Arbres rouge-noir

Motivation : Comprendre le fonctionnement d'une structure d'arbre équilibrée utilisée en pratique (ex : ordonnanceur de Linux).

Définition : Un arbre rouge-noir est un arbre binaire de recherche où chaque noeud porte une couleur (rouge ou noir) et vérifiant :

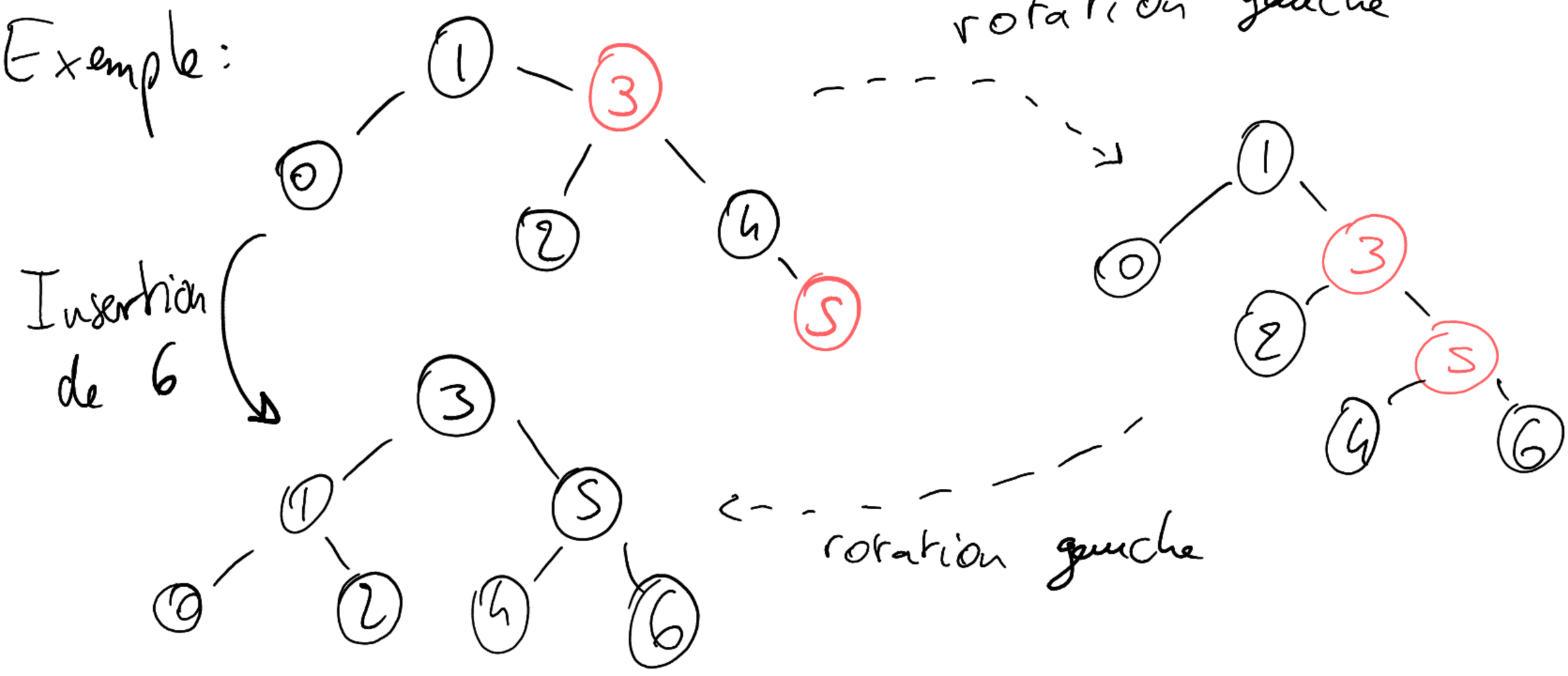
(1) le père d'un noeud rouge, s'il existe, est rouge

(2) le nombre de noeuds noirs le long d'un chemin de la racine à un sous-arbre vide est toujours le même.

Notations : Soit t un ARN. On note :

- $n(t)$: nombre de sommets de t
- $h(t)$: hauteur de t
- $b(t)$: quantité définie en (2).

Exemple :



Remarque : Tout sous-arbre d'un ARN est un ARN.

Propriété : Soit t un ARN. Alors, on a :

(i) $h(t) \leq 2b(t)$

(ii) $2^{b(t)} \leq n(t)+1$

Preuve : Par induction structurelle sur t (voir au 2ti(s).

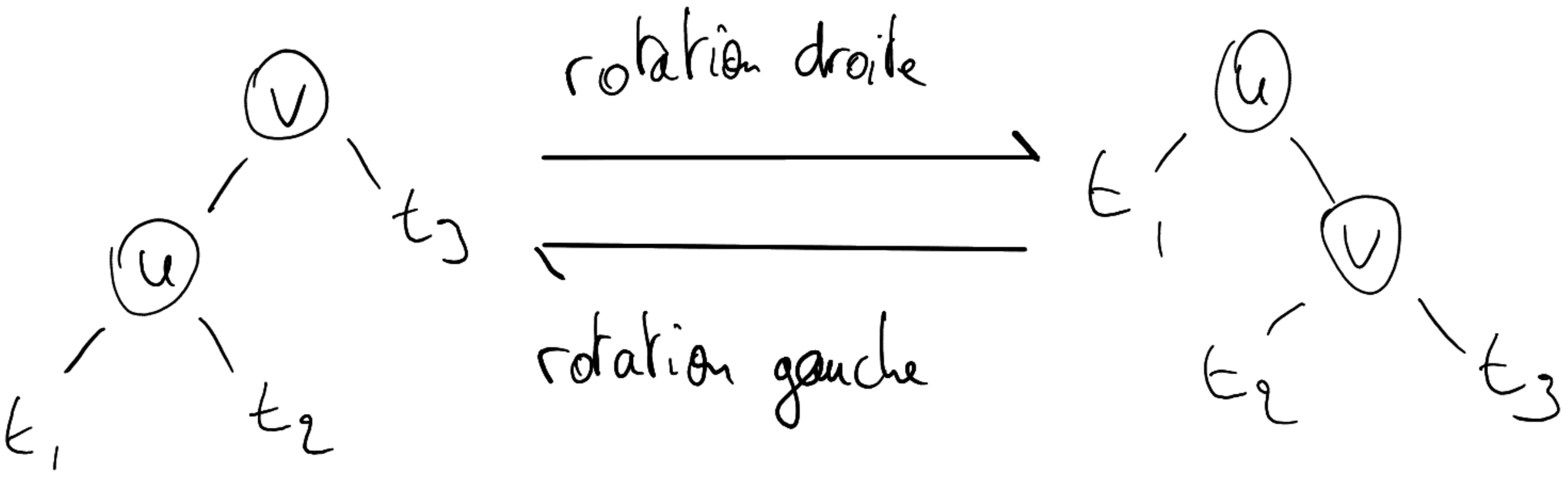
Corollaire : L'ensemble des ARN forme une famille d'arbres équilibrés.

Preuve : $h(t) \leq 2b(t) \leq 2\log_2(n(t)+1) \leq 4\log_2(n(t))$

$h(t) \in \mathcal{O}(\log(n(t)))$

↳ Recherche (find) en $\mathcal{O}(\log(n))$

Rotations

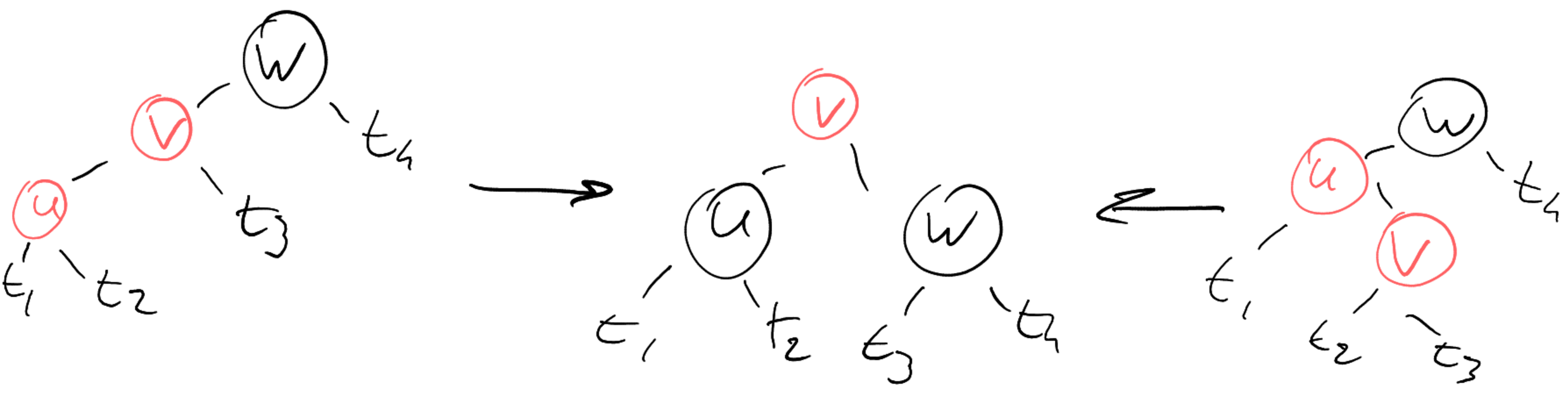


Insertion → Pb : garantir la propriété des ARN

↳ Idée : . Insertion d'un sommet rouge comme pour les ABR.

. Si l'arbre est un ARN, ok, sinon (si n'est plus vérifié : on effectue des rotations pour rétablir la structure ARN.

- On colorie la racine en noir.



Exemple : Initial à compléter

Complexité : Insertion initiale : $O(\log(n))$

Rotations : $O(1)$ / rotation

Il y en a au plus $h(t)$ car l'arbre est équilibré.

Donc : $O(\log(n))$.