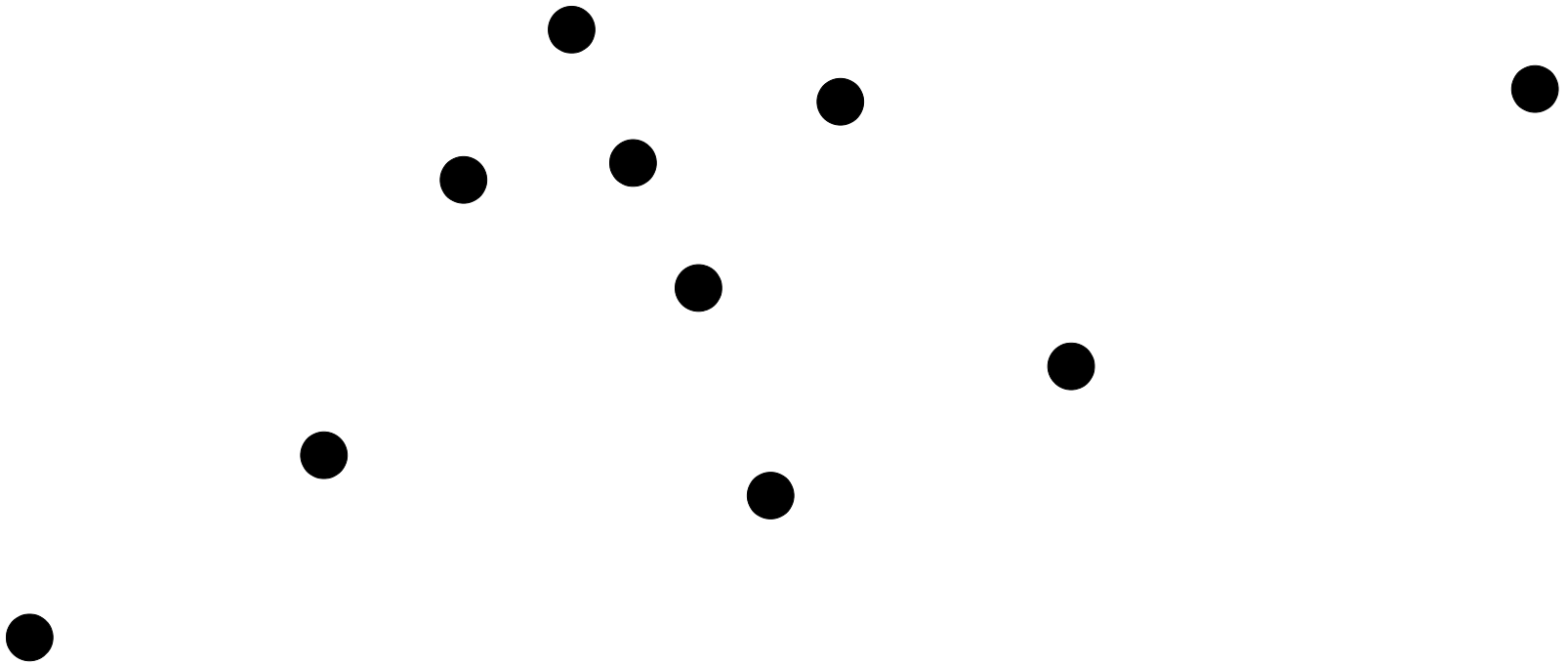
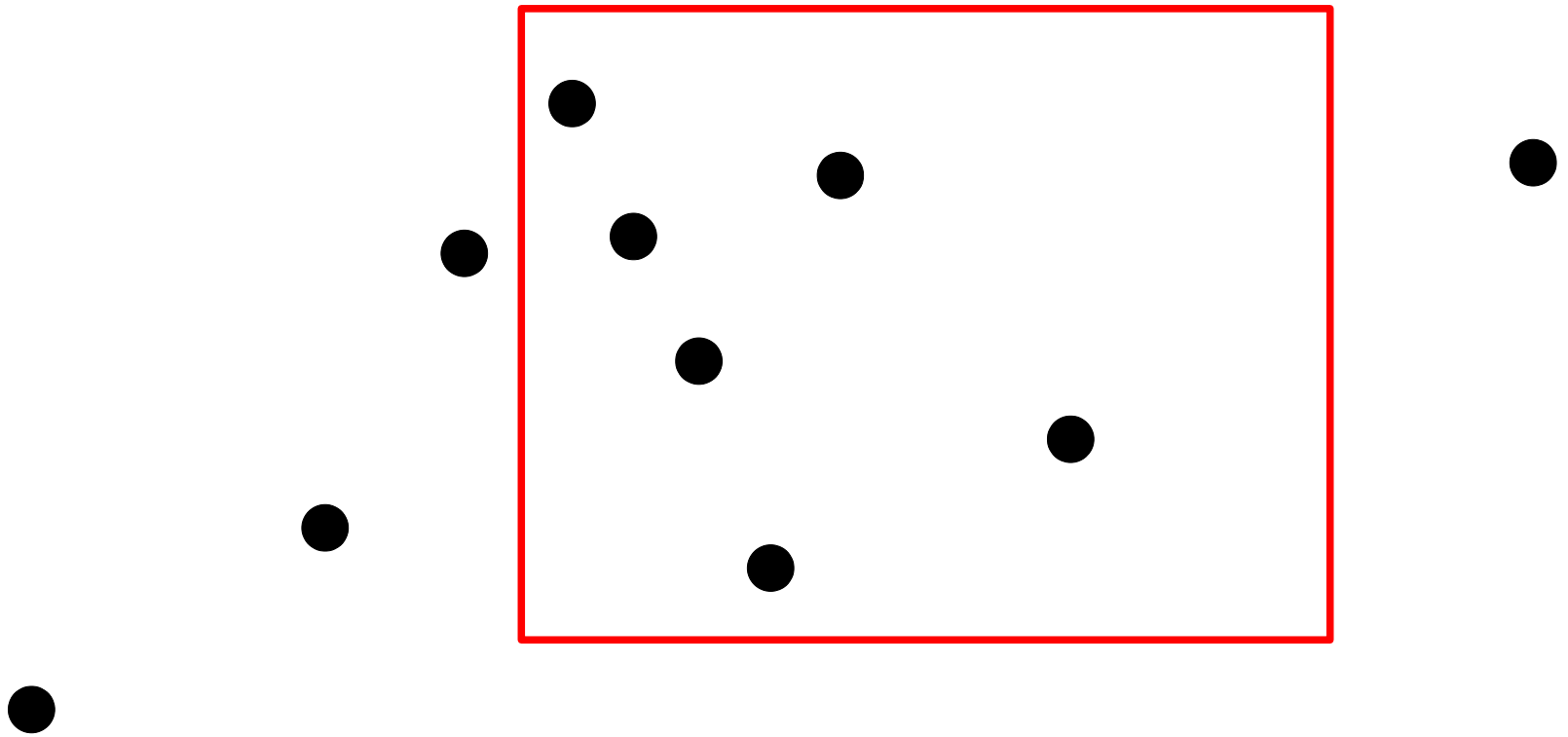
Orthogonale Bereichsanfragen

1. Beschreibung der Aufgabenstellung

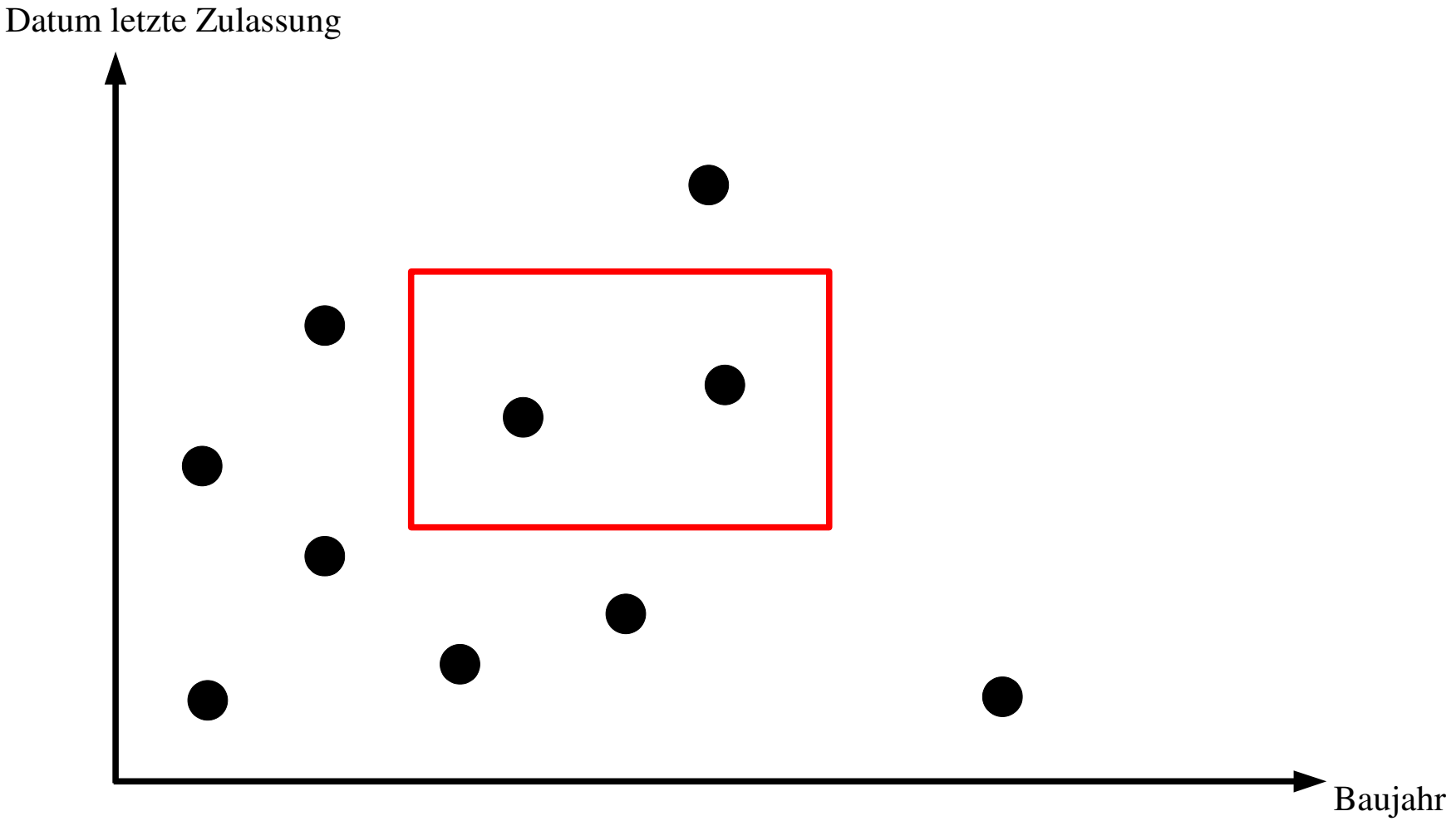
Gegeben ist eine Menge von **Punkten in der Ebene**.



Es soll eine **Datenstruktur** aufgebaut werden, die orthogonale Bereichsanfragen unterstützt.

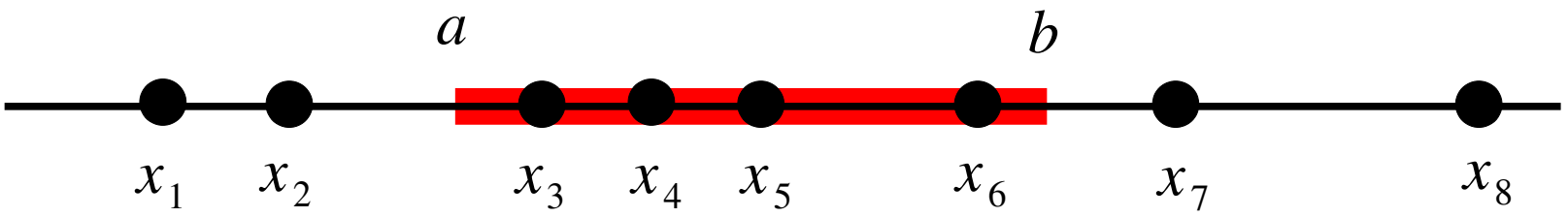


Anfragen an Datenbanken können Bereichsanfragen sein.



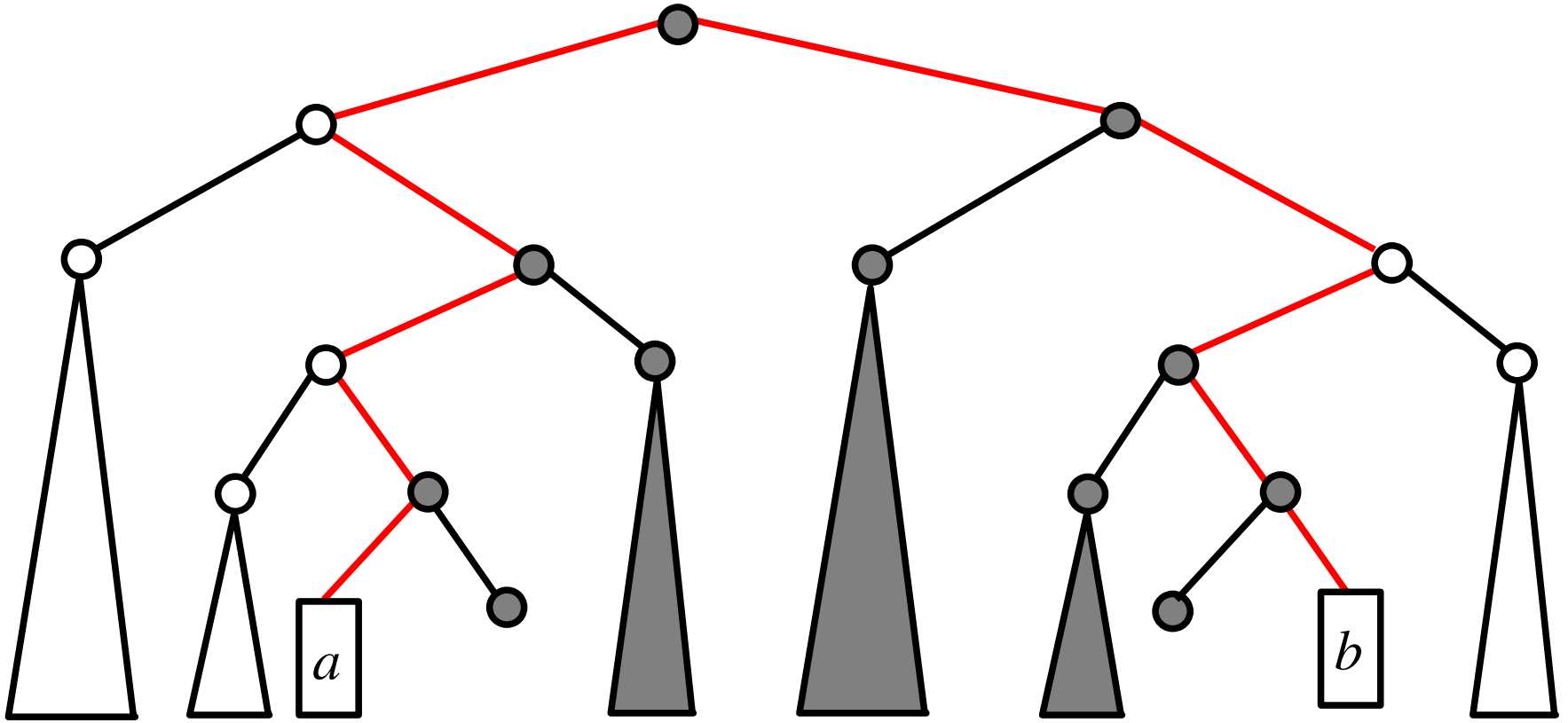
2. Vorüberlegungen in Dimension d=1

Eine Bereichsanfrage in **Dimension $d = 1$:**



Hier können wir einen **höhenbalancierten binären Suchbaum** (z.B. einen Rot-Schwarz-Baum) einsetzen.

Man sucht nach *a* und nach *b* und gibt **alle Werte dazwischen** aus.



Der **Aufwand** zur Beantwortung einer Bereichsanfrage lässt sich abschätzen:

$O(\log n)$ für die Suche nach a und b

$O(k)$ für das Traversieren ganzer Teilbäume

Dabei ist k die **Anahl der Werte** im angefragten Bereich.

Man erhält also ein **output-sensitives Verfahren**.

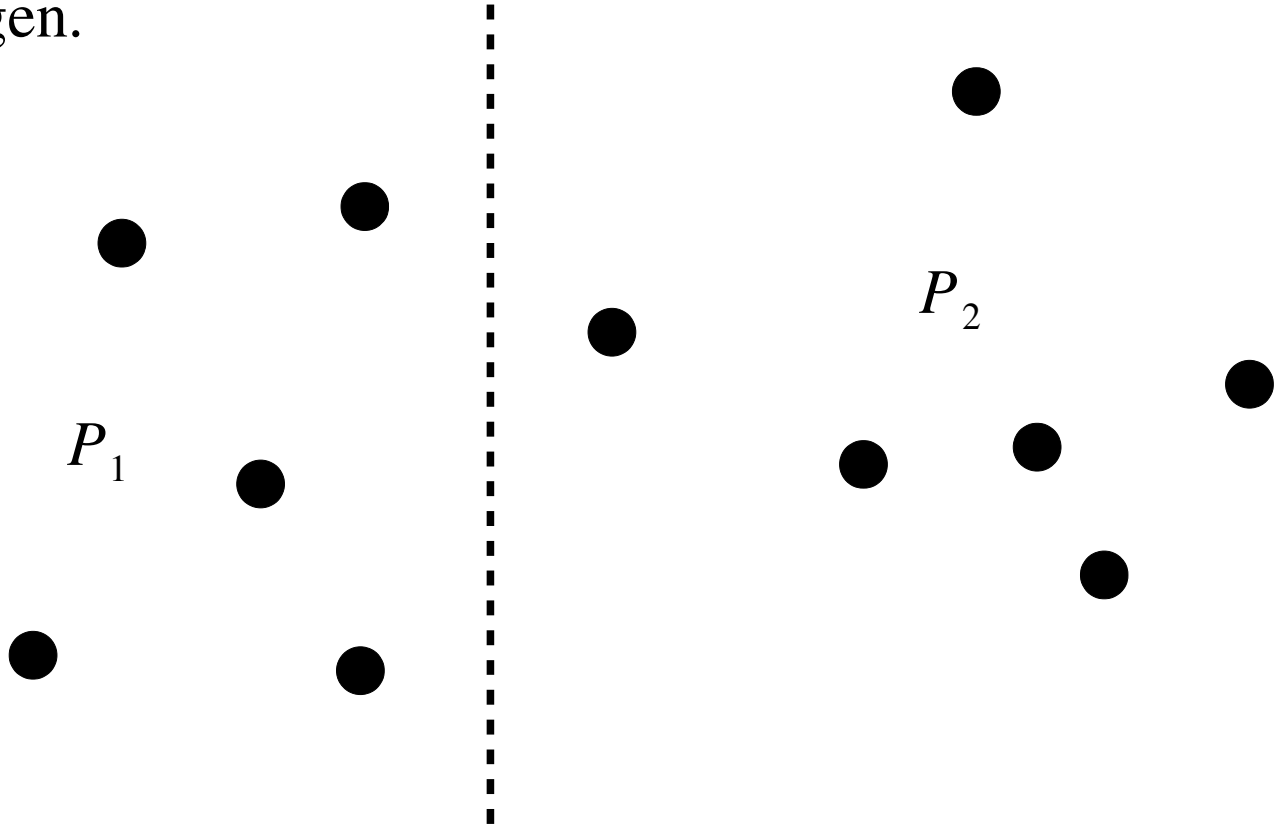
Zusammenfassung für Dimension d=1:

Aufbau der Datenstruktur:	$O(n\log n)$
Bearbeitung einer Anfrage:	$O(\log n + k)$
Speicherbedarf:	$O(n)$

3. Eine erste Datenstruktur für d=2

P sei die gegebene Punktemenge in der Ebene.

Eine **vertikale Splitgerade** ***s*** für P separiert P in zwei “gleich” große Teilmengen.



Analog sind **horizontale Splitgeraden** definiert.

Grundidee zum Aufbau der Datenstruktur:

Immer abwechselnd vertikale und horizontale Splits durchführen und diese Splits den Knoten in einem binären Baum zuordnen.

BAUE_2d_BAUM(P, *flag*)

1. Wenn P genau einen Punkt enthält, dann bilde ein Blatt.

2. Sonst: Wenn *flag* = *h* ist:

3. Bilde neuen inneren Knoten u.

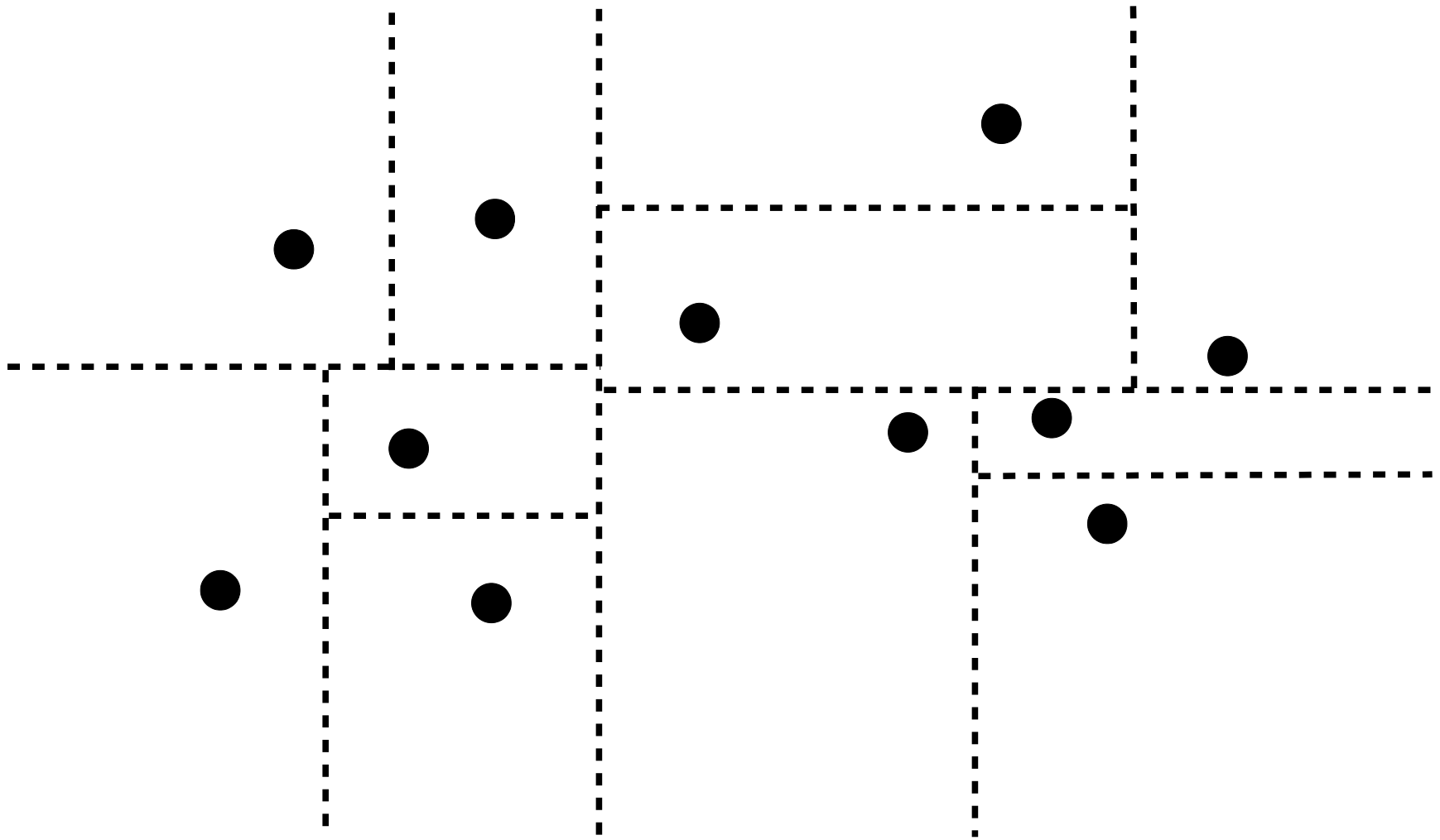
4. Finde eine horizontale Splitgerade für P.

5. Linker Teilbaum von u: BAUE_2d_BAUM(P_1 , v)

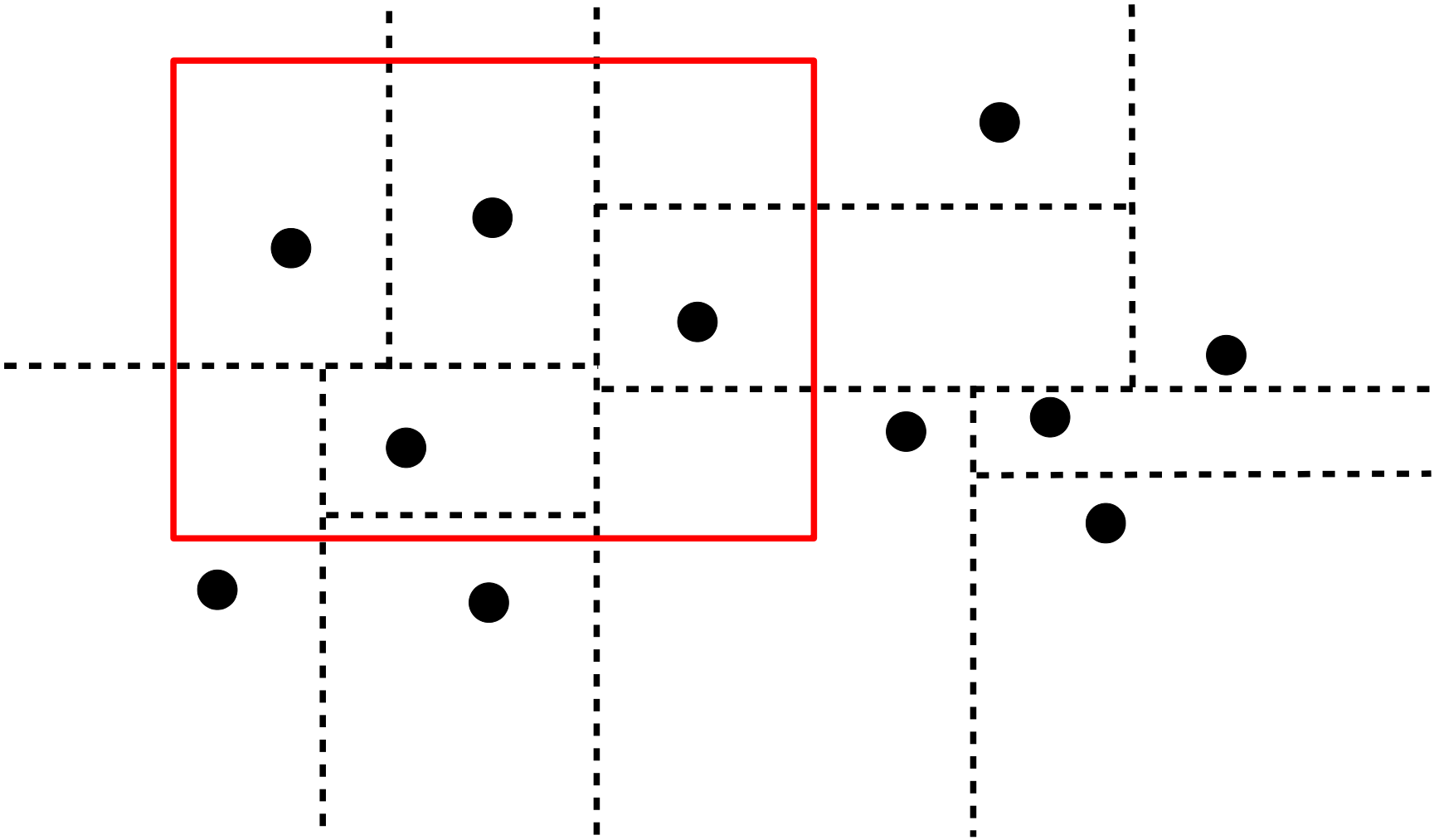
6. Rechter Teilbaum von u: BAUE_2d_BAUM(P_2 , v)

7. Wenn *flag* = *v* ist: ...

Die dem Baum zu Grunde liegende Zerlegung.



Ein Anfragebereich.



ANFRAGE_2d_BAUM(*R*, *u*)

1. Wenn *u* ein Blatt ist, dann teste, ob der in *u* gespeicherte Punkt in *R* liegt.

2. Sonst: Wenn die zu *u* gehörende Region der Zerlegung in *R* **enthalten ist**, dann gib alle Punkte aus, die im Teilbaum mit Wurzel *u* gespeichert sind.

3. Sonst

Wenn *R* die Region von *u*-s linkem Sohn schneidet:
ANFRAGE_2d_BAUM(*R*, linker Sohn von *u*)
Wenn *R* die Region von *u*-s rechtem Sohn schneidet:
ANFRAGE_2d_BAUM(*R*, rechter Sohn von *u*)

Um den **Aufwand abzuschätzen**, den eine Bereichsanfrage an einen 2d-Baum kostet, bemerken wir zuerst:

Im Fall, in dem **alle Punkte eines Teilbaums** zurückgegeben werden müssen, ist dies mit einem Aufwand, der proportional zur Anzahl der zurückgegebenen Punkte ist, möglich.

Wir müssen dazu einen Binärbaum vollständig traversieren und den Inhalt aller Blätter ausgeben.

Es bleibt nun noch die **Anzahl derjenigen Regionen** der Zerlegung abzuschätzen, die bei der Suche besucht werden, aber nicht in R enthalten sind.

Diese ist in $O(\sqrt{n})$.

Zusammenfassung zu 2d-Bäumen

Aufbau des 2d-Baumes:	$O(n \log n)$
Bereichsanfrage:	$O(k + \sqrt{n})$
Speicherbedarf:	$O(n)$

4. Eine verbesserte Datenstruktur für d=2

Wir wollen versuchen, ohne die Zeit zum Aufbau der Datenstruktur oder den Speicherplatzbedarf sehr zu erhöhen, eine **polylogarithmische Zeit zur Beantwortung** von Bereichsanfragen zu erhalten.

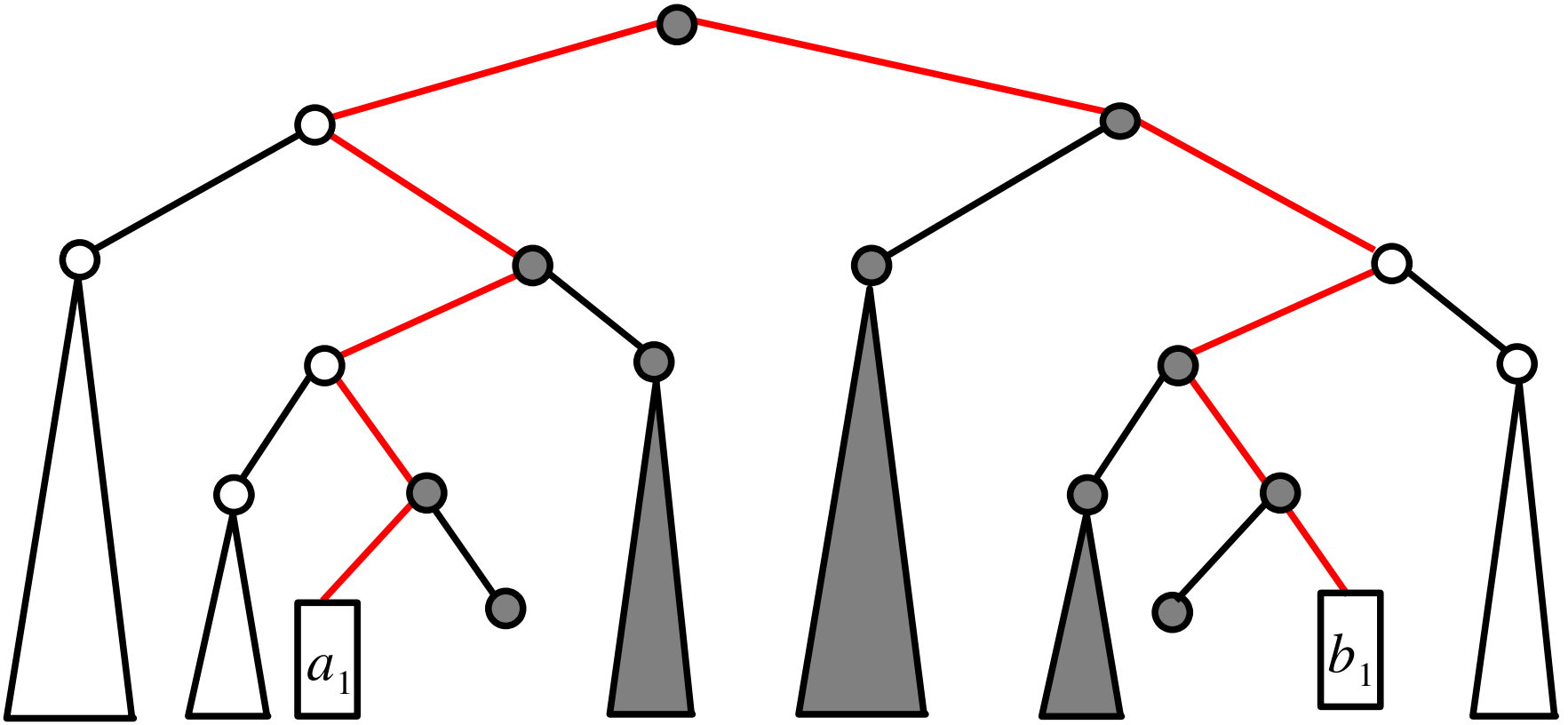
Um dies zu erreichen, nutzen wir aus, dass sich eine *2*-dimensionale Bereichsanfrage aus *2* eindimensionalen Bereichsanfragen zusammensetzt.

Man nennt die entstehenden Datenstrukturen **Bereichsbäume** oder **Range Trees**.

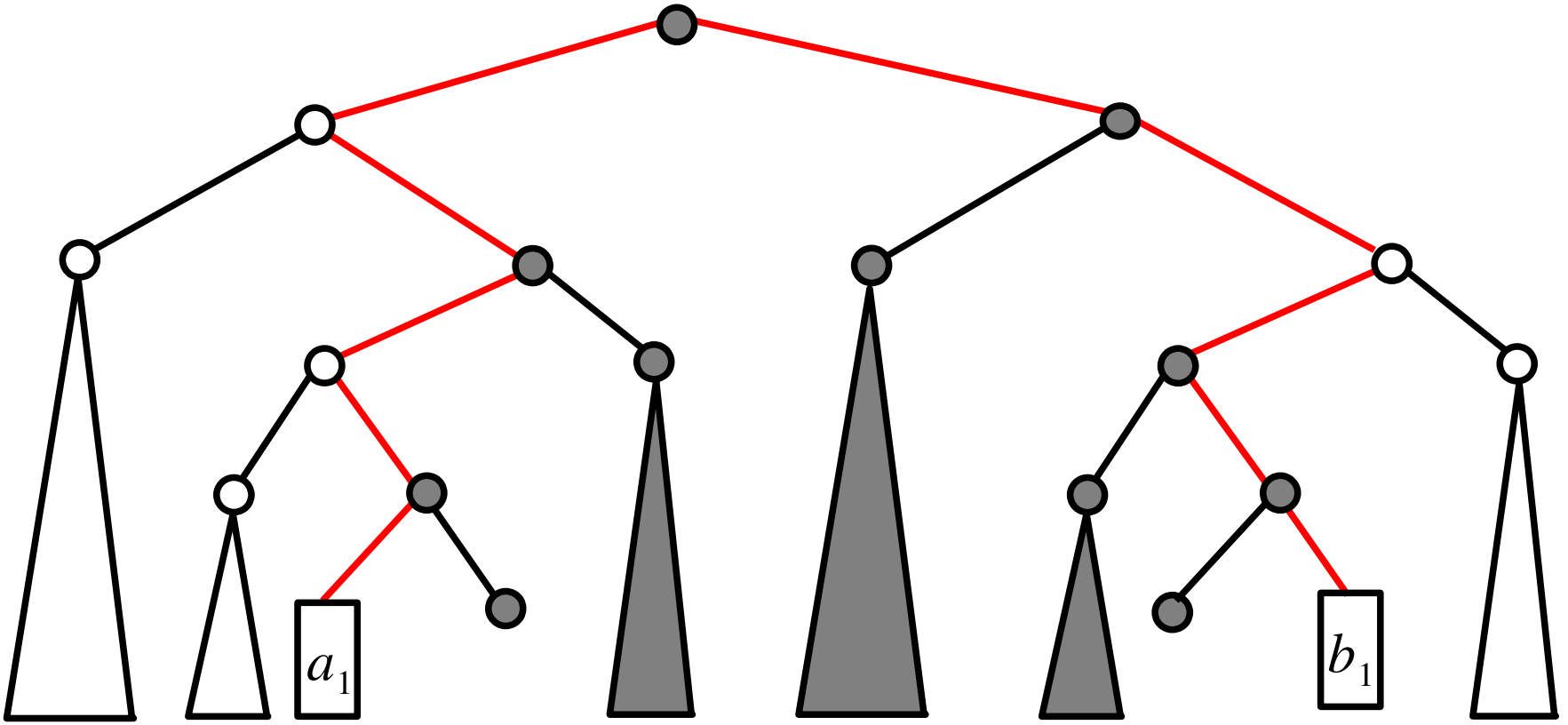
<u>Zur Erinnerung:</u>
Wir suchen in einer Punktmenge P nach allen Punkten, deren
- x_1-Koordinate zwischen a_1 und b_1 liegt
- x_2-Koordinate zwischen a_2 und b_2 liegt

Wir bauen als Basisstruktur einen binären Suchbaum bzgl. der x_1-Koordinaten der Punkte in P auf.

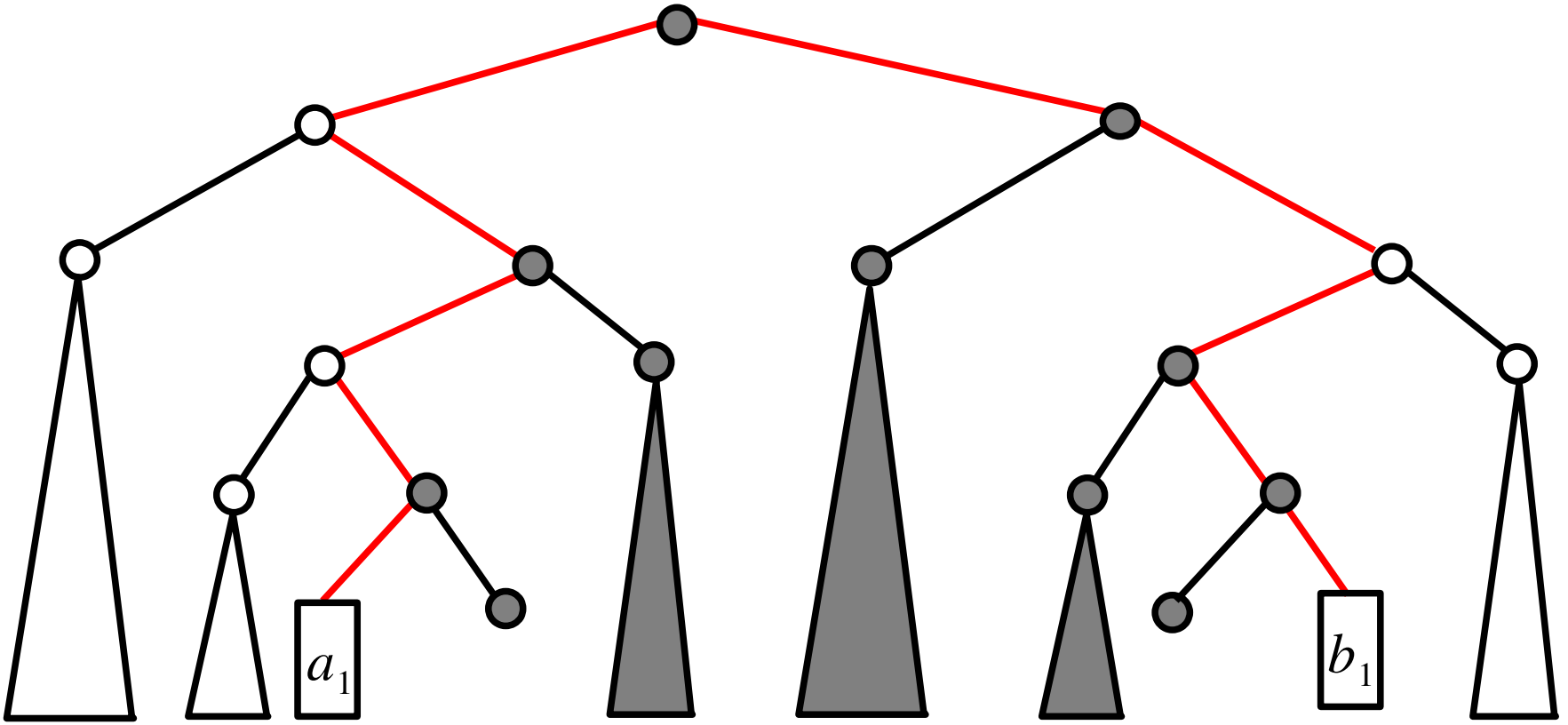
Darin suchen wir dann a_1 und b_1.



Für jeden Punkt $q=(x_1, x_2)$, der sich in einem grauen Knoten auf einem der Suchpfade befindet, gilt $a_1 \leq x_1 \leq b_1$.
Wir müssen jeweils testen, ob auch $a_2 \leq x_2 \leq b_2$ gilt.



Bleiben noch die grauen Teilbäume. Wir könnten diese natürlich vollständig traversieren. Aber dann besuchen wir möglicherweise sehr viele Knoten, ohne dass ein Punkt zurückgegeben wird.



Aus diesem Grund rüsten wir unseren Basisbaum in jedem Knoten u mit einer sekundären Datenstruktur $S(u)$ aus.

Dabei ist $S(u)$ wieder ein binärer Suchbaum, der als Schlüssel alle Punkte enthält, die im Basisbaum im Teilbaum mit Wurzel u gespeichert sind.

Allerdings sind die Punkte in $S(u)$ nach den x_2-Koordinaten geordnet.

Somit muss für jeden grauen Teilbaum mit Wurzel u im Basisbaum eine eindimensionale Bereichsanfrage für $S(u)$ bearbeitet werden.

Um einen zweidimensionalen Bereichsbaum aufzubauen, sortieren wir die Punkte in P zuerst nach x_1-Koordinaten und dann nach x_2-Koordinaten.

Dann benötigen wir pro Ebene des Basisbaumes $O(n)$ Zeit.

Da der Basisbaum $O(\log n)$ Ebenen hat, kann der Bereichsbaum in $O(n \log n)$ Zeit aufgebaut werden.

Die Sekundärbäume belegen pro Ebene des Basisbaums $O(n)$ Speicher.

Damit kommen wir insgesamt mit $O(n \log n)$ Speicher aus.

Um den **Aufwand für eine Bereichsanfrage** auf Bereichsbäumen abzuschätzen, müssen wir zuerst den Aufwand für die Bearbeitung der Bereichsanfrage auf dem Basisbaum berücksichtigen.

Diese kann in $O(\log n)$ Zeit abgearbeitet werden und liefert $O(\log n)$ Sekundärbäume, für die jeweils auch eine Bereichsanfrage gestellt wird.

Jede Bereichsanfrage auf einem Sekundärbaum $S(u)$ kann in $O(\log n + k(u))$ Zeit bearbeitet werden, wobei $k(u)$ die Anzahl der zurückgegebenen Punkt aus $S(u)$ ist.

Somit kommen wir insgesamt mit $O(\log^2 n + k)$ Zeit aus.

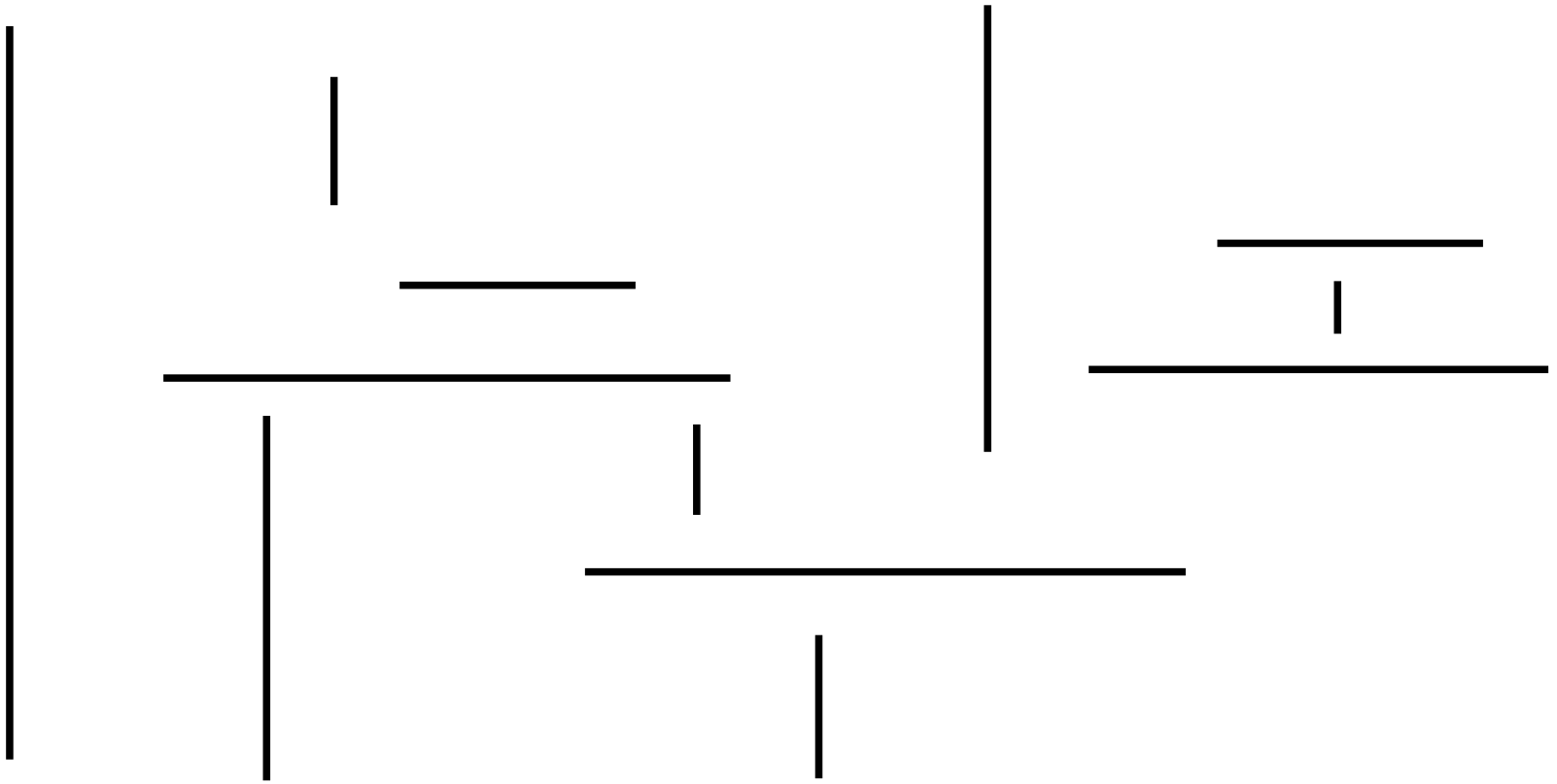
Zusammenfassung

Aufbau des Bereichsbaums:	$O(n \log n)$
Anfragen:	$O(\log^2 n + k)$
Speicherplatzbedarf:	$O(n \log n)$

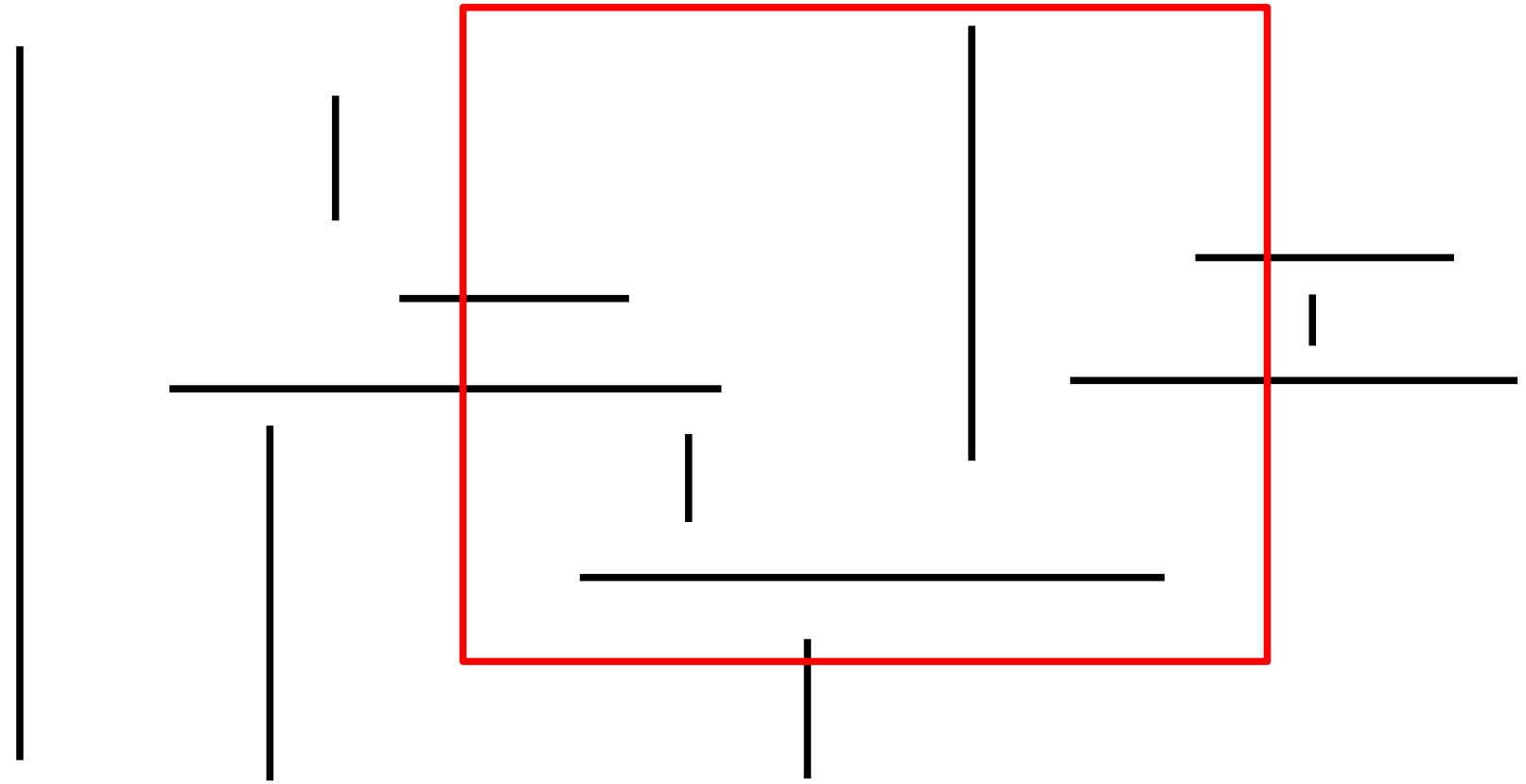
Es ist möglich, den Aufwand für Anfragen auf $O(\log n + k)$ zu reduzieren.

5. Anfragen für komplexere Objekte

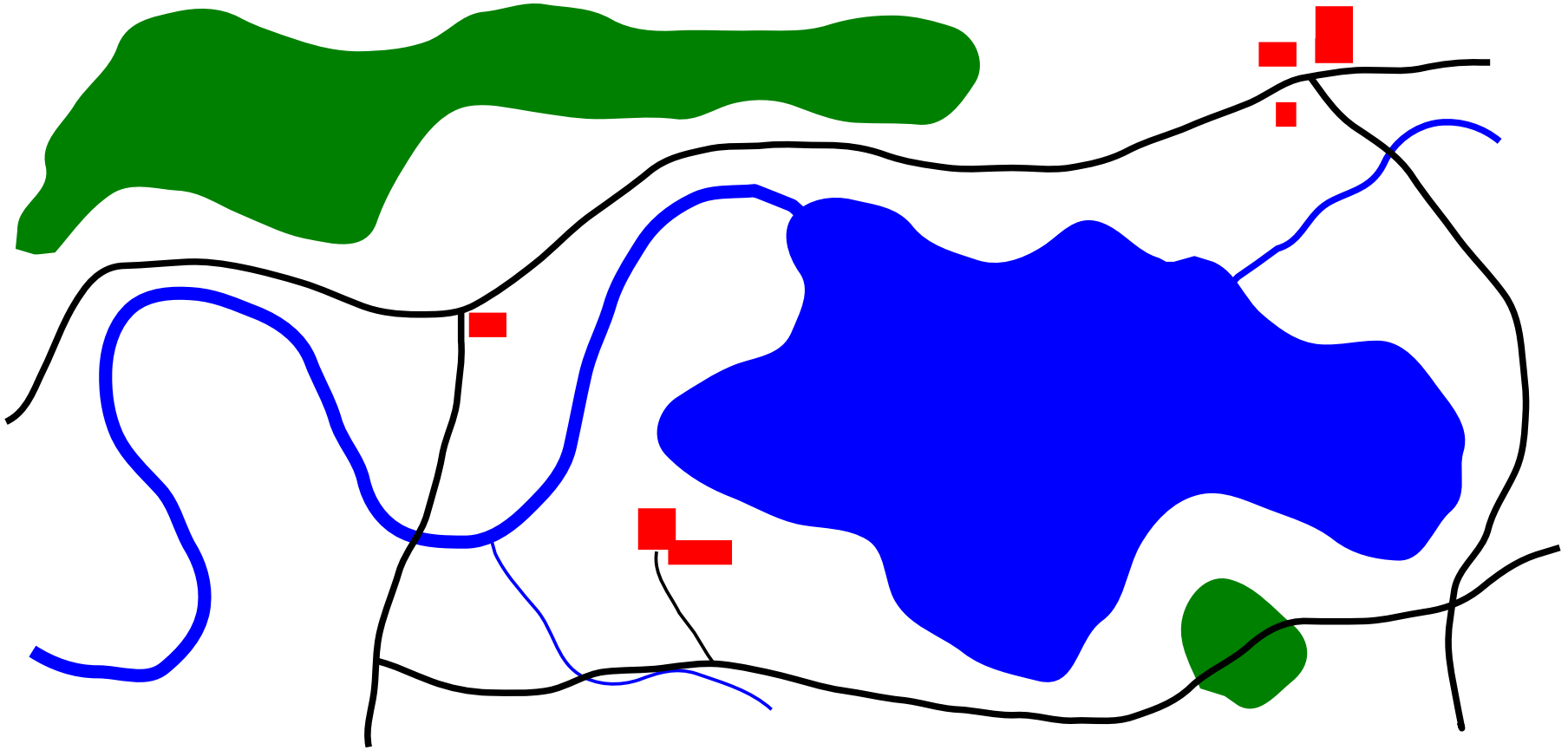
Gegeben ist eine Menge *S* von **achsenparallelen Strecken**.



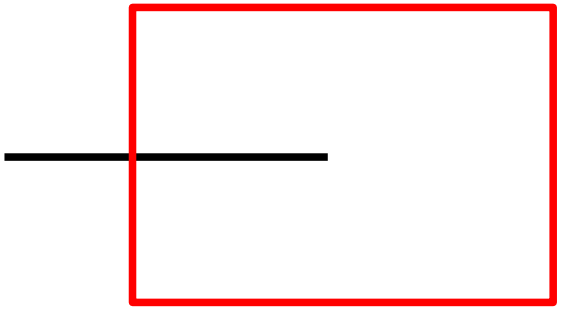
Gesucht sind diejenigen Elemente aus S, die einen nicht leeren Durchschnitt mit einem rechteckigen Anfragebereich haben.



Solche Anfragen treten bei der Berechnung von rechteckigen Ausschnitten komplexer Objekte wie etwa Landkarten oder Schaltkreisen auf.

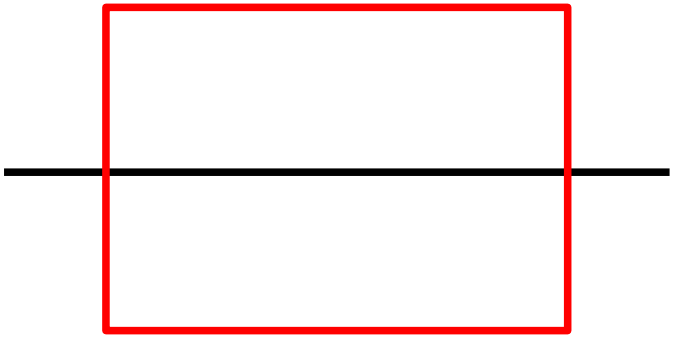


Es gibt **zwei Typen** von Strecken, die den Anfragebereich schneiden:



Typ 1:

Einer der Endpunkte der Strecke liegt im Anfrage-bereich.



Typ 2:

Die beiden Endpunkte der Strecke liegen außerhalb des Anfragebereichs.

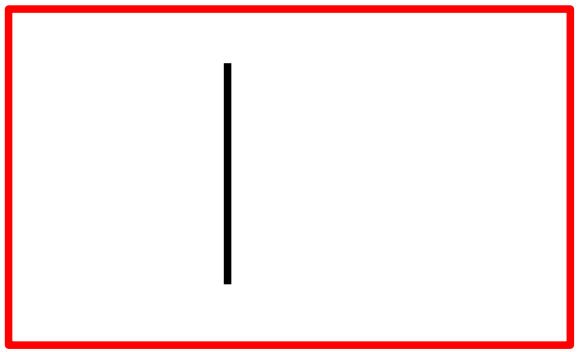
Wir werden beide Typen getrennt betrachten.

Sei $P(S)$ die Menge der Endpunkte der Strecken in S.

Wir bauen für $P(S)$ einen **zweidimensionalen Bereichsbaum** auf.

Damit können wir **alle Strecken vom Typ 1** finden.

Beachten:

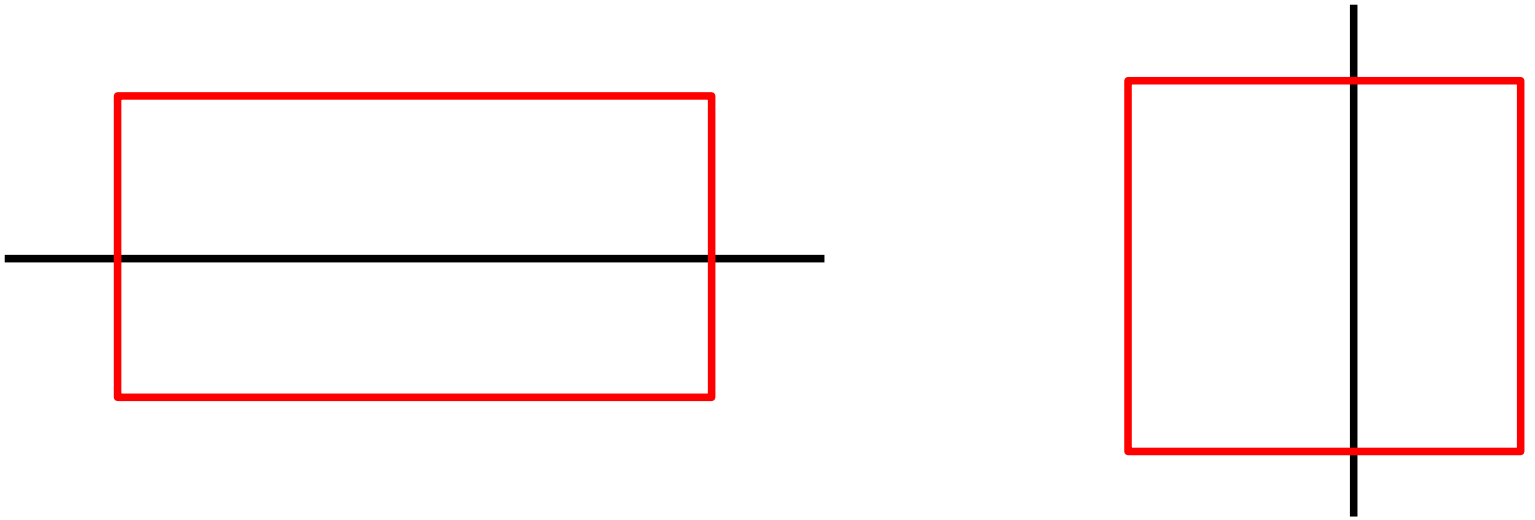


Diese Strecke darf **nur ein Mal** zurückgegeben werden, obwohl man im Bereichsbaum beide Endpunkte findet.

Zusammenfassung für Strecken vom Typ 1

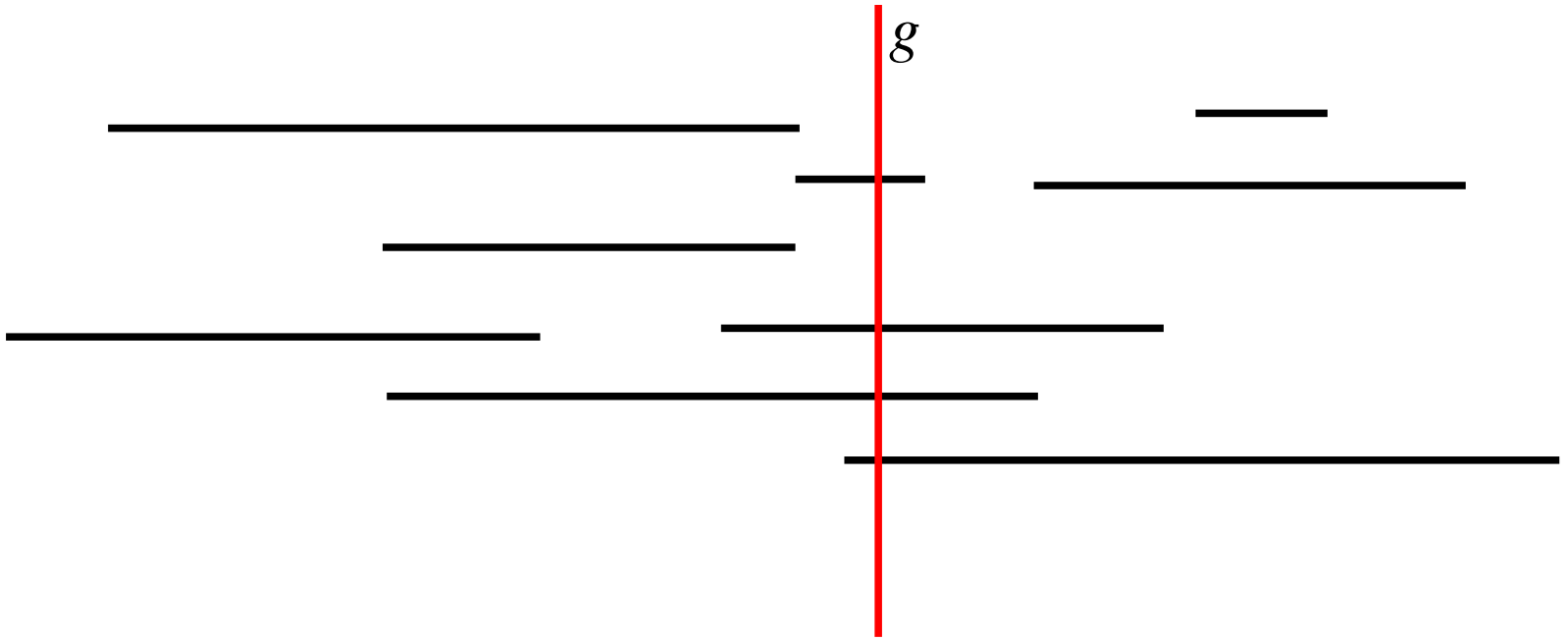
Aufbau des Bereichsbaums:	$O(n \log n)$
Anfragen:	$O(\log n + k)$
Speicherplatzbedarf:	$O(n \log n)$

Bei Strecken vom Typ 2 genügt es, alle Strecken zu finden, die die linke Seite oder die untere Seite des Anfragerechtecks schneiden.



Unsere Aufgabe besteht also darin, alle waagerechten (senkrechten) Strecken in S so in einer geeigneten Datenstruktur abzulegen, dass wir für eine senkrechte (waagerechte) **Anfragestrecke q** schnell alle Strecken finden können, die q schneidet.

Wir betrachten hier nur eine Datenstruktur für die etwas einfachere Situation einer senkrechten **Anfragegeraden *g***.



Jetzt haben wir es im Grunde genommen mit einem eindimensionalen Problem zu tun.

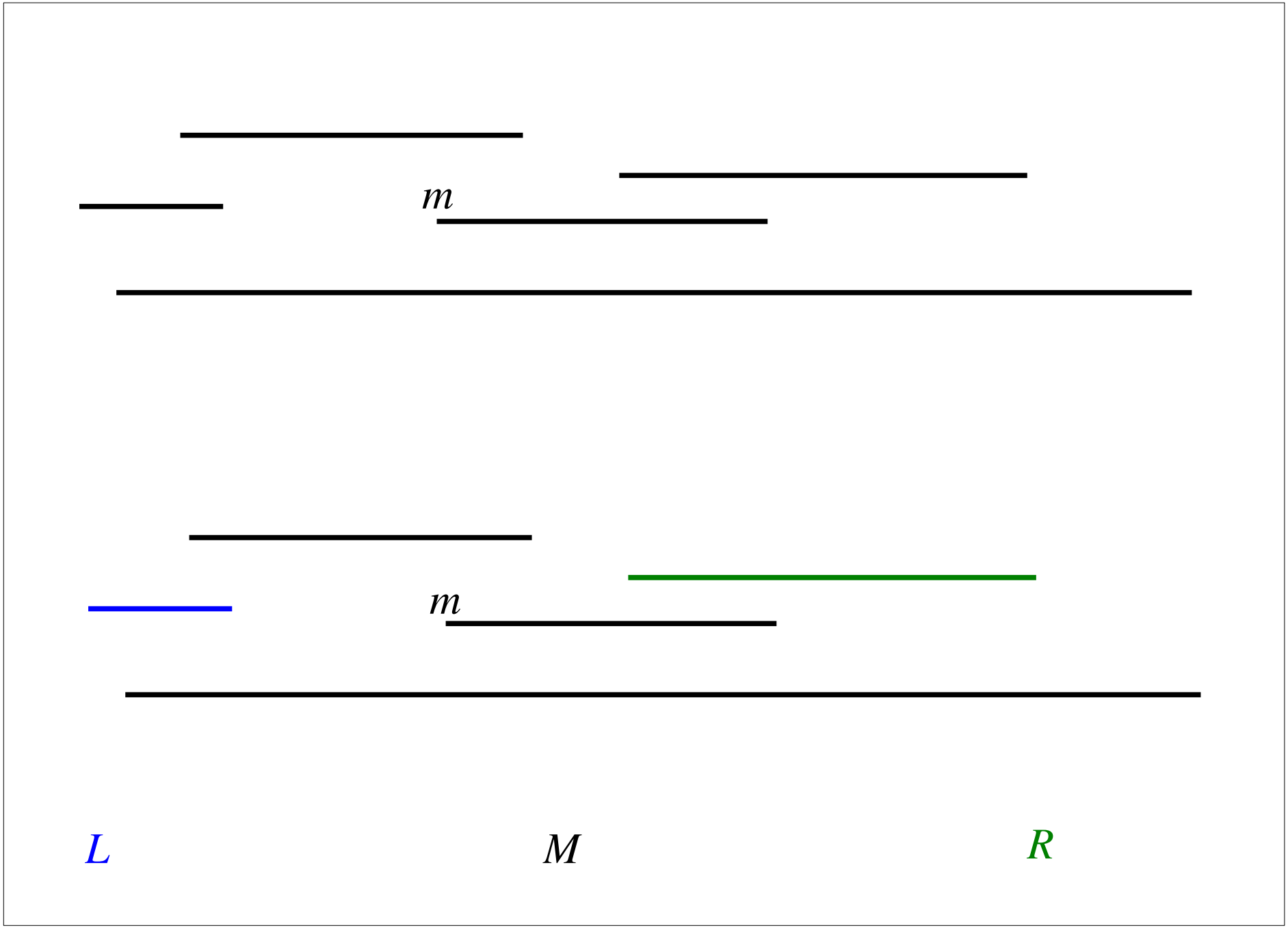
Grundidee

Wir **sortieren** als Erstes die Endpunkte der Strecken nach wachsenden x-Koordinaten.

Dann suchen wir den **Median** m der Endpunkte heraus.

Wir bilden drei Teilmengen:

L	Strecken, deren rechter Endpunkt links von m liegt.
M	Strecken, bei denen m zwischen linkem und rechtem Endpunkt liegt.
R	Strecken, deren linker Endpunkt rechts von m liegt.


m
m
L
M
R

Nach diesem Schema bauen wir einen Binärbaum (**Intervallbaum** genannt) auf:

In die Wurzel kommen die Strecken aus M.

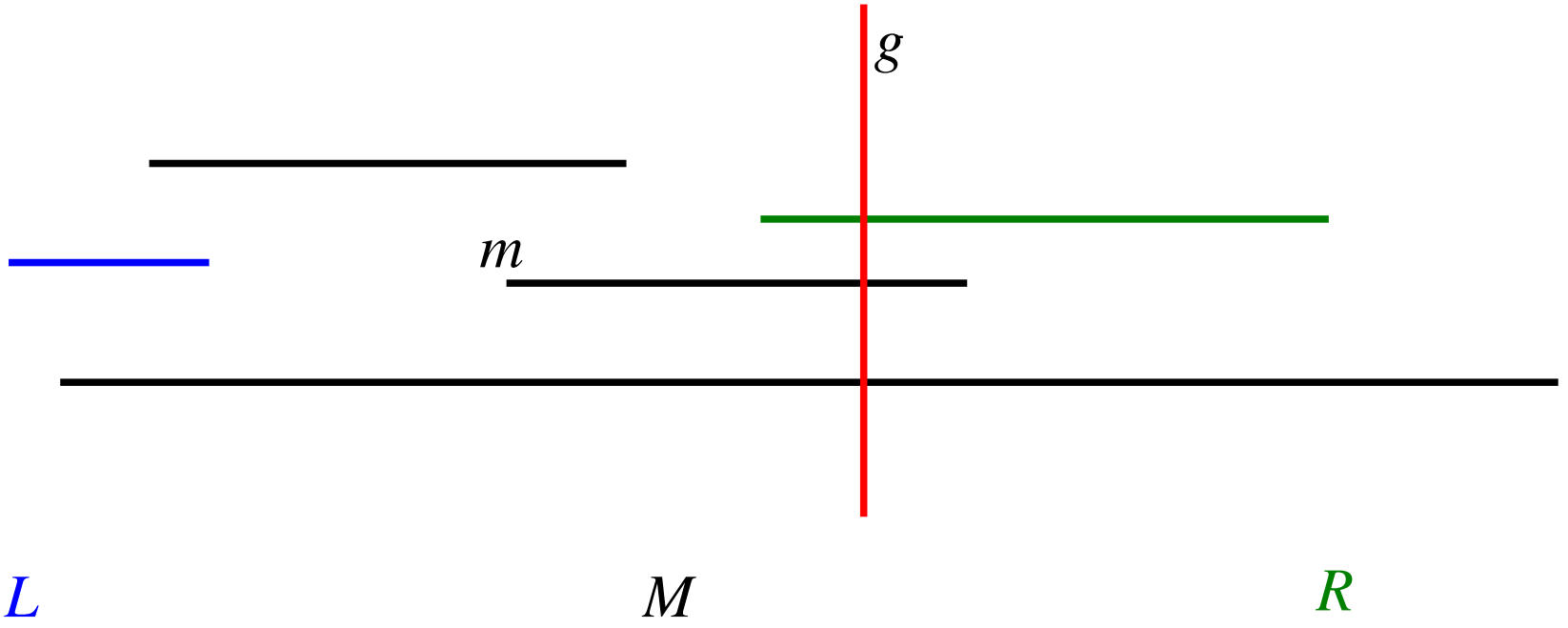
Den linken Teilbaum bauen wir aus den Strecken in L.

Den rechten Teilbaum bauen wir aus den Strecken in R.

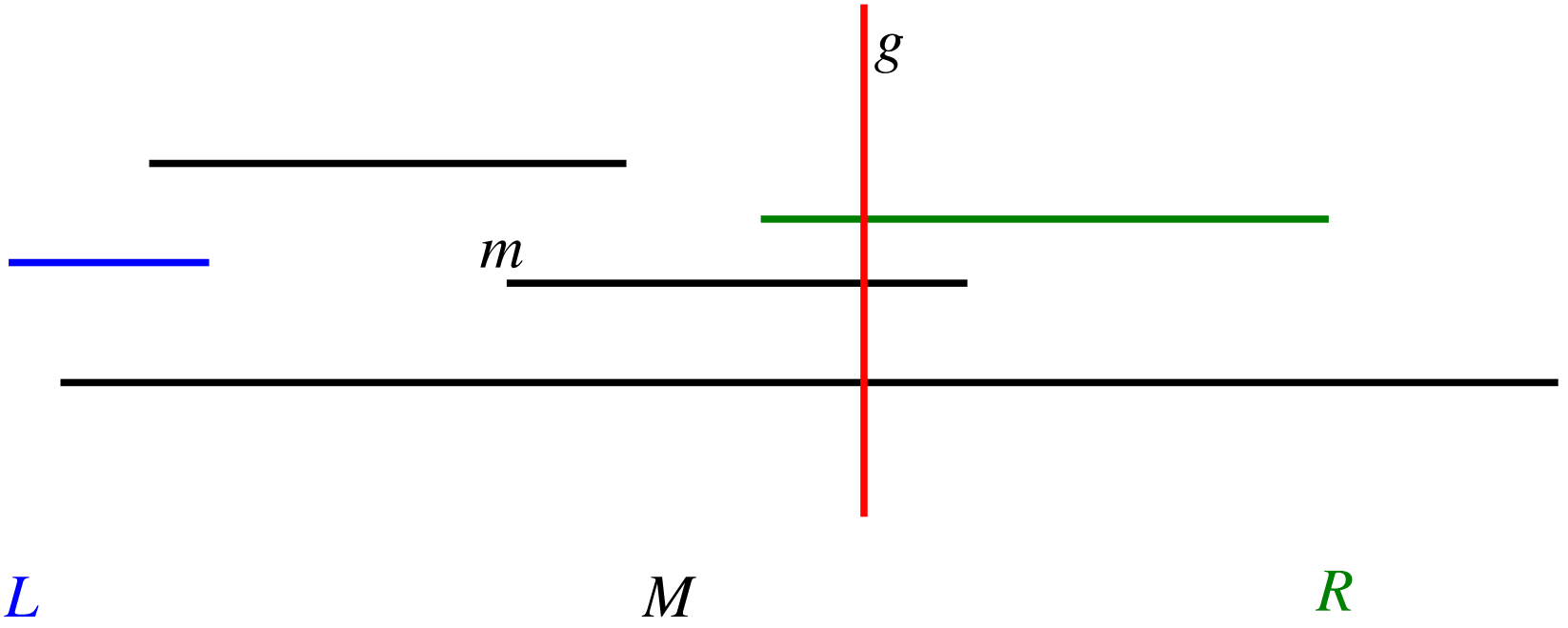
Damit ist mit jedem Knoten eine x-Koordinate assoziiert und eine Menge von Strecken.

Wenn wir nun eine senkrechte Anfragegerade g bearbeiten wollen, dann können wir leicht diejenigen Strecken finden, die g schneidet.

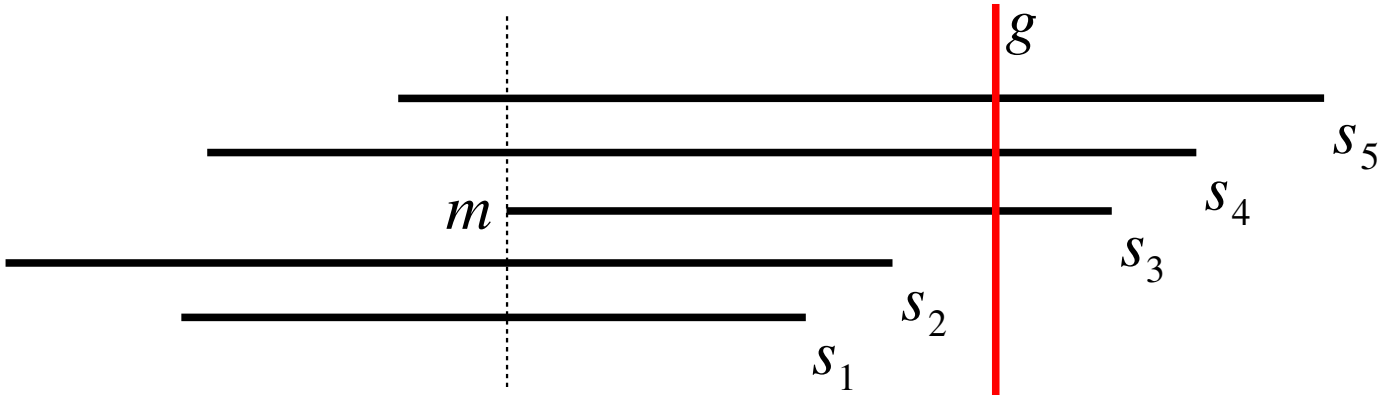
Als Erstes prüfen wir, ob g links oder rechts vom Median in der Wurzel des Baumes liegt.



Wenn g links (rechts) von m liegt, dann finden wir alle mit der Wurzel assoziierten Strecken, die g schneidet und gehen dann in den linken (rechten) Teilbaum der Wurzel.



Dabei ist es zweckmäßig, die mit einem Knoten assoziierten Strecken je ein Mal nach x-Koordinaten der linken Endpunkte und nach x-Koordinaten der rechten Endpunkte sortiert abzulegen.



Dann ist es möglich bei k in einem Knoten zurückzugebenden Strecken mit $O(1 + k)$ Zeit auszukommen.

Zusammenfassung zur Verarbeitung einer Anfragegerade

Aufbau des Intervallbaums:	$O(n \log n)$
Anfragen:	$O(\log n + k)$
Speicherplatzbedarf:	$O(n)$

Aus den vorgestellten Ideen kann man eine Datenstruktur konstruieren, mit deren Hilfe wir für rechteckige Anfragebereiche diejenigen Strecken ermitteln können, die diesen Anfragebereich schneiden.

Aufbau des Datenstruktur:	$O(n \log n)$
Anfragen:	$O(\log^2 n + k)$
Speicherplatzbedarf:	$O(n \log n)$