Lektion 12

Datenstrukturen: Listen und Bäume

Datenstrukturen

Listen und Bäume

Nehmen wir an wir wollen die Einträge
für ein Telefonbuch speichern.

Wir kennen bisher:

Ein- und mehrdimensionale Arrays
-> um mehrere gleichartige Daten zu verarbeiten
-> „Ein Array von Einträgen in dem Telefonbuch“

Klassen
-> zur Definition komplexerer Datentypen
-> Eintrag im Telefonbuch
bestehend aus Name und Telefonnummer

Welche Probleme treten bei dem Ansatz auf?

Arrays haben eine feste Dimension

Wir können bspw. ein Telefonbuch für ein Dorf mit 20000 Einwohnern anlegen.

Was passiert, wenn im Laufe der Jahre die Einwohnerzahl darüber hinaus wächst?

Speicher ist in der Größenordnung kein Problem mehr! Aber was passiert bei Millionen oder Milliarden Einträgen?

Welche Probleme treten bei dem Ansatz auf?

Die Einträge sind nach Nachnamen sortiert!
Was passiert, wenn Einwohner sich an- und abmelden?

Beim Abmelden/Löschen entstehen Lücken im Array

Entweder lässt man die Lücken leer

oder man rückt regelmäßig alle Einträge im Array auf

Speicherverschwendung, wenn sich Daten häufig ändern

erhöhter Prozessorbedarf

Welche Probleme treten bei dem Ansatz auf?

Die Einträge sind nach Nachnamen sortiert!
Was passiert, wenn Einwohner sich an- und abmelden?

Beim Einfügen in das Array geht i.d.R.
die Sortierung verloren

Entweder Array neu
sortieren

erhöhter Speicherbedarf
und Prozessorbedarf

oder man schiebt alle
nachfolgenden Einträge und
fügt direkt sortiert ein

erhöhter Prozessorbedarf

Zusammenfassung des Problems

- Wir kennen bisher:
  - Ein- und mehrdimensionale Arrays
- Aber:
  - Länge des Arrays ist nach der Erzeugung festgelegt.
  - sortiertes Einfügen ist nicht ohne größeren Aufwand möglich
  - Löschen von Einträgen hinterlässt Lücken
- Hilfreich wäre eine Datenstruktur ähnlich einem Array,
  - in die man an beliebiger Stelle neue Elemente einfügen könnte,
  - aus der man Elemente löschen könnte, ohne Lücken zu hinterlassen,
  - die man problemlos vergrößern könnte.

einfach verkettete Liste

- Eine Liste ist eine Verknüpfung von meist gleichartigen Variablen eines Datentyps, z.B. von Objekten einer Klasse.
- Jeder **Knoten** (d. h. jedes Element) der Liste kennt dabei seinen Nachfolger in der Liste.
- Der Startknoten der Liste ist bekannt.
- Wie sieht ein Listenknoten für einen Telefonbucheintrag aus?

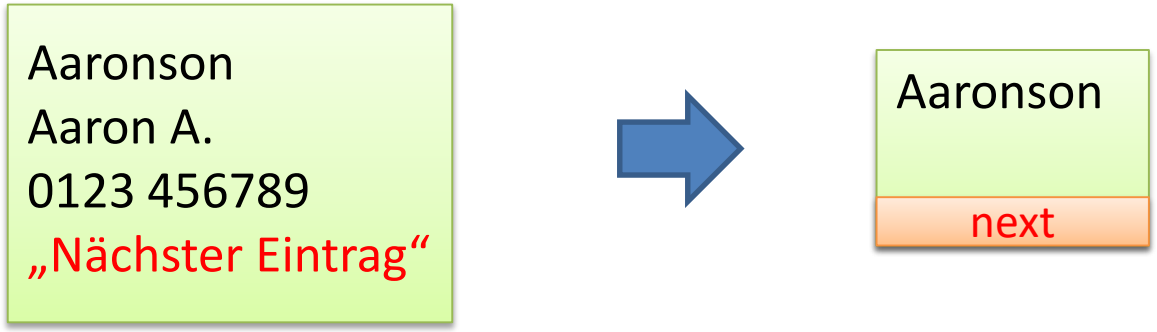
Aaronson
Aaron A.
0123 456789
„Nächster Eintrag“

```
public class Knoten {
  String nachname;
  String vorname;
  String telefonnummer;
  Knoten next;

  public Knoten(String nachname) {
    this.nachname = nachname;
  }
  //getter und setter
}
```

Darstellung

- Wir verwenden für die Listenknoten eine vereinfachte Darstellung. Diese beinhaltet:
  - Nachname, weil danach sortiert wird
  - next, als Verweis auf den nächsten Listenknoten

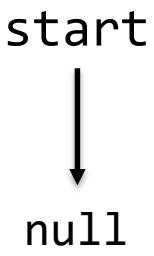
Vereinfachung der Darstellung



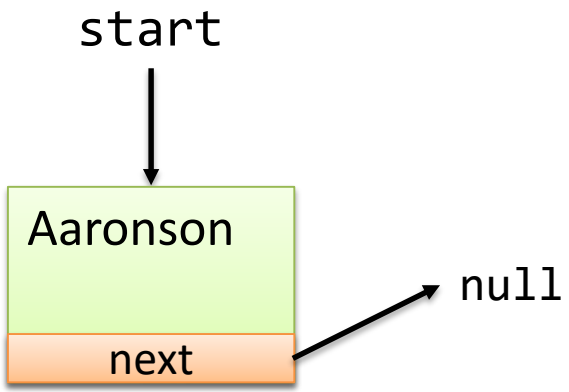
Aufbau einer einfach verketteten Liste

- `start` ist eine Referenz auf den ersten Knoten der Liste
- das letzte Listenelement verweist auf `null`

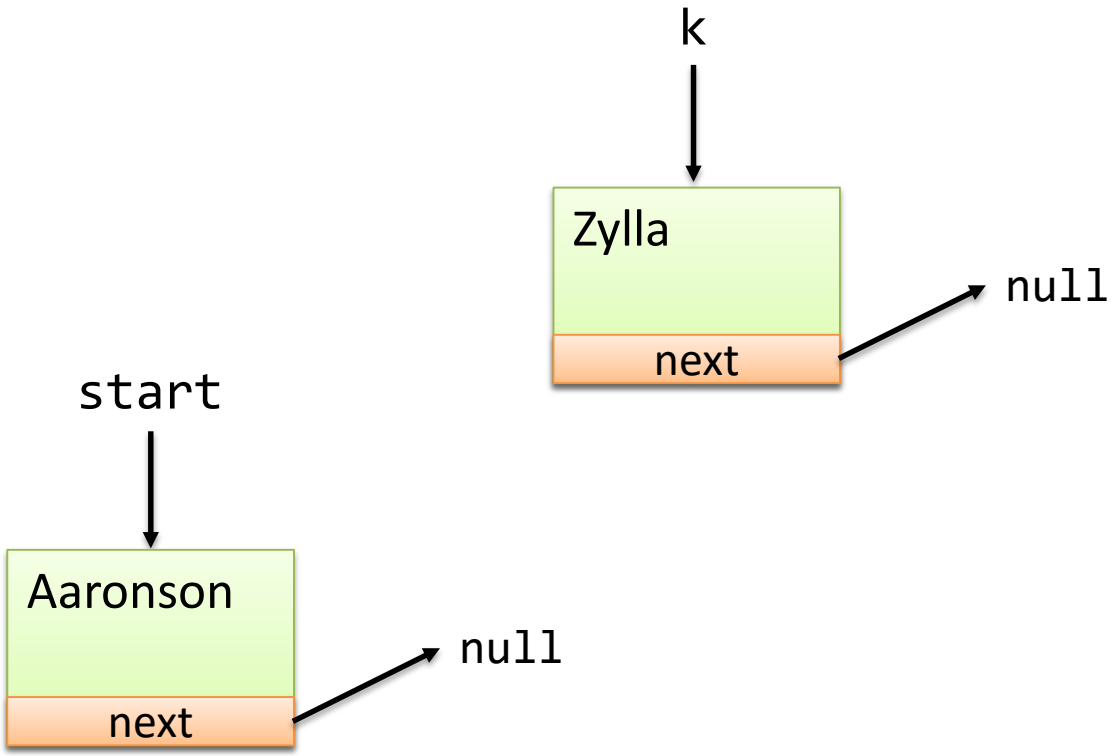
```
Knoten start = null;
```



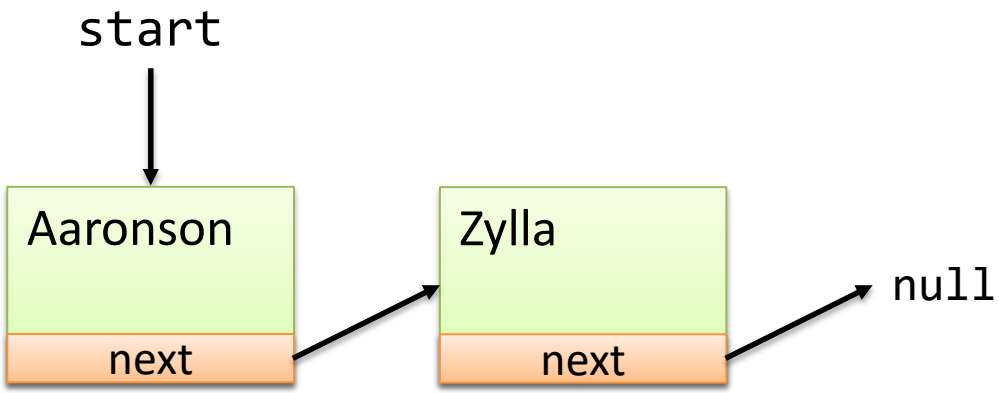
```
Knoten start = null;
start = new Knoten("Aaronson");
```



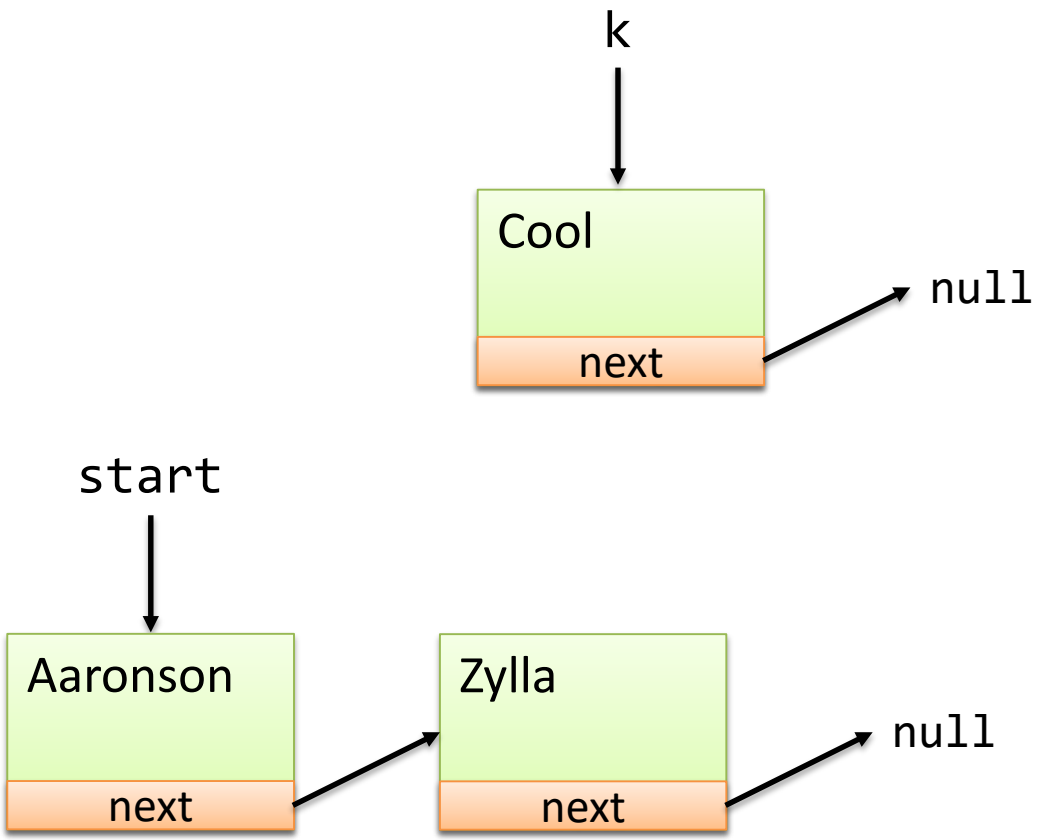
```
Knoten start = null;
start = new Knoten("Aaronson");
Knoten k;
k = new Knoten("Zylla");
```



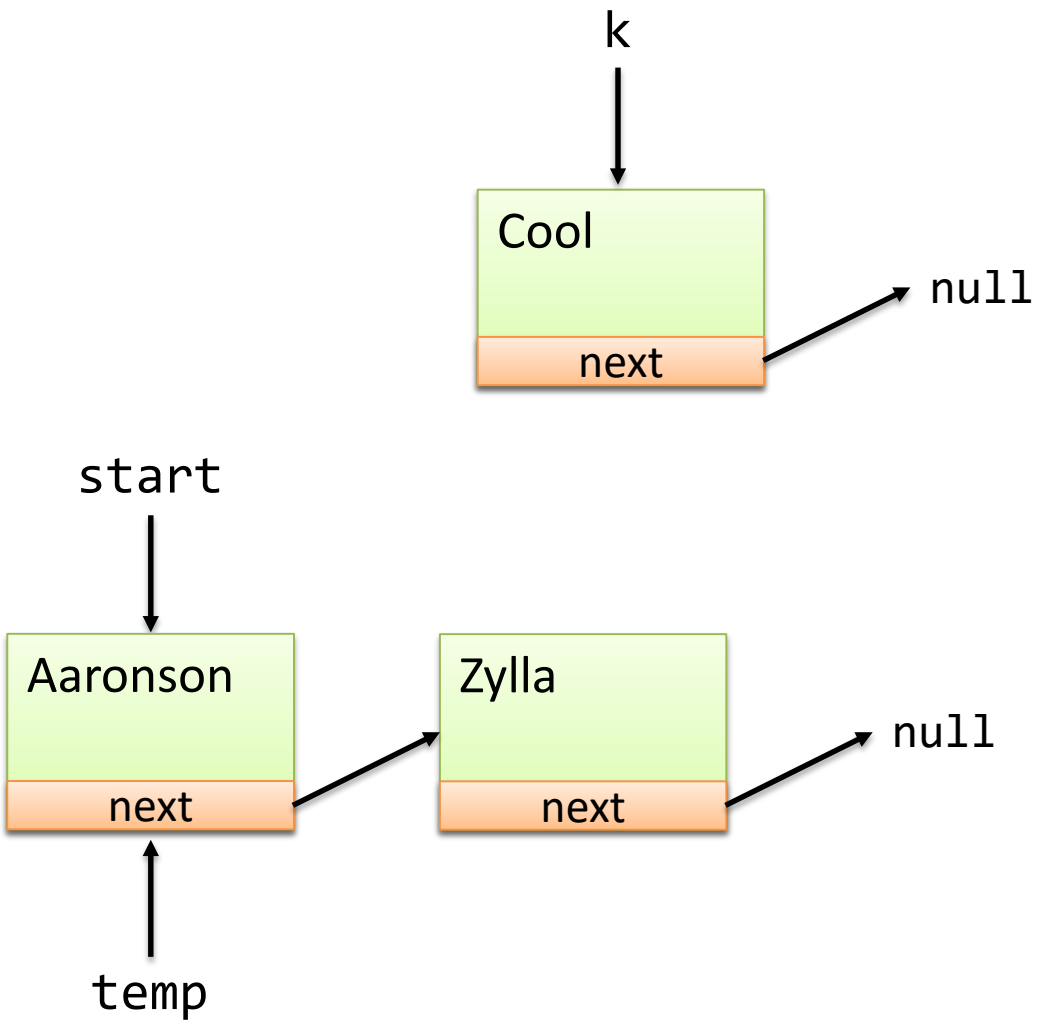
```
Knoten start = null;
start = new Knoten("Aaronson");
Knoten k;
k = new Knoten("Zylla");
start.next = k;
```



```
k = new Knoten("Cool");
```

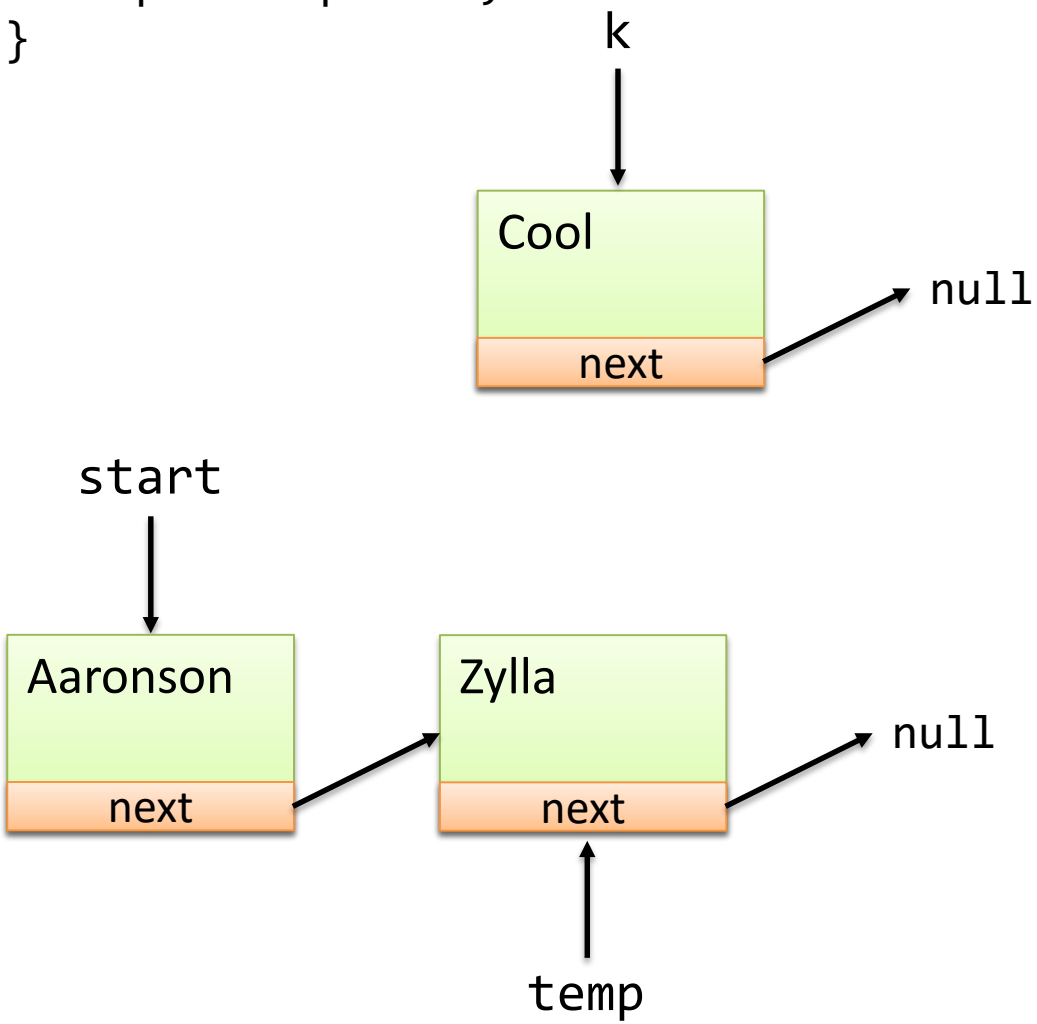


```
k = new Knoten("Cool");
Knoten temp = start;
```

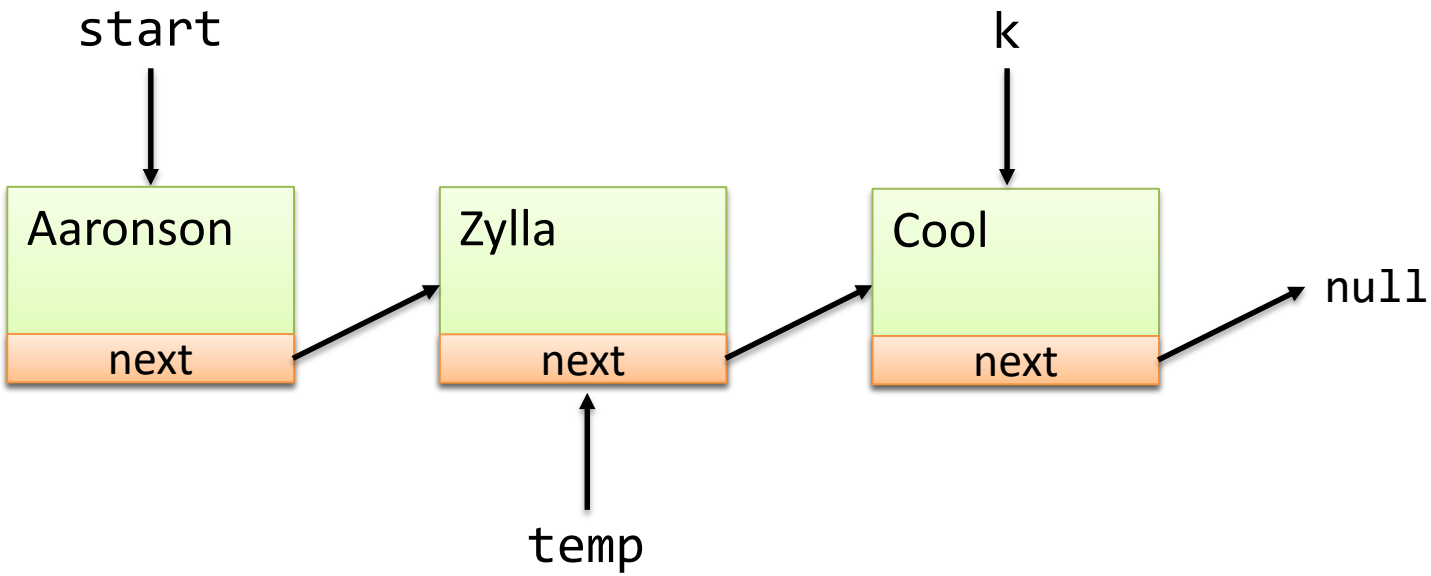


```
k = new Knoten("Cool");
Knoten temp = start;
while (temp.next != null) {
  temp = temp.next;
}
```

Laufe zum Ende der Liste



```
k = new Knoten("Cool");
Knoten temp = start;
while (temp.next != null) {
  temp = temp.next;
}
temp.next = k;
```



Liste durchlaufen und ausgeben

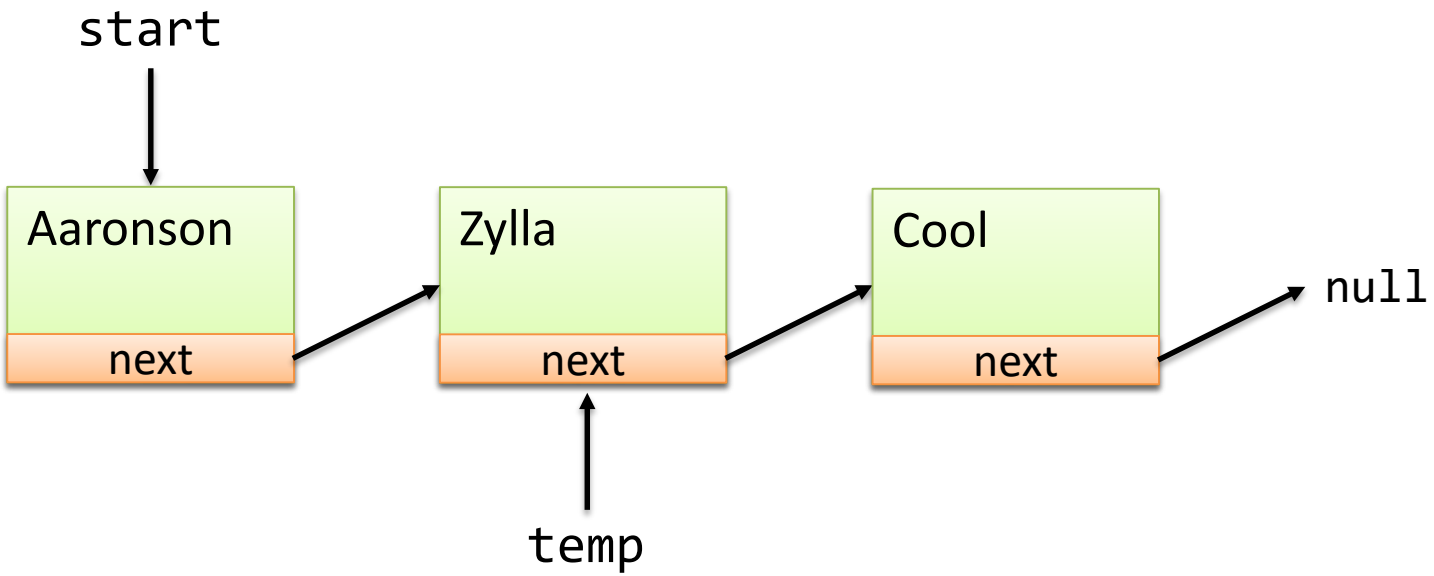
```
Knoten temp = start;
while (temp != null) {
  System.out.println(temp.getNachname());
  temp = temp.next;
}
```

- temp wird auf den Anfang der Liste gesetzt
- Solange die temp-Referenz nicht null ist (d.h. solange es weitere Elemente gibt)…
- …wird der Nachname ausgegeben und temp referenziert das nächste Element der Liste

```
k = new Knoten("Cool");
Knoten temp = start;
while (temp.next != null) {
  temp = temp.next;
}
temp.next = k;
```

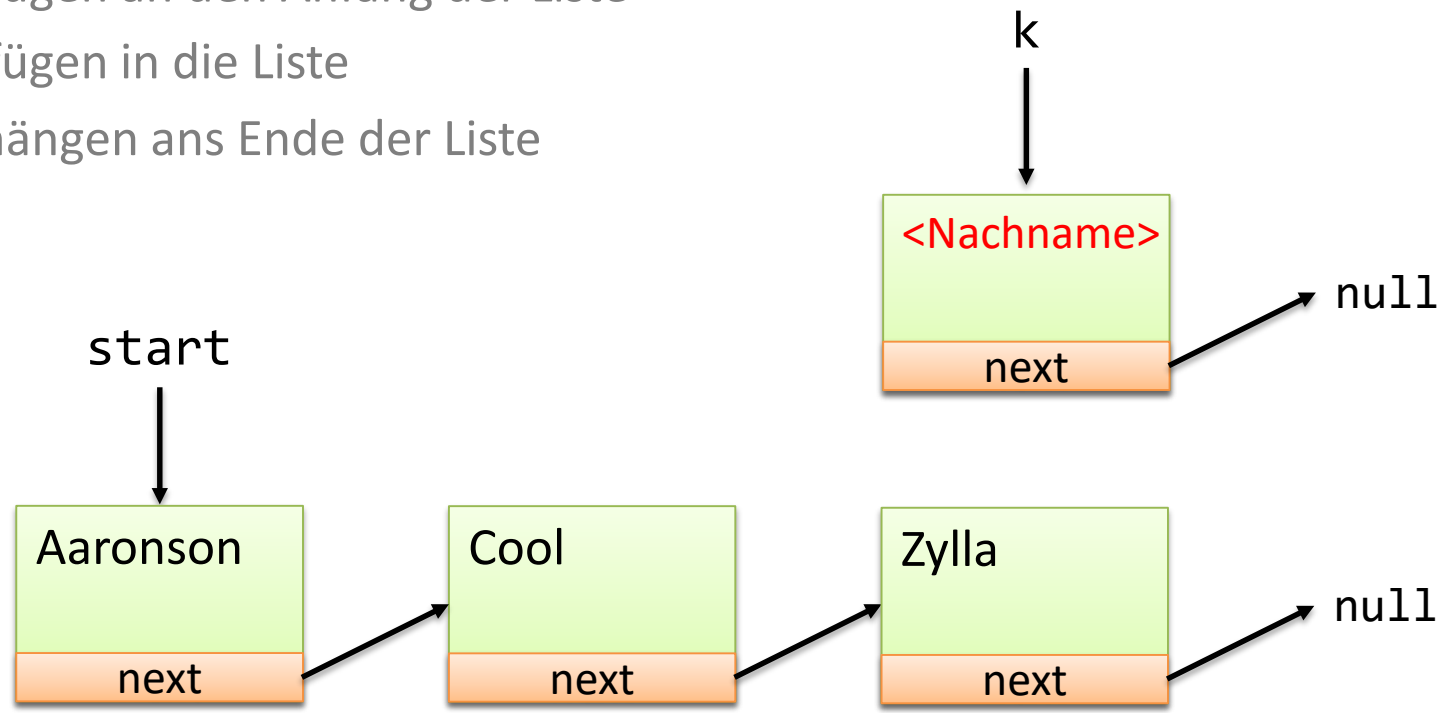
Wir haben das Element einfach an das Listenende eingefügt!

Wie können wir die Sortierung nach Nachnamen sicherstellen?



Sortiertes Einfügen

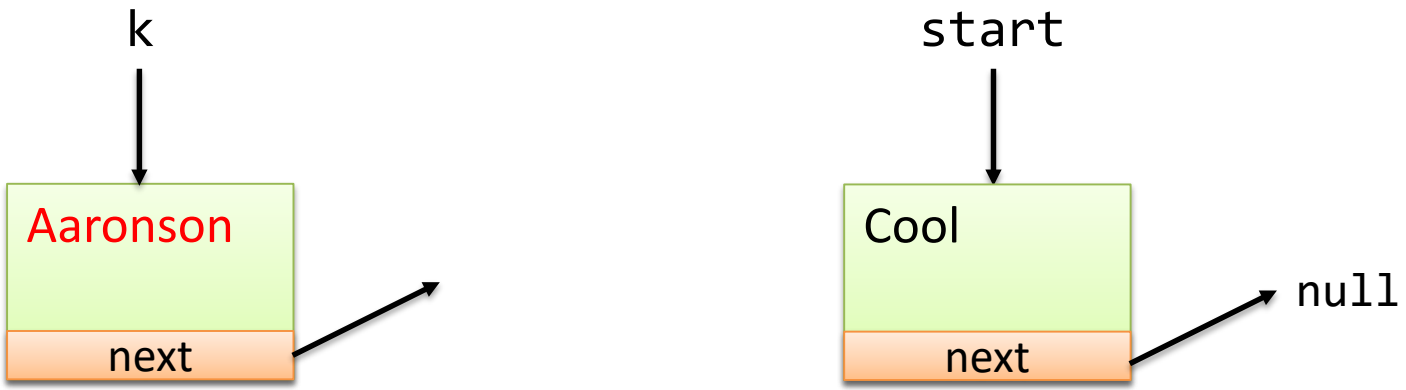
- Es gibt drei mögliche Fälle für das Einfügen neuer Element abhängig vom Nachnamen:
  - Einfügen an den Anfang der Liste
  - Einfügen in die Liste
  - Anhängen ans Ende der Liste



Überprüfung, ob der neue Knoten vor dem Startknoten eingefügt werden muss

```
Scanner scanner = new Scanner(System.in);
String nachname = scanner.nextLine();
Knoten k = new Knoten(nachname);
…
if (k.getNachname().compareTo(start.getNachname()) < 0)
{



}
```

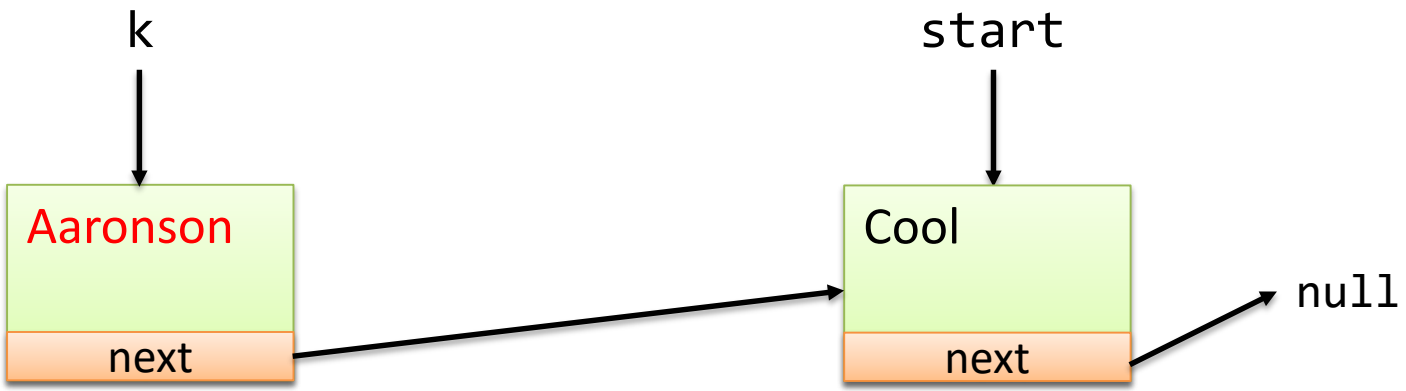
Aaronson



Überprüfung, ob der neue Knoten vor dem Startknoten eingefügt werden muss

```
Scanner scanner = new Scanner(System.in);
String nachname = scanner.nextLine();
Knoten k = new Knoten(nachname);
…
if (k.getNachname().compareTo(start.getNachname()) < 0)
{
  k.next = start; //k hat den Startknoten als Nachfolger

}
```

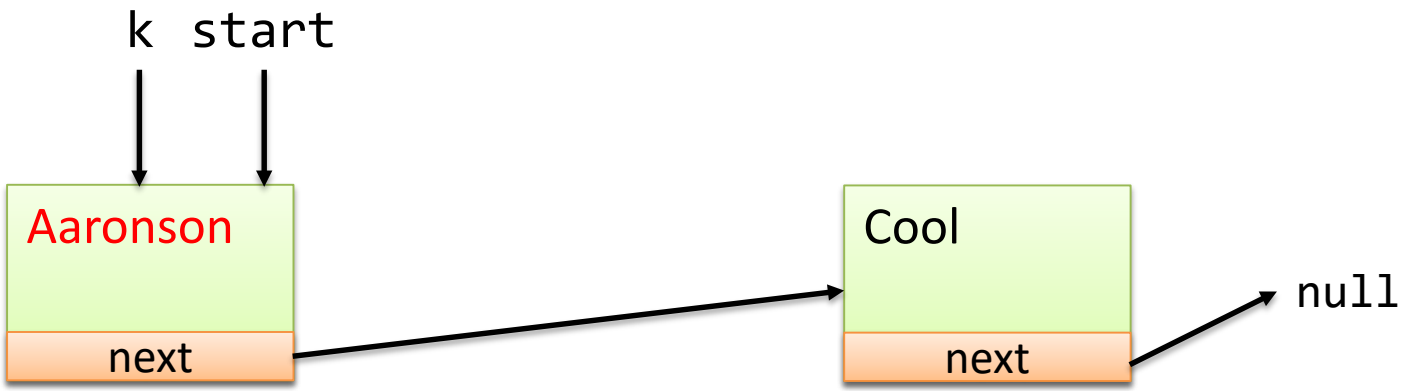
Aaronson



Überprüfung, ob der neue Knoten vor dem Startknoten eingefügt werden muss

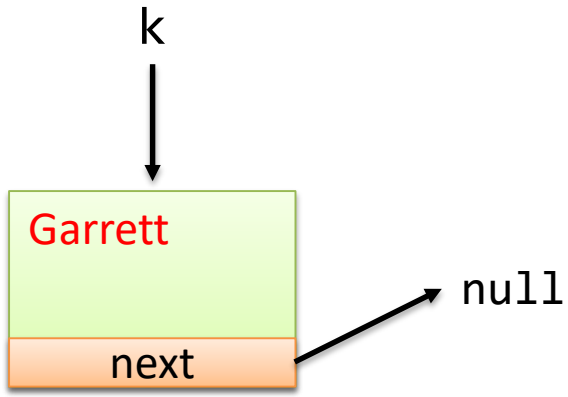
```
Scanner scanner = new Scanner(System.in);
String nachname = scanner.nextLine();
Knoten k = new Knoten(nachname);
…
if (k.getNachname().compareTo(start.getNachname()) < 0)
{
  k.next = start; //k hat den Startknoten als Nachfolger
  start = k;         //k wird neuer Startknoten
}
```

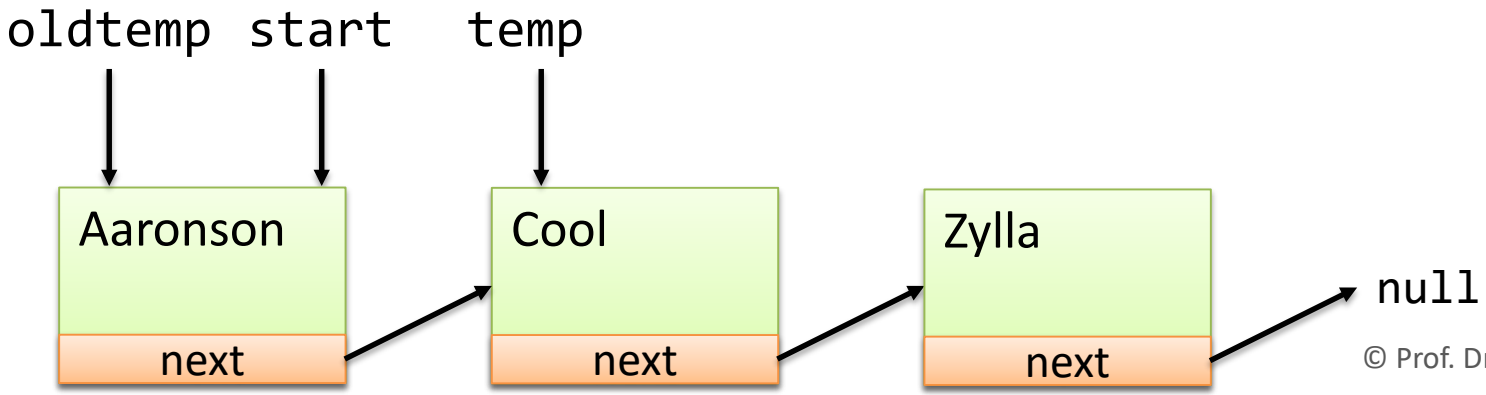
Aaronson



Überprüfung, ob der Knoten mitten in die Liste eingefügt werden muss

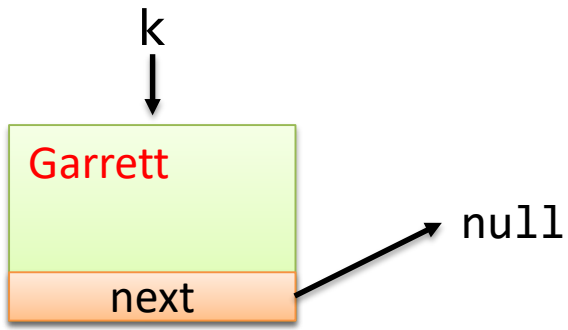
```
Knoten temp = start.next;
Knoten oldtemp = start;
```

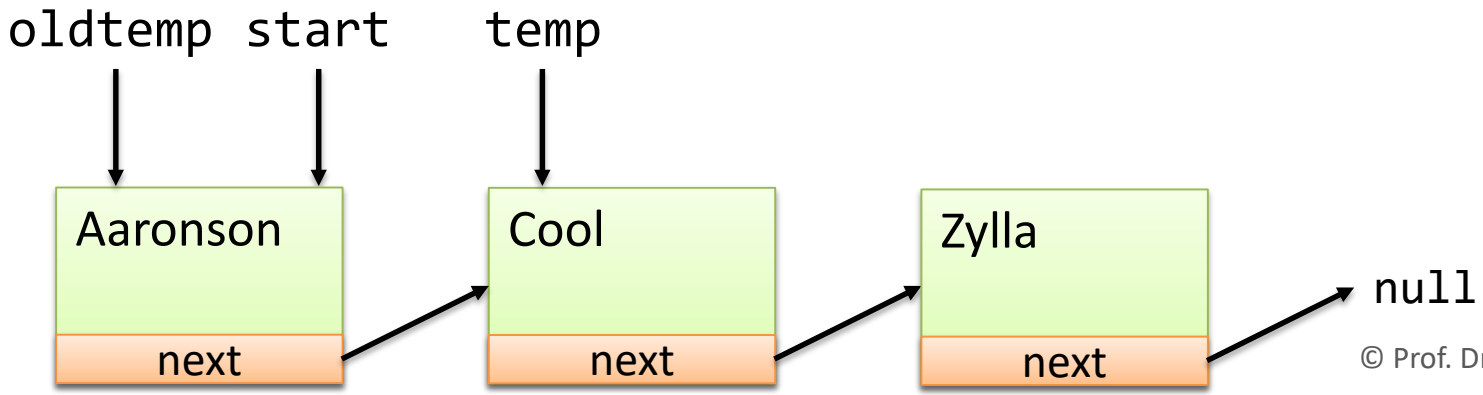




Überprüfung, ob der Knoten mitten in die Liste eingefügt werden muss

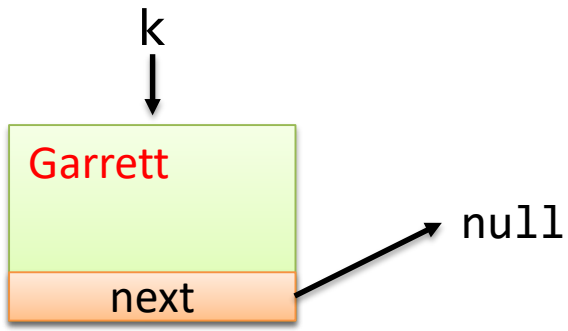
```
Knoten temp = start.next;
Knoten oldtemp = start;
while (temp != null) {
  if (k.getNachname().compareTo(temp.getNachname()) < 0) {


  }
  else {


  }
}
```

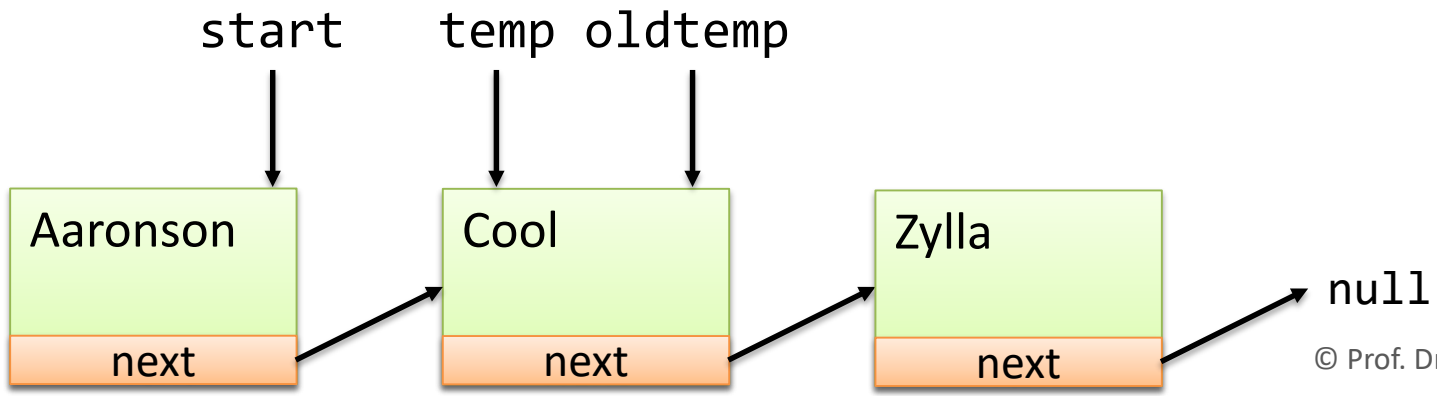




Überprüfung, ob der Knoten mitten in die Liste eingefügt werden muss

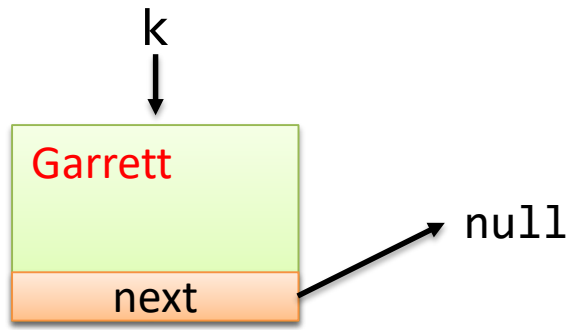
```
Knoten temp = start.next;
Knoten oldtemp = start;
while (temp != null) {
  if (k.getNachname().compareTo(temp.getNachname()) < 0) {



  }
  else {
    oldtemp = temp;

  }
}
```

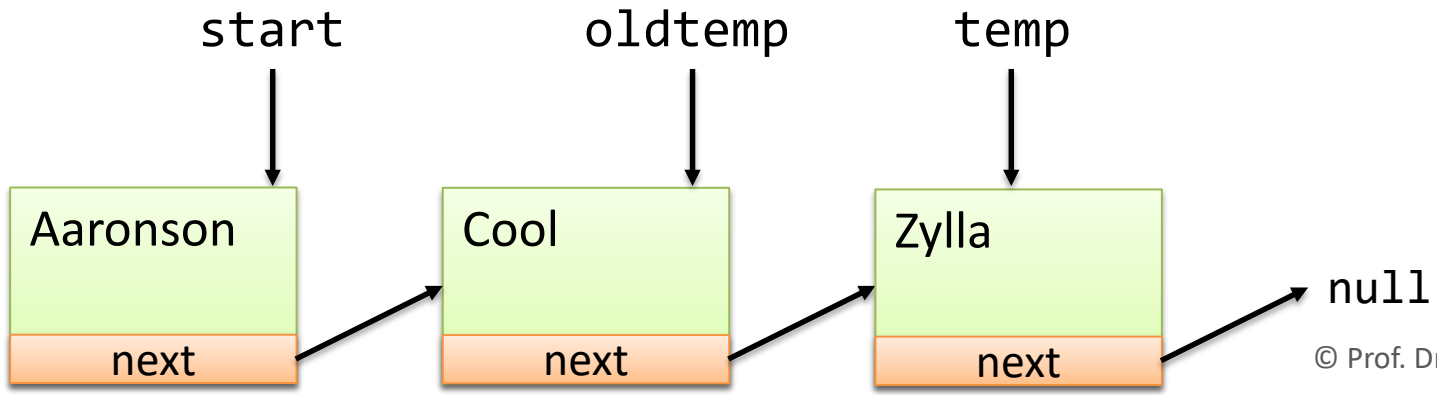




Überprüfung, ob der Knoten mitten in die Liste eingefügt werden muss

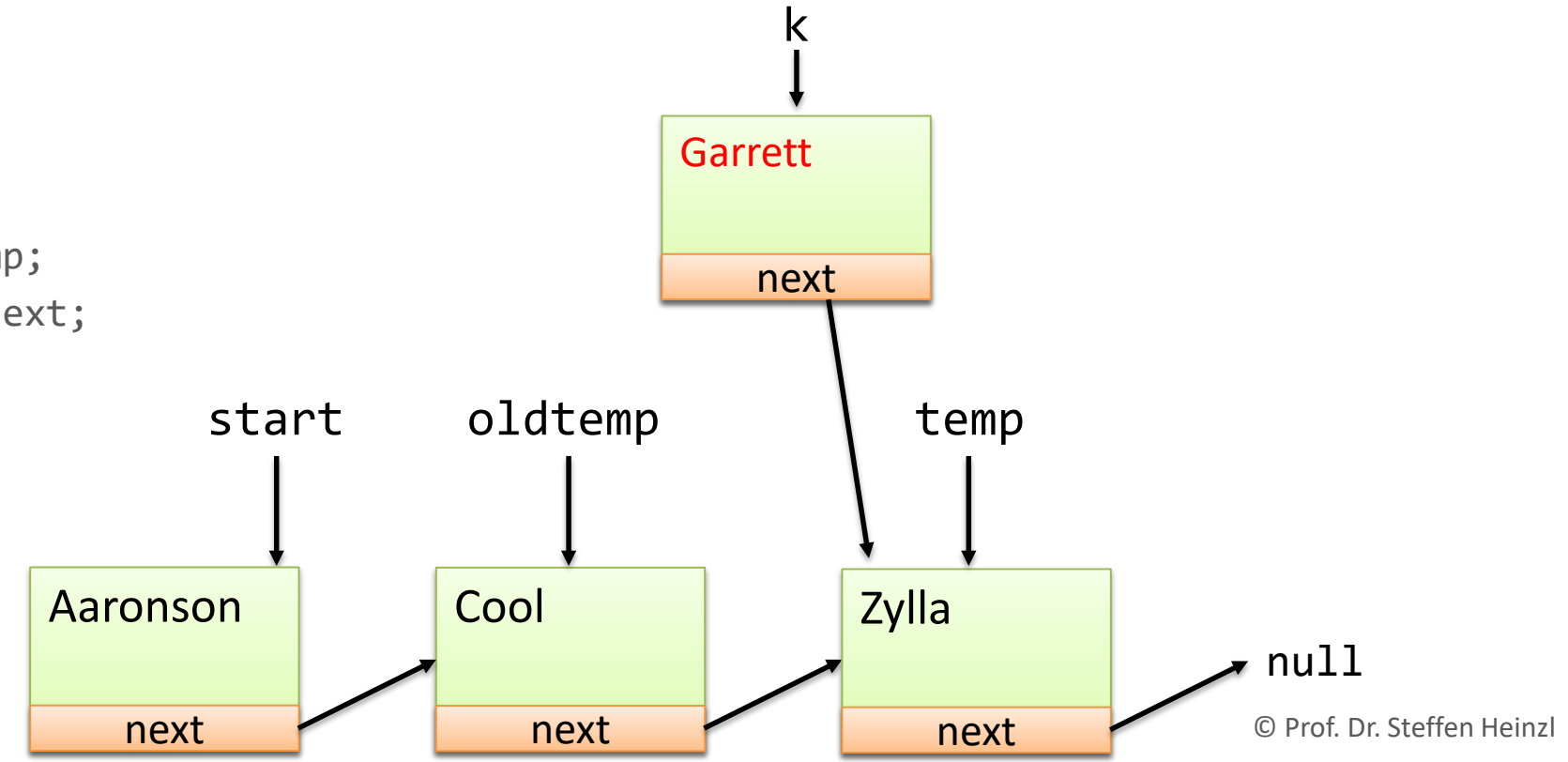
```
Knoten temp = start.next;
Knoten oldtemp = start;
while (temp != null) {
  if (k.getNachname().compareTo(temp.getNachname()) < 0) {



  }
  else {
    oldtemp = temp;
    temp = temp.next;
  }
}
```





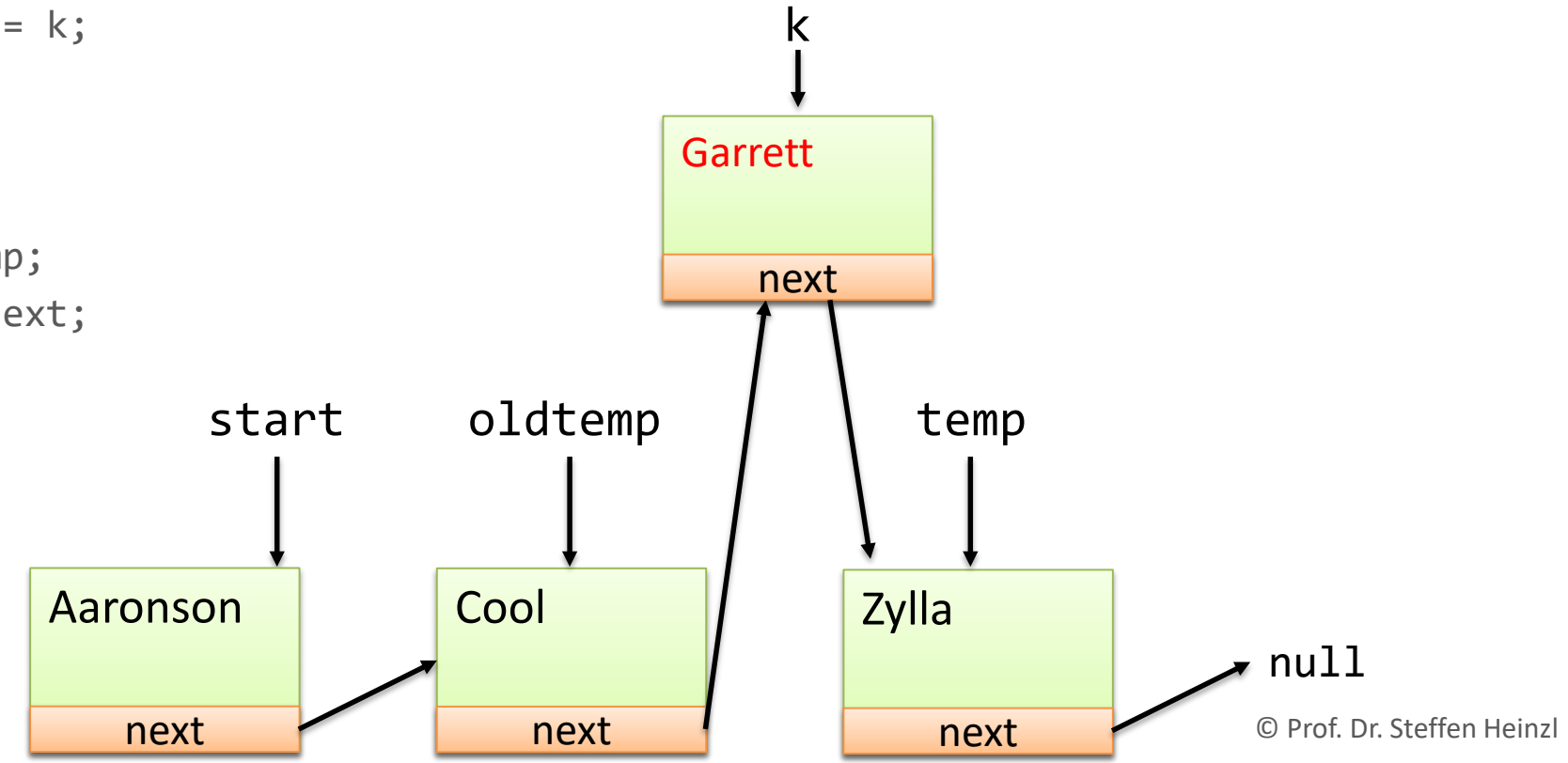
Überprüfung, ob der Knoten mitten in die Liste eingefügt werden muss

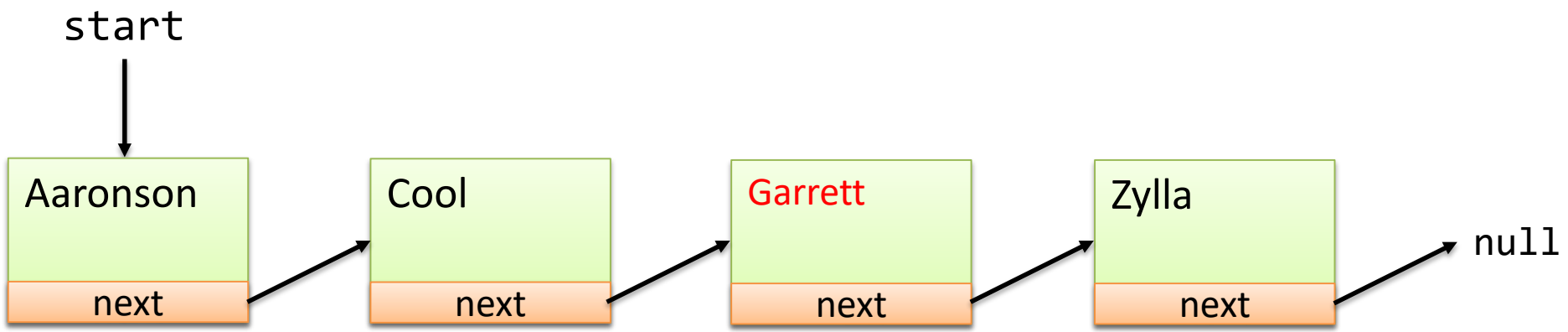
```
Knoten temp = start.next;
Knoten oldtemp = start;
while (temp != null) {
  if (k.getNachname().compareTo(temp.getNachname()) < 0) {
    k.next = temp; //k hat den aktuellen Knoten als Nachfolger



  }
  else {
    oldtemp = temp;
    temp = temp.next;
  }
}
```



Überprüfung, ob der Knoten mitten in die Liste eingefügt werden muss

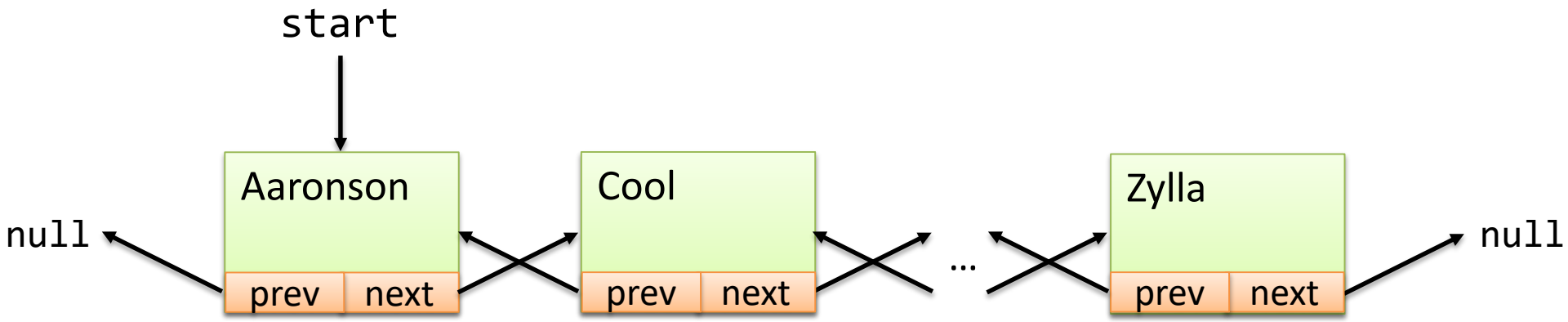
```
Knoten temp = start.next;
Knoten oldtemp = start;
while (temp != null) {
  if (k.getNachname().compareTo(temp.getNachname()) < 0) {
    k.next = temp; //k hat den aktuellen Knoten als Nachfolger
    oldtemp.next = k;
    break;
  }
  else {
    oldtemp = temp;
    temp = temp.next;
  }
}
```





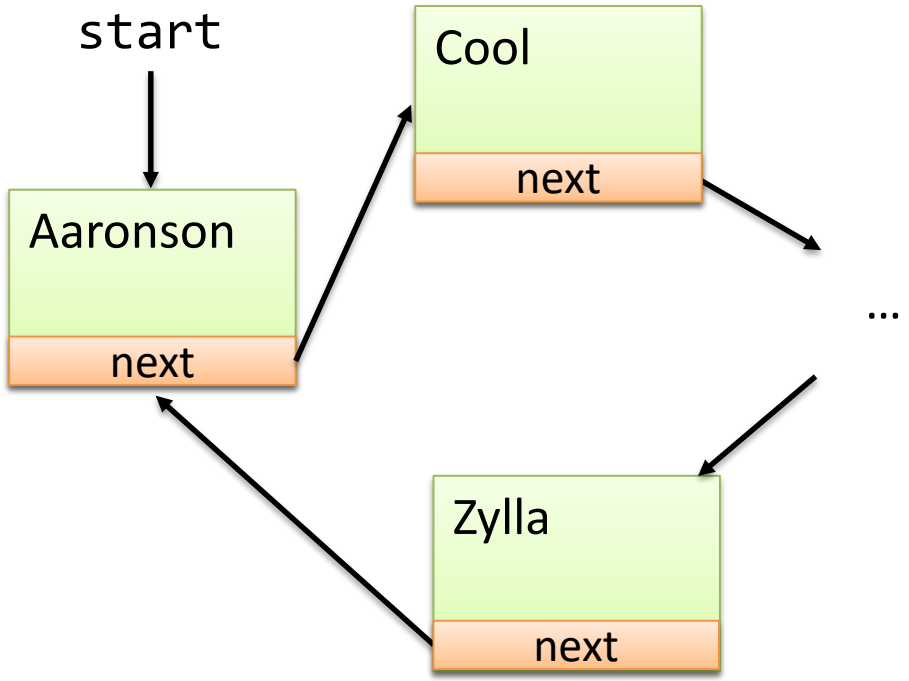
Doppelt verkettete Liste

- Jeder Knoten verfügt über eine Referenz auf seinen Nachfolger und zusätzlich über eine Referenz auf seinen Vorgänger.



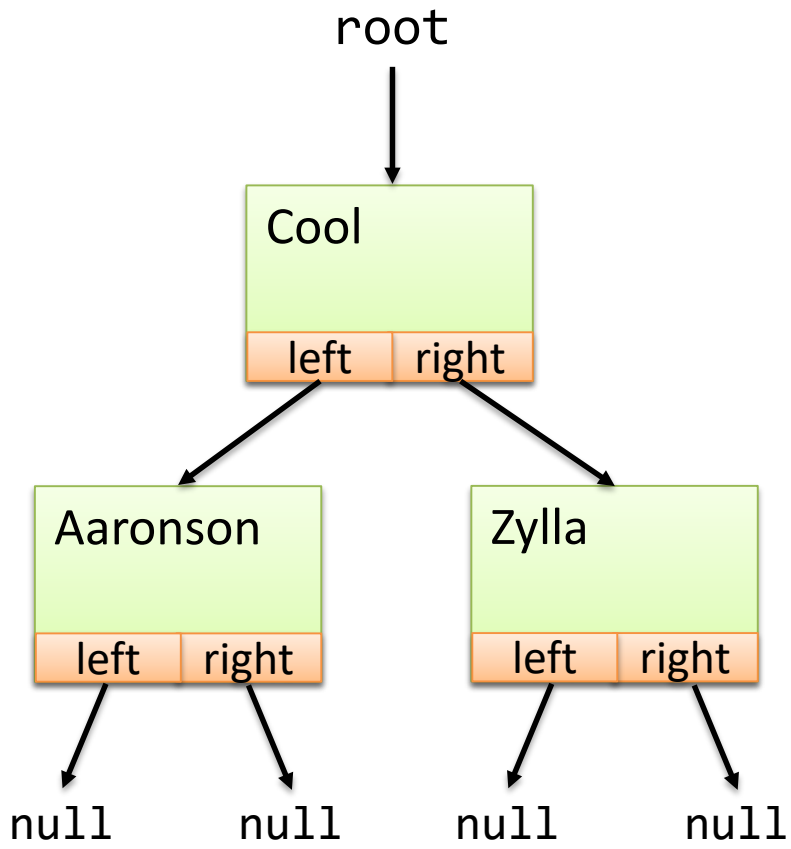
ringverkettete Liste

- Das letzte Element der Liste referenziert den Startknoten.



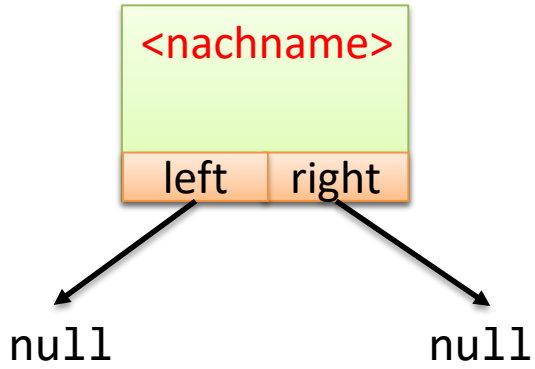
Bäume

- Ein Baum ist ein alternativer Ansatz zur Speicherung von Daten, der i.d.R. eine schnellere Suche ermöglicht.
- Ein Baum hat genau ein Wurzelelement.
- Das Wurzelelement hat Zweige, die wiederum Zweige haben können.
- Wir betrachten **Binär**bäume, d.h. jeder Knoten kann **zweimal** weiterverzweigen.
- Bäume haben i.d.R. eine Sortierung.



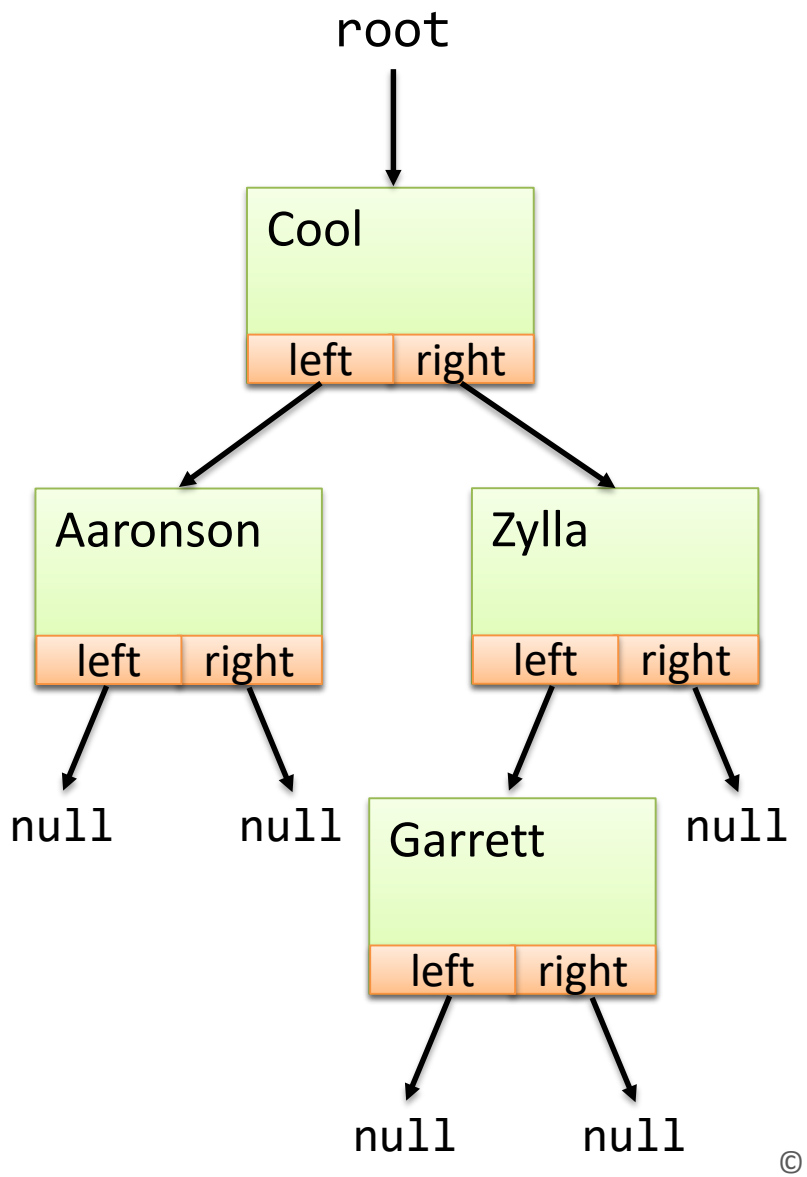
Klasse Knoten

```
public class Knoten
{
  String nachname;
  Knoten left;
  Knoten right;

  public Knoten(String nachname)
  {
    this.nachname = nachname;
  }
  //getter und setter
}
```



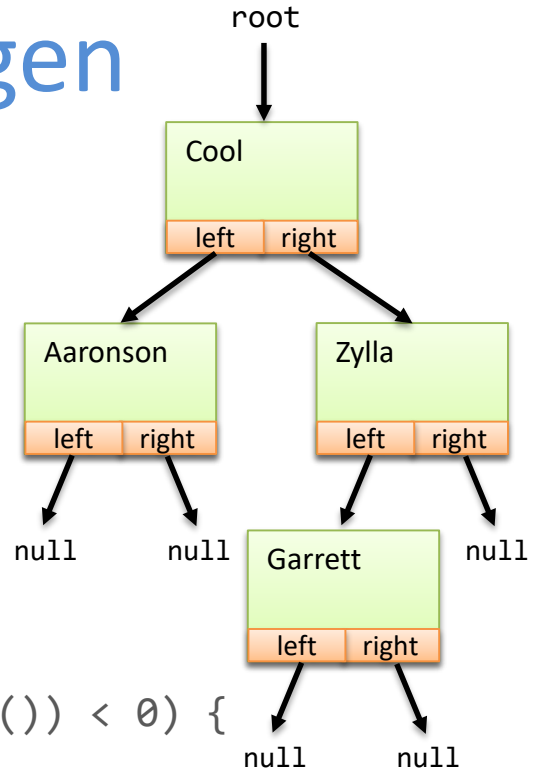
Baum aufbauen

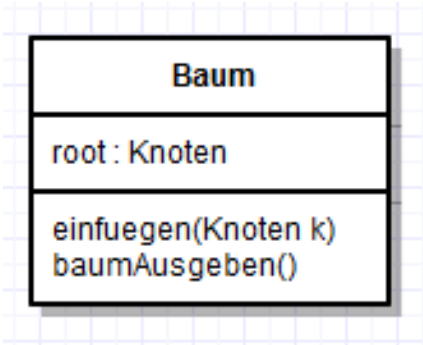
```
public class Main
{
  public static void main(String[] args)
  {
    Baum baum = new Baum();
    Knoten k;
    k = new Knoten("Cool");
    baum.einfuegen(k);
    k = new Knoten("Zylla");
    baum.einfuegen(k);
    k = new Knoten("Aaronson");
    baum.einfuegen(k);
    k = new Knoten("Garrett");
    baum.einfuegen(k);
  }
}
```



Klasse Baum - Einfügen

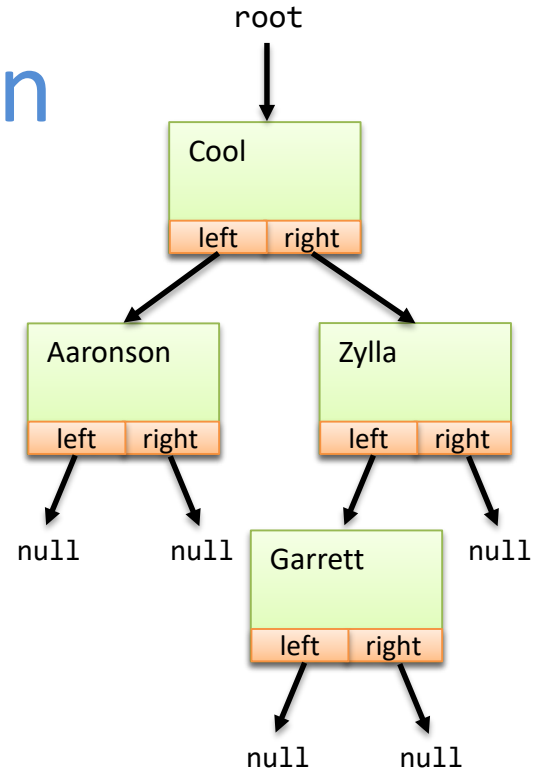
```
public class Baum {
  Knoten root;

  public void einfuegen(Knoten neu) {
    if (root == null) root = neu;
    else einfuegen(root, neu);
  }

  private void einfuegen(Knoten temp, Knoten neu) {
    if (neu.getNachname().compareTo(temp.getNachname()) < 0) {
      if (temp.left == null) temp.left = neu;
      else einfuegen(temp.left, neu);
    }
    else {
      if (temp.right == null) temp.right = neu;
      else einfuegen(temp.right, neu);
    }
  }
  …
}
```





Klasse Baum – Traversieren

```
public class Baum
{
  …
  public void baumAusgeben()
  {
    baumAusgeben(root);
  }

  private void baumAusgeben(Knoten temp)
  {
    if (temp.left != null) baumAusgeben(temp.left);
    System.out.println(temp.getNachname());
    if (temp.right != null) baumAusgeben(temp.right);
  }
}
```



„in order“-Ausgabe

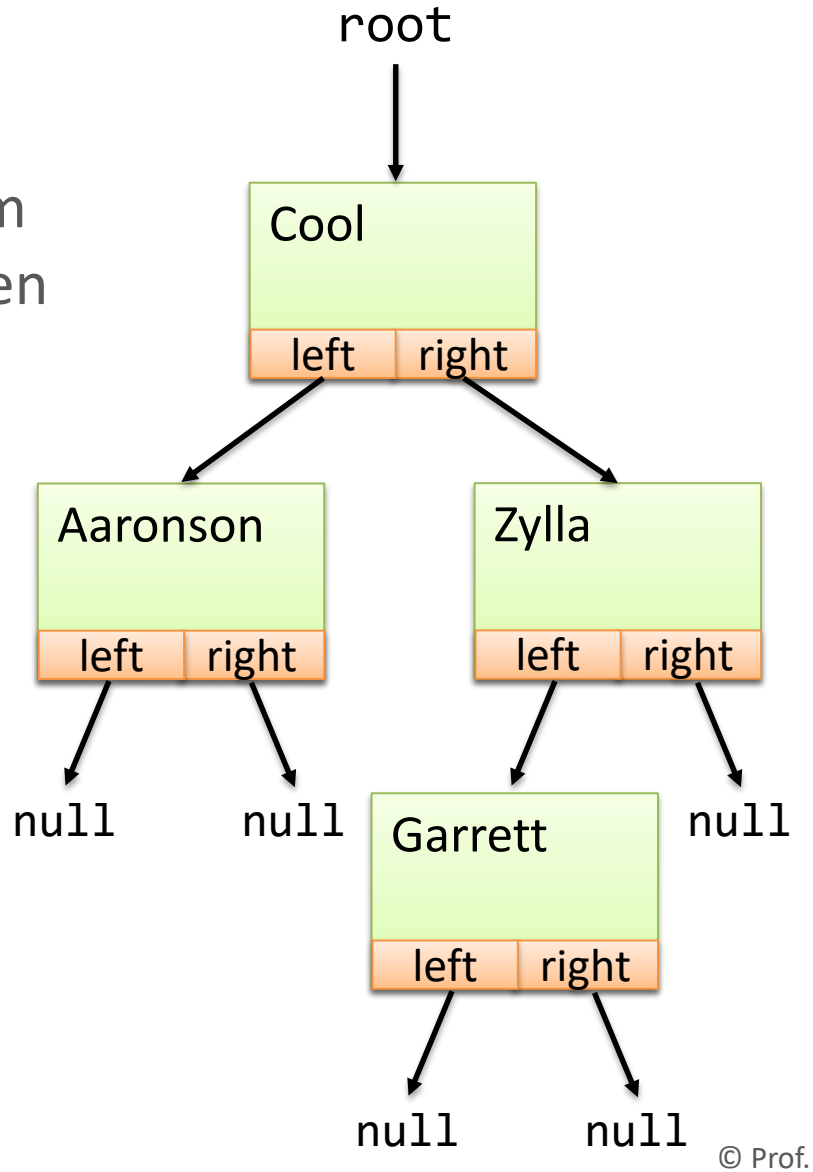
Klasse Baum - Suchen

- Wie kann man einen bestimmten Nachnamen im Baum suchen?

Klasse Baum – Knoten löschen

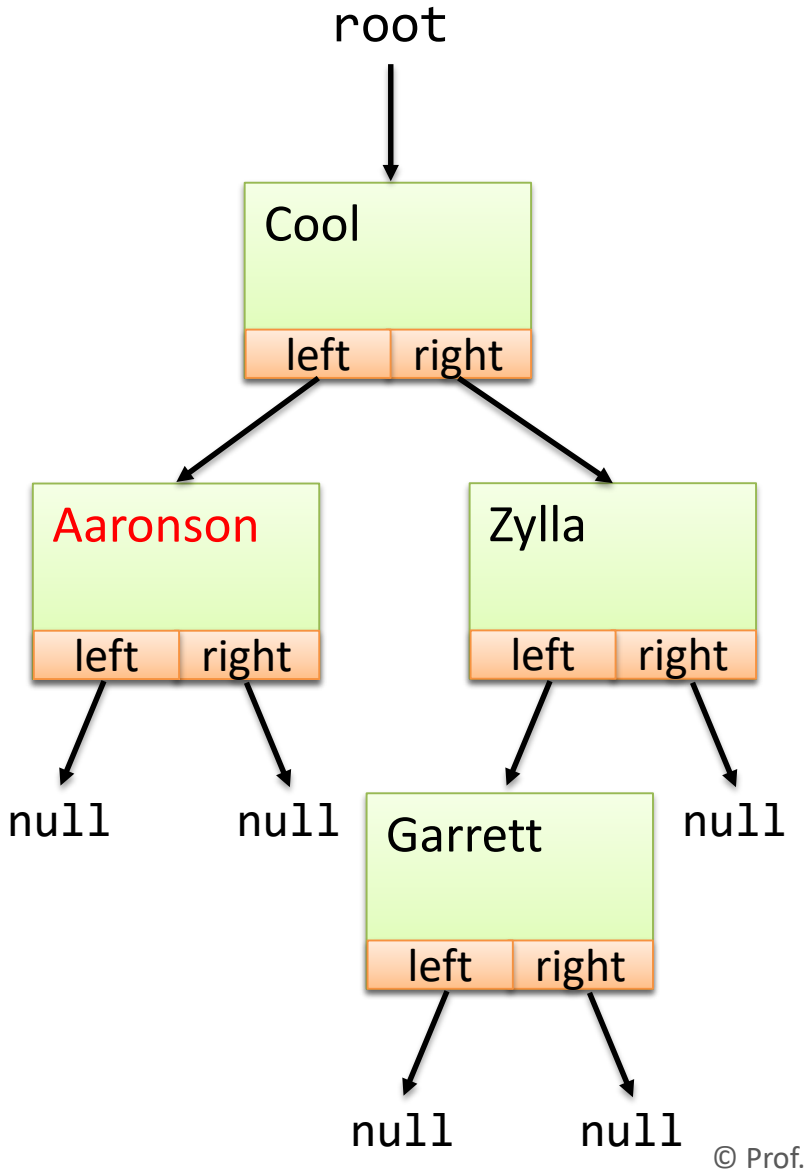
Wesentliche Schritte:

- Rekursiv den Baum durchlaufen, um den zu löschenden Knoten zu suchen
- Vorgänger merken, um den gefundenen Knoten ausgliedern zu können



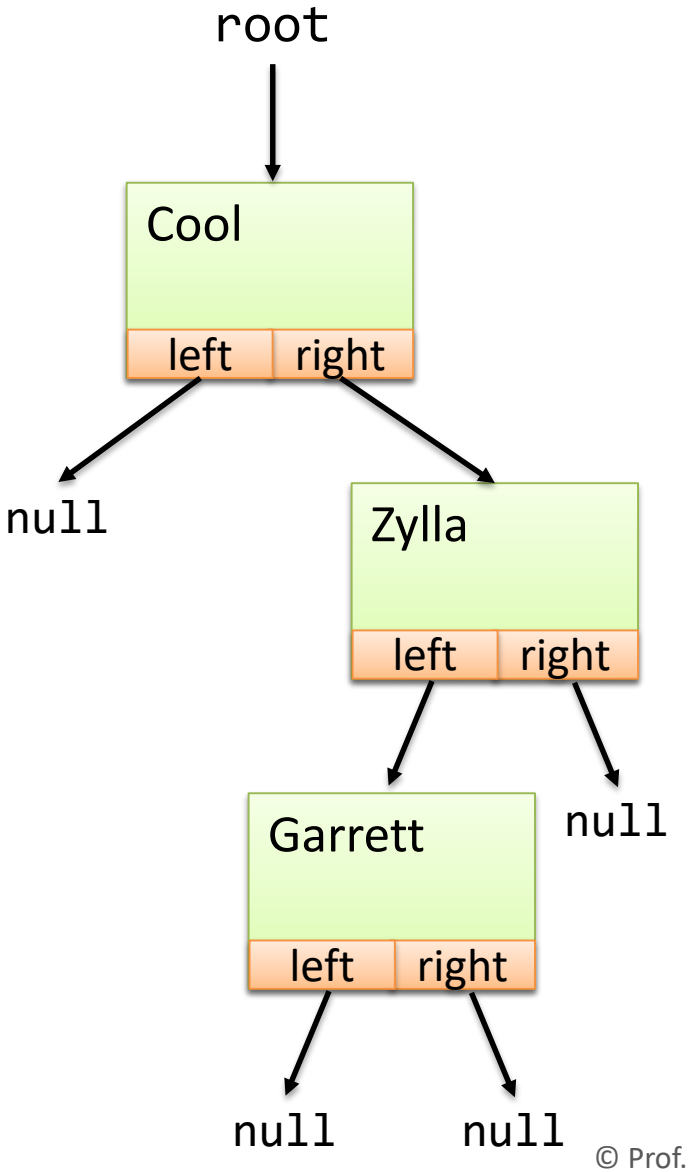
Klasse Baum – Knoten löschen

- Knoten „Aaronson“ soll gelöscht werden
- Kein Problem, da er keine weiteren Knoten an sich hängen hat.



Klasse Baum – Knoten löschen

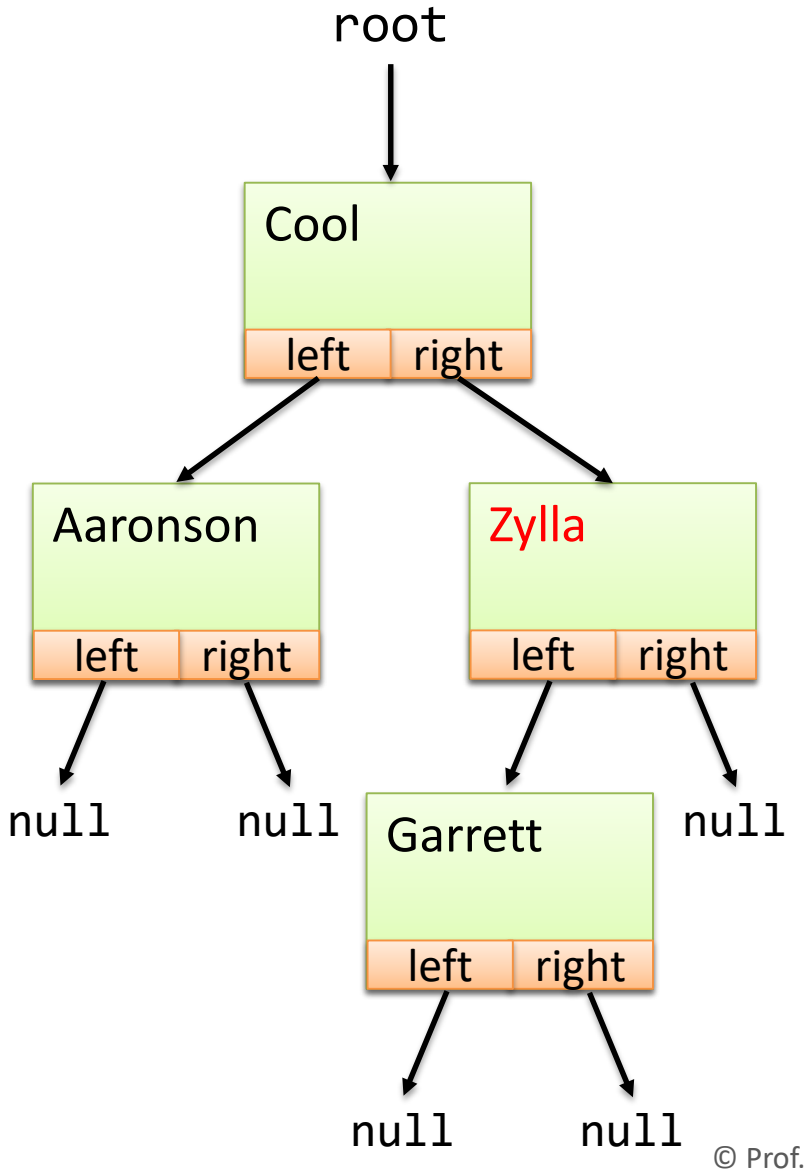
- Knoten „Aaronson“ soll gelöscht werden
- Kein Problem, da er keine weiteren Knoten an sich hängen hat.



Klasse Baum – Knoten löschen

- Knoten k „Zylla“ soll gelöscht werden
- Knoten v “Cool” ist der Vorgänger.

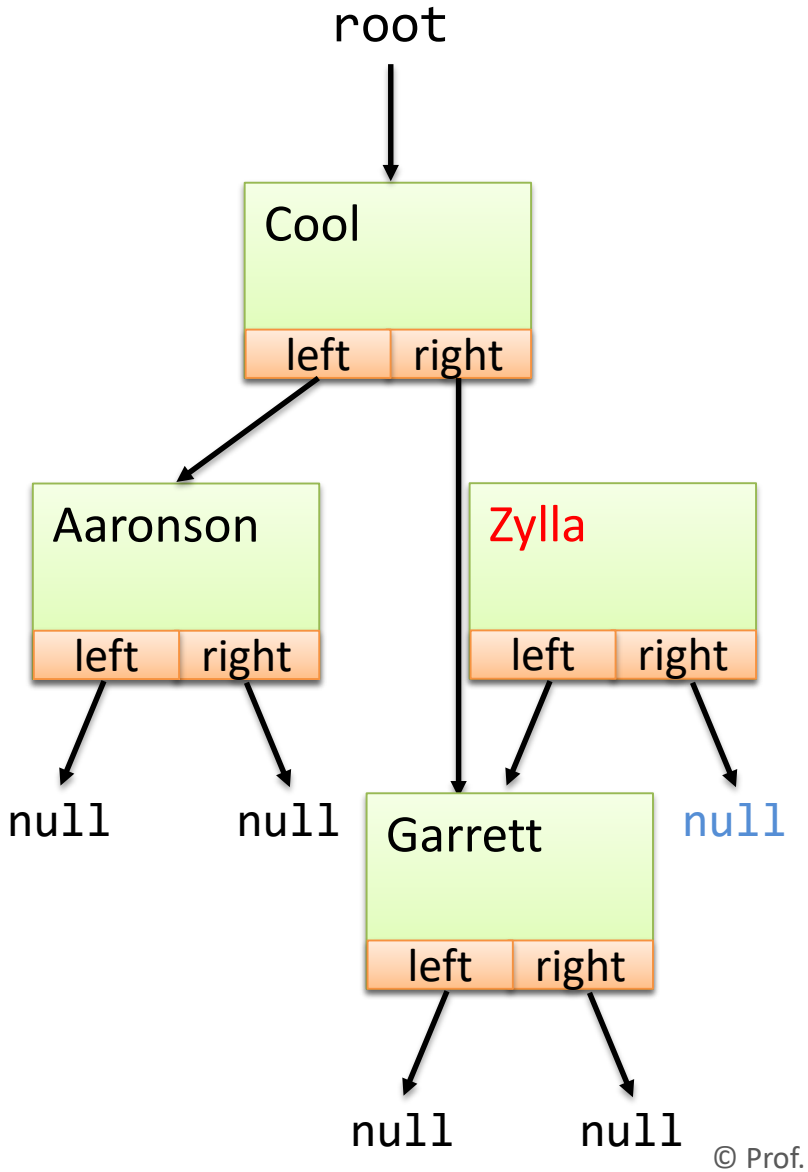
- Generelle Vorgehensweise:
  - Hängt k rechts oder links an v?



Klasse Baum – Knoten löschen

- Knoten k „Zylla“ soll gelöscht werden
- Knoten v “Cool” ist der Vorgänger.

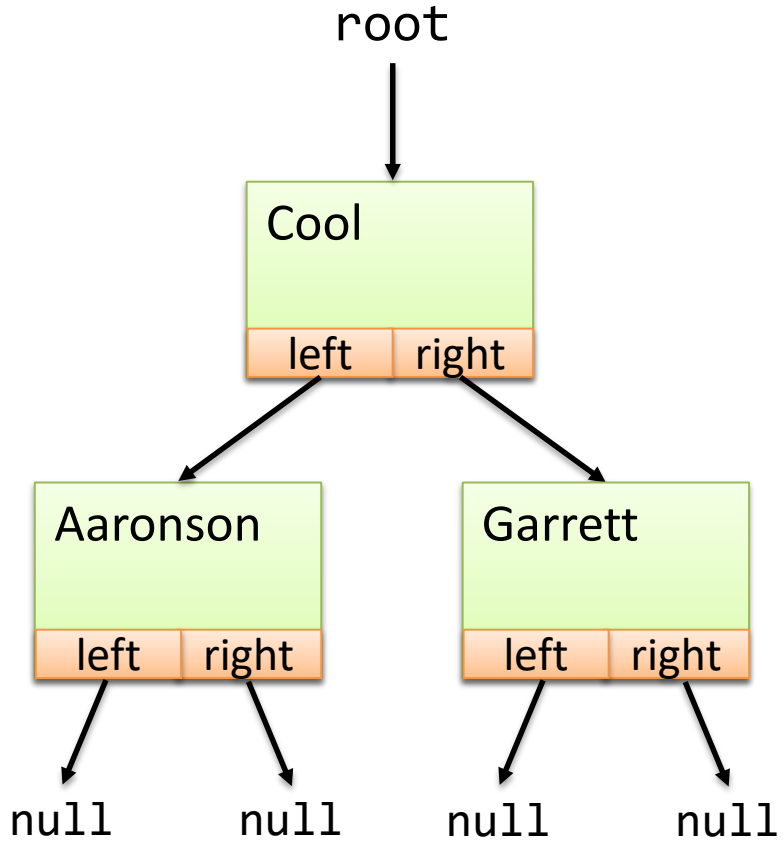
- Generelle Vorgehensweise:
  - Hängt k rechts oder links an v?
  - v.right = k.left
  - k.right wird neu in den Baum eingefügt



Klasse Baum – Knoten löschen

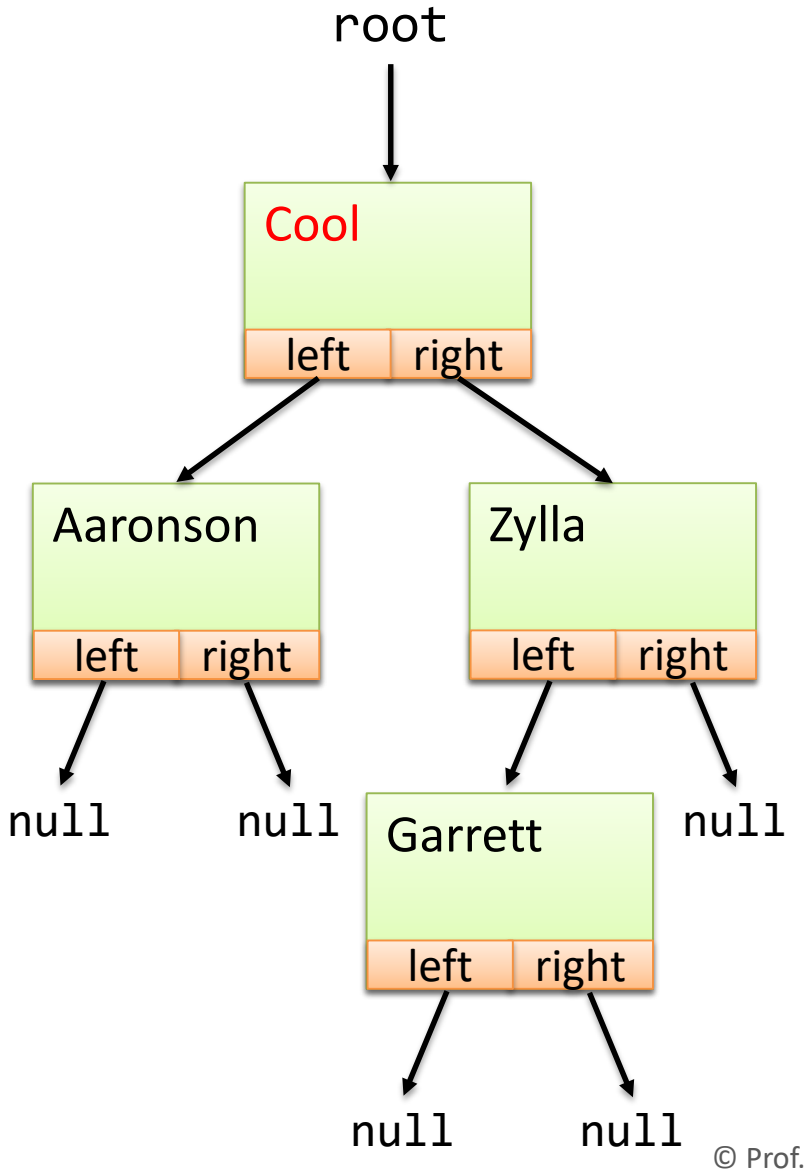
- Knoten k „Zylla“ soll gelöscht werden
- Knoten v “Cool” ist der Vorgänger.

- Generelle Vorgehensweise:
  - Hängt k rechts oder links an v?
  - v.right = k.left
  - k.right wird neu in den Baum eingefügt



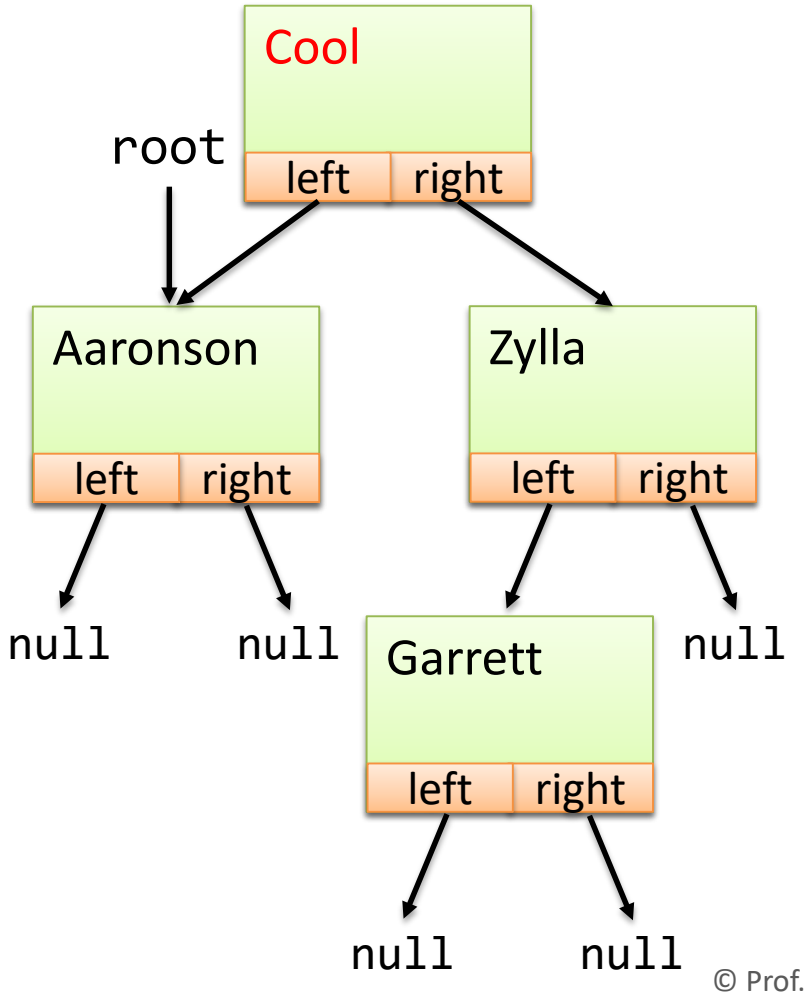
Klasse Baum – Knoten löschen

- Knoten „Cool“ soll gelöscht werden



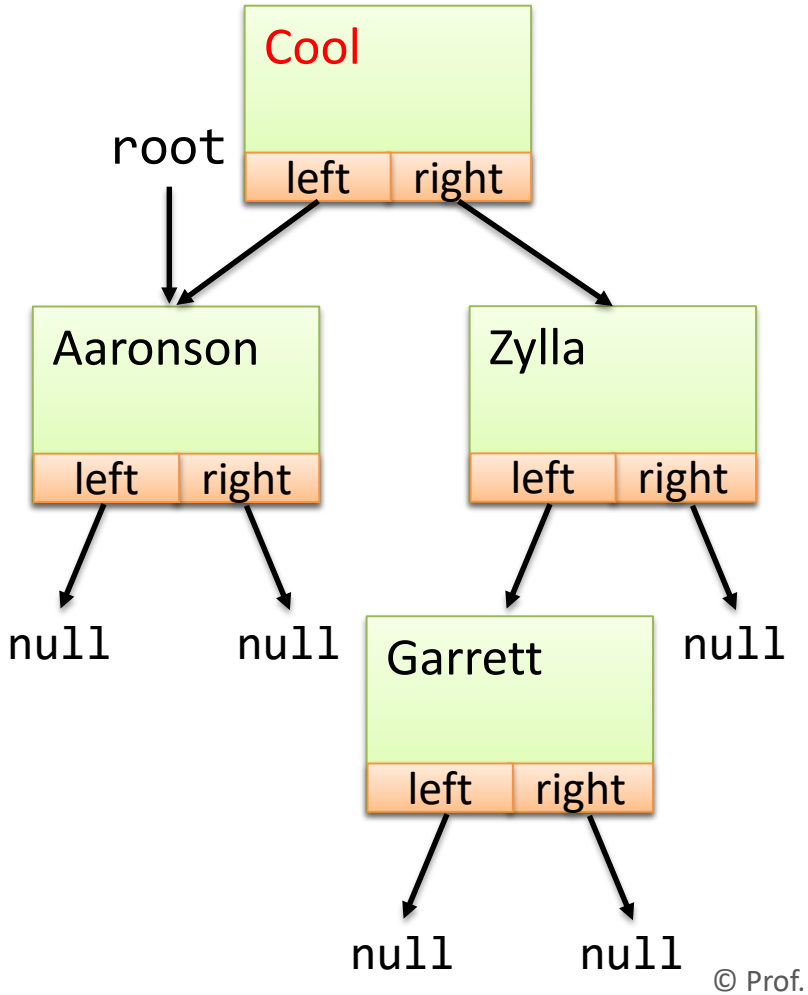
Klasse Baum – Knoten löschen

- Knoten „Cool“ soll gelöscht werden
- „Aaronson“ wird neue Wurzel



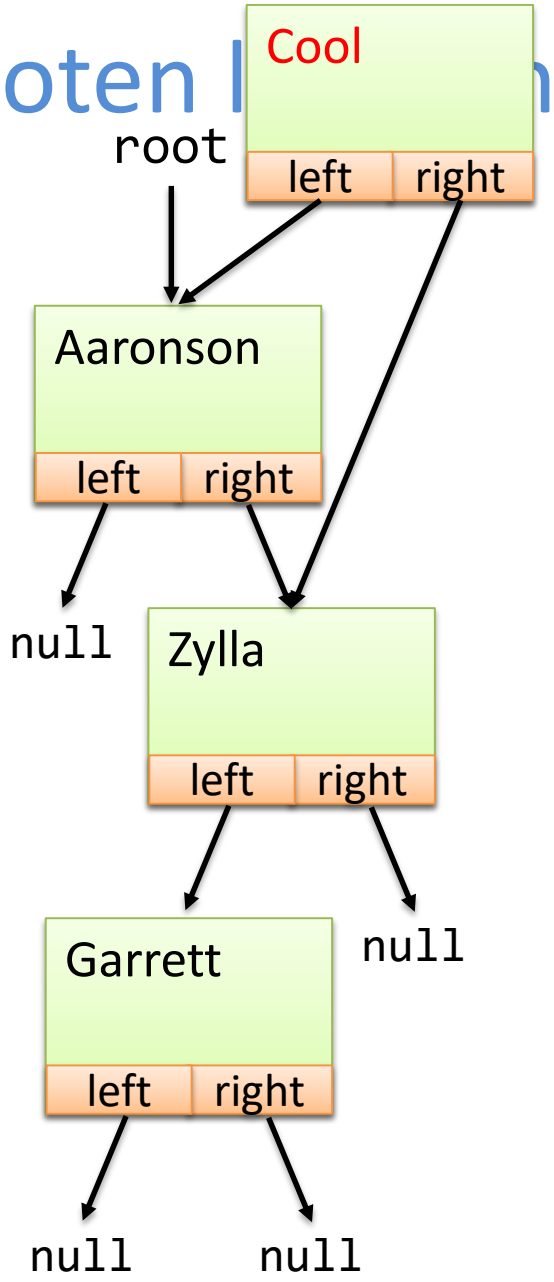
Klasse Baum – Knoten löschen

- Knoten „Cool“ soll gelöscht werden
- „Aaronson“ wird neue Wurzel



Klasse Baum – Knoten l

- Knoten „Cool“ soll gelöscht werden
- „Aaronson“ wird neue Wurzel
- der rechte Zweig von Knoten Cool wird wieder in den Baum eingefügt



Klasse Baum - Suchen

- Wie kann man einen bestimmten Nachnamen im Baum suchen?