Wykład 3

Drzewa czerwono-czarne

Drzewa zbalansowane

- **Wprowadzenie**
- **Drzewa czerwono-czarne**
  - Definicja, wysokość drzewa
  - Rotacje, operacje wstawiania i usuwania

Literatura

- Cormen, Leiserson, Rivest, “Wprowadzenie do algorytmów”, rozdz. 14

Wprowadzenie

- **Drzewa BST są wygodne do efektywnego implementowania operacji: Search, Successor, Predecessor, Minimum, Maximum, Insert, Delete w czasie $O(h)$, (gdzie h jest wysokością drzewa)**
- **Jeśli drzewo jest zbalansowane (ma wtedy wysokość $h = O(\lg n)$), operacje te są najbardziej efektywne.**
- **Operacje wstawiania i usuwania elementów mogą powodować, że drzewo przestaje być zbalansowane. W najgorszym przypadku drzewo staje się listą liniową ($h = O(n)$)!**

Drzewa zbalansowane

- **Staramy się znaleźć takie metody operowania na drzewie, żeby pozostawało ono zbalansowane.**
- **Jeżeli operacja Insert lub Delete spowoduje utratę zbalansowania drzewa, będziemy przywracać tę własność w czasie co najwyżej $O(\lg n)$ → tak aby nie zwiększać złożoności.**
- **Potrzebujemy zapamiętywać dodatkowe informacje, aby to osiągnąć.**
- **Najbardziej popularne struktury danych dla binarnych drzew zbalansowanych to:**
  - Drzewa czerwono-czarne: o wysokości co najwyżej $2(\lg n + 1)$
  - Drzewa AVL: różnica wysokości dla poddrzew wynosi co najwyżej 1.

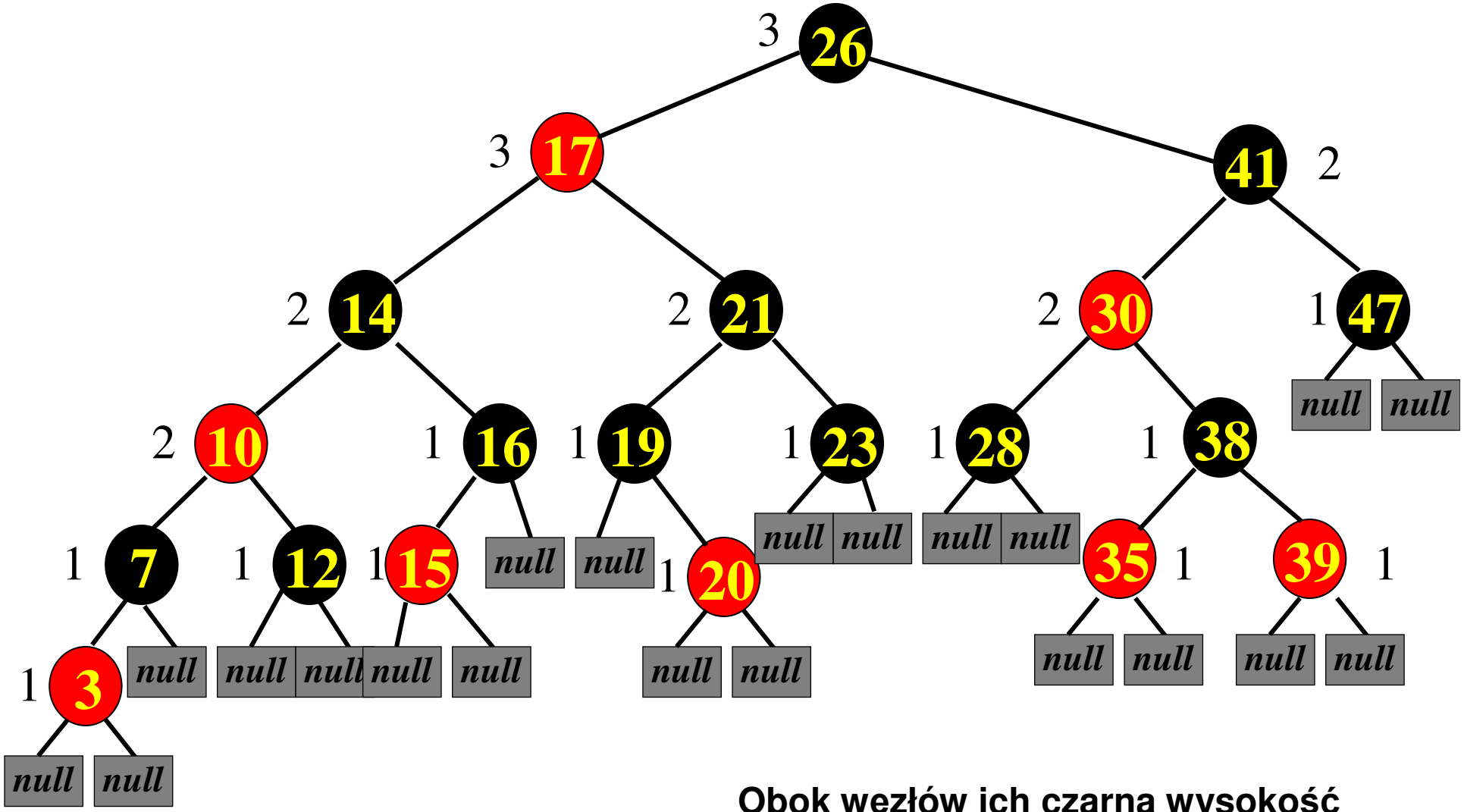
Drzewa czerwono-czarne: definicja

Drzewem czerwono-czarnym (RB tree) nazywany drzewo poszukiwań binarnych, dla którego każdy węzeł posiada dodatkowy bit koloru (albo czerwony albo czarny) o następujących własnościach:

1. Każdy węzeł posiada kolor – czerwony lub czarny.
2. Korzeń jest koloru czarnego.
3. Każdy liść (*null*) jest czarny.
4. Dla czerwonego węzła obydwoje dzieci są czarne.
5. Każda ścieżka od ustalonego węzła do liścia musi zawierać tą samą ilość czarnych węzłów.

Definicja: Czarną wysokością węzła– *bh*(*x*), nazywamy ilość czarnych węzłów na ścieżce prowadzącej od tego węzła do dowolnego liścia.

Przykład



Obok węzłów ich czarna wysokość
(węzły *null* mają czarną wysokość 0)

Wysokość w drzewie czerwono-czarnym

- **Lemat: drzewo czerwono-czarne o n wewnętrznych węzłach ma wysokość co najwyżej $2 \lg(n+1)$**

- **Dowód: dowiedziemy najpierw, że: dla każdego węzła x poddrzewo o korzeniu w x zawiera co najmniej $2^{bh(x)} - 1$ węzłów wewnętrznych.**

  Dowiedziemy tego przez indukcję względem wysokości węzła

  dla $h = 0$, węzeł jest liściem, wtedy $bh(x) = 0$. Stąd poddrzewo o korzeniu w x zawiera $2^0 - 1 = 0$ węzłów wewnętrznych.

Wysokość w drzewie czerwono-czarnym

Krok indukcyjny: niech *x* będzie węzłem wewnętrznym o wysokości $h > 0$, posiada więc dwóch potomków, oznaczmy ich *y* i *z*. Wtedy jeśli:

- *y* jest czarny → $bh(y) = bh(x)-1$
- *y* jest czerwony → $bh(y) = bh(x)$

Stąd: $bh(y) \geq bh(x)-1$

Możemy teraz wykorzystać założenie indukcyjne dla *y* ponieważ jego wysokość (nie czarna wysokość!) jest mniejsza od wysokości *x*. Stąd poddrzewo, o korzeniu w *y* zawiera co najwyżej $2^{bh(x)-1}-1$ wewnętrznych węzłów.

Mnożąc to przez 2 (dla oby potomków) i dodając 1, dostajemy ilość wewnętrznych węzłów dla poddrzewa o korzeniu w *x* nie przekraczającą $(2^{bh(x)-1}-1) + (2^{bh(x)-1}-1) + 1 = 2^{bh(x)}-1$.

Wysokość w drzewie czerwono-czarnym

- Niech teraz h będzie wysokością drzewa, o korzeniu w x. Dowiedliśmy poprzednio, że $n \geq 2^{bh(x)} - 1$
- Z własności 4. drzewa RB mamy – przynajmniej połowa węzłów na ścieżce od korzenia do liścia (nie licząc korzenia) musi być czarna (w drzewie nie może być sąsiednich czerwonych węzłów!)
- Stąd czarna wysokość drzewa wynosi co najmniej $h/2$, więc ilość węzłów wewnętrznych n wyniesie: $n \geq 2^{h/2} - 1$
- Daje to :

  $n + 1 \geq 2^{h/2} \rightarrow \lg (n+1) \geq 2 \lg h/2 \rightarrow h \leq 2 \lg (n+1)$

Statyczne operacje dla drzewa RB

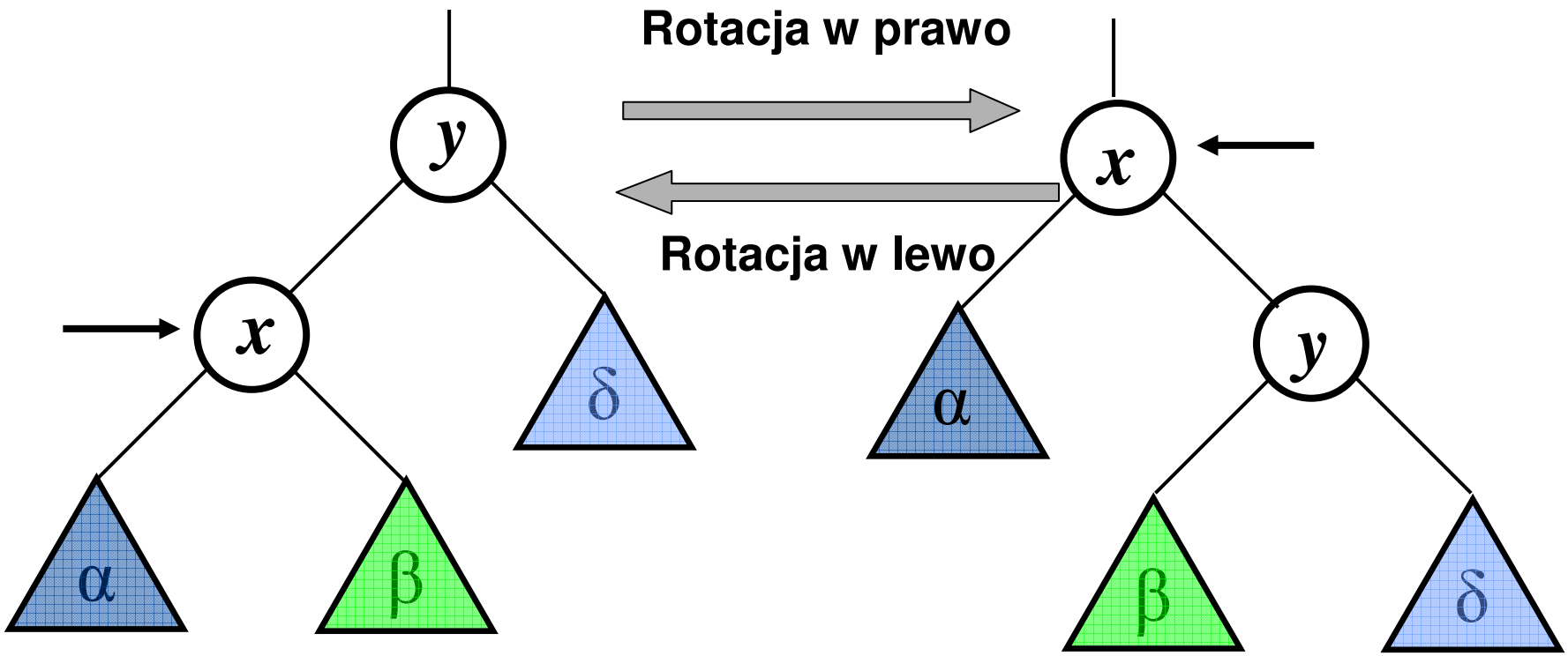
- **Operacje Max, Min, Search, Successor oraz Predecessor dla drzewa RB zajmują czas $O(\lg n)$.**
- **<u>Dowód:</u> operacje takie mogą być przeprowadzanie analogicznie, jak dla drzew BST, ponieważ nie modyfikują one drzewa (kolor węzła może być przy nich ignorowany).**

Dla drzew BST umiemy takie operacje przeprowadzać w czasie $O(h)$ gdzie h jest wysokością drzewa, a z lematu dostajemy wysokość drzewa $O(\lg n)$.

Dynamiczne operacje dla drzewa RB

- **Operacje Insert oraz Delete modyfikują drzewo.**
- **Po przeprowadzeniu takiej operacji drzewo może przestać być zbalansowane, oraz stracić własności drzewa RB.**
- **Dla utrzymania struktury RB, musimy niekiedy zmieniać kolory węzłów i ponownie balansować drzewo, zmieniając położenie poddrzew.**
- **Ponowne balansowanie odbywa się przy pomocy rotacji, a następnie „przekolorowaniu” węzłów (jeśli zajdzie taka potrzeba).**

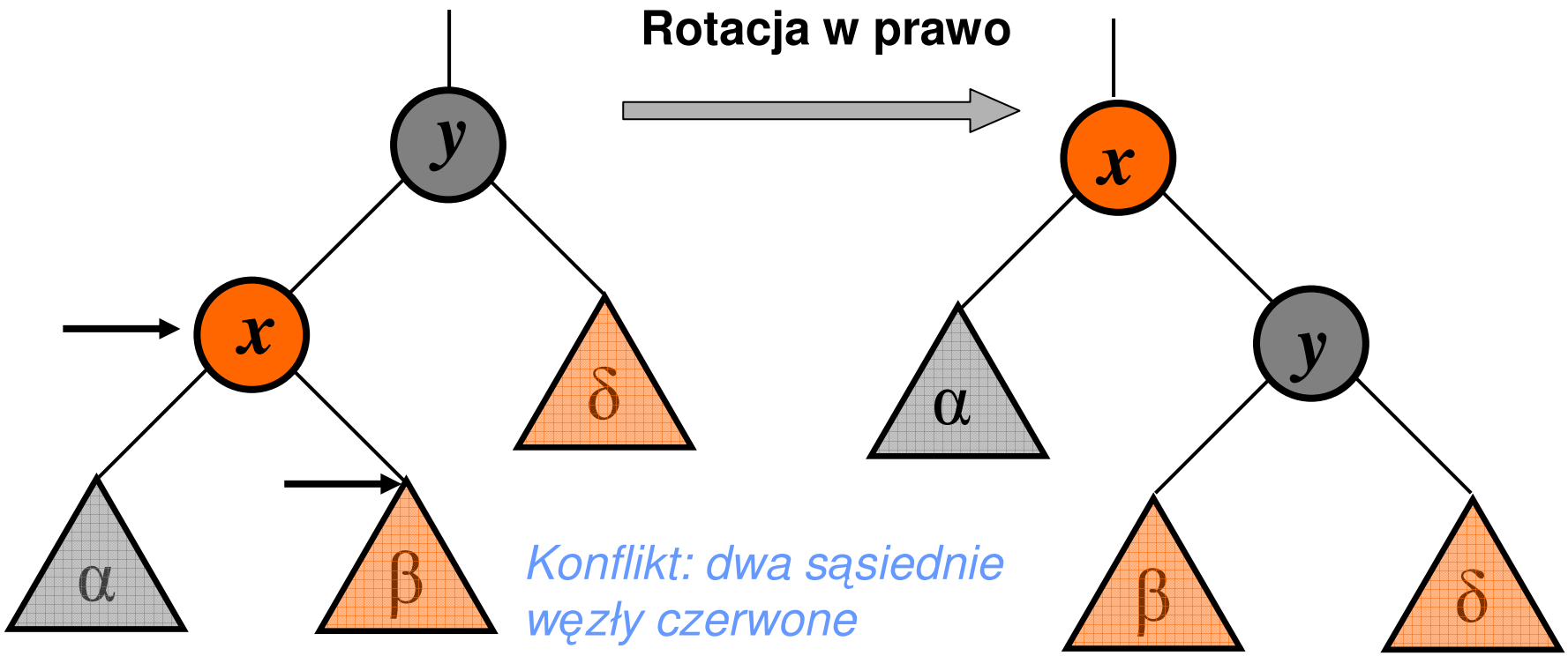
Rotacje



$\alpha \le x \le \beta$ i $x \le y \le \delta$

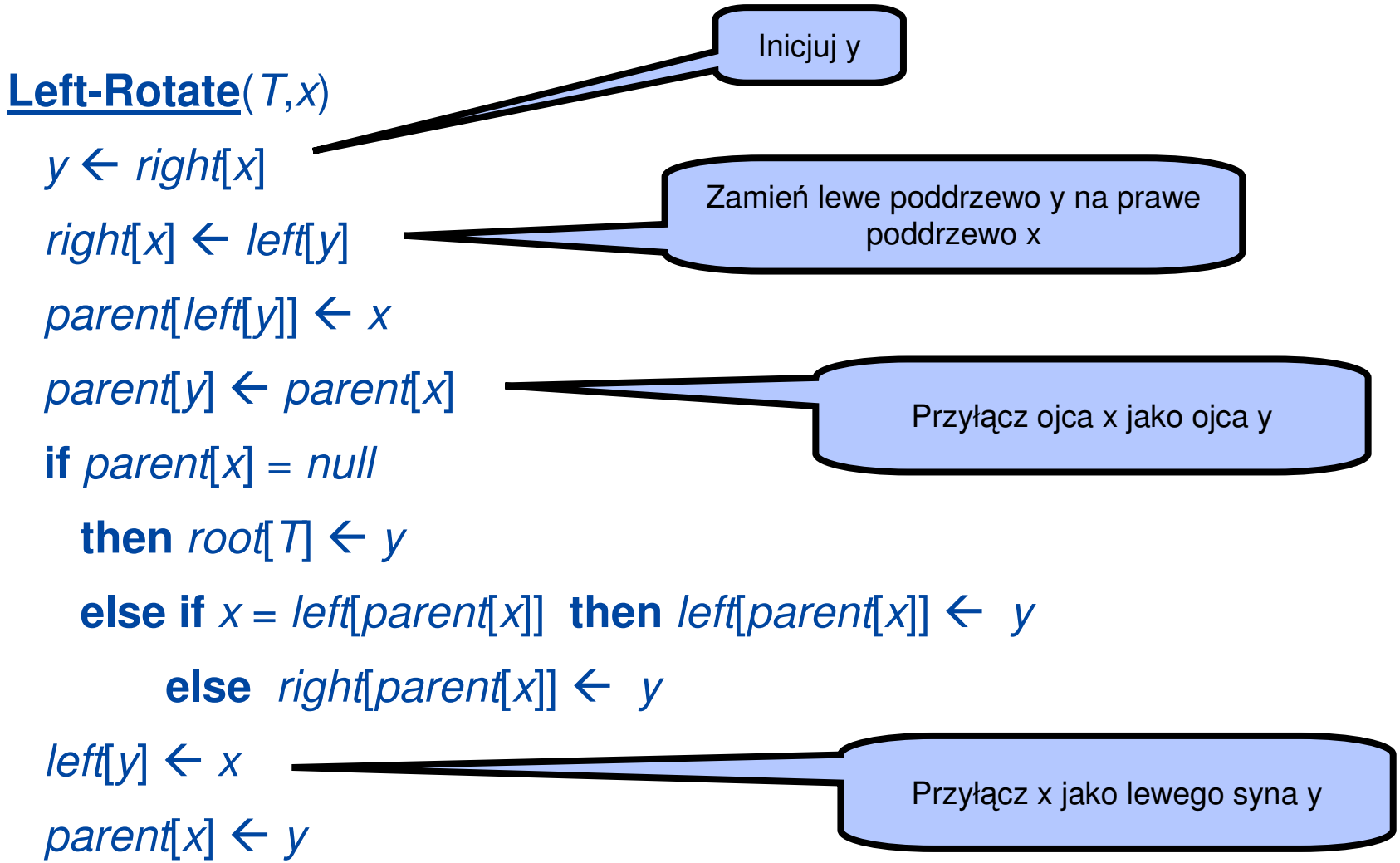
$\alpha \le x \le y$ i $\beta \le y \le \delta$

Rotacje mogą przywrócić własność RB

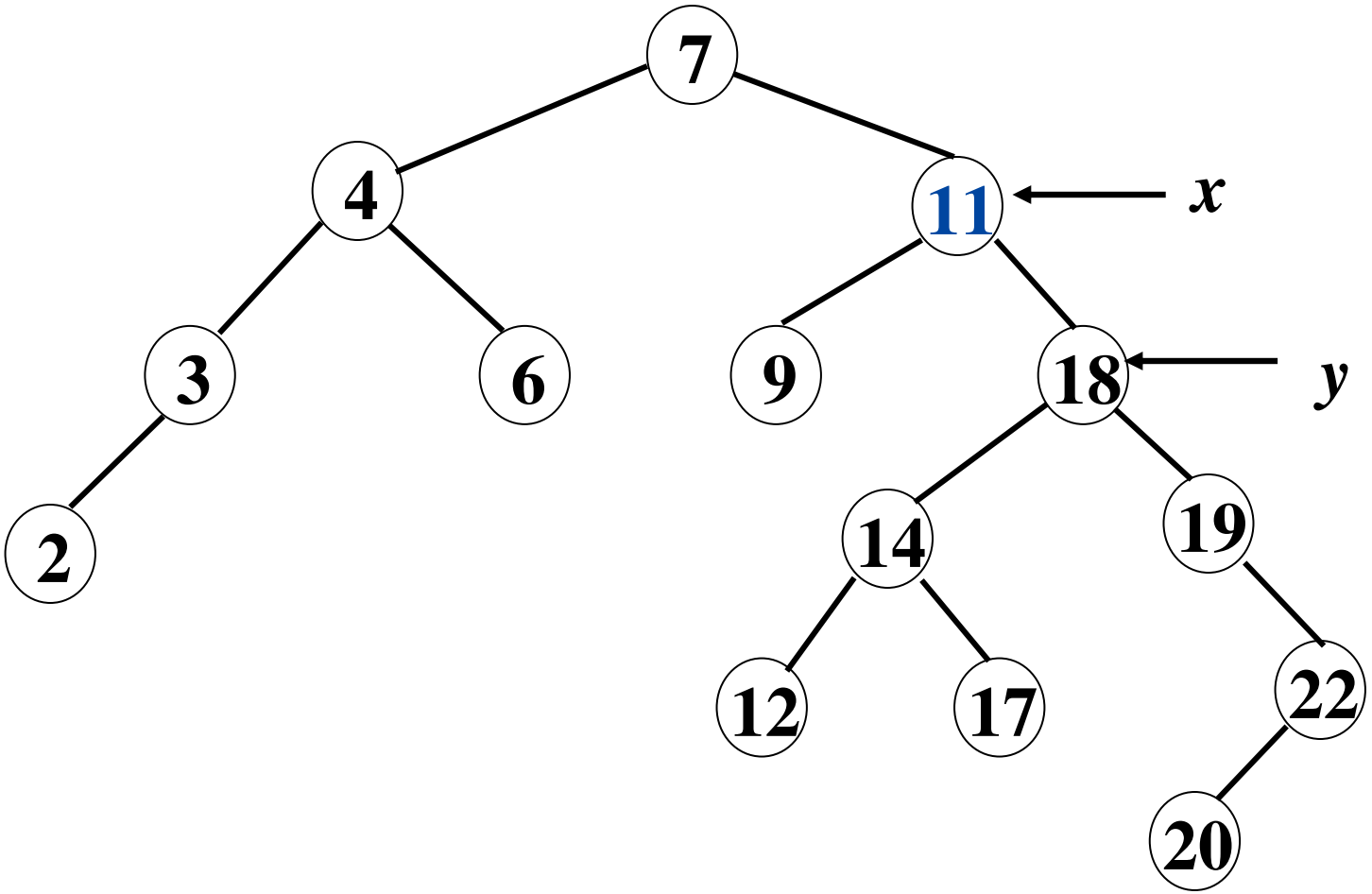


Rotacja rozwiązuje konflikt!

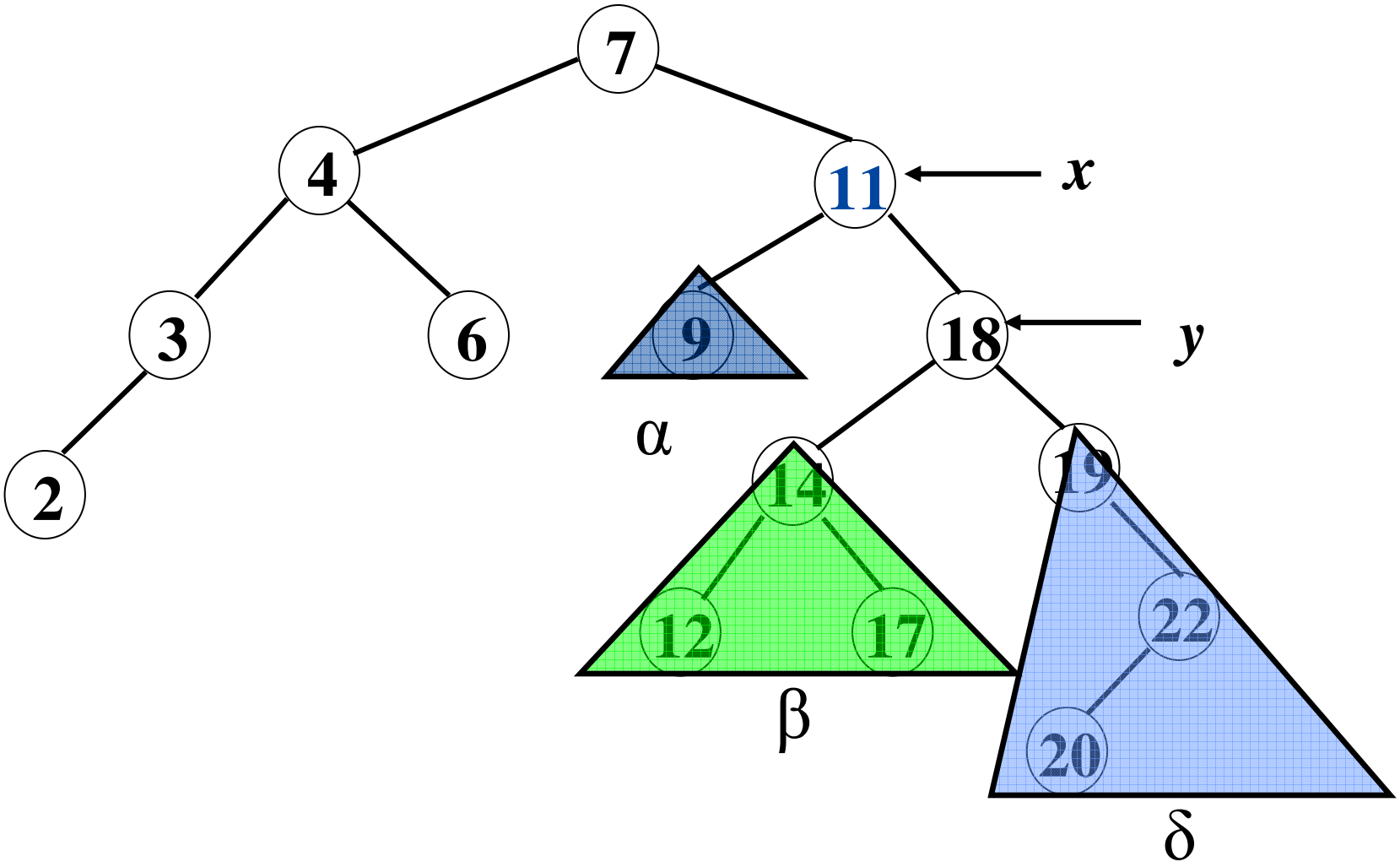
Left-Rotate


Left-Rotate(T,x)
y ← right[x]
right[x] ← left[y]
parent[left[y]] ← x
parent[y] ← parent[x]
if parent[x] = null
then root[T] ← y
else if x = left[parent[x]] then left[parent[x]] ← y
else right[parent[x]] ← y
left[y] ← x
parent[x] ← y
Inicjuj y
Zamień lewe poddrzewo y na prawe poddrzewo x
Przyłącz ojca x jako ojca y
Przyłącz x jako lewego syna y

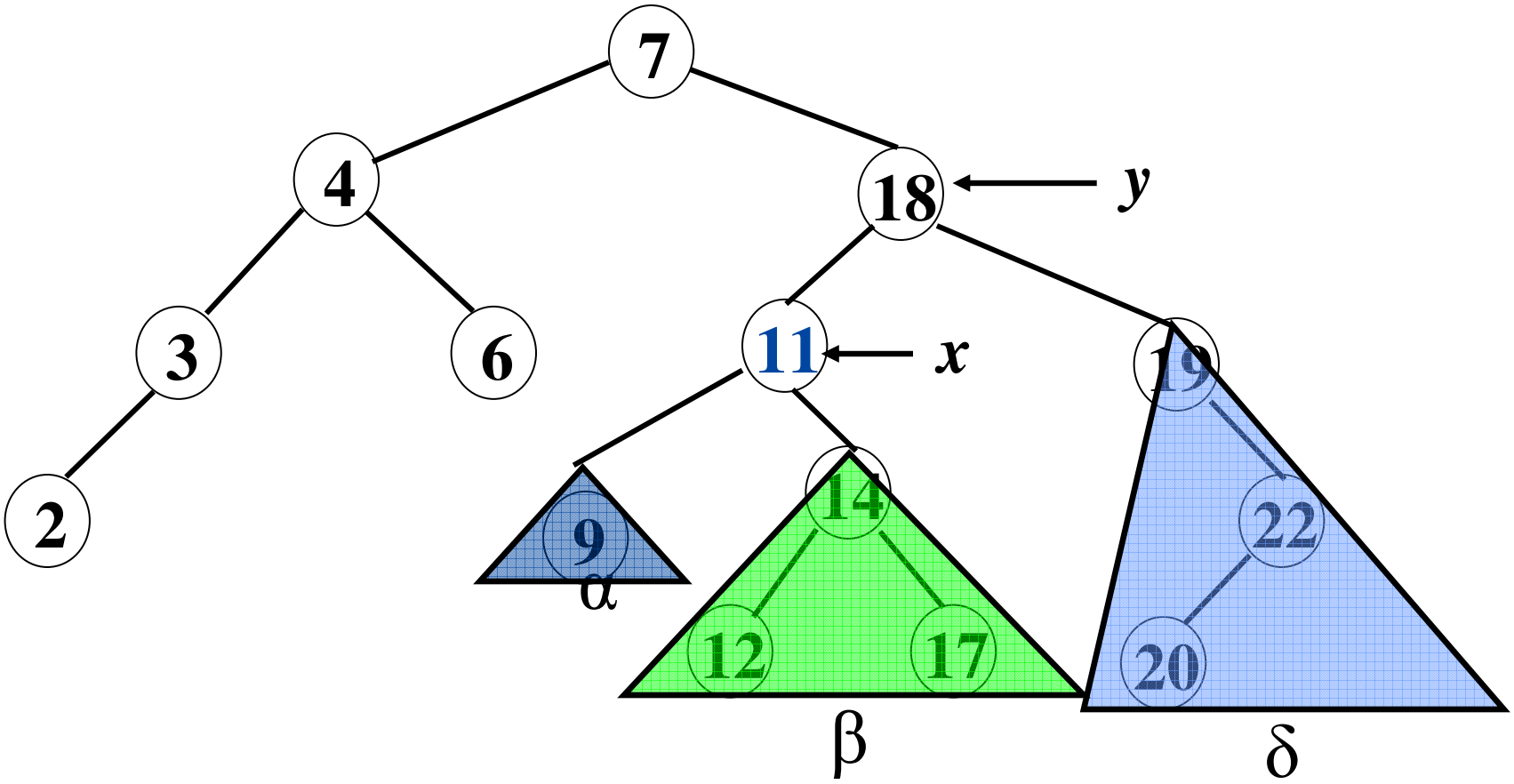
Przykład: Left-Rotate (1)



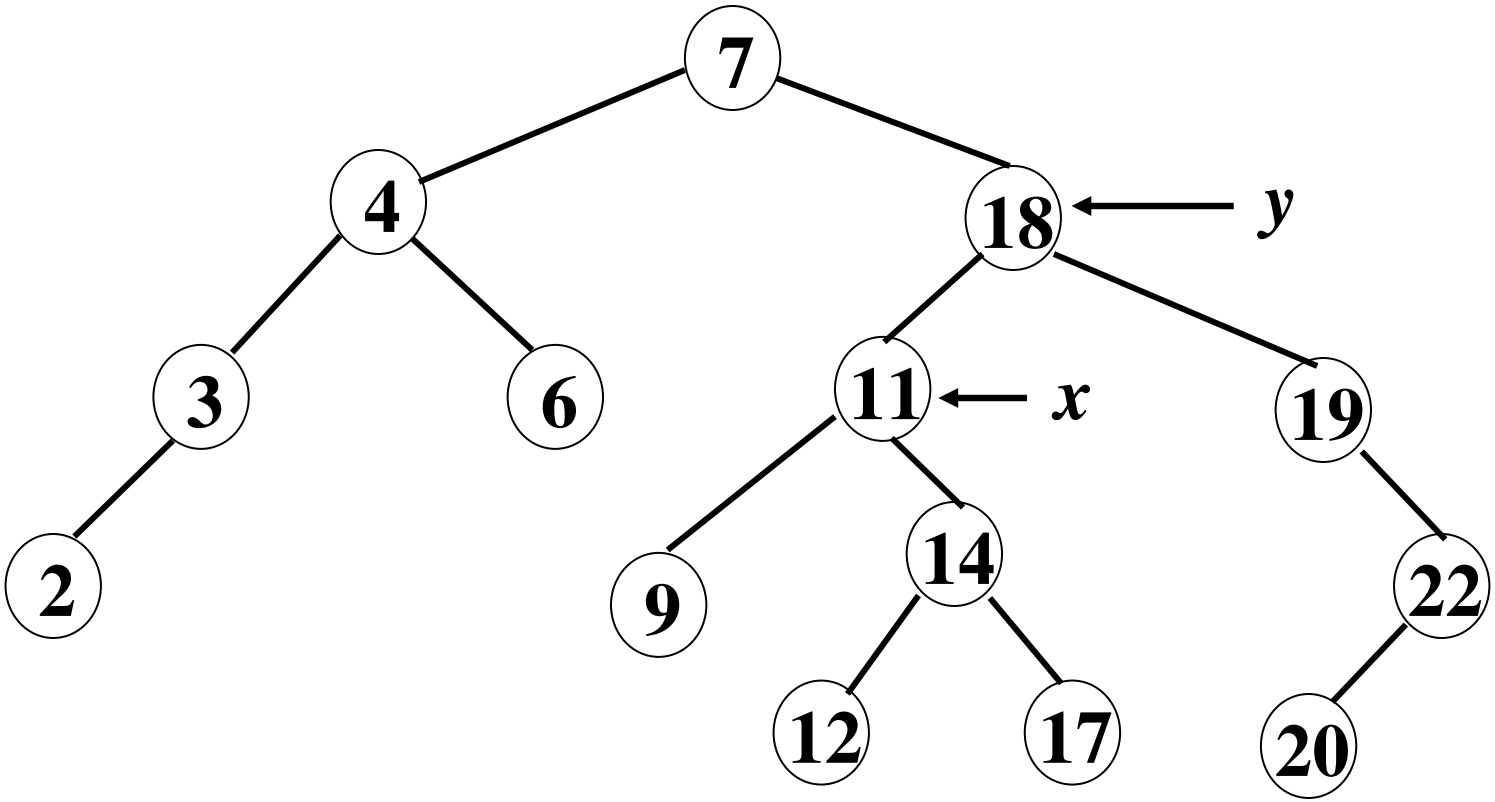
Przykład: Left-Rotate (2)



Przykład: Left-Rotate (3)



Przykład: Left-Rotate (4)



Rotacje

- **Zachowują własność drzewa BST.**
- **Zajmują stały czas $O(1)$ – stała ilość operacji na wskaźnikach.**
- **Rotacje w lewo i w prawo są symetryczne.**

Operacja wstawiania (INSERT) (1)

- **Nowy węzeł wstawiamy tak, jak do zwykłego drzewa BST i kolorujemy na czerwono.**
- **Jeśli własności drzewa RB zostały zaburzone, to naprawiamy drzewo poprzez zamianę kolorów i odpowiednie rotacje.**
- **Które z własności mogą zostać zaburzone?**

1. Każdy węzeł jest czerwony lub czarny → OK
2. Korzeń jest czarny → ?
3. Każdy liść (*null)* jest czarny → OK
4. Obydwaj potomkowie czerwonego węzła są czarni → ?
5. Każda ścieżka od węzła do liścia ma tę samą ilość czarnych węzłów → OK

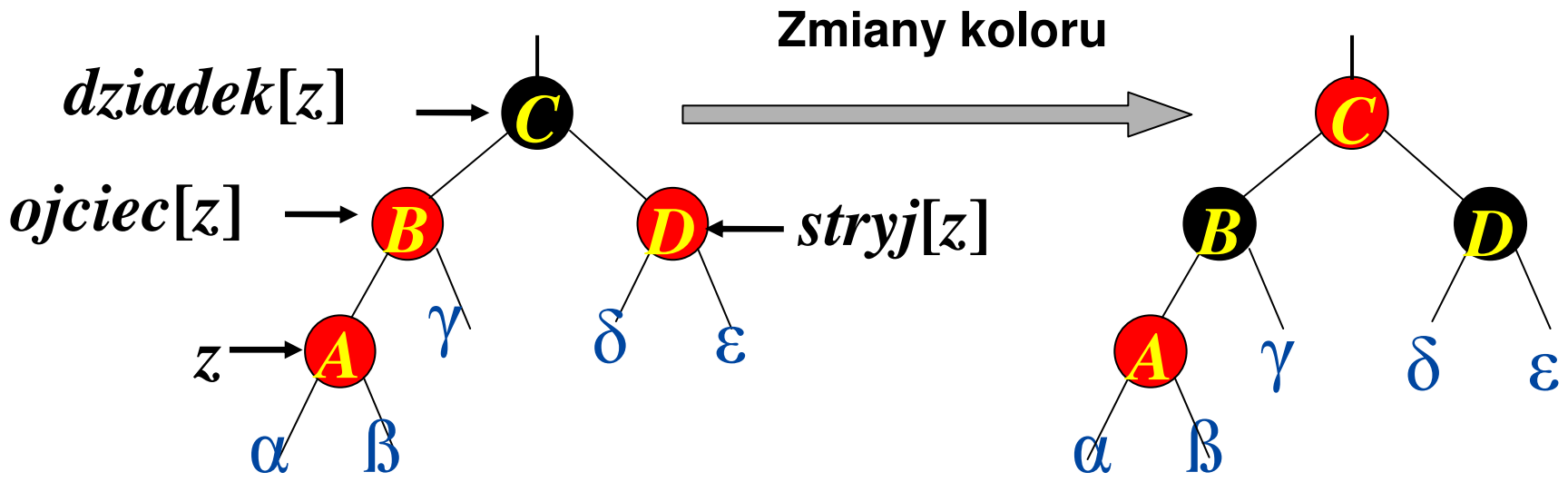
Operacja wstawiania (INSERT) (2)

- **Zaburzenia i metody ich naprawy:**
  - 2. jeśli wstawiany węzeł *x* jest korzeniem przekolorujemy go na czarno → OK
  - 4. co zrobić, kiedy ojciec jest również czerwony?

    Rozpatrzymy trzy przypadki dla wstawianego węzła:
    - przypadek 1: „stryj” wstawionego węzła jest też czerwony
    - przypadek 2: „stryj” jest czarny, a wstawiany węzeł jest prawym potomkiem
    - przypadek 3: „stryj” jest czarny, a wstawiany węzeł jest lewym potomkiem

przypadek 1: „stryj” wstawionego węzła jest też czerwony

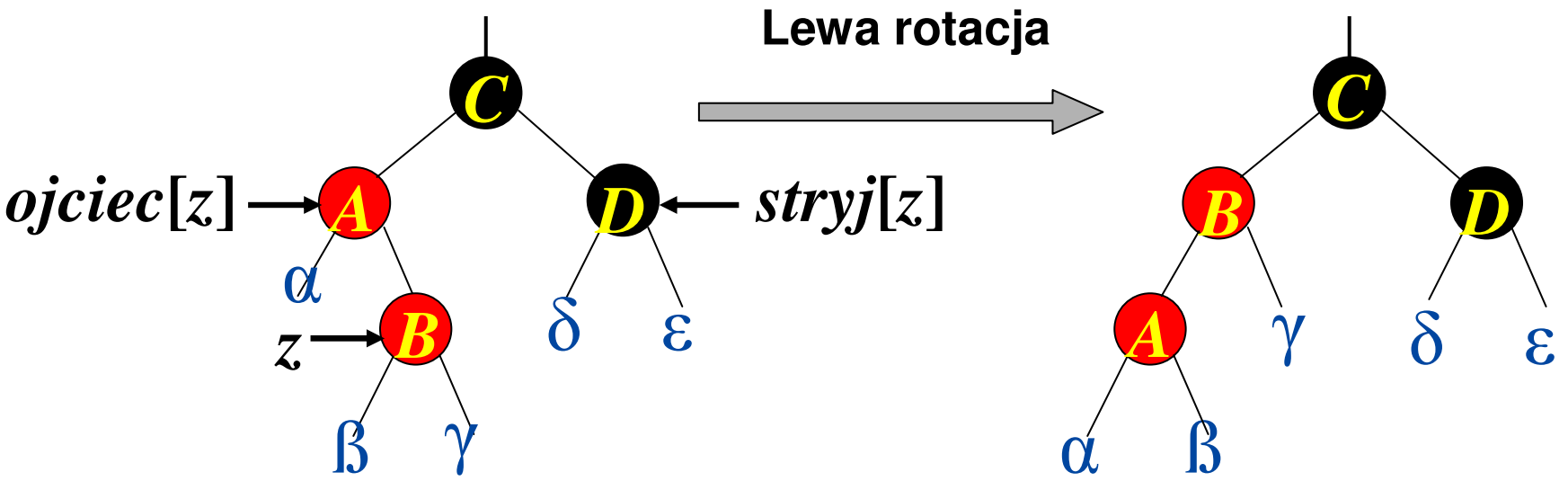
- Jeżeli *z* ma zarówno czerwonego ojca (*B*) i stryja (*D*), zmieniamy kolor ojca i stryja na czarny, a dziadka (*C*) na czerwony:



- jeżeli *C* jest nowym korzeniem – kolorujemy go na czarno
- Jeżeli dziadek *C* dalej zaburza własności drzewa – naprawiamy dalej drzewo (w *C*)

przypadek 2: „stryj" wstawionego węzła jest czarny, a z jest prawym potomkiem

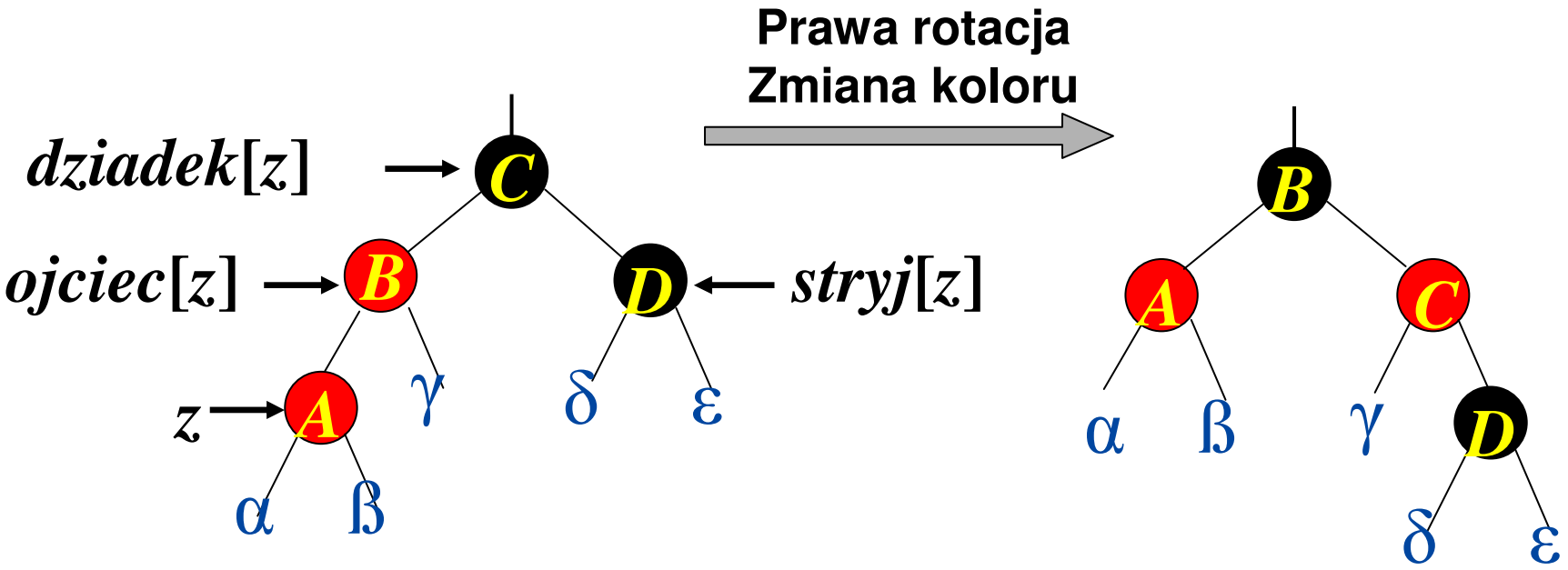
- **Jeżeli *z* jest prawym potomkiem czerwonego węzła *A* i ma czarnego stryja *D*, przeprowadzamy rotację lewą dla *A*:**



- **to doprowadza do sytuacji opisanej przez przypadek 3**

przypadek 3: „stryj" wstawionego węzła jest czarny, a z jest lewym potomkiem

- **Jeżeli *z* jest lewym potomkiem czerwonego ojca *B* i ma czarnego stryja *D*, przeprowadzamy prawą rotację w C i zmieniamy kolor *B* i *C*:**



Po takiej operacji nie ma już zaburzenia własności drzewa RB!

RB: wstawianie

- **Aby wstawić nowy węzeł *z* do drzewa RB, należy:**
  1. Wstawić nowy węzeł *z* nie zważając na kolory.
  2. Kolorujemy *z* na czerwono.
  3. Naprawiamy otrzymane drzewo stosując, w razie potrzeby, przypadki 1,2 i 3
- **Złożoność czasowa tej operacji wynosi $O(\lg n)$**

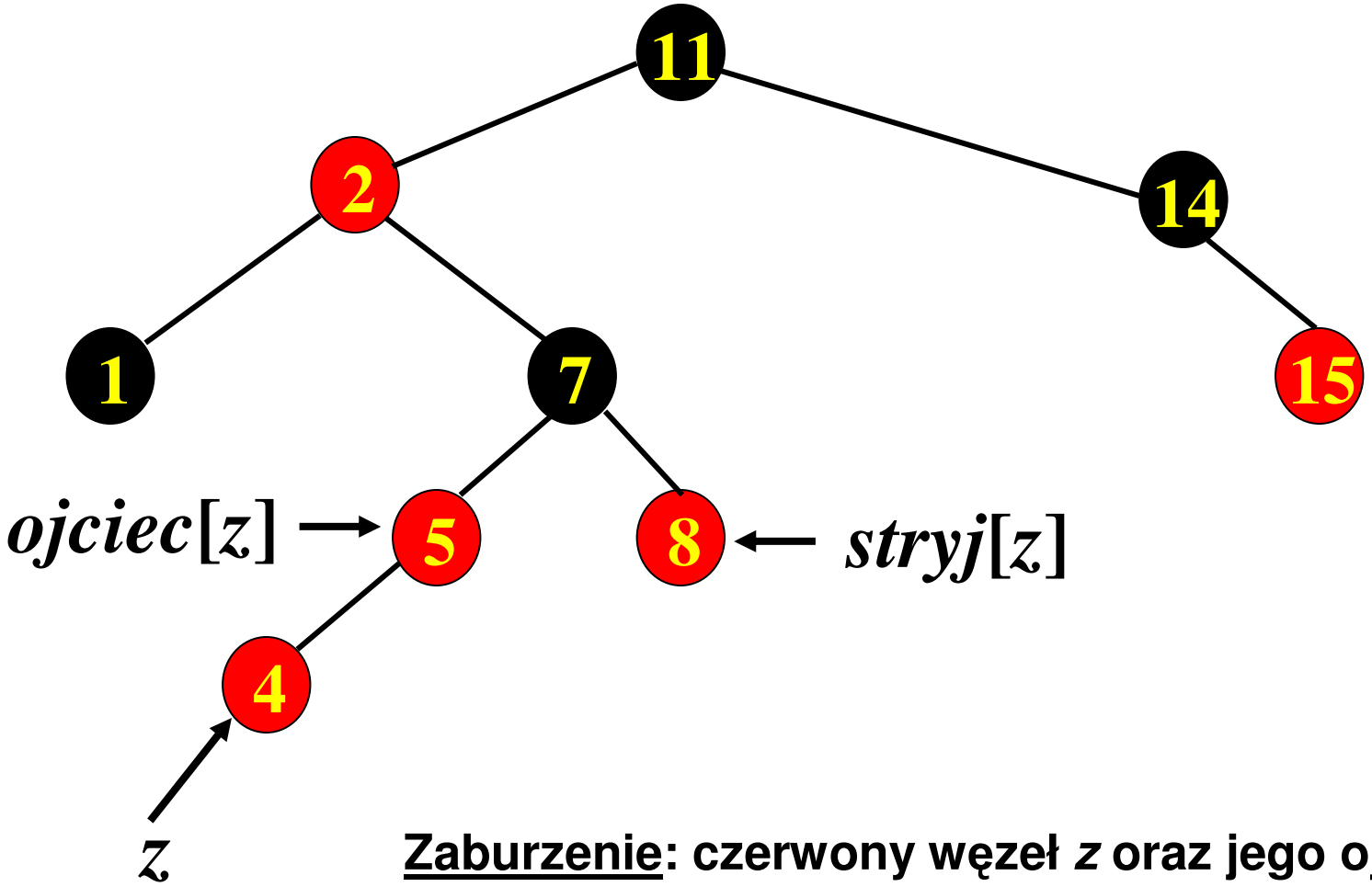
RB: Insert-Fixup (pseudokod)

```
RB-Insert-Fixup(T,z)
    while color[parent[z]] = "red"
     do y ← z's uncle
        if color[y] = "red" then do Case 1
        else do
            if z = right[parent[z]] then do Case 2
            do Case 3
  color[root[T]] ← "black"
```

RB: Insert-Fixup

1. **Węzeł wstawiany jest czerwony**
2. **Jeśli *parent*[*z*] jest korzeniem drzewa, kolorujemy do na czarno**
3. **Jeżeli własności drzewa zostały zaburzone to tylko jedna własność może być niespełniona i jest to albo wł. 2, albo wł. 4.**
   - Własność 2: korzeń drzewa jest czerwony
   - Własność 4: zarówno *z,* jak i jego ojciec (*parent*[*z*]) są czerwoni

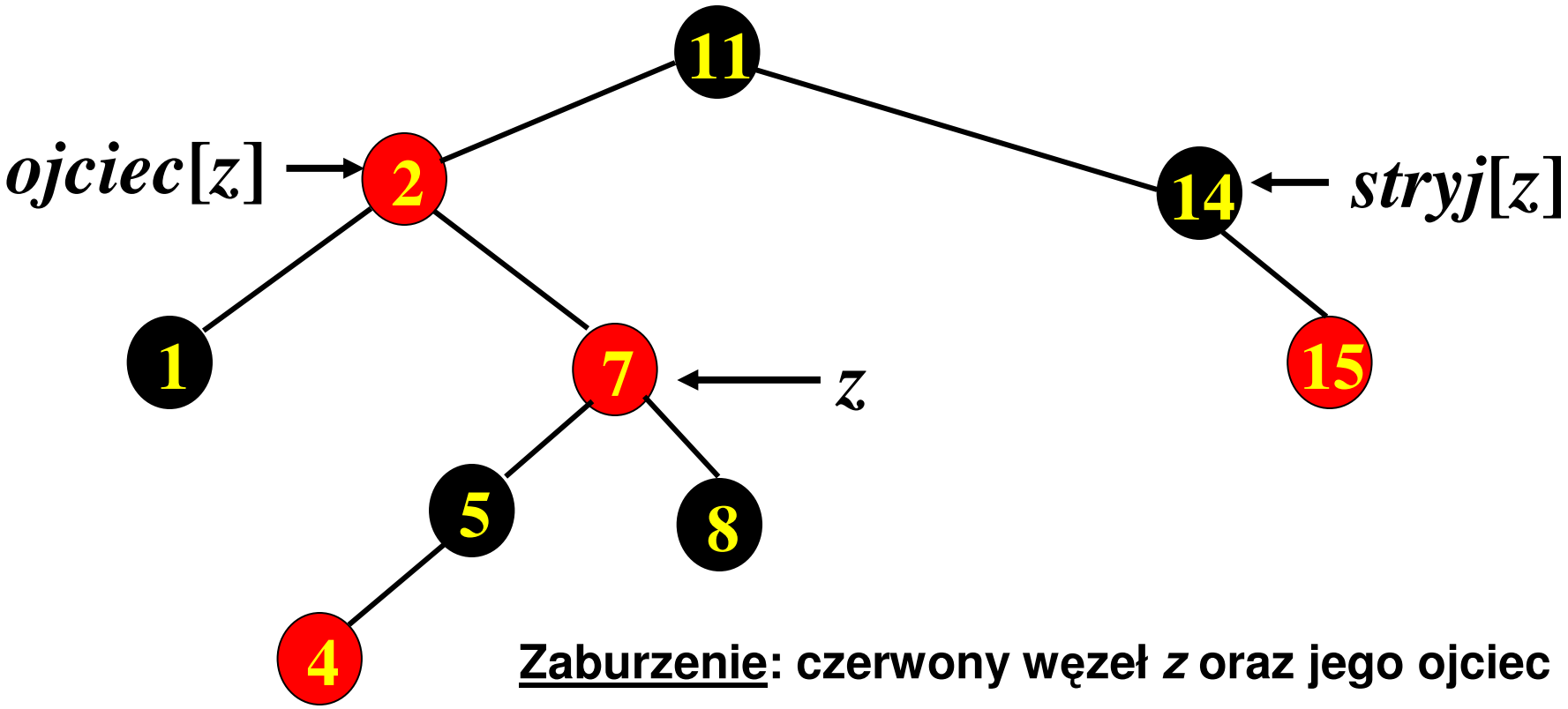
Przykład (1)



Zaburzenie: czerwony węzeł *z* oraz jego ojciec

Przyp. 1: stryj *z* jest czerwony → zmieniamy kolor

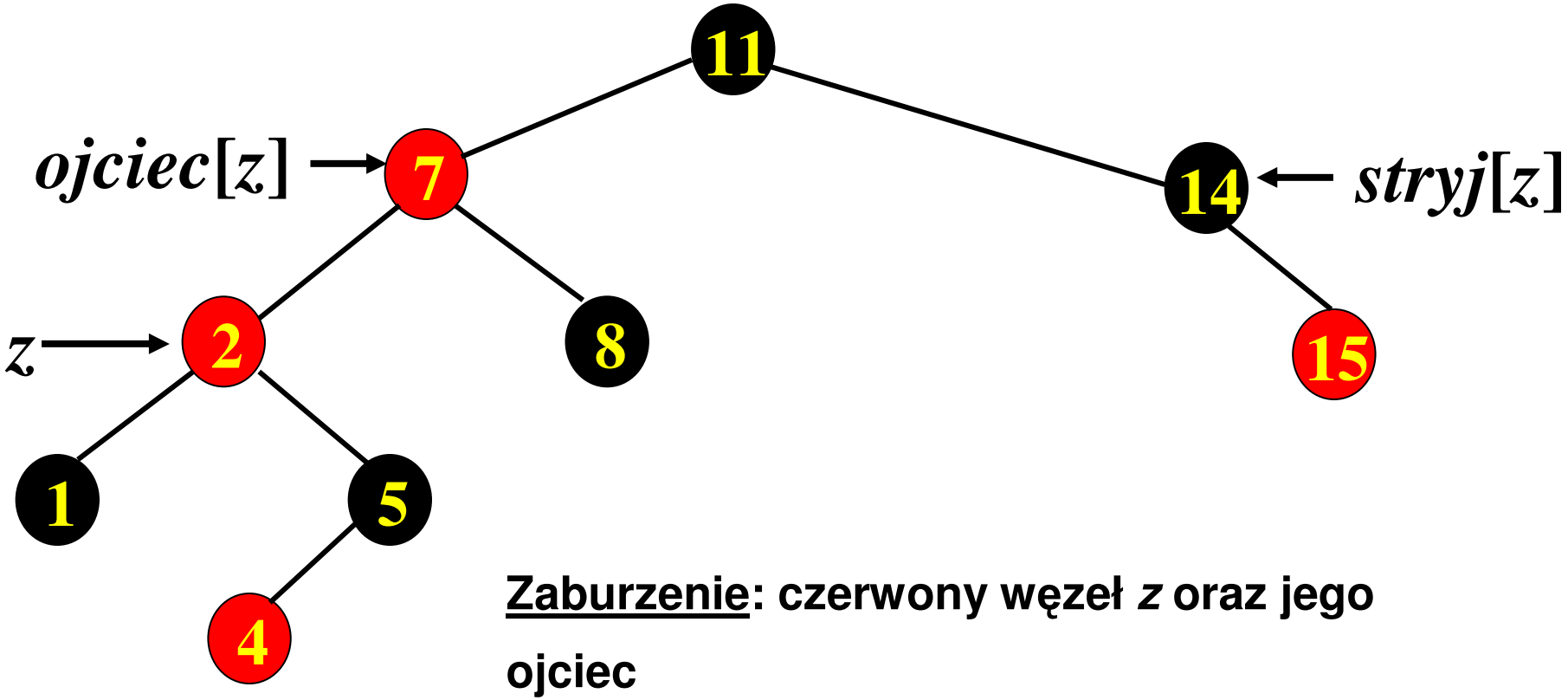
Przykład (2)



Zaburzenie: czerwony węzeł *z* oraz jego ojciec

Przyp. 2: stryj *z* jest czarny i *z* jest prawym potomkiem → rotacja w lewo

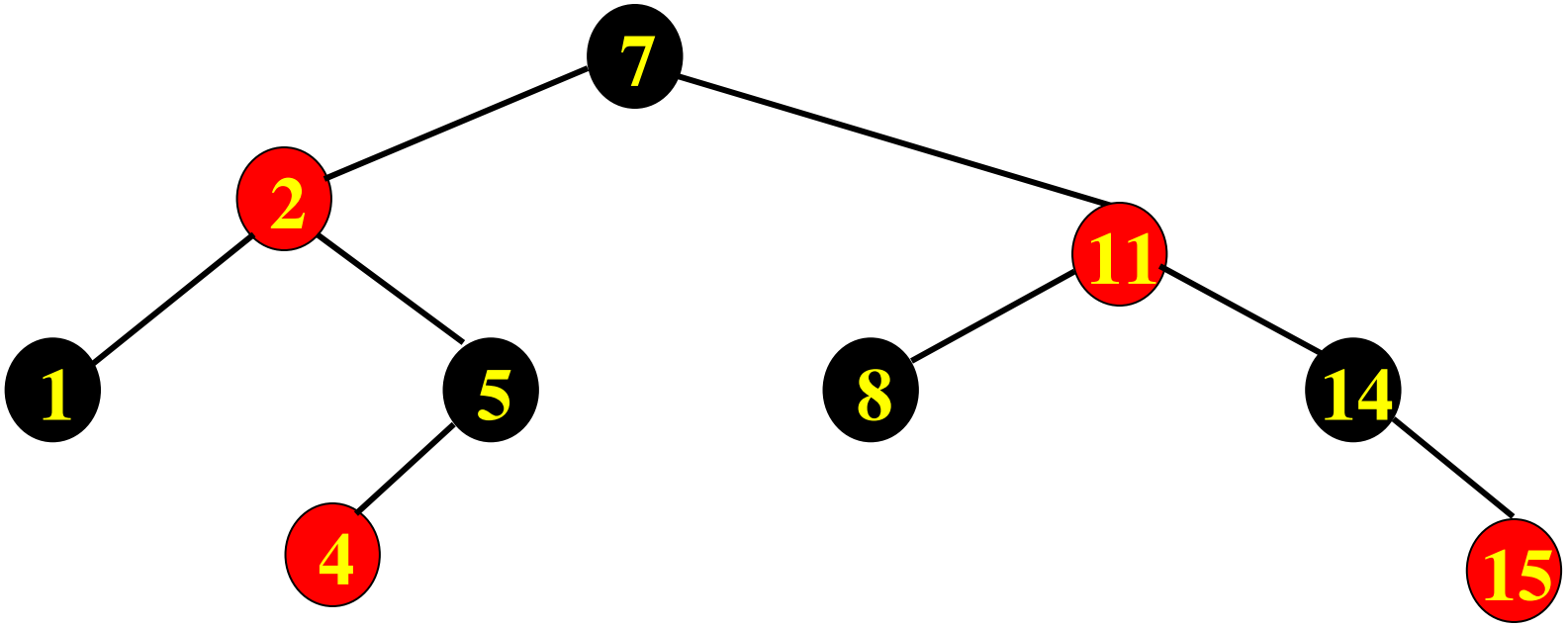
Przykład (3)



<u>Zaburzenie</u>: czerwony węzeł *z* oraz jego ojciec

<u>Przyp. 3</u>: stryj *z* jest czarny i *z* jest lewym potomkiem potomkiem → rotacja w prawo + zmiana koloru

Przykład (4)



Dostajemy poprawne drzewo RB

Nie ma potrzeby dalszych działań!

RB: usuwanie (1)

- **Korzystamy ze standardowej operacji usuwania dla BST.**
- **Jeżeli zaburzymy własności drzewa RB, to naprawiamy je wykorzystując rotacje i zmiany kolorów węzłów.**
- **Jakie własności drzewa mogą zostać zaburzone?**

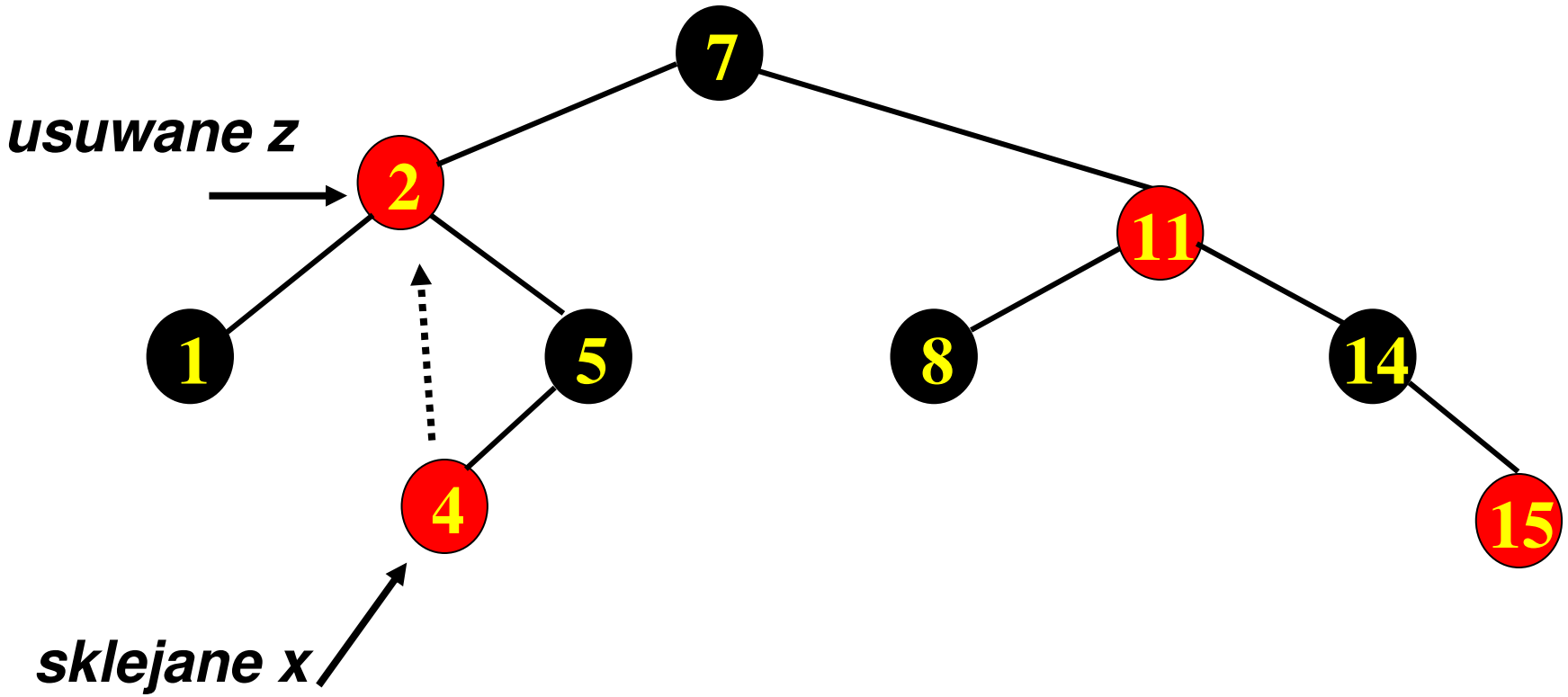
1. Każdy węzeł ma kolor czarny lub czerwony → OK
2. Korzeń jest czarny → ?
3. Każdy liść (*null*) jest czarny → OK
4. Obaj potomkowie czerwonego węzła są czarni → ?
5. Każde ścieżka od wybranego węzła do liścia zawiera tyle samo węzłów czarnych → ?

RB: usuwanie (2)

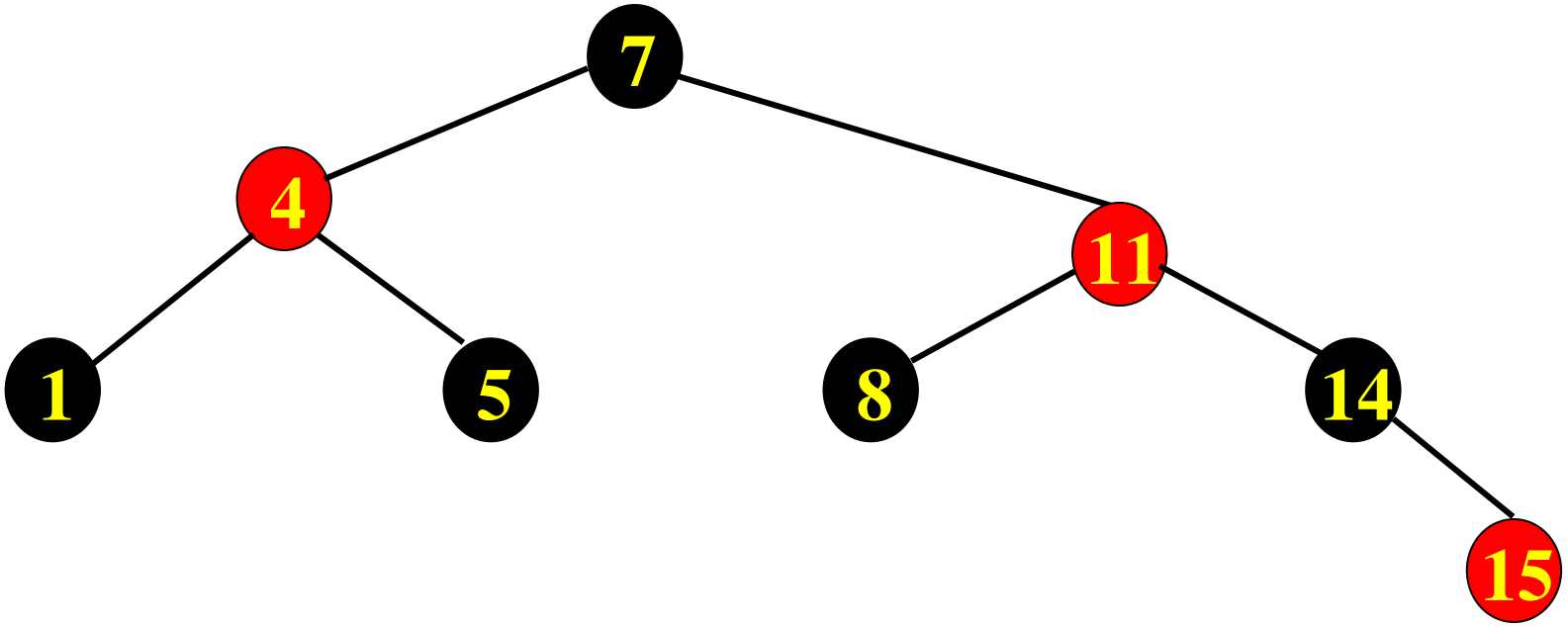
- **Zaburzenia:**
  - Mogą zostać zaburzone własności 2, 4, 5.

- **Mamy cztery przypadki:**
  - Przypadek 1: brat *w* węzła *x* jest czerwony (x – „sklejony” węzeł)
  - Przypadek 2: brat *w* węzła *x* jest czarny, podobnie jak obaj potomkowie *w*
  - Przypadek 3: brat *w* węzła *x* jest czarny, lewy potomek *w* jest czerwony, prawy - czarny
  - Przypadek 4: brat *w* węzła *x* jest czarny, obaj potomkowie *w* są czerwoni.

Przykład (1)



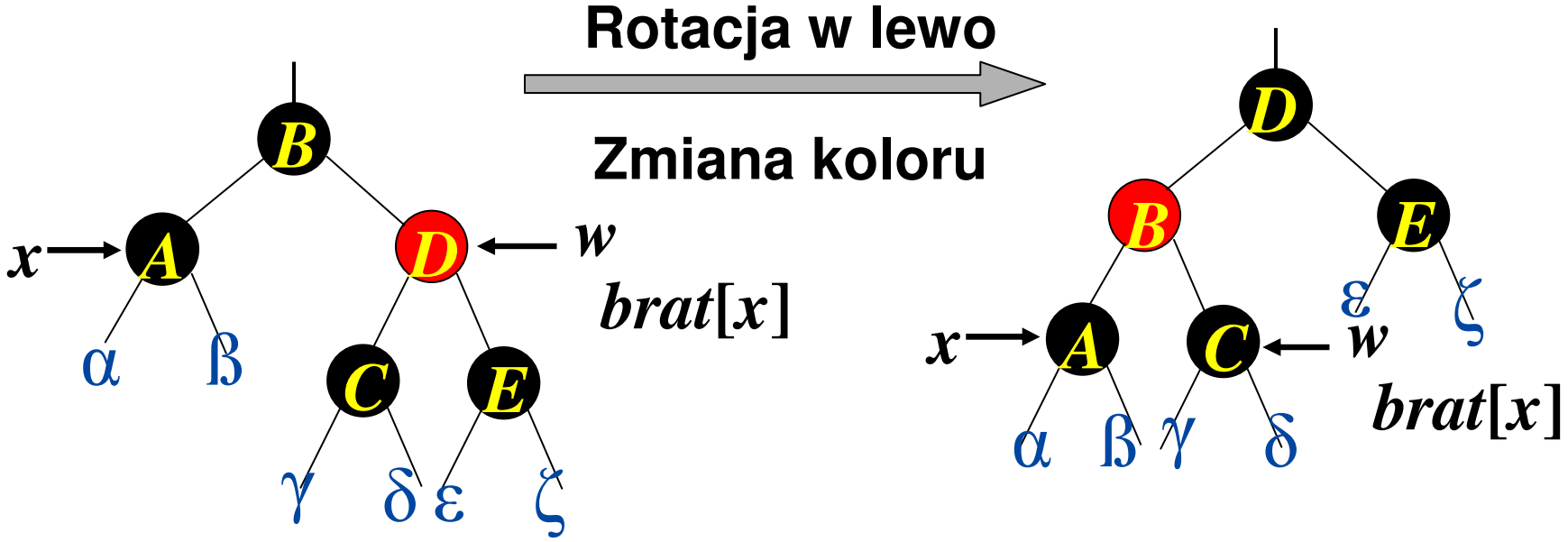
Przykład (2)



Nie ma zaburzeń !

Przypadek 1: brat x jest czerwony

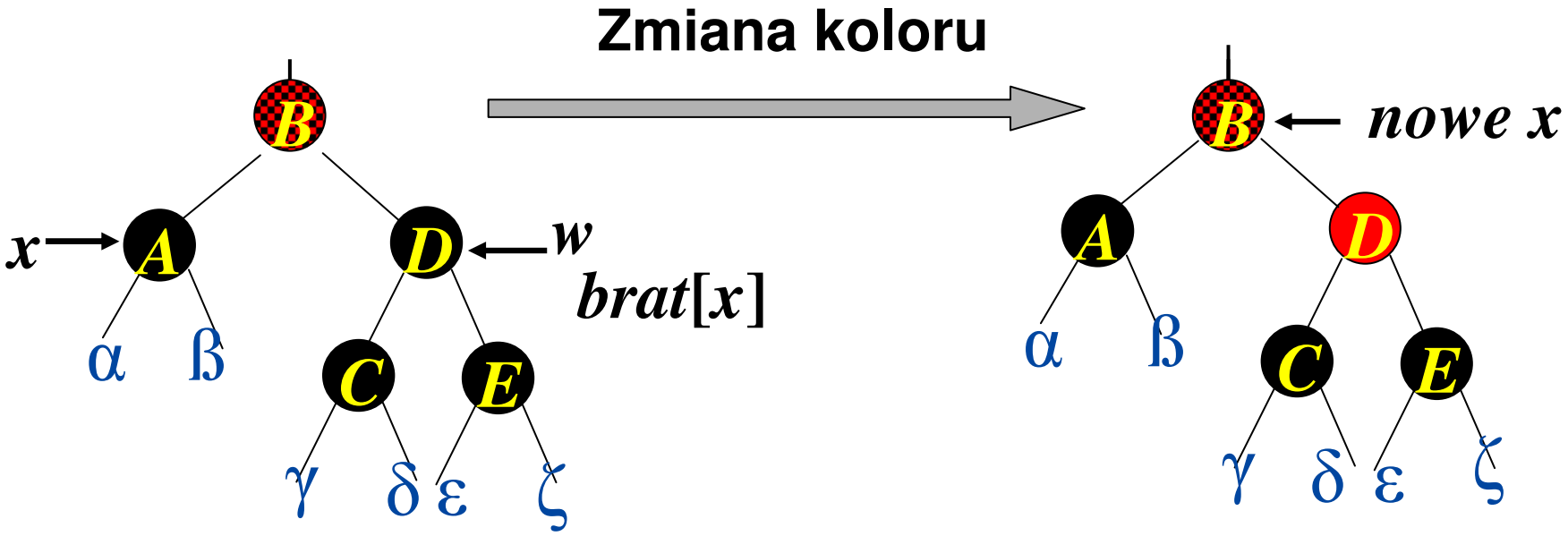
- **Przypadek 1 przekształca się w jeden z pozostałych przypadków poprzez zamianę kolorów węzłów *B* i *D* oraz zastosowanie lewej rotacji:**



Nie ma zmiany czarnej wysokości!

Przypadek 2: brat x i obaj jego potomkowie są czarni

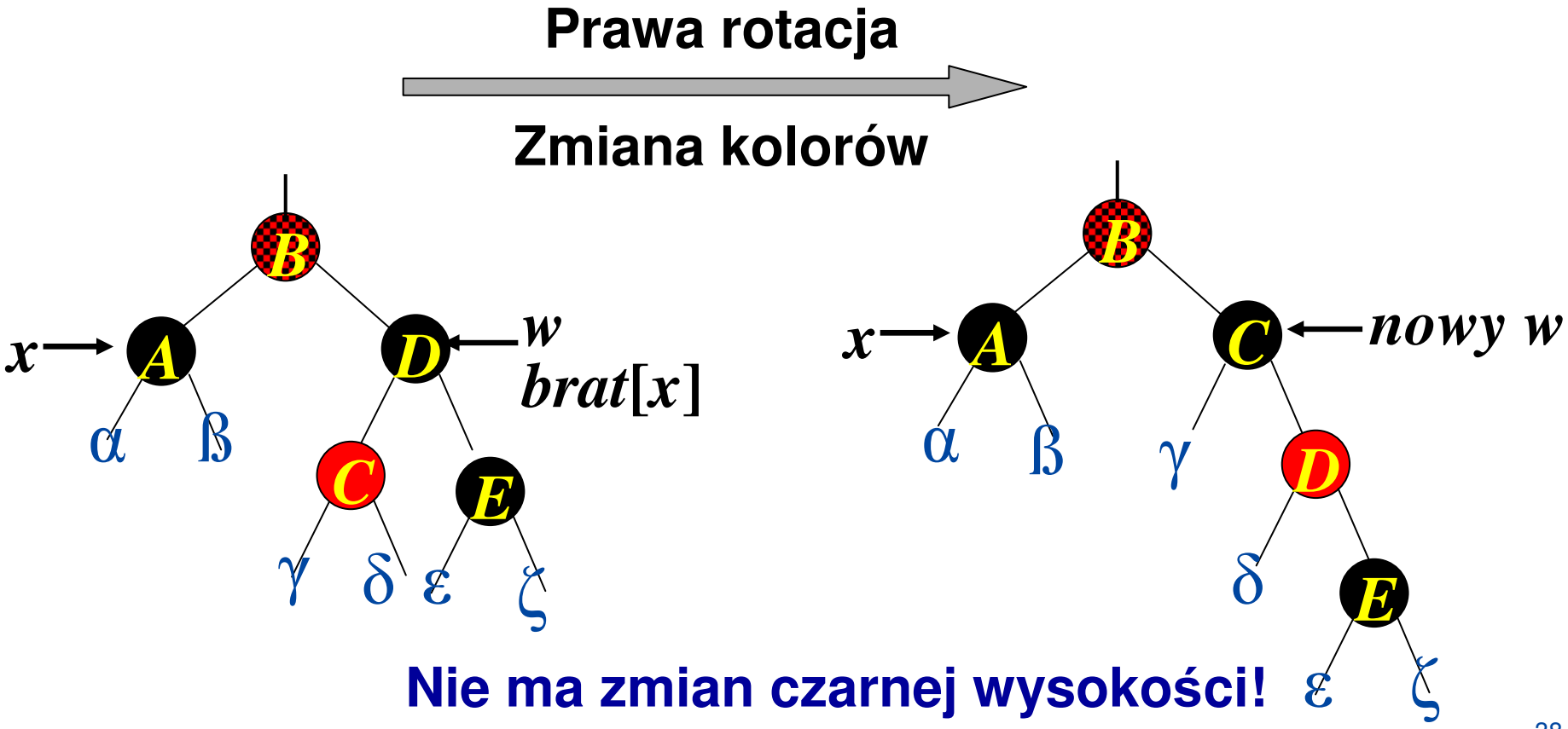
- **Przypadek 2 – przenosimy problem o jeden poziom wyżej zmieniając kolor *D* na czerwony:**



Zmniejszamy o 1 czarną wysokość potomków D!

Przypadek 3: brat x jest czarny, a jego potomkowie czerwony i czarny

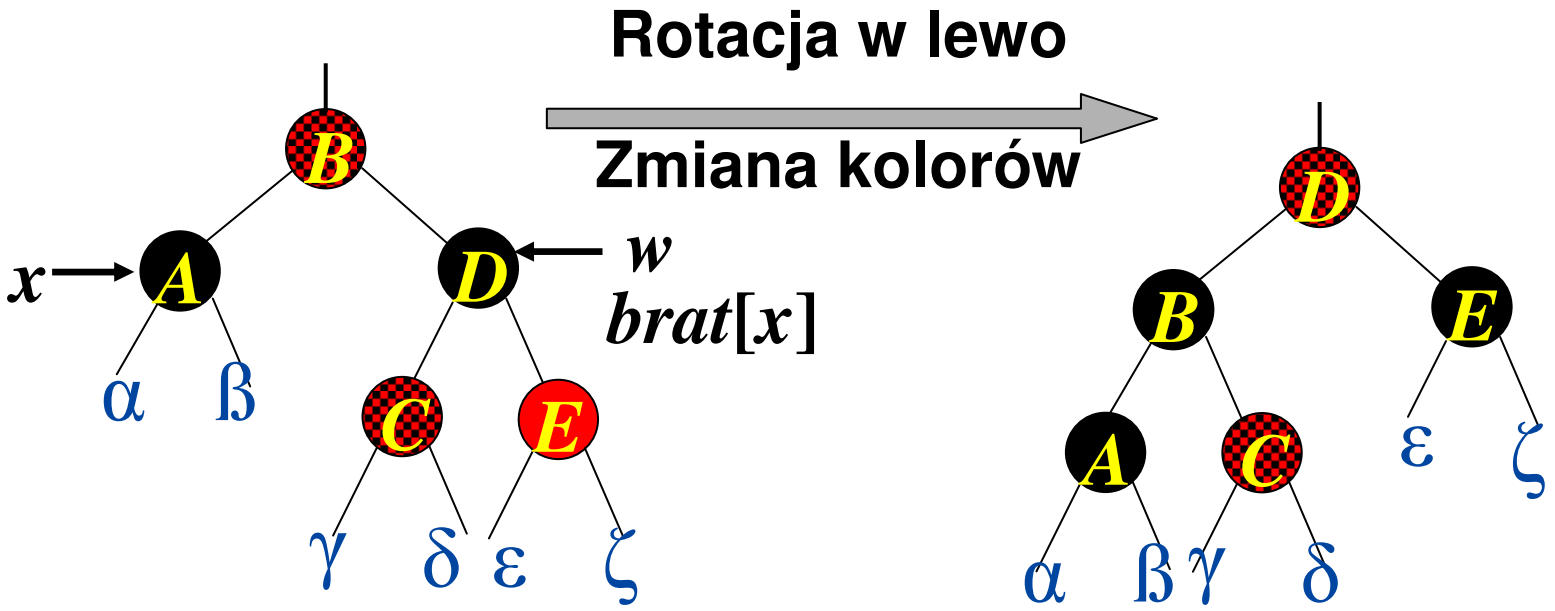
- **Przypadek ten przechodzi w przypadek 4 przez zamianę koloru węzłów C i D oraz prawą rotacje:**



Nie ma zmian czarnej wysokości!

Przypadek 4: brat x jest czarny, a jego prawy potomek czerwony

- **Zaburzenie można usunąć poprzez zamianę kolorów oraz rotację w lewo:**



Czarna wysokość węzłów poniżej A zwiększa się o 1!

RB-Delete

- **Aby usunąć węzeł *x* z drzewa RB należy:**
  1. Usuwamy węzeł *x* tak, jak z drzewa BST nie zważając na kolory.
  2. Naprawiamy drzewo (jeśli jest taka potrzeba) stosując przypadki 1, 2, 3, i 4, aż do naprawienia drzewa
- **Złożoność takiej operacji wynosi *O*(lg *n*)**

RB-Delete-Fixup (pseudokod)

RB-Delete-Fixup(*T*, *x*)

while ***x*** ≠ ***root*[*T*]** and ***color*[*x*] = "black"**

do if ***x* = *left*[*parent*[*x*]]**

then ***w* ← *x*'s brother**

if ***color*[*w*] = "red"** then do **Case 1**

if ***color*[*w*] = "black"** and ***color[right[w]]= "black"*** and ***color[left[w]]= "black"***

then do **Case 2.**

else if ***color*[*w*] = "black"** and ***color*[*right*[*w*]] = "black"**

then do **Case 3**

if ***color*[*w*] = "black"** and ***color*[*right*[*w*]] = "red"**

then do **Case 4**

else **to samo dla *x* = *right*[*parent*[*x*]]**

***color*[*x*] ← "black"**

po tym dostajemy przyp. 2

***x* przechodzi na *parent*[*x*]**

po tym dostajemy przyp. 4

po tym dostajemy *x* = *root*[*T*].

Złożoność RB-Delete

- **Jeśli zajdzie przyp. 2 w następstwie przyp. 1, to nie wchodzimy ponownie do pętli – ojciec *x* jest czerwony po wykonaniu przyp. 2.**
- **Jeśli wykonane zostają przyp. 3 lub 4 również nie wchodzimy ponownie do pętli.**
- **Jedyny sposób wielokrotnego wykonywania pętli to otrzymywanie przypadku 2 w następstwie przyp. 2. Stąd wchodzimy do pętli co najwyżej $O(h)$ razy.**
- **Daje to złożoność $O(\lg n)$.**

Drzewa RB - podsumowanie

- **Pięć prostych zasad kolorowania gwarantuje, że wysokość drzewa nie przekroczy $2(\lg n + 1) = O(\lg n)$**
- **Wstawianie i usuwanie odbywa się na takiej samej zasadzie jak dla zwykłych drzew BST**
- **Operacje te mogą zaburzać własności drzewa RB. Przywracanie tych własności odbywa się poprzez rotacje i przekolorowanie niektórych węzłów**
- **We wszystkich przypadkach wymagany jest czas co najwyżej $O(\lg n)$.**