Non-Standard-Datenbanken

First-n-, Top-k- und Skyline-Anfragen

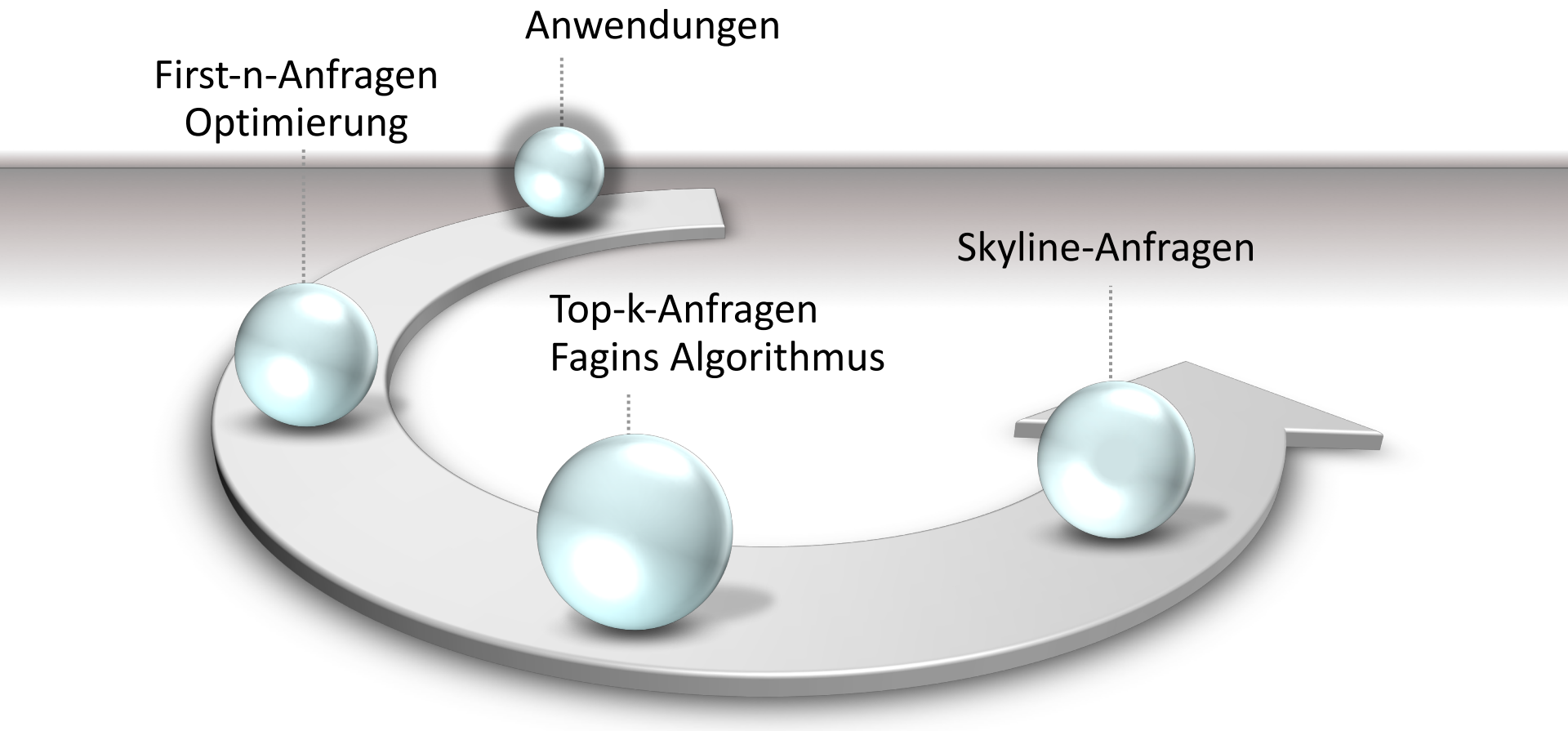
Prof. Dr. Ralf Möller

Universität zu Lübeck

Institut für Informationssysteme

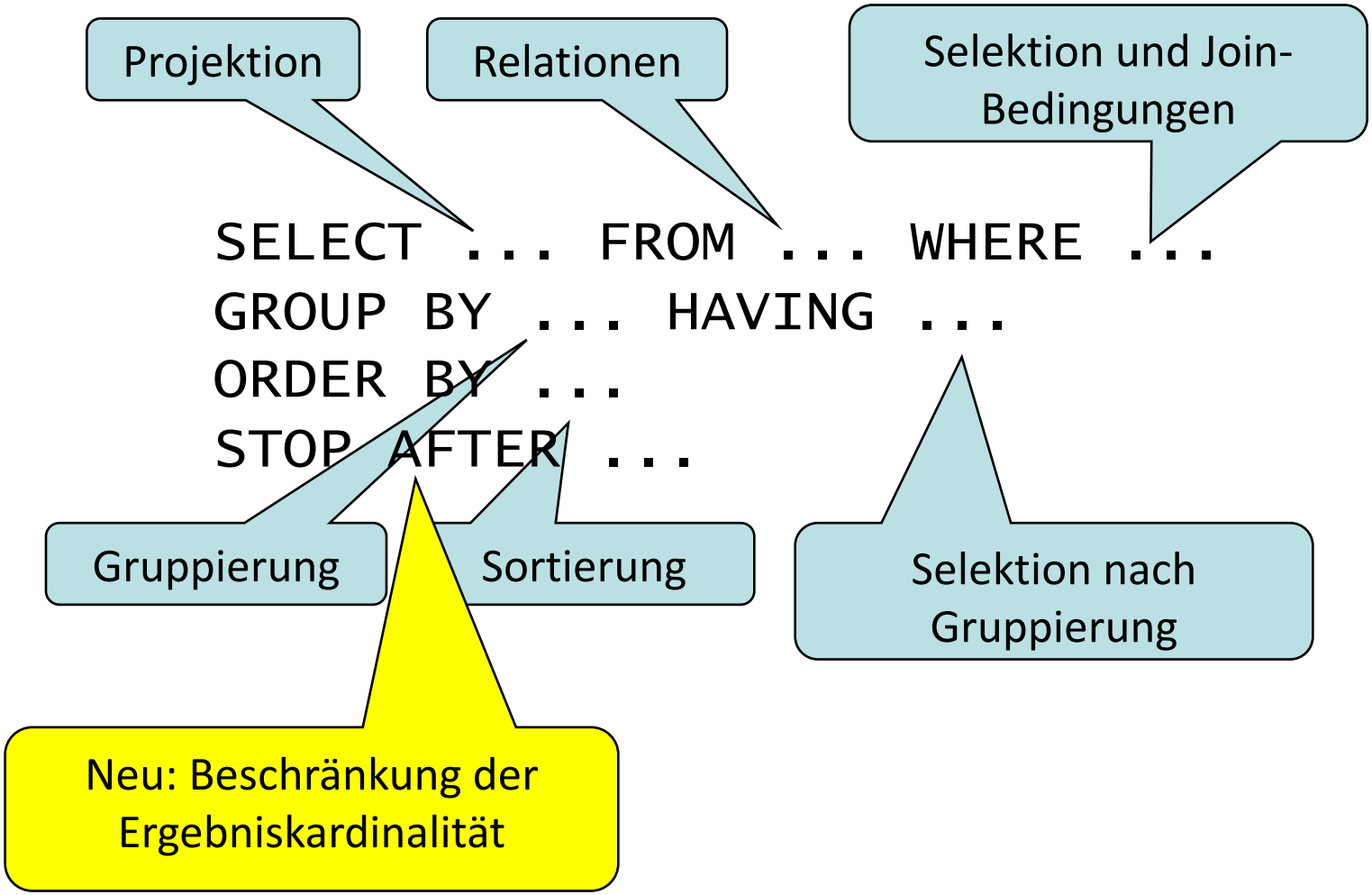
Non-Standard-Datenbanken

First-n und Top-k-Anfragen



Danksagung an Felix Naumann für Material
aus VL Informationsintegration, WS 05/06

STOP AFTER – Syntax



M.J. Carey, Donald Kossmann: On Saying "Enough Already!" in SQL. SIGMOD Conference: 219-230, **1997**

STOP AFTER – Semantik

- Ohne Sortierung
  - Willkürlich n Tupel aus dem SQL-Ergebnis
- Mit Sortierung
  - Erste n Tupel aus dem SQL-Ergebnis (sortiert)
  - Bei Gleichheit bzgl. des Sortierprädikats willkürliche Auswahl falls n+1-tes Tupel usw. gleich n-tem Tupel

- Daraus folgt:
  Falls Ergebnis nur bis zu n Tupel → keine Wirkung

STOP AFTER – Beispiel

```
SELECT h.name, h.adresse, h.tel
FROM   hotels h, flughäfen f
WHERE  f.name = ‚LBC‘
ORDER BY distance(h.ort, f.ort)
STOP AFTER 5
```

- Ergebnis: 5 Hotels mit aufsteigender Entfernung zu LBC
- Können wir Indexstrukturen für distance() nutzen?

STOP AFTER: Berechnete Anzahl

- Liste Name und Umsatz der 10% umsatzstärksten Softwareprodukte

```
SELECT p.name, v.umsatz
FROM   Produkte p, Verkäufe V
WHERE  p.typ = ´software´
AND    p.id = v.prod_id
ORDER BY v.umsatz DESC
STOP AFTER (
                SELECT count(*) / 10
                FROM   Produkte p, Verkäufe V
                WHERE  p.typ = ´software´
                AND    p.id = v.prod_id
                )
```

Praktische Ausprägungen

PostgreSQL:

```
SELECT ...
FROM ...
...
LIMIT { count | ALL }
OFFSET start
```

- Verwendung in Kombination mit `ORDER BY` möglich und sinnvoll

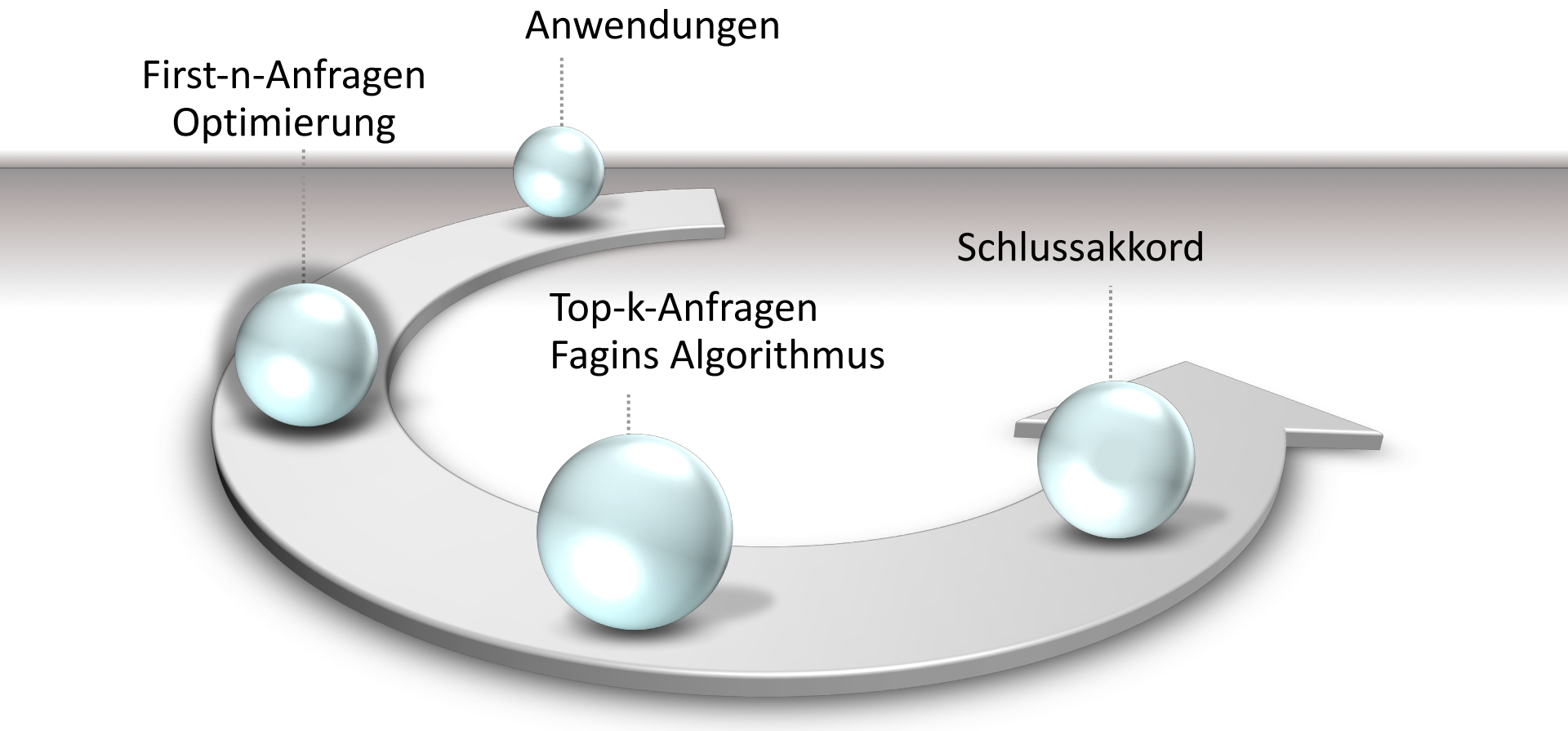
- Falls der Ausdruck ***count*** sich zu `NULL` evaluiert, wird `ALL` angenommen.
- Falls der Ausdruck ***start*** sich zu `NULL` evaluiert, wird 0 angenommen.

SQL Server:

```
SELECT TOP n WITH TIES FROM tablename
```

Non-Standard-Datenbanken

First-n und Top-k-Anfragen



Danksagung an Felix Naumann für Material
aus VL Informationsintegration, WS 05/06

Optimierung mit Stop-Operator

- Platzierung des Stop Operators im Anfrageplan
- Fundamentales Problem: Frühe Platzierung vorteilhaft aber risikoreich
  - Vorteil: Kleine Zwischenergebnisse $\Rightarrow$ geringe Kosten
  - Risiko: Endergebnis nicht groß genug $\Rightarrow$ Erneute Ausführung
- Zwei Strategien
  - „Konservativ“ und „aggressiv“

Optimierung mit Stop-Operator

- Konservative Strategie
  - Kostenminimal:
    Platziere Stop so früh wie möglich in Plan.
  - Korrekt: Platziere Stop nie so, dass Tupel entfernt werden, die später eventuell gebraucht werden.
    - Operatoren, die Tupel filtern, müssen also früher ausgeführt werden.

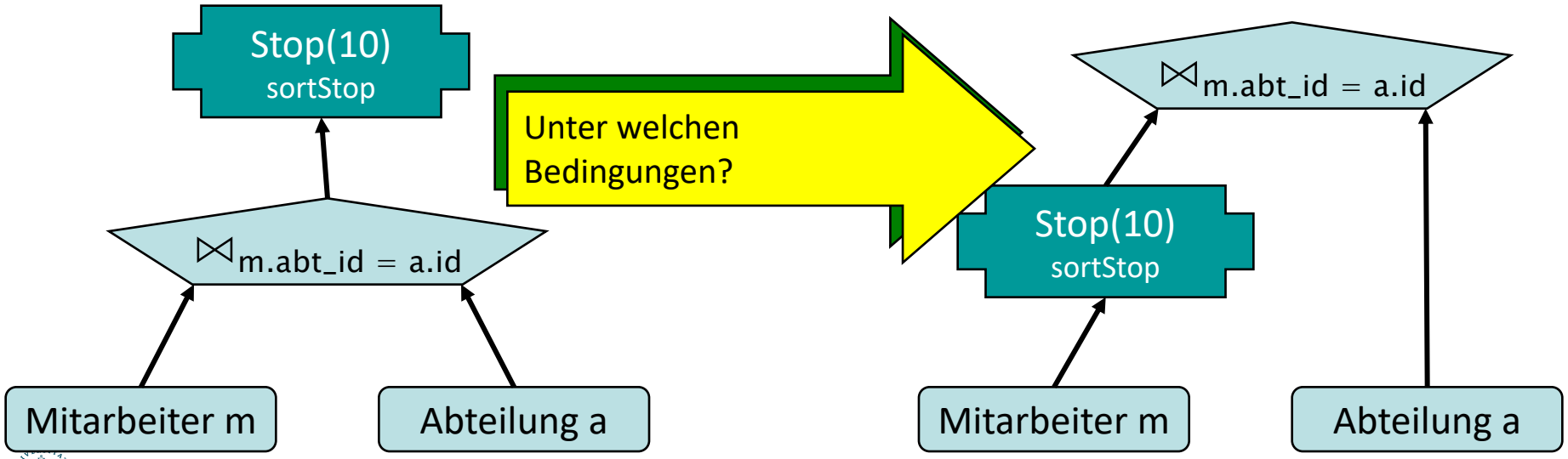
STOP AFTER: Optimierung

```
SELECT *
FROM   mitarbeiter m, abteilung a
WHERE  m.abt_id = a.id
ORDER BY m.gehalt DESC
STOP AFTER 10
```

- Wie würden Sie den Anfragebeantwortungsplan gestalten?
- Unter welchen Bedinungen ist eine Optimierung möglich?

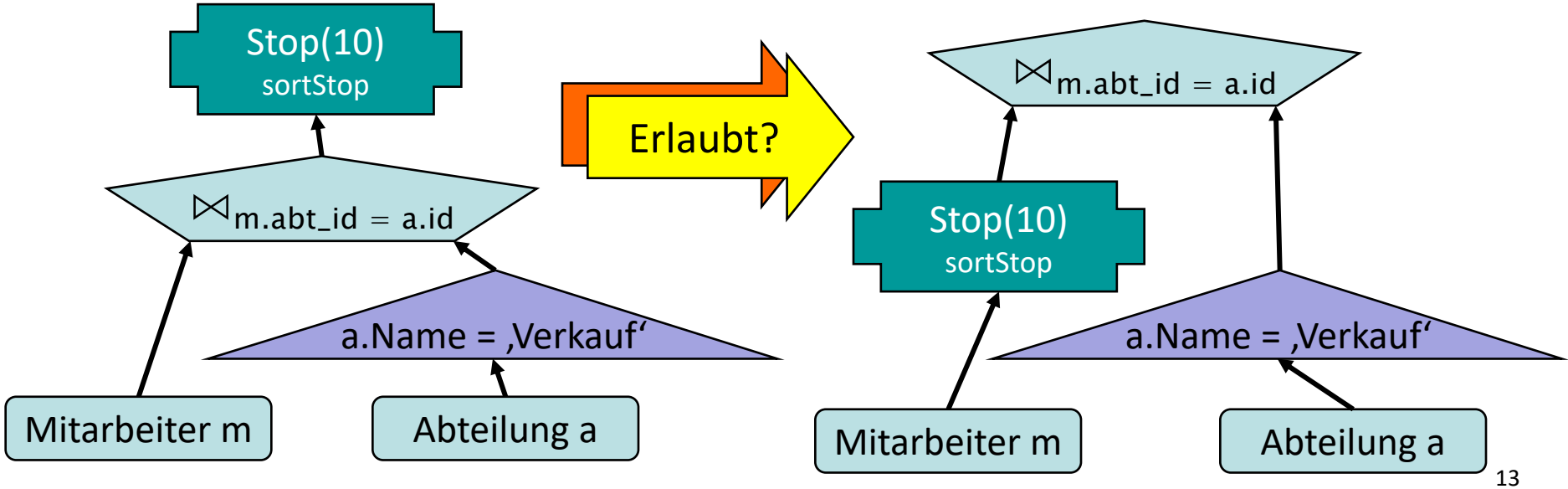
Optimierung mit Stop-Operator

```
SELECT *
FROM   mitarbeiter m, abteilung a
WHERE  m.abt_id = a.id
ORDER BY m.gehalt DESC
STOP AFTER 10
```



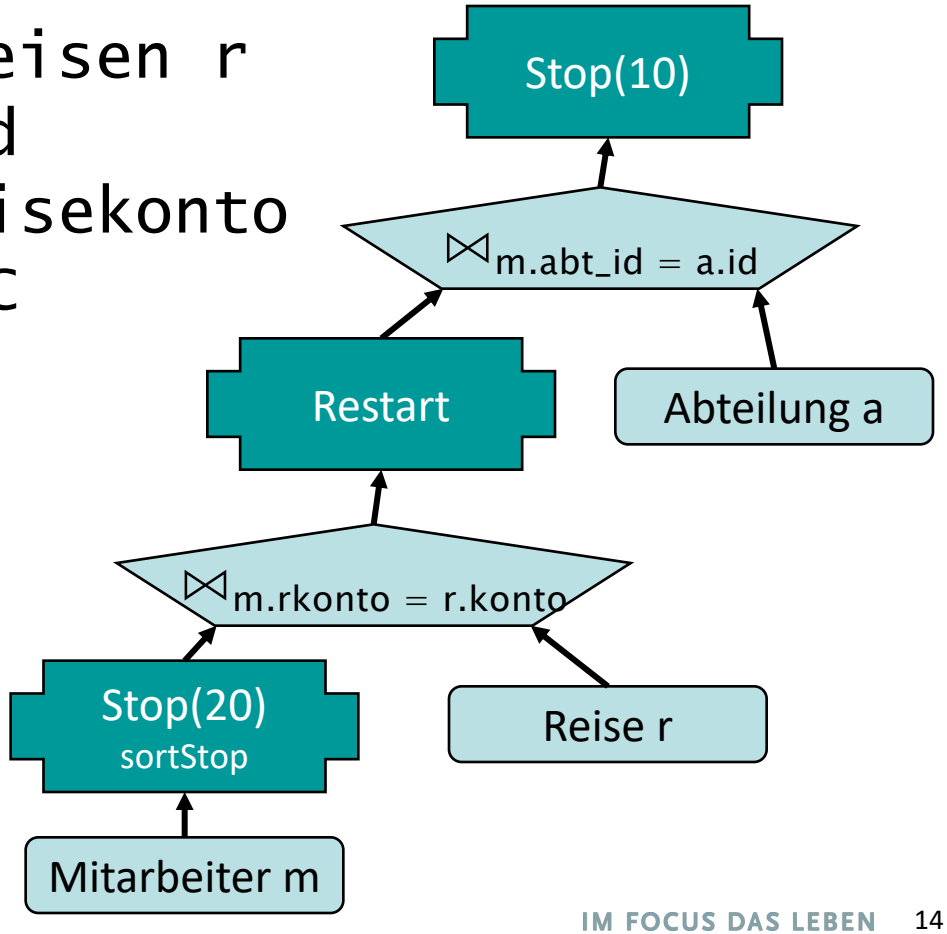
Optimierung mit Stop-Operator

```
SELECT *
FROM   mitarbeiter m, abteilung a
WHERE  m.abt_id = a.id
AND    a.name = ‚Verkauf‘
ORDER BY m.gehalt DESC
STOP AFTER 10
```



Optimierung mit Stop-Operator (aggressiv)

```
SELECT *
FROM   mitarbeiter m,
       abteilung a, reisen r
WHERE  m.abt_id = a.id
AND    r.konto = m.reisekonto
ORDER BY m.gehalt DESC
STOP AFTER 10
```



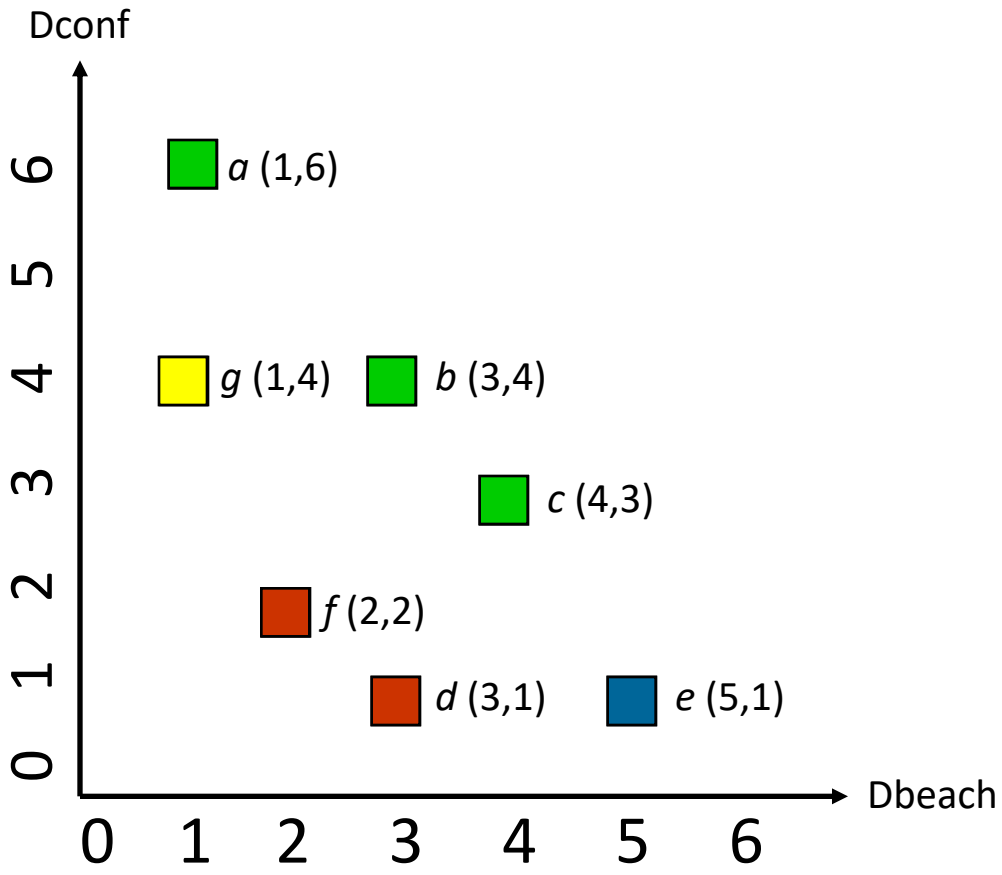
Top-k-Anfragen

- Gegeben eine Menge von Tupeln jeweils mit einer Bewertung hergeleitet aus ggf. mehreren Attributen
- Bespiele für Bewertungsfunktionen:
  - Min, Max, Linearkombination
- Beispiel: Hotels für Dienstreise
  - Distanz Strand
  - Distanz Conference Center

id	Dbeach	Dconf	Score
a	1	6	7
b	3	4	7
c	4	3	7
d	3	1	4
e	5	1	6
f	2	2	4
g	1	4	5

Ziel: Dbeach(*x*)+Dconf(*x*) minimieren

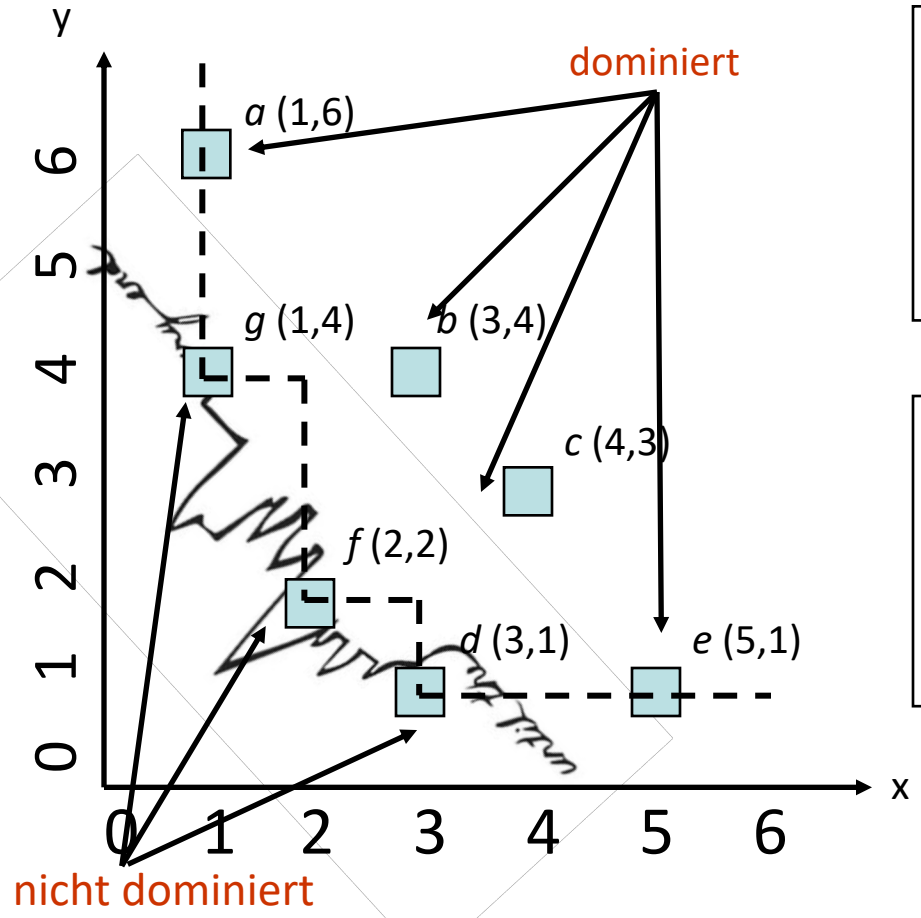
Bewertung basierend auf Dbeach(*x*)+Dconf(*x*)



id	Dbeach	Dconf	Score
a	1	6	7
b	3	4	7
c	4	3	7
d	3	1	4
e	5	1	6
f	2	2	4
g	1	4	5

Beste Hotels: *d*, *f*
Zweitbeste: *g*
Drittbeste: *e*
Nächstbeste: *a*, *b*, *c*

Alternative: Skyline-Anfragen



Dominierung:

g dominiert *b* weil 1<3 und 4=4
f dominiert *c* weil 2<4 und 2<3
d dominiert *e* weil 3<5 und 1=1

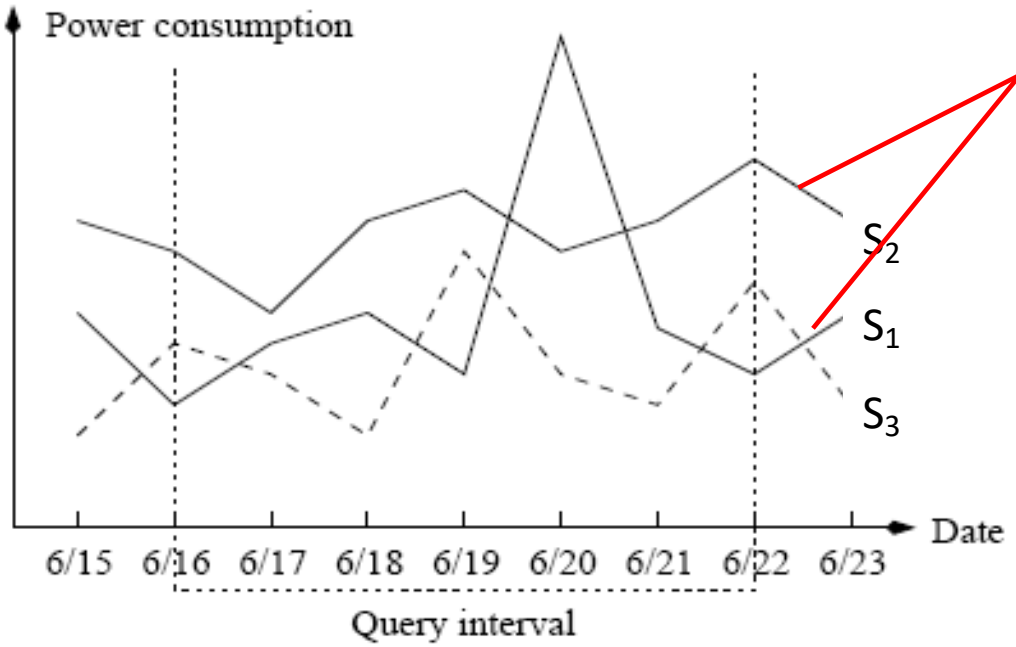
Skyline ist definiert als Menge: { *g*, *f*, *d* }

Diese Objekte werden nicht
durch ein anderes Objekt dominiert.

Vorteil: Schwellwert k muss nicht spezifiziert werden

Intervall-Skyline

- Gebeben eine Menge S von Zeitreihen und ein Intervall [i; j]. Eine Intervall-Skyline ist ein Zeitreihe s ∈ S, die nicht dominiert wird von anderen Zeitreihen aus S.
- Wir schreiben: $Sky[i:j] = \{s \in S | \nexists s' \in S, s' \succ_{[i:j]} s\}$



Bsp: S={S_1, S_2, S_3}

S_1 and S_2 sind in Sky[16:22], während S_3 durch S_2 dominiert wird.

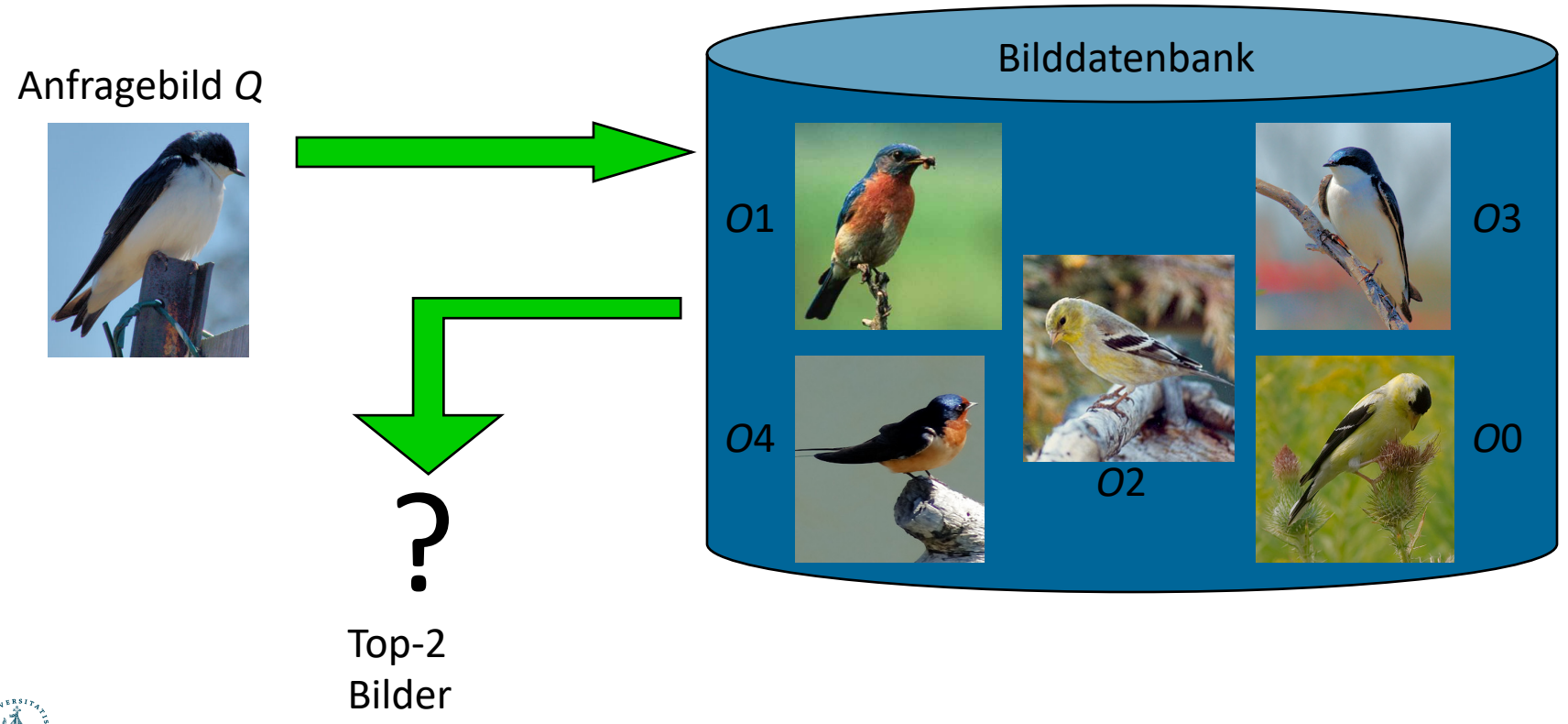
Wird im Rahmen von Stromdatenbanken später behandelt

Agenda

- Top-k-Anfragen
  - Fagins Algorithmus (FA)
  - Threshold-Algorithmus (TA), No Random Access Algorithmus (NRA) und Kombinationen davon (CA)
- Skyline-Anfragen
  - Nested-Block-Loop, Teile-und-Herrsche, Nächste Nachbarn
  - Branch-and-Bound-Skyline-Algorithmus (Verwendung von R-Bäumen)

Top-*k*-Berechnung

Nehmen wir an, die Datenbasis enthält 5 Bildobjekte *O*0, *O*1, *O*2, *O*3 and *O*4



Top-*k*-Berechnung

Datenbasis entspricht *n* x *m* Bewertungsmatrix für die Bewertungen eines jeden Objekts bzgl. aller Attribute

a1	a2	a3	a4	a5
O_3, 99	O_1, 91	O_1, 92	O_3, 74	O_3, 67
O_1, 66	O_3, 90	O_3, 75	O_1, 56	O_4, 67
O_0, 63	O_0, 61	O_4, 70	O_0, 56	O_1, 58
O_2, 48	O_4, 07	O_2, 16	O_2, 28	O_2, 54
O_4, 44	O_2, 01	O_0, 01	O_4, 19	O_0, 35

Für jedes Attribut sind die Bewertungen absteigend sortiert

Top-*k*-Berechnung – Algorithmus FA

Schritt 1:

- Lese Tupel für jede sortierte Liste (sortierter Zugriff)
- Halte, wenn k Objekte mit allen Attributwerten bekannt

Schritt 2:

- Generiere Tabellenzugriffe, um fehlende Werte zu erhalten (zufälliger Zugriff)

Schritt 3:

- Berechne die Bewertungen der gesehenen Objekte
- Gebe k höchstbewertete Objekte zurück

Top-*k*-Berechnung – Algorithmus FA

Schritt 1:

Lese Tupel für jede sortierte Liste (sortierter Zugriff)

Halte, wenn k Objekte mit allen Attributwerten bekannt

a1	a2	a3	a4	a5
*O*3, 99	*O*1, 91	*O*1, 92	*O*3, 74	*O*3, 67
*O*1, 66	*O*3, 90	*O*3, 75	*O*1, 56	*O*4, 67
*O*0, 63	*O*0, 61	*O*4, 70	*O*0, 56	*O*1, 58
*O*2, 48	*O*4, 07	*O*2, 16	*O*2, 28	*O*2, 54
*O*4, 44	*O*2, 01	*O*0, 01	*O*4, 19	*O*0, 35

id	a1	a2	a3	a4	a5
*O*3	99	90	75	74	67
*O*1	66	91	92	56	58
*O*4			70		67
*O*0	63	61		56	

Kein weiterer sortierter Zugriff nötig, denn es sind k=2 Objekte mit allen Attributwerten bekannt (Objekte *O*1 and *O*3).

Top-*k*-Berechnung – Algorithmus FA

Schritt 2:

- Generiere Tabellenzugriffe, um fehlende Werte zu erhalten (zufälliger Zugriff)

a1	a2	a3	a4	a5
$O3$, 99	$O1$, 91	$O1$, 92	$O3$, 74	$O3$, 67
$O1$, 66	$O3$, 90	$O3$, 75	$O1$, 56	$O4$, 67
$O0$, 63	$O0$, 61	$O4$, 70	$O0$, 56	$O1$, 58
$O2$, 48	$O4$, 07	$O2$, 16	$O2$, 28	$O2$, 54
$O4$, 44	$O2$, 01	$O0$, 01	$O4$, 19	$O0$, 35

id	a1	a2	a3	a4	a5
$O3$	99	90	75	74	67
$O1$	66	91	92	56	58
$O4$	44	07	70	19	67
$O0$	63	61	01	56	35

Alle fehlenden Werte für gesehene Objekte sind bestimmt. Keine weiteren Zugriffe nötig.

Top-*k*-Berechnung – Algorithmus FA

Schritt 3:

- Berechne die Bewertungen der gesehenen Objekte
- Gebe k höchstbewertete Objekte zurück

id	a1	a2	a3	a4	a5
*O*3	99	90	75	74	67
*O*1	66	91	92	56	58
*O*4	44	07	70	19	67
*O*0	63	61	01	56	35

Totale Bewertung

405

363

207

216

Top-2

Daher sind die besten Objekte für die Anfrage:
*O*3 mit Bewertung 405 and *O*1 mit Bewertung 363.

Die besten sind nicht notwendigerweise unter den ersten k.

Top-k: Fagins Algorithmus

- Gegeben: Tabelle mit Objekten
  und Bewertungsfunktion $g_A(x)$ mit $A = A_1 \wedge A_2 \wedge ... \wedge A_m$
- Behauptung:
  - Fagins Algorithmus findet die Top-k Objekte gemäß $g_A(x)$.
- Beweis:
  - Idee: Wir zeigen für jedes ungesehene Objekt *y*,
    dass es nicht unter den Top-k sein kann:
  - Notation *x*: gesehene Objekte, *y*: ungesehene Objekte
  - Wir haben absteigend sortiert: Für jedes *x* der Joinmenge nach Phase 1 und jedes Prädikat *Ai* gilt: $g_{Ai}(y) \leq g_{Ai}(x)$, *wenn* $g_{Ai}(x)$ *schon definiert und das ist ja schon für k Objekte der Fall* Abbruch-Kriterium Phase 1*)*
  - *Phase 3:* $g_{A1 \wedge A2 \wedge ... \wedge Am}(x)$ = reduce(+, {$g_{A1}(x)$, $g_{A2}(x)$,..., $g_{Am}(x)$}, 0)
  - Da ***jedes*** Attribut der k vollständig gesehenen Objekte besser, gilt:
    $g_{A1 \wedge A2 \wedge ... \wedge Am}(y) \leq g_{A1 \wedge A2 \wedge ... \wedge Am}(x)$
  - Es gibt kein *y*, das besser ist als die k gesehenen Objekte *x*
    (vielleicht sind ja die in Phase 2 vervollständigten Objekte z noch besser als die k vollständig nach Phase 1 gesehenen Objekte, aber die z werden ja auch überprüft)

Top-k: Fagins Algorithmus

- Aufwand: $O(n^{(m-1)/m}k^{1/m})$ (Beweis: siehe [Fa96])
  - n = DB-Größe; m = Anzahl der Konjunkte (Attribute/DBs)
    - Beispiel: 10000 Objekte, 3 Konjunkte, Top 10
    - $10.000^{2/3} \times 10^{1/3} = 1.000$
  - Gilt falls A_i-Werte unabhängig.
- Zum Vergleich: Naiver Algorithmus in O(nm)
  - Im Beispiel: 10.000 x 3 = 30.000
- Weiterentwicklung: Threshold Algorithmus (TA), No Random Access Algorithmus (NRA), Combined Algorithm (CA)

Ronald Fagin. Combining Fuzzy Information from Multiple Systems. PODS-96, 216-226., **1996**

Top-*k*-Berechnung – Algorithmus TA

FA benötigt relativ viel Pufferspeicher, daher wurden Verbesserung vorgeschlagen, z.B. den Algorithmus TA (Threshold Algorithm)

Hauptidee:

Einführung eines Schwellwerts, um zu bestimmen, wann der Zugriff auf die sortierten Werte beendet werden kann

Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. In Proceedings of the 20th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS '01). ACM, New York, NY, USA, 102-113., **2001**

Top-*k*-Berechnung – Algorithmus TA

Überblick über TA

Schritt 1:

- Lese Attributwerte aus jeder sortierten Liste (sortierter Zugriff)
- Für jedes gesehene Objekt x:
  - Verwende zufälligen Zugriff, um fehlende Werte zu bestimmen
  - Bestimme die Bewertung F(*x*) von Objekt *x*.
  - Falls Objekt unter den ersten top-k, behalte es im Puffer

Schritt 2:

- Bestimme Schwellwert *T* basierend auf mit sortiertem Zugriff schon gesehenen Objekte
- $T = a_1(p) + a_2(p) + \ldots + a_m(p)$ wobei *p* die aktuelle Position des sortierten Zugriffs ist
- Falls es *k* Objekte mit Gesamtbewertung ≥ *T* gibt
  dann HALTE und bestimme Ergebnis
  sonst $p := p + 1$ und GOTO Schritt 1

Top-*k*-Berechnung – Algorithmus TA

Schritt 1:

- Lese Attribute aus jeder sortierten Liste (sortierter Zugriff)
- Für jedes gesehene Objekt x:
  - Verwende zufälligen Zugriff, um fehlende Werte zu bestimmen
  - Bestimme die Bewertung F(*x*) von Objekt *x*.
  - Falls Objekt unter den ersten top-k, behalte es im Puffer

BUFFER:
(*O*3, 405)
(*O*1, 363)

p=1

a1	a2	a3	a4	a5
*O*3, 99	*O*1, 91	*O*1, 92	*O*3, 74	*O*3, 67
*O*1, 66	*O*3, 90	*O*3, 75	*O*1, 56	*O*4, 67
*O*0, 63	*O*0, 61	*O*4, 70	*O*0, 56	*O*1, 58
*O*2, 48	*O*4, 07	*O*2, 16	*O*2, 28	*O*2, 54
*O*4, 44	*O*2, 01	*O*0, 01	*O*4, 19	*O*0, 35

id	a1	a2	a3	a4	a5	F
*O*3	99	90	75	**74**	**67**	**405**
*O*1	66	**91**	**92**	56	58	**363**

Top-*k*-Berechnung – Algorithmus TA

Schritt 2:

- Bestimme Schwellwert *T* basierend auf mit sortiertem Zugriff schon gesehene Objekte
- $T = a_1(p) + a_2(p) + \ldots + a_m(p)$ wobei *p* die aktuelle Position des sortierten Zugriffs ist
- Falls es *k* Objekte mit Gesamtbewertung ≥ *T* gibt
  dann HALTE und bestimme Ergebnis
- sonst $p := p + 1$ und GOTO Schritt 1

BUFFER:
(*O*3, 405)
(*O*1, 363)

p=1

a1	a2	a3	a4	a5
*O*3, 99	*O*1, 91	*O*1, 92	*O*3, 74	*O*3, 67
*O*1, 66	*O*3, 90	*O*3, 75	*O*1, 56	*O*4, 67
*O*0, 63	*O*0, 61	*O*4, 70	*O*0, 56	*O*1, 58
*O*2, 48	*O*4, 07	*O*2, 16	*O*2, 28	*O*2, 54
*O*4, 44	*O*2, 01	*O*0, 01	*O*4, 19	*O*0, 35

id	a1	a2	a3	a4	a5	F
*O*3	99	90	75	74	67	405
*O*1	66	91	92	56	58	363

T = 99+91+92+74+67 = 423

Es gibt keine *k* Objekte mit Bewertung ≥ *T*, GOTO Schritt 1…

Top-*k*-Berechnung – TA algorithm

Schritt 1: (zweite Ausführung)

- Lese Attribute aus jeder sortierten Liste (sortierter Zugriff)
- Für jedes gesehene Objekt x:
  - Verwende zufälligen Zugriff, um fehlende Werte zu bestimmen
  - Bestimme die Bewertung F(*x*) von Objekt *x*.
  - Falls Objekt unter den ersten top-k, behalte es im Puffer

BUFFER:
(*O*3, 405)
(*O*1, 363)

p=2

a1	a2	a3	a4	a5
*O*3, 99	*O*1, 91	*O*1, 92	*O*3, 74	*O*3, 67
*O*1, 66	*O*3, 90	*O*3, 75	*O*1, 56	*O*4, 67
*O*0, 63	*O*0, 61	*O*4, 70	*O*0, 56	*O*1, 58
*O*2, 48	*O*4, 07	*O*2, 16	*O*2, 28	*O*2, 54
*O*4, 44	*O*2, 01	*O*0, 01	*O*4, 19	*O*0, 35

id	a1	a2	a3	a4	a5	F
*O*3	99	90	75	74	67	**405**
*O*1	66	91	92	56	58	**363**
***O*4**	44	07	70	19	**67**	**207**

Top-*k*-Berechnung – TA algorithm

Schritt 2: (zweite Ausführung)

- Bestimme Schwellwert *T* basierend auf mit sortiertem Zugriff schon gesehene Objekte
- $T = a_1(p) + a_2(p) + \ldots + a_m(p)$ wobei *p* die aktuelle Position des sortierten Zugriffs ist
- Falls es *k* Objekte mit Gesamtbewertung ≥ *T* gibt
  dann HALTE und bestimme Ergebnis
- sonst $p := p + 1$ und GOTO Schritt 1

BUFFER:
(*O*3, 405)
(*O*1, 363)

p=2

a1	a2	a3	a4	a5
*O*3, 99	*O*1, 91	*O*1, 92	*O*3, 74	*O*3, 67
*O*1, 66	*O*3, 90	*O*3, 75	*O*1, 56	*O*4, 67
*O*0, 63	*O*0, 61	*O*4, 70	*O*0, 56	*O*1, 58
*O*2, 48	*O*4, 07	*O*2, 16	*O*2, 28	*O*2, 54
*O*4, 44	*O*2, 01	*O*0, 01	*O*4, 19	*O*0, 35

id	a1	a2	a3	a4	a5	F
*O*3	99	90	75	74	67	**405**
*O*1	66	91	92	56	58	**363**
*O*4	44	07	70	19	67	**207**

T = 66+90+75+56+67 = 354

Beide Objekte im Puffer haben Bewertung >*T*. STOP erzeuge Antwort

Top-*k*-Berechnung - FA vs. TA

- TA betrachtet i.a. weniger Objekte als FA
  - TA hält mindestens so früh wie FA
    - Wenn wir k Objekte in FA sehen, ist ihre Bewertung größer oder gleich dem TA-Schwellwert

- TA könnte mehr zusätzliche zufällige Zugriffe erzeugen als FA
  - In TA, (*m*-1) zufällige Zugriffe pro Objekt
  - In FA, zufällige Zugriff am Ende, nur für fehlende Werte

- TA benötigt nur begrenzten Pufferspeicher (*k*) bei möglicherweise mehr zufälligen Zugriffen

- FA verwendet unbegrenzten Pufferspeicher (ggf. alle Tupel)

Top-*k*-Berechnung – Andere Methoden

Fagin et al. haben weitere relevante Varianten vorgeschlagen:

- Algorithmus NRA (No Random Access): verwendet nur sortierten Zugriff, keinen zufälligen Zugriff
- Algorithmus CA (Combined Algorithm): Kombination von TA und NRA für bessere Performanz

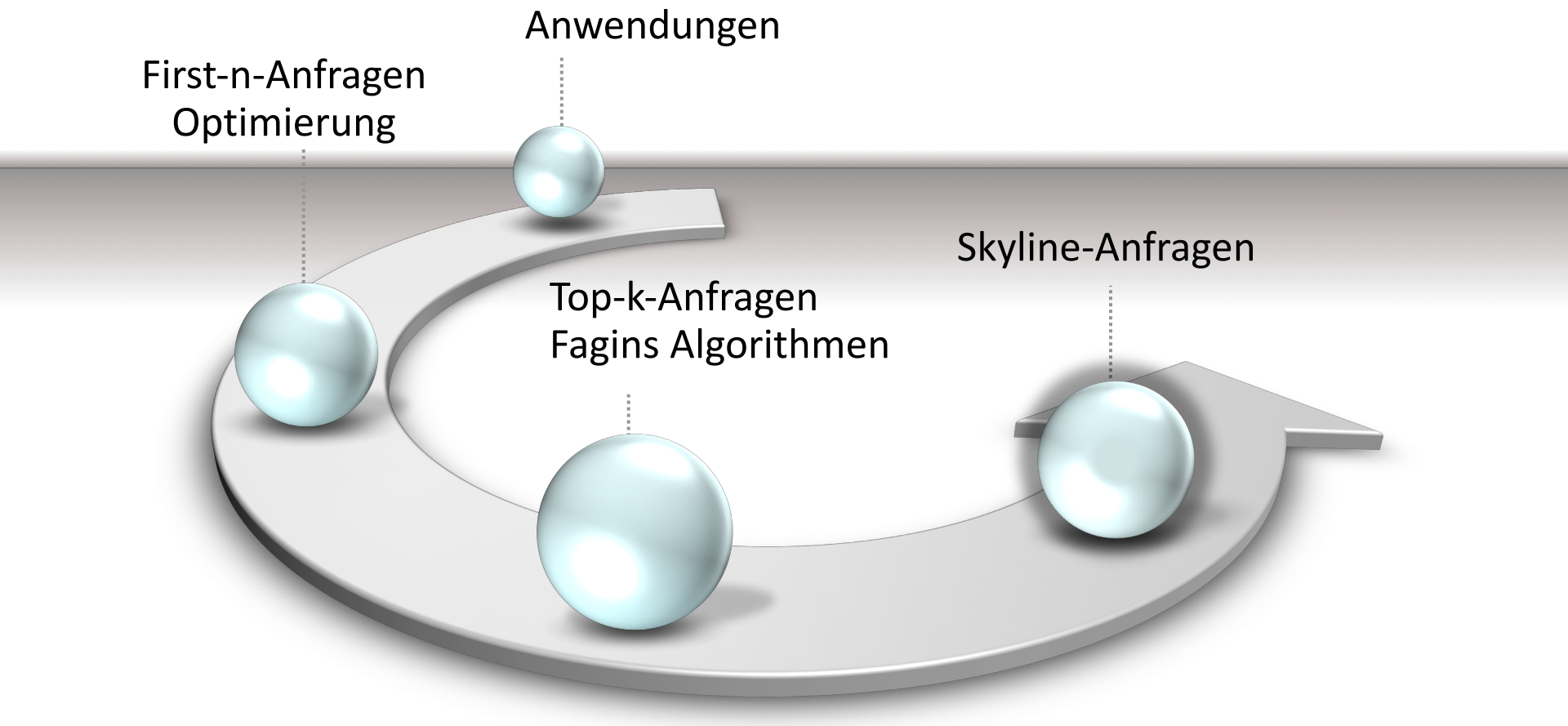
Weitere Entwicklungen:

- Verteilte Top-k-Berechnung
- Top-k-Berechnung mit Joins für Bewertung
- Top-k mit probabilistischen Daten (kommt später)
- Vermeidung der Angabe von k und der Aggregation der Attributwerte

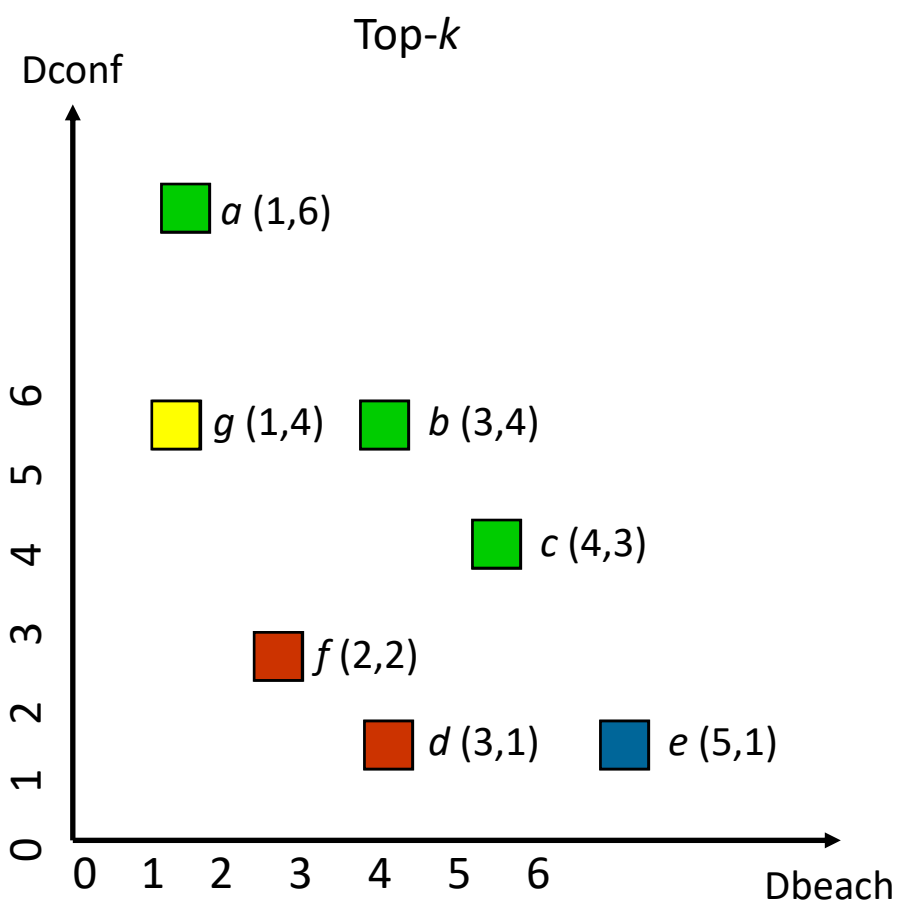
Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. In Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS '01). ACM, New York, NY, USA, 102-113., **2001**

Non-Standard-Datenbanken

Von First-n- und Top-k-Anfragen zu Skyline-Anfrage



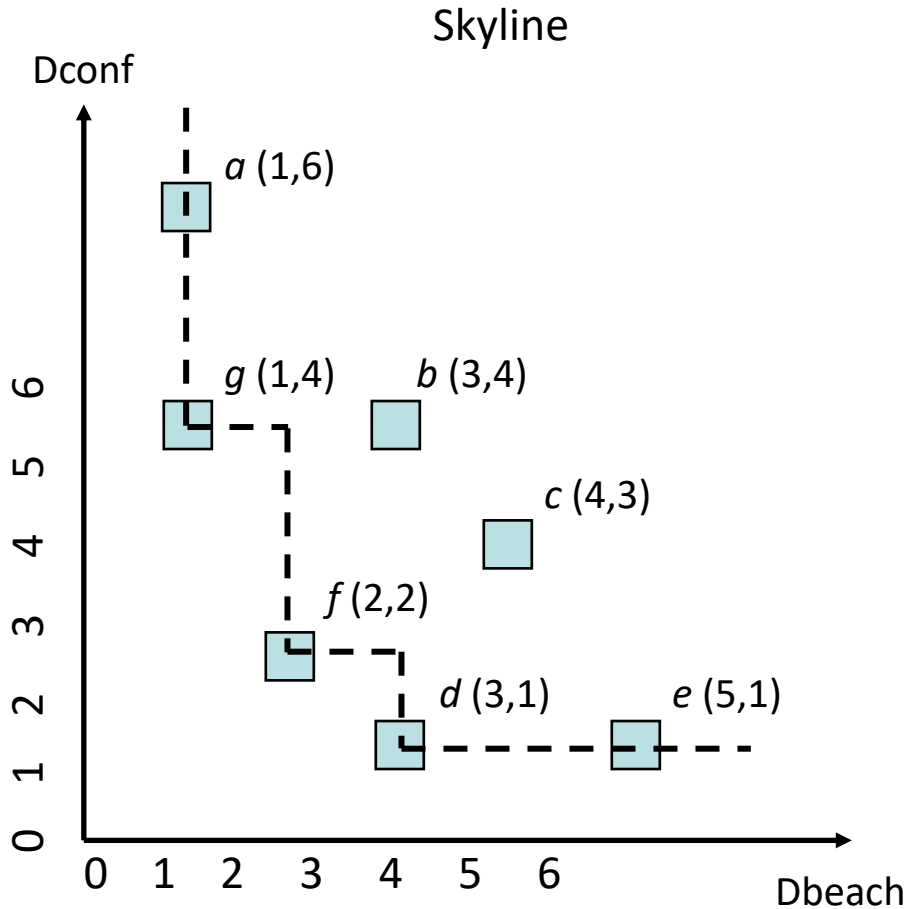
Skyline-Berechnung



f, *d* (best objects)

g (next best)

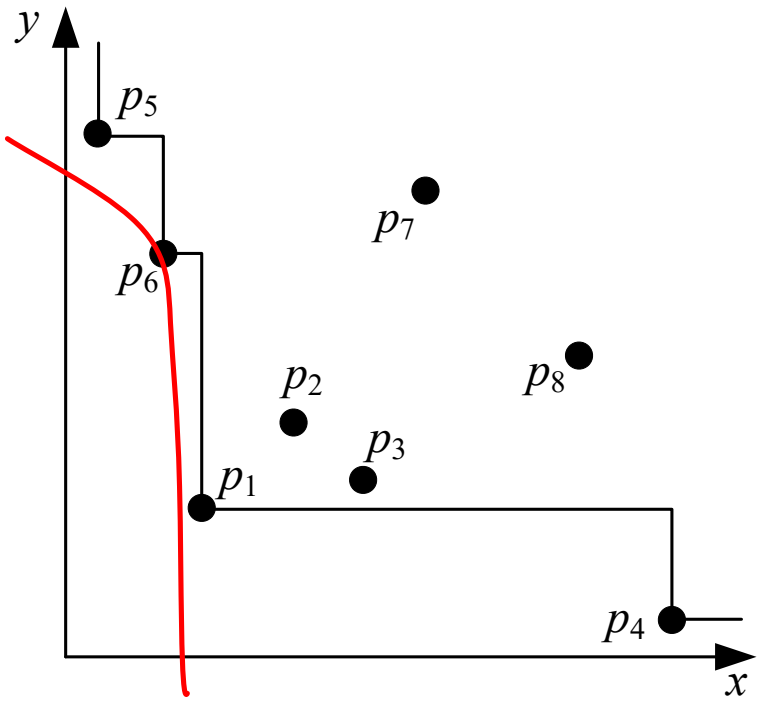
e (next best)



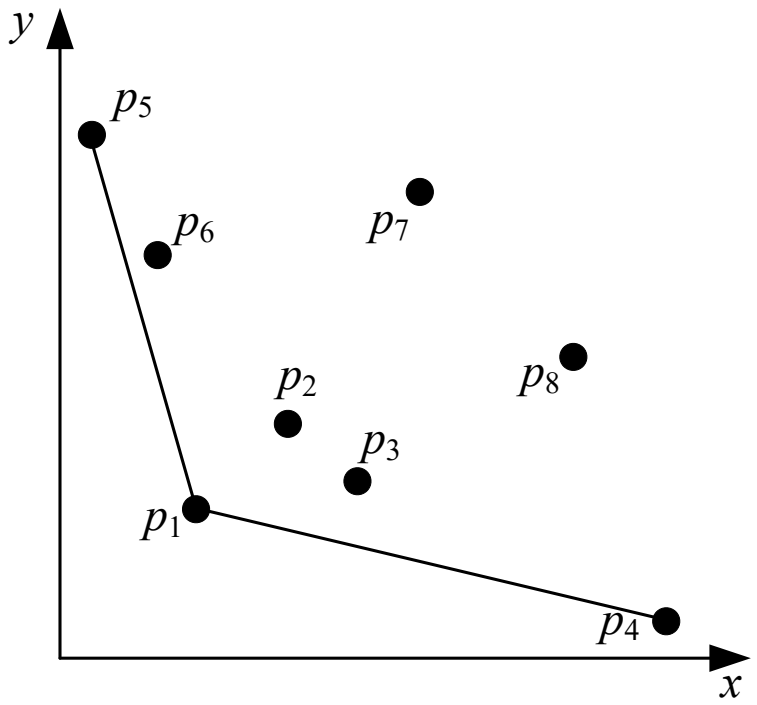
Skyline objects: *g*, *f*, *d*

Skyline vs. konvexe Hülle

Skyline



Konvexe Hülle

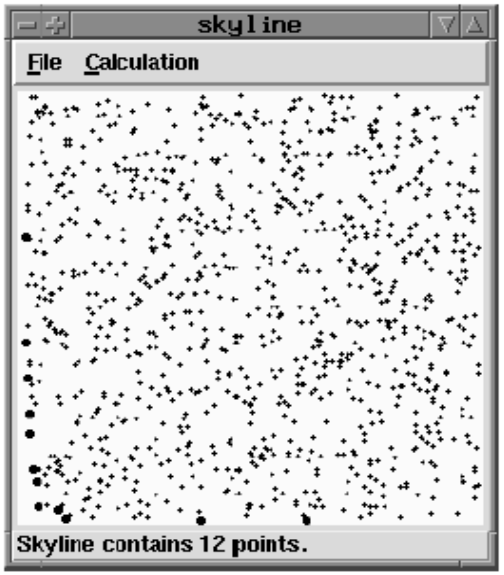


Skyline in Standard-SQL?

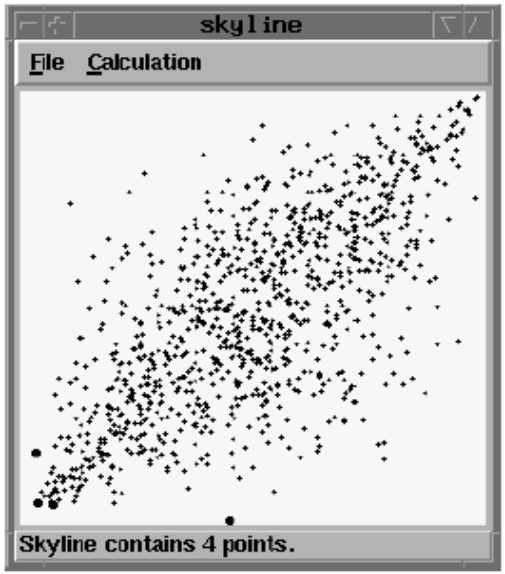
```
select k.Ort
from KostenVergleich k
where not exists
        (select * from KostenVergleich dom
         where dom.Miete <= k.Miete and dom.Beitrag <= k.Beitrag and
                 (dom.Miete < k.Miete or dom.Beitrag < k.Beitrag))
```

Datenbanksysteme werden kaum in der Lage sein,
eine solche Anfrage angemessen zu optimieren

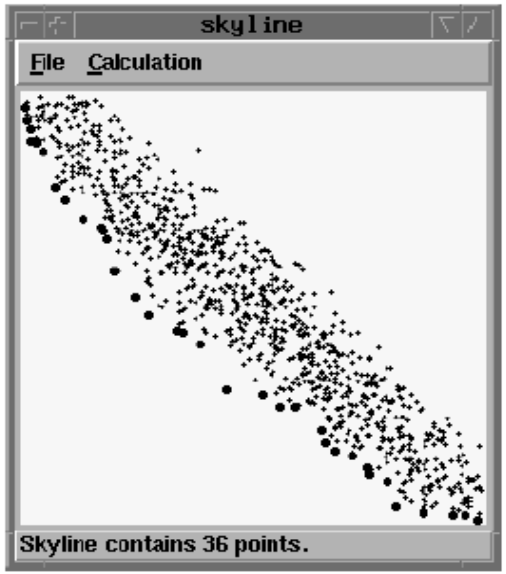
Pures SQL ineffektiv auf großen Datenmengen



indendent



correlated



anti-correlated

Je nach Korrelation der Daten
kann Skyline groß oder klein sein

Skyline als SQL-Erweiterung

```
with KostenVergleich as (select m.Ort, m.Miete, k.Beitrag
                         from Mietspiegel m, Kindergarten k
                         where m.Ort=k.Ort)

select k.Ort
from KostenVergleich k
skyline of k.Miete min, k.Beitrag min
```

```
SELECT ... FROM ... WHERE ...
GROUP BY ... HAVING ...
SKYLINE OF [DISTINCT] d1 [MIN | MAX | DIFF],
                    ..., dm [MIN | MAX | DIFF]
ORDER BY ...
```

MIN: Minimierung der Skyline-Werte, MAX: Maximierung, DIFF: Unterschiedlich (different)

Verfahren zur Skyline-Berechnung

- Index-basierte Skyline
- Block Nested Loop (BNL): Führt geschachtelte Schleifen über Datenblöcke aus, blockweises Lesen
- Divide and Conquer (DC): Teilt den Raum auf, löst das Problem in Teilräumen und fügt die Teillösungen zusammen
- Nearest-Neighbor (NN): Verwendet R-Baum-Index und führt Folge von Nächste-Nachbarn-Anfragen aus, bis Skyline-Objekte gefunden sind

Indexbasierte Skyline-Bestimmung

- Organisiere Datenpunkte $p=(x_1, x_2, \ldots, x_d)$ in d Listen, so dass p in Liste i kommt, wenn p_i am kleinsten
- Listen L_i aufsteigend sortiert (nach Attribut p_i)
- Bearbeitung von Teilmengen aus L_i mit gleichem Wert p_i

list 1		list 2	
a (1, 9)	*minC*=1	*k* (9, 1)	*minC*=1
b (2, 10)	*minC*=2	*i* (3, 2), *m* (6, 2)	*minC*=2
c (4, 8)	*minC*=4	*h* (4, 3), *n* (8, 3)	*minC*=3
g (5, 6)	*minC*=5	*l* (10, 4)	*minC*=4
d (6, 7)	*minC*=6	*f* (7, 5)	*minC*=5
e (9, 10)	*minC*=9		

Indexbasierte Skyline-Bestimmung

- Berechnung der Skyline auf Teilmengen der Punkte
- Füge die nicht dominierten Punkte in Skyline-Liste ein

list 1		list 2	
a (1, 9)	*minC*=1	*k* (9, 1)	*minC*=1
b (2, 10)	*minC*=2	*i* (3, 2), *m* (6, 2)	*minC*=2
c (4, 8)	*minC*=4	*h* (4, 3), *n* (8, 3)	*minC*=3
g (5, 6)	*minC*=5	*l* (10, 4)	*minC*=4
d (6, 7)	*minC*=6	*f* (7, 5)	*minC*=5
e (9, 10)	*minC*=9		

- Skyline := {}
- Lade erste Elemente aus jeder Liste, behandle Element mit kleinstem minC (hier a, k), wähle a und füge a zur Skyline-Liste hinzu.

- Betrachte Elemente b und k bzgl. minC, füge k zur Skyline hinzu, b dominiert von a
- Lade [i,m] ; Betrachte Elemente in [i,m]
- Füge i zur Skyline-Liste hinzu
- Algorithmus hält, weil keine anderen Elemente besser sind als i
  - Für den Test wird B-Baum in der jeweiligen Dimension benötigt
- Skyline ist {a,k,i}

Warum hält der Algorithmus mit i in der Skyline?

- i = (3, 2)
- Verwende B-Baum der Dimension 1, um 3 zu finden
- Scanne nach links bis $MinC_1$:
  b schon gesehen
  - Bei i=(4, 2) wäre c noch zu prüfen

list 1		list 2	
a (1, 9)	*minC*=1	*k* (9, 1)	*minC*=1
b (2, 10)	*minC*=2	*i* (3, 2), *m* (6, 2)	*minC*=2
c (4, 8)	*minC*=4	*h* (4, 3), *n* (8, 3)	*minC*=3
g (5, 6)	*minC*=5	*l* (10, 4)	*minC*=4
d (6, 7)	*minC*=6	*f* (7, 5)	*minC*=5
e (9, 10)	*minC*=9		

- Verwende B-Baum der Dimension 2, um 2 zu finden
- Scanne nach links bis $MinC_2$:
  k schon gesehen
  - Bei i=(3, 3) wäre h noch zu prüfen (m käme auch noch vorher)

Nur, effektiv, wenn B-Baum-Indexe vorhanden!

Meist für Punkte nicht vorgesehen

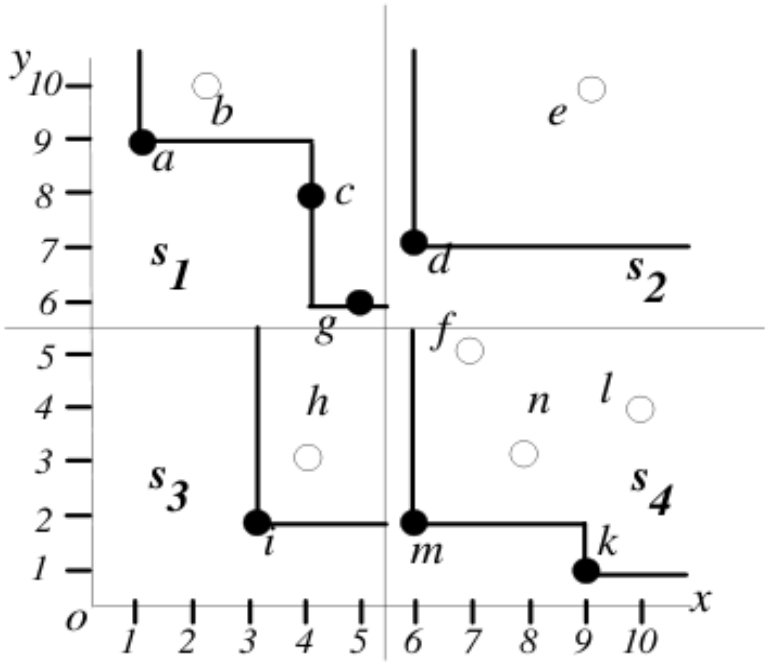
Algorithmus Block-Nested-Loop (BNL)

- Gegeben:
  - Datei 1 mit Tupeln, Hilfsdatei Datei 2
  - Puffer (Block, klein) und Fenster (klein) im Hauptspeicher
- Bis Datei 1 leer:
  - Lese nächste Tupel blockweise aus Datei 1 in Puffer
  - Für alle Tupel t im Block:
    - Für alle Tupel t' im Fenster
      - Vergleiche t mit t', prüfe, ob t dominiert wird
      - Entferne t' aus Fenster, wenn durch t dominiert
    - Übernehme t in Fenster, wenn nicht dominiert
    - Falls Fenster voll, schreibe t in Datei 2
- Wenn Tupel in Datei 2 gefunden, schreibe Fenster in Datei 2 mache mit Datei 2 als Datei 1 weiter

Stephan Börzsönyi, Donald Kossmann, Konrad Stocker. "The Skyline Operator". In Proceedings 17th International Conference on Data Engineering: 421–430, **2001**

Divide and Conquer (Sykline-Algorithmus DC)

- Zerlege Datenmenge in Partitionen, so dass jede Partition in den Speicher passt (ggf. mit Vorabfilterung pro Block – Early Skyline – siehe indexbasierte Verfahren)
- Bestimme Skyline pro Partition (z.B. mit BNL)
- Kombiniere Teillösungen zur Gesamtlösung, ggf. m-Wege-Mischen



Teillösungen	Gesamtlösung
{a,c,g}	{a,i,k}
{d}	
{i}	
{m,k}	

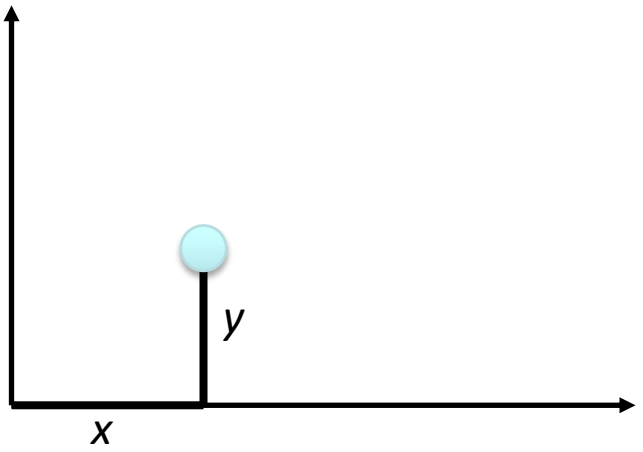
Nachteil: Hohe IO-Kosten
Funktioniert nur bei kleiner Skyline

Bewertung

- BNL besser als DC, sofern Blockgröße hoch
- Early Skyline effektiv für DC
  - Kleinere Partitionen : Algorithmus terminiert schnell
- DC ohne Early Skyline:
  - Schwach: Hohe I/O-Komplexität
- BNL-Varianten gut, wenn Skyline klein
  - Bei mehr Dimensionen (Skyline wird größer) wird DC besser
  - Dito bei mehr Speicher (weniger Partititionen)

Annahme

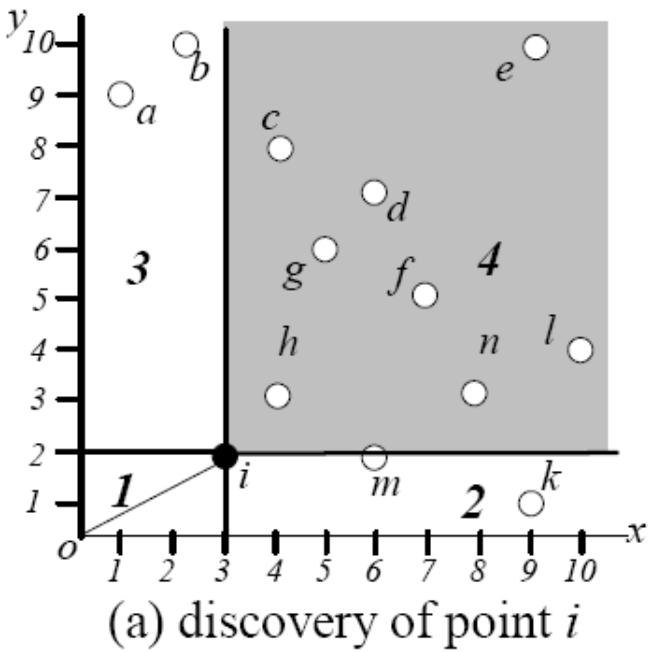
- Datenpunkte in k-d-B-Baum eingetragen
- Abstand eines Punktes vom Ursprung als Manhatten-Anstand modelliert
- NN-Anfrage mit k-d-Baum



mindist(*e.mbr*) = x + y

Skyline-Algorithmus Nächste Nachbarn (NN)

- Führe NN-Anfrage vom Ursprung aus zur Bestimmung des nächsten Nachbarn von o
- Kein Punkt in der dominierten Region wird mehr betrachtet
- Ergebnis der NN-Suche verwendet zur Partitionierung des Raumes



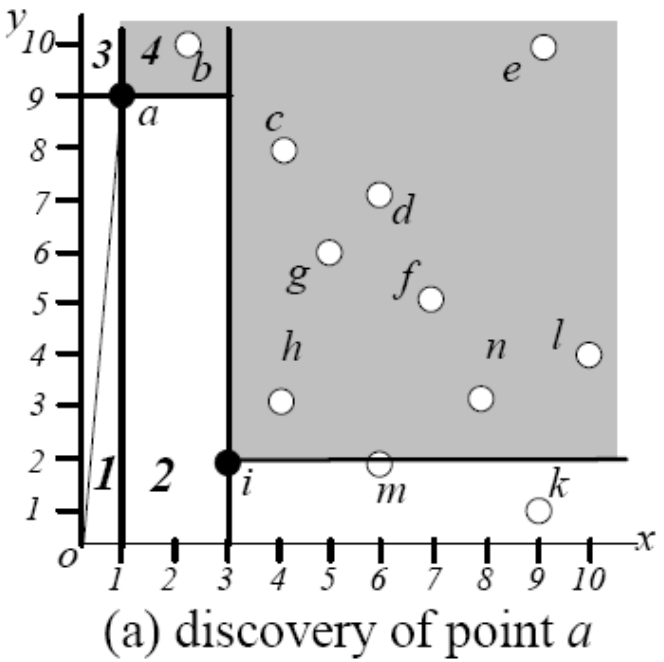
(a) discovery of point *i*

Zwei Partitionen

Partition1: Fläche 3 *Partition2: Fläche 2*

Nächste Nachbarn (NN)

- Partitionen kommen auf Agenda
- Solange Agenda nicht leer:
  - Nehme Partition von Agenda und führe NN mit entsprechendem Ursprung aus

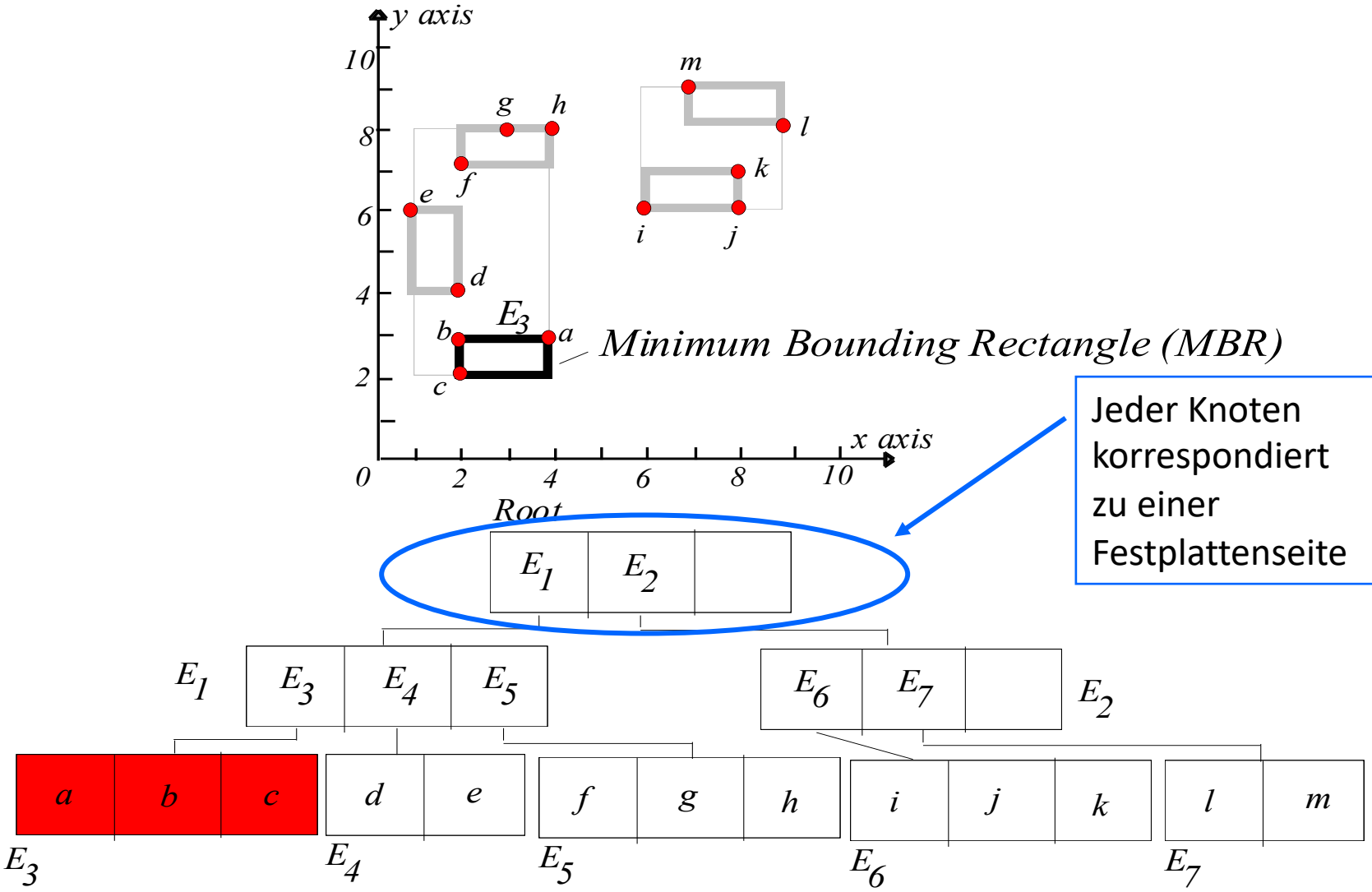


(a) discovery of point *a*

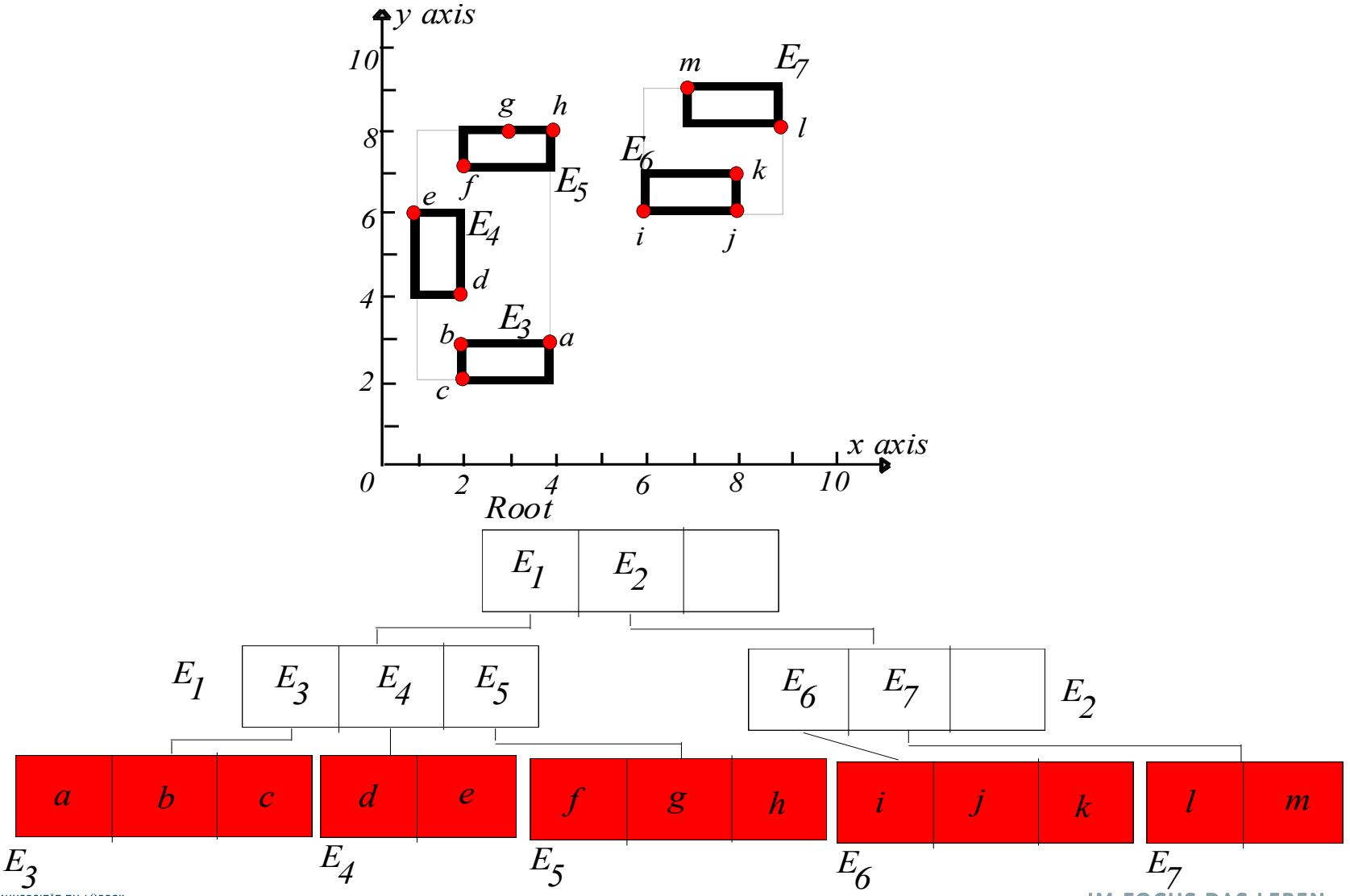
- Vergleich mit BNL, DC?
  - K-d-Baum notwendig

- Skyline mit R-Bäumen mit Punktdaten?
  - Branch and Bound Skyline (BBS)

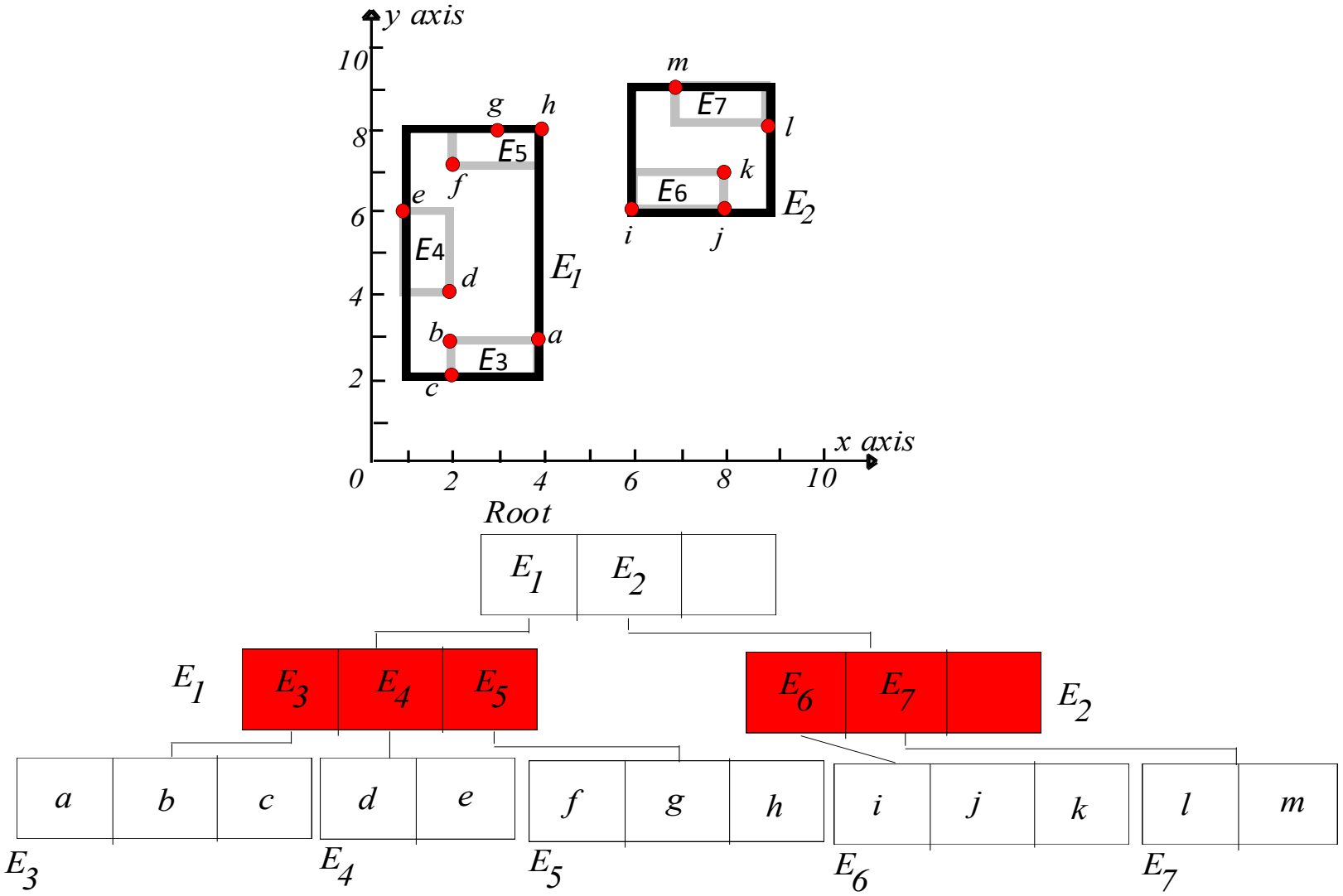
BBS-Algorithmus: R-Bäume mit Punktdaten



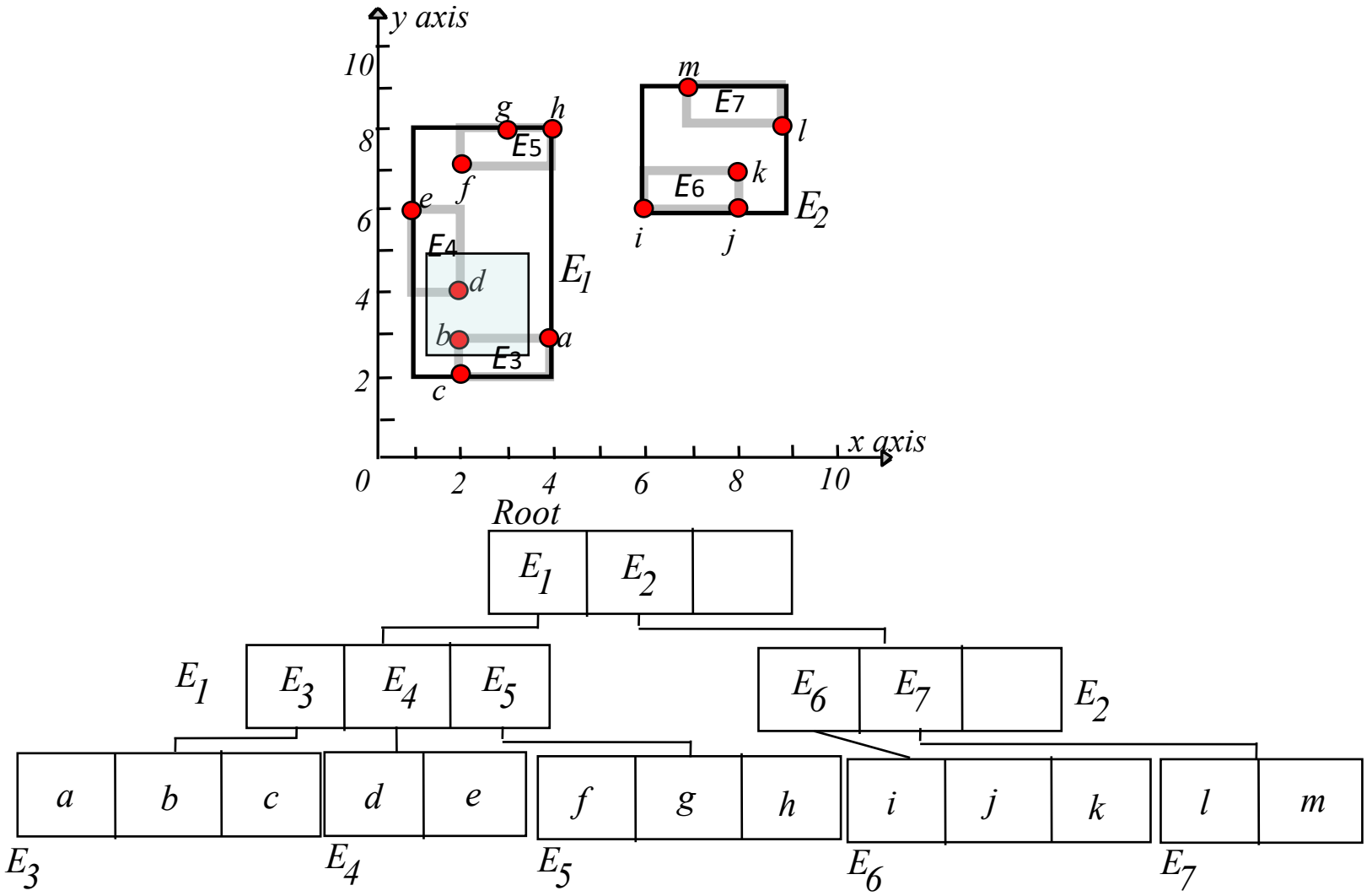
R-Bäume – Struktur



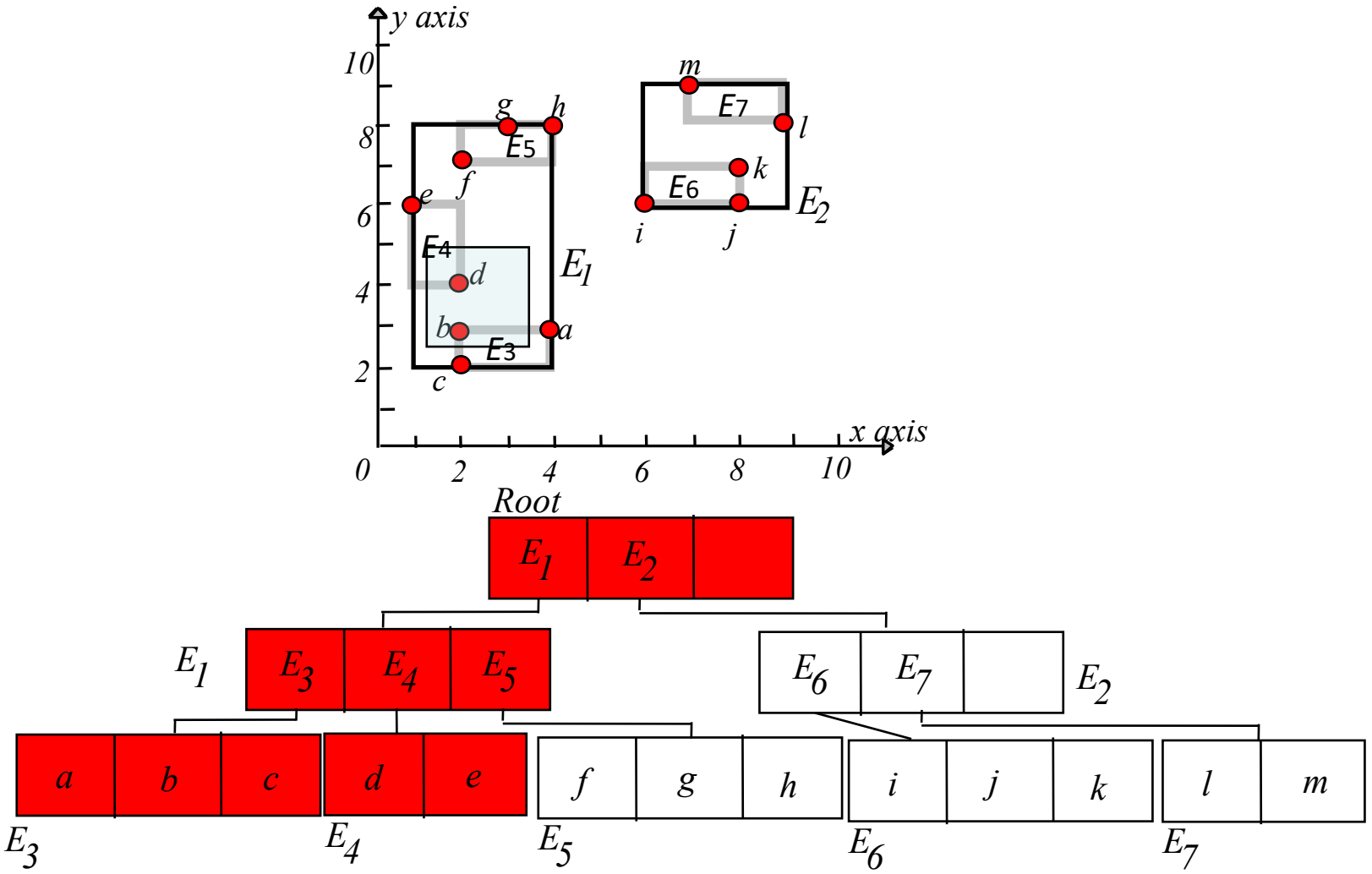
R-Bäume – Struktur



R-Bäume – Bereichsanfrage



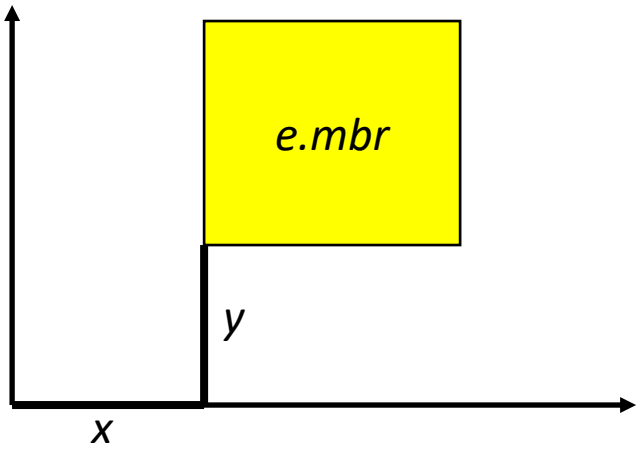
R-Bäume – Bereichsanfrage



BBS Algorithmus

Verwendung einer Prioritätswarteschlange,
in der R-Baum-Einträge e nach Manhattan-Abstand
verwaltet werden:

Links-unten von
mbr(e) bis Nullpunkt



mindist(*e.mbr*) = x + y

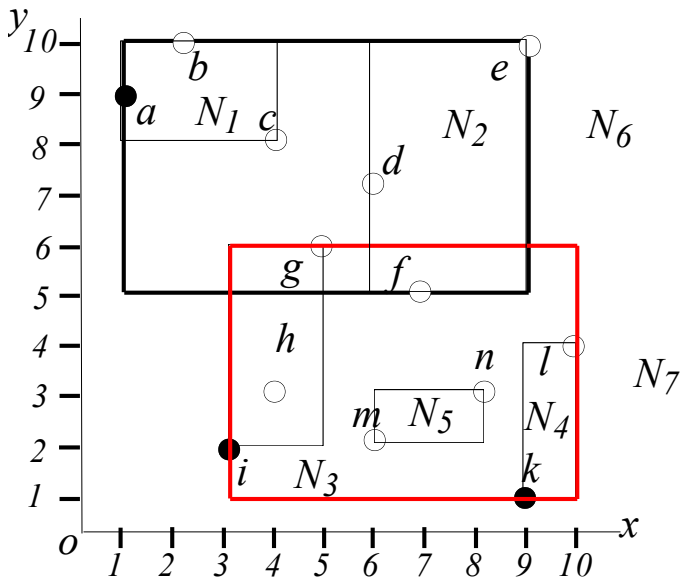
Dimitris Papadias, Yufei Tao, Kyriakos Mouratidis, and Chun Kit Hui.
Aggregate nearest neighbor queries in spatial databases,
ACM Trans. Database Syst. 30, 2, 529-576, **2005**.

BBS Algorithmus – Entscheidungen

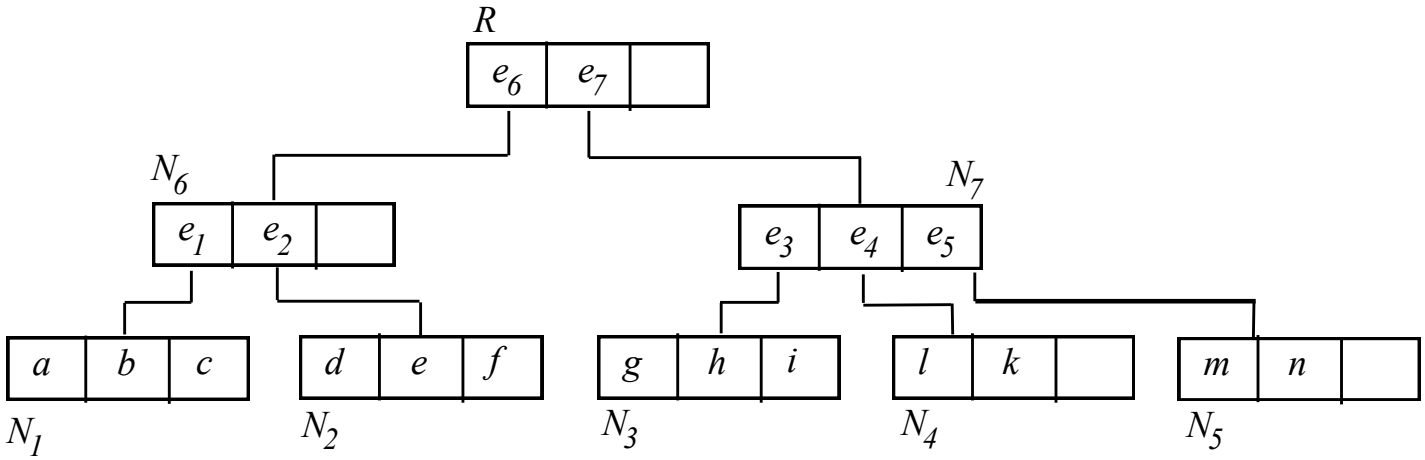
1. Wann verzweigen (branch): Welchen Teil des Suchraums soll als nächstes betrachtet werden?

2. Wie begrenzen (bound): Welche Teile des Suchraums können sicher eliminiert werden

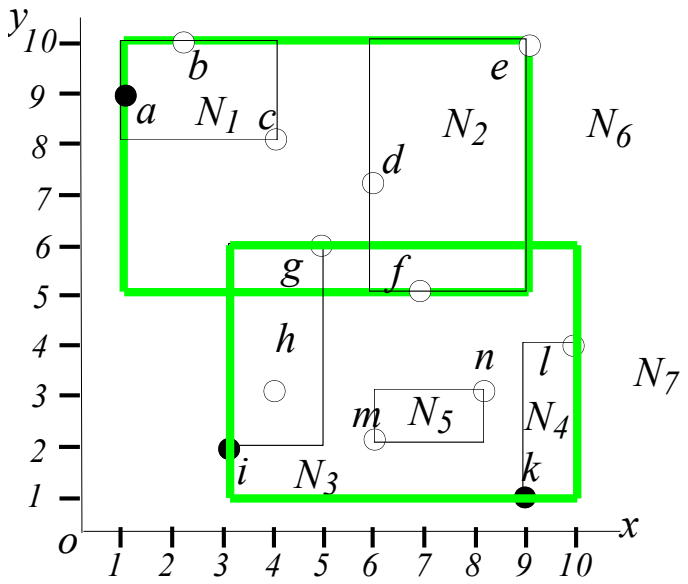
BBS Algorithmus – Beispiel



- Nehme an, alle Punkte sind in einem R-Baum eingetragen
- mindist(MBR) = Manhatten-Distanz zwischen Punkt links-unten und Ursprung

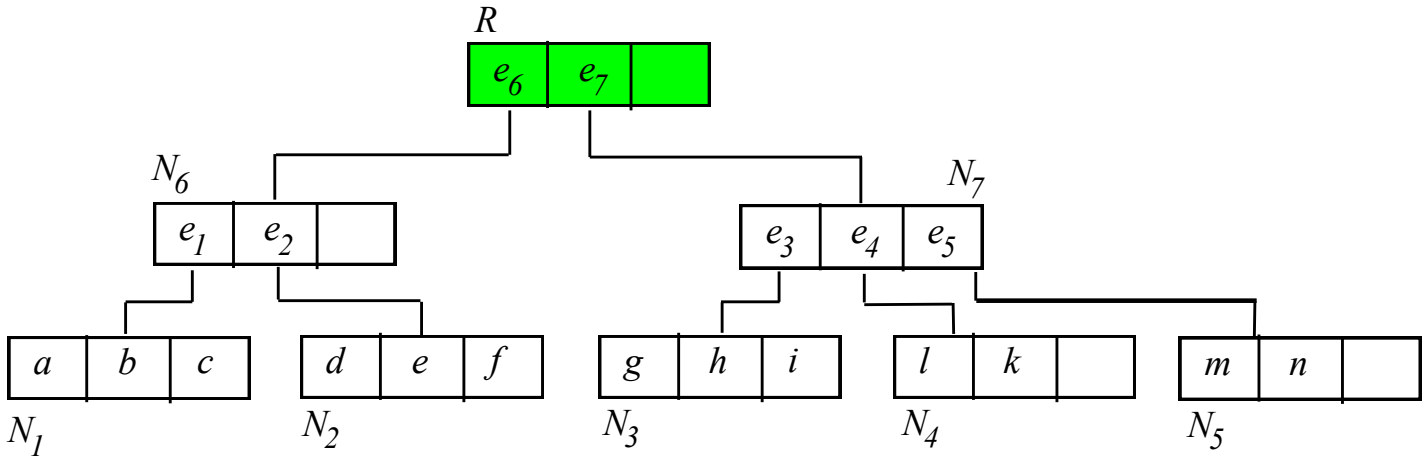


BBS Algorithmus – Beispiel

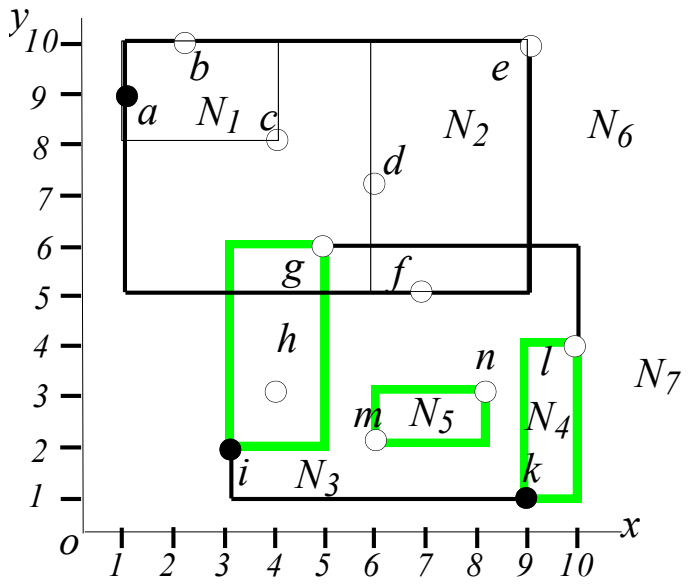


- Warteschlangenschlüssel = mindist vom MBR.

Aktion	Warteschlange	S
access root	$<e_7,4><e_6,6>$	$\varnothing$

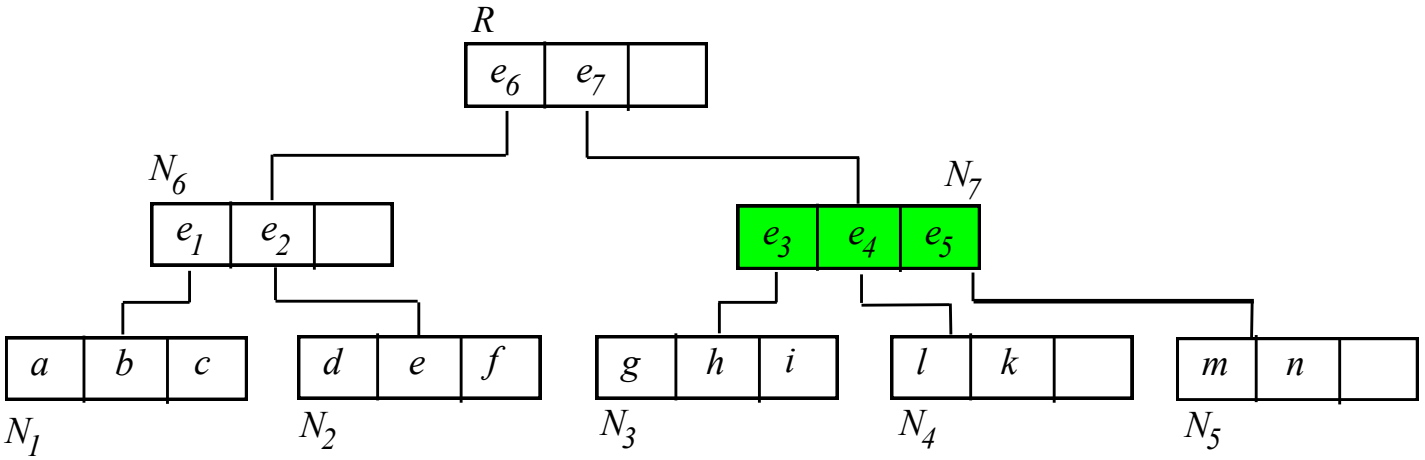


BBS Algorithmus – Beispiel

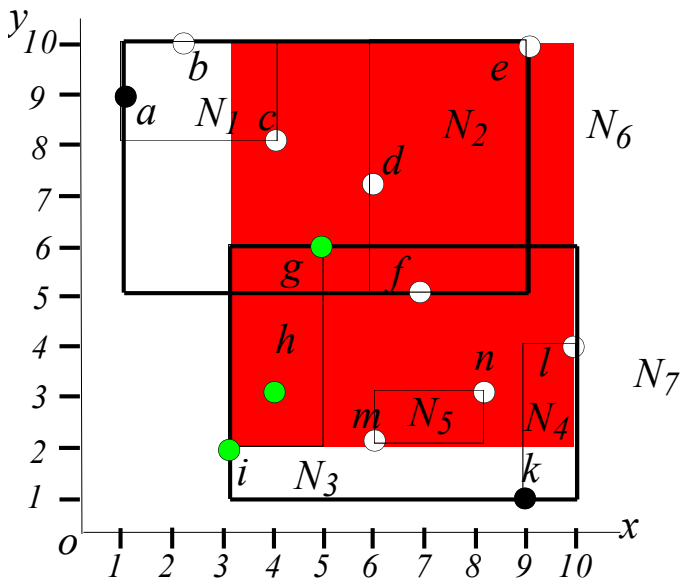


- Verarbeitung nach mindist-Werten sortiert

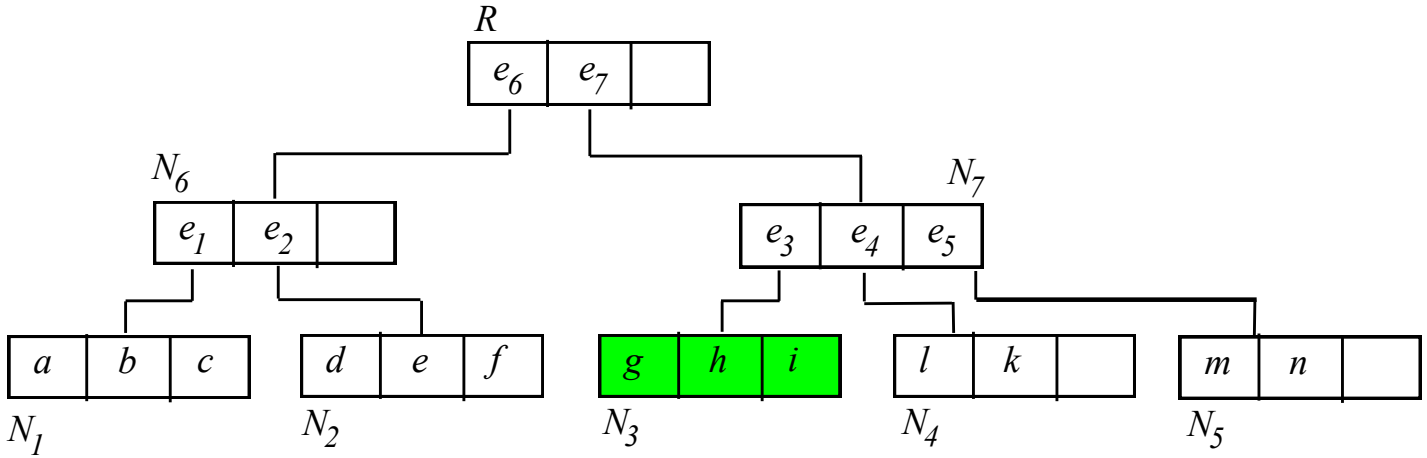
Aktion	Warteschlange	*S*
access root	$\langle e_7,4\rangle\langle e_6,6\rangle$	∅
expand e_7	$\langle e_3,5\rangle\langle e_6,6\rangle\langle e_5,8\rangle\langle e_4,10\rangle$	∅



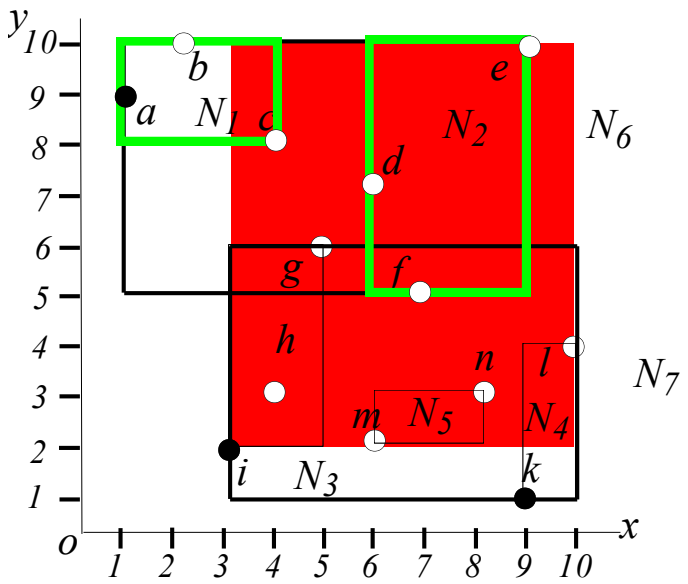
BBS Algorithmus – Beispiel



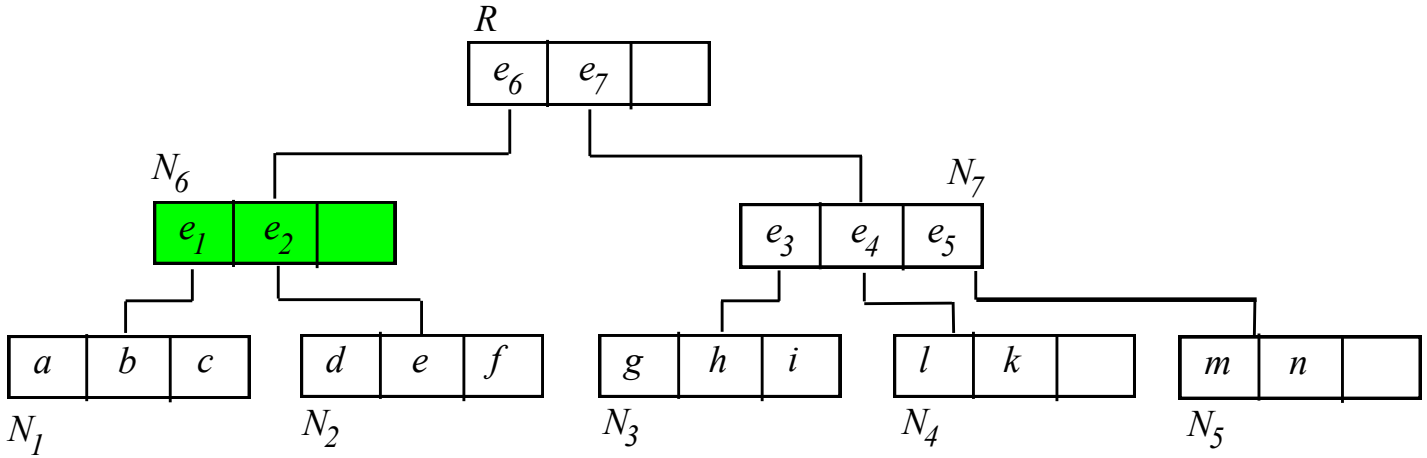
Aktion	Warteschlange	*S*
access root	$<e_7,4><e_6,6>$	∅
expand e_7	$<e_3,5><e_6,6><e_5,8><e_4,10>$	∅
expand e_3	**$<i,5>$**$<e_6,6><e_5,8><e_4,10>$	$\{i\}$



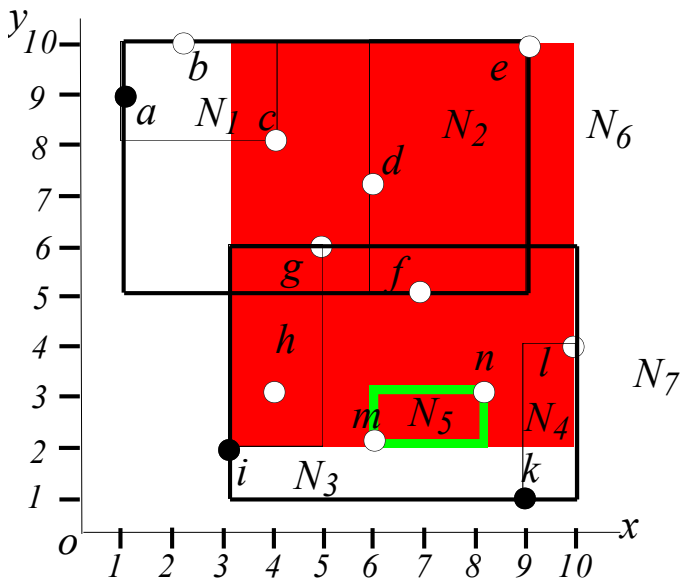
BBS Algorithmus – Beispiel



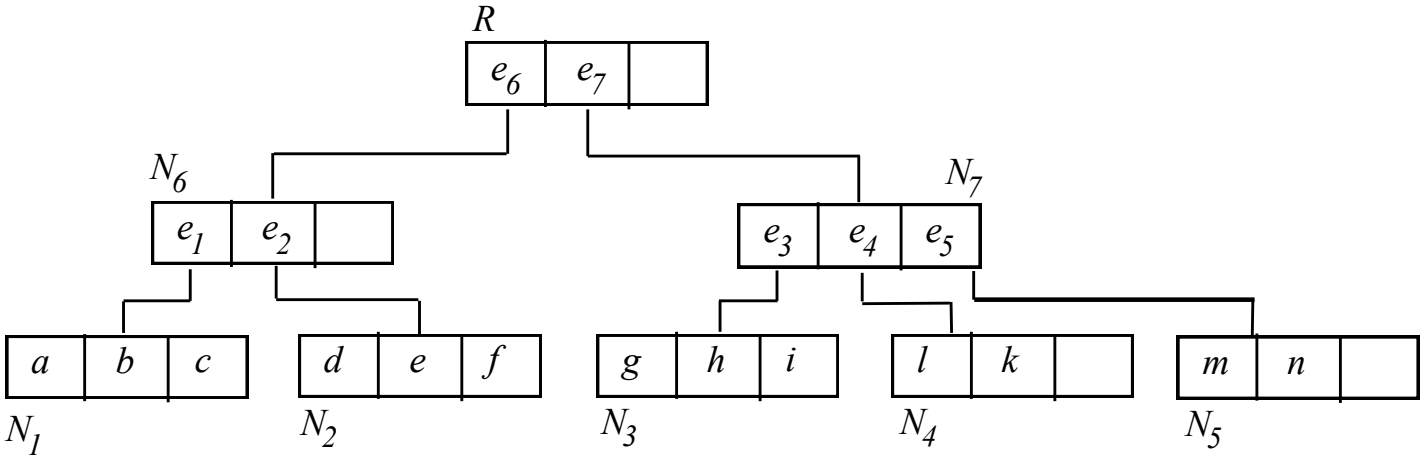
Aktion	Warteschlange	S
access root	$<e_7,4><e_6,6>$	∅
expand e_7	$<e_3,5><e_6,6><e_5,8><e_4,10>$	∅
expand e_3	$\mathbf{<i,5>}<e_6,6><e_5,8><e_4,10>$	$\{i\}$
expand e_6	$<e_5,8><e_1,9><e_4,10>$	$\{i\}$



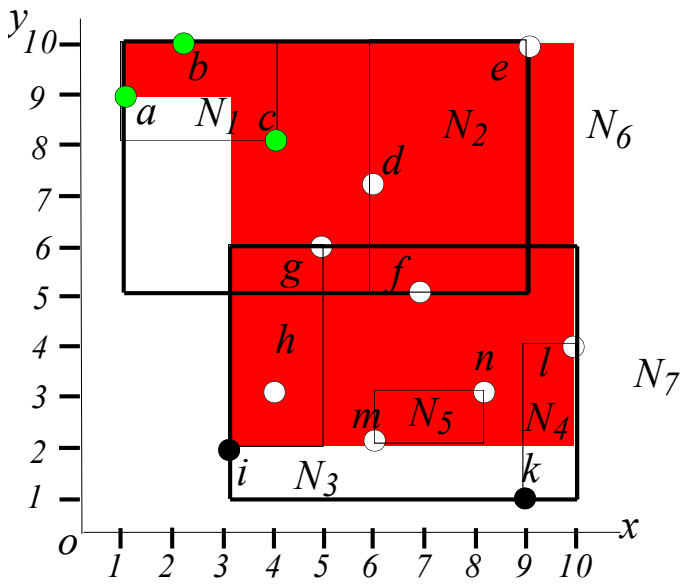
BBS Algorithmus – Beispiel



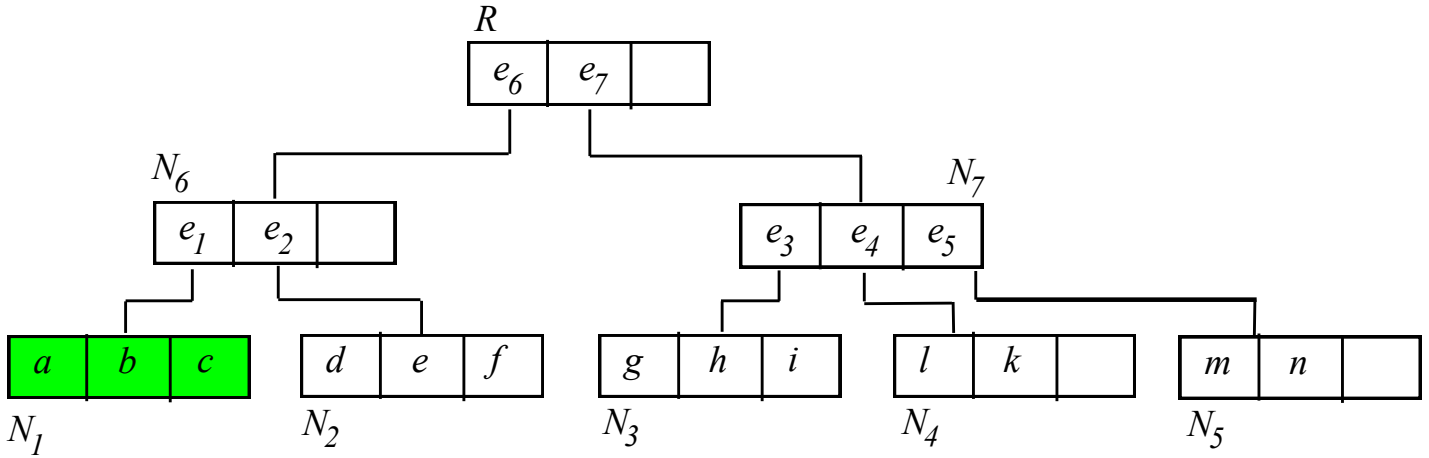
Aktion	Warteschlange	S
access root	$<e_7,4><e_6,6>$	∅
expand e_7	$<e_3,5><e_6,6><e_5,8><e_4,10>$	∅
expand e_3	**$<i,5>$**$<e_6,6><e_5,8><e_4,10>$	$\{i\}$
expand e_6	$<e_5,8><e_1,9><e_4,10>$	$\{i\}$
remove e_5	$<e_1,9><e_4,10>$	$\{i\}$



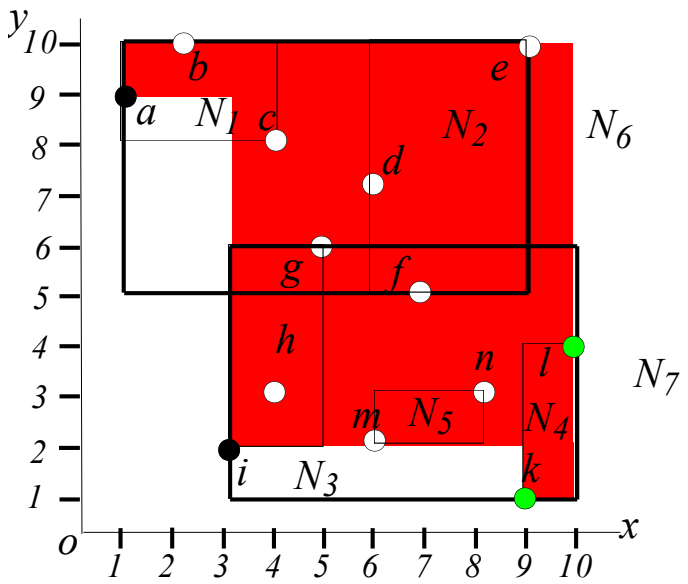
BBS Algorithmus – Beispiel



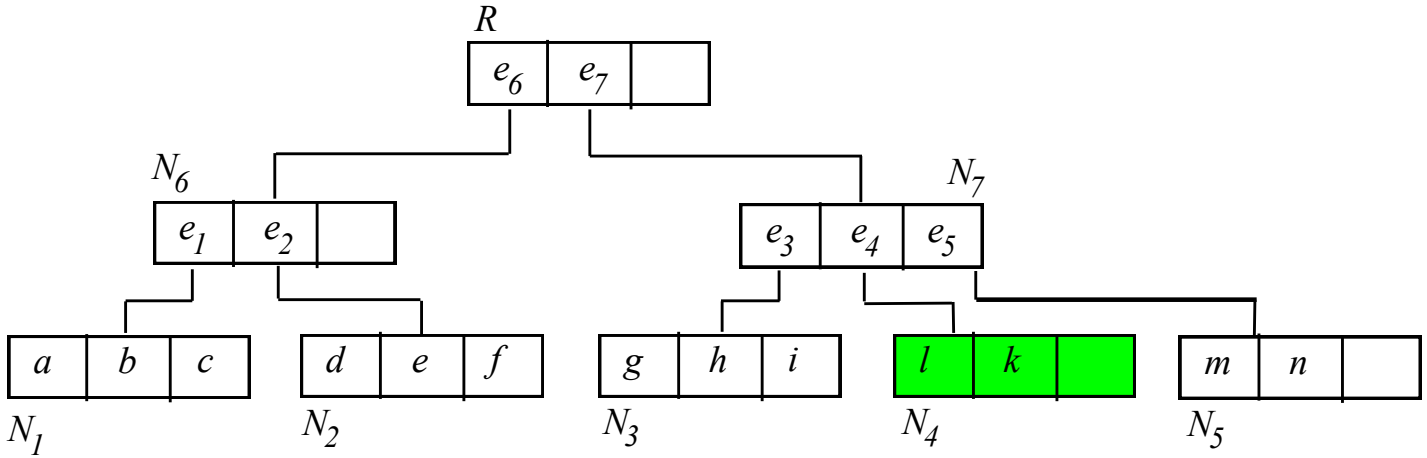
Aktion	Warteschlange	S
access root	$<e_7,4><e_6,6>$	∅
expand e_7	$<e_3,5><e_6,6><e_5,8><e_4,10>$	∅
expand e_3	**$<i,5>$**$<e_6,6><e_5,8><e_4,10>$	$\{i\}$
expand e_6	$<e_5,8><e_1,9><e_4,10>$	$\{i\}$
remove e_5	$<e_1,9><e_4,10>$	$\{i\}$
expand e_1	**$<a,10>$**$<e_4,10>$	$\{i,a\}$



BBS Algorithmus – Beispiel



Aktion	Warteschlange	*S*
access root	$<e_7,4><e_6,6>$	∅
expand e_7	$<e_3,5><e_6,6><e_5,8><e_4,10>$	∅
expand e_3	**<*i*,5>**$<e_6,6><e_5,8><e_4,10>$	{*i*}
expand e_6	$<e_5,8><e_1,9><e_4,10>$	{*i*}
remove e_5	$<e_1,9><e_4,10>$	{*i*}
expand e_1	**<*a*,10>**$<e_4,10>$	{*i*,*a*}
expand e_4	**<*k*,10>**	{*i*,*a*,*k*}



Branch-and-Bound-Skyline (BBS)

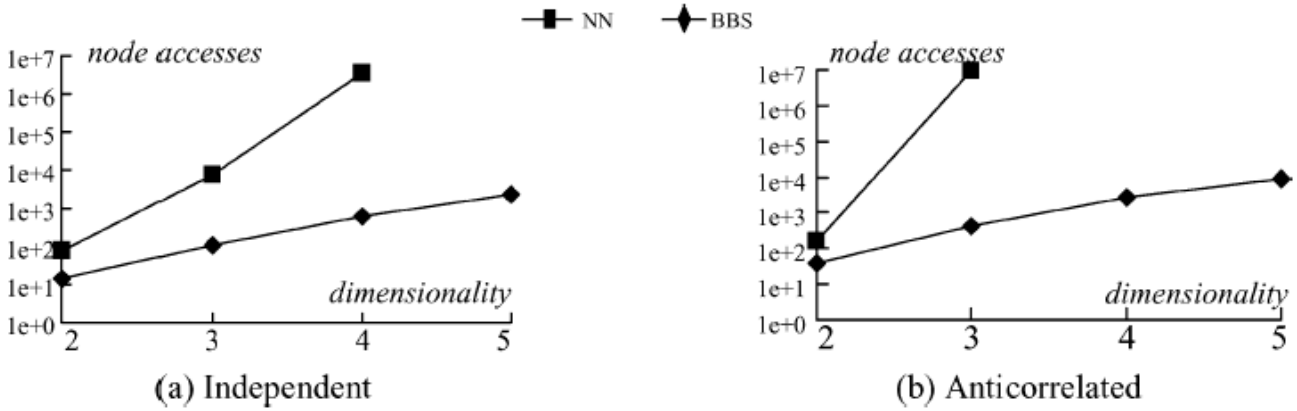
```
algorithm BBS (rt, type, f, test)
   // rt is an R-tree on the dataset, type denotes the type of priority queue
   // f is a point distance function, test is a tester for the data points
  let pq = build_priority_queue(type, map(lambda(o) (o, f(o)), root(rt))
      result = {}   // skyline
      object
   while not empty?(pq) do
      object := delete_next(pq)
      if test(object) then
         if always(lambda(o) not dominates?(o, object), result) then
            if point?(object) then
               result := filter(lambda(o) not dominates?(object, o), result) U {object}
            else // intermediate object
               for child in children(object) do
                  if always(lambda(o) not dominates?(o, child), result) then
                     insert((child, f(child)), pq)
   return result

function mindist(o)
   if point?(o) then
      return point_x(o) + point_y(o)
   else
      return point_x(leftbottom(mbr(o))) + point_y(leftbottom(mbr(o)))

BBS(data-rtree, 'min', mindist, lambda(x) true)
```

33.D. Papadias, Y. Tao, G. Fu und B. Seeger, „An Optimal and Progressive Algorithm for Skyline Queries," ACM SIGMOD, pp. 467-478, **2003**

BBS Algorithm - Vergleich

BBS besser als vorige Skyline-Algorithmen bzgl.

CPU-Zeit und I/O-Zeit



Number of R-tree node accesses vs dimensionality

Weiterentwicklungen:

Subspace-Skylines, verteilte Skylines, Approximationen

Wir haben erst die Spitze des Eisbergs betrachtet!

Dimitris Papadias, Yufei Tao, Kyriakos Mouratidis, and Chun Kit Hui.
Aggregate nearest neighbor queries in spatial databases,
ACM Trans. Database Syst. 30, 2, 529-576, **2005**.

Monotonie der Bewertungsfunktion

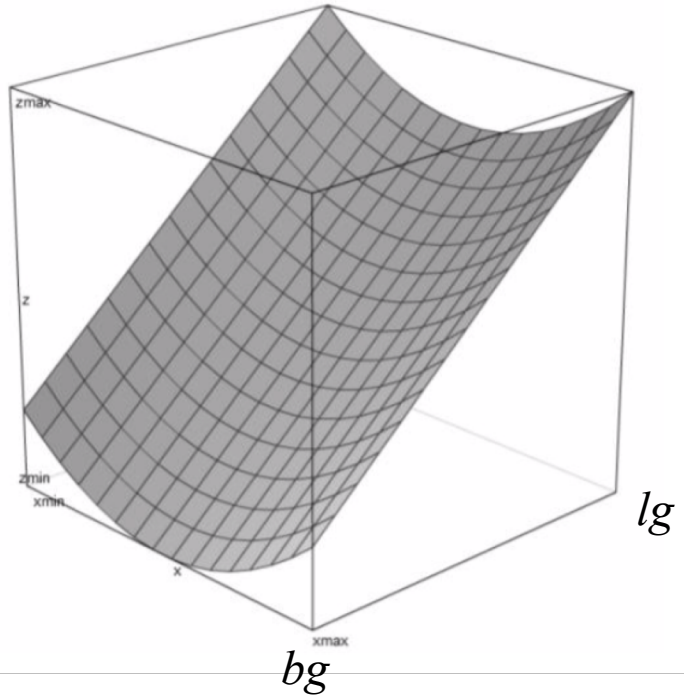
- BBS benötigt eine monotone Bewertungsfunktion
  - Monoton steigend oder monoton fallend
- Verwendet folgende Anfrage eine monotone Bewertungsfunktion?

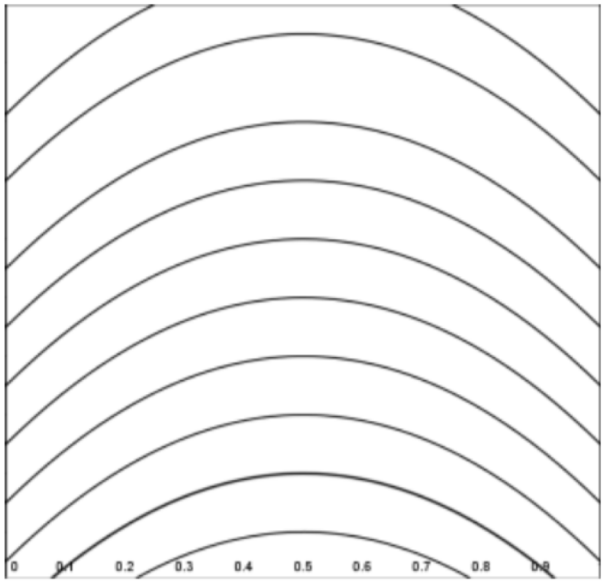
SELECT $Top\ 3\ *$
FROM $Immobilien$
WHERE PLZ BETWEEN 80331 AND 81929
ORDER BY $(BG - 0.5)^2 + LG$ DESC

- Bewertungsfunktion, die maximiert werden soll:

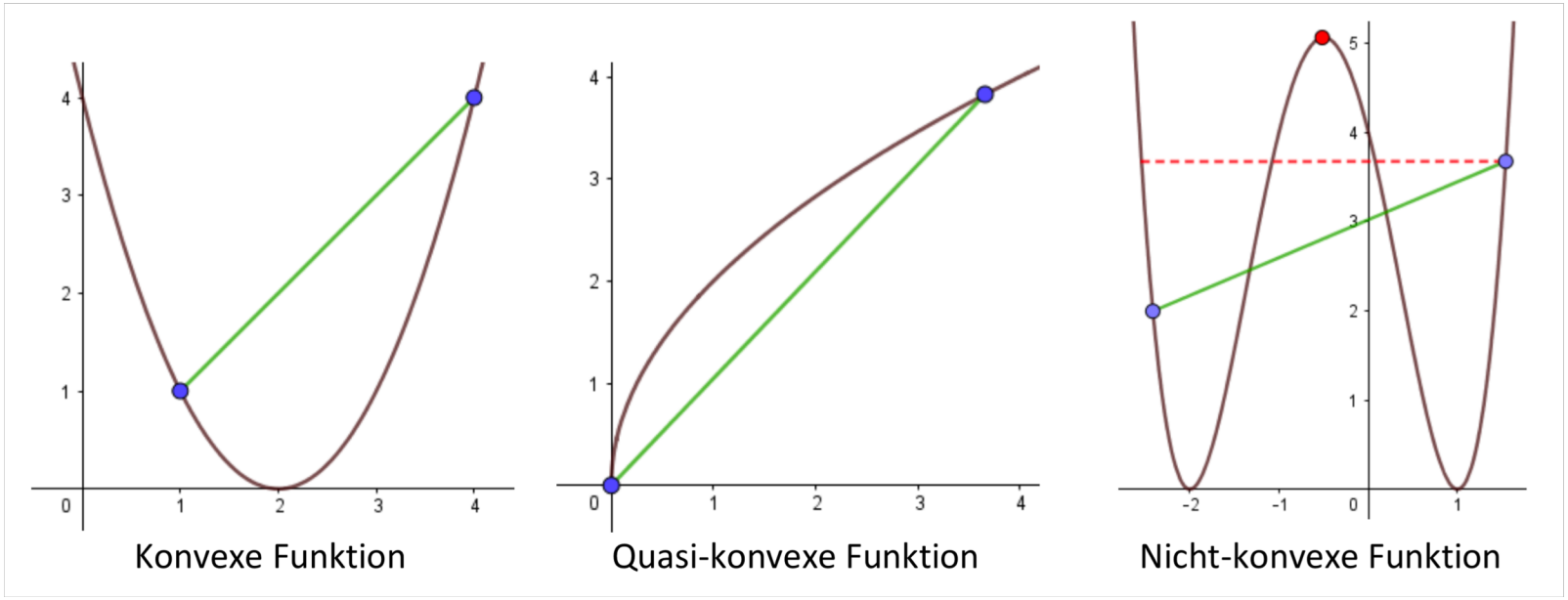
$$f(t) = (bg(t) - 0{,}5)^2 + \lg(t)$$

Graph und Höhenlinie von f

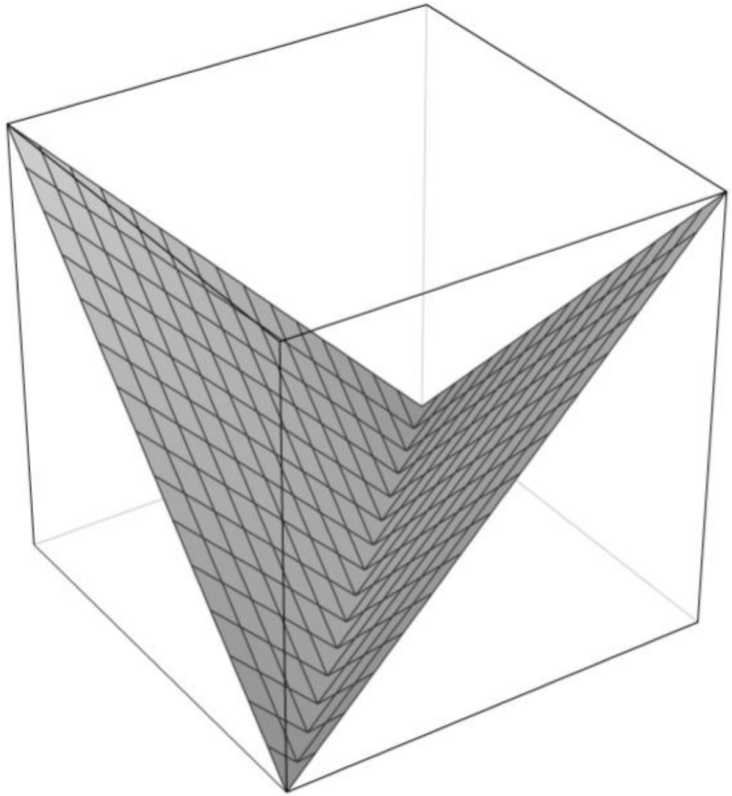


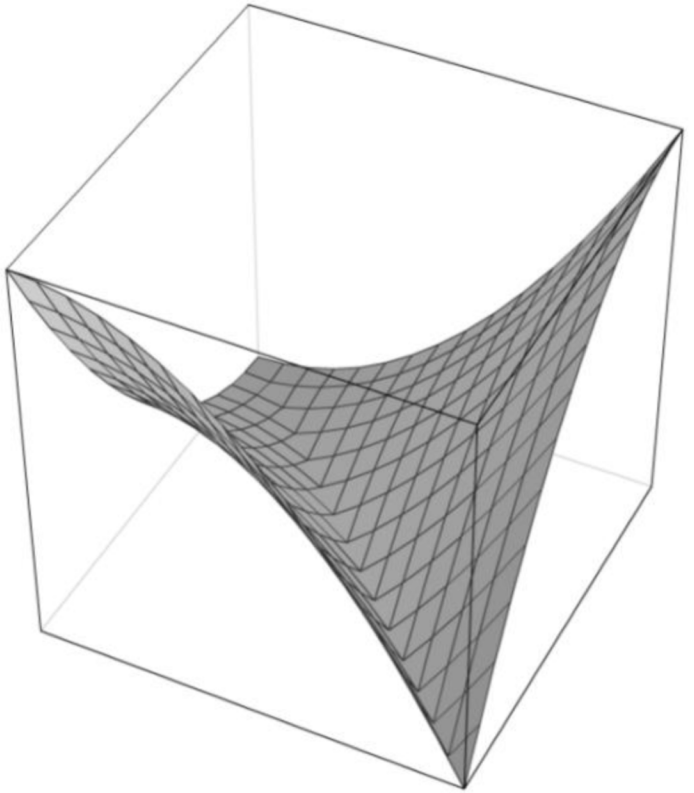


Konvexe und Quasi-Konvexe Funktionen



Globale und lokale Extrempunkte





Maximumprinzip Quasi-Konvexer Funktionen

Es sei $f: H \to \mathbb{R}$ eine quasi-konvexe Bewertungsfunktion, $H \subset \mathbb{R}^n$ ein beliebig gegebenes Hyper-rechteck (MBR) mit seinen 2^n Ecken $e_1, \ldots, e_{2^n}$, dann gilt:

$$f_{max}(H) = max\{f(e_1), \ldots, f(e_{2^n})\}.$$

Anders formuliert bedeutet diese Aussage, dass eine quasi-konvexe Funktion auf einem Hyper-rechteck ihr Maximum immer auf einer, oder auf mehreren Ecken annimmt.

```
function maxdistqc(o)
   if point?(o) then
      return point_d1(o) + … + point_d2n(o)
   else
      let p = max({f(e1(mbr(o))), … f(e2n(mbr(o)))})
        return return point_d1(p) + … + point_d2n(p)

BBS(data-rtree, 'max', maxdistqc, lambda(x) true)
```

Minimumprinzip?

- Leider für quasi-konvexe Funktionen nicht anwendbar
- Gibt es Funktionsklassen, für die wir ein Minimumprinzip finden?

SELECT $Top\ 3\ *$
FROM $Immobilien$
WHERE $Typ = 'Apartment'$ AND $Standort = 'Berlin'$
ORDER BY $(Wohnraum - 0{,}4)^2 + (Kaufpreis - 0{,}5)^2$ ASC

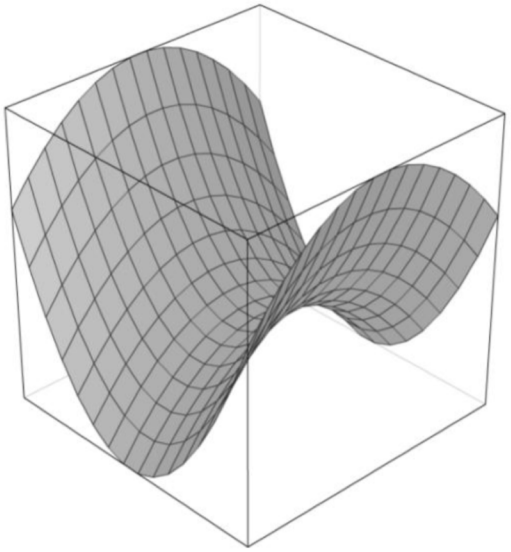
SELECT $Top\ k\ *$
FROM R
[WHERE…]
ORDER BY $\omega_1(a_1 - q_1)^2 + \omega_2(a_2 - q_2)^2$ ASC

Quadratische Funktionen

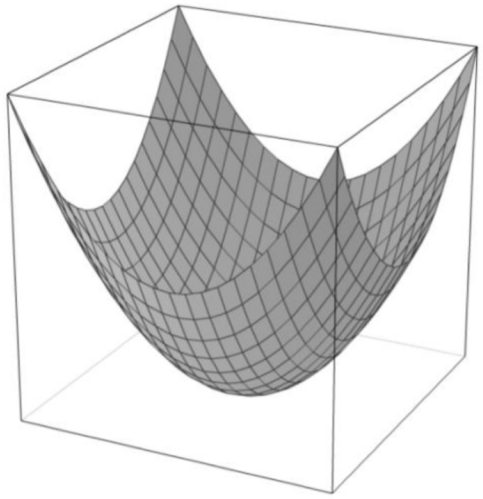
- Eine Abbildung $f: D \to \mathbb{R}, D \subseteq \mathbb{R}^n$ der Form

$$(x_1, \ldots, x_n) \mapsto \sum_{i=1}^{n} \omega_i \, (x_i - q_i)^2$$

mit $\omega_i \in \mathbb{R}$ für jedes $i \in \{1, \ldots, n\}$ heißt multivariate *quadratische* Funktion.

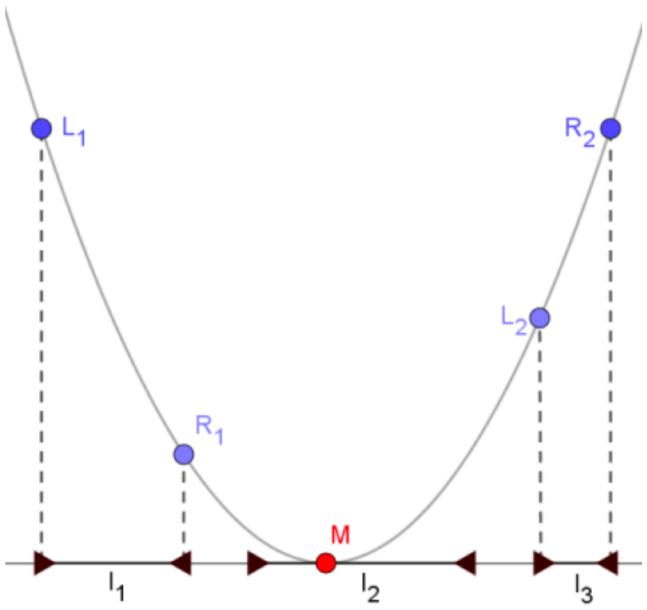


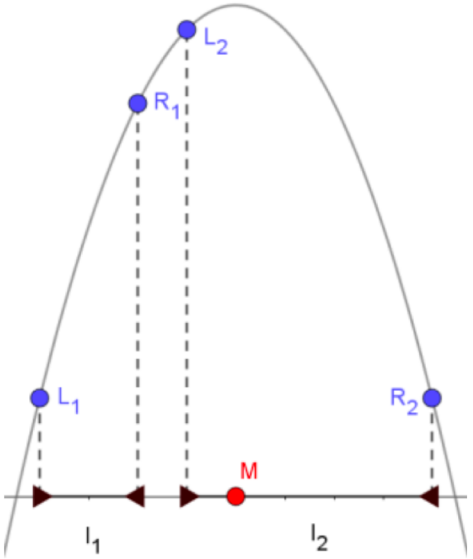
Hyperbolisches Paraboloid (ein $\omega_i < 0$)



Elliptisches Paraboloid (alle $\omega_i > 0$)

Eindimensionale Paraboloide

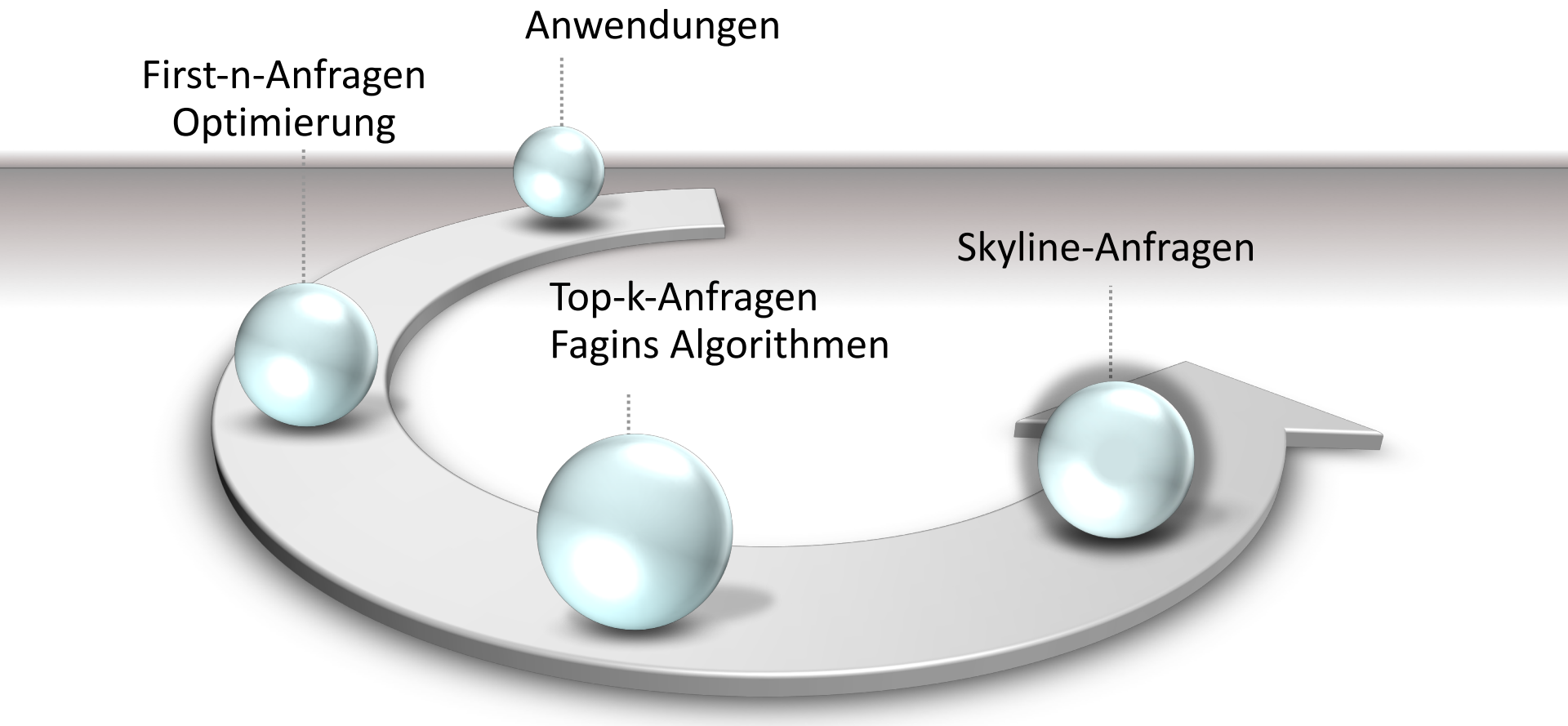




Minimumprinzip für quadratische Funktionen

Es sei $f: H \to \mathbb{R}^n$ quadratisch, H ein Hyperrechteck und $x_{min} \in H$ der Punkt auf dem f minimal auf H ist. Wenn $f(x_{min}) \neq 0$, so liegt x_{min} auf dem Rand von H. Ist $f(x_{min}) = 0$, dann liegt x_{min} im Inneren von H.

```
function mindistquadr(o)
   if point?(o) then
      return point_x(o) + point_y(o)
   else
    return
      reduce(+,
             map(lambda(dim) mindistquadr_dim(f, start_dim(o), end_dim(o), ω_dim,
q_dim),
                 1:d),
             0)

function mindistquadr_dim(f, a, b, ω, q)
   if q < 0 then
      return min({f(a), f(b)})
   else
      if q < a or q > b then
         return min({f(a), f(b)})
      else
         return 0

BBS(data_rtree, 'min', mindistquadr, lambda(x) true)
```

Non-Standard-Datenbanken

Von First-n- und Top-k-Anfragen zu Skyline-Anfrage