Pagination Done the PostgreSQL Way

iStockPhoto/Mitshu

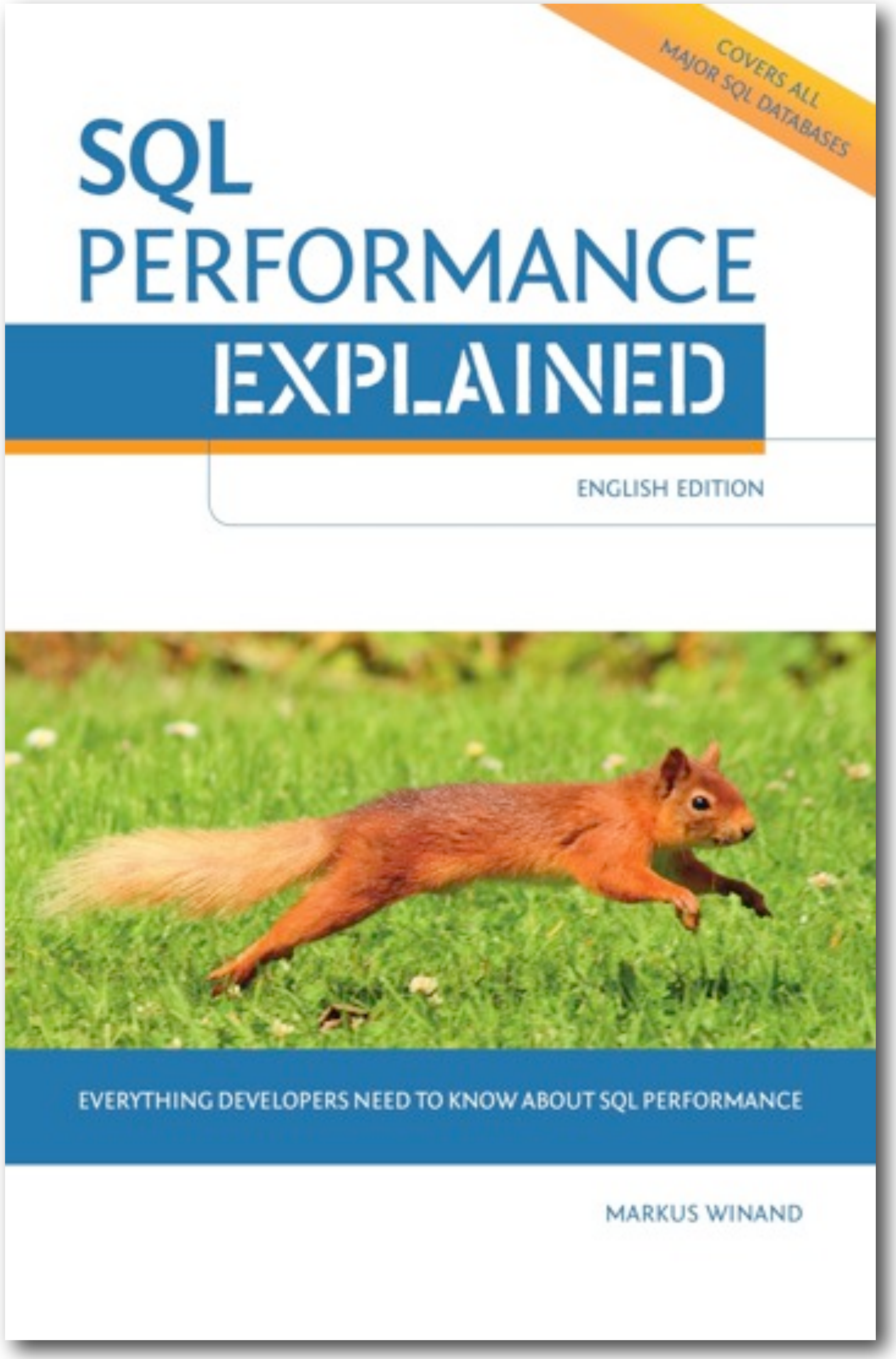
About Markus Winand

Tuning <u>developers</u> to
SQL performance

Training & co:
<u>winand.at</u>

Geeky blog:
<u>use-the-index-luke.com</u>

Author of:
SQL Performance Explained


COVERS ALL MAJOR SQL DATABASES
SQL PERFORMANCE EXPLAINED
ENGLISH EDITION
EVERYTHING DEVELOPERS NEED TO KNOW ABOUT SQL PERFORMANCE
MARKUS WINAND

Note

In this presentation

index means B-tree index.

A Trivial Example

A query to fetch the 10 most recent news:

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

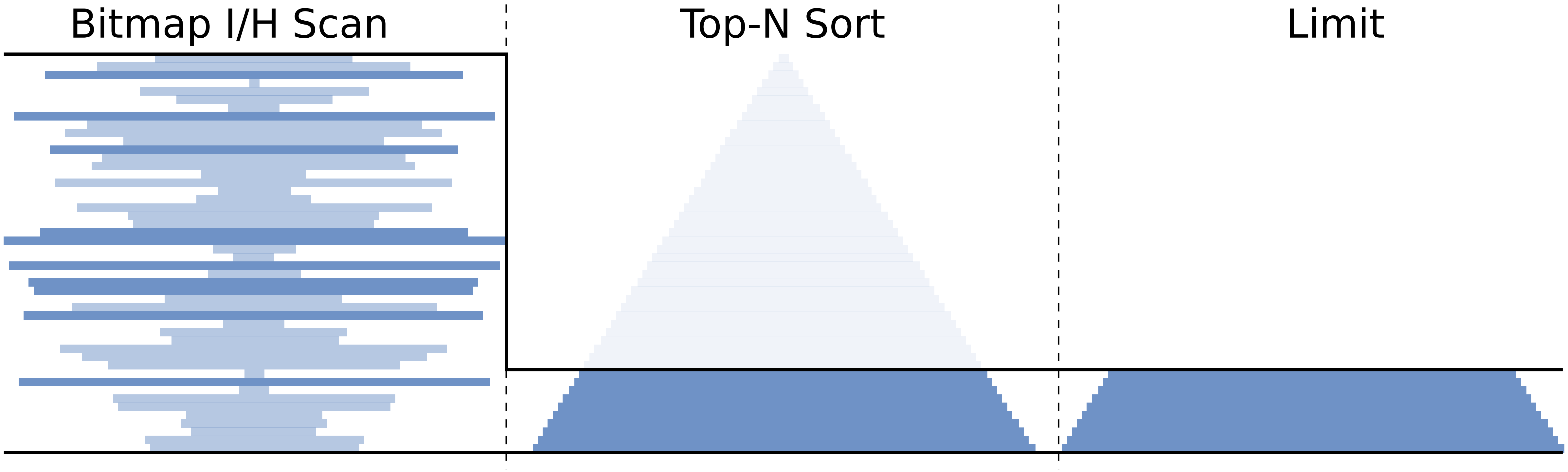
```
create index .. on news(topic);
```

Using `order` by to get the most recent first and `limit` to fetch only the first 10.

Alternative SQL-2008 syntax (since PostgreSQL 8.4)

```
fetch first 10 rows only
```

Worst Case: No Index for `order by`

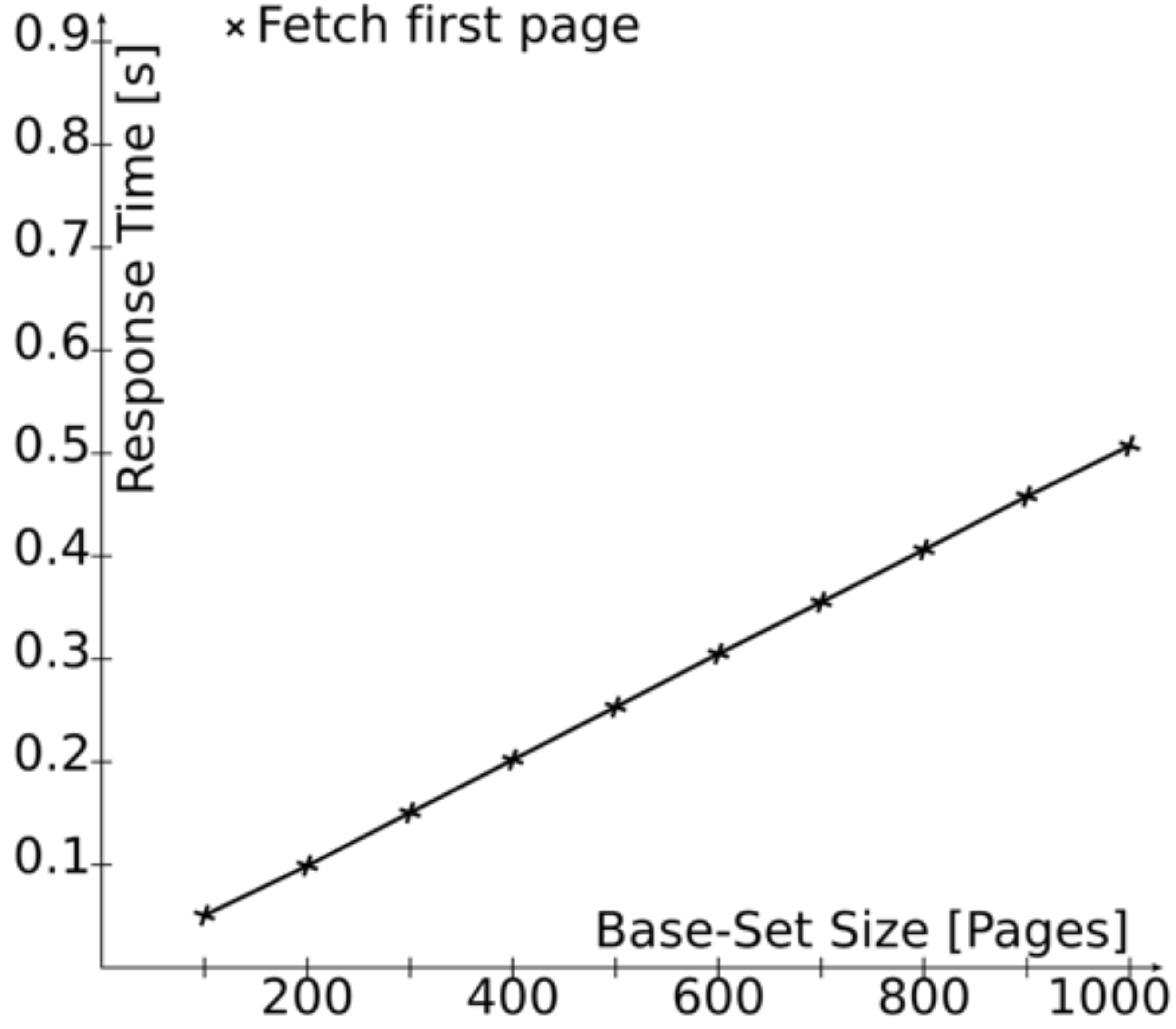


```
Limit (actual rows=10)
-> Sort (actual rows=10)
   Sort Method: top-N heapsort  Memory: 18kB
   -> Bitmap Heap Scan (rows=10000)
      Recheck Cond: (topic = 1234)
      -> Bitmap Index Scan (rows=10000)
         Index Cond: (topic = 1234)
```

Worst Case: No Index for `order by`

The limiting factor is the number of rows that match the `where` clause ("Base-Set Size").

The database might use an index to satisfy the `where` clause, but must still fetch all matching rows to "sort" them.

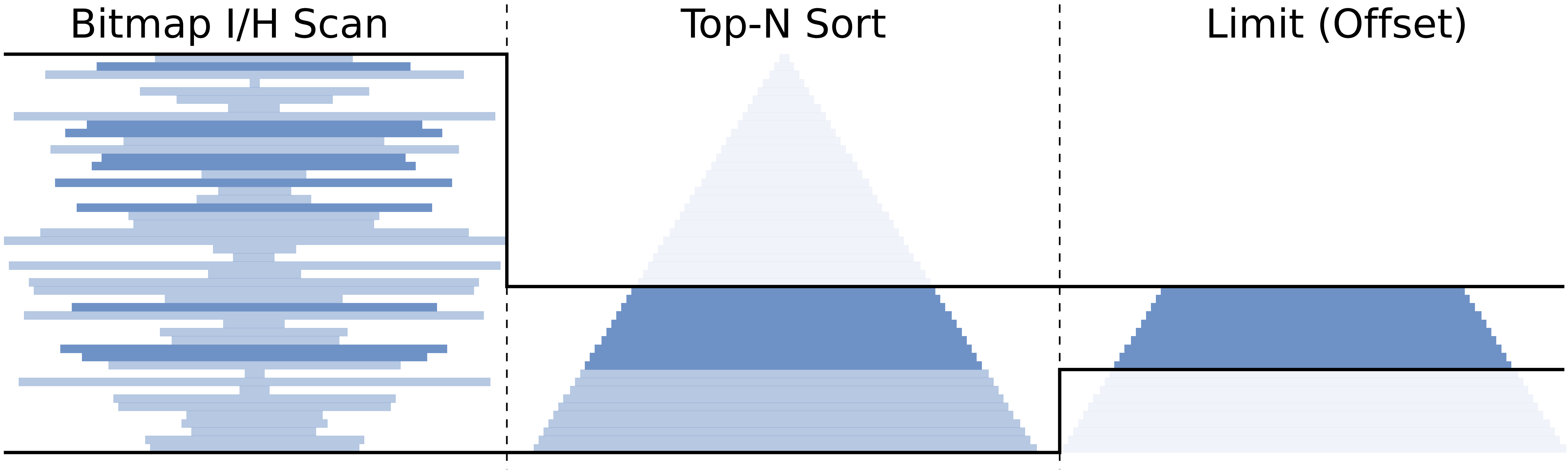


Another Benchmark: Fetch Next Page

Fetching the next page is easy using the `offset` keyword:

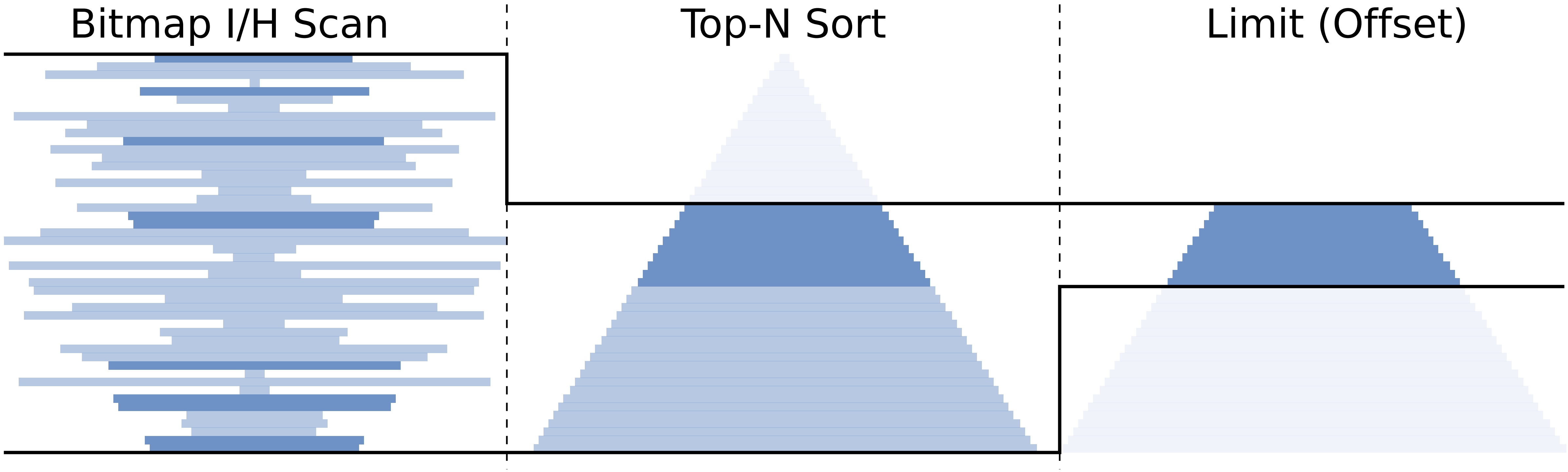
```
select *
  from news
 where topic = 1234
 order by date desc, id desc
offset 10
 limit 10;
```

Worst Case: No Index for `order by`



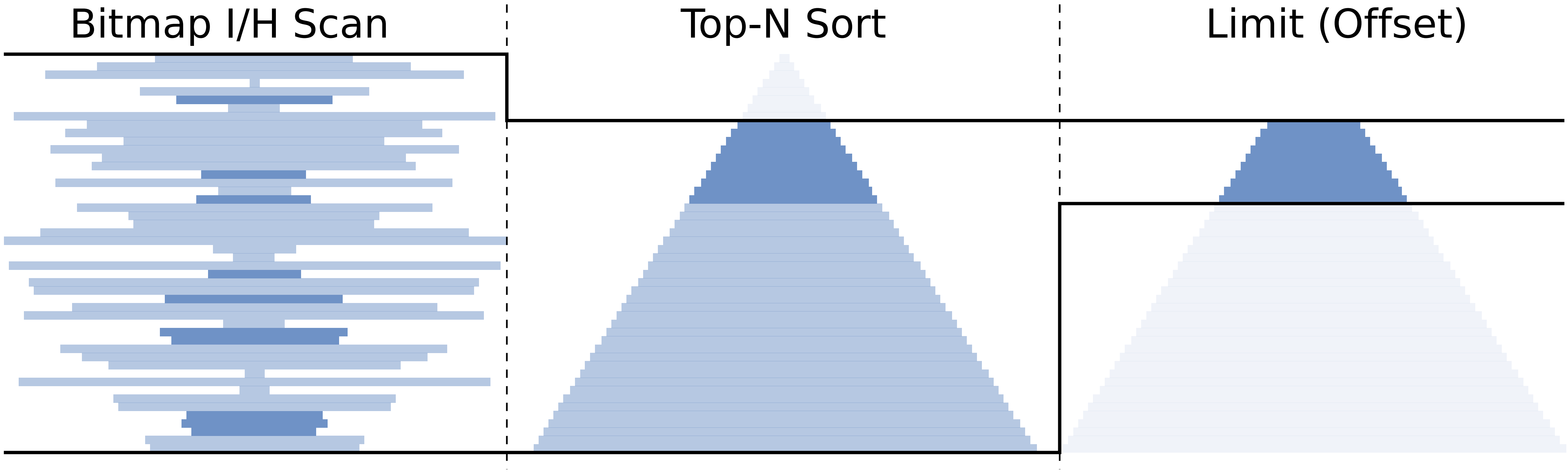
```
Limit (actual rows=10)
-> Sort (actual rows=20)
   Sort Method: top-N heapsort  Memory: 19kB
   ->  Bitmap Heap Scan (actual rows=10000)
       Recheck Cond: (topic = 1234)
       -> Bitmap Index Scan (actual rows=10000)
          Index Cond: (topic = 1234)
```

Worst Case: No Index for `order by`



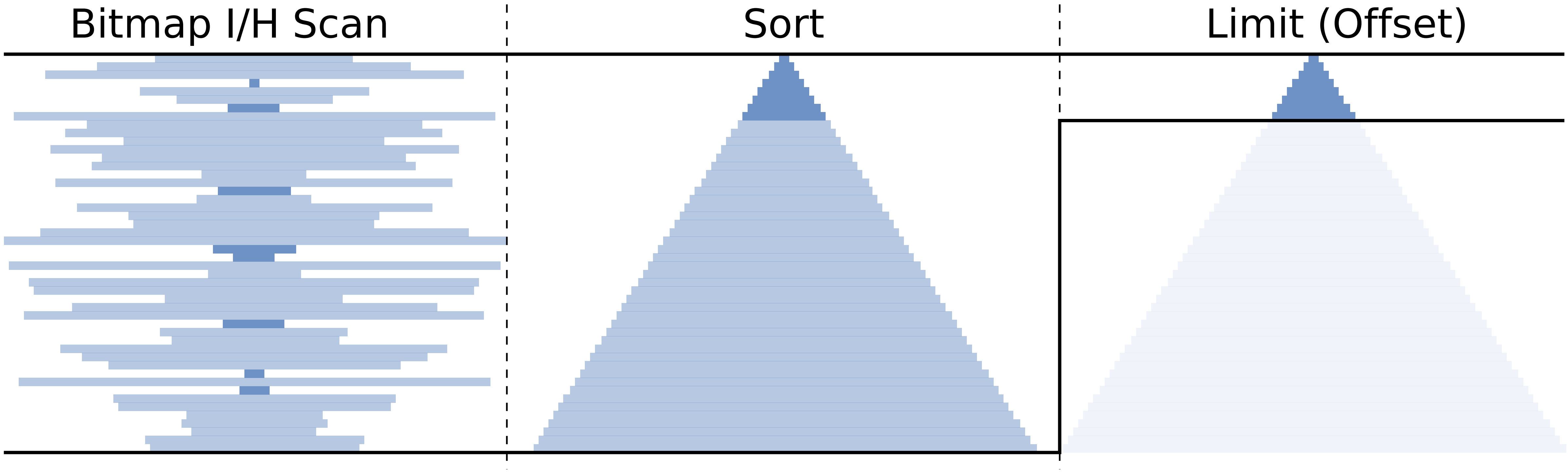
```
Limit (actual rows=10)
-> Sort (actual rows=30)
   Sort Method: top-N heapsort  Memory: 20kB
   ->  Bitmap Heap Scan (actual rows=10000)
       Recheck Cond: (topic = 1234)
       -> Bitmap Index Scan (actual rows=10000)
          Index Cond: (topic = 1234)
```

Worst Case: No Index for `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: No Index for `order by`

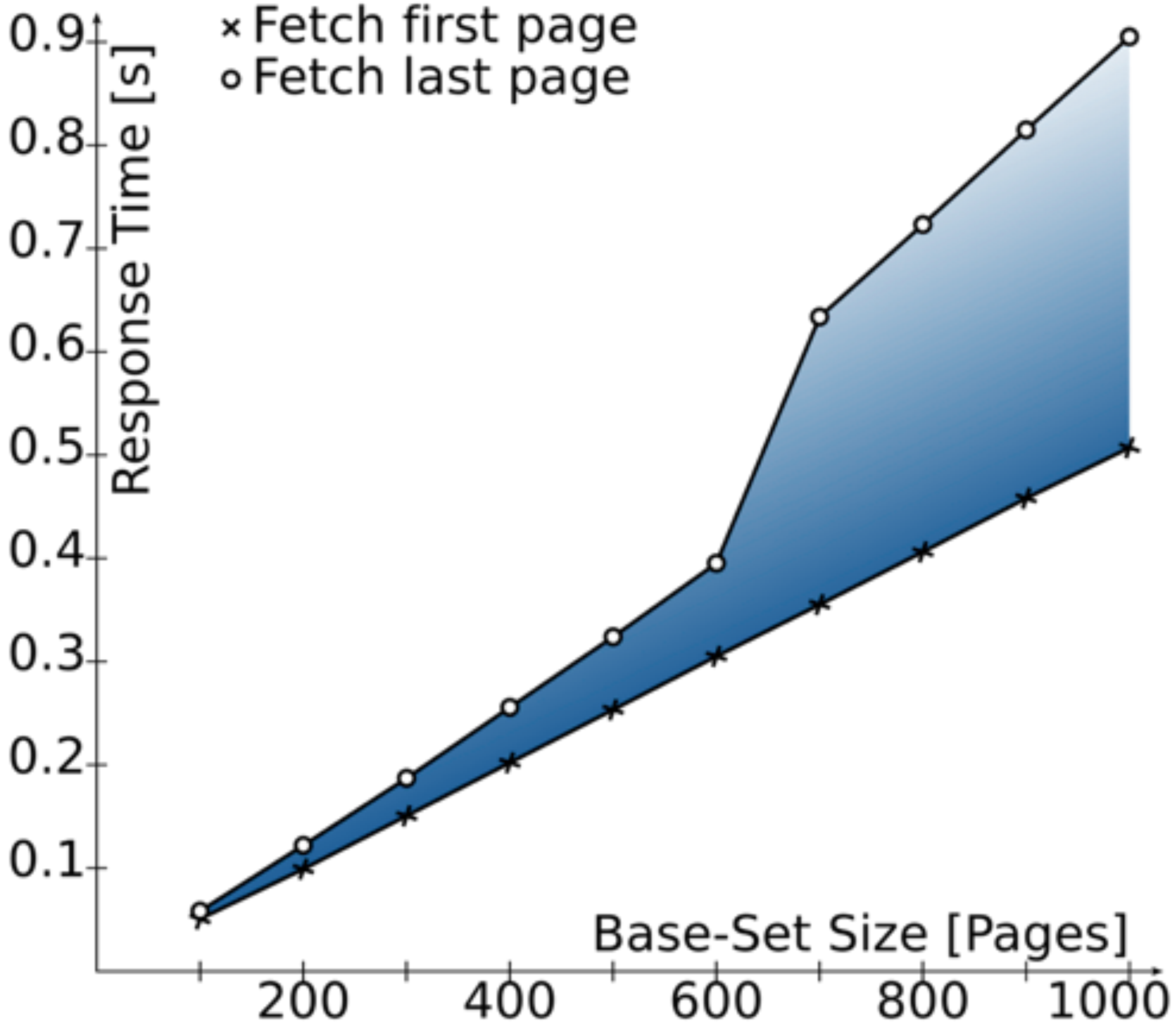


```
Limit (actual rows=10)
-> Sort (actual rows=10000)
   Sort Method: external merge  Disk: 1200kB
   ->  Bitmap Heap Scan (actual rows=10000)
       Recheck Cond: (topic = 1234)
       -> Bitmap Index Scan (actual rows=10000)
          Index Cond: (topic = 1234)
```

Worst Case: No Index for `order by`

Sorting might become the limiting factor when browsing farther back.

Fetching the last page can take considerably longer than fetching the first page.



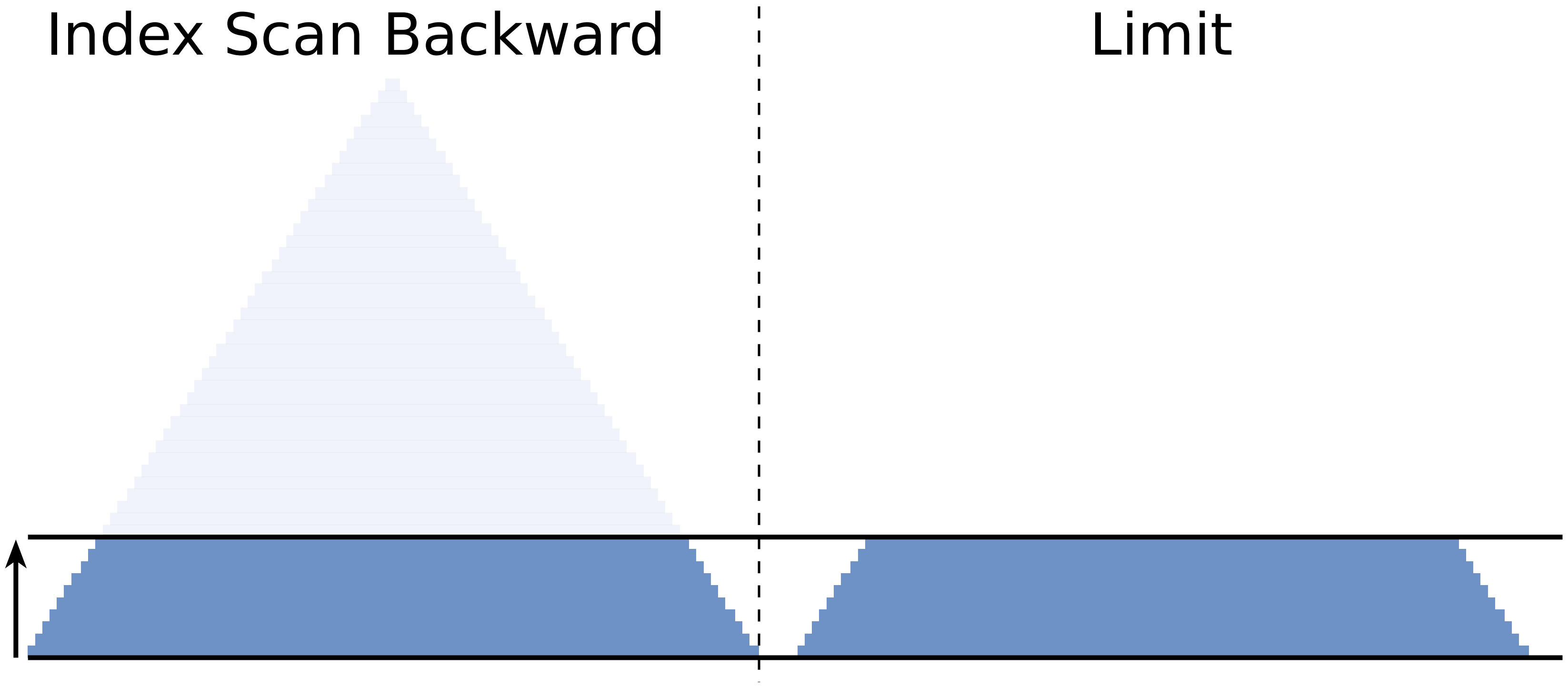
Improvement 1: Indexed `order by`

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

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

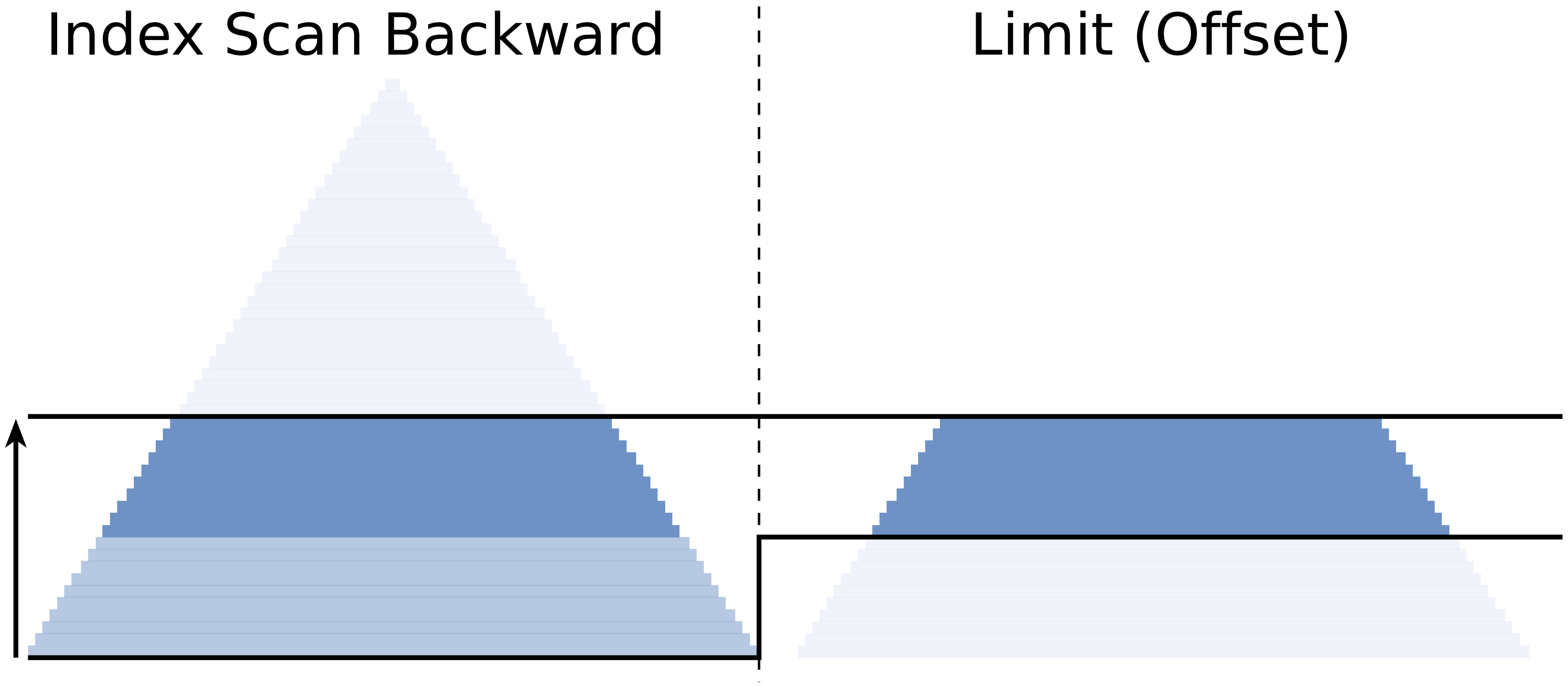
A single index to support the `where` and `order by` clauses.

Improvement 1: Indexed `order by`



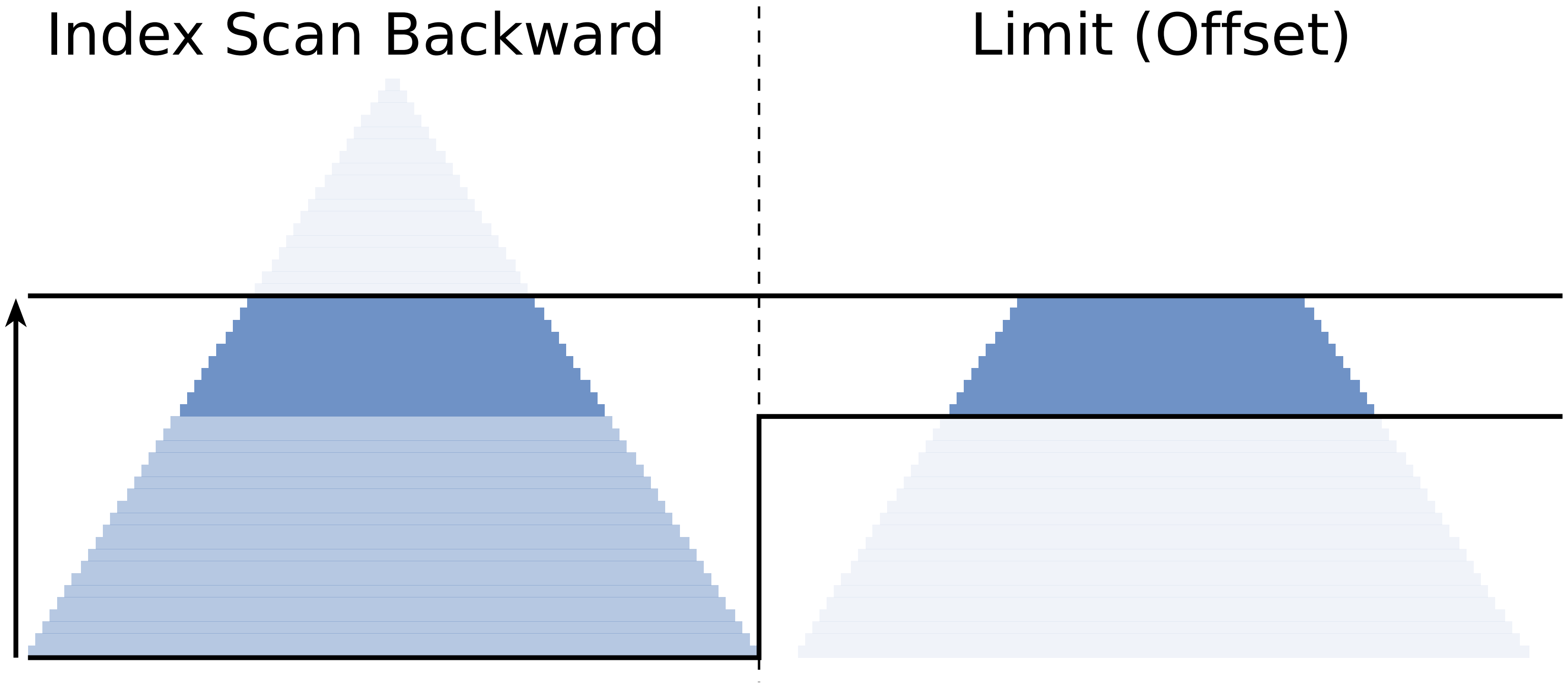
```
Limit (actual rows=10)
-> Index Scan Backward (actual rows=10)
   Index Cond: (topic = 0)
```

Improvement 1: Indexed `order by`



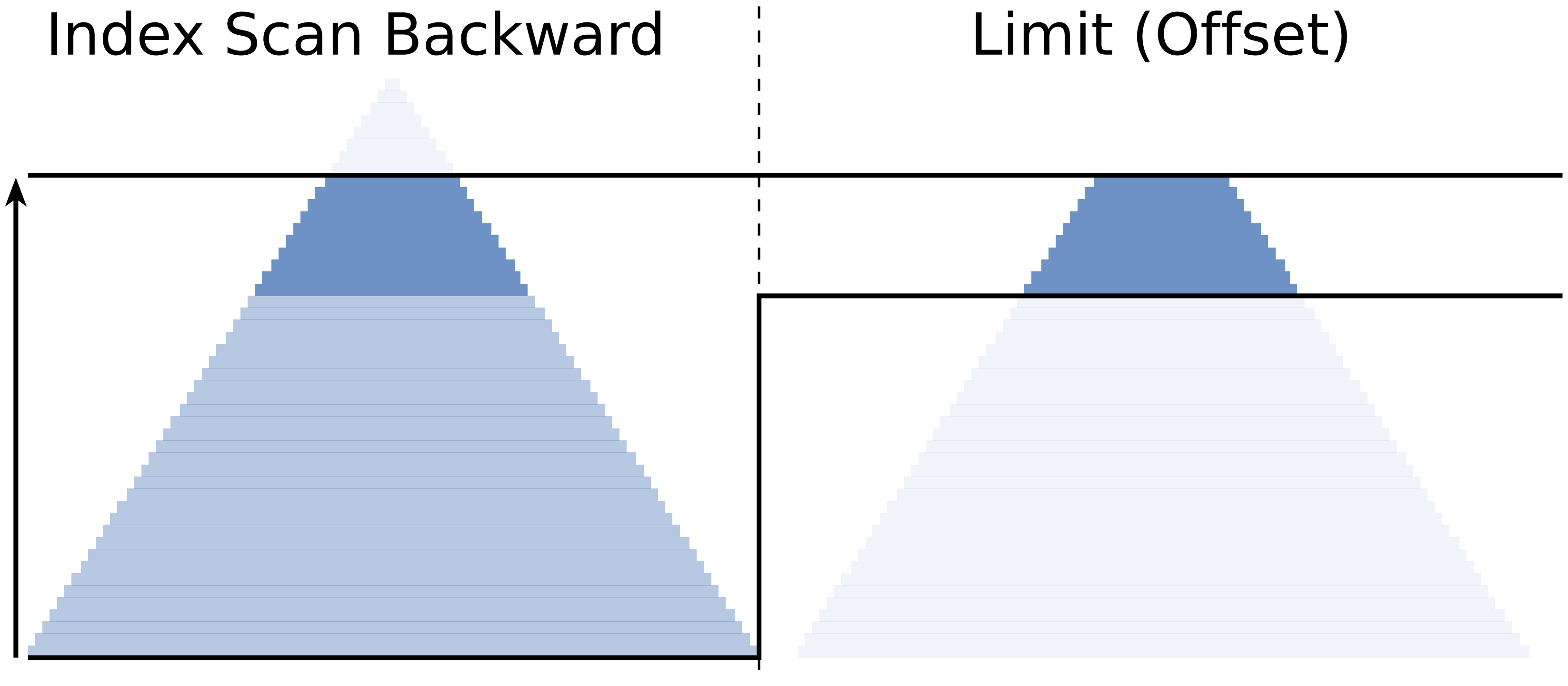
```
Limit (actual rows=10)
-> Index Scan Backward (actual rows=20)
   Index Cond: (topic = 0)
```

Improvement 1: Indexed `order by`



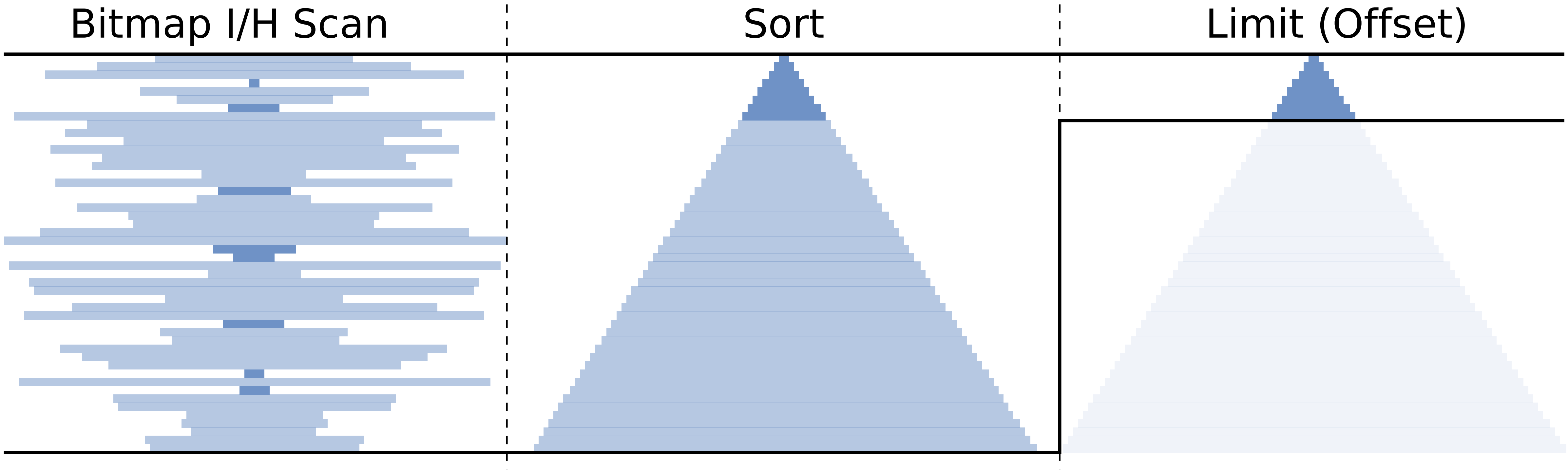
```
Limit (actual rows=10)
-> Index Scan Backward (actual rows=30)
   Index Cond: (topic = 0)
```

Improvement 1: Indexed `order by`



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

Improvement 1: Indexed `order by`

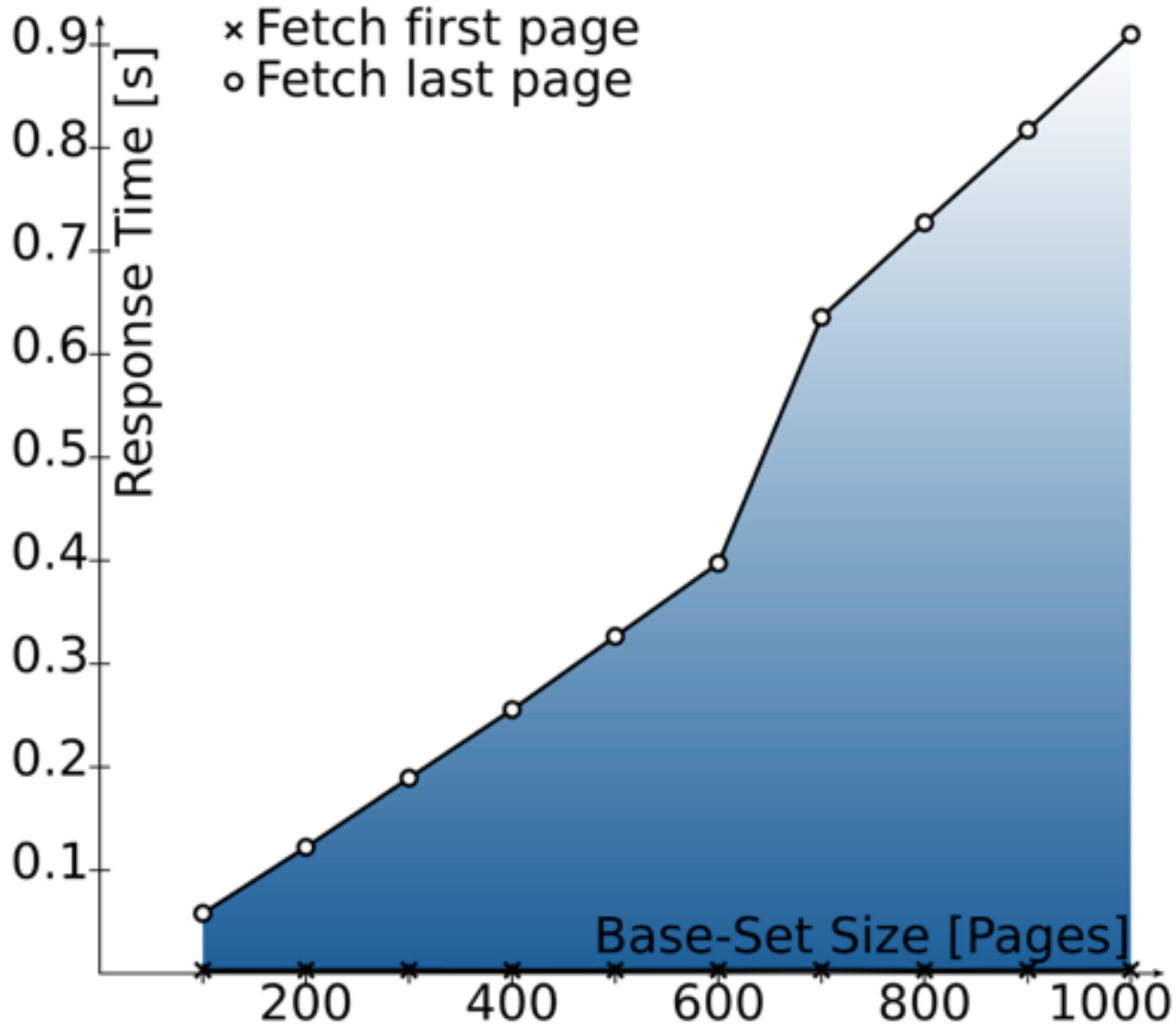


```
Limit (actual rows=10)
-> Sort (actual rows=10000)
   Sort Method: external merge  Disk: 1200kB
   ->  Bitmap Heap Scan (actual rows=10000)
       Recheck Cond: (topic = 1234)
       -> Bitmap Index Scan (actual rows=10000)
          Index Cond: (topic = 1234)
```

Improvement 1: Indexed `order by`

Fetching the first page is not affected by the Base-Set size!

Fetching the <u>next page</u> is also faster. However, PostgreSQL might take a Bitmap Index Scan when browsing to the end.



We can do better!

Don’t touch what you don’t need

Improvement 2: The Seek Method

Instead of `offset`, use a `where` filter to remove the rows from previous pages.

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

Only select the rows “before” (=earlier date) the last row from the previous page.

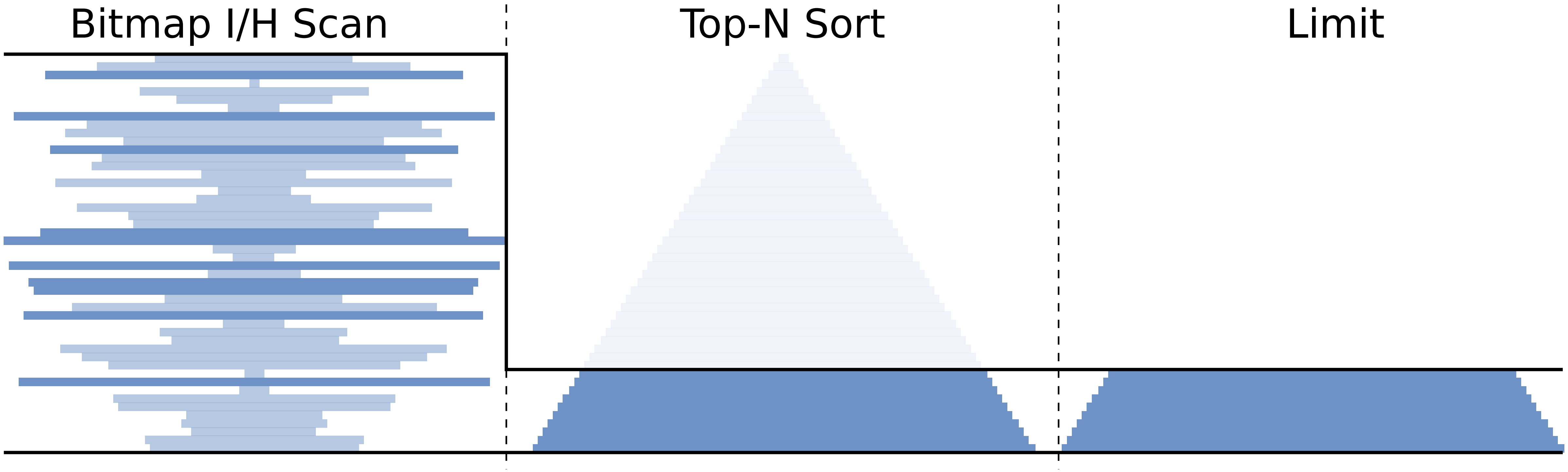
A definite sort order is really required!

Side Note: Row Values/Constructors

Besides scalar values, SQL also defines "row values" or "composite values."

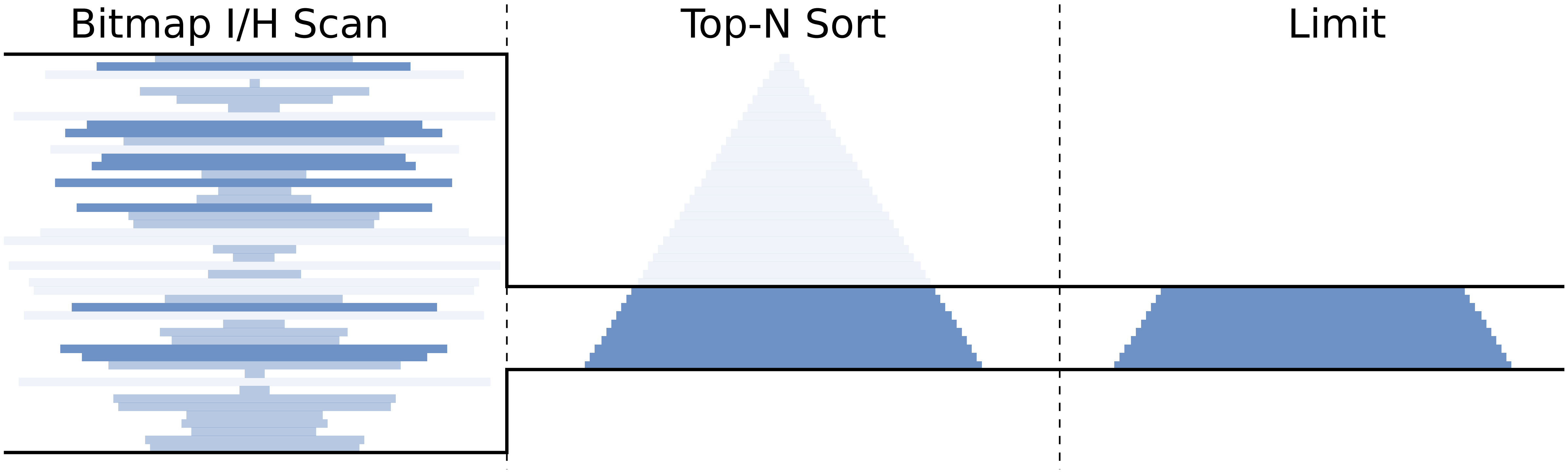
- In the SQL standard since ages (SQL-92)
- All comparison operators are well defined
  - E.g.: (x, y) > (a, b) is true iff
    (x > a or (x=a and y>b))
  - In other words, **when (x,y) sorts after (a,b)**
- Great PostgreSQL support since 8.0!

Seek Method w/o Index for `order by`



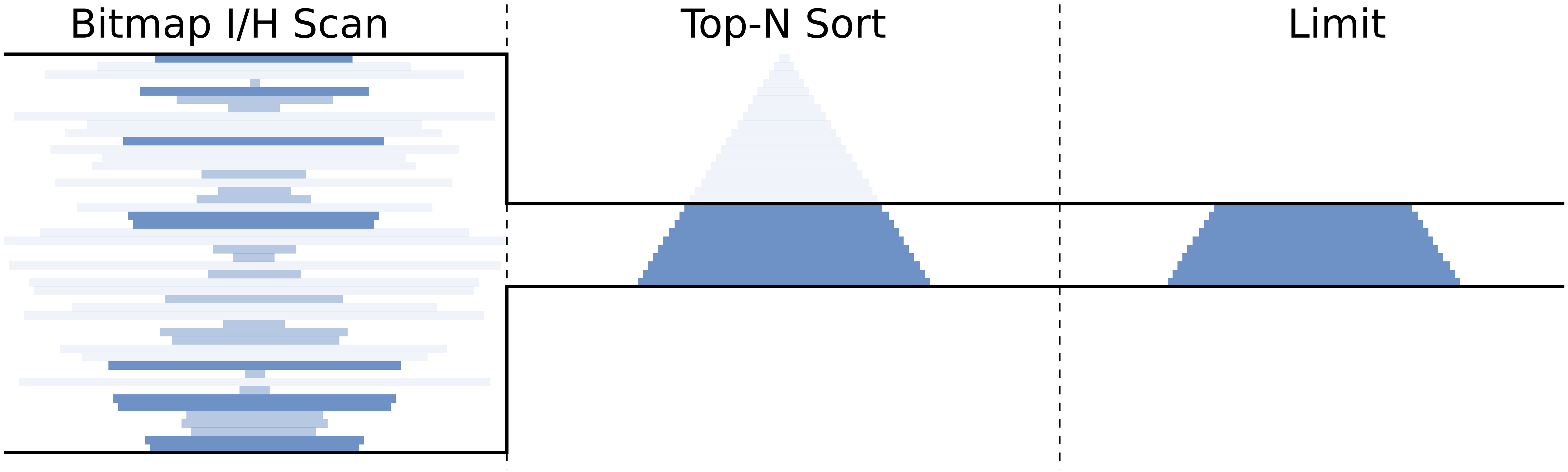
```
Limit (actual rows=10)
-> Sort (actual rows=10)
   Sort Method: top-N heapsort  Memory: 18kB
   -> Bitmap Heap Scan (rows=10000)
      Recheck Cond: (topic = 1234)
      -> Bitmap Index Scan (rows=10000)
         Index Cond: (topic = 1234)
```

Seek Method w/o Index for `order by`



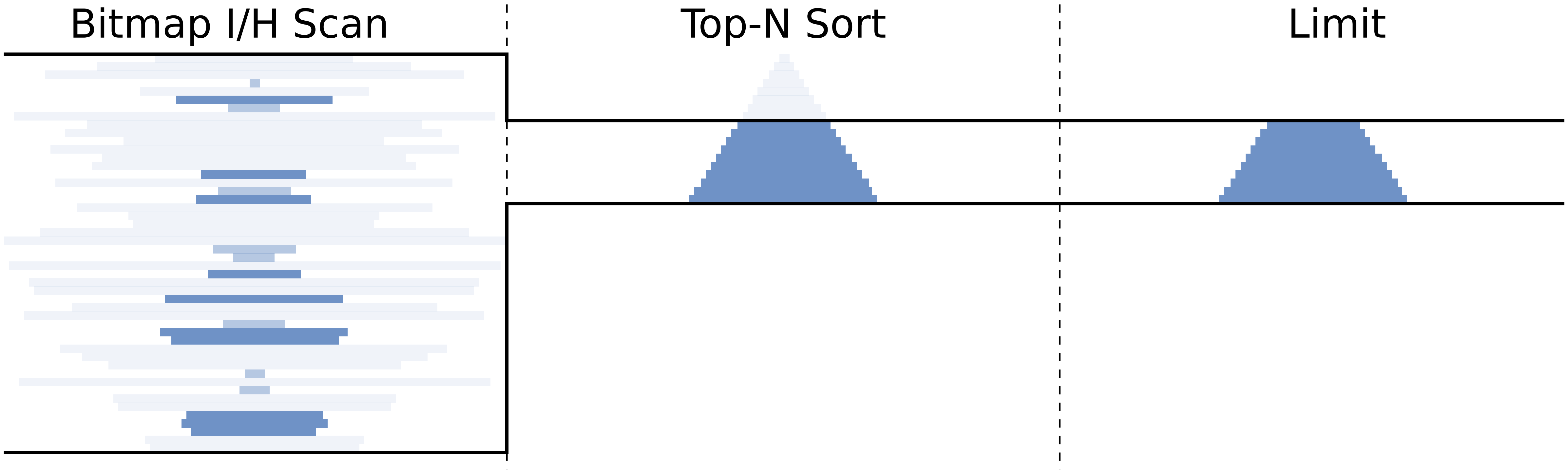
```
Limit (actual rows=10)
 -> Sort (actual rows=10)
    Sort Method: top-N heapsort  Memory: 18kB
    -> Bitmap Heap Scan (actual rows=9990)
       Rows Removed by Filter: 10 (new in 9.2)
       -> Bitmap Index Scan (actual rows=10000)
          Index Cond: (topic = 1234)
```

Seek Method w/o Index for `order by`



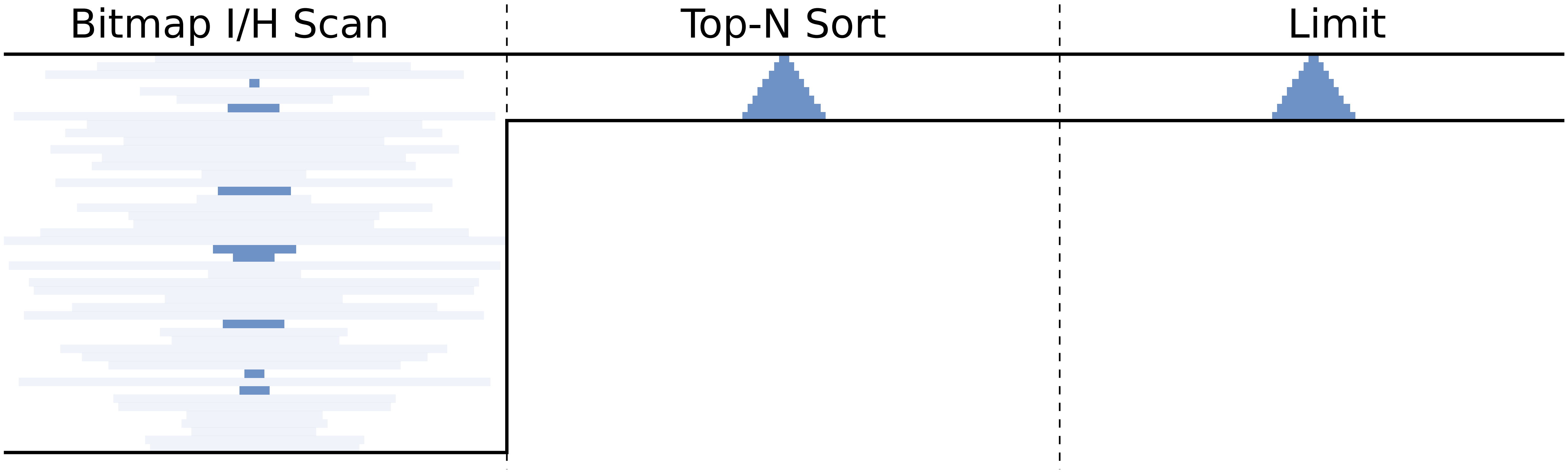
```
Limit (actual rows=10)
 -> Sort (actual rows=10)
    Sort Method: top-N heapsort  Memory: 18kB
    -> Bitmap Heap Scan (actual rows=9980)
       Rows Removed by Filter: 20 (new in 9.2)
       -> Bitmap Index Scan (actual rows=10000)
          Index Cond: (topic = 1234)
```

Seek Method w/o Index for `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)
```

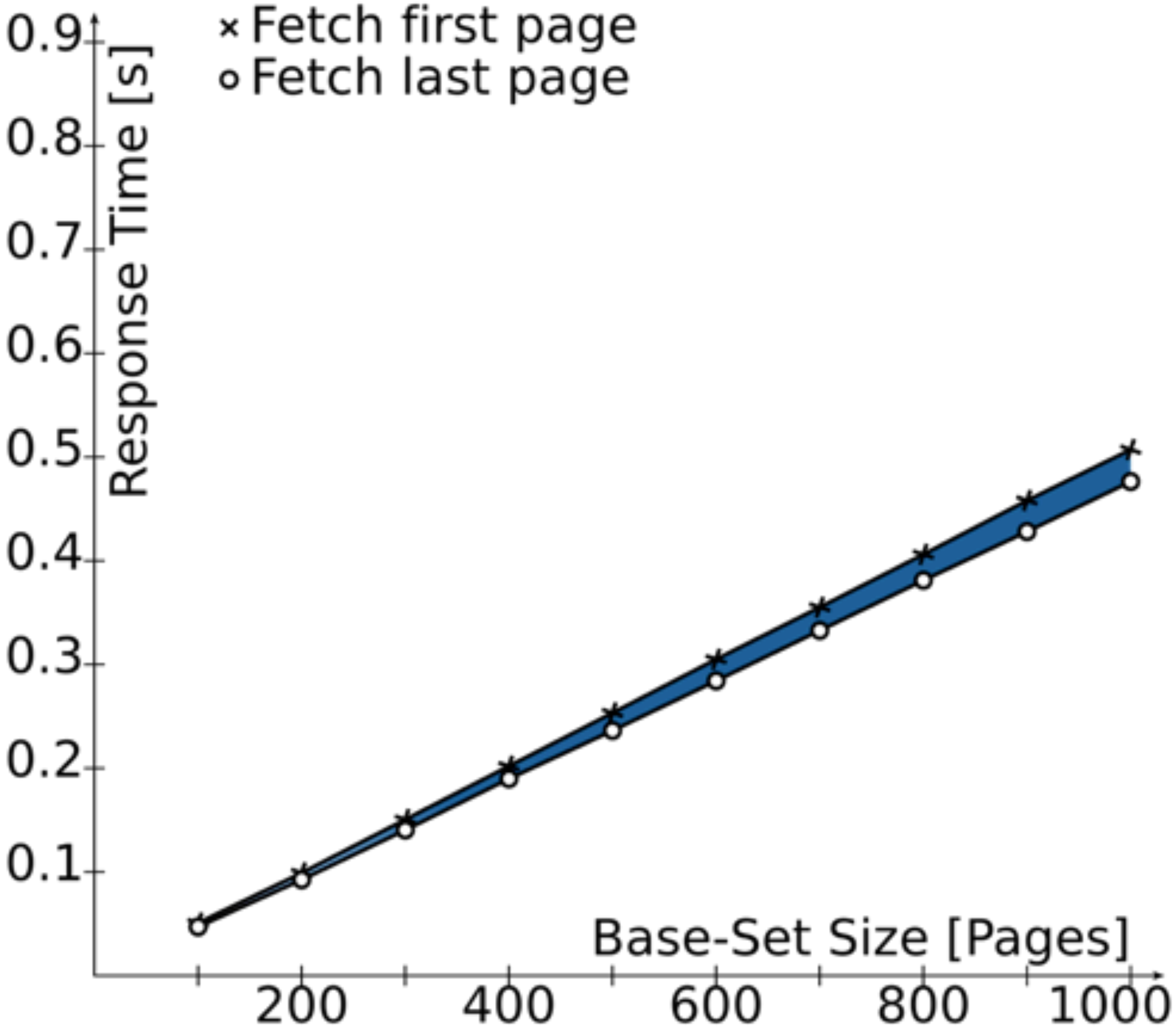
Seek Method w/o Index for `order by`



```
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)
```

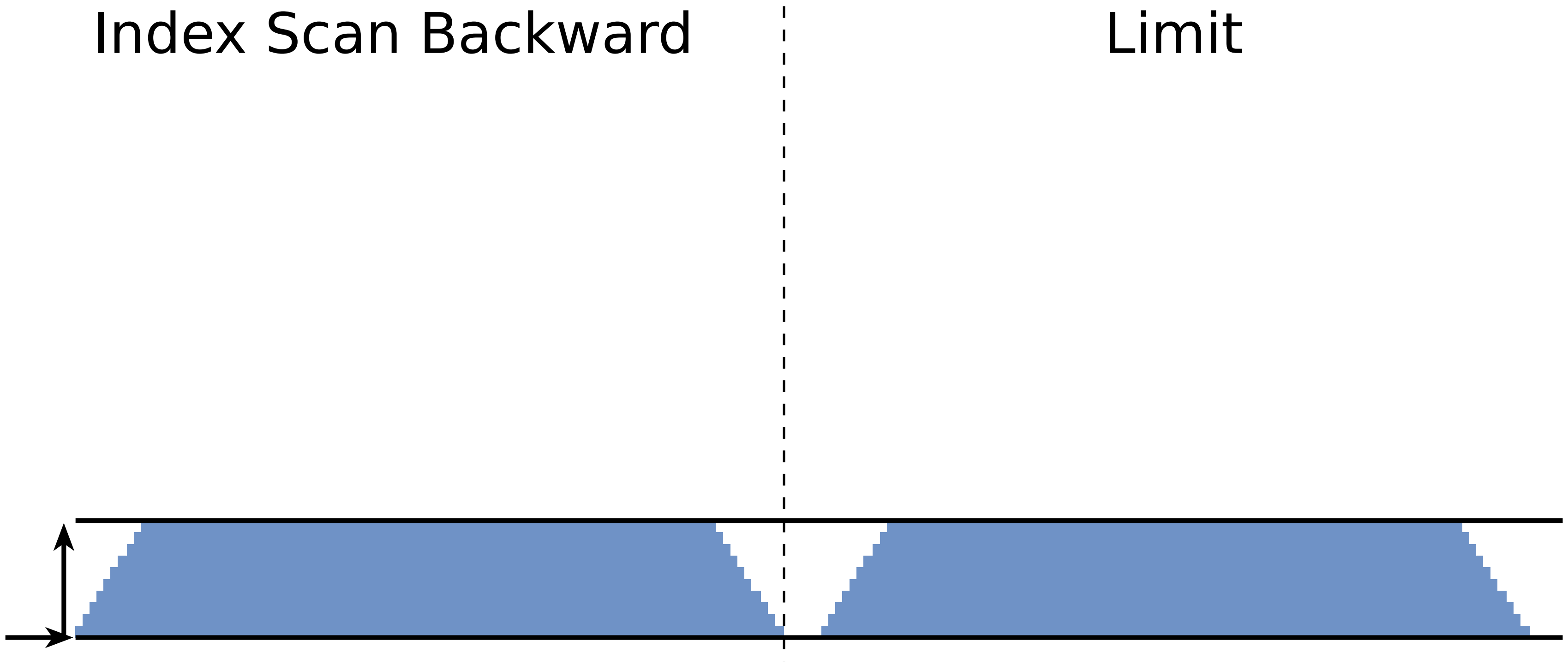
Seek Method w/o Index for `order by`

Always needs to retrieve the full base set, but the top-n sort buffer needs to hold only 10 rows.



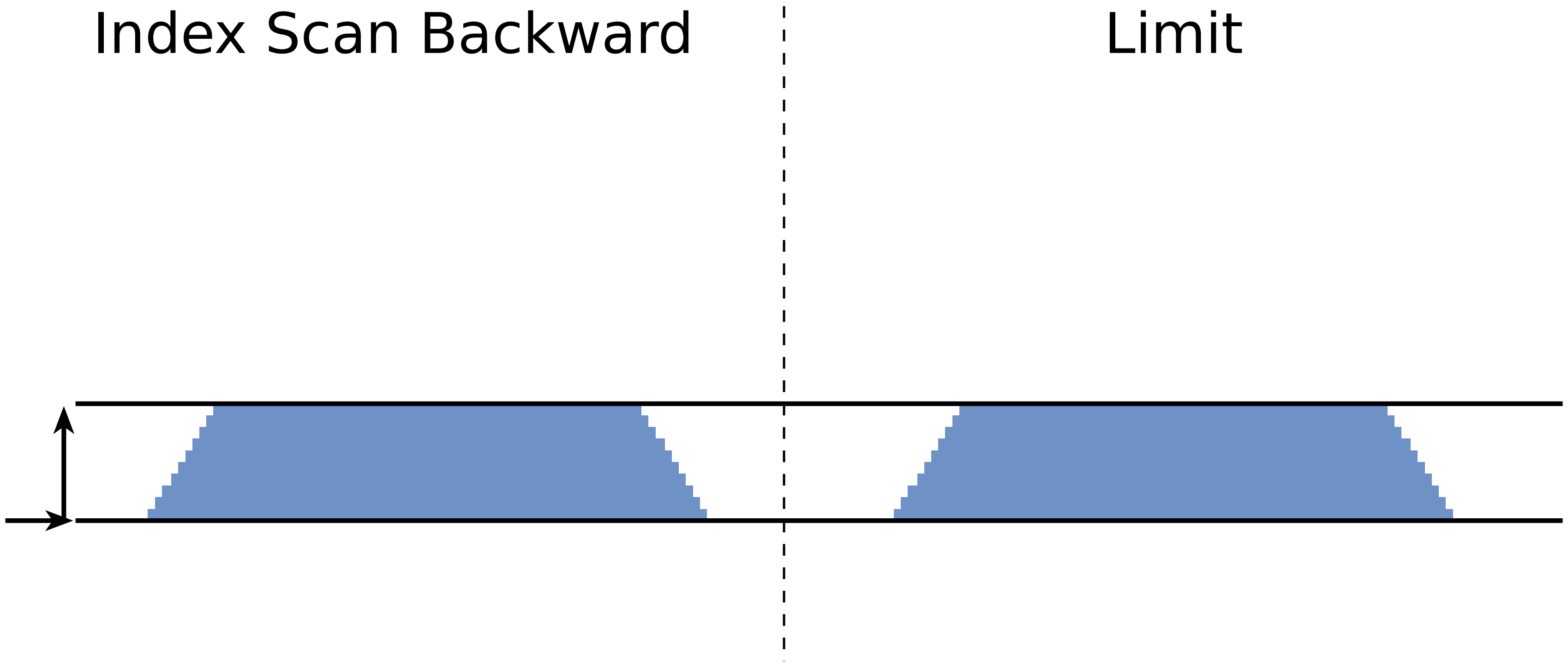
The response time remains the same even when browsing to the last page. And the memory footprint is very low!

Seek Method with Index for `order by`



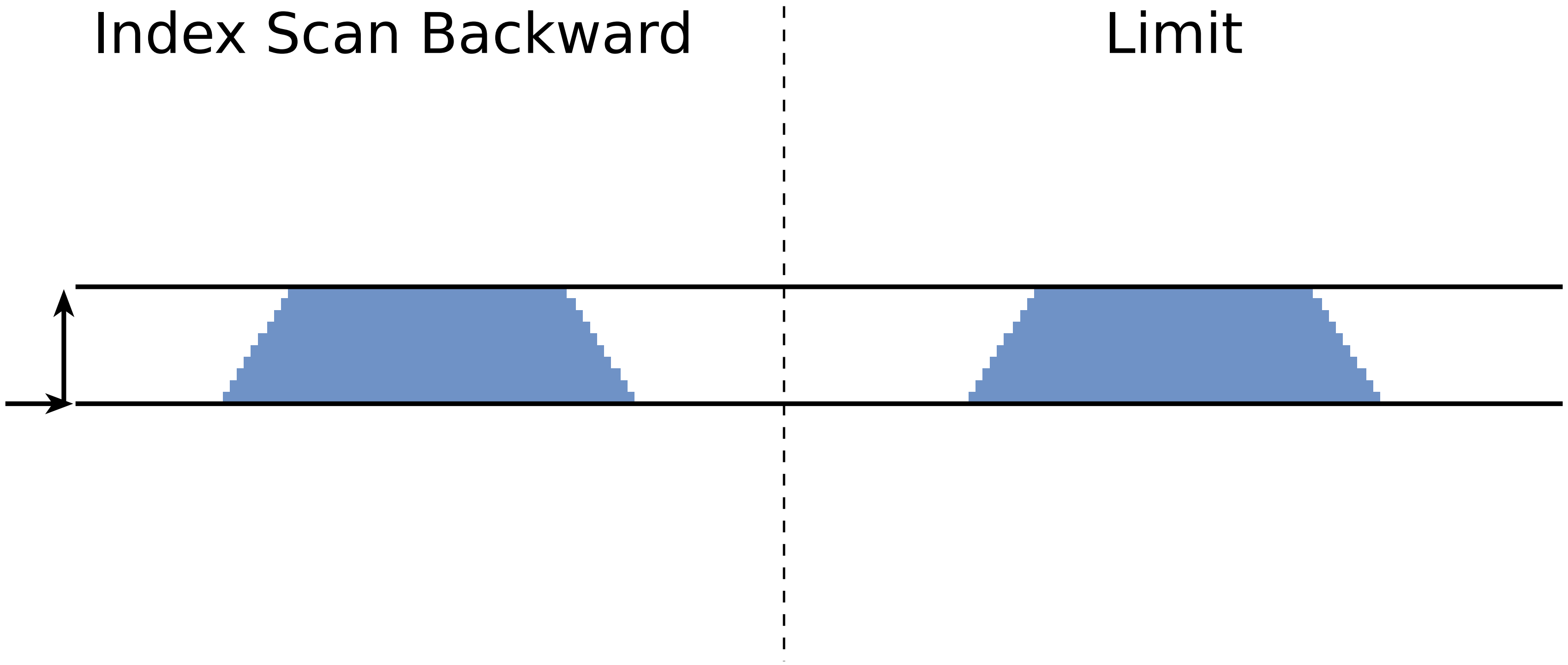
```
Limit (actual rows=10)
-> Index Scan Backward (actual rows=10)
   Index Cond: (topic = 1234)
```

Seek Method with Index for `order by`



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

Seek Method with Index for `order by`

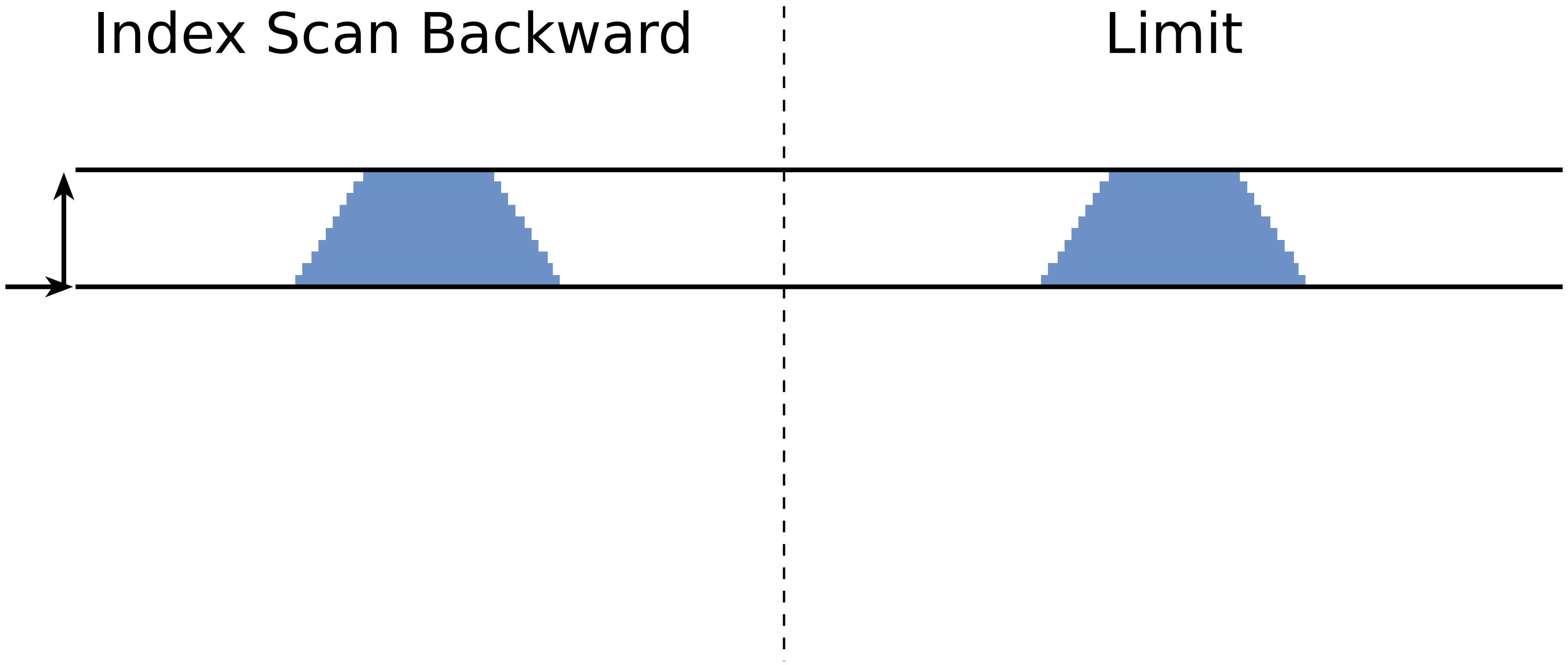


Index Scan Backward

Limit

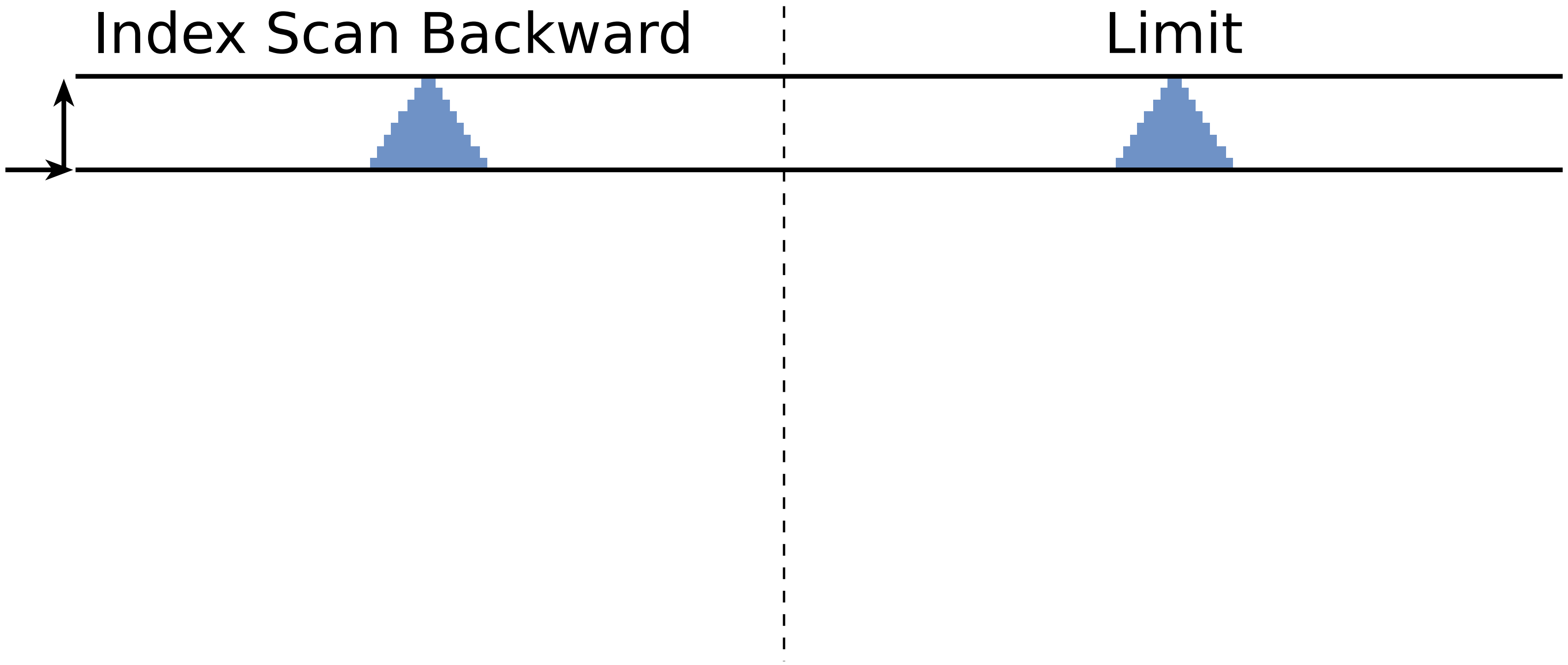
```
Limit (actual rows=10)
-> Index Scan Backward (actual rows=10)
   Index Cond: ((topic = 1234)
                AND (ROW(dt, id) < ROW(‘...’, 34567)))
```

Seek Method with Index for `order by`



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

Seek Method with Index for `order by`

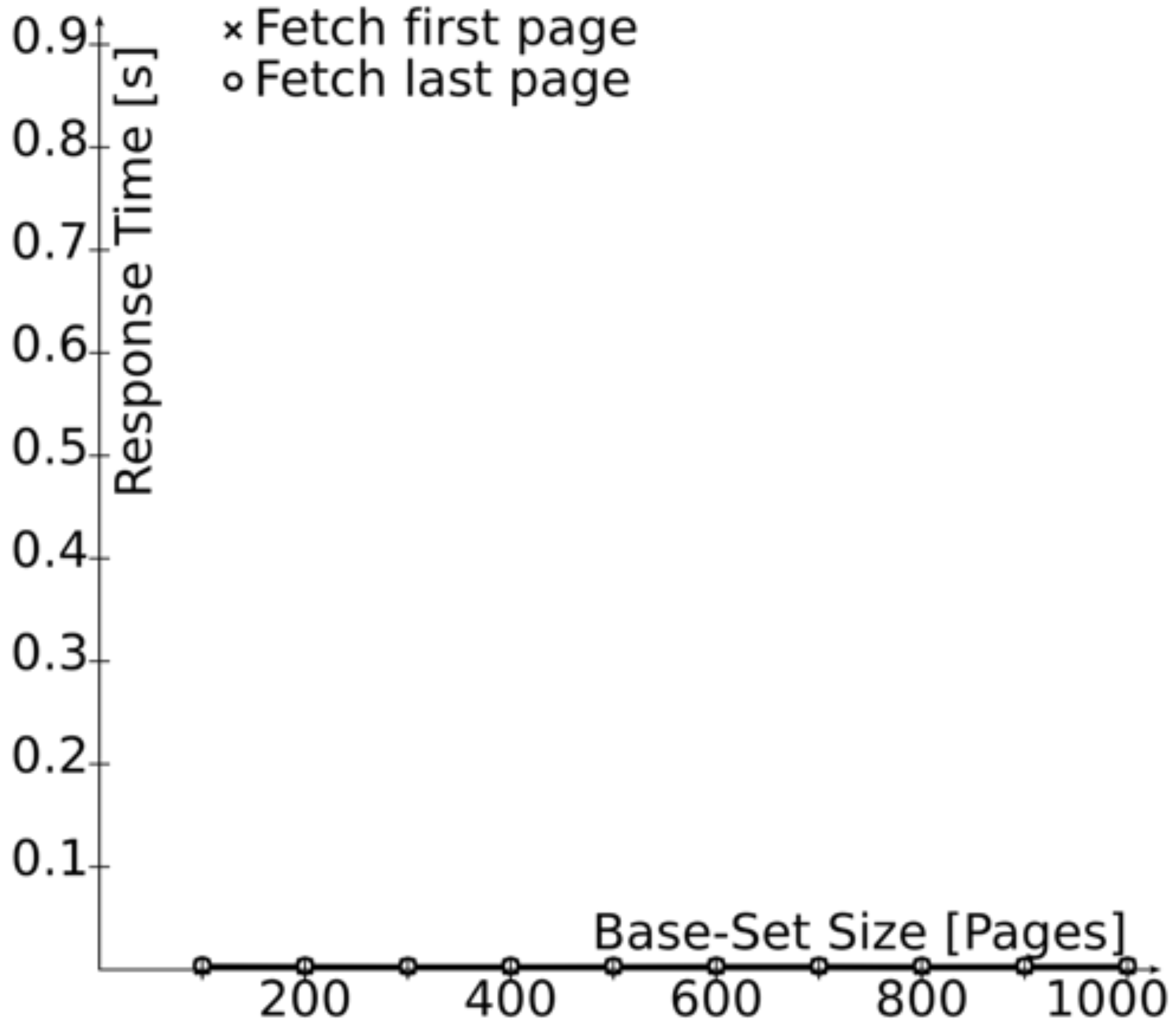


```
Limit (actual rows=10)
-> Index Scan Backward (actual rows=10)
   Index Cond: ((topic = 1234)
                AND (ROW(dt, id) < ROW('...', 56789)))
```

Seek Method with Index for `order by`

Successively browsing back doesn’t slow down.

Neither the size of the base set(*) nor the fetched page number affects the response time.

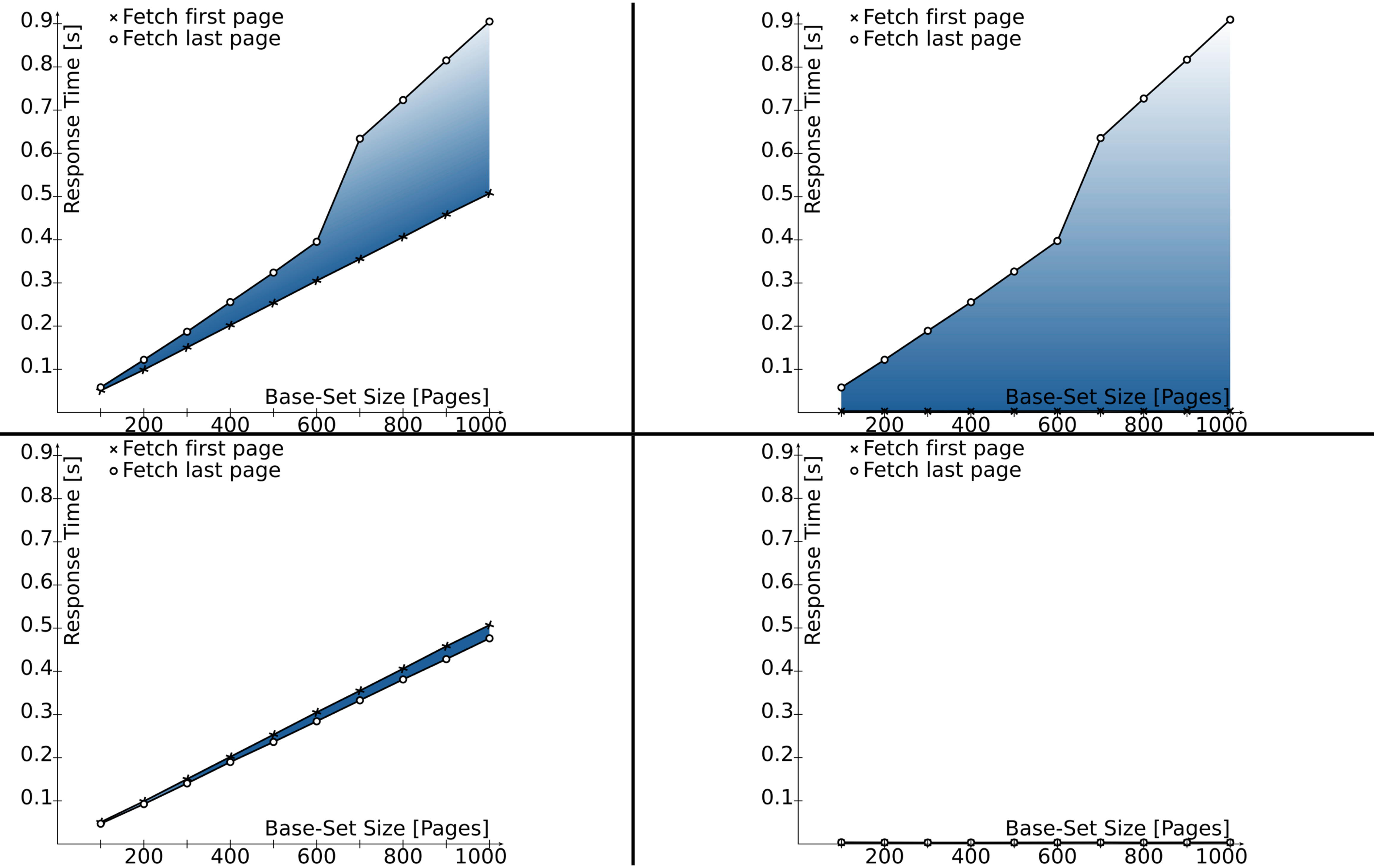


(*) the index tree depth still affects the response time.

Comparison

Offset

Seek



W/O index for `order by` With index for `order by`

Too good to be true?

The Seek Method has serious limitations

Too good to be true?

The Seek Method has serious limitations

- You cannot directly navigate to arbitrary pages
  - because you need the values from the previous page

Too good to be true?

The Seek Method has serious limitations

- You cannot directly navigate to arbitrary pages
  - because you need the values from the previous page

- Bi-directional navigation is possible but tedious
  - you need to revers the `order by` direction and RV comparison

Too good to be true?

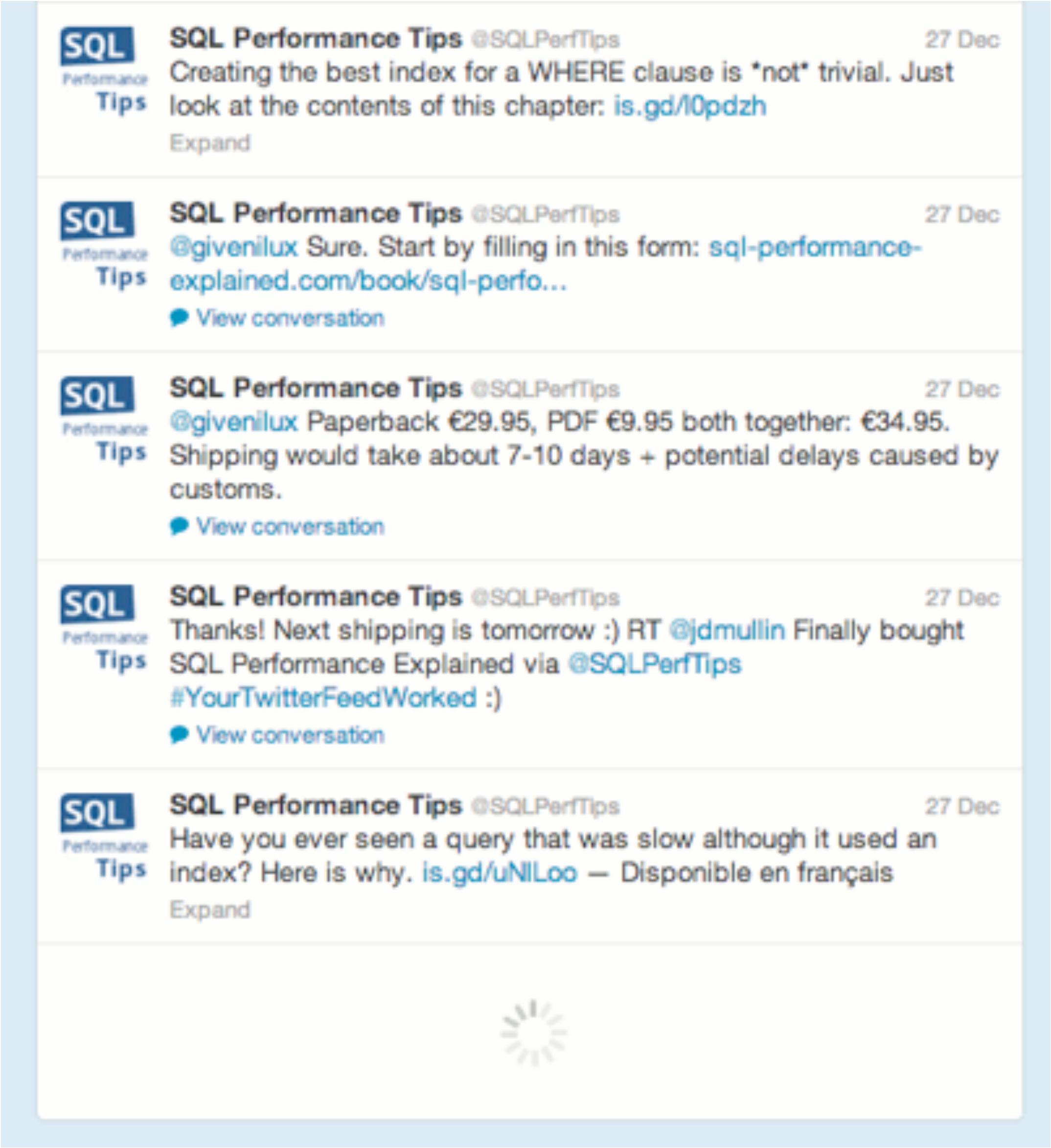
The Seek Method has serious limitations

- You cannot directly navigate to arbitrary pages
  - because you need the values from the previous page
- Bi-directional navigation is possible but tedious
  - you need to revers the `order by` direction and RV comparison
- Works best with full row values support
  - Workaround is possible, but ugly and less performant
  - Framework support?

A Perfect Match for Infinite Scrolling

The “Infinite Scrolling” UI doesn’t need to ...

- navigate to arbitrary pages
  - there are no buttons
- Browse backwards
  - previous pages are still in the browser
- show total hits
  - if you need to, you are doomed anyway!



SQL Performance Tips @SQLPerfTips 27 Dec
Creating the best index for a WHERE clause is *not* trivial. Just look at the contents of this chapter: is.gd/l0pdzh
Expand

SQL Performance Tips @SQLPerfTips 27 Dec
@givenilux Sure. Start by filling in this form: sql-performance-explained.com/book/sql-perfo...
View conversation

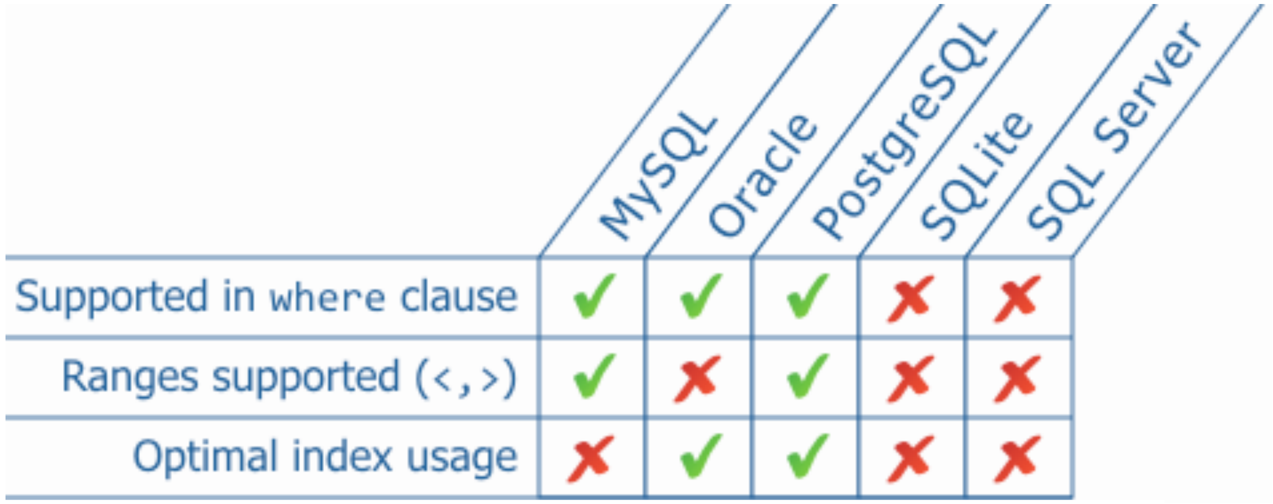
SQL Performance Tips @SQLPerfTips 27 Dec
@givenilux Paperback €29.95, PDF €9.95 both together: €34.95. Shipping would take about 7-10 days + potential delays caused by customs.
View conversation

SQL Performance Tips @SQLPerfTips 27 Dec
Thanks! Next shipping is tomorrow :) RT @jdmullin Finally bought SQL Performance Explained via @SQLPerfTips #YourTwitterFeedWorked :)
View conversation

SQL Performance Tips @SQLPerfTips 27 Dec
Have you ever seen a query that was slow although it used an index? Here is why. is.gd/uNlLoo — Disponible en français
Expand

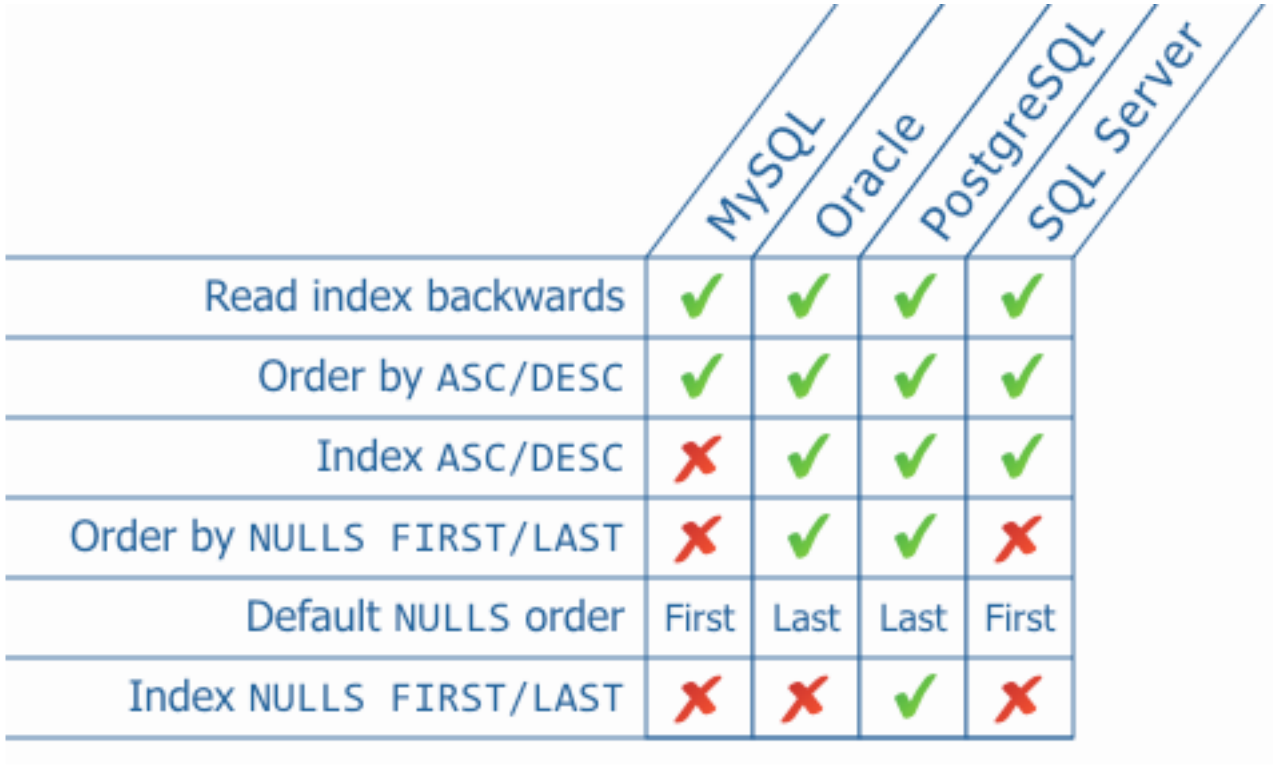
Also a Perfect Match for PostgreSQL

row values
support matrix



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

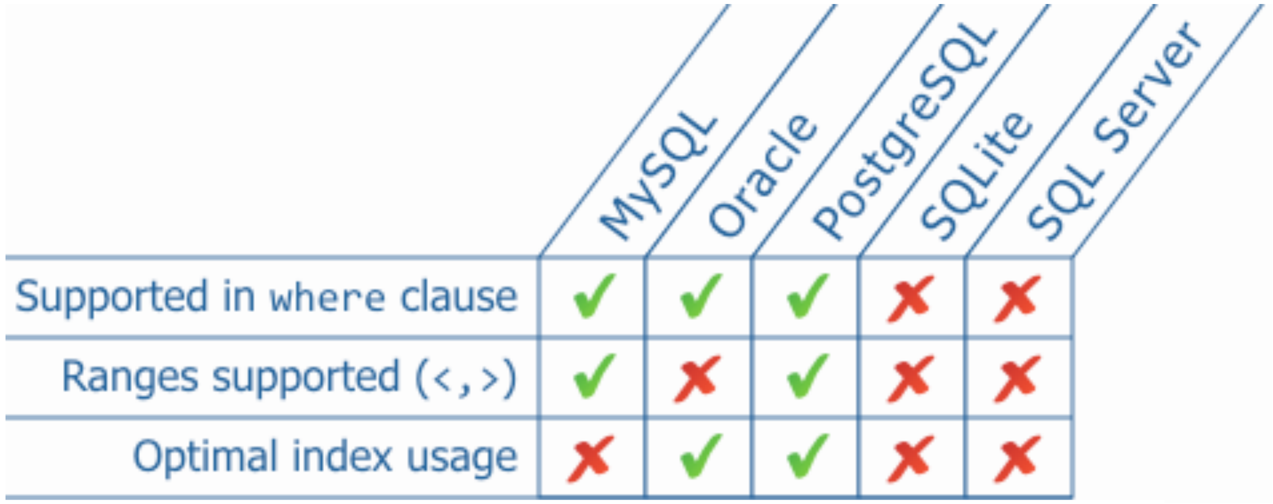
`order by`
support matrix



	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`	✗	✗	✓	✗

Also a Perfect Match for PostgreSQL

row values
support matrix

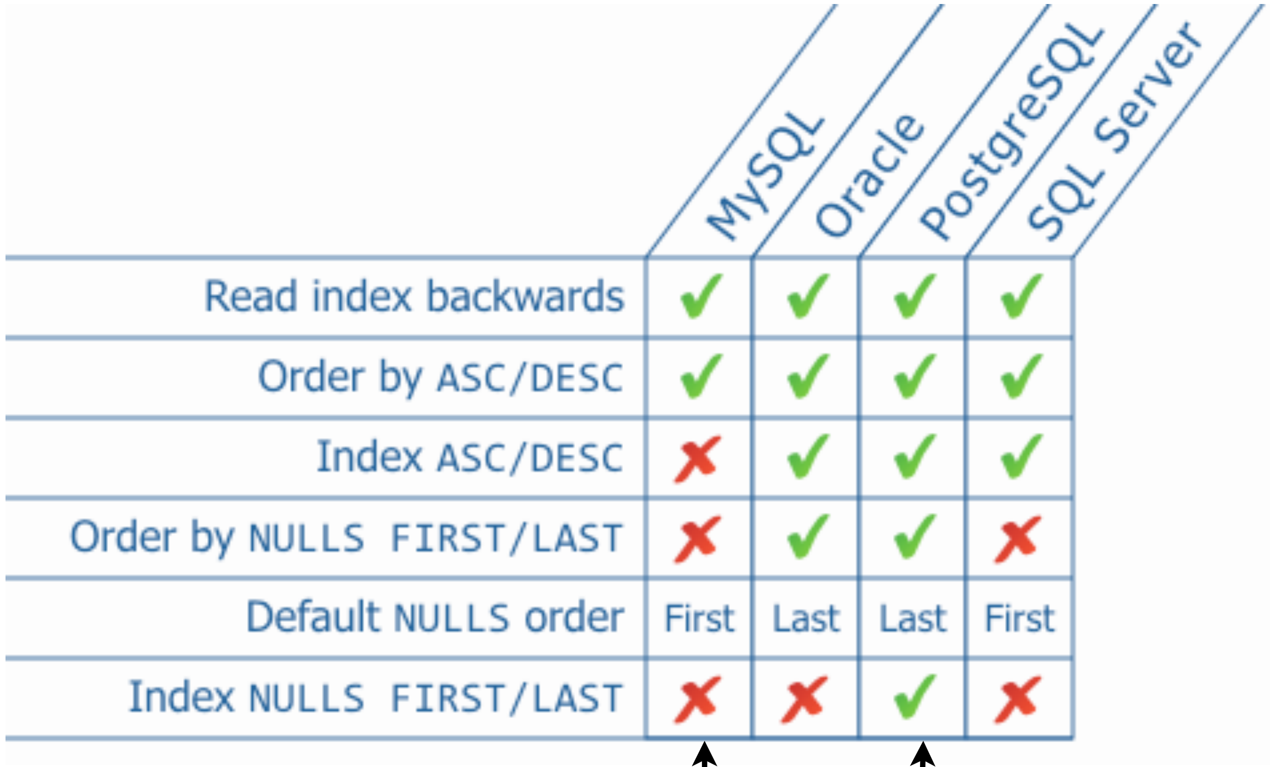


	MySQL	Oracle	PostgreSQL	SQLite	SQL Server
Supported in where clause	✓	✓	✓	✗	✗
Ranges supported (<,>)	✓	✗	✓	✗	✗
Optimal index usage	✗	✓	✓	✗	✗

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