



Recuperación de la Información

Text Document Retrieval

Semana 09



Optimización de Indexación



Optimización de Indexación

- ¿Como construimos un índice eficiente y escalable?
- ¿Qué estrategias puedo usar con memoria principal limitada?
- ¿Cómo podemos hacer uso de la memoria secundaria?



Índice Invertido Escalable

- Una vez analizados todos los documentos, el archivo invertido se ordena en función de los términos.

Centremenos en este paso
de ordenación

Term	Doc #
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2



Term	Doc #
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

Optimización de Indexación: A Reuters RCV1 document



You are here: [Home](#) > [News](#) > [Science](#) > [Article](#)

Go to a Section: [U.S.](#) [International](#) [Business](#) [Markets](#) [Politics](#) [Entertainment](#) [Technology](#) [Sports](#) [Oddly Enough](#)

Extreme conditions create rare Antarctic clouds

Tue Aug 1, 2006 3:20am ET

[Email This Article](#) [Print This Article](#) [Reprints](#)

[\[-\] Text](#) [\[+\]](#)



SYDNEY (Reuters) - Rare, mother-of-pearl colored clouds caused by extreme weather conditions above Antarctica are a possible indication of global warming, Australian scientists said on Tuesday.

Known as nacreous clouds, the spectacular formations showing delicate wisps of colors were photographed in the sky over an Australian meteorological base at Mawson Station on July 25.

Datos recogidos por un año entre 1995 y 1996

Optimización de Indexación: Reuters RCV1 statistics

statistical	value
documents	800,000
avg. # tokens per doc	200
terms (= word types)	400,000
avg. # bytes per token (incl. punct.)	6
avg. # bytes per token (without punct.)	4.5
avg. # bytes per term	7.5
non-positional postings	100,000,000

Optimización de Indexación

- **Sort-based index construction:**

- A medida que vamos contruyendo el índice, analizamos los documentos uno por uno.
 - Las publicaciones finales de cualquier término se completan al final.
- Con 8 bytes por (*termID*, *docID*), demana de mucho espacio para grandes colecciones.
- Para el caso de RCV1 $T = 100\ 000\ 000$.
 - Entonces ... es probable que esto lo podamos hacer en memoria, pero las colecciones típicas son mucho más grandes. Por ejemplo, el New York Times proporciona un índice de > 150 años de noticias.
- Por lo tanto, necesitamos almacenar los resultados intermedio en disco.

Optimización de Indexación

- **Scaling index construction**

- Construir un índice en memoria principal no es escalable.
 - No puede guardarse toda la colección en la memoria => ordenar y luego volver a escribir.
- ¿Cómo puedo escribir un índice para colecciones muy grandes?
- Tomando en cuenta las restricciones del hardware. . .
 - Memoria, disco, velocidad, etc.
- Transferir datos por bloques del disco a la memoria es más rápido que transferir muchos fragmentos pequeños.

Optimización de Indexación:

Blocked Sort-Based Indexing (BSBI)

- 8 bytes cada entrada (*termID*, *docID*)
- Estos se generan a medida que analizamos cada documento
- Luego se debe ordenar las entradas mediante el *termID*.
 - Ejemplo 100 entradas.
 - Si definimos un bloque equivalente a 10 entradas.
 - Se require 10 bloques
- Idea básica del algoritmo:
 - Acumular publicaciones para cada bloque, ordenar, y escribir en el disco
 - Luego mezclar los bloques en una estructura grande y ordenada.

Optimización de Indexación:

Blocked sort-based Indexing (BSBI)


BSBIINDEXCONSTRUCTION()

```
1   $n \leftarrow 0$ 
2  while (all documents have not been processed)
3  do  $n \leftarrow n + 1$ 
4       $block \leftarrow \text{PARSENEXTBLOCK}()$ 
5       $\text{BSBI-INVERT}(block)$ 
6       $\text{WRITEBLOCKTODISK}(block, f_n)$ 
7   $\text{MERGEBLOCKS}(f_1, \dots, f_n; f_{\text{merged}})$ 
```

termId	DocId
W1ID	1
W2ID	1
W2ID	2
W1ID	3
W2ID	3
W3ID	4

Sort and Build a
local index

termId	DocId
W1ID	1,3
W2ID	1,2,3
W3ID	4



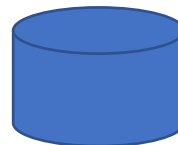
termId	DocId
W1ID	1
W2ID	1
W2ID	2
W1ID	3
W2ID	3
W3ID	4



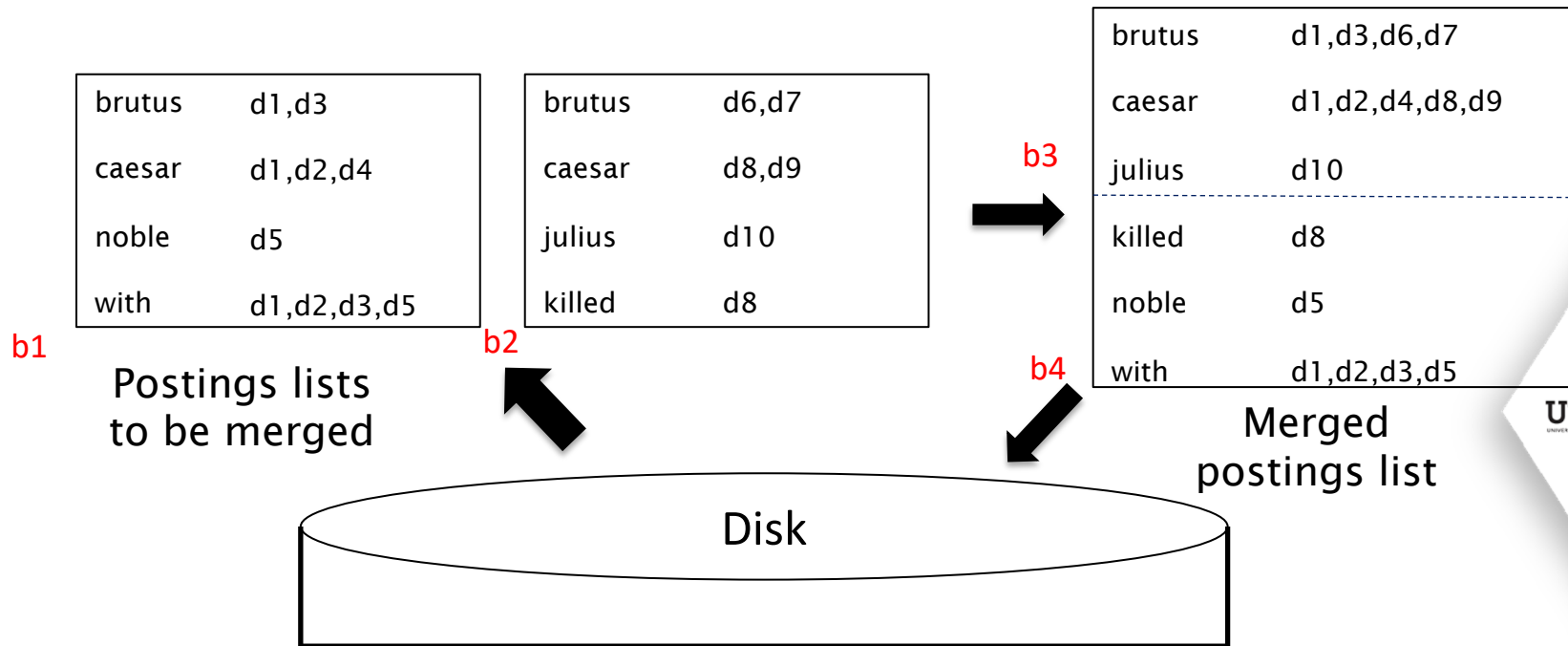
termId	DocId
W1ID	1
W1ID	3
W2ID	1
W2ID	2
W2ID	3
W3ID	4



termId	DocId
W1ID	1,3
W2ID	1,2,3
W3ID	4



Optimización de Indexación: Blocked sort-based Indexing (BSBI)



Dictionary (RAM)



Term	TermID
Term1	W1ID
Term2	W2ID
Term3	W3ID
Term4	W4ID
Term5	W5ID
Term6	W6ID
Term7	W7ID

Local Indexes (DISK BLOCKS)

termId	DocId
W1ID	1,3
W2ID	1,2,3
W3ID	4

termId	DocId
W2ID	4,5
W4ID	6
W5ID	4,5,6

termId	DocId
W1ID	7, 8
W3ID	6, 7
W6ID	6,8

termId	DocId
W2ID	9
W4ID	10,11
W7ID	9,10

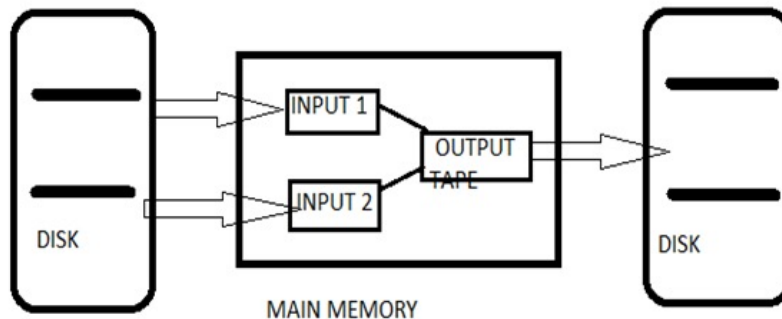
termId	DocId
W2ID	11
W3ID	11, 12
W7ID	11

Optimización de Indexación:

Blocked sort-based Indexing (BSBI)

- **¿Como mezclar los bloques?**

- Si se considera 10 bloques de 10 registros.
- Se puede hacer mezclas binarias, con un árbol de mezcla de $\log_2 10 = 4$ niveles.
- Durante cada capa, la lectura en memoria se ejecuta en bloques de 10M, se fusiona, se escribe de nuevo.



Hard Disk

2, 3, 5, 9, 10

1, 2, 5, 8

4, 5, 8, 10

2, 3, 6, 7

1, 3, 4, 10

2, 5, 7

1, 2, 3

4, 5, 7, 10

RAM (L=1)

Input 1

Output

Input 2

Hard Disk

Hard Disk

2, 3, 5, 9, 10

1, 2, 5, 8

4, 5, 8, 10

2, 3, 6, 7

1, 3, 4, 10

2, 5, 7

1, 2, 3

4, 5, 7, 10

RAM (L=1)

Input 1

Output

Input 2

Hard Disk

1, 2, 3, 5

5, 8, 9, 10

2, 3, 4, 5

6, 7, 8, 10

1, 2, 3, 4

5, 7, 10

1, 2, 3

4, 5, 7, 10

Hard Disk

1, 2, 3, 5

5, 8, 9, 10

2, 3, 4, 5

6, 7, 8, 10

1, 2, 3, 4

5, 7, 10

1, 2, 3

4, 5, 7, 10

RAM (L=2)

Input 1

Output

Input 2

Hard Disk

1, 2, 3

3, 4, 5

5, 6, 7, 8

8, 9, 10

1, 2

3, 4

5, 7

7, 10

Hard Disk

1,2,3

3,4,5

5,6,7,8

8,9,10

1,2

3,4

5,7

7,10

RAM (L=3)

Input 1

Output

Input 2

Hard Disk

1,2

2,3

3,4

4,5

5,6

7

8,9

10

Optimización de Indexación:

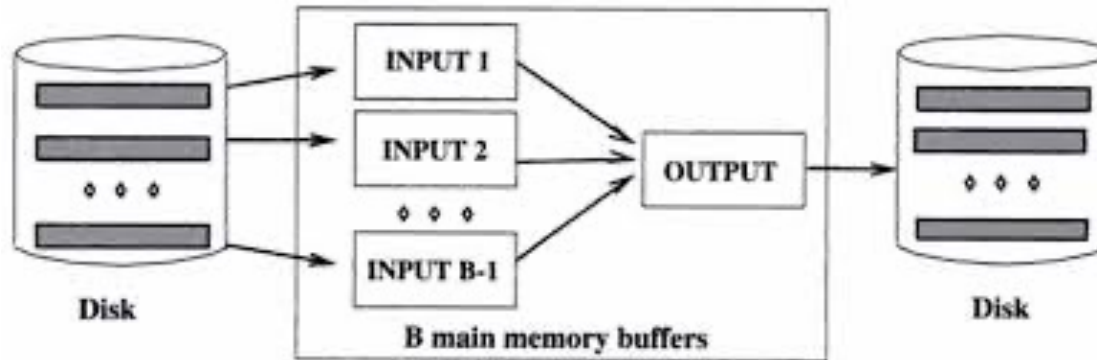
Blocked sort-based Indexing (BSBI)

- Pero es más eficiente hacer una fusión de múltiples mezclas, donde se está leyendo todos los bloques simultáneamente:
 - Abra todos los archivos de bloque simultáneamente y mantenga un búfer de lectura para cada uno y un búfer de escritura para el archivo de salida.
 - En cada iteración, seleccione el ID del término más bajo que no se haya procesado utilizando una cola de prioridad.
 - Combine todas las listas de publicaciones para ese termID y escríbalo al disco.

Optimización de Indexación:

Blocked sort-based Indexing (BSBI)

- Pero es más eficiente hacer una fusión de múltiples mezclas, donde se está leyendo todos los bloques simultáneamente:



Hard Disk

2, 3, 5, 9, 10

1, 2, 5, 8

4, 5, 8, 10

2, 3, 6, 7

1, 3, 4, 10

2, 5, 7

1, 2, 3

4, 5, 7, 10

RAM (L=1)

Input 1

Input 2

Input 3

Input 4

Output

Hard Disk

Hard Disk

2, 3, 5, 9, 10

1, 2, 5, 8

4, 5, 8, 10

2, 3, 6, 7

1, 3, 4, 10

2, 5, 7

1, 2, 3

4, 5, 7, 10

RAM (L=1)

Input 1

Input 2

Input 3

Input 4

Output

Hard Disk

1,2

2,3,4

5,6,7

8,9,10

1,2

3

4,5

7,10

Optimización de Indexación:

Blocked sort-based Indexing (BSBI)

- **Problemas pendientes con éste algoritmo:**

- Hemos trabajado bajo el supuesto de:
 - Podemos mantener el diccionario en la memoria principal.
- Entonces necesitamos que el diccionario (el cual crece dinámicamente) al ser implementado soporte eficientemente el mapeo de <term – termID>.
 - O mejor aun, trabajar directamente con el term.

Optimización de Indexación:

Single-pass in-memory indexing (SPIMI)

- Generar diccionarios (hash) separados para cada bloque, sin necesidad de mantener el mapeo <term-termID> entre los bloques.
- No ordenar (slide 38). Acumular en el hash las publicaciones en listas a medida que van ocurriendo.
- De esta manera, podemos generar un índice invertido completo para cada bloque.
- Estos índices separados pueden luego ser mezclados en un big index.

Optimización de Indexación:

Single-pass in-memory indexing (SPIMI)

BSBINDEXCONSTRUCTION()

```
1   $n \leftarrow 0$ 
2  while (all documents have not been processed)
3  do
4       $n \leftarrow n + 1$ 
5       $token\_stream \leftarrow ParseDocs()$ 
6       $fn \leftarrow SPIMI-INVERT(token\_stream)$ 
7  MERGEBLOCKS( $f_1, \dots, f_n; f_{merged}$ )
```

Optimización de Indexación:

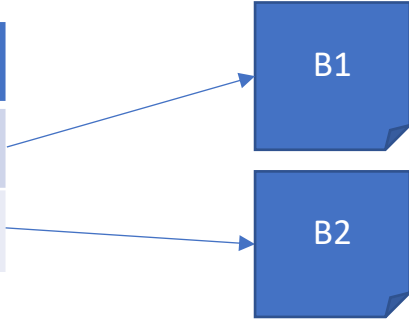
Single-pass in-memory indexing (SPIMI)

```
SPIMI-INVERT(token_stream)
1  output_file = NEWFILE()
2  dictionary = NEWHASH()
3  while (free memory available)
4  do token  $\leftarrow$  next(token_stream)
5      if term(token)  $\notin$  dictionary
6          then postings_list = ADDTODICTIONARY(dictionary, term(token))
7          else postings_list = GETPOSTINGSLIST(dictionary, term(token))
8          if full(postings_list)
9              then postings_list = DOUBLEPOSTINGSLIST(dictionary, term(token))
10         ADDTOPOSTINGSLIST(postings_list, docID(token))
11  sorted_terms  $\leftarrow$  SORTTERMS(dictionary)
12  WRITEBLOCKTODISK(sorted_terms, dictionary, output_file)
13  return output_file
```

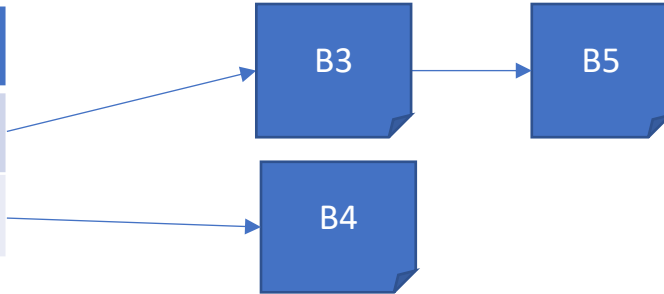
Luego, la mezcla de bloques es análogo a BSBI \rightarrow

Local Dictionary

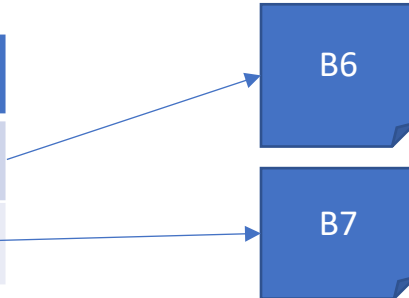
key	df	Bucket
w1		B1
w2		B2



key	df	Bucket
w2		B3
w3		B4

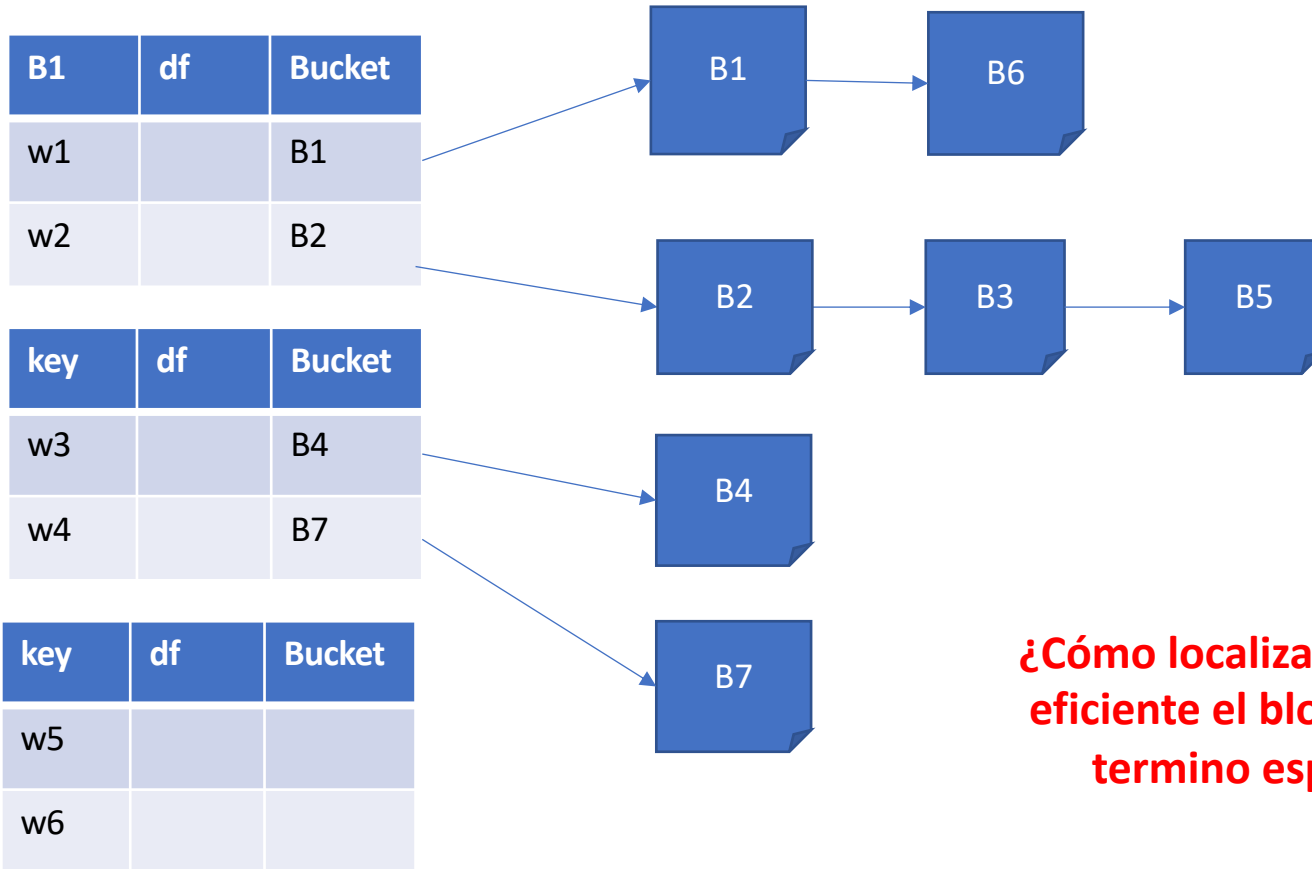


key	df	Bucket
w1		B6
w4		B7



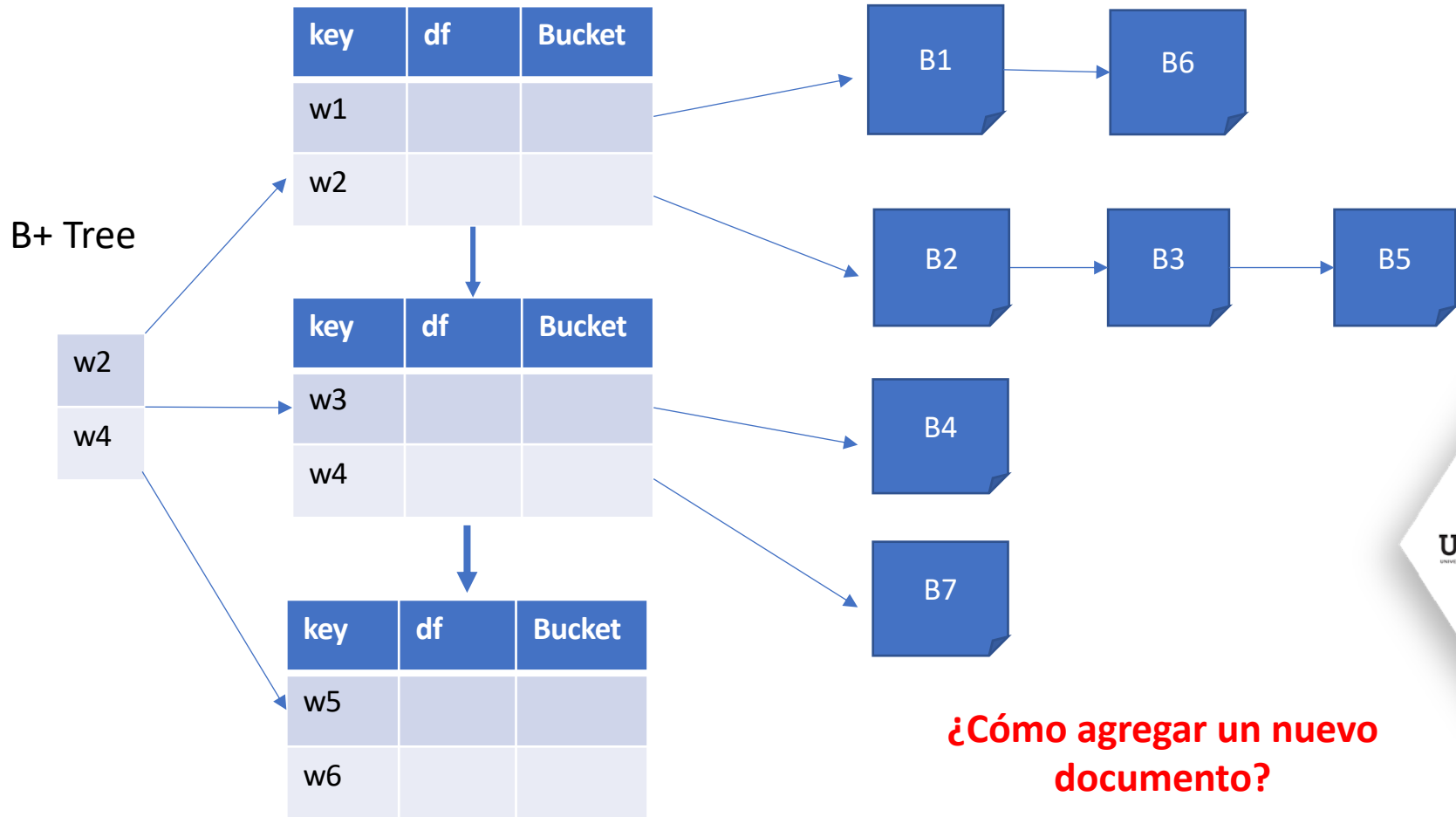
Posting Lists: puede requerir mas de un bloque, principalmente si el diccionario local ocupa toda la memoria RAM disponible (varios input buffers)

Merge Blocks

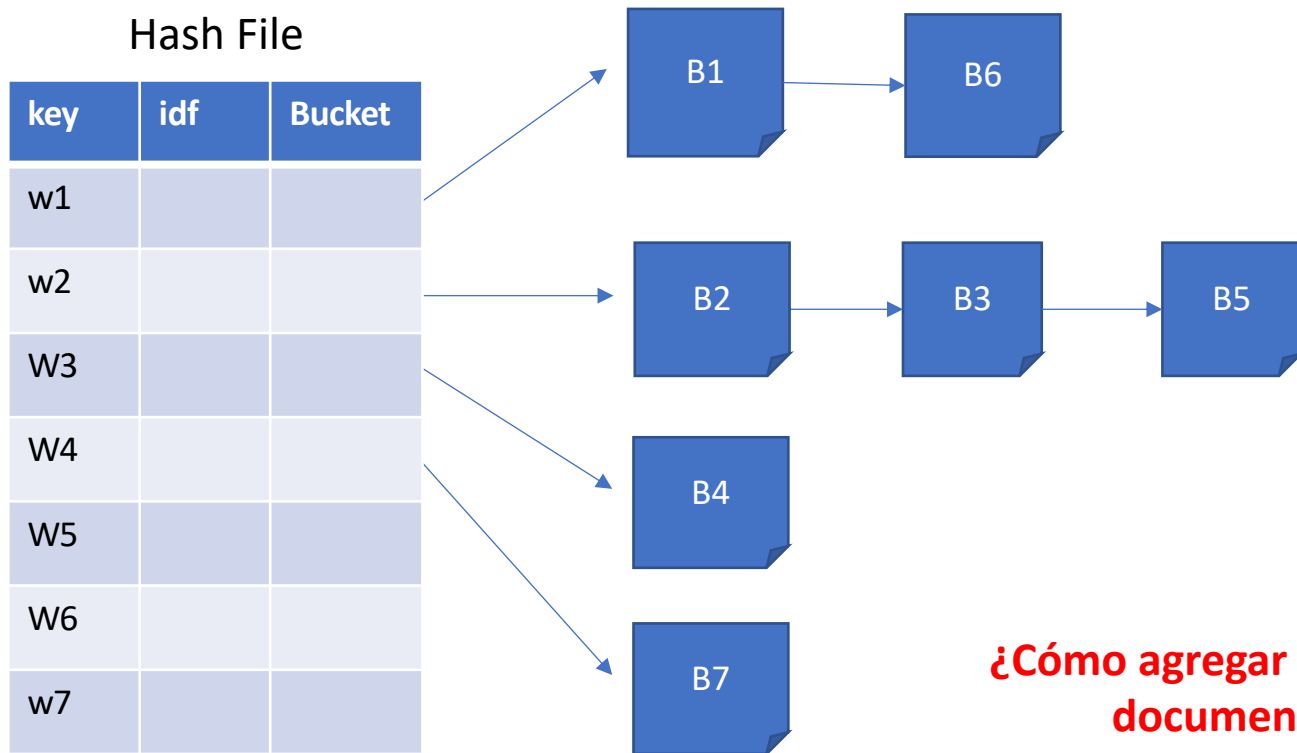


¿Cómo localizar de manera eficiente el bloque de un término específico?

Alternativa de Implementación con B+ File



Alternativa de Implementación con Hash File



¿Cómo agregar un nuevo documento?

Optimización de Indexación:

- **Dynamic indexing**
 - **GiST**
- **Distributed indexing**
 - **Map Reduce**

Papers de referencia

<https://nlp.stanford.edu/IR-book/html/htmledition/references-and-further-reading-4.html>

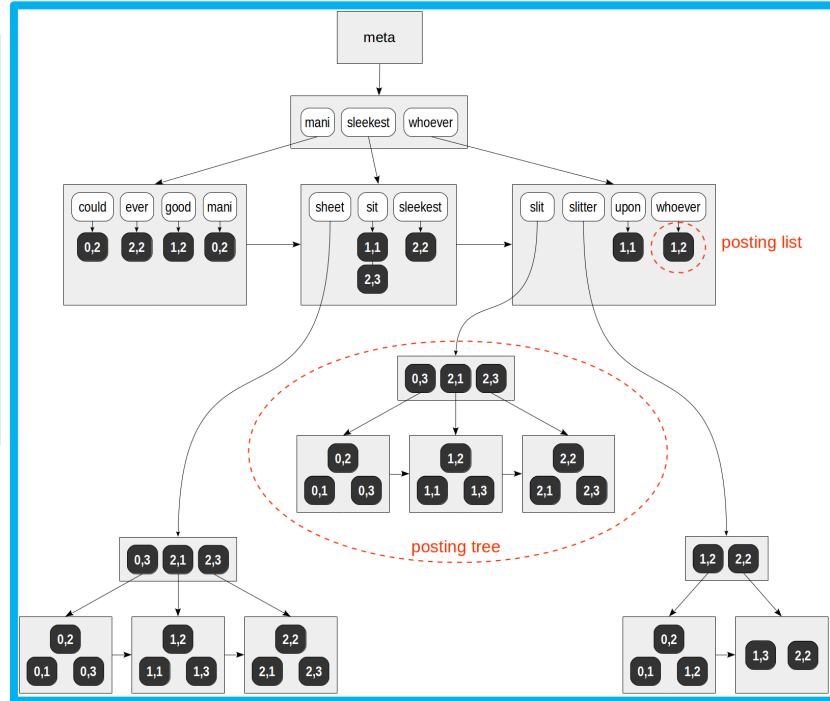
Indexing for Full-text Search in PostgreSQL

GIN vs GiST

INDEX			
the	[1,3,4]	TABLE	
quick	[1]		
brown	[1]		
fox	[1]	1	the quick brown fox
jumps	[2]	2	jumps over
over	[2]	3	the lazy dog
lazy	[3,4]	4	because the lazy dog
dog	[3,4]	5	is sleeping
is	[5]		
sleeping	[5]		

"As a rule, GIN beats GiST in accuracy and search speed. If the data is updated not frequently and fast search is needed, most likely GIN will be an option."

<https://postgrespro.com/blog/pgsql/4261647>



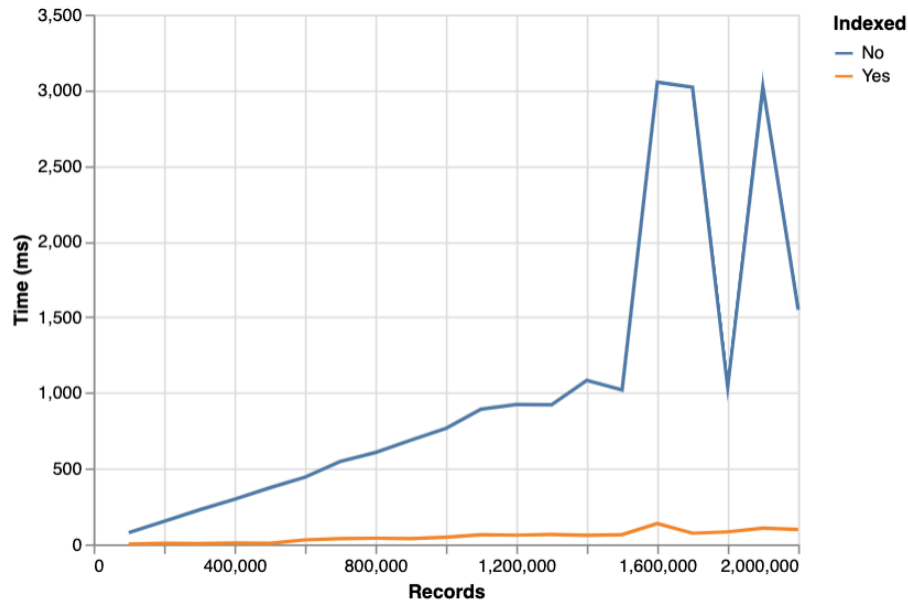
Generalized Inverted Index (GIN)

```
# create table
CREATE TABLE articles (
  body text,
  body_indexed text
);

# add an index
CREATE INDEX articles_search_idx ON articles USING gin
(body_indexed gin_trgm_ops);

# populate table with data
INSERT INTO articles
SELECT
  md5(random()::text),
  md5(random()::text)
from (
  SELECT * FROM generate_series(1,100000) AS id
) AS x;

# test
SELECT count(*) FROM articles where body ilike '%abc%';
SELECT count(*) FROM articles where body_indexed ilike '%abc%';
```



Indexación de textos con Solr

Cuando se actualiza la tabla se debe reindexar la información en Solr

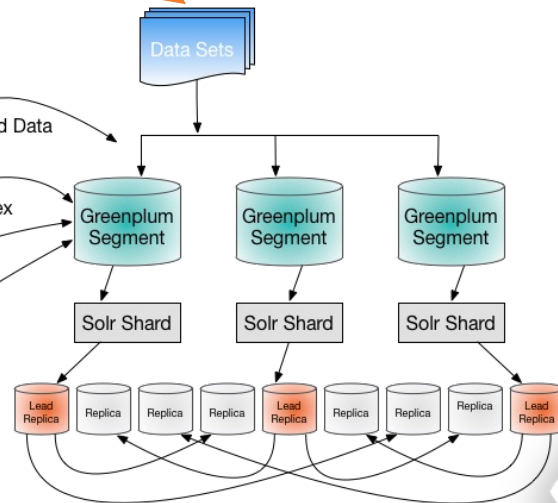
Id	Title	Description	Date

① Create Table / Load Data

② Create GPText Index

③ Populate Index

④ Commit Index



ADVANCED SEARCH

Accessories

Color

Size

Sale

Time

Type

100 results

RESET

SEARCH

Laboratorio

UTEC