



# Recherche d'image par le contenu

Master1 – Analyse et traitement d'images - GMIN215

Alexis JOLY - alexis.joly@inria.fr

# Plan

**Recherche d'information par le contenu I:** principes, applications, malédiction de la dimension, méthodes de partitionnement et de recherche

**Recherche d'information par le contenu II:** Présentation détaillée des arbres KD aléatoires (randomized kd-trees), Présentation détaillée de LSH et ses variantes, Présentation de deux librairies (FLANN et VLFeat), KNN-graphs

# Introduction, Principes et Applications de la recherche par le contenu

# Introduction

## Recherche d'Information uniquement le contenu t



### Pas de reconnaissance d'image pour Google.

L'unique objectif du test est de valider le fait que l'image ci-dessus se positionnera dans GG images pour la requête « pizza aux anchois ». Il n'y a pas de raison que ce ne soit pas le cas. Pour le moment, Google ne reconnaît pas les images, mais il prend en compte l'environnement textuel de celles-ci pour en déterminer le sujet. Et l'environnement ici, il est bien plus orienté **pizzeria** que mannequinat.

Maintenant, au train où vont les évolutions techniques de la reconnaissance d'image, il y a fort à parier que les moteurs sauront un de ces jours différencier la jolie dame ci-dessus d'une vulgaire pizza.

### Que dire d'autre sur cette pizza ?

On peut dire comme pour notre test que cette recette de pizza ne fait pas l'unanimité. En effet, les anchois sont accueillis comme un met délicat par certains, et comme le pire pour les papilles par d'autres.

Pour les hurluberlus qui voudraient en savoir plus sur ce sujet, [passez voir ici](#), mais globalement, on s'en fiche un peu, tout ce qui nous intéresse, c'est de voir comment notre pizza va prendre chez le roi des moteurs.

### La recette de la pizza aux anchois

Pour les maniaques à qui nous aurions donné faim, il faut :

- des tomates,
- **des anchois** (forcément hein !)
- de l'ail,
- du sel du poivre,
- un peu d'huile d'olive,

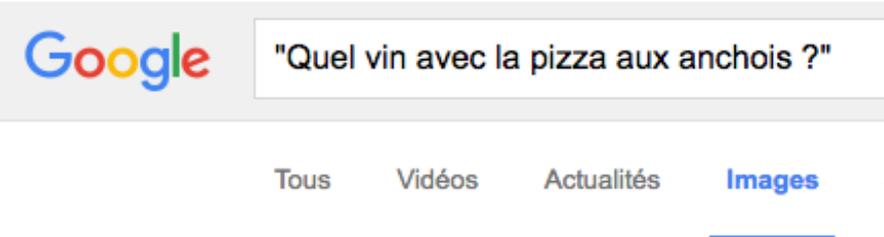
vous mélangez tout, et si ce n'est pas bon (enfin... chez nous, lors du test, ce n'était pas bon), vous allez chez Pizza Hut, et il vous livre une pizza bien meilleure. On en trouve aussi dans toutes les bonnes pizzerias.

Bon, en même temps, nous, on fait du SEO, on n'est pas pizzaiolo.

### Quel vin avec la pizza aux anchois ?

Là, on s'y connaît un peu mieux 😊

Et puis il faut dire que l'on est super bien conseillé. Voilà les vins que vous pouvez choisir avec votre pizza. Pas trop cher, ou du moins pour toutes les bourses, il n'y a qu'à sélectionner et commander.



# Introduction

L'indexation et la recherche par le contenu visuel exploite l'information contenue dans les images



http://app2.belga.be:8086 - Picture details - Mozilla Firefox

Picture details Picture 554 of 872 |< < > >|

Name: AUSTRALIA TENNIS AUSTRALIAN OPEN

**Image:** 

**Caption:** epa00911824 Kim Clijsters of Belgium jubilates after winning against Martina Hingis of Switzerland during their quarterfinals match at the Australian Open tennis tournament in Melbourne, Australia, Wednesday 24 January 2007. Clijsters won 3-6, 6-4, 6-3. EPA/BARBARA WALTON

**Date:** 24/01/2007 07:16

**Source:** EPA

**Dimensions:** 1776 x 2048

**Photo:** 24/01/2007 07:16

**Category:** SPO

**Sub Category:** TENNIS

**City:** MELBOURNE

**Country:** Australia

**ID:** 5946049

**Theme:** sports

Done

# Introduction

## Approches Content-to-text

- Reconnaissance d'objets, de visages, de textes (OCR)
- Speech-to-text, reconnaissance de locuteurs

Utilisation d'outils classiques d'indexation et de recherche de données

## Approches Content-based

- Utilisation de **descripteurs** (*signatures* ou *features*) = des vecteurs calculés à partir du contenu visuel ou audio caractérisant les couleurs, les formes, les fréquences, etc.
- Requête par l'exemple (Shazam, Google similarity search, Goggles)
- Query-by-window, Query-by-sketch, Query-by-singing, Query-by-tapping

# Introduction

## Approches Content-to-text

→ Reconnaissance d'objets, de visages, de textes (OCR)

→ Speed

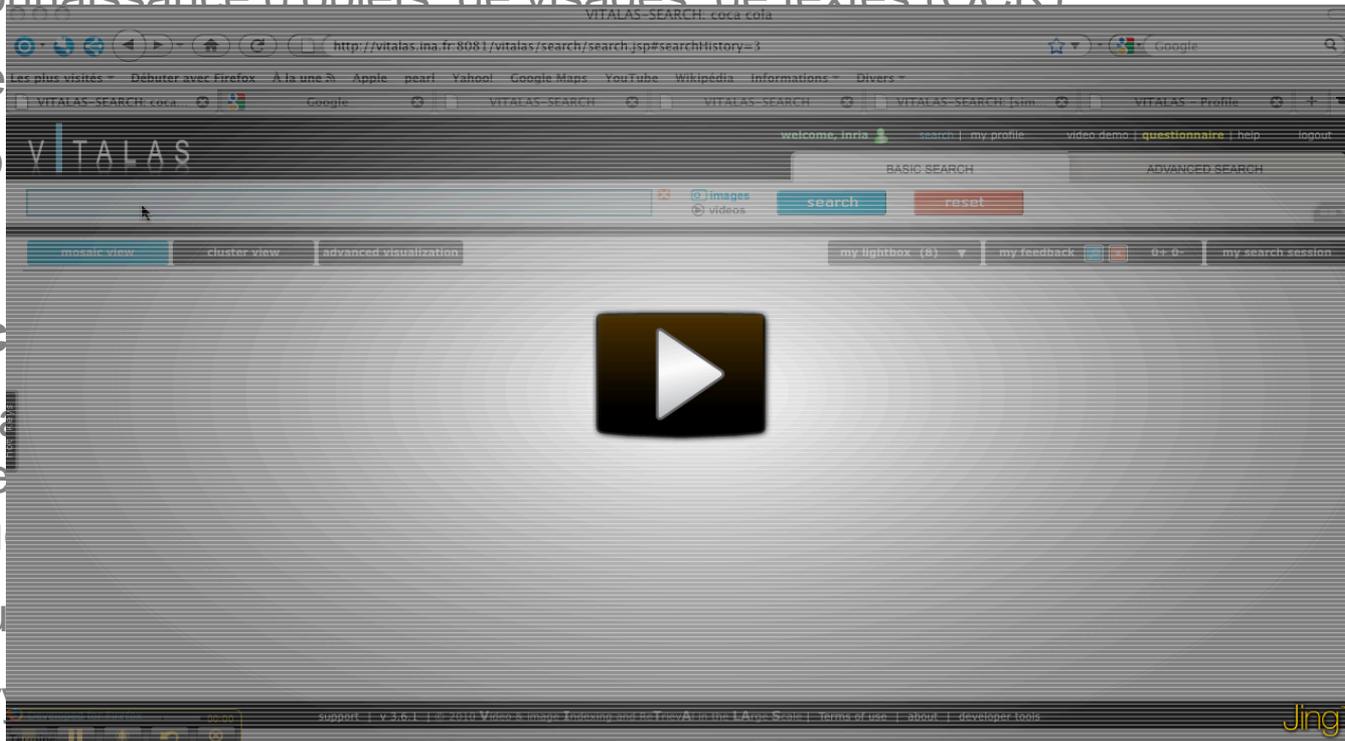
Utilisation

## Approches

→ Utilisation de contenu fréquent

→ Requêtes

→ Query

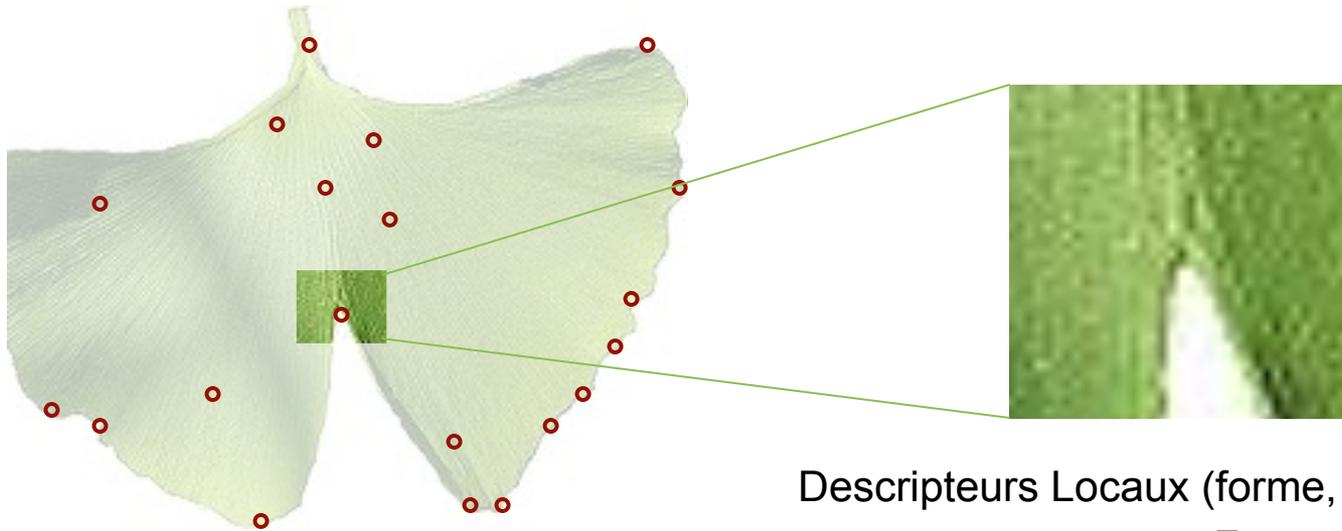


ir du

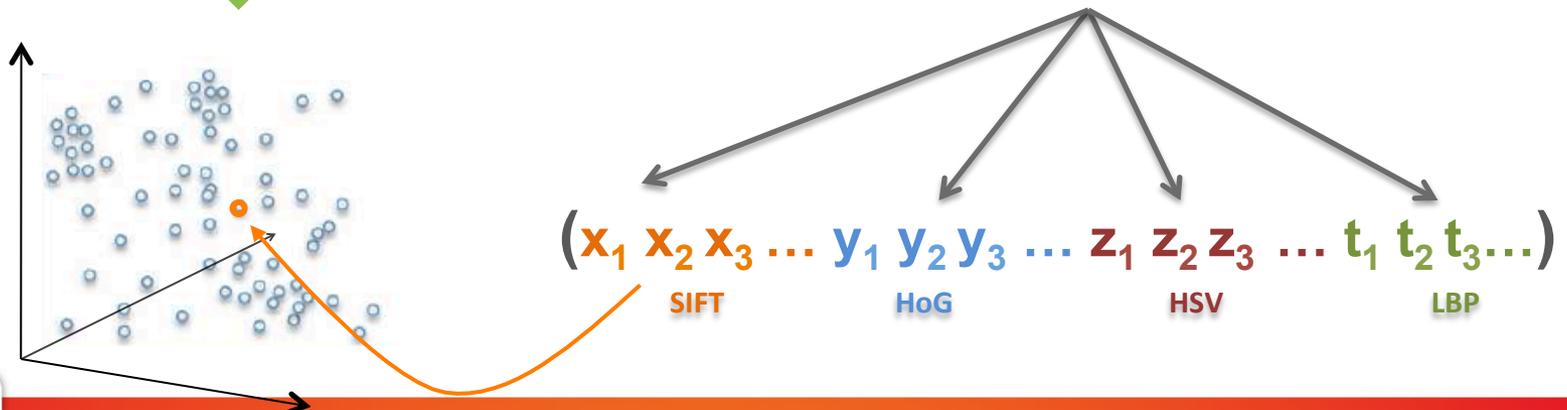
ping

# Principe 1/4

## Des signatures du contenu visuel

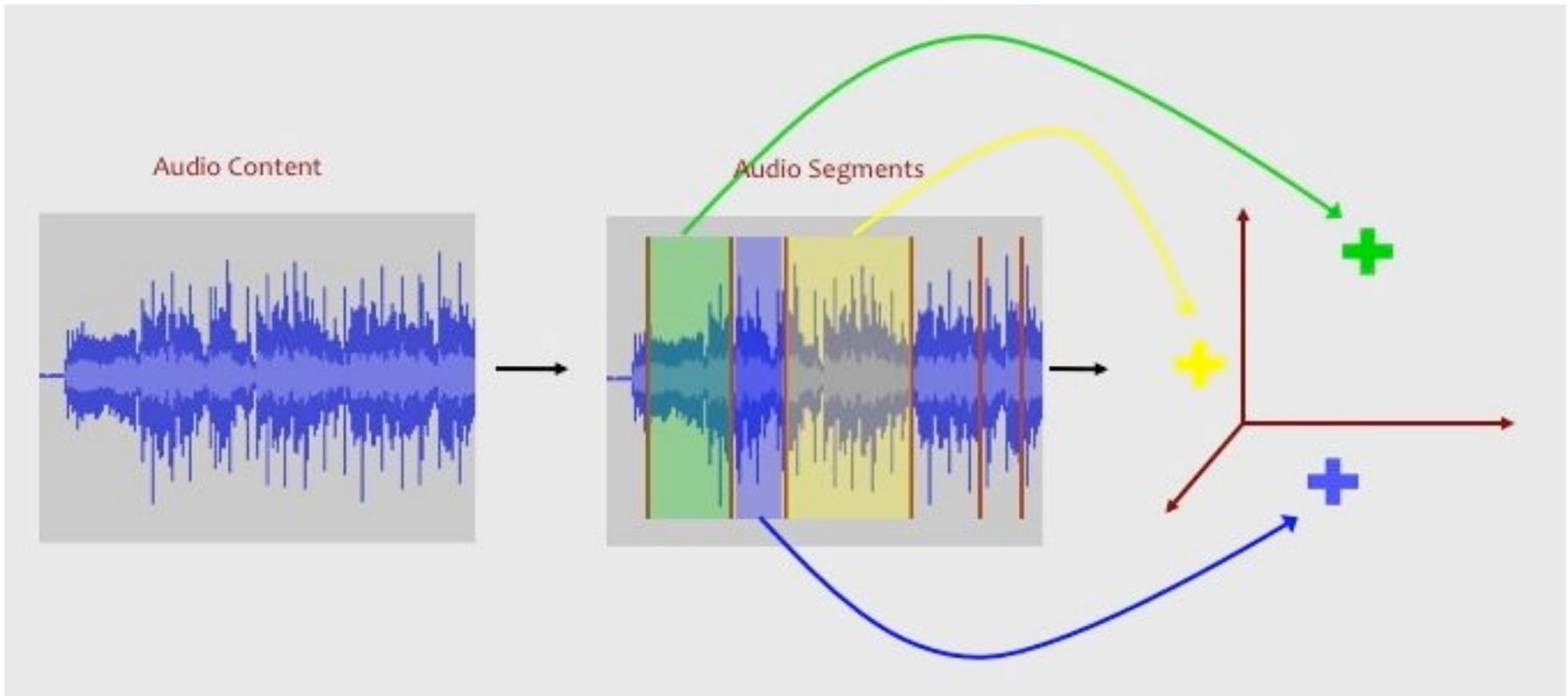


Descripteurs Locaux (forme, couleur, texture)  
=  
Vecteurs de grande Dimension (128, 1024, etc.)



# Principe 2/4

## ou du contenu audio



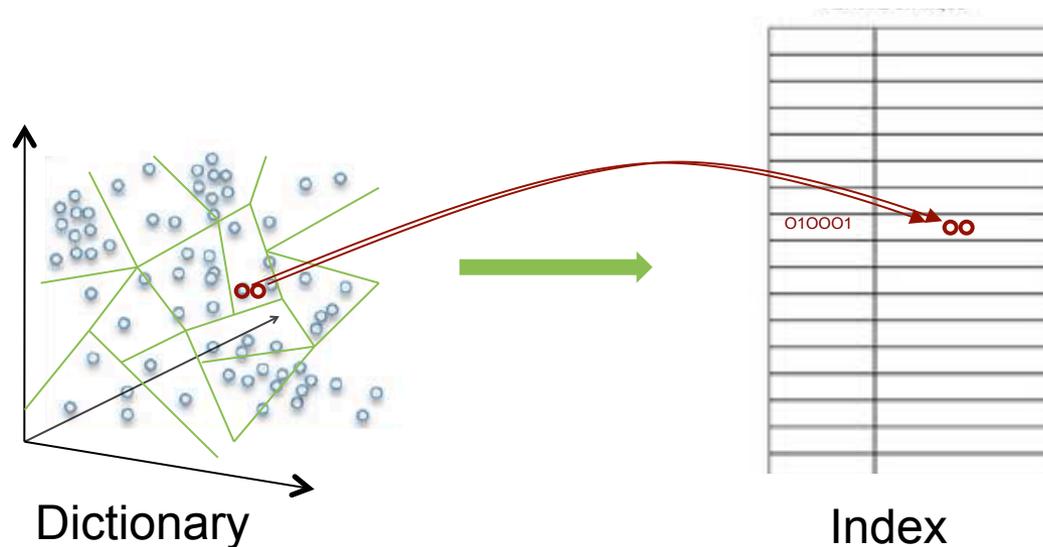
# Principe 3/4

## Des techniques d'indexation

**Objectif:** comparer efficacement une signature requête avec les signatures de la base

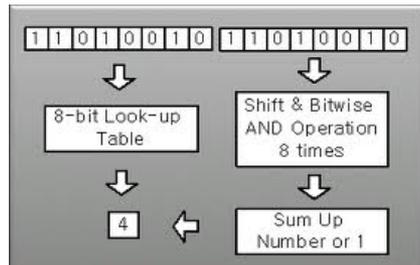
**Problème:** des centaines de milliards de comparaisons de vecteurs si on les traite exhaustivement + croissance exponentielle

**Solution =** structures d'indexation, VQ, hachage, compression, codage, encapsulation, regroupement, etc.



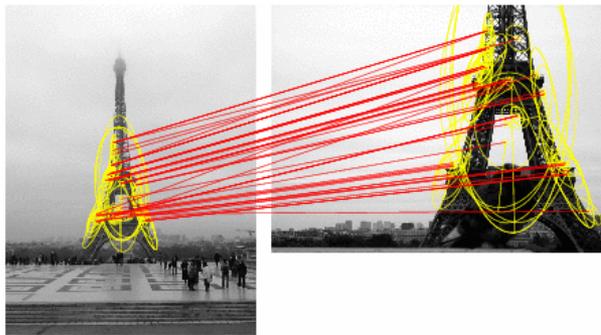
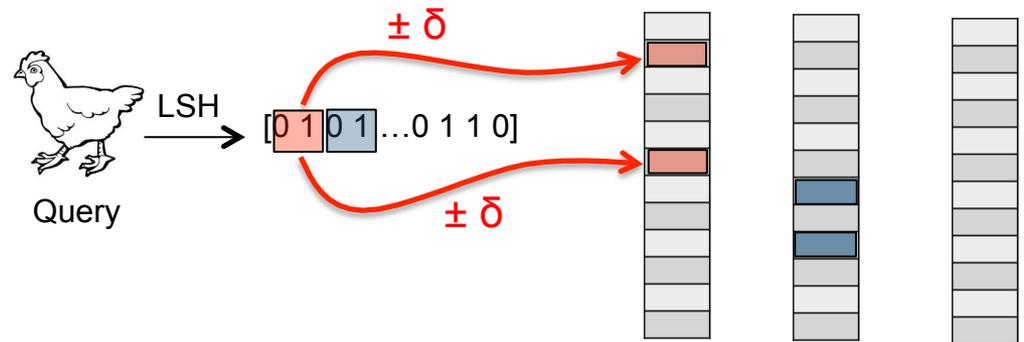
# Principe 4/4

## Des algorithmes de recherche par similarité efficaces et scalables

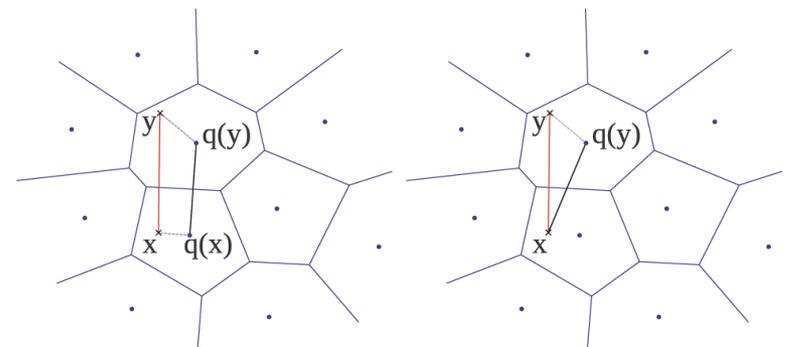


LUT-based Hamming distance

multi-probe queries in hash tables



Weak geometry ( $\sigma, \theta$ )



Asymmetric distance

# Les applications 1/5

## Détection de copies vidéo par le contenu

Monitoring de flux TV ou Web en temps-réel

15 M de vidéos indexées chez Youtube, 500 000 h de TV à l'INA

Forte robustesse aux attaques & transformations



(a) *Alexandrie*. 1978 (c).



(b) *Samedi et Compagnie* 1970 (c) ORTF.



# Les applications 2/5

## Identification de chansons / musiques

Shazam > 10 Millions de titres

MusicID > 40 Millions titres

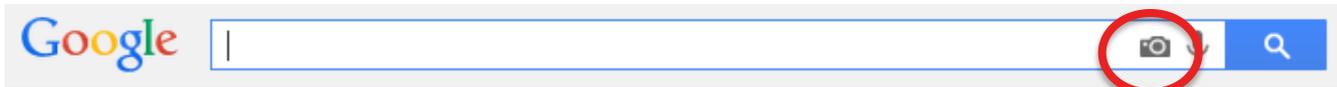
## Query-by-humming (plus compliqué)

SoundHound >10 Millions de titre



# Les applications 3/5

## Recherche d'images similaire (ou near-duplicates)



[mediateur-Google.jpg](#)  
[blog.youseemii.fr](#) Partager  
950 x 534 - Google Archives - Yo  
[Images similaires](#) Autres tailles

Google > 50 Milliards d'images  
LTU > 100 Millions images

## Reverse image search

Tineye > 3 Milliards d'images

## Mobile search

Kooaba > 10M images  
Goggles > ??

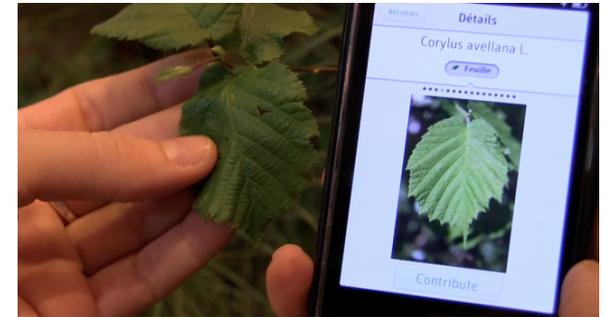


## Les applications 4/5

L'identification des végétaux

Pl@ntNet > 3800 espèces

LeafSnap > 250 espèces



L'identification des animaux

Encore à l'état de recherche

Reconnaissance de chants d'oiseaux

Vidéosurveillance de poissons dans les récifs

Suivi de baleines (images de la queue)

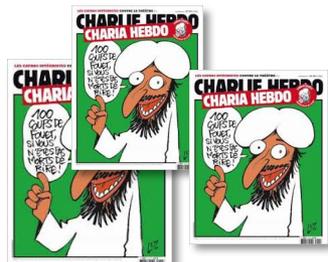




# Les applications 5/5

## Découverte d'évènements par contenu visuel

1. Dans les médias sociaux
2. A travers différents médias (TV, web , AFP)



Arte TV

Lemonde



corto74.blogspot.com

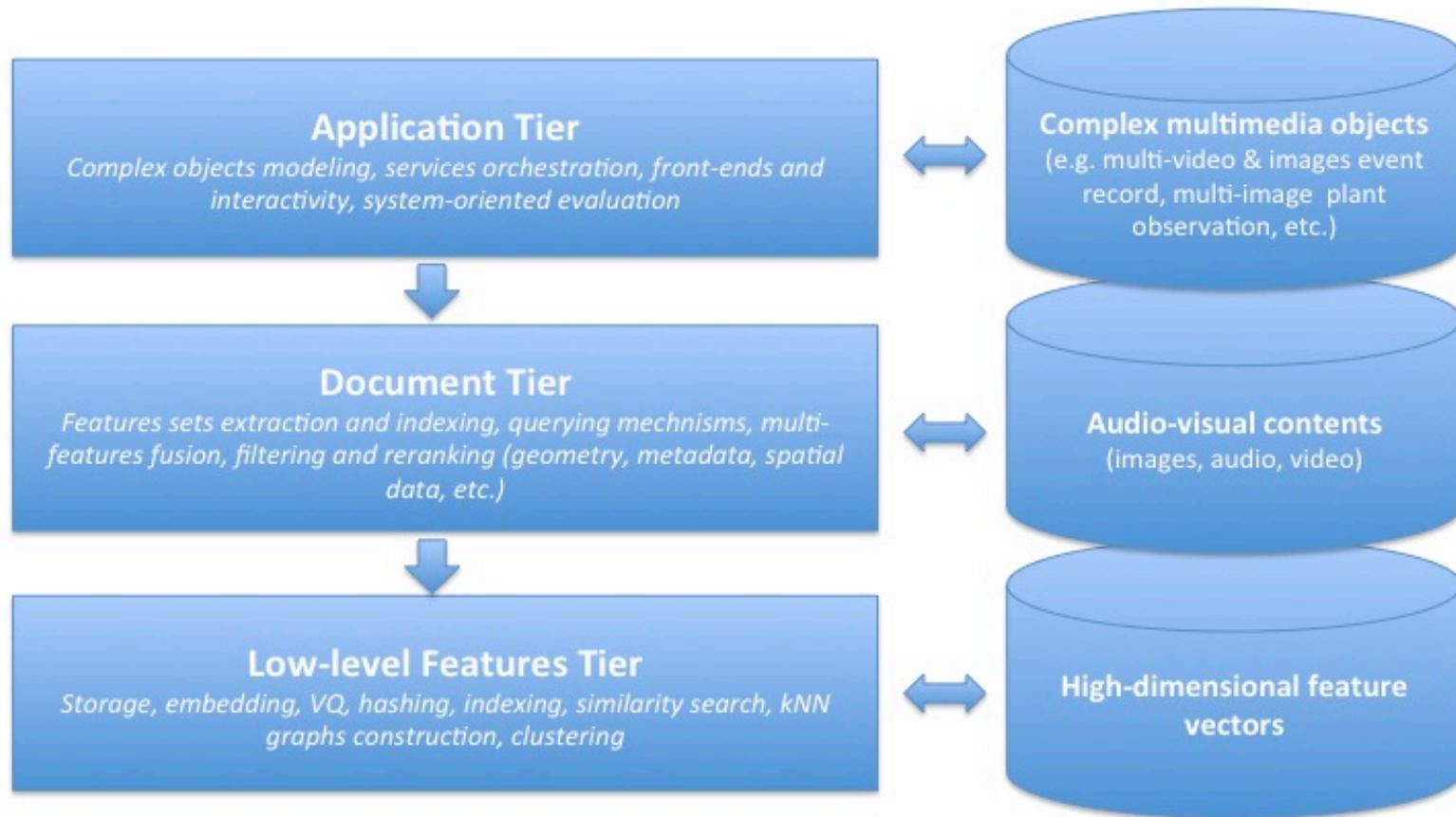


2011-10-17 to 2011-10-24

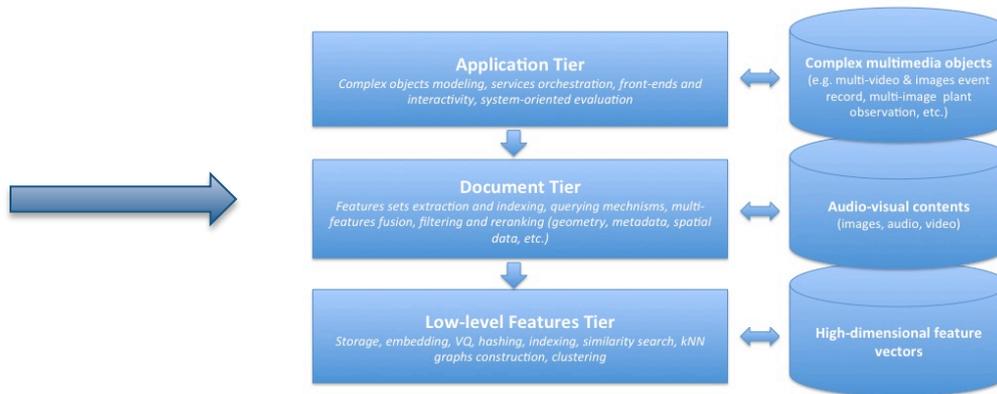
Le régime de Mouammar Khadafi aura duré 42 ans. Un régime inauguré à la fin des années 60 par un jeune officier qui deviendra un dictateur redouté dans son pays et au delà des frontières libyennes, notamment en raison de son soutien au terrorisme.

keywords : [ Libye ] [ Mouammar Khadafi ] [ OTAN ] [ FRA ]

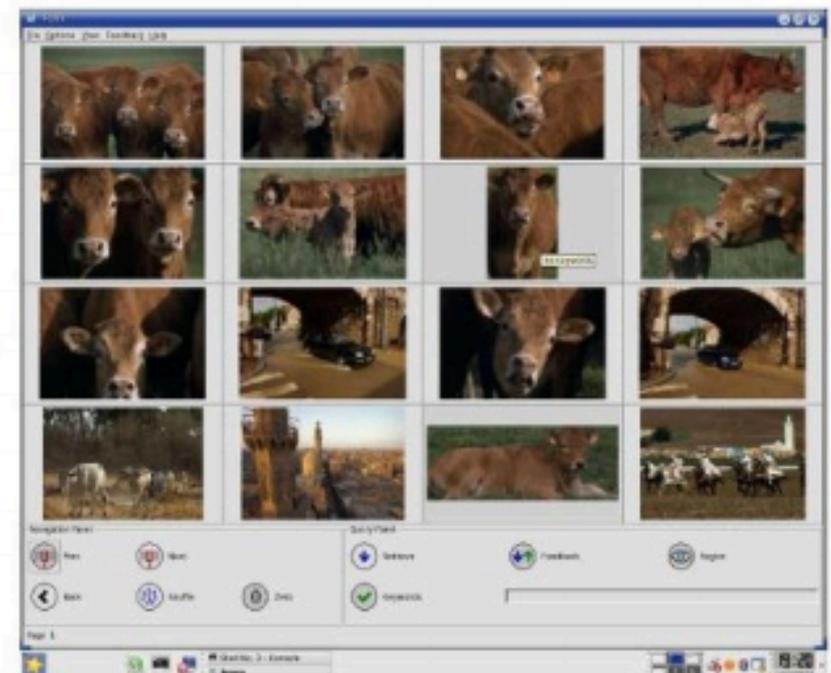
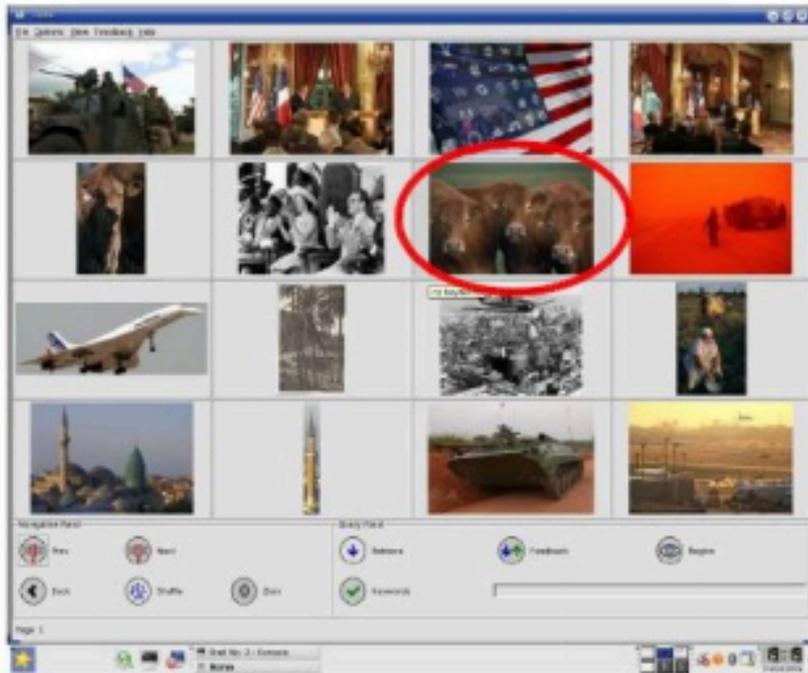
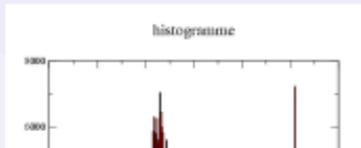
# Architecture



# Quelques descripteurs de contenu et mesures de similarité associées



- Cas d'école: Histogramme HSV



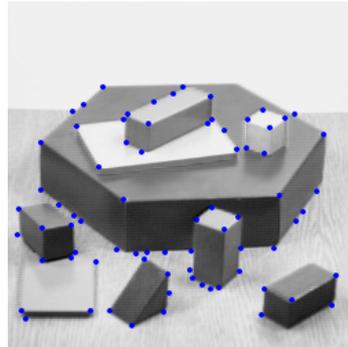
# La détection de points ou de régions d'intérêt

## Détection de points d'intérêt par différence de Gaussienne

Grille dense



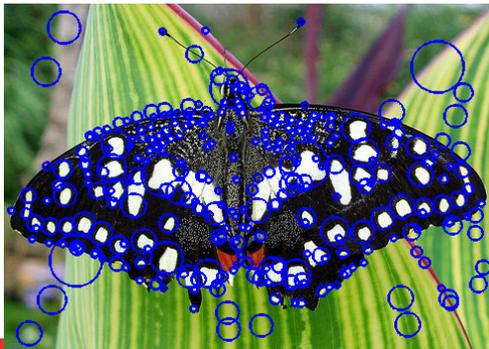
Détecteur de Harris



Maximally Stable  
Extremal Regions



Difference of Gaussians



Superpixels



# Les descripteurs

```

133.92 135.88 14.38 -2.732
 3 12 23 38 10 15 78 20 39 67 42 8 12 8 39 35 118 43 17 0
 0 1 12 109 9 2 6 0 0 21 46 22 14 18 51 19 5 9 41 52
65 30 3 21 55 49 26 30 118 118 25 12 8 3 2 60 53 56 72 20
 7 10 16 7 88 23 13 15 12 11 11 71 45 7 4 49 82 38 38 91
118 15 2 16 33 3 5 118 98 38 6 19 36 1 0 15 64 22 1 2
 6 11 18 61 31 3 0 6 15 23 118 118 13 0 0 35 38 18 40 96
24 1 0 13 17 3 24 98

```

# Calculés automatiquement

```

132.36 99.75 11.45 -2.910
94 32 7 2 13 7 5 23 121 94 13 5 0 0 4 59 13 30 71 32
 0 6 32 11 25 32 13 0 0 16 51 5 44 50 0 3 33 55 11 9
121 121 12 9 6 3 0 18 55 60 48 44 44 9 0 2 106 117 13 2
 1 0 1 1 37 1 1 25 80 35 15 41 121 3 0 2 14 3 2 121
51 11 0 20 93 6 0 20 109 57 3 4 5 0 0 28 21 2 0 5
13 12 75 119 35 0 0 13 28 14 37 121 12 0 0 21 46 5 11 93
29 0 0 3 14 4 11 99
102.33 26.00 13.65 2.815

```

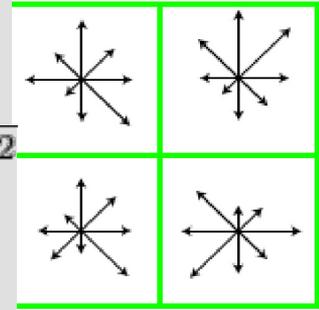
- SIFT (Lowe, 2004)

que L2

Métrique pour comparer deux vecteurs SIFT =  
Distance L2 = distance Euclidienne

$$d(\mathbf{p}, \mathbf{q}) = d(\mathbf{q}, \mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \dots + (q_n - p_n)^2}$$

$$= \sqrt{\sum_{i=1}^n (q_i - p_i)^2}$$



- Autres descripteurs  
Binary Pattern

```

19 10 3 1 29 94 54 54
88.61 134.04 13.25 2.721
27 56 20 6 9 21 12 17 33 120 114 6 4 8 8 0 41 106 23 9
19 42 3 0 45 54 13 0 14 26 1 3 45 12 6 22 24 22 29 35
61 52 8 1 35 92 55 23 120 106 0 0 16 24 5 35 92 4 0 0
33 40 9 30 81 74 22 31 4 2 5 8 52 77 34 9 39 37 15 32
120 38 3 0 5 17 28 120 41 5 19 10 7 32 89 63 34 27 4 0
 0 6 33 17 40 60 29 0 0 8 23 22 120 113 79 5 0 0 2 16
 9 21 120 58 0 1 10 3

```

- SIFT + some (

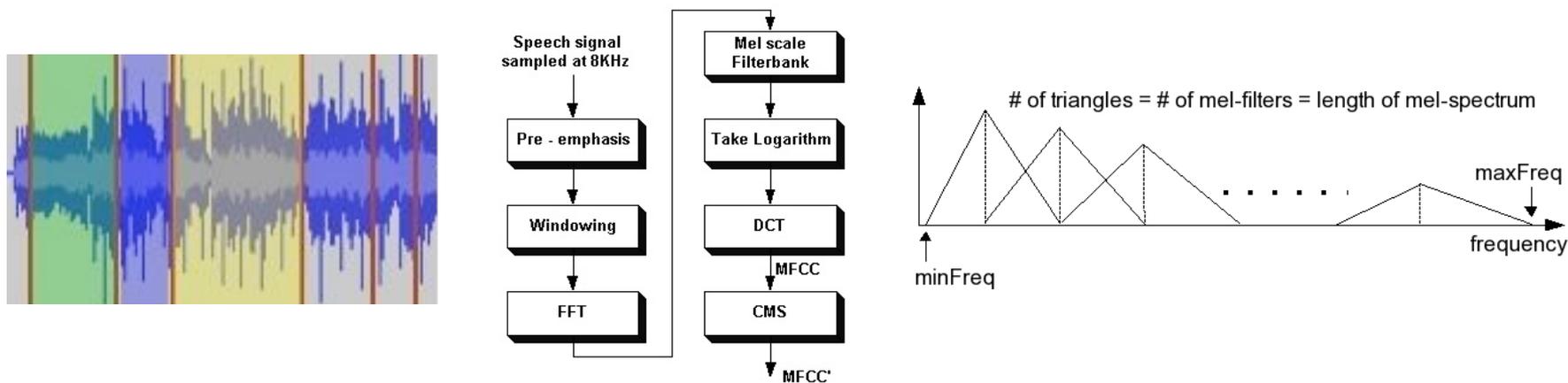
BRIEF, Local



# Les descripteurs audio les plus utilisés

Calculés sur segments fixes ou variables

- MFCC, dimension = 13 (valeurs float), distance L2



- **Autres:** Linear Predictor Coefficients, Line Spectral Frequency, Loudness coefficients, OBSI, etc.
- MFCC + many others available in **Yaafe**

# Représentation vectorielle de documents textes

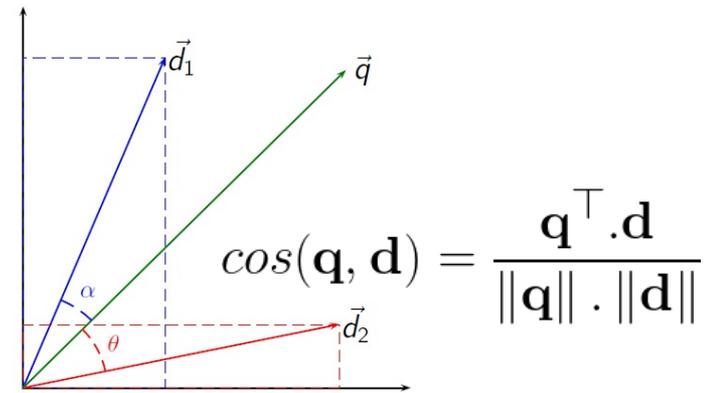
Documents et requêtes représentés par des vecteurs de fréquence d'apparition des mots

$$d_j = (w_{1,j}, w_{2,j}, \dots, w_{t,j})$$
$$q = (w_{1,q}, w_{2,q}, \dots, w_{t,q})$$

Taille du dictionnaire (nombre de mots possibles) = dimension de l'espace de description (jusqu'à plusieurs centaines de milliers)

Mesure de similarité = **Produit scalaire** ou **Cosine measure**

$$\mathbf{q}^\top \cdot \mathbf{d} = \sum_{i=1}^n q_i \cdot d_i = \|\mathbf{q}\| \cdot \|\mathbf{d}\| \cdot \cos(\mathbf{q}, \mathbf{d})$$



NB: La grande majorité des composantes étant nulles, on utilise en pratique une représentation parcimonieuse des vecteurs, e.g:

q: (id\_mot1, frequence\_mot1) (id\_mot2, frequence\_mot2)

# Représentation vectorielle de documents textes

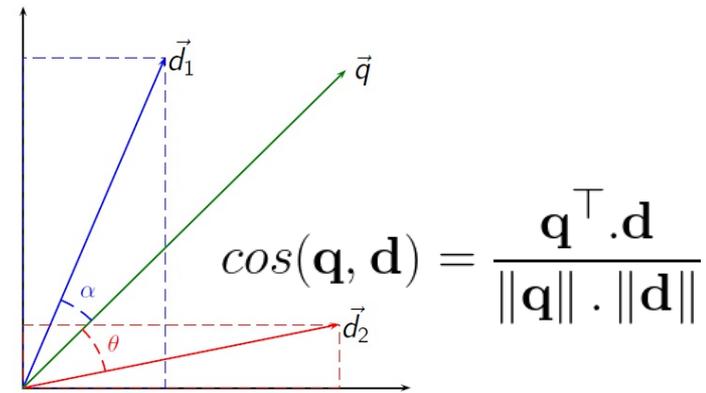
Documents et requêtes représentés par des vecteurs de fréquence d'apparition des mots

$$d_j = (w_{1,j}, w_{2,j}, \dots, w_{t,j})$$
$$q = (w_{1,q}, w_{2,q}, \dots, w_{t,q})$$

Taille du dictionnaire (nombre de mots possibles) = dimension de l'espace de description (jusqu'à plusieurs centaines de milliers)

Mesure de similarité = **Produit scalaire** ou **Cosine measure**

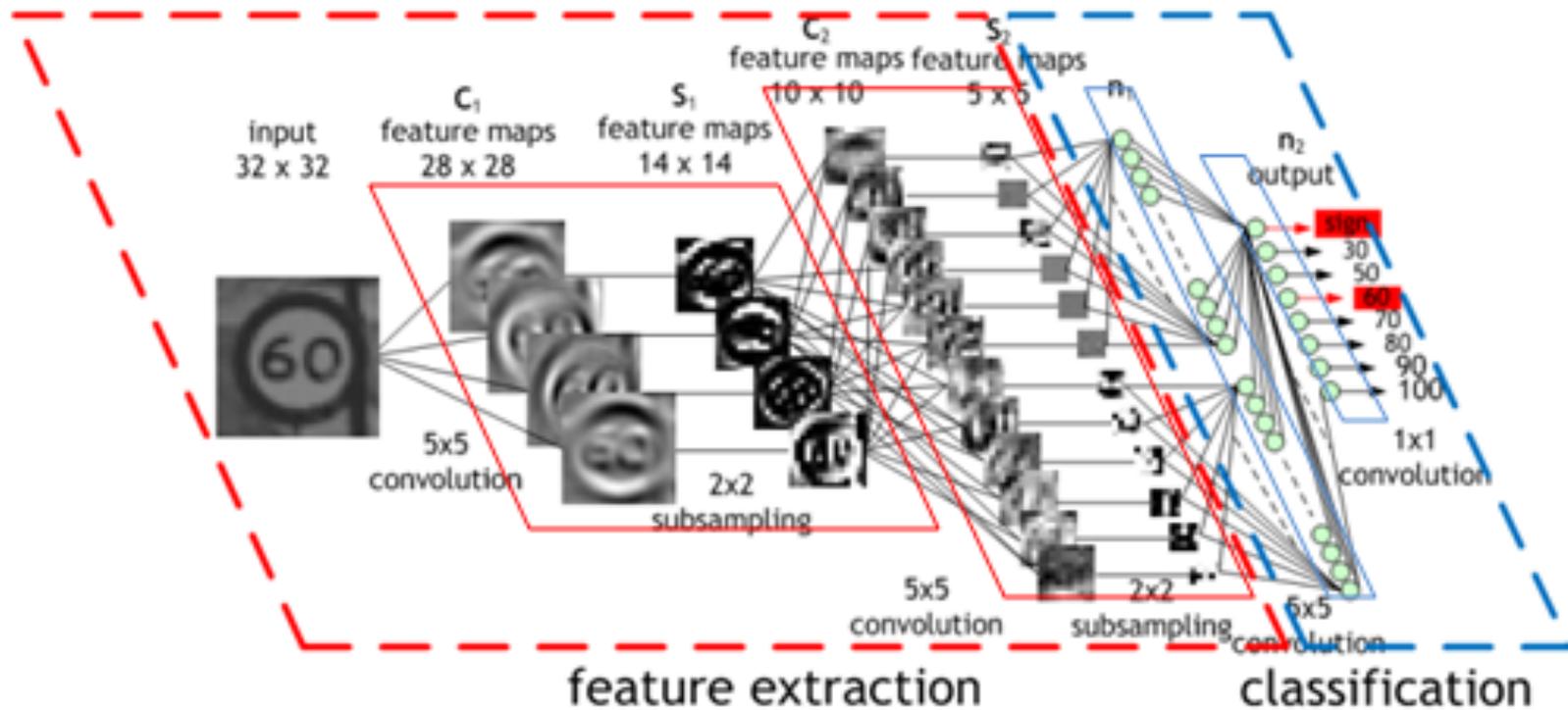
$$\mathbf{q}^\top \cdot \mathbf{d} = \sum_{i=1}^n q_i \cdot d_i = \|\mathbf{q}\| \cdot \|\mathbf{d}\| \cdot \cos(\mathbf{q}, \mathbf{d})$$



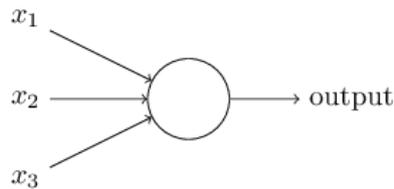
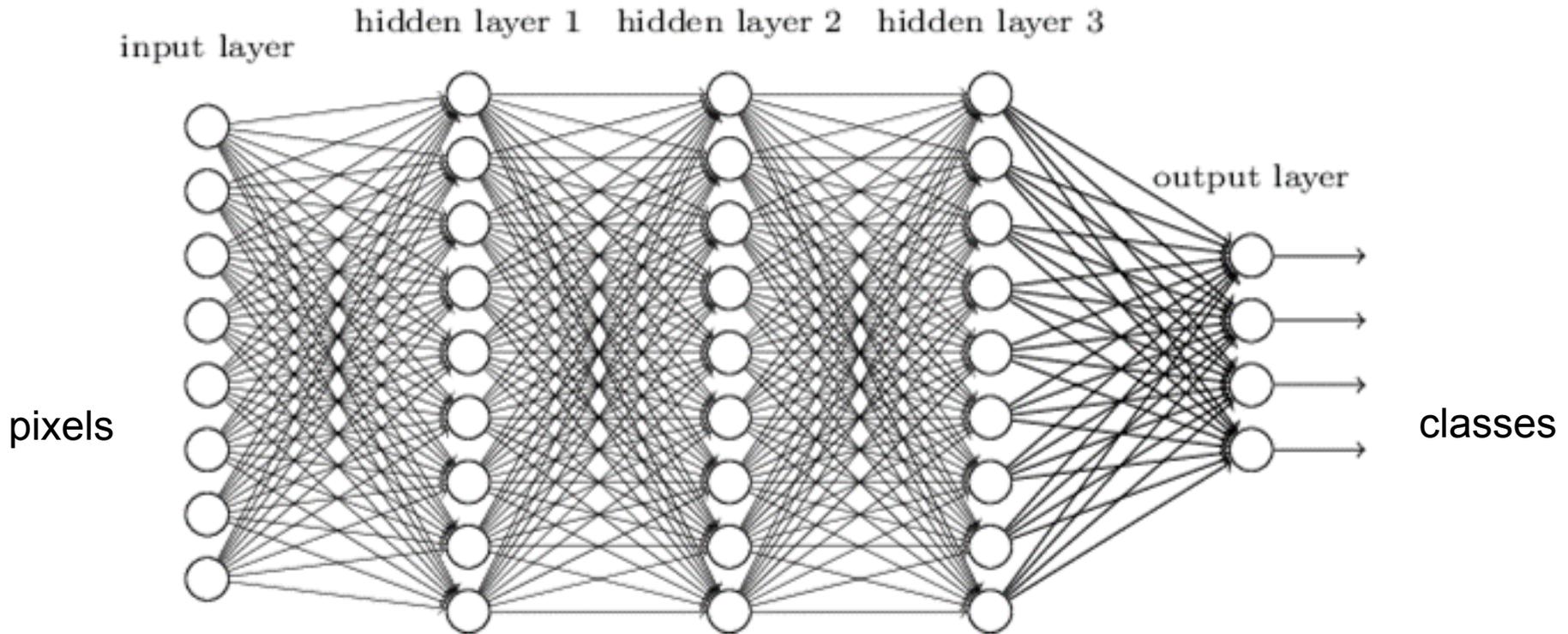
NB: La grande majorité des composantes étant nulles, on utilise en pratique une représentation parcimonieuse des vecteurs, e.g:

q: (id\_mot1, frequence\_mot1) (id\_mot2, frequence\_mot2)

# Apprentissage de représentations visuelles par réseau de neurone convolutionnel



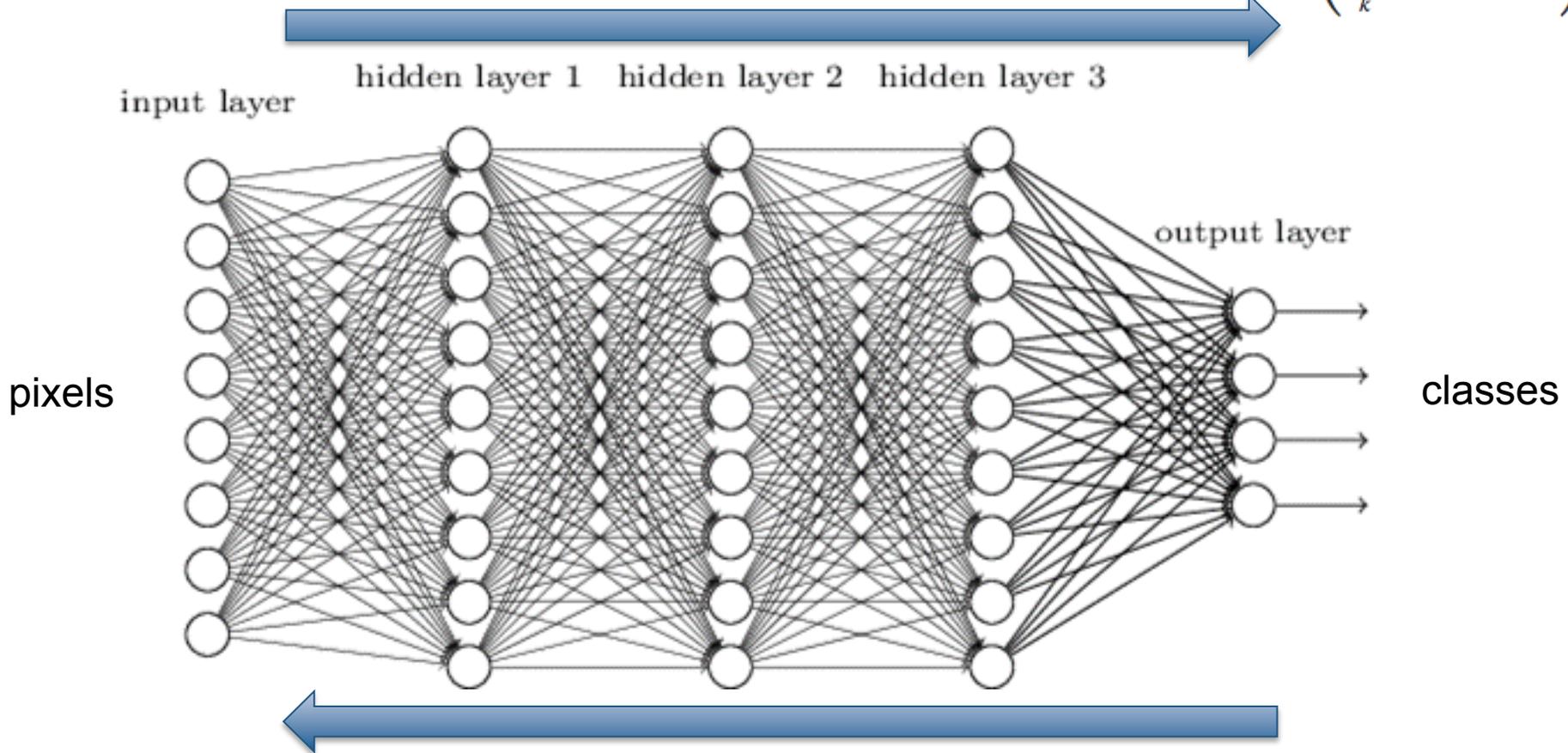
# Réseau de neurones profond



$$\frac{1}{1 + \exp(-\sum_j w_j x_j - b)}$$

# Réseau de neurones profond

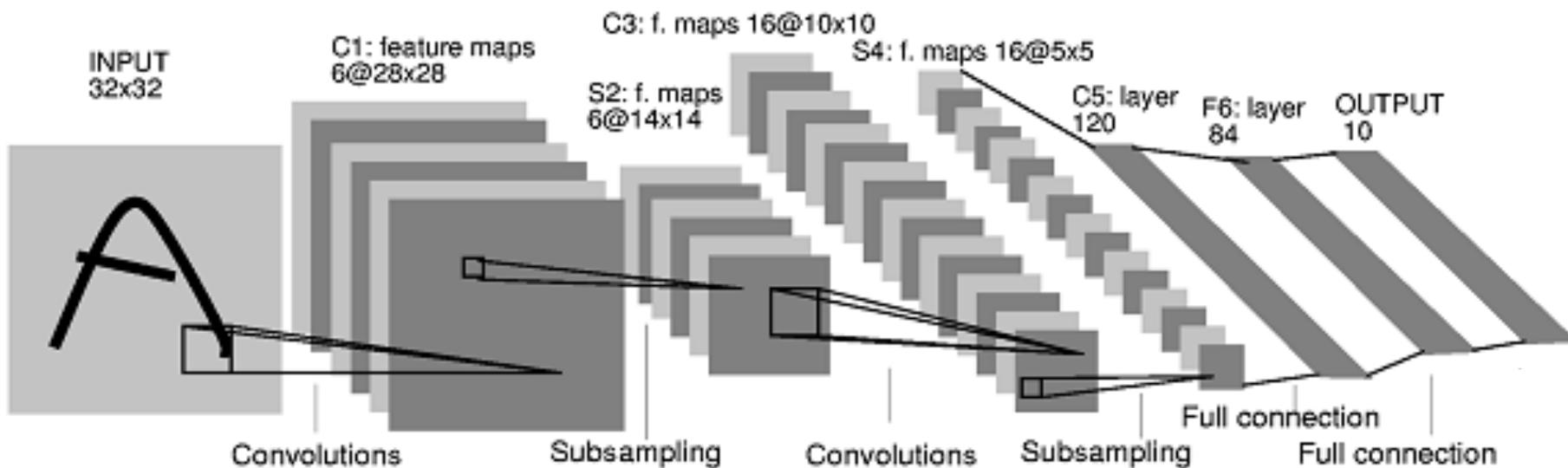
1. Feed forward: recursive activation function  $a_j^l = \sigma \left( \sum_k w_{jk}^l a_k^{l-1} + b_j^l \right)$



2. Back-propagate error to update weights thanks to a derivative cost function such as

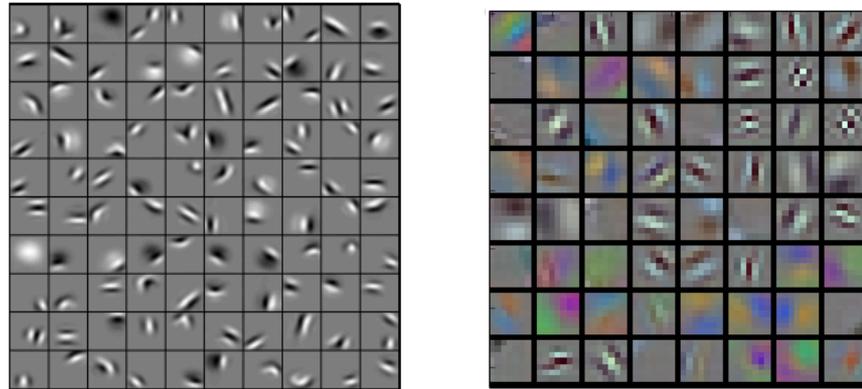
$$C = \frac{1}{2n} \sum_x \|y(x) - a^L(x)\|^2,$$

# Réseau de neurone convolutionnel

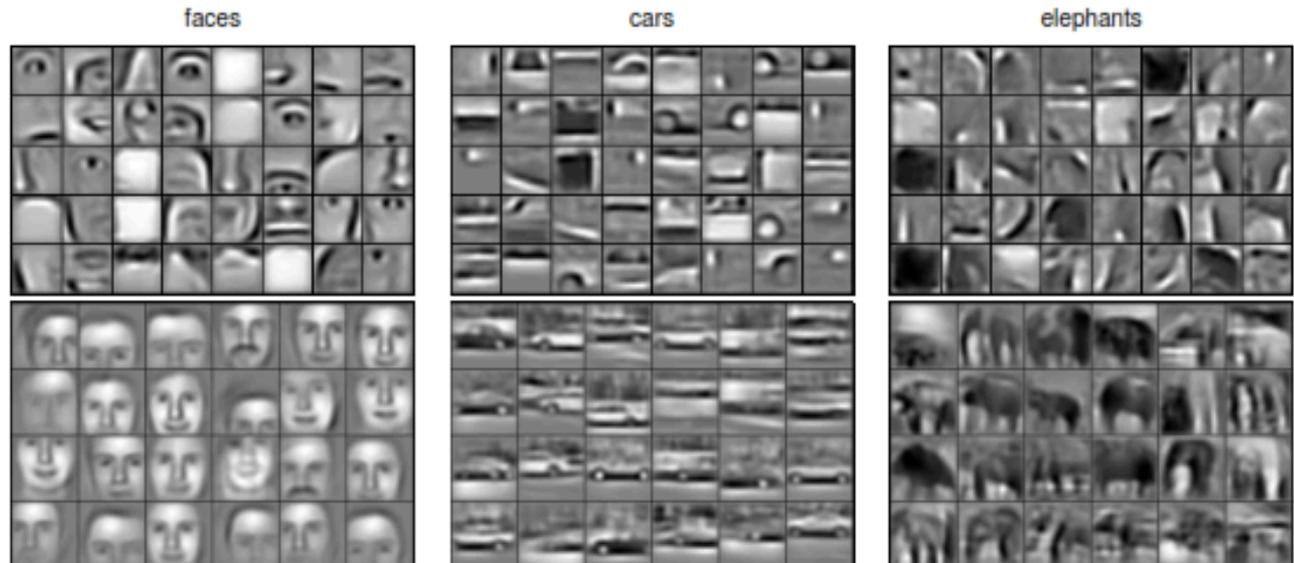


# Réseau de neurone convolutionnel

Pixel layer  
→ Low-level features

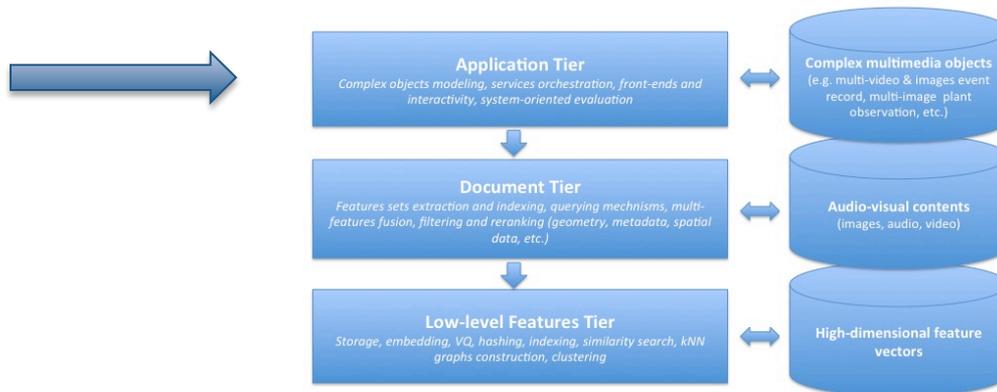


Middle Layer  
→ Mid-level features



Top layer 3  
→ High-level features

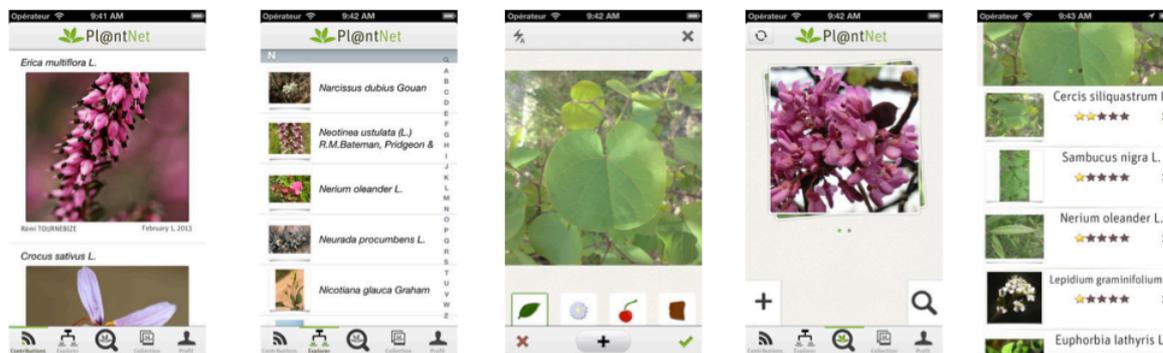
# Un exemple de chaîne complète d'indexation et de recherche par le contenu



# Pl@ntNet: un exemple de système réel de recherche par le contenu visuel

## L'application mobile Pl@ntNet

iPhone



Version Publique

108,000	images
4500	espèces

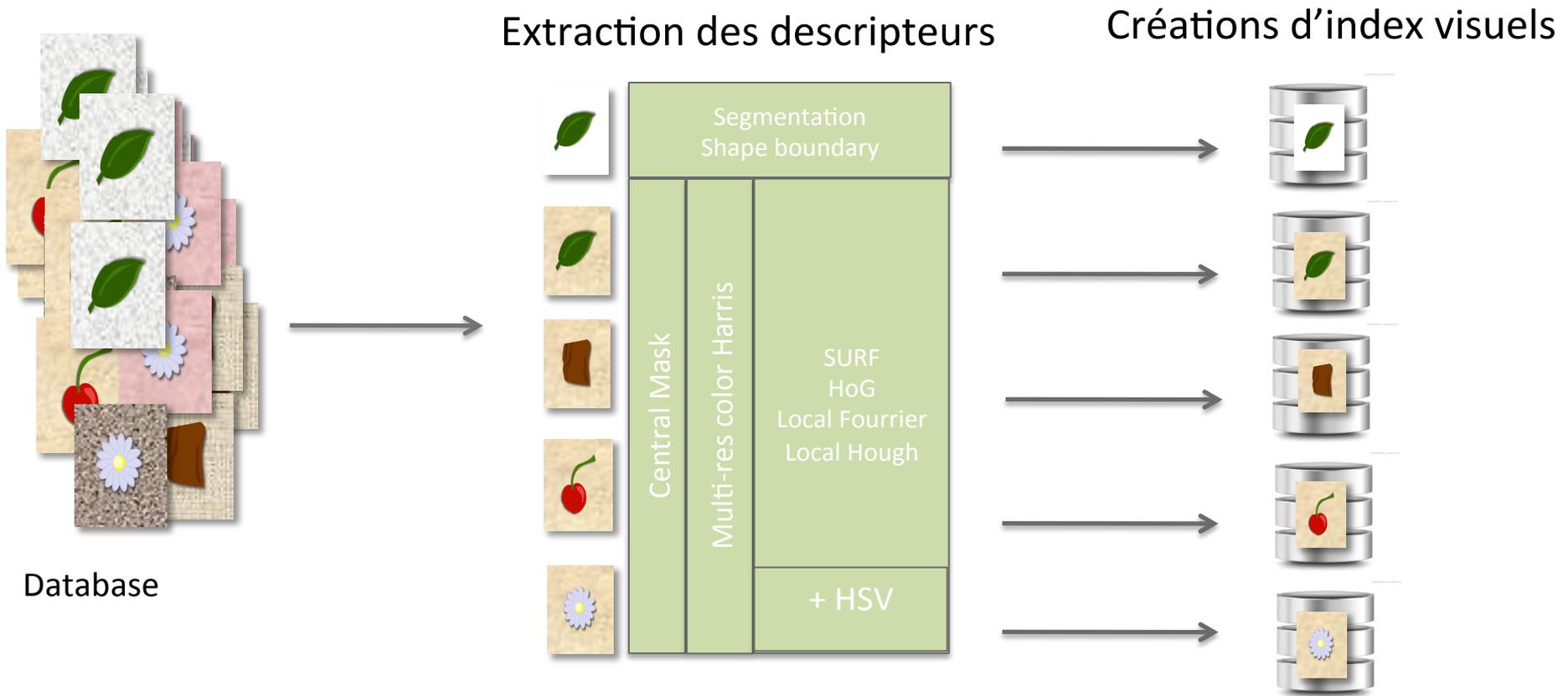


**BETA**

140,000	images
6K espèces	= flore française

# Pl@ntNet: un exemple de système réel de recherche par le contenu visuel

## Indexation



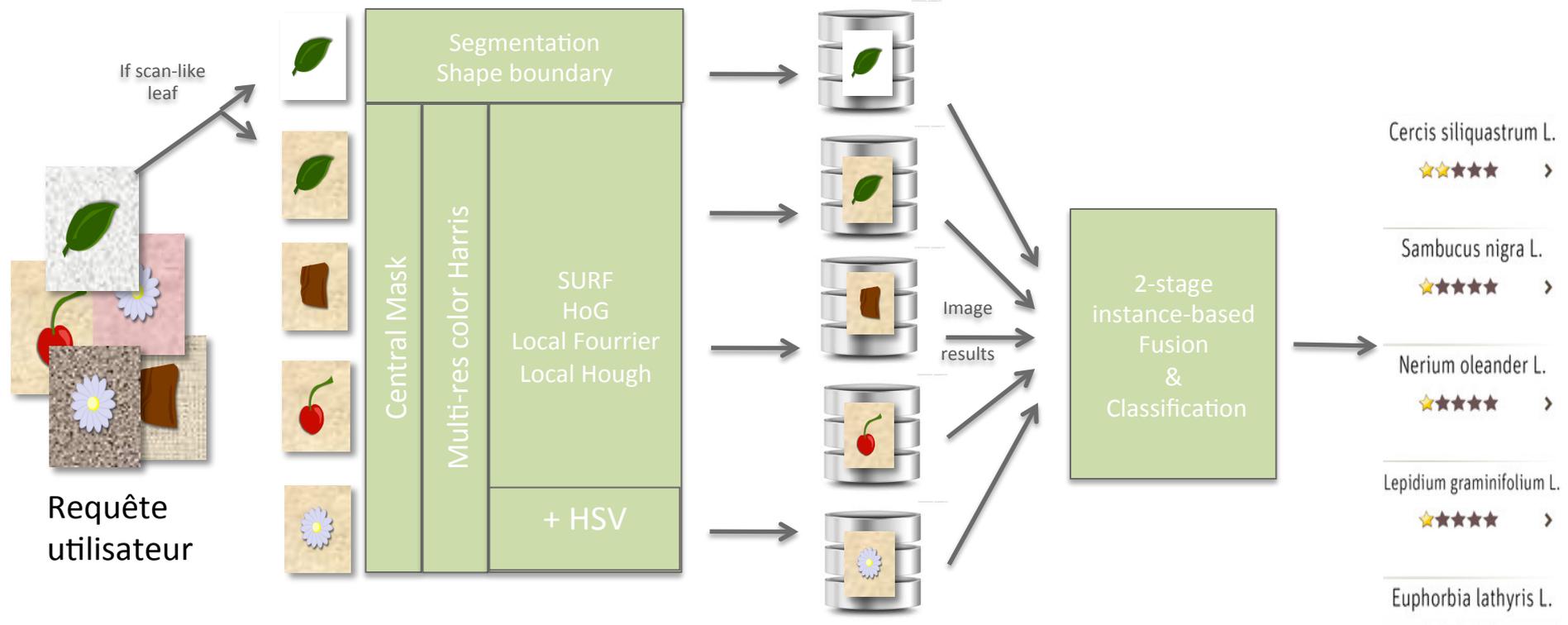
# PI@ntNet: un exemple de système réel de recherche par le contenu visuel

## Recherche

## Extraction des descripteurs

## Recherche dans les index visuels

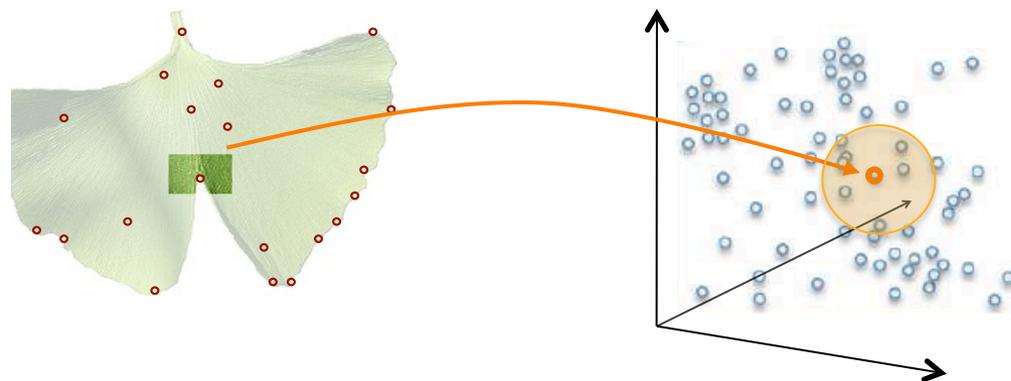
## Species ranked list



# PI@ntNet: un exemple de système réel de recherche par le contenu visuel

## Recherche d'une image dans un index visuel (appariement)

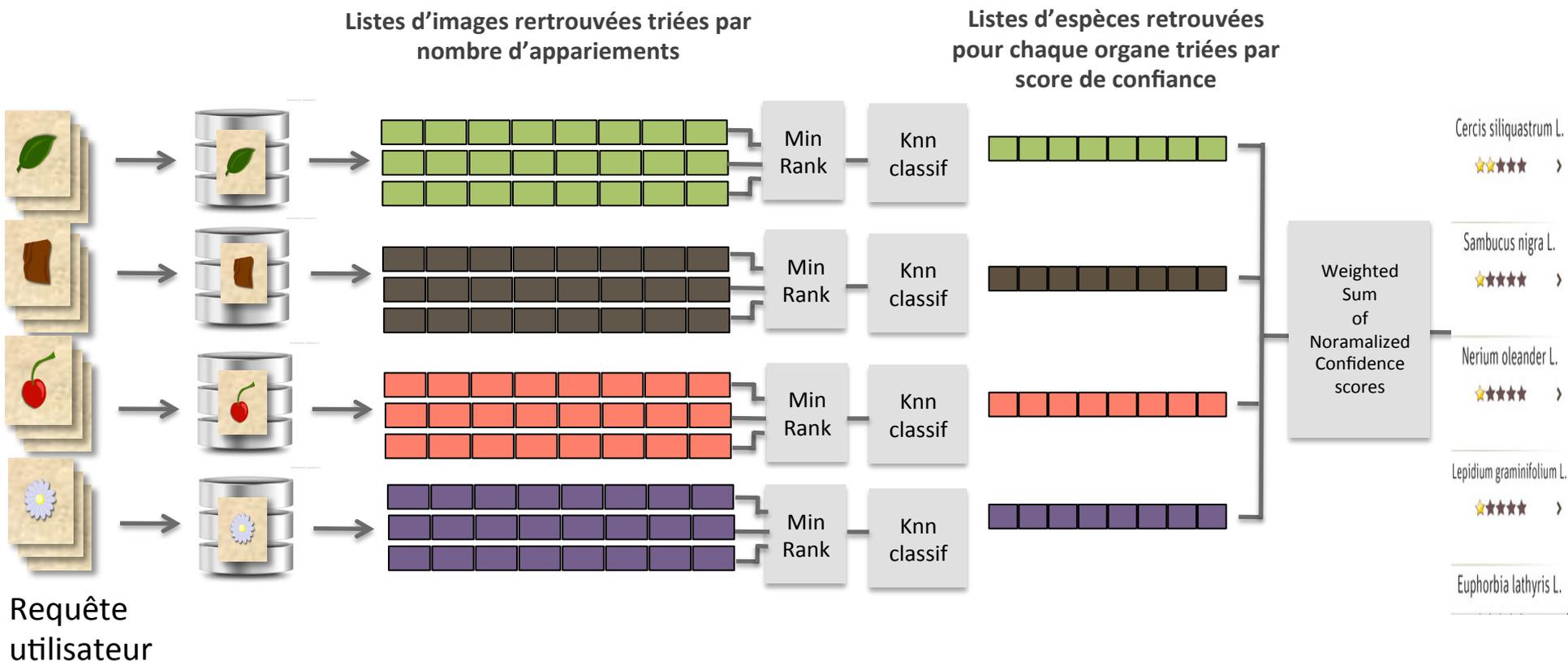
1. Pour chaque descripteur local, on recherche les descripteurs les plus similaires dans la base



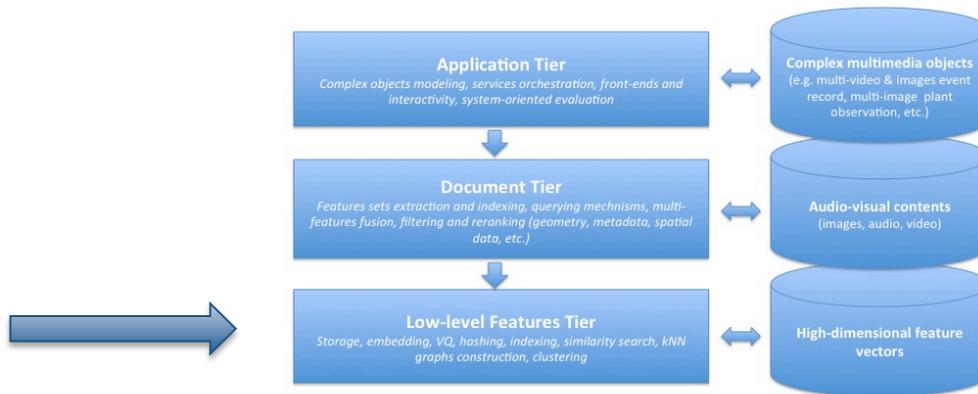
2. On regroupe les descripteurs retrouvés appartenant à une même image de la base
3. On compte le nombre de descripteurs retrouvés par image (= le nombre d'appariements)
4. On trie les images retrouvées par nombre d'appariements décroissants

# PI@ntNet: un exemple de système réel de recherche par le contenu visuel

## Des listes d'images à la liste d'espèce



# Méthode de recherche exhaustive et malédiction de la dimension



# Le problème

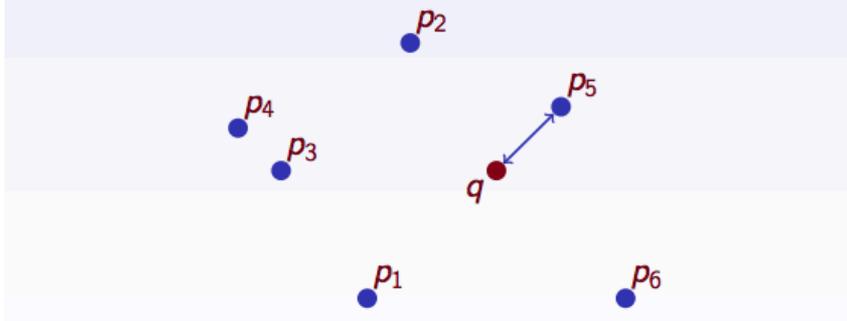
## Recherche du (ou des k) plus proches voisins

**Espace de recherche:** Espace  $\mathbb{U}$ , fonction similarité  $\sigma$

**Entrée:** database  $S = \{p_1, \dots, p_n\} \subseteq \mathbb{U}$

**Requête:**  $q \in \mathbb{U}$

**Tâche:** trouver  $\operatorname{argmax}_{p_i} \sigma(p_i, q)$



## Exemple

Desc. SIFT, dim=128, L2  
100K images x 1000 SIFT  
=  
100M de vecteurs SIFT

Problème fondamental abordé depuis l'époque de Voronoi (1908)

Il n'existe pas de solution exacte de complexité sous-linéaire en particulier pour les grandes dimensions

# La recherche exhaustive

1. Principe: parcours exhaustif de tous les vecteurs de la base et calcul de toutes les distances à la requête

```
for (int i=1; i<n ; i++) ComputeDistance(q,X[i]);
```

2. Recherche des top-k dans la liste des distances

- méthode naïve: on insère les éléments  $\langle i, \text{dist}[i] \rangle$  dans un container (une liste de Map.Entry par exemple) et on utilise un sort

```
for (int i=1; i<n ; i++) {  
    list.add(<i, d(q,X[i])>); //insertion du n-ième élément  
}
```

```
sort(list, comparator); //tri du container
```

```
for (int j=1; j<k ; j++) knn.add(j,list.get(j)); //extraction des k premiers éléments
```

**Problème = sort est un algorithme de complexité  $O(n \log n)$  qui peut être trop coûteux pour une recherche temps réel**

# La recherche exhaustive

1. Principe: parcours exhaustif de tous les vecteurs de la base et calcul de toutes les distances à la requête

```
for (int i=1; i<n ; i++) ComputeDistance(q,X[i]);
```

2. Recherche des top-k dans la liste des distances

- méthode **plus efficace**: on met à jour progressivement les top-k en conservant la position et la distance du k-nn courant

```
dist_max_knn=MAX_DIST;
for (int i=1; i<n ; i++) {
    if (ComputeDistance(q,X[i]) < dist_max_knn) { //si on a trouvé un meilleur voisin
        list.add(<i, d(q,X[i])>); //insertion dans la liste des knn
        [dist_max_knn,index_max_knn]=SearchFurthestElementInList(list); // cherche nouveau max O(k)
        if (i>k) list.remove(index_max_knn); //retire nouveau max
    }
}
```

La complexité est  $O(n.k)$  dans le pire des cas mais en pratique beaucoup plus rapide car de moins en moins de meilleurs voisins

**Problème = ne marche pas lorsque k est trop grand  $k \gg \log(n)$**

# La recherche exhaustive

1. Parcours exhaustif de tous les vecteurs de la base et calcul de toutes les distances à la requête

```
for (int i=1; i<n ; i++) dist[i]=d(q,X[i]);
```

2. Recherche des top-k dans la liste des distances

- méthode pour k grand ( $\gg \log N$ ): on utilise un Max heap ou une **PriorityQueue**

```
public class PriorityQueue<E>  
extends AbstractQueue<E>  
implements Serializable
```

An unbounded priority queue based on a priority heap. The elements of the priority queue are ordered according to their **natural ordering**, or by a **Comparator** provided at queue construction time, depending on which constructor is used. A priority queue does not permit `null` elements. A priority queue relying on natural ordering also does not permit insertion of non-comparable objects (doing so may result in `ClassCastException`).

→ provides  $O(\log(n))$  time for the enqueueing and dequeueing methods: `add` & `poll`;

# La recherche exhaustive

1. Parcours exhaustif de tous les vecteurs de la base et calcul de toutes les distances à la requête

```
for (int i=1; i<n ; i++) dist[i]=d(q,X[i]);
```

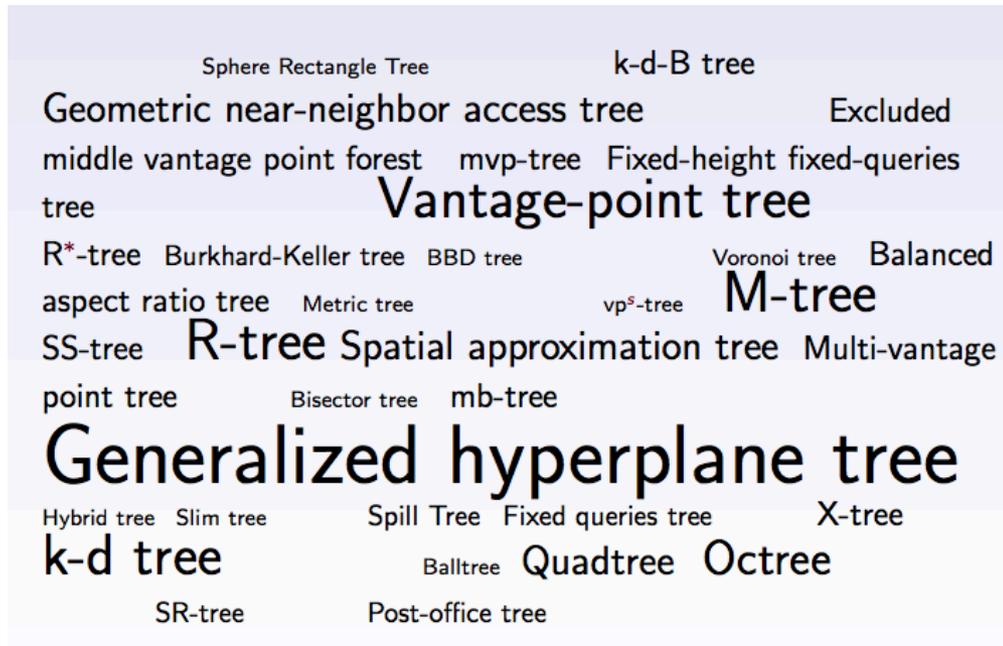
2. Recherche des top-k dans la liste des distances

- méthode pour k grand ( $\gg \log N$ ): on utilise un Max heap ou une **PriorityQueue**

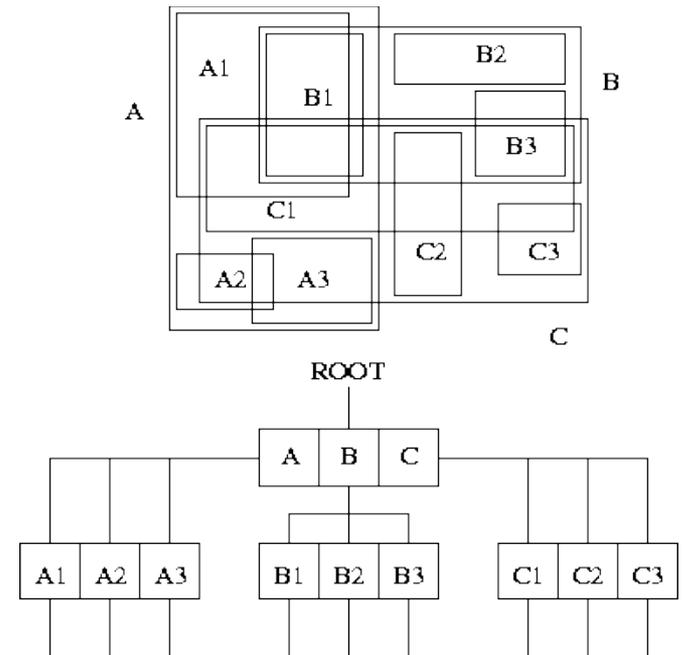
```
PriorityQueue<DistElt> TopKQueue = new PriorityQueue<DistElt>();  
for (int i=1; i<n ; i++) {  
    TopKQueue.add(new DistElt(i, ComputeDistance(q,X[i]) ) );  
    if (i>k) TopKQueue.poll();  
}
```

Complexité  **$O(n \log k)$**

# Les structures d'indexation métriques et multi-dimensionnelles (1980-2000)



R-tree



Weber a montré en 1998 qu'elles étaient moins efficaces que la recherche exhaustive pour  $\text{dim} > 16$  (environ)

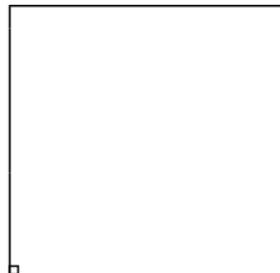
# Curse of dimensionality (malédiction de la dimensionalité)

Lorsque la dimension de l'espace devient très grande il faut parcourir toutes les cellules de la partition pour être sûr de retrouver les k-nn exacts

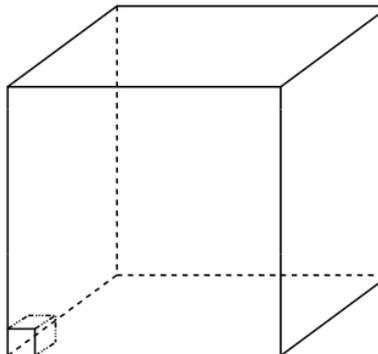
- Assume 5000 points uniformly distributed in the unit hypercube and we want to apply 5-nn. Suppose our query point is at the origin.
  - In 1-dimension, we must go a distance of  $5/5000 = 0.001$  on the average to capture 5 nearest neighbors
  - In 2 dimensions, we must go  $\sqrt{0.001}$  to get a square that contains 0.001 of the volume.
  - In d dimensions, we must go  $(0.001)^{1/d}$



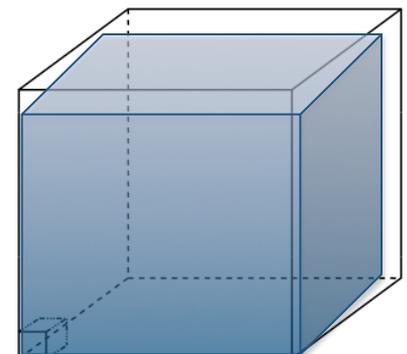
1 D



2 D



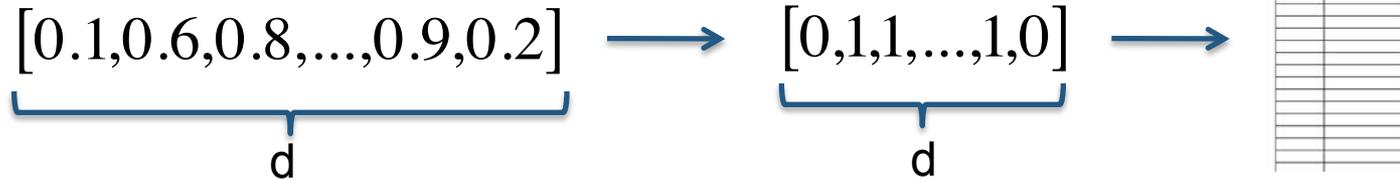
3 D



D >> 3

# Curse of dimensionality (malédiction de la dimensionalité)

La quantification binaire illustre bien l'aspect combinatoire du problème en grande dimension



Recherche à un rayon près  $r \approx \alpha \cdot d$  (en nb de bits dans l'espace d'arrivée)

**Nb de cases à visiter  $n(d) = C_d^r = \frac{d!}{r!(d-r)!}$**

$\frac{n(d+1)}{n(d)} > 1$  et  $\frac{n(d+1)}{n(d)} \approx \frac{1}{1-\alpha}$   $\longrightarrow$   $n(d)$ : suite géométrique et donc une croissance exponentielle

Exemple:  $d=1000$ ,  $\alpha=0.05 \rightarrow n(d)=9.46e+84$

# En pratique on utilise des méthodes approximatives et des heuristiques

## Changements d'espace

- techniques de réduction de la dimension

## Algorithmes de regroupement

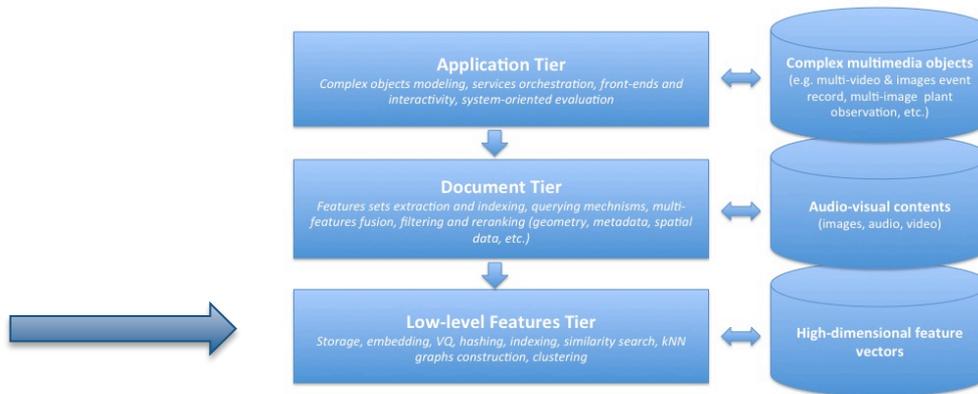
## Compression/quantification (avec perte)

- Hachage
- Hamming embedding

## Algorithmes de recherche approximatifs

- à epsilon près
- statistiques
- heuristiques

# Les changements d'espace



# Techniques de réduction de la dimension

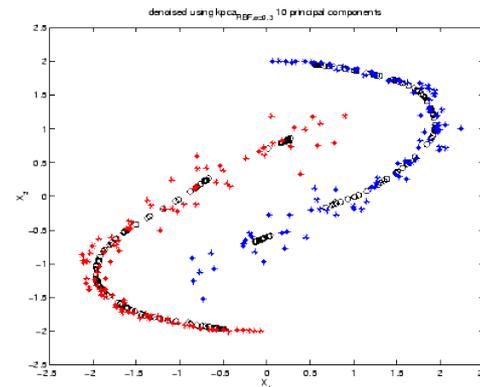
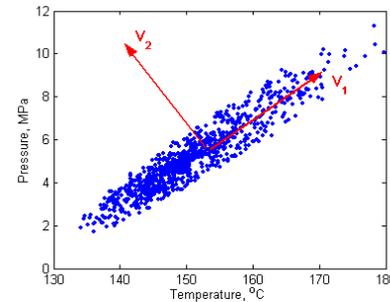
## Analyse en composantes principales Projections aléatoires

Les deux plus  
utilisées

*Autres techniques linéaires:*  
Analyse en composante indépendantes  
Multilinear subspace learning  
Latent semantic analysis

*Techniques non linéaires:*  
Isomap  
Kernel PCA  
Autoencoder (réseaux de neurones)  
Manifold learning

...

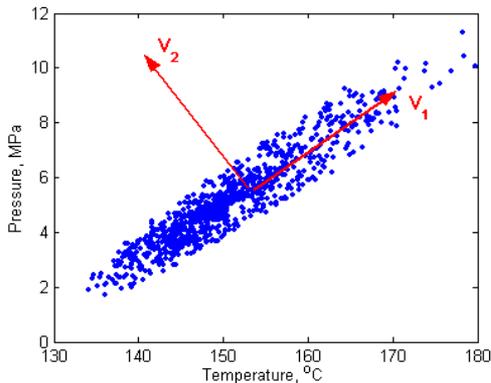


# Analyse en composantes principales

$$y = Ax$$

$y \in R^k$        $x \in R^d$

$A$  = matrice de projection  $d \times k$   
Constituée des  $k$  vecteurs propres de  $Cov(X) = X^T X$   
ayant les valeurs propres les plus grandes



Intérêt:  
**Rotation** puis conservation des axes de variance  
maximale  
+  
Décorrélation (la nouvelle matrice de corrélation est  
diagonale)

# Analyse en composantes principales

$$y = Ax$$

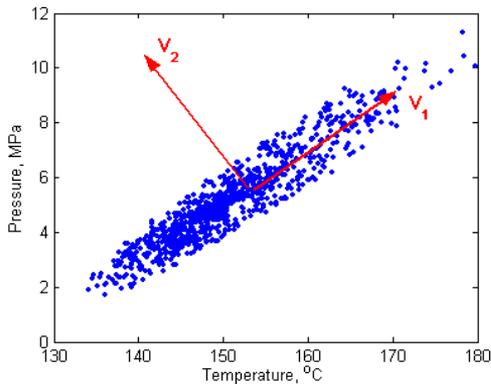
$y \in \mathbb{R}^k$        $x \in \mathbb{R}^d$

$A$  = matrice  $d \times k$

Méthode de calcul de  $A$ :

1) Centrage des données  $X \leftarrow X - \mu_X$

$\mu_X$  = centre de gravité des données



# Analyse en composantes principales

$$y = Ax$$

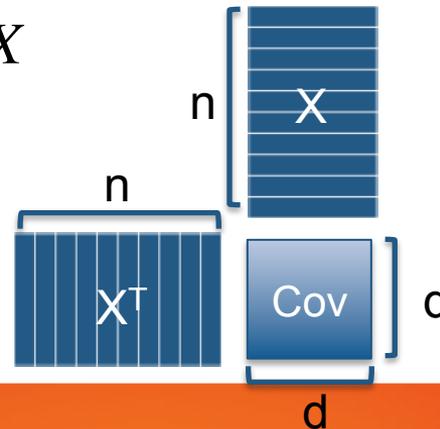
$y \in \mathbb{R}^k$        $x \in \mathbb{R}^d$

$A$  = matrice  $d \times k$

Méthode de calcul de  $A$ :

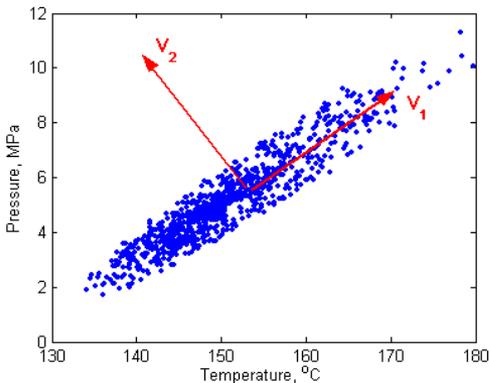
2) Calcul de la matrice de covariance sur un échantillon de  $n$  vecteurs  $\rightarrow O(n^2 \cdot d^2)$

$$Cov = X^T X$$



$$Cov(i, j) = \sigma_{i,j}^2$$

$$Cov(i, i) = \sigma_i^2$$



# Analyse en composantes principales

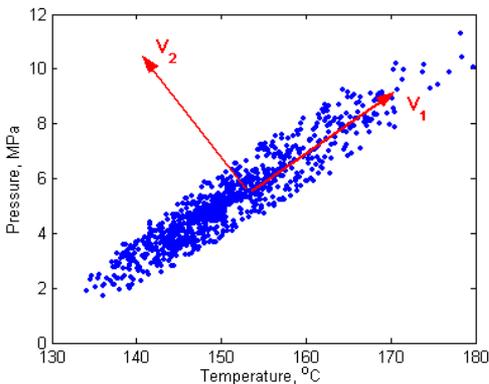
$$y = Ax$$

$y \in R^k$        $x \in R^d$

$A$  = matrice  $d \times k$

Méthode de calcul de  $A$ :

## 3) Diagonalisation de la matrice de covariance



$$Cov = P^T \Delta P$$

$$\underbrace{\begin{matrix} \boxed{\text{Cov}} \\ \boxed{d} \end{matrix}}_d = \boxed{P^T} \times \boxed{\Delta} \times \boxed{P}$$

# Analyse en composantes principales

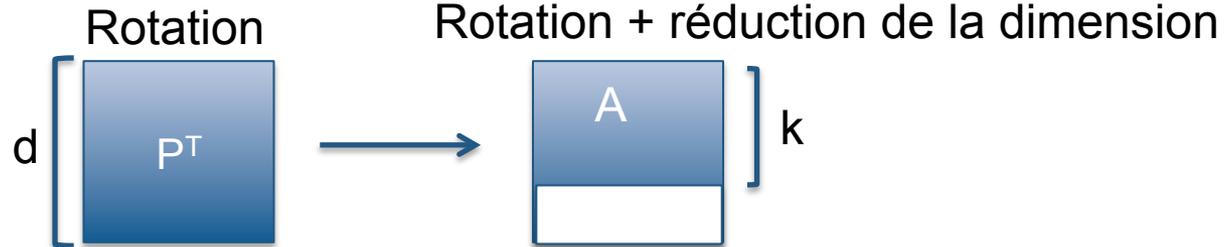
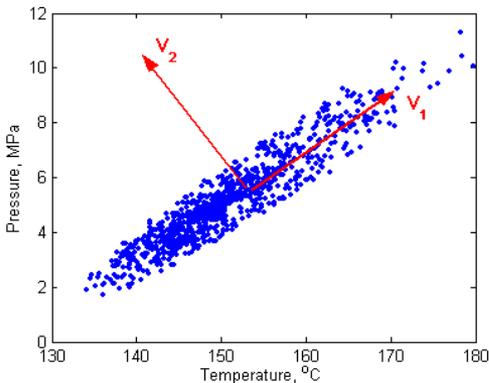
$$y = Ax$$

$y \in R^k$        $x \in R^d$

$A$  = matrice  $d \times k$

Méthode de calcul de  $A$ :

- 4) On ne conserve que les  $k$  vecteurs propres ayant les valeurs propres les plus élevées



# Projections aléatoires

$$y = Ax$$

The diagram shows the equation  $y = Ax$  at the top. Three blue arrows originate from this equation: one points to the left towards  $y \in \mathbb{R}^k$ , one points downwards towards the matrix  $A$ , and one points to the right towards  $x \in \mathbb{R}^d$ .

$$A = \text{randn}(d, k)$$

= i.i.d sampled from Gaussian  $N(0, 1)$

Un corollaire du lemme de Johnson-Lindenstrauss affirme qu'une telle transformation tend à préserver le produit scalaire

**Corollary 2.1.** *Let  $u, v \in \mathbb{R}^d$  and that  $\|u\| \leq 1$  and  $\|v\| \leq 1$ . Let  $f = \frac{1}{\sqrt{k}}Ax$  where  $A$  is a  $k \times d$  matrix, where each entry is sampled i.i.d from a Gaussian  $N(0, 1)$  (or from  $U(-1, 1)$ ). Then,*

$$\Pr(|u \cdot v - f(u) \cdot f(v)| \geq \epsilon) \leq 4e^{-(\epsilon^2 - \epsilon^3)k/4}$$

# Projections aléatoires

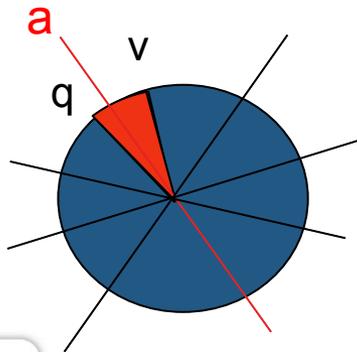
$$y = Ax$$

$y \in \mathbb{R}^k$        $x \in \mathbb{R}^d$

$$A = \text{randn}(d, k)$$

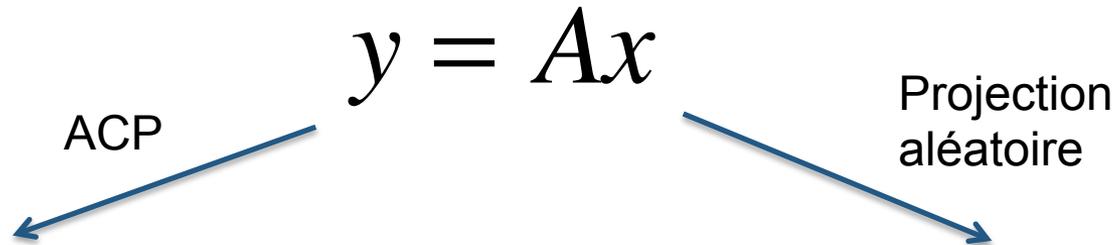
= i.i.d sampled from Gaussian  $\mathcal{N}(0, 1)$

Intuitivement, c'est le jeu de la roue. La probabilité que l'axe tiré aléatoirement sépare  $q$  et  $v$  est fonction de leur angle



$$\Pr[h_w(q) \neq h_w(v)]_w = \text{angle}(q, v) / \pi = \frac{1}{\pi} \cos^{-1}(q^T v)$$

# Comparaison



**Décorrélation = réduction de la dimension plus effective**

**Premières composantes très bonnes**

**Calcul plus coûteux de A et dépendant des données**

**C'est un apprentissage avec des problèmes possibles de généralisation à d'autres données**

**Universalité = aucune dépendance aux données**

**Scalabilité = Génération de A rapide même en très grande dimension**

**Pas de composante a priori plus utile que d'autres**

**Composantes non décorréliées = réduction moins effective**

# Combiner projections aléatoires et orthogonalisation

$$y \in R^k \leftarrow y = Ax \rightarrow x \in R^d$$

$A$  = matrice  $d \times k$  orthogonale générée aléatoirement

Procédure de construction de  $A$ :

1. Matrice  $k \times d$  aléatoire Gaussienne:  $G$
2. Factorisation QR de  $G$ :

$$G = Q R$$

Matrice orthogonale  
 $Q^T Q = I$

3. Conserver les  $k$  premières lignes de  $Q$

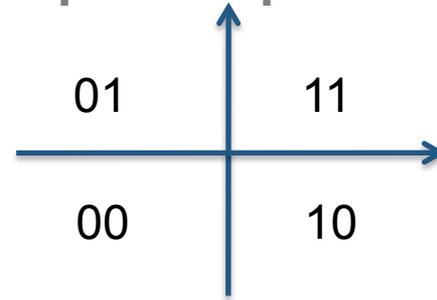
$$A = \begin{matrix} Q \\ \hline \end{matrix}$$

# Les méthodes de partitionnement et de quantification

# Quantification scalaire binaire

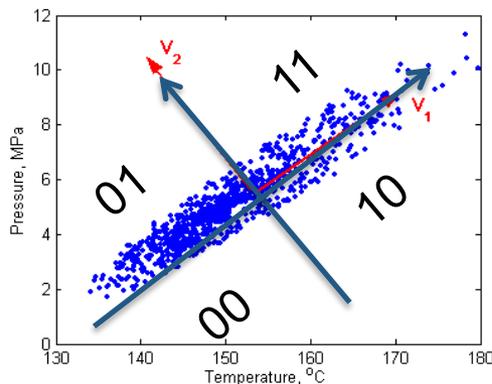
La plus simple = binarisation de chaque composante du vecteur

$$z_j = \text{sign}(x_j)$$



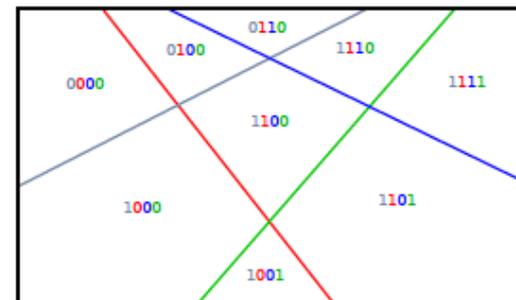
Utilisée en générale après centrage et normalisation des données ou changement d'espace

Ex1, après une ACP:



Ex2, après projections aléatoires

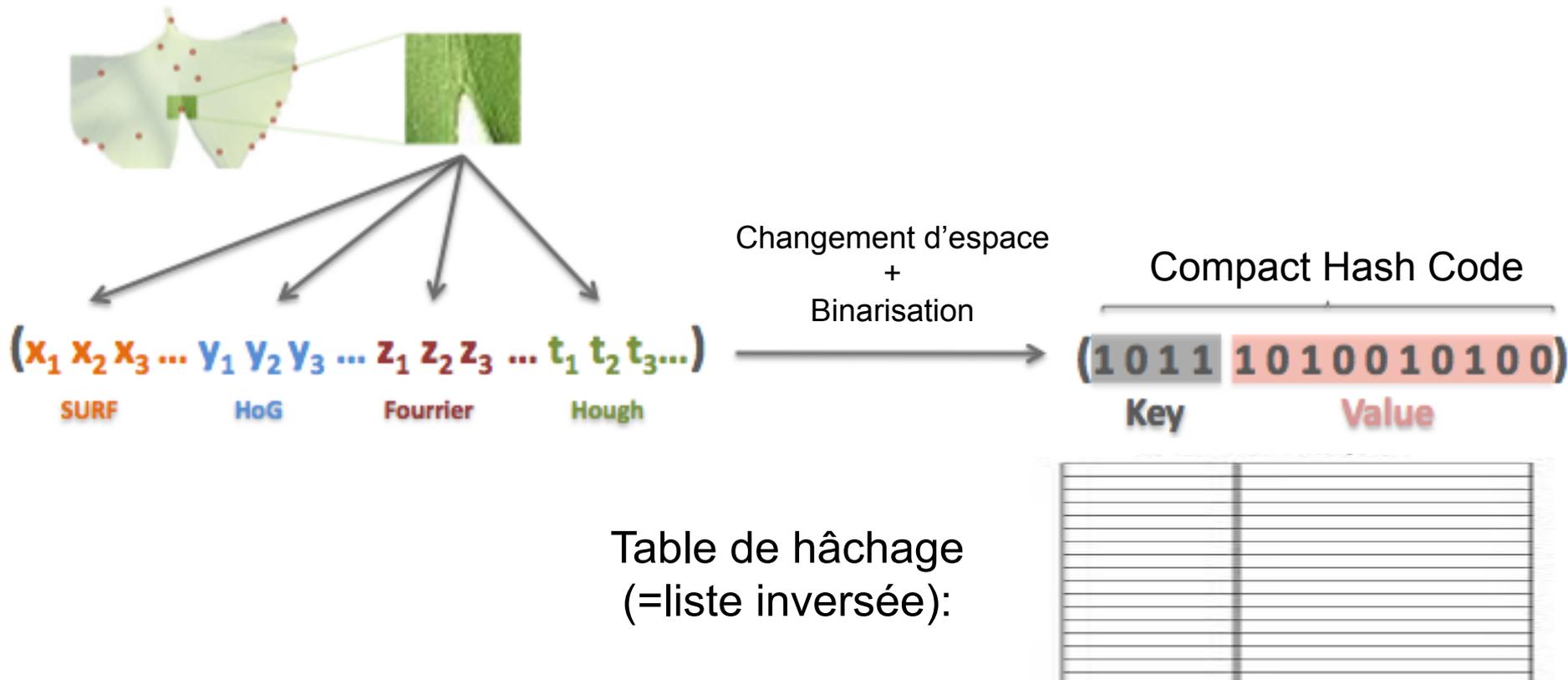
$$z = \text{sign}(Ax + b)$$



= LSH  
(Locality  
Sensitive  
Hashing)

# Quantification scalaire binaire

Avantage: produit des hash codes binaires très compacts et très efficaces pour le stockage, l'adressage ou le calcul de distances



# Quantification scalaire binaire

**Avantage:** produit des hash codes binaires très compactes et très efficaces pour le stockage, l'adressage ou le calcul de distances

**Hamming embedding:** on remplace les distances dans l'espace vectoriel d'origine par une distance de Hamming dans l'espace binarisé d'arrivée

$$d(q,v) \rightarrow d_H(z(q),z(v))$$
$$z(q) = 1 \text{ hash code} = [0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 1 \ 0 \ \dots 0 \ 0 \ 1 \ 0]$$
$$z(v) = 1 \text{ hash code} = [1 \ 0 \ 1 \ 1 \ 1 \ 1 \ 0 \ 1 \ \dots 0 \ 1 \ 0 \ 0]$$



On verra que dans certains cas,  $d_H(z(q),z(v))$  converge vers  $d(q,v)$  lorsqu'on augmente le nombre de bits

La distance de Hamming peut-être de 10 à 100 fois plus rapide qu'une distance naïve dans l'espace d'origine

# Quantification scalaire binaire

**Avantage:** produit des hash codes binaires très compactes et très efficaces pour le stockage, l'adressage ou le calcul de distances

## 1. Nombreuses méthodes très efficaces de calcul de la distance de Hamming

- Utilisation de Look-Up Tables (LUT): typiquement par blocs de 8 ou 16 bits

$$d_H(z(q), z(v)) = \sum d_H(z_i^8(q), z_i^8(v))$$

Chaque bloc n'a que  $2^8$  valeurs possibles pour une requête  $q$  donnée. Ces valeurs peuvent être pré-calculées et stockées dans une table dont la clé est le hash code  $z_i^8(v)$

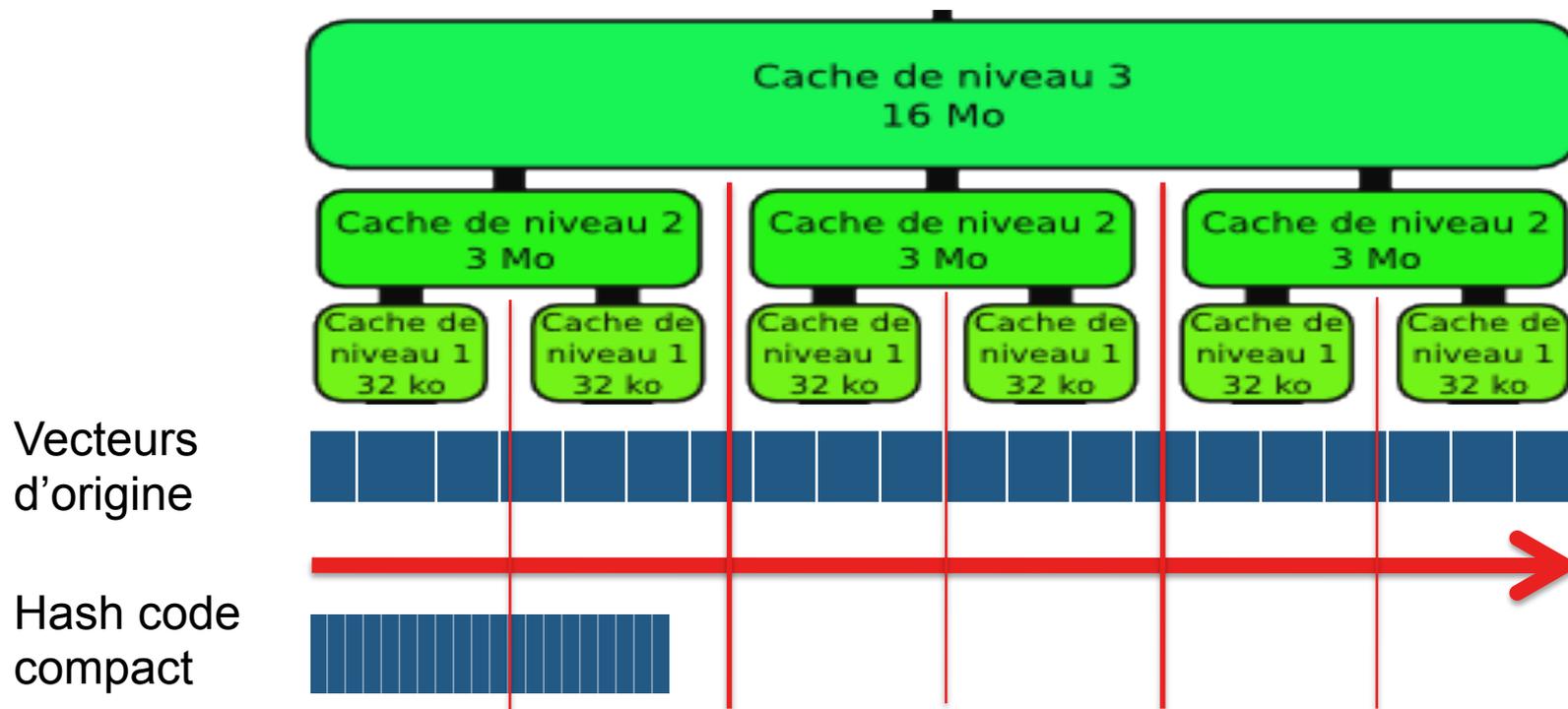
- Instructions assembleurs, e.g. SSE pop-count compte le nombre de bits à 1 de manière extrêmement efficace

$$d_H(z(q), z(v)) = \text{POPCNT}(\text{NOR}(z(q), z(v)))$$

# Quantification scalaire binaire

Avantage: produit des hash codes binaires très compacts et très efficaces pour le stockage, l'adressage ou le calcul de distances

2. Données plus compactes = moins de sorties de cache



# La recherche exhaustive dans le cas binarisé

Il n'y a que **d valeurs possibles pour la distance** (d bits max différent entre deux hash codes). La distance peut donc servir de clé dans une multiMap:

```
MultiMap match_map = new MultiHashMap();
for (i=1; i<n ; i++) match_map.put(ComputeDistance(q,X[i]), i); //rempli la MultiMap O(n)

dist=0; nb_nn=0;
While (nb_nn<k) {
    nb_nn+=match_map.get(m); // accède aux éléments par distance croissante
    dist++;
}
```

**Complexité linéaire  $O(n+d) \approx O(n)$  meilleure que  $O(n \log n)$ ,  $O(n.k)$ ,  $O(n \log k)$**

# Grilles et lattices

**Avantage:** Les mêmes que quantification scalaire binaire mais avec une erreur de quantification qui peut être plus faible

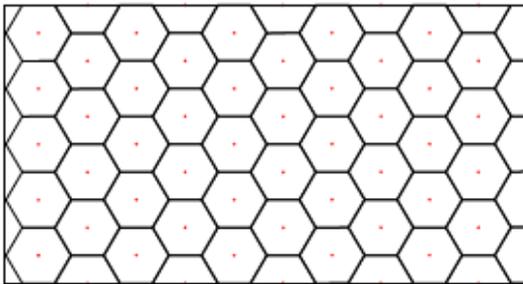
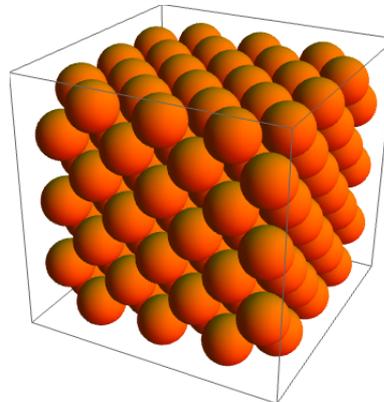
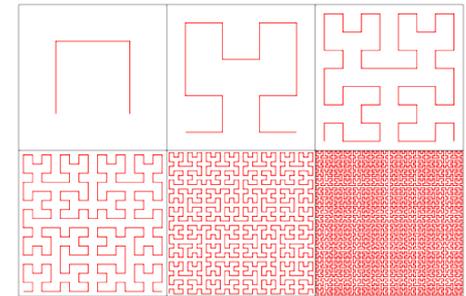


FIGURE 2.2 – Exemple d'un treillis hexagonal



Leech lattices

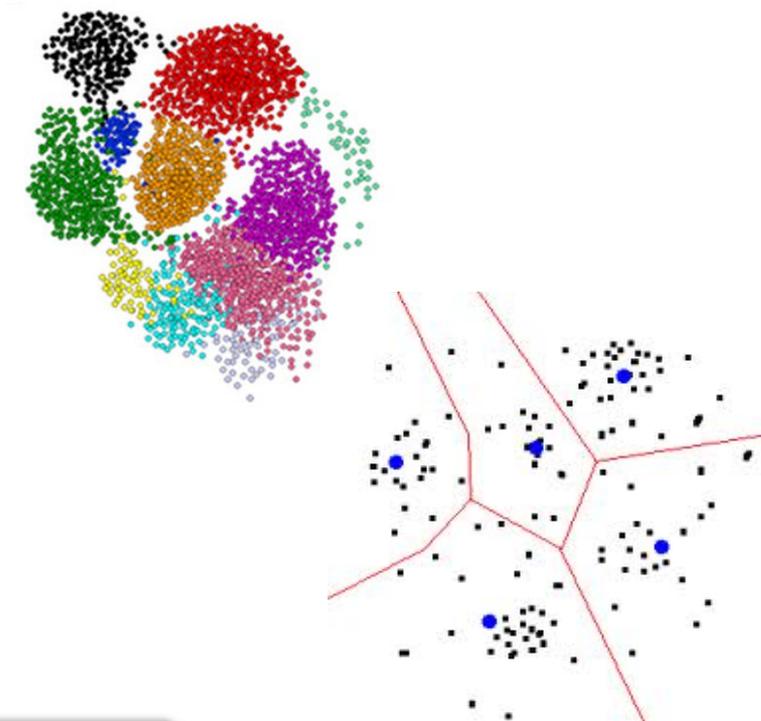


Space-filling curves

**Inconvénients:** Temps d'encodage peut être beaucoup plus long & la gain sur l'erreur de quantification dépend des données et peut être faible au final

# Les méthodes de partitionnement par regroupement de données (clustering)

Contrairement aux méthodes de quantification scalaires ou vectorielles vues précédemment, le partitionnement dépend des données



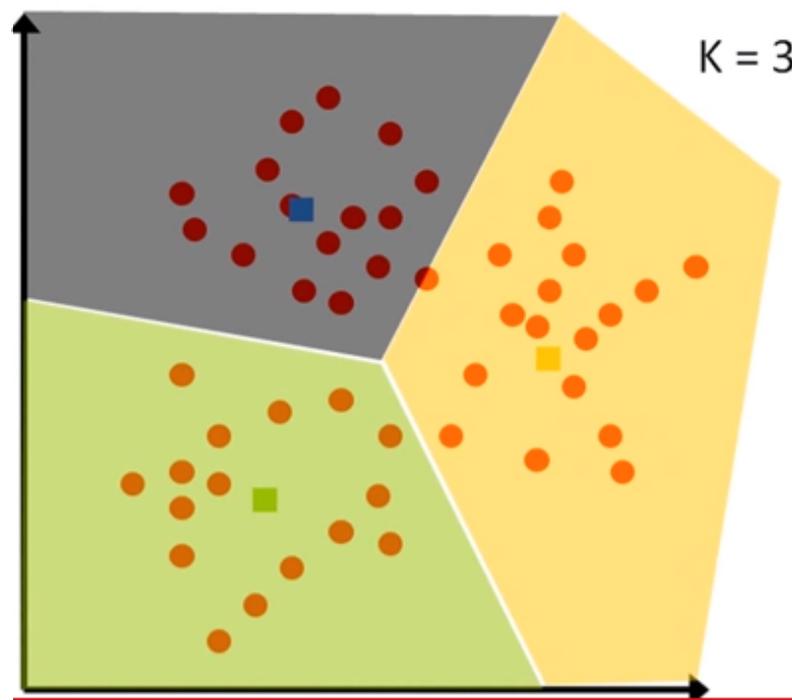
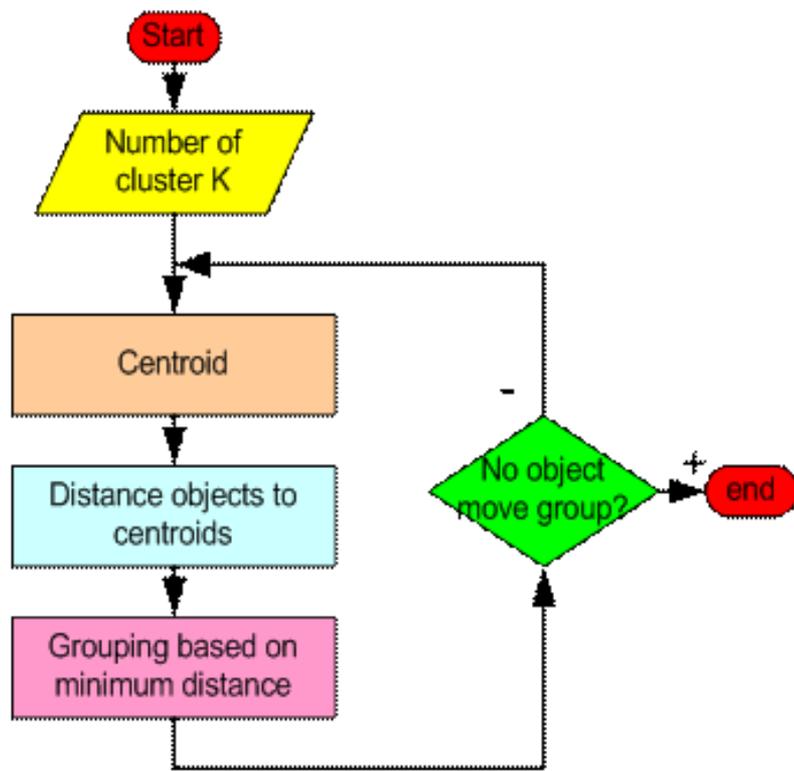
Des centaines de méthodes de clustering dans la littérature:

- Avec ou sans recouvrement
- Nombre de cluster fixes ou non
- Espace euclidien ou non
- Approches hiérarchiques ou non
- Approches probabilistes ou non
- Semi-supervisées (pair-wise constraints)
- Complexité

Un des problèmes les plus transversaux de tout l'informatique

# Les méthodes de partitionnement par regroupement de données (clustering)

Algo le plus utilisée en recherche d'information par le contenu = k-means



# Les méthodes de partitionnement par regroupement de données (clustering)

Algo le plus utilisée en recherche d'information par le contenu = k-means

## Intérêts:

- Adaptabilité aux données
- Minimisation de l'erreur de quantification L2
- Complexité d'apprentissage raisonnable  $O(k.N)$
- Contrôle de la taille de la partition

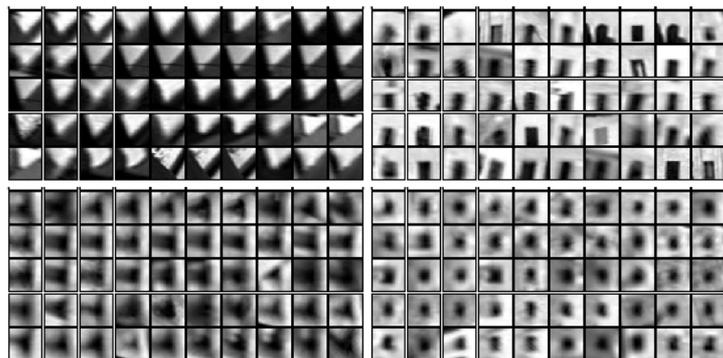
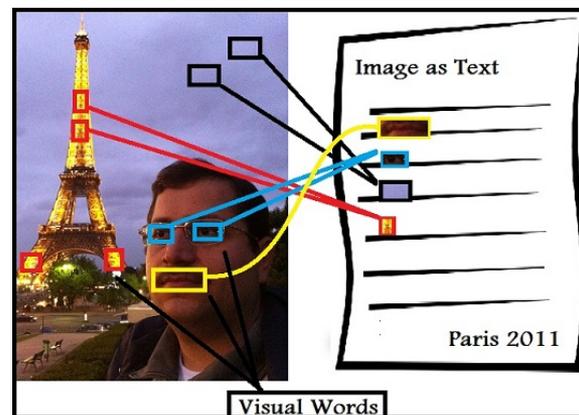
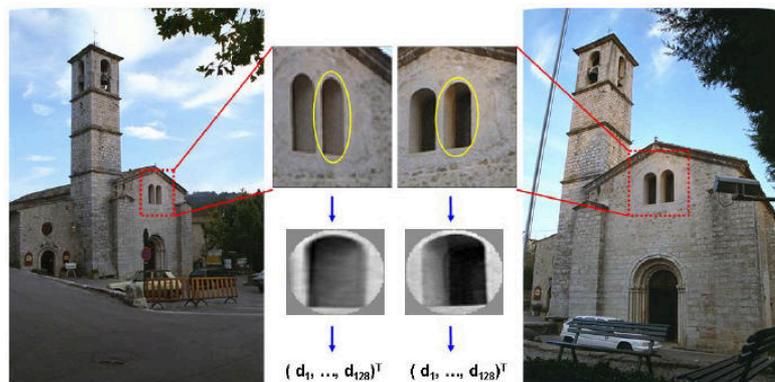
## Inconvénients:

- Temps d'encodage  $O(k)$  devient problématique lorsque  $k$  est grand
- Utile pour partitionner les données mais pas pour compresser finement (pas assez d'information !!)

# Les méthodes de partitionnement par regroupement de données (clustering)

K-means énormément utilisé pour partitionner les espaces de descripteurs visuels locaux

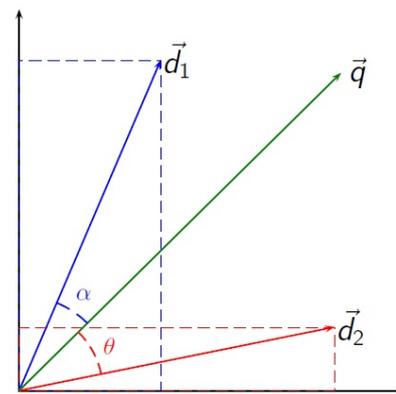
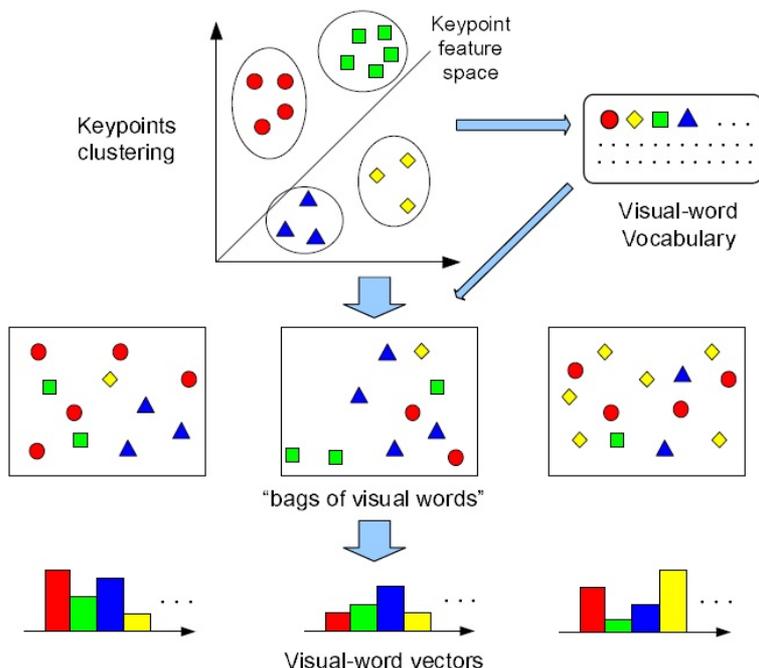
On parle de **mots visuels** pour qualifier les partitions produites



# Les méthodes de partitionnement par regroupement de données (clustering)

K-means énormément utilisé pour partitionner les espaces de descripteurs visuels locaux

Une image = un **sac de mots visuels** par analogie au texte

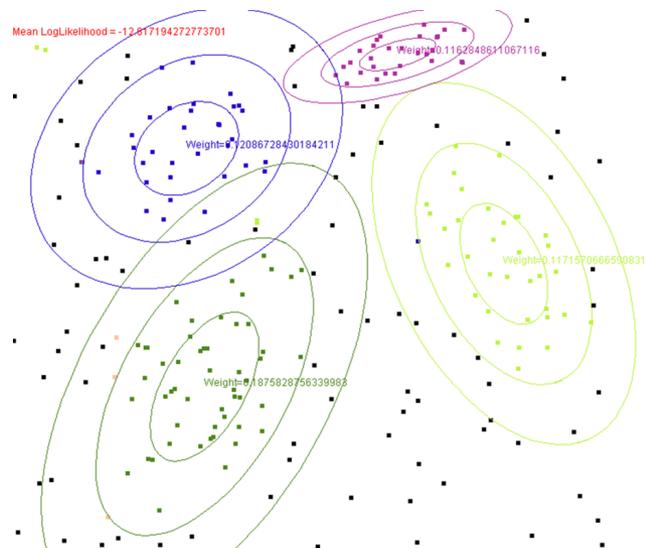
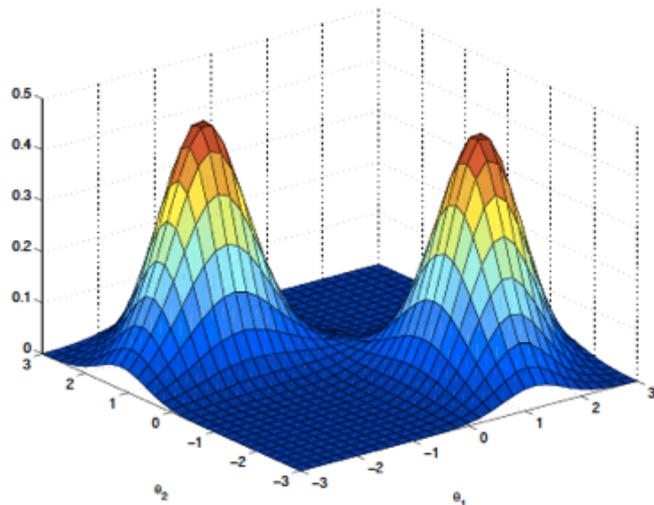


$$d_j = (w_{1,j}, w_{2,j}, \dots, w_{t,j})$$
$$q = (w_{1,q}, w_{2,q}, \dots, w_{t,q})$$

# Les méthodes de partitionnement par regroupement de données (clustering)

## Expectation Maximisation et mixtures de gaussiennes beaucoup utilisées en audio

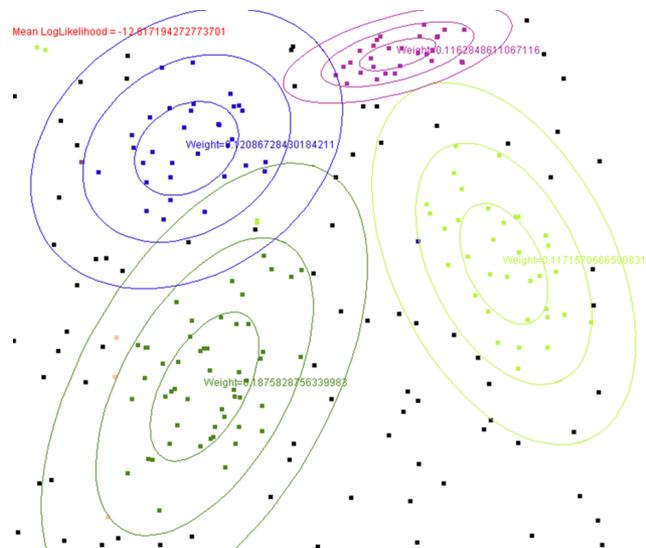
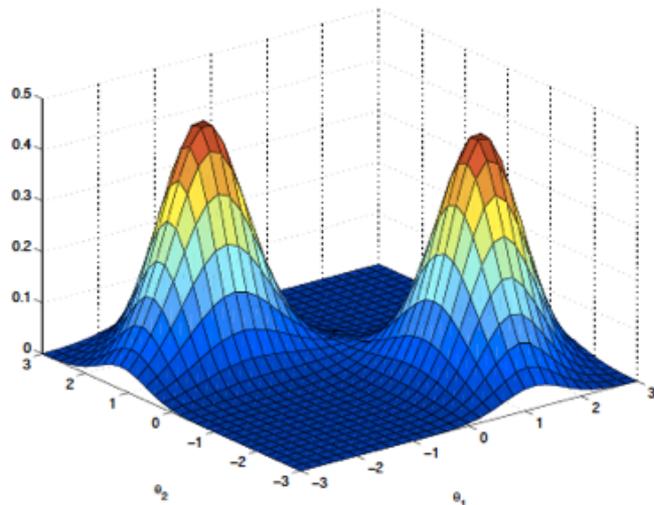
Même principe itératif que k-means mais on estime les paramètres d'une gaussienne multi-dimensionnelles au lieu de uniquement les coordonnées du centroid



# Les méthodes de partitionnement par regroupement de données (clustering)

Expectation Maximisation et mixtures de gaussiennes beaucoup utilisées en audio

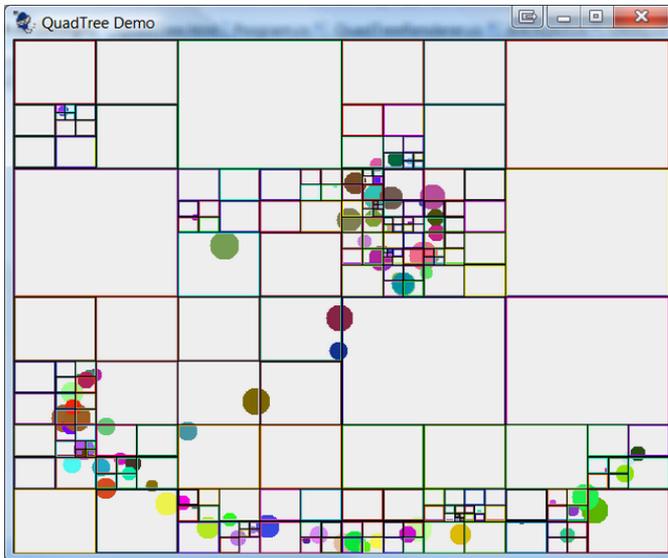
Modèle beaucoup plus fin des données (idéale pour reconnaissance de locuteurs, speech-to-text, etc.), mais plus couteux pour indexation, compression, etc.



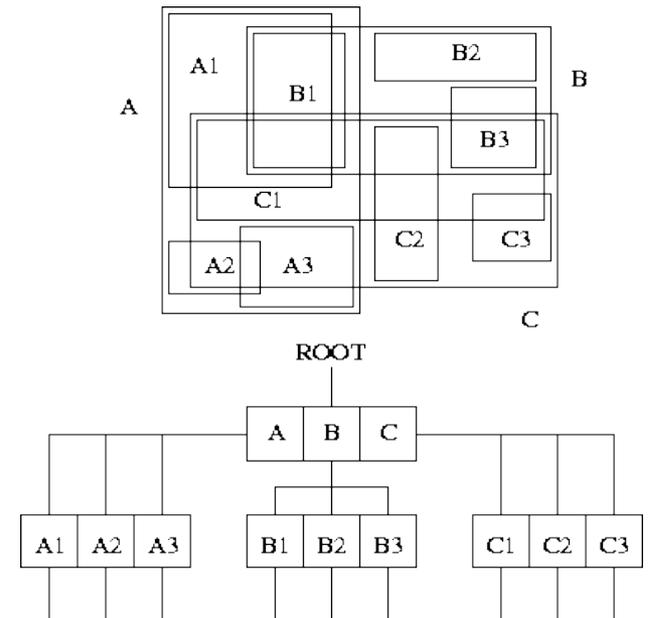
# Les méthodes de partitionnement hiérarchiques

De nombreuses méthodes à base d'arbres existent mais la plupart ne sont pas utilisés pour la recherche par le contenu à cause de la malédiction de la dimension

Quad-tree:  $2^d$  new buckets at each split !!



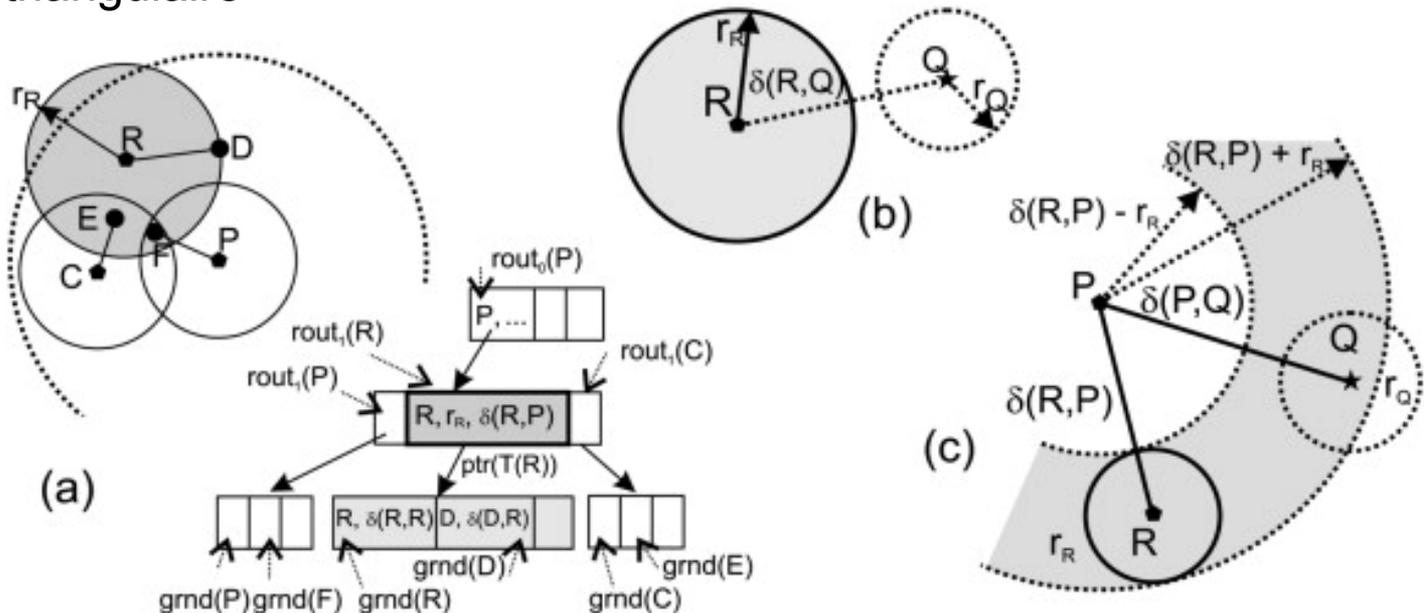
R-tree: Bounding box rectangulaires tendent à couvrir tout l'espace lorsque  $d$  est très grand



# Les méthodes de partitionnement hiérarchiques

Le M-tree (Metric Tree) est encore utilisé dans certaines applications de recherche d'information par le contenu

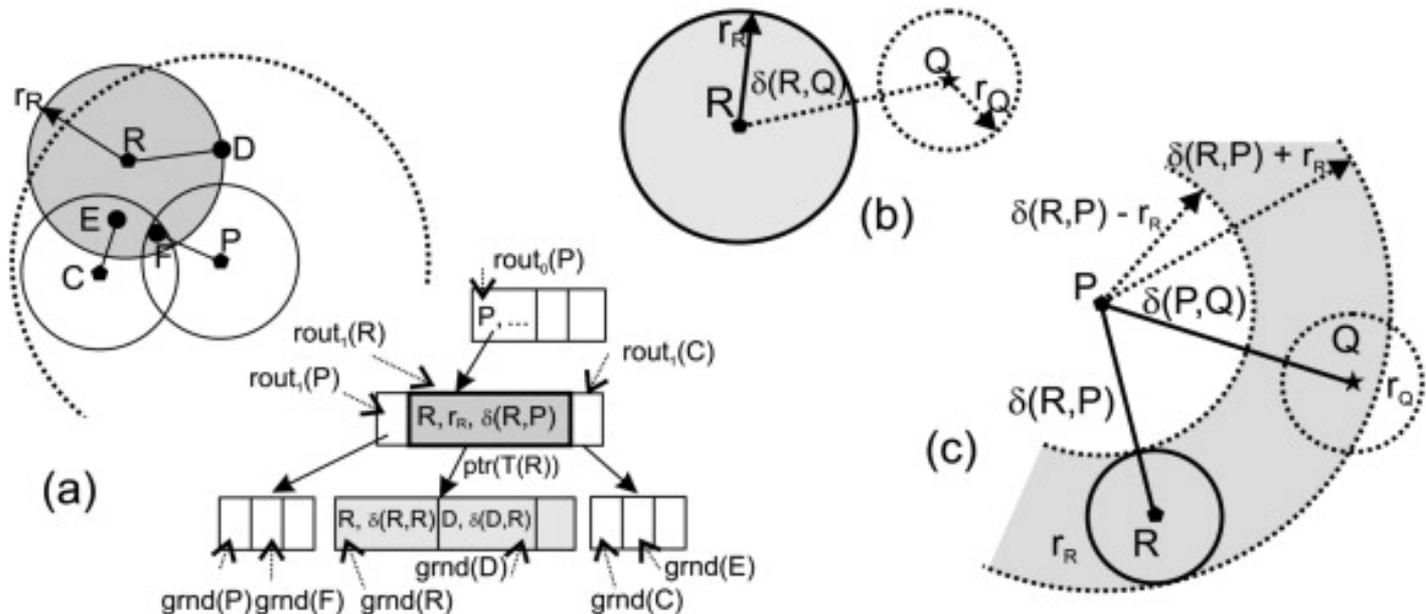
Intérêt: Le M-tree utilise uniquement des points pivots et des distances à ces points pour partitionner les données. On peut indexer n'importe quels objets (même non vectoriel) associés à une métrique respectant l'inégalité triangulaire



# Les méthodes de partitionnement hiérarchiques

Le M-tree (Metric Tree) est encore utilisé dans certaines applications de recherche d'information par le contenu

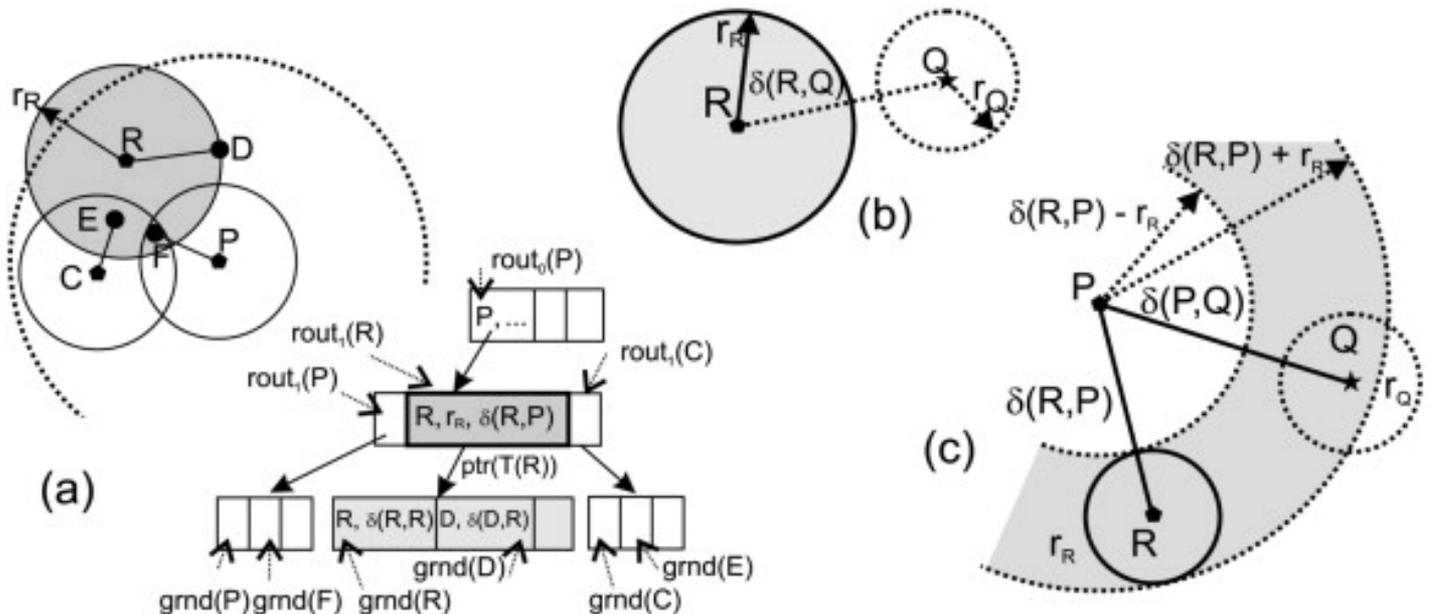
Intérêt: insertions ou suppressions dynamiques de complexité logarithmiques



# Les méthodes de partitionnement hiérarchiques

Le M-tree (Metric Tree) est encore utilisé dans certaines applications de recherche d'information par le contenu

Intérêt: Le M-tree souffre tout de même de la malédiction de la dimension et n'est rentable qu'avec des algorithmes de recherche approximatifs de type  $(1+\epsilon)$ -approximatifs (cf. chapitre algorithmes de recherche)

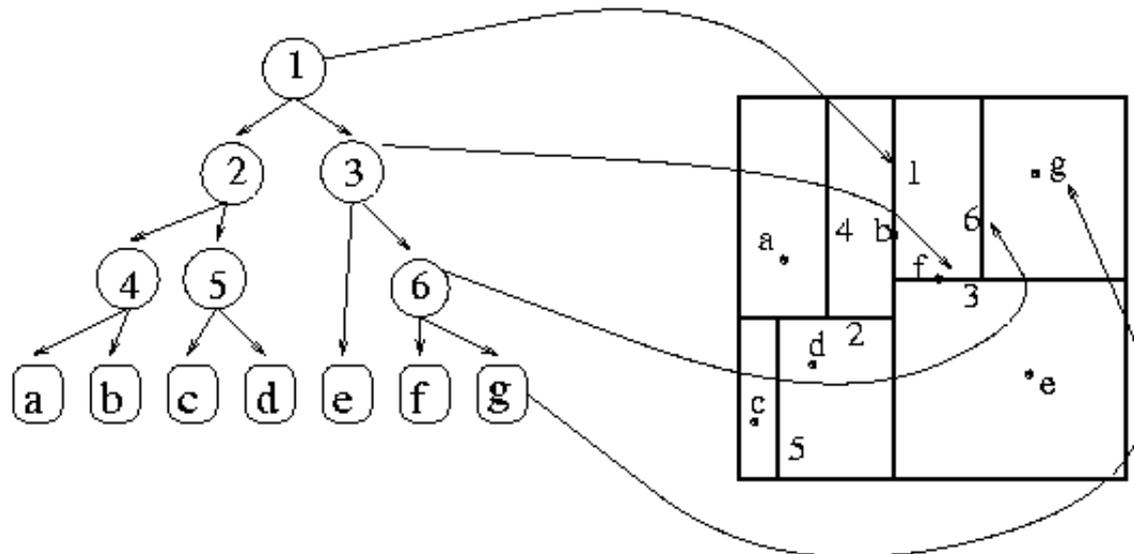


# Les méthodes de partitionnement hiérarchique

Le kd-tree est utilisé pour la recherche d'image par le contenu (e.g. dans la librairie FLANN)

## kd-tree classique:

A chaque niveau  $i$ , partitionne l'ensemble courant en deux parties égales par un hyperplan orthogonal à l'axe  $i \bmod d$

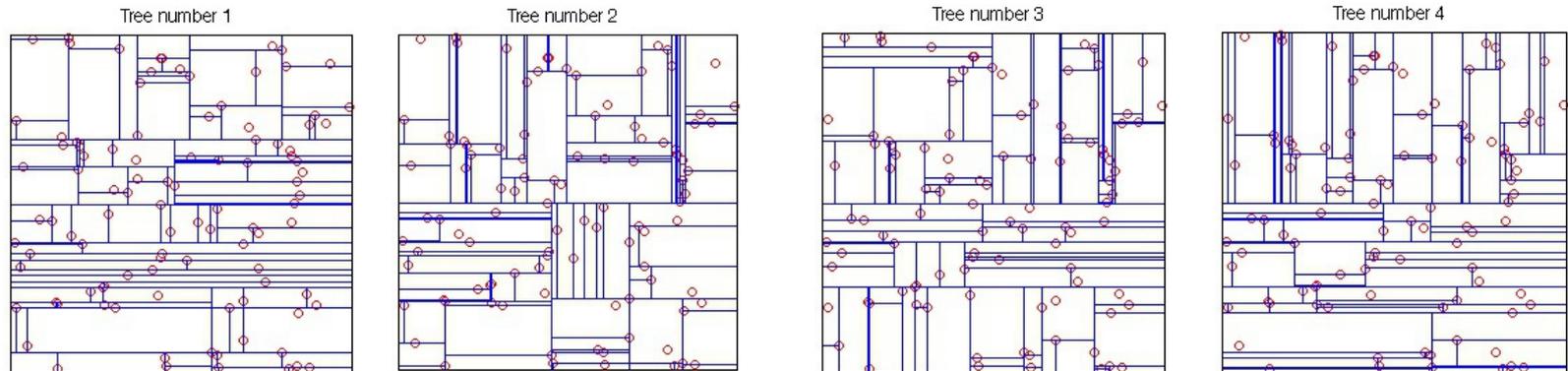


# Les méthodes de partitionnement hiérarchique

Le kd-tree est utilisé pour la recherche d'image par le contenu (e.g. dans la librairie FLANN)

## Randomized kd-trees:

En très grande dimension, on utilise plutôt une forêt d'arbres dans lesquels le choix de la dimension se fait aléatoirement parmi les  $d \leq d$  dimensions ayant la plus grande variance



# Les méthodes de partitionnement hiérarchique

## Hierarchical K-means

On applique un K-Means avec un très faible nombre de cluster ( $K = 10$  par exemple), puis on recommence un nouveau K-Means sur chaque sous-ensemble de descripteurs créés par les  $K$  clusters.

L'algorithme itère ensuite jusqu'à obtention d'un nombre de clusters total suffisant. Ce nombre étant égal à  $K^n$ , avec  $n$  le nombre d'itérations.

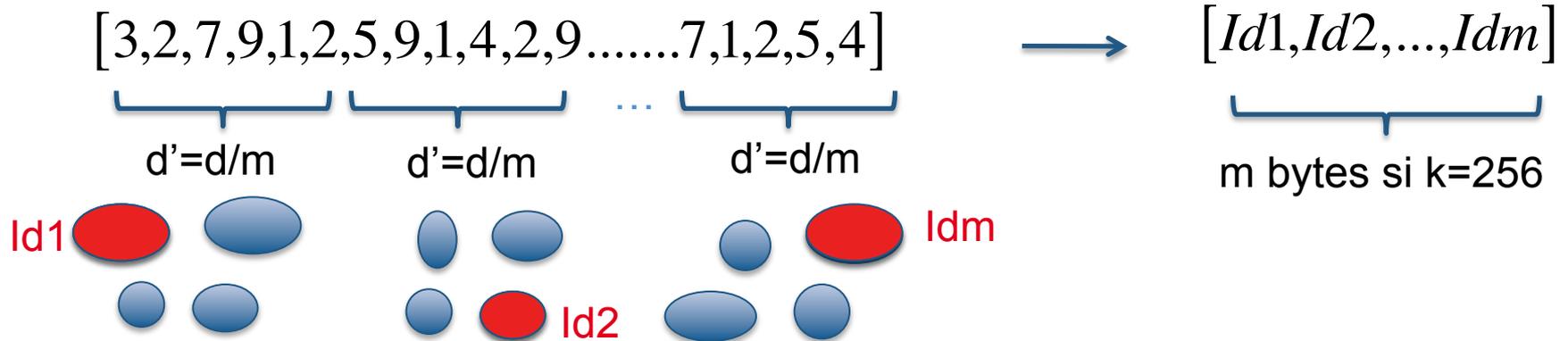
HKM permet de réduire la complexité de l'algorithme à  $O(N \log K)$  au lieu de  $O(NK)$ .

Bien que beaucoup plus efficace en termes de temps de calcul, HKM ne produit pas d'aussi bons résultats de clustering qu'un K-Means standard.

# Les méthodes de partitionnement dans des sous espaces

On découpe l'espace d'entrée en plusieurs sous espaces de dimension moindre puis on partitionne chaque sous espace

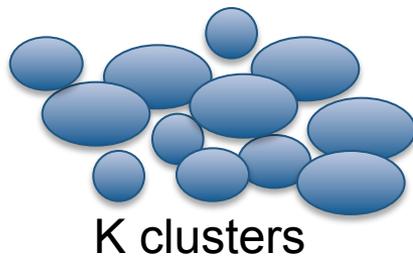
Exemple: Kmeans Product Quantizer



# Les méthodes de partitionnement dans des sous espaces

Exemple: Kmeans Product Quantizer

→ avantage vs. kmeans classique = on peut créer des partitions (vocabulaires) de très grande taille sans problème de temps d'encodage ou d'espace mémoire

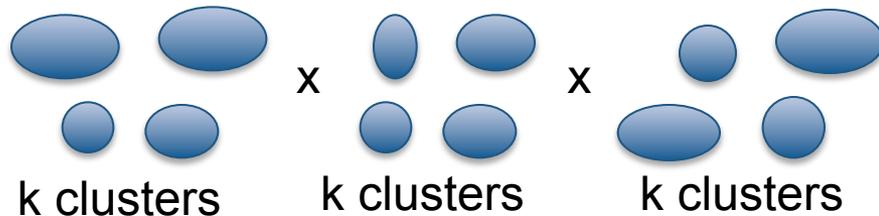


= **K régions**

Construction  $O(Knd)$   
Encodage  $O(Kd)$   
Mémoire  $O(Kd)$



$K \rightarrow m.k$   $d \rightarrow d'=d/m$

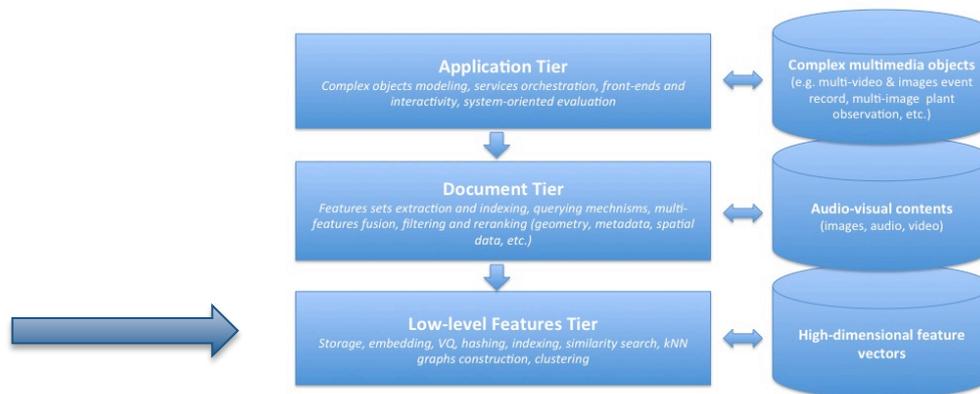


=  **$k^m$  régions**

Construction  $O(knd)$   
Encodage  $O(kd)$   
Mémoire  $O(kd)$   
 **$k \ll K$**

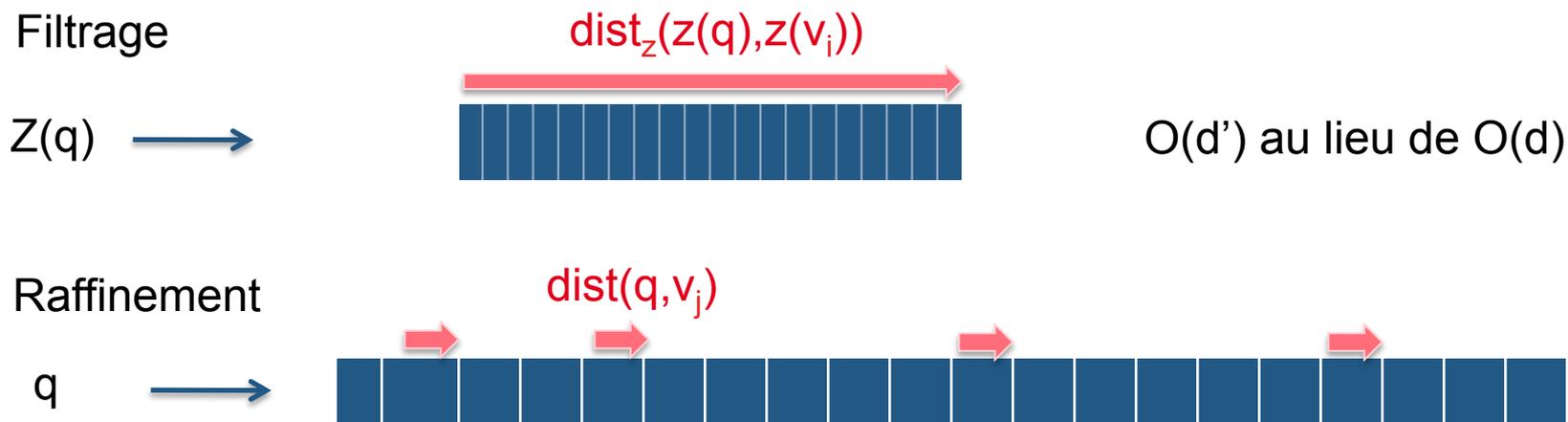
e.g.  $k:256, m:8, d:128 \Rightarrow 18.10^{18}$  régions

# Les structures d'index et algorithmes de recherche associées



# Tableaux de vecteurs quantifiés et recherche exhaustive à deux étages

On cherche un ensemble de voisins dans l'espace quantifié puis on raffine dans l'espace d'origine

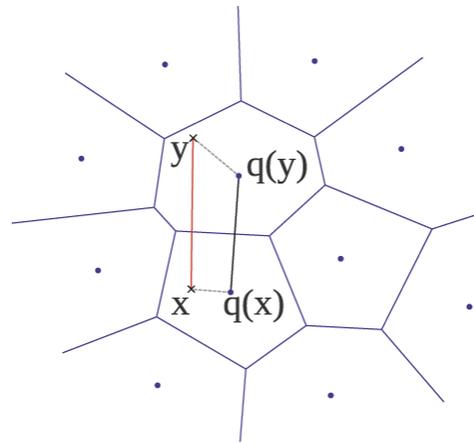
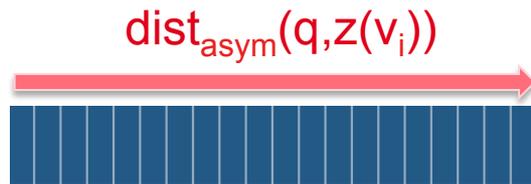


Le filtrage dans l'espace quantifié peut se faire de plusieurs façons:

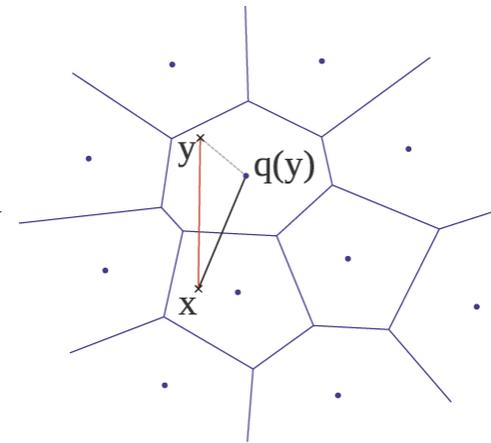
- Les  $k'$  plus proches voisins avec  $k' \gg k$  voulu dans l'espace d'origine
- A un rayon près
  - méthode exact parfois possible (borne inf sur la bucket)
  - méthode probabiliste parfois possible (Ish: projections aléatoires)

# Les distances asymétriques

On peut ne pas quantifier la requête et utiliser une distance asymétrique avec les vecteurs quantifiés de l'index



symmetric case

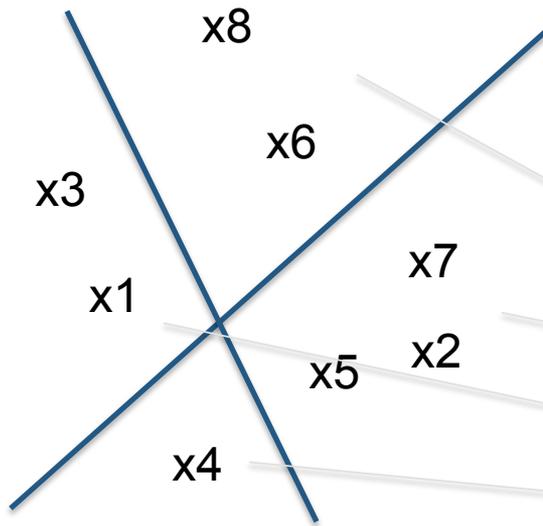


asymmetric case

Avantage: meilleure approximation de la distance dans l'espace d'origine sans augmentation de l'espace mémoire de l'index

Inconvénient: en général un peu plus lent que distance de Hamming

# Les tables et listes inversées



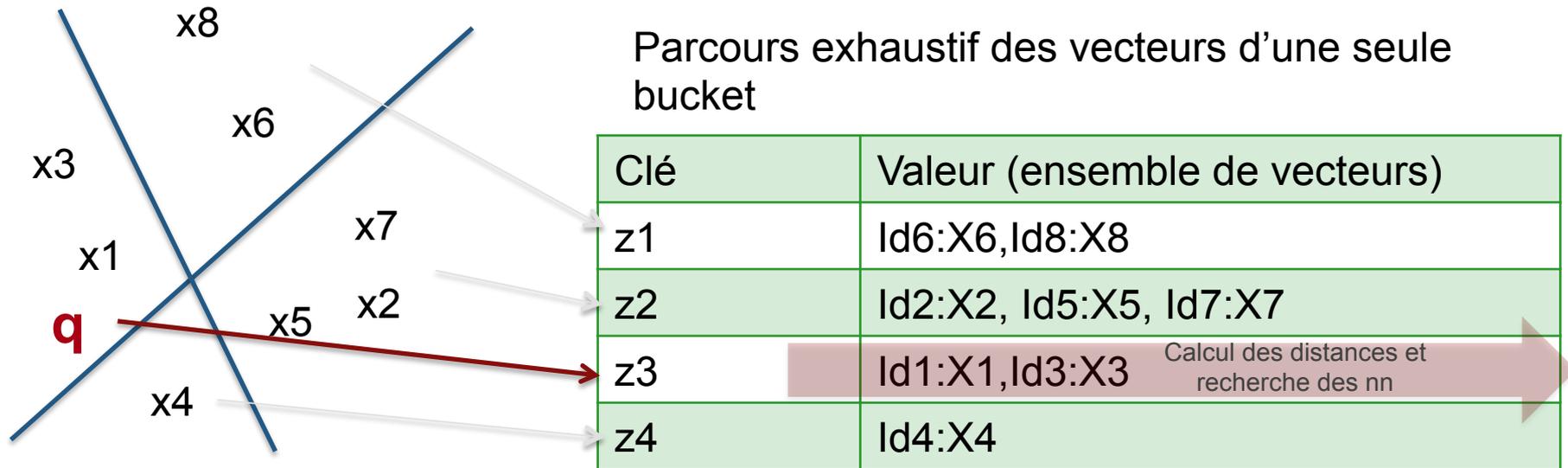
Clé=identifiant d'une région de la partition= 1 valeur dans l'espace quantifiée

Clé	Valeur (ensemble de vecteurs)
z1	Id6:X6, Id8:X8
z2	Id2:X2, Id5:X5, Id7:X7
z3	Id1:X1, Id3:X3
z4	Id4:X4

Zj peut être:

- Un identifiant de cluster (par exemple mot visuel)
- Un hash code binaire (stocké sous forme de string)
- Une position dans une grille
- Un identifiant de feuille dans un arbre
- ...

# Les tables et listes inversées: accès simple

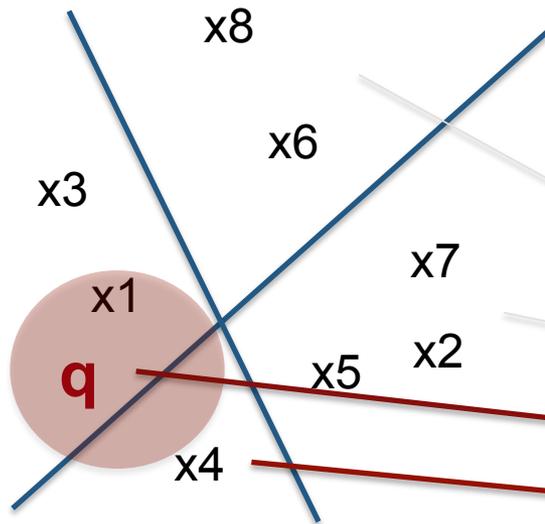


Avec ou sans calcul des distances pour affiner la recherche (knn ou range)

Très efficace, mais qualité faible en grande dimension quelle que soit la méthode de partitionnement:

- malédiction de la dimension
- beaucoup de plus proches voisins n'appartiennent pas à  $z(q)$
- Taille de la partition = compromis entre qualité et efficacité

# Les tables et listes inversées: accès multiples



Parcours exhaustif des vecteurs de plusieurs buckets voisines

Clé	Valeur (ensemble de vecteurs)
z1	Id6:X6, Id8:X8
z2	Id2:X2, Id5:X5, Id7:X7
z3	Id1:X1, Id3:X3
z4	Id4:X4

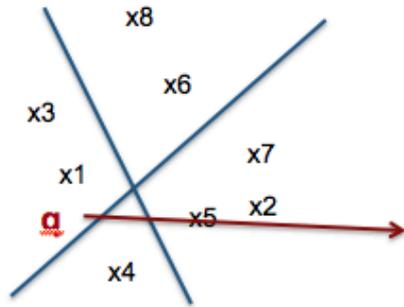
Calcul des distances et recherche des nn

Calcul des distances et recherche des nn

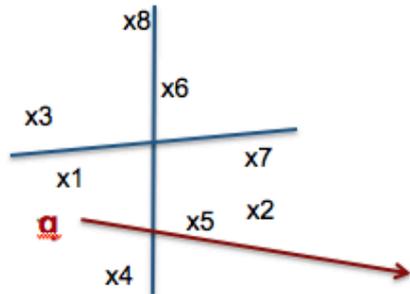
Amélioration du rappel en visitant les buckets voisines. La détermination des buckets voisines dépend du type de partition et peut se faire de plusieurs façons:

- Les k plus proches (typiquement pour kmeans, compromis qualité / temps de calcul ajustable en ligne)
- à un rayon près (pas toujours possible, danger: nombres de bucket peut être énorme)
- probabiliste (par apprentissage ou théorique dans le cas de projections aléatoires → étude détaillée de LSH)

# Les tables et listes inversées: tables multiples



<u>Clé</u>	<u>Valeur (ensemble de vecteurs)</u>
z1	Id6:X6, Id8:X8
z2	Id2:X2, Id5:X5, Id7:X7
z3	Id1:X1, Id3:X3
z4	Id4:X4



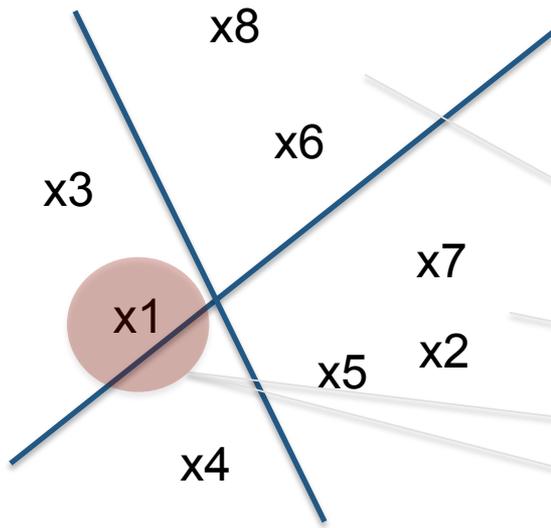
<u>Clé</u>	<u>Valeur (ensemble de vecteurs)</u>
z1	Id3:X3, Id8:X8
z2	Id6:X6
z3	Id2:X2, Id5:X5, Id7:X7
z4	Id1:X1, Id4:X4

La manière de construire plusieurs tables dépend du type de partition:

- Plusieurs kmeans sur des données d'apprentissage différentes
- Plusieurs ensembles de projections aléatoires (cf. LSH)
- Analyse en composantes principales sur des sous-espaces

Peut être plus efficace que les accès multiples (moins d'accès au final) mais nécessite plus de mémoire pour stocker toutes les tables

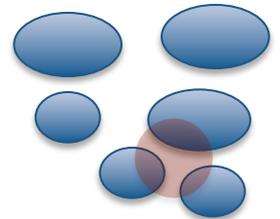
# Les tables et listes inversées: indexation dense



Indexation dans plusieurs buckets voisines, accès unique pendant le search

Clé	Valeur (ensemble de vecteurs)
z1	Id3:X3, Id6:X6, Id8:X8
z2	Id2:X2, Id5:X5, Id6:X6, Id7:X7
z3	Id1:X1, Id3:X3, Id4:X4
z4	Id1:X1, Id4:X4

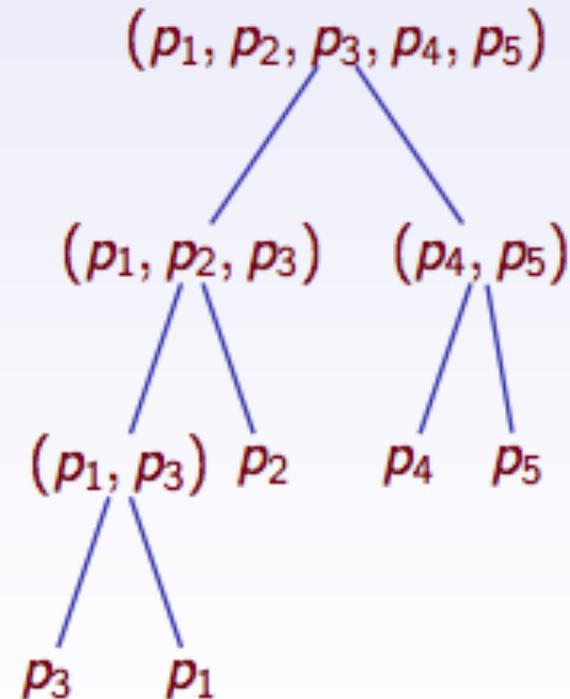
- On troque du temps de calcul contre de la mémoire
- La mémoire requise peut devenir très grande pour certaines partitions
- La détermination des buckets voisines peut se faire de plusieurs manières (k plus proches, rayon, probabiliste)
- Particulièrement utilisée avec partition kmeans (



# Les arbres (pour méthodes de partitionnement hiérarchiques)

Base  $S = \{p_1, \dots, p_n\}$   
représentée par un arbre:

- Chaque noeud est un sous-ensemble de  $S$
- La racine est  $S$  entier
- L'ensemble père est entièrement couvert par les ensembles fils
- Chaque noeud contient une "description" de son sous-arbre fournissant une **borne** inférieure pour  $d(q, \cdot)$  dans le sous-ensemble correspondant

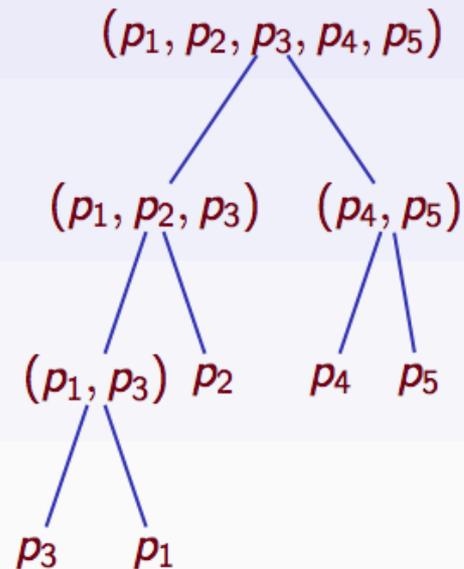


# Les arbres (pour méthodes de partitionnement hiérarchiques)

## Branch and Bound: Recherche à un rayon près

**Objectif:** Trouver tous les  $p_i$  tels que  $d(p_i, q) \leq r$ :

- 1 Parcours en profondeur de l'arbre (algorithme récursif)
- 2 A chaque noeud, calcul de la borne inférieure du sous-ensemble correspondant
- 3 Elimination des branches avec une borne inférieure supérieure à  $r$



# Les arbres (pour méthodes de partitionnement hiérarchiques)

## B&B: Recherche du plus proche voisin

**Objectif:** trouver  $\operatorname{argmin}_{p_i} d(p_i, q)$ :

- 1 Choisir aléatoirement un  $p_i$ , initialiser  $p_{NN} := p_i, r_{NN} := d(p_i, q)$
- 2 Commencer une recherche à  $r_{NN}$  près
- 3 Lorsque l'on rencontre un  $p'$  tel que  $d(p', q) < r_{NN}$ , mise à jour  $p_{NN} := p', r_{NN} := d(p', q)$

# Les arbres (pour méthodes de partitionnement hiérarchiques)

## B&B: Best Bin First

**Objectif:** trouver  $\operatorname{argmin}_{p_i} d(p_i, q)$ :

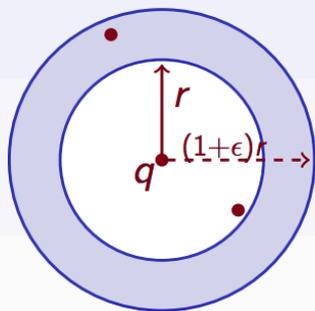
- 1 Choisir aléatoirement un  $p_i$ , initialiser  $p_{NN} := p_i, r_{NN} := d(p_i, q)$
- 2 Mettre le noeud racine dans une **queue d'inspection**
- 3 Répéter: choisir le noeud de la queue d'inspection ayant la borne inférieure la plus petite, calculer les bornes inf. des sous-ensembles fils
- 4 Insérer dans le queue les fils dont la borne inf. est inférieure à  $r_{NN}$  ; éliminer les branches des autres fils
- 5 Lorsque l'on rencontre un  $p'$  tel que  $d(p', q) < r_{NN}$ , mise à jour  $p_{NN} := p', r_{NN} := d(p', q)$

# Les arbres (pour méthodes de partitionnement hiérarchiques)

En dimension  $>15$ , seuls des algorithmes de recherche approximatifs permettent d'être plus rapide qu'une recherche exhaustive (malédiction de la dimension)

- early stopping
- requête à  $(1+\epsilon)r$

**Requêtes à un rayon  $r$  près  $(1 + \epsilon)$ -approximatives:**  
s'il y a au moins un  $p \in S : d(q, p) \leq r$  retourne des  $p' : d(q, p') \leq (1 + \epsilon)r$



C'est une borne max sur l'erreur relative entre le(s) point(s) retrouvé(s) et le(s) knn exact

$$\frac{d(q, p') - d(q, p)}{d(q, p)} \leq \epsilon$$

# Les arbres (pour méthodes de partitionnement hiérarchiques)

## Quand utiliser des arbres ?

- Dimensionnalité modérée (<128)
- Taille de base modérée (<1M de vecteur)
- Lorsque les données arrivent de manière incrémentale
- Lorsqu'il y a des besoins d'insertion/suppression dynamique

Les arbres ne sont en revanche pas adaptés aux très grandes tailles de base notamment lorsque l'accélération du temps de recherche reste le facteur primordial

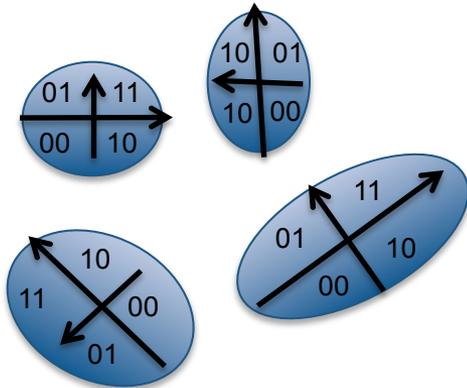
- Surcoût mémoire pour stocker la structure
- Parcours d'arbre plus coûteux que des accès dans une table (sauts mémoires, sorties de cache, des « if », etc.)

# Les approches combinées à deux étages (coarse-to-fine)

On peut utiliser une quantification grossière sous forme de table ou d'arbre et des approches de quantifications vectorielles plus fines à l'intérieur des buckets

Exemple: k-means + ACP locale + Hamming Embedding

4 clusters  
 $z_i$



Id cluster	Ensemble de vecteurs quantifiés appartenant au cluster
z1	Id6:01001, Id8:10101
z2	Id2:00110, Id5:11110, Id7:00000
z3	Id1:01001, Id3:10101
z4	Id4:10100

- La quantification binaire s'adapte à chaque cluster et est au final plus efficace qu'une ACP globale
- Au moment de la requête, on détermine d'abord le cluster puis on quantifie avec la bonne matrice de projection

# Les approches combinées

On peut imaginer beaucoup de combinaisons... nombreux travaux de recherche porte sur le sujet

- Randomized KD-tree pour filtrer + quantification fine avec projection aléatoire et ACP
- Kmeans normal pour construction table + indexation fine avec product quantizer
- ACP binarisée pour construction table + projections aléatoires pour quantification fine des vecteurs
- Possibilité de faire plus de 2 étages de quantification (coarse-to-fine)
- Construction de graphes de similarité approximatifs sur vecteurs quantifiés puis raffinement lors de la marche, etc.