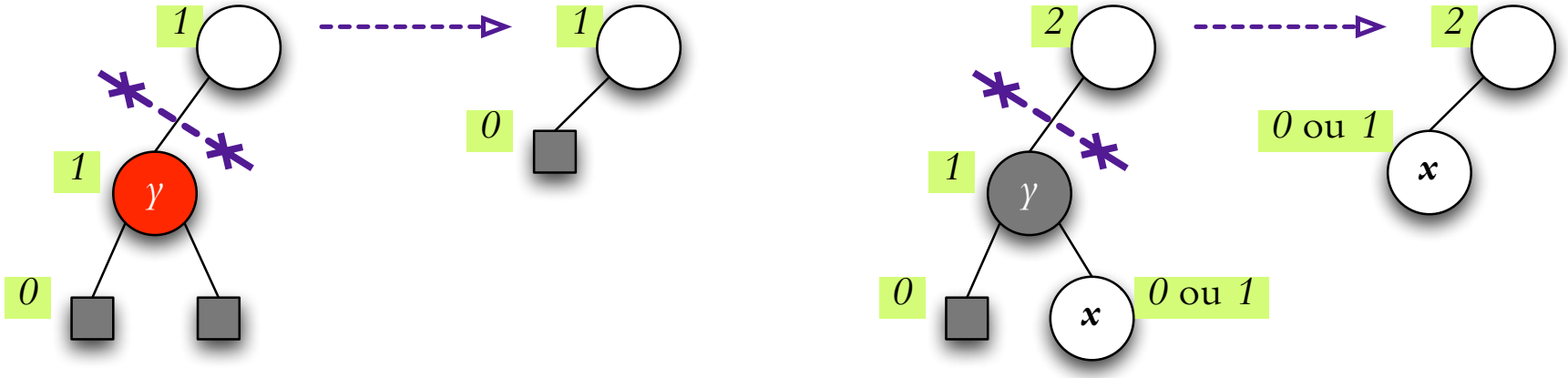
# Arbres RN — suppression

Et suppression d'un nœud ?

Technique similaire : procéder comme avec l'arbre binaire de recherche, puis retrogradations en ascendant vers la racine + $O(1)$ rotations (trois au plus) à la fin

# Arbres RN — problème

On enlève un nœud $y$ : remplacement par null (si aucun enfant) ou par l'enfant non-null. Ce dernier peut être de rang trop petit (nœud noir remplacé par nœud noir $x$).




Plusieurs cas :

Cas 0 : nœud rouge $x$ : rétrogradation — il devient noir, et rien plus à faire

Cas 1 : nœud noir avec une sœur noire

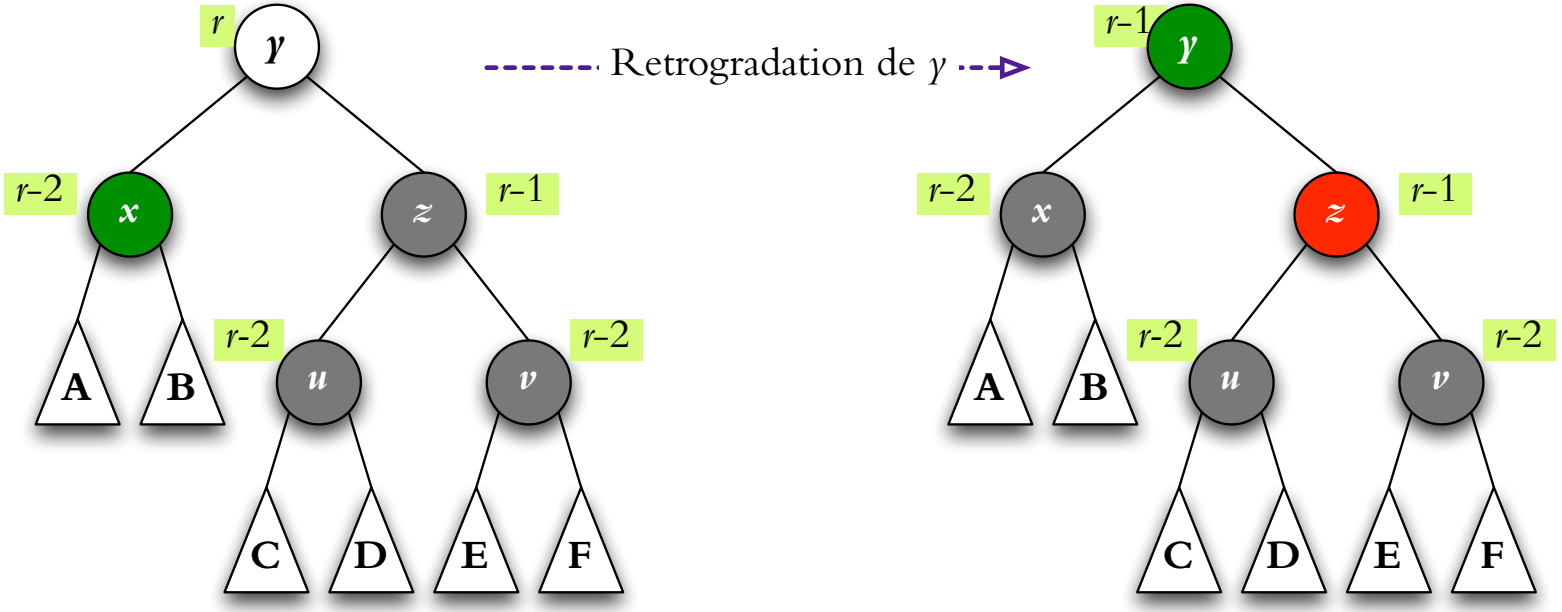
Cas 2 : nœud noir avec une sœur rouge

En cas 1, il faut vérifier la couleur des enfants de la sœur (les neveux)

# Arbres RN — suppression 1A

Cas 1A : sœur noire, neveux noirs

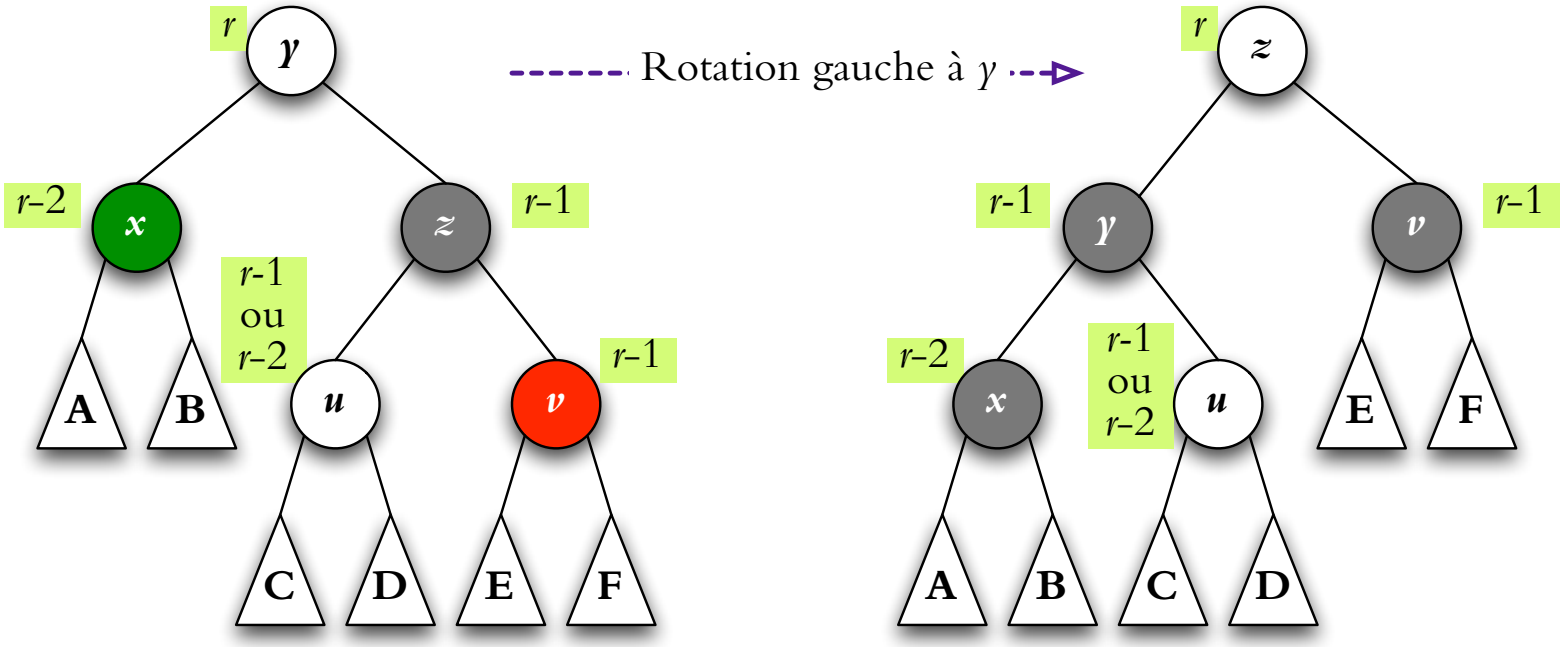
décrementer $\mathsf{rang}\big(\mathsf{parent}(x)\big)$ : continuer avec $x \leftarrow \mathsf{parent}(x)$ en cas 0,1 ou 2.



# Arbres RN — suppression 1B

Cas 1B : sœur noire, neveu distant ($v$) rouge

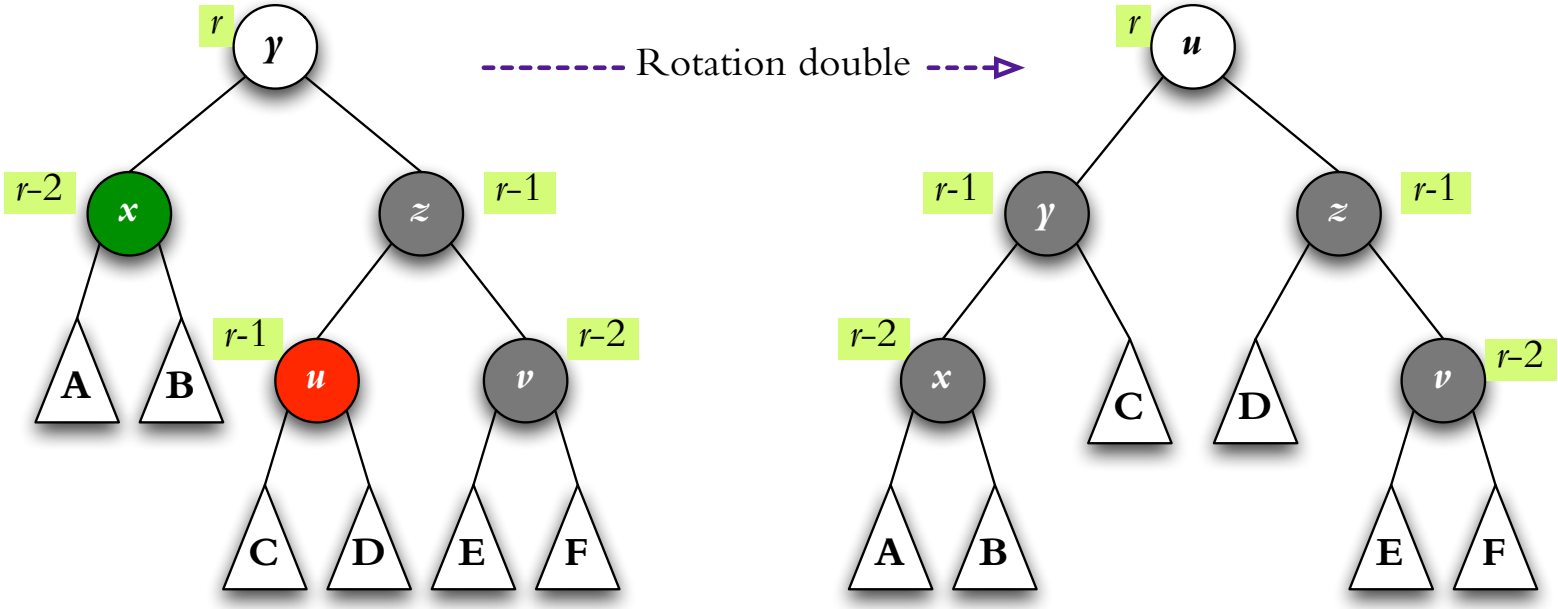
faire une rotation simple et arrêter



# Arbres RN — suppression 1C

Cas 1C : sœur noire, neveu proche ($u$) rouge

faire une rotation double et arrêter

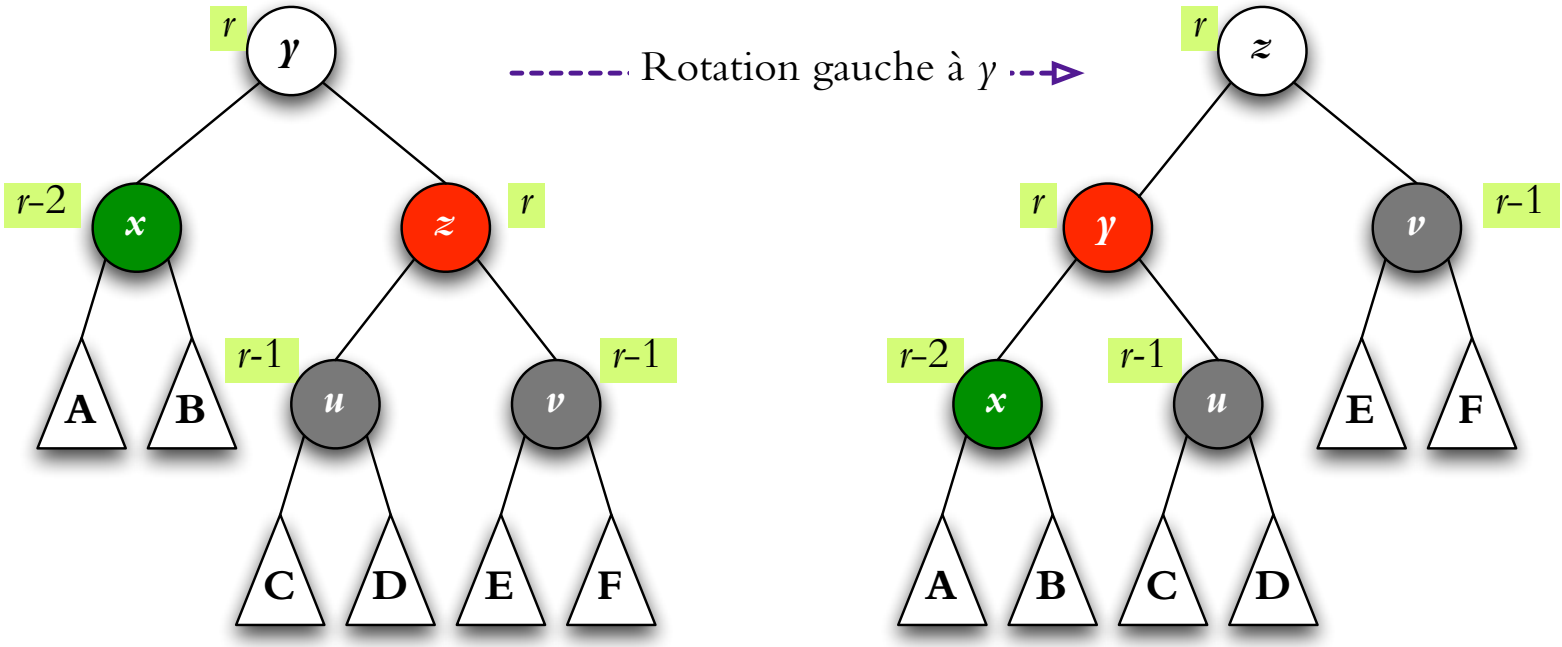



(quand $\mathsf{rang}(v) = r - 1$, on peut toujours faire cette rotation mais cas 1B est plus rapide)

# Arbres RN — suppression 2

Cas 2 : sœur rouge

faire une rotation simple et continuer en cas 1 avec $x$




si on continue en cas 1a (les enfants de $u$ sont noirs), alors la récursion se termine avec la rétrogradation de $y$ qui est rouge (cas 0 pour $y$)

# Arbres RN — efficacité

Un arbre rouge et noir avec $n$ nœuds internes et hauteur $h \in O(\log n)$.

**Recherche** : $O(h)$ mais $h \in O(\log n)$ donc $O(\log n)$

**Insertion** :
1. $O(h)$ pour trouver le placement du nouveau nœud
2. $O(1)$ pour initialiser les pointeurs
3. $O(h)$ promotions en ascendant si nécessaire
4. $O(1)$ pour une rotation simple ou double si nécessaire

$O(h)$ en total mais $h \in O(\log n)$ donc $O(\log n)$

**Suppression** : $O(\log n)$

Usage de mémoire : il suffit de stocker la couleur (1 bit) de chaque nœud interne (astuce pour épargner un bit par nœud : échanger les pointeurs gauche $\leftrightarrow$ droit pour les nœuds noirs — couleur testée par la comparaison des clés aux enfants)

# Arbre 2-3-4

Arbre **2-3-4** : c'est un arbre de recherche *non-binaire* où chaque nœud peut avoir 2, 3 ou 4 enfants et stocke 1,2 ou 3 valeurs
toutes les feuilles sont au même niveau

Équivaut à l'arbre rouge et noir : fusionner les nœuds rouges et leurs parents noirs.

# Transformation entre arbres RN et 2-3-4