Karlsruher Institut für Technologie

Algorithmen II
Vorlesung am 03.12.2013

Algorithmische Geometrie: Schnitte von Strecken – Sweep-Line

INSTITUT FÜR THEORETISCHE INFORMATIK · PROF. DR. DOROTHEA WAGNER

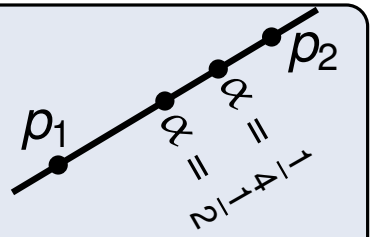
Grundlagen

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Strecken

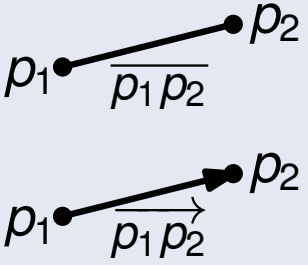
Definition: Konvexkombination **(Definition 6.1)**

Für zwei Punkte p_1 und p_2 (in der Ebene) und $\alpha \in [0,1]$ ist der Punkt $p = \alpha p_1 + (1-\alpha)p_2$ eine *Konvexkombination* von p_1 und p_2.



Definition: Strecke **(Definition 6.2)**

Die Menge aller Konvexkombinationen zweier Punkte p_1, p_2 ist die *Strecke* $\overline{p_1 p_2}$ zwischen p_1 und p_2. Ist die Reihenfolge von p_1 und p_2 von Bedeutung, so sprechen wir von der *gerichteten Strecke* $\overrightarrow{p_1 p_2}$.

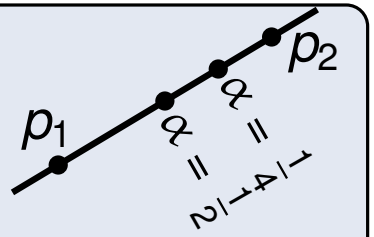


Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Strecken

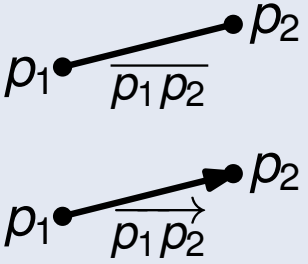
Definition: Konvexkombination **(Definition 6.1)**

Für zwei Punkte p_1 und p_2 (in der Ebene) und $\alpha \in [0,1]$ ist der Punkt $p = \alpha p_1 + (1-\alpha)p_2$ eine *Konvexkombination* von p_1 und p_2.



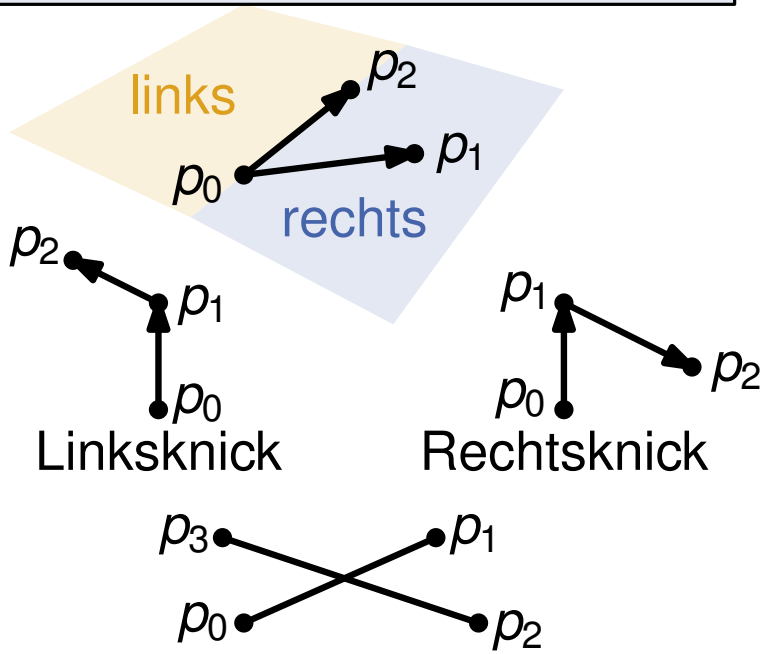
Definition: Strecke **(Definition 6.2)**

Die Menge aller Konvexkombinationen zweier Punkte p_1, p_2 ist die *Strecke* $\overline{p_1 p_2}$ zwischen p_1 und p_2. Ist die Reihenfolge von p_1 und p_2 von Bedeutung, so sprechen wir von der *gerichteten Strecke* $\overrightarrow{p_1 p_2}$.



Grundlegende Fragestellungen:

1. Gegeben drei Punkte p_0, p_1, p_2 in der Ebene. Liegt $\overrightarrow{p_0 p_1}$ rechts oder links von $\overrightarrow{p_0, p_2}$?
2. Bilden die beiden Strecken $\overrightarrow{p_0, p_1}$ und $\overrightarrow{p_1, p_2}$ bei p_1 einen Rechts- oder einen Linksknick?
3. Schneiden sich zwei gegebene Strecken $\overline{p_0 p_1}$ und $\overline{p_2 p_3}$?



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Rechts oder Links?

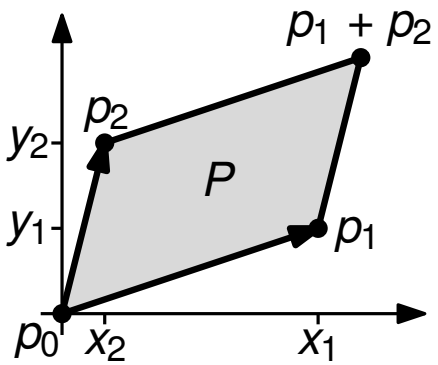
Definition: Kreuzprodukt

Für Strecken $\overrightarrow{p_0p_1}$ und $\overrightarrow{p_0p_2}$ mit $p_0 = (0,0)$, $p_1 = (x_1, y_1)$ und $p_2 = (x_2, y_2)$ ist das *Kreuzprodukt* $p_1 \times p_2$ definiert als

$$p_1 \times p_2 = \det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} = x_1 y_2 - x_2 y_1 = -(p_2 \times p_1)$$

Behauptung: Fläche eines Parallelprogramms **(Siehe Lineare Algebra)**

$|p_1 \times p_2|$ ist die Fläche P des Parallelogramms mit Ecken $(0,0)$, p_1, p_2 und $p_1 + p_2$.



Rechts oder Links?

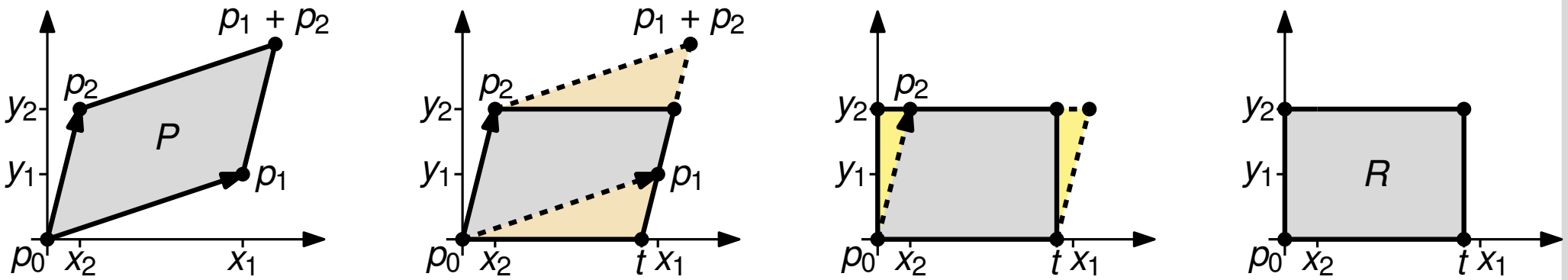
Definition: Kreuzprodukt

Für Strecken $\overrightarrow{p_0p_1}$ und $\overrightarrow{p_0p_2}$ mit $p_0 = (0,0)$, $p_1 = (x_1, y_1)$ und $p_2 = (x_2, y_2)$ ist das *Kreuzprodukt* $p_1 \times p_2$ definiert als

$$p_1 \times p_2 = \det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} = x_1 y_2 - x_2 y_1 = -(p_2 \times p_1)$$

Behauptung: Fläche eines Parallelprogramms **(Siehe Lineare Algebra)**

$|p_1 \times p_2|$ ist die Fläche P des Parallelogramms mit Ecken $(0,0)$, p_1, p_2 und $p_1 + p_2$.



Lemma

Für $p_0 = (0,0)$, p_1 und p_2 liegt $\overrightarrow{p_0p_1}$ rechts von $\overrightarrow{p_0p_2}$ genau dann wenn $p_1 \times p_2 > 0$.

Rechts oder Links?

Lemma
Für $p_0 = (0,0)$, p_1 und p_2 liegt $\overrightarrow{p_0p_1}$ rechts von $\overrightarrow{p_0p_2}$ genau dann wenn $p_1 \times p_2 > 0$.

- Um nun zu entscheiden, ob $\overrightarrow{p_0p_1}$ für beliebiges p_0 rechts von $\overrightarrow{p_0p_2}$ liegt, muss getestet werden ob $(p_1 - p_0) \times (p_2 - p_0) > 0$ gilt.
- Die Strecken $\overrightarrow{p_0p_1}$ und $\overrightarrow{p_1p_2}$ bilden einen Rechtsknick genau dann wenn $\overrightarrow{p_0p_2}$ rechts von $\overrightarrow{p_0p_1}$ liegt.

Rechts oder Links?

Lemma
Für $p_0 = (0,0)$, p_1 und p_2 liegt $\overrightarrow{p_0p_1}$ rechts von $\overrightarrow{p_0p_2}$ genau dann wenn $p_1 \times p_2 > 0$.

- Um nun zu entscheiden, ob $\overrightarrow{p_0p_1}$ für beliebiges p_0 rechts von $\overrightarrow{p_0p_2}$ liegt, muss getestet werden ob $(p_1 - p_0) \times (p_2 - p_0) > 0$ gilt.
- Die Strecken $\overrightarrow{p_0p_1}$ und $\overrightarrow{p_1p_2}$ bilden einen Rechtsknick genau dann wenn $\overrightarrow{p_0p_2}$ rechts von $\overrightarrow{p_0p_1}$ liegt.

Vorteile dieser Methode:
Es müssen Additionen, Subtraktionen, Multiplikationen und Vergleiche durchgeführt werden (insbesondere keine Divisionen, Wurzeln, etc.)
$\Rightarrow$ Bei Ganzzahligen Koordinaten ist das Ergebnis und alle Zwischenschritte Ganzzahlig.
$\Rightarrow$ Keine Fehler durch Ungenauigkeiten bei Gleitkommaarithmetik.

Schneiden sich zwei Strecken?

Definition: Umschließendes Rechteck (bounding box) **(Definition 6.4)**
Das *Umschließende Rechteck (bounding box)* eines geometrischen Objekts ist das kleinste achsenparallele Rechteck, das diese Objekt enthält.

Notwendige Bedingung: Zwei Strecken können sich nur schneiden, wenn sich auch ihre umschließenden Rechtecke schneiden.

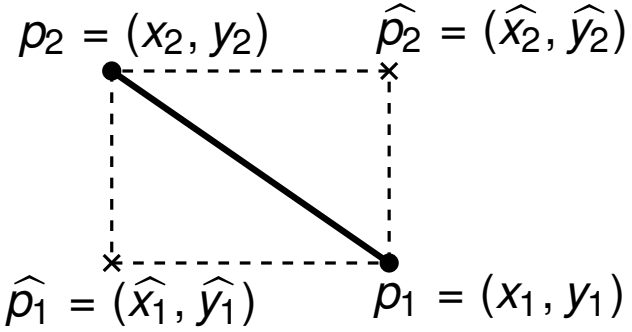
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Schneiden sich zwei Strecken?

Definition: Umschließendes Rechteck (bounding box) **(Definition 6.4)**
Das *Umschließende Rechteck (bounding box)* eines geometrischen Objekts ist das kleinste achsenparallele Rechteck, das diese Objekt enthält.

Notwendige Bedingung: Zwei Strecken können sich nur schneiden, wenn sich auch ihre umschließenden Rechtecke schneiden.

Das umschließende Rechteck $(\widehat{p_1}, \widehat{p_2})$ zu $\overline{p_1 p_2}$ ist beschrieben durch den linken unteren Punkt $\widehat{p_1} = (\widehat{x_1}, \widehat{y_1})$ und den rechten oberen Punkte $\widehat{p_2} = (\widehat{x_2}, \widehat{y_2})$ mit:



$$\widehat{x_1} = \min(x_1, x_2) \quad \widehat{y_1} = \min(y_1, y_2)$$
$$\widehat{x_2} = \max(x_1, x_2) \quad \widehat{y_2} = \max(y_1, y_2)$$

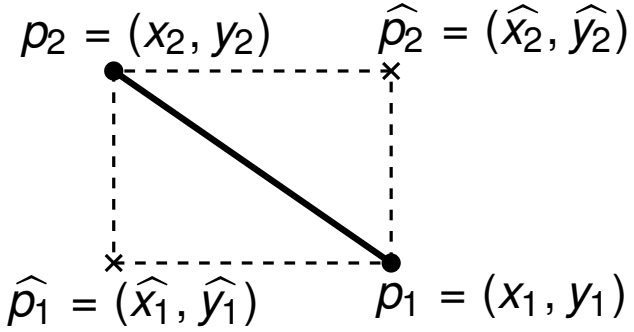
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Schneiden sich zwei Strecken?

Definition: Umschließendes Rechteck (bounding box) **(Definition 6.4)**
Das *Umschließende Rechteck (bounding box)* eines geometrischen Objekts ist das kleinste achsenparallele Rechteck, das diese Objekt enthält.

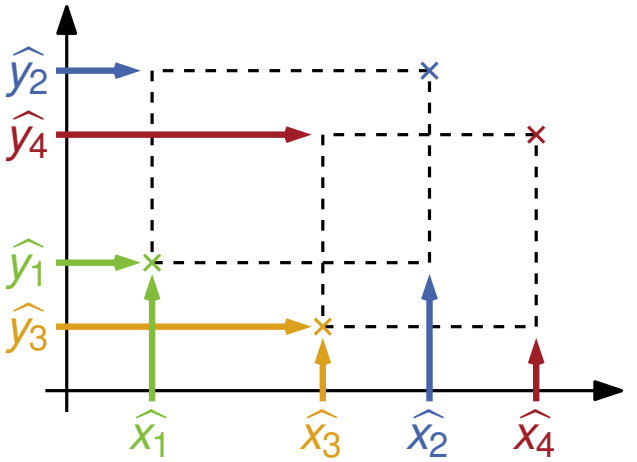
Notwendige Bedingung: Zwei Strecken können sich nur schneiden, wenn sich auch ihre umschließenden Rechtecke schneiden.

Das umschließende Rechteck $(\widehat{p}_1, \widehat{p}_2)$ zu $\overline{p_1 p_2}$ ist beschrieben durch den linken unteren Punkt $\widehat{p}_1 = (\widehat{x}_1, \widehat{y}_1)$ und den rechten oberen Punkte $\widehat{p}_2 = (\widehat{x}_2, \widehat{y}_2)$ mit:



$$\widehat{x}_1 = \min(x_1, x_2) \quad \widehat{y}_1 = \min(y_1, y_2)$$
$$\widehat{x}_2 = \max(x_1, x_2) \quad \widehat{y}_2 = \max(y_1, y_2)$$

Zwei Rechtecke $(\widehat{p}_1, \widehat{p}_2)$ und $(\widehat{p}_3, \widehat{p}_4)$ schneiden sich genau dann, wenn
$\widehat{x}_2 \geq \widehat{x}_3 \wedge \widehat{x}_4 \geq \widehat{x}_1 \wedge \widehat{y}_2 \geq \widehat{y}_3 \wedge \widehat{y}_4 \geq \widehat{y}_1$.



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Schneiden sich zwei Strecken?

Definition **(Definition 6.5)**

$\overline{p_1 p_2}$ *berührt* die Gerade durch $\overline{p_3 p_4}$ falls p_1 und p_2 auf unterschiedlichen Seiten von $\overline{p_3 p_4}$ liegen oder einer von ihnen auf der Gerade liegt.

Ob eine Strecke die Gerade durch zwei Punkte berührt kann wieder mithilfe des Kreuzprodukts getestet werden.

Schneiden sich zwei Strecken?

Definition **(Definition 6.5)**

$\overline{p_1 p_2}$ *berührt* die Gerade durch $\overline{p_3 p_4}$ falls p_1 und p_2 auf unterschiedlichen Seiten von $\overline{p_3 p_4}$ liegen oder einer von ihnen auf der Gerade liegt.

Ob eine Strecke die Gerade durch zwei Punkte berührt kann wieder mithilfe des Kreuzprodukts getestet werden.

Lemma

Zwei Strecken schneiden sich genau dann, wenn ihre umschließenden Rechtecke sich schneiden und jede der beiden Strecken die Gerade durch die jeweils andere Strecke berührt.

Vorteile dieser Methode:

Wie zuvor müssen Additionen, Subtraktionen, Multiplikationen und Vergleiche durchgeführt werden (insbesondere keine Divisionen, Wurzeln, etc.)

$\Rightarrow$ Bei Ganzzahligen Koordinaten ist das Ergebnis und alle Zwischenschritte Ganzzahlig.

$\Rightarrow$ Keine Fehler durch Ungenauigkeiten bei Gleitkommaarithmetik.

Karlsruher Institut für Technologie

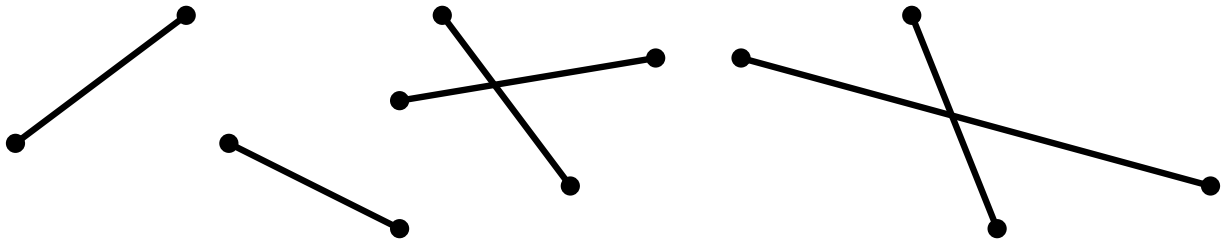
Schnitte von Strecken – Sweep-Line

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Schnitte von Strecken

Problem: Schnitte von Strecken
Gegeben seien n Strecken. Schneiden sich zwei dieser Strecken?

Einfache Lösung:
Teste für jedes Streckenpaar ob sie sich schneiden. $\rightarrow \binom{n}{2} \in O(n^2)$ Paare.

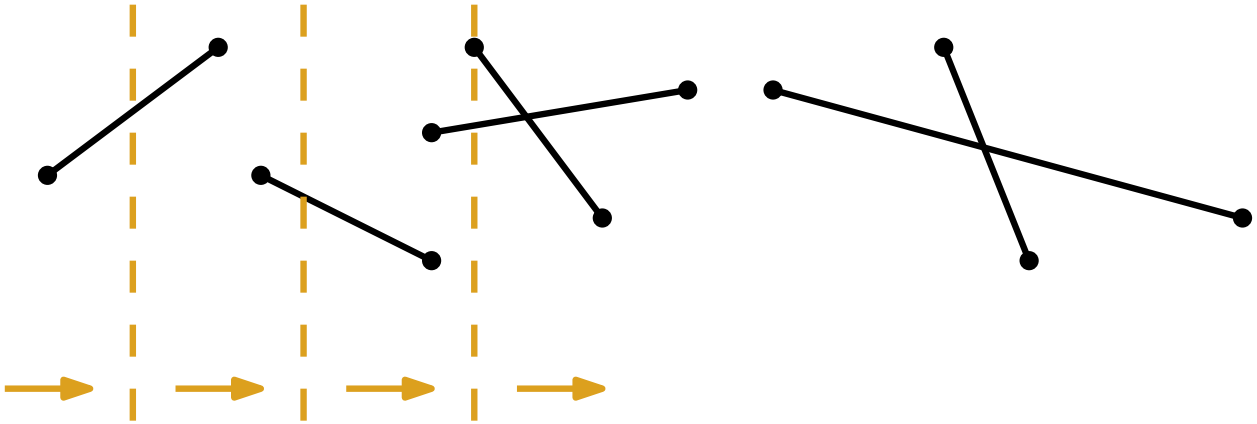


Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Schnitte von Strecken

Problem: Schnitte von Strecken
Gegeben seien n Strecken. Schneiden sich zwei dieser Strecken?

Einfache Lösung:
Teste für jedes Streckenpaar ob sie sich schneiden. $\rightarrow \binom{n}{2} \in O(n^2)$ Paare.



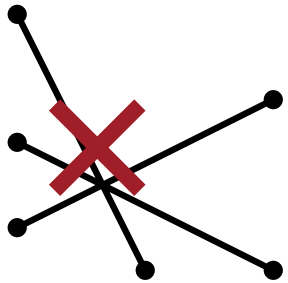
Heute:
Sweep-Line-Algorithmus mit Laufzeit $O(n \log n)$.

Definition: Sweep-Line **(Definition 6.6)**
Eine *Sweep-Line* ist eine imaginäre vertikale Linie, die eine Menge geometrischer Objekte von links nach rechts passiert.
Eine Sweep-Line ist durch ihre x-Koordinate eindeutig bestimmt.

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Vergleichbare Strecken

Wir machen folgende Annahmen:

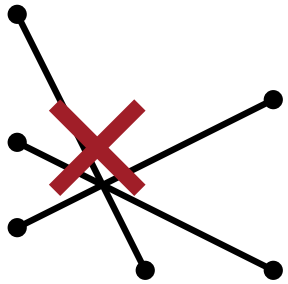


- Keine der Strecken ist vertikal.
- In jedem Punkt schneiden sich höchstens zwei Strecken.

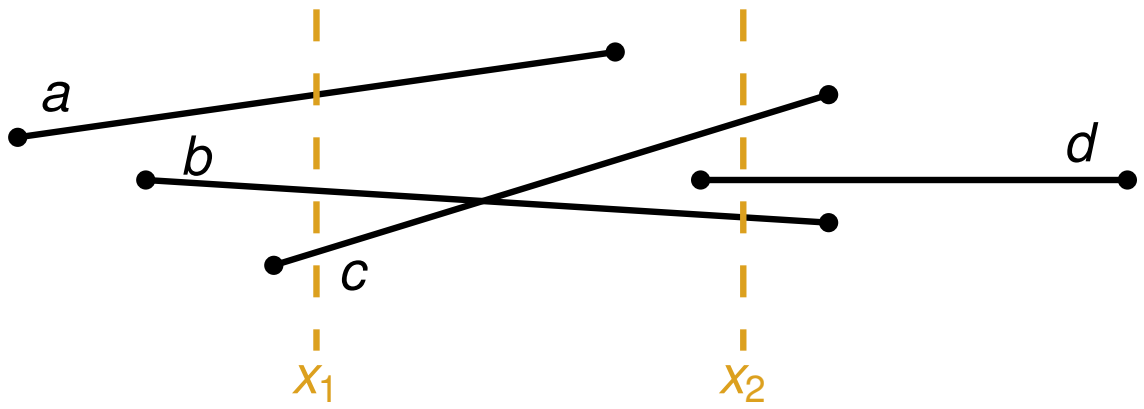
Vergleichbare Strecken

Wir machen folgende Annahmen:

- Keine der Strecken ist vertikal.
- In jedem Punkt schneiden sich höchstens zwei Strecken.



Definition: Vergleichbare Strecken
Zwei Strecken s_1 und s_2 sind bezüglich Sweep-Line x *vergleichbar*, falls beide x schneiden und sich nicht gegenseitig an der Stelle x schneiden.
Es gilt dann: $s_1 <_x s_2$ ($s_2 <_x s_1$) falls s_1 in x unterhalb (oberhalb) von s_2 liegt.



$c <_{x_1} b <_{x_1} a$

$b <_{x_2} d <_{x_2} c$

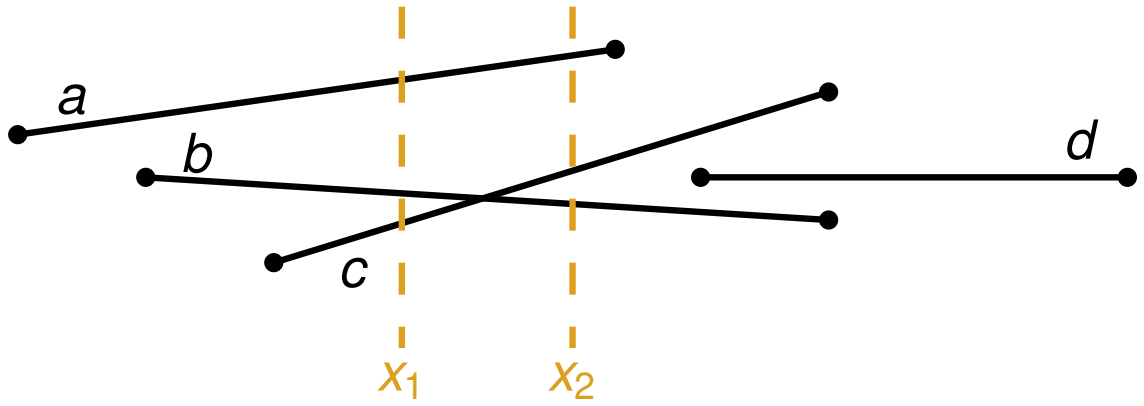
Für jedes x ist $<_x$ eine totale Ordnung auf den Strecken, die x schneiden und sich nicht bei x gegenseitig schneiden.

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Sich schneidende Strecken

Beobachtung: Sich schneidende Strecken

- Für zwei Strecken s_1, s_2 die sich schneiden gibt es Sweep-Lines x_1 und x_2, sodass $s_1 <_{x_1} s_2$ und $s_2 <_{x_2} s_1$.
- x_1 und x_2 können sogar so gewählt werden, dass s_1 und s_2 bezgl. $<_{x_1}$ und $<_{x_2}$ direkt aufeinanderfolgen.



$c <_{x_1} b <_{x_1} a$

$b <_{x_2} c <_{x_2} a$

Folgerung

Es ist ausreichend, alle Streckenpaare auf Schnitt zu prüfen, die bezüglich $<_x$ irgendeiner Sweep-Line x direkt aufeinanderfolgen.

Sweep-Line Algorithmus

Sweep-Line Algorithmen verwenden typischerweise zwei Datenmengen:

1. **Sweep-Line Zustand:** Objekte (und ggf. Beziehungen zwischen ihnen), die von der aktuellen Sweep-Line geschnitten werden.
2. **Event-Point Schedule:** Folge sortierter x-Koordinaten; die „Haltepunkte“ (Event-Points) der Sweep-Line. Der Sweep-Line Zustand ändert sich nur an solchen Event-Points.

Sweep-Line Algorithmus

Sweep-Line Algorithmen verwenden typischerweise zwei Datenmengen:

1. **Sweep-Line Zustand:** Objekte (und ggf. Beziehungen zwischen ihnen), die von der aktuellen Sweep-Line geschnitten werden.
2. **Event-Point Schedule:** Folge sortierter x-Koordinaten; die „Haltepunkte“ (Event-Points) der Sweep-Line. Der Sweep-Line Zustand ändert sich nur an solchen Event-Points.

Für unser Problem:

1. **Sweep-Line Zustand:** Alle Strecken, die die aktuelle Sweep-Line schneiden, von unten nach oben sortiert.
2. **Event-Point Schedule:** Die Endpunkte aller Strecken, von links nach rechts sortiert.

Events: An jedem Event-Point taucht eine neue Strecke auf oder eine schon vorhandene verschwindet. Folgendes muss getan werden:

- Neue Strecke in den Sweep-Line Zustand einfügen bzw. alte herauslöschen.
- Überprüfen ob sich neuerdings benachbarte Streckenpaare schneiden.

Sweep-Line Algorithmus

INTERSECT(S)

$T \leftarrow \emptyset$ — Sweep-Line Zustand

Sortiere Endpunkte der Strecken von links nach rechts — Event-Point Schedule

if *zwei Endpunkte sind gleich* **then** gib TRUE zurück

foreach *Endpunkt p in der sortierten Liste* **do**

- **if** *p linker Endpunkt von Strecke s* **then**
  - INSERT(T, s)
  - **if** *s schneidet* ABOVE(T, s) **then** gib TRUE zurück
  - **if** *s schneidet* BELOW(T, s) **then** gib TRUE zurück
- **if** *p rechter Endpunkt von Strecke s* **then**
  - **if** ABOVE(T, s) *schneidet* BELOW(T, s) **then** gib TRUE zurück
  - DELETE(T, s)

gib FALSE zurück

(ABOVE(T, s) (BELOW(T, s)) ist die Strecke unmittelbar über (unter) s in T)

Sweep-Line Algorithmus

INTERSECT(S)

$T \leftarrow \emptyset$	Sweep-Line Zustand	$O(1)$
Sortiere Endpunkte der Strecken von links nach rechts	Event-Point Schedule	$O(n \log n)$
if *zwei Endpunkte sind gleich* **then** gib TRUE zurück		$O(n)$

foreach *Endpunkt p in der sortierten Liste* **do**
- **if** *p linker Endpunkt von Strecke s* **then**
  - INSERT(T, s)
  - **if** *s schneidet* ABOVE(T, s) **then** gib TRUE zurück
  - **if** *s schneidet* BELOW(T, s) **then** gib TRUE zurück
- **if** *p rechter Endpunkt von Strecke s* **then**
  - **if** ABOVE(T, s) *schneidet* BELOW(T, s) **then** gib TRUE zurück
  - DELETE(T, s)

gib FALSE zurück

(ABOVE(T, s) (BELOW(T, s)) ist die Strecke unmittelbar über (unter) s in T)

Sweep-Line Algorithmus

INTERSECT(S)

$T \leftarrow \emptyset$	Sweep-Line Zustand	$O(1)$
Sortiere Endpunkte der Strecken von links nach rechts	Event-Point Schedule	$O(n \log n)$
if *zwei Endpunkte sind gleich* **then** gib TRUE zurück		$O(n)$

foreach *Endpunkt p in der sortierten Liste* **do**

- **if** *p linker Endpunkt von Strecke s* **then**
  - INSERT(T, s)
  - **if** *s schneidet* ABOVE(T, s) **then** gib TRUE zurück
  - **if** *s schneidet* BELOW(T, s) **then** gib TRUE zurück
- **if** *p rechter Endpunkt von Strecke s* **then**
  - **if** ABOVE(T, s) *schneidet* BELOW(T, s) **then** gib TRUE zurück
  - DELETE(T, s)

gib FALSE zurück

(ABOVE(T, s) (BELOW(T, s)) ist die Strecke unmittelbar über (unter) s in T)

Benutze für T einen AVL-Baum (oder ähnliches): Operationen INSERT(T, s), DELETE(T, s), ABOVE(T, s) und BELOW(T, s) benötigen $O(\log n)$ Zeit.

Sweep-Line Algorithmus

INTERSECT(S)

$T \leftarrow \emptyset$ — Sweep-Line Zustand — $O(1)$

Sortiere Endpunkte der Strecken von links nach rechts — Event-Point Schedule — $O(n \log n)$

if *zwei Endpunkte sind gleich* **then** gib TRUE zurück — $O(n)$

foreach *Endpunkt p in der sortierten Liste* **do**

- **if** *p linker Endpunkt von Strecke s* **then** — $O(\log n)$
  - INSERT(T, s)
  - **if** *s schneidet* ABOVE(T, s) **then** gib TRUE zurück
  - **if** *s schneidet* BELOW(T, s) **then** gib TRUE zurück
- **if** *p rechter Endpunkt von Strecke s* **then** — $O(\log n)$
  - **if** ABOVE(T, s) *schneidet* BELOW(T, s) **then** gib TRUE zurück
  - DELETE(T, s)

gib FALSE zurück

(ABOVE(T, s) (BELOW(T, s)) ist die Strecke unmittelbar über (unter) s in T)

Benutze für T einen AVL-Baum (oder ähnliches): Operationen INSERT(T, s), DELETE(T, s), ABOVE(T, s) und BELOW(T, s) benötigen $O(\log n)$ Zeit.

Sweep-Line Algorithmus

INTERSECT(S)

$T \leftarrow \emptyset$	Sweep-Line Zustand	$O(1)$
Sortiere Endpunkte der Strecken von links nach rechts	Event-Point Schedule	$O(n \log n)$
if *zwei Endpunkte sind gleich* **then** gib TRUE zurück		$O(n)$
foreach *Endpunkt p in der sortierten Liste* **do**		$O(n \log n)$
if *p linker Endpunkt von Strecke s* **then**		$O(\log n)$
INSERT(T, s)		
if *s schneidet* ABOVE(T, s) **then** gib TRUE zurück		
if *s schneidet* BELOW(T, s) **then** gib TRUE zurück		
if *p rechter Endpunkt von Strecke s* **then**		$O(\log n)$
if ABOVE(T, s) *schneidet* BELOW(T, s) **then** gib TRUE zurück		
DELETE(T, s)		
gib FALSE zurück		$O(1)$

(ABOVE(T, s) (BELOW(T, s)) ist die Strecke unmittelbar über (unter) s in T)

Benutze für T einen AVL-Baum (oder ähnliches): Operationen INSERT(T, s), DELETE(T, s), ABOVE(T, s) und BELOW(T, s) benötigen $O(\log n)$ Zeit.

Sweep-Line Algorithmus

INTERSECT(S) $O(n \log n)$

- $T \leftarrow \emptyset$ — Sweep-Line Zustand — $O(1)$
- Sortiere Endpunkte der Strecken von links nach rechts — Event-Point Schedule — $O(n \log n)$
- **if** *zwei Endpunkte sind gleich* **then** gib TRUE zurück — $O(n)$
- **foreach** *Endpunkt p in der sortierten Liste* **do** — $O(n \log n)$
  - **if** *p linker Endpunkt von Strecke s* **then** — $O(\log n)$
    - INSERT(T, s)
    - **if** *s schneidet* ABOVE(T, s) **then** gib TRUE zurück
    - **if** *s schneidet* BELOW(T, s) **then** gib TRUE zurück
  - **if** *p rechter Endpunkt von Strecke s* **then** — $O(\log n)$
    - **if** ABOVE(T, s) *schneidet* BELOW(T, s) **then** gib TRUE zurück
    - DELETE(T, s)
- gib FALSE zurück — $O(1)$

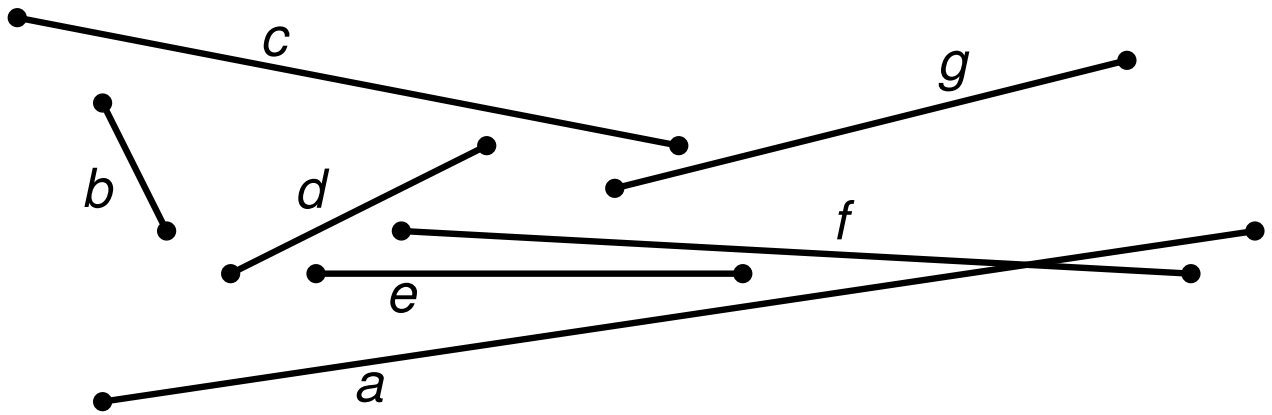
(ABOVE(T, s) (BELOW(T, s)) ist die Strecke unmittelbar über (unter) s in T)

Benutze für T einen AVL-Baum (oder ähnliches): Operationen INSERT(T, s), DELETE(T, s), ABOVE(T, s) und BELOW(T, s) benötigen $O(\log n)$ Zeit.

$\Rightarrow$ Gesamtlaufzeit von $O(n \log n)$

Sweep-Line Algorithmus – Beispiel

Sweep-Line Zustand

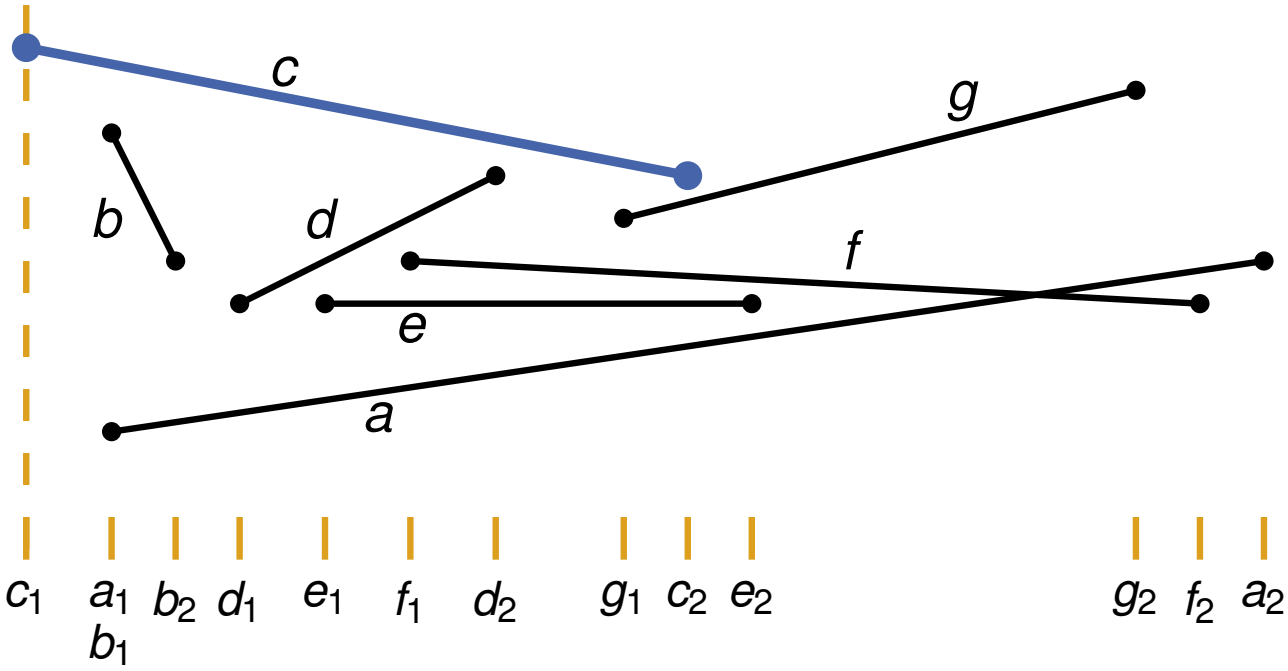


Event-Point Schedule

Sweep-Line Algorithmus – Beispiel

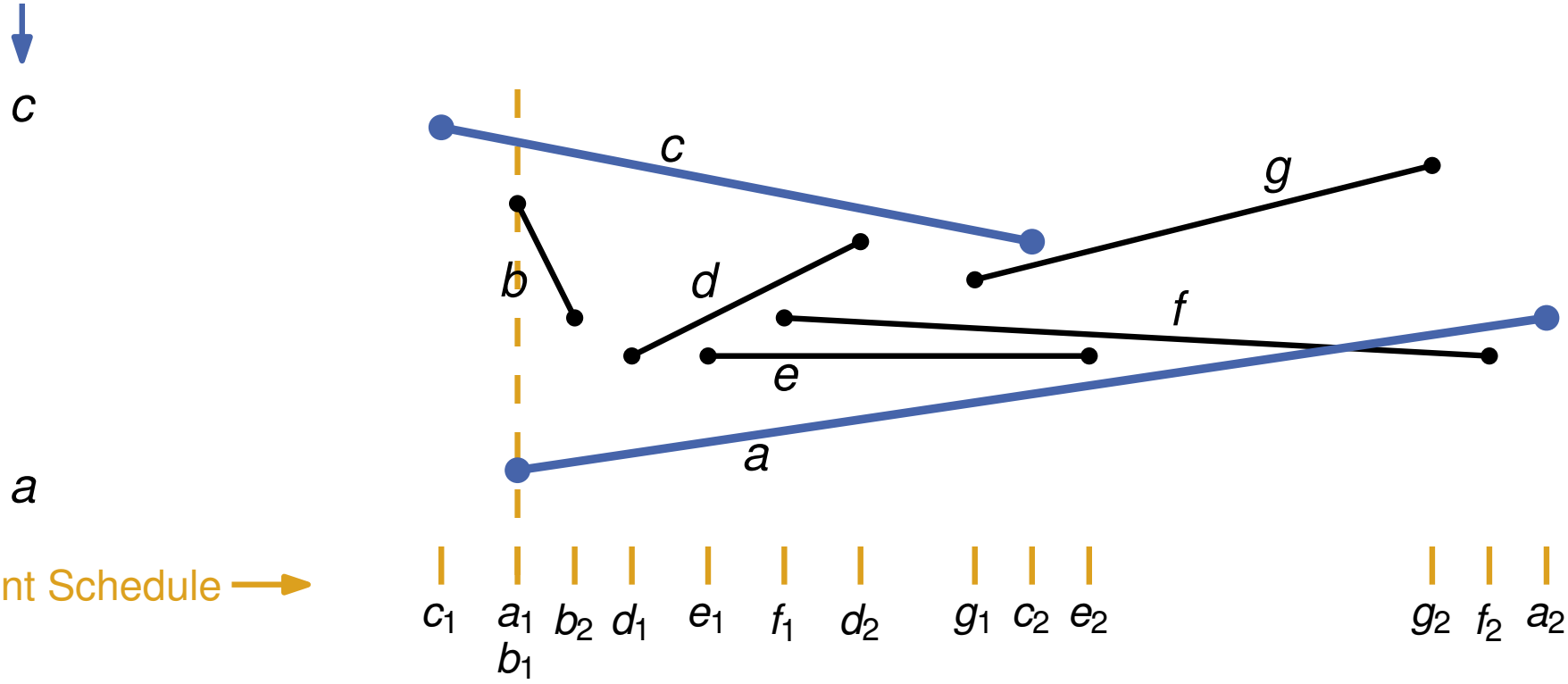
Sweep-Line Zustand

+ c



Sweep-Line Algorithmus – Beispiel

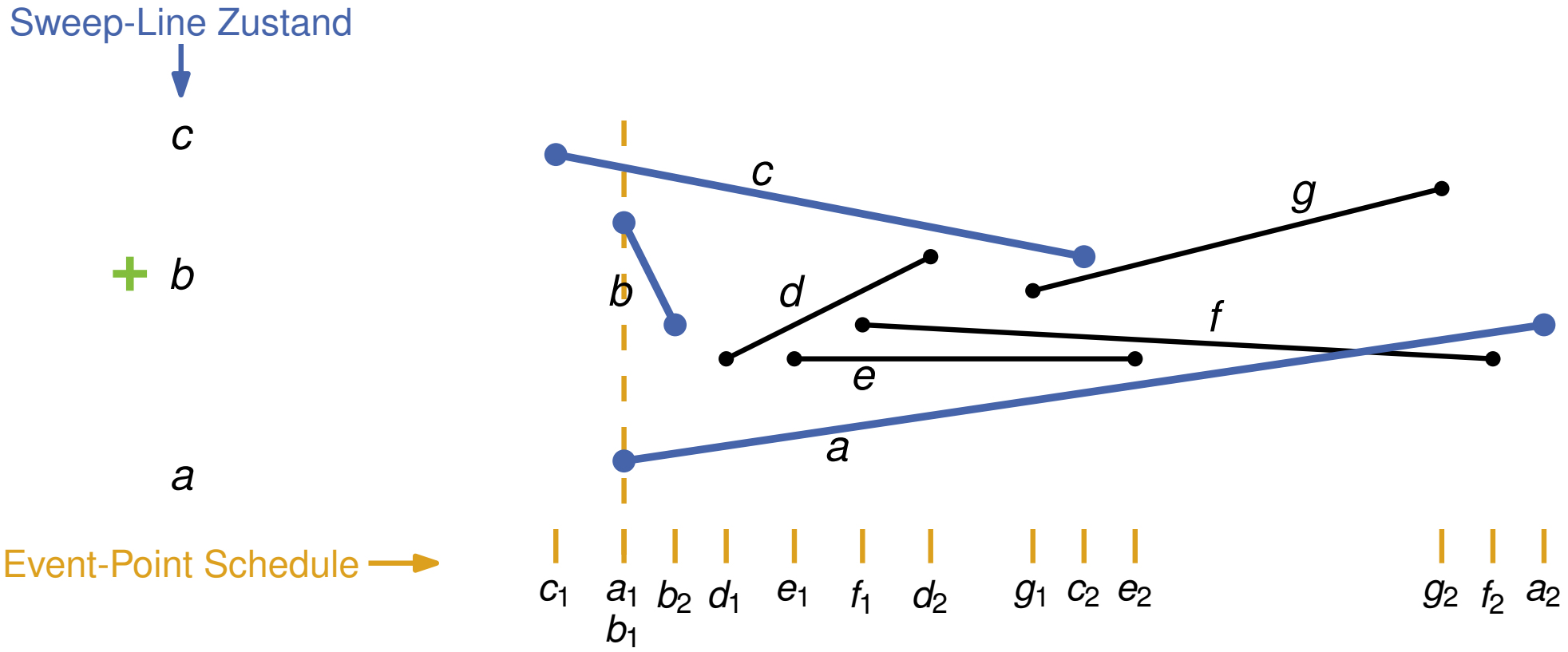
Sweep-Line Zustand



a schneidet *c* nicht

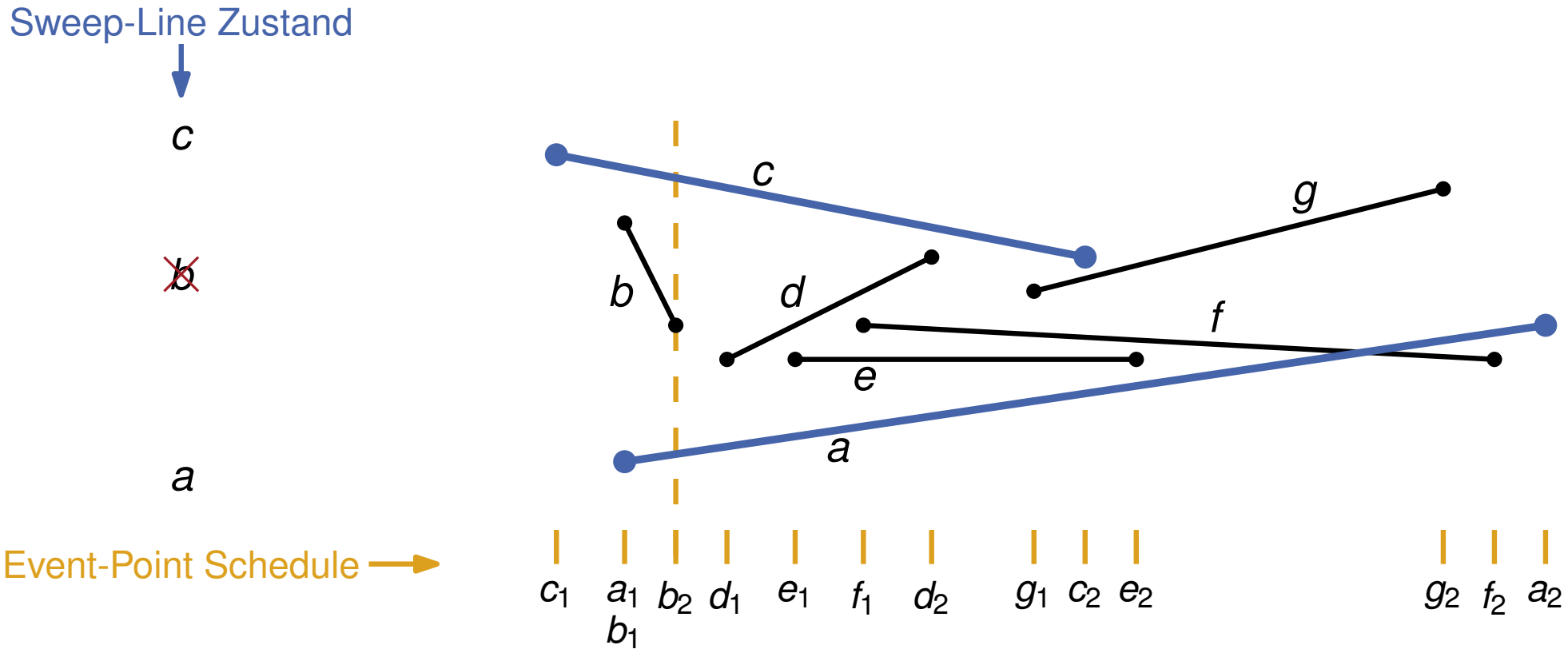
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Sweep-Line Algorithmus – Beispiel



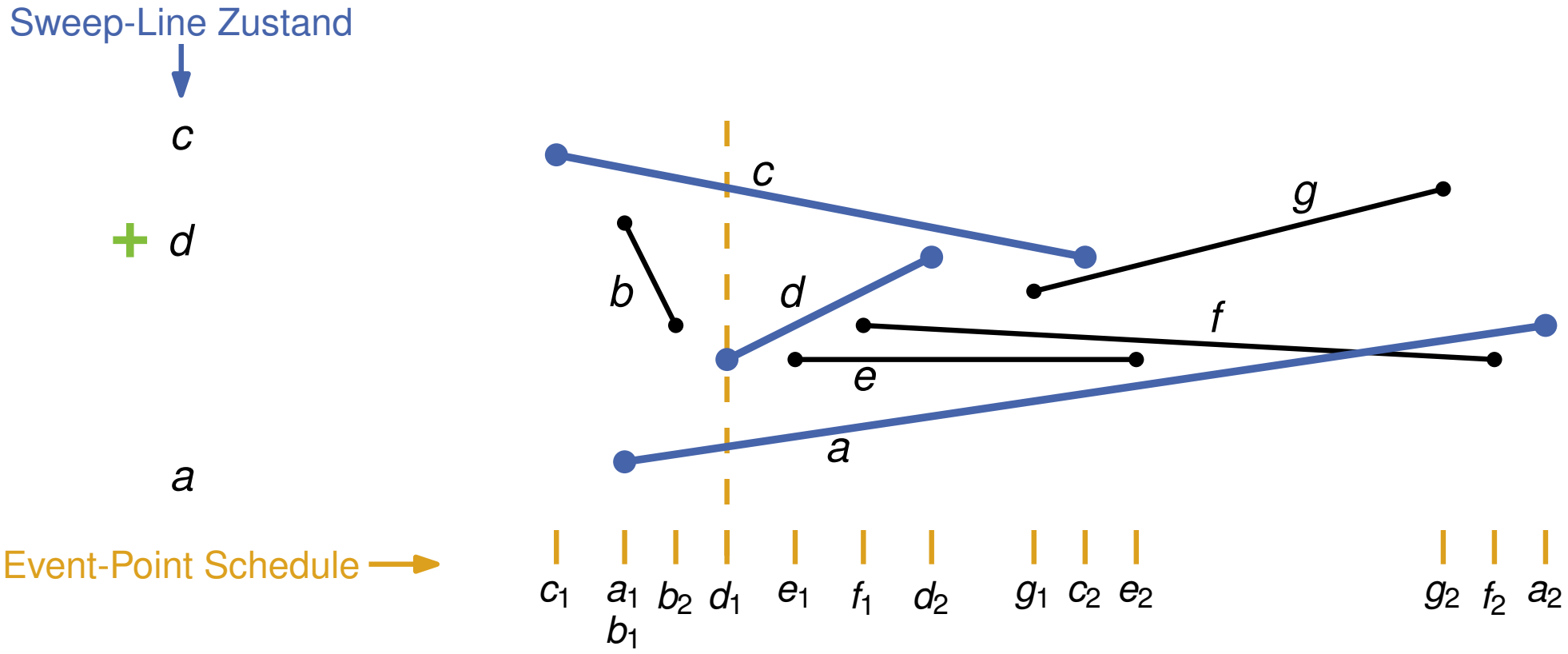
b schneidet weder *a* noch *c*

Sweep-Line Algorithmus – Beispiel



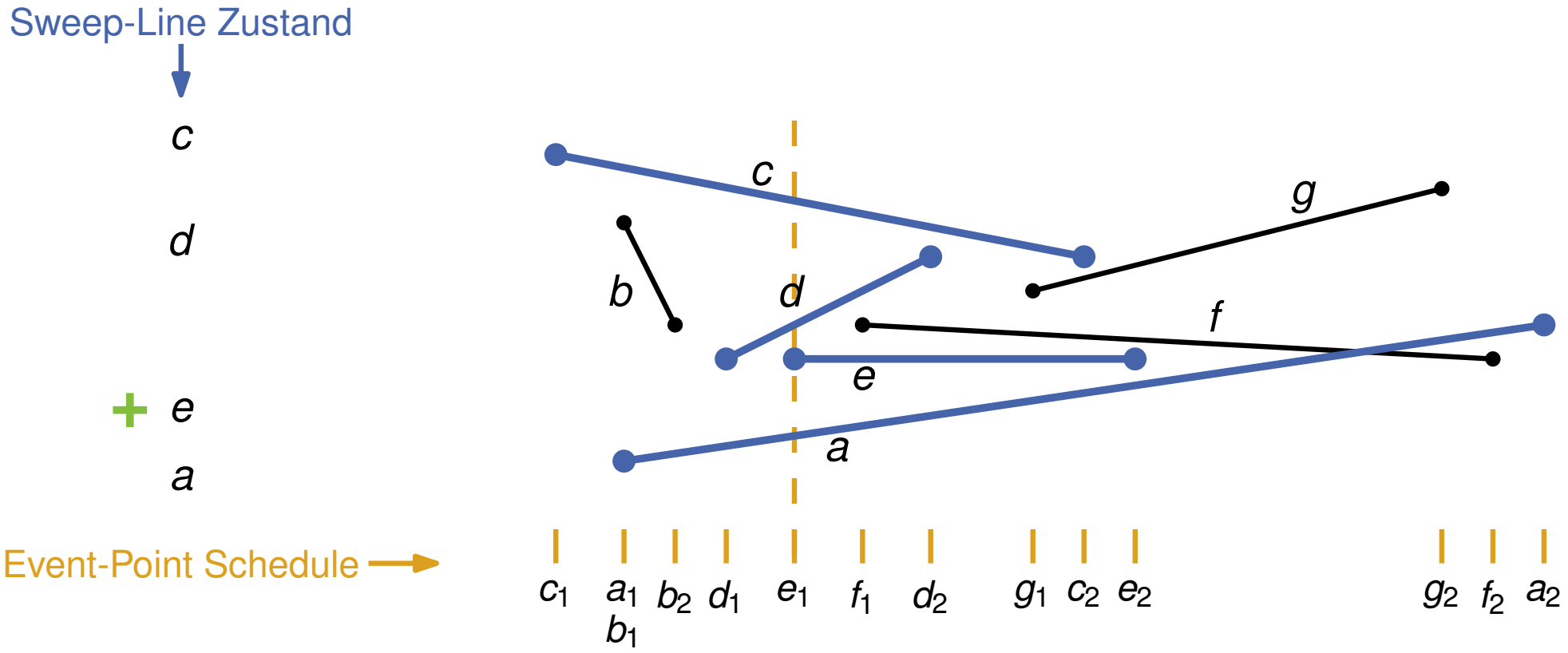
a schneidet *c* nicht

Sweep-Line Algorithmus – Beispiel



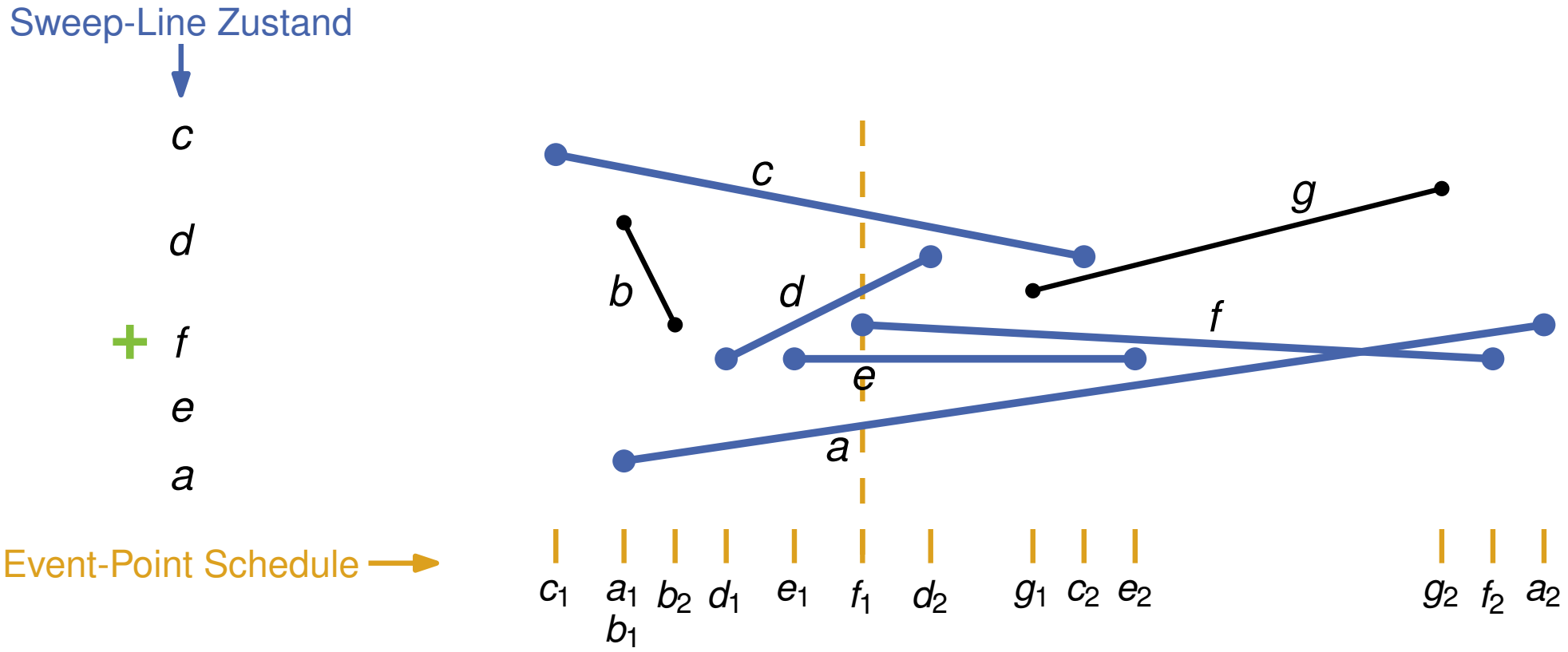
d schneidet weder *a* noch *c*

Sweep-Line Algorithmus – Beispiel



e schneidet weder *d* noch *a*

Sweep-Line Algorithmus – Beispiel

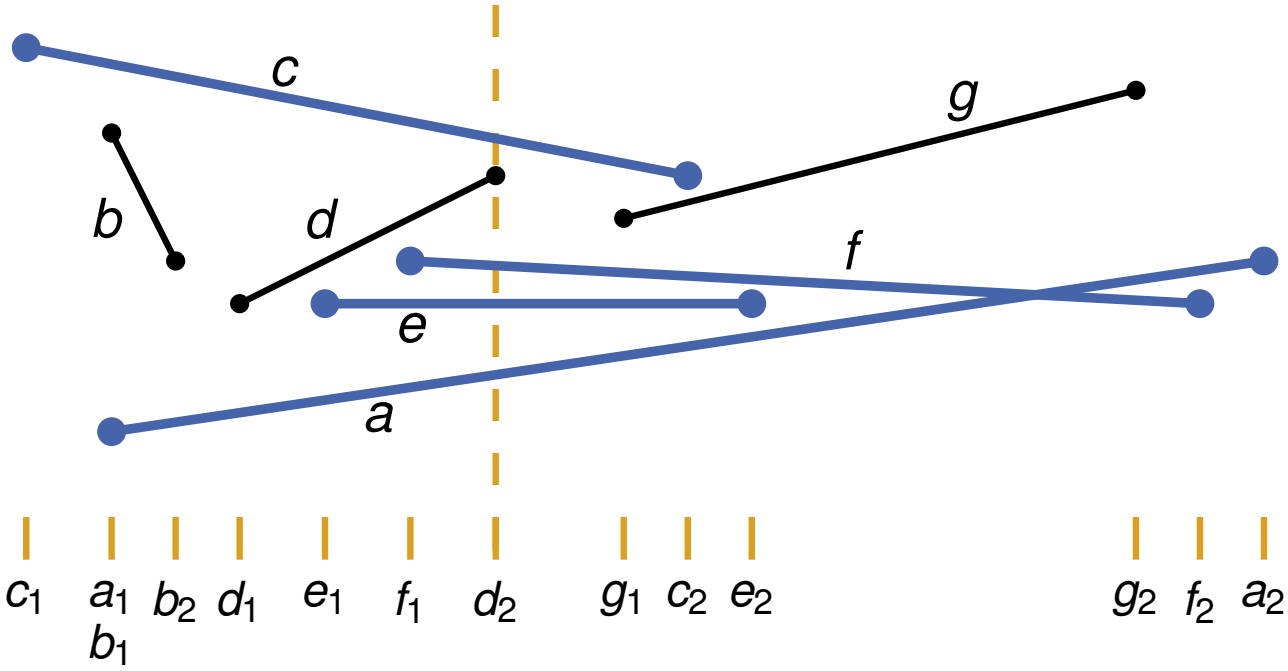


f schneidet weder *d* noch *e*

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Sweep-Line Algorithmus – Beispiel

Sweep-Line Zustand



Event-Point Schedule →

c_1 a_1 b_2 d_1 e_1 f_1 d_2 g_1 c_2 e_2 g_2 f_2 a_2
b_1

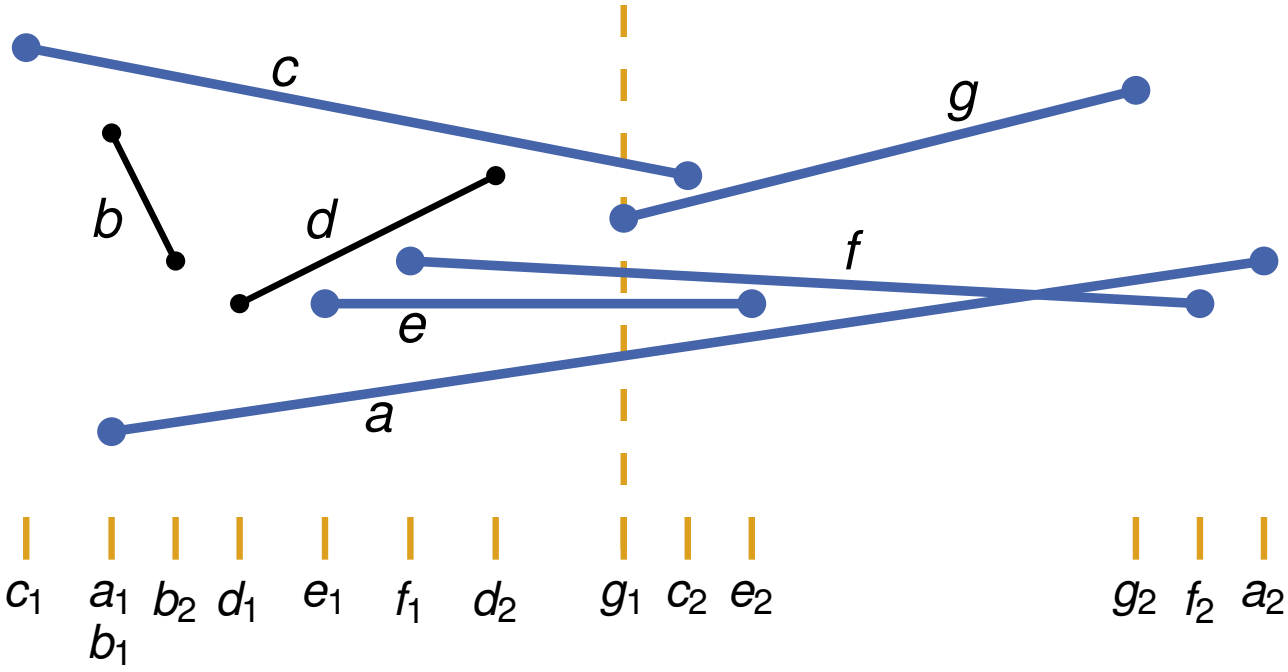
f schneidet *c* nicht

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Sweep-Line Algorithmus – Beispiel

Sweep-Line Zustand

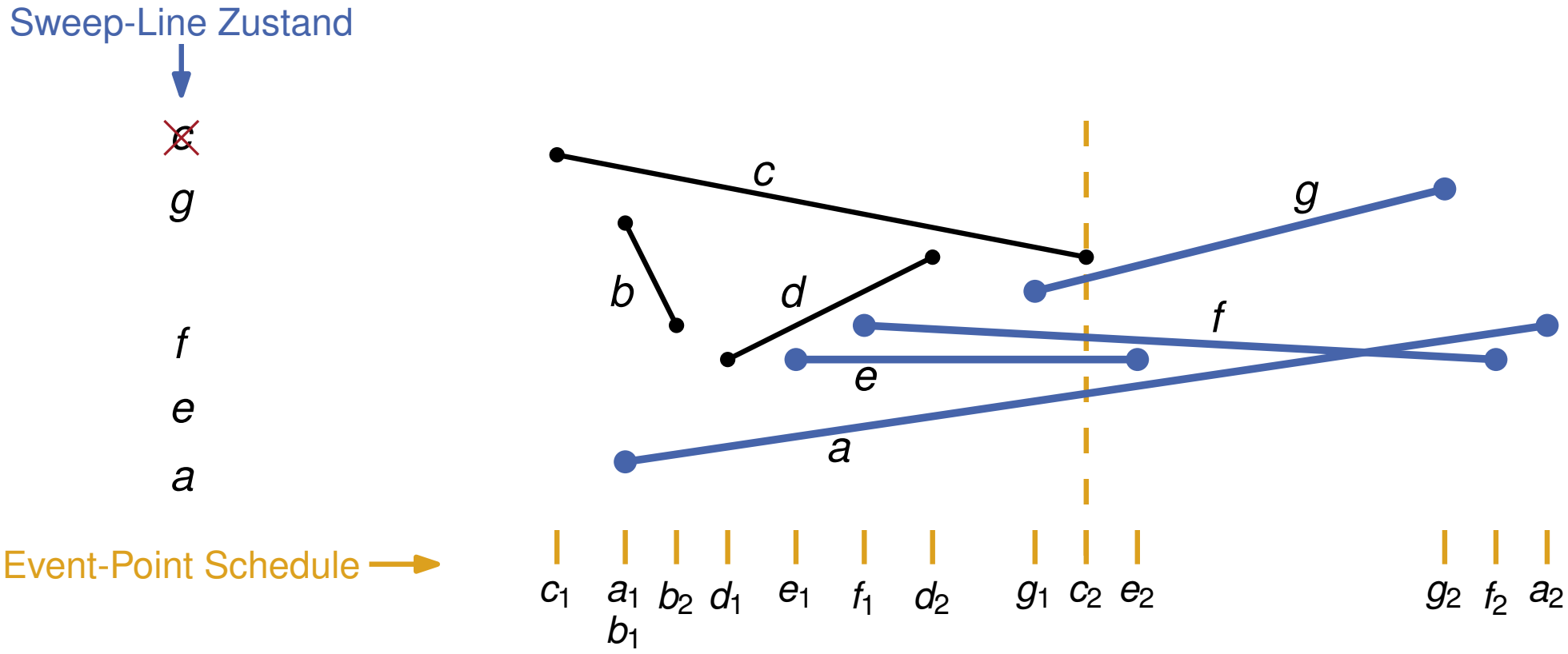
Event-Point Schedule →



g schneidet weder *f* noch *c*

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

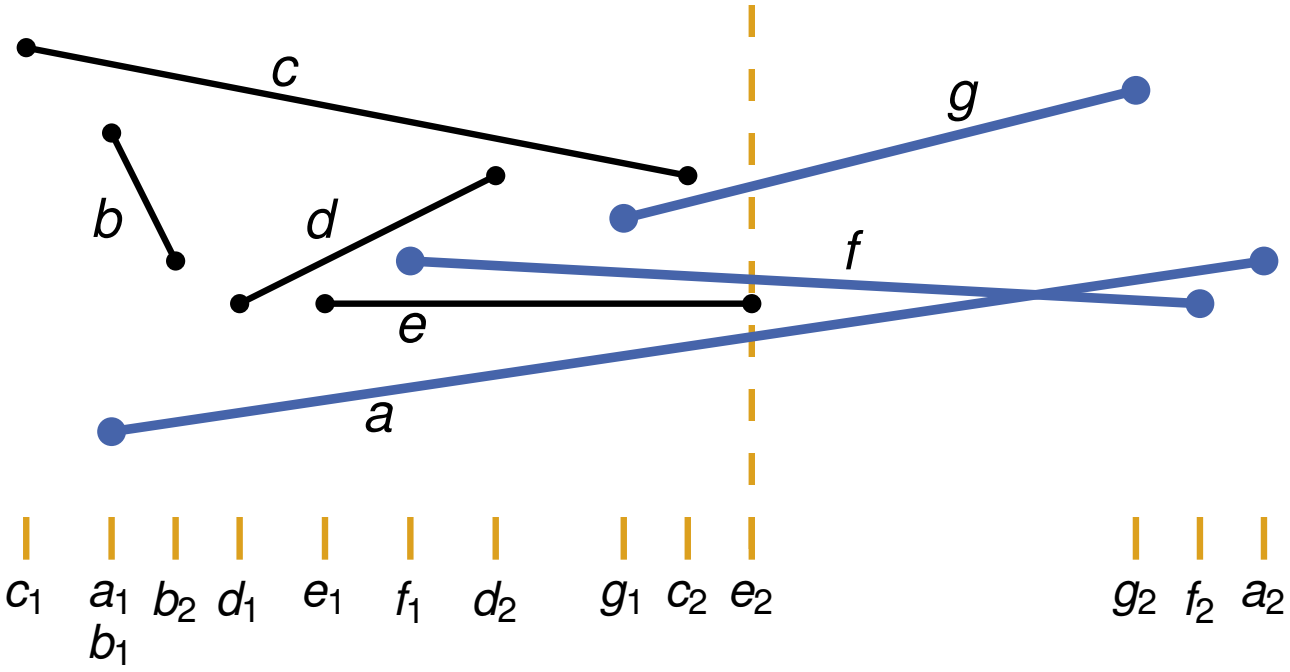
Sweep-Line Algorithmus – Beispiel



Sweep-Line Algorithmus – Beispiel

Sweep-Line Zustand

Event-Point Schedule



f schneidet a! $\Rightarrow$ gib TRUE zurück

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Anmerkungen

- Der vorgestellte Algorithmus prüft nur, ob es mindestens eine Kreuzungen gibt und kann die erste gefundene Kreuzung ausgeben.

- Man kann den Algorithmus leicht anpassen um alle Kreuzungen zu finden. Man benötigt dann $O(n \log n + k \log n)$ Zeit, wobei k die Anzahl an Kreuzungen ist. **Beachte:** $k \in O(n^2)$.

- Es gibt auch einen Algorithmus mit Laufzeit $O(n \log n + k)$, der alle Kreuzungen findet.

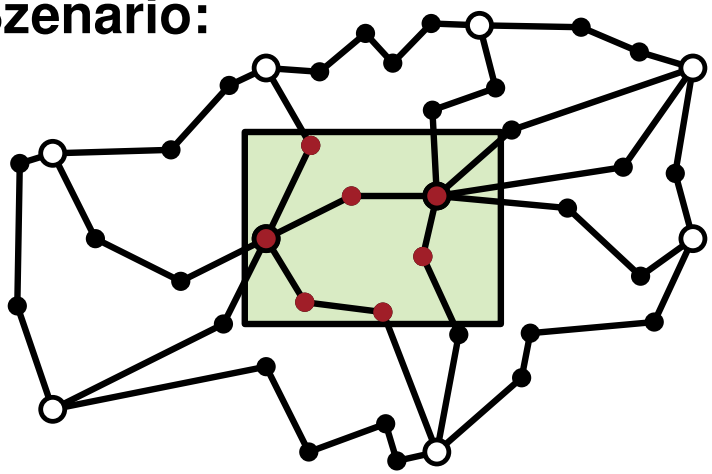
Karlsruher Institut für Technologie

Eine geometrische Datenstruktur – Quadtree

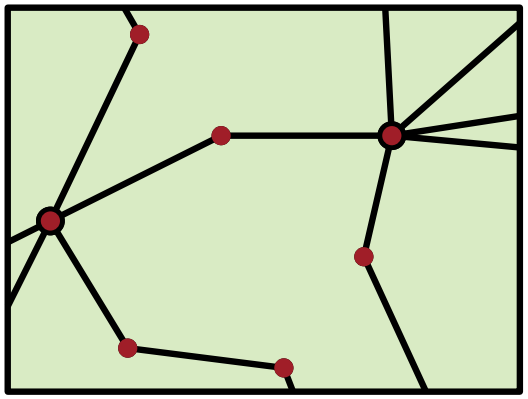
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Motivation

Szenario:



großer Straßengraph



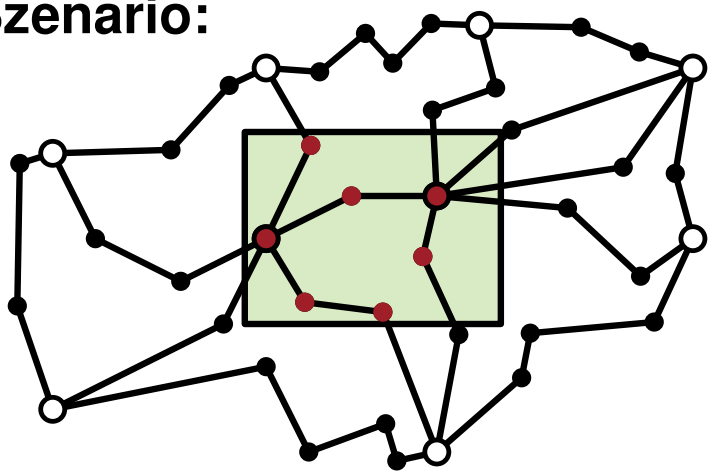
zeige nur einen kleinen Ausschnitt

Problem: Bereichsanfrage

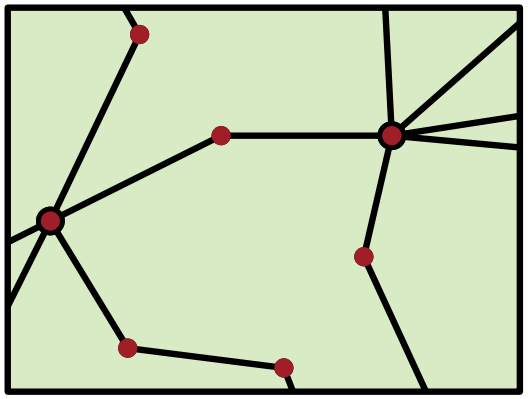
Gegeben eine Punktmenge P (z.B. die Knoten in einem Straßengraph) sowie ein Rechteck R, finde alle Punkte aus P, die in R liegen.

Motivation

Szenario:



großer Straßengraph



zeige nur einen kleinen Ausschnitt

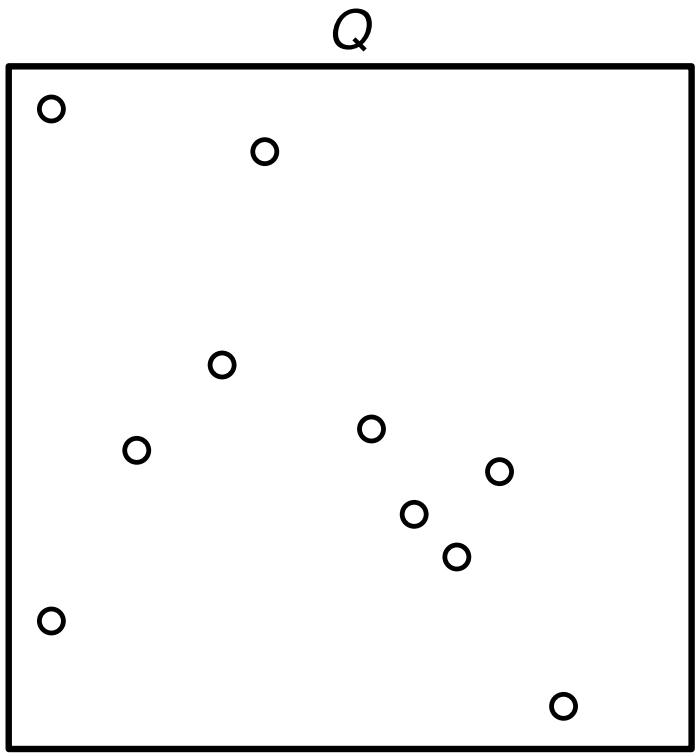
Problem: Bereichsanfrage
Gegeben eine Punktmenge P (z.B. die Knoten in einem Straßengraph) sowie ein Rechteck R, finde alle Punkte aus P, die in R liegen.

Einfache Lösung:
Teste für jeden Punkt $p \in P$, ob p in R liegt.
$\Rightarrow$ Laufzeit $O(|P|)$, dabei werden ggf. sehr viel weniger Punkte angezeigt.

Ziel im Folgenden:
Nutze aus, dass nur ein sehr kleiner Anteil der $|P|$ Punkte angezeigt werden muss.

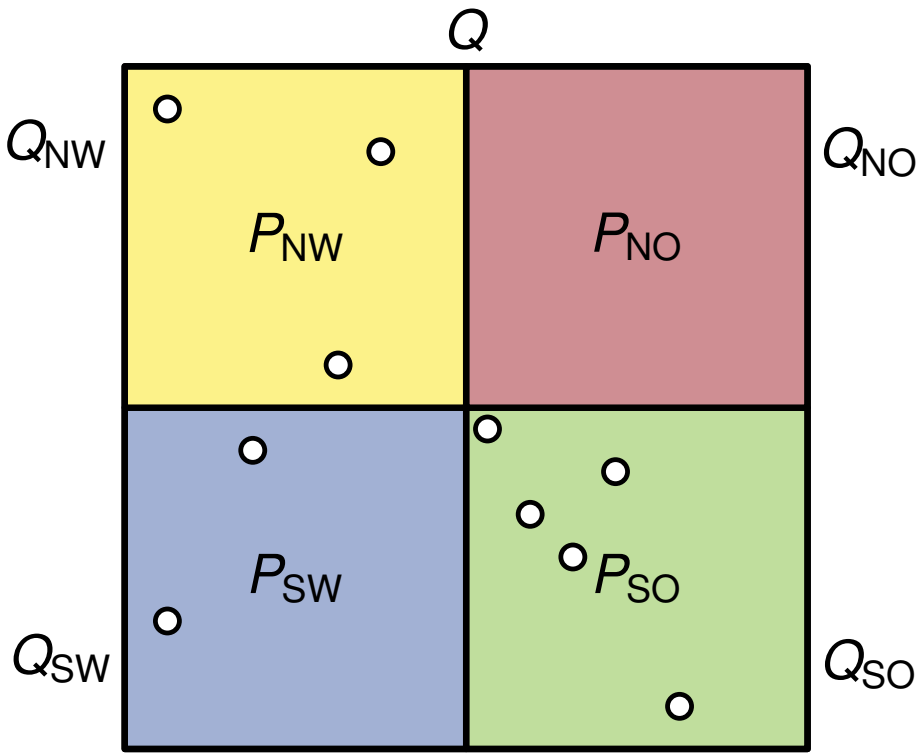
Hierarchische Unterteilung: Quadtree

Betrachte Punktmenge P in einem Quadrat Q.



Hierarchische Unterteilung: Quadtree

Betrachte Punktmenge P in einem Quadrat Q.

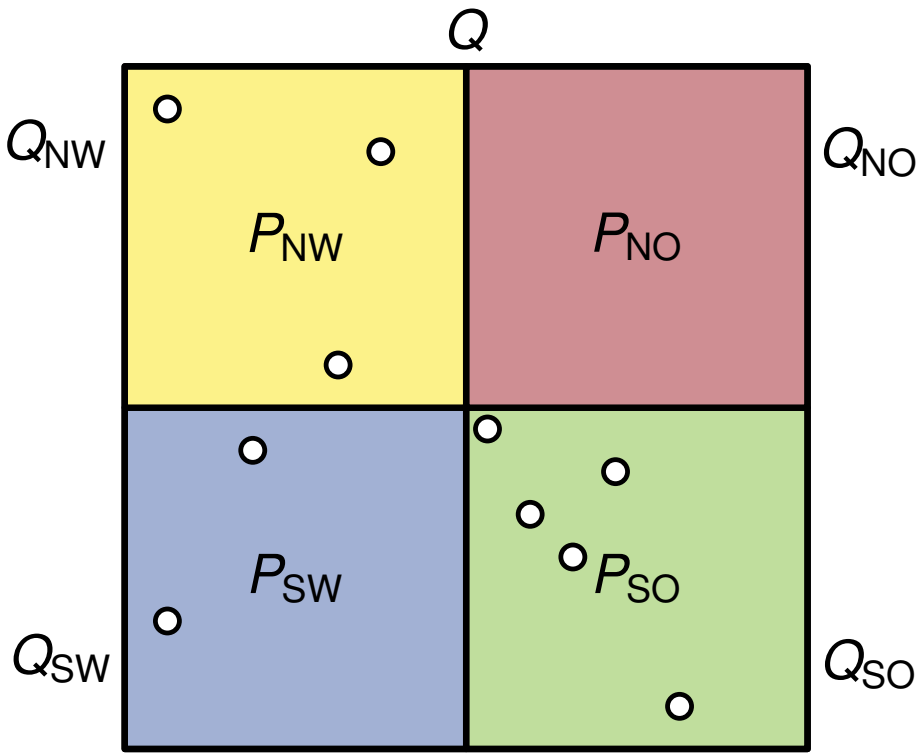


Wenn $|P| > 1$:

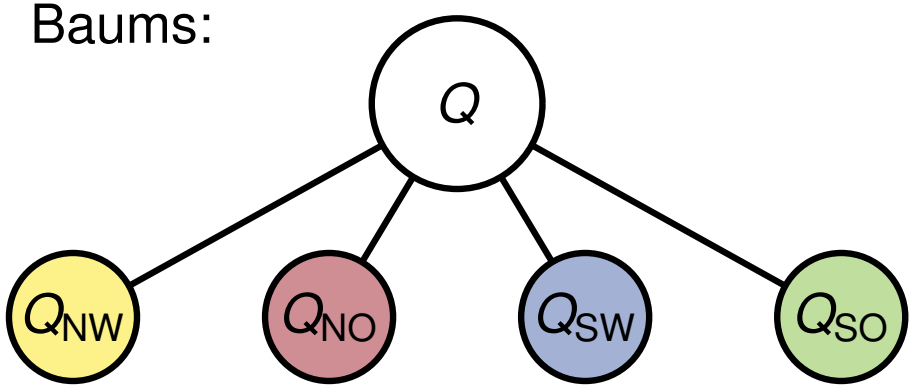
- Unterteile Q in vier Quadrate Q_{NW}, Q_{NO}, Q_{SW}, Q_{SO}.
- Zerlege P entsprechend in P_{NW}, P_{NO}, P_{SW}, P_{SO}, sodass P_i komplett in Q_i liegt (für $i \in \{NW, NO, SW, SO\}$).
- Fahre rekursiv mit P_i und Q_i fort (für $i \in \{NW, NO, SW, SO\}$).

Hierarchische Unterteilung: Quadtree

Betrachte Punktmenge P in einem Quadrat Q.



Interpretiere die Quadrate als Knoten eines Baums:

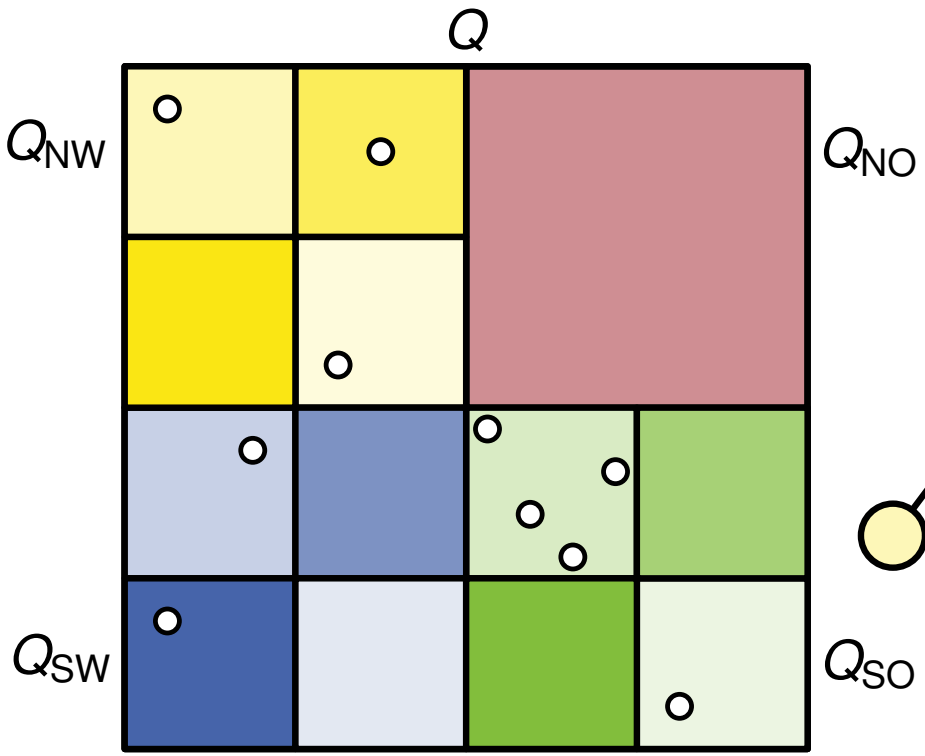


Wenn $|P| > 1$:

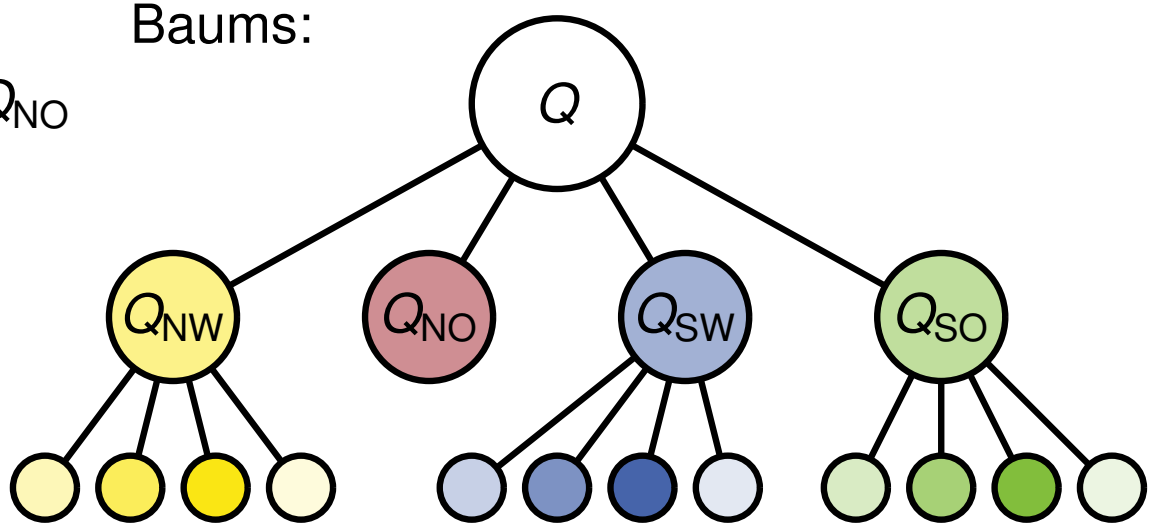
- Unterteile Q in vier Quadrate Q_{NW}, Q_{NO}, Q_{SW}, Q_{SO}.
- Zerlege P entsprechend in P_{NW}, P_{NO}, P_{SW}, P_{SO}, sodass P_i komplett in Q_i liegt (für $i \in \{NW, NO, SW, SO\}$).
- Fahre rekursiv mit P_i und Q_i fort (für $i \in \{NW, NO, SW, SO\}$).

Hierarchische Unterteilung: Quadtree

Betrachte Punktmenge P in einem Quadrat Q.



Interpretiere die Quadrate als Knoten eines Baums:



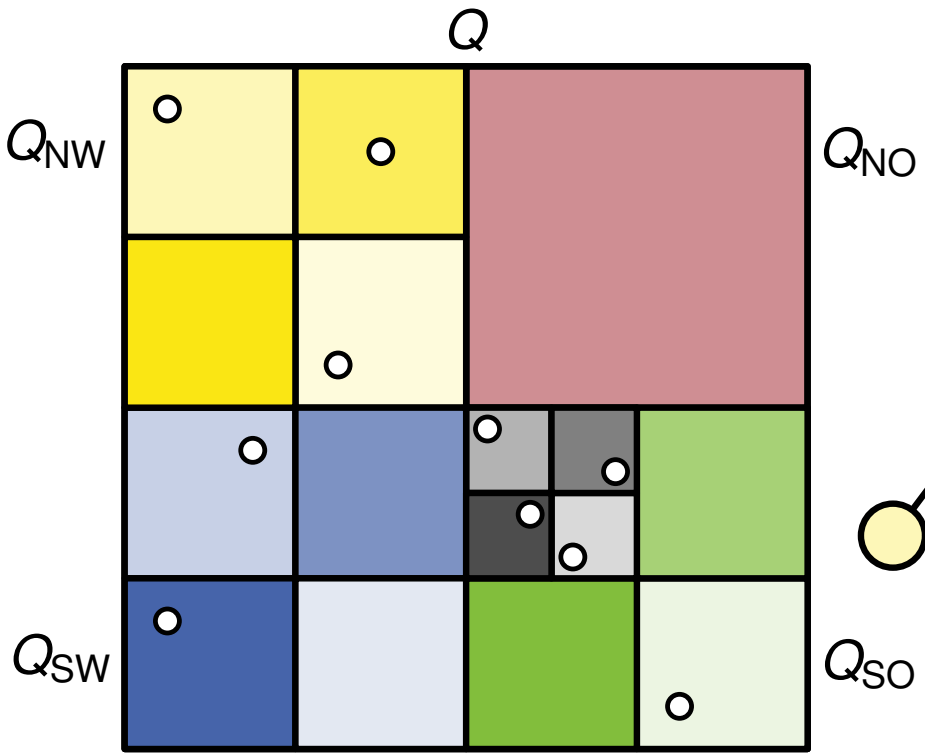
Wenn $|P| > 1$:

- Unterteile Q in vier Quadrate Q_{NW}, Q_{NO}, Q_{SW}, Q_{SO}.
- Zerlege P entsprechend in P_{NW}, P_{NO}, P_{SW}, P_{SO}, sodass P_i komplett in Q_i liegt (für $i \in \{NW, NO, SW, SO\}$).
- Fahre rekursiv mit P_i und Q_i fort (für $i \in \{NW, NO, SW, SO\}$).

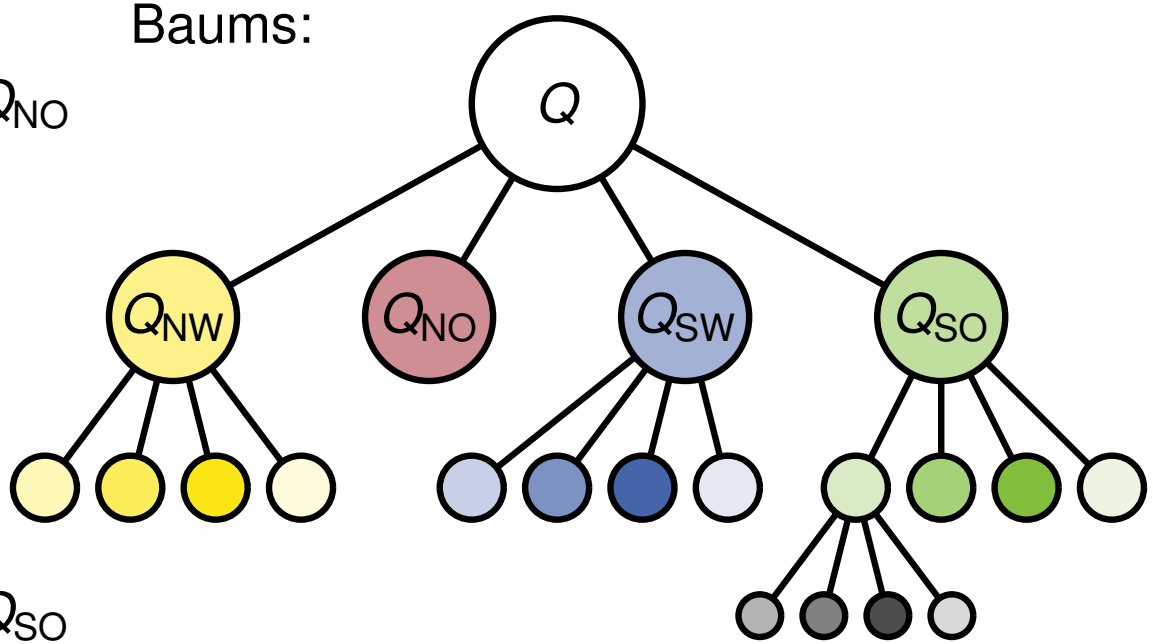
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Hierarchische Unterteilung: Quadtree

Betrachte Punktmenge P in einem Quadrat Q.



Interpretiere die Quadrate als Knoten eines Baums:



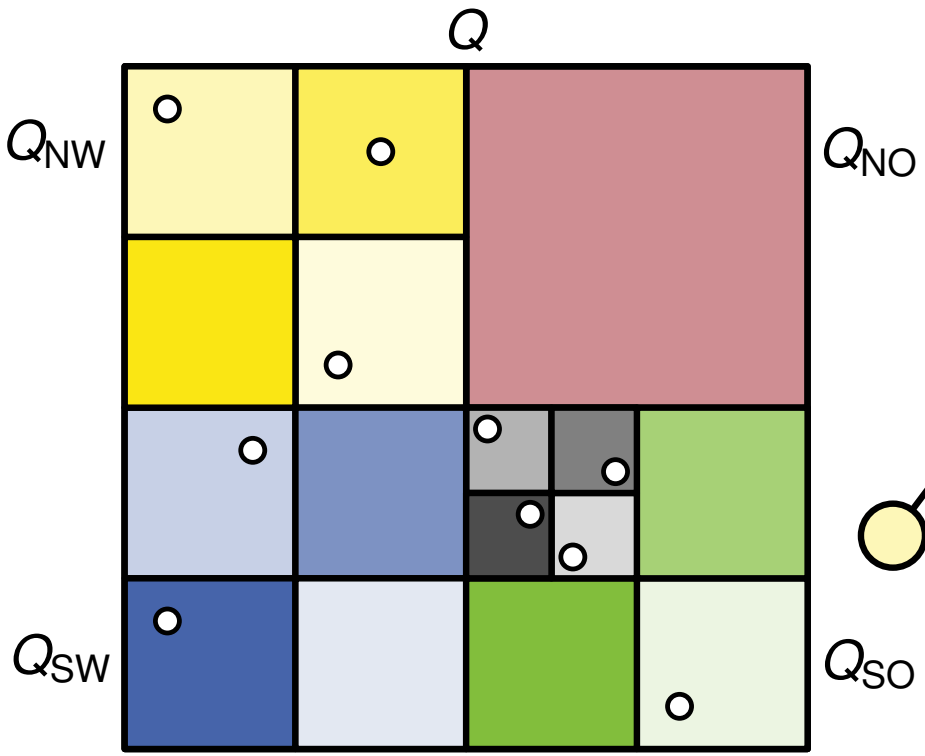
Wenn $|P| > 1$:

- Unterteile Q in vier Quadrate Q_{NW}, Q_{NO}, Q_{SW}, Q_{SO}.
- Zerlege P entsprechend in P_{NW}, P_{NO}, P_{SW}, P_{SO}, sodass P_i komplett in Q_i liegt (für $i \in \{NW, NO, SW, SO\}$).
- Fahre rekursiv mit P_i und Q_i fort (für $i \in \{NW, NO, SW, SO\}$).

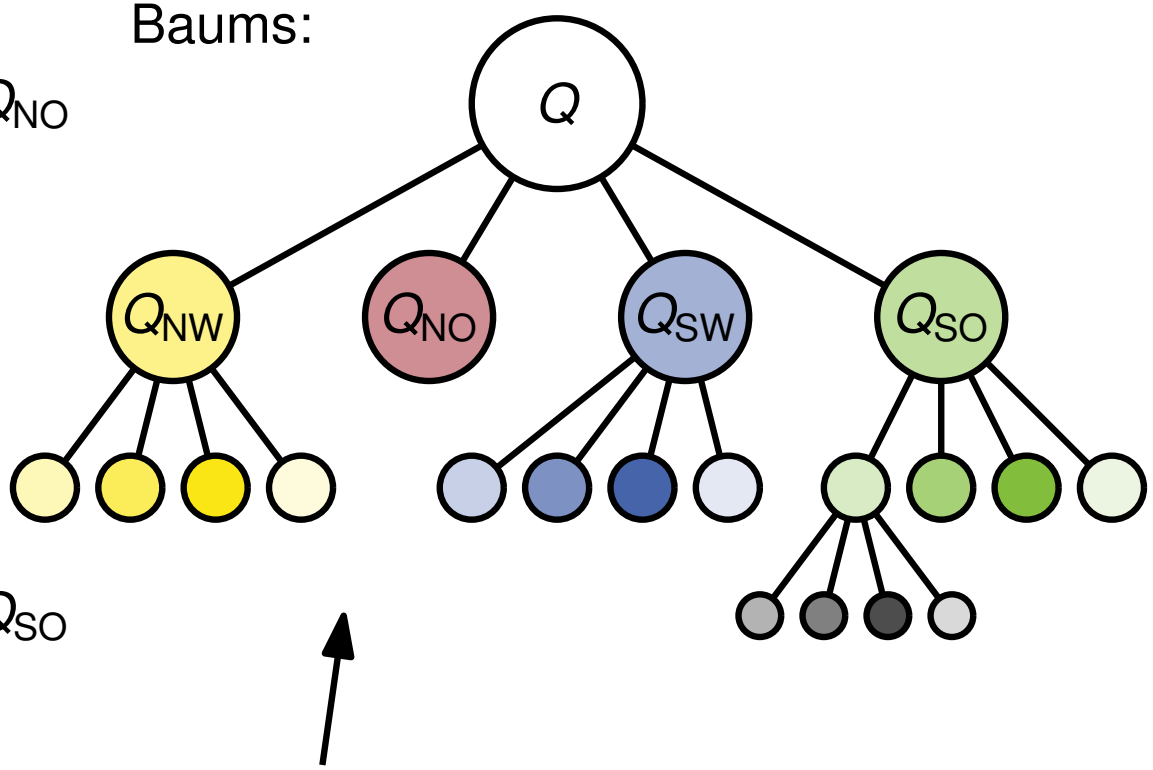
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Hierarchische Unterteilung: Quadtree

Betrachte Punktmenge P in einem Quadrat Q.



Interpretiere die Quadrate als Knoten eines Baums:



Quadtree $T(P)$ bezüglich Quadrat Q

Speichere zusätzlich für jedes Blatt in $T(P)$ den eventuell enthaltenen Knoten.

Aufbau des Quadtrees

QUADTREE(P, Q) (Rückgabe: Wurzel von $T(P)$ bezüglich Q)

$v_Q \leftarrow$ neuer Knoten, der Q repräsentiert

if $|P| > 1$ **then**

  foreach Q_i *mit* $i \in \{\text{NW}, \text{NO}, \text{SW}, \text{SO}\}$ **do**

    $P_i \leftarrow$ Punkte aus P, die in Q_i liegen

    $v_{Q_i} \leftarrow$ QUADTREE(P_i, Q_i)

    Füge v_{Q_i} als Kind von v_Q ein

else

  points(v_Q) $\leftarrow P$

return v_Q

Aufbau des Quadtrees

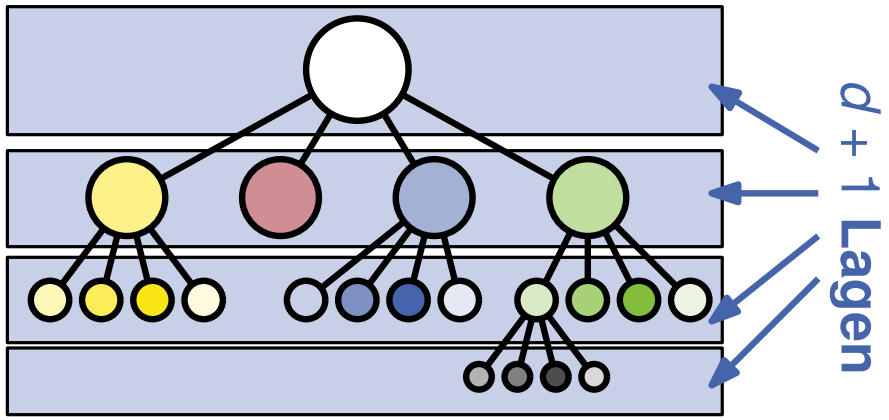
QUADTREE(P, Q) (Rückgabe: Wurzel von $T(P)$ bezüglich Q)

- $v_Q \leftarrow$ neuer Knoten, der Q repräsentiert — $O(1)$
- **if** $|P| > 1$ **then**
  - **foreach** Q_i *mit* $i \in \{\text{NW}, \text{NO}, \text{SW}, \text{SO}\}$ **do**
    - $P_i \leftarrow$ Punkte aus P, die in Q_i liegen — $O(|P|)$
    - $v_{Q_i} \leftarrow$ QUADTREE(P_i, Q_i) — Rekursion
    - Füge v_{Q_i} als Kind von v_Q ein — $O(1)$
- **else**
  - points(v_Q) $\leftarrow P$ — $O(1)$
- **return** v_Q

Aufbau des Quadtrees

QUADTREE(P, Q) (Rückgabe: Wurzel von $T(P)$ bezüglich Q)

- $v_Q \leftarrow$ neuer Knoten, der Q repräsentiert — $O(1)$
- **if** $|P| > 1$ **then**
  - **foreach** Q_i *mit* $i \in \{\text{NW}, \text{NO}, \text{SW}, \text{SO}\}$ **do**
    - $P_i \leftarrow$ Punkte aus P, die in Q_i liegen — $O(|P|)$
    - $v_{Q_i} \leftarrow$ QUADTREE(P_i, Q_i) — Rekursion
    - Füge v_{Q_i} als Kind von v_Q ein — $O(1)$
- **else**
  - points(v_Q) $\leftarrow P$ — $O(1)$
- **return** v_Q



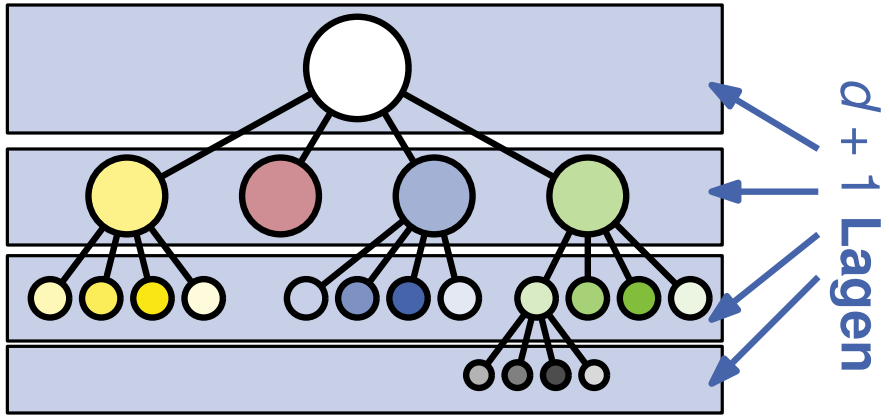
sei d die Tiefe des Quadtrees $T(P)$

Satz: Berechnung des Quadtrees
Der Quadtree $T(P)$ einer Punktmenge P enthält $O((d+1) \cdot |P|)$ Knoten und kann in $O((d+1) \cdot |P|)$ Zeit berechnet werden.

Beweis: nächste Folie

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Aufbau des Quadtrees



sei d die Tiefe des Quadtrees $T(P)$

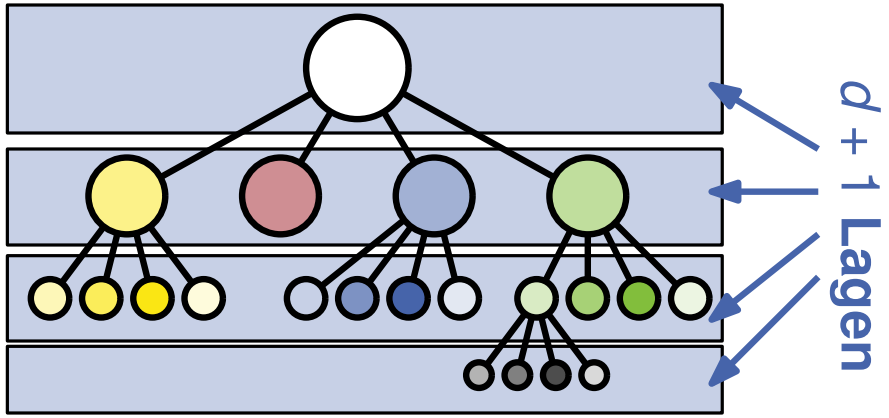
Satz: Berechnung des Quadtrees
Der Quadtree $T(P)$ einer Punktmenge P enthält $O((d+1) \cdot |P|)$ Knoten und kann in $O((d+1) \cdot |P|)$ Zeit berechnet werden.

Beweis: zeige die folgenden Aussagen:

- Jede Lage enthält nur $O(|P|)$ Knoten.

- Für alle Knoten v_{Q_i} in einer Lage benötigt der Aufruf QUADTREE(P_i, Q_i) insgesamt nur $O(|P|)$ Zeit.

Aufbau des Quadtrees



sei d die Tiefe des Quadtrees $T(P)$

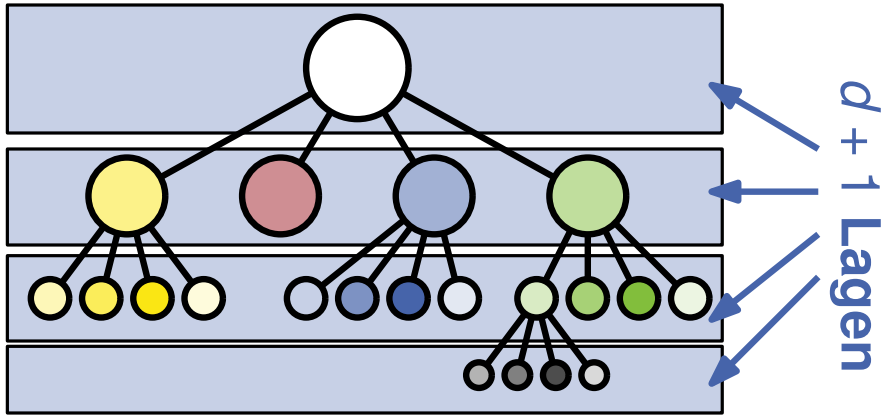
Satz: Berechnung des Quadtrees
Der Quadtree $T(P)$ einer Punktmenge P enthält $O((d+1) \cdot |P|)$ Knoten und kann in $O((d+1) \cdot |P|)$ Zeit berechnet werden.

Beweis: zeige die folgenden Aussagen:

- Jede Lage enthält nur $O(|P|)$ Knoten.
  - Immer vier Knoten in einer Lage haben einen gemeinsamen Vorgänger.
  - Das Quadrat von mind. einem der vier Knoten enthält einen Punkt aus P.
  - Jeder Punkt aus P ist pro Lage in nur einem Quadrat enthalten.
- Für alle Knoten v_{Q_i} in einer Lage benötigt der Aufruf QUADTREE(P_i, Q_i) insgesamt nur $O(|P|)$ Zeit.

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Aufbau des Quadtrees



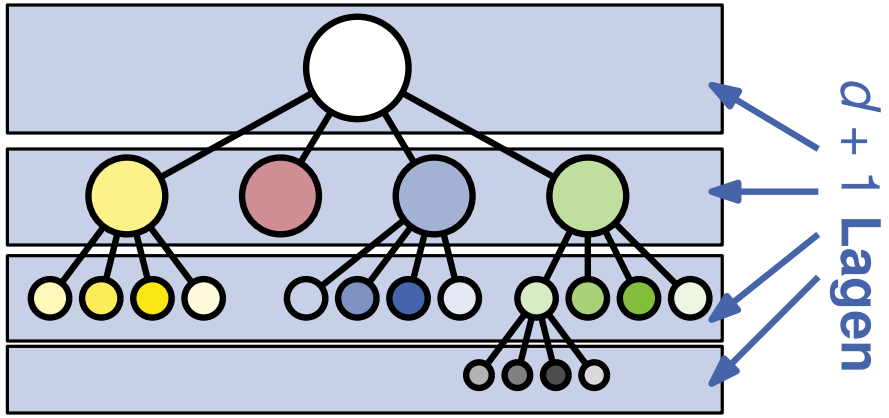
sei d die Tiefe des Quadtrees $T(P)$

Satz: Berechnung des Quadtrees
Der Quadtree $T(P)$ einer Punktmenge P enthält $O((d+1) \cdot |P|)$ Knoten und kann in $O((d+1) \cdot |P|)$ Zeit berechnet werden.

Beweis: zeige die folgenden Aussagen:

- Jede Lage enthält nur $O(|P|)$ Knoten.
  - Immer vier Knoten in einer Lage haben einen gemeinsamen Vorgänger.
  - Das Quadrat von mind. einem der vier Knoten enthält einen Punkt aus P.
  - Jeder Punkt aus P ist pro Lage in nur einem Quadrat enthalten.
- Für alle Knoten v_{Q_i} in einer Lage benötigt der Aufruf QUADTREE(P_i, Q_i) insgesamt nur $O(|P|)$ Zeit.
  - Betrachte Knoten v_{Q_i} für den Q_i keinen Punkt aus P enthält (also $P_i = \emptyset$): $O(1)$ Zeit pro Knoten $\Rightarrow$ insgesamt $O(|P|)$.
  - Andernfalls: $O(|P_i|)$ Zeit für Knoten v_{Q_i} $\Rightarrow$ insgesamt $O(|P|)$, da $\sum |P_i| \leq |P|$

Aufbau des Quadtrees



sei d die Tiefe des Quadtrees $T(P)$

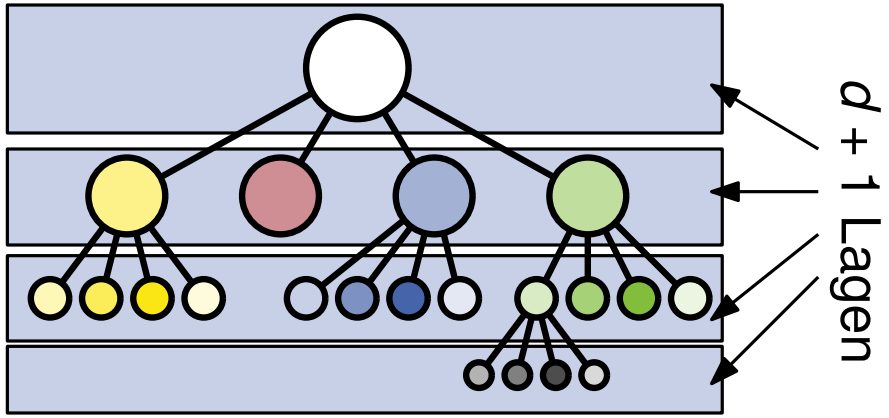
Satz: Berechnung des Quadtrees
Der Quadtree $T(P)$ einer Punktmenge P enthält $O((d+1) \cdot |P|)$ Knoten und kann in $O((d+1) \cdot |P|)$ Zeit berechnet werden.

(Hinweis: $(d+1)$, da $d = 0$ gelten könnte)

Beweis: zeige die folgenden Aussagen:

- Jede Lage enthält nur $O(|P|)$ Knoten.
  - Immer vier Knoten in einer Lage haben einen gemeinsamen Vorgänger.
  - Das Quadrat von mind. einem der vier Knoten enthält einen Punkt aus P.
  - Jeder Punkt aus P ist pro Lage in nur einem Quadrat enthalten.
- Für alle Knoten v_{Q_i} in einer Lage benötigt der Aufruf QUADTREE(P_i, Q_i) insgesamt nur $O(|P|)$ Zeit.
  - Betrachte Knoten v_{Q_i} für den Q_i keinen Punkt aus P enthält (also $P_i = \emptyset$): $O(1)$ Zeit pro Knoten $\Rightarrow$ insgesamt $O(|P|)$.
  - Andernfalls: $O(|P_i|)$ Zeit für Knoten $v_{Q_i} \Rightarrow$ insgesamt $O(|P|)$, da $\sum |P_i| \leq |P|$

Tiefe des Quadtrees

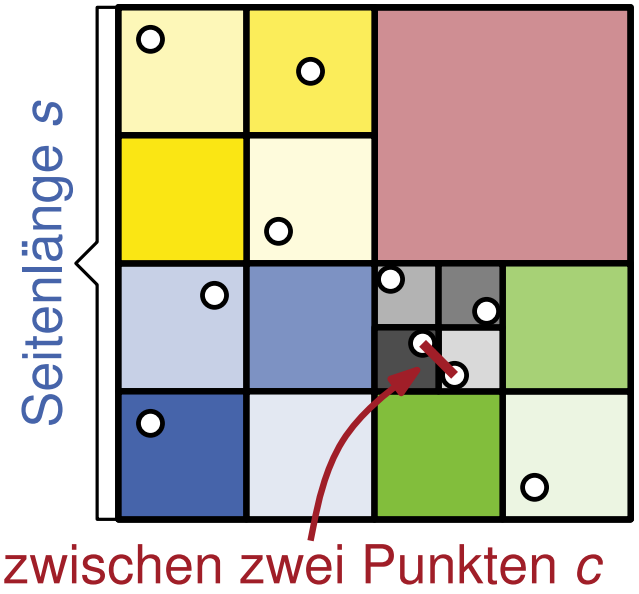


sei d die Tiefe des Quadtrees $T(P)$

Satz: Berechnung des Quadtrees
Der Quadtree $T(P)$ einer Punktmenge P enthält $O((d+1)\cdot|P|)$ Knoten und kann in $O((d+1)\cdot|P|)$ Zeit berechnet werden.

Satz: Tiefe des Quadtrees
Die Tiefe d von $T(P)$ ist höchstens $\log_2\left(\frac{s}{c}\right)+\frac{3}{2}$.

Beweis: Übungsaufgabe

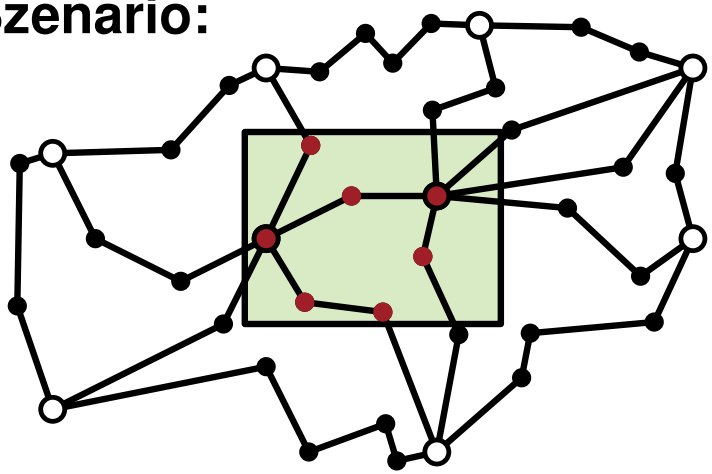


Für „gleichmäßig verteilte“ Punkte ist die Tiefe d von $T(P)$ „klein“.
Beispiel: Liegen die Punkte auf einem polynomiell großen Gitter (z.B.: $|P|^2\times|P|^3$), so gilt $d\in O(\log|P|)$.

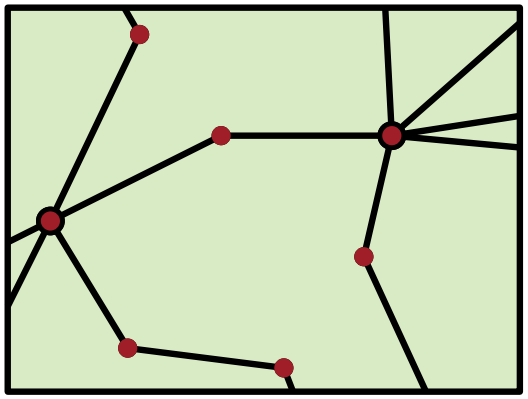
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Bereichsanfrage

Szenario:



großer Straßengraph



zeige nur einen kleinen Ausschnitt

Problem: Bereichsanfrage
Gegeben eine Punktmenge P (die Knoten in dem Straßengraph) sowie ein Rechteck R, finde alle Punkte aus P die in R liegen.

Einfache Lösung:
Teste für jeden Punkt $p \in P$, ob p in R liegt.
$\Rightarrow$ Laufzeit $O(|P|)$, dabei werden ggf. sehr viel weniger Punkte angezeigt.

Ziel im Folgenden:
Nutze aus, dass nur ein sehr kleiner Anteil der $|P|$ Punkte angezeigt werden muss.

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Bereichsanfrage

BEREICHSANFRAGE(v_Q, R) (v_Q ist ein Knoten von $T(P)$ (zu Beginn die Wurzel))

if $Q \subseteq R$ **then**
  gib alle Punkte in Q aus
else if v_Q *ist ein Blatt* **then**
  if $Q \cap R$ *enthält einen Punkt* $p \in P$ **then** gibt p aus
else
  foreach *Kind* v_{Q_i} *von* v_Q *mit* $i \in \{\text{NW}, \text{NO}, \text{SW}, \text{SO}\}$ **do**
    if $Q_i \cap R \neq \emptyset$ **then** BEREICHSANFRAGE(v_{Q_i}, R)

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Bereichsanfrage

BEREICHSANFRAGE(v_Q, R) (v_Q ist ein Knoten von $T(P)$ (zu Beginn die Wurzel))

if $Q \subseteq R$ **then** — vorzeitiges Rekursionsende
  gib alle Punkte in Q aus

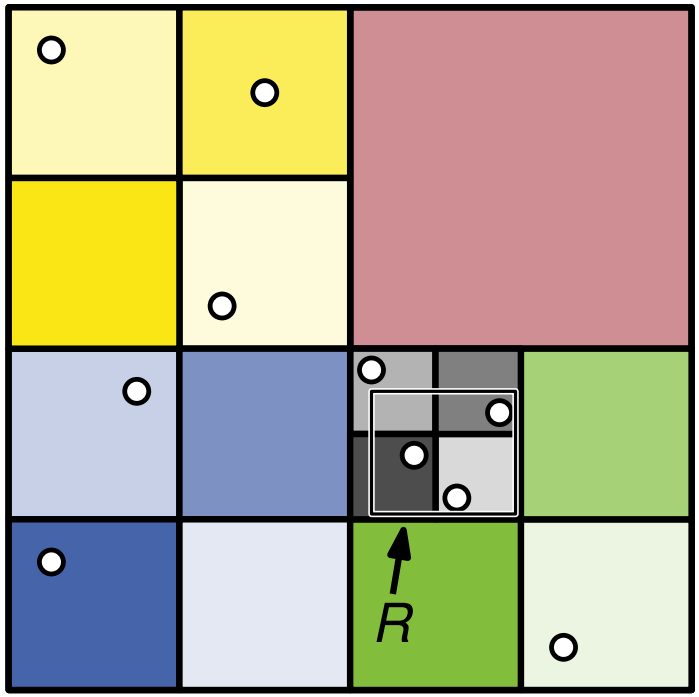
else if v_Q *ist ein Blatt* **then** — Rekursionsende
  if $Q \cap R$ *enthält einen Punkt* $p \in P$ **then** gibt p aus

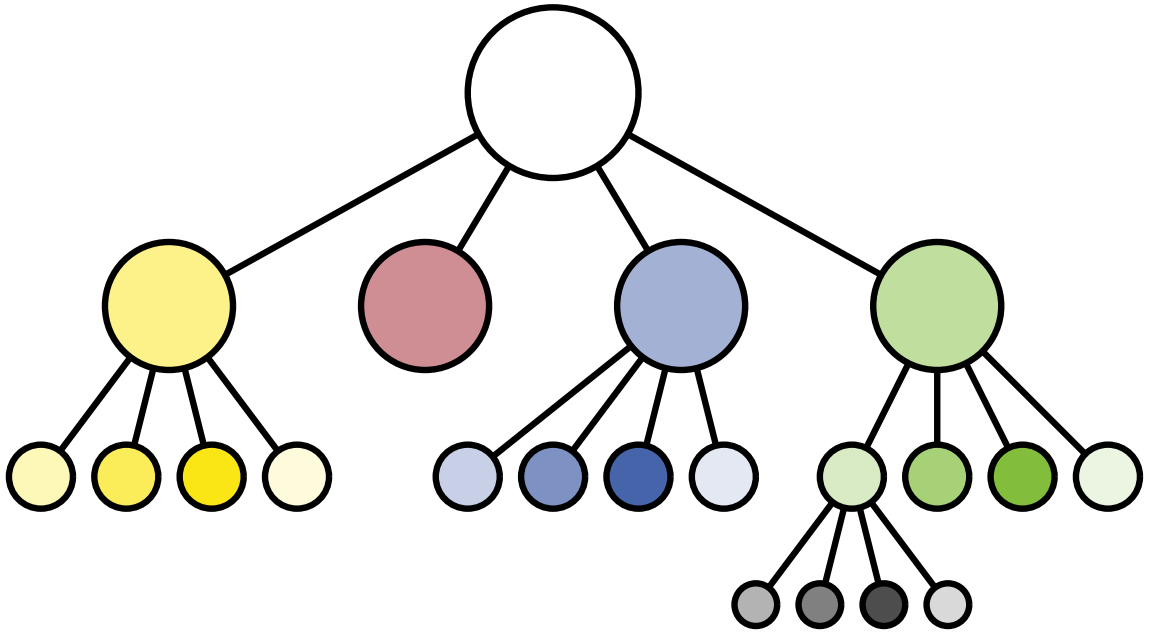
else — Rekursionsaufruf
  foreach *Kind* v_{Q_i} *von* v_Q *mit* $i \in \{\text{NW}, \text{NO}, \text{SW}, \text{SO}\}$ **do**
    if $Q_i \cap R \neq \emptyset$ **then** BEREICHSANFRAGE(v_{Q_i}, R)

Es kann passieren, dass der gesamte Baum $T(P)$ traversiert wird, aber:

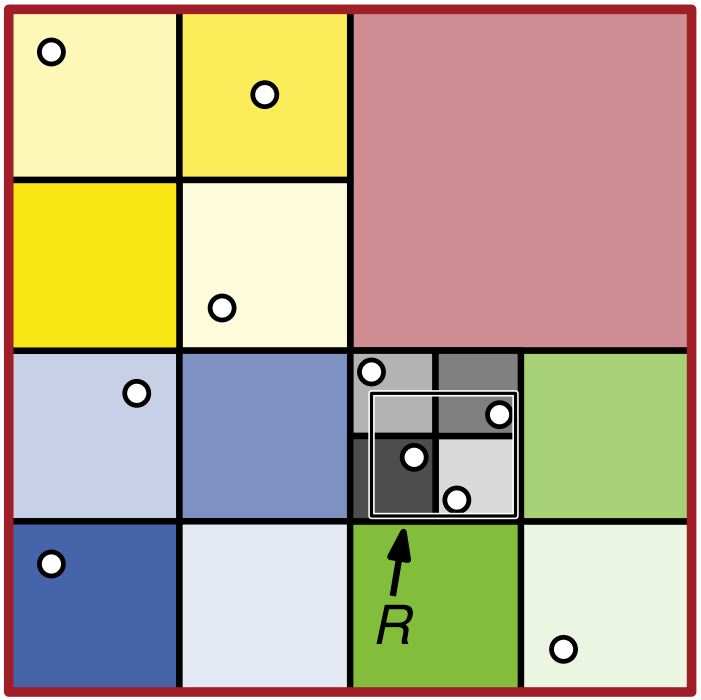
- Ein Quadrat außerhalb von R wird im Rekursionsaufruf nicht weiter verfolgt.
- Die Traversierung wird vorzeitig abgebrochen, wenn ein Quadrat komplett in R liegt. (Es werden nur noch die enthaltenen Knoten ausgegeben. Übung: Wie geht das schnell?)

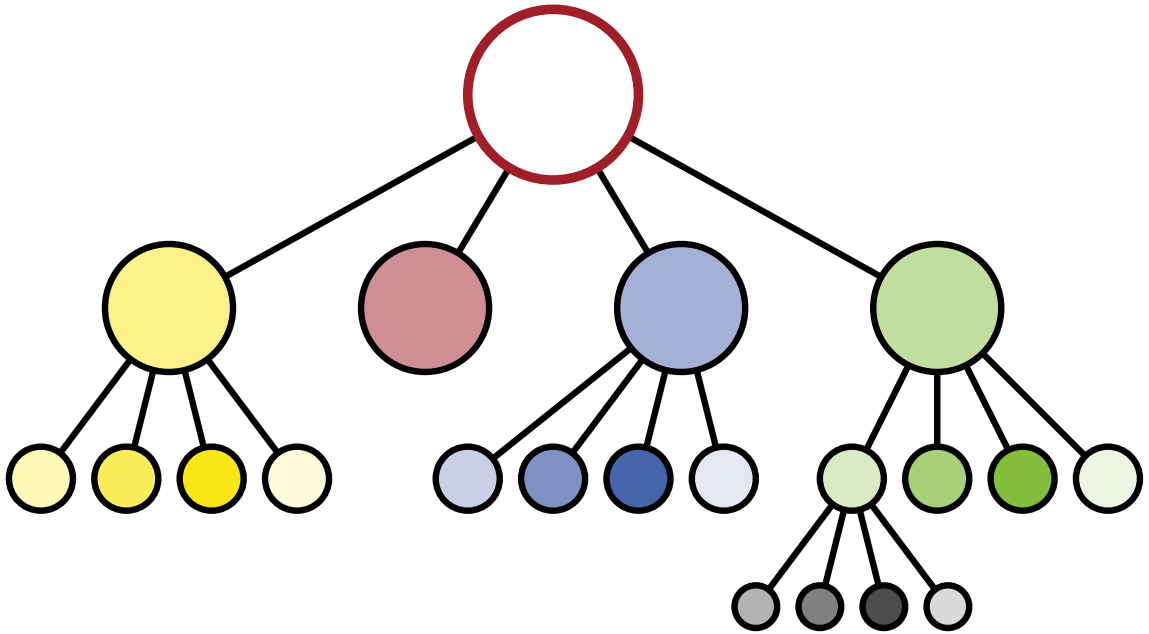
Bereichsanfrage – Beispiel





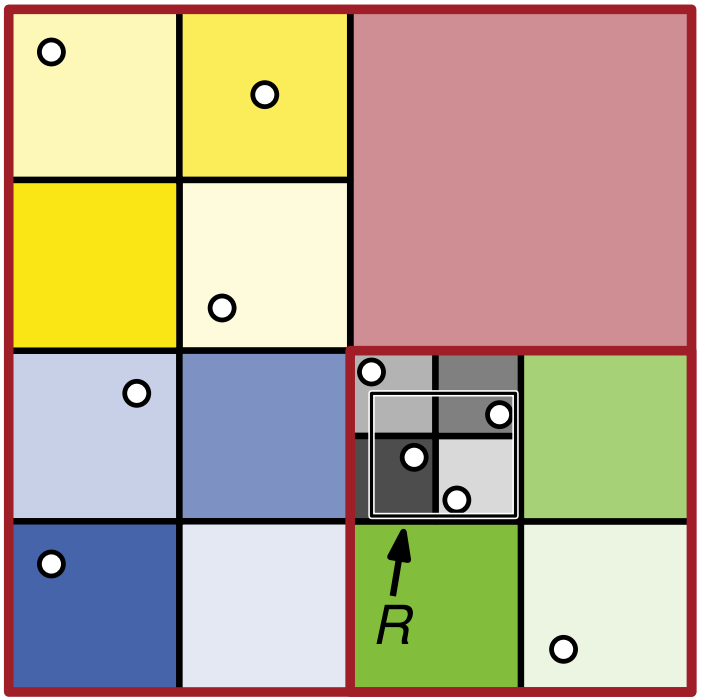
Bereichsanfrage – Beispiel

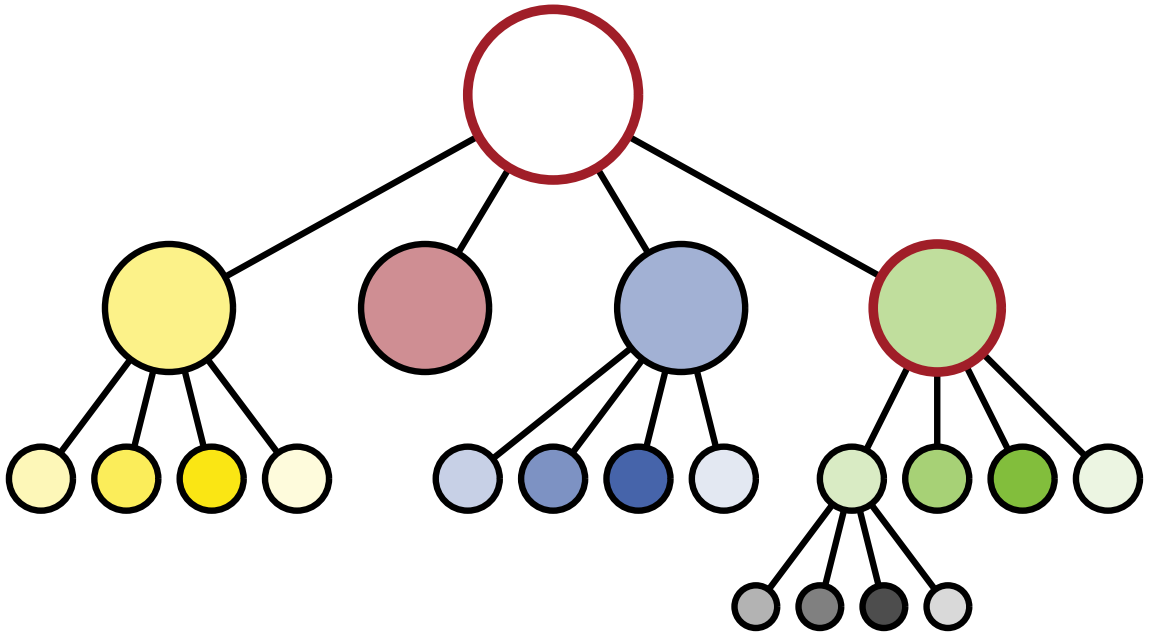




Starte mit der Wurzel

Bereichsanfrage – Beispiel

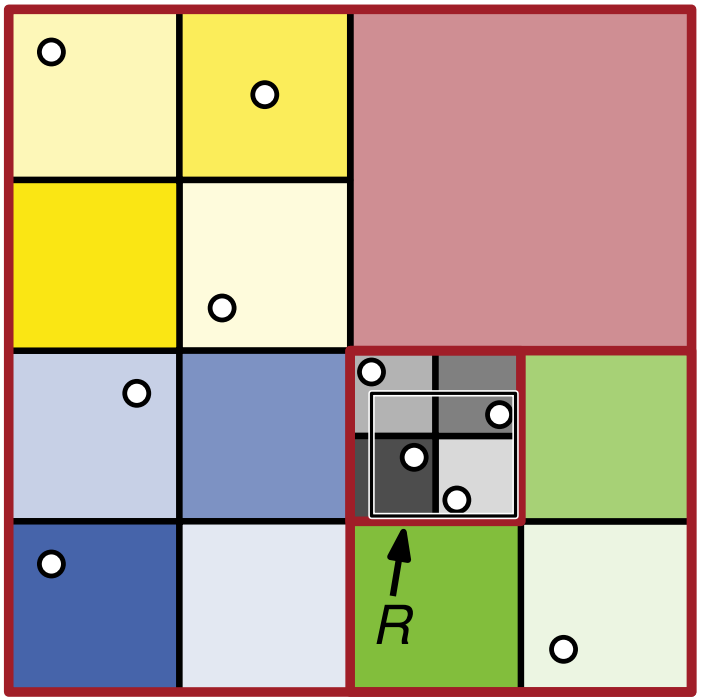


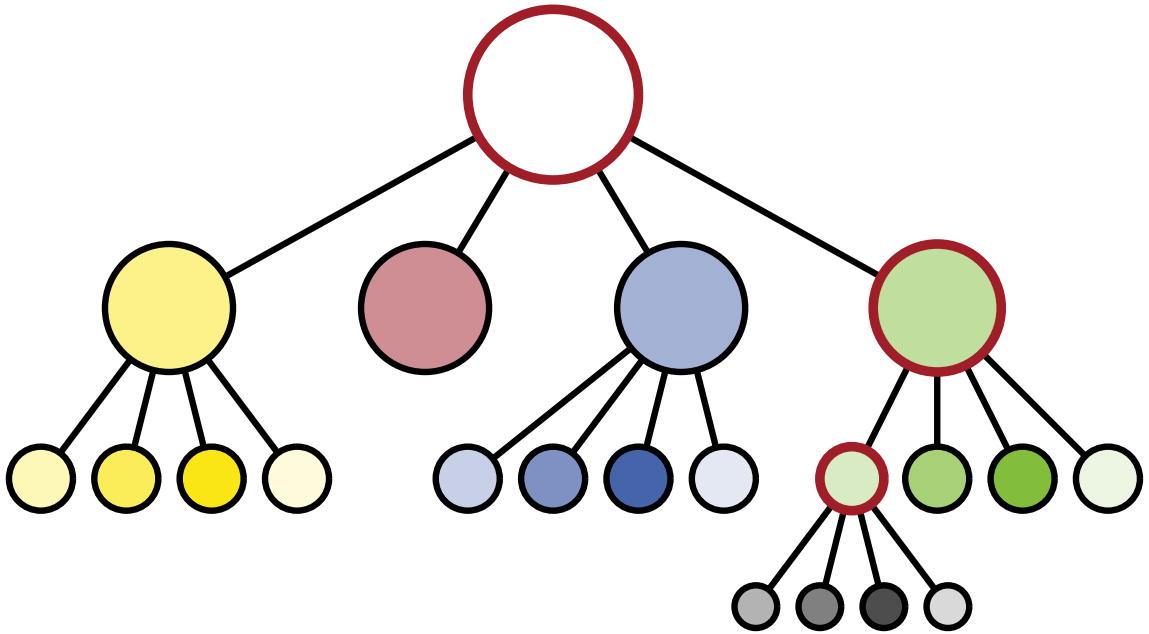


Starte mit der Wurzel

$\rightarrow$ Rekursionsaufruf für nur eins der Kinder

Bereichsanfrage – Beispiel



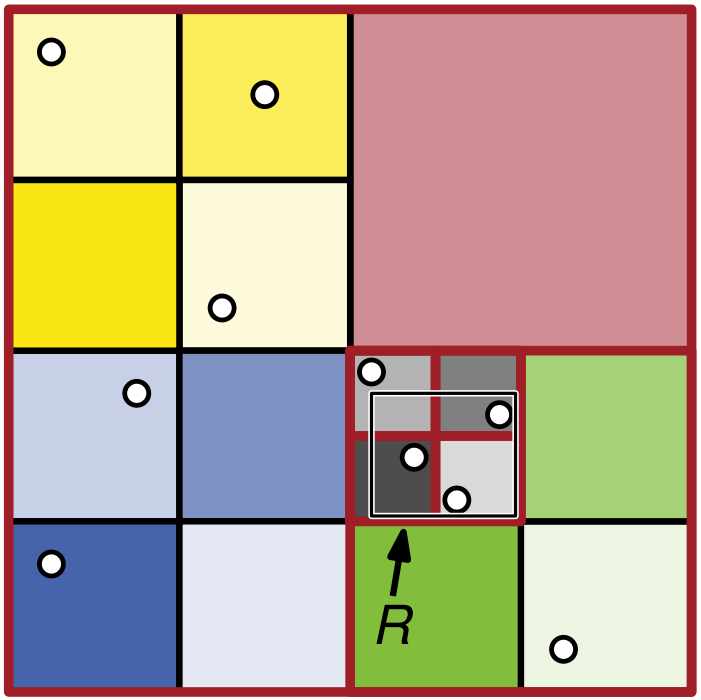


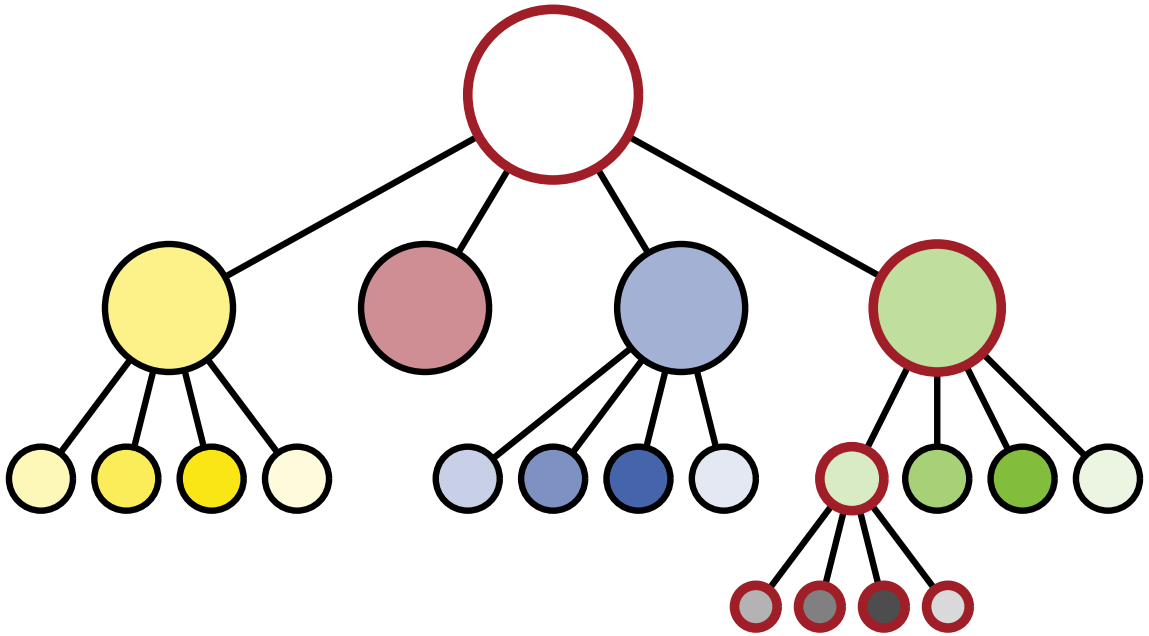
Starte mit der Wurzel

$\rightarrow$ Rekursionsaufruf für nur eins der Kinder

$\rightarrow$ Rekursionsaufruf für nur eins der Kinder

Bereichsanfrage – Beispiel





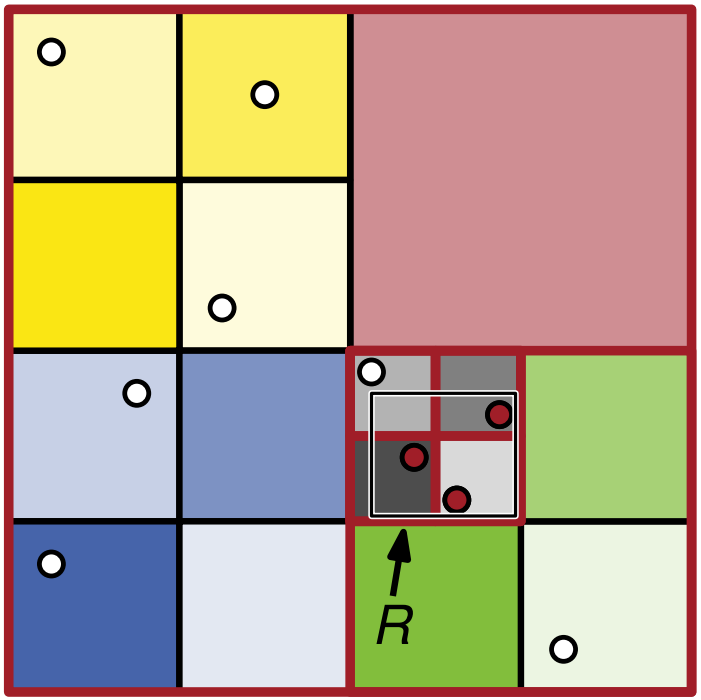
Starte mit der Wurzel

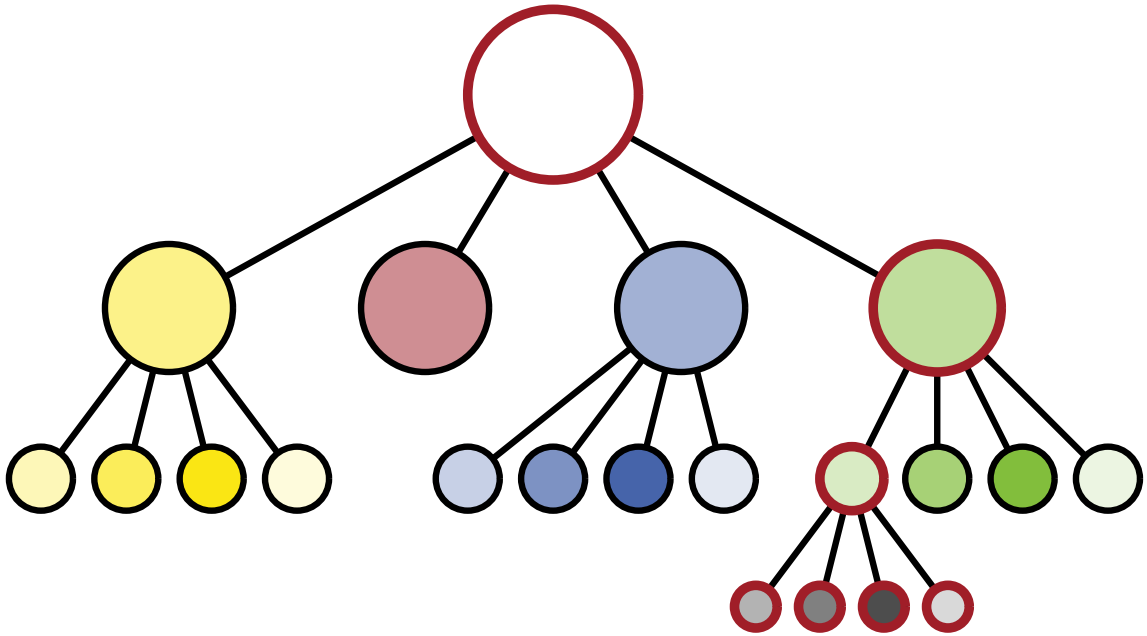
$\rightarrow$ Rekursionsaufruf für nur eins der Kinder

$\rightarrow$ Rekursionsaufruf für nur eins der Kinder

$\rightarrow$ Rekursionsaufruf für alle vier Kinder

Bereichsanfrage – Beispiel





Starte mit der Wurzel

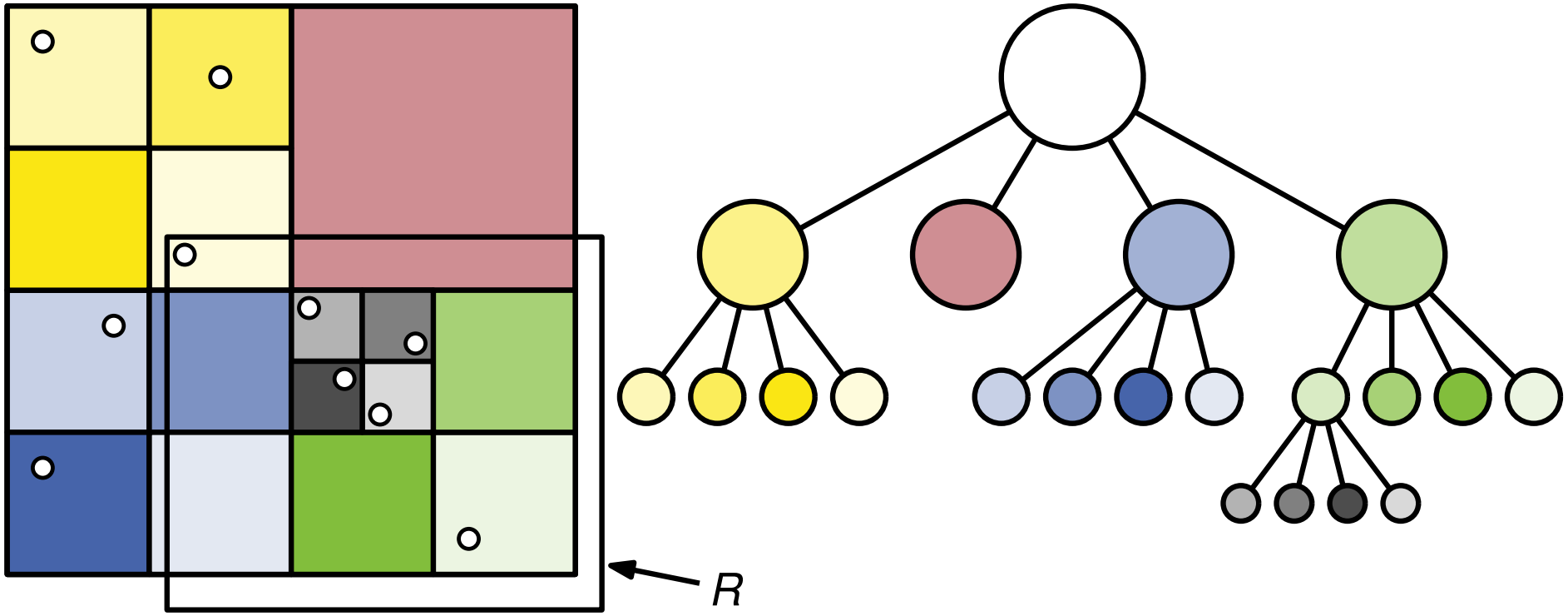
→ Rekursionsaufruf für nur eins der Kinder

→ Rekursionsaufruf für nur eins der Kinder

→ Rekursionsaufruf für alle vier Kinder

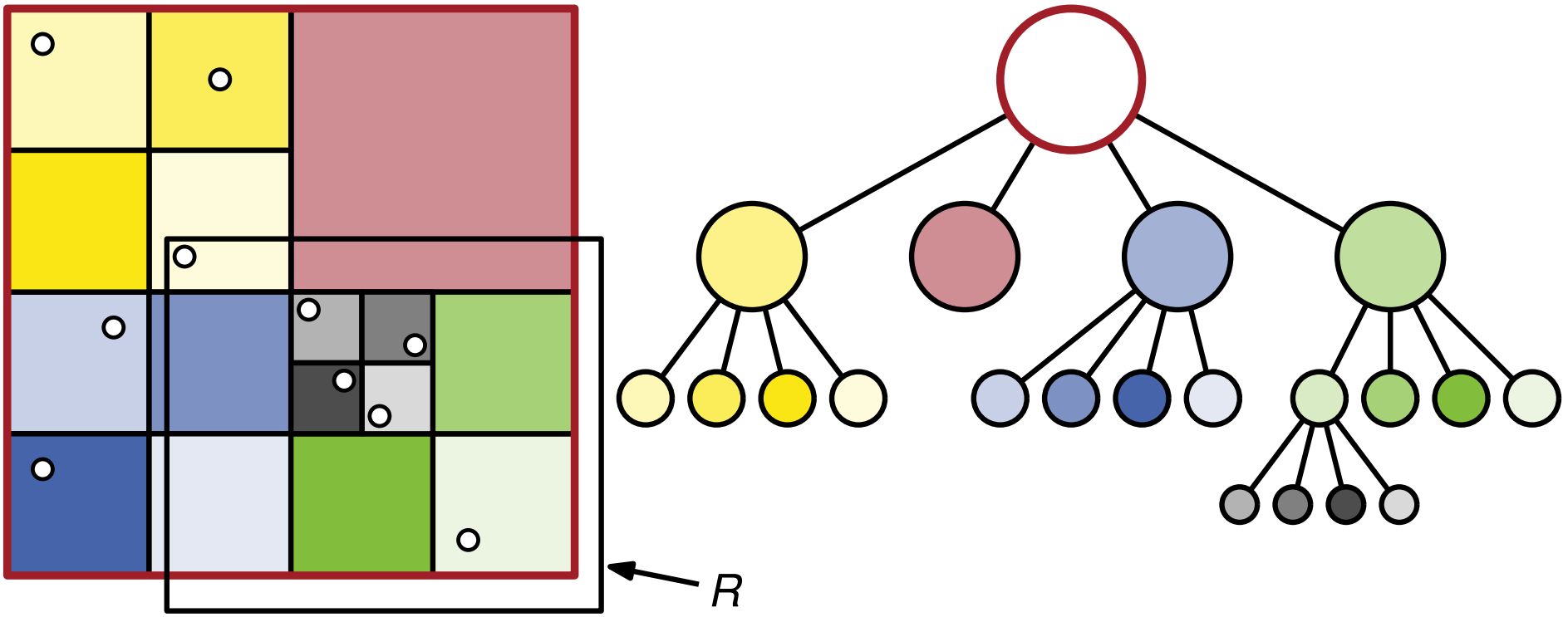
→ Rekursionsende (Blätter erreicht): gib resultierende Knoten aus

Bereichsanfrage – Beispiel



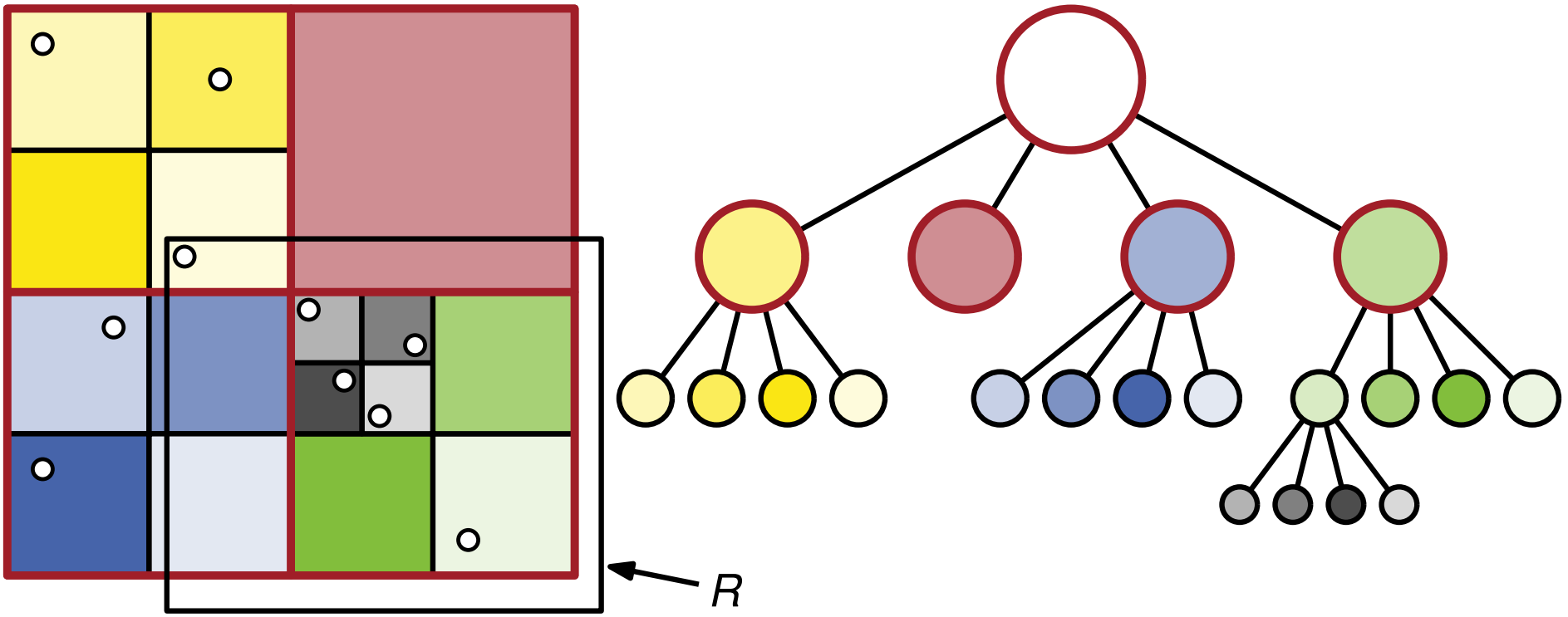
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Bereichsanfrage – Beispiel



Starte mit der Wurzel

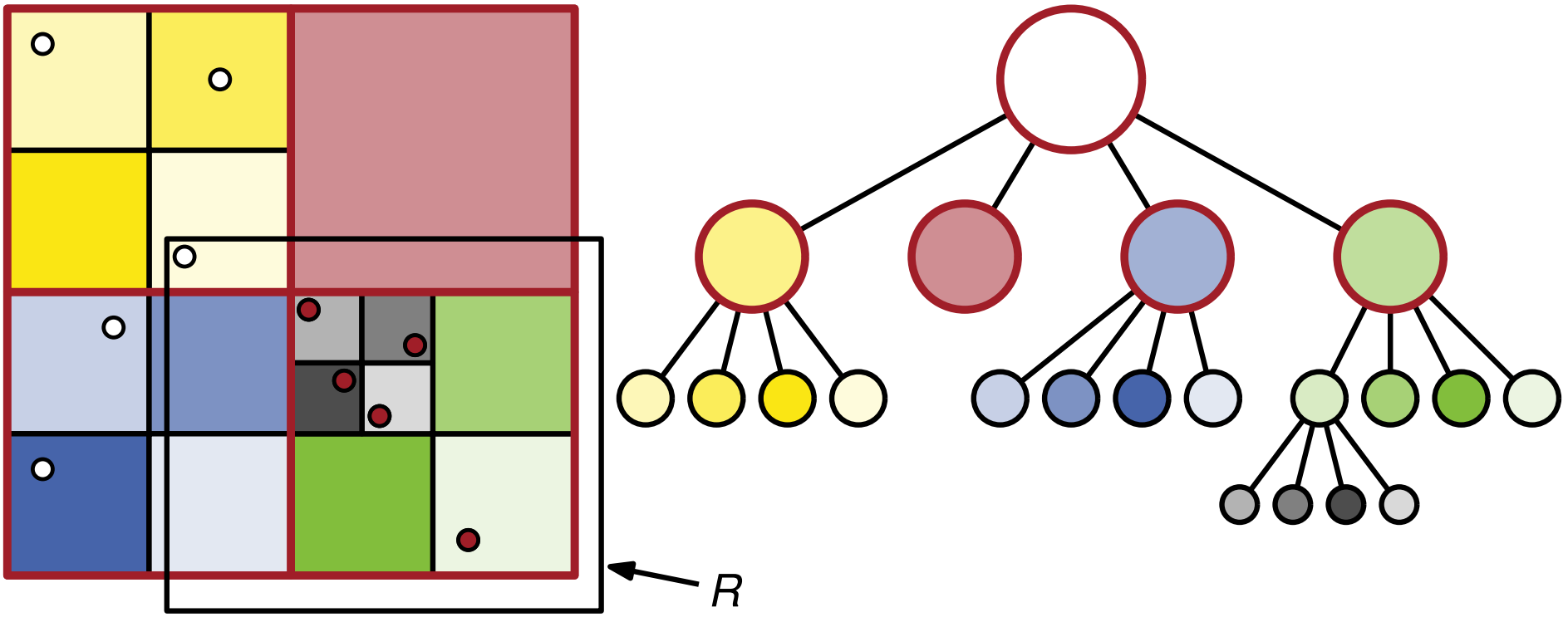
Bereichsanfrage – Beispiel



Starte mit der Wurzel

$\rightarrow$ Rekursionsaufruf für alle vier Kinder

Bereichsanfrage – Beispiel

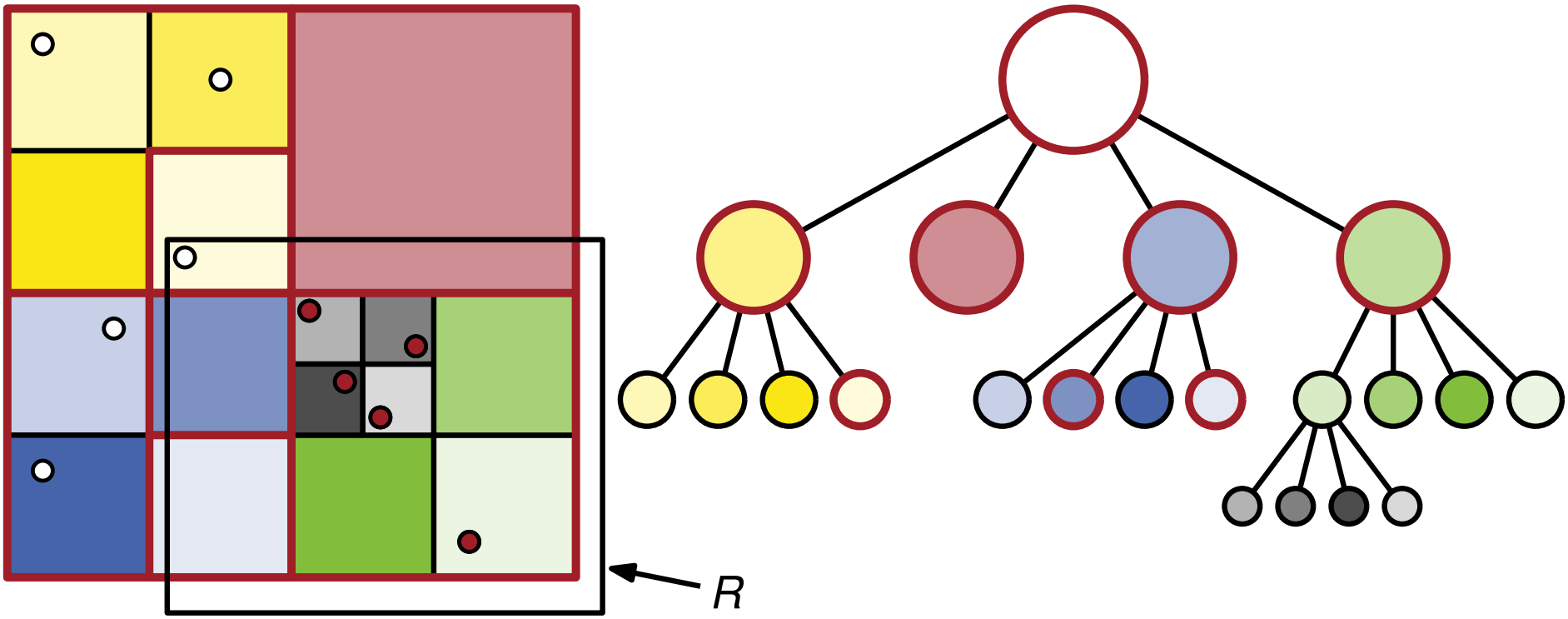


Starte mit der Wurzel

$\rightarrow$ Rekursionsaufruf für alle vier Kinder

$\rightarrow$ Süd-Ost-Knoten: vorzeitiger Abbruch und Ausgabe der enthaltenen Punkte

Bereichsanfrage – Beispiel



Starte mit der Wurzel

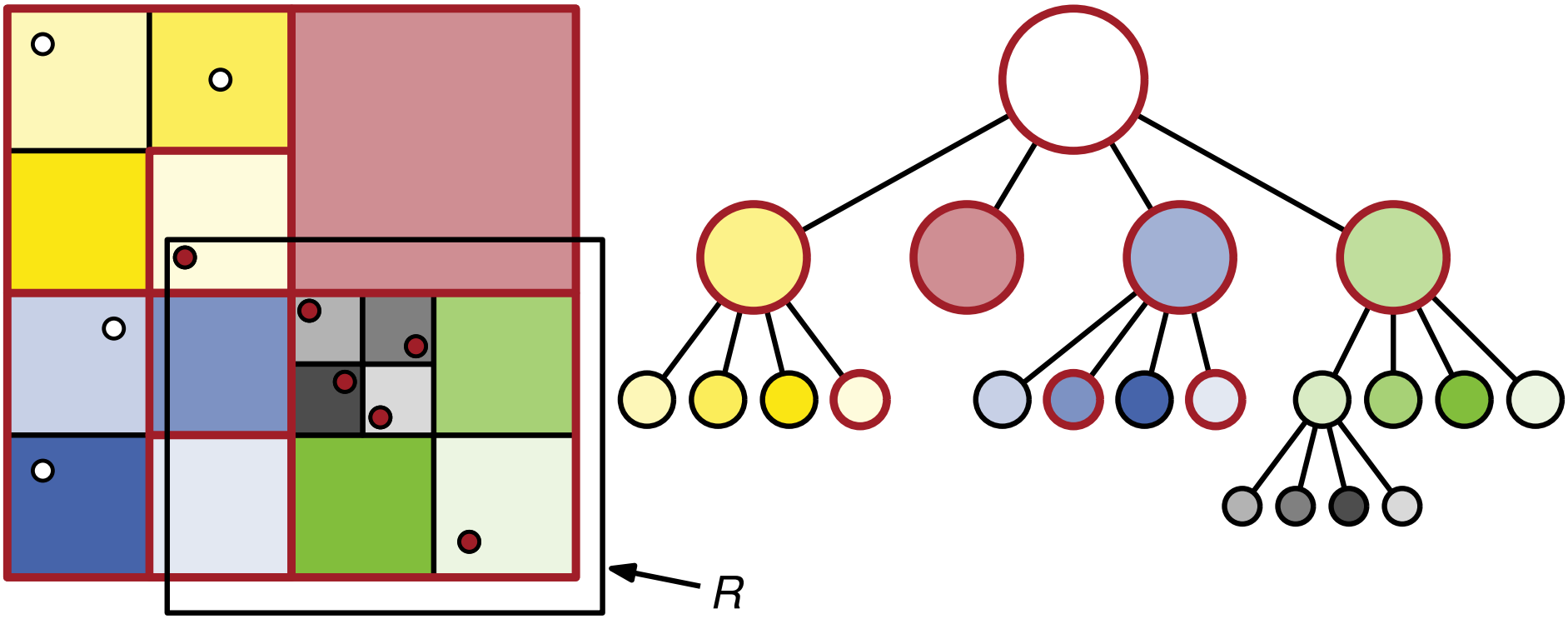
→ Rekursionsaufruf für alle vier Kinder

→ Süd-Ost-Knoten: vorzeitiger Abbruch und Ausgabe der enthaltenen Punkte

→ Süd-West-Knoten: Rekursionsaufruf für zwei Kinder

→ Nord-West-Knoten: Rekursionsaufruf für ein Kind

Bereichsanfrage – Beispiel



Starte mit der Wurzel

→ Rekursionsaufruf für alle vier Kinder

→ Süd-Ost-Knoten: vorzeitiger Abbruch und Ausgabe der enthaltenen Punkte

→ Süd-West-Knoten: Rekursionsaufruf für zwei Kinder

→ Nord-West-Knoten: Rekursionsaufruf für ein Kind

→ Rekursionsende (Blätter erreicht): gib resultierende Knoten aus