



Binäre Suchbäume

Mengen, Funktionalität, Binäre
Suchbäume, Heaps, Treaps



Mengen

- **Ziel:** Aufrechterhalten einer Menge (hier: ganzer Zahlen) unter folgenden Operationen:



Mengen

- **Ziel:** Aufrechterhalten einer Menge (hier: ganzer Zahlen) unter folgenden Operationen:
 - Einfügen eines Elements
 - Löschen eines Elements



Mengen

- **Ziel:** Aufrechterhalten einer Menge (hier: ganzer Zahlen) unter folgenden Operationen:
 - Einfügen eines Elements
 - Löschen eines Elements
 - Suchen eines Elements



Mengen

- **Ziel:** Aufrechterhalten einer Menge (hier: ganzer Zahlen) unter folgenden Operationen:
 - Einfügen eines Elements
 - Löschen eines Elements
 - Suchen eines Elements
- **Anforderung:** effizient auch bei grossen Mengen!



Mengen: Anwendungen (I)

- Telefonbuch (Menge von Namen mit zugehörigen Telefonnummern)
 - **Einfügen:** Neue Telefonanschlüsse
 - **Löschen:** Aufgegebene Telefonanschlüsse
 - **Suchen:** Telefonnummer einer Person



Mengen: Anwendungen (II)

- Nutzerverwaltung (Menge von Nutzern mit Passwörtern und/oder weiteren Informationen)
 - **Einfügen:** Neue Nutzer
 - **Löschen:** Ex-Nutzer
 - **Suchen:** Einloggen

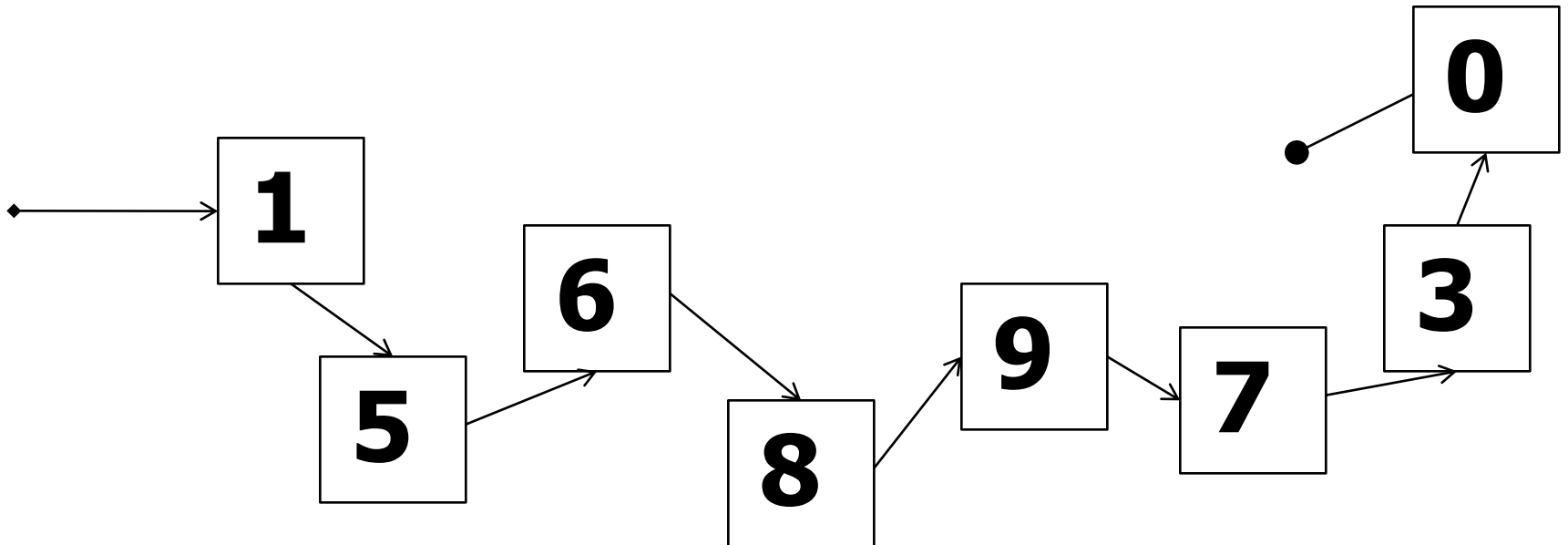


Mengen: Anwendungen (II)

- Nutzerverwaltung (Menge von Nutzern mit Passwörtern und/oder weiteren Informationen)
 - **Einfügen:** Neue Nutzer
 - **Löschen:** Ex-Nutzer
 - **Suchen:** Einloggen
- **Effizienz:** Keine Wartezeit beim Einloggen, auch bei Millionen von Nutzern.

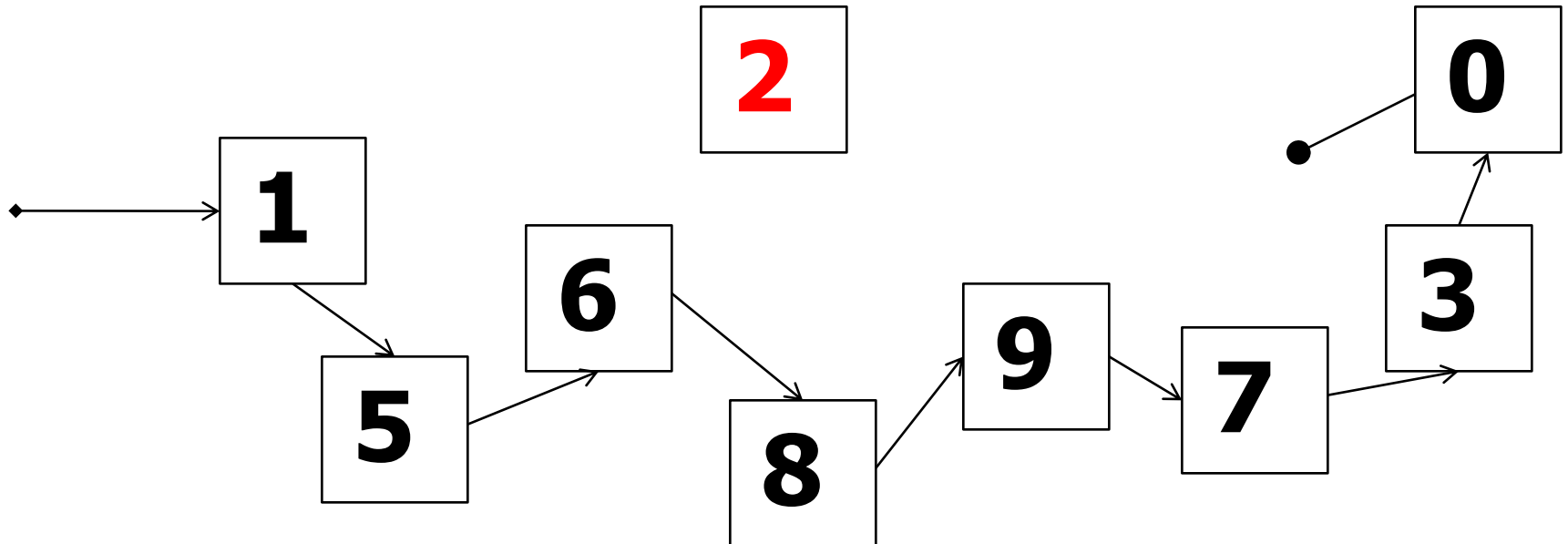
Mengen: Lösung mit Listen ?

- Menge wird als Liste gespeichert.



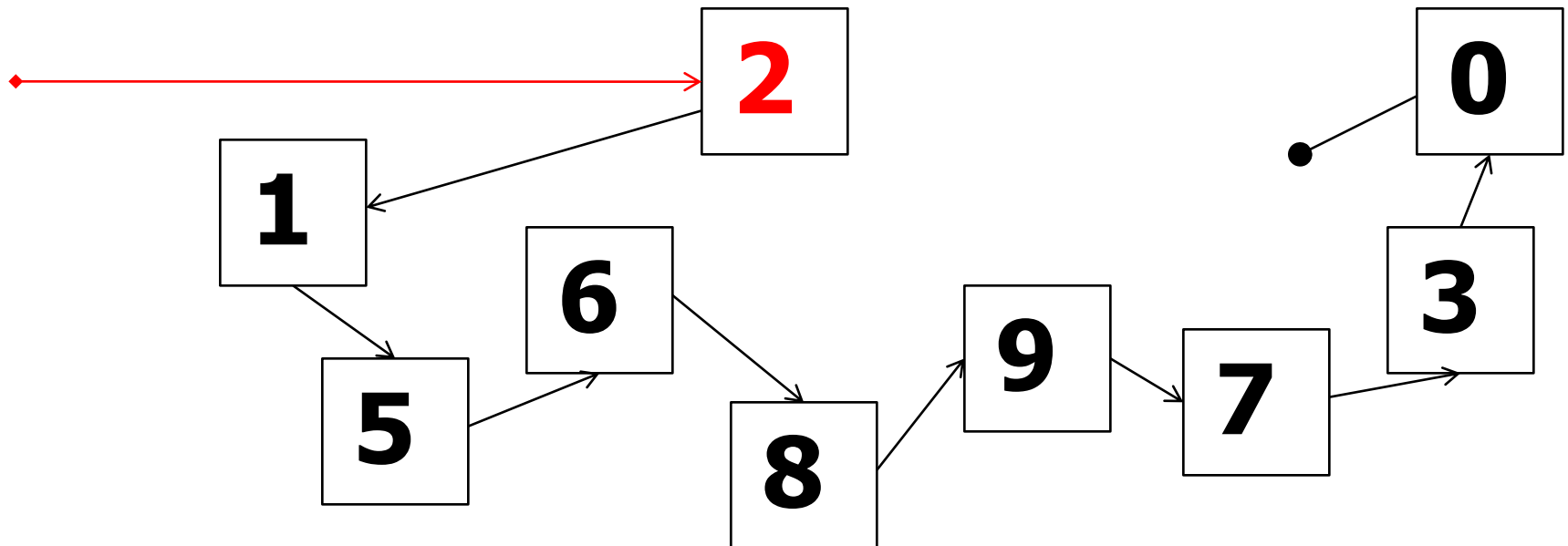
Mengen: Lösung mit Listen ?

- Menge wird als Liste gespeichert.
 - **Einfügen:** zum Beispiel vorne anfügen (effizient)



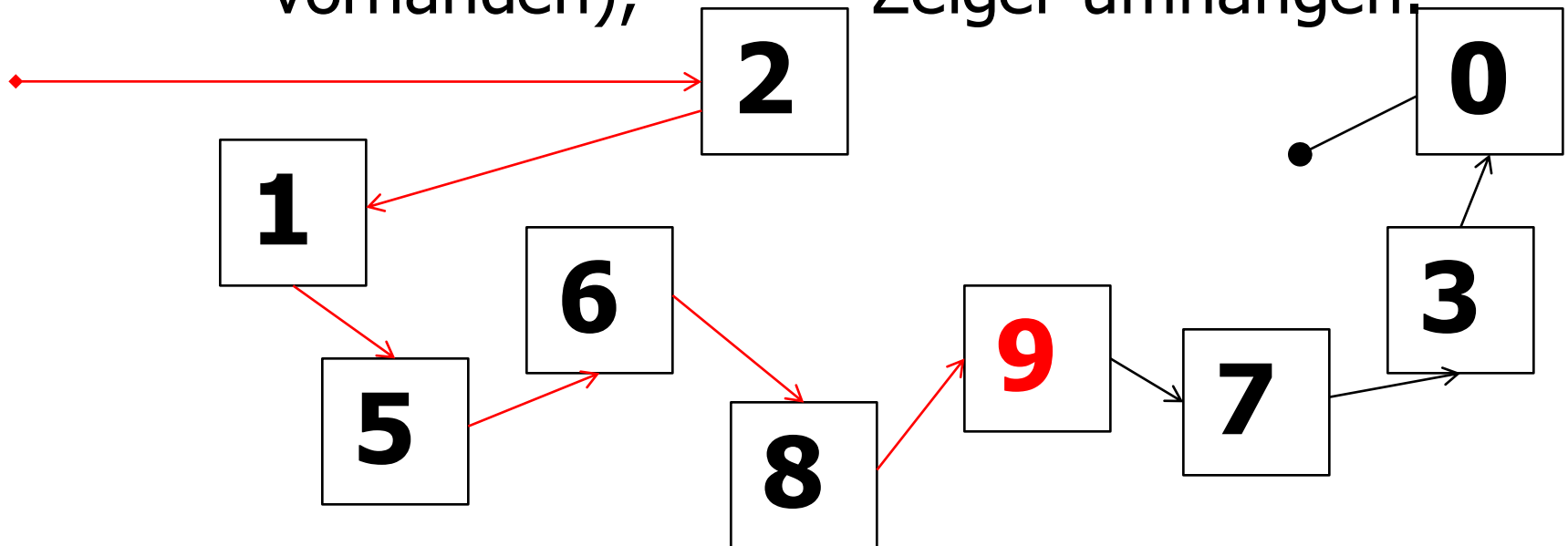
Mengen: Lösung mit Listen ?

- Menge wird als Liste gespeichert.
 - **Einfügen:** zum Beispiel vorne anfügen (effizient)



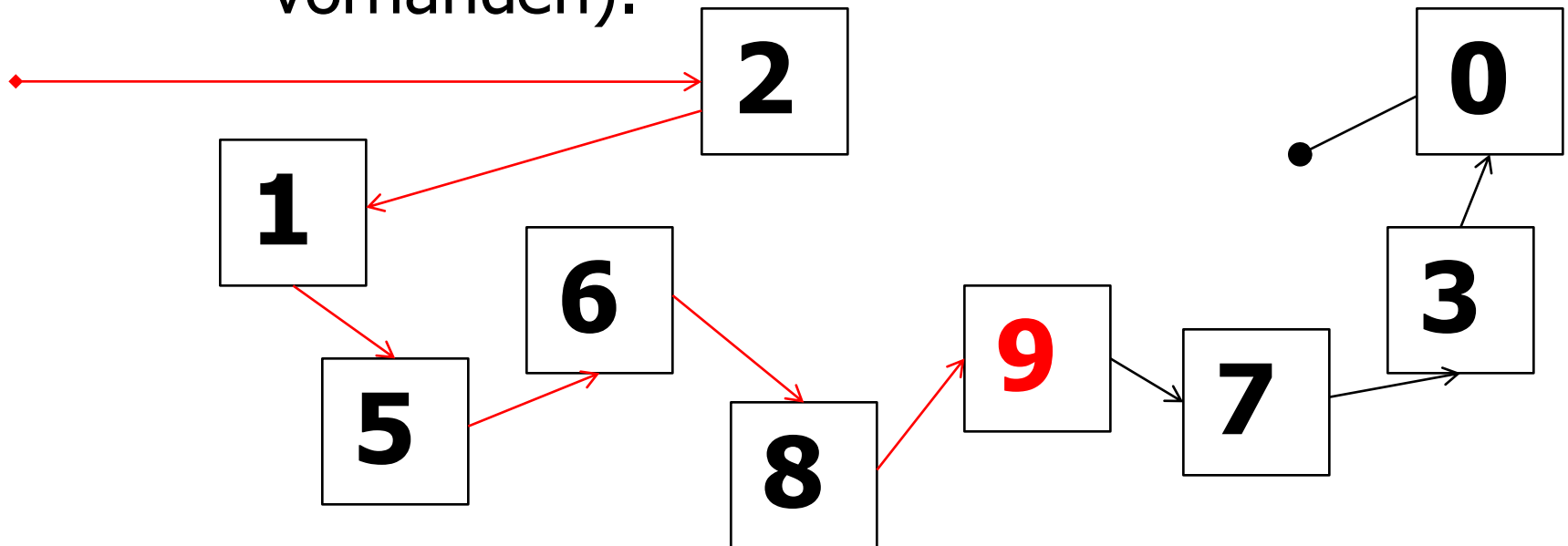
Mengen: Lösung mit Listen ?

- Menge wird als Liste gespeichert.
 - **Löschen:** Durchlaufen der Liste bis zum Element (oder bis zum Ende, falls nicht vorhanden), Zeiger umhängen.



Mengen: Lösung mit Listen ?

- Menge wird als Liste gespeichert.
 - **Suchen:** Durchlaufen der Liste bis zum Element (oder bis zum Ende, falls nicht vorhanden).





Mengen: Lösung mit Listen ?

- Falls die Menge n Elemente enthält, so ist der Aufwand fürs Suchen eines Elements im schlimmsten Fall proportional zu n .



Mengen: Lösung mit Listen ?

- Falls die Menge n Elemente enthält, so ist der Aufwand fürs Suchen eines Elements im schlimmsten Fall proportional zu n .
- Soziales Netzwerk: Zeit zum Einloggen wäre proportional zur Anzahl der Nutzer.





Mengen: Lösung mit Listen ?

- Beispiel: Suche nach den letzten 100 Elementen in einer Liste von 10,000,000 Elementen



Mengen: Lösung mit Listen ?

- Beispiel: Suche nach den letzten 100 Elementen in einer Liste von 10,000,000 Elementen

```
List l;
```

```
for (int i=0; i<10000000; ++i)  
    l.push_front(i);
```

```
// ...and search for i = 0..99
```

```
for (int i=0; i<100; ++i)  
    l.find(i);
```



Mengen: Lösung mit Listen ?

- Beispiel: Suche nach den letzten 100 Elementen in einer Liste von 10,000,000 Elementen

```
List l;
```

```
for (int i=0; i<10000000; ++i)
```

```
    l.push_front(i);
```

```
// ...and search for i = 0..99
```

```
for (int i=0; i<100; ++i)
```

```
    l.find(i);
```

6.5 Sekunden



Mengen: Lösung mit Listen ?

- Beispiel: Suche nach den letzten 100 Elementen in einer Liste von 10,000,000 Elementen

```
List l;  
for (int i=0; i<10000000; ++i)  
    l.push_front(i);  
  
// ...and search for i = 0..99  
for (int i=0; i<100; ++i)  
    l.find(i);
```

Facebook hat
1,400,000,000
Nutzer, *eine*
Suche würde
10 Sekunden
dauern!

6.5 Sekunden



Mengen: Lösung mit Binären Suchbäumen.

- Falls die Menge n Elemente enthält, so ist der Aufwand fürs Suchen eines Elements im schlimmsten Fall proportional zu $\log_2 n$.
- Facebook: Zeit zum Einloggen ist proportional zum **Logarithmus** der Anzahl der Nutzer.



Mengen: Lösung mit Binären Suchbäumen.

- Falls die Menge n Elemente enthält, so ist der Aufwand fürs Suchen eines Elements im schlimmsten Fall proportional zu $\log_2 n$.
- Facebook: Zeit zum Einloggen ist proportional zum **Logarithmus** der Anzahl der Nutzer.

$$\log_2 (1,400,000,000) \approx 30.$$



Mengen: Lösung mit Binären Suchbäumen.

- Falls die Menge n Elemente enthält, so ist der Aufwand fürs Suchen eines Elements im schlimmsten Fall proportional zu $\log_2 n$.
- Auch Einfügen und Löschen geht in Zeit proportional to $\log_2 n$.



Binärbäume, Definition und Terminologie

- Ein binärer Baum ist entweder leer,...

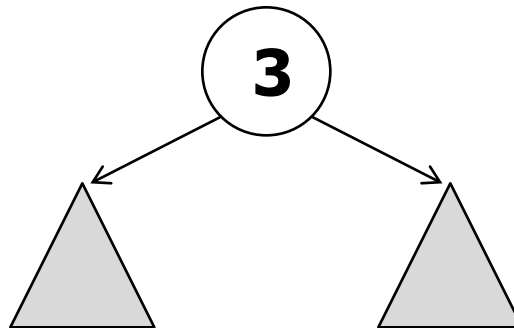
Binärbäume, Definition und Terminologie



- Ein binärer Baum ist entweder leer,...
- oder er besteht aus einem **Knoten** (Kreis)
 - mit einem **Schlüssel** (hier: ganze Zahl),...

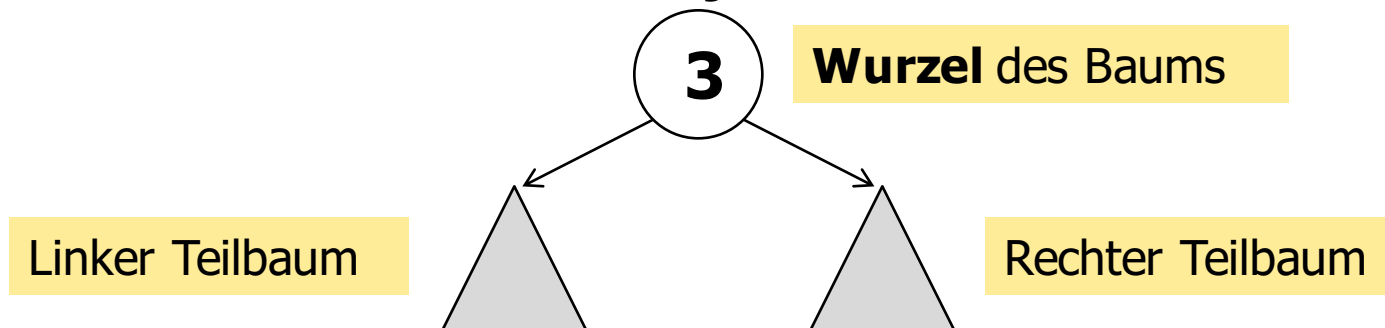
Binärbäume, Definition und Terminologie

- Ein binärer Baum ist entweder leer,...
- oder er besteht aus einem **Knoten** (Kreis)
 - mit einem **Schlüssel** (hier: ganze Zahl),...
 - und einem **linken** und **rechten Teilbaum**, die jeweils Binärbäume sind.



Binärbäume, Definition und Terminologie

- Ein binärer Baum ist entweder leer,...
- oder er besteht aus einem **Knoten** (Kreis)
 - mit einem **Schlüssel** (hier: ganze Zahl),...
 - und einem **linken** und **rechten Teilbaum**, die jeweils Binärbäume sind.

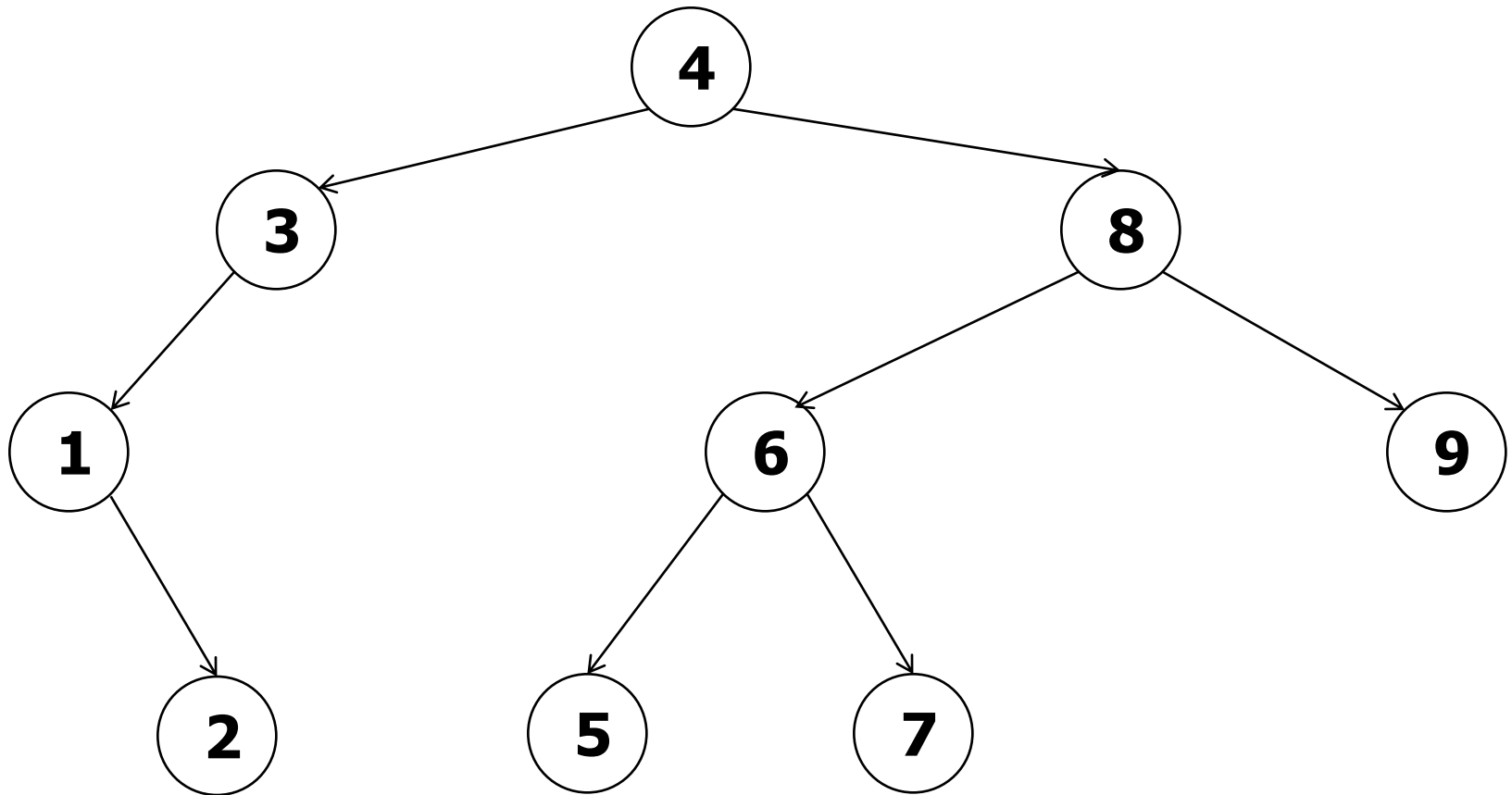




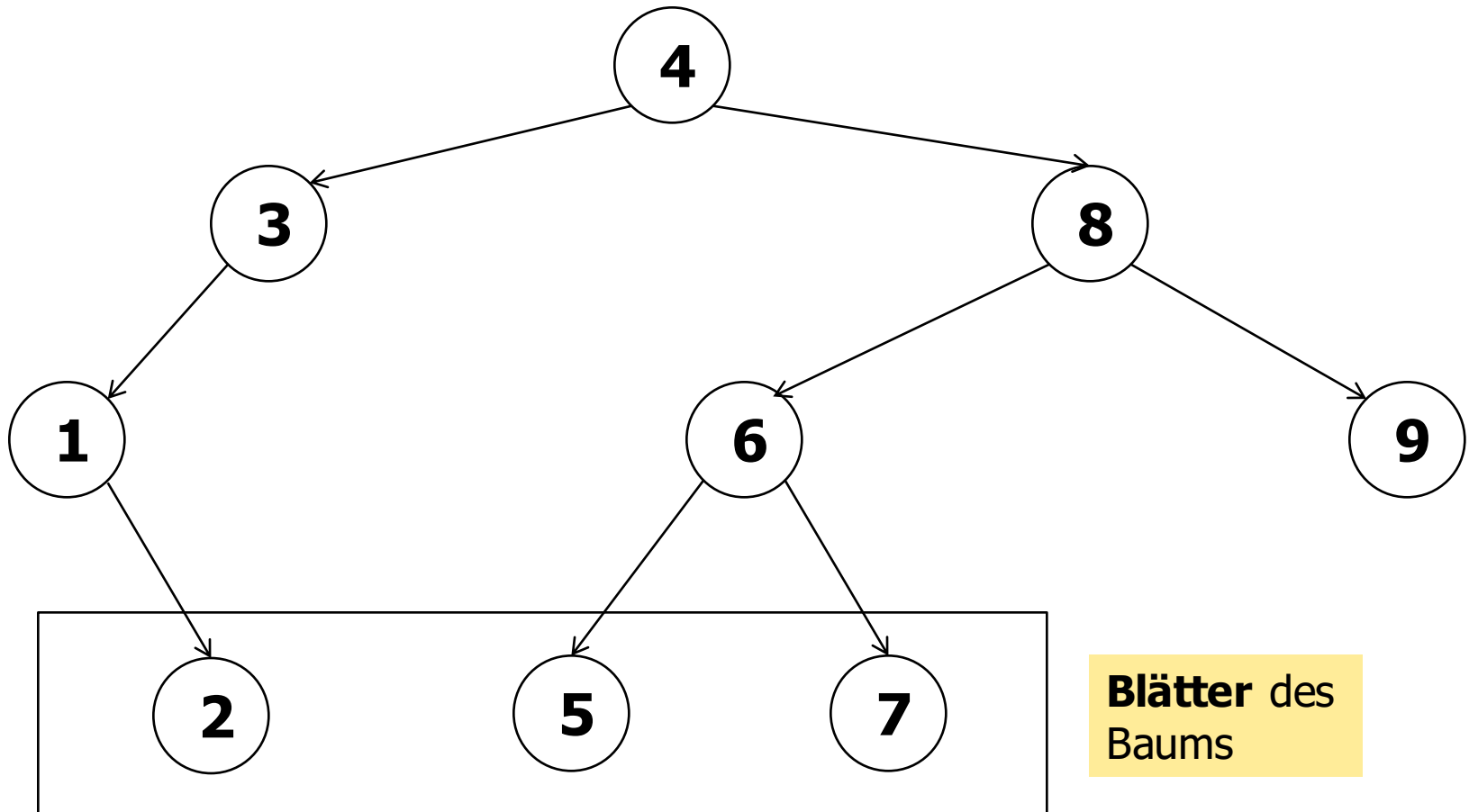
Binäre Suchbäume

- Ein Binärbaum heisst **binärer Suchbaum**, wenn er leer ist, oder wenn folgende Bedingungen gelten:
 - Sowohl der linke als auch der rechte Teilbaum sind binäre Suchbäume.
 - Der Schlüssel der Wurzel ist
 - $>$ alle Schlüssel im linken Teilbaum, und
 - \leq alle Schlüssel im rechten Teilbaum.

Binäre Suchbäume: Beispiel



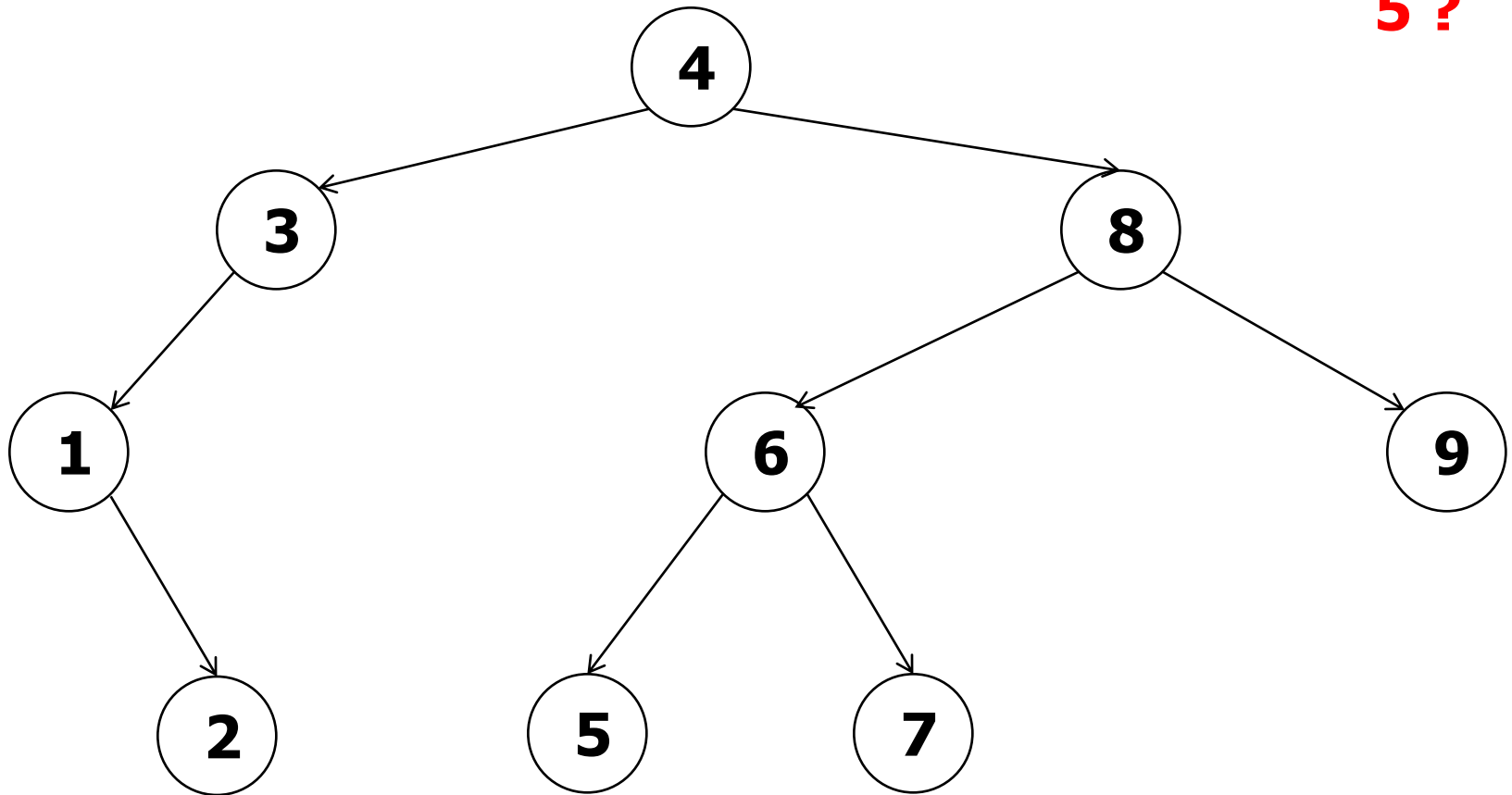
Binäre Suchbäume: Beispiel



Binäre Suchbäume:

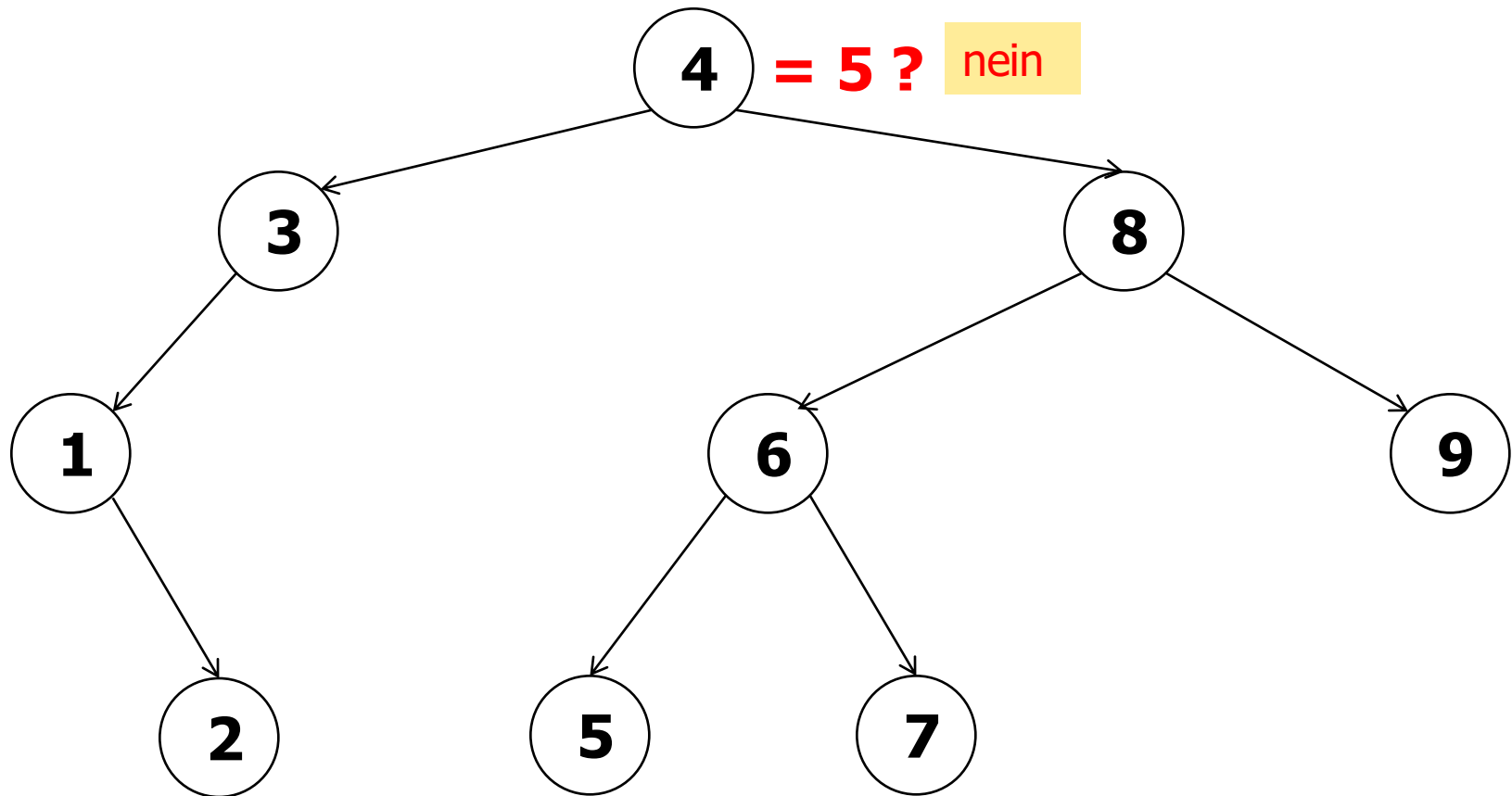
Suchen nach einem Schlüssel

5?

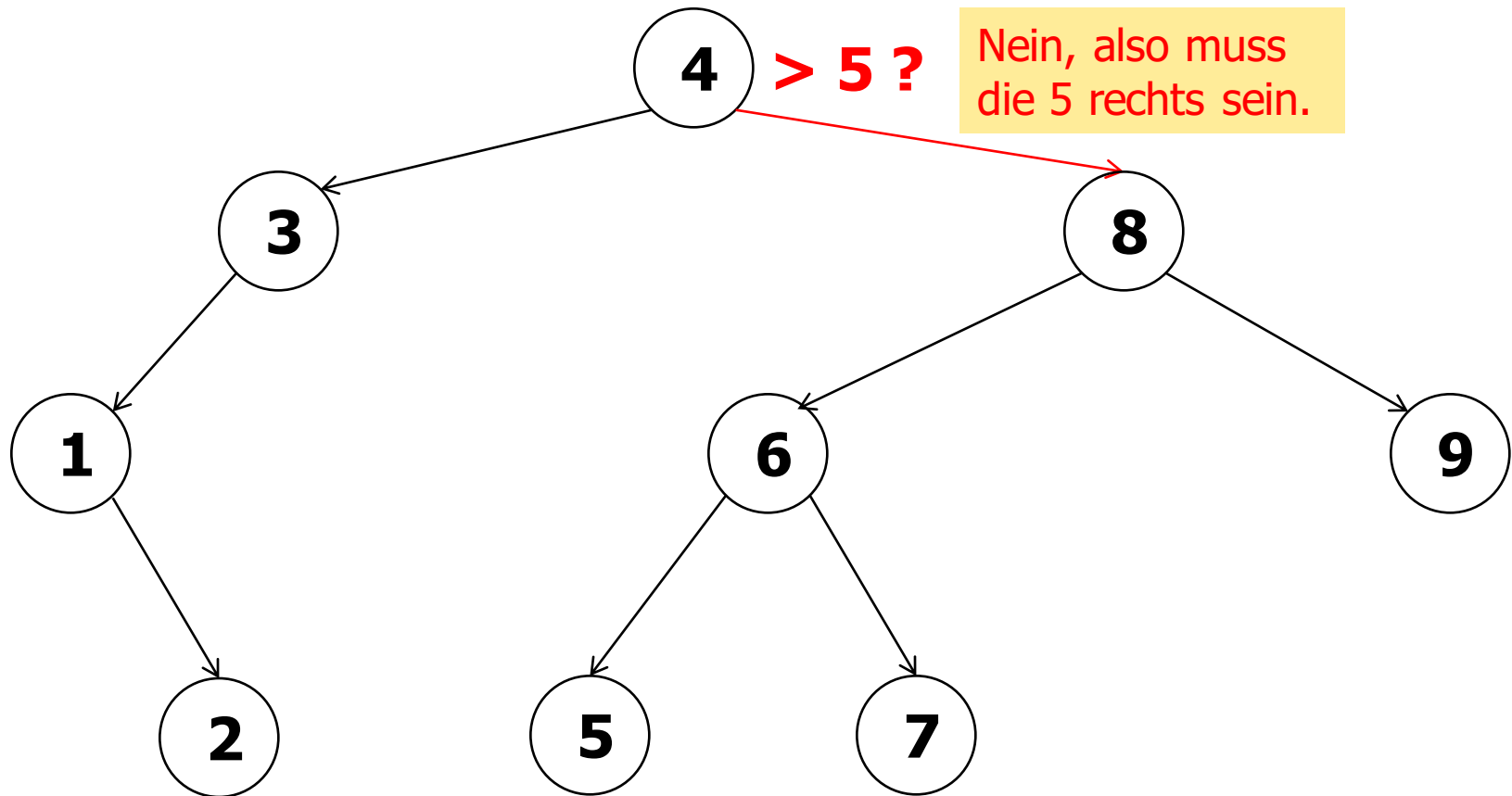


Binäre Suchbäume:

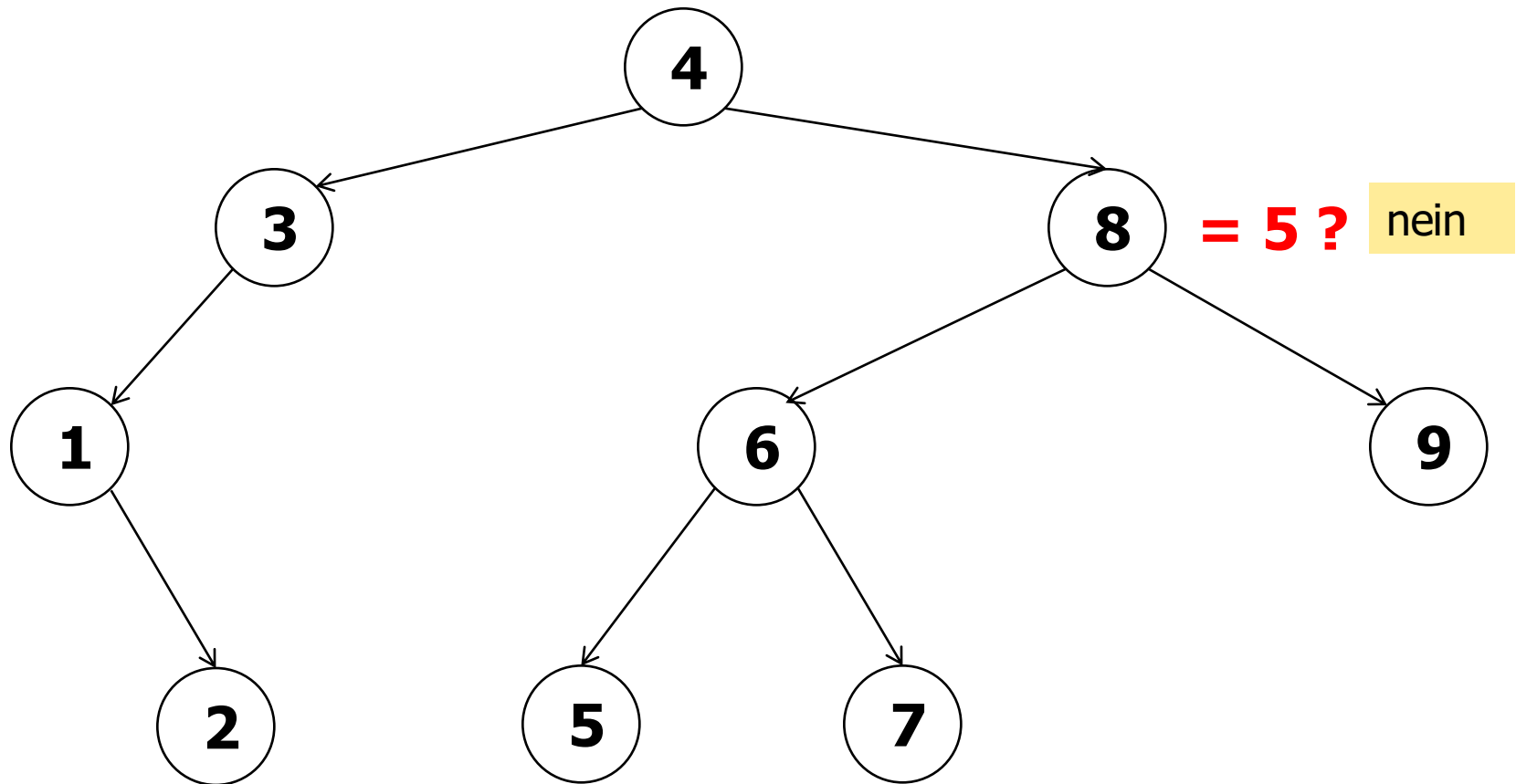
Suchen nach einem Schlüssel



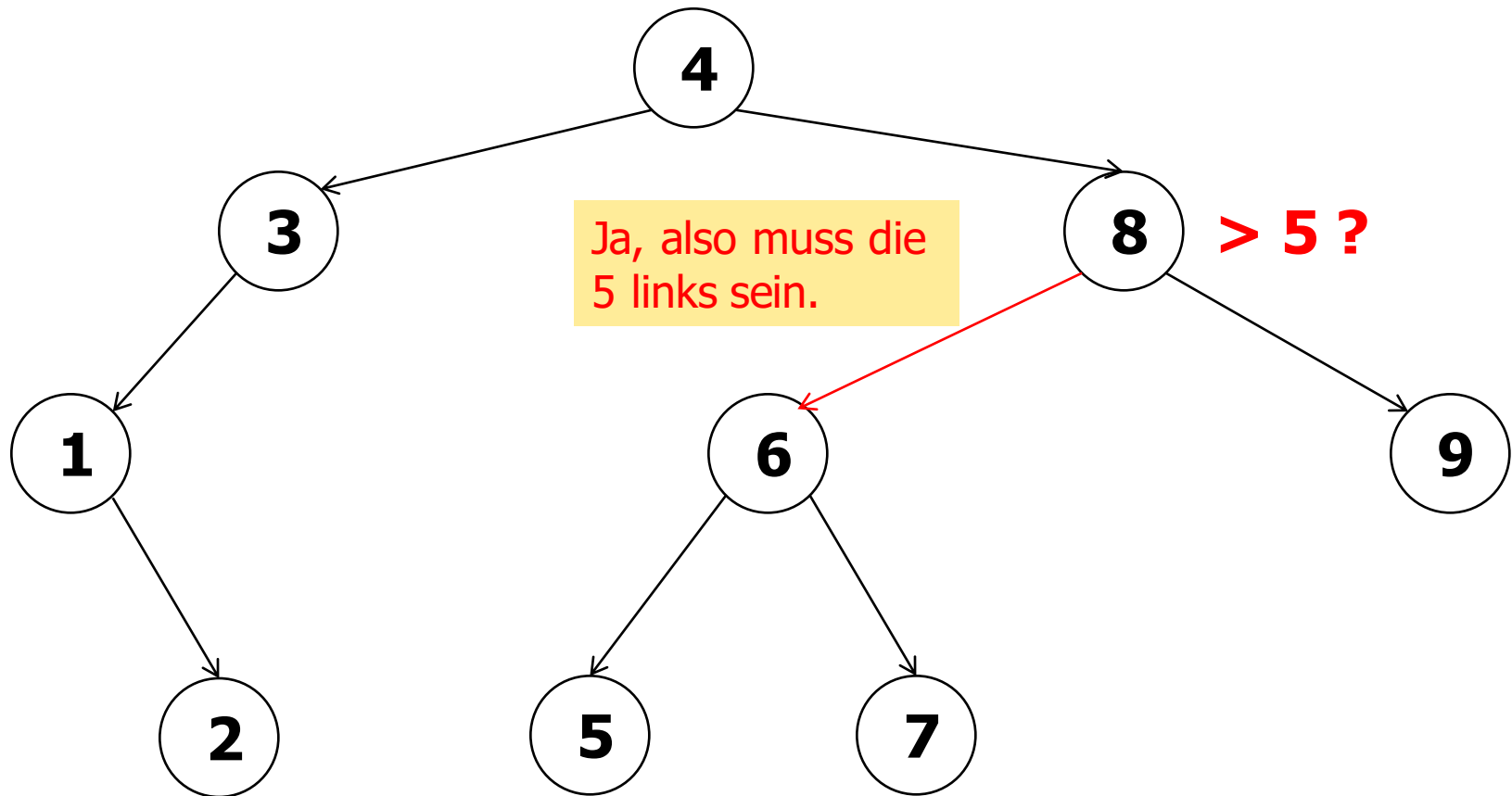
Binäre Suchbäume: Suchen nach einem Schlüssel



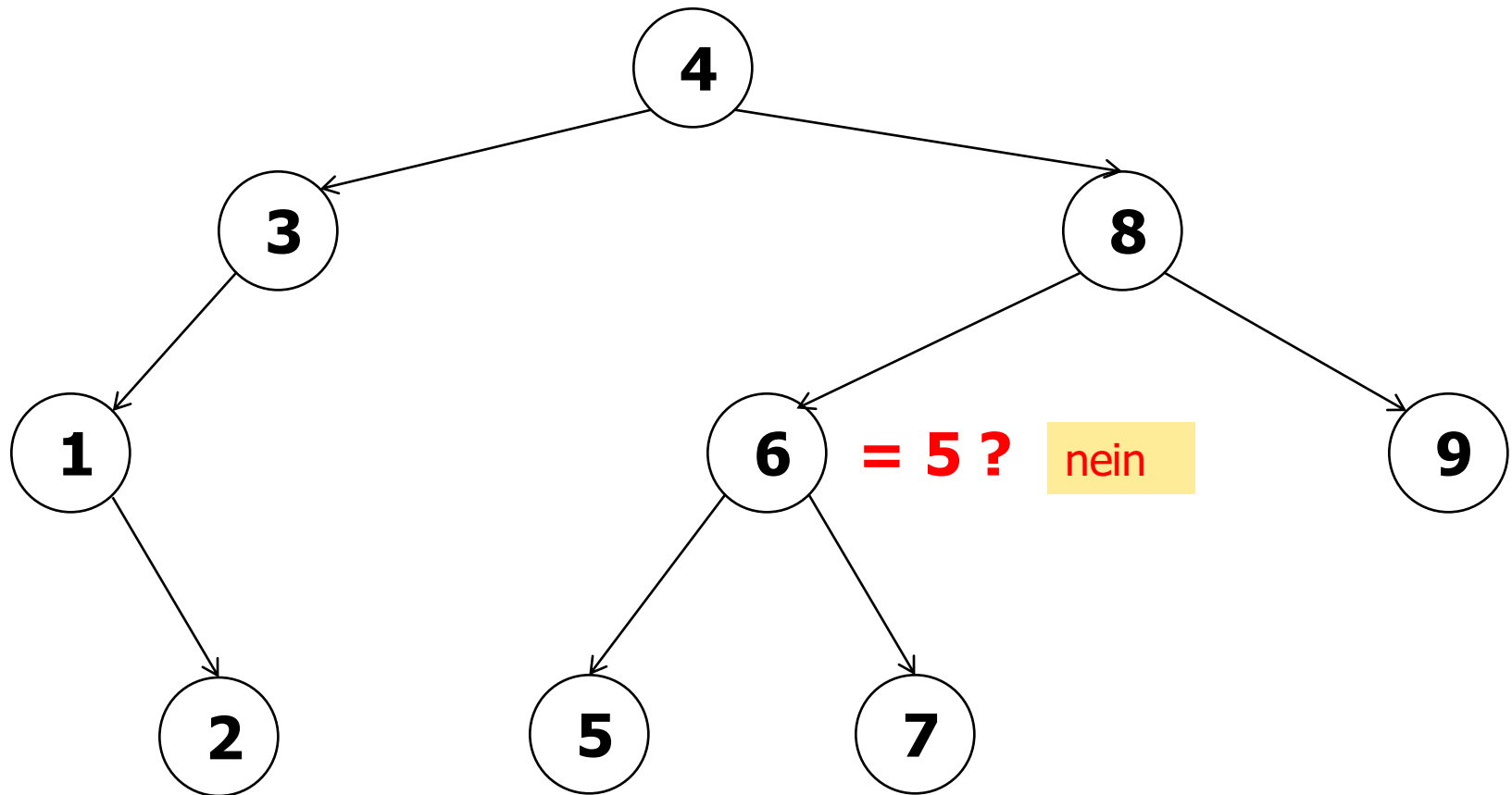
Binäre Suchbäume: Suchen nach einem Schlüssel



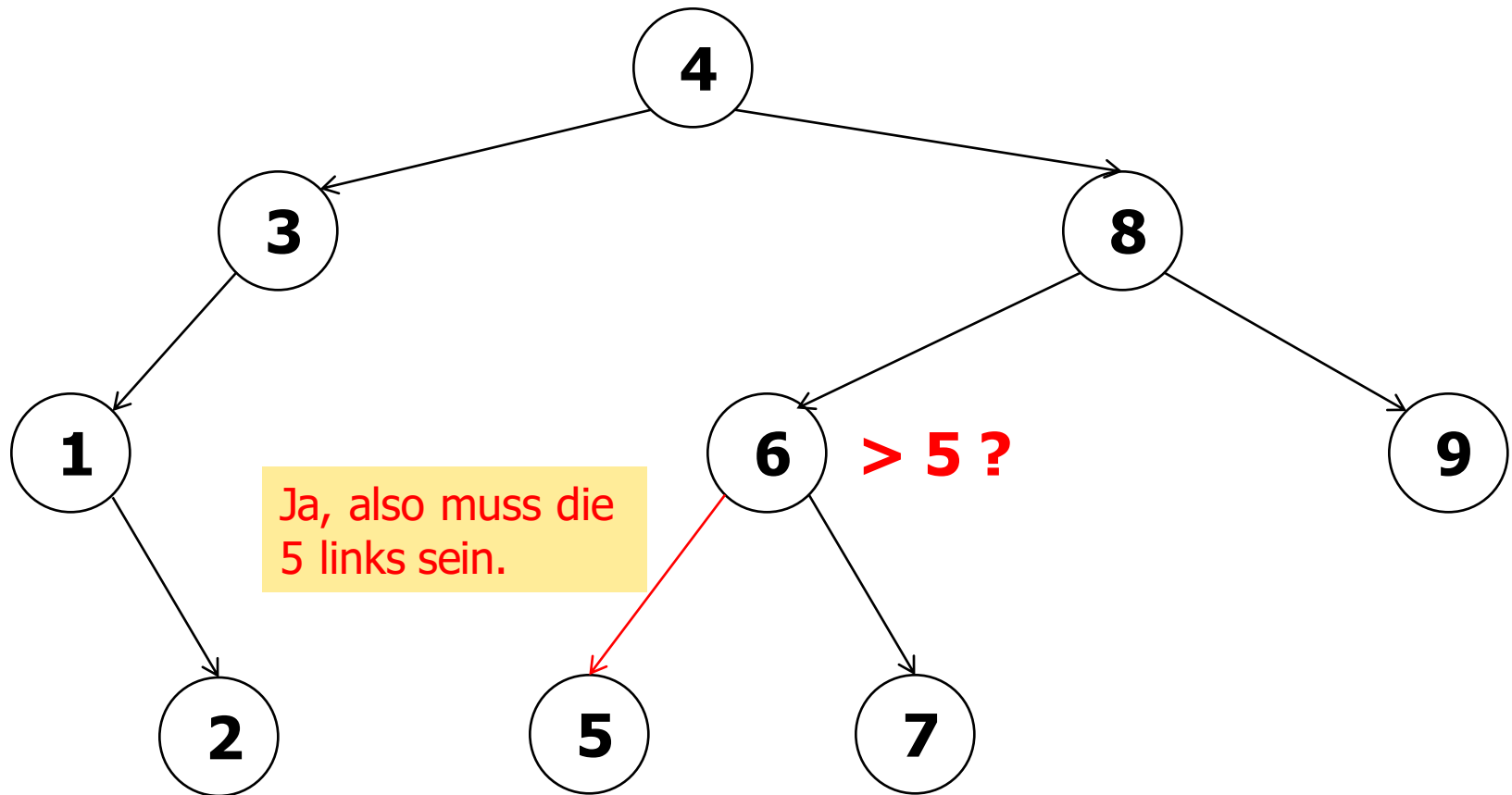
Binäre Suchbäume: Suchen nach einem Schlüssel



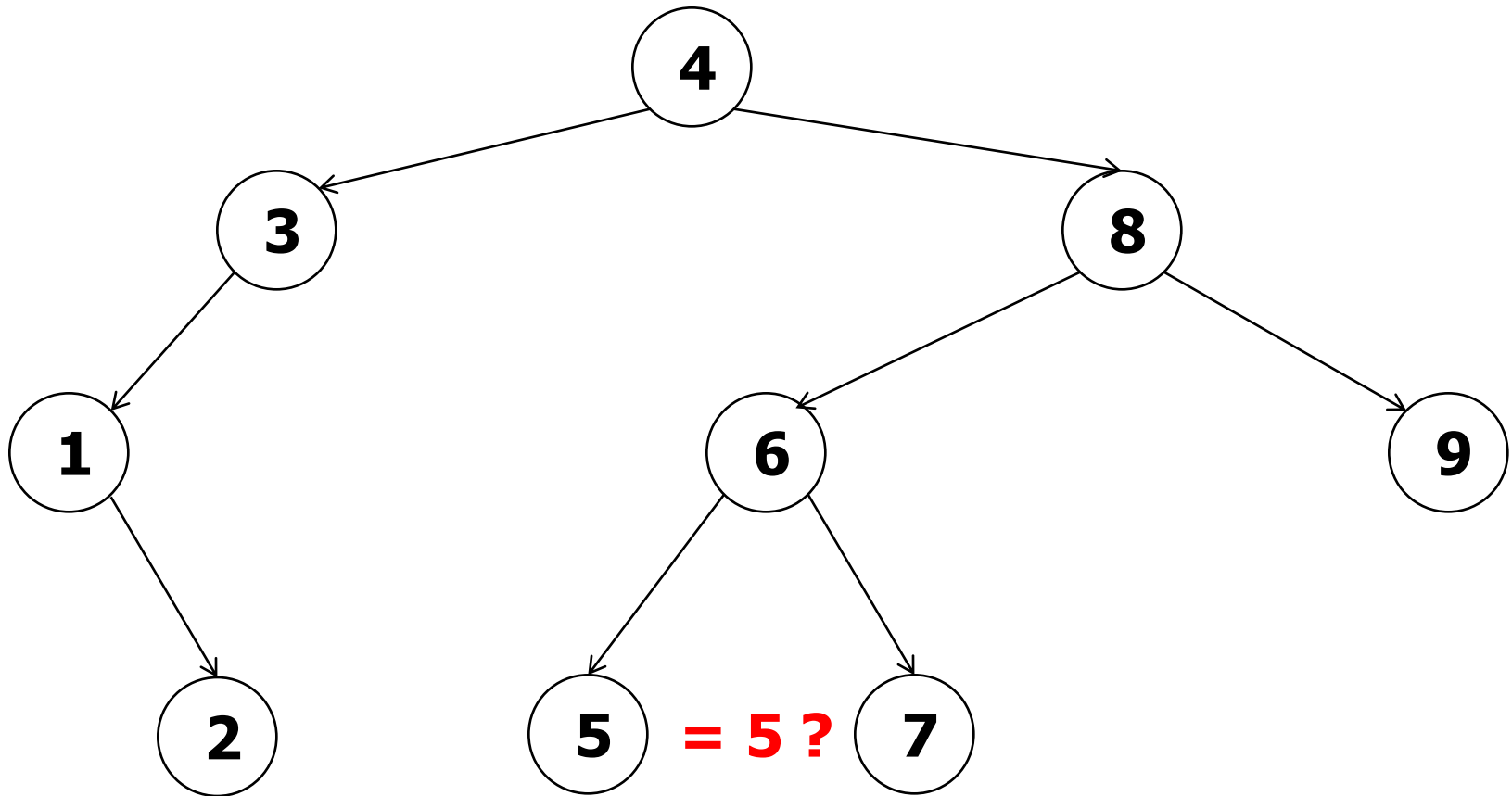
Binäre Suchbäume: Suchen nach einem Schlüssel



Binäre Suchbäume: Suchen nach einem Schlüssel

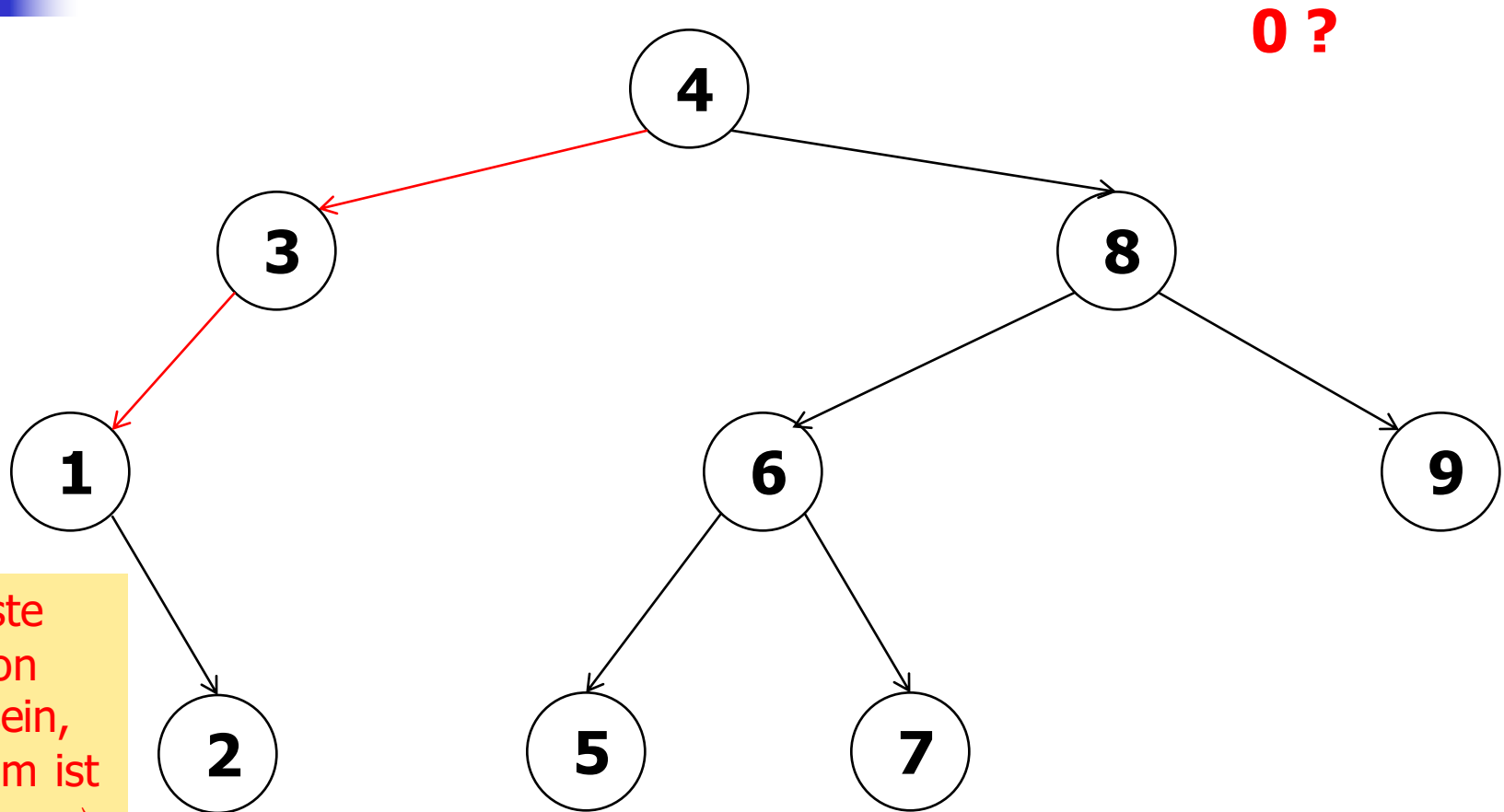


Binäre Suchbäume: Suchen nach einem Schlüssel



Ja, Schlüssel gefunden!

Binäre Suchbäume: Suchen nach einem Schlüssel





Mengen: Lösung mit binären Suchbäumen

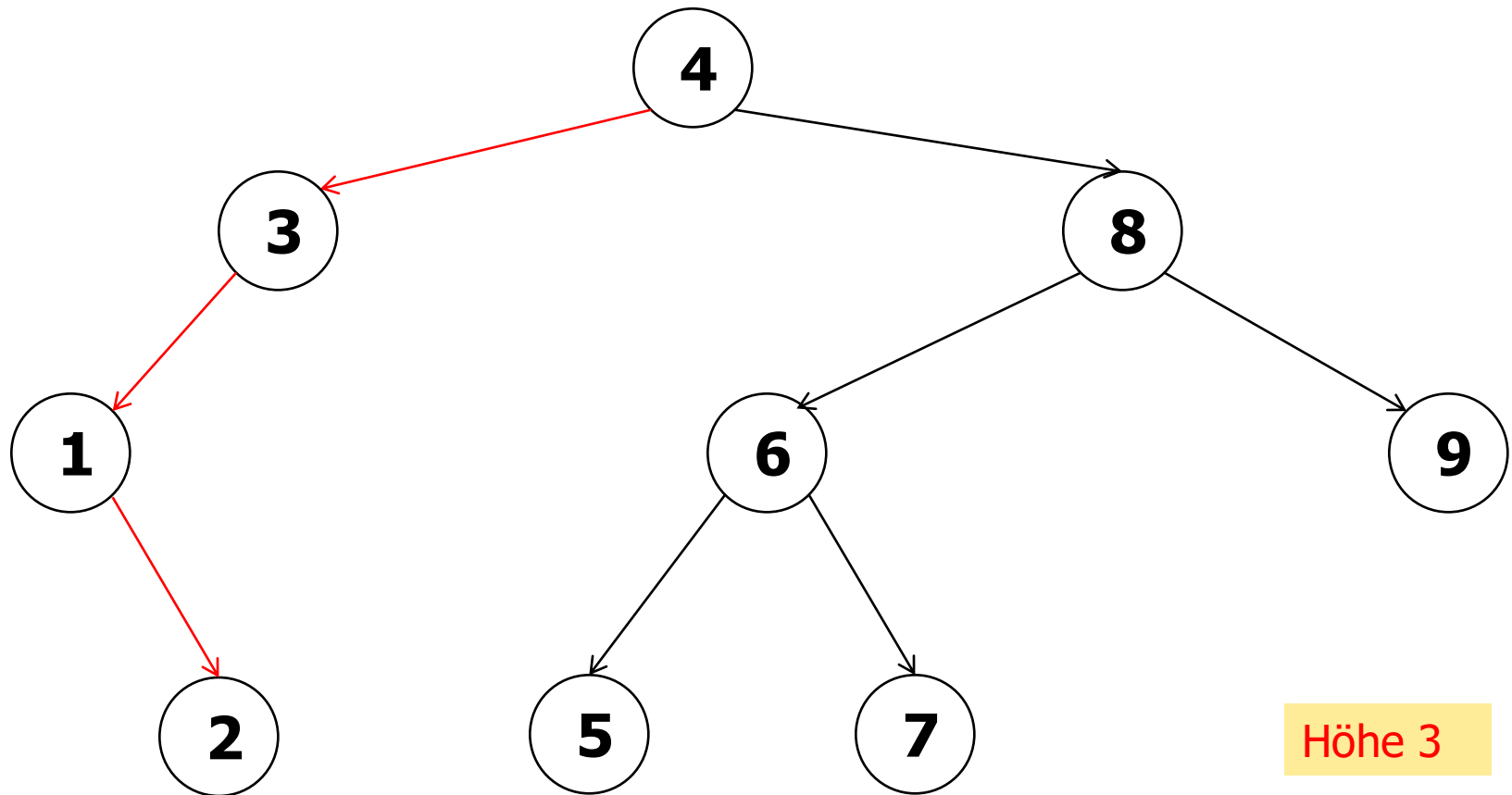
- Speichere die Elemente der Menge als Schlüssel in einem binären Suchbaum!



Mengen: Lösung mit binären Suchbäumen

- Speichere die Elemente der Menge als Schlüssel in einem binären Suchbaum!
- Suchzeit ist proportional zur **Höhe** des Baums (Länge des längsten Pfades von der Wurzel bis zu einem Blatt).

Mengen: Lösung mit binären Suchbäumen





Mengen: Lösung mit binären Suchbäumen

- Speichere die Elemente der Menge als Schlüssel in einem binären Suchbaum!
- Suchzeit ist proportional zur **Höhe** des Baums (Länge des längsten Pfades von der Wurzel bis zu einem Blatt).
- Einfügen und Löschen von Elementen?

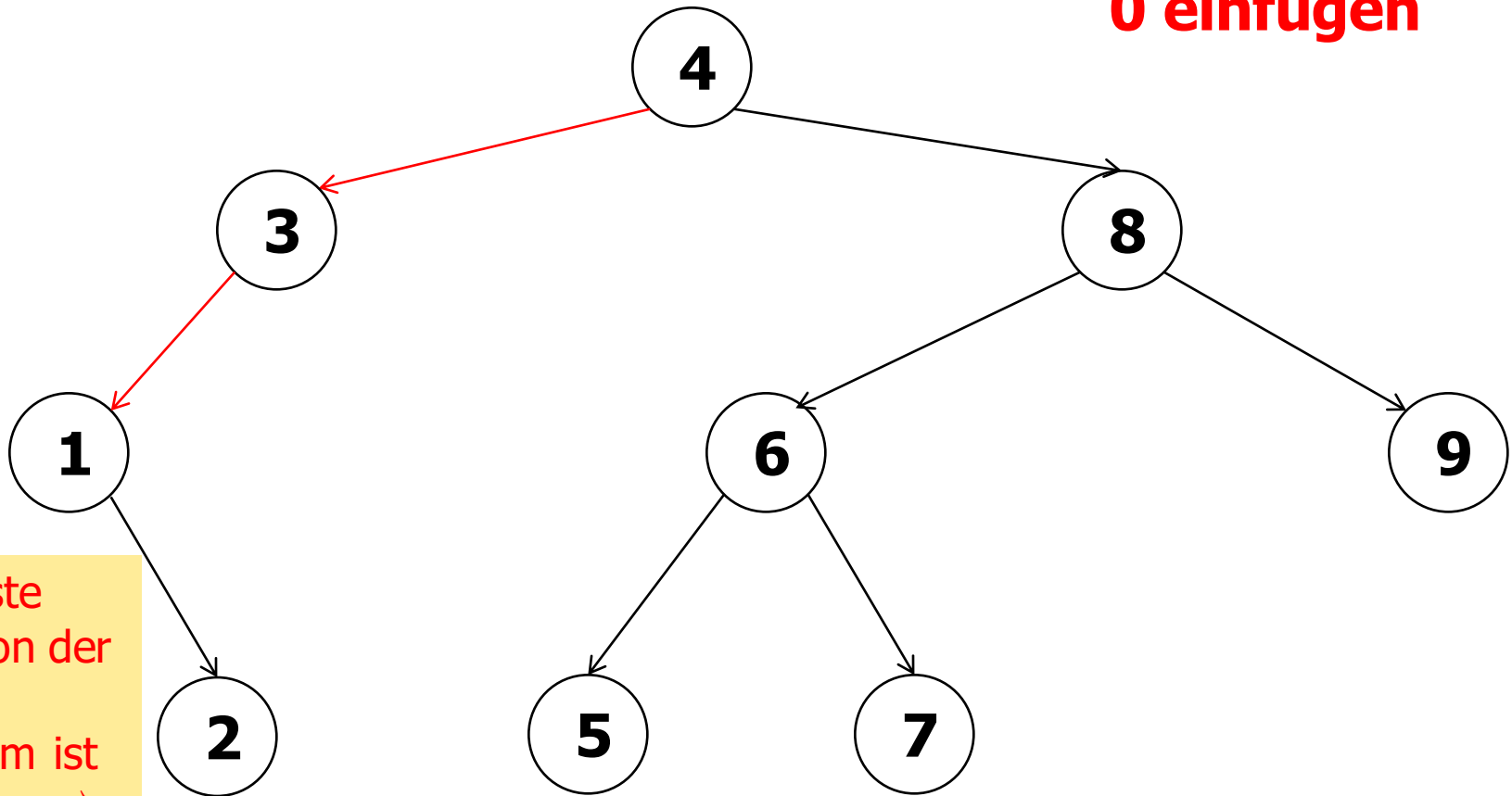
Binäre Suchbäume: Einfügen



- Neues Element suchen und als Blatt an der passenden Stelle anhängen!

Binäre Suchbäume: Einfügen

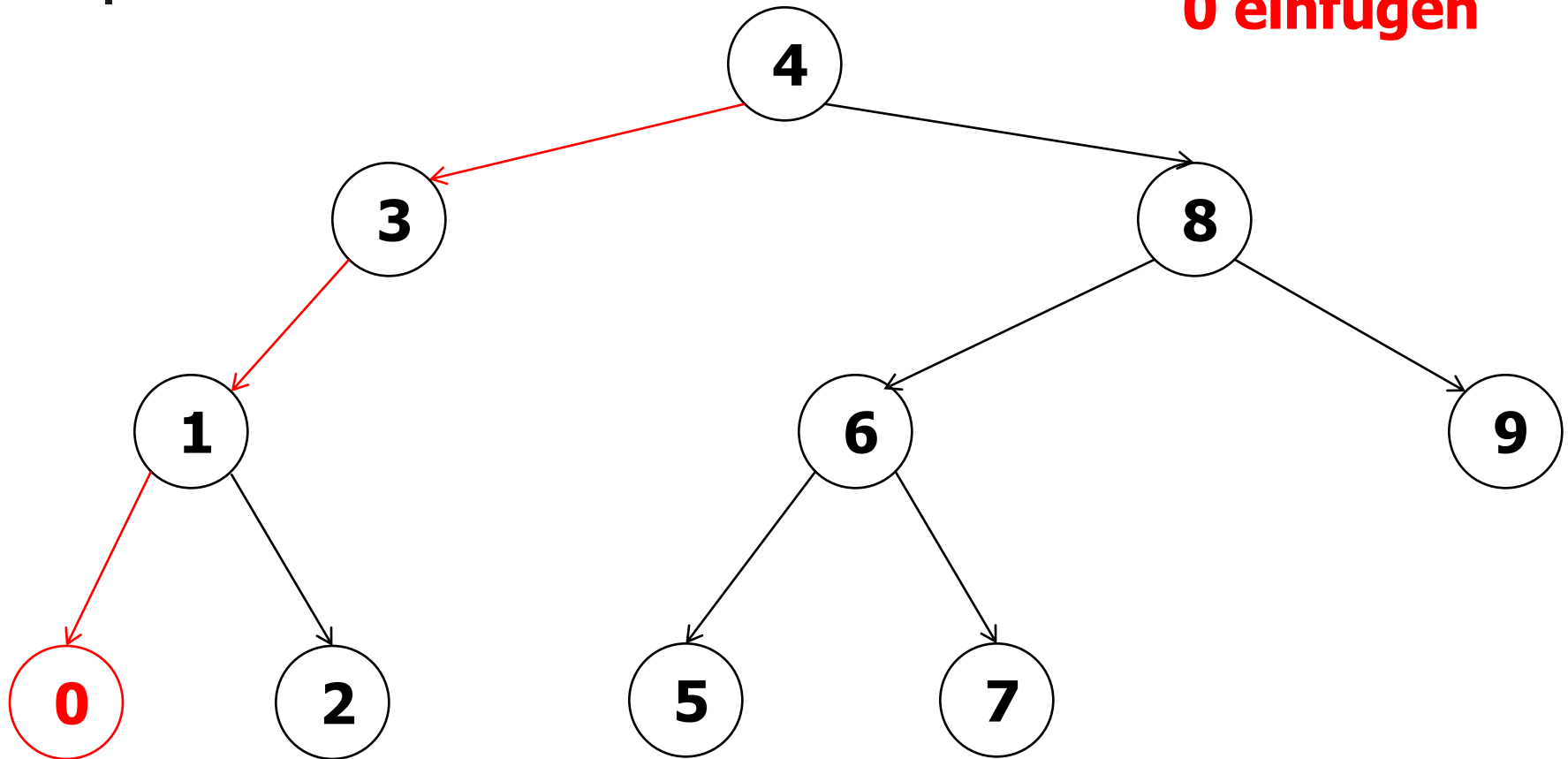
0 einfügen



0 müsste
links von der
1 sein,
Teilbaum ist
aber leer ⇒
neues Blatt

Binäre Suchbäume: Einfügen

0 einfügen



Binäre Suchbäume: Einfügen



- Neues Element suchen und als Blatt an der passenden Stelle anhängen!
- Auch die Einfügezeit ist proportional zur Höhe des Baums.

Binäre Suchbäume:

Löschen

- Element suchen, seinen Knoten durch **Rotationen** zu einem Blatt machen, Blatt löschen!

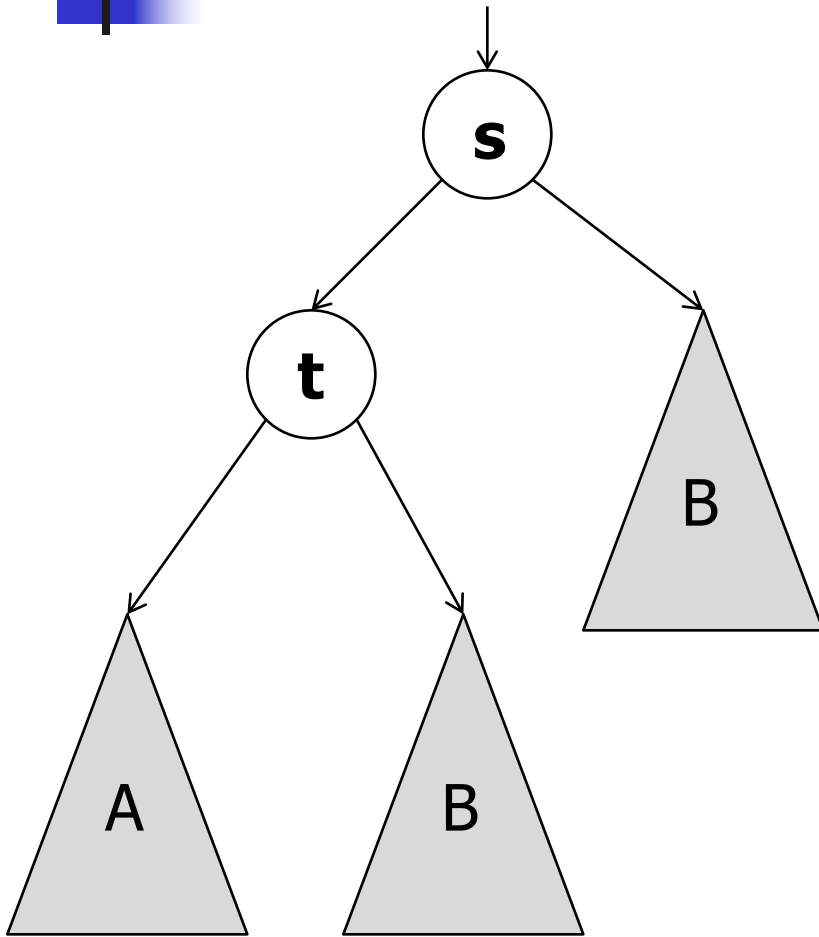
Binäre Suchbäume:

Löschen

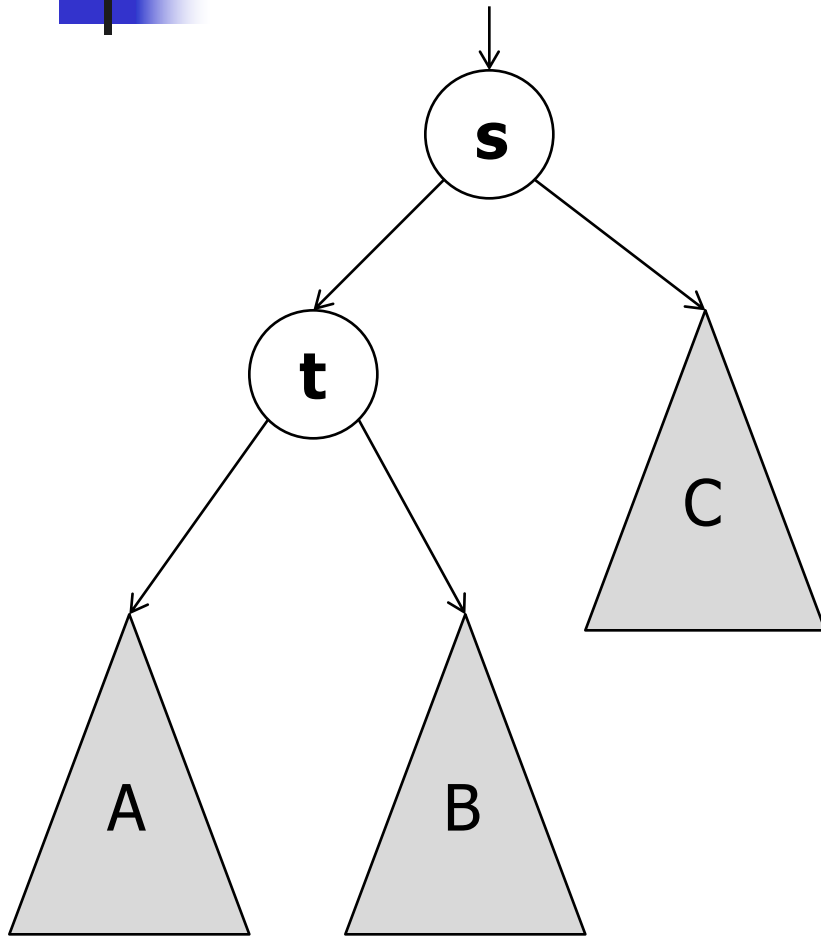
- Element suchen, seinen Knoten durch **Rotationen** zu einem Blatt machen, Blatt löschen!
- Rotation: lokale Operation, welche die Positionen von Knoten verändert, **nicht** jedoch die Suchbaumeigenschaften.



Rotation eines Teilbaums

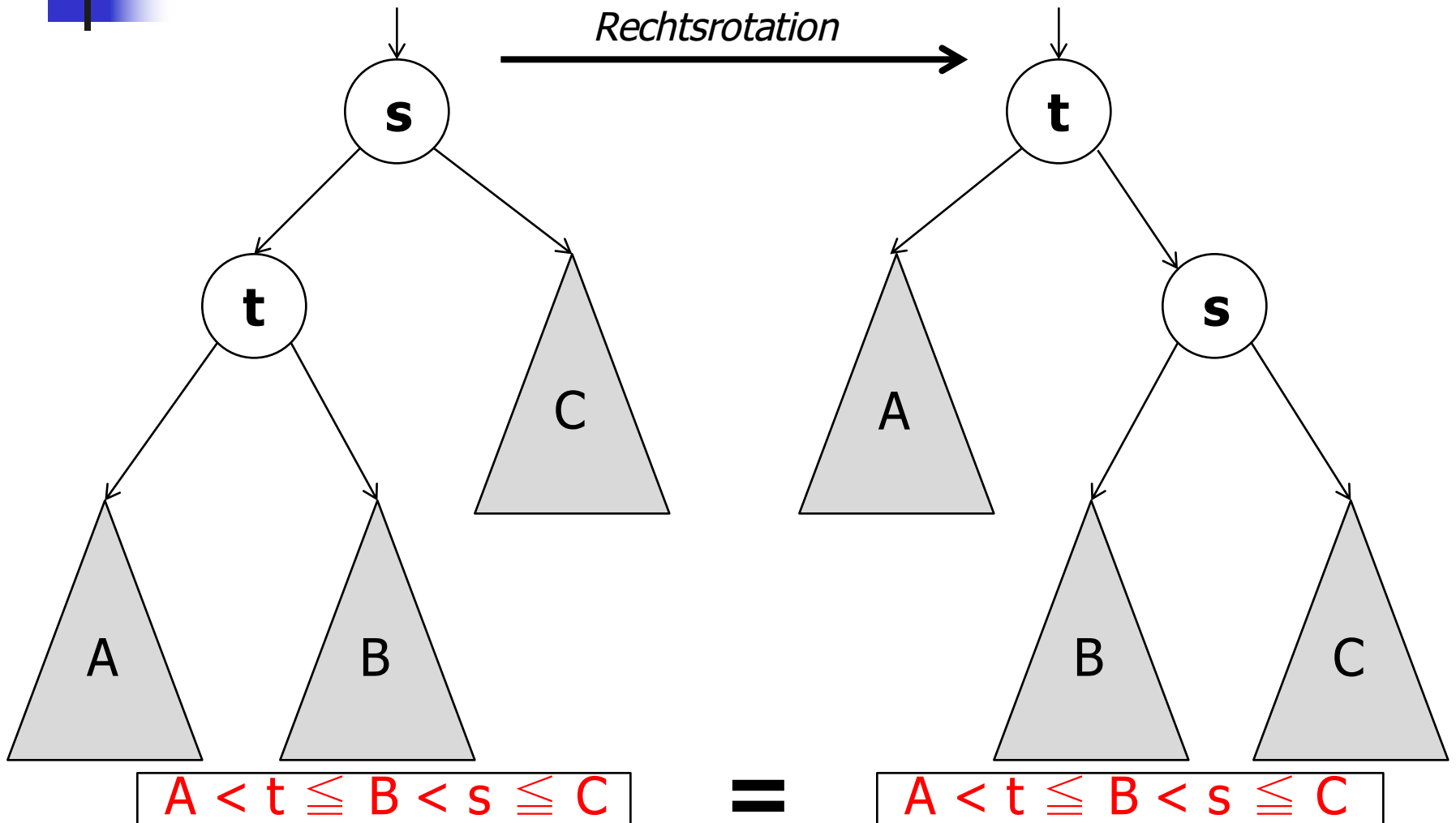


Rotation eines Teilbaums

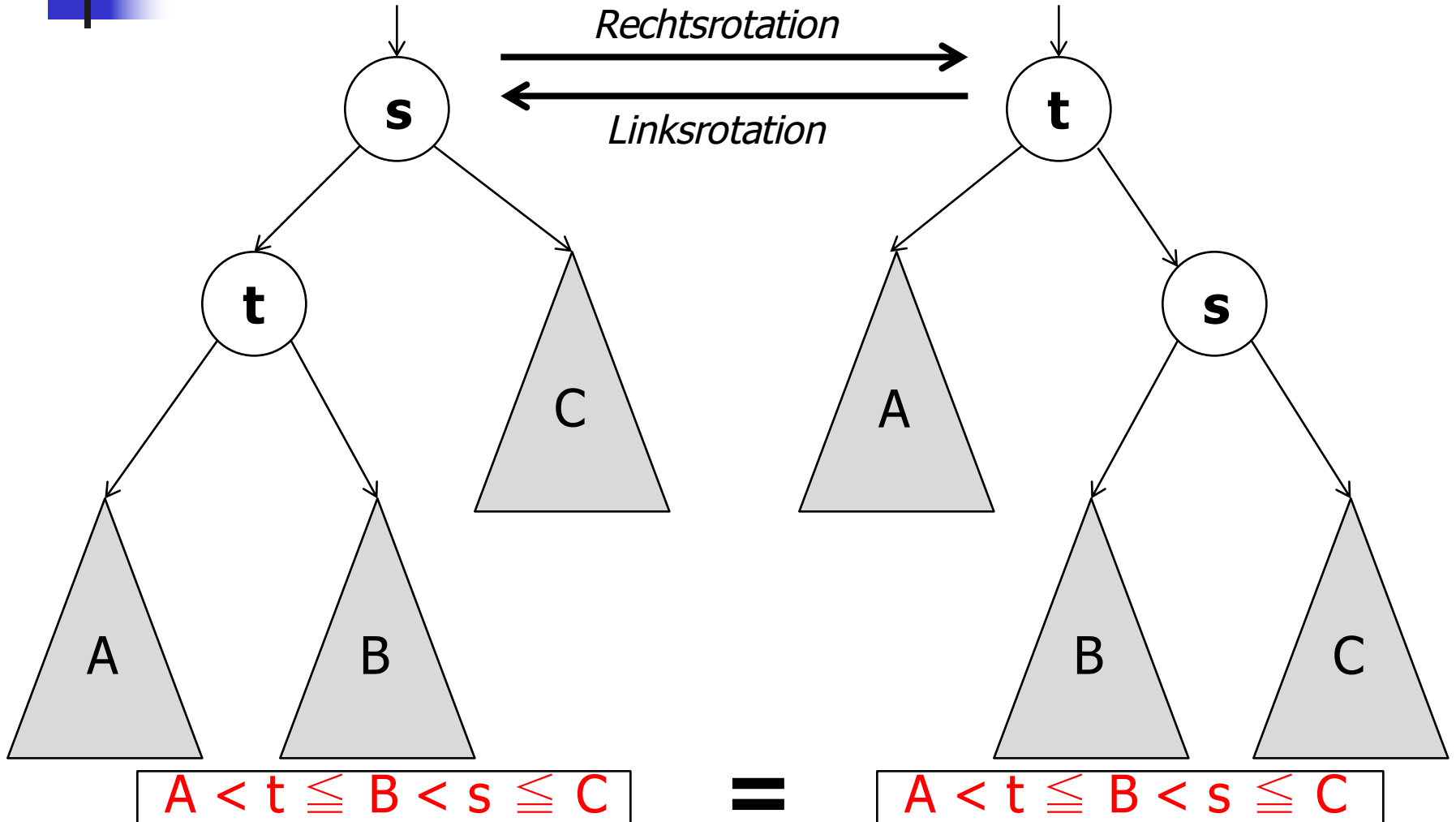


$$A < t \leq B < s \leq C$$

Rotation eines Teilbaums

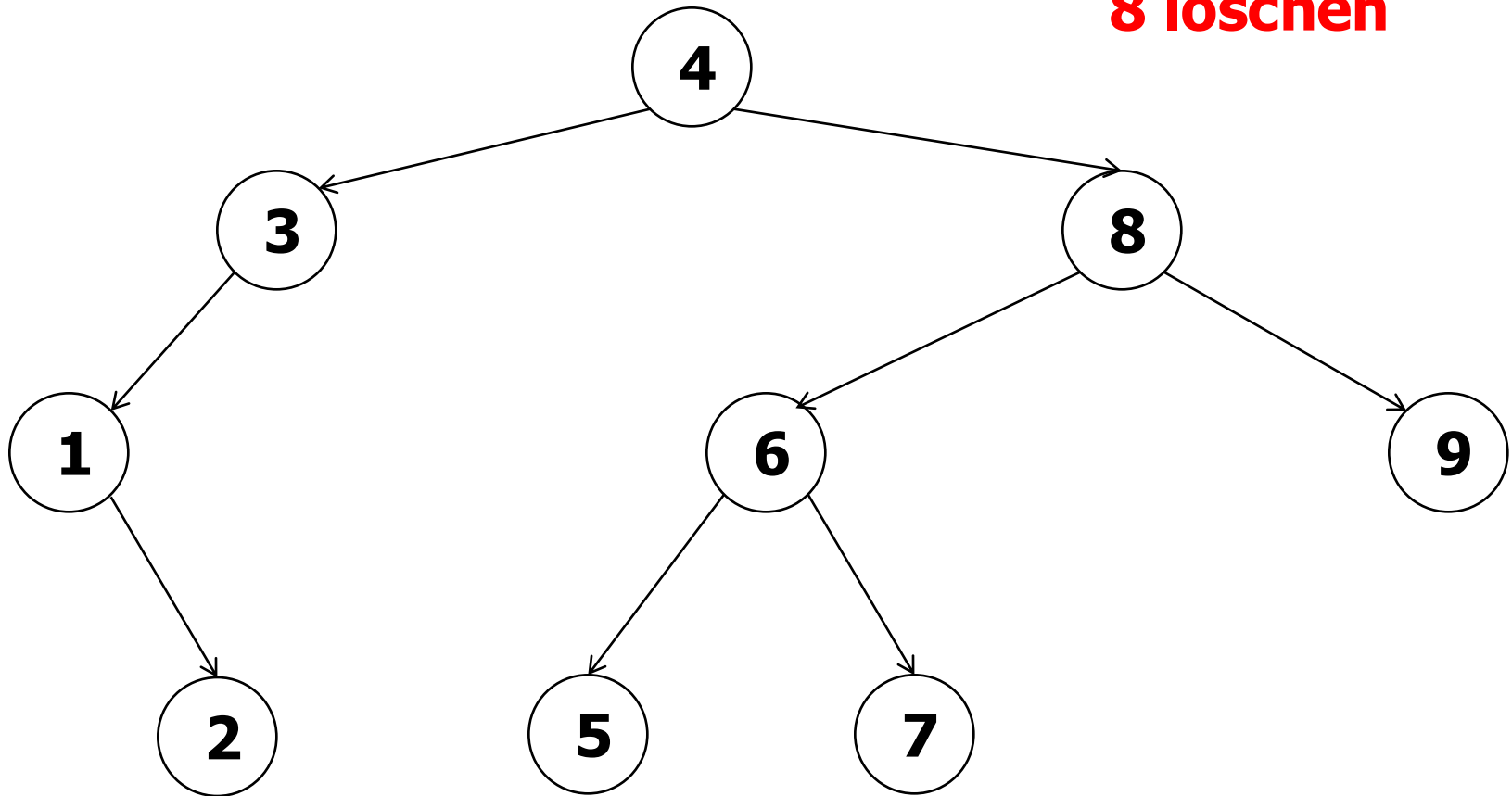


Rotation eines Teilbaums



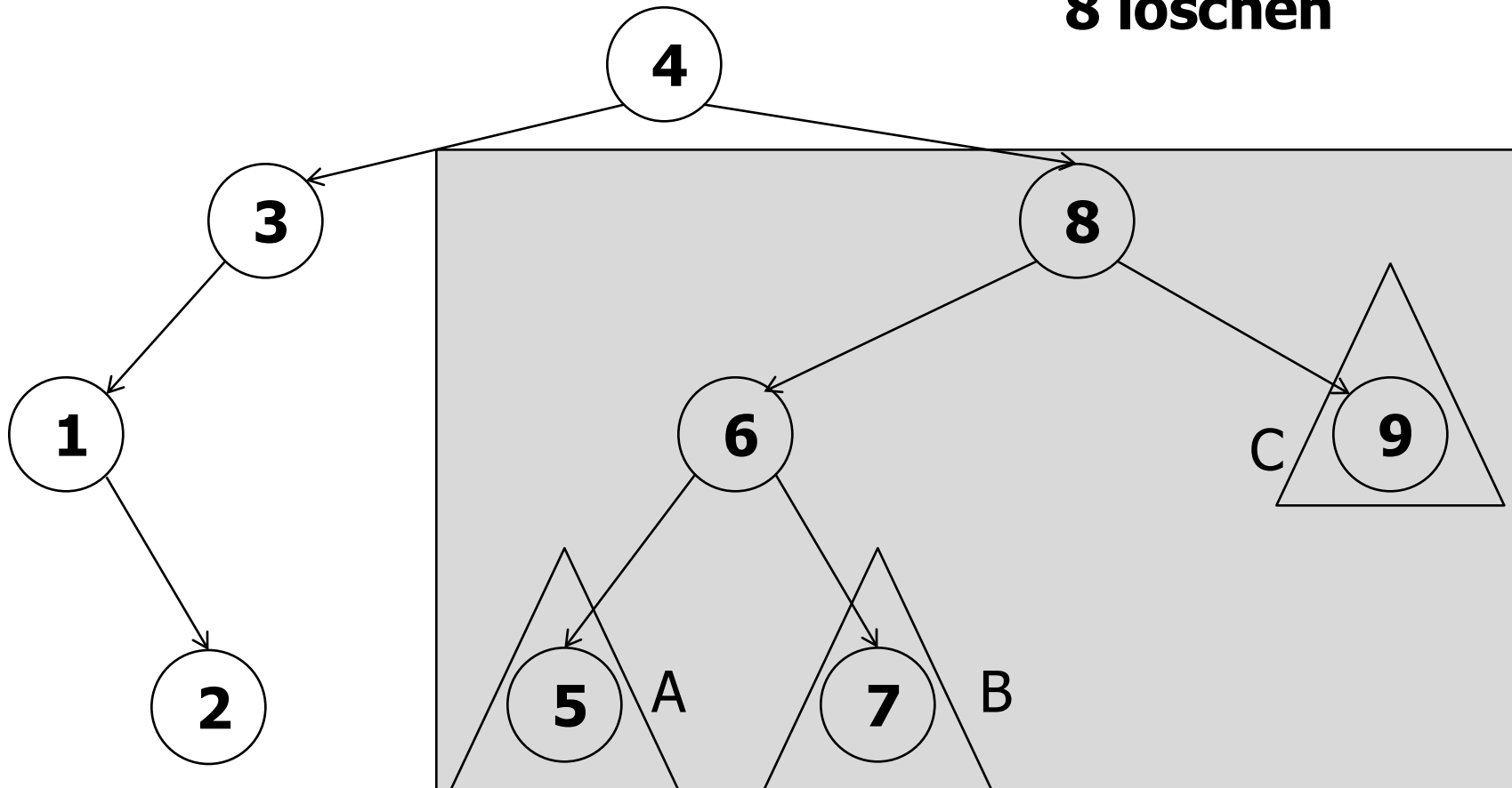
Binäre Suchbäume: Löschen

8 löschen



Binäre Suchbäume: Löschen

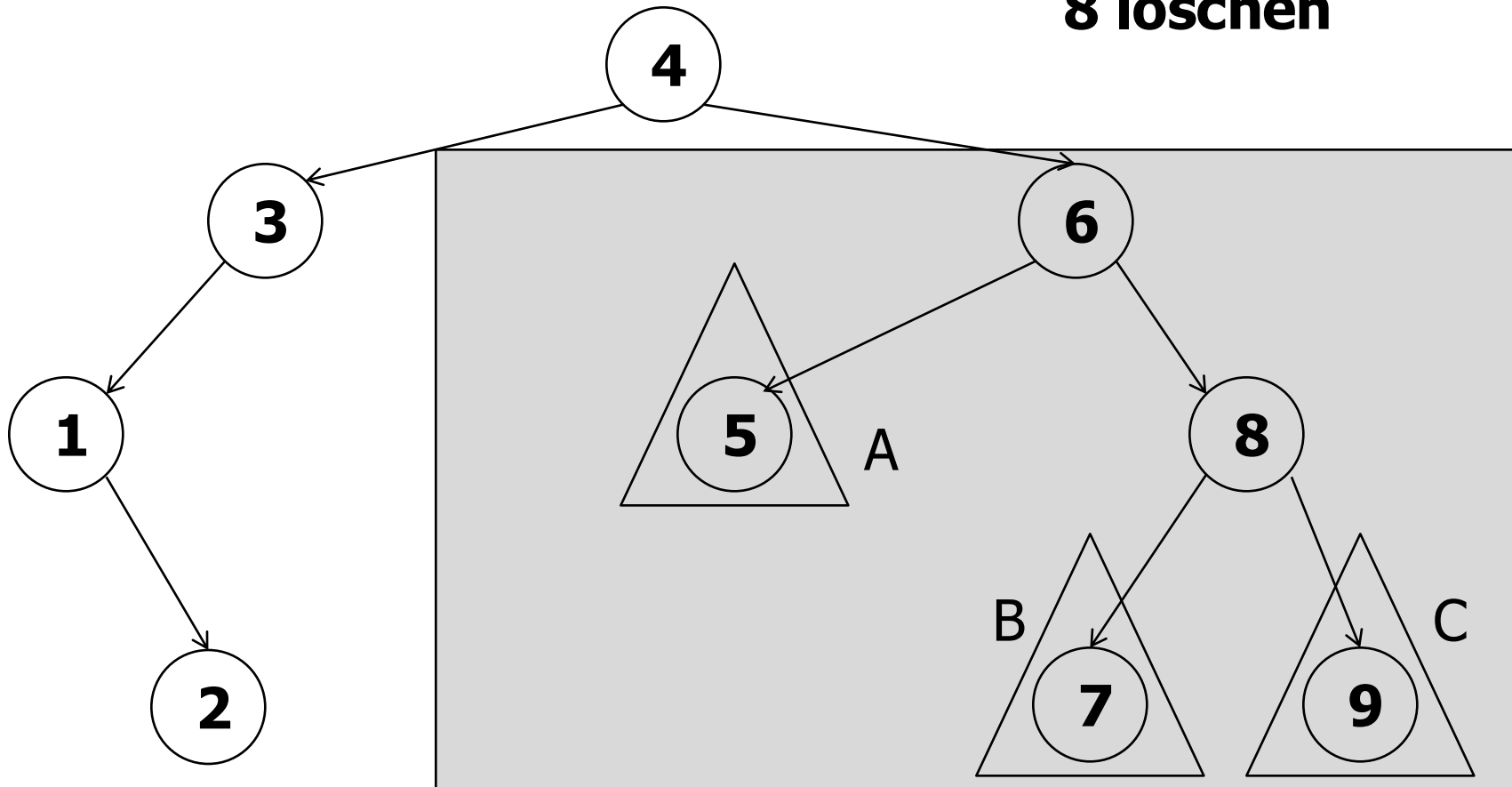
8 löschen



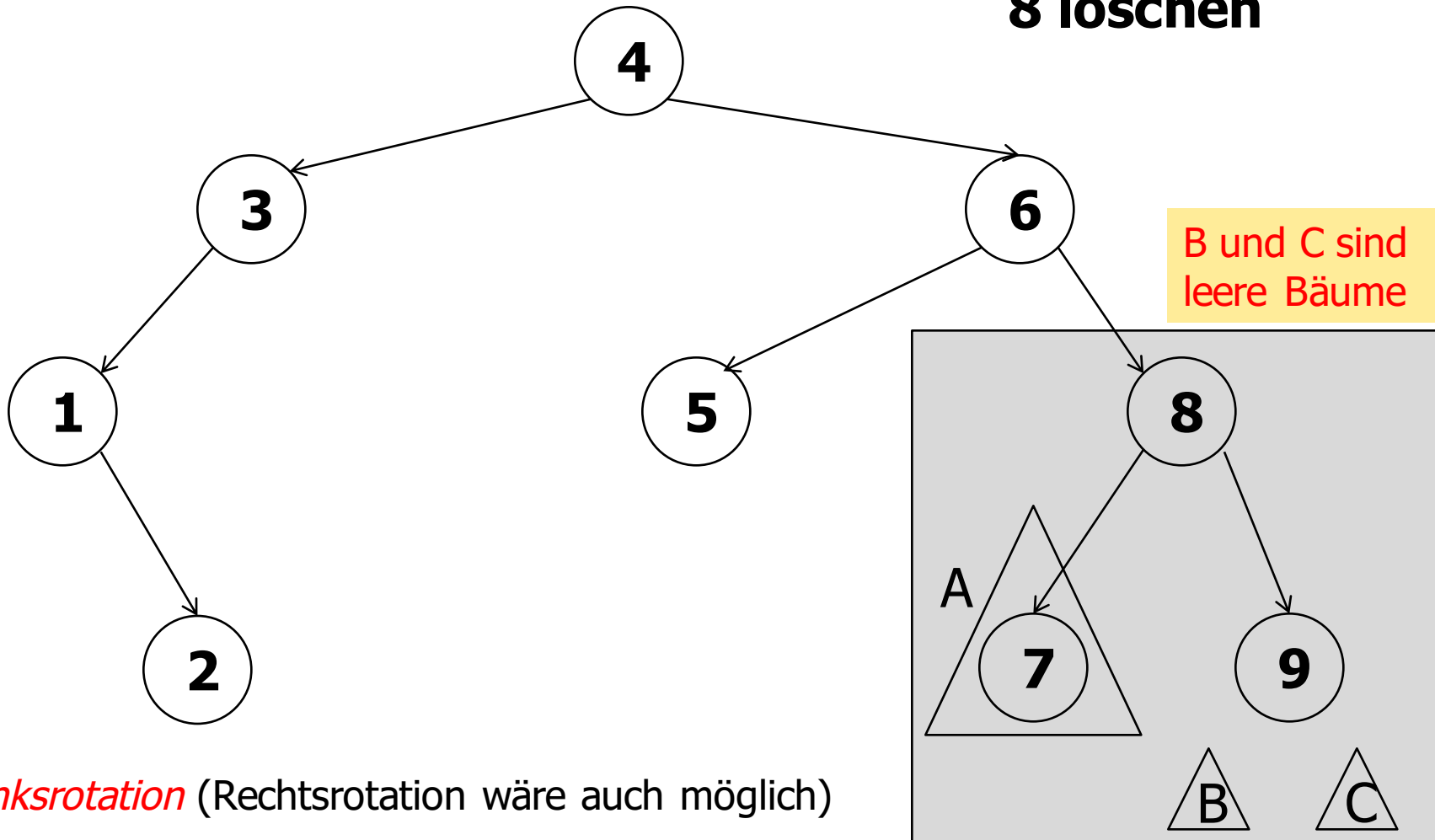
Rechtsrotation (Linksrotation wäre auch möglich)

Binäre Suchbäume: Löschen

8 löschen

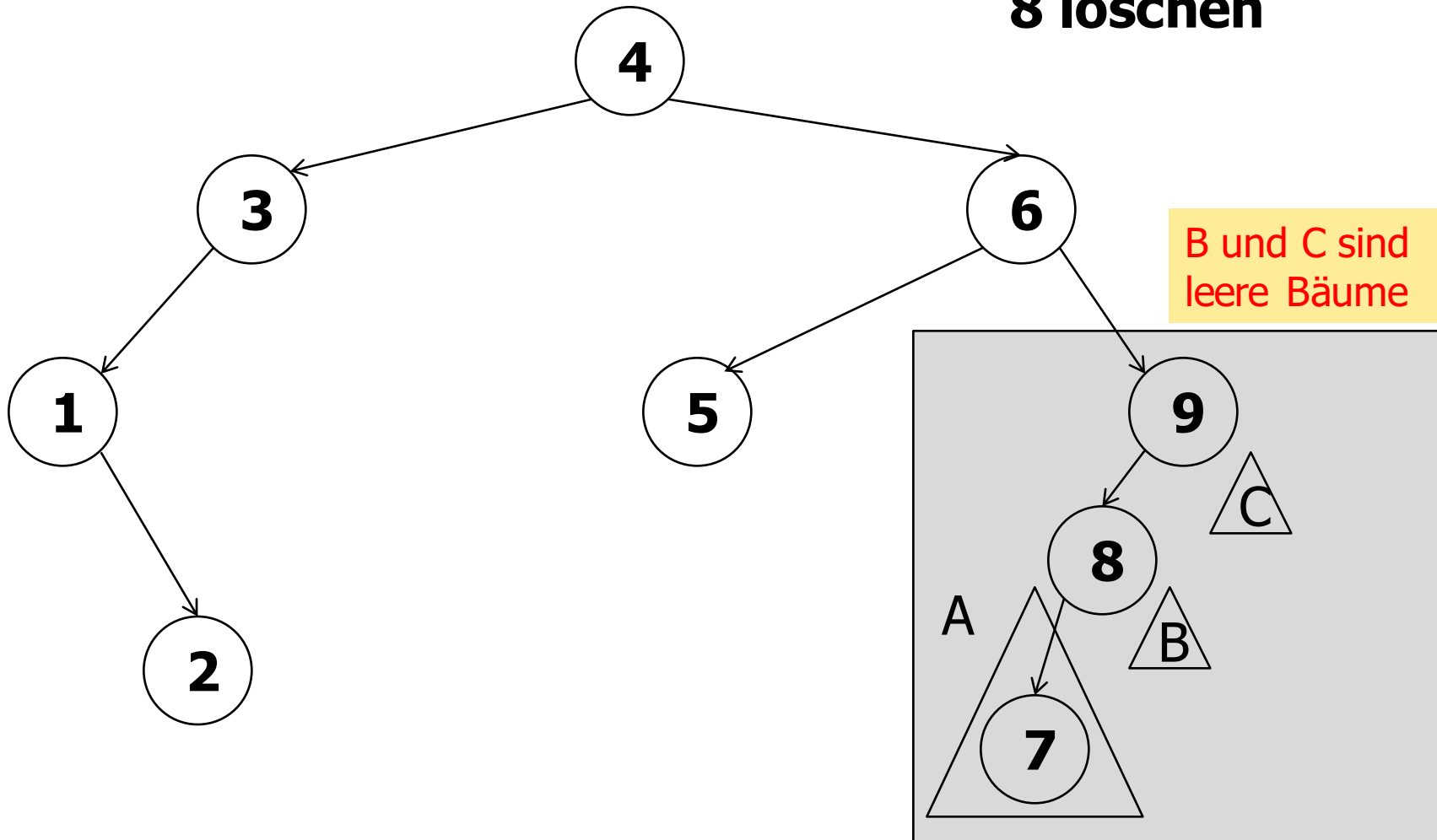


Binäre Suchbäume: Löschen



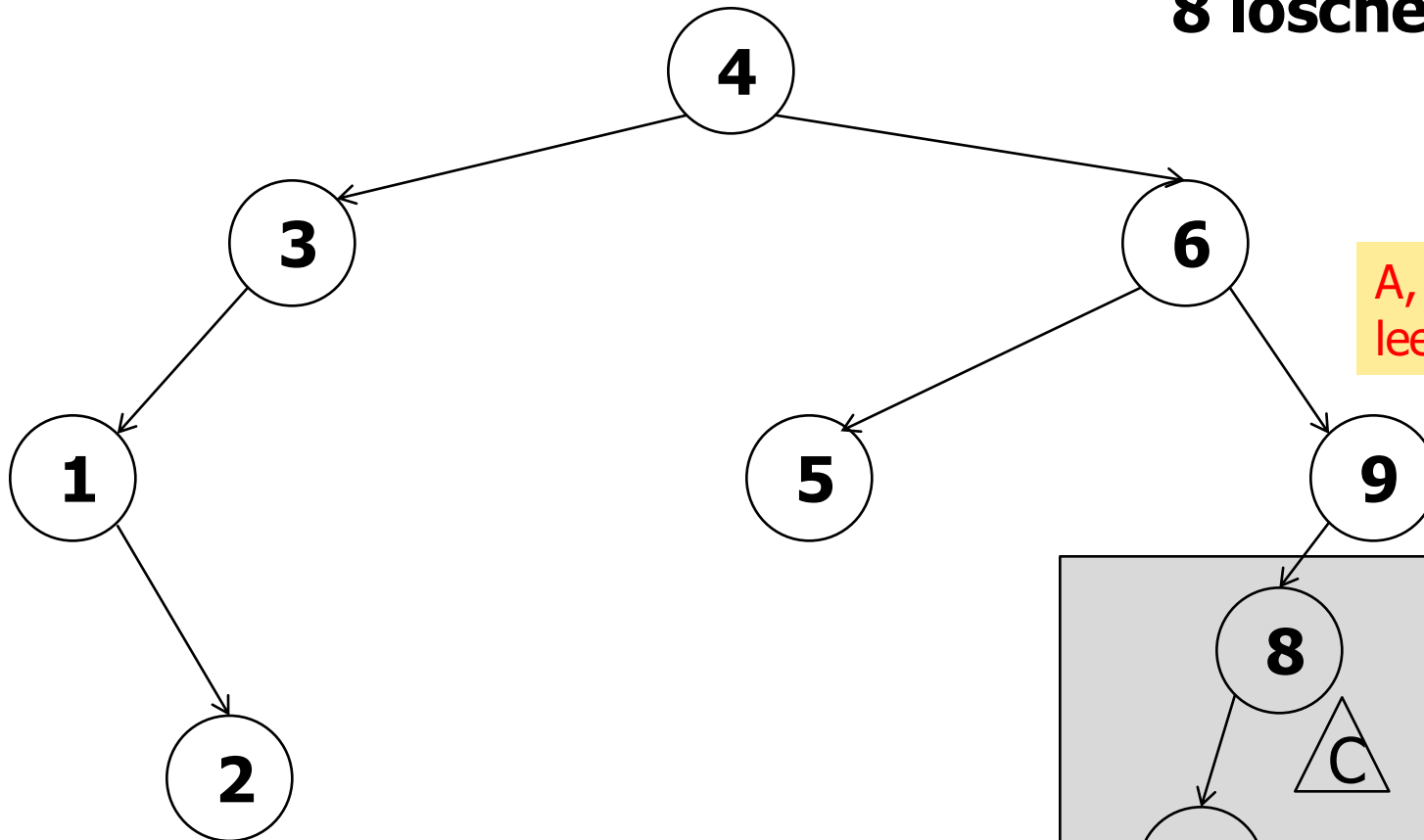
Binäre Suchbäume: Löschen

8 löschen



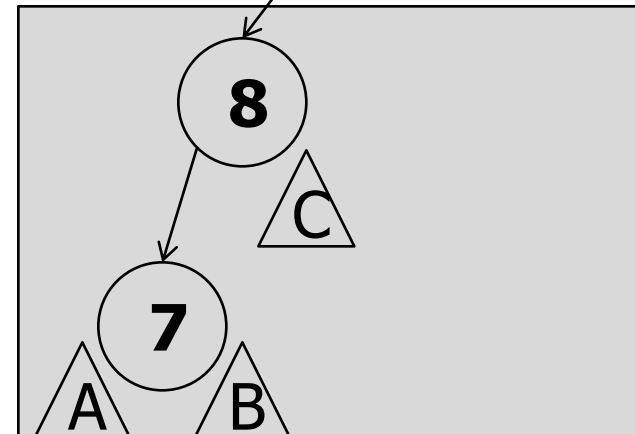
Binäre Suchbäume: Löschen

8 löschen



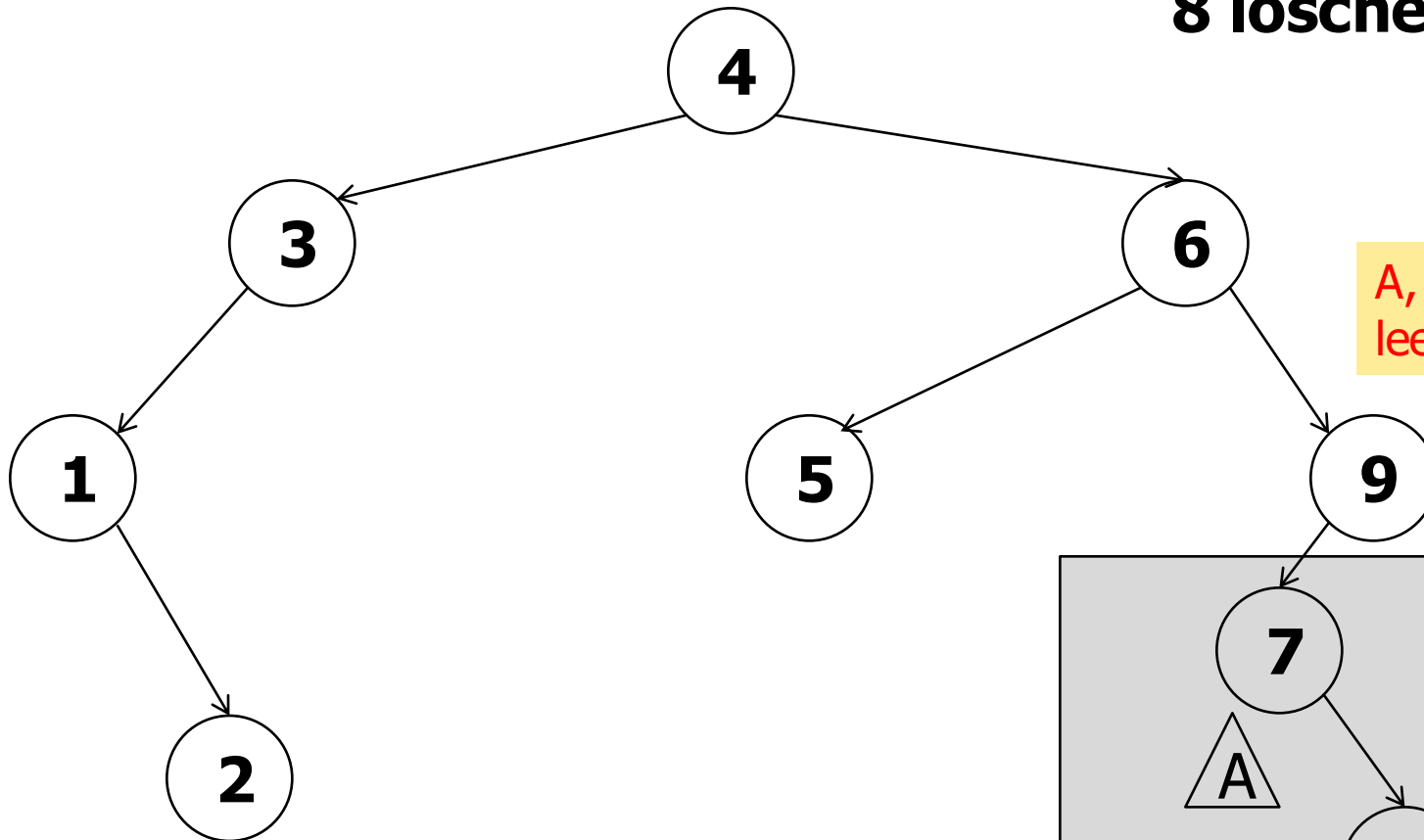
A, B, C sind
leere Bäume

Rechtsrotation (Linksrotation wäre nicht möglich)

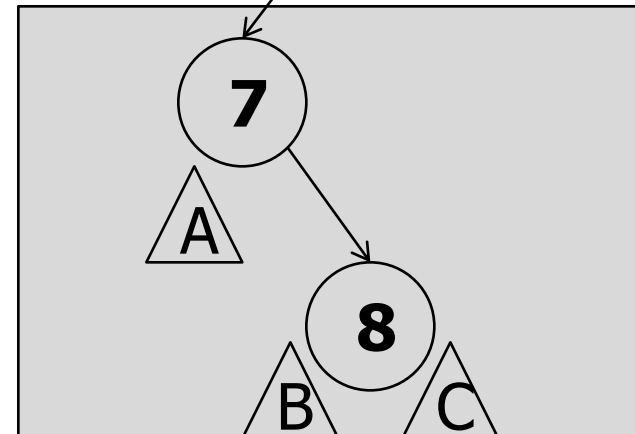


Binäre Suchbäume: Löschen

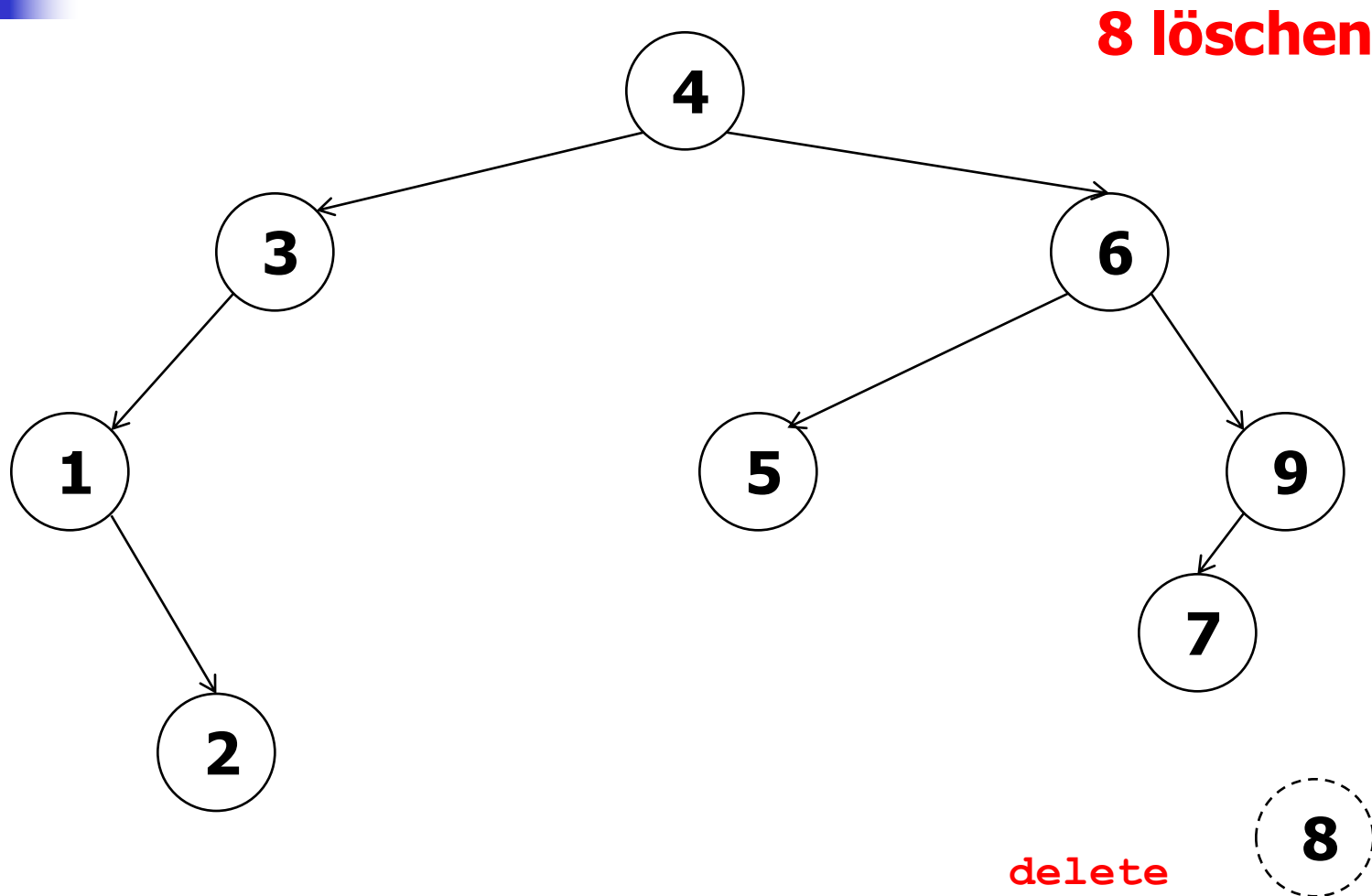
8 löschen



A, B, C sind
leere Bäume



Binäre Suchbäume: Löschen



Binäre Suchbäume:

Löschen

- Element suchen, seinen Knoten durch **Rotationen** zu einem Blatt machen, Blatt löschen!
- Rotation: lokale Operation, welche die Positionen von Knoten verändert, **nicht** jedoch die Suchbaumeigenschaften.
- Auch die Löschzeit ist proportional zur Höhe des Baums (nicht mehr ganz so offensichtlich)



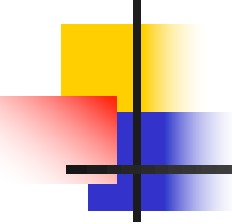
Die Höhe eines binären Suchbaums...

- bestimmt die Laufzeit des
 - Suchens,
 - Einfügens,
 - Löschens.



Die Höhe eines binären Suchbaums...

- bestimmt die Laufzeit des
 - Suchens,
 - Einfügens,
 - Löschens.
- kann aber bereits bei einer ungünstigen Einfügereihenfolge sehr gross werden!



Die Höhe eines binären Suchbaums...

- bestimmt die Laufzeit des
 - Suchens,
 - Einfügens,
 - Löschens.
- kann aber bereits bei einer ungünstigen Einfügereihenfolge sehr gross werden!
- Laufzeit im schlimmsten Fall nicht besser als bei einer Liste!

Die Höhe eines binären Suchbaums...

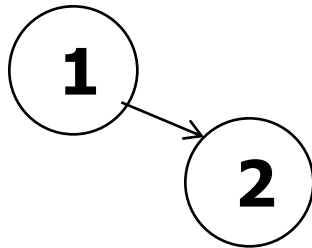
**Beispiel: Einfügen in
sortierter Reihenfolge**



1

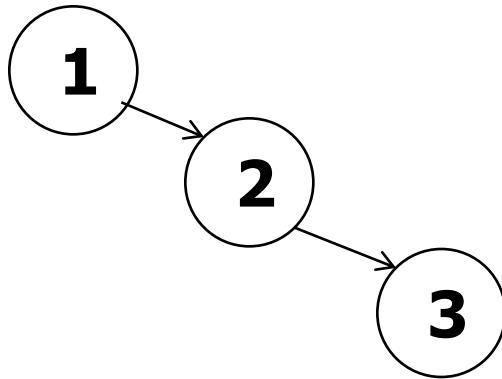
Die Höhe eines binären Suchbaums...

**Beispiel: Einfügen in
sortierter Reihenfolge**



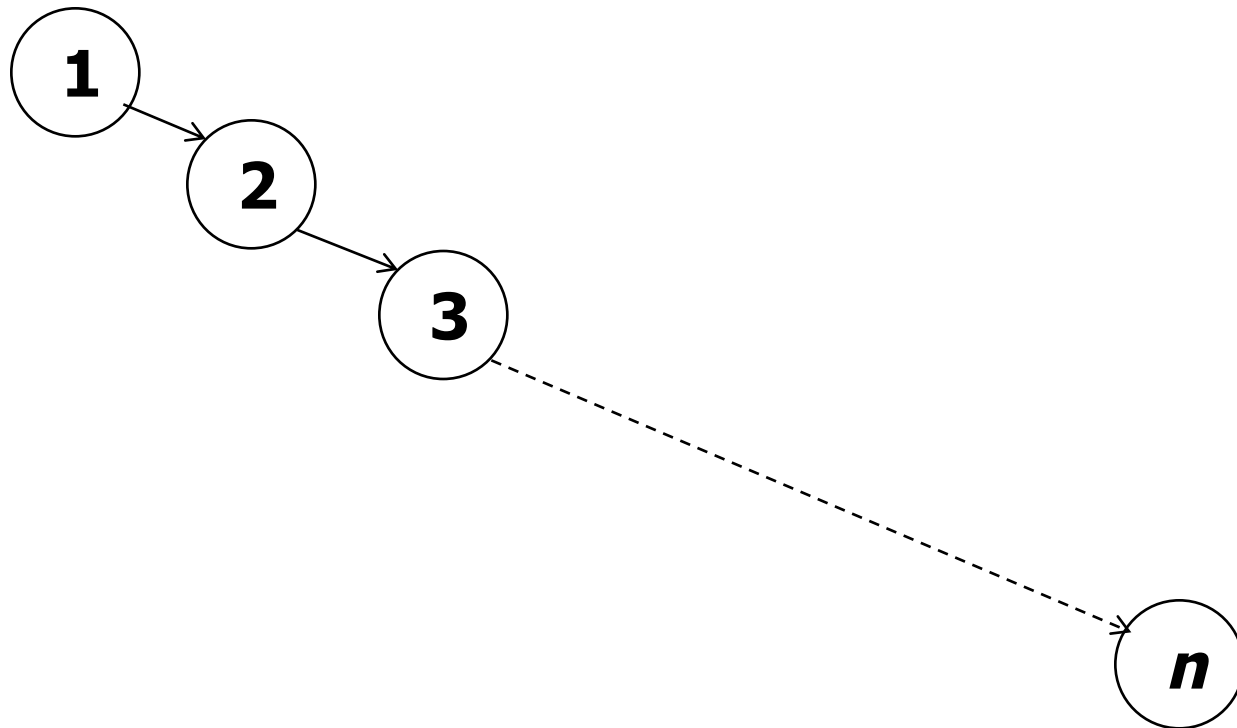
Die Höhe eines binären Suchbaums...

**Beispiel: Einfügen in
sortierter Reihenfolge**



Die Höhe eines binären Suchbaums...

Beispiel: Einfügen in sortierter Reihenfolge



Höhe = $n-1$, Baum sieht aus und verhält sich wie eine Liste.



Abhilfe: Balancierung

- Beim Einfügen und Löschen stets darauf achten, dass der Baum „balanciert“ ist, (d.h. keine grossen Höhenunterschiede zwischen linken und rechten Teilbäumen)



Abhilfe: Balancierung

- Beim Einfügen und Löschen stets darauf achten, dass der Baum „balanciert“ ist, (d.h. keine grossen Höhenunterschiede zwischen linken und rechten Teilbäumen)
- Verschiedene Möglichkeiten (mehr oder weniger kompliziert):
 - AVL-Bäume, Rot-Schwarz-Bäume,...



Abhilfe: Balancierung

- Beim Einfügen und Löschen stets darauf achten, dass der Baum „balanciert“ ist, (d.h. keine grossen Höhenunterschiede zwischen linken und rechten Teilbäumen)
- Verschiedene Möglichkeiten (mehr oder weniger kompliziert):
 - AVL-Bäume, Rot-Schwarz-Bäume,...
 - Hier: **Treaps** (randomisierte Suchbäume)



Treap = *Tree* + *Heap*

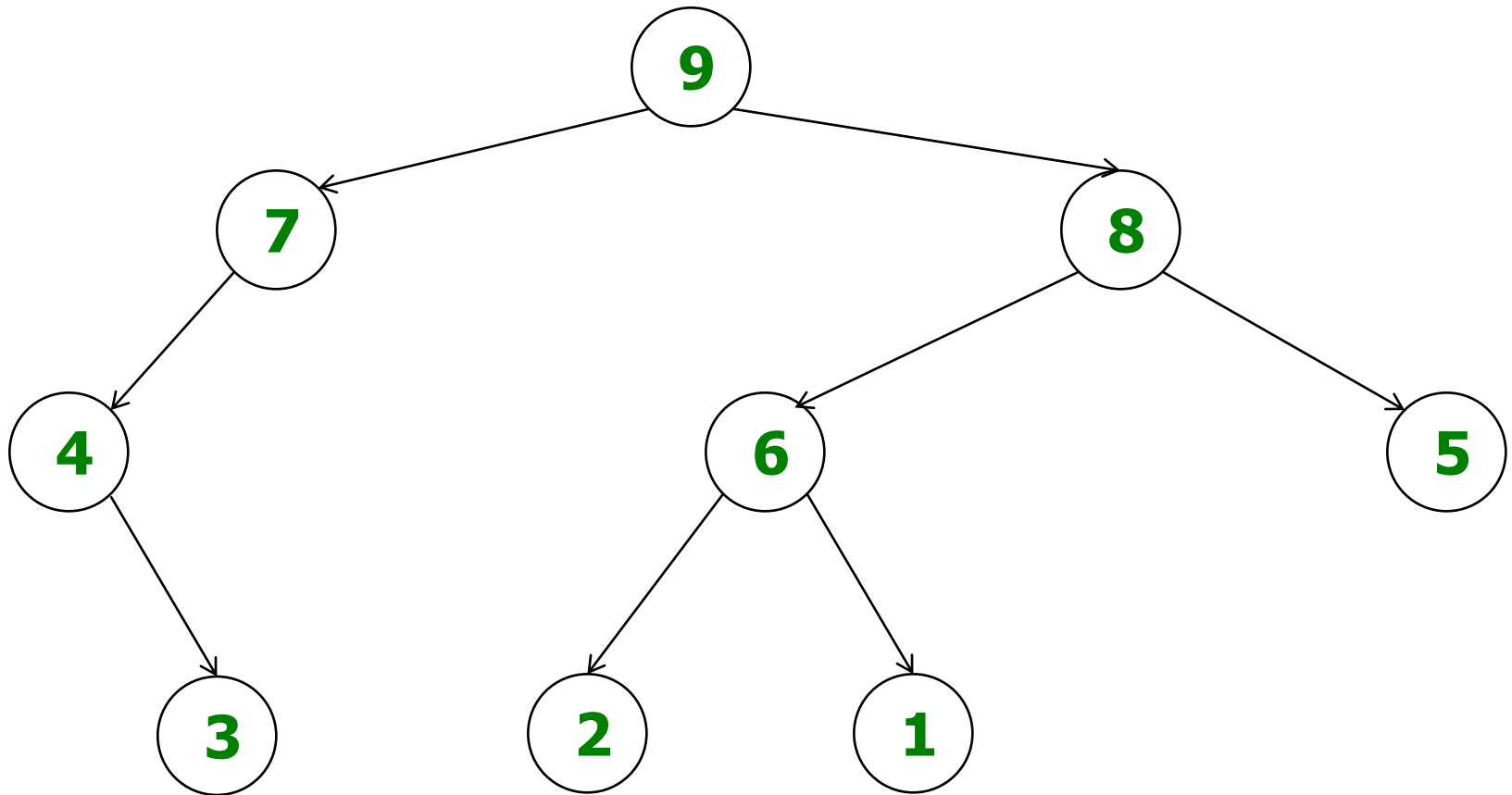
- Ein Binärbaum heisst *Heap*, wenn er leer ist, oder wenn folgende Bedingungen gelten:
 - Sowohl der linke als auch der rechte Teilbaum sind *Heaps*.
 - Der Schlüssel der Wurzel ist
 - das Maximum aller Schlüssel.

Heap: Schlüssel heissen

Prioritäten

- Ein Binärbaum heisst *Heap*, wenn er leer ist, oder wenn folgende Bedingungen gelten:
 - Sowohl der linke als auch der rechte Teilbaum sind Heaps.
 - Die **Priorität** der Wurzel ist
 - das Maximum aller **Prioritäten**.

Heap: Beispiel



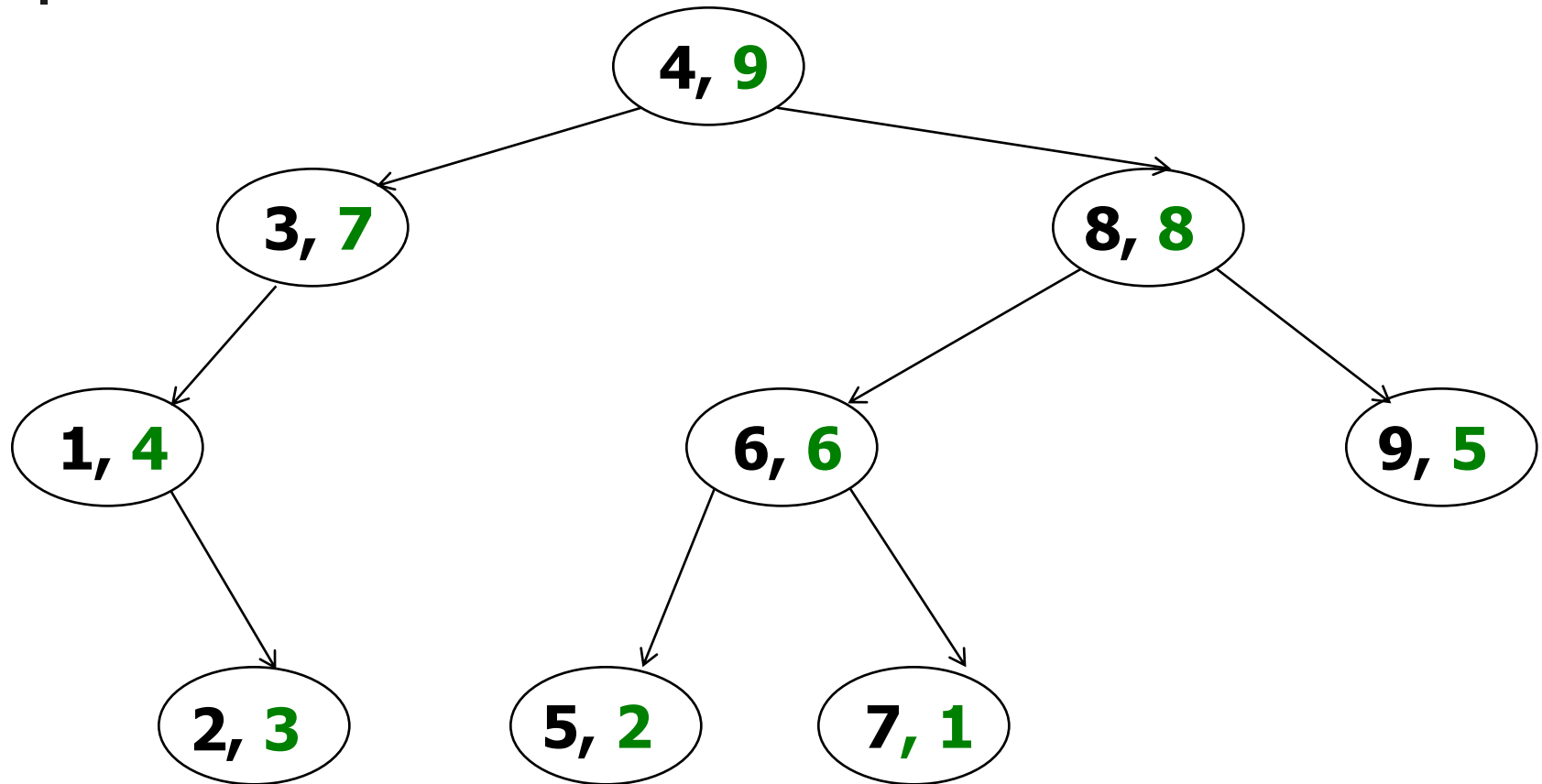


Treaps

- Ein Binärbaum heisst *Treap*, wenn jeder Knoten einen Schlüssel *und* eine Priorität hat, so dass folgende Eigenschaften gelten:
 - Der Baum ist ein binärer Suchbaum bzgl. der Schlüssel.
 - Der Baum ist ein Heap bzgl. der Prioritäten.

Treap: Beispiel

Schlüssel Prioritäten





Treaps: Eindeutigkeit

- **Satz:** Für n Knoten mit verschiedenen Schlüsseln *und* Prioritäten gibt es genau einen Treap mit diesen Knoten.



Treaps: Eindeutigkeit

- **Satz:** Für n Knoten mit verschiedenen Schlüsseln *und* Prioritäten gibt es genau einen Treap mit diesen Knoten.
- **Beweis:** Die Wurzel ist der Knoten mit der höchsten Priorität; im linken Teilbaum sind alle Knoten mit kleineren und im rechten Teilbaum alle Knoten mit grösseren Schlüsseln. Mit Induktion sind die Teilbäume auch eindeutig.



Treaps mit *zufälligen* Prioritäten

- Wir benutzen einen Treap als binären Suchbaum.



Treaps mit *zufälligen* Prioritäten

- Wir benutzen einen Treap als binären Suchbaum.
- Beim Einfügen eines neuen Schlüssels wird seine Priorität **zufällig** gewählt.



Treaps mit *zufälligen* Prioritäten

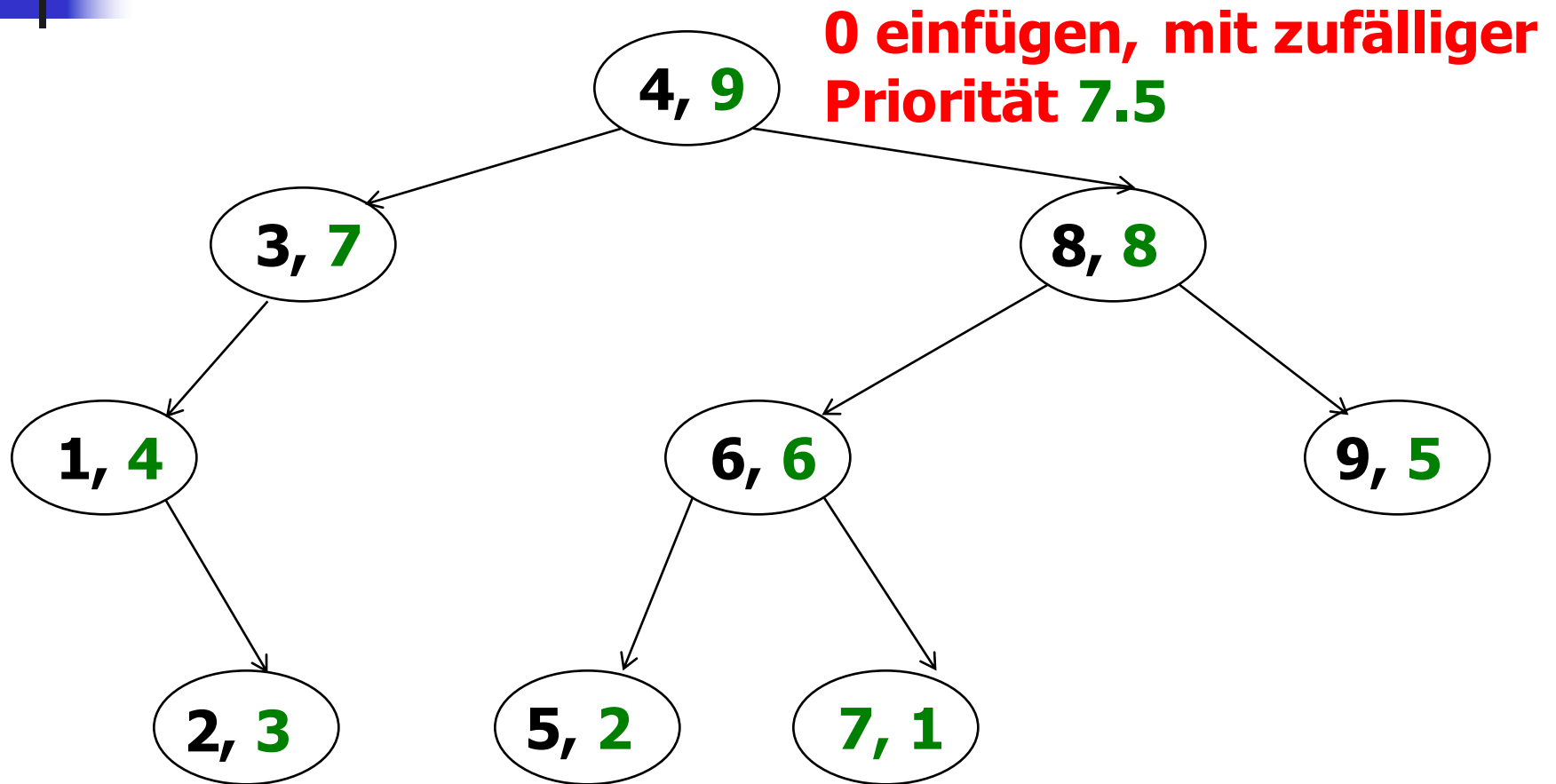
- Wir benutzen einen Treap als binären Suchbaum.
- Beim Einfügen eines neuen Schlüssels wird seine Priorität zufällig gewählt.
- Wie bei normalen binären Suchbäumen: neuer Knoten wird neues Blatt.

Treaps mit *zufälligen* Prioritäten

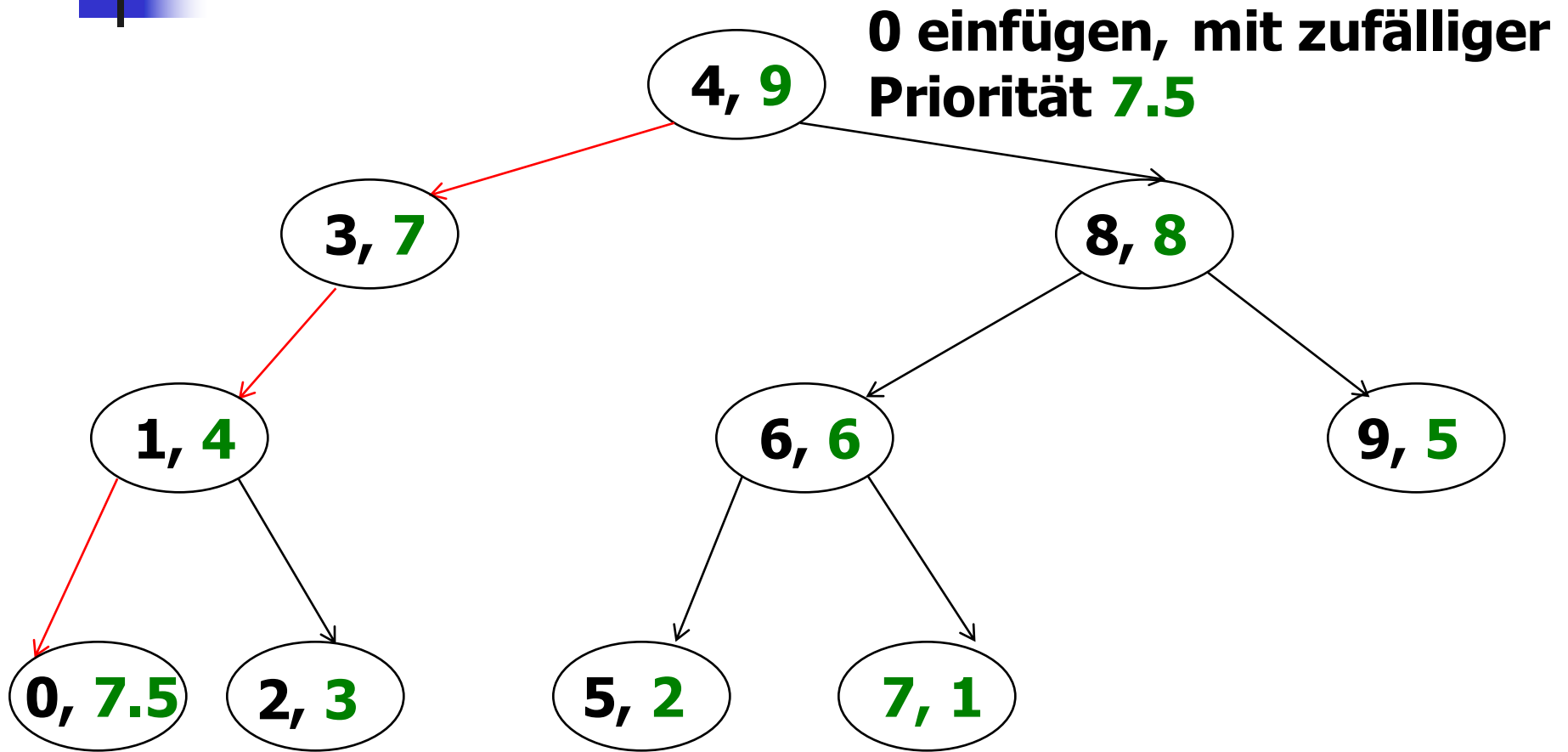


- Wir benutzen einen Treap als binären Suchbaum.
- Beim Einfügen eines neuen Schlüssels wird seine Priorität zufällig gewählt.
- Wie bei normalen binären Suchbäumen: neuer Knoten wird neues Blatt.
- **Zusätzlich muss die Treap-Eigenschaft wieder hergestellt werden!**

Treaps: Einfügen

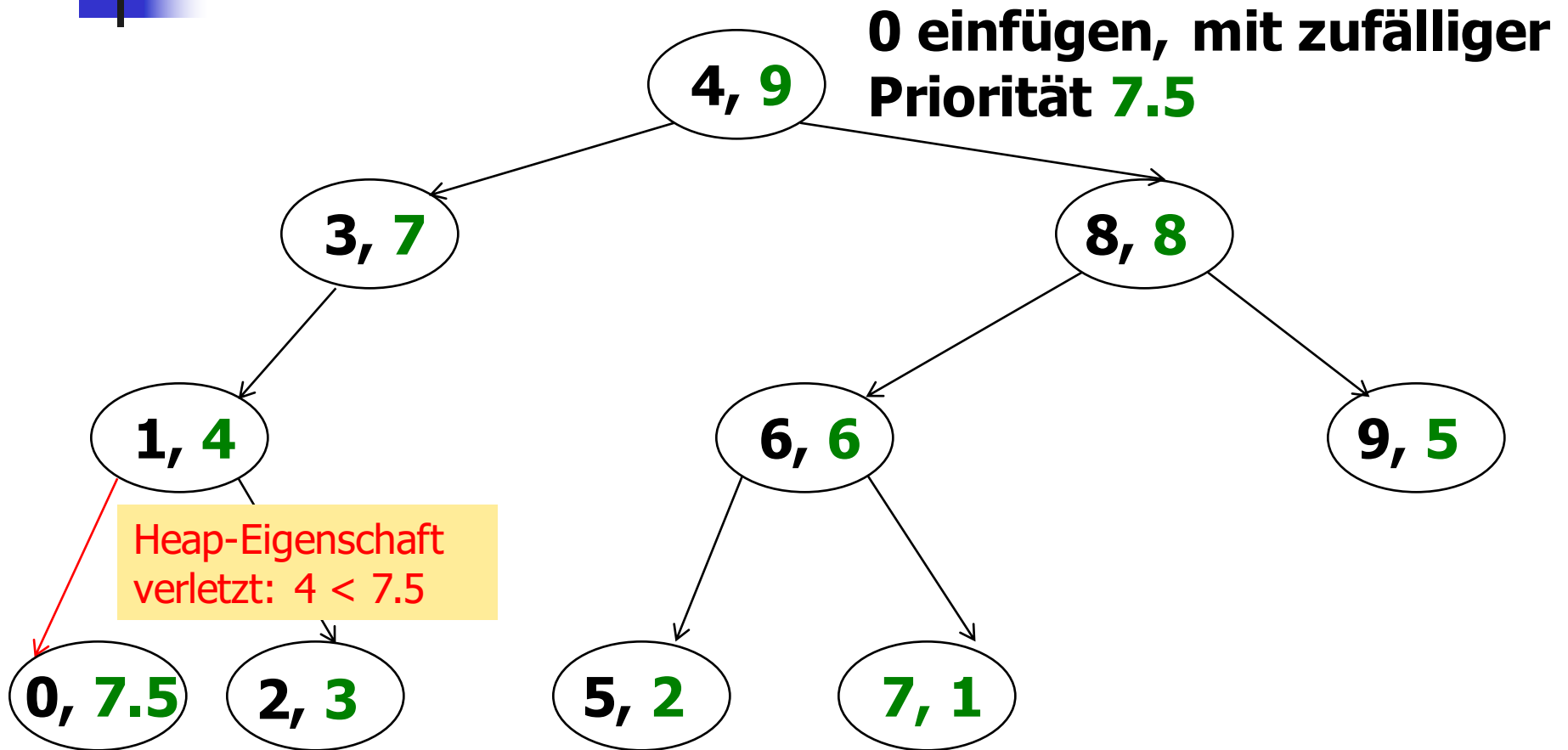


Treaps: Einfügen



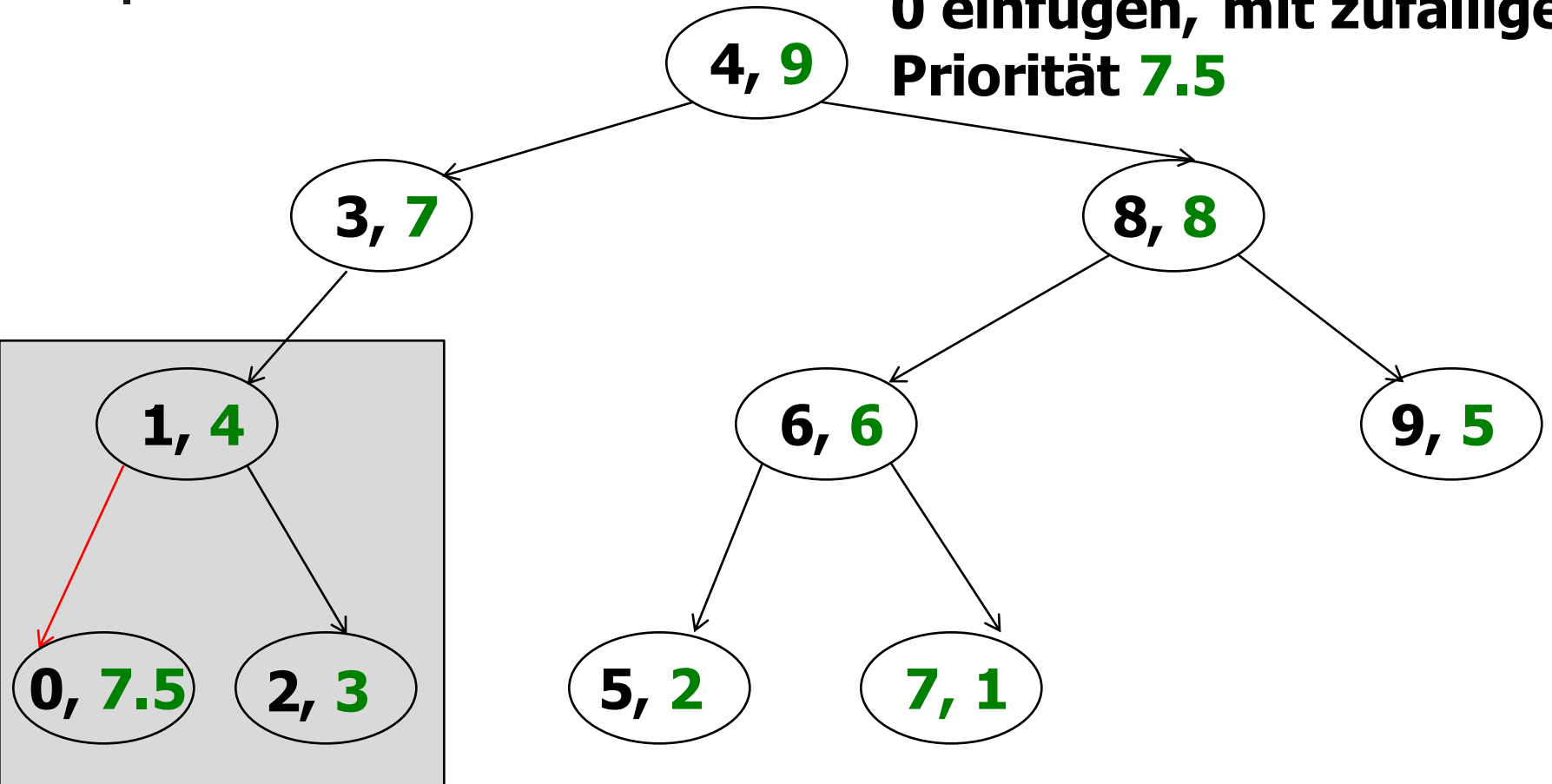
Neues Blatt

Treaps: Einfügen



Treaps: Einfügen

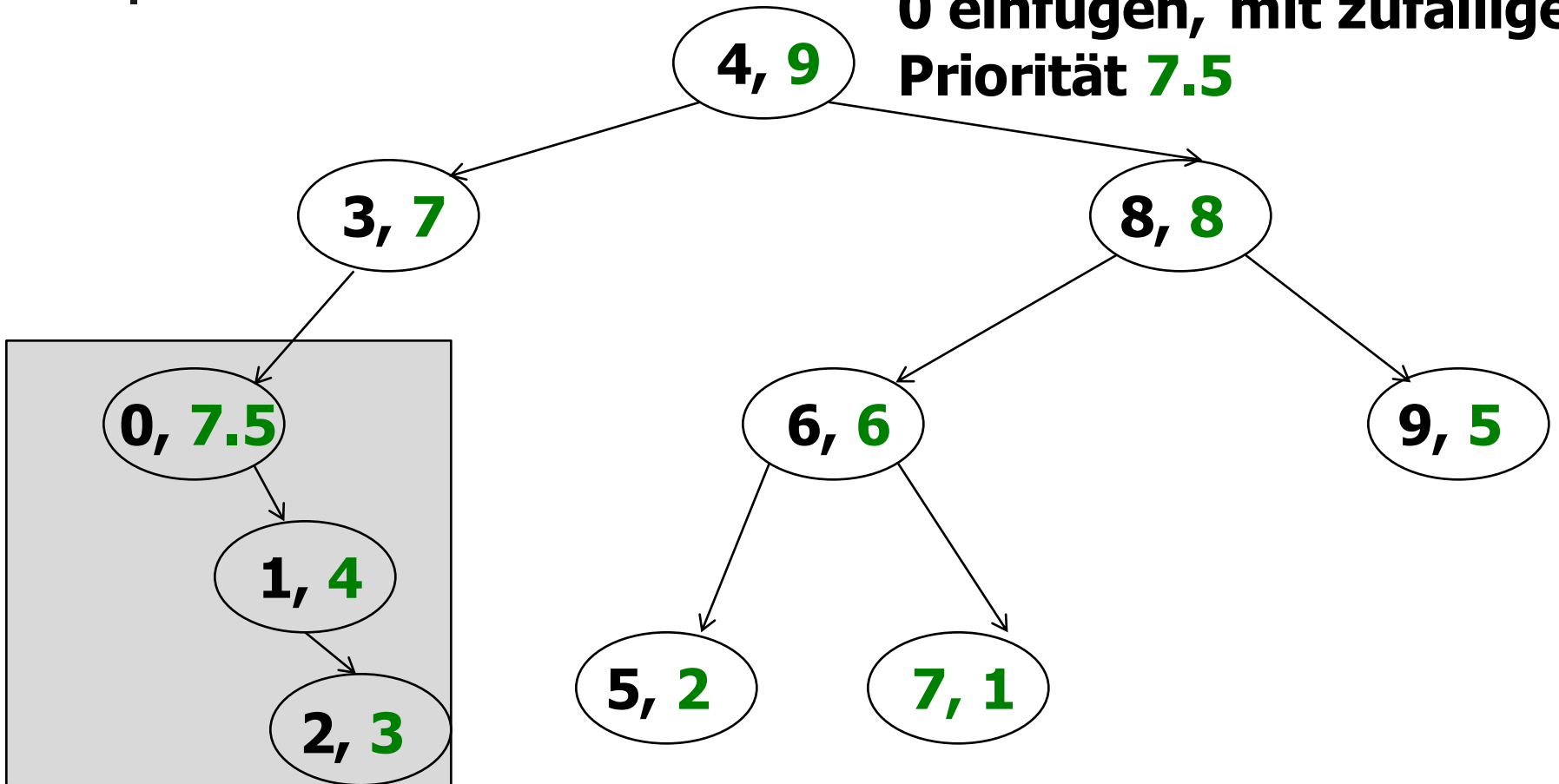
0 einfügen, mit zufälliger
Priorität **7.5**



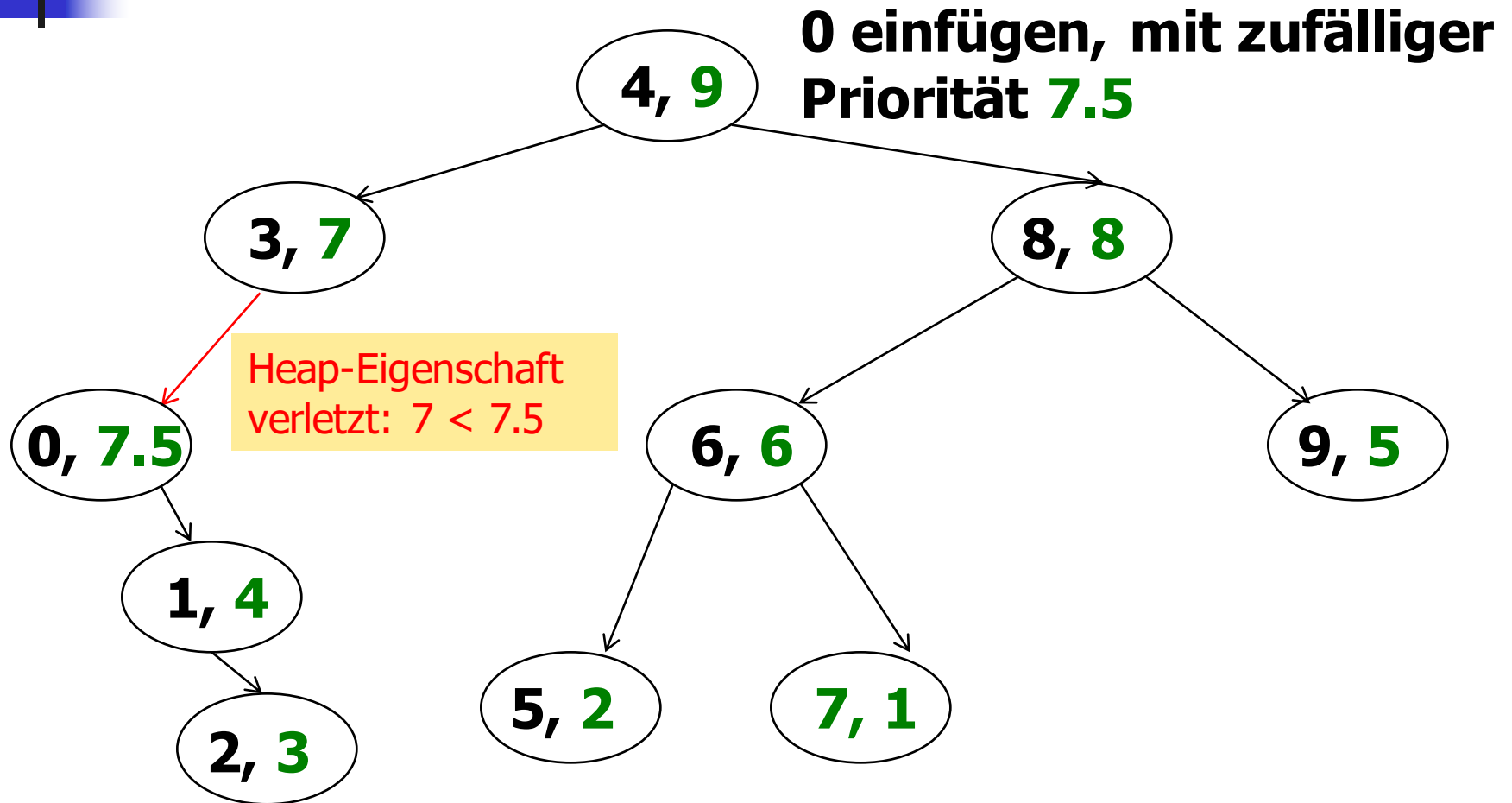
Rechtsrotation!

Treaps: Einfügen

0 einfügen, mit zufälliger
Priorität 7.5

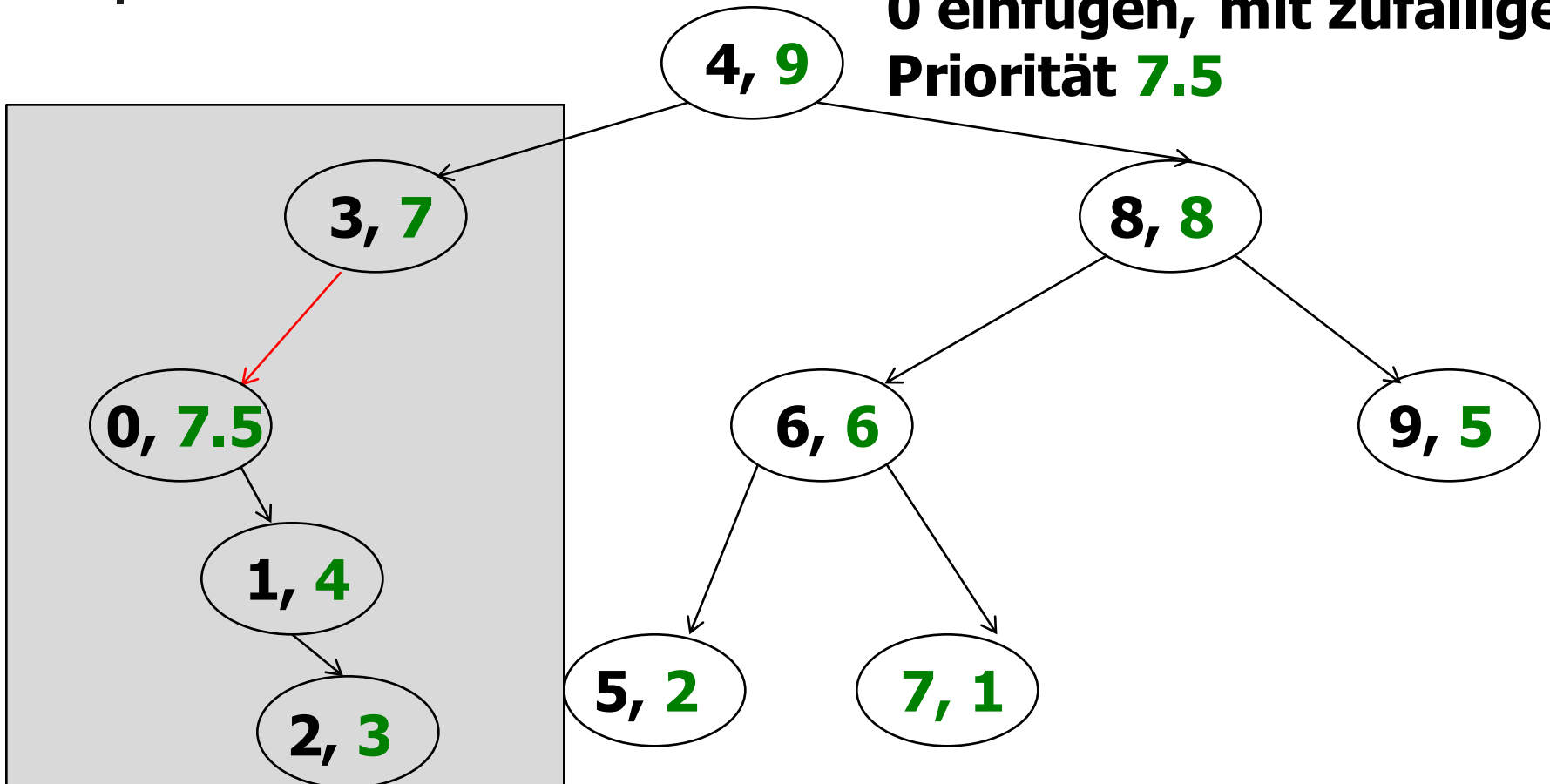


Treaps: Einfügen



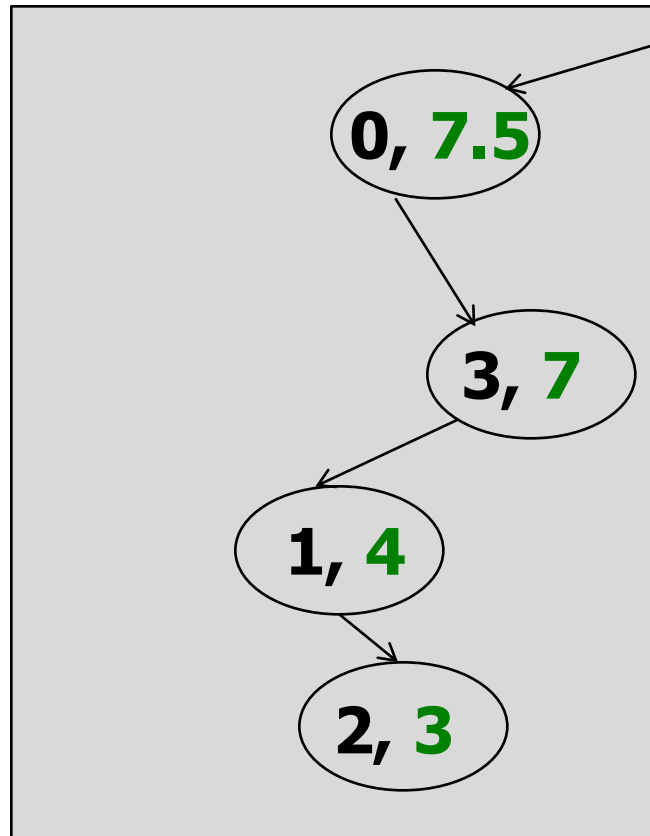
Treaps: Einfügen

0 einfügen, mit zufälliger
Priorität 7.5

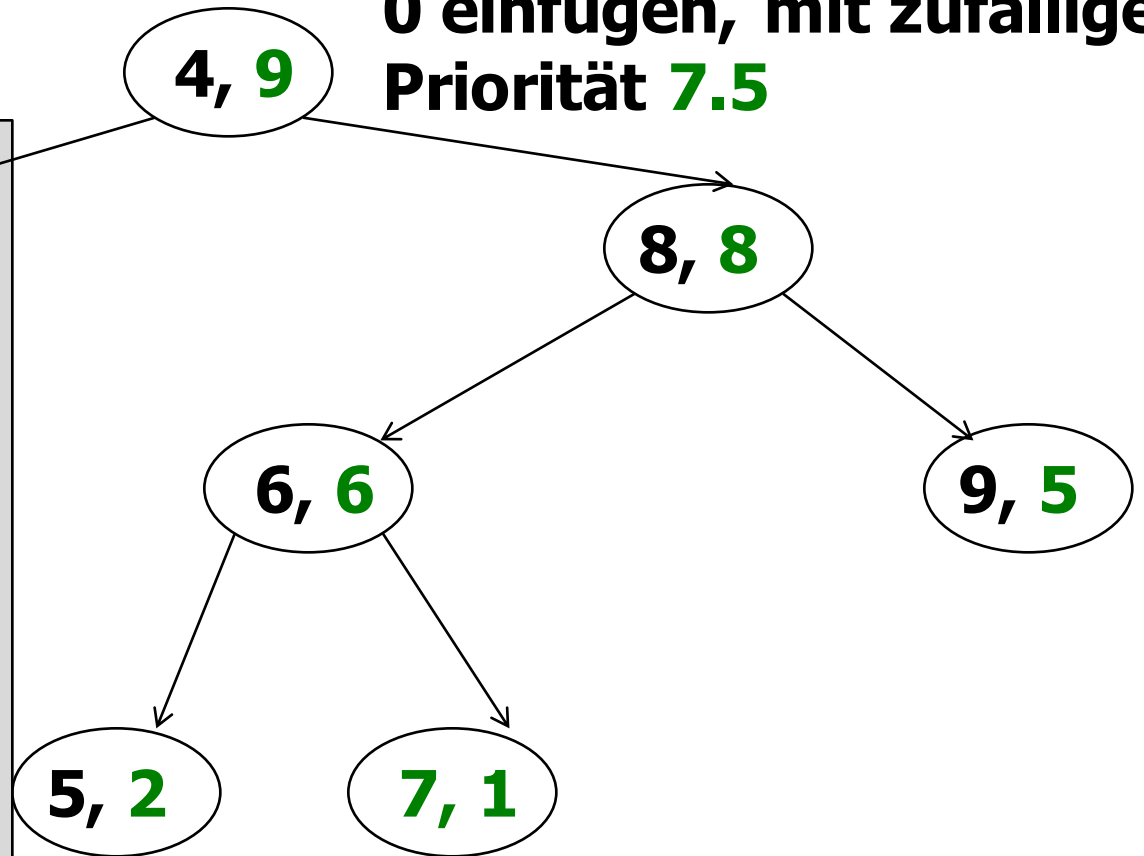


Rechtsrotation!

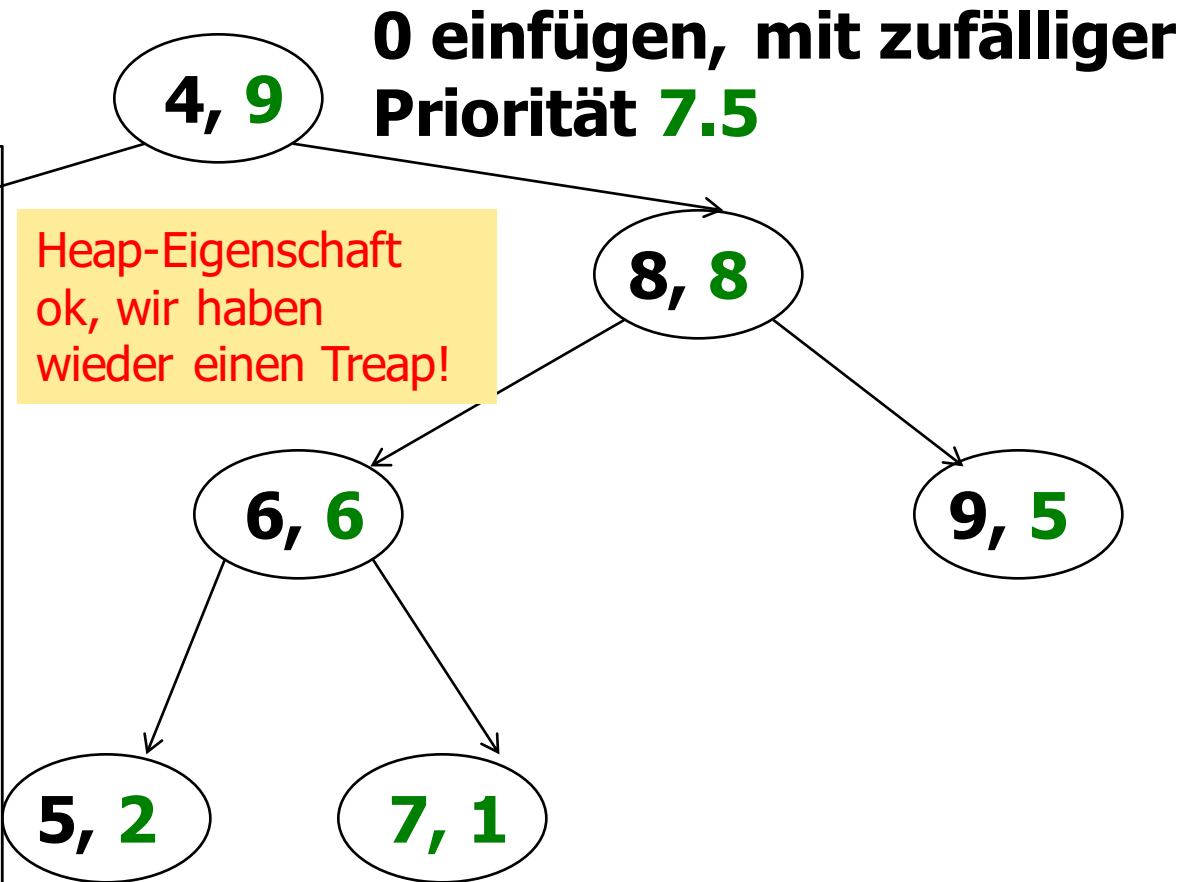
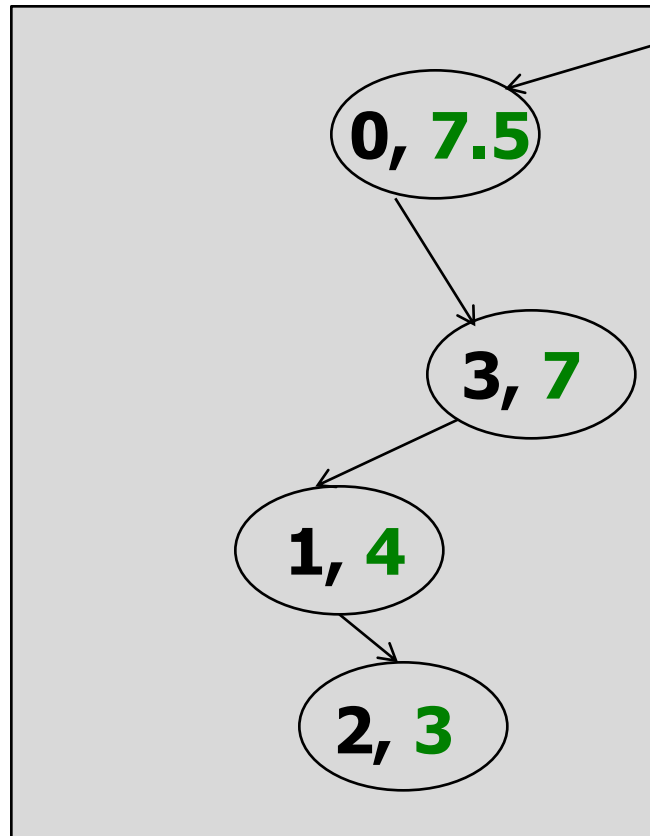
Treaps: Einfügen



0 einfügen, mit zufälliger
Priorität 7.5



Treaps: Einfügen





Treaps: Löschen

- Element suchen, seinen Knoten durch **Rotationen** zu einem Blatt machen, Blatt löschen!



Treaps: Löschen

- Element suchen, seinen Knoten durch **Rotationen** zu einem Blatt machen, Blatt löschen!
- Stets so rotieren, dass der Knoten mit seinem Nachfolger **höherer Priorität** vertauscht wird. Das erhält die Heap-Eigenschaft.



Treaps: Löschen

- Element suchen, seinen Knoten durch **Rotationen** zu einem Blatt machen, Blatt löschen!
- Stets so rotieren, dass der Knoten mit seinem Nachfolger **höherer Priorität** vertauscht wird. Das erhält die Heap-Eigenschaft.
- Suchbaum-Eigenschaft gilt wieder nach Löschen des Blatts.



Treaps: Laufzeit

- **Satz** (Seidel, Aragon 1996): In einem Treap über n Knoten mit zufälligen Prioritäten ist der zeitliche Aufwand für eine Einfüge-, Lösch- oder Suchoperation **im Erwartungswert** proportional zu $\log_2 n$.



Treaps: Laufzeit

Randomized Search Trees

RAIMUND SEIDEL*

Computer Science Division
University of California Berkeley
Berkeley CA 94720

Fachbereich Informatik
Universität des Saarlandes
D-66041 Saarbrücken, GERMANY

CECILIA R. ARAGON†

Computer Science Division
University of California Berkeley
Berkeley CA 94720



Mengen: Lösung mit Treaps

- Zur Erinnerung: Suche nach den letzten 100 Elementen in einer **Liste** von 10,000,000 Elementen

```
List l;
```

```
for (int i=0; i<10000000; ++i)
```

```
    l.push_front(i);
```

```
// ...and search for i = 0..99
```

```
for (int i=0; i<100; ++i)
```

```
    l.find(i);
```

6.5 Sekunden



Mengen: Lösung mit Treaps

- Neu: Suche nach den gleichen 100 Elementen in einem **Treap** von 10,000,000 Elementen

```
Treap t;
```

```
for (int i=0; i<10000000; ++i)  
    t.insert(i);
```

```
// ...and search for i = 0..99
```

```
for (int i=0; i<100; ++i)  
    t.find(i);
```

0.017 Sekunden



Sortieren mit Treaps

- Folgerung: Durch Einfügen einer Folge von n Zahlen in einen Treap mit zufälligen Prioritäten kann die Folge in Zeit proportional zu $n \log_2 n$ sortiert werden.

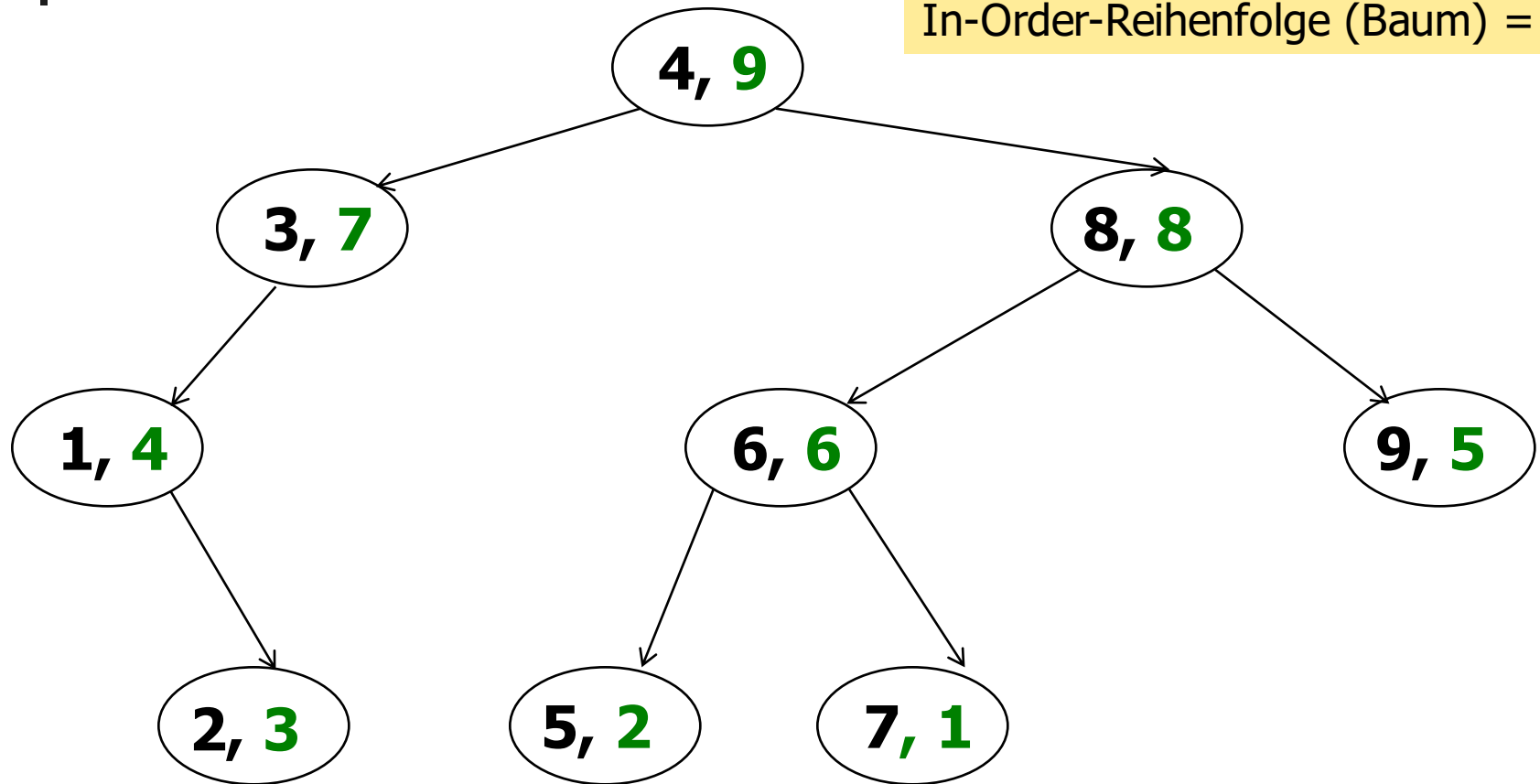


Sortieren mit Treaps

- Folgerung: Durch Einfügen einer Folge von n Zahlen in einen Treap mit zufälligen Prioritäten kann die Folge in Zeit proportional zu $n \log_2 n$ sortiert werden.
- Sortierte Reihenfolge: Elemente des Treaps „von links nach rechts“ (*in-order*-Reihenfolge)

Sortieren mit Treaps

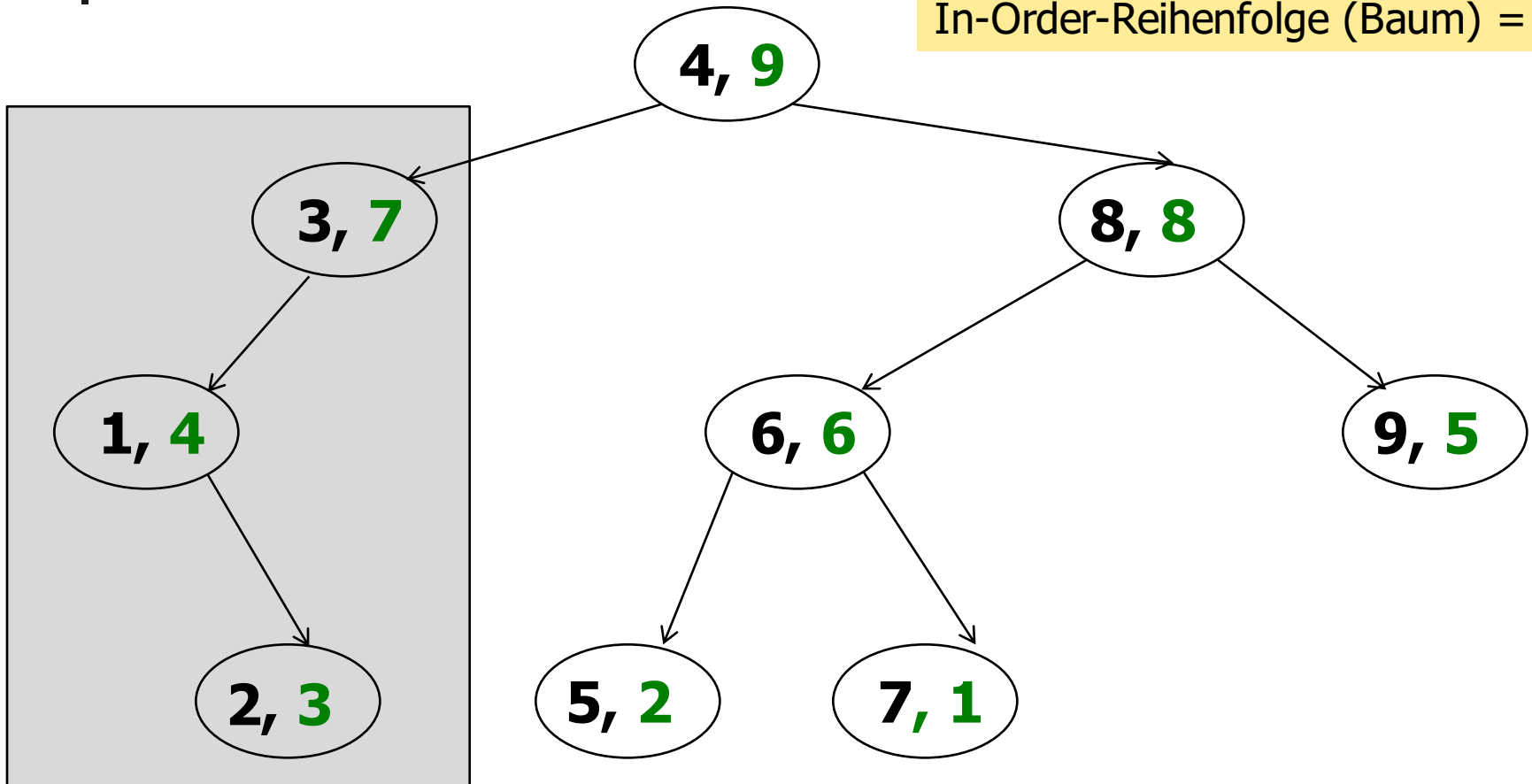
In-Order-Reihenfolge (Baum) =



1,2,3

Sortieren mit Treaps

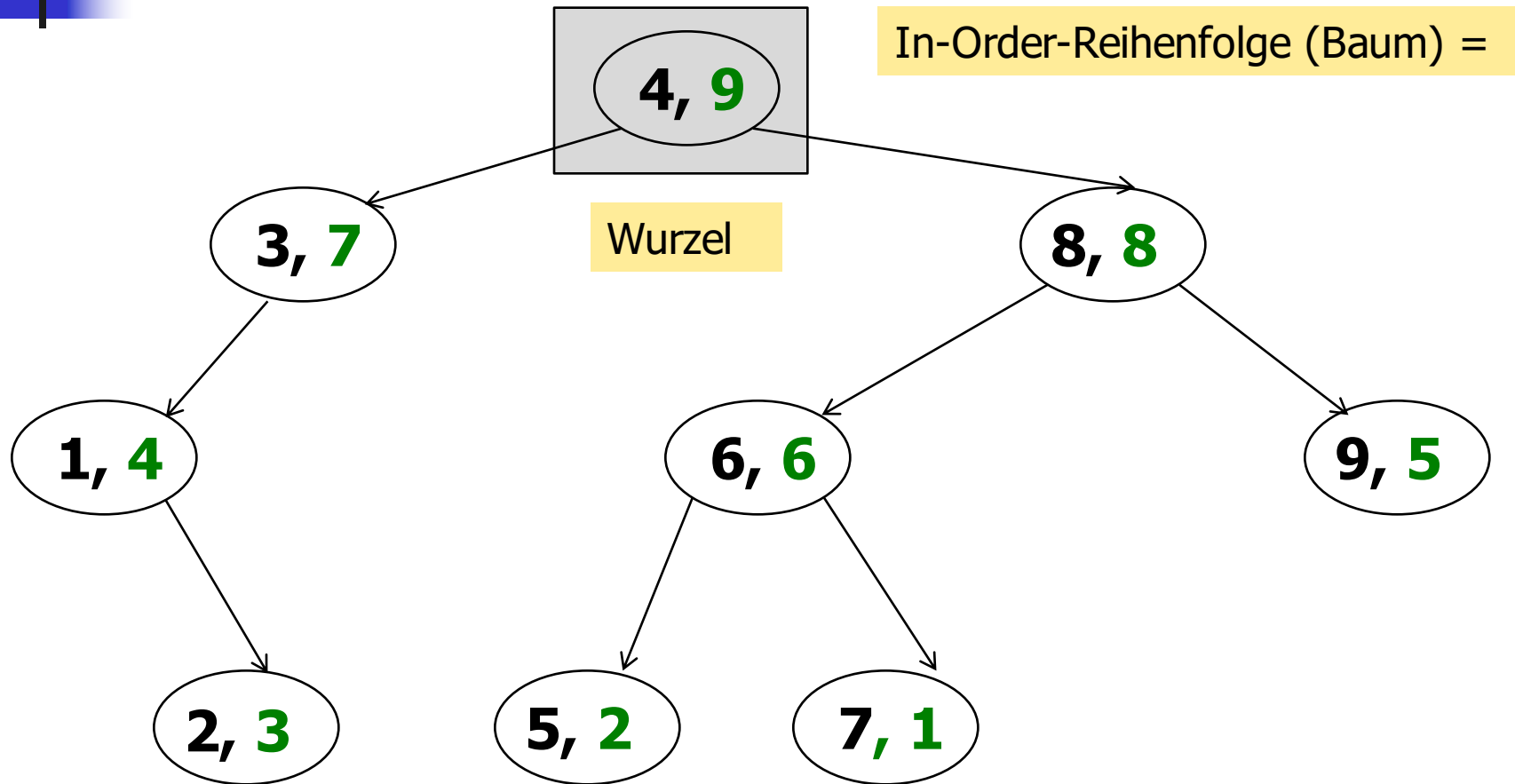
In-Order-Reihenfolge (Baum) =



In-Order-Reihenfolge (linker Teil Baum)

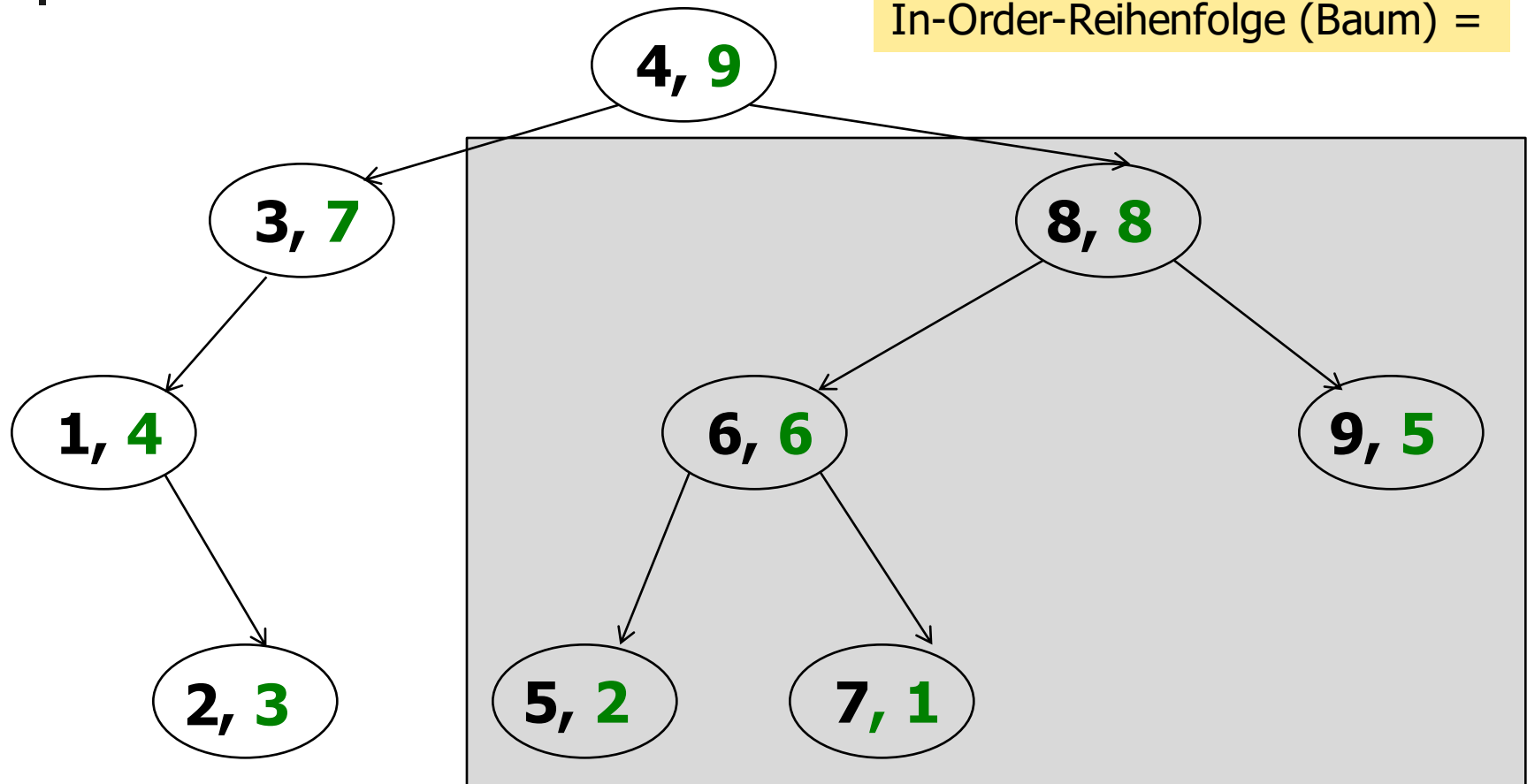
1,2,3,4

Sortieren mit Treaps



1,2,3,4,5,6,7,8,9

Sortieren mit Treaps



In-Order-Reihenfolge (Baum) =

In-Order-Reihenfolge (rechter Teil Baum)



Sortieren mit Treaps

- Folgerung: Durch Einfügen einer Folge von n Zahlen in einen Treap mit zufälligen Prioritäten kann die Folge in Zeit proportional zu $n \log_2 n$ sortiert werden.
- Das Treap-basierte Verfahren ist im wesentlichen äquivalent zu **Quicksort**.