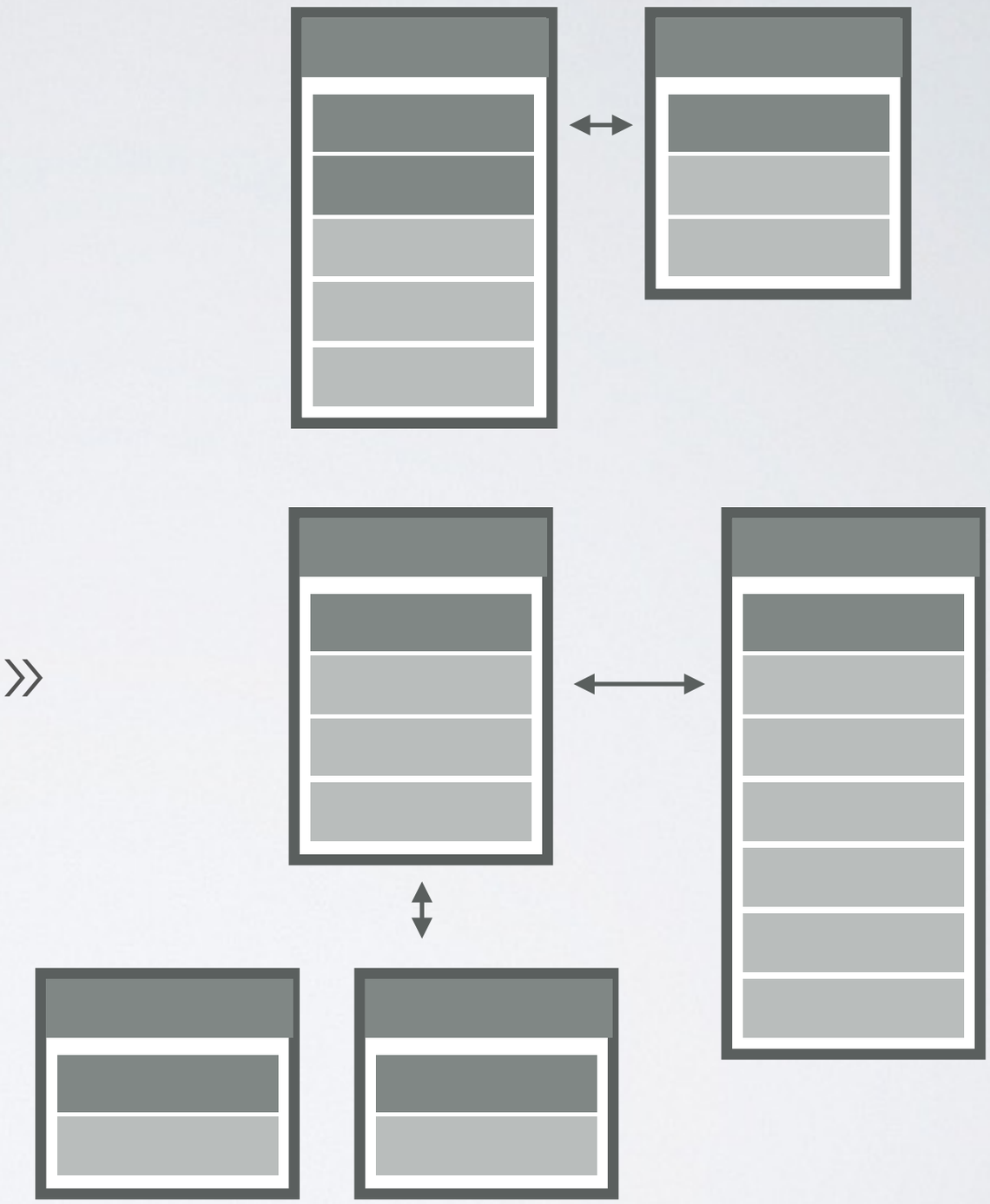


SCHÉMA
RELATIONNEL

« Documented by design »



SQL

```
$requete = "SELECT glpi_computers.id AS id,
            glpi_plugin_customfields_computers.*,
            glpi_states.name AS etat,
            main_user.email AS user_email,
            resp_tech_user.email AS resp_tech_email,
            glpi_computers.contact AS contact,
            glpi_operatingsystems.name AS os,
            glpi_locations.name AS location,
            glpi_groups.name AS group_name,
            glpi_infocoms.buy_date AS date_achat,
            glpi_plugin_order_orders_items.delivery_date AS date_livraison,
            glpi_infocoms.warranty_duration AS duree_garantie,
            glpi_computermodels.name AS model

            FROM glpi_plugin_customfields_computers

            LEFT OUTER JOIN glpi_computers ON glpi_plugin_customfields_computers.id = glpi_computers.id
            LEFT OUTER JOIN glpi_useremails AS main_user ON main_user.users_id = glpi_computers.users_id
            LEFT OUTER JOIN glpi_useremails AS resp_tech_user ON resp_tech_user.users_id = glpi_computers.users_id_tech
            LEFT OUTER JOIN glpi_groups ON glpi_groups.id = glpi_computers.groups_id
            LEFT OUTER JOIN glpi_states ON glpi_states.id = glpi_computers.states_id
            LEFT OUTER JOIN glpi_operatingsystems ON glpi_computers.operatingsystems_id = glpi_operatingsystems.id
            LEFT OUTER JOIN glpi_locations ON glpi_computers.locations_id = glpi_locations.id
            LEFT OUTER JOIN glpi_infocoms ON glpi_infocoms.items_id = glpi_computers.id AND glpi_infocoms.itemtype = 'co
            LEFT OUTER JOIN glpi_plugin_order_orders_items ON glpi_plugin_order_orders_items.items_id = glpi_computers.i
            LEFT OUTER JOIN glpi_computermodels ON glpi_computermodels.id = glpi_computers.computermodels_id

            WHERE (glpi_states.name LIKE '%actif%' OR glpi_states.name IS NULL)
                AND glpi_computers.id IS NOT NULL
                AND host NOT LIKE ''
                AND glpi_computers.is_deleted = '0'
                AND glpi_plugin_customfields_computers.hors_reseau = 1;";
```

ACID

Atomicité
Cohérence
Isolation
Durabilité

NOSQL

NOT ONLY SQL

DOCUMENT BASED DATABASE

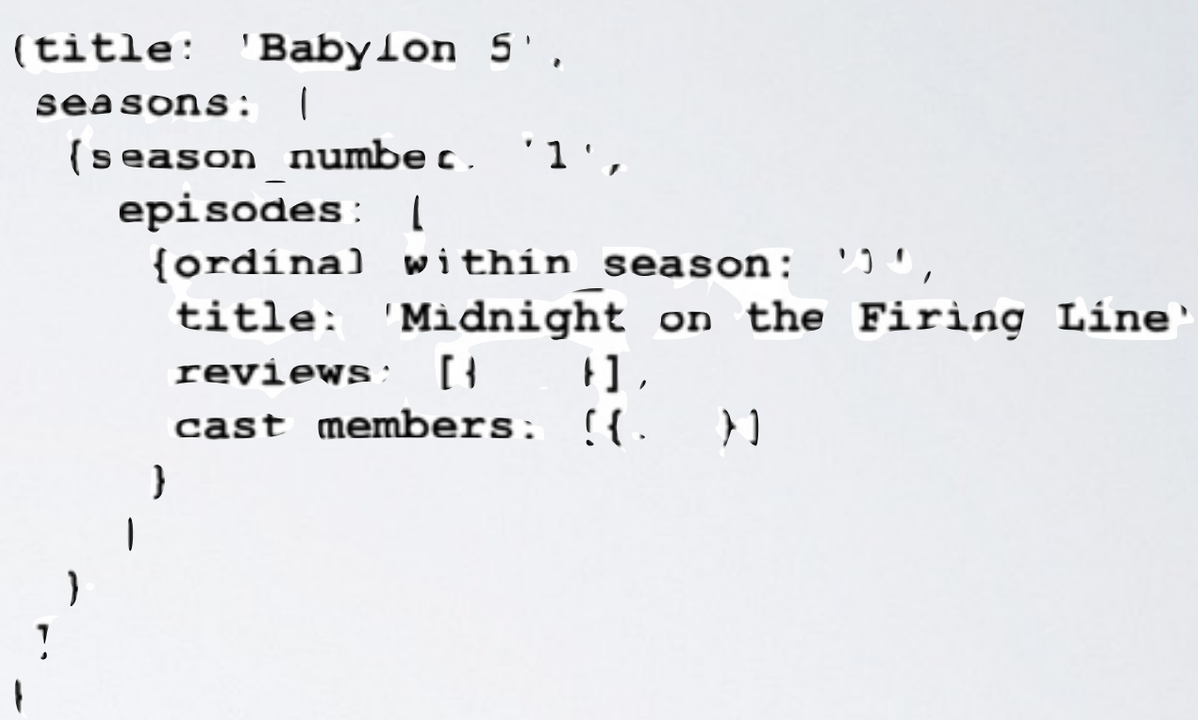
Structure imbriquée

DOCUMENT BASED DATABASE

Structure imbriquée

Web 2.0 JSON (REST)

Ergonomie



DOCUMENT BASED DATABASE

Structure imbriquée

DOCUMENT BASED DATABASE

Structure imbriquée

Pas de relations
dans le SGBD

Pas ACID

DOCUMENT BASED DATABASE

Système de cache

Construire et retourner
un format adapté
(ex. elastic search)

Plus difficile à débogguer

SCHEMA LESS DATABASES

BDD sans schéma

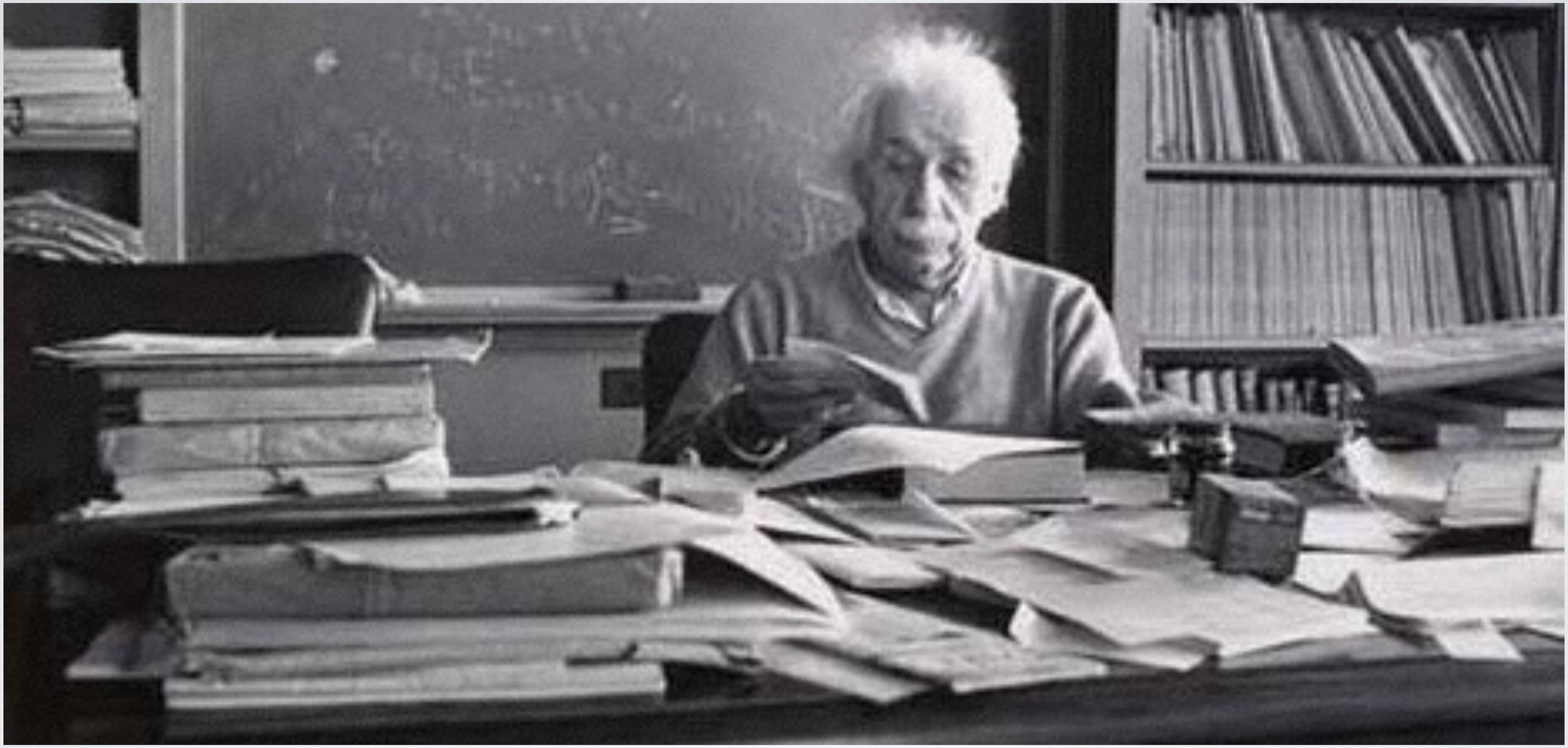
SCHEMALESS +

- Agilité (liberté lors du développement)

- Productivité

SCHEMALESS -

- Extraction de l'information
  - Métaphore du tiroir de bureau
- Difficulté de migration



MODÈLES CLÉS / VALEURS

Redis, (virtuellement n'importe quel NoSQL), etc.

```
SET foo bar
GET foo => bar
```

HYBRIDES

Ex: Postgres et le type « hstore »

BDD CÔTÉ CLIENT

HTML5 - Web SQL Database + Web Storage

DÉVELOPPEMENT ET AGILITÉ

DEVELOPPER AVEC UNE BDD

- Abondance de doc, frameworks, etc.

- Tous les langages

OBJECT RELATIONAL MAPPING

Mapping objet relationnel

MAPPING OBJET

11 lines (7 sloc) | 169 Bytes

```
1  require 'mongoid'
2  Mongoid.load!("mongoid.yml")
3
4  class DataSource
5
6    include Mongoid::Document
7    field :title,       type: String
8    field :description, type: String
9
10 end
```

MAPPING OBJET

- On manipule des objets
  - Object.create(attribute: « value », etc.)
  - Objet.find(name: « john »)
- Les requêtes sont générées dans le language du SGDB

MAPPING OBJET

- + Documentation: Dans le modèle
- + On peut ignorer le SGDB derrière
  (différences entre développement et production)

MAPPING OBJET

- NoSQL
  Idem que pour BDD relationnelles

- Avantages

  - **++** Documentation: dans le modèle

  - **+** Encore moins de gestion de la BDD

MAPPING OBJET

- Semble simple

- Ce n’est pas « fool-proof »
  - boucle for + modification en base

PERFORMANCES

PERFORMANCES

- Dépendent:
  - De la RAM
  - Des entrées/sorties du stockage (I/O)
  - Du CPU

AVANT D'ENVISAGER UN PLUS GROS SERVEUR

- Mise en cache
- D'une page complète (ou 304)
- D'un fragment HTML
- Du résultat d'une requête

BASES DE DONNÉES **RAPIDES**

Redis
Memcached

BDD **RAPIDES**

redis

- En RAM
- Pas d'écriture disque (systématique)
- Cluster

●●●

BDD **RAPIDES**

- Mise en cache
  - Données extraites
  - Fragments HTML
- Sessions
- Files d'attentes

BDD **RAPIDES**

- Pas de persistence / ACID

  sinon:
  pas mieux que BDD
  relationnelle

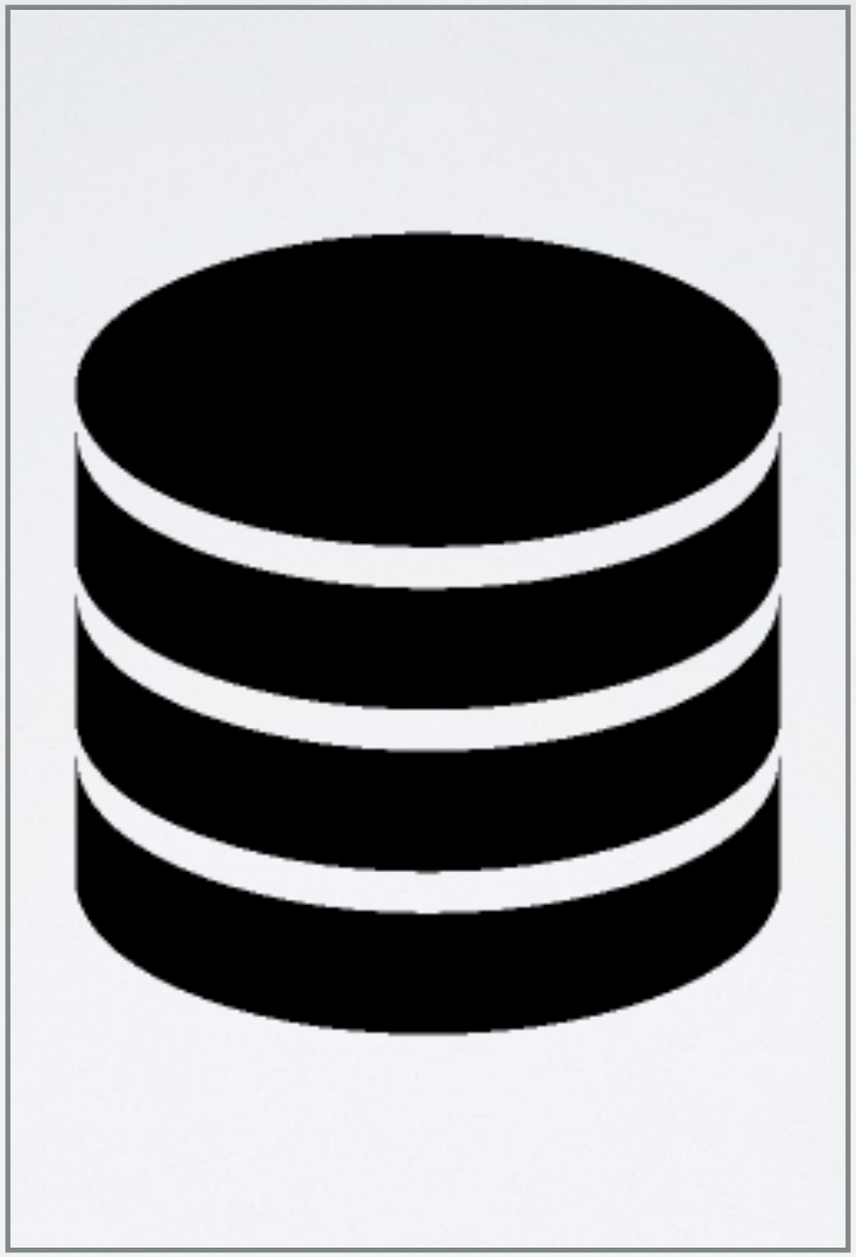
SCALING
(MISE À L'ÉCHELLE)

VERTICAL SCALING

VERTICAL SCALING

4 Go RAM
2 CPU
1 HDD 5400 RPM

VERTICAL SCALING



16 Go RAM
8 CPU
1 SSD (x100)

SI GOOGLE FAISAIT ÇA

- Connait **30 trillions** de **documents**

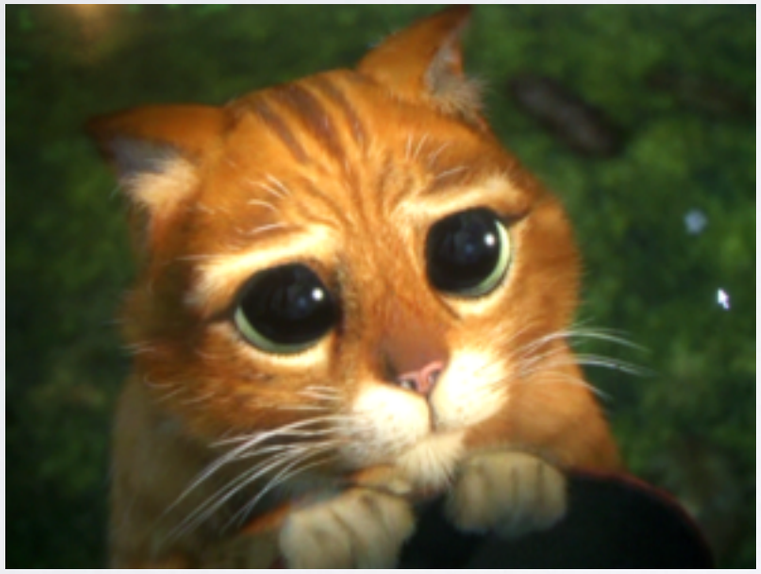
- Crawle **20 milliards** de **pages** / **jour**

RETOUR À LA RÉALITÉ

- 2 millions d'ordinateurs répartis
- dans +60 data centers
- sur le globe

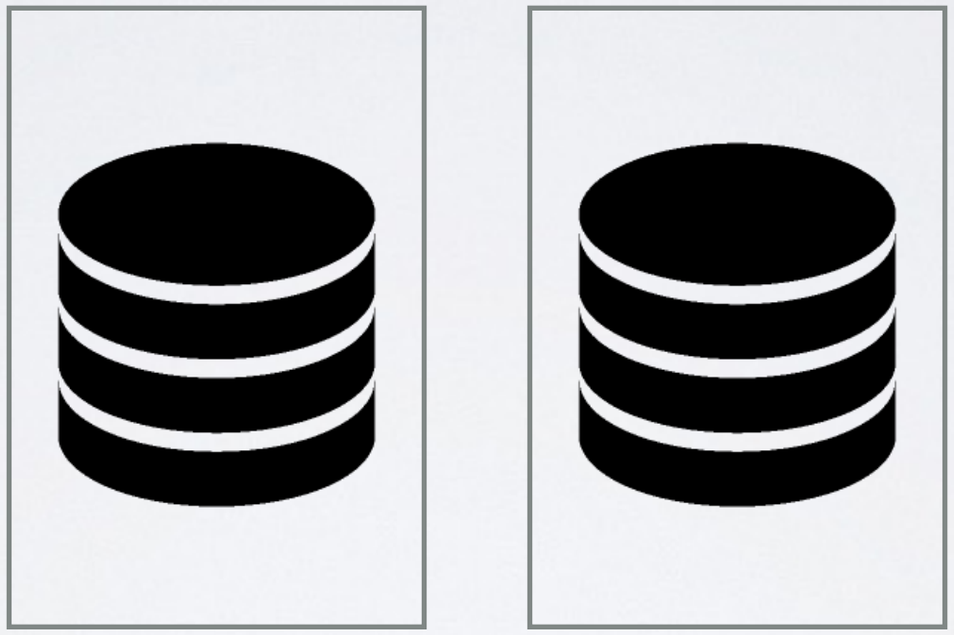
HORIZONTAL SCALING

PET VS CATTLE



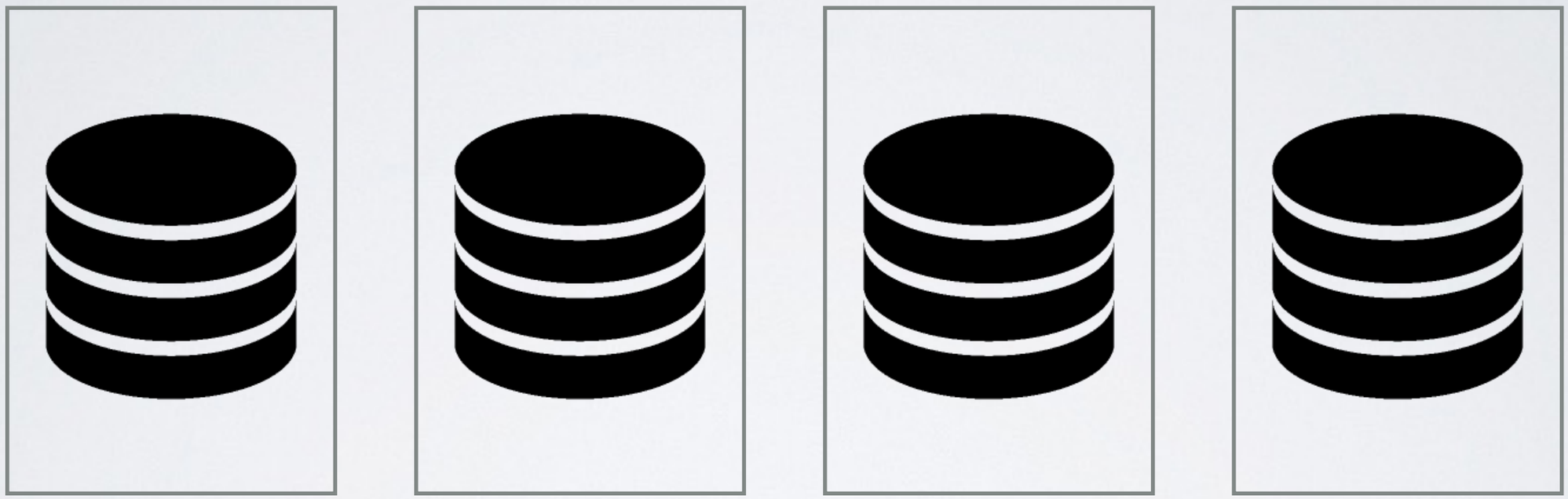
HORIZONTAL SCALING

HORIZONTAL SCALING



HORIZONTAL SCALING

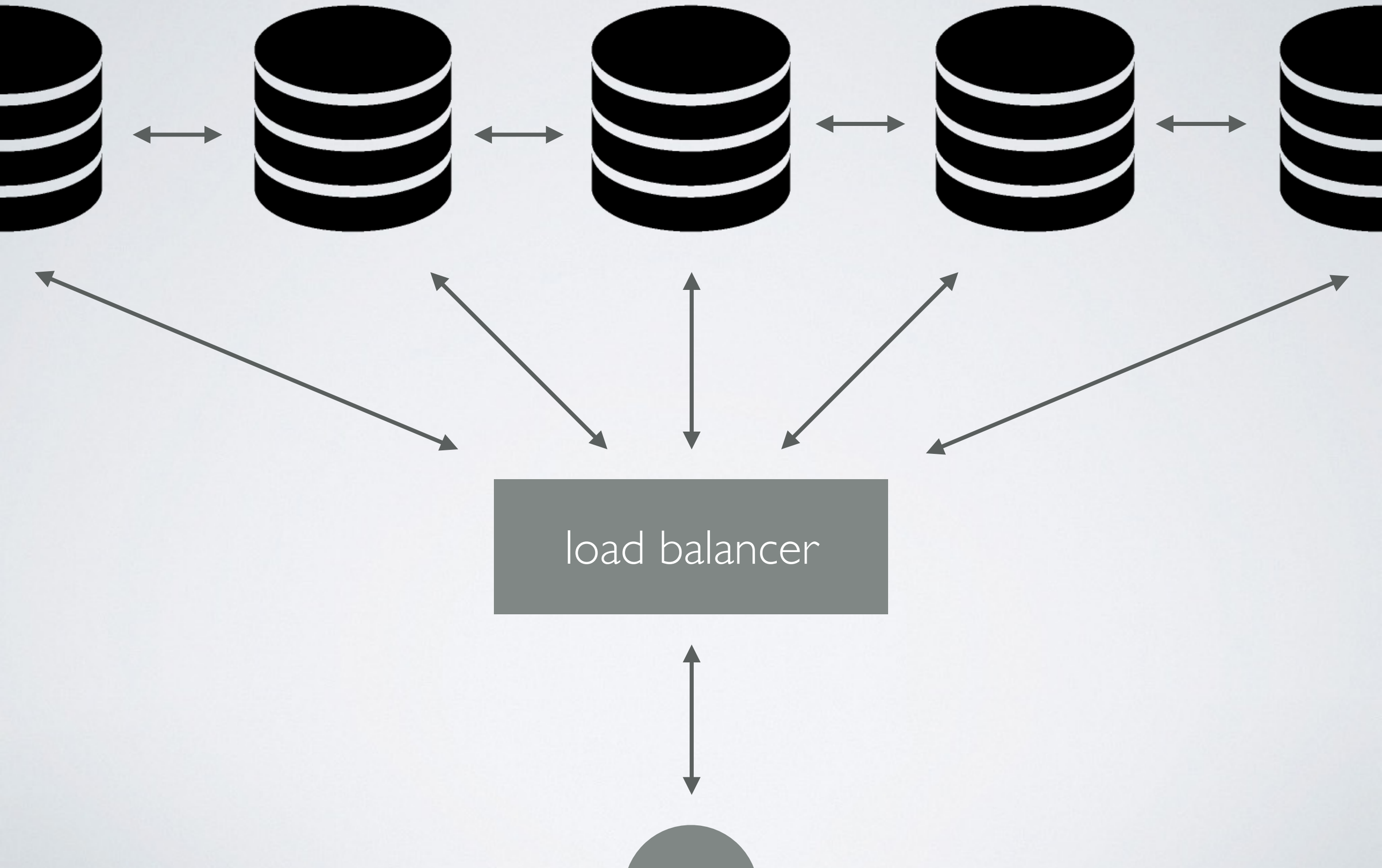
HORIZONTAL SCALING



MISE À L'ÉCHELLE

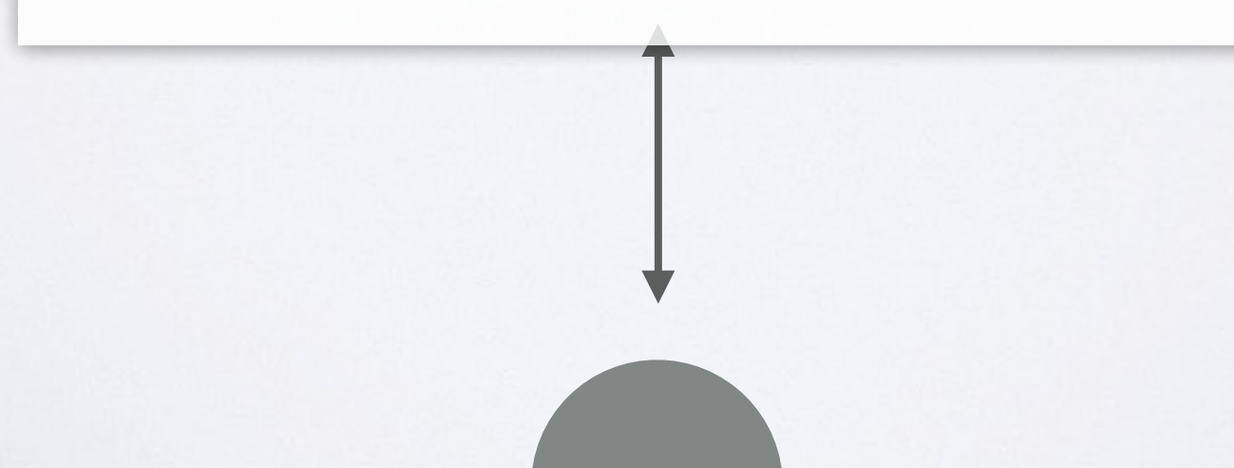
	Méthode (ex.)	Technologie	Mise en oeuvre
Code	Déploiement / Provisionning	Puppet, Docker, etc.	Trivial
Stockage	File System en réseau	AWS-S3, NFS, etc.	Trivial
BDD	?	?	?

SQL: MASTER / MASTER



Résultat

- ✓ Cohérence (ACID)
- ✓ Disponibilité
- load balancer
- **Limitation par I/O**



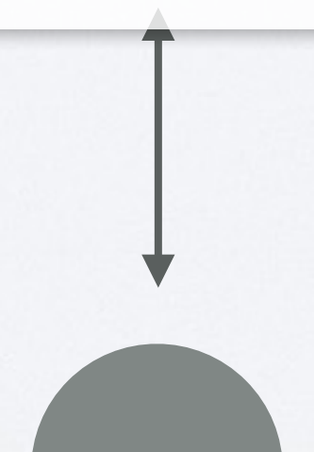
Méthode adaptée pour

- Garantir la disponibilité d'une base

Pas adaptée pour

- Scaling horizontal

load balancer



THÉORÈME CAP

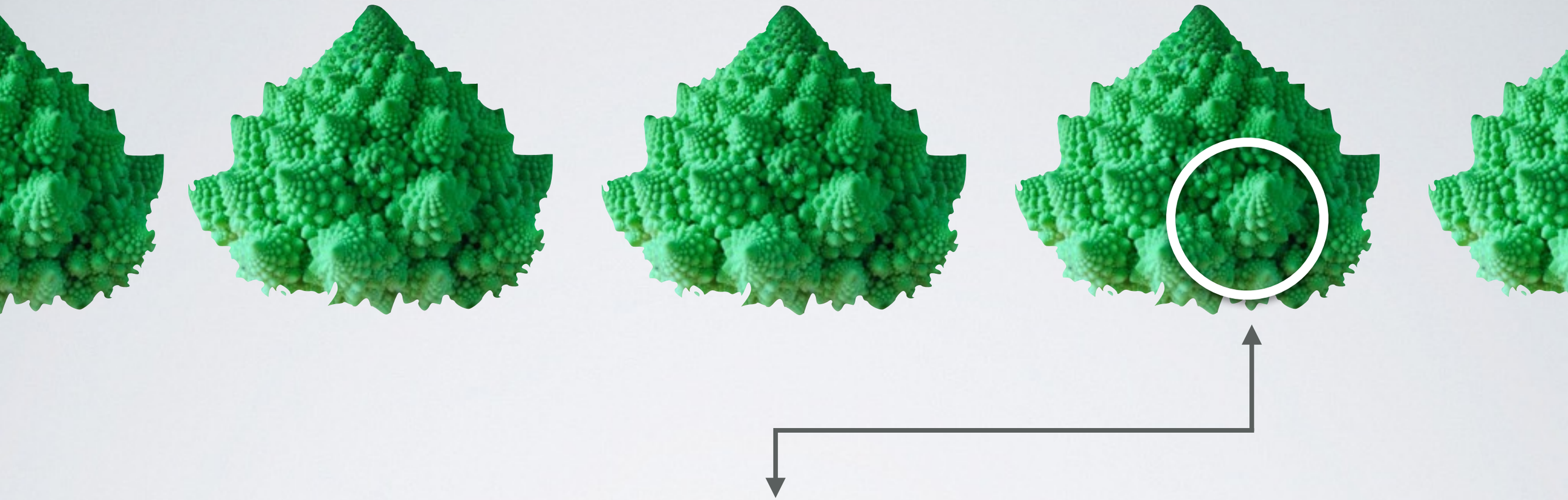
- Dans un système distribué,
  impossible de garantir en même temps:

  - Cohérence
    (tous les clients partagent la même vue des données)

  - Disponibilité
    (lecture / écriture pour tous les clients)

  - Tolérance au partionnement
    (le système fonctionne malgré la séparation physique des données)

MongoDB, partitionnement



load balancer

Théorème de CAP

✓ Cohérence

- ~~Disponibilité~~

✓ Tolérance au partionnement

CAP - COMPROMIS

(QUELQUES EXEMPLES)

	Availability (disponibilité)	Consistency (cohérence)	Partition Tolerance	Data Models
MySQL / Postgres	oui	oui	-	Relationnel
MongoDB	-	oui	oui	Document
Redis	-	oui	oui	Key-Value
Cassandra	oui	-	oui	Orienté Colonne
CouchDB	oui	-	oui	Document

* utilisation du SGBD par défaut

BASE

Basically **A**vailable

Soft state

Eventual consistency

- Les données
  - sont temporaires et pas à jour
  - doivent être « réconciliée »
  - vont finir par être cohérentes

- Elastic Search
- Cassandra (Facebook)
- Dynamo (Amazon)
- Oracle NoSQL Database
- Riak

QUELLE BDD UTILISER POUR MON APPLICATION ?

PROBLEMES

- Abondance de solutions
- Contradictions
- Buzz / Hype et évolutions

THE RIGHT
TOOL
FOR THE RIGHT
JOB

Le bon outil
pour le bon usage.

CHOIX D'UNE BASE DE DONNÉE

- Attention au « One size fits all »
- 1 marteau = tous les problèmes sont des clous
- Pas optimal
- Arrachage de cheveux

DIVISER LE PROBLÈME EN SOUS PROBLÈMES

- Plusieurs technologies de BDD
- ex: Redis + Postgres + MongoDB
- Microservices / machines virtuelles
- Chaque techno est adaptée à un sous-problème

CONNAITRE PLUSIEURS TECHNOLOGIES

- Forces, faiblesses

- Les essayer, se familiariser

- Identifier un problème
  qui nécessite une BDD en particulier

SE POSER LES BONNES QUESTIONS

- Ex pour Google
  - Est-ce que c'est ok d'afficher une donnée pas à jour ?
  - Est-ce que je dois réduire:
    - Le temps de crawling ?
    - Ou le temps de traitement des requêtes ?

EX: CRAIGSLIST (PETITES ANNONCES)

- MySQL
- Memcached
- Redis
- MongoDB
- Sphinx
- Filesystem

EX: CRAIGSLIST (PETITES ANNONCES)

- SQL (MySQL)
- Vertical (SSD) + RAM
- Horizontal
  - Cache (consultation 99%)

craigslist

EX: CRAIGSLIST (PETITES ANNONCES)

- NoSQL (MongoDB)
- Utilisé pour les archives
  - Schemaless
  - Scaling

- http://instagram-engineering.tumblr.com/post/40781627982/handling-growth-with-postgres-5-tips-from
  Scaling Postgres (Instagram)

- https://muut.com/blog/technology/redis-as-primary-datastore-wtf.html
  Un mec qui a détourné Redis pour son forum

- http://www.sarahmei.com/blog/2013/11/11/why-you-should-never-use-mongodb/
  Titre « troll » mais très instructif (Diaspora, problèmes de MongoDB dans la vraie vie)

- https://www.percona.com/live/mysql-conference-2012/sessions/living-sql-and-nosql-craigslist-pragmatic-approach
  Craigslist - Cohabitation de SQL et de NoSQL