

The Inverted Index

Sebuah **struktur data** penting yang "berada di balik layar"
modern IR systems

Alfan Farizki Wicaksono

Information Retrieval Problem

- Pujangga dunia, [William Shakespeare](#), menulis banyak buku drama.
- Asumsikan buku-buku tersebut tersimpan secara tekstual dalam sebuah direktori di storage (/mydir/shakespeare).
- Kita ingin mencari buku drama W. Shakespeare yang mengandung kata Brutus **AND** Caesar.

You do a **grepping** process -> a linear scan through text

linear scan

```
grep -R -l "Brutus.*Caesar\|Caesar.*Brutus" /mydir/shakespeare
```

Information Retrieval Problem

Apakah proses grepping (linear scan) bisa dilakukan untuk kasus berikut?

- Koleksi buku yang sangat besar Kalau udah jutaan pasti lama
- Menangani proximity query --> "**Brutus /3 Caesar**" (Brutus muncul dalam jarak maksimal 3 kata dengan Caesar) Kalau udah terpisah setidaknya 3, kalau pakai grep susah
- Ranked Retrieval seperti Bing & Google, dimana setiap dokumen diberikan score agar mengetahui siapa jawaban terbaik.

Simple Index: Incidence Matrix

To avoid a linear scan process, we must index all documents in the collection in advance.

Baris = kata-kata
Kolom = Dokumen

dokumen biasa > term (if it's huge number)

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet 1 menandakan kalau ada di dokumen.	Othello	Macbeth	...
Antony	1	1	0	0	0	1	
Brutus	1	1	0	1	0	0	
Caesar	1	1	0	1	1	1	
Calpurnia	0	1	0	0	0	0	
Cleopatra	1	0	0	0	0	0	
mercy	1	0	1	1	1	1	
worser	1	0	1	1	1	0	
...							

Query: Brutus AND Caesar

Brutus: 1 1 0 1 0 0

Caesar: 1 1 0 1 1 1

AND : 1 1 0 1 0 0 intersection

Results: {Antony and Cleopatra,
Julius Caesar, Hamlet}

► **Figure 1.1** A term-document incidence matrix. Matrix element (t, d) is 1 if the play in column d contains the word in row t , and is 0 otherwise.

This is a Boolean retrieval system!

The Library of Congress, USA



Ada sekitar 170 juta buku.

Misal, #unique words = 500.000

Jadi, berapa banyak bit 1/0 pada incidence matrix:
 $170.000.000 \times 500.000$

Extremely sparse! Terlalu banyak 0 di matrix (> 90%)

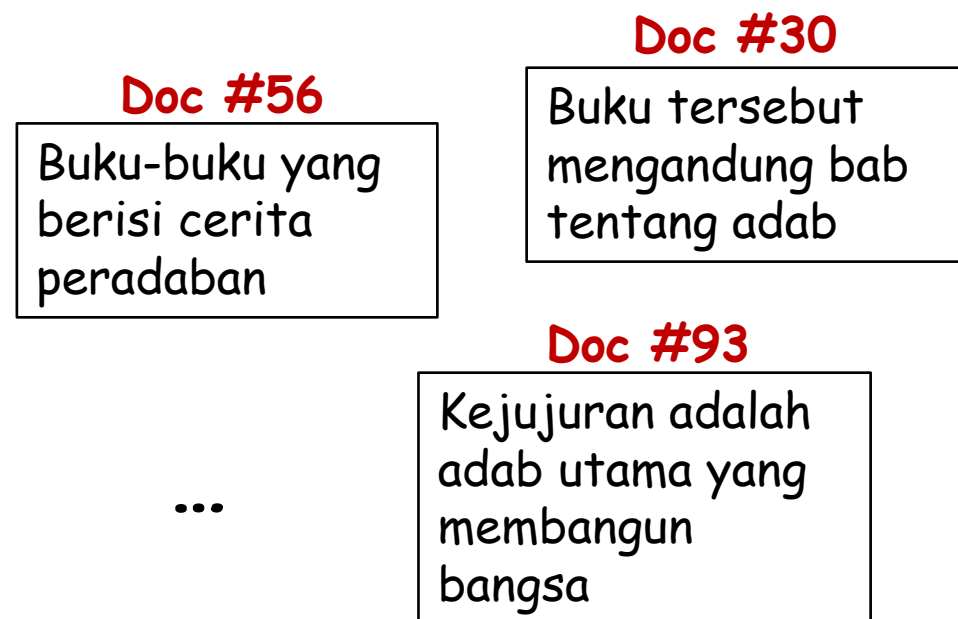
[Terlalu banyak matrix nya in dimension.](#)

We need a better representation or indexing strategy!

Inverted Index

- A term t is associated with a list of all documents that contain t .
- A document is typically identified with an integer.

A collection of documents:



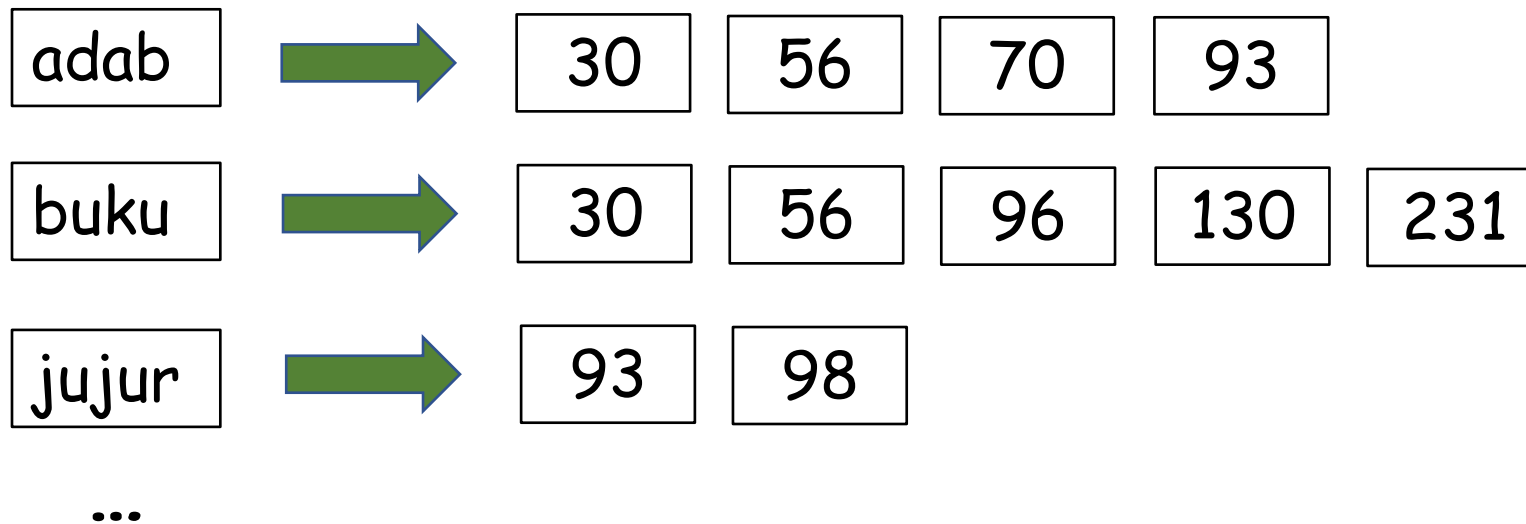
Document direpresentasikan sebagai integer:
kayak linked list



Inverted Index

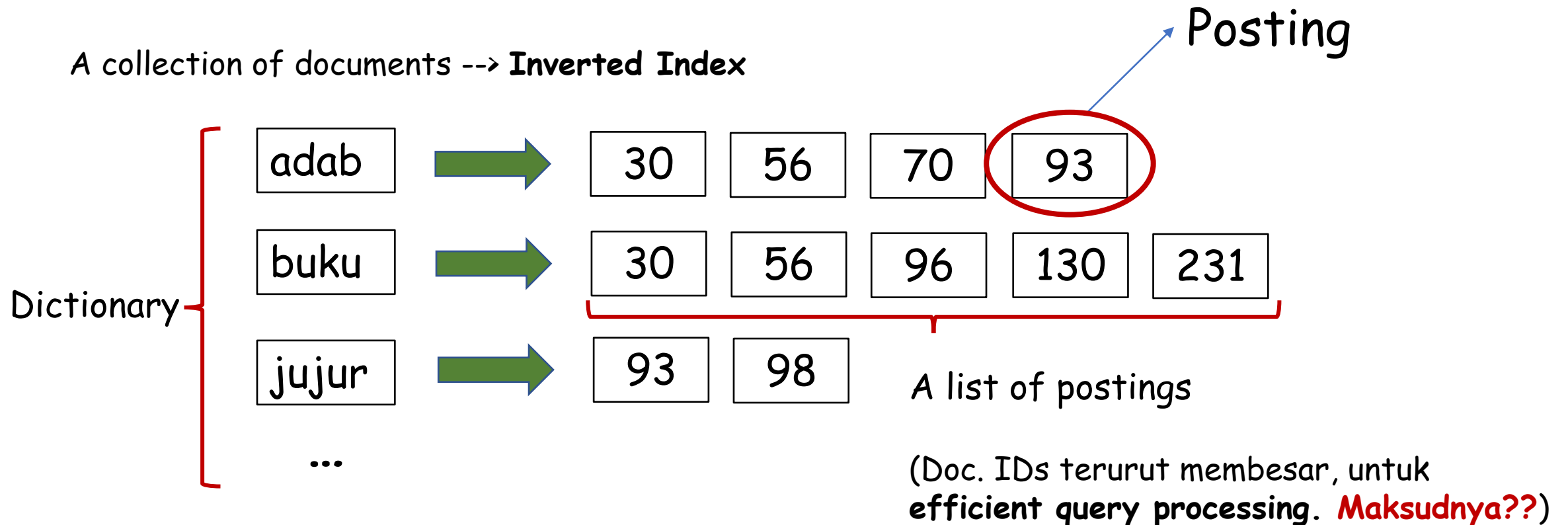
- A term t is associated with a list of all documents that contain t .
- A document is typically identified with *an integer*.

A collection of documents --> **Inverted Index**



Inverted Index

- A term t is associated with a list of all documents that contain t .
- A document is typically identified with **an integer**.

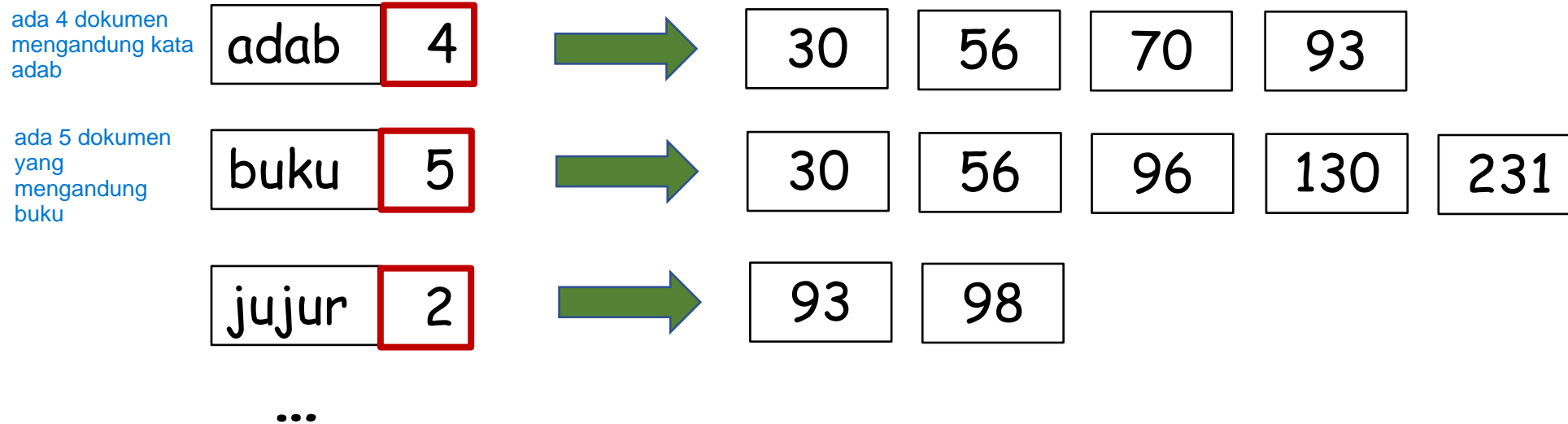


Inverted Index

Sekarang kita masih di topik *Boolean Retrieval*. Kita akan bahas isu ini di kuliah-kuliah mendatang.

The dictionary **sometimes** records some statistics, such as *document frequency* (DF). This information improves search efficiency (query time), and is used for *ranked retrieval*.

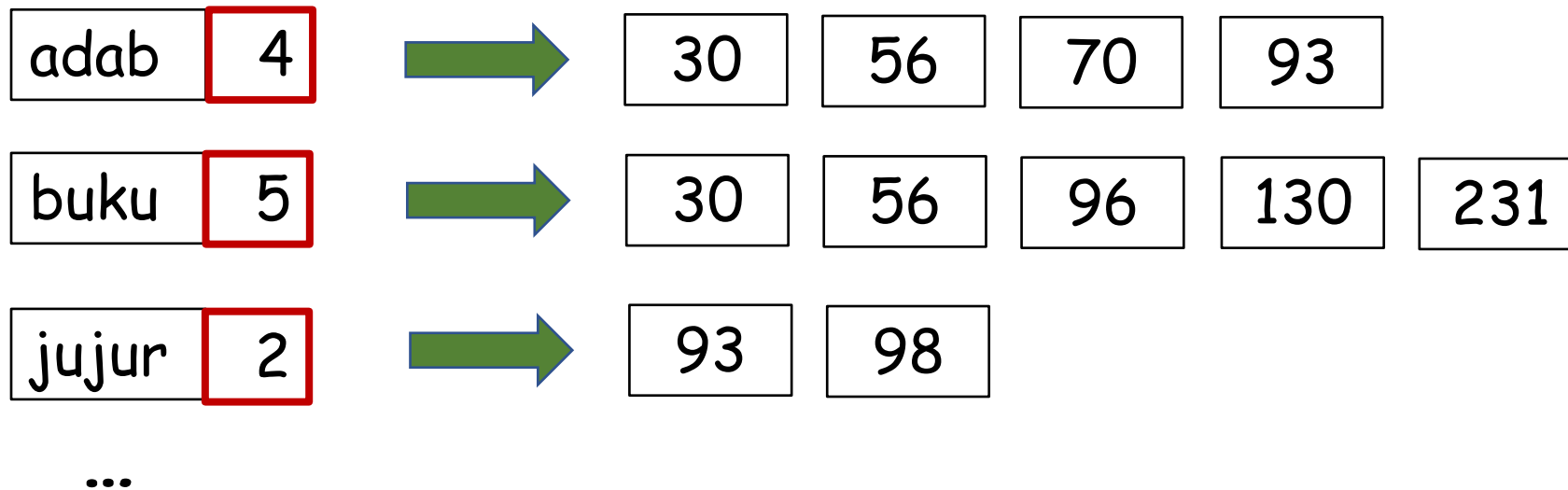
A collection of documents --> **Inverted Index**



How to store dictionary & postings lists?

- Dictionary is normally much smaller than postings lists. karena banyak kata yang terulang
- Dictionary can be kept in **memory**, while postings lists are normally kept on **disk**. dictionary = list of kata

A collection of documents --> **Inverted Index**



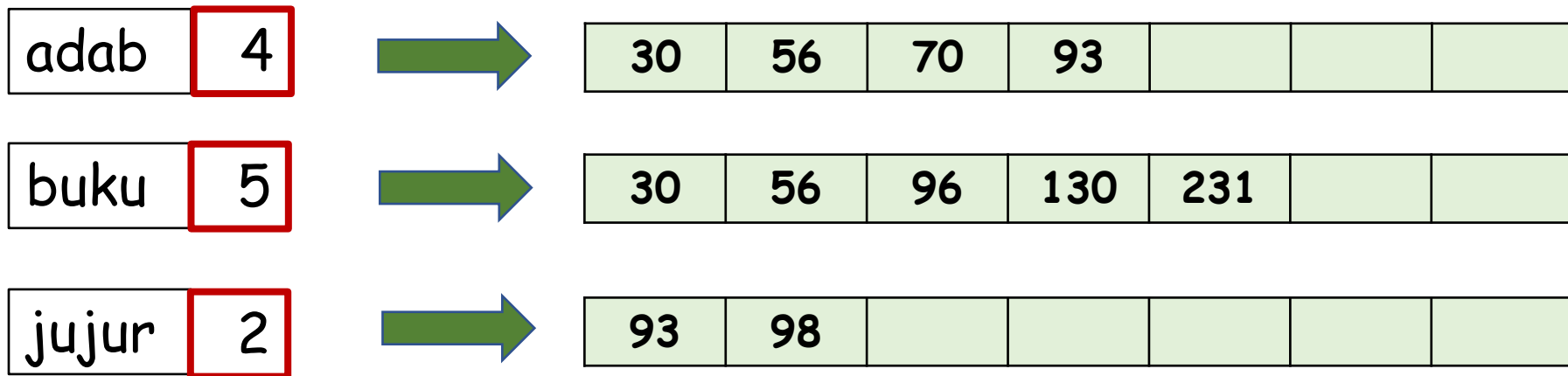
How to store dictionary & postings lists?

Kita tidak bisa menggunakan *fixed-size arrays* untuk menyimpan postings! **Mengapa?**

A collection of documents --> **Inverted Index**

Misal, size = 7

akan banyak list kosong



...

Variable-length Postings List

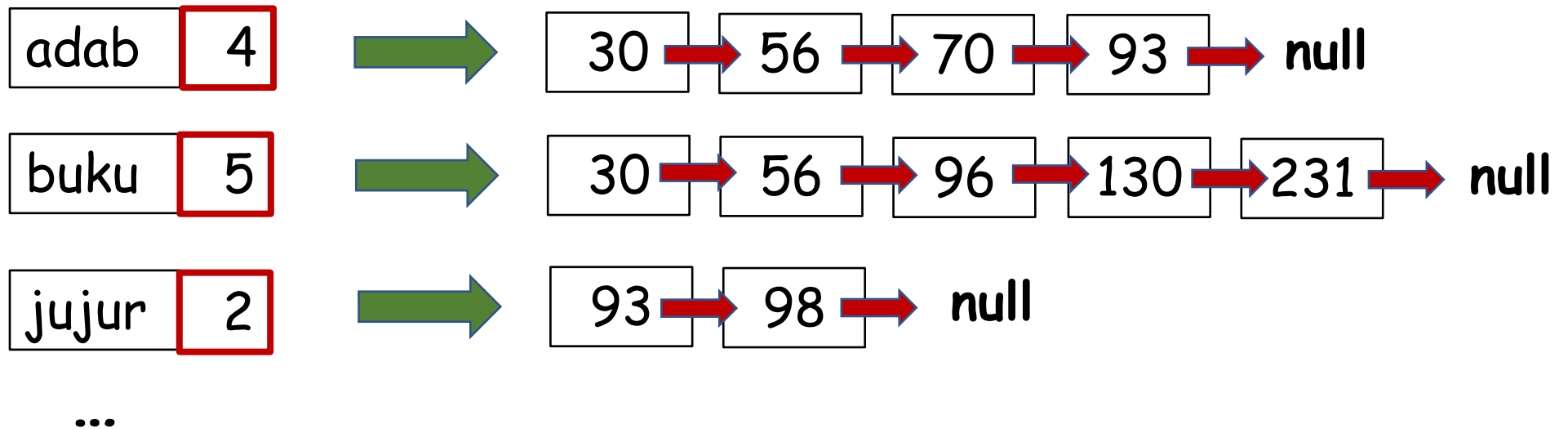
Asumsi posting (jumlah dokumen) muat di memory

Untuk *in-memory postings lists*, **linked-list** dapat digunakan.

Ini harus ada di disk buat yang isi dalemnya (selain head dan tail)

A collection of documents --> **Inverted Index**

Efisien untuk proses **insertion**; cocok ketika **updates** sering dilakukan (e.g. tambah dokumen baru)



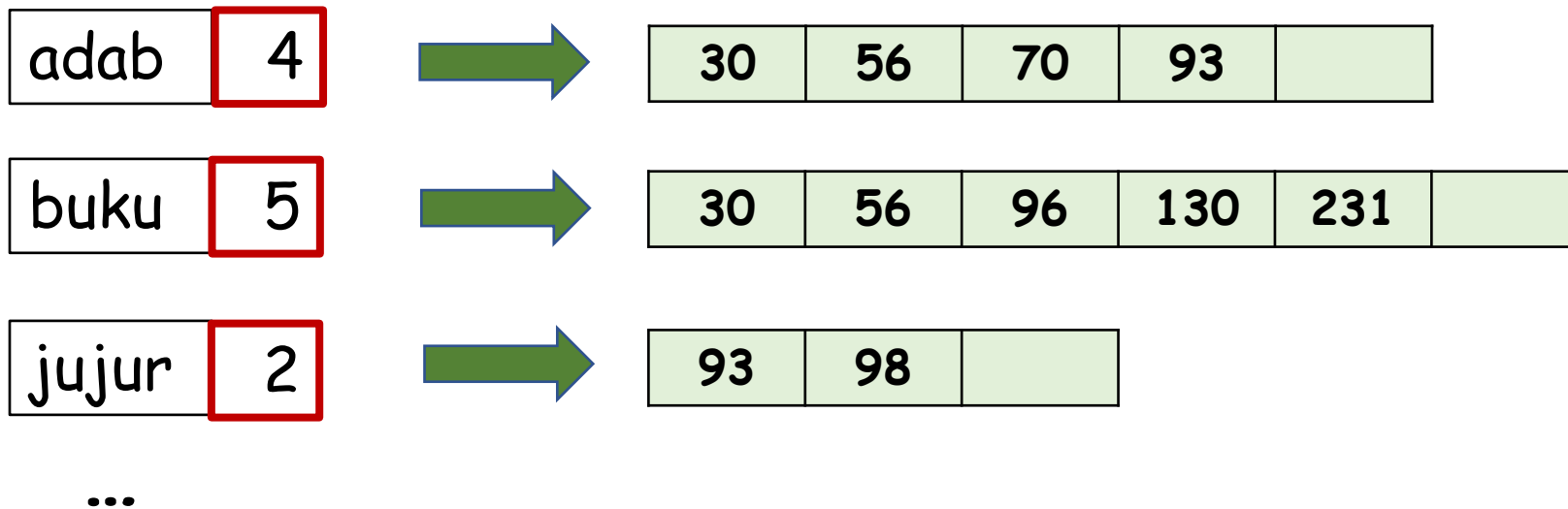
Variable-length Postings List

Untuk *in-memory postings lists*, **variable-length array** juga dapat digunakan.

lebih cepat untuk cari, cuman lebih bagus kalo jarang di update

Efisien dalam hal penggunaan **space** dan lebih cepat ketika **traversal**; cocok ketika **updates jarang** dilakukan.

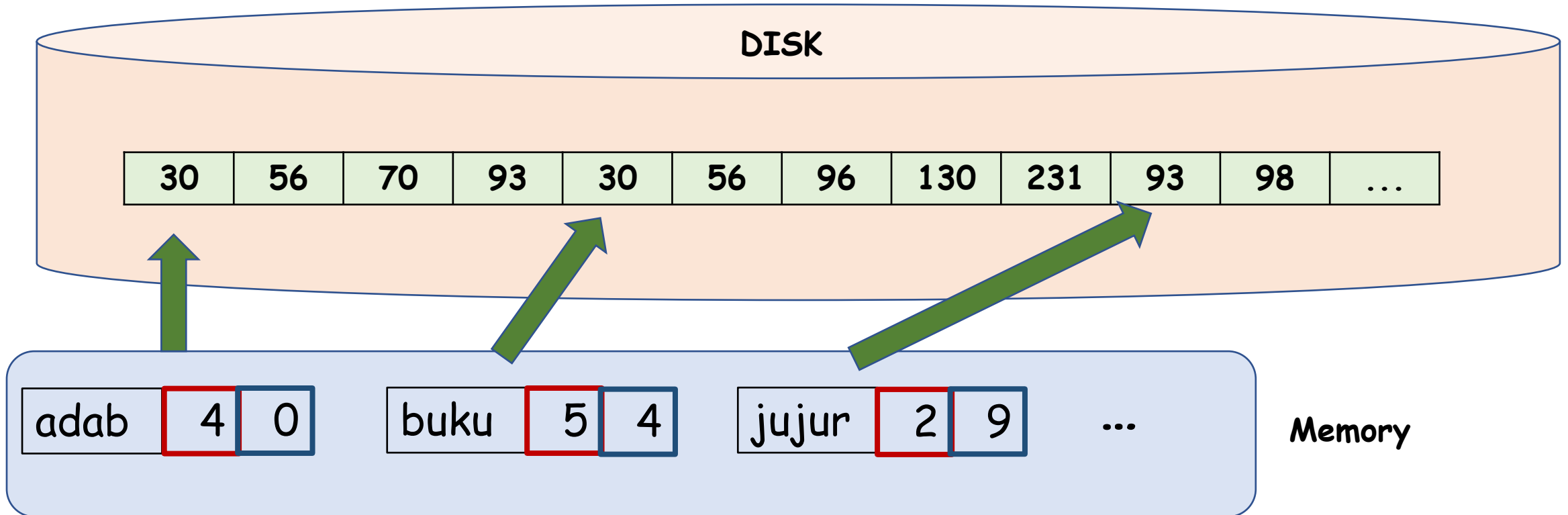
A collection of documents --> **Inverted Index**



Variable-length Postings List

When postings lists are kept on **disk**, they are stored in a contiguous positions.

untuk kata adab, postingnya dari kata berapa di disk ? 0, kalau buku mulai dari index berapa di disk (4).



Variable-length Postings List

When
containing

Selain informasi banyaknya posting,
perlu juga disimpan informasi lokasi
pada storage dimana posting pertama
berada (biasanya dalam satuan **byte**).

Di contoh ini, posisi dinyatakan dalam
indeks yang dimulai dari 0

sk, they are stored in a

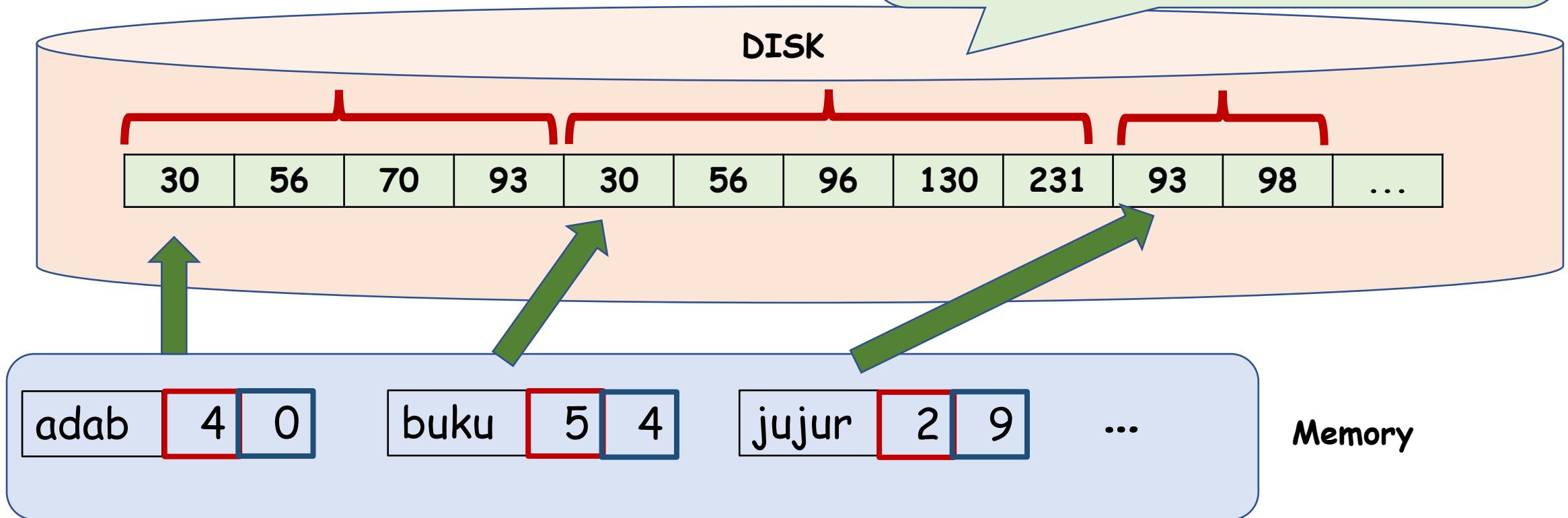


Variable-length Postings List

perlu kompres untuk mengecilkan ukuran dari list posting

When postings lists are kept on disk, they are stored in contiguous positions.

Postings lists are also **compressed**. Compression can minimize the size of postings lists and the number of disk seeks.



(Boolean) Query Processing with an Inverted Index

Boolean Retrieval Model

Retrieval Model ada 2 model: Boolean dan Ranked Model

Tidak peduli kalau misalnya lebih relevan.

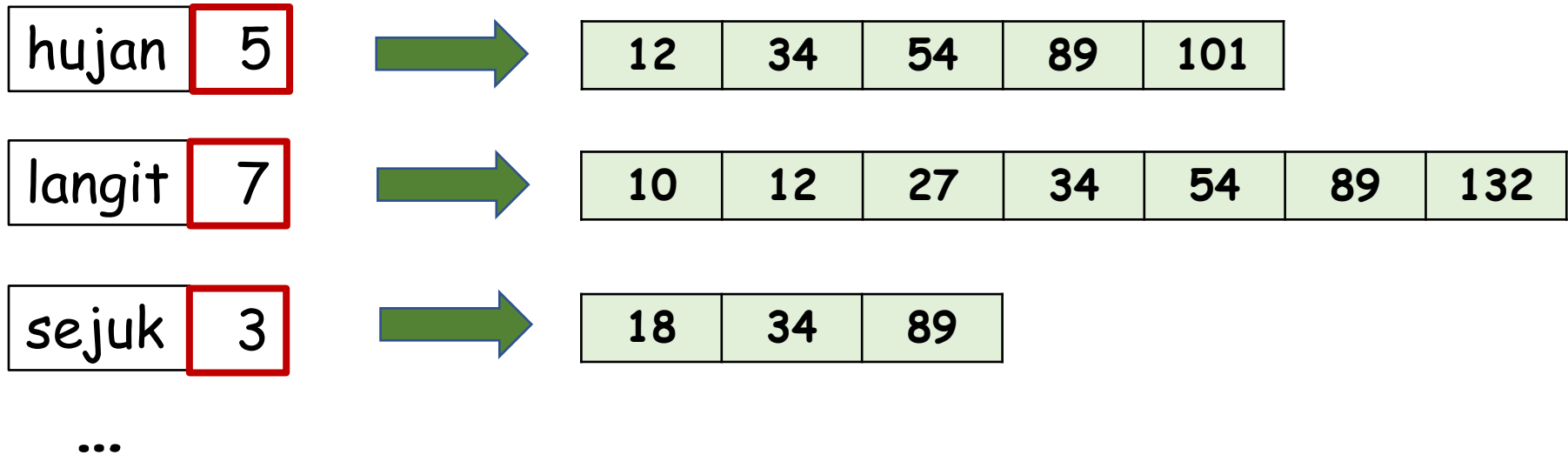
- Query is a Boolean expression using **AND**, **OR**, and **NOT**
- A document is assumed to be **a set of words**.
- The result is **an unordered set of documents**, **NOT** a ranked-list of documents.
- The result is also precise. A document matches condition or not.
- It has a long history (> 30 years). Many search engines are still Boolean:
 - Email & Library Catalog
 - You can still Boolean-search on Google right now!

AND Query

Saya ingin mencari buku yang mengandung kata **sejuk** dan **hujan**.

QUERY: **sejuk** **AND** **hujan**

pergi ke hard disk, cek bytenya dimana, dibaca dulu, baru kita decompressed ke memory. Lalu setelah dari memory baru di process.

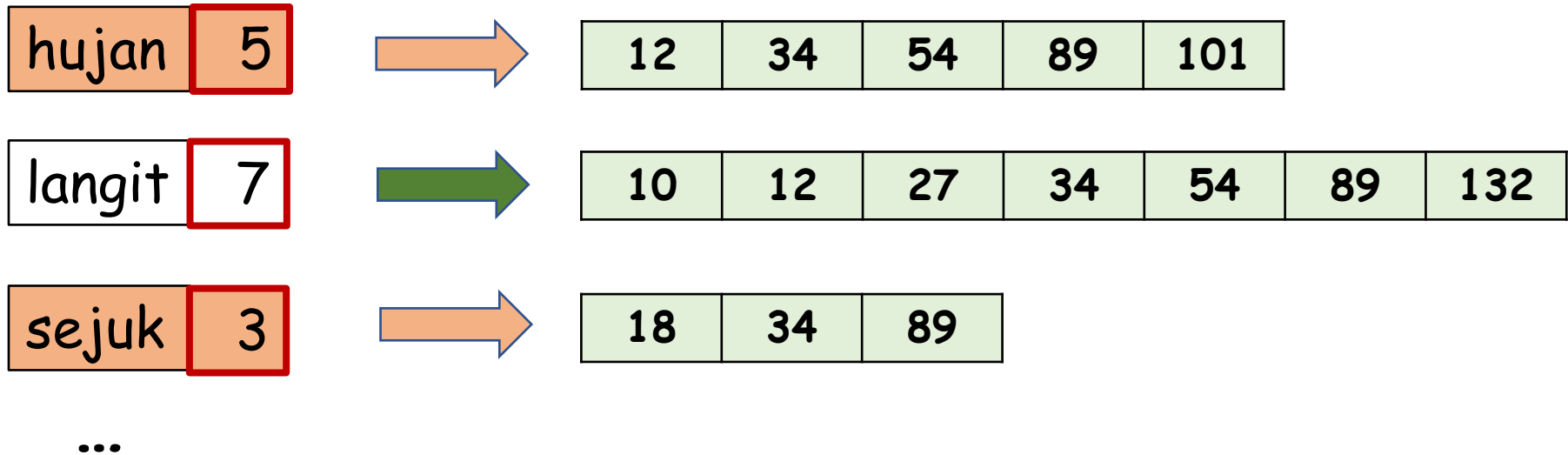


AND Query

Saya ingin mencari buku yang mengandung kata **sejuk** dan **hujan**.

QUERY: **sejuk** **AND** **hujan**

#1 Cari posisi term sejuk dan hujan di Dictionary, dan kemudian ambil Postings listnya

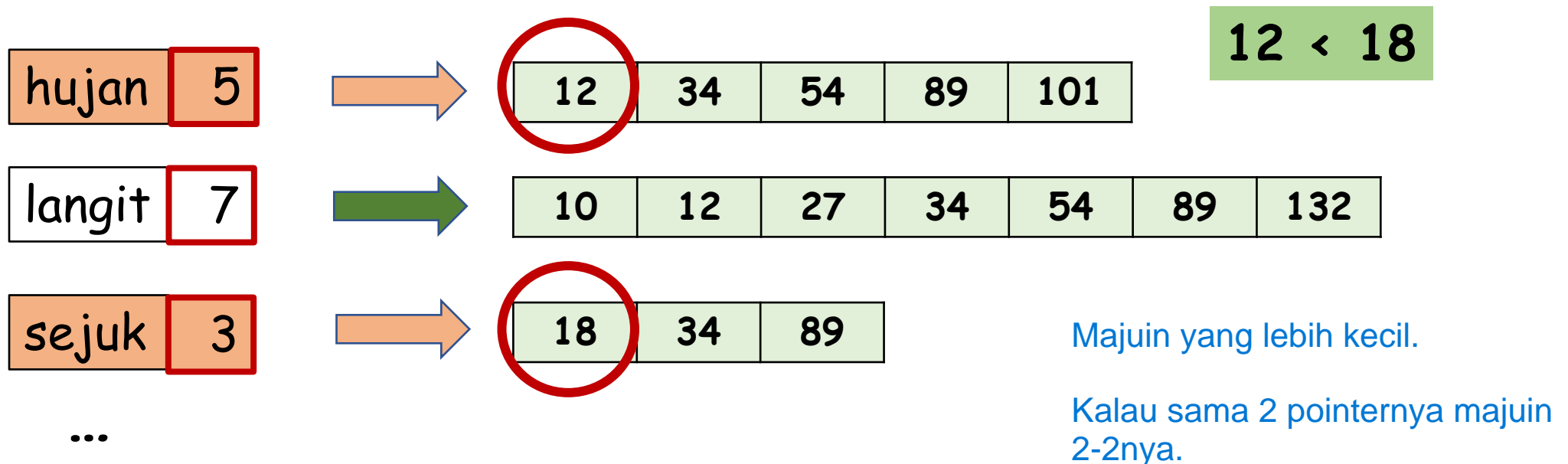


AND Query

Saya ingin mencari buku yang mengandung kata **sejuk** dan **hujan**.

QUERY: **sejuk** **AND** **hujan**

#2 merge the two postings lists (intersection)



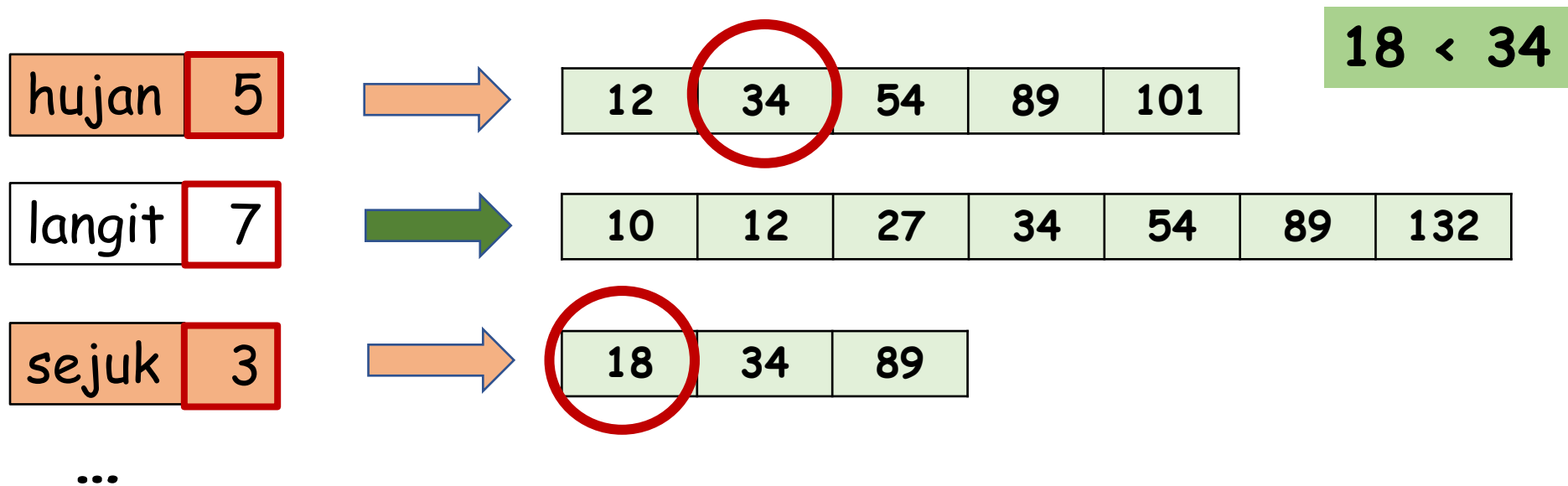
Answer = []

AND Query

Saya ingin mencari buku yang mengandung kata **sejuk** dan **hujan**.

QUERY: **sejuk** **AND** **hujan**

#2 merge the two postings lists (intersection)



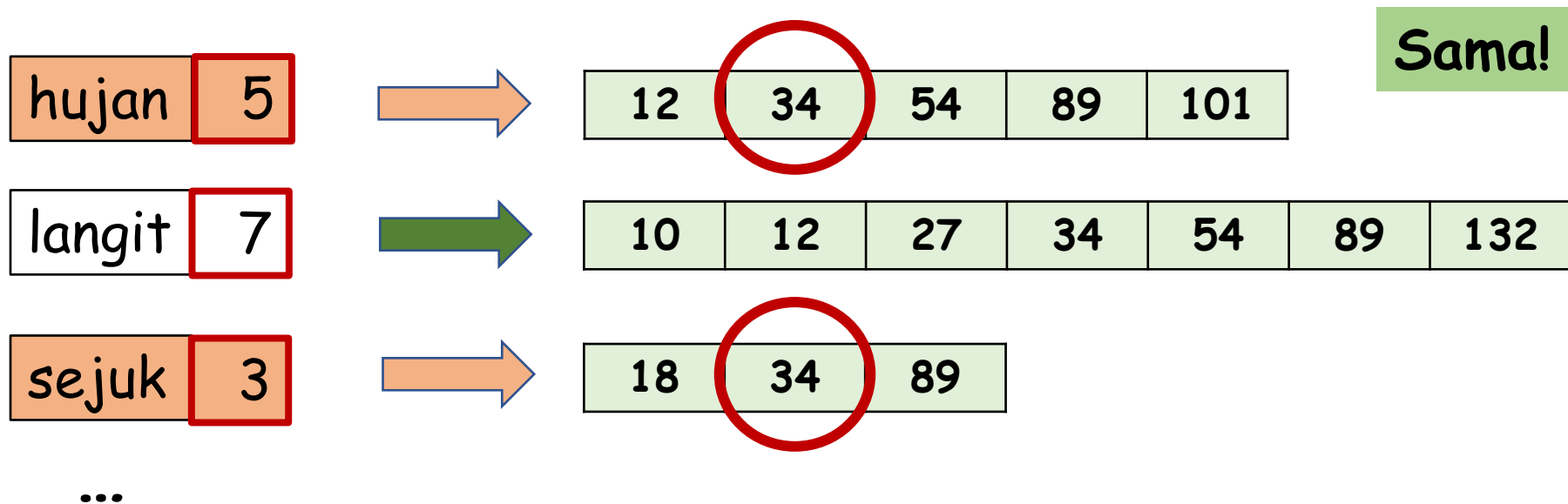
Answer = []

AND Query

Saya ingin mencari buku yang mengandung kata **sejuk** dan **hujan**.

QUERY: **sejuk** **AND** **hujan**

#2 merge the two postings lists (intersection)



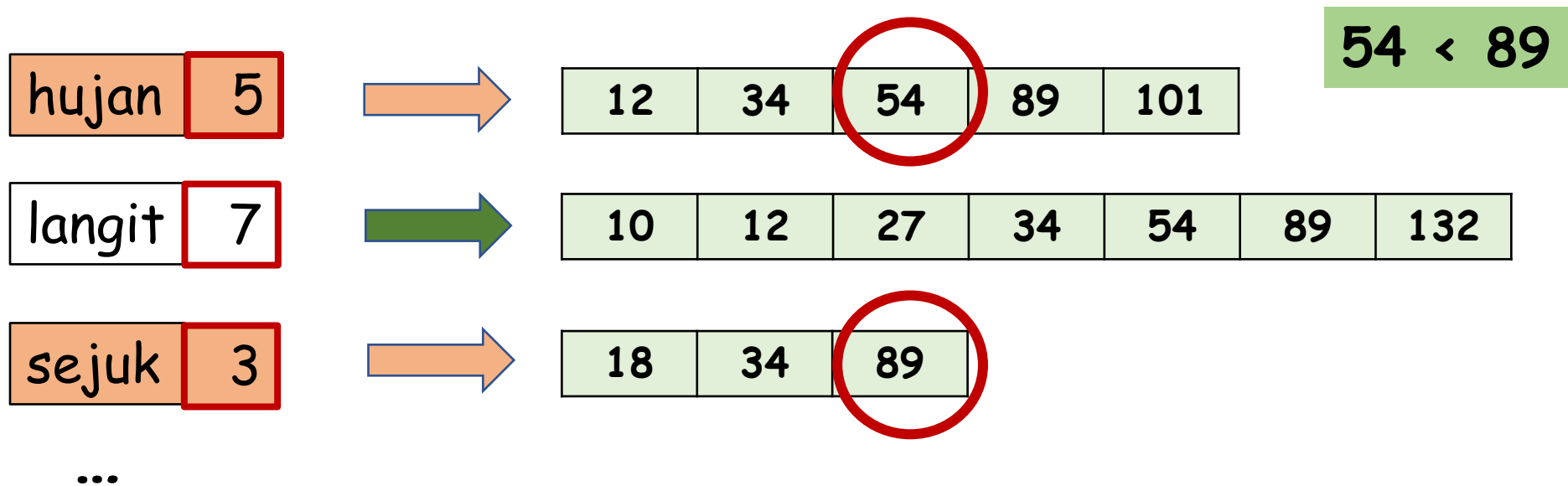
Answer = [34]

AND Query

Saya ingin mencari buku yang mengandung kata **sejuk** dan **hujan**.

QUERY: **sejuk** **AND** **hujan**

#2 merge the two postings lists (intersection)



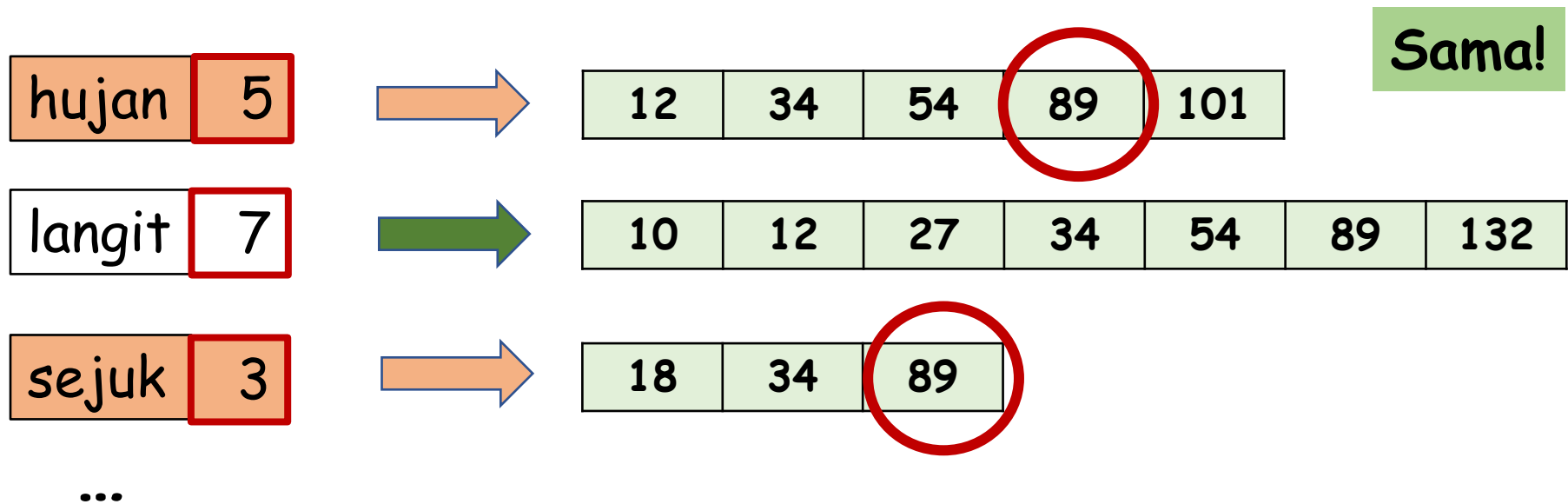
Answer = [34]

AND Query

Saya ingin mencari buku yang mengandung kata **sejuk** dan **hujan**.

QUERY: **sejuk** **AND** **hujan**

#2 merge the two postings lists (intersection)



Answer = [34, 89]

AND Query

Kompleksitas: $O(M + N)$

M: Panjang postings list pertama

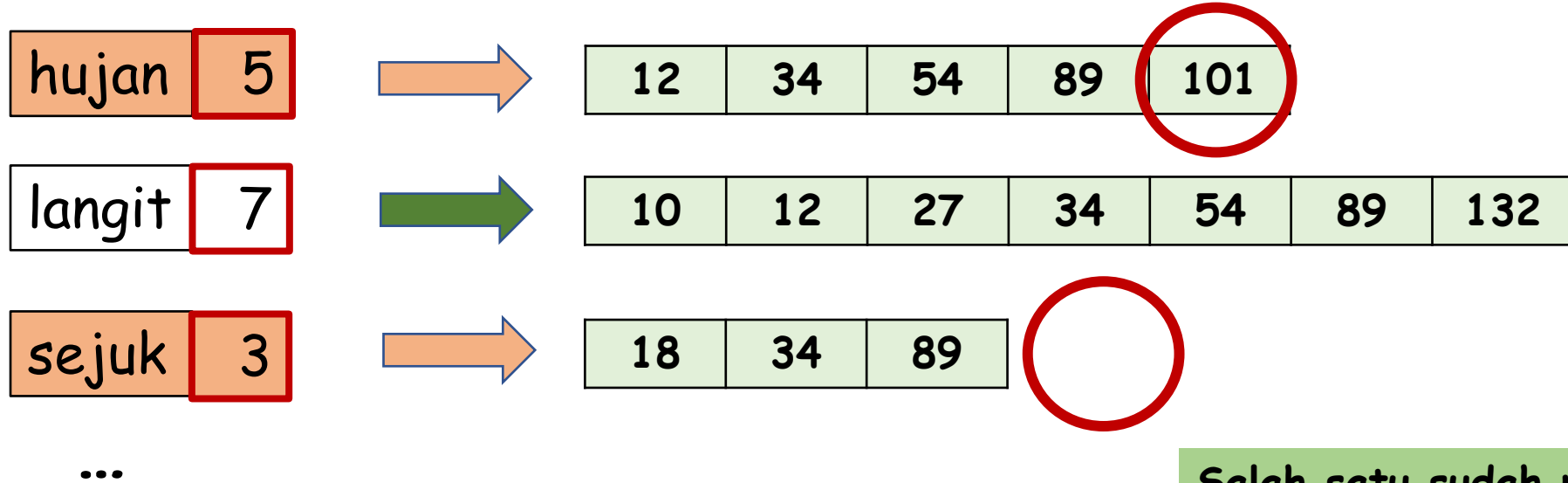
N: Panjang postings list kedua

Saya ingin mencari buku yang mengandung kata **sejuk** dan **hujan**.

QUERY: **sejuk** **AND** **hujan**

#2 merge the two postings lists (intersection)

Terurut karena kalau gak bakal $O(M*N)$. Looping 2 kali if thats the case.



Answer = [34, 89]

Intersecting two postings lists

Kalau pakai set gak terurut

```
Intersect(p1, p2):  
    answer ← []  
    while p1 != null and p2 != null:  
        if docID(p1) == docID(p2):  
            add(answer, docID(p1))  
            p1 ← next(p1)  
            p2 ← next(p2)  
        elif docID(p1) < docID(p2):  
            p1 ← next(p1)  tambahkan add(answer)  
        else:  
            p2 ← next(p2)  tambahkan add(answer)  
    return answer
```

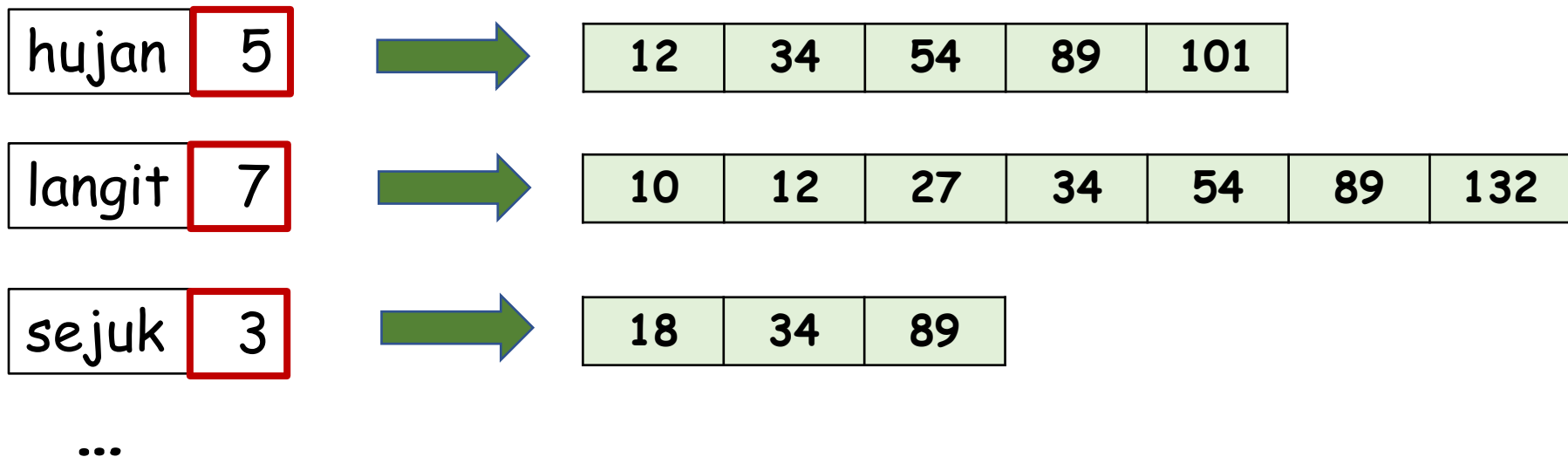
tambahkan while p1 != null, add answer
tambahkan while p2 != null, add answer

Saatnya Anda Berpikir :)

Bagaimana dengan query: sejuk **OR** hujan ?

Bagaimana dengan query: sejuk **AND NOT** langit ?

Can we still run the “merge” process in $O(M + N)$? Write your codes!



Query Optimization - Conjunctive Queries

Bagaimana dengan query: sejuk **AND** hujan **AND** langit

maximum

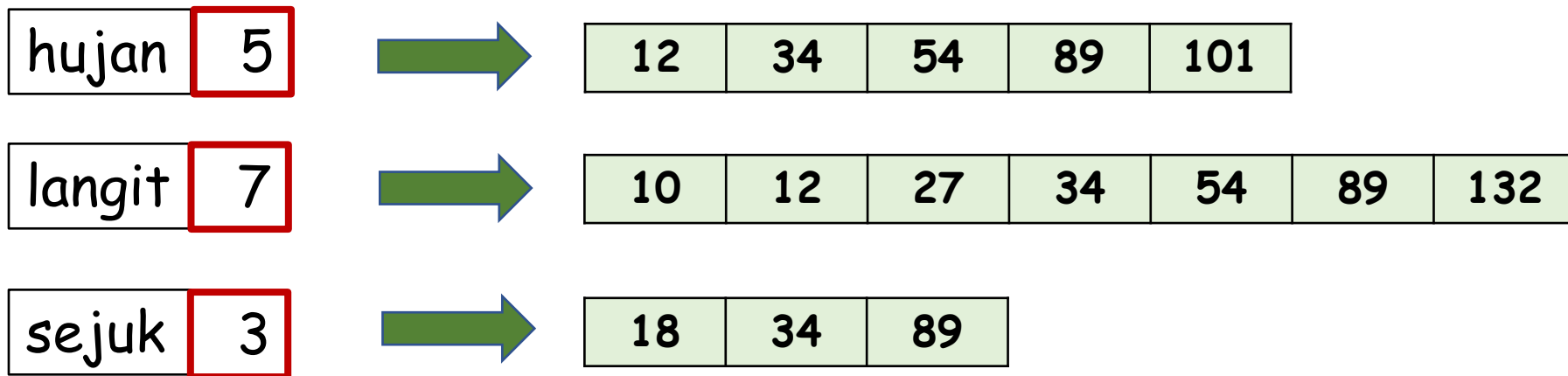
1) $(5+7) + (5+3) = 20$ step maksimum perbandingan

Masih bisakah kita selesaikan dalam waktu "linier"?

Can we improve its efficiency?

maximum hujan dan sejuk:

$(5+3) + (3 \rightarrow \text{maksimal dari (hujan + sejuk) + 7}) = 18$ perbandingan



...

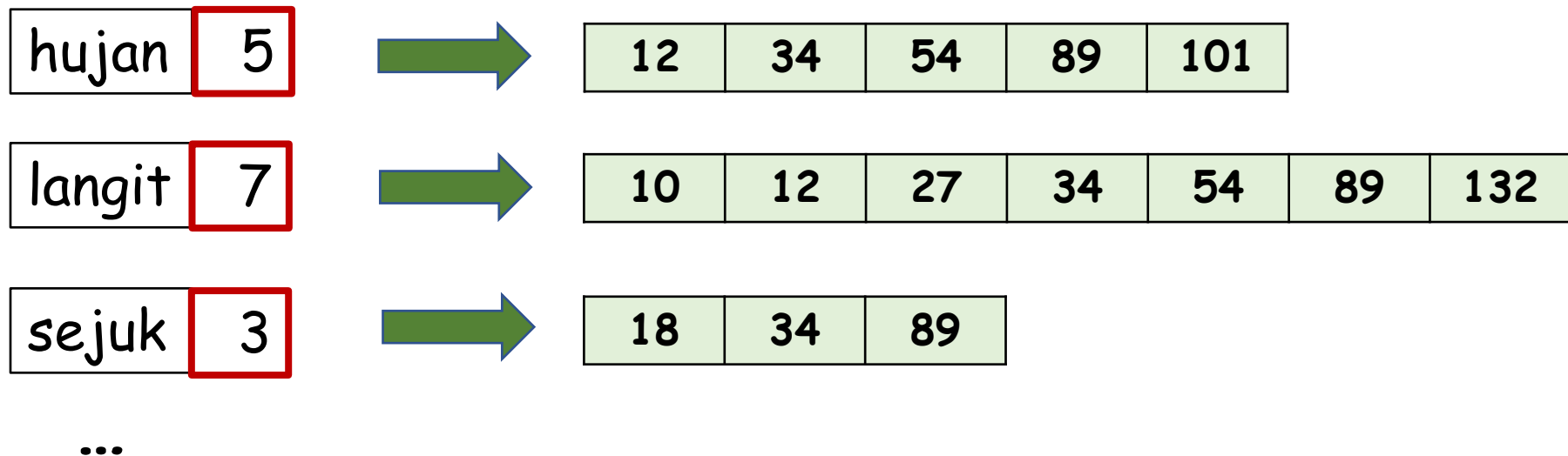
Query Optimization - Conjunctive Queries

Bagaimana dengan query: sejuk **AND** hujan **AND** langit

Mana yang kita eksekusi dahulu? Mana yang lebih efisien?

1. (hujan **AND** langit) **AND** sejuk
2. (hujan **AND** sejuk) **AND** langit

di sort by increasing frequency, jadijnya lebih baik hujan dan sejuk karena hanya 18 perbandingan



Query Optimization

Bagaimana dengan query: sejuk **AND** hujan

Mana yang kita eksekusi dahulu? Mana yang lebih cepat?

1. (hujan **AND** langit) **AND** sejuk
2. (hujan **AND** sejuk) **AND** langit

Jika kita mulai iriskan dua *postings lists* terpendek, maka panjang *intermediate postings*-nya tidak akan melebihi panjang *postings list* terpendek.

Kerjaan untuk memproses query secara keseluruhan menjadi lebih efisien.

hitung ini dulu. fix.

hujan 5



12	34	54	89	101
----	----	----	----	-----

langit 7



10	12	27	34	54	89	132
----	----	----	----	----	----	-----

sejuk 3



18	34	89
----	----	----

...

Dan ini salah satu alasan mengapa informasi *document frequency* perlu disimpan di Dictionary.

Query Optimization - Conjunctive Queries

```
Intersect([p1, ..., pn]):
```

```
    terms    ← SortByIncreasingFrequency([p1, ..., pn])
```

```
    results  ← postings(first(terms))
```

```
    terms    ← rest(terms)
```

```
    while terms != null and results != null:
```

```
        results ← Intersect(results, postings(first(terms)))
```

```
        terms   ← rest(terms)
```

```
    return results
```


Boolean Retrieval Model?

- Populer antara periode 1960 – 1990, khususnya untuk aplikasi search pada perpustakaan dan juga pada domain hukum (pencarian dokumen legal & patent).

jawaban nya himpunan.

- Query yang berbentuk **Boolean expression** kurang ekspresif untuk memenuhi kebutuhan informasi pengguna secara umum.
 - You pose “free text queries” when using Google, don't you?
- Boolean retrieval model mengembalikan **unordered set of answers!**
 - Anda suka dengan search engine yang seperti ini? 😊

Ranked Retrieval Model

- "I want you to tell me what the best answer is"
- Setiap dokumen perlu ada skor yang menandakan "seberapa relevan" dokumen tersebut dengan kebutuhan informasi kita.

Not recommended because if boolean 10 document if its only unordered, and might not be as important as the other. We would need to read all 10 documents



apakah benar anak UI kaya dan individualis



[All](#)

[News](#)

[Images](#)

[Videos](#)

[Shopping](#)

[More](#)

[Tools](#)

About 63,600 results (0.47 seconds)

<https://id.quora.com/Apakah-anak-UI-kebanyakan-indiv...>

[Apakah anak UI kebanyakan individualis dan ketika bergaul ...](#)

Tidak semua **anak UI** itu cantik/ganteng dan **kaya** raya. Ada **anak UI** yang tidak gud luling dan bukan termasuk dalam golongan ekonomi mapan. Kalau ada **anak UI** yang ...

6 answers · 11 votes: Jawabannya. Ya Eitss,, tetapi ternyata nggak cuma mahasiswa UI saja !...

<https://id.quora.com/Apakah-benar-mahasiswa-Universit...>

[Apakah benar mahasiswa Universitas Indonesia hedon dan ...](#)

Ada benarnya tapi tidak dapat digeneralisir semua **anak UI** tuh hedon dan **individualis**. Saya termasuk **anak UI** yang "hedon" tapi tidak **individualis** banget.

<https://www.anakui.com/katanya-a...> · [Translate this page](#)

[Katanya Anak UI Kaya-kaya, Pintar dari Lahir, Kuliahnya Enak ...](#)

Jun 13, 2015 — Katanya **Anak UI Kaya-kaya**, Pintar dari Lahir, Kuliahnya Enak, Abis Lulus Pasti Sukses. Bener Nggak Sih? Mungkin tak sedikit yang mengira ...

Missing: **individualis** | Must include: **individualis**

<https://rencanamu.id/post/karakte...> · [Translate this page](#)

[Karakter Mahasiswa ITB Versus UI. Mana yang Paling Cocok ...](#)

Apr 15, 2017 — Karakter Mahasiswa ITB Versus **UI**. Mana yang Paling Cocok Sama Kamu? · 1. "Beragam! Ada yang kompetitif, ambisius, ada yang santai tapi hasilnya ...

Missing: **benar** | Must include: **benar**

Ranking Model

Kita berharap agar yang ada di ranking pertama ini adalah yang paling menjawab pertanyaan kita

- "I want you to tell me what the best answer is"
- Setiap dokumen perlu ada skor yang menandakan "seberapa relevan" dokumen tersebut dengan kebutuhan informasi kita.

apakah benar anak UI kaya dan individualis

[All](#) [News](#) [Images](#) [Videos](#) [Shopping](#) [More](#) [Tools](#)

About 63,600 results (0.47 seconds)

<https://id.quora.com> > Apakah-anak-UI-kebanyakan-indiv... [More](#)

Apakah anak UI kebanyakan individualis dan ketika bergaul ...

Tidak semua **anak UI** itu cantik/ganteng dan **kaya** raya. Ada **anak UI** yang tidak gud luling dan bukan termasuk dalam golongan ekonomi mapan. Kalau ada **anak UI** yang ...

6 answers · 11 votes: Jawabannya. Ya Eitss,, tetapi ternyata nggak cuma mahasiswa UI saja !...

Rank 1

<https://id.quora.com> > Apakah-benar-mahasiswa-Universit... [More](#)

Apakah benar mahasiswa Universitas Indonesia hedon dan ...

Ada benarnya tapi tidak dapat digeneralisir semua **anak UI** tuh hedon dan **individualis**. Saya termasuk **anak UI** yang "hedon" tapi tidak **individualis** banget.

Rank 2

<https://www.anakui.com> > katanya-a... · [Translate this page](#) [More](#)

Katanya Anak UI Kaya-kaya, Pintar dari Lahir, Kuliahnya Enak ...

Jun 13, 2015 — Katanya **Anak UI Kaya-kaya**, Pintar dari Lahir, Kuliahnya Enak, Abis Lulus Pasti Sukses. Bener Nggak Sih? Mungkin tak sedikit yang mengira ...

Missing: **individualis** | Must include: **individualis**

Rank 3

<https://rencanamu.id> > post > karakte... · [Translate this page](#) [More](#)

Karakter Mahasiswa ITB Versus UI. Mana yang Paling Cocok ...

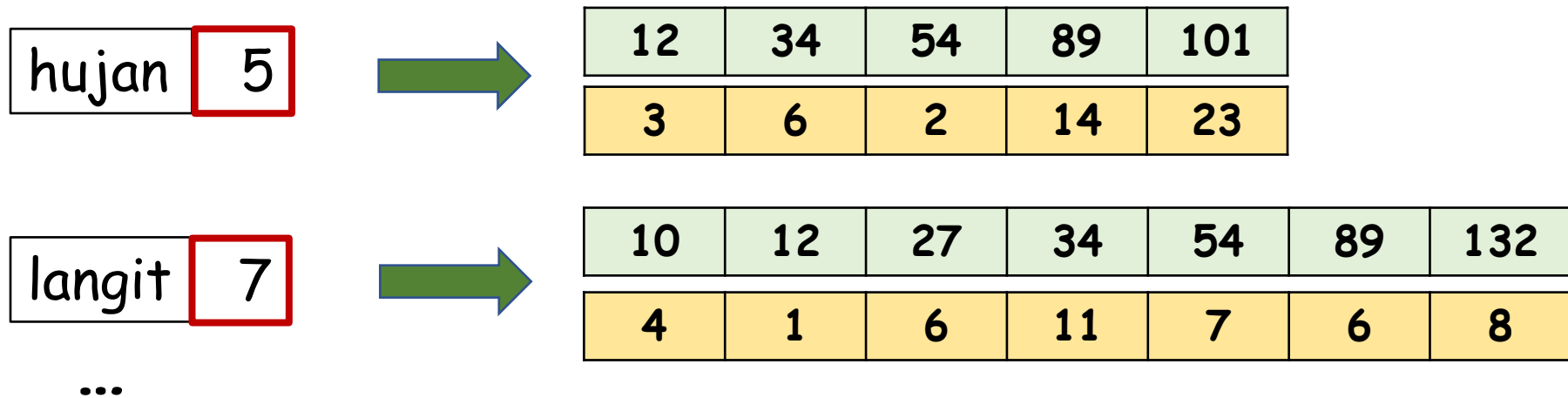
Apr 15, 2017 — Karakter Mahasiswa ITB Versus **UI**. Mana yang Paling Cocok Sama Kamu? · 1. "Beragam! Ada yang kompetitif, ambisius, ada yang santai tapi hasilnya ...

Missing: **benar** | Must include: **benar**

Rank 4

Ranked Retrieval Model

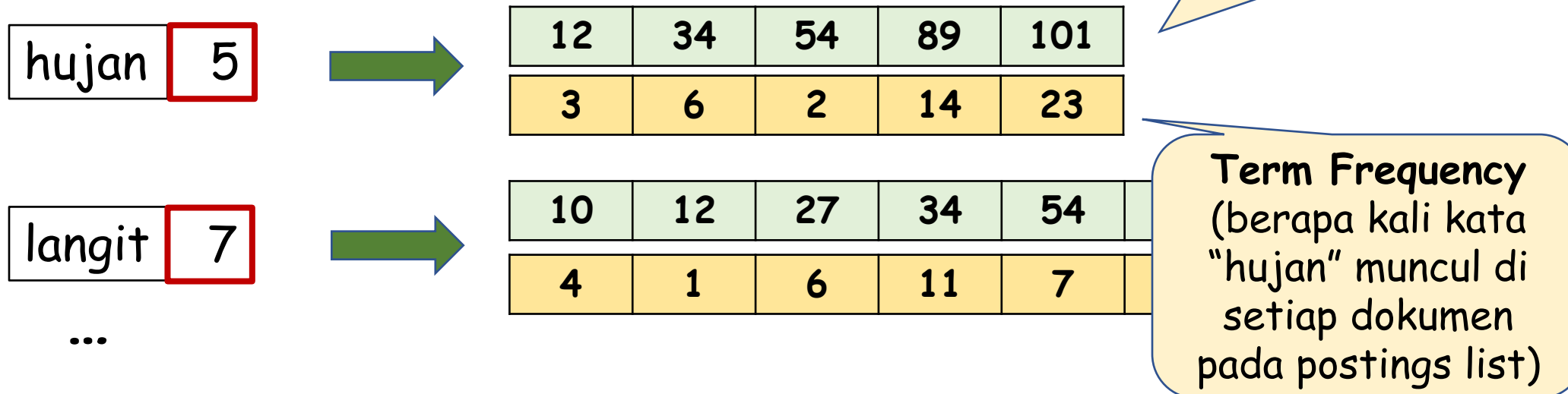
- Apakah Ranked Retrieval Model juga menggunakan **Inverted Index**?
Ya, dengan beberapa modifikasi.
- Sebagai contoh, beberapa ranked retrieval model seperti **vector space model** membutuhkan inverted index yang seperti:



* Anda akan belajar tentang ini di pertemuan yang akan datang

Ranked Retrieval Model

- Apakah Ranked Retrieval Model juga menggunakan **Inverted Index**?
Ya, dengan beberapa modifikasi.
- Sebagai contoh, beberapa ranked retrieval model seperti **model** membutuhkan inverted index yang seperti:



* Anda akan belajar tentang ini di pertemuan yang akan datang

Phrase Queries & Proximity Search

Apa yang harus diubah agar bisa handle *phrase queries*?

Di buku teks, dikatakan bahwa sekitar 10% queries pada Web search adalah Phrase Queries.

Query: "Universitas Indonesia"

Doc #23

Presiden Jokowi mengunjungi kampus **Universitas Indonesia** untuk menghadiri diskusi di bidang ekonomi dengan ...

Match 😊

Doc #24

Dalam hal ini DIKTI mengatakan bahwa **indonesia** mempunyai target tinggi untuk meningkatkan kualitas **universitas** di pulau ...

Do not match 😞

Doc #25

Universitas Pendidikan **Indonesia** bekerja sama dengan Dinas Pendidikan Bandung untuk menyelenggarakan ...

Do not match 😞

Doc #26

Indonesia membutuhkan **Universitas** berkualitas untuk meningkatkan taraf hidup ...

Do not match 😞

Bagaimana jika *k-word proximity queries*?

Cari semua dokumen dimana kata "Universitas" dan "Indonesia" muncul bersamaan dalam jarak 2

Query: Universitas /2 Indonesia

Doc #23

Presiden Jokowi mengunjungi kampus **Universitas Indonesia** untuk menghadiri diskusi di bidang ekonomi dengan ...

Match 😊

Doc #24

Dalam hal ini DIKTI mengatakan bahwa **indonesia** mempunyai target tinggi untuk meningkatkan kualitas **universitas** di pulau ...

Do not match 😞

Doc #25

Universitas Pendidikan **Indonesia** bekerja sama dengan Dinas Pendidikan Bandung untuk menyelenggarakan ...

Match 😊

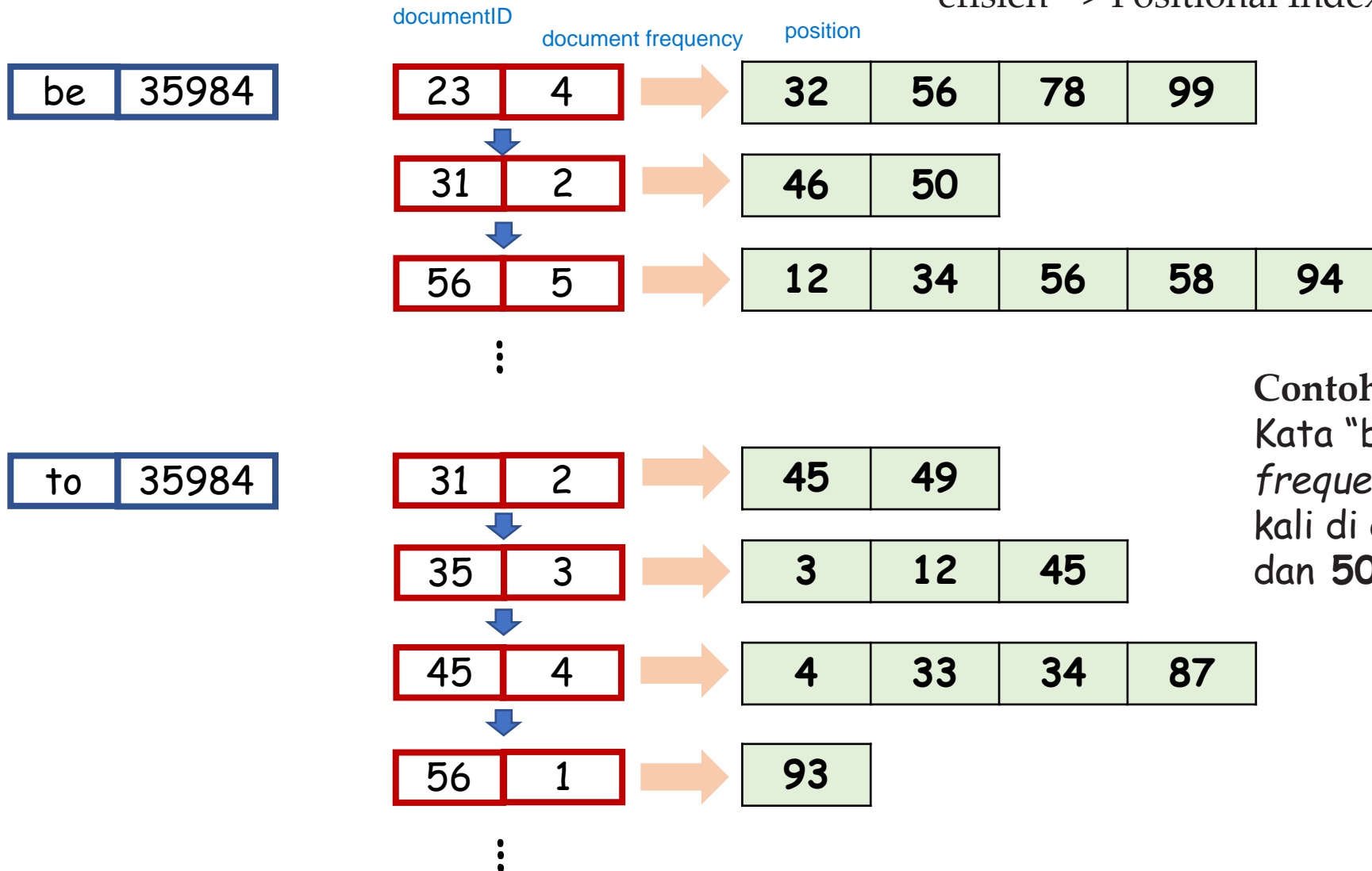
Doc #26

Indonesia membutuhkan **Universitas** berkualitas untuk meningkatkan taraf hidup ...

Match 😊

Positional Indexes

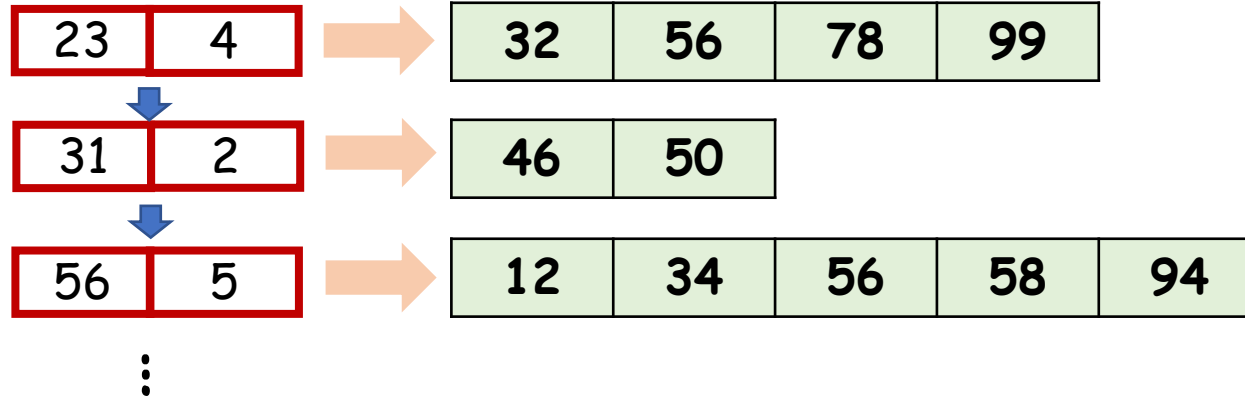
Inverted Index biasa sudah tidak bisa handle kasus Phrase Queries & K-Word Proximity Queries. Kita perlu struktur data lain, namun juga harus tetap efisien --> Positional Indexes



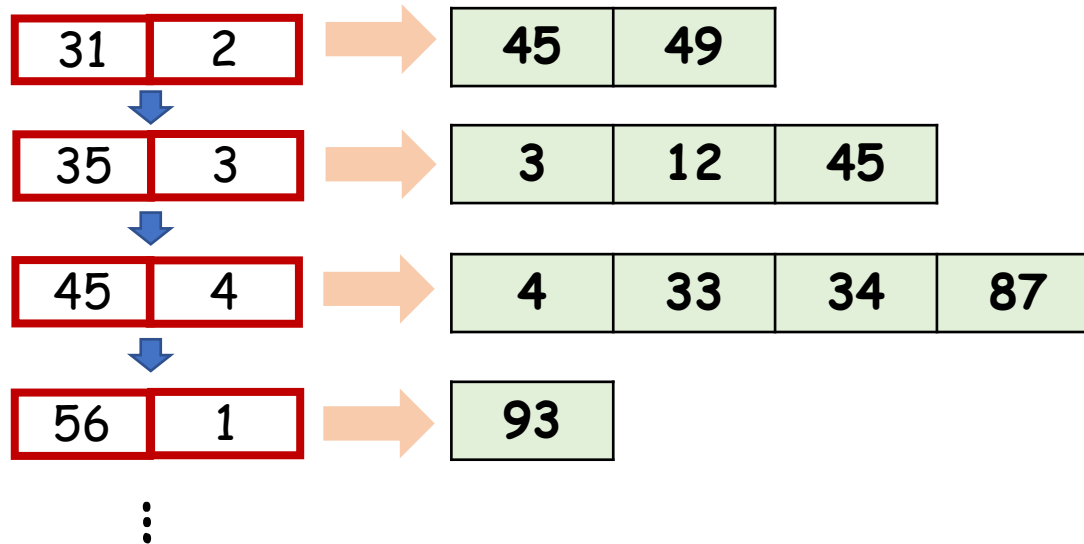
Contoh:
Kata "be" mempunyai *document frequency* **35984**, dan muncul 2 kali di **doc #31** pada posisi **46** dan **50**.

Query: "to be"

be 35984



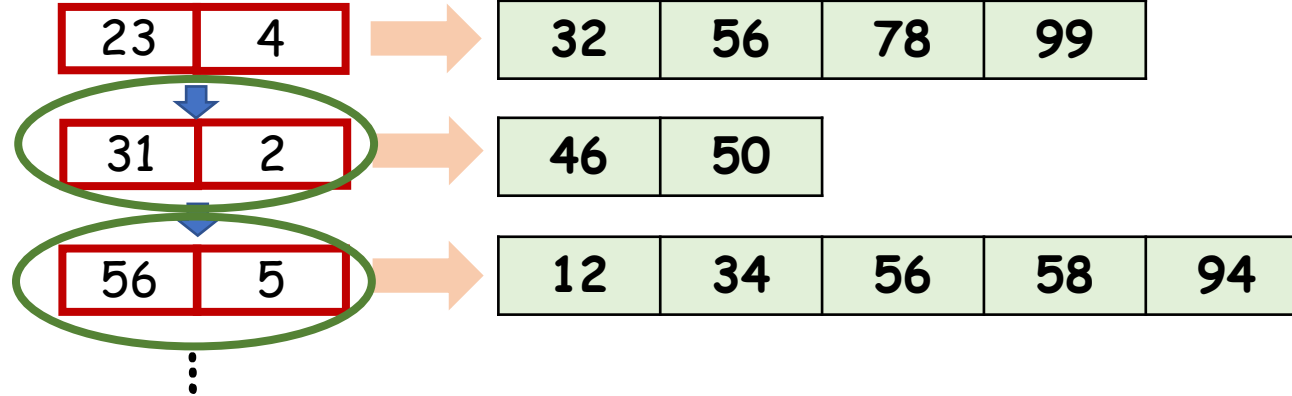
to 35984



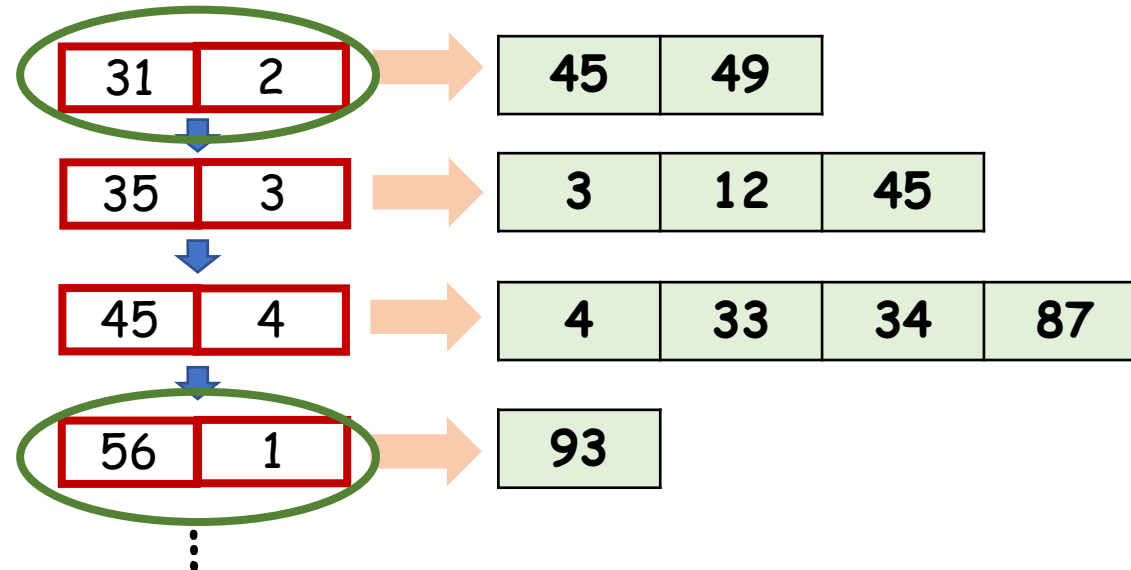
Query: "to be"

Seperti sebelumnya, kita lakukan *intersection* pada level dokumen.

be 35984



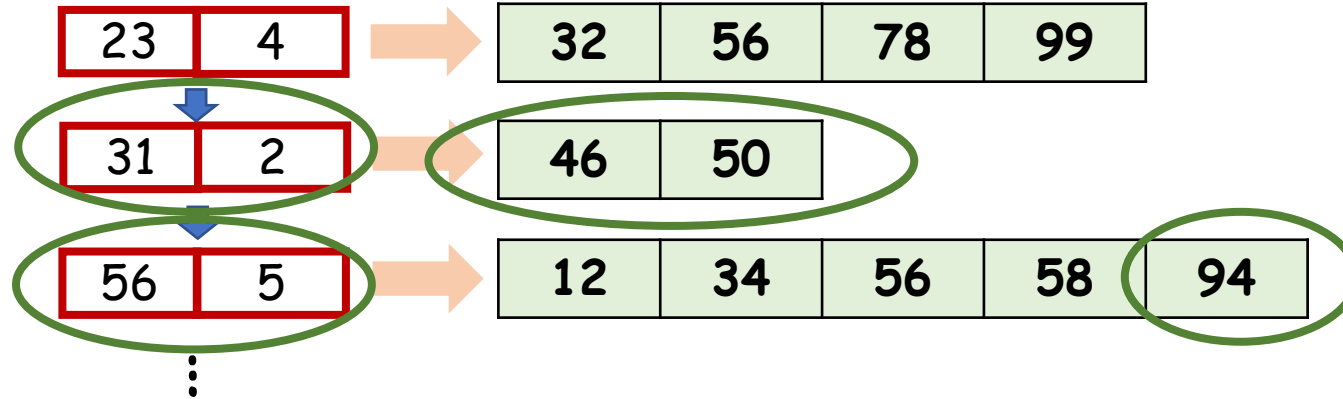
to 35984



Query: "to be"

Kemudian kita periksa posisi kata-kata pada kedua dokumen dan memastikan sesuai dengan query.

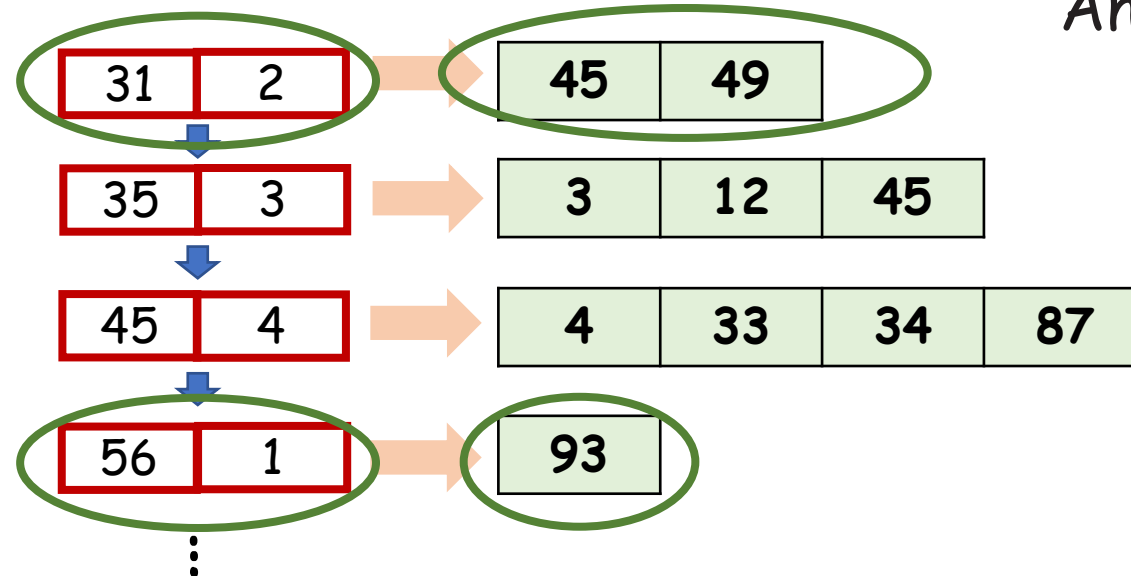
be 35984



Posisi
to

Posisi
be

to 35984



Answer: [<doc 31, 45, 46>,
<doc 31, 49, 50>,
<doc 56, 93, 94>]

Intersection of two postings lists


```
Intersect(p1, p2):  
    answer ← []  
    while p1 != null and p2 != null:  
        if docID(p1) == docID(p2):  
            add(answer, docID(p1))  
            p1 ← next(p1)  
            p2 ← next(p2)  
        elif docID(p1) < docID(p2):  
            p1 ← next(p1)  
        else:  
            p2 ← next(p2)  
    return answer
```

Mengingatikan kembali, algoritme ini melakukan "merge" 2 postings lists untuk kasus Inverted Index biasa.

Apa yang harus dimodifikasi untuk kasus Positional Indexes?

Proximity Intersection of two postings lists

Find places where the two terms appear within k words of each other!

```
PositionalIntersect(p1, p2, k):  
    answer ← []  
    while p1 != null and p2 != null:  
        if docID(p1) == docID(p2):  
              
            p1 ← next(p1)  
            p2 ← next(p2)  
        elif docID(p1) < docID(p2):  
            p1 ← next(p1)  
        else:  
            p2 ← next(p2)  
    return answer
```

```
l = []  
pp1 = positions(p1)  
pp2 = positions(p2)  
while pp1 != null:  
    while pp2 != null:  
        if |pos(pp1) - pos(pp2)| <= k:  
            add(l, pos(pp2))  
            elif pos(pp2) > pos(pp1):  
                break  
            pp2 ← next(pp2)  
  
    while l != [] and |l[0] - pos(pp1)| > k:  
        delete(l[0])  
  
    for ps in l:  
        add(answer, <docID(p1), pos(pp1), ps>)  
  
    pp1 ← next(pp1)
```