Informatik II
Prüfungsvorbereitungskurs

Tag 4, 23.6.2016

Giuseppe Accaputo

g@accaputo.ch

Programm für heute

- Repetition Datenstrukturen
  - Unter anderem Fragen von gestern
- Point-in-Polygon Algorithmus
- Shortest Path Algorithmus
- Datenbanksysteme

Themenübersicht

- 20.3: **Java**
- 21.3: **Objektorientierte Programmierung**
- 22.3: **Dynamische Datenstrukturen**
- **23.3: Datenbanksysteme**


●●○○○ Salt
22:44
66 %
Bearbeiten
Wecker
06:00
Wecker
06:05
Wecker
06:10
Wecker
06:15
Wecker
06:20
Wecker
Weltuhr
Wecker
Stoppuhr
Timer

Repetition:
Datenstrukturen

Übersicht Laufzeiten (worst-case)

- Array
  - **Zugriff auf i-tes Element:** $O(1)$
  - Einfügen: $O(n)$
  - Löschen: $O(n)$
  - Suche:
    - Array sortiert: $O(\log(n))$ (mittels binärer Suche)
    - Sonst: $O(n)$
- Liste
  - Zugriff auf i-tes Element: $O(n)$
  - **Einfügen**: $O(1)$
  - Löschen: $O(n)$
  - Suche: $O(n)$

Übersicht Laufzeiten (worst-case)

- Unbalancierter Baum: degeneriert zu Liste im worst-case Fall
  - Einfügen: $O(n)$
  - Löschen: $O(n)$
  - Suche: $O(n)$

- Balancierter Baum: Hat immer eine Höhe von $O(log_2(n))$, d.h. alle Laufzeiten sind gleich
  - **Einfügen**:$O(log_2(n))$
  - **Löschen**:$O(log_2(n))$
  - **Suche**:$O(log_2(n))$

Übersicht Laufzeiten (worst-case)

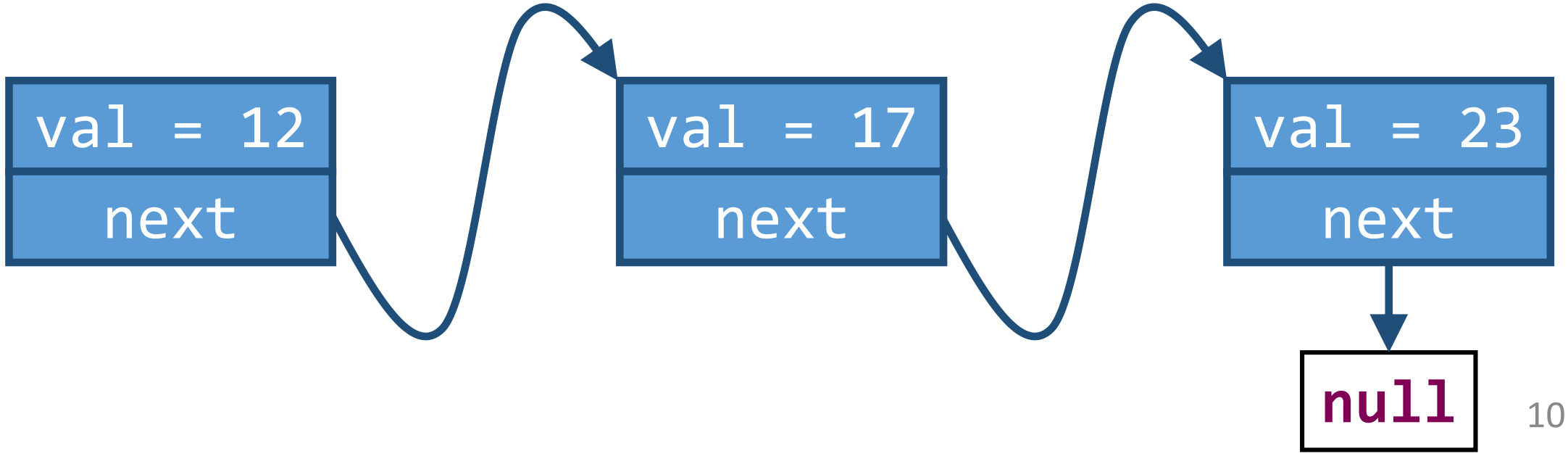
- Min- oder Max-Heap
  - **min / max extrahieren:** $O(log_2(n))$
  - Einfügen:$O(log_2(n))$
  - Löschen:$O(log_2(n))$
  - Suche:$O(log_2(n))$

LIFO vs. FIFO

- LIFO (Stack): Last-In-First-Out
  - Element, das zuletzt hinzugefügt wurde wird als erstes gelöscht
- FIFO (Warteschlange): First-In-First-Out
  - Warteschlange
  - Element, das am längsten in Datenstruktur befindet wird als erstes gelöscht

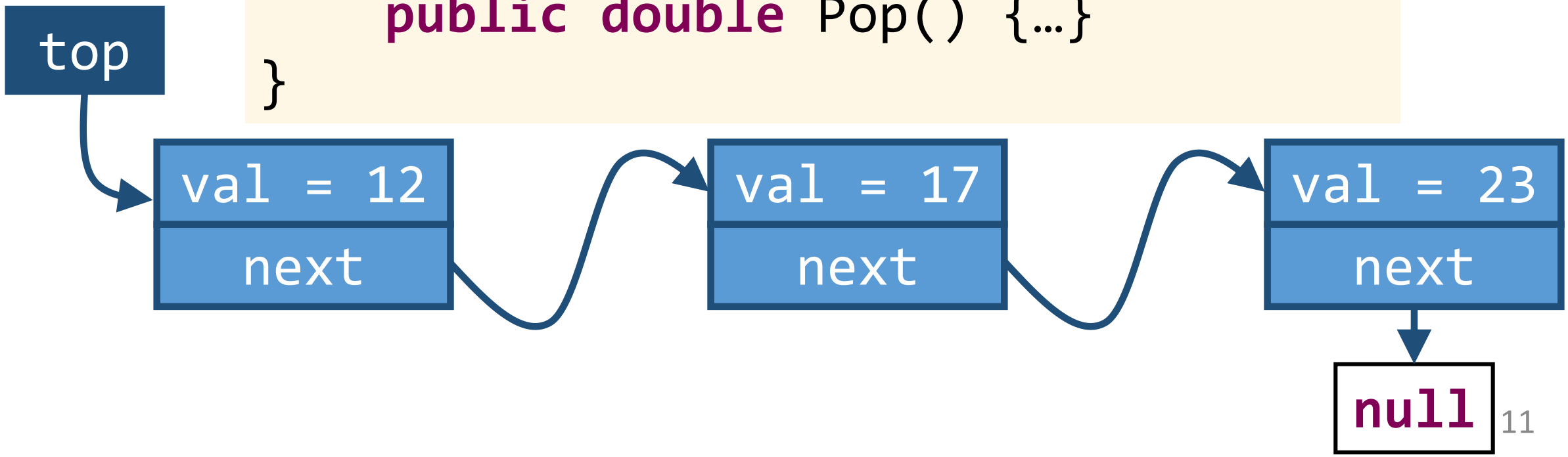
Einfach verkettete Liste: Im Speicher

```
Node n1 = new Node(23, null);
Node n2 = new Node(17, n1);
Node n3 = new Node (12, n2);
```



Stapel (Stack) mittels verketteter Liste

```
public class Stack {
    private Node top = null;
    public void Push(double val) {…}
    public double Pop() {…}
}
```



Prüfung 08.2014 Aufgaben 8a & 8b

- FIFO und LIFO
- Whiteboard

Aufbau einer Hashtabelle

1. Array für Schlüssel und Daten
2. Hashfunktion zur Berechnung des Array-Index (Position) der einzufügenden Objekten
3. Datenzugriff auf Array
   - Bei Kollisionen verwende lineares Sondieren

Beispiel Telefonbuch

```
class Telefonbuch{
   String [ ] keys = new int [10];
   int [ ] values = new int [10];
   int n = 0;

   int hash (String name) {
      return name.charAt(0) % 10;
   }
   … // insert, probe, etc.
}
```

Beispiel Telefonbuch

```
insert("Giu", 112523);
insert("Peter", 051823);
```

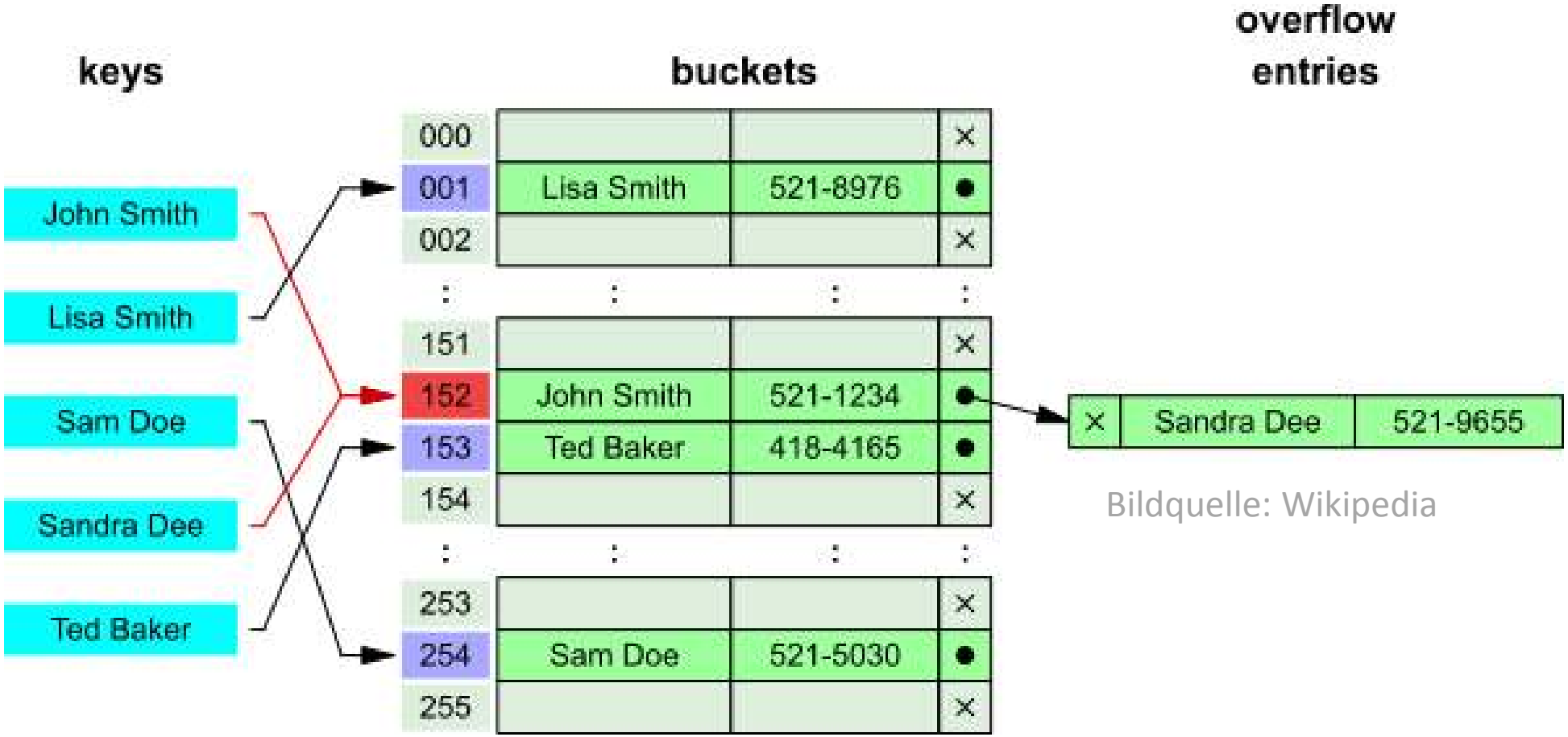
	keys	values
0	Peter	051823
1	Giu	112523
…	…	…

Beispiel Hashtabelle

- Prüfung 01.2015 Aufgabe 8

Bonus: Verkettung mittels Überläufer

- Kollision beim Einfügen: Verkettung mittels Überläufer



Bildquelle: Wikipedia

Bonus: Lineares Sondieren

- Wenn lineares Sondieren verwendet wird, um Kollisionen zu vermeiden, dann muss Suche nach Element – nebst Einfügen – angepasst werden
- Beginne Suche bei Position `hash(key)` und Sondiere linear bis der Eintrag mit `key` gefunden wurde
- Quelle:
  «Algorithmen und Datenstrukturen» (Ottmann & Widmayer)

Bonus: Binärbaum-Höhe

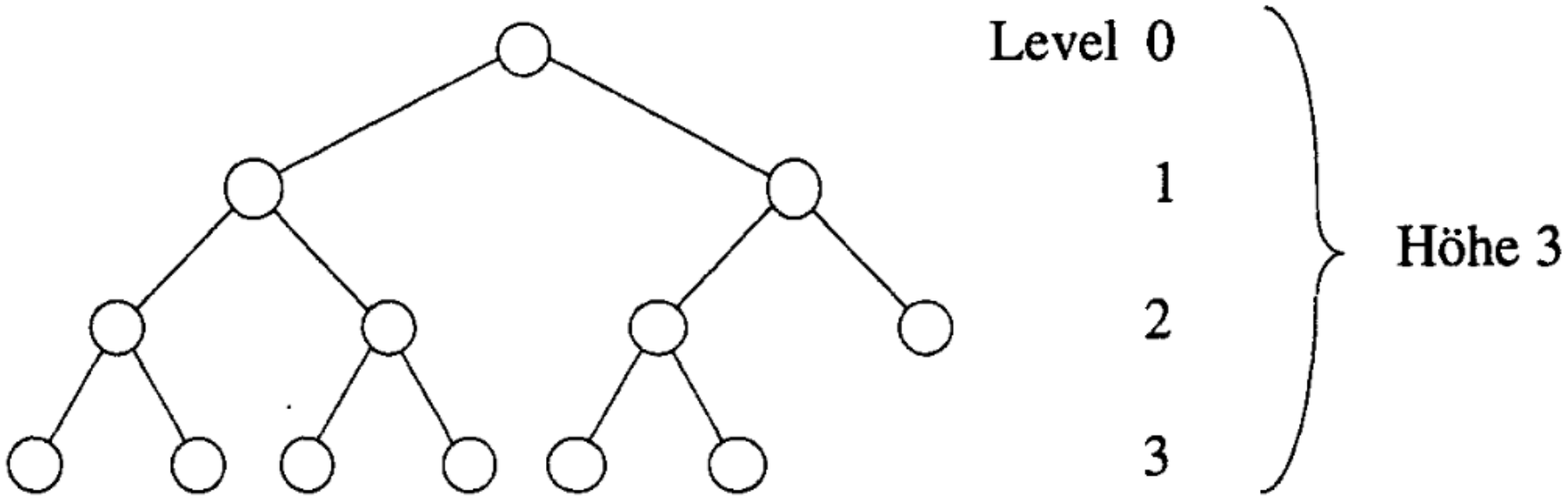
- Höhe ist definiert als die max. Weglänge zwischen Wurzel und Blattknoten
- Minimale Höhe eines Binärbaums mit n Knoten ist $O(log_2(n))$ (exakt: $\lfloor log_2(n) \rfloor$)
  - Höhe: Anzahl Level – 1, wobei Level 0 die Wurzel ist
- Quelle:
  «Datenstrukturen und Algorithmen» (Güting & Dieker)

Laufzeit von Operationen auf Binärbäumen

- Im Mittel benötigt Einfügen / Löschen / Suchen in einem Binärbaum $O(log_2(n))$
  - **Achtung**: nur im Mittel! Worst-case Verhalten ist trotzdem $O(n)$ bei nicht-balancierten Binärbäumen
- $O(log_2(n))$ weil bei jeder Abzweigung von Wurzel bis zu Blattknoten jeder Teilbaum halbiert wird
  - $log_2(n)$ gibt an wie oft n halbiert werden kann

Balancierte Bäume

- Garantiere, dass Binärbaum mit n Knoten stets eine Höhe von $O(log_2(n))$



Bildquelle: «Datenstrukturen und Algorithmen» (Güting & Dieker)

Prüfung 01.2015 Aufgabe 5a

- Balancierten Baum zeichnen
- Whiteboard

Prüfung 08.2014 Aufgaben 5a & 5d

- Code um Baum zu traversieren
- Asymptotischer worst-case Speicherplatzverbrauch
- Whiteboard

Algorithmen:
Point-in-Polygon, Shortest Path

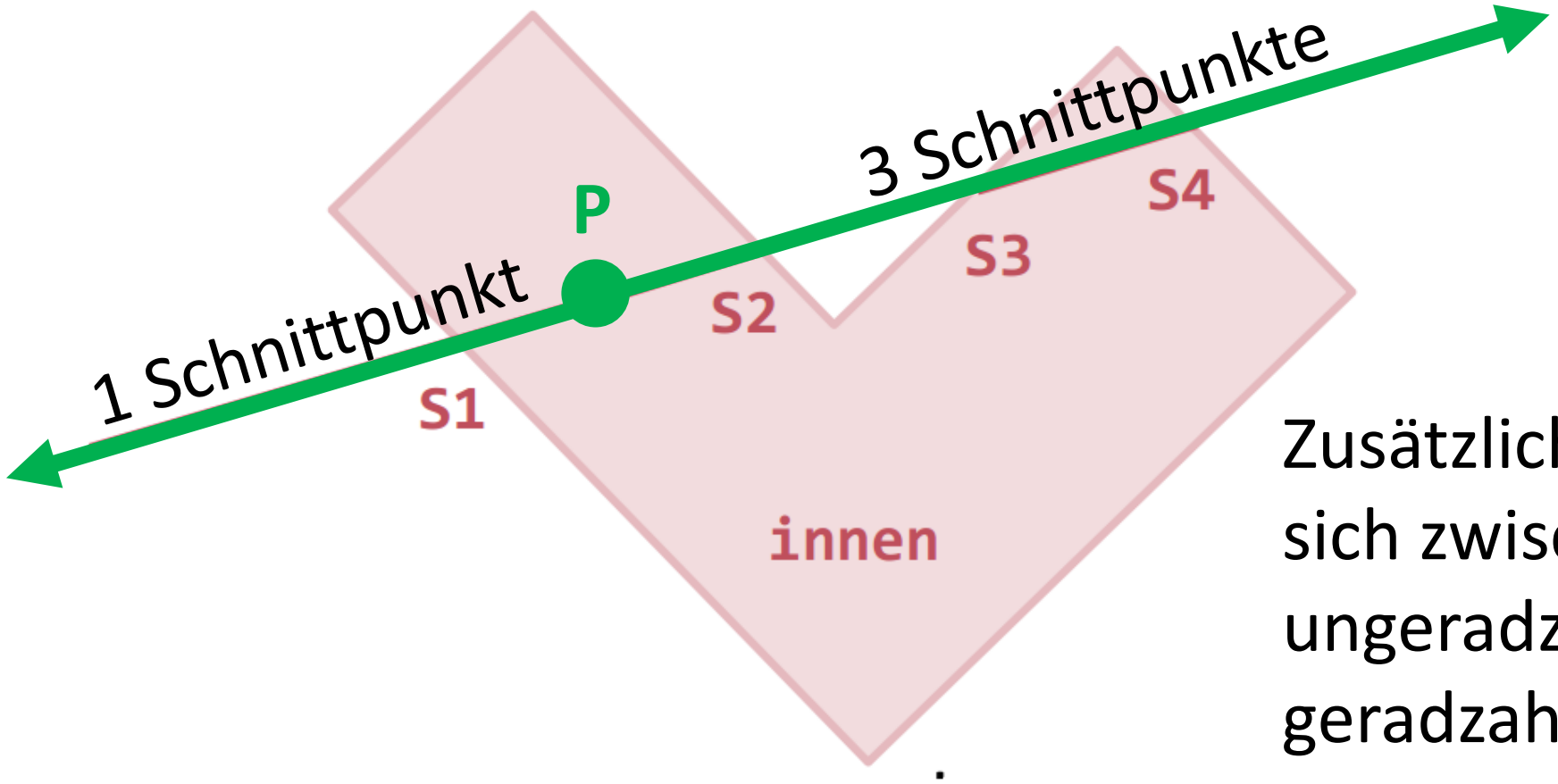
Point-in-Polygon Algorithmus

- Schnittfreie Polygone sind Jordankurven und zerlegen die Ebene in zwei Gebiete: das Innere und das Äussere
- Wir können dies ausnutzen, indem man Polygon mit Geraden schneidet

Abzählmethode

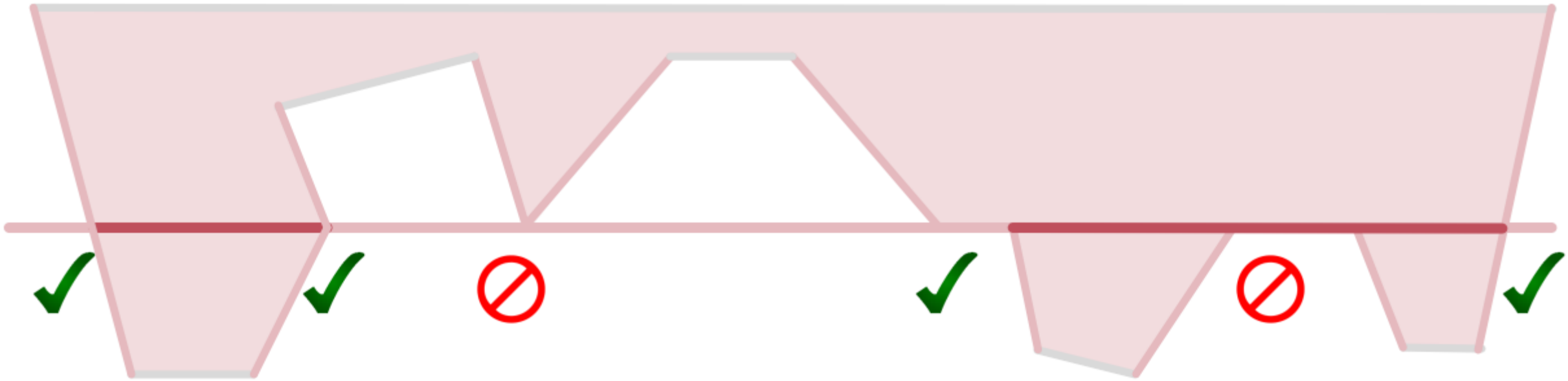
- Wird verwendet um zu überprüfen, ob sich ein Punkt im Polygon befindet
- Punkt liegt im Inneren eines Polygons, wenn von Punkt aus in beliebige Richtung ausgehender Strahl ungeradzahlig viele Polygonkanten kreuzt
  - Bereiche zwischen ungeradzahligen und geradzahligen Schnittpunkten liegen im Inneren des Polygons

Abzählmethode



Zusätzlich: Punkt befindet sich zwischen einem ungeradzahligen und einem geradzahligen Schnittpunkt

Echte und unechte Schnitte



- Echte Schnitte
- Unechte Schnitte

Bildquelle:
Informatik II Vorlesung

Echte Schnitte überprüfen

```
boolean IntersectHorizontalLeft(int x, int y, Vertex a, Vertex b)
{
	if (b.y < y && y <= a.y) { // Kante von oben nach unten
		return (x-a.x) * (b.y - a.y) <= (b.x-a.x)*(y-a.y);
	}
	else if(a.y < y && y <= b.y) { // Kante von unten nach oben
		return (x-a.x) * (b.y - a.y) >= (b.x-a.x)*(y-a.y);
	}
	else
		return false;
}
```

Prüfung 08.2015 Aufgabe 8

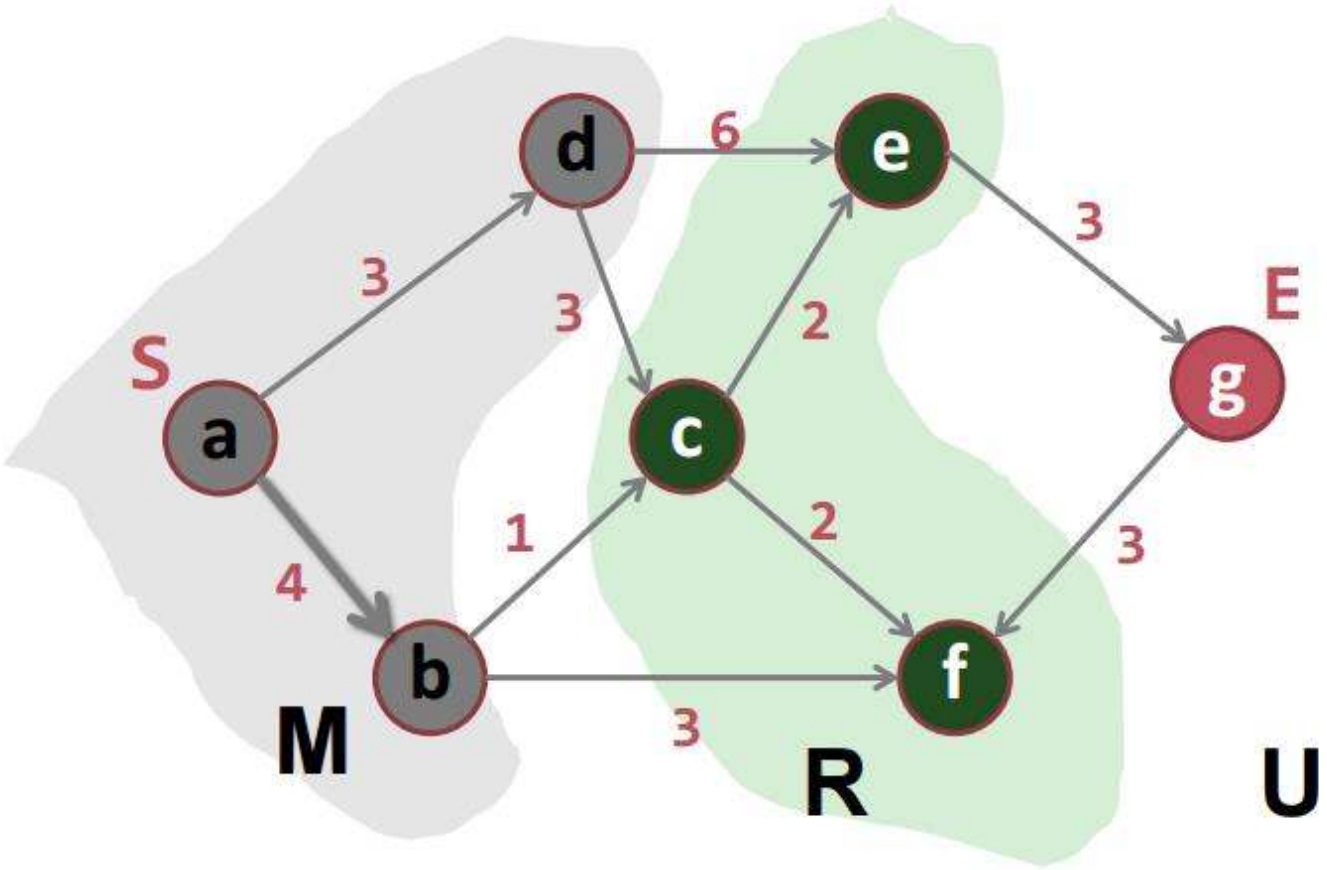
- Schaf vs. Wolf
- Whiteboard

Dijkstra's Shortest Path

- Gegeben: Gerichteter Graph (V, E)mit Knotenmenge V und Kantenmenge E, bei dem jeder Kante eine Länge zugeordnet ist
- Problem: Finde zu Startpunkt $S \in V$ und Endpunkt $K \in V$ den kürzesten Pfad entlang der Kanten im Graph

Dijkstra's Shortest Path: Implementation

- M: Knoten, die Teil eines minimalen Pfades erkannt wurden
- R: Knoten, die von M aus direkt erreichbar sind
  - Min-Heap, da wir aus R Knoten mit kleinster Pfadlänge nehmen
- U: Knoten die noch nicht berücksichtigt wurden



Bildquelle:
Informatik II Vorlesung

Dijkstra’s Shortest Path: Algorithmus

- Beispiel (Whiteboard)

Part 4: **Datenbanksysteme**

Vorbereitungstipps

- In Gruppen lernen kann vorteilhaft sein
  - Bei Fragen ist gleich jemand da
  - Fragen beantworten hilft Verständnis
- Unbedingt alle alten Prüfungen lösen (**Wichtig**)
  - Eine Prüfung verwenden, um Prüfungssituation zu simulieren (auf Zeit)
- Übungen anschauen und versuchen zu verstehen
- Blick in alte Übungen werfen
  - http://lec.inf.ethz.ch/baug/informatik2/2014/
  - http://lec.inf.ethz.ch/baug/informatik2/2015/

Fragen zu Übungen oder alten Prüfungen

- Ich stehe gerne zur Verfügung für Fragen zu den Übungen und alten Prüfungen, falls welche auftauchen sollten während der Vorbereitung:

  g@accaputo.ch

- Meine Slides zu den diesjährigen Informatik II Übungen: http://accaputo.ch/hilfsassistenz/informatik-2-d-baug-2016/

Prüfungstipps

- Zeit pro Punkt ausrechnen: Dauer Prüfung / Anz. Punkte
  - Uhr an Prüfung mitnehmen (z.B. Wecker)
  - Gute Abschätzung um zu sehen, ob man zu viel Zeit mit einer Aufgabe verbringt
- Prüfung zuerst durchblättern und schauen, welche Aufgaben einfach zu lösen sind. **Diese dann gleich als erstes lösen!**