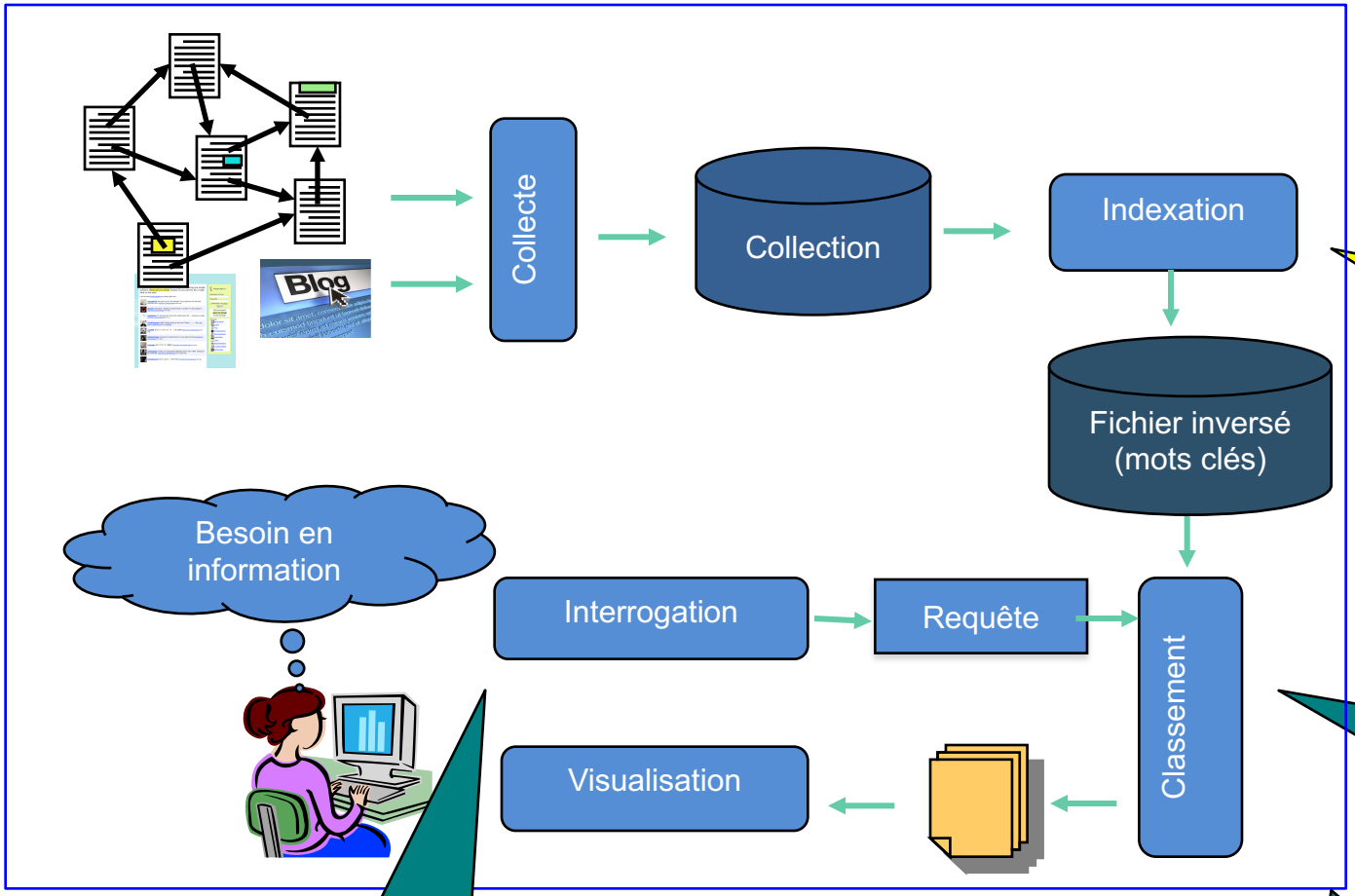
Chapitre 2 : Représentation de l'information/Indexation

Plan du cours



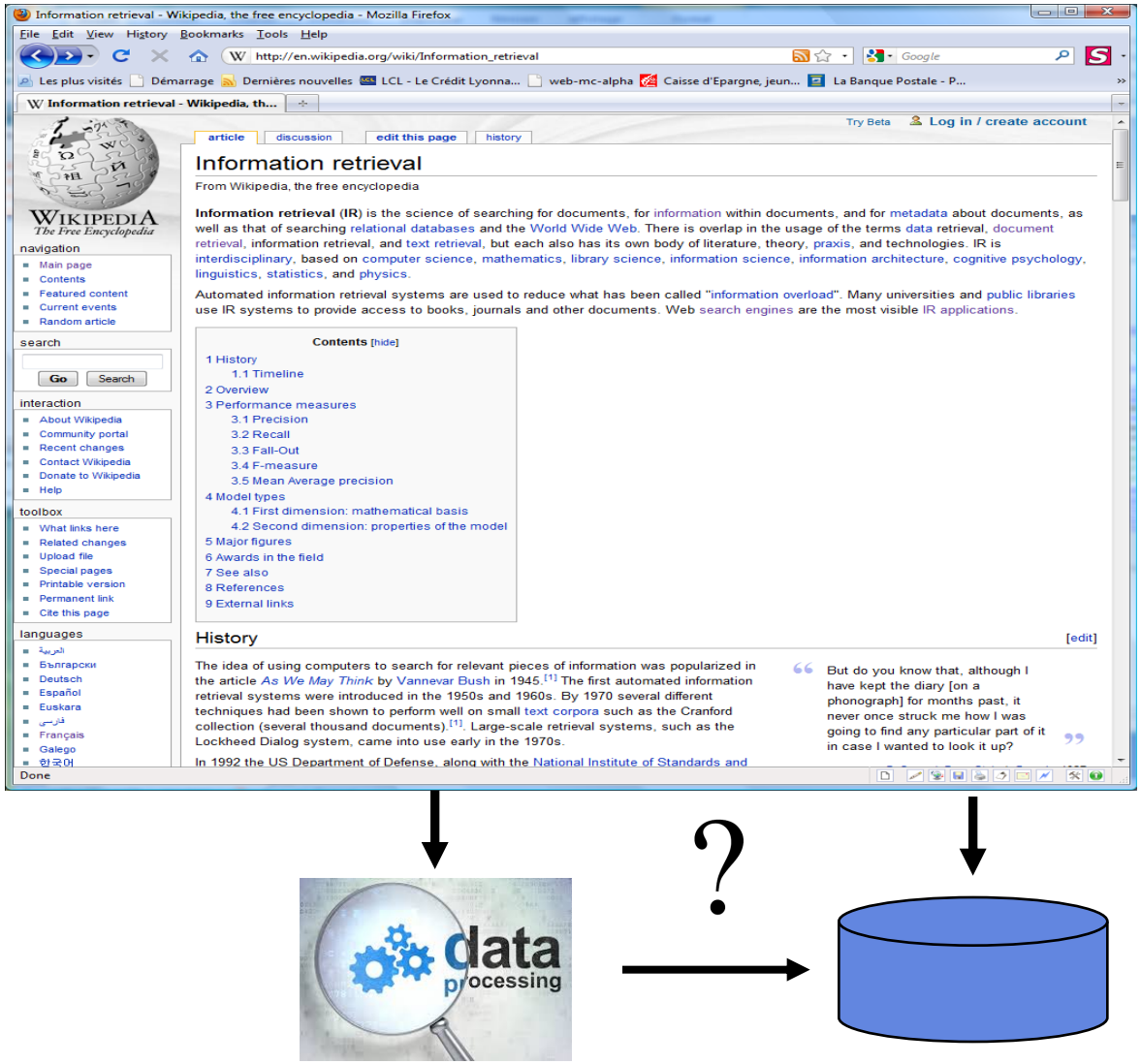
Chapitre 2 :
- Indexation,
- Fichier inversé
- Pondération

Chap 43: Modèles de RI :
Vectoriel,
Probabiliste, ML

Langages d'interrogation

Cadre applicatif : moteurs de recherche Web

Représentation de l'information ?



Représentation des documents

- Représentation des documents (indexation)
  - Processus permettant de construire un ensemble d'éléments, « mots clés », permettant de caractériser le contenu d'un document
- Éléments clés
    - mots simples : pomme
    - groupe de mots : pomme de terre
    - Concepts issus d'ontogie
    - Vecteurs de mots
- Résultat d'une indexation textuelle
  - Chaque document est représenté par une liste d'index (de mots clés)
  - L'index permet d'accéder (sélectionner) aux documents

Options d'indexation

- Type d'indexation
  - Manuelle (faite par un expert) **vs.** Automatique
  - Guidée (contrôlée) vs. Libre
    - Guidée : index (mots clés) choisis à partir d'un vocabulaire contrôlé (thesaurus, ontologie, dictionnaire, lexique, …)
    - Libre (Index (mots clés) pris du texte du document)
- Approches
  - Statistique (distribution des mots) et/ou TALN (compréhension du texte)
  - Approche courante est plutôt statistique avec des hypothèses simples
    - Redondance d'un mot marque son importance
    - Cooccurrence des mots marque le sujet d'un document

Indexation : approche générale

- Décomposer le texte (Parsing)

<Title>: Information retrieval
(Corps du texte> : Information retrieval (IR) is the science of searching of documents. N.I.S.T launched TREC

- Segmenter les séquences de caractères en mots (Tokenizing)

Information retrieval Information retrieval IR is the science of searching of documents N.I.S.T launched TREC

- Normaliser
  - Textuelle: ponctuation, dates, case
  - Linguistique : Racinisation (stemming)/lemmatisation

information retrieval information retrieval IR is the science, of search, document NIST launched TREC

- Supprimer les mots communs (stop word removal)
  - En fonction d'une "short list" "the", "and", "or", ou mots fréquents

information, retrieval, information, retrieval, IR, science, search, document NIST launch TREC

- Regrouper les mots

Un sac de mots (BOW)

information 2, retrieval 2, IR 1, science 1, search 1, document 1, NIST 1, launch 1, TREC 1

Segmentation (Tokenization)

– Pas d'espaces en chinois et en japonais

- Ne garantit pas l'extraction d'un terme de manière unique

Chinese tockenization

1. Original text

旱灾在中国造成的影响

(the impact of droughts in China)

2. Word segmentation

旱灾 在 中国 造成 的 影响

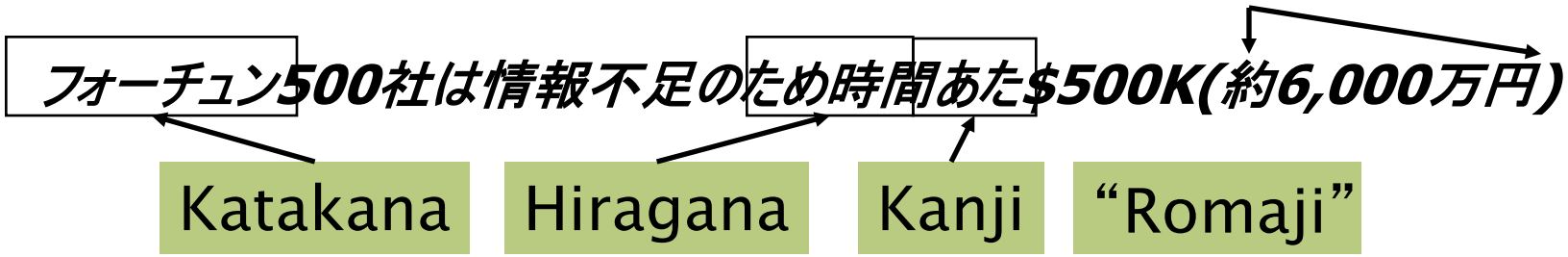
drought at china make impact

3. Bigrams

旱灾 灾在 在中 中国 国造
造成 成的 的影 影响

Segmentation (Tokenization)

– Japonais encore plus compliqué avec différents alphabets



L’utilisateur peut exprimer sa requête entièrement en Hiragana

Normalisation

- Processus morphologique permettant de regrouper les variantes d'un mot
- Normalisation linguistique
  - Racinisation (stemming) : supprimer suffixes et flexions
    - Ex : économie, économiquement, économiste, ⇨ économ
    - pour l'anglais : retrieve, retrieving, retrieval, retrieved, retrieves ⇨ retriev
  - Lemmatisation : analyse linguistique pour ramener les mots à leur lemme (verbe à l'infinitif, noms à leur forme singulier, …)

Racinisation

- Utilisation de règles de transformations
  - règle de type : condition action
    - Ex : si mot se termine par s supprimer la terminaison
    - Plusieurs algos les plus connus : Porter, Lovins
    - Stemmer Snowball (http://snowball.tartarus.org/) disponibles en téléchargement
- Troncature à X caractères

Algorithme de Porter

- Basée sur la mesure de séquences voyelles-consonnes
  - mesure m pour un «stem» est $[C](VC)^m[V]$ ou C est une séquence de consonnes et V est une séquence de voyelles [] = option
  - m=0 (tree, by), m=1 (trouble,oats, trees, ivy), m=2 (troubles, private)
- Algorithme basé sur un ensemble de conditions actions
  - old suffix → new suffix
  - Les règles sont divisées en étapes et sont examinées en séquence
  - e.g. Step 1a:
    - sses → ss *(caresses → caress)*
    - ies → i *(ponies → poni)*
    - s → NULL *(cats → cat)*
  - e.g. Step 1b:
    - if m>0 eed → ee *(agreed → agree)*
    - if *v*ed → NULL *(plastered → plaster but bled → bled)*
- Plusieurs implantations sont accessibles
  - http://www.tartarus.org/~martin/PorterStemmer/

Lemmatisation

- Réduire les variantes flexionnelles des mots à leur forme de base
- Ex.
  - *am, are, is* → *be*
  - *car, cars, car's, cars'* → *car*
  - *chante, chantons, chanterons* → *chanter*
- Utilisation d'un lexique (dictionnaire)
- Analyse grammaticale fine
  - Tree-tagger (http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/)

Ex. de résultats d'indexation (normalisation avec Porter)

- Texte original:

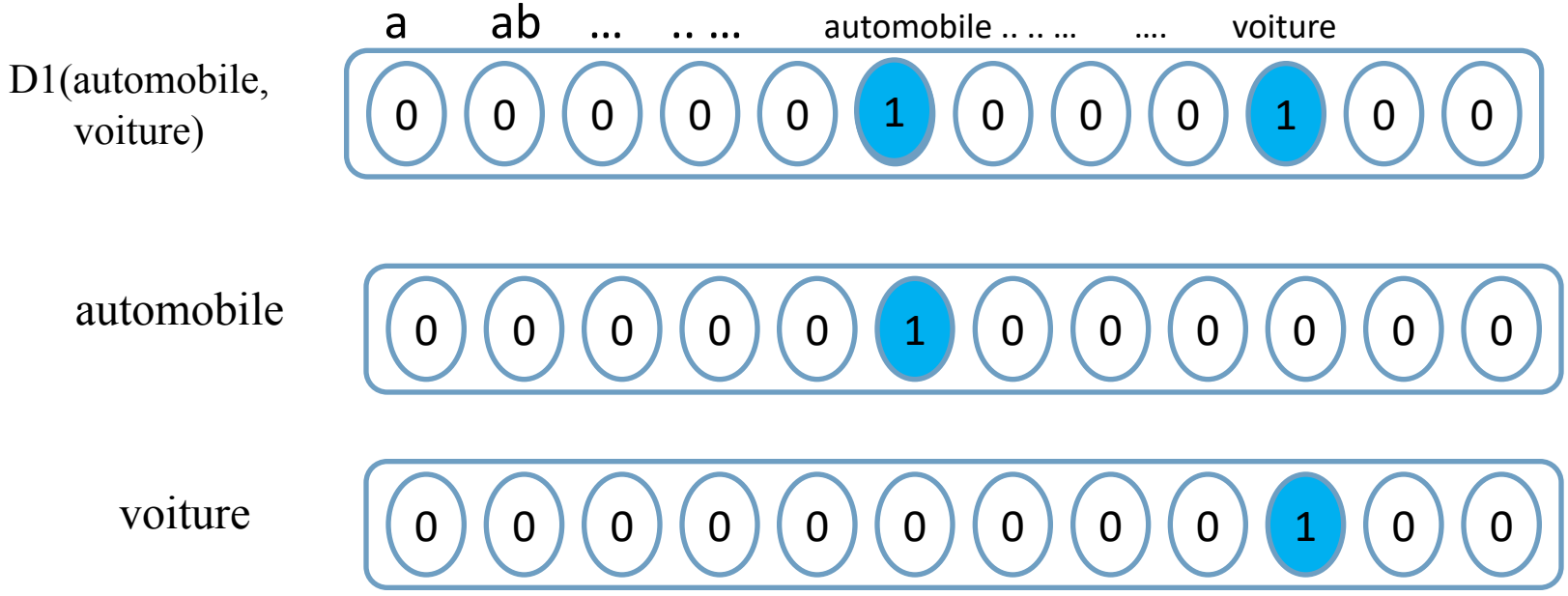
  marketing strategies carried out by U.S. companies for their agricultural chemicals, report predictions for market share of such chemicals, or report market statistics for agrochemicals, pesticide, herbicide, fungicide, insecticide, fertilizer, predicted sales, market share, stimulate demand, price cut, volume of sales

- Texte après porter + suppression mots vides

  Market 4, strateg 1, carr 1, compan 1, US 1, agricultur 1, chemic 2, report 2, predict 2, share 1, statist 1, agrochem 1, pesticid 1, herbicid 1, fungicid 1, insecticid 1, fertil 1, sale 2, stimul 1, demand 1, price 1, cut 1, volum 1

Représentation formelle

- Représentation formelle
  - Ensembliste
    - D1(automobile, voiture, ..)
  - representation vectorielle (“One-hot representation (local)”)
    - D1(automobile, voiture)



Similarité (automobile,voiture) = 0

Représentation physique

- Le fichier inversé
  - Ces termes sont ensuite stockés dans une structure appelée fichier inversé

Fichier Inversé

d1:
So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious

d2:
I did enact Julius Caesar I was killed i' the Capitol; Brutus killed me.

Traitement = Indexation

Term	N docs	Tot Freq	Ptr
ambitious	1	1	1
be	1	1	2
brutus	2	2	3
capitol	1	1	5
caesar	2	3	6
did	1	1	
enact	1	1	
hath	1	1	
I	1	2	
i'	1	1	
it	1	1	
julius	1	1	
killed	1	2	
let	1	1	
me	1	1	
noble	1	1	
so	1	1	
the	2	2	
told	1	1	
you	1	1	
was	2	2	
with	1	1	

Doc #	Freq
2	1
2	1
1	1
2	1
1	1
1	1
2	2
1	1
1	1
2	1
1	2
1	1
2	1
1	1
1	2
2	1
1	1
2	1
2	1
1	1
2	1
2	1
2	1
1	1
2	1
2	1

d1:
So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious

d2:
I did enact Julius Caesar I was killed i' the Capitol; Brutus killed me.

d3:
I did enact Julius Caesar I was killed i' the Capitol; Brutus killed me.

Fichier inversé

Dictionnaire

Mot	Nb Doc	FrqTotal	Ptr
Ambitious	2	5	1
Brutus	2	8	3
capitol	5	15	6

- Liste triée
- B-Arbre
- Table de hashage (hash-code)
- ...

Posting simple

doc	Freq
doc1	3
doc2	2
doc1	1
doc3	7

Posting riche

Position du terme dans le document (important pour la recherche d'expressions)

doc	Freq	position	balise
doc1	3	1, 4, 12	1, 5
doc2	2	1	
doc3	2	3	

Balises (title, body, anchor, ..)

I did enact Julius Caesar I was killed
1 2 3 4 5 6 7 8
i' the Capitol; Brutus killed me.

Répondre à une requête

Caesar, brutus

Term	N docs	Tot Freq	Ptr
ambitious	1	1	1
be	1	1	2
brutus	2	2	3
capitol	1	1	5
caesar	2	3	6
did	1	1	
enact	1	1	
hath	1	1	
I	1	2	
i'	1	1	
it	1	1	
julius	1	1	
killed	1	2	
let	1	1	
me	1	1	
noble	1	1	
so	1	1	
the	2	2	
told	1	1	
you	1	1	
was	2	2	
with	1	1	

Doc #	Freq
2	1
2	1
1	1
2	1
1	1
1	1
2	2
1	1
1	1
2	1
1	2
1	1
2	1
1	1
1	2
2	1
1	1
2	1
2	1
1	1
2	1
2	1
2	1
1	1
2	1
2	1

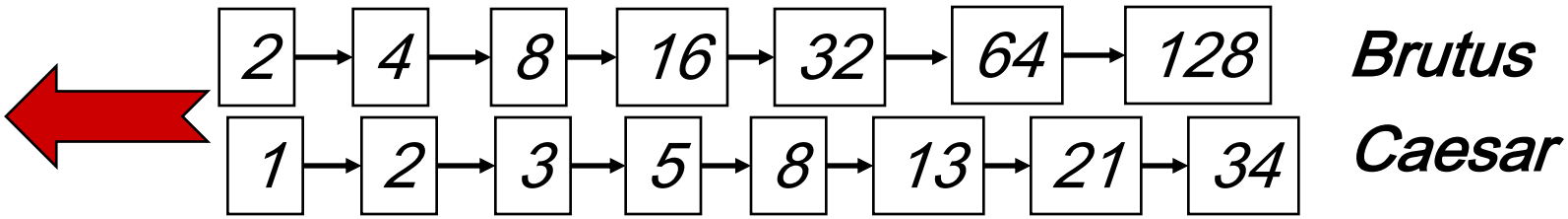
d1:
So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious

d2:
I did enact Julius Caesar I was killed i' the Capitol; Brutus killed me.

d3:
I did enact Julius Caesar I was killed i' the Capitol; Brutus killed me.

Répondre à une requête

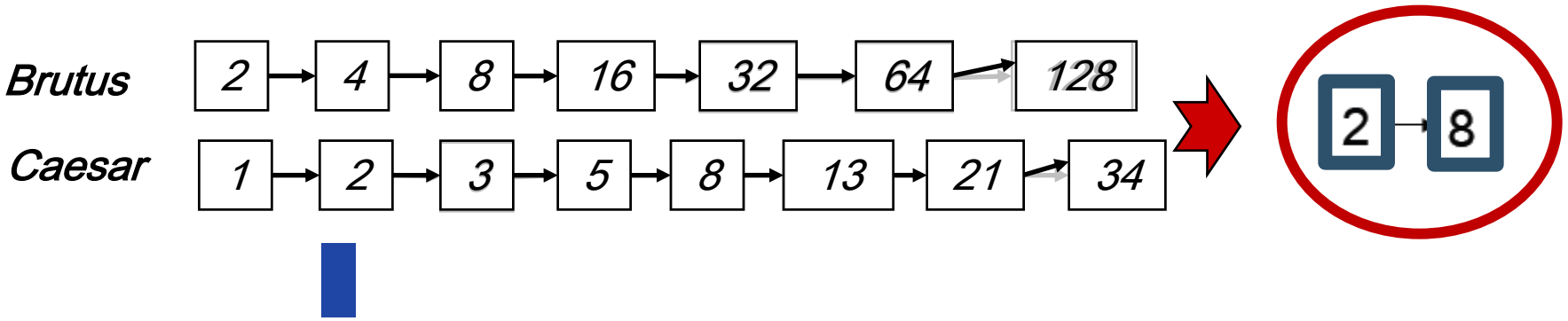
- Soit la requête :
  - ***Brutus*** *AND* ***Caesar***
  - Chercher ***Brutus*** dans le dictionnaire;
    - Sélectionner sa liste postings.
  - Chercher *Caesar* dans le dictionnaire ;
    - Sélectionner sa liste postings.
  - "Fusion" des deux postings:



Répondre à une requête: Fusion

- Pour accélérer la recherche les listes des *postings* sont triées par numéro de document

Fusion → parcours des deux listes simultanément
Si les longueurs des listes sont x et y, l'algo est en O(x+y)



Construction du fichier inversé

- La construction d'un fichier inverse est une «étape importante »
- Impossible à faire en mémoire
- Démarche
  - Attribuer à chaque document un numéro
  - Extraire les mots (mots, doc, fréquence)
  - Trier le fichier par mot terme

  - → transposition de matrice

Extraire les mots de chaque document

- Attribuer numéro : Doc1→1, Doc2→2
- Extraire les termes de chaque document dans un fichier (1fichier par document) ou un fichier pour plusieurs documents)

1

I did enact Julius Caesar I was killed i' the Capitol; Brutus killed me.

2

So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious

Term	Doc #
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Trier le fichier termes-documents

- Trier le fichier par ordre alphabétique des termes et par document

Term	Doc #
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Term	Doc #
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

Trier le fichier termes-documents

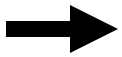
- Pour chaque terme, on dispose
  - de la liste de documents ou il apparaît
  - du nombre de documents comportant ce terme

Term	Doc #
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

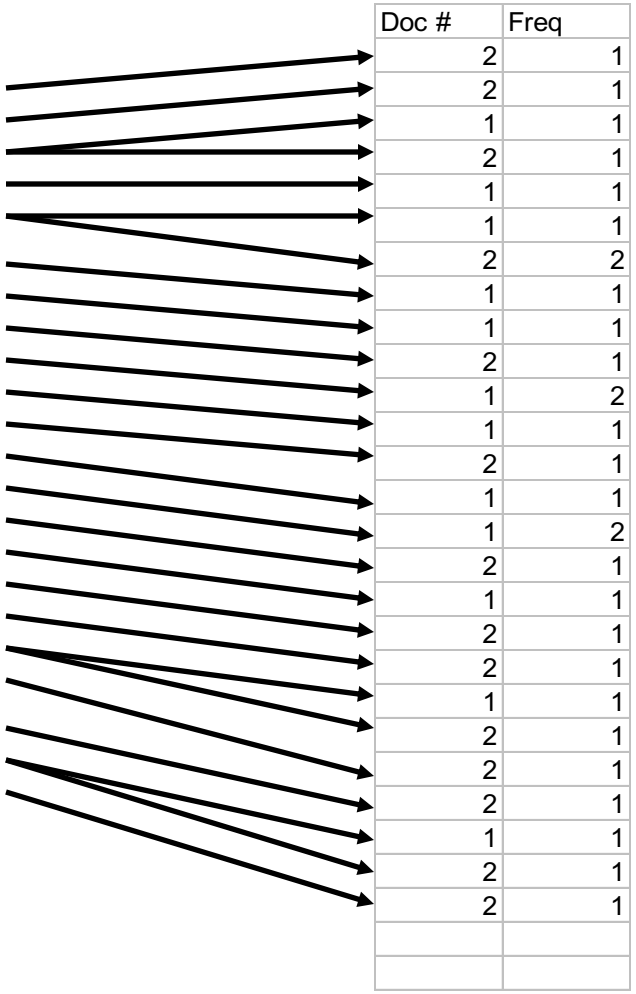
Term	Doc #	Freq
ambitious	2	1
be	2	1
brutus	1	1
brutus	2	1
capitol	1	1
caesar	1	1
caesar	2	2
did	1	1
enact	1	1
hath	2	1
I	1	2
i'	1	1
it	2	1
julius	1	1
killed	1	2
let	2	1
me	1	1
noble	2	1
so	2	1
the	1	1
the	2	1
told	2	1
you	2	1
was	1	1
was	2	1
with	2	1

Construire le dictionnaire et le « posting »

Term	Doc #	Freq
ambitious	2	1
be	2	1
brutus	1	1
brutus	2	1
capitol	1	1
caesar	1	1
caesar	2	2
did	1	1
enact	1	1
hath	2	1
I	1	2
i'	1	1
it	2	1
julius	1	1
killed	1	2
let	2	1
me	1	1
noble	2	1
so	2	1
the	1	1
the	2	1
told	2	1
you	2	1
was	1	1
was	2	1
with	2	1



Term	N docs	Tot Freq
ambitious	1	1
be	1	1
brutus	2	2
capitol	1	1
caesar	2	3
did	1	1
enact	1	1
hath	1	1
I	1	2
i'	1	1
it	1	1
julius	1	1
killed	1	2
let	1	1
me	1	1
noble	1	1
so	1	1
the	2	2
told	1	1
you	1	1
was	2	2
with	1	1

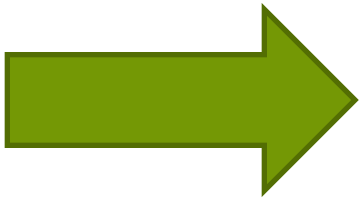


Doc #	Freq
2	1
2	1
1	1
2	1
1	1
1	1
2	2
1	1
1	1
2	1
1	2
1	1
2	1
1	1
1	2
2	1
1	1
2	1
2	1
1	1
2	1
2	1
2	1
1	1
2	1
2	1

Traitement de collections volumineuses

Terme	*Id. Doc*
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1

rme	*Id. Doc*
	2
	2
	2
	2
th	2
esar	2
e	2
ble	2
utus	2
th	2
d	2
u	2
caesar	2
was	2
ambitious	2



Tri par termes

(puis par documents)

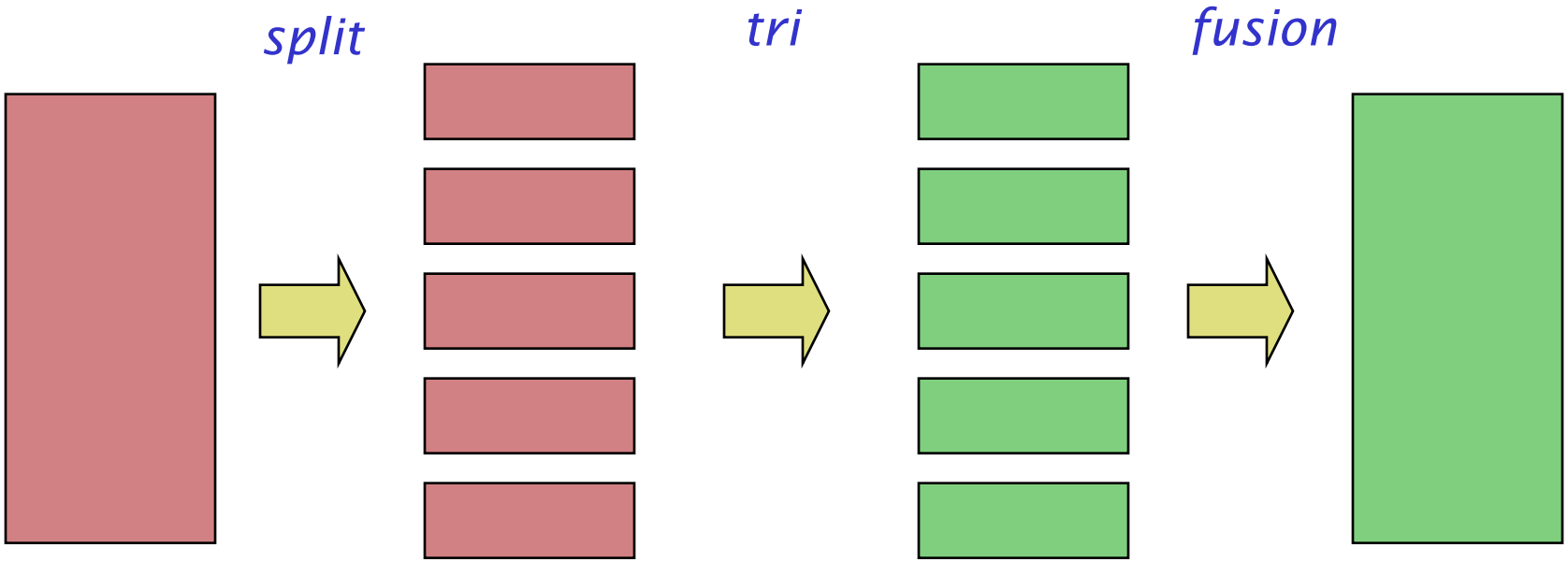
Terme	*Id. Doc*
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
.....	.....

Problème dans l'étape de tri

- Tri, généralement, effectué en **mémoire vive**
- Mais, … impossible pour les collections volumineuses
  - On analyse un document à la fois
  - Les listes des index ne sont complètes qu'à la fin du processus
- **On peut le faire sur disque (tri sur disque) mais très lent**
- Solutions
  - → Tri par bloc (Blocked sort-based Indexing)
  - → Indexation distribuée (→)

Tri par bloc

- Diviser la collection en *n* parties gérables en mémoire
- Trier chaque partie séparément, et réécrire le résultat sur le disque
- Fusionner les résultats→ plusieurs approches de fusion



Indexation distribuée

- → Indexation distribuée
  - Pour des collections volumineuses (Web)
  - Un serveur principal dirige le tout (doit être très sûr)
  - Il divise la tâche d'indexation en un ensemble de tâches parallèles
  - Il assigne chaque tâche à une machine libre et fonctionnelle du réseau

Indexation distribuée

- Les moteurs de recherche utilisent une architecture semblable
  - un système de fichiers distribués
  - un système de contrôle de tâches (job scheduler : quel programme est exécuté sur quelle machine à quel moment)
- Architecture initiale proposée par Google

  (Google File System & Map Reduce)
- Implémentation libre développée dans le projet Hadoop

Centre de données (data centers) de Google

- Centres de données (Data center) Google contiennent principalement des machines de base, distribuées dans le monde entier.

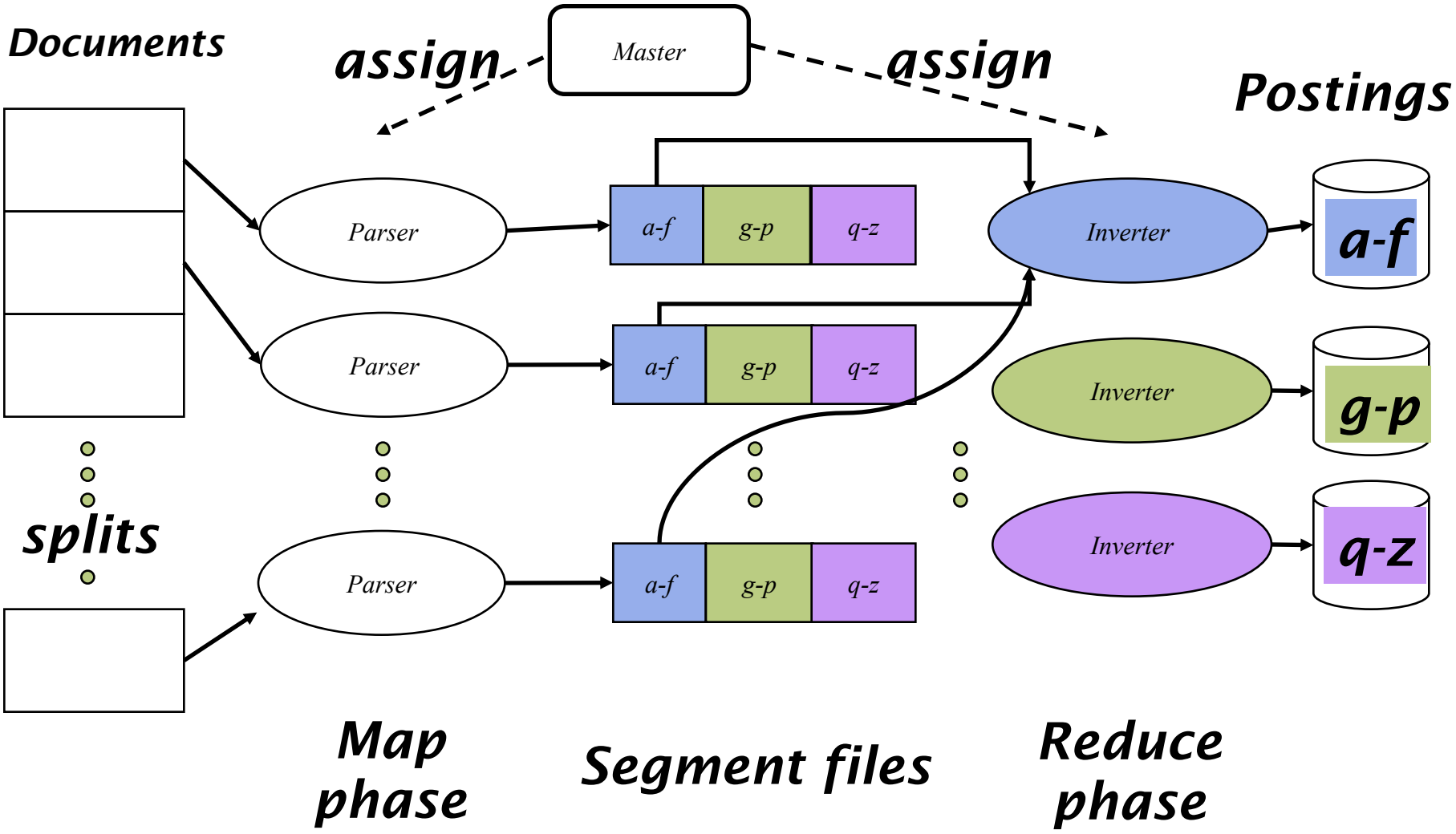
- Estimation:
  - Total de 1 million de serveurs, 3 millions de processeurs/cœurs
  - Google installe 100.000 serveurs chaque trimestre.
  - Dépenses de 200-250 millions de dollars par an

MapReduce

- Principe :
  - Tous les algos sont écrits sous la forme de deux fonctions :
    1. Une fonction ***map*** qui réalise un traitement des données
    2. Une fonction ***reduce*** qui fusionne les résultats intermédiaires produits par *map*

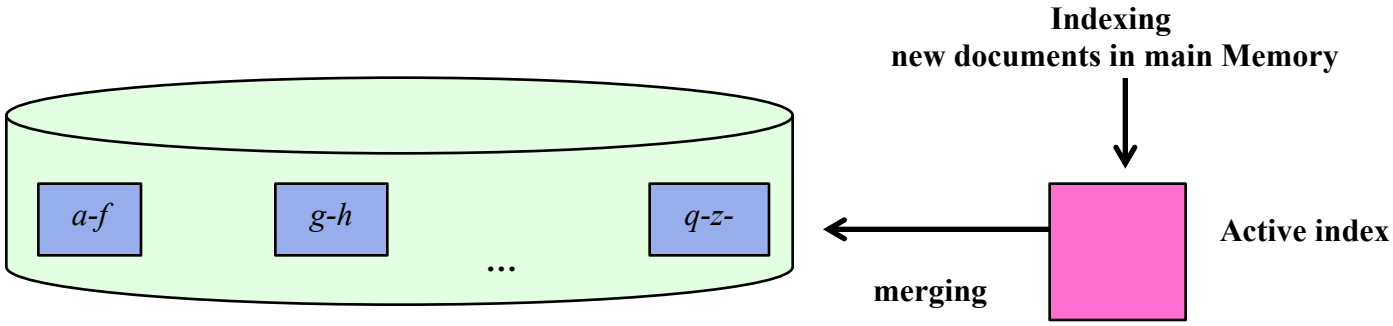
- Interêt :
  - Les tâches *map* sont exécutées sur les machines sur lesquelles sont stockées les données
  - Elles sont exécutées en parallèle

Construction de l'Index avec MapReduce



Indexation des flux de données (ex. Tweets)

- Les collections sont souvent dynamiques (cas du Web, Twitter))
  - → Ajout suppression et modification de documents
  - → Mise à jour du dictionnaire et du *posting*
- Solution (Simple mais inefficace) :
  - Reconstruire l'index à partir de zéro
- Une meilleure solution :
  - Maintenir un index en mémoire qui garde la trace de tous les changements
  - Dès que l'index est "plein" fusionner avec celui (ou ceux) du disque
  - Maintenir un vecteur comportant les documents supprimés



Coût du stockage

Dictionnaire

Term	N docs	Tot Freq
ambitious	1	1
be	1	1
brutus	2	2
capitol	1	1
caesar	2	3
did	1	1
enact	1	1
hath	1	1
I	1	2
i'	1	1
it	1	1
julius	1	1
killed	1	2
let	1	1
me	1	1
noble	1	1
so	1	1
the	2	2
told	1	1
you	1	1
was	2	2
with	1	1

Doc #	Freq
2	1
2	1
1	1
2	1
1	1
1	1
2	
1	
1	
2	
1	2
1	1
2	1
1	1
1	2
2	1
1	1
2	1
2	1
1	1
2	1
2	1
2	1
1	1
2	1
2	1

Postings

Implantation simple du fichier inversé :
- Dictionnaire : 20 octets par terme, 4 octets nbDoc, 4 octets pointeurs
- *Posting*: 4 octets pour l'Id du document, 2 octets pour la fréquence

Réduction du coût de stockage → Compression

FIN

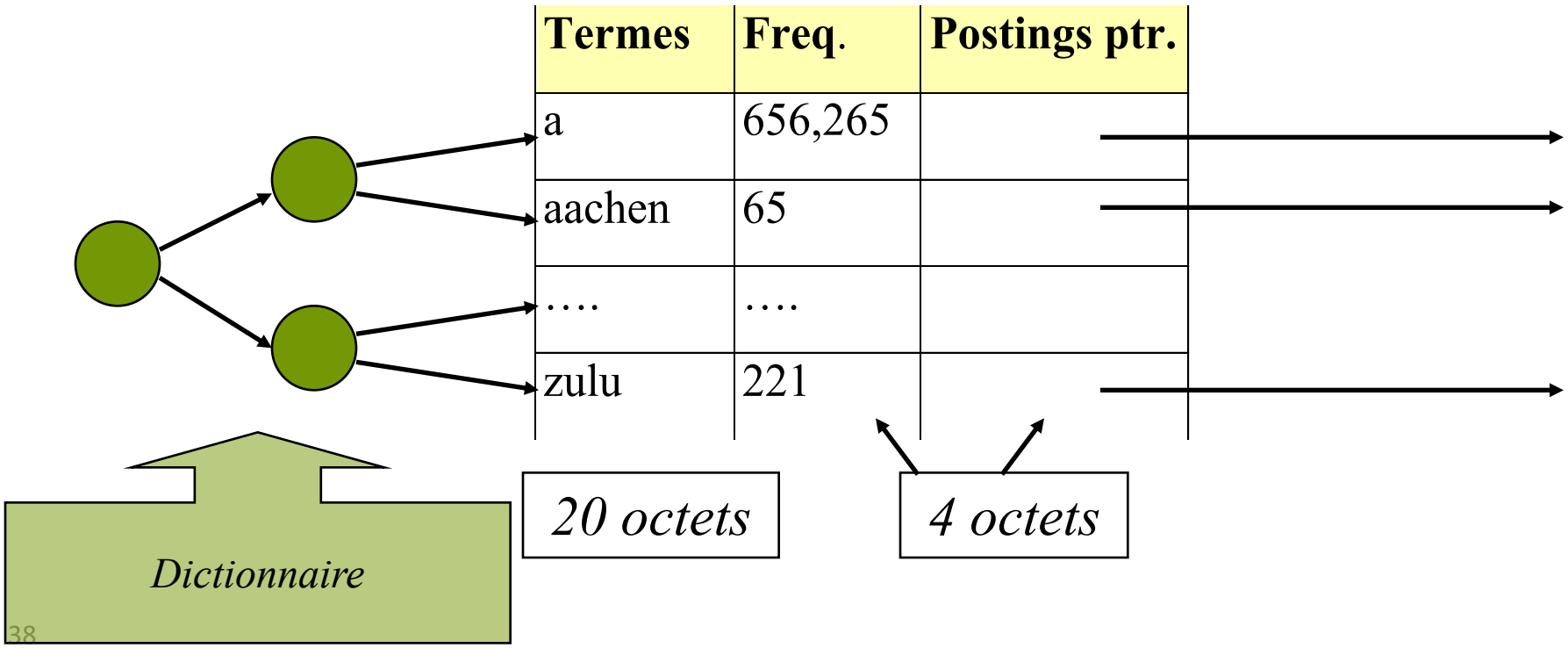
- Vous trouverez la suite du cours sur les techniques de compression du dictionnaire et de l'index (posting) dans les dispos suivantes.

documents additionnels

Compression du dictionnaire (Annexe)

Taille du dictionnaire

- Tableau de taille fixe
  - ~400,000 termes; 28 octets/terme = 11.2 MO.



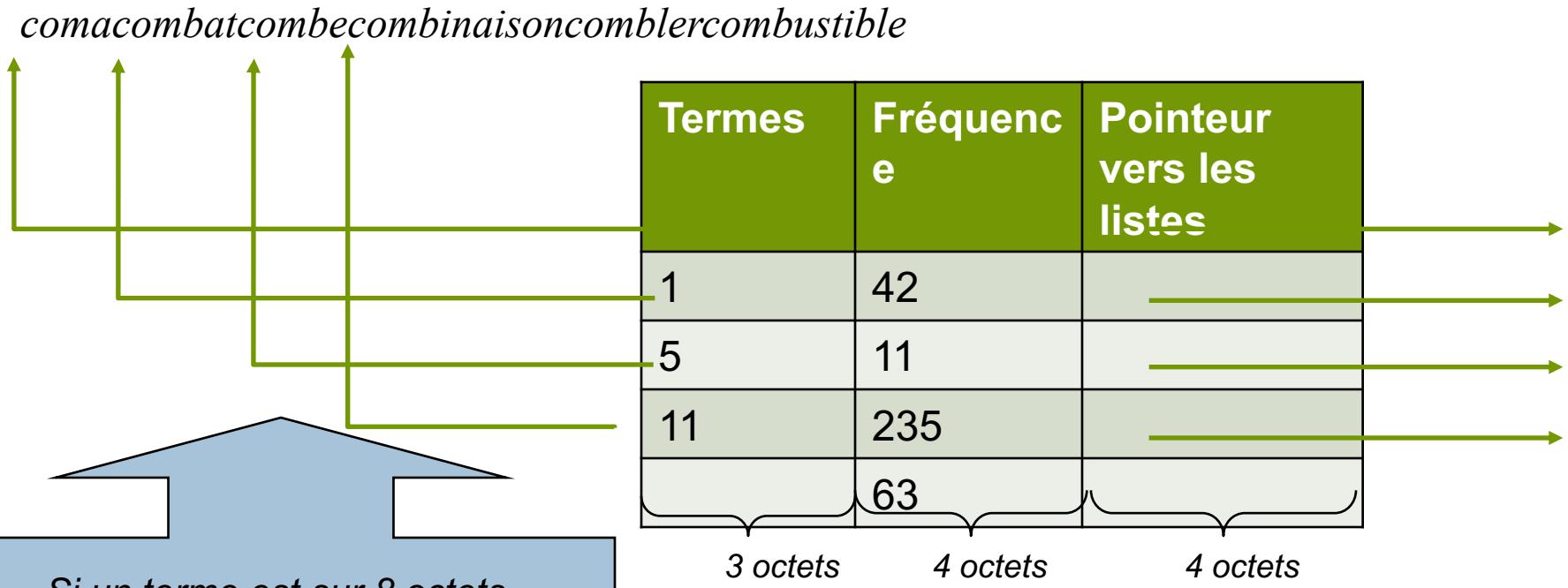
Beaucoup d'espace perdu

- Beaucoup d'espace perdu, les mots à une lettre (a, à, ..) occupe le même espace que des mots longs
  - Il y a des mots qui ne passent pas « anticonstitutionnellement » « *supercalifragilisticexpialidocious* » ou « *hydrochlorofluorocarbons* ».

- Taille moyenne des mots (en anglais), est autour de ~8 caractètes
  - Comment peut-on exploiter ce nombre (~8 caractères par terme)?

Compression du dictionnaire

- Stocker le dictionnaire comme un (long) string de caractères
  - Pointeur vers le terme suivant donne la fin du terme courant



Si un terme est sur 8 octets,
le dictionnaire est sur 11 octets
→ 19 octets par terme au lieu de 28
400K termes x 19 ⇒ 7.6 MB

Exercice : determiner la taille optimale du pointeur pour 400 Mille termes

Compression de la liste de termes : pointeurs par blocs

- Stocker les pointeurs à chaque k termes (Exemple : k=4).
- Besoin de rajouter un octet pour stocker la taille du terme

4coma6combat5combe11combinaison7combler11combustible

Termes	Fréquence	Pointeur vers les listes
	42	
	11	
	235	
	63	
3 octets	*4 octets*	*4 octets*

- On économise 3 pointeurs (9 octets) tous les k=4 termes.
- On dépense 1 octet de plus à chaque mot pour la taille
- On utilise 3+4 octets au lieu 3X4 octets, → économie de 0.5MO
- on réduit la taille → 7,1 Mo

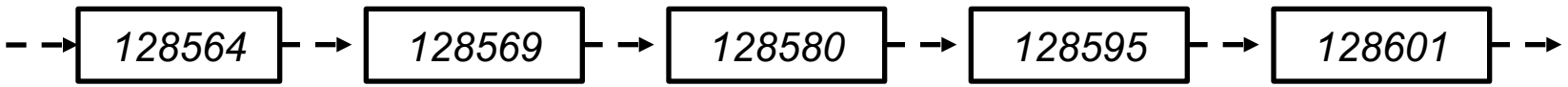
Exercice

- Pourquoi ne pas augmenter k?
- Estimer l'espace nécessaire pour l'index (ce que l'on gagne vis-à-vis des 7,6 MO) pour *k = 4, 8* et *16.*

Compression du *posting (A lire)*

Compression du *posting*

- Le fichier *Posting* est au moins 10 fois plus volumineux que le dictionnaire
  - Le *Posting* est formé de DocId(s) (numéro de document) → un entier codé sur 4 octets
  - Au mieux, pour 1 M de documents, sur $\log_2$ 1 000 000 ≈ 20 bits
- Peu de termes fréquents, beaucoup de termes rares
  - « ***arachnocentric*** » apparaît peut-être une fois dans toute la collection → donc pour une collection d'un million de documents, 20 bits devraient suffire
  - « **the** » apparaît probablement dans tous les documents, donc potentiellement 20bitsX1M=20M de bits pour stocker la liste (c'est trop!!!)



Compression du *posting*

- Les docid(s) du *posting* sont stockés par ordre croissant
  - ***computer***: 33,47,154,159,202 …
- Stocker l'écart (intervalle) entre les docid(s) au lieu des docids .
  - 33,14,107,5,43 …

- L'espoir est de pouvoir stocker les écarts (intervalles) dans moins de 20 bits (moins de bits que si l'on gardait les docIds)

Compression du posting

Ex. trois entrées de postings

	encoding	postings list									
THE	docIDs	...		283042		283043		283044		283045	...
	gaps				1		1		1		...
COMPUTER	docIDs	...		283047		283154		283159		283202	...
	gaps				107		5		43		...
ARACHNOCENTRIC	docIDs	252000		500100							
	gaps	252000	248100								

20 bits, ça reste excessif pour "the"

Compression du posting

- But:
  - Pour ***arachnocentric***, on utilise ~20 bits/écart .
  - Pour ***the***, on peut utiliser ~1 bit/écart.

- → Pour une valeur d'écart l, on veut utiliser aussi peu de bits possible (l'entier au-dessus de $\log_2 l$).
- En pratique, on arrondit à l'octet supérieur

- → Encodage *variable*

Compression du posting

- Encodage variable
  - On consacre 7 bits d'un octet à représenter le nombre (l'écart), et le dernier est le bit de continuation *c*.
  - Si $l \leq 127$, 7 bits suffisent → $c = 1$.
  - Sinon, $c = 0$ et on continue sur l'octet suivant.
  - $c = 1$ signifie toujours que le nombre se termine à cet octet.

Exemple

docIDs	824	829	215406
gaps		5	214577
VB code	00000110 10111000	10000101	00001101 00001100 10110001

Postings stockés comme concaténattion d'octets

00000110 10111000 10000101 00001101 00001100 10110001

Bit de continuation

Pour de petits écarts (5), VB Utilise tout l'octet.

Conclusion

- Indexation est au cœur du processus de RI
  - Permet la sélection des termes importants caractérisant le contenu d'un document
  - Souvent une liste de termes simples → modèle sac de mots
  - Idéalement combinaison linguistique + statistique→ aujourd'hui dominée par les modèles statistiques
- Indexation multimédia (Images, vidéos, …)
  - Contextuelle (prendre le texte qui autour l'objet) → on se ramène à une problématique de RI textuelle
  - Contenu (basé sur le signal)