

# Arbres AVL

AV - L

Adel'son - Vel'skii

Landis (1962)

# Arbres AVL

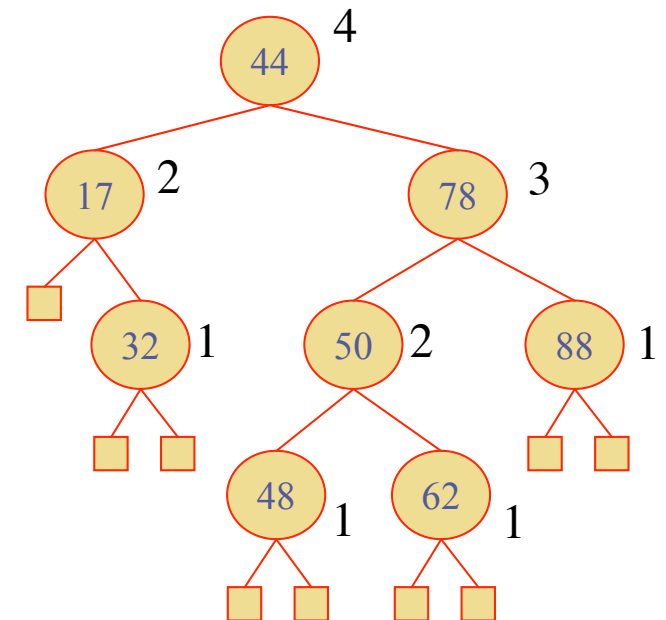
- Un arbre AVL est un arbre binaire de recherche **balancé** i.e un arbre binaire ayant les propriétés suivantes:

arbres  
binaires  
de  
recherche

- Soient  $u, v$  et  $w$  trois noeuds tels que  $u$  est dans le sous-arbre gauche de  $v$  et  $w$  dans son sous-arbre droit. Alors, on a

$$\text{clé}(u) < \text{clé}(v) \leq \text{clé}(w)$$

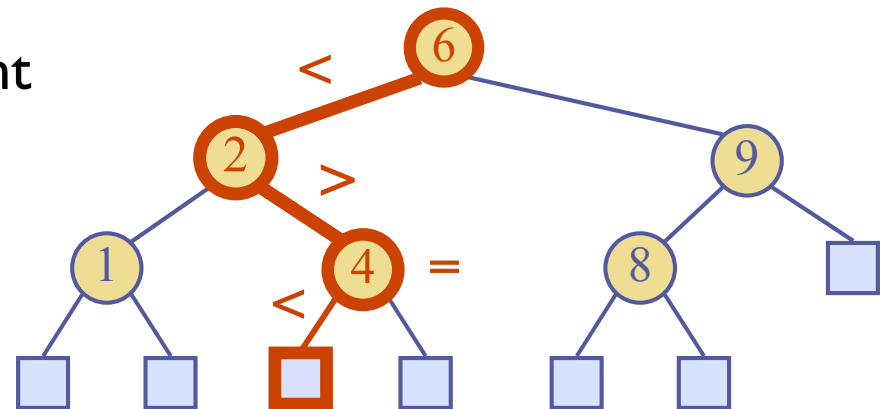
- Les éléments sont gardés en mémoire dans les noeuds internes
- Un parcours symétrique de l'arbre visite les clés en ordre croissant
- **Propriété de balance:** Pour chaque noeud interne  $v$ , la hauteur des enfants de  $v$  diffère d'au plus 1



- La hauteur d'un arbre AVL est en  $O(\log n)$

# Chercher dans les arbres AVL

- Pour chercher un élément de clé  $k$  dans un arbre AVL, on procède exactement de la même façon que pour la recherche dans les arbres binaires de recherche:
- **Exemple 1:** Chercher(4)
- Le prochain noeud visité dépend du résultat de la comparaison de  $k$  avec la clé du noeud dans lequel on se trouve.
- Si on trouve un noeud interne de clé  $k$ , on retourne la valeur correspondant à cette entrée de clé  $k$
- Sinon, on retourne NULL
  - **Exemple 1:** Chercher(3)



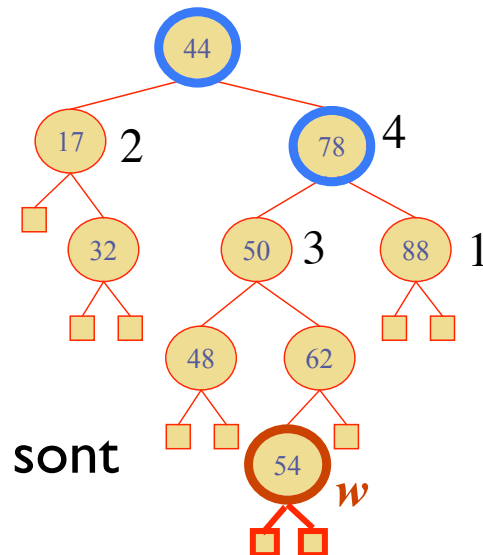
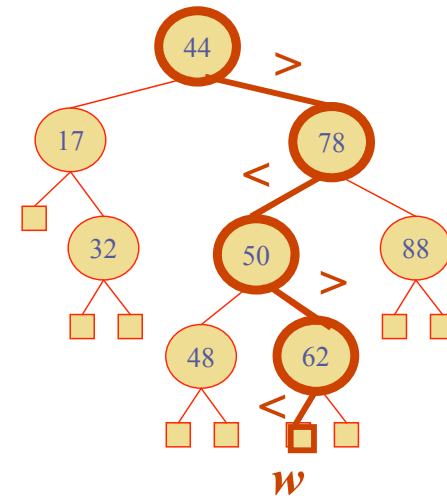
# Insérer dans les arbres AVL

- Pour insérer un élément  $(k,v)$  dans un arbre AVL, on commence par exécuter l'algorithme d'insertion d'un arbre binaire de recherche.

- **Exemple 1:** Insérer  $(54,v)$

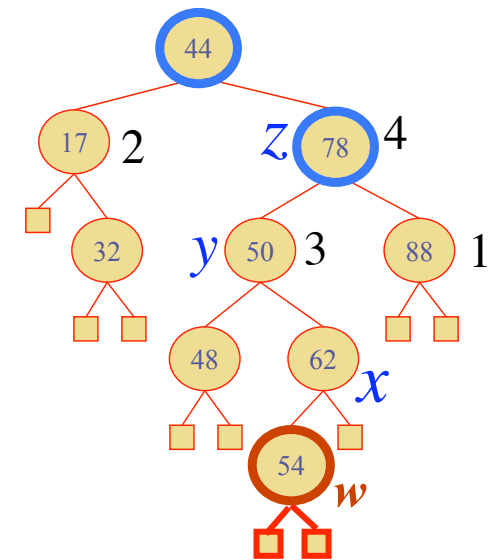
- On commence par exécuter  $\text{chercher}(k)$
- Si  $k$  n'est pas dans l'arbre l'algorithme  $\text{chercher}(k)$  se terminera dans une feuille  $w$
- On insère  $k$  dans  $w$  et on change  $w$  en un noeud interne
- Si  $k$  est dans l'arbre l'algorithme  $\text{chercher}(k)$  se terminera dans un noeud interne  $w$ . On applique alors récursivement l'algorithme  $\text{chercher}(k)$  sur le filsDroit de  $w$ , jusqu'à ce qu'on trouve une feuille.

- Après l'insertion de  $(54,v)$ , certains noeuds ne sont plus balancés!!



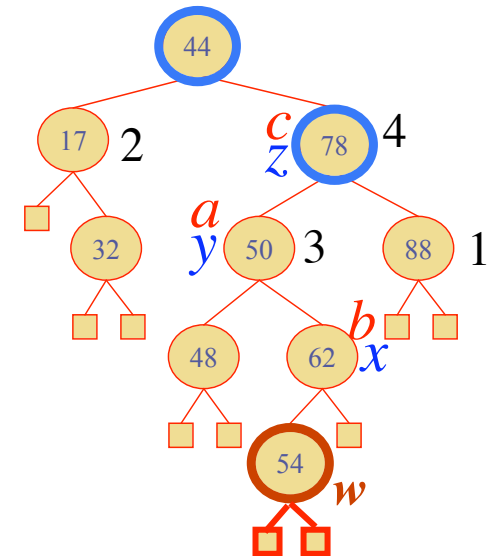
# Restructuration de l'arbre après une insertion

- Après l'insertion d'une entrée dans un arbre AVL, il est possible que certains noeuds internes soient débalancés.
- Pour retrouver la propriété de balance, on va devoir restructurer l'arbre comme suit:
  - Les noeuds débalancés sont situés sur le chemin du noeud  $w$  à la racine (pourquoi?)
  - On va nommer  $z$  le premier noeud débalancé qu'on trouve sur ce chemin.
  - On va nommer  $y$  le fils de  $z$  de plus grande hauteur
  - On va nommer  $x$  le fils de  $y$  de plus grande hauteur (si égalité, on choisit le fils qui est un ancêtre de  $w$ )



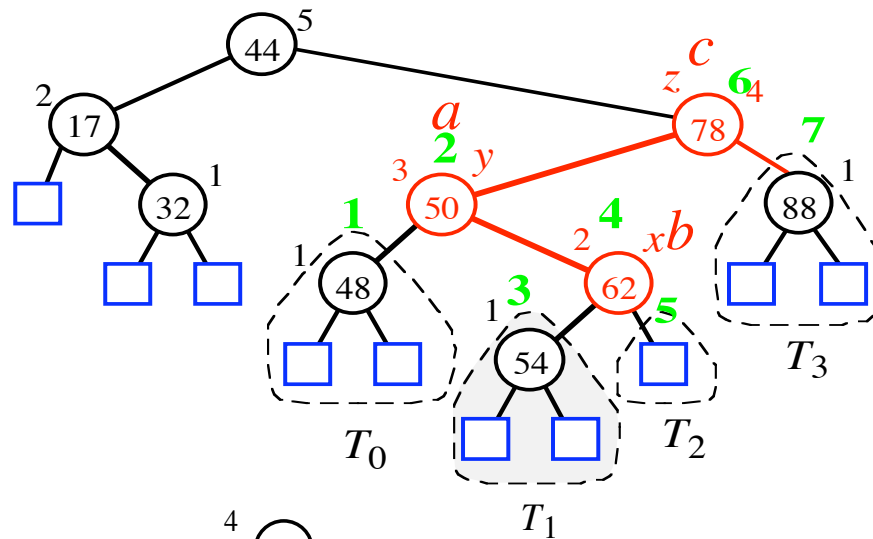
# Restructuration de l'arbre après une insertion (suite)

- Étant donné les noeuds internes  $x$ ,  $y$  et  $z$ , on renomme par “ $a$ ” le premier de ces sommets visités lors d'un parcours symétrique de l'arbre, par “ $b$ ”, le deuxième et par “ $c$ ”, le troisième.
- Appliquer un algorithme de restructuration qui va rebalancer le noeud interne  $z$  (rebalancement local)
  - Goodrich et Tamassia (tableau)
  - Rotations et lois associatives
- À la fin de la restructuration tous les noeuds internes seront balancés (rebalancement global)

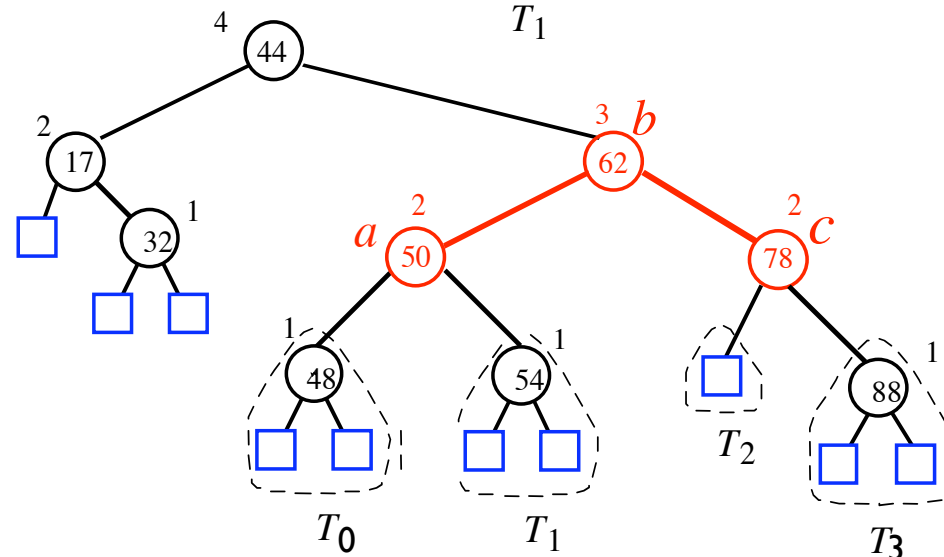


# Restructuration de l'arbre après une insertion version Goodrich et Tamassia:

débalancé...

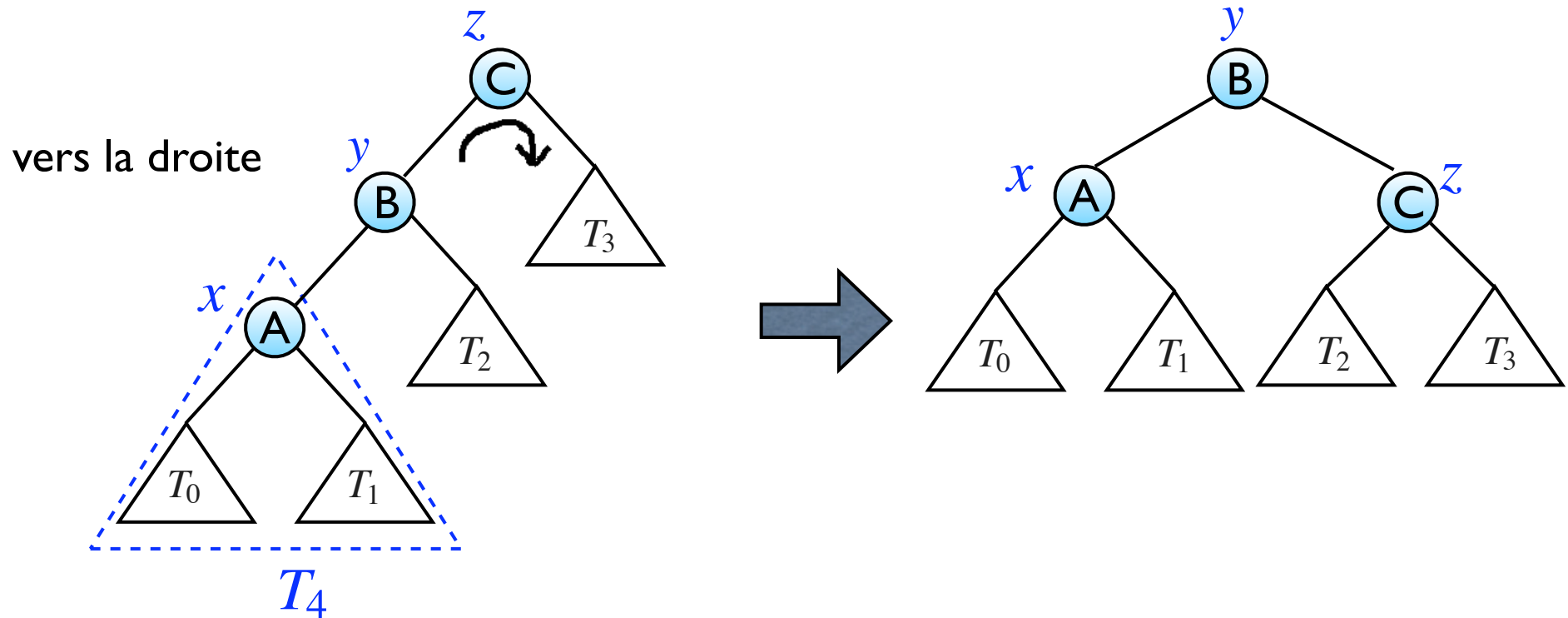


...balancé



# Rotations et lois associatives

## 1) Rotations simples

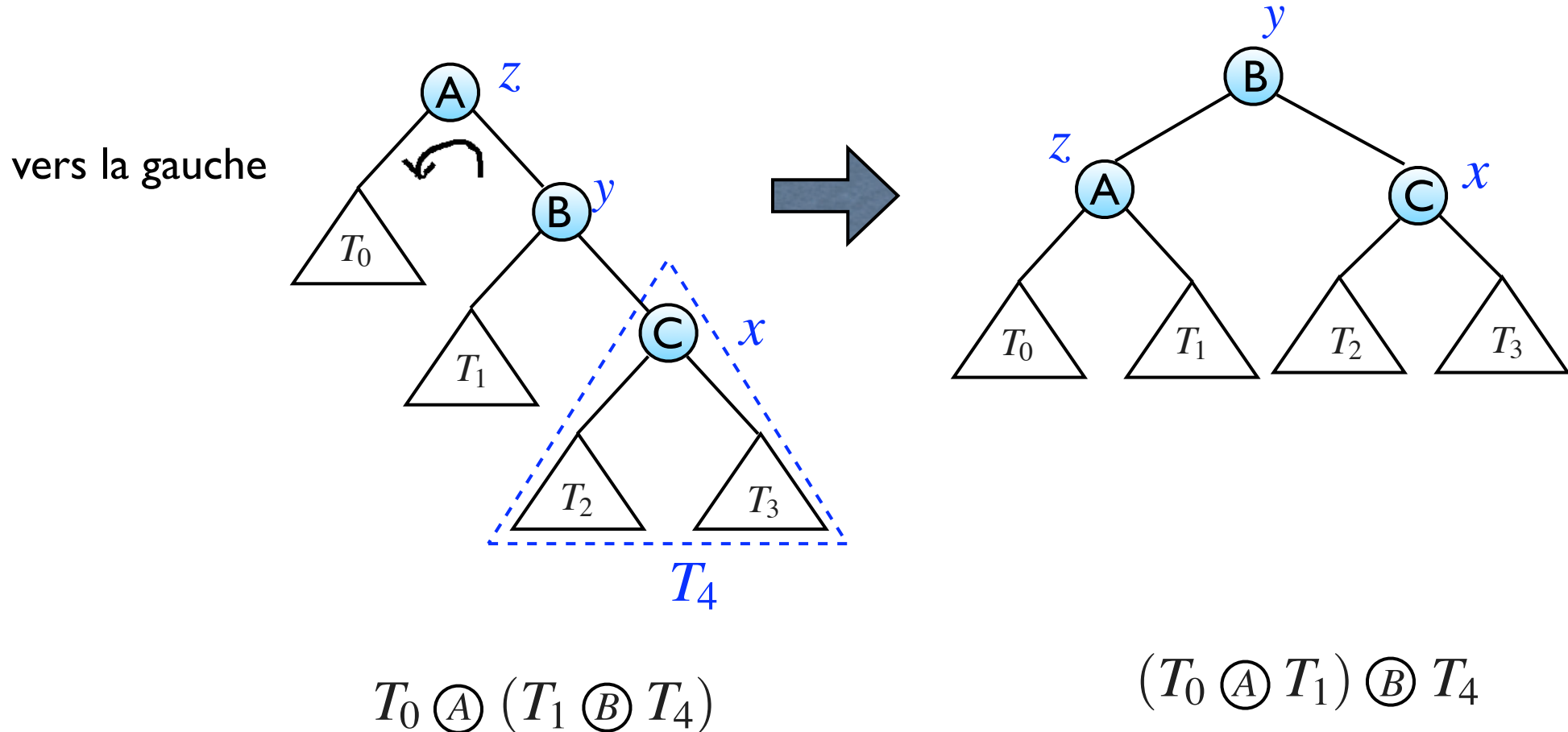


$$(T_4 \textcircled{B} T_2) \textcircled{C} T_3$$

$$T_4 \textcircled{B} (T_2 \textcircled{C} T_3)$$

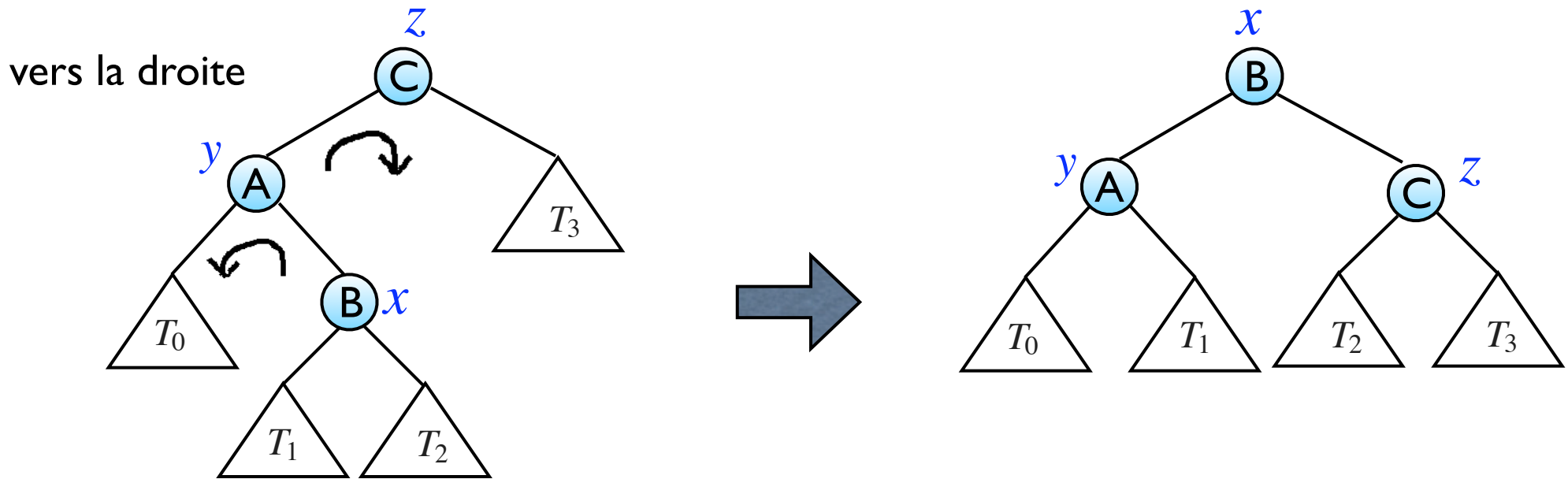
# Rotations et lois associatives (suite)

## 1) Rotations simples (suite)



# Rotations et lois associatives

## 2) Rotations doubles



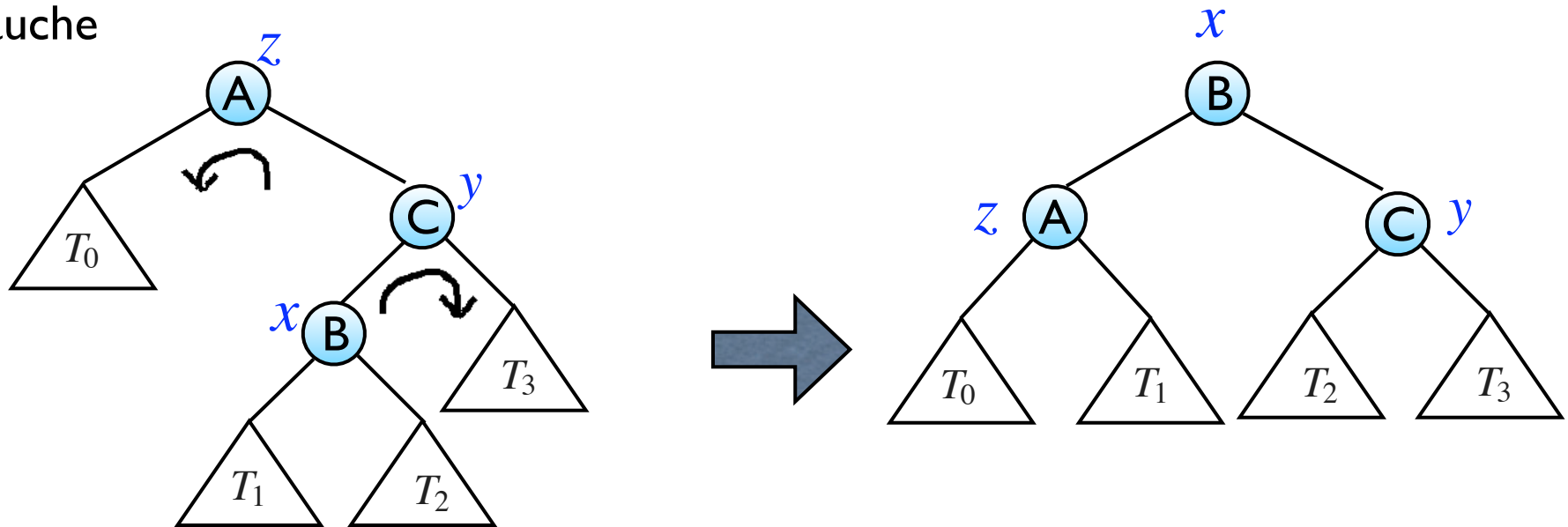
$$(T_0 \textcircled{A} (T_1 \textcircled{B} T_2)) \textcircled{C} T_3$$

$$(T_0 \textcircled{A} T_1) \textcircled{B} (T_2 \textcircled{C} T_3)$$

# Rotations et lois associatives (suite)

## 2) Rotations doubles (suite)

vers la gauche



$$T_0 \textcircled{A} ((T_2 \textcircled{B} T_3) \textcircled{C} T_4)$$

$$(T_0 \textcircled{A} T_1) \textcircled{B} (T_2 \textcircled{C} T_3)$$

# Supprimer dans les arbres AVL

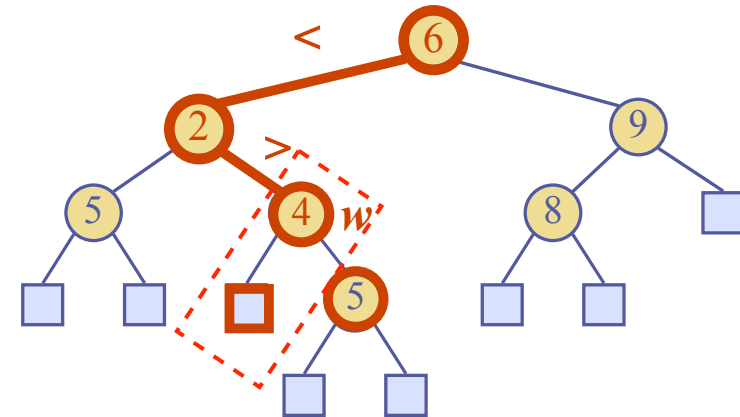
- Pour supprimer un élément de clé  $k$  dans un arbre AVL, on commence par exécuter l'algorithme de suppression d'un arbre binaire de recherche.
- Après la suppression, il est possible que certains noeuds internes soient déséquilibrés (en fait un seul noeud sera déséquilibré)
- On va faire une restructuration de l'arbre autour de ce noeud, ce qui causera un rééquilibrage local. Il est possible que cette restructuration cause le déséquilibrage d'un autre noeud.
- On va restructurer l'arbre tant qu'il y a un noeud déséquilibré sur le chemin de  $p$  à la racine, où  $p$  est le parent du noeud enlevé

# Supprimer dans un arbre binaire de recherche (rappel)

- Pour enlever un élément de clé  $k$  dans un arbre binaire de recherche, on commence par exécuter l'algorithme  $\text{chercher}(k)$ .

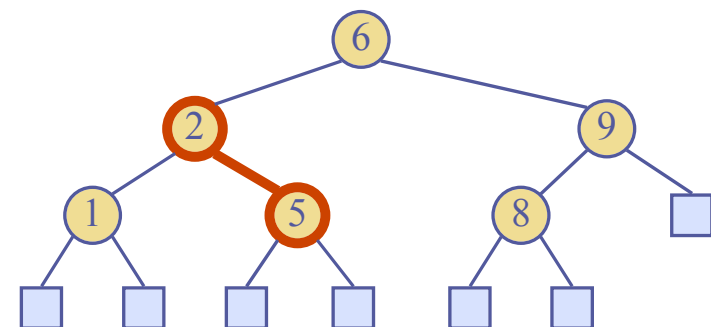
- **Exemple 1:** Enlever(4)

- Si  $k$  est pas dans l'arbre l'algorithme  $\text{chercher}(k)$  se terminera dans un noeud interne  $w$



- Si l'un des enfant de  $w$  est une feuille, on enlève cette feuille et  $w$

- Sinon...

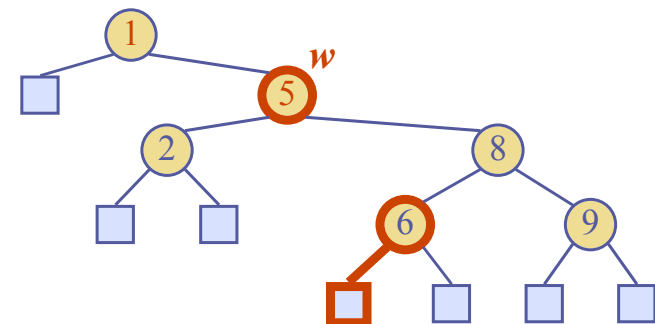
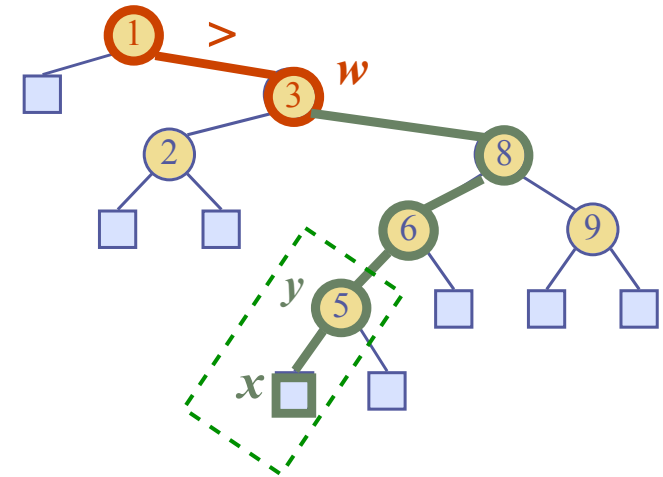


## Supprimer dans un arbre binaire de recherche (suite)

- Si  $k$  est pas dans l'arbre, l'algorithme  $\text{chercher}(k)$  se terminera dans un noeud interne  $w$ . Si les fils de  $w$  sont tous les deux des noeuds internes alors

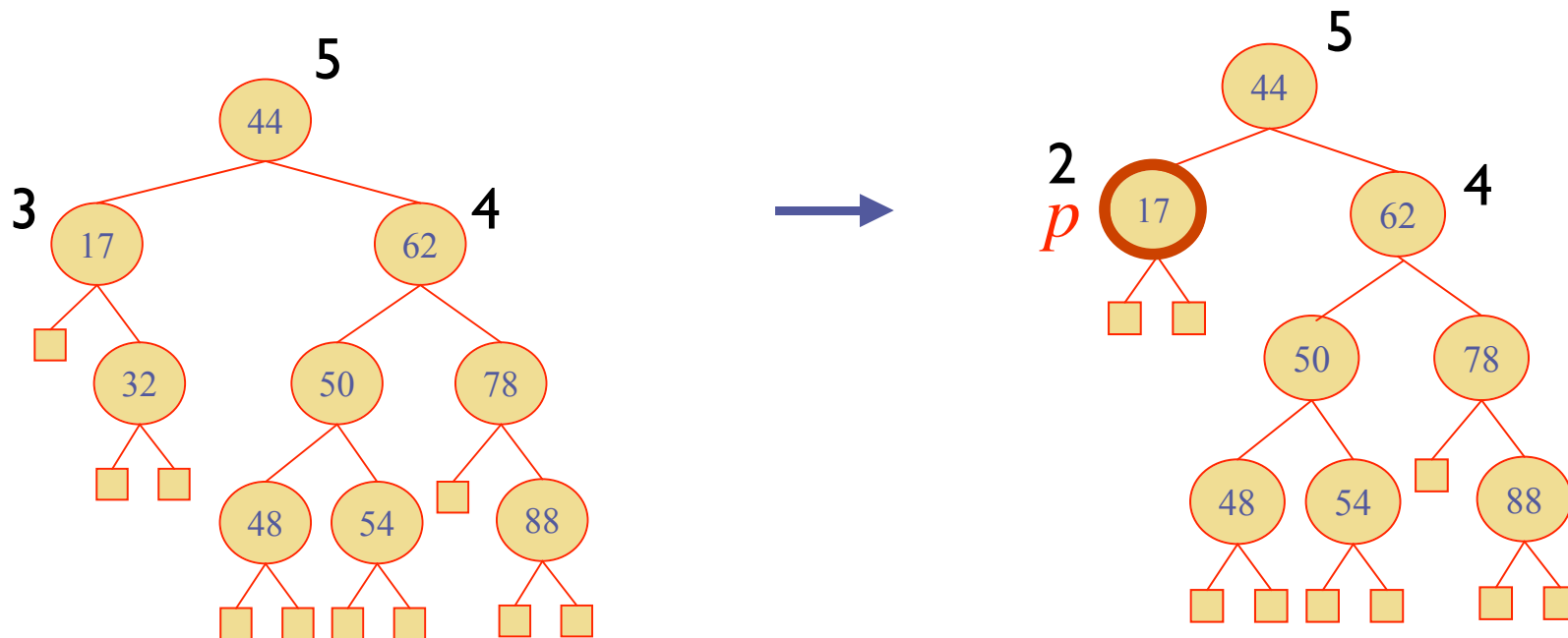
- **Exemple 2: Enlever(3)**

- On trouve le noeud interne  $y$  qui suit  $w$  lors d'un parcours symétrique de l'arbre et son fils gauche  $x$
- On enlève l'entrée dans  $w$  et on la remplace par l'entrée dans  $y$
- On enlève les noeuds  $y$  et  $x$



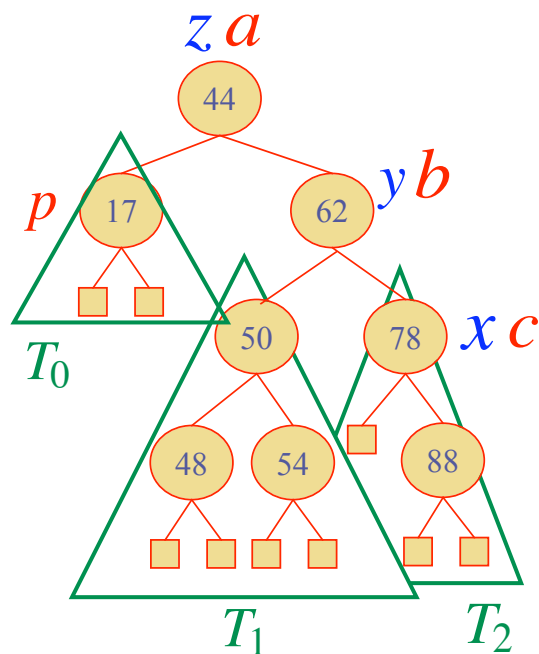
# Supprimer dans les arbres AVL

- Après la suppression d'un noeud, il est possible que son parent  $p$  cause un débilancement
- Exemple:** Enlever(32)

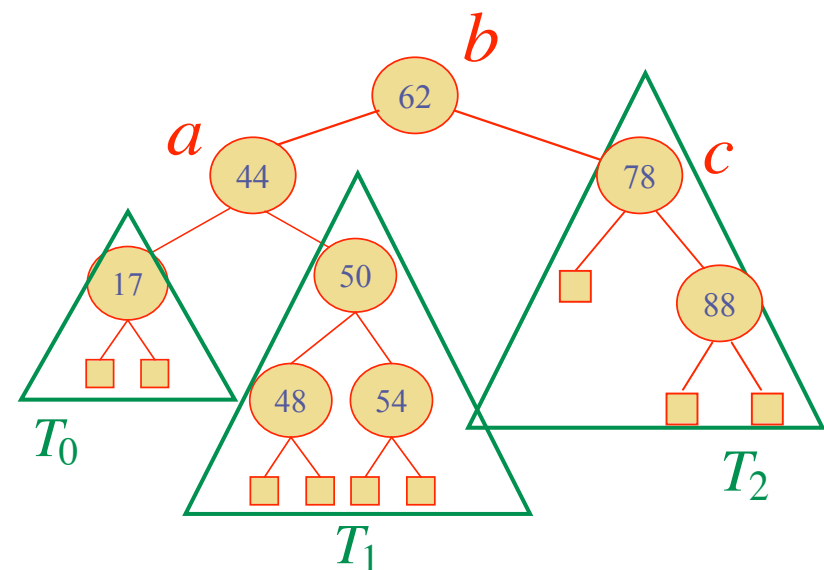


# Supprimer dans les arbres AVL (suite)

- Soit  $z$  le premier noeud non balancé sur le chemin allant de ce parent  $p$  à la racine de l'arbre. Soit  $y$  le fils de  $z$  de hauteur maximale et soit  $x$ , le fils de  $y$  de hauteur maximale
- Étant donné les noeuds internes  $x, y$  et  $z$ , on renomme par  $a$  le premier de ces sommets visités lors d'un parcours symétrique de l'arbre, par  $b$ , le deuxième et par  $c$ , le troisième.



$$T_0 \textcircled{a} (T_1 \textcircled{b} T_2)$$



$$(T_0 \textcircled{a} T_1) \textcircled{b} T_2$$

# Supprimer dans les arbres AVL (suite)

**Attention!!** Il est possible qu'après le rebalancement d'un noeud, un autre noeud (sur le chemin allant vers la racine) se retrouve déséquilibré

On continue donc à balancer l'arbre jusqu'à ce qu'on atteigne la racine

Exemple au tableau!!

# Performances pour les arbres AVL

- Rebalancer un noeud prend un temps  $O(1)$ , si on implémente l'arbre AVL à l'aide d'une structure chaînée
- L'algorithme de recherche prend un temps  $O(\log n)$ 
  - La hauteur de l'arbre est  $O(\log n)$  et aucune restructuration est nécessaire
- L'algorithme d'insertion prend un temps  $O(\log n)$ 
  - Chercher l'endroit où insérer prend un temps  $O(\log n)$
  - Trouver un noeud non balancé (si il y en a un) prend un temps  $O(\log n)$
  - Restructurer l'arbre prend un temps  $O(1)$
- L'algorithme de suppression prend un temps  $O(\log n)$ 
  - Chercher le noeud à enlever prend un temps  $O(\log n)$
  - On devra faire, au plus,  $O(\log n)$  restructuration
  - Chaque restructuration prend un temps  $O(1)$