2nd order operators

A tale of second order

Problem

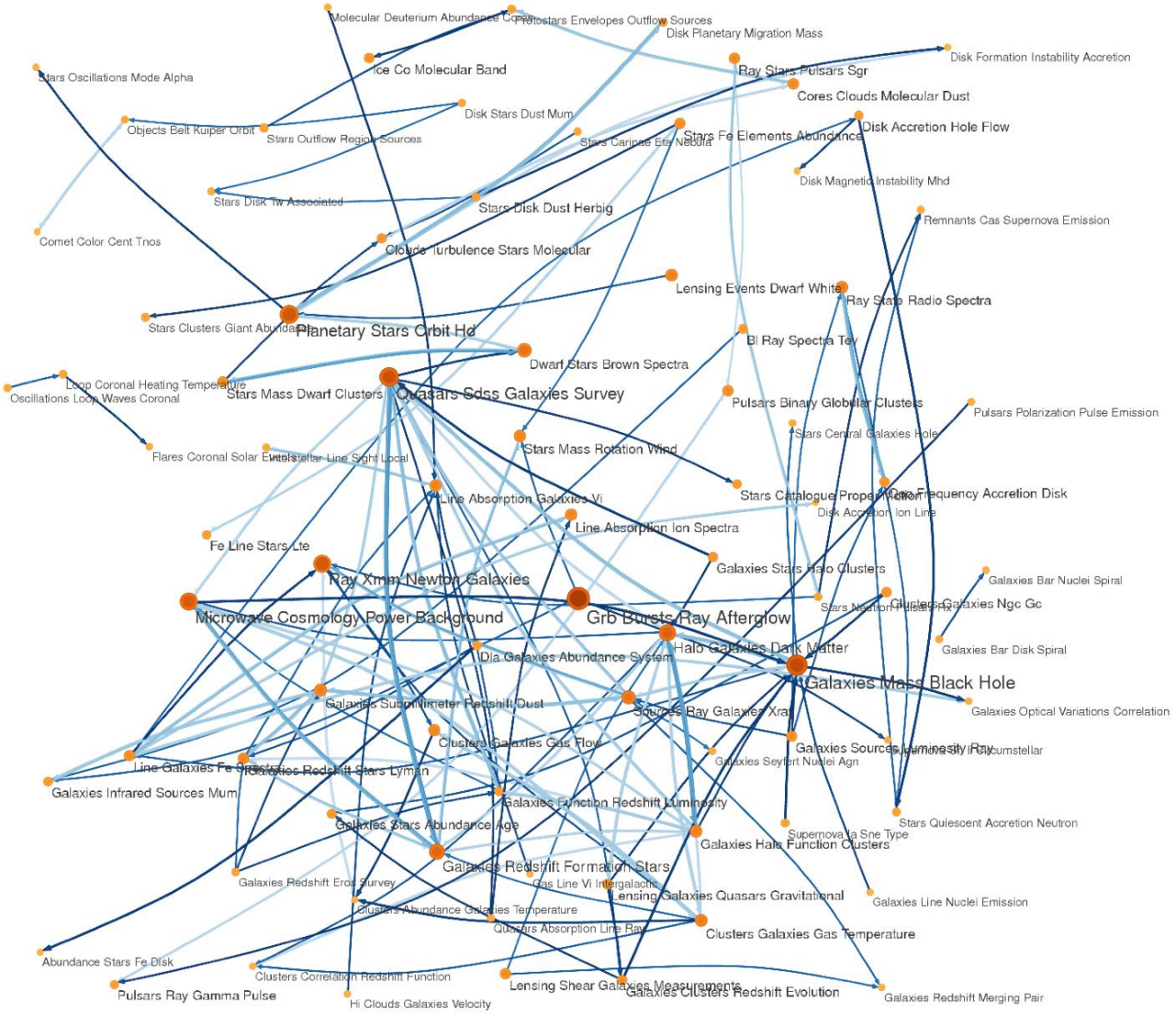
Search for **related "things"**

What other people cite
read
watch
listen to
buy...

...but using **lucene**

Relations

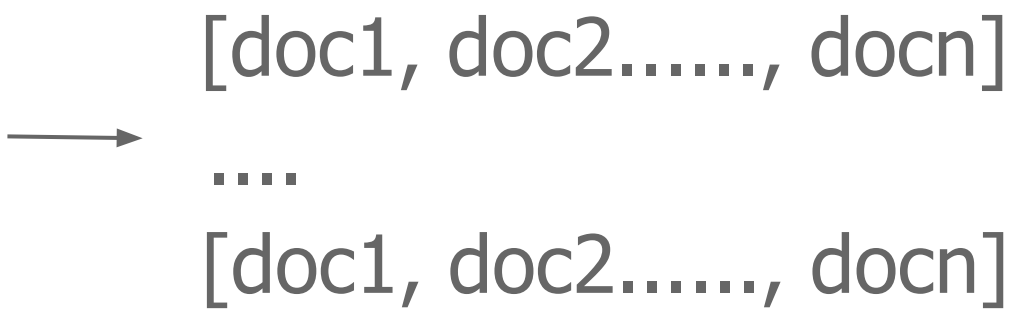
exploring astronomy literature



But inverted index is "flat"

Good for free text search

Not so good for relational data



Real-life examples

http://adslabs.org/adsabs

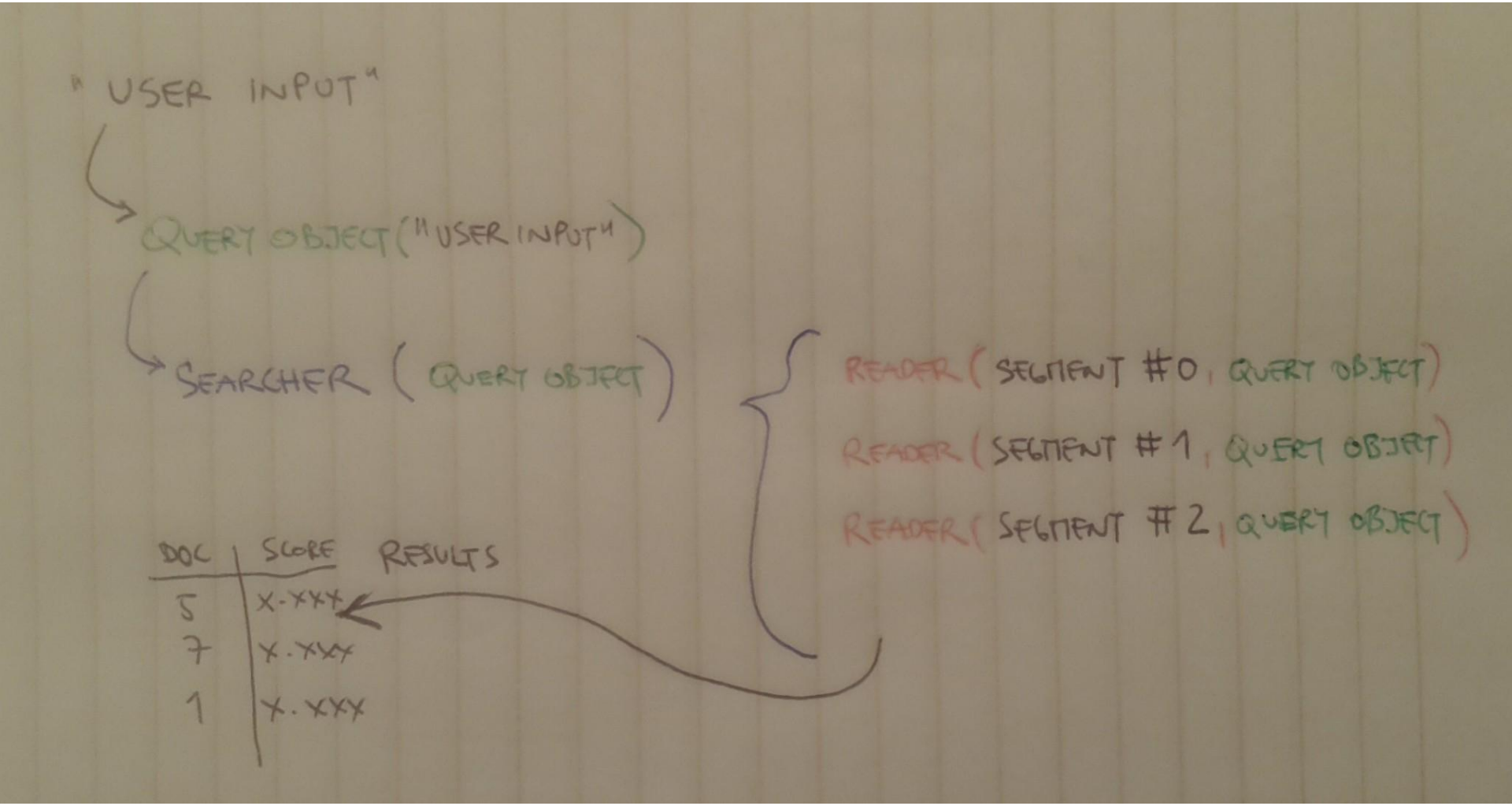
bibcode:2000A&AS..143...41K
citations(bibcode:2000A&AS..143...41K)
citations(author:accomazzi) and year:2012
citations(author:accomazzi and year:2007)
references(title:"recommender system")

warning: pre-release, may crash anytime

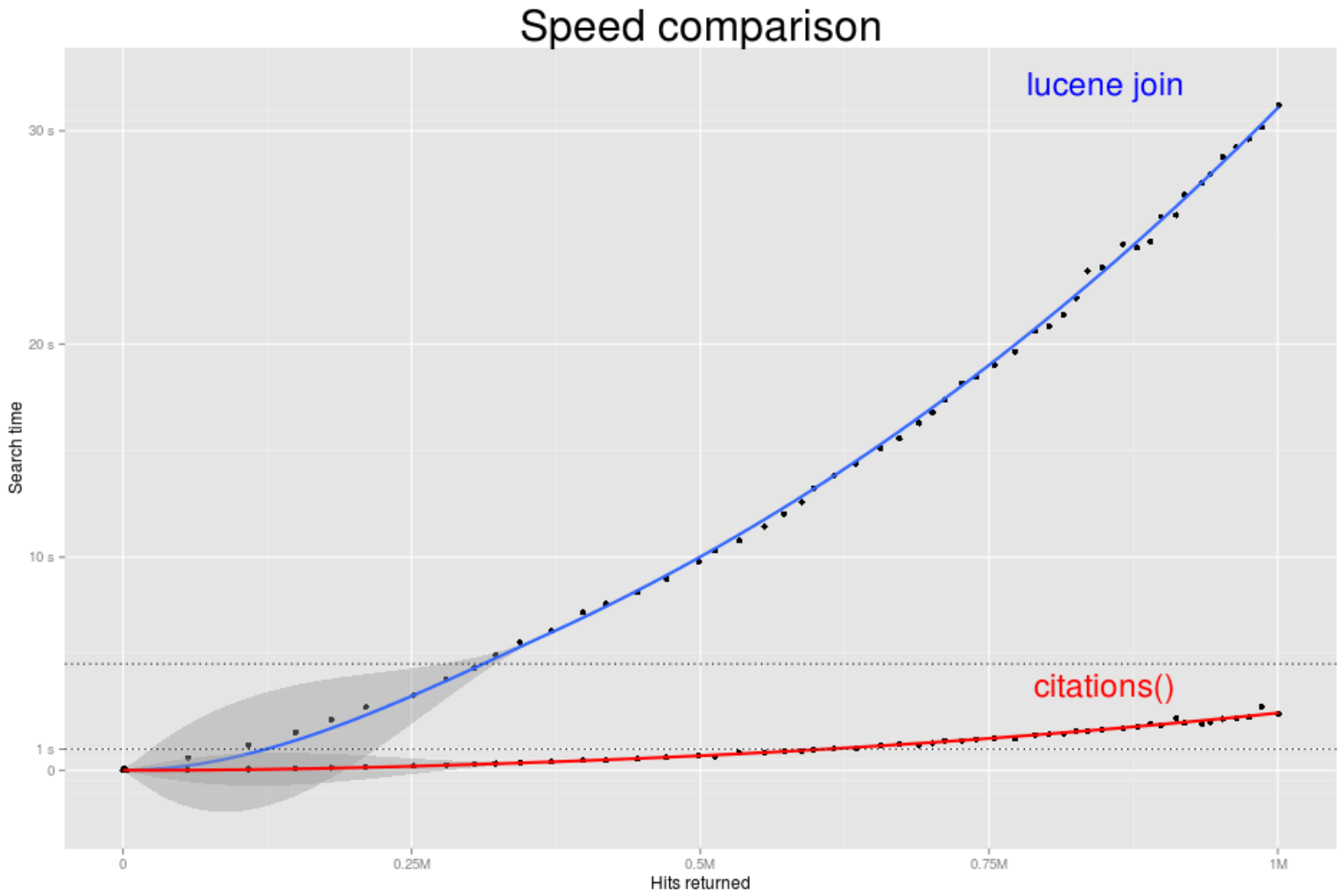
Relations


```
- <doc>
  - <str name="title">
      Controlled transcription of a human α-interferon gene introduced into mouse L cells
    </str>
  - <str name="pub_raw">
      Nature, Volume 297, Issue 5862, pp. 128-132 (1982).
    </str>
  - <arr name="reference">
      <str>1981PNAS...78.5435L</str>
      <str>1972PNAS...69.1408A</str>
      <str>1981Sci...212.1159L</str>
      <str>1979PNAS...76.3683W</str>
      <str>1980Natur.284..316N</str>
      <str>1979Natur.281...40M</str>
      <str>1980PNAS...77..740B</str>
      <str>1981Natur.290...20G</str>
      <str>1981Natur.291..629K</str>
      <str>1981PNAS...78.3123O</str>
      <str>1980Natur.287..408A</str>
      <str>1981PNAS...78.2848S</str>
      <str>1981PNAS...78.1411D</str>
      <str>1981PNAS...78.7038C</str>
      <str>1981PNAS...78.2038H</str>
      <str>1980Natur.287..401N</str>
      <str>1980Sci...209.1343S</str>
    </arr>
    <str name="bibcode">1982Natur.297..128M</str>
  </doc>
```

General flow of search



Comparison w lucene join



2nd order operator query

```
new SecondOrderQuery(
    innerQuery,
    new SecondOrderCollectorCitedBy(...),
    );
```

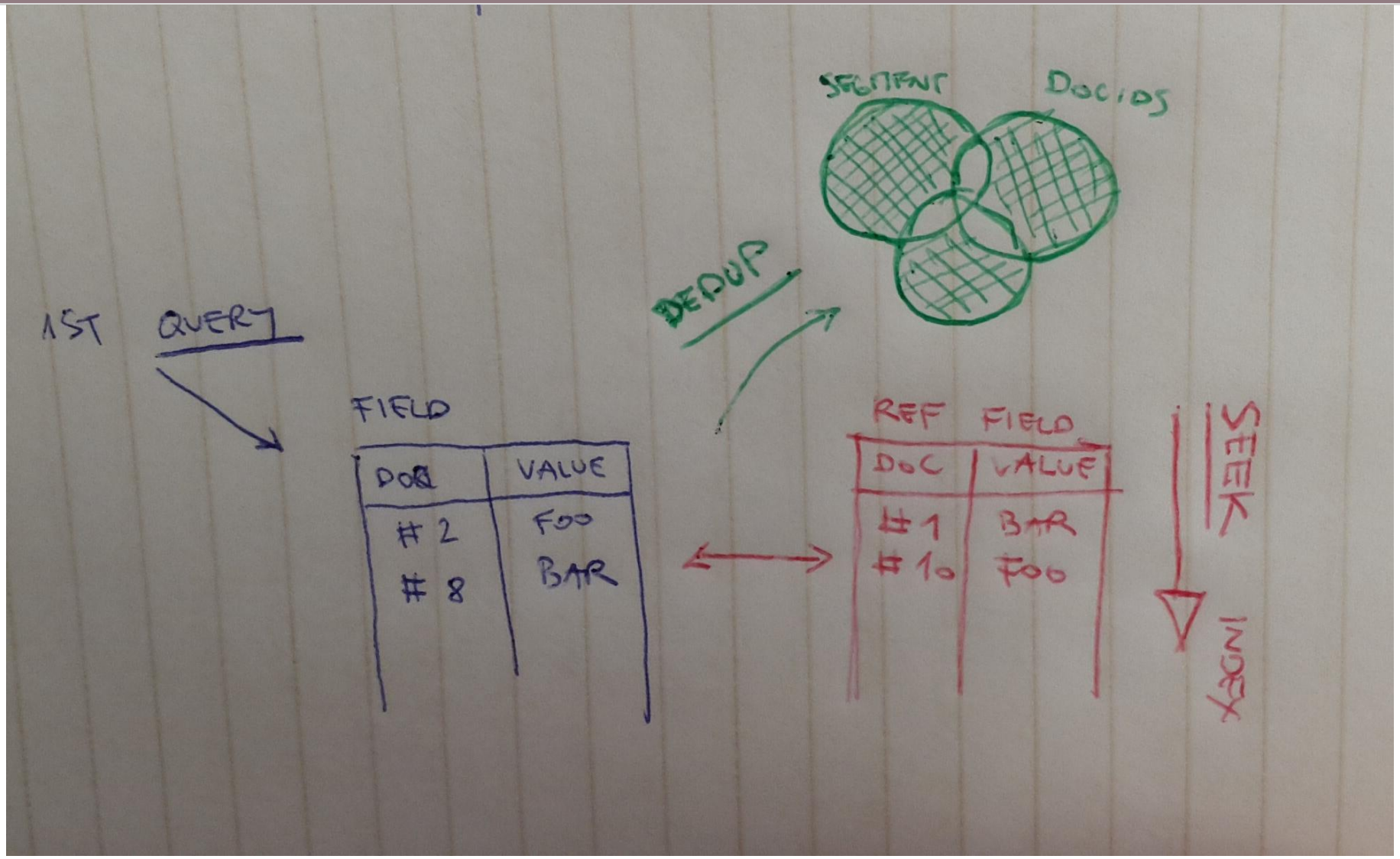
General flow

2ndOrderQuery initializes cache and collector

collector **must know the relation**

```
@Override
public void collect(int doc) throws IOException {
    if (invertedIndex[doc+docBase] == null) return;
    float s = scorer.score();
    float freq = (float) invertedIndex[doc+docBase].length;
    for (int v: invertedIndex[doc+docBase]) {
        hits.add(new CollectorDoc(v, s, -1, freq));
    }
}
```

General flow



Cache

Lucene already comes with a cache

- un-inverted index

Array of values ordered by lucene docids

```
[abc,        #0
 def,        #1
 ghi,        #2
 ....
 xxx]        #maxDoc
```

2nd order query

Operator is a **mapping function**

$f:docid_X \rightarrow docid_Y$

User can implement any relation

E, #0	[G, X],	E:(G, X)	G:(E,...)
A, #1 +	[],	→	A:()	→ x:(A,...)
H, #2	[E,],	H:(E,)	E:(H,....)
G #3	[B, H]	G:(B,H)	H:(G,...)

Conclusions

Not tested with distributed search

Probably good only for smaller collections

- our index = 9.5M docs, 160GB
- cache implementation matters

But flexible and relatively fast/easy to set up

1. write a collector (which initializes cache)
2. create 2ndOrderQuery object

Thanks!

řchýla@cfa.harvard.edu
jlůkeŕ@cfa.harvard.edu

Links

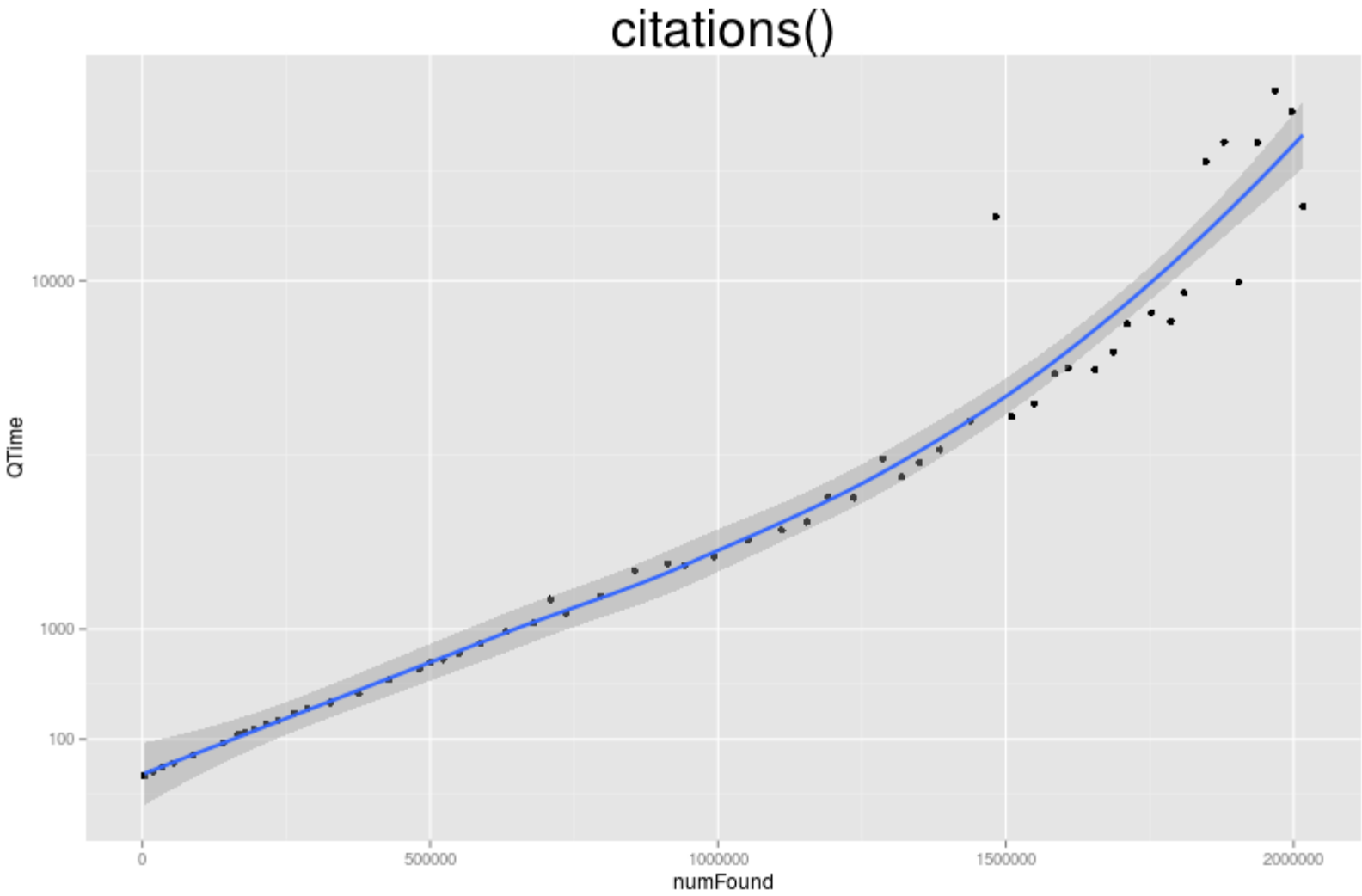
Code released under Apache license:

https://github.com/romanchyla/montysolr/tree/master/contrib/invenio/src/java/org/apache/lucene/search

Code tree changes often, if the link doesn't work, search the project for:

SecondOrderQuery.java

If there is time left



Available Solutions

- Index based
  - Document denormalization
  - Block joins
  - Grouping

INDEX
<segment0: 0,1,....600>
<segment1: 601,602.....>
.....
<segmentn:....maxDoc>

- Query based
  - Subsequent searches
  - Lucene join
  - 2nd order operator

Joins explained