Cours n° 8 : les arbres équilibrés et les tas

Christophe Gonzales

POLYTECH MARSEILLE
Aix*Marseille Université

HUGO3 — Algorithmique

Comment équilibrer un ABR

- **Ce que l'on aimerait :**
  - $A(n)$: sous-arbre de racine le nœud n
    $A_g(n)$ et $A_d(n)$: ses sous-arbres gauche et droit
  - $\forall n, |A_g(n)| = |A_d(n)|$: même nombre de nœuds à gauche et à droite
  - $\forall n, h(A_g(n)) = h(A_d(n))$: même hauteur des sous-arbres gauche et droit

- Propriétés précédentes difficiles à maintenir dans un ABR

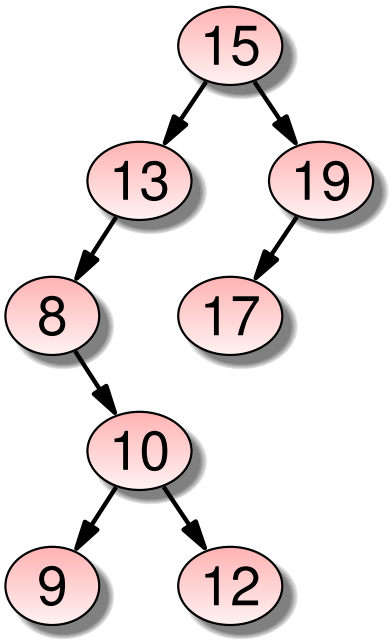
Version relachée

- $\forall n, |h(A_g(n)) - h(A_d(n))| < 2$: les 2 hauteurs sont les mêmes, à 1 près.

Arbres équilibrés

Arbre AVL [1962 - Adelson-Velsky et Landis]

Un arbre AVL est tel que **la différence des hauteurs des fils gauche et droit de tout noeud ne peut excéder 1**.

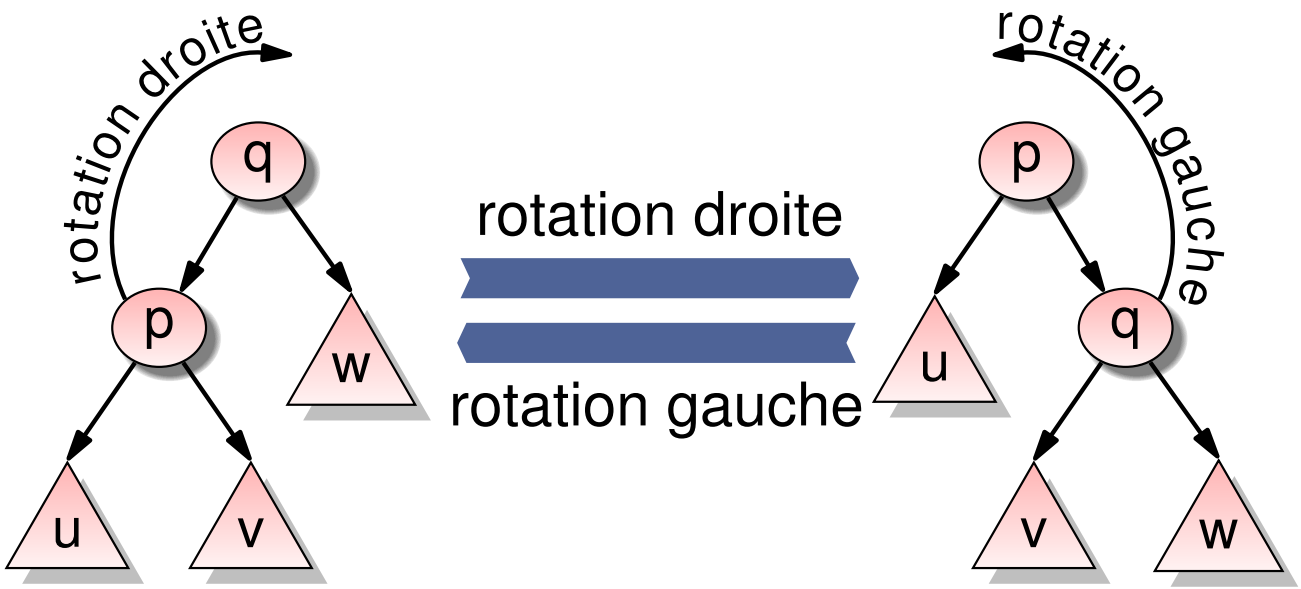


- Nœuds 15, 13 et 8 : violent la propriété
- Tous les autres nœuds : OK
- **Pour maintenir un arbre AVL, il faut garder en tout nœud cette différence de hauteur.**

Propriété des arbres AVL

La hauteur d'un arbre AVL est de l'ordre de $\log_2 n$.

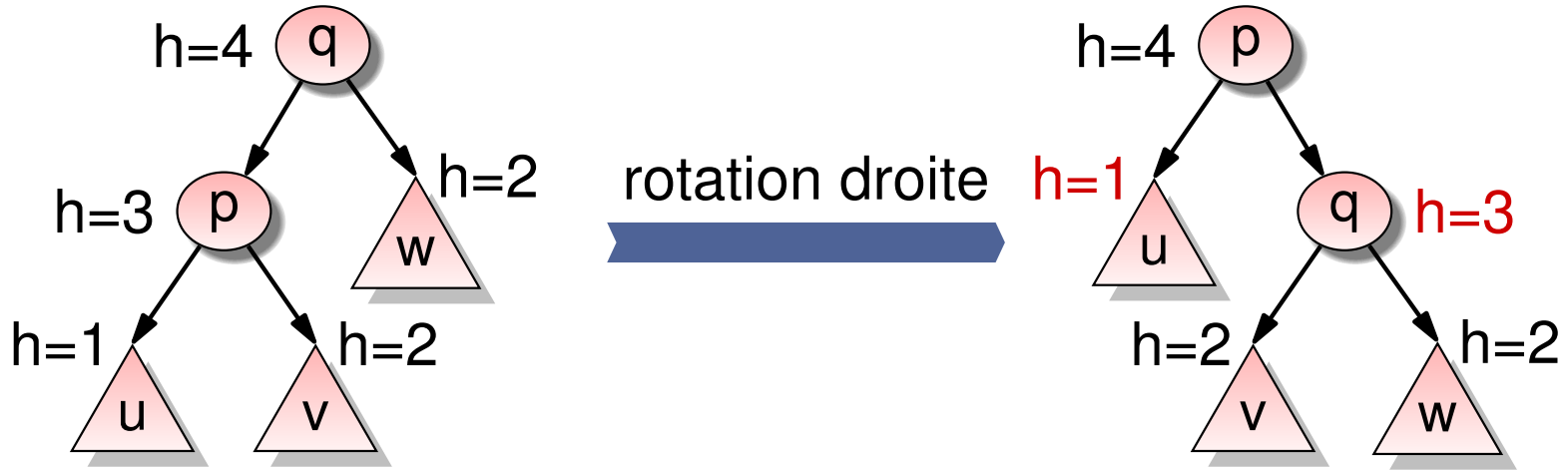
- **Les rotations d'un arbre pour le rééquilibrer :**



- **Remarque 1 :** Rotation $\Longrightarrow$ ABR transformé en ABR
- **Remarque 2 :** Rotation $\Longrightarrow$ conserve l'ordre infixe

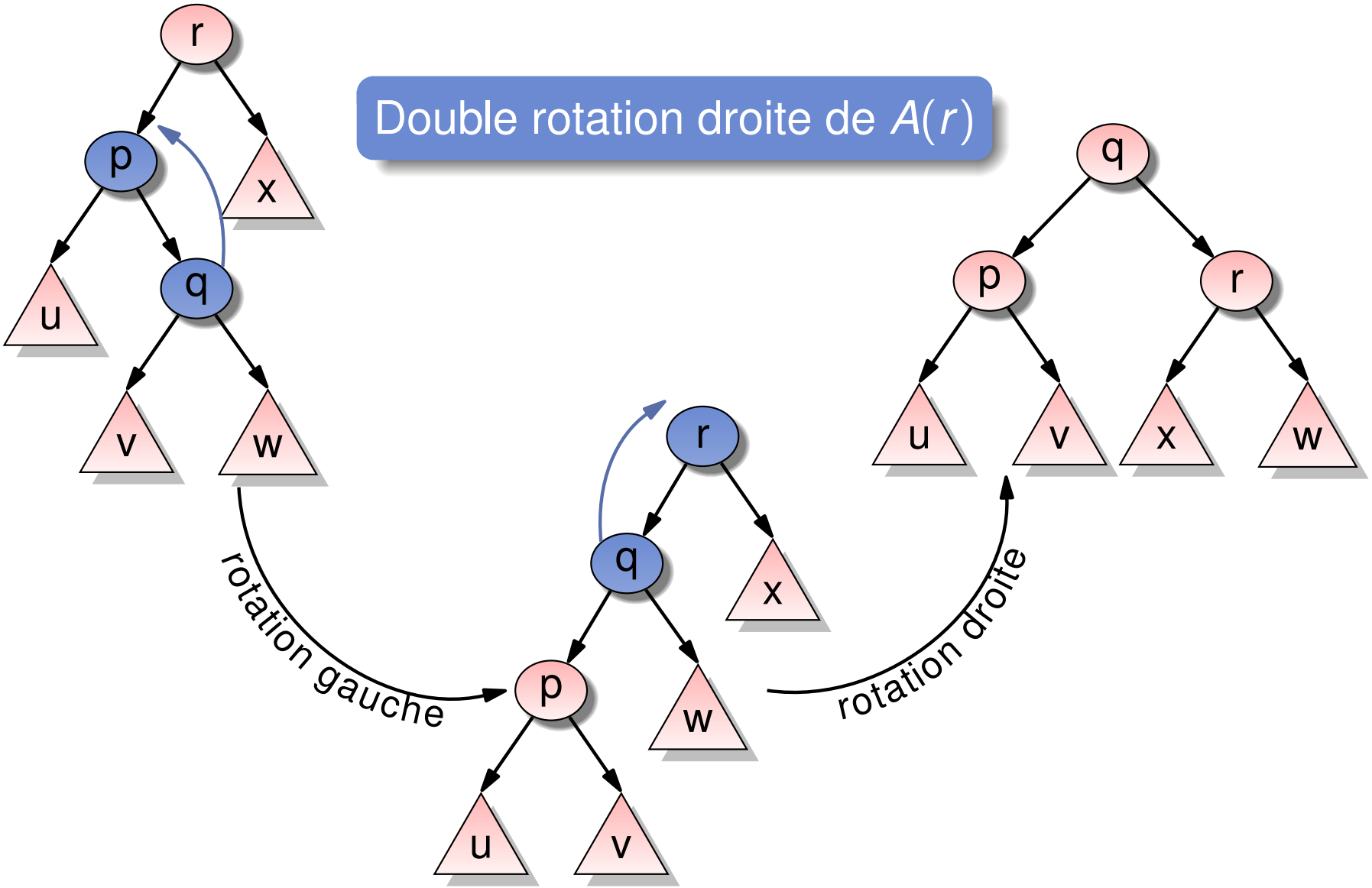
Arbre AVL et rotation « simple »

La propriété AVL n'est pas conservée par 1 rotation



$\Longrightarrow$ Rotations à effectuer avec précautions !

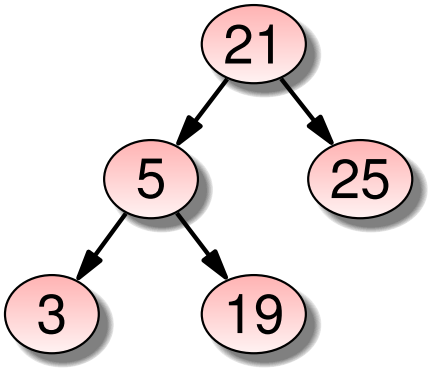
Rééquilibrage : doubles rotations



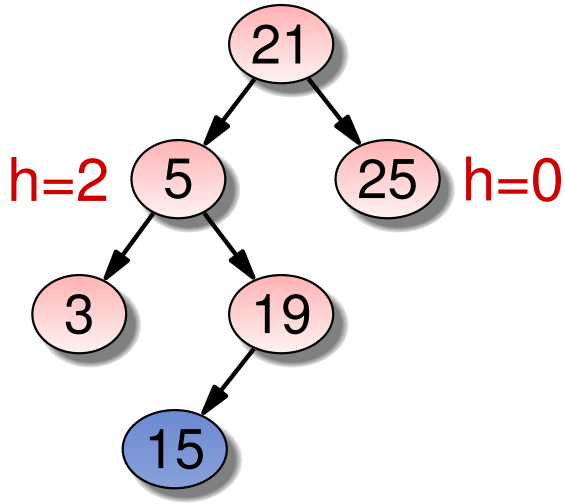
La double rotation gauche est définie similairement.

À quoi servent les doubles rotations

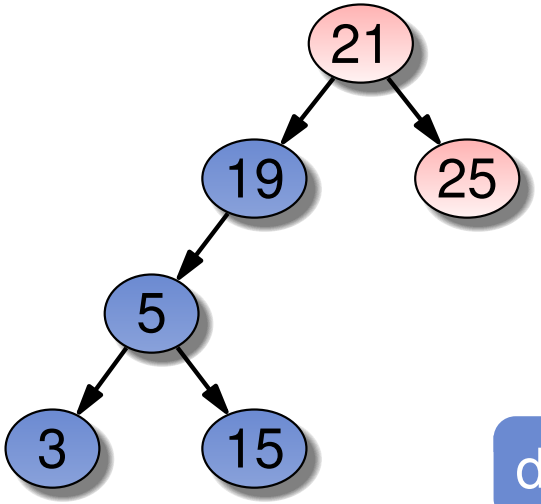
1. arbre AVL de départ :



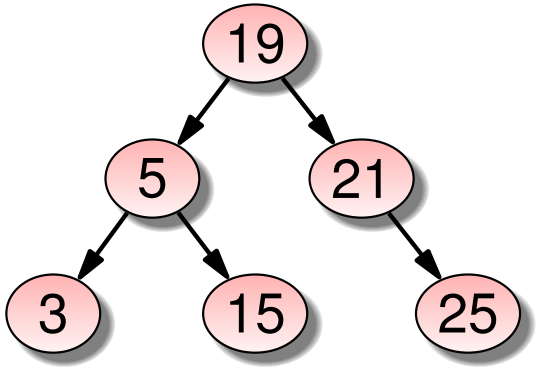
2. après insertion de 15 :



3. après rotation gauche de $A(5)$:



4. après rotation droite de $A(21)$:



double rotation droite de $A(21)$

Insertion/suppression d'un élément

Insertion

1. Insérer l'élément comme dans un ABR classique
2. Soit z le parent du nœud inséré
3. Si z n'est pas équilibré ($|h(A_g(z) - h(A_d(z)| = 2$) alors rééquilibrer l'arbre de racine z
4. Si z n'est pas la racine de l'ABR, remonter au parent de z et revenir en 3 avec ce nœud

Suppression

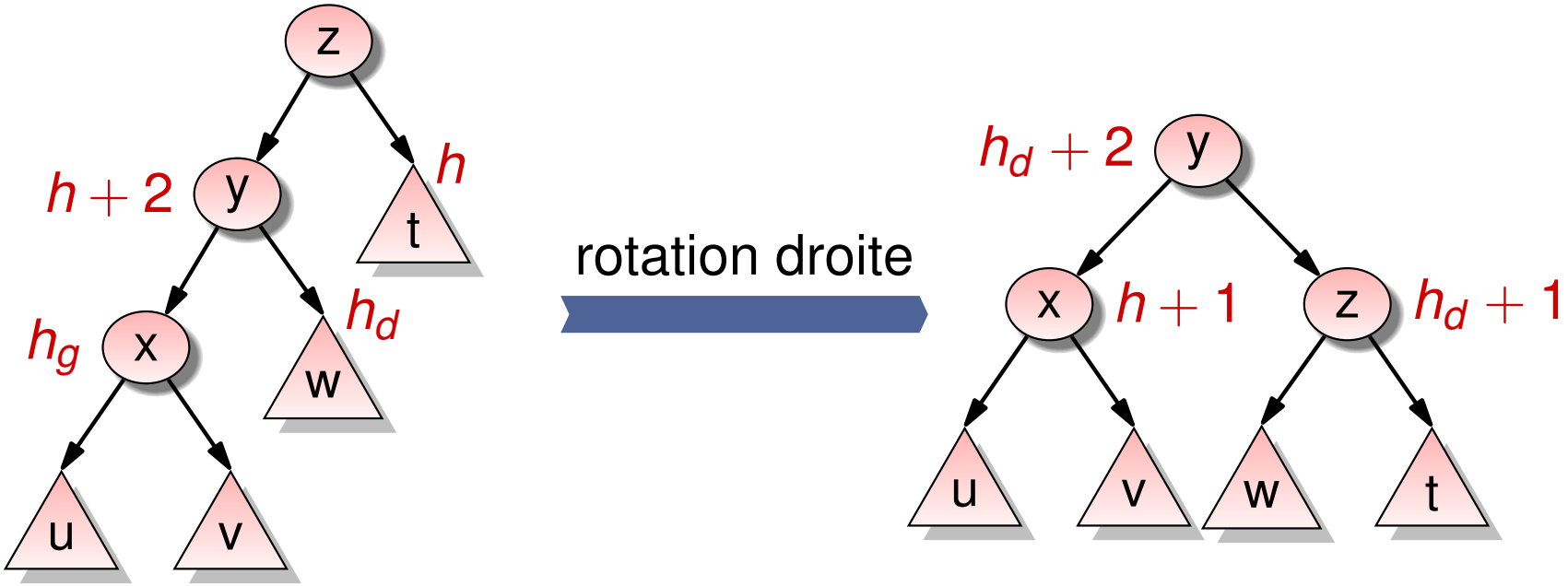
1. Supprimer l'élément comme dans un ABR classique
2. Soit z le parent du nœud physiquement supprimé de l'ABR
3. Si z n'est pas équilibré ($|h(A_g(z) - h(A_d(z)| = 2$) alors rééquilibrer l'arbre de racine z
4. Si z n'est pas la racine de l'ABR, remonter au parent de z et revenir en 3 avec ce nœud

Les 4 cas de rééquilibrage d'un arbre (1/4)

- 1er cas : $A(y)$ Arbre AVL mais pas $A(z)$ et $h_g \geq h_d$

  $\Longrightarrow h_g = h + 1$

  $\Longrightarrow h \leq h_d \leq h + 1$

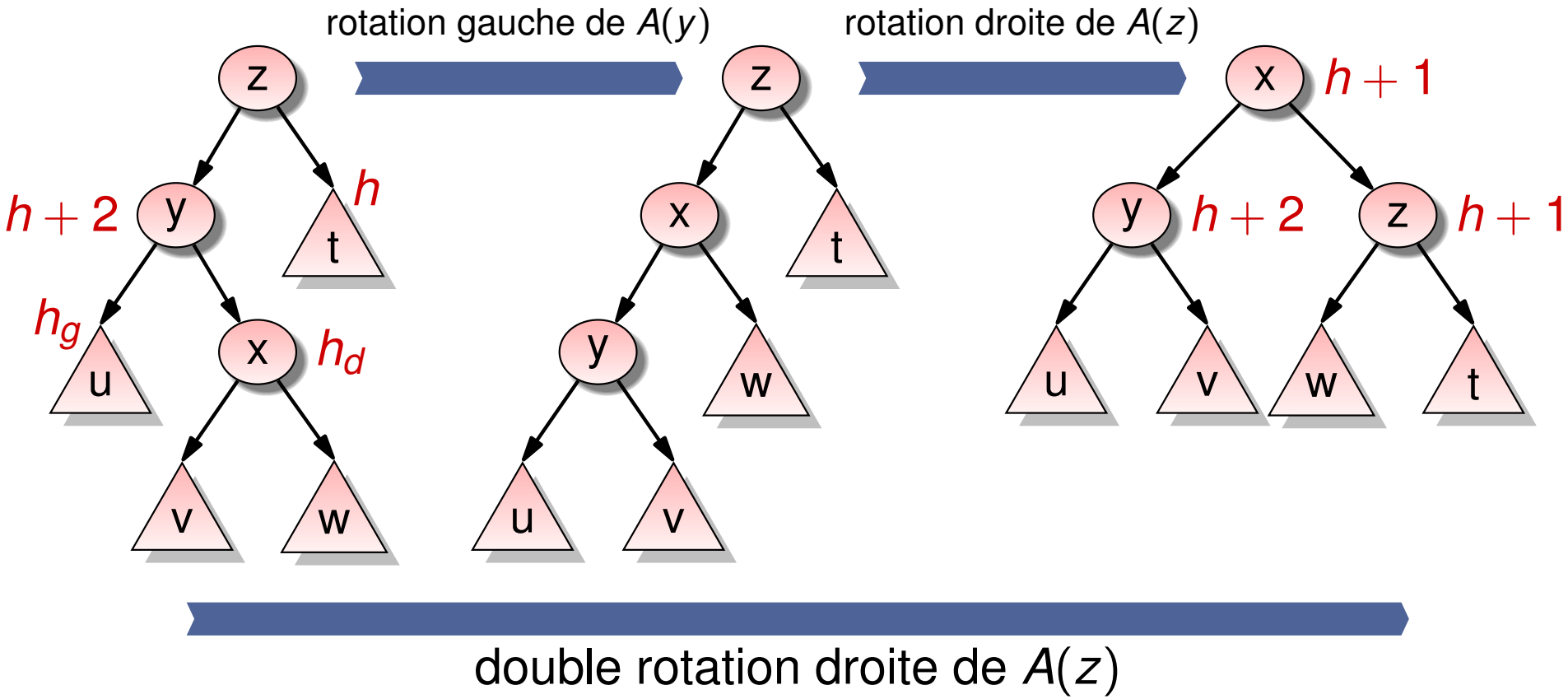


Les 4 cas de rééquilibrage d'un arbre (2/4)

- 2ème cas : $A(y)$ Arbre AVL mais pas $A(z)$ et $h_d > h_g$

  $\Longrightarrow h_d = h + 1$

  $\Longrightarrow h \leq h_g < h + 1 \Longrightarrow h_g = h$

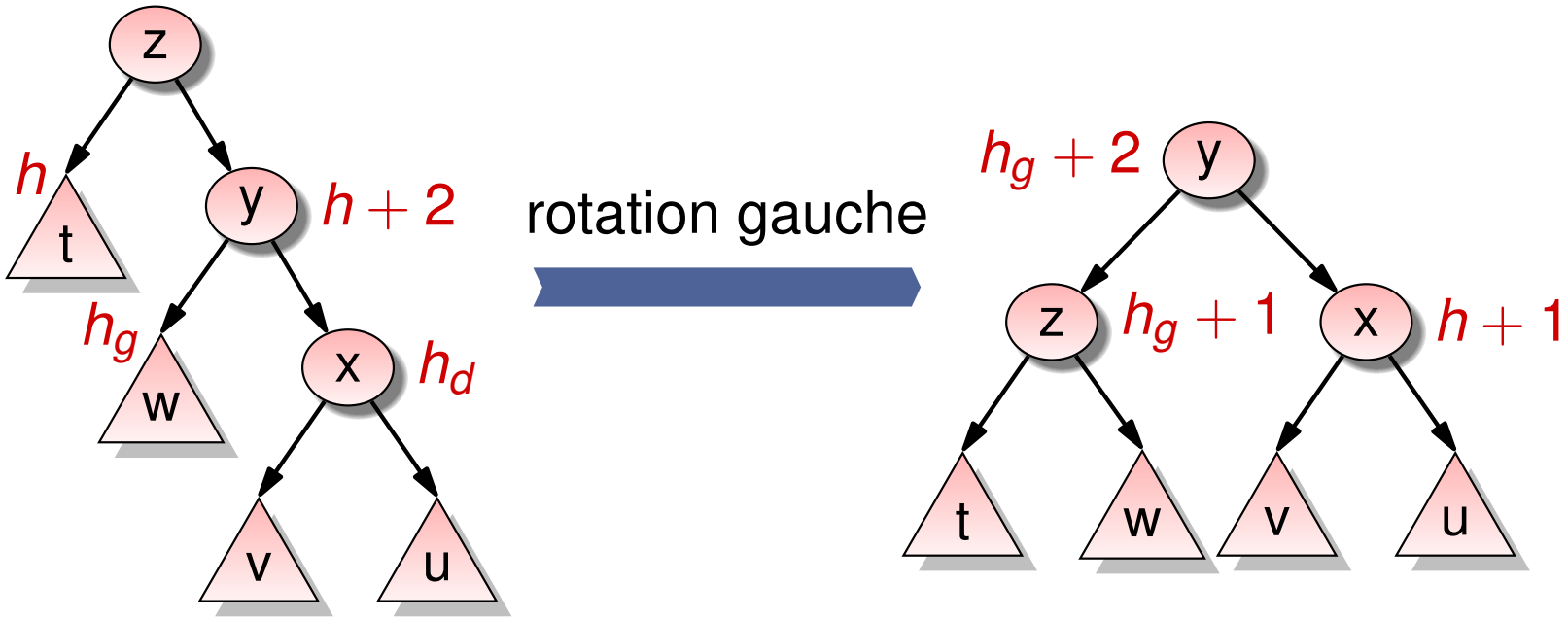


Les 4 cas de rééquilibrage d'un arbre (3/4)

- 3ème cas : $A(y)$ Arbre AVL mais pas $A(z)$ et $h_d \geq h_g$

  $\Longrightarrow h_d = h + 1$

  $\Longrightarrow h \leq h_g \leq h + 1$

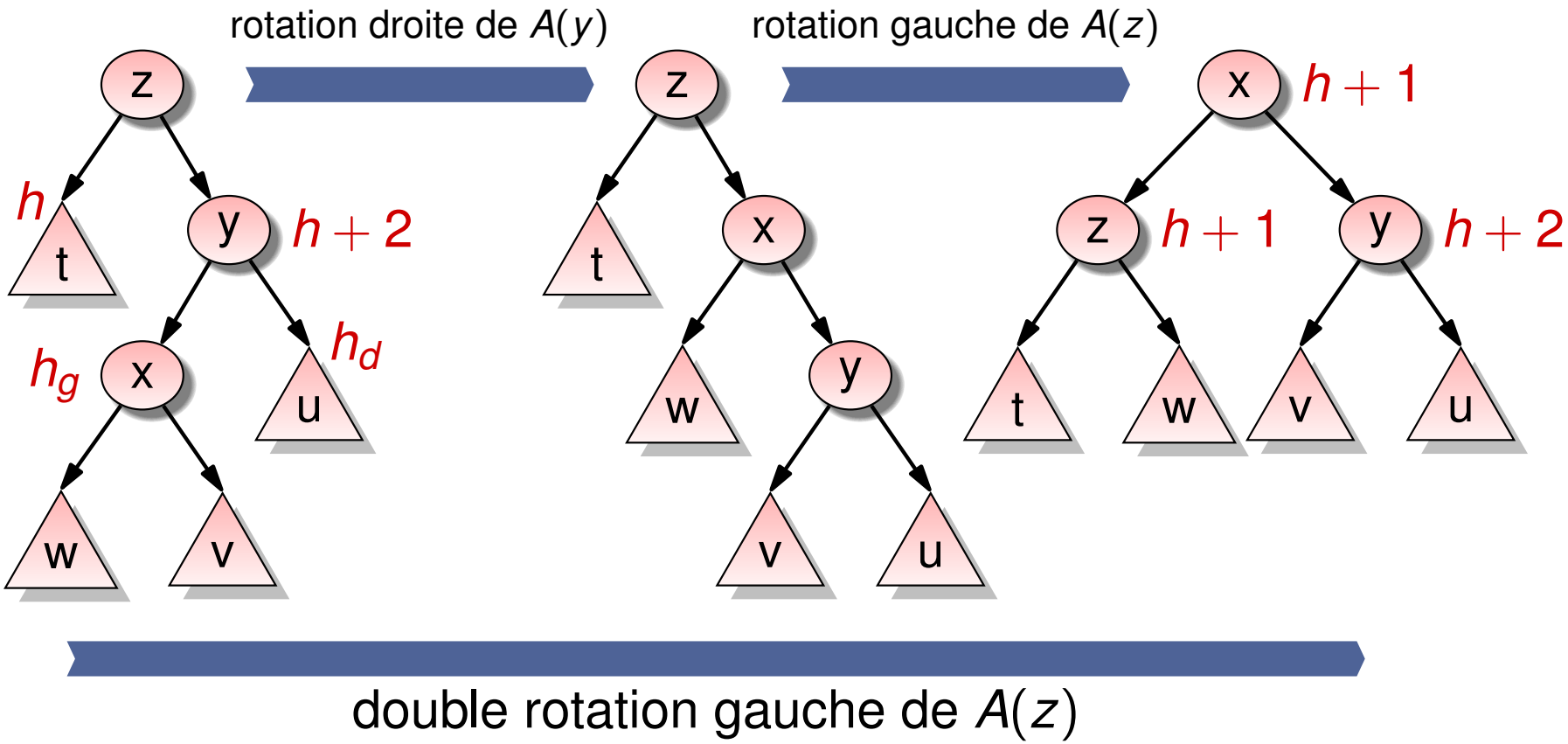


Les 4 cas de rééquilibrage d'un arbre (4/4)

- 4ème cas : $A(y)$ Arbre AVL mais pas $A(z)$ et $h_g > h_d$

  $\Longrightarrow h_g = h + 1$

  $\Longrightarrow h \leq h_d < h + 1 \Longrightarrow h_d = h$



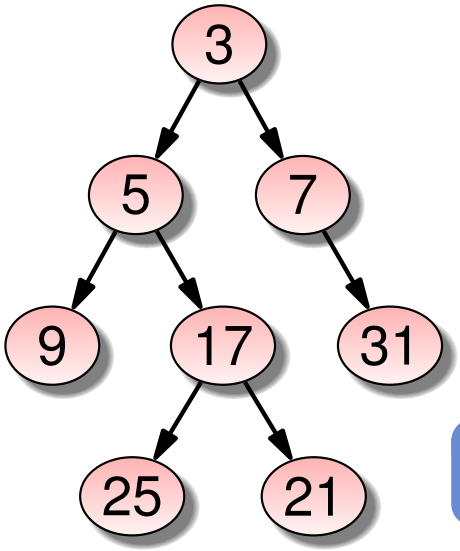
2 Les tas

Les tas

Définition d'un tas

- Tas = arbre binaire
- Il est ***ordonné*** :

  Tous les nœuds, autre que la racine, ont une valeur plus grande que leur père.



- Structure de tas faible :
- On sait uniquement que :
  - le min est à la racine
  - le max est sur une feuille

Dans un tas, on ne s'intéresse qu'au min

$\Longrightarrow$ peut servir pour des tris (heapsort)

Opérations sur les tas

Seulement 3 opérations dans un tas :

- Rajouter des éléments dans le tas
- Récupérer le min du tas
- Supprimer le min du tas (racine de l'arbre)

Rajouter un élément dans un tas

- Essayer d'équilibrer l'arbre
  $\Longrightarrow$ garder le tas le plus proche d'un arbre complet

ajouter toujours dans le fils gauche puis échanger les deux fils $\Longrightarrow$ garantit une hauteur proportionnelle à $\log_2 n$

```
bintree_t insert (bintree_t tree, int elt) {
  if (tree == NULL) return new_node (elt);

  int min_x, max_x;
  if (tree->label < elt) {
    min_x = tree->label;
    max_x = elt;
  }
  else {
    min_x = elt;
    max_x = tree->label;
  }

  node_t* fils_gauche = tree->fils_gauche;
  tree->fils_gauche = insert (tree->fils_droit, max_x);
  tree->fils_droit = fils_gauche;
  tree->label = min_x;
  return tree;
}
```

Supprimer le min d'un tas

```
bintree_t suppr_min (bintree_t tree) {
  if (tree == NULL) throw std::exception ();

  node_t* node;
  // si la racine a au plus un fils
  if (tree->fils_gauche == NULL) {
    node = tree->fils_droit;
  }
  else if (tree->fils_droit == NULL) {
    node = tree->fils_gauche;
  }

  // si la racine a deux fils
  else if (tree->fils_gauche->label < tree->fils_droit->label) {
    node_t* node = new_node (tree->fils_gauche->label);
    node->fils_gauche = suppr_min (tree->fils_gauche);
    node->fils_droit  = tree->fils_droit;
  }
  else {
    node_t* node = new_node (tree->fils_droit->label);
    node->fils_droit = suppr_min (tree->fils_droit);
    node->fils_gauche  = tree->fils_gauche;
  }

  free(tree);
  return node;
}
```

Synthèse des arbres AVL et des tas

Opération	arbre AVL	tas
Insertion d'un élément	O(log(n))	O(log(n))
Suppression d'un élément	O(log(n))	O(log(n))
Accéder au premier élément (racine)	O(1)	O(1)
Accéder au dernier élément	O(log(n))	O(log(n))
Accéder à l'élément le plus petit	O(log(n))	O(1)
Accéder à l'élément le plus grand	O(log(n))	O(n)
Chercher si un élément existe	O(log(n))	O(n)