# NFE204

## Introduction à la Recherche d'Information

Auteurs : Raphaël Fournier-S'niehotta, Philippe Rigaux
(fournier@cnam.fr, philippe.rigaux@cnam.fr)

EPN Informatique
Conservatoire National des Arts & Métiers, Paris, France

# Plan du cours

# Recherche d'information

3 cours :

- 16 octobre : introduction générale à la RI
- 23 octobre : indexation
- 30 octobre : recherche avec classement

Travaux pratiques :

- 11-12 octobre : Cassandra
- 18-19 octobre : MongoDB
- 8-9 novembre : ElasticSearch (1)
- 15-16 novembre : ElasticSearch (2)
- 10-11 janvier 2018 : calcul distribué (1)
- 17-18 janvier 2018 : calcul distribué (2)

# Plan du cours

# *Information Retrieval*, définition

## Une définition

La Recherche d'Information (*Information Retrieval*, IR) consiste à trouver des documents peu ou faiblement structurés, dans une grande collection, en fonction d'un besoin d'information.

- Recherche sur le Web. Utilisée quotidiennement par des milliards d'utilisateurs.
- Recherche dans votre boîte mail.
- Recherche sur votre ordinateur (*Spotlight*).
- Recherche dans une base documentaire, publique ou privée.

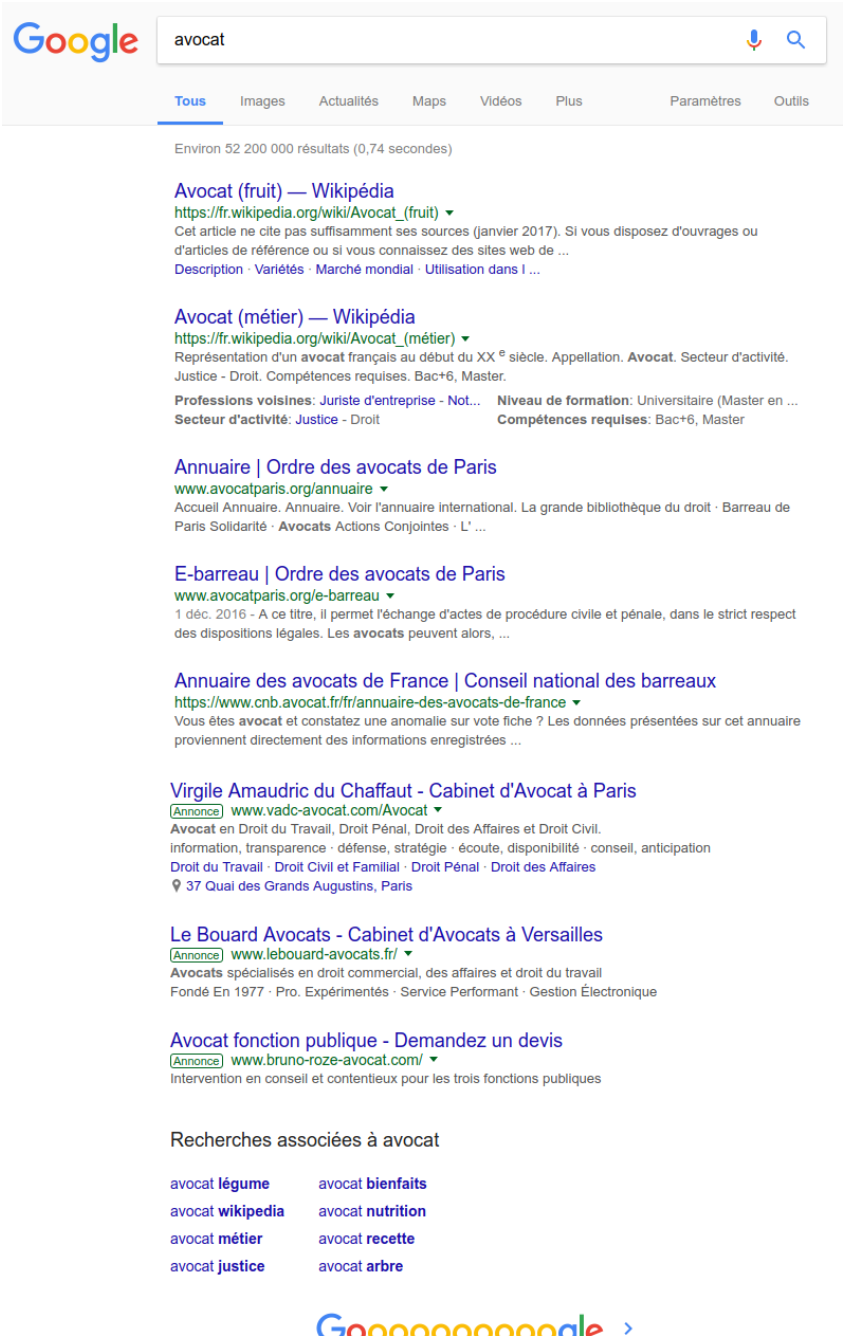
# *Information Retrieval*, définition

- Recherche plein texte : on cherche à examiner tous les mots de chaque document enregistré et à essayer de les faire correspondre à ceux fournis par l'utilisateur
- À bien distinguer d'une recherche type "base de données" : requête structurée, données structurées, réponse « exacte ».

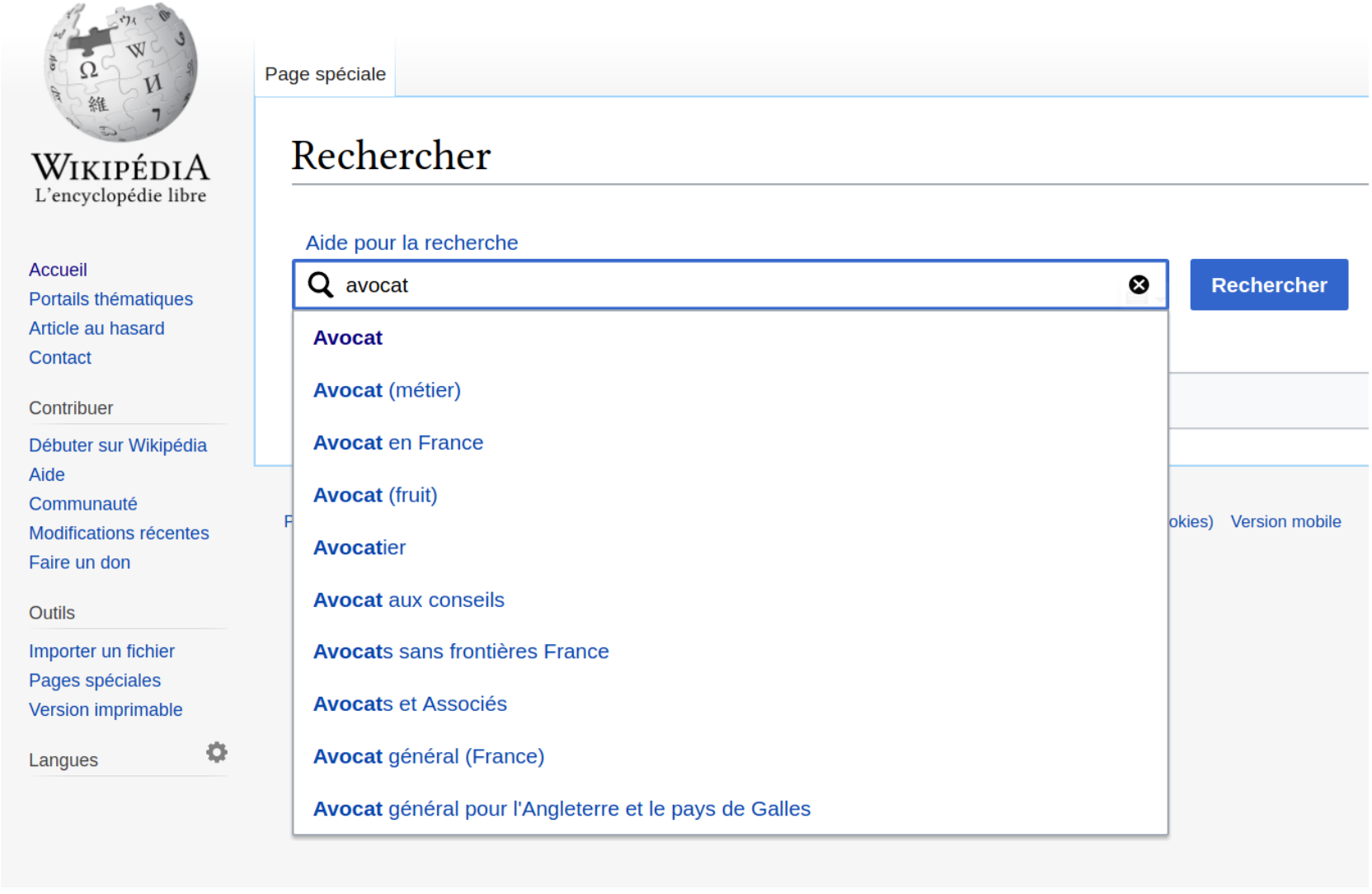
Contrairement à une base de données (SQL), le résultat dépend de l'interprétation d'un besoin. On ne peut jamais dire qu'un résultat est totalement exact (ou totalement faux).

# Un exemple de besoin

# Un moteur de recherche bien connu



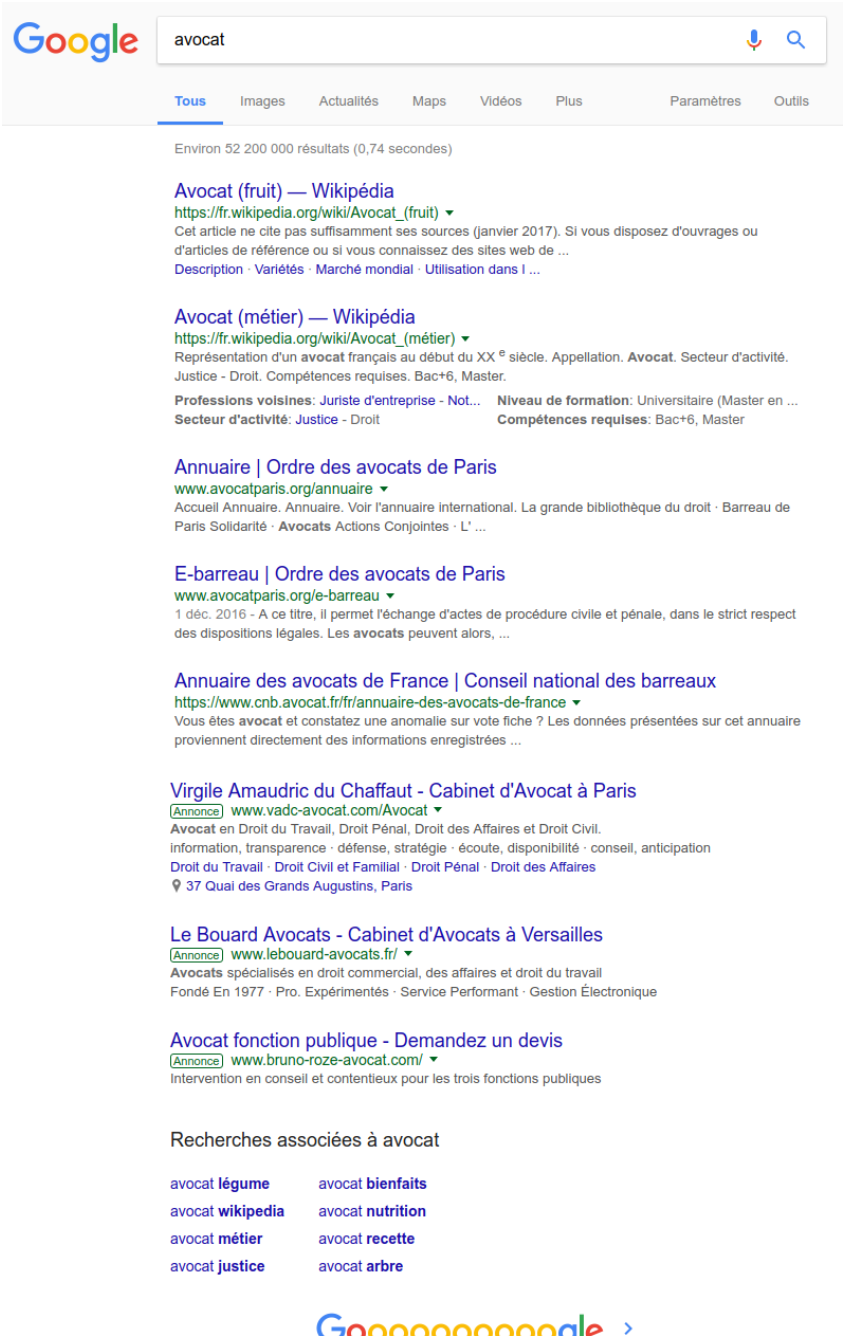
# Fonctionnalités avancées



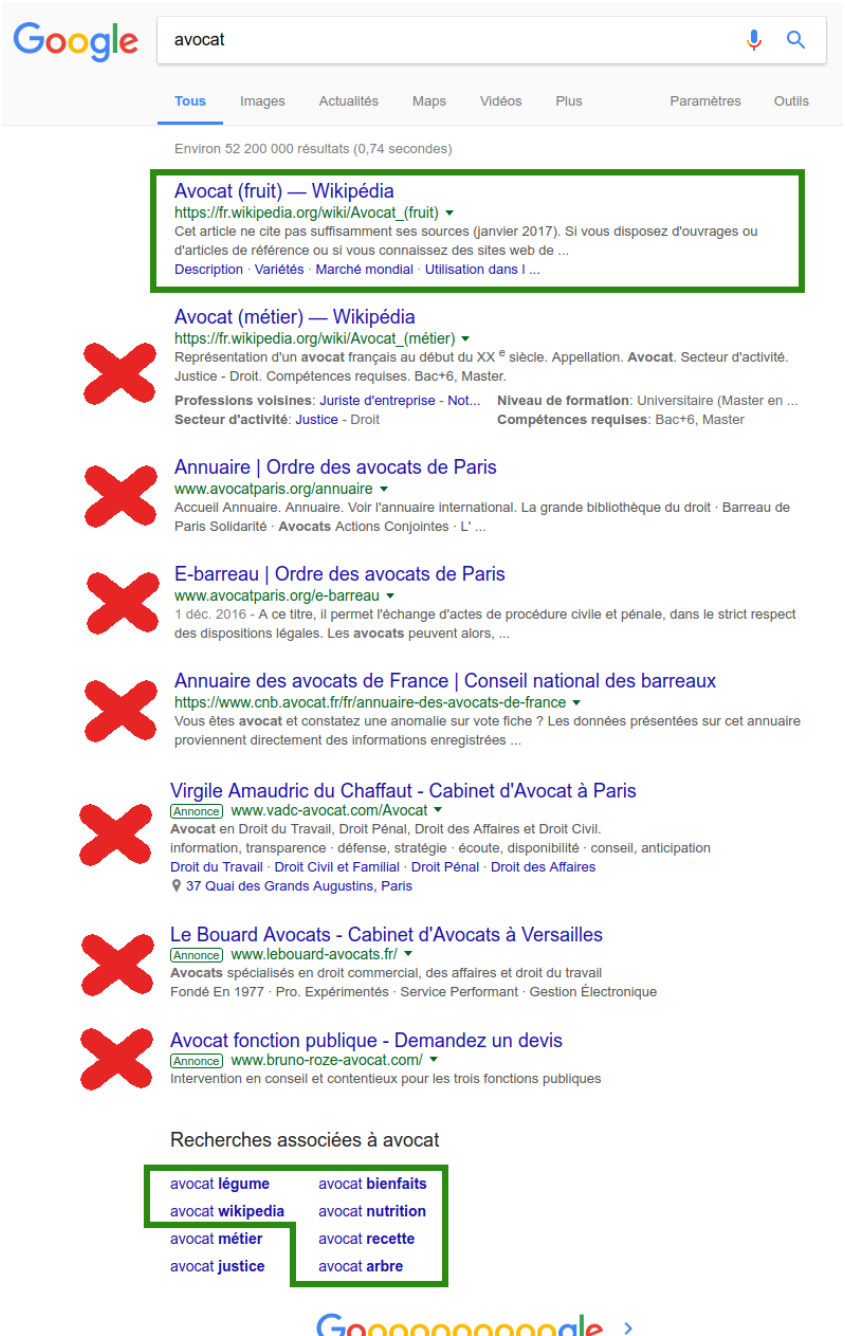
# Wikipédia : recherche avancée

# ECommerce : recherche à facettes

# Évaluation d'un moteur de recherche



# Évaluation d'un moteur de recherche



# Évaluation d'un moteur de recherche

## Deux notions importantes

- Faux positifs : ce sont les documents non pertinents inclus dans le résultat; ils ont été sélectionnés à tort.
- Faux négatifs : ce sont les documents pertinents qui ne sont pas inclus dans le résultat.

- La recherche plein texte est susceptible de récupérer beaucoup de faux positifs.
- La récupération de documents non pertinents est souvent provoquée par l'ambiguïté inhérente au langage naturel ;
- En général, chercher à réduire les faux positifs entraîne l'augmentation des faux négatifs, et réciproquement.

# Évaluation de la pertinence : précision et rappel

La précision mesure la fraction des vrais positifs dans le résultat $r$.

Si on note $t_p(r)$ et $f_p(r)$ le nombre de vrais et de faux positifs dans $r$, alors

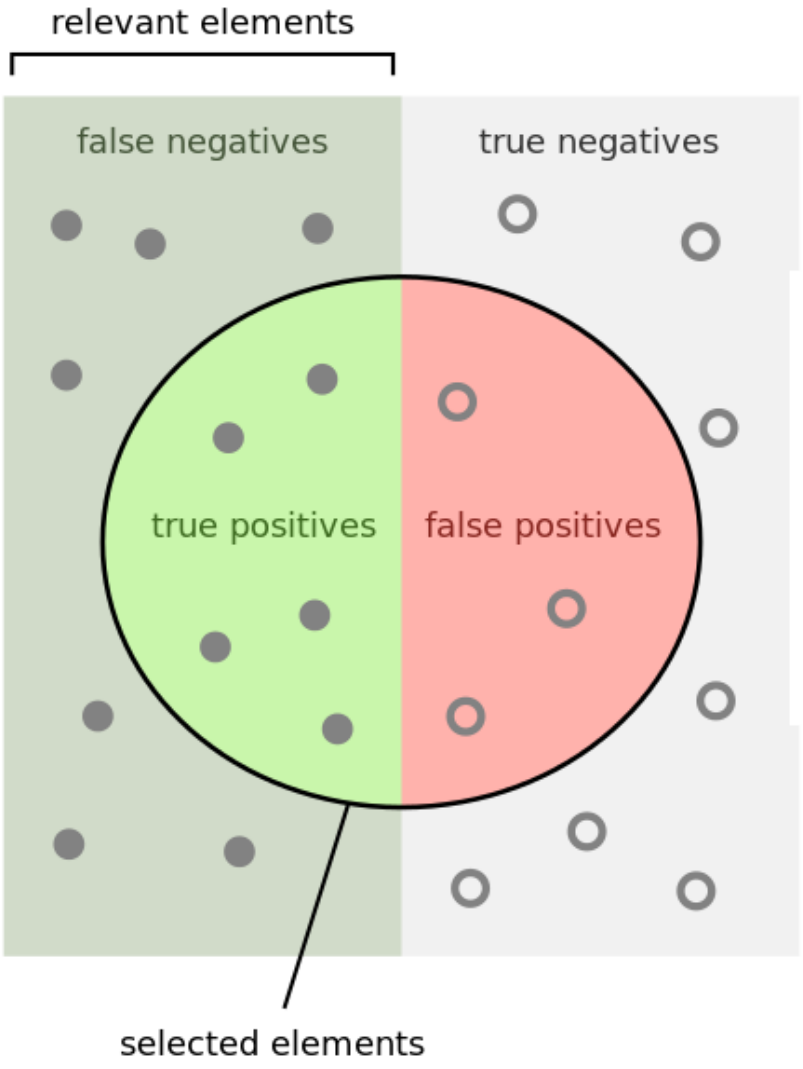
$$\text{précision} = \frac{t_p(r)}{t_p(r) + f_p(r)} = \frac{t_p(r)}{|r|}$$

Le rappel mesure la fraction de faux négatifs.

$$\text{rappel} = \frac{t_p}{t_p + f_n}$$

L'évaluation d'un système de RI est difficile : implique des tests rigoureux avec des utilisateurs sur un échantillon.

# Illustration des faux-positifs et négatifs




How many selected items are relevant?

$$\text{Precision} = \frac{\text{true positives}}{\text{selected elements}}$$

How many relevant items are selected?

$$\text{Recall} = \frac{\text{true positives}}{\text{relevant elements}}$$

# Performances

On peut améliorer les performances des moteurs de recherche à l'aide de plusieurs techniques :

- des requêtes plus structurées
  - requêtes booléennes,
  - expressions rationnelles,
  - proximité,
  - recherche d'expression
- des résultats classés
  - modèle vectoriel
  - PageRank
  - analyse sémantique latente

# Plan du cours

# Moteurs de recherche

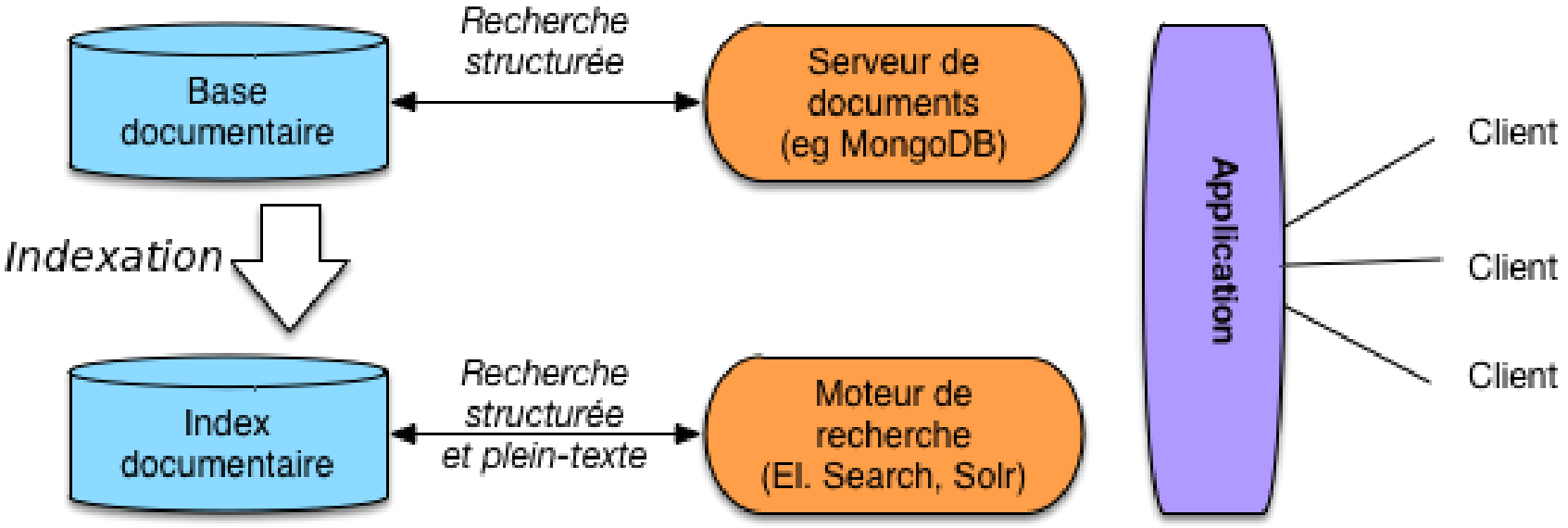
Moteurs libres

- Apache Solr (Lucene)
- ElasticSearch (Lucene)
- Sphinx
- Xapian

Moteurs commerciaux

- Google Search Appliance
- Exalead, Qwant
- Amazon CloudSearch
- Microsoft Azure Search, Bing

# Bases documentaires et moteur de recherche



# Bases documentaires et moteur de recherche

Un scénario typique :

- une recherche par mot-clé dans un site
- Les données du site sont gérées classiquement par une base documentaire (MySQL, Postgres, MongoDB)
- On extrait de la base tous les textes à indexer et on en fait des documents ES/Solr, à qui on les confie
- Ce dernier se charge alors de répondre quand un utilisateur emploie un champ de recherche.

# Bases documentaires et moteur de recherche

- Un moteur de recherche comme ElasticSearch ou Solr s'appuie sur un index
- Pourquoi ne pas utiliser directement le moteur de recherche comme gestionnaire des documents ?
- ElasticSearch permet des recherches puissantes, efficaces, ainsi que le stockage et l'accès aux documents.

# Bases documentaires et moteur de recherche

- Mais, comme ElasticSearch est entièrement consacré à la recherche, c'est-à-dire la lecture la plus efficace possible de documents.
- il s'appuie pour cela sur des structures compactes, compressées, optimisées (les **index inversés**)
- ce n'est pas un très bon outil pour les autres fonctionnalités d'une base de données :
  - Le stockage par exemple n'est ni aussi robuste ni aussi stable.
  - Pour des raisons qui tiennent à la structure de ces index, les mises à jour sont difficiles et s'effectuent difficilement en temps réel

# Exercices simples

Exercices : fin de section 1 du chapitre 9 sur `http://b3d.bdpedia.fr`

# Plan du cours

# Exemple de base

Un ensemble (modeste) de documents nous servira de guide.

- $d_1$ Le loup est dans la bergerie.
- $d_2$ Le loup et le trois petits cochons.
- $d_3$ Les moutons sont dans la bergerie.
- $d_4$ Spider Cochon, Spider Cochon, il peut marcher au plafond.
- $d_5$ Un loup a mangé un mouton, les autres loups sont restés dans la bergerie.
- $d_6$ Il y a trois moutons dans le pré, et un mouton dans la gueule du loup.
- $d_7$ Le cochon est à 12€ le Kg, le mouton à 10€/Kg.
- $d_8$ Les trois petits loups et le grand méchant cochon.

# Le besoin et la solution

## Besoin

On veut chercher tous les documents parlant de loups, de moutons mais pas de bergerie.

Parcourir tous les documents ? (Grep)

- potentiellement long;
- critère "pas de bergerie" n'est pas facile à traiter;
- autres types de recherche ("le mot 'loup' doit être près du mot 'mouton'") sont difficiles;
- comment classer par pertinence les documents trouvés ?

Structure spécialisée : la matrice d'incidence et surtout son inversion.

# Matrice avec documents en ligne

On sélectionne un ensemble de mots (ou termes), constituant notre vocabulaire (ou dictionnaire).

Documents en ligne, termes en colonnes. Dans chaque cellule : 1 si le terme est dans le document, 0 sinon.

La matrice d'incidence

| | loup | mouton | cochon | bergerie | pré | gueule |
|---|---|---|---|---|---|---|
| $d_1$ | 1 | 0 | 0 | 1 | 0 | 0 |
| $d_2$ | 1 | 0 | 1 | 0 | 0 | 0 |
| $d_3$ | 0 | 1 | 0 | 1 | 0 | 0 |
| $d_4$ | 0 | 0 | 1 | 0 | 0 | 0 |
| $d_5$ | 1 | 1 | 0 | 1 | 0 | 0 |
| $d_6$ | 1 | 1 | 0 | 0 | 1 | 1 |
| $d_7$ | 0 | 1 | 1 | 0 | 0 | 0 |
| $d_8$ | 1 | 0 | 1 | 0 | 0 | 0 |

# Pour effectuer la recherche

(loup, mouton, et pas bergerie) On prend les vecteurs binaires des termes (les colonnes).

- Loup : 11001101
- Mouton : 00101110
- Bergerie : 01010011

Puis :

- ET logique sur les vecteurs de Loup et Mouton, on obtient 00001100.
- ET logique avec le complément du vecteur de Bergerie (01010111)

On obtient 00000100, d'où on déduit que la réponse est limitée au document $d_6$.

Opération binaires très efficace, mais...

# Passons à grande échelle

Quelques hypothèses :

- Un million de documents, mille mots chacun en moyenne.
- Disons 6 octets par mot, soit 6 Go (ce n'est pas une si grosse base que cela)
- Disons 500 000 termes distincts

$\Rightarrow$ la matrice d'incidence a :

- $10^6$ lignes et 500 000 colonnes soit $500 \times 10^9$ bits
- soit 62 Go approximativement

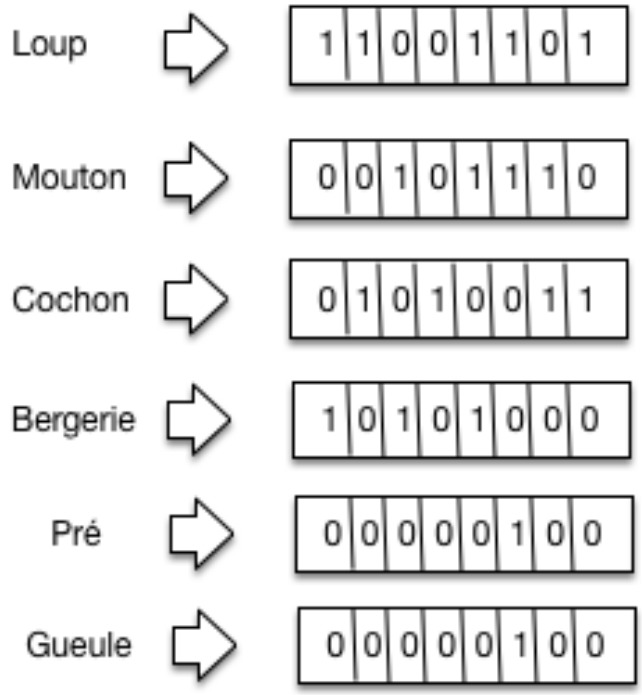
Ne tient pas en mémoire, ce qui va beaucoup compliquer les choses....

Comment faire mieux ?

# On peut faire mieux

Il vaut mieux avoir les termes en ligne pour disposer des vecteurs dans une zone mémoire contigue.

On parle de matrice inversée, et de liste inversée. Une liste par terme; dans chaque liste, 1 pour les documents contenant le terme.

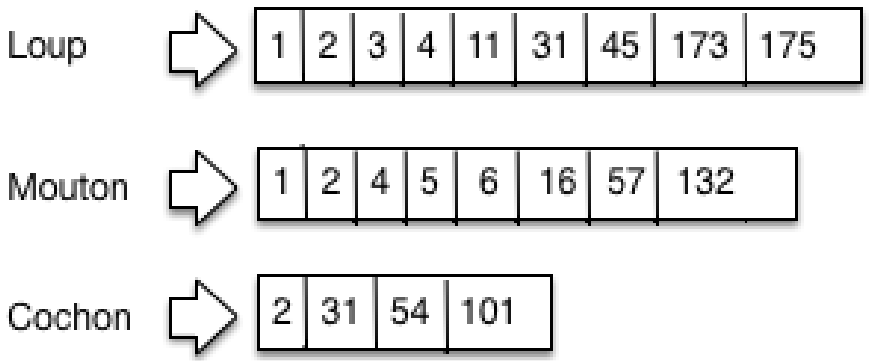



## Important

Les mises à jour beaucoup plus difficiles car elles impliquent la réorganisation d'une liste compacte. Prix à payer pour l'efficacité en lecture (recherche).

# On peut encore faire mieux

La matrice est creuse : il n'y a que $10^9$ positions avec des 1, soit un sur 500.




## Essentiel

On place dans les cellules l'identifiant du document. De plus chaque liste est triée sur l'identifiant du document.

# Index inversé

La structure utilisée dans tous les moteurs de recherche.

- Un répertoire contient tous les termes.
- Une liste (inversée) est associée à chaque terme, triée par docId.
- Chaque élément de la liste est appelé une entrée.

Concept : la notion de terme (token en anglais) est différente de celle de "mot".

Vocabulaire : le répertoire est parfois appelé *dictionnaire* ; les listes sont des *posting list* en anglais; les entrées sont des *postings*.

Efficacité : le répertoire devrait toujours être en mémoire ; les listes, autant que possible en mémoire, sinon fichiers contigus sur le disque.

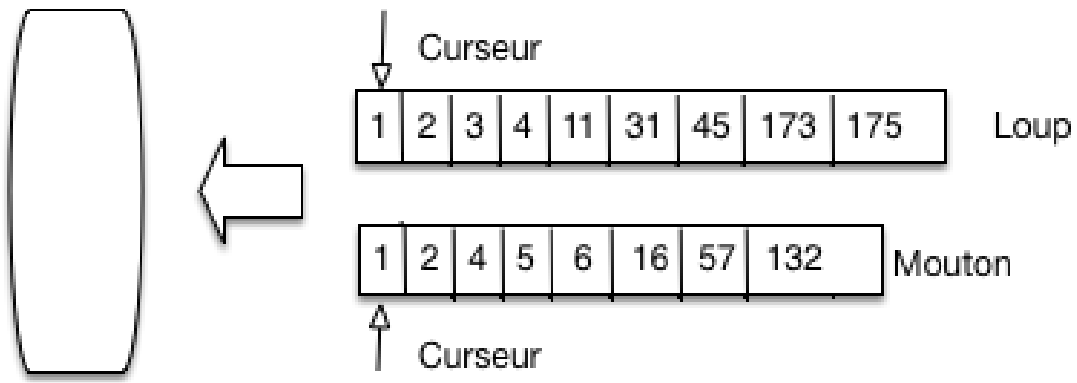
# Traitement d'une recherche par *fusion* de listes triées

Recherchons les documents parlant de loup **ET** de mouton.

Algorithme par fusion : on parcourt séquentiellement les deux listes.

On compare à chaque étape les docId ; s'ils sont égaux : trouvé!

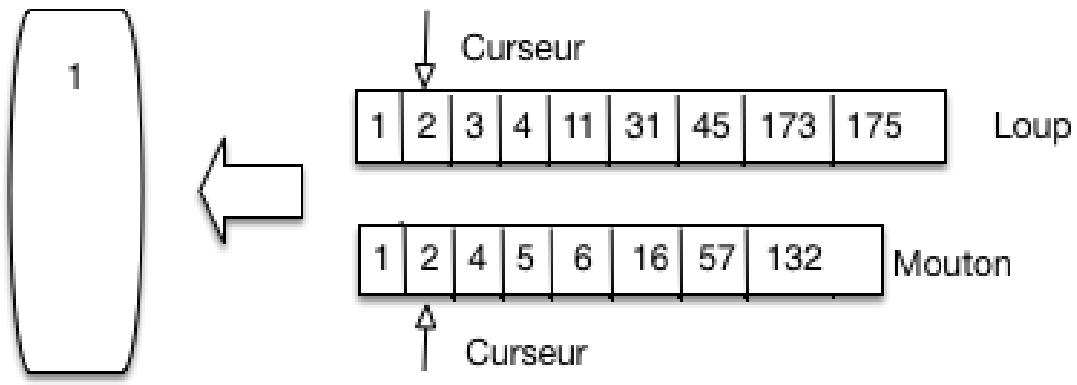
On avance sur la liste du plus petit docId.



# Traitement d'une recherche par *fusion* de listes triées

Recherchons les documents parlant de loup **ET** de mouton.

Algorithme par fusion : on parcourt séquentiellement les deux listes.
On compare à chaque étape les docId ; s'ils sont égaux : trouvé!
On avance sur la liste du plus petit docId.



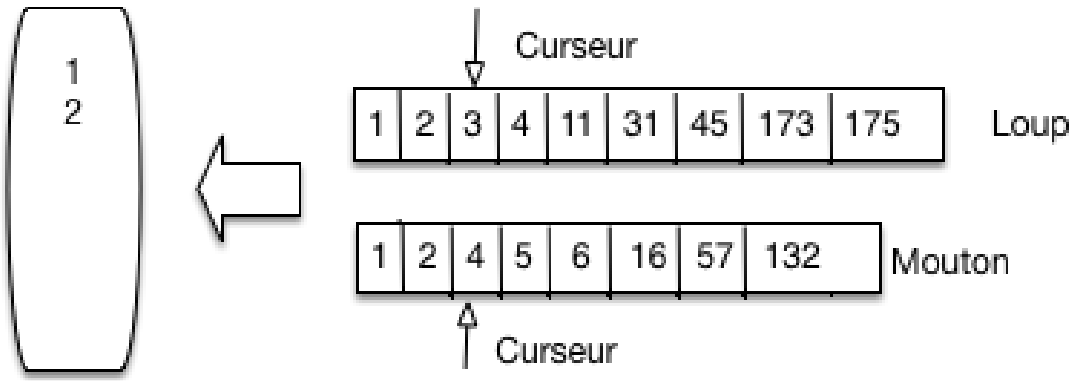

Un seul parcours suffit : recherche linéaire, parcours séquentiel. Pas mieux.

C'est une recherche dite Booléenne : pas de classement, résultat exact.

# Traitement d'une recherche par *fusion* de listes triées

Recherchons les documents parlant de loup **ET** de mouton.

Algorithme par fusion : on parcourt séquentiellement les deux listes.
On compare à chaque étape les docId ; s'ils sont égaux : trouvé!
On avance sur la liste du plus petit docId.



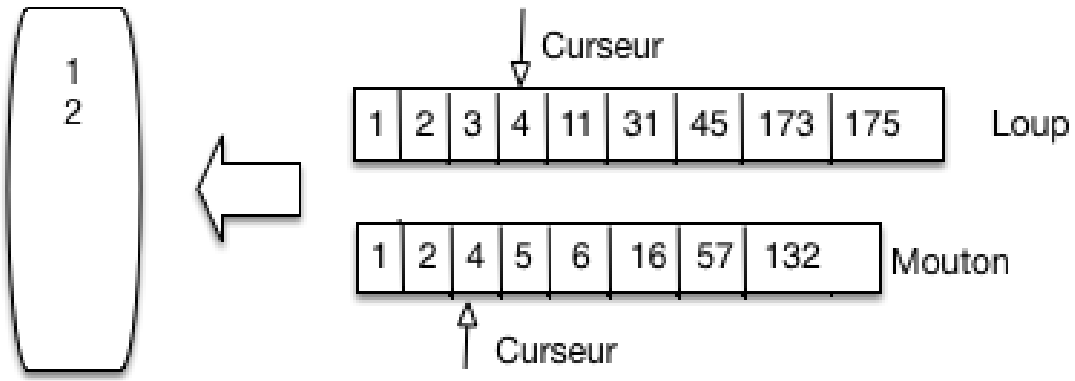

Un seul parcours suffit : recherche linéaire, parcours séquentiel. Pas mieux.

C'est une recherche dite Booléenne : pas de classement, résultat exact.

# Traitement d'une recherche par *fusion* de listes triées

Recherchons les documents parlant de loup **ET** de mouton.

Algorithme par fusion : on parcourt séquentiellement les deux listes.
On compare à chaque étape les docId ; s'ils sont égaux : trouvé!
On avance sur la liste du plus petit docId.



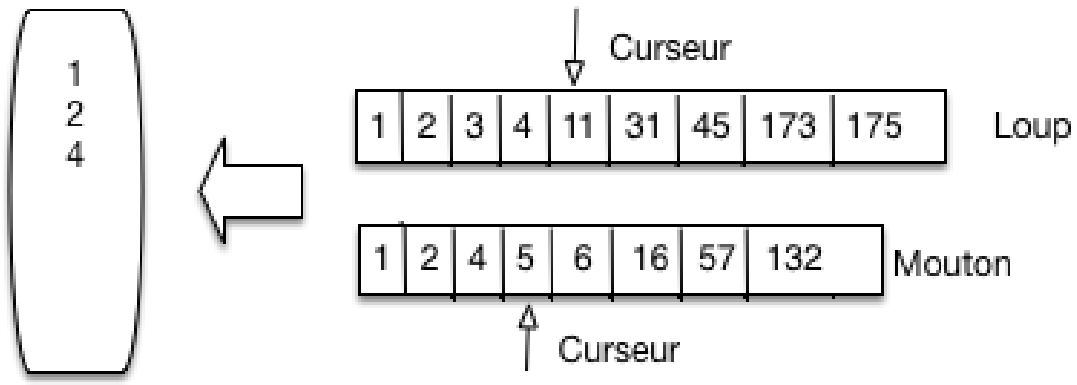

Un seul parcours suffit : recherche linéaire, parcours séquentiel. Pas mieux.

C'est une recherche dite Booléenne : pas de classement, résultat exact.

# Traitement d'une recherche par *fusion* de listes triées

Recherchons les documents parlant de loup **ET** de mouton.

Algorithme par fusion : on parcourt séquentiellement les deux listes.
On compare à chaque étape les docId ; s'ils sont égaux : trouvé!
On avance sur la liste du plus petit docId.



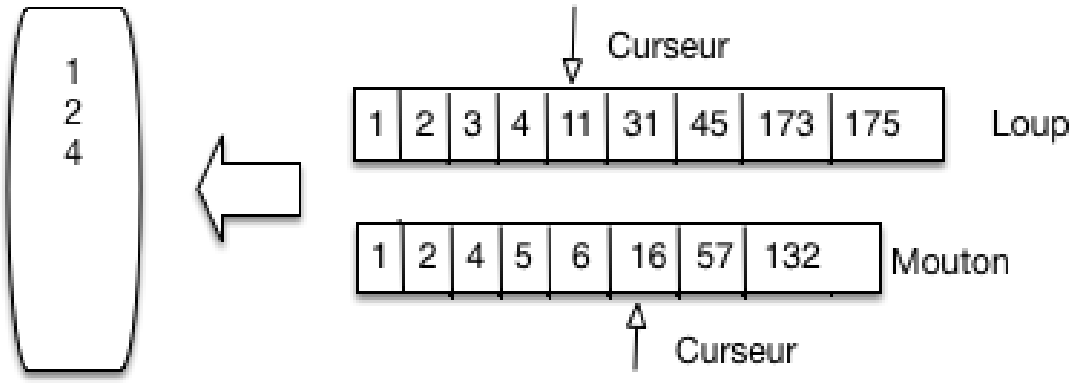

Un seul parcours suffit : recherche linéaire, parcours séquentiel. Pas mieux.

C'est une recherche dite Booléenne : pas de classement, résultat exact.

# Traitement d'une recherche par *fusion* de listes triées

Recherchons les documents parlant de loup **ET** de mouton.

Algorithme par fusion : on parcourt séquentiellement les deux listes.
On compare à chaque étape les docId ; s'ils sont égaux : trouvé!
On avance sur la liste du plus petit docId.



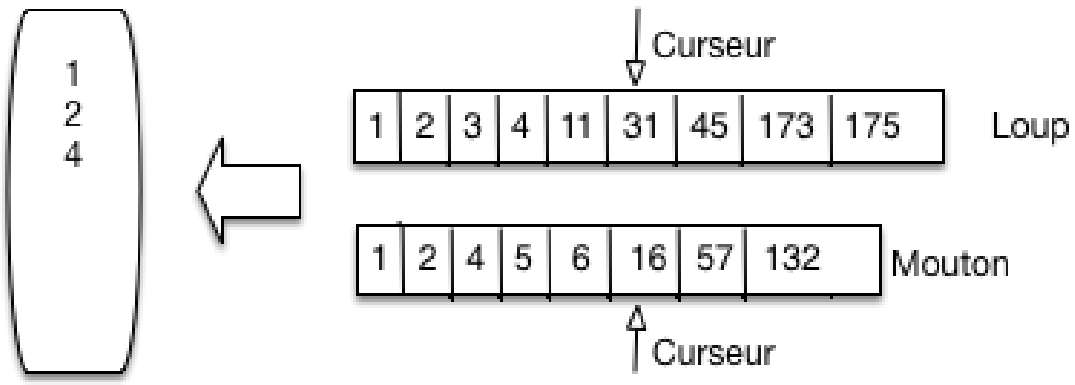

Un seul parcours suffit : recherche linéaire, parcours séquentiel. Pas mieux.

C'est une recherche dite Booléenne : pas de classement, résultat exact.

# Traitement d'une recherche par *fusion* de listes triées

Recherchons les documents parlant de loup **ET** de mouton.

Algorithme par fusion : on parcourt séquentiellement les deux listes.
On compare à chaque étape les docId ; s'ils sont égaux : trouvé!
On avance sur la liste du plus petit docId.



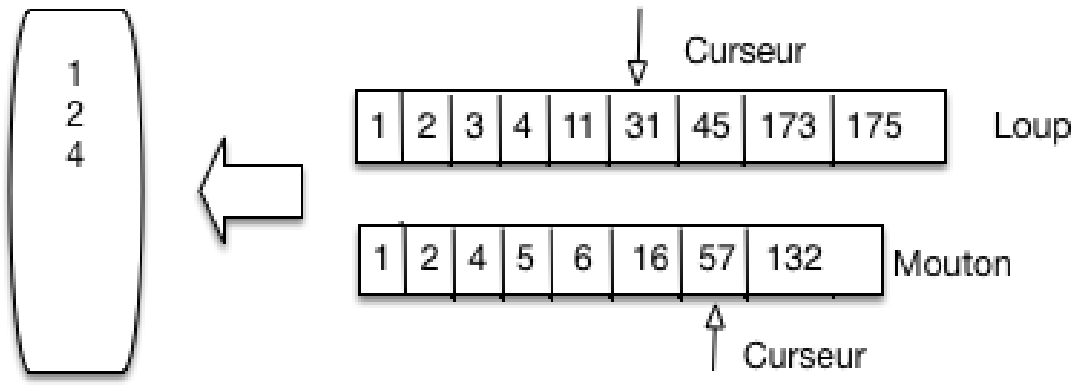

Un seul parcours suffit : recherche linéaire, parcours séquentiel. Pas mieux.

C'est une recherche dite Booléenne : pas de classement, résultat exact.

# Traitement d'une recherche par *fusion* de listes triées

Recherchons les documents parlant de loup **ET** de mouton.

Algorithme par fusion : on parcourt séquentiellement les deux listes.
On compare à chaque étape les docId ; s'ils sont égaux : trouvé!
On avance sur la liste du plus petit docId.



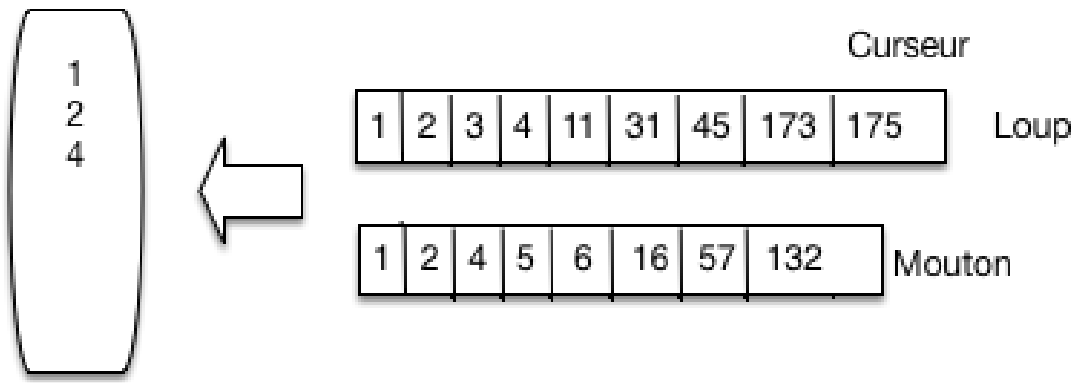

Un seul parcours suffit : recherche linéaire, parcours séquentiel. Pas mieux.

C'est une recherche dite Booléenne : pas de classement, résultat exact.

# Traitement d'une recherche par *fusion* de listes triées

Recherchons les documents parlant de loup **ET** de mouton.

Algorithme par fusion : on parcourt séquentiellement les deux listes.
On compare à chaque étape les docId ; s'ils sont égaux : trouvé!
On avance sur la liste du plus petit docId.




Un seul parcours suffit : recherche linéaire, parcours séquentiel. Pas mieux.

C'est une recherche dite Booléenne : pas de classement, résultat exact.

# Algorithme

```
// Fusion de deux listes l1 et l2
function Intersect($l1, $l2)
{
  $résultat = [];
  // Début de la fusion des listes
  while ($l1 != null and $l2 != null) {
    if ($l1.docId == $l2.docId) {
      // On a trouvé un document contenant les deux termes
      $résultat += $l1.docId;
      // Avançons sur les deux listes
      $l1 = $l1.next; $l2 = $l2.next;
    }
    else if ($l1.docId < $l2.docId) {
      // Avançons sur l1
      $l1 = $l1.next;
    }
    else {
       // Avançons sur l2
      $l2 = $l2.next;
    }
  }
}
```

# Optimisation

- Si l'on veut effectuer la recherche :

```
A AND B AND C
```

- Quelle manière de procéder va optimiser le résultat ?

# Optimisation

- Si l'on veut effectuer la recherche :

```
A AND B AND C
```

- Quelle manière de procéder va optimiser le résultat ?
- on stocke la taille des listes et l'on commence par faire l'intersection des plus petites
- de nombreuses autres questions d'optimisation existent
- exemple : stocker l'une des listes en mémoire et calculer les intersections à la volée, en lisant depuis le disque

# Premier bilan

En résumé : index inversé, tri et compaction, parcours linéaire très rapide sont les fondements techniques de la RI.

- Permet d'effectuer des recherches puissantes (combinaison de critères) et flexibles.
- Garantissent une très grande efficacité.

Que reste-t-il à faire ?

Quels termes ? Quels termes indexe-t-on ? Beaucoup moins facile que ça n'en a l'air...

Quelles requêtes ? De la plus simple ("sac de mots") à plus structurée (index struturé, requêtes Boolénnes, recherche de phrases).

Performance. Construction, compression, distribution, optimisation des accès, mises à jour, etc.

Classement ? Si j'ai des millions de documents dans le résultat, je veux les classer. Comment ?

# Plan du cours

# Présentation d'Elasticsearch

Elasticsearch est un moteur de recherche basé sur Lucene

- grande communauté d'utilisateurs
- open source, profite de la recherche dans le domaine sur Lucene
- utilisé par de grands opérateurs du Web sur des collections immenses

En fait, plusieurs composants dont :

- Logstash, l'ETL, pour extraire transformer et charger les données
- ElasticSearch, le moteur lui-même
- Kibana, pour produire des tableaux de bords de surveillance

# Installer ElasticSearch 2.4 et le plugin Kopf

```
sudo docker run -d --name es1  -p 9200:9200 -p 9300:9300 elasticsearch:2.4 \
   -Des.index.number_of_shards=1 \
   -Des.index.number_of_replicas=0
```

```
   $ sudo docker ps -a
   $ sudo docker exec <containerId> plugin install lmenezes/elasticsearch-kopf

   # Adresse IP de votre ElasticSearch
   $ ip="$(docker inspect --format '{{ .NetworkSettings.IPAddress }}' es1)"
   $ echo $IP
```

# Un premier document

Télécharger la liste des films sur `http://webscope.bdpedia.fr/index.php?ctrl=xml`
(Exports, Listes des films complets, un fichier JSON par film ZIPpé)

```
$ unzip movies-json.zip
$ cd movies-json/
<EDITER le fichier pour enlever le champ id>
```

# Notre premier document


```
{
  "title": "Vertigo",
  "year": 1958,
  "genre": "drama",
  "summary": "Scottie Ferguson, ancien inspecteur de police, est sujet au vertige depuis qu'il a
              vu mourir son collègue. Elster, son ami, le charge de surveiller sa femme,
              Madeleine, ayant des tendances suicidaires. Amoureux de la jeune femme Scottie ne
              remarque pas le piège qui se trame autour de lui et dont il va être la victime... ",
  "country": "DE",
  "director":         {
    "_id": "artist:3",
    "last_name": "Hitchcock",
    "first_name": "Alfred",
    "birth_date": "1899"
  },
  "actors": [
  {
    "_id": "artist:15",
    "first_name": "James",
    "last_name": "Stewart",
    "birth_date": "1908",
    "role": "John Ferguson"
  },
  {
    "_id": "artist:282",
    "first_name": "Arthur",
    "last_name": "Pierre",
    "birth_date": null,
    "role": null
  }
  ]
}
```

# Un premier document

```
# Utiliser l'API REST pour mettre le document dans ElasticSearch
$ curl -X PUT http://localhost:9200/nfe204/movies/movie:1 --data-binary @movie_1.json

# constater que l'on a bien un index appelé nfe204
$ firefox http://localhost:9200/_plugin/kopf/

# Ensuite, récupérer le document avec curl :
$ curl -X GET http://localhost:9200/nfe204/movies/movie:1
```

# D'autres documents

```
$ wget http://b3d.bdpedia.fr/files/movieselastic.json

# Utiliser l'API REST et l'interface bulk pour déposer les documents dans ElasticSearch
$ curl -XPUT http://localhost:9200/_bulk --data-binary @movieselastic.json
# constater avec Kopf que l'on a 4849 documents importés dans l'index movies
```

# Plan du cours

# Interrogation

- ElasticSearch s'appuie sur le système d'indexation **Lucene**, dont le rôle est essentiellement de créer les index inversés, et d'implanter les algorithmes de parcours brièvement introduits dans la session précédente.
- Lucene propose un langage de recherche basé sur des combinaisons de mot-clés, langage étendu et raffiné par ElasticSearch (cf plus tard)
- La première méthode pour transmettre des recherches est de passer une expression en paramètre à l'URL :

```
$ curl http://localhost:9200//nfe204/movies/_search?q=alien
# utilisable dans l'interface Kopf, onglet Rest
```

# la réponse d'ES


```
{
"took": 3,
"timed_out": false,
"_shards": {
        "total": 4,
        "successful": 4,
        "failed": 0
},
"hits": {
        "total": 20,
        "max_score": 1.2078758,
        "hits": [
                {
                        "_index": "movies",
                        "_type": "movie",
                        "_id": "764",
                        "_score": 1.2078758,
                        "_source": {
                                "fields": {
                                        "directors": [
                                                "Duncan Jones"
                                        ],
                                        "genres": [
                                                "Action",
                                                "Adventure",
                                                "Fantasy"
                                        ],
                                        "plot": "An epic fantasy/adventure based on the popular video game series.",
                                        "title": "Warcraft",
                                        "rank": 764,
                                        "actors": [
                                                "Paula Patton",
                                                "Paul Dano",
                                                "Anson Mount"
                                        ],
                                        "year": 2015
                                },
                                "id": "tt0803096",
                                "type": "add"
                        }
                },
```

# Termes

- Notion de base : le **terme**
- c'est un mot au sens usuel
- ou une séquence de mots entre apostrophes

On peut interroger un index avec :

```
space vessel
```

Puis :

```
"space vessel"
```

- Première recherche : documents avec "space", "vessel" ou les deux
- Deuxième : seulement "space vessel" (côte à côte)

# Termes (suite)

- la recherche d'un terme s'effectue toujours sur un champ.
- La syntaxe complète pour associer le champ et le terme est:

```
champ:terme
```

- si non précisé, c'est le champ par défaut qui est utilisé
- pratique courante : concaténer toutes les chaînes de caractères en un champ "text" général, défini par défaut
- Nos requêtes deviennent :

```
text:space text:vessel
```

- et

```
text:"space vessel"
```

# Termes (suite)

- Les valeurs des termes (dans la requête) et le texte indexé sont tous deux soumis à des transformations spécifiées dans le schéma.
- Une transformation simple est de tout transcrire en minuscules.

```
text:"Space Vessel"
```

- Les transformations appliquées à la requête ET au texte indexé doivent être cohérentes : si les termes sont transformés en majuscules, et le texte indexé en minuscules, on n'aura jamais de résultat!

# Termes (suite)

On peut spécifier des termes (pas des séquences) incomplets

- le '?' indique un caractère inconnu
  - "opti?al" désigne "optimal", "optical", etc.
- le '*' indique n'importe quelle séquence de caractères
  - "opti*" pour toute chaîne commençant par "opti"

Approximations avec "∼" :

- Rechercher "optimal" et "optimal∼"
- 0 et 1 résultat ("optical")
- Proximité des termes par une distance d'édition :
  (nb opérations pour passer de "optimal" à "optical")

Intervalles :

- [] bornes comprises
- { } bornes exclues

---

```
%price:[100 TO 200]
```

---

# Requêtes Booléennes

- Les critères peuvent être combinés avec des opérateurs Booléens : **AND**, **OR** et **NOT**
- Attention : majuscules

```
%price:[100 TO 300] OR popularity:5
%price:[100 TO 300] AND NOT popularity:5
%popularity:6 AND features:matrix
```

- Par défaut, c'est un **OR** qui est appliqué
- Recherche sur plusieurs critères ramène l'union des résultats sur chaque critère pris individuellement

**Exercice** Exprimez les recherches suivantes sur votre base de données :

- les films dans lesquels on parle de "hunter";
- même critère, mais en ajoutant le mot-clé "bounty";
- films avec Kate Winslett et Leonardo di Caprio;
- films qui sont soit des drames, soit du fantastique;
- films avec le mot-clé « France »; obtient-on les films produits en France? Sinon pourquoi? Que faudrait-il faire?
- on recherche le film « Sleepy Hollow »; effectuez une recherche sur le titre (« Sleepy », « Hollow », « Sleepy Hollow ») puis sur le résumé.
- films satisfaisant une combinaison de critères: parus entre 1990 et 2000 et aux USA, ou contenant les mots-clés « Michael » et « Sonny »;

Vous êtes invités à effectuer les recherches avec ou sans majuscules, à chercher des phrases comme « bounty and hunter », à indiquer ou non des noms de champs, et à interpréter les résultats (ou l'absence de résultat) obtenus.

# Plan du cours

# La semaine prochaine

- Indexation
  - Choix des champs
  - Configuration de l'analyse
  - Effets de l'analyse