Index Creation

Sven Fiergolla

5. April 2018

Übersicht

Einführung

Effiziente Suche über:

- Sammlung von Büchern
- das Web
- andere große Datenmengen

Einführung

Effiziente Suche über:

Sammlung von Büchern
das Web
andere große Datenmengen

zu viel für Main Memory!

Einführung

Typische Systemeigenschaften (stand 2018)

- *clock rate* 2-4 GHz, 4-8 Kerne
- *main memory* 4-32 GHz
- *disk space* $\leq$ 1 TB SSD oder $\geq$ 1 TB HDD

Einführung

Typische Systemeigenschaften (stand 2018)

- *clock rate* 2-4 GHz, 4-8 Kerne
- *main memory* 4-32 GHz
- *disk space* $\leq$ 1 TB SSD oder $\geq$ 1 TB HDD
  - HDD (hard disk drive)
    - *average seek time* zwischen 2 und 10 ms
    - *transfer time* 150 - 300 MB/s

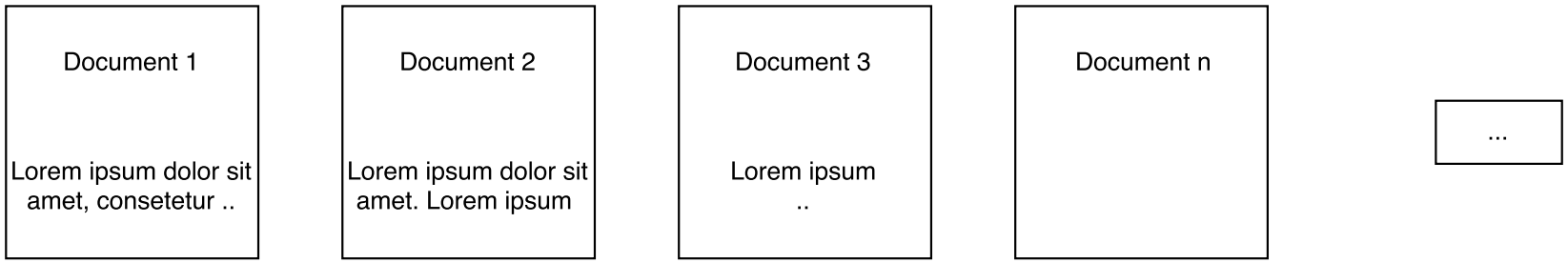
Einführung

Typische Systemeigenschaften (stand 2018)

- *clock rate* 2-4 GHz, 4-8 Kerne
- *main memory* 4-32 GHz
- *disk space* $\leq$ 1 TB SSD oder $\geq$ 1 TB HDD
  - HDD (hard disk drive)
    - *average seek time* zwischen 2 und 10 ms
    - *transfer time* 150 - 300 MB/s
  - SSD (solid state disk)
    - *average seek time* zwischen 0.08 und 0.16 ms
    - *transfer time* Lesen: 545 MB/s, Schreiben: 525 MB/s

hardware constraints

Indizierung einer Sammlung von Daten/Dokumenten auf der Festplatte



Zugriffszeit auf Festplatte als Bottelneck

Index Creation

geeignete Datenstruktur um Zugriff auf die Festplatte zu minimieren

TermID / DocID	
Lorem	1
ipsum	1
consetetur	1
sadipscing	1
amet	1
...	1
Lorem	2
ipsum	2
dolor	2
sit	2
amet	2
...	2

Index Creation

geeignete Datenstruktur um Zugriff auf die Festplatte zu minimieren

TermID / DocID	
Lorem	1
ipsum	1
consetetur	1
sadipscing	1
amet	1
ipsum	1
Lorem	2
ipsum	2
dolor	2
sit	2
amet	2
...	2

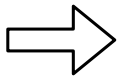
TermID / DocID	
amet	1
amet	2
consetetur	1
dolor	2
ipsum	1
ipsum	2
Lorem	1
Lorem	2
sadipscing	1
sit	2
...	...

Index Creation

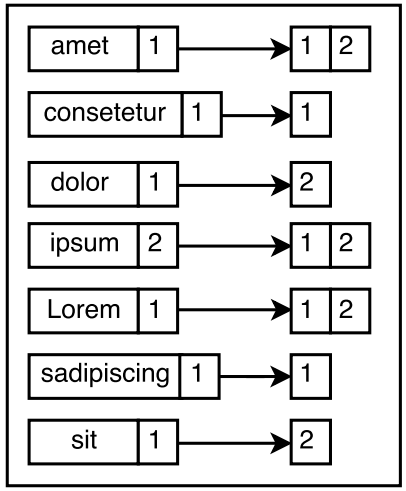
geeignete Datenstruktur um Zugriff auf die Festplatte zu minimieren

TermID / DocID	
Lorem	1
ipsum	1
consetetur	1
sadipscing	1
amet	1
ipsum	1
Lorem	2
ipsum	2
dolor	2
sit	2
amet	2
...	2

TermID / DocID	
amet	1
amet	2
consetetur	1
dolor	2
ipsum	1
ipsum	2
Lorem	1
Lorem	2
sadipscing	1
sit	2
...	...



TermID, doc Frequenz / Postings



Index Creation - hardware constraints

Reuters-RCV1 besitzt 100.000.000 Terme...

Sortieren dieser Terme von einer Festplatte:

Index Creation - hardware constraints

Reuters-RCV1 besitzt 100.000.000 Terme...

Sortieren dieser Terme von einer Festplatte:

- Annahme
  - $T \cdot log_2(T)$ Vergleiche
  - 2 Zugriffe auf die Hard Drive zum Vergleichen
  - average seek time 5 ms

Index Creation - hardware constraints

Reuters-RCV1 besitzt 100.000.000 Terme...

Sortieren dieser Terme von einer Festplatte:

- Annahme
  - $T \cdot log_2(T)$ Vergleiche
  - 2 Zugriffe auf die Hard Drive zum Vergleichen
  - average seek time 5 ms
- $(100.000.000 \cdot log_2(100.000.000)) \cdot (5 \cdot 10^{-3})$ Sekunden
- $2.6575424759... \cdot 10^7$ Sekunden
- 307.59 Tage

Blocked sort-based indexing (BSI)

Lösung:

- Sammlung von Dokumenten in einzelne Blocks unterteilen
- Index über einzelne Blöcke erstellen
- Teilindizes mergen

Blocked sort-based indexing (BSI)

Lösung:

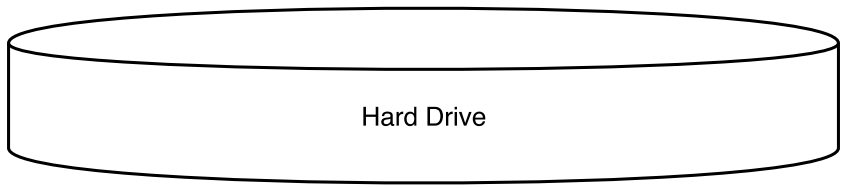
- Sammlung von Dokumenten in einzelne Blocks unterteilen
- Index über einzelne Blöcke erstellen
- Teilindizes mergen

n = 0;
while *all documents have not been processed* **do**
  n = n + 1;
  block = ParseNextBlock();
  BSBI-INVERT(block);
  WriteBlockToDisk(block, f_n);
end
MergeBlocks($f_1, \cdots, f_n; f_{merged}$);

Algorithm 2: BSI Algorithmus

Blocked sort-based indexing (BSI) - merging Blocks

Zusammenführen von Postings Lists bei block sort-based Indexing

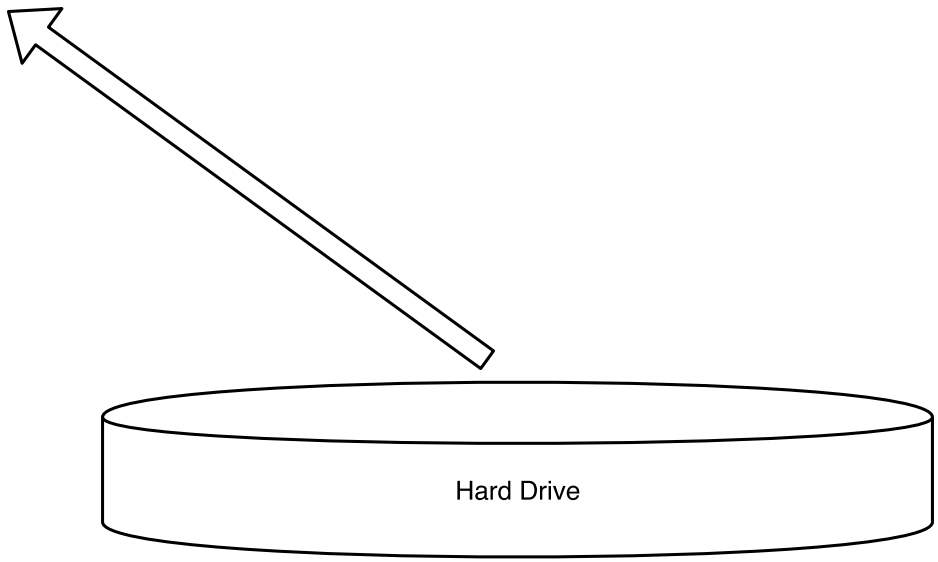


Blocked sort-based indexing (BSI) - merging Blocks

Zusammenführen von Postings Lists bei block sort-based Indexing

amet	d1,d2
dolor	d2
Lorem	d1
ipsum	d2

consetetur	d3, d4
Lorem	d3
ipsum	d3
sit	d4

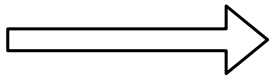


Blocked sort-based indexing (BSI) - merging Blocks

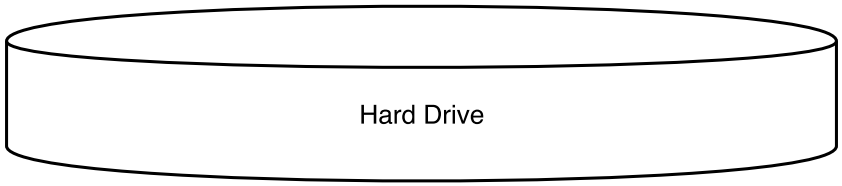
Zusammenführen von Postings Lists bei block sort-based Indexing

amet	d1,d2
dolor	d2
Lorem	d1
ipsum	d2

consetetur	d3, d4
Lorem	d3
ipsum	d3
sit	d4



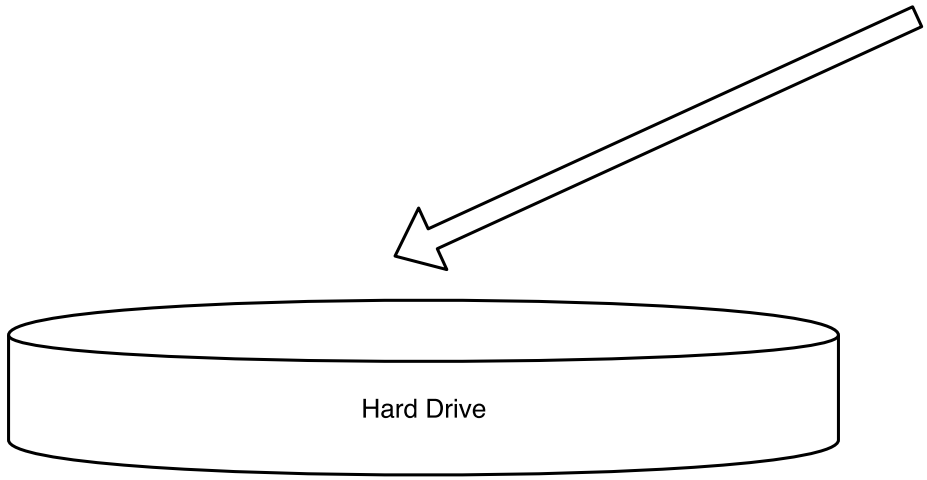
amet	d1,d2
consetetur	d3,d4
dolor	d2
Lorem	d1, d3
ipsum	d2, d3
sit	d4



Blocked sort-based indexing (BSI) - merging Blocks

Zusammenführen von Postings Lists bei block sort-based Indexing

amet	d1,d2
consetetur	d3,d4
dolor	d2
Lorem	d1, d3
ipsum	d2, d3
sit	d4



Blocked sort-based indexing (BSI) - Fazit

Fazit zu BSBI:

- Zeitkompplexität: $\Theta(T \cdot log(T))$
  - das Sortieren hat die höchste Komplexität
  - das Parsen und Mergen der Blocks ist jedoch in der Regel am Zeitaufwendigsten
- Datenstruktur für Mapping zwischen termen und termID's muss in Main Memory liegen
  - kann für sehr große Datenmengen auch Server überlasten

Single-pass in-memory indexing (SPIMI)

- einzelne dictionaries für jeden Block
  - keine Datenstruktur für das Mapping von termen und termID's
- kein Sortieren der einzelnen Blöcke
  - Postings in der Reihenfolge ihres Vorkommens in die Postingslist aufnehmen
  - PostingsList sollte jedoch sortiert werden, da dann auf Sortieren/Suchen beim mergen der Blöcke verzichtet werden kann

Single-pass in-memory indexing (SPIMI)

todo ablauf visualisieren

Single-pass in-memory indexing (SPIMI)

```
SPIMI-invert(TokenStream)
outputFile = new HashFile();
dictionary = new HashFile();
while free memory available do
    token = next(TokenStream);
    if term(token) ∉ dictionary then
        PostingsList = AddToDictionary(dictionary, term(token));
    else
        PostingsList = GetPostingsList(dictionary, term(token));
    end
    if full(PostingsList) then
        PostingsList = DoublePostingsList(dictionary, term(token));
    end
    AddToPostingsList(PostingsList, docID(token));
    SortedTerms = SortTerms(dictionary);
    WriteBlockToDisk(SortedTerms, dictionary, OutputFile);
end
```

Algorithm 3: SPIMI-invert Algorithmus

Single-pass in-memory indexing (SPIMI)

Vorteile gegenüber BSI:

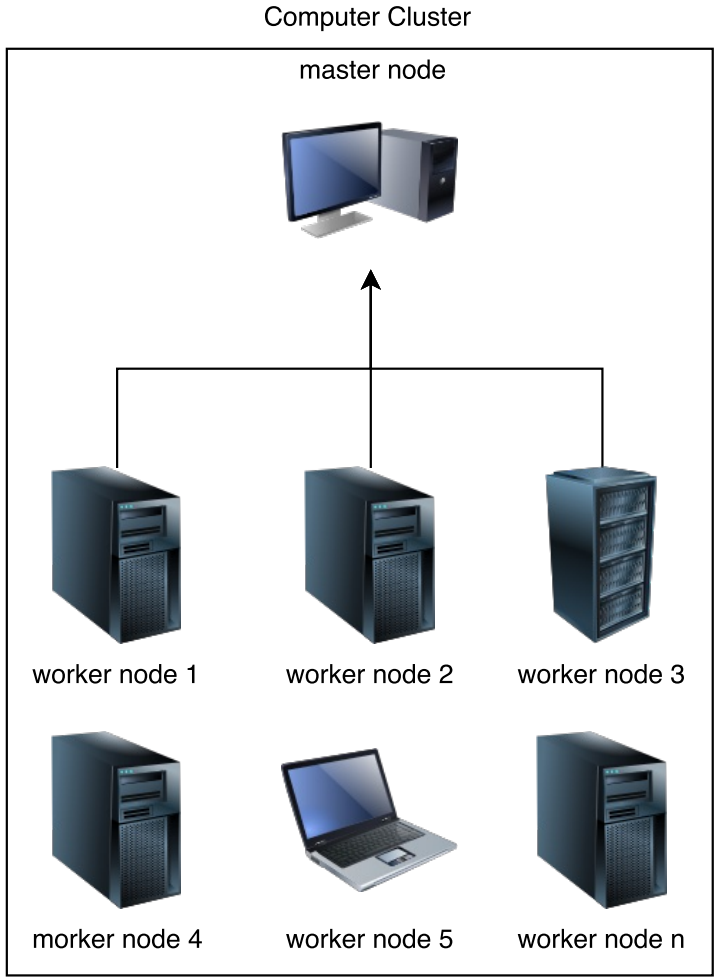
- kann für beliebig große Datenmengen einen Index erstellen, solange das Festplattenvolumen nicht überstiegen wird
- einzelne Blocke können größer sein
  - Indexerstelung effizienter
- distionaries und die erstellte PostingsList kann durch Kompression kompakt auf der Festplatte gespeichert werden
- Zeitkompplexität: $\Theta(T)$, kein Sortieren von TermID-DocID Paaren, alle Operationen linear

Distributed indexing

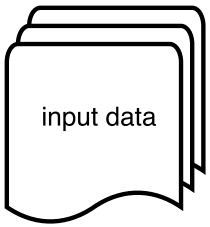
- manche Sammlungen übersteigen die Leistung eines einzelnen Rechners
  - beispielsweise das Web
- um Indizes über solche Sammlungen zu erstellen, muss die Arbeit auf mehrere Rechner verteilt werden

Distributed indexing

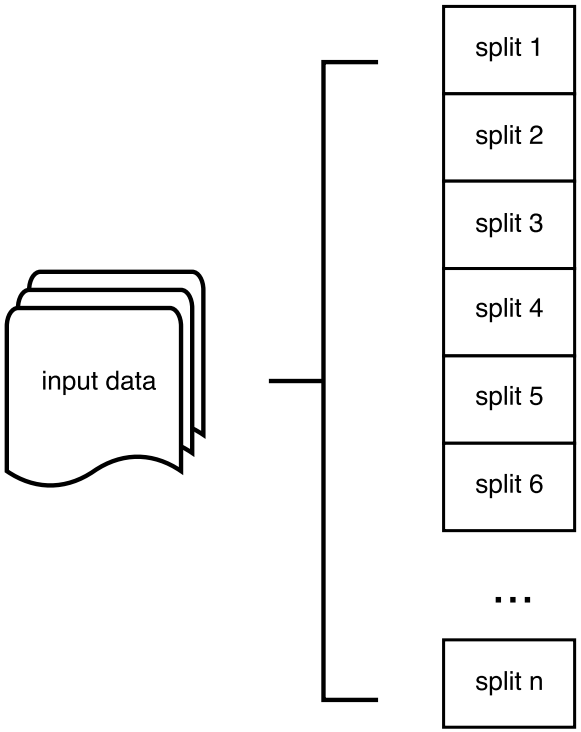
- manche Sammlungen übersteigen die Leistung eines einzelnen Rechners
  - beispielsweise das Web
- um Indizes über solche Sammlungen zu erstellen, muss die Arbeit auf mehrere Rechner verteilt werden



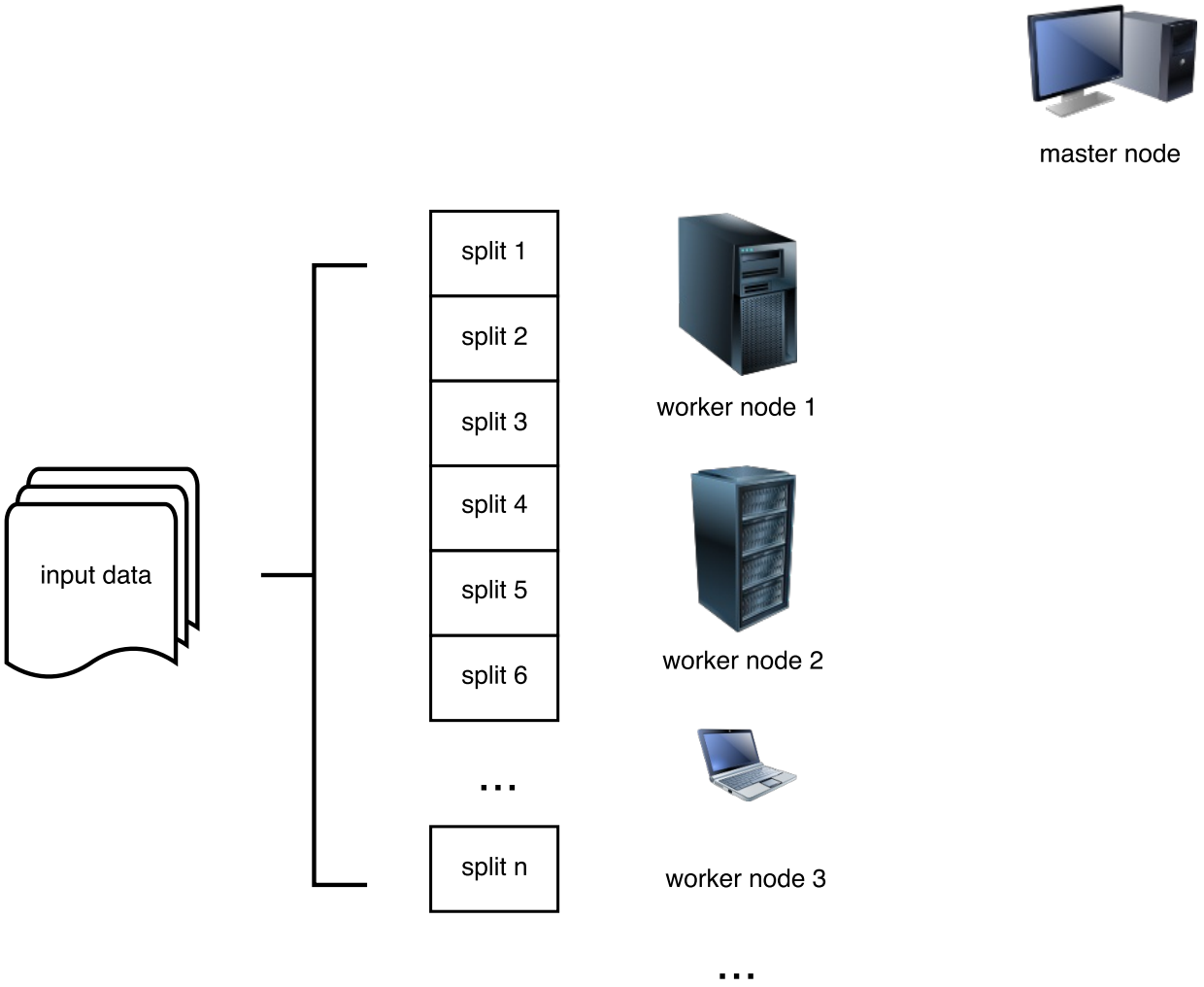
Distributed indexing - MapReduce



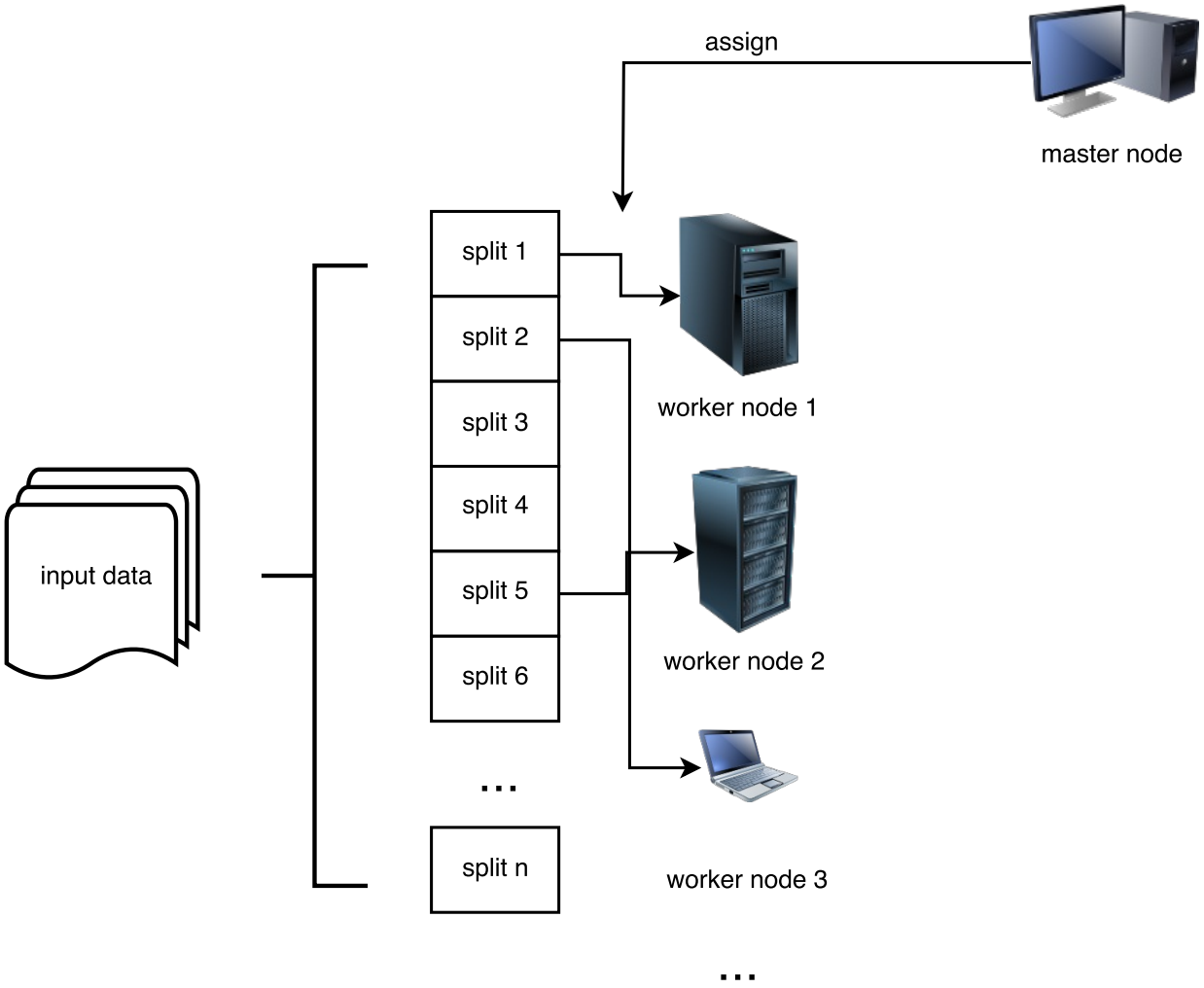
Distributed indexing - MapReduce



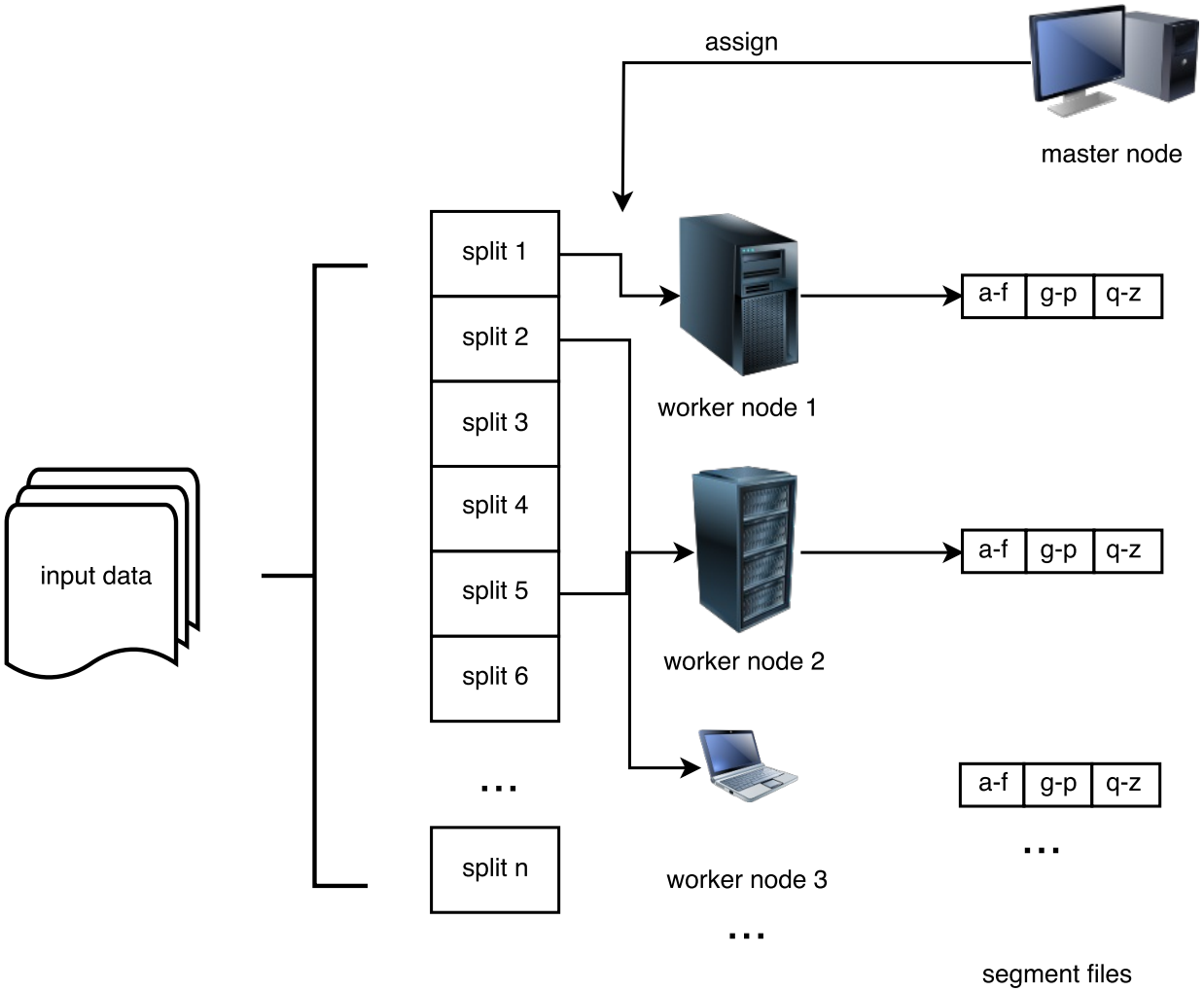
Distributed indexing - MapReduce



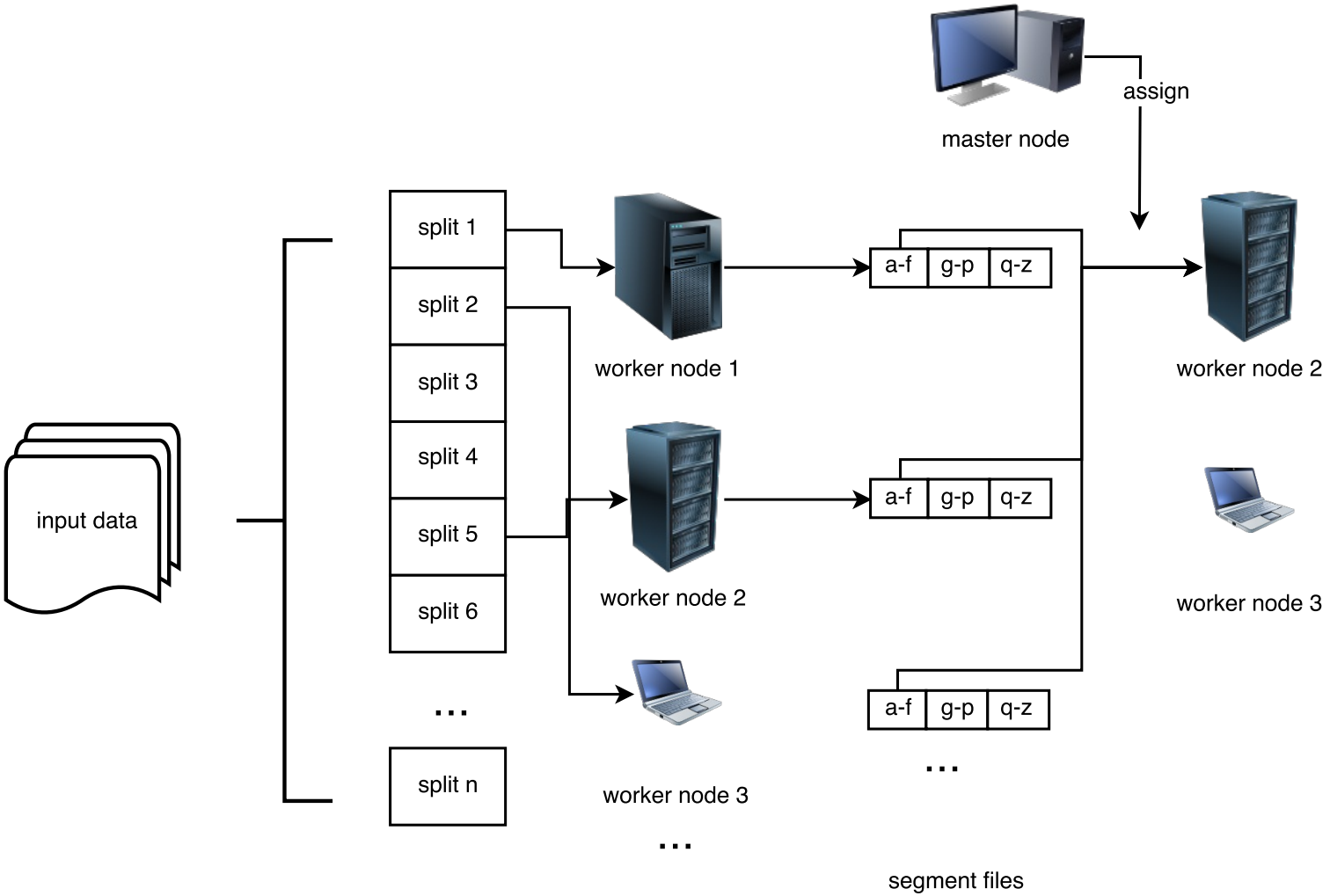
Distributed indexing - MapReduce



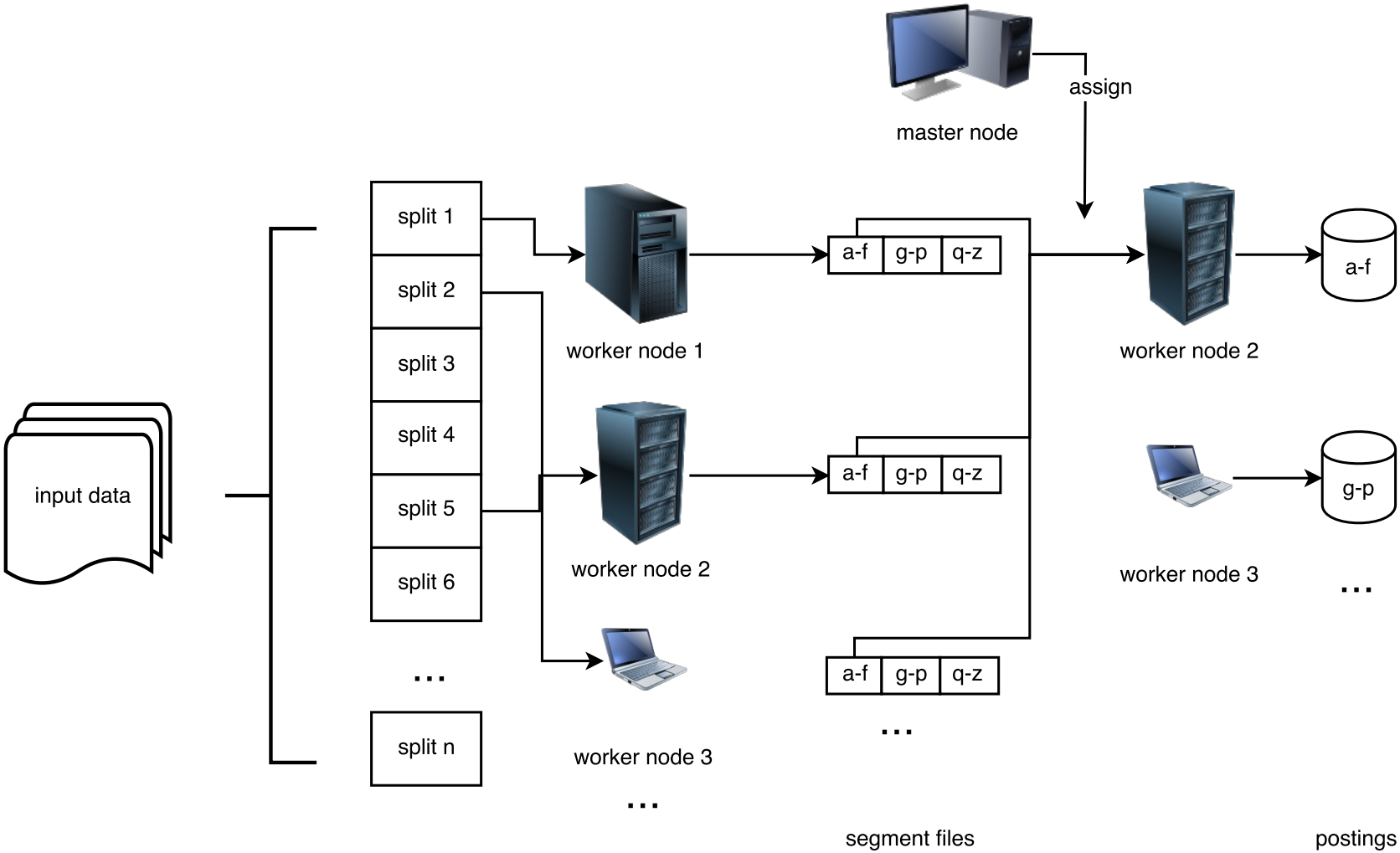
Distributed indexing - MapReduce



Distributed indexing - MapReduce



Distributed indexing - MapReduce



Dynamic indexing

- viele Sammlungen von Dokumenten ändern sich häufig
  - Webseiten werden geändert, gelöscht oder neue hinzugefügt..

Dynamic indexing

- viele Sammlungen von Dokumenten ändern sich häufig
  - Webseiten werden geändert, gelöscht oder neue hinzugefügt..
- Indexerstellung über eine solche Sammlung ebenfalls dynamsich

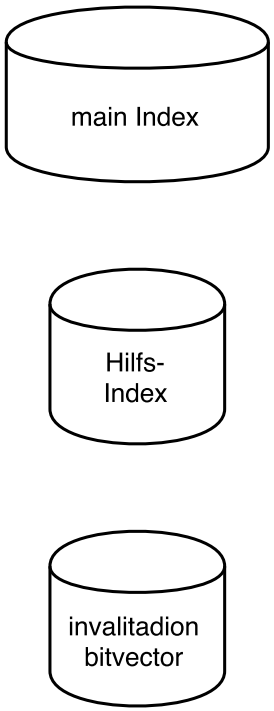
Dynamic indexing

- Index periodisch neu erstellen
  - akzeptabel wenn Änderungen nicht sehr groß
  - wenn Änderungen nicht sofort sichtbar sein müssen
- Hauptindex behalten und neue Dokumente in einen Hilfsindex speichern
  - beide Indizes regemäßig mergen

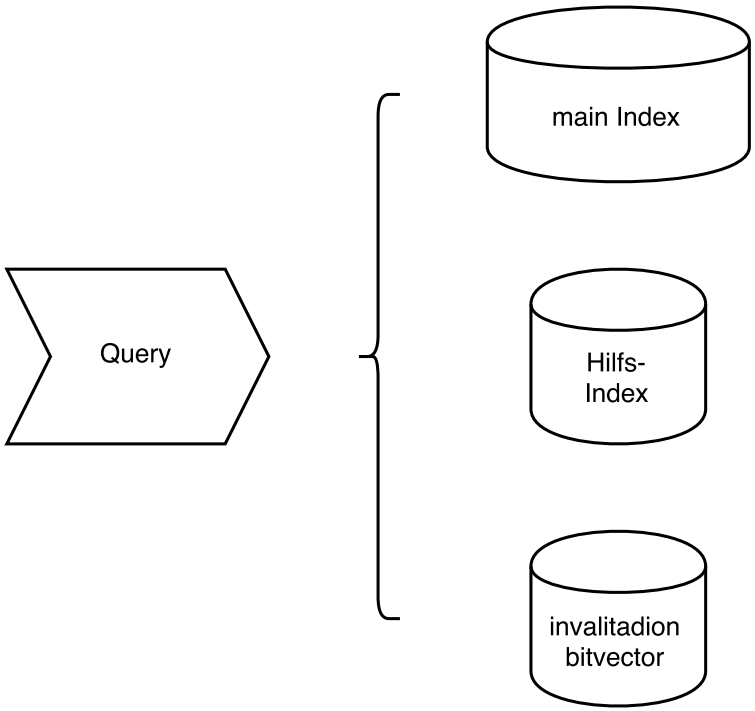
Dynamic indexing

- Index periodisch neu erstellen
  - akzeptabel wenn Änderungen nicht sehr groß
  - wenn Änderungen nicht sofort sichtbar sein müssen
- Hauptindex behalten und neue Dokumente in einen Hilfsindex speichern
  - beide Indizes regemäßig mergen

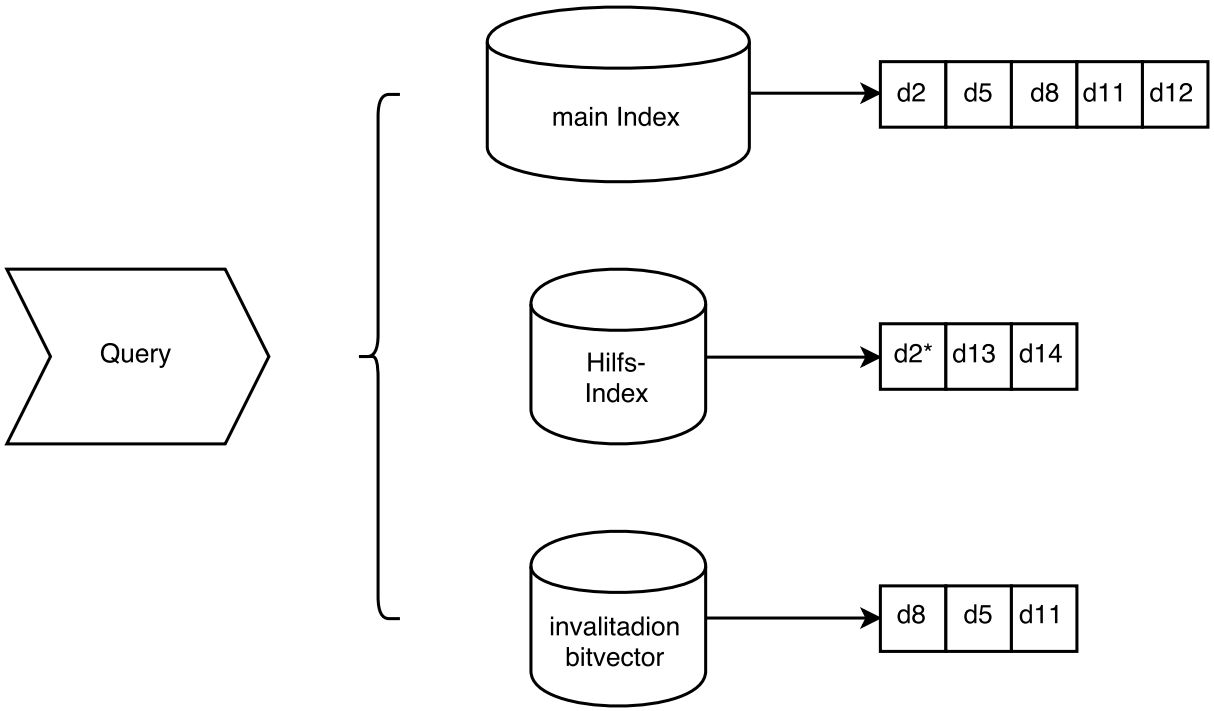
Dynamic indexing



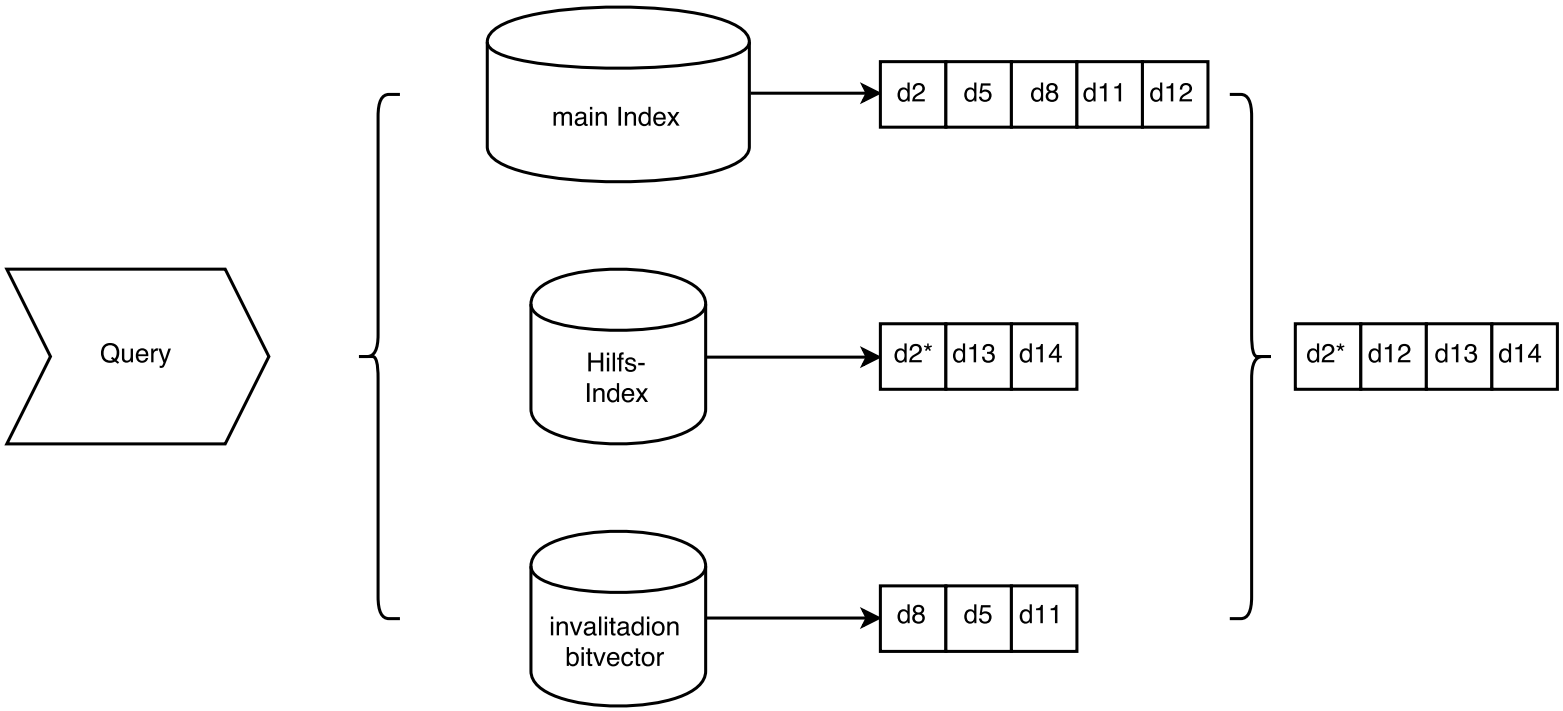
Dynamic indexing



Dynamic indexing



Dynamic indexing



andere Indexierungsverfahren

- ranked retrieval systems
- zugriffsbeschränkte Indizes
- „in situ“-Indexerstellung

Indexierung mit Solid State Drives

todo Tree Indexing, Datenstruktur FD-Tree erläutern, auf Vorteile eingehen..

Fazit

todo

Quellen

- Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze „Introduction to Information Retrieval“ [1] Cambridge University Press 2008, pp. 1-18 and 67-84.
- Ian H. Witten, Alistair Moffat, Timothy C. Bell „Managing Gigabytes: Compressing and Indexing Documents and Images“ [2] Morgan Kaufman Publishers 1999, pp. 223-261.
- Yinan Li, Bingsheng He, Robin Jun Yang, Qiong Luo, Ke YiTree (Hong Kong University of Science and Technology) „Indexing on Solid State Drives“ [3] The 36th International Conference on Very Large Data Bases, September 13-17, 2010, Singapore.

[1] https://nlp.stanford.edu/IR-book/pdf/04const.pdf

[2] https://books.google.de/books?id=2F74jyPl48EC&dq=Witten+et+al.+index+1999&lr=&hl=de&source=gbs_navlinks_s

[3] http://pages.cs.wisc.edu/~yinan/paper/fdtree_pvldb.pdf