In-Memory Text Search Engines

Peter Sanders,
Frederik Transier

Universität Karlsruhe (TH)
Forschungsuniversität • gegründet 1825

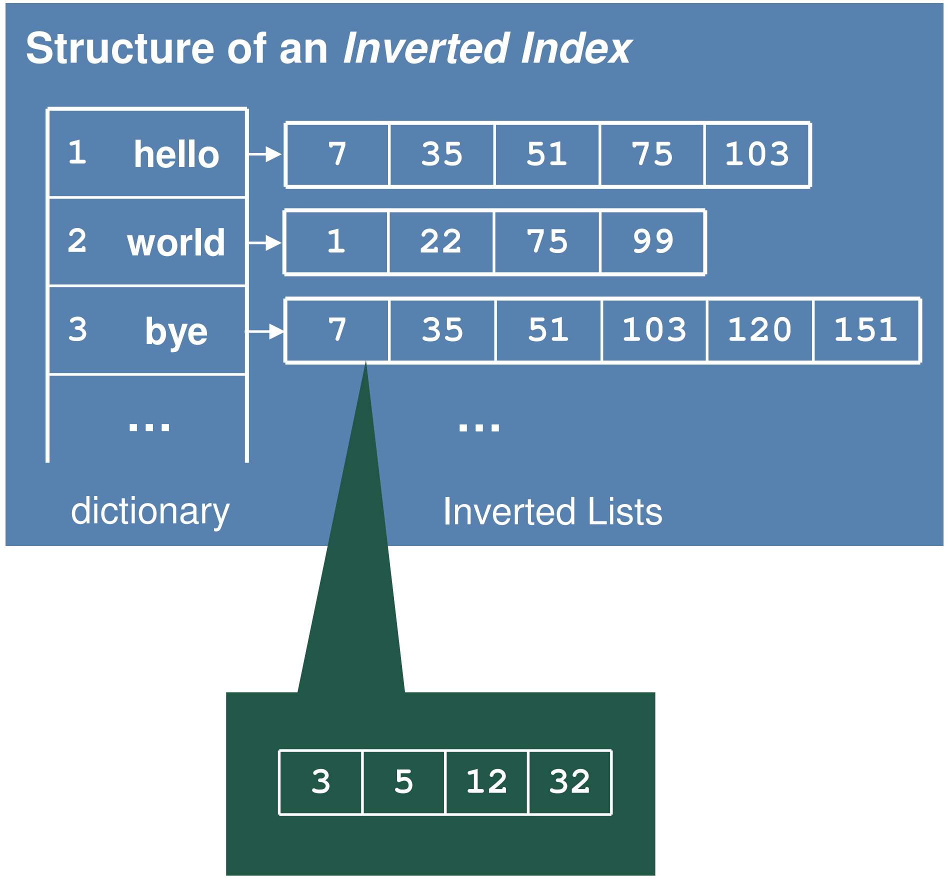
SAP®

Demands and Goals of Text Search Engines

- Locating words in a huge number of documents.
- Important Operations: AND queries, phrase queries, document reporting.
- Using main-memory for maximum query performance.

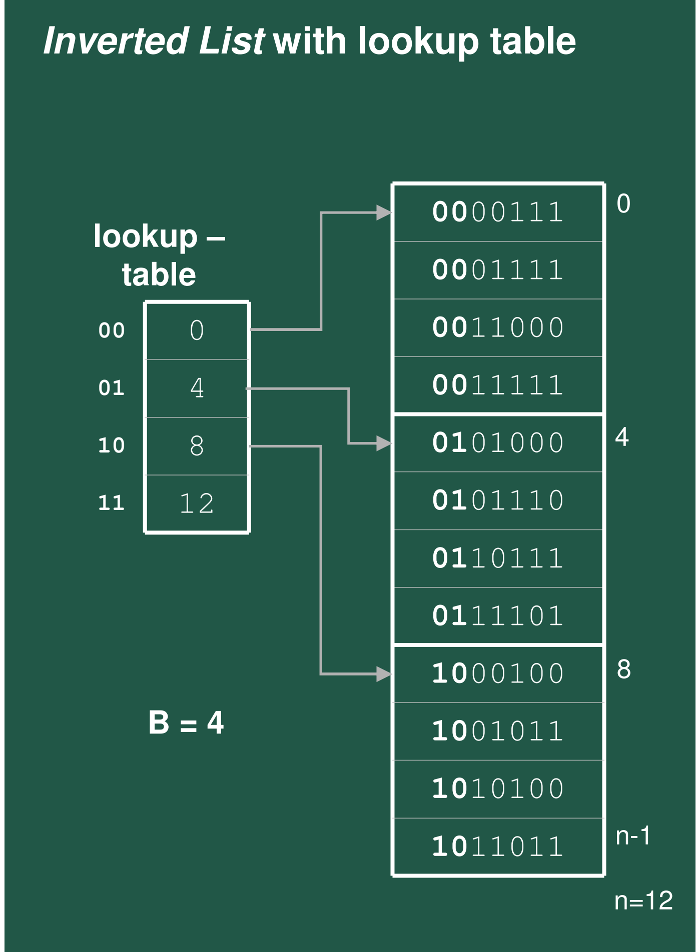
Inverted Index

- Unique IDs for:
  - documents
  - terms
- A document becomes a list of term IDs.
- Inverted Index: For each term we store a list of document IDs.
- Positional Inverted Index: Positions list for each term / document pair.



Intersection Algorithms: Randomized Inverted Indices

- **Randomized Inverted Index: document IDs are assigned randomly. (e.g. using pseudo-random permutation)**
- **Two-level data structure:**
  - **Split the range of document IDs into buckets based on their most significant bits.**
  - ***Lookup-table*: direct access to the first value of a bucket**



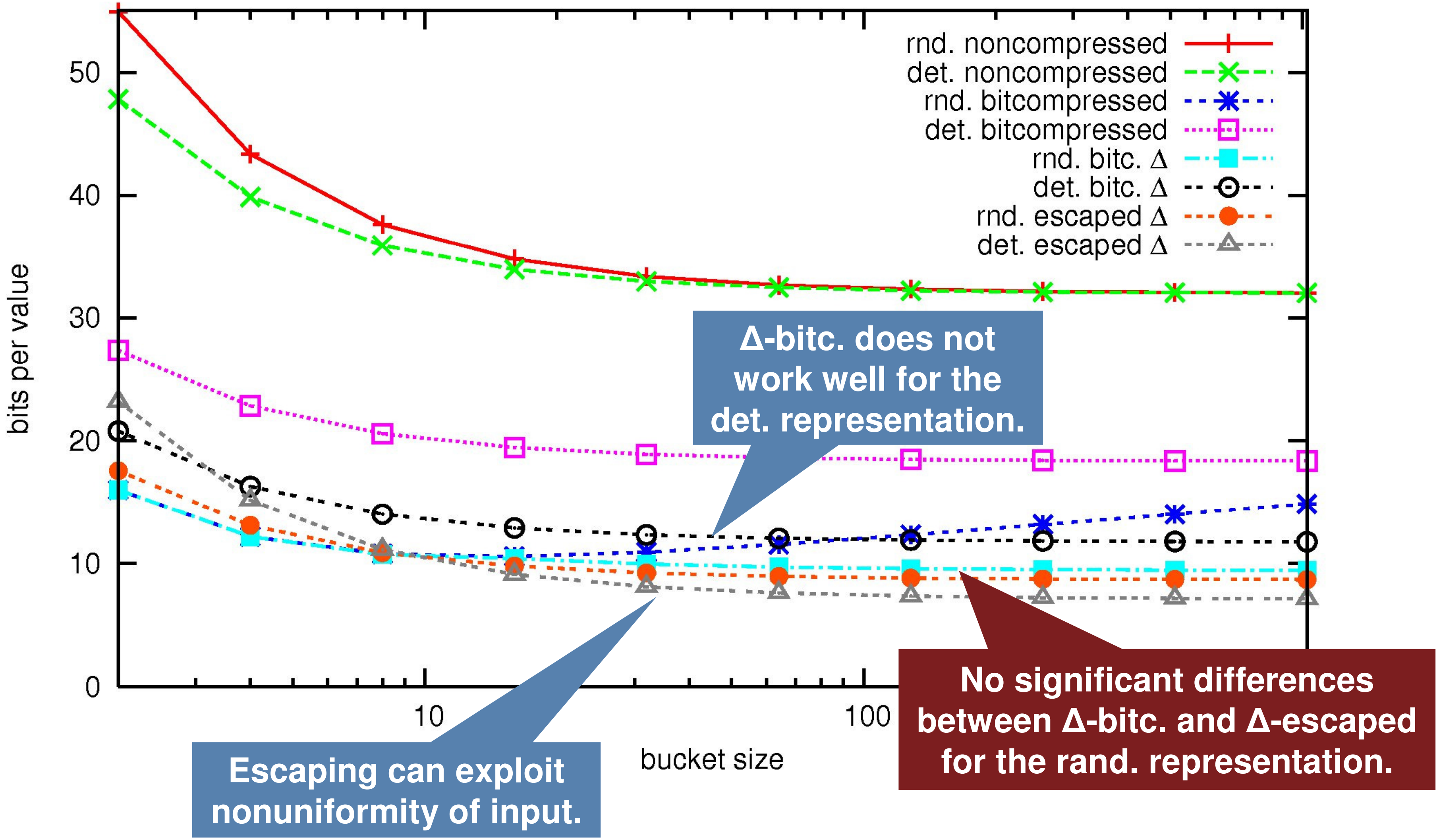
(lookup table size: $\lceil \log_2(n/B) \rceil$)

- *Lookup* is an intersection algorithm that runs on this data structure:

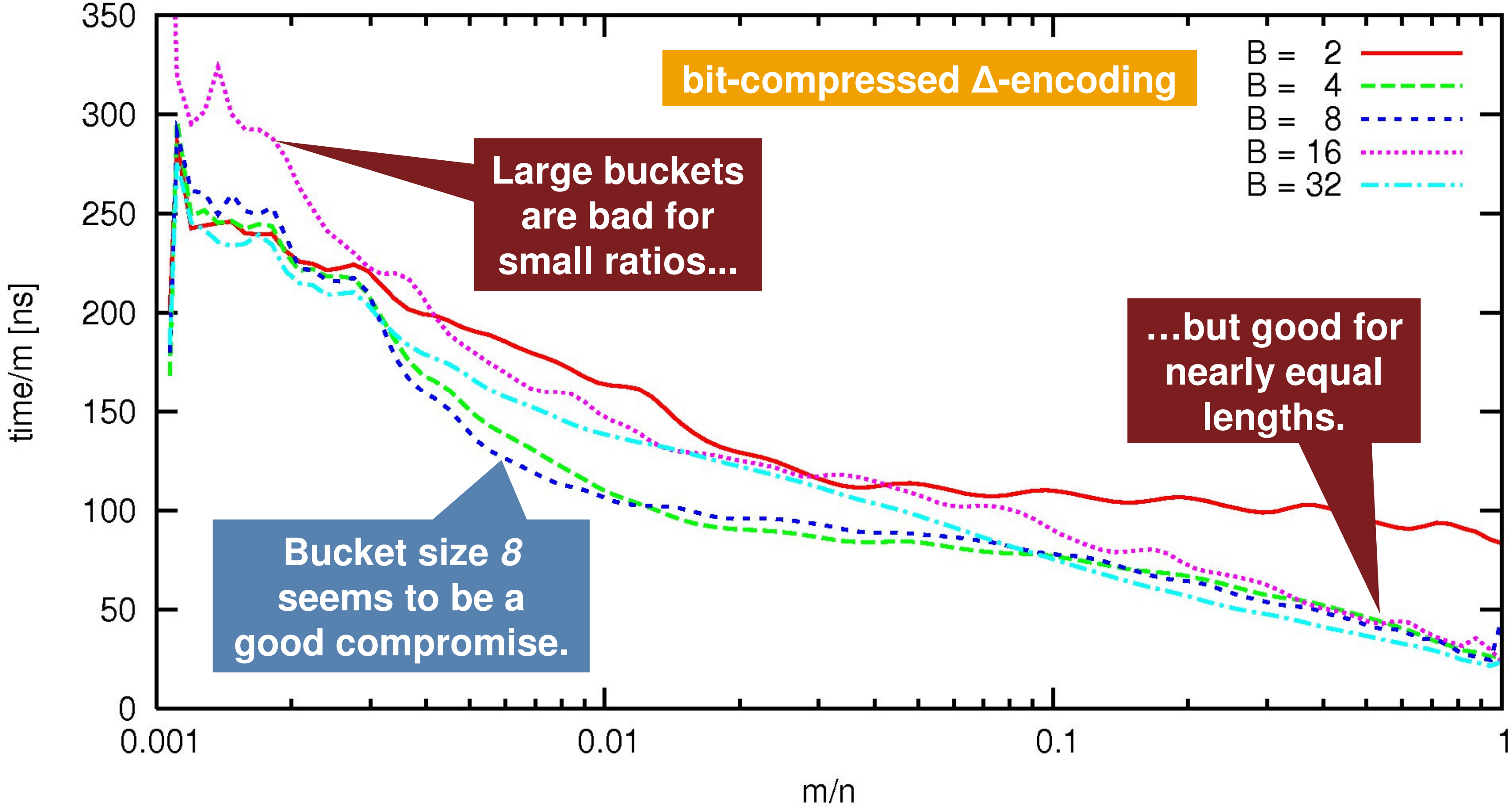
```
Function lookup(M,N)
      O := {}                                            // output
      i := -1                    // current bucket key (now a dummy)
      foreach d ∈ M do                                   // unpack M
            h := d >> k_N                               // bucket key
            l := d & (2^{k_N}-1)                // least significant bits
            if h > i then                            //  a new bucket
                  i := h                       // set current bucket
                  j := t[i]                     // get start position
                  e := t[i+1]                   //  get end position
            while j < e do                   // bucket not exhausted
                  l' := N[j]                  // unpack if necessary
                  if l ≤ l' then
                        if l = l' then O := O ∪ {d}
                        break while
                  j++
      return O
```

Lookup runs in expected time O(m+min{n,Bm})

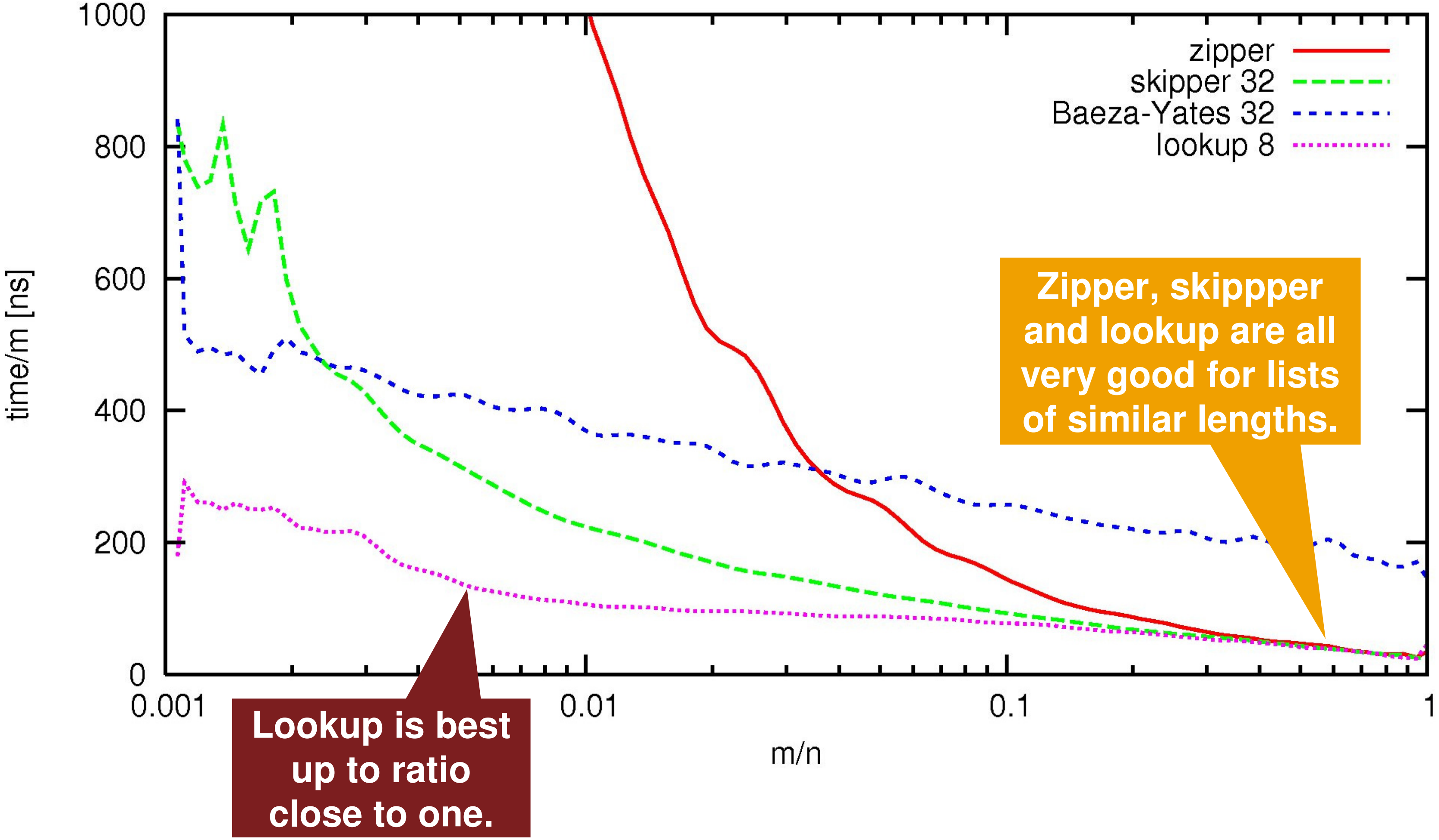
Experiments: Space Consumption (WT2g)



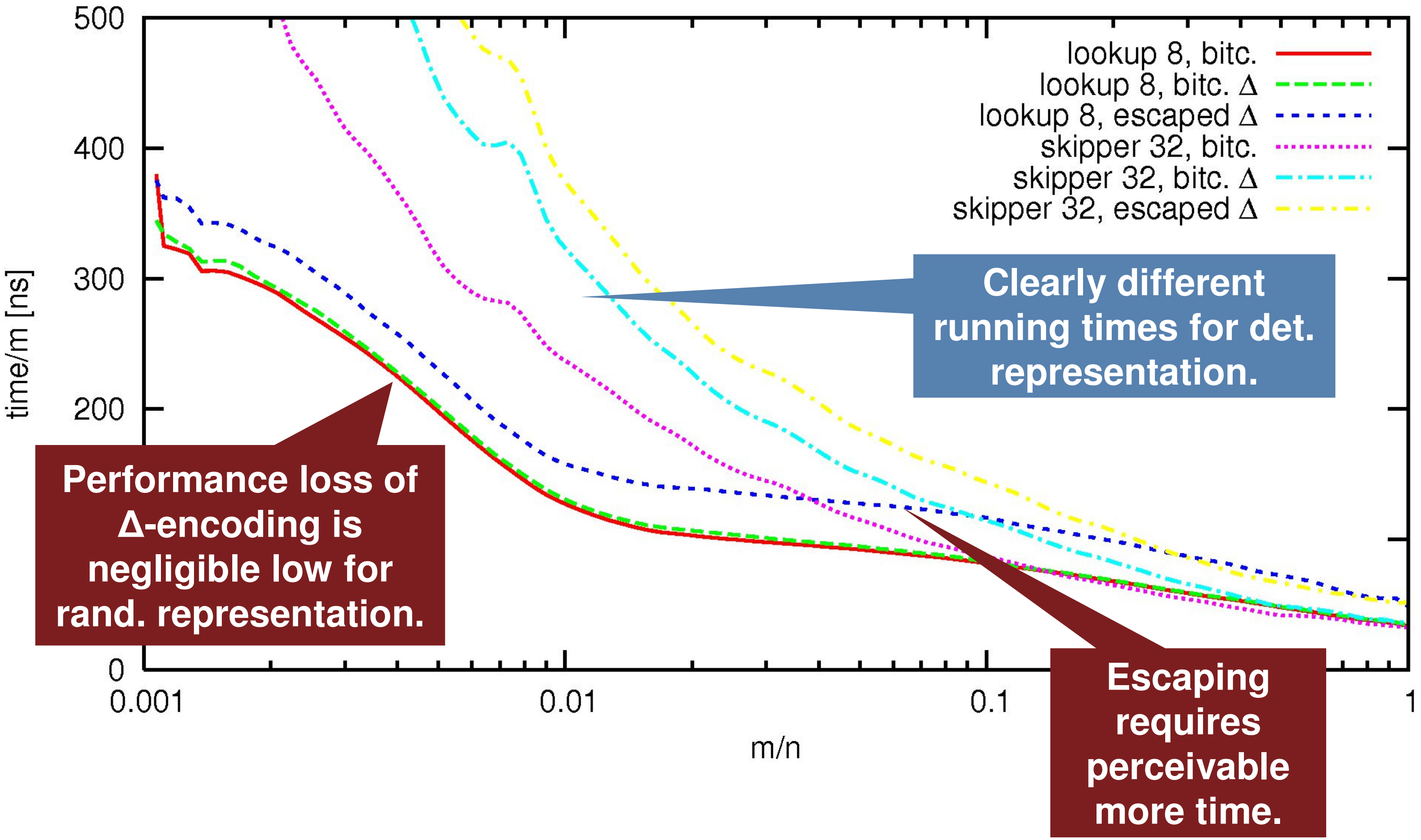
Experiments: Performance of *Lookup* (WT2g)



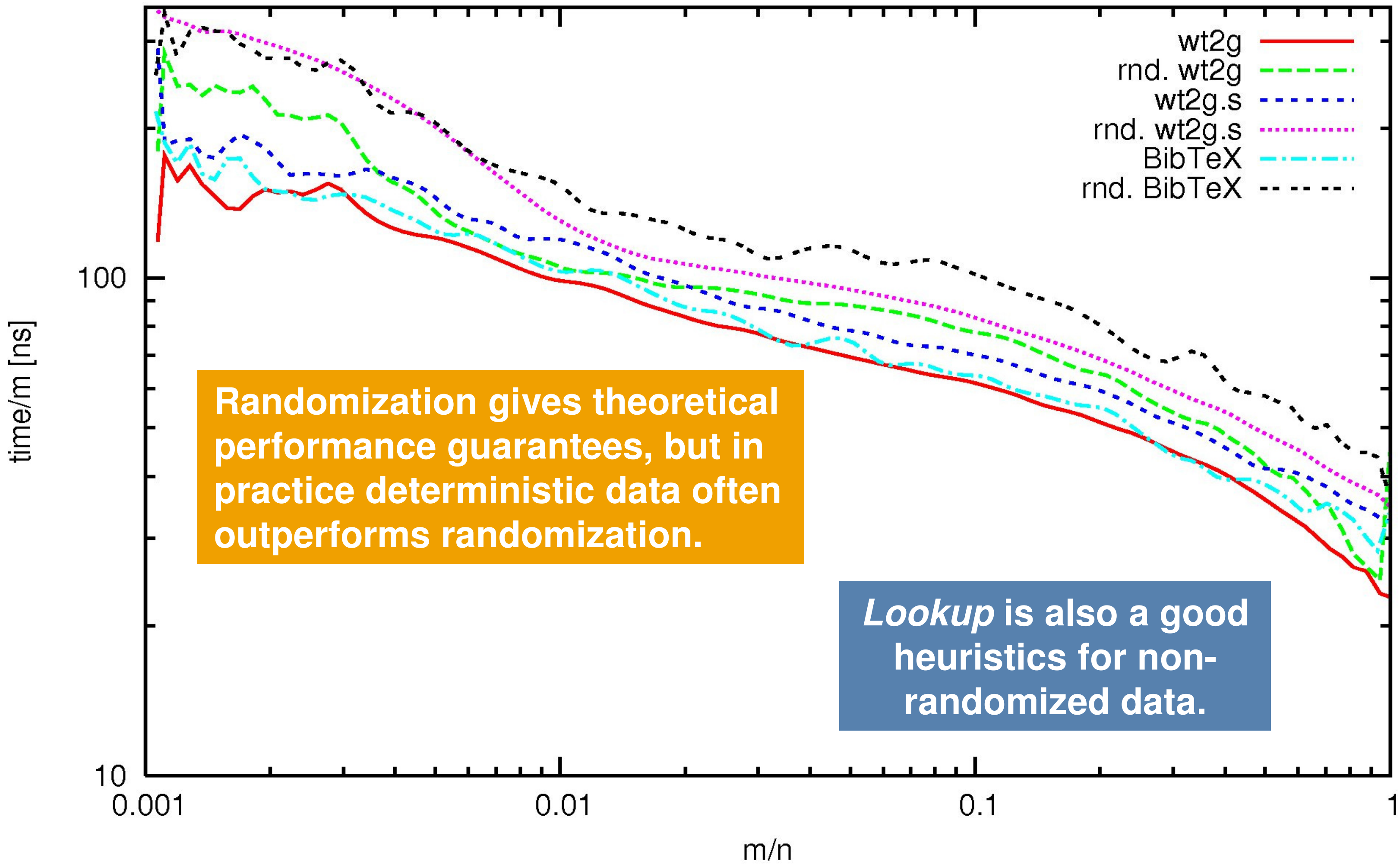
Experiments: Performance Comparison (WT2g)



Experiments: Space-time Tradeoff of Encodings (WT2g.s)



Experiments: Impact of Randomization on *Lookup*



Randomization gives theoretical performance guarantees, but in practice deterministic data often outperforms randomization.

***Lookup* is also a good heuristics for non-randomized data.**

Suffix Arrays

- Suffix Arrays as full-text index: concatenate all text documents.
- Phrase search: A home match for SAs.
- AND searches:
  - search for terms
  - Intersect the occurence lists.
- Document reporting
- Compressed or distinct SAs are available from the Pizza&Chili website:

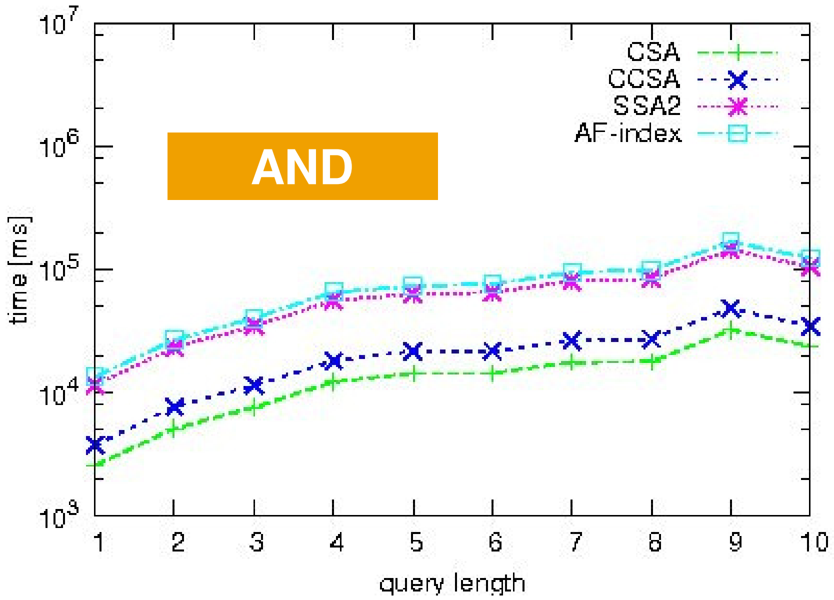
  `http://pizzachili.di.unipi.it`

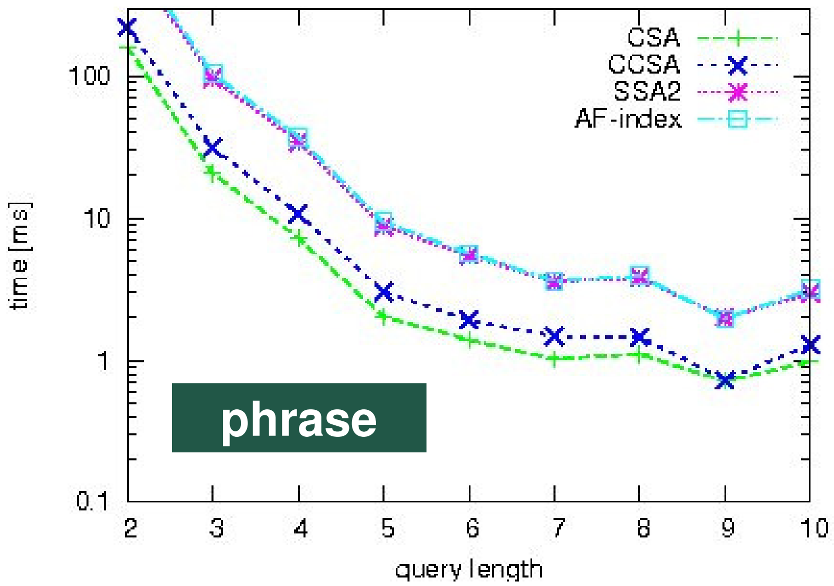
```
a
abra
abracadabra
acadabra
adabra
bra
bracadabra
cadabra
dabra
ra
racadabra
```

Suffix array of „abracadabra“

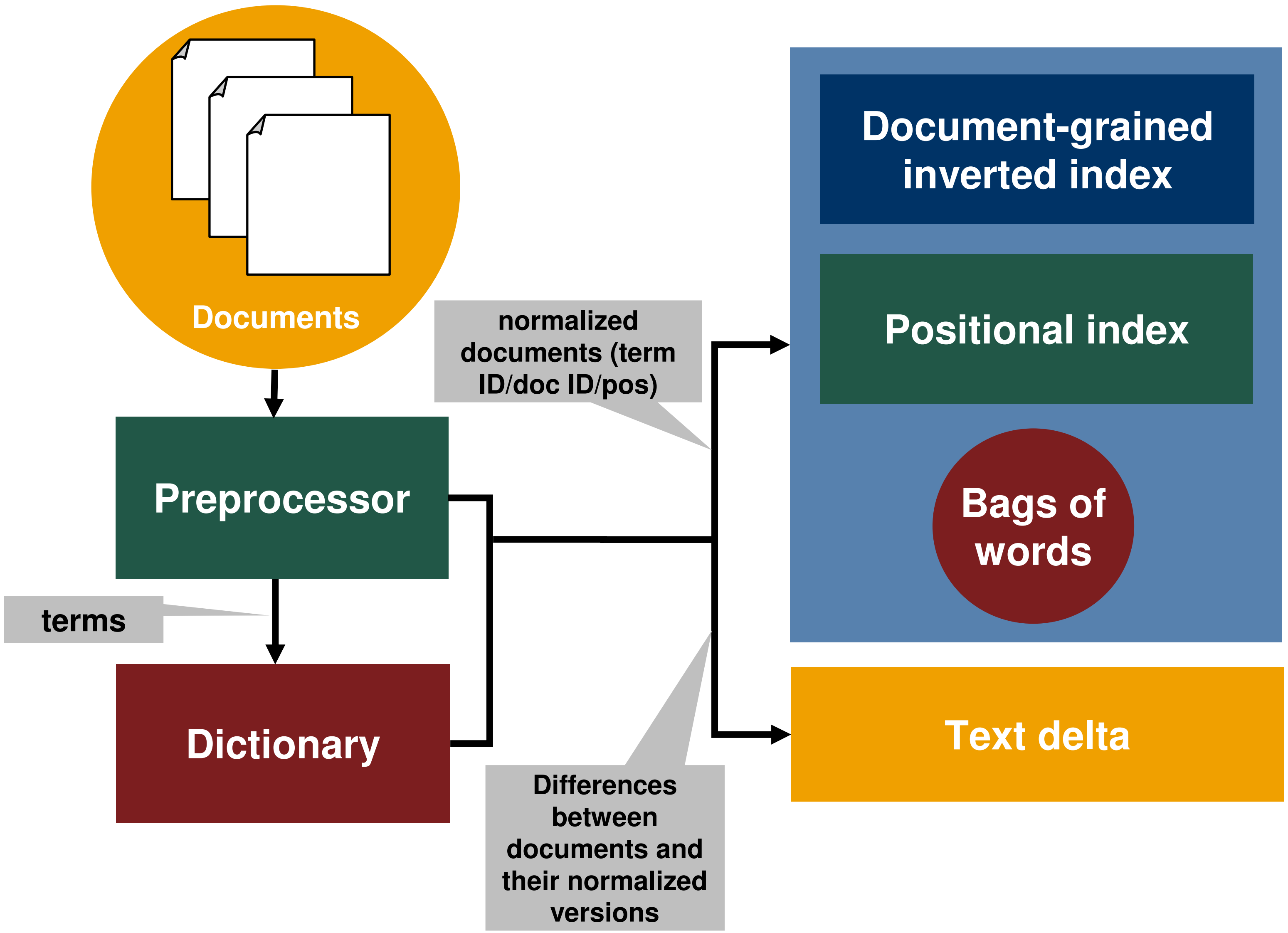
Performance of different SAs on WT2g 1-50000

	CSA	CCSA	SSA2	AF-index
size [MB]	**230.9**	500.9	302.6	279.3
compression	**0.64**	1.39	0.84	0.77
indexing time [min]	**9.3**	11.5	8.7	23.0
peak mem usage [GB]	3.2	3.1	**2.1**	3.1

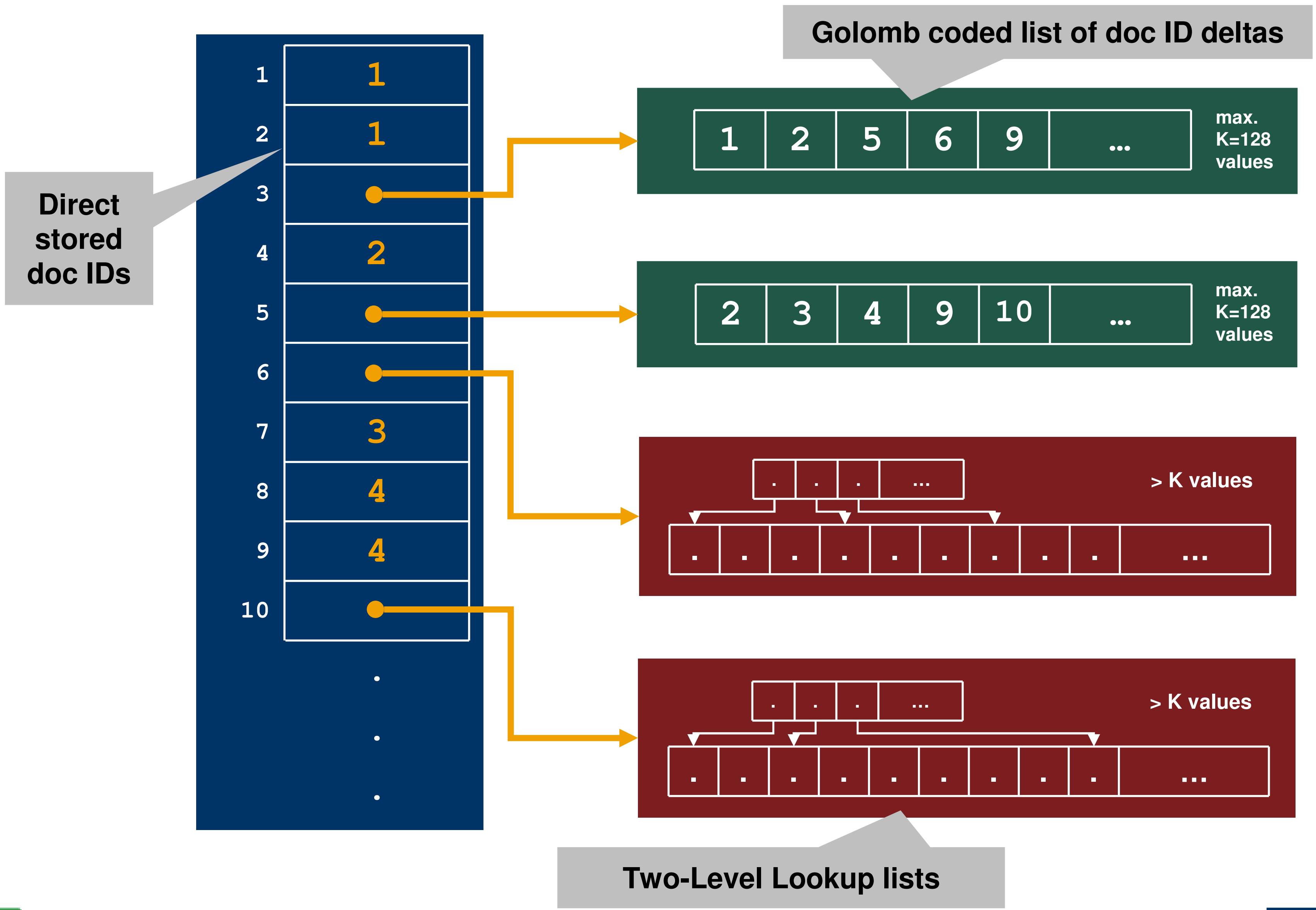




Modular Design of the Compressed Inverted Index (CII)



CII: Document-grained Inverted Index



Document-grained Index: Lookup List

- Two-level data structure, refinement of [Sanders and Transier, 2007]:
  - Split the range of document IDs into buckets based on their most significant bits.
  - **Lookup-table**: direct access to the first value of a bucket
  - **Rank information**: number of values smaller than the contents of a bucket.
  - **Variable Golomb coding***:* estimating the average of each bucket.

***Inverted List* with lookup table**

lookup – table

	pos	rank
00	0	0
01	28	4
10	56	8
11	84	12

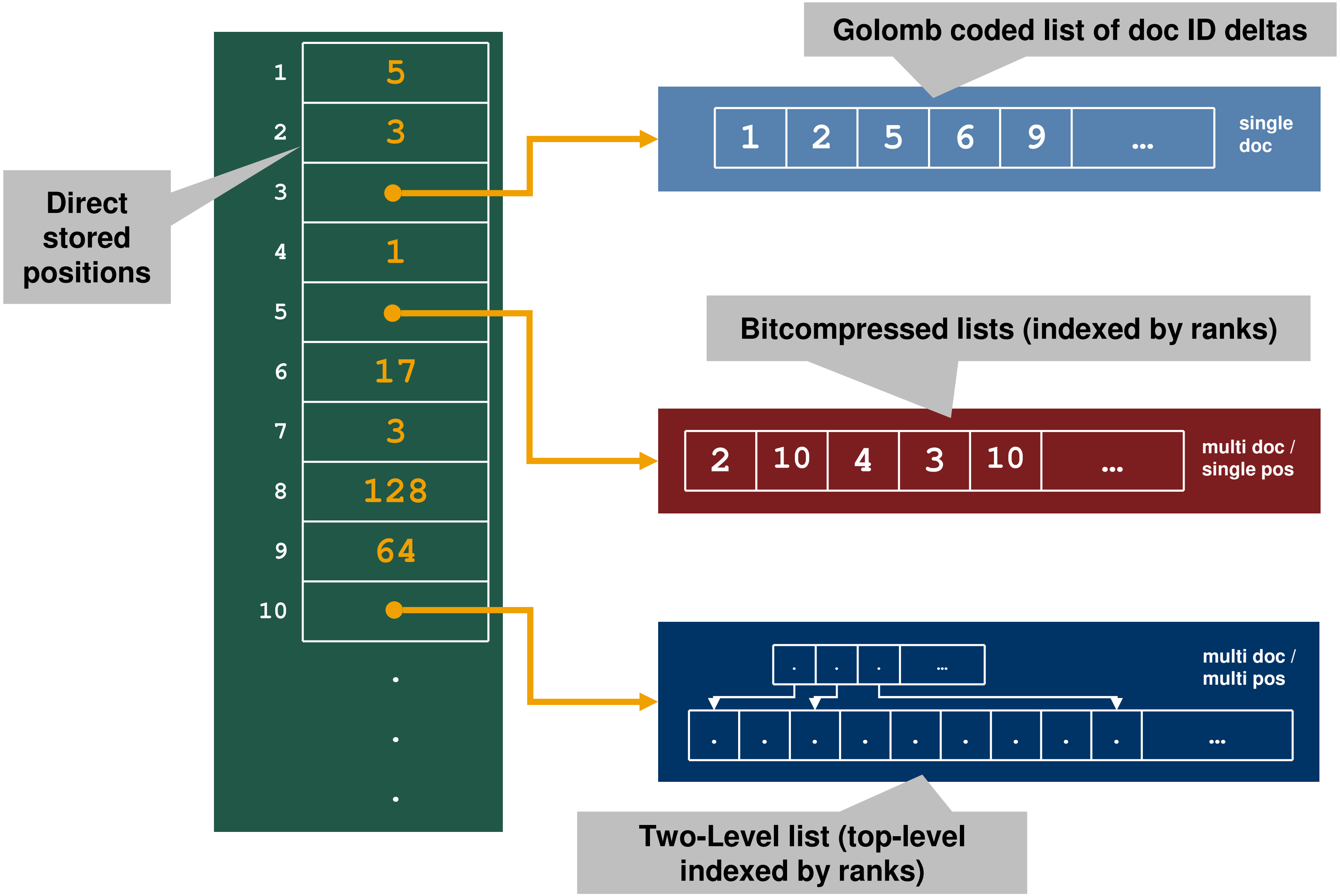
B = 4

0000111	0
0001111	
0011000	
0011111	
0101000	4
0101110	
0110111	
0111101	
1000100	8
1001011	
1010100	
1011011	n-1

n=12

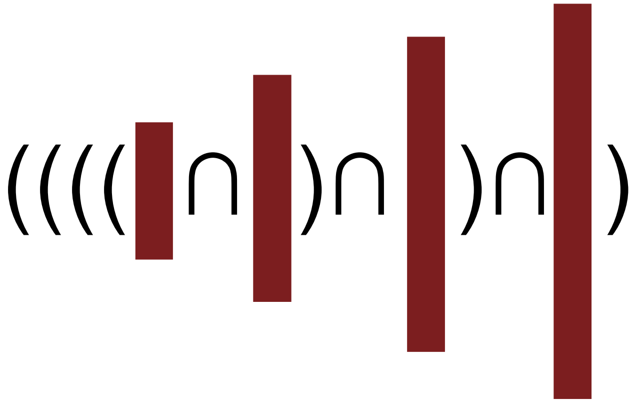
(lookup table size: $\lceil \log_2(n/B) \rceil$)

CII: Positional Index



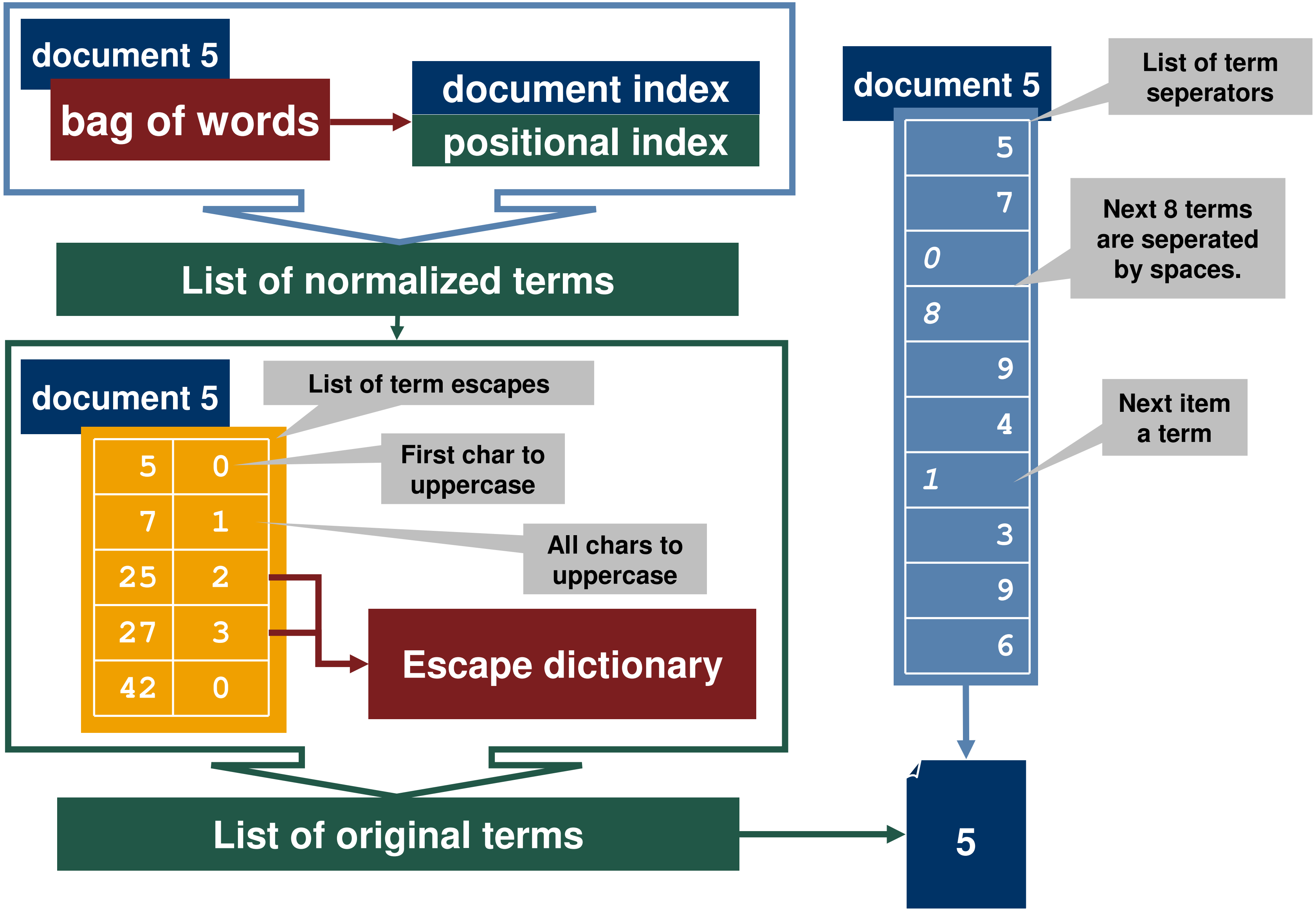
Querying the Compressed Inverted Index

- **AND Query: Intersection of inverted lists in increasing order of list lengths.**



- **Phrase Query:**
  - **As AND queries, but keeping track of the current ranks.**
  - **Retrieve corresponding position lists.**
  - **Check positions.**

CII: Document Reporting with Text Delta



Experiments: Test System

- We have implemented all algorithms using C++.
- One core of an Intel Core 2 Duo E6600, clocked at 2.4 GHz with 2 x 2MB L2 cache and 4 GB main memory.
- openSuSE 10.2 (kernel 2.6.18), gcc 4.1.2 (-O3)
- Timing with PAPI 3.5.0

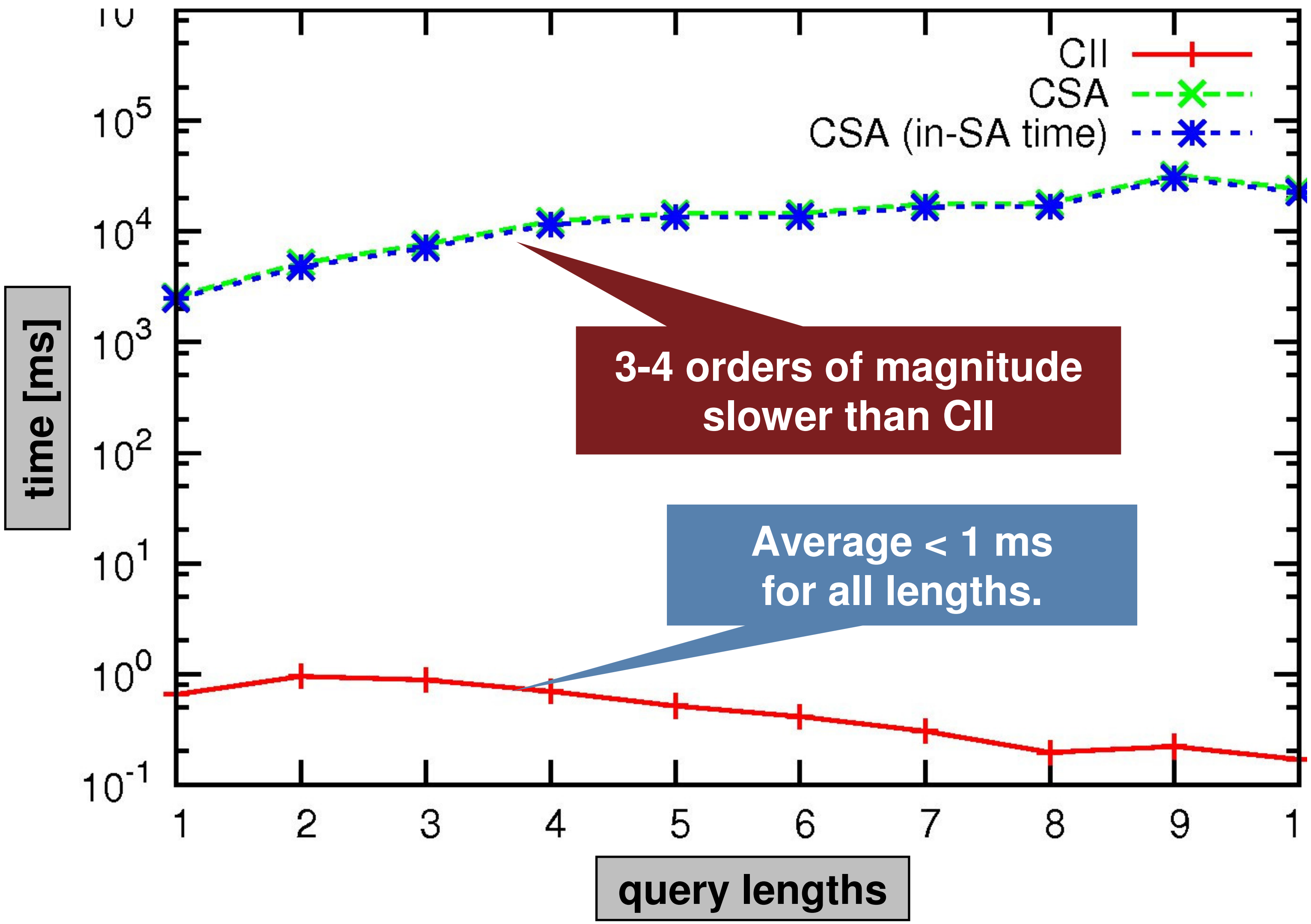
Experiments: Test Data

- Real world instance: first 50.000 docs of WT2g.
- Pseudo real-world queries:
  - AND and phrase queries.
  - Selecting random hits.
  - Query lengths: 1-10 terms.

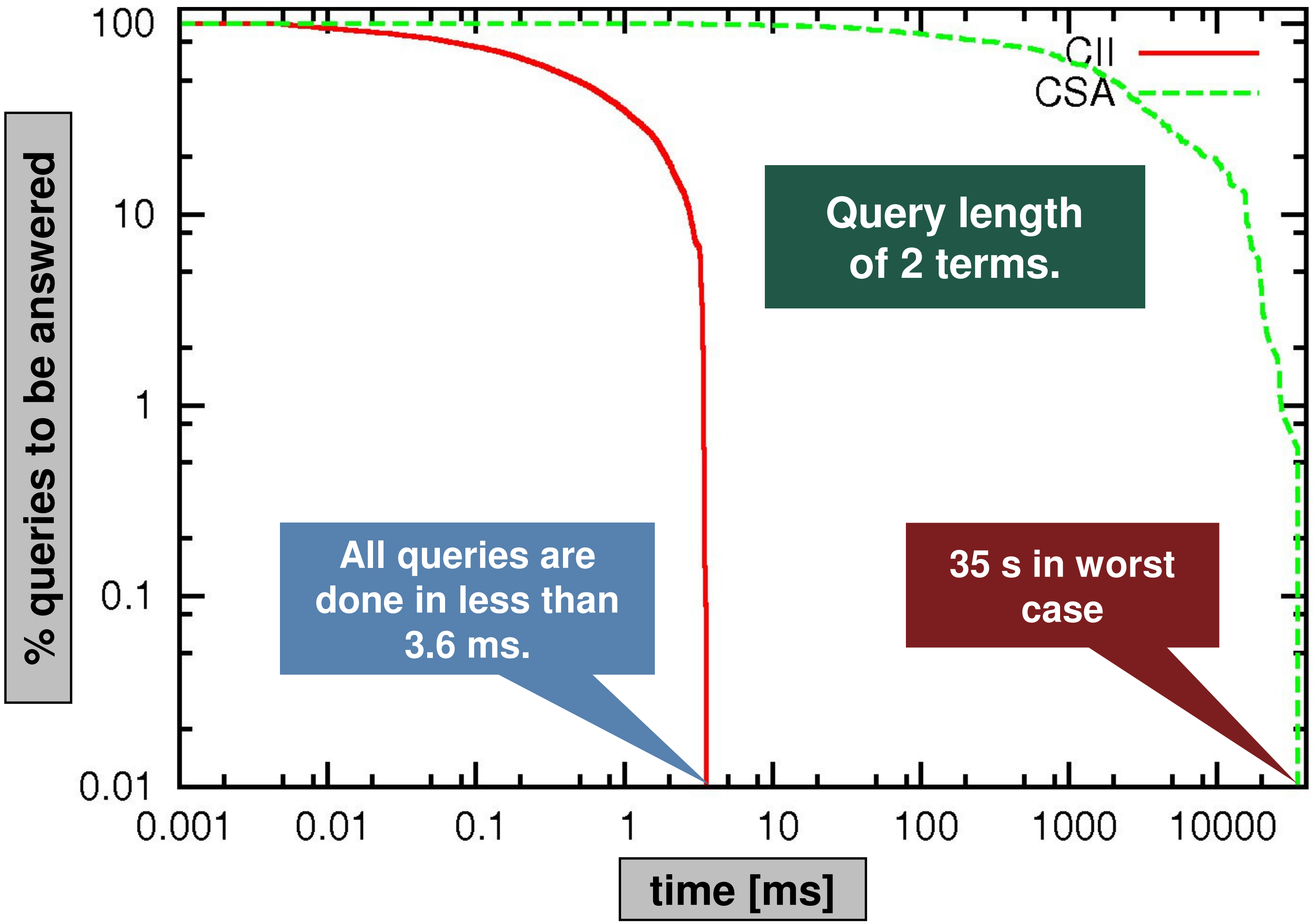
Indexing: Space requirements

	CII	CSA
dictionary	23.9	
document Index	32.4	
positional Index	126.3	
bag of words	25.1	
suffix array		230.8
doc bounds		0.1
sum [MB]	**206.1**	**230.9**
text delta [MB]	108.7	
sum + text delta [MB]	**314.8**	
input size (norm.)	**412.8 (360.5)**	
compression	**0.76 (0.57)**	**- (0.64)**
indexing time [min]	**5.6 (5.1)**	**- (9.3)**
peak mem usage [GB]	**0.7**	**3.2**

Experiments: Average AND Query Time

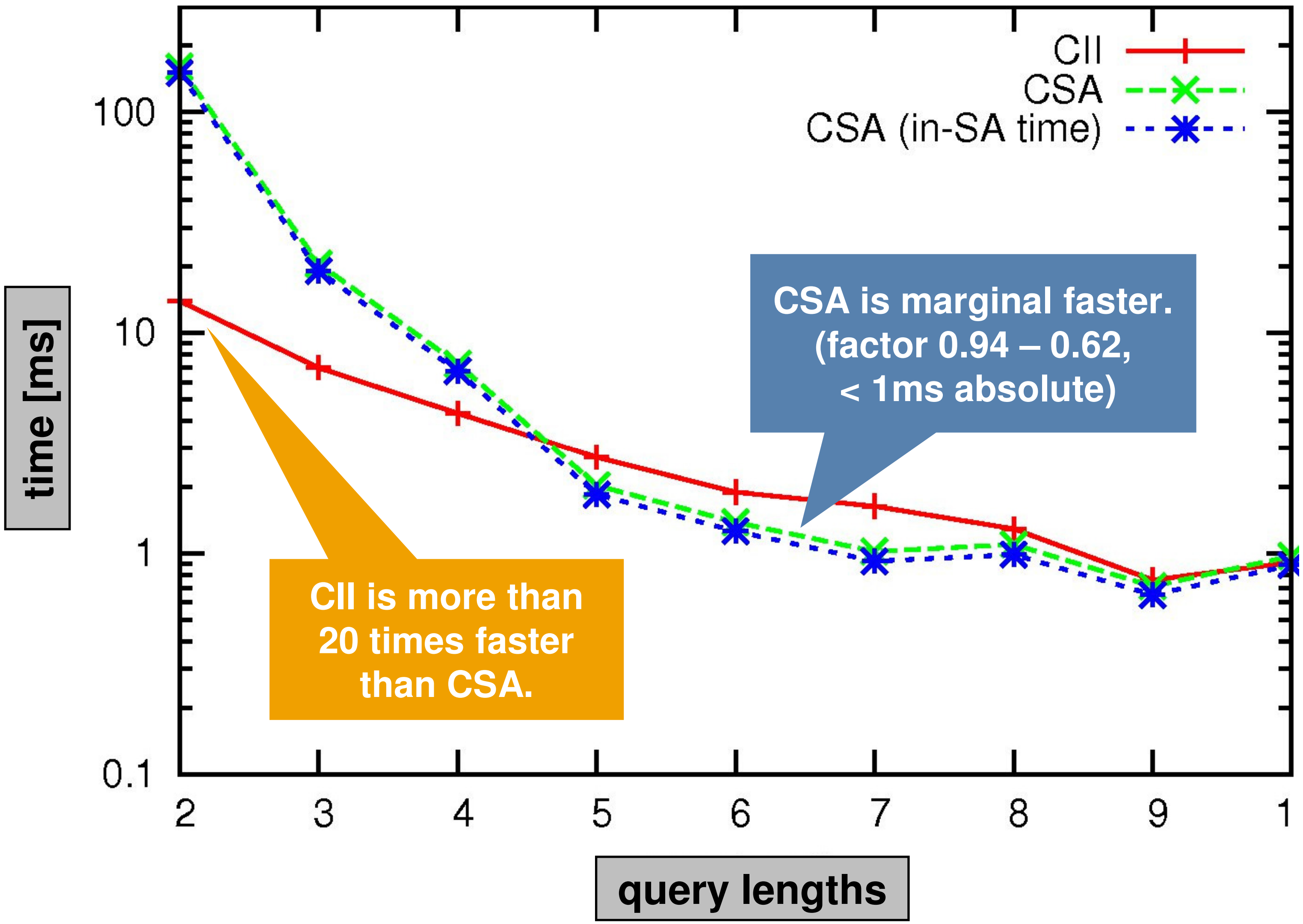


Experiments: How many % AND queries take longer than t?

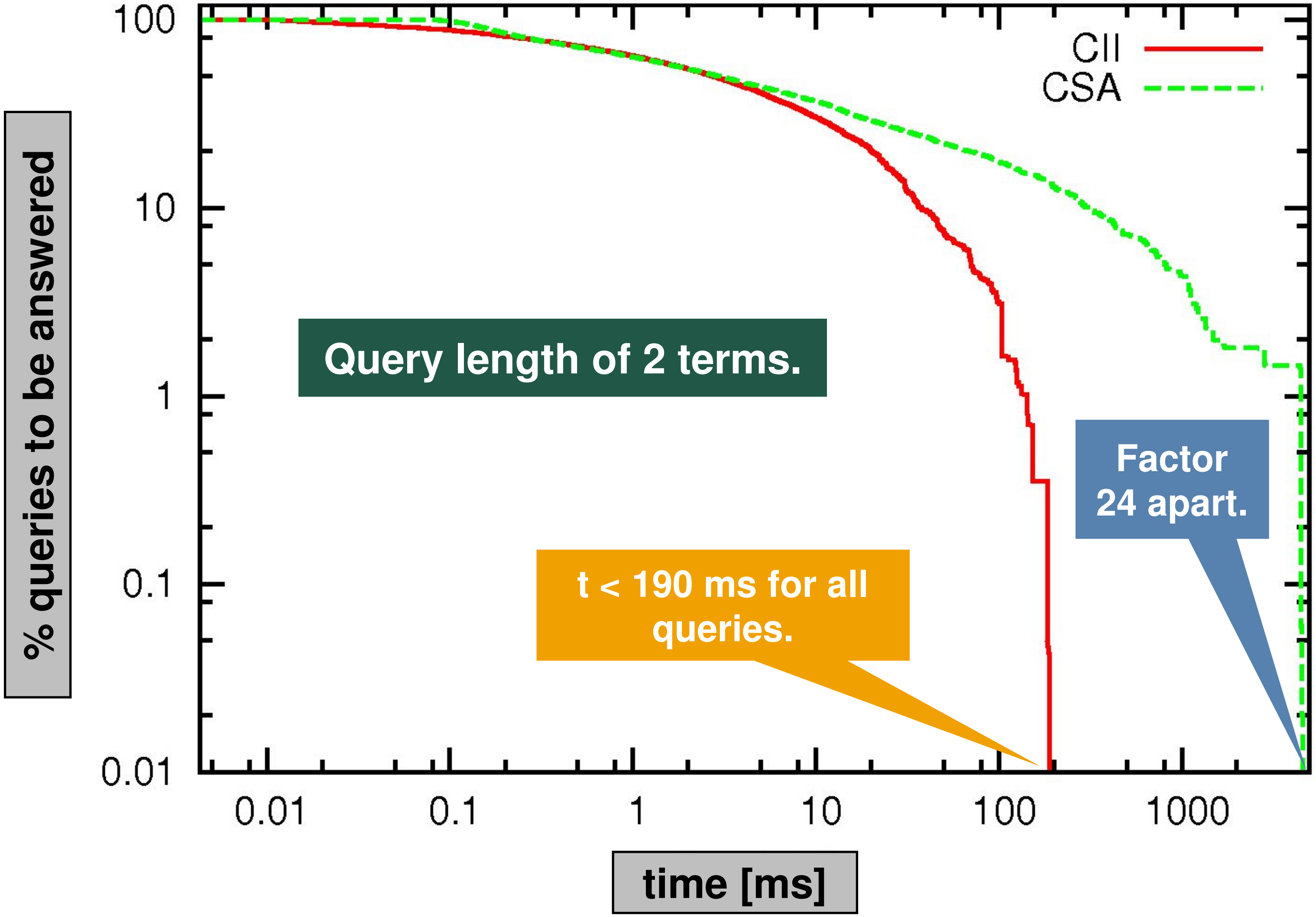


Universität Karlsruhe (TH)
Forschungsuniversität · gegründet 1825

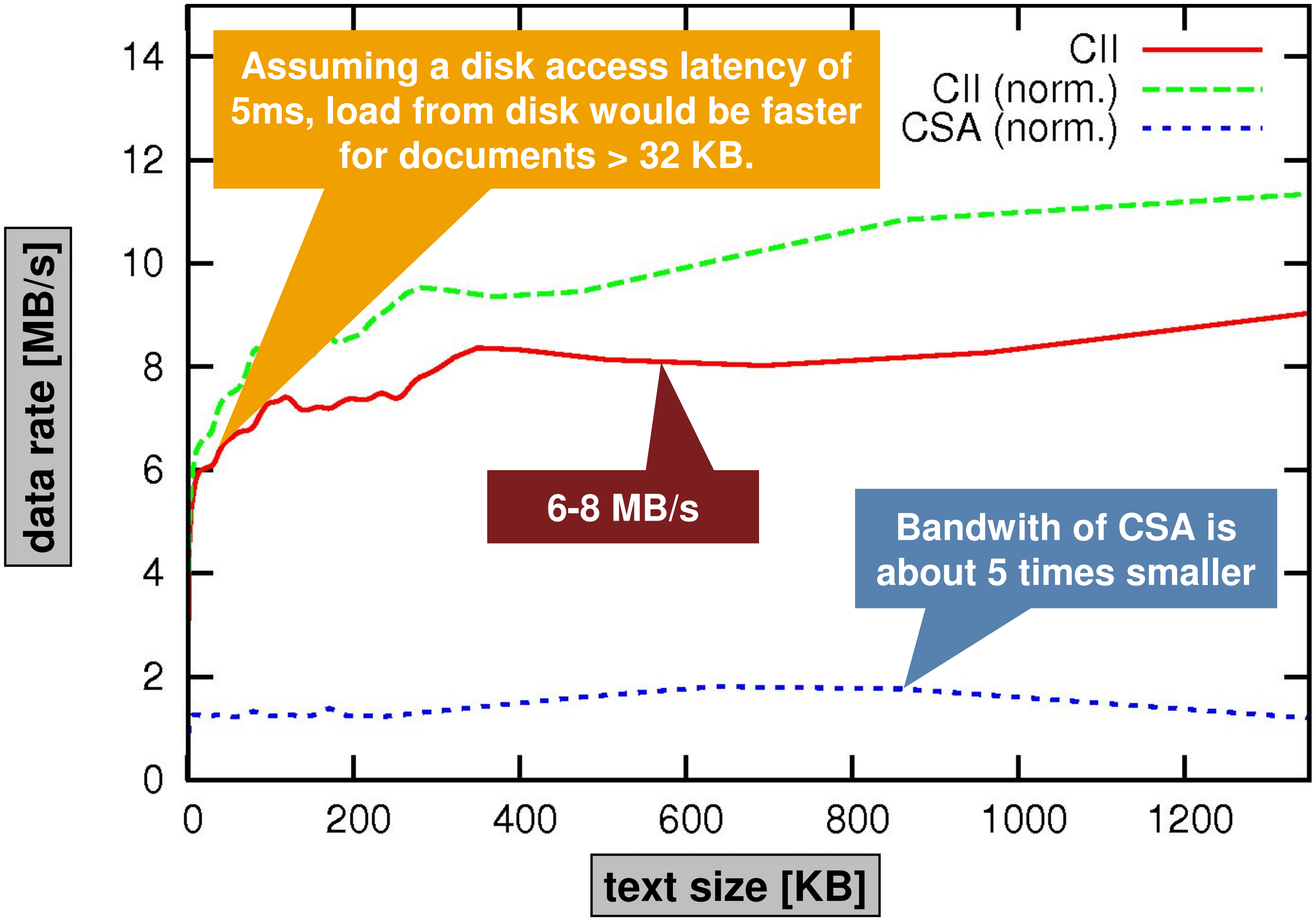
Experiments: Average Phrase Query Time



Experiments: How many % phrase queries take longer than t?



Experiments: Document Reporting

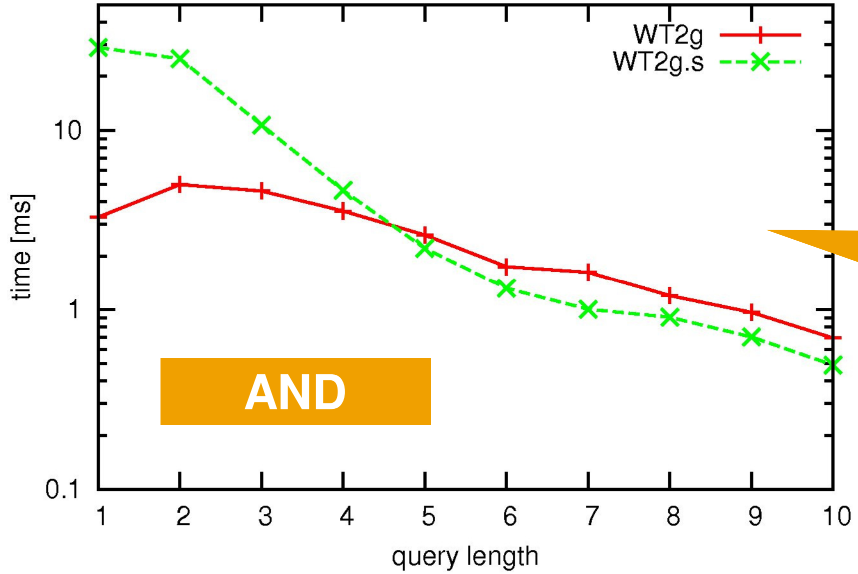


Outlook & Future Work

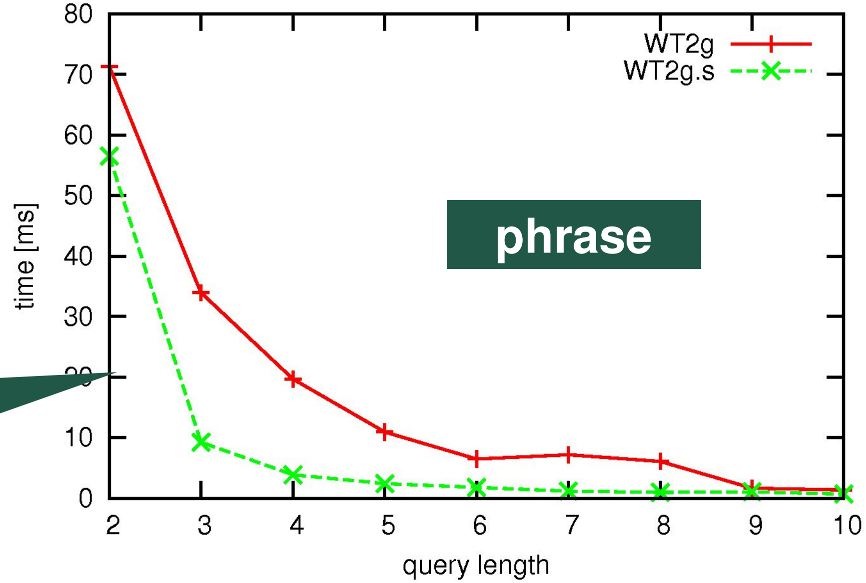
- Bag of words: adaptive coding scheme?
- Compression of the dictionary.
- Speeding up the most expensive phrase queries.
- Construction time?
- Fast updates?

Thank You for Your attention!

CII on the complete WT2g / WT2g.s



short queries are slower on WT2g.s, as there are more results. – Large queries are faster, as they benefit from more lookup lists.



Phrase queries are faster on WT2g.s, because the position lists are shorter.

Copyright 2007 SAP AG. Alle Rechte vorbehalten