UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

INF6
ALGORITHMIQUE AVANCÉE

Manuele Kirsch Pinheiro
Carine Souveyet

Contenu prévisionnel

- Piles et files
- Listes
- Récursivité
  - Récursivité dans le calcul
  - Récursivité structurelle
- Arbres binaires
  - Parcours en profondeur et en largeur
- Généralisation de la notion d'arbre
  - Insertion et suppression de nœuds
- **Arbre de recherche**
  - **Recherche & manipulation**
  - **Rééquilibrage**

UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

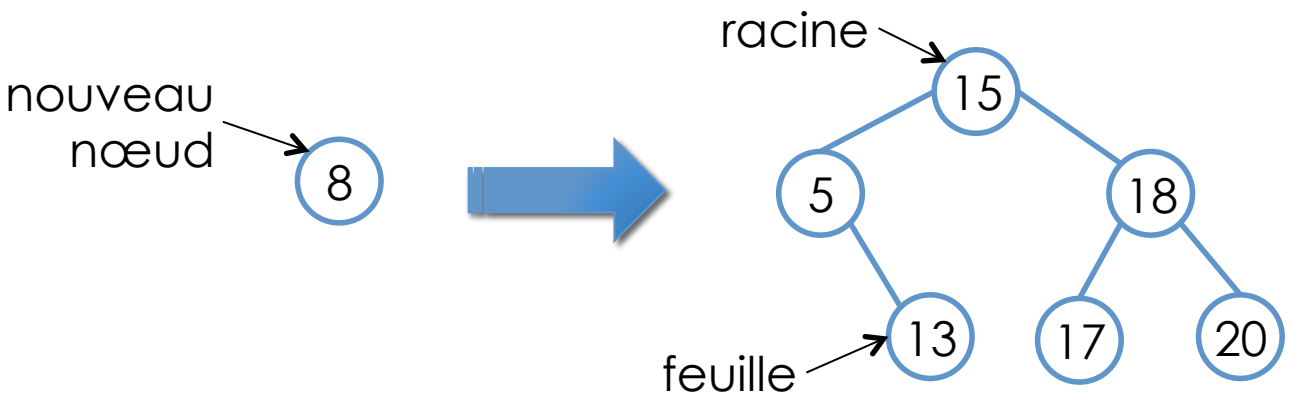
ARBRES DE RECHERCHE

Manipulation

Insertion ABR

- **Insertion**
  - L'insertion des nouveaux nœuds dans les Arbres Binaires de Recherche se réalise essentiellement dans les **feuilles**
  - On doit retrouver la première feuille où on peut attacher le nouveau nœud tout en respectant la relation d'ordre

clé(gauche) < clé(racine) < clé (droite)

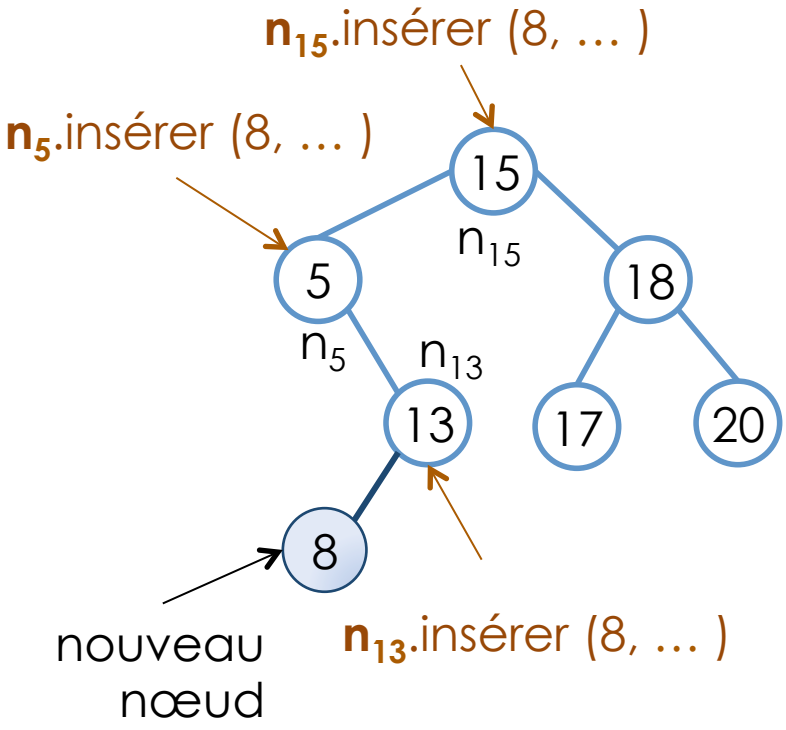


Insertion ABR

- Insertion → on distingue trois cas :
  - **clé (nouveau) < clé (nœud)**
    - l'insertion se fait dans le sous arbre gauche
    - Si celui-ci existe déjà, on délègue au fils gauche l'insertion
      - **this.left.insertion (clé, valeur)**
    - Sinon, le nouveau nœud devient le sous arbre gauche
      - **this.left = new ABR (clé, valeur)**
  - **clé (nouveau) > clé (nœud)**
    - l'insertion se fait dans le sous arbre droit
    - Si celui-ci existe déjà, on délègue au fils droit l'insertion
      - **this.right.insertion (clé, valeur)**
    - Sinon, le nouveau nœud devient le sous arbre droit
      - this.right = new ABR (clé, valeur)
  - **clé (nouveau) = clé (nœud)**
    - c'est lors de l'insertion qu'on va gérer le cas des clés dupliquées
    - normalement, celles-ci sont interdites dans un ABR
    - **Exception clé existante**

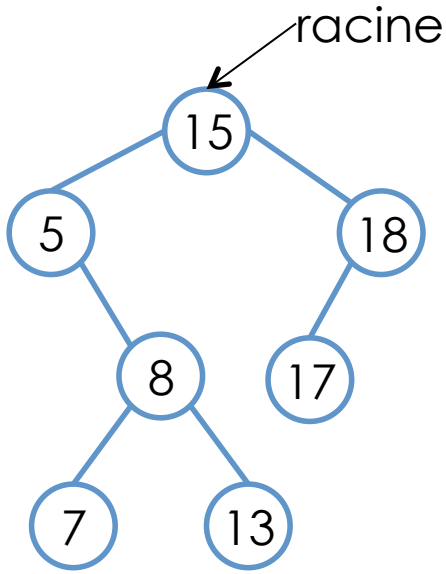
```
Insérer ( K key , V value )

Si estVide() Alors
    this.key = key
    this.value = value
Sinon
    Si key < this.key Alors
        Si this.left ≠ null Alors
            this.left.insertion (key, value)
        sinon
            this.left = nouveau ABR (key, value)
        fin Si
    Sinon Si key > this.key Alors
        Si this.right ≠ null Alors
            this.right.insertion (key, value)
        Sinon
            this.right = nouveau ABR (key, value)
        Fin si
     Sinon  Lancer exception Clé existante
   fin Si
fin
```

- Insertion à partir de la racine (***this***)
  - attention au cas d'arbre vide (this.key == null)



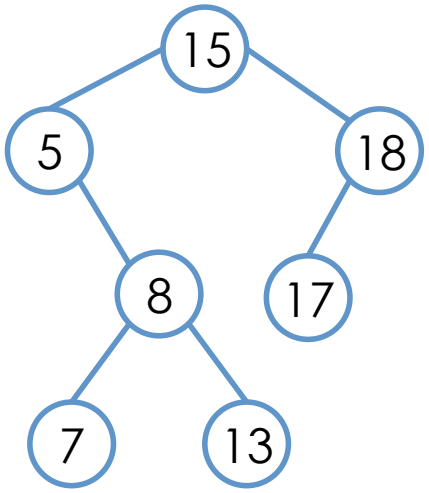
Suppression

- **Suppression** d'un nœud à partir de la **racine** d'un arbre ABR
- Deux étapes :
  - **Suppression (clé)**
    - on va d'abord trouver le nœud avec la clé à supprimer
    - parcours préfixe
    - dans les structures dans parent, permet de « retracer » les parents
  - **Suppression ()**
    - on supprime un nœud choisi
    - traitement des cas possibles



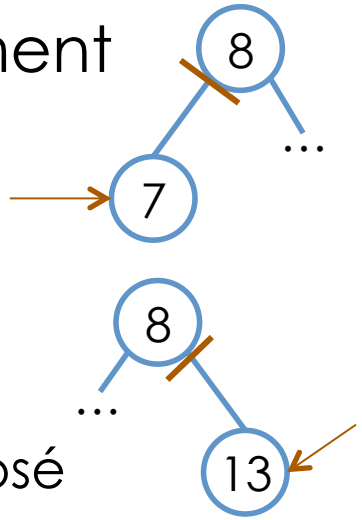
Suppression

- La **suppression** d'un nœud doit s'assurer que la **relation d'ordre** est **maintenue**
- **Trois cas principaux** doivent être considérés, avec, dans chaque cas, **la possibilité d'être ou non racine**
  - **suppression d'une feuille**
    - exemple : suppression de 7, 13 ou 17
  - **suppression d'un nœud interne avec 1 fils**
    - exemple : suppression de 5 ou de 18
  - **suppression d'un nœud interne avec 2 fils**
    - exemple : suppression de 8 ou de 15



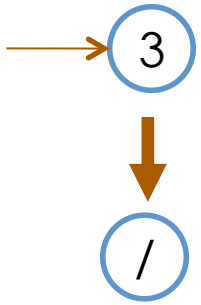
Suppression

- **Suppression d'une feuille**
  - cas le plus simple, puisqu'il s'agit uniquement de mettre à jour le père
  - il suffit de changer la sous arbre droit ou gauche du nœud parent
  - cas particulier : racine
    - lors qu'on supprime la racine d'un arbre composé de ce seul nœud, l'arbre devient un arbre vide



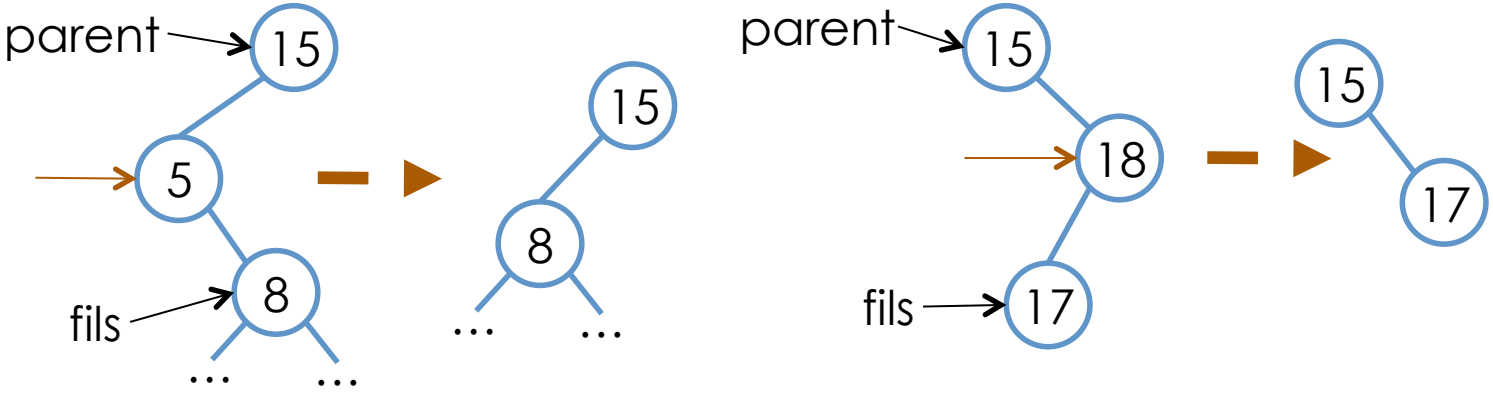

```
Si  estFeuille() Alors
    Si parent != null Alors
        Si parent.left == this Alors
            parent.left = null
        Sinon parent.right = null
        fin si
…
```

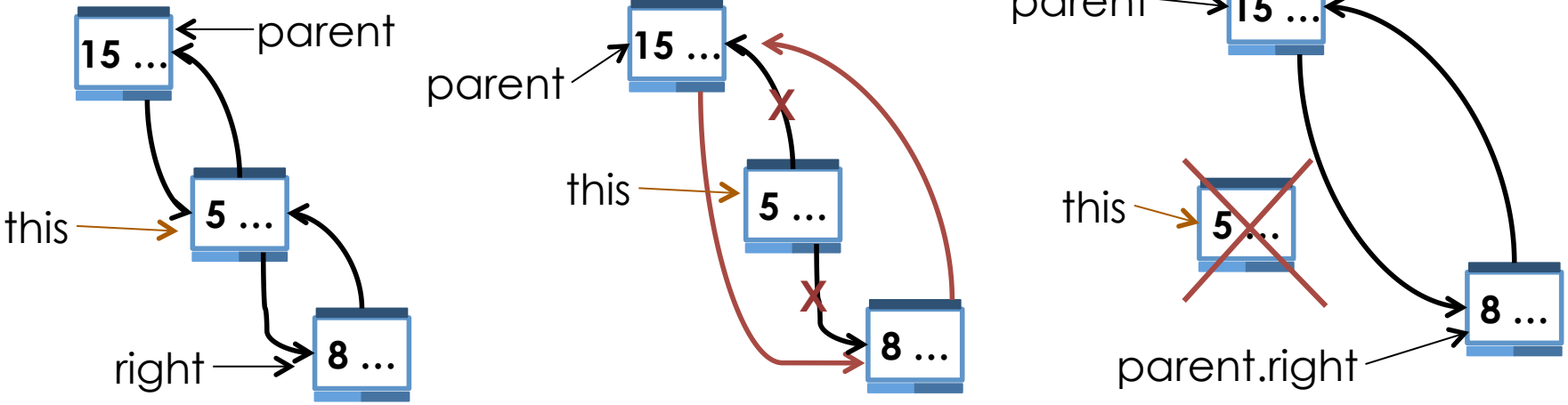
```
…
    Sinon
        key = null
        value = null
    fin Si
fin Si
…
```



Suppression

- **Suppression d'un nœud interne avec 1 fils**
  - Le fils prend alors la place du nœud supprimé

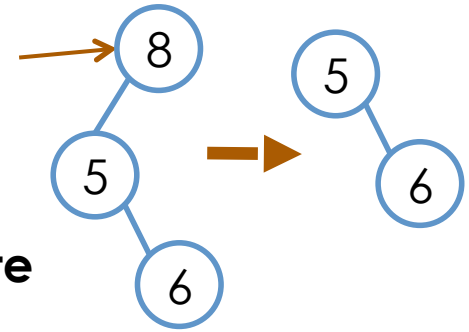


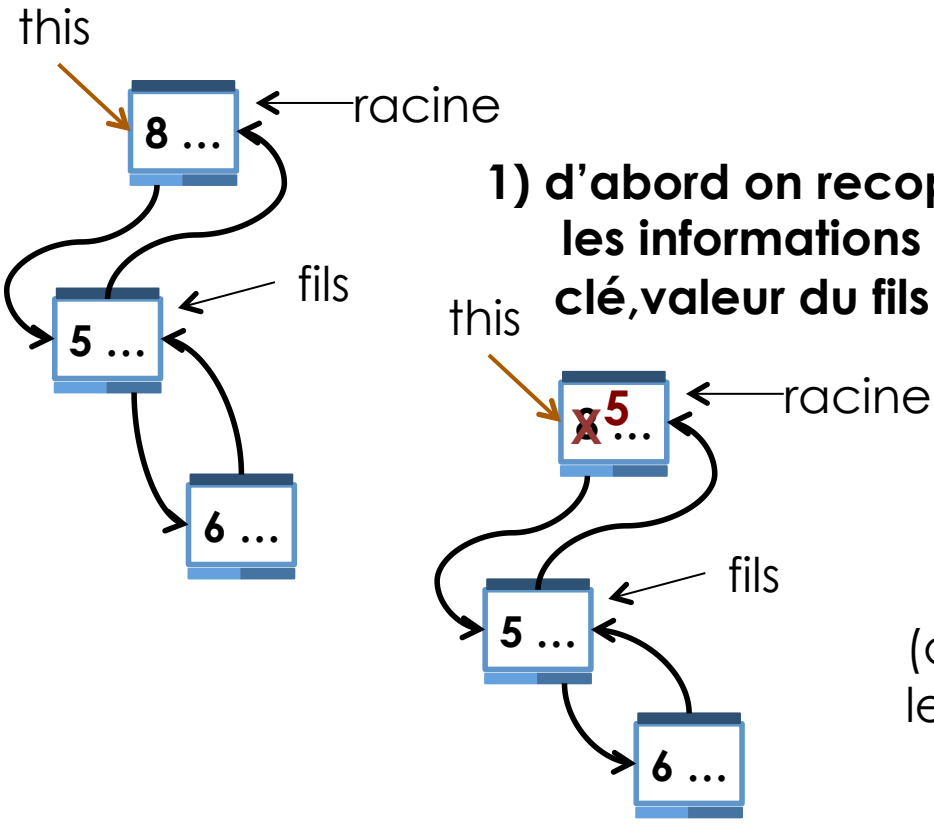


UNIVERSITÉ PARIS 1 PANTHÉON SORBONNE

Suppression

- **Suppression d'un nœud interne avec 1 fils**
  - Problème : lorsqu'on supprime la racine
  - Solution : transfert de contenu





Suppression

- **Suppression d'un nœud interne avec 1 fils**
  - on peut généraliser ce traitement…

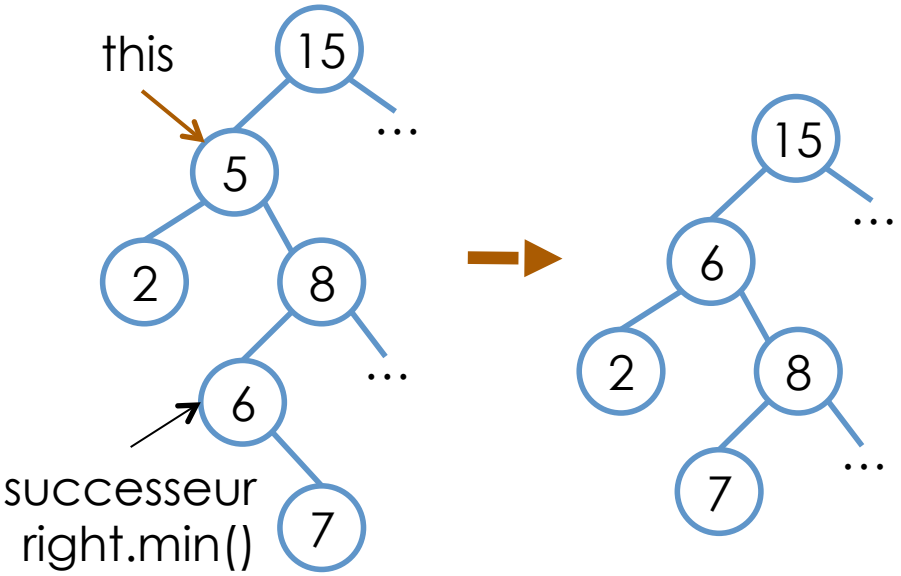
```
RemplacePar (ABR fils)
 Si fils != null Alors
     key = fils.key
     value = fils.value
     right = fils.right
     left = fils.left
     Si right != null Alors
        right.parent = this
    fin si
    Si left != null Alors
        left.parent = this
    fin si
    fils.parent = null
    fils.right = null
    fils.left = null
  fin Si
fin
```

```
…

 Si left != null ET right == null  Alors
     this.remplacePar (left)
 Sinon Si left == null ET right != null  Alors
     this.remplacePar (right)

…
```

UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

Suppression

- **Suppression d'un nœud interne avec 2 fils**
  - lorsque le nœud à supprimer contient 2 fils (gauche et droit), c'est son successeur qui doit prendre sa place.
    - successeur ➔ min (droit)
  - en remplaçant le nœud par son successeur, la relation d'ordre est garantie

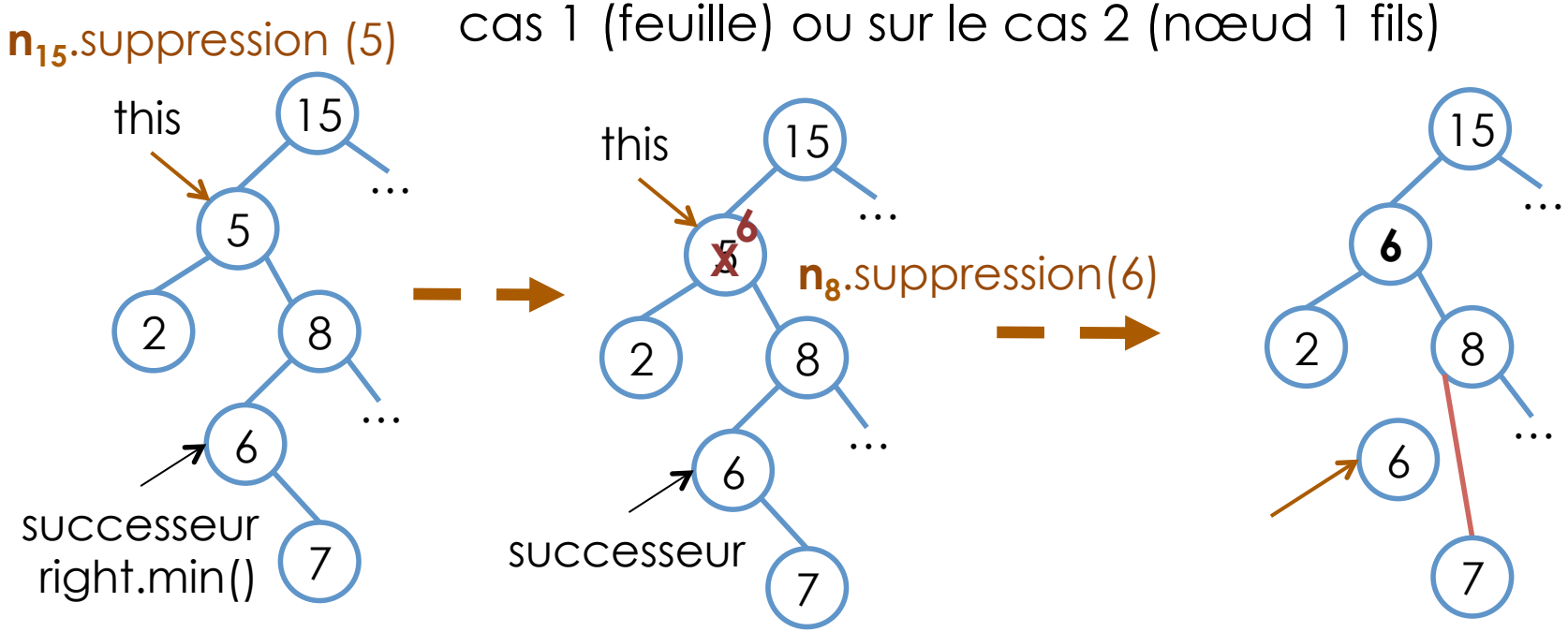


- le successeur de 5 est forcément supérieur à tous les nœuds à gauche de 5 (car il est à droite) et il est inférieur à tous les autres nœuds à la droite de 5 (car il est le minimum du sous arbre droit)

UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

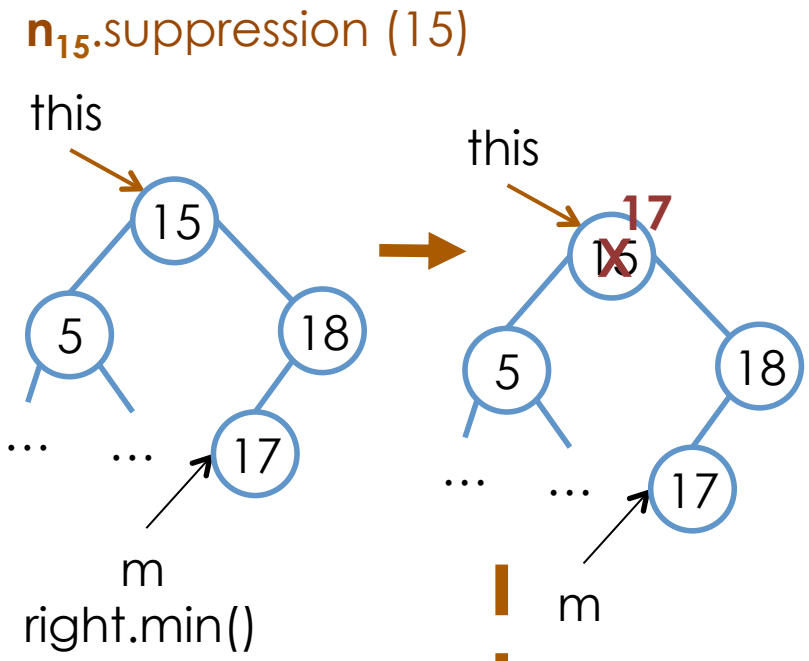
Suppression

- **Suppression d'un nœud interne avec 2 fils**
  - le successeur doit donc prendre la place du nœud à supprimer
  - le plus simple est de transférer le contenu (clé, valeur) du successeur puis de le supprimer à son tour
    - la suppression du successeur tombe forcément sur le cas 1 (feuille) ou sur le cas 2 (nœud 1 fils)

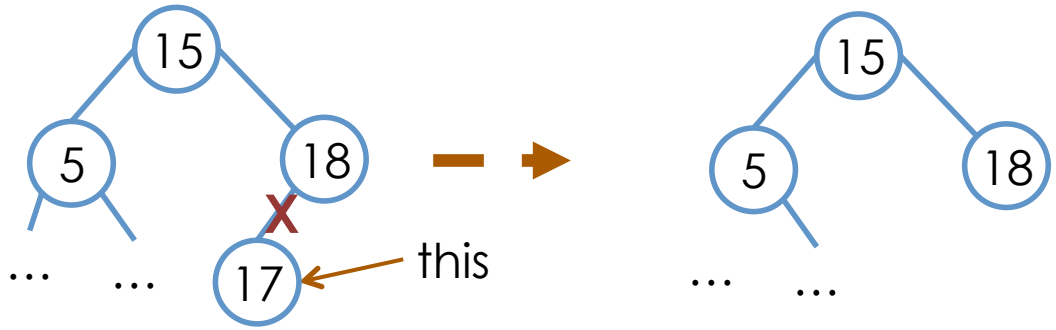


Suppression

- ce procédé va aussi s'appliquer à la racine




```
…
Sinon //aucun cas précédent

  ABR m = this.right ()
  this.key = m.key
  this.value = m.value
  right.supprimer (m.key)
     /* ou simplement m.supprimer() si la
   structure contient le pointeur parent */
…
```



Suppression

- Vue d'ensemble

```
Supprimer ()

Si estFeuille() Alors
    … // cas 1 : feuille
Sinon  Si left != null ET right == null  Alors
        … // cas 2 : 1 fils gauche
Sinon Si left == null ET right != null  Alors
      … // cas 2 : 1 fils droit
Sinon
      … // cas 3 : 2 fils
fin si
fin
```

un retour (boolean ou le nœud supprimé) est aussi possible

```
Supprimer ( K clé )

ABR n = this.search(clé)
Si n != null Alors
    n.supprimer()
Sinon
    lancer exception key not found
 fin si
fin
```

Suppression

- Problème : structure de données sans pointeur parent
  - un seul méthode publique
    - public void supprimer (K clé)
  - on va se servir de la récursivité pour retracer l'information sur le parent
    - protected void supprimer (K clé, **ABR parent**)
  - puis on supprimer le nœud (avec les 3 cas à traiter)
    - protected void supprimer (**ABR parent**)

↳ le parent est passé en **paramètre**

Suppression

- Problème : structure de données sans pointeur parent

Supprimer (K clé)

```
Si estVide ()
    lancer exception …
Sinon
    this.supprimer ( clé, null )
fin si
fin
```

on commence par la racine
parent = null

Supprimer (K clé , ABR parent)

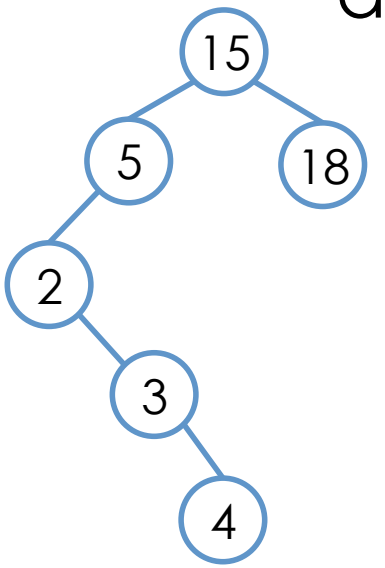
```
Si clé < this.key Alors
   Si this.left != null Alors
      this.left.supprimer ( clé, this )
   Sinon lancer exception …
Sinon Si this.key < clé Alors
   Si this.right != null Alors
      this.right.supprimer ( clé, this )
   Sinon lancer exception
   fin si
Sinon
   this.supprimer ( parent )
Fin si
fin
```

si le nœud à supprimer est dans un de mes sous arbre, le parent c'est moi : parent = this

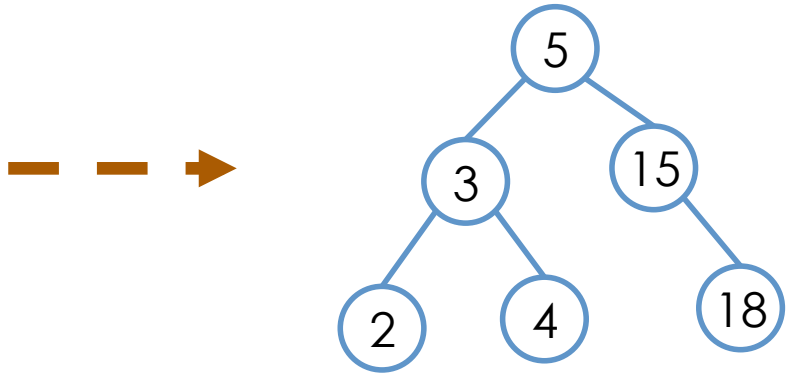
je suis le nœud à supprimer

Rééquilibrage

- L'insertion et la suppression d'éléments d'un ABR peut le « déséquilibrer »
  - Ce déséquilibre conduit à une perte de performance
    - Afin d'éviter cela, l'arbre doit être rééquilibré



exemple d'arbre ABR déséquilibré

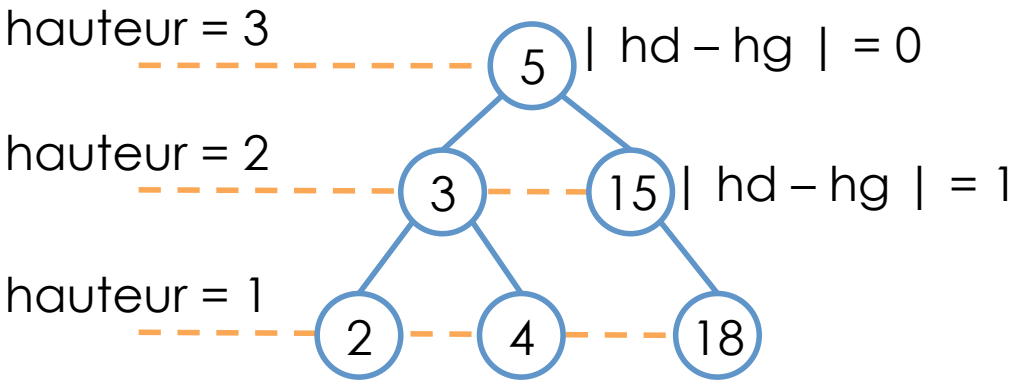


ABR rééquilibré

Rééquilibrage : Arbres AVL

- **Arbres AVL**
  - les arbres AVL (Adelson-Velskii et Landis, 1962) représentent un forme d'arbre équilibré
  - la différence d'hauteur entre les sous arbre droit et gauche n'excède jamais 1

| Hauteur (droit) – Hauteur (gauche) | ≤ 1

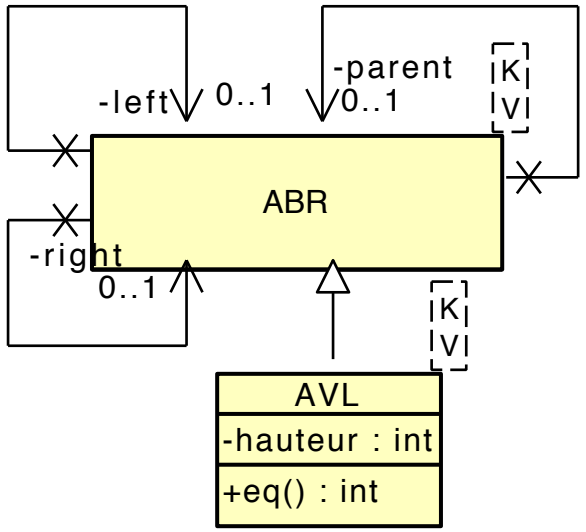


L'intérêt des arbres AVL repose sur la **recherche**, car en limitant la **profondeur de l'arbre**, on garanti une meilleure **performance de recherche**.

UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

Rééquilibrage : Arbres AVL

- **Hauteur et facteur d'équilibrage**
  - Afin de déterminer si l'arbre est équilibré, les arbres AVL doivent connaître leur **hauteur**
  - **Facteur d'équilibrage (eq)** permet de déterminer si un rééquilibrage est nécessaire



Soit *s* un arbre *AVL*, et *g* et *d* ses sous arbres *gauche* et *droit*

hauteur (s) = 1 + Max (hauteur (g), hauteur (d))

eq (s) = hauteur (d) – hauteur (g)

- Le facteur d'équilibrage d'un arbre AVL doit être entre **-1 et 1**, sinon l'arbre est déséquilibré

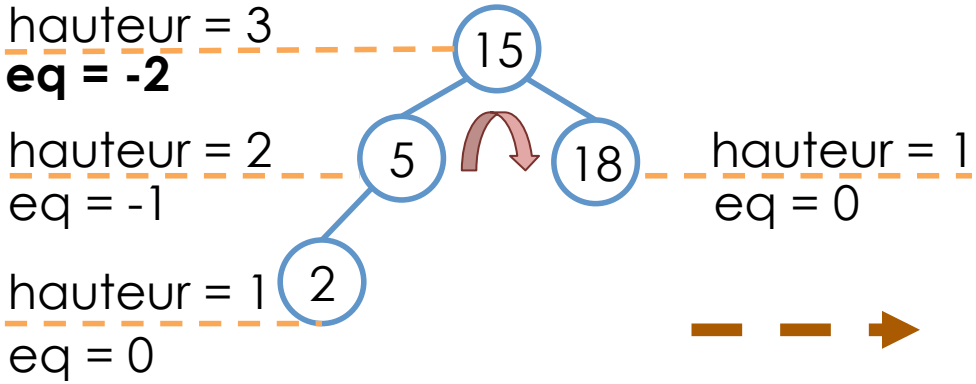
Si **eq(s) ⊆ [-1 , 0 , +1]** → **arbre équilibré**

Sinon si **eq(s) ≤ -2** ou **eq(s) ≥ +2** → **rééquilibrage** nécessaire

UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

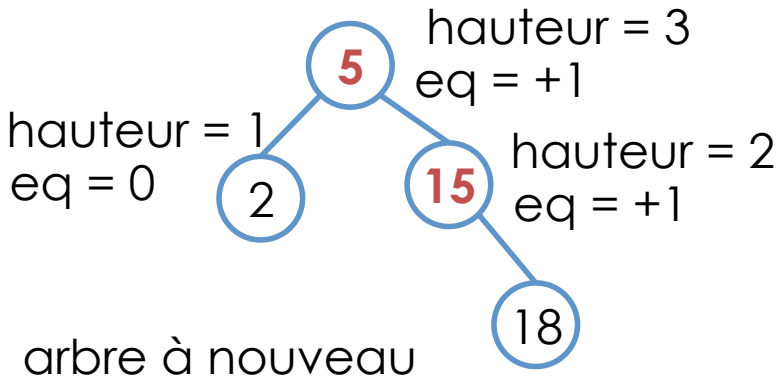
Rééquilibrage : Arbres AVL

- On rééquilibre un arbre AVL par le moyen de rotations
  - rotations simples : **rotation à gauche**, **rotation à droite**
  - **rotations doubles**



déséquilibre côté gauche

hauteur(d) – hauteur(g) = +1
eq ⊆ [-1, 0, +1]

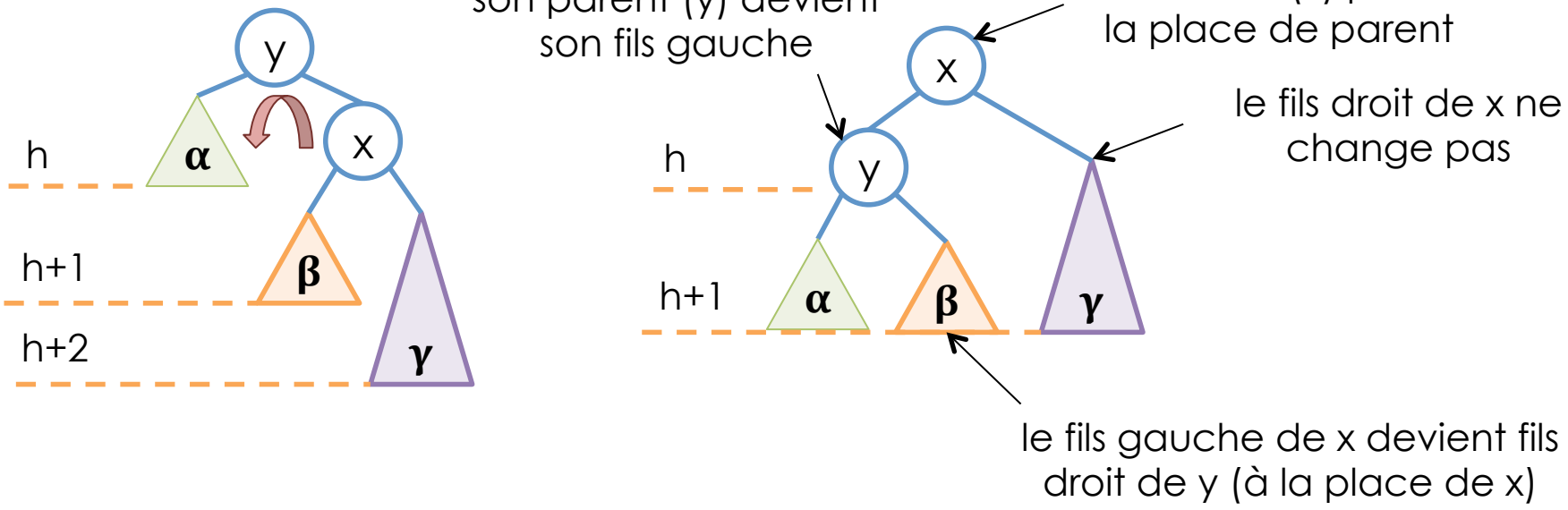


arbre à nouveau déséquilibré

UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

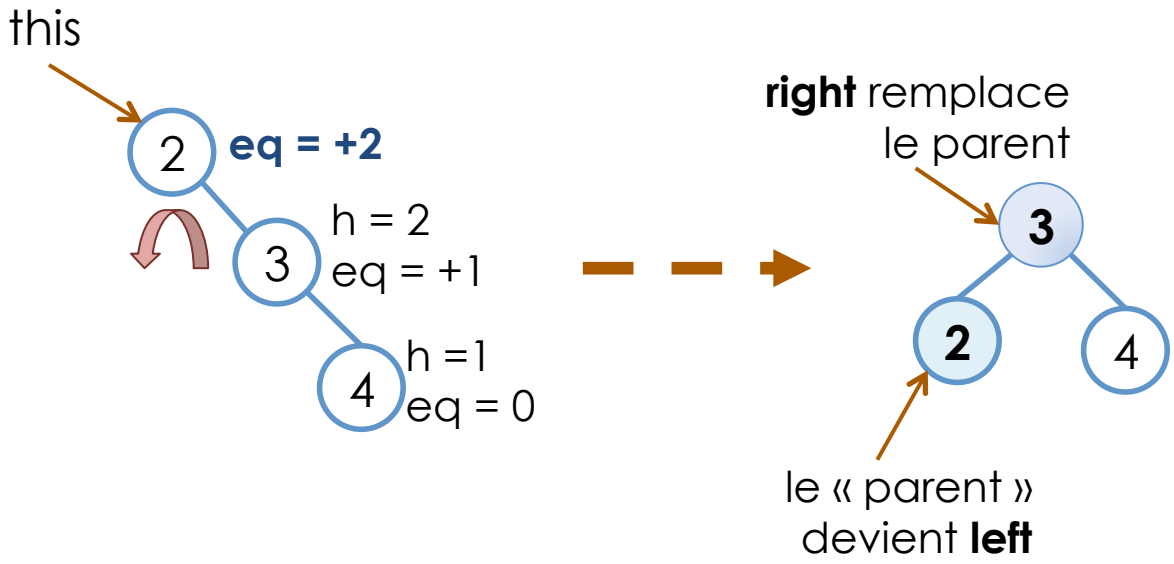
Rééquilibrage : Arbres AVL

- **Rotations à gauche**
  - le sous arbre droit est trop grand par rapport au sous arbre gauche
  - on descend le sommet vers la **gauche** et on remonte le fils **droit**



Rééquilibrage : Arbres AVL

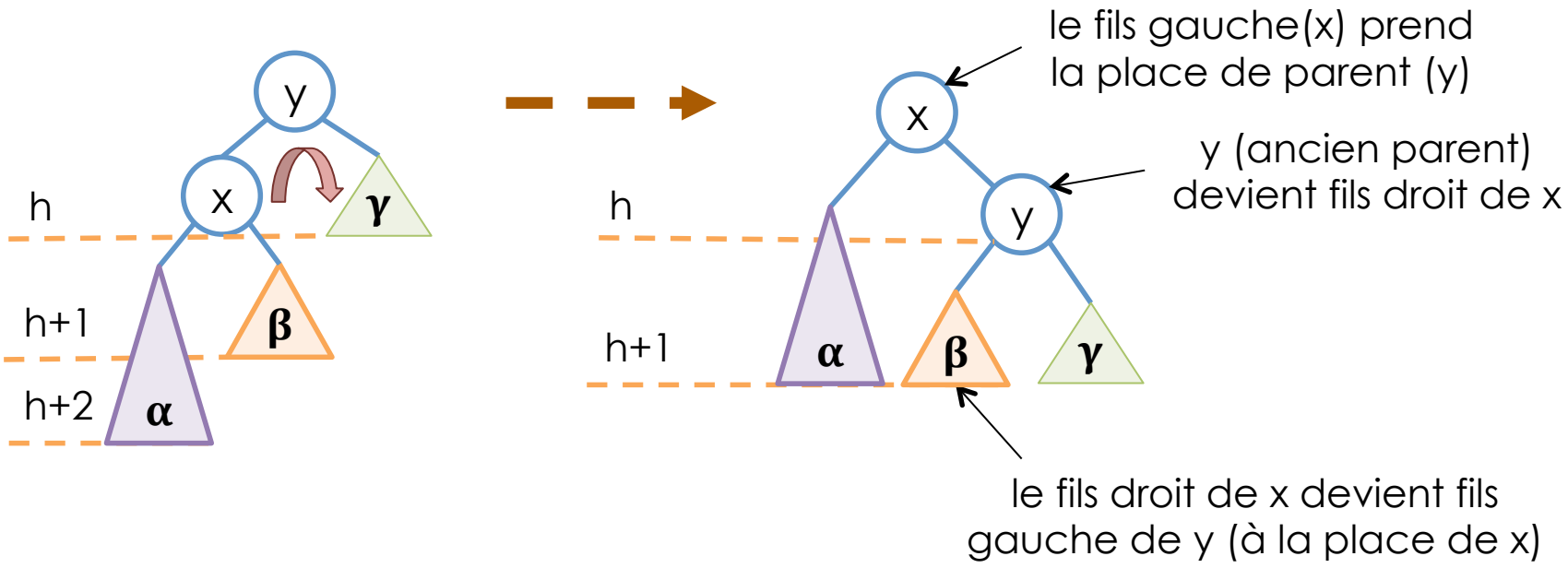
- **Rotations à gauche**
  - une rotation à gauche est nécessaire lorsque le facteur d'équilibrage atteint +2 (eq ≥ +2)



UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

Rééquilibrage : Arbres AVL

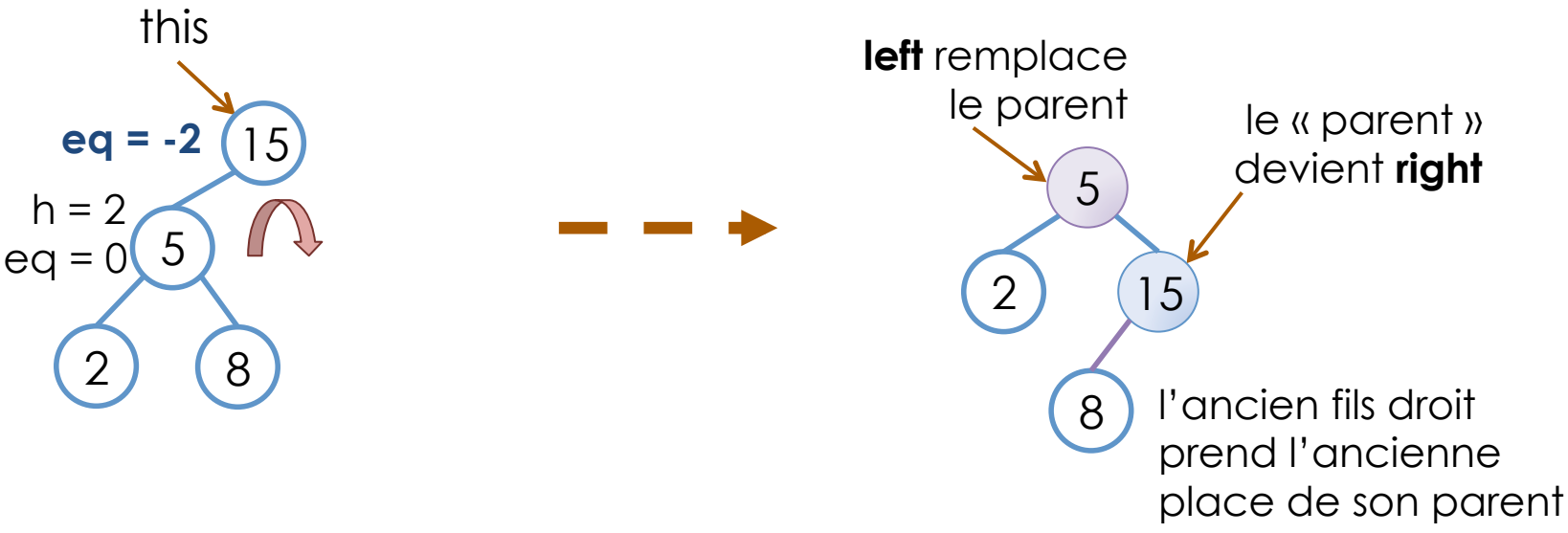
- **Rotations à droite**
  - le sous arbre gauche est trop grand par rapport au sous arbre droit
  - on descend le sommet vers la **droit** et on remonte le fils **gauche**



UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

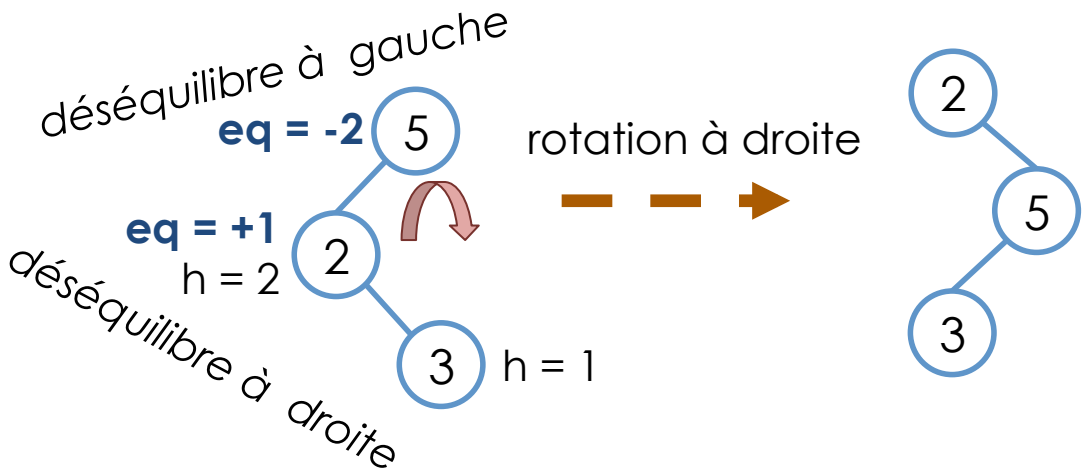
Rééquilibrage : Arbres AVL

- **Rotations à droit**
  - une rotation à droit est nécessaire lorsque le facteur d'équilibrage atteint -2 (eq ≤ -2)



Rééquilibrage : Arbres AVL

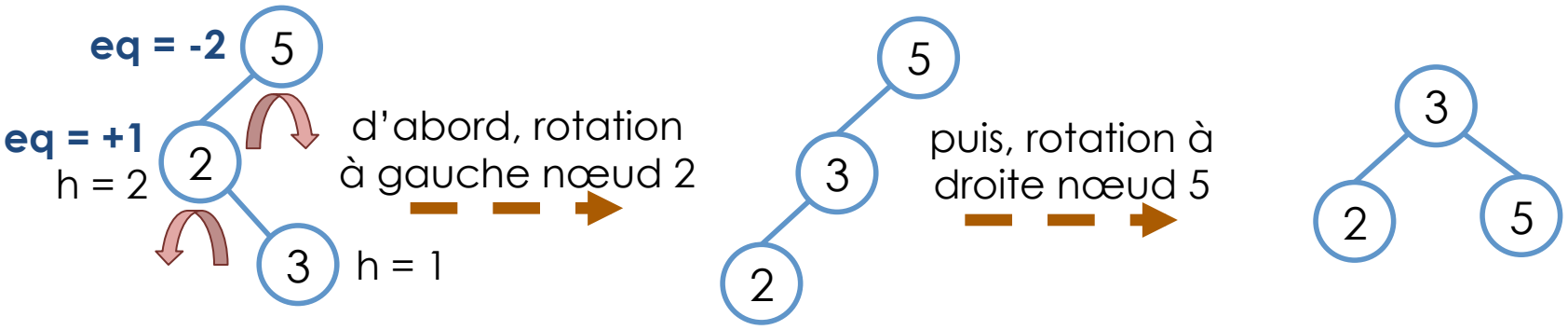
- **Rotation doubles**
  - une rotation simple ne résout pas tous les cas



Même **après la rotation**, l'arbre **demeure déséquilibré**.

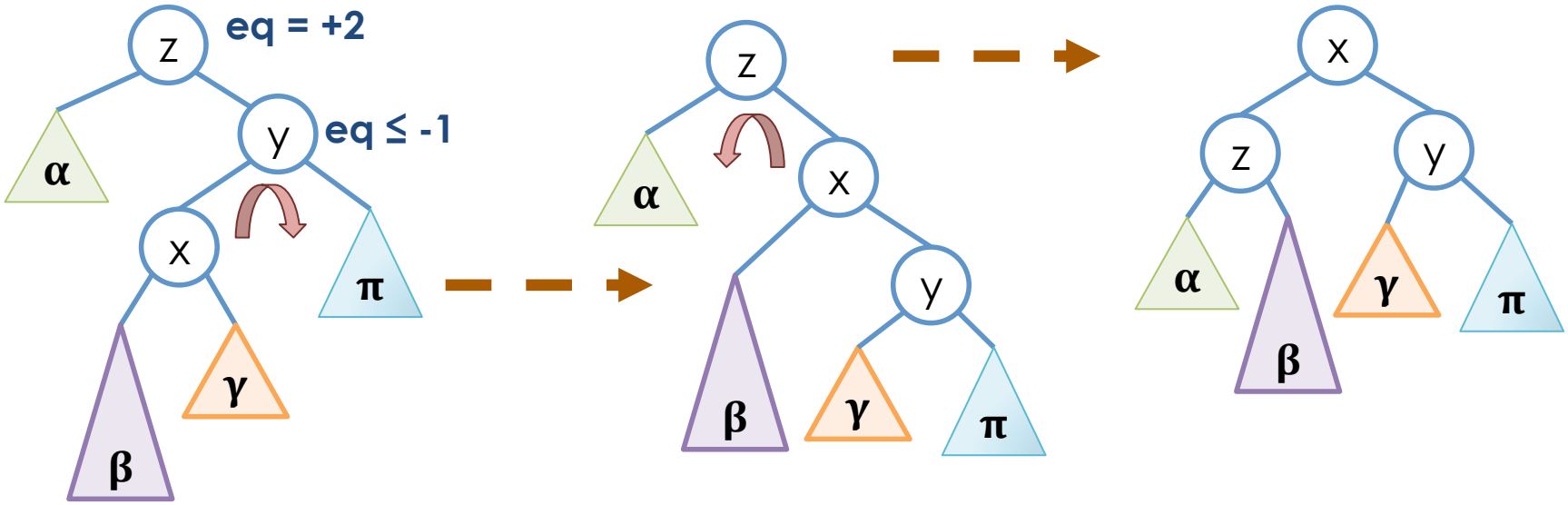
A l'origine du problème, le nœud et son fils « s'inclinent » différemment : le **nœud** va à **gauche** (**eq ≤ -2**), alors que son **fils gauche** va à **droite** (**eq ≥ +1**).

Une **double rotation** est nécessaire !



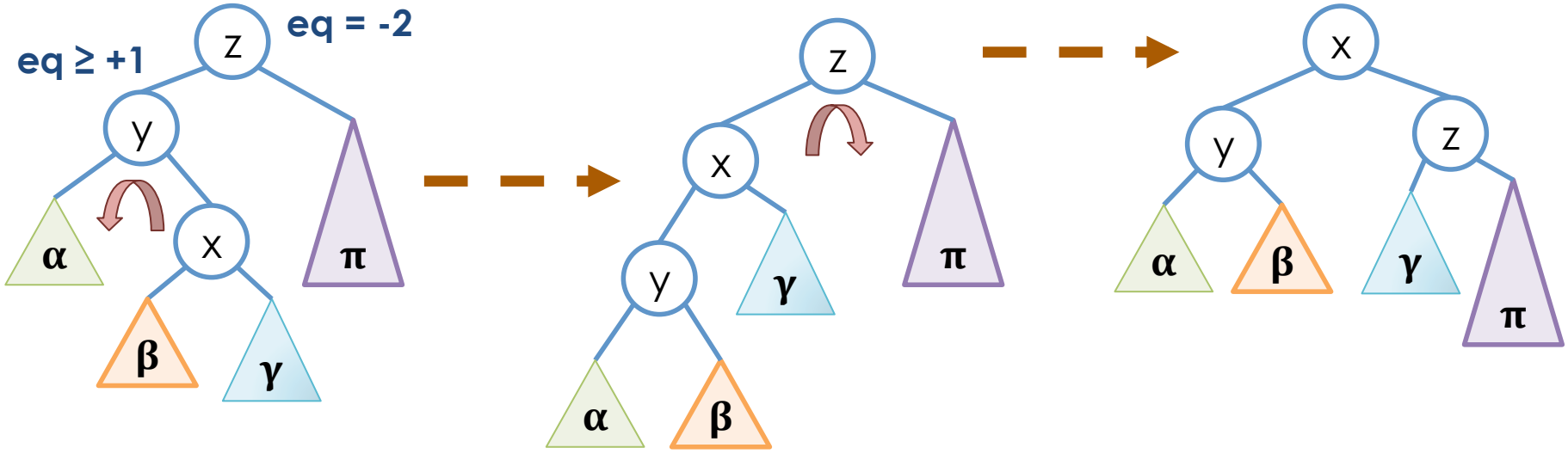
Rééquilibrage : Arbres AVL

- **Rotation doubles droite gauche**
  - lors que le facteur d'équilibrage d'un nœud est **+2** (**déséquilibre à droite**) et que celui de son **fils droit** est d'au moins **-1** (**déséquilibre à gauche**)
  - Double rotation : **rotation à droite** sur le **fils droit** (**y**), suivie d'une **rotation à gauche** sur le **sommet** (**z**).



Rééquilibrage : Arbres AVL

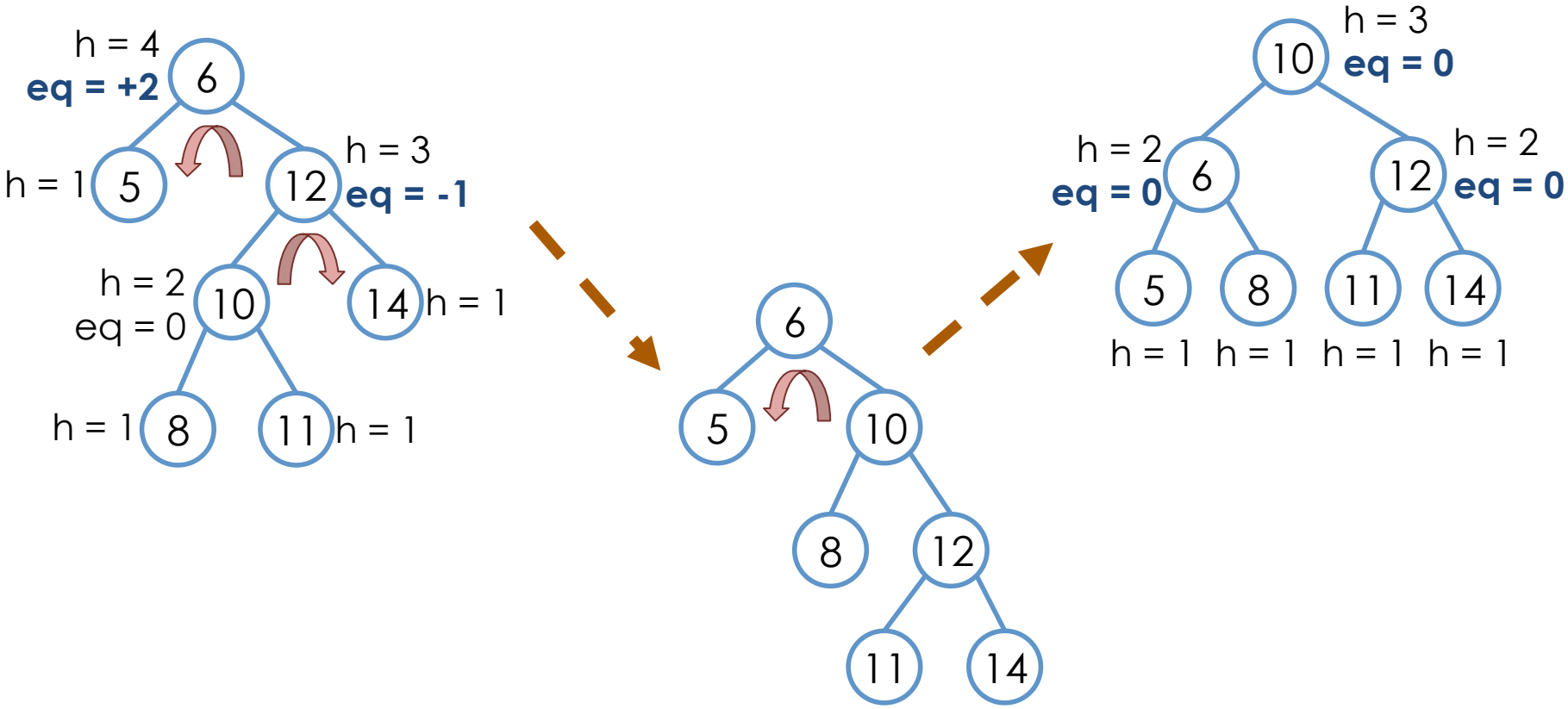
- **Rotation doubles gauche droite**
  - lors que le facteur d'équilibrage d'un nœud est **-2** (**déséquilibre à gauche**) et que celui de son **fils gauche** est d'au moins **+1** (**déséquilibre à droit**)
  - Double rotation : **rotation à gauche** sur le **fils gauche**, suivie d'une **rotation à droite** sur le **sommet**.



UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

Rééquilibrage : Arbres AVL

- **Rotation doubles**



Rééquilibrage : Arbres AVL

```
Reequilibrage ( )
  entier eq = this.eq ( )
  Si eq ≥ 2 Alors
     entier eqd = this.right.eq()
     Si eqd < 0 Alors
        Rotation_Droit()
        Rotation_Gauche()
     Sinon
        Rotation_Gauche()
     Fin Si
   Sinon si eq ≤ -2 Alors
      entier eqg = this.left.eq()
      Si eqg > 0 Alors
         Rotation_Gauche()
         Rotation_Droit()
     Sinon
        Rotation_Droit()
     Fin Si
   Fin Si
fin
```

```
eq () : entier
   Sortie : entier eq
   entier hd = 0, hg = 0
   Si this.right != null Alors
      hd = this.right.hauteur
   Fin Si
   Si this.left != null Alors
      hg = this.left.hauteur
   Fin Si
   eq = hd – hg
   retourner eq
fin
```

```
ajusterHauter ()
  entier hd=0, hg=0
  Si this.left != null Alors
     hg = this.left.hauteur
  Fin Si
   Si this.right != null Alors
     hg = this.right.hauteur
  Fin Si
  this.hauteur = 1 + max (hd, hg)
fin
```

Rééquilibrage : Arbres AVL

```
Rotation_Droit ( )
   AVL r = this.left
   K cle = this.key
   V val = this.value
   this.key = r.key
   this.value = r.value
   r.key = cle
   r.value = val

   this.left = r.left
   Si r.left != null Alors
      r.left.parent = this
   Fin Si
   r.left = r.right
   r.right = this.right
   Si r.right != null Alors
      r.right.parent = r
   Fin Si
   this.right = r
   r.ajusterHauteur()
   this.ajusterHauteur()
fin
```

```
Rotation_Gauche ( )
   AVL r = this.right
   K cle = this.key
   V val = this.value
   this.key = r.key
   this.value = r.value
   r.key = cle
   r.value = val

   this.right = r.right
   Si r.right != null Alors
      r.right.parent = this
   Fin Si
   r.right = r.left
   r.left = this.left
   Si r.left != null Alors
      r.left.parent = r
   Fin Si
   this.left = r
   r.ajusterHauteur()
   this.ajusterHauteur()
fin
```

Rééquilibrage : Arbres AVL

- Un arbre AVL doit être constamment équilibré
  - Les opérations d'insertion et de suppression peuvent modifier l'équilibre de l'arbre
    - Ces opérations peuvent affecter l'hauteur de l'arbre
      - l'hauteur ne varie pas forcément de +1 ou de -1 à chaque insertion/suppression
        - l'hauteur ne varie que si l'opération affecte la branche a plus longue (*hauteur = 1 +* ***max (hd, hg)***)
  - Il faut donc s'assurer de son équilibre à chacune de ces opérations
    - il faut rééquilibrer l'arbre à chaque opération
    - pour cela, il faut garder à jour l'attribut hauteur à chaque insertion et suppression

Rééquilibrage : Arbres AVL

```
Insérer (clé, valeur)
  Entrée : K clé, V valeur
  Sortie : entier hauteur
  entier h = 0

  Si cle < this.key Alors
    Si this.left == null Alors
      AVL n = nouveau AVL(cle, val)
      this.left = n
      n.parent = this
      n.hauteur = 1
      h = 1
    Sinon
      h  =this.left.inserer(cle, val)
    Fin Si

…
```

```
  Sinon Si cle > this.key Alors
      Si this.right == null alors
        AVL n = nouveau AVL(cle,val)
        this.right = n
        n.parents = this
        n.hauteur = 1
        h = 1
      Sinon
        h = this.right.inserer(cle,val)
      Fin Si
   Sinon
      lancer Exception
   Fin Si

  Si this.hauteur < (h+1) Alors
     this.hauteur = h+1
  Fin Si
  this.reequilibrage()
  retourner this.hauteur

fin
```

Rééquilibrage : Arbres AVL

```
Supprimer (clé)
  Entrée : K clé
  Si estVide() Alors lancer exception
  Fin Si

  Si cle < this.key Alors
    Si this.left != null Alors
      this.left.supprimer(cle)
    Sinon
      lancer exception
    Fin Si
  Sinon Si cle > this.key Alors
    Si this.right != null Alors
        this.right.supprimer(cle)
    Sinon
        lancer exception
    Fin Si
  Sinon
     this.supprimer()
  Fin Si
  Si !estVide() Alors
     this.ajusterHauteur()
     this.reequilibrage()
  Fin Si
fin
```

Rééquilibrage : Arbres AVL

Supprimer ( )

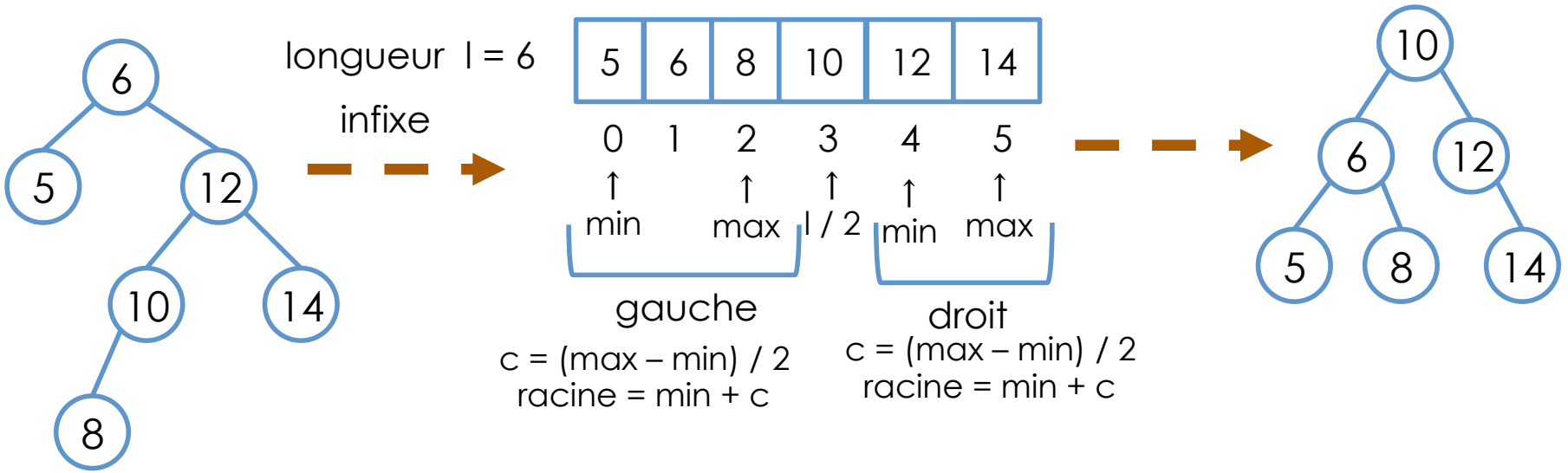
```
Si estFeuille() Alors
   Si parent != null Alors
      Si parent.left == this Alors
         parent.left = null
      Sinon
         parent.right = null
      FinSi
      key = value = null
      hauteur = 0
   Sinon Si left != null && right == null Alors
      hauteur = left.hauteur
      remplacePar(left)
      Si parent != null Alors
         parent.ajusterHauteur()
         parent.reequilibrage()
      FinSi
   Sinon Si left == null && right != null Alors
      hauteur = right.hauteur
      remplacePar(right)
      Si parent != null Alors
         parent.ajusterHauteur()
         parent.reequilibrage()
      FinSi
   Sinon
      AVL m  = right.min()
      this.key = m.key
      this.value = m.value
      AVL p = m.parent
      m.supprimer()
      Tant que p != this faire
         p.ajusterHauteur()
         p.reequilibrage()
         p = p.parent
      Fin faire
      this.ajusterHauteur()
      this.reequilibrage()
   FinSi
fin
```

Rééquilibrage

- Le rééquilibrage constant demandé par les arbres AVL peut s'avérer coûteux
- On peut réaliser l'opération de **rééquilibrage à la demande** sur un arbre ABR
  - **Parcours post-fixe**

```
ReeqOnDemande()

  Si this.left != null Alors
      left.ReeqOnDemande()
  fin Si

  Si this.right != null Alors
      right.ReeqOnDemande()
  fin Si

   this.ajusterHauteur()
   this.reequilibrage ()
fin
```

UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

Rééquilibrage

- Une autre possibilité est la reconstruction de l'arbre
  - parcours infixe offre le contenu de l'arbre dans l'ordre
  - en organisant les nœud, à l'aide d'un parcours infixe, dans une liste
    - l'élément central de la liste → racine
    - éléments à gauche → sous arbre gauche
    - éléments à droite → sous arbre droit