

Kap. 8 Geometrische Algorithmen

Petra Mutzel

Lehrstuhl für Algorithm Engineering, LS11

27. VO (letzte)

13. Juli 2006

Überblick

- Sweepline Algorithmen
 - Beispiel: Schnittproblem für iso-orientierte Liniensegmente
- Mehrdimensionale Bereichssuche

Motivation

„Warum soll ich heute hier bleiben?“

mal was ganz anderes

„Was gibt es heute Besonderes?“

letzte Vorlesung 😊

Geometrische Algorithmen

- Gegeben: Daten, die Koordinaten im k -dimensionalen Raum besitzen: **geometrische Daten**
- Die **algorithmische Geometrie** (engl. *Computational Geometry*) nützt geometrische Eigenschaften der Daten aus.

- **Anwendungen:** Geographie, GIS-Systeme, Chip-Layout, Bildverarbeitung, CAD, Computergraphik, Computerspiele, Robotik, Sensornetzwerke,...

Geometrische Fragestellungen

- Wir betrachten die folgenden Fragestellungen:
 - Schnittberechnung von Segmenten in der Ebene
 - Schnittberechnung von Rechtecken in der Ebene
 - Mehrdimensionale Bereichssuche

Kap. 8.1 Sweepline Algorithmen

- Anwendbar für viele Probleme P in der Ebene, d.h. wenn die Punkte im 2-dim Raum liegen
- Problem: R^2 ist nicht diskret
- **Sweepline** Verfahren: Technik um Probleme zu diskretisieren

- **Idee:** eine **Sweepline (Scanline)** „wandert“ (parallel verschoben) über die Ebene.
- statt P auf R^2 zu lösen, betrachten wir nur Ereignisse, die auf der Sweepline auftreten:
- ein 2-dim. Problem wird also in eine Folge von 1-dim. Problemen verwandelt.

Schnittproblem für Liniensegmente

- **Gegeben:** Menge $S = \{s_1, \dots, s_n\}$ von Liniensegmenten im 2-dim. Raum
- **Gesucht:** alle Paare sich schneidender Segmente

Schnittproblem für Liniensegmente

Naiver Ansatz:

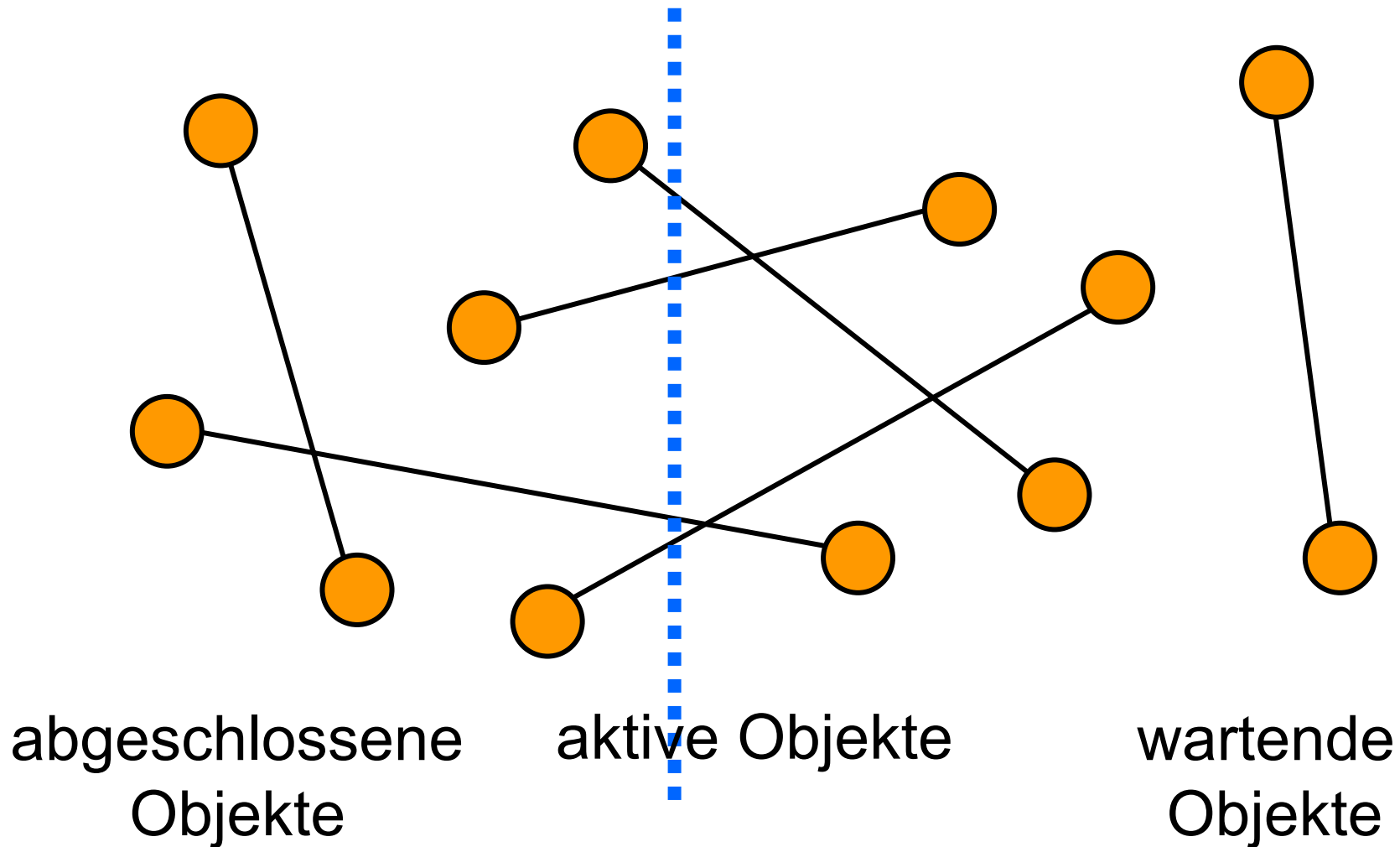
- Für alle Segmentpaare:
 - berechne eventuelle Schnittpunkte der Geradengleichungen
 - prüfe, ob diese Schnittpunkte Teil der Segmente sind

- Laufzeit: $\Theta(n^2)$

- besser: mit Sweepline-Verfahren

Schnittproblem für Liniensegmente

Sweepeline hält an Ereignispunkten:



Sweepline Algorithmen allgemein

- Eine vertikale Sweepline L wird von links nach rechts über die betrachtete Ebene geführt.
- Unabhängig von der Problemstellung teilt sie die Objekte zu jedem Zeitpunkt in 3 Klassen ein:
 - **Abgeschlossene Objekte:** Objekte, die vollständig links von L liegen → also bereits behandelt wurden
 - **Aktive Objekte:** werden momentan von L geschnitten
 - **Wartende Objekte:** Objekte, die vollständig rechts von L liegen → wurden noch nicht entdeckt

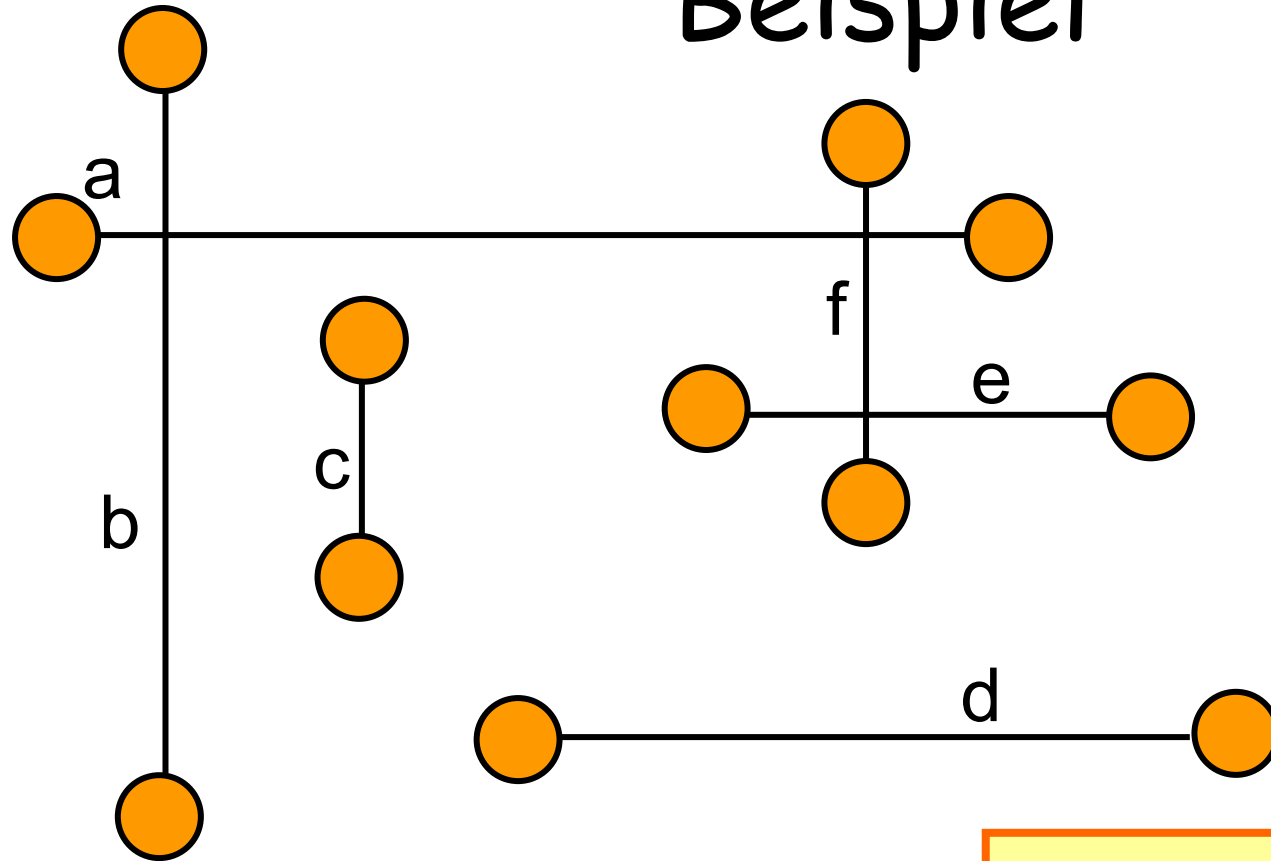
Sweepline Algorithmen allgemein

- Wir interessieren uns für die Punkte, an denen sich etwas ändert: **Ereignispunkte**.
- Ereignispunkte sind z.B. die Stellen an denen wartende Objekte „aktiv“ werden, oder „aktive“ Objekte abgeschlossen werden.
- Diese lassen sich vorher statisch berechnen.
- Es gibt manchmal auch **dynamische Ereignispunkte**: diese tauchen erst während der Sweepline-Bewegung auf und werden während des Verfahrens dynamisch hinzugefügt.

Schnittproblem für iso-orientierte Liniensegmente

- **Gegeben:** Menge $S = \{s_1, \dots, s_n\}$ von **vertikalen und horizontalen** Liniensegmenten im 2-dim. Raum
- **Gesucht:** alle Paare sich schneidender Segmente

Beispiel



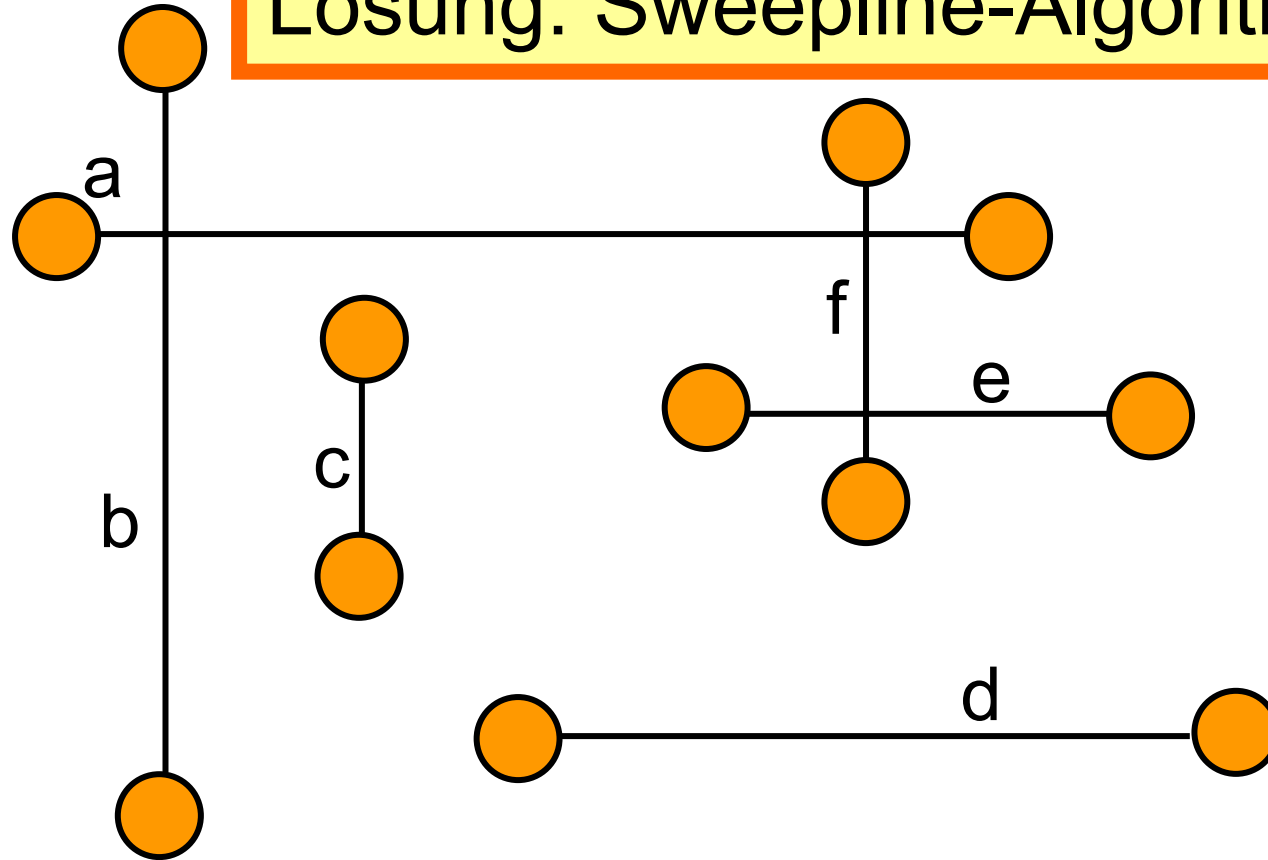
Naive Lösung: $\Theta(n^2)$

Geht es besser?

Im Worst Case: NEIN, denn es kann bis zu $\Theta(n^2)$ viele Schnittpunkte geben

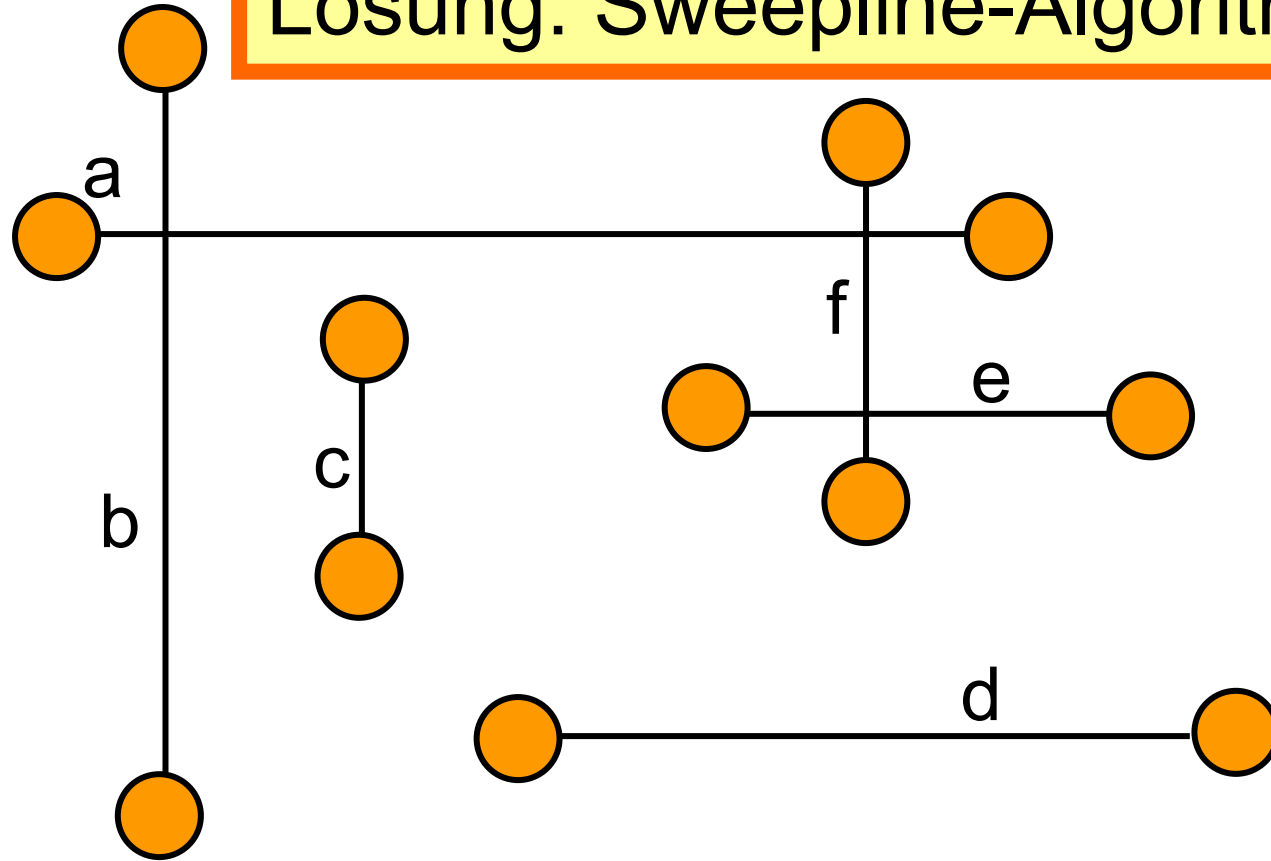
Aber eben nur im Worst Case...

Lösung: Sweepline-Algorithmus



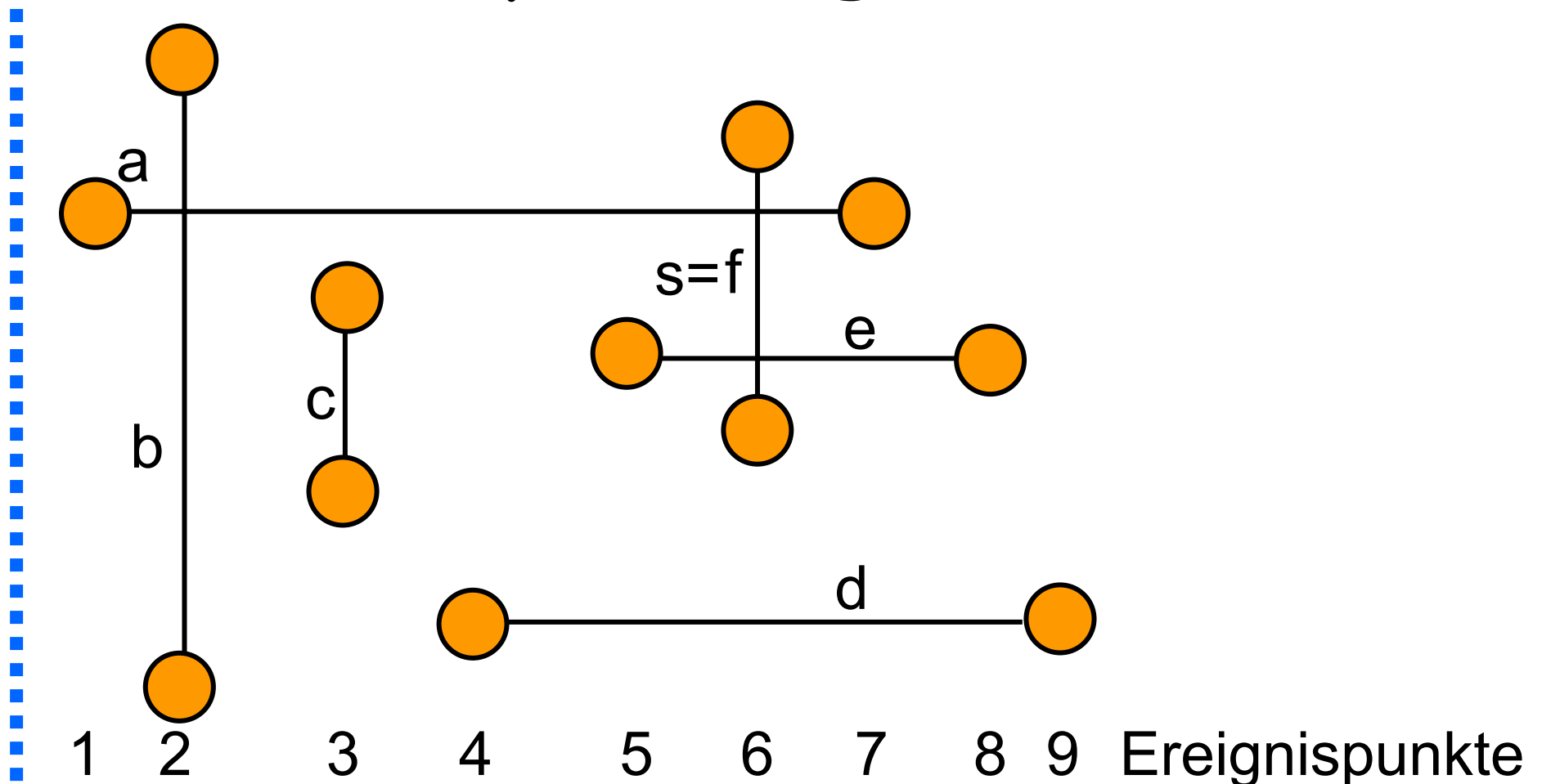
Wunsch: **output sensitiver** Algorithmus:
Laufzeit hängt von der Länge der Ausgabe ab,
d.h., wenn Anzahl der Schnittpunkte in $o(n^2)$,
dann auch die Laufzeit.

Lösung: Sweepline-Algorithmus



Annahme: Die x -Koordinaten aller Anfangs- und Endknoten der horizontalen Segmente und aller vertikalen Segmente sind verschieden.

Sweepline Algorithmus



Beobachtung: Trifft die Sweepline L auf ein vertikales Liniensegment s , so kann sich s nur mit den derzeit aktiven Segmenten schneiden.

Ereignispunkte

Dies führt zu drei Typen von Ereignispunkten:

(E1) Für alle horizontalen Segmente: x -Koordinate des linken Endpunkts: **an dieser Stelle wird das Segment aktiv**

(E2) Für alle horizontalen Segmente: x -Koordinate des rechten Endpunkts: **an dieser Stelle wird das Segment abgeschlossen**

(E3) Für alle vertikalen Segmente: x -Koordinate: **an dieser Stelle werden die Schnitte berechnet.**

Berechnung der Ereignispunkte

- Berechnung durch einfaches Ablaufen der Eingabedaten
- Die Ereignispunkte müssen noch sortiert werden
- Da es $\Theta(n)$ Ereignispunkte gibt, ist dies in $O(n \log n)$ möglich.

Verwaltung der Menge der aktiven Segmente

- Wir benötigen folgende Operationen:
 - **Einfügen** eines Elements (bei Ereignispunkten des Typs E1)
 - **Entfernen** eines Elements (bei Ereignispunkten des Typs E2)
 - Alle Elemente bestimmen, die in einen gegebenen **Bereich** $[y_{\min}, y_{\max}]$ fallen (bei Ereignispunkten des Typs E3)

Verwaltung der Menge der aktiven Segmente

- Realisierung z.B. durch **balancierte Suchbäume**, z.B. **AVL-Bäume**.
- Dann geht Einfügen und Entfernen in $O(\log n)$.
- Bereichsabfrage: $O(\log n + r)$, wobei r die output-Größe ist:
 - Suche des kleinsten Elements, das in den Suchbereich fällt (SEARCH und evtl. ein zusätzlicher SUCCESSOR-Aufruf) und
 - Durchlaufen des Baumes in Inorder-Reihenfolge bis zu einem Element außerhalb des Suchbereiches

Sweepline Algorithmus für Schnittproblem für iso-orientierte Liniensegmente

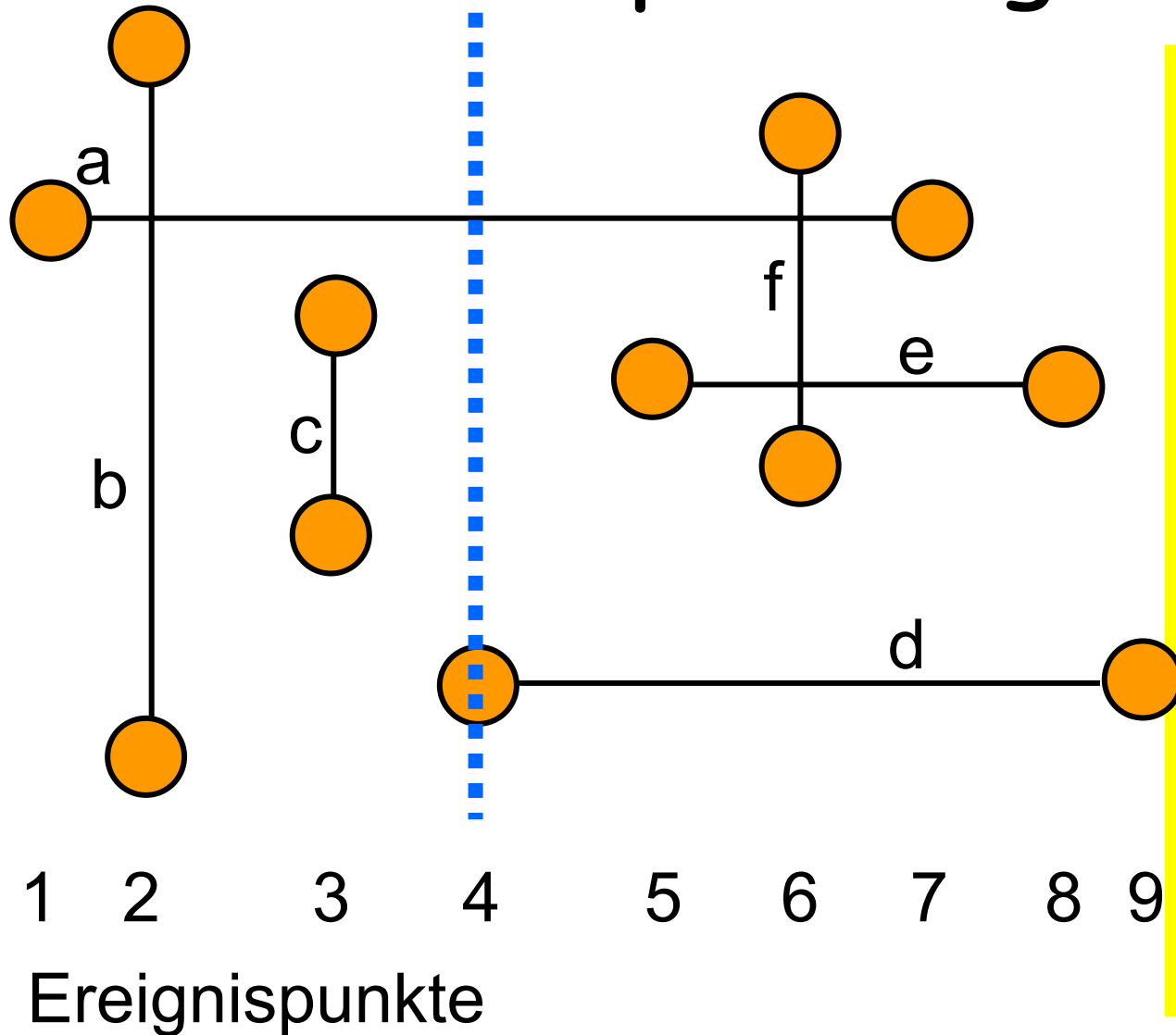
Idee:

- Menge der aktiven Elemente A besteht nur aus den horizontalen Segmenten
- Sobald man auf ein vertikales Segment s trifft, muss man die aktiven Segmente auf Schnittpunkte prüfen.
- Seien y_{\min} und y_{\max} die Koordinaten der Endpunkte von s , dann sind dies genau diejenigen Segmente mit y -Koordinate im Bereich $[y_{\min}, y_{\max}]$.

Sweepline Algorithmus

- (1) Bestimme die Menge der Ereignispunkte Q aus S
- (2) Sortiere Q nach aufsteigender x -Koordinate, $A := \emptyset$
- (3) **for all** Ereignispunkte $p \in Q$ **do** {
- (4) $s :=$ Liniensegment das zu p gehört
- (5) **if** s ist horizontal, dann
- (6) **if** p ist linker Endpunkt von s then
- (7) Einfügen von s in A // (aktive Menge)
- (8) **else** Entfernen von s aus A
- (9) **else** // s ist vertikal
- (10) Bestimme alle Segmente $t \in A$, deren y -Koordinate im Bereich $[y_{\min}, y_{\max}]$ liegen und gib (s, t) als Paar sich schneidender Segmente aus
- (11) }

Sweepline Algorithmus



E	s	A	out
1	a	{a}	
2	b	{a}	→(b,a)
3	c	{a}	→∅
4	d	{d,a}	
5	e	{d,e,a}	
6	f	{d,e,a}	→(f,e) →(f,a)
7	a	{d,e}	
8	e	{d}	
9	d	∅	

Analyse des Sweepline Algorithmus

- Anzahl der Ereignispunkte: jeweils $O(n)$
Ereignispunkte des Typs E1, E2 und E3.
- für E1 und E2: Laufzeit je $O(\log n)$
- für E3: $O(\log n + r)$.
- Da jeder Schnittpunkt genau einmal gefunden wird, ist die Laufzeit insgesamt (über alle Ereignispunkte) $O(n \log n + R)$, wobei R die Anzahl der Schnittpunkte ist.

- Man kann zeigen: Es kann keinen Algorithmus geben, der weniger als $\Omega(n \log n + R)$ Schritte benötigt.

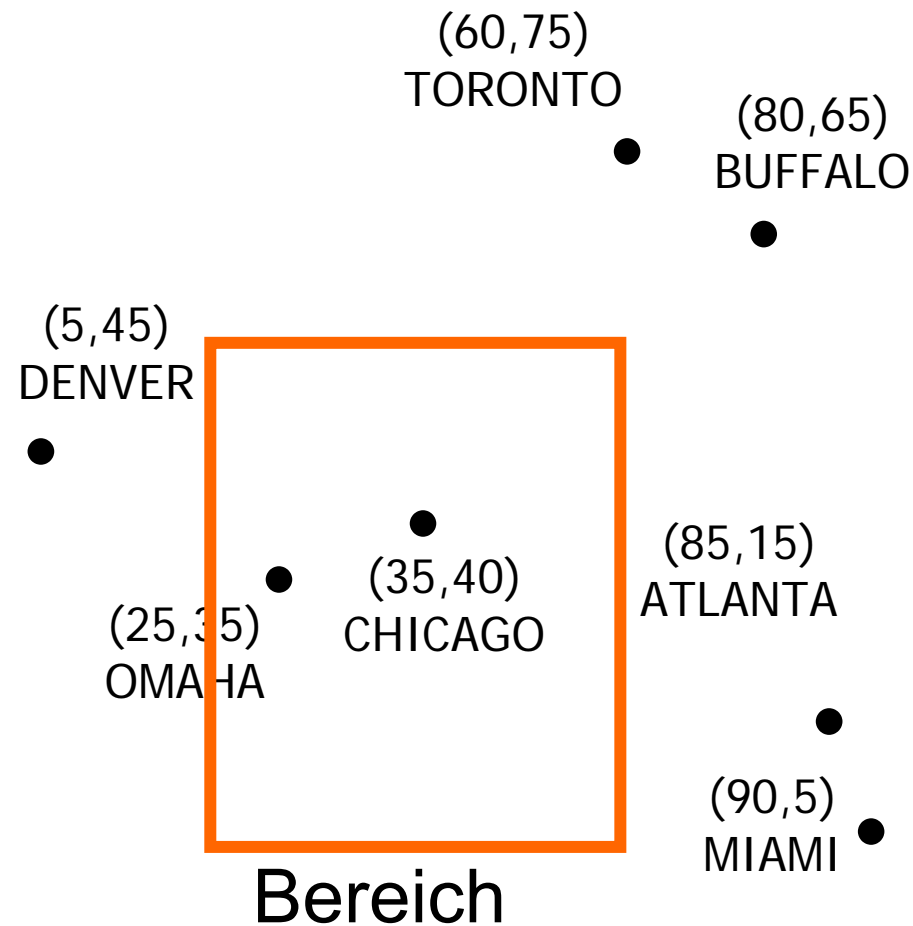
Erweiterungen

- Ähnliche Verfahren lassen sich auch auf folgende Probleme anwenden:
 - Schnittproblem für Segmente (allgemein)
 - Schnittberechnung bzw. Inklusionen von achsenparallelen Rechtecken
 - auch Anwendung auf höherdimensionale Bereiche: Sweep-plane statt Sweepline.

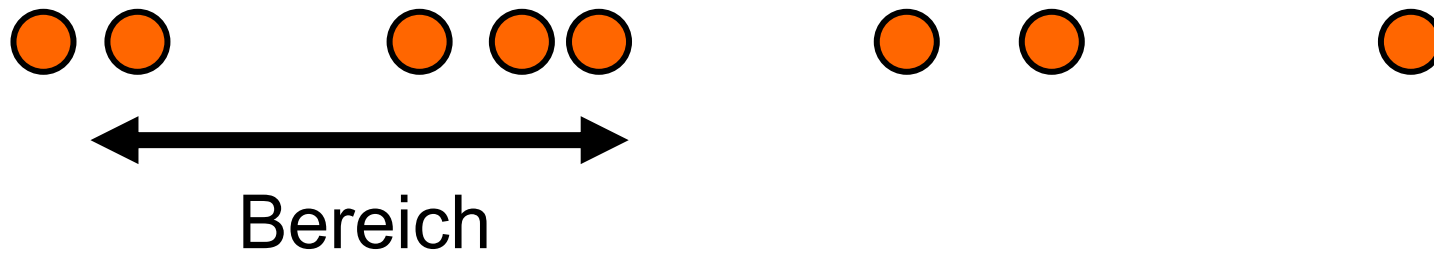
8.2 Mehrdimensionale Bereichssuche

- **Gegeben:** Punktmenge $P = \{p_1, p_2, \dots, p_n\}$ im k -dimensionalen Raum und ein k -dimensionaler (achsenparalleler) Bereich D
 - **Gesucht:** alle Punkte, die in D liegen
-
- **Z.B. Datenbankabfrage:** Ausgabe aller Studierenden an der Universität Dortmund, die zwischen 22 und 26 Jahre alt sind, in Essen wohnen, im Nebenfach Anglistik studieren, und DAP2 gut finden.

Beispiel: Bereichssuche in 2D



Beispiel: Bereichssuche in 1D



Mehrdimensionale Bereichssuche

- Naive Methode: Teste jeden Punkt, ob seine Koordinaten innerhalb des Bereichs liegen.
- Test dauert für einen Punkt jeweils $O(k)$
- Insgesamt: $O(kn)$

- Für $k=1$: bessere Methode (wie bei Bereichssuche für Schnittproblem in Kap. 8.1):
- Wenn die **Punkte sortiert sind**, dann kann man mit Binärsuche einen Punkt finden, der in den Bereich fällt und von dort aus alle angrenzenden Punkte testen.

Mehrdimensionale Bereichssuche

- Vorverarbeitung mit geometrischen Algorithmen:
 - **Gittermethode:** Fläche wird in regelmäßiges Gitter unterteilt
 - **Quadtree:** rekursive Unterteilung der Fläche, bis nur noch wenige Punkte darin liegen
 - **kD-Bäume:** 2-dim binäre Bäume

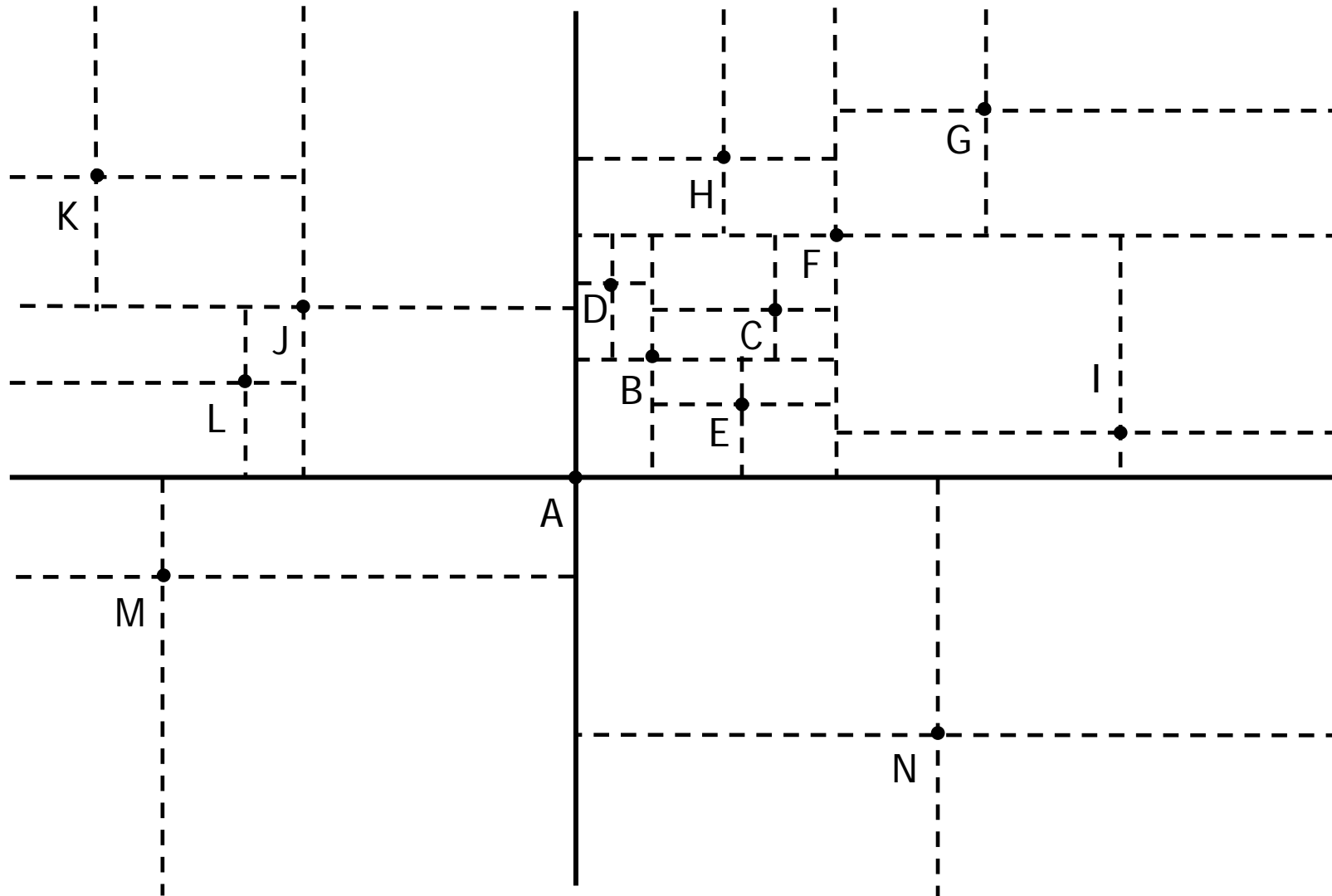
Point Quadrees

Finkel & Bentley 1974

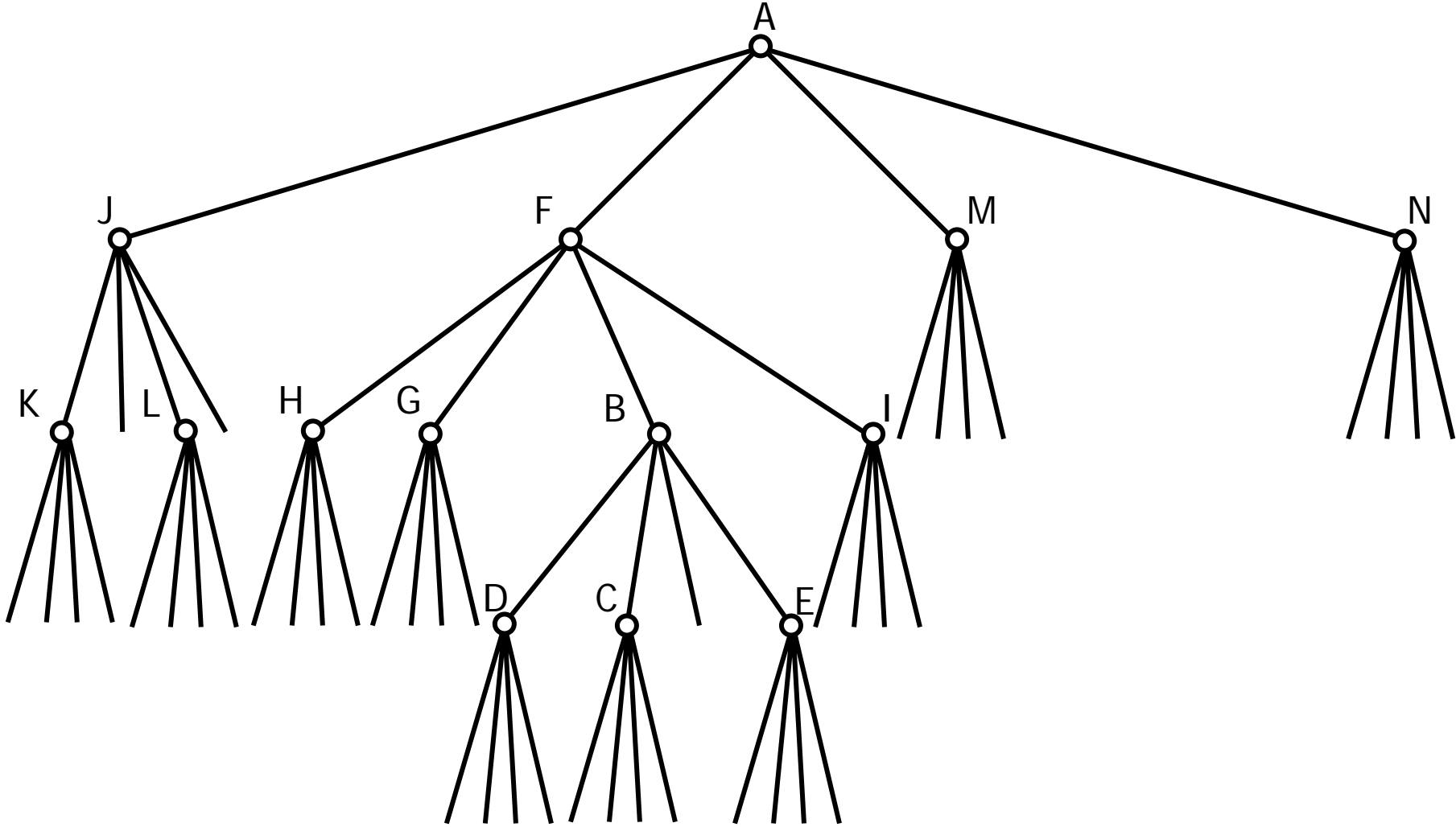
Idee

- Multidimensionale Verallgemeinerung von binären Suchbäumen
- Verheiratung von Gittermethode mit binären Suchbäumen
- Rekursive Teilung an Datenpunkten in jeweils vier Teile: NW, NE, SW, SE

Beispiel:



Point Quadtree zu Beispiel



Bentley 1975

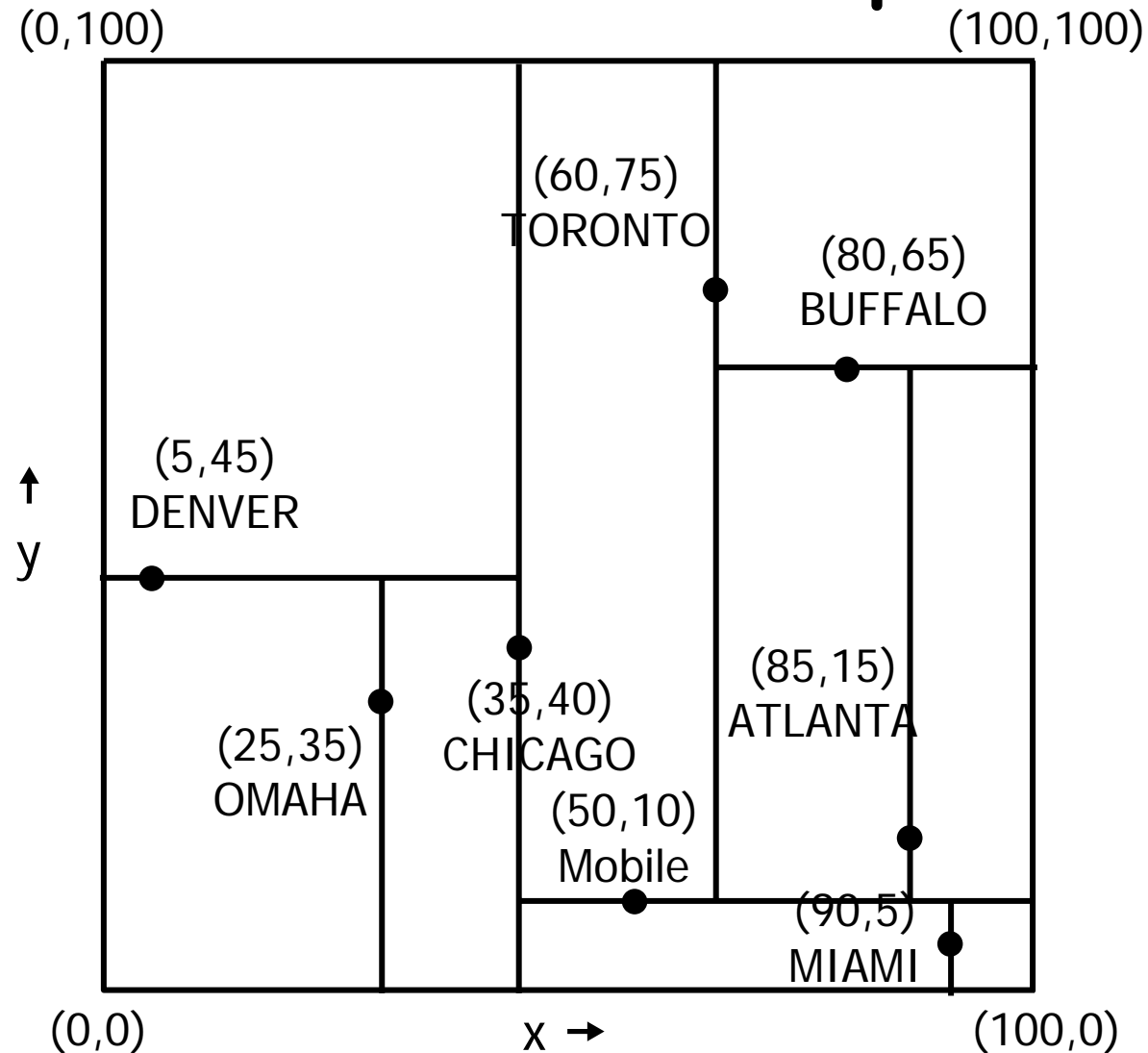
K-D Trees:

Idee:

- Binärer Suchbaum mit der Eigenschaft, dass in jeder Tiefe nach einer anderen Dimension orthogonal aufgeteilt wird.
- Z.B. $k=2$: nach x -Koordinaten auf den Schichten mit gerader Nummer (Beginn bei Schicht 0), nach y -Koordinaten auf den ungeraden Schichten.
- Aufteilung basiert auf den Datenpunkten

mehr zu Bereichssuche für Interessierte: s. Skript

K-D Tree: Beispiel



in eigener Sache / Klausur:

- Da ich es nicht geschafft habe, Kapitel 7 und 8 des Skriptes rechtzeitig herauszubringen, gilt für den Klausurtermin im Juli 2006: Kapitel 7 und 8 sind vom Stoff ausgeschlossen! ...

- Vielleicht ist das ein Ansporn für Sie richtig gut zu lernen, denn beim Nebentermin kommt das alles wieder dran...

- **VIEL ERFOLG** bei der Klausur! Vielleicht sehen wir uns ja wieder, wenn Sie eine meiner Vorlesungen / Seminare / Projektgruppen besuchen.