Algorithmen und Datenstrukturen

Prof. Dr. Ralf Möller

Universität zu Lübeck

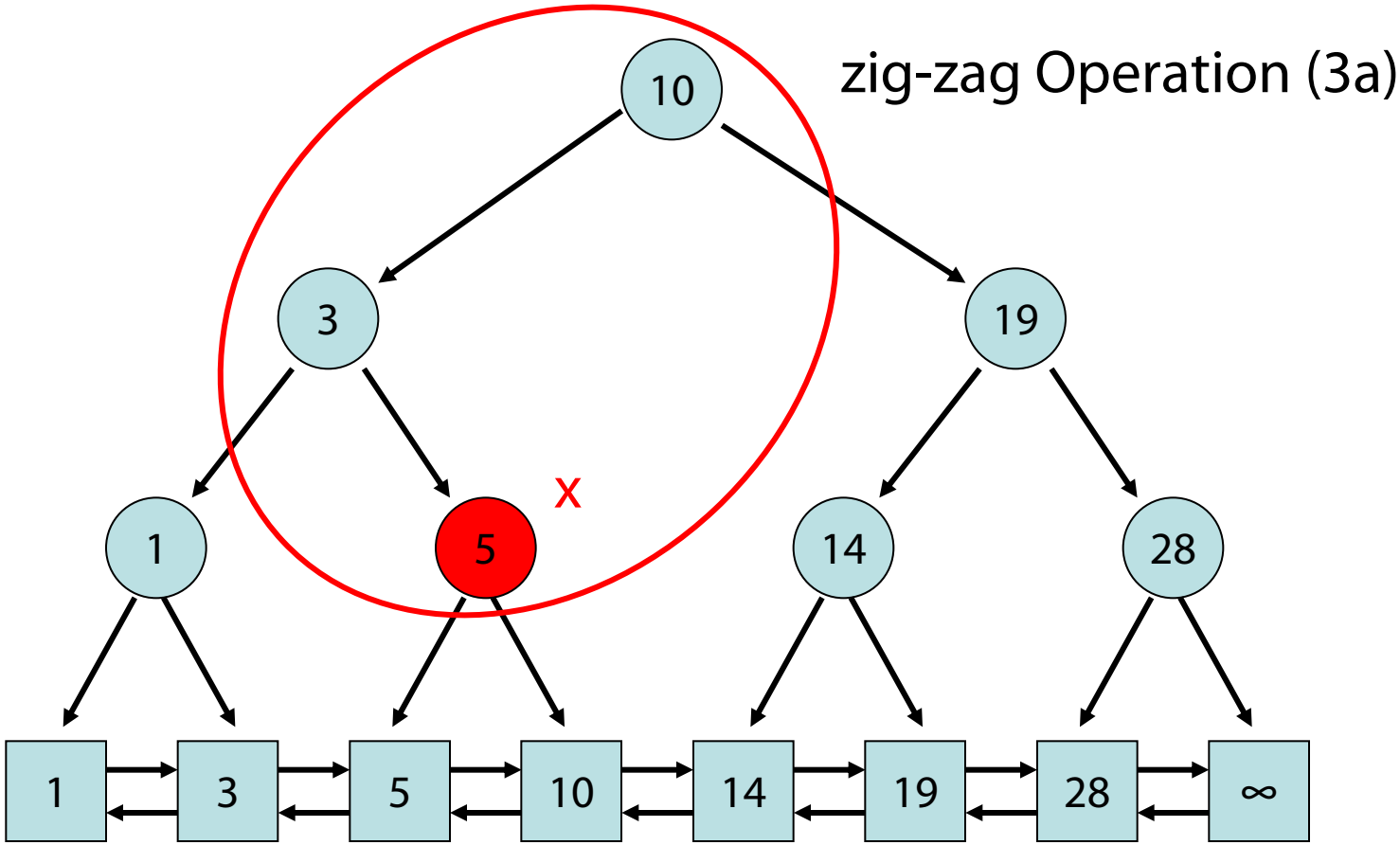
Institut für Informationssysteme

Felix Kuhr (Übungen)

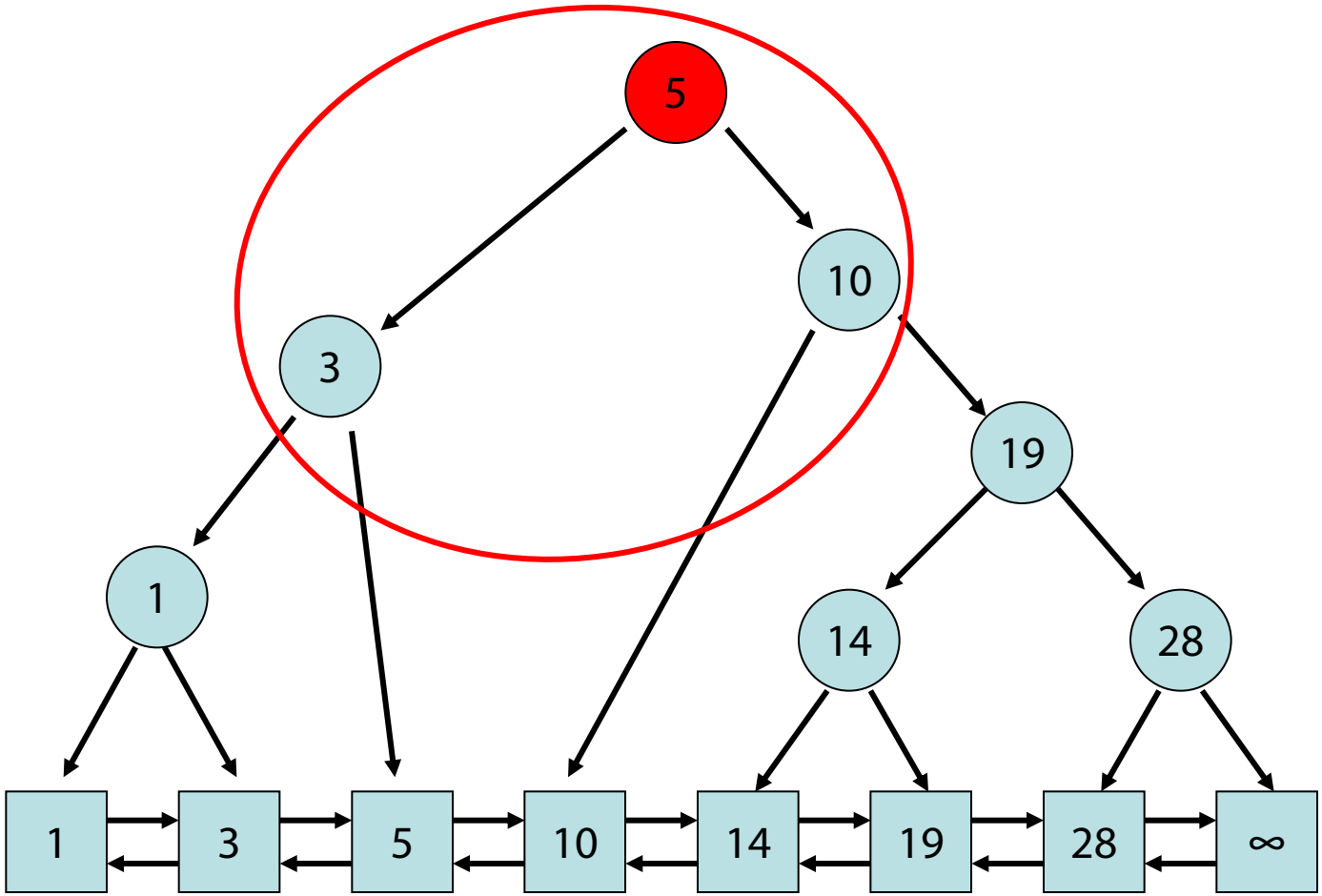
sowie viele Tutoren

Splay-Operation

Beispiel:



Splay-Operation



Splay

procedure splay(x, T)

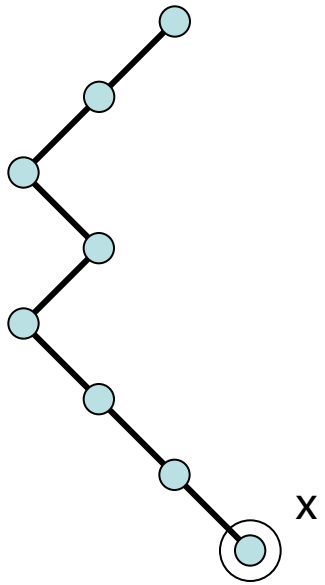
while parentExists(x, T) **do**

// x wandert hoch

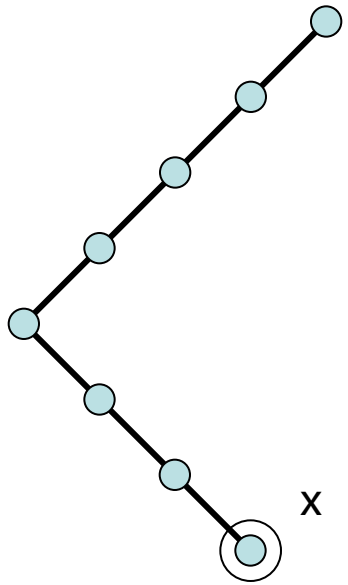
Wende geeignete Splay-Operation an

Splay-Operation: Maximal ein zig

Beispiele:

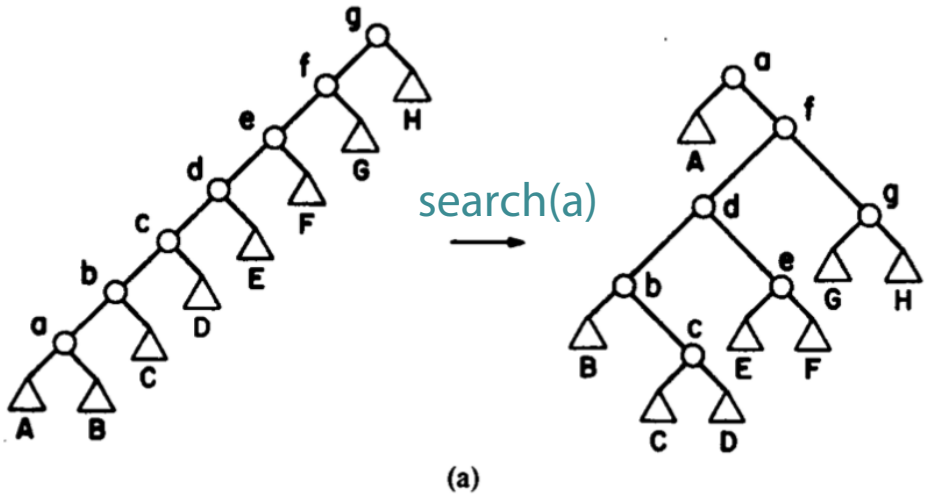


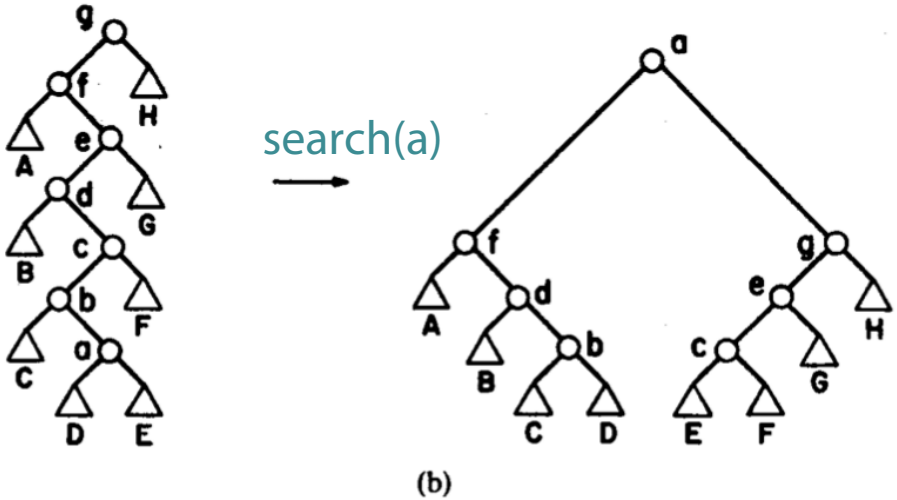
zig-zig, zig-zag, zig-zag, zig



zig-zig, zig-zag, zig-zig, zig

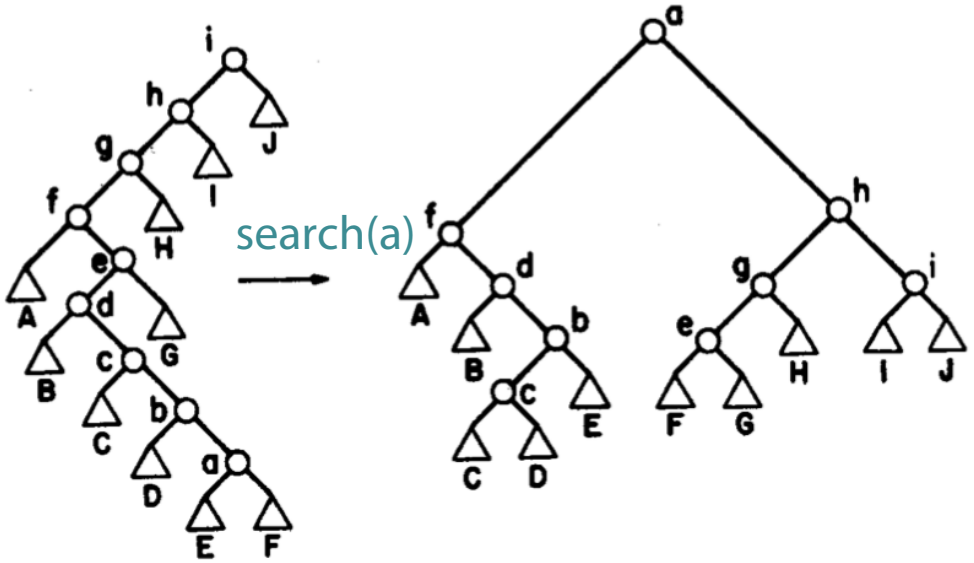
Wirkung der Splay-Operationen





Analyse von Splay-Bäumen

- Über eine Folge von Zugriffen mit search entsteht ein ausgeglichener Baum
- Argumentationstechnik:
  - $T_{search}(n)$ amortisiert in $O(\log n)$



Daniel D. Sleator, Robert Tarjan: Self-Adjusting Binary Search Trees, In: Journal of the ACM (Association for Computing Machinery). 32, Nr. 3, S. 652–686, **1985**

Splay-Bäume: Amortisierte Analyse

search(k)-Operation: (exakte Suche)

- Laufe von Wurzel startend nach unten,
  bis k im Baumknoten gefunden (Abkürzung zur Liste) oder
  bei Liste angekommen (in diesem Fall Schlüssel nicht vorhanden)
- k in Baum: rufe splay(k) auf

Amortisierte Analyse:

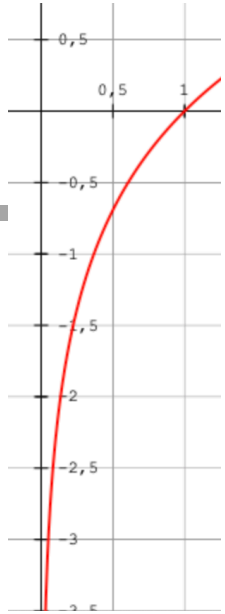
Sei F eine Folge von m Splay-Operationen auf beliebigem Anfangsbaum mit n Elementen (m>n)

$T(F) \leq c + \Sigma_{i=1}^{m} A_{Op_i}(s_{i-1})$

$A_X(s) := \phi(s') - \phi(s) + T_X(s) := \Delta\, \phi(s) + T_X(s)$ für $s \xrightarrow{X} s'$

$T_X(s)$ = Anzahl der Rotationen

Splay-Operation: Potential



- Gewicht von Knoten x: w(x) // z.B. 1/n
  relative Zugriffshäufigkeit ∈ [0, 1]
- Baumgewicht von Baum T mit Wurzel x:
  $tw(x) = \Sigma_{y \in T\_x} w(y) \leq 1$
- Rang von Knoten x: $r(x) = \log(tw(x))$ // für Wurzel = 0
- Potential von Baum T: $\phi(T) = \Sigma_{x \in T} r(x)$

Behauptung "amortisierte Splaykosten": Sei T ein Splay-Baum mit Wurzel u und x ein Knoten in T.
Die amortisierten Kosten für splay(x, T)
sind max. $1+3(r(u)-r(x)) \in O(\log(tw(u)/tw(x)))$

T_x denotiere alle Knoten in T unter (und einschl.) x

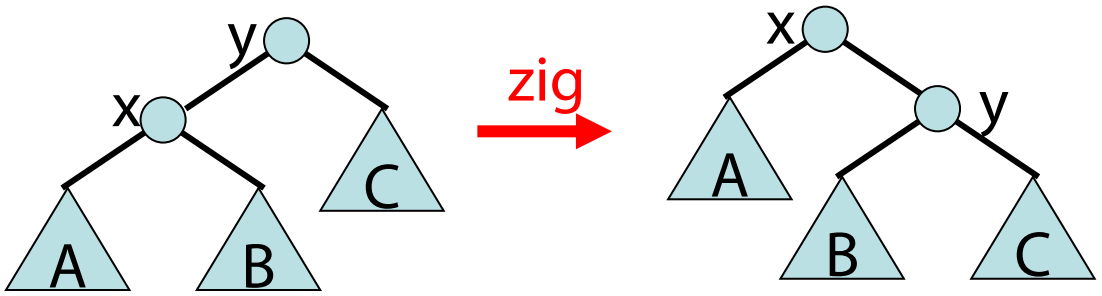
Splay-Operation

Begründung für Korrektheit der Behauptung:

Induktion über die Folge der Rotationen.

- r und tw : Rang und Gewicht vor Rotation
- r' und tw': Rang und Gewicht nach Rotation

1. Fall:



Laufzeit
(1 Rotation)

Amortisierte Kosten:

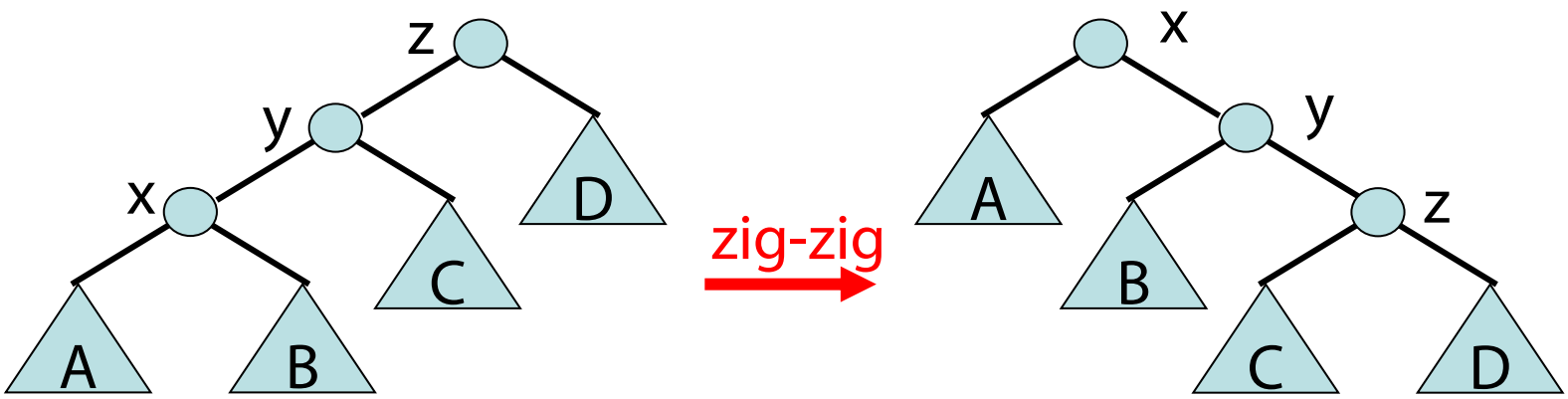
$\leq 1+r'(x)+r'(y)-r(x)-r(y) \leq 1+r'(x)-r(x)$ da $r'(y) \leq r(y)$

$\leq 1+3(r'(x)-r(x))$ da $r'(x) \geq r(x)$

Änderung von ϕ

Splay-Operation

2. Fall:



Amortisierte Kosten:

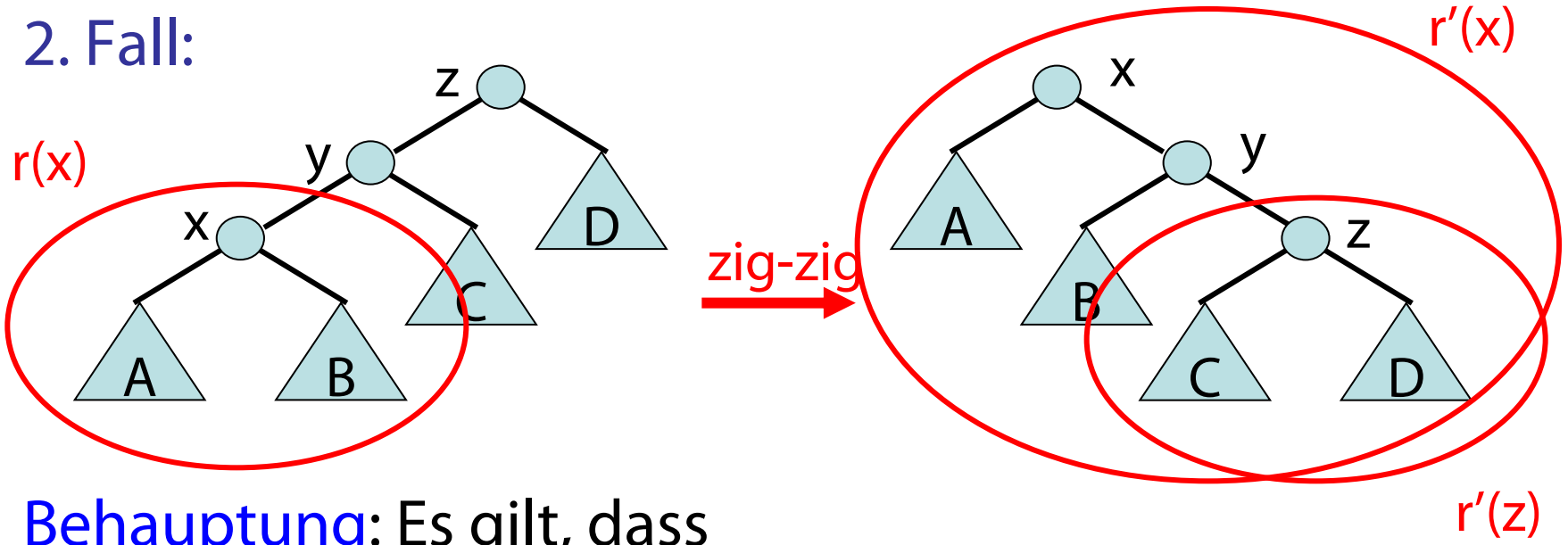
$\leq 2+r'(x)+r'(y)+r'(z)-r(x)-r(y)-r(z)$

$= 2+r'(y)+r'(z)-r(x)-r(y)$ da $r'(x)=r(z)$

$\leq 2+r'(x)+r'(z)-2r(x)$ da $r'(x)\geq r'(y)$ und $r(y)\geq r(x)$

Splay-Operation

2. Fall:



Behauptung: Es gilt, dass

$$2+r'(x)+r'(z)-2r(x) \leq 3(r'(x)-r(x))$$

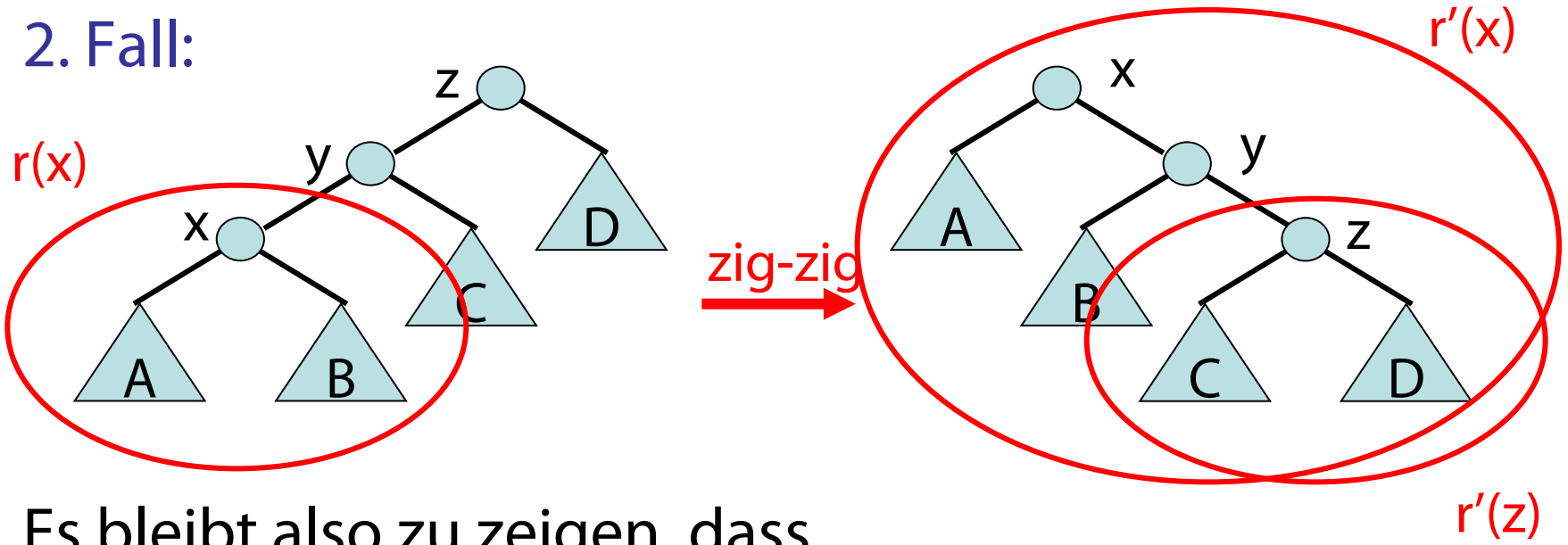
d.h.

$$r(x)+r'(z) \leq 2(r'(x)-1)$$

und damit $r(x)-r'(x)+r'(z)-r'(x) \leq -2$

Splay-Operation

2. Fall:



Es bleibt also zu zeigen, dass

$$r(x)-r'(x) + r'(z)-r'(x) \leq -2$$

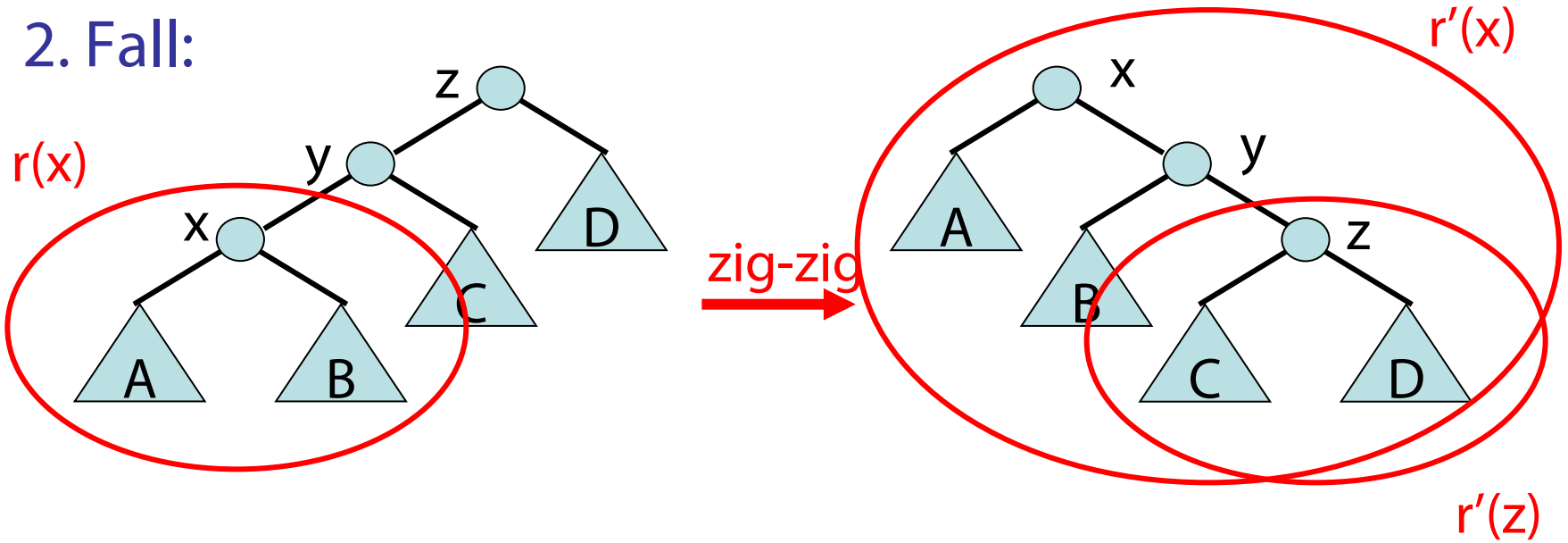
Anders gesagt:

$$\log(tw(x)) - \log(tw'(x)) + \log(tw'(z)) - \log(tw'(x)) \leq -2$$

$$\log(tw(x)/tw'(x)) + \log(tw'(z)/tw'(x)) \leq -2$$

Splay-Operation

2. Fall:



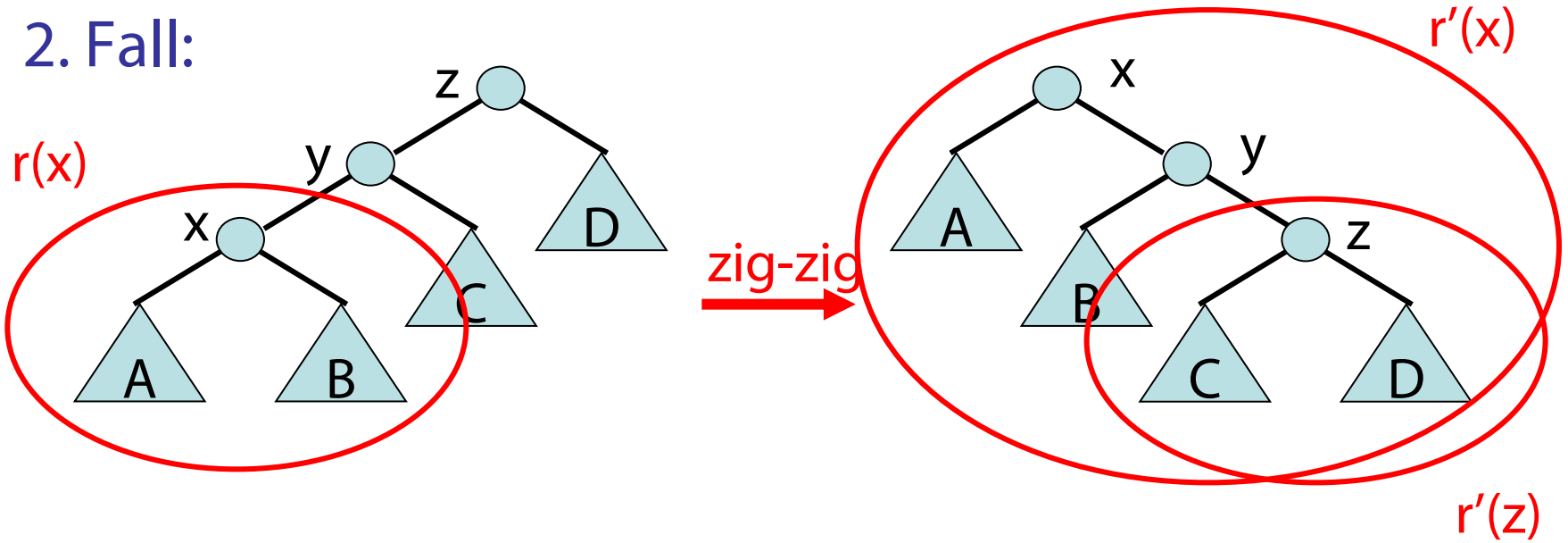
$$\log(tw(x)/tw'(x)) + \log(tw'(z)/tw'(x)) \leq -2$$

Beh: $tw(x)/tw'(x) > 0$ $tw'(z)/tw'(x) > 0$

$tw(x)/tw'(x) + tw'(z)/tw'(x) < 1$

Splay-Operation

2. Fall:



$$tw(x)/tw'(x) \quad + \quad tw'(z)/tw'(x) \quad < 1$$

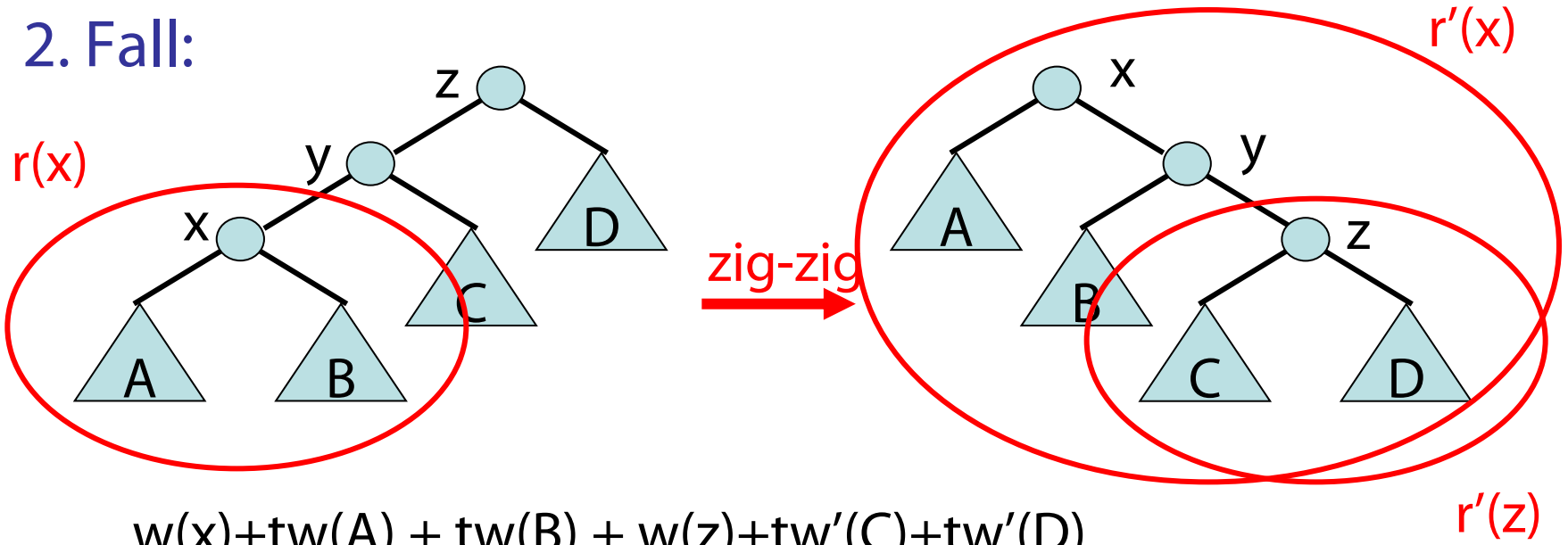
$$tw(x) + tw'(z) < tw'(x)$$

$$w(x)+tw(A) + tw(B) + w(z)+tw'(C)+tw'(D)$$

$$< w(x)+tw'(A)+ w(y) + tw'(B) + w(z)+tw'(C)+tw'(D)$$

Splay-Operation

2. Fall:



$$w(x)+tw(A) + tw(B) + w(z)+tw'(C)+tw'(D)$$
$$< w(x)+tw'(A) + w(y) + tw'(B) + w(z)+tw'(C)+tw'(D)$$

$0 < w(y) \rightarrow$ true

Also: $tw(x)/tw'(x) + tw'(z)/tw'(x) < 1$

Splay-Operation

Wir wissen nun: $tw(x)/tw'(x) + tw'(z)/tw'(x) < 1$

Zurück zu: $\log(tw(x)/tw'(x)) + \log(tw'(z)/tw'(x)) \leq -2$

Betrachte die Funktion $f(x,y)=\log x + \log y$.

Wir zeigen: $f(x,y) \leq -2$ für alle $x,y>0$ mit $x+y<1$.

Splay-Operation

Behauptung: Die Funktion $f(x,y)=\log x + \log y$ hat in dem Bereich $x,y>0$ mit $x+y \leq 1$ im Punkt $(½,½)$ ihr Maximum.

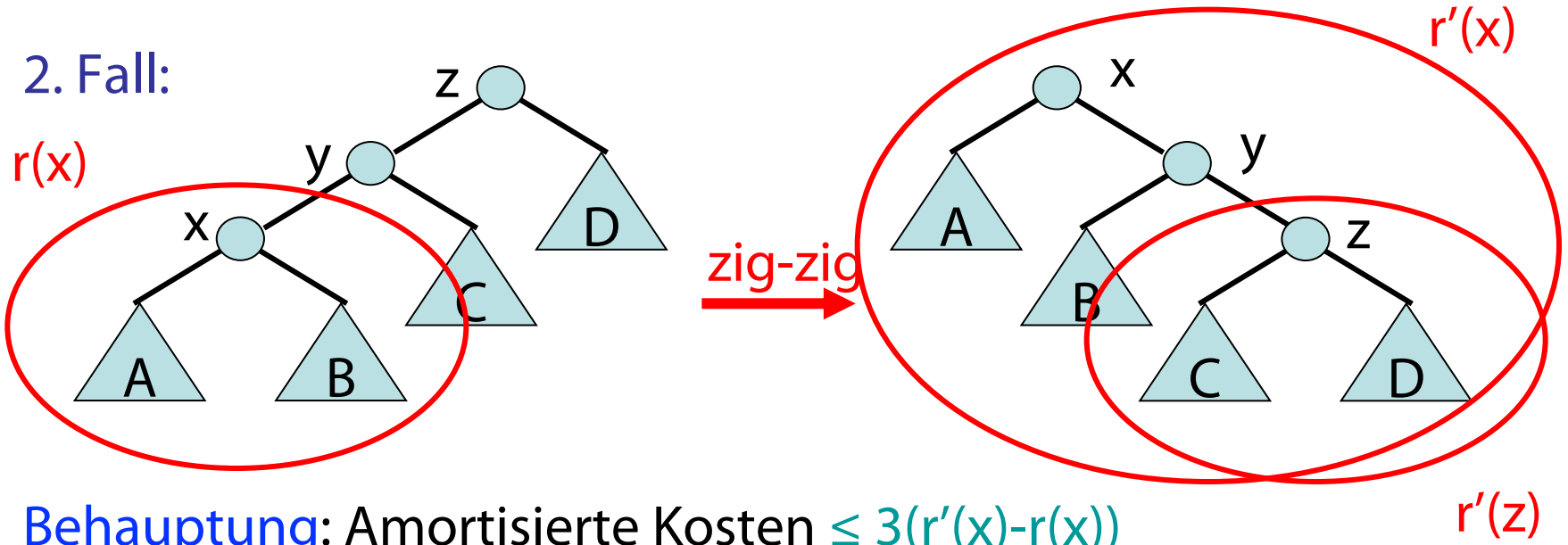
Begründung für Korrektheit:

- Da die Funktion log streng monoton wachsend ist, kann sich das Maximum nur auf dem Geradensegment $x+y=1$, $x,y>0$, befinden.
- Neues Maximierungsproblem: betrachte $g(x) = \log x + \log (1-x)$
- Einzige Nullstelle von $g'(x) = 1/x - 1/(1-x)$ ist $x=1/2$.
- Für $g''(x)= -(1/x^2 + 1/(1-x)^2))$ gilt $g''(1/2) < 0$.
- Also hat Funktion f im Punkt $(½,½)$ ihr Maximum.

Splay-Operation

Zurück zum

2. Fall:



Behauptung: Amortisierte Kosten $\leq 3(r'(x)-r(x))$

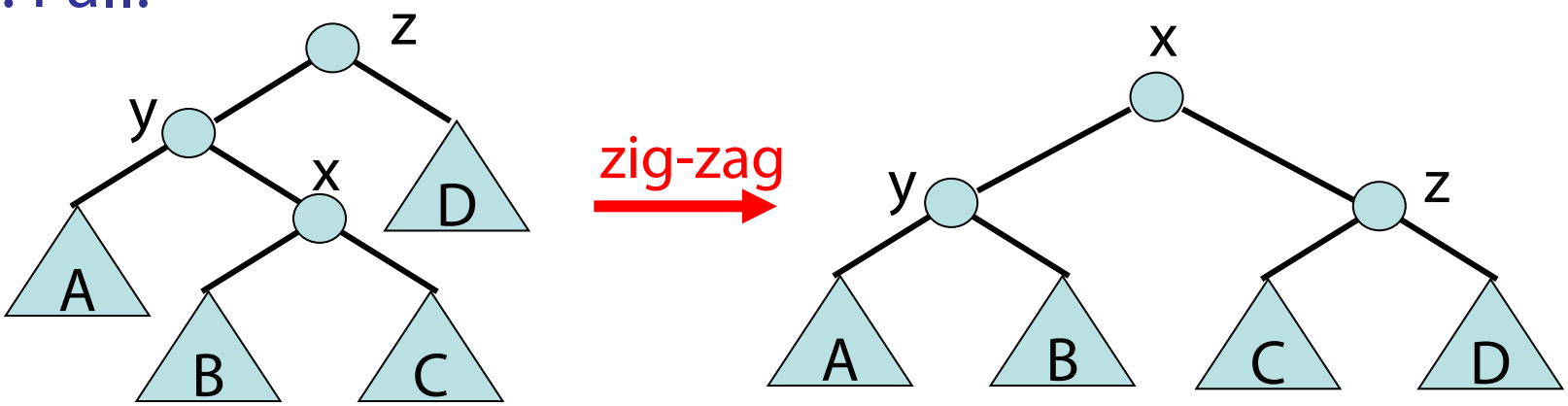
Zwischenresultate:

Behauptung gilt, wenn: $\log x + \log y \leq -2$ für alle $x,y>0$ mit $x+y<1$
Maximum von $\log x + \log y$ für $x+y=1$ bei $(½, ½)$

Folgerung: Die Behauptung ist korrekt,
da $\log(½) = -1$ und damit $\log(½) + \log(½) \leq -2$

Splay-Operation

3. Fall:



Amortisierte Kosten:

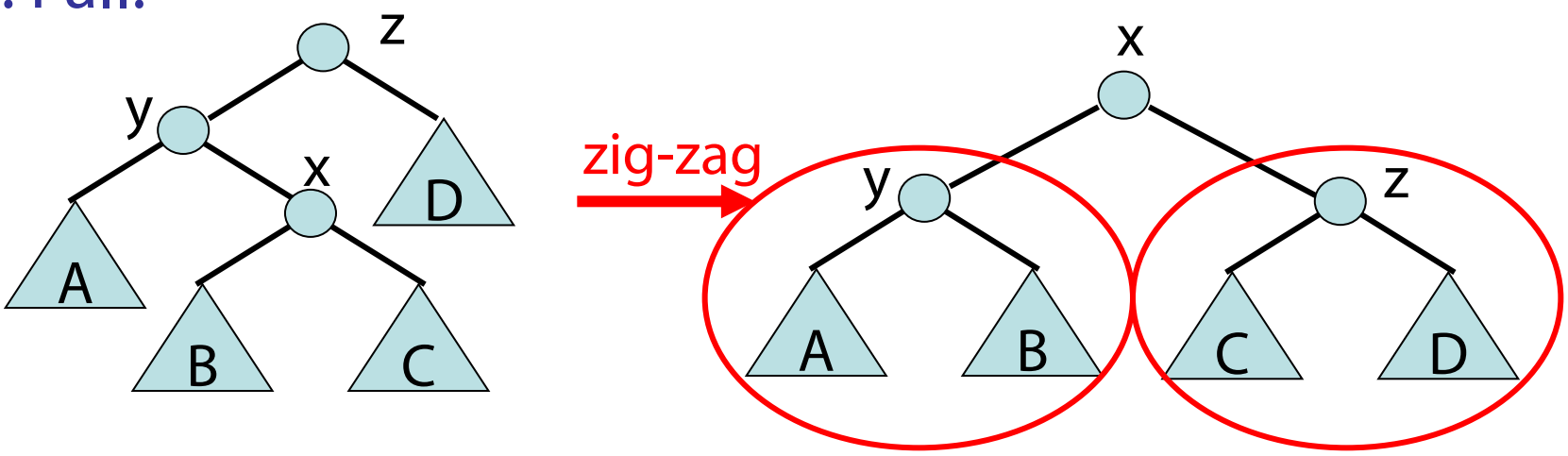
$\leq 2+r'(x)+r'(y)+r'(z)-r(x)-r(y)-r(z)$

$\leq 2+r'(y)+r'(z)-2r(x)$ da $r'(x)=r(z)$ und $r(x)\leq r(y)$

$\leq 2(r'(x)-r(x))$ behaupten wir, und…

Splay-Operation

3. Fall:



…es gilt:

$$2+r'(y)+r'(z)-2r(x) \leq 2(r'(x)-r(x))$$

gdw $2r'(x)-r'(y)-r'(z) \geq 2$

gdw $r'(y)-r'(x)+r'(z)-r'(x) \leq -2$

Analog zu Fall 2 gilt das, und $2(r'(x)-r(x)) \leq 3(r'(x)-r(x))$

Splay-Operation

Beweis von Behauptung “amortisierte Splaykosten”:
(Fortsetzung)

Induktion über die Folge der Rotationen.

- r und tw : Rang und Gewicht vor Rotation
- r’ und tw’: Rang und Gewicht nach Rotation
- Für jede Rotation ergeben sich amortisierte Kosten von max. $1+3(r'(x)-r(x))$ (Fall 1, zig, NB: max 1 zig) bzw. $3(r'(x)-r(x))$ (Fälle 2 und 3)
- Aufsummierung der Kosten pro Zugriff ergibt max. $1 + \Sigma_{Rot.} 3(r'(x)-r(x)) = 1+3(r(u)-r(x))$

Splay-Operation

- Baumgewicht von Baum T mit Wurzel x:
  $tw(x) = \Sigma_{y \in T(x)} w(y)$
- Rang von Knoten x: $r(x) = \log(tw(x))$
- Potential von Baum T: $\phi(T) = \Sigma_{x \in T} r(x)$
- Sei $W = \Sigma_{x \in T} w(x)$ und
  w_i das Gewicht von Knoten x in i-tem Aufruf von search.

"amortisierte Splaykosten": Sei T ein Splay-Baum mit Wurzel u und x ein Knoten in T. Die amortisierten Kosten für splay(x,T) sind max. $1+3(r(u)-r(x)) = 1+3 \cdot \log (tw(u)/tw(x))$.

Korollar: Für m search-Operationen der Folge F sind die amortisierten Kosten $A(F) = m+3\Sigma_{i=1}^{m} \log (W/w_i)$.

Splay-Baum: Balance-Theorem

Balance Theorem: " Splay-Bäume arbeiten wie ausgeglichene Bäume"
Die Laufzeit für m search Operationen in einem n-elementigen Splay-Baum T ist in

$$O(m \cdot \log n) \quad (\text{NB: } m > n)$$

Begründung für Gültigkeit des Theorems:

- Sei $w(x) = 1/n$ für alle Knoten x in T (gleiche relative Zugriffshäufigkeit).
- Dann ist $W=1$ und $r(x) \leq \log W = \log 1$ für alle x in T.
- Erinnerung: für eine Operationsfolge F ist die Laufzeit $T(F) \leq A(F) + \phi(s_0)$ für amortisierte Kosten A und Anfangszustand s_0
- $\phi(s_0) = \Sigma_{x \in T}\, r_0(x) \leq n \log 1 = 0$
- Aus dem Korollar von oben ergibt sich das Theorem:
- $T(F) \leq m+3\Sigma_{i=1}^{m} \log (W/w_i) = m + 3\Sigma_{i=1}^{m} \log(1/ (1/n))$
  $= m + 3\Sigma_{i=1}^{m} \log n = m + 3m \log n \in O(m \log n)$

Conclusio

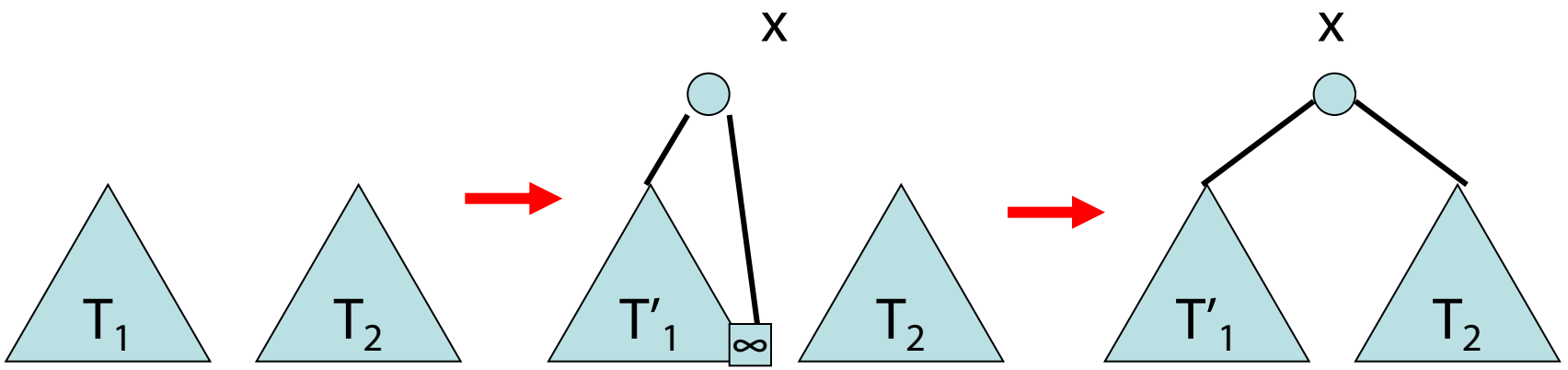
- $T_{search}(n)$ liegt amortisiert in $O(\log n)$
- Deutung: Splay-Bäume arbeiten so effektiv wie ausgeglichene Bäume bei langen Search-Sequenzen
- Vernünftig, wenn…
- … es als sinnvoll erachtet wird, dass so modelliert wird:
  - Gewicht von Knoten x: w(x) // z.B. 1/n
    relative Zugriffshäufigkeit $\in [0, 1]$
  - Baumgewicht von Baum T mit Wurzel x:
    $tw(x) = \Sigma_{y \in T\_x} w(y) \leq 1$
  - Rang von Knoten x: $r(x) = \log(tw(x))$ // für Wurzel = 0
  - Potential von Baum T: $\phi(T) = \Sigma_{x \in T} r(x)$

Daniel D. Sleator, Robert Tarjan: Self-Adjusting Binary Search Trees, In: Journal of the ACM (Association for Computing Machinery). 32, Nr. 3, S. 652–686, **1985**

Splay-Baum Operationen

Annahme: zwei Splay-Bäume T_1 und T_2 mit key(x)<key(y) für alle $x \in T_1$ und $y \in T_2$.

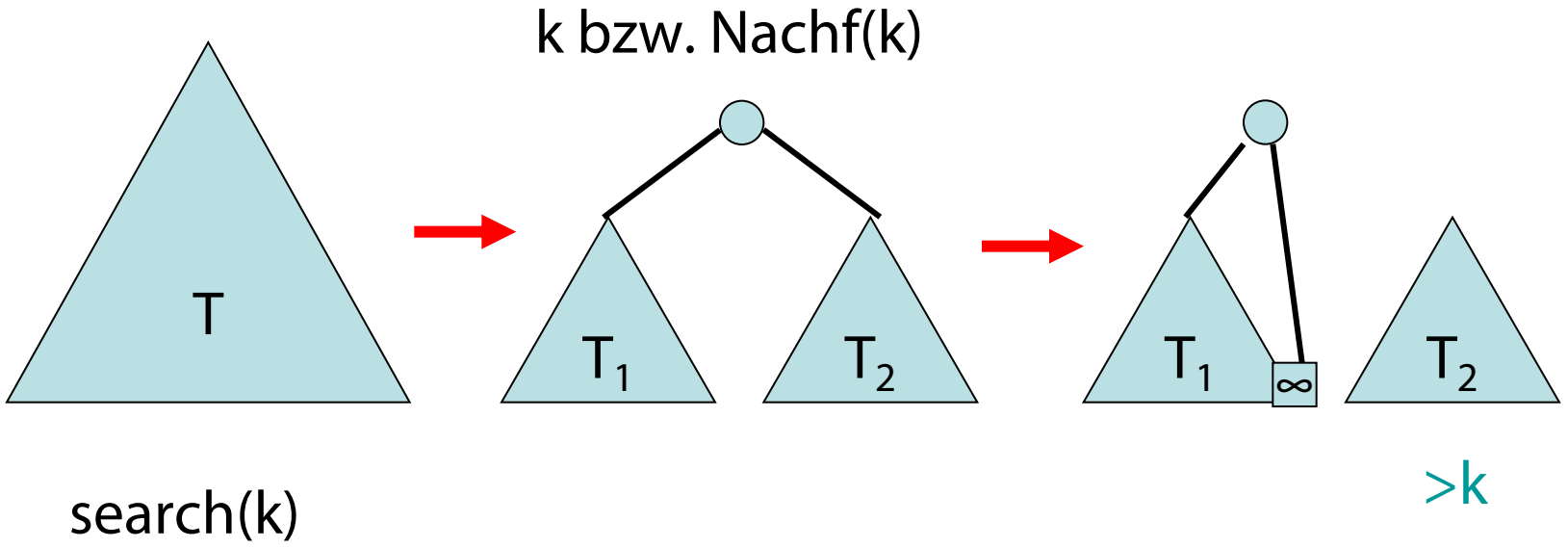
merge(T_1,T_2):



search(x), $x < \infty$ max. in T_1

Splay-Baum Operationen

split(k,T):



Splay-Baum Operationen

insert(e):

- insert wie im Binärbaum
- splay-Operation, um key(e) in Wurzel zu verschieben

delete(k):

- führe search(k) aus (bringt k in die Wurzel)
- entferne Wurzel und führe merge(T_1,T_2) der beiden Teilbäume durch

Zusammenfassung

- Für Splay-Bäume liegt $T_{search}(n)$ amortisiert in $O(\log n)$
- $T_{insert}(n)$ liegt amortisiert in $O(\log n)$
- $T_{delete}(n)$ liegt amortisiert in $O(\log n)$