Recherche et tri 2

Arbre 2-3-4

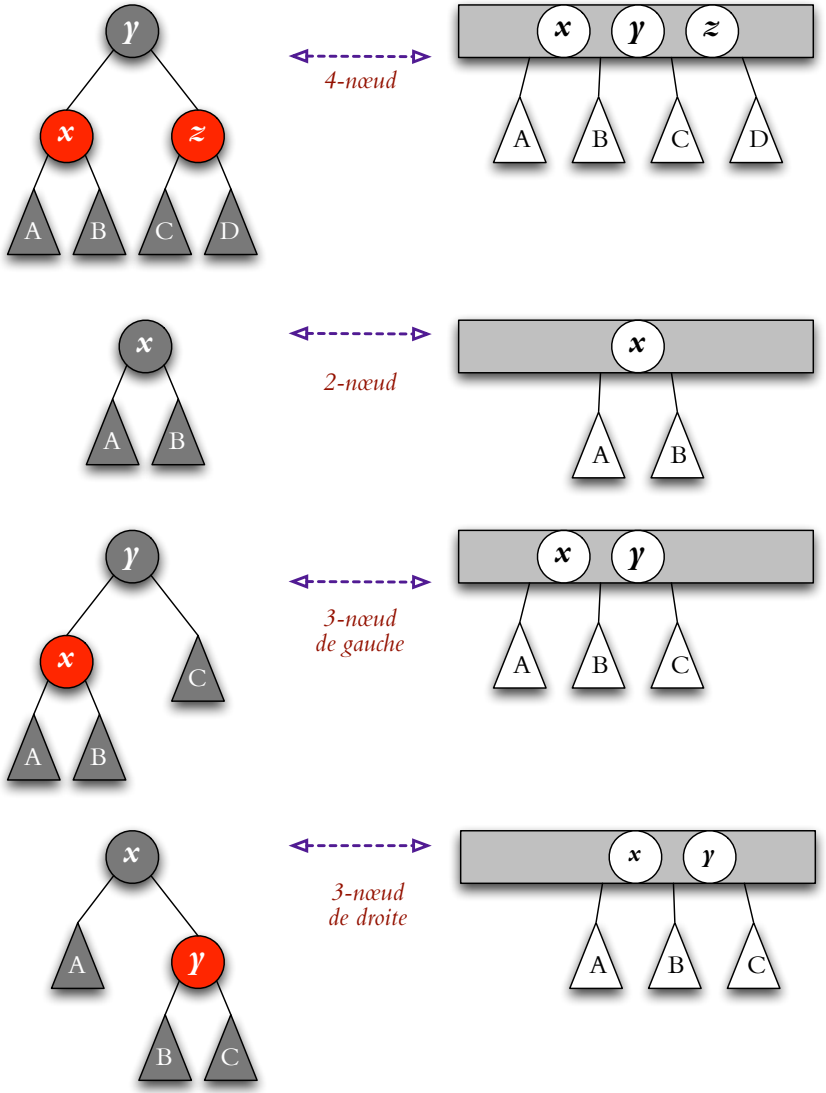
Arbre 2-3-4

⋆ arbre de recherche *non-binaire*
⋆ nœud interne peut avoir 2, 3 ou 4 enfants et 1,2 ou 3 clés
⋆ tous les nœuds externes sont au même niveau

☞on peut transformer un arbre rouge-et-noir en un arbre 2-3-4 facilement :

fusionner les nœuds rouges et leurs parents noirs
→ nœuds composés avec 2,3 ou 4 enfants et 1,2, ou 3 clés

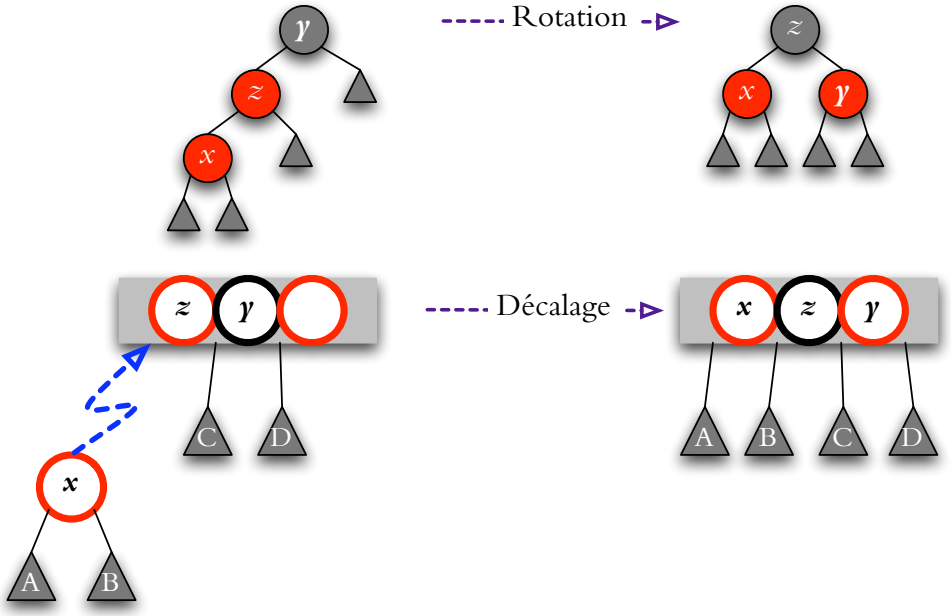
Arbre RN $\leftrightarrow$ arbre 2-3-4



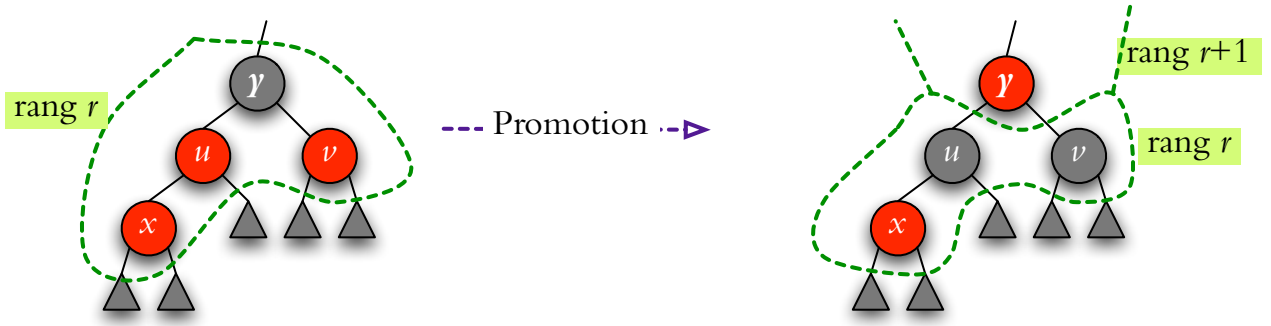
Insertion dans l'arbre 2-3-4

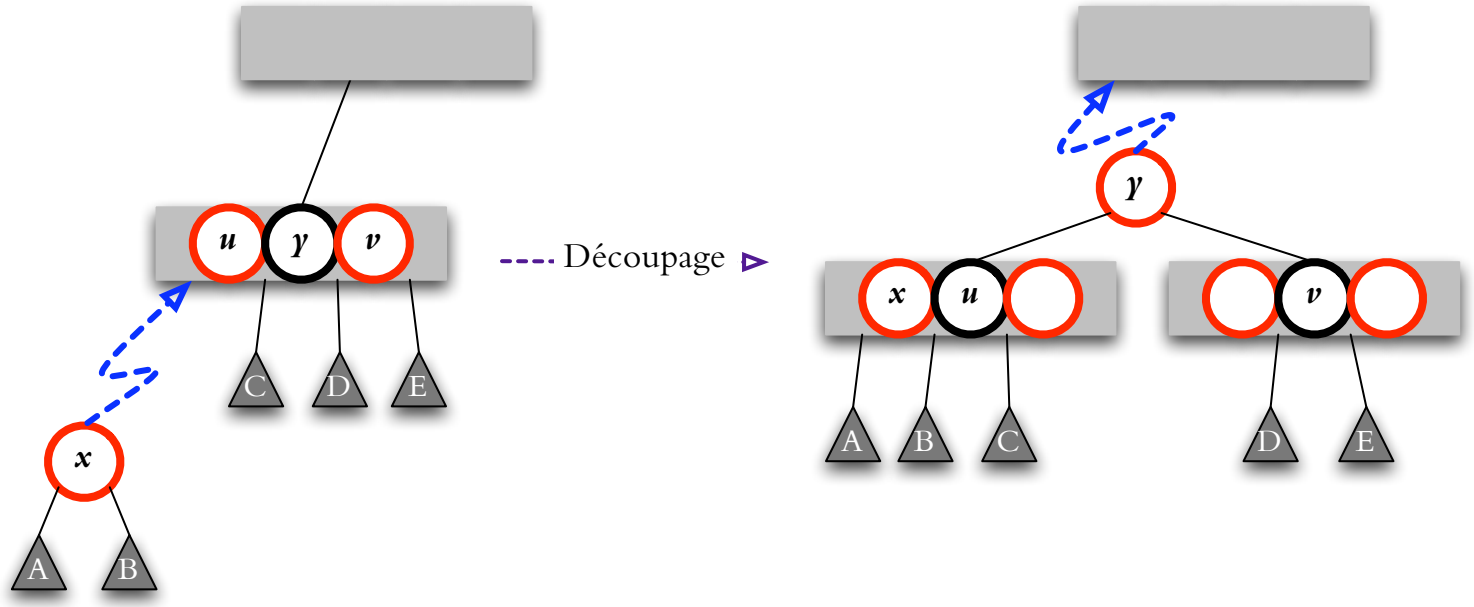
Qu'est-ce qui se passe lors d'une insertion ?
On crée un nœud rouge : promotions+rotations en ascendant vers la racine

Rotation : nœud noir avec un enfant rouge et son grand-enfant rouge transformé en un nœud noir avec deux enfants rouges



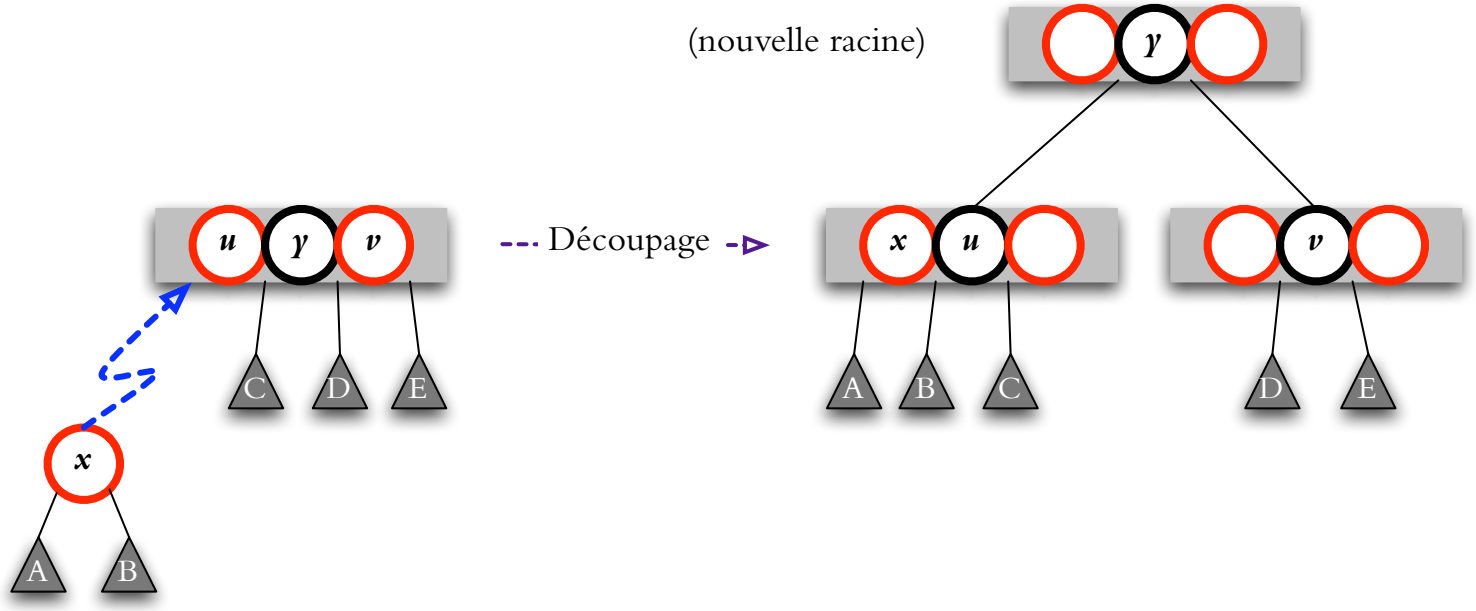
Promotion et découpage





Promotion à la racine

Cas spécial : promotion de la racine



$\Rightarrow$ la hauteur de l'arbre croît par le découpage de la racine

(arbre binaire de recherche : la hauteur croît par l'ajout de feuilles)

Recherche externe

Recherche dans mémoire externe

enregistrements avec clés dans une banque de données —— comment chercher ?

On veut minimiser l'accès au disque dur :
plus lent (par bcp de magnitudes) que l'accès au mémoire vive

Concept principal : **page** = bloc contigu de données (unité de lecture/écriture sur le disque)

(++ contextes : «sectors/tracks» sur disque dur, RAID, système d'exploitation, base de données, swapping du mémoire virtuel)

Taille typique : 1k, 4k, 8k, . . . octets

Accéder à une partie de la page est aussi couteux que d'accéder à toute la page

Opération typique pour caractériser la performance : (1er) accès à une page = **probe**

Accès séquentiel indexé

indexed sequential access : supporte la recherche rapide par clé dans un gros fichier

tableau d'indices (*index table*) — structure de données pour clés ordonnées

⋆ chaque cellule contient une paire de

$$\big(\mathsf{clé}, \mathsf{addr}\big)$$

addr : addresse sur le disque (d'une page)

⋆ chaque page contient M cellules :

$$M = \left\lfloor \frac{\text{taille de la page}}{|\mathsf{clé}| + |\mathsf{add}|} \right\rfloor$$

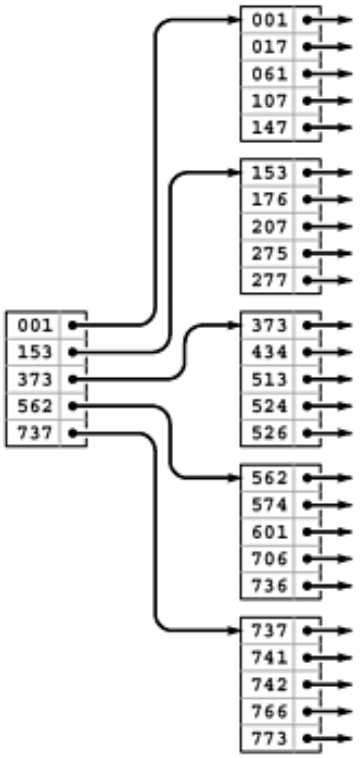
$\Rightarrow$ arbre M-aire, copier (min-)clés aux nœuds internes (chacun sur 1 page), stocker enregistrements aux nœuds externes

recherche : nombre de pages à consulter = $\left\lfloor \log_M(n+1) \right\rfloor$

$M = 1024$ ou $M = 256$ sont valeurs typiques $\Rightarrow$ «pratiquement» constante !

mais il faut refaire quasiment tout après insertion ou suppression

Accès indexé



☞même structure pour accès rapide par position dans un fichier :
mettre clé = position/indice de l'enregistrement

Arbre de recherche

Stocker un arbre rouge et noir ? ?

Arbre 2-3-4 est plus efficace : on modifie «quelques» nœuds seulement

Comment améliorer ?
on généralise à M sous-arbres au lieu de 4

Arbre B

Données stockées aux nœuds externes : L enregistrements par nœud

Clés stockées aux nœuds internes : $(M-1)$ clés à un nœud interne, M enfants

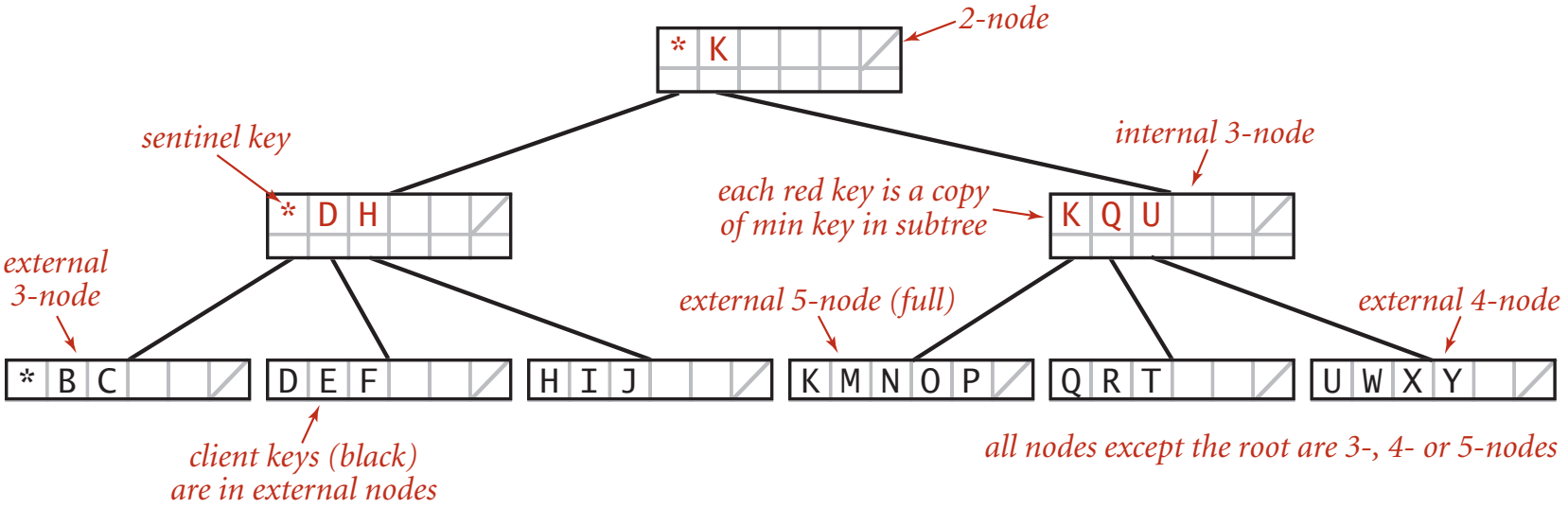
Clé i : valeur minimale dans le sous-arbre $(i+1)$

Racine : $2..M$ enfants
Nœuds internes : $\lceil M/2 \rceil..M$ enfants (taille=$M \cdot |\mathsf{clé}| + (M-1) \cdot |\mathsf{addr}|$)
Externes : $\lceil L/2 \rceil..L$ enregistrements (taille= $L \cdot |\mathsf{enregistrement}|$)

Nœuds externes àla même profondeur

Choix de M et L : on veut une page par nœud

Arbre B



Anatomy of a B-tree set (M = 6)

Arbre B

Thm. La hauteur h de l'arbre B est bornée par

$$h \leq 1 + \log_{\lceil M/2 \rceil} \frac{N}{2} \leq 1 + \frac{\lg \frac{n}{L}}{\lg M - 1}$$

où N est le nombre de nœuds externes et n est le nombre d'enregistrements.

Exemple : blocs de 8k, clés de 32 octets, enregistrements de 256 octets, $M = 228$, $L = 32$
$h = 4$ suffit jusqu'à $N = 2.9 \cdot 10^6$ ou $n = 47 \cdot 10^6$

$\Rightarrow$ nombre d'accès au disque est determiné par h : très peu (en plus, on peut garder la racine et peut-être même le premier niveau en mémoire principale)

Arbre B : insertion

comme avec arbre 2-3-4

Insertion d'un enregistrement : s'il y a de la place dans la page, aucun problème

s'il n'y a pas de place : **débordement**

solution : découpage $\longrightarrow$ éléments distribués entre deux pages/nœuds de tailles $\lfloor \frac{L}{2} \rfloor + 1$ et $\lceil \frac{L}{2} \rceil$.

peut causer un débordement au parent : découpage (ou «éclatement») si nécessaire (tailles $\sim M/2$) en ascendant vers la racine

$\Rightarrow$ la hauteur croît en découpant la racine

Arbre B (cont)

Suppression d'un élément : si la page est toujours assez remplie, aucun problème

et si le nombre d'éléments tombe en-dessous de $\lceil L/2 \rceil$?

1. prendre des éléments des sœurs immédiates

2. si elles sont au minimum, alors fusionner les pages $\rightarrow$ le parent perd un enfant

3. continuer avec le parent de la même manière si nombre d'enfants $< \lceil M/2 \rceil$

$\Rightarrow$ la hauteur décroît en enlevant la racine (quand elle a un enfant seulement)

Tri en temps linéaire

Tri linéaire ?

☞ distribution uniforme (ou connue) : $O(n)$ en espérance

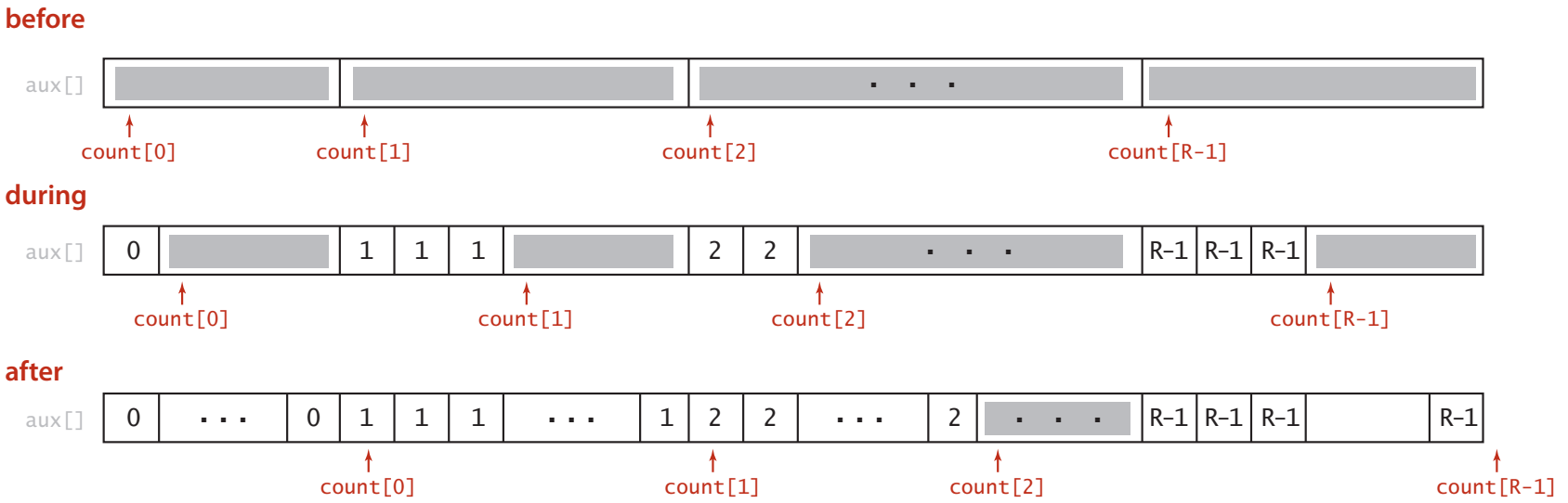
p.e., clés numériques uniformes sur $[0.0, 1.0]$: partitionner en n intervalle : chacune de taille 1 en espérance
($\approx$ hachage avec $f(x) = \lfloor n \cdot x \rfloor$)

☞ clés de b bits : $O(nb)$ au pire

Trier par compter

seulement R clés possibles : compter chacune + placer selon compte

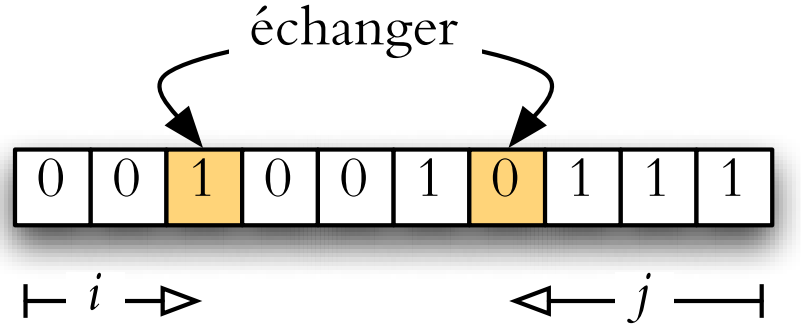
1. compter : `count[]=new int[R+1];`
2. calculer les cumuls = indices pour placement
3. remplir tableau auxiliaire, indices dans `count` mis à jour



4. recopier au tableau original

Tri de clés binaires

clé 1-bit – facile




```
     TRI01(A[0..n − 1])                                          // tri binaire
B1   i ← 0 ; j ← n − 1
B2   loop
B3       while i < j && A[i] = 0 do i ← i + 1
B4       while i < j && A[j] = 1 do j ← j − 1
B5       if i < j then échanger A[i] ↔ A[j]
B6       else return
```

Tri MSD

clé sur b bits — on peut trier en $O(nb)$

approche MSD (*most significative digit*) :
trier selon le bit de poids le plus significatif (BPS) + récurrence

```
TRIMSD(A[], g, d, k)
      // tri de A[g..d] selon bits k, k+1, ... ; k = b-1 est le BPS, k = 0 est le moins significatif
B1  if g ≥ d ou k < 0 then return
B2  i ← g; j ← d
B3  loop                                                  // tri binaire selon le k-ème bit
B4      while bit(A[i], k) = 0 et i < j do i ← i + 1
B5      while bit(A[j], k) = 1 et i < j do j ← j − 1
B6      if i < j then échanger A[i] ↔ A[j] else sortir de la boucle
B7  if bit(A[i], k) = 0 then j ← j + 1                    // au cas où A[g..d] ≡ 0
B8  TRIMSD(A, g, j − 1, k − 1)
B9  TRIMSD(A, j, d, k − 1)
```

Tri LSD

clé / string sur b caractères

approche LSD (*least-significant digit*) : sur toute position, du moins significatif vers le plus significatif, trier avec un tri stable⋆

⋆**tri stable** = ordre d'éléments avec clés égales est préservé

tri comptage est stable ! (tri par fusion aussi)

```
for (int k = 0; k<b; k++)
{
   int[] count = new int[R+1];                    // 0<=A[i]<R
   for (int i=0; i<A.length; i++) count[A[i].+1]++;
   for (int r=0, r<R; r++) count[r+1] += count[r]; // cumuls
   // ... tri comptage
}
```

Tri LSD 2

exemple :

```
4PGC938  2IYE230  3CIO720  2IYE230  2RLA629  1ICK750  3ATW723  1ICK750  1ICK750
2IYE230  3CIO720  3CIO720  4JZY524  2RLA629  1ICK750  3CIO720  1ICK750  1ICK750
3CIO720  1ICK750  3ATW723  2RLA629  4PGC938  4PGC938  3CIO720  1OHV845  1OHV845
1ICK750  1ICK750  4JZY524  2RLA629  2IYE230  1OHV845  1ICK750  1OHV845  1OHV845
1OHV845  3CIO720  2RLA629  3CIO720  1ICK750  1OHV845  1ICK750  1OHV845  1OHV845
4JZY524  3ATW723  2RLA629  3CIO720  1ICK750  1OHV845  2IYE230  2IYE230  2IYE230
1ICK750  4JZY524  2IYE230  3ATW723  3CIO720  3CIO720  4JZY524  2RLA629  2RLA629
3CIO720  1OHV845  4PGC938  1ICK750  3CIO720  3CIO720  1OHV845  2RLA629  2RLA629
1OHV845  1OHV845  1OHV845  1ICK750  1OHV845  2RLA629  1OHV845  3ATW723  3ATW723
1OHV845  1OHV845  1OHV845  1OHV845  1OHV845  2RLA629  1OHV845  3CIO720  3CIO720
2RLA629  4PGC938  1OHV845  1OHV845  1OHV845  3ATW723  4PGC938  3CIO720  3CIO720
2RLA629  2RLA629  1ICK750  1OHV845  3ATW723  2IYE230  2RLA629  4JZY524  4JZY524
3ATW723  2RLA629  1ICK750  4PGC938  4JZY524  4JZY524  2RLA629  4PGC938  4PGC938
```

[Sedgewick & Wayne]

Données en ADN

Structure de données en ADN

Fun : utiliser l'ADN pour stocker de l'information

ADN : molécule dont la structure primaire est linéaire — séquence composée de quatre nucléotides (A, C, G, T)

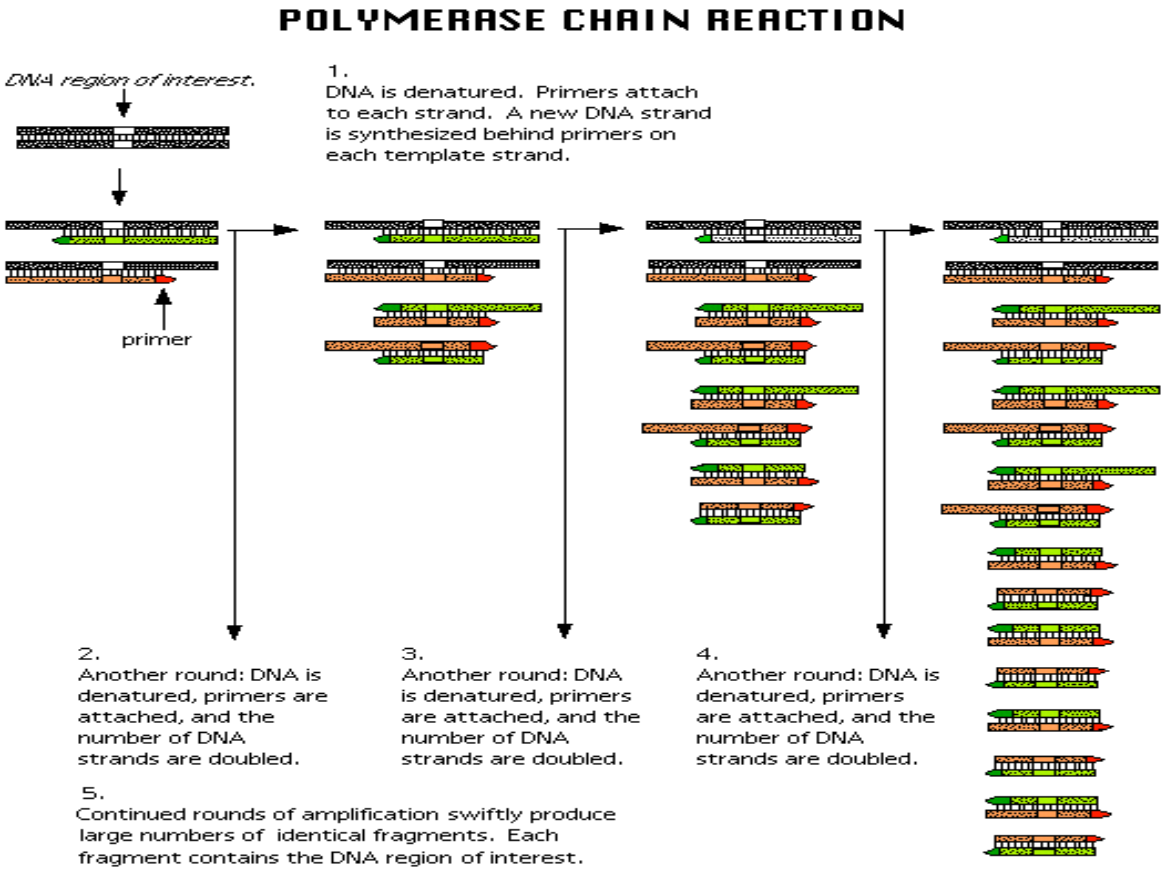
C'est durable (v. ADN de l'Homme de Néanderthal), efficace (séquençage et manipulation de très petites quantités)

On peut représenter 2 bits par nucléotides

Opérations élémentaires du «hardware»

- écriture : synthèse
- lecture : séquençage
- pointeurs : PCR

Polymerase Chain Reaction



Access Excellence

Encodage

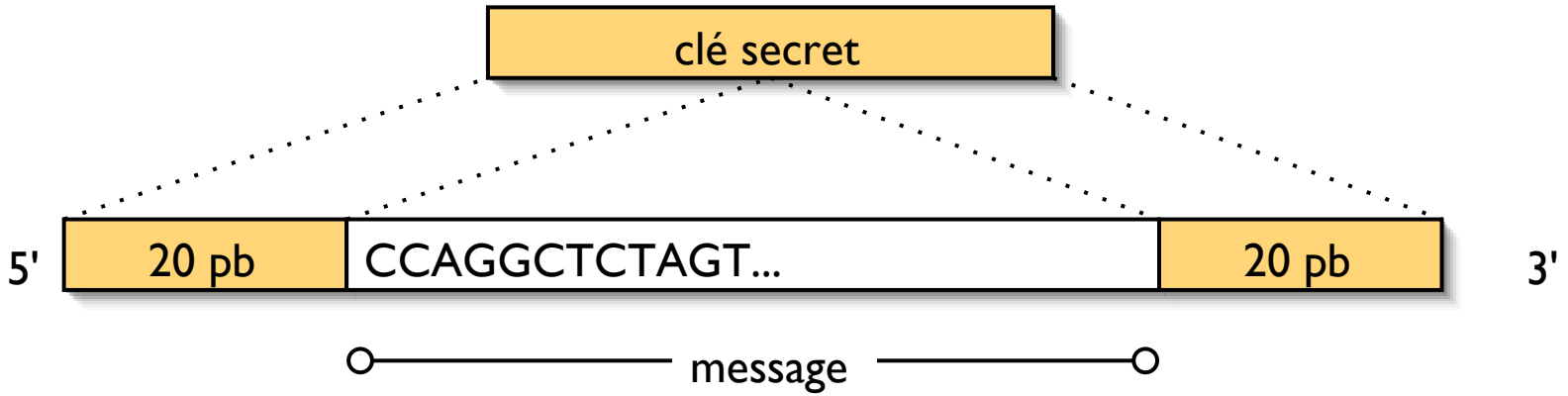
Encodage d'information par ADN : c'est presque trivial :)

Exemple : message secret (*microdot*)

```
A=CGA    K=AAG
B=CCA    L=TGC
C=GTT    M=TCC
...
```

«Bonjour la tristesse» $\longrightarrow$ `CCAGGCTCTAGT...`

Message secret – 2

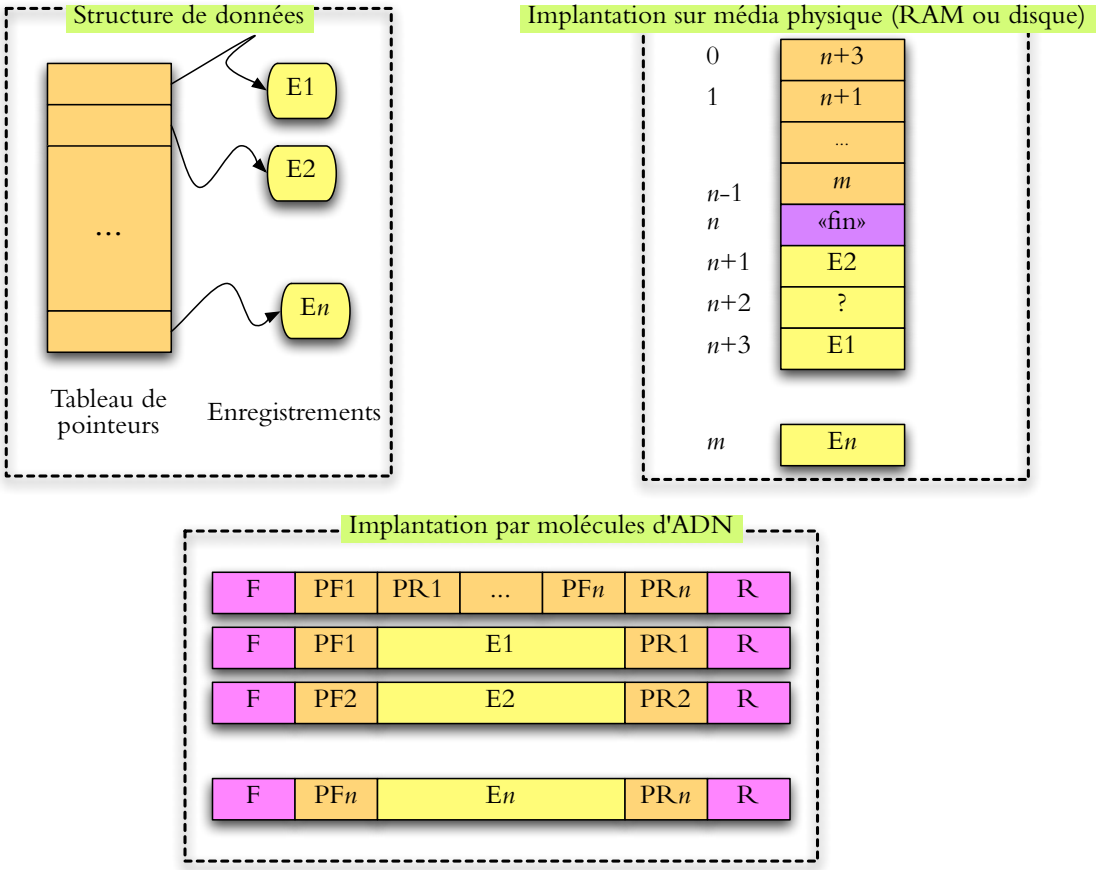


mélanger le message secret avec d'autres molécules ADN

amplification du message par PCR — il faut savoir le clé secret (nombre de possibilités : 4^{40})

Clelland & al. *Nature* 399 :533 (1999)

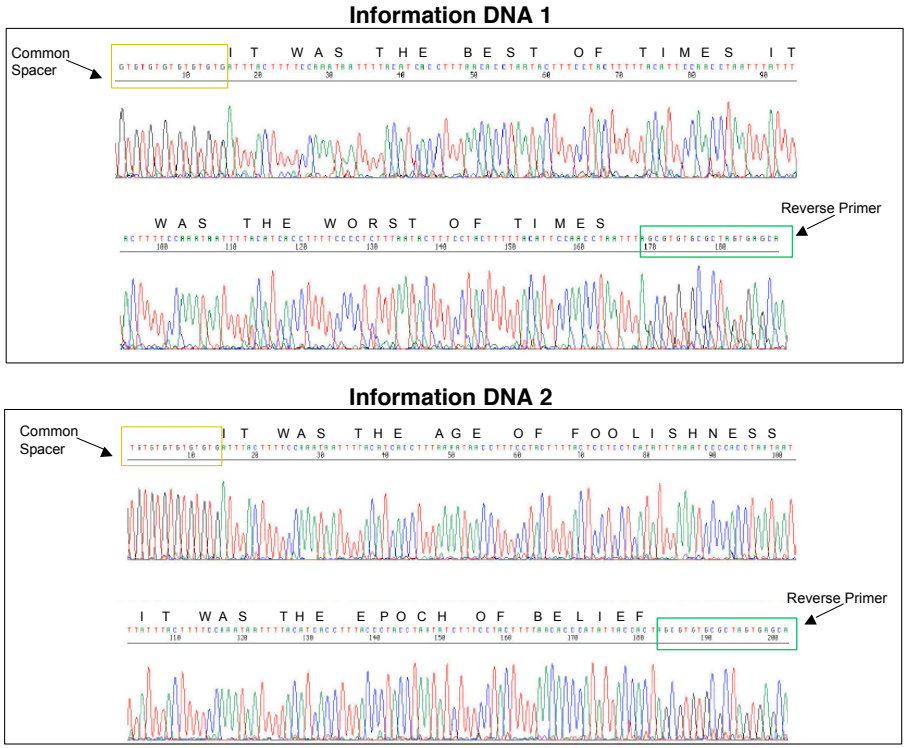
Stockage d'information



Bancroft & al. *Science* 293 :1763 (2001)

Un vrai exemple

[livre stocké en ADN]



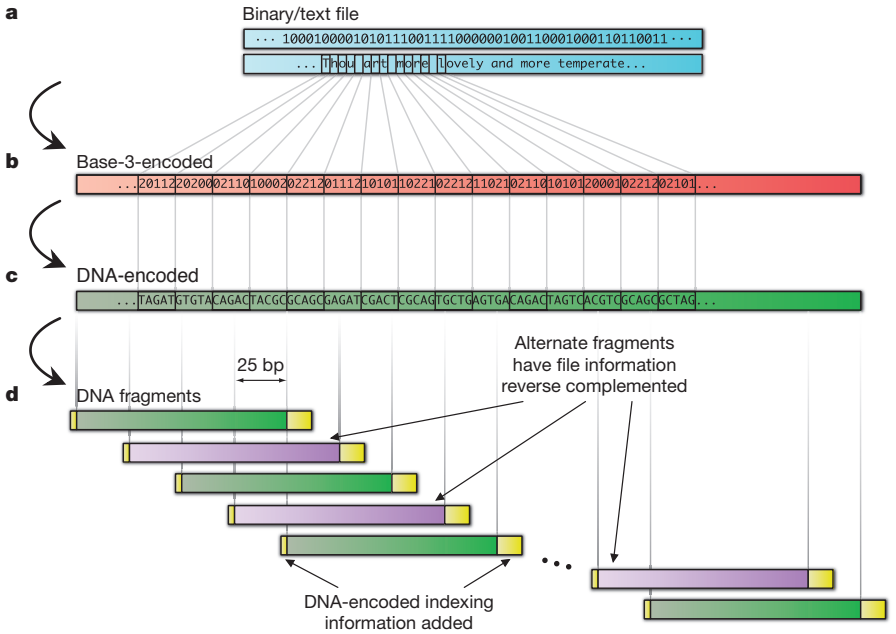
Experimental prototype: Readout (sequencing plus decoding) of Information DNAs.

Bancroft & al. *Science* 293 :1763 (2001)

Meilleur encodage

encodage de correction d'erreur en ternaire

$\longrightarrow$ séquence ADN sans nucléotides répétés (p.e., 'AAA')

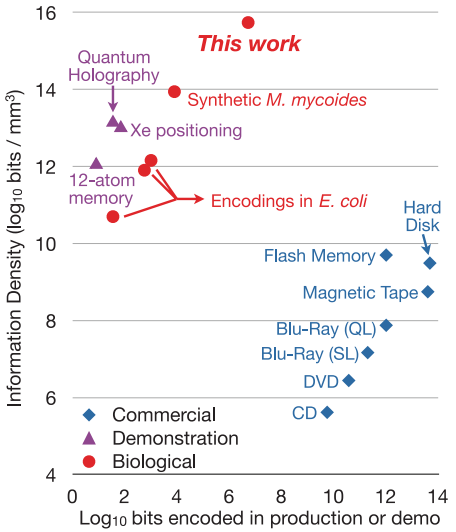


5 fichiers, 5×10^6 bits

Goldman & al. 2013

Densité d'information

très grande densité d'information



pas cher à maintenir

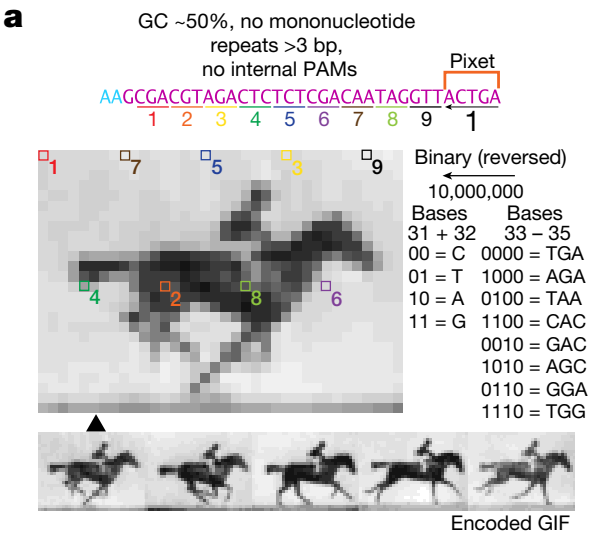
[Church, Gao & Kosuri 2012]

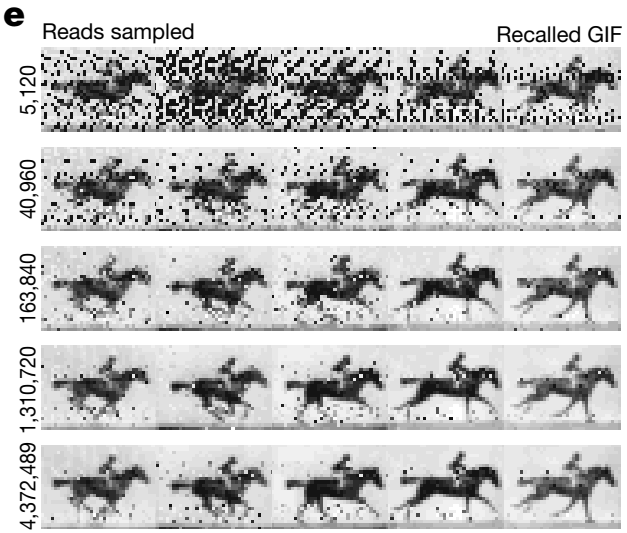
doi:10.1038/nature23017

CRISPR–Cas encoding of a digital movie into the genomes of a population of living bacteria

Seth L. Shipman[1,2,3], Jeff Nivala[1,3], Jeffrey D. Macklis[2] & George M. Church[1,3]

encoder le fichier du film (GIF animé) + insérer dans des bactéries + faire croître la population





// prochaine étape : bactéries qui peuvent enregistrer une obsérvation dans leur

génome