FernUniversität in Hagen
Fakultät für Mathematik und Informatik

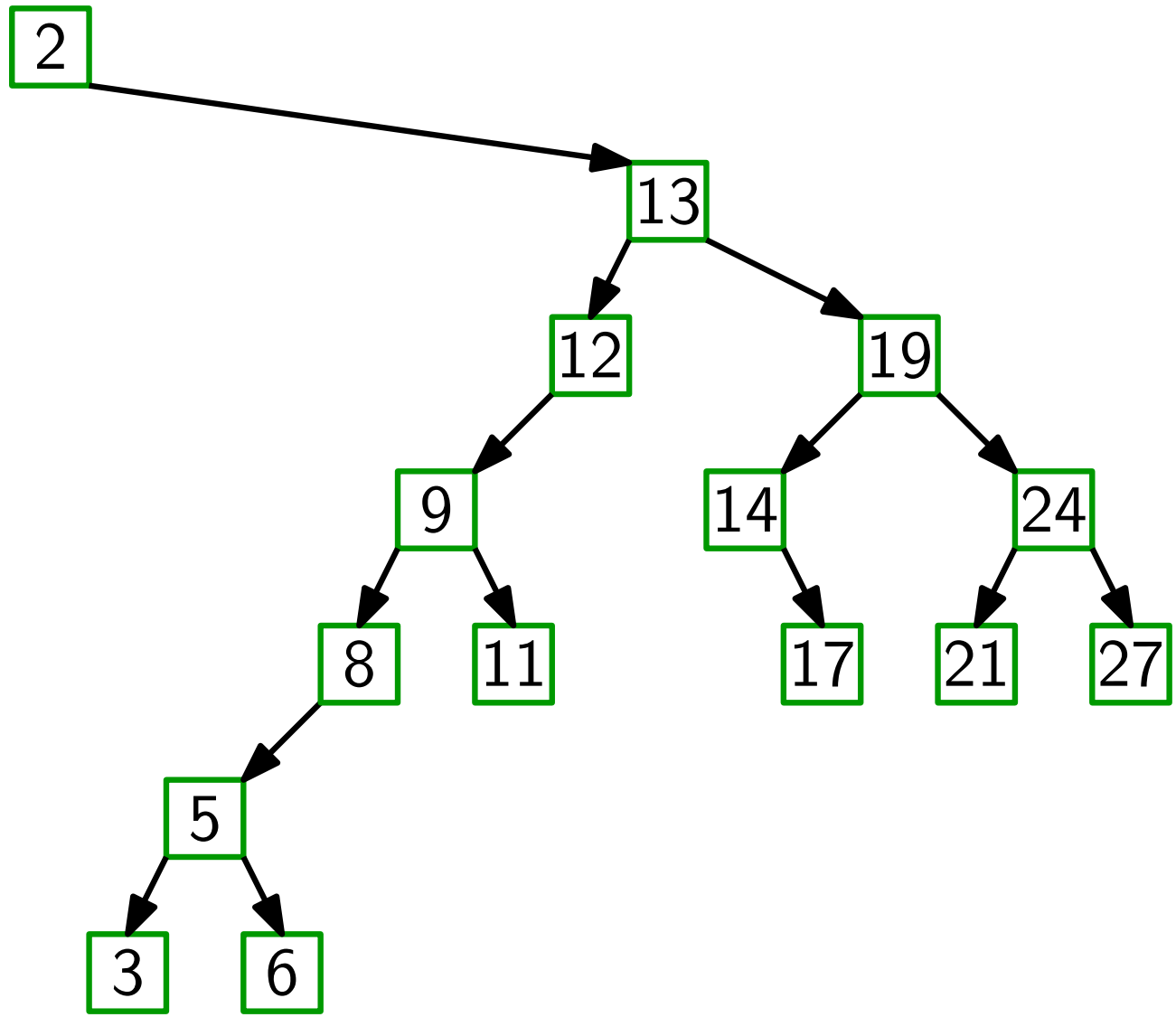
theoretische informatik

Dynamische Optimalität: Splay-Bäume

Dr. Philipp Kindermann
LG Theoretische Informatik
FernUniversität in Hagen

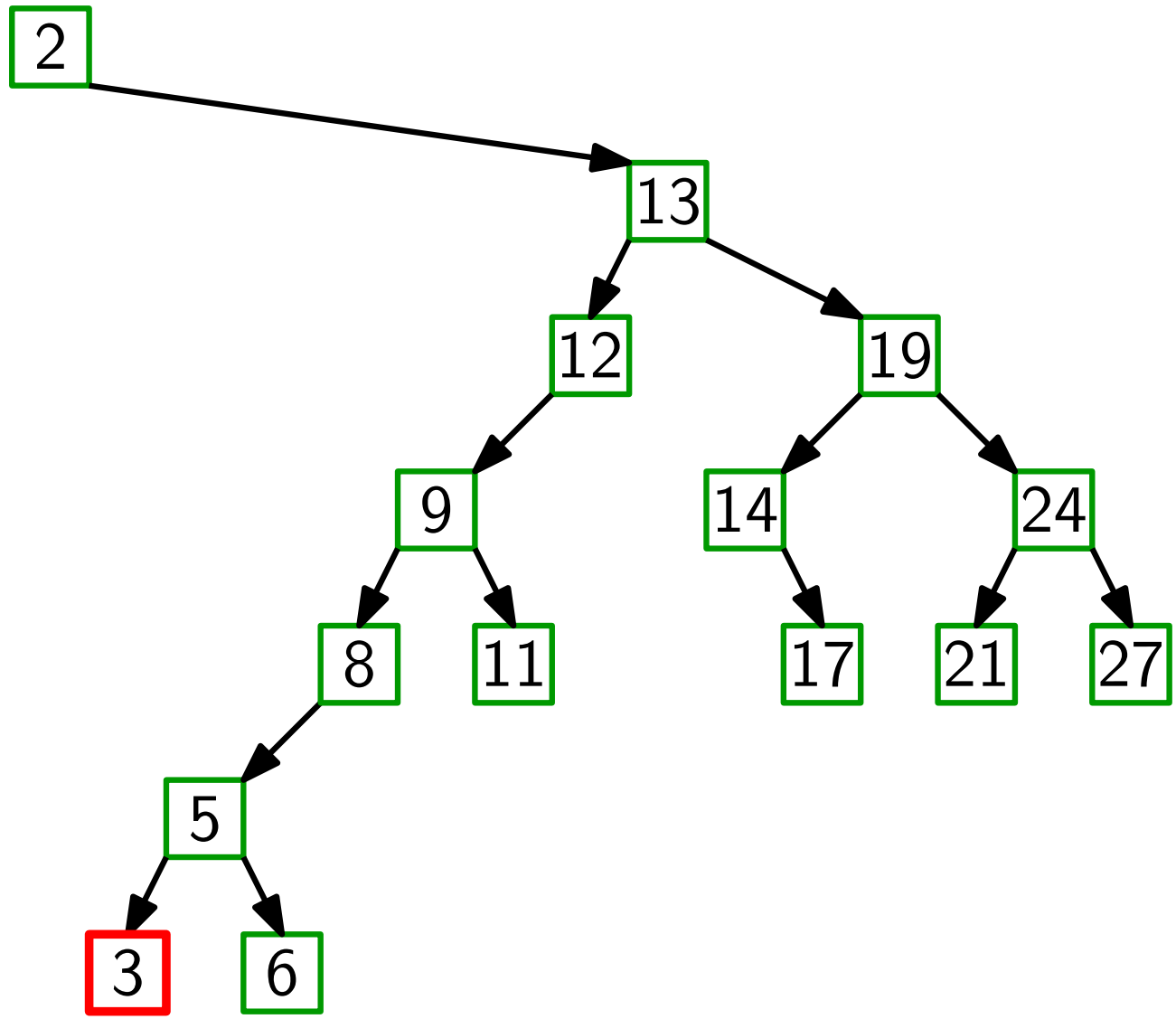
Binäre Suchbäume

Binärer Suchbaum:



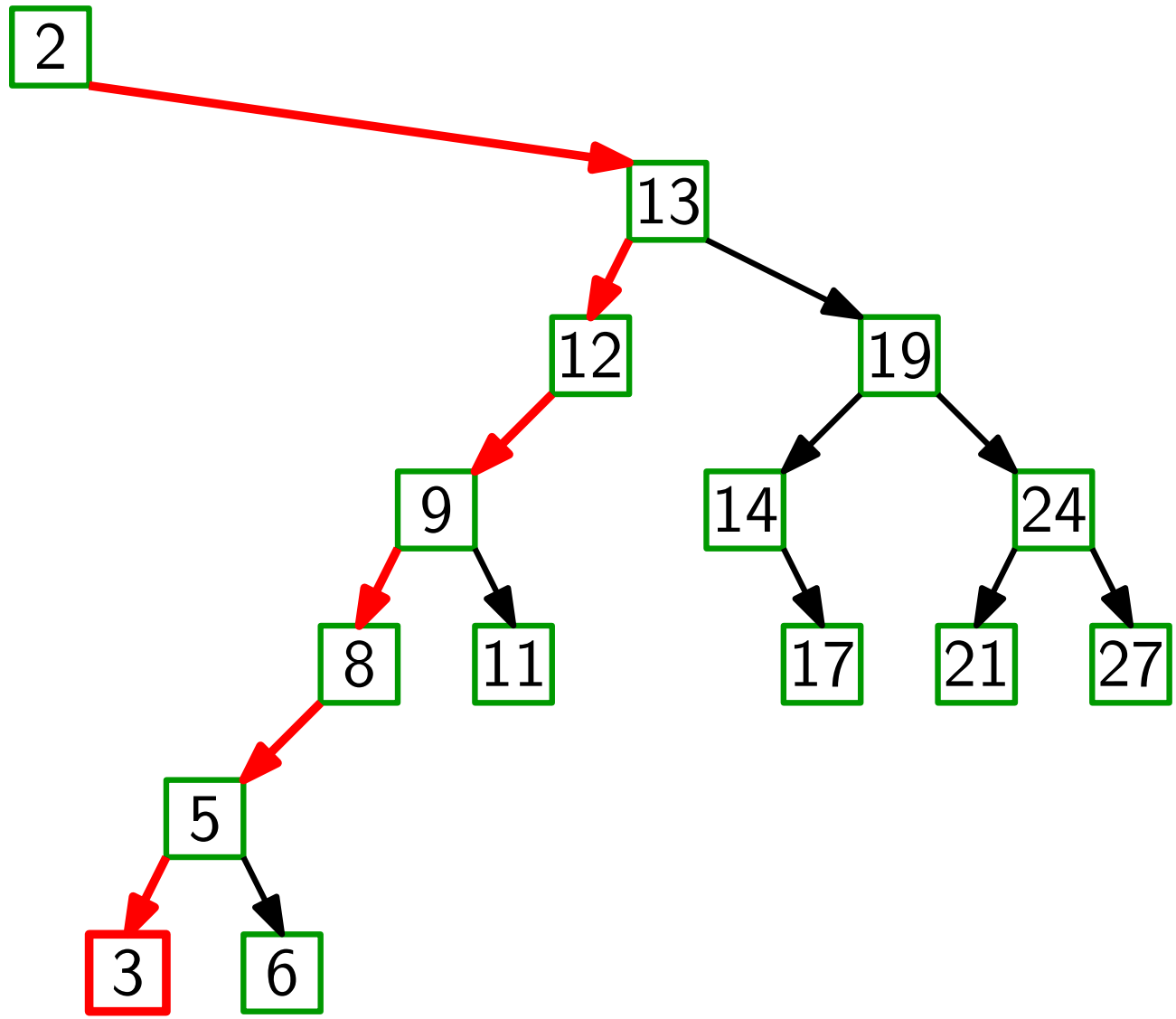
Binäre Suchbäume

Binärer Suchbaum:



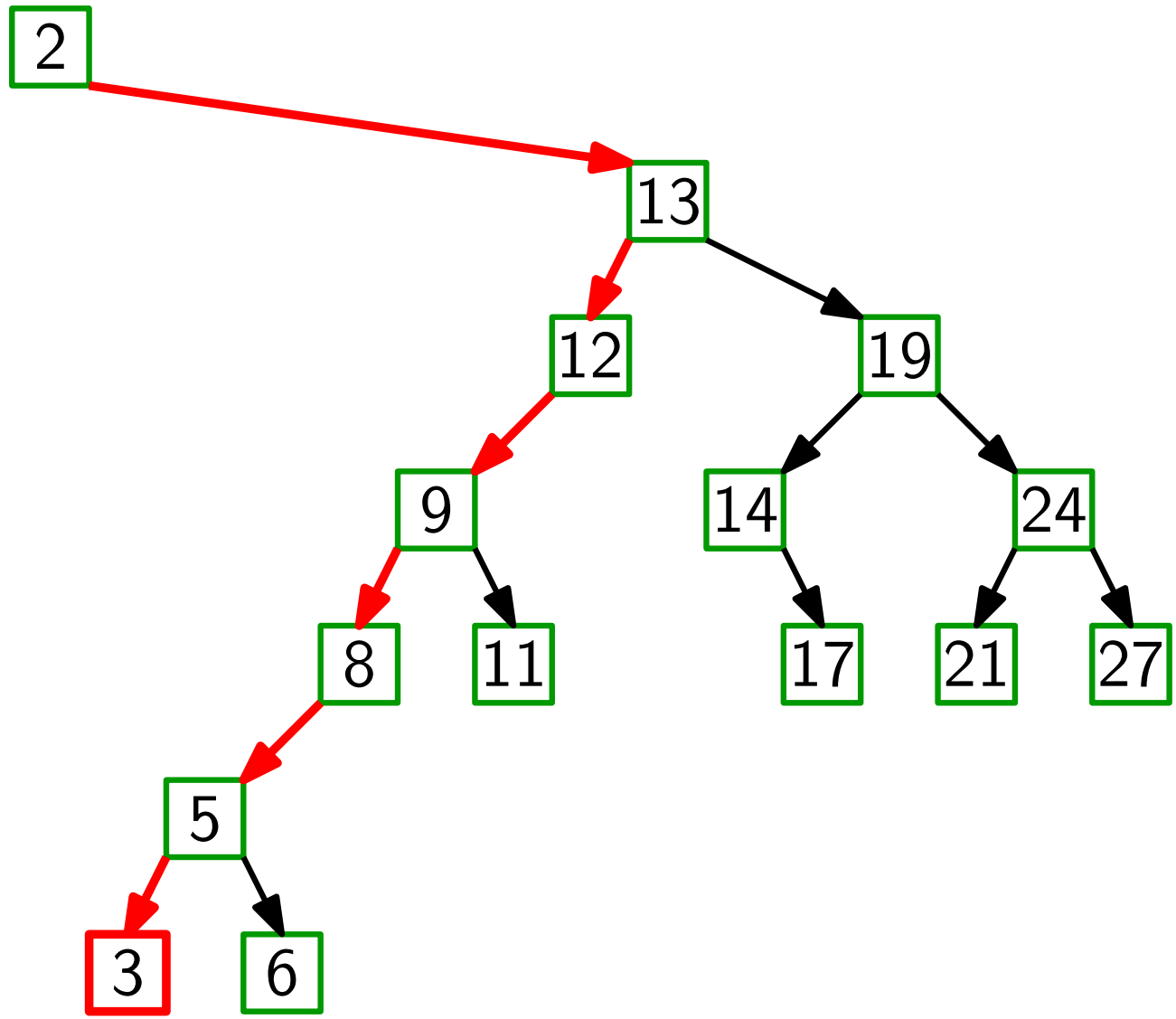
Binäre Suchbäume

Binärer Suchbaum:



Binäre Suchbäume

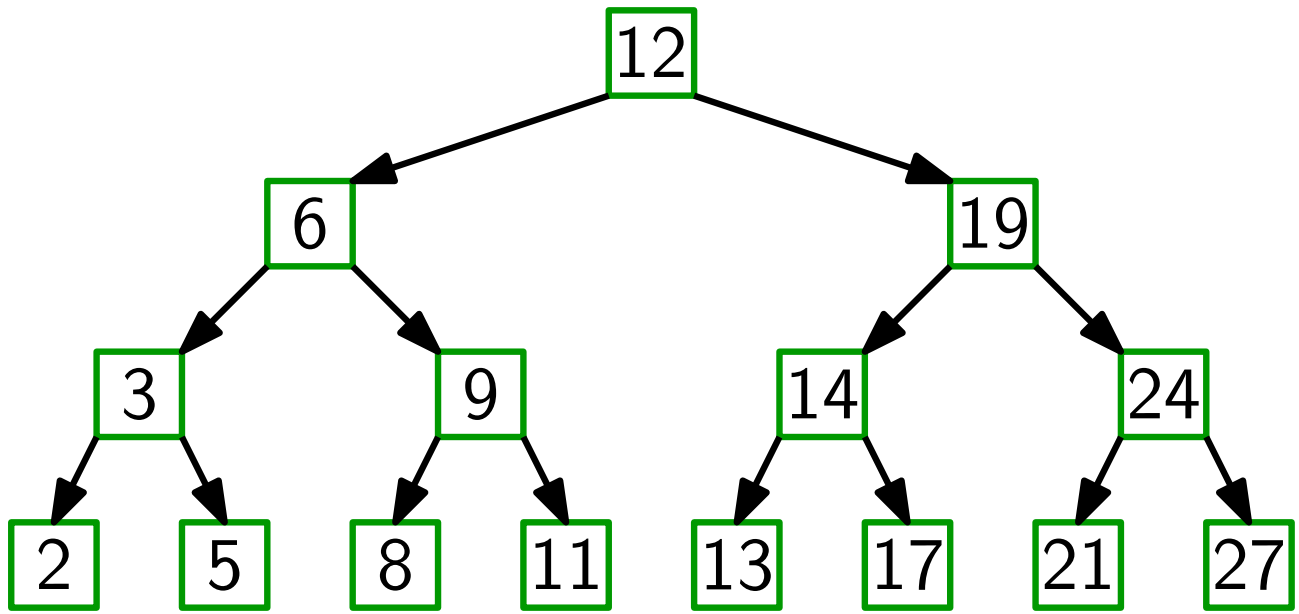
Binärer Suchbaum: Anfragezeit $O(n)$



Binäre Suchbäume

Binärer Suchbaum: Anfragezeit $O(n)$

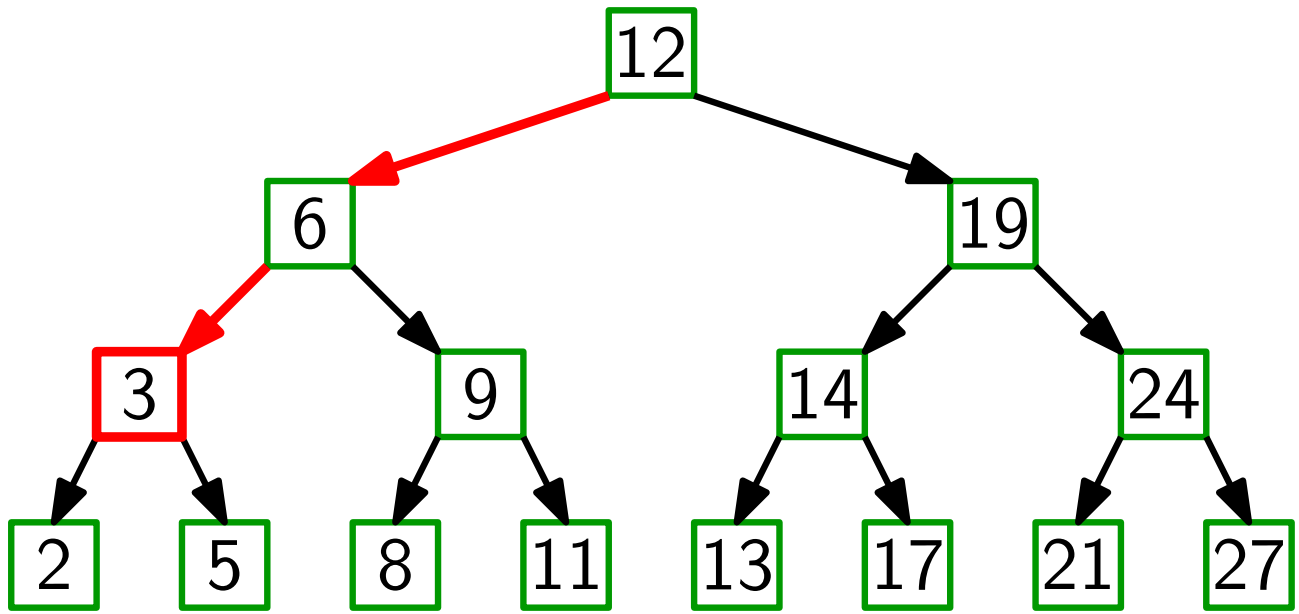
Balancierter Suchbaum:
(z.B. AVL-Baum)



Binäre Suchbäume

Binärer Suchbaum: Anfragezeit $O(n)$

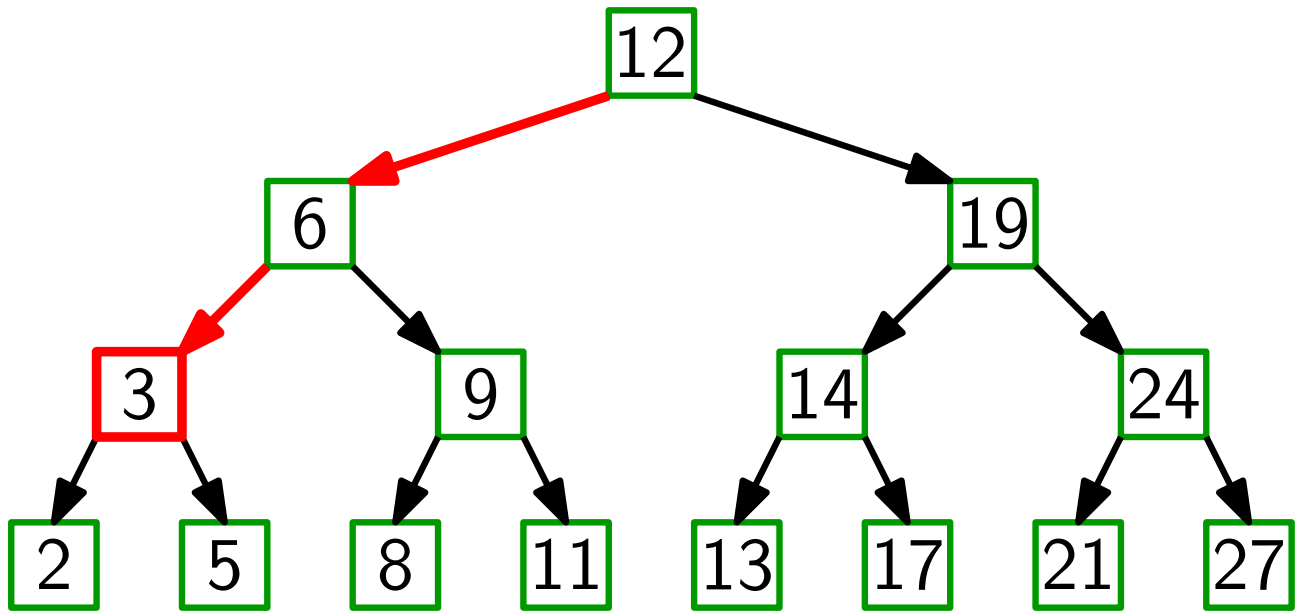
Balancierter Suchbaum:
(z.B. AVL-Baum)



Binäre Suchbäume

Binärer Suchbaum: Anfragezeit $O(n)$

Balancierter Suchbaum: Anfragezeit $O(\log n)$
(z.B. AVL-Baum)

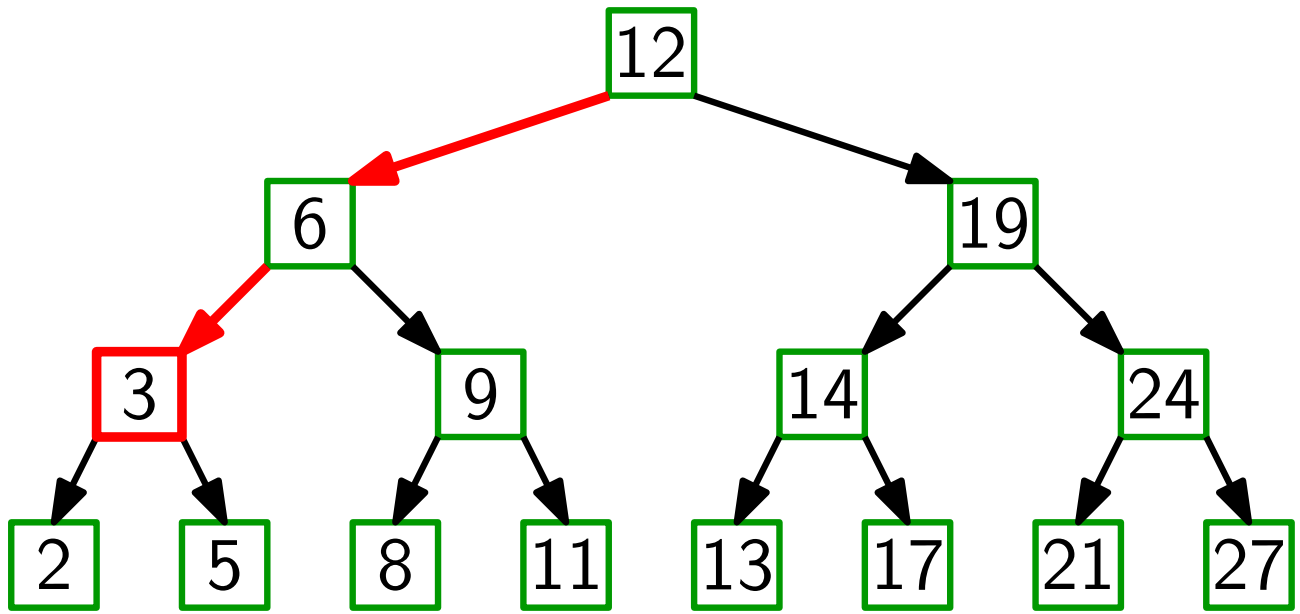


Binäre Suchbäume

Binärer Suchbaum: Anfragezeit $O(n)$

optimal

Balancierter Suchbaum: Anfragezeit $O(\log n)$
(z.B. AVL-Baum)



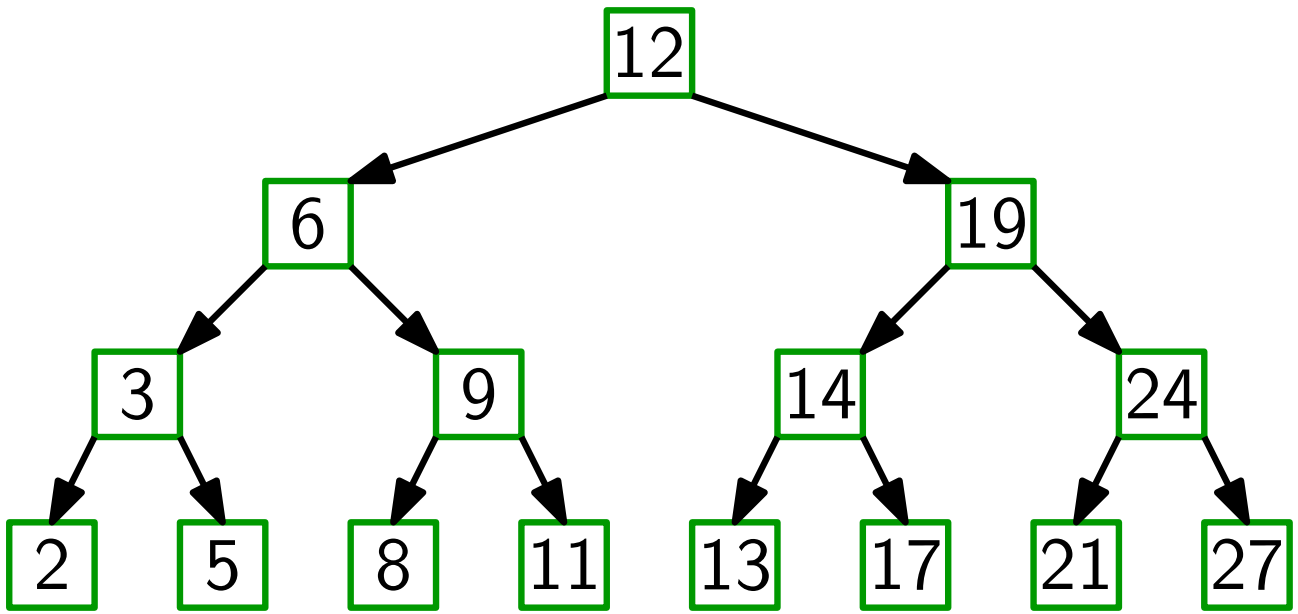
Binäre Suchbäume

Binärer Suchbaum: Anfragezeit $O(n)$

optimal

Balancierter Suchbaum: Anfragezeit $O(\log n)$
(z.B. AVL-Baum)

Anfragezeit für Such*sequenz*?



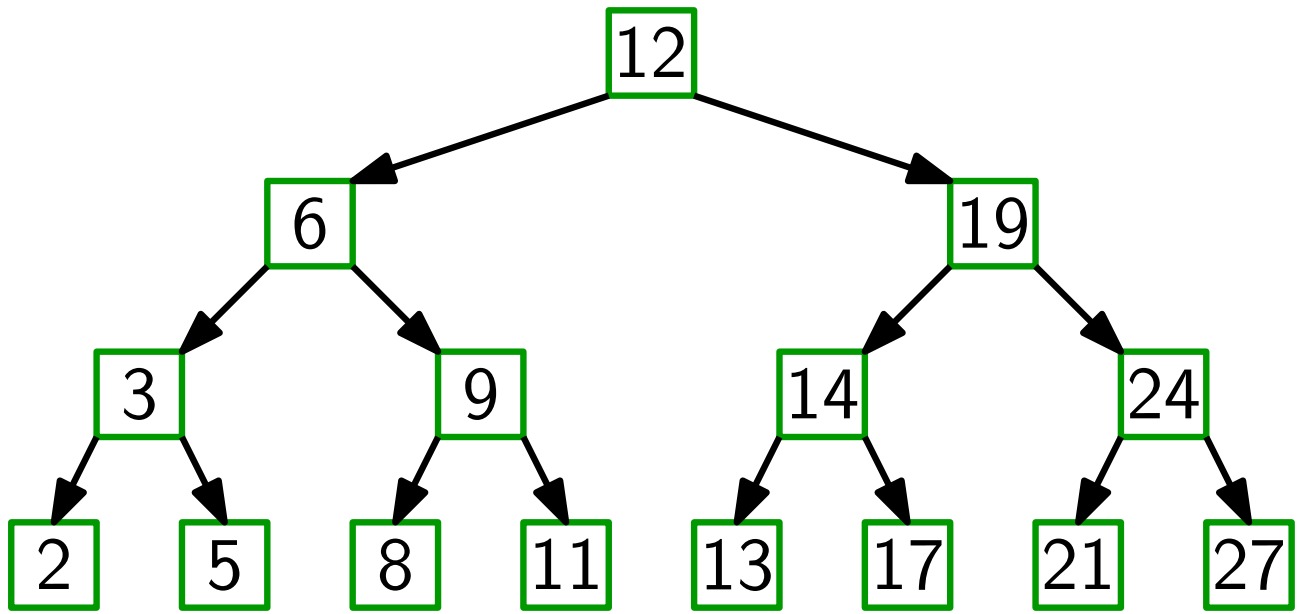
Binäre Suchbäume

Binärer Suchbaum: Anfragezeit $O(n)$

optimal

Balancierter Suchbaum: Anfragezeit $O(\log n)$
(z.B. AVL-Baum)

Anfragezeit für Such*sequenz*?
z.B. 2—13—5



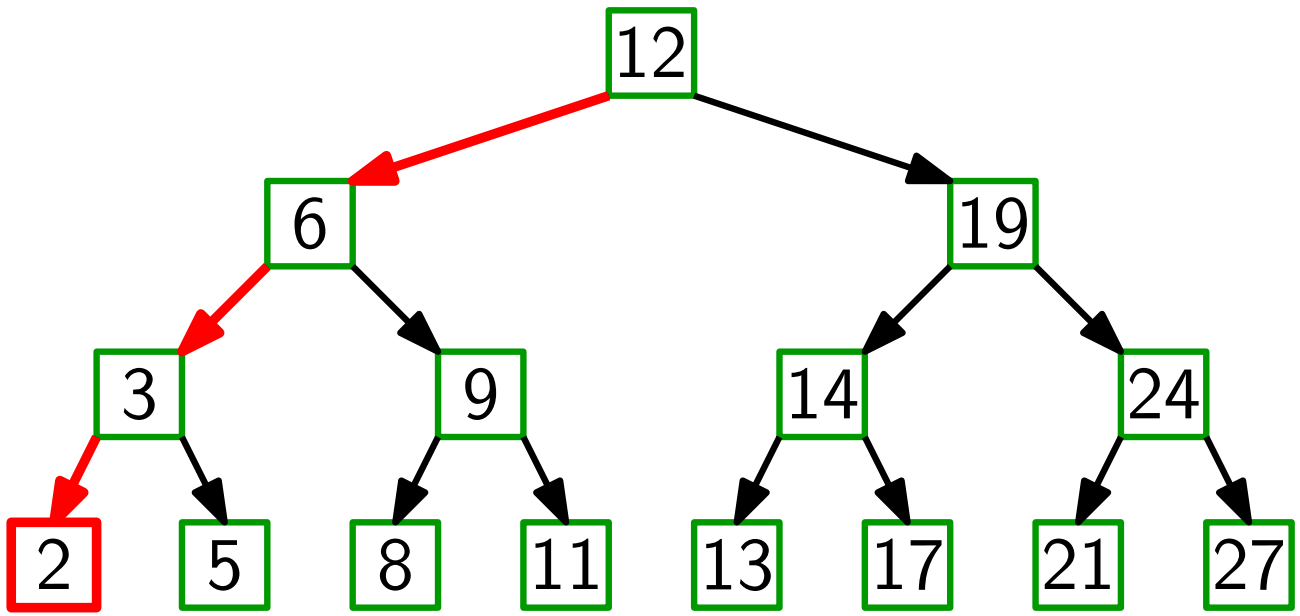
Binäre Suchbäume

Binärer Suchbaum: Anfragezeit $O(n)$

optimal

Balancierter Suchbaum: Anfragezeit $O(\log n)$
(z.B. AVL-Baum)

Anfragezeit für Such*sequenz*?
z.B. 2—13—5



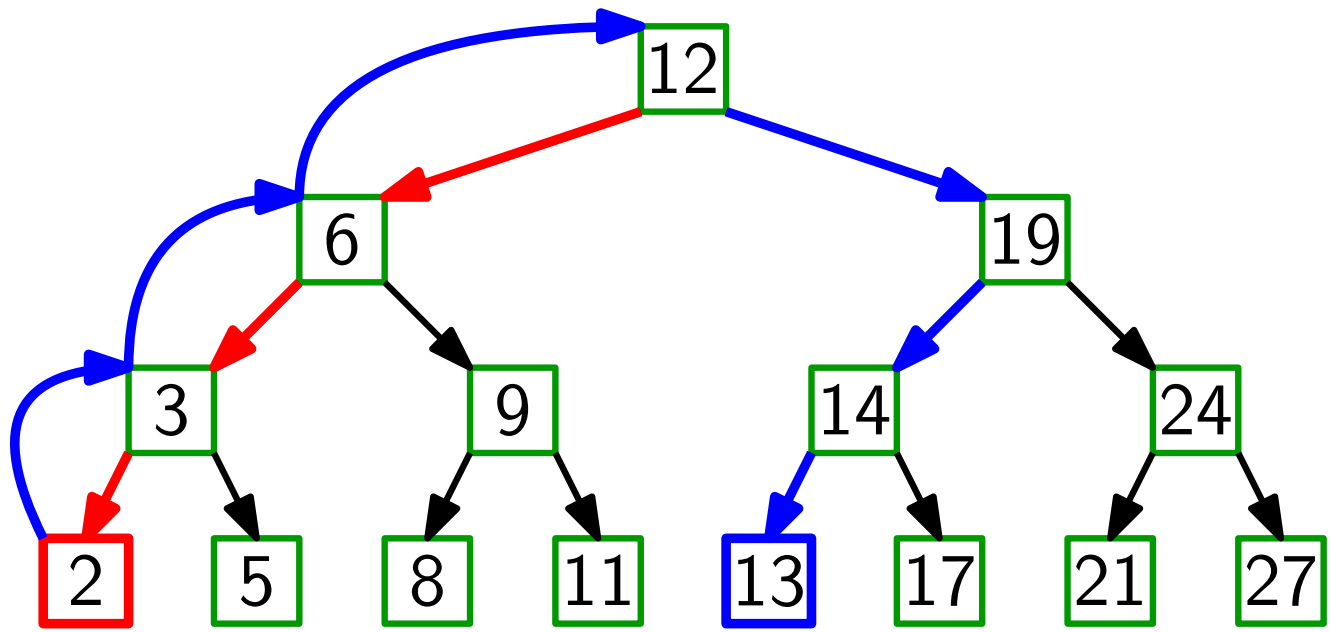
Binäre Suchbäume

Binärer Suchbaum: Anfragezeit $O(n)$

optimal

Balancierter Suchbaum: Anfragezeit $O(\log n)$
(z.B. AVL-Baum)

Anfragezeit für Such*sequenz*?
z.B. 2—13—5



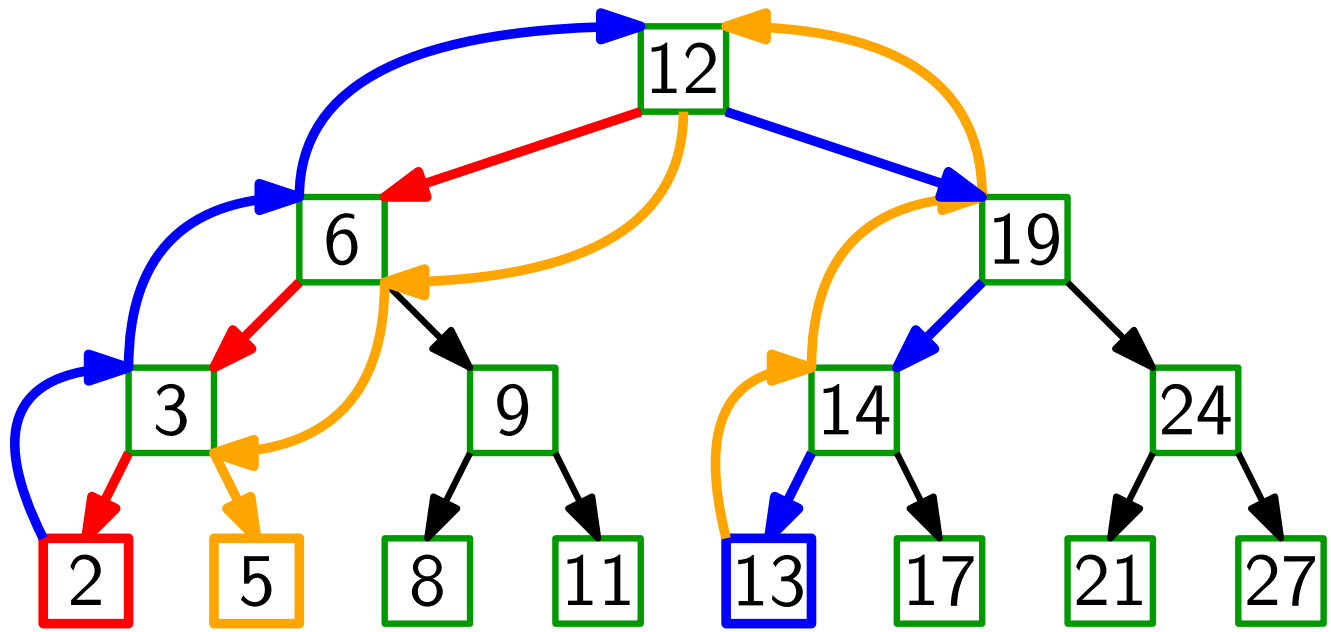
Binäre Suchbäume

Binärer Suchbaum: Anfragezeit $O(n)$

optimal

Balancierter Suchbaum: Anfragezeit $O(\log n)$
(z.B. AVL-Baum)

Anfragezeit für Such*sequenz*?
z.B. 2—13—5



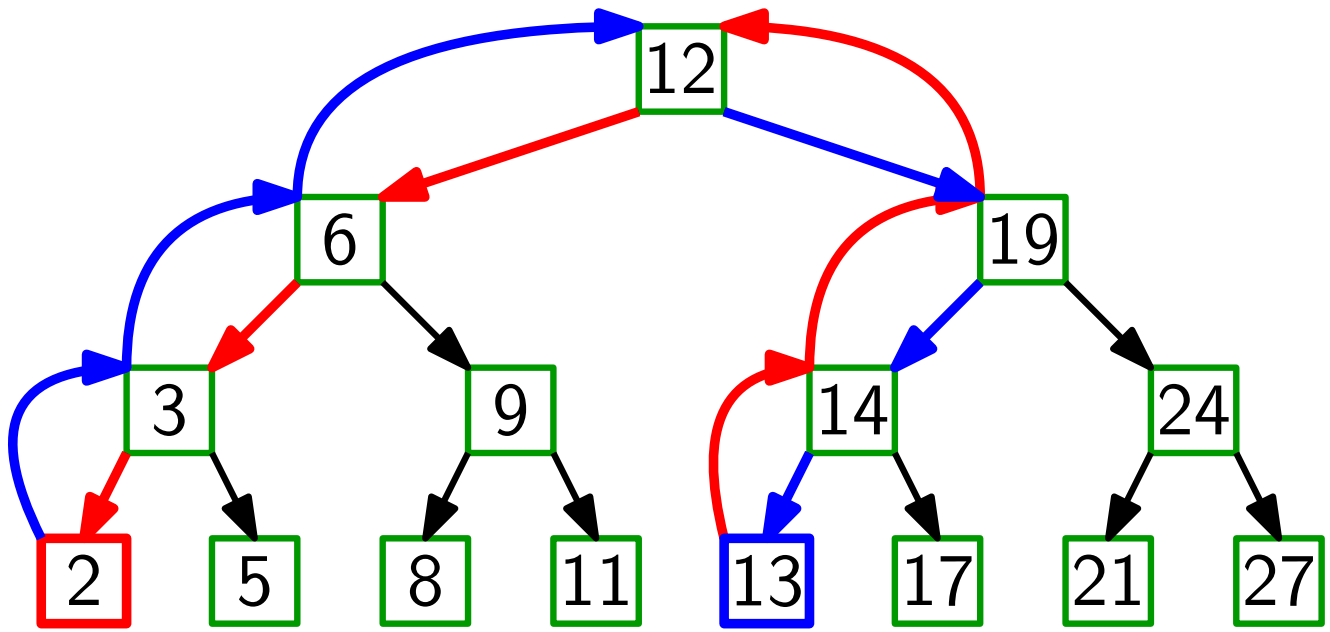
Binäre Suchbäume

Binärer Suchbaum: Anfragezeit $O(n)$

optimal

Balancierter Suchbaum: Anfragezeit $O(\log n)$
(z.B. AVL-Baum)

Anfragezeit für Such*sequenz*?
z.B. 2—13—5
oder 2—13—2—13—2...



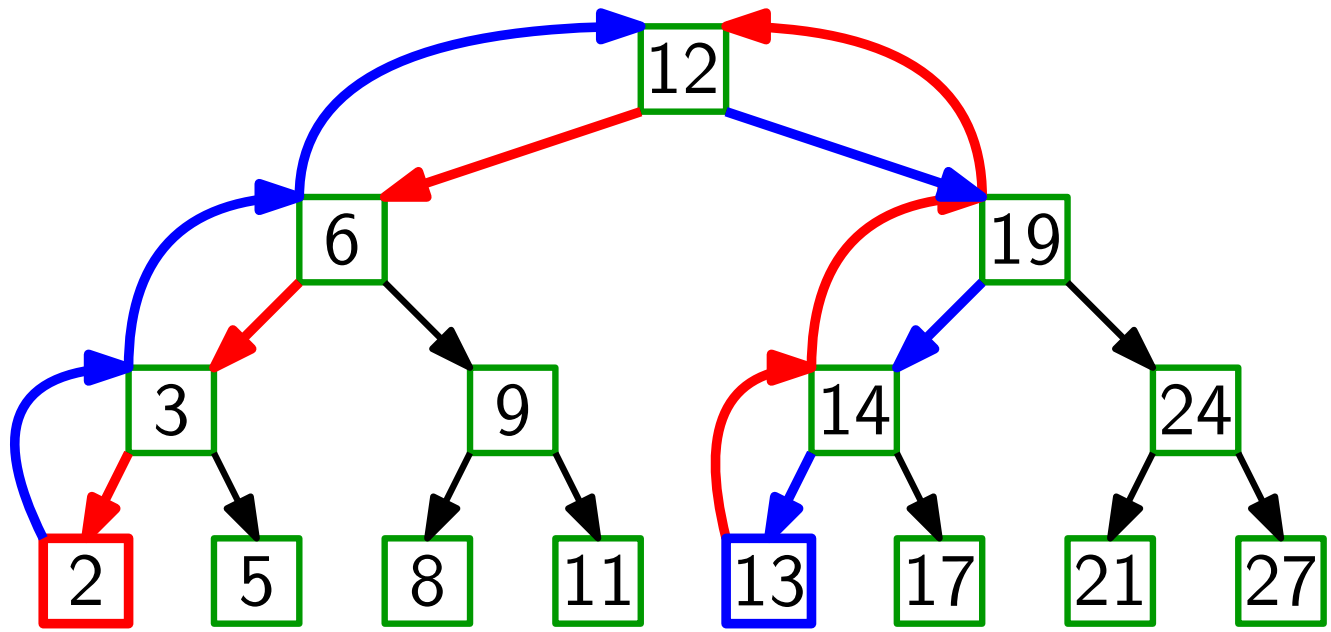
Binäre Suchbäume

Binärer Suchbaum: Anfragezeit $O(n)$

optimal

Balancierter Suchbaum: Anfragezeit $O(\log n)$
(z.B. AVL-Baum)

Anfragezeit für Such*sequenz*? $O(\log n)$ pro Schritt
z.B. 2—13—5
oder 2—13—2—13—2...



Binäre Suchbäume

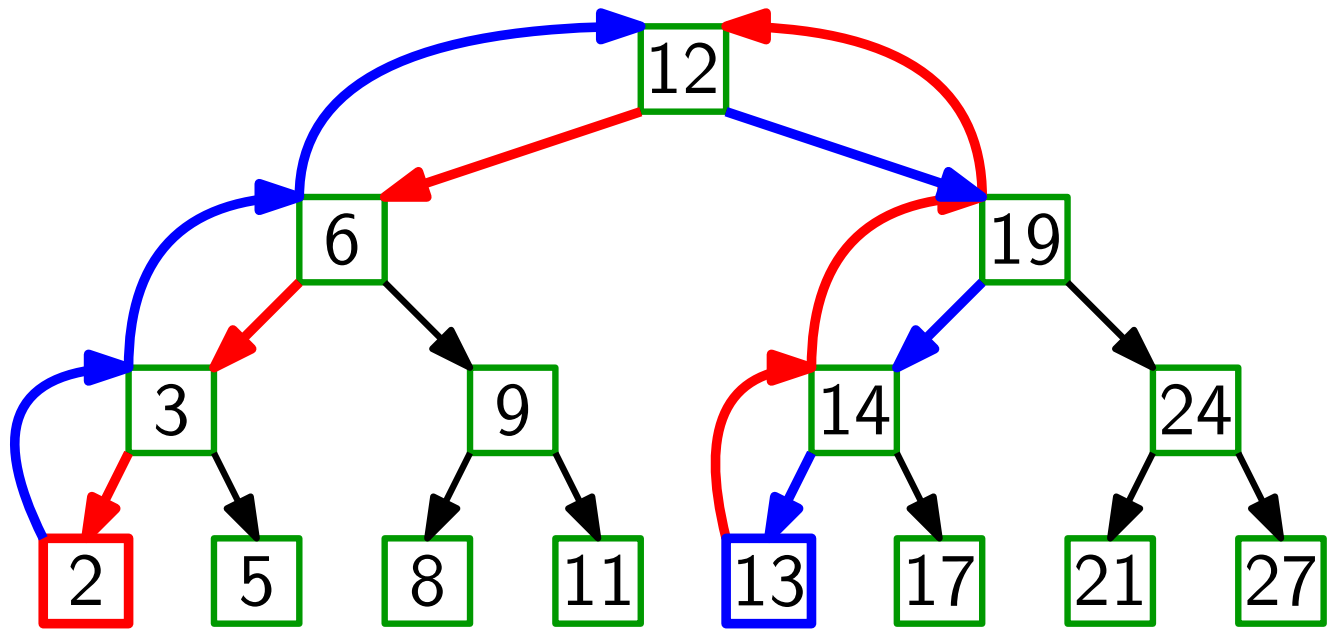
Binärer Suchbaum: Anfragezeit $O(n)$

optimal

Balancierter Suchbaum: Anfragezeit $O(\log n)$
(z.B. AVL-Baum)

optimal?

Anfragezeit für Such*sequenz*? $O(\log n)$ pro Schritt
z.B. 2—13—5
oder 2—13—2—13—2...

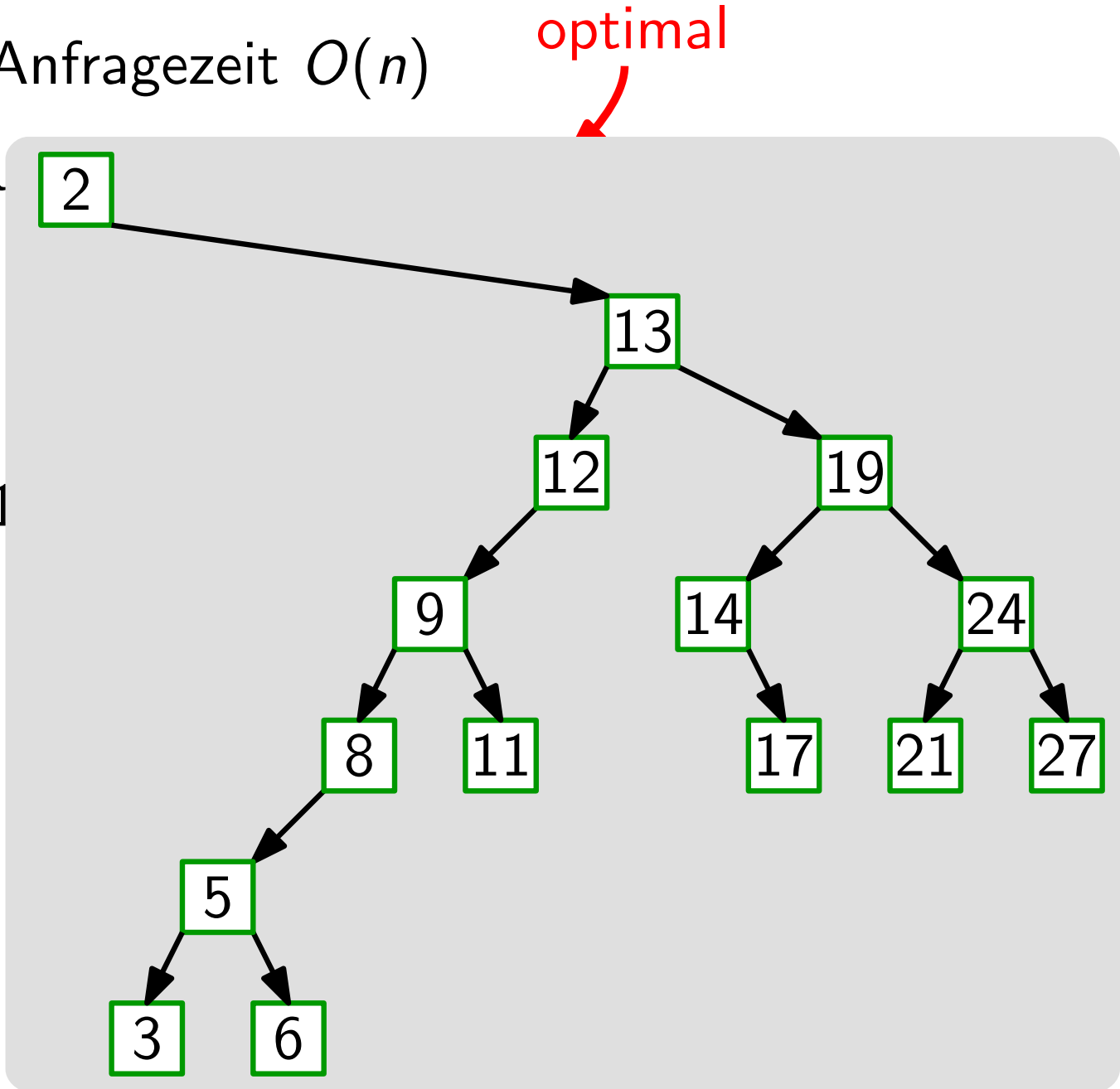


Binäre Suchbäume

Binärer Suchbaum: Anfragezeit $O(n)$

Balancierter Suchbau
(z.B. AVL-Baum)

Anfragezeit für Such
z.B. 2—13—5
oder 2—13—2—1

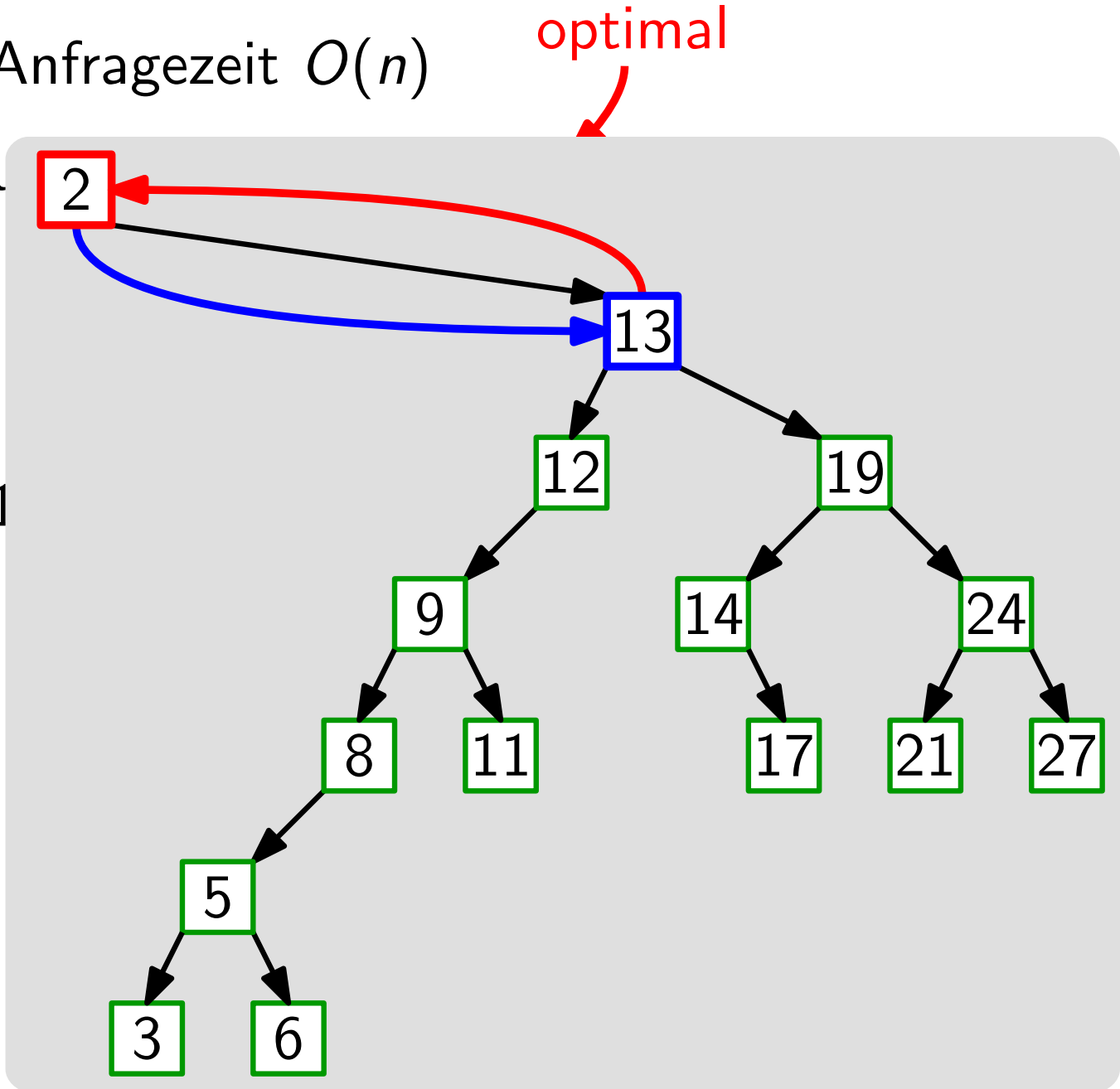


Binäre Suchbäume

Binärer Suchbaum: Anfragezeit $O(n)$

Balancierter Suchbau
(z.B. AVL-Baum)

Anfragezeit für Such
z.B. 2—13—5
oder 2—13—2—1



Binäre Suchbäume

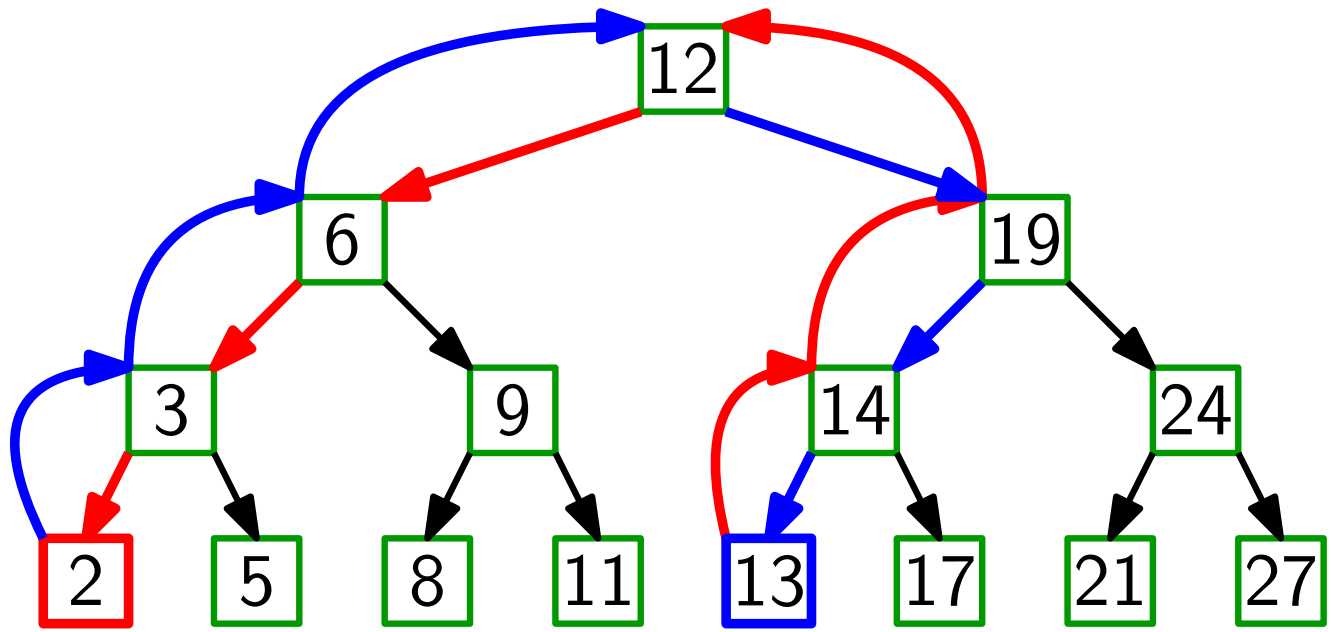
Binärer Suchbaum: Anfragezeit $O(n)$

optimal

Balancierter Suchbaum: Anfragezeit $O(\log n)$
(z.B. AVL-Baum)

optimal? nicht immer!

Anfragezeit für Such*sequenz*? $O(\log n)$ pro Schritt
z.B. 2—13—5
oder 2—13—2—13—2...



Binäre Suchbäume

Binärer Suchbaum: Anfragezeit $O(n)$

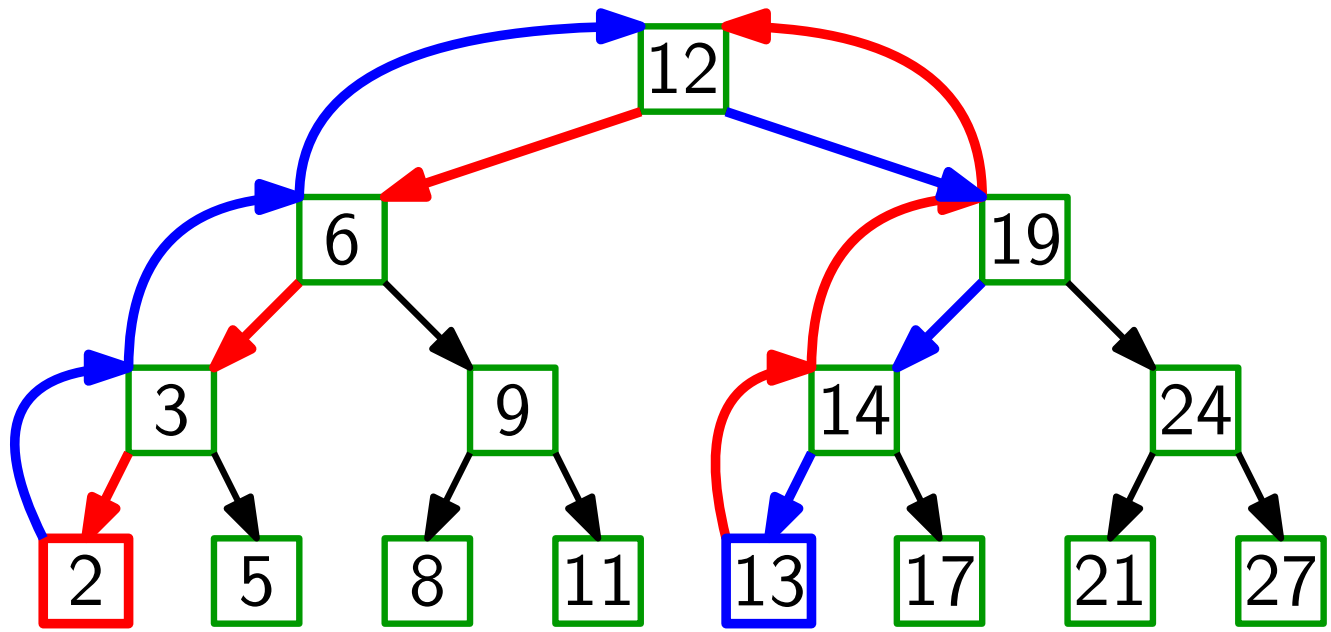
optimal

Balancierter Suchbaum: Anfragezeit $O(\log n)$
(z.B. AVL-Baum)

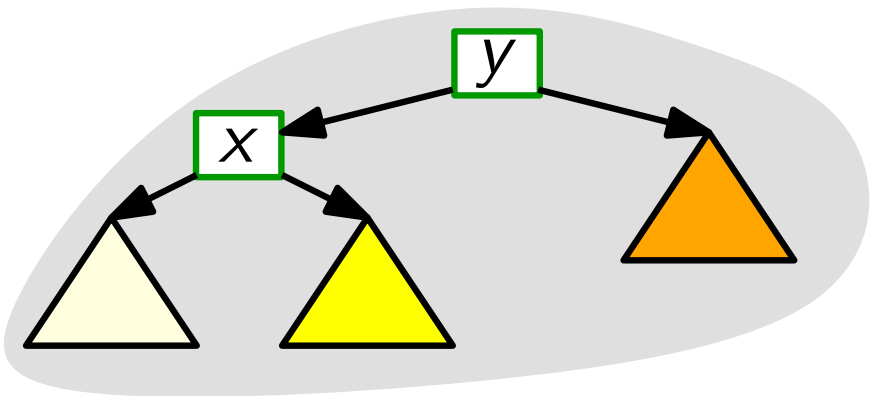
optimal? nicht immer!

Anfragezeit für Such*sequenz*? $O(\log n)$ pro Schritt
z.B. 2—13—5
oder 2—13—2—13—2...

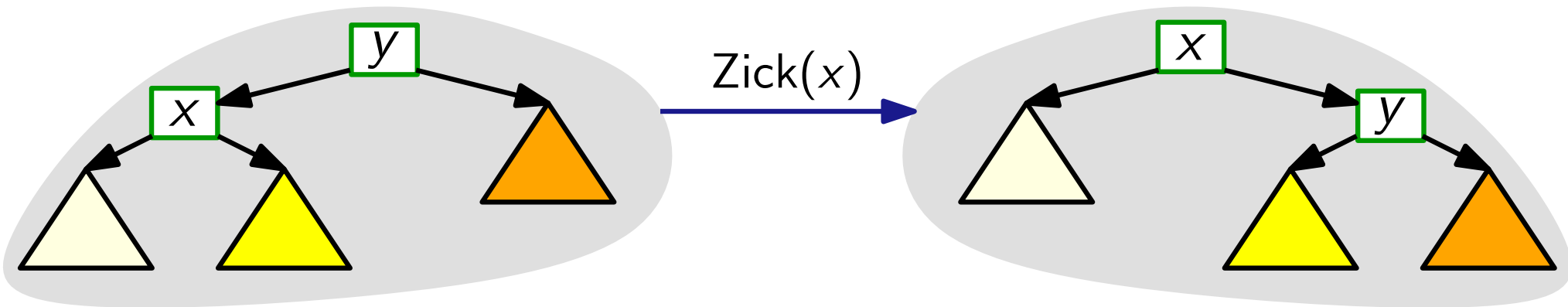
Erlaube Modifikation!



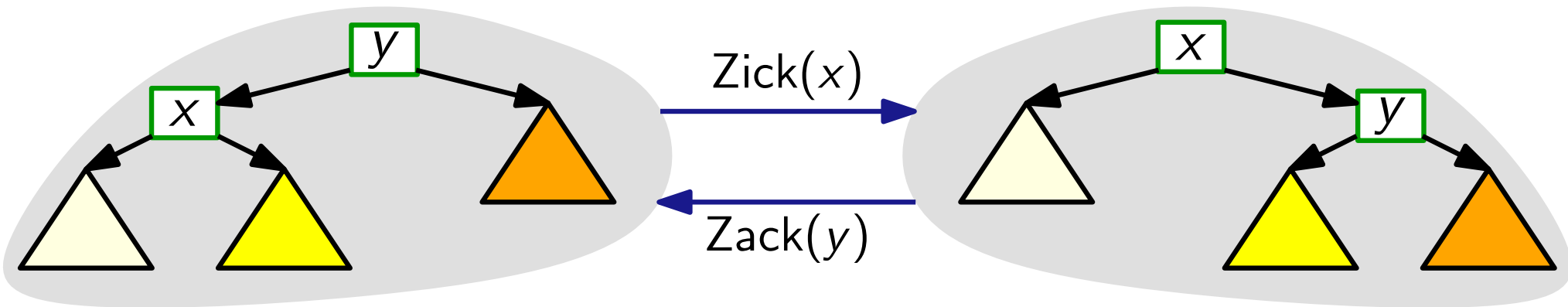
Rotationen



Rotationen



Rotationen



Dynamische Optimalität

Gegeben eine Sequenz von Anfragen $X = x_1, \ldots, x_m$
und ein Zeiger auf die Wurzel des Suchbaumes

Dynamische Optimalität

Gegeben eine Sequenz von Anfragen $X = x_1, \ldots, x_m$
und ein Zeiger auf die Wurzel des Suchbaumes

Aufgabe: Führe die Anfragen nacheinander aus

Dynamische Optimalität

Gegeben eine Sequenz von Anfragen $X = x_1, \ldots, x_m$
und ein Zeiger auf die Wurzel des Suchbaumes

Aufgabe: Führe die Anfragen nacheinander aus

Erlaubte Operationen:

Dynamische Optimalität

Gegeben eine Sequenz von Anfragen $X = x_1, \ldots, x_m$
und ein Zeiger auf die Wurzel des Suchbaumes

Aufgabe: Führe die Anfragen nacheinander aus

Erlaubte Operationen:  bewege Zeiger entlang Kante

Dynamische Optimalität

Gegeben eine Sequenz von Anfragen $X = x_1, \ldots, x_m$
und ein Zeiger auf die Wurzel des Suchbaumes

Aufgabe: Führe die Anfragen nacheinander aus

Erlaubte Operationen:

- bewege Zeiger entlang Kante
- rotiere um aktuellen Knoten

Dynamische Optimalität

Gegeben eine Sequenz von Anfragen $X = x_1, \ldots, x_m$
und ein Zeiger auf die Wurzel des Suchbaumes

Aufgabe: Führe die Anfragen nacheinander aus

Erlaubte Operationen:
- bewege Zeiger entlang Kante
- rotiere um aktuellen Knoten

Sei OPT_X minimale Anzahl an Operationen die benötigt werden

Dynamische Optimalität

Gegeben eine Sequenz von Anfragen $X = x_1, \ldots, x_m$
und ein Zeiger auf die Wurzel des Suchbaumes

Aufgabe: Führe die Anfragen nacheinander aus

Erlaubte Operationen:
- bewege Zeiger entlang Kante
- rotiere um aktuellen Knoten

Sei OPT_X minimale Anzahl an Operationen die benötigt werden

Dynamisch optimal: Suchbaum benötigt max.
$O(\mathrm{OPT}_X)$ Operationen

Dynamische Optimalität

Gegeben eine Sequenz von Anfragen $X = x_1, \ldots, x_m$
und ein Zeiger auf die Wurzel des Suchbaumes

Aufgabe: Führe die Anfragen nacheinander aus

Erlaubte Operationen:
- bewege Zeiger entlang Kante
- rotiere um aktuellen Knoten

Sei OPT_X minimale Anzahl an Operationen die benötigt werden

Dynamisch optimal: Suchbaum benötigt max.
$O(\mathrm{OPT}_X)$ Operationen

Offen

Dynamische Optimalität

Gegeben eine Sequenz von Anfragen $X = x_1, \ldots, x_m$
und ein Zeiger auf die Wurzel des Suchbaumes

Aufgabe: Führe die Anfragen nacheinander aus

Erlaubte Operationen:
- bewege Zeiger entlang Kante
- rotiere um aktuellen Knoten

Sei OPT_X minimale Anzahl an Operationen die benötigt werden

Dynamisch optimal: Suchbaum benötigt max.
$O(\mathrm{OPT}_X)$ Operationen

Offen

Kompetitives Verhältnis k: Datenstruktur benötigt max.
$O(k \cdot \mathrm{OPT}_X)$ Operationen

Dynamische Optimalität

Gegeben eine Sequenz von Anfragen $X = x_1, \ldots, x_m$
und ein Zeiger auf die Wurzel des Suchbaumes

Aufgabe: Führe die Anfragen nacheinander aus

Erlaubte Operationen:
- bewege Zeiger entlang Kante
- rotiere um aktuellen Knoten

Sei OPT_X minimale Anzahl an Operationen die benötigt werden

Dynamisch optimal: Suchbaum benötigt max.
$O(\mathrm{OPT}_X)$ Operationen

Offen

Kompetitives Verhältnis k: Datenstruktur benötigt max.
$O(k \cdot \mathrm{OPT}_X)$ Operationen

- Statischer Suchbaum: $k = \log n$

Dynamische Optimalität

Gegeben eine Sequenz von Anfragen $X = x_1, \ldots, x_m$
und ein Zeiger auf die Wurzel des Suchbaumes

Aufgabe: Führe die Anfragen nacheinander aus

Erlaubte Operationen:
- bewege Zeiger entlang Kante
- rotiere um aktuellen Knoten

Sei OPT_X minimale Anzahl an Operationen die benötigt werden

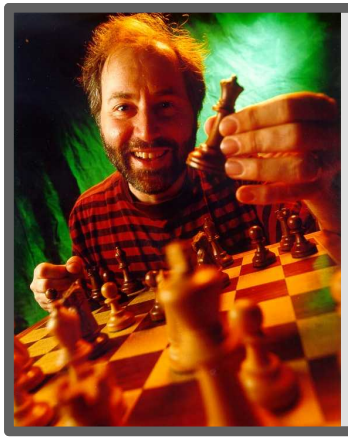
Dynamisch optimal: Suchbaum benötigt max.
$O(\mathrm{OPT}_X)$ Operationen

Offen

Kompetitives Verhältnis k: Datenstruktur benötigt max.
$O(k \cdot \mathrm{OPT}_X)$ Operationen

- Statischer Suchbaum: $k = \log n$
- Tango-Baum: $k = \log \log n$

Splay-Bäume

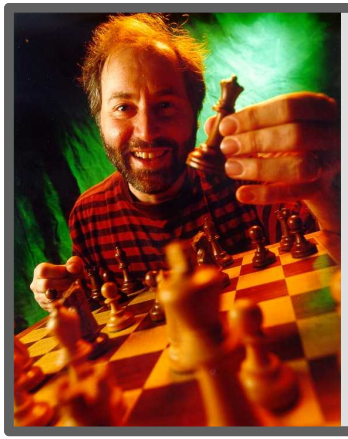


Daniel D. Sleator

Robert E. Tarjan

J. ACM 1985

Splay-Bäume

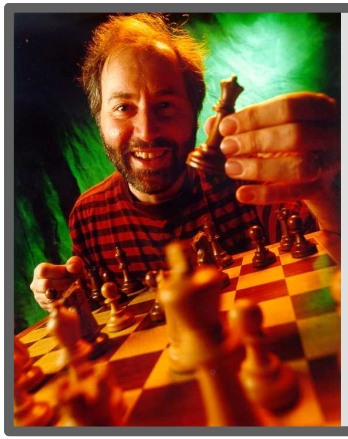


Daniel D. Sleator Robert E. Tarjan

J. ACM 1985

Idee: Nach jeder Suche nach x wird x zur Wurzel rotiert

Splay-Bäume



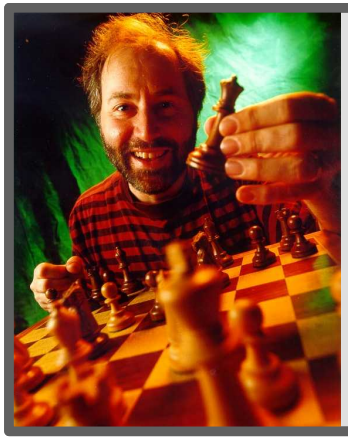
Daniel D. Sleator Robert E. Tarjan

J. ACM 1985

Idee: Nach jeder Suche nach x wird x zur Wurzel rotiert

Vermutung: Splay-Bäume sind dynamisch optimal

Splay-Bäume



Daniel D. Sleator Robert E. Tarjan

J. ACM 1985

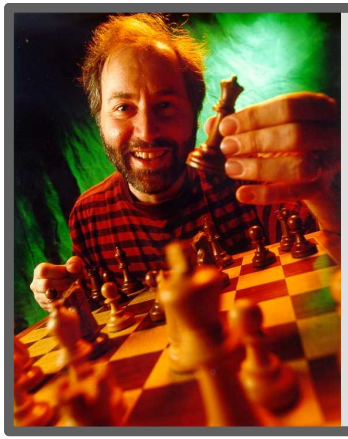
Idee: Nach jeder Suche nach x wird x zur Wurzel rotiert

Vermutung: Splay-Bäume sind dynamisch optimal

Sequentielle Zugriffseigenschaft: [Sleator & Tarjan]

$X = 1, 2, \ldots, n \Rightarrow O(n)$ Operationen insgesamt

Splay-Bäume



Daniel D. Sleator Robert E. Tarjan

J. ACM 1985

Idee: Nach jeder Suche nach x wird x zur Wurzel rotiert

Vermutung: Splay-Bäume sind dynamisch optimal

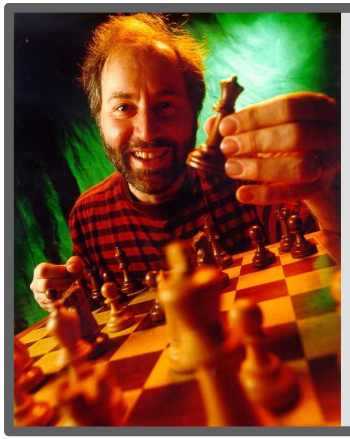
Sequentielle Zugriffseigenschaft: [Sleator & Tarjan]

$X = 1, 2, \ldots, n \Rightarrow O(n)$ Operationen insgesamt

Dynamische Fingereigenschaft: [Cole - SICOMP 2000]

$|x_i - x_{i-1}| = k \Rightarrow O(\log k)$ Operationen (amort.)

Splay-Bäume



Daniel D. Sleator Robert E. Tarjan

J. ACM 1985

Idee: Nach jeder Suche nach x wird x zur Wurzel rotiert

Vermutung: Splay-Bäume sind dynamisch optimal

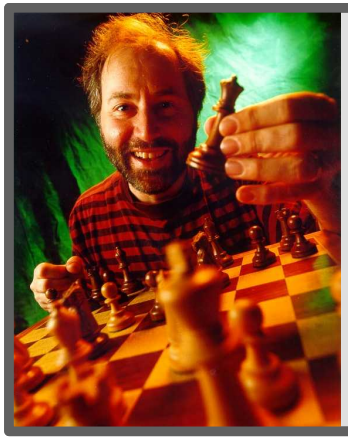
Sequentielle Zugriffseigenschaft: [Sleator & Tarjan]

$X = 1, 2, \ldots, n \Rightarrow O(n)$ Operationen insgesamt

Dynamische Fingereigenschaft: [Cole - SICOMP 2000]

$|x_i - x_{i-1}| = k \Rightarrow O(\log k)$ Operationen (amort.)

Splay-Bäume



Daniel D. Sleator Robert E. Tarjan

J. ACM 1985

Idee: Nach jeder Suche nach x wird x zur Wurzel rotiert

Vermutung: Splay-Bäume sind dynamisch optimal

Sequentielle Zugriffseigenschaft: [Sleator & Tarjan]

$X = 1, 2, \ldots, n \Rightarrow O(n)$ Operationen insgesamt

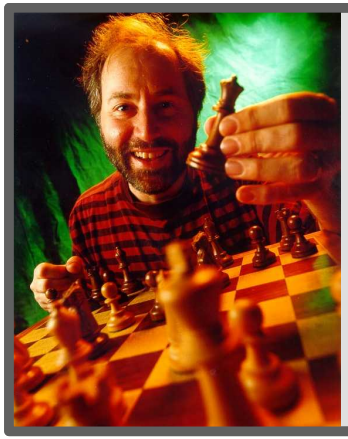
Dynamische Fingereigenschaft: [Cole - SICOMP 2000]

$|x_i - x_{i-1}| = k \Rightarrow O(\log k)$ Operationen (amort.)

Statische Optimalität: [Sleator & Tarjan]

$O(\mathrm{OPT}_X^{\mathrm{stat}})$ Operationen insg. (bester statischer Suchbaum)

Splay-Bäume



Daniel D. Sleator Robert E. Tarjan

J. ACM 1985

Idee: Nach jeder Suche nach x wird x zur Wurzel rotiert

Vermutung: Splay-Bäume sind dynamisch optimal

Sequentielle Zugriffseigenschaft: [Sleator & Tarjan]

$X = 1, 2, \ldots, n \Rightarrow O(n)$ Operationen insgesamt

Dynamische Fingereigenschaft: [Cole - SICOMP 2000]

$|x_i - x_{i-1}| = k \Rightarrow O(\log k)$ Operationen (amort.)

Statische Optimalität: [Sleator & Tarjan]

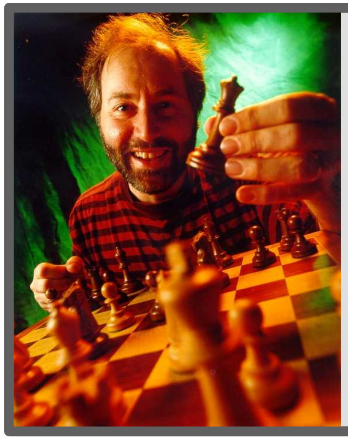
$O(\mathrm{OPT}_X^{\mathrm{stat}})$ Operationen insg. (bester statischer Suchbaum)

Arbeitsmengeneigenschaft: [Sleator & Tarjan]

$x_i = x_j$, dazwischen t_i andere Schlüssel gesucht

$\Rightarrow O(\log t_i)$ Operationen (amort.)

Splay-Bäume



Daniel D. Sleator Robert E. Tarjan

J. ACM 1985

Idee: Nach jeder Suche nach x wird x zur Wurzel rotiert

Vermutung: Splay-Bäume sind dynamisch optimal

Sequentielle Zugriffseigenschaft: [Sleator & Tarjan]

$X = 1, 2, \ldots, n \Rightarrow O(n)$ Operationen insgesamt

Dynamische Fingereigenschaft: [Cole - SICOMP 2000]

$|x_i - x_{i-1}| = k \Rightarrow O(\log k)$ Operationen (amort.)

Statische Optimalität: [Sleator & Tarjan]

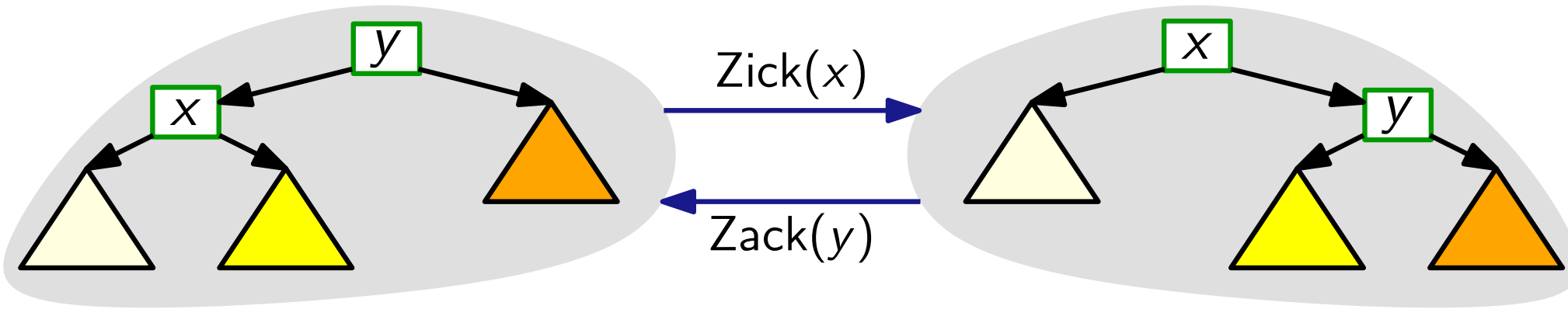
$O(\mathrm{OPT}_X^{\mathrm{stat}})$ Operationen insg. (bester statischer Suchbaum)

Arbeitsmengeneigenschaft: [Sleator & Tarjan]

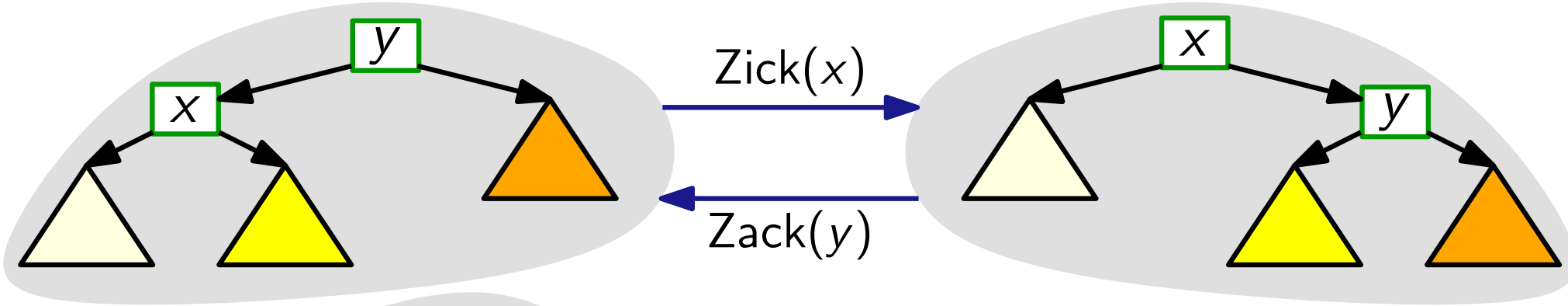
$x_i = x_j$, dazwischen t_i andere Schlüssel gesucht

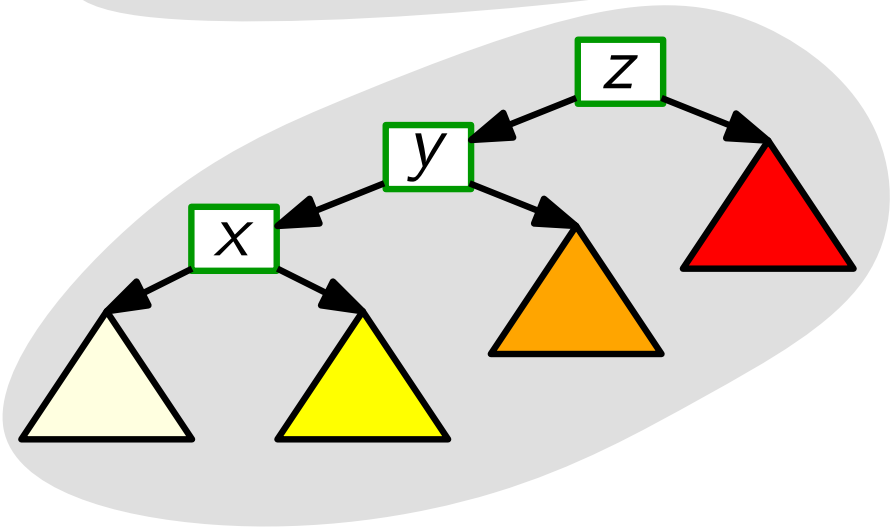
$\Rightarrow O(\log t_i)$ Operationen (amort.)

Rotationen II

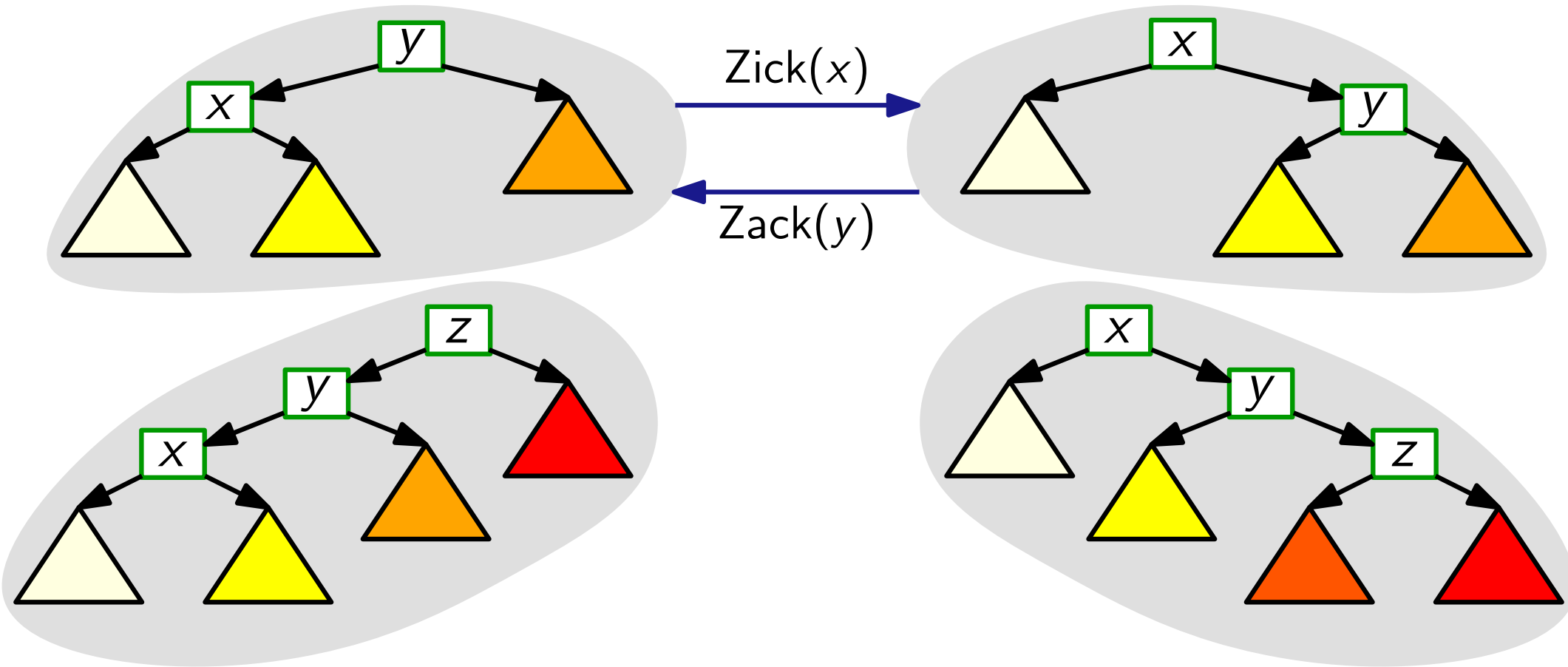


Rotationen II

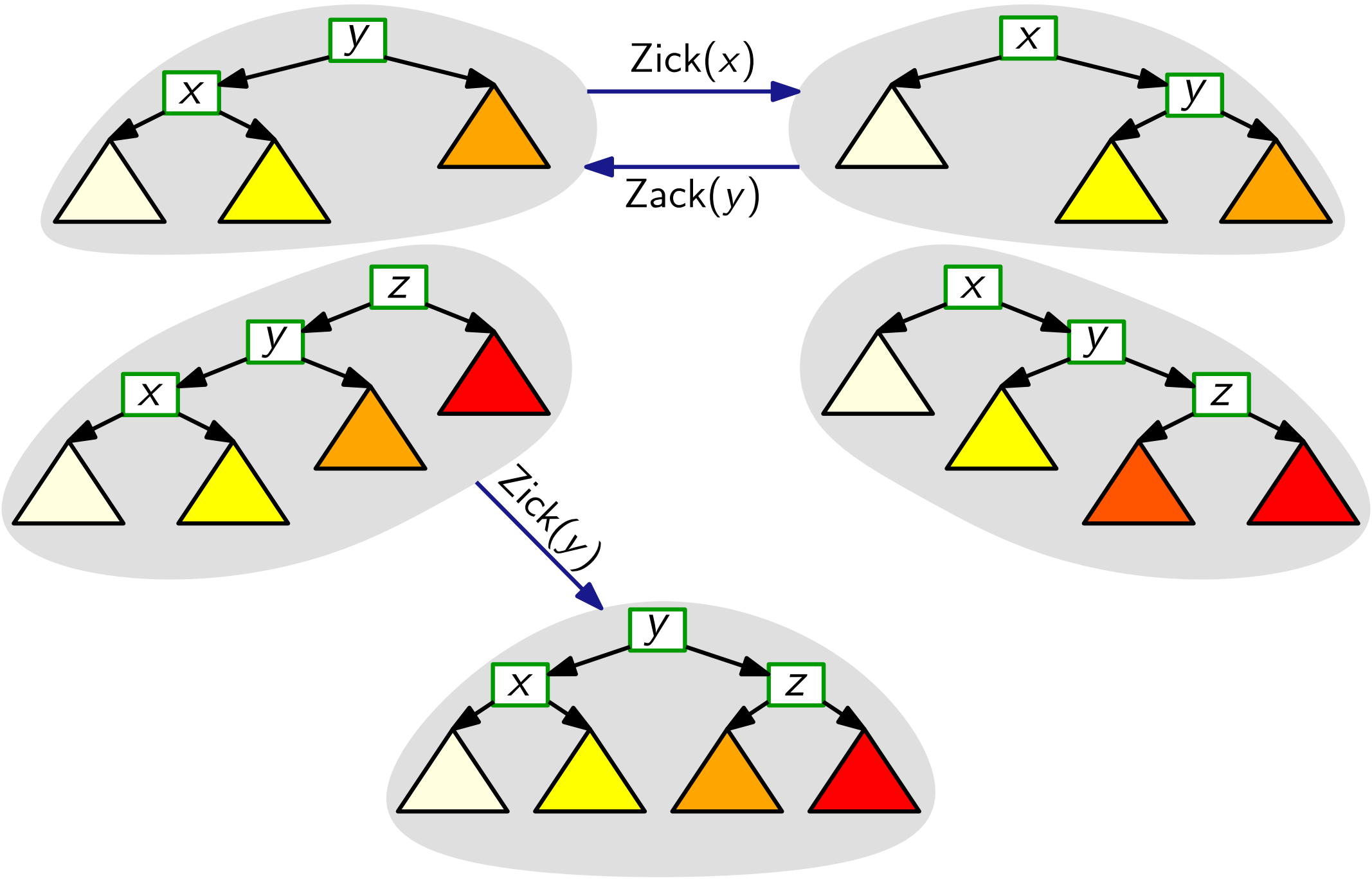




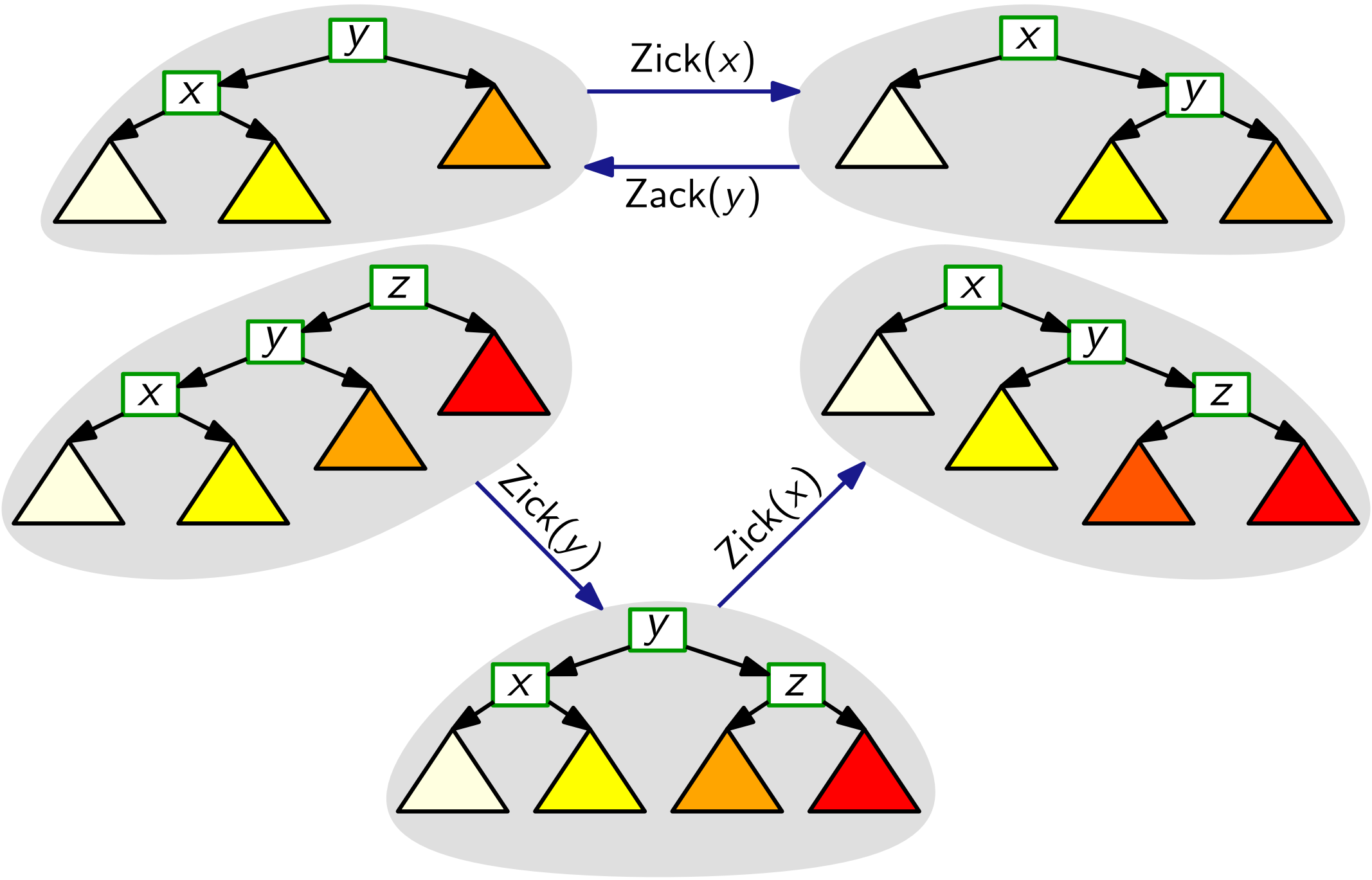
Rotationen II



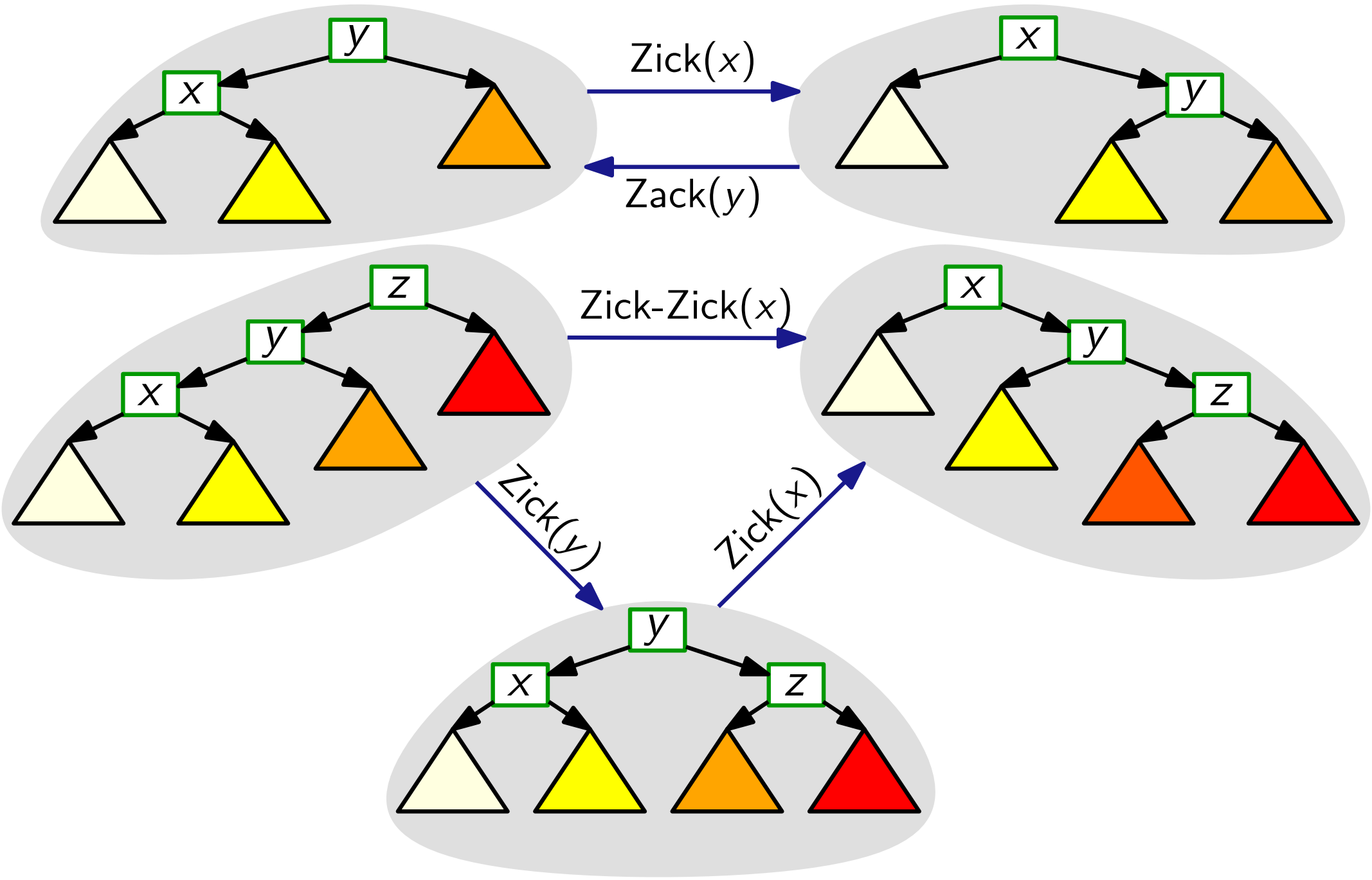
Rotationen II



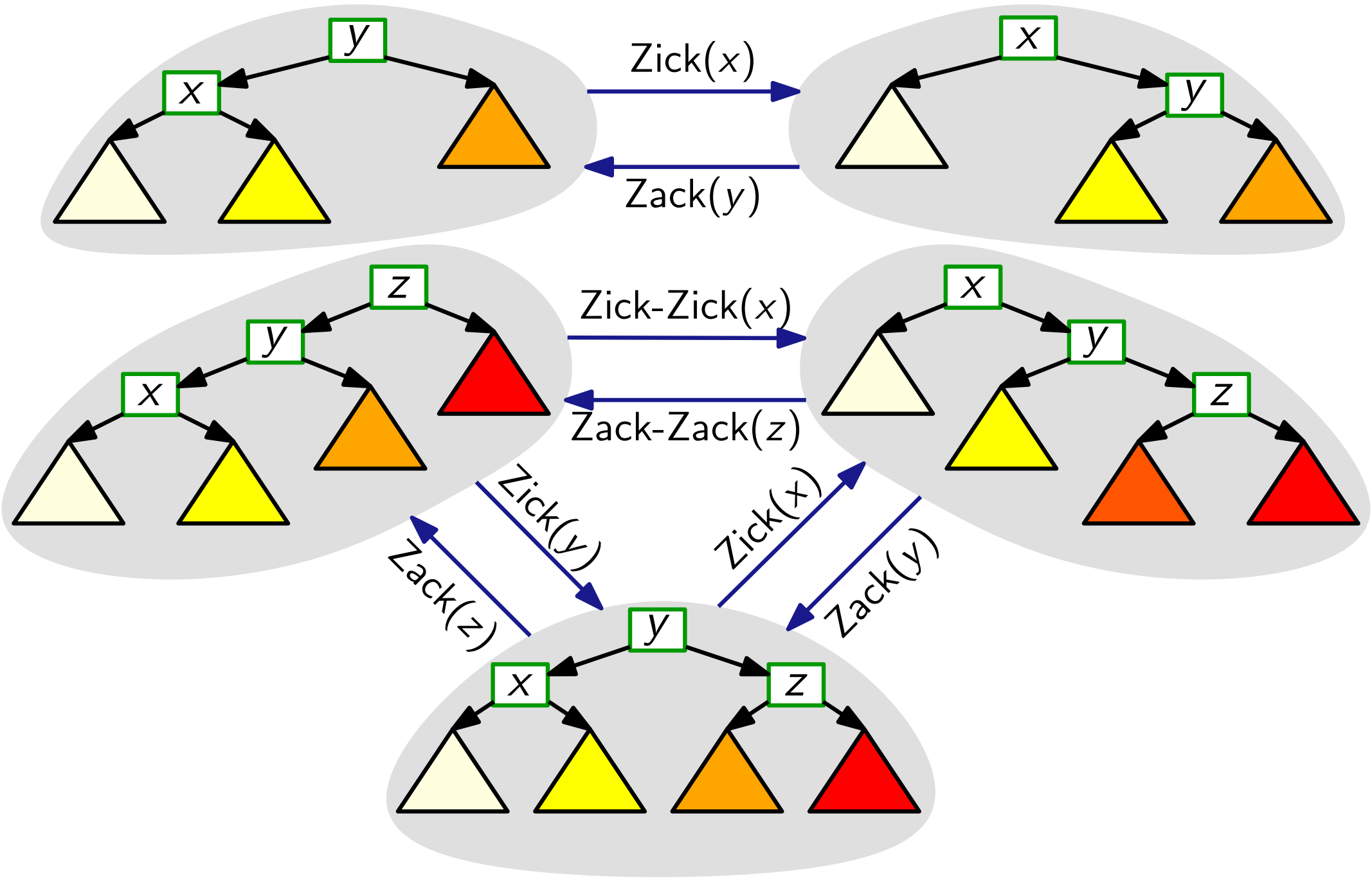
Rotationen II



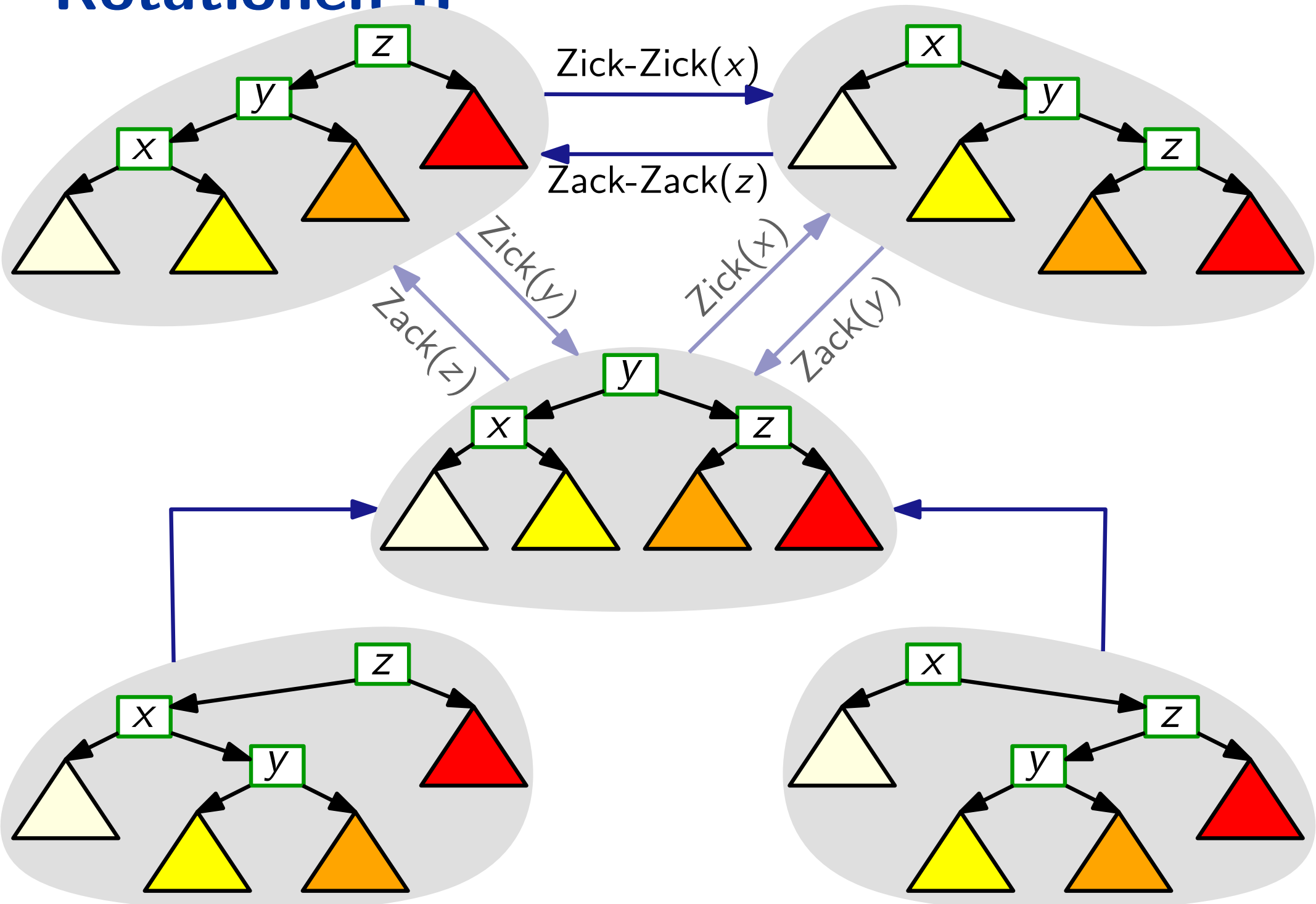
Rotationen II



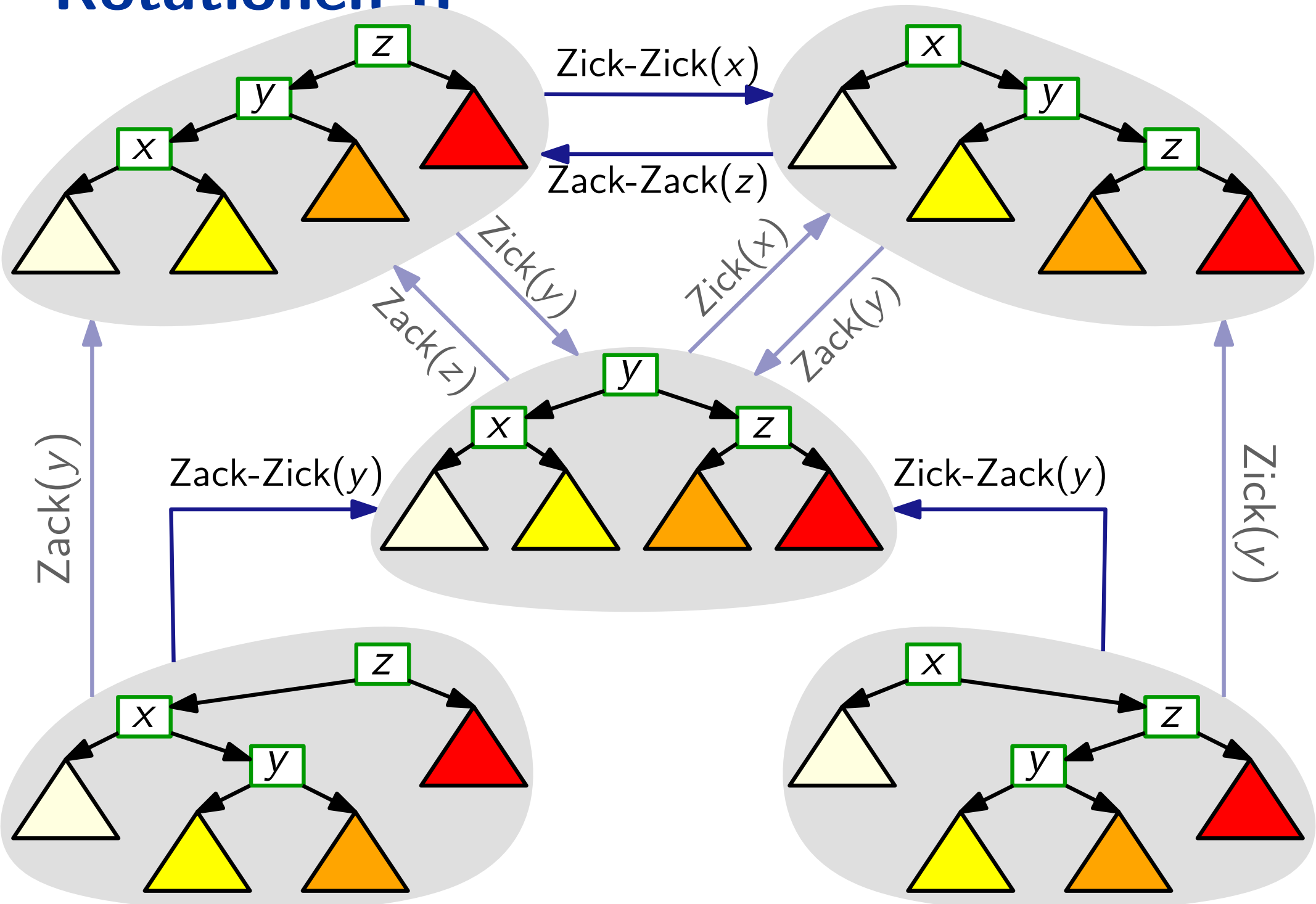
Rotationen II



Rotationen II



Rotationen II



Splay

Algorithm: Splay(x)

Splay

Algorithm: Splay(x)

if $x \neq root$ **then**

end

Splay

Algorithm: Splay(x)

if $x \neq root$ **then**

    $y =$ Elternknoten von x ;

end

Splay

Algorithm: Splay(x)

if $x \neq root$ **then**

    $y =$ Elternknoten von x ;

    if $y = root$ **then**

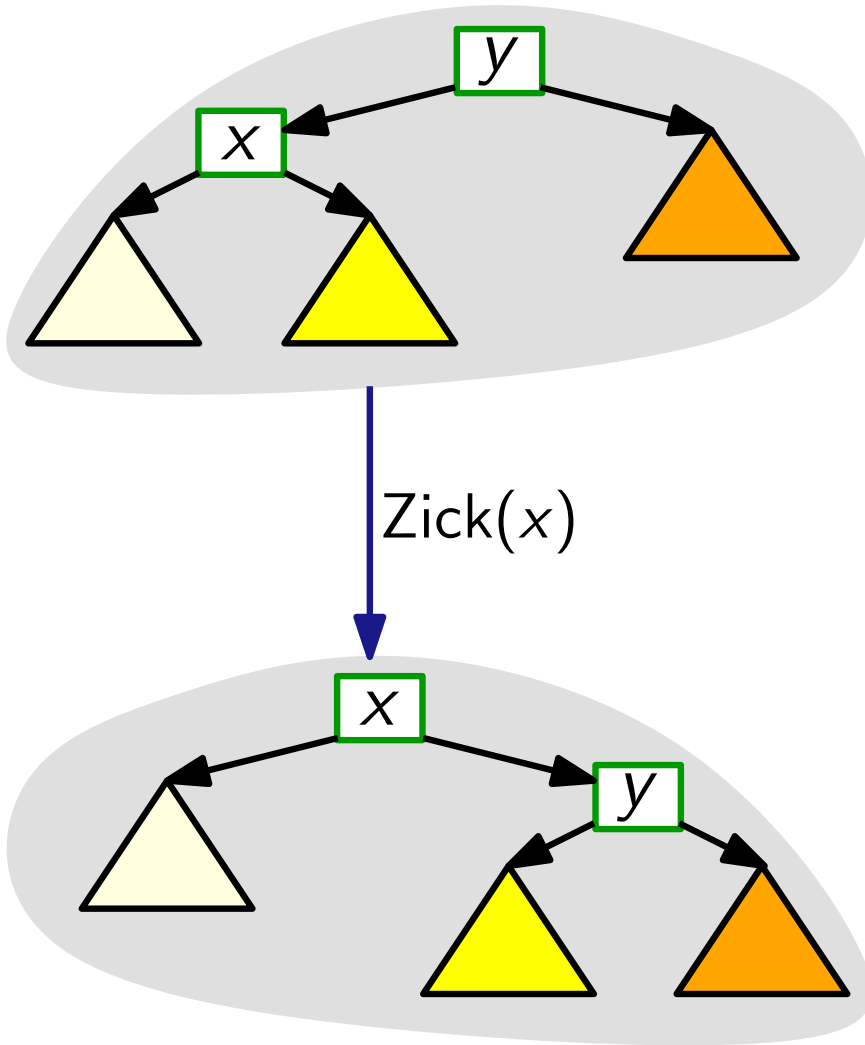
end

Splay

Algorithm: Splay(x)

if $x \neq root$ **then**
  $y =$ Elternknoten von x ;
  if $y = root$ **then**
    if $x < y$ **then** Zick(x);

end

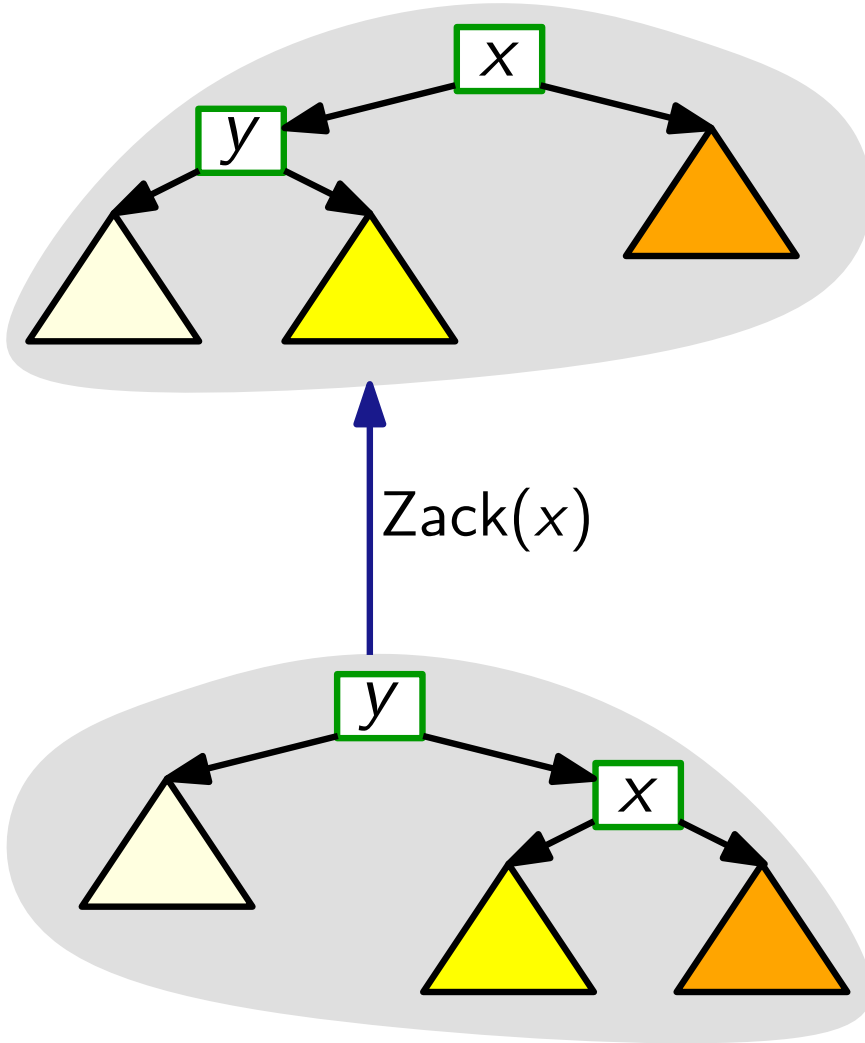


Splay

Algorithm: Splay(x)

if $x \neq root$ **then**
  $y =$ Elternknoten von x ;
  if $y = root$ **then**
    if $x < y$ **then** Zick(x);
    if $y < x$ **then** Zack(x);

end



Splay

Algorithm: Splay(x)

if $x \neq root$ **then**
  $y =$ Elternknoten von x ;
  if $y = root$ **then**
    if $x < y$ **then** Zick(x);
    if $y < x$ **then** Zack(x);
  else
    $z =$ Elternknoten von y ;

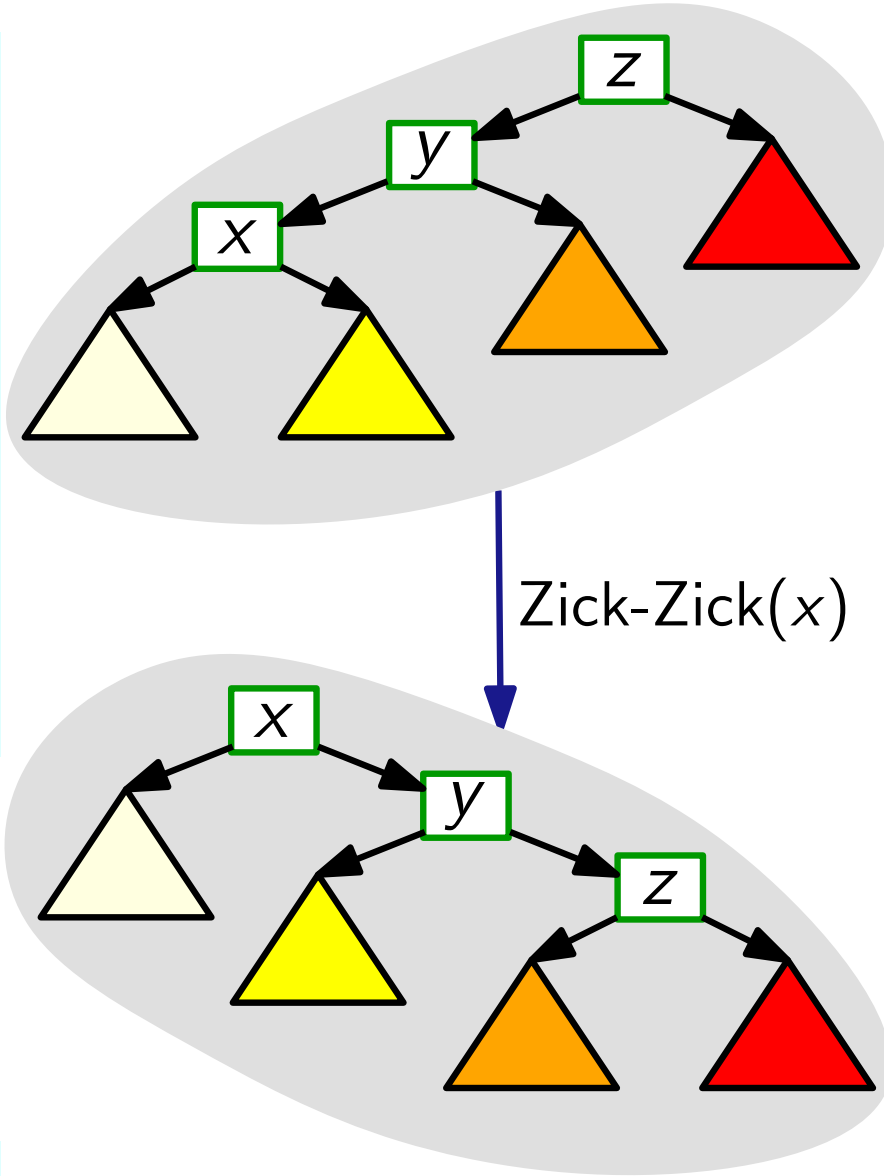
end

Splay

Algorithm: Splay(x)

if $x \neq root$ **then**
  $y =$ Elternknoten von x ;
  if $y = root$ **then**
    if $x < y$ **then** Zick(x);
    if $y < x$ **then** Zack(x);
  else
    $z =$ Elternknoten von y ;
    if $x < y < z$ **then** Zick-Zick(x);

end

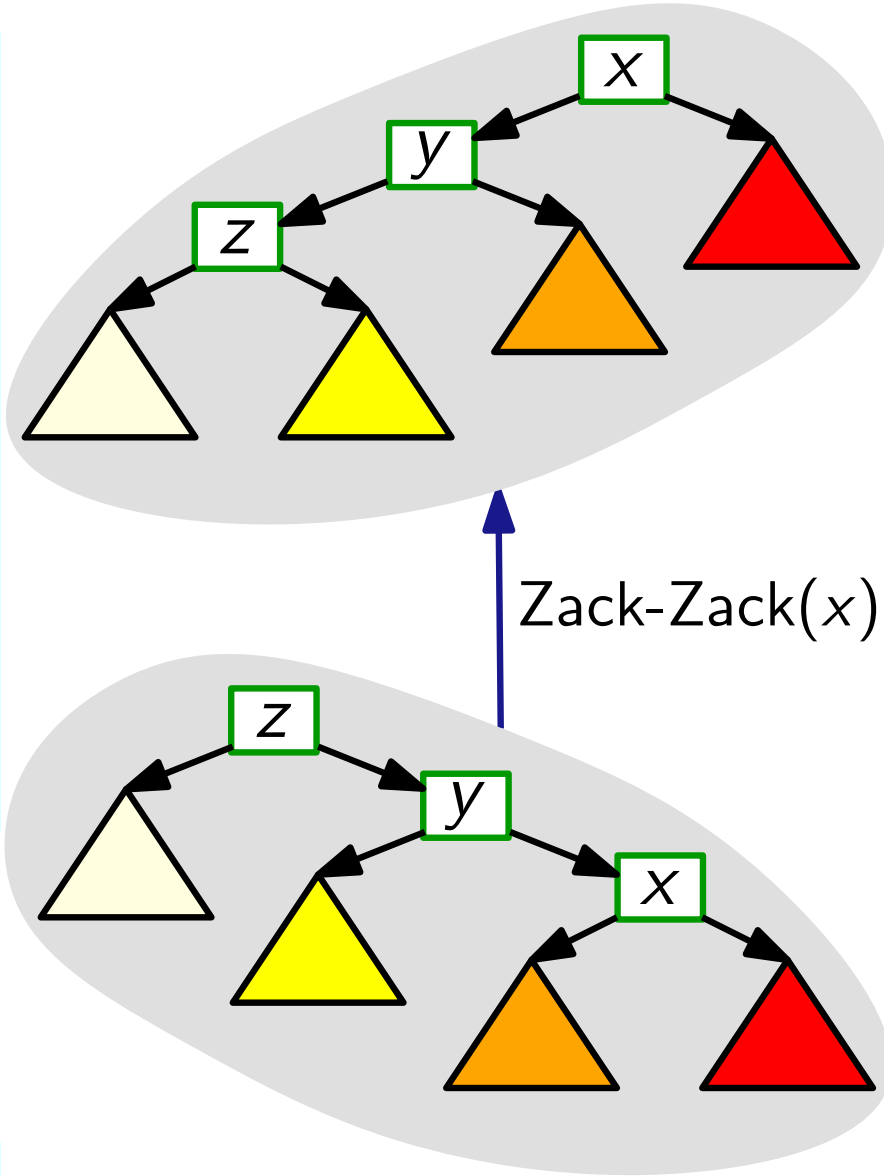


Splay

Algorithm: Splay(x)

if $x \neq root$ **then**
  $y =$ Elternknoten von x ;
  if $y = root$ **then**
    if $x < y$ **then** Zick(x);
    if $y < x$ **then** Zack(x);
  else
    $z =$ Elternknoten von y ;
    if $x < y < z$ **then** Zick-Zick(x);
    if $z < y < x$ **then** Zack-Zack(x);

end

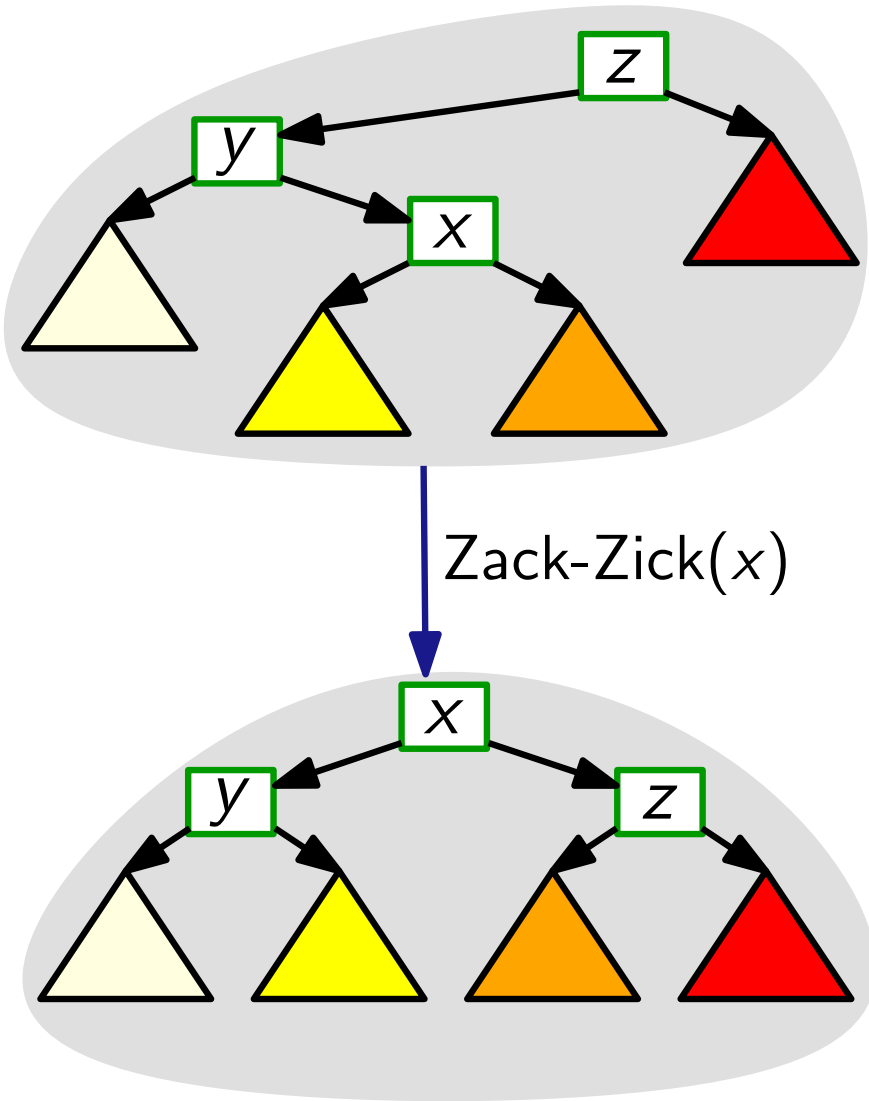


Splay

Algorithm: Splay(x)

if $x \neq root$ **then**
- $y =$ Elternknoten von x ;
- **if** $y = root$ **then**
  - **if** $x < y$ **then** Zick(x);
  - **if** $y < x$ **then** Zack(x);
- **else**
  - $z =$ Elternknoten von y ;
  - **if** $x < y < z$ **then** Zick-Zick(x);
  - **if** $z < y < x$ **then** Zack-Zack(x);
  - **if** $y < x < z$ **then** Zack-Zick(x);

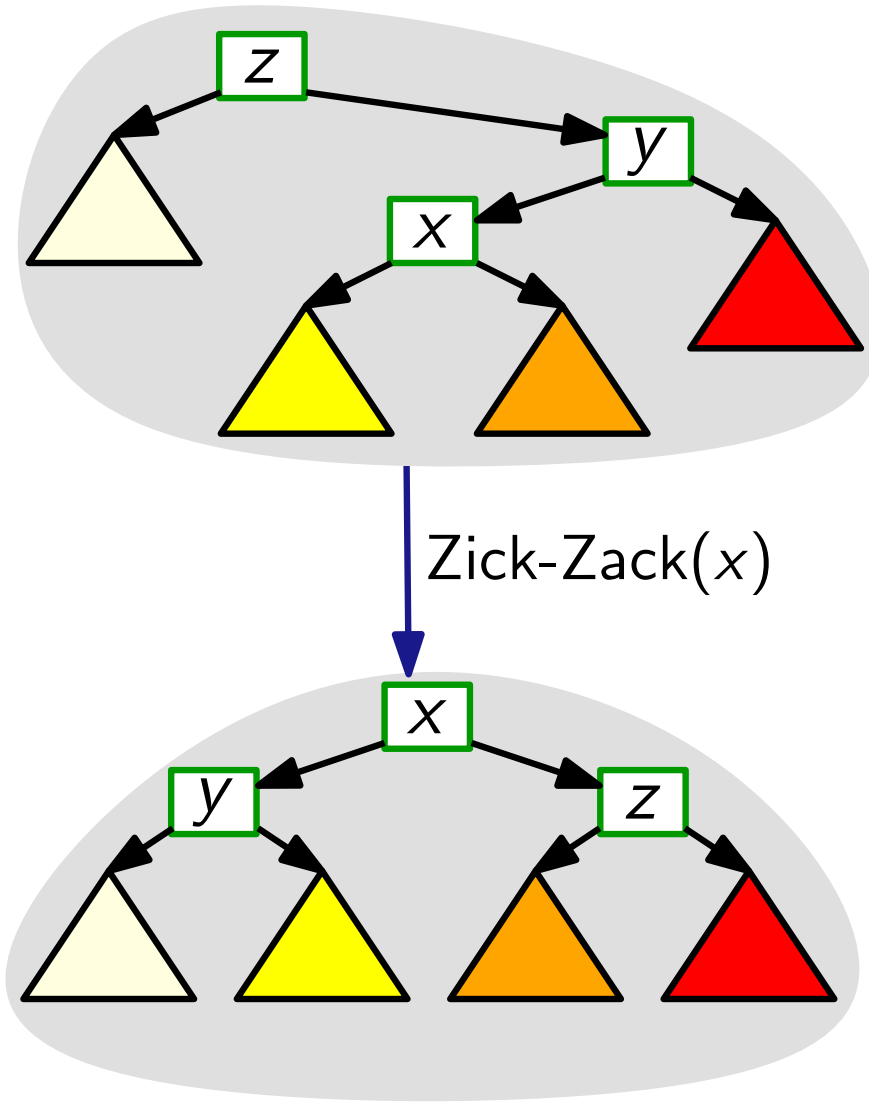
end



Splay

Algorithm: Splay(x)

if $x \neq root$ **then**
  $y =$ Elternknoten von x ;
  if $y = root$ **then**
    if $x < y$ **then** Zick(x);
    if $y < x$ **then** Zack(x);
  else
    $z =$ Elternknoten von y ;
    if $x < y < z$ **then** Zick-Zick(x);
    if $z < y < x$ **then** Zack-Zack(x);
    if $y < x < z$ **then** Zack-Zick(x);
    if $z < x < y$ **then** Zick-Zack(x);
  end
end



Splay

Algorithm: Splay(x)

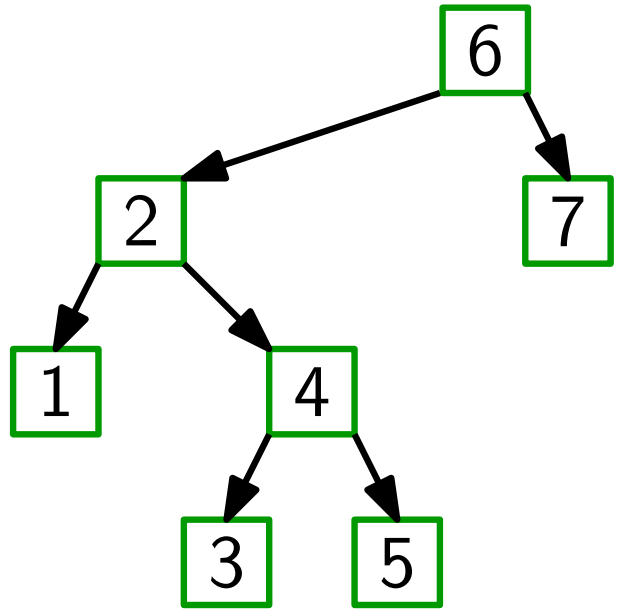
if $x \neq root$ **then**
  $y =$ Elternknoten von x ;
  if $y = root$ **then**
    if $x < y$ **then** Zick(x);
    if $y < x$ **then** Zack(x);
  else
    $z =$ Elternknoten von y ;
    if $x < y < z$ **then** Zick-Zick(x);
    if $z < y < x$ **then** Zack-Zack(x);
    if $y < x < z$ **then** Zack-Zick(x);
    if $z < x < y$ **then** Zick-Zack(x);
  end
  Splay(x)
end

Splay

Algorithm: Splay(x)

```
if x ≠ root then
    y = Elternknoten von x ;
    if y = root then
        if x < y then Zick(x);
        if y < x then Zack(x);
    else
        z = Elternknoten von y ;
        if x < y < z then Zick-Zick(x);
        if z < y < x then Zack-Zack(x);
        if y < x < z then Zack-Zick(x);
        if z < x < y then Zick-Zack(x);
    end
    Splay(x)
end
```

Splay(3):

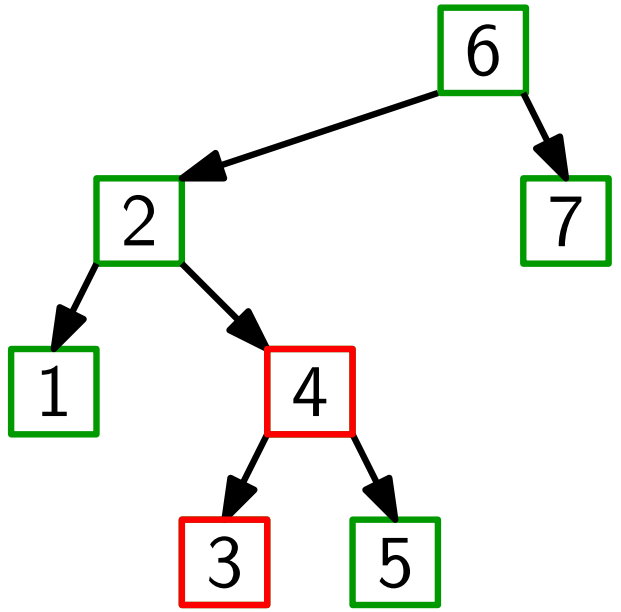


Splay

Algorithm: Splay(x)

```
if x ≠ root then
    y = Elternknoten von x ;
    if y = root then
        if x < y then Zick(x);
        if y < x then Zack(x);
    else
        z = Elternknoten von y ;
        if x < y < z then Zick-Zick(x);
        if z < y < x then Zack-Zack(x);
        if y < x < z then Zack-Zick(x);
        if z < x < y then Zick-Zack(x);
    end
    Splay(x)
end
```

Splay(3):

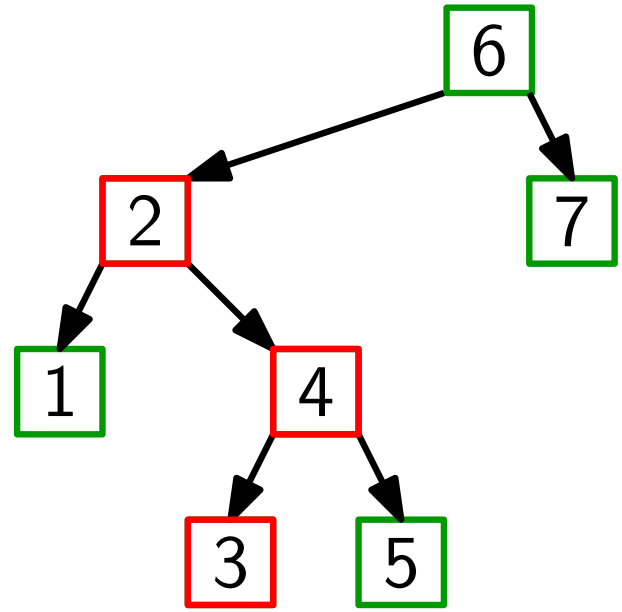


Splay

Algorithm: Splay(x)

if $x \neq root$ **then**
  $y =$ Elternknoten von x ;
  if $y = root$ **then**
    if $x < y$ **then** Zick(x);
    if $y < x$ **then** Zack(x);
  else
    $z =$ Elternknoten von y ;
    if $x < y < z$ **then** Zick-Zick(x);
    if $z < y < x$ **then** Zack-Zack(x);
    if $y < x < z$ **then** Zack-Zick(x);
    if $z < x < y$ **then** Zick-Zack(x);
  end
  Splay(x)
end

Splay(3):

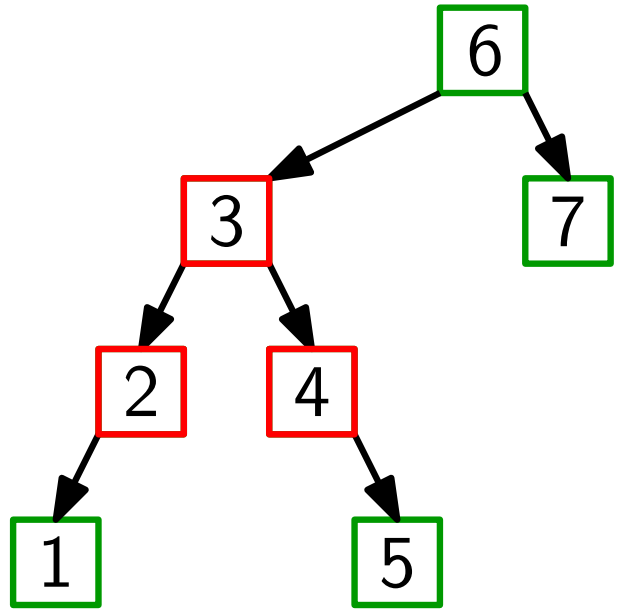


Splay

Algorithm: Splay(x)

if $x \neq root$ **then**
  $y =$ Elternknoten von x ;
  if $y = root$ **then**
    if $x < y$ **then** Zick(x);
    if $y < x$ **then** Zack(x);
  else
    $z =$ Elternknoten von y ;
    if $x < y < z$ **then** Zick-Zick(x);
    if $z < y < x$ **then** Zack-Zack(x);
    if $y < x < z$ **then** Zack-Zick(x);
    if $z < x < y$ **then** Zick-Zack(x);
  end
  Splay(x)
end

Splay(3):

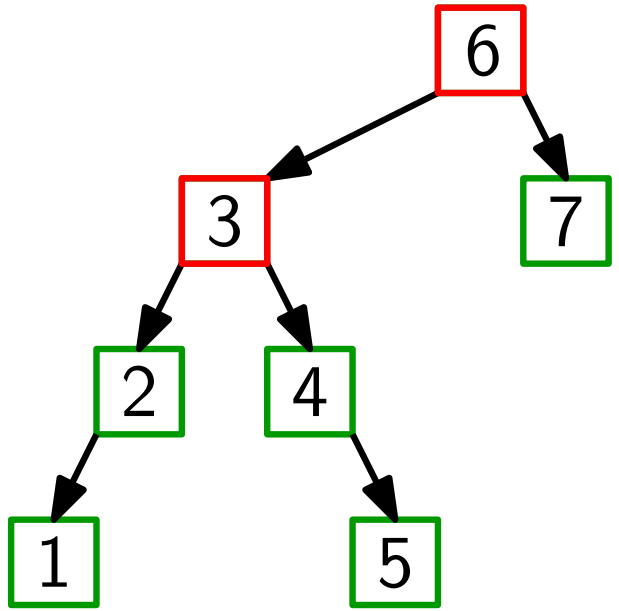


Splay

Algorithm: Splay(x)

if $x \neq root$ **then**
  $y =$ Elternknoten von x ;
  if $y = root$ **then**
    if $x < y$ **then** Zick(x);
    if $y < x$ **then** Zack(x);
  else
    $z =$ Elternknoten von y ;
    if $x < y < z$ **then** Zick-Zick(x);
    if $z < y < x$ **then** Zack-Zack(x);
    if $y < x < z$ **then** Zack-Zick(x);
    if $z < x < y$ **then** Zick-Zack(x);
  end
  Splay(x)
end

Splay(3):

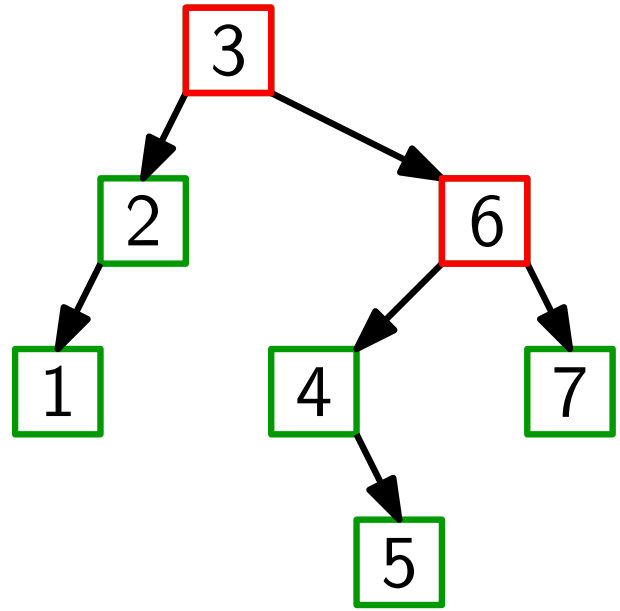


Splay

Algorithm: Splay(x)

```
if x ≠ root then
    y = Elternknoten von x ;
    if y = root then
        if x < y then Zick(x);
        if y < x then Zack(x);
    else
        z = Elternknoten von y ;
        if x < y < z then Zick-Zick(x);
        if z < y < x then Zack-Zack(x);
        if y < x < z then Zack-Zick(x);
        if z < x < y then Zick-Zack(x);
    end
    Splay(x)
end
```

Splay(3):

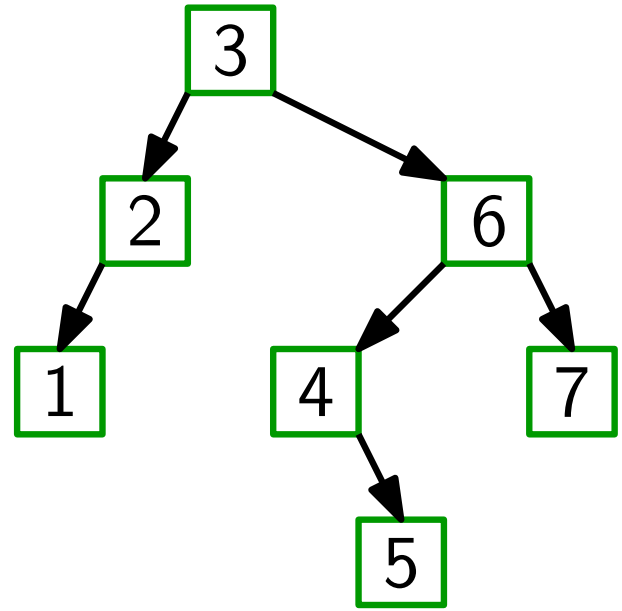


Splay

Algorithm: Splay(x)

if $x \neq root$ **then**
  $y =$ Elternknoten von x ;
  if $y = root$ **then**
    if $x < y$ **then** Zick(x);
    if $y < x$ **then** Zack(x);
  else
    $z =$ Elternknoten von y ;
    if $x < y < z$ **then** Zick-Zick(x);
    if $z < y < x$ **then** Zack-Zack(x);
    if $y < x < z$ **then** Zack-Zick(x);
    if $z < x < y$ **then** Zick-Zack(x);
  end
  Splay(x)
end

Splay(3):



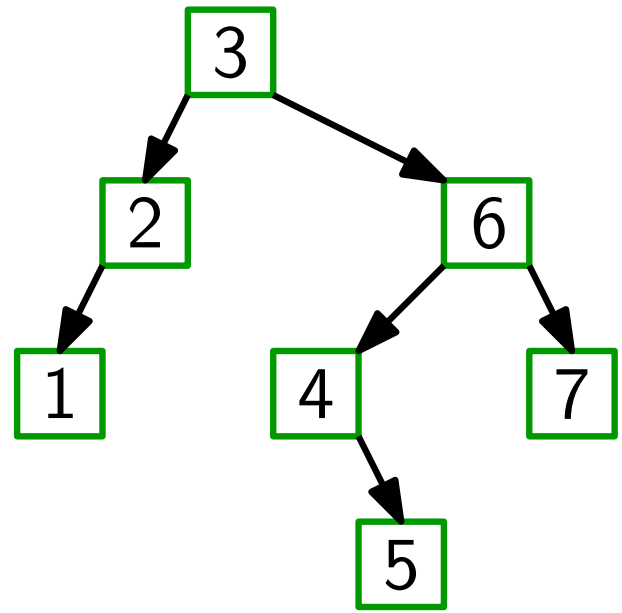
Rufe Splay(x) auf:

Splay

Algorithm: Splay(x)

```
if x ≠ root then
    y = Elternknoten von x ;
    if y = root then
        if x < y then Zick(x);
        if y < x then Zack(x);
    else
        z = Elternknoten von y ;
        if x < y < z then Zick-Zick(x);
        if z < y < x then Zack-Zack(x);
        if y < x < z then Zack-Zick(x);
        if z < x < y then Zick-Zack(x);
    end
    Splay(x)
end
```

Splay(3):



Rufe Splay(x) auf:

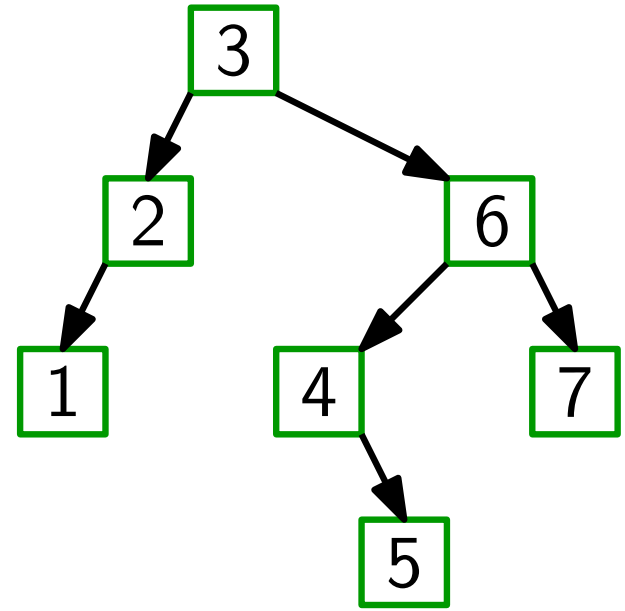
- nach Search(x)

Splay

Algorithm: Splay(x)

if $x \neq root$ **then**
  $y =$ Elternknoten von x ;
  if $y = root$ **then**
    if $x < y$ **then** Zick(x);
    if $y < x$ **then** Zack(x);
  else
    $z =$ Elternknoten von y ;
    if $x < y < z$ **then** Zick-Zick(x);
    if $z < y < x$ **then** Zack-Zack(x);
    if $y < x < z$ **then** Zack-Zick(x);
    if $z < x < y$ **then** Zick-Zack(x);
  end
  Splay(x)
end

Splay(3):



Rufe Splay(x) auf:

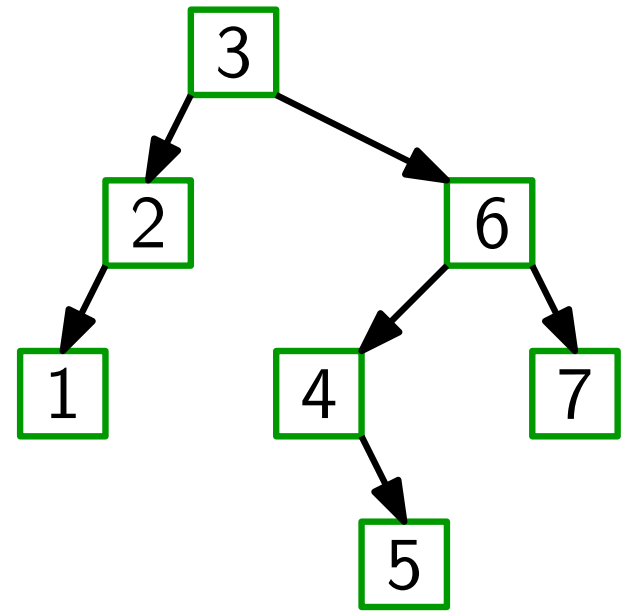
- nach Search(x)
- nach Insert(x)

Splay

Algorithm: Splay(x)

```
if x ≠ root then
    y = Elternknoten von x ;
    if y = root then
        if x < y then Zick(x);
        if y < x then Zack(x);
    else
        z = Elternknoten von y ;
        if x < y < z then Zick-Zick(x);
        if z < y < x then Zack-Zack(x);
        if y < x < z then Zack-Zick(x);
        if z < x < y then Zick-Zack(x);
    end
    Splay(x)
end
```

Splay(3):



Rufe Splay(x) auf:

- nach Search(x)
- nach Insert(x)
- vor Delete(x)

Restliche Vorlesung

Analyse der (amort.) Laufzeit von Splay(x)

Restliche Vorlesung

Analyse der (amort.) Laufzeit von Splay(x)

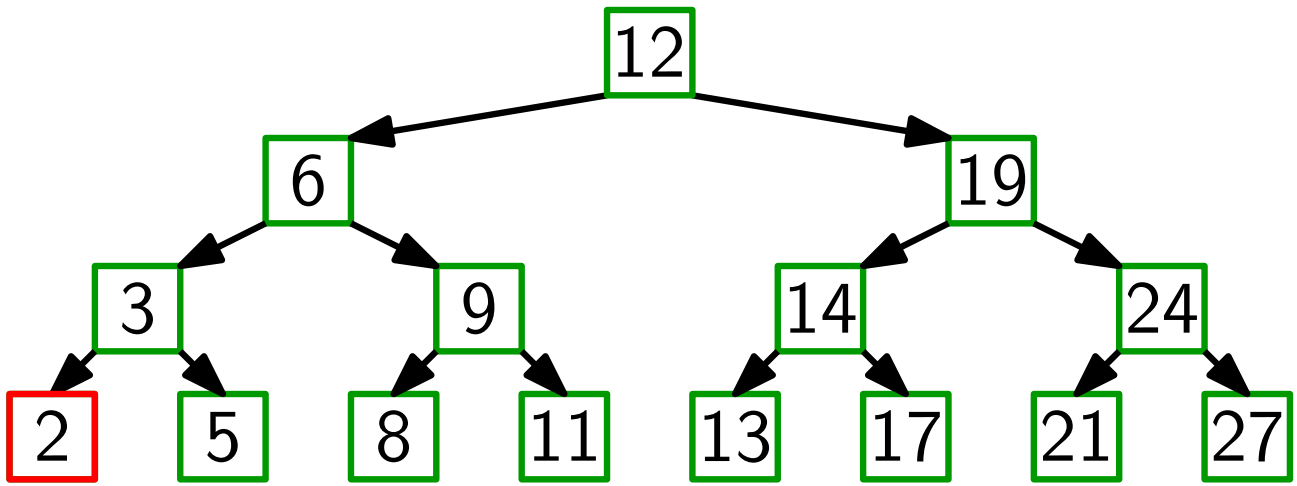
Beweis der statischen Optimalität

Restliche Vorlesung

Analyse der (amort.) Laufzeit von Splay(x)

Beweis der statischen Optimalität

X =2—13—2—13—2...

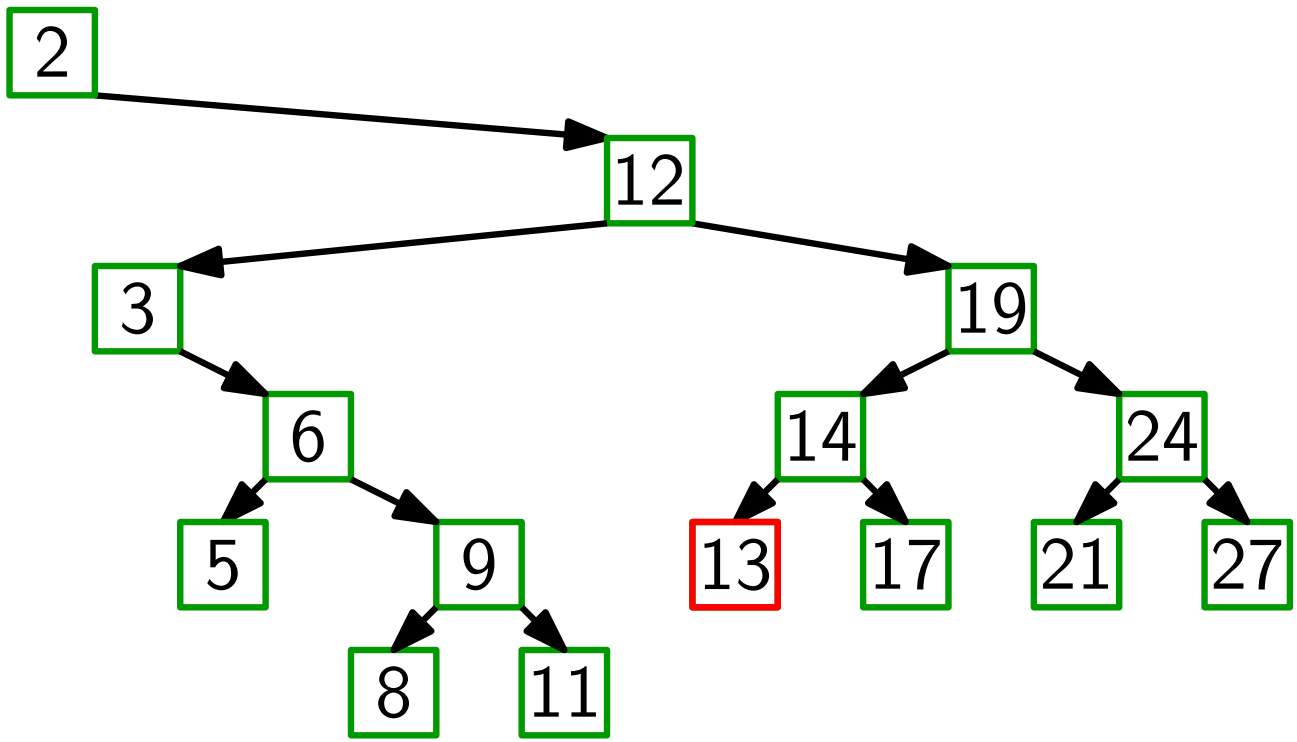


Restliche Vorlesung

Analyse der (amort.) Laufzeit von Splay(x)

Beweis der statischen Optimalität

X =2—13—2—13—2...

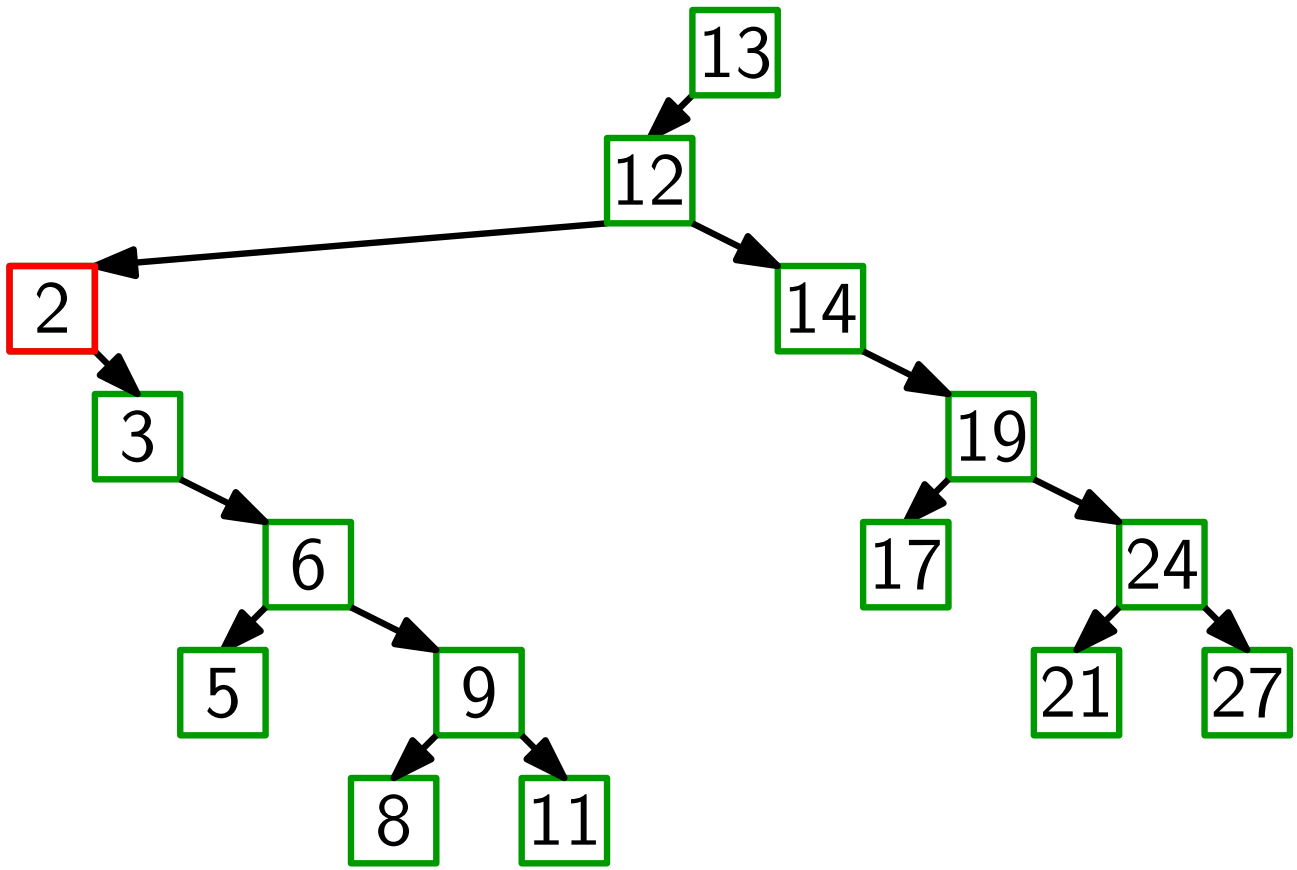


Restliche Vorlesung

Analyse der (amort.) Laufzeit von Splay(x)

Beweis der statischen Optimalität

X =2—13—2—13—2...



Restliche Vorlesung

Analyse der (amort.) Laufzeit von Splay(x)

Beweis der statischen Optimalität

X =2—13—2—13—2...