Index Construction (Part 2)

Alfan F. Wicaksono
Fakultas Ilmu Komputer, Universitas Indonesia

A High Level View of Index Construction

Doc #56

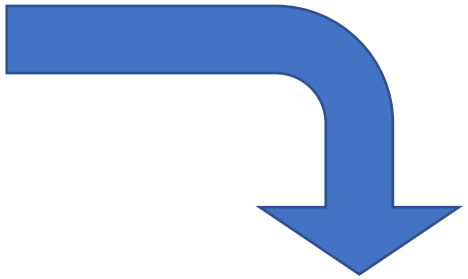
Buku-buku yang berisi cerita peradaban

Doc #30

Buku tersebut mengandung bab tentang adab

Doc #93

Kejujuran adalah adab utama yang membangun bangsa



adab	→	30	56	93
bab	→	30		
bangsa	→	93		

...

A Detail View – Step #1 Tokenization & Linguistic Preprocessing

Doc #56

Buku-buku yang berisi cerita peradaban

Doc #30

Buku tersebut mengandung bab tentang adab

Doc #93

Kejujuran adalah adab utama yang membangun bangsa

Tokenization, Normalization, Stemming, Stopwords Removal

Doc #56

buku | isi | cerita | adab

Doc #30

buku | sebut | kandung | bab | adab

Doc #93

jujur | adab | utama | bangun | bangsa

A Detail View – Step #1 Tokenization & Linguistic Preprocessing

-> Sequence of <Term, docID>

Doc #56

buku isi cerita

adab

Doc #30

buku sebut kandung

bab adab

Doc #93

jujur adab utama

bangun bangsa

Term	Doc ID
buku	30
sebut	30
kandung	30
bab	30
adab	30
buku	56
isi	56
cerita	56
adab	56
jujur	93
adab	93
utama	93
bangun	93
bangsa	93

A Detail View – Step #2 Sorting the Sequence of Terms

Term	Doc ID
buku	30
sebut	30
kandung	30
bab	30
adab	30
buku	56
isi	56
cerita	56
adab	56
jujur	93
adab	93
utama	93
bangun	93
bangsa	93

Term	Doc ID
adab	30
adab	56
adab	93
bab	30
bangsa	93
bangun	93
buku	30
buku	56
cerita	56
isi	56
jujur	93
kandung	30
sebut	30
utama	93

External Sorting

Recall– Step #2 Sorting the Sequence of Terms

Term	Doc ID
buku	30
sebut	30
kandung	30
bab	30
adab	30
buku	56
isi	56
cerita	56
adab	56
jujur	93
adab	93
utama	93
bangun	93
bangsa	93

Term	Doc ID
adab	30
adab	56
adab	93
bab	30
bangsa	93
bangun	93
buku	30
buku	56
cerita	56
isi	56
jujur	93
kandung	30
sebut	30
utama	93

Recall– Step #2 Sorting the Sequence of Terms

Term	Doc ID
buku	30
sebut	30
kandung	30
bab	30
adab	30
buku	56
isi	56
cerita	
adab	
jujur	
adab	
utama	
bangun	
bangsa	93

Term	Doc ID
adab	30
adab	56
adab	93
bab	30
bangsa	93
bangun	93
buku	30
buku	56
cerita	56
isi	56
jujur	93
kandung	30
sebut	30
utama	93

Di kuliah SDA, Anda sudah belajar berbagai macam teknik sorting dan analisis kompleksitasnya.

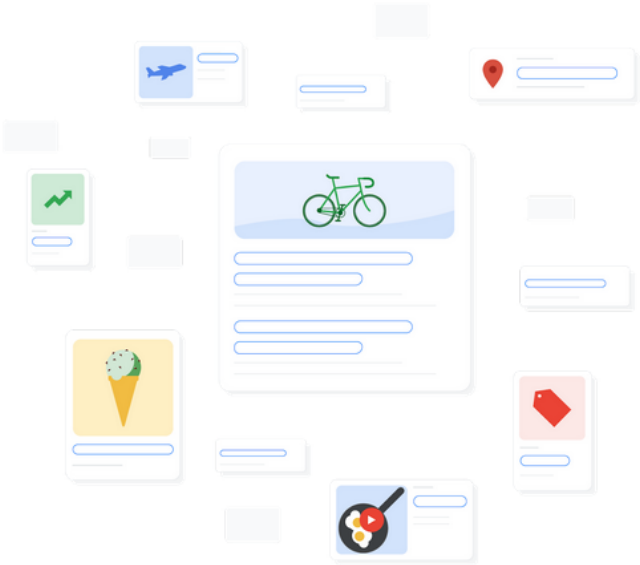
Menurut Anda proses pengurutan sequence of terms ini dapat dilakukan semuanya di memori?

Google Search Overview Our approach How Search works Features Our history

Organizing information Ranking results Rigorous testing Detecting spam

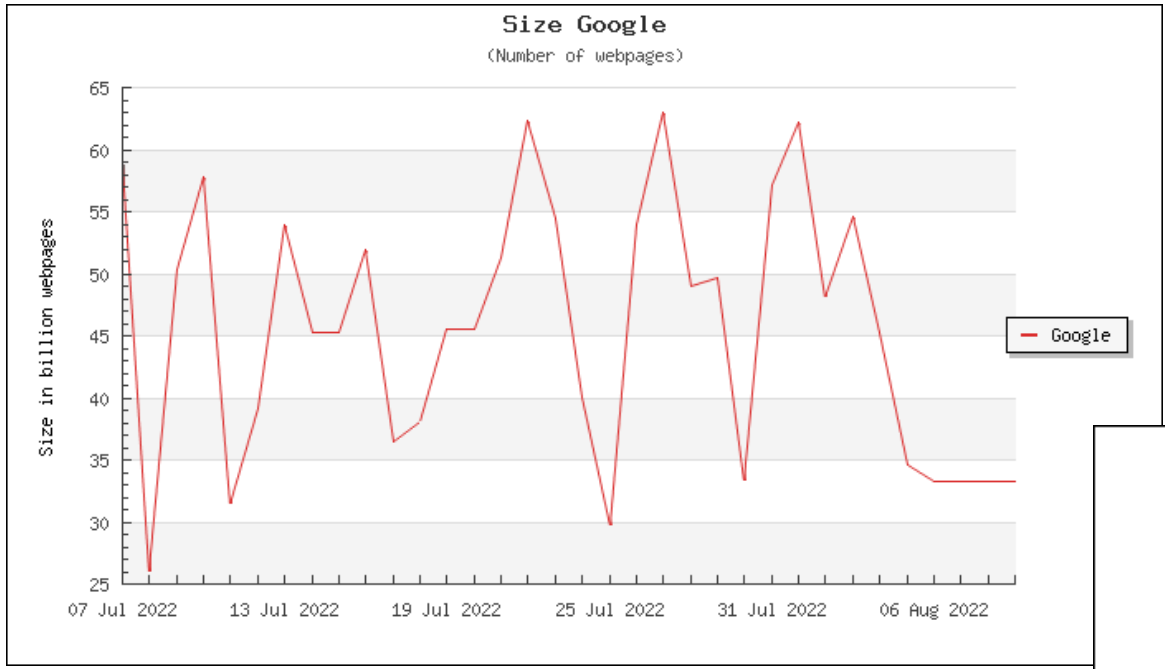
How Google Search organizes information

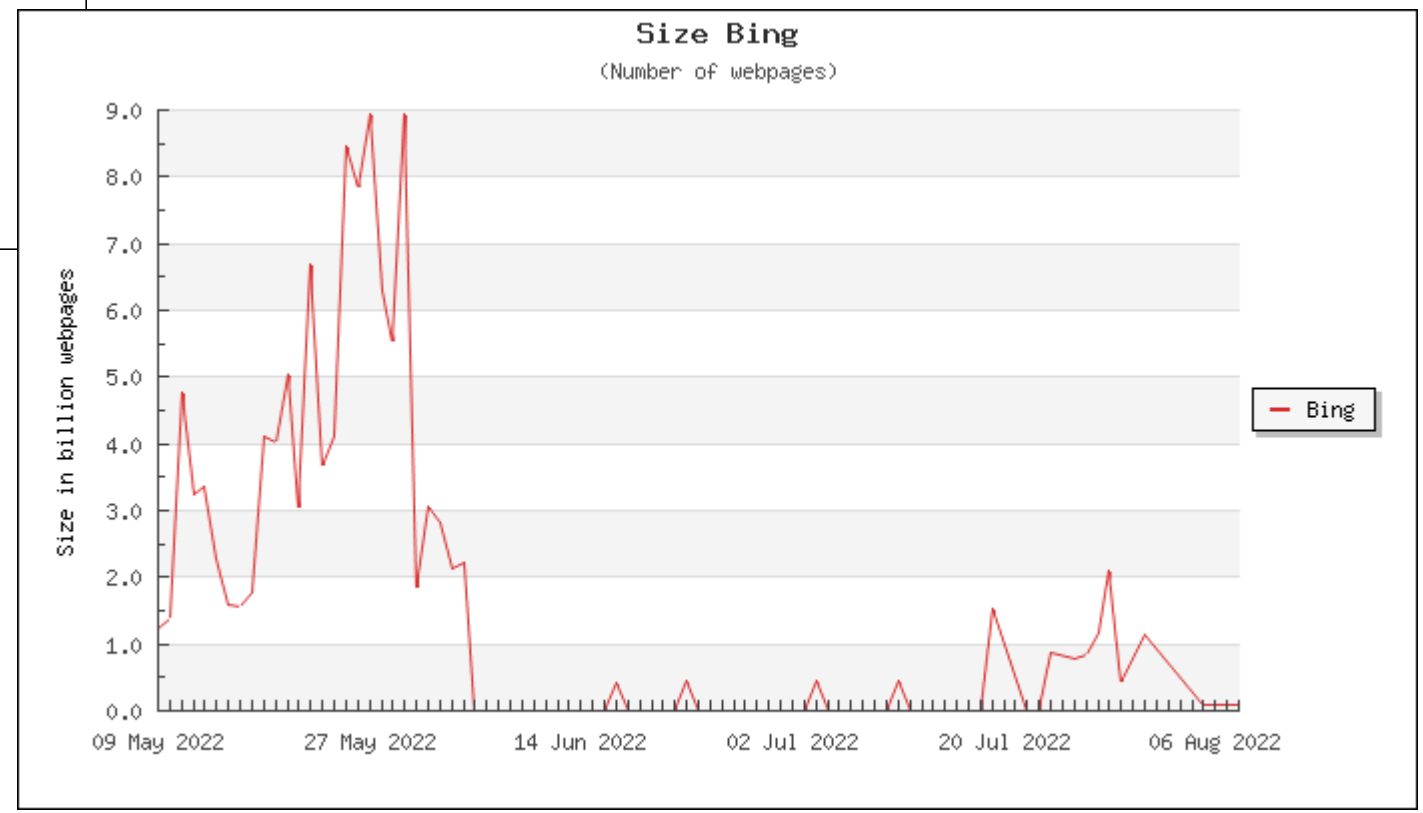
When you Search, Google looks through hundreds of billions of webpages and other content stored in our Search index to find helpful information — more information than all of the libraries of the world.



"The Google Search index contains **hundreds of billions of webpages** and is well over **100,000,000 gigabytes in size**. It's like the index in the back of a book — with an entry for every word seen on every webpage we index. When we index a webpage, we add it to the entries for all of the words it contains."

https://www.google.com/search/howsearchworks/how-search-works/organizing-information/





https://www.worldwidewebsize.com/

Pengukuran banyaknya Web yang di-index oleh major search engines.

Antal van den Bosch, Toine Bogers, Maurice de Kunder, "Estimating search engine index size variability: a 9-year longitudinal study", Scientometrics, 2016

Scaling Index Construction

- Sorting the whole sequence of <term, docID> inside a memory is not scalable. We need to store them in external storage.

- The problem is that access to data on disk is **much slower** than access to data on memory.

- How can we sort **very large collections** stored in a disk?

Some Hardware Basics --> POK

Statistics	Values
Average Seek Time (Disk)	Around 3ms (high end servers) – 15ms (mobile drives) Around 0.08ms – 0.16ms (SSDs)
Disk Transfer Time / Byte	0.5×10^{-8} s – 1.0×10^{-8} s
Memory Transfer Time / Byte (Memory -> CPU)	DDR3 --> 0.7×10^{-10} s – 1.0×10^{-10} s DDR4 --> 0.4×10^{-10} s – 0.6×10^{-10} s
Block (A smallest unit of storage, made up of sectors, and used to read a file or write data to a file)	Block sizes of 8, 16, 32, and 64 KB are common.

Baca tulis dari disk lebih lama dibandingkan memory

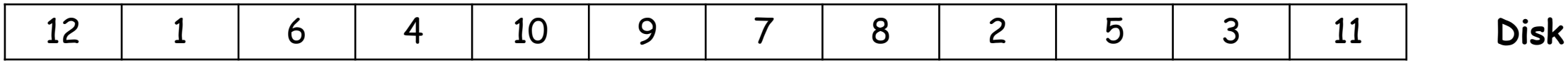
Berapa perkiraan lama waktu yang dibutuhkan untuk akses 10 MB data dari Disk,

- jika data tersimpan secara berurutan di sebuah block?
- jika data tersebar di 100 block berjauhan?

External Sort-Merge Algorithm

- Data berukuran sangat besar dan tidak dapat dimuat ke memori.
- Data kemudian dianggap sebagai sekumpulan blok/chunk, dimana setiap blok/chunk dapat dimuat ke memori.
- Untuk setiap blok/chunk, muat data ke memori, kemudian terapkan proses sorting. Jika sudah, kembali tulis ke Disk.
- Lakukan proses "merging" terhadap semua blok/chunk (yang sebelumnya sudah diurutkan) di Disk (**K-way merge**).

Demo: External Sort-Merge, dengan 3-Way Merge



Disk

Memori (hanya muat 8 angka)

Demo: External Sort-Merge, dengan 3-Way Merge

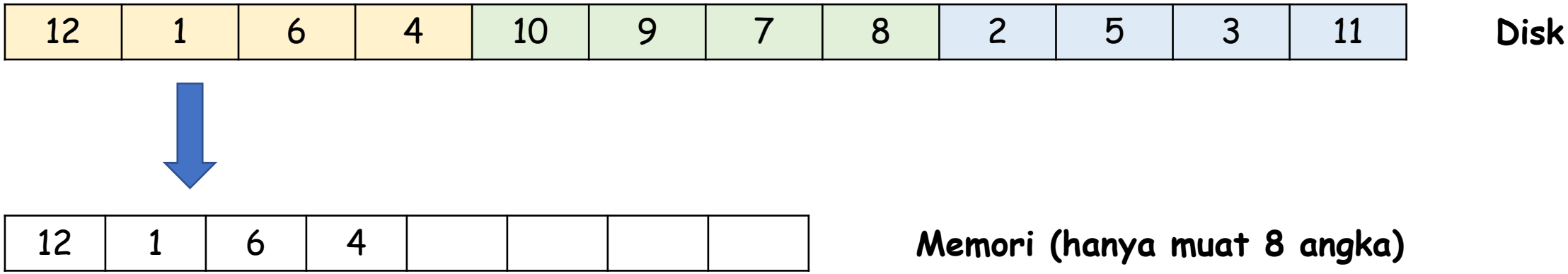
Kita definisikan sebuah blok/chunk terdiri dari 4 angka

Disk

Memori (hanya muat 8 angka)

Demo: External Sort-Merge, dengan 3-Way Merge

Kita definisikan sebuah blok/chunk terdiri dari 4 angka

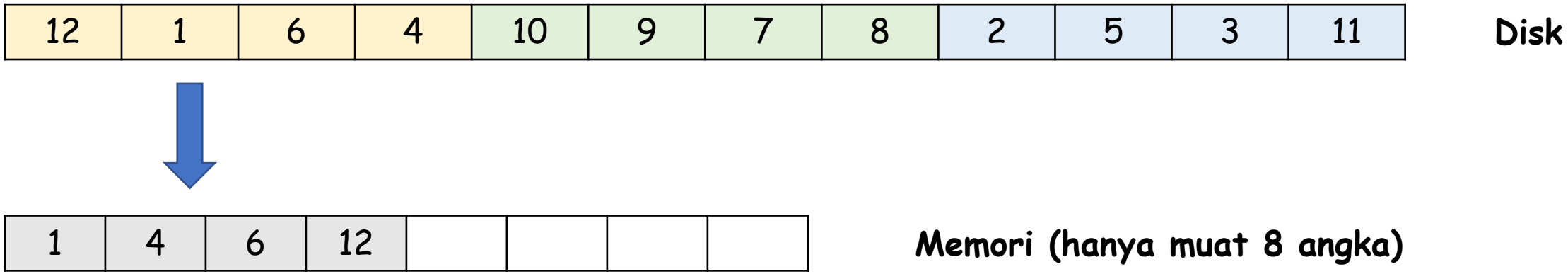


Untuk setiap blok/chunk, muat data ke memori, kemudian terapkan proses sorting. Jika sudah, kembali tulis ke Disk.

Untuk proses sorting di memori, gunakan algoritme populer seperti quick-sort, in-memory merge sort, atau yang lainnya.

Demo: External Sort-Merge, dengan 3-Way Merge

Kita definisikan sebuah blok/chunk terdiri dari 4 angka

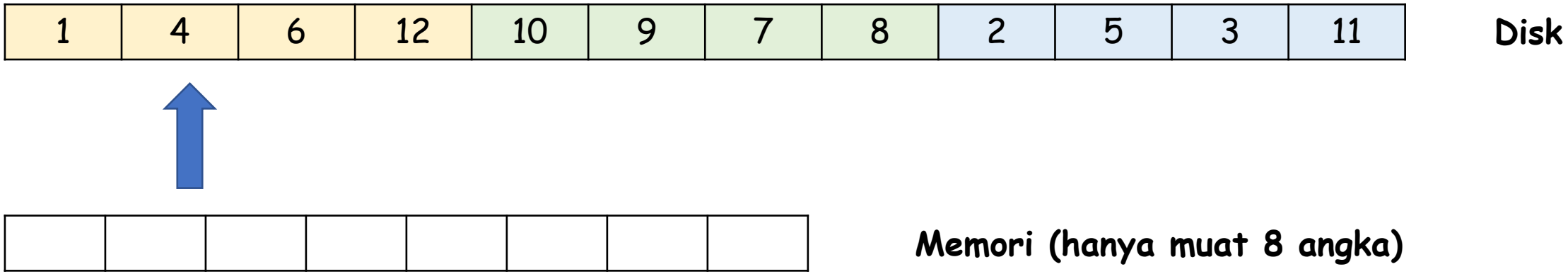


Untuk setiap blok/chunk, muat data ke memori, kemudian terapkan proses sorting. Jika sudah, kembali tulis ke Disk.

Untuk proses sorting di memori, gunakan algoritme populer seperti quick-sort, in-memory merge sort, atau yang lainnya.

Demo: External Sort-Merge, dengan 3-Way Merge

Kita definisikan sebuah blok/chunk terdiri dari 4 angka

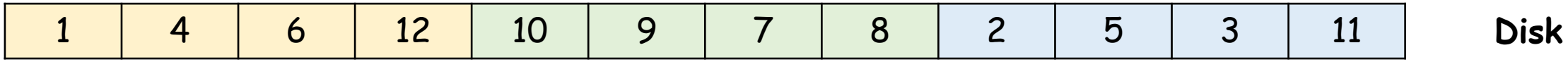


Untuk setiap blok/chunk, muat data ke memori, kemudian terapkan proses sorting. Jika sudah, kembali tulis ke Disk.

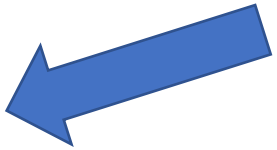
Untuk proses sorting di memori, gunakan algoritme populer seperti quick-sort, in-memory merge sort, atau yang lainnya.

Demo: External Sort-Merge, dengan 3-Way Merge

Kita definisikan sebuah blok/chunk terdiri dari 4 angka



Disk



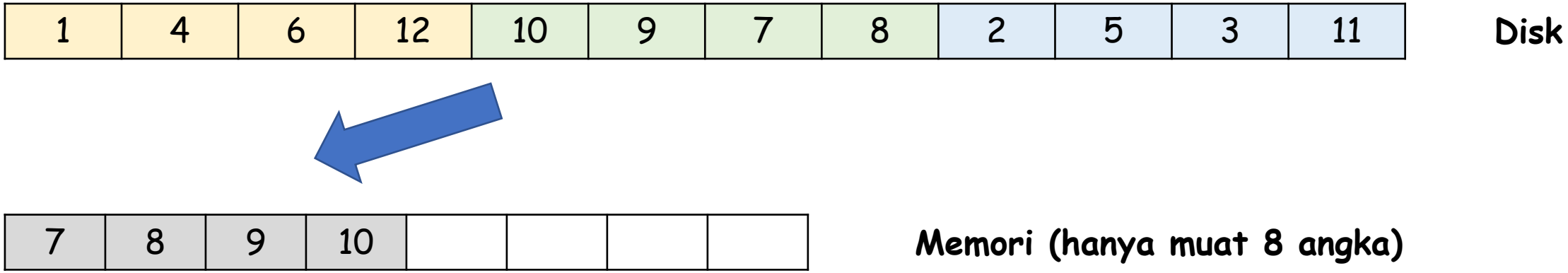
Memori (hanya muat 8 angka)

Untuk setiap blok/chunk, muat data ke memori, kemudian terapkan proses sorting. Jika sudah, kembali tulis ke Disk.

Untuk proses sorting di memori, gunakan algoritme populer seperti quick-sort, in-memory merge sort, atau yang lainnya.

Demo: External Sort-Merge, dengan 3-Way Merge

Kita definisikan sebuah blok/chunk terdiri dari 4 angka

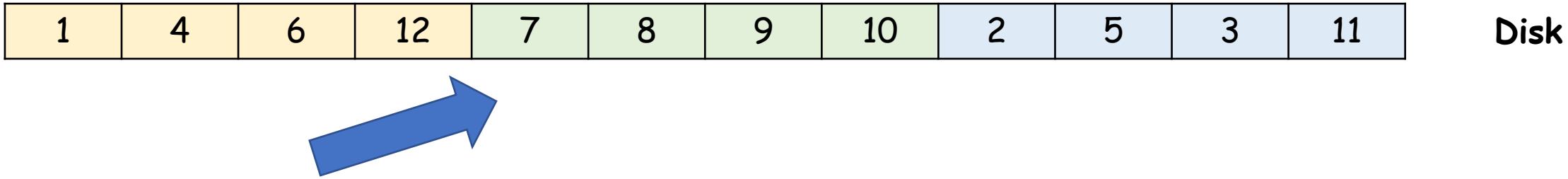


Untuk setiap blok/chunk, muat data ke memori, kemudian terapkan proses sorting. Jika sudah, kembali tulis ke Disk.

Untuk proses sorting di memori, gunakan algoritme populer seperti quick-sort, in-memory merge sort, atau yang lainnya.

Demo: External Sort-Merge, dengan 3-Way Merge

Kita definisikan sebuah blok/chunk terdiri dari 4 angka

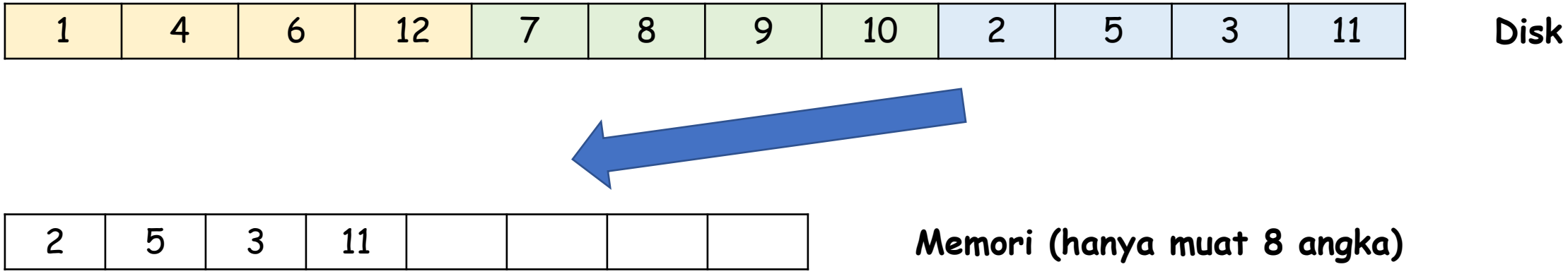


Untuk setiap blok/chunk, muat data ke memori, kemudian terapkan proses sorting. Jika sudah, kembali tulis ke Disk.

Untuk proses sorting di memori, gunakan algoritme populer seperti quick-sort, in-memory merge sort, atau yang lainnya.

Demo: External Sort-Merge, dengan 3-Way Merge

Kita definisikan sebuah blok/chunk terdiri dari 4 angka

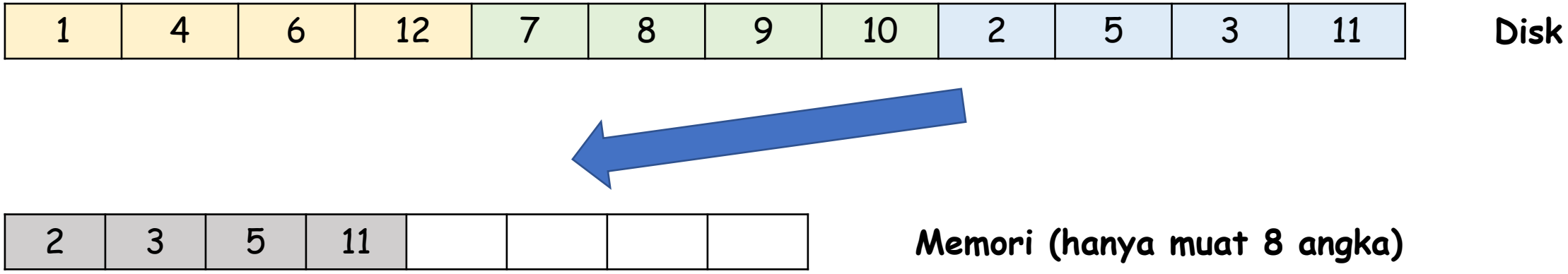


Untuk setiap blok/chunk, muat data ke memori, kemudian terapkan proses sorting. Jika sudah, kembali tulis ke Disk.

Untuk proses sorting di memori, gunakan algoritme populer seperti quick-sort, in-memory merge sort, atau yang lainnya.

Demo: External Sort-Merge, dengan 3-Way Merge

Kita definisikan sebuah blok/chunk terdiri dari 4 angka

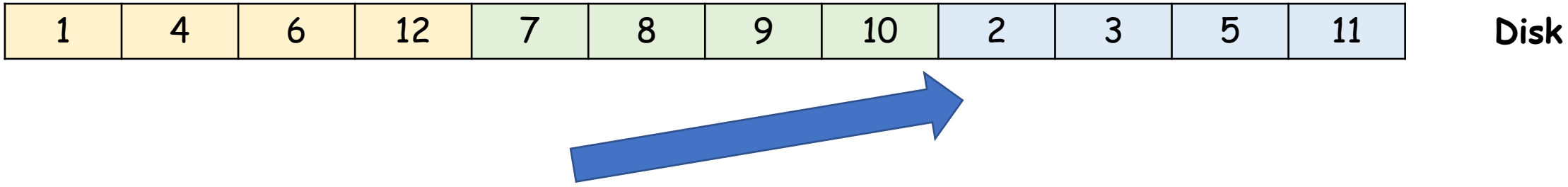


Untuk setiap blok/chunk, muat data ke memori, kemudian terapkan proses sorting. Jika sudah, kembali tulis ke Disk.

Untuk proses sorting di memori, gunakan algoritme populer seperti quick-sort, in-memory merge sort, atau yang lainnya.

Demo: External Sort-Merge, dengan 3-Way Merge

Kita definisikan sebuah blok/chunk terdiri dari 4 angka



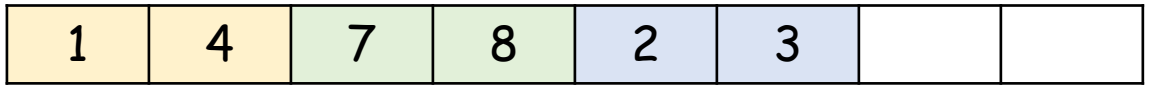
Untuk setiap blok/chunk, muat data ke memori, kemudian terapkan proses sorting. Jika sudah, kembali tulis ke Disk.

Untuk proses sorting di memori, gunakan algoritme populer seperti quick-sort, in-memory merge sort, atau yang lainnya.

Demo: External Sort-Merge, dengan 3-Way Merge

Kita definisikan sebuah blok/chunk terdiri dari 4 angka

Disk



Memori (hanya muat 8 angka)

harus ada untuk write data

Read Buffer

Write Buffer

Intinya, yang kecil dimasukkin di write buffer, tergantung dengan pointer, kalo misalnya pointer udah lewat dari chunk, maka ambil yang terkecil dari disk (dari chunk yang dia ambil)

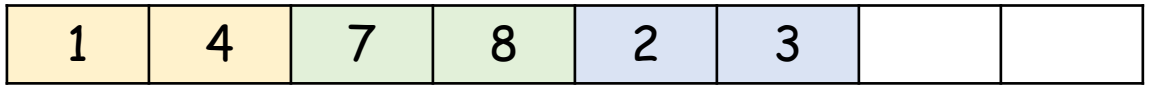
Copy sebagian (mis. 50%) dari masing-masing chunk ke memori.

Sebagian area memori dikosongkan untuk menyimpan hasil intermediate dari proses 3-way merge.

Demo: External Sort-Merge, dengan 3-Way Merge

Kita definisikan sebuah blok/chunk terdiri dari 4 angka

Disk



Memori (hanya muat 8 angka)

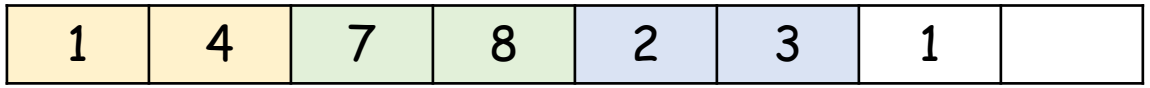
Lakukan proses 3-way merge.

Bandingkan 3 angka, lalu simpan angka yang **paling kecil** di bagian kosong memori (bisa menggunakan struktur data **Heap** untuk perbandingan ini).

Demo: External Sort-Merge, dengan 3-Way Merge

Kita definisikan sebuah blok/chunk terdiri dari 4 angka

Disk



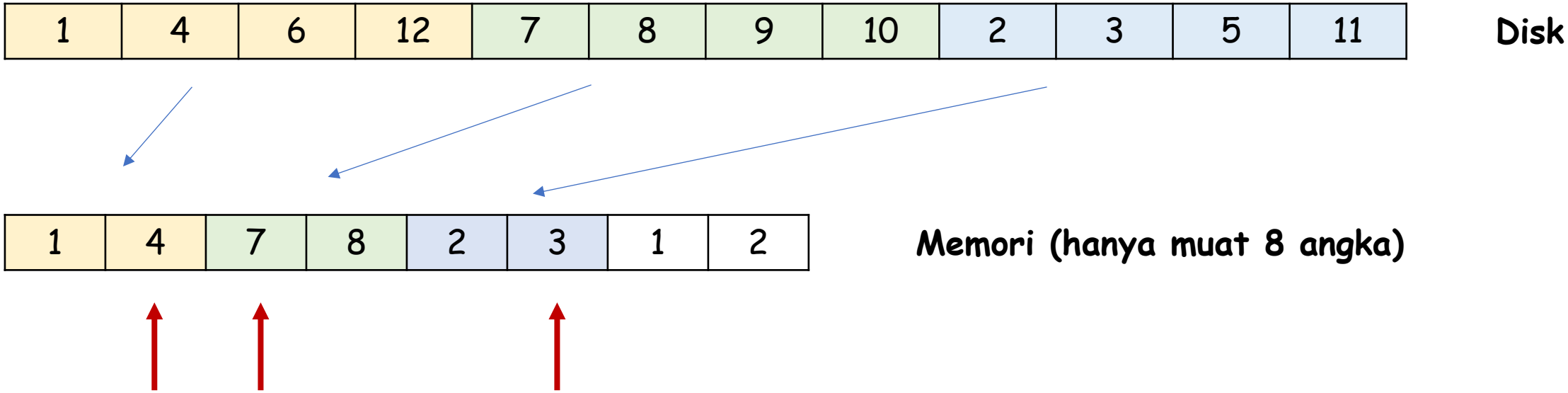
Memori (hanya muat 8 angka)

Lakukan proses 3-way merge.

Bandingkan 3 angka, lalu simpan angka yang **paling kecil** di bagian kosong memori (bisa menggunakan struktur data **Heap** untuk perbandingan ini).

Demo: External Sort-Merge, dengan 3-Way Merge

Kita definisikan sebuah blok/chunk terdiri dari 4 angka



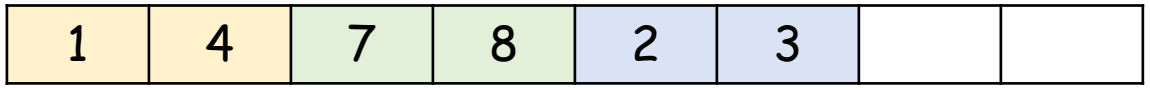
Lakukan proses 3-way merge.

Bandingkan 3 angka, lalu simpan angka yang **paling kecil** di bagian kosong memori (bisa menggunakan struktur data **Heap** untuk perbandingan ini).

Demo: External Sort-Merge, dengan 3-Way Merge

Kita definisikan sebuah blok/chunk terdiri dari 4 angka

Disk



Memori (hanya muat 8 angka)

Jika **write buffer** sudah penuh, tulis ke file hasil di Disk

Disk (Di Bagian Lain)

Demo: External Sort-Merge, dengan 3-Way Merge

Kita definisikan sebuah blok/chunk terdiri dari 4 angka

Disk

Memori (hanya muat 8 angka)

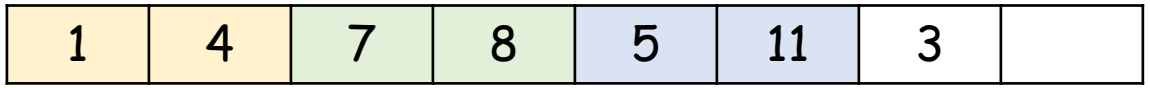
Ada **read buffer** yang sudah diproses semua!
Saatnya isi kembali dengan input berikutnya.

Disk
(Di Bagian Lain)

Demo: External Sort-Merge, dengan 3-Way Merge

Kita definisikan sebuah blok/chunk terdiri dari 4 angka

Disk



Memori (hanya muat 8 angka)

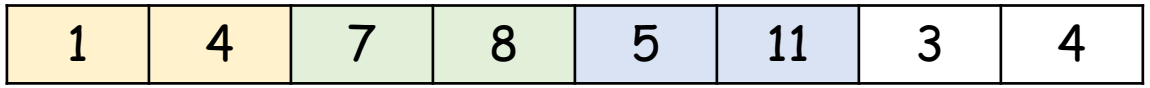
Ada **read buffer** yang sudah diproses semua!
Saatnya isi kembali dengan input berikutnya.

Disk
(Di Bagian Lain)

Demo: External Sort-Merge, dengan 3-Way Merge

Kita definisikan sebuah blok/chunk terdiri dari 4 angka

Disk



Memori (hanya muat 8 angka)

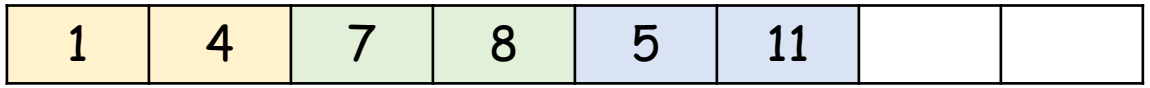
Dan seterusnya … semoga Anda paham ☺

Disk
(Di Bagian Lain)

Demo: External Sort-Merge, dengan 3-Way Merge

Kita definisikan sebuah blok/chunk terdiri dari 4 angka

Disk



Memori (hanya muat 8 angka)

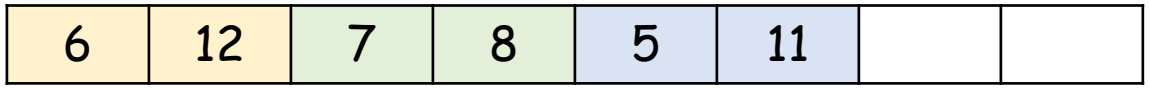
Dan seterusnya … semoga Anda paham ☺

Disk (Di Bagian Lain)

Demo: External Sort-Merge, dengan 3-Way Merge

Kita definisikan sebuah blok/chunk terdiri dari 4 angka

Disk



Memori (hanya muat 8 angka)

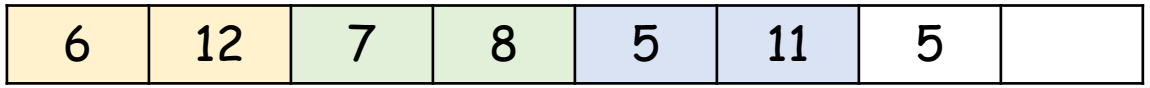
Dan seterusnya … semoga Anda paham ☺

Disk (Di Bagian Lain)

Demo: External Sort-Merge, dengan 3-Way Merge

Kita definisikan sebuah blok/chunk terdiri dari 4 angka

Disk



Memori (hanya muat 8 angka)

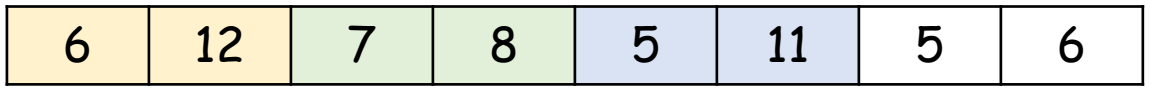
Dan seterusnya … semoga Anda paham ☺

Disk (Di Bagian Lain)

Demo: External Sort-Merge, dengan 3-Way Merge

Kita definisikan sebuah blok/chunk terdiri dari 4 angka

Disk



Memori (hanya muat 8 angka)

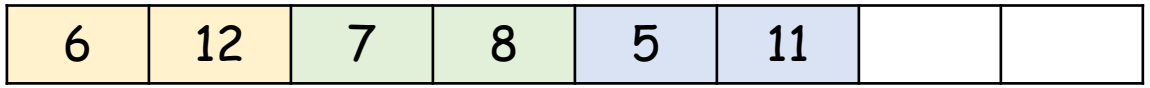
Dan seterusnya … semoga Anda paham ☺

Disk (Di Bagian Lain)

Demo: External Sort-Merge, dengan 3-Way Merge

Kita definisikan sebuah blok/chunk terdiri dari 4 angka

Disk



Memori (hanya muat 8 angka)

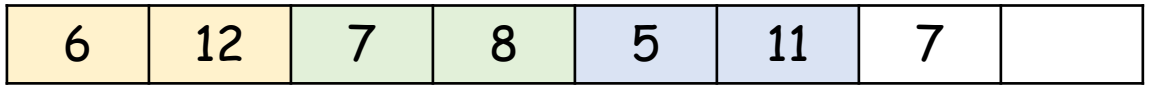
Dan seterusnya … semoga Anda paham ☺

Disk (Di Bagian Lain)

Demo: External Sort-Merge, dengan 3-Way Merge

Kita definisikan sebuah blok/chunk terdiri dari 4 angka

Disk



Memori (hanya muat 8 angka)

Dan seterusnya … semoga Anda paham ☺

Disk (Di Bagian Lain)

Demo: External Sort-Merge, dengan 3-Way Merge

Kita definisikan sebuah blok/chunk terdiri dari 4 angka

Disk

Memori (hanya muat 8 angka)

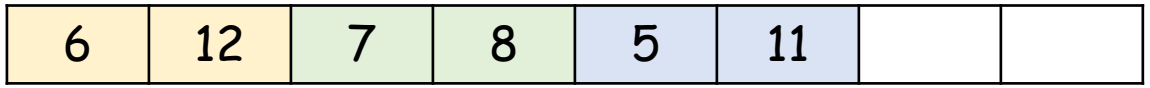
Dan seterusnya … semoga Anda paham ☺

Disk (Di Bagian Lain)

Demo: External Sort-Merge, dengan 3-Way Merge

Kita definisikan sebuah blok/chunk terdiri dari 4 angka

Disk



Memori (hanya muat 8 angka)

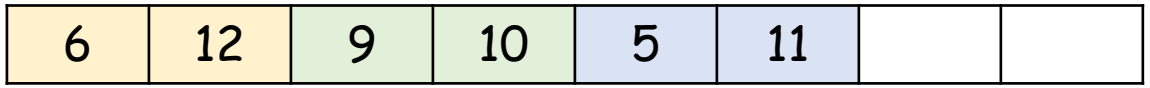
Dan seterusnya … semoga Anda paham ☺

Disk (Di Bagian Lain)

Demo: External Sort-Merge, dengan 3-Way Merge

Kita definisikan sebuah blok/chunk terdiri dari 4 angka

Disk



Memori (hanya muat 8 angka)

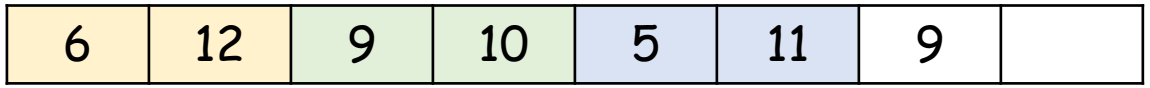
Dan seterusnya … semoga Anda paham ☺

Disk (Di Bagian Lain)

Demo: External Sort-Merge, dengan 3-Way Merge

Kita definisikan sebuah blok/chunk terdiri dari 4 angka

Disk



Memori (hanya muat 8 angka)

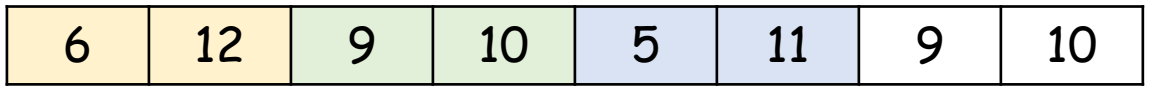
Dan seterusnya … semoga Anda paham ☺

Disk (Di Bagian Lain)

Demo: External Sort-Merge, dengan 3-Way Merge

Kita definisikan sebuah blok/chunk terdiri dari 4 angka

Disk



Memori (hanya muat 8 angka)

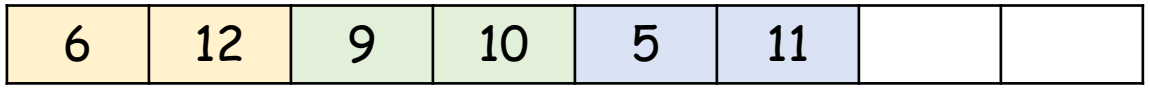
Dan seterusnya … semoga Anda paham ☺

Disk (Di Bagian Lain)

Demo: External Sort-Merge, dengan 3-Way Merge

Kita definisikan sebuah blok/chunk terdiri dari 4 angka

Disk



Memori (hanya muat 8 angka)

Dan seterusnya … semoga Anda paham ☺

Disk (Di Bagian Lain)

Demo: External Sort-Merge, dengan 3-Way Merge

Kita definisikan sebuah blok/chunk terdiri dari 4 angka

Disk

Memori (hanya muat 8 angka)

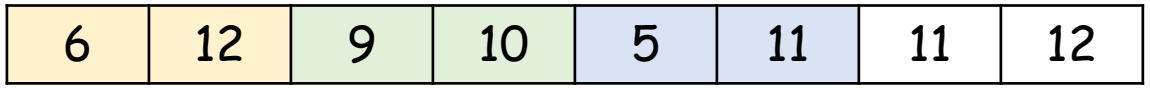
Dan seterusnya … semoga Anda paham ☺

Disk (Di Bagian Lain)

Demo: External Sort-Merge, dengan 3-Way Merge

Kita definisikan sebuah blok/chunk terdiri dari 4 angka

Disk



Memori (hanya muat 8 angka)

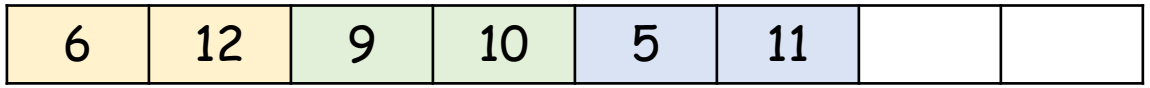
Dan seterusnya … semoga Anda paham ☺

Disk (Di Bagian Lain)

Demo: External Sort-Merge, dengan 3-Way Merge

Kita definisikan sebuah blok/chunk terdiri dari 4 angka

Disk

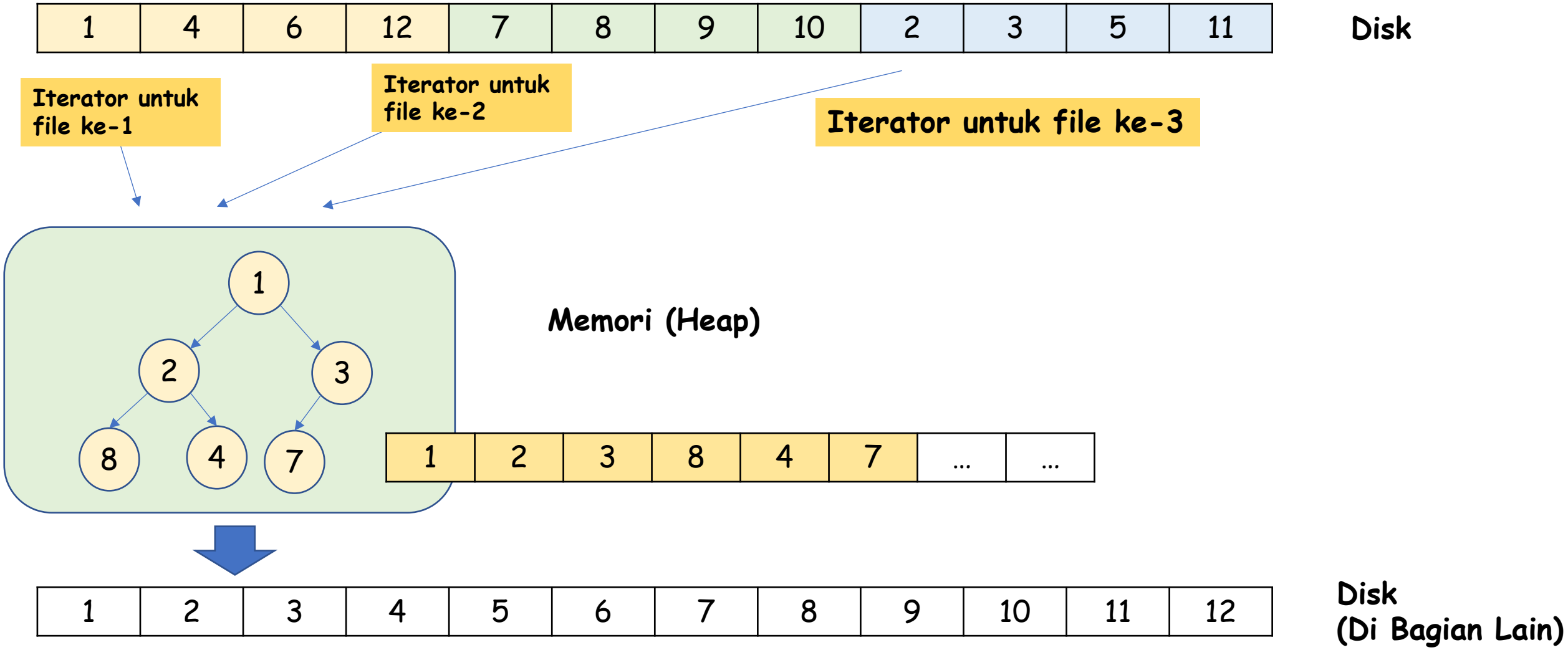


Memori (hanya muat 8 angka)

Dan seterusnya … semoga Anda paham ☺

Disk (Di Bagian Lain)

Implementasi K-Way Merge dengan HEAP



Ok, Jadi bagaimana menggunakan external sort untuk indexing?

Blocked Sort-Based Indexing (BSBI)

BSBINDEXCONSTRUCTION()
1 $n \leftarrow 0$
2 **while** (all documents have not been processed) **do**
3 $n \leftarrow n + 1$
4 *block* ← PARSENEXTBLOCK()
5 BSBI-INVERT(*block*)
6 WRITEBLOCKTODISK (*block*, *fn*)
7 MERGEBLOCKS(*f*1, . . . , *fn*; *f*merged)

Sumber Pseudocode: Text book kuliah IR

Blocked Sort

(1) Segment the collection into parts of equal size (block)

[Doc1, Doc2, Doc3, Doc4, Doc5, Doc6, Doc7, Doc8, Doc9, Doc10]

[Doc1, Doc2, Doc3, Doc4, Doc5, Doc6, Doc7, Doc8, Doc9, Doc10]

Lalu loop untuk setiap blok tersebut!

BSBINDEXCONSTRUCTIO

```
1 n ← 0
2 while (all documents have not been processed) do
3     n ← n +1
4     block ← PARSENEXTBLOCK()
5     BSBI-INVERT(block)
6     WRITEBLOCKTODISK (block, fn)
7 MERGEBLOCKS( f1, . . . , fn; fmerged)
```

Sumber Pseudocode: Text book kuliah IR

Blocked Sort-

(2) Grab the next block, turns them into a stream of **term-docID pairs**

Ex. from block **[Doc1, Doc2, Doc3]**, we get
[(be, Doc1), (not, Doc2), (to, Doc2), (be, Doc3), (to, Doc3)]

BSBINDEXCONSTRUCTIO

1 $n \leftarrow 0$
2 **while** (all documents have
3 $n \leftarrow n + 1$
4 *block* ← PARSENEXTBLOCK()
5 BSBI-INVERT(*block*)
6 WRITEBLOCKTODISK (*block*, *fn*)
7 MERGEBLOCKS(*f*1, . . . , *fn*; *f*merged)

Sumber Pseudocode: Text book kuliah IR

Blocked Sort-B

BSBINDEXCONSTRUCTION(
1 $n \leftarrow 0$
2 **while** (all documents hav
3 $n \leftarrow n + 1$
4 *block* ← PARSENEXTBLO
5 BSBI-INVERT(*block*)
6 WRITEBLOCKTODISK (*block*, *fn*)
7 MERGEBLOCKS(*f*1, . . . , *fn*; *f*merged)

(3) Sort the **termID-docID pairs** of that block **in memory**, and invert the block!

Dictionary that maps **term** to **TermID**:
{(be, 1), (not, 2), (to, 3)}

Ex. from block **[Doc1, Doc2, Doc3]**, we get
[(1, Doc1), (2, Doc2), (3, Doc2), (1, Doc3), (3, Doc3)]

After sorting, we get
[(1, Doc1), (1, Doc3), (2, Doc2), (3, Doc2), (3, Doc3)]

➔ **1,2 --> Doc1, Doc3**
2,1 --> Doc2
3,2 --> Doc2, Doc3

To make index construction more efficient, we represent terms as termIDs, where each *termID* is a unique serial number. We can build the **mapping from terms to termIDs** on the fly while we are processing the collection.

Mapping term-termID disimpan pada **in-memory data structure** dan sharing ke semua blok!

Sumber Pseudocode: Text book kuliah IR

Blocked Sort-Based Indexing (BSBI)

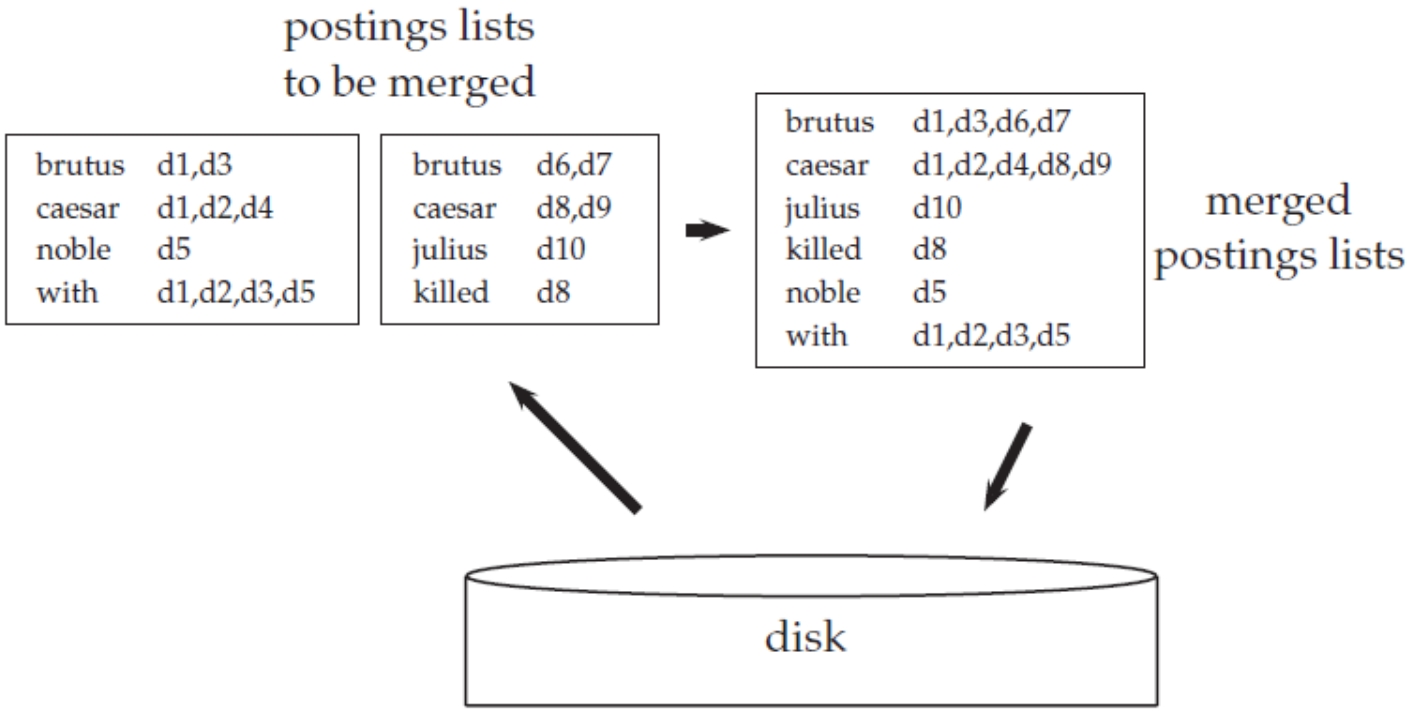
```
BSBINDEXCONSTRUCTION()
1 n ← 0
2 while (all documents have not been processed) do
3     n ← n +1
4     block ← PARSENEXTBLOCK()
5     BSBI-INVERT(block)
6     WRITEBLOCKTODISK (block, fn)
7 MERGEBLOCKS( f1, . . . , fn; fmerged)
```

(4) Store intermediate inverted (and sorted) blocks in files f1, f2, f3, …, fn

(5) Gabung semua intermediate results dengan menggunakan **external merge (n-way merge)**

Sumber Pseudocode: Text book kuliah IR

Blocked Sort-Based Indexing (BSBI)



► **Figure 4.3** Merging in blocked sort-based indexing. Two blocks ("postings lists to be merged") are loaded from disk into memory, merged in memory ("merged postings lists") and written back to disk. We show terms instead of termIDs for better readability.

Sumber Pseudocode: Text book kuliah IR

Blocked Sort-Based Indexing (BSBI)

```
BSBINDEXCONSTRUCTION()
1 n ← 0
2 while (all documents have not been
3     n ← n +1
4     block ← PARSENEXTBL
5     BSBI-INVERT(block)
6     WRITEBLOCKTODISK (block, fn)
7 MERGEBLOCKS( f1, . . . , fn; fmerged)
```

Step dengan **Time Complexity** paling tinggi dari BSBI adalah saat melakukan sorting terhadap **sequence of termID-docID pairs**.

Time Complexity: O(T log T)
Dimana **T** adalah banyaknya pasangan termID-docID pada collection (upper bound banyaknya item yang perlu di-sort).

Namun, *actual indexing time* biasanya didominasi oleh proses parsing dokumen dan proses *merging blocks*.

Sumber Pseudocode: Text book kuliah IR

BSBI: Assumption muat di memory, kayak kalau dictionary muat di memory apa enggak

Single-Pass In-Memory Indexing (SPIMI)

- BSBI needs a data structure for mapping terms to termIDs.
- When using BSBI, we assume that this data structure (or a dictionary) can be kept in memory.
- For very large collections, the dictionary does not fit into memory.

- SPIMI:

  Separate dictionaries are generated for each block. Hence, we don't need an in-memory data structure that maintains term-termID mappings across blocks.

Single-Pass In-Memory Indexing (SPIMI)

BSBI maintain Term-TermID accross all block

SPIMI dictionary ada di masing-masing2 block.

BSBI semua block punya 1 dict

SPIMIINDEXCONSTRUCTION()
1 $n \leftarrow 0$
2 **while** (all documents have not been processed) **do**
3 $n \leftarrow n$ +1
4 *token_stream* ← PARSENEXTBLOCK()
5 *sorted_terms, dictionary* ← SPIMI-INVERT(*token_stream*)
6 WRITEBLOCKTODISK (*sorted_terms, dictionary, fn*)
7 MERGEBLOCKS(*f*1, . . . , *fn*; *f*merged)

Single-Pass I

Sebuah posting langsung ditambahkan ke dalam posting lists; tanpa perlu membuat sequence of termID-docID tuples lalu di-sort

SPIMIINDEXCONSTRUC
1 $n \leftarrow 0$
2 **while** (all documents h
3 $n \leftarrow n + 1$
4 *token_stream* ← PARSENEXTBL ER()
5 *sorted_terms, dictionary* ← SPIMI-INVERT(*token_stream*)
6 WRITEBLOCKTODISK (*sorted_terms, dictionary, fn*)
7 MERGEBLOCKS(*f*1, . . . , *fn*; *f*merged)

SPIMI-INVERT(*token_stream*)
1 *dictionary* = NEWHASH()
2 **while** (free memory available) **do**
3 *token* ← *next*(*token_stream*)
4 **if** *term*(*token*) **not in** *dictionary* **then**
5 *postings_list* = ADDTODICTIONARY(*dictionary*, *term*(*token*))
6 **else**
7 *postings_list* = GETPOSTINGSLIST(*dictionary*, *term*(*token*))
8
9 **if** *full*(*postings_list*) **then**
10 *postings_list* = DOUBLEPOSTINGSLIST(*dictionary*, *term*(*token*))
11 ADDTOPOSTINGSLIST(*postings_list*, *docID*(*token*))
12 *sorted_terms* ← SORTTERMS(*dictionary*)
13 **return** *sorted_terms, dictionary*

kalau ketemu suatu kata, di cek termnya ada di dictionary ada apa engga.

Kalau belum ada masukkin ke dictionary

digedein posting list

bukan sorting pasangan term-id

Yang di sorting adalah TERM aja (KEY nya saja yang di sort)

Makanya dia O(T) karena yang di sort itu ucman Termnya aja (T)

Single-Pass I

SPIMIINDEXCONSTRUC
1 $n \leftarrow 0$
2 **while** (all documents h
3 $n \leftarrow n$ +1
4 *token_stream* ← PARSENEXTBL CK()
5 *sorted_terms, dictionary* ← SPIMI-INVERT(*token_stream*)
6 WRITEBLOCKTODISK (*sorted_terms, dictionary, fn*)
7 MERGEBLOCKS(*f*1, . . . , *fn*; *f*merged)

SPI
1 *di*
2 **w**
3
4
5
6
7
8
9
10
11 ADDTOPOSTINGSLIST(*postings_list, docID*(*token*))
12 *sorted_terms* ← SORTTERMS(*dictionary*)
13 **return** *sorted_terms, dictionary*

Ini merupakan proses sort terhadap **terms (keys)** di dictionary agar terurut secara leksikografis (bukan sort terhadap sequence of termID-docID seperti pada BSBI!).

Proses sorting ini bisa dianggap less significant (sorting pada daftar sepanjang banyaknya term) dari pada proses sorting terhadap sequence of termID-docID pairs pada BSBI.

Ingat, banyaknya term << banyaknya pasangan termID-docID.

Sorting daftar term pada keys dictionary penting untuk final merging step.

Single-Pass I

Time complexity adalah **O(T)** karena **tidak ada proses sorting sequence of termID-docID pairs**, dan secara umum linier terhadap banyaknya postings (**T**) pada collection.

SPIMIINDEXCONSTRUC
1 $n \leftarrow 0$
2 **while** (all documents h
3 $n \leftarrow n + 1$
4 *token_stream* ← PARSENEXTBL CK()
5 *sorted_terms, dictionary* ← SPIMI-INVERT(*token_stream*)
6 WRITEBLOCKTODISK (*sorted_terms, dictionary, fn*)
7 MERGEBLOCKS(*f*1, . . . , *fn*; *f*merged)

```
SPIMI-INVERT(token_stream)
1 dictionary = NEWHASH()
2 while (free memory available) do
3     token ← next(token_stream)
4     if term(token) not in dictionary then
5         postings_list = ADDTODICTIONARY(dictionary, term(token))
6     else
7         postings_list = GETPOSTINGSLIST(dictionary, term(token))
8
9     if full(postings_list) then
10        postings_list = DOUBLEPOSTINGSLIST(dictionary, term(token))
11    ADDTOPOSTINGSLIST(postings_list, docID(token))
12 sorted_terms ← SORTTERMS(dictionary)
13 return sorted_terms, dictionary
```

Distributed Indexing

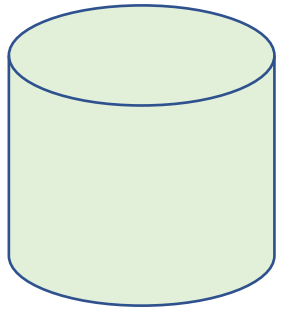
Distributed Indexing?

- World Wide Web berukuran sangat besar, sehingga hampir tidak mungkin kita melakukan indexing untuk semuanya dengan hanya **sebuah mesin atau komputer**.

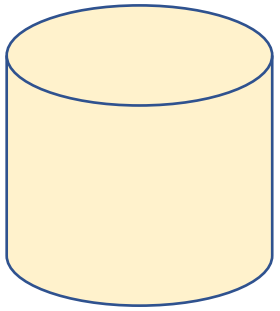
- Perlu ada sebuah *distributed computing framework* yang mampu menghasilkan *distributed index* yang terbagi ke banyak mesin.

Term-Partitioned Index

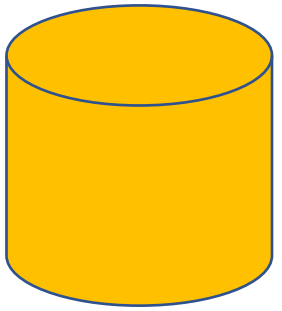
Dictionary is partitioned into subsets, where each subset resides at a node.



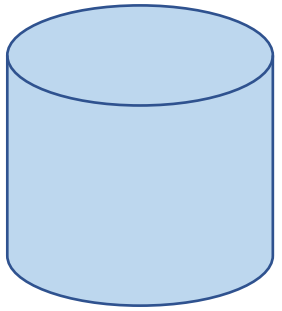
Postings Lists untuk terms dengan awalan **a – f**



Postings Lists untuk terms dengan awalan **g – l**



Postings Lists untuk terms dengan awalan **m – r**

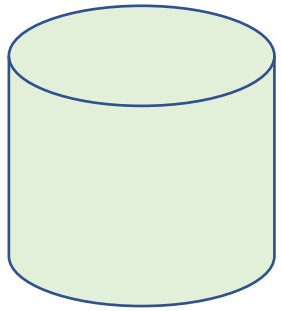


Postings Lists untuk terms dengan awalan **s – z**

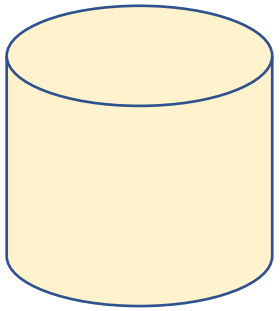
Apa kelebihan dan kekurangannya? Coba kaitkan apa yang terjadi jika query terdiri dari single term, dan jika query multi-terms (phrase query)?

Document-Partitioned Index

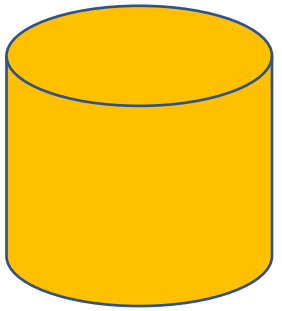
Each node contains the index for a subset of all documents.



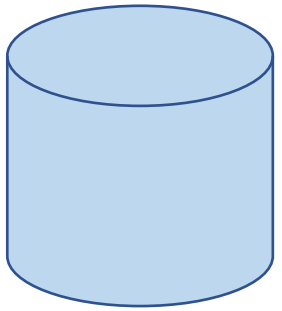
be,20 -> d10, d45, …
to,45 -> d10, d23, …
…



be,10 -> d3, d77, …
not,89 -> d3, d65, …
…



be,45 -> d34, d98, …
to,89 -> d50, d233, …
…



apply,23 -> d55, d85, …
not,84 -> d56, d85, …
…

Sebuah query akan didistribusikan ke semua node. Hasil dari semua node kemudian di-merge untuk kemudian ditampilkan ke pengguna.

Statistik seperti **df** (*document frequency*) untuk sebuah term perlu usaha lebih untuk menghitungnya (dapat dilakukan dengan proses *background*).

Bagaimana menghasilkan Term-Partitioned Index?

Konsep Map-Reduce

This computing paradigm breaks a large computing problem into smaller parts by recasting it in terms of manipulation of **key-value pairs**.

In this indexing case, the **key-value pair** is represented as a **termID-docID** pair.

Map Phase
Secara paralel di banyak node (yang dipilih sebagai parser)

D1: to be good be
D2: be a good student

[<a, D2>, <be, D1>, <be, D1>, <be, D2>, <good, D1>, <good, D2>, <student, D2>, <to, D1>]

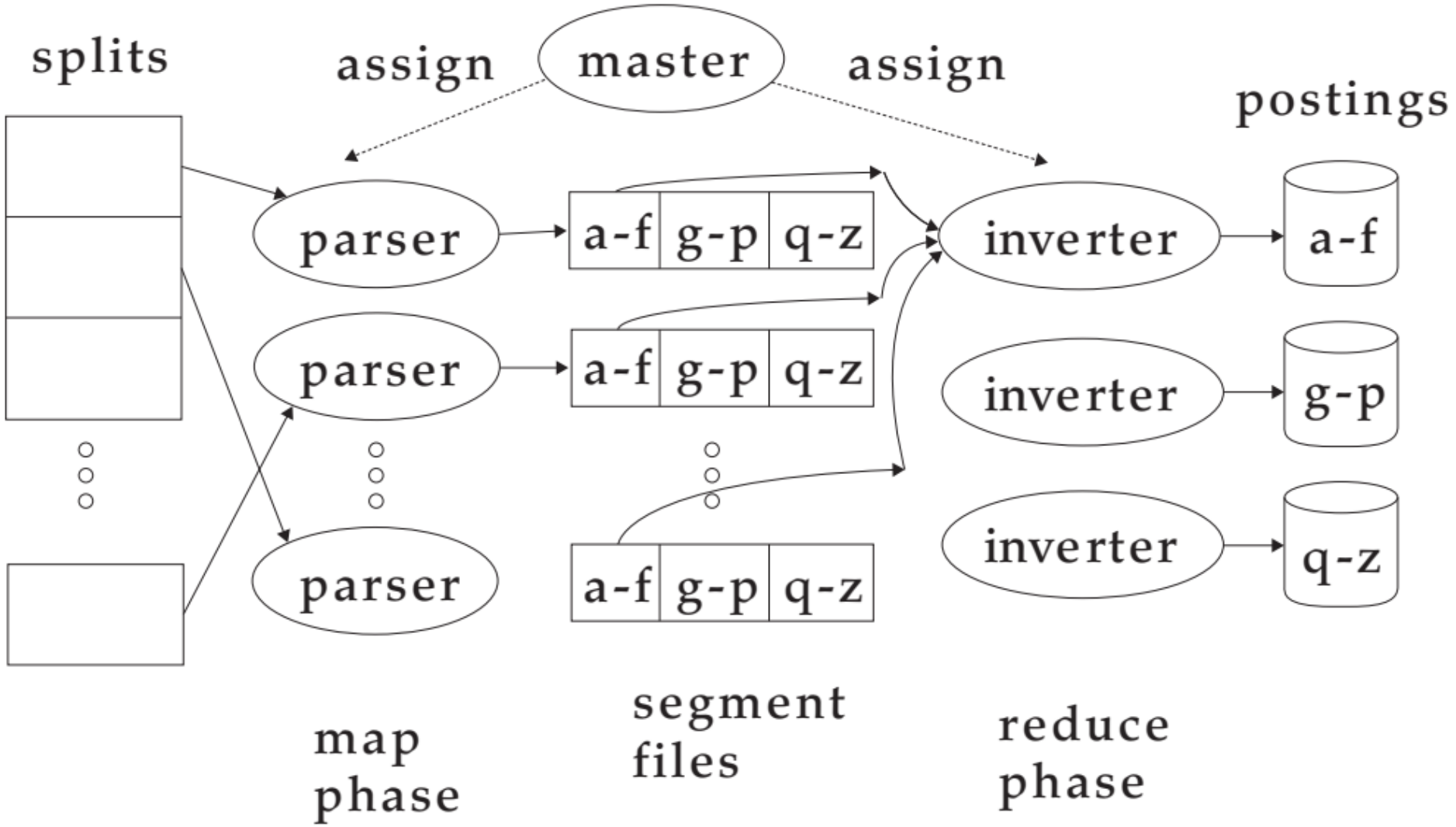
Split data into key-value pairs

Reduce Phase
Secara paralel di banyak node (yang dipilih sebagai inverter)

[<a, (D2)>, <be, (D1, D2, D2)>, <good, (D1, D2)>, <student, (D2)>, <to, (D1)>]

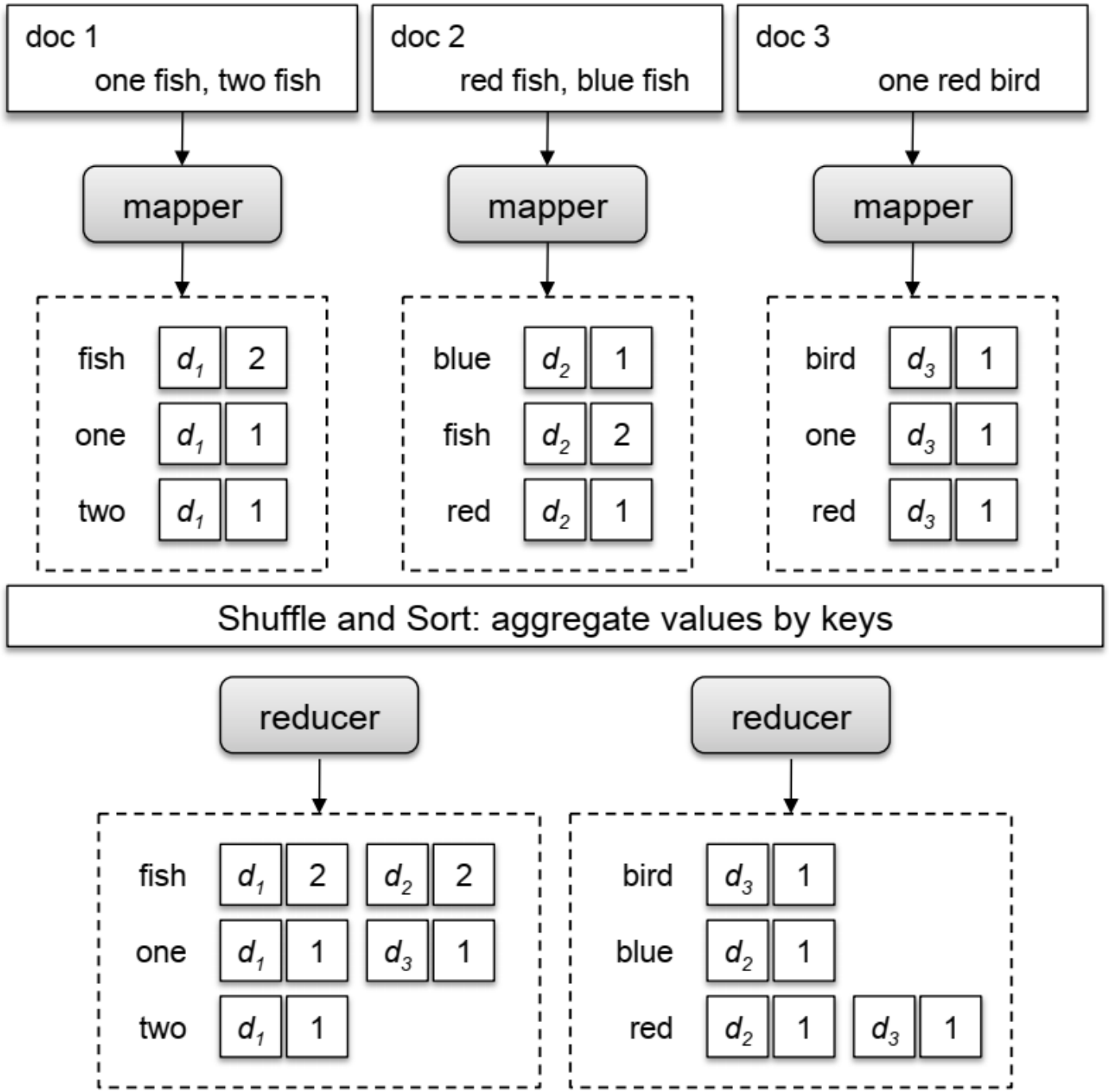
[<a, (D2:1)>, <be, (D1:1, D2:2)>, <good, (D1:1, D2:1)>, <student, (D2:1)>, <to, (D1:1)>]

All values of a particular key are stored close together



► **Figure 4.5** An example of distributed indexing with MapReduce. Adapted from Dean and Ghemawat (2004).

Gambar dari Text Book


doc 1
one fish, two fish
doc 2
red fish, blue fish
doc 3
one red bird
mapper
mapper
mapper
fish d_1 2
one d_1 1
two d_1 1
blue d_2 1
fish d_2 2
red d_2 1
bird d_3 1
one d_3 1
red d_3 1
Shuffle and Sort: aggregate values by keys
reducer
reducer
fish d_1 2 d_2 2
one d_1 1 d_3 1
two d_1 1
bird d_3 1
blue d_2 1
red d_2 1 d_3 1

Dynamic Indexing

Dynamic Indexing

- Sejauh ini, kita mengasumsikan bahwa koleksi dokumen bersifat statik. Kasus ini OK untuk beberapa dokumen yang tidak berubah atau jarang berubah, seperti buku-buku sastra terdahulu.

- Namun tidak untuk Web, yang selalu berubah sepanjang waktu.

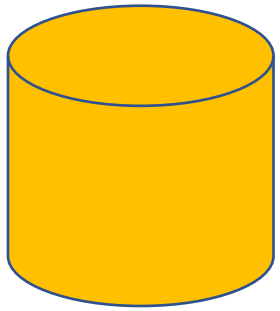
Input sekumpulan termID dan documentID

main index ada di hard disk

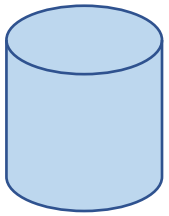
input masuk ke auxiliary index

ketika aux index penuh, maka di merge ke main index.

Auxiliary Index



Main Index
(storage)



Auxiliary Index
(in-memory)

Ketika Auxiliary Index penuh, kita merge isi dari Auxiliary Index dengan Main Index.

Misal, **M** = rata-rata banyaknya kata unik (token type) di sebuah dokumen.

Jika tidak ada Auxiliary Index, proses penambahan/update **sebuah dokumen**, kira-kira membutuhkan **M disk seeks** (sangat mahal).

In-Memory Auxiliary Index mengurangi proses **disk seeks**, yaitu hanya ketika Auxiliary Index penuh dan akan di-merge dengan Main Index.

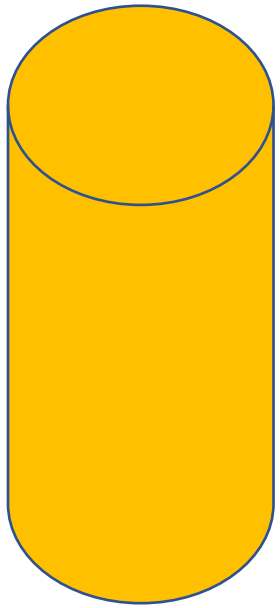
Query

jadi harus di evaluasi dari memori dan hard disk

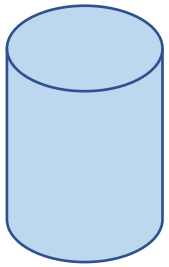
Proses search dilakukan terhadap keduanya, dan hasilnya di-merge

Auxiliary Index

Number of merges: 0



Main Index
(storage)

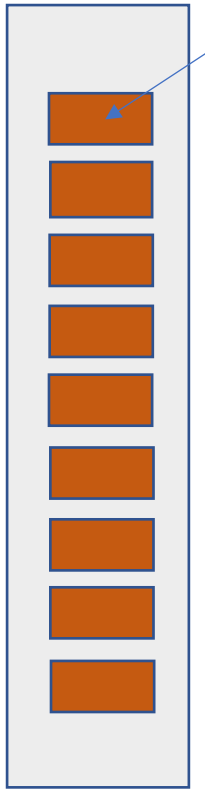


contoh n = 3 jadi kapasitasnya 3

Auxiliary Index
(in-memory)

size = n

ini input



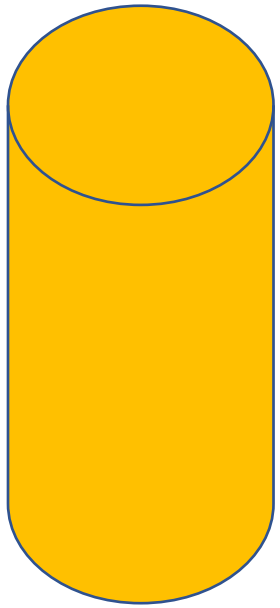
Sebuah posting
(termID, docID)

Daftar postings
yang ada di
collection.

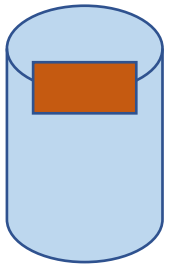
size = T

Auxiliary Index

Number of merges: 0

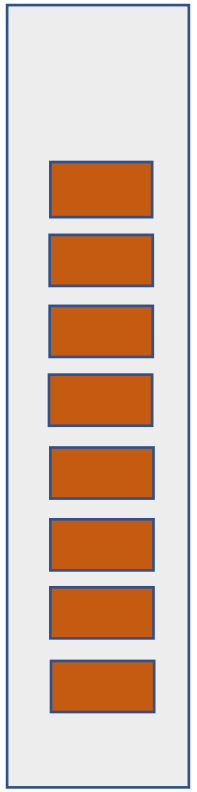


Main Index
(storage)



Auxiliary Index
(in-memory)

size = n

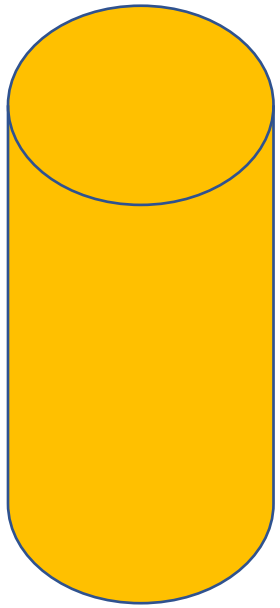


Daftar postings yang ada di collection.

size = T

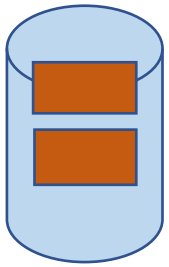
Auxiliary Index

Number of merges: 0



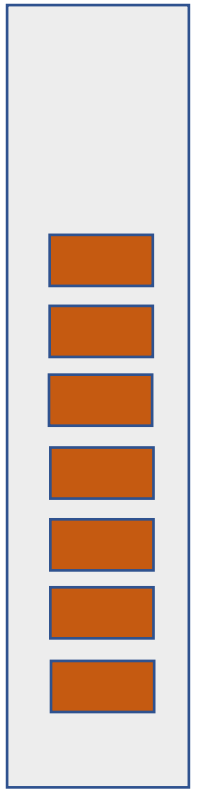
Main Index
(storage)

Mergingnya n
+ n operasi



Auxiliary Index
(in-memory)

size = n

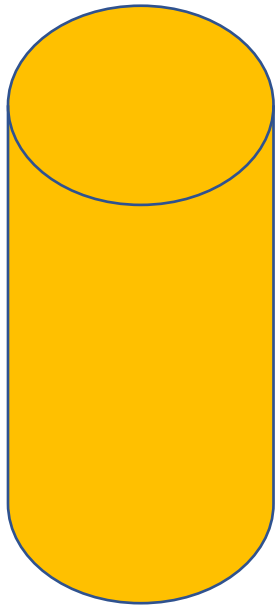


Daftar postings
yang ada di
collection.

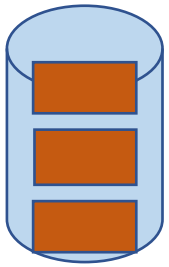
size = T

Auxiliary Index

Number of merges: 0

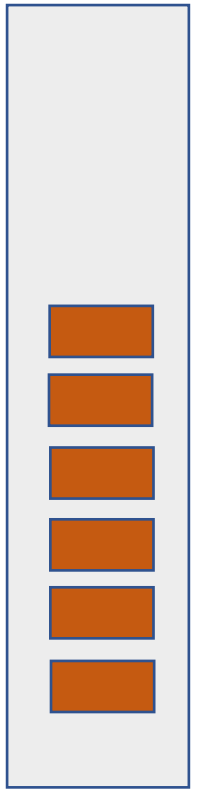


Main Index
(storage)



Auxiliary Index
(in-memory)

size = n



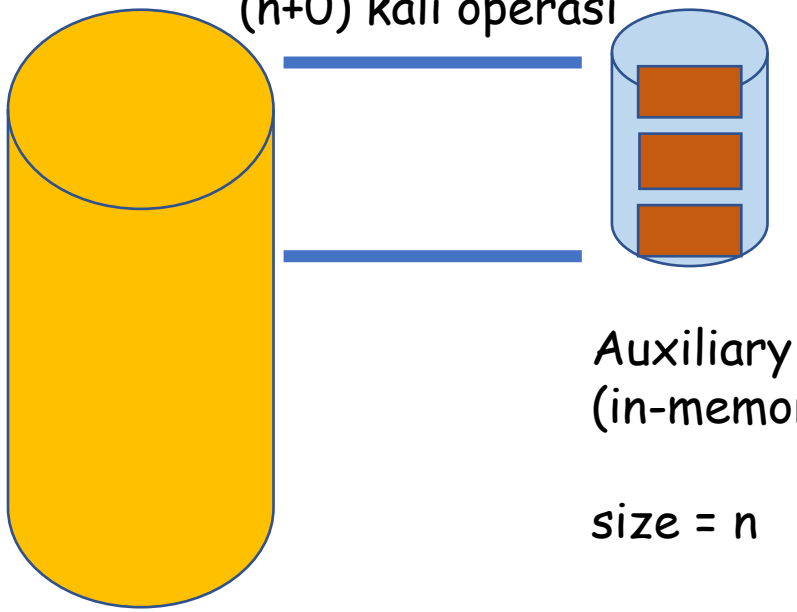
Daftar postings yang ada di collection.

size = T

Auxiliary Index

Number of merges: 1

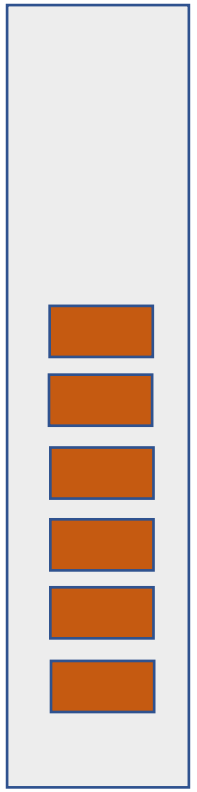
Lakukan Merging!
(n+0) kali operasi



Auxiliary Index
(in-memory)

size = n

Main Index
(storage)

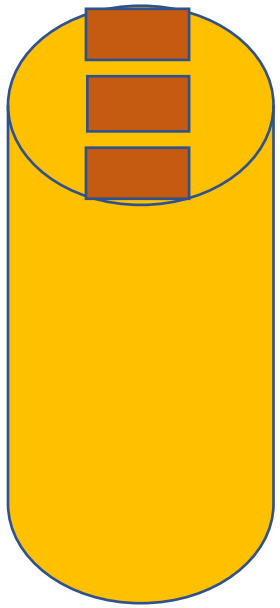


Daftar postings yang ada di collection.

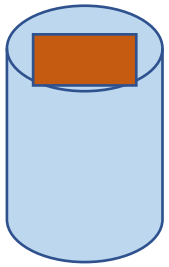
size = T

Auxiliary Index

Number of merges: 1

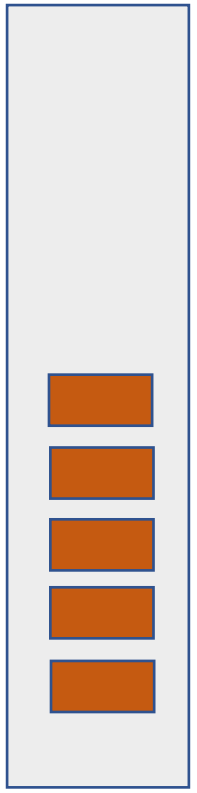


Main Index
(storage)



Auxiliary Index
(in-memory)

size = n

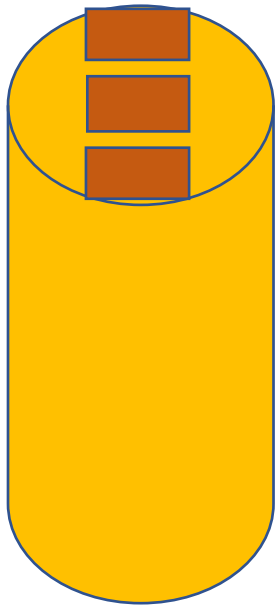


Daftar postings yang ada di collection.

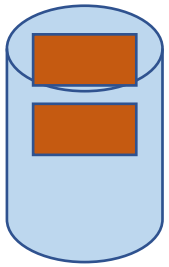
size = T

Auxiliary Index

Number of merges: 1

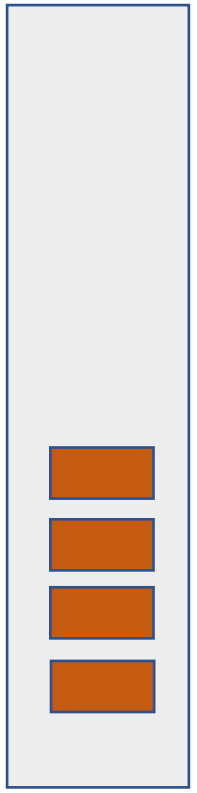


Main Index
(storage)



Auxiliary Index
(in-memory)

size = n

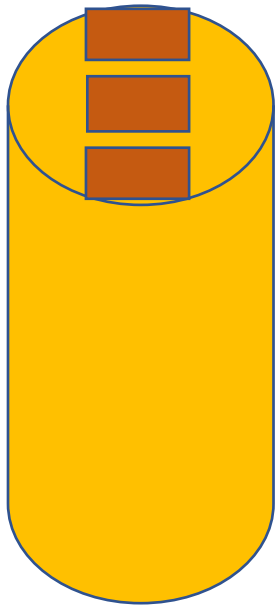


Daftar postings yang ada di collection.

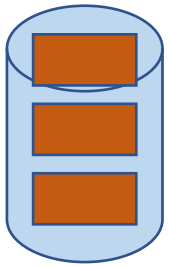
size = T

Auxiliary Index

Number of merges: 1

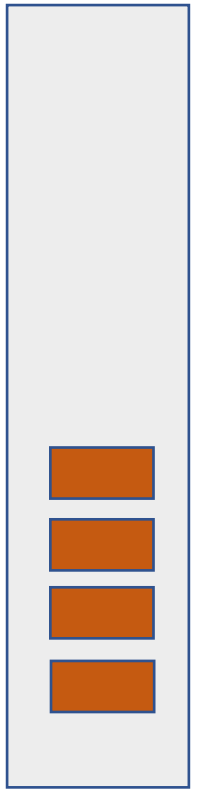


Main Index
(storage)



Auxiliary Index
(in-memory)

size = n



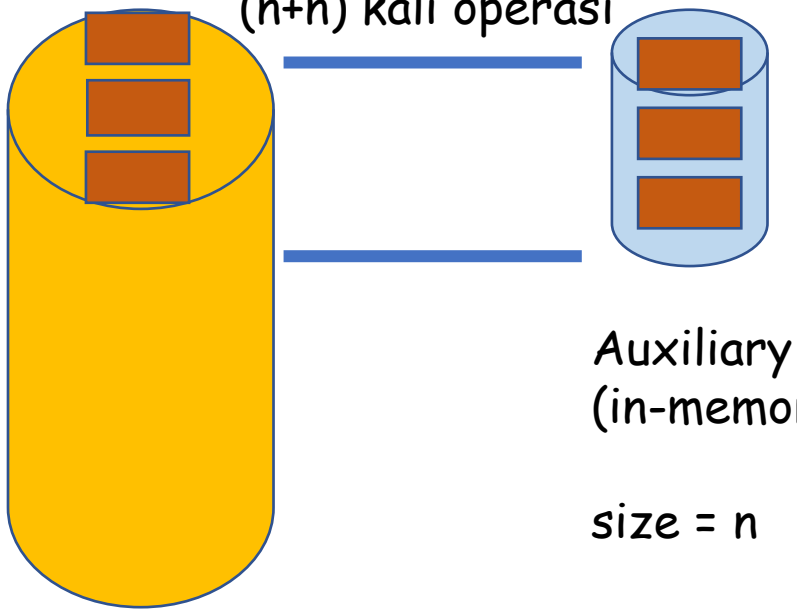
Daftar postings yang ada di collection.

size = T

Auxiliary Index

Number of merges: 2

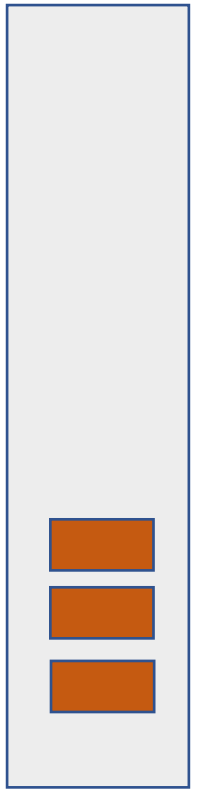
Lakukan Merging!
(n+n) kali operasi



Auxiliary Index
(in-memory)

size = n

Main Index
(storage)

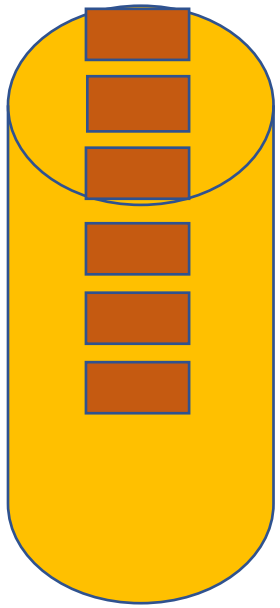


Daftar postings yang ada di collection.

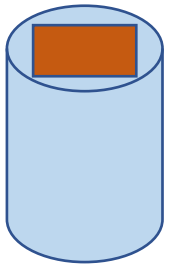
size = T

Auxiliary Index

Number of merges: 2



Main Index
(storage)



Auxiliary Index
(in-memory)

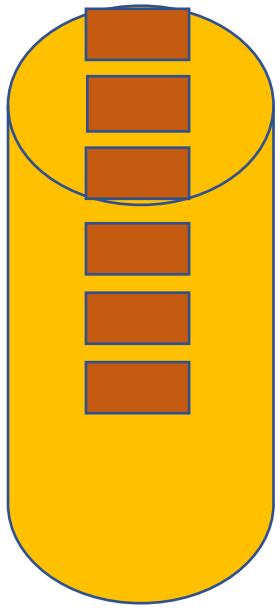
size = n

Daftar postings yang ada di collection.

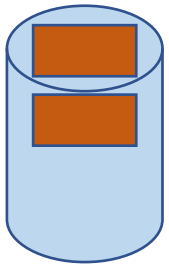
size = T

Auxiliary Index

Number of merges: 2

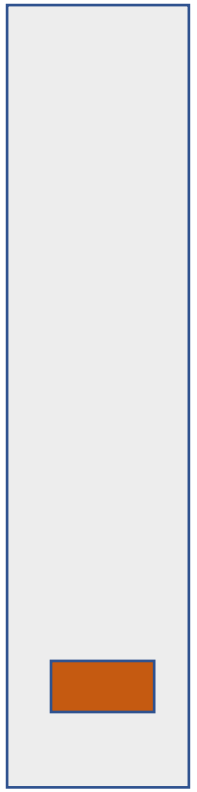


Main Index
(storage)



Auxiliary Index
(in-memory)

size = n

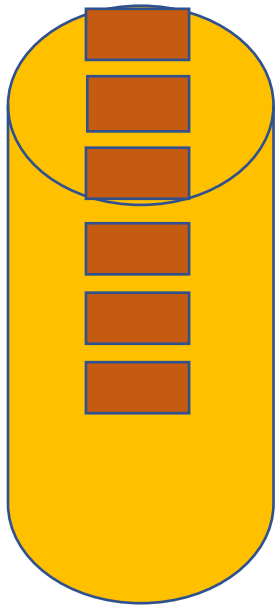


Daftar postings yang ada di collection.

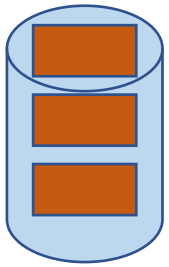
size = T

Auxiliary Index

Number of merges: 2



Main Index
(storage)



Auxiliary Index
(in-memory)

size = n

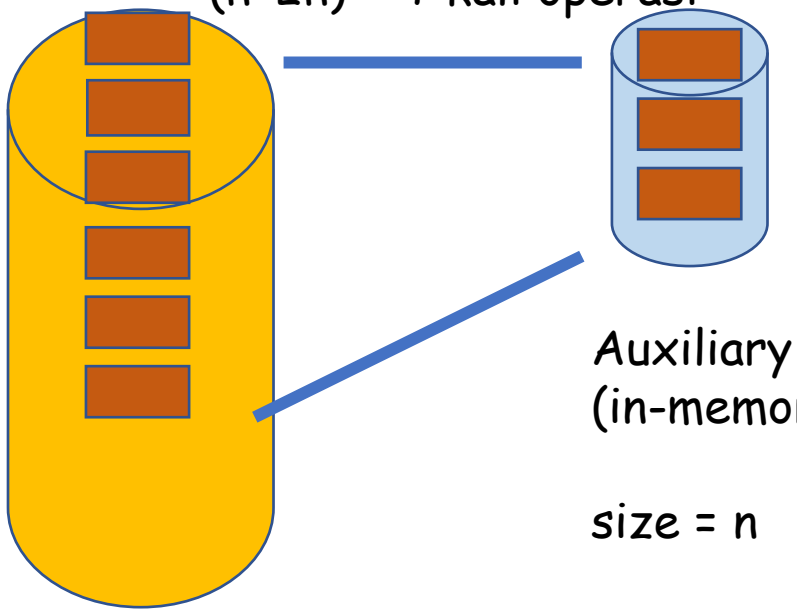
Daftar postings yang ada di collection.

size = T

Auxiliary Index

Lakukan Merging!
(n+2n) = T kali operasi

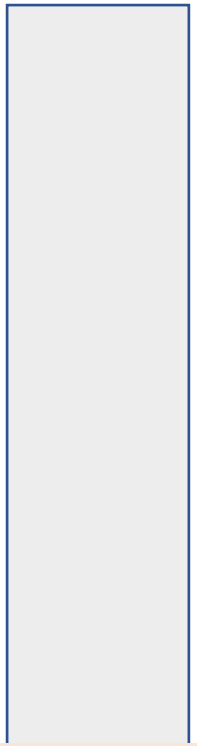
Number of merges: 3



Auxiliary Index
(in-memory)

size = n

Main Index
(storage)



Daftar postings yang ada di collection.

size = T

Jumlah Merging itu Floor(T/n)

Total ada **floor(T / n)** operasi merging.
Sekali merging, upper bound banyaknya operasi adalah **T** kali operasi.

Time complexity = $O(T^2 / n)$

Karena upperbound (last operation) itu pasti merginngya itu operasinya T operasi kompleksitasnya tetep sama

Baca Buku Teks: Logarithmic Merging

- Proses merging dengan auxiliary index sebelumnya dapat dibuat lebih efisien dengan menggunakan **logarithmic merging**.
- Baca buku teks untuk detail terkait hal ini, Chapter 4, Hal 79.
- Proses indexing memang lebih efisien. Namun, hal ini mengorbankan efisiensi dari sisi query processing.

Apa Yang Dilakukan Search Engine Companies?

- Dynamic Indexing sangat complex dan mempunyai trade-off di beberapa pilihan metode.

- Karena hal ini, sebagian besar mengadopsi **reconstruction-from-the-scratch** strategy. Mereka **tidak** membangun index secara dinamis.

- Index baru dibangun secara periodik. Ketika sudah selesai, query diarahkan ke index yang baru.