Karlsruhe Institute of Technology

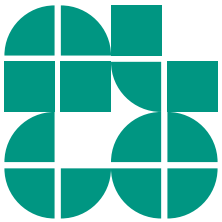
# Vorlesung Algorithmische Geometrie

## Punktlokalisierung

INSTITUT FÜR THEORETISCHE INFORMATIK · FAKULTÄT FÜR INFORMATIK

Martin Nöllenburg

22.05.2012



# Nachtrag: Dynamische Bereichsabfragen

Karlsruhe Institute of Technology

**Letzte Woche:**
kd-Trees und Range-Trees für orthogonale Bereichsabfragen

**Frage:** Lassen diese Datenstrukturen sich für dynamische Punktemengen anpassen?

- Einfügen von Punkten
- Löschen von Punkten

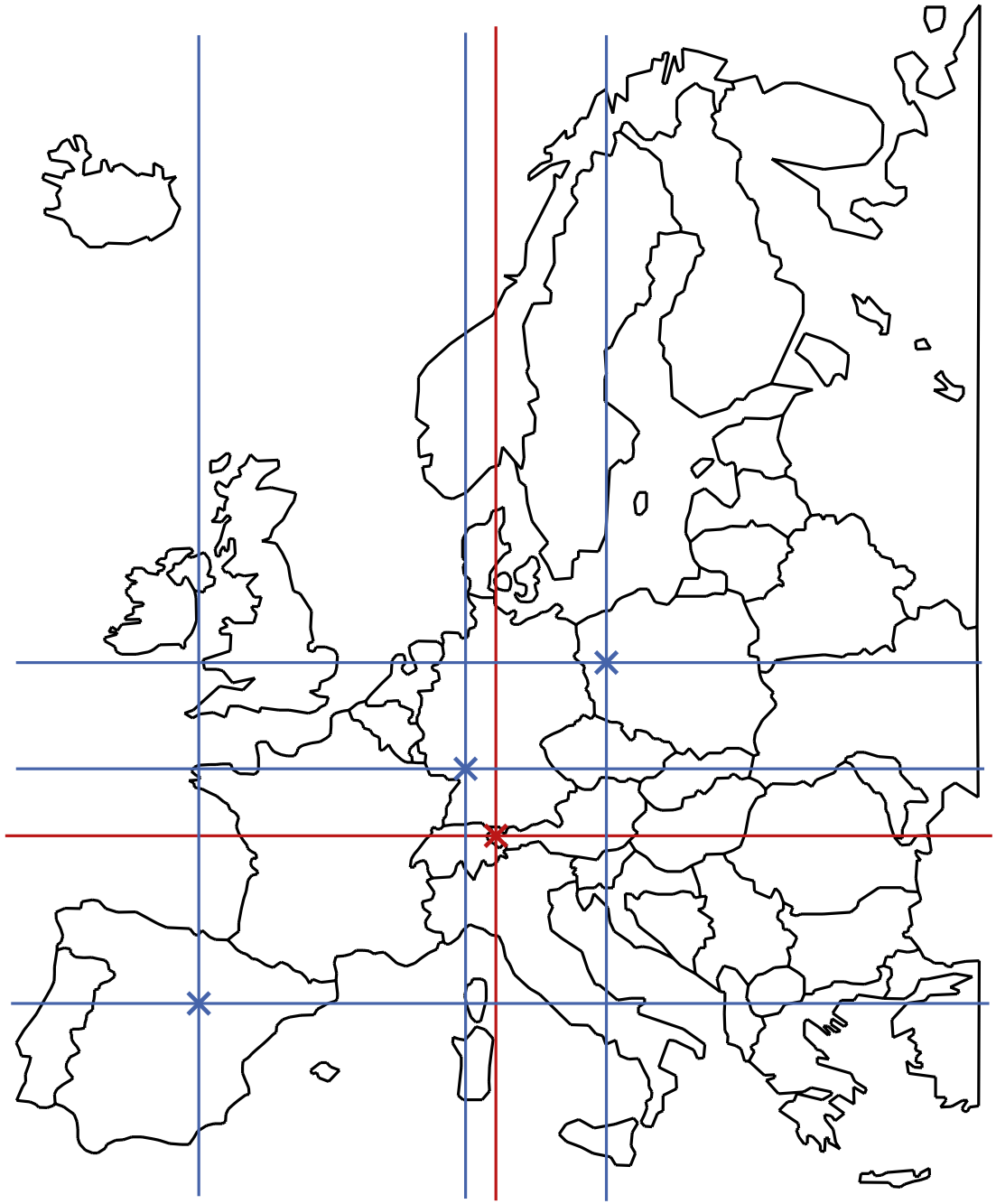
1) **Divided kd-trees** [van Kreveld, Overmars '91]
erlauben Updates in $O(\log n)$ Zeit, aber die Queryzeit steigt auf $O(\sqrt{n \log n} + k)$

2) **Augmented dynamic range-trees** [Mehlhorn, Näher '90]
erlauben Updates in $O(\log n \log \log n)$ Zeit und Queries in $O(\log n \log \log n + k)$ Zeit

Karlsruhe Institute of Technology

# Punktlokalisierung

# Motivation

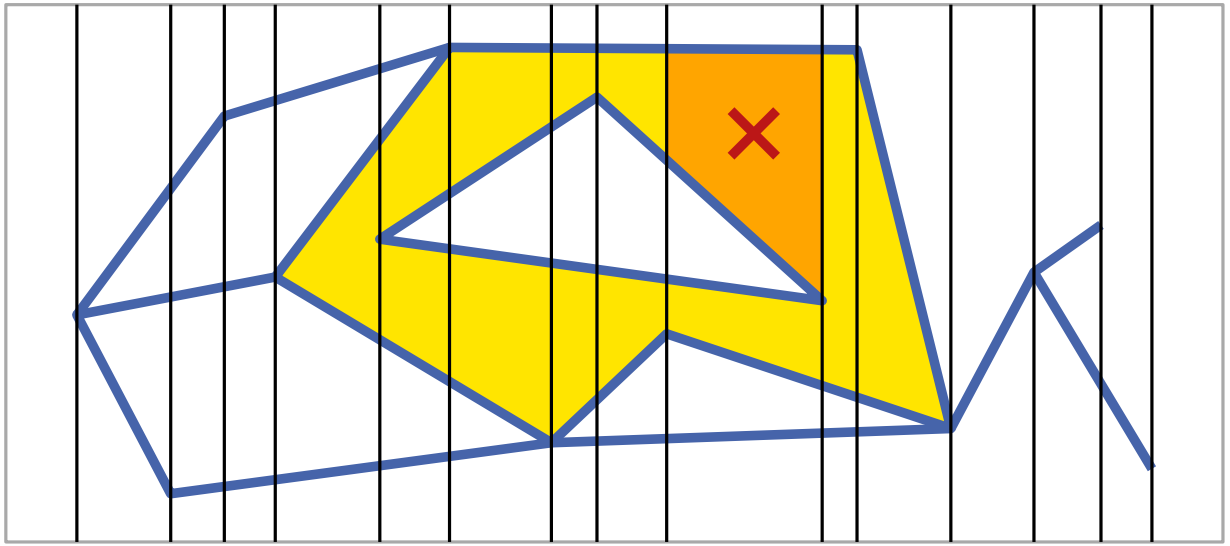


Gegeben eine Position $p = (p_x, p_y)$ in einer Landkarte, bestimme in welchem Land $p$ liegt.

**Genauer:** Finde eine Datenstruktur, die solche Lokalisierungsanfragen effizient beantworten kann.

Dabei ist die Landkarte modelliert als Unterteilung der Ebene in disjunkte Polygone.

# Problemstellung



**Ziel:** Geg. eine Unterteilung $\mathcal{S}$ der Ebene mit $n$ Strecken, erstelle Datenstruktur für schnelle Punktlokalisierung.

**Lösung:** Zerteile $\mathcal{S}$ an Punkten in vertikale Streifen. $O(\log n)$ Zeit

Anfrage:
- finde richtigen Streifen
- durchsuche diesen Streifen

} 2 binäre Suchen

**Aber:** Platz? $\Theta(n^2)$

**Frage:** Bsp. für untere Schranke?

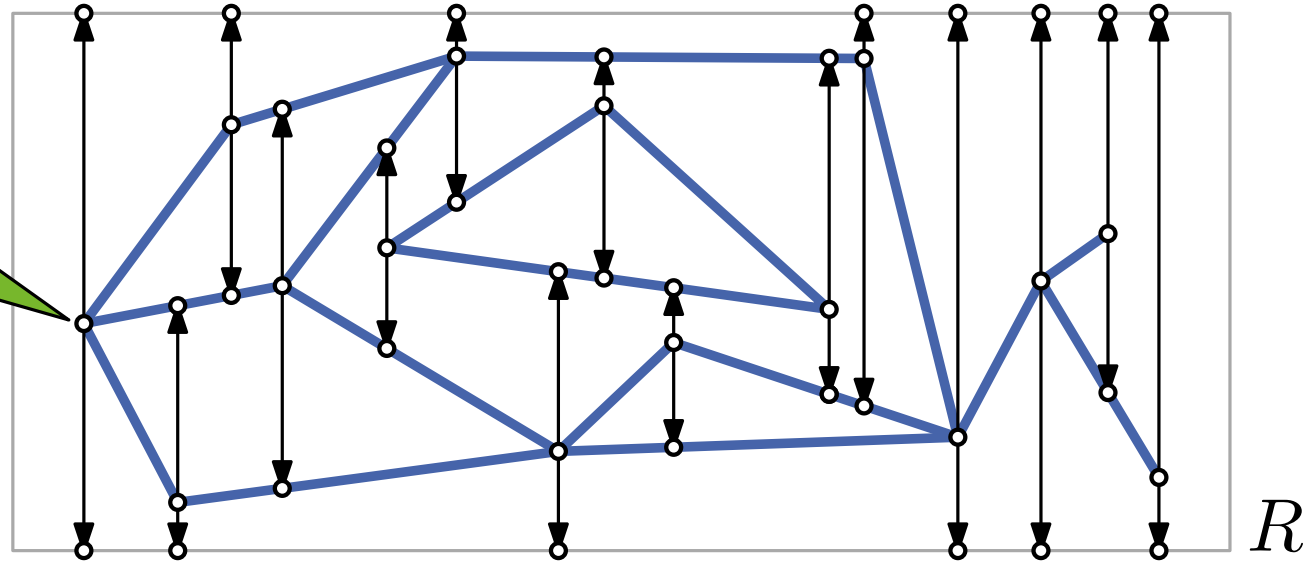
# Verringern der Komplexität

**Beob.:** Die Streifenzerlegung ist eine Verfeinerung $\mathcal{S}'$ von $\mathcal{S}$ in (evtl. degenerierte) Trapeze.

**Ziel:** Finde geeignete Verfeinerung von $\mathcal{S}$ mit geringerer Komplexität!

**Lösung:** *Trapezzerlegung* $\mathcal{T}(\mathcal{S})$

Kante von jedem Endpunkt vertikal nach oben und unten bis zum Nachbarsegment

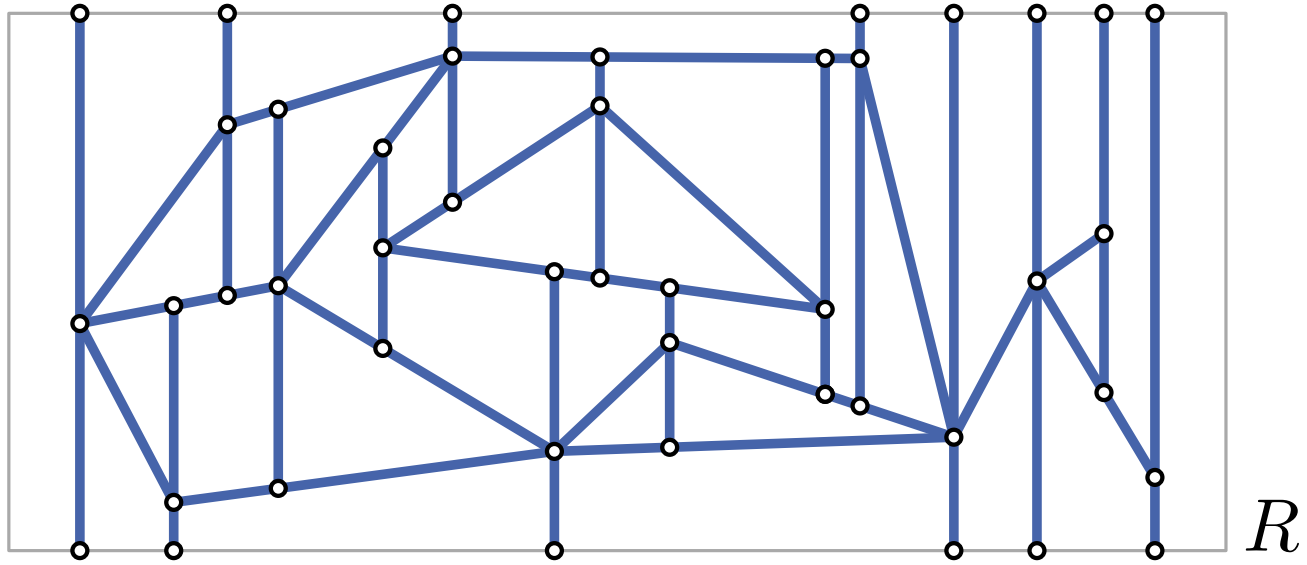


# Verringern der Komplexität

**Beob.:** Die Streifenzerlegung ist eine Verfeinerung $\mathcal{S}'$ von $\mathcal{S}$ in (evtl. degenerierte) Trapeze.

**Ziel:** Finde geeignete Verfeinerung von $\mathcal{S}$ mit geringerer Komplexität!

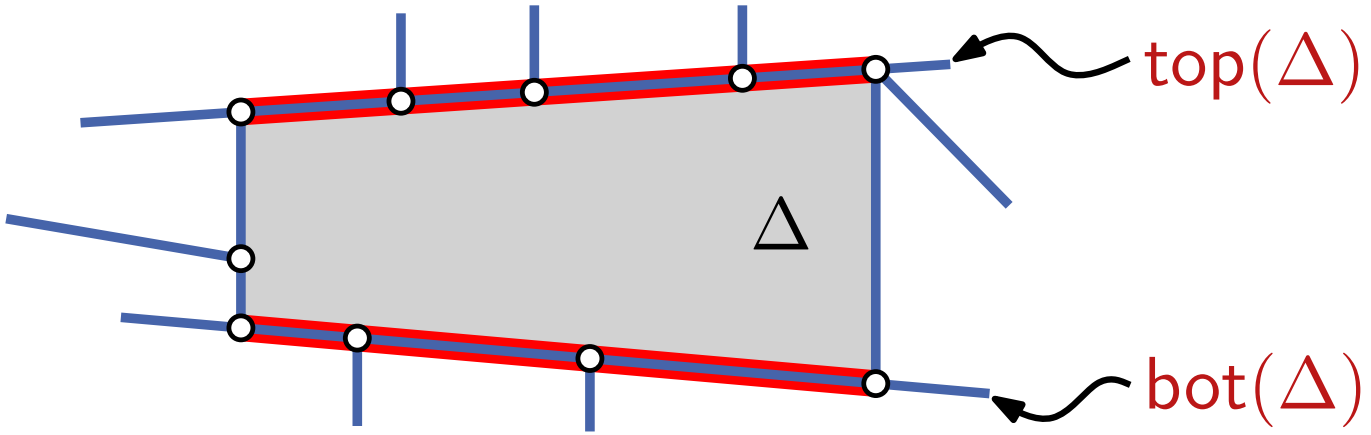
**Lösung:** *Trapezzerlegung* $\mathcal{T}(\mathcal{S})$




**Annahme:** $\mathcal{S}$ ist in *allgemeiner Lage*, d.h. keine zwei Knoten haben gleiche $x$-Koordinaten.

# Notation

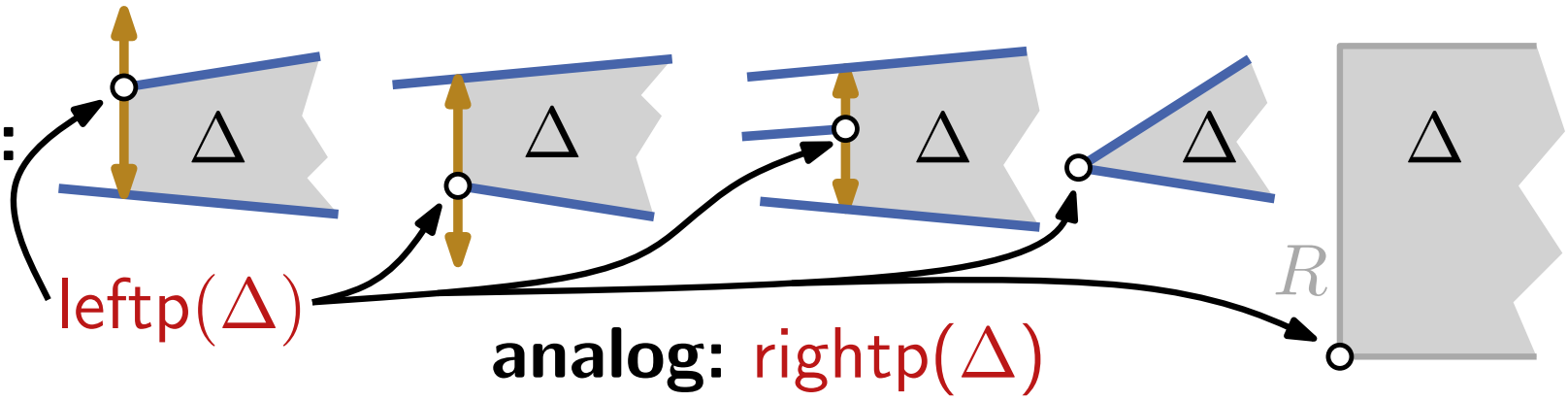
**Definition:** Eine *Seite* einer Facette von $\mathcal{T}(\mathcal{S})$ ist eine maximal lange Teilstrecke der Facettengrenze.




**Beob.:** $\mathcal{S}$ in allg. Lage $\Rightarrow$ jede Facette $\Delta$ von $\mathcal{T}(\mathcal{S})$ hat:

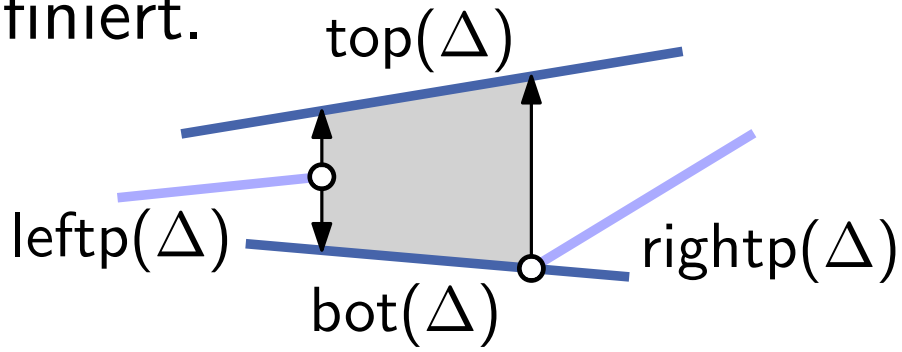
- ein oder zwei vertikale Seiten
- zwei nicht-vertikale Seiten

**Linke Seite:**



# Komplexität der Trapezzerlegung

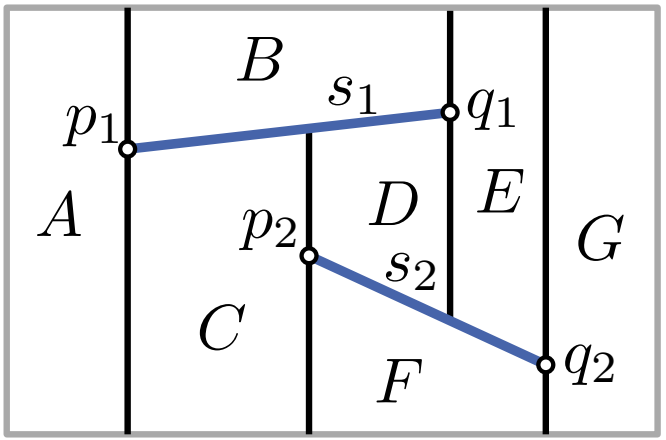
**Beob.:** Ein Trapez $\Delta$ wird eindeutig durch bot($\Delta$), top($\Delta$), leftp($\Delta$) und rightp($\Delta$) definiert.



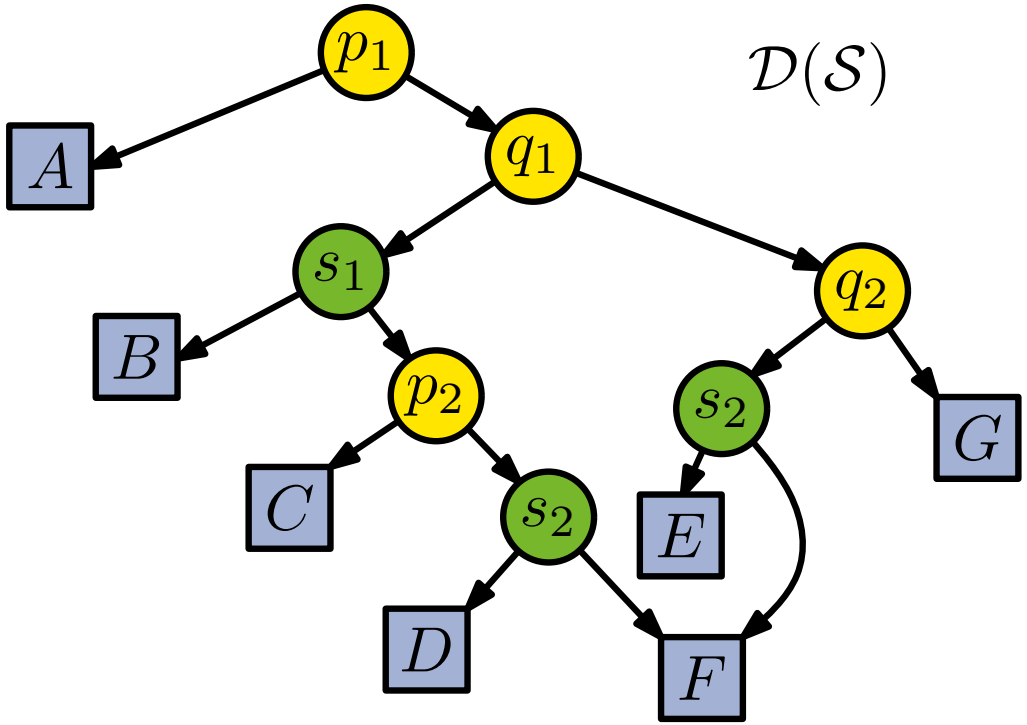

**Lemma 1:** Die Trapezzerlegung $\mathcal{T}(\mathcal{S})$ einer Menge $\mathcal{S}$ von $n$ Strecken in allgemeiner Lage enthält höchstens $6n + 4$ Knoten und höchstens $3n + 1$ Trapeze.

# Suchstruktur

**Ziel:** Berechne die Trapezzerlegung $\mathcal{T}(\mathcal{S})$ und gleichzeitig eine Datenstruktur $\mathcal{D}(\mathcal{S})$ zur Lokalisierung in $\mathcal{T}(\mathcal{S})$.




$\mathcal{T}(\mathcal{S})$




$\mathcal{D}(\mathcal{S})$ ist ein DAG mit:

- $p$ — $x$-Knoten für Punkt $p$ testet auf links/rechts von $p$
- $s$ — $y$-Knoten für Strecke $s$ testet auf oberhalb/unterhalb von $s$
- $\Delta$ — Blattknoten für Trapez $\Delta$

# Inkrementeller Algorithmus

Trapezzerlegung($\mathcal{S}$)

**Input**: Menge $\mathcal{S} = \{s_1, \ldots, s_n\}$ von kreuzungsfreien Strecken
**Output**: Trapezzerlegung $\mathcal{T}(\mathcal{S})$ und Suchstruktur $\mathcal{D}(\mathcal{S})$
Initialisiere $\mathcal{T}$ und $\mathcal{D}$ für $R = \text{BBox}(\mathcal{S})$
$\mathcal{S} \leftarrow$ **RandomPermutation**$(\mathcal{S})$
**for** $i \leftarrow 1$ **to** $n$ **do**
  $H \leftarrow \{\Delta \in \mathcal{T} \mid \Delta \cap s_i \neq \emptyset\}$
  $\mathcal{T} \leftarrow \mathcal{T} \setminus H$
  $\mathcal{T} \leftarrow \mathcal{T} \cup$ neue durch $s_i$ erzeugte Trapeze
  $\mathcal{D} \leftarrow$ ersetze Blätter für $H$ durch Blätter für neue Trapeze
**return** $(\mathcal{T}, \mathcal{D})$

**Problem:** Größe von $\mathcal{D}$ und Anfragezeit sind abhängig von Einfügereihenfolge

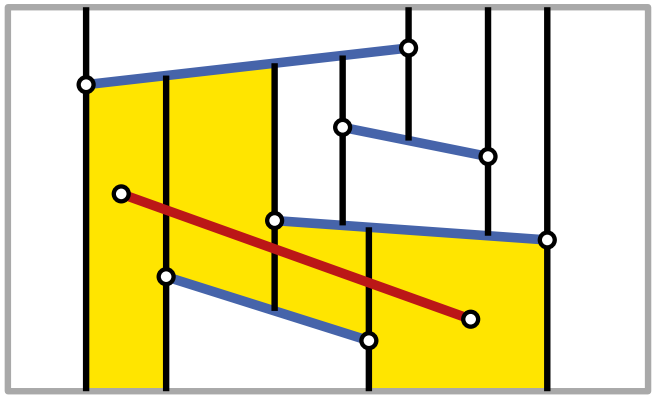
**Lösung:** Randomisierung!

# Randomisierter Inkrementeller Algorithmus

**Invariante:** $\mathcal{T}$ ist Trapezzerlegung für $\mathcal{S}_i = \{s_1, \ldots, s_i\}$ und $\mathcal{D}$ passende Suchstruktur

**Initialisierung:** $\mathcal{T} = \mathcal{T}(\emptyset) = R$ und $\mathcal{D} = (R, \emptyset)$

**Schritt 1:** $H \leftarrow \{\Delta \in \mathcal{T} \mid \Delta \cap s_i \neq \emptyset\}$



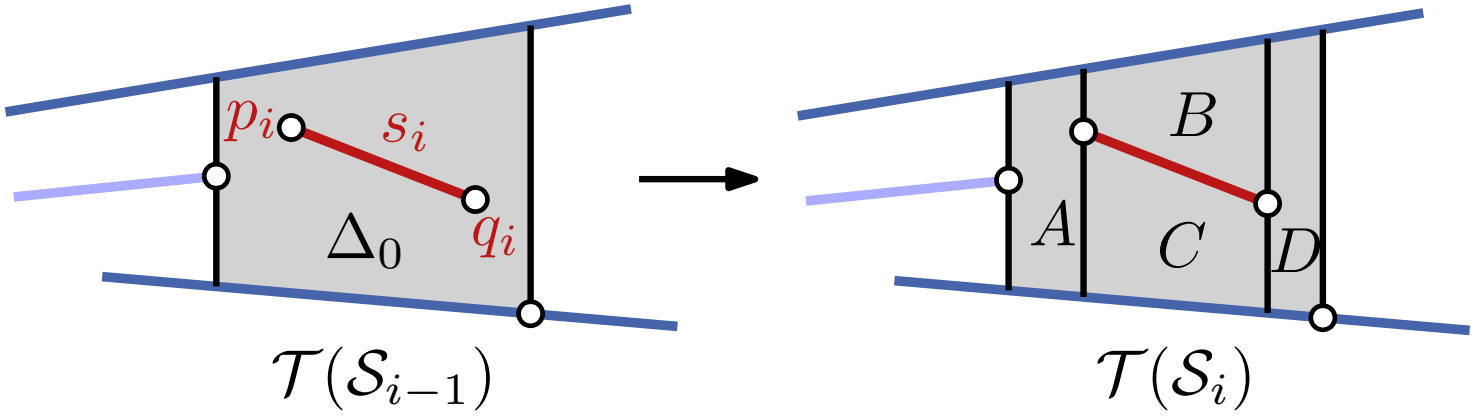
**Aufgabe:** Wie findet man die Trapezmenge $H$ von links nach rechts?

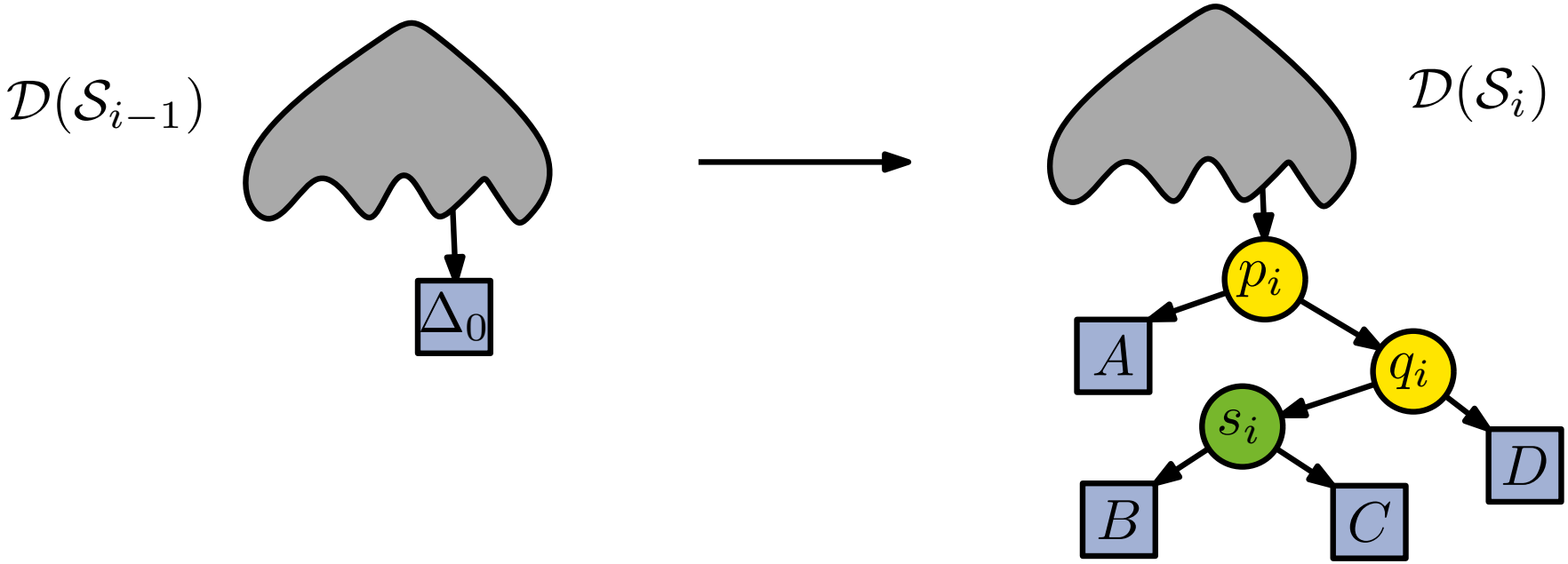
$\Delta_0 \leftarrow$ FindeTrapez$(p_i, \mathcal{D})$; $j \leftarrow 0$
**while** rechter Endpunkt $q_i$ rechts von rightp$(\Delta_j)$ **do**
  **if** rightp$(\Delta_j)$ über $s_i$ **then**
    $\Delta_{j+1} \leftarrow$ unterer rechter Nachbar von $\Delta_j$
  **else**
    $\Delta_{j+1} \leftarrow$ oberer rechter Nachbar von $\Delta_j$
  $j \leftarrow j + 1$
**return** $\Delta_0, \ldots, \Delta_j$

# Update von $\mathcal{T}(\mathcal{S})$ und $\mathcal{D}(\mathcal{S})$

**Schritt 2:** Aktualisiere $\mathcal{T}$ und $\mathcal{D}$

- Fall 1: $s_i \subset \Delta_0$

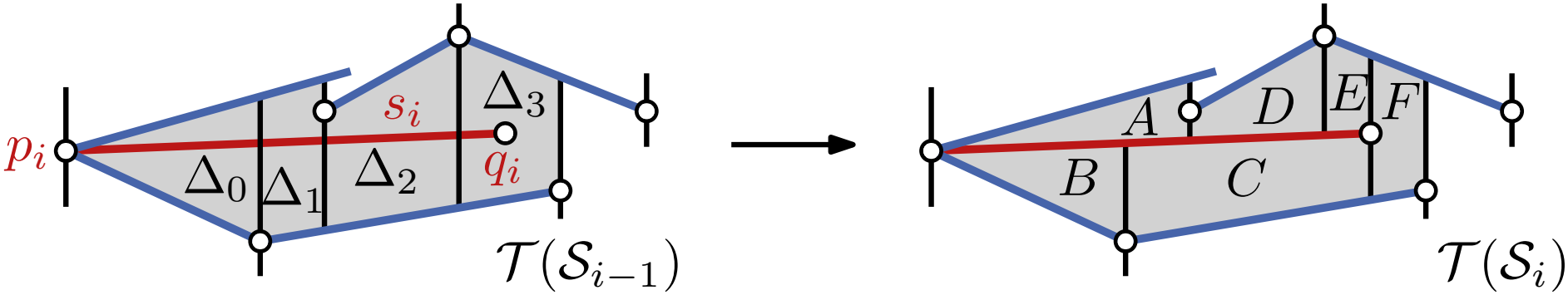


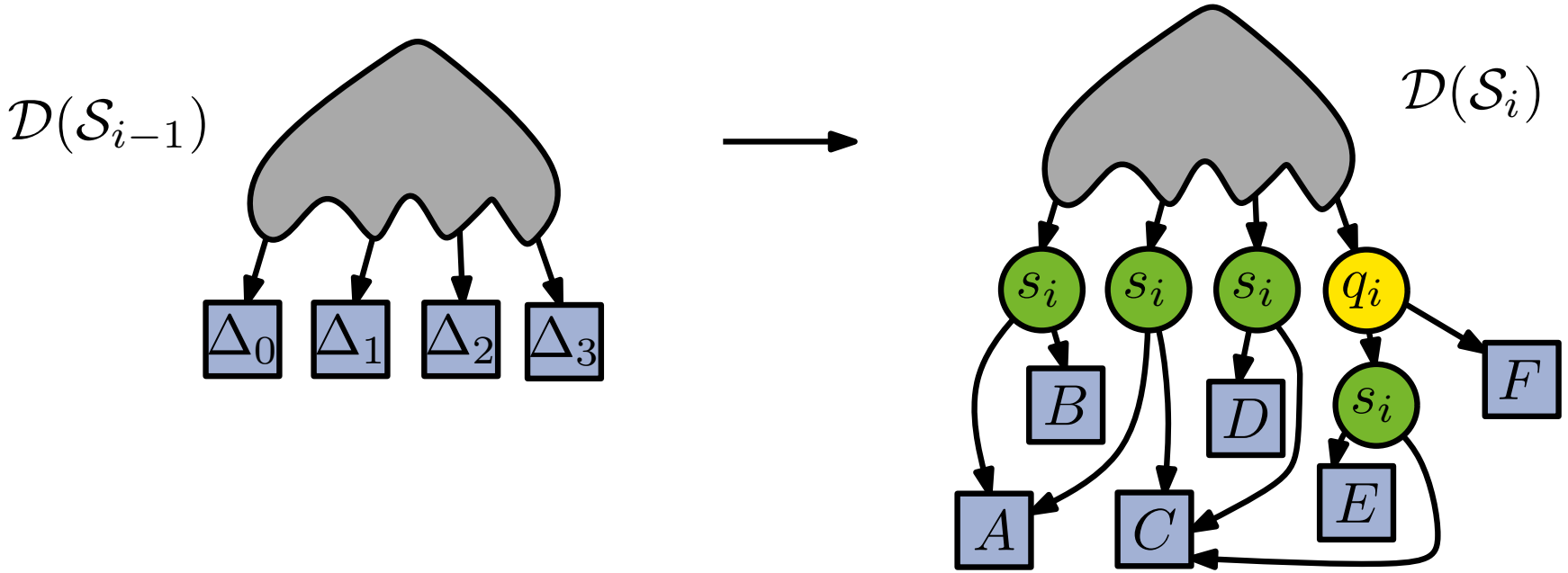



# Update von $\mathcal{T}(\mathcal{S})$ und $\mathcal{D}(\mathcal{S})$

**Schritt 2:** Aktualisiere $\mathcal{T}$ und $\mathcal{D}$

- Fall 1: $s_i \subset \Delta_0$
- Fall 2: $|\mathcal{T} \cap s_i| \geq 2$



# Analyse

**Satz 1:** Der Algorithmus berechnet die Trapezzerlegung $\mathcal{T}(\mathcal{S})$ und die Suchstruktur $\mathcal{D}$ für eine Menge $\mathcal{S}$ von $n$ Strecken in *erwartet* $O(n \log n)$ Zeit. Die *erwartete* Größe von $\mathcal{D}$ ist $O(n)$ und die *erwartete* Anfragezeit $O(\log n)$.

**Beobachtungen:**

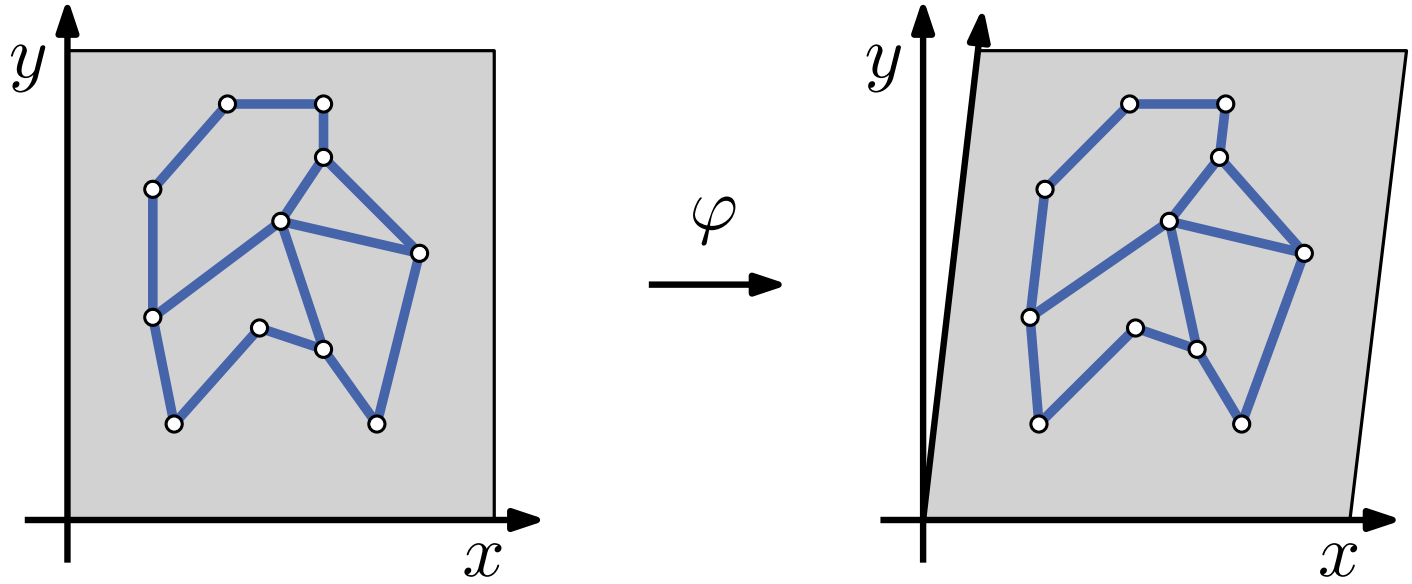
- im schlimmsten Fall kann $\mathcal{D}$ quadratisch groß werden und eine Anfrage lineare Zeit benötigen
- Hoffnung: das passiert nur selten!
- betrachte erwartete Laufzeit und Größe über alle $n!$ möglichen Permutationen von $\mathcal{S}$
- der Satz gilt unabhängig von der Eingabemenge $\mathcal{S}$

# Degenerierte Eingaben

Zwei Annahmen bislang:

- keine zwei Streckenendpunkte habe gleiche $x$-Koordinate
- immer eindeutige Antworten (links/rechts) im Suchpfad

**Ausweg:** symbolische Scherung $\varphi : (x, y) \mapsto (x + \varepsilon y, y)$




Dabei ist $\varepsilon > 0$ so gewählt, dass sich die $x$-Ordnung $<$ der Punkte nicht ändert.

# Degenerierte Eingaben

- Effekt 1: lexikographische Ordnung
- Effekt 2: affine Abb. $\varphi$ erhält Beziehung Punkt–Gerade
- Führe Algorithmus für $\varphi\mathcal{S} = \{\varphi s \mid s \in \mathcal{S}\}$ und $\varphi p$ aus.
- Zwei elementare Operationen beim Aufbau von $\mathcal{T}$ und $\mathcal{D}$:
  1. liegt $q$ links oder rechts der Vertikalen durch $p$?
  2. liegt $q$ oberhalb oder unterhalb der Strecke $s$?
- Auch Lokalisierung eines Punktes $q$ in $\mathcal{T}(\mathcal{S})$ geht durch Lokalisierung von $\varphi q$ in $\mathcal{T}(\varphi\mathcal{S})$.

$\rightarrow$ siehe Kap. 6.3 in de Berg et al. *Computational Geometry*

# Diskussion

## Gibt es ähnliche Methoden auch für höhere Dimensionen?

Die derzeit beste Datenstruktur in drei Dimensionen benötigt $O(n \log n)$ Platz und $O(\log^2 n)$ Anfragezeit [Snoeyink '04]. Ob es mit linearem Platz und $O(\log n)$ Anfragezeit geht ist eine offene Frage. In höheren Dimensionen sind effiziente Verfahren nur für spezielle Unterteilungen durch Hyperebenen bekannt.

## Gibt es dynamische Datenstrukturen, die Einfügen und Löschen zulassen?

Es gibt viele dynamische Datenstrukturen für Punktlokalisierung, siehe das Survey [Chiang, Tamassia '92]. Ein Beispiel von [Fries et al. '85] benötigt $O(n)$ Platz, $O(\log^2 n)$ Abfragezeit und $O(\log^4 n)$ Updatezeit.