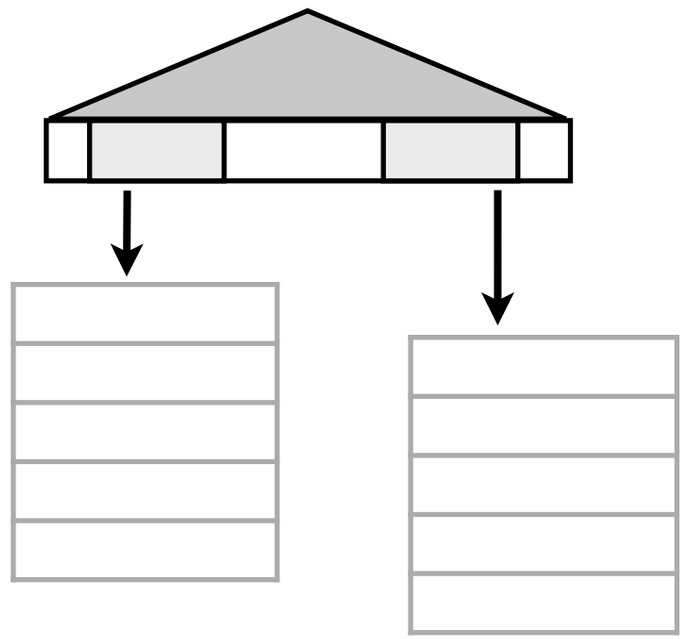
Indexing

Index Organization, Construction, Temporal Query Processing

Inverted Index

- Inverted Indexing basics revisited
- Indexing Static Collections
  - Dictionaries
  - Forward Index
  - Inverted Index Organisation
  - Scalable Indexing
- Indexing Temporal Collections



Text Collections and Indexing

- Why do we index text collections ?
- How do we index documents ?
  - What are the data structures ?
  - What are the design decisions for organising the index?
- How do we index huge collections ?
- How do we index temporal collections ?

Text Collections and Indexing

- Why do we index text collections ?

  Efficient document retrieval

- How do we index documents ?

  - What are the data structures ?

    lexicon, inverted lists

  - What are the design decisions for organising the index?

    document order, score order

- How do we index huge collections ?

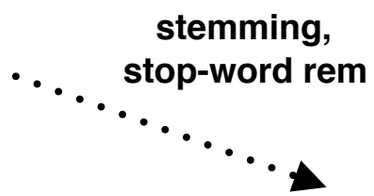
  distributed indexing, term/doc partitioning

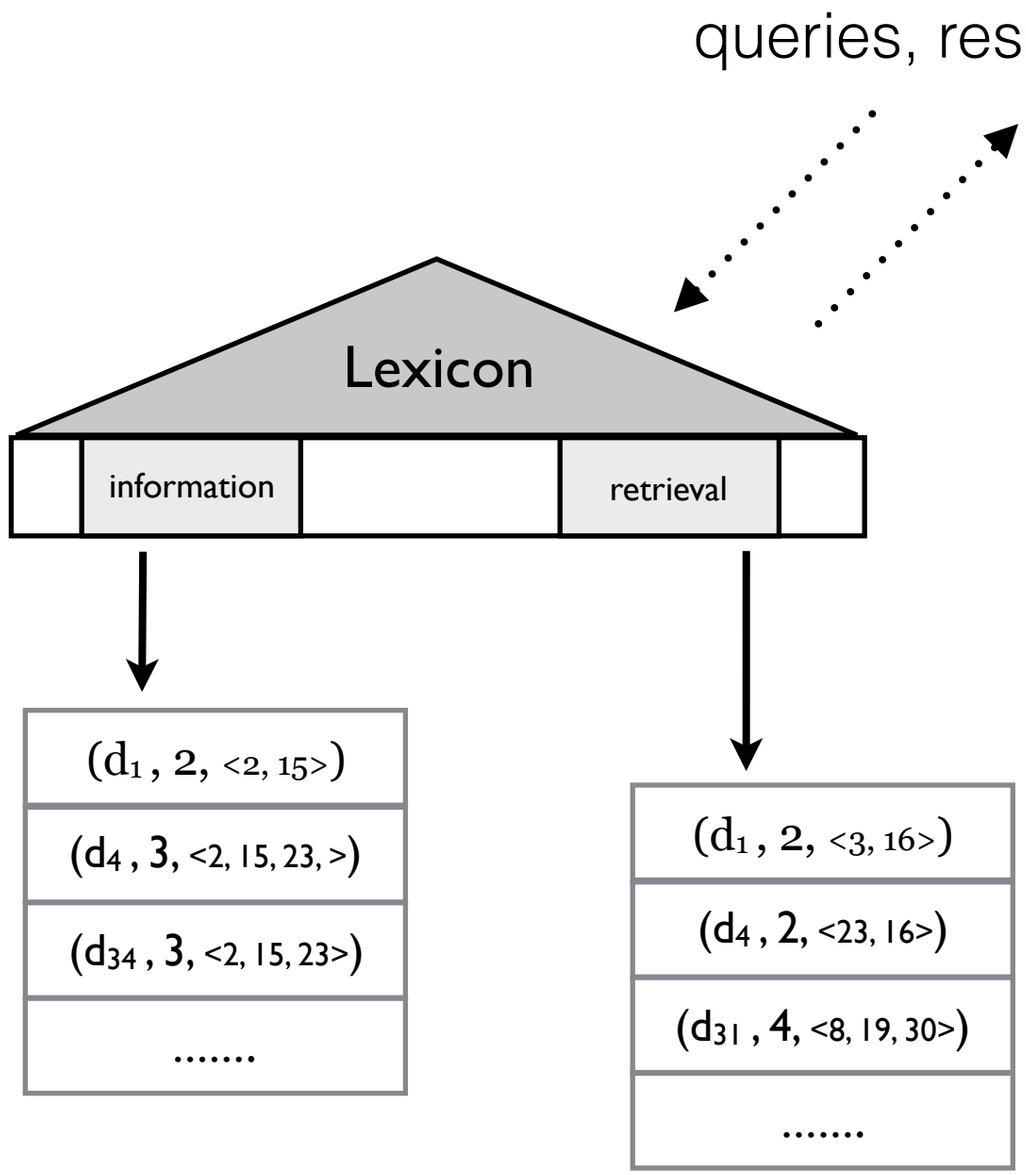
- How do we index temporal collections ?

  index maintenance strategies

Terminology Recap

terms, documents, collection





index, lexicon, posting, posting list

Lexicon or Dictionary

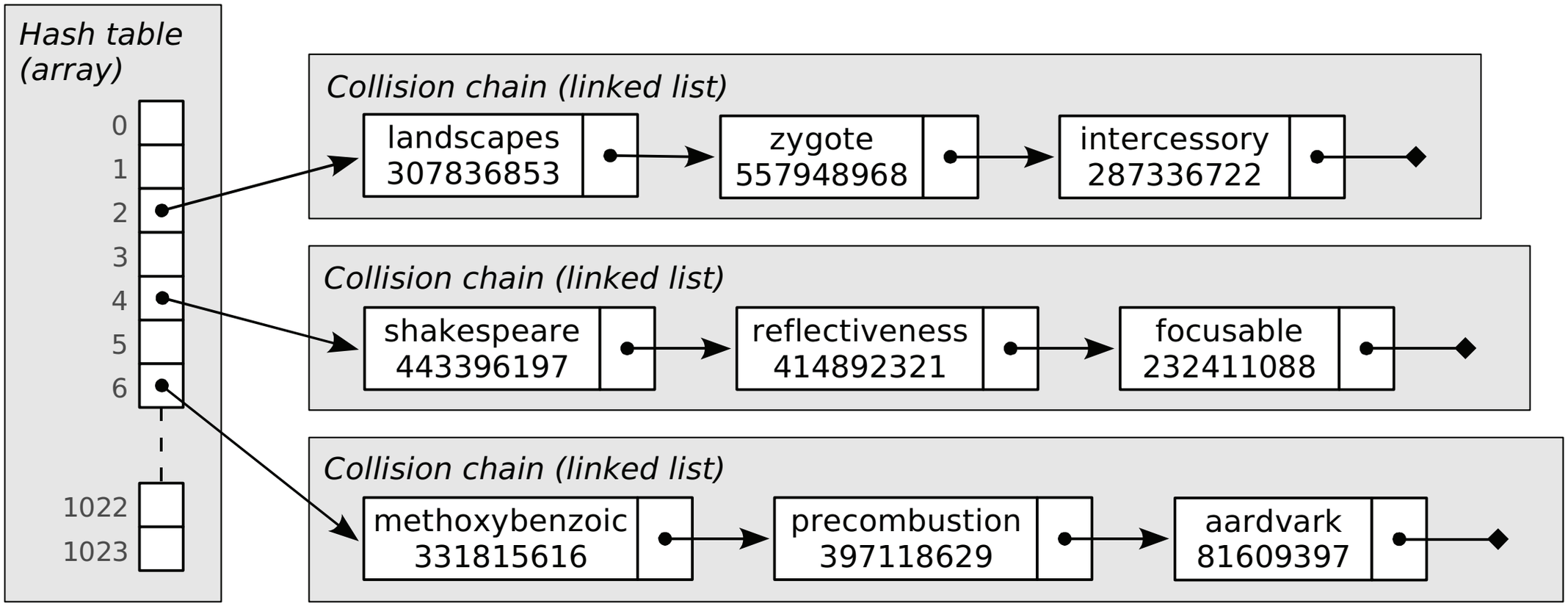
- Maintains **statistics** and **information** about the indexed unit (word, n-gram etc)

  < *hannover* ; **location**: 82271; **tid**:12 ; **df**:23, … >

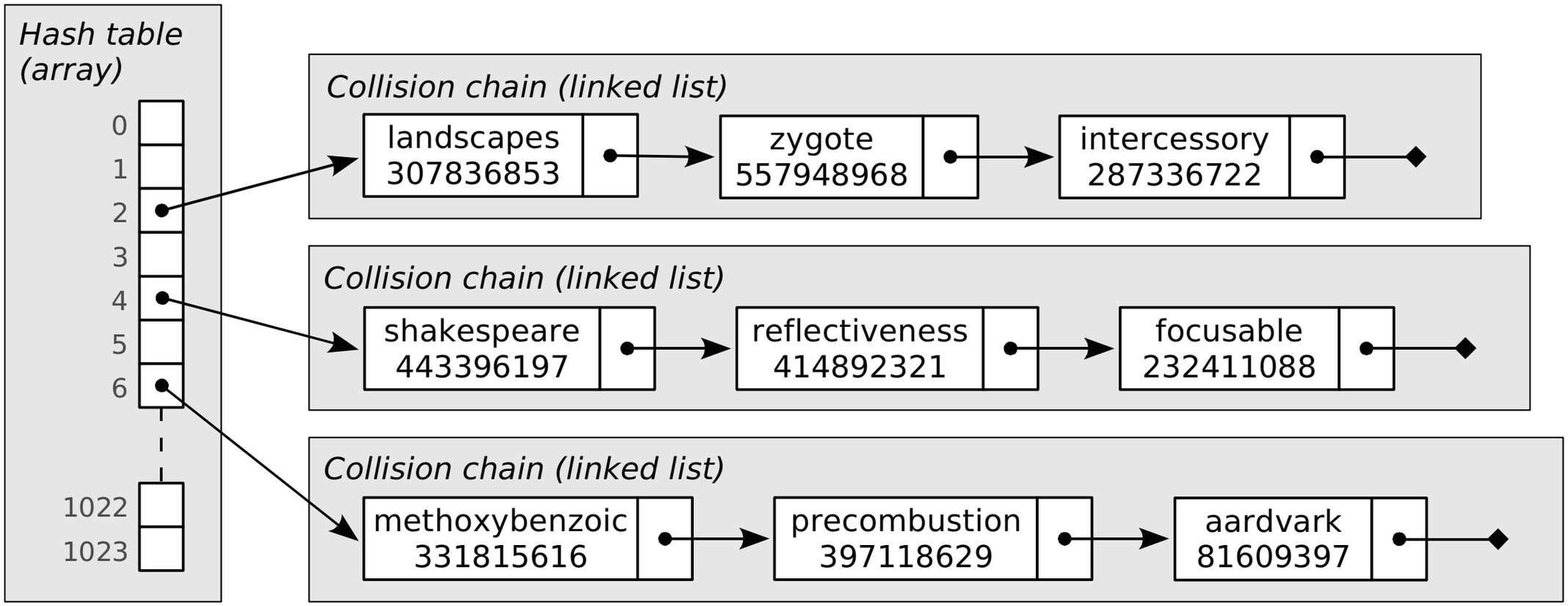
  - Posting list location - for posting list retrieval
  - Term identifier - for term lookups, matching and range queries
  - document frequency and associated statistics - for ranking

- Data Structures for Lexicon
  - Hash-based Lexicon
  - B+-Tree based Lexicon

Hash-Based Lexicon

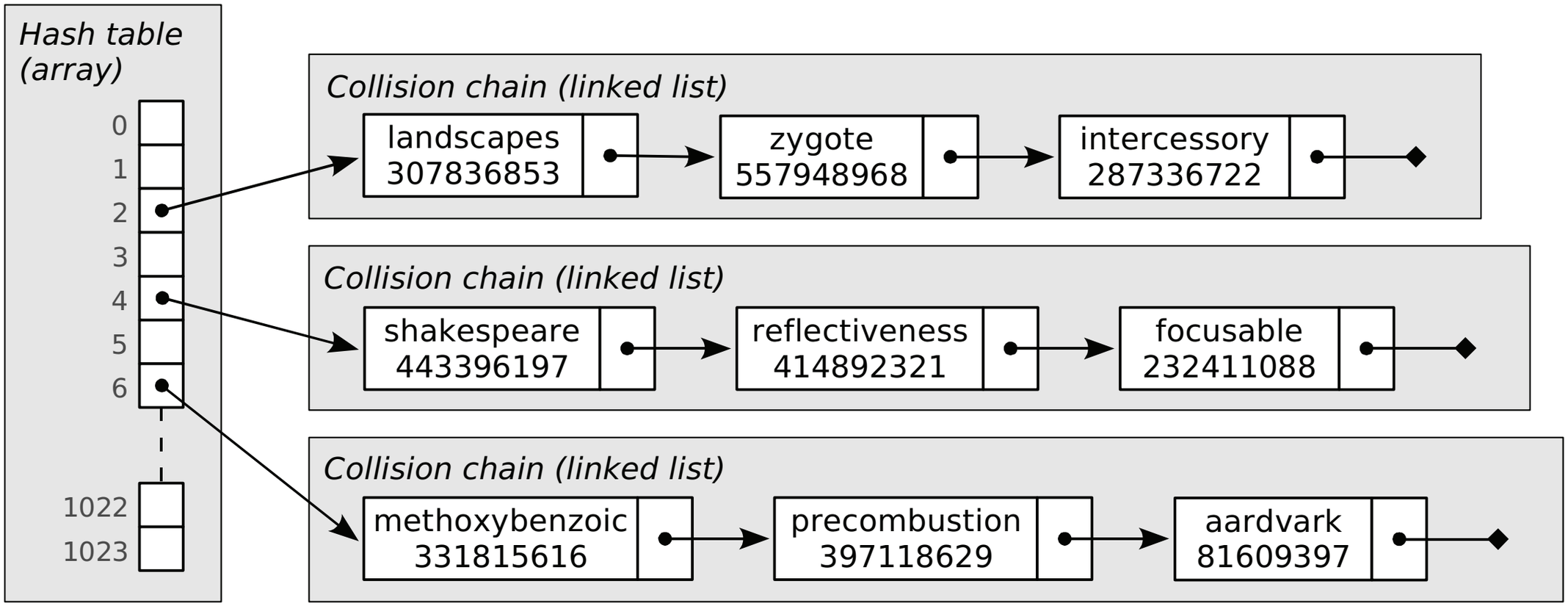


Hash-Based Lexicon



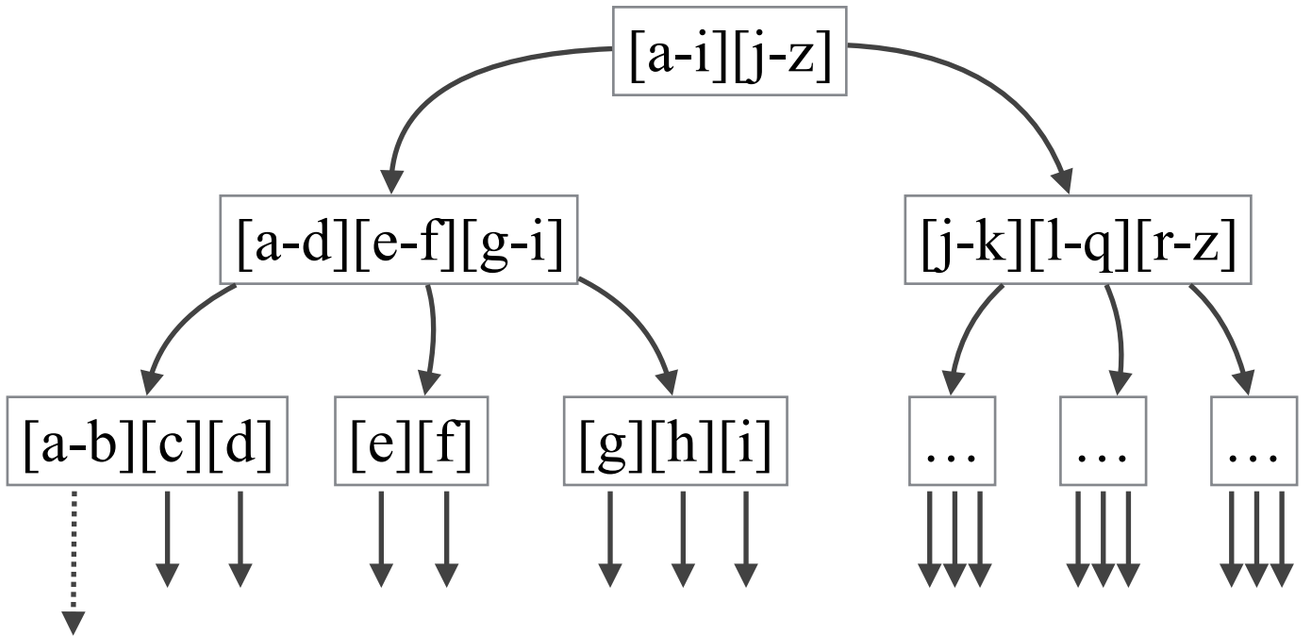
- Constant lookups based on a Hash table
- Entire Lexicon loaded to the memory

Hash-Based Lexicon

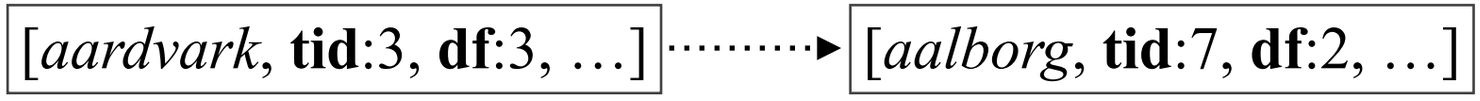


- Constant lookups based on a Hash table
- Entire Lexicon loaded to the memory
- Updates difficult
- Range Searches, Matching, Substring queries not supported

B+-Tree or Sort-based Lexicon



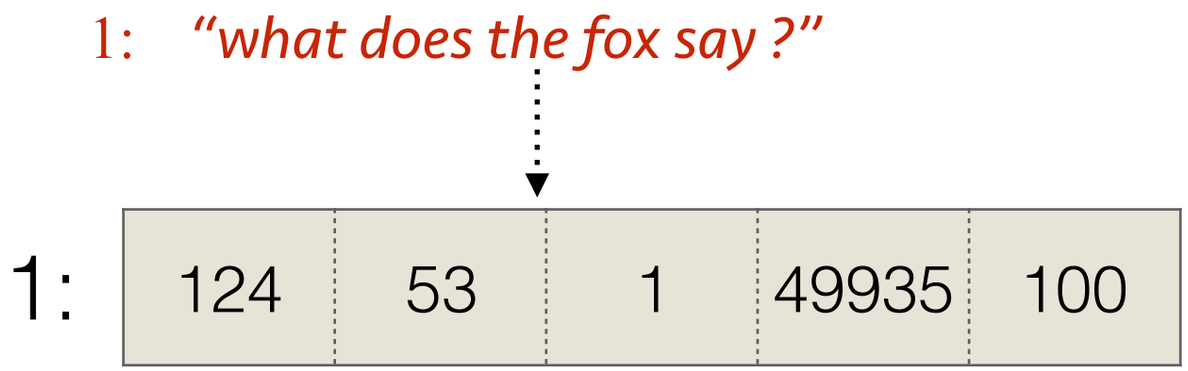
$m = 3$



- **B+-Tree**: Leaf nodes additionally linked for efficient range search
- Supports lookups in $O(\log n)$ and range searches in $O(\log n + k)$
- Vocabulary dynamics (i.e., new or removed terms) no problem
- Works on **secondary storage**

Forward Index

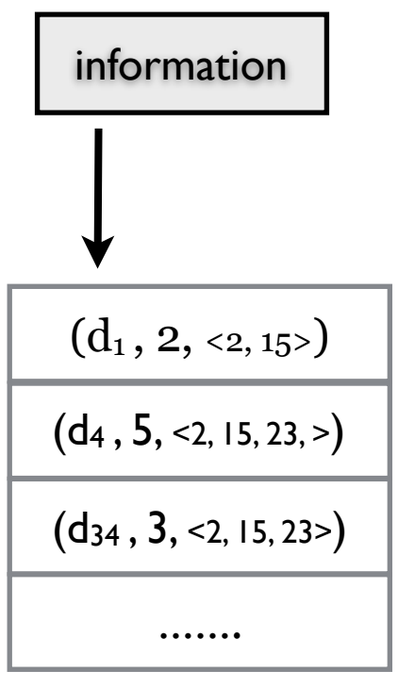
- Mapping of doc-ids to term-ids in the same order



- Efficient retrieval of terms from (already parsed) text
- **snippet generation**
- **proximity features** for proximity-aware ranking
- per-doc term distribution for query expansions

Inverted Index

- Inverted index is a collection of posting lists
- Posting contains document identifiers (as integers) along with scores (integers or doubles) and possibly positions (as integers)
- Postings list can be organised according to
  - document identifiers - document ordering
  - scores - Impact ordering
- What are the merits of these orderings ?



Index Organisation

Document Ordering

- Based on faster intersections
- High compression of index using gap encoding of dids
- Easily updatable

Score/Impact Ordering

- Based on processing Top-k results fast
- Low compression ratio
- Difficult to update

Index organisation depends on query processing style.

Inverted Index Construction

- We are given a set of documents D, where each document d is considered as a bag of terms
- Inverted Lists are created by a process termed as Inversion
- Memory-based Inversion
  - Takes place entirely in-memory
  - For small collections, where the index + lexicon fits in memory
- Disk-based Inversion
  - Sort-based inversion vs Merge-based inversion

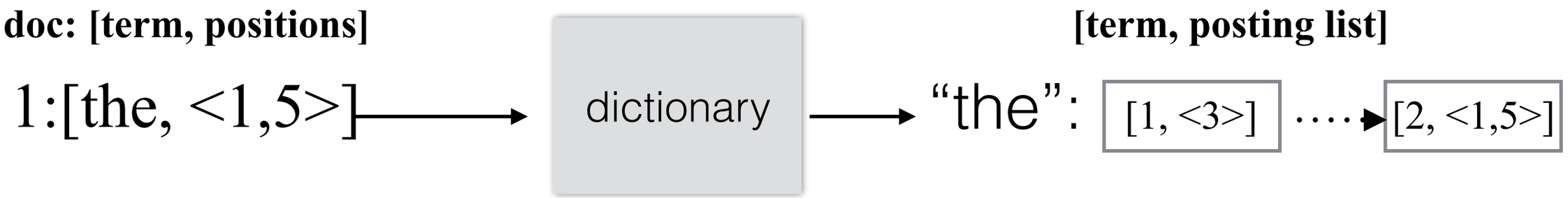
Memory-based Inversion

- A dictionary is required that allows efficient single-term lookup and insertion operations
- An extensible (i.e., dynamic) list data structure is needed that is used to store the postings for each

[term, positions]

1: "what does the fox say ?" …. [the, <3>] [fox, <4>] ….

2: "the fox jumped over the fence" …. [the, <1,5>] [fox, <2>] ….



Sort-based Inversion

- Input Collection D >> memory size M
- Inversion can be seen as a sort operation on the term identifiers
- This method is based on external sort over data which does not fit into the memory
  - Read data of size M into memory, sort them and write back to disk
  - Multiway merge of D/M sorted lists to create index
- Shortcomings
  - Dictionary might not fit in-memory
  - Large memory requirements due to intermediate data

Exercise 1: Analysis of Sort-based Inversion

Simple Computational Model

Total number of postings = N

Number of postings which fit in memory = M

Cost of disk read/write of a posting = c

- What is the estimated cost of sort-based Inversion in terms of N,M and c ?

- How does the cost compare with in-memory sort-based inversion (assuming we had enough memory or N > M) ?

Merge-based Inversion

- Generalisation of in-memory indexing
- Reads input collection to create an in-memory index of size M and write it to disk to create partial indexes with local lexicons
- Compression in posting lists in partial indexes
- Multiway Merge of corresponding lists from the partial indexes to create one consolidated index

Map-Reduce crash course

- Programming paradigm for distributed data processing
- Improves overall throughput by parallelising loading of data
- Data is partitioned into the nodes which process the data in the following phases
- Map : Generates (key, value) pairs
- Shuffle : Shuffles the pairs over the network to the reducers
- Reduce : operates on all values for the same key are

Map-Reduce Example : Word Count

1: "what does the fox say ?"

2: "the fox jumped over the fence"

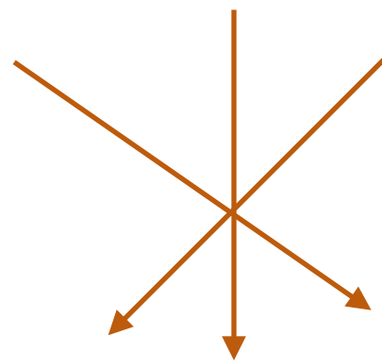
Mapper - 1

what : 1
does : 1
the : 1
fox : 1
say : 1

Mapper - 2

jumped : 1
over : 1
the : 2
fox : 1
fence : 1

mappers emit <word, freq>



Shuffle + Sort

reducers aggr. freq.

Reducer - 1

does : 1
fence : 1
jumped : 1
fox : 1 + fox : 1
over : 1
the : 1 + the : 2
say : 1
what : 1

Reducer - 2

Exercise 2: Index Construction using Map-Reduce

- How would you build the inverted index using Map-reduce ?
  - What are the key-value pairs as defined by the Mapper ?
  - What does the reducer do with the values of the same key ?

Temporal Collections and Queries

- Temporal collections have temporal information
  - Publication times - News Articles
  - Valid times - Wikipedia articles, Web archive versions
  - Temporal references - time mentions in text
- Time-travel Queries : Retrieve all documents relevant to the text and time
  - Point in time queries *house of cards @ 02/03/2013*
  - Time-interval queries *game of thrones @ 2011 - 2014*

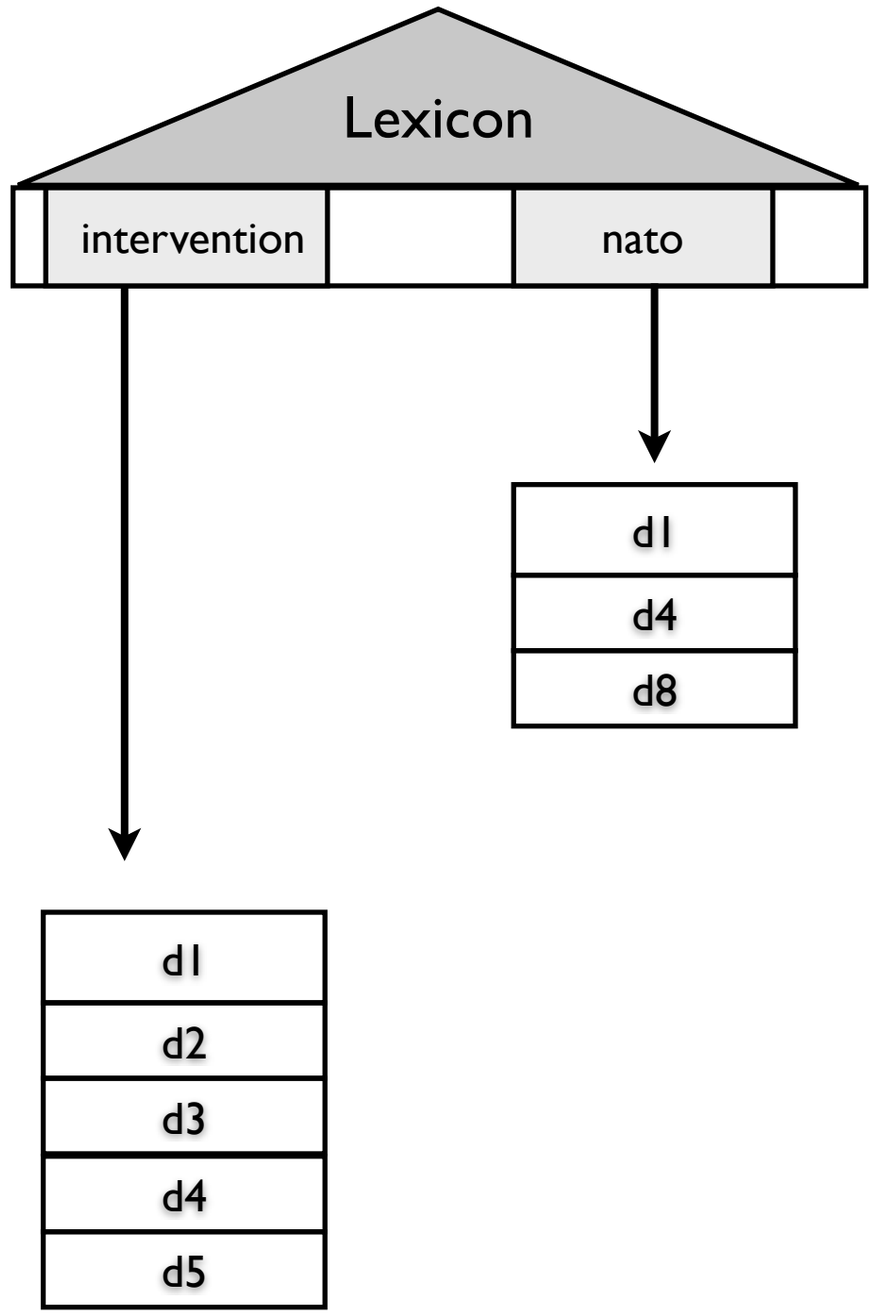
Temporal Indexing

- Given a versioned collection of documents with valid time intervals
- and a time-travel text query: *game of thrones @ 2011 - 2014*
- We want to retrieve documents containing terms "game" ,"thrones" and valid between *2011 - 2014*

How do we efficiently retrieve these documents ?

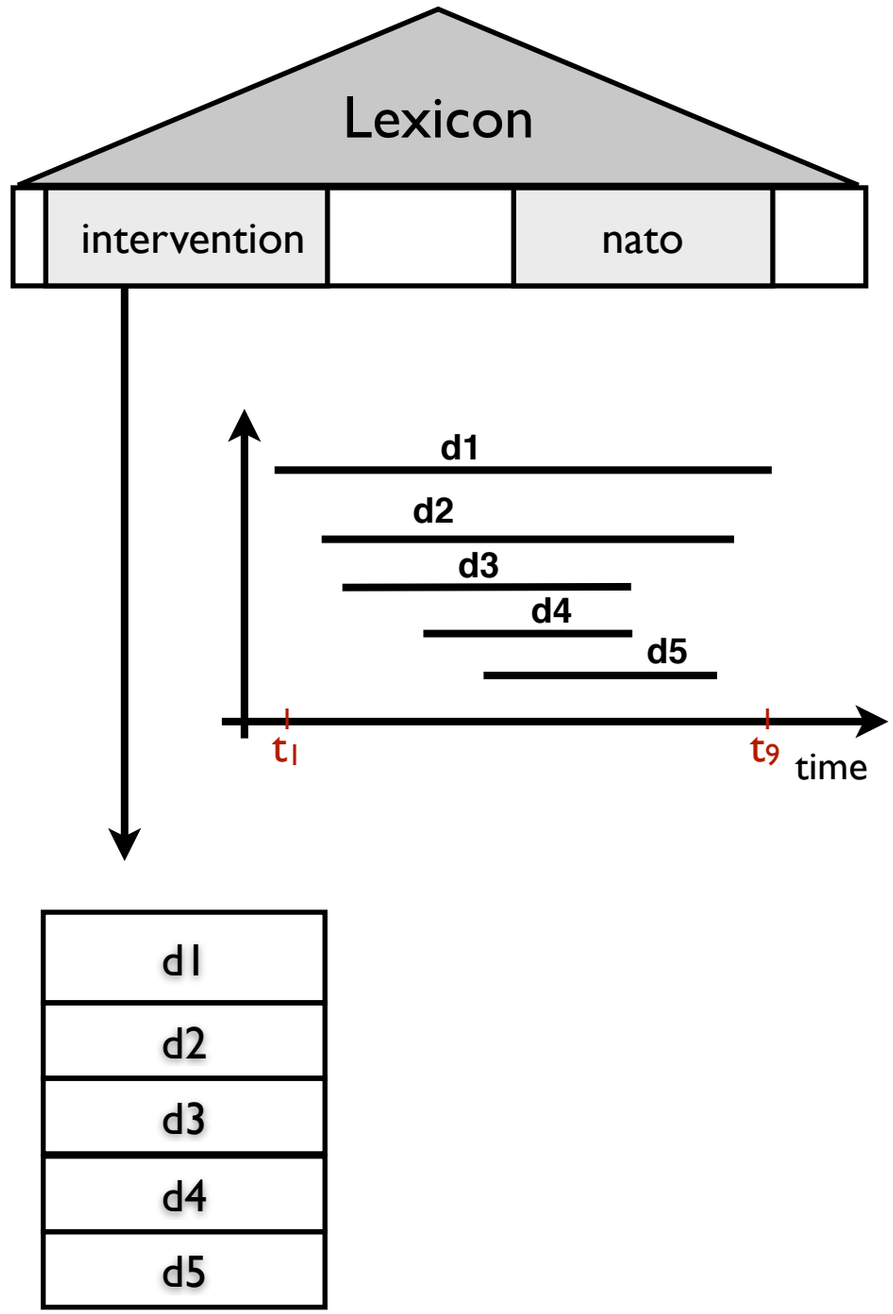
Time-Travel Index

- Inverted index processes keyword queries
- Intersection of posting lists for processing queries
- Versions have valid time intervals
- Augment postings with valid time intervals
- Post filtering after standard query processing



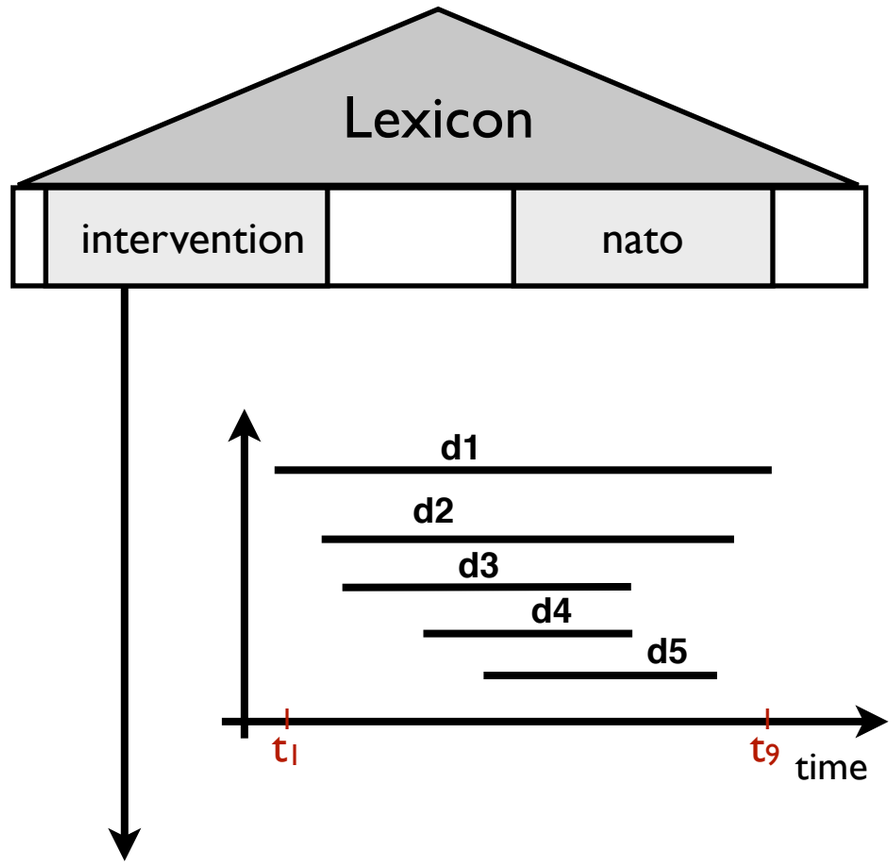
Time-Travel Index

- Inverted index processes keyword queries
- Intersection of posting lists for processing queries
- Versions have valid time intervals
- Augment postings with valid time intervals
- Post filtering after standard query processing



Time-Travel Index

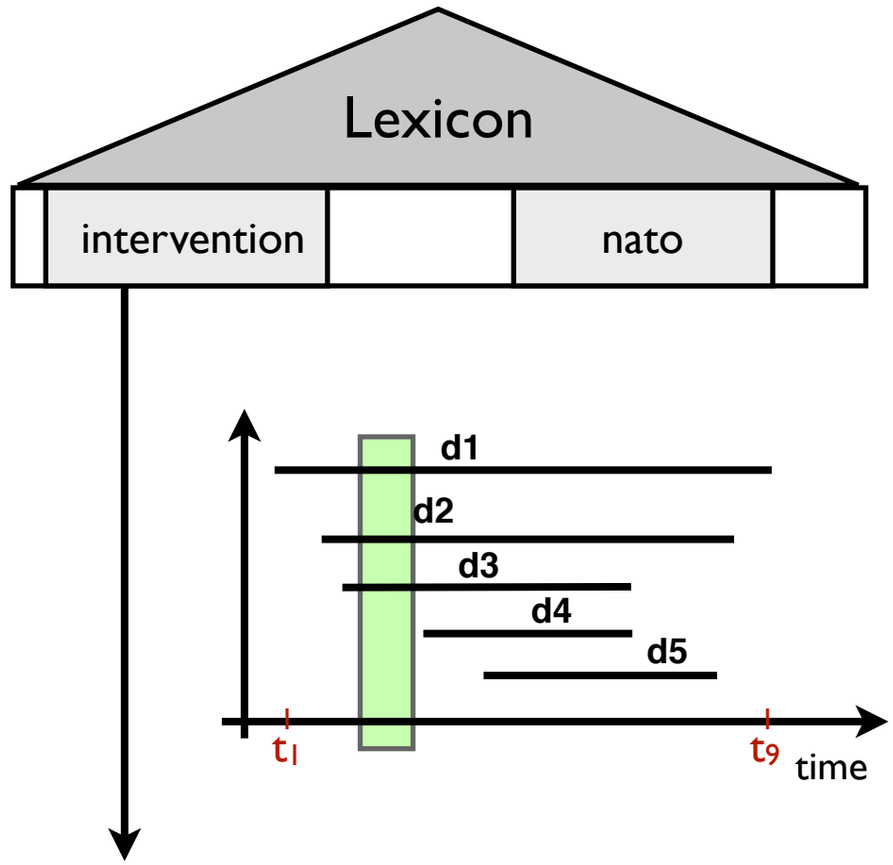
- Inverted index processes keyword queries
- Intersection of posting lists for processing queries
- Versions have valid time intervals
- Augment postings with valid time intervals
- Post filtering after standard query processing



(d1, [t1, t9))
(d2, [t2, t8))
(d3, [t3, t6))
(d4, [t4, t6))
(d5, [t5, t7))

Time-Travel Index

- Inverted index processes keyword queries
- Intersection of posting lists for processing queries
- Versions have valid time intervals
- Augment postings with valid time intervals
- Post filtering after standard query processing



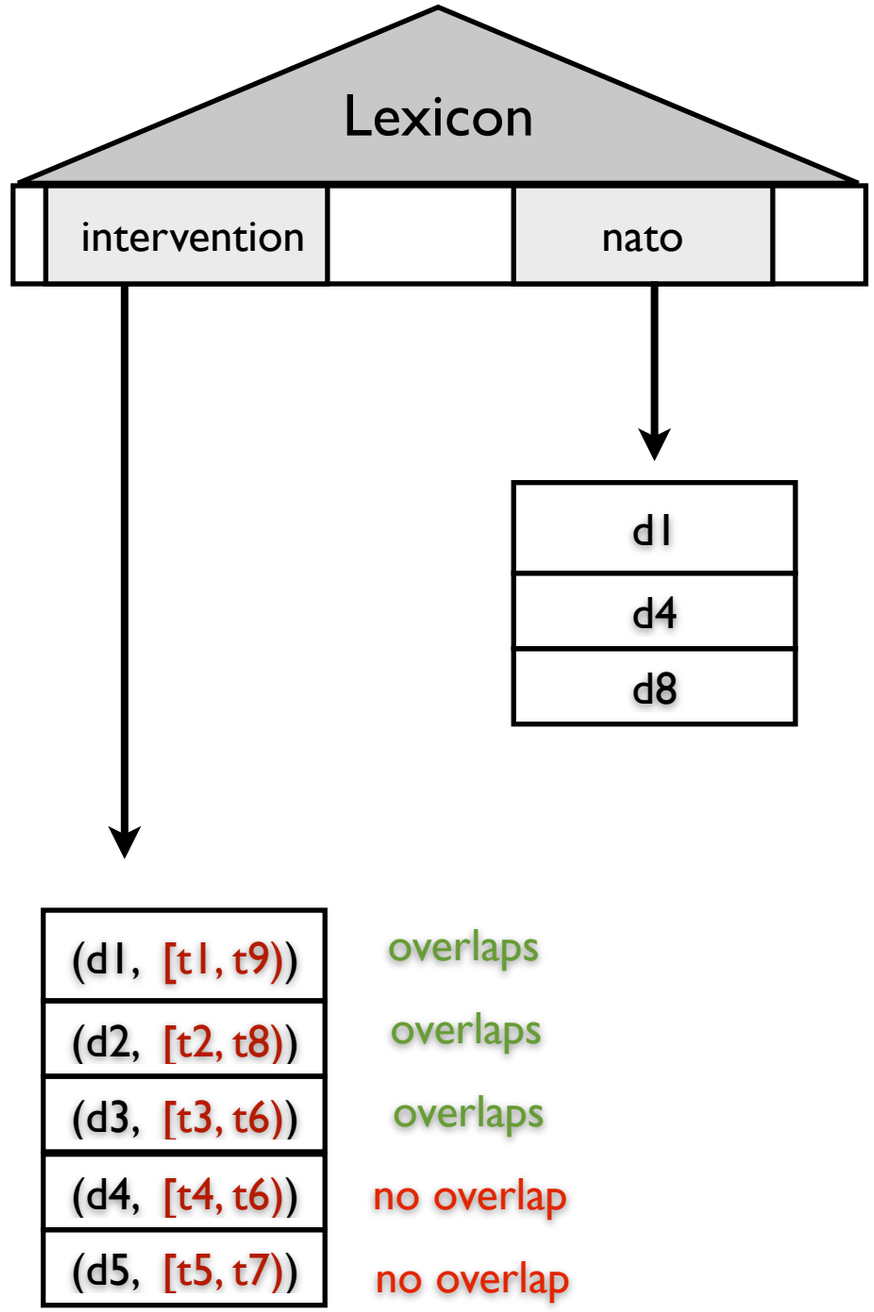
(d1, [t1, t9))	overlaps
(d2, [t2, t8))	overlaps
(d3, [t3, t6))	overlaps
(d4, [t4, t6))	no overlap
(d5, [t5, t7))	no overlap

Challenges in Indexing

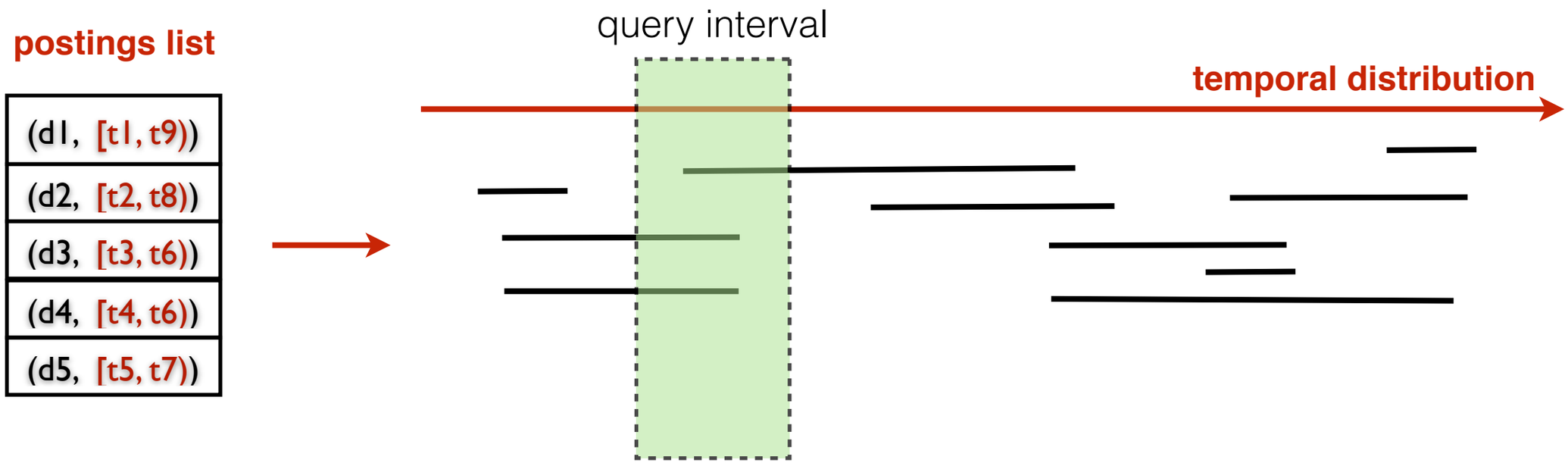
- If documents are points in time how would you organize the index ?

- What is the problem if the documents are associated with time intervals ?

- Query processing expensive due to wasted accesses



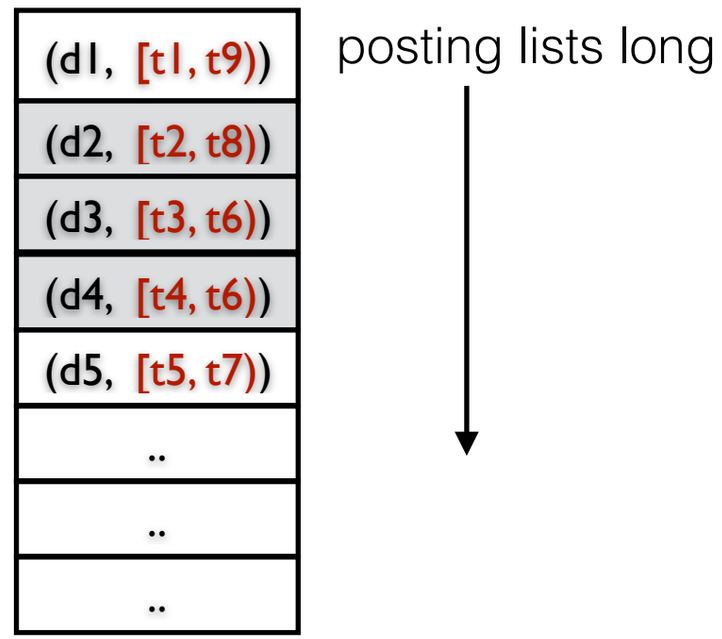
Data and Query Model



- Each interval represents a document in the posting list
- Data Model: Each document is associated with a time interval
- Query Model: Queries are associated with a time interval [tb, te]
  - point in time queries : when begin time = end time
  - time interval queries

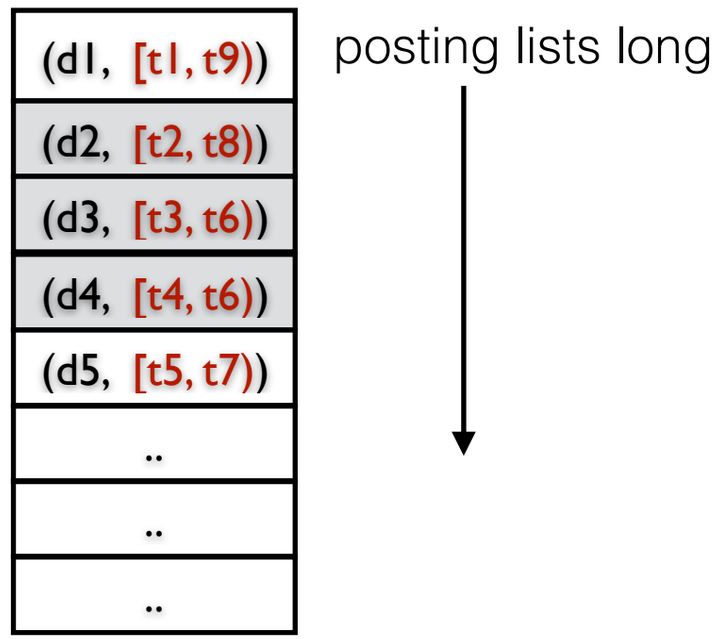
Challenges in Indexing Time

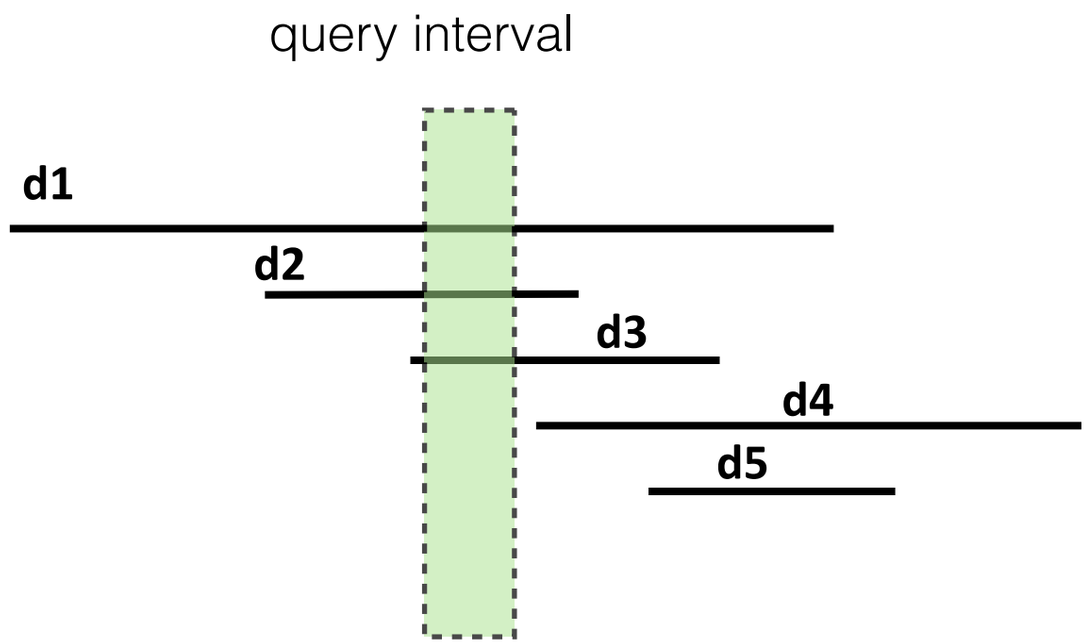
- We would want to avoid unwanted or wasted access to posting lists
- Typically only access those postings that are relevant or a few more (bounded loss)
- Dealing with time points easy, akin to range queries (sorting acc to begin time and range search)



Challenges in Indexing Time

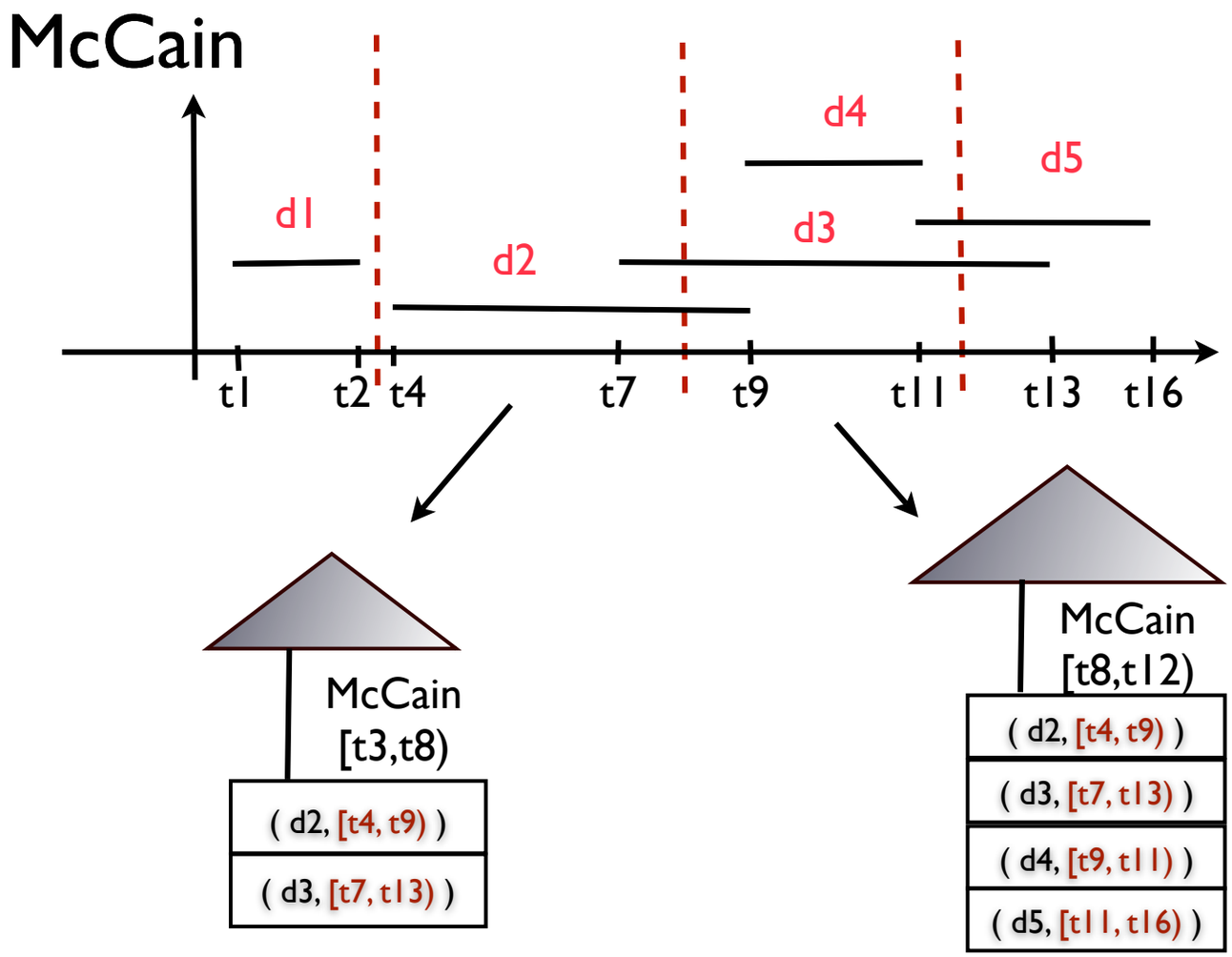
- We would want to avoid unwanted or wasted access to posting lists
- Typically only access those postings that are relevant or a few more (bounded loss)
- Dealing with time points easy, akin to range queries (sorting acc to begin time and range search)





Index List Partitioning

- Vertically Partition the temporal space and each partition
- Now multiple posting lists per term, each with a valid time interval
- Limits index access, introduces replication

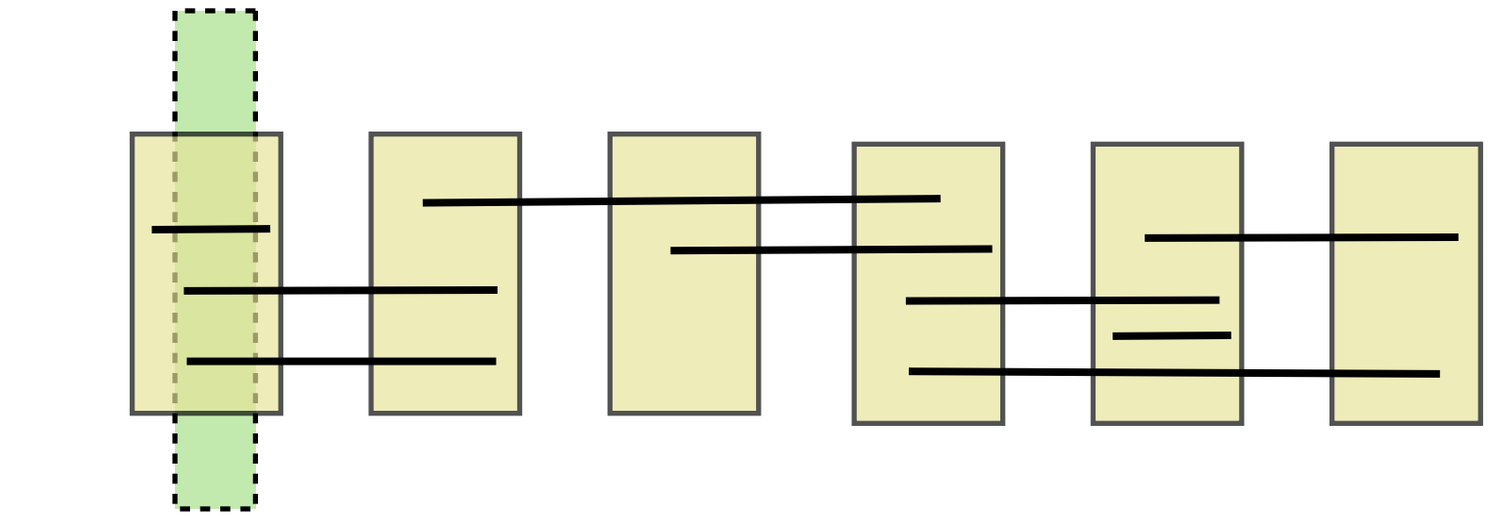


Vertical Partitioning - Query Processing

- Dictionary or Lexicon should contain partitioning information
- For each temporal query, select a subset of affected partitions and only read them
- Filter postings which do not overlap with query time interval

term	partition	offset
hannover	[t1 - t5)	12646
hannover	[t5 - t7)	12673
hannover	[t7 - t25)	13446
hannover	[t25 - t43)	15324

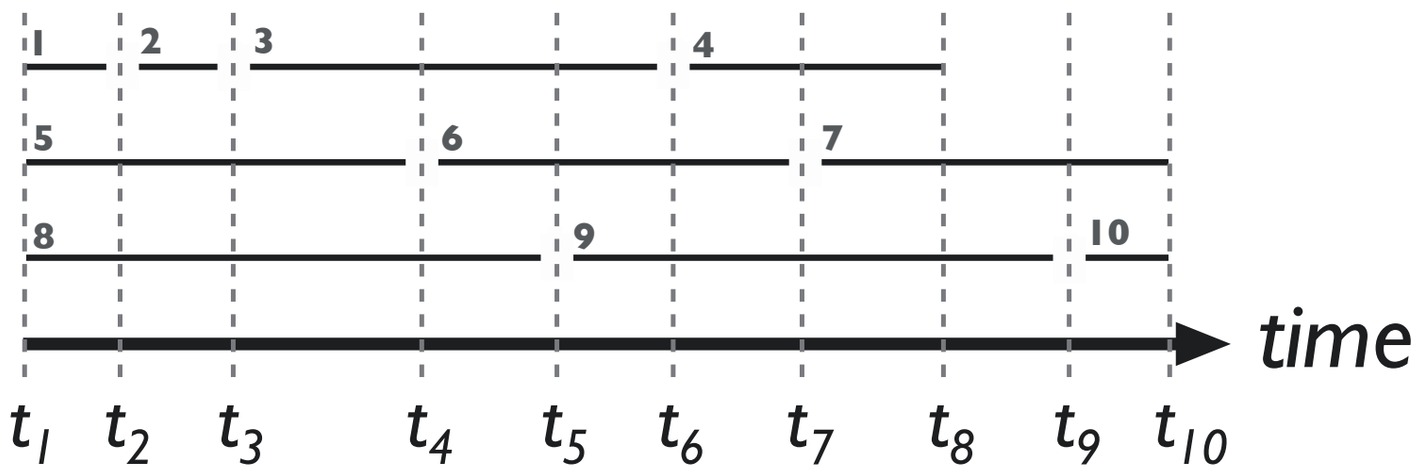
“hannover”



time interval query

Optimal Approaches

- **Performance Optimal Approach**



  - Keeps one posting list per every elementary time interval,
  - Achieves optimal performance but large space overhead

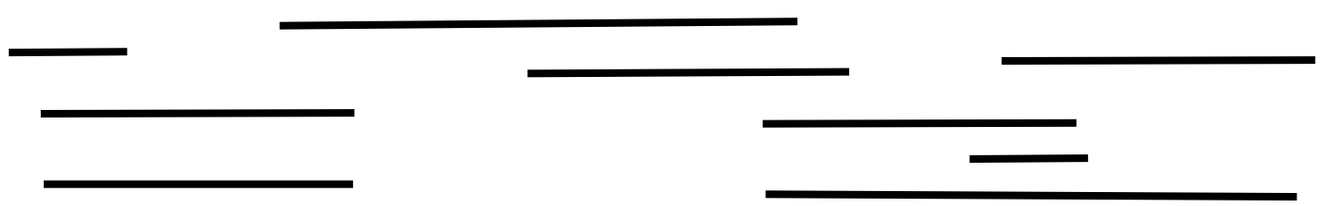
- **Space Optimal Approach**
  - No replication of postings, no blowup, sub-optimal performance

Partitioning Strategies

- Given an input sequence of intervals how do we partition them into sublists ?

- Space Bound Materialization Approach : We have a limited budget for space, need to maximize our performance

- Performance Guarantee Approach: For any query we need a guarantee on the performance loss, need to minimize blowup



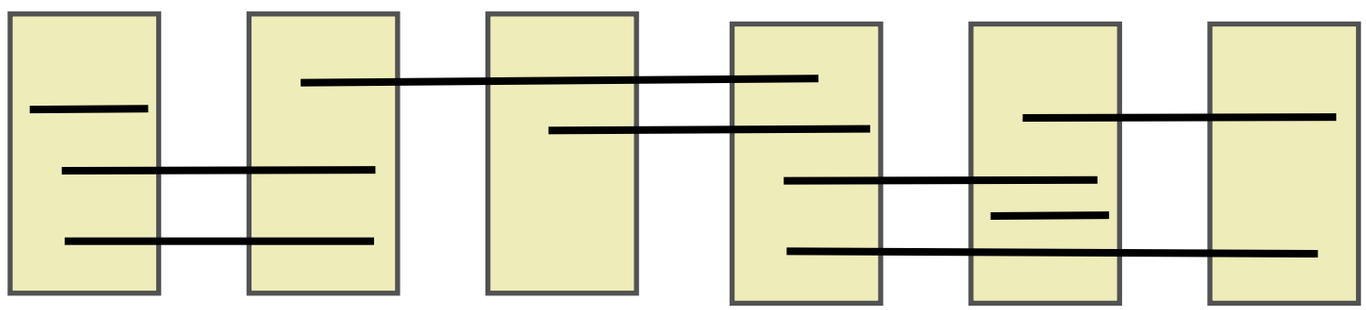
Trade-off size and performance

Partitioning Strategies

- Given an input sequence of intervals how do we partition them into sublists ?

- Space Bound Materialization Approach : We have a limited budget for space, need to maximize our performance

- Performance Guarantee Approach: For any query we need a guarantee on the performance loss, need to minimize blowup

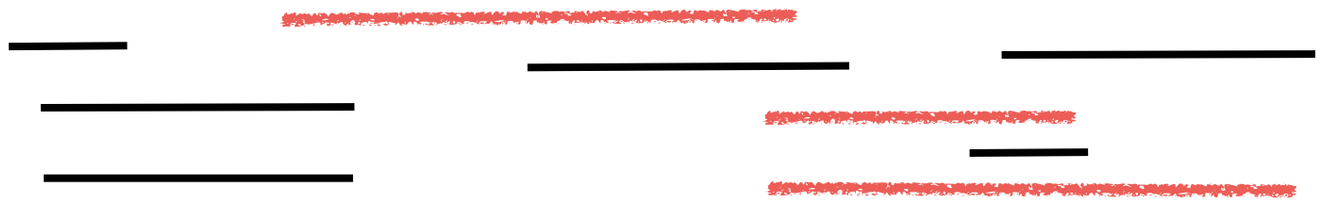


Trade-off size and performance

Space Bound Approach

- Minimize expected number of postings read for a time-point query, while ensuring that the index contains at most κ times the optimal number of postings

- Optimal solution computable in **O(|S| × n2)** time and **O(|S| × n)** space using dynamic programming over prefix subproblems [t1, tk) and space bounds **s ≤ κ · |Lv|**

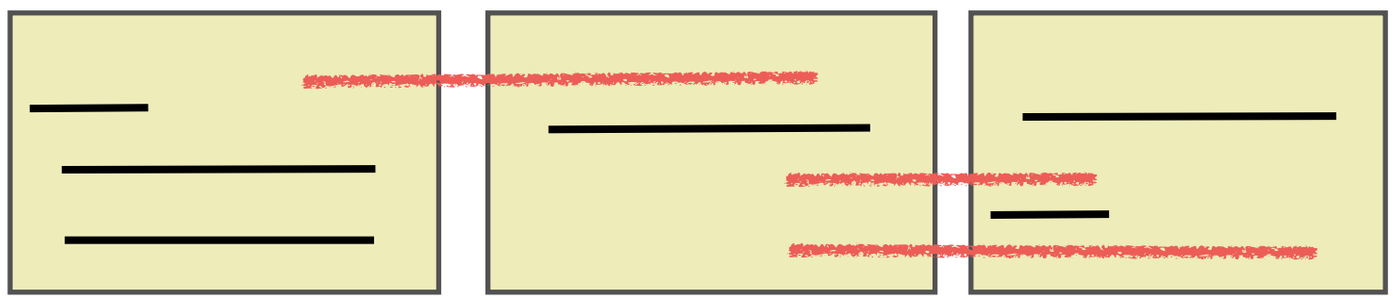


Space budget = 1/3 . (optimal number of postings)

Space Bound Approach

- Minimize expected number of postings read for a time-point query, while ensuring that the index contains at most κ times the optimal number of postings

- Optimal solution computable in **O(|S| × n2)** time and **O(|S| × n)** space using dynamic programming over prefix subproblems [t1, tk) and space bounds **s ≤ κ · |Lv|**

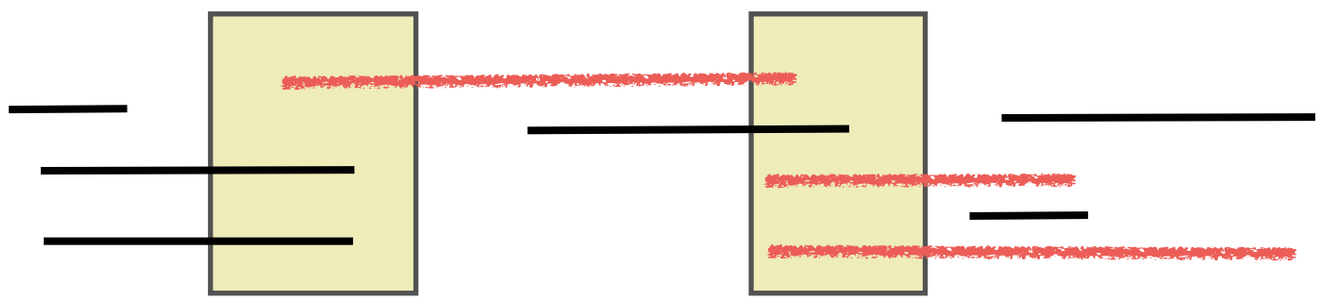


Space budget = 1/3 . (optimal number of postings)

Performance Guarantee

- Minimize total number of postings kept in the index, while guaranteeing that for any time-point query the number of postings read is at most a factor γ worse than optimal

- Optimal solution computable in time O(|Lv| + n2) and space O(n2) using dynamic programming over prefix subproblems [t1, tk)

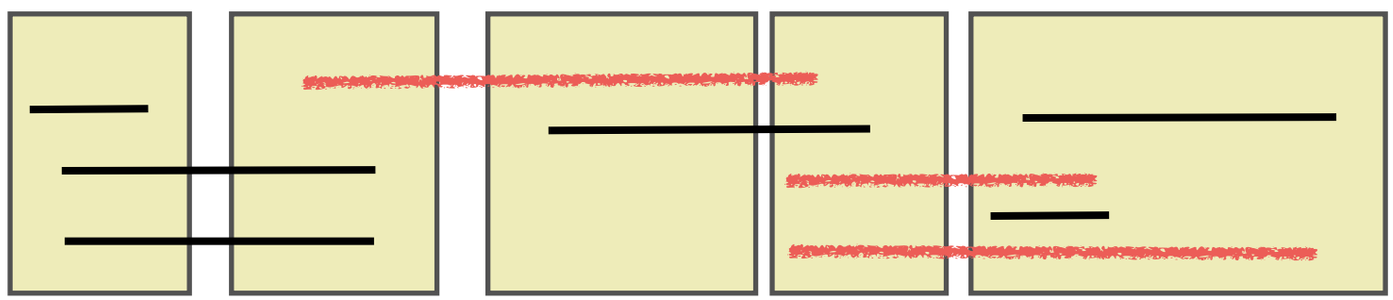


performance guarantee = γ (times the number of optimal results at that time)

Performance Guarantee

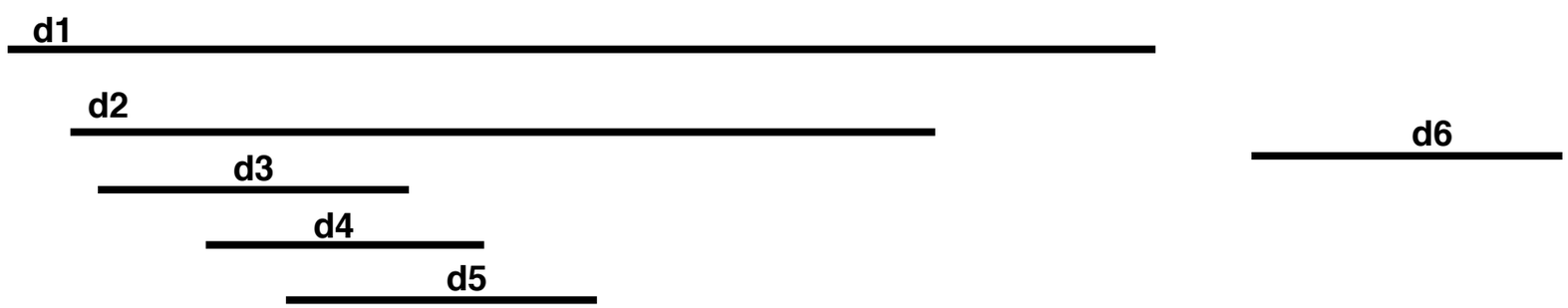
- Minimize total number of postings kept in the index, while guaranteeing that for any time-point query the number of postings read is at most a factor γ worse than optimal

- Optimal solution computable in time O(|Lv| + n2) and space O(n2) using dynamic programming over prefix subproblems [t1, tk)



performance guarantee = γ (times the number of optimal results at that time)

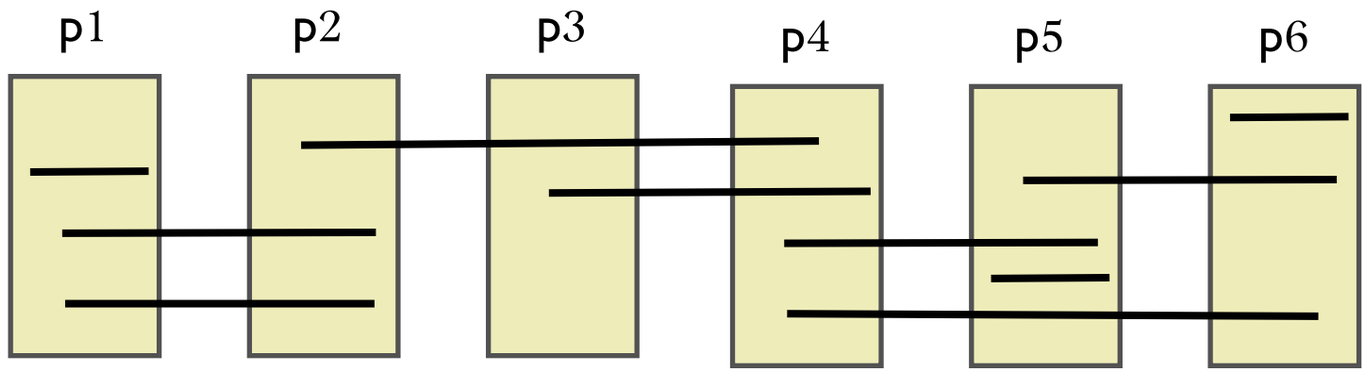
Exercise



- Performance Guarantee with γ = 2 **(read at max twice the number of postings than optimal)**

- Space bound approach with **κ = 1.33 (1/3 more than overall space)**

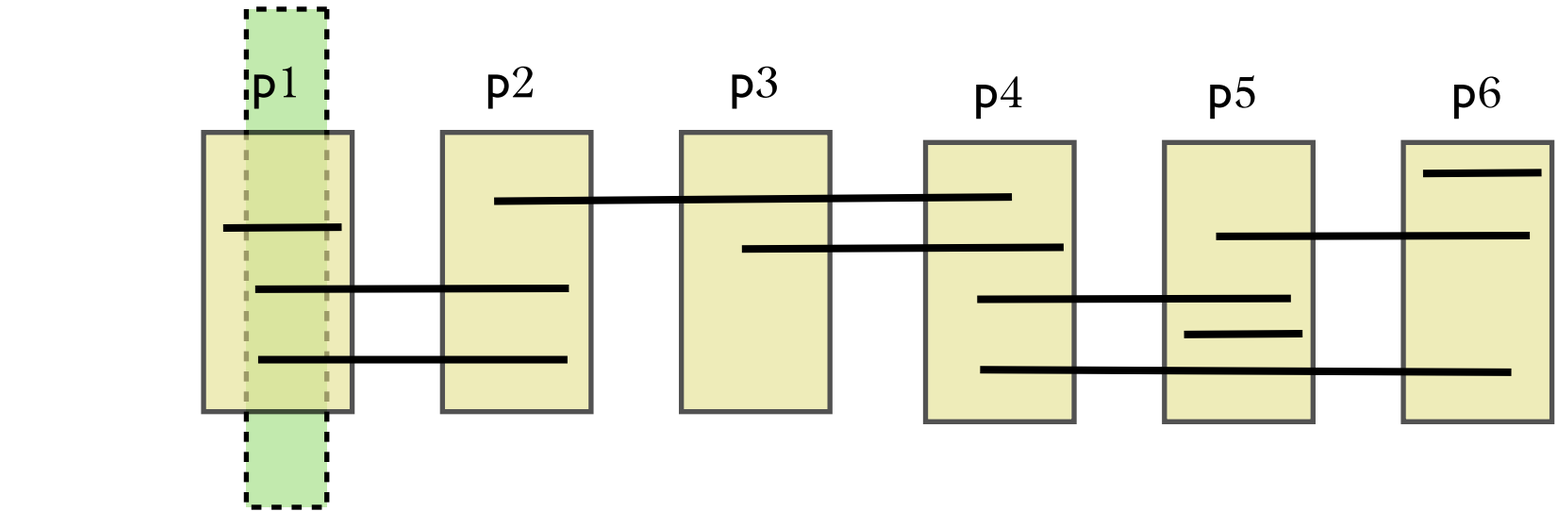
Horizontal Partitioning - Sharding



- Index size blowup due to replication of postings across slices
- Query processing inefficient if replicated postings are accessed multiple times

Can we partition a posting list without replicating postings ?

Horizontal Partitioning - Sharding



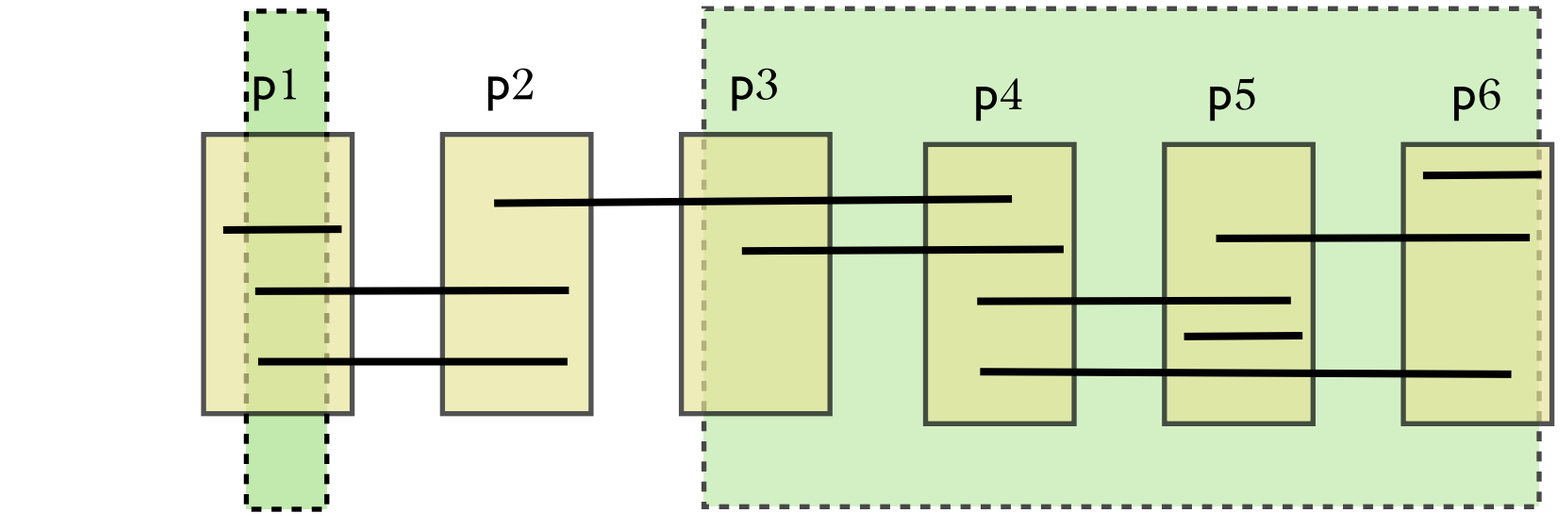
short time interval query

Relevant postings = 3
Postings read : 3

- Index size blowup due to replication of postings across slices
- Query processing inefficient if replicated postings are accessed multiple times

Can we partition a posting list without replicating postings ?

Horizontal Partitioning - Sharding

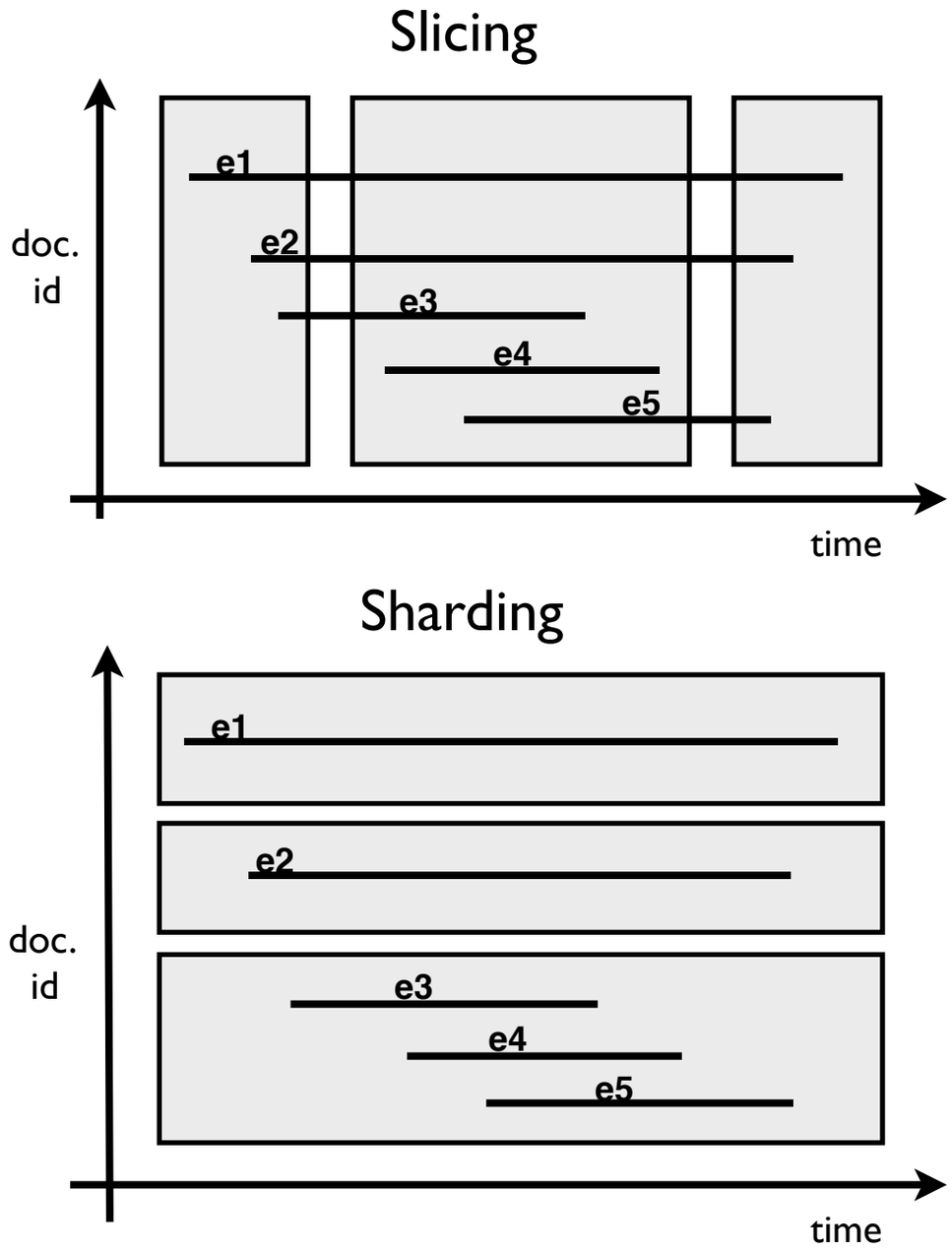


- Index size blowup due to replication of postings across slices
- Query processing inefficient if replicated postings are accessed multiple times

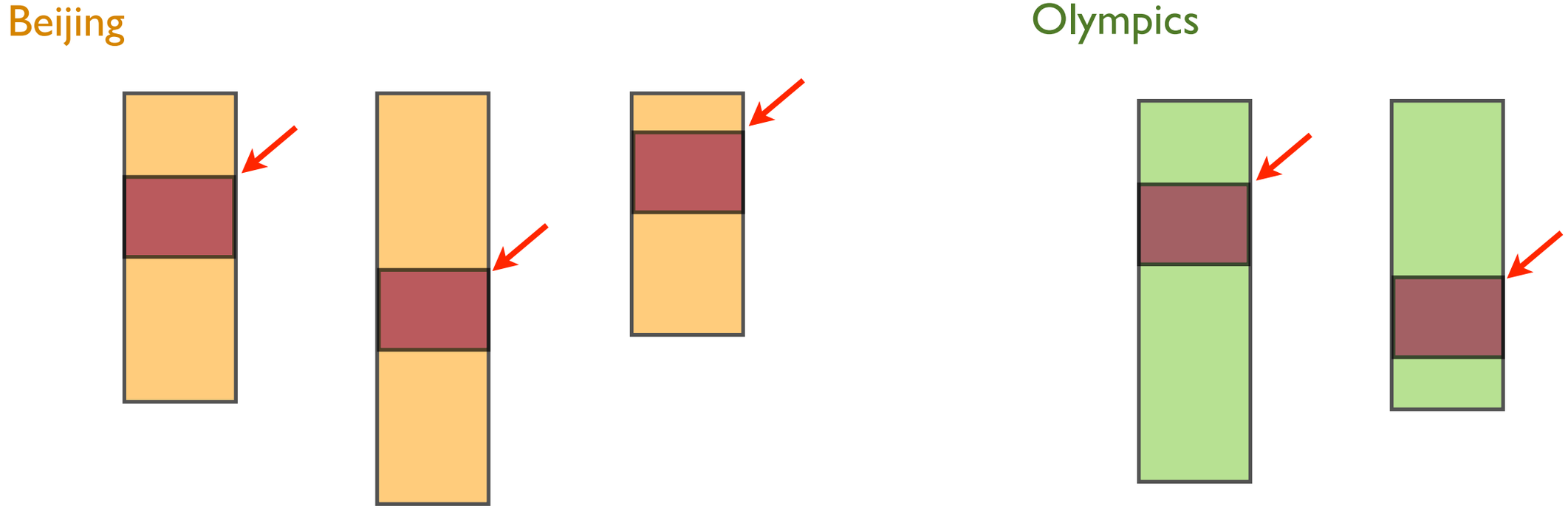
Can we partition a posting list without replicating postings ?

Index Sharding

- Partition documents in each posting list into sublists called shards
- Contents of each shard disjoint - no replication, no index blowup
- Postings stored in begin time order
- Access structure over each shard for efficient query processing



Index Sharding



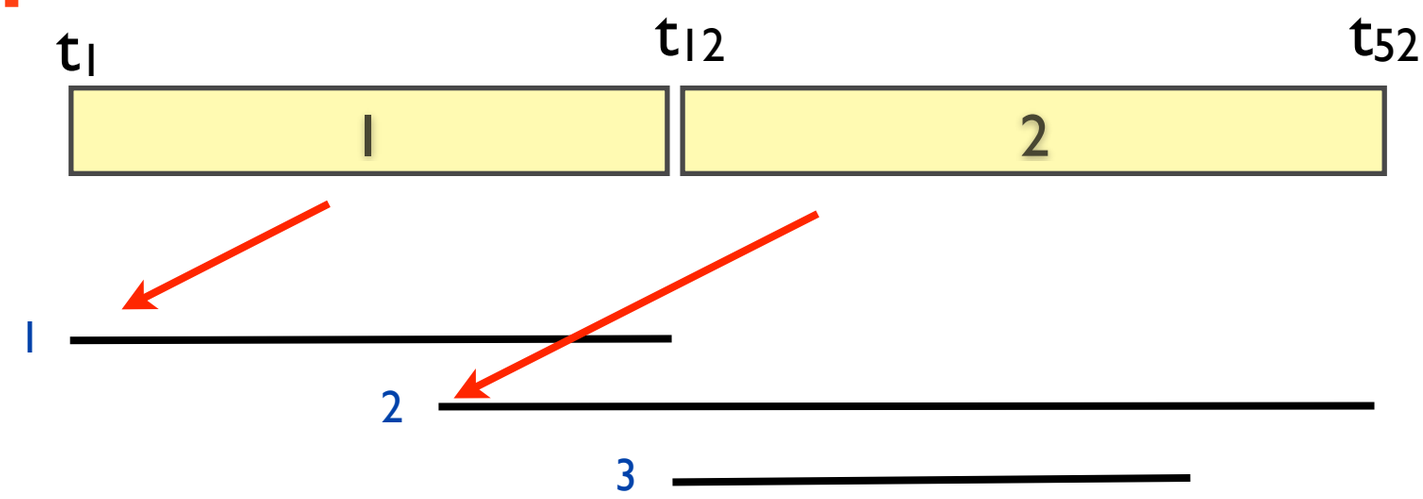
beijing olympics @ [8 Aug 2008, 24 Aug 2008]

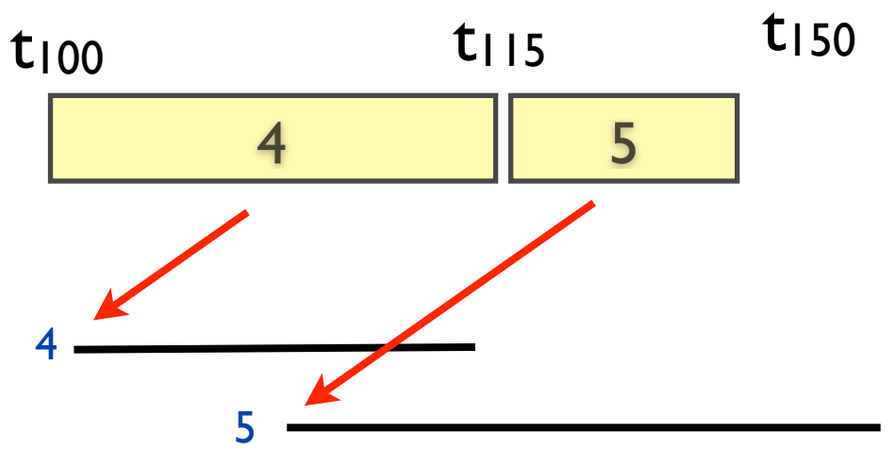
- All shards for a given query term are accessed
- Open-skip-scan on each shard assisted by impact lists
- Result list constructed by merging results from each shard

Index Sharding - Impact Lists

- Open - Each shard of a query term opened for access
- Skip - Given a query begin time seek to appropriate offset
- Scan - Read while postings still have overlap with query time interval

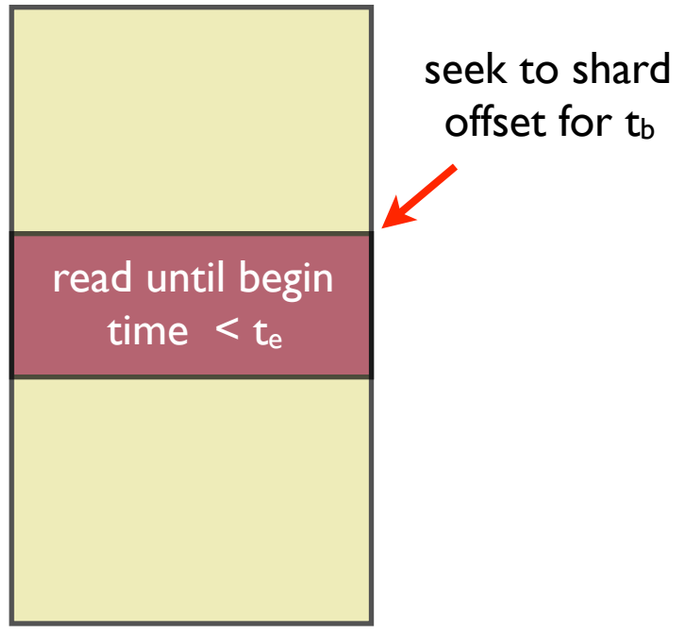
Impact list



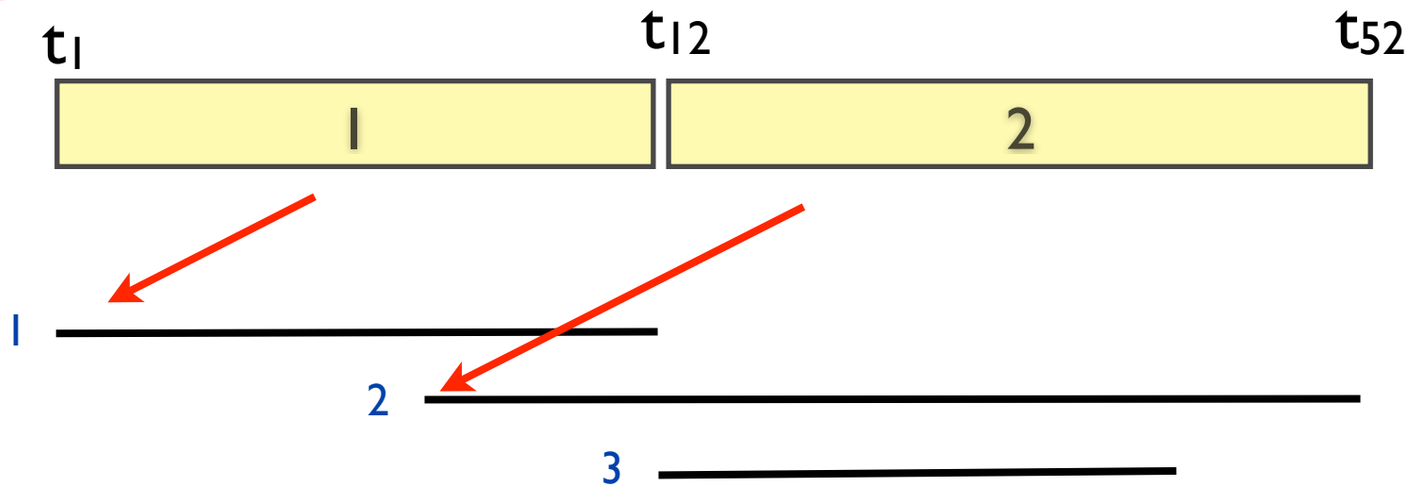


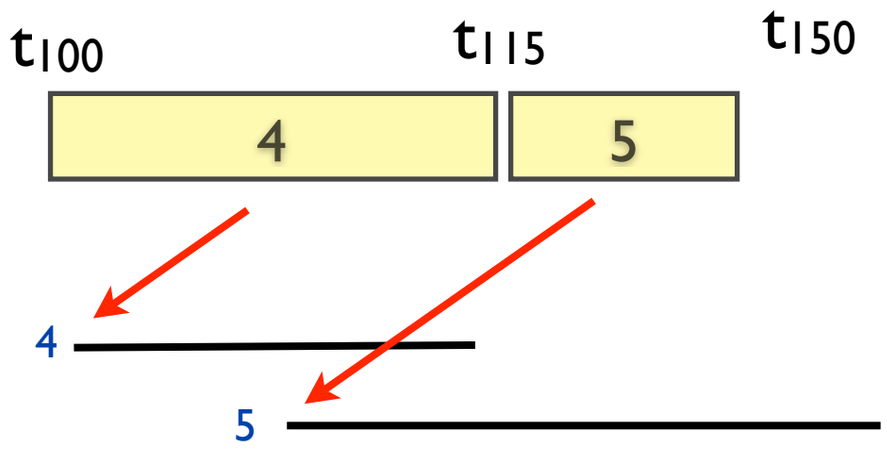
Index Sharding - Impact Lists

- Open - Each shard of a query term opened for access
- Skip - Given a query begin time seek to appropriate offset
- Scan - Read while postings still have overlap with query time interval



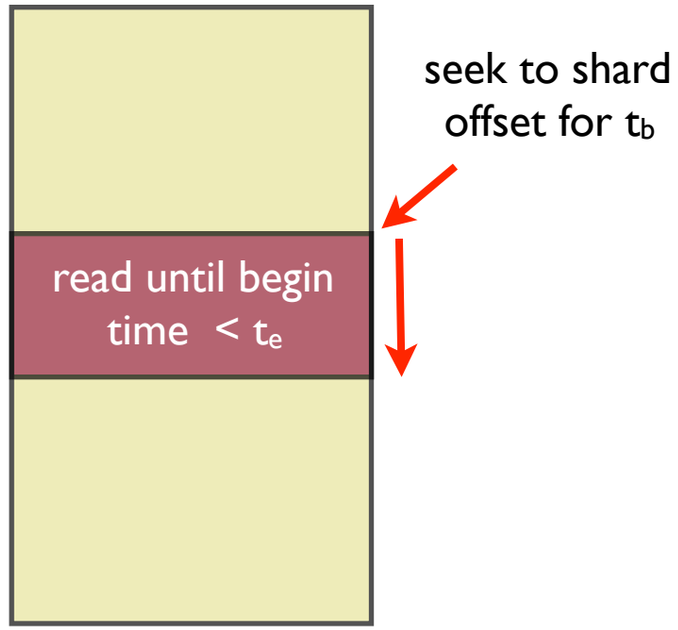
Impact list



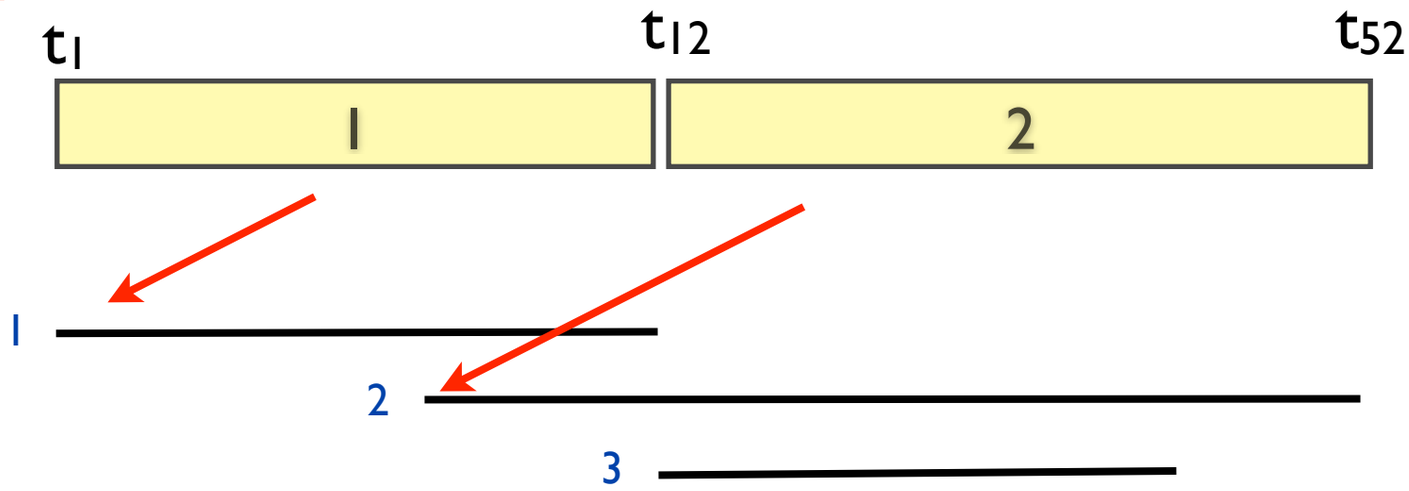


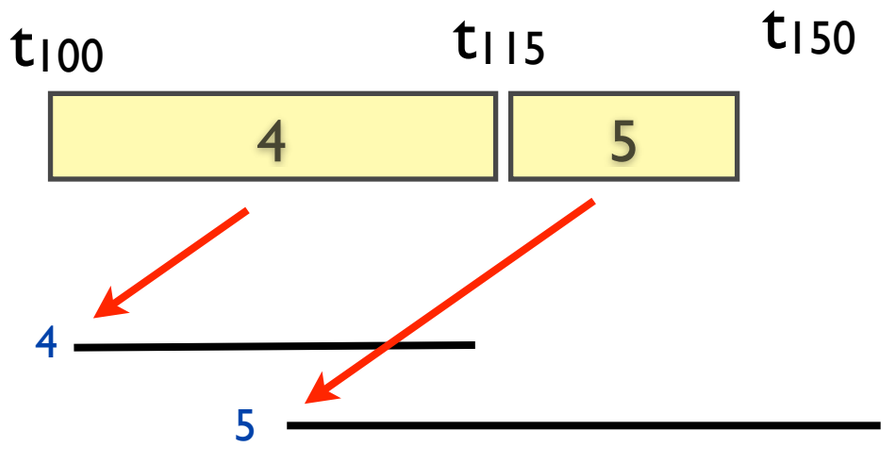
Index Sharding - Impact Lists

- Open - Each shard of a query term opened for access
- Skip - Given a query begin time seek to appropriate offset
- Scan - Read while postings still have overlap with query time interval



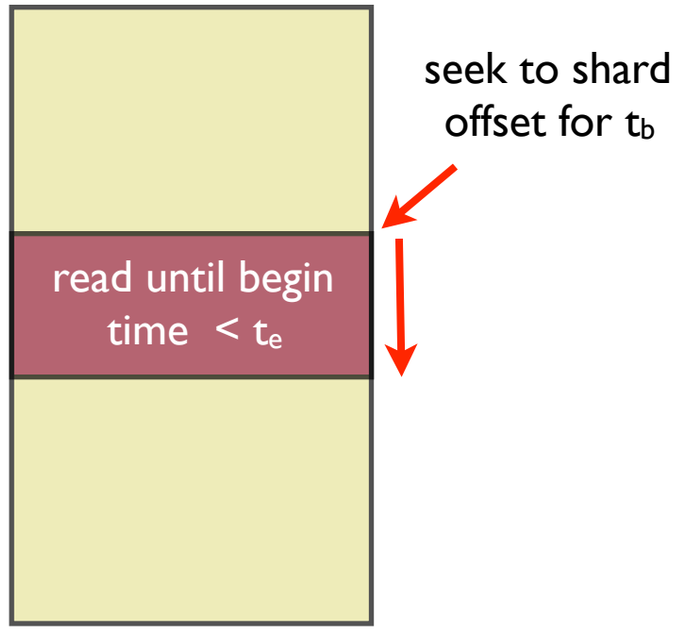
Impact list



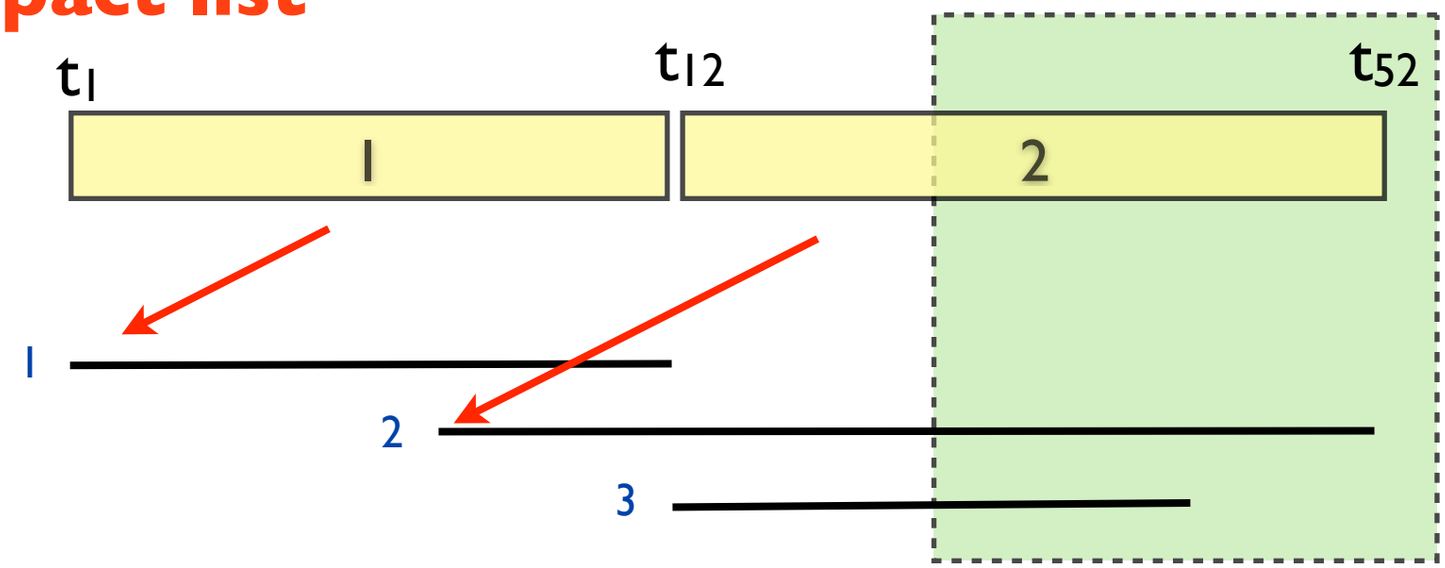


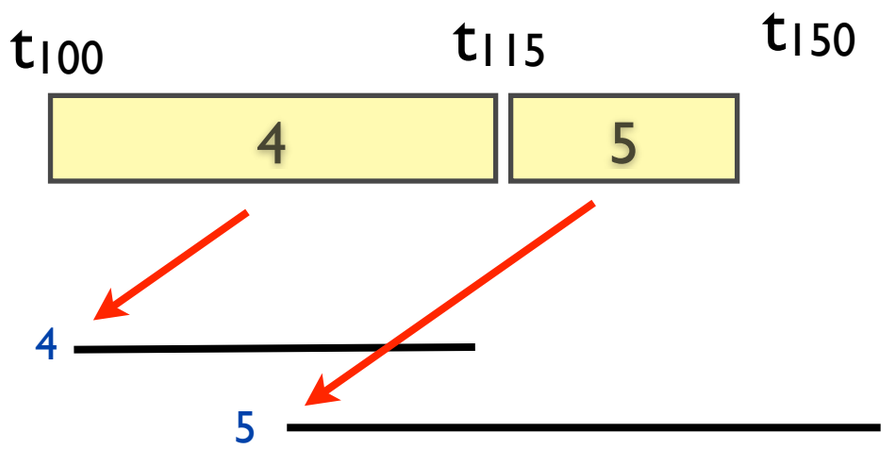
Index Sharding - Impact Lists

- Open - Each shard of a query term opened for access
- Skip - Given a query begin time seek to appropriate offset
- Scan - Read while postings still have overlap with query time interval



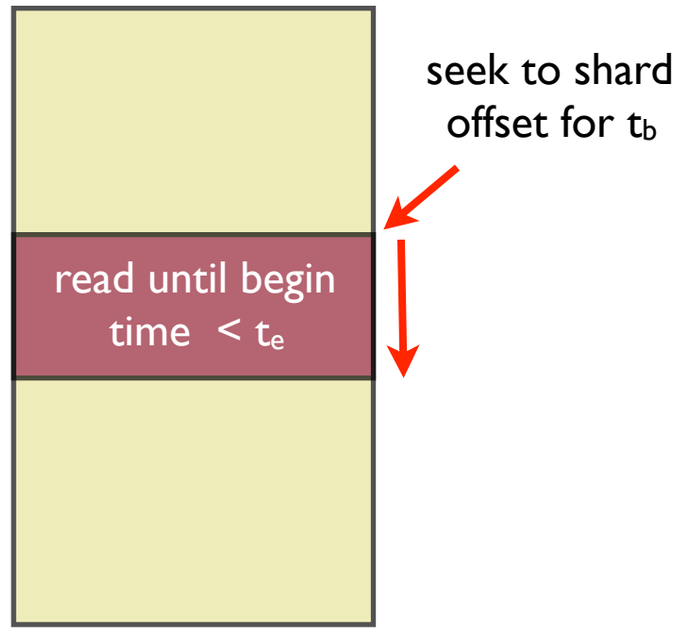
Impact list



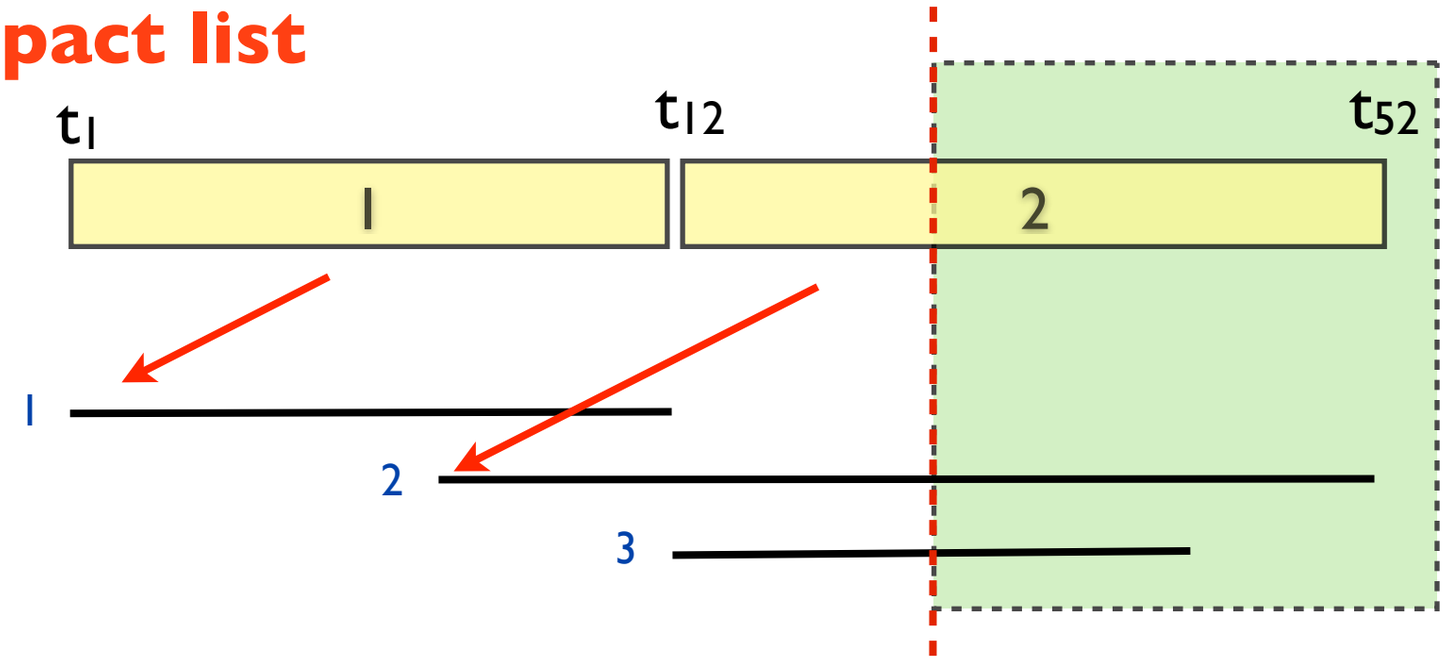


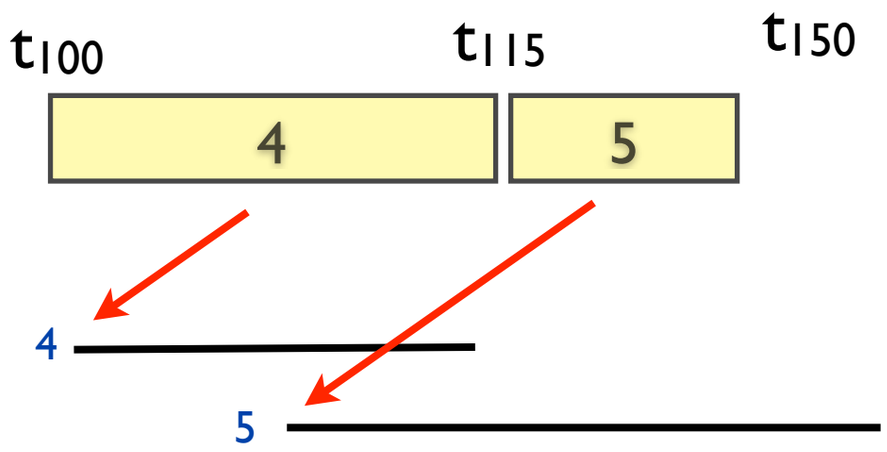
Index Sharding - Impact Lists

- Open - Each shard of a query term opened for access
- Skip - Given a query begin time seek to appropriate offset
- Scan - Read while postings still have overlap with query time interval



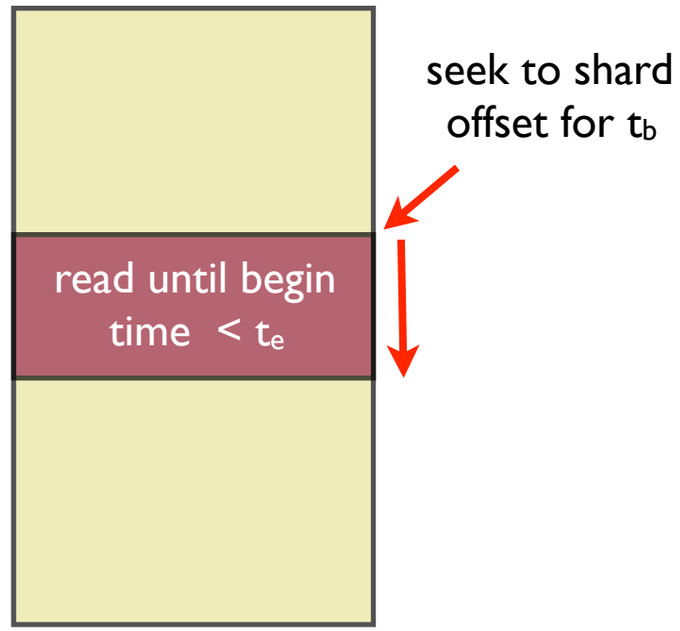
Impact list



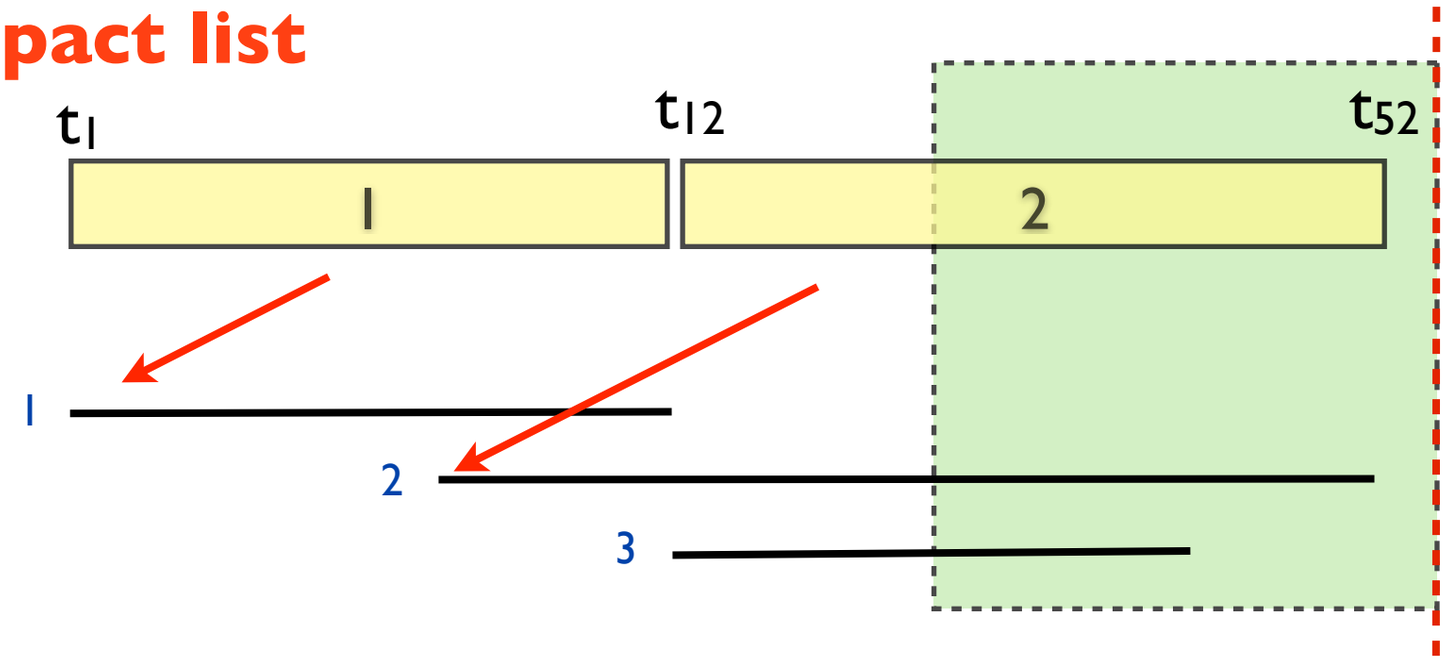


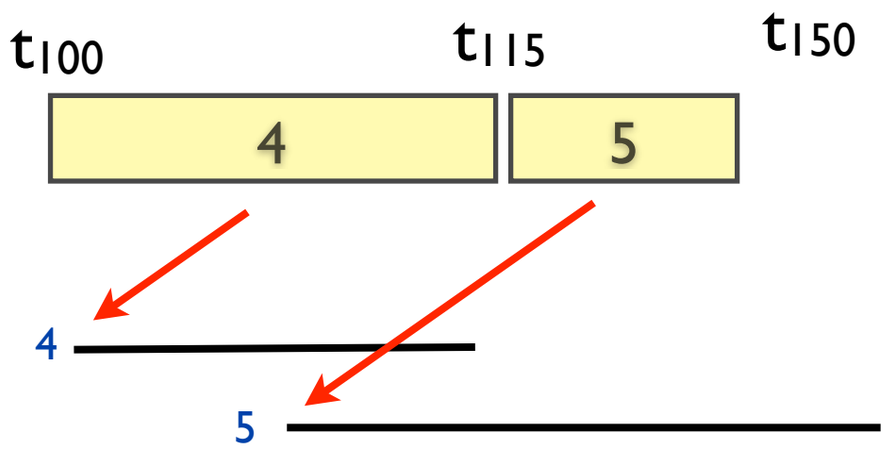
Index Sharding - Impact Lists

- Open - Each shard of a query term opened for access
- Skip - Given a query begin time seek to appropriate offset
- Scan - Read while postings still have overlap with query time interval



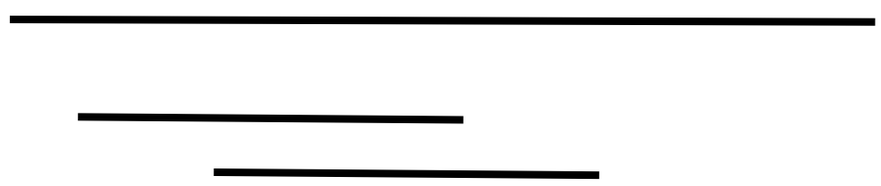
Impact list





Index Sharding - Staircase Property

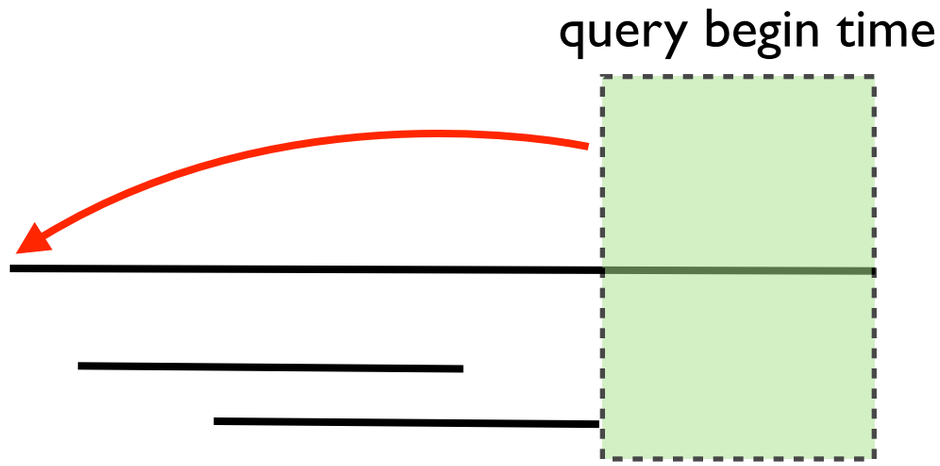
- Wasted reads are processed but do not overlap with the query time interval
- Staircase property in a shard
  - Intervals arranged in begin time order
  - No interval completely subsumes another interval
- Eliminates wasted reads



wasted reads due to subsumption

Index Sharding - Staircase Property

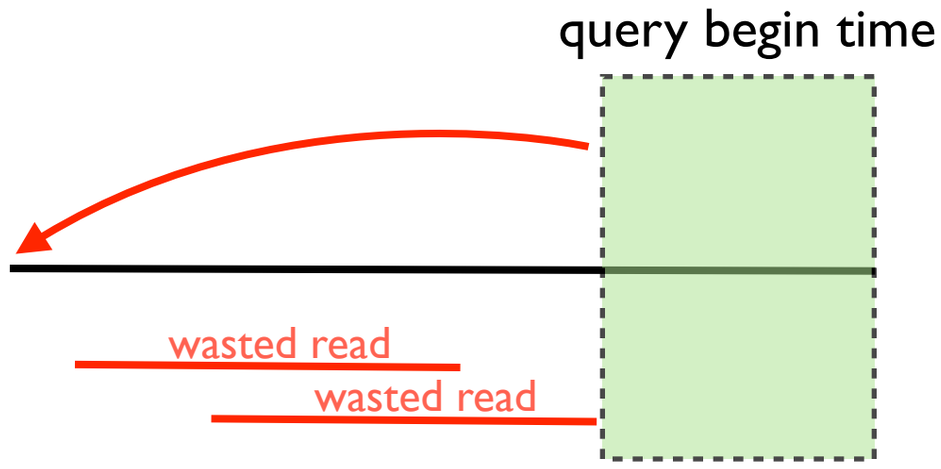
- Wasted reads are processed but do not overlap with the query time interval
- Staircase property in a shard
  - Intervals arranged in begin time order
  - No interval completely subsumes another interval
- Eliminates wasted reads



wasted reads due to subsumption

Index Sharding - Staircase Property

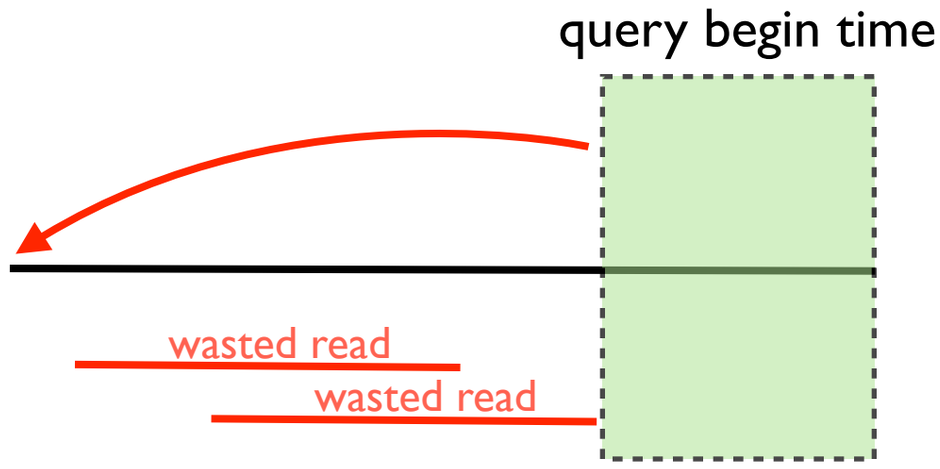
- Wasted reads are processed but do not overlap with the query time interval
- Staircase property in a shard
  - Intervals arranged in begin time order
  - No interval completely subsumes another interval
- Eliminates wasted reads



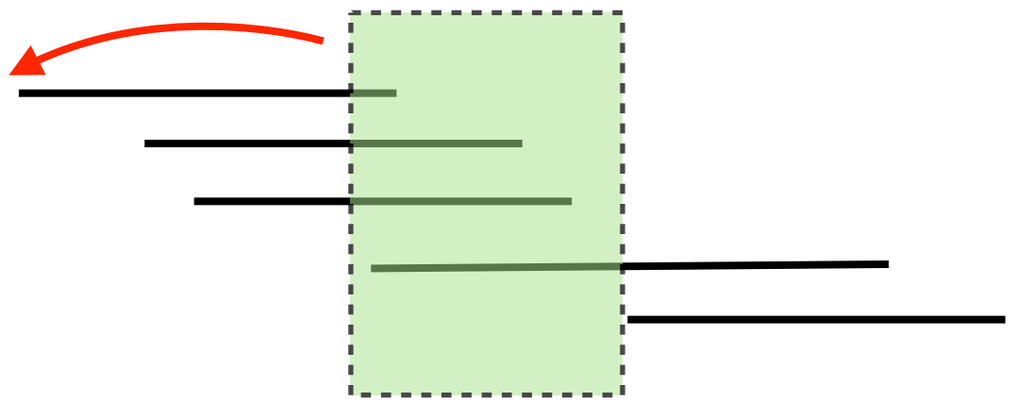
wasted reads due to subsumption

Index Sharding - Staircase Property

- Wasted reads are processed but do not overlap with the query time interval
- Staircase property in a shard
  - Intervals arranged in begin time order
  - No interval completely subsumes another interval
- Eliminates wasted reads



wasted reads due to subsumption



staircase property

Index Sharding - Idealized Sharding

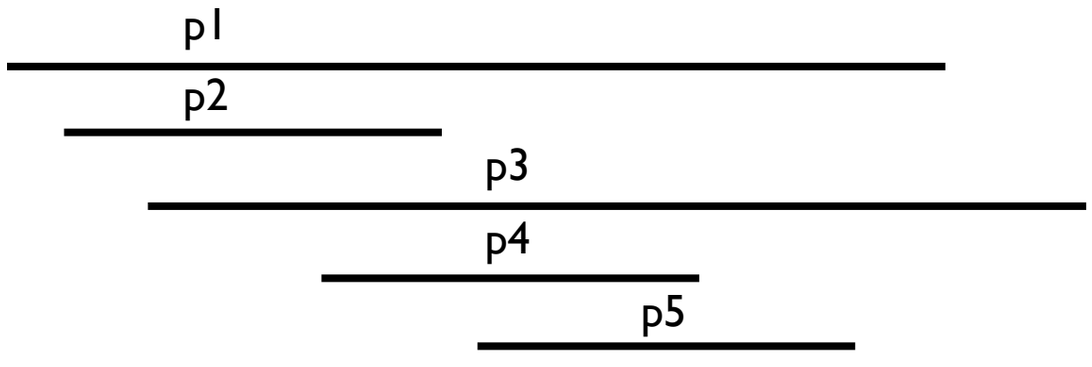
- Staircase property eliminates sequential accesses of postings non-overlapping with query time interval
- Minimizing number of shards is essential in minimizing number of random accesses

- Input : Set of postings/intervals corresponding to a postings list
- Problem Statement : Minimize the number of shards where each shard exhibits the staircase property

Greedy Algorithm exists which is proven to be optimal

Index Sharding - Idealized Sharding

Input :



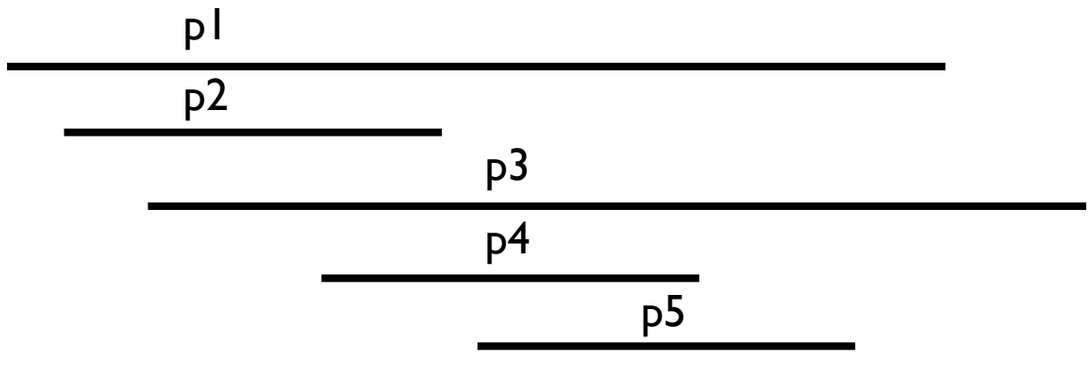
- Postings arrive in begin time order
- For each posting chose a shard which
  - does not violate staircase prop.
  - has min. end time difference

  append posting to the end of chosen shard

- Runtime complexity O(n log n)

Index Sharding - Idealized Sharding

Input :



Shard 1

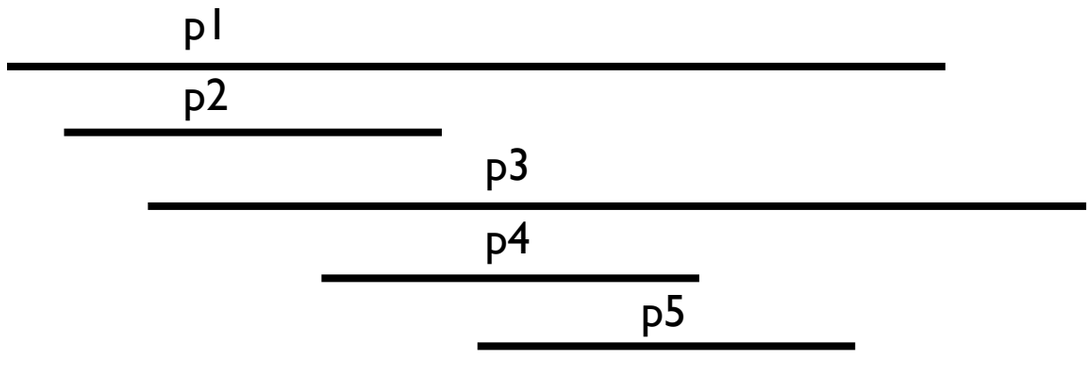
- Postings arrive in begin time order
- For each posting chose a shard which
  - does not violate staircase prop.
  - has min. end time difference

append posting to the end of chosen shard

- Runtime complexity O(n log n)

Index Sharding - Idealized Sharding

Input :



Shard 1

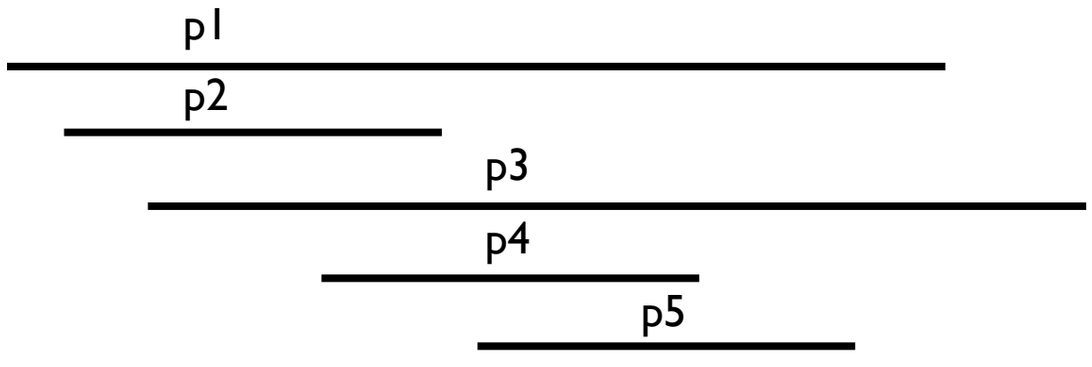
- Postings arrive in begin time order
- For each posting chose a shard which
  - does not violate staircase prop.
  - has min. end time difference

append posting to the end of chosen shard

- Runtime complexity O(n log n)

Index Sharding - Idealized Sharding

Input :



Shard 1

Shard 2

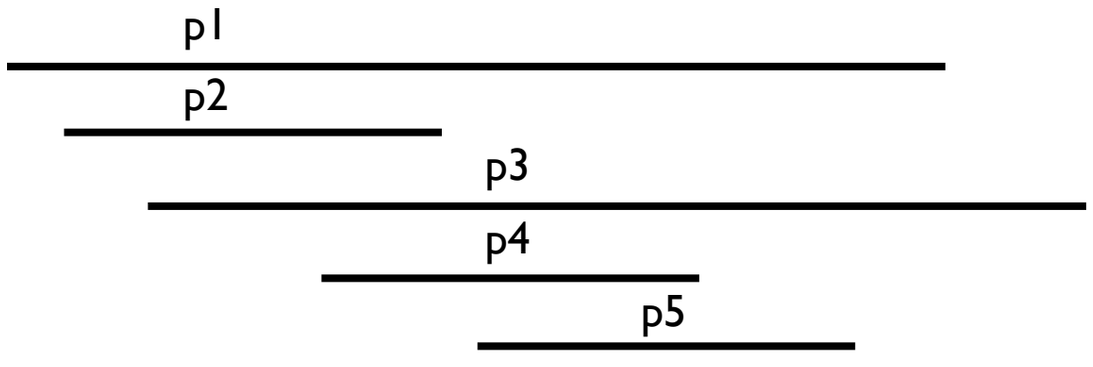
- Postings arrive in begin time order
- For each posting chose a shard which
  - does not violate staircase prop.
  - has min. end time difference

append posting to the end of chosen shard

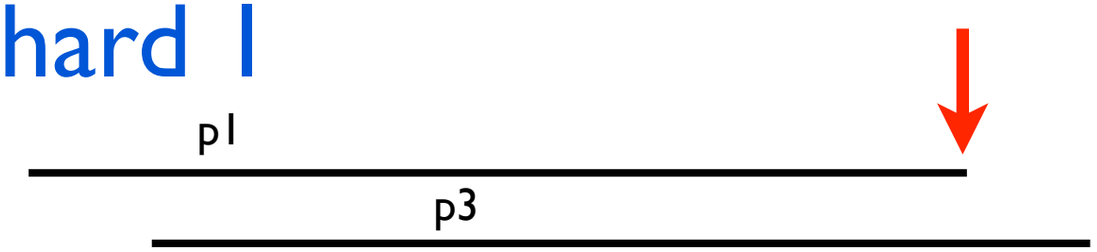
- Runtime complexity O(n log n)

Index Sharding - Idealized Sharding

Input :



Shard 1



Shard 2

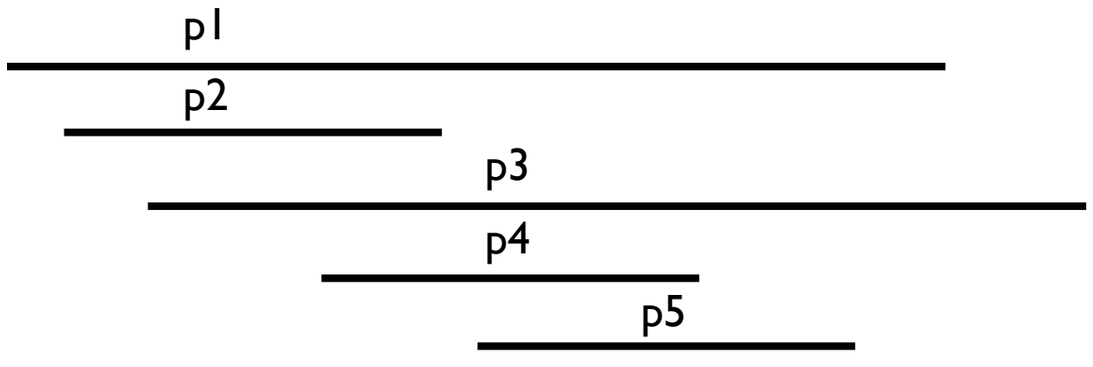
- Postings arrive in begin time order
- For each posting chose a shard which
  - does not violate staircase prop.
  - has min. end time difference

append posting to the end of chosen shard

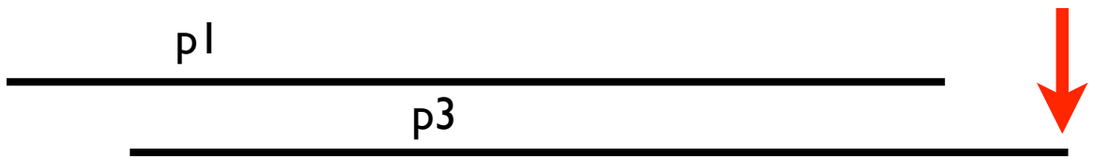
- Runtime complexity O(n log n)

Index Sharding - Idealized Sharding

Input :



Shard 1



Shard 2

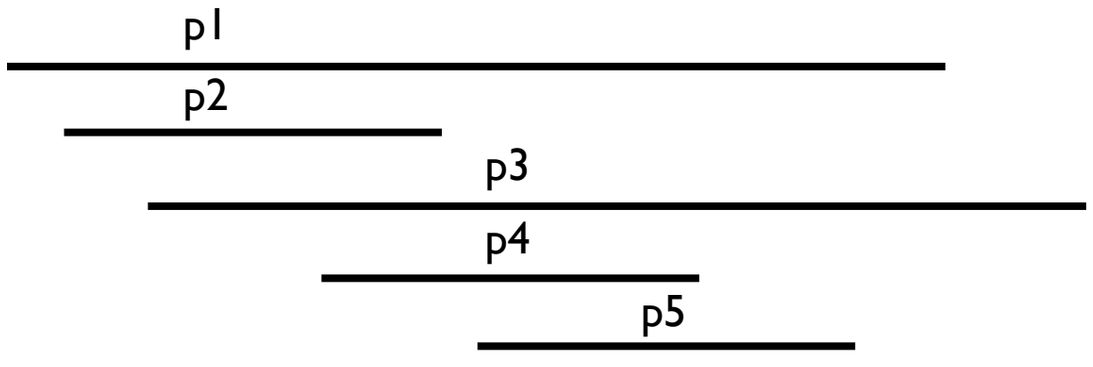
- Postings arrive in begin time order
- For each posting chose a shard which
  - does not violate staircase prop.
  - has min. end time difference

append posting to the end of chosen shard

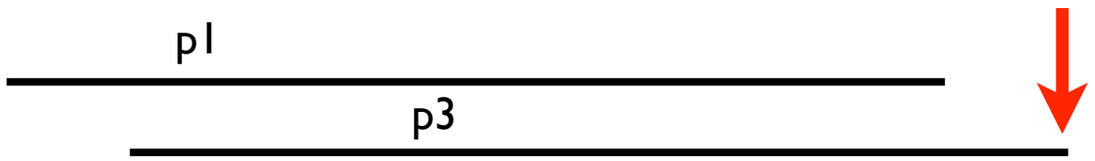
- Runtime complexity O(n log n)

Index Sharding - Idealized Sharding

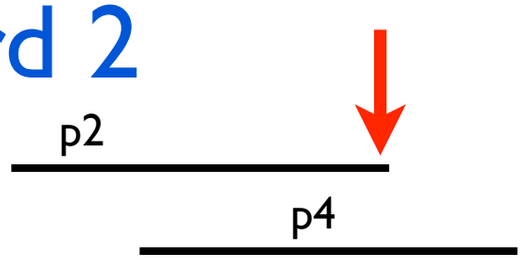
Input :



Shard 1



Shard 2



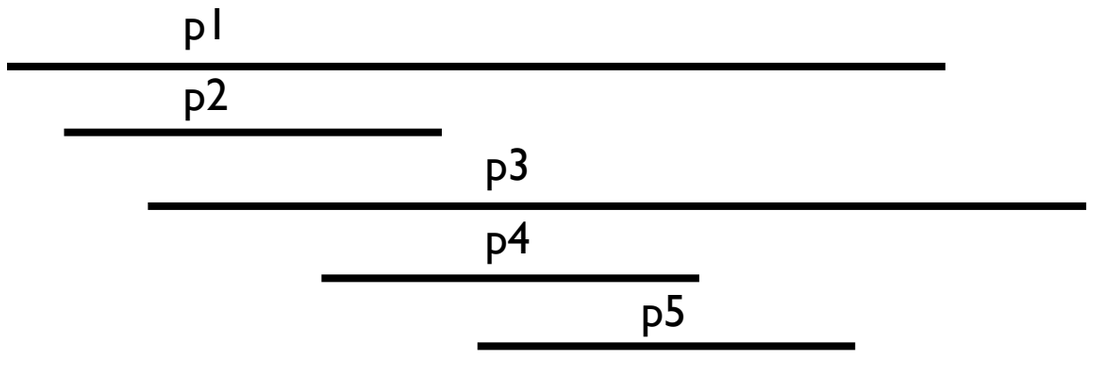
- Postings arrive in begin time order
- For each posting chose a shard which
  - does not violate staircase prop.
  - has min. end time difference

append posting to the end of chosen shard

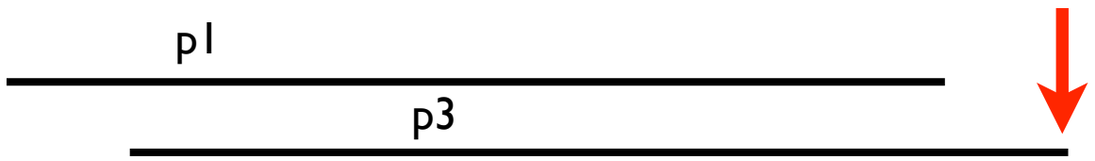
- Runtime complexity O(n log n)

Index Sharding - Idealized Sharding

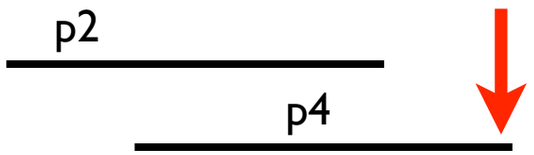
Input :



Shard 1



Shard 2



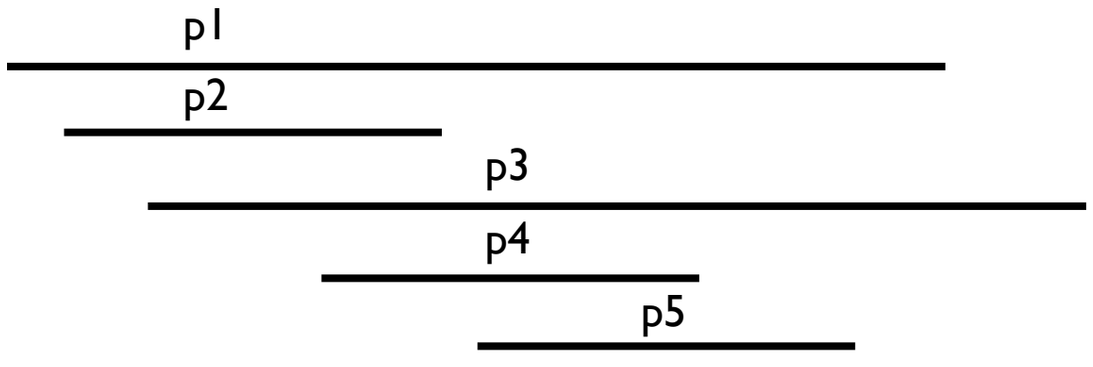
- Postings arrive in begin time order
- For each posting chose a shard which
  - does not violate staircase prop.
  - has min. end time difference

append posting to the end of chosen shard

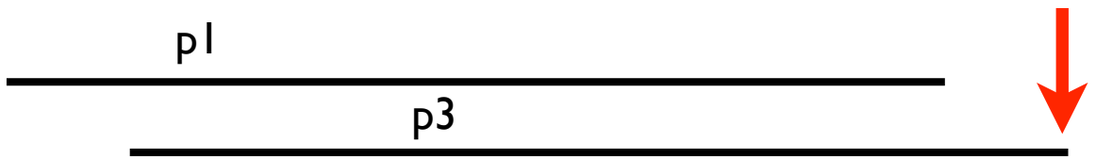
- Runtime complexity O(n log n)

Index Sharding - Idealized Sharding

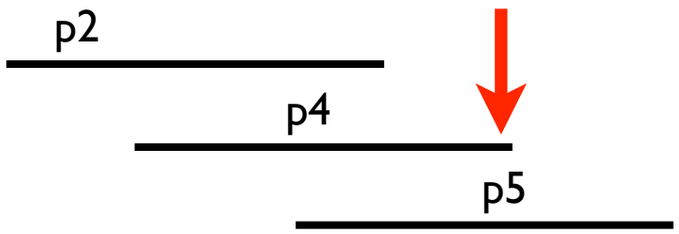
Input :



Shard 1



Shard 2



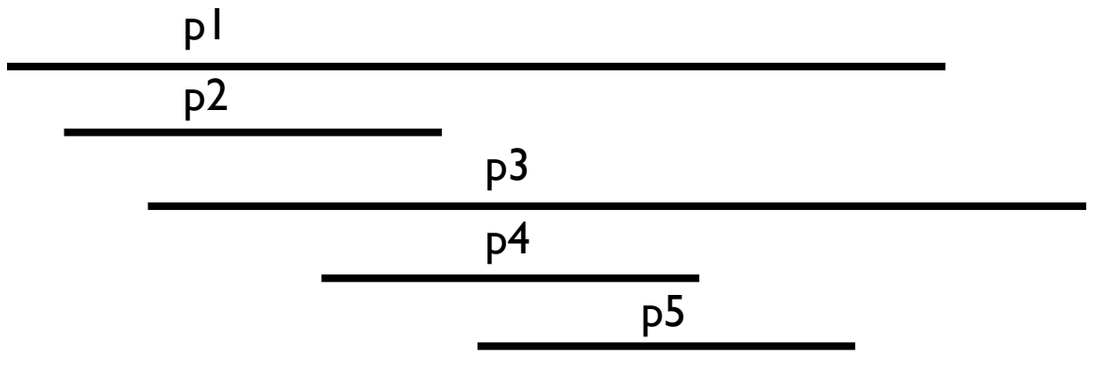
- Postings arrive in begin time order
- For each posting chose a shard which
  - does not violate staircase prop.
  - has min. end time difference

append posting to the end of chosen shard

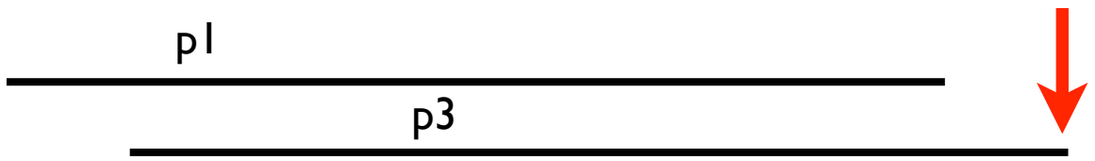
- Runtime complexity O(n log n)

Index Sharding - Idealized Sharding

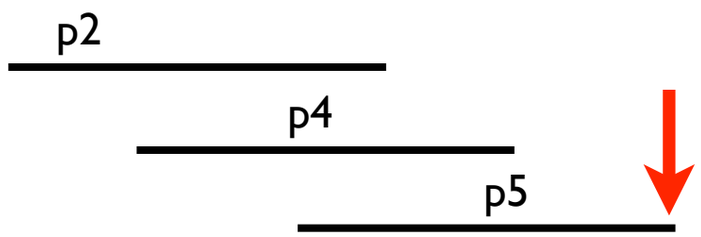
Input :



Shard 1



Shard 2

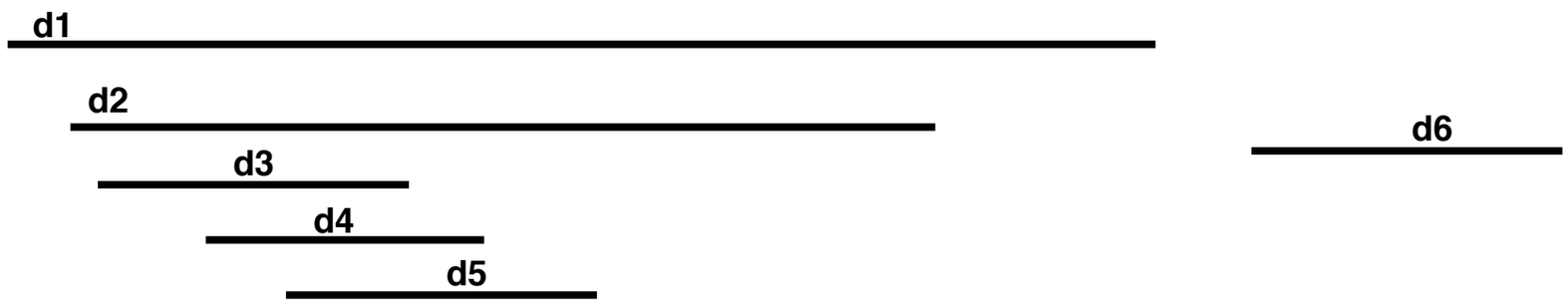


- Postings arrive in begin time order
- For each posting chose a shard which
  - does not violate staircase prop.
  - has min. end time difference

append posting to the end of chosen shard

- Runtime complexity O(n log n)

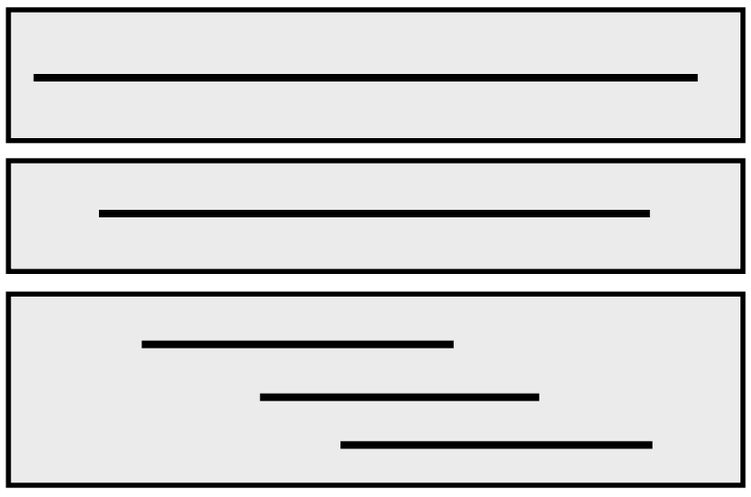
Exercise



- What is the Idealized sharding for the given input ?
- What are Impact lists for each idealized shard ?
- What is the worst case (in the number of shards) example for idealised sharding ?

Index Sharding - Challenges

Idealized Sharding

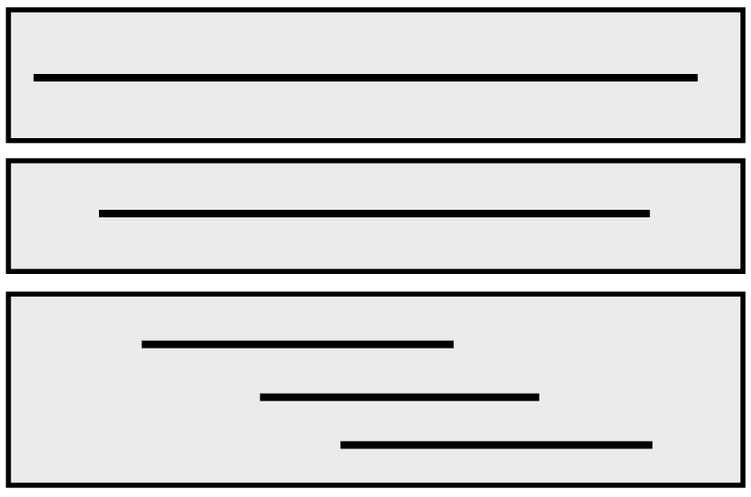


- No wasted reads
- Many shards
- QP suffers. Why ?

- Random accesses (RA) are typically much more expensive than sequential accesses (SA)

Index Sharding - Challenges

Idealized Sharding



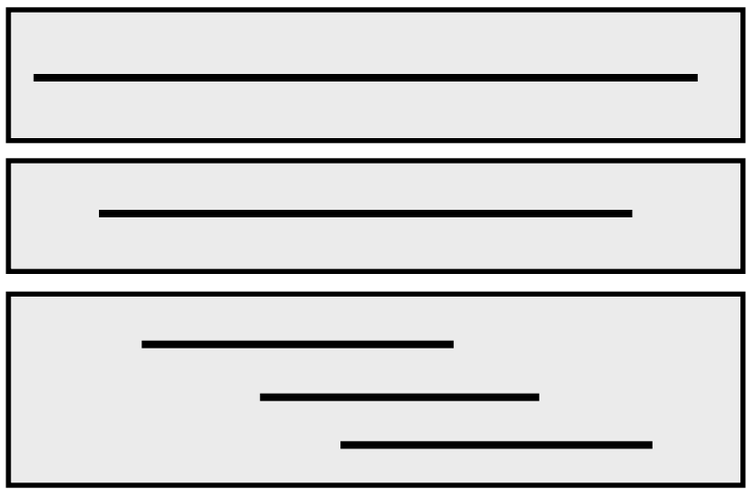
- No wasted reads
- Many shards
- QP suffers. Why ?

More shards the more the random accesses to disk

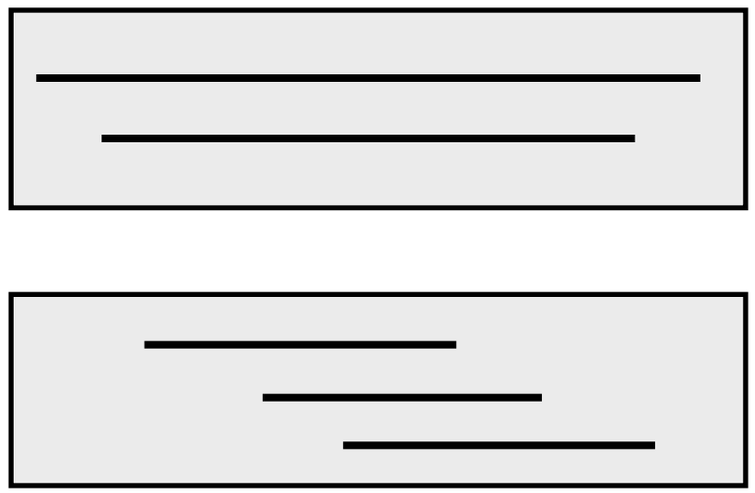
- Random accesses (RA) are typically much more expensive than sequential accesses (SA)
- Allow wasted reads to balance SA and RA

Index Sharding - Challenges

Idealized Sharding



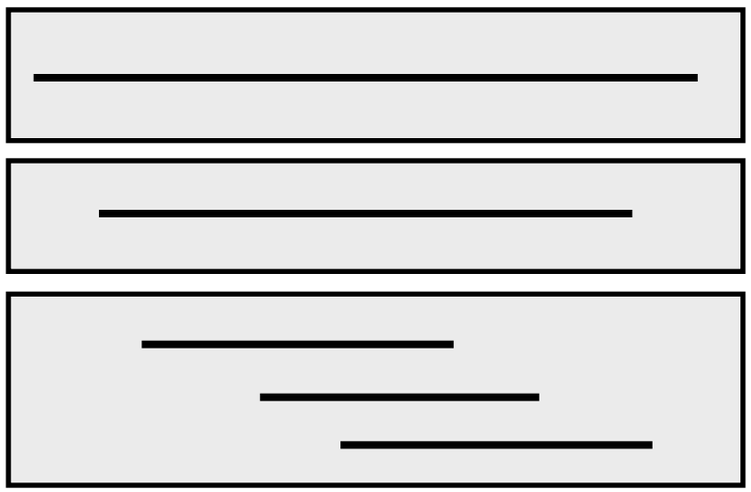
Relaxing the Sharding



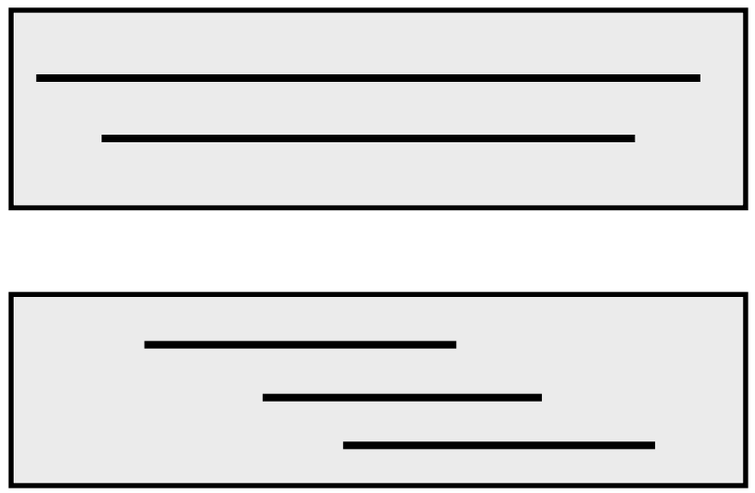
- Random accesses (RA) are typically much more expensive than sequential accesses (SA)

Index Sharding - Challenges

Idealized Sharding



Relaxing the Sharding

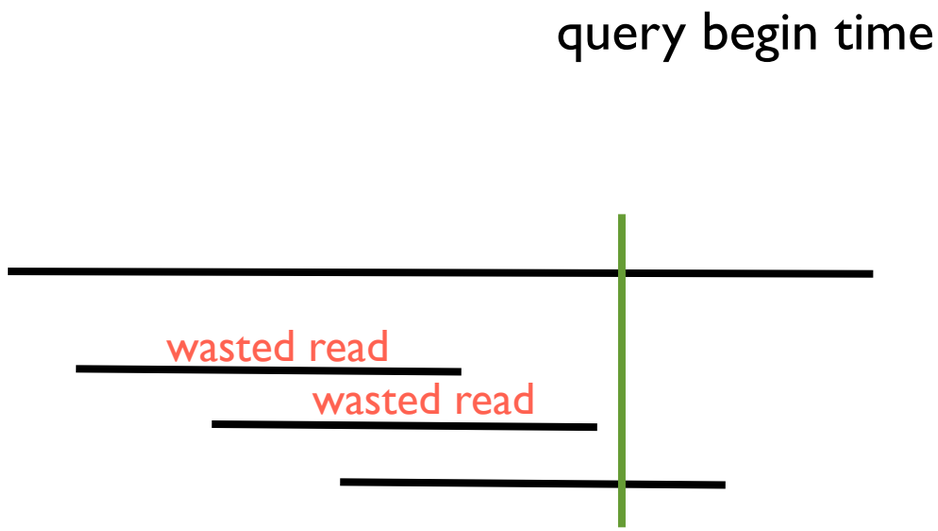


More shards the more the random accesses to disk

- Random accesses (RA) are typically much more expensive than sequential accesses (SA)
- Allow wasted reads to balance SA and RA

Bounded Subsumption

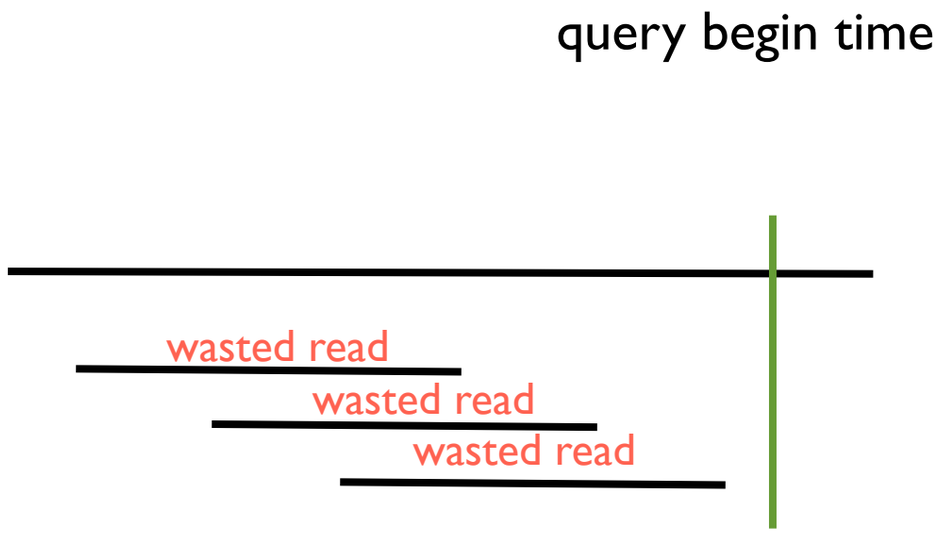
- Balancing sequential and random accesses
- Bounded subsumption: no more than η wasted reads for any query begin time



- Bounded Subsumption Problem: Minimize number of shards s.t. each shard has bounded subsumption

Bounded Subsumption

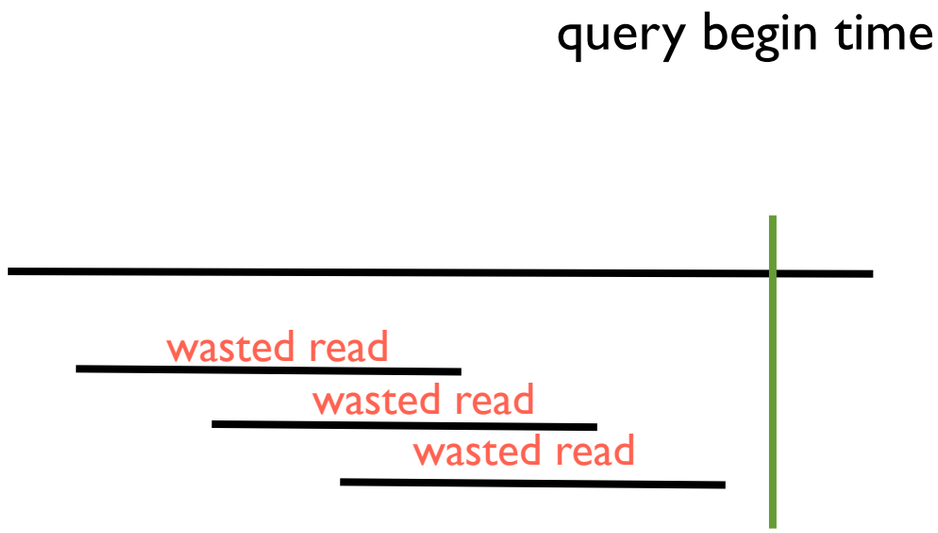
- Balancing sequential and random accesses
- Bounded subsumption: no more than η wasted reads for any query begin time



- Bounded Subsumption Problem: Minimize number of shards s.t. each shard has bounded subsumption

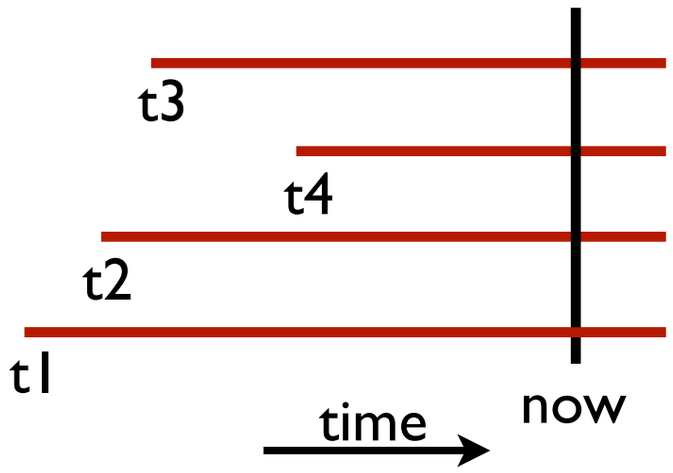
Bounded Subsumption

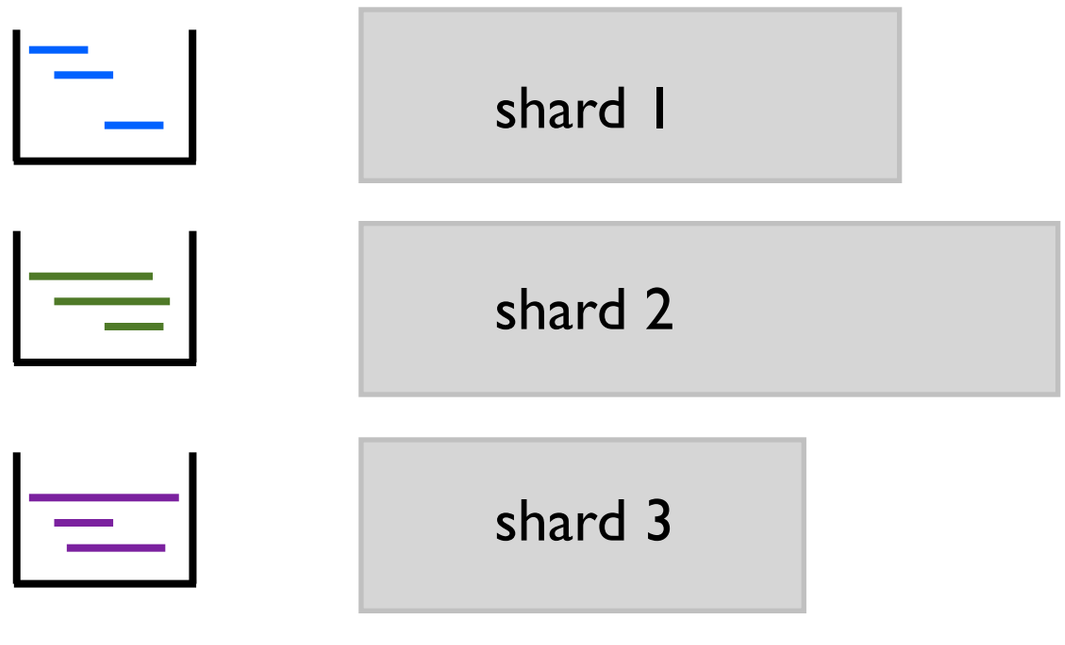
- Balancing sequential and random accesses
- Bounded subsumption: no more than η wasted reads for any query begin time



Can we create shards solving the bounded subsumption problem?

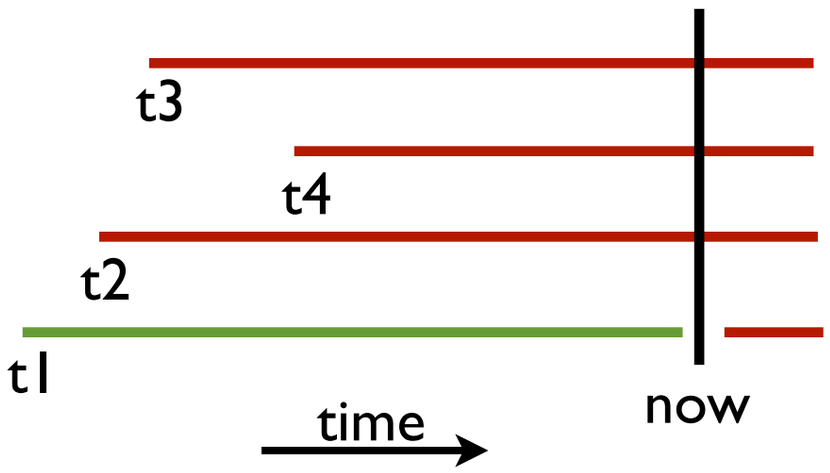
Incremental Sharding

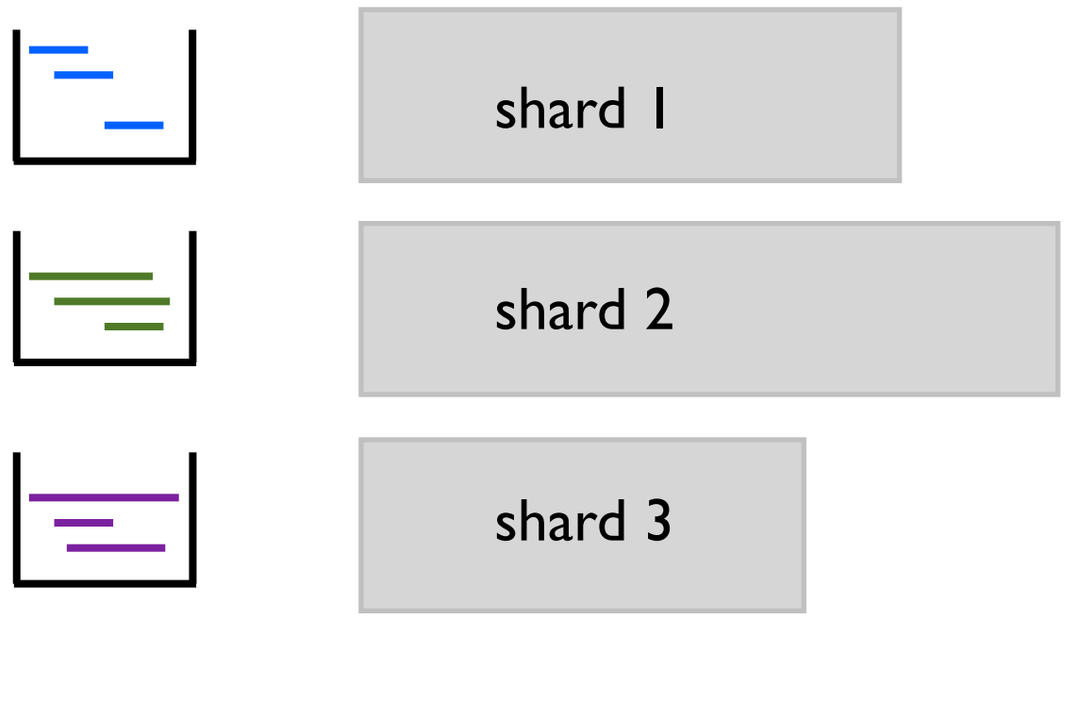




- Algorithm assigns incoming posting to a shard
- Posting inserted into shard buffer maintaining begin time order
- Top posting popped and appended to the shard end

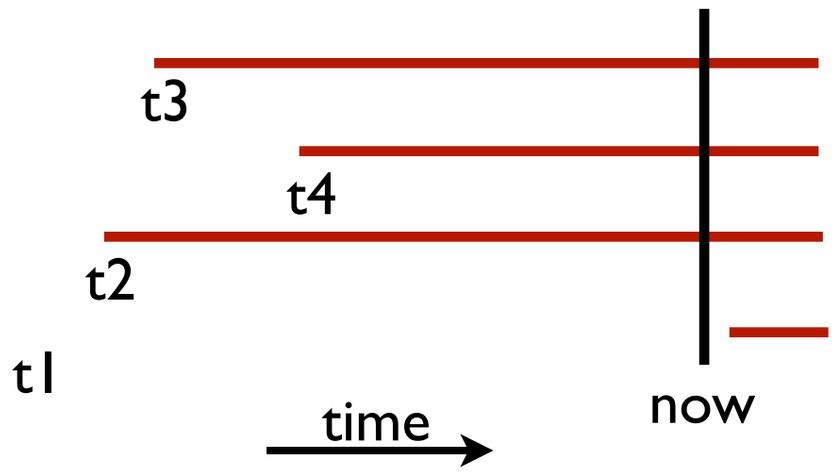
Incremental Sharding

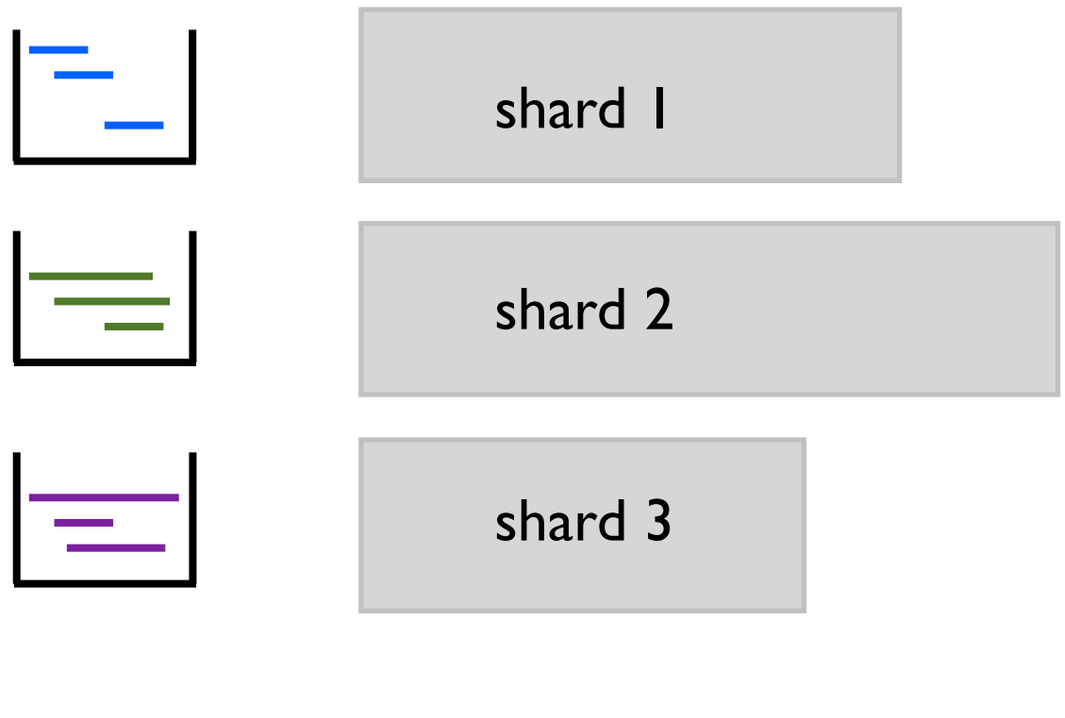




- Algorithm assigns incoming posting to a shard
- Posting inserted into shard buffer maintaining begin time order
- Top posting popped and appended to the shard end

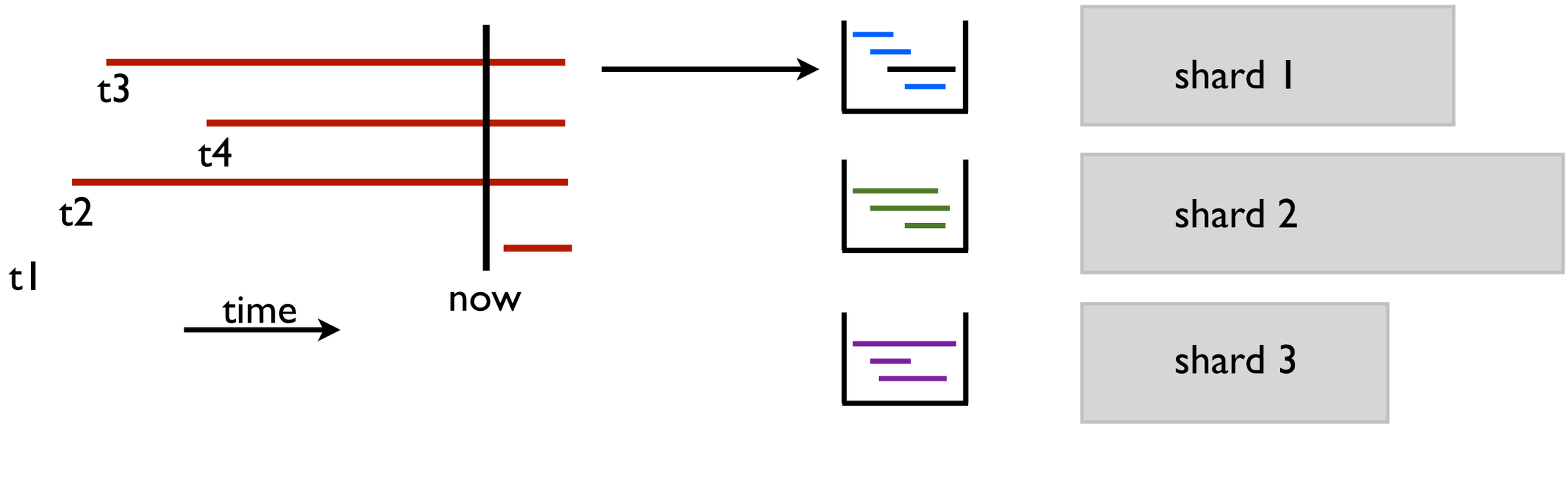
Incremental Sharding





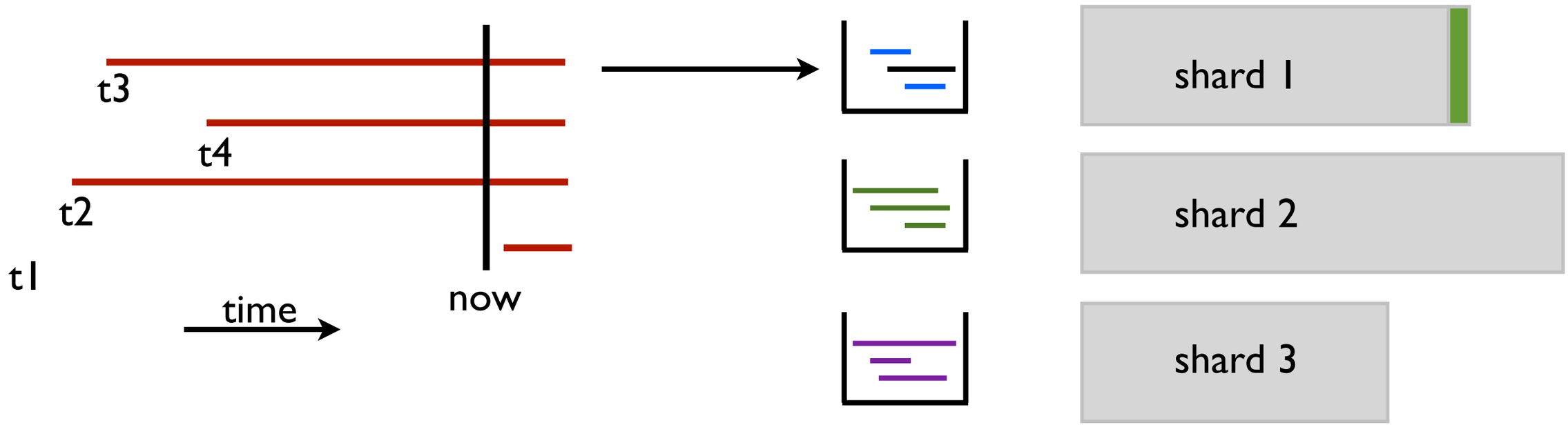
- Algorithm assigns incoming posting to a shard
- Posting inserted into shard buffer maintaining begin time order
- Top posting popped and appended to the shard end

Incremental Sharding



- Algorithm assigns incoming posting to a shard
- Posting inserted into shard buffer maintaining begin time order
- Top posting popped and appended to the shard end

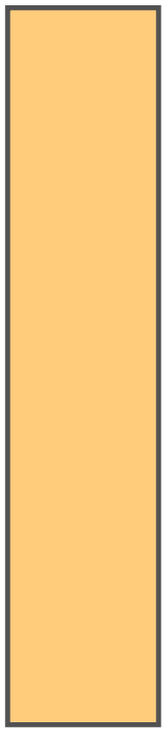
Incremental Sharding



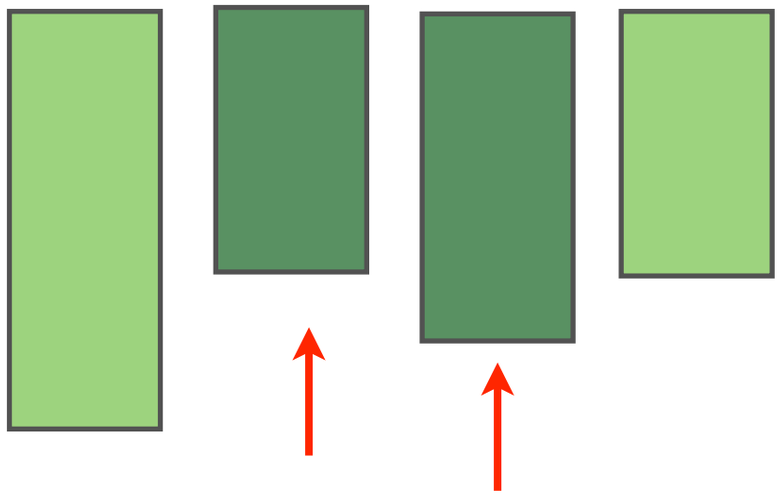
- Algorithm assigns incoming posting to a shard
- Posting inserted into shard buffer maintaining begin time order
- Top posting popped and appended to the shard end

Putting Things Together

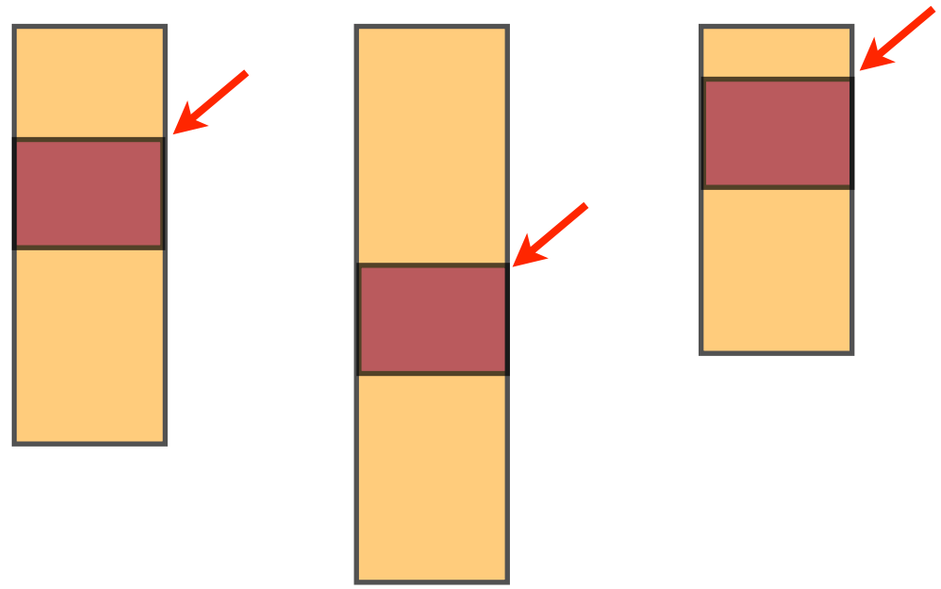
Original Postings List



Vertical Partitions



Horizontal Partitions



- Start with initial posting list and partition them vertically or horizontally
- Given a query time interval for each term
  - Access either subset of entire lists or part of all lists
- Intersect or Union the results for all query terms

Open Source Full-text Indexing Software

Lemur

elasticsearch.

:::MG4J:

Terrier

Lucene

Apache Solr

Zettair
an open source search engine
written in C

Task 1

- Given the Temporalia collection
  - Document collections ~ 50k documents
  - Query workload
- Index the doc collection both
  - Text
  - Time - publication dates + temporal references
- Should support temporal queries “earthquakes” @ 1998
  - Result documents should have documents either published in 1998 or contain references which overlap with 1998

Task 1 - Doc Collection

```
<doc id=lk-20130223040102_592>
<meta-info>
<tag name="host">www.spiegel.de</tag>
<tag name="data">2013-02-22</tag>
<tag name="url">http://www.spiegel.de/international/business/eu-widens-libor-scanda
l-investigation-and-threatens-heavy-fines-a-884948.html#ref=rss</tag>
<tag name="sourcerss">http://www.spiegel.de/international/index.rss</tag>
<tag name="title">EU Widens LIBOR Scandal Investigation and Threatens Heavy Fines</
tag>
</meta-info>
<text>
...
</text>
```

```
:"E:ORGANIZATION:CORPORATION">"The King's Speech"</E> was a
is clear that the <E type="E:ORG_DESC:POLITICAL">party</E>
r</T> instead of <T val="201102">early February</T>.</SE>
```

Task 1 - Queries

```
<topics>
<topic>
        <id>001</id>
        <title>Earthquakes</title>
        <description>I suspect that these days the intensity of harsh weather conditions such as
earthquakes is increased when compared to the past. In order to make sure I need to collect information
on earthquake, their past occurrences, and future forecasts, etc.</description>
        <query_issue_time>Mar 29, 2013 GMT+0:00</query_issue_time>
        <subtopics>
                <subtopic id="001a" type="atemporal">What is an earthquake and how severe it can be?
</subtopic>
                <subtopic id="001p" type="past">What past earthquakes were most deadly?</subtopic>
                <subtopic id="001r" type="recency">What was the latest earthquake in Asia?</subtopic>
                <subtopic id="001f" type="future">What are predictions regarding the occurrence of
earthquakes in the near future?</subtopic>
        </subtopics>
</topic>
```

- Expected finish date: Before the next task is distributed (2 weeks)
- Evaluate the results yourself and meet us after the lecture

References

- Information retrieval: (http://www.ir.uwaterloo.ca/book/)
  Stefan Büttcher, Google Inc. , Charles L. A. Clarke, Univ. of Waterloo,
  Gordon V. Cormack, Univ. of Waterloo

- Managing Gigabytes: by Justin Zobel, Alistair Moffat, Ian Witten

- Indexing Methods for Web Archives: Avishek Anand