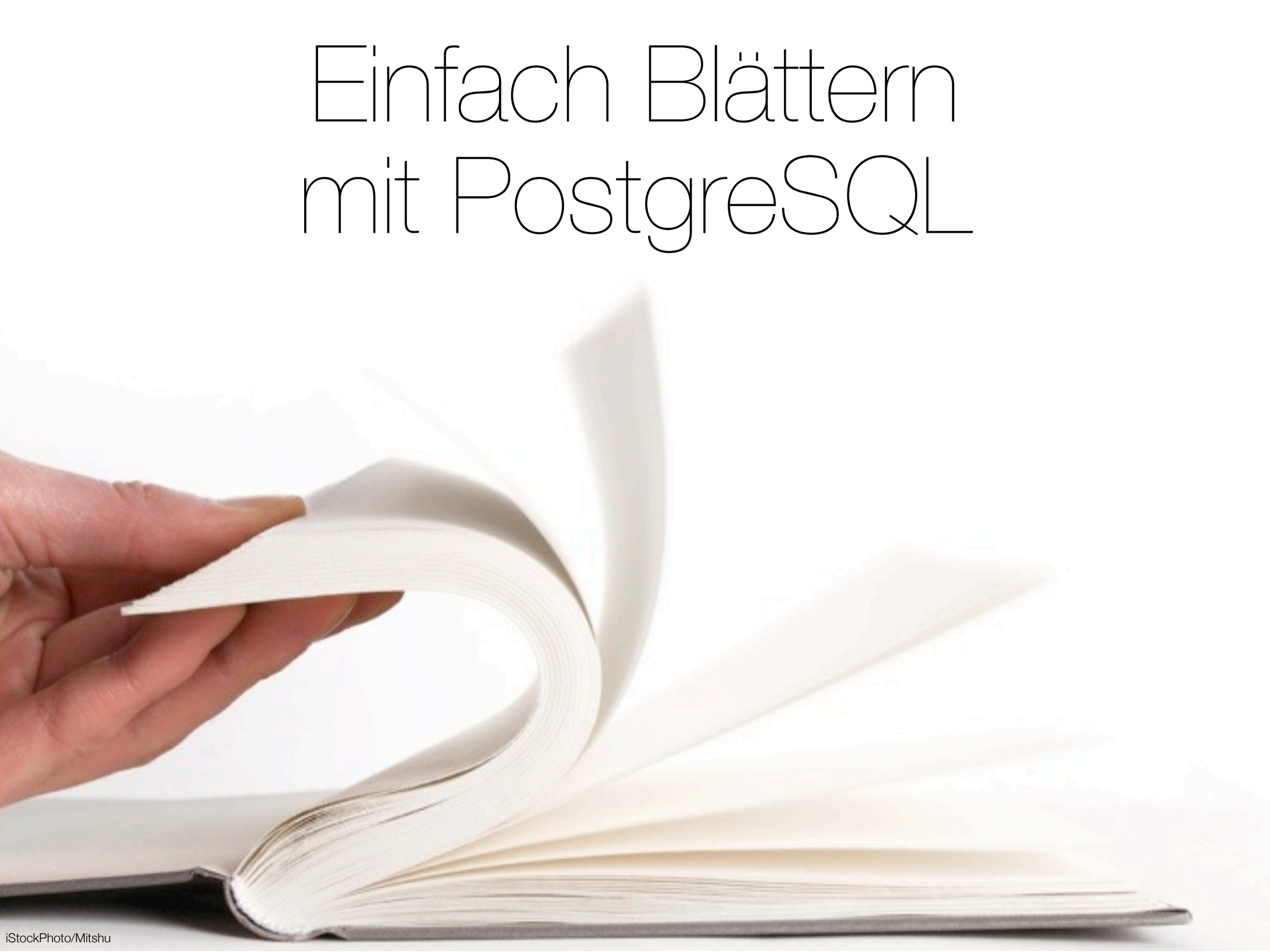
Einfach Blättern mit PostgreSQL



iStockPhoto/Mitshu

Was dich erwartet

- OFFSET ist ein Performance-Killer
- Man kann auch ohne OFFSET blättern
- Es ist schneller und hat weniger Nebeneffekte

Hinweis

In dieser Präsentation steht “Index” für “B-tree Index”

Ein einfaches Beispiel

Eine Abfrage der 10 aktuellsten Nachrichten:

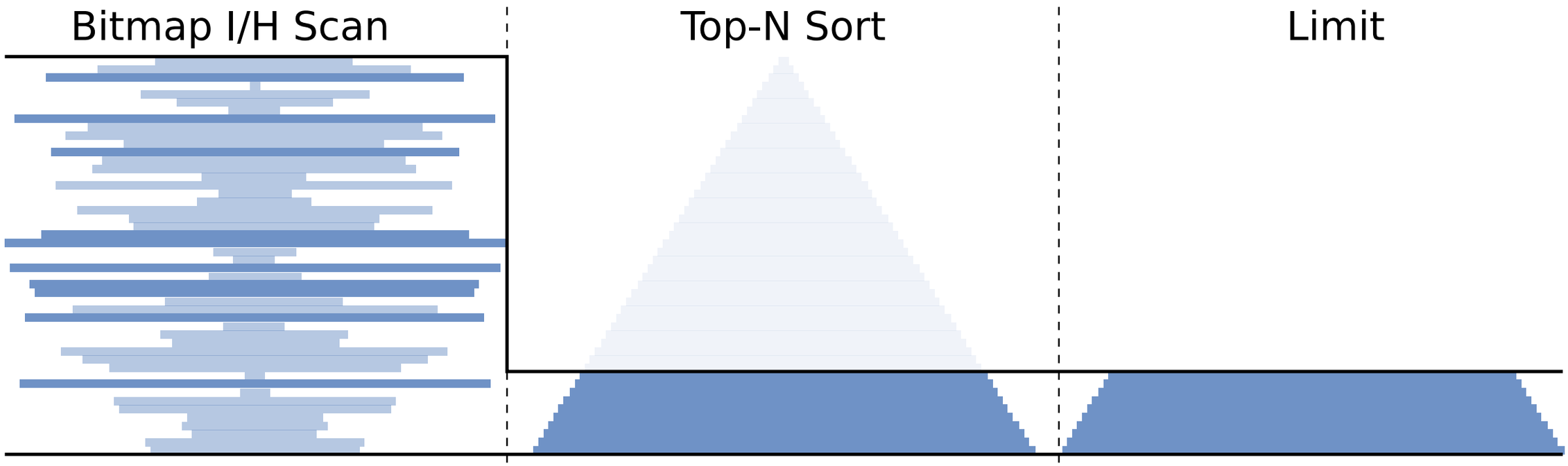
```
select *
  from news
 where topic = 1234
 order by date desc, id desc
 limit 10;
create index .. on news(topic);
```

`order by` um die aktuellsten zuerst zu bekommen.
`limit` um das Ergebnis auf 10 Zeilen zu begrenzen.

Alternative SQL-2008 Syntax (seit PostgreSQL 8.4)

```
fetch first 10 rows only
```

Worst Case: Kein Index für `order by`

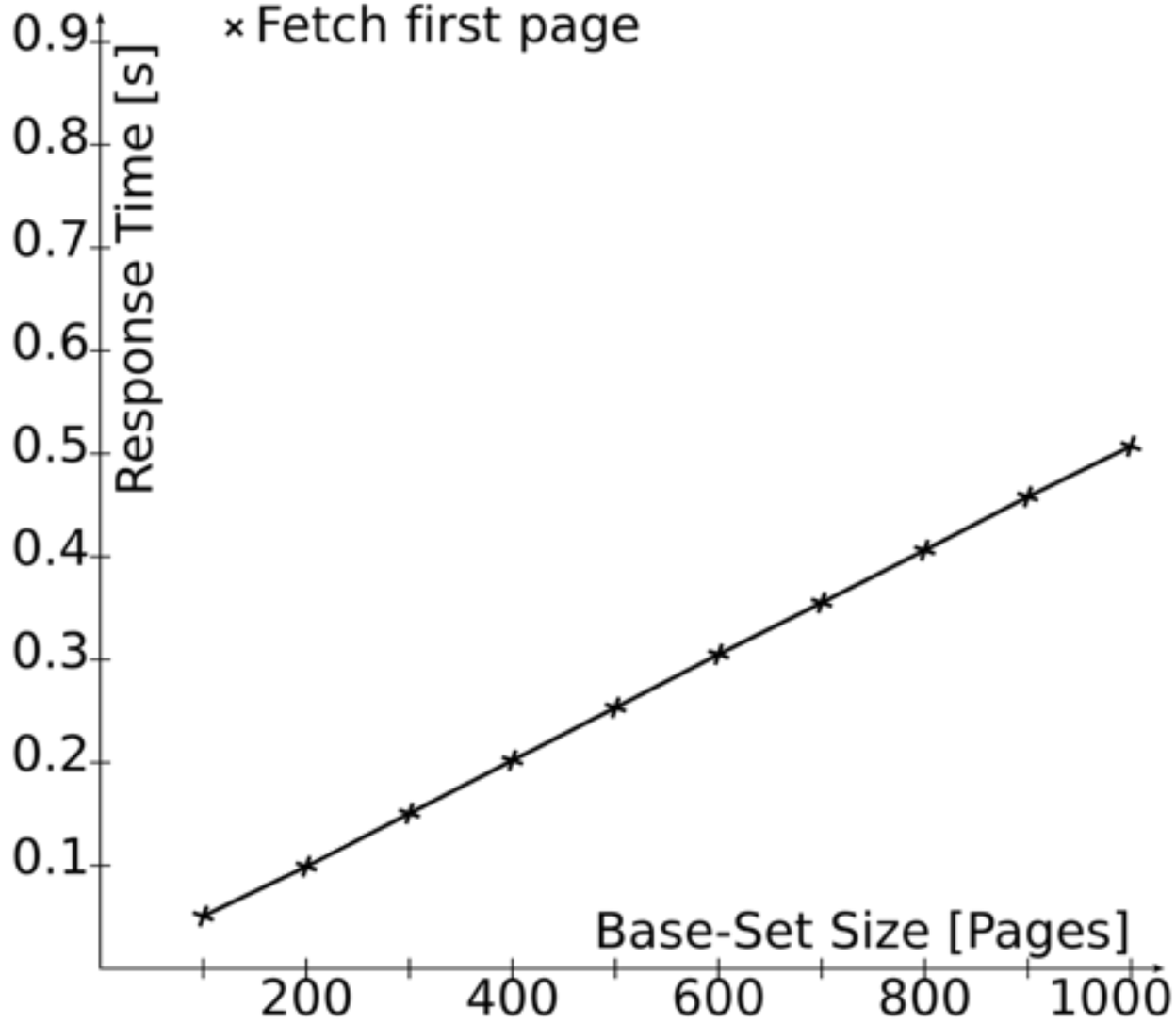



```
Limit (actual rows=10)
-> Sort (actual rows=10)
   Sort Method: top-N heapsort  Memory: 18kB
   -> Bitmap Heap Scan (rows=10000)
      Recheck Cond: (topic = 1234)
      -> Bitmap Index Scan (rows=10000)
         Index Cond: (topic = 1234)
```

Worst Case: Kein Index für `order by`

Die Performance wird durch die Zahl der Zeilen beschränkt, die die where-Klausel erfüllen (“Basis-Menge”).

Die Datenbank kann den Index für die where-Klausel nutzen, muss aber alle Zeilen laden und “sortieren”!

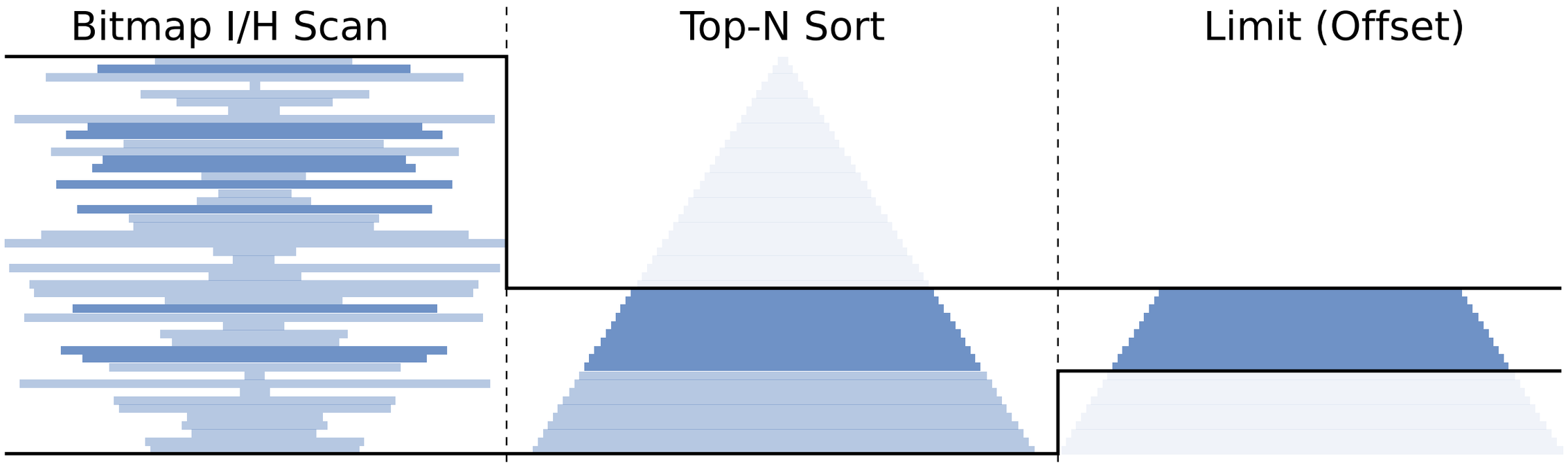


Anderes Benchmark: Nächste Seite holen

Das Laden der nächsten Seite geht mit `offset` sehr einfach:

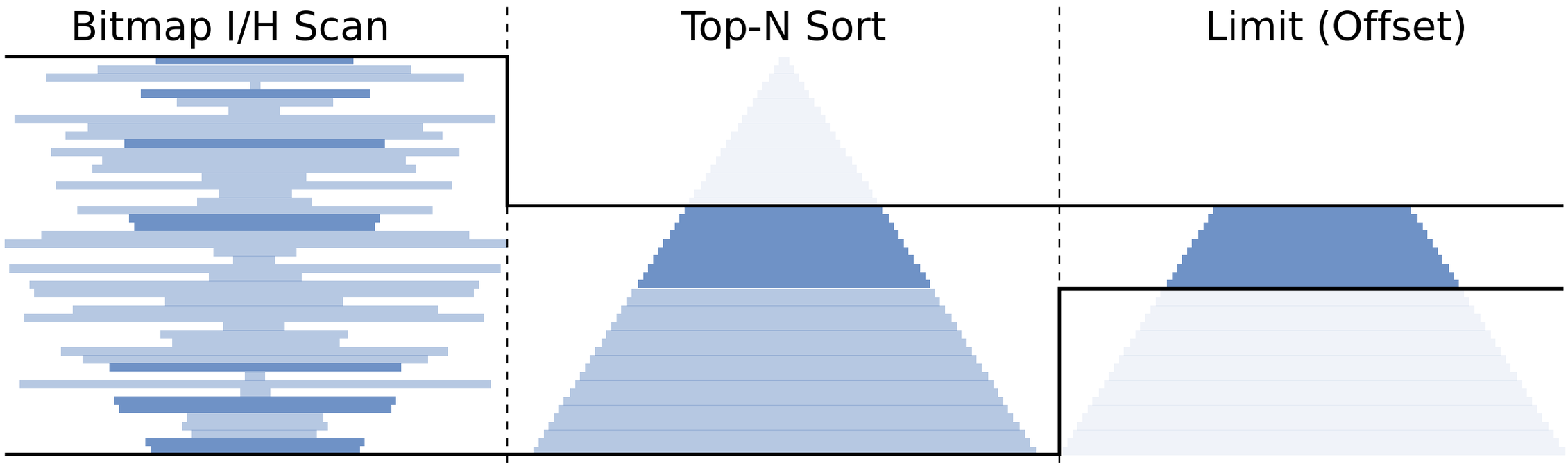
```
    select *
      from news
     where topic = 1234
     order by date desc, id desc
    offset 10
     limit 10;
```

Worst Case: Kein Index für `order by`



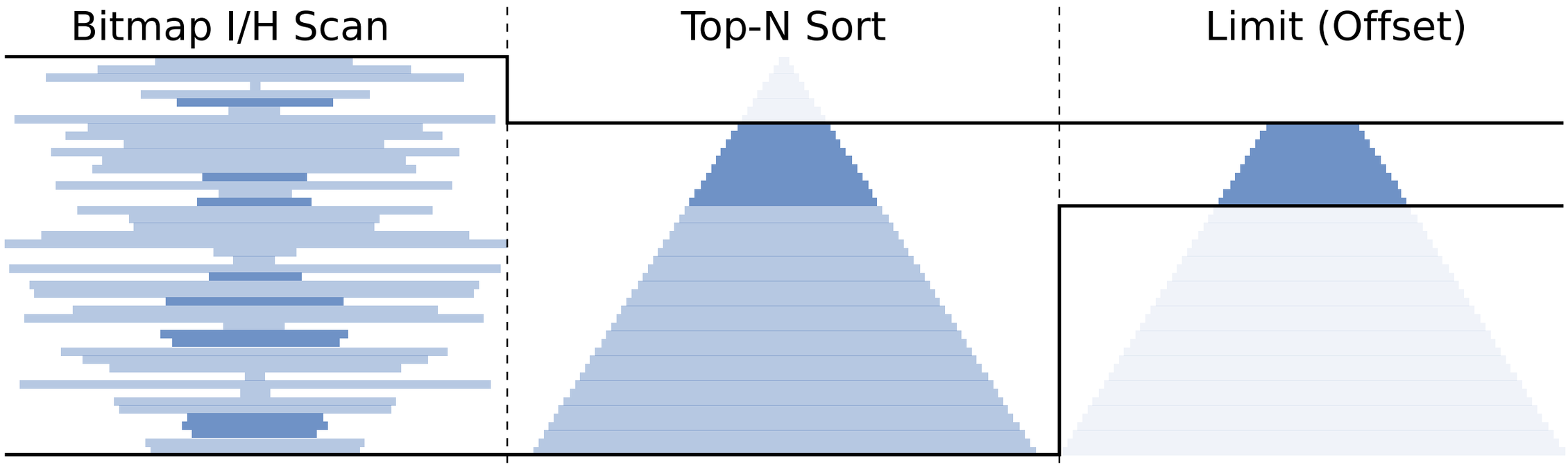

```
Limit (actual rows=10)
-> Sort (actual rows=20)
    Sort Method: top-N heapsort  Memory: 19kB
    ->  Bitmap Heap Scan (actual rows=10000)
        Recheck Cond: (topic = 1234)
        -> Bitmap Index Scan (actual rows=10000)
            Index Cond: (topic = 1234)
```

Worst Case: Kein Index für `order by`



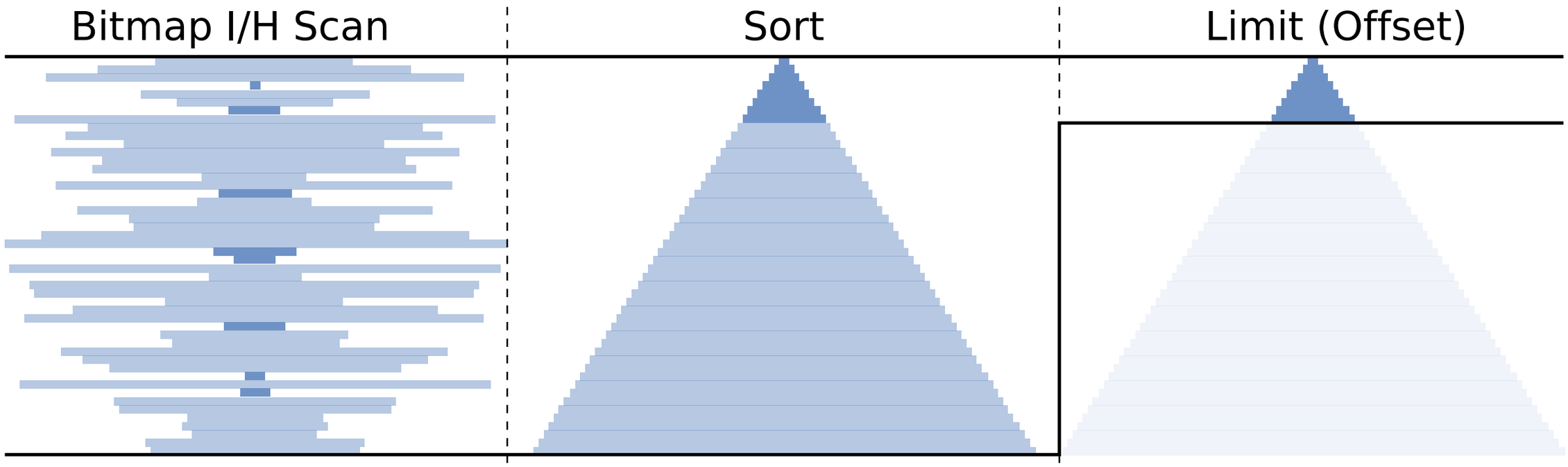

```
Limit (actual rows=10)
-> Sort (actual rows=30)
   Sort Method: top-N heapsort  Memory: 20kB
   ->  Bitmap Heap Scan (actual rows=10000)
       Recheck Cond: (topic = 1234)
       -> Bitmap Index Scan (actual rows=10000)
          Index Cond: (topic = 1234)
```

Worst Case: Kein Index für `order by`




```
Limit (actual rows=10)
-> Sort (actual rows=40)
   Sort Method: top-N heapsort  Memory: 22kB
   ->  Bitmap Heap Scan (actual rows=10000)
       Recheck Cond: (topic = 1234)
       -> Bitmap Index Scan (actual rows=10000)
          Index Cond: (topic = 1234)
```

Worst Case: Kein Index für `order by`

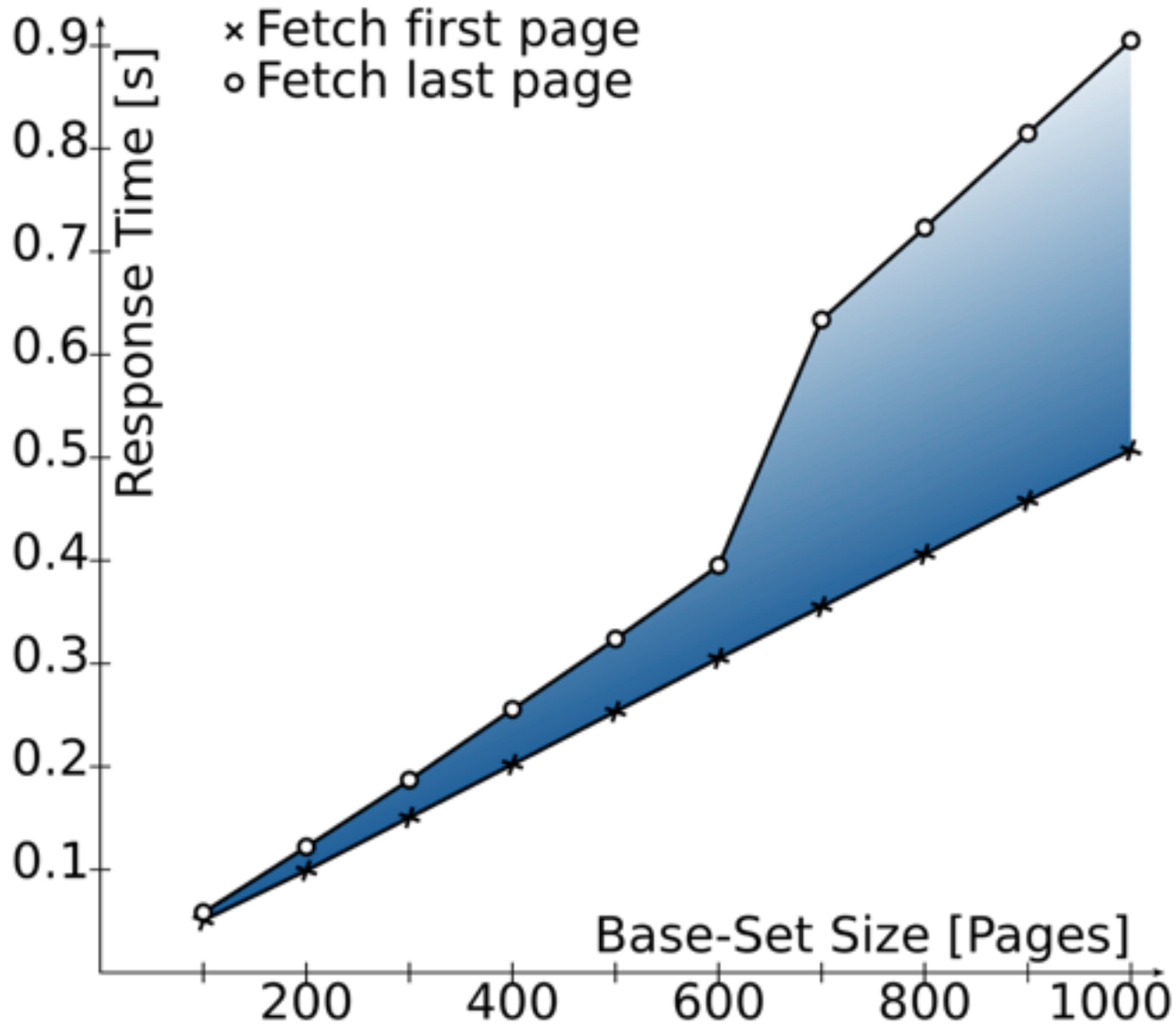



```
Limit (actual rows=10)
-> Sort (actual rows=10000)
   Sort Method: external merge  Disk: 1200kB
   ->  Bitmap Heap Scan (actual rows=10000)
       Recheck Cond: (topic = 1234)
       -> Bitmap Index Scan (actual rows=10000)
          Index Cond: (topic = 1234)
```

Worst Case: Kein Index für `order by`

Das Sortieren kann zum Bottleneck werden, wenn man weit nach hinten blättert.

Das Laden der letzten Seite kann deutlich länger dauern als das Laden der ersten Seite.



Verbesserung 1: order by indizieren

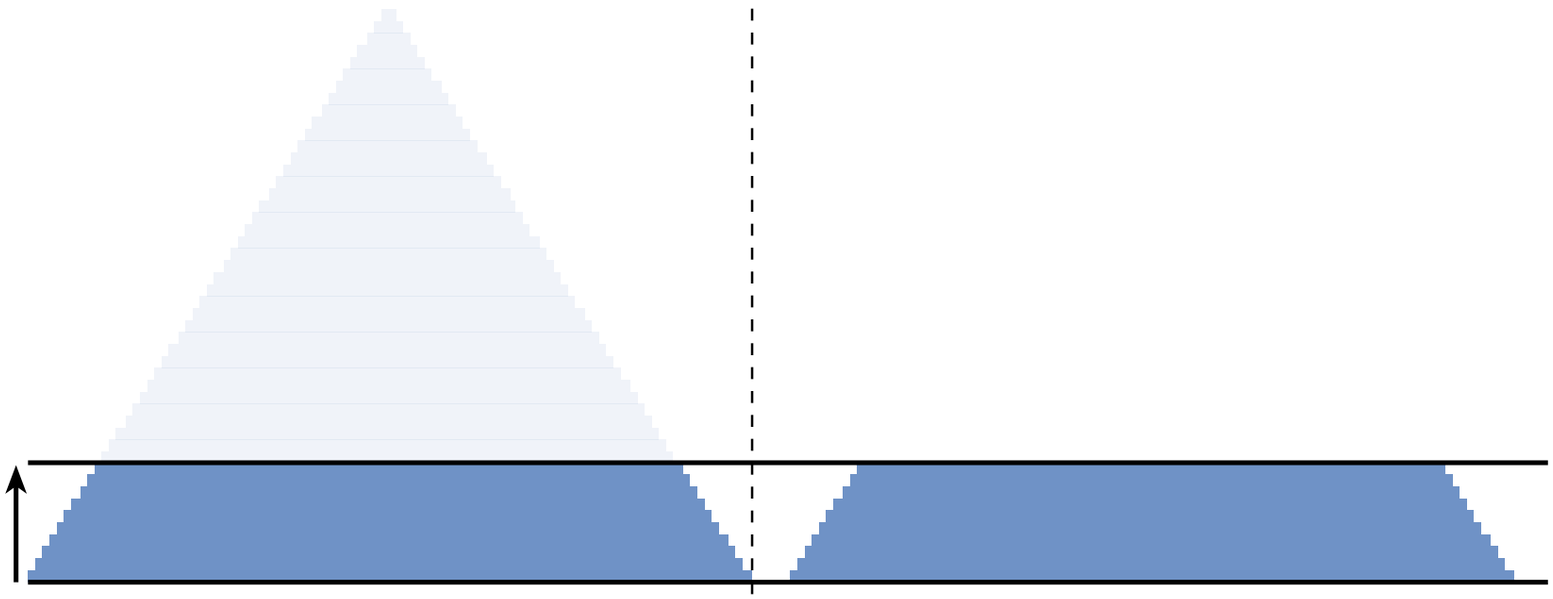
```
select *
  from news
 where topic = 1234
 order by date desc, id desc
offset 10
 limit 10;
```

```
create index .. on news (topic, date, id);
```

Ein Index der die `where`-Klausel und die `order by`-Klausel abdeckt.

Verbesserung 1: order by indizieren

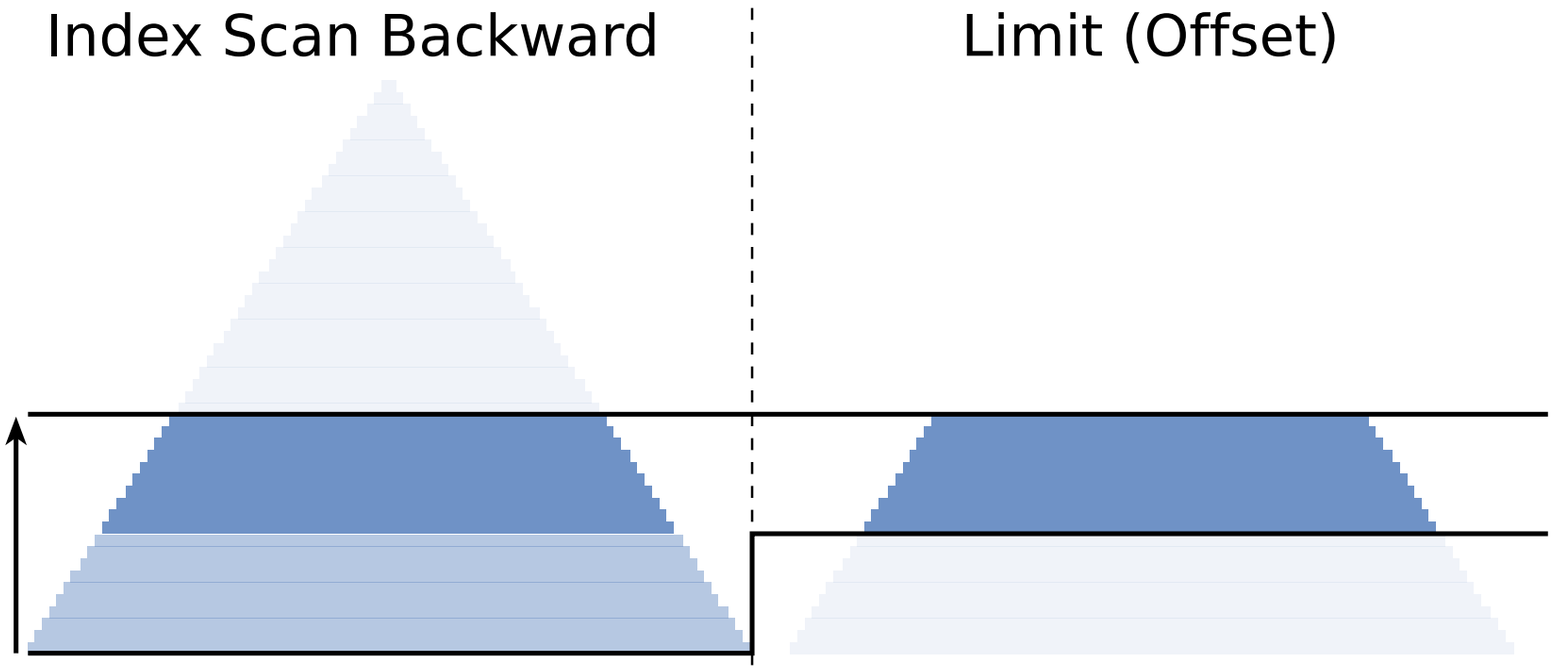
Index Scan Backward



Limit

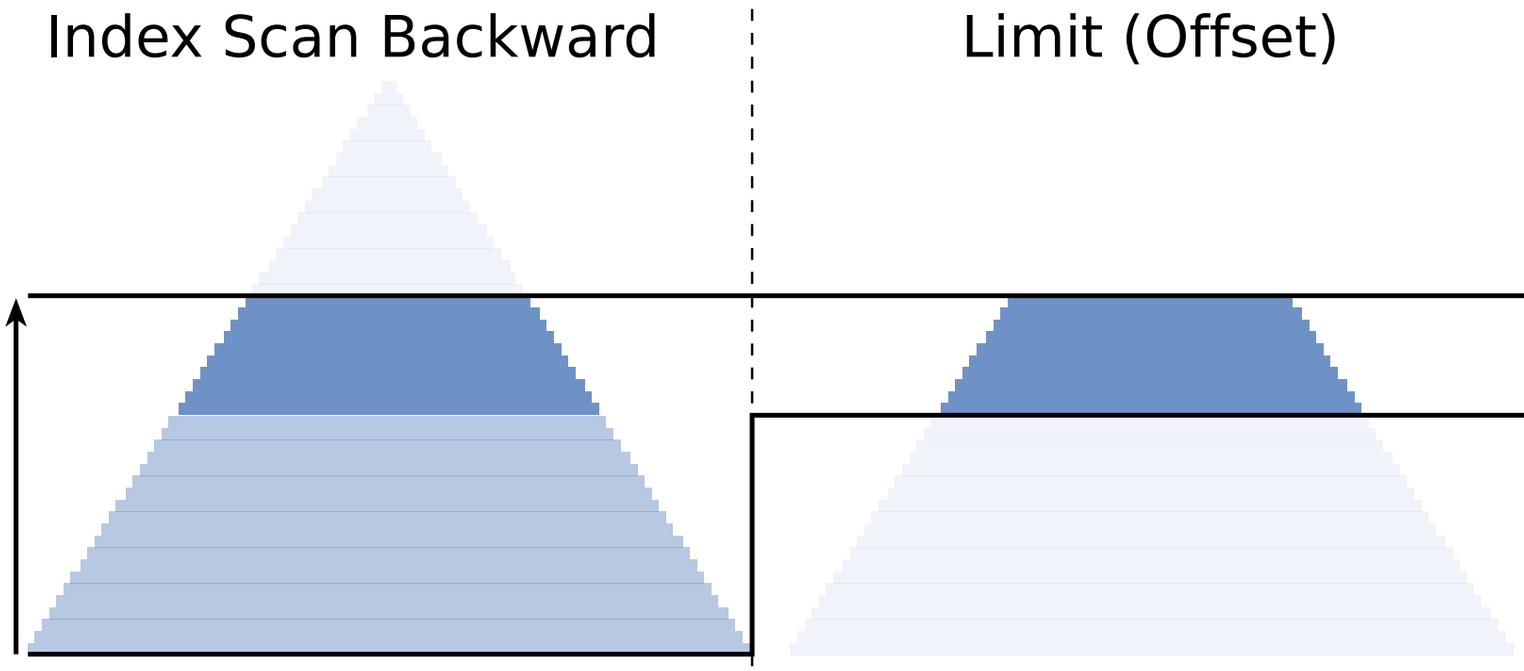
```
Limit (actual rows=10)
-> Index Scan Backward (actual rows=10)
   Index Cond: (topic = 0)
```

Verbesserung 1: order by indizieren



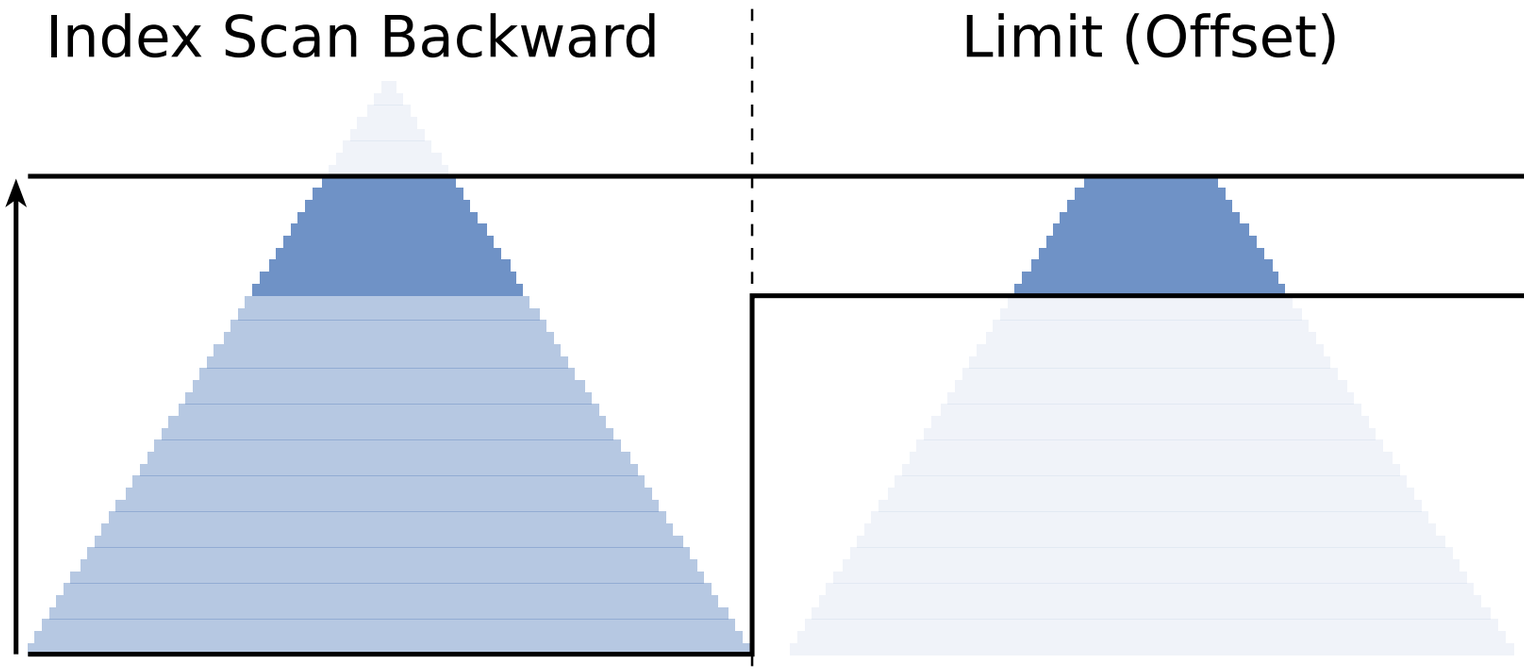

```
Limit (actual rows=10)
-> Index Scan Backward (actual rows=20)
   Index Cond: (topic = 0)
```

Verbesserung 1: order by indizieren



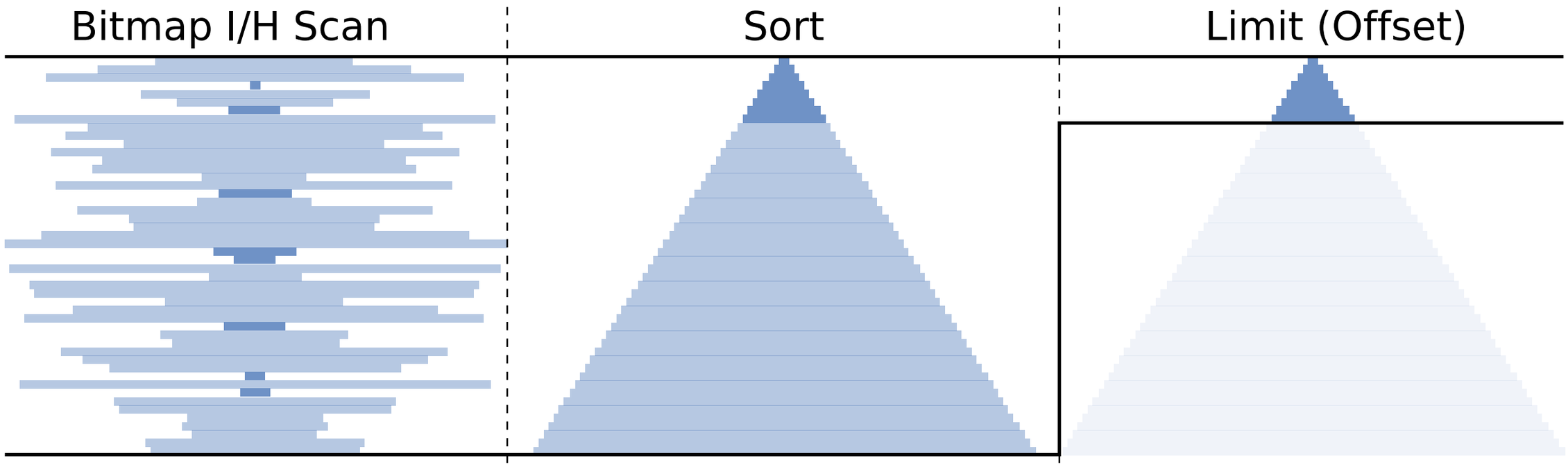

```
Limit (actual rows=10)
-> Index Scan Backward (actual rows=30)
   Index Cond: (topic = 0)
```

Verbesserung 1: order by indizieren




```
Limit (actual rows=10)
-> Index Scan Backward (actual rows=40)
   Index Cond: (topic = 0)
```

Verbesserung 1: order by indizieren

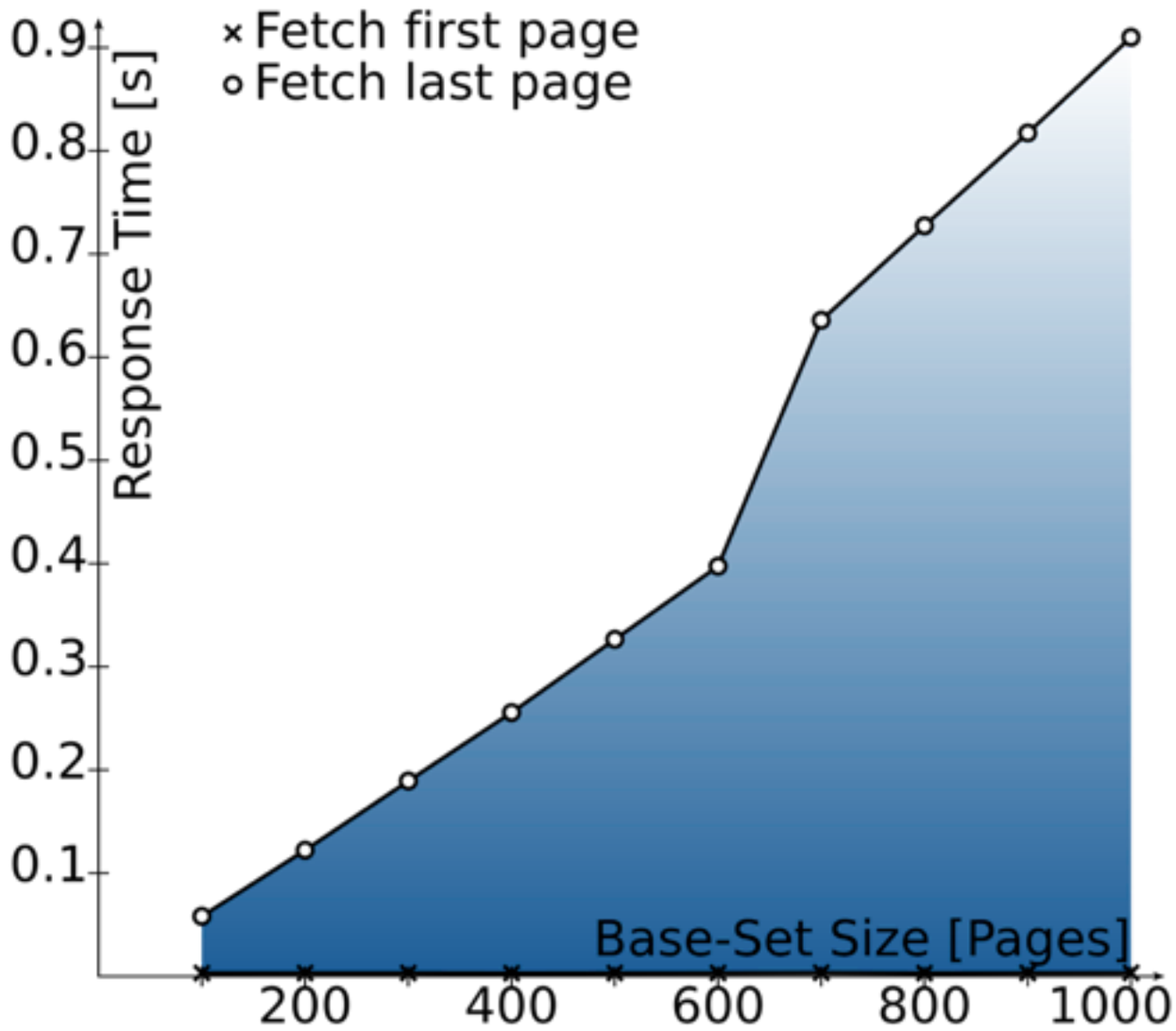



```
Limit (actual rows=10)
-> Sort (actual rows=10000)
   Sort Method: external merge  Disk: 1200kB
   ->  Bitmap Heap Scan (actual rows=10000)
       Recheck Cond: (topic = 1234)
       -> Bitmap Index Scan (actual rows=10000)
          Index Cond: (topic = 1234)
```

Verbesserung 1: order by indizieren

Das Laden der ersten Seite ist schnell—unabhängig von der Größe der Basis-Menge.

Die nächste Seite lädt auch schneller. Wenn man weiter blättert nimmt PostgreSQL aber wieder einen Bitmap Index Scan.



Das können wir besser!

Greife nichts an das du nicht brauchst!

Verbesserung 2: Die Seek Methode

Anstatt `offset` verwenden wir einen `where`-Filter um die Zeilen der vorherigen Seiten auszublenden.

```
select *
  from news
 where topic = 1234
   and (date, id) < (prev_date, prev_id)
 order by date desc, id desc
 limit 10;
```

Selektiert nur Zeilen “bevor” (=früheres Datum) der letzten Zeile der vorhierigen Seite.

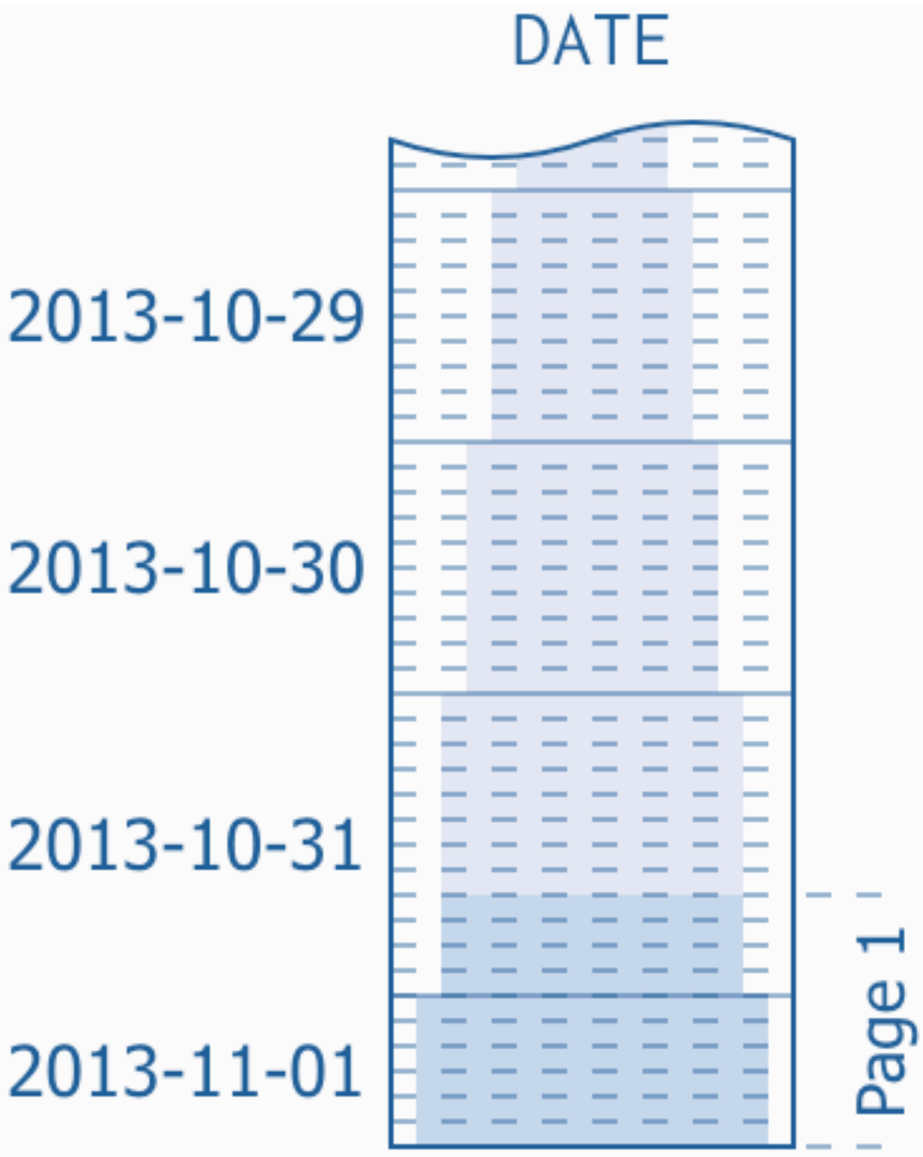
Benötigt unbedingt eine eindeutige Reihenfolge.

Exkurs: Row Values

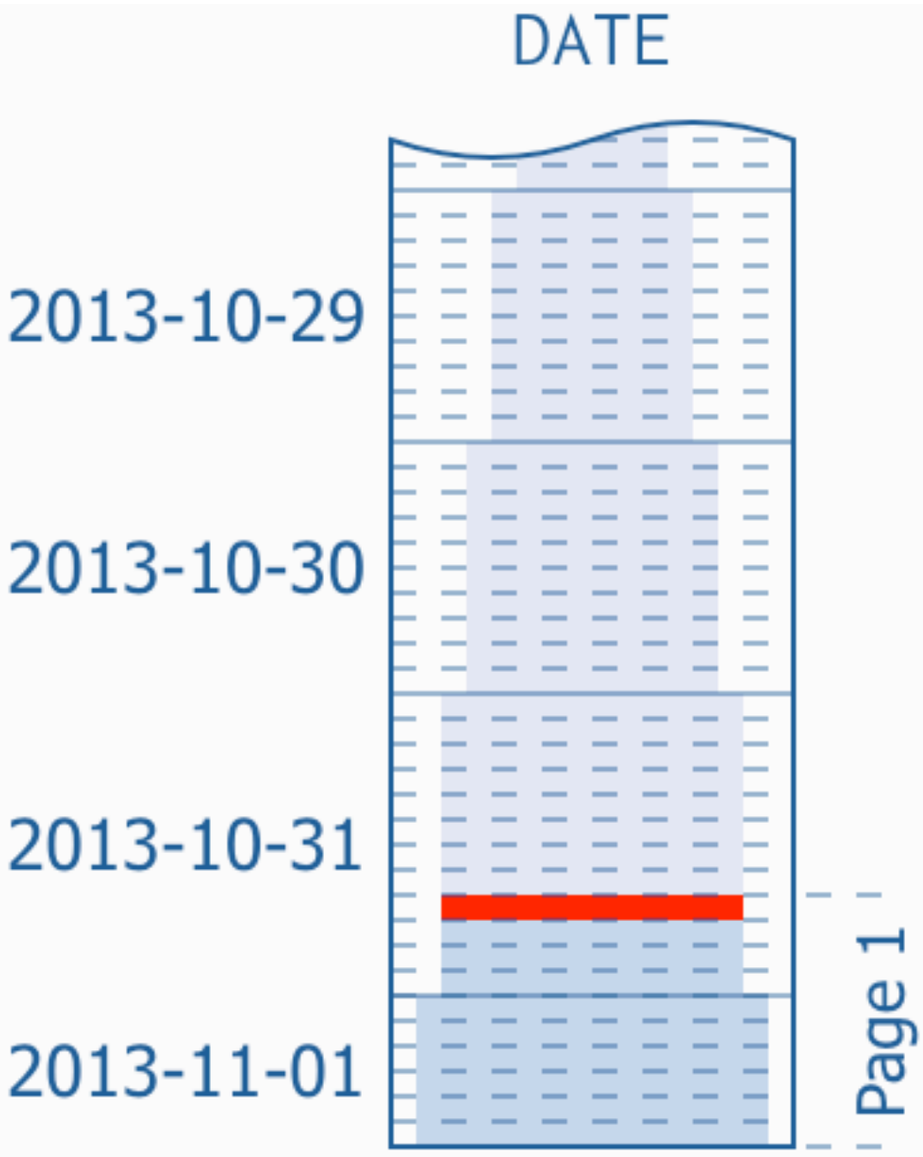
Neben einfachen, skalaren Werten kennt SQL auch “row values”

- Schon ewig im SQL standard (SQL-92)
- Alle Vergleichsoperatoren sind definiert:
  - Z.B.: (x, y) > (a, b) ist nur wahr wenn
    (x > a or (x=a and y>b))
  - auf Deutsch: wenn (x,y) vor (a,b) sortiert wird
- Hervorragender PostgreSQL support seit 8.0!

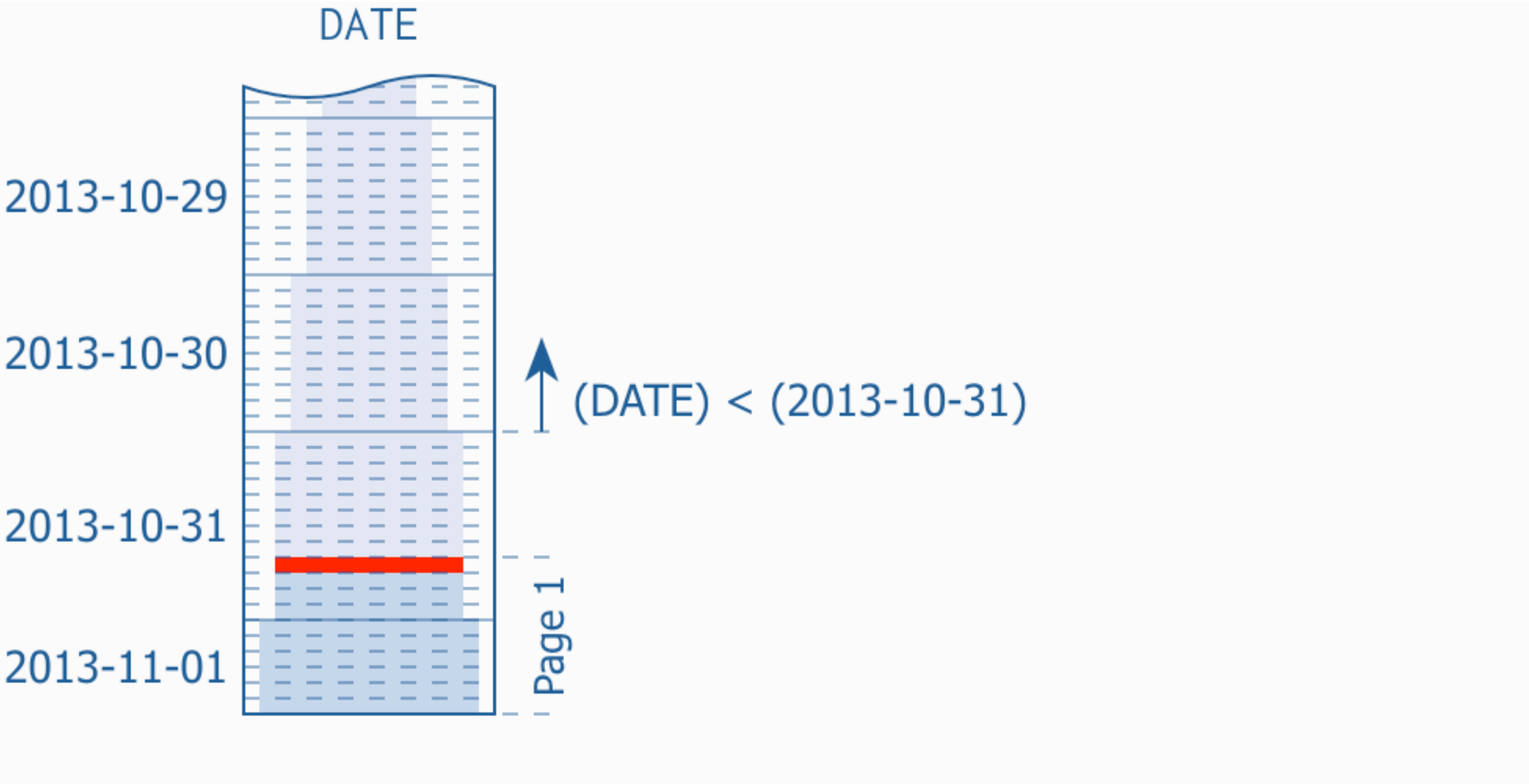
Wie Funktonieren Row-Values?



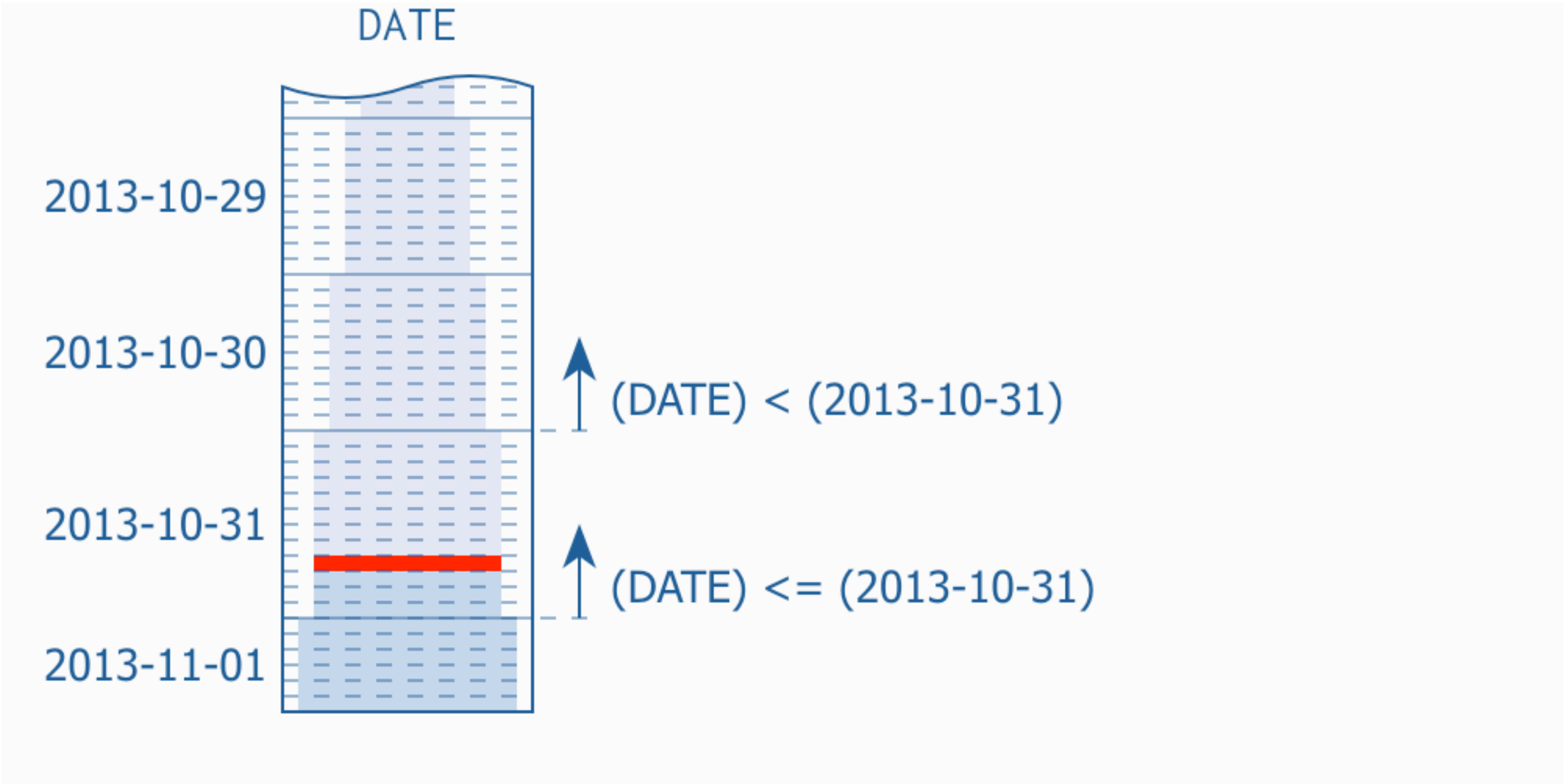
Wie Funktonieren Row-Values?



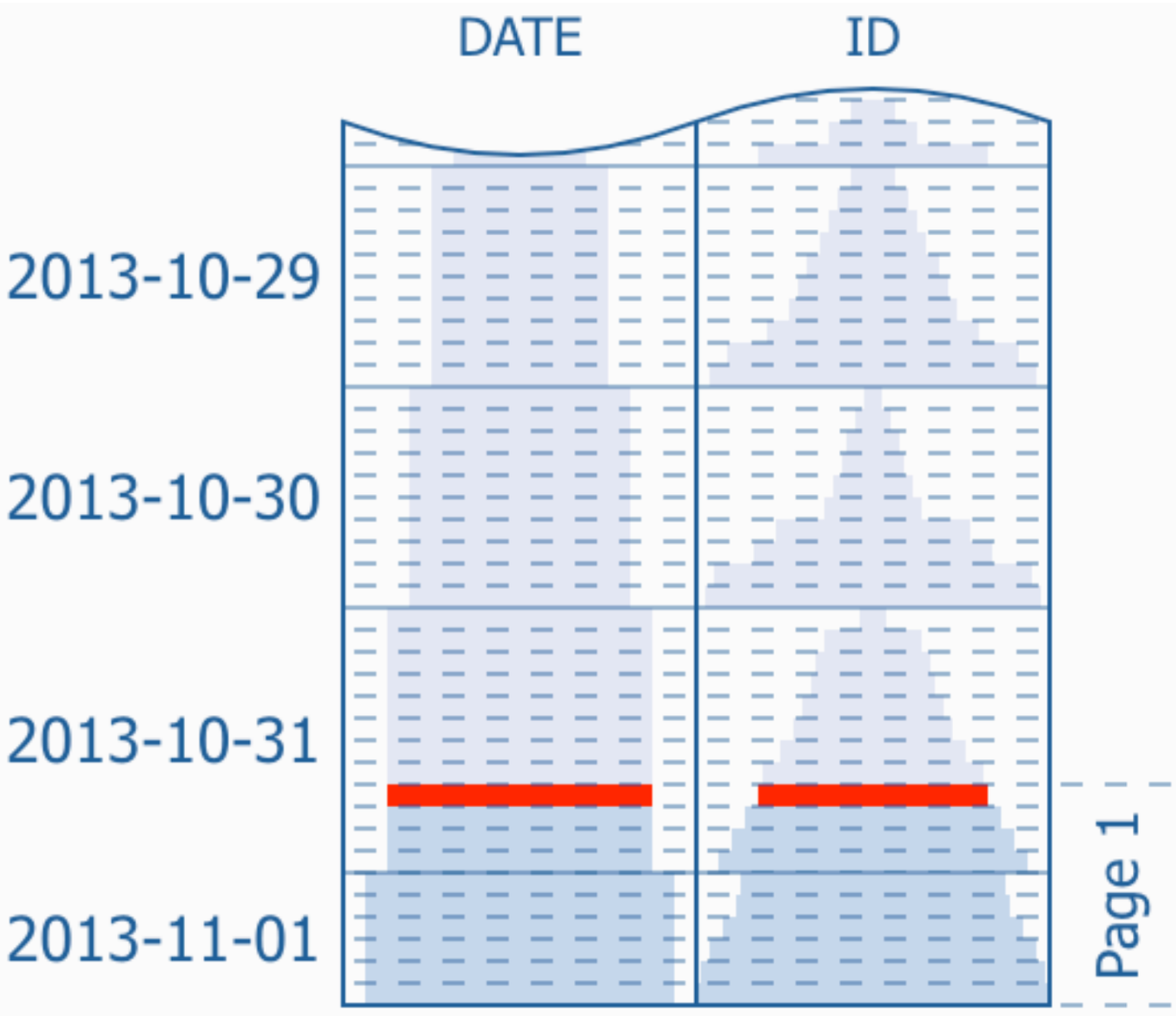
Wie Funktonieren Row-Values?



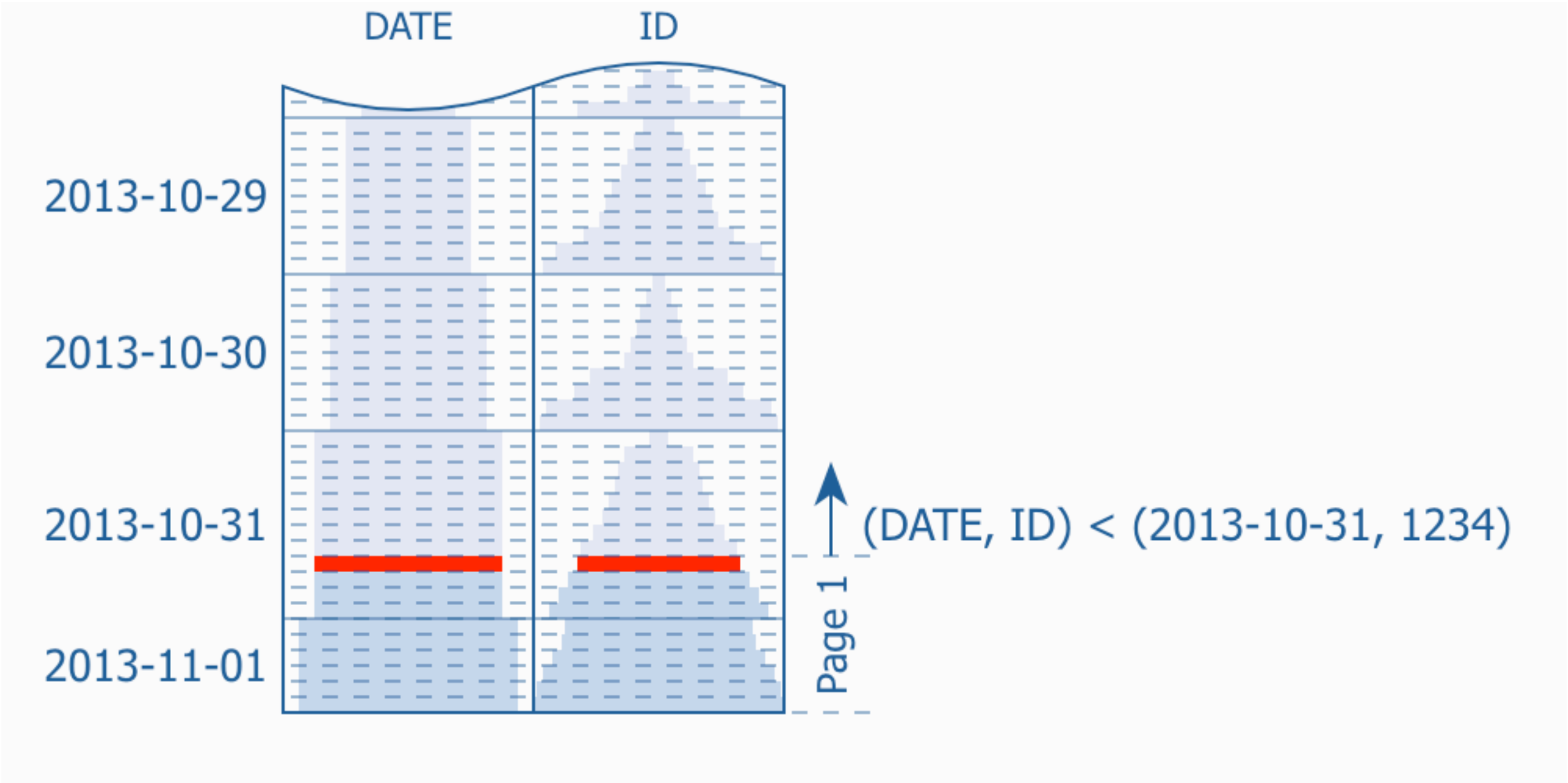
Wie Funktonieren Row-Values?



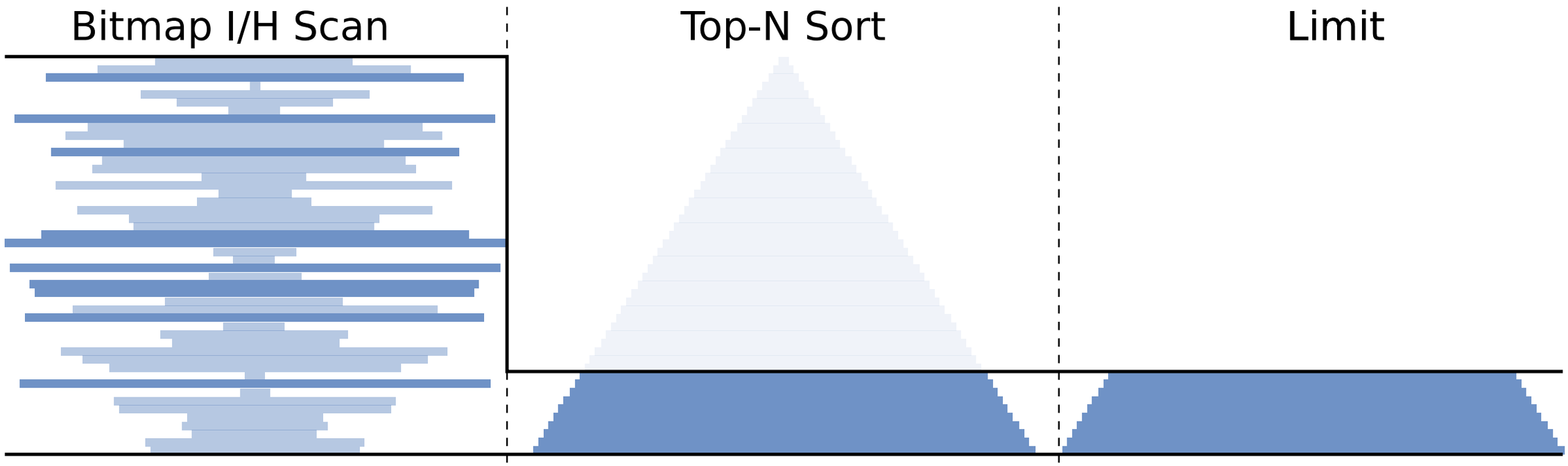
Wie Funktonieren Row-Values?



Wie Funktonieren Row-Values?

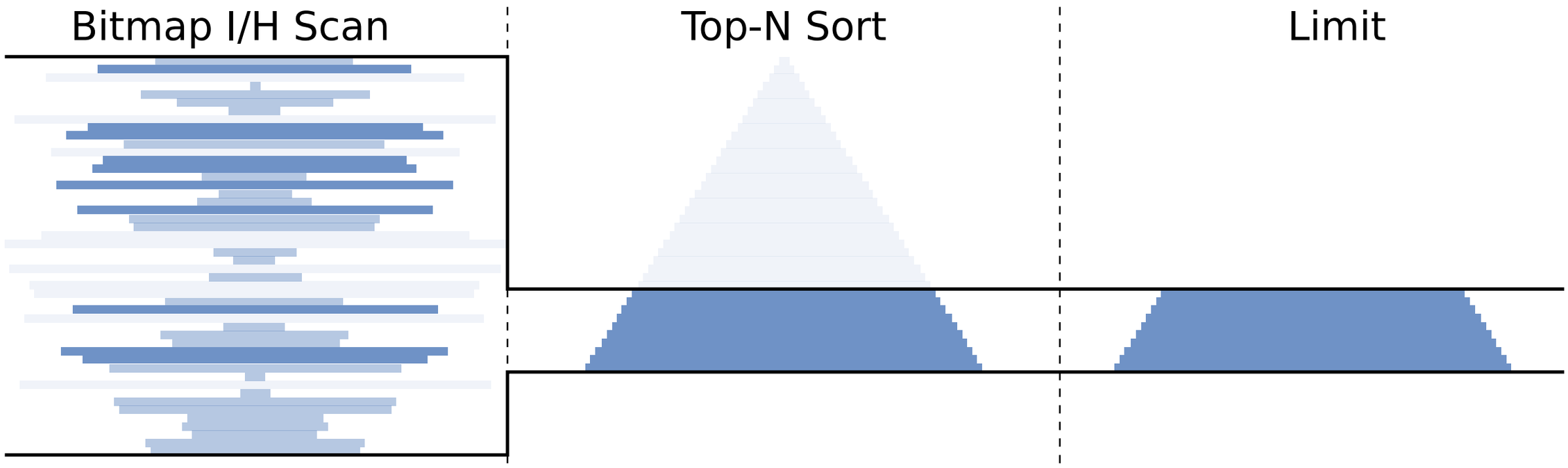


Seek-Methode ohne Index für order by



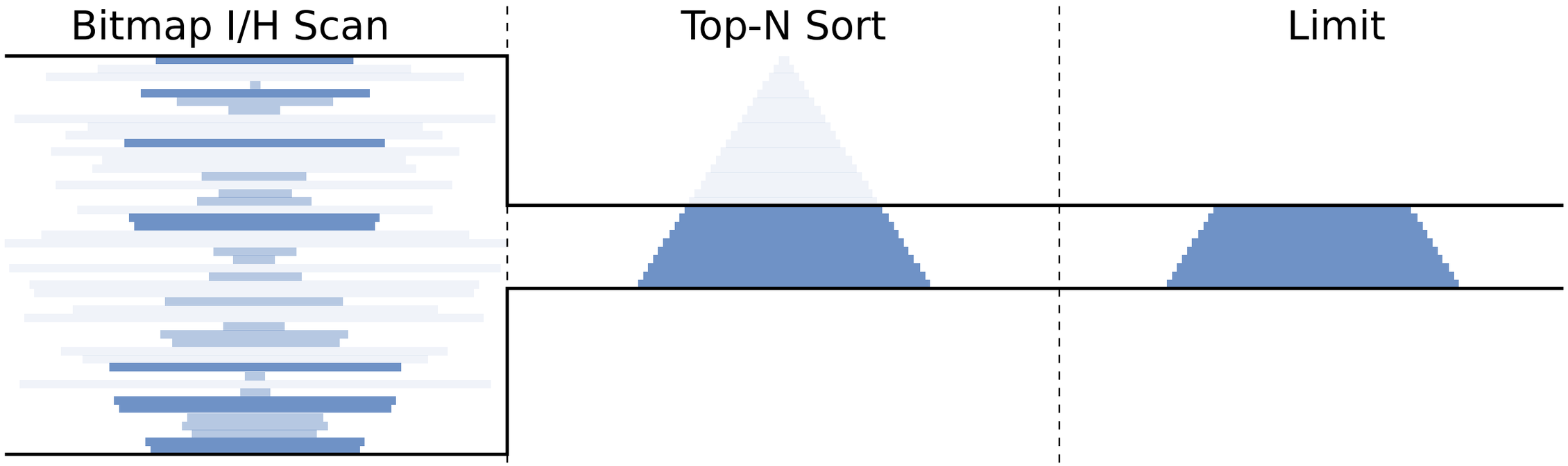

```
Limit (actual rows=10)
-> Sort (actual rows=10)
   Sort Method: top-N heapsort  Memory: 18kB
   -> Bitmap Heap Scan (rows=10000)
      Recheck Cond: (topic = 1234)
      -> Bitmap Index Scan (rows=10000)
         Index Cond: (topic = 1234)
```

Seek-Methode ohne Index für order by



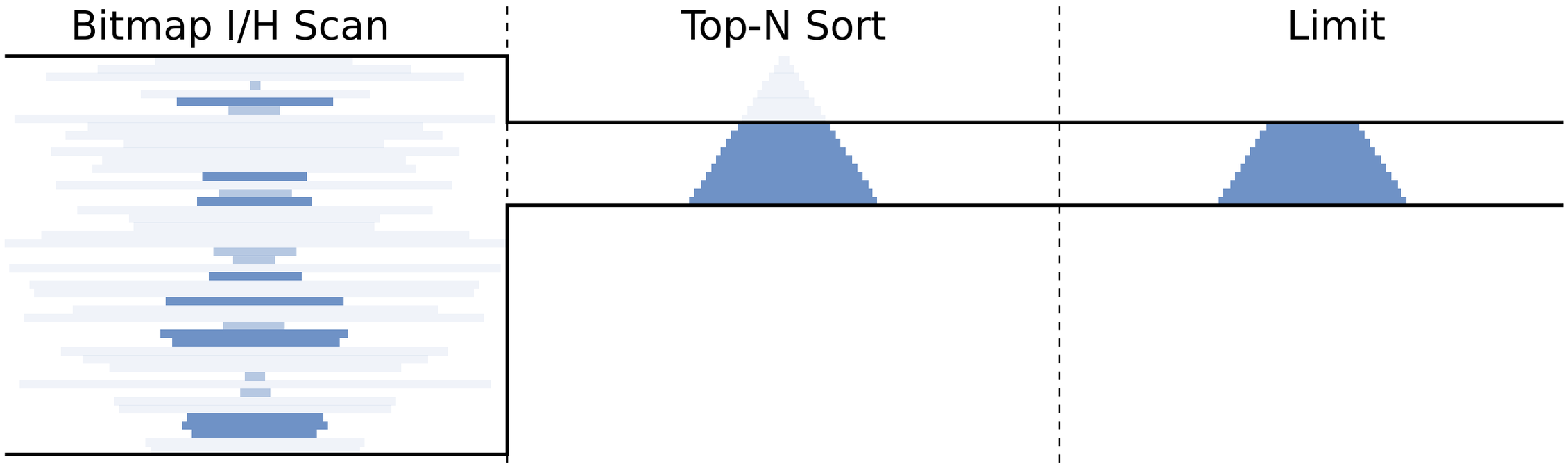

```
Limit (actual rows=10)
 -> Sort (actual rows=10)
    Sort Method: top-N heapsort  Memory: 18kB
    -> Bitmap Heap Scan (actual rows=9990)
       Rows Removed by Filter: 10 (new in 9.2)
       -> Bitmap Index Scan (actual rows=10000)
          Index Cond: (topic = 1234)
```

Seek-Methode ohne Index für order by



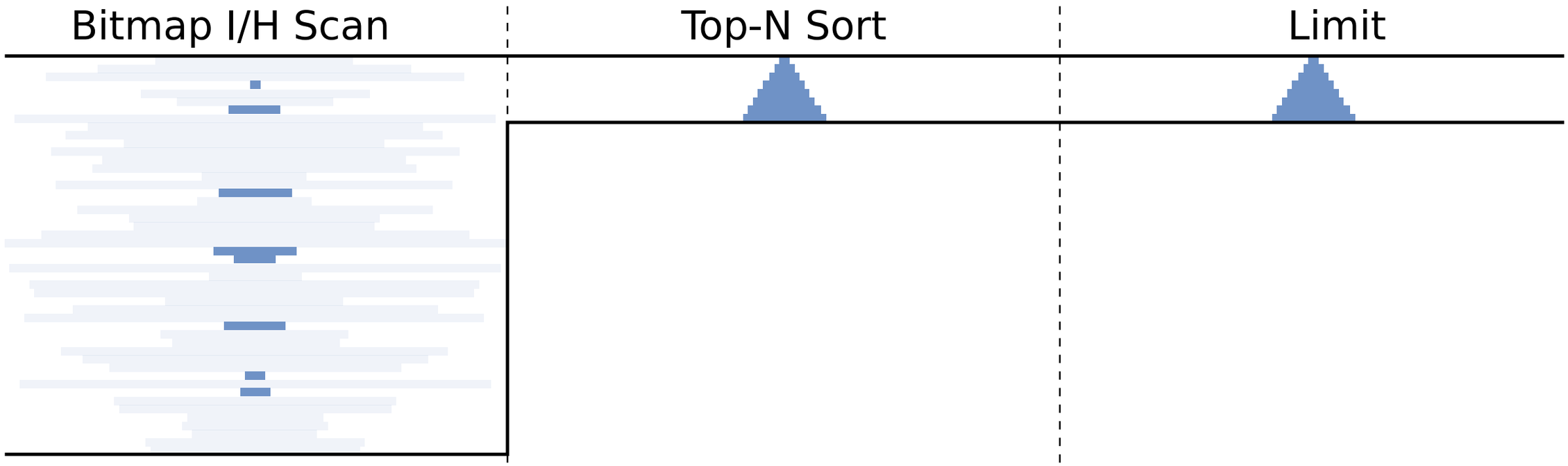

```
Limit (actual rows=10)
 -> Sort (actual rows=10)
    Sort Method: top-N heapsort  Memory: 18kB
    -> Bitmap Heap Scan (actual rows=9980)
       Rows Removed by Filter: 20 (new in 9.2)
       -> Bitmap Index Scan (actual rows=10000)
          Index Cond: (topic = 1234)
```

Seek-Methode ohne Index für order by




```
Limit (actual rows=10)
 -> Sort (actual rows=10)
    Sort Method: top-N heapsort  Memory: 18kB
    -> Bitmap Heap Scan (actual rows=9970)
       Rows Removed by Filter: 30 (new in 9.2)
       -> Bitmap Index Scan (actual rows=10000)
          Index Cond: (topic = 1234)
```

Seek-Methode ohne Index für order by

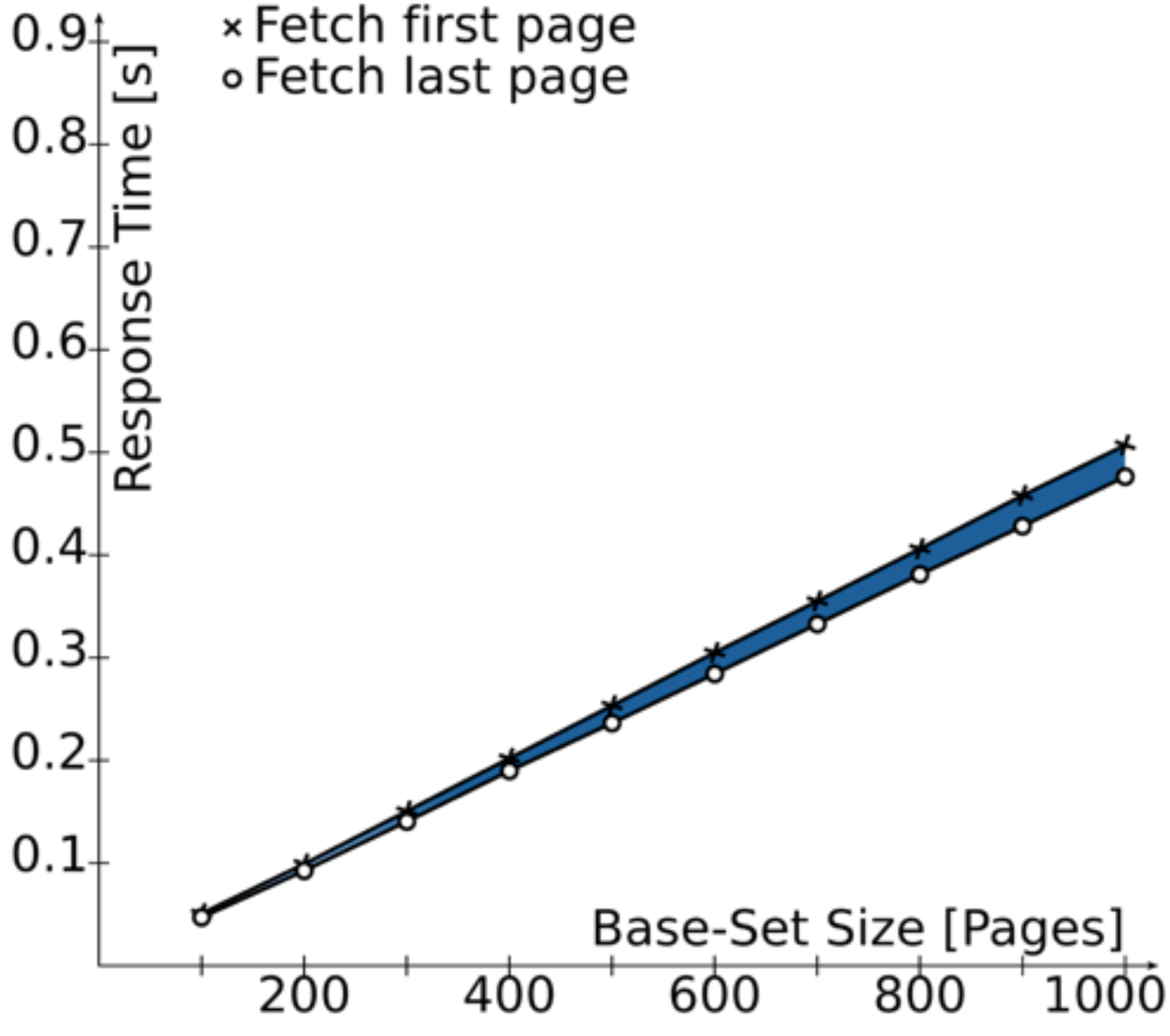



```
Limit (actual rows=10)
 -> Sort (actual rows=10)
    Sort Method: top-N heapsort  Memory: 18kB
    -> Bitmap Heap Scan (actual rows=10)
       Rows Removed by Filter: 9990 (new in 9.2)
       -> Bitmap Index Scan (actual rows=10000)
          Index Cond: (topic = 1234)
```

Seek-Methode ohne Index für order by

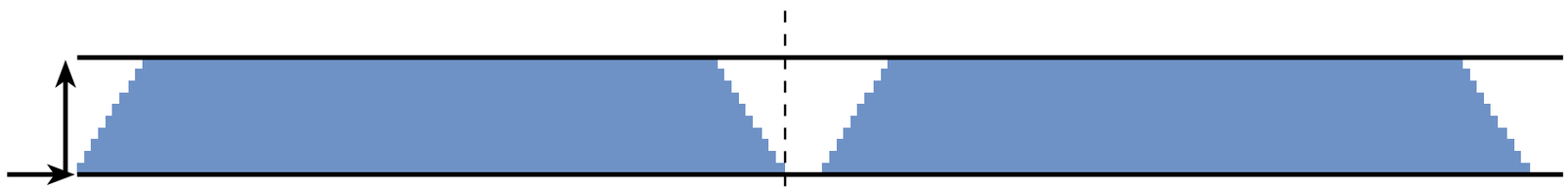
Muss immer die komplette Basis-Menge laden, das Top-N Sort muss aber nur 10 Zeilen im Speicher halten.

Die Antwortzeit bleibt konstant, auch wenn man ganz nach hinten blättert. Speicherbedarf bleibt auch gering.



Seek-Methode mit Index für order by

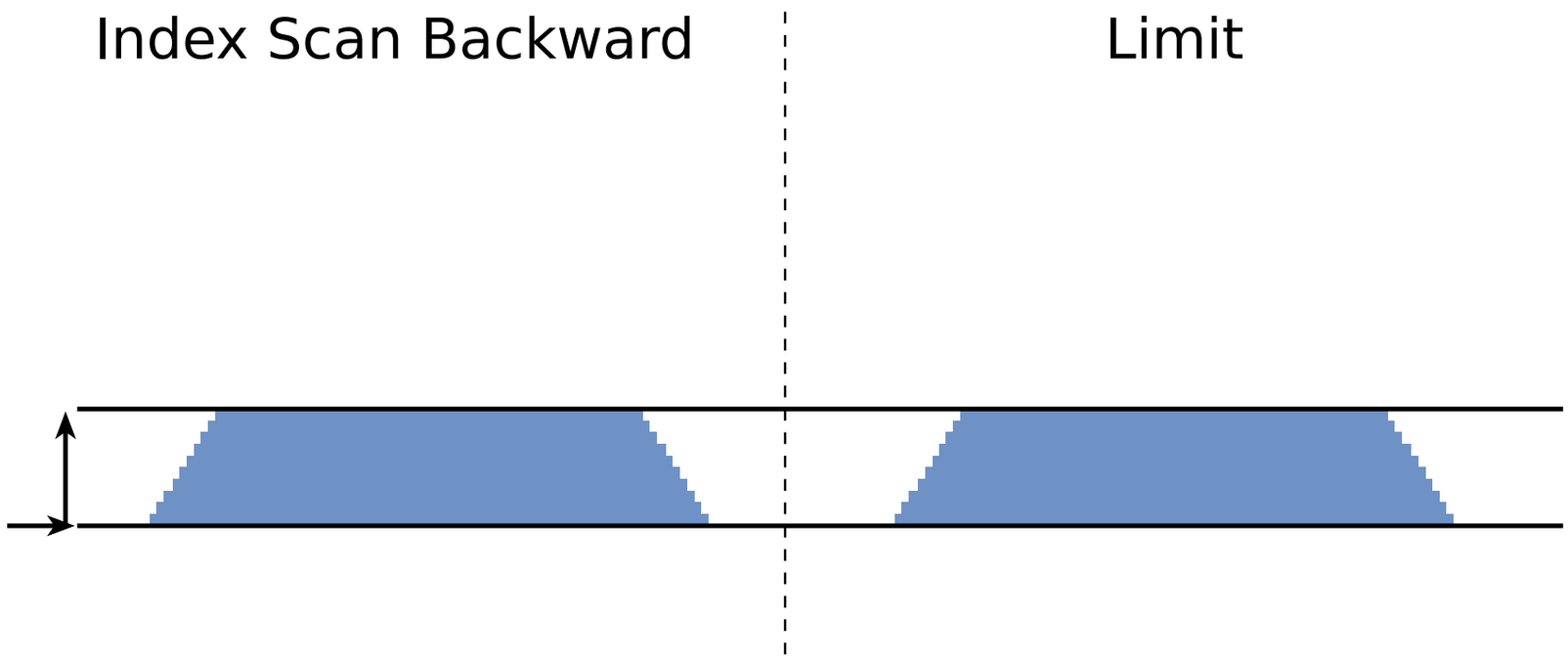
Index Scan Backward



Limit

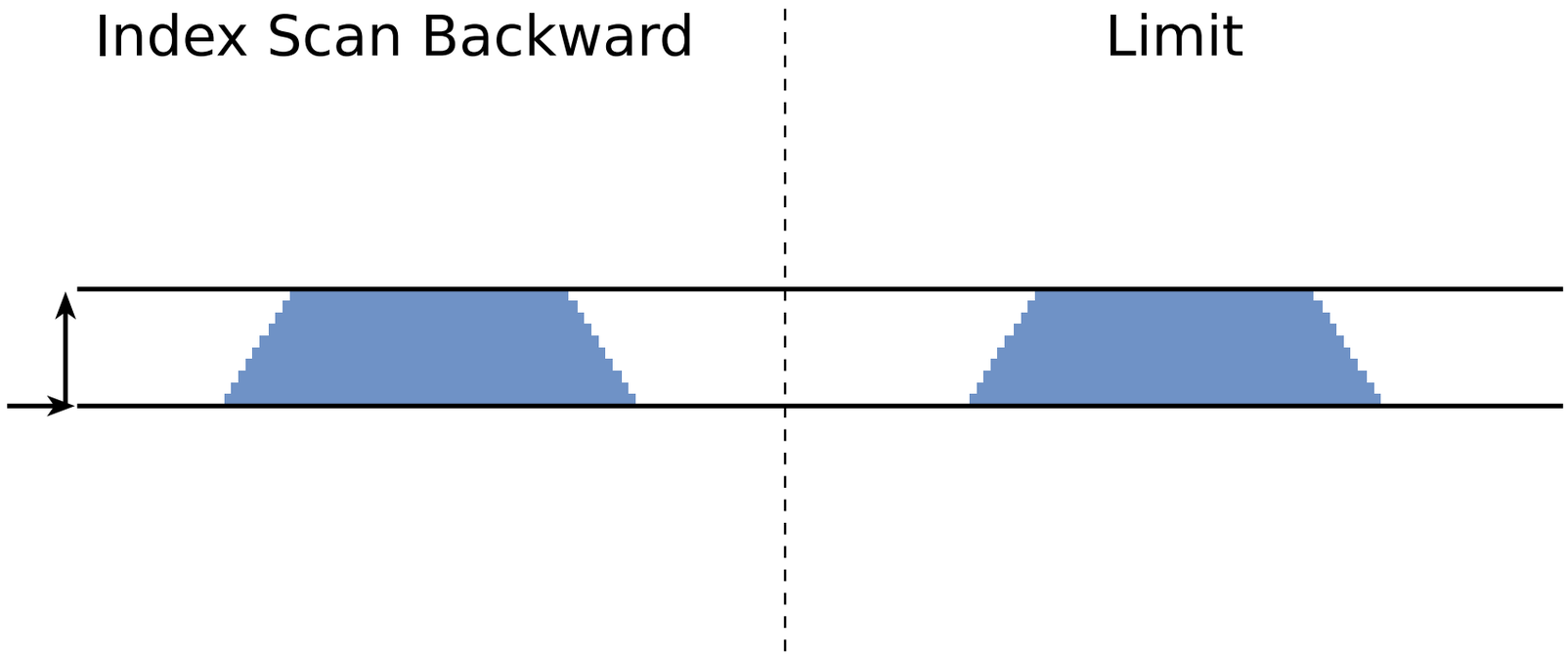
```
Limit (actual rows=10)
-> Index Scan Backward (actual rows=10)
   Index Cond: (topic = 1234)
```

Seek-Methode mit Index für order by



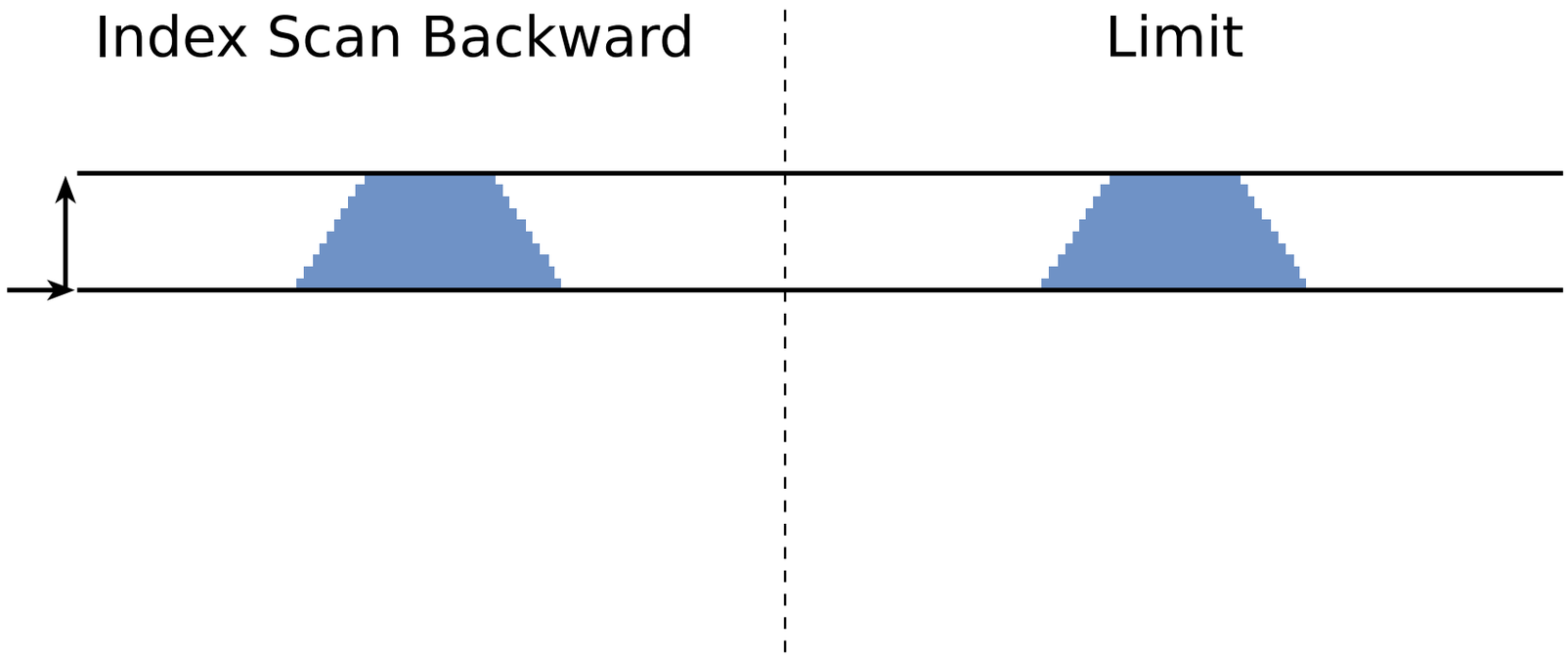

```
Limit (actual rows=10)
-> Index Scan Backward (actual rows=10)
   Index Cond: ((topic = 1234)
            AND (ROW(dt, id) < ROW('...', 23456)))
```

Seek-Methode mit Index für order by



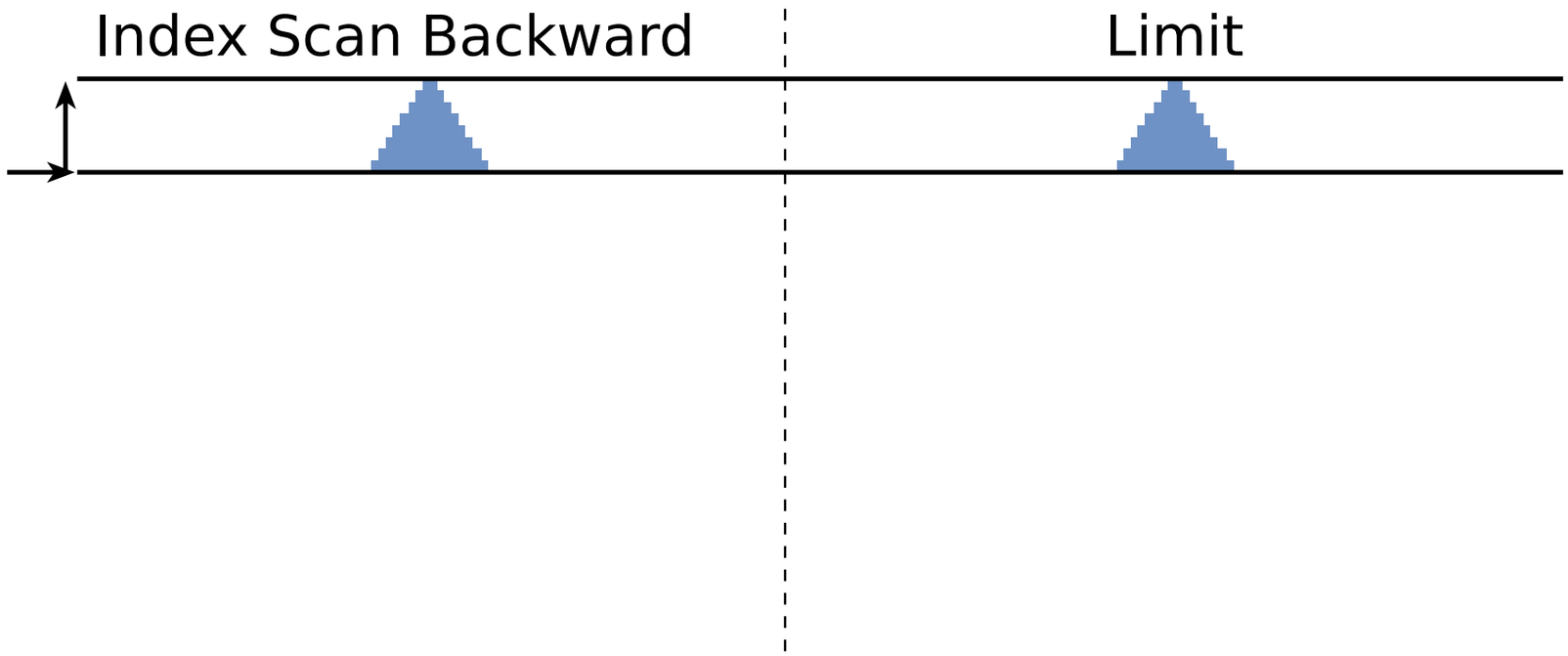

```
Limit (actual rows=10)
-> Index Scan Backward (actual rows=10)
   Index Cond: ((topic = 1234)
               AND (ROW(dt, id) < ROW('...', 34567)))
```

Seek-Methode mit Index für order by




```
Limit (actual rows=10)
-> Index Scan Backward (actual rows=10)
   Index Cond: ((topic = 1234)
            AND (ROW(dt, id) < ROW('...', 45678)))
```

Seek-Methode mit Index für order by

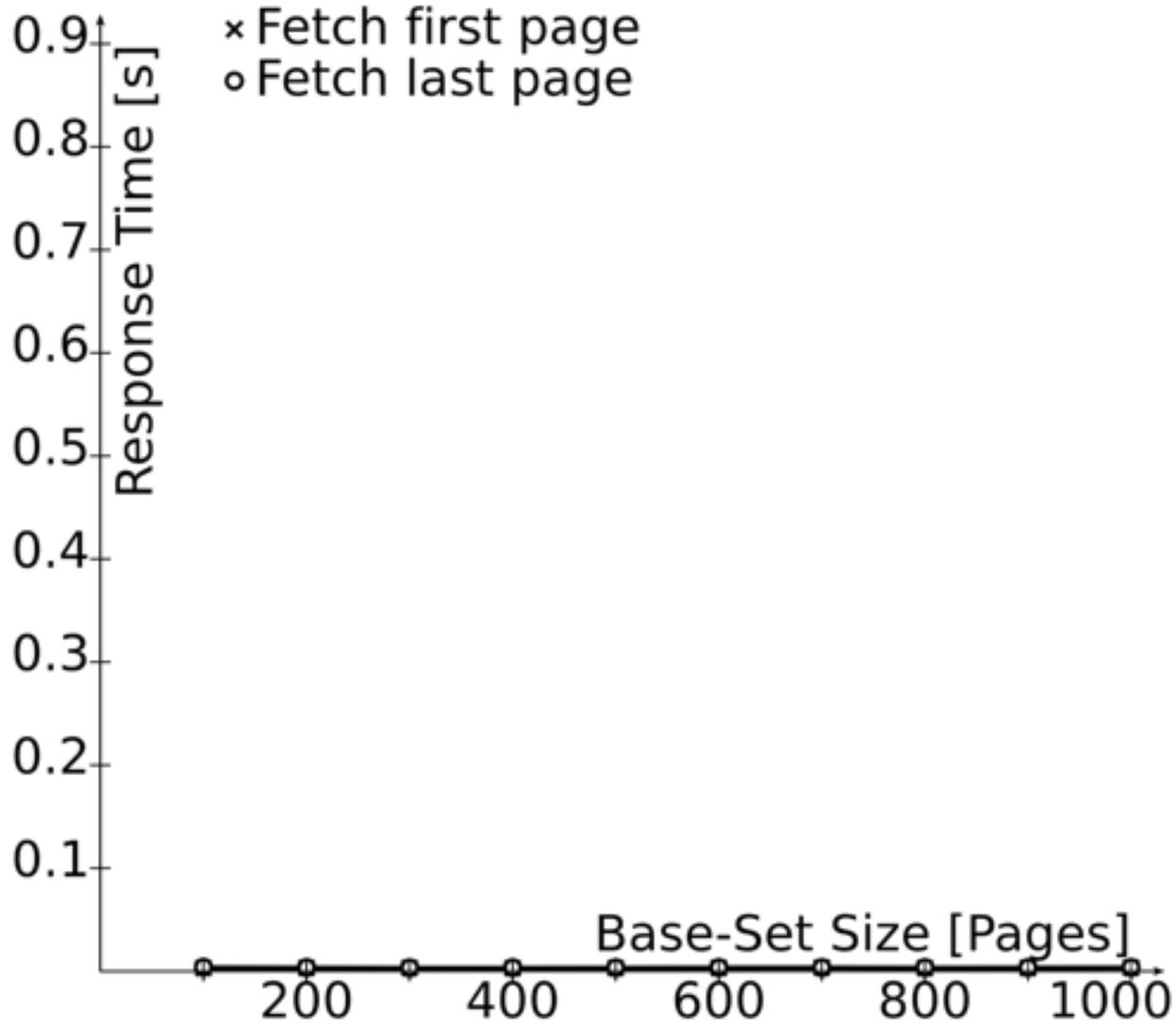



```
Limit (actual rows=10)
-> Index Scan Backward (actual rows=10)
   Index Cond: ((topic = 1234)
            AND (ROW(dt, id) < ROW('...', 56789)))
```

Seek-Methode mit Index für order by

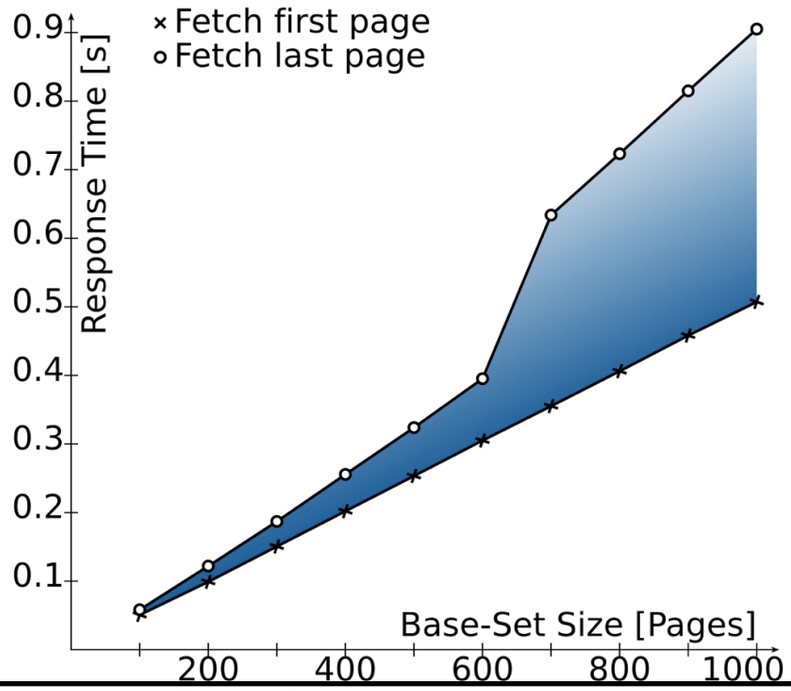
Weiter nach hinten blättern wird nicht langsamer.

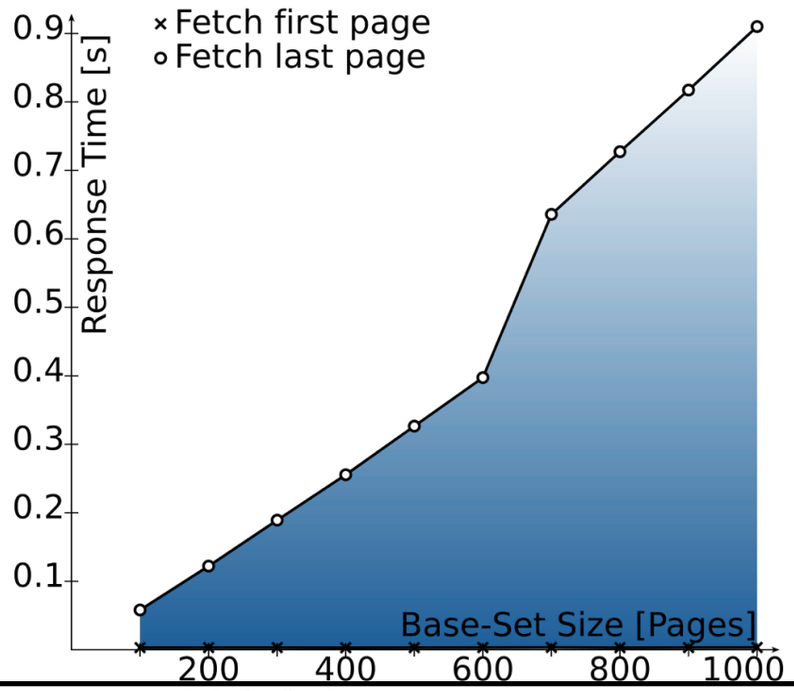
Weder die Größe der Basis-Menge(*) noch die Seitennummer beinflusst die Geschwindigkeit.



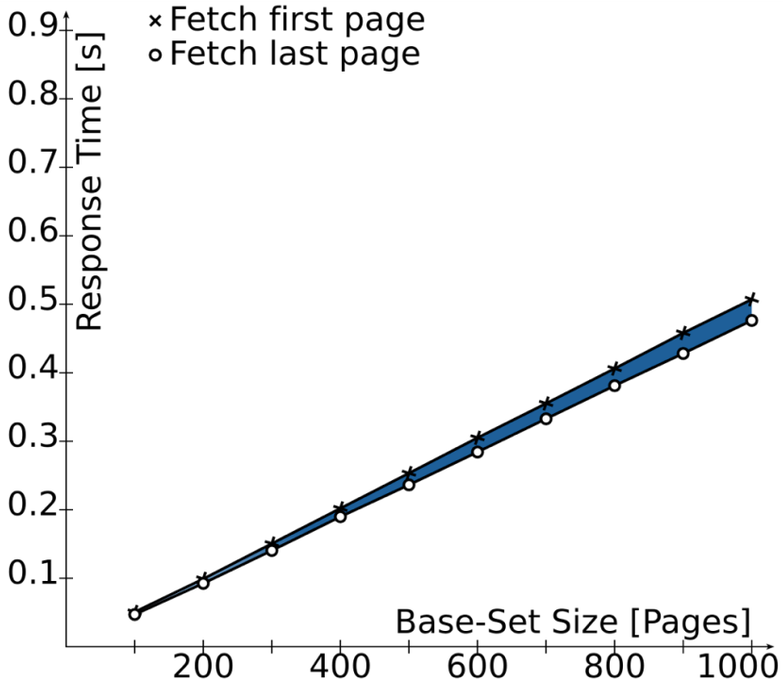
Vergleich

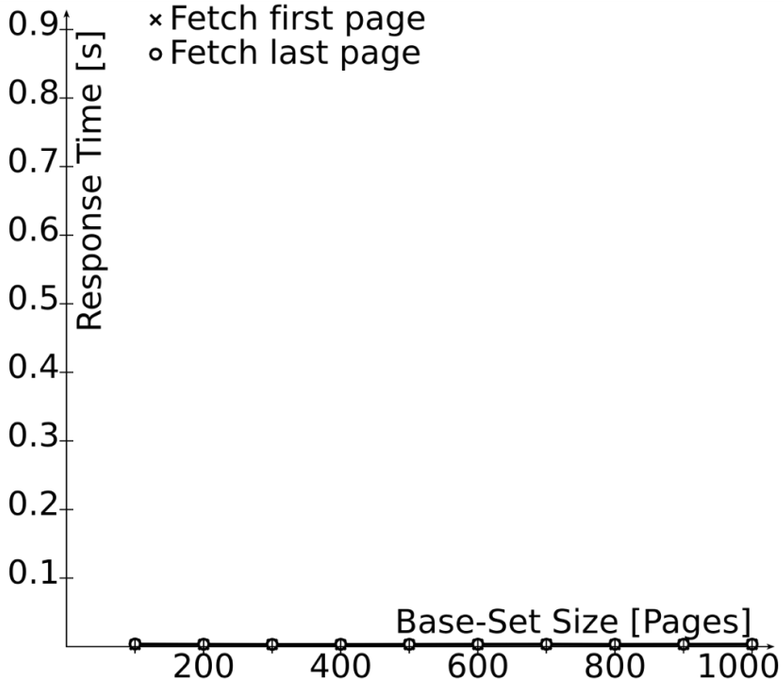
Offset





Seek





Ohne Index für `order by`

Mit Index für `order by`

Zu gut um wahr zu sein?

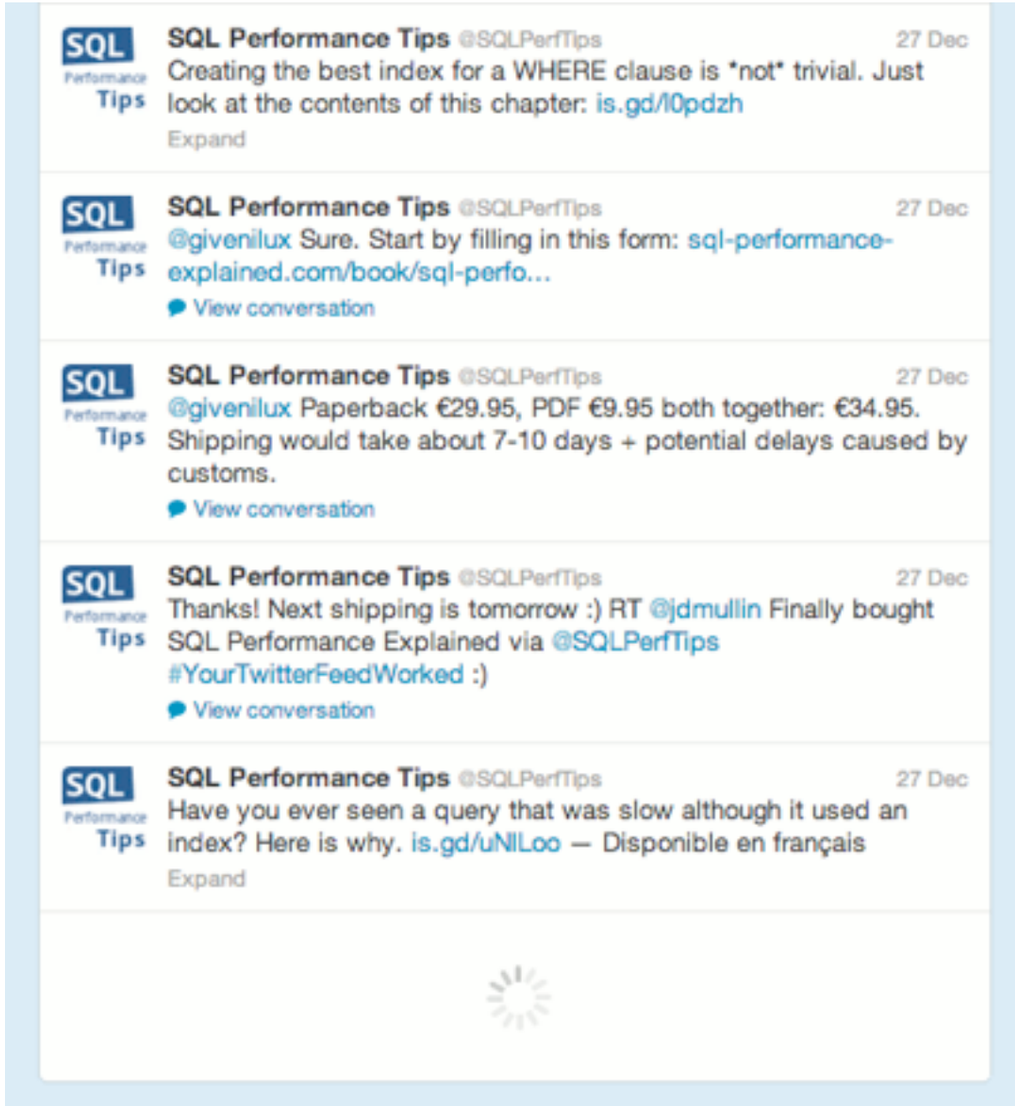
Die Seek-Methode hat einige Einschränkungen:

- Keine direkte Navigation zu beliebigen Seiten
  - Weil man die Werte der vorherigen Seite benötigt
- Richtungswechsel ist möglich, aber umständlich
  - man muss das `order by` und die RV-Bedingungen umdrehen
- Am besten mit guten Row-Values Support
  - Es geht auch ohne, aber weniger elegant und performant
  - Framework support?
    - jOOQ 3.3 ist das erste ORM mit nativem Support der Seek-Methode

Passt gut zum “Unendlichen Listen”

Bei einer “unendlichen” Liste muss man nicht...

- zu beliebigen Seiten springen
  - dafür gibt es keine Knöpfe
- die Richtung wechseln
  - vorherige Seiten sind noch im Browser
- Trefferzahl zeigen
  - Wenn man es doch muss, hat man ohnehin ein Problem!



Passt auch gut zu PostgreSQL

Row Values Support-Matrix

	MySQL	Oracle	PostgreSQL	SQLite	SQL Server
Supported in `where` clause	✓	✓	✓	✗	✗
Ranges supported (`<`,`>`)	✓	✗	✓	✗	✗
Optimal index usage	✗	✓	✓	✗	✗

`order by` Support-Matrix

	MySQL	Oracle	PostgreSQL	SQL Server
Read index backwards	✓	✓	✓	✓
Order by `ASC/DESC`	✓	✓	✓	✓
Index `ASC/DESC`	✗	✓	✓	✓
Order by `NULLS FIRST/LAST`	✗	✓	✓	✗
Default `NULLS` order	First	Last	Last	First
Index `NULLS FIRST/LAST`	✗	✗	✓	✗

Passt auch gut zu PostgreSQL

Row Values Support-Matrix

	MySQL	Oracle	PostgreSQL	SQLite	SQL Server
Supported in `where` clause	✓	✓	✓	✗	✗
Ranges supported (<,>)	✓	✗	✓	✗	✗
Optimal index usage	✗	✓	✓	✗	✗

Popular

Advanced

`order by` Support-Matrix

	MySQL	Oracle	PostgreSQL	SQL Server
Read index backwards	✓	✓	✓	✓
Order by `ASC/DESC`	✓	✓	✓	✓
Index `ASC/DESC`	✗	✓	✓	✓
Order by `NULLS FIRST/LAST`	✗	✓	✓	✗
Default `NULLS` order	First	Last	Last	First
Index `NULLS FIRST/LAST`	✗	✗	✓	✗

Popular

Advanced

Über Markus Winand

Ich tune Entwickler auf
SQL-Performance

Training & co:
winand.at

Geeky blog:
use-the-index-luke.com

Mein Buch:
SQL Performance Explained