Räumliche Zugriffsverfahren

Referent: Farzin Ranjbar Mirzakhani

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Agenda

- Motivation
- Eindimensionale Indexstruktur
  - Binärbaum
- Mehrdimensionale Indexstrukturen
  - KD-Tree
  - Quad-Tree
  - Grid-File
  - R-Tree

Motivation

Räumliche Zugriffsverfahren →
Unterschiedliche Indexierungsverfahren

Bilden von Indexstrukturen = Strukturierungen von Daten in einer Form, die eine spätere Suche optimal unterstützt.

In unserem Fall: Strukturierung von räumlichen Daten → Die Unterstützung der Suche und damit Zugriff auf räumlichen Daten.

Motivation

Häufig werden B-Baum und dessen Varianten als Indexstrukturen in DB eingesetzt.

Exakte Suche = logarithmischer Aufwand

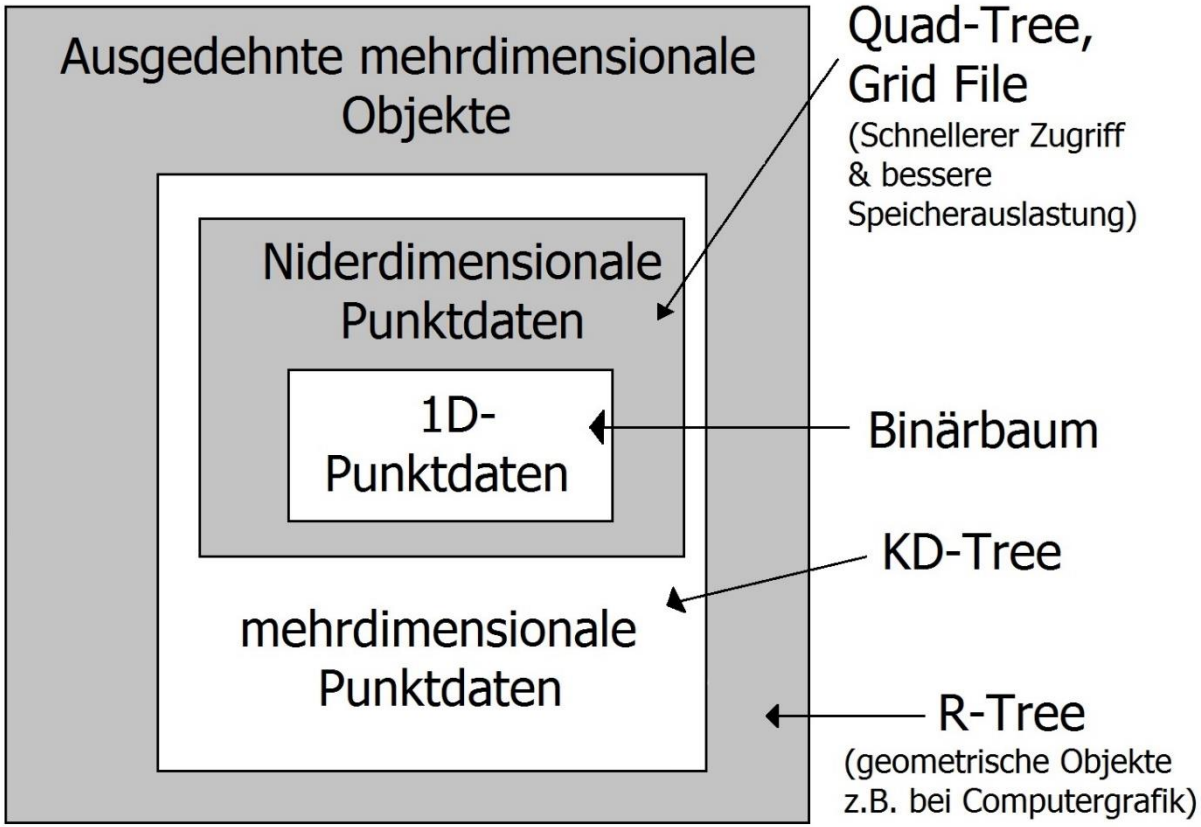
Unterstützt nur eindimensionale Suche.

Eine Dimension kann ein Schlüsselwert sein, nach dem gesucht wird z.B. Alter einer Person oder jeder einzelne Feature-Wert eines Medien-Objektes (Bei einem Bild der Grau- oder Kontrastwert …)

Motivation

Wir wollen mit Anfragen im hochdimensionalen Raum arbeiten, weil Ähnlichkeitssuche in der Regel auf einer Vielzahl von Feature-Werten basiert z.B. Grauwert, Kontrast … → Einführung anderer Indexstrukturen als der B-Baum

Motivation

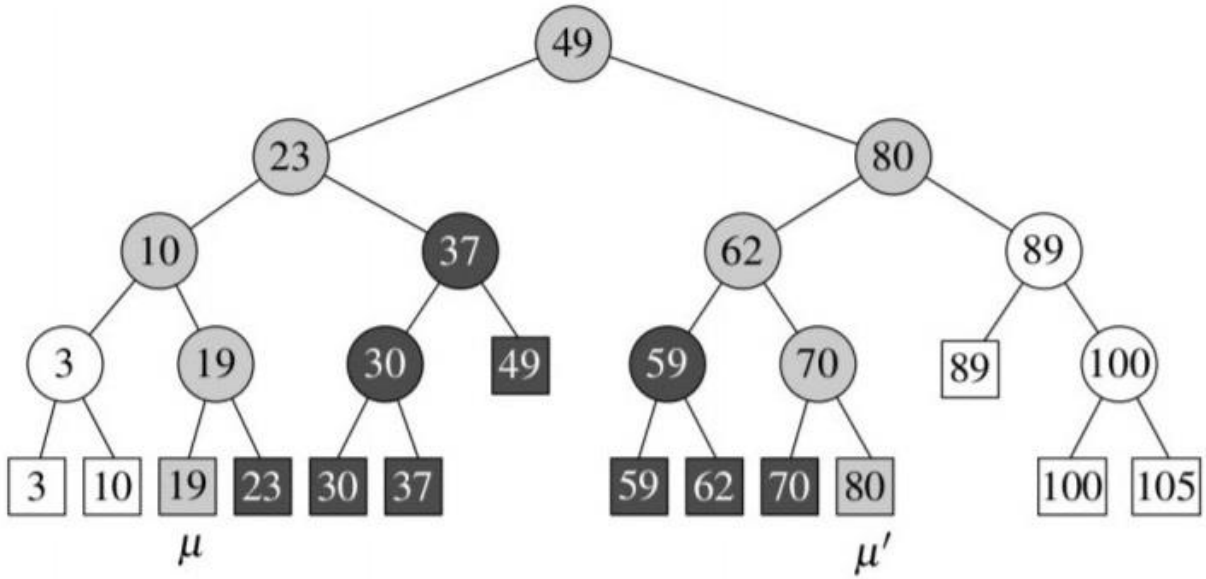

Ausgedehnte mehrdimensionale Objekte
Niderdimensionale Punktdaten
1D-Punktdaten
mehrdimensionale Punktdaten
Quad-Tree, Grid File
(Schnellerer Zugriff & bessere Speicherauslastung)
Binärbaum
KD-Tree
R-Tree
(geometrische Objekte z.B. bei Computergrafik)

Eindimensionale Indexstruktur - Binärbaum

Binärbaum – Aufbau & Bereichsanfrage

– Datenstruktur: balancierter binärer Suchbaum
– gegeben: Suchintervall [x,x'], 1D Punktemenge $P := \{p_1, p_2, \ldots, p_n\}$
– Ausgabe: alle Punkte im Suchbaum in [x,x']

→ Beispiel: [x, x'] =[18, 77]



– weiße Knoten: nicht besucht da außerhalb [x,x']
– graue Knoten: besucht, prüfe individuell, ob Knoten in [x,x']
– schwarze Knoten: enthalten in [x,x']

Mehrdimensionale Indexstrukturen

KD-Tree

KD-Tree - Einführung

Gegeben sei eine 2D Punktmenge $P := \{p_1, p_2, \ldots, p_n\}$

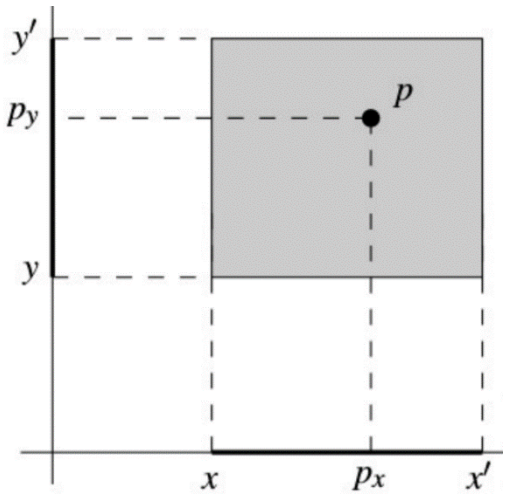
Gib alle Punkte bzw. Datenobjekte, die sich im Suchintervall $[x : x'] \times [y : y']$ befinden, z.B. {Alter, Einkommen}

Naive Suche: Teste für jeden Punkt, ob er im Intervall liegt → O(n)

$$p := (p_x, p_y)$$

$$p_x \in [x : x']$$

$$p_y \in [y : y']$$



KD-Tree - Einführung

Schlauerer Ansatz: Objekte der DB in einem Binärbaum (balanciert) einordnen. Darin nach Punkte suchen, die im Suchintervall liegen.

Problem: Man kann in einem Binärbaum jeweils nach nur einem Sortierschlüssel - z.B. nach X- oder Y-Koordinate – absteigen!

→ Einführung und Nutzung von KD-Tree

KD-Tree - Einführung

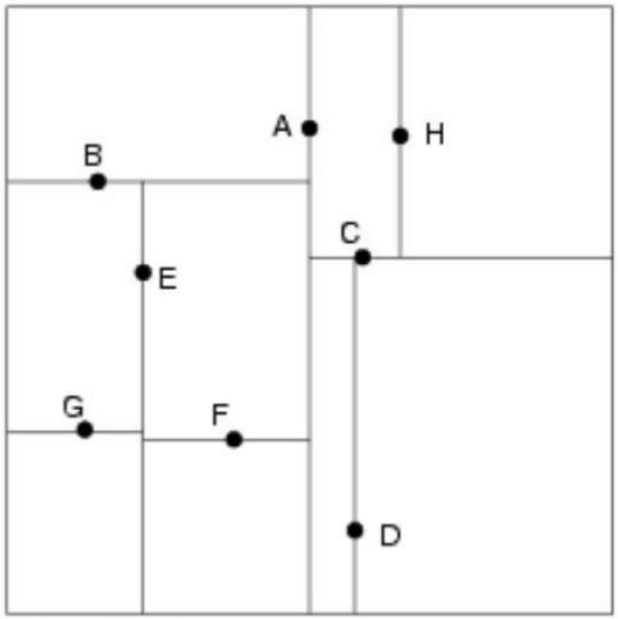
Mit einem KD-Tree wird die Suche von K-dimensionalen Punkten in einem durch K Intervalle beschriebene Bereich unterstützt.

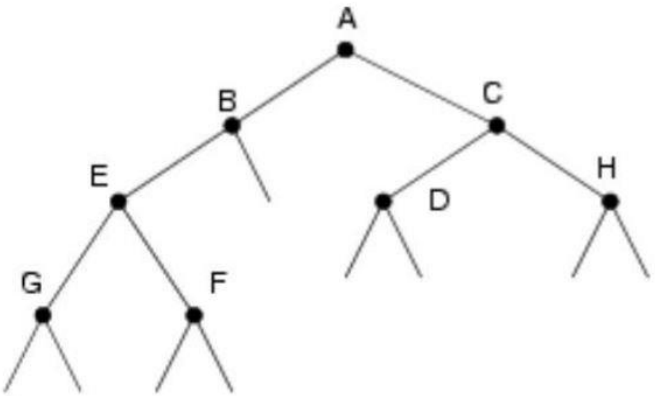
Für unser Beispiel mit einer 2D-Punktmenge benötigen wir einen KD-Tree mit K=2 (2D-Tree).
→ Suche nach 2-dimensionalen Punkten in einem durch 2 Intervalle beschriebenen Bereich möglich (hier: X-, Y-Koordinate).

KD-Tree – Idee & Konstruktion

Klassischer KD-Tree:

- Es wird abwechselnd nach X- bzw. Y-Koordinate zerlegt.
- Sortierschlüssel ändert sich mit jeder Hierarchieebene.
- Jeder Knoten zerlegt den Raum (achsenparallel) in 2 Teile.





KD-Tree – Idee & Konstruktion

Klassischer KD-Tree zu unbalanciert bei schlechter Auswahl von Punkten (aus bereits vorhandenen Punkten)
→ Damit der KD-Tree für bereits vorhandenen Punkten balanciert: Einführung und Nutzung von „adaptiver KD-Tree“

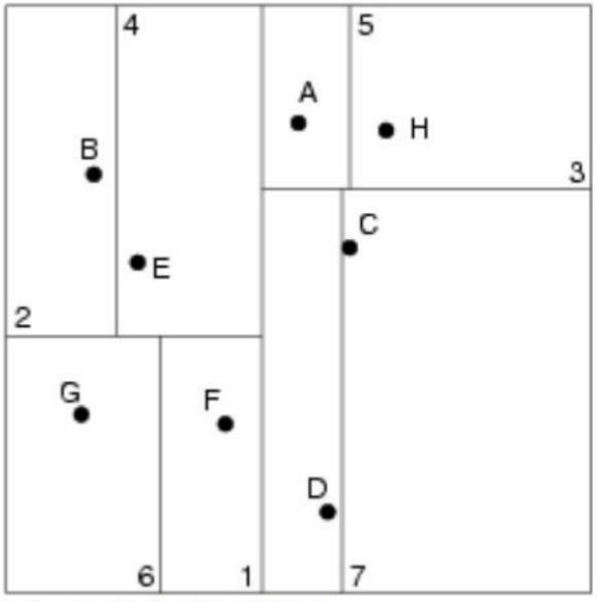
Adaptiver KD-Tree – Idee & Konstruktion

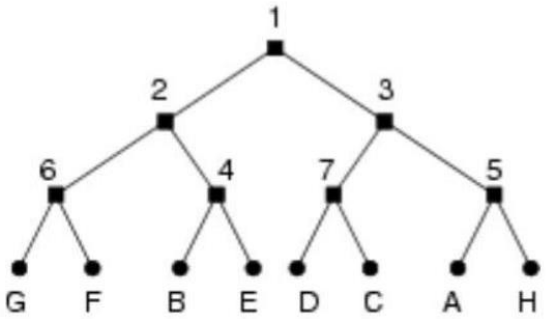
Adaptiver KD-Tree:

- Die innere Knoten bestehen nicht aus dem Datensatz.
- Innere Knoten sind selbst eingefügte Ebenen.

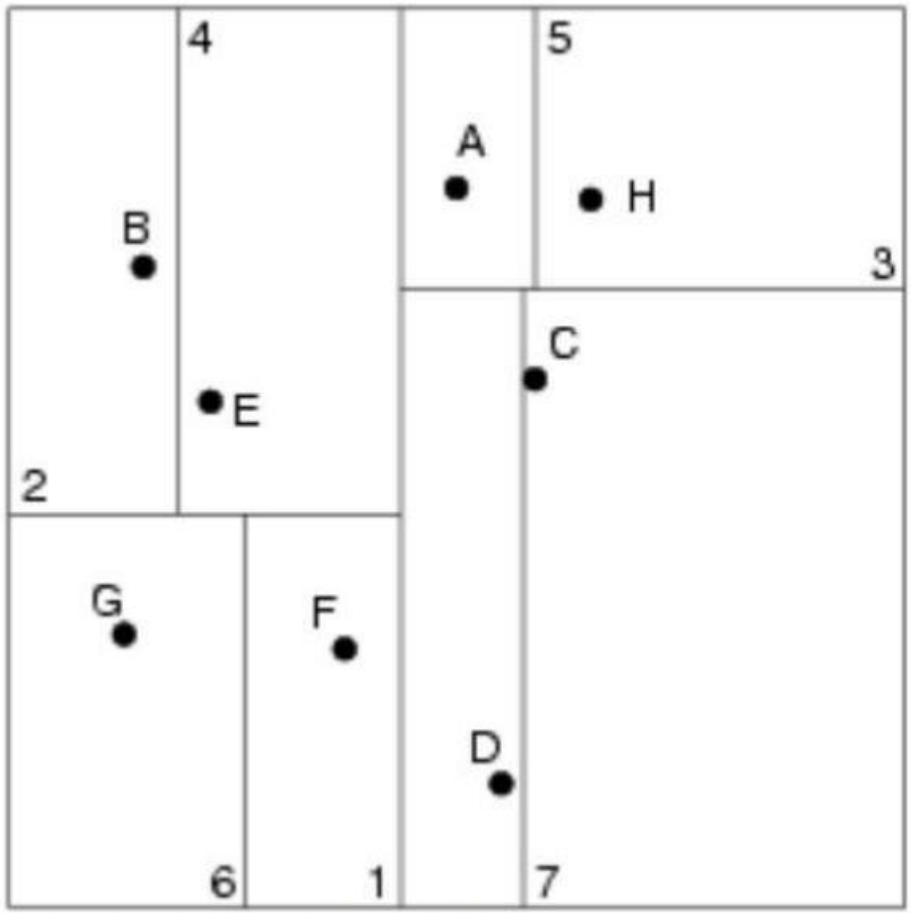
→ Jeder innerer Knoten (= Ebene) zerlegt den Raum achsenparallel in 2 Teile.

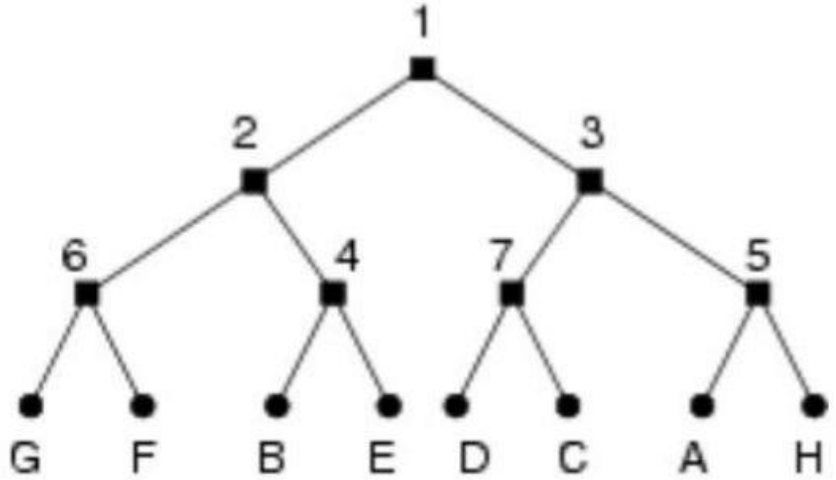
- Nur Blätter sind Datensätze.



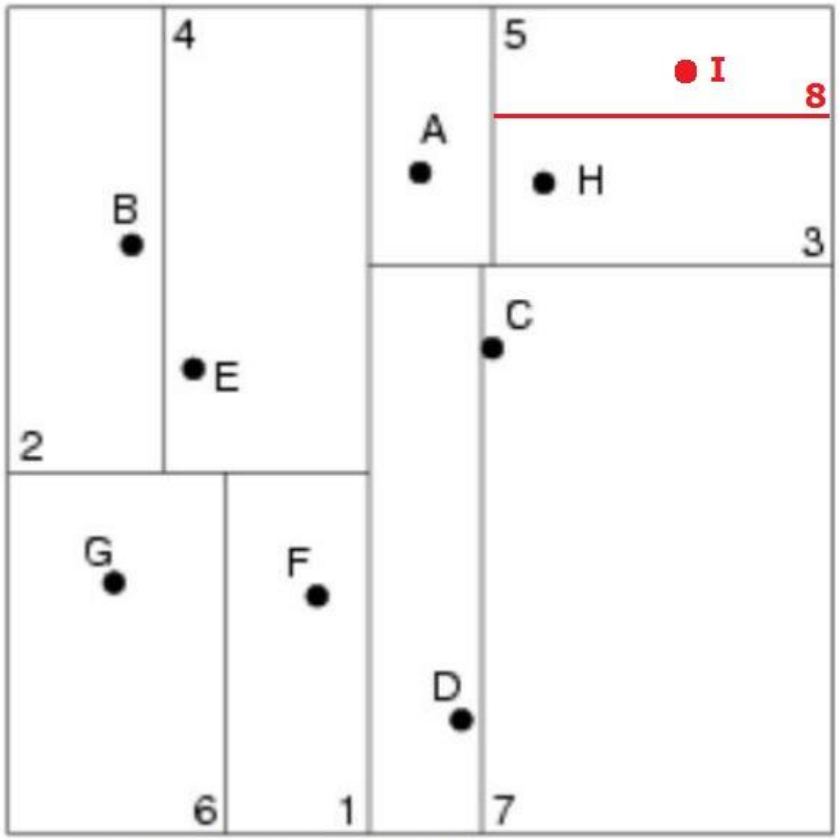


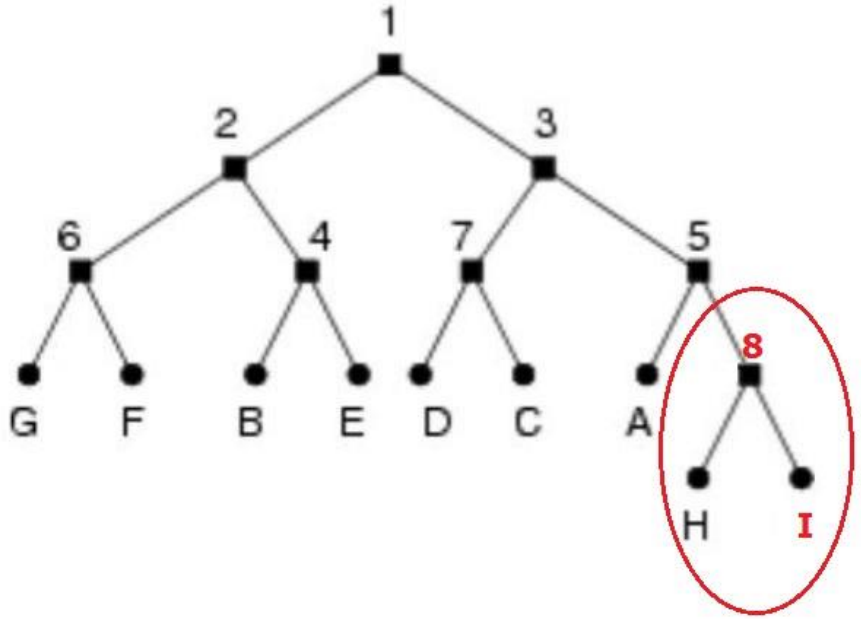
Adaptiver KD-Tree – Einfügen



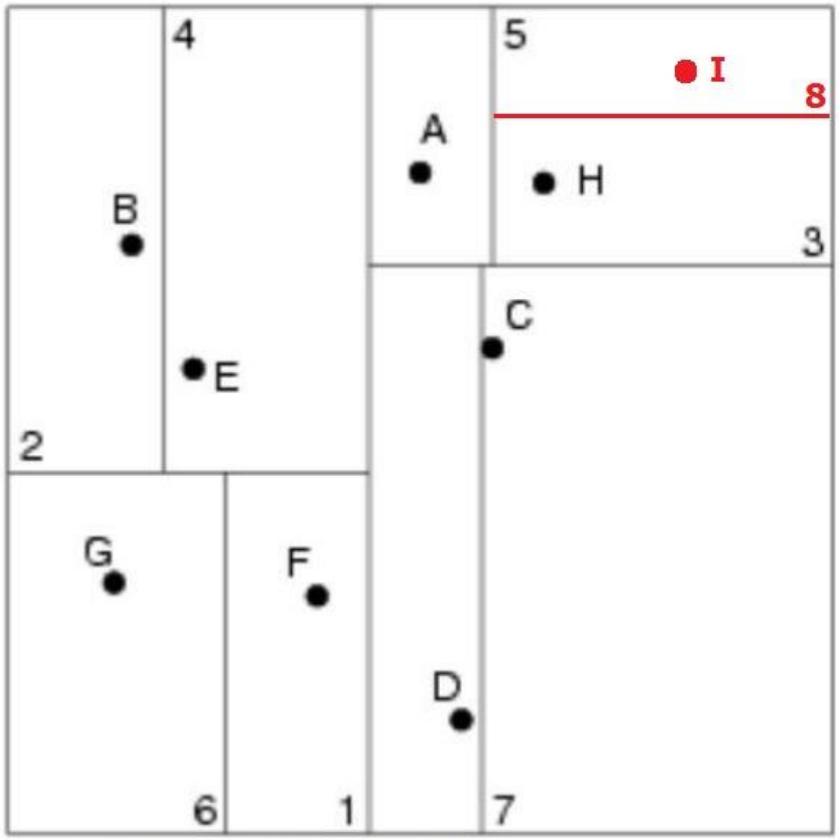


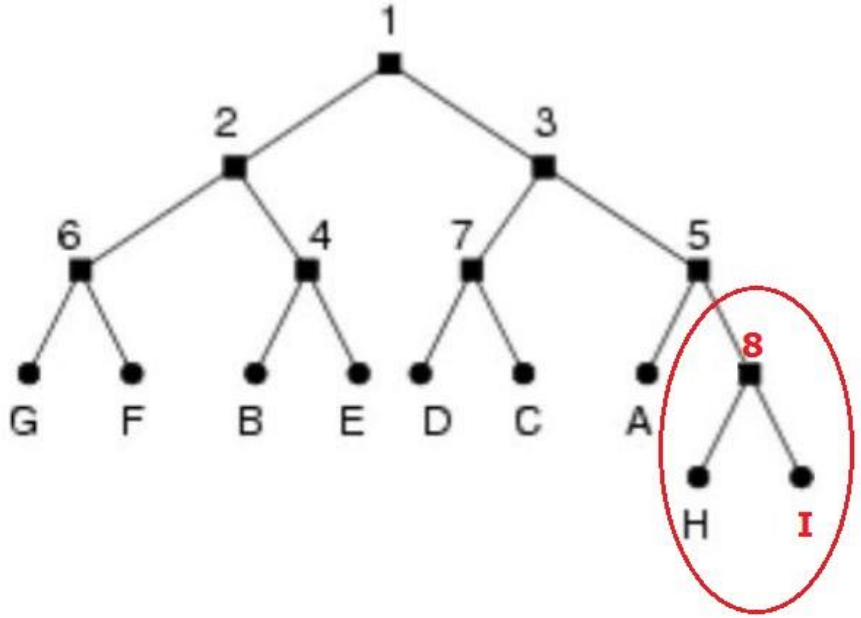
Adaptiver KD-Tree – Einfügen



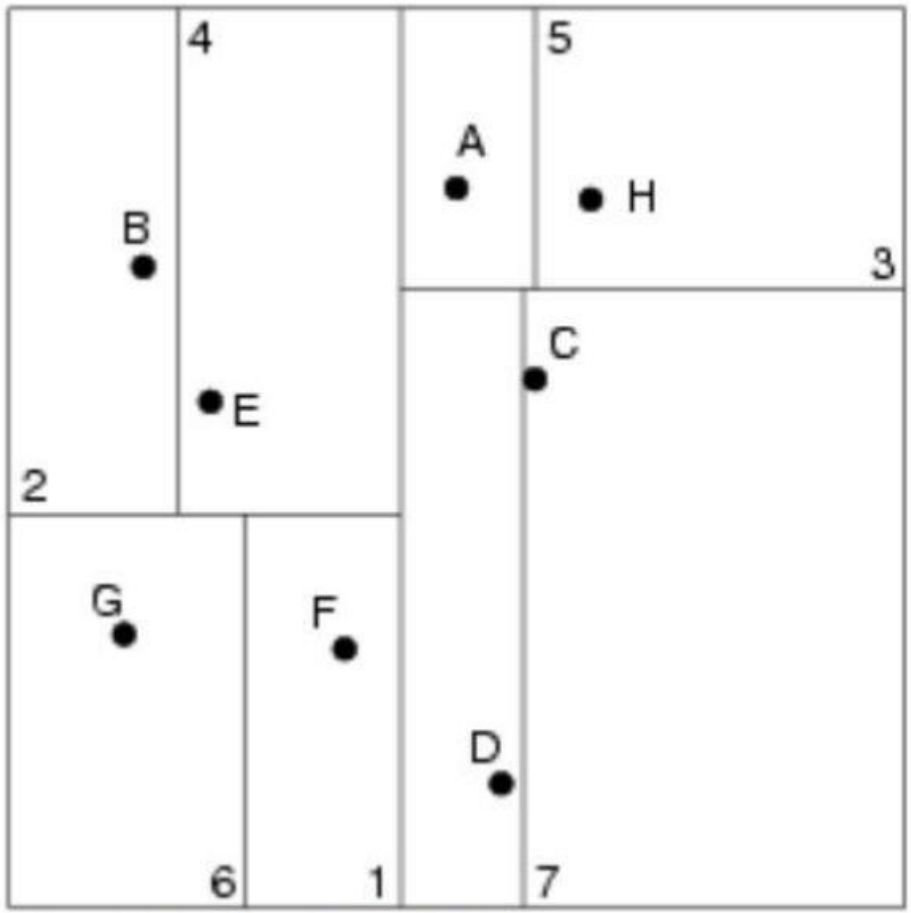


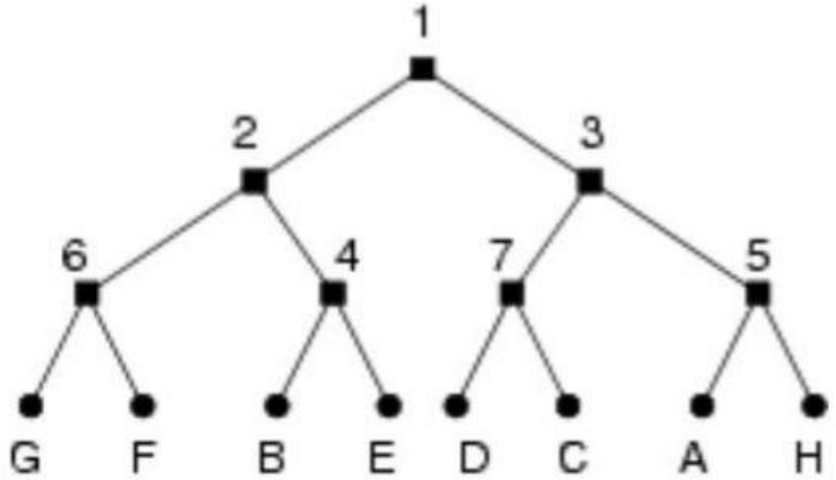
Adaptiver KD-Tree – Löschen



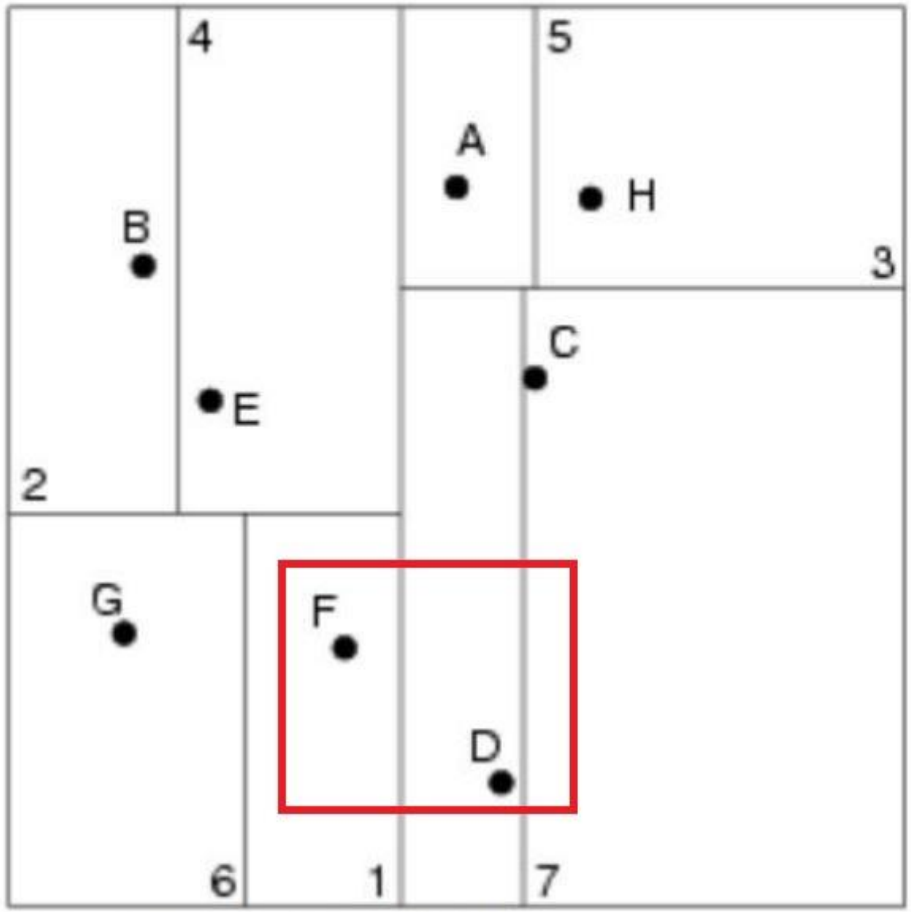


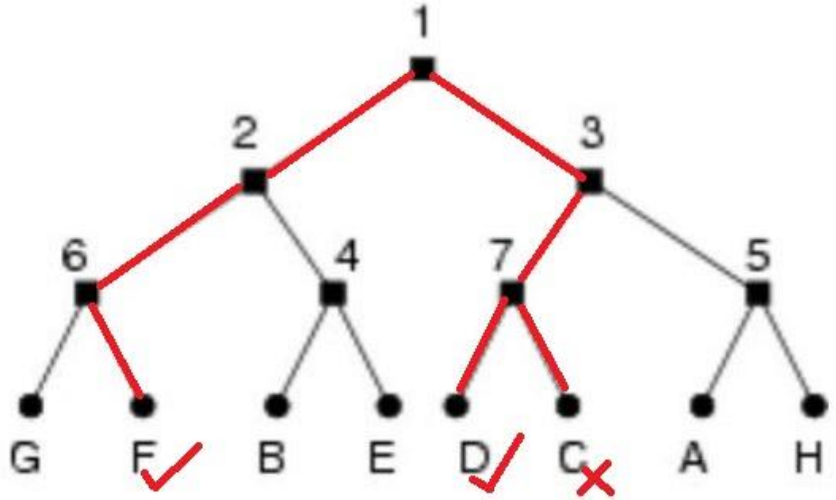
Adaptiver KD-Tree – Löschen





Adaptiver KD-Tree – Suche





KD-Tree – Bewertung

Relativ leicht zu implementierender Index.

Seine Form (auch initial) abhängig davon, in welche Reihenfolge die Punkte ausgewählt werden, die den Raum teilen.
→ Adaptiver KD-Tree löst das Problem.

Auch hier kann der Baum - anfänglich balanciert - schnell durch Einfüge- und Löschoperationen degenerieren.

Im Durchschnittsfall sind jedoch die Such- und Einfüge-Operationen günstig (Bei N Knoten nur $O(log2N)$)

Quad-Tree

Quad-Tree - Einführung

Bereits darauf eingegangen, wie mehrdimensionale Daten (Punktgeometrien) in einem binären Suchbaum abgelegt werden können. → Siehe KD-Tree: Bei Zuordnung immer jeweils nur einen Schlüsselwert anschauen.

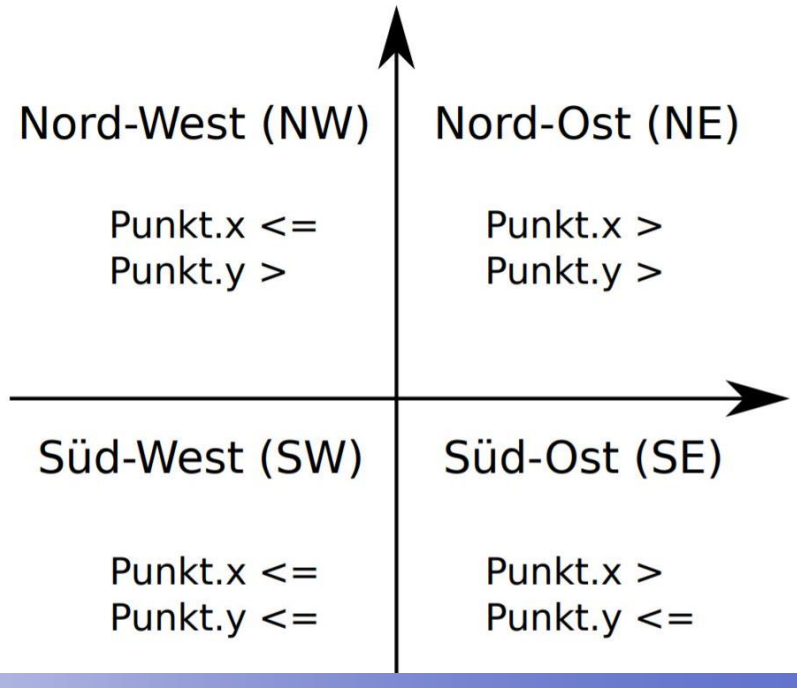
Eine entsprechende Zuordnung ist auch bei Verwendung beider Werte möglich → Voraussetzung hierfür: Die Erhöhung der Anzahl der Teilbäume.

Quad-Tree wie der KD-Tree für Strukturierung von Punkten in K-dimensionalen Räumen gedacht.

Quad-Tree - Einführung

Jedem Teilbaum eine bestimmte Kombination - bzgl. der Ordnung der Einzelwerte zueinander - zuordnen. →
Es resultieren 4 verschiedene Kombinationen.
Denen wird die eng. Abkürzung der Himmelsrichtungen zugeordnet.

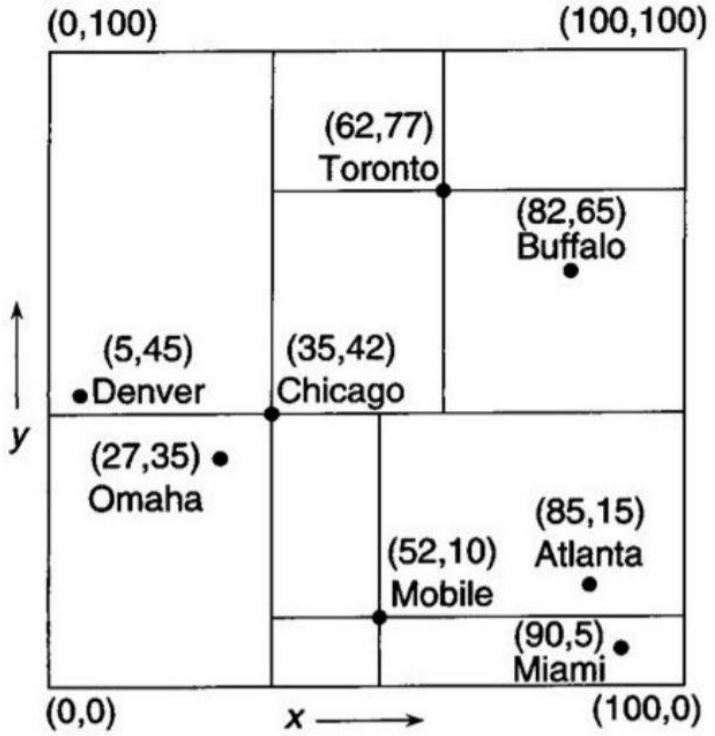
Viertel eines auf diese Weise unterteilten Knotens = Quadrant
→ Quad-Tree

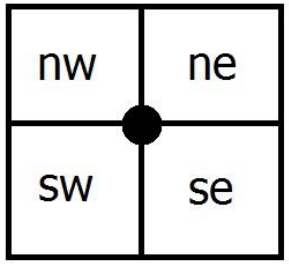


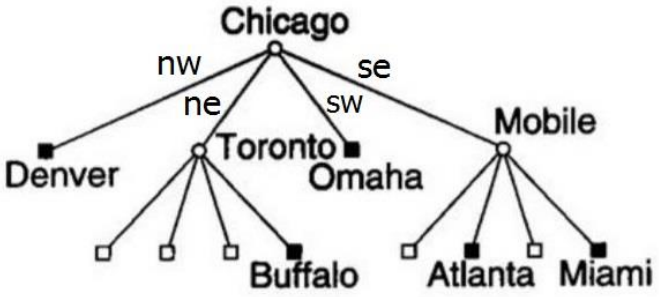
Quad-Tree - Aufbau

Point Quad (PQ)-Tree:

- Nutze Punkte zur Unterteilung der Ebene
- Ein Punkt hat 4 Nachfolger (NW, NE, SW, SE)



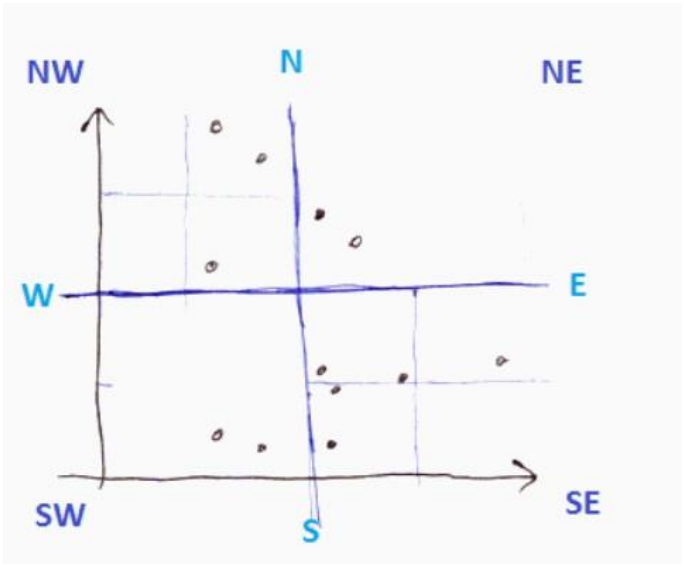


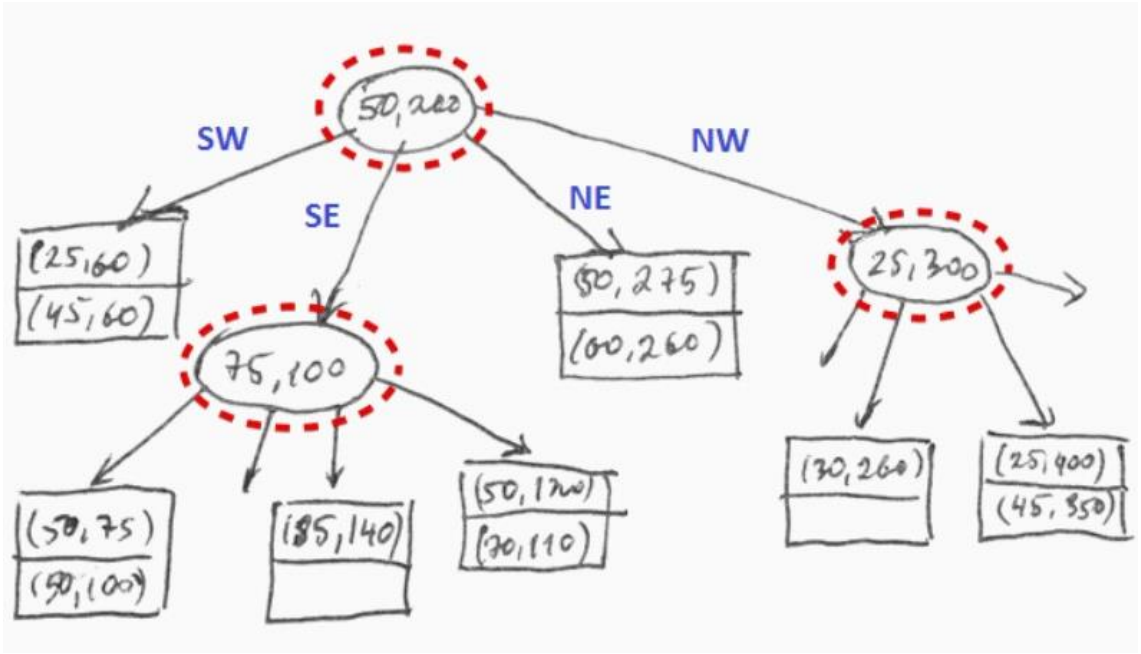


Quad-Tree - Aufbau

Point Region Quad (PRQ)-Tree:

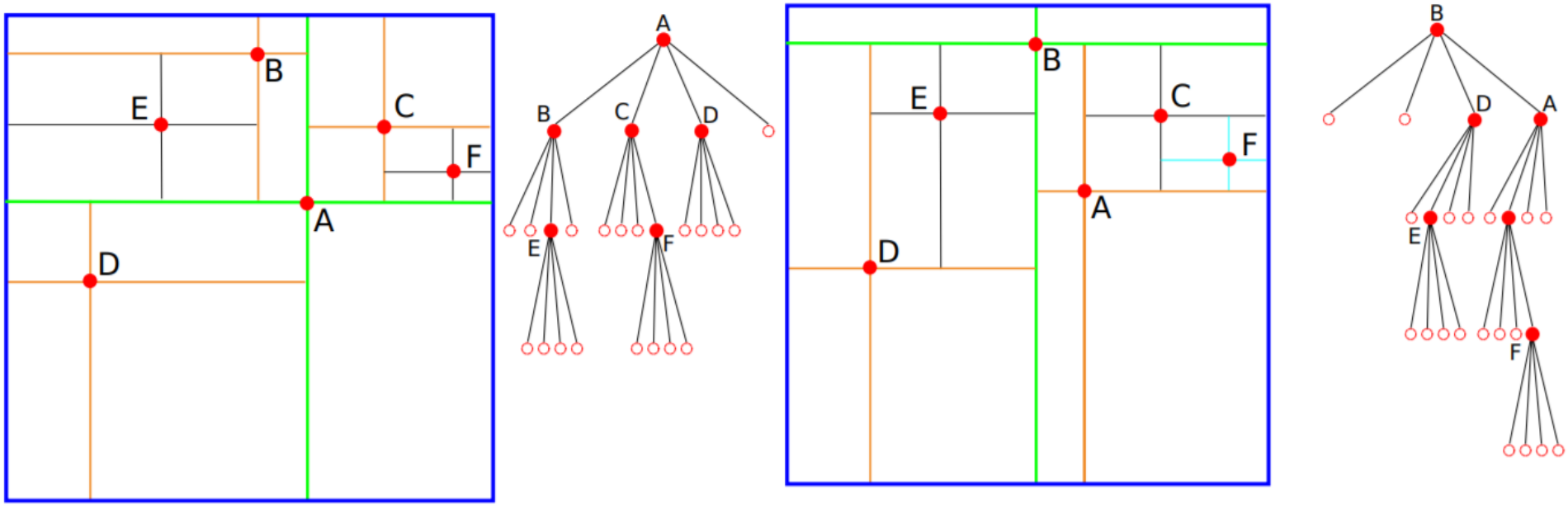
- Zerlege den Bereich in Zellen
- Jede Zelle hat Maximalkapazität (hier 2)
- Wird diese überschritten, teilt sich die Zelle in 4 kleineren Zellen





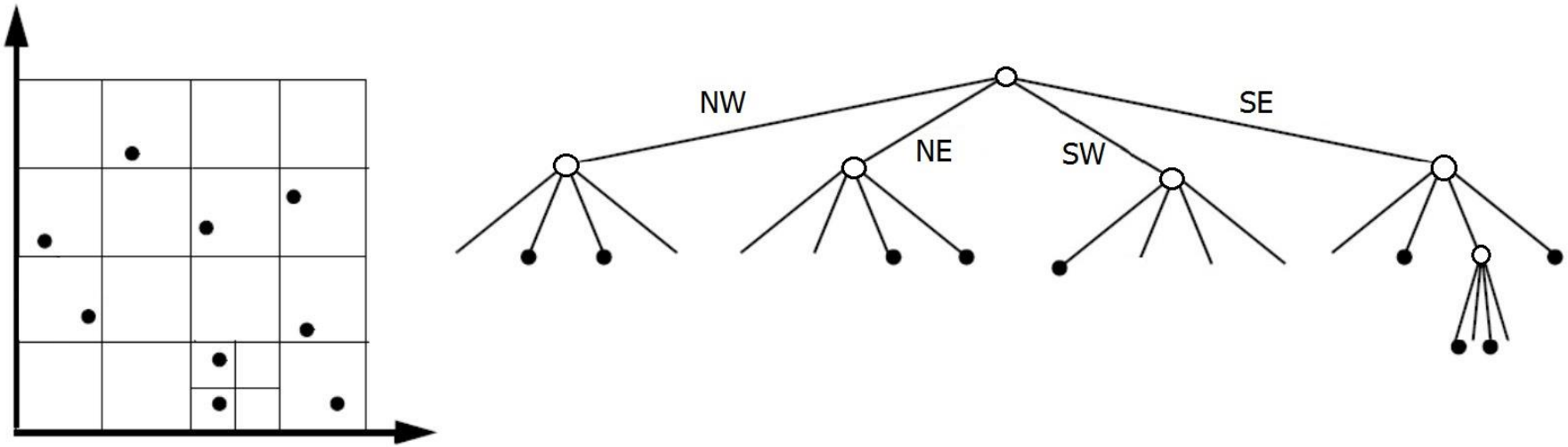
PQ vs. PRQ

PQ - Wie klassischer KD-Tree abhängig von der Reihenfolge der Einfügeoperationen der Features (auch initial):



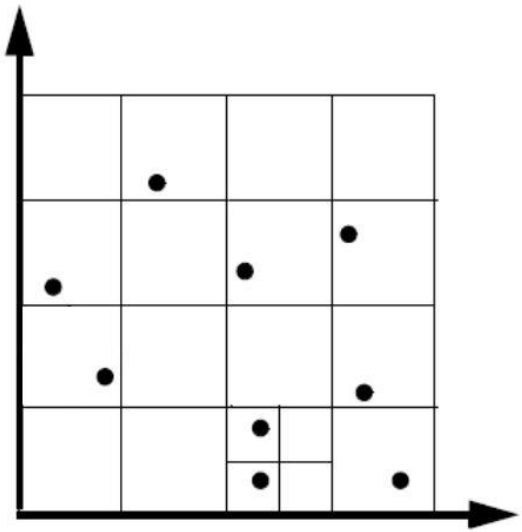
PQ vs. PRQ

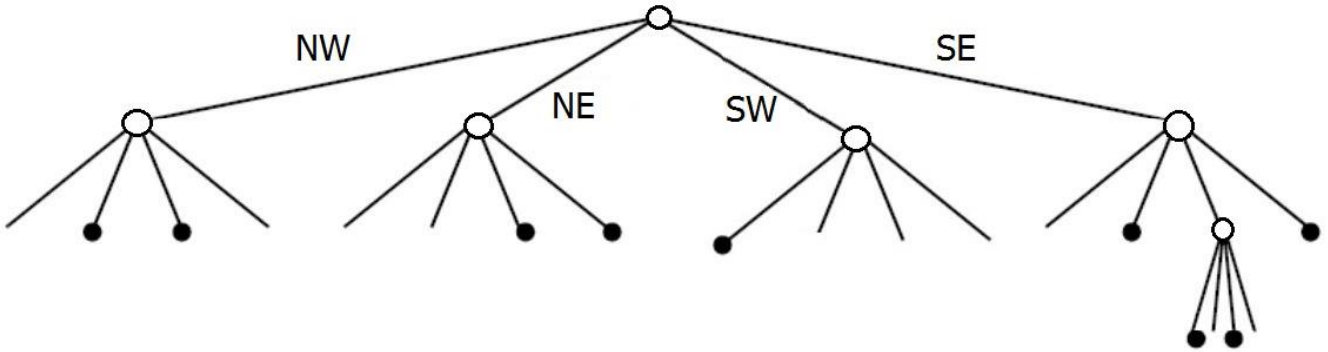
PRQ aber resistent - sowohl initial als auch danach:



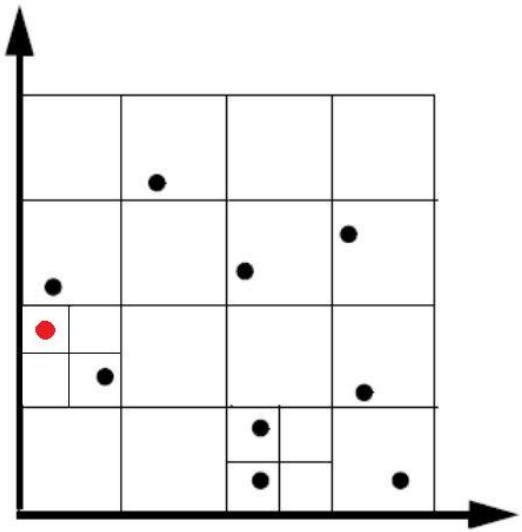
→ Fokussierung liegt im Folgenden auf PRQ

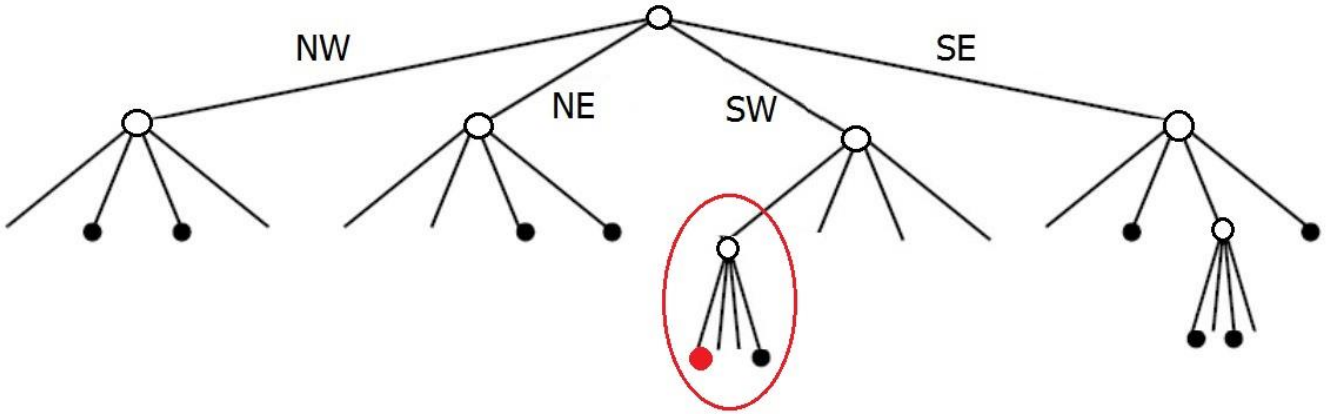
PRQ – Einfügen – 1.Fall



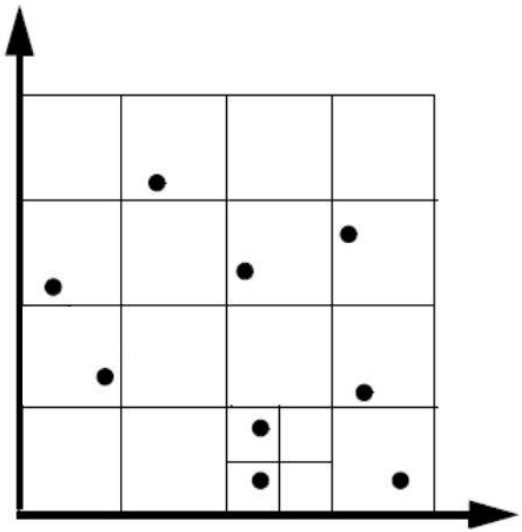


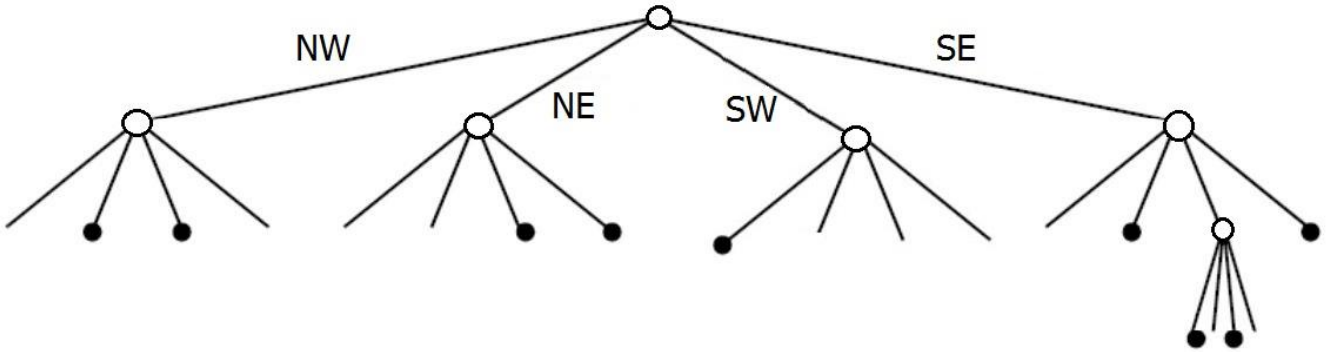
PRQ – Einfügen – 1.Fall



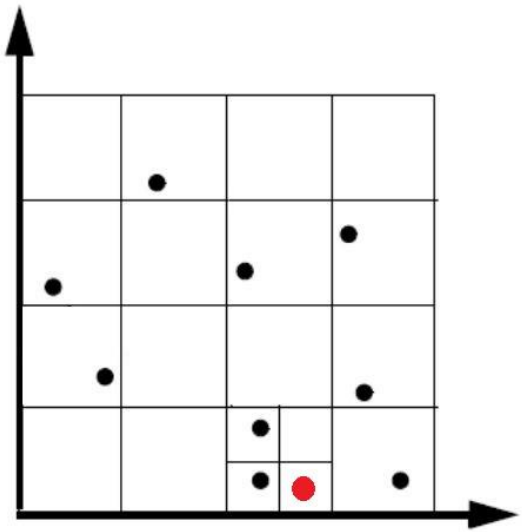


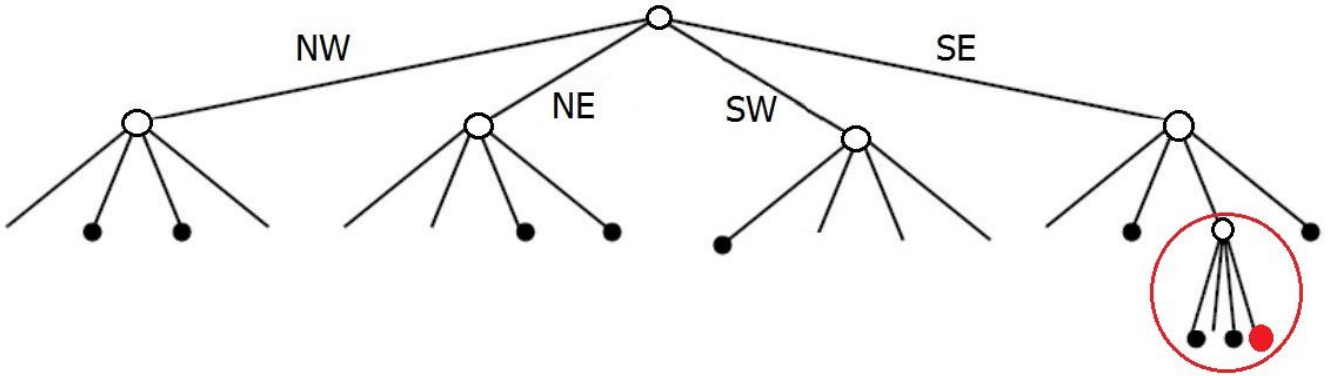
PRQ – Einfügen – 2.Fall



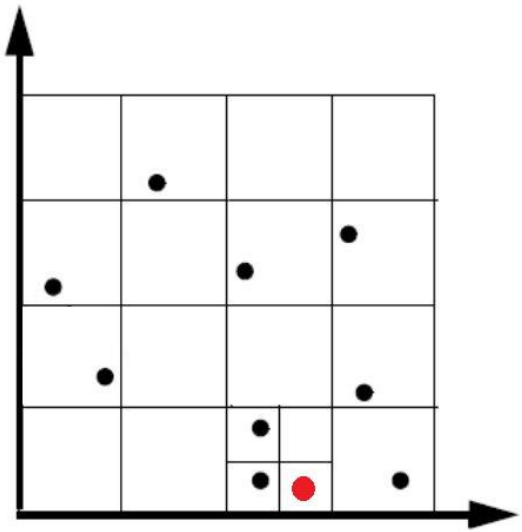


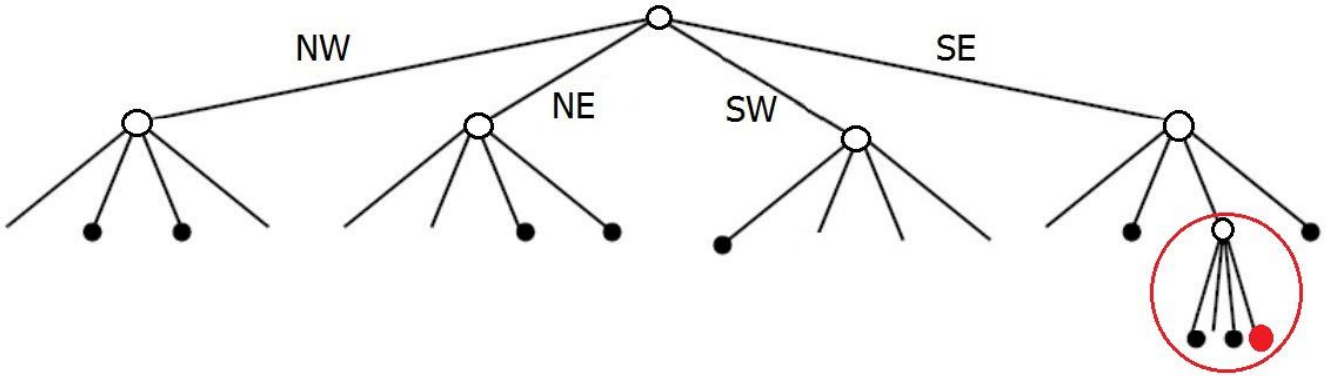
PRQ – Einfügen – 2.Fall



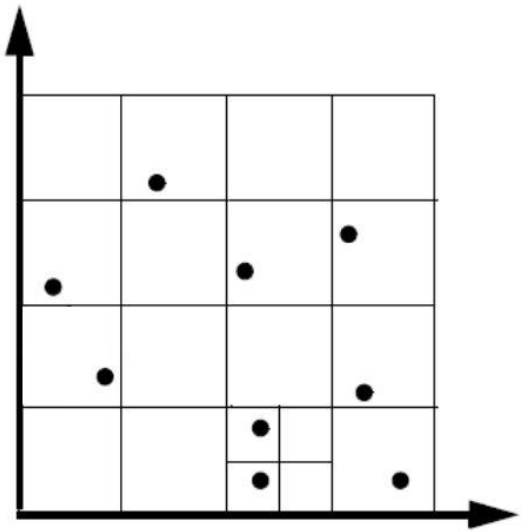


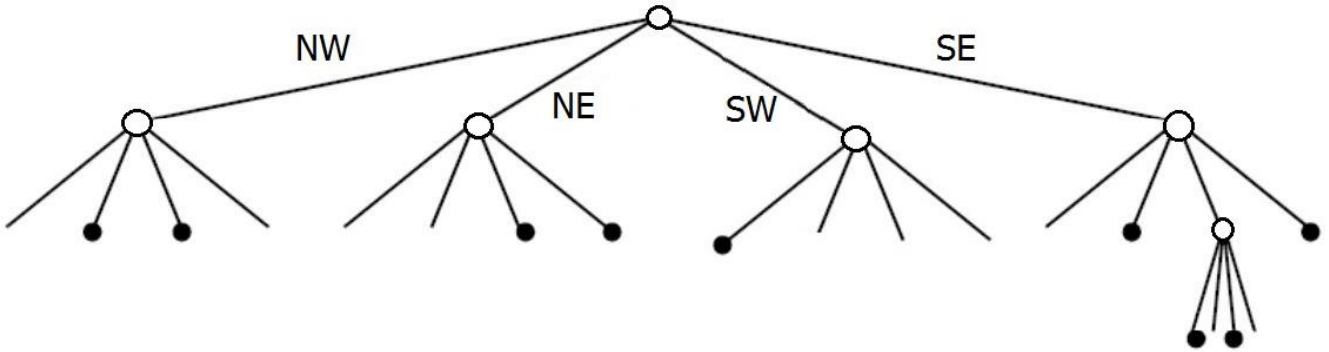
PRQ – Löschen – 1.Fall



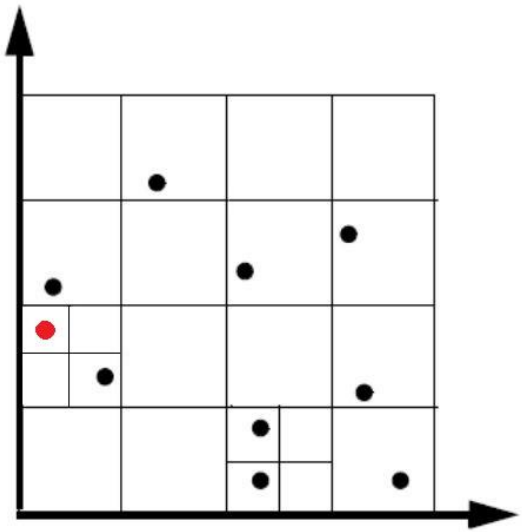


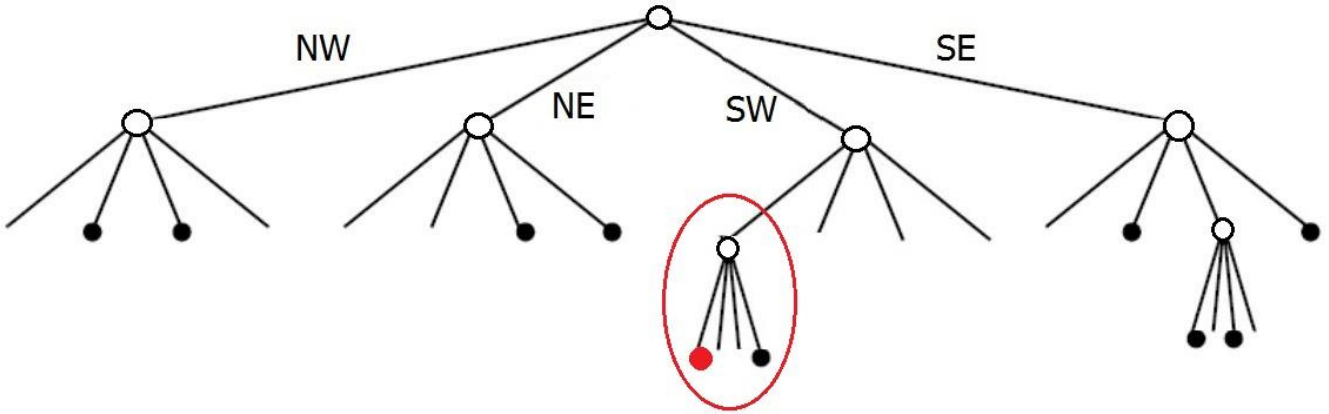
PRQ – Löschen – 1.Fall



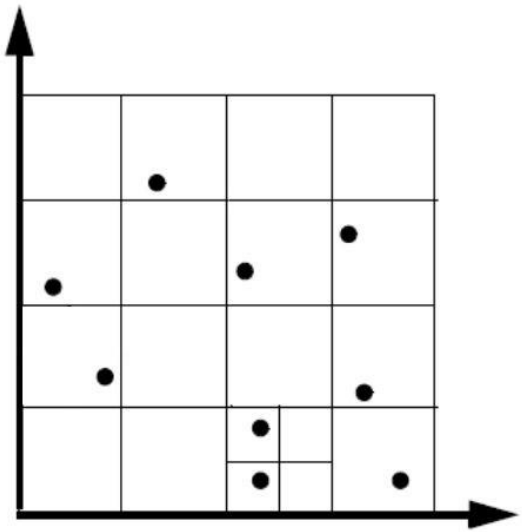


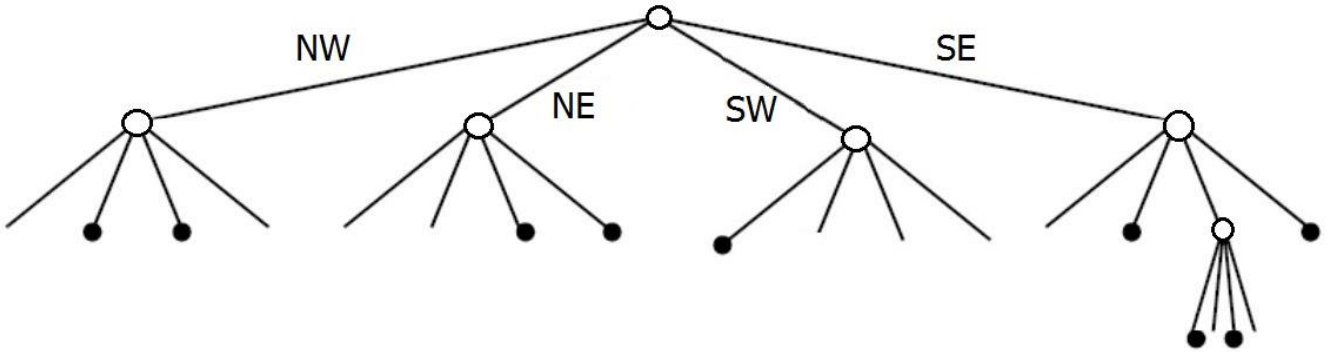
PRQ – Löschen – 2.Fall





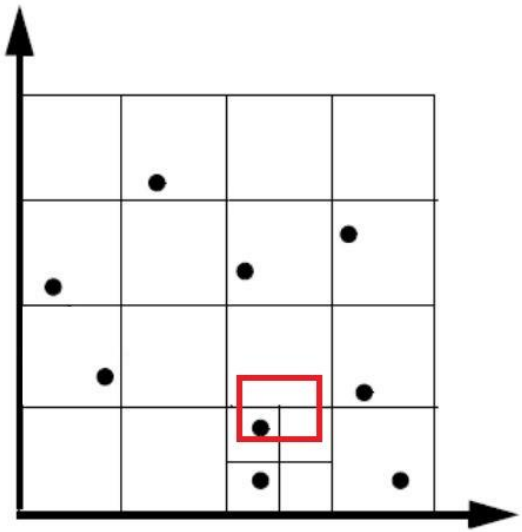
PRQ – Löschen – 2.Fall

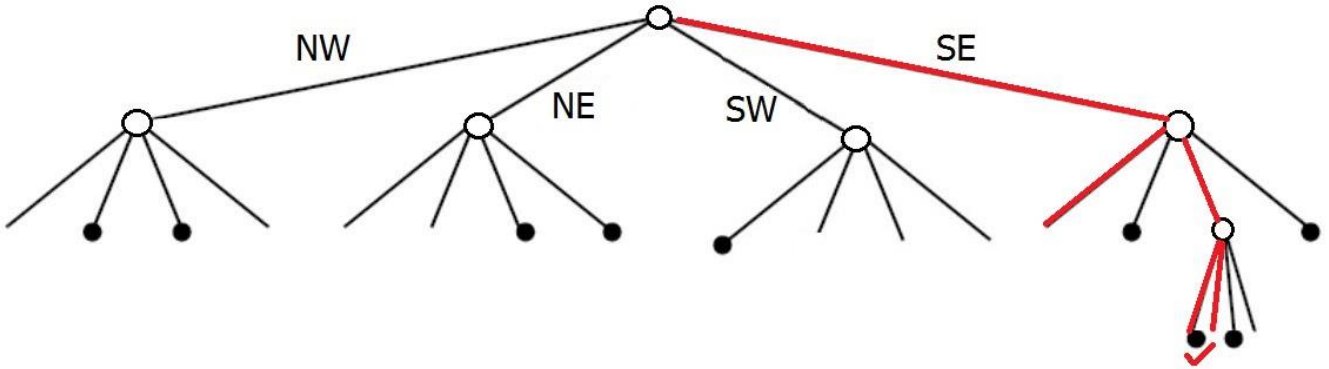




PRQ – Bereichssuche

Ähnlich der Suche in KD-Tree:



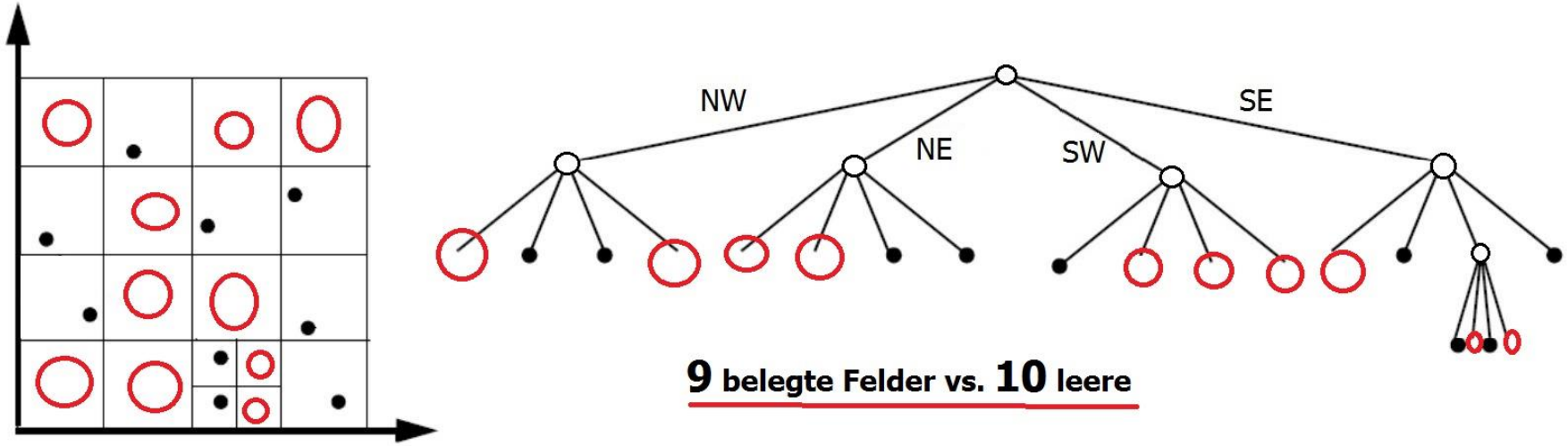


Quad-Tree – Bewertung

Die Datensätze befinden sich am Ende des Baums, wo die Suche endet.

Folgen der Partitionierung in der geometrischen Mitte in 4 Bereichen →

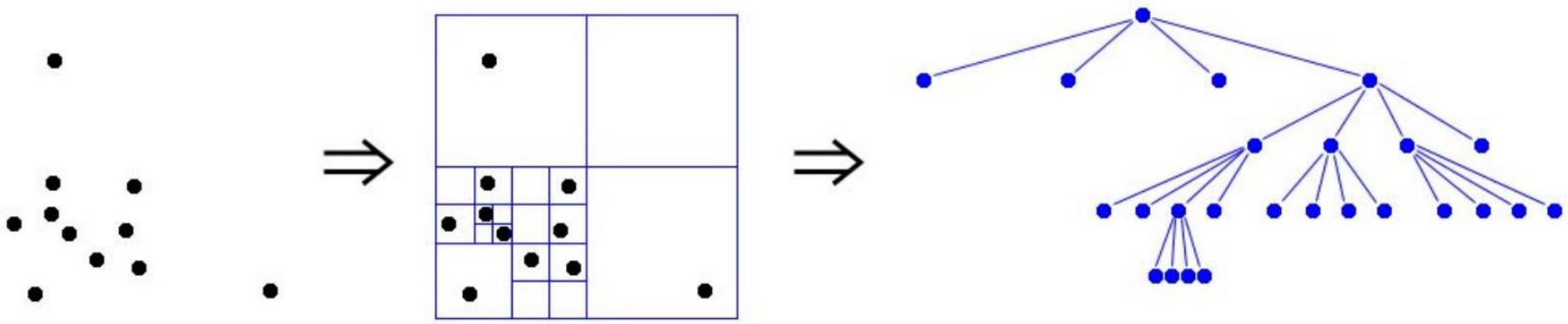
1.Folge: Das Problem der leeren Referenzen



Quad-Tree – Bewertung

2.Folge:

Durch ungleichmäßige Verteilung der Punkte im Raum - oder durch entsprechend ungünstigen Lösch- und Einfüge-Operationen - kann der Baum schnell degenerieren.



Quad-Tree – Bewertung

Quad-Tree kann mehr als zwei Dimensionen unterstützen. Folgen nach dem Hinzukommen von weiteren Dimensionen:

Die Anzahl der Quadranten verdoppelt sich bei jeder weiteren Dimension → Mehr leere Referenzen

Name „Quad-Tree" ist bei mehr als 2 Dimensionen nicht korrekt. Bei 3 Dimensionen = „Oc-Tree".

Grid-File

Grid-File - Einführung

Bisher schlechte Speicherauslastung und ungleiche Zugriffs-zeiten.
→ Mängel durch Einführung von Grid-File abschwächen.

Zugriffszeiten hängen von der Geschwindigkeit und Anzahl der Plattenzugriffe ab. Durch folgende zwei Prinzipien werden diese verkürzt:

Grid-File - Einführung

1. Eine voll spezifizierte Anfrage muss einen Datensatz nach mindestens zwei Plattenzugriffe zurückgegeben haben.
2. Datensätze, die sich in den Werten ihrer Attribute ähneln, sollen auch im physischen Speicher so nahe wie möglich beieinander liegen.

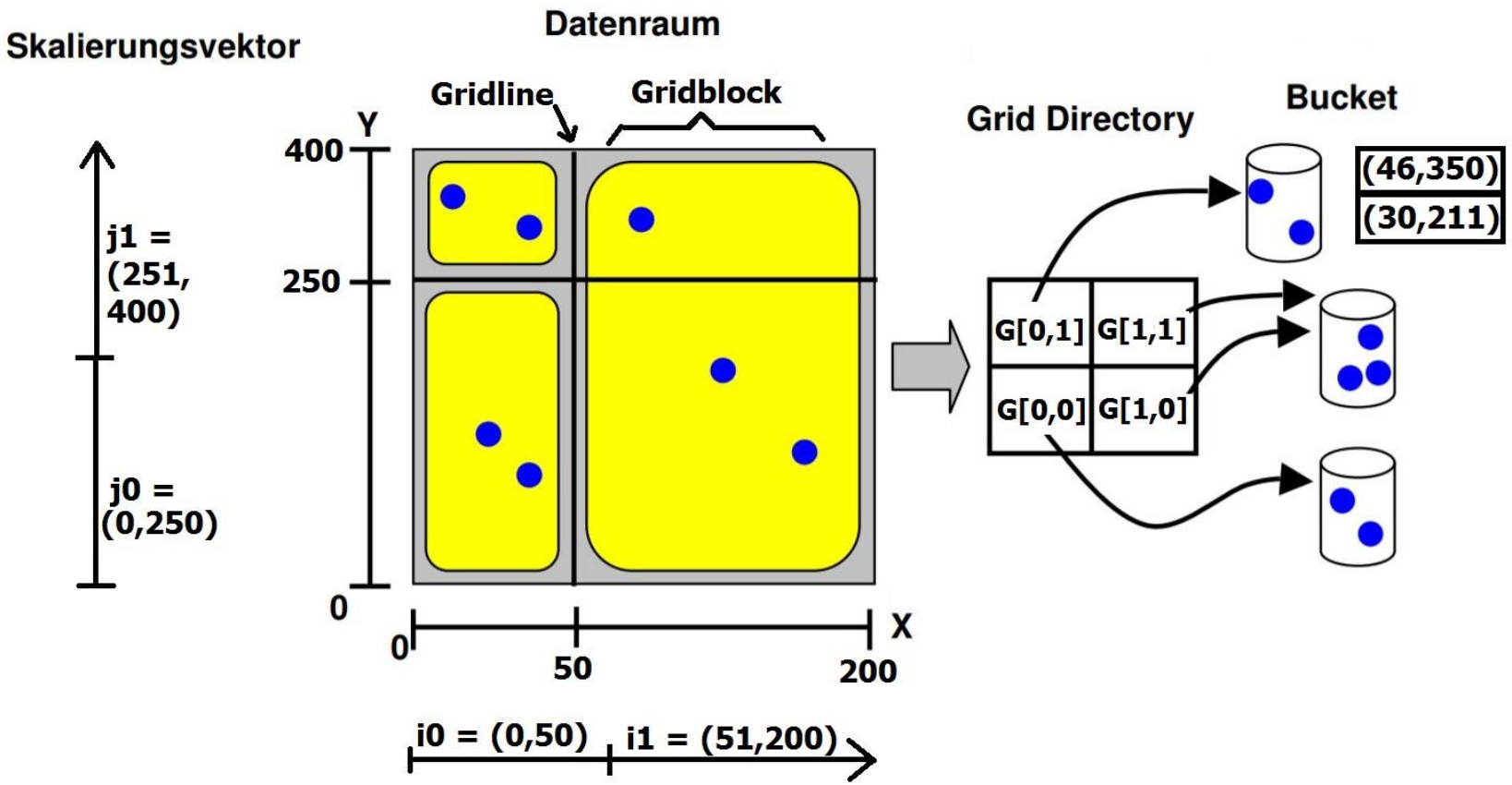
→ Grid-File setzt diese Prinzipien um!

Grid-File – Konstruktion (K-Dimensionen)

Komponenten

- K * Skalen bzw. K * 1-dimensionale **Skalierungsvektoren** zum Einstieg ins Grid-Directory
  - K=2 → Zwei 1-dimensionale Skalierungsvektoren, welche z.B. die Unterteilung der X- und Y-Achse enthalten
- Ein K-dimensionales **Grid-Directory** zum Finden der Bucket-Nr.
  - K=2 → Ein 2-dimensionales Grid-Directory, welches Verweise auf die Bucket enthält
- **Bucket** für Datensätze
  - K=2 → Mehrere Bucket, welche jeweils ein maximale Zahl von Datensätze aufnehmen können.

Grid-File - Konstruktion



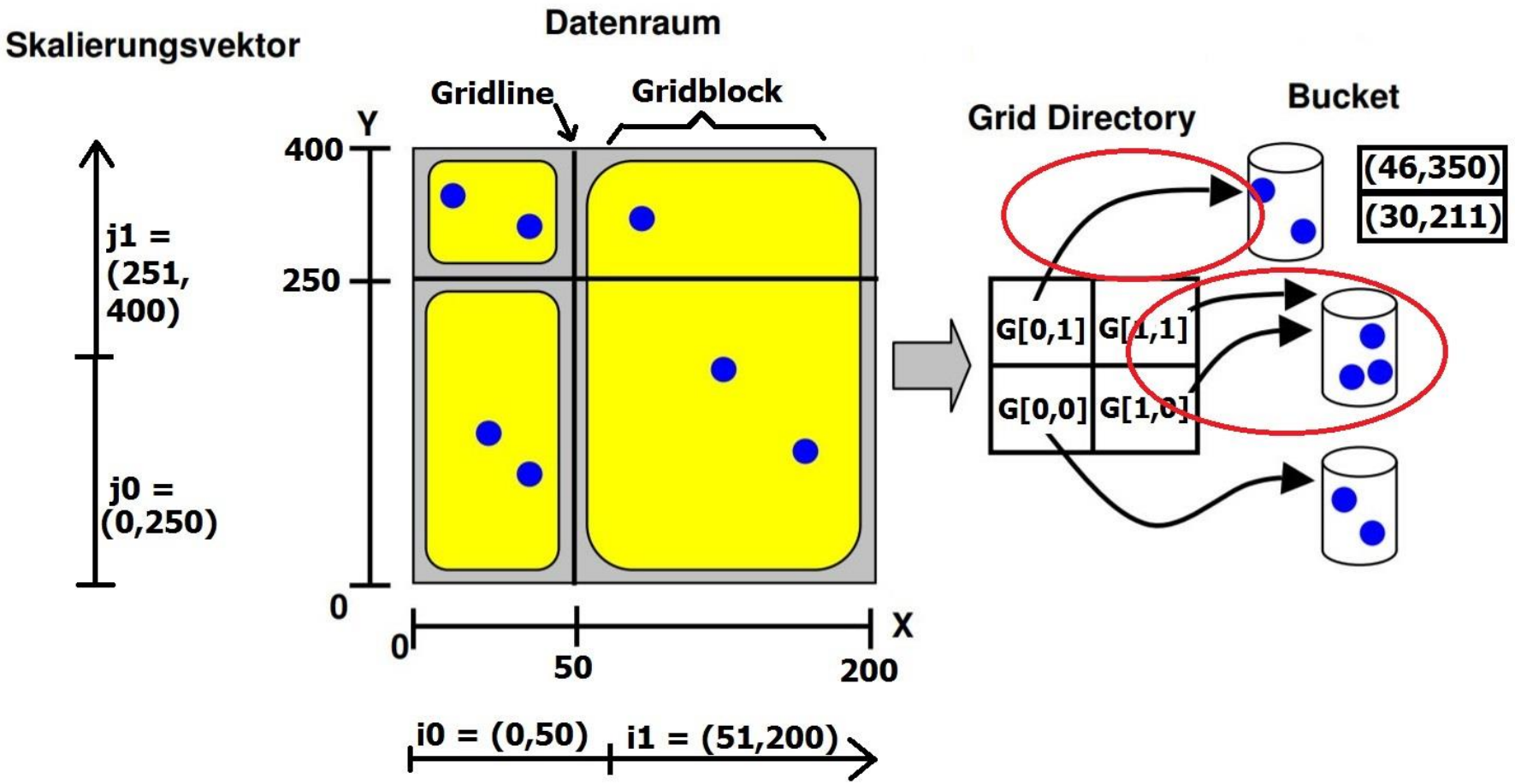
Grid-File - Konstruktion

Achtung:

In **einem** Bucket können die Datensätze von **mehreren** Gridblocks gespeichert sein.

Die Datensätze **eines** Gridblocks sind aber **in dem selben** Bucket gespeichert.

Grid-File - Konstruktion

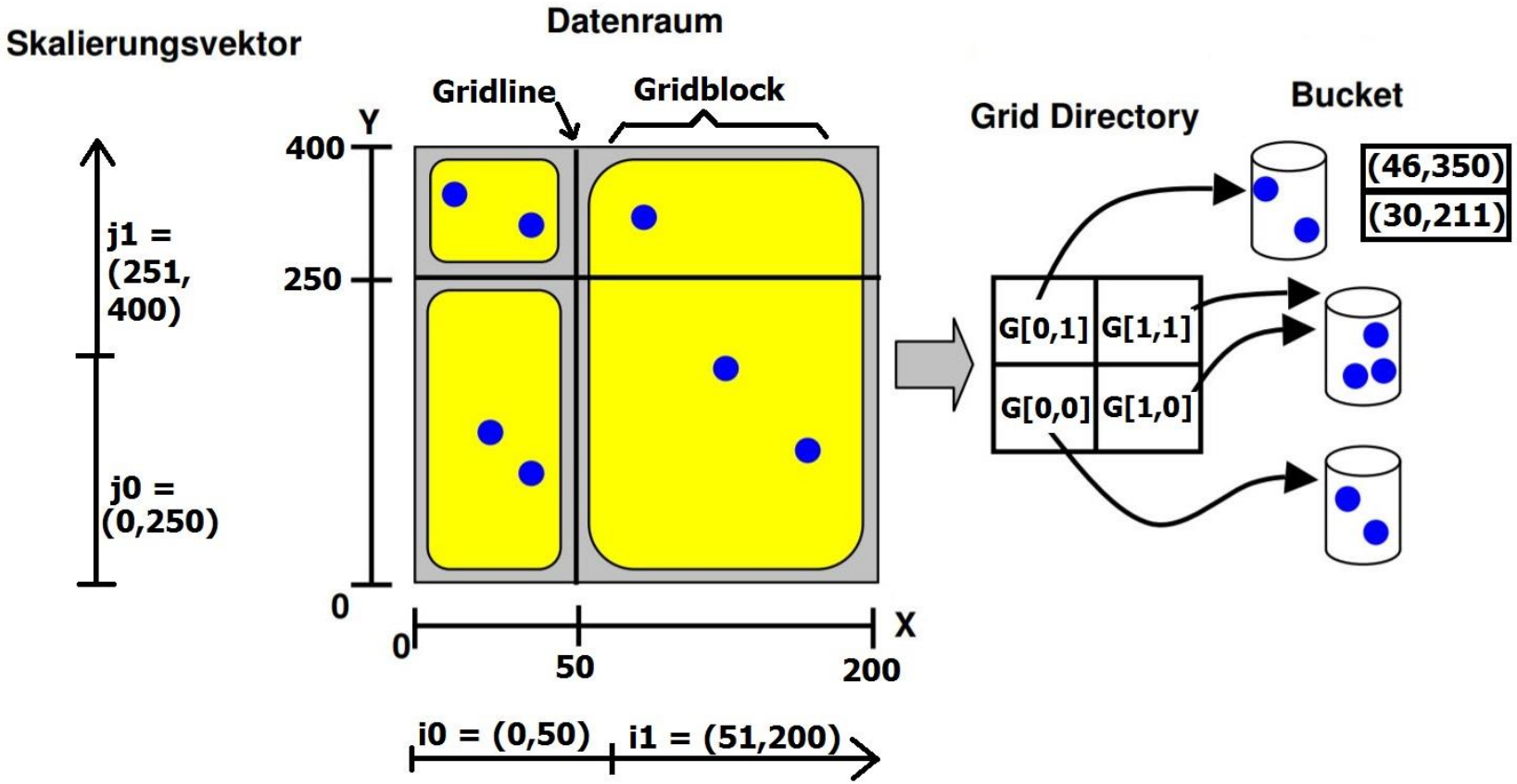


Grid-File – Punktsuche

Für einen Nachschlag mit x=46, y=350:

- Suche in Skalierungsvektor X nach dem Index des Feldes mit x=46 enthält → **0**
- Analog für y=350 in Skalierungsvektor Y → **1**
- Lade nun den Teil des Grid-Directory G[xi,yi] – also G[0,1] in den Hauptspeicher
- Lade Bucket mit der Adresse aus G[0,1]
- Suche explizit nach Elementen für die gilt: x=46, y=350

Grid-File – Punktsuche

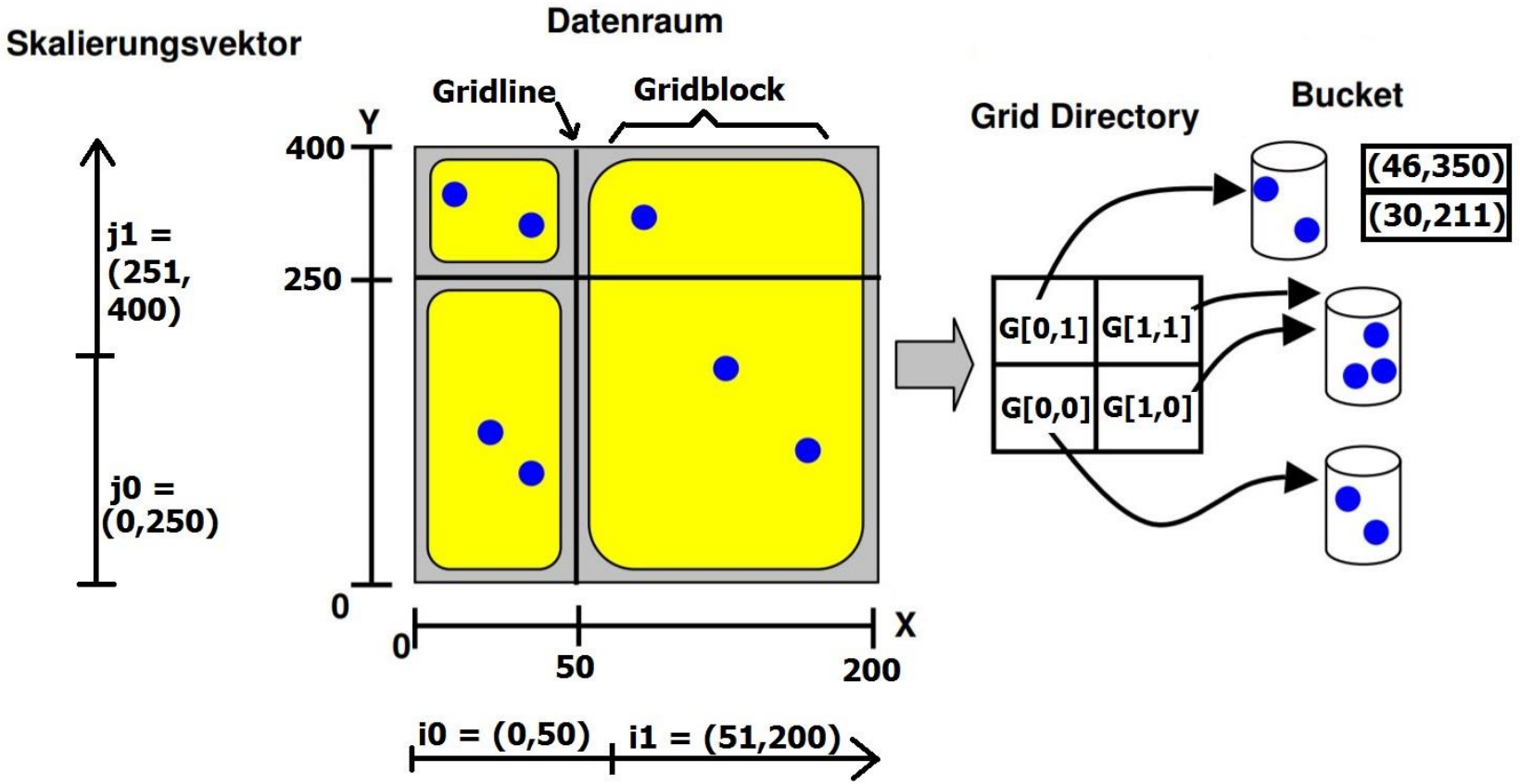


Grid-File – Bereichssuche

Für einen Nachschlag mit [40,200 – 80,300]:

- Äquivalent x=40 – 80, y=200 – 300
- Suche in Skalierungsvektor X nach den Indexen der Felder mit x1=40 bzw. x2=200 → **0** & **1**
- Analog für y1=200 bzw. y2=300 in Skalierungsvektor Y → **1** & **1**
- Lade nun den Teil des Grid-Directory G[x1i,y1i] – G[x2i,y2i] also G[0,1] – G[1,1] in den Hauptspeicher
- Lade Bucket mit den Adressen aus G[0,1] – G[1,1]
- Gib alle Elementen aus den Bucket mit: x=40 – 80 **&** y=200 – 300

Grid-File – Bereichssuche



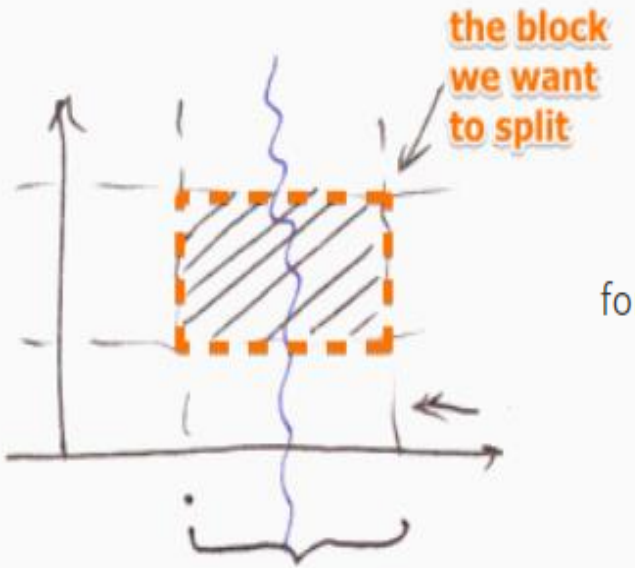
Grid-File – Einfügen

Einfügen: Bucket wird gesucht und geladen. Daten werden eingefügt, das Bucket wieder auf die Festplatte geschrieben.

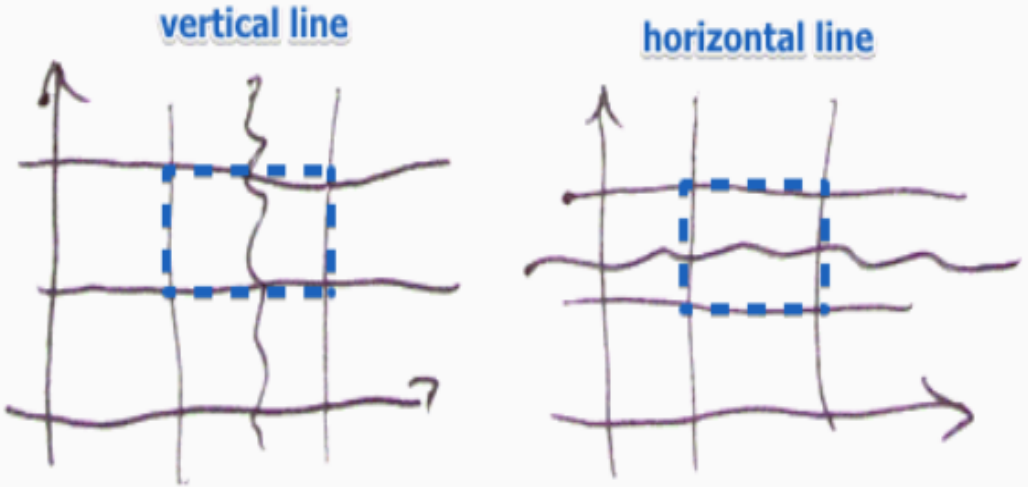
Teilung: Wird Bucketbelegung zu groß, so wird Inhalt auf zwei Bucket verteilt. Dabei wird dessen Region (Gridblock) und damit der gesamte Datenraum geteilt.

Grid-File – Einfügen

so adding a grid line will split all the buckets along this line



for n dimensions we may choose which dimension to split



Grid-File – Löschen

Löschen: Analog zum Einfügen, nur wird der Datensatz entfernt.

Verschmelzung: Wenn Bucketbelegung unter einem festgelegten Schwellwert, bietet es sich an: dieses Bucket zu löschen und seine Region (Gridblock) mit der Region eines anderen zu verschmelzen.

Grid-File - Bewertung

Vorteile:

Dynamische Struktur (siehe Bucket-Größe)

Durch Teilung & Verschmelzung → Bessere Speicherauslastung

Kürzere Zugriffszeiten

Unterstützt neben Punkt- & Bereichsanfragen, Nächster-Nachbar-Anfragen (Zunächst gucken wir Bucket an, dann die Nachbar-Bucket – entnommen aus dem Grid-Directory)

Grid-File - Bewertung

Nachteile:

Wie Quad-Tree viele leere Bucket, wenn Daten schlecht verteilt sind.

→ Wir brauchen einen guten Algorithmus zum Teilen & Verschmelzen.

Genannte Nachteile sind in höheren Dimensionen gravierender.

R-Tree

R-Tree - Einführung

Im R-Raum können Geometrien mit einer beliebigen räumlichen Ausdehnung eingefügt werden.
Diese können etwa Punkte, aber auch beliebige Regionen sein.

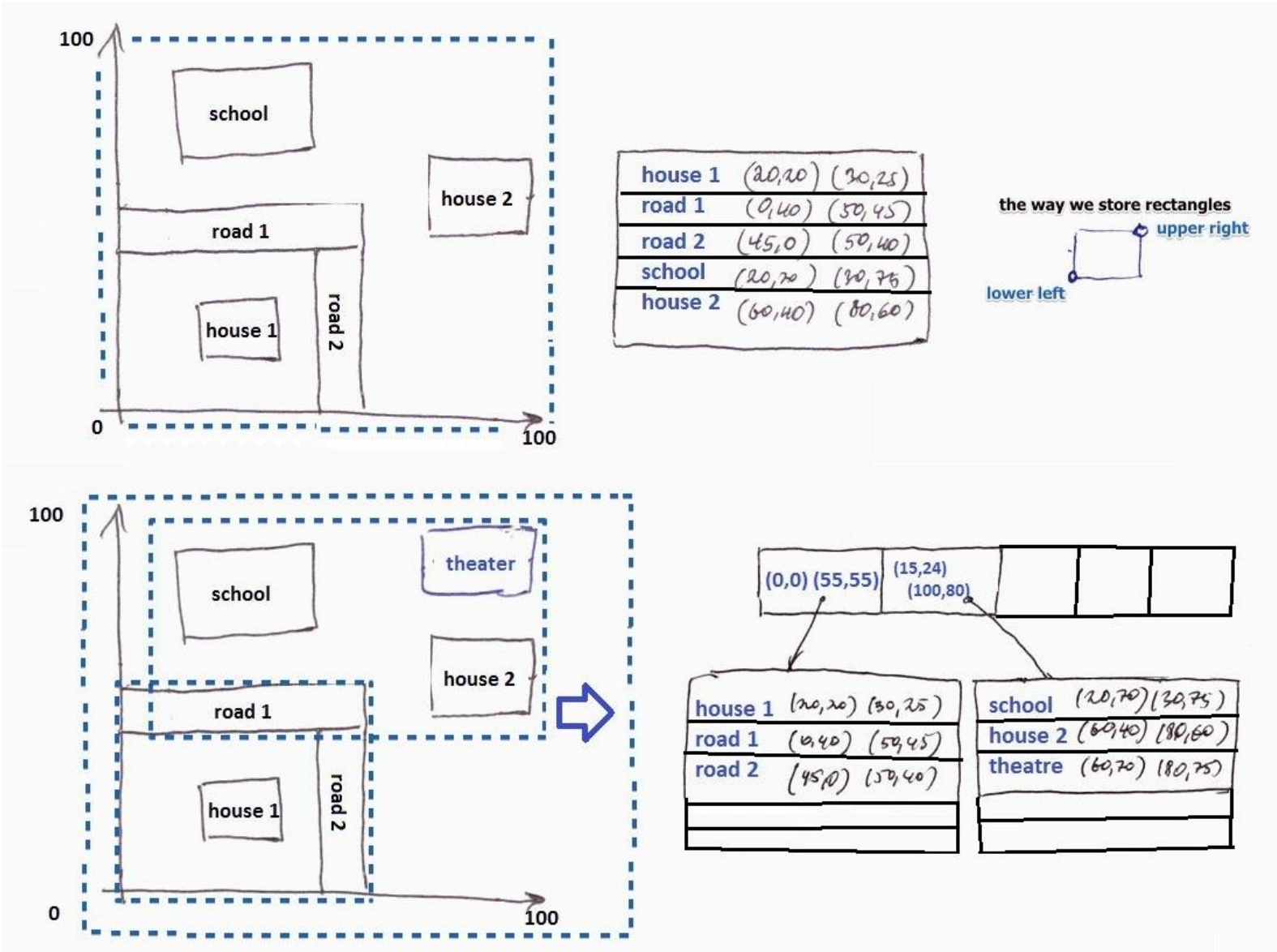
Nutzung z.B. bei der Navigation (Range- bzw. Window-Query auf 2D-Karte).

R-Tree - Aufbau

Der R-Baum ist ein Mehrwegbaum →
Ein innerer Knoten kann aus mehreren Einträgen bestehen und mehrere Kinder haben.
Anzahl der Einträge wird durch vordefinierten Min & Max eingeschränkt.

Als Einheit im R-Raum bzw. R-Baum werden MBR (Minimum Bounding Rectangle) verwendet.

R-Tree - Aufbau



R-Tree - Aufbau

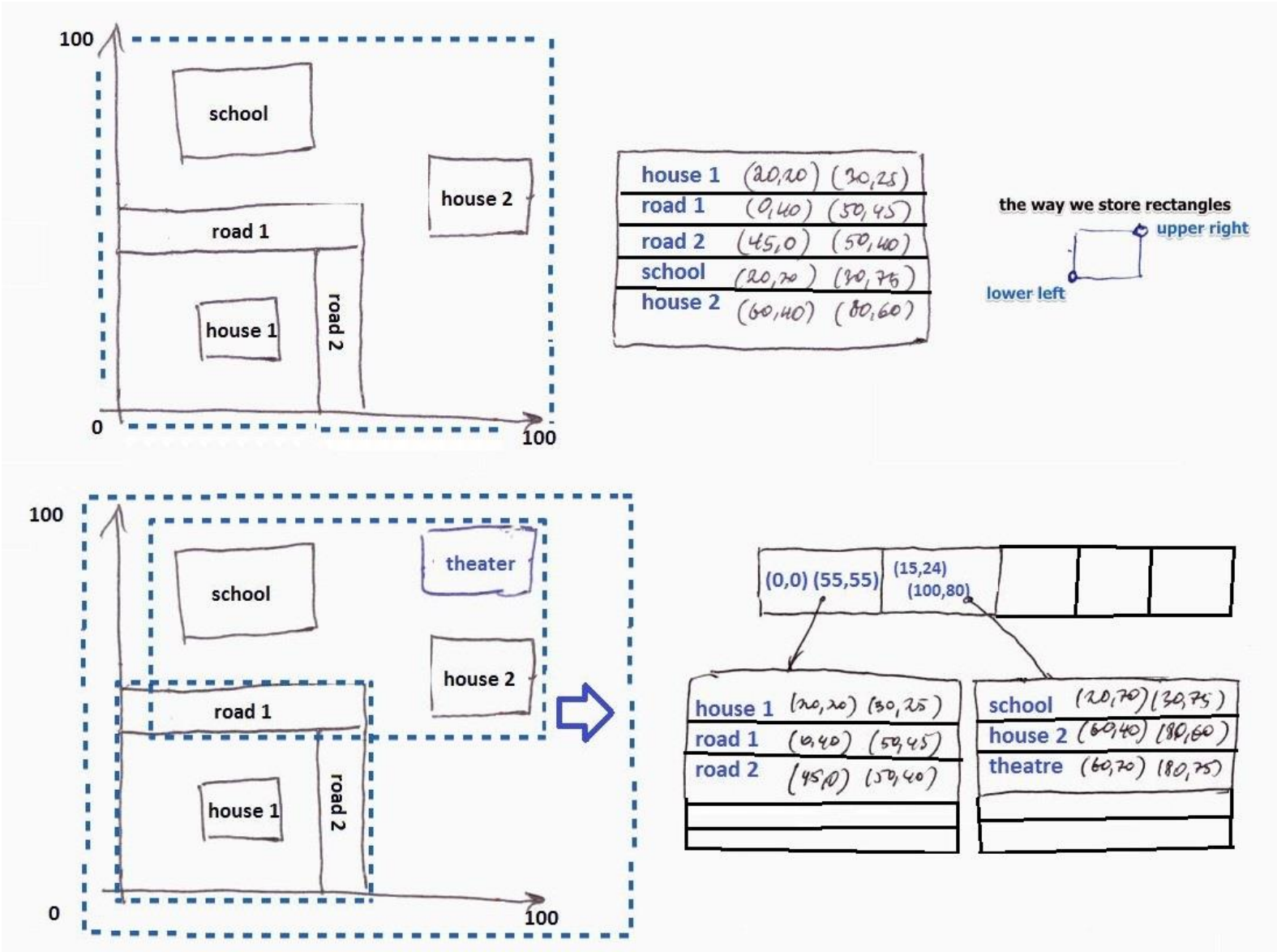
Zunächst nur ein MBR vorhanden.

Es kommen so viele Datensätze rein, so viele Einträge ein Knoten maximal haben darf.

Wenn mehr Einträge als Max, dann Überlauf beim Knoten → Teilung in zwei kleineren MBR.

Einen MBR zusätzlich als neue Wurzel erstellen, der wiederum diesen beiden MBR enthält.

R-Tree - Aufbau

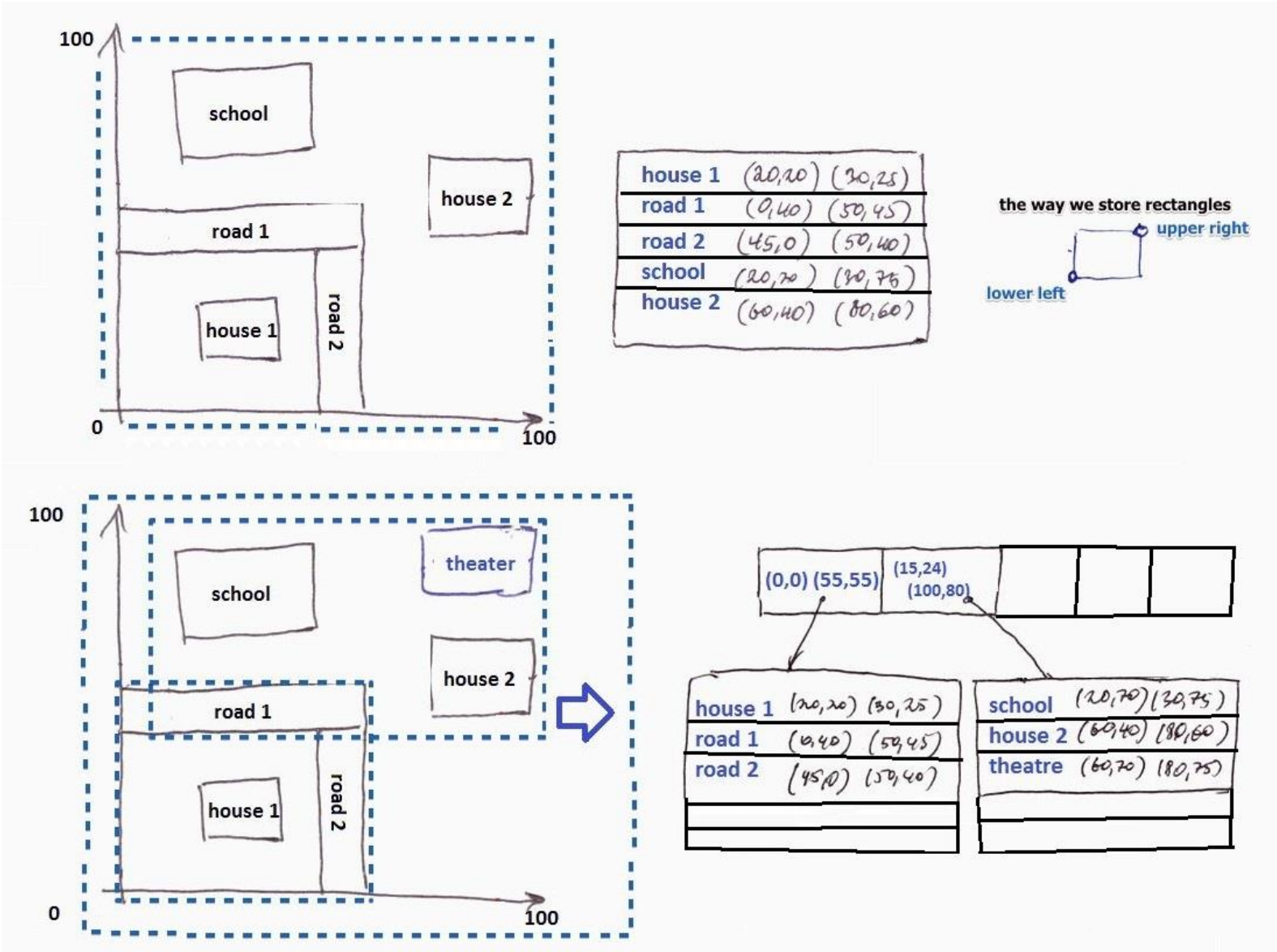


R-Tree - Aufbau

Wenn sich diese MBR durch noch mehr Datensätzen teilen und vermehren, dann Überlauf bei der Wurzel →
Wurzel und seinen MBR in zwei Teile zerlegen und neue Wurzel erstellen:

1) Baum wächst von unten nach oben, ist damit immer balanciert.
2) In den Blättern befinden sich nur MBR und Verweis auf Datensätze. Z.B: „House".
3) In den inneren Knoten bzw. Wurzel befinden sich nur MBR und Verweis auf Kinderknoten.

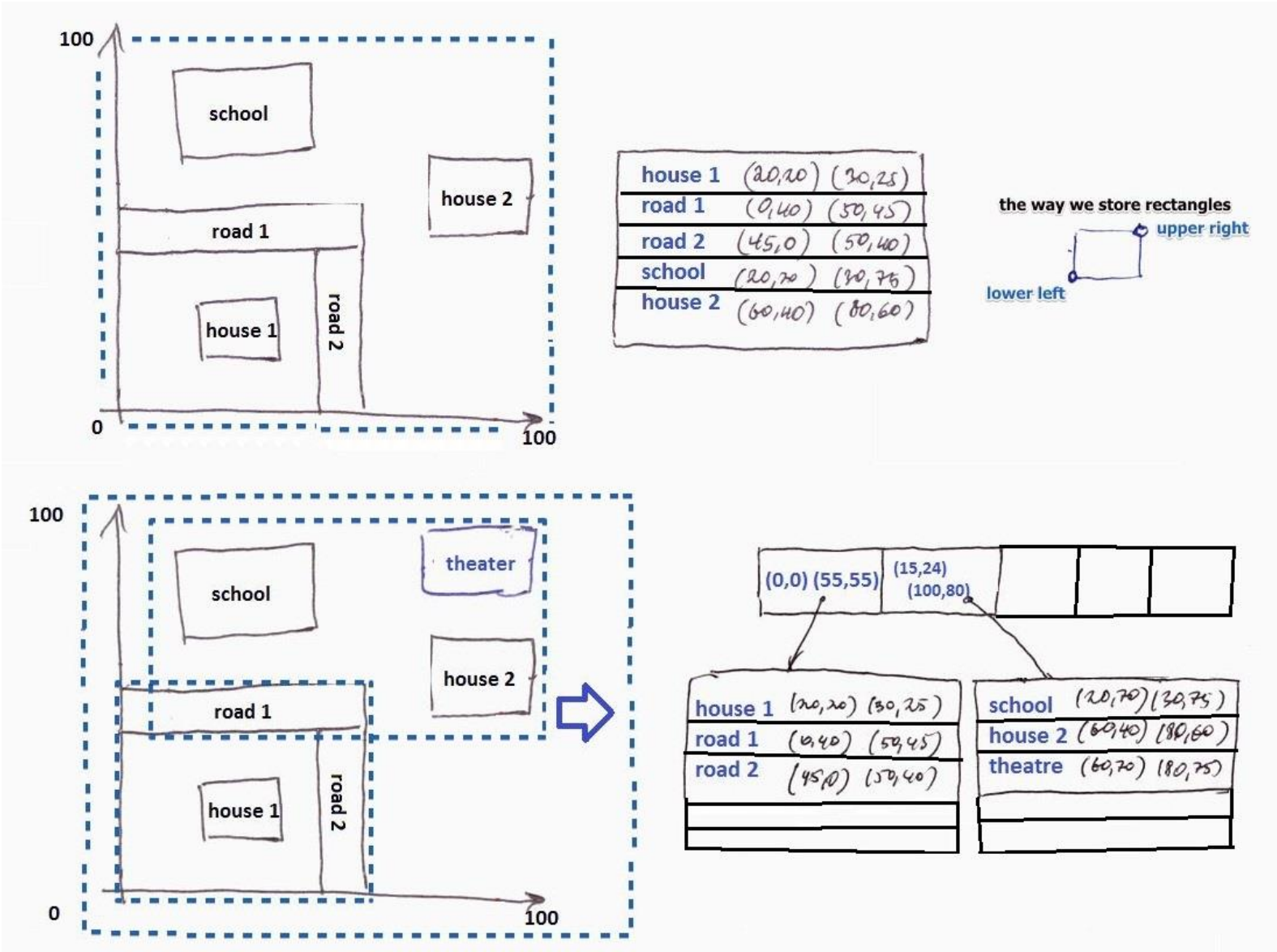
R-Tree - Aufbau



R-Tree - Aufbau

Die MBR können sich überlappen und dadurch die Suche verlangsamen. (Beispiel …)

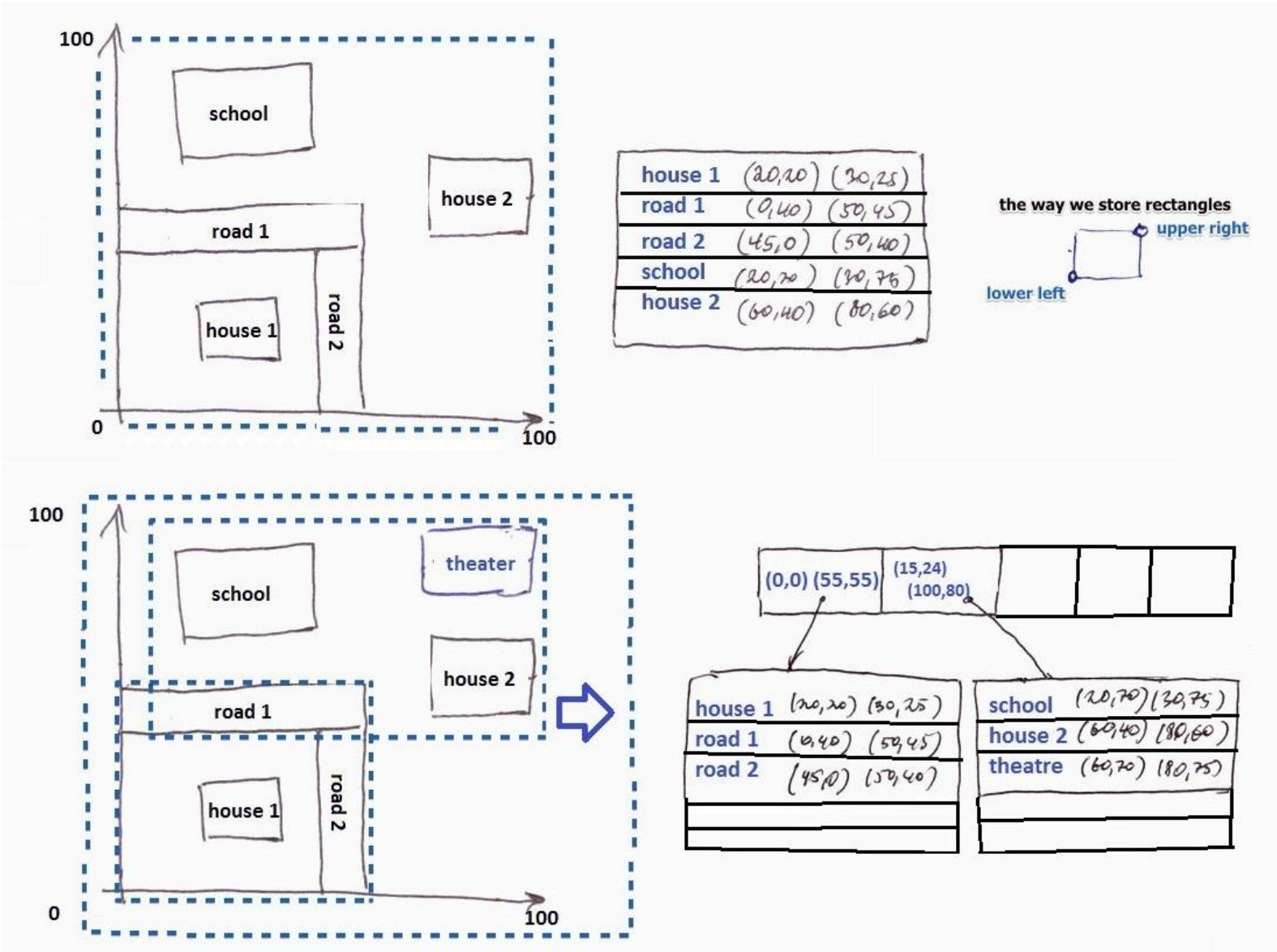
R-Tree - Aufbau



R-Tree - Aufbau

Damit es zu weniger Überlappungen kommt, müssen die **MBR** bei ihrer Entstehung - also **beim Überlauf bzw. bei der Teilung - so klein wie möglich** gehalten werden. → Denn beim Entstehen neuer MBR (falls wenig andere vorhanden) geringere Kollisionsgefahr.

R-Tree - Aufbau



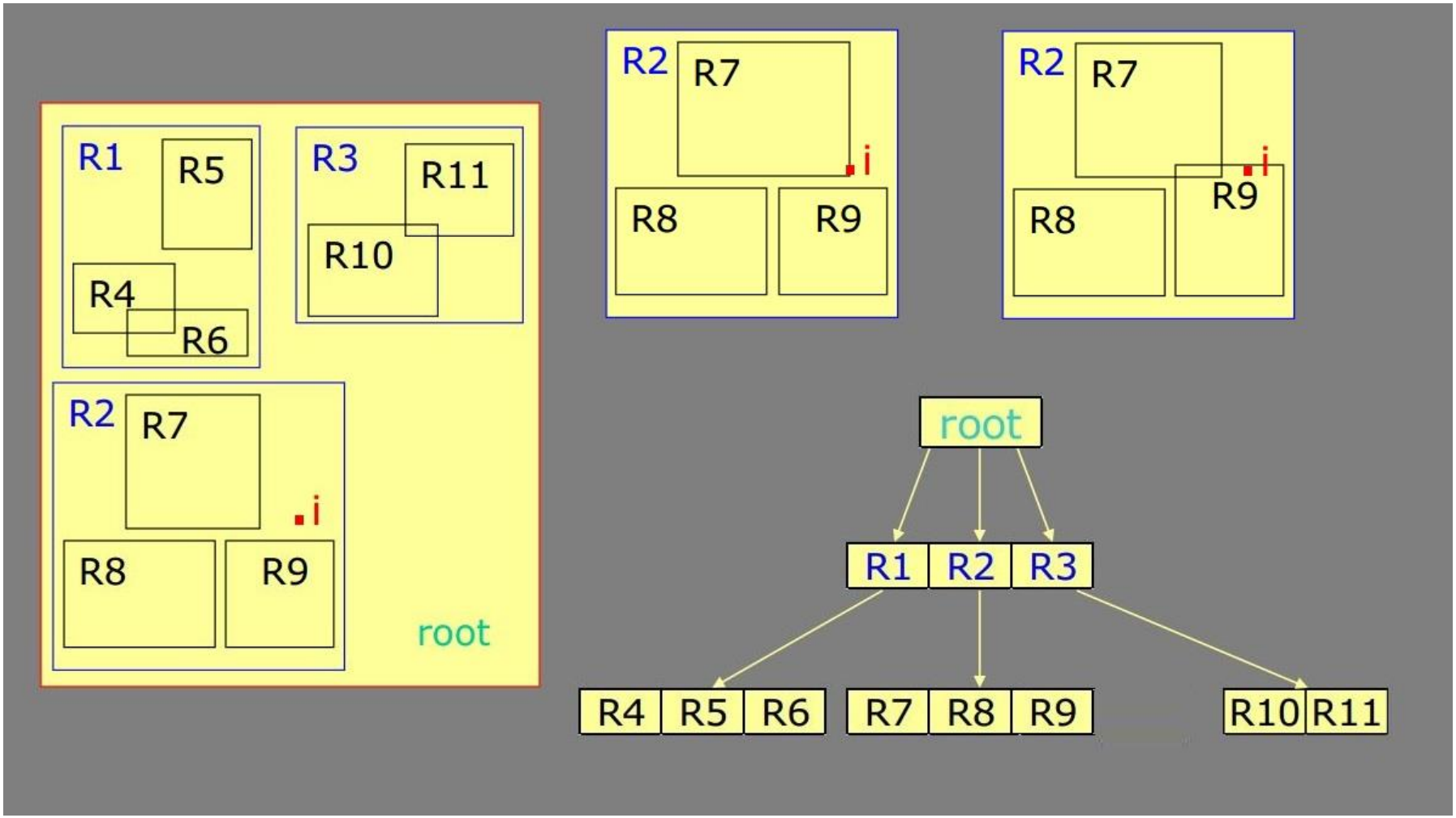
R-Tree - Einfügen

Teilung kann nur beim Einfügen stattfinden →

Einfügen:

- Fange bei der Wurzel als Knoten an und fahre bis zum Blatt fort:
  Finde auf der Ebene (auf der du dich befindest) Knoten mit MBR, in dem/denen der Datensatz passt.
  → Wenn gefunden: Fahre bei diesem(n) Knoten fort.
  → Wenn nicht gefunden: Fahre beim Knoten fort, bei dem der MBR am wenigsten Vergrößert werden muss, damit der Datensatz reinpasst.
- Sobald du bei einem Blatt angekommen bist: Füge den Datensatz hier ein. Falls notwendig: vergrößere den MBR dieses Blattes. Passe die Größe der Elternknoten rekursiv an.

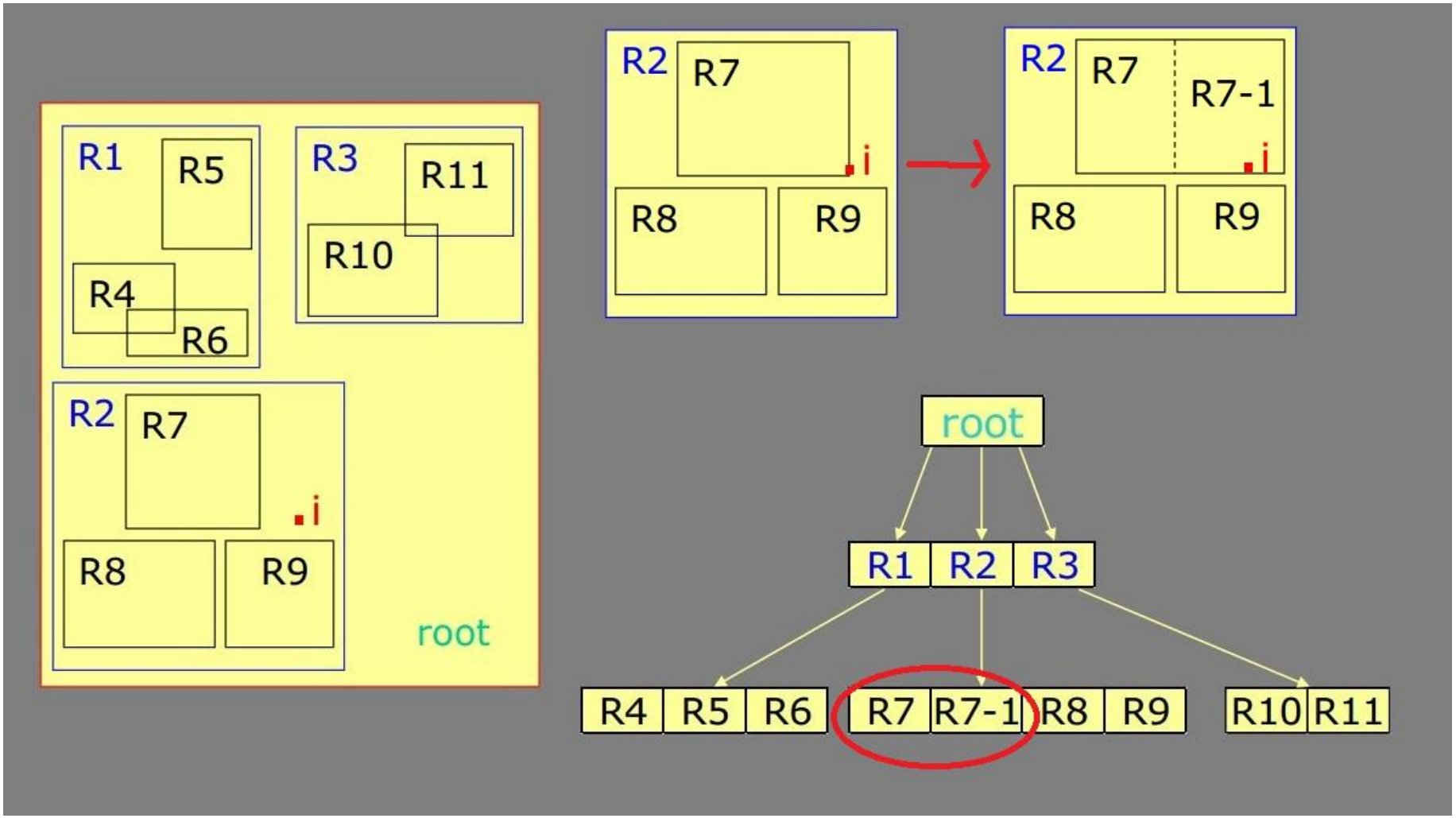
R-Tree - Einfügen



R-Tree - Einfügen

- Falls nach dem Einfügen im Blatt, Überlauf im Blatt, dann teile Blatt und damit seinen MBR in zwei Teile.
  Splitte - falls Überlauf auch bei den Elternknoten - rekursiv bis Überlauf abgefangen.

R-Tree - Einfügen



R-Tree – Einfügen (Split-Operation)

Die Entscheidung, welche Feature-Objekte welchem der beiden neuen MBR zugeordnet werden sollen, ist nicht trivial.

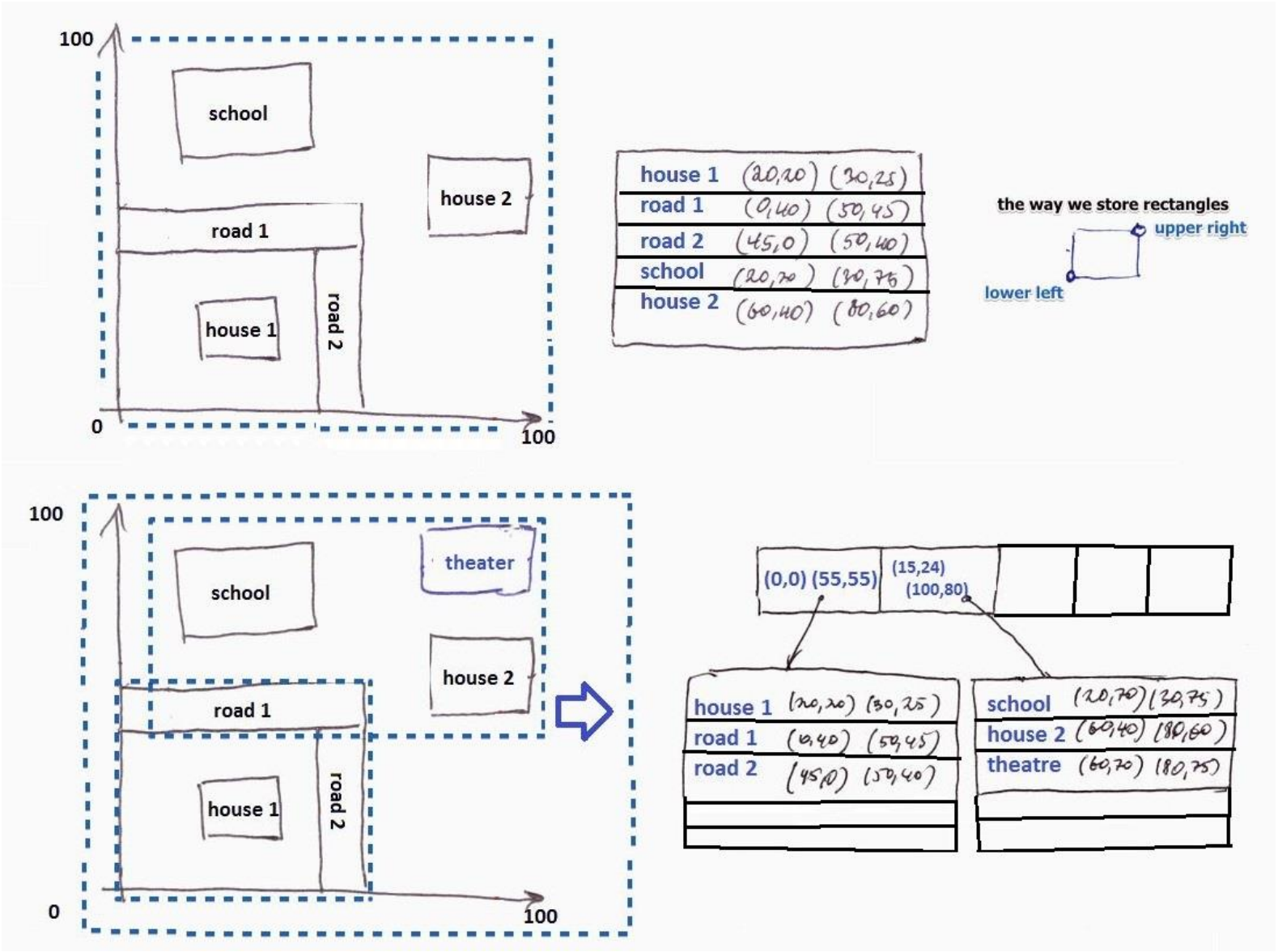
Ziel: Minimierung der Volumensumme der beiden neuen MBR, damit später kleinere Überlappungen.

R-Tree – Einfügen (Split-Operation)

Algorithmus zum Erstellen zweier MBR aus einem, der bezogen auf Anzahl der Feature-Objekte einen linearen Berechnungsaufwand hat:

- Es werden zwei Startobjekte mittels eines linearen Algorithmus gesucht. (Dieser findet zwei extrem außen liegende Feature-Objekte oder auch MBR als geeignete Startobjekte für die zwei neuen MBR.)
- Danach werden die restlichen Feature-Objekte in beliebiger Reihenfolge dem MBR zugewiesen, welcher das jeweils geringere Erweiterungsvolumen benötigt.

R-Tree – Einfügen (Split-Operation)



R-Tree - Löschen

Beim Entfernen von Feature-Objekten muss umgekehrt vorgegangen werden: Ein Unterlauf wird durch ein Zusammenfassen von zwei MBR überwunden.

R-Tree vs. R+-Tree

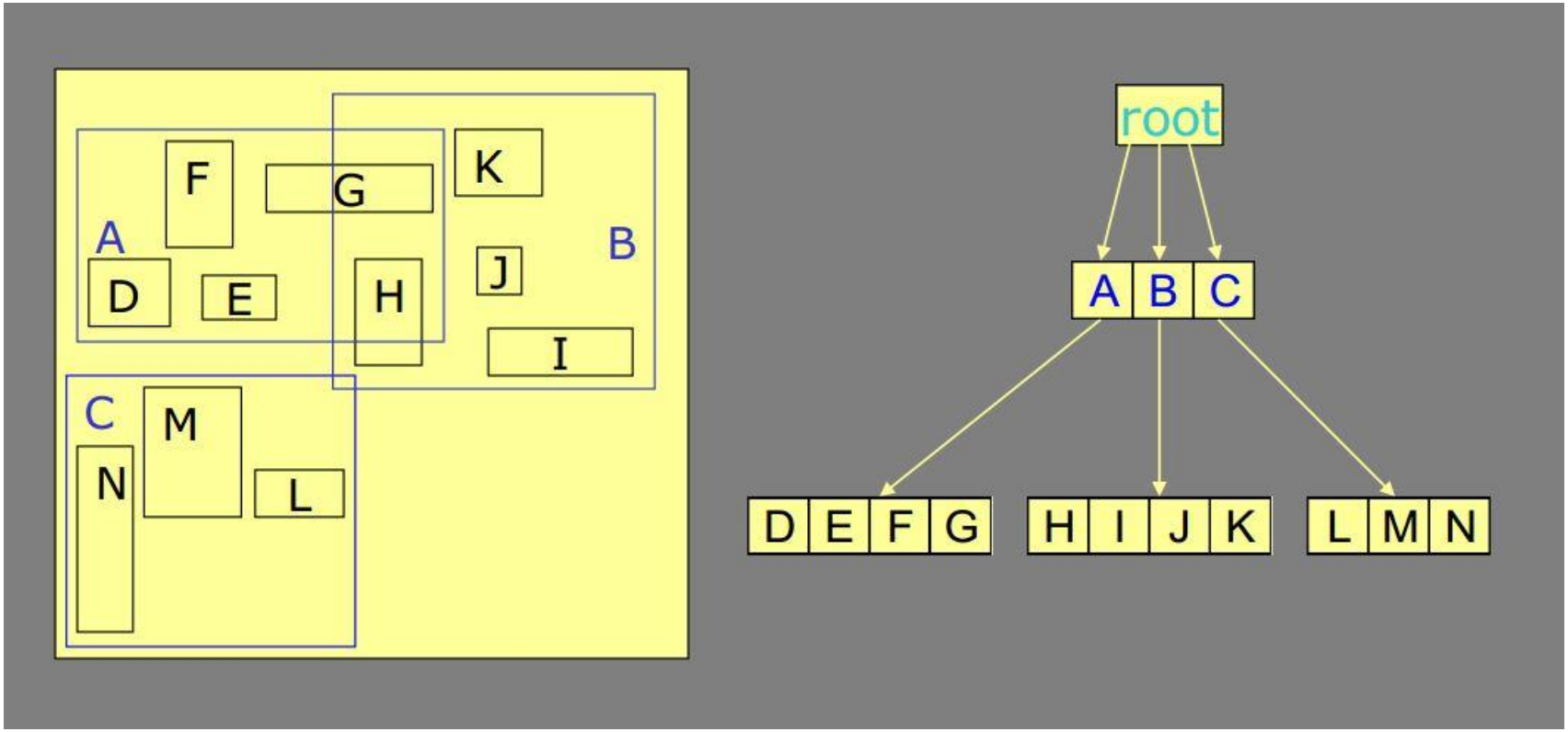
Der R-Baum ist eine effiziente Indexstruktur, so lange die Anzahl der Dimensionen nicht zu hoch wird.
Ab etwa 10 Dimensionen wird die Suche im R-Baum ineffizient.
→ Eine schnellere Variante R+-Tree

R-Tree vs. R+-Tree

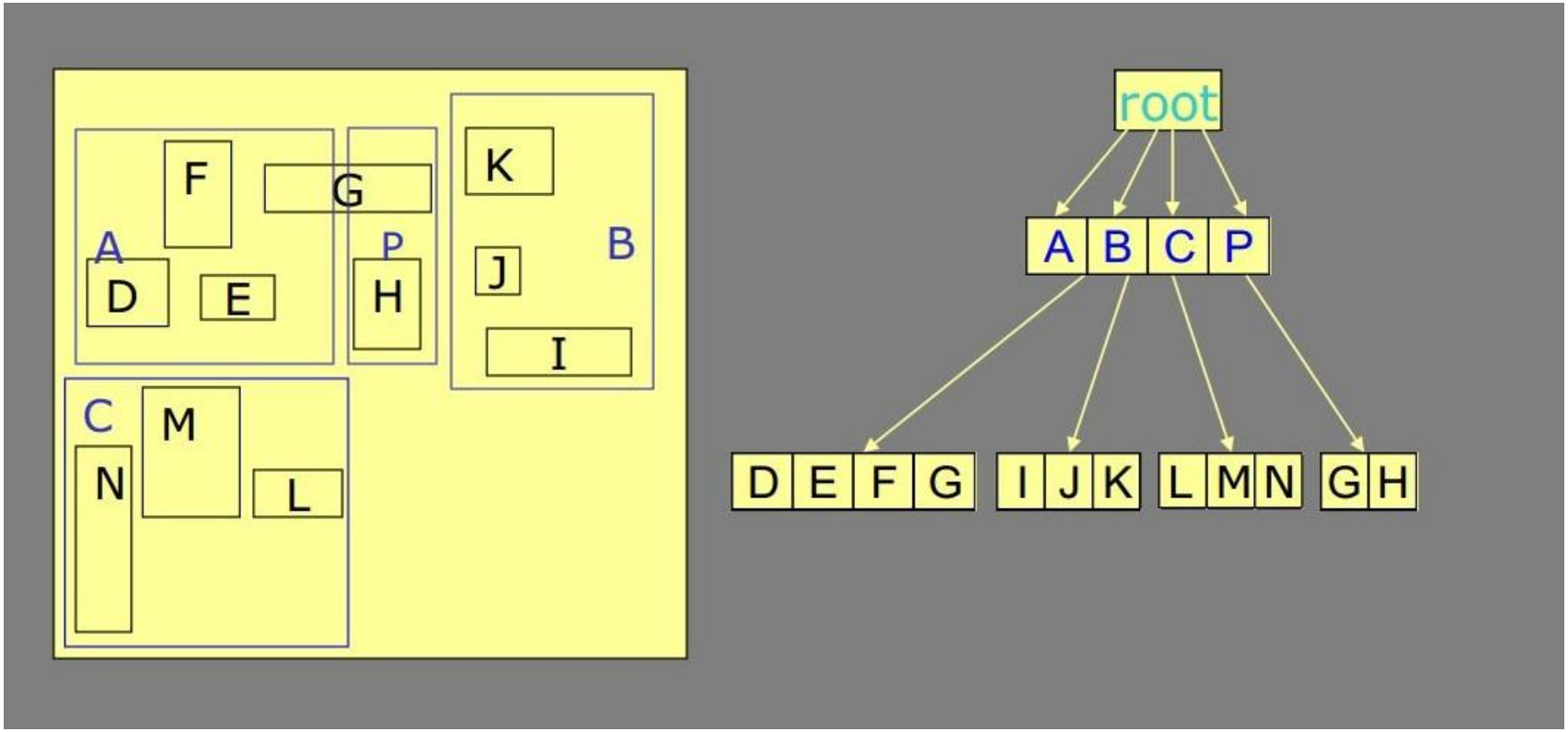
Grundlage für die Entwicklung des R+-Baums ist die Beobachtung, dass der R-Baum wegen der starken Überlappung benachbarter MBR im hochdimensionalen Raum ineffizient wird.

→ Grundidee des R+-Baums liegt darin, eine Überlappung im Baum benachbarter MBR zu verbieten.

R-Tree vs. R+-Tree



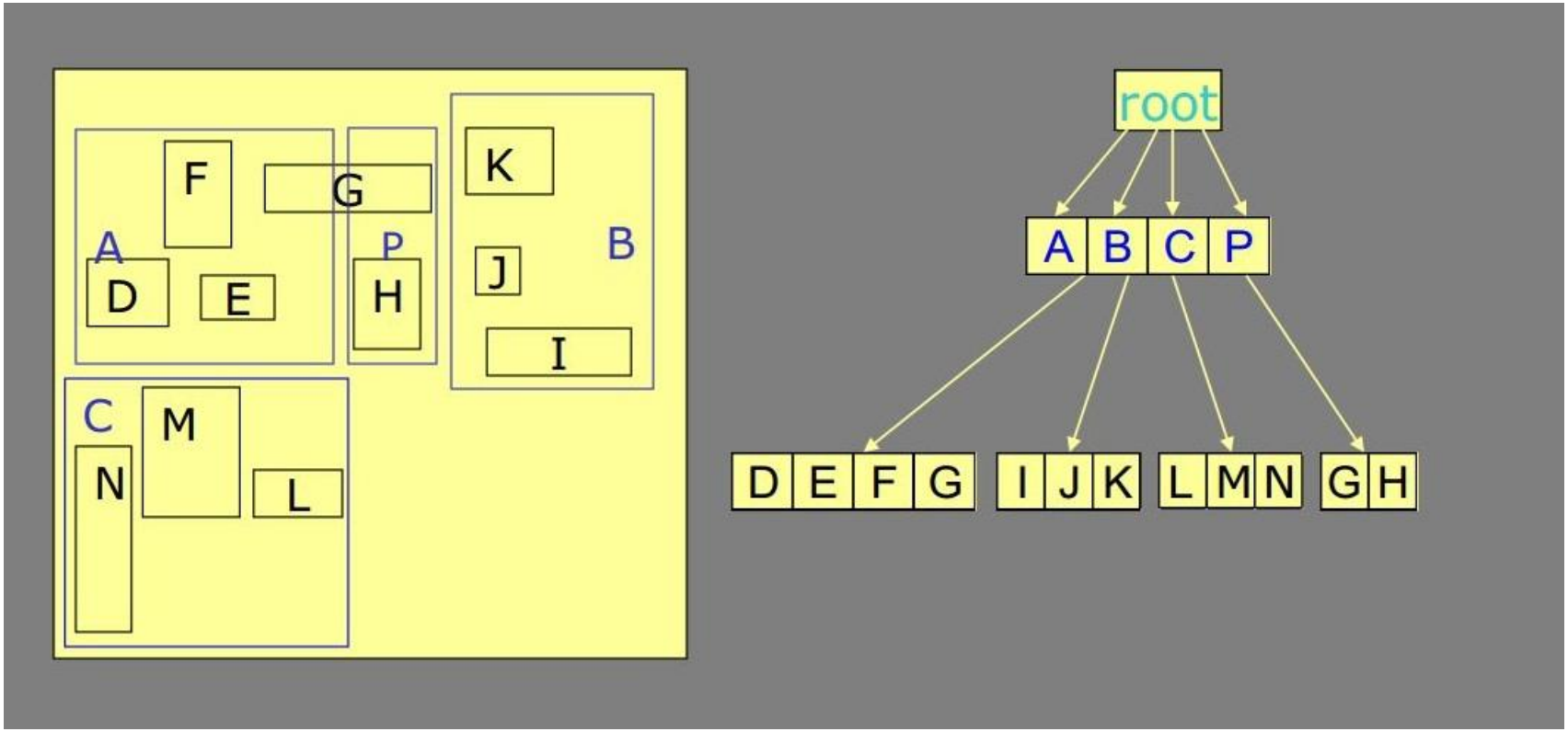
R-Tree vs. R+-Tree



R-Tree vs. R+-Tree

Ein Problem: Die Verwaltung von Feature-Objekten mit einer räumlichen Ausdehnung. Wenn überhaupt kein überlappungsfreier MBR gefunden werden kann, der das Feature-Objekt komplett umfasst. In diesem Fall erstreckt sich ein Feature-Objekt über mehrere MRBs und muss geteilt werden. → Dies führt zu Mehrfacheinträgen im R+-Baum.

R-Tree vs. R+-Tree



R-Tree vs. R+-Tree

Trotzdem besser als R-Baum: Insgesamt kann häufig eine Verbesserung der Sucheffizienz trotz der genannten Probleme gegenüber dem R-Baum festgestellt werden.

Jedoch versagt auch dieser Baum im hochdimensionalen Raum. → X-Tree (Wird in diesem Vortrag nicht behandelt)

Literatur- & Abbildungsverzeichnis

Literaturverzeichnis

- http://www.ifis.cs.tu-bs.de/sites/default/files/course/wsem1112gischap05_pdf_13685.pdf
- https://slideplayer.com/slide/4749198/
- https://www.techfak.uni-bielefeld.de/~mhanheid/lehre/alggeo/kdbaeume.pdf
- http://www.gdmc.nl/publications/2005/Spatial_Access_Methods.pdf
- https://www.cise.ufl.edu/~mschneid/files/Spatial%20Data%20Management.pdf
- http://ls11-www.cs.tu-dortmund.de/people/gutweng/AD08/VO23_26_QuadTreesWS08.pdf
- https://www.academia.edu/15213636/A_Review_on_Spatial_Access_Methods
- https://core.ac.uk/download/pdf/51447664.pdf
- https://dbs.uni-leipzig.de/file/RBaum_0.pdf

Literaturverzeichnis

- http://docplayer.org/31263841-Quadtrees-christian-hoener-zu-siederdissen.html
- https://d-nb.info/1105876586/34
- http://www.dbs.ifi.lmu.de/Lehre/GIS/WS1415/Skript/GIS_WS14_04_part2.pdf
- http://www.dbs.ifi.lmu.de/Lehre/GIS/WS1415/Skript/GIS_WS14_04_part3.pdf
- http://www.inf.uni-konstanz.de/dbis/teaching/ws0304/datatypes/download/slides-r-tree.pdf
- https://www.informatik.uni-leipzig.de/~stjaenicke/ag/7-Bereichssuche.pdf
- http://mlwiki.org/index.php/Kd-Trees
- http://wwwpub.zih.tu-dresden.de/~photo/teachlets/kdTree/index.html
- https://sebastianhaeni.github.io/space-partitioning/docs/#chapter-4-1

Literaturverzeichnis

- https://epb.bibl.th-koeln.de/frontdoor/deliver/index/docId/244/file/thesis.pdf
- https://cg.informatik.uni-freiburg.de/course_notes/info2_14_bereichsBaum.pdf
- https://www.cs.cmu.edu/~ckingsf/bioinfo-lectures/kdtrees.pdf
- http://mlwiki.org/index.php/Quad_Trees
- http://www.inf.uni-konstanz.de/dbis/teaching/ws0304/datatypes/download/slides-grid-file.pdf
- http://mlwiki.org/index.php/Grid_File_Index
- http://www-lehre.informatik.uni-osnabrueck.de/~dbs/2001/skript/node36.html
- http://www.cs.cmu.edu/~christos/courses/826.S10/FOILS-pdf/140_SAMs4.pdf
- http://mlwiki.org/index.php/R-Tree

Abbildungsverzeichnis

Folie	Abbildungsverzeichnis	Bearbeitet
6	https://d-nb.info/1105876586/34, **S.106**	✓
8	https://www.informatik.uni-leipzig.de/~stjaenicke/ag/7-Bereichssuche.pdf, **S.11**	
14, 16-21	http://www.gdmc.nl/publications/2005/Spatial_Access_Methods.pdf, **S.2**	✓ **(18,19,21)**
25	https://epb.bibl.th-koeln.de/frontdoor/deliver/index/docId/244/file/thesis.pdf, **S.55**	
26	https://www.informatik.uni-leipzig.de/~stjaenicke/ag/7-Bereichssuche.pdf, **S.19**	✓
27	http://mlwiki.org/index.php/Quad_Trees	
28	https://epb.bibl.th-koeln.de/frontdoor/deliver/index/docId/244/file/thesis.pdf, **S.57**	
29-39	http://www.dbs.ifi.lmu.de/Lehre/GIS/WS1415/Skript/GIS_WS14_04_part3.pdf, **S.11**	✓
40	https://www.techfak.uni-bielefeld.de/~mhanheid/lehre/alggeo/kdbaeume.pdf, **S.3**	
46,48,50,52	http://www.inf.uni-konstanz.de/dbis/teaching/ws0304/datatypes/download/slides-grid-file.pdf	✓
54	http://mlwiki.org/index.php/Grid_File_Index	✓
61,63,65,67,69,76	http://mlwiki.org/index.php/R-Tree	✓
71,73,80,81,83	http://www.inf.uni-konstanz.de/dbis/teaching/ws0304/datatypes/download/slides-r-tree.pdf	✓

Vielen Dank für Eure Aufmerksamkeit

Farzin Ranjbar Mirzakhani

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG