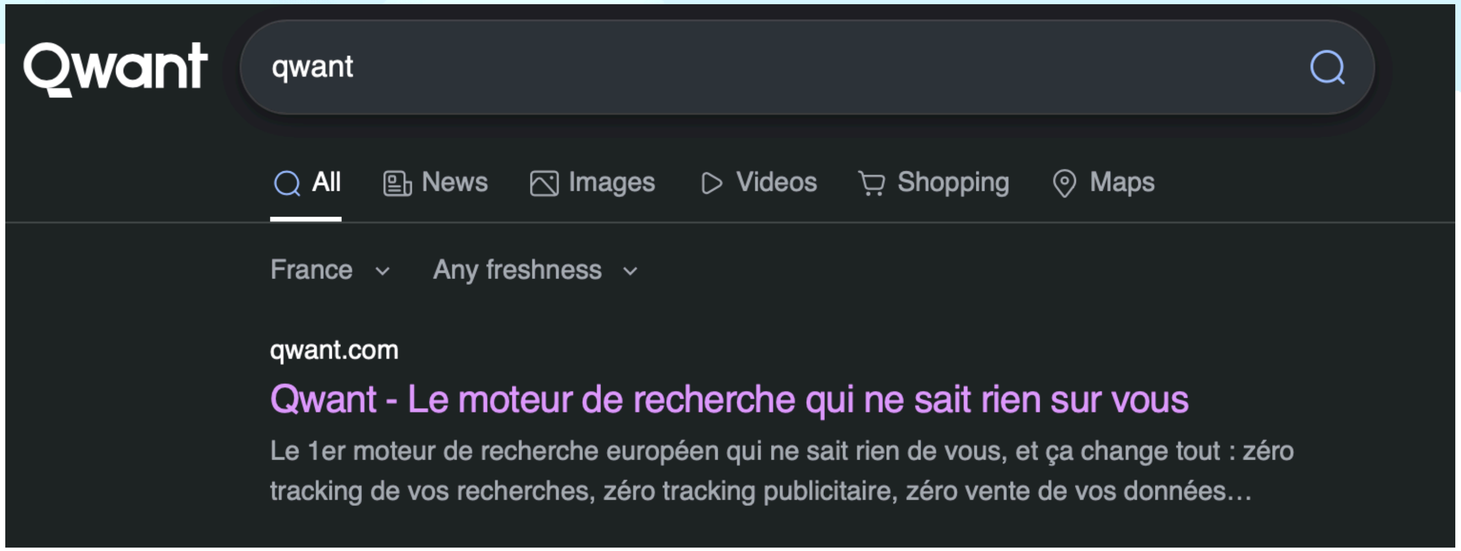
# Indexation Web

## Janvier 2023

# Qwant





- Lara Perinetti
  l.perinetti@protonmail.com
  Data Scientist NLP | IR

# Parlez-moi de vous

- Combien vous êtes dans chaque groupe ?
- Quels langages de programmation vous connaissez ?
- A quel point vous êtes familier avec les notions du web ? (HTML, robots.txt etc.)
- Avez-vous déjà étudié le NLP ? IR ?

# Cours Indexation Web

- 6h CM en commun
- 3 * 3h TP
  - Crawl
  - TF-IDF
  - Query Expansion

# Cours Indexation Web

## Rendus des TPs

- A chaque fin de TP il faudra m'envoyer le code et vous aurez une semaine de plus si vous n'avez pas fini pour me (re) envoyer le code amélioré Vous aurez une semaine de plus pour m'envoyer la version définitive

- TP Crawler - Fonctionnaires 10/01 -> 17/01
  - Ingénieurs 19/01 -> 26/01

- TP Index - Fonctionnaires 20/01 -> 27/01
  - Ingénieurs 23/01 -> 02/02

- TP Query - Fonctionnaires 24/01 -> 03/02
  - Ingénieurs 24/01 -> 09/02

# Cours Indexation Web

- Notation
  - Il y aura un mini projet + des features à faire en plus en points bonus
  - Une partie de la note du TP sera sur les commentaires et la simplicité de compréhension et de lancement du code
  - Il faudra toujours m’envoyer votre code avec un fichier main.py et un README.md avec comment lancer le programme + les paramètres par défaut

- Discord du cours: https://discord.gg/nPrvscBV

# Table of content

Partie 1: Constitution d’un index web

1. Architecture d’un moteur de recherche
2. Crawler le web
3. Document Processing

Partie 2: Traitement de la requête et ordonnancement des résultats

1. Index
2. Compréhension de la requête
3. Ordonnancement des résultats et évaluation

# Qu’est-ce qu’un moteur de recherche?

## Differents types

- **Web Search**: Google, Qwant, DuckDuckGo, Ecosia etc.
  - Differents formats: HTML, pdf, docx etc.
  - Champs les plus communs: title, content, page rank
- **Shopping Search**: Amazon, Cdiscount, La Redoute etc.
  - Leurs propres structures de données, appliquer des filtres
  - Champs les plus communs: nom du produit, prix, image
- **Desktop search**: sur les ordinateurs personnels
  - fichiers, emails, pages webs dans l’historiques des recherches
  - Séparer les sources de données

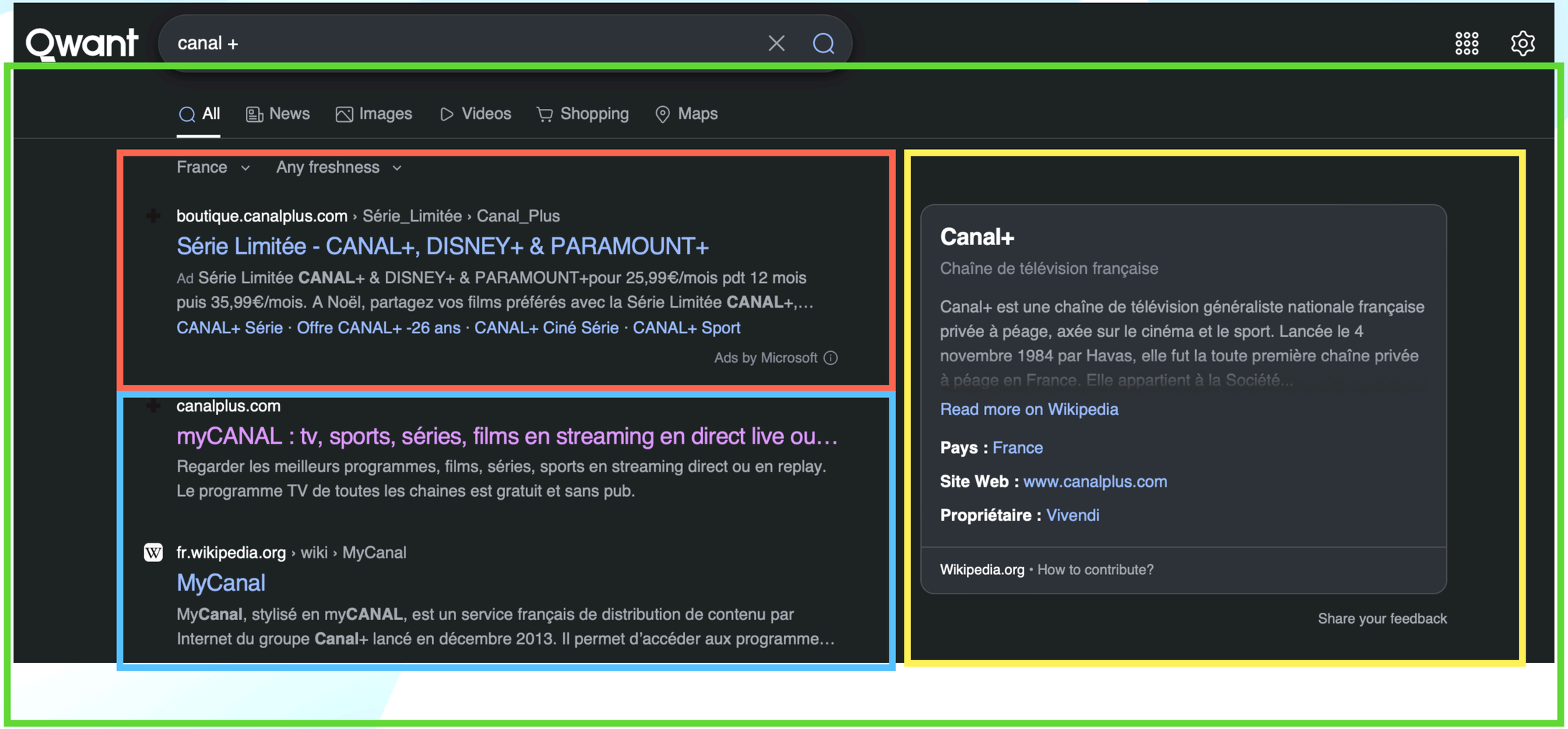
# Qu’est-ce qu’un moteur de recherche?

SERP Search Engine Results Pages

Paid Results

Organic Results

Search Engine Features



# Information Retrieval

- *"Information retrieval is a field concerned with the structure, analysis, organization, storage, searching, and retrieval of information."* (Salton, 1968)
- La recherche d'information consiste à trouver des documents non structurés qui répondent à un besoin d'information dans une grande collection de documents.
- Recommandation systems (Netflix, Spotify) —> ne seront pas traités dans ce cours
- Deux problématiques majeures
  - De nombreux facteurs influencent la décision d'une personne concernant ce qui est pertinent —> évaluation subjective
  - Les requêtes par mot-clé sont souvent de mauvaises descriptions des besoins réels

# Information Retrieval

## Qu’est-ce qu’un document web?

- On va se concentrer sur les pages HTML
  Balises les plus communes:

```
<head>
    <title></title>
    <link></link>
</head>
<body>
    <p></p>
    <div></div>
    <h1></h1>
</body>
```

Exemple

- La structure et le nom des balises sont reglementés
- Peuvent contenir du texte, des films, des images, ou tout à la fois
- Dans le web, les pages peuvent avoir plusieurs sujets/topics que l’on va essayer d’extraire
- Sont augmentées de métadonnées
  -> Les métadonnées sont des informations sur un document qui ne font pas partie du contenu du texte, comme le type de document

# Information Retrieval

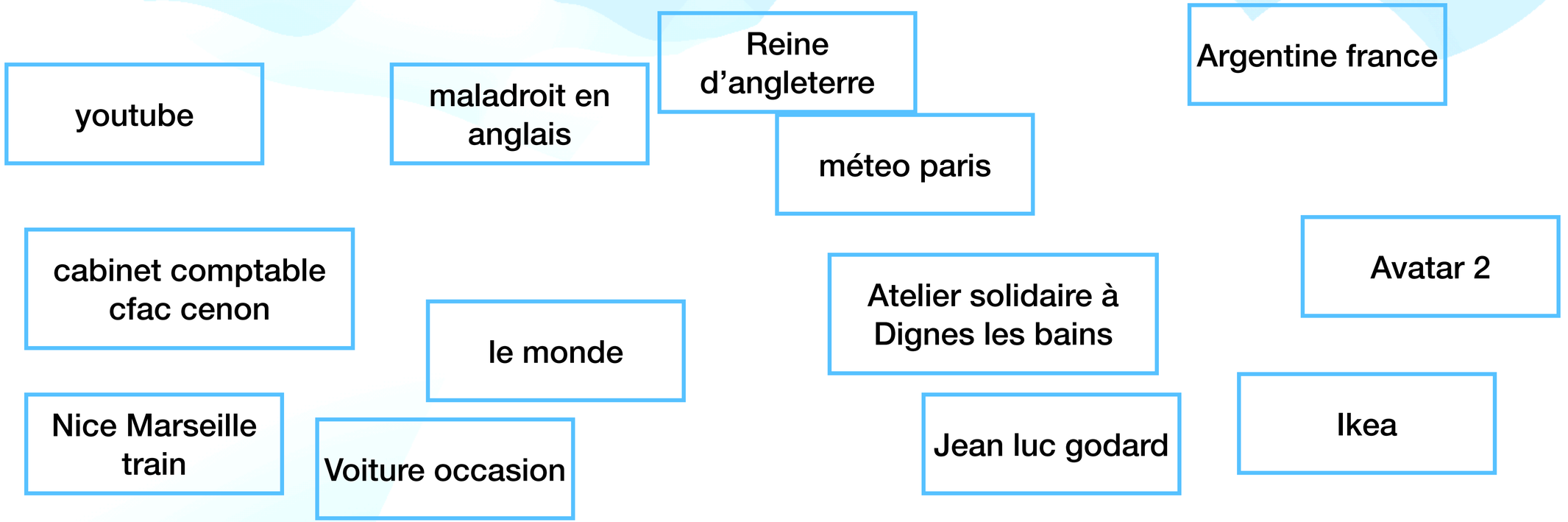
## Qu’est-ce qu’une requête web ?

- Spécificités d’une requête web: souvent courte (un ou deux termes) et ambiguë
- On définit 3 intentions principales pour une requête web
  - Navigationelle
  - Transactionnelle
  - Informationnelle
- On peut définir plusieurs topics pour une requête web
  - Exemple:
    requête: Jaguar
    topics: animal / voiture

# Information Retrieval

## What is a web query ?

Exemples de requetes d’utilisateurs Qwant en décembre 2022

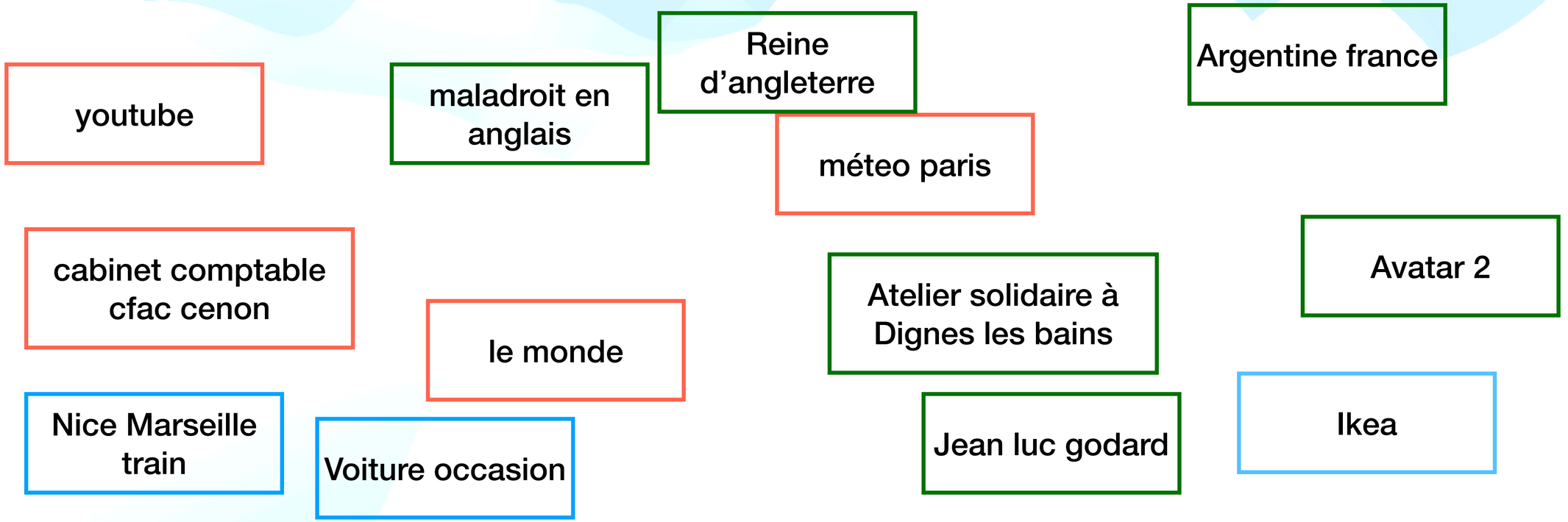



Les classer par intention (informationnelle, transactionnelle, navigationnelle)

# Information Retrieval

## What is a web query ?

Requetes d’utilisateurs Qwant en décembre 2022

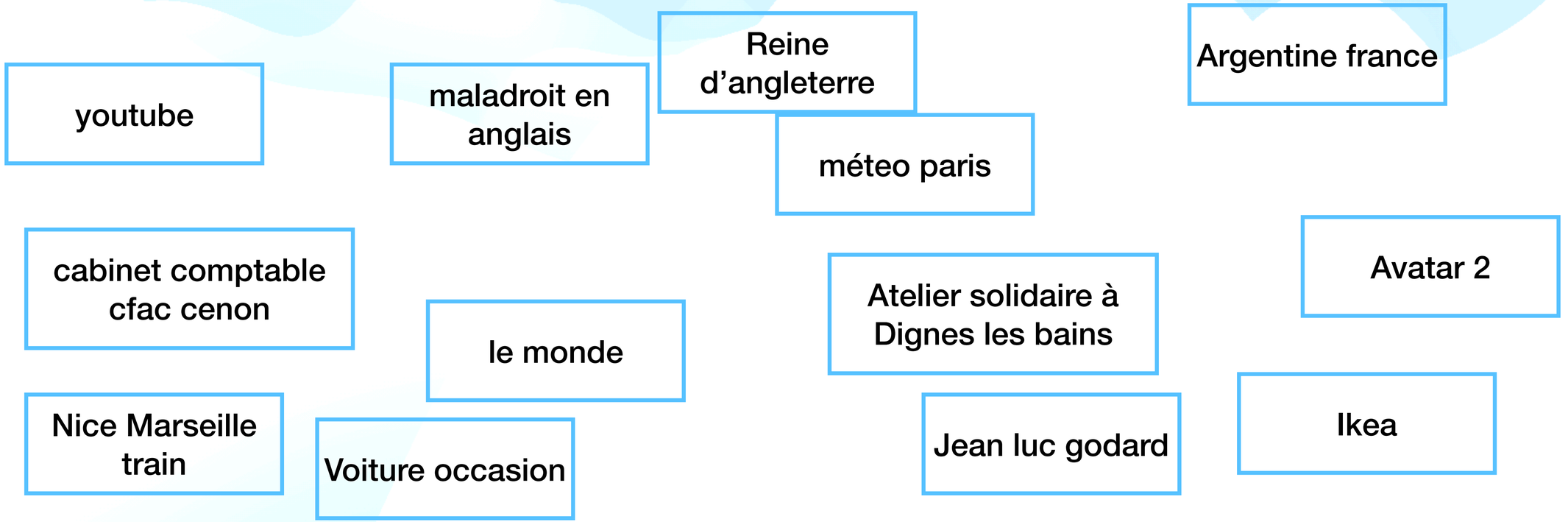



Les classer par intention (informationnelle, transactionnelle, navigationnelle)

# Information Retrieval

## What is a web query ?

Requêtes d’utilisateurs Qwant en décembre 2022

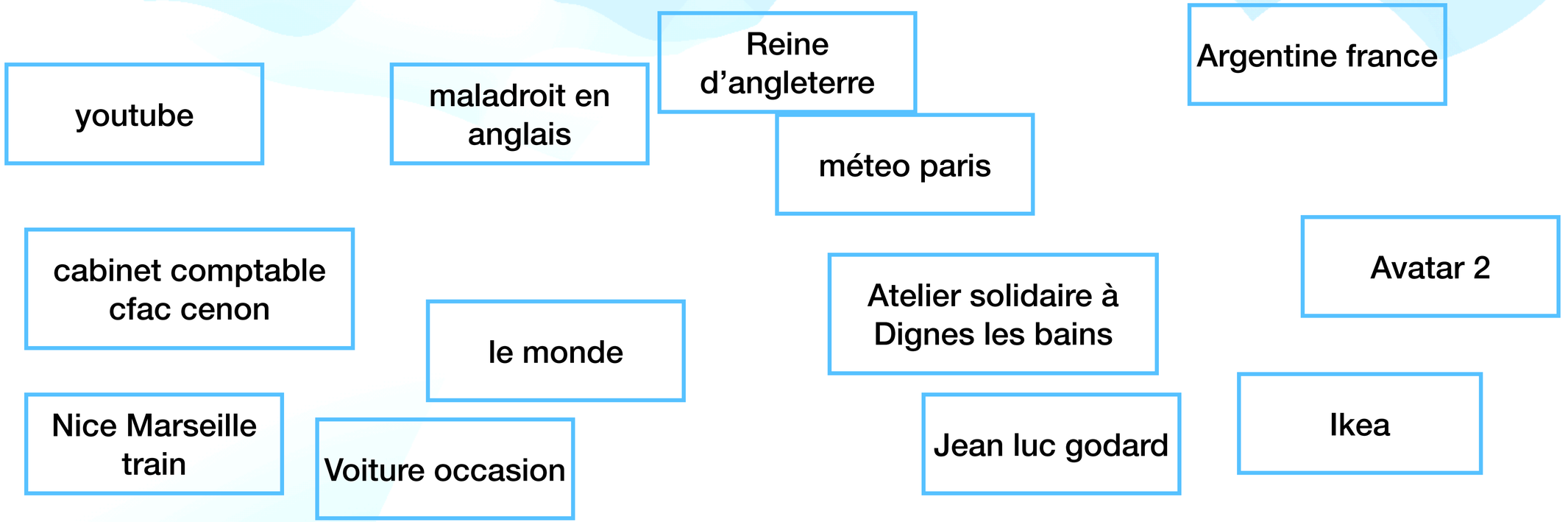



Les classer par topics (pas plus de deux topics par requête)

# Information Retrieval

## What is a web query ?

Requêtes d’utilisateurs Qwant en décembre 2022




shopping, news, entertainment, sports, etc.

# Partie 1
# Constitution d’un index web

## Architecture d’un moteur de recherche

# Architecture d’un moteur de recherche

## Objectifs

- Qualité : Nous voulons être en mesure de récupérer l'ensemble de documents les plus pertinents possibles pour une requête donnée.
- Rapidité : Nous voulons traiter les requêtes des utilisateurs aussi rapidement que possible.

# Architecture d’un moteur de recherche

## Des pages webs à l’index

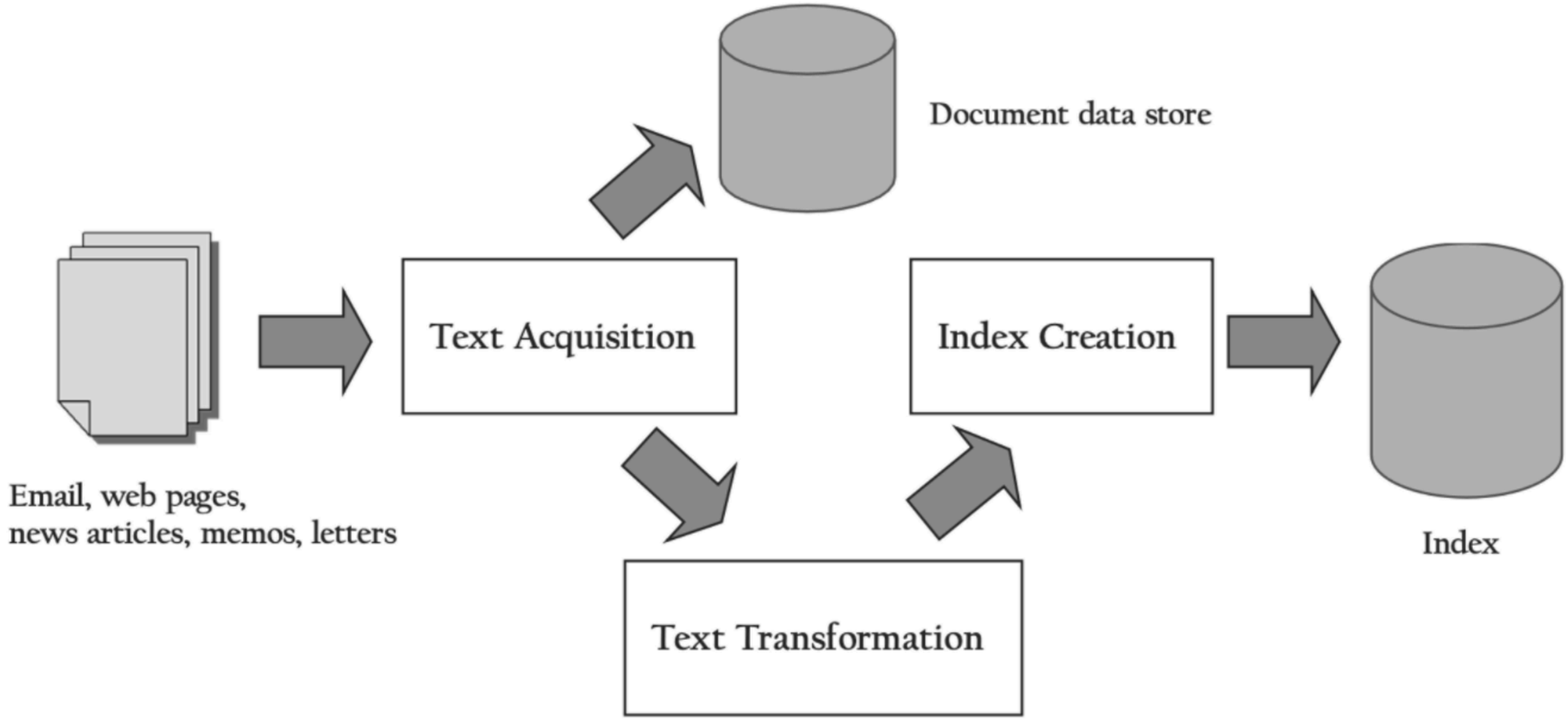



**Fig. 2.1.** The indexing process

# Des pages webs à l’index
## Text Acquisition

- identifie et rend disponible les documents que l’on voudra chercher

1. Quels documents ?
    - partir d’une collection de documents clés en main
    - chercher ces documents via une tache de **crawl**
2. Où les stocker?
   Une **base de données relationnelle** peut être utilisée pour stocker les documents et leurs métadonnées.
3. La notion de crawler
    - Un crawler est conçu pour suivre les liens sur les pages web pour découvrir et télécharger de nouvelles pages
    - Un bon crawler doit être capable
      de gérer efficacement l'énorme volume de nouvelles pages sur le Web,
      tout en veillant à ce que les pages qui ont pu être modifiées depuis la dernière visite d'un crawler restent "fraîches" pour le moteur de recherche

# Des pages webs à l’index

## Text Transformation

- Parsing —> hierarchy extraction
- Stopping —> removing stop words
- Stemming / Lemmatization
- Link extraction / analysis
- Information extraction (language, topic etc.)

# Des pages webs à l’index

## Index creation

- Prend la sortie de la transformation de texte pour créer les index qui contiendront les informations sur les documents
- Compte tenu du grand nombre de documents, la création d'un index doit être efficace, tant en termes de temps que d'espace.

# Architecture d’un moteur de recherche

## De la requête aux résultats finaux

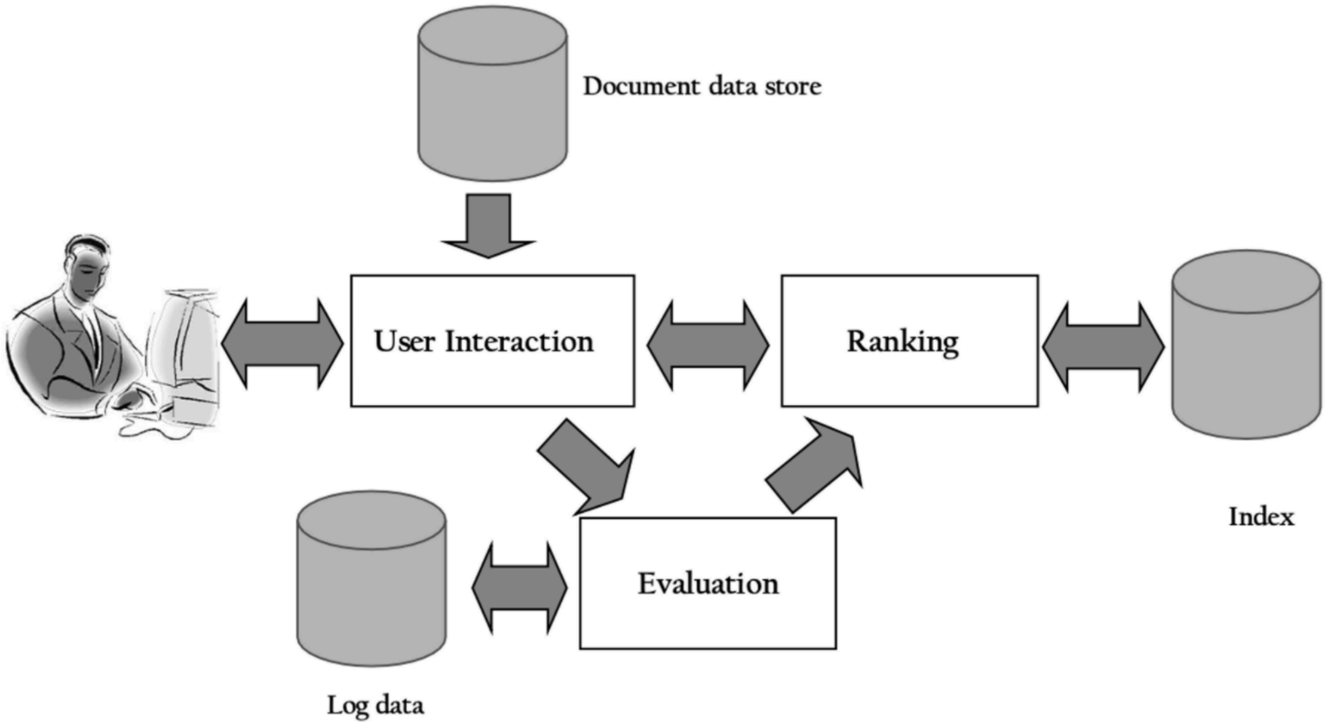



**Fig. 2.2.** The query process

# De la requête aux résultats finaux

## User interaction

- Reçoit en input la requête de l’utilisateur et la transformer pour qu’elle soit comprise par l’index
- La requête de l’utilisateur peut comprendre le texte meme de la requête mais aussi toutes les métadonnées associées (langue de l’utilisateur, topics de la requête, heure, jour de la semaine etc.)
- Inclut la transformation de la requête
  - Spell checking
  - Augmentation de la requête
  -

# De la requête aux résultats finaux

## Ranking

- Reçoit en input les documents qui ont survécu au filtre de la requête et leur donne un score en fonction de la requête.
- L’ordonnancement doit être à la fois efficace, puisque de nombreuses requêtes doivent être traitées en peu de temps, et effectif, puisque la qualité du classement détermine si le moteur de recherche atteint l'objectif de trouver des informations pertinentes.

# De la requête aux résultats finaux

## Evaluation

- Evaluation “offline”
- Evaluation “online”

# Partie 1
# Constitution d’un index web

## Crawler le web

# Crawler le web

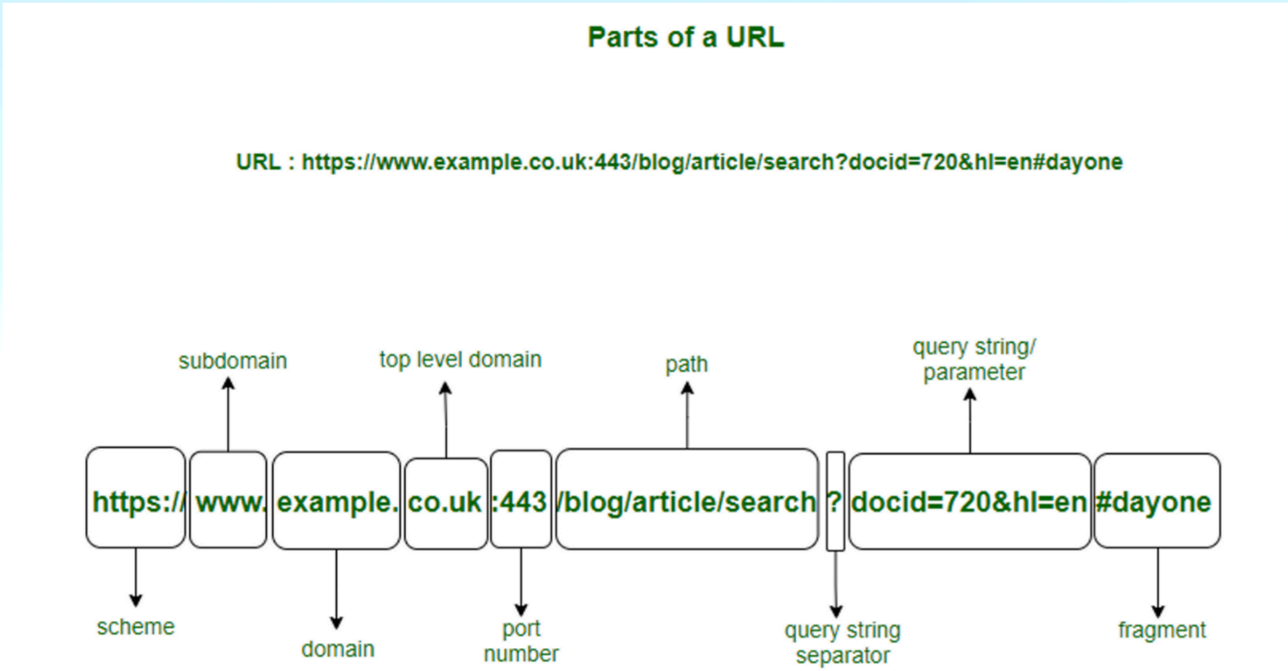
## Décider de ce qu'il faut rechercher

- Le Web est immense et en constante expansion
- Les pages Web changent constamment
  Même les documents utiles peuvent devenir moins utiles avec le temps.
- On ne peut pas se permettre de stocker toutes les pages du web
- On évalue un crawler par la **couverture (coverage)** qui mesure la quantité de documents pertinents que l'on réussit à récupérer
  Et par la **fraicheur (freshness)** des pages stockées.
- Un crawler se doit d'être poli, il se doit d'espacer les crawl pour une url donnée

# Crawler le web

## Qu’est-ce qu’une page web ?

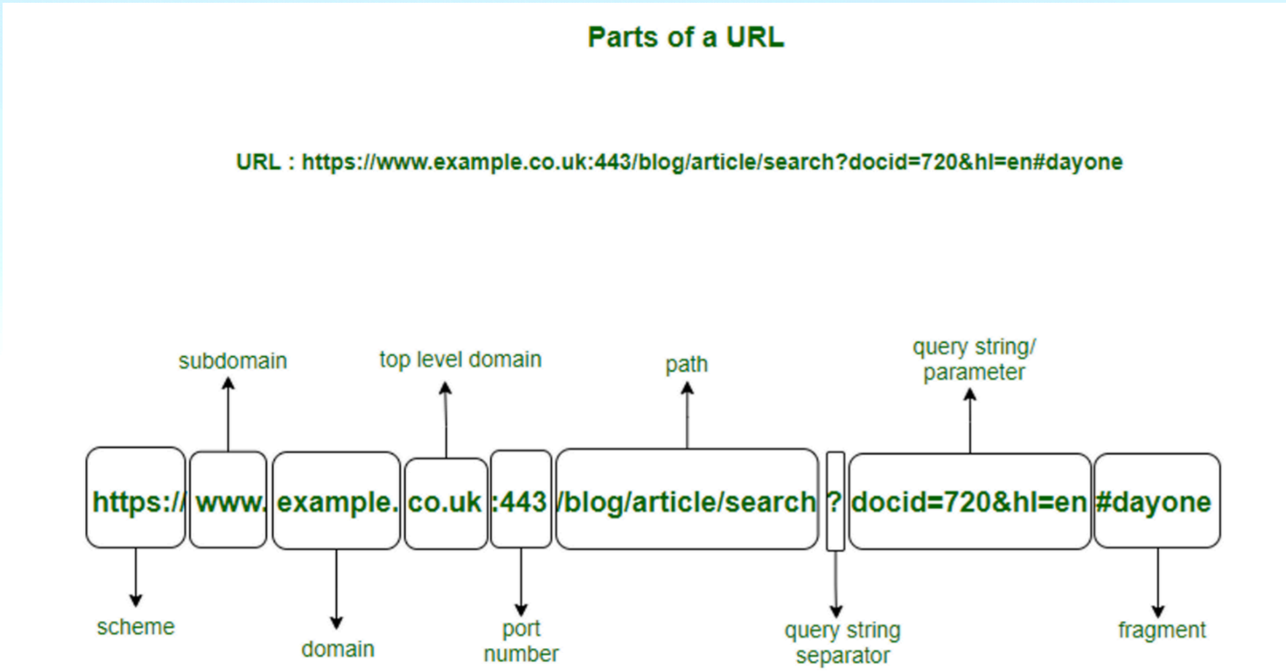
- Scheme
  http | https | ftp | smtp
- Subdomain
  indique le type de resource
  www | blog | audio etc.
- Domain
  indique l’organisation
- TLD
  indique le type d'organisation à laquelle le site web est enregistré.
  pays: fr | co.uk | de
  region: re
  autre: com | org | net



# Crawler le web

## Qu’est-ce qu’une page web ?

- Path
  indique la localisation de la page dans son domaine
- Query
  les paramètres d'un site web.
- Fragment
  pour arriver directement à une certaine balise dans le document html



# Crawler le web

## Algorithme

- Input: un ensemble d’urls
- Les urls sont ajoutées à une **file d’attente (ou frontier)** de requêtes URL
- Le crawler commence à récupérer les pages de la file d'attente des requêtes
- Les pages téléchargées sont analysées pour trouver des balises de liens qui pourraient contenir d'autres URL utiles à récupérer.
- Si le crawler trouve une nouvelle URL qu'il n'a pas vue auparavant, elle est ajoutée à la frontier.
- Le robot continue jusqu'à ce qu'il n'y ait plus de nouvelles URL à ajouter à la file d’attente ou que le disque soit plein.

- La frontier peut être une file d'attente standard ou être ordonnée de manière à ce que les pages importantes soient placées en tête de liste.

# Crawler le web

## Controler le crawler: robots.txt

- Les propriétaires de sites Web utilisent le fichier /robots.txt pour donner des instructions sur leur site aux crawlers
- Exemples de robots.txt
  - Site important
  - Site de niche
- Guidelines de google
- Informations sur le crawler de Qwant

# Crawler le web

## La notion de fraicheur

- Il n'est pas possible de vérifier constamment la fraicheur de toutes les pages
- On choisit donc de vérifier les pages importantes et celles qui changent fréquemment
- La fraîcheur est la proportion des pages explorées qui sont actuellement fraîches.
  Pb: si une page se met à jour toutes les 2 secondes

# Crawler le web

## La notion de fraicheur

- Il n'est pas possible de vérifier constamment la fraicheur de toutes les pages
- On choisit donc de vérifier les pages importantes et celles qui changent fréquemment
- La **fraîcheur** est la proportion des pages explorées qui sont actuellement fraîches.
  Pb: si une page se met à jour toutes les 2 secondes
  - La meilleure stratégie est alors d’arrêter de la crawler
- C’est pour cela que l’on va plutôt vouloir calculer l’âge d’une page

# Crawler le web

## La notion de fraicheur

- Une page est **fraîche** si elle représente la copie la plus récente d'une page Web, et périmée dans le cas contraire.
  La fraîcheur est la fraction des pages crawlées qui sont actuellement fraîches.

- Les pages deviennent tout de suite fraiches quand elles sont crawlées, mais dès que la page change, la page explorée devient périmée.

- La page a un **âge** de 0 jusqu'à ce qu'elle soit modifiée, puis son âge augmente jusqu'à ce que la page soit à nouveau explorée.

- Optimiser l’âge ne nous fera jamais dire qu’il vaut mieux arrêter de crawler une page

# Crawler le web

## Sitemaps

- Certains sites web exposent une sitemap aux crawler dans le robots.txt
- Exemple
- Quel est l’interet de créer cette sitemap pour l’administrateur du site?

# Crawler le web

## Sitemaps

- Certains sites web exposent une **sitemap** aux crawler dans le robots.txt
- Exemple
- Quel est l’interet de créer cette sitemap pour l’administrateur du site?
  - Il indique aux moteurs de recherche des pages qu'ils ne trouveraient pas autrement.
  - Cela permet de réduire le nombre de requêtes que le crawler envoie à un site web sans sacrifier la fraîcheur des pages.

# Crawler le web

## Stocker les documents

- La recherche de documents est couteuse en termes de CPU et de réseau.
- On va vouloir conserver une copie des documents déjà crawlés au lieu d'essayer de les récupérer à chaque fois qu’on lance un tour de crawl.
- En général on utilise une BDD relationnelle avec pour identifiant unique du document un hash de l’URL
- Exemple de BDD: Big Table

# Crawler le web

## Détecter les documents dupliqués

- Tache plutôt simple
- On va souvent utiliser des techniques qui calculent les bytes des documents
- Baseline: **checksum technique**:
  T r o p i c a l f i s h
  54 72 6F 70 69 63 61 6C 20 66 69 73 68 result = 508
- **cyclic redundancy check**, une autre technique qui prend en compte la position des bytes.
- Attention: **http://monsite.com** != **https://monsite.com**

# Crawler le web

## Détecter les documents “presque” dupliqués (ou near duplicates)

- Tache plus complexe
- Pour une url donnée, on va chercher à trouver tous ses “presque” duplicats dans la BDD.
- Pour cela, il faut représenter les documents
  - Soit en le hashant (e.g: sim-hash)
  - Soit avec des embeddings (vecteur dense de représentation d’un document)

# Crawler le web

## Le deep web

- Le deep web représente toutes les pages web qui sont compliquées à trouver pour un crawler et qui sont bien souvent non indexées par les moteurs de recherche
- Ces urls sont bien plus nombreuses que les pages web conventionnelles
  Et elles sont légales
- Contrairement aux pages du dark web

# Partie 1
# Constitution d’un index web

Document Processing

# Document Processing

## Extraire des informations des documents

- Avant d’ajouter nos documents dans un index, on va vouloir en extraire le plus d’informations possibles qui nous aideront à répondre de la manière la plus pertinente
  - Text Processing
  - Structure du document
  - Analyse des liens
  - Document embeddings

# Text processing

# Document Processing

## Text Processing | Vocabulaire

- Quelques mots apparaissent très souvent, beaucoup de mots n'apparaissent presque jamais.
- La taille du vocabulaire augmente avec le corpus
  Mais il y a de moins en moins de mots nouveaux lorsque le corpus est déjà important
- Le vocabulaire est constitué de
  - mots bien orthographiés (dans différentes langues)
  - mots avec fautes d’orthographe
  - mots inventés (e.g: noms de produits, de sociétés, etc.)
  - adresses mails, suites de chiffres
  - etc.

# Document Processing

## Text Processing | Des mots aux tokens

- On ne va plus parler de mots mais de token
- On va subdiviser le texte en plusieurs tokens
- Une tokenization par langue:
  - En anglais ou en français, plutôt simple:
    “The apple is red” —> [“the”, “apple”, “is”, “red”]
  - Les langues CJK ont une tokenization adapté aux signes.
- Il doit y avoir la meme tokenization du côté du document que du côté de la requête
- Une étape clé dans le text processing, si la tokenization est mal faite, les étapes d’après vont en pâtir. Elle est tellement important qu’aujourd’hui on utilise souvent un modèle entrainé pour tokenizer.
- On parle aussi de tokenization pour diviser les documents en phrases, en characters, en morphemes ou “word-pieces”

# Document Processing

## Text Processing | Stemming et lemmatisation

- En fonction des langues, on peut avoir besoin de faire référence à une représentation simplifiée des tokens, pour des questions de match avec la requête ou de taille du vocabulaire

- Documents: “nous allons à la boulangerie”

| Token original | Token stemmé | Token Lemmatisé |
|---|---|---|
| allons | all | aller |

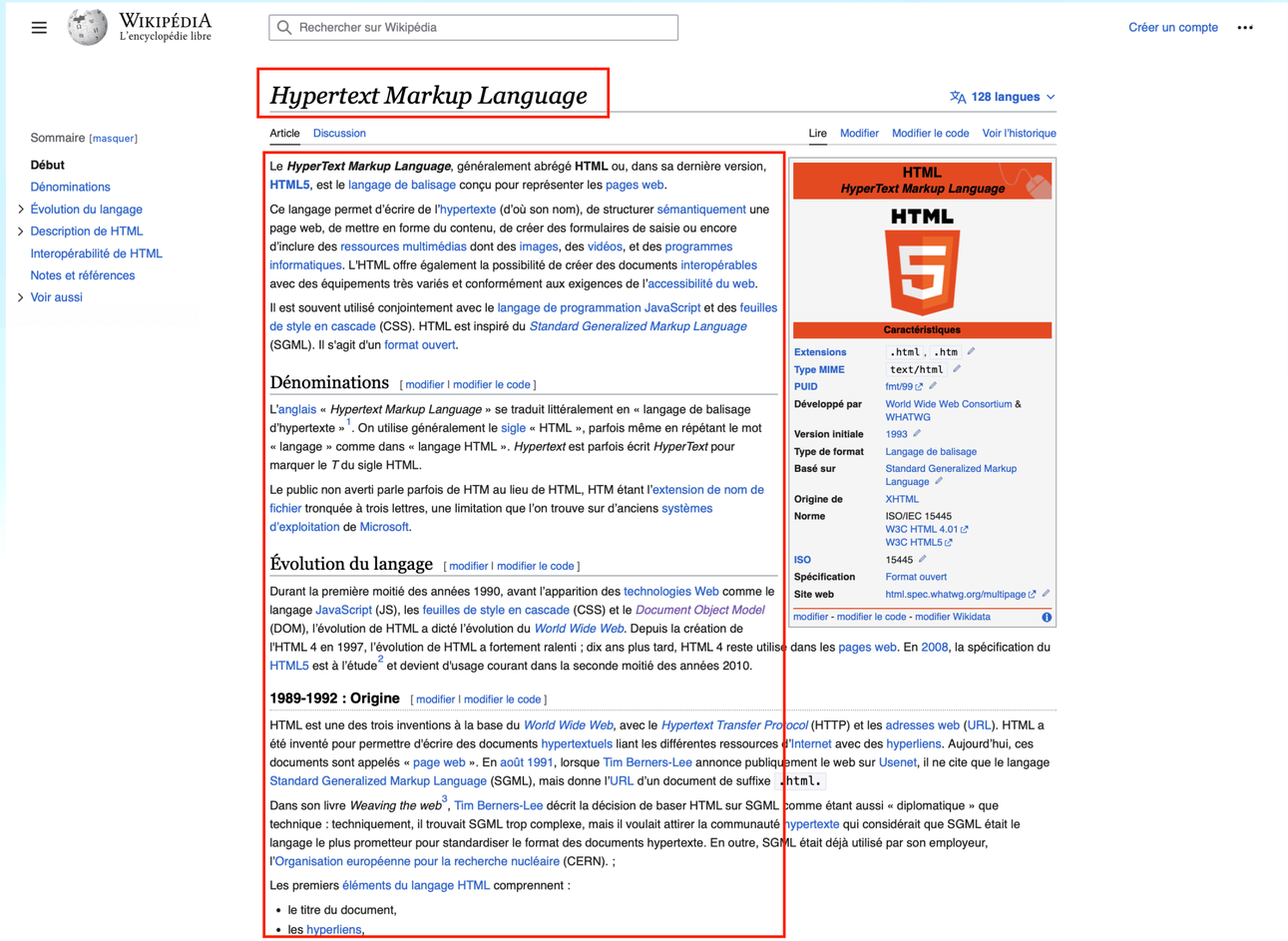
- En fonction des langues peut etre plus ou moins intéressant:
  - Les langues CKJ n’ont pas de variation morphologique
  - Les langues slaves ont parfois plus de 7 déclinaisons possibles pour un meme nom commun

# Structure du document

# Document Processing

**Document Structure | Supprimer le bruit contenu dans les documents**

- Tache: Boilerplate removal
- De nombreuses pages web contiennent du texte, des liens et des images qui ne sont pas directement liés au contenu principal de la page.
- Ces éléments supplémentaires sont pour la plupart du bruit et peuvent avoir un impact négatif sur le classement de la page.



# Document Processing

## Document Structure | Extraire d’autres informations que le contenu principal

- Interet de l’HTML: le nom des balises est codifié.
- Il y a certaines balises qui nous aident à extraire des informations:
  - <meta> pour les métadonnées
  - <img> pour les images
  - <title> pour les titres
  - <a> pour les ancres

# Analyse des liens

# Document Processing

## Analyse des liens

- Le crawler s’aide des liens entre les pages pour trouver de nouvelles pages à crawler
- Ces liens nous permettent
  - de créer un graphe du web
  - de comprendre les relations entre les pages
  - d’ordonner les pages en fonction de leur importance
- Exemple

# Document Processing

## Analyse des liens | Le texte des ancres

- En général, un texte court qui décrit la page vers laquelle on va cliquer
  Il peut avoir les memes caractéristiques qu'une requête web

- Extraire le texte des ancres nous permet ensuite de matcher ce texte avec la requête

- Peut etre très utile pour les abréviations:
  YouTube.com -> yt, ytb

- Mais doit etre nettoyé:
  "Cliquez ici", "check here", "découvrir"

# Document Processing

## Analyse des liens | La popularité d’une page

- La majeure partie du web est composée de pages assez peu interessantes, datées, ou de pages que l’on qualifie de spam
- On va donner un ou plusieurs score de qualité, de confiance aux urls
- Le plus connu: Page Rank

# Document Processing

## Analyse des liens | Page rank

- Algorithme qui calcule un score de popularité pour une url donnée
- Idée:
  - Plus il y a d'urls qui pointent vers une url A, plus A est populaire
  - Tous les liens ne comptent pas de la meme manière:
    Si toutes les urls qui pointent vers A ne sont pas populaires, la popularité de A sera faible —> si on prend en compte la popularité des pages qui pointent vers A, on ne donnera pas un gros score aux spams
- Keywords:
  - Page = document = site internet = url
  - Inlinks: liens pointant vers une page web
  - Outlinks: liens d'une page vers les autres pages

# Document Processing

## Analyse des liens | Page rank

- Algorithme “Random surfer model”

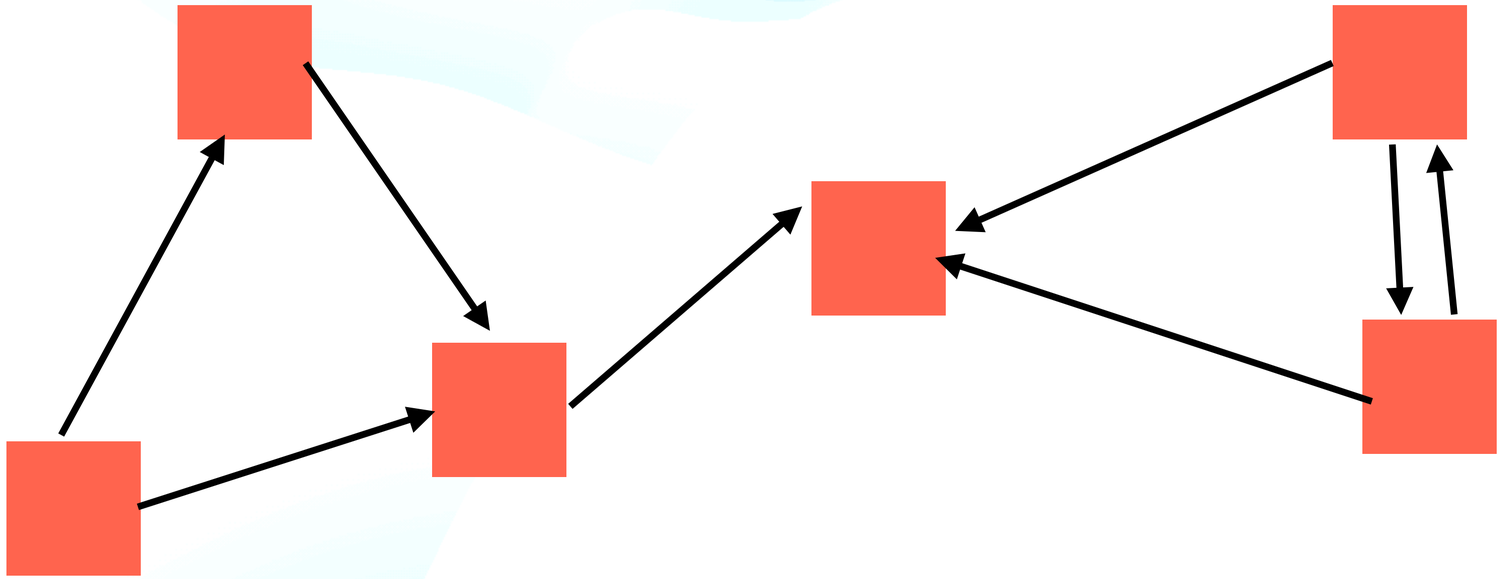
```
func random_surfer_model(threshold)
        r  <- random(0,1)

         if r < threshold
            choisis une page web random
         else
            click sur un lien dans la page web courante

         random_surfer_model(threshold)
```

  - En général, la variable threshold est plutôt petite, par exemple 0.15, pour que l’algorithme ait plus de chances de cliquer sur un lien que de choisir une autre page web

- Le score de page rank d’une page est la probabilité que le surfer arrive sur cette page

# Document Processing

## Analyse des liens | Page rank



# Document Processing

## Analyse des liens | Page rank

- L’algorithme tourne “à l’infini” et va donc visiter tous les sites webs plusieurs fois. Mais il est plus probable qu’il visitera un site populaire des milliers de fois plus souvent qu'un site impopulaire.
- Si on ne lui permettait pas de chercher une nouvelle page random (premier if) il resterait bloqué
  - sur des pages qui n'ont pas de liens,
  - des pages dont les liens ne pointent plus vers aucune page,
  - ou des pages qui forment une boucle.

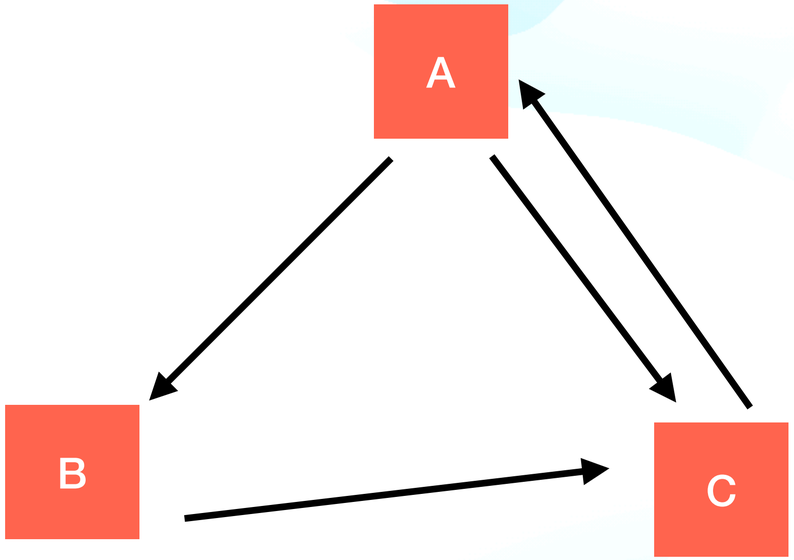
# Document Processing

## Analyse des liens | Page rank

- Au départ toutes les pages ont le meme page rank

# Document Processing

## Analyse des liens | Page rank




- Le page rank d'une page dépend de celui des autres pages
- Plus une page a de liens, moins ses liens ont un poids dans le calcule du page rank
- **Si on ne prend en compte que les inlinks**
  Temps 1:
  PR(C) = PR(A) / 2 + PR(B) / 1 = 0.33 / 2 + 0.33 = 0.5
  PR(A) = PR(C) / 1 = 0.33
  PR(B) = PR(A) / 2 = 0.17

  Temps 2:
  PR(C) = 0.33/2 + 0.17 = 0.33
  PR(A) = 0.5
  PR(B) = 0.25

.

# Document Processing

## Analyse des liens | Page rank

$$PR(u) = \frac{\lambda}{N} + (1 - \lambda) \cdot \sum_{v \in B_u} \frac{PR(v)}{L_v}$$

- **Si on ne prend en compte que les inlinks ET le random sur les pages**

- N -> le nombre de documents
- Lambda -> probabilité de choisir une page random
  1-lambda -> probabilité de cliquer sur un lien
- Bu -> inlinks de l'url u
- Lv -> # outlinks depuis l'url v

# Document Embeddings

# Document Embeddings

## Dense representations | BERT-like embeddings

- Embeddings: représentation vectorielle d'une string (plus ou moins longue)
- BERT-like embeddings: représentation contextualisée au niveau du word-piece
  Exemple:
  - Danone est mon yaourt préféré
  - Je déteste Danone
  Le vecteur de Danone sera différente
- Problèmes en recherche d'information:
  - Ce sont des representation assez longues à générer
    Comme elles dependent du contexte, on ne peut pas simplement avoir une hashmap de token -> vecteur
  - Ces embeddings ont une contrainte de taille de la string en input (en général max 512 word pieces), un document web est souvent bien plus long
    On choisit alors
    - soit de représenter le début du document
    - soit de diviser le document en chunks de max 512 wp et de multiplier les vecteurs résultants
    - Soit de partir d'un summary du document

# Document Embeddings

## Dense representations | ColBERT

- Similarité entre la requête et le document
- Un meme modele qui encode la requête et le document séparément et donne un score de similarité entre un document et une requête
- On va encoder le document au moment de l’indexation
  On n’aura plus qu’à encoder la requête

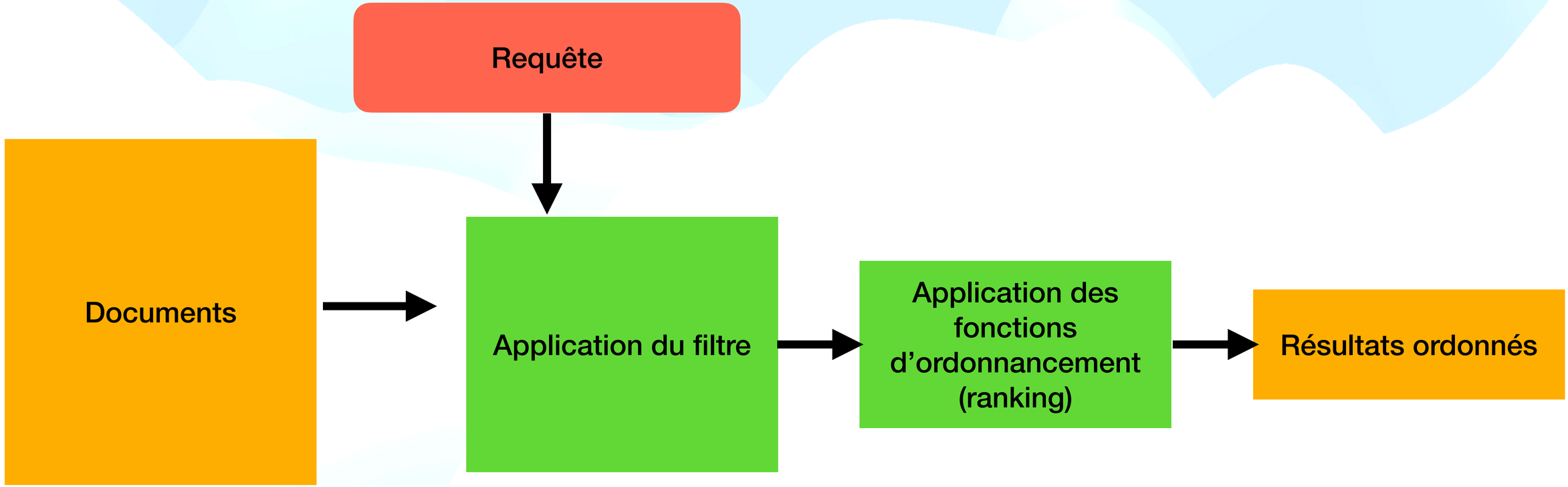
# Partie 2
# Traitement de la requête et ordonnancement des résultats

Rappels

# Traitement de la requête et ordonnancement des résultats

- A ce stade, on a:
  - Une base de données avec des milliards de documents
  - Ces documents ont été process pour en extraire plusieurs informations:
    - Textuelles: title, content, metatags, headers etc.
    - Scores: popularité (ex: page rank), confiance, spam etc.
    - Induites: langue, topic etc.

# Traitement de la requête et ordonnancement des résultats



# Partie 2
# Traitement de la requête et ordonnancement des résultats

Index

# Index

- On va parler d’index, de la base de donnée qui contient les documents
- Il faut garder en tête que l’on veut qu’ils matchent avec la requête
  Toute la construction d’un index est faite par rapport au processing de la requête
- La structure de données la plus commune est appelée **index inversé** (ou inverted index)

# Index inversé | Inverted index

- Un index inversé est l'équivalent informatique de l'index que l'on trouve à la fin d’un livre, une page d’un livre étant l’équivalent d’une page web
  L'index des livres est classé dans l'ordre alphabétique des tokens.

- Une difference majeure entre un index de livre et un index de moteur de recherche c’est qu’on ne va pas se concentrer sur les tokens les plus importants mais sur tous les tokens du vocabulaire.

- Un index inversé consiste en une association document -> token, plutôt que token -> documents

# Création d’un index inversé

Document 1

Paul mange une pomme dans le jardin

Document 2

Jean mange une poire dans la cuisine

| Index<br><br>Tokens | inversé<br><br>Documents |
|---|---|
| Paul | 1 |
| mange | 1 |
| une | 1 |
| pomme | 1 |
| dans | 1 |
| le | 1 |
| jardin | 1 |
| Jean | 2 |
| mange | 2 |
| une | 2 |
| poire | 2 |
| dans | 2 |
| la | 2 |
| cuisine | 2 |

# Création d’un index inversé

Document 1

Paul mange une pomme dans le jardin

Document 2

Jean mange une poire dans la cuisine

| Index<br><br>Tokens | inversé<br><br>Documents |
|---|---|
| Paul | 1 |
| mange | 1,2 |
| une | 1,2 |
| pomme | 1 |
| dans | 1,2 |
| le | 1 |
| jardin | 1 |
| Jean | 2 |
| poire | 2 |
| la | 2 |
| cuisine | 2 |

# Création d’un index inversé

## Avec un compte de chaque token

Document 1

Paul mange la pomme
dans la chambre

Document 2

Jean mange la poire
dans la cuisine

| Index<br><br>Tokens | inversé<br><br>Documents |
|---|---|
| Paul | 1:1 |
| mange | 1:1,2:1 |
| la | 1:2,2:2 |
| pomme | 1:1 |
| dans | 1:1,2:1 |
| chambre | 1:1 |
| Jean | 2:1 |
| poire | 2:1 |
| cuisine | 2:1 |

# Création d’un index inversé

## Avec la position de chaque token

Document 1

Paul mange la pomme
dans la chambre

Document 2

Jean mange la poire
dans la cuisine

| Index<br><br>Tokens | inversé<br><br>Documents |
|---|---|
| Paul | 1:[0] |
| mange | 1:[1],2:[1] |
| la | 1:[2,5],2:[2,5] |
| pomme | 1:[3] |
| dans | 1:[4],2:[4] |
| chambre | 1:[6] |
| Jean | 2:[0] |
| poire | 2:[3] |
| cuisine | 2:[6] |

# Création d’un index inversé

## N-grams

Document 1

Paul mange la pomme
dans la chambre

Document 2

Jean mange la poire
dans la cuisine

| Index<br><br>Tokens | inversé<br><br>Documents |
|---|---|
| Paul mange | [1] |
| mange la | [1, 2] |
| la pomme | [1] |
| pomme dans | [1] |
| dans la | [1, 2] |
| la chambre | [1] |
| Jean mange | [2] |
| la poire | [2] |
| poire dans | [2] |
| la cuisine | [2] |

# Index inversé | Inverted index

## Compression

- Les listes inversées générées par token ou n gram peuvent devenir très vite très longue
- On va vouloir compresser ces listes pour économiser de l'espace disque + de l'espace mémoire

# Document Fields

- On parle d’un index mais en général on en a plusieurs pour représenter nos documents
- Les documents ne sont pas de simples listes de mots. Ils sont composés de phrases et de paragraphes qui séparent les concepts en unités logiques.
- Intéressant de garder cette division dans l’index pour pouvoir utiliser l’importance de certains champs par rapport à d’autres lorsque l’on voudra filtrer ou calculer un score

# Document fields

- Exemple:
  - Requete: le bon coin
    Document 1 - title : Le bon coin | content : Trouvez tout ce dont vous avez besoin
    Document 2 - title : La déco de Lucette | content: Je trouve tous mes meubles sur le bon coin!

# Document fields

## Option 1: plusieurs index inversés

- L’idée est de construire plusieurs index inversés,
- Un par champs textuel :
  - Title
  - Content
  - Les champs de l’url (domain, path)
- Si on cherche les tokens de la requête dans le titre, on peut directement taper dans l’index titre. Mais en général, on veut chercher dans plusieurs champs et avec cette méthode, on doit récupérer les listes inversées depuis plusieurs index.

# Document fields

## Option 2: one index to rule them all

Document 1

Title:
Paul

Content:
Paul mange une pomme
dans le jardin

Document 2

Title:
Jean

Content:
Jean mange une poire
dans la cuisine

| Index<br><br>Tokens | inversé<br><br>Documents |
|---|---|
| Paul | 1: [0,1] |
| mange | 1:[1],2:[1] |
| une | 1:[1],2:[1] |
| pomme | 1:[1] |
| dans | 1:[1],2:[1] |
| le | 1:[1] |
| jardin | 1:[1] |
| Jean | 2: [0,1] |
| poire | 2:[1] |
| la | 2:[1] |
| cuisine | 2:[1] |

Title: 0 | Content: 1

# Document fields

## Option 3: extent lists

| | |
|---|---|
| Maison | Inverted index with word positions:<br>[document, word position]<br>[1,2] [1,4] [2,7] [2,18] [2,23] [3,2] [3,6] [4,3] [4,13] |
| Title | Positions of title in the document<br>[document, (begin title, end title)]<br>[1: (1,3)] [2: (1,5)] [4: (9,15)] |

- On peut étendre ce concept à tous les champs et meme définir de nouvelles listes comme la position des verbes dans le document, ou des entités nommées

# Constructions de l’index

## Algorithme

- Inputs:
  - liste de documents
  - options de processing des tokens (ex: downcassing, stemming, lemmatisation etc.)

1. On initialise un index vide
2. Chaque document est tokenisé
3. Optionnel: on process chaque token en fonction des options sélectionnées
4. Pour chaque token, on créer une liste inversée et on l’ajoute dans l’index

Output: l’index créé

# Constructions de l’index

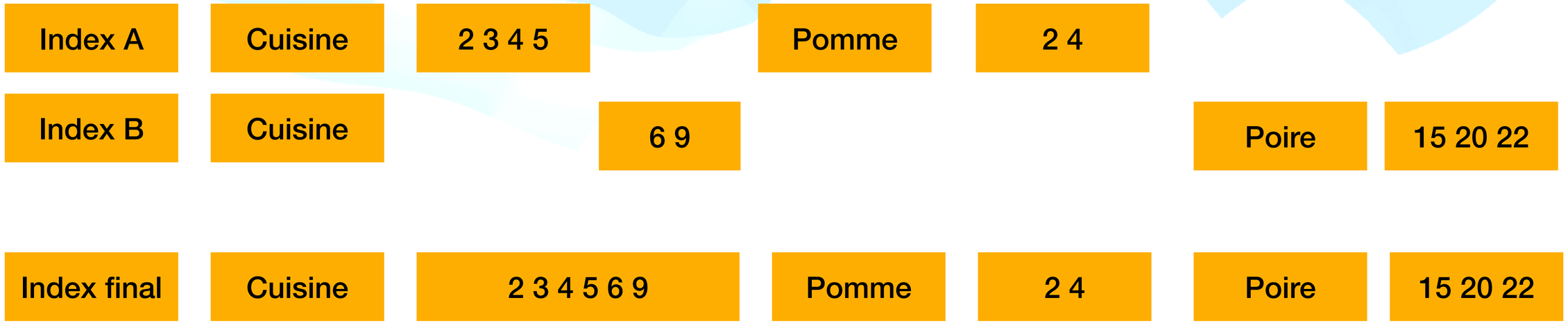
## Limites de cet algorithme

- Fonctionne pour des applications très simples, sur peu de documents (quelques milliers)
- Limite 1: il garde en mémoire toutes les listes inversées
- Limite 2: compliqué à paralleliser en tant que tel avec un index que l’on accède en permanence

# Constructions de l’index

## Solutions: merging + distributed programming

- Pour régler le problème de mémoire: fusion des index



# Constructions de l’index

## Solutions: merging + distributed programming

- Pour régler le problème de parallélisation: Map Reduce
- Processus de base

1. MAP: transforme l’input en paires clé:valeur
   clé: token
   valeur: document:position

2. SHUFFLE: utilise une fonction de hachage afin que toutes les paires ayant la même clé se retrouvent les unes à côté des autres et sur la même machine

3. REDUCE: traite les données par batch, où toutes les paires ayant la même clé sont traitées en même temps.

# Put / update

- Jusqu'à présent, nous avons supposé que la collection de documents était statique.
- Dès qu’on ajoute un nouveau document:
  - de nouveaux tokens doivent être ajoutés
  - Les listes pour les tokens déjà présents doivent etre mis à jour

# Put / update

- Un des solutions: maintenir deux index :
  - un grand index principal
  - un petit index auxiliaire qui stocke les nouveaux documents, il est conservé en mémoire.
- Les recherches sont effectuées sur les deux index et les résultats sont fusionnés.
- Les suppressions sont stockées dans une liste qui nous servira de filtre
- Les documents sont mis à jour en les supprimant et en les réinsérant.
- Dès que l'index auxiliaire est trop grand, on le fusionne avec le grand et on en créer un nouveau pour les prochaines opérations.

# Partie 2
# Traitement de la requête et ordonnancement des résultats

## Compréhension de la requête

# Besoin de l’utilisateur

- le nombre de documents pertinents recherchés
- type d'information nécessaire
- type de tâche qui a conduit au besoin d'information

# Besoin de l’utilisateur

Une requête peut représenter des besoins d'information très différents

- Peut nécessiter des techniques de recherche et des algorithmes de classement différents pour produire les meilleurs classements.

Une requête peut être une mauvaise représentation du besoin:

- L'utilisateur peut trouver difficile d'exprimer son besoin d’information
- L'utilisateur est encouragé à saisir des requêtes courtes

# Types de requêtes

Keyword queries

**Split str python**

Natural language queries

**How to split a string in python?**

# Opérateurs dans une requête

L’utilisateur peut aussi nous donner des informations sur le type de recherche qu’il veut faire:

- Quotes: recherche exacte
- OR, | : un token ou un autre
- - : pour exclure un token
- inurl: cherche explicitement le token qui suit dans le texte de l’url

Google’s query operators

# Transformations de la requête

# Transformations de la requête

- Les **méthodes globales** sont des techniques permettant d'étendre ou de reformuler les termes d'une requête indépendamment de la requête et des résultats qu'elle renvoie, de sorte que les changements dans la formulation de la requête entraîneront la correspondance de la nouvelle requête avec d'autres termes sémantiquement similaires.
  - Des techniques de stemming
  - Des techniques comme la correction orthographique
- Les **méthodes locales** ajustent une requête par rapport aux documents qui semblent initialement correspondre à la requête.
  - Relevance feedback
  - Pseudo relevance feedback
  - Retour de pertinence indirect

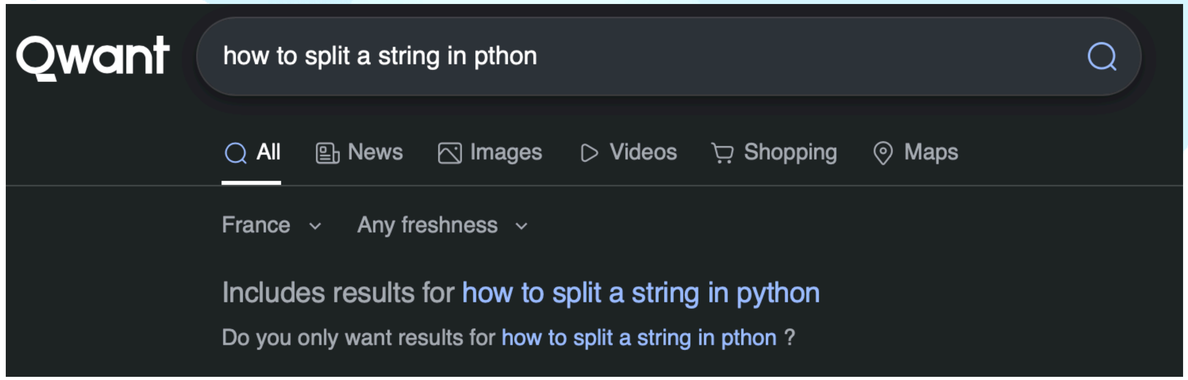
# Transformations de la requête

## Méthode globale | Stemming

- Rappel sur le stemming:
  - Allons -> all (en français, -ons est une terminaison)
  - Permet d'accroître la flexibilité du moteur de recherche.
- Si on veut stemmer la requête:
  - il faut en détecter la langue
  - il faut la confronter à un ou plusieurs index stemmés
- Tache d’expansion de la requête:
  - On garde les deux versions pour matcher plus de documents

# Transformations de la requête

## Méthode globale | Correction orthographique




- vaccances → vacances (erreur d'insertion)
- redit → reddit (erreur de suppression)
- youyube → youtube (erreur de substitution)
- frnace info → france info (erreur de transposition).

# Transformations de la requête

## Méthode globale | Correction orthographique

- Une méthode simple:
  - Construire un vocabulaire, par exemple à partir de notre index, en excluant les mots les moins fréquents (qui pourraient etre eux aussi mal orthographiés)
  - Si un mot de la requête ne se trouve pas dans ce vocabulaire, chercher un mot qui s’en approche par un calcule de similarité
- Une autre méthode:

# Transformations de la requête

## Méthode globale | Correction orthographique

- Une méthode simple:
  - Construire un vocabulaire, par exemple à partir de notre index, en excluant les mots les moins fréquents (qui pourraient etre eux aussi mal orthographiés)
  - Si un mot de la requête ne se trouve pas dans ce vocabulaire, chercher un mot qui s'en approche par un calcule de similarité
- Tache complexe en NLP, pour essayer d'avoir à l'utiliser le moins possible, on développe l'autocomplete:

# Transformations de la requête

## Méthodes locales | Query expansion

- On cherche à matcher le plus de documents pertinents possibles
- La tache de query expansion se matérialise par différentes techniques:
  - Shingle:
    Requete: **le bon coin**, on va alors aussi vouloir chercher dans l'url **leboncoin**
  - Termes liés:
    Requête: **yt**, on va aussi vouloir chercher YouTube
  - Stemming peut etre vu comme une technique d'expansion de la requête

# Transformations de la requête

## Méthodes locales | Query expansion

Approches généralement basées sur une analyse de la co-occurrence des termes

- soit dans la collection entière de documents,
- soit dans une grande collection de requêtes,
- soit dans les documents les mieux classés dans une liste de résultats.

On va chercher à trouver les tokens associés dans nos documents aux mots dans la requete:

**Exemple**: aquarium pourrait etre associé à zoologie, poisson, reptile, corail, animal, mollusque, marin, sous-marin, parc etc.

# Transformations de la requête

## Méthodes locales | Query expansion

Toutes les techniques qui reposent sur l'analyse des documents sont confrontées à des problèmes de calcul et de précision en raison de la taille de l’index et de la variabilité de la qualité des documents.

Au lieu de s’aider des informations contenues dans l’index, on peut partir de logs de requêtes

La tache de query expansion consiste alors à trouver des requêtes similaires

# Transformations de la requête

## Méthodes locales | Relevance feedback

- Impliquer l'utilisateur afin d'améliorer l'ensemble des résultats finaux.
- L'utilisateur donne son avis sur la pertinence  ou non des documents dans un ensemble initial de résultats.
  - L'utilisateur lance une requête dans le moteur
  - Le moteur renvoie un ensemble initial de résultats de recherche lié à cette requête
  - L'utilisateur annote certains documents retournés comme pertinents ou non pertinents.
  - Le système calcule une meilleure représentation du besoin d'information en fonction des commentaires de l'utilisateur
  - Le système affiche un ensemble révisé de résultats de recherche

Exemple

# Transformations de la requête

## Méthodes locales | Relevance feedback

- On peut lancer plusieurs itérations de ce process, jusqu’à ce que l’utilisateur soit pleinement satisfait des résultats présentés

- On peut aussi utiliser ces informations pour suivre l'évolution du besoin d'information d'un utilisateur : le fait de voir certains documents peut amener les utilisateurs à affiner sa compréhension des informations qu’il recherche

- Dans le web, on utilise peu cette technique:
  difficile à expliquer à l'utilisateur moyen, qu’il comprenne qu’il aura une meilleure réponse si il nous informe de ce qui est pertinent ou non dans les résultats qu’il voit

# Transformations de la requête

## Méthodes locales | Pseudo-relevance feedback

- Relevance feedback automatisé
- En général on utilise les clicks des utilisateurs en inférant que si l'utilisateur a cliqué sur un document, il est pertinent.
- Mais comme toutes les méthodes d'annotation automatiques, elles apportent des biais :

# Transformations de la requête

## Méthodes locales | Pseudo-relevance feedback

- Relevance feedback automatisé
- En général on utilise les clicks des utilisateurs en inférant que si l'utilisateur a cliqué sur un document, il est pertinent.
- Mais comme toutes les méthodes d'annotation automatiques, elles apportent des biais :
  - Qu'est-ce qu'il se passe si tous les documents sont non pertinents ?
  - Qu'est-ce qu'il se passe si la requête est ambiguë et que le moteur se focus sur un seul des sens ?

# Transformations de la requête

## Utiliser le contexte de la requête, les informations de l’utilisateur

- Pour transformer la requête, on peut aussi s’aider des informations que l’utilisateur nous donne sans le savoir:
  - Sa localisation pour des requêtes liées à un lieu
    Exemple:
    Requete: restaurant
    —> on va vouloir lui présenter les restaurants prêt d’où il se trouve
  - Son historique de requêtes pour :
    - Désambiguiser sa requête
    - Utiliser les clicks pour remonter les documents/domains préférés

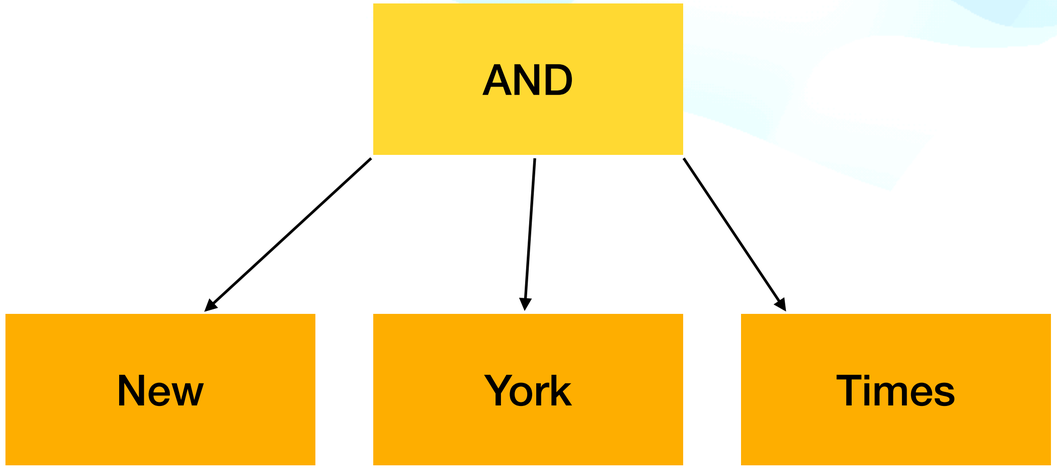
# Transformations de la requête

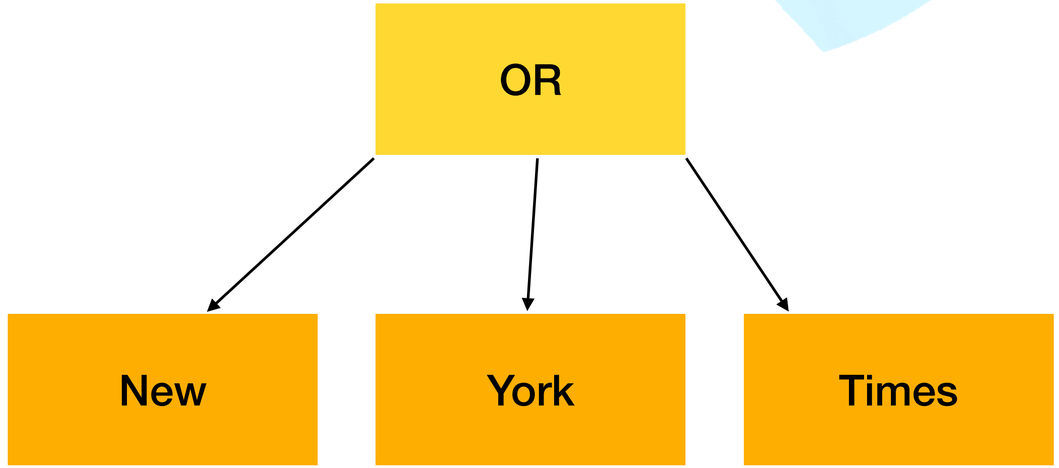
## Transformer la requête en vecteur

# Représentation de la requête

# Représentation de la requête

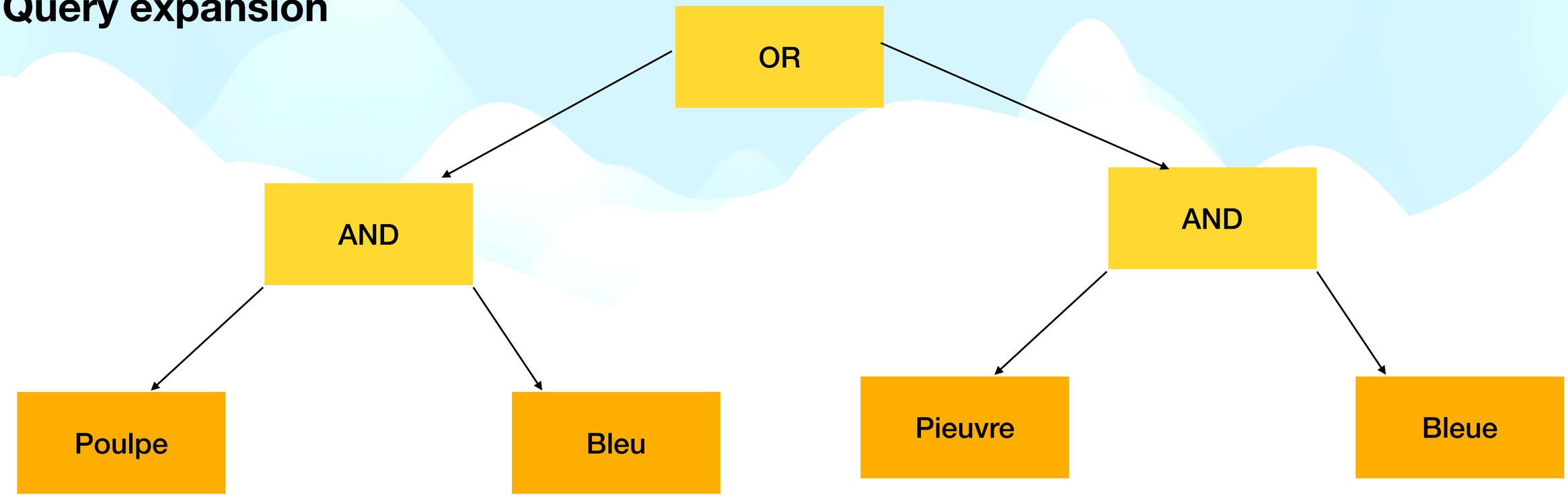
## Tel un arbre



# Représentation de la requête

## Query expansion



# Partie 2
# Traitement de la requête et ordonnancement des résultats

## Ordonnancement des résultats et évaluation

# Ordonnancement des résultats

- On a maintenant:
  - Un index rempli de documents, chaque document est représenté par des champs (title, content, url, page rank etc.)
  - Un traitement de requêtes effectif avec de l'expansion de requête tels que le stemming qui nous permet de filtrer les documents qui pourraient correspondre à la requête
- On veut:
  - Ordonner ces documents qui semblent pertinents pour répondre au mieux à l'utilisateur

# Signaux

# Ordonnancement des résultats (ranking)

## Signaux

- Pour ordonner les résultats on va donner un score sur chaque document qui a survécu au filtre de la requête
- On va classer les signaux en plusieurs types en fonction de leur dépendance à la requête ou au document

# Ordonnancement des résultats (ranking)

## Signaux dépendant de la requête (et de l’utilisateur)

- Date et heure à laquelle a été lancée la requête
- Localisation de l’utilisateur
- Age de l’utilisateur
- Intention / topic
- Etc.

# Ordonnancement des résultats (ranking)

## Signaux dépendant du document

- Page rank et autres scores de popularité
- Spam / ham
- Topics
- Date de derniere mise a jour
- Language of the document
- Etc.

# Ordonnancement des résultats (ranking)

## Signaux dépendant du document et de la requête

- Ce sont des signaux qui vont calculer un score de similarité entre le texte de la requête (et de la requête étendue) avec le texte d'un ou plusieurs champs du document
-

# Ordonnancement des résultats (ranking)

## Bm25(query, document.field, b, k)

$$\sum_{i}^{n} IDF(q_i) \frac{f(q_i, D) * (k1 + 1)}{f(q_i, D) + k1 * (1 - b + b * \frac{fieldLen}{avgFieldLen})}$$

- qi -> token i de la requête
- f(qi, D) -> fréquence de qi dans le champs du document, plus le token qi apparait plus le score final sera élevé
- fieldLen/avg(fieldLen) —> En d'autres termes, plus le document contient de tokens - du moins ceux qui ne correspondent pas à la requête - plus le score du document est faible. Si un document de 300 pages mentionne ma requête une fois, il est moins probable qu'il ait un rapport avec elle qu'un court tweet qui la mentionne une fois.
- b est un paramètre entre 0 et 1, plus il est grand et plus fieldLen/avg(fieldLen) a d'effet, défaut: 0.75
- K1 limite à quel point un seul token dans la requête peut affecter le score —> stopwords, default: 1.2

# Ordonnancement des résultats (ranking)

## Bm25(query, document.field)

$$\sum_{i}^{n} IDF(q_i) \frac{f(q_i, D) * (k1 + 1)}{f(q_i, D) + k1 * (1 - b + b * \frac{fieldLen}{avgFieldLen})}$$

- IDF(qi) -> inverse document frequency
  diminue le poids des termes qui apparaissent très fréquemment dans l'ensemble de documents et augmente le poids des termes qui apparaissent rarement

$$: \log \frac{N}{|\{d \in D : t \in d\}|}$$

# Ordonnancement des résultats (ranking)

## Bm25(query, document.field)

Exemple q = president Lincoln
# docs -> 500 000
freq lincoln -> 40 000
freq pommes -> 300
K1 = 1.2, b = 0.75

| Frequency of "president" | Frequency of "lincoln" | BM25 score |
|---|---|---|
| 15 | 25 | 20.66 |
| 15 | 1 | 12.74 |
| 15 | 0 | 5.00 |
| 1 | 25 | 18.2 |
| 0 | 25 | 15.66 |

# Tache de ranking

# Ordonnancement des résultats (ranking)

## Linear Ranking

- On va prendre plusieurs scores/fonctions de scoring et on va donner une importance plus ou moins forte sur chaque score:
- ranking_function(q,d) ->

```
Score <-   bm25(q, document.title) * 10
         + page_rank * 20
         + same_language(q, document)
```

- A quoi ça vous fait penser ?

# Ordonnancement des résultats (ranking)

## Learning to Rank

- Modèle d'apprentissage automatique qui apprend à prédire un score à partir d’un input x = (q, d)
  Au lieu d’apprendre à minimiser une loss, il va apprendre à maximiser une métrique (choisie en amont, ex: recall, ndcg, map etc.)
- Decision tree, SVM, modèle multi couches
- Supervised, semi supervised, reinforcement learning
- 3 approches différentes
  - Pointwise
  - Pairwise
  - Listwise

# Ordonnancement des résultats (ranking)

## Learning to Rank - point wise

- Input: une paire (q,d) unique
- Loss: distance entre le score prédit et le vrai score
- Il faut le voir comme une tache de régression

# Ordonnancement des résultats (ranking)

## Learning to Rank - pair wise

- Input: une paire (q, di, dj)
- Loss: prédire lequel des deux documents est le plus pertinents
- Il faut le voir comme une tache de classification binaire

# Ordonnancement des résultats (ranking)

## Learning to Rank - list wise

- Input: une paire (q, [di, …, dn])
- Prend en compte tous les documents qui ont survécu au filtre
- A ce moment là, on intègre la métrique dans la loss
  Au lieu de minimiser une loss, on va maximiser la métrique

# Evaluation des résultats

# Evaluation des résultats

- Plusieurs types d’évaluation de la pertinence d’un résultat
- Requete : url et non pas Requête : liste d’urls
- Pertinence de topic
  Si les résultats ont le bon topic
  VS
  pertinence en fonction de l’utilisateur
  Si les résultats sont assez “frais”, s’il a la bonne langue etc.
- Binaire
  pertinent ou non pertinent
  VS
  à plusieurs valeurs
  plus ou moins pertinent

# Evaluation des résultats

## Evaluation binaire | Recall @ k

- Cette mesure indique combien de résultats pertinents réels ont été affichés parmi tous les résultats pertinents réels pour la requête
- Recall = TP/(TP + FN)


Recall@1 = 1/3 = 0.33


Recall@3 = 2/(2+1) = 2/3 = 0.67

Quel est le recall@5 de ces deux modèles?

# Evaluation des résultats

## Evaluation binaire | Precision @ k

- Cette métrique quantifie le nombre d'éléments pertinents dans les résultats du top-K
- Precision = TP/(TP + FP)



quel est la Precision@5 de ces deux modèles?

# Evaluation des résultats

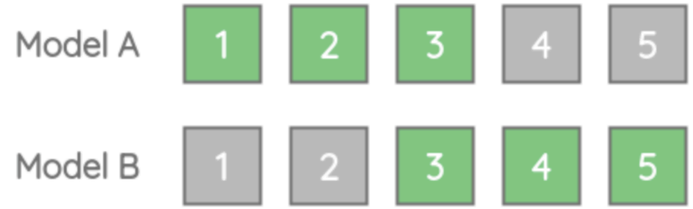
## Evaluation binaire | Average Precision

- Une métrique qui prend en compte l'ordre des documents retournés

$$AP = \frac{\sum_{k=1}^{n}(P(k) * rel(k))}{number\ of\ relevant\ items}$$


$$AP = \frac{(1 + 2/3 + 3/5)}{3} = 0.7555$$

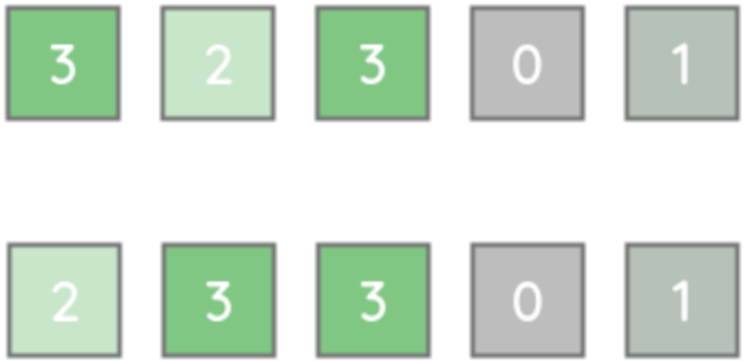
quel est l' Average Precision de ces deux modèles?



# Evaluation des résultats

## Evaluation multivaleurs | Cumulative Gain (CG)

- Si on a un score de pertinence sur nos documents, il y a des métriques qui prennent en compte ces scores:
- Cumulative Gain -> somme des scores
  - CG@3 ?



# Evaluation des résultats

## Evaluation multivaleurs | Discounted Cumulative Gain (DCG)

- Nous avons donc besoin d'un moyen de pénaliser les scores en fonction de leur position. Le GDC introduit une fonction de pénalité basée sur le logarithme pour réduire le score de pertinence à chaque position.

- Discounted Cumulative Gain

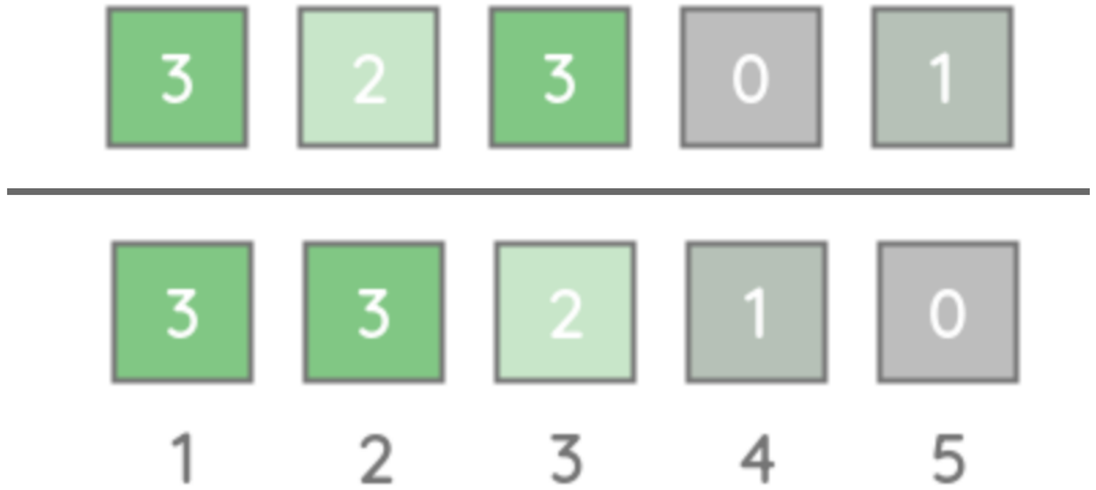
$$DCG@k = \sum_{i=1}^{k} \frac{rel_i}{log_2(i+1)}$$


- DCG@3 = 3 + 1.26 + 1.5 = 5.76

# Evaluation des résultats

## Evaluation multivaleurs | Normalised Discounted Cumulative Gain (NDCG)

- Pour permettre une comparaison du GDC entre les requêtes avec des résultats de taille différente
  - NDCG@k = DCG@k / IDCG@k
  - Avec IDCG = ideal dcg



# Indexation web

# Et après?

- Un ranking pour toutes les requêtes ou plusieurs rankings spécialisés ? Pour ou contre?
- Représentations multimodales, que fait-on des images ?
- Comment présenter les résultats?

# Industrie

- Quel moteur de recherche devrais-je utiliser ?
  - Apache Solr
    Solr est le projet de moteur de recherche le plus ancien
    Grande communauté
  - Elastic Search
    d'abord concentré sur la facilité d'utilisation
    Axé sur les applications analytiques
    La plus grande des communautés
  - Vespa.ai
    Vespa a l'avantage de prendre en charge nativement les fonctions de recherche “modernes” (LTR)
    La communauté Vespa est encore petite

# Qwant propose des stages en Data Science

- NLP: Web Document summary
- NLP: Exploiting the structure of HTML to learn document representations
- IR: Large-scale Neural Information Retrieval for real-time Ranking