INFORMATION RETRIEVAL

Week 5 – Index Compression

28.03.2025

Severin Mills

Today

1

Exercise Recap

- Discussion
- Questions

2

Theory

- BSBI
- SPIMI
- Index updating
- Logarithmic merging

3

Kahoot / Exam questions

- Exercise 4: Index Construction

Online Resource

B+-trees visualization

Visualization of Data Structure by Prof. Dr. Galles
www.cs.usfca.edu/~galles/visualization/BPlusTree.html
Definition might differ from what we defined in the lecture though!

Bonus Exercise

Tolerant Retrieval

Which of the following is a correct statement?

- a. Lookup in a hash index is constant. Lookup in a B-tree index is linear.
- b. Lookup in a hash index is linear. Lookup in a B-tree index is logarithmic.
- c. Lookup in a hash index is constant. Lookup in a B-tree index is constant.
- d. Lookup in a hash index is constant. Lookup in a B-tree index is logarithmic.
- e. Lookup in a hash index is logarithmic. Lookup in a B-tree index is linear.

Tolerant Retrieval

Which of the following is a correct statement?

- ○ a. Lookup in a hash index is constant. Lookup in a B-tree index is linear.
- ○ b. Lookup in a hash index is linear. Lookup in a B-tree index is logarithmic.
- ○ c. Lookup in a hash index is constant. Lookup in a B-tree index is constant.
- ◉ d. Lookup in a hash index is constant. Lookup in a B-tree index is logarithmic.
- ○ e. Lookup in a hash index is logarithmic. Lookup in a B-tree index is linear.

Bonus Exercise

Tolerant Retrieval

Mark each one of the following statemnts as True or False

True	False	
○	○	A B+-Tree is always balanced.
○	○	B+-Trees don't support range search.
○	○	The linked list structure of leaf nodes in a B+-Tree allows range queries to run in O(N) time complexity.
○	○	B+-Trees are ideal for all types of key-value storage, including workloads with frequent insertions and deletions.

Tolerant Retrieval

Mark each one of the following statemnts as True or False

True	False	
●	○	A B+-Tree is always balanced.
○	●	B+-Trees don't support range search.
●	○	The linked list structure of leaf nodes in a B+-Tree allows range queries to run in O(N) time complexity.
○	●	B+-Trees are ideal for all types of key-value storage, including workloads with frequent insertions and deletions.

Bonus Exercise

Tolerant Retrieval

Edit (Levenshtein) Distance (Part 1)

Complete the following table on a notepad to compute the edit distance between the words s_1="cake" and s_2="lame" (refer to the algorithm in Figure 3.5, page 59 of the book):

		c	**a**	**k**	**e**
	0	1	2	3	4
l	1				
a	2				
m	3				
e	4				

1. What are the four missing numbers in the last ROW?

Column c:

Column a:

Column k:

Column e:

28.03.2025

Bonus Exercise

Tolerant Retrieval

Edit (Levenshtein) Distance (Part 1)

Complete the following table on a notepad to compute the edit distance between the words s_1="cake" and s_2="lame" (refer to the algorithm in Figure 3.5, page 59 of the book):

		c	a	k	e
	0	1	2	3	4
l	1				
a	2				
m	3				
e	4				

1. What are the four missing numbers in the last ROW?

Column c: 4

Column a: 3

Column k: 3

Column e: 2

Bonus Exercise

Tolerant Retrieval

Edit (Levenshtein) Distance (Part 2)

We use subscript i for the rows and j for the columns in the table.

This question is about the table *m* used to calculate the edit (levenshtein) distance between words s_1 and s_2. Recall that cell m[i,j] (i.e. the edit distance between the first i characters of the first word (s_1) and the first j characters of the second (s_2)) is the minimum of m[i-1,j]+1, m[i,j-1]+1 and either m[i-1,j-1] (if s_1[i+1] = s_2[j+1]) or m[i-1,j-1]+1 otherwise.

What does each of these terms represent, in terms of character edits (insert, delete, replace)? Choose the option with the correct values to replace the one or multiple underlines ___. (the order matters, the first value should replace the first ___, the second value the second ___, etc.)

a) m[i-1,j]+1 is the cost of deleting s_1[___] from s_1[0 ... i], then editing s_1[0 ... i-1] into s_2[0 ... j].

- ○ [i]
- ○ [i-1]
- ○ [j-1]
- ○ [j]

b) m[i,j-1]+1 is the cost of editing s_1[___] into s_2[___], then inserting s_2[___].

- ○ [0 ... i], [0 ... j-1], [j]
- ○ [0 ... i], [0 ... j-1], [j-1]
- ○ [0 ... j-1], [0 ... i-1], [i]

c) m[i-1,j-1] is the cost of editing s_1[___] into s_2[___]. In addition, +1 is added to the cost for replacing s_1[___] with s_2[___], but only if they are not already the same.

- ○ [0 ... i-1], [0 ... j-1], [j], [i]
- ○ [0 ... i-1], [0 ... j-1], [i], [j]
- ○ [0 ... j-1], [0 ... i-1], [j], [i]

Bonus Exercise

Tolerant Retrieval

Edit (Levenshtein) Distance (Part 2)

We use subscript i for the rows and j for the columns in the table.

This question is about the table *m* used to calculate the edit (levenshtein) distance between words s_1 and s_2. Recall that cell m[i,j] (i.e. the edit distance between the first i characters of the first word (s_1) and the first j characters of the second (s_2)) is the minimum of m[i-1,j]+1, m[i,j-1]+1 and either m[i-1,j-1] (if s_1[i+1] = s_2[j+1]) or m[i-1,j-1]+1 otherwise.

What does each of these terms represent, in terms of character edits (insert, delete, replace)? Choose the option with the correct values to replace the one or multiple underlines ___. (the order matters, the first value should replace the first ___, the second value the second ___, etc.)

a) m[i-1,j]+1 is the cost of deleting s_1[___] from s_1[0 ... i], then editing s_1[0 ... i-1] into s_2[0 ... j].

◉ [i]
○ [i-1]
○ [j-1]
○ [j]

b) m[i,j-1]+1 is the cost of editing s_1[___] into s_2[___], then inserting s_2[___].

◉ [0 ... i], [0 ... j-1], [j]
○ [0 ... i], [0 ... j-1], [j-1]
○ [0 ... j-1], [0 ... i-1], [i]

c) m[i-1,j-1] is the cost of editing s_1[___] into s_2[___]. In addition, +1 is added to the cost for replacing s_1[___] with s_2[___], but only if they are not already the same.

○ [0 ... i-1], [0 ... j-1], [j], [i]
◉ [0 ... i-1], [0 ... j-1], [i], [j]
○ [0 ... j-1], [0 ... i-1], [j], [i]

28.03.2025

Tolerant Retrieval

Jaccard Coefficient (Part 1)

Calculating the edit distance between strings is expensive. A useful heuristic to estimate which strings are likely to have a small edit distance is the number of tri-grams (3-grams) they share. In this example, consider the symbol $ at the start and end of every word when computing tri-grams.

a) How many tri-grams does the misspelt word ***recivee*** share with possible correction ***receive***?

b) How many tri-grams does the misspelt word ***recivee*** share with possible correction ***recipe***?

Tolerant Retrieval

Jaccard Coefficient (Part 1)

Calculating the edit distance between strings is expensive. A useful heuristic to estimate which strings are likely to have a small edit distance is the number of tri-grams (3-grams) they share.
In this example, consider the symbol $ at the start and end of every word when computing tri-grams.

a) How many tri-grams does the misspelt word ***recivee*** share with possible correction ***receive***?

3

b) How many tri-grams does the misspelt word ***recivee*** share with possible correction ***recipe***?

3

Tolerant Retrieval

Jaccard Coefficient (Part 2)

Calculating the edit distance between strings is expensive. A useful heuristic to estimate which strings are likely to have a small edit distance is the number of tri-grams they share.
In this example, consider the symbol $ at the start and end of every word when computing tri-grams.

Counting the number of shared elements in two sets will be biased towards larger sets, as the larger the set, the more likely it is to contain a given element, simply by chance. The Jaccard coefficient corrects for this by normalising with respect to the size of the sets: $J(A,B) = |A \cap B| / |A \cup B|$

Calculate the Jaccard coefficients for the following word pairs:

a) The Jaccard coefficient for J(*recivee*, *recipe*) is ...

- ◯ 3/8
- ◯ 3/10
- ◯ 2/7

b) The Jaccard coefficient for J(*recivee*, *receive*) is ...

- ◯ 5/11
- ◯ 3/11
- ◯ 4/9

Bonus Exercise

Tolerant Retrieval

Jaccard Coefficient (Part 2)

Calculating the edit distance between strings is expensive. A useful heuristic to estimate which strings are likely to have a small edit distance is the number of tri-grams they share.
In this example, consider the symbol $ at the start and end of every word when computing tri-grams.

Counting the number of shared elements in two sets will be biased towards larger sets, as the larger the set, the more likely it is to contain a given element, simply by chance. The Jaccard coefficient corrects for this by normalising with respect to the size of the sets: $J(A,B) = |A \cap B| / |A \cup B|$

Calculate the Jaccard coefficients for the following word pairs:

a) The Jaccard coefficient for J(*recivee, recipe*) is ...

- ○ 3/8
- ● 3/10
- ○ 2/7

b) The Jaccard coefficient for J(*recivee, receive*) is ...

- ○ 5/11
- ● 3/11
- ○ 4/9

Tolerant Retrieval

Jaccard Coefficient (Part 3)

In this example, consider the symbol $ at the start and end of every word when computing tri-grams.

Which is the more likely correction for the misspelt word *recivee*, based on the Jaccard similarity index?

- ○ *recipe*
- ○ *receive*

Tolerant Retrieval

Jaccard Coefficient (Part 3)

In this example, consider the symbol $ at the start and end of every word when computing tri-grams.

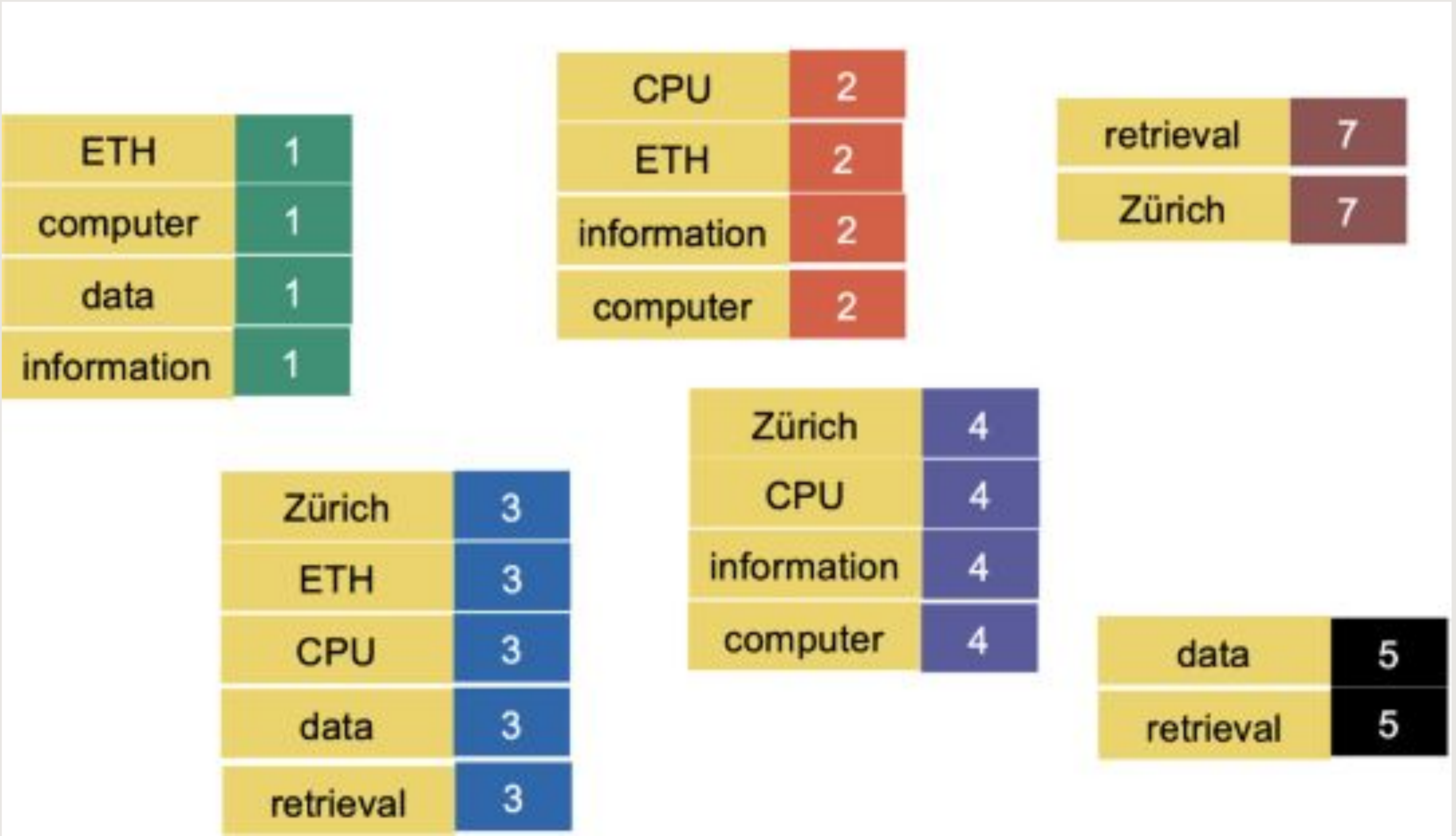
Which is the more likely correction for the misspelt word *recivee*, based on the Jaccard similarity index?

- (●) *recipe*
- ( ) *receive*

Term conversion

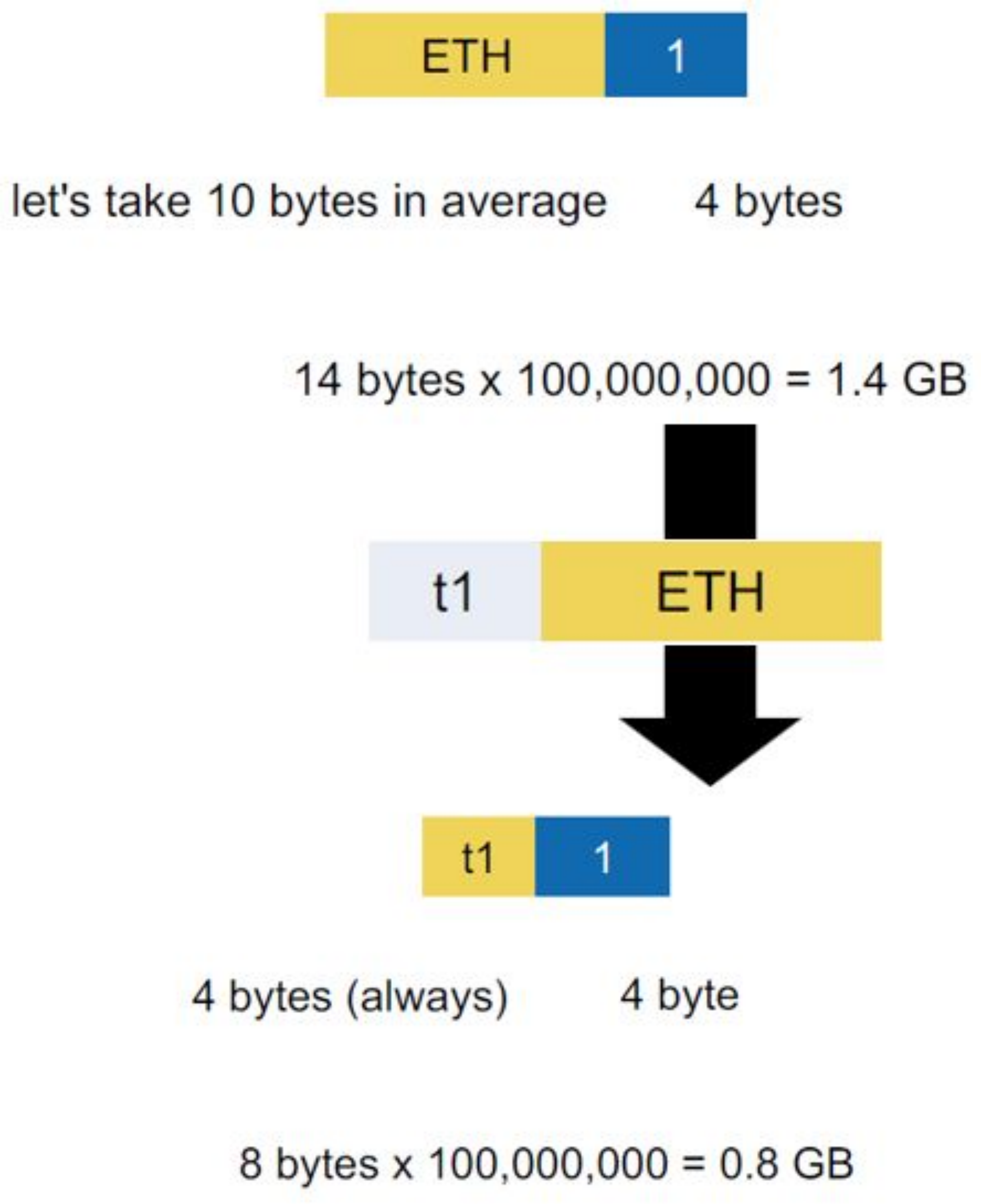
Inefficient!

Better: Use TermIDs



Term conversion

Term-TermID mapping



Constructing the index

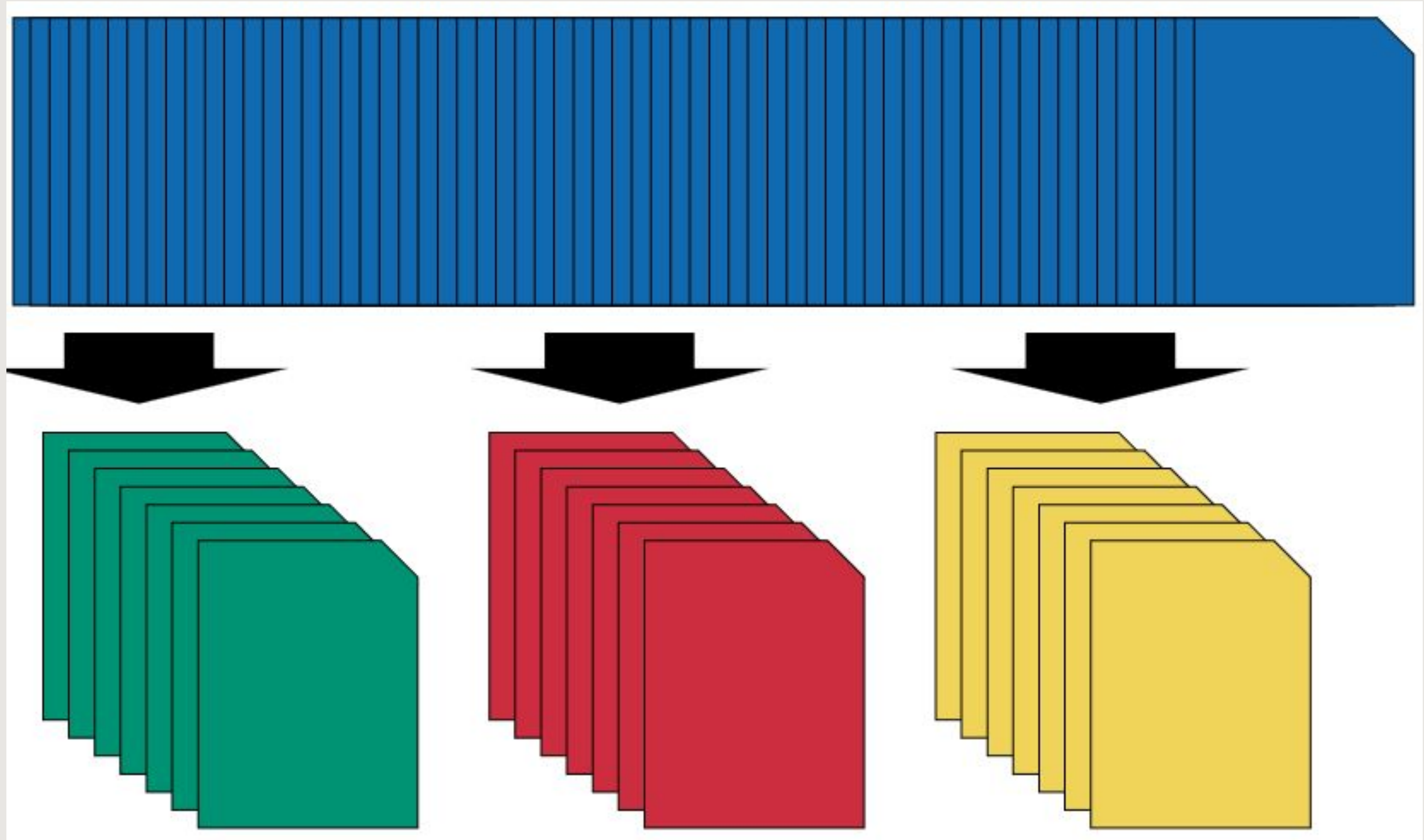
Optimize:

- Capacity (we want high)
- Latency (we want low)
- Throughput (we want high)

Use RAM for most of the work. Try to have few requests to disk.

Blocked Sort-Based Indexing (BSBI)

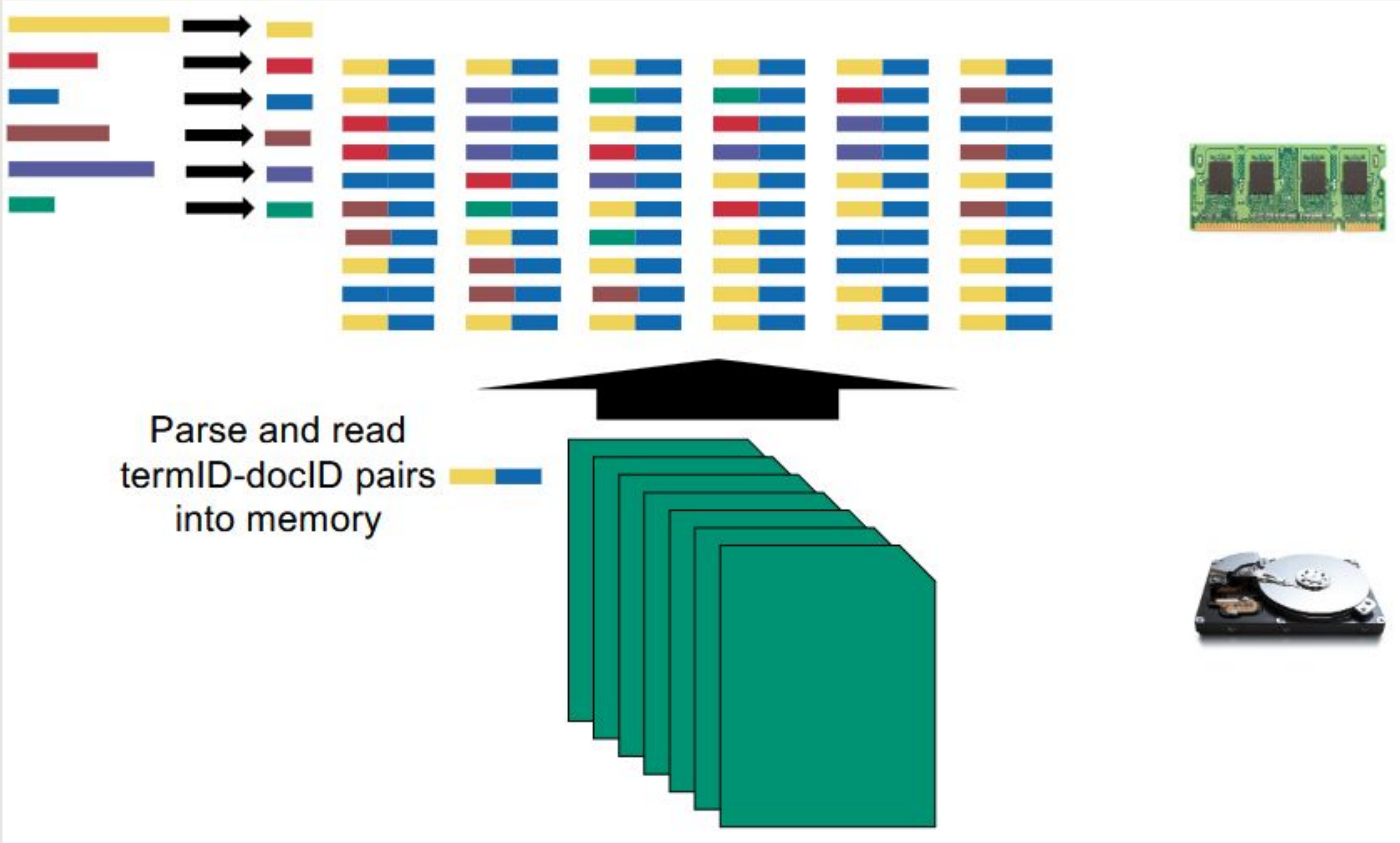
1. Shard the collection of documents (i.e. split them up into blocks)
-> Batch processing



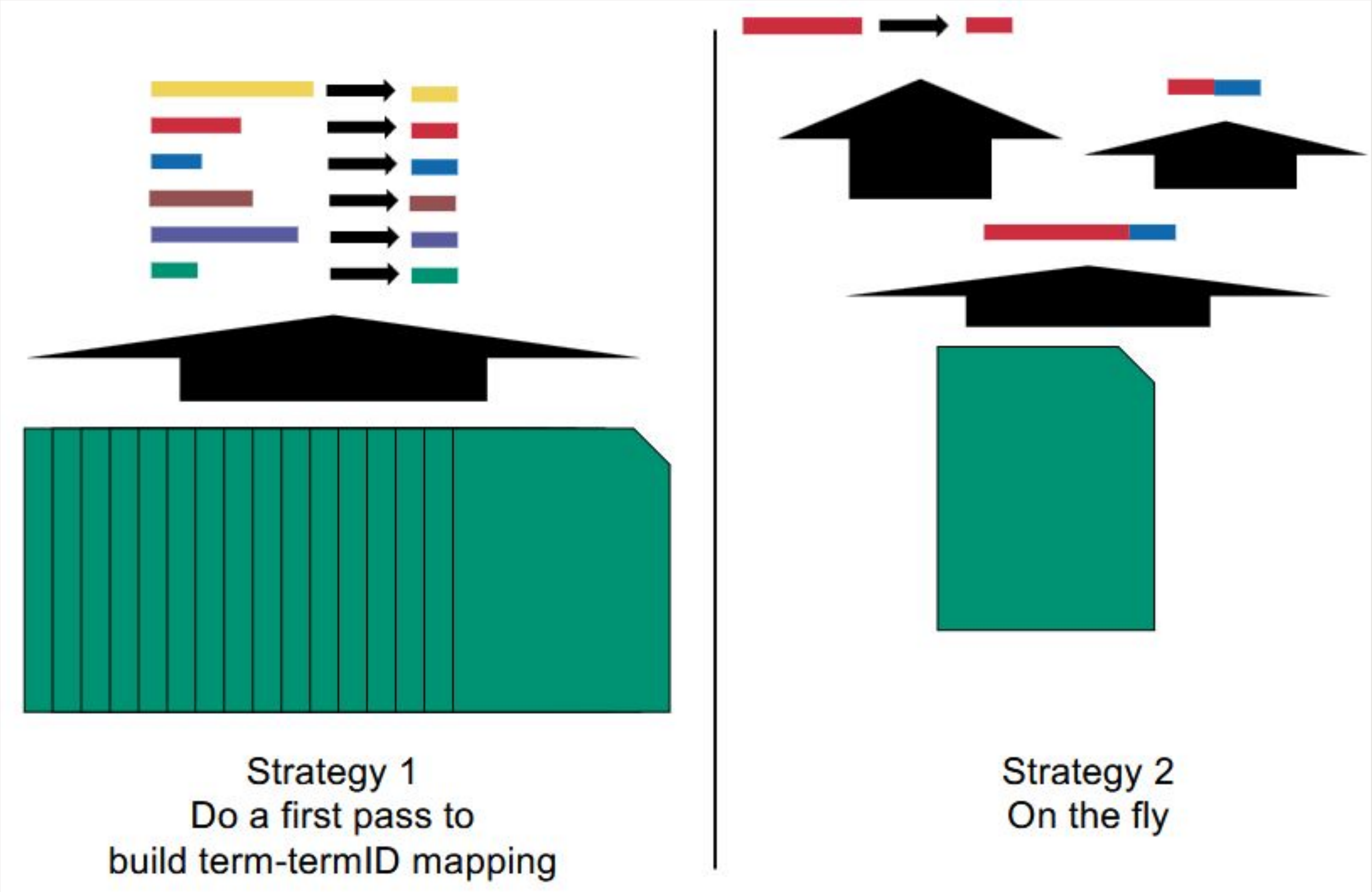
Blocked Sort-Based Indexing (BSBI)

2. Process each block one by one in memory

- Parse termID-docID pairs
- Sort pairs according to termID
- Write back intermediate results



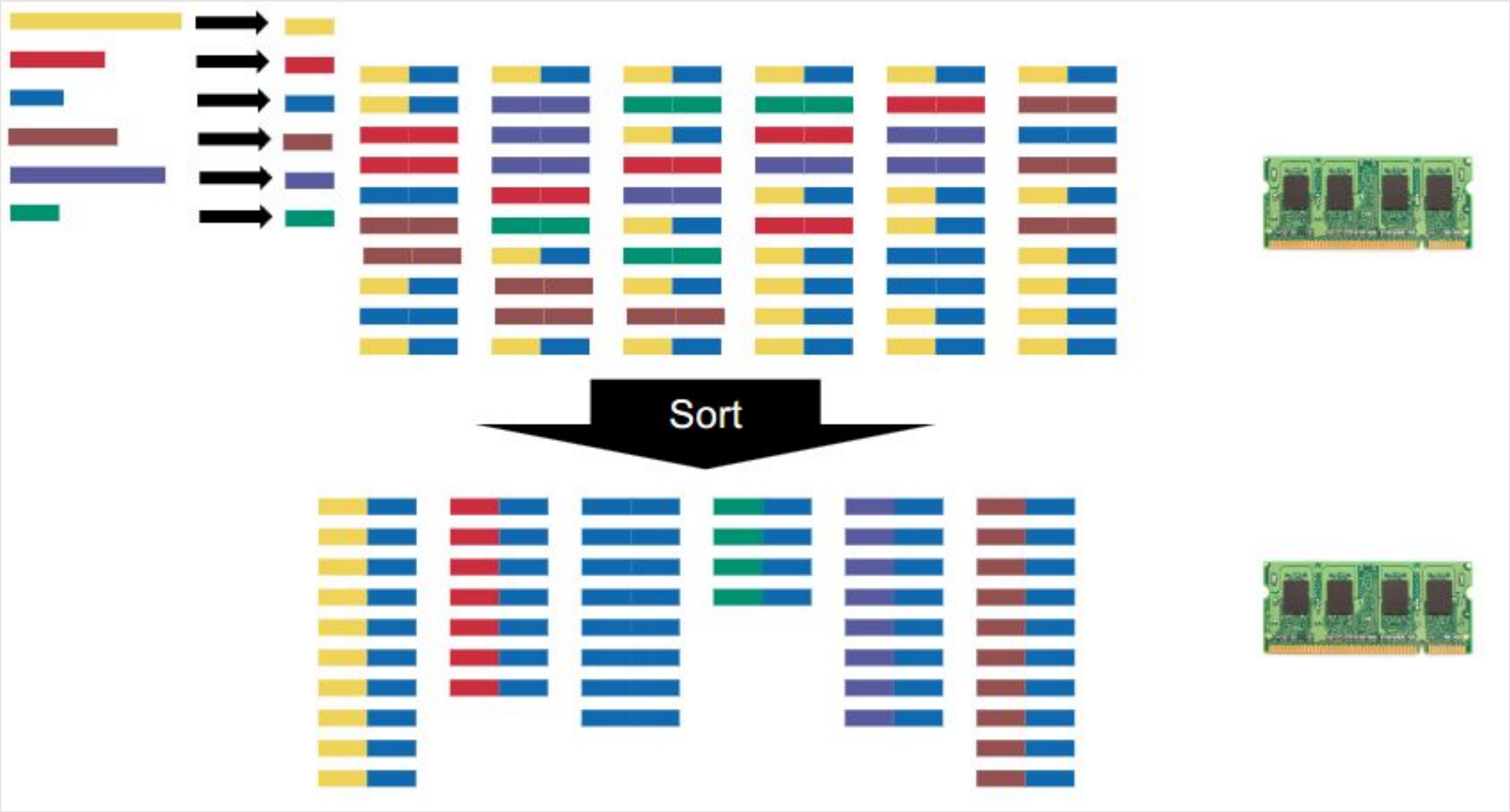
Generating termIDs



Blocked Sort-Based Indexing (BSBI)

2. Process each block one by one in memory

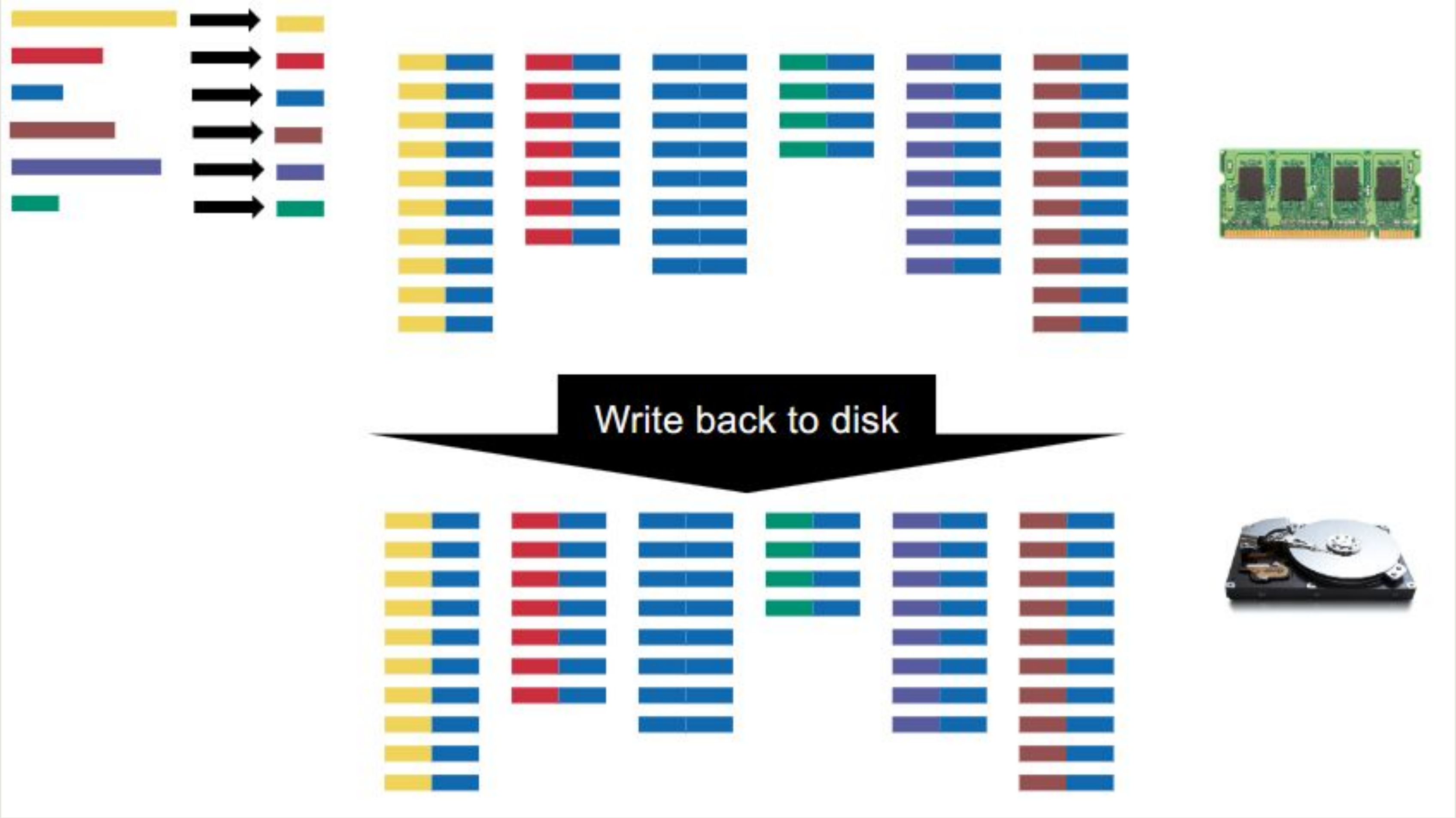
- Parse termID-docID pairs
- Sort pairs according to termID
- Write back intermediate results



Blocked Sort-Based Indexing (BSBI)

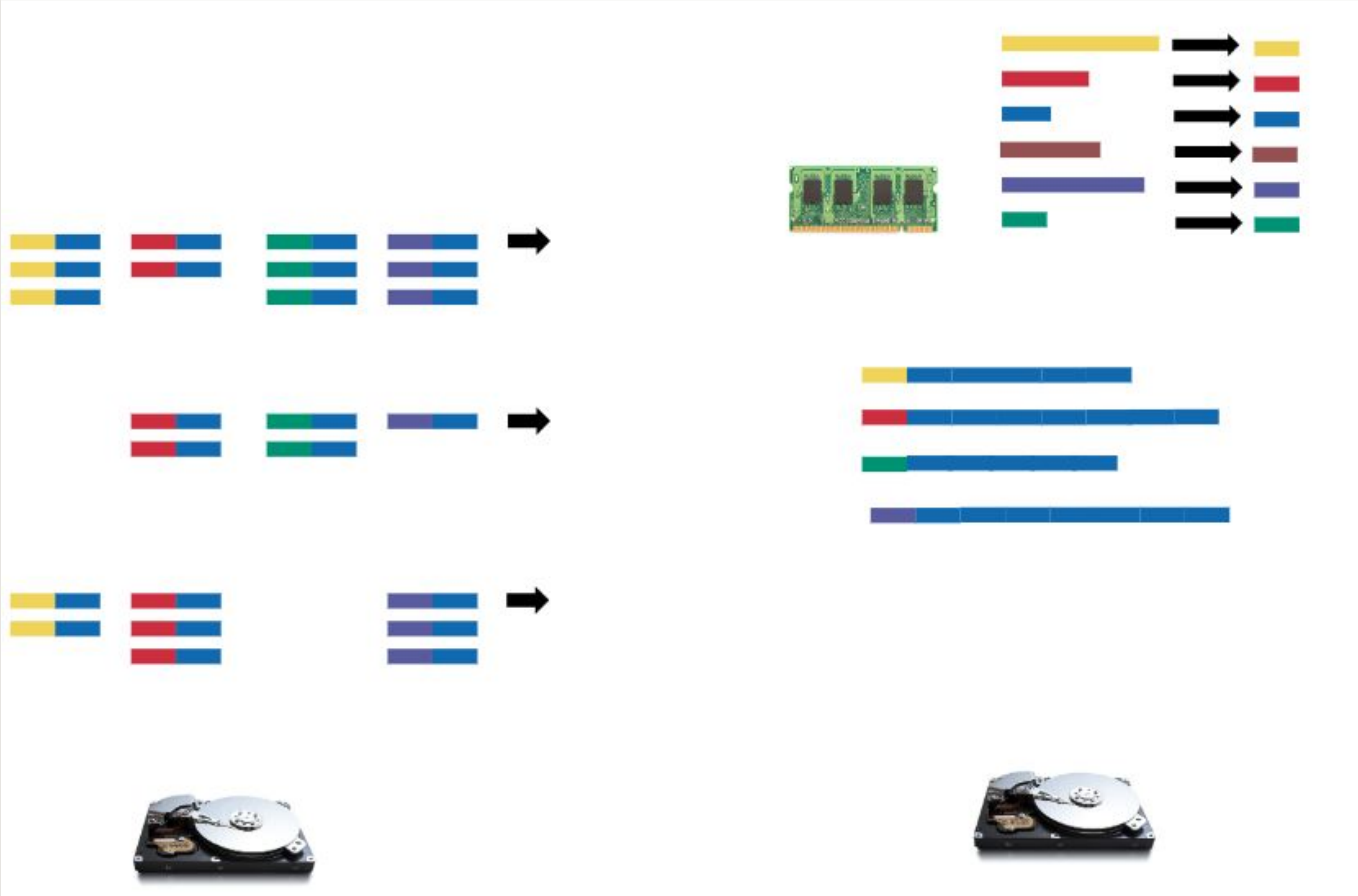
2. Process each block one by one in memory

- Parse termID-docID pairs
- Sort pairs according to termID
- Write back intermediate results



Blocked Sort-Based Indexing (BSBI)

3. Merge intermediate results into the final index.



Blocked Sort-Based Indexing (BSBI)

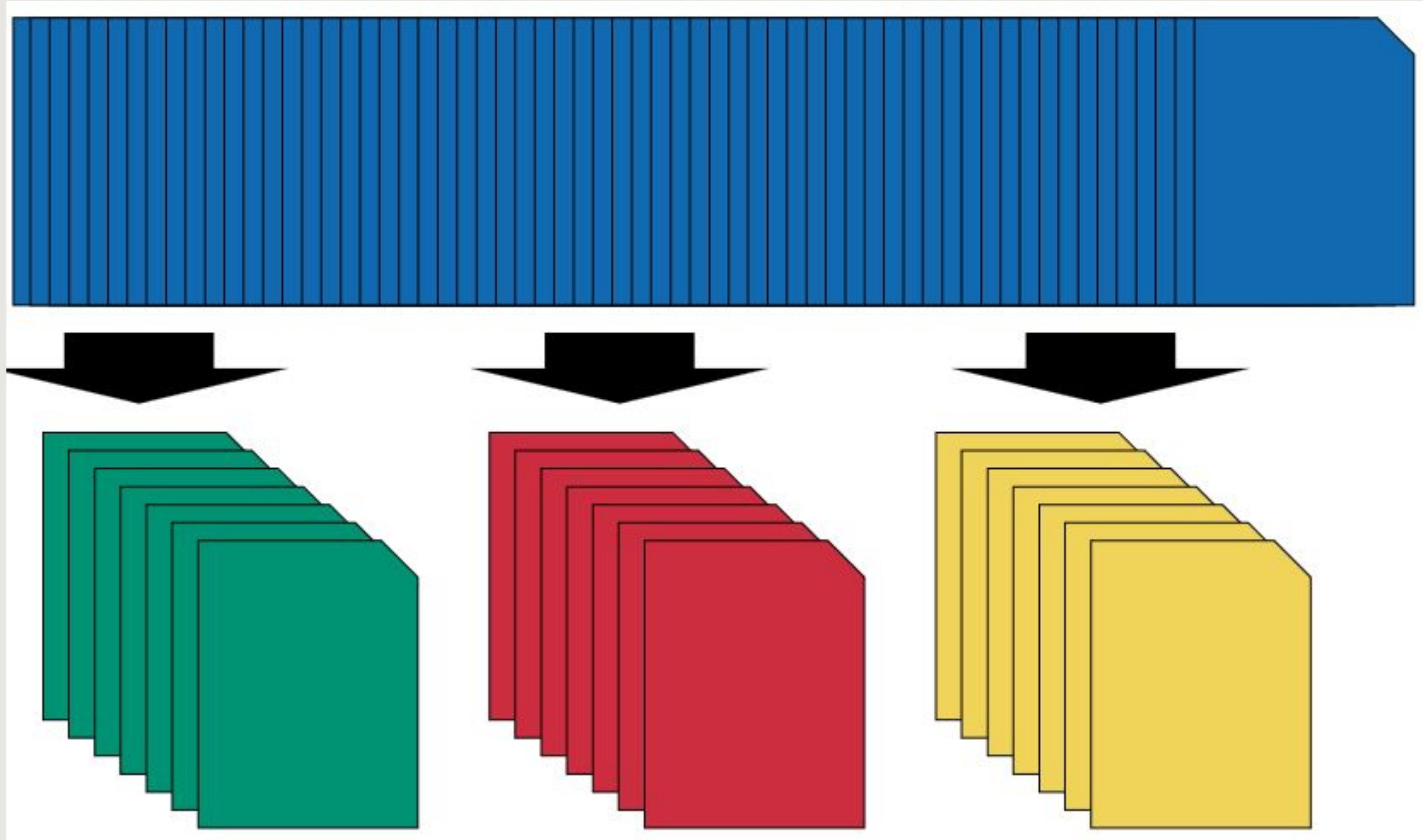
Complexity: O(T log T)

- First and second step (sorting): O(T log T)
- Third step (merging): O(T)

T = #tokens

Single-Pass In-Memory Indexing (SPIMI)

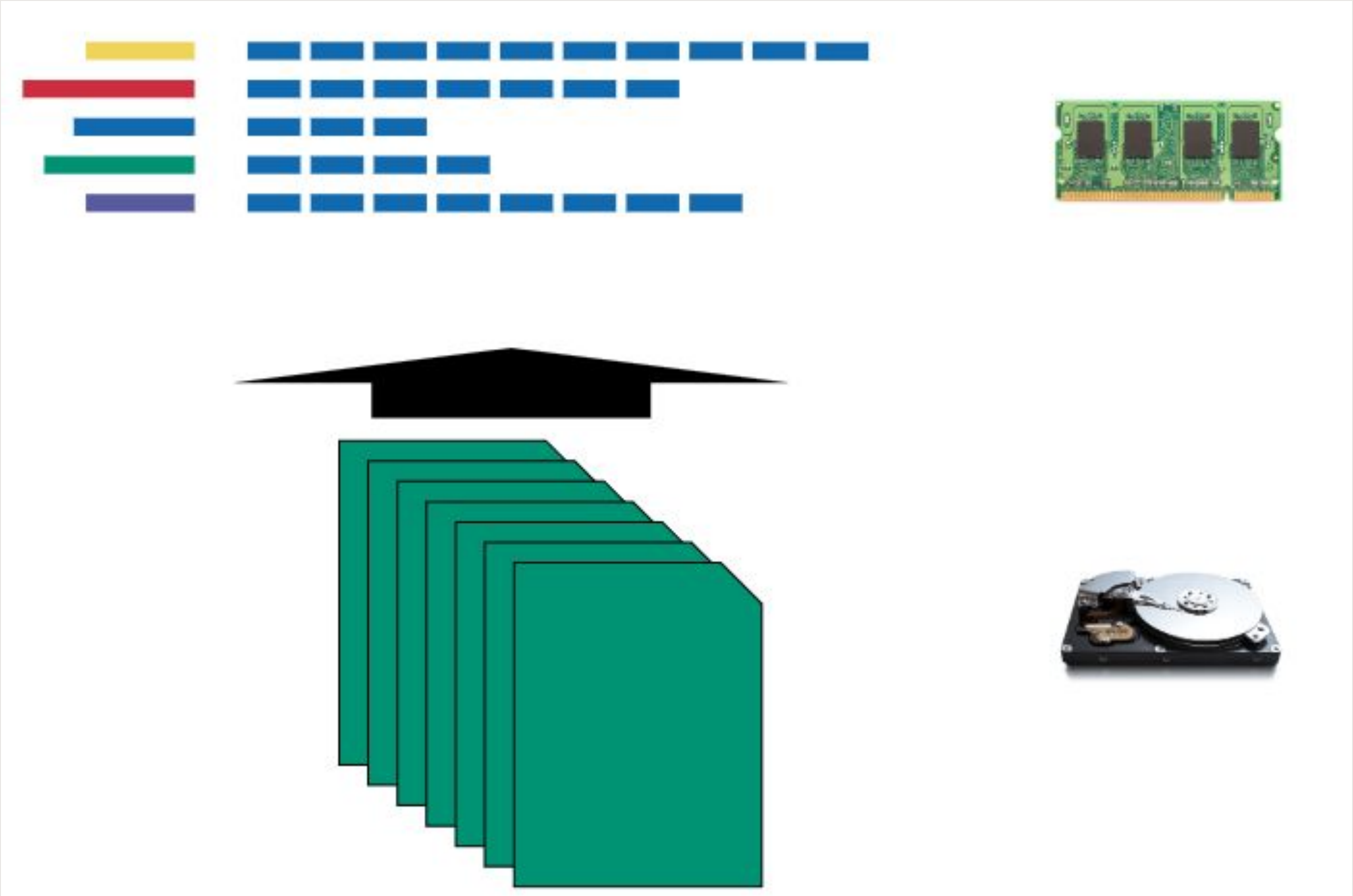
1. Shard collection of documents to blocks



Single-Pass in-Memory Indexing (SPIMI)

2. Process each block

- Parse term-docID pairs
- Create a dictionary
- Sort on terms
- Write back intermediate results



Single-Pass in-Memory Indexing (SPIMI)

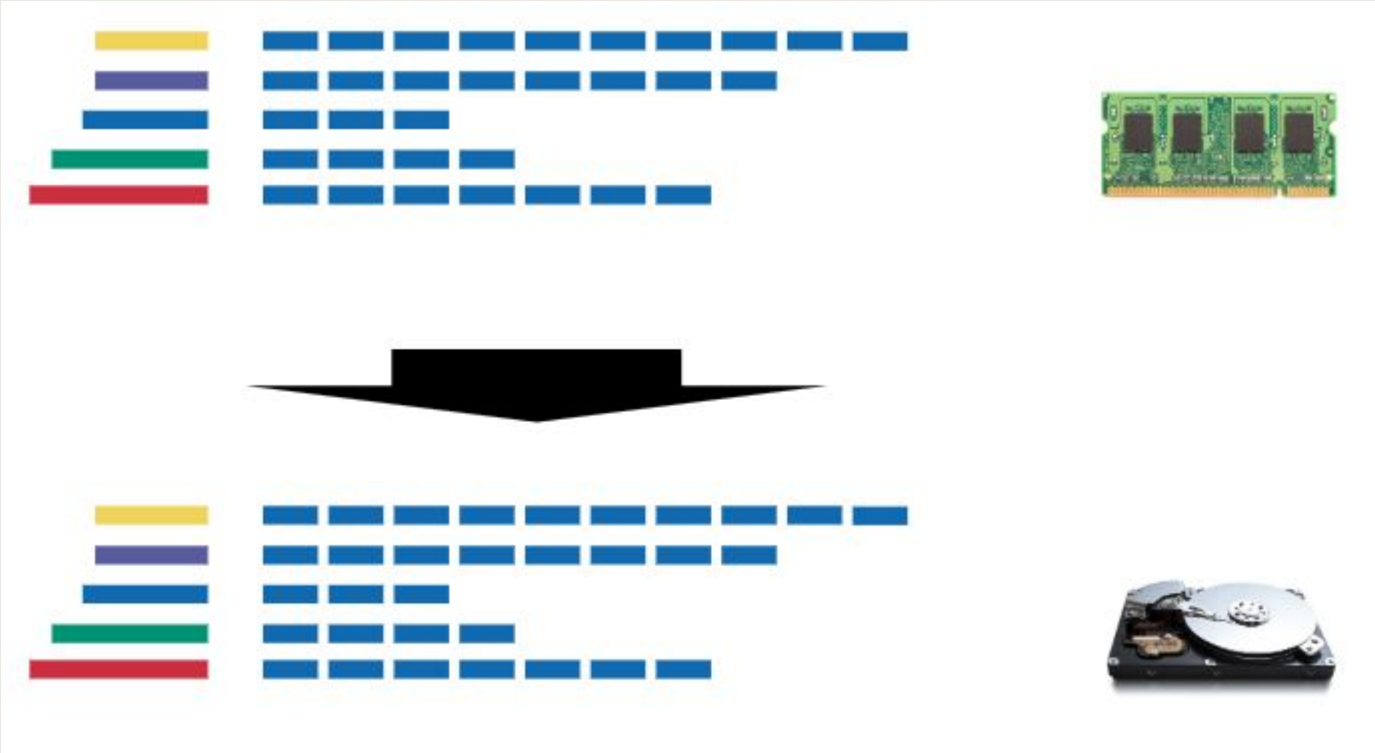
2. Process each block

- Parse term-docID pairs
- Create a dictionary
- Sort by terms
- Write back intermediate results

Single-Pass in-Memory Indexing (SPIMI)

2. Process each block
- Parse term-docID pairs
- Create a dictionary
- Sort by terms
- Write back intermediate results

-> No term-termID mapping in memory!



Single-Pass in-Memory Indexing (SPIMI)

3. Final merge

Single-Pass in-Memory Indexing (SPIMI)

Complexity: O(T)

- First step (processing): O(T log M)
- Second step (final merge): O(T)

T = #tokens, M = #terms

What is the difference?

BSBI

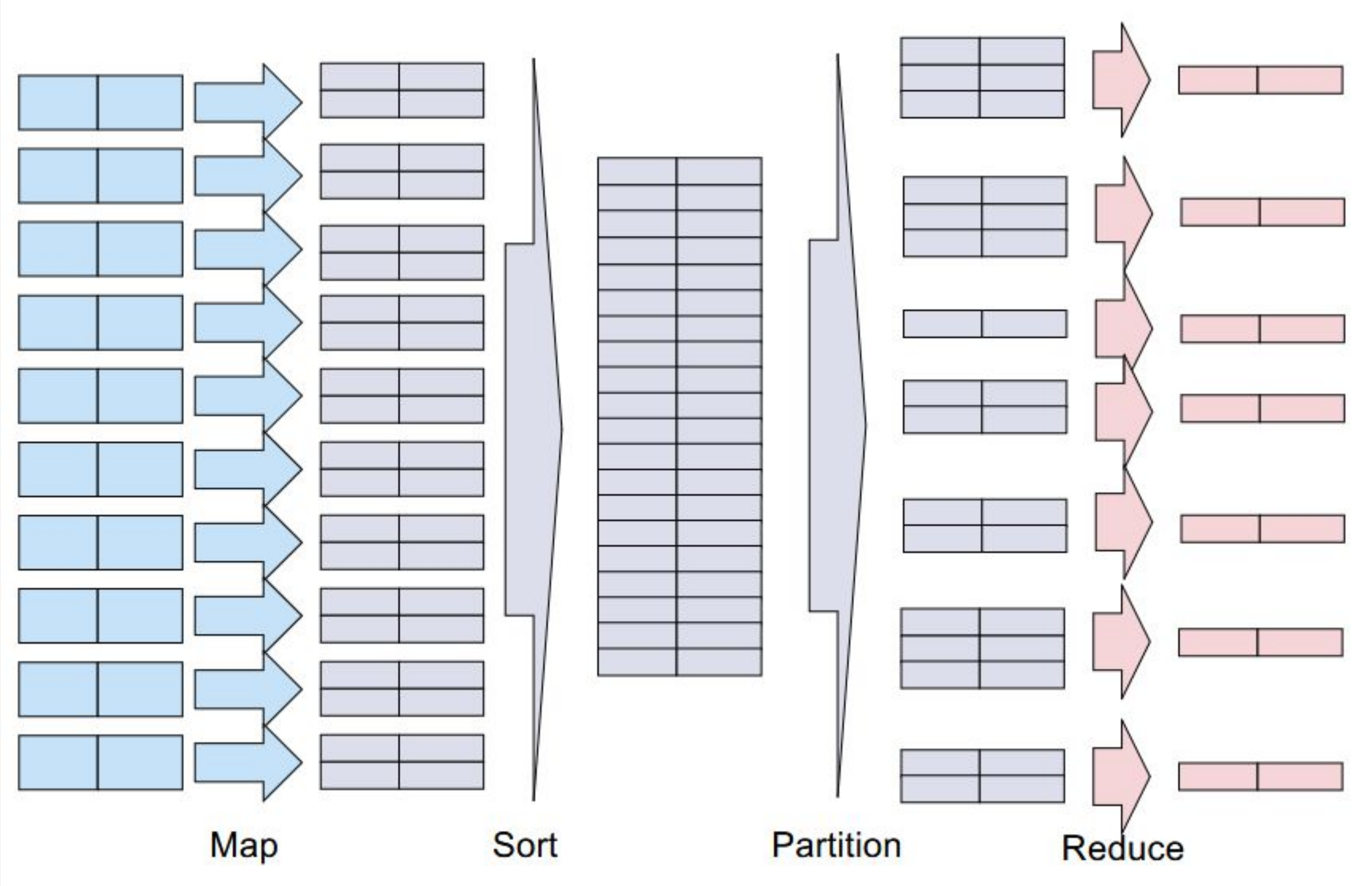
- Keeps term-termID mapping in memory
- First pass or “on the fly” for collecting term-termID mapping
- Merges to postings list in disk at the end

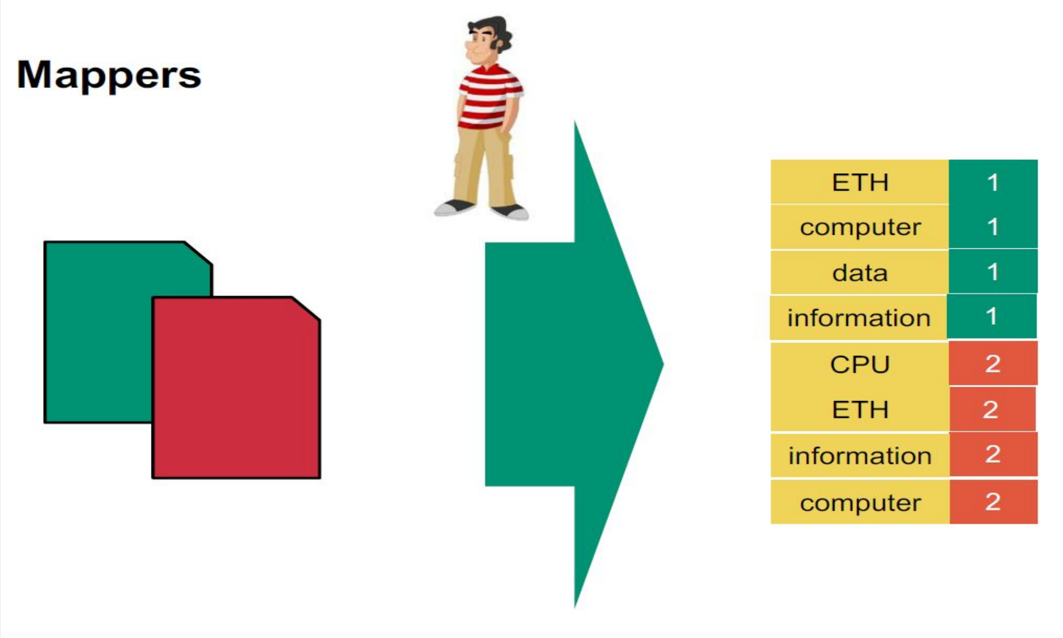
SPIMI

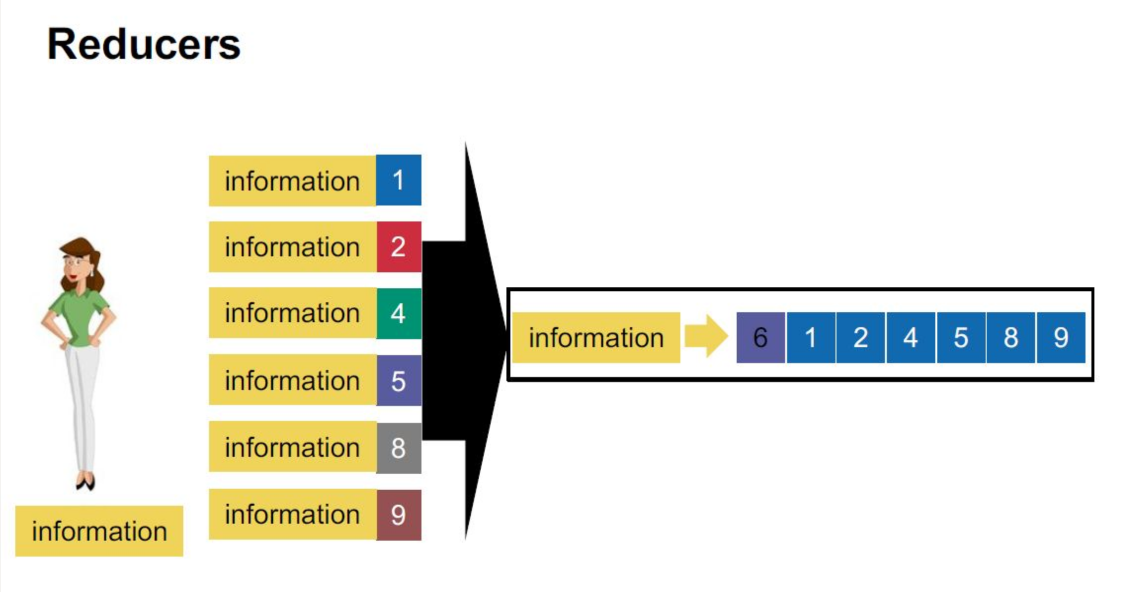
- Does not need a term-termID mapping
- Adds postings directly to intermediate postings lists
- More scalable since not limited by memory size
- Uses less memory, is faster

Index Construction

MapReduce

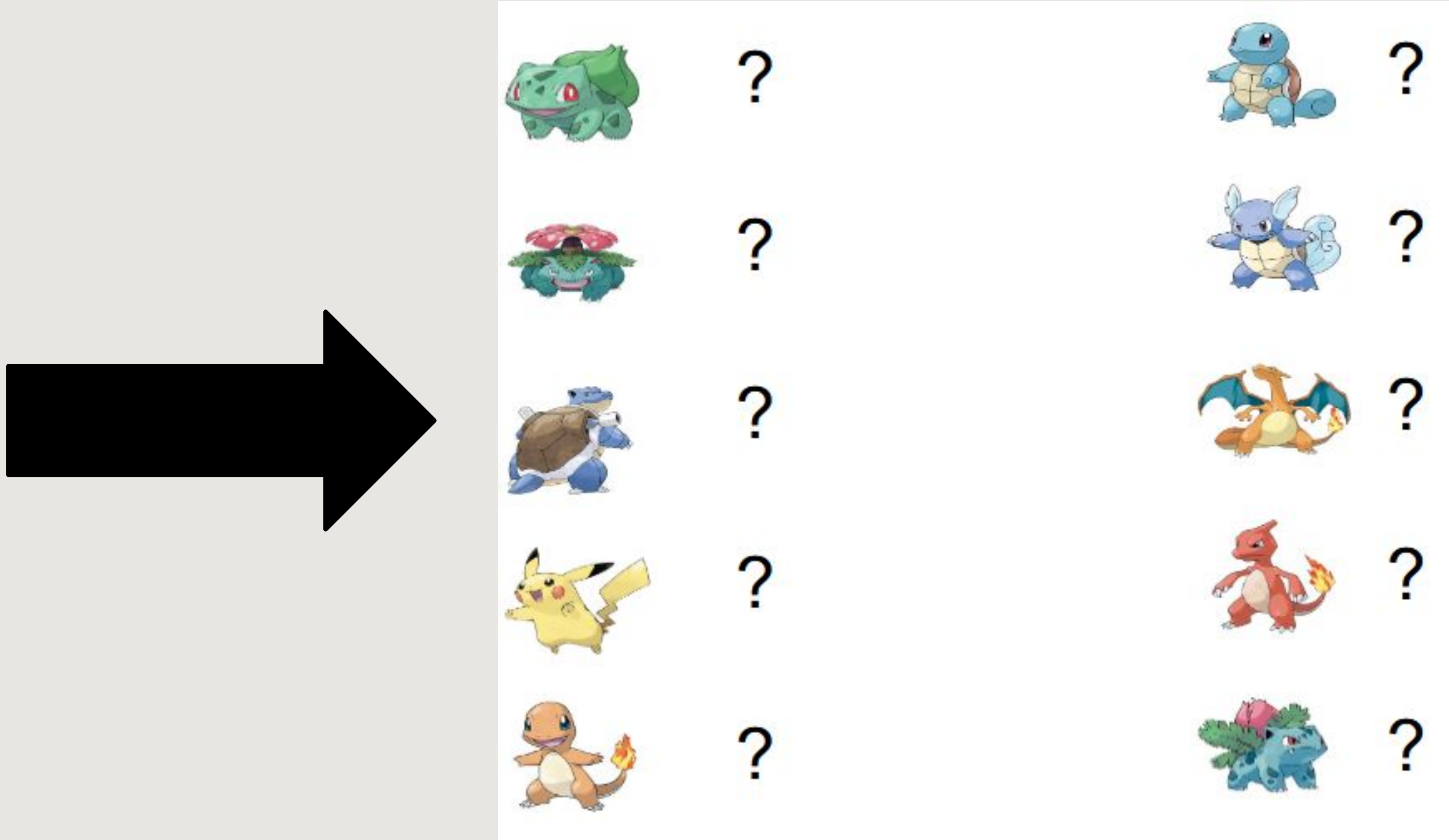




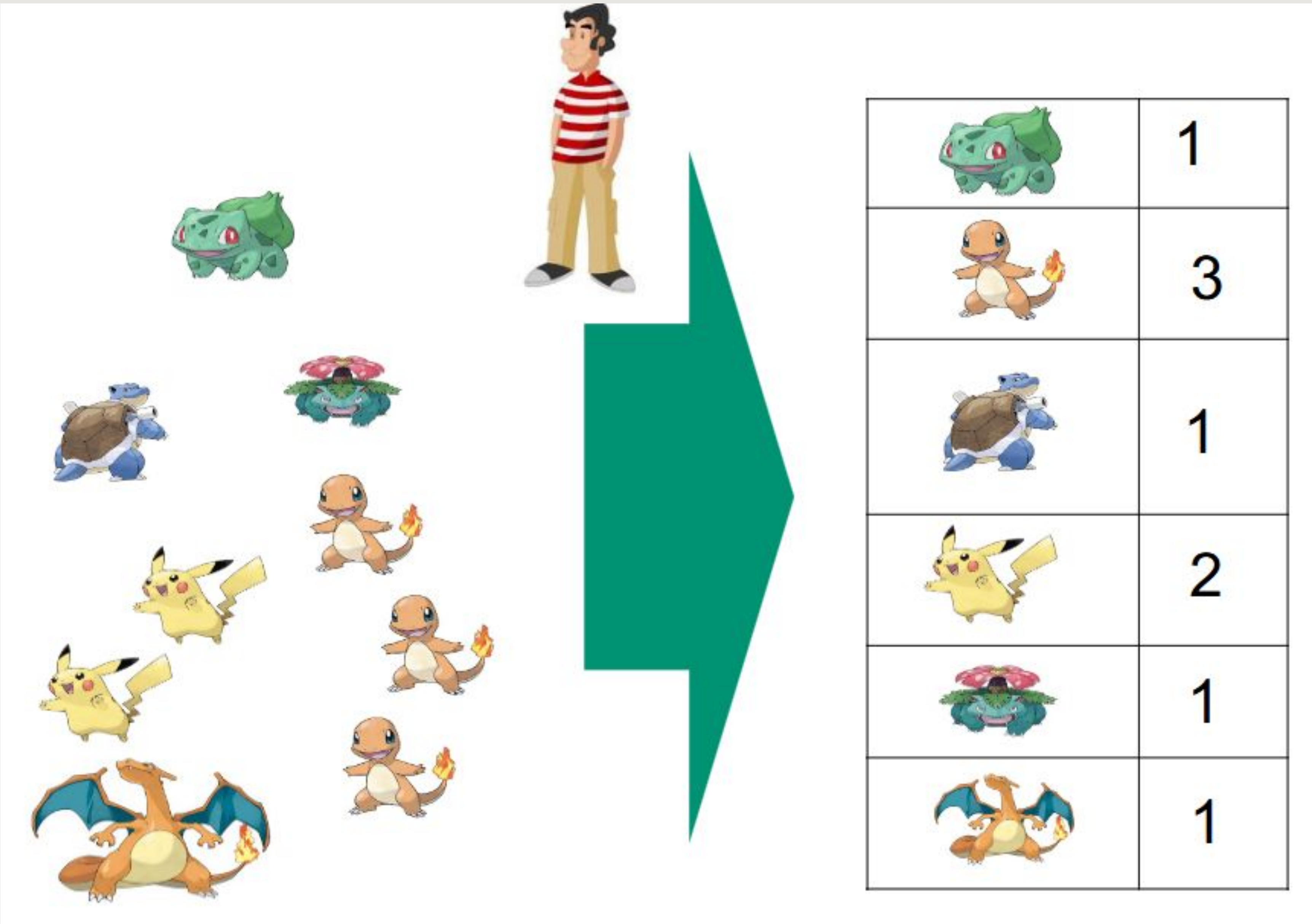


28.03.2025

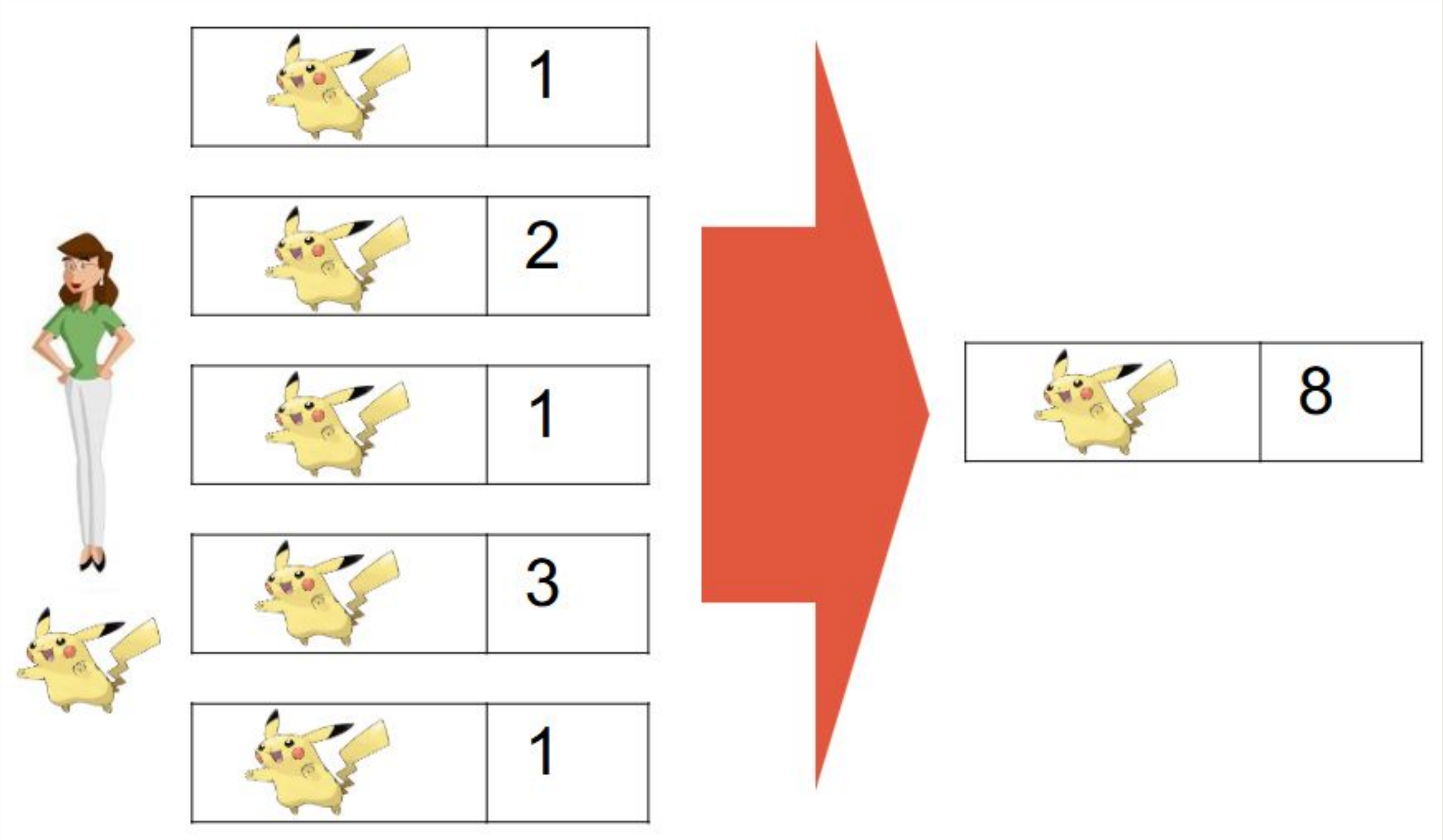
Counting Pokémon



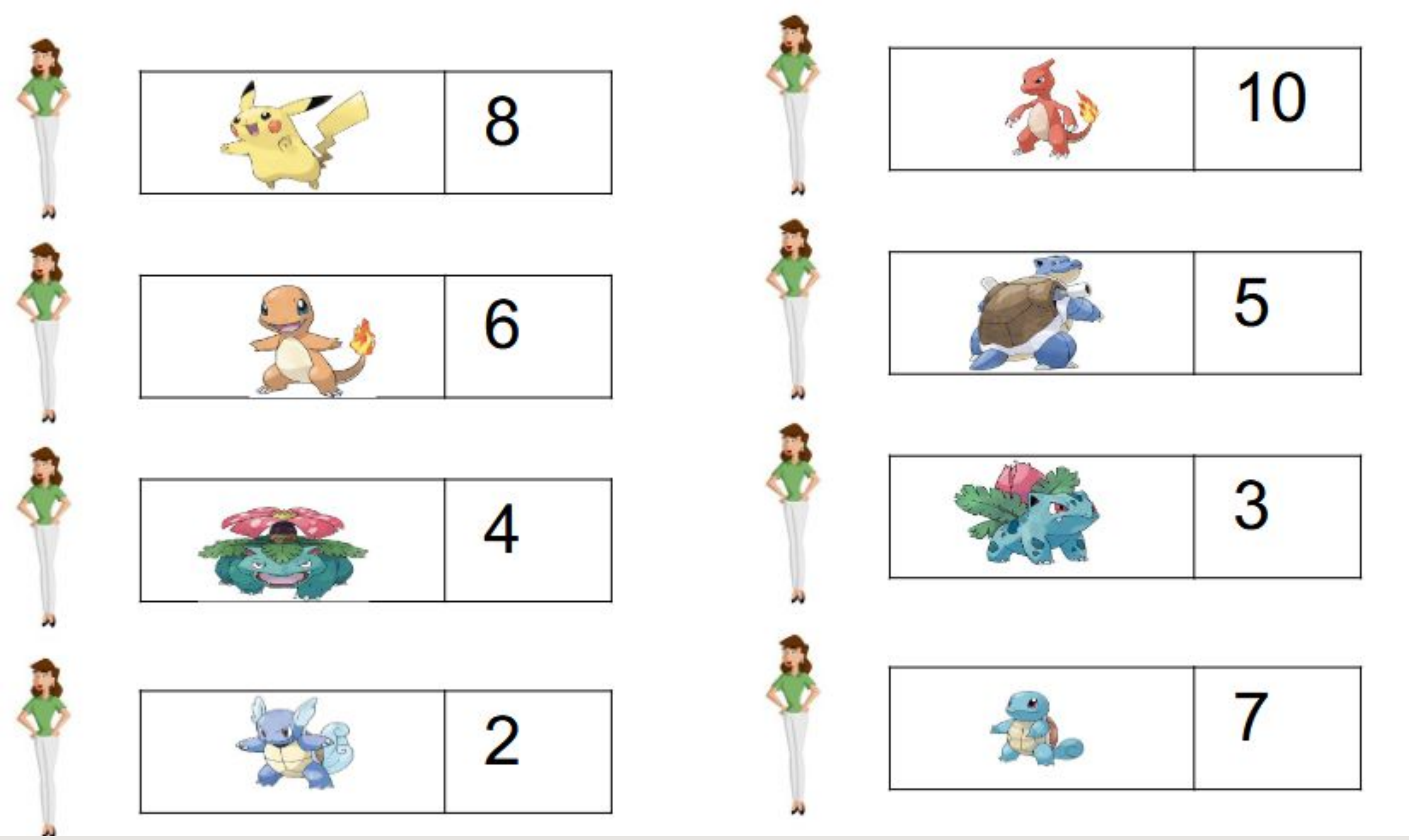
Mappers



Reducers



Final summary



Updating an index

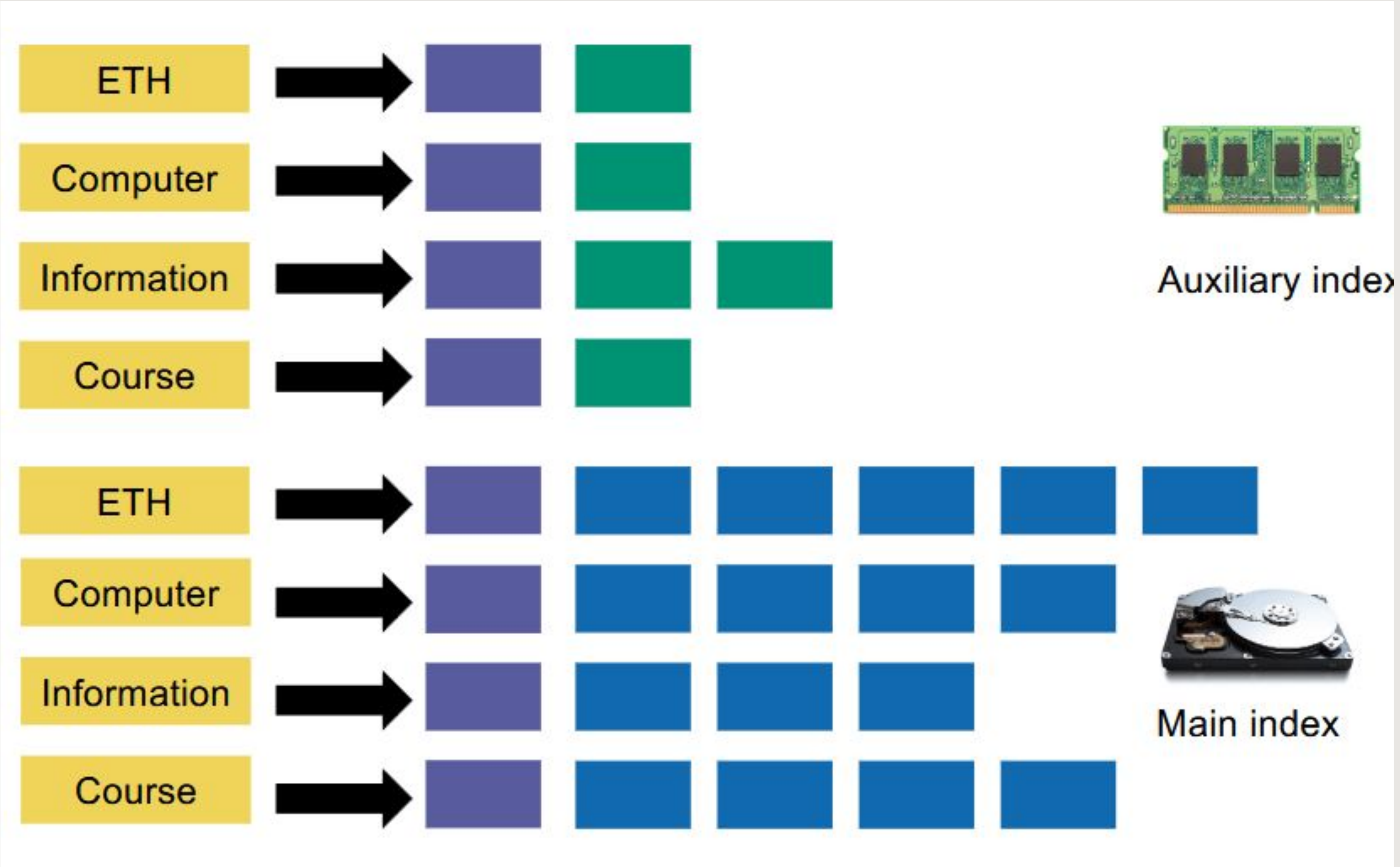
Large document collections are typically not static => documents being added, deleted, updated

Two ways:

- Periodic reconstruction
- Auxiliary index

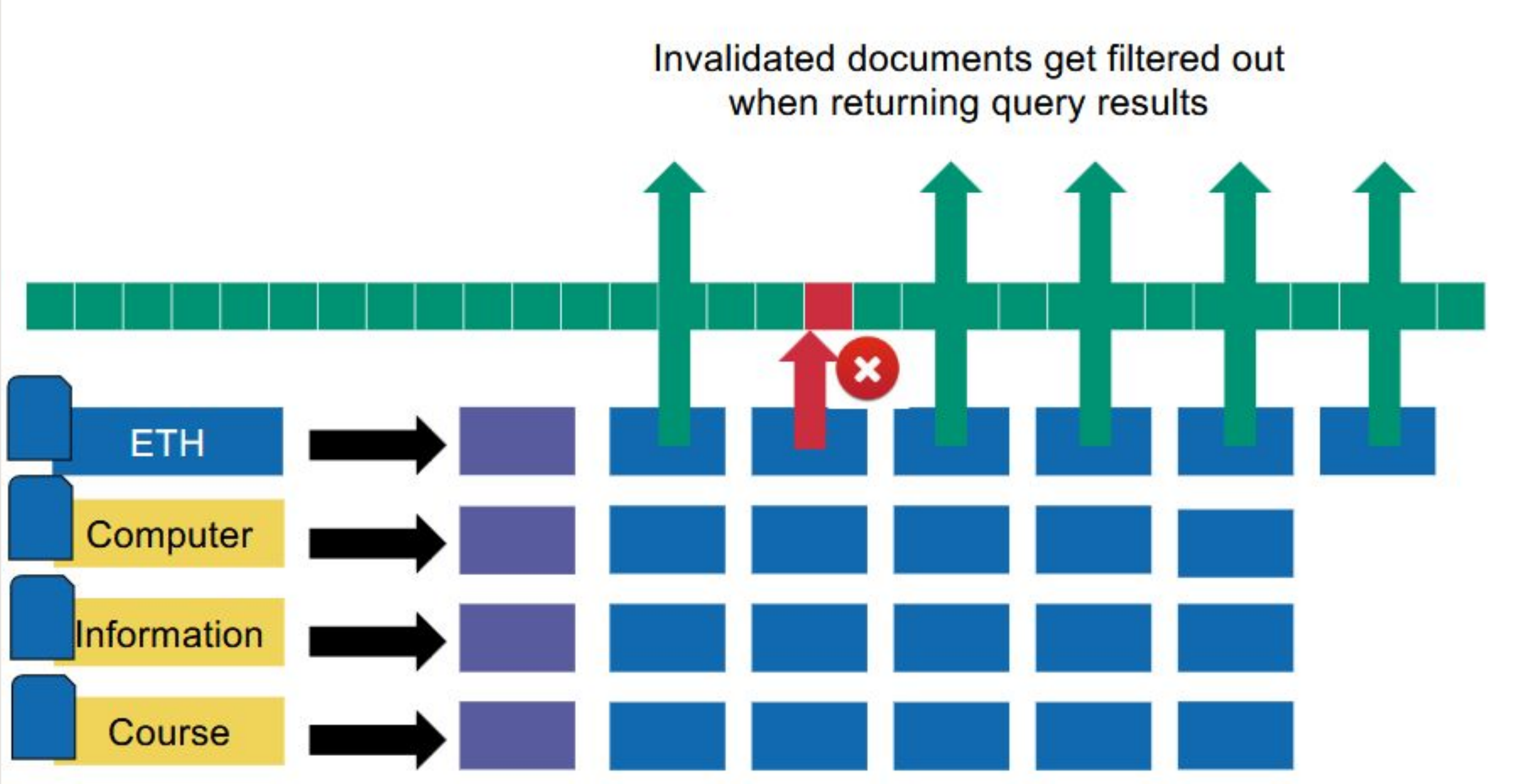
Auxiliary index

Store new documents in memory. Merge when memory full.



Deletion

Use invalid bit to filter results away.



Logarithmic merging

28.03.2025

Logarithmic merging

Complexity: $O(T \log (T / n)$ instead of $O(T^2 / n)$

n: size of auxiliary index

T: total number of postings

Exercise 4

Index construction

- Questions about BSBI / SPIMI
- Logarithmic merging

28.03.2025

Kahoot

https://create.kahoot.it/details/ex-04-index-construction/89fc6ef7-2262-4924-888d-9549940c0e74