Efficient Scoring in Lucene

Stefan Pohl

Nokia Berlin

stefan.pohl@nokia.com

NOKIA

Agenda

- Motivation
- Review: Query Processing Modes in Lucene
- Scoring Efficiency Optimization
- Experiments

Motivation

- Speed !

  Human Reaction Time: 200 ms*
  → Backend latency: << 200 ms

- Load ?

  → Secs / Q ↓ *means* Q / secs ↑

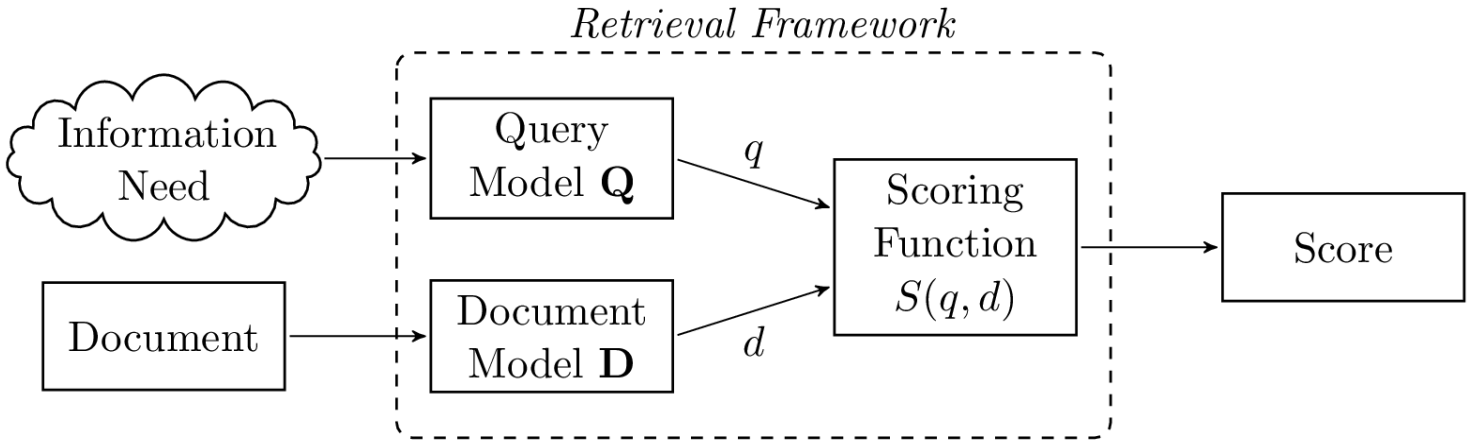
- Why not *Scale Out* ?

  → Costs

* Steven C. Seow, Designing and Engineering Time: The Psychology of Time Perception in Software, Addison-Wesley Professional, 2008.

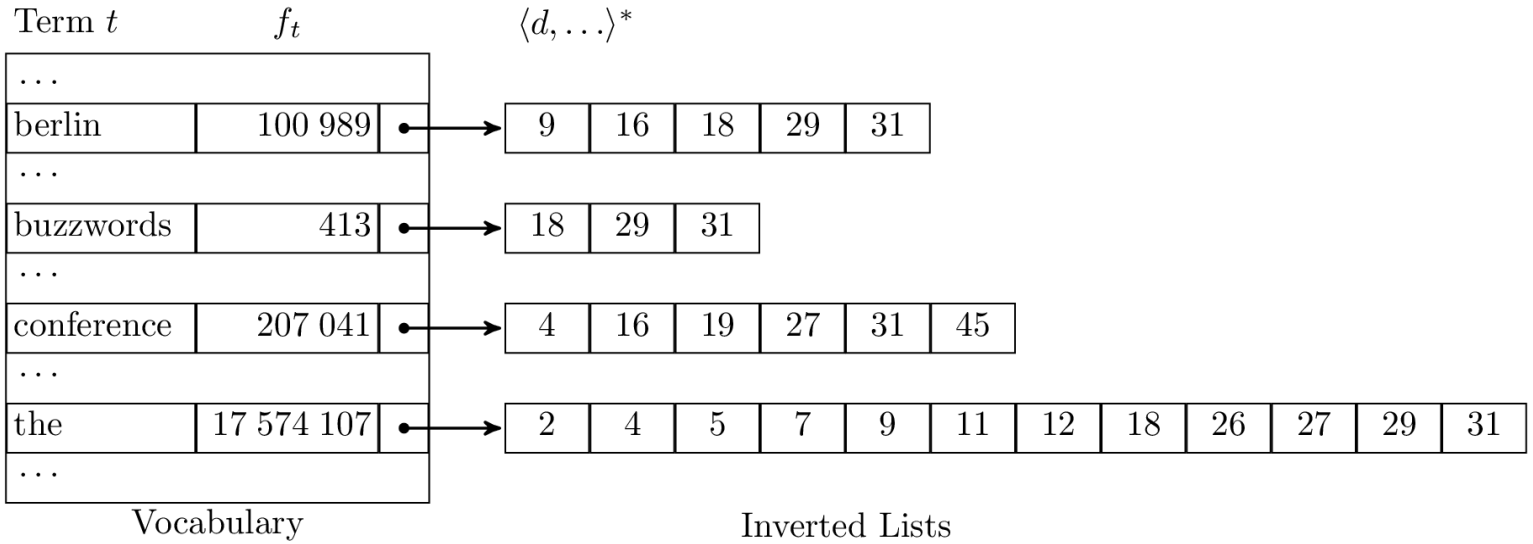
Ranked Retrieval in IR Engines

- Conceptually:



→ sort docs by score (descending)

- Technically:



Running Example

- **Collection**
  24 900 500 docs, 1kB each, from English Wikipedia (used in Lucene's nightly benchmark: http://people.apache.org/~mikemccand/lucenebench)

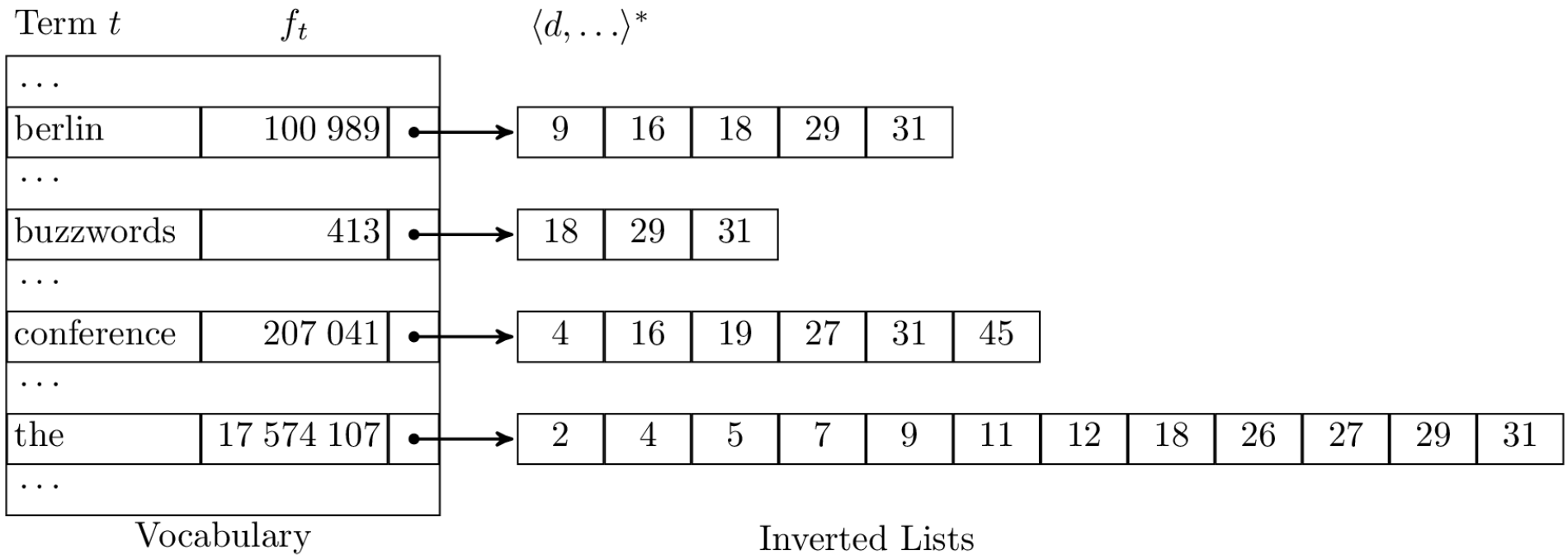
- **Query:** ”The Berlin Buzzwords Conference”
  - 10 results queried

- **Stats:**

Term t	**Doc. Freq. f_t**
The	17,574,107
Berlin	100,989
Buzzwords	413
Conference	207,041

Conjunctions (AND)

”+The +Berlin +Buzzwords +Conference”

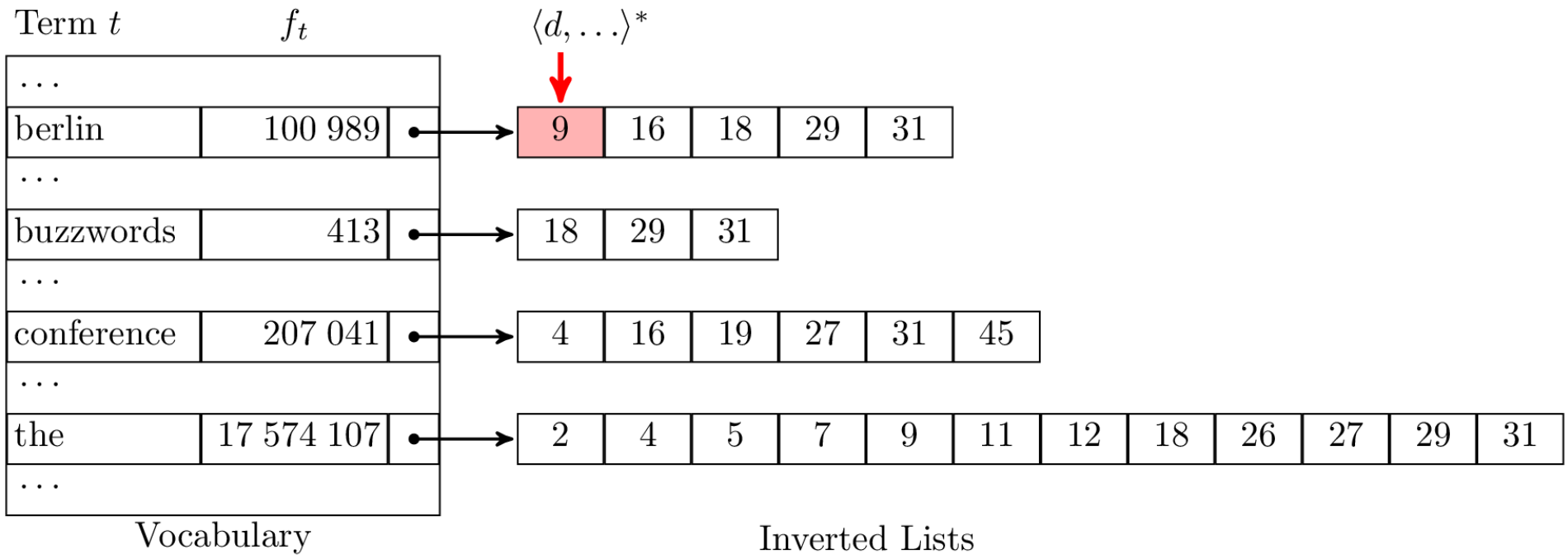


→ *Matching* requirement:
All terms *MUST** occur in result docs

* see *o.a.l.search.BooleanClause.Occur.MUST*

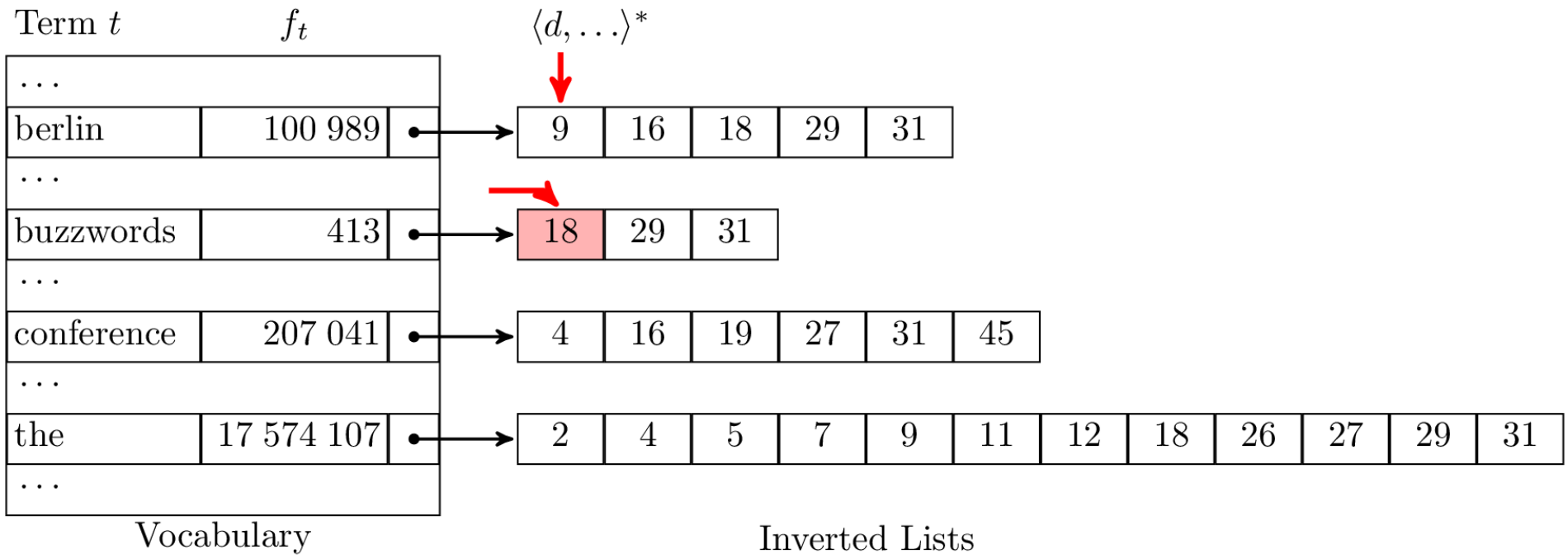
Conjunctions (AND)

”+The +Berlin +Buzzwords +Conference”



Conjunctions (AND)

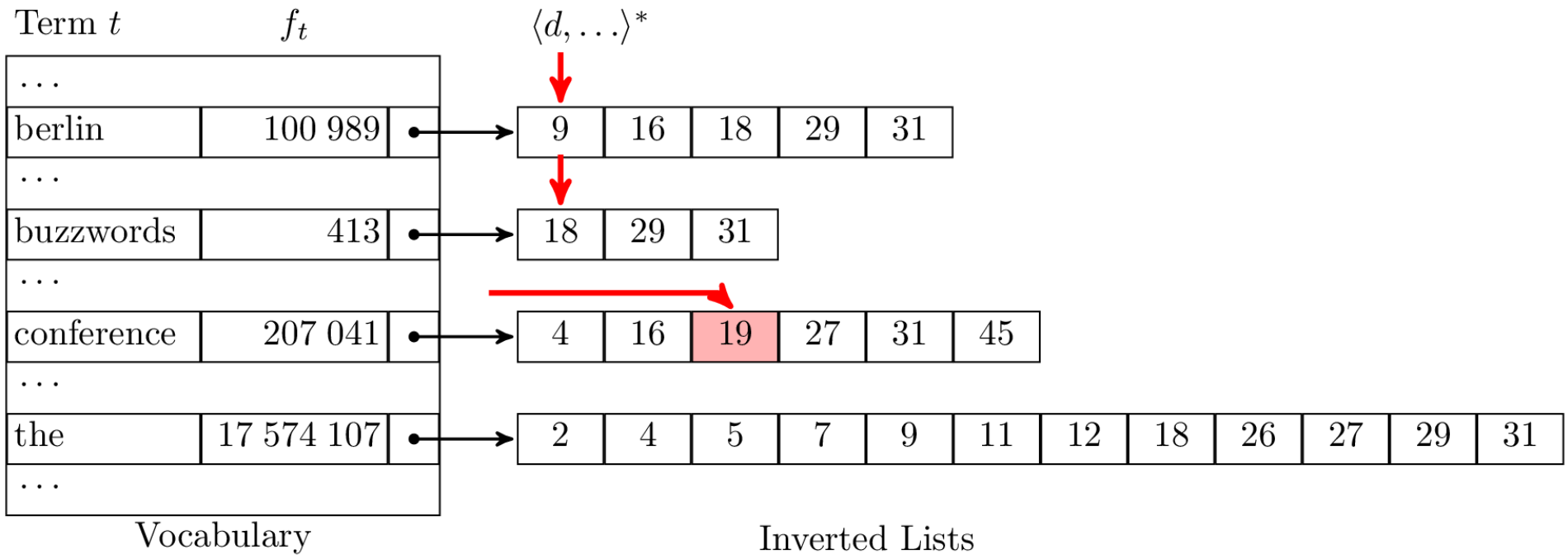
”+The +Berlin +Buzzwords +Conference”



→ *advance(9)* *...uses skip lists*

Conjunctions (AND)

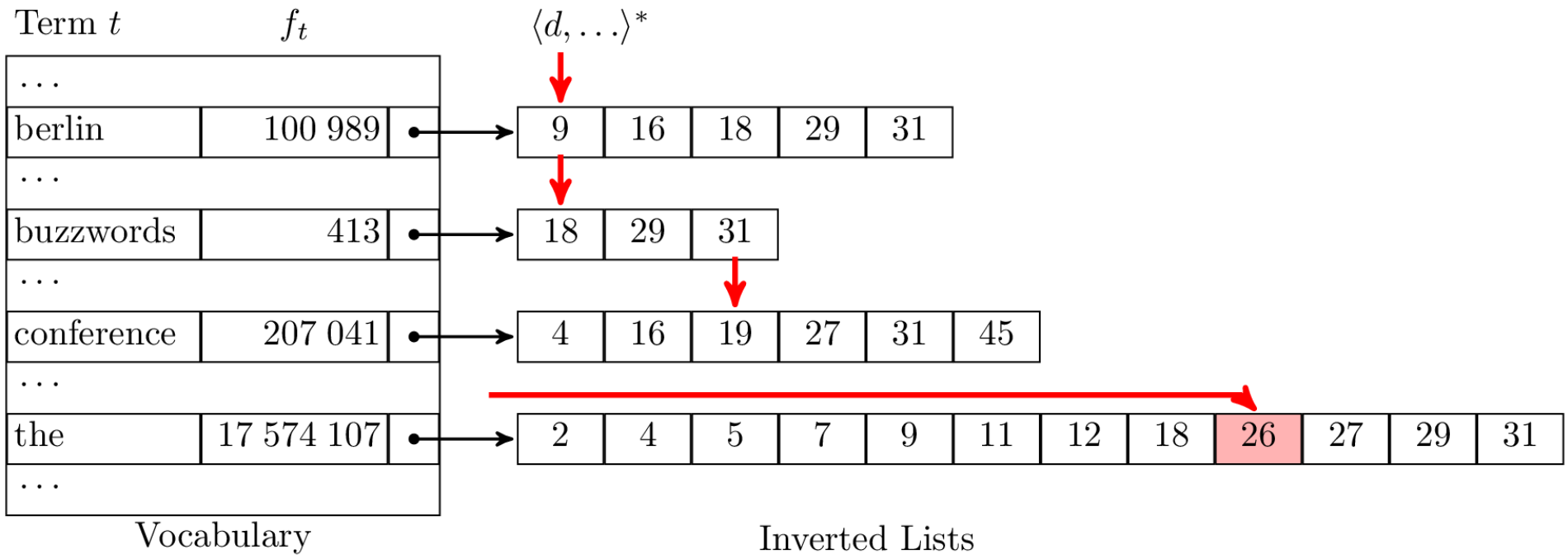
”+The +Berlin +Buzzwords +Conference”



→ *advance(18)*

Conjunctions (AND)

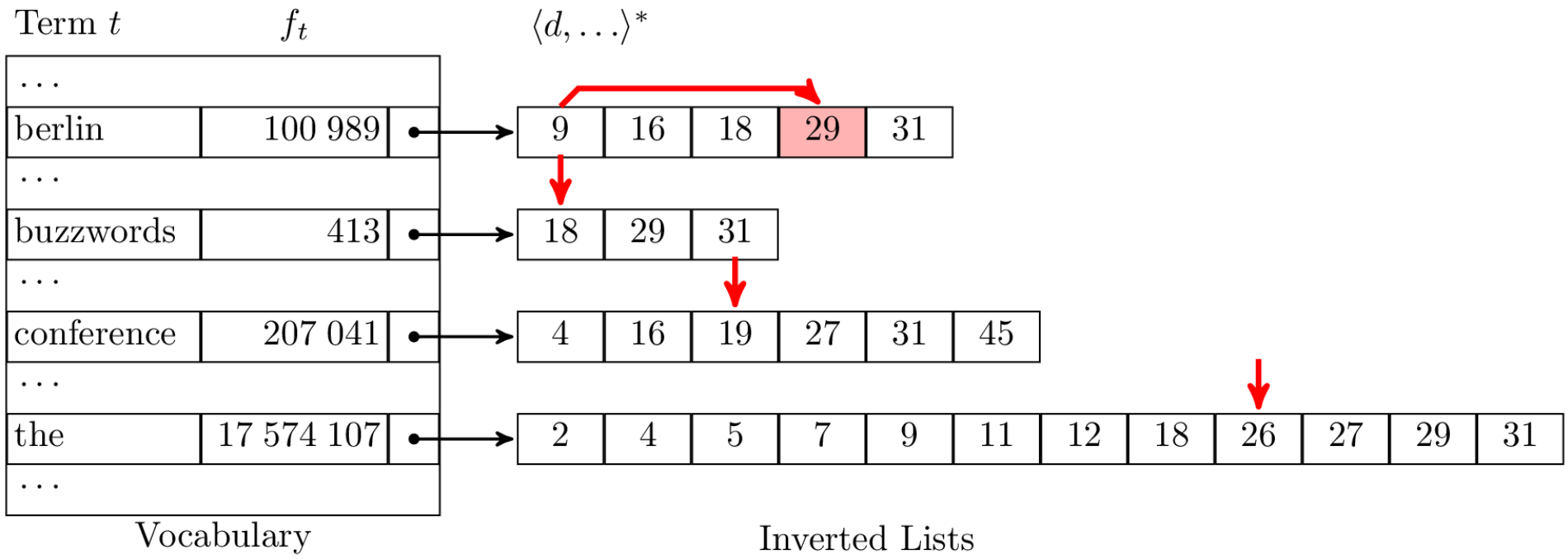
”+The +Berlin +Buzzwords +Conference”



→ *advance(19)*

Conjunctions (AND)

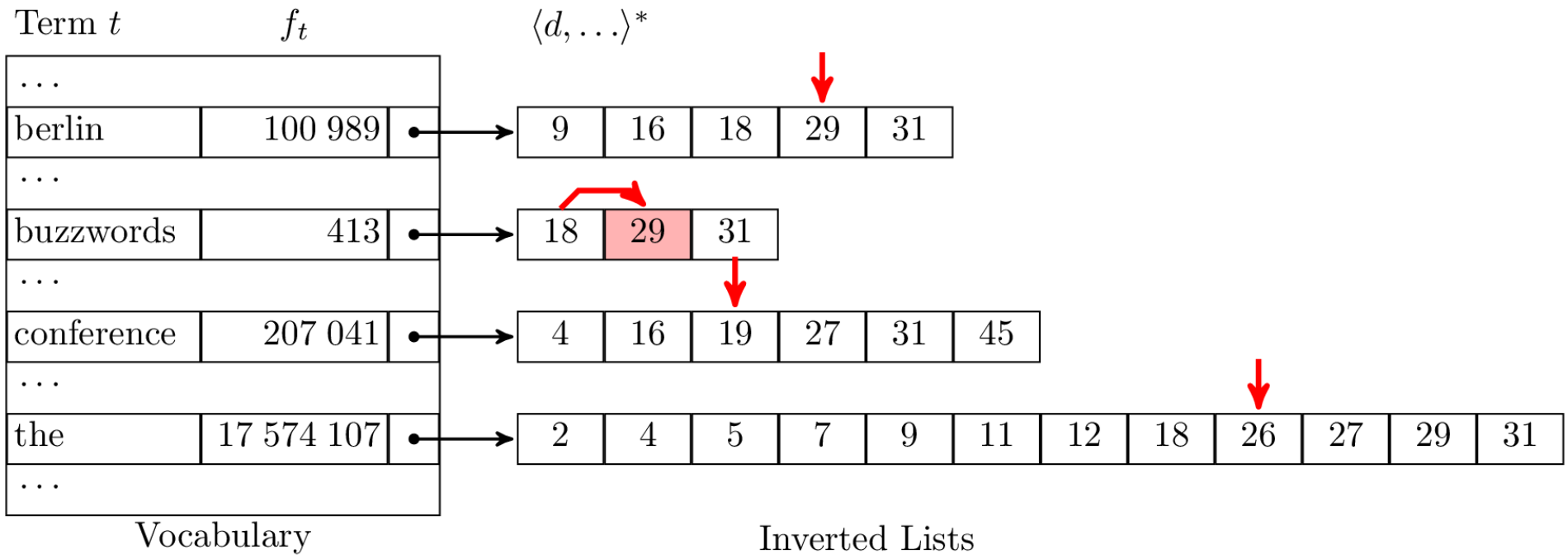
”+The +Berlin +Buzzwords +Conference”



→ *advance(26)*

Conjunctions (AND)

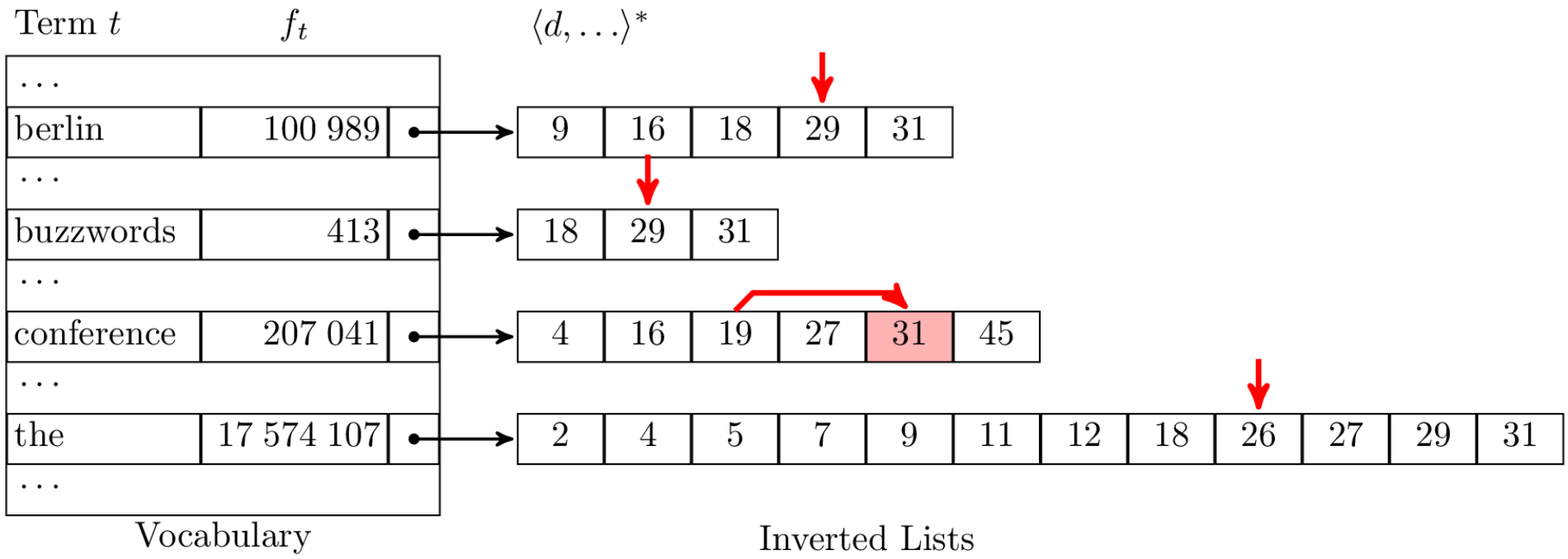
”+The +Berlin +Buzzwords +Conference”



→ *advance(29)*

Conjunctions (AND)

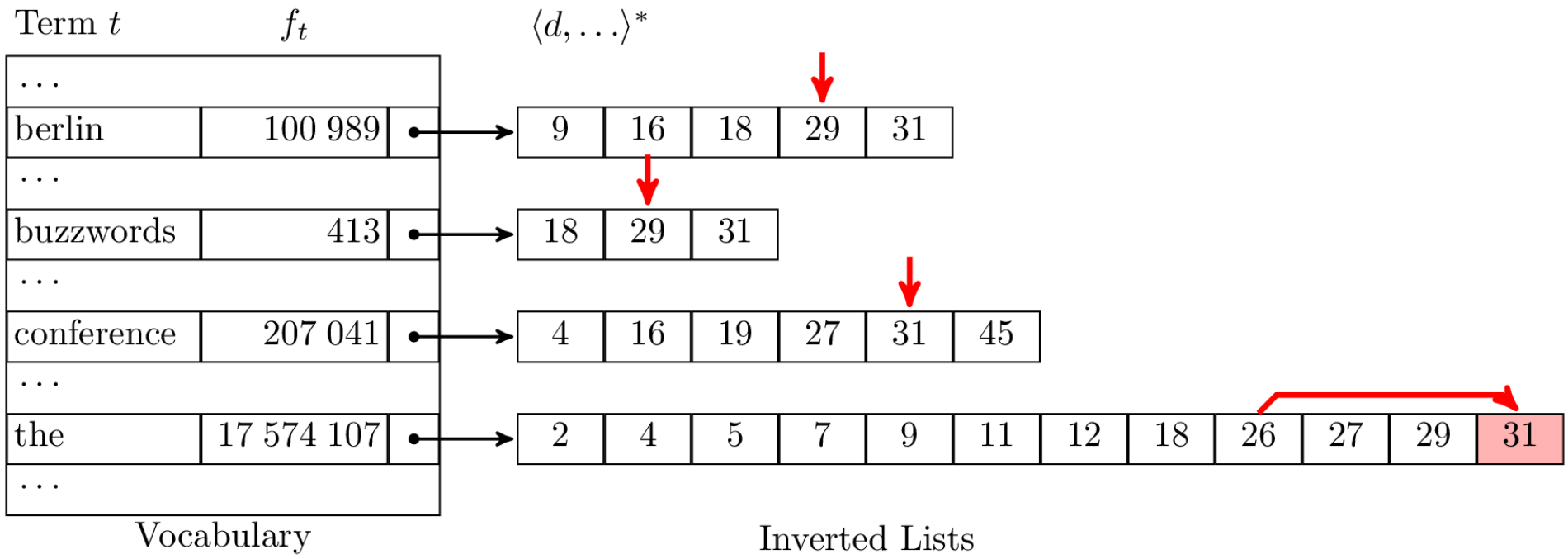
”+The +Berlin +Buzzwords +Conference”



→ *advance(29)*

Conjunctions (AND)

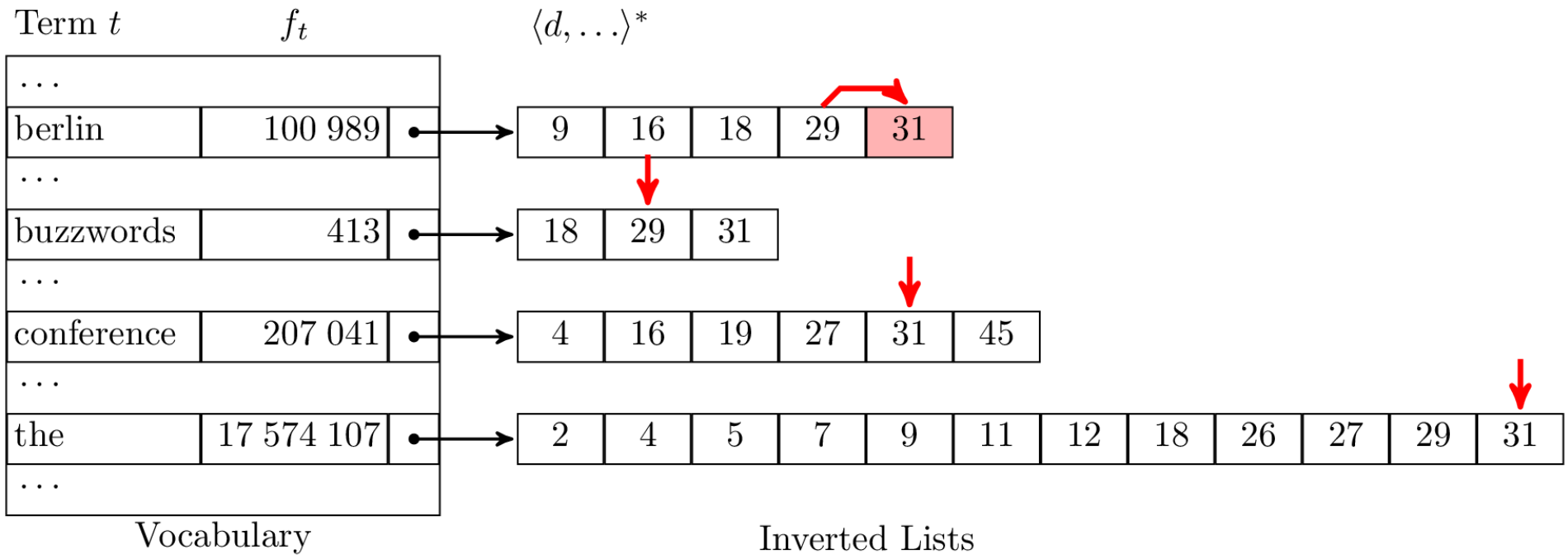
”+The +Berlin +Buzzwords +Conference”



→ *advance(31)*

Conjunctions (AND)

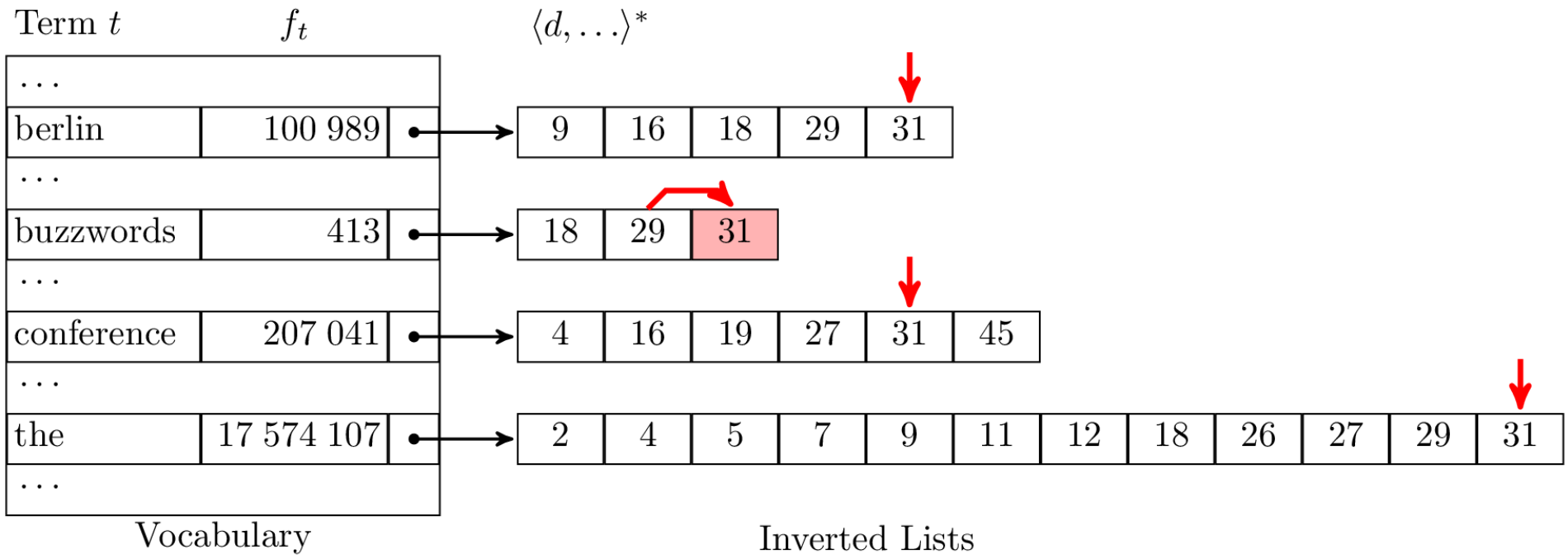
”+The +Berlin +Buzzwords +Conference”



→ *advance(31)*

Conjunctions (AND)

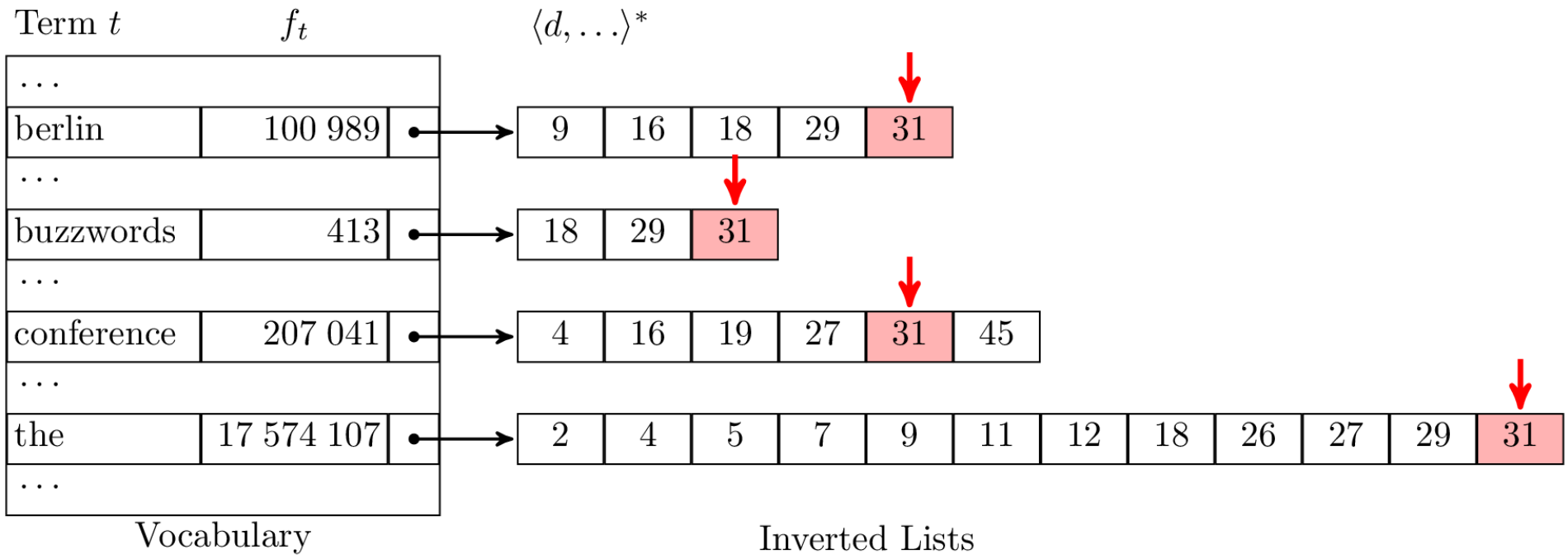
”+The +Berlin +Buzzwords +Conference”



→ *advance(31)*

Conjunctions (AND)

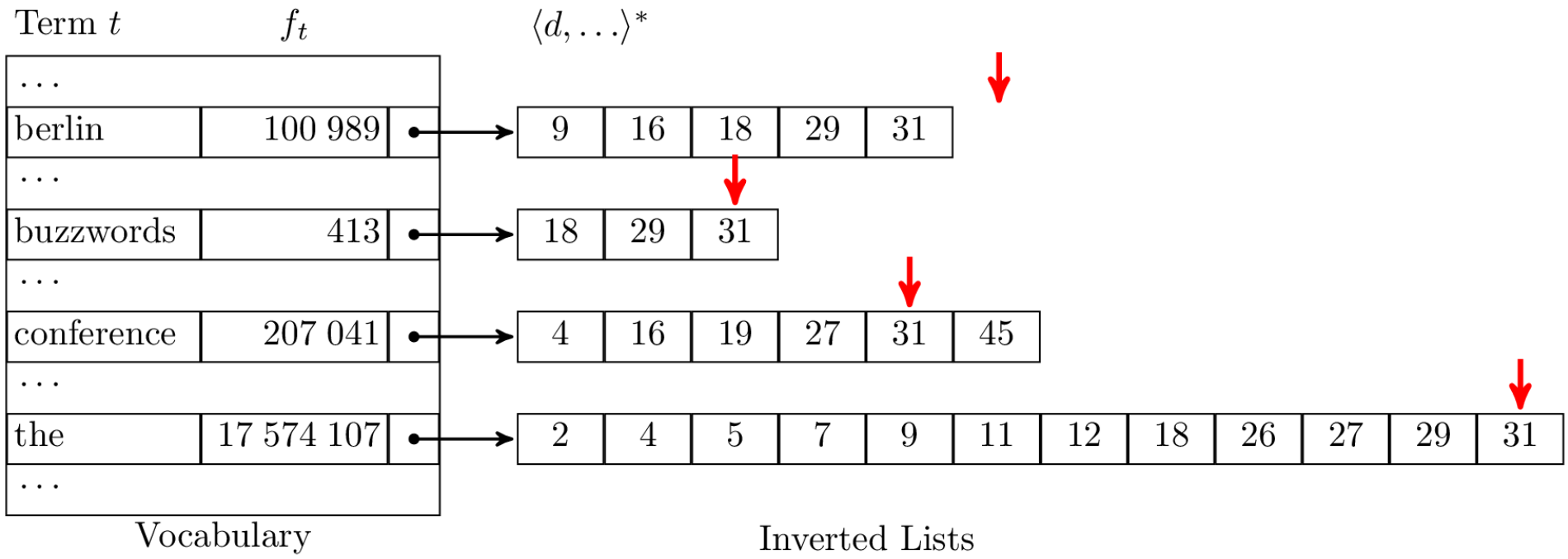
”+The +Berlin +Buzzwords +Conference”



Result Set: {31,

Conjunctions (AND)

”+The +Berlin +Buzzwords +Conference”



Result Set: {31}

Conjunctions (AND)

”+The +Berlin +Buzzwords +Conference”

- Few matches,
  only a few candidates to score

- **Wikipedia 25M:**

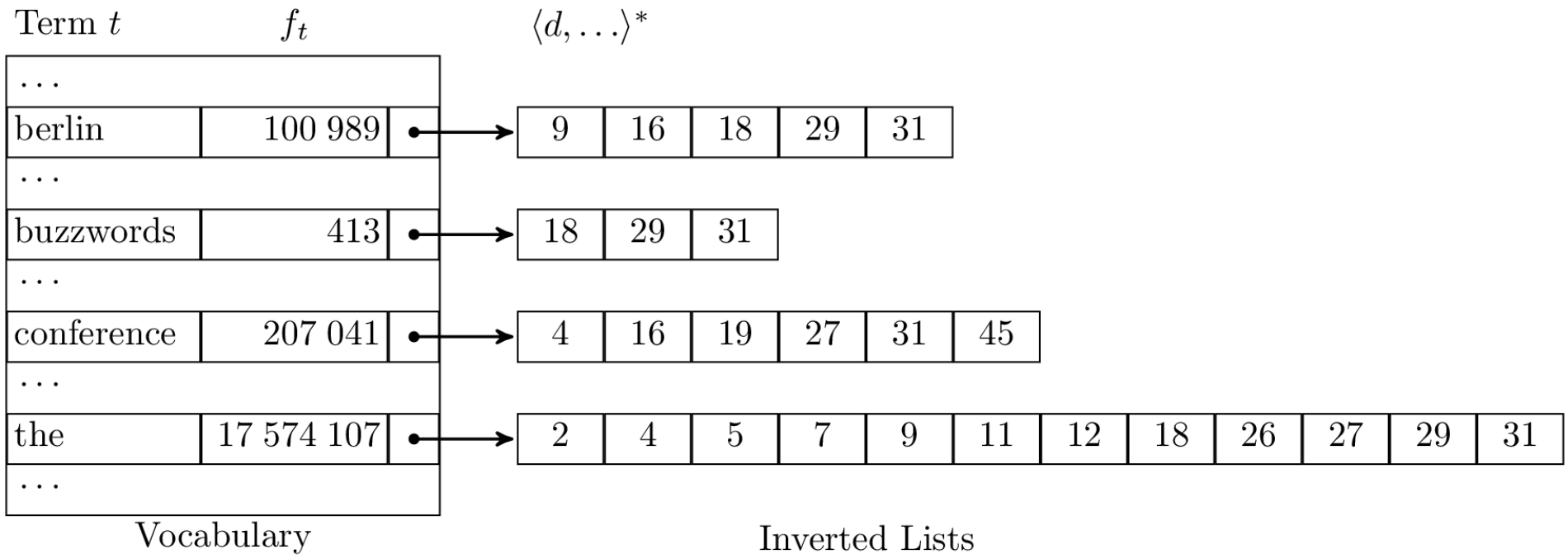
  10 ms

  → Very efficient due to *skipping*, but

  0 results → No partial match !

Disjunctions (OR)

”The Berlin Buzzwords Conference”

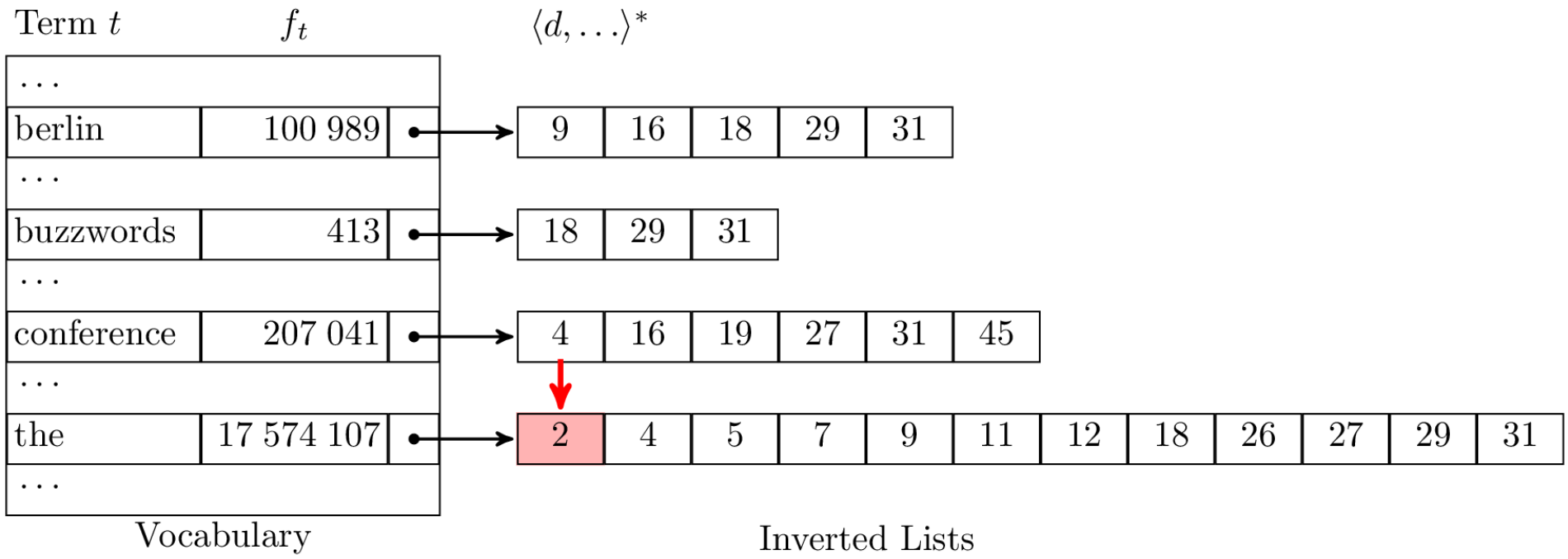


→ k-way merge (using min-heap over terms)*

** see o.a.l.search.BooleanScorer2*

Disjunctions (OR)

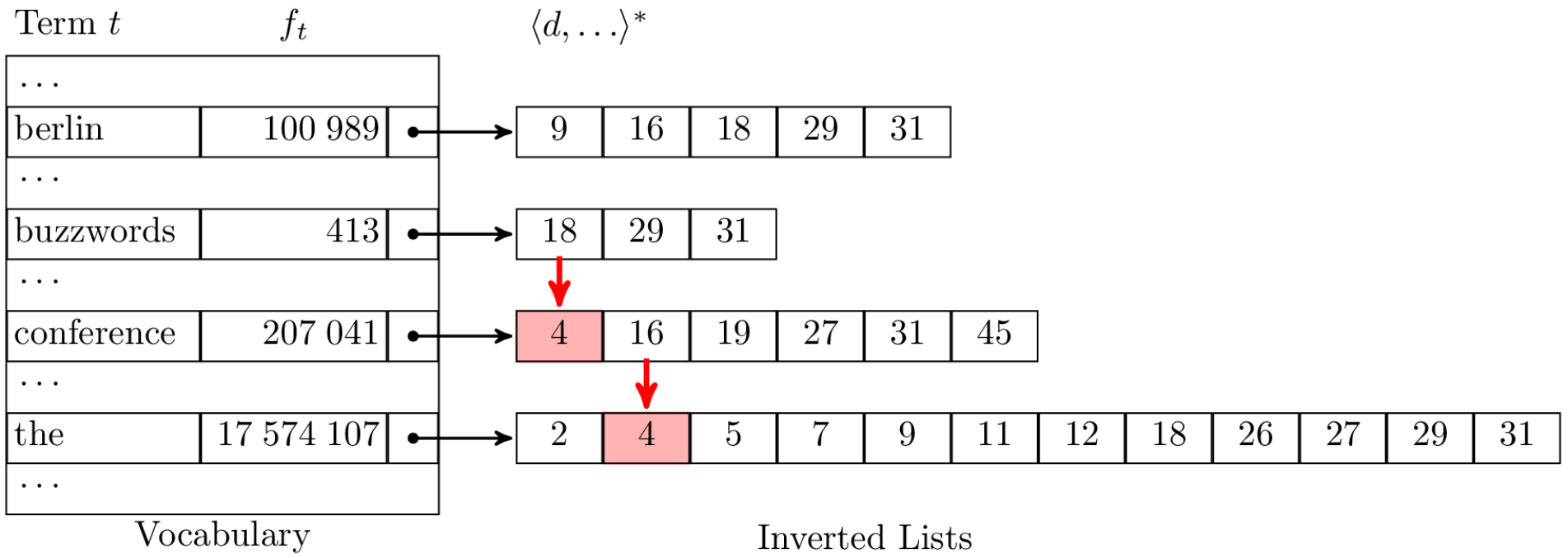
”The Berlin Buzzwords Conference”



→ *next()*

Disjunctions (OR)

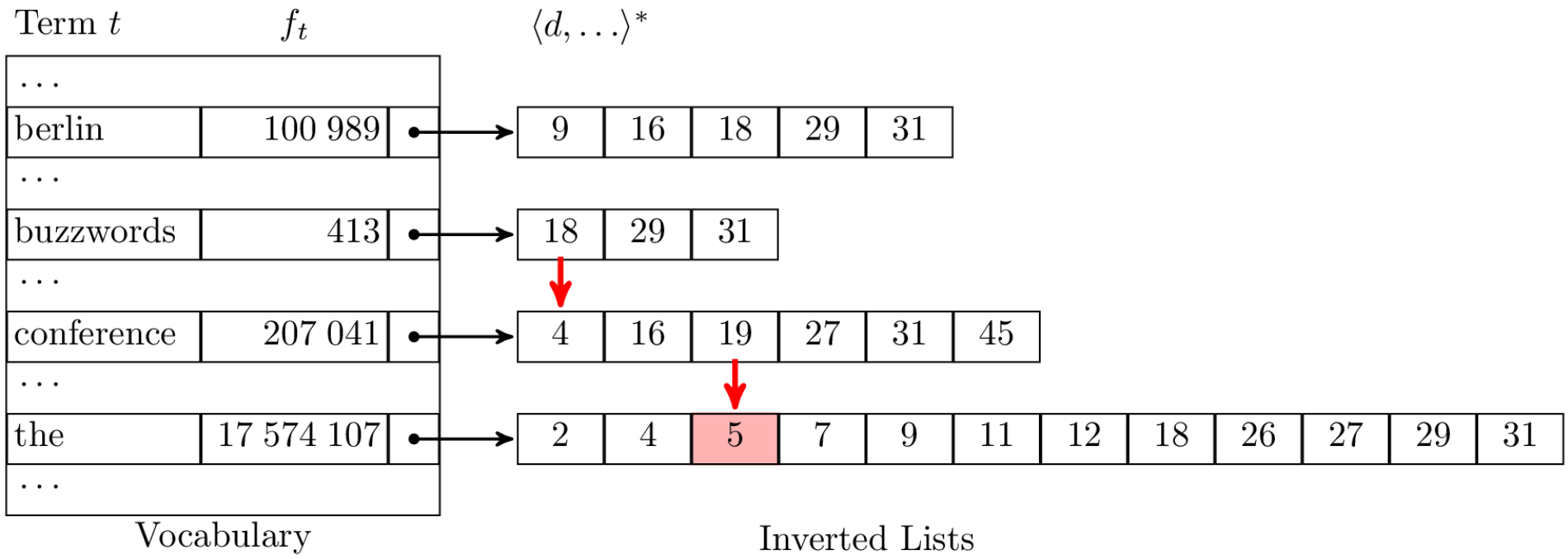
”The Berlin Buzzwords Conference”



→ *next()*

Disjunctions (OR)

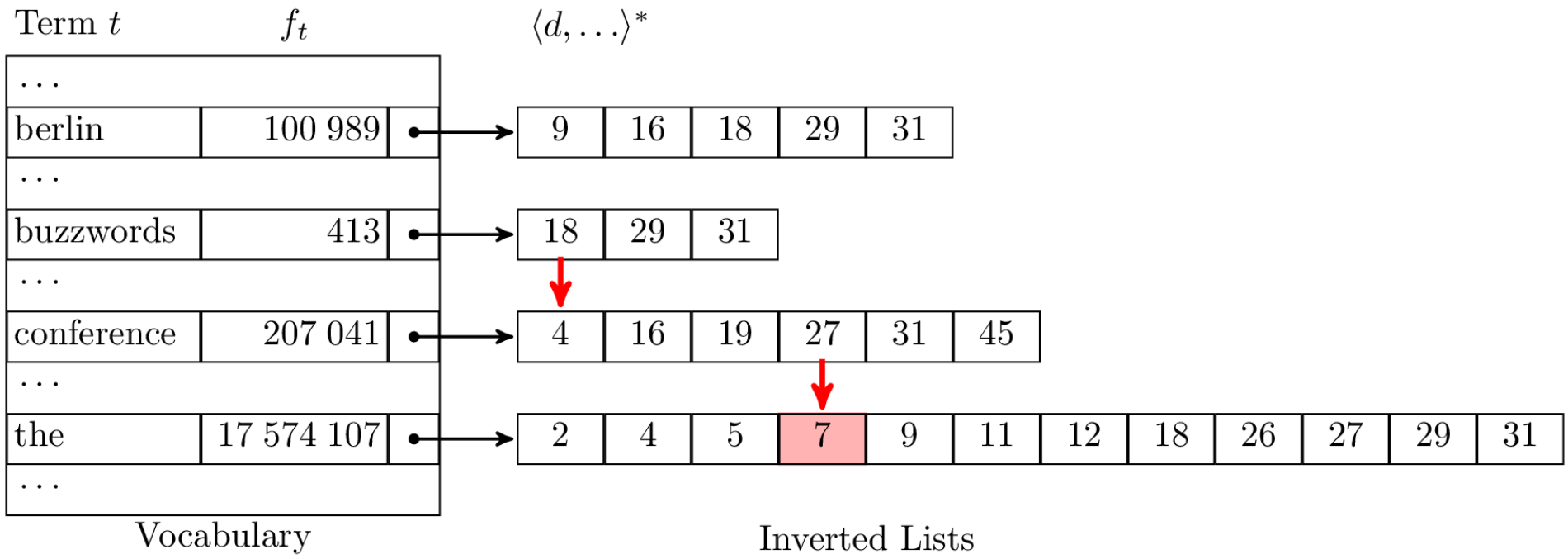
”The Berlin Buzzwords Conference”



→ *next()*

Disjunctions (OR)

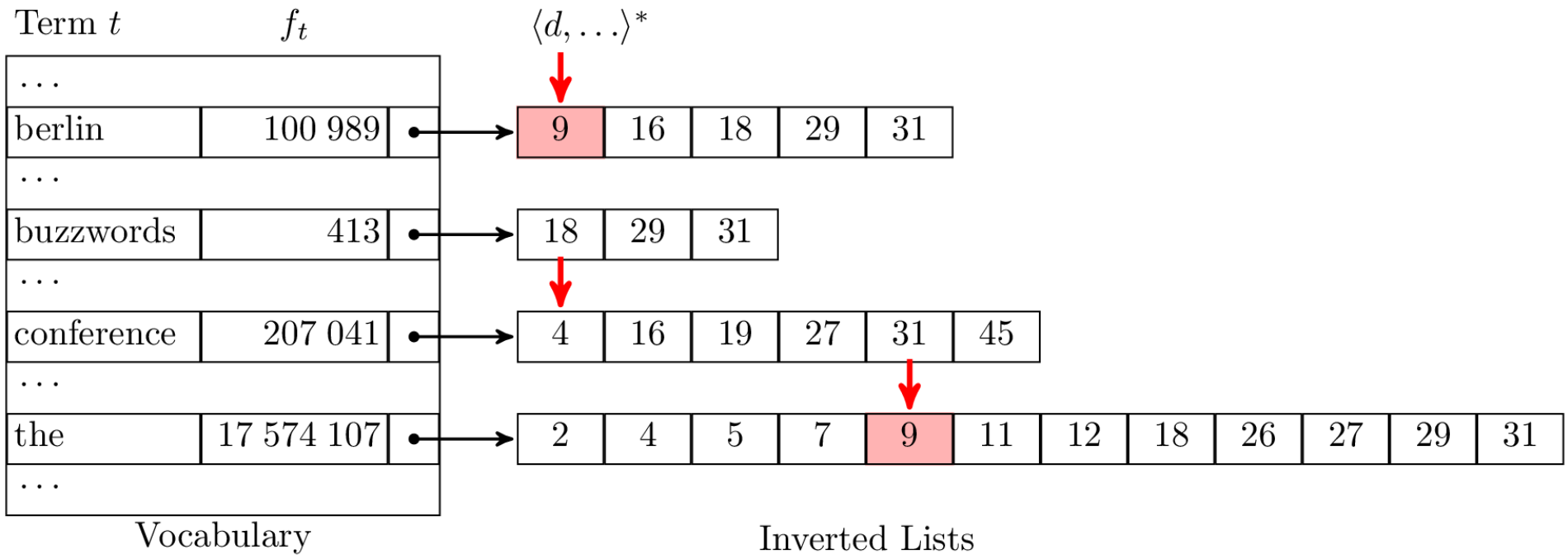
”The Berlin Buzzwords Conference”



→ *next()*

Disjunctions (OR)

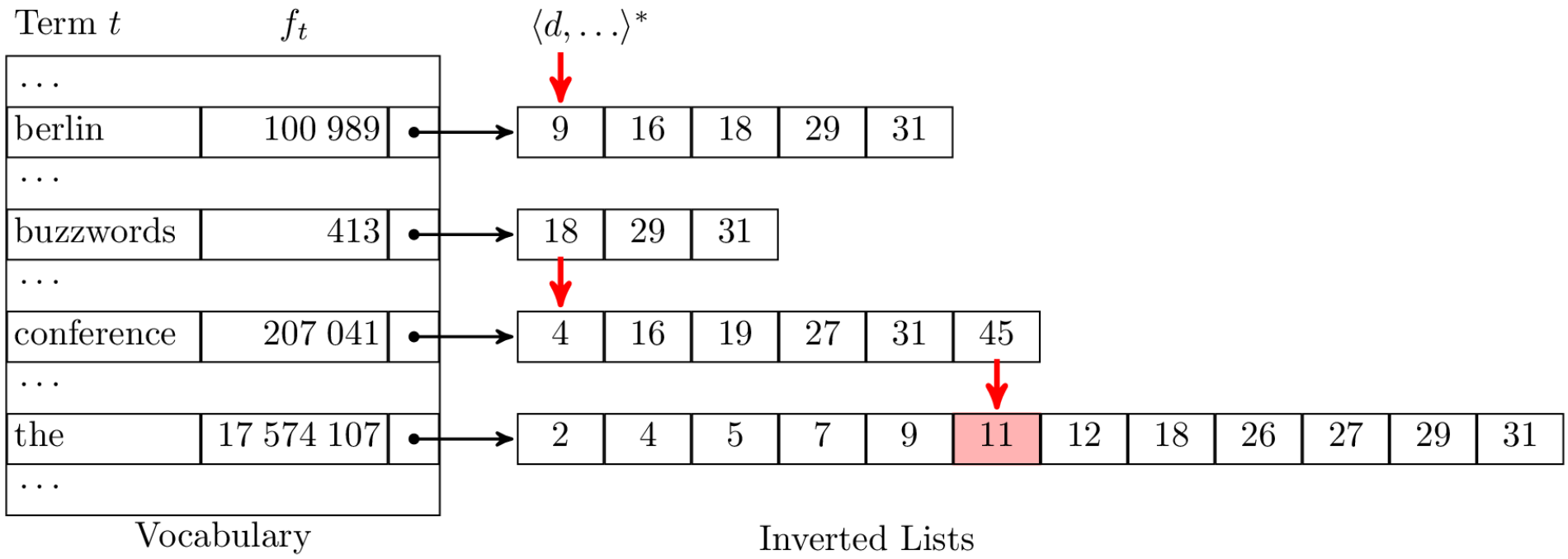
”The Berlin Buzzwords Conference”



→ *next()*

Disjunctions (OR)

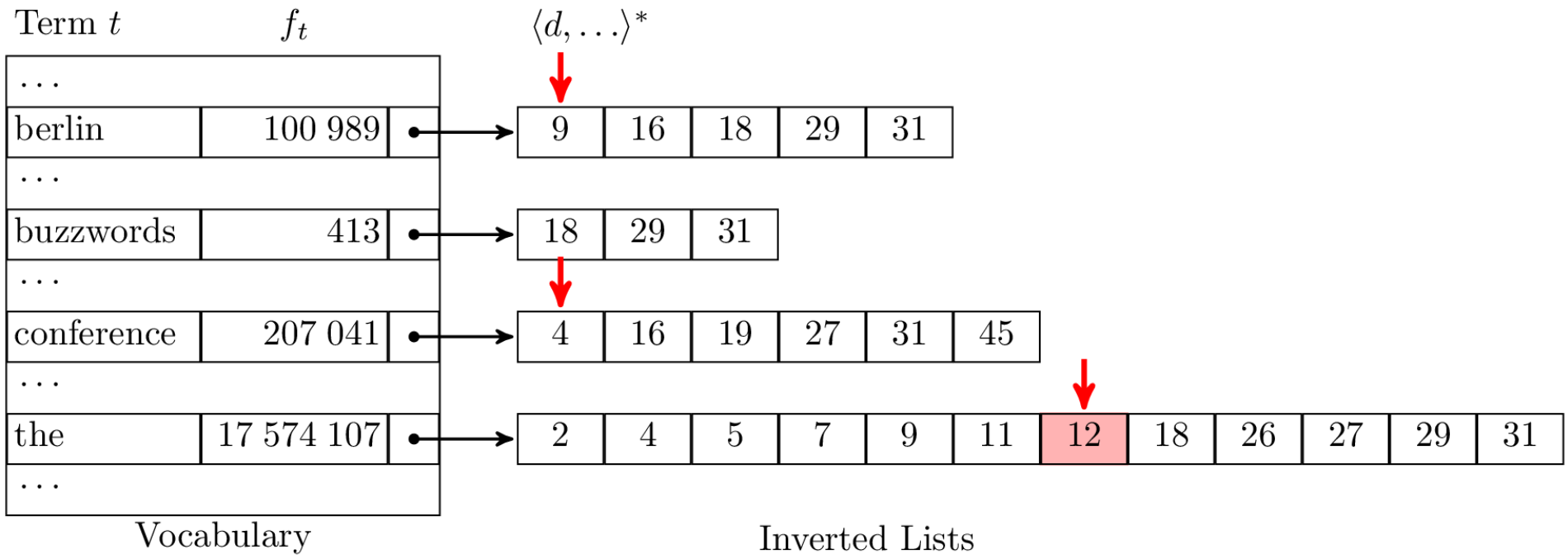
”The Berlin Buzzwords Conference”



→ *next()*

Disjunctions (OR)

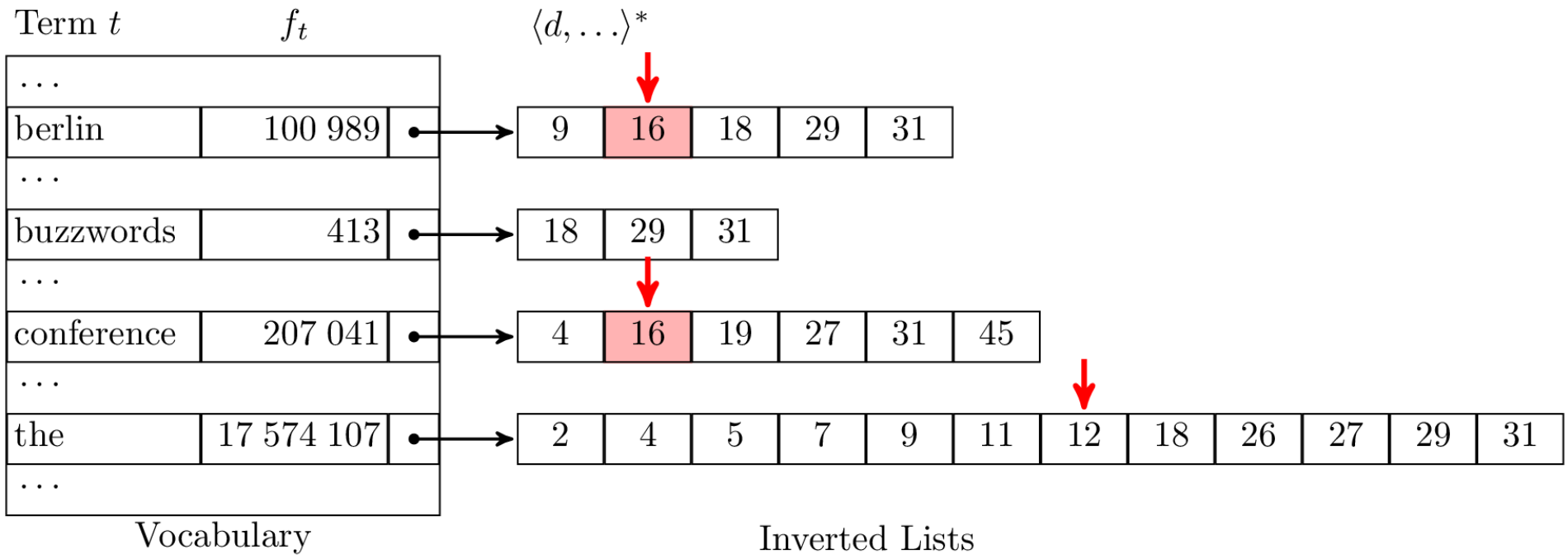
”The Berlin Buzzwords Conference”



→ *next()*

Disjunctions (OR)

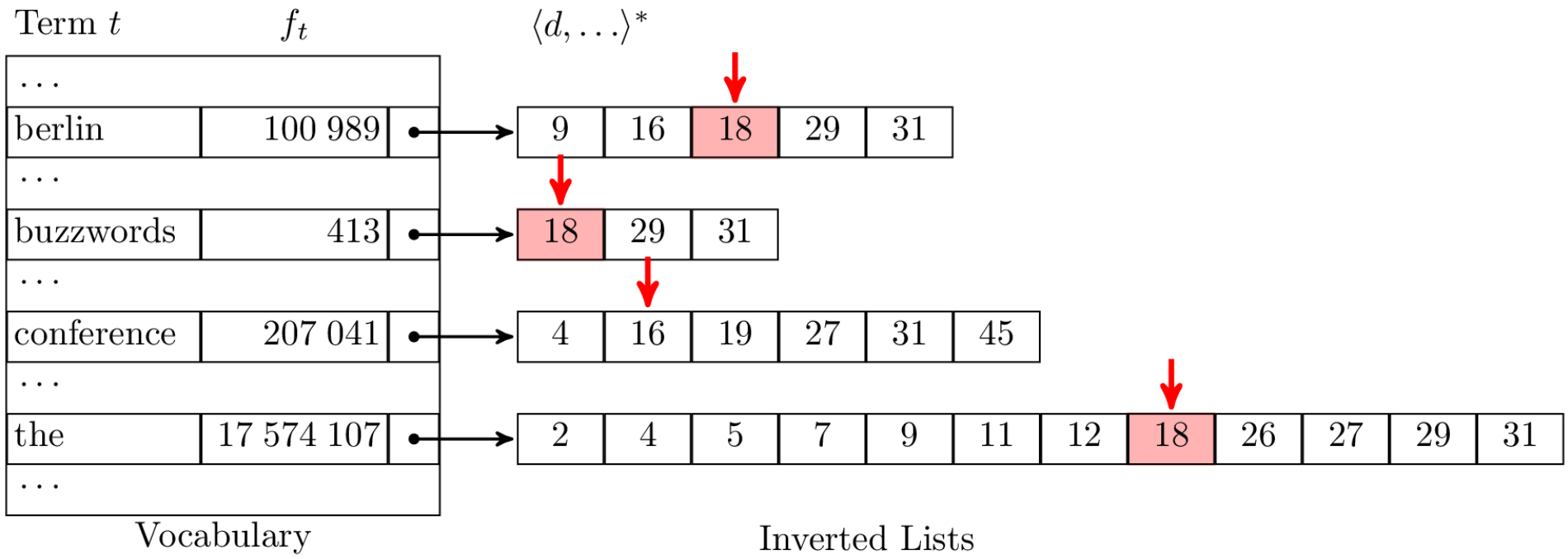
”The Berlin Buzzwords Conference”



→ *next()*

Disjunctions (OR)

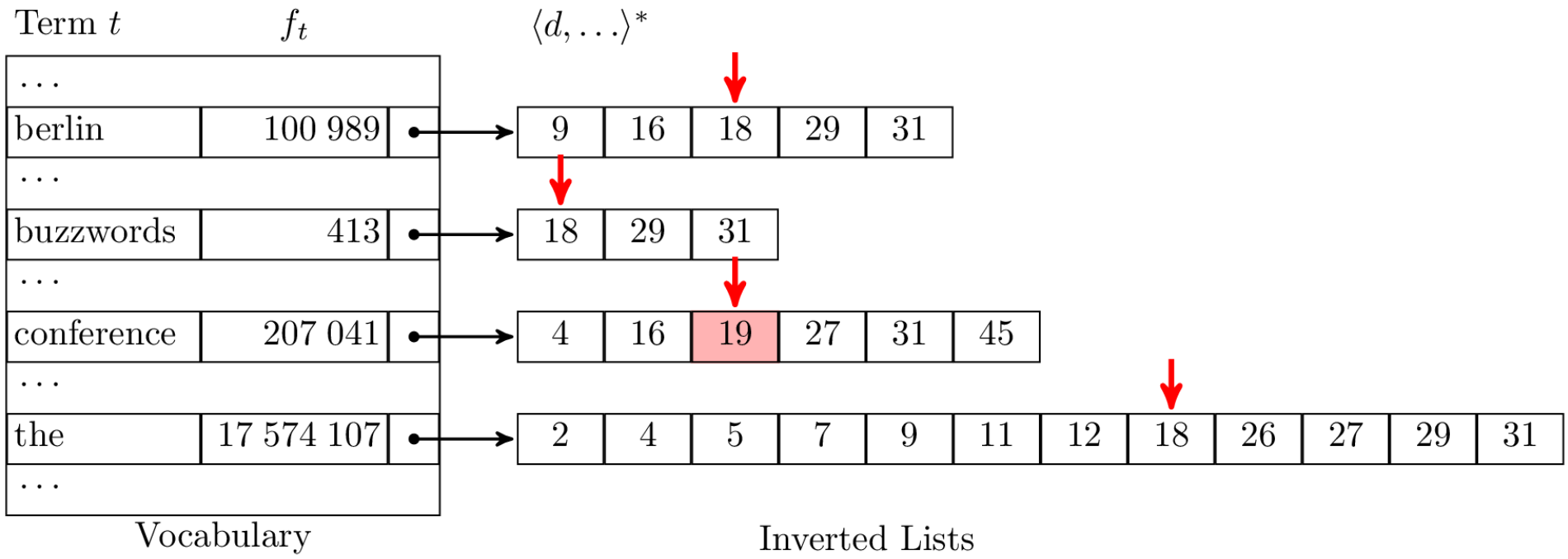
”The Berlin Buzzwords Conference”



→ *next()*

Disjunctions (OR)

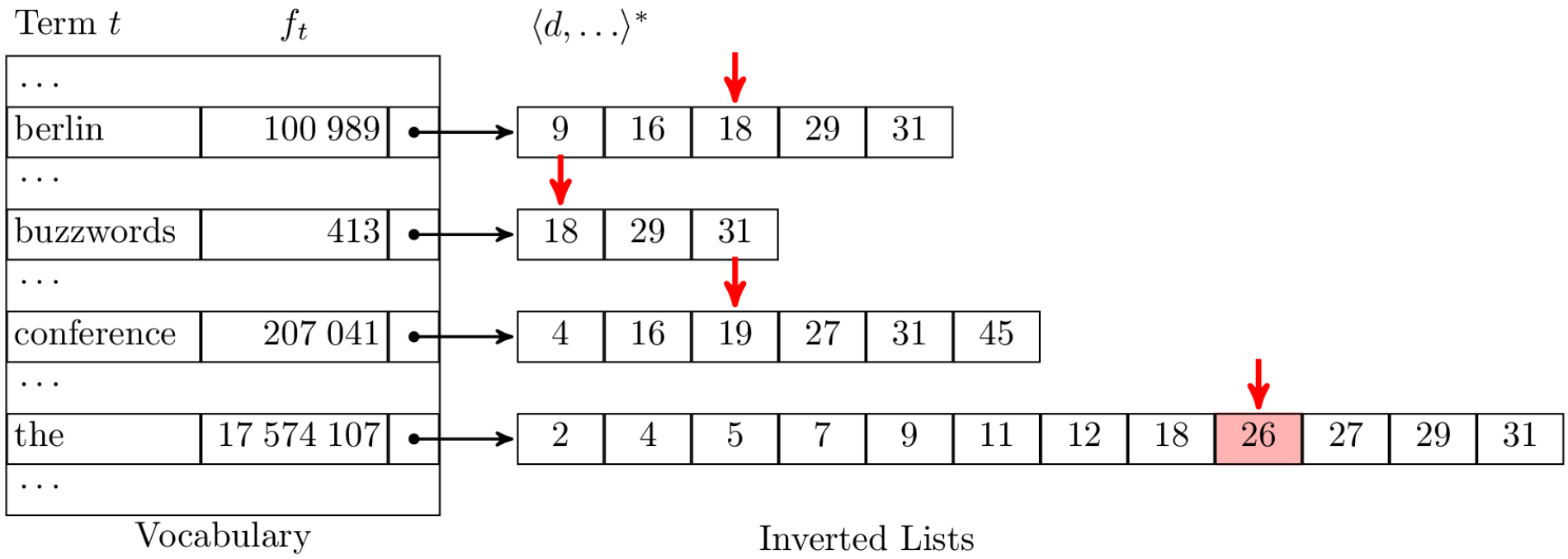
”The Berlin Buzzwords Conference”



→ *next()*

Disjunctions (OR)

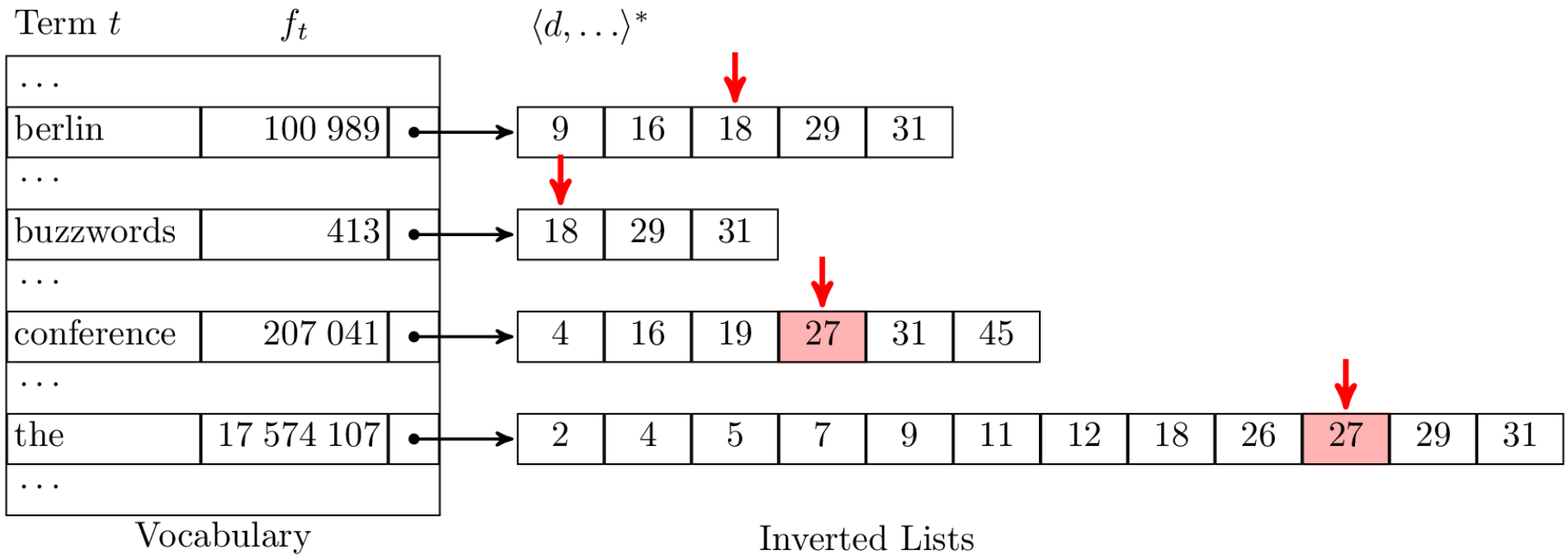
”The Berlin Buzzwords Conference”



→ *next()*

Disjunctions (OR)

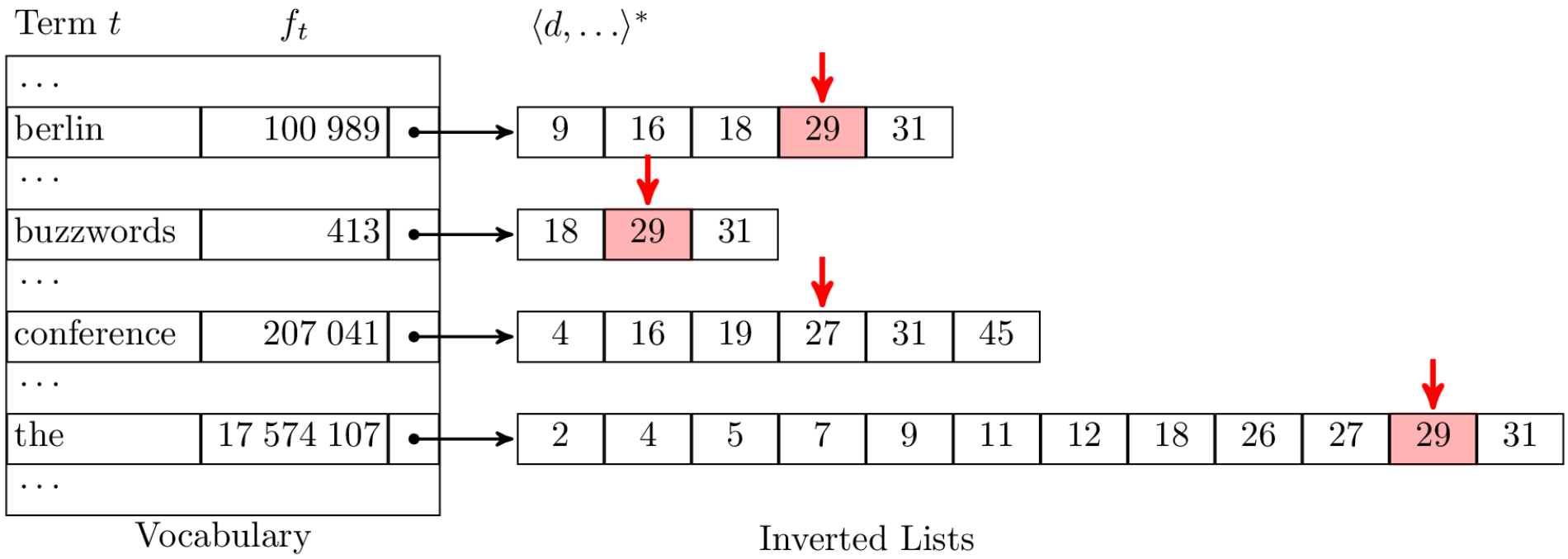
”The Berlin Buzzwords Conference”



→ *next()*

Disjunctions (OR)

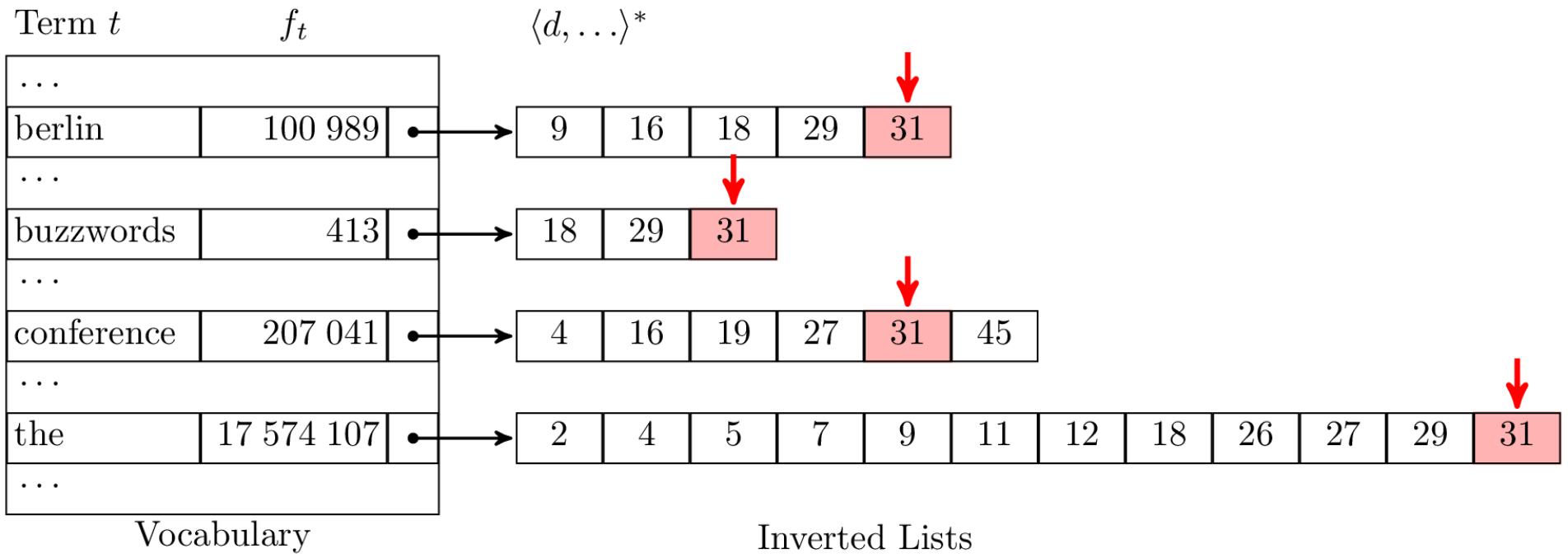
”The Berlin Buzzwords Conference”



→ *next()*

Disjunctions (OR)

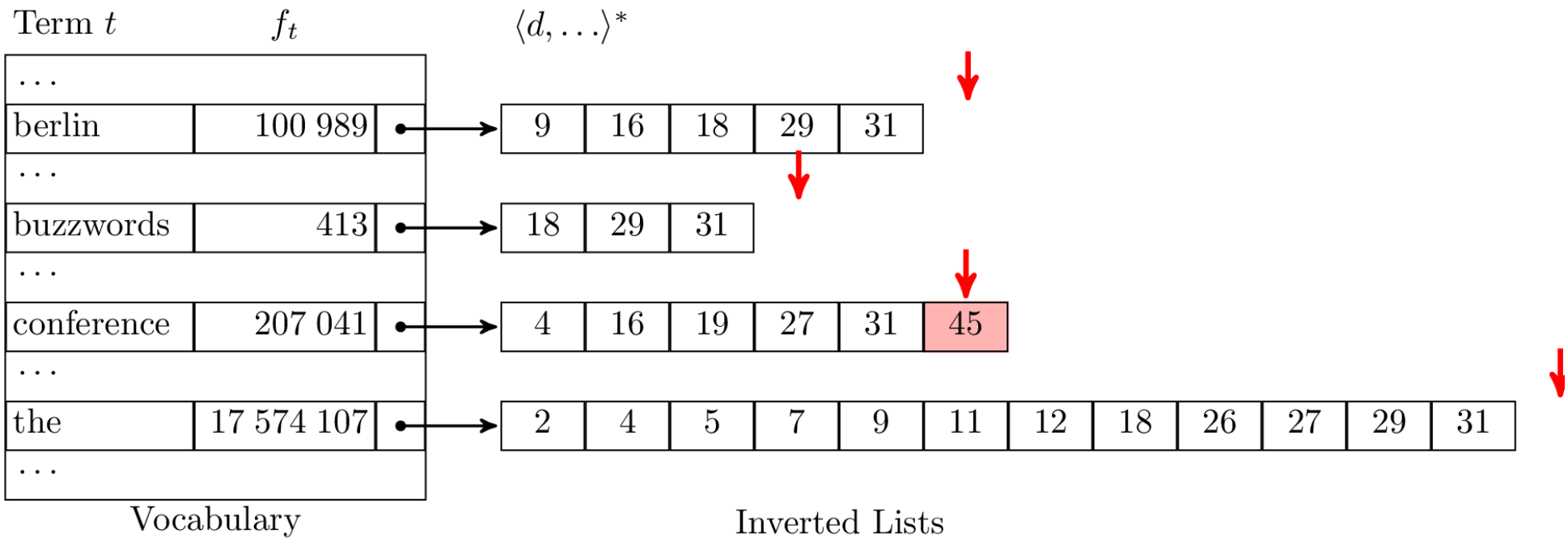
”The Berlin Buzzwords Conference”



→ *next()*

Disjunctions (OR)

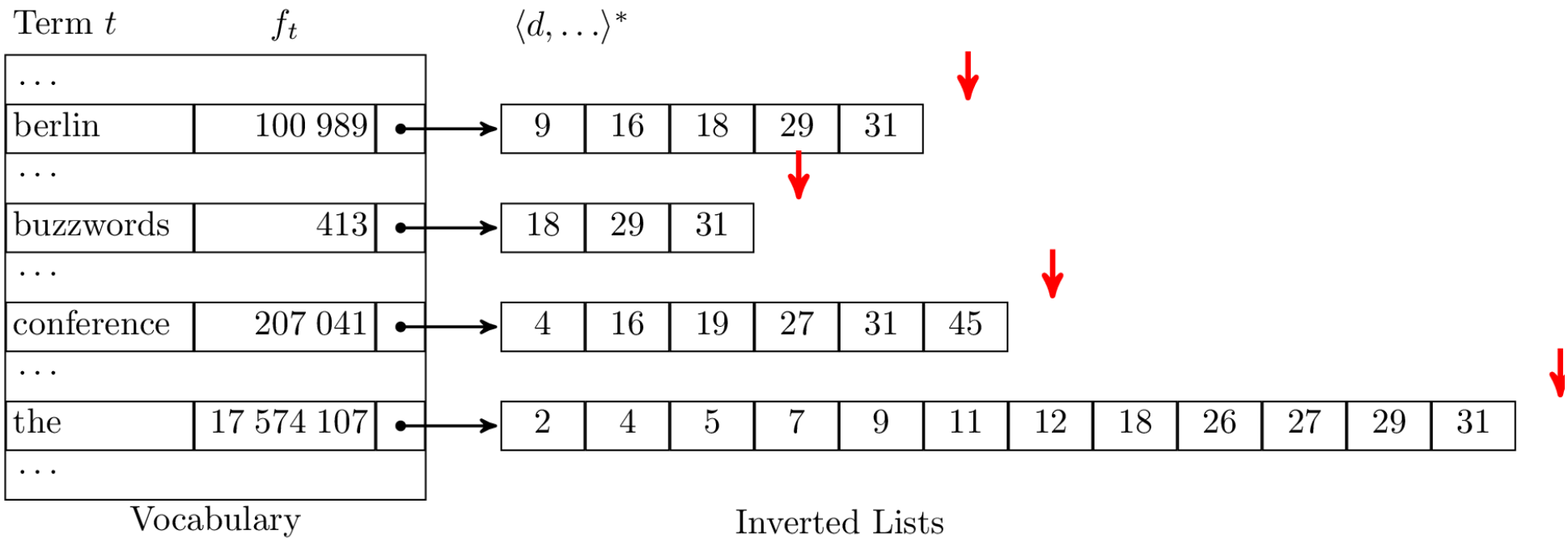
”The Berlin Buzzwords Conference”



→ *next()*

Disjunctions (OR)

”The Berlin Buzzwords Conference”



- *No skipping*; all postings decompressed, merged & scores computed

Disjunctions (OR)

”The Berlin Buzzwords Conference”

- **Wikipedia 25M:**

  750 ms, 17,628,190 totalHits (vs. 10 queried)

  → Scoring of almost **ALL** documents

Can we do better?

Optimized Scoring with Maxscore*

- **Maxscore***

  H. Turtle, J. Flood. *Query Evaluation: Strategies and Optimizations*, IPM, 31(6), **1995**.

- **Maxscore Variants**

  A. Z. Broder, D. Carmel, M. Herscovici, A. Soffer, J. Y. Zien. *Efficient Query Evaluation using a Two-Level Retrieval Process*, in Proc. of CIKM, **2003**.

  T. Strohman, H. Turtle, W. B. Croft. *Optimization Strategies for Complex Queries*, in Proc. of ACM SIGIR, **2005**.

- **Maxscore for Block-Compressed Indexes**

  K. Chakrabarti, S. Chaudhuri, V. Ganti. *Interval-Based Pruning for Top-k Processing over Compressed Lists*, in Proc. of ICDE, **2011**.

- **Maxscore with Structured Queries**

  S. Pohl, A. Moffat, J. Zobel. *Efficient Extended Boolean Retrieval*, IEEE TKDE, 24(6), **2012**.

Retrieval Model Scoring Functions

- Lucene's DefaultSimilarity:

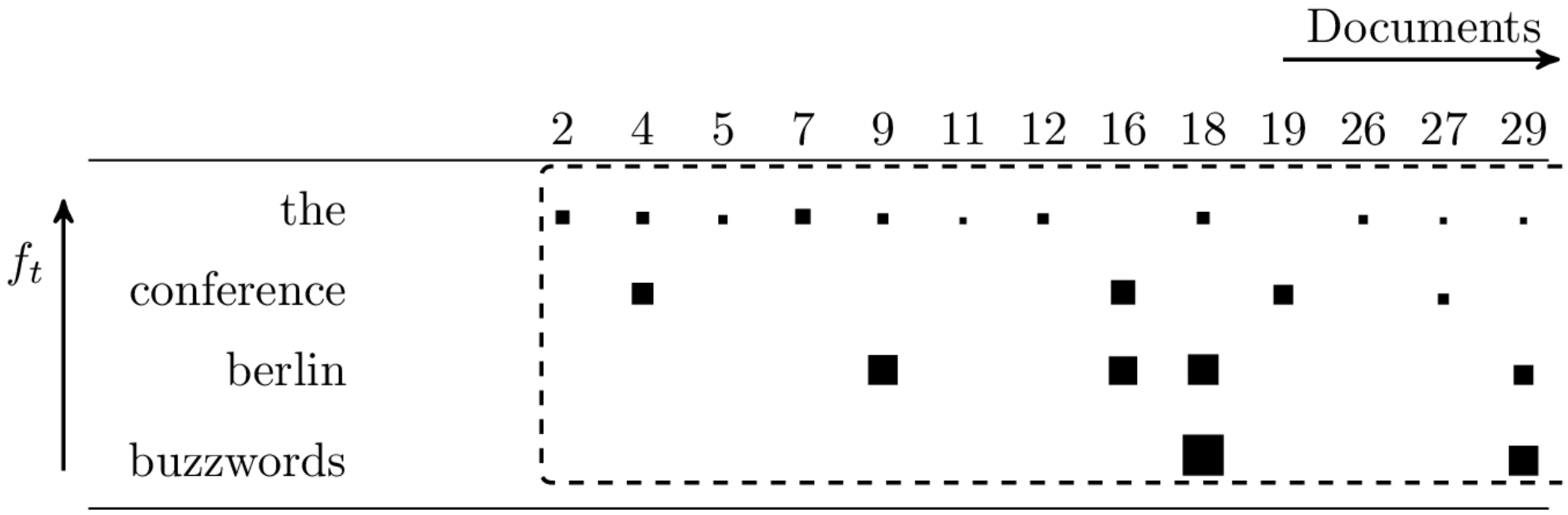
$$S(q,d) = \sum_{t \in q \cap d} \left(1 + \log \frac{N}{f_t + 1}\right)^2 \cdot \frac{\sqrt{f_{d,t}}}{\sqrt{l_d}} \cdot \text{boost}_{\text{field}(t)}$$

- BM25:

$$S(q,d) = \sum_{t \in q \cap d} \log\left(1 + \frac{N - f_t + 0.5}{f_t + 0.5}\right) \cdot \frac{(k_1 + 1) \cdot f_{d,t}}{k_1 \cdot ((1 - b) + b \cdot l_d / l_{\text{avg}}) + f_{d,t}}$$

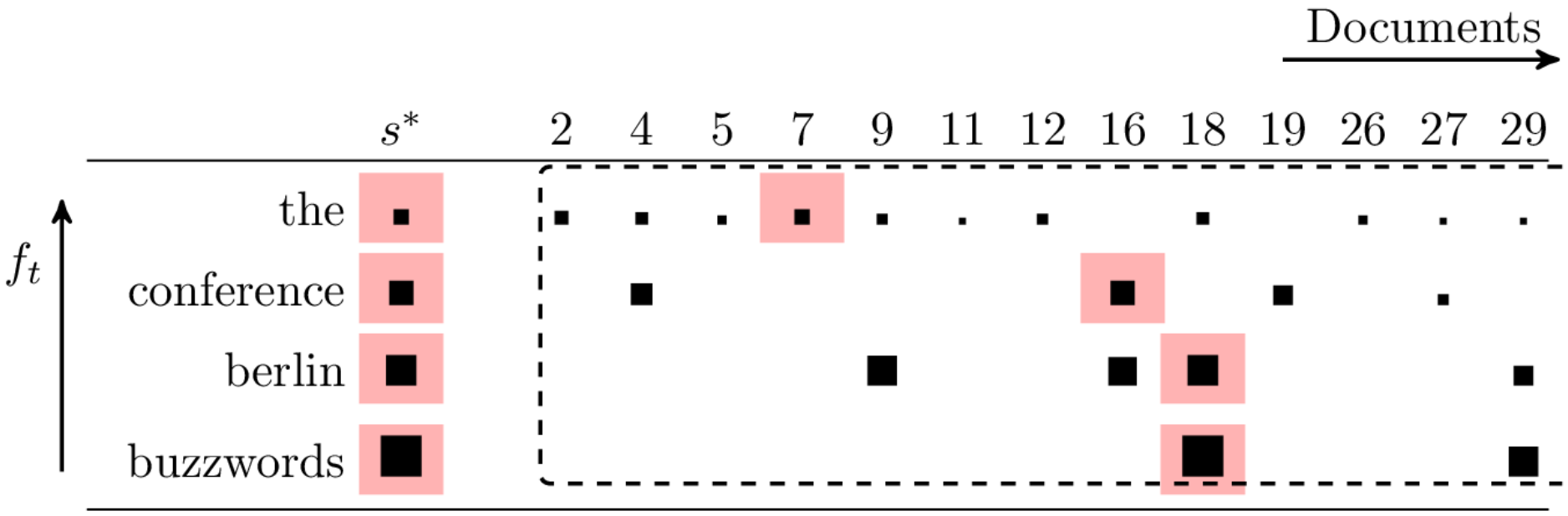
- Scoring functions of (standard) retrieval models are **SUMs** over *term score contributions*

Maxscore



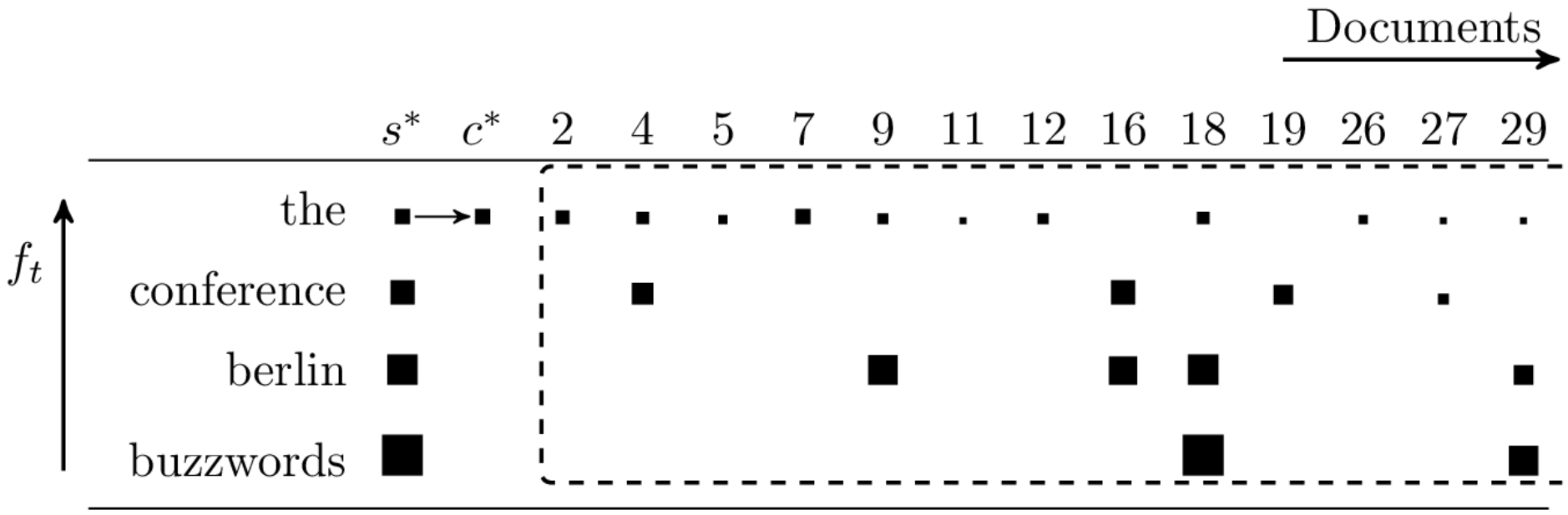
→ Order query terms by doc frequency f_t
→ Box size refers to *term score contribution*

Maxscore



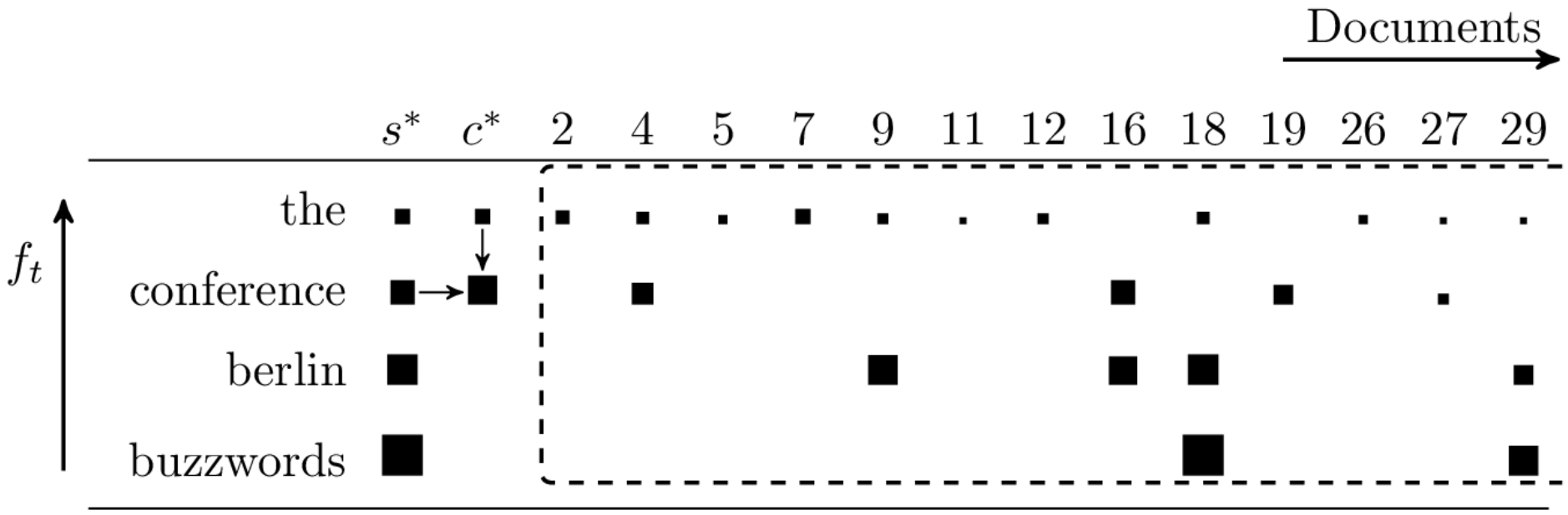
→ At indexing time,
determine *maxscore s**

Maxscore



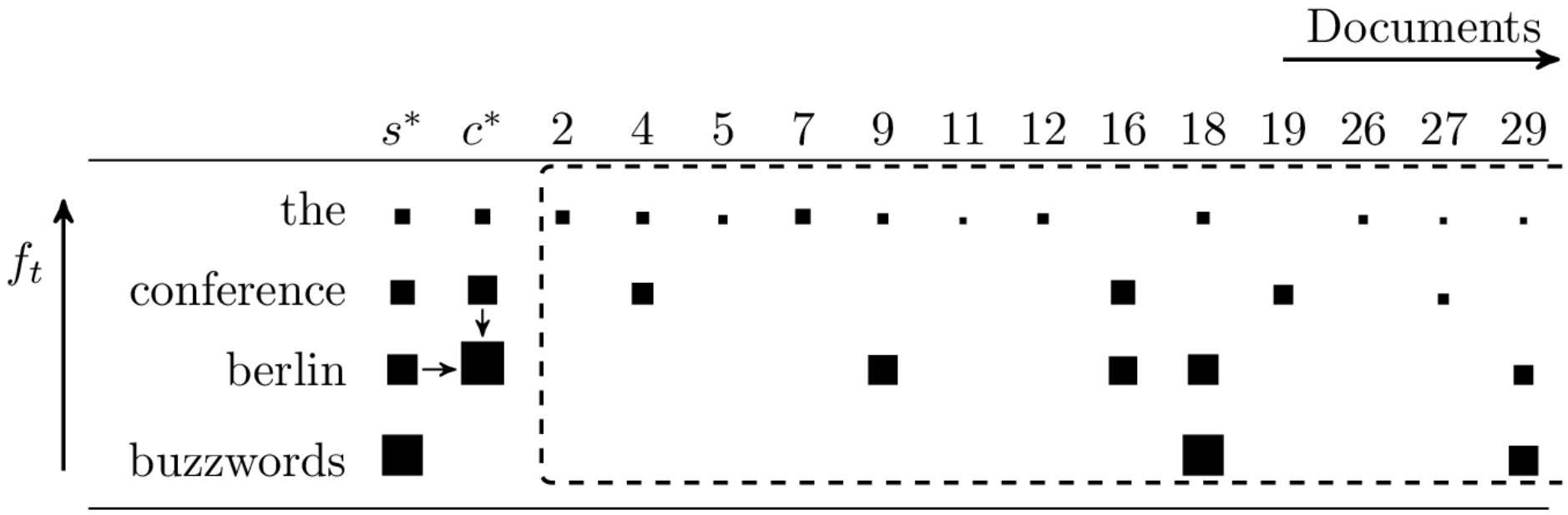
→ At search time,
compute *cumulative maxscores c**

Maxscore



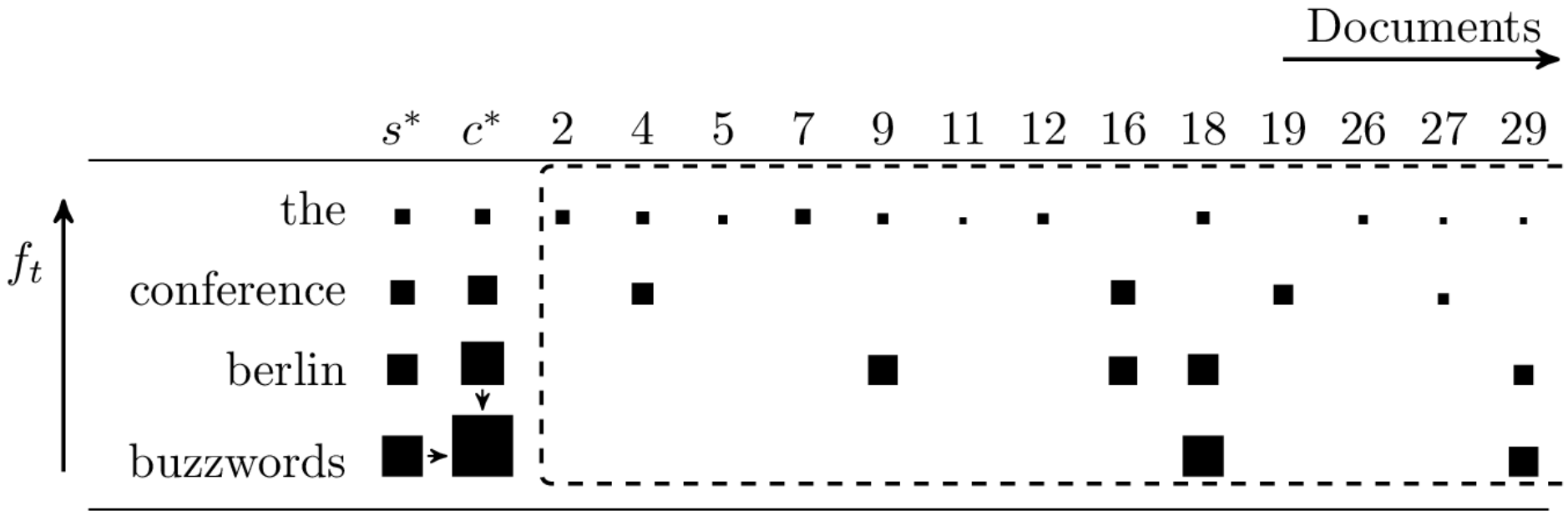
→ At search time,
compute *cumulative maxscores c**

Maxscore



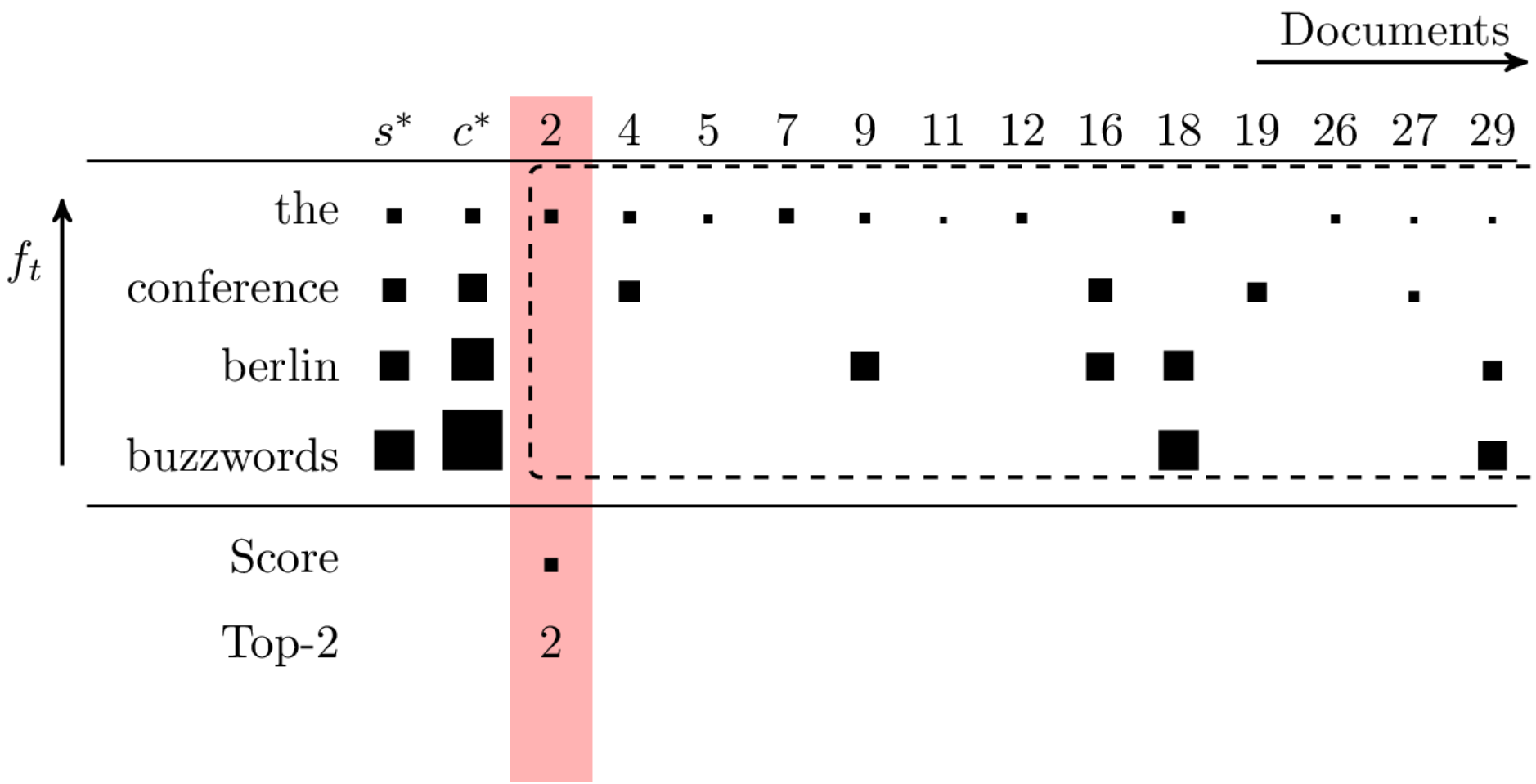
→ At search time,
compute *cumulative maxscores c**

Maxscore



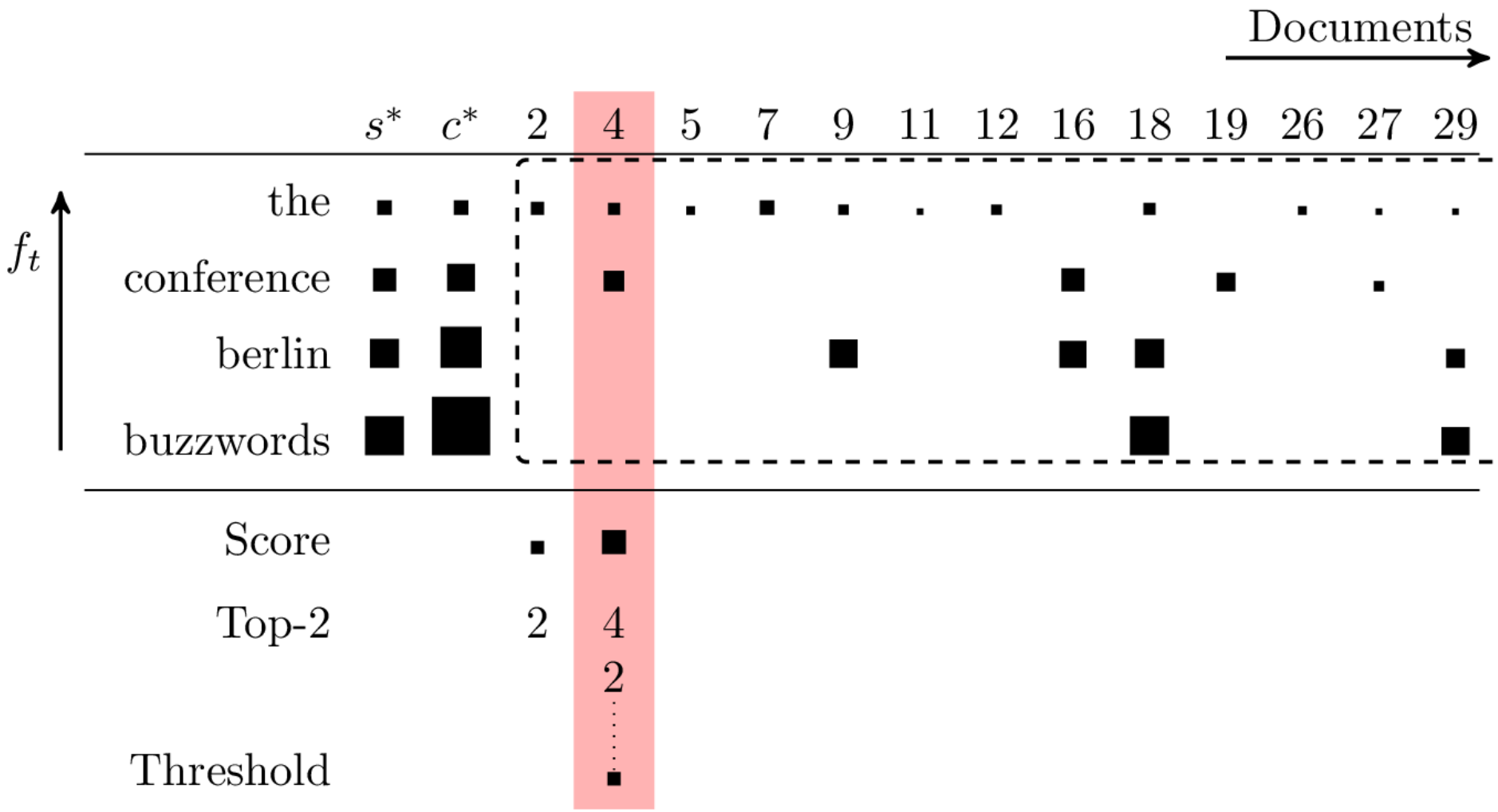
→ At search time,
compute *cumulative maxscores c**

Maxscore



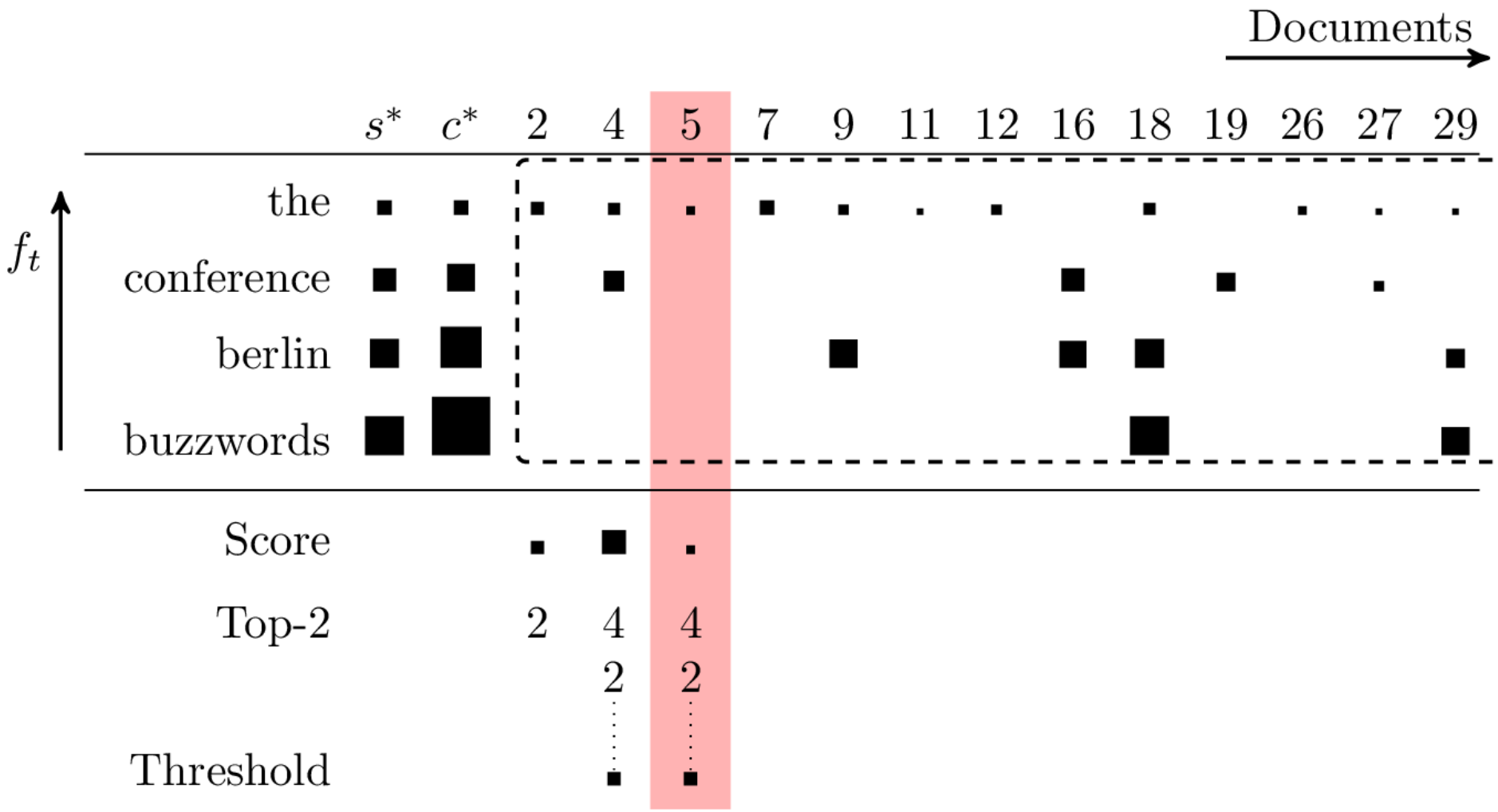
→ Score top-k

Maxscore

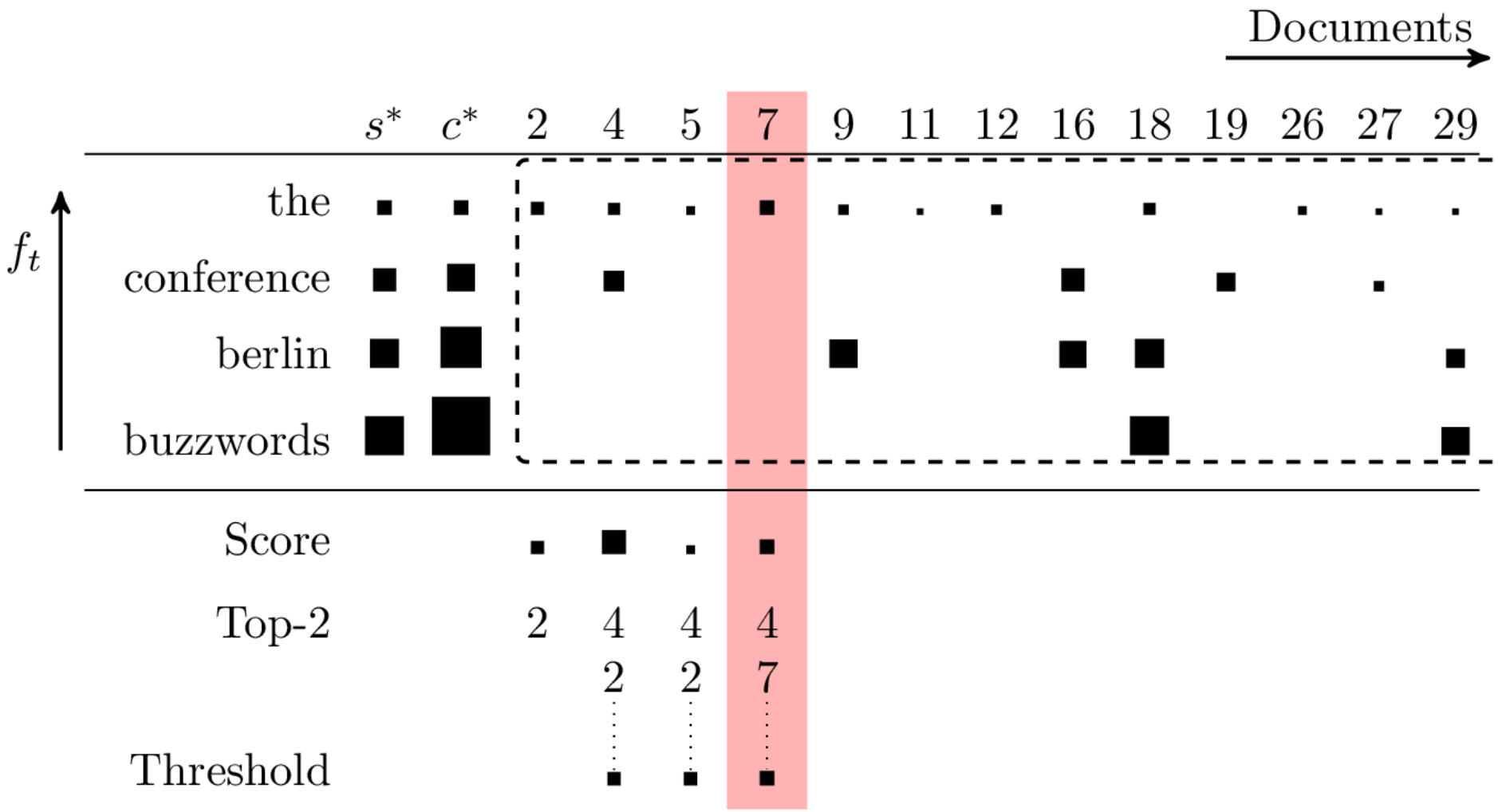


→ Score top-k, track lowest score as *threshold*

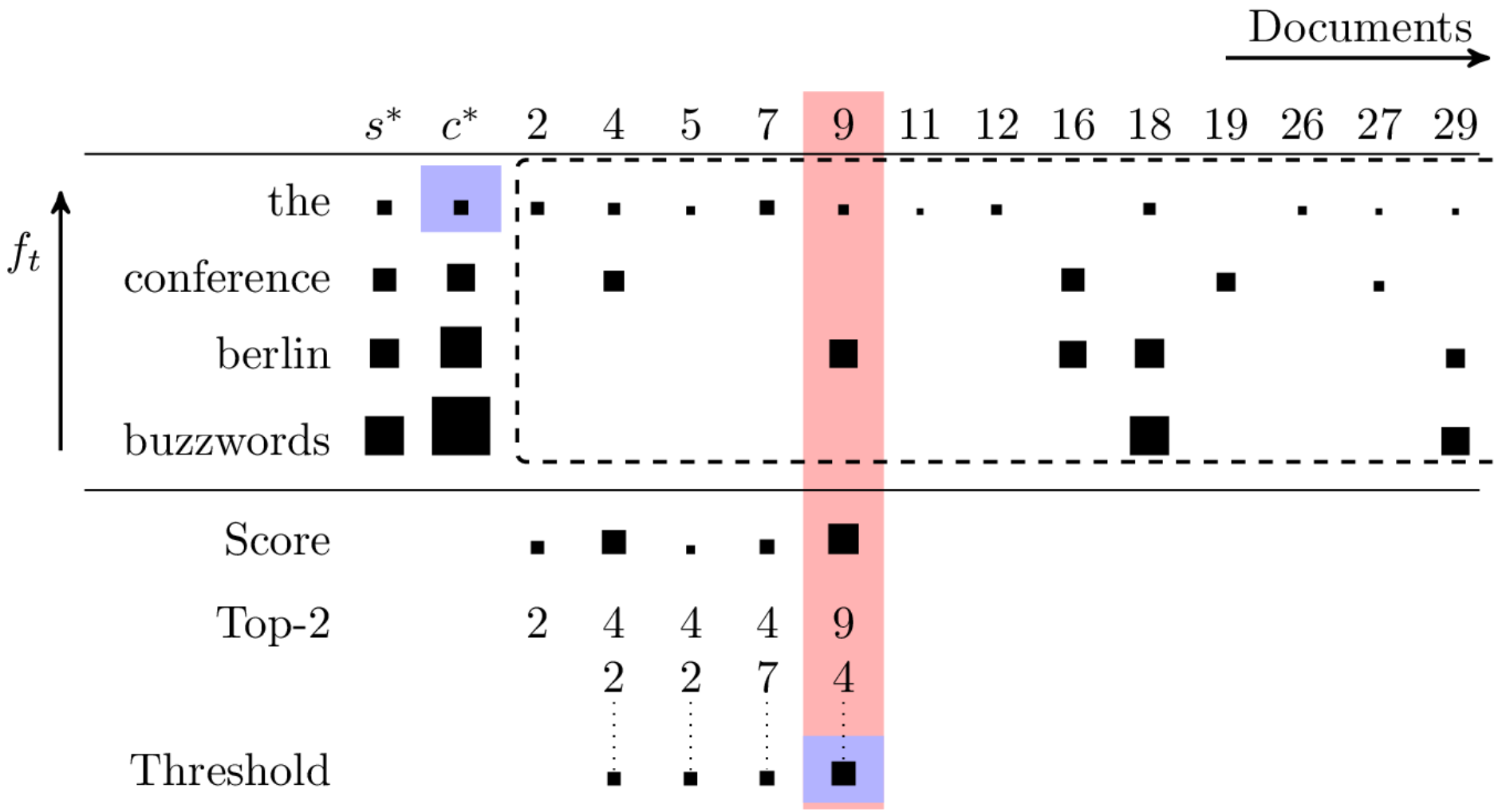
Maxscore


Documents
s^* c^* 2 4 5 7 9 11 12 16 18 19 26 27 29
f_t
the
conference
berlin
buzzwords
Score
Top-2
2 4 4
2 2
Threshold

Maxscore

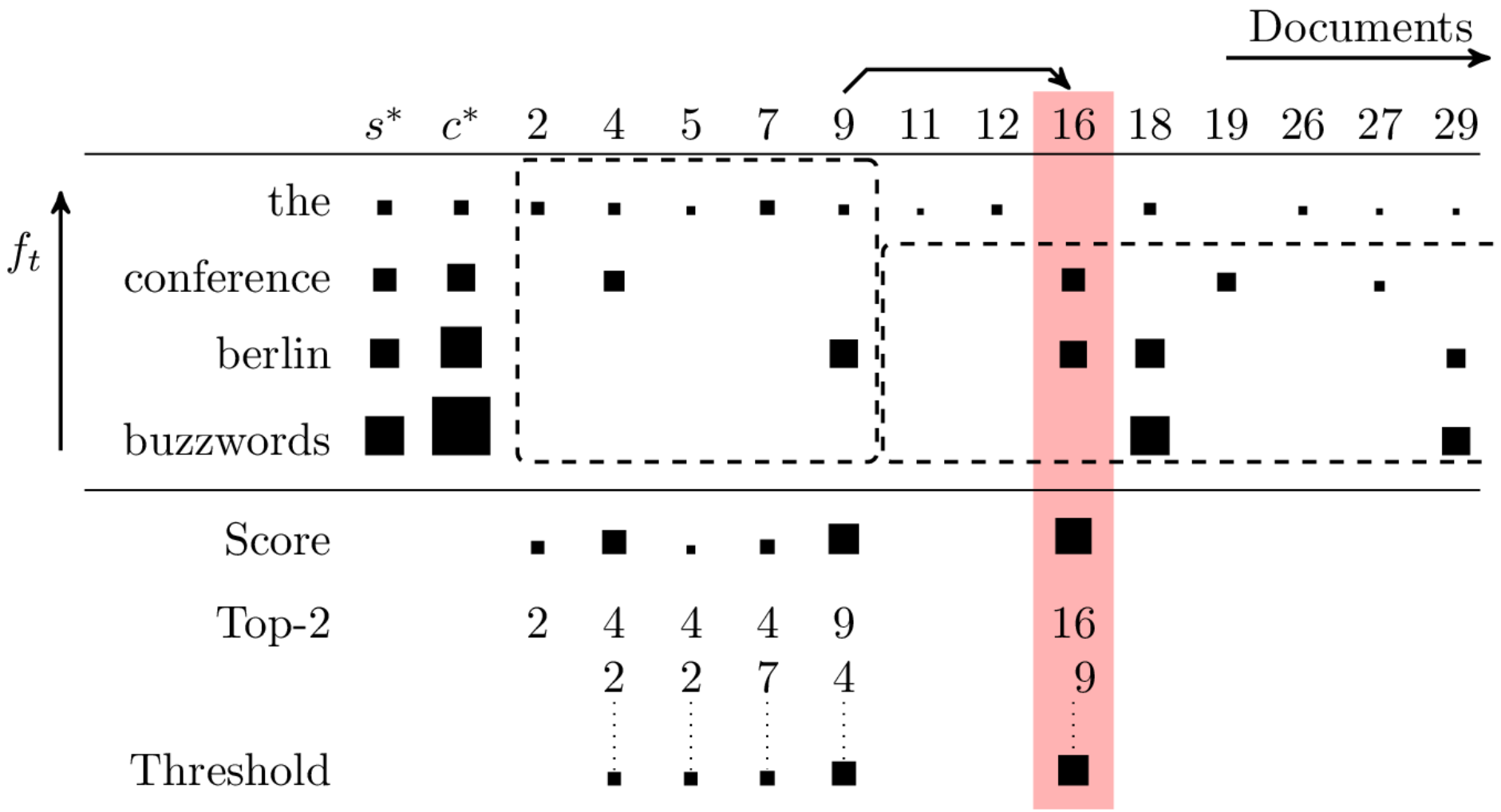

Documents
s^* c^* 2 4 5 7 9 11 12 16 18 19 26 27 29
f_t
the
conference
berlin
buzzwords
Score
Top-2
2 4 4 4
2 2 7
Threshold

Maxscore



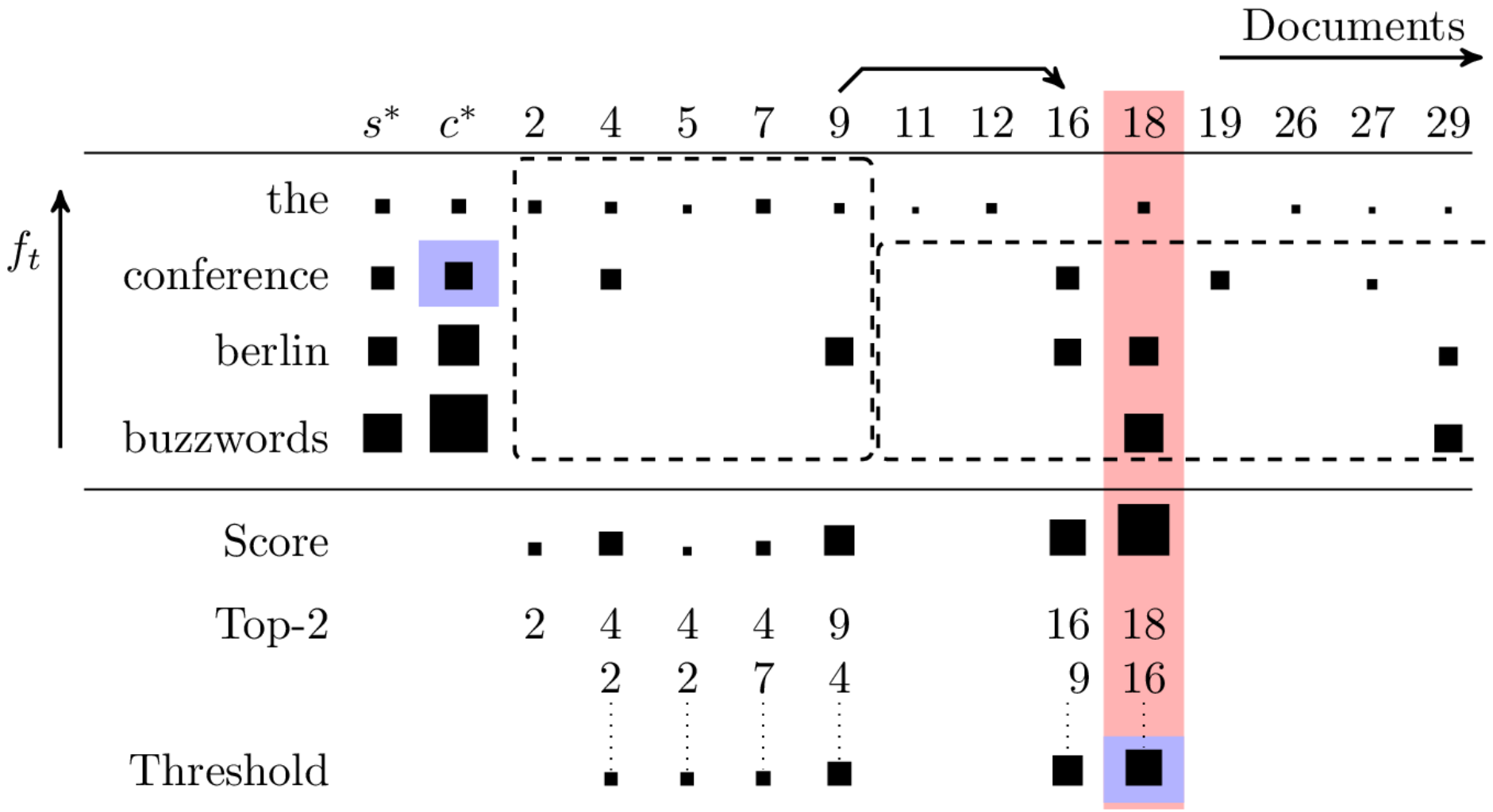
→ Threshold exceeds *c**

Maxscore



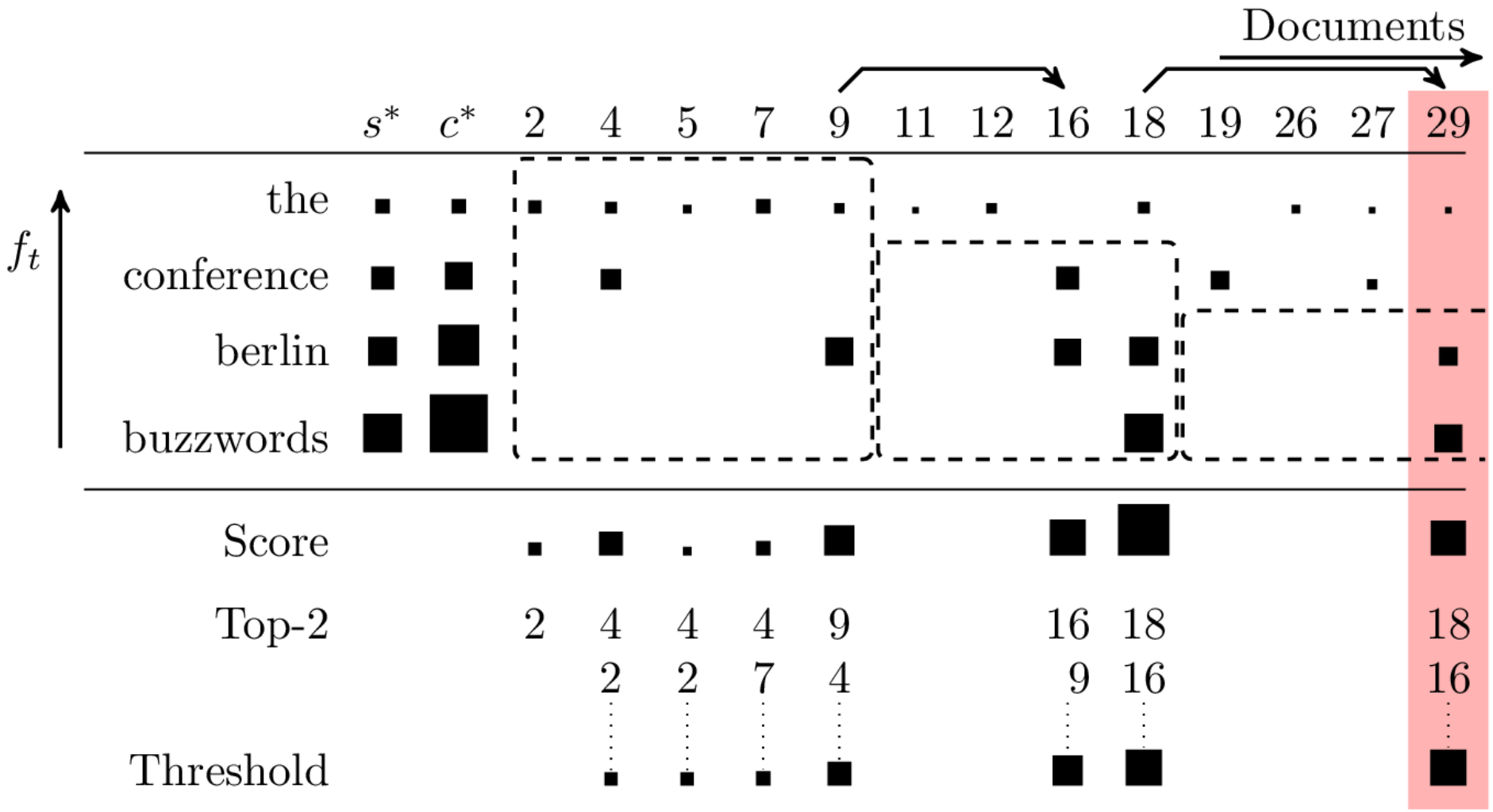
→ Merge m-1 terms, advance(16)

Maxscore



→ Threshold exceeds next *c**

Maxscore



→ Merge m-2 terms, advance(29)

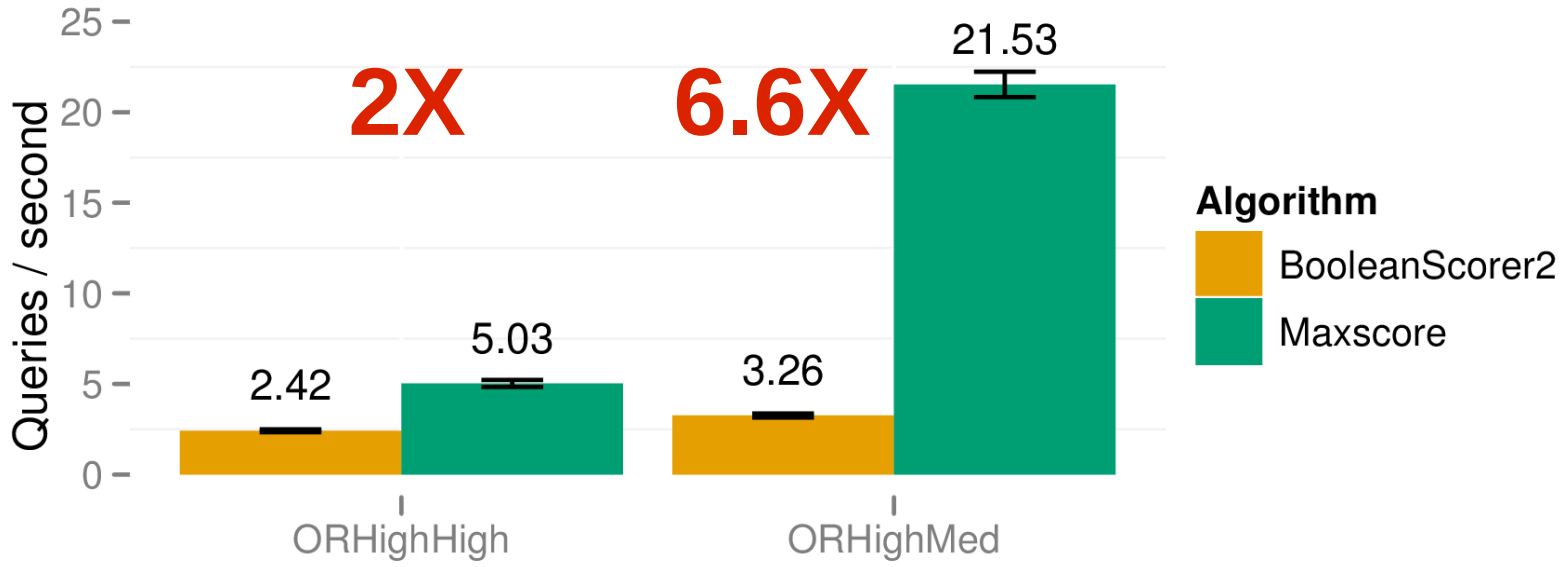
Maxscore – Experiments

- ”The Berlin Buzzwords Conference”:

System	Scored Docs	Time [ms]
Lucene40	17 628 190	**750** ±11
Lucene40 w/ Maxscore	298 800	**94** ± 3

8X speed up !

- Hard queries from Lucene Benchmark:



Maxscore – Summary

- Most effective for:
  - Large collections
  - Queries w/ high-freq terms, or large result sets resp.
  - Queries w/ many terms
- Benefits
  - Exact (identical results) → easy testing, debugging
  - Negligible overhead → never slower
  - More expensive scoring fct. possible
- Caveats
  - TotalHitCount → approximate, or say ”1000+”
  - Have to decide on `Similarity` at indexing time

Conclusion

DON'T BE AFRAID
to score millions of docs.

Follow and vote for LUCENE-4100 !

Thank you!