

---

# Non-Standard-Datenbanken

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

Prof. Dr. Ralf Möller

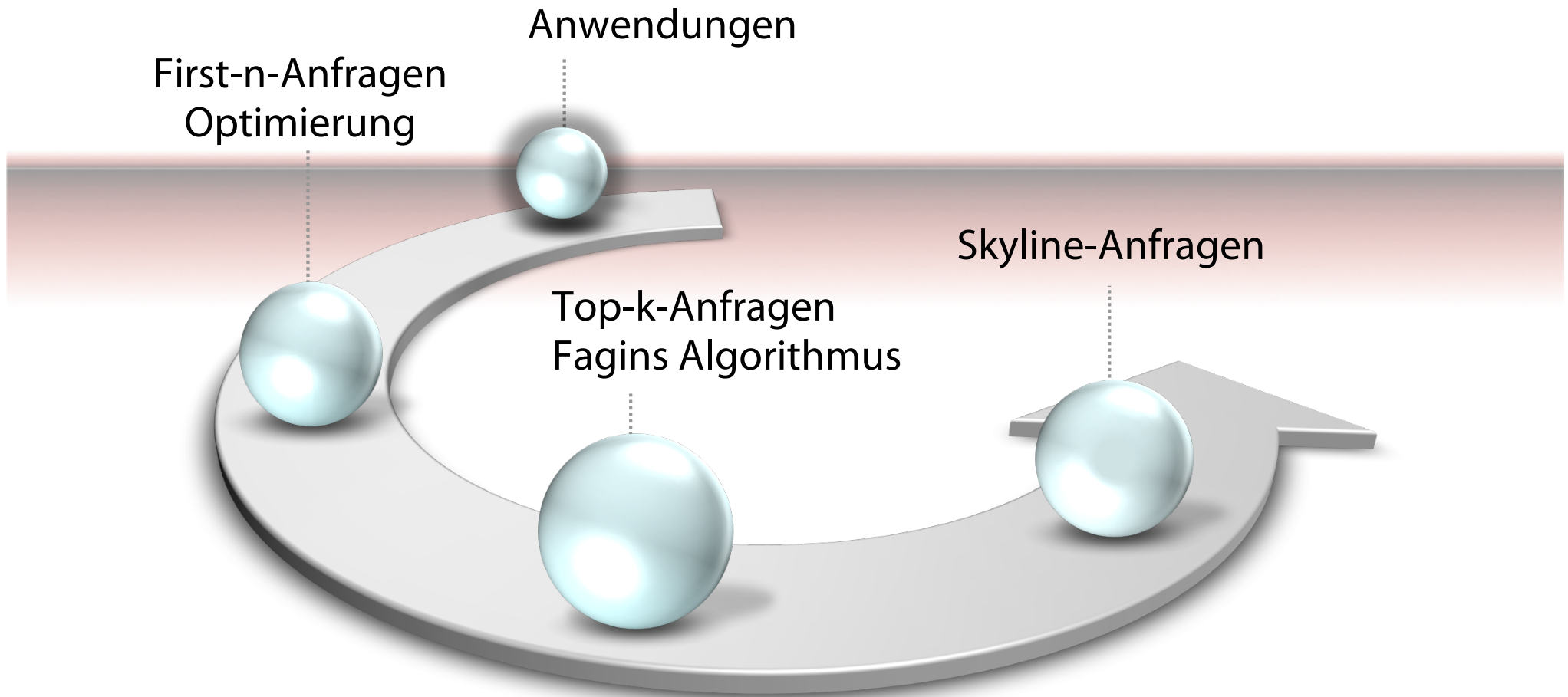
Universität zu Lübeck

Institut für Informationssysteme



# Non-Standard-Datenbanken

## First-n und Top-k-Anfragen



# Beispiel

---

```
SELECT h.name, h.adresse, h.tel  
FROM hotels h, flughäfen f  
WHERE f.name = 'LBC'  
ORDER BY distance(h.ort, f.ort)
```

20.000

1.000

1

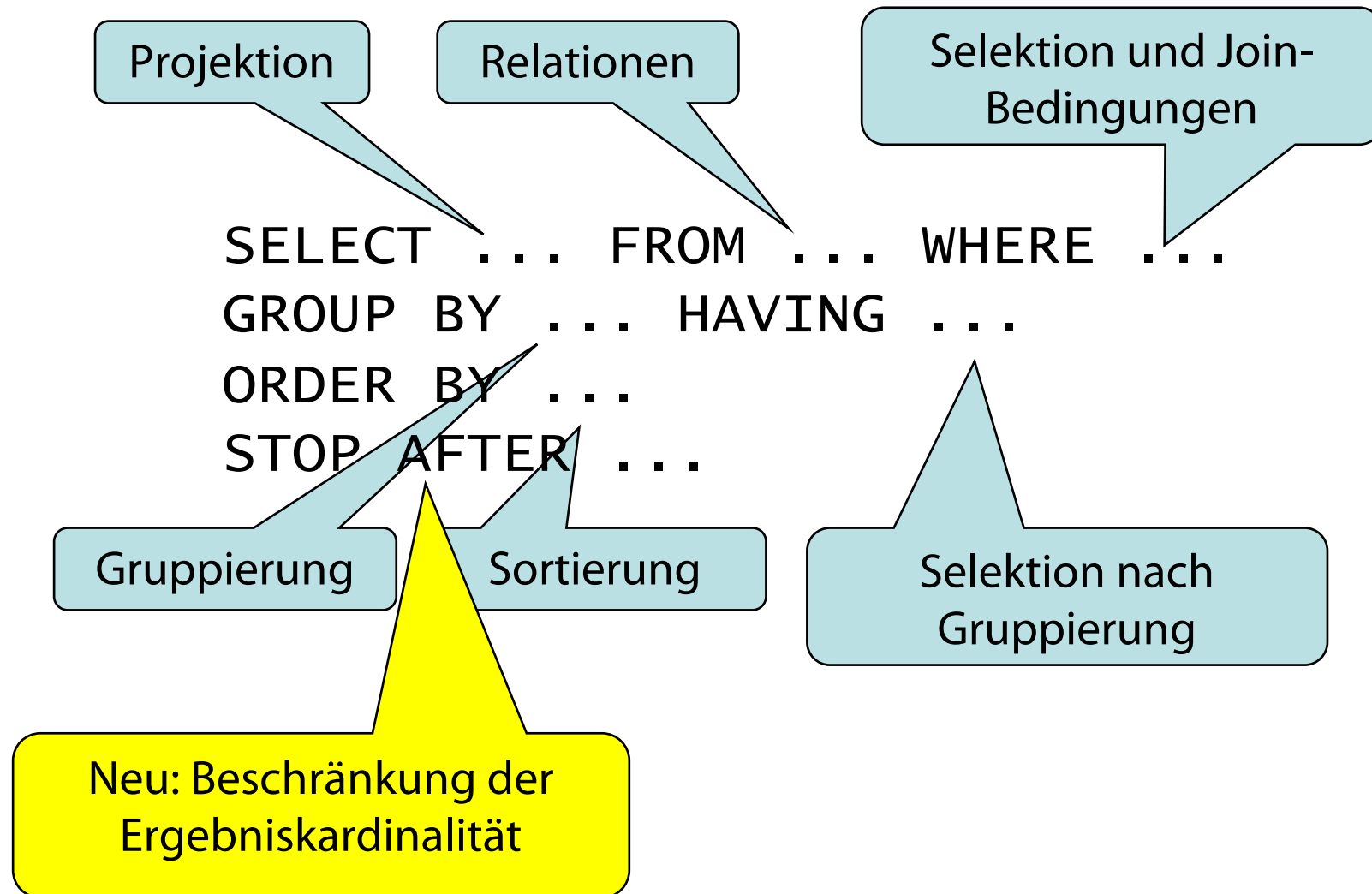
- Naive Auswertung: 20.000 Hotels mit aufsteigender Entfernung zu LBC
- Zur Sortierung: mind. 20.000 Mal distance() ausführen

# Problem

---

- Was machen wir, wenn nur 5 Hotels gezeigt werden sollen?
  - Auswahl im Anwendungsprogramm
    - Großer Aufwand
      - Eventuell werden sehr viele Daten kommuniziert
  - Auswahl durch Programmiersprache auf Seiten der DB angewendet auf Anfrageergebnis
    - Keine Optimierung
  - Gewünschte Anzahl in Anfrage spezifizieren
    - Optimierungsmöglichkeiten

# STOP AFTER – Syntax



# 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
```

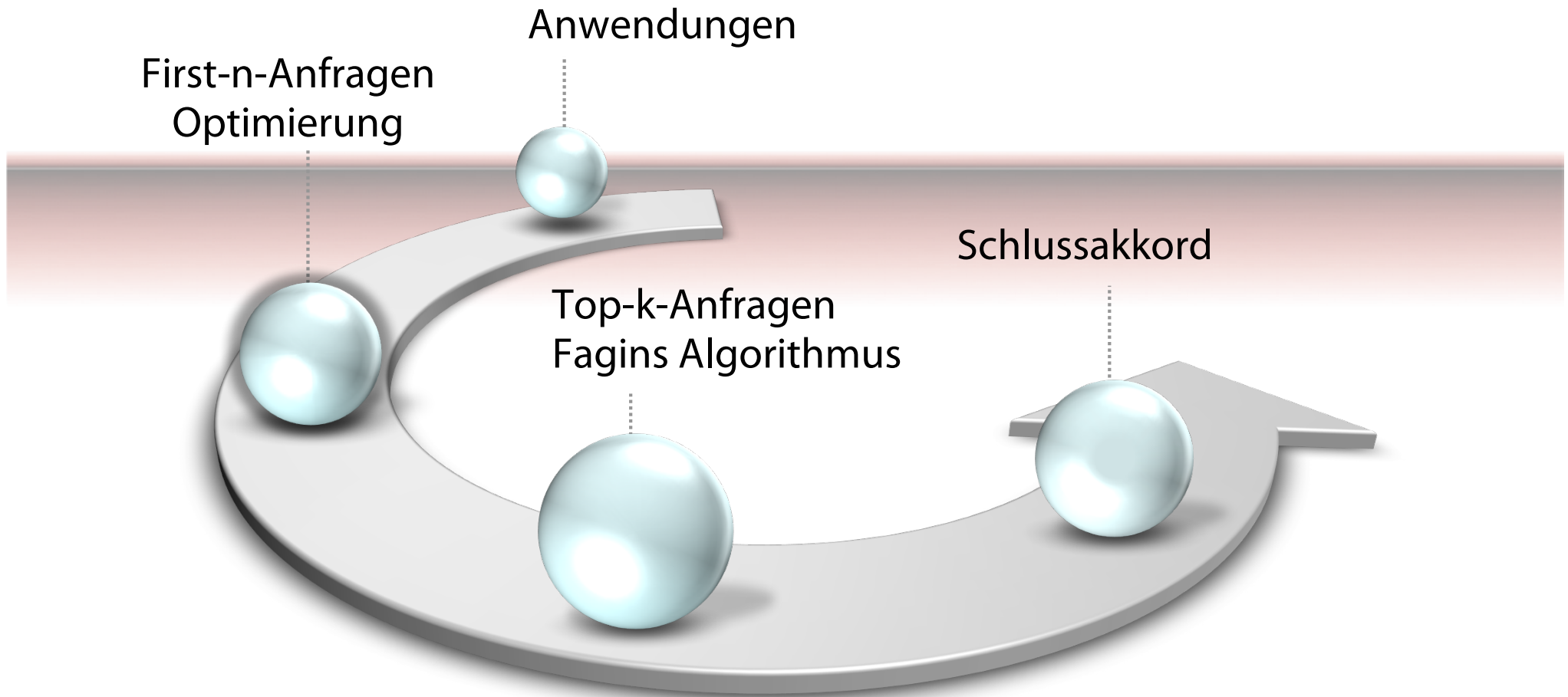
- Verwendung in Kombination mit ORDER BY möglich und sinnvoll
- Falls der Ausdruck **count** sich zu NULL evaluiert, wird ALL angenommen.
- Falls der Ausdruck **start** sich zu NULL evaluiert, wird 0 angenommen.

## SQL Server:

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

# Non-Standard-Datenbanken

## First-n und Top-k-Anfragen



# 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.

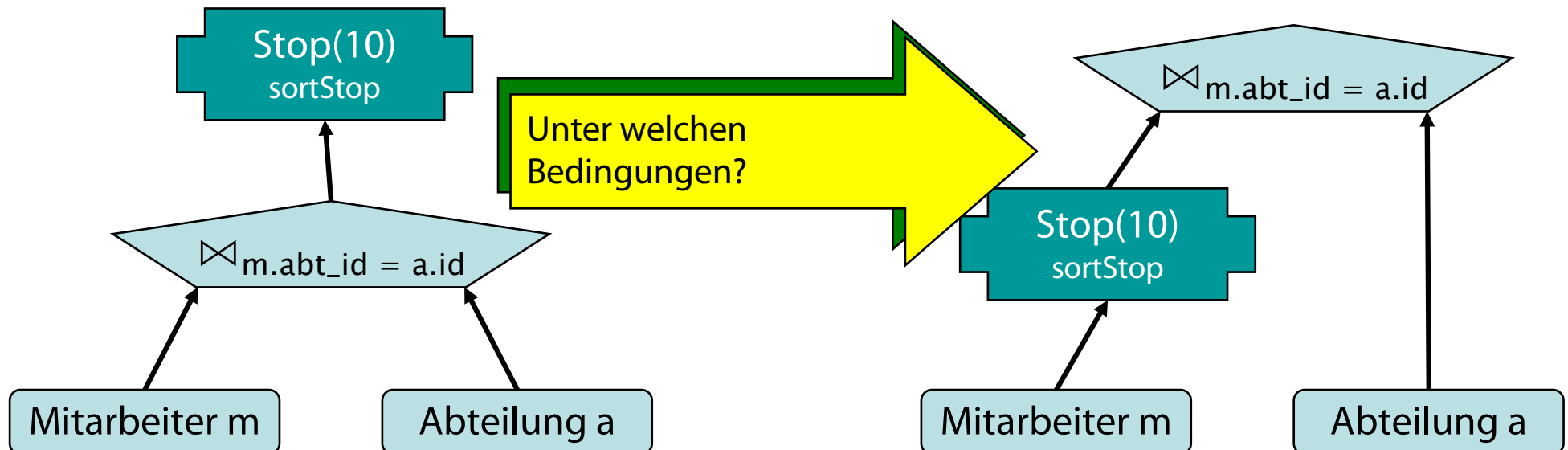
# STOP AFTER: Optimierung

```
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 Bedingungen ist eine Optimierung möglich?

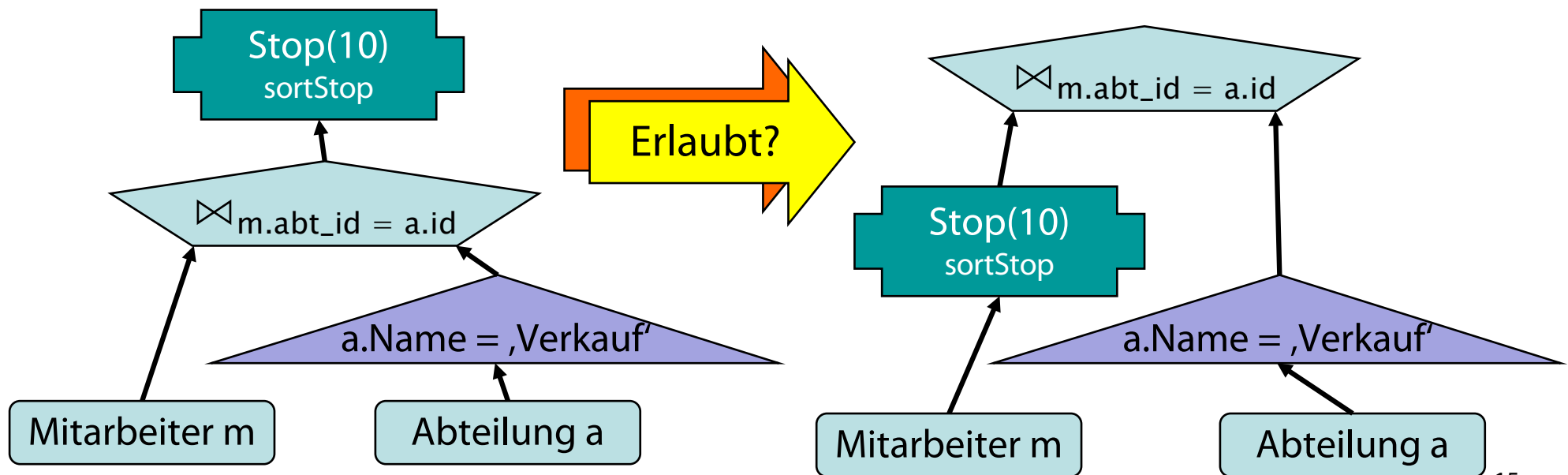
# Optimierung mit Stop-Operator

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



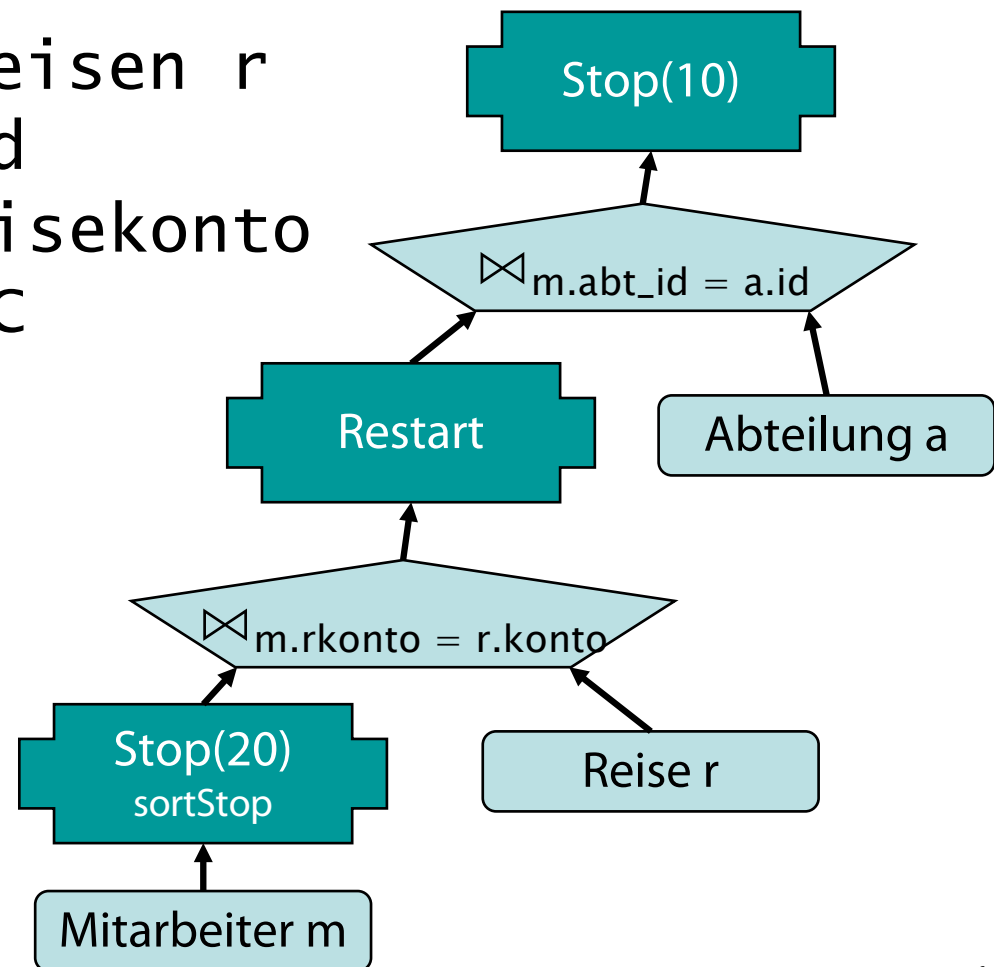
# Optimierung mit Stop-Operator

```
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)

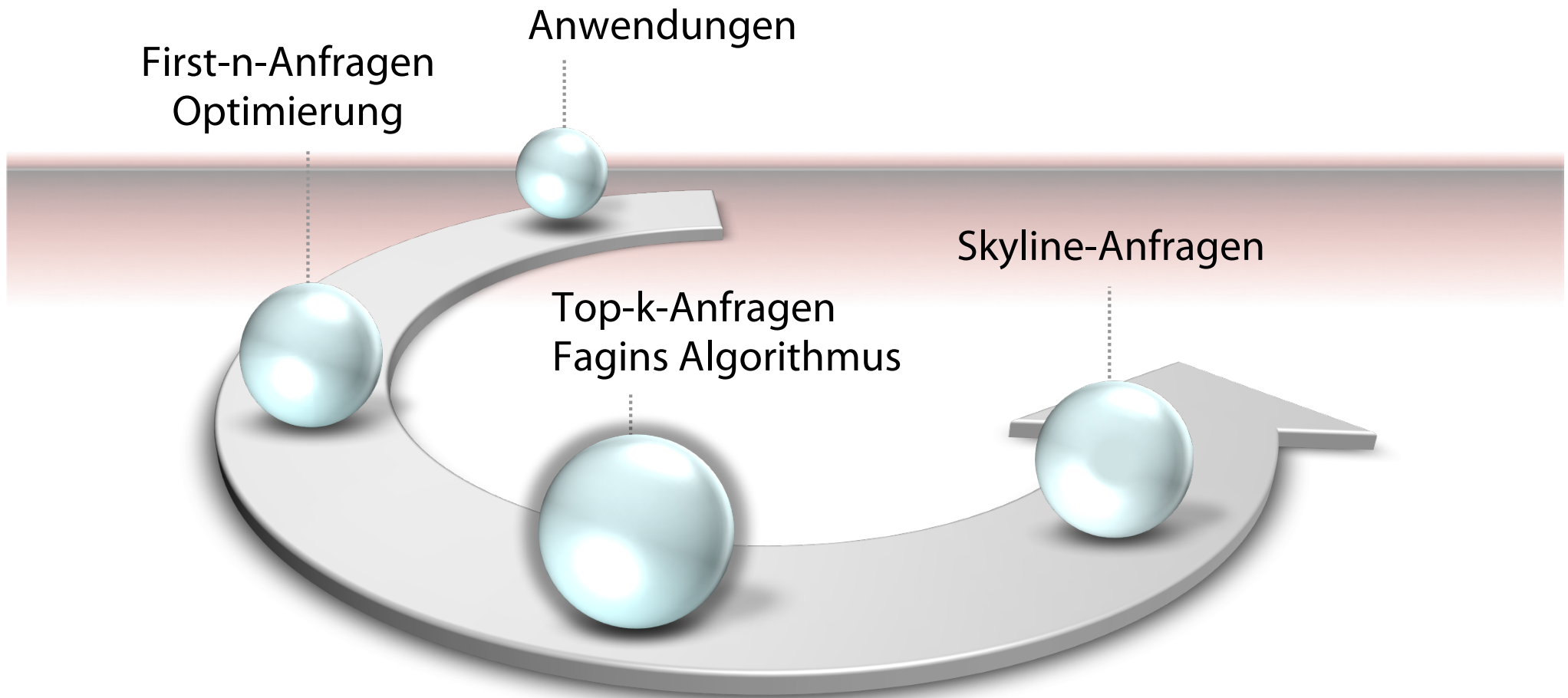
```
SELECT *  
FROM  mitarbeiter m,  
       abteilung a, reisen r  
WHERE m.abt_id = a.id  
AND   r.konto = m.reisekonto  
ORDER BY m.gehalt DESC  
STOP AFTER 10
```





# Non-Standard-Datenbanken

## First-n und Top-k-Anfragen



# Top-k Ergebnisse: Motivation

---

- First-n beschränkt Ergebnismenge bzgl. eines Attributs
  - Bei Verwendung mehrerer Attribute beim Sortieren (Sekundärschlüssel) bleibt erstes Attribut dominant
- Top-k beschränkt Ergebnismenge bzgl. mehrerer Attribute
  - Maße sind oft *unscharf*
  - Sortierung nach einem (komplexen) Maß
    - Verschiedene Dimensionen eventuell auf eine Einheit umgerechnet (lineare Kombination mit Gewichten)
    - Sortierung erst nach min über alle Attribute

# Top-k in Multimedia DBMS

- Beatles „Red Album“
- Anfrage:
  - Farbe = ‚rot‘  $\wedge$  Name = ‚Beatles‘

Als Standard-Prädikat  
Antwort ist (unsortierte) Menge

Als unscharfes Prädikat:  
Antwort ist sortierte Liste



# Top-k: Benotete Mengen

---

- Benotete Menge:
  - Menge aus Paaren  $(x,g)$
  - $x$  ist ein Objekt
  - $g \in [0,1]$  ist eine Note (*grade*)
- Anfrage: Name = ‚Beatles‘
  - Antwort: benotete Menge mit  $g \in \{0,1\}$
- Anfrage: Farbe = ‚rot‘
  - Antwort: benotete Menge mit  $g \in [0,1]$

# Top-k: Benotete Mengen

---

- Anfrage:
  - Name = ‚Beatles‘  $\wedge$  Farbe = ‚rot‘
  - Name = ‚Beatles‘  $\vee$  Farbe = ‚rot‘
- Problem:
  - Maß: Benotung der Objekte in Antwort
- Sei  $g_A(x)$  die Note von Objekt  $x$  unter Anfrage  $A$ .
- Erwünschte Eigenschaften
  - Falls  $g \in \{0,1\}$  sollte Standard-Logik gelten.
  - Bewahrung der logischen Äquivalenz
    - $g_{A \wedge A}(x) = g_A(x)$
    - $g_{A \wedge (B \vee C)}(x) = g_{(A \wedge B) \vee (A \wedge C)}(x)$
  - Monotonie:  $g_A(x) \leq g_A(y), g_B(x) \leq g_B(y) \Rightarrow g_{A \wedge B}(x) \leq g_{A \wedge B}(y)$

# Top-k: Benotete Mengen

---

- Vorschlag
  - Konjunktionsregel:
    - $g_{A \wedge B}(x) = \min\{g_A(x), g_B(x)\}$
  - Disjunktionsregel:
    - $g_{A \vee B}(x) = \max\{g_A(x), g_B(x)\}$

# Top-k: Benotete Mengen

---

- $g_{A \wedge B}(x) = \min\{g_A(x), g_B(x)\}$ ,  $g_{A \vee B}(x) = \max\{g_A(x), g_B(x)\}$
- Standardlogik ( $g \in \{0,1\}$ )
  - $0 \wedge 1 = \min\{0,1\} = 0$
  - $0 \vee 1 = \max\{0,1\} = 1$
- Äquivalenz
  - $g_{A \wedge A}(x) = \min\{g_A(x), g_A(x)\} = g_A(x)$
  - $g_{A \wedge (B \vee C)}(x) = \min\{g_A(x), \max\{g_B(x), g_C(x)\}\} = \max\{\min\{g_A(x), g_B(x)\}, \min\{g_A(x), g_C(x)\}\} = g_{(A \wedge B) \vee (A \wedge C)}(x)$
- Monotonie
  - $g_A(x) \leq g_A(y), g_B(x) \leq g_B(y) \Rightarrow g_{A \wedge B}(x) \leq g_{A \wedge B}(y)$
  - $g_A(x) \leq g_A(y), g_B(x) \leq g_B(y) \Rightarrow \min\{g_A(x), g_B(x)\} \leq \min\{g_A(y), g_B(y)\}$

# Anfragebeantwortung

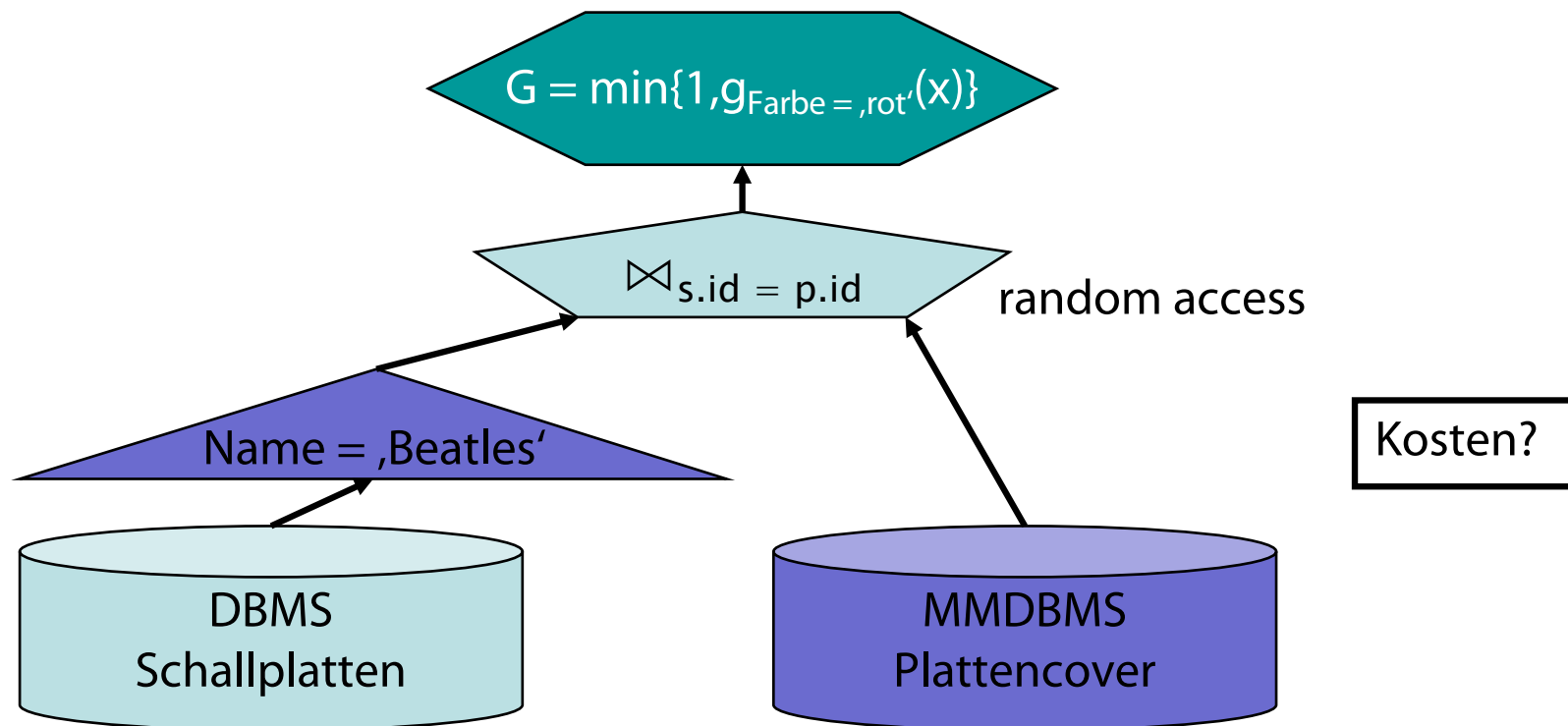
---

- Gegeben: Konjunktive Anfrage mit teilweise unscharfen Prädikaten.
- Gesucht: Benotete Menge, so dass  $k$  beste Elemente identifiziert werden können
- Zugriffsmodell auf MMDBMS
  - *Sorted access*: Cursor auf sortierte Liste
  - *Random access*: Note eines bestimmten Objekts
- Kostenmodell:
  - Jedes angefragte Objekt kostet 1
- Optimierung:
  - Minimiere Kosten



# Top-k: Beispiel

- Anfrage:
  - Name = ‚Beatles‘  $\wedge$  Farbe = ‚rot‘



# Top-k: Fagins Algorithmus

---

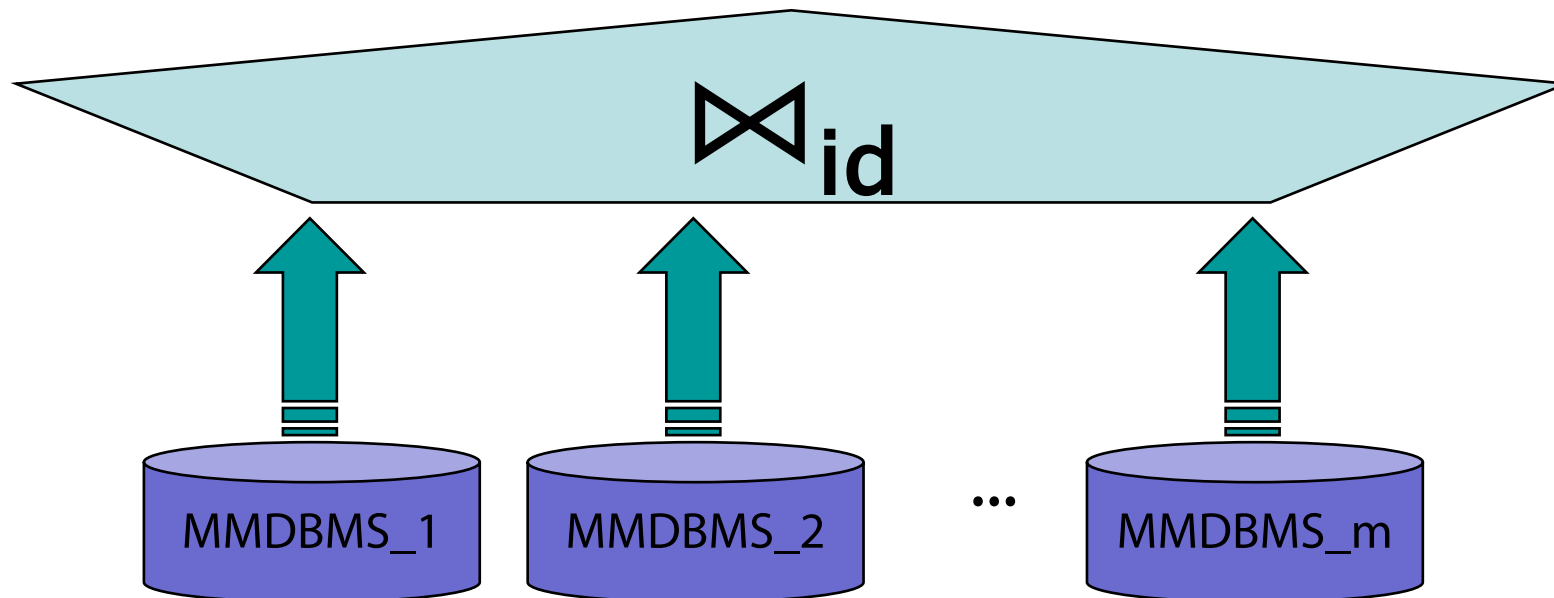
- Allgemeineres Problem:
  - Anfrage statt  $A \wedge B$  nun  $A_1 \wedge A_2 \wedge \dots \wedge A_m$
  - Für jedes Atom eine Quelle.
    - bzw. Zugriffsmöglichkeit durch sorted und random access
- Phase 1: Sorted access
- Phase 2: Random access
- Phase 3: Berechnung und Sortierung

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

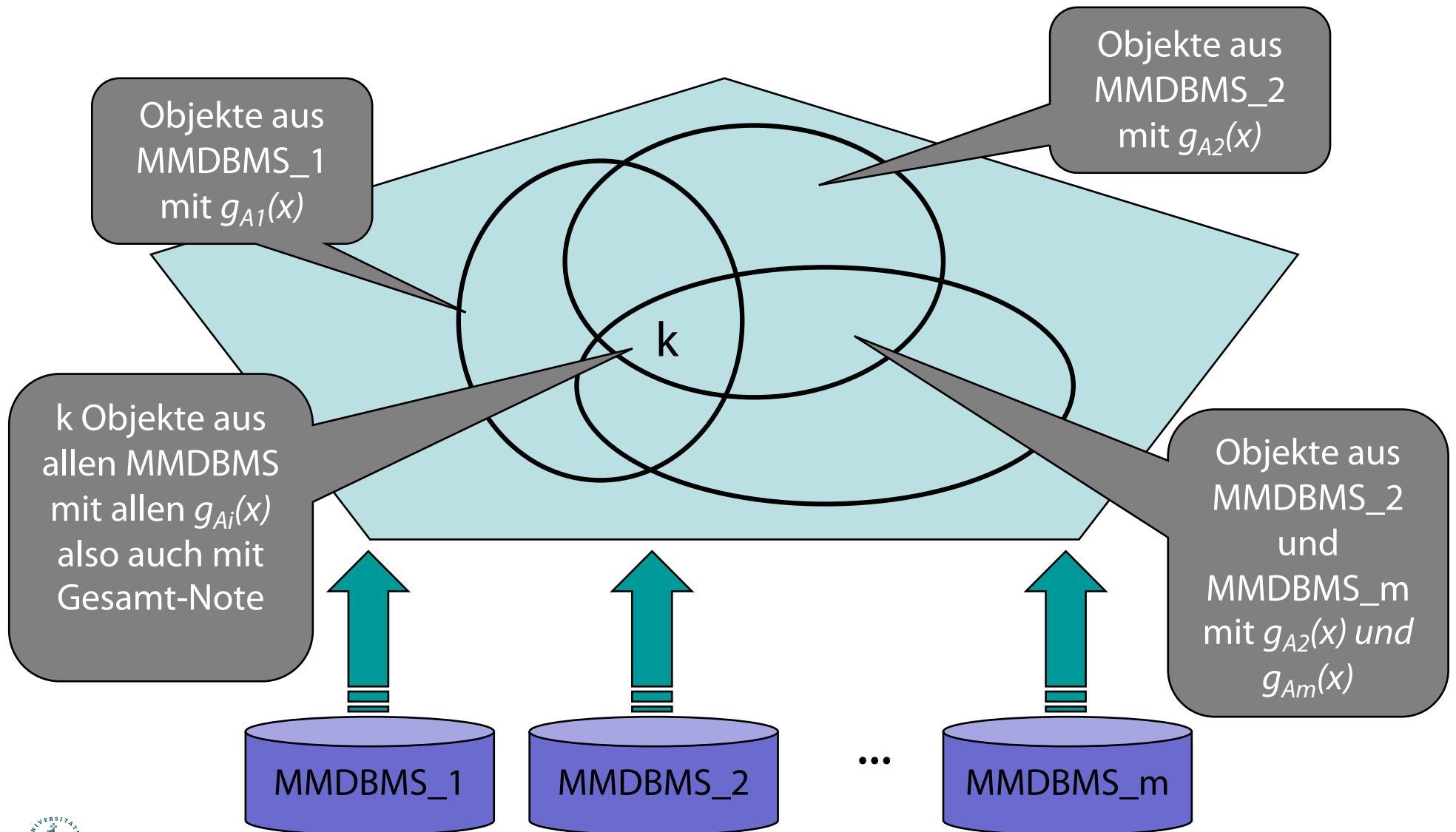
Ronald Fagin: Fuzzy Queries in Multimedia Database Systems. Proc. PODS-98, 1-10, **1998**

# Top-k: Fagins Algorithmus

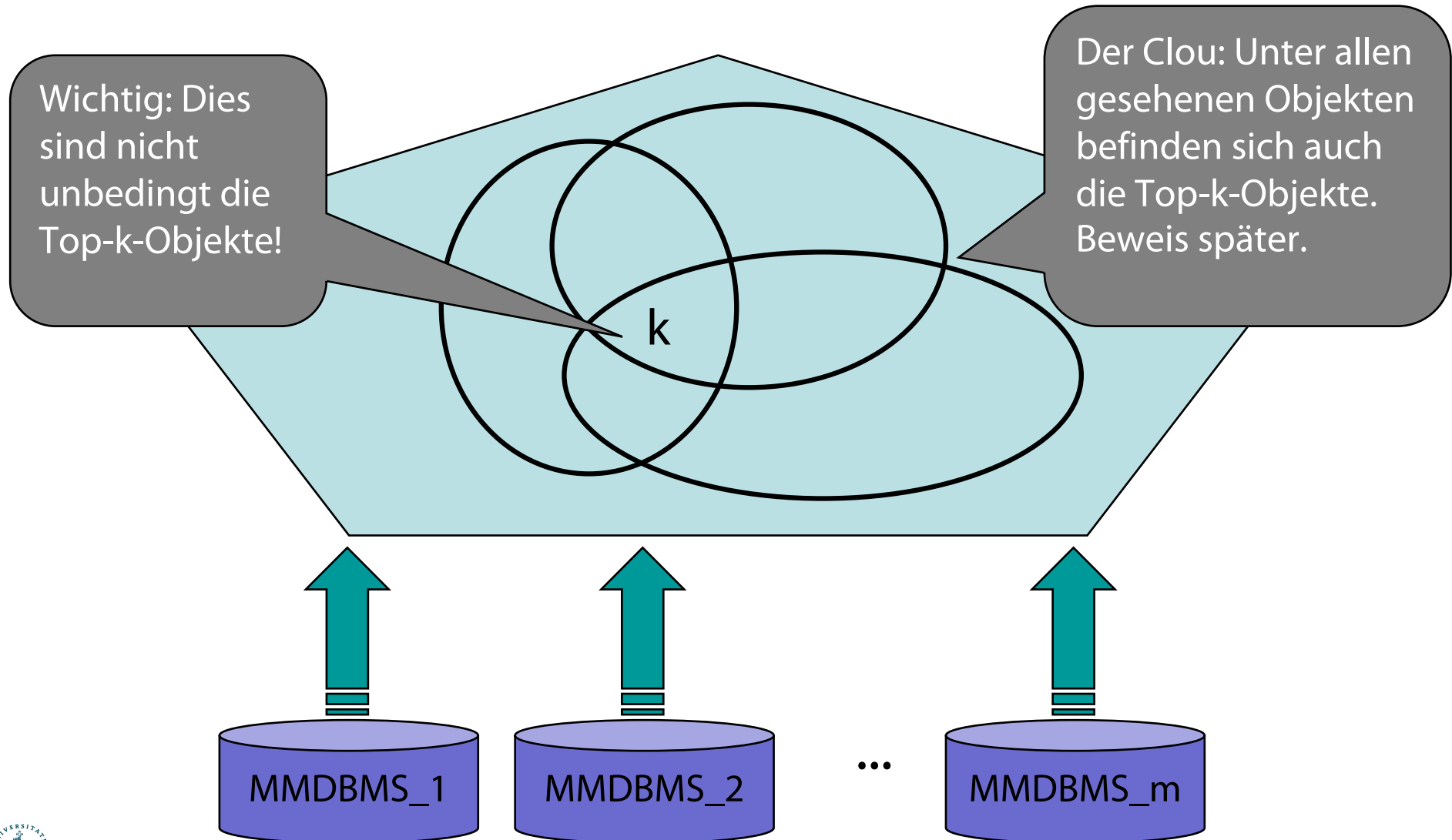
- $A_1 \wedge A_2 \wedge \dots \wedge A_m$
- Phase 1: Sorted access
  - Für jedes  $i$ : Schicke  $A_i$  an Quelle  $i$
  - Schreite sukzessive voran, bis Join über alle Teilergebnisse die Größe  $k$  hat.



# Top-k: Fagins Algorithmus

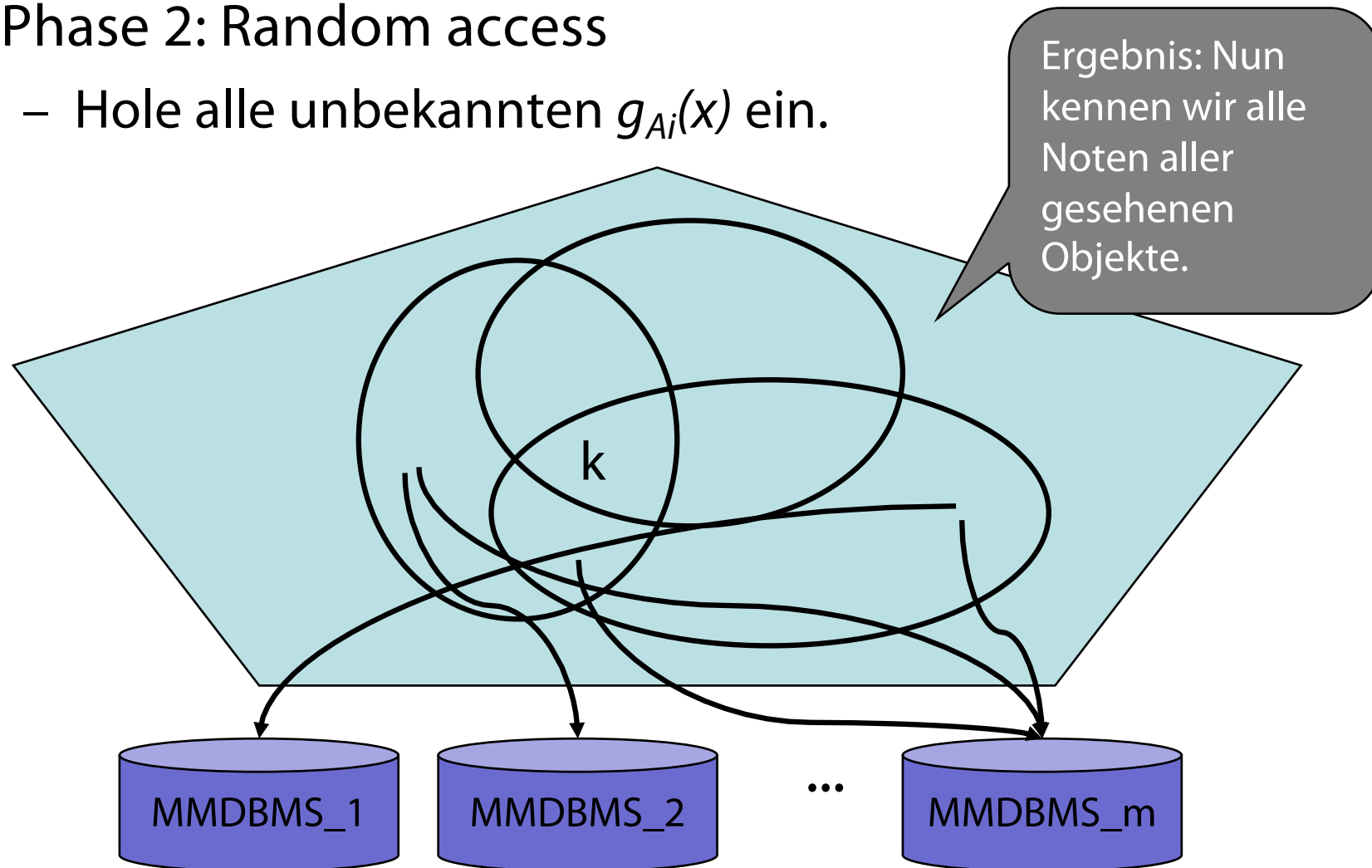


# Top-k: Fagins Algorithmus



# Top-k: Fagins Algorithmus

- Phase 2: Random access
  - Hole alle unbekanntes  $g_{A_i}(x)$  ein.



# Top-k: Fagins Algorithmus

---

- Phase 3: Berechnung und Sortierung

- Berechne für jedes Objekt

$$g_{A1 \wedge A2 \wedge \dots \wedge Am}(x) \\ = \min\{g_{A1}(x), g_{A2}(x), \dots, g_{Am}(x)\}$$

- Sortiere alle Objekte nach  $g_{A1 \wedge A2 \wedge \dots \wedge Am}(x)$
- Selektiere die höchsten k Objekte.
- Ausgabe dieser Top-k Objekte.

# Fagins Algorithmus – Beispiel

- Anfrage:
  - Form = ‚rund‘  $\wedge$  Farbe = ‚rot‘  $\wedge$  Stil = ‚Modern‘
  - k = 2

MMDBMS\_1

ID	Form	Rundheit
1	oval	0.8
2	achteck	0.6
3	viereck	0.15
4	dreieck	0.1
5	strich	0

MMDBMS\_2

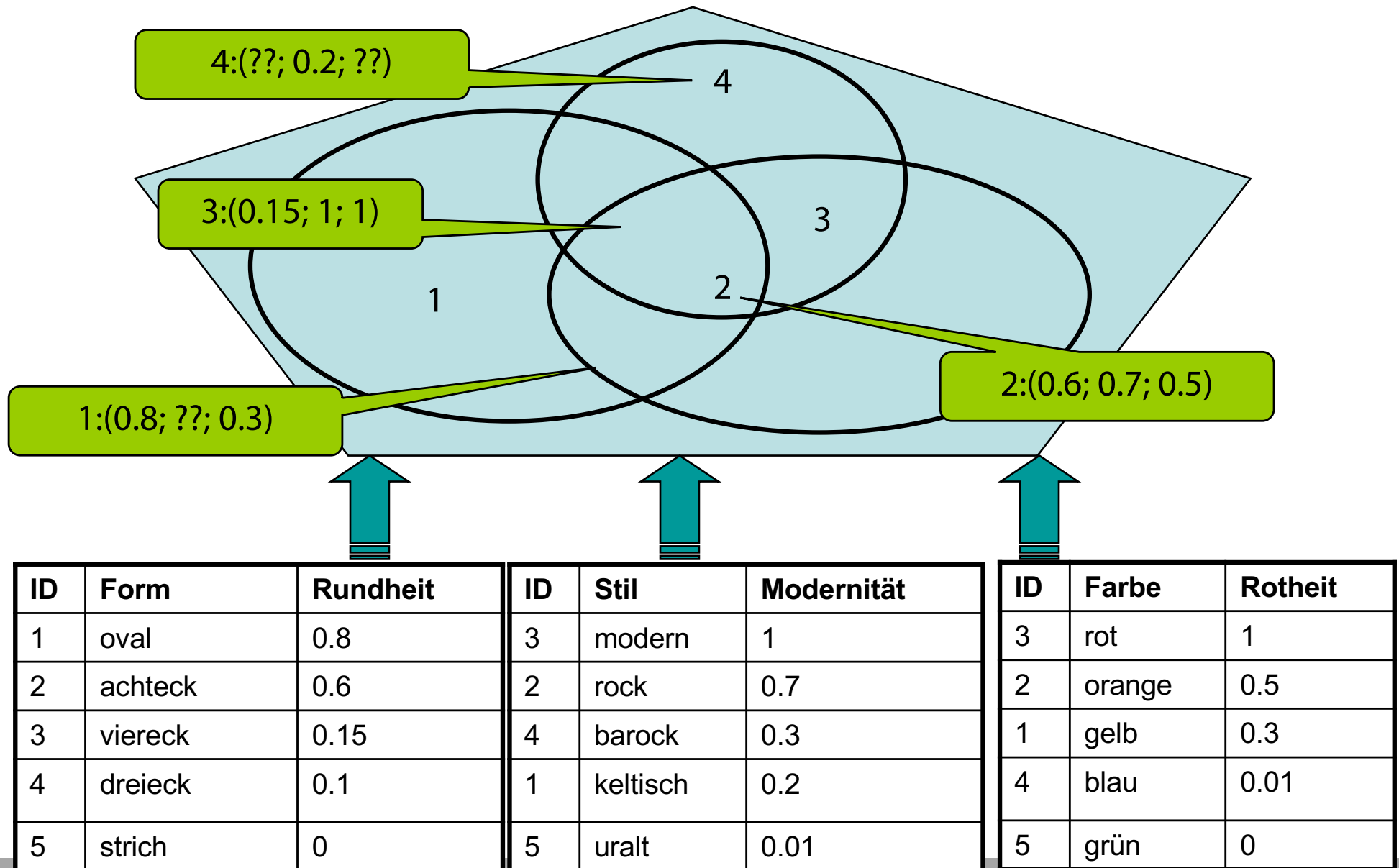
ID	Stil	Modernität
3	modern	1
2	rock	0.7
4	barock	0.2
1	keltisch	0.1
5	uralt	0.01

MMDBMS\_3

ID	Farbe	Rotheit
3	rot	1
2	orange	0.5
1	gelb	0.3
4	blau	0.01
5	grün	0



# Fagins Algorithmus – Beispiel



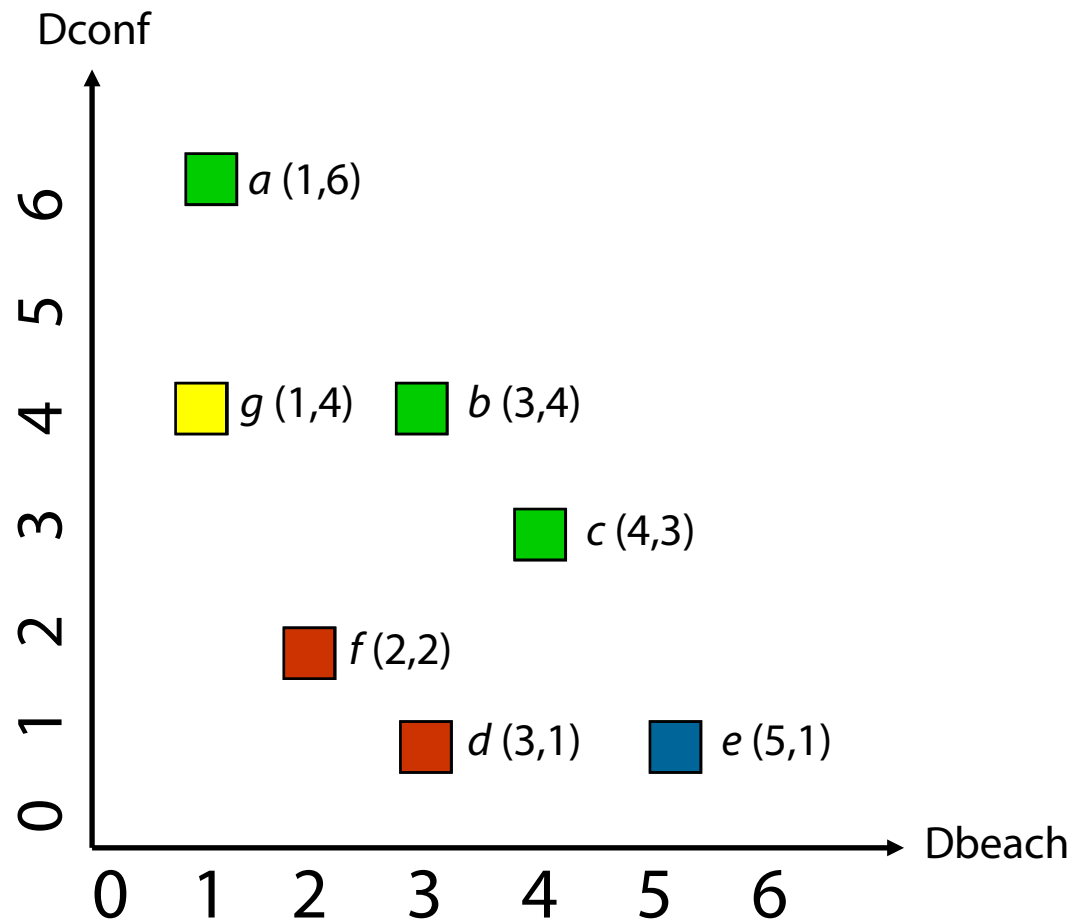
# Wiederholung: Top-k-Anfragen

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

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

# Ziel: $D_{\text{beach}}(x) + D_{\text{conf}}(x)$ minimieren

Bewertung basierend auf  $D_{\text{beach}}(x) + D_{\text{conf}}(x)$



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

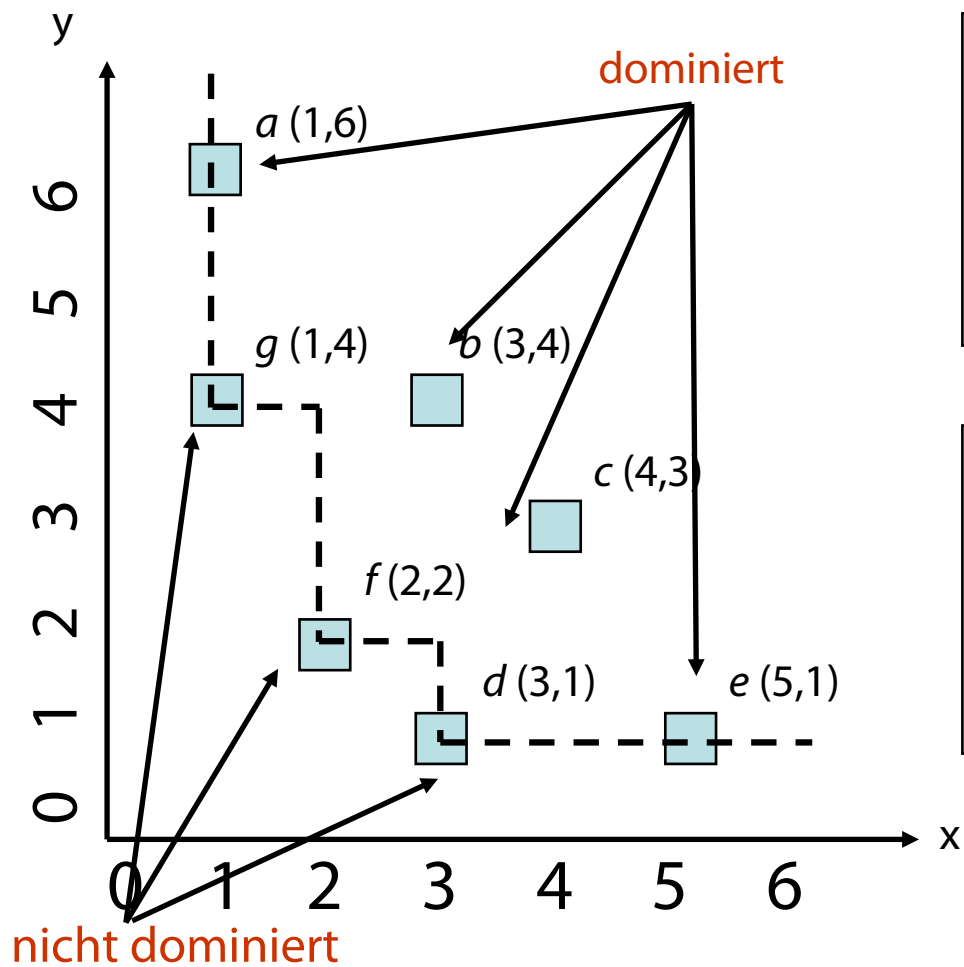
Beste Hotels: *d, f*

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

# Agenda

---

- Top-k-Anfragen
  - Wiederholung 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

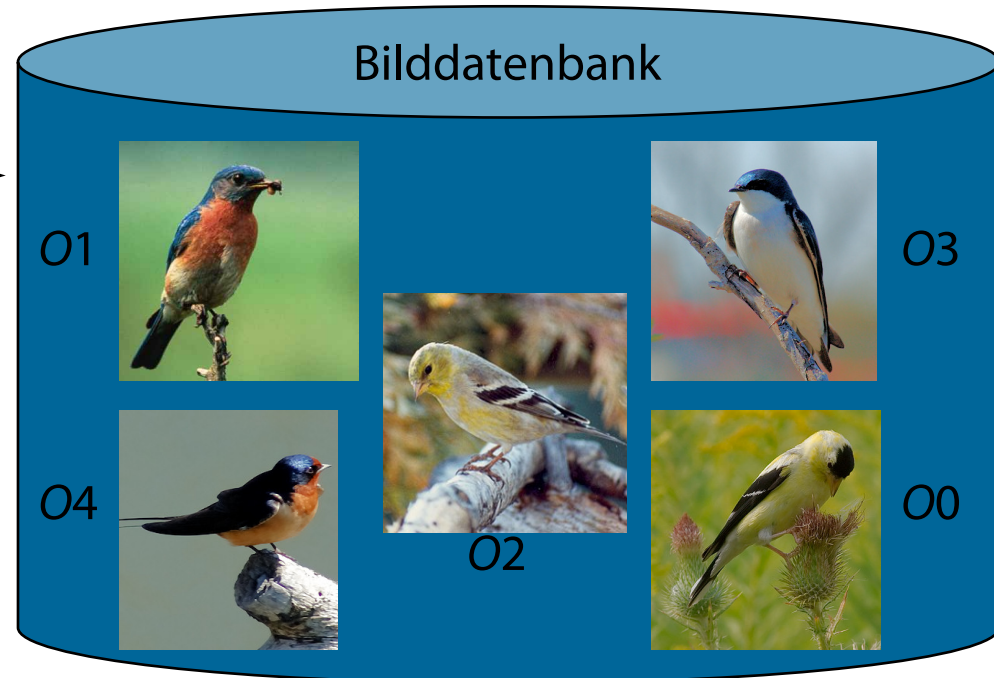
Nehmen wir an, die Datenbasis enthält 5 Bildobjekte  
 $O_0$ ,  $O_1$ ,  $O_2$ ,  $O_3$  and  $O_4$

Anfragebild  $Q$



?

Top-2  
Bilder



# Top- $k$ -Berechnung

---

Datenbasis entspricht  $n \times m$  Bewertungsmatrix für die Bewertungen eines jeden Objekts bzgl. aller Attribute

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

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
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			70		67
O0	63	61		56	

Kein weiterer sortierter Zugriff nötig, denn es sind k=2 Objekte mit allen Attributwerten bekannt (Objekte O1 and O3).

# 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	
O3	99	90	75	74	67	→ 405
O1	66	91	92	56	58	→ 363
O4	44	07	70	19	67	→ 207
O0	63	61	01	56	35	→ 216

Totale Bewertung

} Top-2

Daher sind die besten Objekte für die Anfrage:  
O3 mit Bewertung 405 and O1 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  $A = A_1 \wedge A_2 \wedge \dots \wedge A_m$
- **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
  - Für jedes  $x$  der Joinmenge nach Phase 1 und jedes Prädikat  $A_i$  gilt:  
 $g_{A_i}(y) \leq g_{A_i}(x)$ , denn wir haben absteigend sortiert
  - Wegen Monotonie von  $\min\{\}$  gilt:  
 $g_{A_1 \wedge A_2 \wedge \dots \wedge A_m}(y) \leq g_{A_1 \wedge A_2 \wedge \dots \wedge A_m}(x)$
  - Es gibt mindestens  $k$  solcher Objekte  $x$  (Abbruch-Kriterium Phase 1).
  - Es gibt kein  $y$ , das besser ist als die besten  $k$  gesehenen Objekte  $x$

# 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:  $10.000 \times 3 = 30.000$
- Weiterentwicklung: Threshold Algorithmus (TA),  
No Random Access Algorithmus (NRA),  
Combined Algorithm (CA)

# 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 twentieth 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 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

### Schritt 2:

- Bestimme Schwellwert  $T$  basierend auf mit sortiertem Zugriff schon gesehene Objekte
- $T = a_1(p) + a_2(p) + \dots + a_m(p)$  wobei  $p$  die aktuelle Position des sortierten Zugriffs ist
- Falls es  $k$  Objekte mit Gesamtbewertung  $\geq T$  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:
(O3, 405)
(O1, 363)

$p=1$  →

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	F
O3	99	90	75	74	67	405
O1	66	91	92	56	58	363



# Top-k-Berechnung – Algorithmus TA

## Schritt 2:

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

BUFFER:
(O3, 405)
(O1, 363)

$p=1$  →

	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	F
O3	99	90	75	74	67	405
O1	66	91	92	56	58	363

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

Es gibt keine  $k$  Objekte mit Bewertung  $\geq T$ , 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:
(O3, 405)
(O1, 363)

$p=2$  

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	F
O3	99	90	75	74	67	405
O1	66	91	92	56	58	363
O4	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
- $T = a_1(p) + a_2(p) + \dots + a_m(p)$  wobei  $p$  die aktuelle Position des sortierten Zugriffs ist
- Falls es  $k$  Objekte mit Gesamtbewertung  $\geq T$  gibt dann HALTE und bestimme Ergebnis
- sonst  $p := p + 1$  und GOTO Schritt 1

BUFFER:
(O3, 405)
(O1, 363)

$p=2$  →

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	F
O3	99	90	75	74	67	405
O1	66	91	92	56	58	363
O4	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

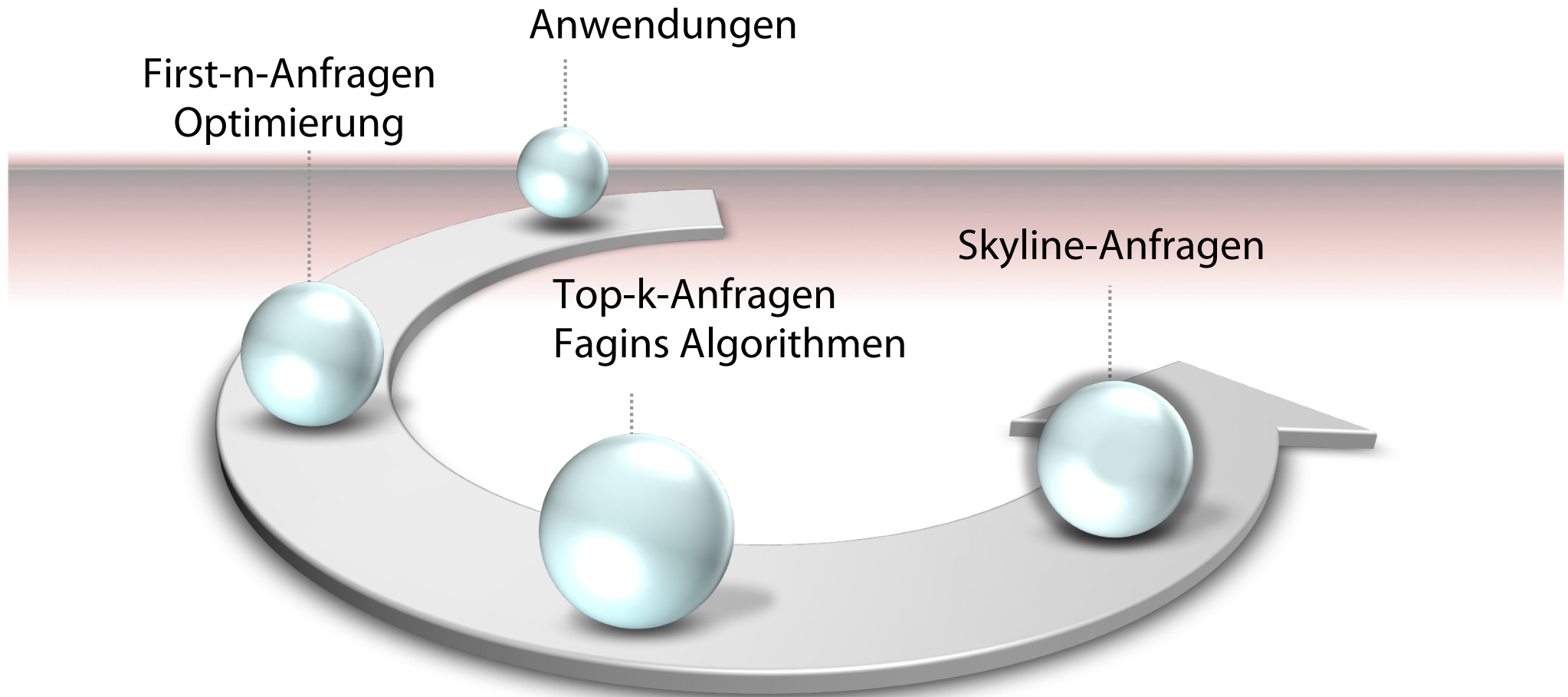
Weitere Entwicklungen:

- Verteilte Top- $k$ -Berechnung
- Top- $k$ -Berechnung mit Joins für Bewertung
- Top- $k$  mit probabilistischen Daten
- Vermeidung der Angabe von  $k$  und der Aggregation der Attributwerte

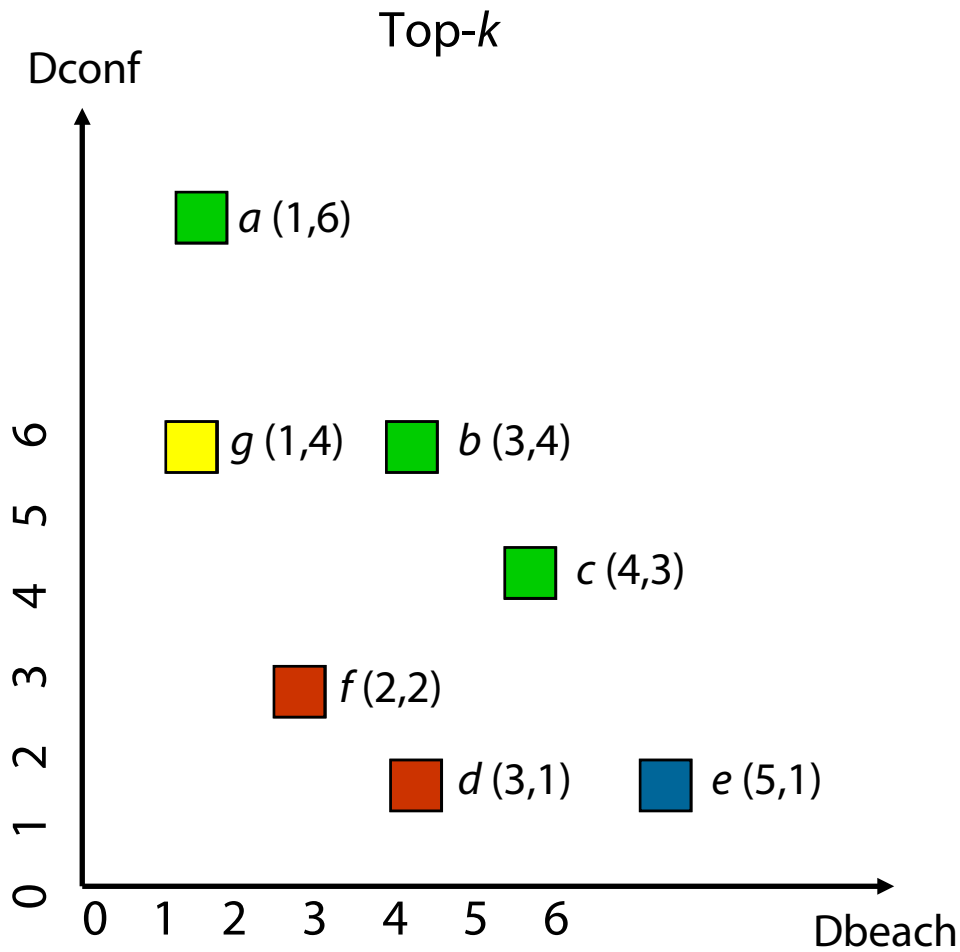
Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. In Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS '01). ACM, New York, NY, USA, 102-113., **2001**

# Non-Standard-Datenbanken

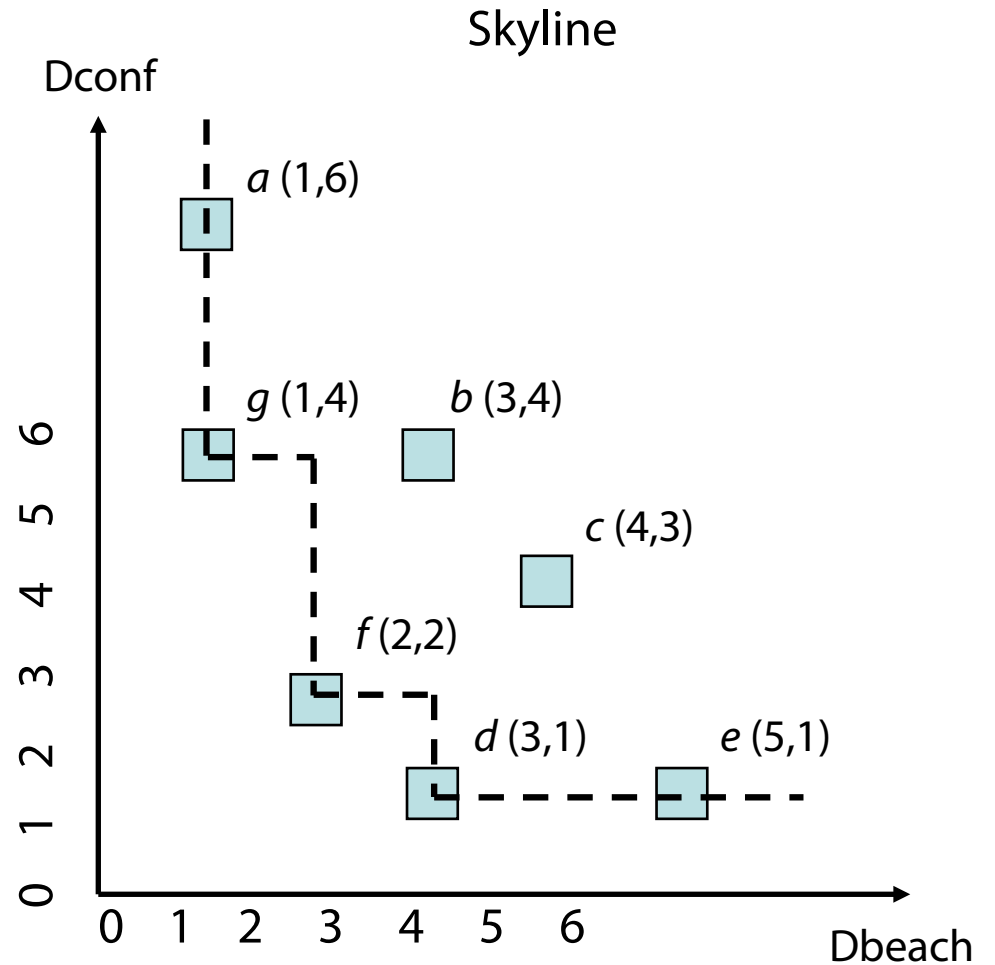
Von First-n- und Top-k-Anfragen zu Skyline-Anfrage



# Skyline-Berechnung



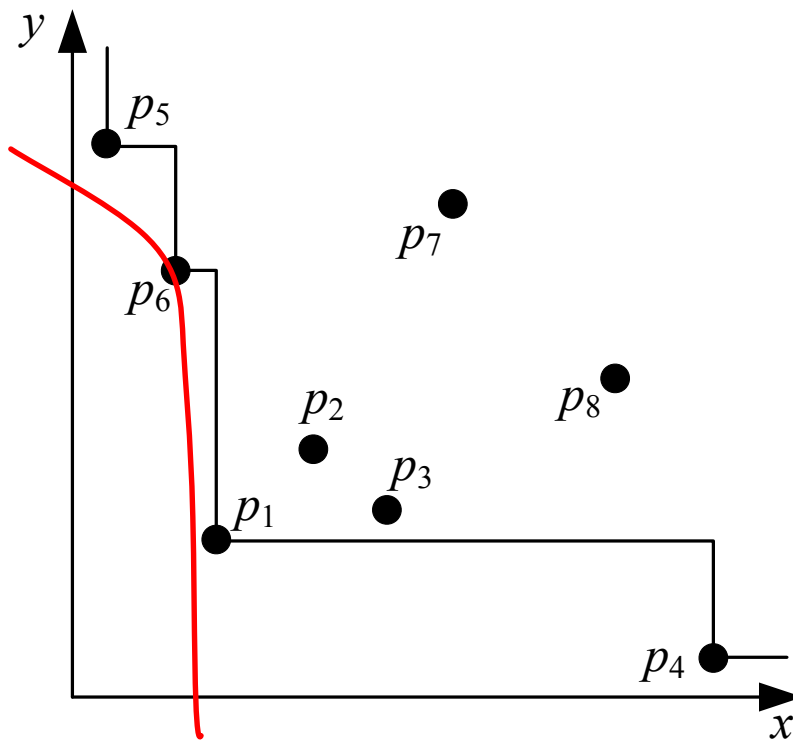
- $f, d$  (best objects)
- $g$  (next best)
- $e$  (next best)



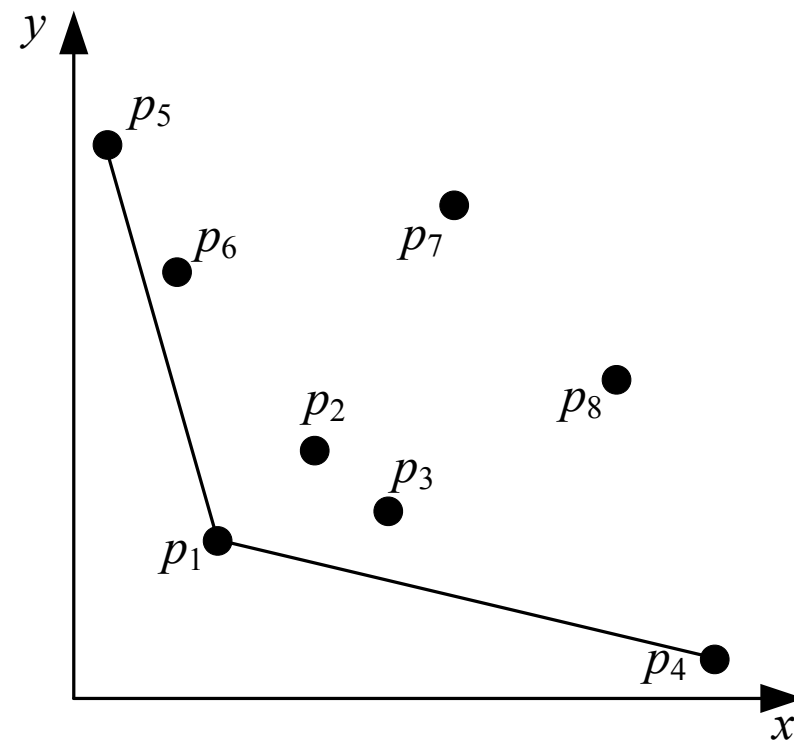
Skyline objects:  $g, f, d$

# Skyline vs. konvexe Hülle

Enthält Top-1-Objekt jeder  
**monotonen** Funktion



Enthält Top-1-Objekt jeder  
**linearen** Funktion





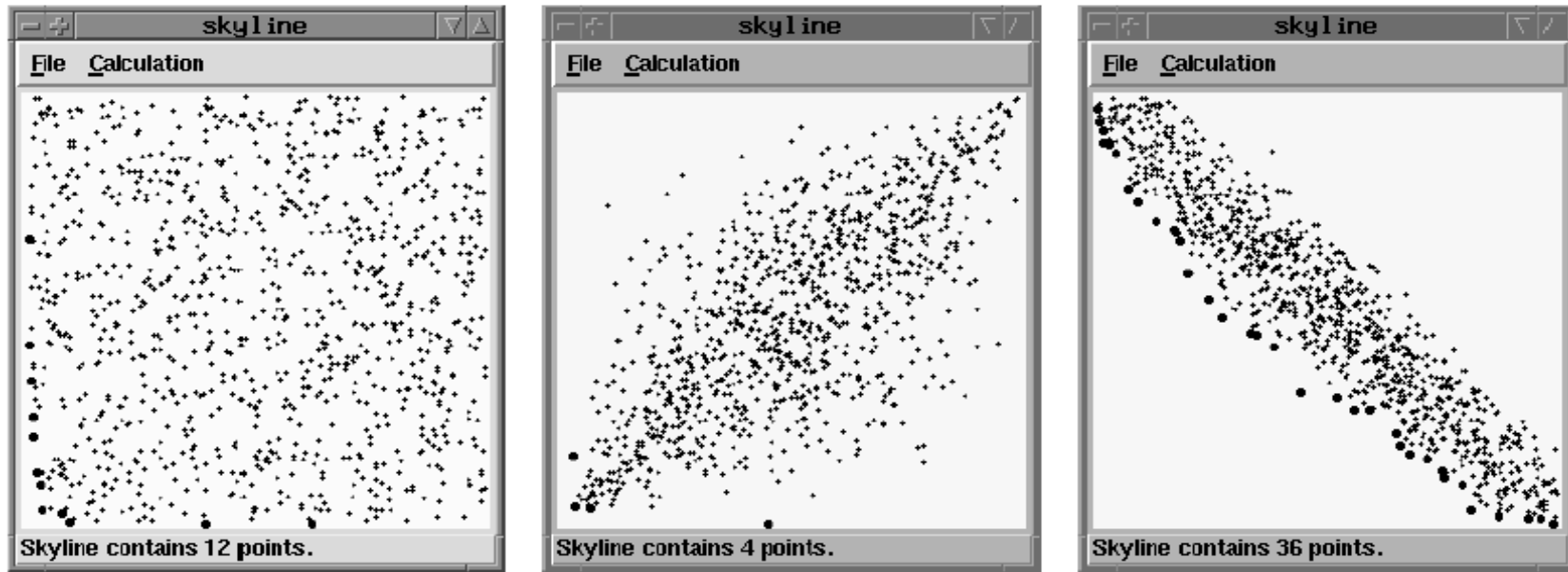
# Skyline in Standard-SQL?

---

```
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



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 ...
```

# 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  $p=(x_1, x_2, \dots, x_d)$  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	
$a (1, 9)$	$minC=1$	$k (9, 1)$	$minC=1$
$b (2, 10)$	$minC=2$	$i (3, 2), m (6, 2)$	$minC=2$
$c (4, 8)$	$minC=4$	$h (4, 3), n (8, 3)$	$minC=3$
$g (5, 6)$	$minC=5$	$l (10, 4)$	$minC=4$
$d (6, 7)$	$minC=6$	$f (7, 5)$	$minC=5$
$e (9, 10)$	$minC=9$		

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

Meist für  
Punkte nicht  
vorgesehen

# Indexbasierte Skyline-Bestimmung

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

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

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

- ▶ Vergleiche Mengen {*b*} und {*k*}, füge {*k*} zur Skyline-Liste
- ▶ Lade {*b*} und {*i,m*}; Find zuerst Skyline in {*i,m*} first, also {*i*}
- ▶ Vergleiche {*i*} und {*b*}, füge {*i*} zur Skyline-Liste hinzu
- ▶ Algorithmus hält, weil keine anderen Teilmengen besser sind al {*i*}
- ▶ Skyline ist {*a,k,i*}

# Algorithmus Block-Nested-Loop (BNL)

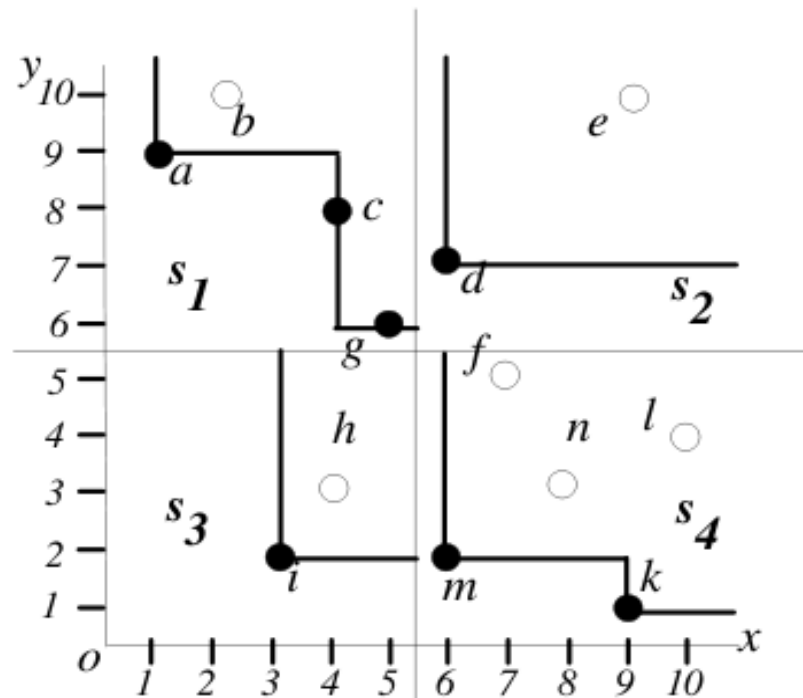
---

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

# Divide and Conquer (Skyline-Algorithmus DC)

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



Teillösungen

{a,c,g}

{d}

{i}

{m,k}

Gesamtlösung

{a,i,k}

Nachteil: Hohe IO-Kosten

Funktioniert nur bei kleiner Skyline



# Bewertung

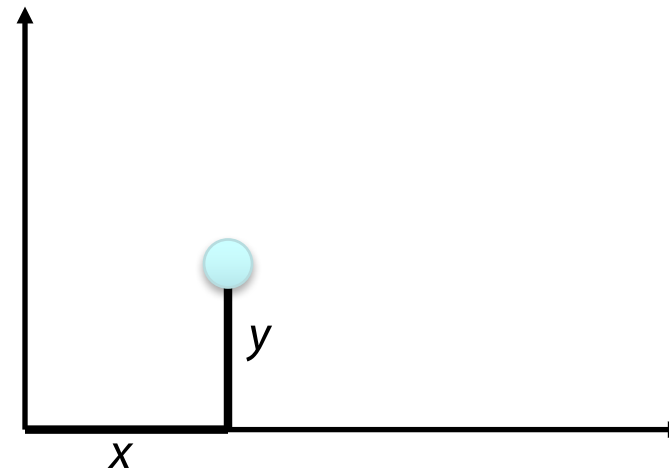
---

- 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 Partitionen)

# Annahme

---

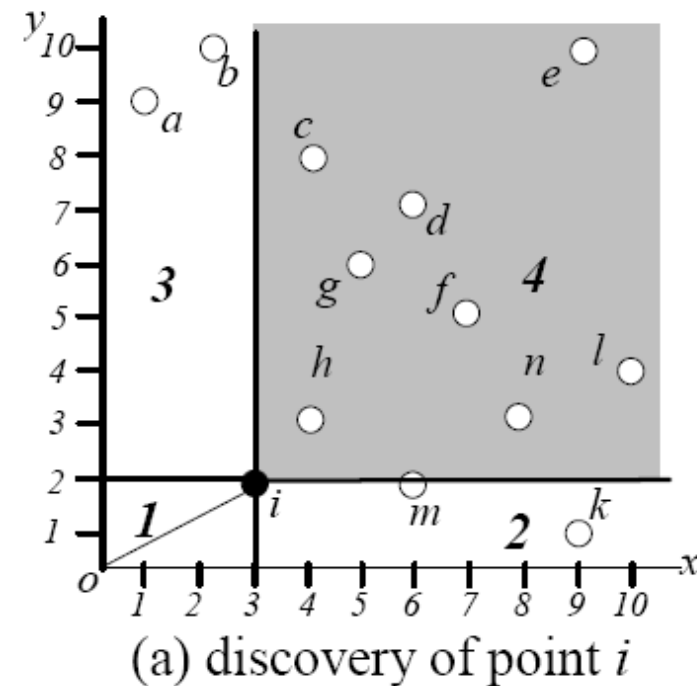
- Datenpunkte in k-d-Baum eingetragen
- Abstand eines Punktes vom Ursprung als Manhattan-Anstand modelliert
- NN-Anfrage mit größer werdendem Rechteck implementiert



$$\text{mindist}(e.mbr) = x + y$$

# Skyline-Algorithmus Nächste Nachbarn (NN)

- Führe NN-Anfrage auf R-Baum vom Ursprung aus zur Bestimmung des nächsten Nachbarn  $o$
- Alle Punkt in der dominierten Region nicht mehr betrachtet
- Ergebnis der NN-Suche verwendet zur Partitionierung des Raumes



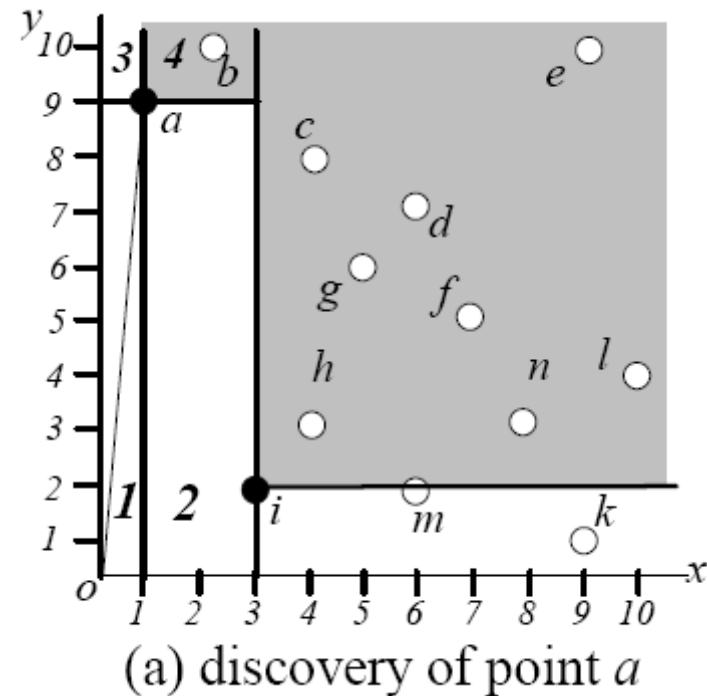
Zwei Partitionen  $[0, i_x) [0, \infty)$  und  $(ii) [0, \infty) [0, i_y)$

Partition1: 1, 3

Partition2: 1, 2

# Nächste Nachbarn (NN)

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



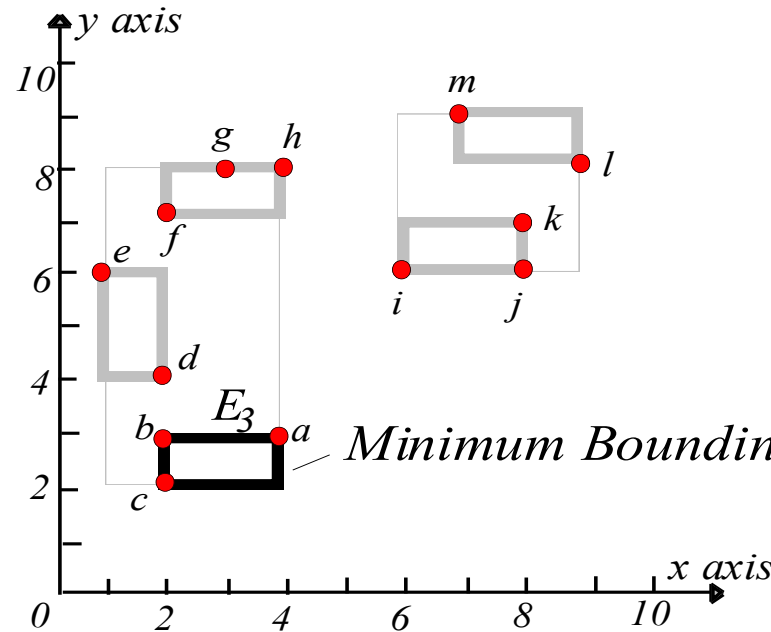
$[0, a_x)$   $[0, \infty)$

*Unterteilungen 1 and 3*

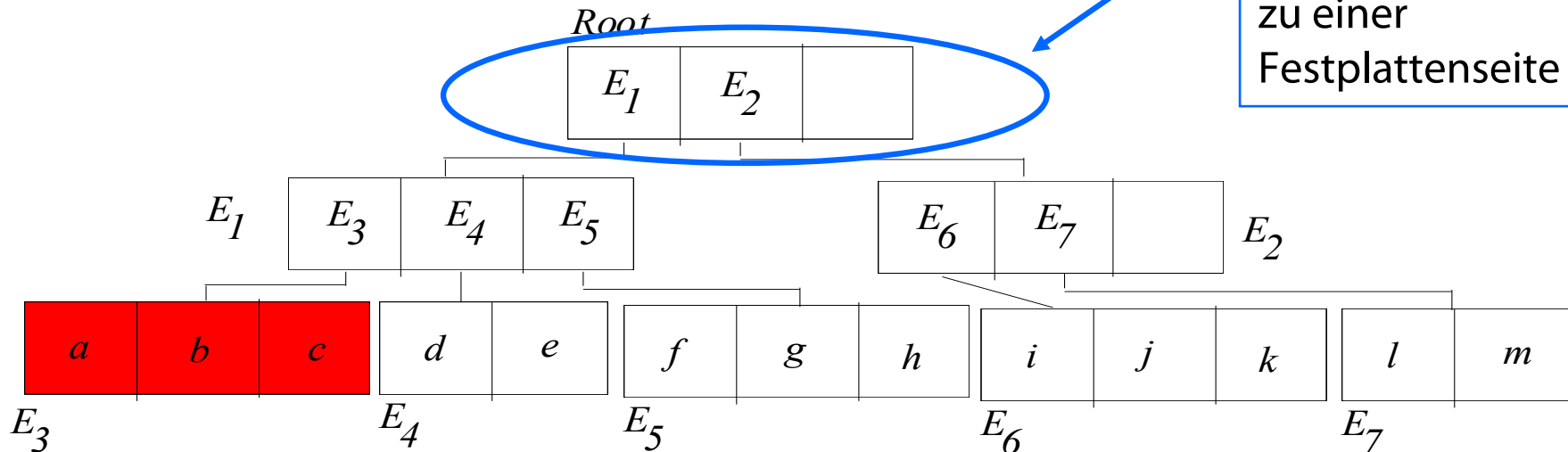
$[0, i_x)$   $[0, a_y)$

*Unterteilungen 1 and 2*

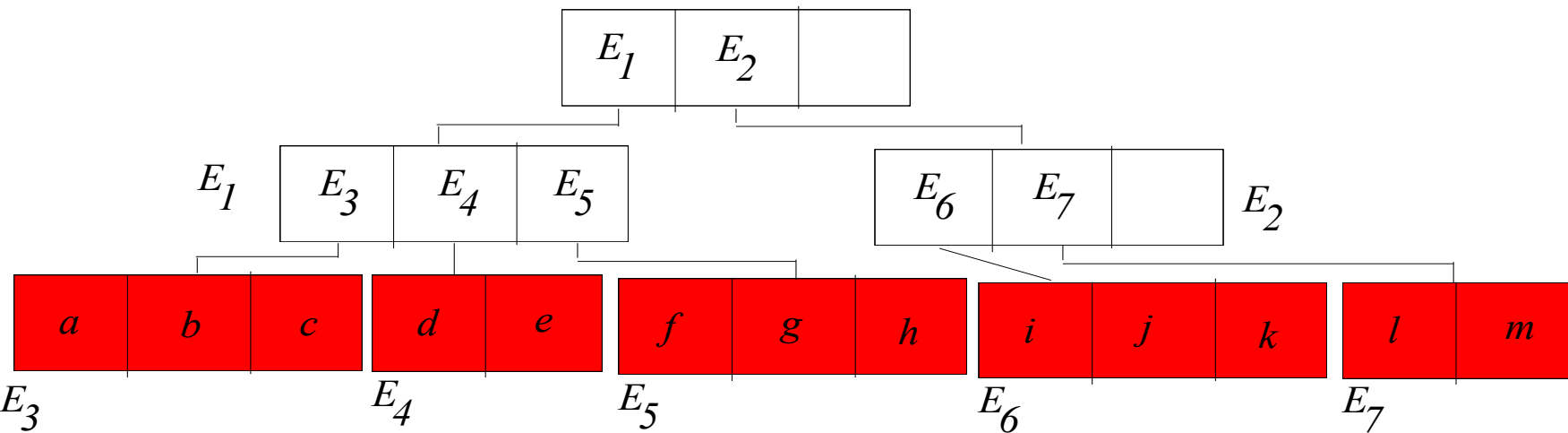
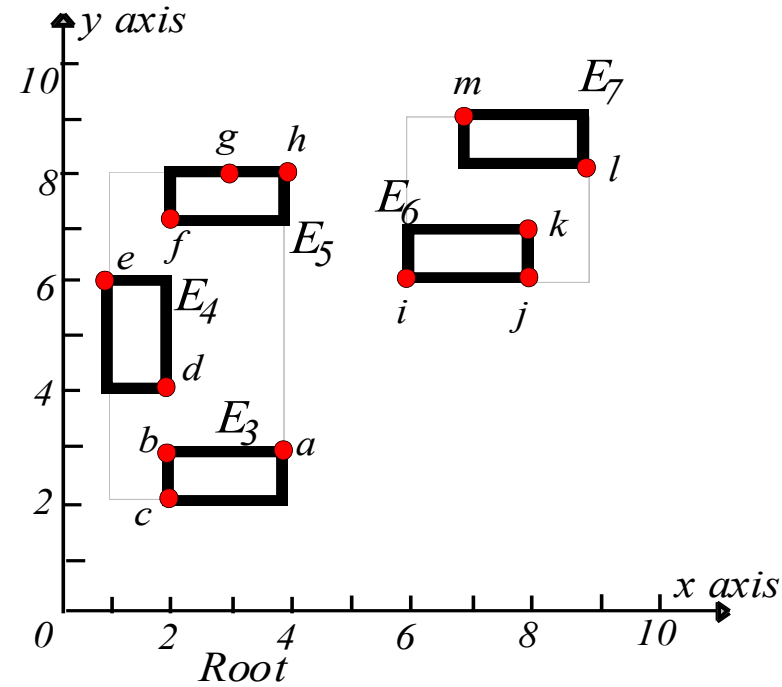
# R-Bäume mit Punktdaten



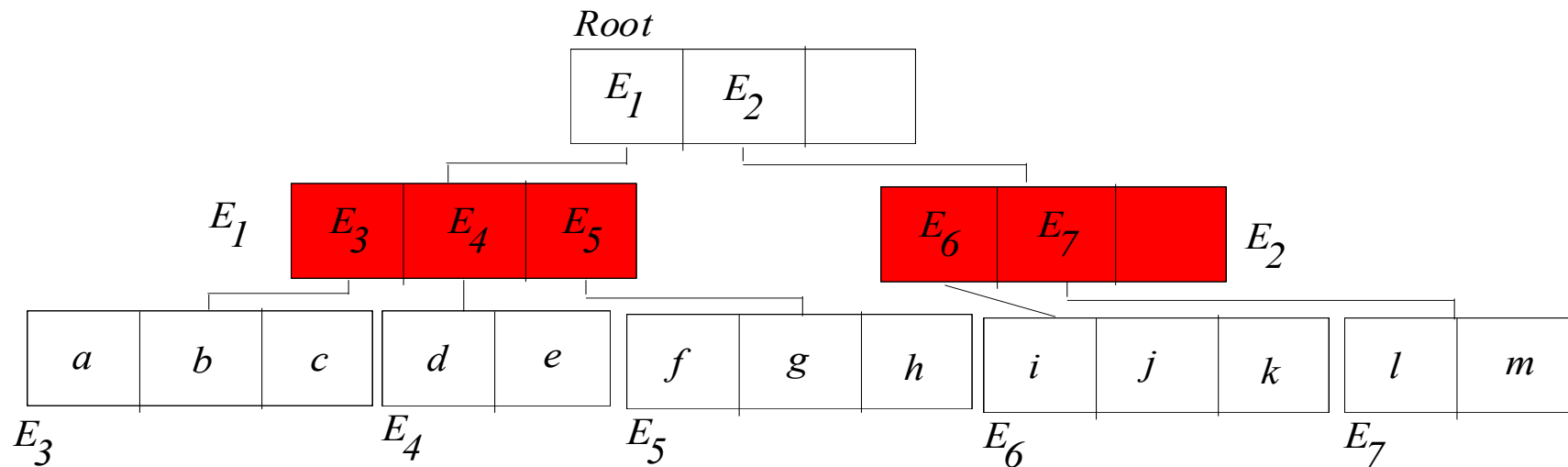
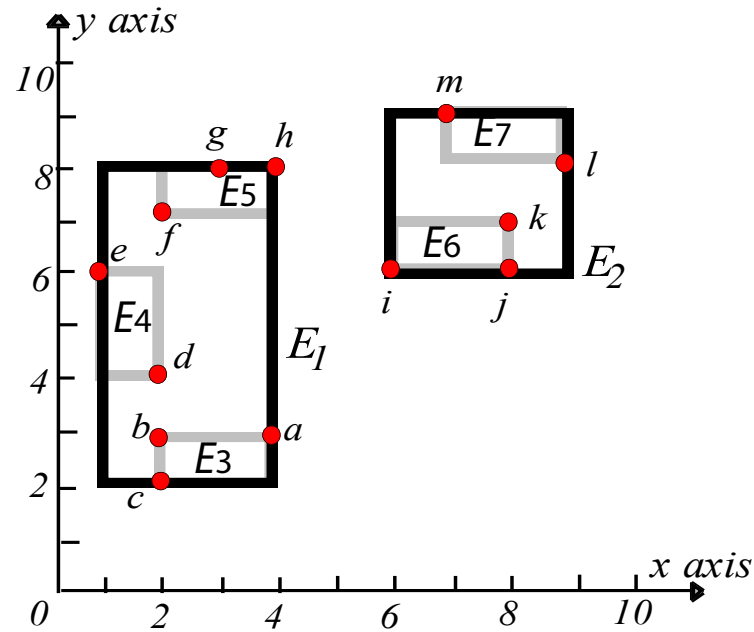
Jeder Knoten korrespondiert zu einer Festplattenseite



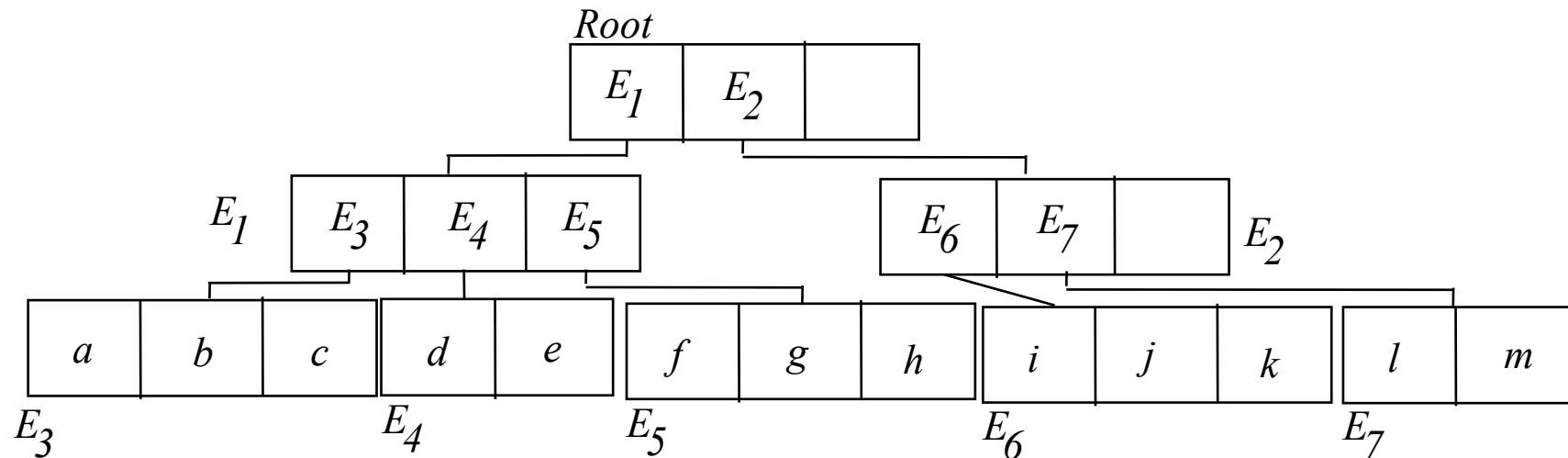
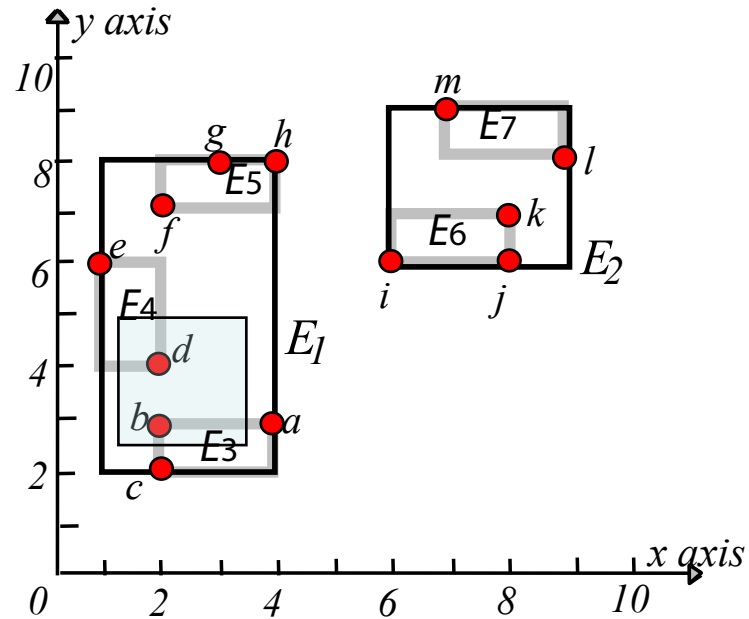
# R-Bäume – Struktur



# R-Bäume – Struktur

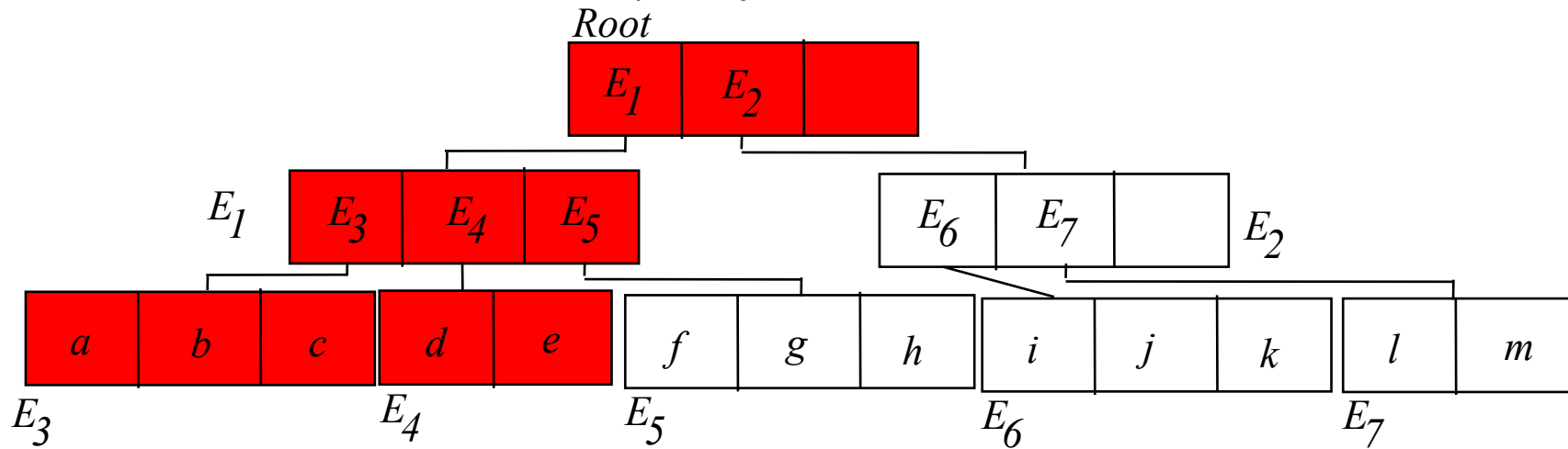
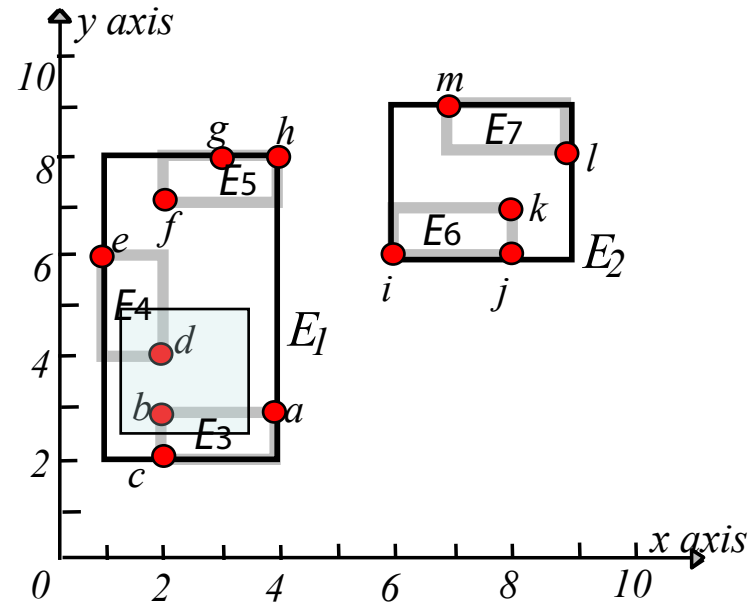


# R-Bäume – Bereichsanfrage





# R-Bäume – Bereichsanfrage

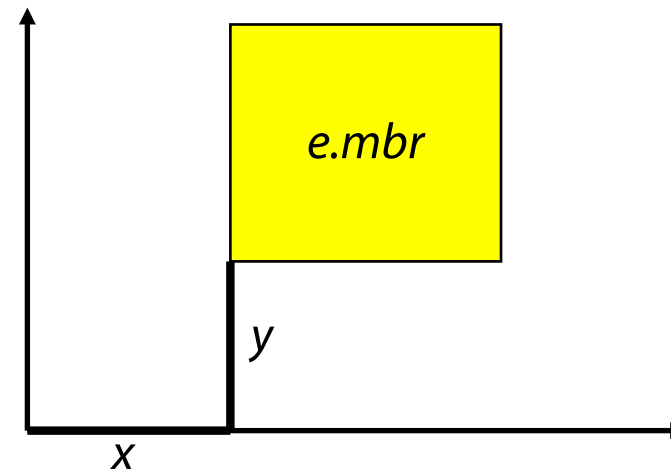


# BBS Algorithmus

---

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

Links-unten von  $mbr(e)$  bis Nullpunkt



$$\text{mindist}(e.mbr) = x + y$$

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

---

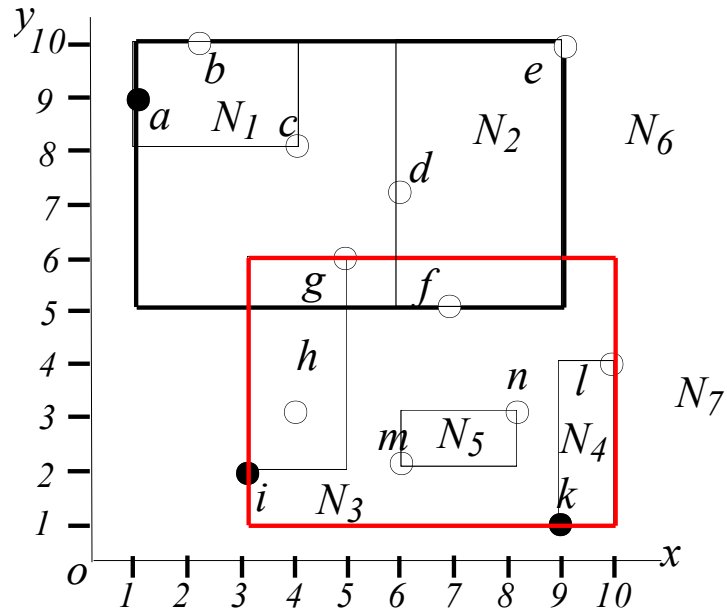
1. Wann verzweigen (**branch**): Welchen Teil des Suchraums soll als nächstes betrachtet werden?
2. Wie begrenzen (**bound**): Welche Teile des Suchraums können sicher eliminiert werden

# BBS Algorithmus(r-tree)

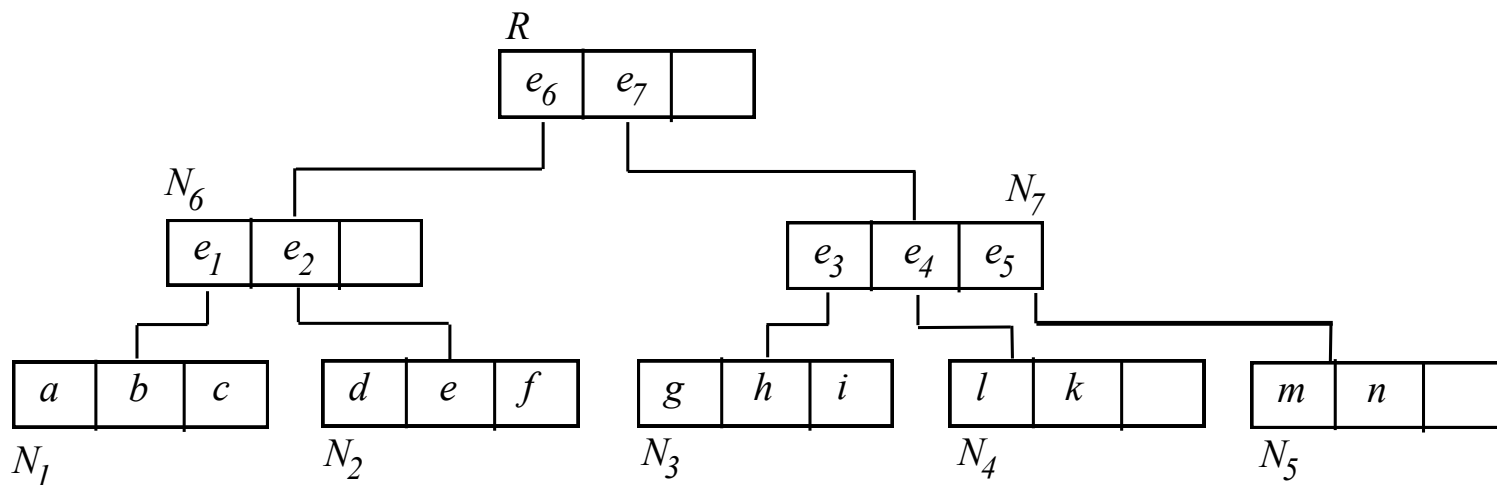
---

```
S := ∅ // Skyline
PQ := { R ∈ root(r-tree) } // Warteschlange
while not empty(PQ)
  x := min(PQ)
  if datapoint(x)
    then prüfe ob x von einem Punkt in S dominiert wird
      Falls nein, füge x zu S hinzu
    sonst Betrachte Teilrechtecke von x
      und sortiere Teilrechtecke gemäß
      jeweiligem mindist-Maß in Warteschlange ein
return S
```

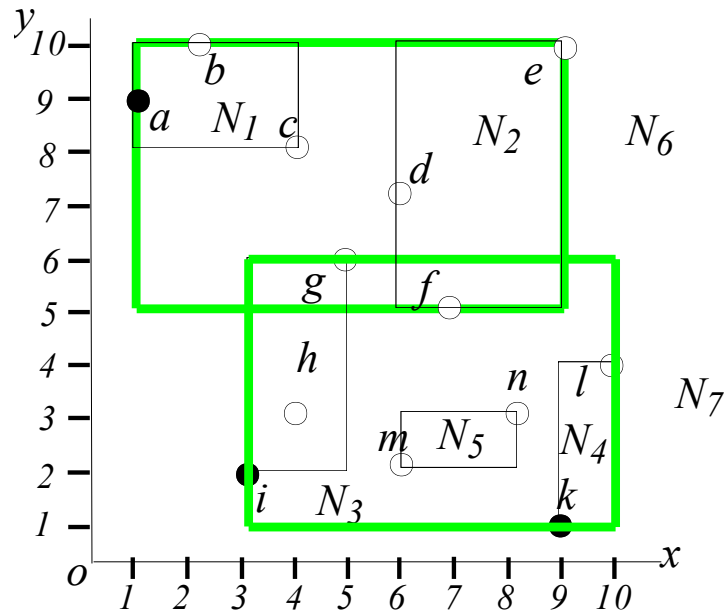
# BBS Algorithmus – Beispiel



- Nehme an, alle Punkte sind in einem R-Baum eingetragen
- $\text{mindist}(\text{MBR}) = \text{Manhattan-Distanz zwischen Punkt links- unten und Ursprung}$



# BBS Algorithmus – Beispiel

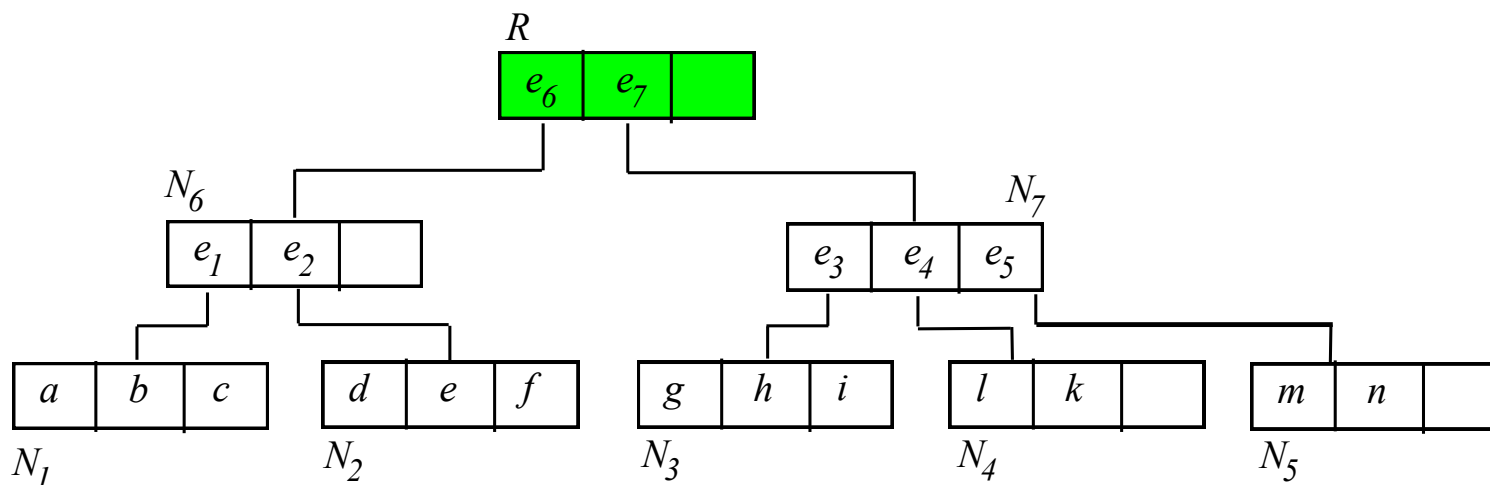


- Warteschlangenschlüssel = **mindist** vom MBR.

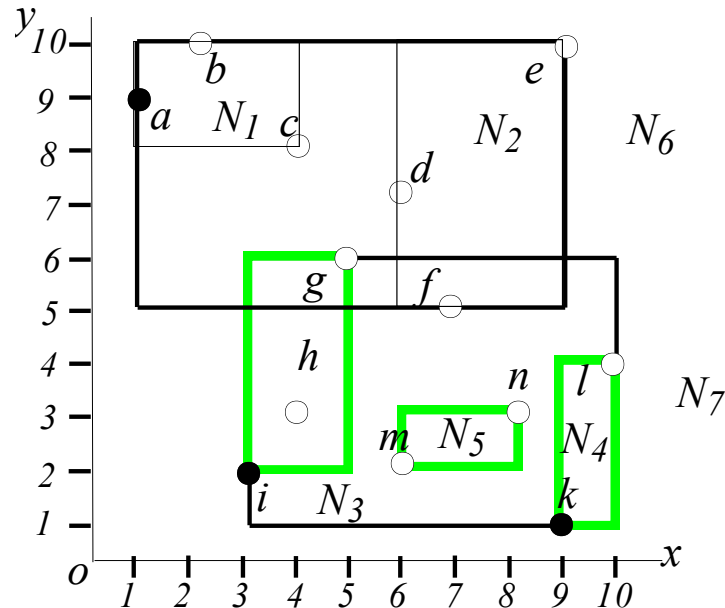
Aktion  
access root

Warteschlange  
 $\langle e_7, 4 \rangle \langle e_6, 6 \rangle$

$S$   
 $\emptyset$

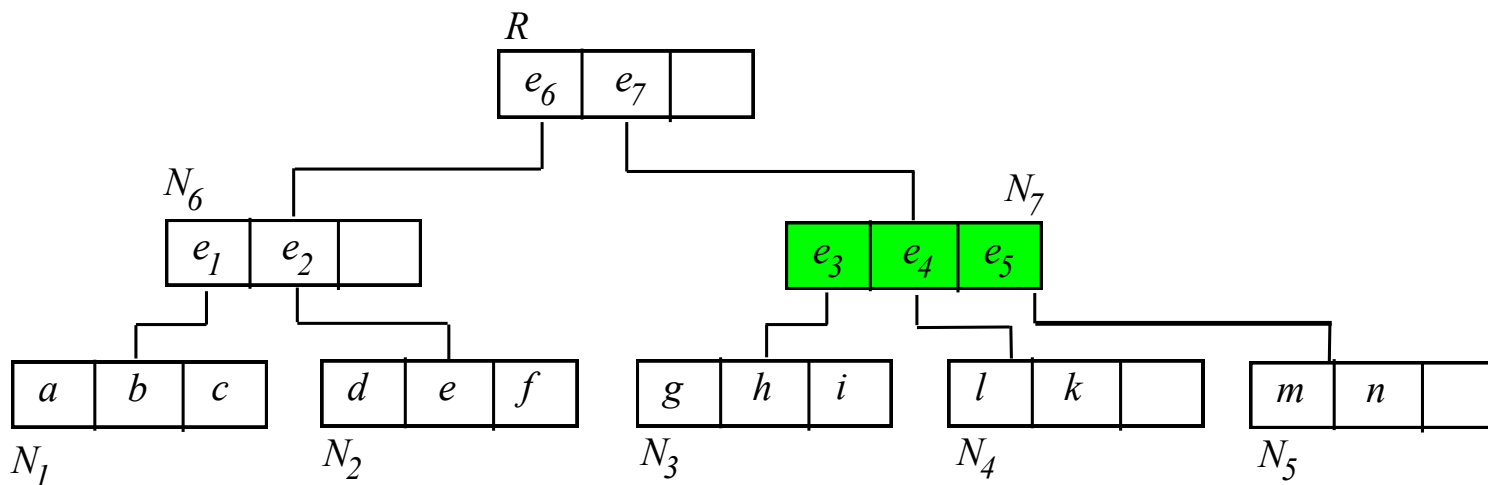


# BBS Algorithmus – Beispiel

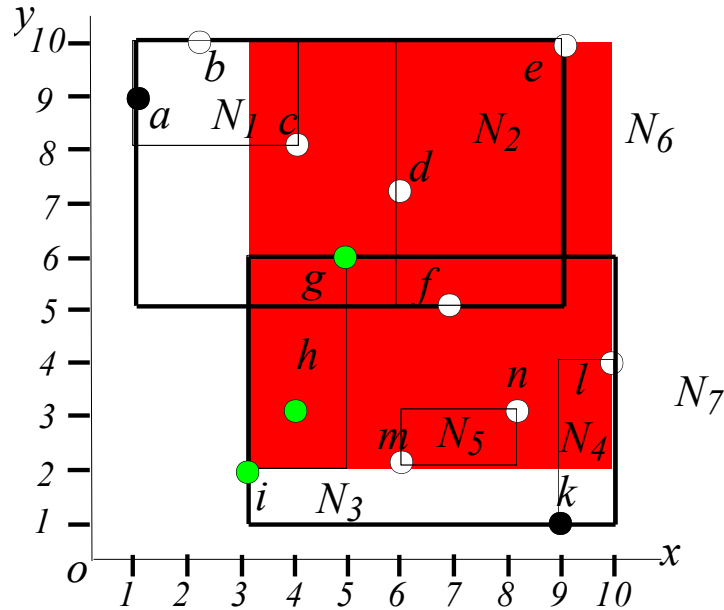


- Verarbeitung nach mindist-Werten sortiert

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



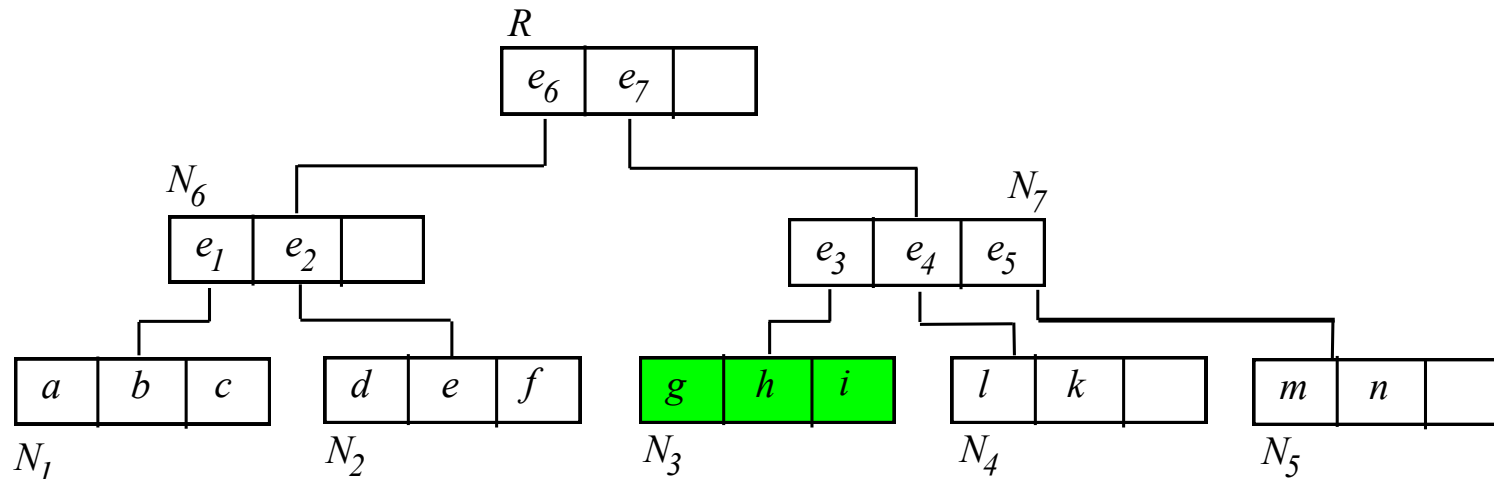
# BBS Algorithmus – Beispiel



Aktion  
 access root  
 expand  $e_7$   
 expand  $e_3$

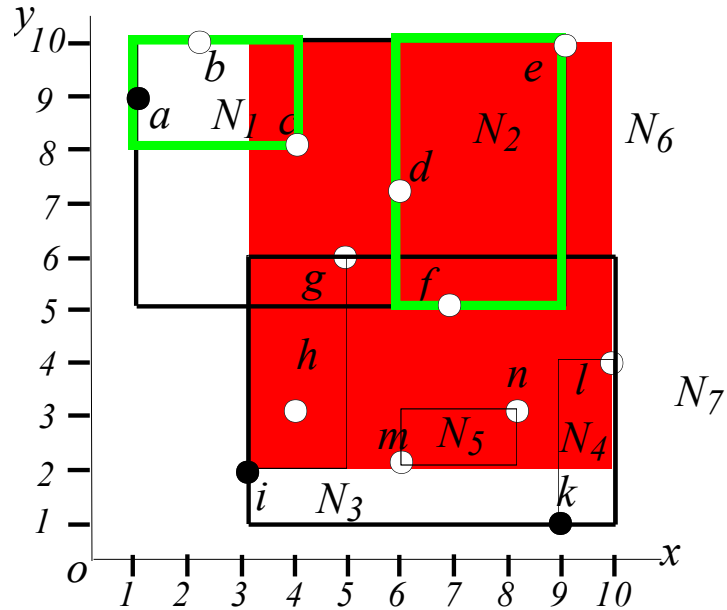
Warteschlange  
 $\langle e_7, 4 \rangle \langle e_6, 6 \rangle$   
 $\langle e_3, 5 \rangle \langle e_6, 6 \rangle \langle e_5, 8 \rangle \langle e_4, 10 \rangle$   
 $\langle i, 5 \rangle \langle e_6, 6 \rangle \langle e_5, 8 \rangle \langle e_4, 10 \rangle$

$S$   
 $\emptyset$   
 $\emptyset$   
 $\{i\}$





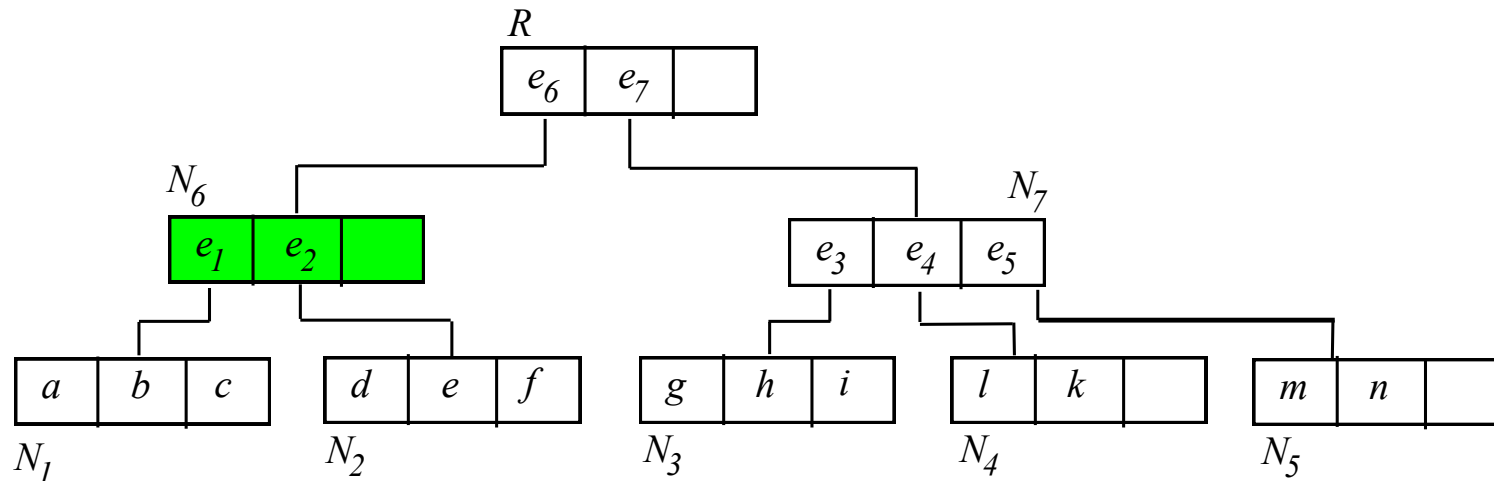
# BBS Algorithmus – Beispiel



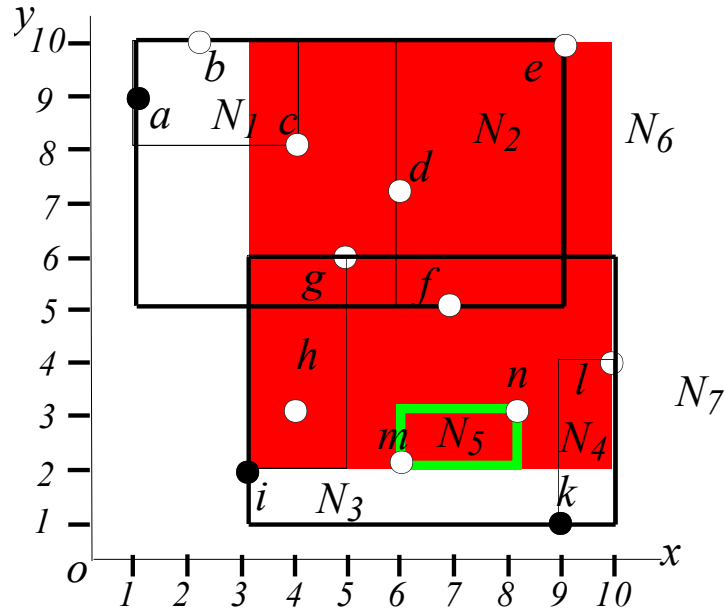
Aktion  
 access root  
 expand  $e_7$   
 expand  $e_3$   
 expand  $e_6$

Warteschlange  
 $\langle e_7, 4 \rangle \langle e_6, 6 \rangle$   
 $\langle e_3, 5 \rangle \langle e_6, 6 \rangle \langle e_5, 8 \rangle \langle e_4, 10 \rangle$   
 $\langle i, 5 \rangle \langle e_6, 6 \rangle \langle e_5, 8 \rangle \langle e_4, 10 \rangle$   
 $\langle e_5, 8 \rangle \langle e_1, 9 \rangle \langle e_4, 10 \rangle$

$S$   
 $\emptyset$   
 $\emptyset$   
 $\{i\}$   
 $\{i\}$



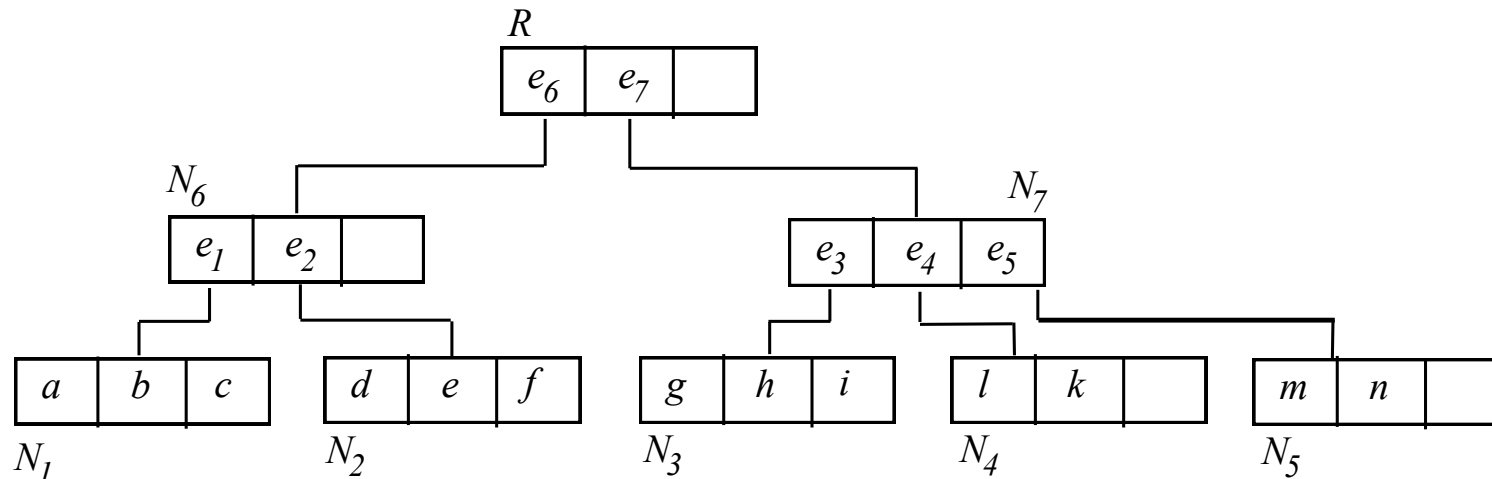
# BBS Algorithmus – Beispiel



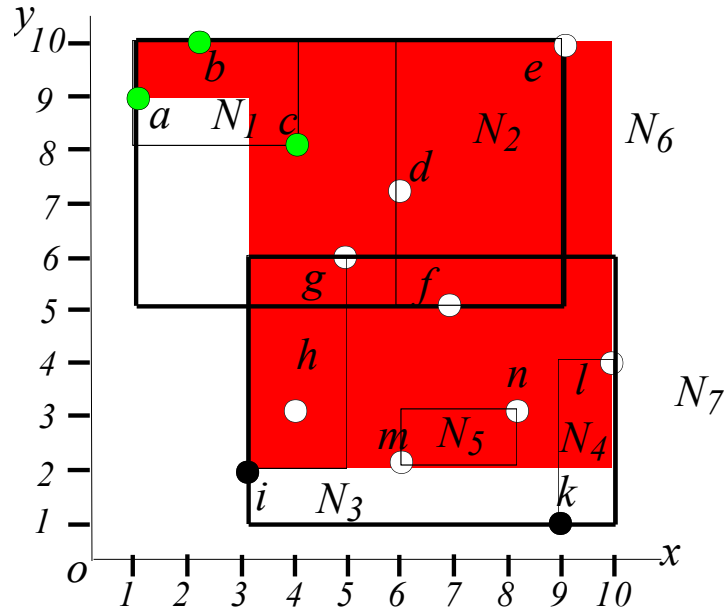
Aktion  
 access root  
 expand  $e_7$   
 expand  $e_3$   
 expand  $e_6$   
 remove  $e_5$

Warteschlange  
 $\langle e_7, 4 \rangle \langle e_6, 6 \rangle$   
 $\langle e_3, 5 \rangle \langle e_6, 6 \rangle \langle e_5, 8 \rangle \langle e_4, 10 \rangle$   
 $\langle i, 5 \rangle \langle e_6, 6 \rangle \langle e_5, 8 \rangle \langle e_4, 10 \rangle$   
 $\langle e_5, 8 \rangle \langle e_1, 9 \rangle \langle e_4, 10 \rangle$   
 $\langle e_1, 9 \rangle \langle e_4, 10 \rangle$

$S$   
 $\emptyset$   
 $\emptyset$   
 $\{i\}$   
 $\{i\}$   
 $\{i\}$



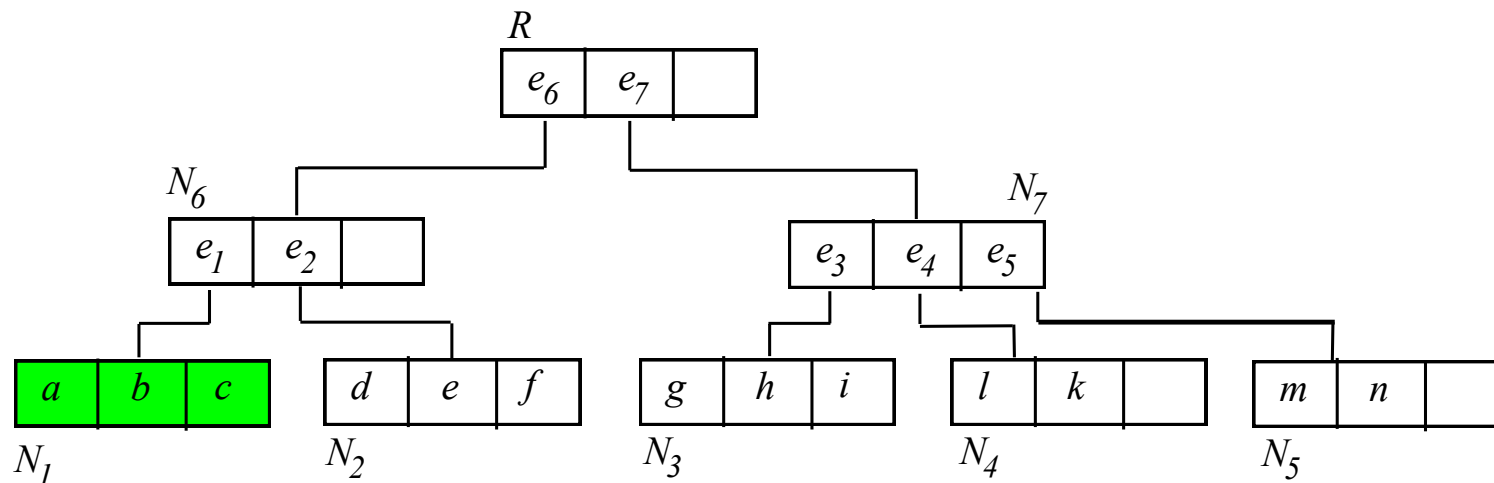
# BBS Algorithmus – Beispiel



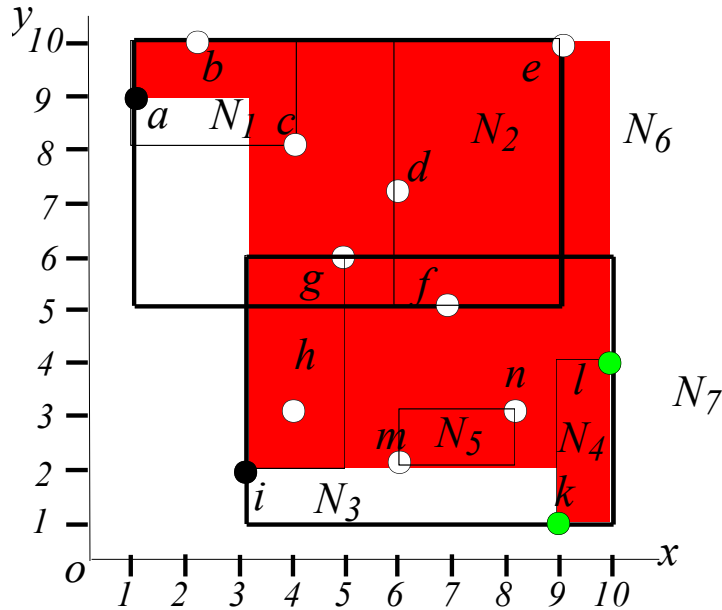
Aktion  
 access root  
 expand  $e_7$   
 expand  $e_3$   
 expand  $e_6$   
 remove  $e_5$   
 expand  $e_1$

Warteschlange  
 $\langle e_7, 4 \rangle \langle e_6, 6 \rangle$   
 $\langle e_3, 5 \rangle \langle e_6, 6 \rangle \langle e_5, 8 \rangle \langle e_4, 10 \rangle$   
 $\langle i, 5 \rangle \langle e_6, 6 \rangle \langle e_5, 8 \rangle \langle e_4, 10 \rangle$   
 $\langle e_5, 8 \rangle \langle e_1, 9 \rangle \langle e_4, 10 \rangle$   
 $\langle e_1, 9 \rangle \langle e_4, 10 \rangle$   
 $\langle a, 10 \rangle \langle e_4, 10 \rangle$

$S$   
 $\emptyset$   
 $\emptyset$   
 $\{i\}$   
 $\{i\}$   
 $\{i\}$   
 $\{i, a\}$



# BBS Algorithmus – Beispiel

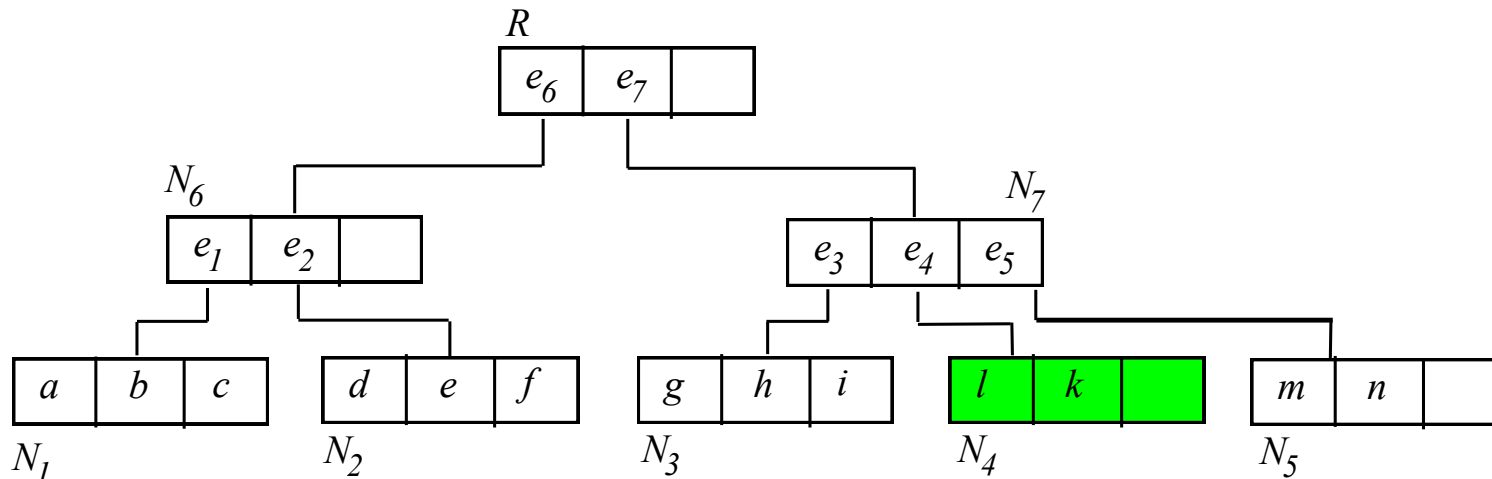


**Aktion**  
 access root  
 expand  $e_7$   
 expand  $e_3$   
 expand  $e_6$   
 remove  $e_5$   
 expand  $e_1$   
 expand  $e_4$

**Warteschlange**

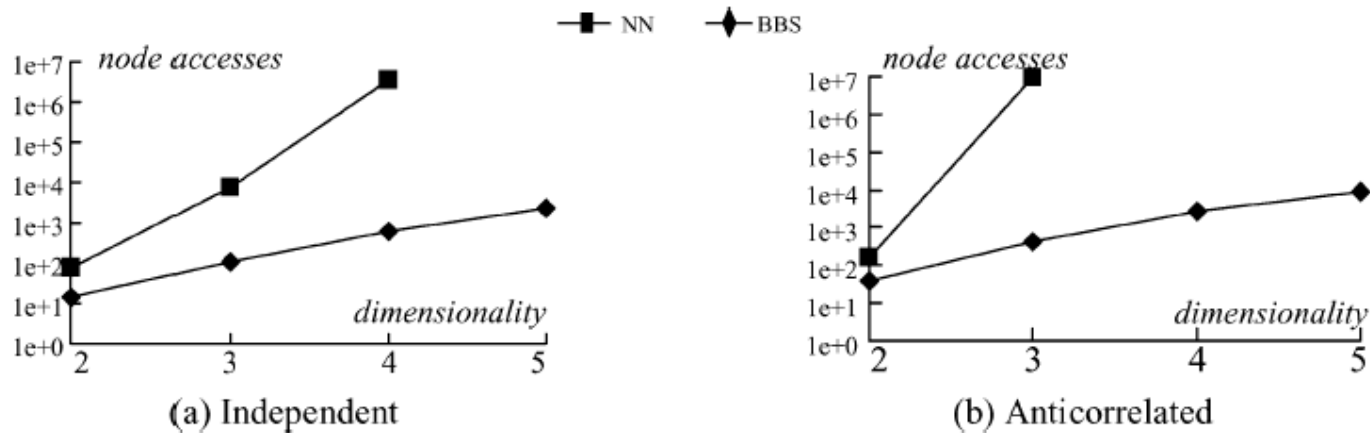
$\langle e_7, 4 \rangle \langle e_6, 6 \rangle$   
 $\langle e_3, 5 \rangle \langle e_6, 6 \rangle \langle e_5, 8 \rangle \langle e_4, 10 \rangle$   
 $\langle i, 5 \rangle \langle e_6, 6 \rangle \langle e_5, 8 \rangle \langle e_4, 10 \rangle$   
 $\langle e_5, 8 \rangle \langle e_1, 9 \rangle \langle e_4, 10 \rangle$   
 $\langle e_1, 9 \rangle \langle e_4, 10 \rangle$   
 $\langle a, 10 \rangle \langle e_4, 10 \rangle$   
 $\langle k, 10 \rangle$

**S**  
 $\emptyset$   
 $\emptyset$   
 $\{i\}$   
 $\{i\}$   
 $\{i\}$   
 $\{i, a\}$   
 $\{i, a, k\}$



# BBS Algorithm - Vergleich

BBS besser als vorige Skyline-Algorithmen in bezug auf  
CPU-Zeit and 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.

# Non-Standard-Datenbanken

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

