

# Voyage au cœur d'un **index Lucene**

Comprendre les moteurs de recherche de l'intérieure



Quentin Lobbé  
Phd @TélécomParisTech

Inspiré des travaux d'Adrien Grand  
Software Ingeneer @Elasticsearch

## Qui suis je ?

> 2015 Doctorant @Télécom Paris Tech

> 2013 – 2015 Ingénieur R&D Moteur de recherche @Twenga



[www.twenga.fr](http://www.twenga.fr)

- > 500 Millions d'offres ecommerce
- > 15 langues
- > 10000 requêtes secondes
- > répondre en moins de 15 ms

# Un moteur de recherche !?

> Le search est partout !



Recherche Google

J'ai de la chance



TOUS NOS RAYONS

VENTES FLASH

HAPPY HOUR VINYLES : -20%  
AVEC LE CODE JUIN

VENTES FLASH APPLE

MacBook

Rayons ▾

Que recherchez-vous ?



BONS PLANS HIGH TECH -50%

2 LIVRES ACHETÉS = 1 LIVRE OFFERT

HAPPY WEEK-END

Accueil - Ventes Flash

**Livraison gra**  
(Hors marketplace)

{BnF} Bibliothèque  
nationale de France

Soutenez  
la BnF >

Recherche 

LA BNF

COLLECTIONS ET SERVICES

ÉVÉNEMENTS ET CULTURE

POUR LES PROFESSIONNELS

ACCÈS DÉDIÉS

# L'écosystème du Search en open source



- > Fondation Apache – Java
- > **Librairie** d'indexation
- > 2001



- > Fondation Apache – Java
- > **Serveur** de search
- > 2006



elastic

- > Entreprise – Java
- > **Serveur** de search ( et bien plus ... )
- > 2012
- > Valorisation à hauteur de 700 M\$

## Focus sur élasticsearch



**elastic**

- > suite de logiciels open source autour du search
- > stratégie basée sur la vente de service
- > stack orientée big data, logs utilisateurs, logs machines



**elasticsearch.**



**logstash**

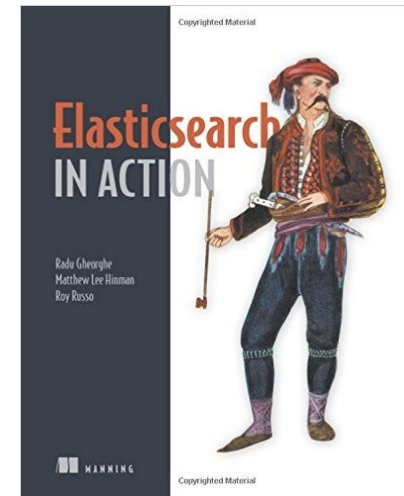
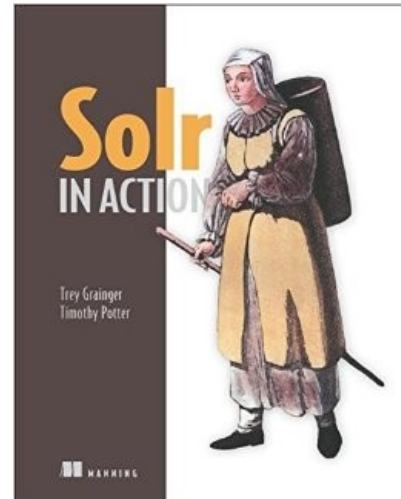
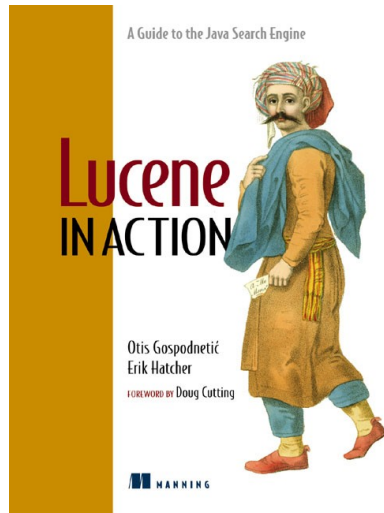
- > traitement des logs



- > visualisation on top of elasticsearch

beats, watcher, shield, marvel ...

## Quelques ressources



- > <https://www.elastic.co/blog>
- > <https://lucidworks.com/blog/>
- > <https://blog.algolia.com/>
- > blogs perso des committers Lucene

Algolia

> Solution propriétaire, en SAAS

exalead



altavista™

# Un peu de vocabulaire

- > **requête** : question posée par un humain ou une machine au moteur de recherche
- > **index** ( collection ) : structure de données et / ou ensemble de documents ( abus de langage )
- > **document** : objet texte à indexer ( page html, offre e-commerce, roman, log ... )
- > **champ** ( *field* ) : contenu ( texte, entier, booléen, binaire ... ) associé au document
- > **terme** ( token ) : élément de texte élémentaire indexé

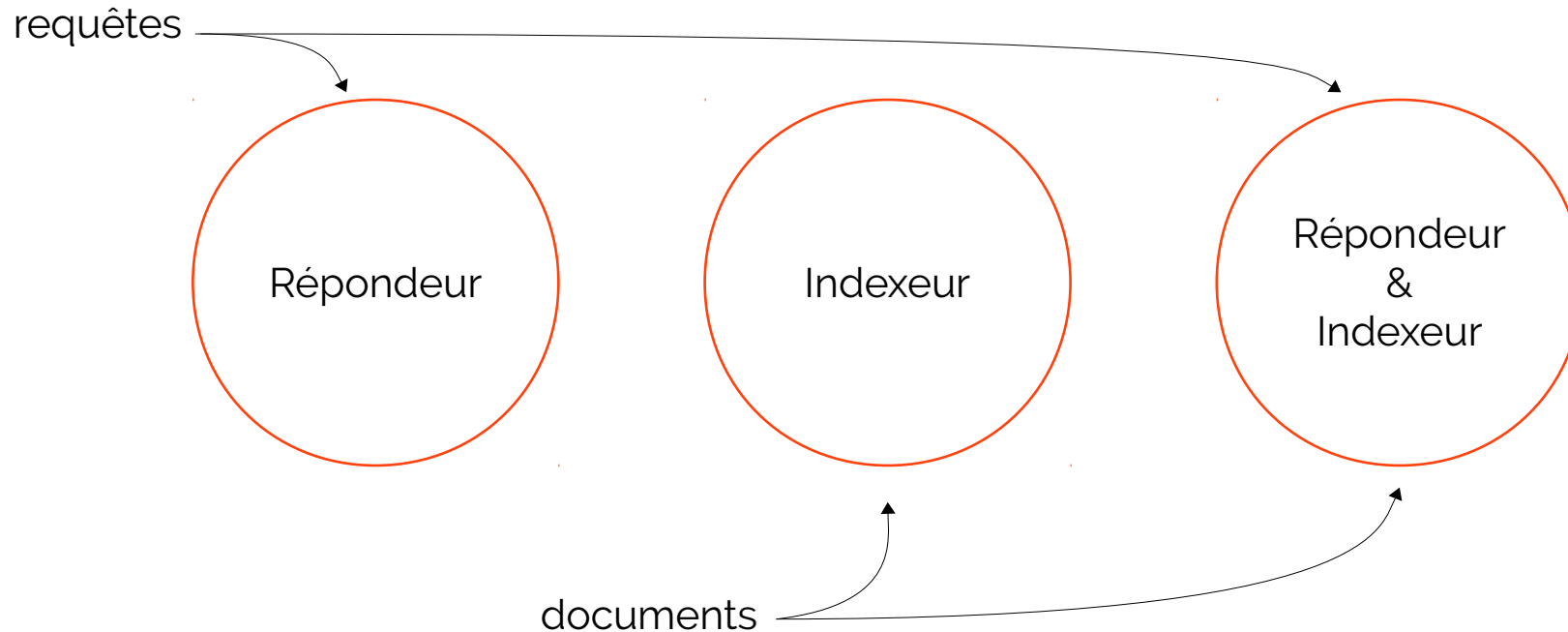
# **Configurer un serveur de search**

> Elasticsearch & Solr

> Rôles / schéma / analyseur / requêtes / indexation



# Comprendre les rôles d'un serveur de search



- > Indexer est coûteux en terme de cpu et de mémoire
- > Une application très sollicitée ( beaucoup de requêtes ) devra bien séparer les tâches
  - ex : un master indexe et les slaves répondent ( mettre en place une stratégie de mise à jour )

# Créer un schéma / un mapping

- > Le schéma décrit la structure du document à indexer
- > On y définit des champs et des comportements / actions associés à chacun d'eux
- > Le schéma doit être pensé en terme d'indexation et de réponse aux requêtes
- > Savoir construire un schéma demande beaucoup de pratique et des connaissances ad hoc

## > Solr

```
<field name="id" type="string" indexed="true" stored="true" multiValued="false" required="true" />  
<field name="active" type="boolean" indexed="true" stored="true" multiValued="false"/>  
<field name="date" type="date" indexed="true" stored="true" docValues="true" multiValued="false"/>
```

## > Elasticsearch

```
"id" : { "type" : "string", "index" : "analyzed", "analyzer" : "trans_standard" },  
"description" : { "type" : "string", "index" : "analyzed", "analyzer" : "trans_standard" },  
"created_at" : { "type" : "long", "index" : "not_analyzed" }, ( ... )
```

# Les analyseurs, les tokenizers et les filtres

- > Un analyseur **standardise le traitement** des textes à indexer et des requêtes utilisateurs
- > L'analyseur examine le texte d'un champ et génère un *token stream* sur lequel il applique et articule un ensemble de tokenizers et de filtres

Apache Lucene is a high-performance, full-featured text search engine library

- > Un tokenizer divise le texte en termes ( token ) en se basant sur les espaces, la ponctuation ...

- ex : whitespace tokenizer

Apache | Lucene | is | a | high-performance, | full-featured | text | search | engine | library

- Standard Tokenizer, Lower Case Tokenizer, N-Gram Tokenizer, Regular Expression Tokenizer, ...

- > Un filtre sélectionne, écarte, transforme ... certains termes du *token stream*

Apache | Lucene —> Lucene

- > L'analyser intervient à l'indexation et au traitement d'une requête

# Les requests handlers

- > Un request handler définit la logique et les traitements appliqués aux requêtes
- ajouter des paramètres à la requête, les modifier, en supprimer ...
- jouer plusieurs requêtes en parallèle
- modifier le format de retour de la réponse

`http://localhost:8800/solr/indexTest/select?q=toto&fq=type:website&fl=url`

# L'indexation

- > Plusieurs formats sont disponibles en entrée de l'indexation ( xml, csv ... )
- > L'indexation peut se connecter à d'autres sdb ( mongo, mysql, hadoop ... )
- > Comment rendre les documents rapidement *searchable* ?
  - Dupliquer les documents au besoin
  - L'indexation est fonction des requêtes de search, penser son index en conséquence
- > Troquer les performances de mise à jour contre une meilleure vitesse de search
  - Grep vs indexation full-text
  - Prefix queries vs edge n-grams
  - Phrase queries vs shingles
- > L'indexation est rapide
  - 710 GB/h pour 4k docs !
  - <http://people.apache.org/~mikemccand/lucenebench/indexing.html>
- > Des stratégie avancée d'indexations peuvent être mises en place mais cela demande Une bonne maîtrise de Lucene

# **Comment fonctionne Lucene ?**

- > Back to basic

- > Index / Insertion / Suppression

# Pourquoi s'intéresser à Lucene ?

- > Pour comprendre le **coût d'usage** des APIs ( éviter les boites noires )
  - pour construire des applications de Search rapides et performantes
  - pour savoir quand commiter son index
  - pour savoir si Lucene est le bon outil pour votre application
- > Pour comprendre l'**espace occupé** par un index
  - Pourquoi les *Term Vectors* représentent la moitié de la taille d'un index !?
  - J'ai supprimé 20 % de mes documents mais la taille de l'index n'a pas changée !?
- > Parce que c'est incroyablement intéressant ! ( si si )

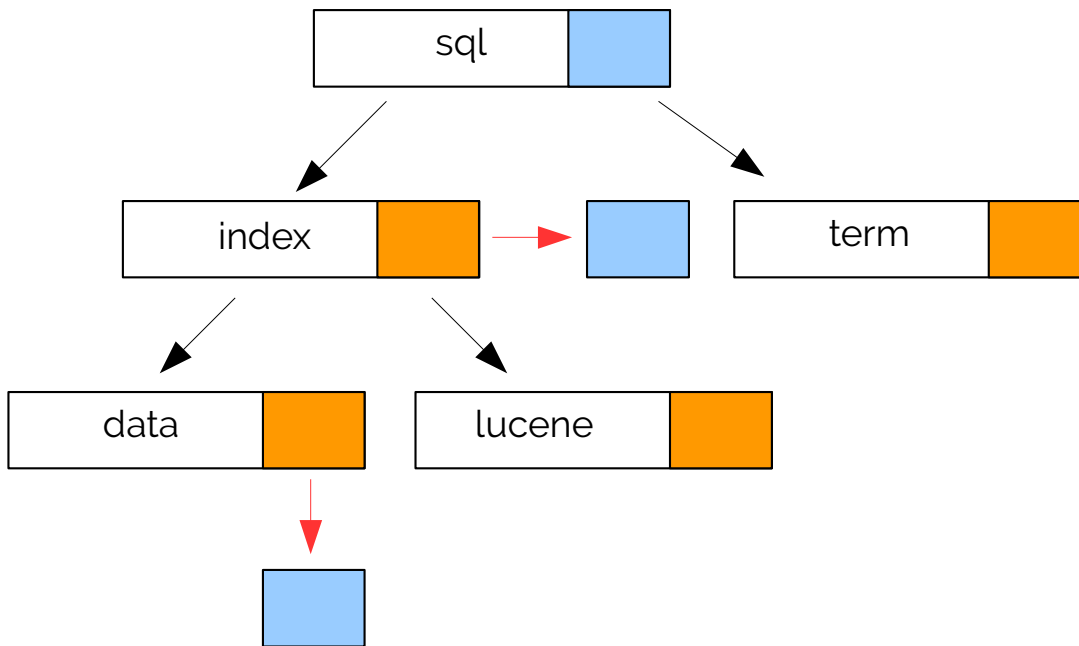
# Mon premier index naïf

> Hash table ? Pourquoi pas mais :

- Associer un document à un mot
- Ne fonctionne pas pour les range / prefix queries par exemple > l'index doit être trié ou triable

> Tree structure ( tree map ou B tree ), une structure d'index classique de sbd ( mysql, mongo ... )

- Trié en amont pour des *range queries*
- $O(\log(n))$  search



> Nous indexons deux livres

Lucene in action

Database



**Mais Lucene ne fonctionne pas sur ce modèle !**

> Découvrons la structure interne d'un index Lucene

# Index Lucene

> Stocker les terms et les documents dans un tableau

- *binary search*

0	data	0,1
1	index	0,1
2	lucene	0
3	term	0
4	sql	1

0	Lucene in action
1	Database

# Index Lucene

> Stocker les terms et les documents dans un tableau

- *binary search*

## > segment

0	data	0,1
1	index	0,1
2	lucene	0
3	term	0
4	sql	1

term  
ordinal

term  
dict

postings  
list

0	Lucene in action
1	Database

doc  
id

document

# Insertion

- > Insertion = enregistrer un nouveau segment
- > Merger les segments quand ils sont trop nombreux
  - Concaténer les docs, merger les terms dicts & postings lists ( *merge sort* )

0	data	0
1	index	0
2	lucene	0
3	term	0

0	Lucene in action
---	------------------

0	data	0
1	index	0
2	sql	0

0	Database
---	----------

0	data	0,1
1	index	0,1
2	lucene	0
3	term	0
4	sql	1

0	Lucene in action
1	Database

# Insertion

- > Insertion = enregistrer un nouveau segment
- > Merger les segments quand ils sont trop nombreux
  - Concaténer les docs, merger les terms dicts & postings lists ( *merge sort* )

0	data	0
1	index	0
2	lucene	0
3	term	0

0	Lucene in action
---	------------------

0	data	1
1	index	1
2	sql	1

1	Database
---	----------

0	data	0,1
1	index	0,1
2	lucene	0
3	term	0
4	sql	1

0	Lucene in action
1	Database

# Suppression

- > Suppression = changer un bit
- > Ignorer les documents supprimés pour le search et le merge
- > On va de préférence merger les segments avec beaucoup de suppressions

0	data	0,1
1	index	0,1
2	lucene	0
3	term	0
4	sql	1

0	Lucene in action	0
1	Database	1

live docs: 1 = live, 0 = deleted

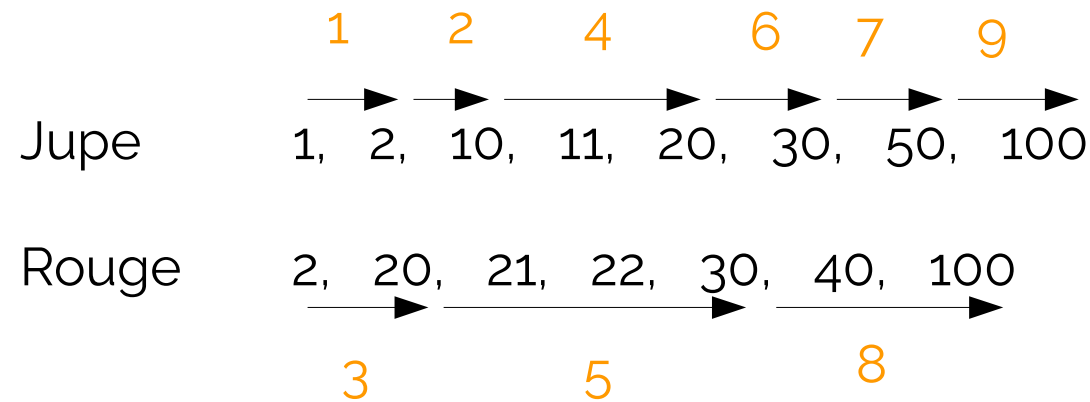
## Pour / Contre

- > La mise à jour implique l'écriture d'un nouveau segment
  - Mettre à jour un seul doc est coûteux, préférer une mise à jour par lots
  - L'écriture est séquentielle
  
- > Un segment n'est jamais réellement modifié sur le disque ( *immutable data* )
  - Filesystem-cache-friendly
  - Lock-free !
  
- > Les terms sont dédupliqués
  - Sauver de l'espace pour les terms de haute fréquence
  
- > Les docs sont uniquement identifiés par des *ordinal*
  - Pratique pour la communication cross API
  - Lucene peut interroger plusieurs indexes pour une même requête
  
- > Les terms sont uniquement identifiés par des *ordinal*
  - Important pour le tri : comparer des Long plutôt que des String
  - Important pour les facets

**➤ Lucene peut utiliser plusieurs indexes  
contrairement à d'autre bases de données !**



## Intersection entre 2 *postings lists*



- > la structure des postings lists permet de ne pas visiter l'ensemble des documents d'un index
- *leap-frog*

## **D'autres fonctionnalités ?**

- > Nous n'avons vu que le Search
- > Lucene permet d'aller plus loin encore
  - term vectors
  - norms
  - numeric doc values
  - binary doc values
  - sorted doc values
  - sorted set doc values

# Term Vectors ( une évolution de l'index inversé )

- > Index inversé par documents
- > Utilisé pour *more-like-this* search ( similarité entre deux documents )
- > Utilisé pour le *highlighting*

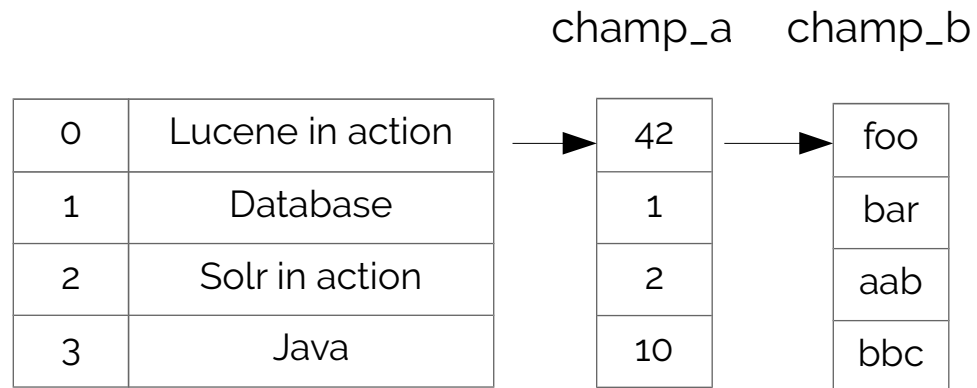
0	data	0,1
1	index	0,1
2	lucene	0
3	term	0
4	sql	1

		terms	freq	pos	offset
0	Lucene in action	data	1	2	[10 ,14]
		index	1	3	[15,20]
		lucene	2	1,4	...
		term	2	7,9	...
1	Database	data	1	1	...
		index	1	3	...
		sql	3	2,4	...

- > On peut facilement croisé l'index d'un document avec l'index de l'ensemble des documents  
Pour trouver des documents similaires

# Doc Values

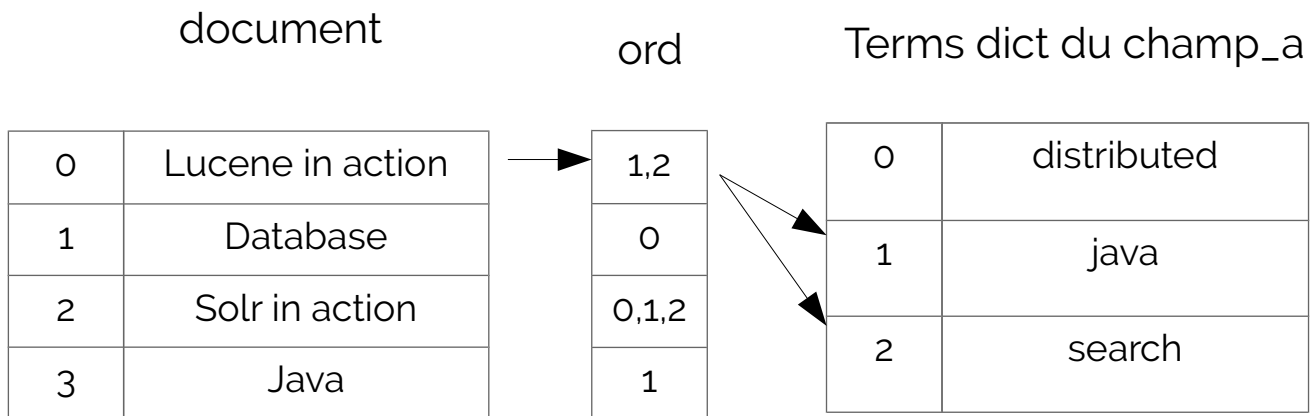
- > Associer un champ ( *field* ) à un document
- > Une valeur par document stockée sous forme de colonne
- > Numérique ou binaire
- > Utilisé pour le trie et le ranking ( score personnalisé )



# Sorted Doc Values

> Associer un champ à un document via un ordinal

- Champ monovalué : Améliore le trie
- Champ multivalué : Permet le faceting



# Faceting

> Recherche par dimension pour une requête ( calculé à la volée )

> Effectuer un comptage par dimension

- Ex : taille de vêtements ( M, L, XL ... ) dans un site ecommerce

> Solution

- D'abord agréger les *ordinals* des documents dans une hash table
- Puis retrouver les valeurs de la dimension grâce aux *ordinals* des documents
- Linéaire  $O(\#docs)$  en parcourant les ordinals
- Linéaire  $O(\#dim)$  en parcourant beaucoup moins de dimensions

| 0 : 2, 1 : 3, 2 : 2 | —————▶ | distributed : 2, java : 3, search : 2 |

> Solution très efficace car les valeurs d'une dimension sont souvent peu nombreuses

Bon ... mais concrètement ...

**Que se passe il lorsque l'on lance une requête ?**

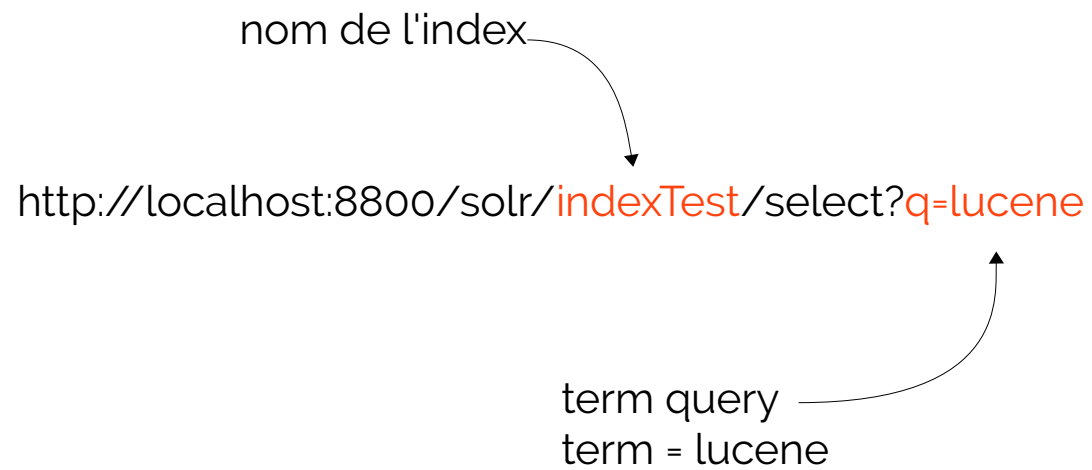
## Soit la requête suivante

> ici, jouée dans Solr

nom de l'index

<http://localhost:8800/solr/indexTest/select?q=lucene>

term query  
term = lucene





# De la requête aux documents

> 4 étapes pour 4 structures de données

1. Terms Index      —————> Retrouver le préfixe
2. Terms Dictionary      —————> Retrouver le terme
3. Postings Lists      —————> Retrouver les documents
4. Stored Fields      —————> Retrouver les champs ( le contenu )

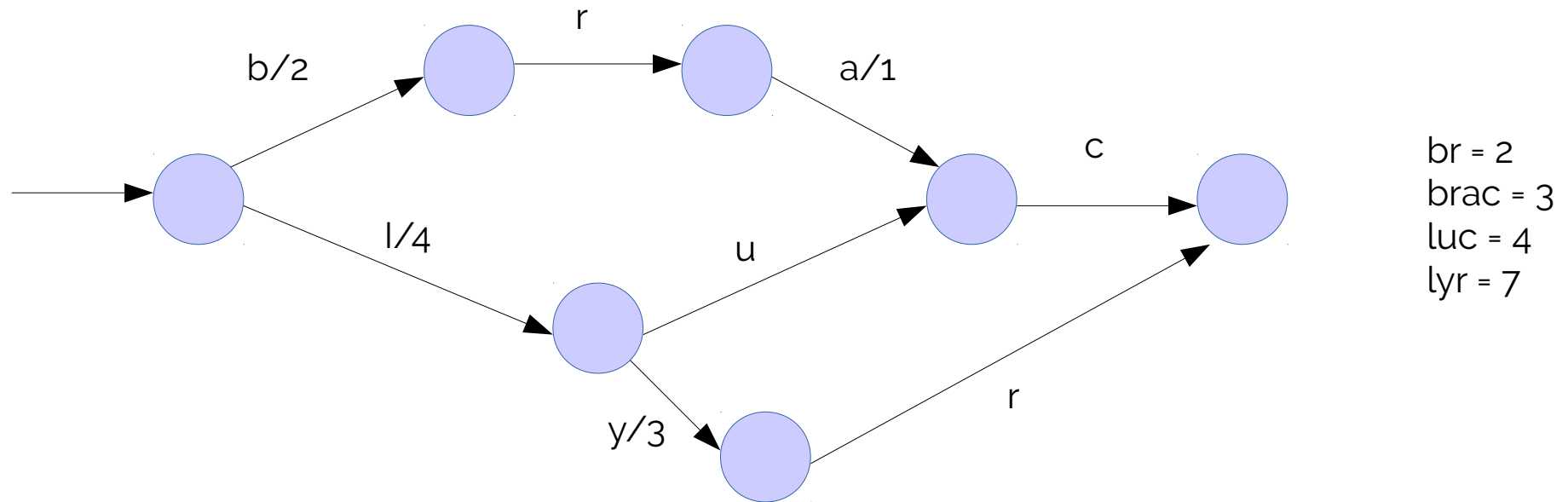
# Étape 1 : Terms index – partir des préfixes

> Parcourir chaque terme de l'index

- *In-memory FST storing* pour chaque préfixe
- Donne l'offset correspondant dans le terms dictionary
- Permet d'échouer rapidement si aucun terme n'a ce préfixe

> FST = finite state transducers

- Mapper chaque préfixe trié à son ordinal dans l'index
- Parcourir l'arbre nous donne l'ordinal



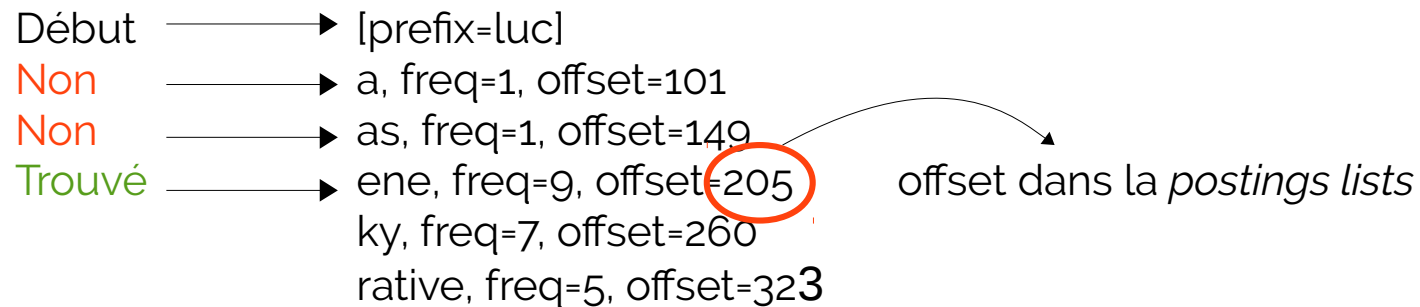
> pour q=lucene nous commencerons à chercher à l'ordinal 4 de l'index

## Étape 2 : Terms dictionary

> Se rendre à l'offset du *terms dictionary* obtenu à l'étape 1

- Compression basée sur les préfixes communs

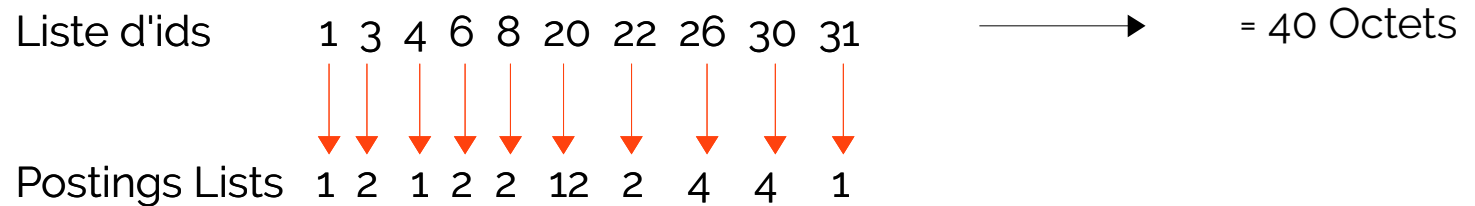
> Lire séquentiellement jusqu'à retrouver « lucene »



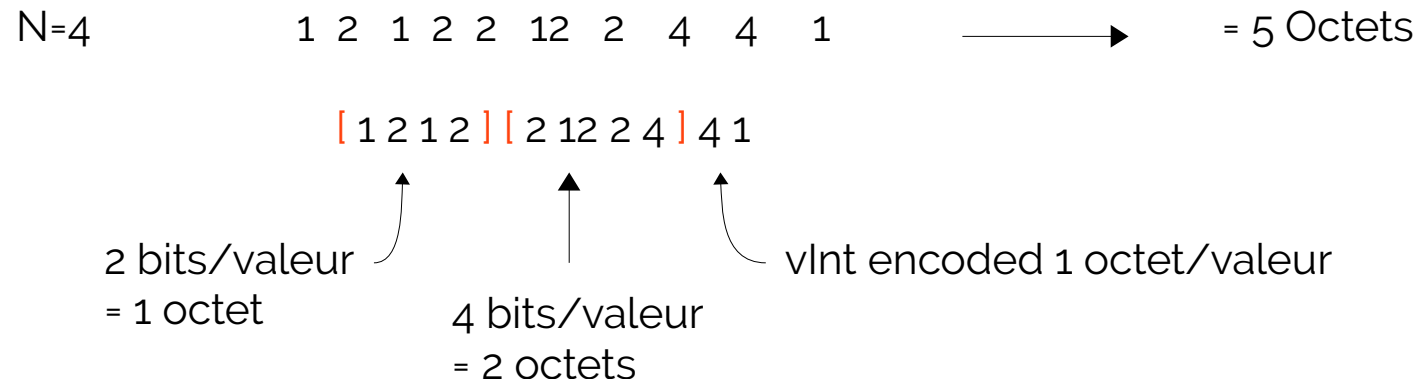
## Étape 3 : Postings Lists

> L'objectif ici est de retrouver la liste de tous les documents contenant le terme « lucene »

> Postings Lists = Liste triée de *docs ids* delta encodés



> Usuellement la *postings lists* est divisée en blocs fixes de N=128 valeurs

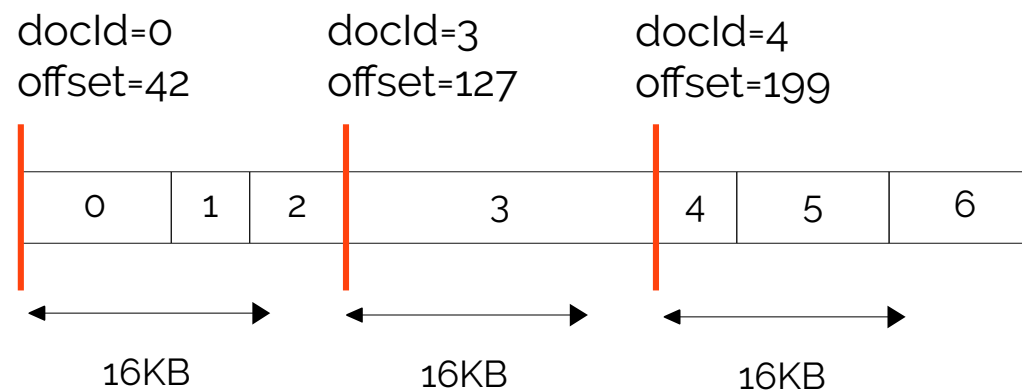


> Le nombre de bits alloués est fonction des valeurs ( + la valeur est petite – elle prend de place )

> vInt = code universel qui associe un nombre arbitraire d'octets à un entier ( cf format MIDI )

## Étape 4 : Stored fields

- > 2 fichiers : l'index ( *stored fields index* ) et les données ( *stored fields data* )
- > Les données sont stockées séquentiellement dans un unique fichier
  - d'abord en mémoire ( buffer ) puis sur disque tous les 16KB
  - Compression LZ4
- > Les *stored fields data* représentent 60 à 70 % de la taille de l'index
- > La recherche dans l'index est simple ( binary search ), par bloc & docId, en mémoire
- > Il faut décompresser chaque bloc puis un nouvel index nous donne l'offset de chaque document



# Le score lucene

> Comment sont ordonnés les documents résolvant ma requête ?

( normaliser le score  
pour le rendre comparable  
entre 2 requêtes  
ici c'est la distance euclidienne )

boost à l'indexation  
les champs les – volumineux  
contribuent plus

$$score(q, d) = coord(q, d) \times queryNorm(q) * \sum_{t \in q} (tf(t \in d) * idf(t)^2 * t.getBoost() * norm(t, d))$$

nombre de termes  
présents dans le doc

term freq

inverse doc freq

boost à la requête  
défaut = 1

> Un score personnalisé peut tout à fait le score lucene s'il n'est pas adapté à l'index et aux requêtes

Merci !