Annis on MonetDB

Viktor Rosenfeld
rosenfel@informatik.hu-berlin.de

14. January 2013

Advisors: Prof. Dr. Ulf Leser and Dr. Stefan Manegold

http://www.flickr.com/photos/karola/3623768629

1. What is Annis and how is it used?
2. Current implementation on PostgreSQL
3. What are Column-Stores? How can Annis benefit?
4. New implementation on MonetDB and evaluation

1. What is Annis and how is it used?

What's a corpus?

any principled collection of language

THE WALL STREET JOURNAL.

die tageszeitung

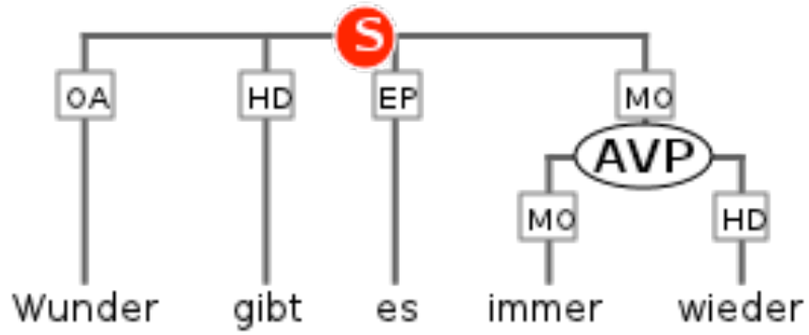
Falko

What's an annotation?

classification and interpretation of the corpus data

additional data to enrich the corpus

Wunder	gibt	es	immer	wieder	!
Wunder	geben	es	immer	wieder	!
Acc.Pl.Neut	3.Sg.Pres.Ind	3.Nom.Sg.Neut	--	--	--
NN	VVFIN	PPER	ADV	ADV	$.



Steilpass Wunder gibt es immer wieder ! Erst spielen die Dallgower Gemeindevertreter so statisch und verzagt wie die deutsche Abwehrreihe der Fußballkicker . Und dann kommt aus der Tiefe solch ein fulminanter Steilpass , von dem man hofft , dass die Seeburger oder Groß-Glienicker Mitspieler ihn aufnehmen können . Ein Befreiungsschlag ist es allerdings nicht , weil es vorerst keine Gefahr fürs Dallgower Tor gab . Die Seeburger und einige Groß-Glienicker haben den Ball erst zurückgespielt und dann um so drängender wieder gefordert . Nun sollen sie zeigen , wie sie die Chance verwerten . Eine Diskussion , wo künftig die Trainerkabine stehen soll , wäre in der jetzigen Spielsituation verheerend . Und eine Parallele zu den deutschen Grotten-Kickern gibt es immer noch . Auch wenn die Spieler aus den verschiedenen Vereinen zusammengewürfelt sind , sie müssen sich daran gewöhnen , dass sie nun in einer Mannschaft " Döberitzer Heide " spielen . Und das heißt gemeinsam und nicht gegeneinander . Ermahnungen von der Seitenlinie , miteinander fair umzugehen und sich nicht beim kleinsten Schubser gegenseitig zu zerfleischen , sind normalerweise überflüssig . Vorerst allerdings hilfreich .

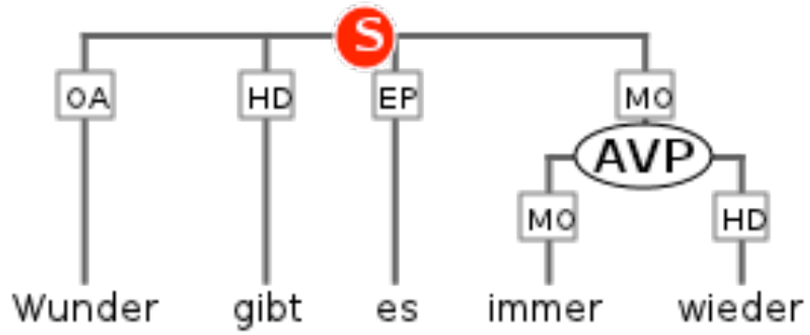
„Steilpass“ – Märkische Allgemeine Zeitung, 12.10.2001
Potsdam Commentary Corpus (Stede, 2004)

What's an annotation?

classification and interpretation of the corpus data

additional data to enrich the corpus

Wunder	gibt	es	immer	wieder	!
Wunder	geben	es	immer	wieder	!
Acc.Pl.Neut	3.Sg.Pres.Ind	3.Nom.Sg.Neut	--	--	--
NN	VVFIN	PPER	ADV	ADV	$.



Steilpass Wunder gibt es immer wieder ! Erst spielen die Dallgower Gemeindevertreter so statisch und verzagt wie die deutsche Abwehrreihe der Fußballkicker . Und dann kommt aus der Tiefe solch ein fulminanter Steilpass , von dem man hofft , dass die Seeburger oder Groß-Glienicker Mitspieler ihn aufnehmen können . Ein Befreiungsschlag ist es allerdings nicht , weil es vorerst keine Gefahr fürs Dallgower Tor gab . Die Seeburger und einige Groß-Glienicker haben den Ball erst zurückgespielt und dann um so drängender wieder gefordert . Nun sollen sie zeigen , wie sie die Chance verwerten . Eine Diskussion , wo künftig die Trainerkabine stehen soll , wäre in der jetzigen Spielsituation verheerend . Und eine Parallele zu den deutschen Grotten-Kickern gibt es immer noch . Auch wenn die Spieler aus den verschiedenen Vereinen zusammengewürfelt sind , sie müssen sich daran gewöhnen , dass sie nun in einer Mannschaft " Döberitzer Heide " spielen . Und das heißt gemeinsam und nicht gegeneinander . Ermahnungen von der Seitenlinie , miteinander fair umzugehen und sich nicht beim kleinsten Schubser gegenseitig zu zerfleischen , sind normalerweise überflüssig . Vorerst allerdings hilfreich .

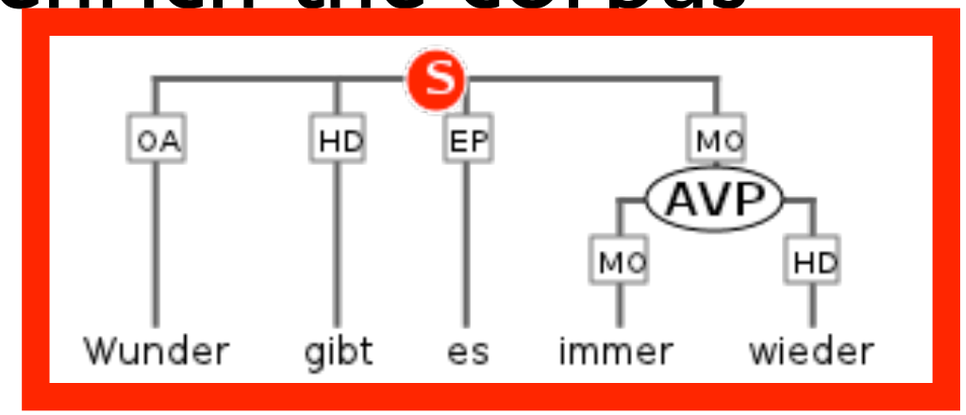
„Steilpass" – Märkische Allgemeine Zeitung, 12.10.2001
Potsdam Commentary Corpus (Stede, 2004)

What's an annotation?

classification and interpretation of the corpus data

additional data to enrich the corpus

Wunder	gibt	es	immer	wieder	!
Wunder	geben	es	immer	wieder	!
Acc.Pl.Neut	3.Sg.Pres.Ind	3.Nom.Sg.Neut	--	--	--
NN	VVFIN	PPER	ADV	ADV	$.



Steilpass Wunder gibt es immer wieder ! Erst spielen die Dallgower Gemeindevertreter so statisch und verzagt wie die deutsche Abwehrreihe der Fußballkicker . Und dann kommt aus der Tiefe solch ein fulminanter Steilpass , von dem man hofft , dass die Seeburger oder Groß-Glienicker Mitspieler ihn aufnehmen können . Ein Befreiungsschlag ist es allerdings nicht , weil es vorerst keine Gefahr fürs Dallgower Tor gab . Die Seeburger und einige Groß-Glienicker haben den Ball erst zurückgespielt und dann um so drängender wieder gefordert . Nun sollen sie zeigen , wie sie die Chance verwerten . Eine Diskussion , wo künftig die Trainerkabine stehen soll , wäre in der jetzigen Spielsituation verheerend . Und eine Parallele zu den deutschen Grotten-Kickern gibt es immer noch . Auch wenn die Spieler aus den verschiedenen Vereinen zusammengewürfelt sind , sie müssen sich daran gewöhnen , dass sie nun in einer Mannschaft " Döberitzer Heide " spielen . Und das heißt gemeinsam und nicht gegeneinander . Ermahnungen von der Seitenlinie , miteinander fair umzugehen und sich nicht beim kleinsten Schubser gegenseitig zu zerfleischen , sind normalerweise überflüssig . Vorerst allerdings hilfreich .

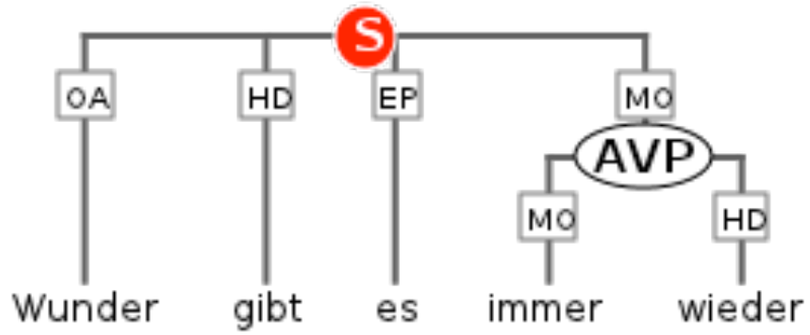
„Steilpass“ – Märkische Allgemeine Zeitung, 12.10.2001
Potsdam Commentary Corpus (Stede, 2004)

What's an annotation?

classification and interpretation of the corpus data

additional data to enrich the corpus

Wunder	gibt	es	immer	wieder	!
Wunder	geben	es	immer	wieder	!
Acc.Pl.Neut	3.Sg.Pres.Ind	3.Nom.Sg.Neut	--	--	--
NN	VVFIN	PPER	ADV	ADV	$.



Steilpass Wunder gibt es immer wieder ! Erst spielen die Dallgower Gemeindevertreter so statisch und verzagt wie die deutsche Abwehrreihe der Fußballkicker . Und dann kommt aus der Tiefe solch ein fulminanter Steilpass , von dem man hofft , dass die Seeburger oder Groß-Glienicker Mitspieler ihn aufnehmen können . Ein Befreiungsschlag ist es allerdings nicht , weil es vorerst keine Gefahr fürs Dallgower Tor gab . Die Seeburger und einige Groß-Glienicker haben den Ball erst zurückgespielt und dann um so drängender wieder gefordert . Nun sollen sie zeigen , wie sie die Chance verwerten . Eine Diskussion , wo künftig die Trainerkabine stehen soll , wäre in der jetzigen Spielsituation verheerend . Und eine Parallele zu den deutschen Grotten-Kickern gibt es immer noch . Auch wenn die Spieler aus den verschiedenen Vereinen zusammengewürfelt sind , sie müssen sich daran gewöhnen , dass sie nun in einer Mannschaft " Döberitzer Heide " spielen . Und das heißt gemeinsam und nicht gegeneinander . Ermahnungen von der Seitenlinie , miteinander fair umzugehen und sich nicht beim kleinsten Schubser gegenseitig zu zerfleischen , sind normalerweise überflüssig . Vorerst allerdings hilfreich .

„Steilpass“ – Märkische Allgemeine Zeitung, 12.10.2001
Potsdam Commentary Corpus (Stede, 2004)

Annis

Annis² Corpus Search

ANNIS² Tutorial — logged in as "viktor"

Search Form

AnnisQL: cat="S" & "Wunder" & #1 _i_ #2

Show Result | Query Builder | History

Result: 1

More Corpora

Name	Texts	Tokens
FalkoEssayL1v2.2	95	70648
FalkoEssayL2v2.2	248	132069
TueBa-D/Z.6.0	2777	975836
b1.aja	117	1452
b2.tangale	49	452
d1.pcc2	2	399
d2.20samplesDEU	22	382
pcc-176	176	33222
ridges.herbology	14	63734
x.ontonotes1.0	597	370789
x.ontonotes1.0_test...	60	38775
x.tiger2	1971	888578

Search | Export

Context Left: 5

Context Right: 5

Results Per Page: 10

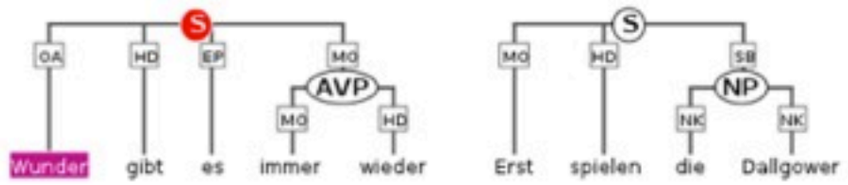
Search Result - cat="S" & "Wunder" & #1 _i_ #2 (5, 5)

Page 1 of 1 | Token Annotations | Show Citation URL | Document Path — Displaying Results 1 - 1 of 1

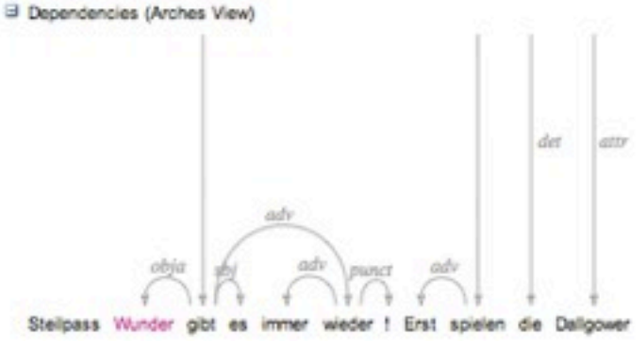
Path: d1.pcc2 > 4282

Steilpass	Wunder	gibt	es	immer wieder	!	Erst	spielen	die	Dallgower
Steilpass	Wunder	geben	es	immer wieder	!	erst	spielen	der	Dallgower
Nom.Sg.Masc	Acc.Pl.Neut	3.Sg.Pres.Ind	3.Nom.Sg.Neut	--	--	--	3.Pl.Pres.Ind	Nom.Pl.Masc	Pos.*.*.*
NN	NN	VVFIN	PPER	ADV	ADV	$. ADV	VVFIN	ART	ADJA

Constituents (Tree View)



Dependencies (Arches View)



Information Structure (Grid View)
Discourse Referents (Grid View)
Rhetorical Structure (Old Grid View)
Coreference (Discourse View)

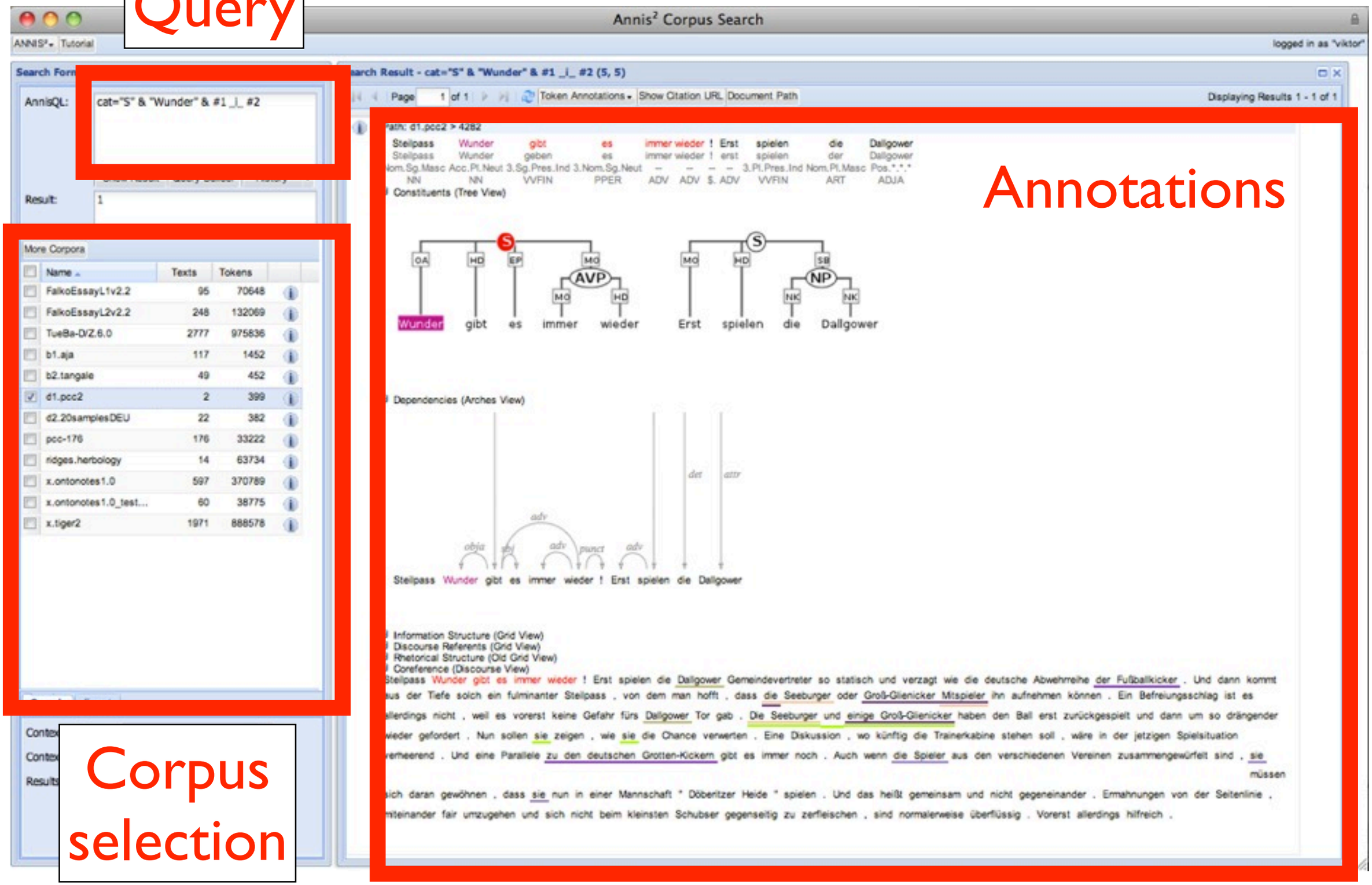
Steilpass Wunder gibt es immer wieder ! Erst spielen die Dallgower Gemeindevertreter so statisch und verzagt wie die deutsche Abwehrreihe der Fußballkicker . Und dann kommt aus der Tiefe solch ein fulminanter Steilpass , von dem man hofft , dass die Seeburger oder Groß-Glienicker Mitspieler ihn aufnehmen können . Ein Befreiungsschlag ist es allerdings nicht , weil es vorerst keine Gefahr fürs Dallgower Tor gab . Die Seeburger und einige Groß-Glienicker haben den Ball erst zurückgespielt und dann um so drängender wieder gefordert . Nun sollen sie zeigen , wie sie die Chance verwerten . Eine Diskussion , wo künftig die Trainerkabine stehen soll , wäre in der jetzigen Spielsituation verheerend . Und eine Parallele zu den deutschen Grotten-Kickern gibt es immer noch . Auch wenn die Spieler aus den verschiedenen Vereinen zusammengewürfelt sind , sie müssen sich daran gewöhnen , dass sie nun in einer Mannschaft " Döberitzer Heide " spielen . Und das heißt gemeinsam und nicht gegeneinander . Ermahnungen von der Seitenlinie , miteinander fair umzugehen und sich nicht beim kleinsten Schubser gegenseitig zu zerfleischen , sind normalerweise überflüssig . Vorerst allerdings hilfreich .

Annis

Query

Annotations

Corpus selection

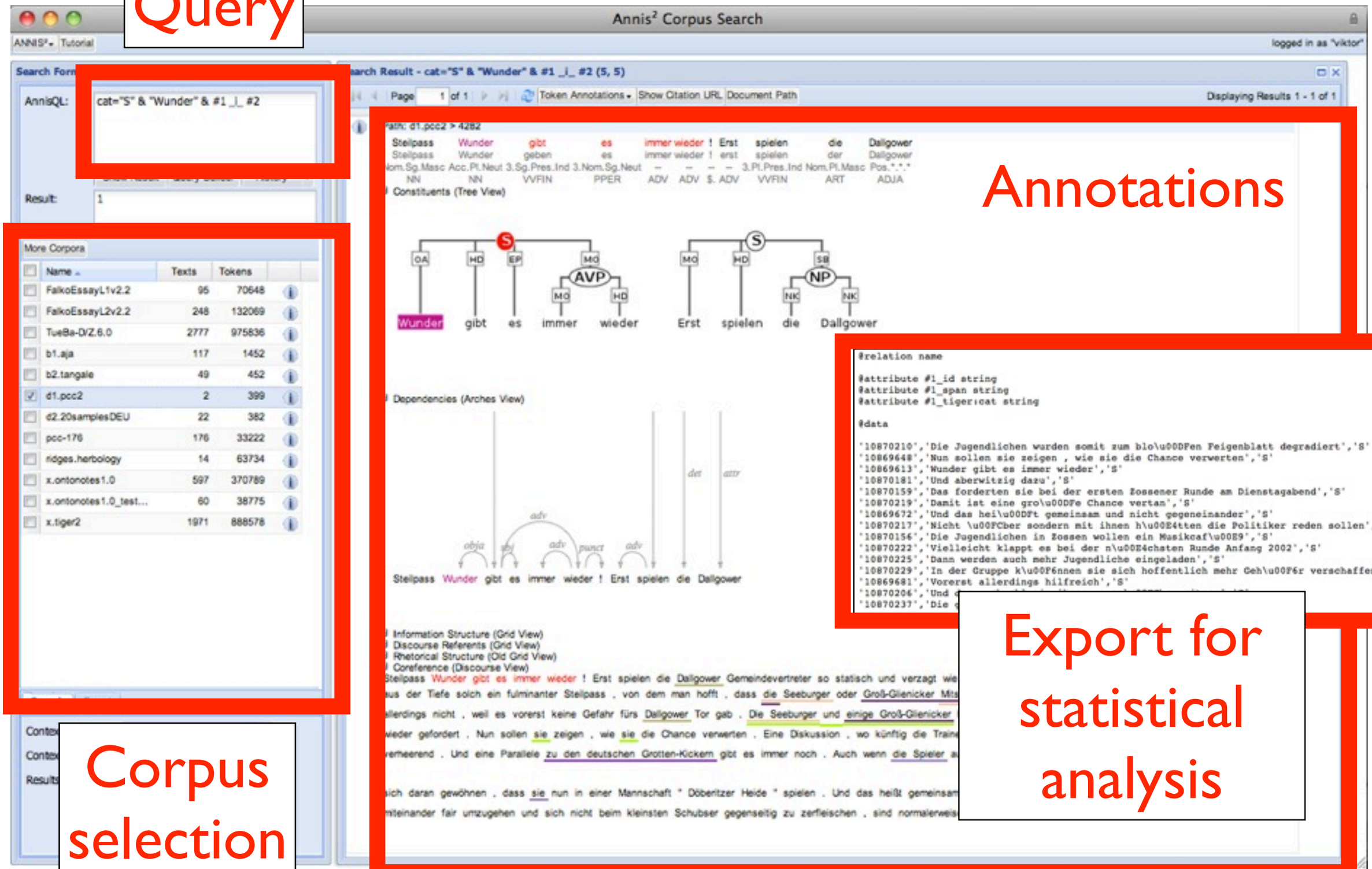


Annis

Universität Potsdam

Humboldt-Universität zu Berlin

Query



Annotations

Corpus selection

Export for statistical analysis

Annis query language

```
cat="S" &
"Wunder" &
#1 _i_ #2
```

find a sentence

and find the phrase “Wunder”

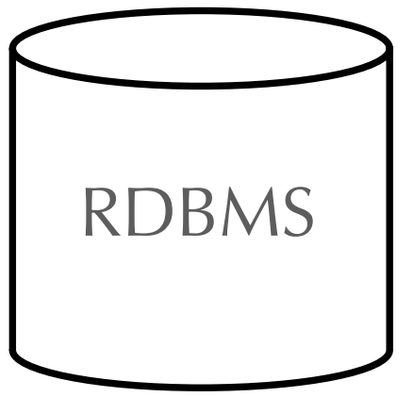
the sentence includes the phrase “Wunder”

Annis query language

```
cat="S" &
"Wunder" &
#1 _i_ #2
```

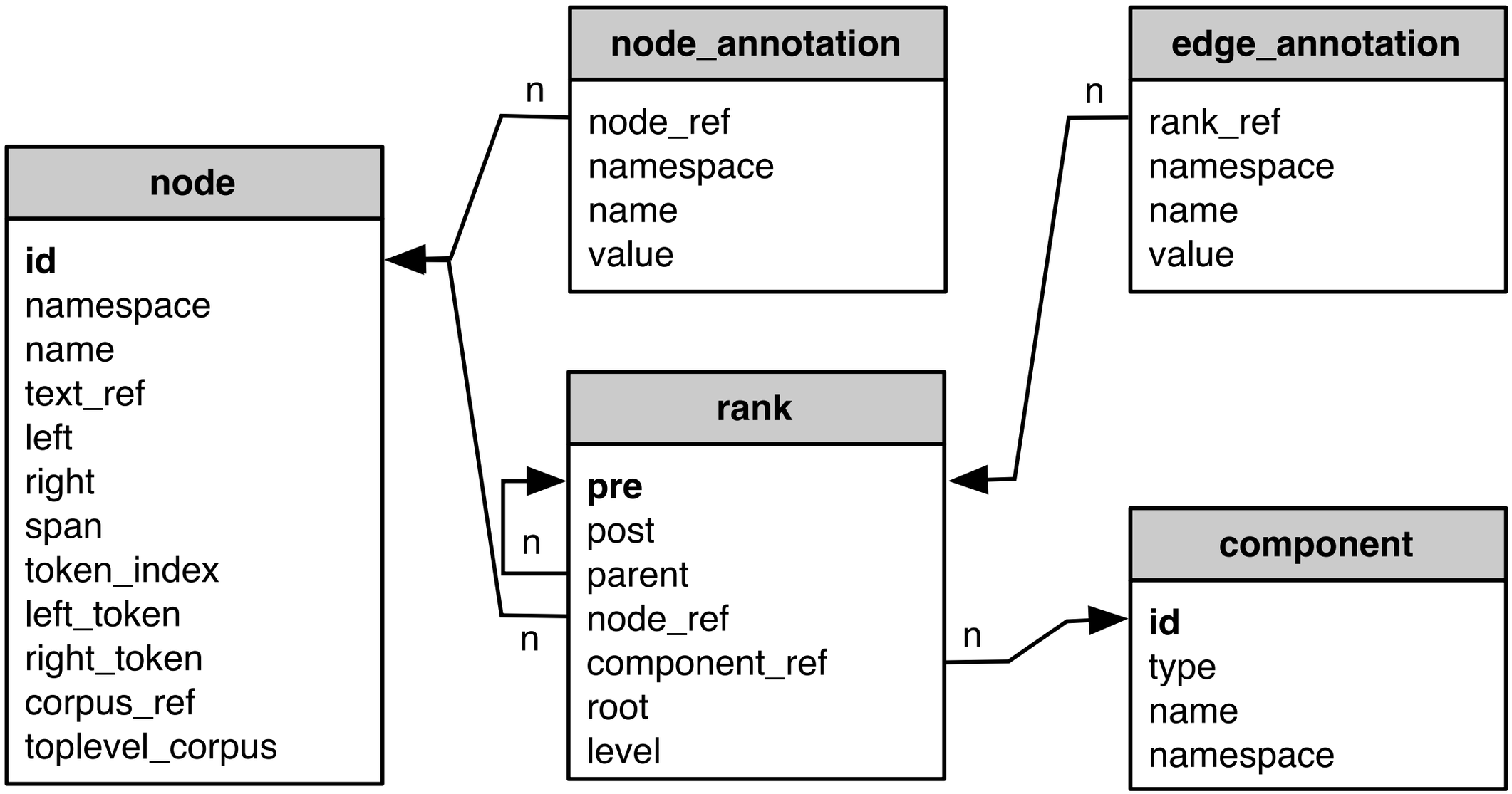
find a sentence
and find the phrase “Wunder”
the sentence includes the phrase “Wunder”

```
SELECT id1, id2
FROM   ...
WHERE  ...
```

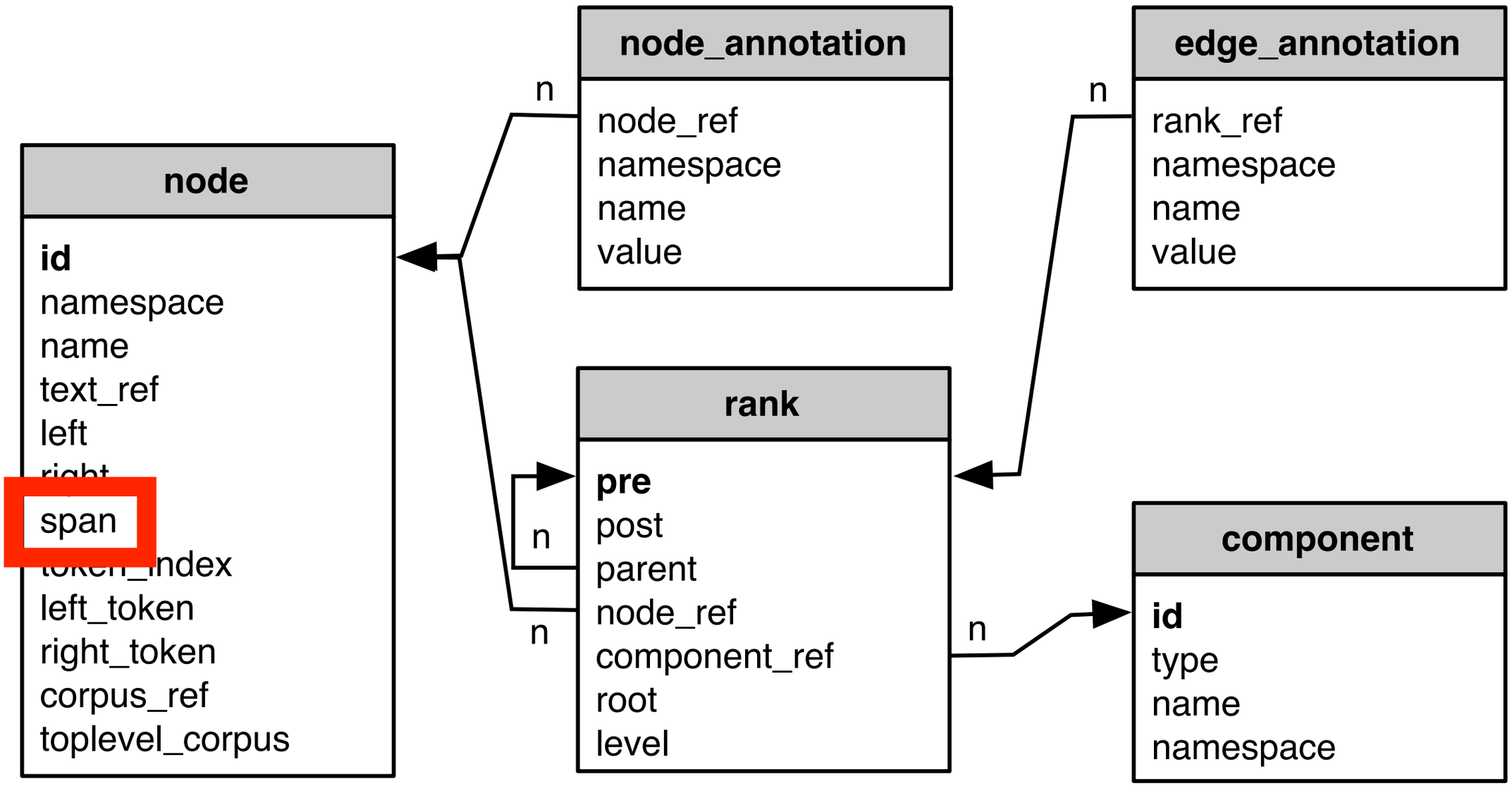


2. Current implementation on PostgreSQL

Database schema

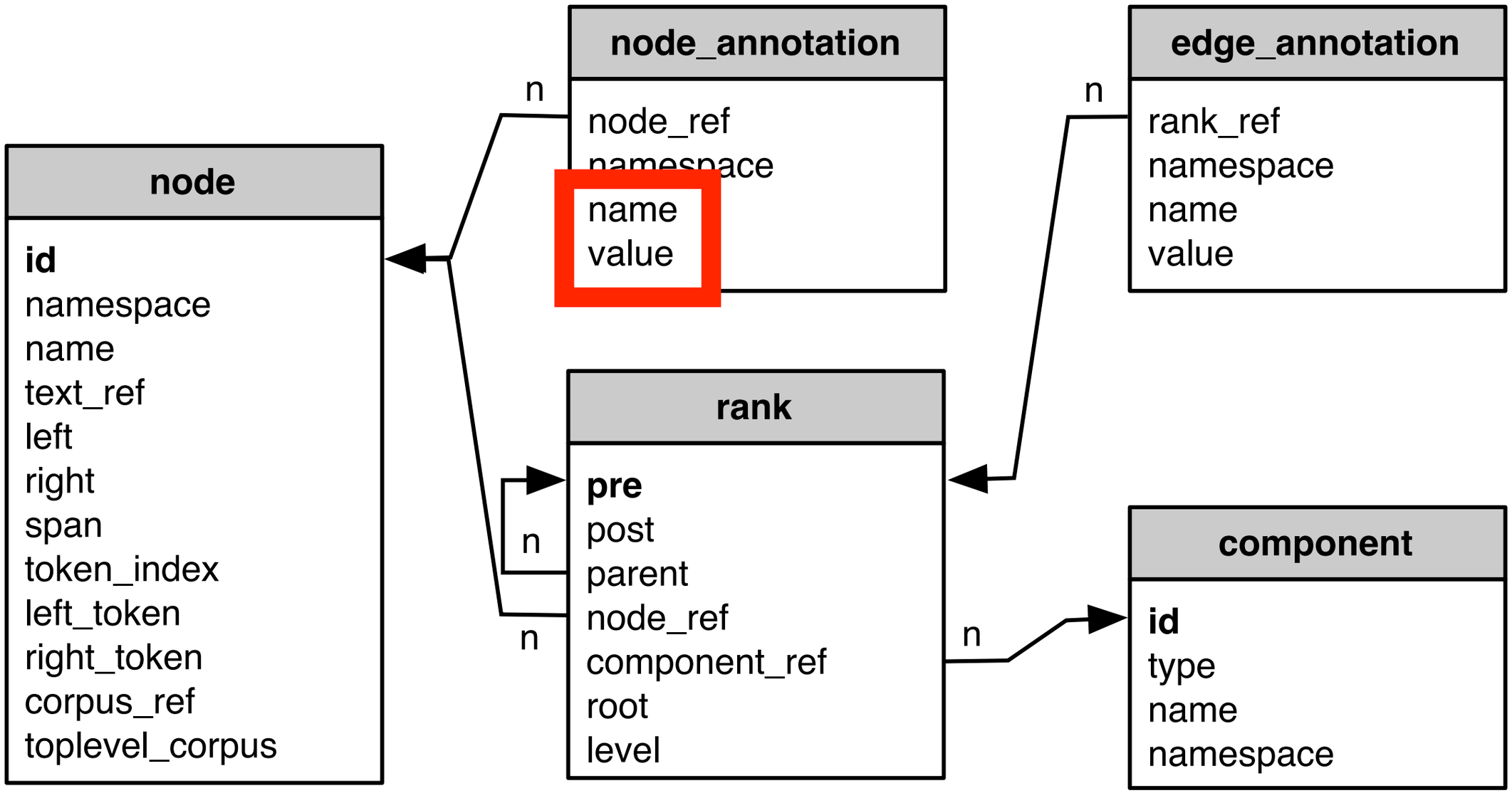


Example 1: Text search



AQL: `"Wunder"` ⇒ SQL: `node`*N*`.span = 'Wunder'`

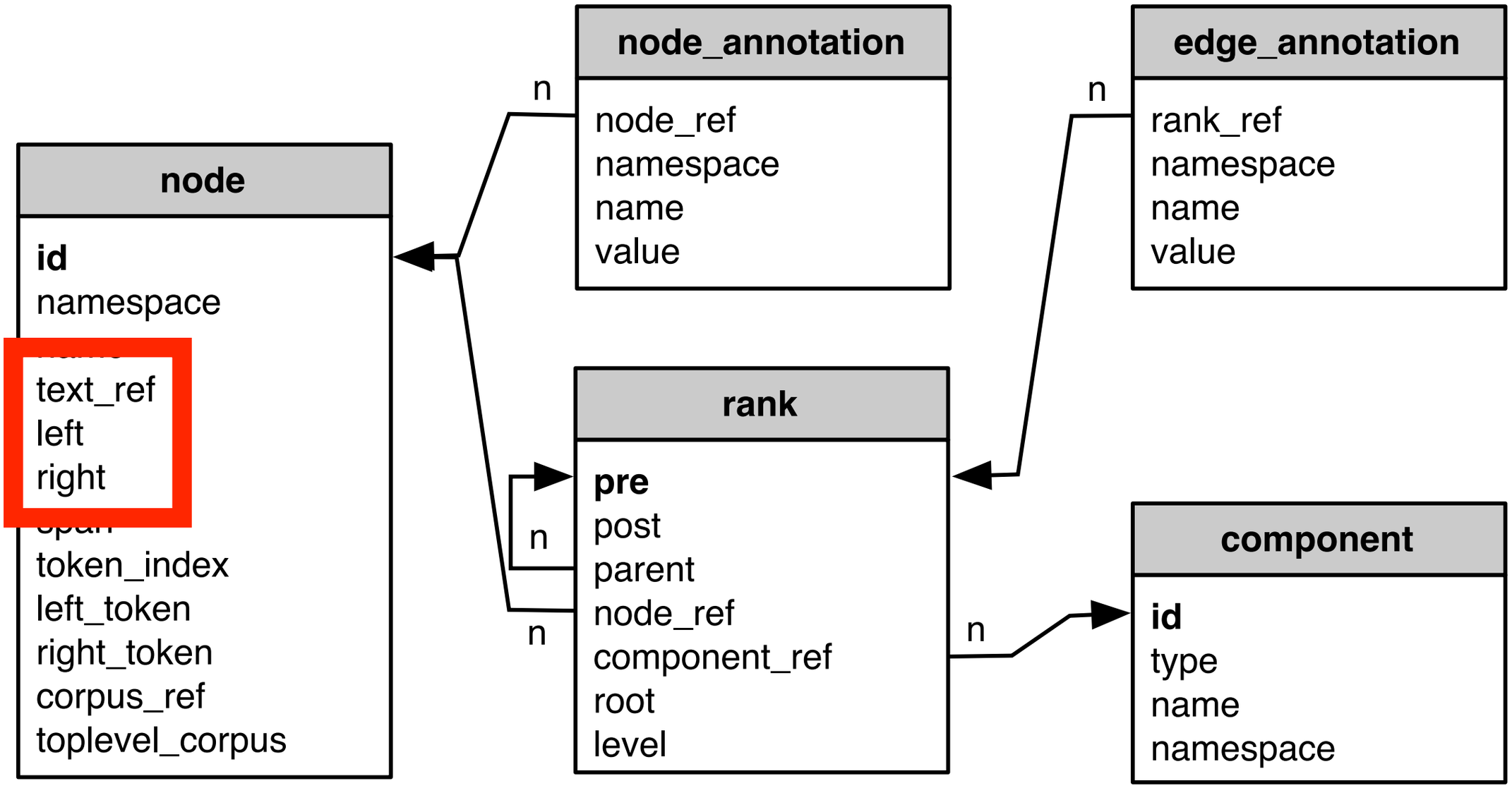
Example 2: Annotation search



cat="S"

```
node_annotationN.name = 'cat'
node_annotationN.value = 'S'
```

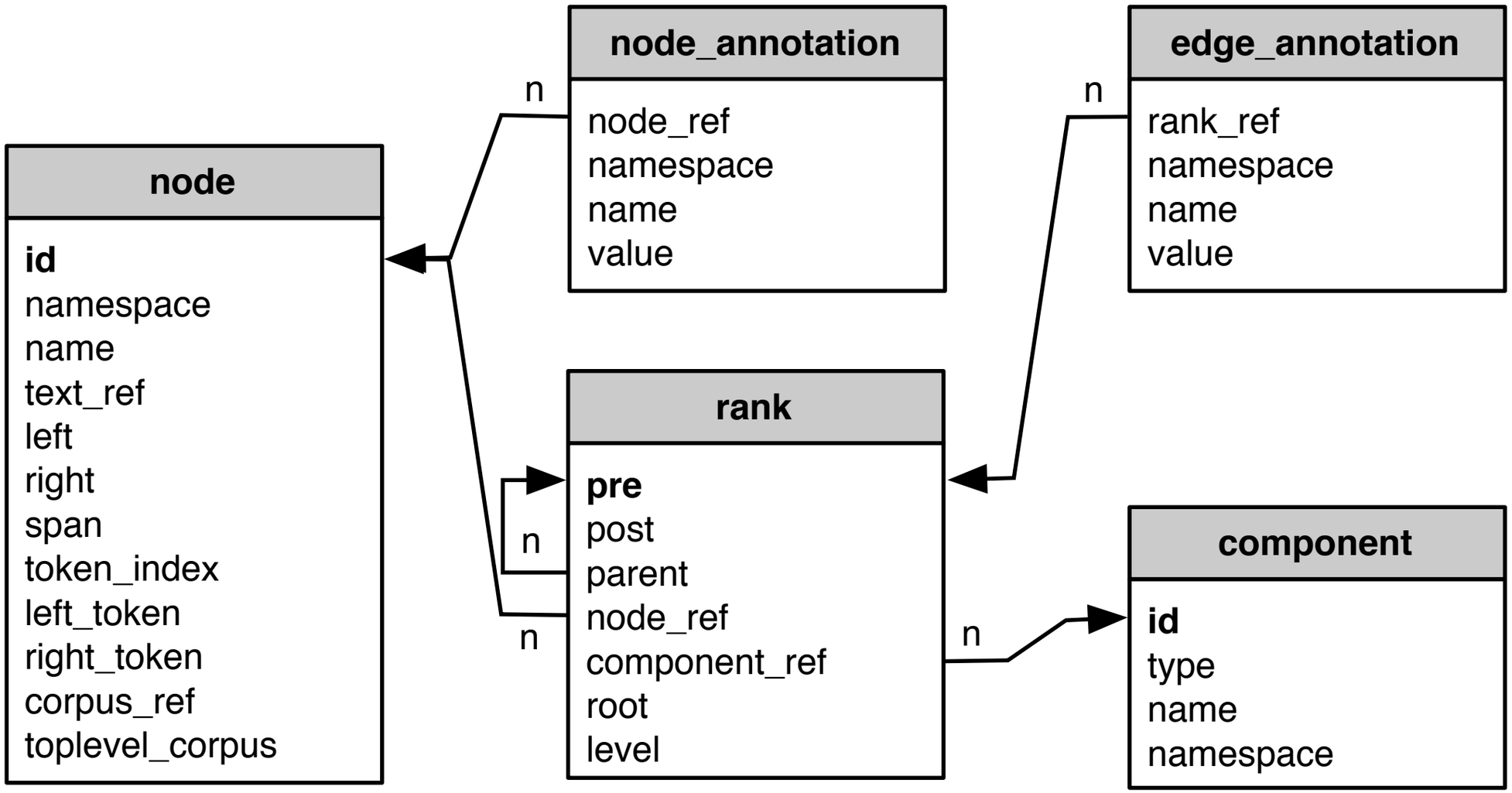
Example 3: Inclusion operator



#1 _i_ #2

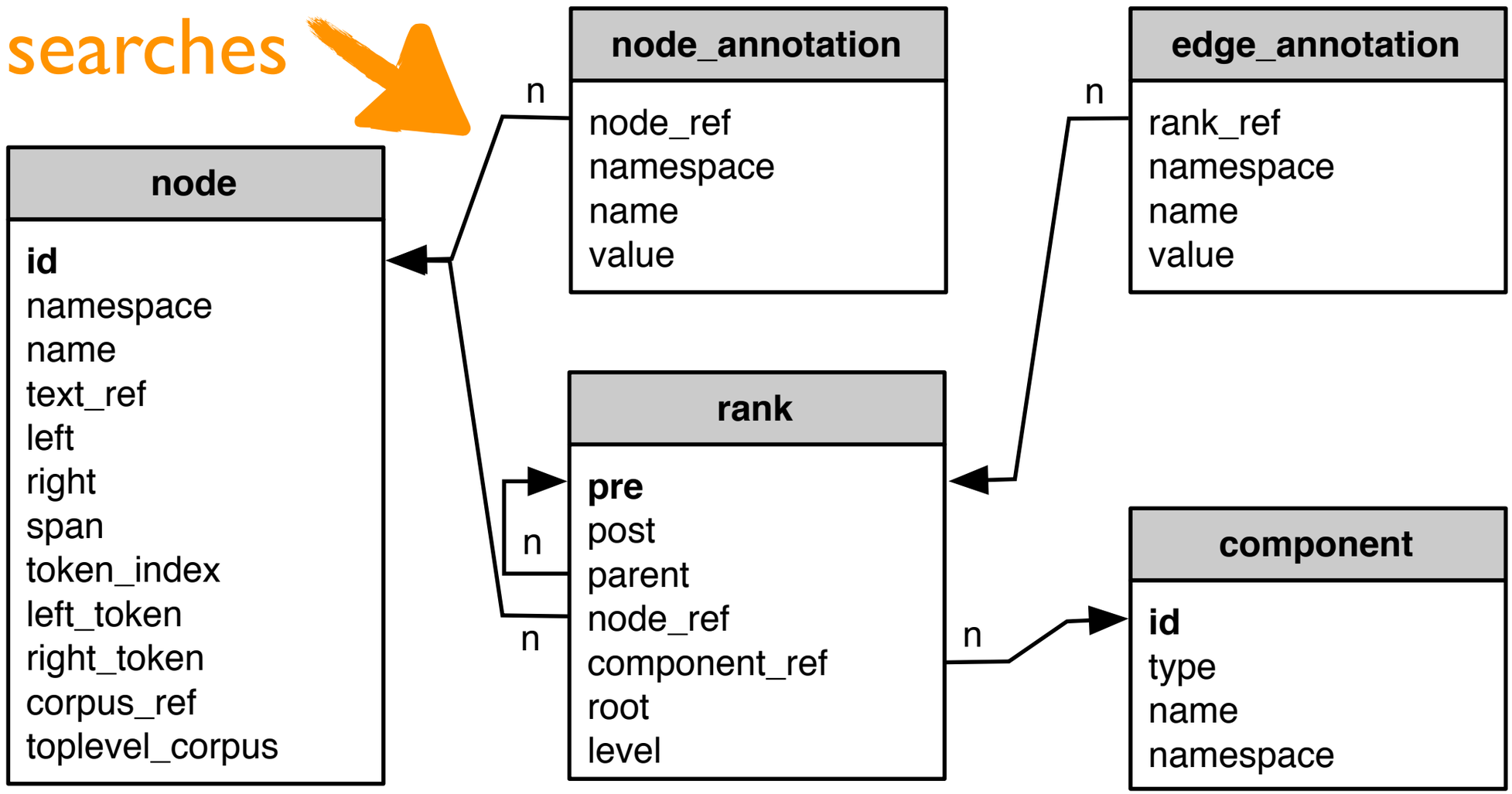
```
node1.text_ref = node2.text_ref
node1.right <= node2.right
node1.left >= node2.left
```

Many tables – Many joins



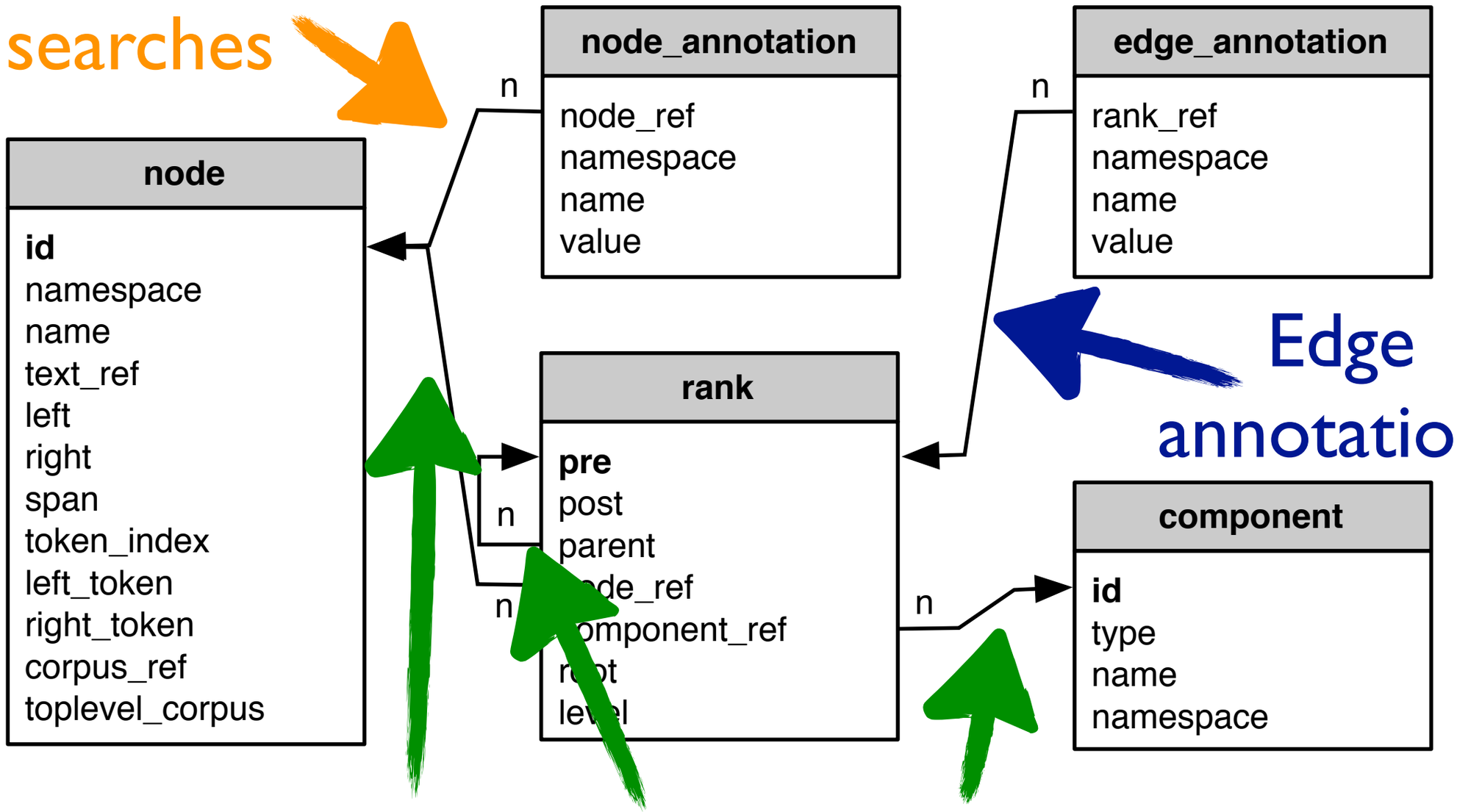
Many tables – Many joins

Annotation searches



Many tables – Many joins

Annotation searches

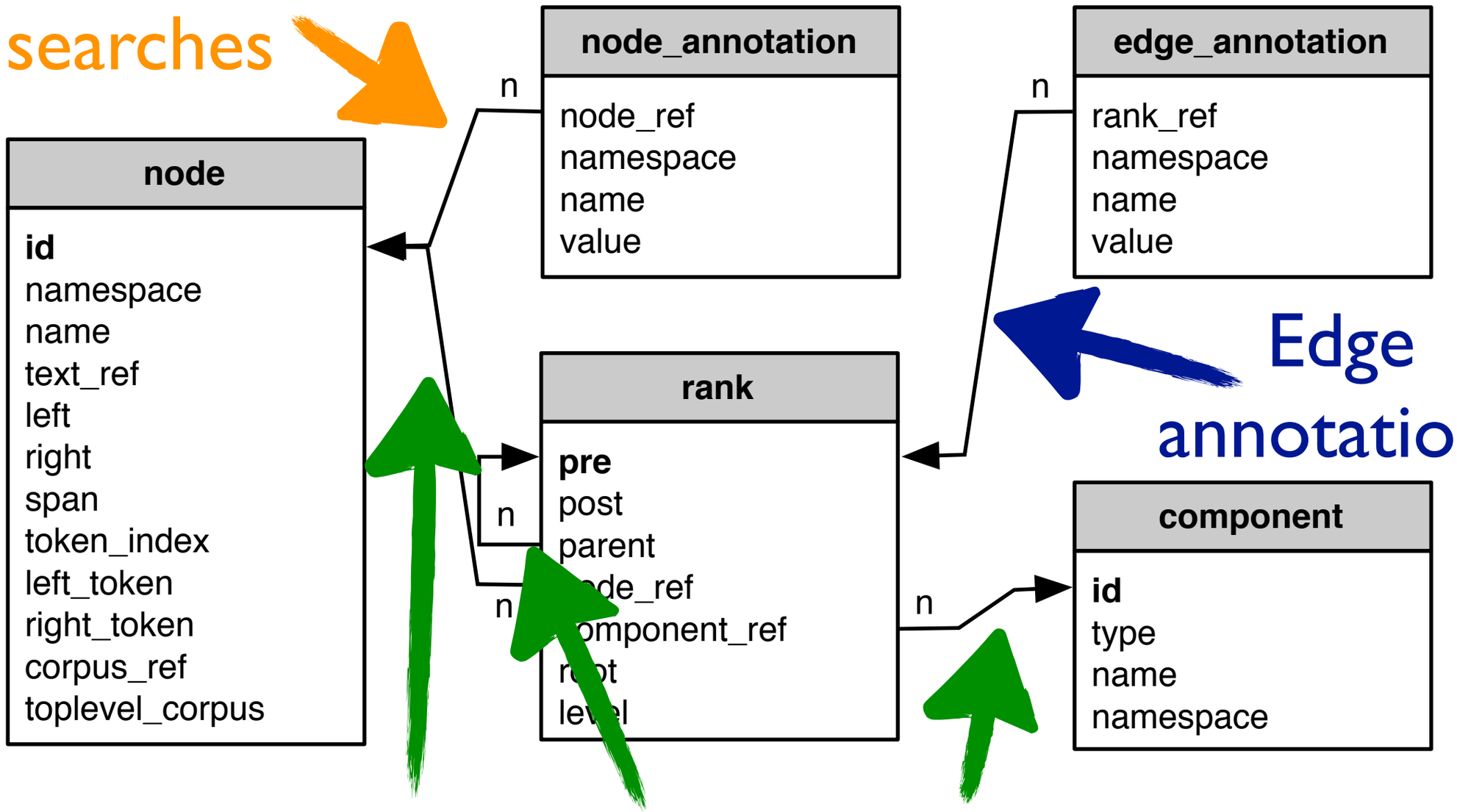


Edge annotations

Binary relations on edges

Many tables – Many joins

Annotation searches

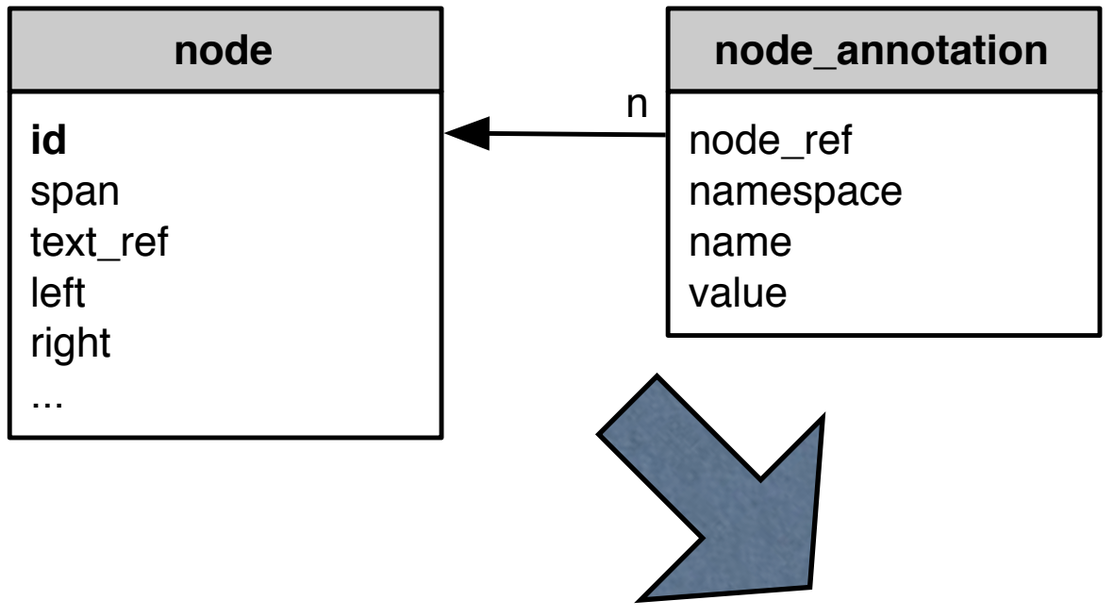


Edge annotations

Binary relations on edges

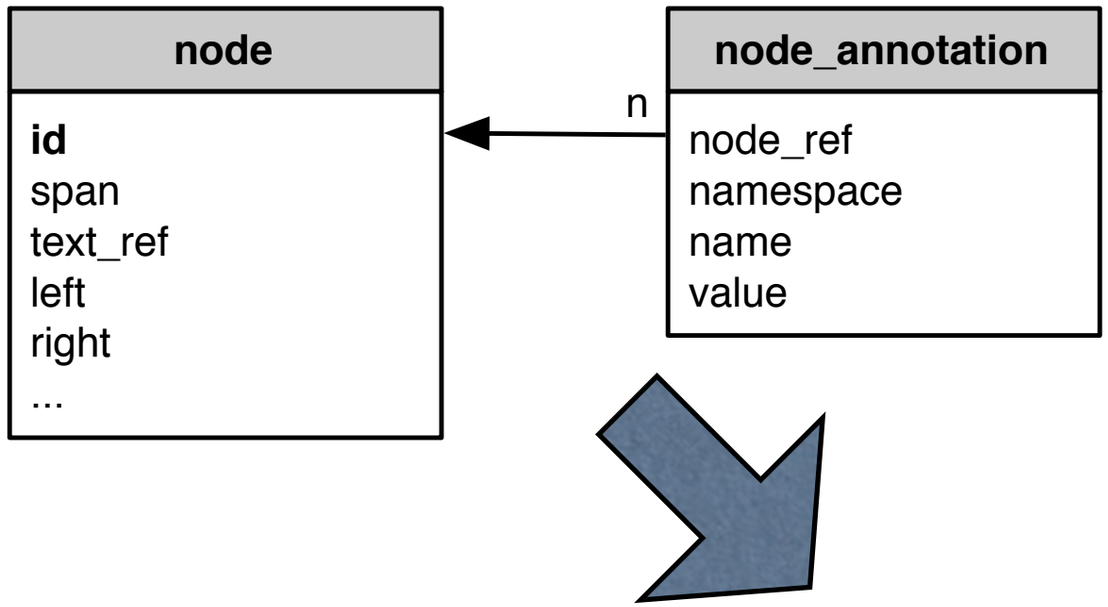
Bad performance on PostgreSQL

Solution 1: One big table



id	span	text_ref	left	right	...	na_name	na_value	...
1		1	1	30		cat	S	
2	Wunder	1	1	5		morph	Acc.Pl.Neut	
2	Wunder	1	1	5		pos	NN	
2	Wunder	1	1	5		lemma	Wunder	
...								

Solution 1: One big table



id	span	text_ref	left	right	...	na_name	na_value	...
1		1	1	30		cat	S	
2	Wunder	1	1	5		morph	Acc.Pl.Neut	
2	Wunder	1	1	5		pos	NN	
2	Wunder	1	1	5		lemma	Wunder	
...								

Pro: Fewer joins

Contra: Increased redundancy, less extensible

Solution 2: Combined indexes

id	span	text_ref	left	right	...	na_name	na_value	...
1		1	1	30		cat	S	
2	Wunder	1	1	5		morph	Acc.Pl.Neut	
2	Wunder	1	1	5		pos	NN	
2	Wunder	1	1	5		lemma	Wunder	
...								

One index over 4 columns

Find nodes

- spanning a certain word,
- in a certain text,
- at a certain position.

```
cat="S" &
"Wunder" &
#1 _i_ #2
```

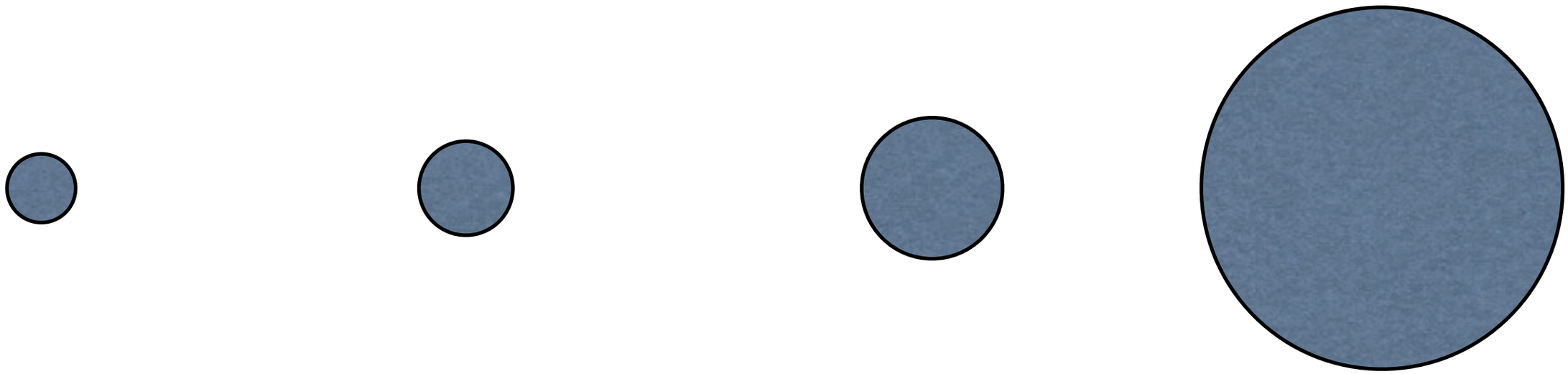
Pro: Potentially very fast

Contra: Uses lots of disk space

Disk usage in PostgreSQL

TIGER Treebank 2.1

ca. 50.000 sentences, 900.000 tokens,
3 million annotations, 1 million edges

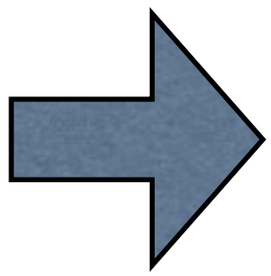


280 MB
Data files

525 MB
Normalized
(many tables)

1.2 GB
Materialized
(one table)

7.7 GB
Materialized
+ Indexes



Increase by factor 15 (or almost 30)

3. What are Column-Stores? How can Annis benefit?

What's a Column-Store?

conceptual model

	node_ref	name	value
1	123	pos	VVINF
2	123	lemma	essen
3	456	pos	NN

table

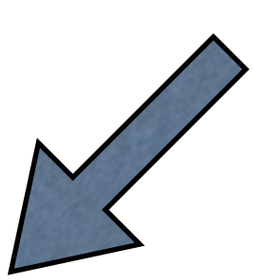
storage model

What's a Column-Store?

conceptual model

	node_ref	name	value
1	123	pos	VVINF
2	123	lemma	essen
3	456	pos	NN

table



storage model

1	123	pos	VVINF
2	123	lemma	essen
3	456	pos	NN

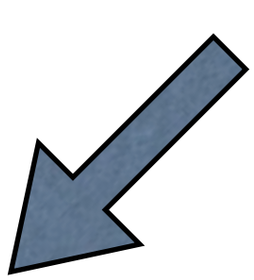
rows

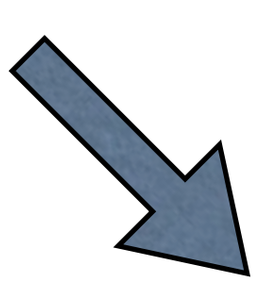
What's a Column-Store?

conceptual model

	node_ref	name	value
1	123	pos	VVINF
2	123	lemma	essen
3	456	pos	NN

table





storage model

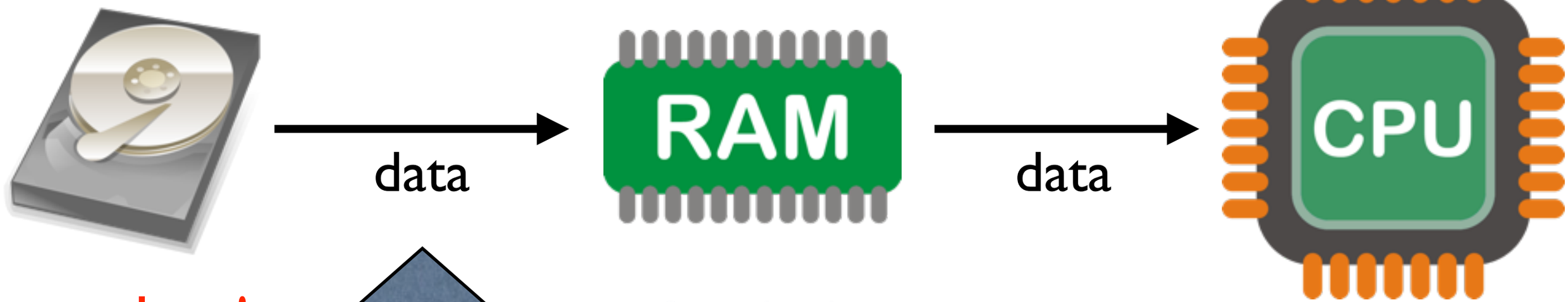
1	123	pos	VVINF
2	123	lemma	essen
3	456	pos	NN

rows

node_ref	name	value
123	pos	VVINF
123	lemma	essen
456	pos	NN

columns

Why Column-Stores? – Why Databases?



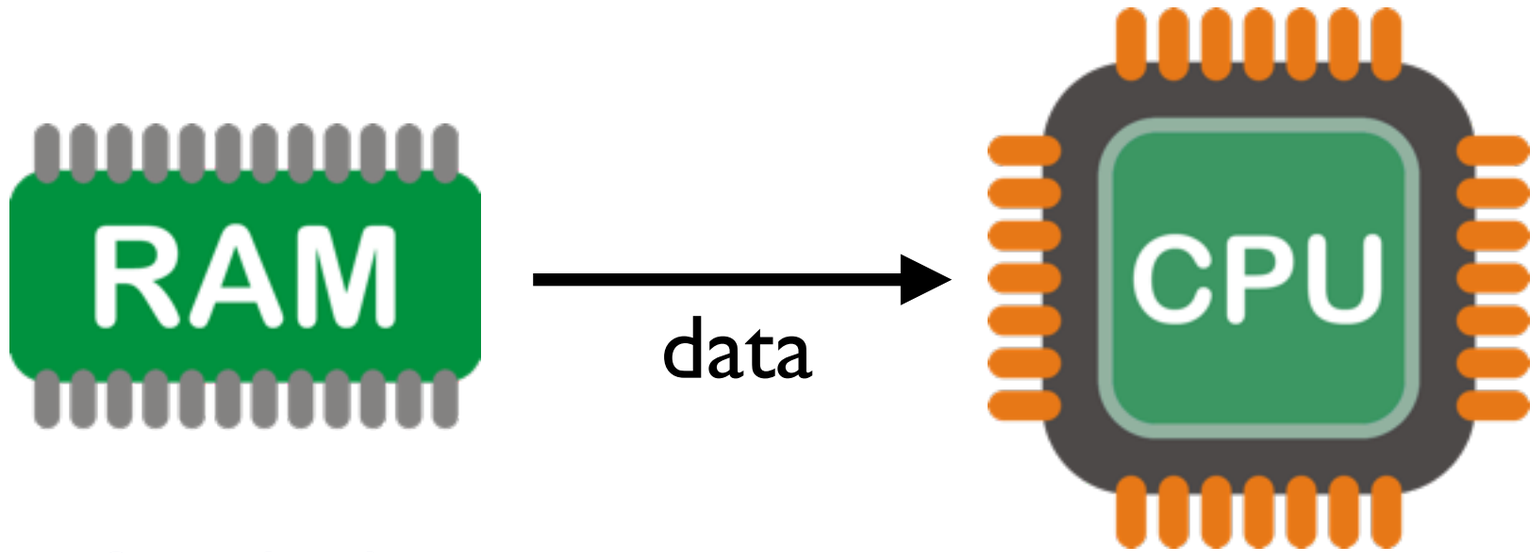
too slow!
very big
sequential

fast(er)
too small!
random

very fast

job of the database
(traditionally)

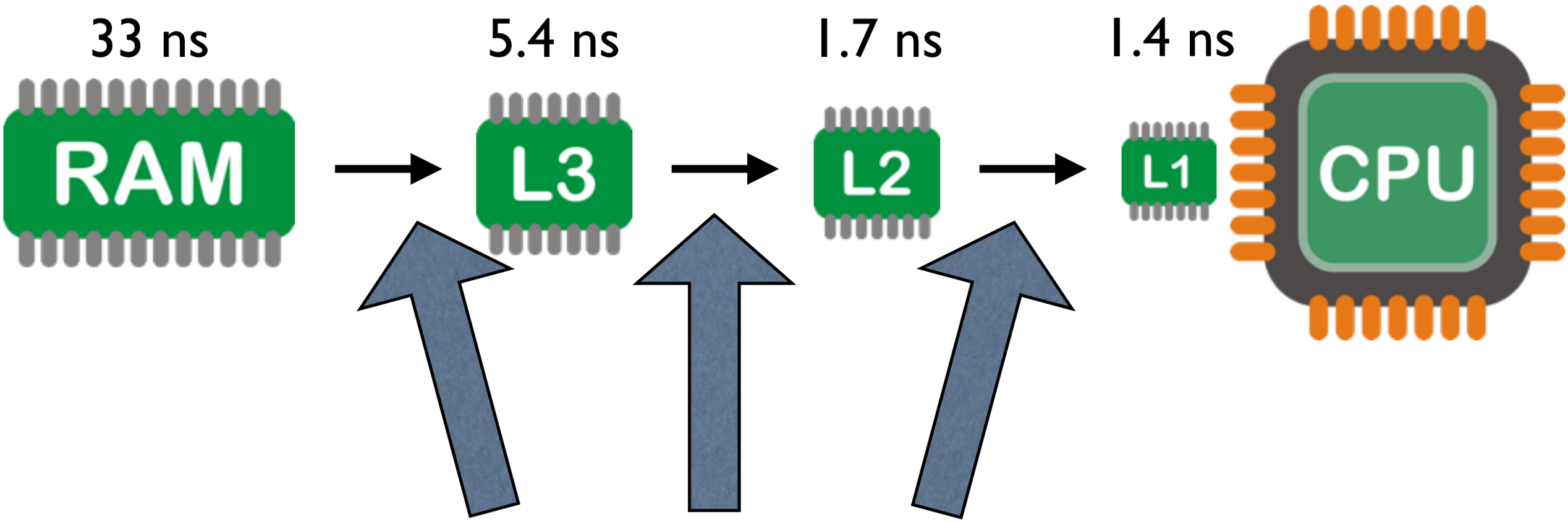
Why Column-Stores? – Why Databases?



fast(er)

very fast

Caches between RAM and CPU



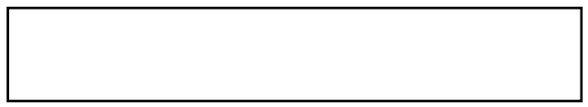
job of the database
on a modern system
(among others)

Cache usage of row layout

query: compare *name* attribute with value 'lemma'

data file:

L1 cache:



Cache usage of row layout

query: compare *name* attribute with value 'lemma'

data file:

1	123	pos	VVINF

2	123	lemma	essen

3	456	pos	NN

L1 cache:

1. load first row

Cache usage of row layout

query: compare *name* attribute with value 'lemma'

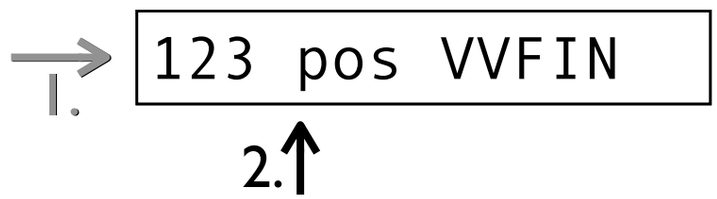
data file:

1	123	pos	VVINF

2	123	lemma	essen

3	456	pos	NN

L1 cache:



1. load first row
2. locate *name* attribute

Cache usage of row layout

query: compare *name* attribute with value 'lemma'

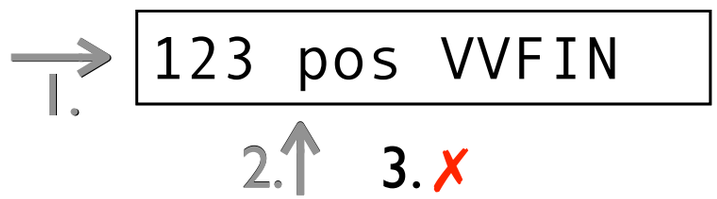
data file:

1	123	pos	VVINF

2	123	lemma	essen

3	456	pos	NN

L1 cache:



1. load first row
2. locate *name* attribute
3. test *name* attribute

Cache usage of row layout

query: compare *name* attribute with value 'lemma'

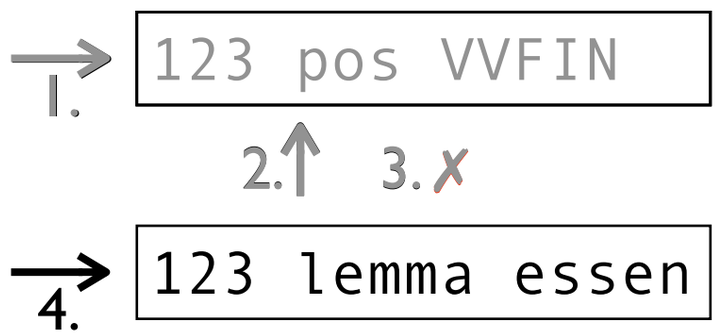
data file:

1	123	pos	VVINF

2	123	lemma	essen

3	456	pos	NN

L1 cache:



1. load first row
2. locate *name* attribute
3. test *name* attribute
4. load second row

Cache usage of row layout

query: compare *name* attribute with value 'lemma'

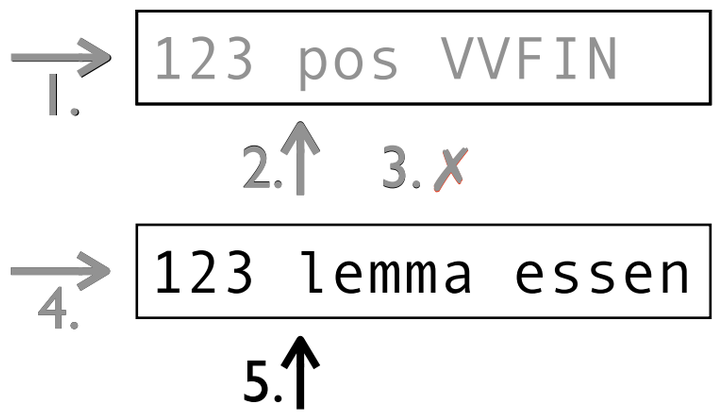
data file:

1	123	pos	VVINF

2	123	lemma	essen

3	456	pos	NN

L1 cache:



1. load first row
2. locate *name* attribute
3. test *name* attribute
4. load second row
5. locate *name* attribute

Cache usage of row layout

query: compare *name* attribute with value 'lemma'

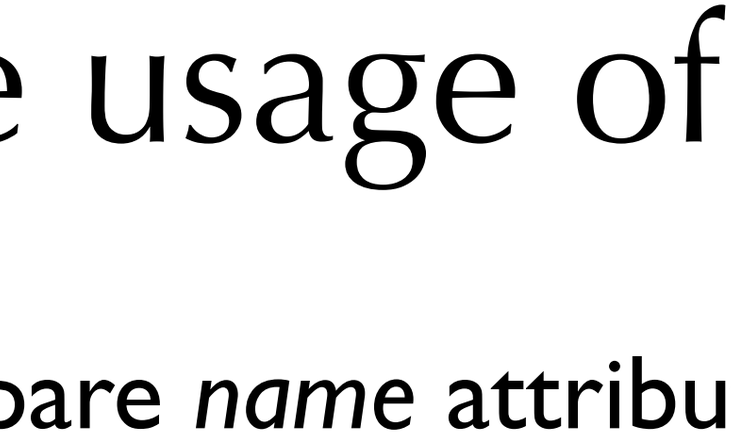
data file:

1	123	pos	VVINF

2	123	lemma	essen

3	456	pos	NN

L1 cache:



1. load first row
2. locate *name* attribute
3. test *name* attribute
4. load second row
5. locate *name* attribute
6. test *name* attribute

Cache usage of row layout

query: compare *name* attribute with value 'lemma'

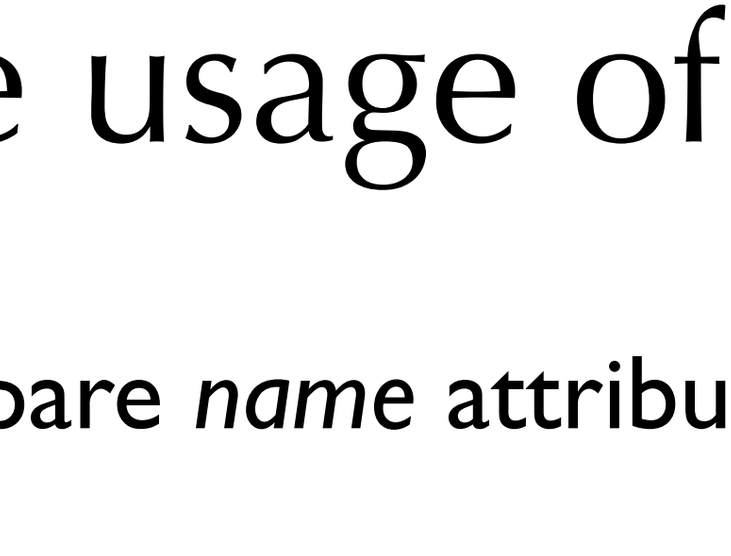
data file:

1	123	pos	VVINF

2	123	lemma	essen

3	456	pos	NN

L1 cache:



1. load first row
2. locate *name* attribute
3. test *name* attribute
4. load second row
5. locate *name* attribute
6. test *name* attribute
7. load third row

Cache usage of row layout

query: compare *name* attribute with value 'lemma'

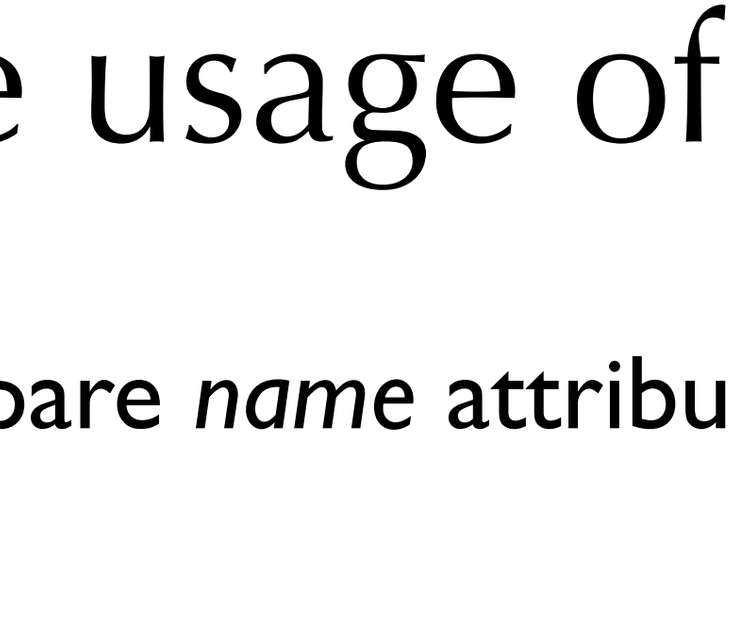
data file:

1	123	pos	VVINF

2	123	lemma	essen

3	456	pos	NN

L1 cache:



1. load first row
2. locate *name* attribute
3. test *name* attribute
4. load second row
5. locate *name* attribute
6. test *name* attribute
7. load third row
8. locate *name* attribute

Cache usage of row layout

query: compare *name* attribute with value 'lemma'

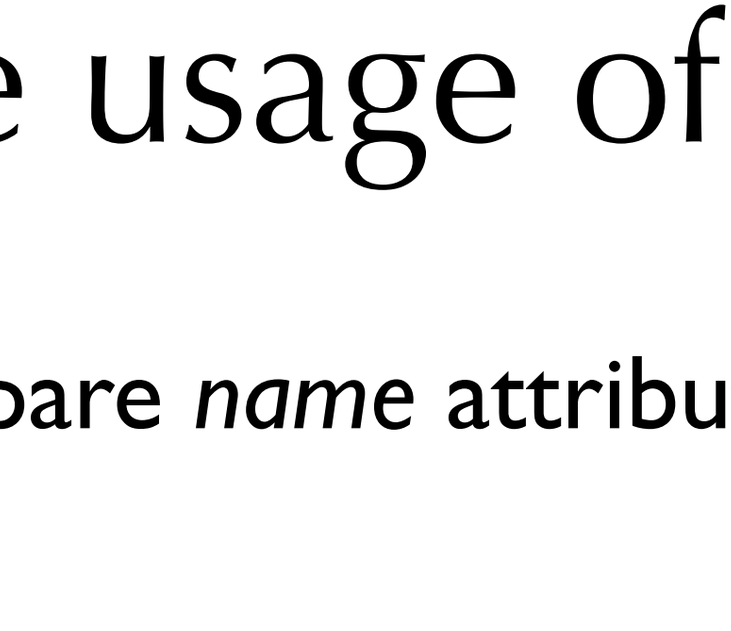
data file:

1	123	pos	VVINF

2	123	lemma	essen

3	456	pos	NN

L1 cache:



1. load first row
2. locate *name* attribute
3. test *name* attribute
4. load second row
5. locate *name* attribute
6. test *name* attribute
7. load third row
8. locate *name* attribute
9. test *name* attribute

Cache usage of row layout

query: compare *name* attribute with value 'lemma'

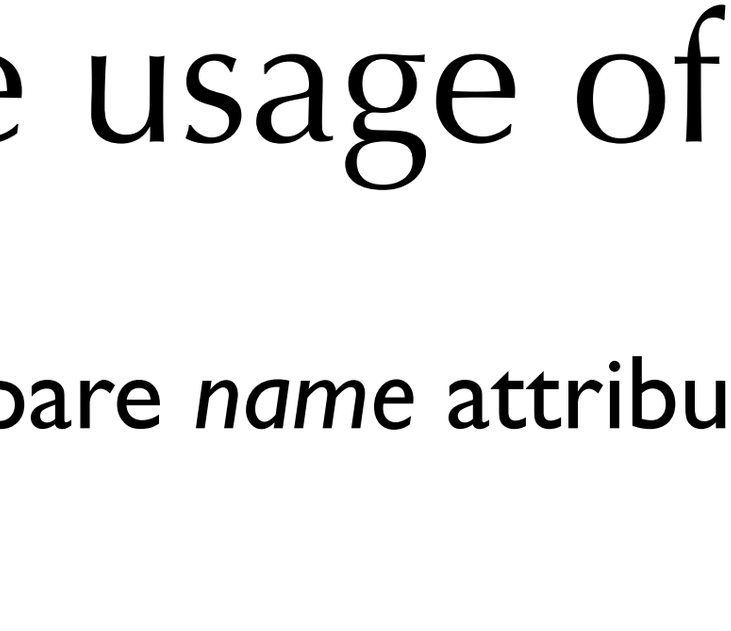
data file:

1	123	pos	VVINF

2	123	lemma	essen

3	456	pos	NN

L1 cache:



1. load first row
2. locate *name* attribute
3. test *name* attribute
4. load second row
5. locate *name* attribute
6. test *name* attribute
7. load third row
8. locate *name* attribute
9. test *name* attribute

Cache usage of column layout

query: compare *name* attribute with value 'lemma'

data file:

node_ref	name	value
123	pos	VVINF
123	lemma	essen
456	pos	NN

L1 cache:

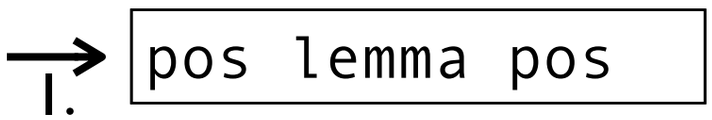
Cache usage of column layout

query: compare *name* attribute with value 'lemma'

data file:

node_ref	name	value
123	pos	VVINF
123	lemma	essen
456	pos	NN

L1 cache:



1. load *name* column

Cache usage of column layout

query: compare *name* attribute with value 'lemma'

data file:

node_ref	name	value
123	pos	VVINF
123	lemma	essen
456	pos	NN

L1 cache:


```
pos lemma pos
```

2. ✗

1. load *name* column
2. test first *name* attribute

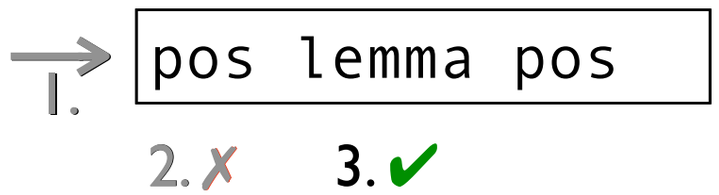
Cache usage of column layout

query: compare *name* attribute with value 'lemma'

data file:

node_ref	name	value
123	pos	VVINF
123	lemma	essen
456	pos	NN

L1 cache:



1. load *name* column
2. test first *name* attribute
3. test second *name* attribute

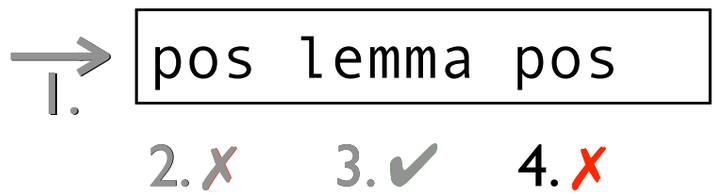
Cache usage of column layout

query: compare *name* attribute with value 'lemma'

data file:

node_ref	name	value
123	pos	VVINF
123	lemma	essen
456	pos	NN

L1 cache:



1. load *name* column
2. test first *name* attribute
3. test second *name* attribute
4. test third *name* attribute

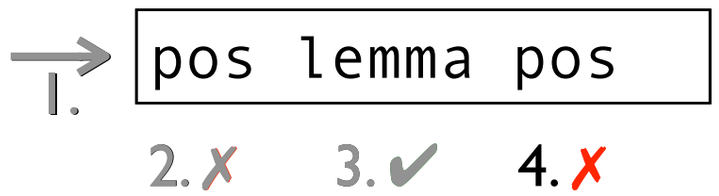
Cache usage of column layout

query: compare *name* attribute with value 'lemma'

data file:

node_ref	name	value
123	pos	VVINF
123	lemma	essen
456	pos	NN

L1 cache:



1. load *name* column
2. test first *name* attribute
3. test second *name* attribute
4. test third *name* attribute

Column operations in Annis

Search terms

- can be indexed

```
"Wunder"
```

Regular expressions

- can often be indexed
- but not always

```
morph=/.*\.Pl\.Neut/
```

Binary operations

- can be indexed
- need many indexes
- slow if there are many index lookups

```
_=_     _i_     >
```

id	span	text	ref	left	right	...
1		1		1	30	
2	Wunder	1		1	5	
2	Wunder	1		1	5	
2	Wunder	1		1	5	
...						

4. New implementation on MonetDB and evaluation

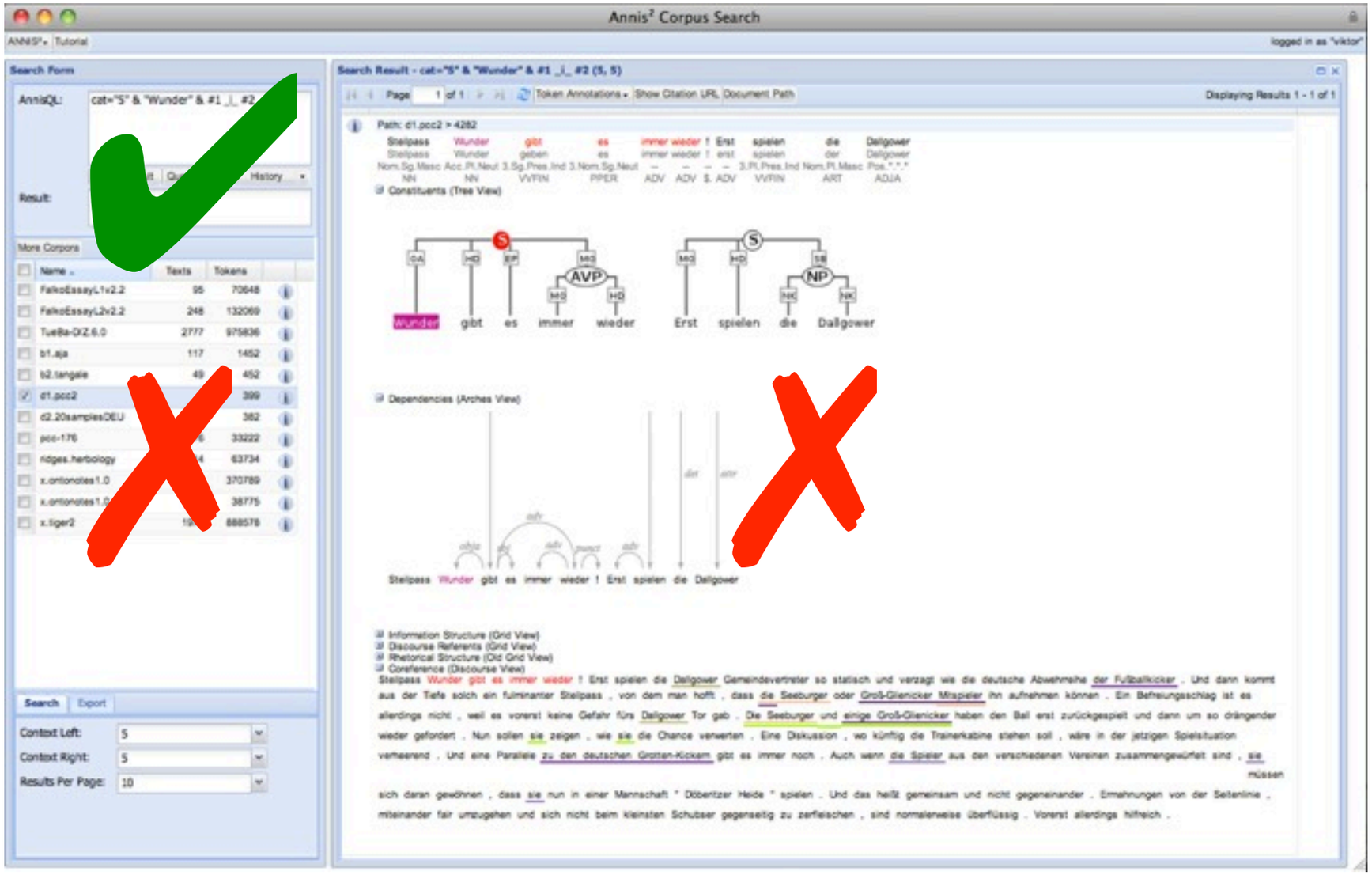
Prototype implementation

Supported

- Annis 2 Query Language
- COUNT queries

Not supported

- Annis 3 language features
- ANNOTATE, MATRIX queries
- corpus selection



Realistic test workload

Corpus: TIGER Treebank 2.1

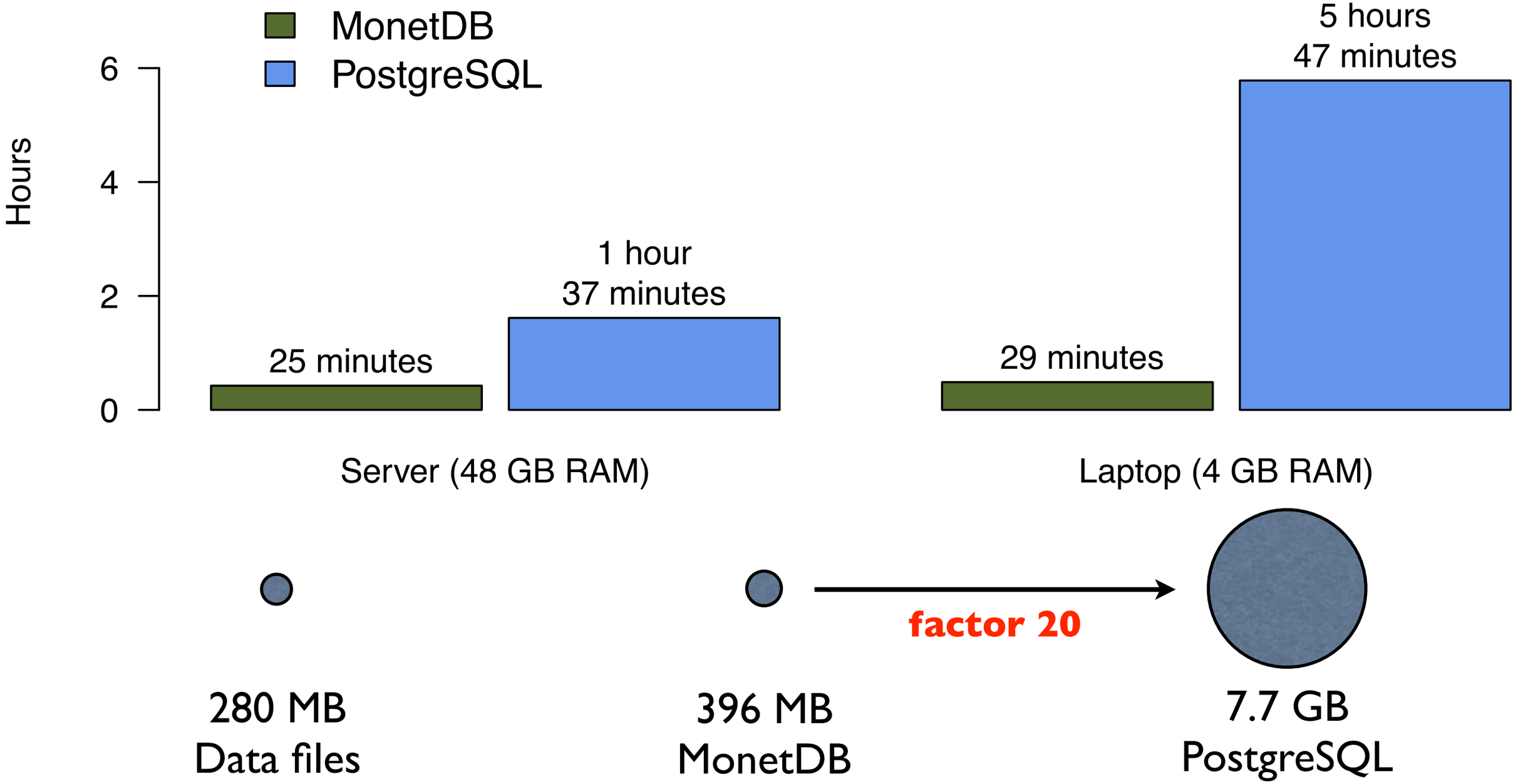
Queries:

- 3 month query log of Annis instance at the SFB 632
- 337 TIGER queries (224 unique)
- up to 4 search terms
- up to 6 binary operators

Random workload:

- 10000 queries
- original distribution
- excluded PostgreSQL timeout

Workload of 10000 queries

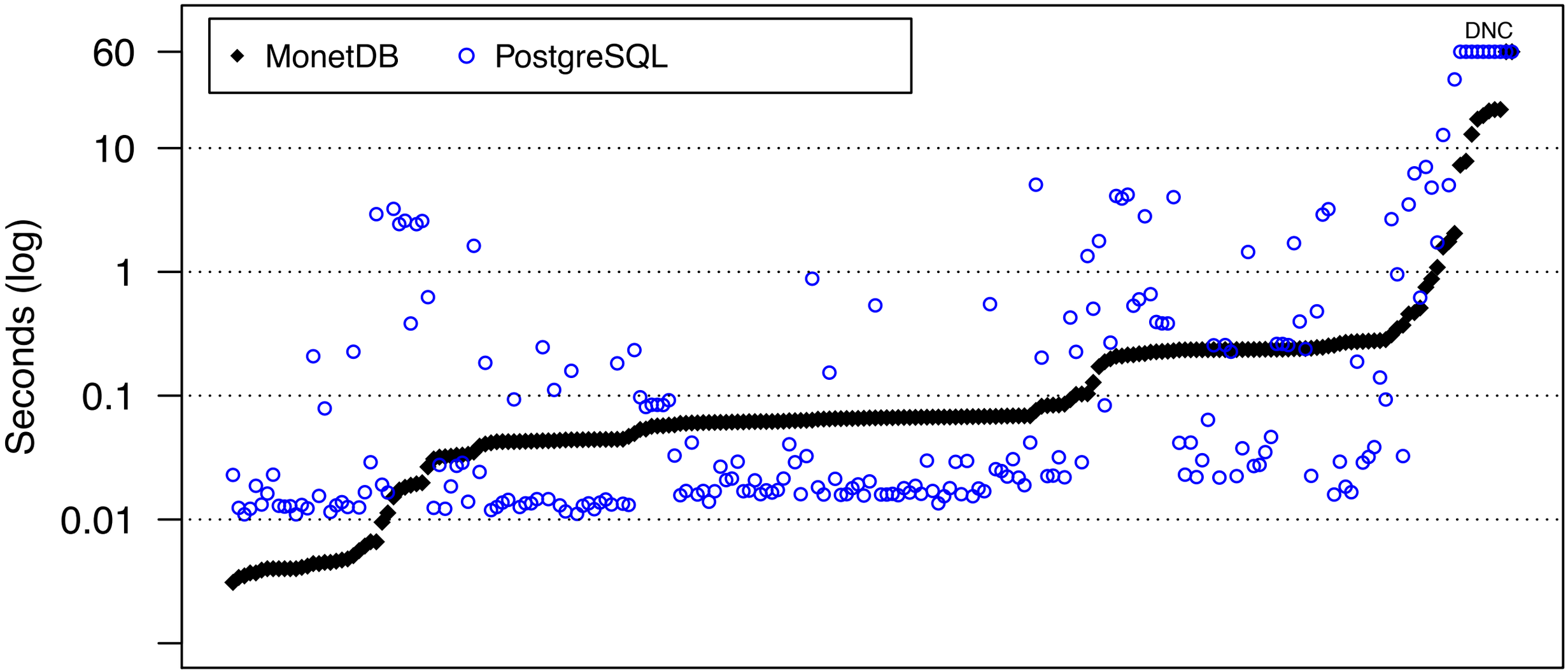


280 MB
Data files

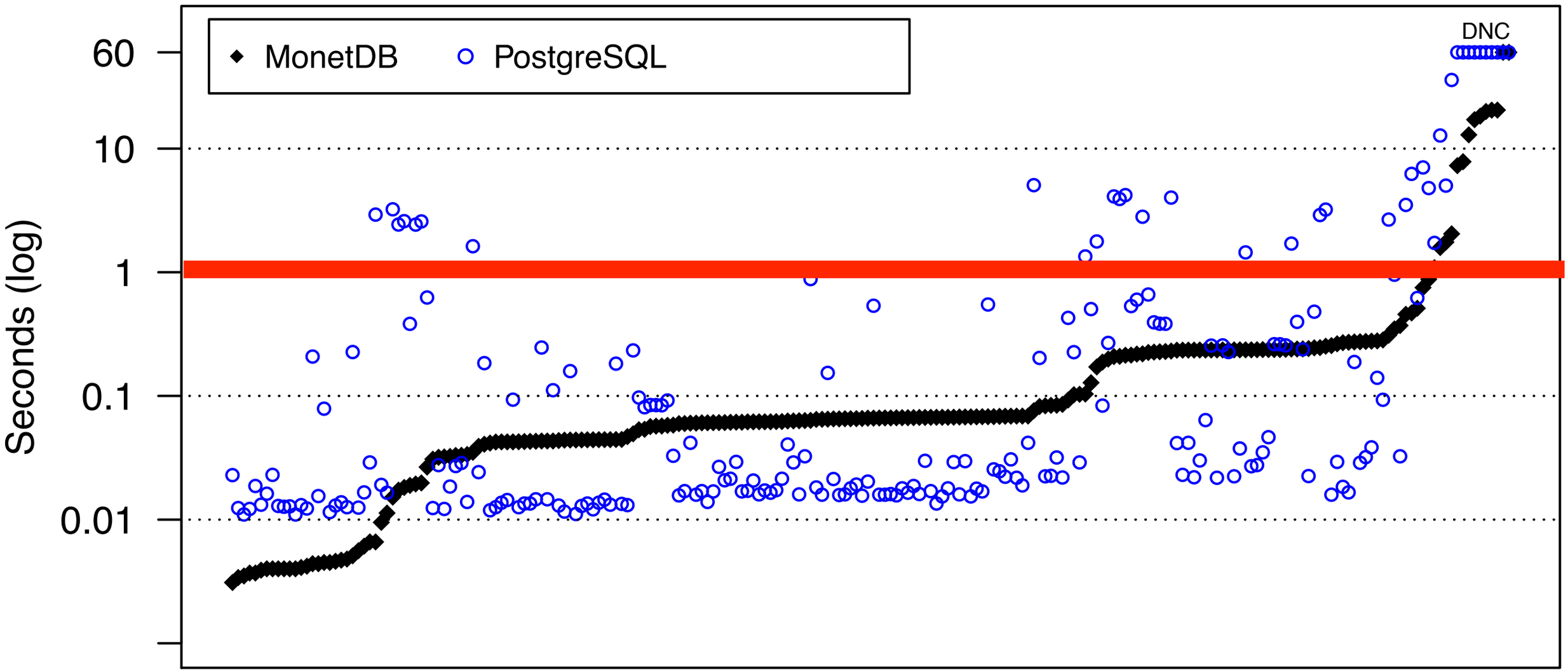
396 MB
MonetDB

7.7 GB
PostgreSQL

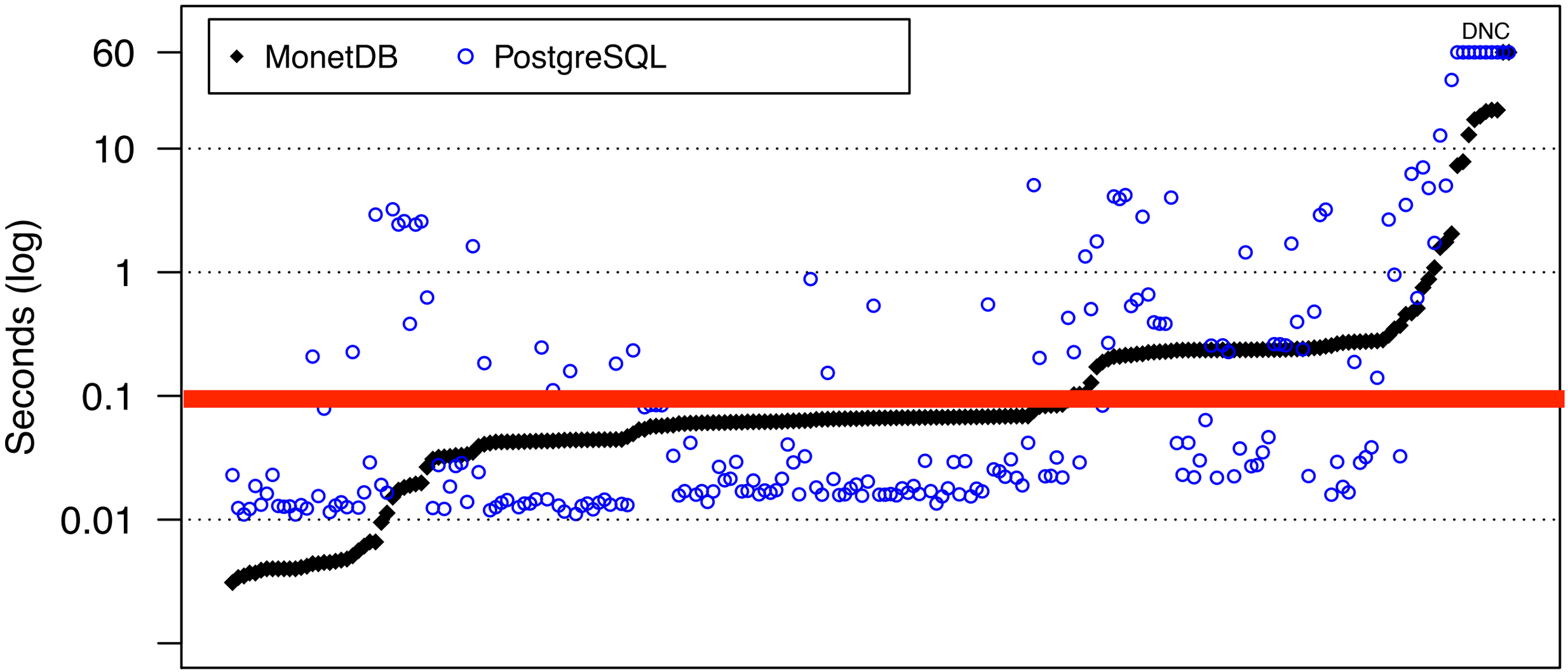
Individual query performance



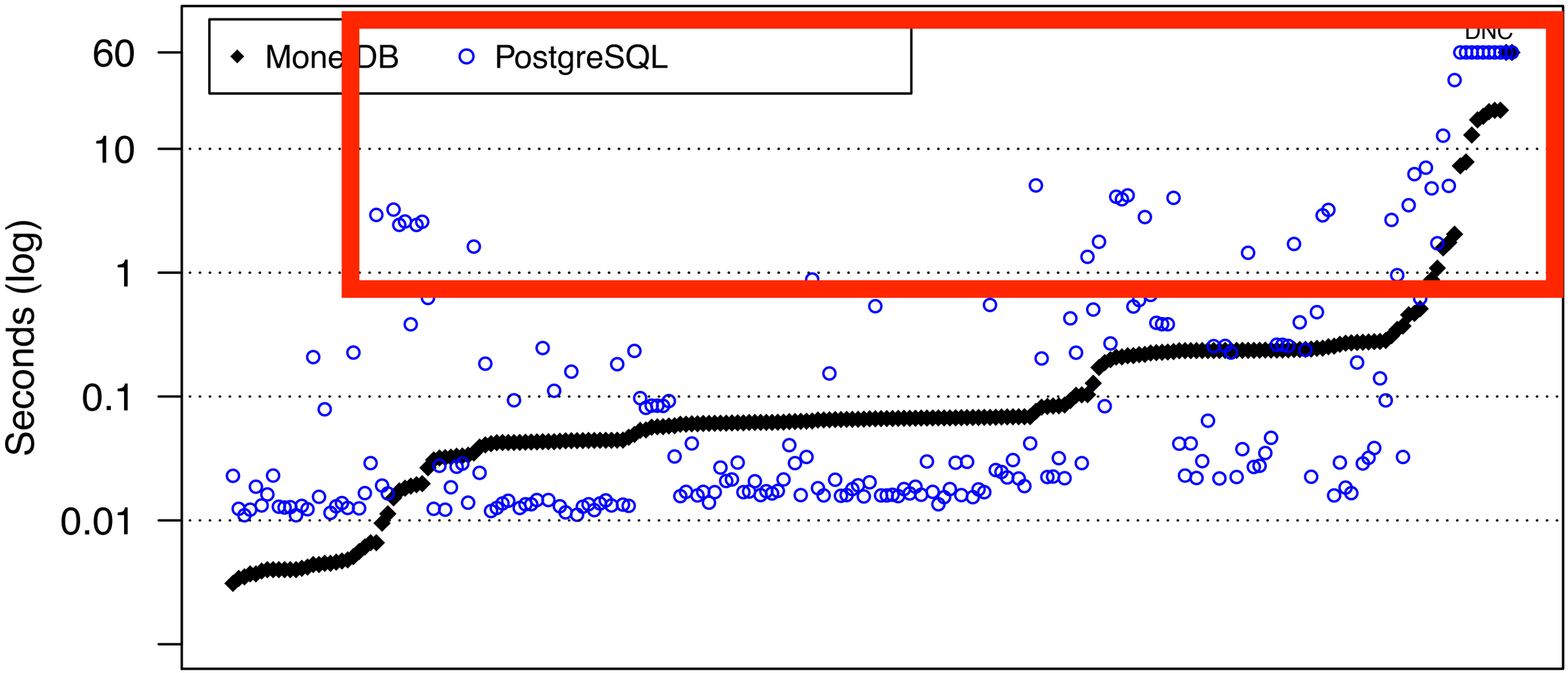
Individual query performance



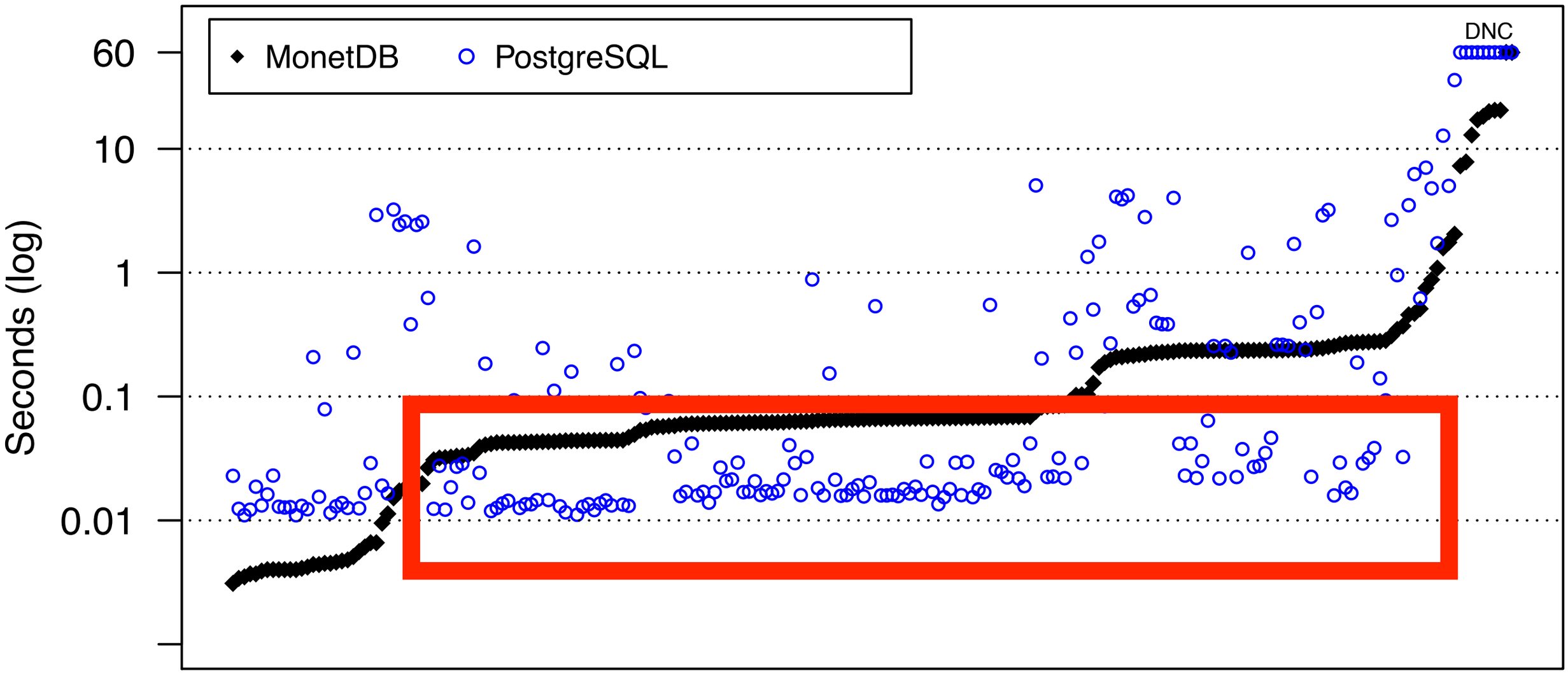
Individual query performance



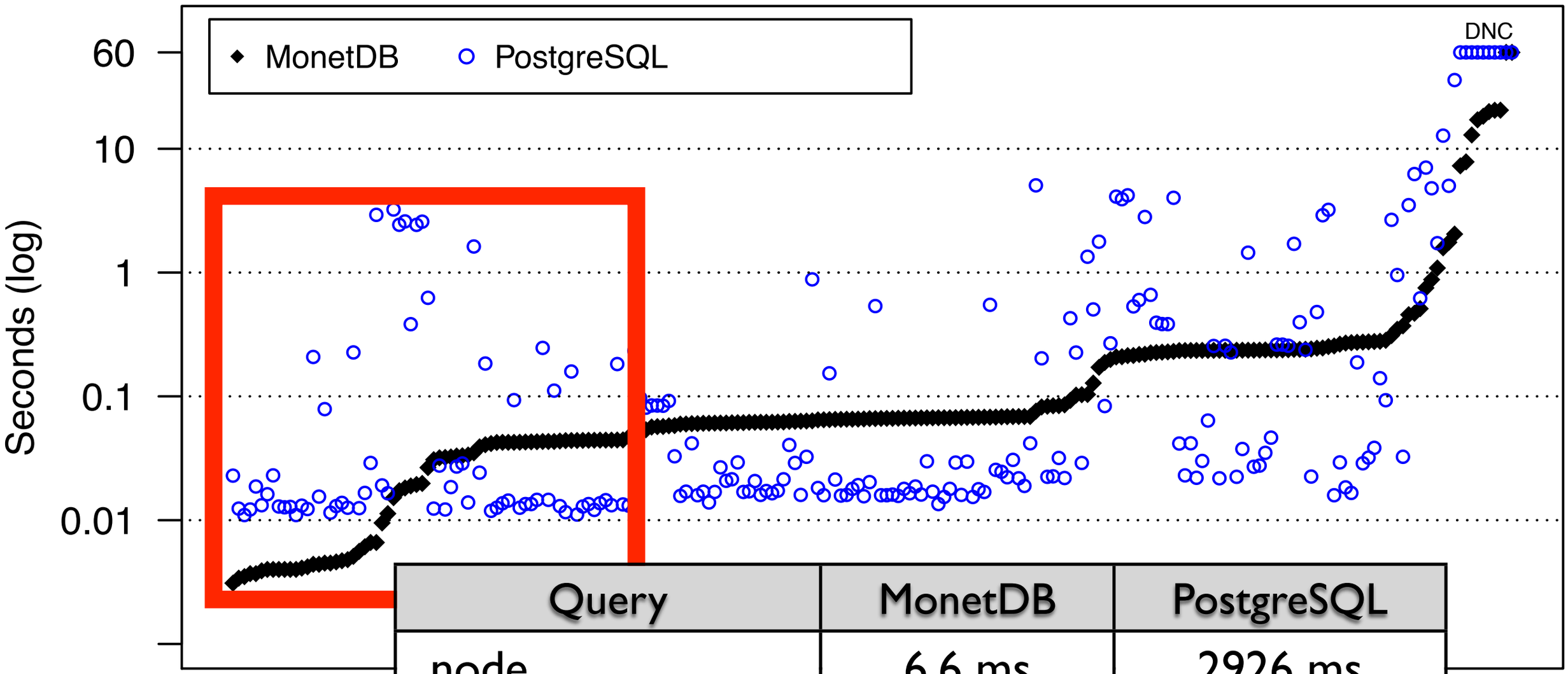
Individual query performance



Individual query performance

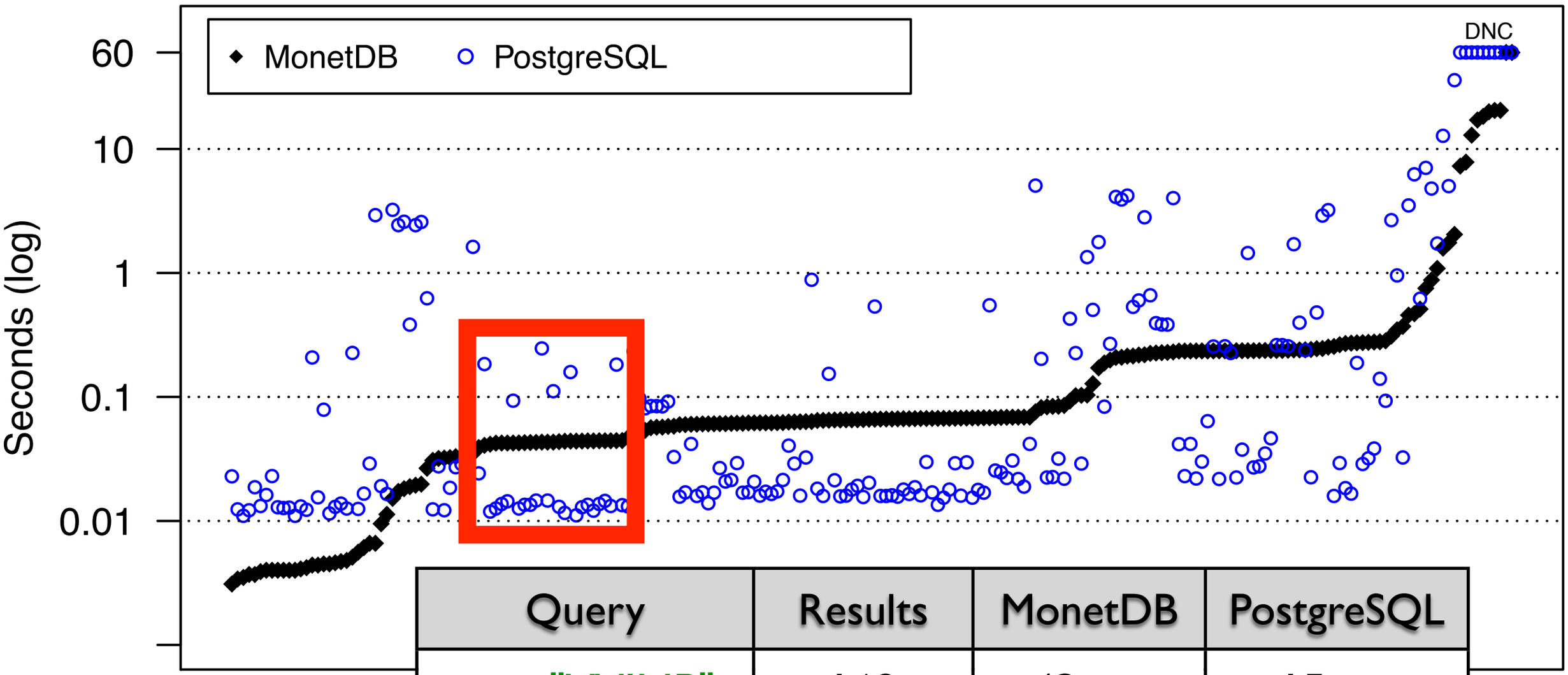


Simple queries are fast



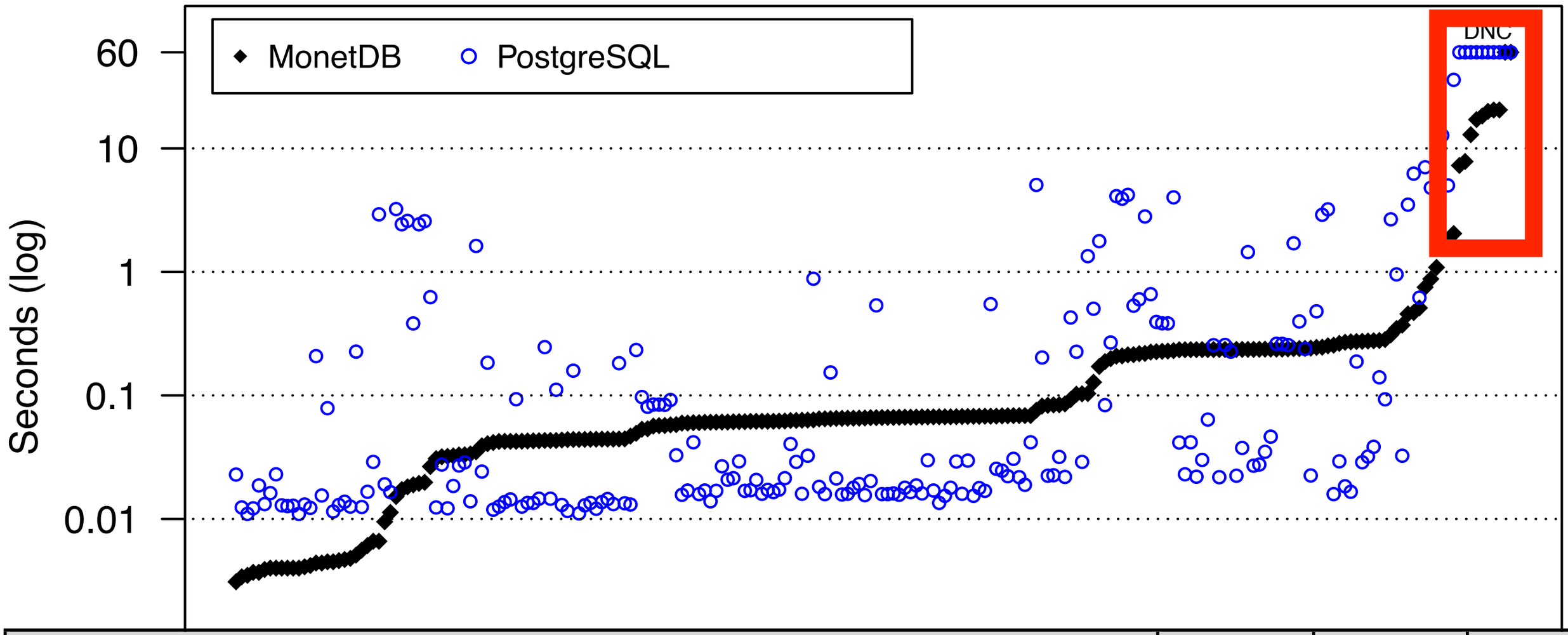
Query	MonetDB	PostgreSQL
node	6.6 ms	2926 ms
"der"	5.1	226
/der.*/	19	383
cat="S"	41	184
lemma="waschen"	43	14

Influence of result size



Query	Results	MonetDB	PostgreSQL
pos="VVIMP"	162	43 ms	15 ms
pos="VVPP"	17770	43	111
pos="VVFIN"	35628	44	182
pos="ADJA"	54534	43	246

Queries with millions of results



Query	Results	Monet	PSQL
lemma="müssen" & pos= /VV.*/ & pos="$." & #1 .* #2 & #2 .* #3	4.5 M	2 s	35 s
pos=/VM.*/ & pos= /VV.*/ & pos=/.*/ & #1 .* #2 & #2 .* #3	384 M	175 s	> 1 h

Fast regular expressions

regular expression without a fixed prefix
can't use an index, need to scan the entire column

Query	MonetDB	PostgreSQL
/.*sich.*/	213 ms	4206 ms
/[Kk]ann.*/	219	2812
pos="VVPP" & lemma=/(ge)?kommen/ & #1 _=_ #2	229	383
pos=/N.*/ & /[12][09][0-9][0-9]/ & #1 _=_ #2	246	2902
lemma=/[^äöü]+/ & /.+[äöü].+/ & pos="NN" & #1 _=_ #2 & #2 _=_ #3	469	6246

Advantages

MonetDB

- better overall performance
- stable query performance
- fast regular expressions
- normalized schema
- greatly reduced disk consumption
- better use of limited resources

PostgreSQL

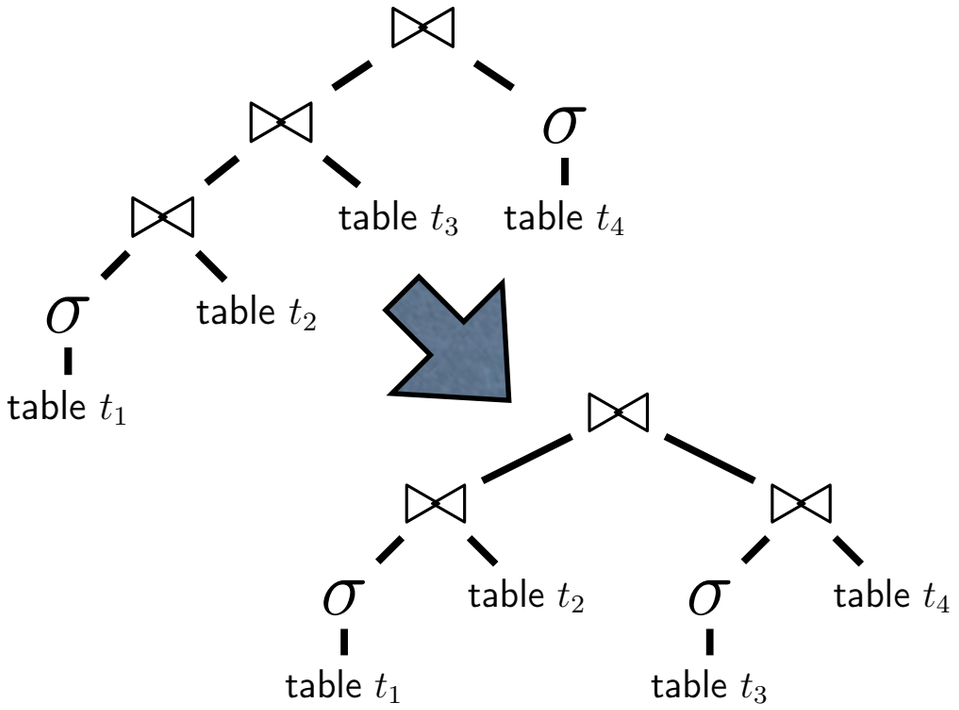
- queries with highly selective search term
- complete implementation
- bug-free SQL processing

Summary

- prototypical implementation of Annis on MonetDB
- test scenario from an Annis installation in service
- in-depth performance comparison of Annis on MonetDB and PostgreSQL

```
SELECT vielen
  FROM dank;
```

Port of Annis to MonetDB



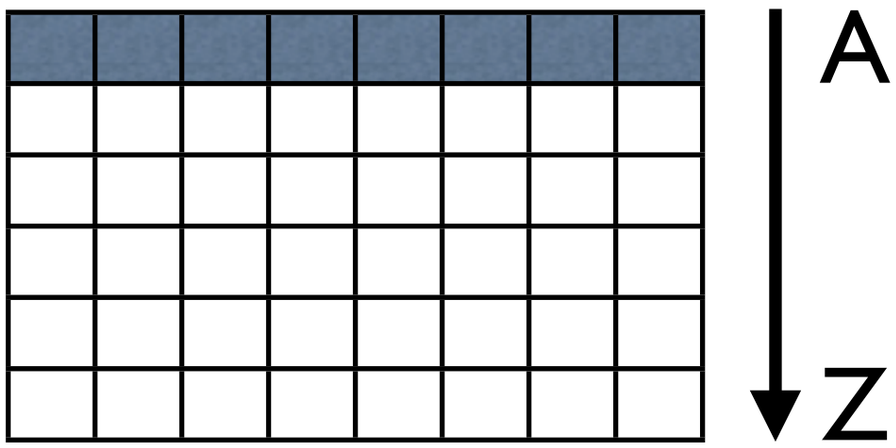
execution plan

(a|b)*

regular expressions

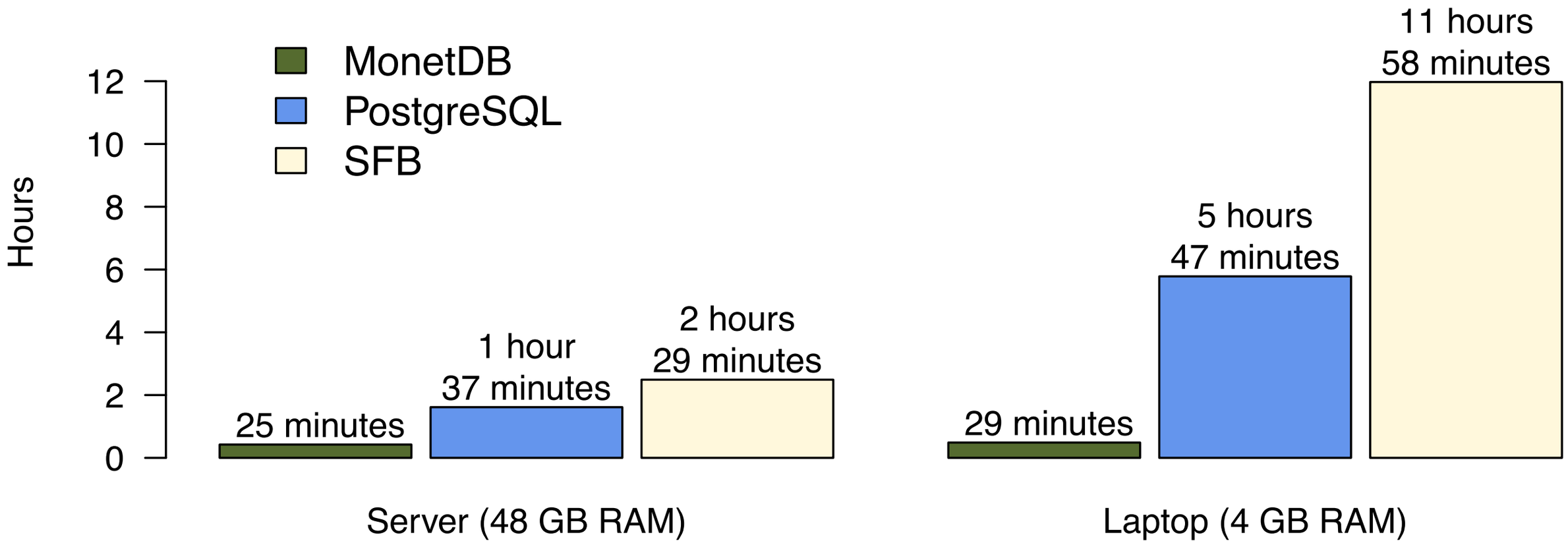
SELECT DISTINCT

SQL optimizations



sorted tables

PostgreSQL vs. MonetDB



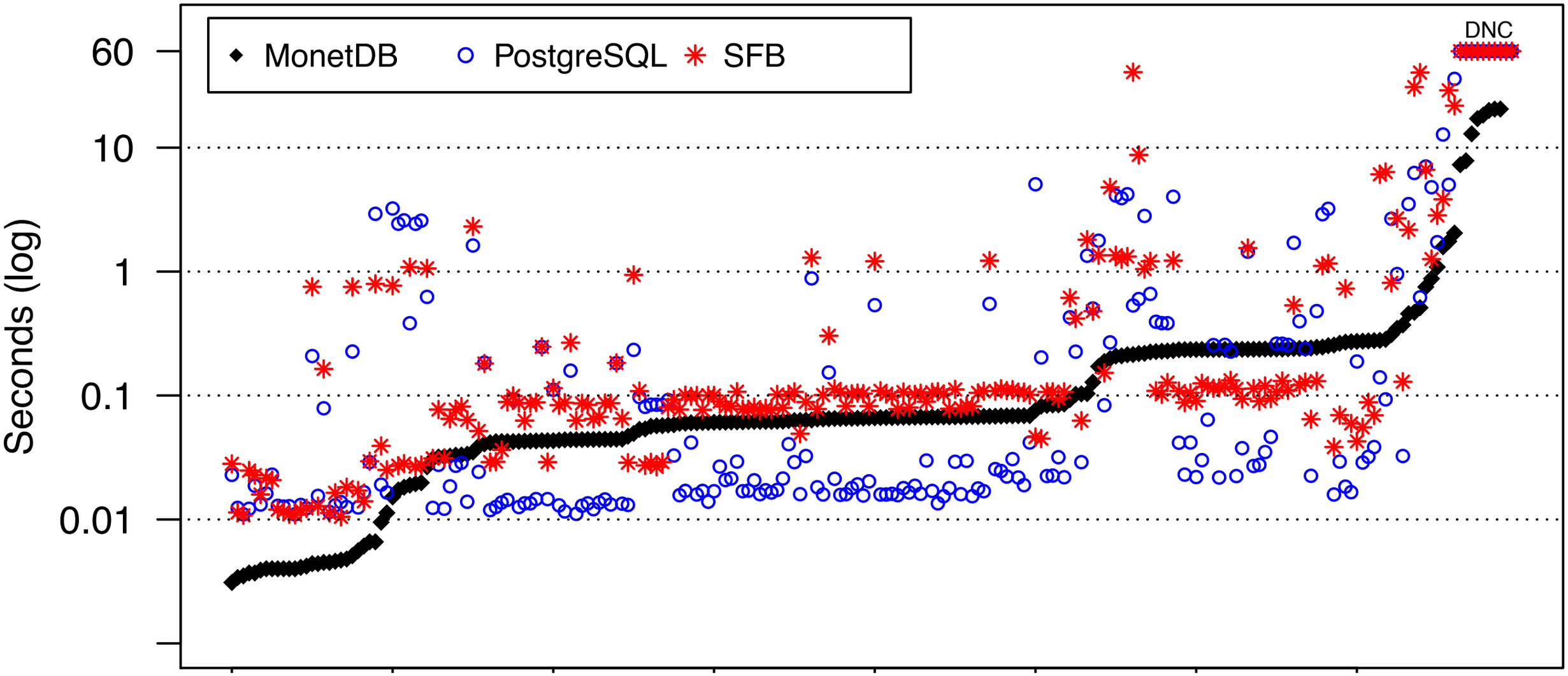
280 MB
Data files

396 MB
MonetDB

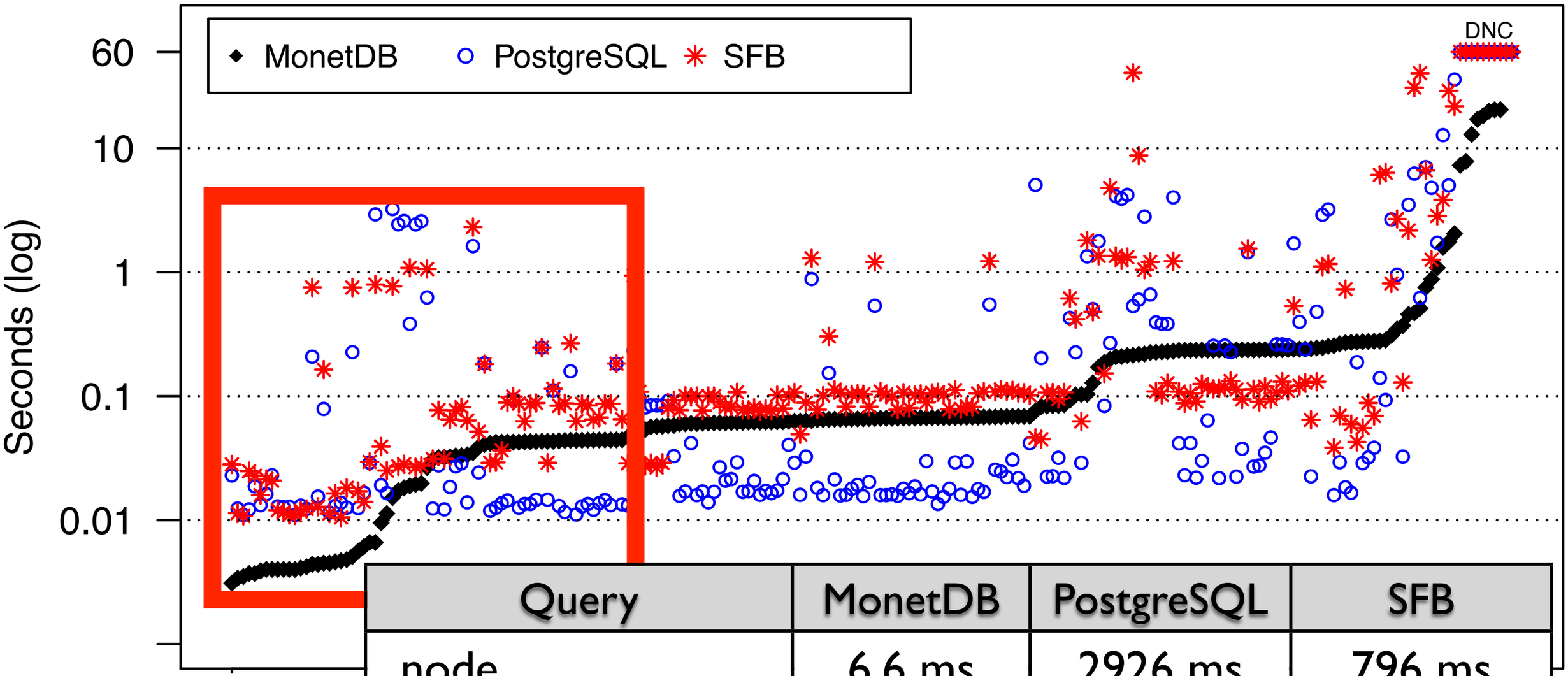
7.7 GB
Studienarbeit

7.5 GB
SFB

Individual query performance

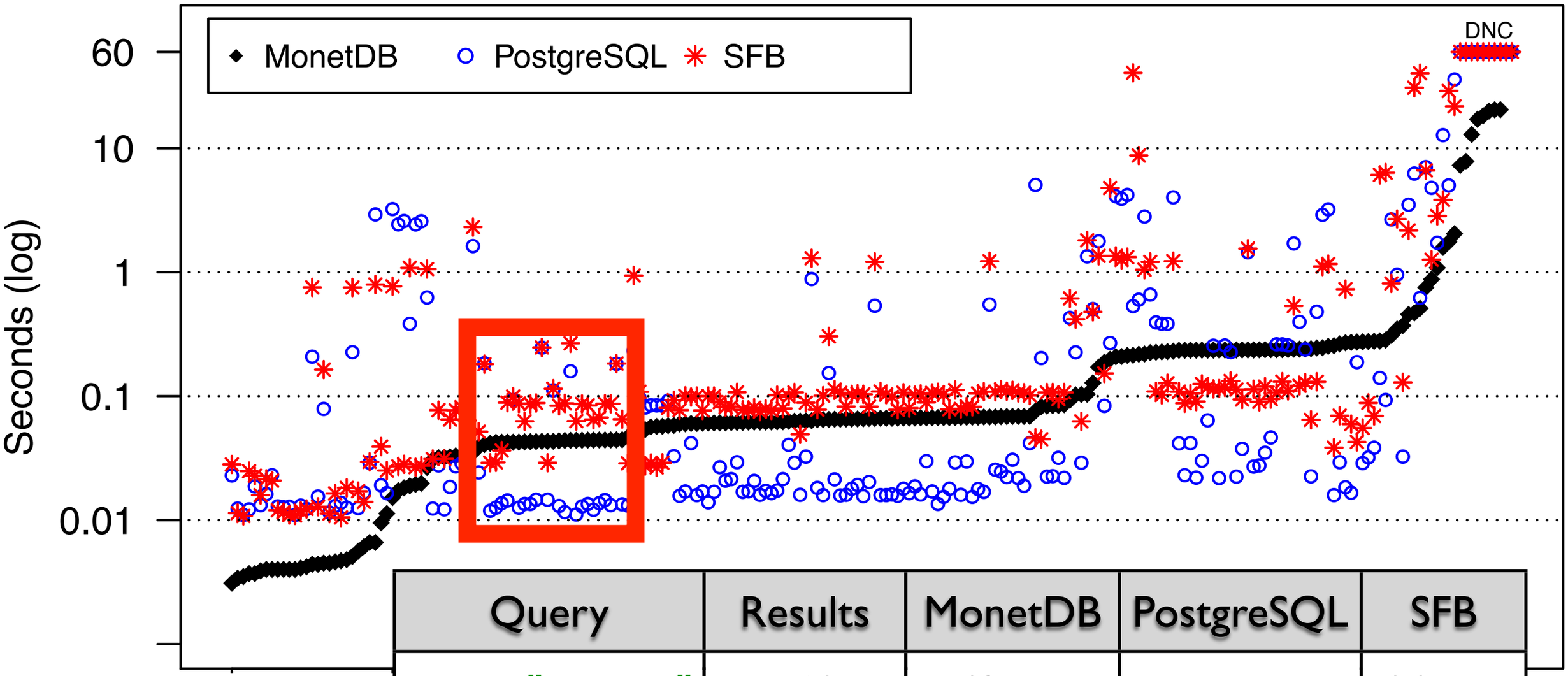


Simple queries



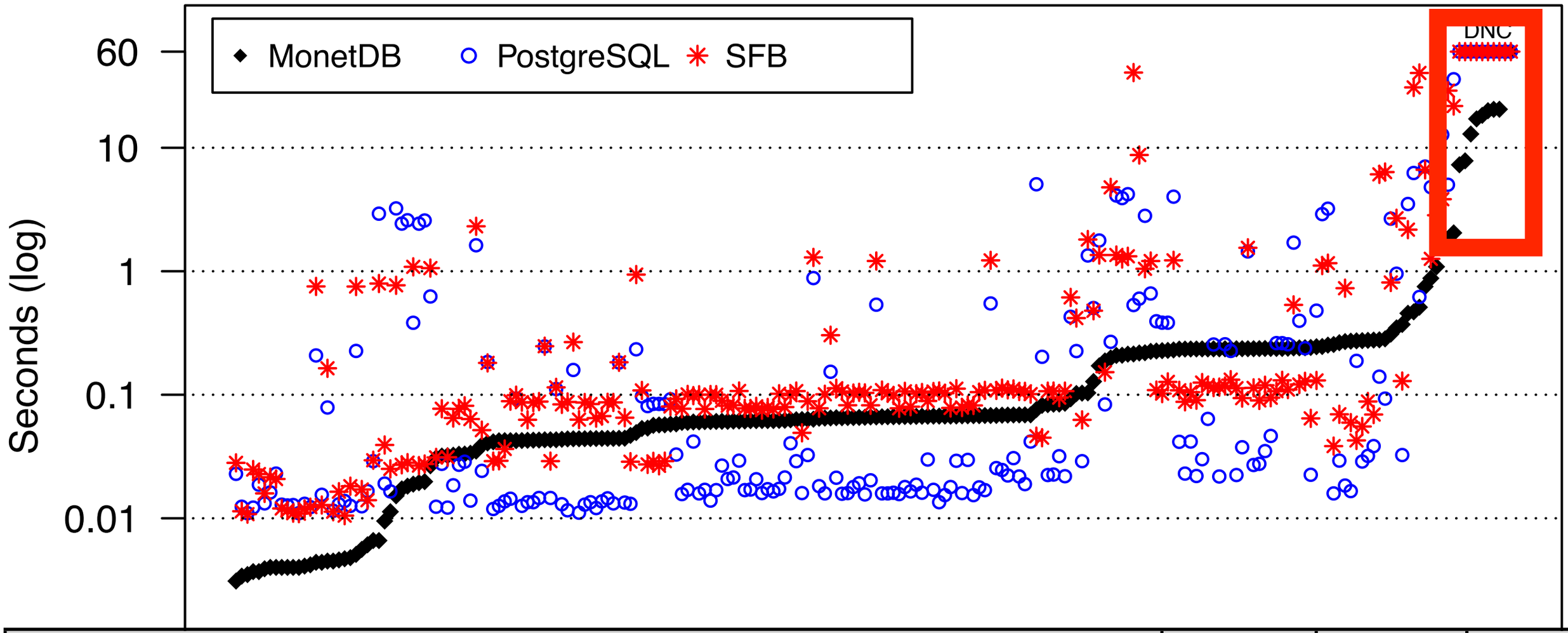
Query	MonetDB	PostgreSQL	SFB
node	6.6 ms	2926 ms	796 ms
"der"	5.1	226	753
/der.*/	19	383	1086
cat="S"	41	184	181
lemma="waschen"	43	14	63

Influence of result size



Query	Results	MonetDB	PostgreSQL	SFB
pos="VVIMP"	162	43 ms	15 ms	29 ms
pos="VVPP"	17770	43	111	115
pos="VVFIN"	35628	44	182	183
pos="ADJA"	54534	43	246	247

Queries with millions of results



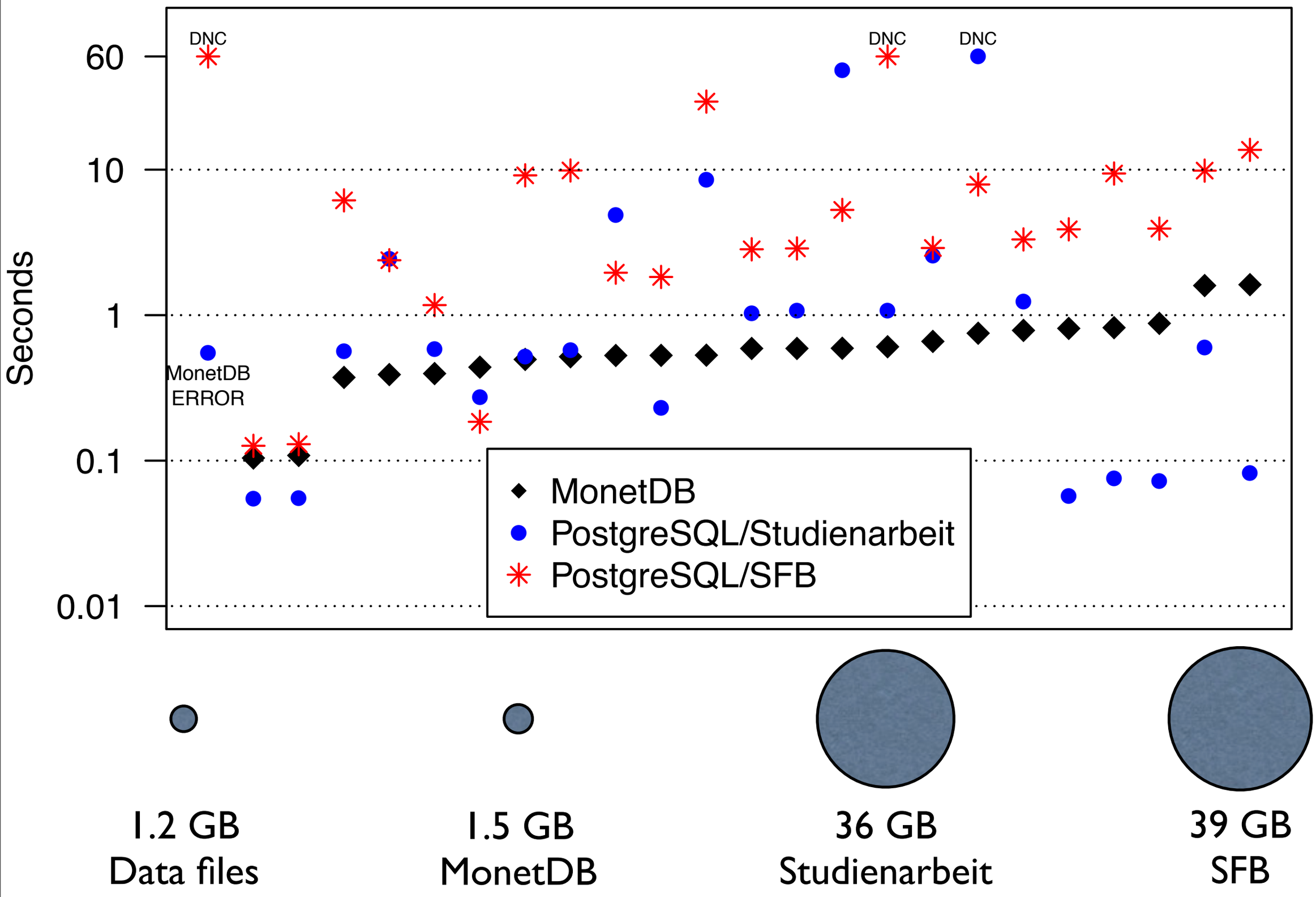
Query	Monet	PSQL	SFB
lemma="müssen" & pos= /VV.*/ & pos="$." & #1 .* #2 & #2 .* #3	2 s	35 s	22 s
pos=/VM.*/ & pos= /VV.*/ & pos=/.*/ & #1 .* #2 & #2 .* #3	175 s	> 1 h	36 m

Regular expressions

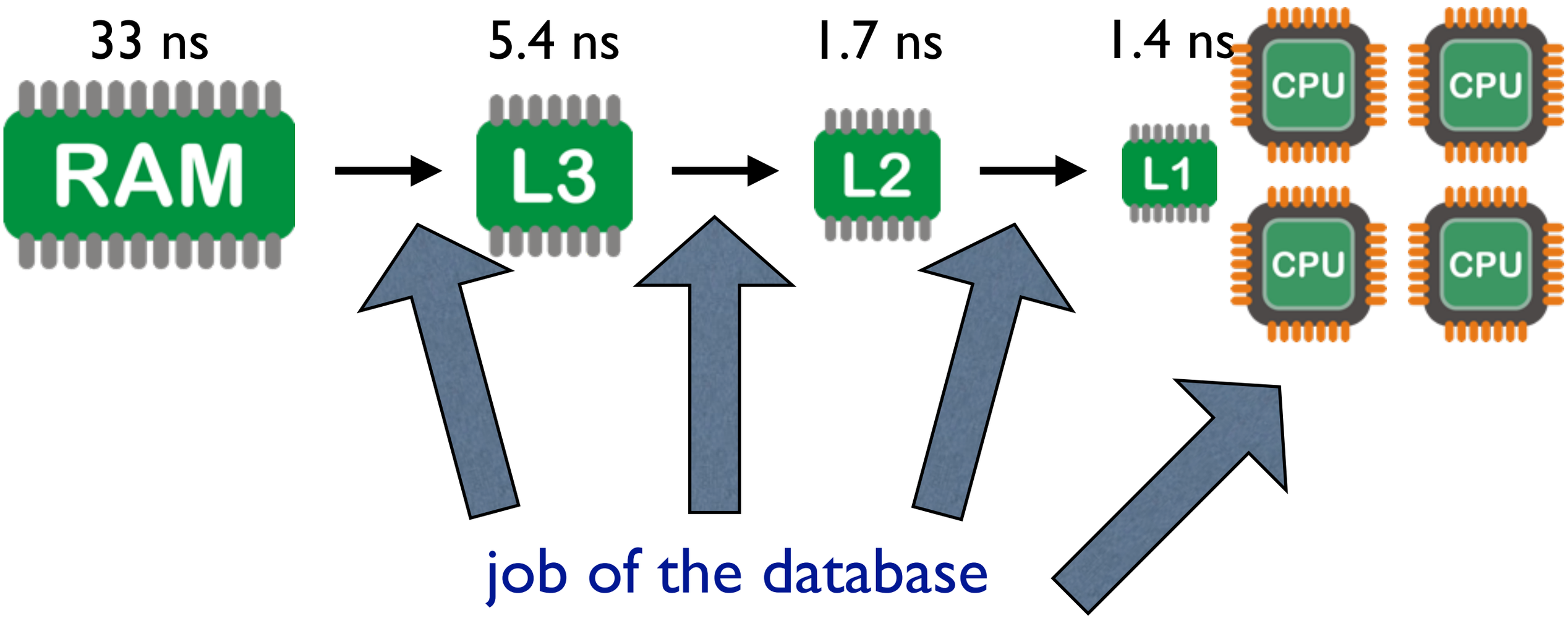
regular expression without a fixed prefix
can't use an index, need to scan the entire column

Query	MonetDB	PSQL	SFB
/.*sich.*/	213 ms	4206 ms	1322 ms
/[Kk]ann.*/	219	2812	1047
pos="VVPP" & lemma=/(ge)?kommen/ & #1 _=_ #2	229	383	127
pos=/N.*/ & /[12][09][0-9][0-9]/ & #1 _=_ #2	246	2902	1111
lemma=/[^äöü]+/ & /.+[äöü].+/ & pos="NN" & #1 _=_ #2 & #2 _=_ #3	469	6246	30803

Example queries on TüBa-D/Z



Multiple CPUs



job of the database
on a modern system
(among others)

Algorithms

Row-Store

1	123	pos	VVINF
2	123	lemma	essen
3	456	pos	NN

- function calls in a loop
  ```
  getNextTuple
  getAttribute('name')
  compareAttribute('lemma')
  ```
- pipeline flush
- instruction cache miss

50% of time spent in resource stalls

Column-Store

node_ref	name	value
123	pos	VVINF
123	lemma	essen
456	pos	NN

- array lookup in a loop
  ```
  column[index] == 'lemma'
  ```
- loop unrolling
- no jumps
- instruction locality

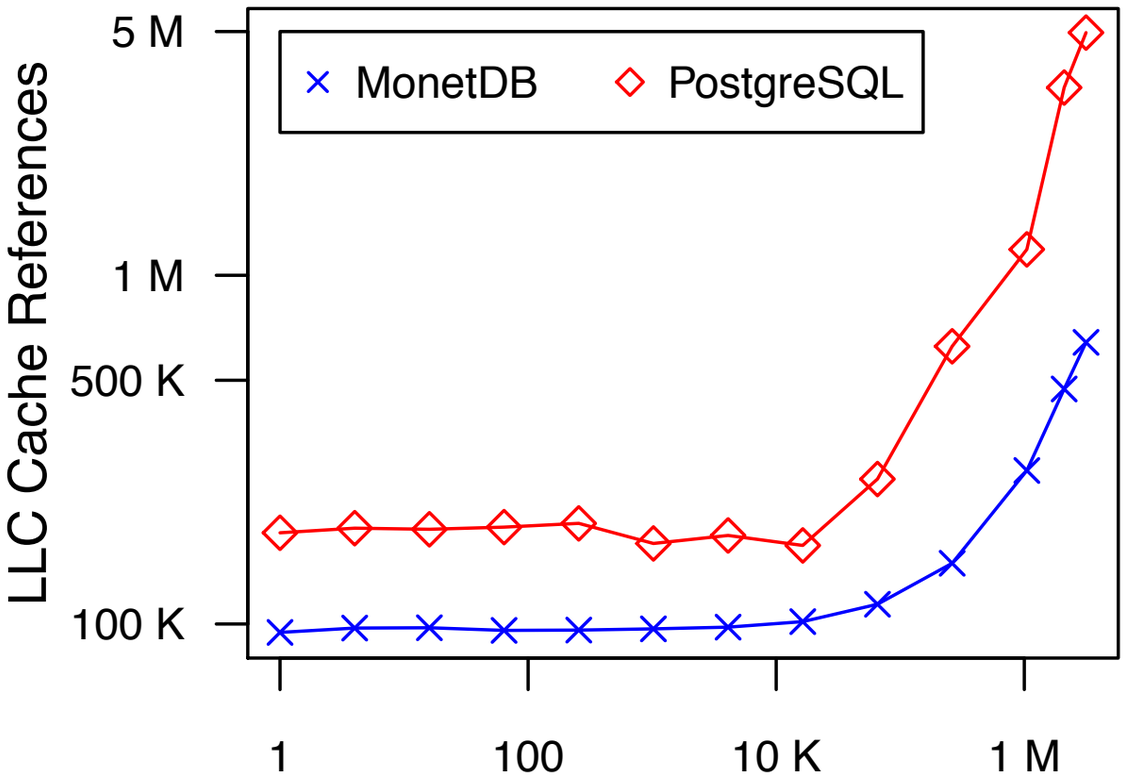
History of Column-Stores

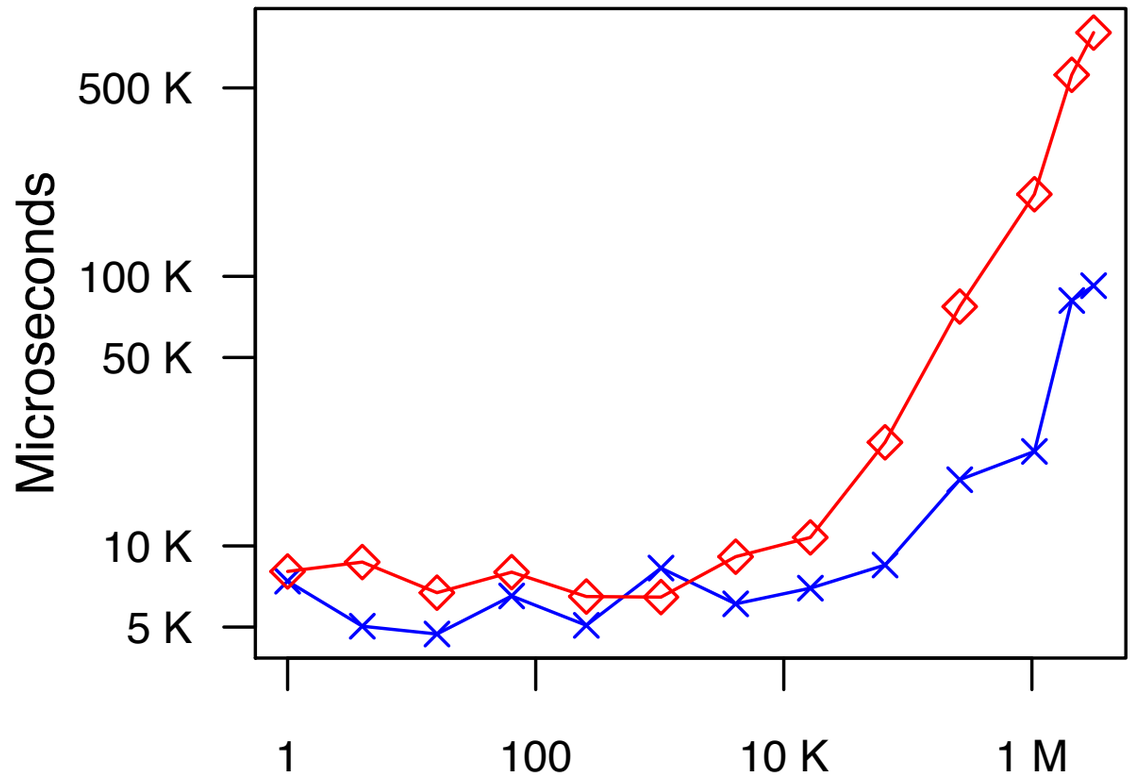
- IBM, 1973: *System R*
- Copeland, 1985: *A Decomposed Storage Model*
- CWI, 1994: *Monet. An Impressionist Sketch of an Advanced Database system*
- 2000s: Column-Stores become mainstream
  - C-Store, Vertica
  - MonetDB/X100, Vectorwise
  - SAP HANA

MonetDB

- pioneered research into Column-Stores
- active research platform
  - adaptive indexing, *database cracking*
  - partial vectorized bulk-processing, *X100*
  - opportunistic materialization, *recycler*
  - array primitives, *SciQL*
- well-received
  - VLDB 10 Years Best Paper Award, 2009: *Database Architecture Optimized For The New Bottleneck: Memory Access*
  - SIGMOD Jim Gray Doctoral Dissertation Award, 2011: *Database Cracking: Towards Auto-tuning Database Kernels*
  - VLDB Challenges & Vision Best Paper Award, 2011: *The Researcher's Guide to the Data Deluge: Querying a Scientific Database in Just a Few Seconds*

Cache usage vs. query speed

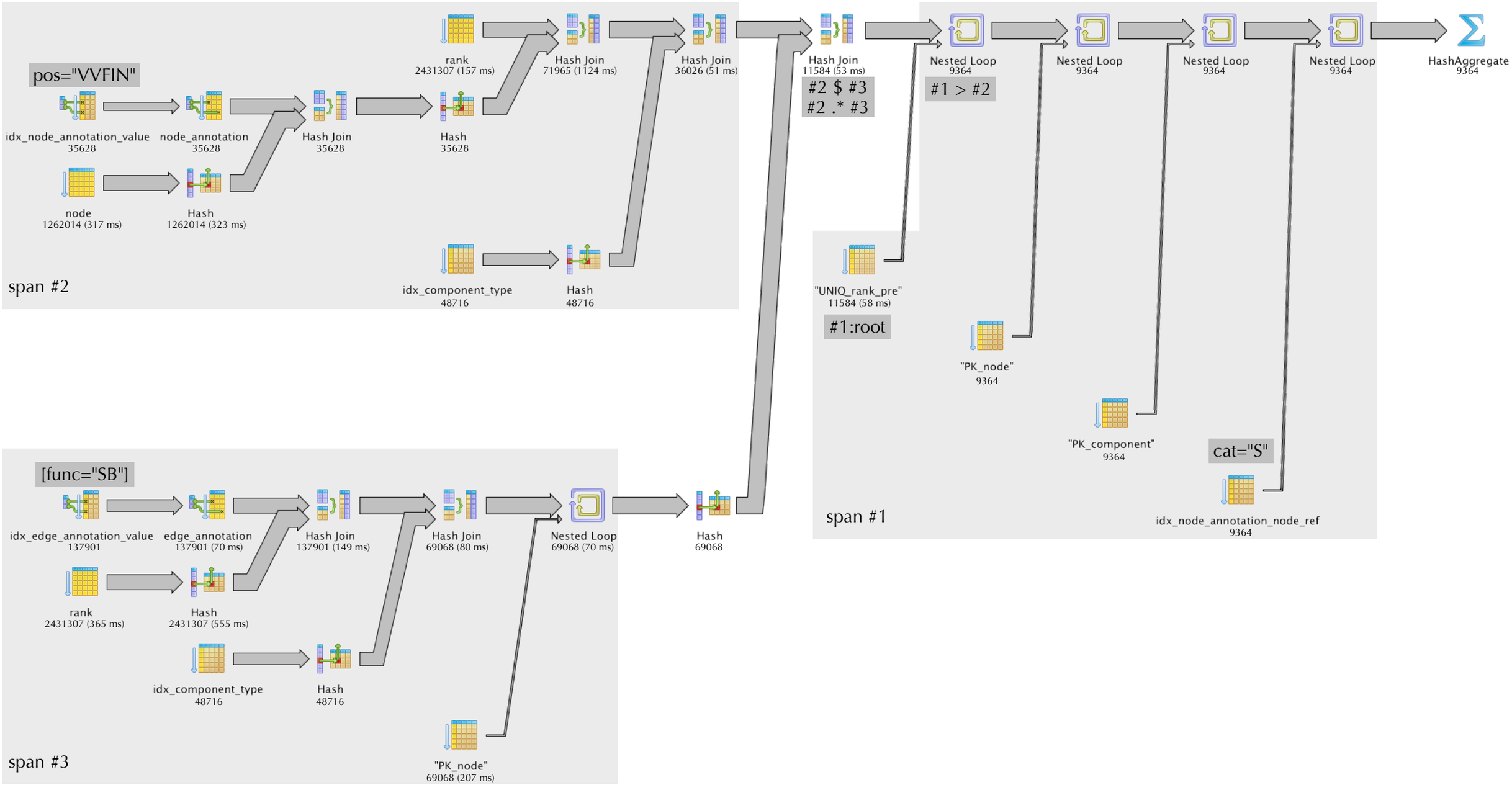




Query: annotation name = 'lemma'

- last level cache references
- indirect measure of processed data
- everything that is processed must pass once through the cache

Normalized schema on PostgreSQL



Materialized schema on PostgreSQL