Självbalanserande AVL-träd
Weiss, avsnitt 4.4

Peter Ljunglöf

DAT036, Datastrukturer

30 nov 2012

Balanserade träd

Nodbalanserat träd:

- skillnaden i antalet noder mellan vänster och höger delträd är högst 1

Höjdbalanserat träd:

- skillnaden i höjd (djup) mellan vänster och höger delträd är högst 1

Nivåbalanserat träd:

- alla nivåer är fyllda, utom den nedersta som fylls på från vänster till höger

Notera att höjden av alla dessa träd är logaritmisk:

- $h = O(\log n)$, där n är antalet noder i trädet

Självbalanserande sökträd

Effektiviteten hos ett binärt sökträd är proportionell mot höjden på trädet:

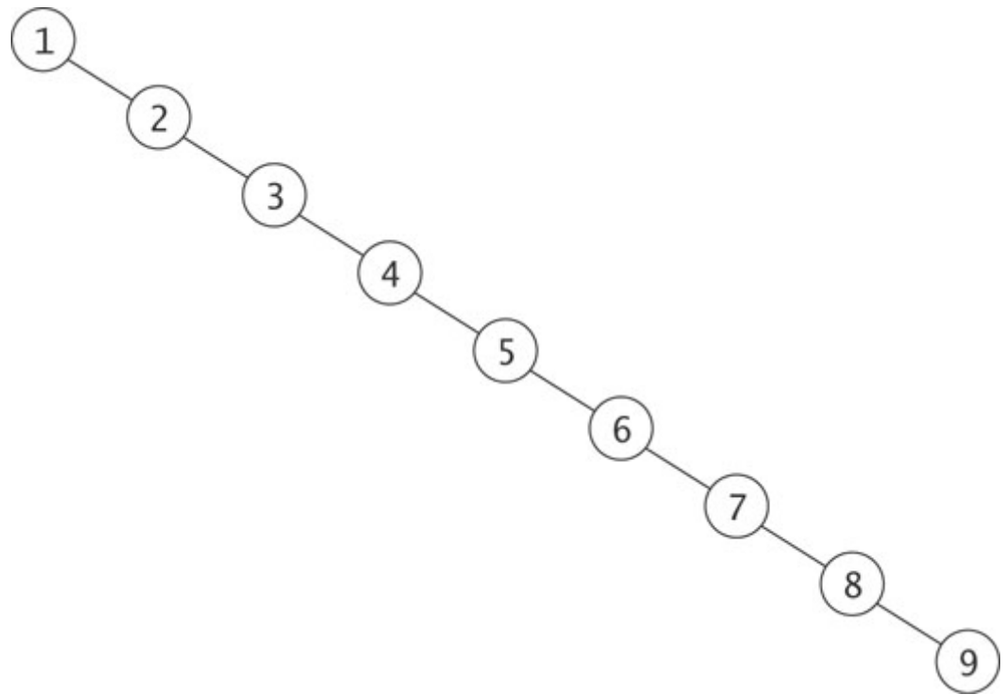
- ett balanserat träd är effektivare än ett obalanserat
- i värsta fall kan sökträdet ha linjär höjd, dvs $h = O(n)$
- då blir komplexiteten också linjär

För att lösa detta problem introducerar vi självbalanserande träd:

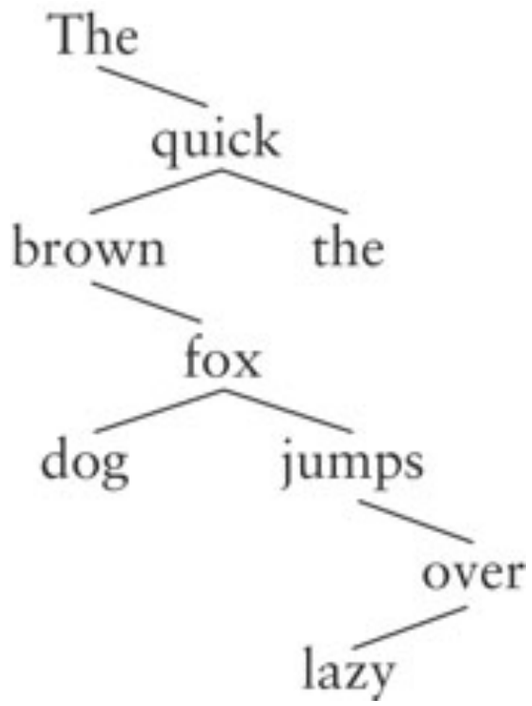
- vid insättning och uttagning så balanserar trädet om sig om det är nödvändigt

Obalanserade träd

Sökningar i detta obalanserade träd är O(n), inte O(log n)

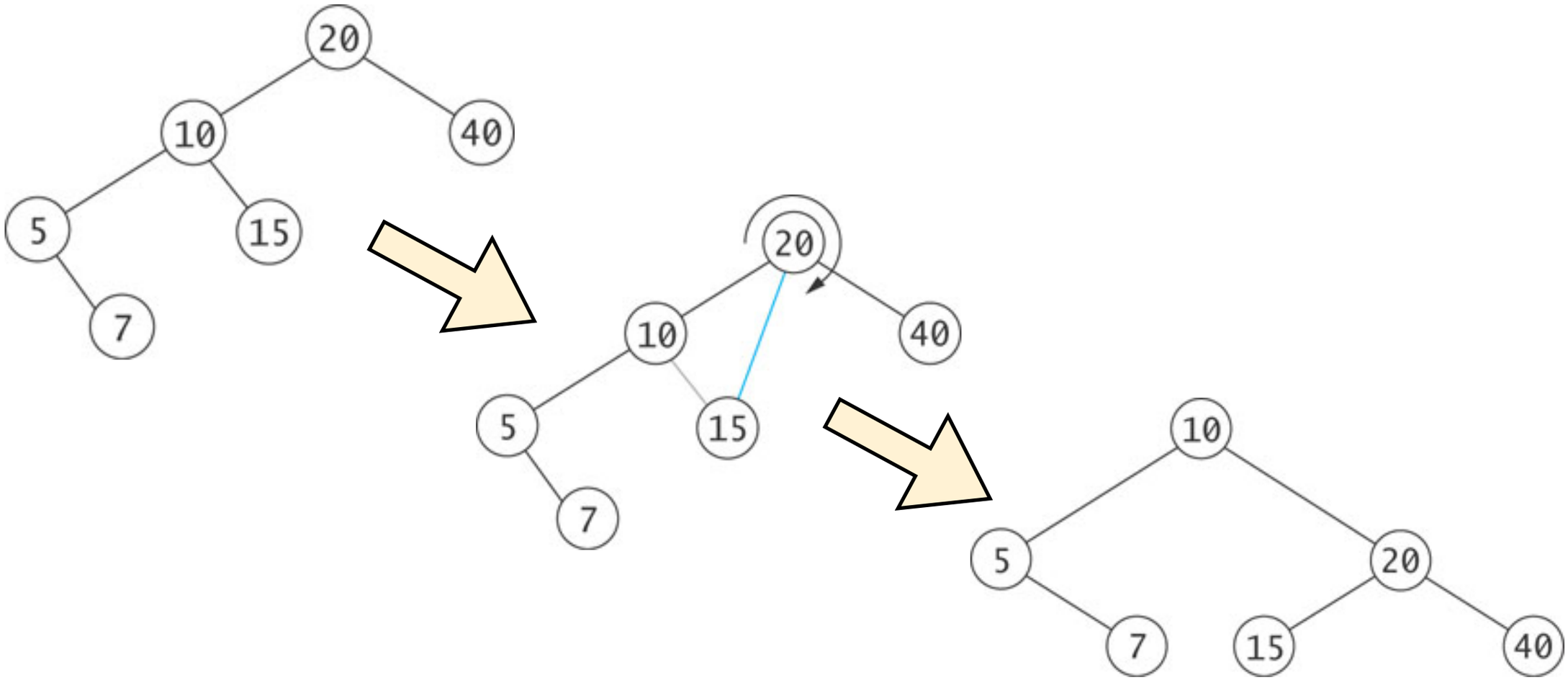


Här är ett mer realistiskt exempel på ett obalanserat träd



Rotering

Vi behöver en operation på binära träd som ändrar de relativa höjderna mellan vänster och höger delträd, men bibehåller sökträdsegenskapen



AVL-träd

Uppkallat efter Adelson-Velskii och Landis (Адельсóн-Вéльский och Лáндис)

AVL-trädet håller reda på höjdbalansen mellan vänster och höger delträd:

- höjdskillnaden mellan vänster och höger delträd får aldrig överstiga 1, i alla noder i trädet (inte bara rotnoden)
- om balansen hamnar utanför –1 till +1, roteras delträdet för att få det i balans igen

Balansera ett vänster-vänsterträd

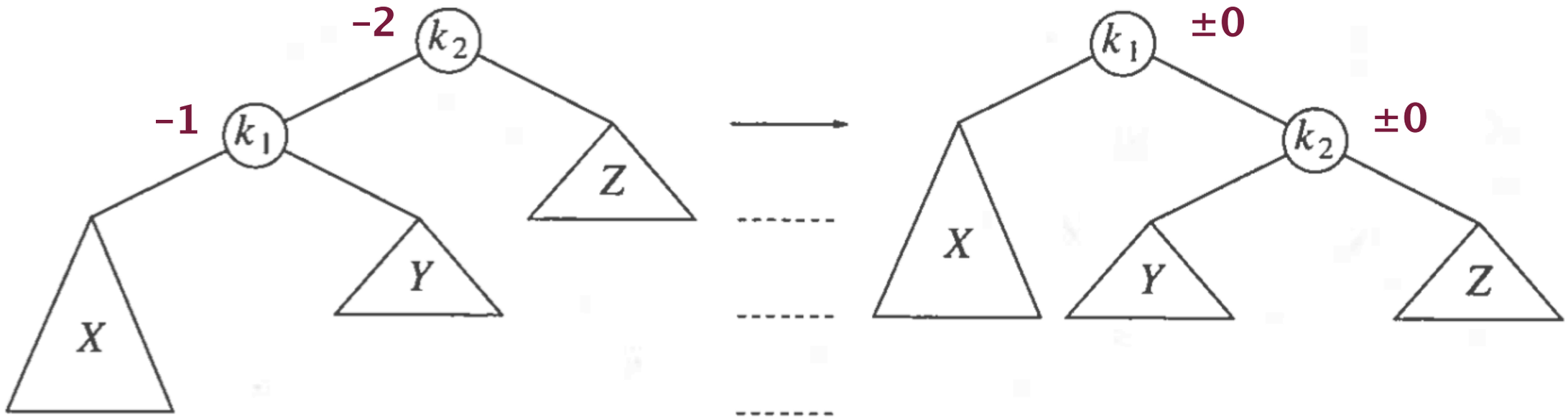


Figure 4.31 Single rotation to fix case 1

Balansera ett vänster-vänsterträd

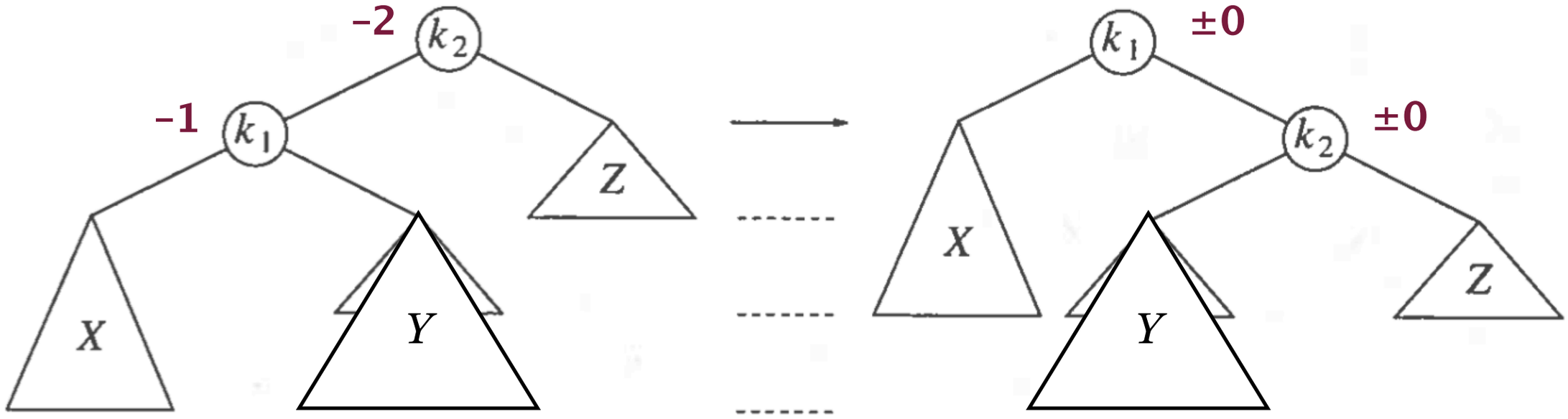


Figure 4.31 Single rotation to fix case 1

Balansera ett vänster-vänsterträd

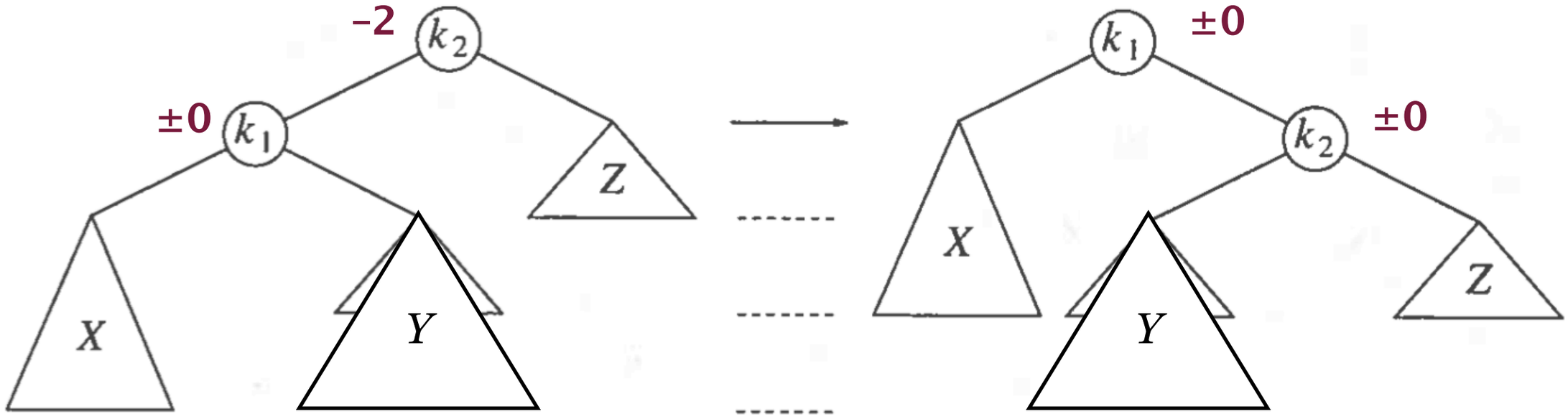


Figure 4.31 Single rotation to fix case 1

Balansera ett vänster-vänsterträd

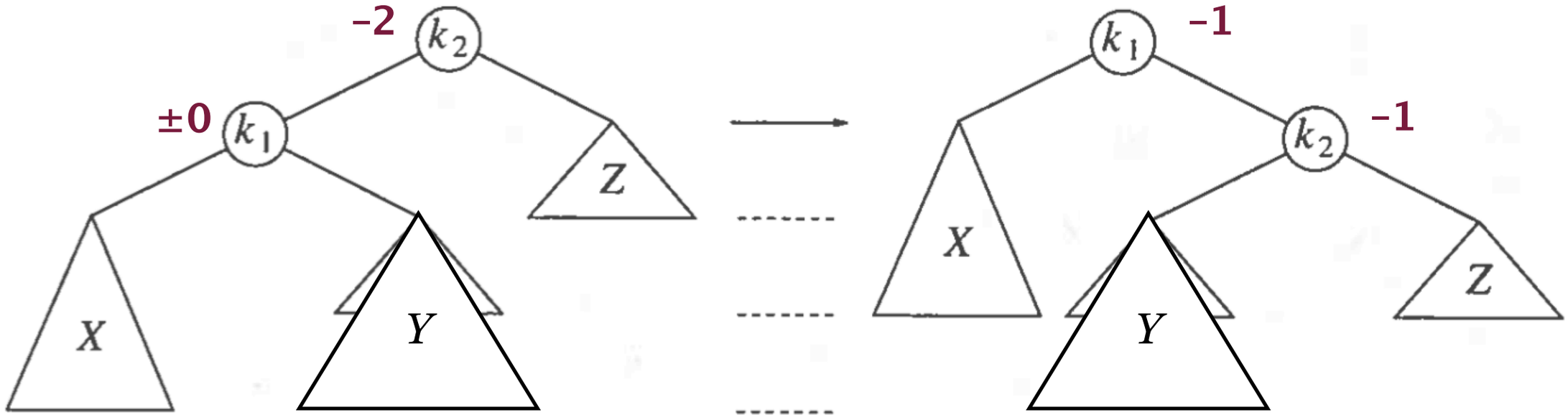


Figure 4.31 Single rotation to fix case 1

Balansera ett vänster-vänsterträd

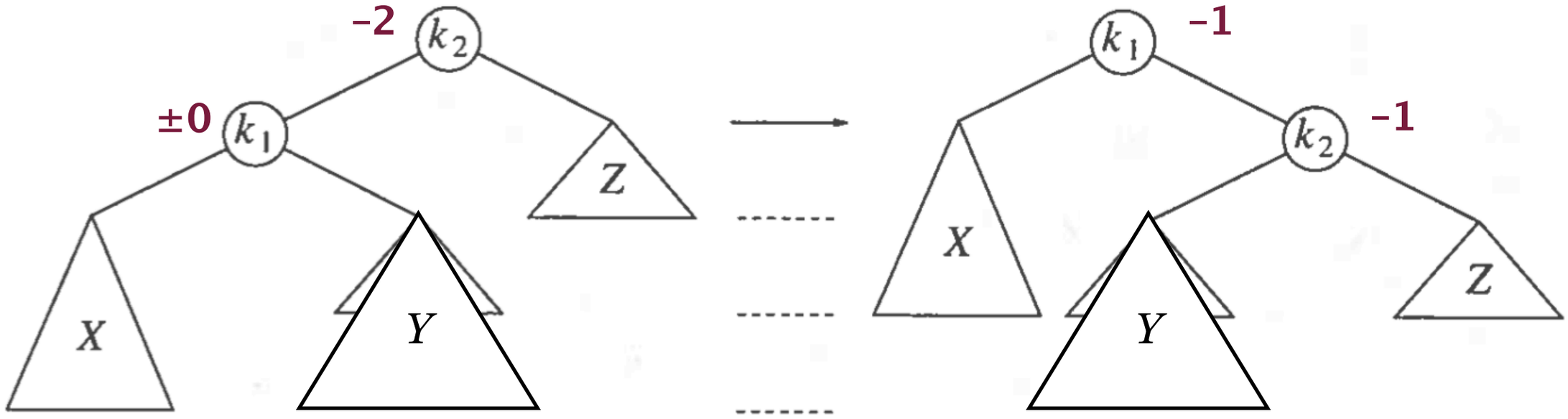


Figure 4.31 Single rotation to fix case 1

Det funkar även om Y är lika hög som X

Balansera ett höger-högerträd

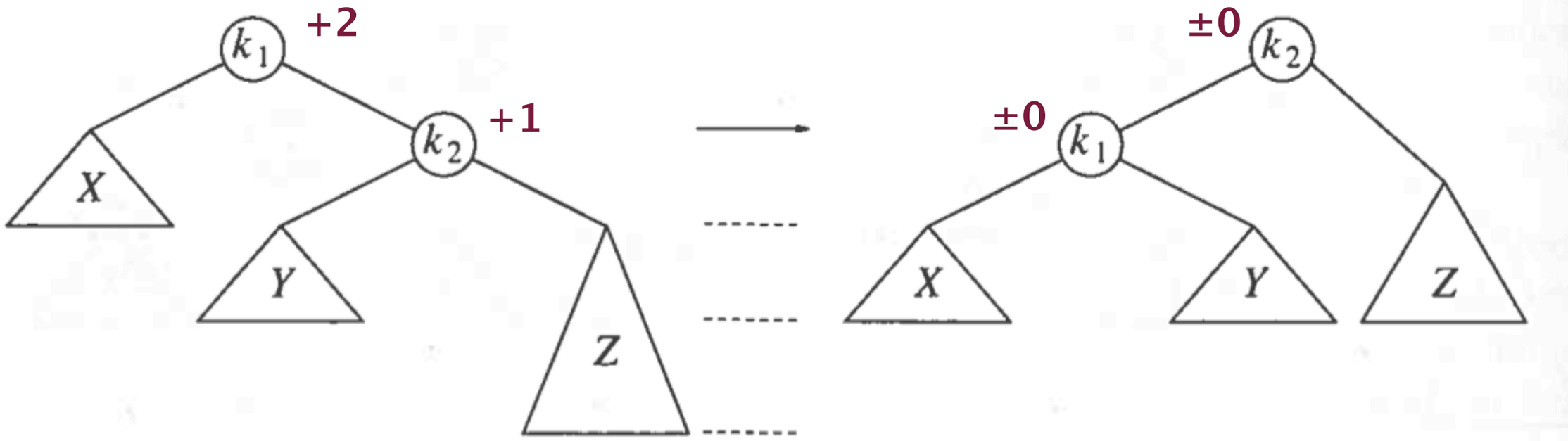


Figure 4.33 Single rotation fixes case 4

Balansera ett höger-högerträd

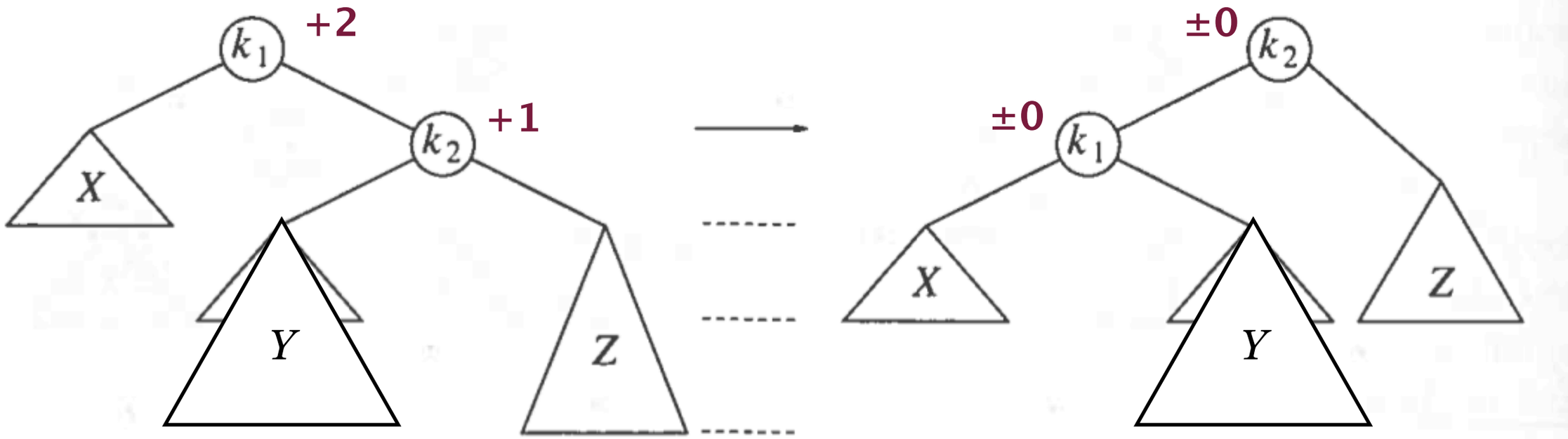


Figure 4.33 Single rotation fixes case 4

Balansera ett höger-högerträd

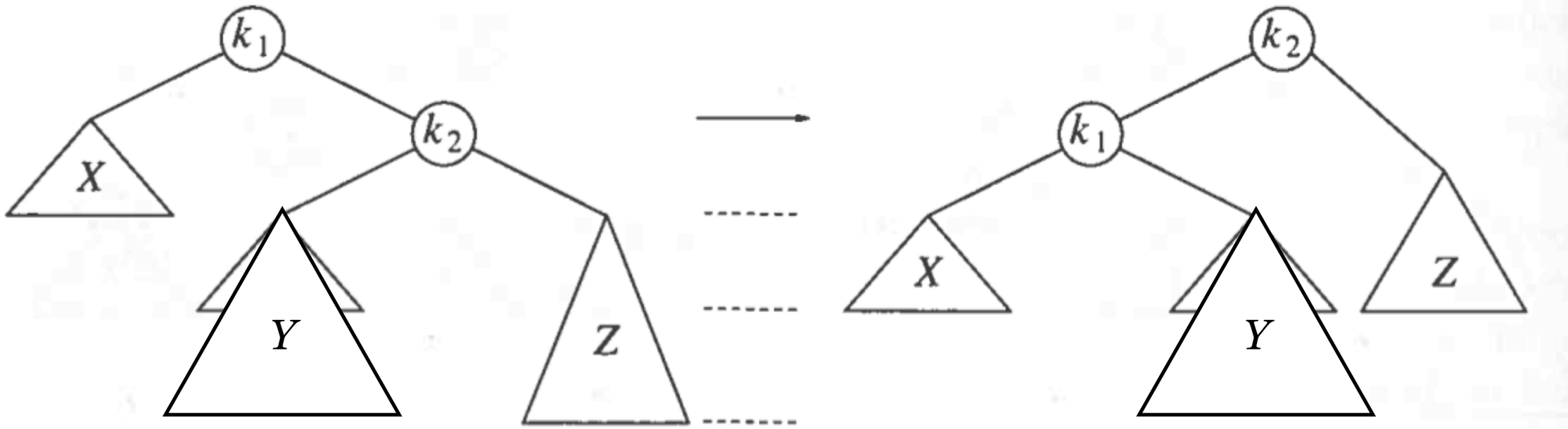


Figure 4.33 Single rotation fixes case 4

Balansera ett höger-högerträd

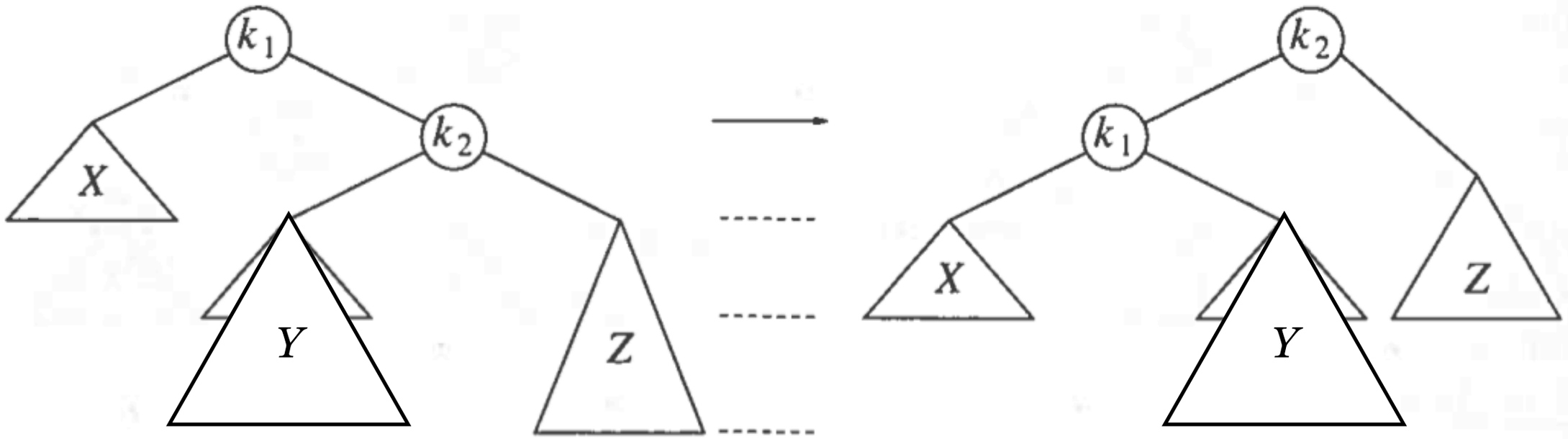


Figure 4.33 Single rotation fixes case 4

Det funkar även om Y är lika hög som Z

Balansera ett vänster-högerträd

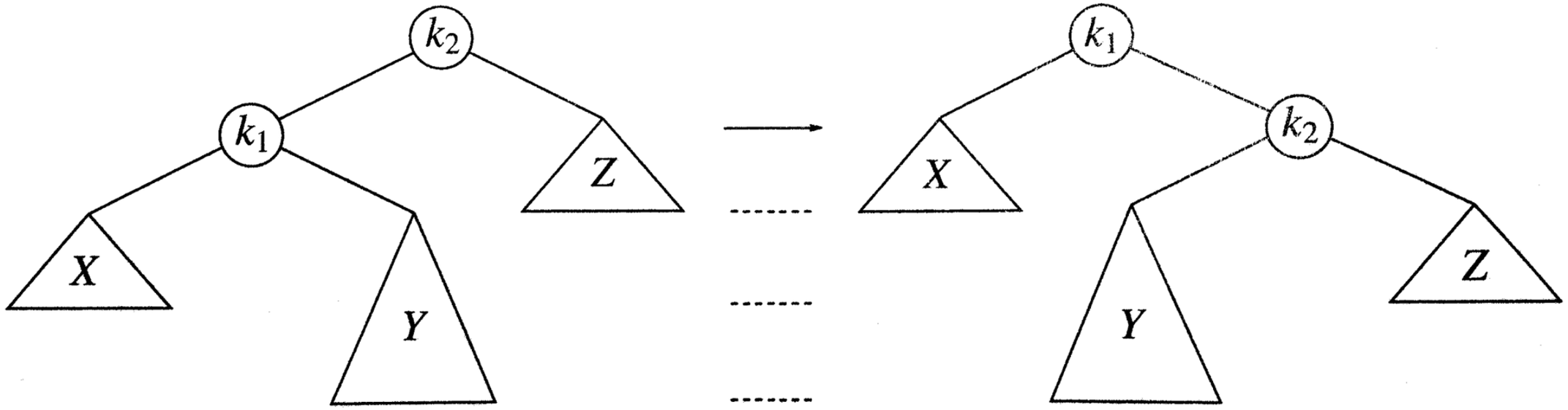


Figure 4.34 Single rotation fails to fix case 2

Balansera ett vänster-högerträd

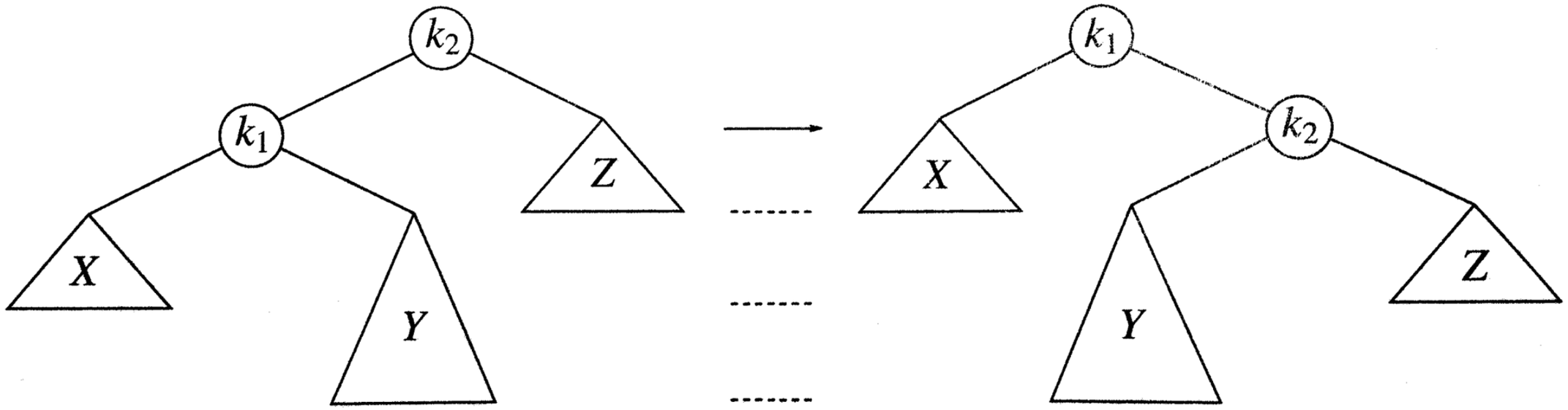


Figure 4.34 Single rotation fails to fix case 2

Enkelrotation funkar inte!

ett vänster-högerträd blir ett höger-vänster

Balansera ett vänster-högerträd

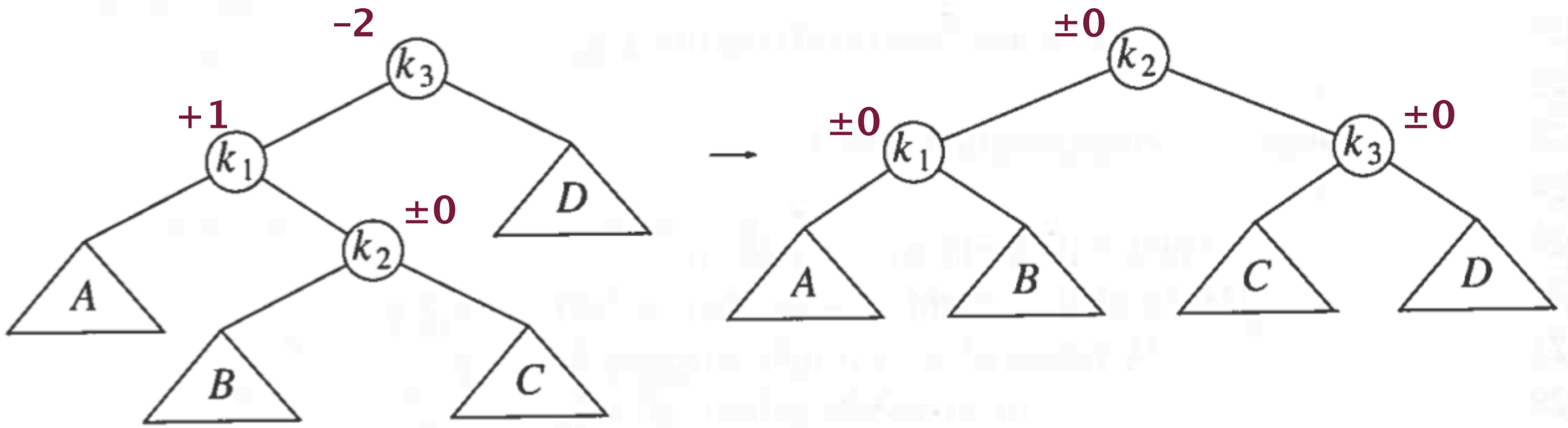


Figure 4.42 Double rotation

Vi behöver dubbelrotera!

Balansera ett vänster-högerträd

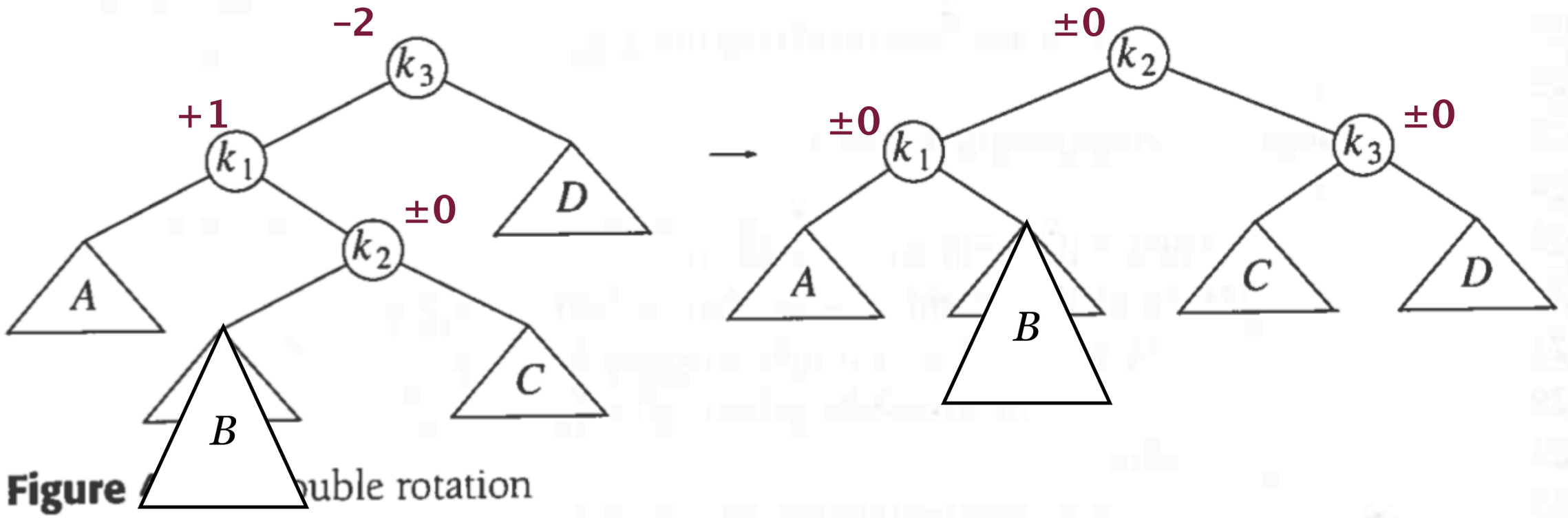


Figure ouble rotation

Vi behöver dubbelrotera!

Balansera ett vänster-högerträd

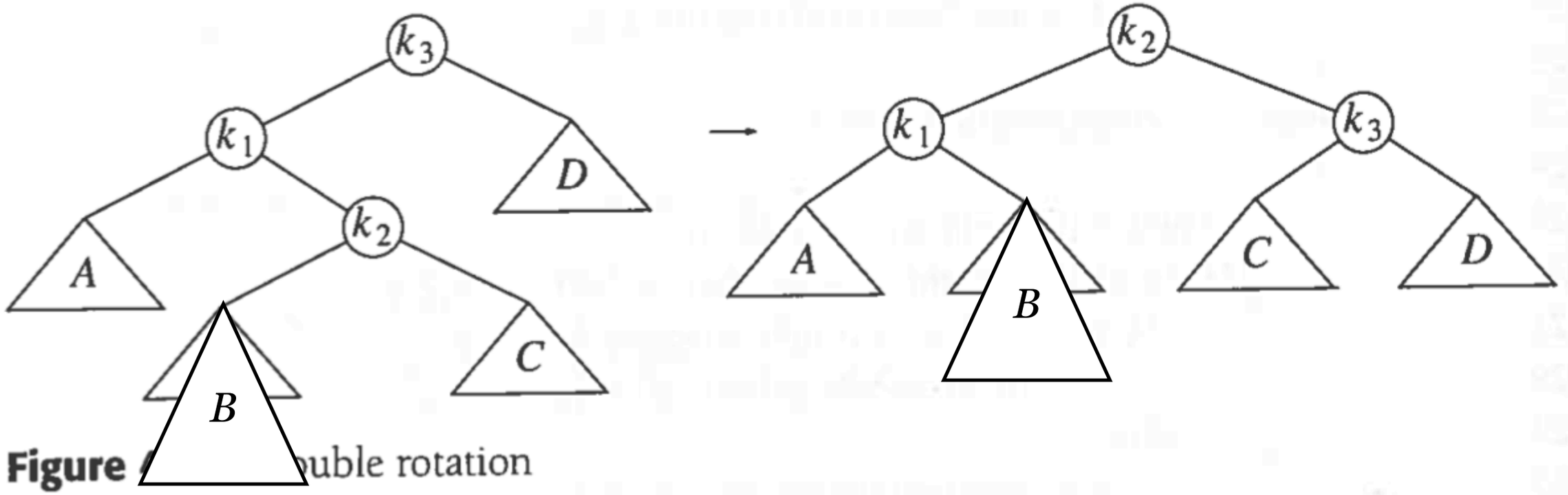


Figure ouble rotation

Vi behöver dubbelrotera!

Balansera ett vänster-högerträd

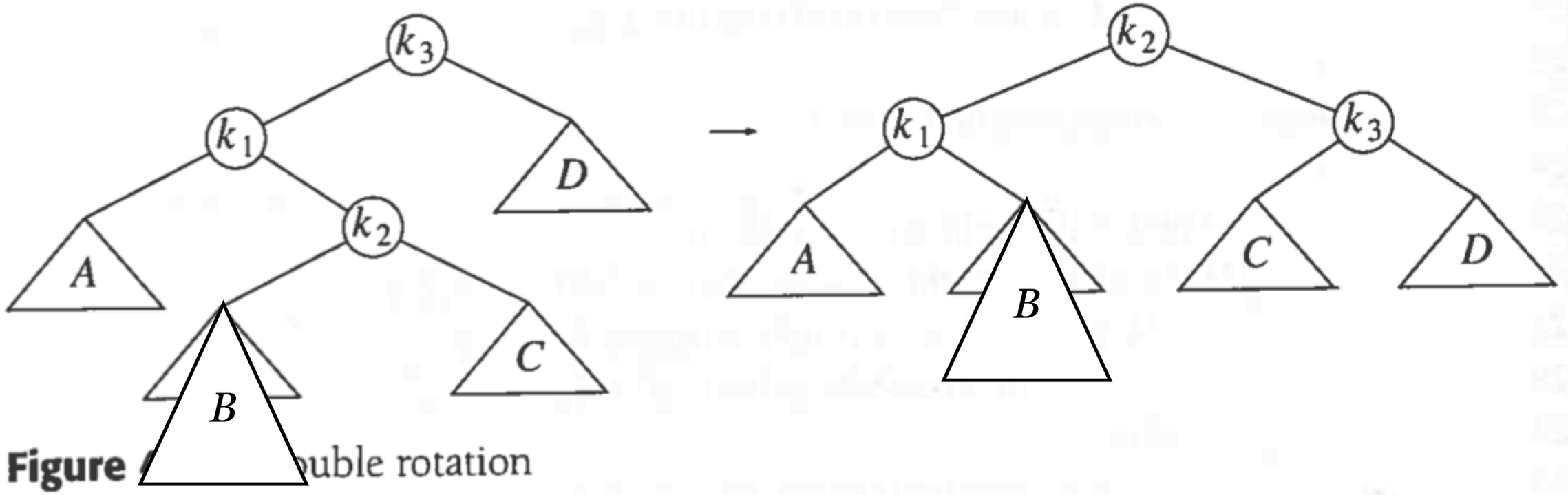


Figure ouble rotation

Vi behöver dubbelrotera!

Det funkar även om k_2 är lätt obalanserad

Balansera ett vänster-högerträd

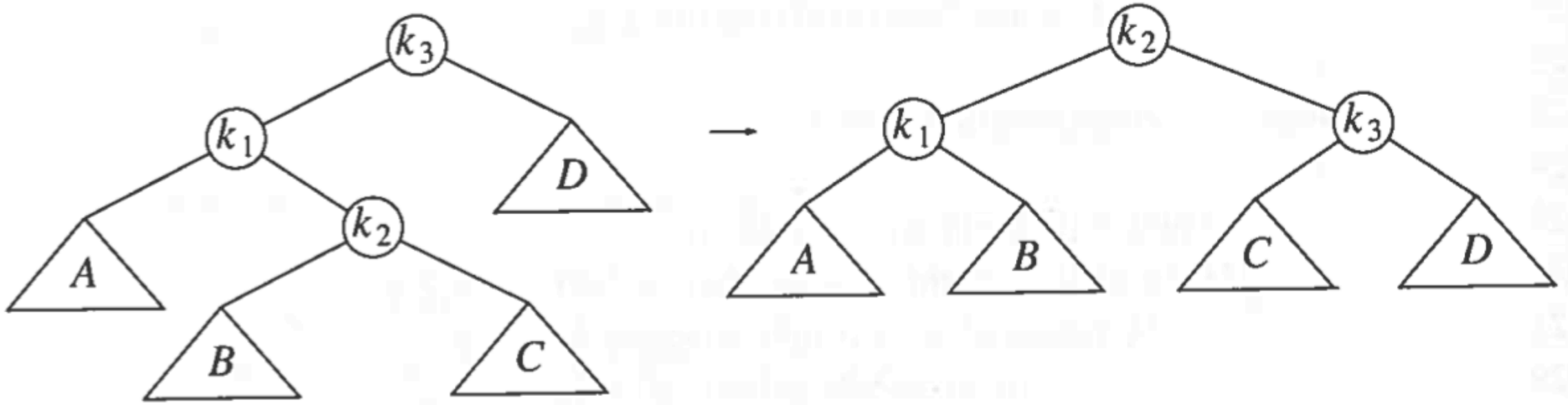


Figure 4.42 Double rotation

Vi behöver dubbelrotera!

Det funkar även om k_2 är lätt obalanserad

Balansera ett vänster-högerträd

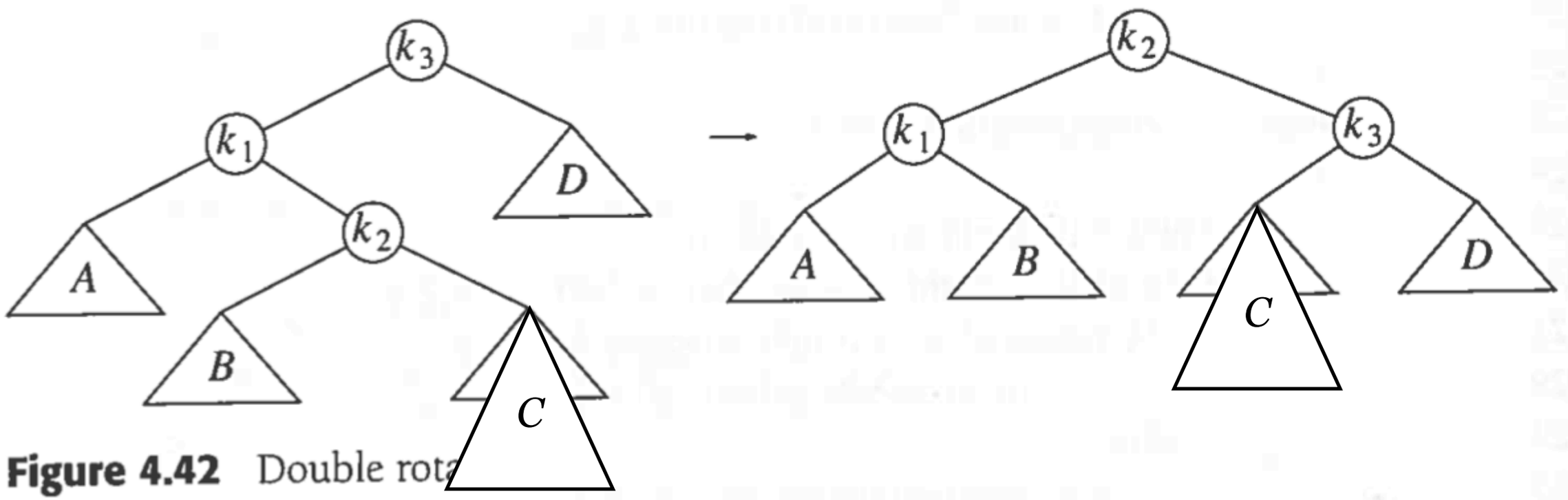


Figure 4.42 Double rota

Vi behöver dubbelrotera!

Det funkar även om k_2 är lätt obalanserad

Fyra sorters obalanserade träd

Vänster-vänster (föräldern < −1, vänster barn ≤ 0)

- rotera åt höger runt föräldern

Vänster-höger (föräldern < −1, vänster barn > 0)

- rotera åt vänster runt barnet
- rotera åt höger runt föräldern

Höger-höger (föräldern > 1, höger barn ≥ 0)

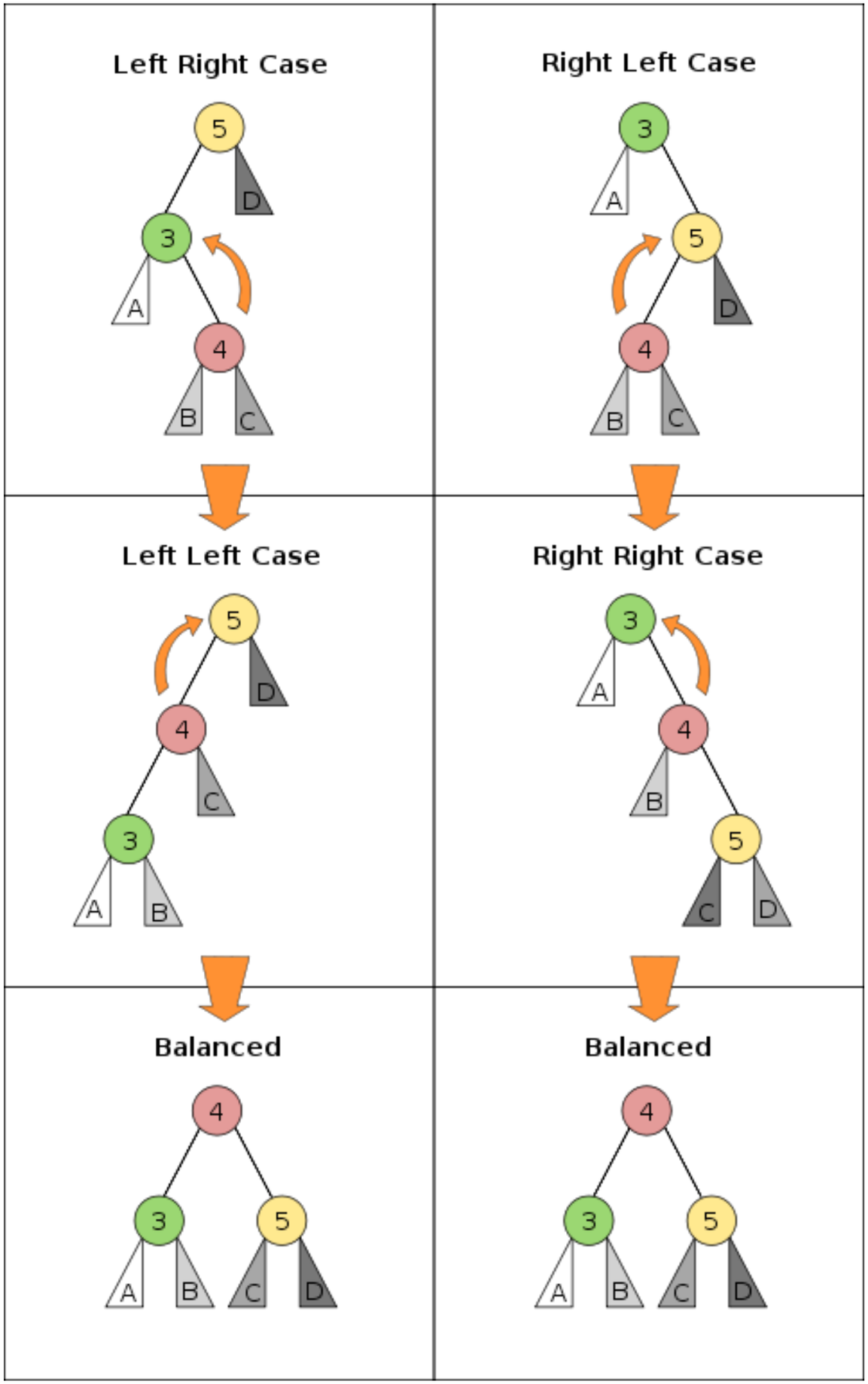
- rotera åt vänster runt föräldern

Höger-vänster (föräldern > 1, höger barn < 0)

- rotera åt höger runt barnet
- rotera åt vänster runt föräldern

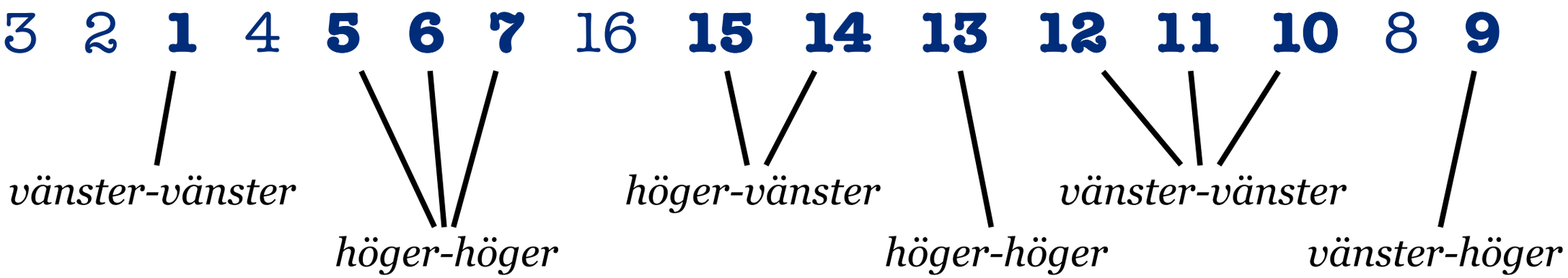
Balansering av de fyra olika fallen

(bilden är från Wikipedia)



Ett större AVL-exempel

Nu ska vi bygga ett AVL-träd för exemplet i avsnitt 4.4.1–4.4.2 (sid 127–131 i Weiss, 2nd ed.):



Vi gör det med följande Java-applet:

http://people.ksp.sk/~kuko/bak/index.html

Att implementera ett AVL-träd

```
private static class AvlNode<T> {
    T element;
    AvlNode<T> left;
    AvlNode<T> right;
    int height;

    AvlNode(T elem, AvlNode<T> lt, AvlNode<T> rt) {
        element = elem;
        left = lt;
        right = rt;
        height = 0;
    }
}

private int height(AvlNode<T> t) {
    return (t == null  ?  -1  :  t.height);
}
```

Att implementera ett AVL-träd

```
private static class AvlNode<T> {
    T element;
    AvlNode<T> left;
    AvlNode<T> right;
    int height;

    AvlNode(T elem, AvlNode<T> lt, AvlNode<T> rt) {
        element = elem;
        left = lt;
        right = rt;
        height = 0;
    }
}

private int height(AvlNode<T> t) {
    return (t == null  ?  -1  :  t.height);
}
```

det går att lagra balansen (–1/0/+1) istället för höjden, men såhär blir koden något enklare

Insättning i ett AVL-träd

```
private AvlNode<T> insert(T x, AvlNode<T> t) {
    if (t == null) return new AvlNode(x);
    if (x < t.element) {
        t.left = insert(x, t.left);
        if (height(t.left) – height(t.right) == 2) {
            if (x < t.left.element)
                t = rotateWithLeftChild(t);
            else
                t = doubleWithLeftChild(t);
        }
    } else if (x > t.element) {
        // symmetriskt med föregående fall
    }
    t.height = 1 + Math.max(height(t.left), height(t.right));
    return t;
}
```

Enkelrotering

```
private AvlNode<T> rotateWithLeftChild(AvlNode<T> k2) {
    AvlNode<T> k1 = k2.left;
    k2.left = k1.right;
    k1.right = k2;
    k2.height = 1 + Math.max(height(k2.left), height(k2.right));
    k1.height = 1 + Math.max(height(k1.left), k2.height);
    return k1;
}
```

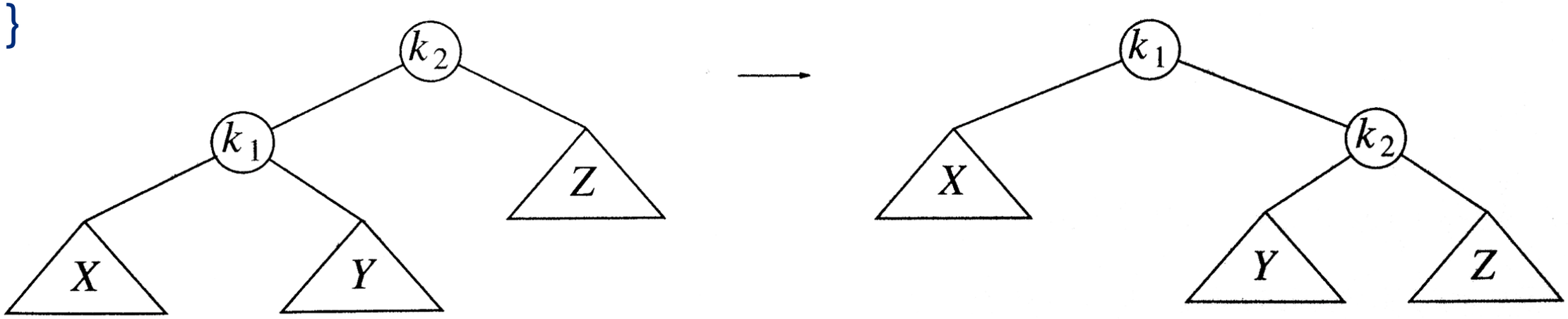


Figure 4.40 Single rotation

Dubbelrotering

```
private AvlNode<T> doubleWithLeftChild(AvlNode<T> k3) {
    k3.left = rotateWithRightChild(k3.left);
    return rotatewithLeftChild(k3);
}
```

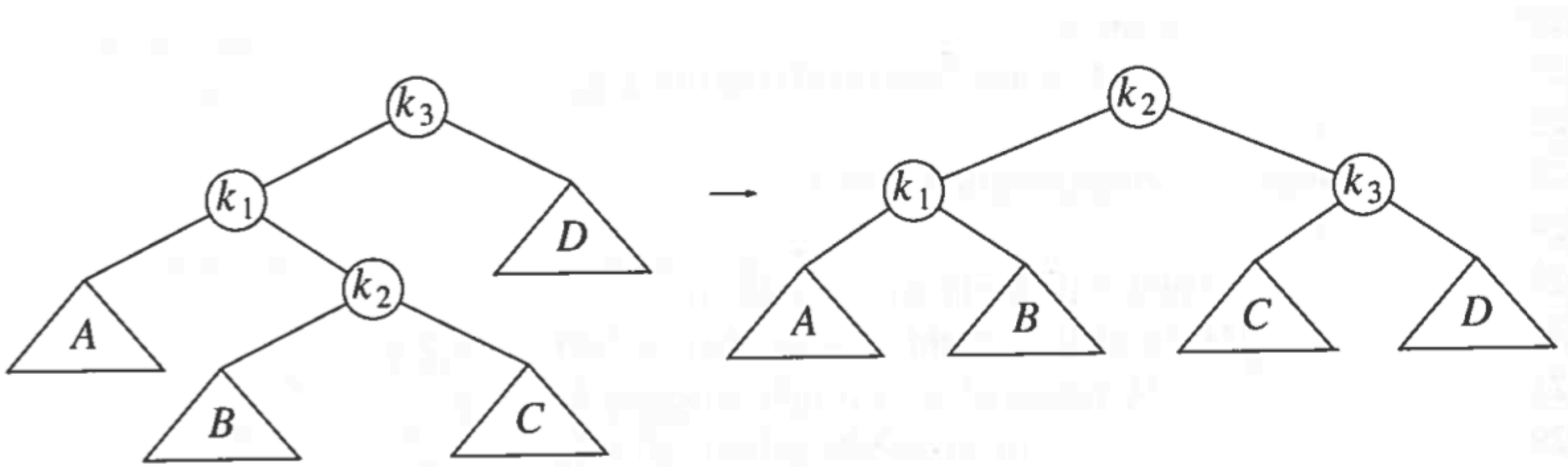


Figure 4.42 Double rotation

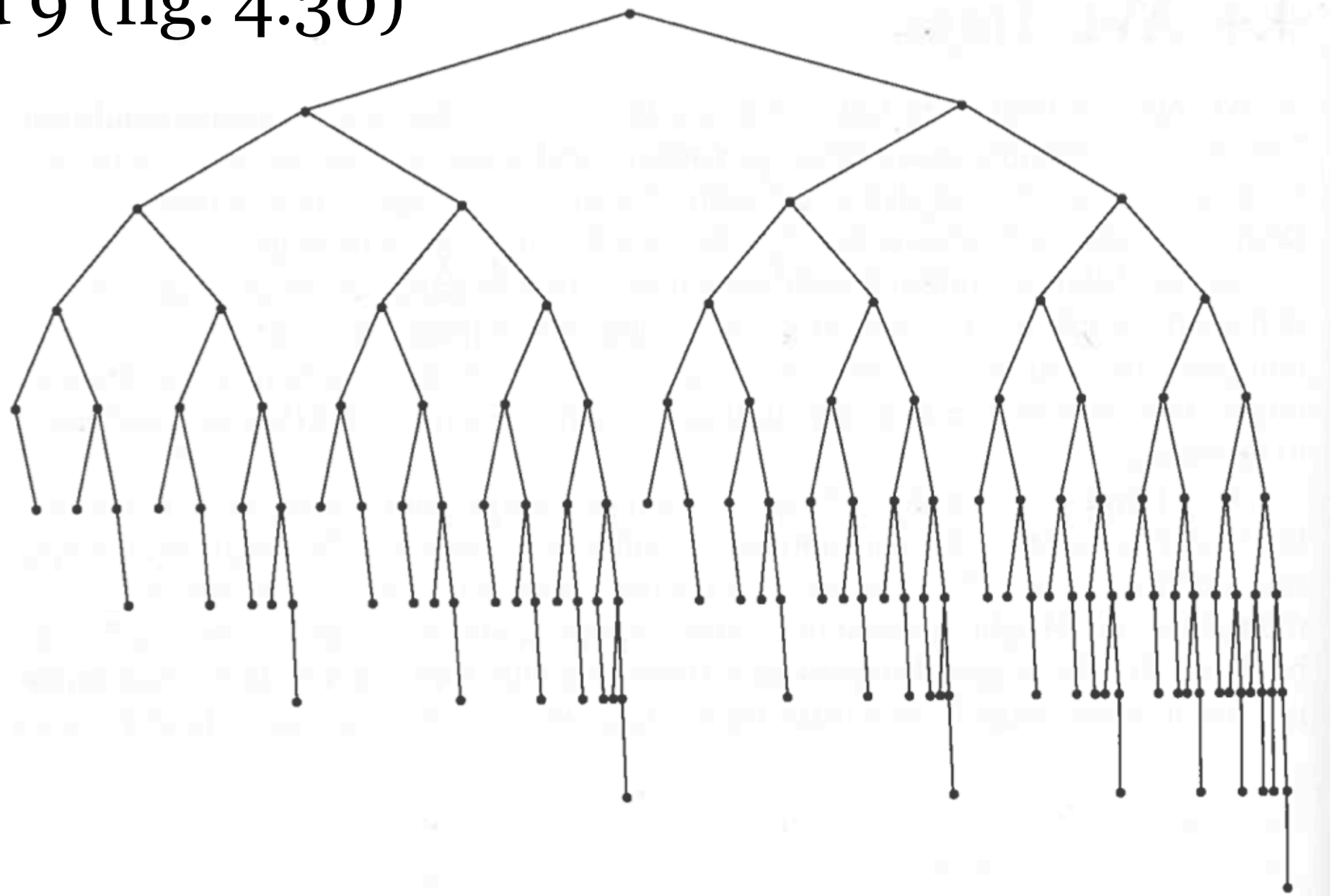
Effektivitet hos AVL-träd

AVL-träd är logaritmiska, O(log n), eftersom varje delträd är balanserat

- varje delträd får vara lite obalanserat (±1 balans), så trädet kan innehålla några hål
- i värsta fall (vilket är ovanligt) kan ett AVL-träd vara 1,44 gånger så högt som ett perfekt nodbalanserat träd

Det ”värsta” AVL-trädet

Följande träd är ”det minsta” AVL-trädet med höjd 9 (fig. 4.30)



Det ”värsta” AVL-trädet

Följande träd är ”det minsta” AVL-trädet med höjd 9 (fig. 4.30)