Modular reference indexing with the de Bruijn graph: overview and challenges

Giulio Ermanno Pibiri

Ca' Foscari University of Venice and ISTI-CNR

@giulio_pibiri

@jermp

INRIA Rennes, France, 20 October 2022

About myself — Education

- Ph.D. in **Computer Science** from the **University of Pisa** in March 2019.
  (Compressed data structures with focus on application in IR and NLP.)

  Visiting Ph.D. student at RIKEN AIP (Tokyo, Japan) and University of Melbourne (Melbourne, Australia).

- Post-doc fellow in Computer Science at **ISTI-CNR** in Pisa from March 2019 - May 2022.

- As of June 2022: Assistant professor (tenure track) of Computer Science at **Ca' Foscari University of Venice**.

- More info at https://jermp.github.io.

About myself — Research

- Keywords: **Data Structures**, **Algorithms**, **Data Compression**, **Indexing**, **Efficiency**
- Design compressed data structures and algorithms to **index and search large quantities of data**.
- Efficiency is the key to:
  - **build better applications** in terms of reduced latency to access information (enhanced user experience);
  - **save computer resources** (power and storage machines) → **save money**.

About myself — Research

- Keywords: **Data Structures**, **Algorithms**, **Data Compression**, **Indexing**, **Efficiency**
- Design compressed data structures and algorithms to **index and search large quantities of data**.
- Efficiency is the key to:
  - **build better applications** in terms of reduced latency to access information (enhanced user experience);
  - **save computer resources** (power and storage machines) → **save money**.

Some Example Problems

- Inverted Indexes (TOIS'17, WSDM'19, TKDE'19, CSUR'20)
- Language Modeling (SIGIR'17, TOIS'19)
- RDF Triples (TKDE'20)
- Query Auto-Completion (SIGIR'20)
- Prefix-Sums (SPE'20)
- Bitmap Compression (DCC'21)
- Rank/Select Queries (INFOSYS'21)
- Minimal Perfect Hashing (SIGIR'21)
- **K-mer Dictionaries** (ISMB'22, WABI'22)

About myself — Research

- Keywords: **Data Structures**, **Algorithms**, **Data Compression**, **Indexing**, **Efficiency**
- Design compressed data structures and algorithms to **index and search large quantities of data**.
- Efficiency is the key to:
  - **build better applications** in terms of reduced latency to access information (enhanced user experience);
  - **save computer resources** (power and storage machines) → **save money**.

Some Example Problems

- Inverted Indexes (TOIS'17, WSDM'19, TKDE'19, CSUR'20)
- Language Modeling (SIGIR'17, TOIS'19)
- RDF Triples (TKDE'20)
- Query Auto-Completion (SIGIR'20)
- Prefix-Sums (SPE'20)
- Bitmap Compression (DCC'21)
- Rank/Select Queries (INFOSYS'21)
- Minimal Perfect Hashing (SIGIR'21)
- **K-mer Dictionaries** (ISMB'22, WABI'22)

https://github.com/jermp

The reference indexing problem

- We are given a collection $\mathscr{R} = \{R_1, \ldots, R_m\}$ of reference sequences. Each R_i is a (large) sequence over the DNA alphabet {A,C,G,T}, $1 \leq i \leq m$.

- **Problem.** We want to build an index for $\mathscr{R}$ so that we can answer the following queries efficiently for any k-mer x.
  - **Membership**: does x appear in $\mathscr{R}$?
  - **Count**: if so, how many times?
  - **Color**: and in what references?
  - **Locate**: and at what positions in the references?

The reference indexing problem

- We are given a collection $\mathscr{R} = \{R_1, \ldots, R_m\}$ of reference sequences. Each R_i is a (large) sequence over the DNA alphabet {A,C,G,T}, $1 \leq i \leq m$.

- **Problem.** We want to build an index for $\mathscr{R}$ so that we can answer the following queries efficiently for any k-mer x.
  - **Membership**: does x appear in $\mathscr{R}$?
  - **Count**: if so, how many times?
  - **Color**: and in what references?
  - **Locate**: and at what positions in the references?

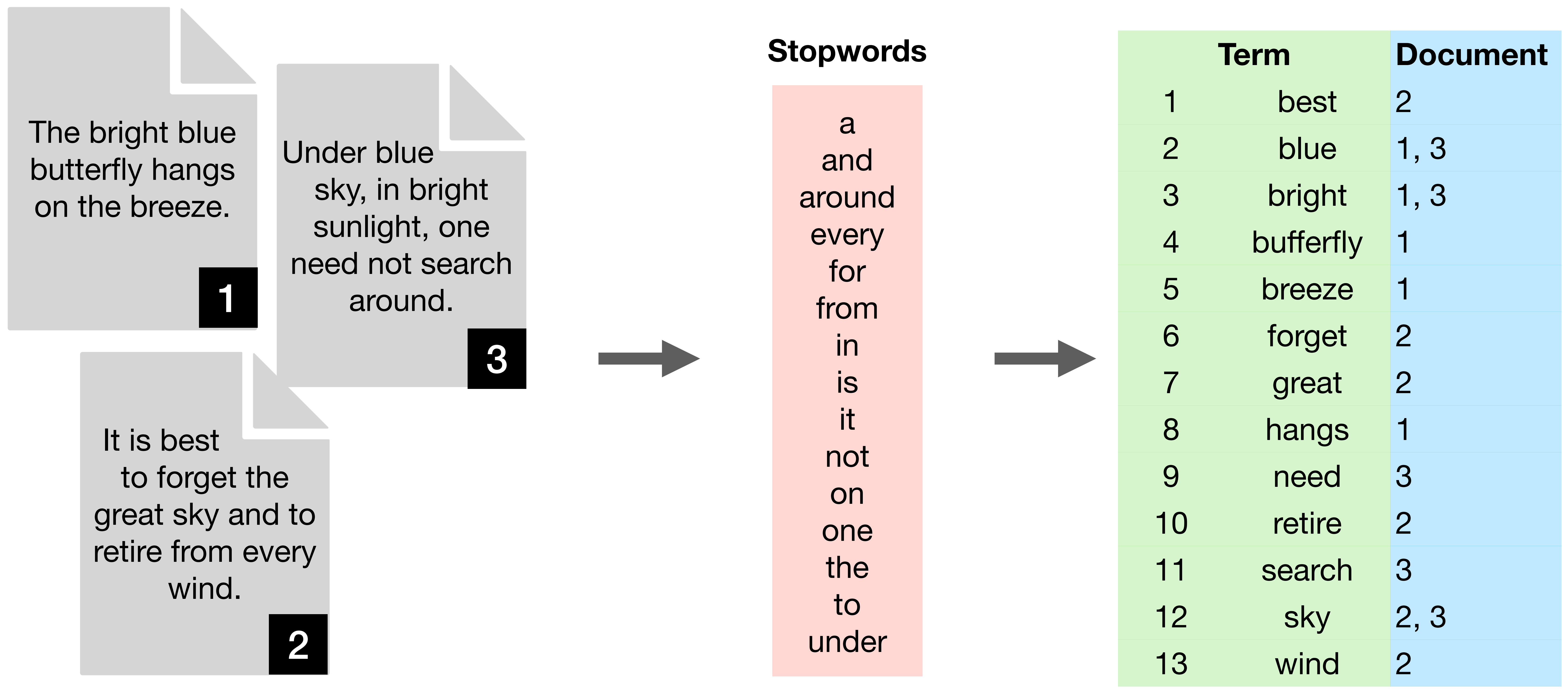
- Applications: This problem is crucial for all applications where sequences are first matched against known references (i.e., mapping/alignment algorithms): single-cell RNA-seq, metagenomics, etc.

A look back to IR

- The reference indexing problem has been studied for **decades** in Information Retrieval (IR).
- $\mathscr{R}$ is a collection of texts in natural-language (e.g., books, articles, Web pages, ect.).
- The classic solution is to use an *inverted index*.

A look back to IR

- The reference indexing problem has been studied for **decades** in Information Retrieval (IR).
- $\mathscr{R}$ is a collection of texts in natural-language (e.g., books, articles, Web pages, ect.).
- The classic solution is to use an *inverted index*.



1: The bright blue butterfly hangs on the breeze.

3: Under blue sky, in bright sunlight, one need not search around.

2: It is best to forget the great sky and to retire from every wind.

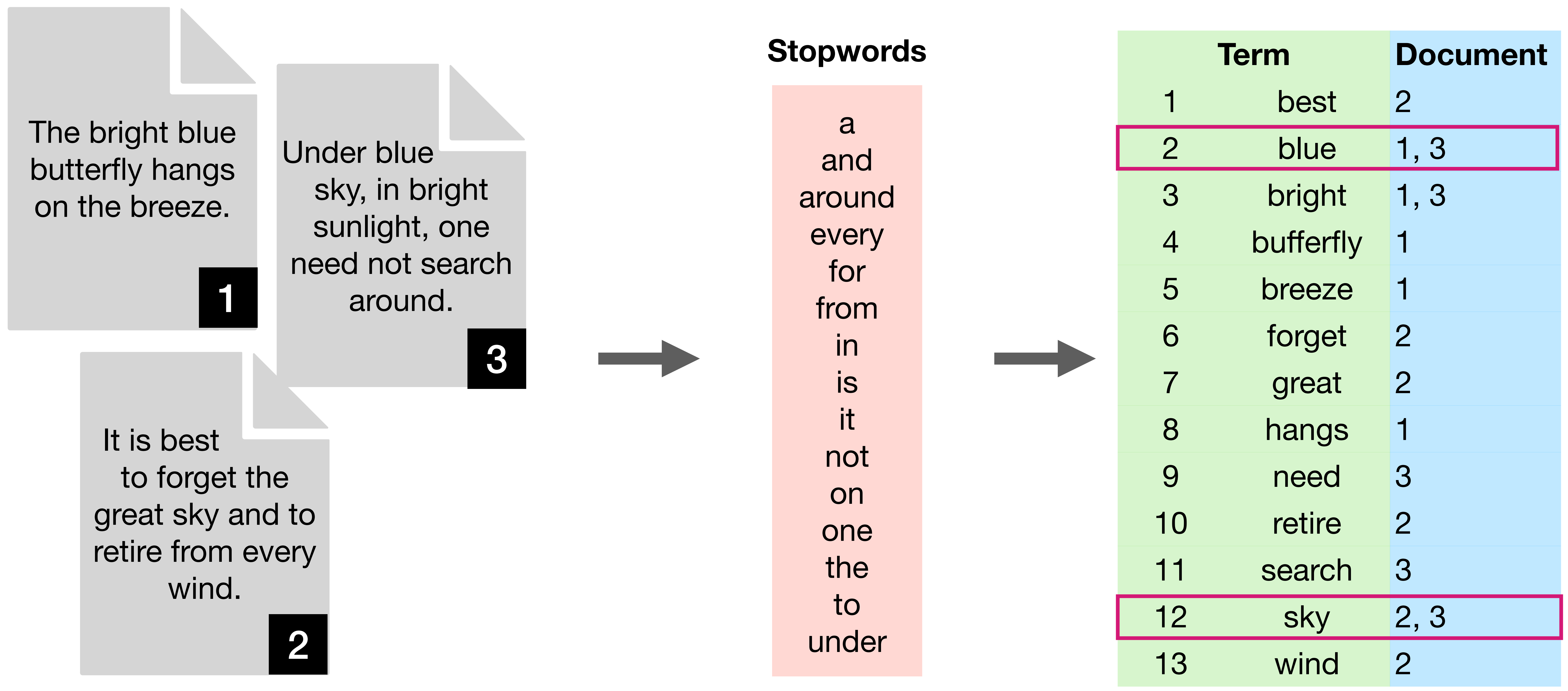
Stopwords

a
and
around
every
for
from
in
is
it
not
on
one
the
to
under

	Term	Document
1	best	2
2	blue	1, 3
3	bright	1, 3
4	bufferfly	1
5	breeze	1
6	forget	2
7	great	2
8	hangs	1
9	need	3
10	retire	2
11	search	3
12	sky	2, 3
13	wind	2

A look back to IR

- The reference indexing problem has been studied for **decades** in Information Retrieval (IR).
- $\mathscr{R}$ is a collection of texts in natural-language (e.g., books, articles, Web pages, ect.).
- The classic solution is to use an *inverted index*.



The bright blue butterfly hangs on the breeze. 1

It is best to forget the great sky and to retire from every wind. 2

Under blue sky, in bright sunlight, one need not search around. 3

Stopwords

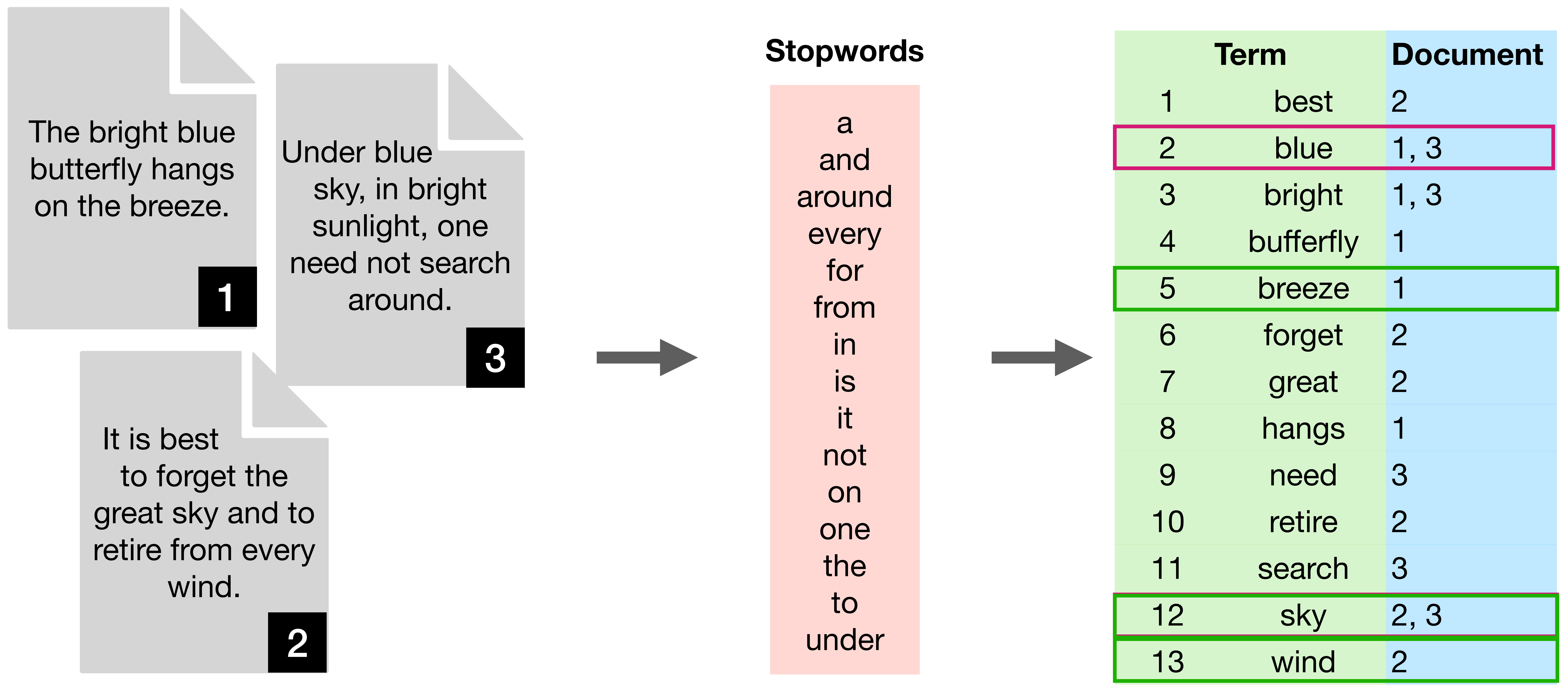
a
and
around
every
for
from
in
is
it
not
on
one
the
to
under

	Term	Document
1	best	2
2	blue	1, 3
3	bright	1, 3
4	bufferfly	1
5	breeze	1
6	forget	2
7	great	2
8	hangs	1
9	need	3
10	retire	2
11	search	3
12	sky	2, 3
13	wind	2

Query1 = “blue **AND** sky”
$\rightarrow [1,3] \cap [2,3] = [3]$

A look back to IR

- The reference indexing problem has been studied for **decades** in Information Retrieval (IR).
- $\mathscr{R}$ is a collection of texts in natural-language (e.g., books, articles, Web pages, ect.).
- The classic solution is to use an *inverted index*.



The bright blue butterfly hangs on the breeze. 1

Under blue sky, in bright sunlight, one need not search around. 3

It is best to forget the great sky and to retire from every wind. 2

Stopwords

a
and
around
every
for
from
in
is
it
not
on
one
the
to
under

	Term	**Document**
1	best	2
2	blue	1, 3
3	bright	1, 3
4	bufferfly	1
5	breeze	1
6	forget	2
7	great	2
8	hangs	1
9	need	3
10	retire	2
11	search	3
12	sky	2, 3
13	wind	2

Query1 = “blue **AND** sky”
$\rightarrow [1,3] \cap [2,3] = [3]$

Query2 = “breeze **OR** wind **OR** sky”
$\rightarrow [1] \cup [2] \cup [2,3] = [1,2,3]$

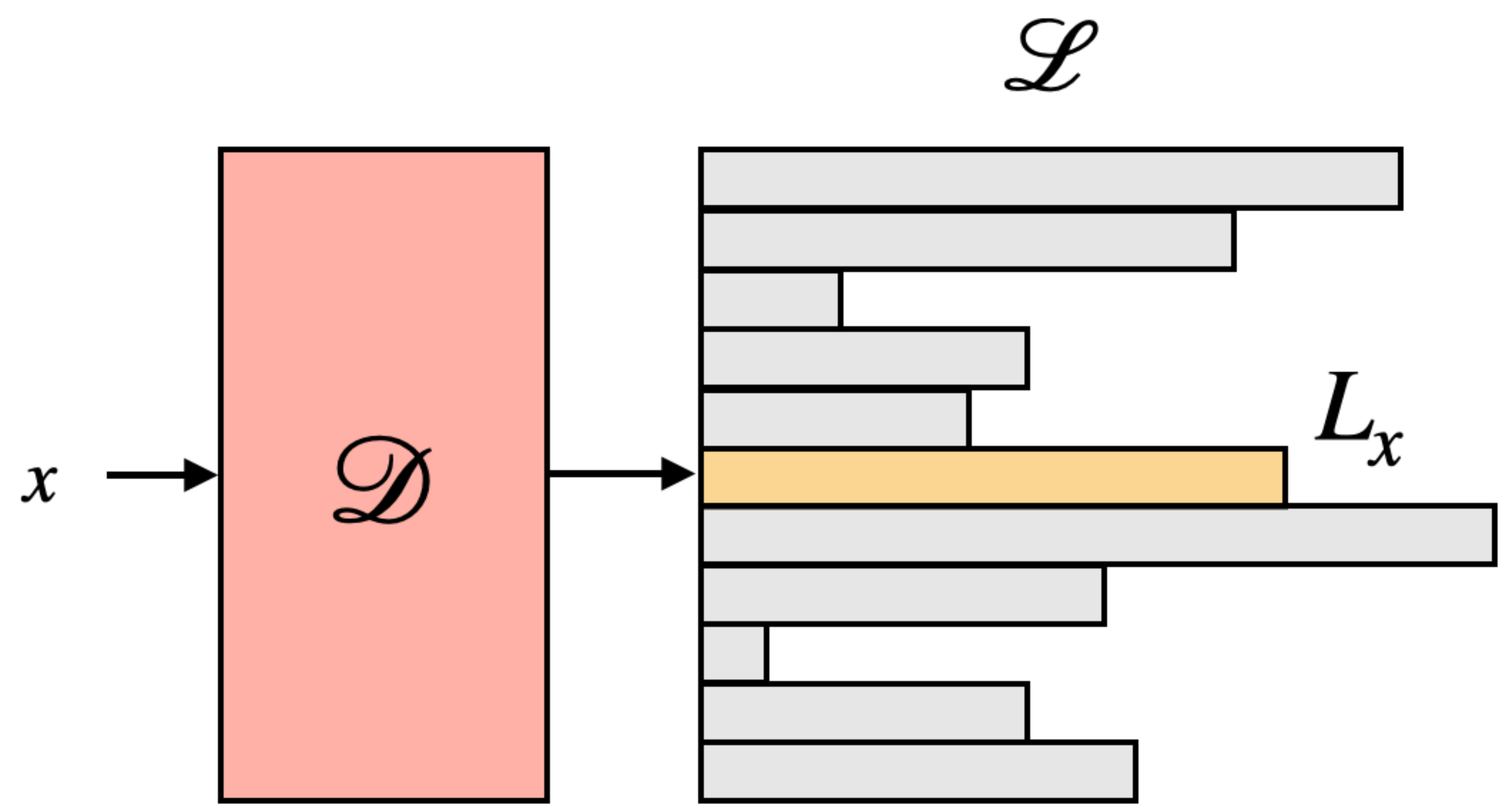
Our setting

- All the distinct k-mers in $\mathscr{R} = \{R_1, \ldots, R_m\}$ are the **dictionary** $\mathscr{D}$.

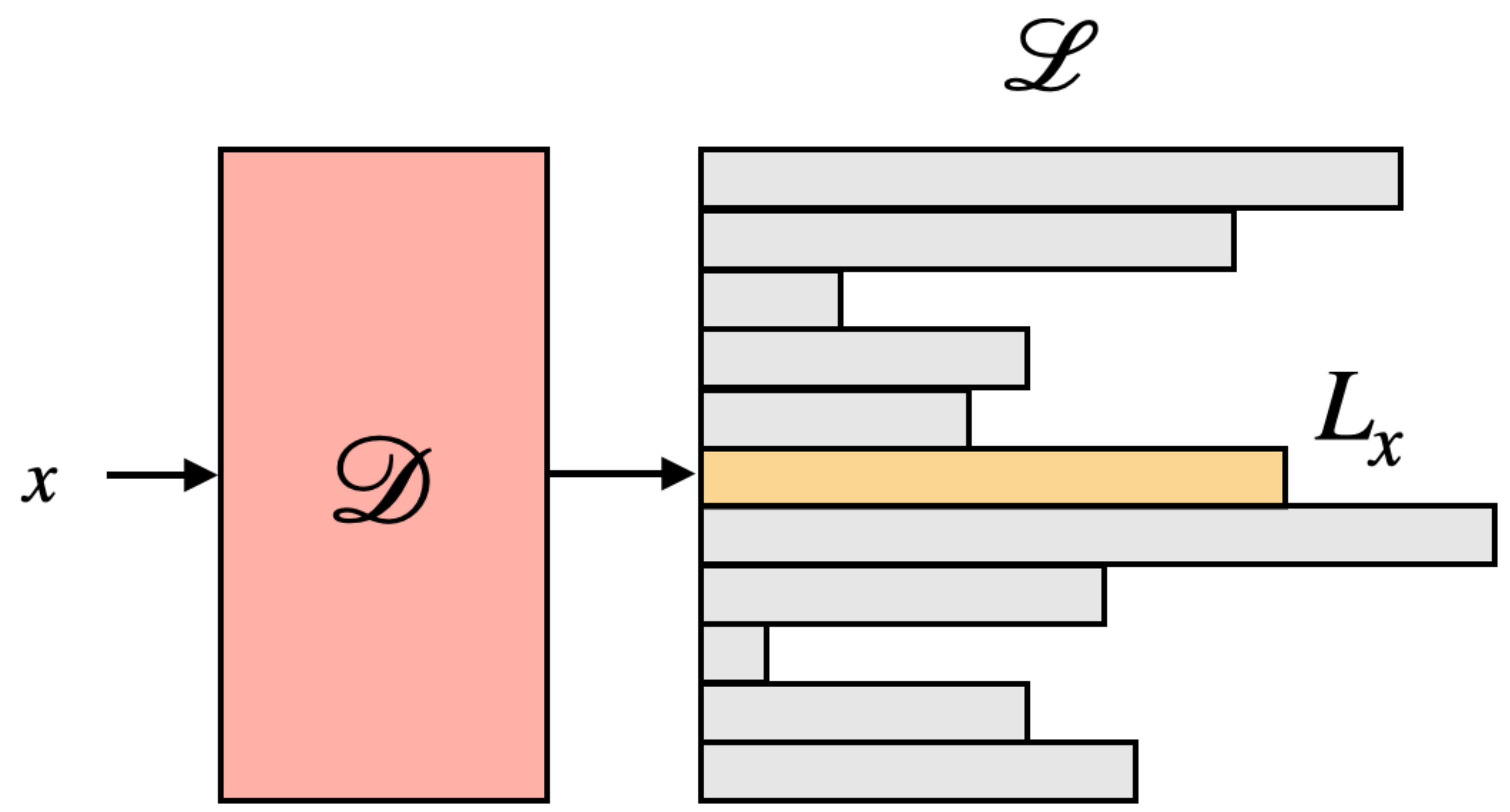
- What we want for a k-mer x is the map:

$$x \rightarrow L_x = \{(i, \{p_{ij}\}) \,|\, x \in R_i\},$$

where p_{ij} := position in R_i of the j-th k-mer of R_i, with $1 \leq p_{ij} \leq |R_i|$ and $1 \leq i \leq m$. The collection of all L_x is the **inverted index** $\mathscr{L}$.



Our setting



- All the distinct k-mers in $\mathscr{R} = \{R_1, \ldots, R_m\}$ are the **dictionary** $\mathscr{D}$.

- What we want for a k-mer x is the map:

  $$x \to L_x = \{(i, \{p_{ij}\}) \,|\, x \in R_i\},$$

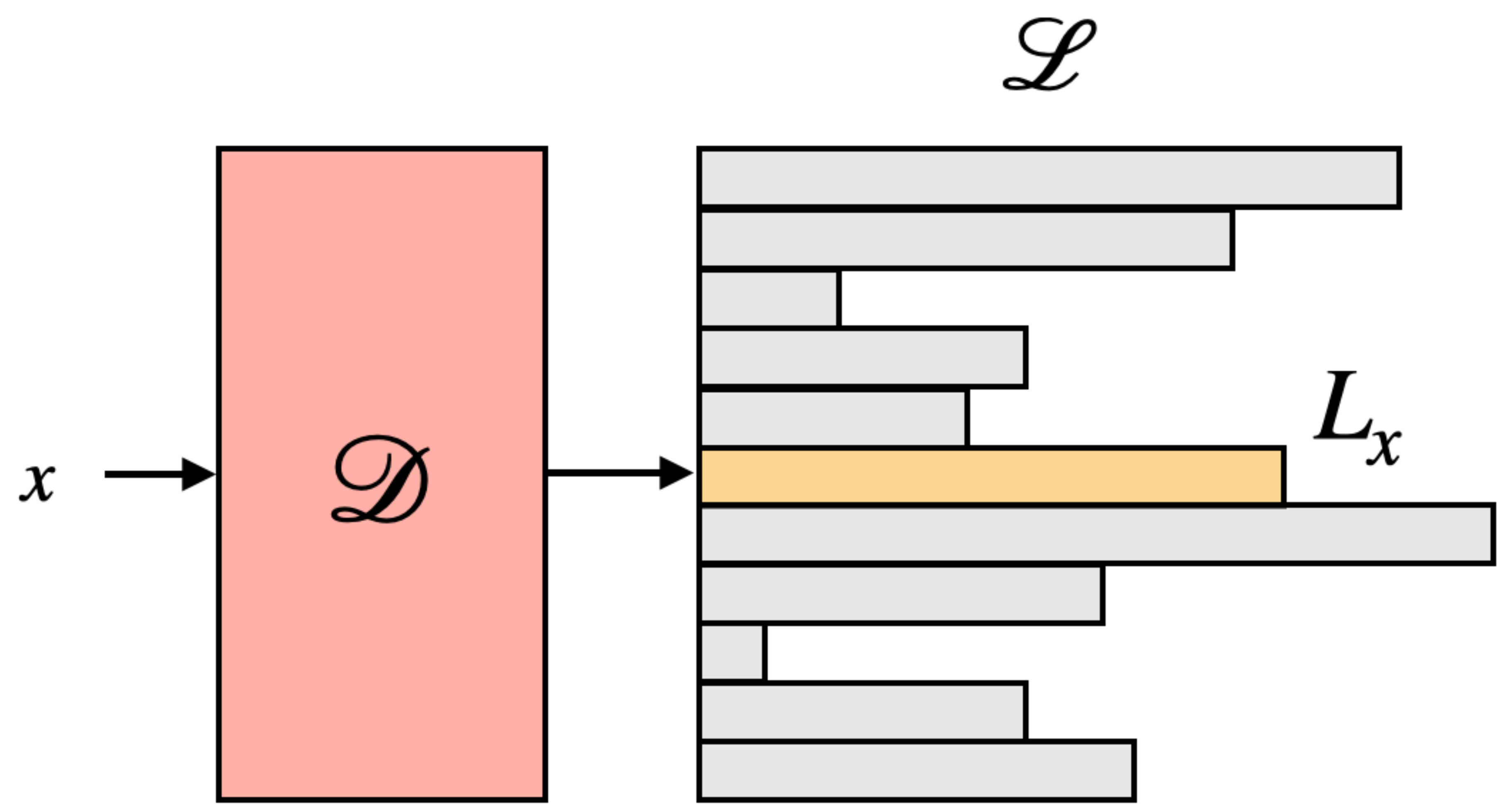
  where $p_{ij} :=$ position in R_i of the j-th k-mer of R_i, with $1 \le p_{ij} \le |R_i|$ and $1 \le i \le m$. The collection of all L_x is the **inverted index** $\mathscr{L}$.

- Queries:
  - **Membership**: does x appear in $\mathscr{R}$? Use the dictionary $\mathscr{D}$.
  - **Count**: if so, how many times? The length of L_x.
  - **Color**: and in what references? The set $\{i \,|\, x \in R_i\}$.
  - **Locate**: and at what positions in the references? The set $\{(i, \{p_{ij}\}) \,|\, x \in R_i\}$.

One index to rule them all

- Many k-mer based indexes (all of them?) are incarnations/adaptations of this **general indexing framework**, $\mathscr{D} + \mathscr{L}$:

  - BIGSI [Bradley et al. 2017]
  - Rainbowfish [Almodaresi et al. 2017]
  - Mantis [Pandey et al. 2018]
  - Pufferfish [Almodaresi et al. 2018]
  - COBS [Bingmann et al. 2019]
  - Reindeer [Marchet et al. 2020]
  - Raptor [Seiler et al. 2021]
  - Metagraph [Karasikov et al. 2022]
  - NIQKI [Agret et al. 2022]
  - Pufferfish2 [Fan et al. 2022]
  - etc.



Data structures based on k-mers for querying large collections of sequencing data sets
C Marchet, C Boucher, SJ Puglisi, P Medvedev, M Salson, R Chikhi
Genome Research

What is special about k-mers?

1. Since we take *all* the distinct k-mers (i.e., consecutive) from our references, **they share $(k-1)$-symbol overlaps**.

```
ACGGTAGAACCGATTCAAATTCGACGTAGC...
ACGGTAGAACCGA
 CGGTAGAACCGAT
  GGTAGAACCGATT
   GTAGAACCGATTC
    TAGAACCGATTCA
     AGAACCGATTCAA
      GAACCGATTCAAA
       AACCGATTCAAAT
        ...
```

Example for $k = 13$.

What is special about k-mers?

```
ACGGTAGAACCGATTCAAATTCGACGTAGC...
ACGGTAGAACCGA
 CGGTAGAACCGAT
  GGTAGAACCGATT
   GTAGAACCGATTC
    TAGAACCGATTCA
     AGAACCGATTCAA
      GAACCGATTCAAA
       AACCGATTCAAAT
        ...
```

← Example for $k = 13$.

1. Since we take *all* the distinct k-mers (i.e., consecutive) from our references, **they share $(k-1)$-symbol overlaps**.

- It is very likely that, given x in a query sequence Q and its answer returned from the index, $next(x)$ has a very similar answer (if not identical) $\rightarrow$ **Compression for satellite data.** Examples: presence/absence, abundance, color, *contig* identifier in a *de Bruijn* graph, etc.
- **Faster query time**: given the answer to x, the answer to $next(x)$ can be computed faster than the answer for another arbitrary k-mer $y \neq next(x)$.

What is special about k-mers?

1. Since we take *all* the distinct k-mers (i.e., consecutive) from our references, **they share $(k-1)$-symbol overlaps**.

```
ACGGTAGAACCGATTCAAATTCGACGTAGC...
ACGGTAGAACCGA
 CGGTAGAACCGAT
  GGTAGAACCGATT
   GTAGAACCGATTC
    TAGAACCGATTCA
     AGAACCGATTCAA
      GAACCGATTCAAA
       AACCGATTCAAAT
        ...
```

$\longleftarrow$ Example for $k = 13$.

- It is very likely that, given x in a query sequence Q and its answer returned from the index, $next(x)$ has a very similar answer (if not identical) $\rightarrow$ **Compression for satellite data.** Examples: presence/absence, abundance, color, *contig* identifier in a *de Bruijn* graph, etc.
- **Faster query time**: given the answer to x, the answer to $next(x)$ can be computed faster than the answer for another arbitrary k-mer $y \neq next(x)$.

Examples:

- findere/fimpera [Robidou and Peterlongo, 2021/22]
- SSHash — “Sparse and skew hashing of k-mers” [P., 2022]
- **NEW:** LPHash — “Locality-preserving minimal perfect hashing of k-mers” [P., Shibuya, Limasset 2022]

What is special about k-mers?

1. Since we take *all* the distinct k-mers (i.e., consecutive) from our references, **they share $(k-1)$-symbol overlaps**.

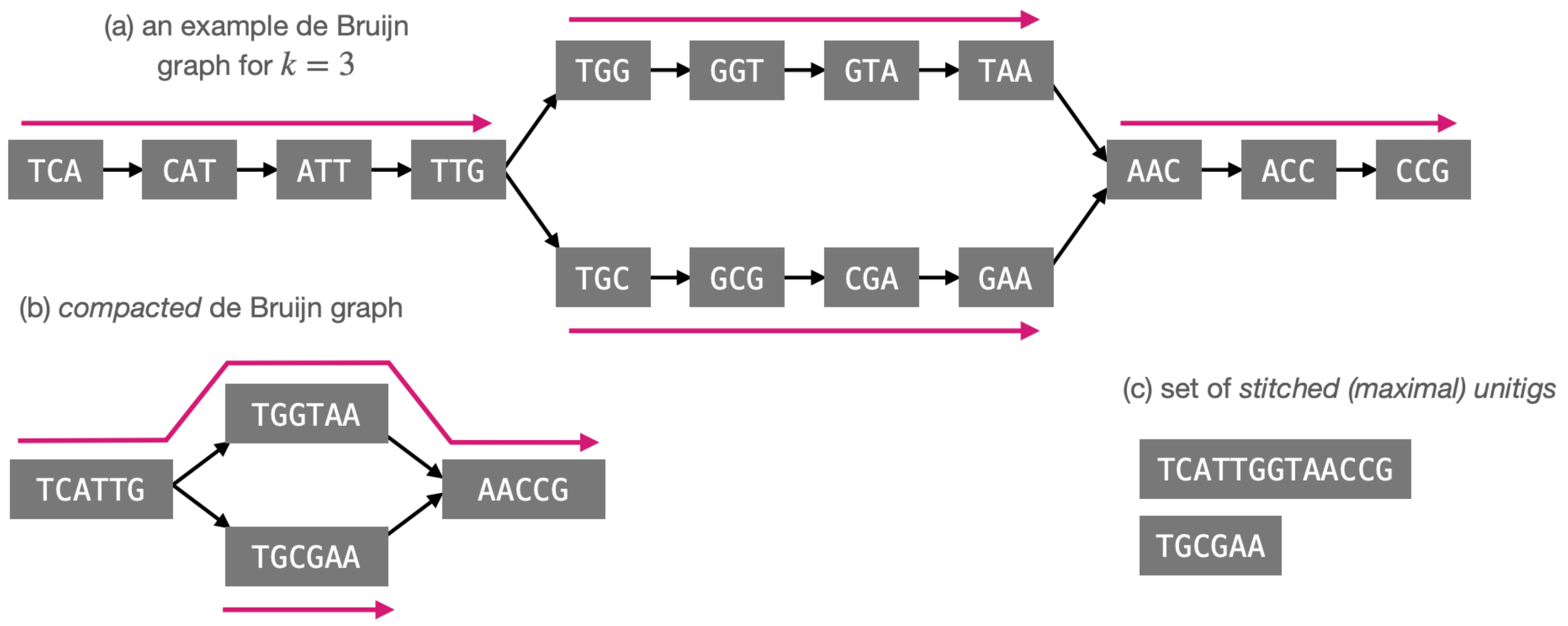
2. There are **many of them in each single reference** — k-mers encode millions of years of evolution vs. words in natural languages that have evolved over just “hundreds” of years.

   We will come back to this property later…

→ Let’s focus now the impact of **Property 1** on the two abstract data types — the dictionary $\mathscr{D}$ and the inverted index $\mathscr{L}$.

The dictionary

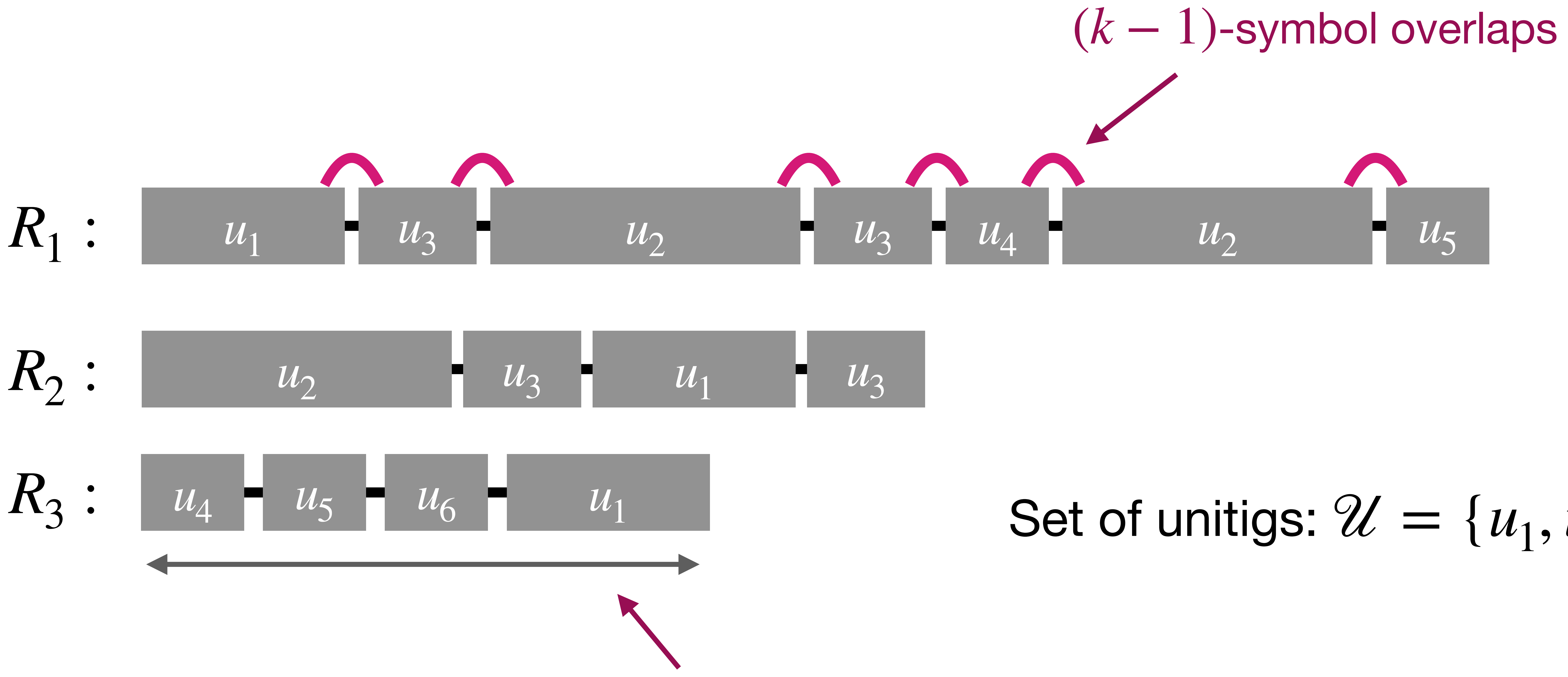
- **Property.** The dictionary $\mathscr{D}$ is a set of k-mers with $(k-1)$-symbol overlaps.
- One-to-one correspondence between $\mathscr{D}$ and a *de Bruijn* graph (dBG).



(a) an example de Bruijn graph for $k = 3$
TCA
CAT
ATT
TTG
TGG
GGT
GTA
TAA
TGC
GCG
CGA
GAA
AAC
ACC
CCG
(b) *compacted* de Bruijn graph
TCATTG
TGGTAA
TGCGAA
AACCG
(c) set of *stitched (maximal) unitigs*
TCATTGGTAACCG
TGCGAA

The dictionary

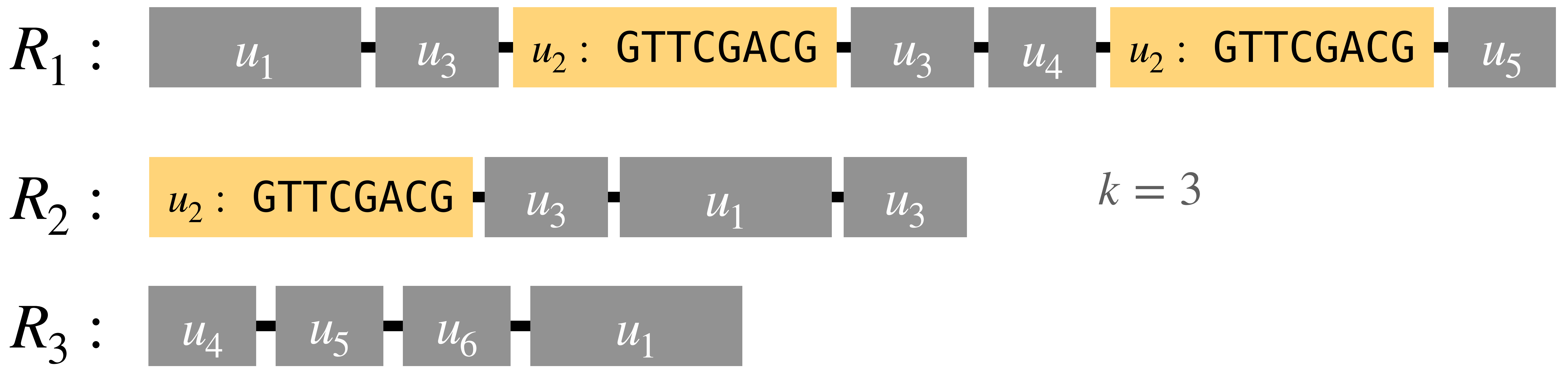
- From the references $\mathscr{R} = \{R_1, \ldots, R_m\}$ we build a ***reference* dBG**: each reference can be spelled by a **tiling of the unitigs** in the graph.
- (Each k-mer appears *once*, in a certain unitig.)



Set of unitigs: $\mathscr{U} = \{u_1, u_2, u_3, u_4, u_5, u_6\}$.

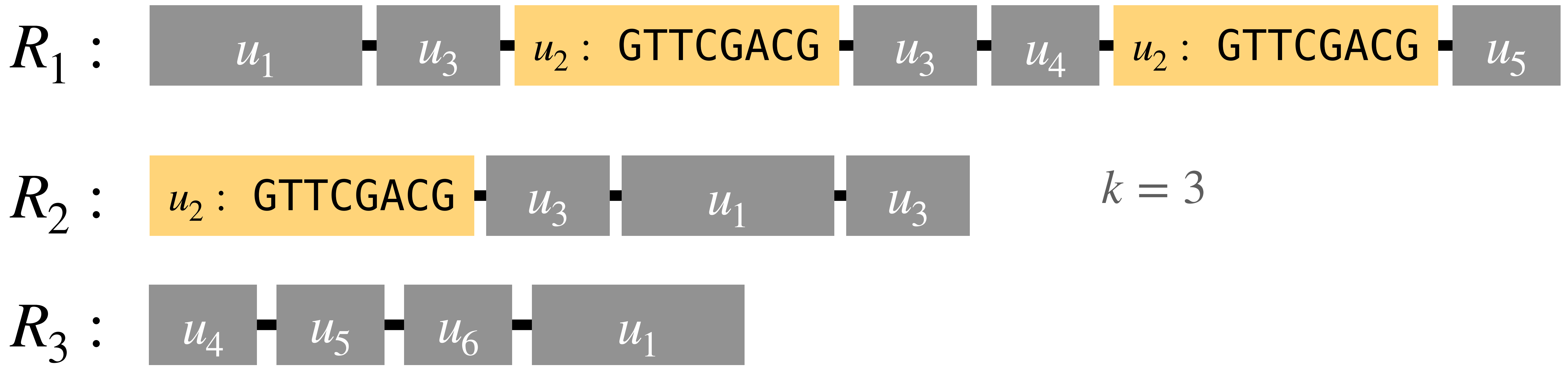
tiling of unitigs: $u_4 \rightarrow u_5 \rightarrow u_6 \rightarrow u_1$

The inverted index



- **Q.** How are the inverted lists of the k-mers in the same unitig, say, u_2?

The inverted index



$k = 3$

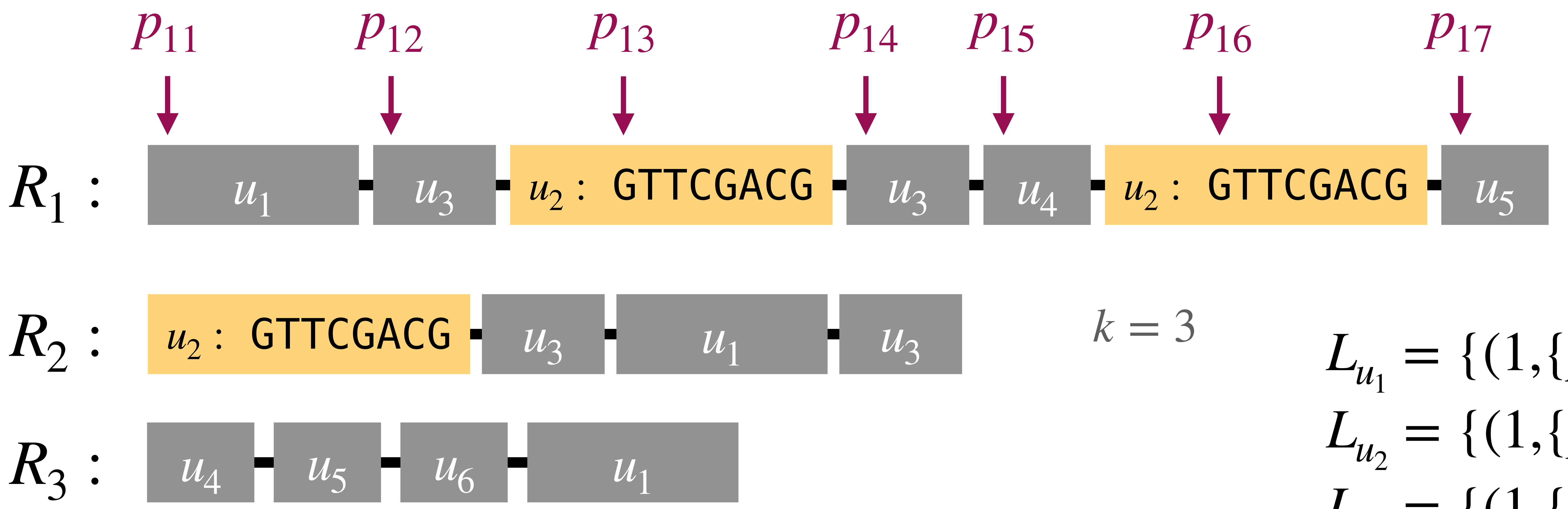
- **Q.** How are the inverted lists of the k-mers in the same unitig, say, u_2?
- **Property.** By construction of reference tilings, the inverted lists of the k-mers in the same unitig are **identical**.
- So instead of keeping a separate inverted list for each k-mer in $\mathscr{D}$, we store inverted lists at the unitig level: **much fewer lists.**

The inverted index

- Our map was $x \to L_x = \{(i, \{p_{ij}\}) \,|\, x \in R_i\}$ but now it looks like this:

  $x \to unitig(x) \to L_{unitig(x)} = \{(i, \{p_{ij}\}) \,|\, unitig(x) \in R_i\}$,

  where p_{ij} is now the position in R_i of the j-th unitig of R_i.



$k = 3$

$$L_{u_1} = \{(1,\{p_{11}\})(2,\{p_{23}\})(3,\{p_{34}\})\}$$
$$L_{u_2} = \{(1,\{p_{13}, p_{16}\})(2,\{p_{21}\})\}$$
$$L_{u_3} = \{(1,\{p_{12}, p_{14}\})(2,\{p_{22}, p_{24}\})\}$$
$$\ldots$$

Bringing the two together

- Our map is $x \rightarrow unitig(x) \rightarrow L_{unitig(x)} = \{(i, \{p_{ij}\}) \,|\, unitig(x) \in R_i\}$.
- But we need the **position of the k-mer** in the reference (not that of the unitig)!

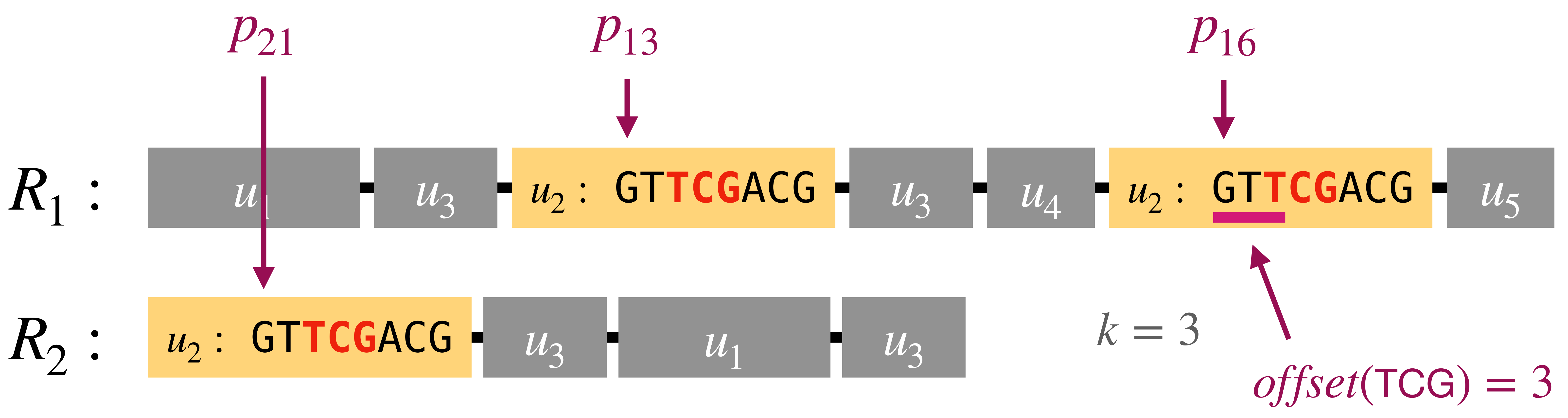
Bringing the two together

- Our map is $x \rightarrow unitig(x) \rightarrow L_{unitig(x)} = \{(i, \{p_{ij}\}) \,|\, unitig(x) \in R_i\}$.
- But we need the **position of the k-mer** in the reference (not that of the unitig)!

- **Solution.** Use the dictionary $\mathscr{D}$ to compute the "local" position of the k-mer x in $unitig(x)$ **on-the-fly**. Let's call it $offset(x)$.
- We need $\mathscr{D}$ to implement the map: $x \rightarrow (unitig(x), offset(x))$.
- Lastly, the positions of x in $\mathscr{R}$ are computed as: $\{(i, \{offset(x) + p_{ij} - 1\}) \,|\, unitig(x) \in R_i\}$.

Bringing the two together

- Our map is $x \rightarrow unitig(x) \rightarrow L_{unitig(x)} = \{(i, \{p_{ij}\}) \,|\, unitig(x) \in R_i\}$.
- But we need the **position of the k-mer** in the reference (not that of the unitig)!
- **Solution.** Use the dictionary $\mathscr{D}$ to compute the "local" position of the k-mer x in $unitig(x)$ **on-the-fly**. Let's call it $offset(x)$.
- We need $\mathscr{D}$ to implement the map: $x \rightarrow (unitig(x), offset(x))$.
- Lastly, the positions of x in $\mathscr{R}$ are computed as: $\{(i, \{offset(x) + p_{ij} - 1\}) \,|\, unitig(x) \in R_i\}$.



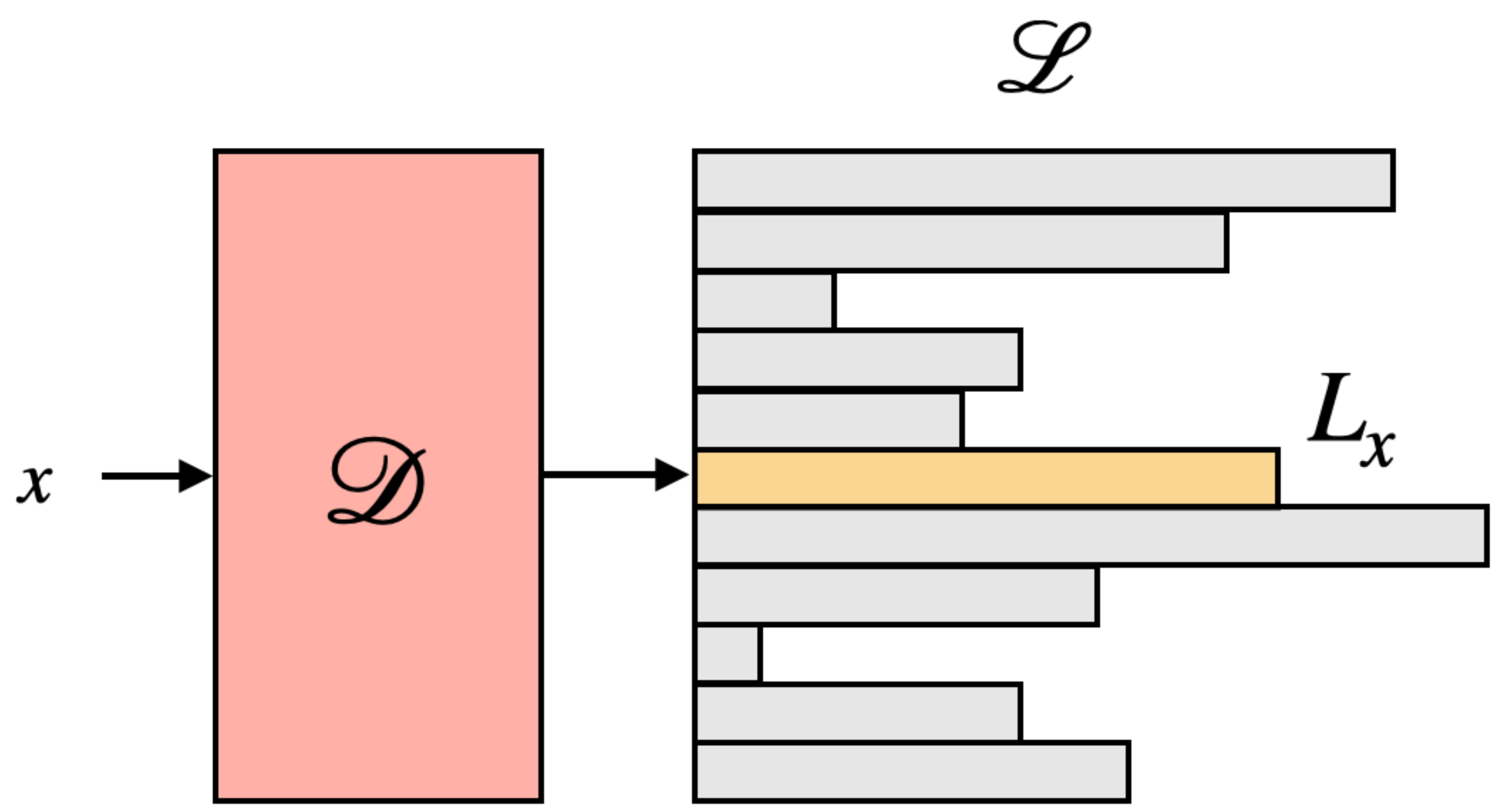
- $x =$ **TCG**
- The result is:
  $\{(1,\{3 + p_{13} - 1,\ 3 + p_{16} - 1\}),\ (2,\{3 + p_{21} - 1\})\}$

Modular reference indexing

- We have therefore decomposed our problem into **two distinct mappings** with **simple** APIs.

  1. From k-mer to unitig.

$$x \xrightarrow{\mathscr{D}} (unitig(x), offset(x))$$



  2. From unitig to inverted list.

$$unitig(x) \xrightarrow{\mathscr{L}} L_{unitig(x)} = \{(i, \{p_{ij}\}) \,|\, unitig(x) \in R_i\}$$

- Depending on the application at hand:
  + query layer (e.g., to support streaming queries)
  + output layer for displaying results

Modular reference indexing

- We have overviewed a general modular indexing framework with two abstract data types — a *dictionary* $\mathscr{D}$ and an *inverted index* $\mathscr{L}$.
- We have described the (minimal) API.
- **Take-away**: any algorithmic effort spent on the reference indexing problem should be devoted to the improvement of $\mathscr{D}$ and/or $\mathscr{L}$.
- So let's now consider:
  - what **data structures** can be used for $\mathscr{D}$ and $\mathscr{L}$;
  - what are (some of) the **open challenges/questions**.

The dictionary data structure

- From k-mer to unitig: $x \xrightarrow{\mathscr{D}} (unitig(x), offset(x))$
- **Q.** Any solutions?

The dictionary data structure

- From k-mer to unitig: $x \xrightarrow{\mathscr{D}} (unitig(x), offset(x))$
- **Q.** Any solutions?
- **A. SSHash** — Sparse and Skew Hashing of k-mers [P., 2022]
  - Fast and compact (builds on minimal perfect hashing and minimizers)
  - Exact, associative, weighted
  - Support for point/streaming/navigational queries
  - Scale to large datasets using external memory
  - **Order-preserving**: *consecutive k-mers in the unitigs get consecutive hash codes* (that's how we implement the mapping)
- SSHash can index *any* spectrum-preserving string set (SPSS). In this case, we are interested in the unitigs of the compacted reference dBG. We can use **Cuttlefish2** [Khan et al. 2022] to compute the unitigs. (Excellent speed and scalability.)

The dictionary data structure — Drop in?

- **Q.** BWT-based indexes (e.g., FM-index, BOSS, SBWT, etc.)?

  A. Not immediately clear.

  Would probably need an extra level of indirection to implement the mapping because k-mers are *sorted lexicographically, not by their order of appearance in the unitigs*.

  This indirection could outweigh the space savings offered by the BWT.

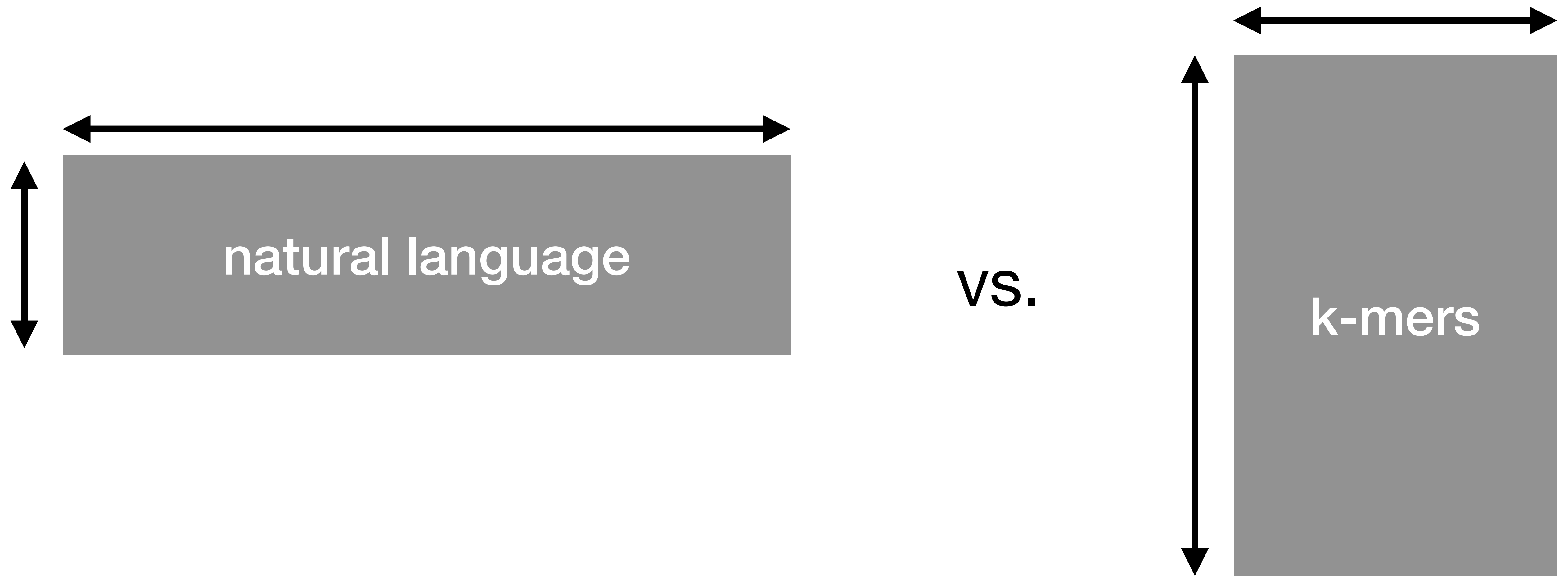
The inverted index data structure

- From unitig to inverted list: $unitig(x) \xrightarrow{\mathscr{L}} L_{unitig(x)} = \{(i, \{p_{ij}\}) \,|\, unitig(x) \in R_i\}$
- Lists are **sorted** (for example, first by reference identifier i, then by positions) and compressed.
- Plethora of compression techniques developed in IR with different space/time trade-offs.
- See for example:
  - "Techniques for Inverted Index Compression"
    [P. and Venturini, ACM Computing Surveys, 2021]
  - Crash course on data compression:
    https://github.com/jermp/data_compression_course

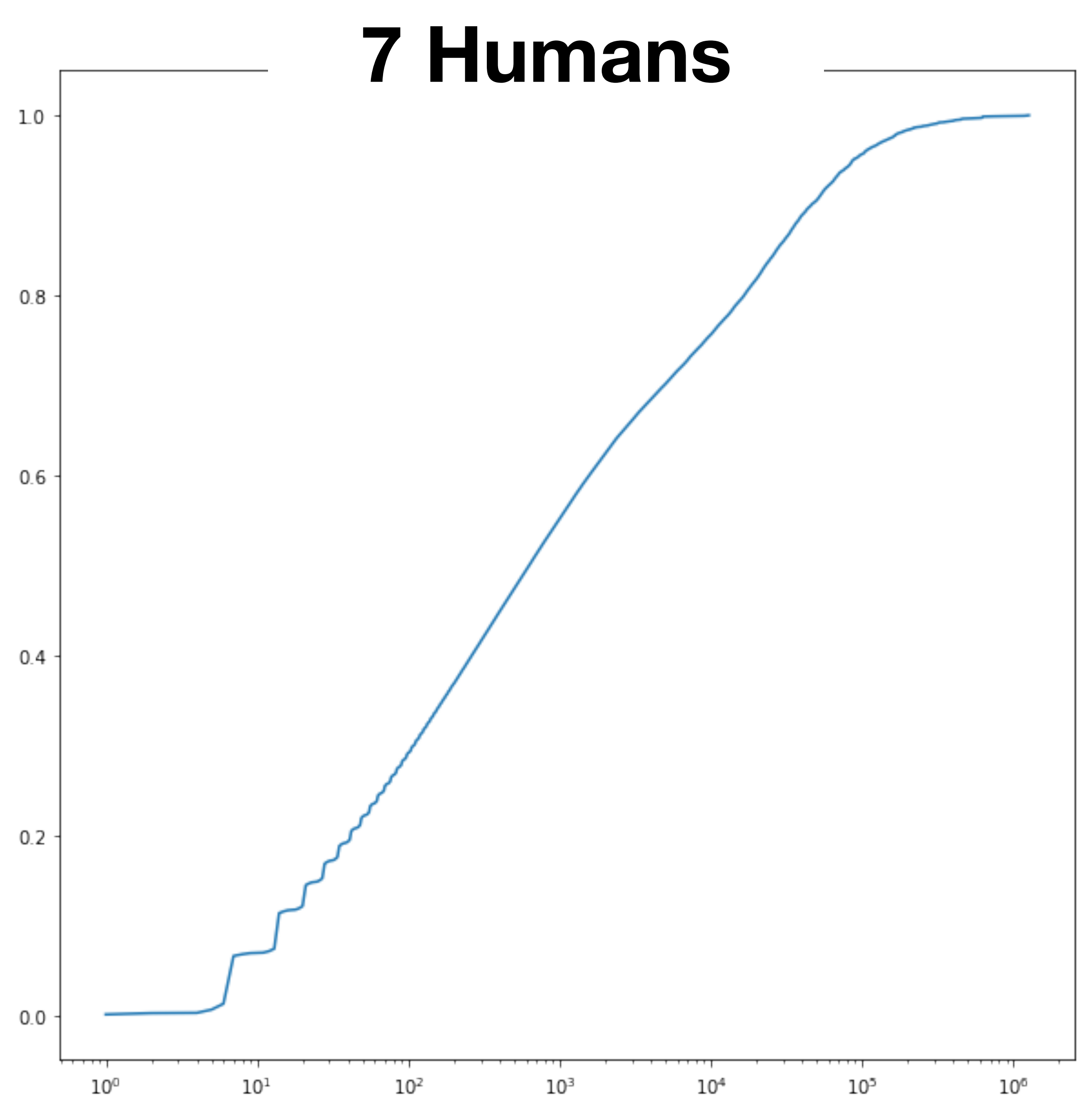
What is special about k-mers? — Reprise

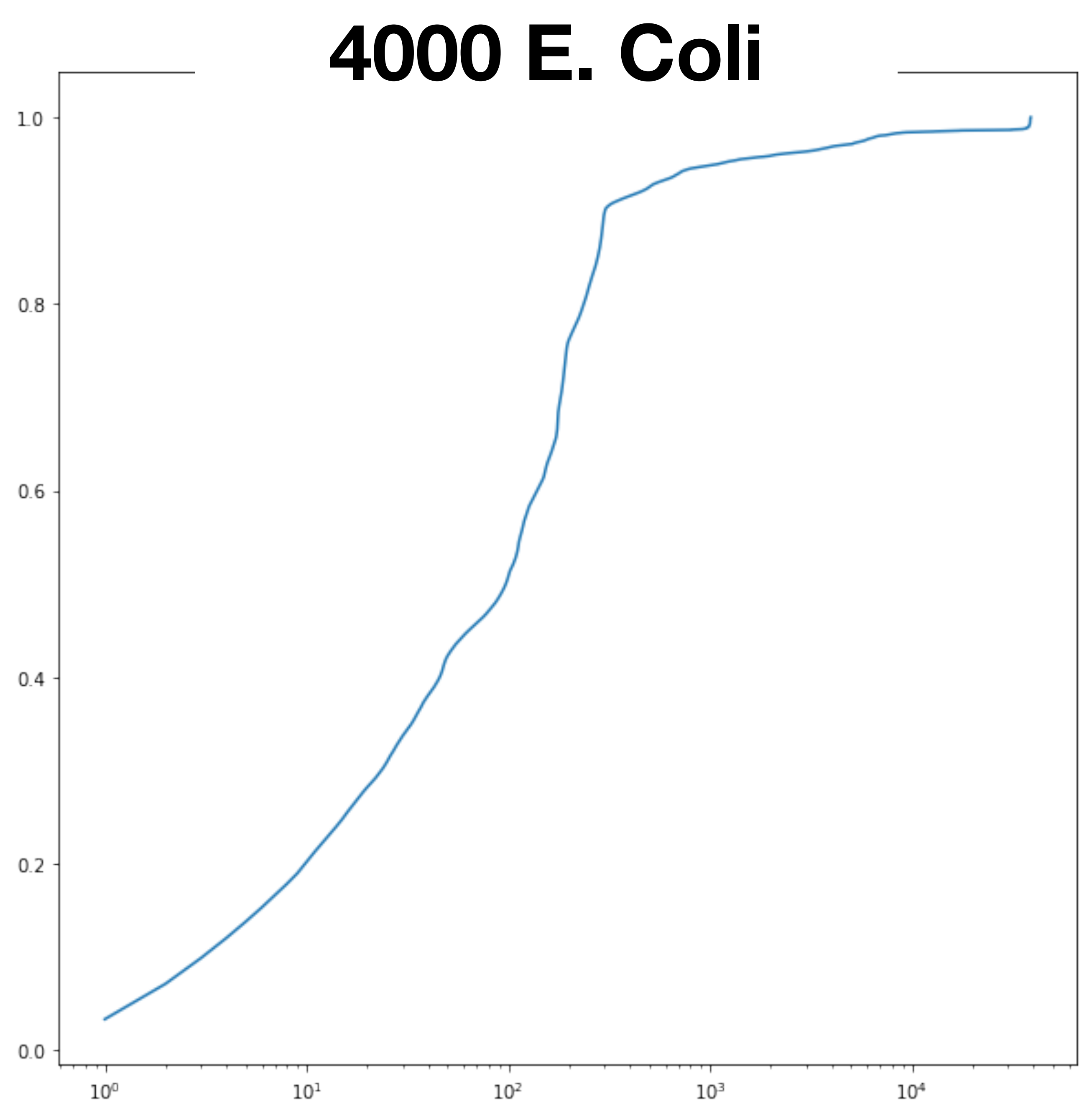
2. There are **many of them in each single reference** — k-mers encode millions of years of evolution vs. words in natural languages that have evolved over just “hundreds” of years.

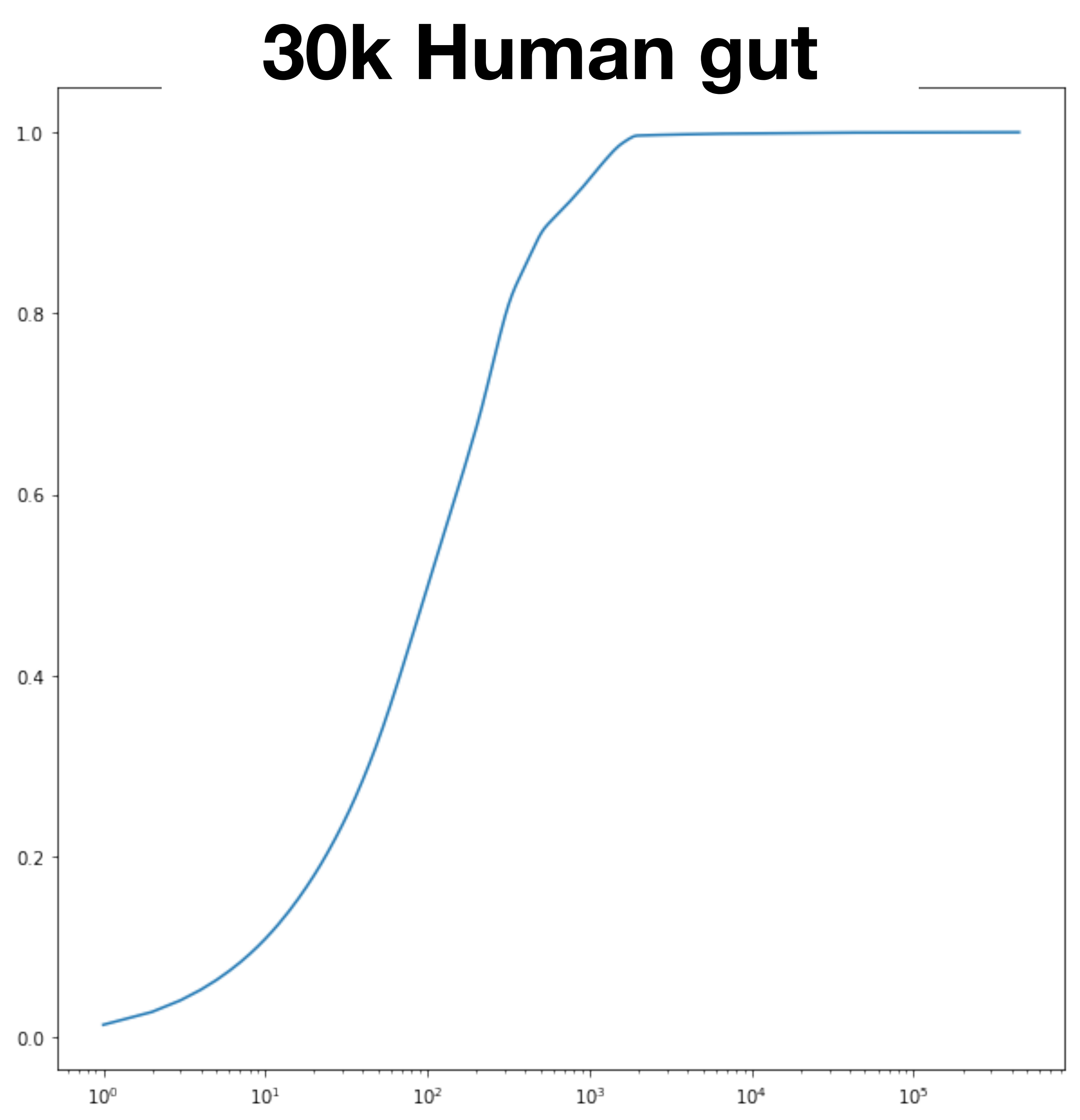
   Natural language: many many documents, but “relatively few” terms;
   DNA: relatively few documents, but many many terms.



Occurrence distribution



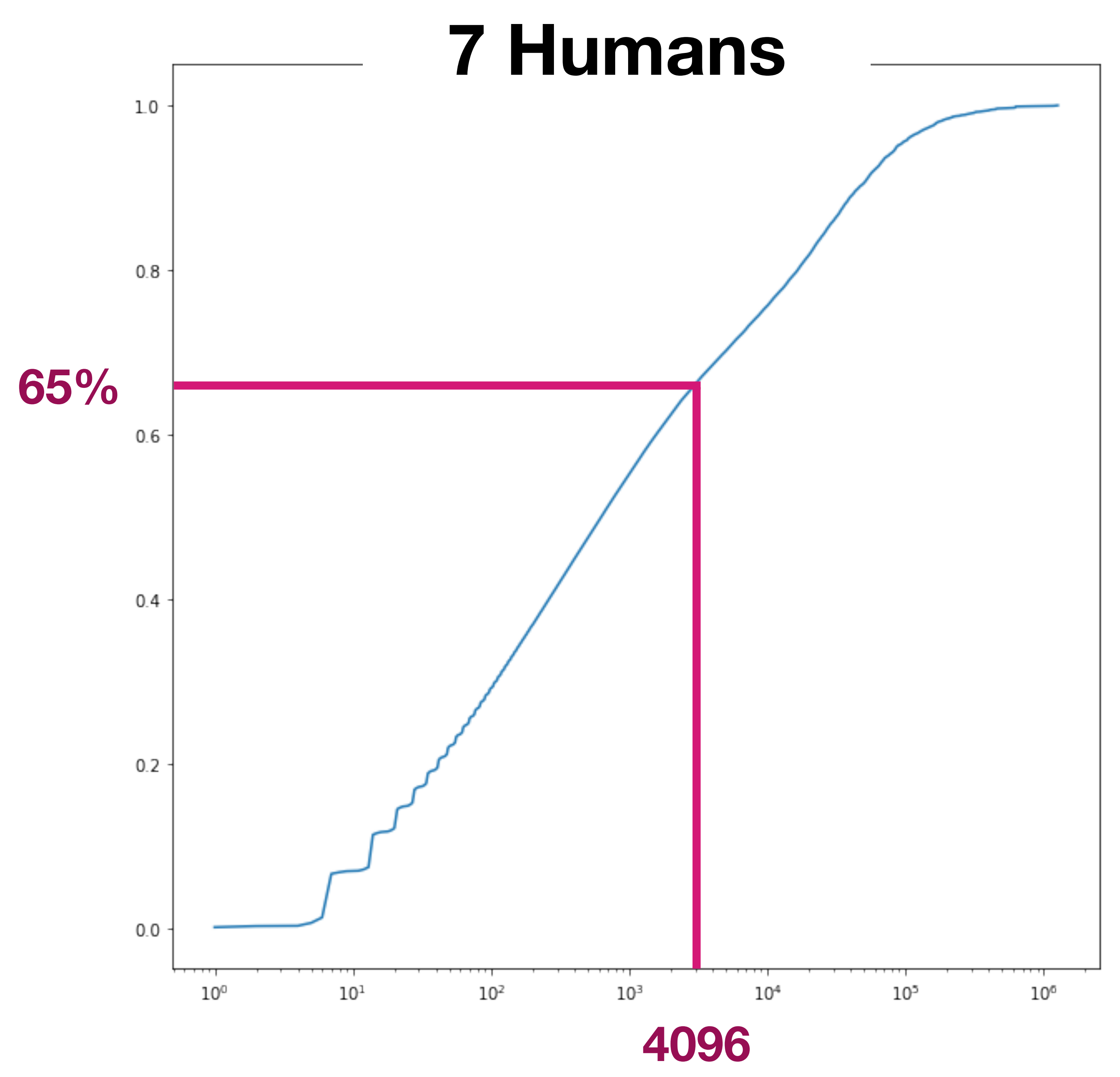


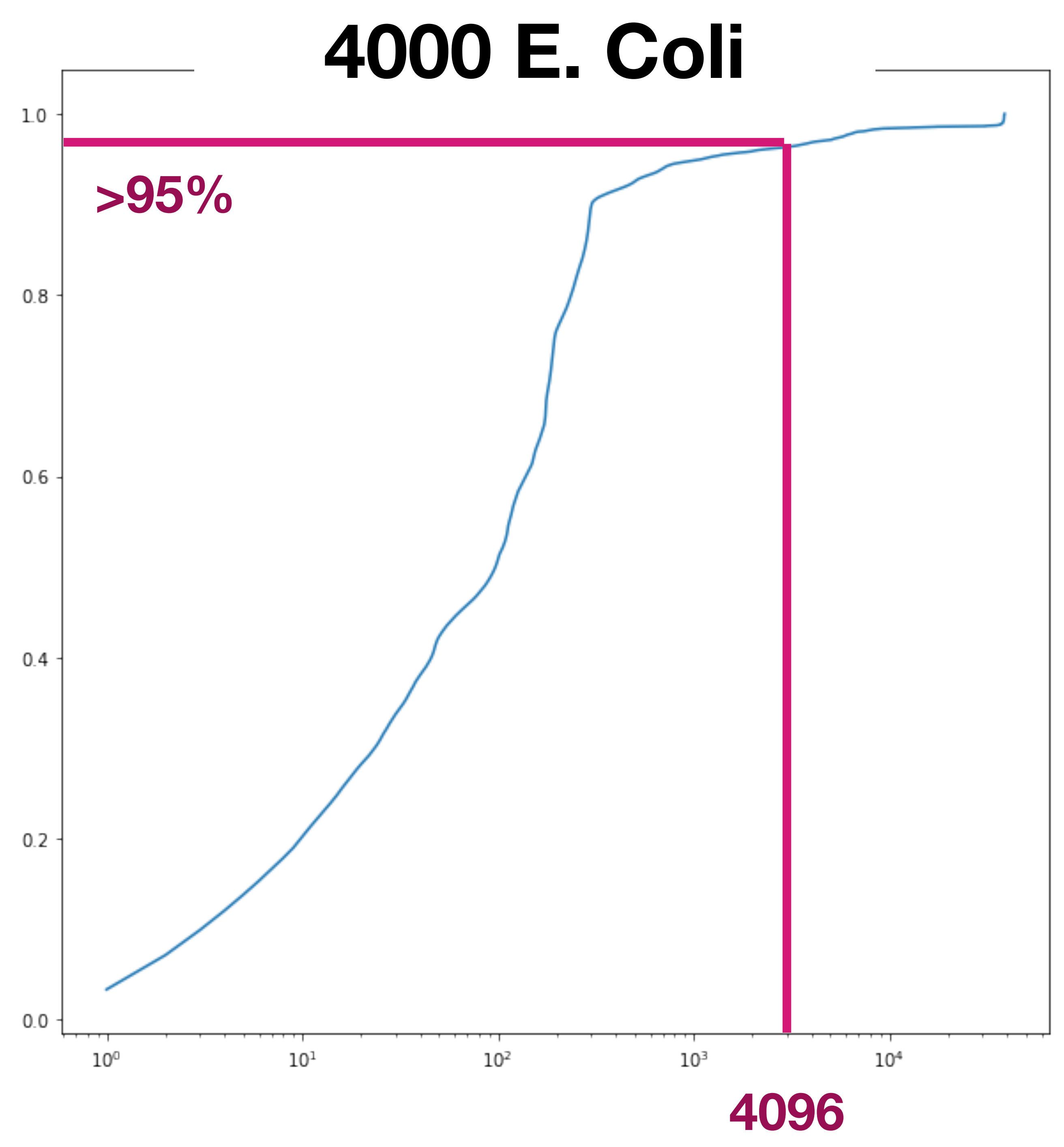


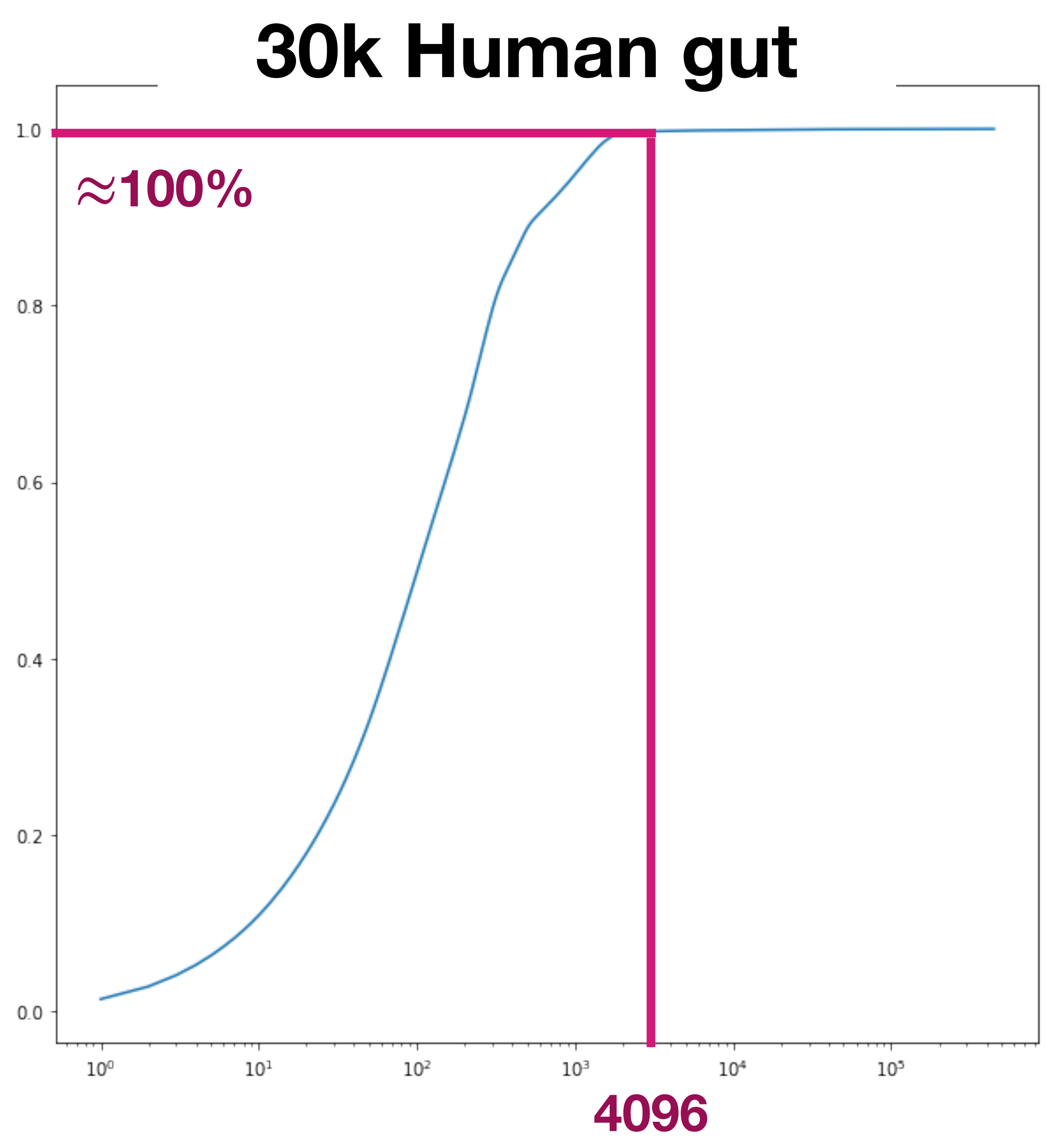
Data	# unitigs (10^6)	Total unitig occs (10^9)
7 humans	46.5	2.2
4000 E. Coli	91.7	0.9
30k human gut	560.7	11.2

- y-axis: cumulative % of occurrences in the inverted index for unitigs that appear < x times.

Occurrence distribution







Data	# unitigs (10^6)	Total unitig occs (10^9)
7 humans	46.5	2.2
4000 E. Coli	91.7	0.9
30k human gut	560.7	11.2

- y-axis: cumulative % of occurrences in the inverted index for unitigs that appear < x times.
- For comparison: on Web-page datasets — Gov2, ClueWeb09, CCNews — we retain 93%, 94%, 98% of the occurrences!

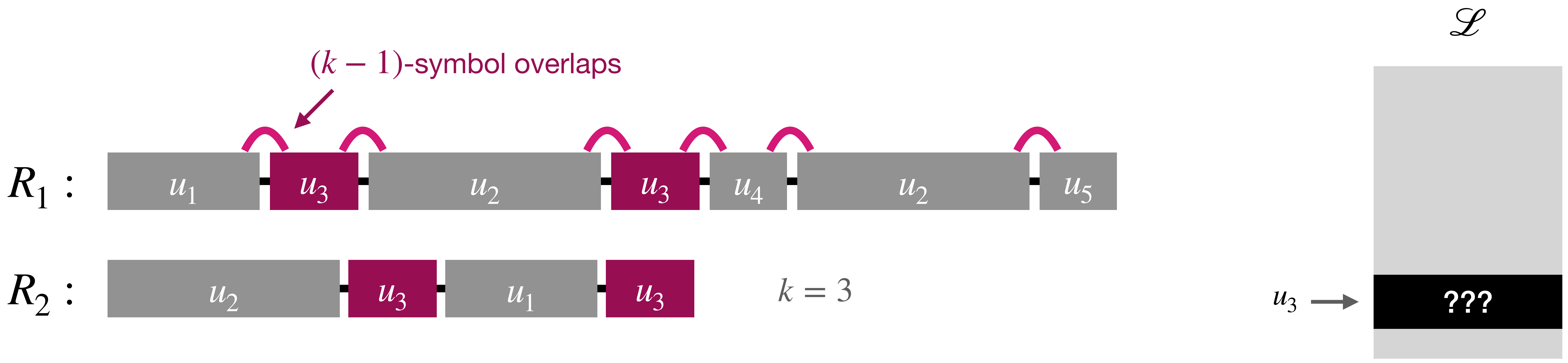
Sampling vs. Compression

- In [Fan, Khan, P., Patro 2022] we use a simple **sampling scheme** where we keep 1 unitig every s unitigs in the inverted index: if a unitig is not sampled, we do **not** store its occurrences.
- It is still possible to “re-construct” exactly its occurrences by walking back over the reference tilings.

Sampling vs. Compression

- In [Fan, Khan, P., Patro 2022] we use a simple **sampling scheme** where we keep 1 unitig every s unitigs in the inverted index: if a unitig is not sampled, we do **not** store its occurrences.
- It is still possible to "re-construct" exactly its occurrences by walking back over the reference tilings.

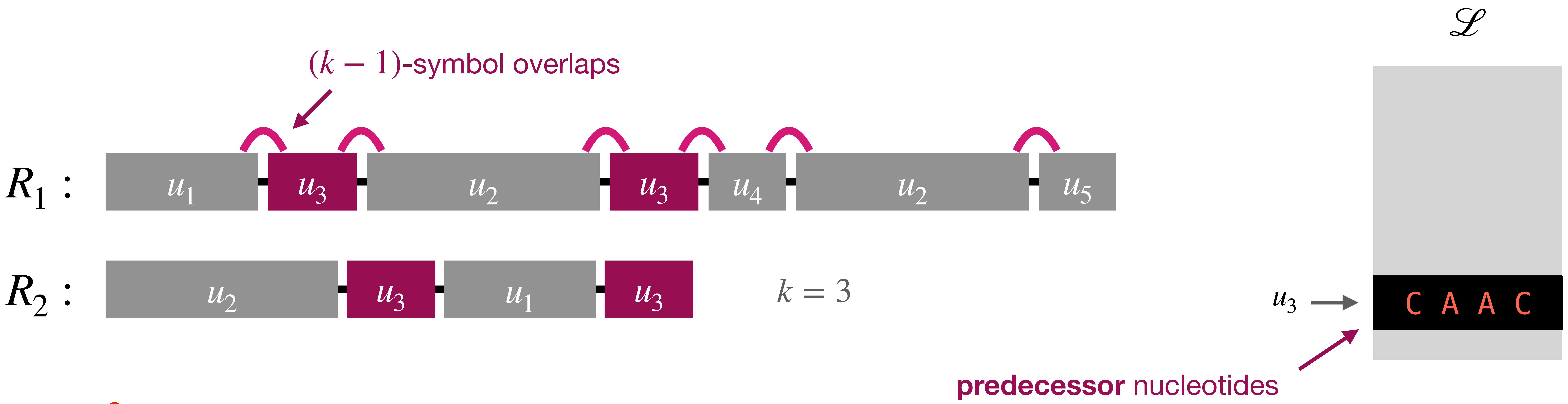
Suppose u_3 = **`CGGT`** is **not** sampled, but u_1 and u_2 are sampled.



Sampling vs. Compression

- In [Fan, Khan, P., Patro 2022] we use a simple **sampling scheme** where we keep 1 unitig every s unitigs in the inverted index: if a unitig is not sampled, we do **not** store its occurrences.
- It is still possible to "re-construct" exactly its occurrences by walking back over the reference tilings.

Suppose u_3 = **`CGGT`** is **not** sampled, but u_1 and u_2 are sampled.



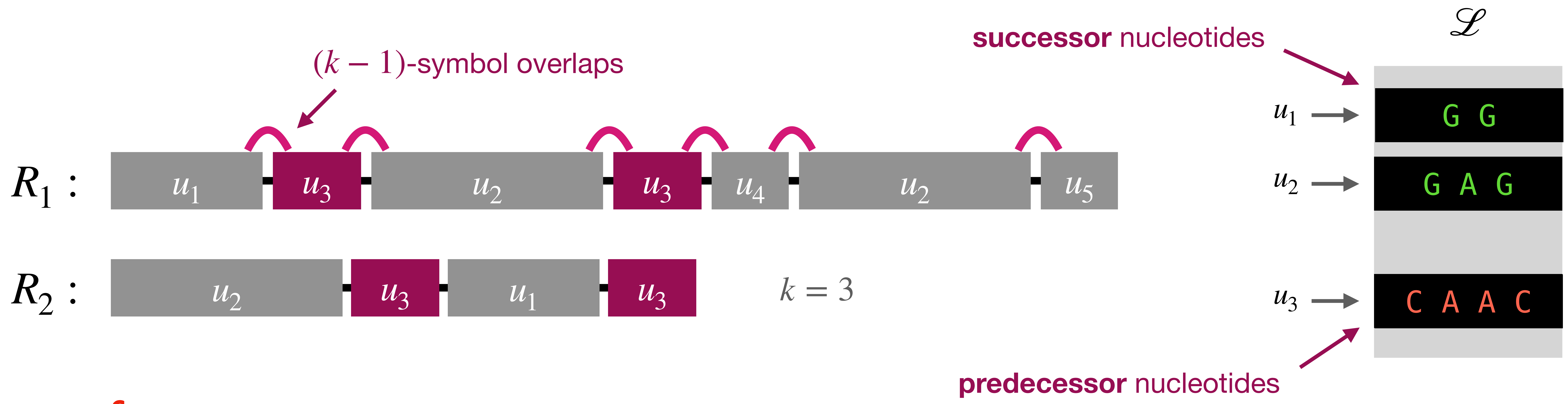
`C`/`A` **`CGGT`** = u_3 $\rightarrow$ query the **dictionary** for CCG and ACG:

$unitig$(CCG) $= u_1$ and $unitig$(ACG) $= u_2$.

Sampling vs. Compression

- In [Fan, Khan, P., Patro 2022] we use a simple **sampling scheme** where we keep 1 unitig every s unitigs in the inverted index: if a unitig is not sampled, we do **not** store its occurrences.
- It is still possible to "re-construct" exactly its occurrences by walking back over the reference tilings.

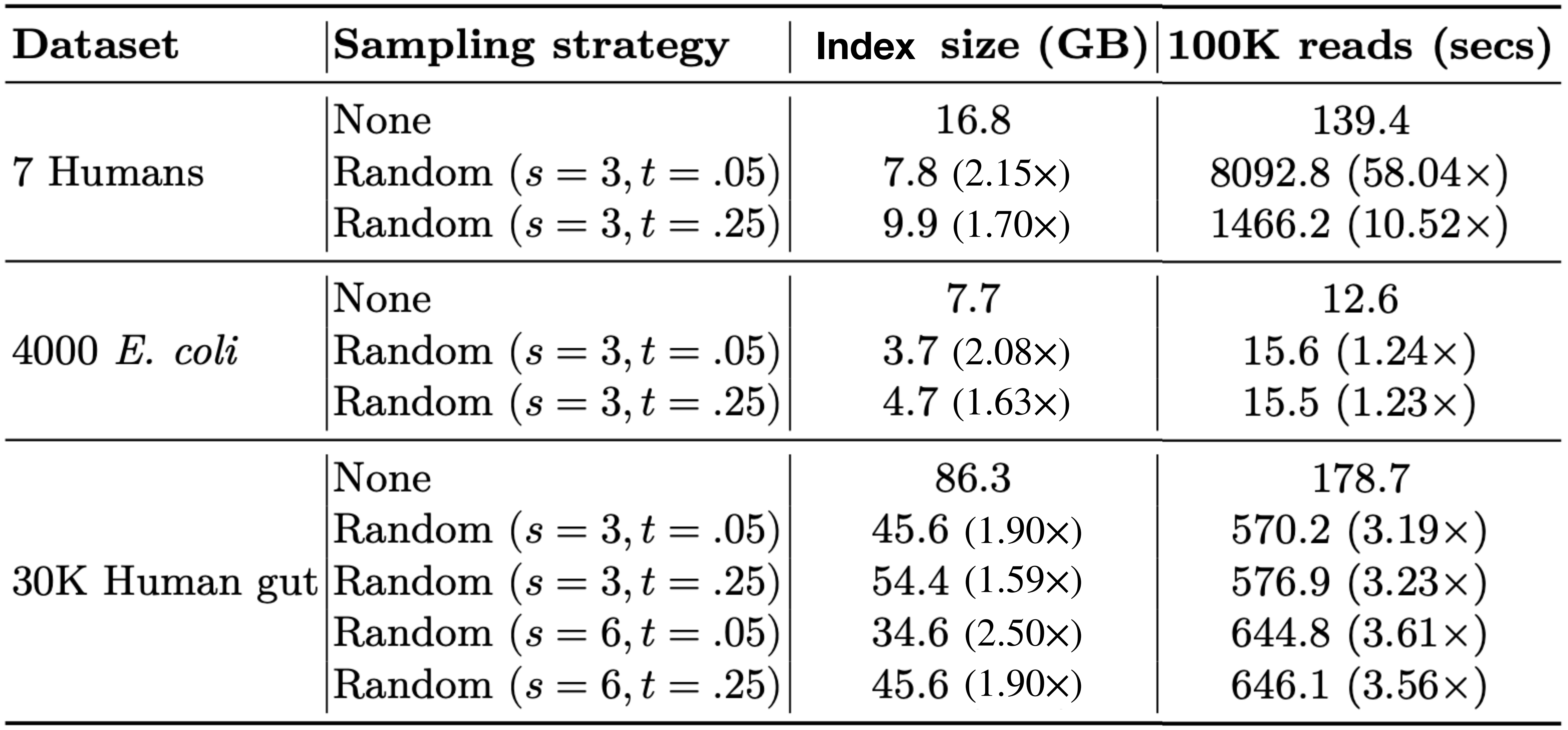
Suppose u_3 = **`CGGT`** is **not** sampled, but u_1 and u_2 are sampled.



`C`
`A` **`CGGT`** = u_3 → query the **dictionary** for CCG and ACG:

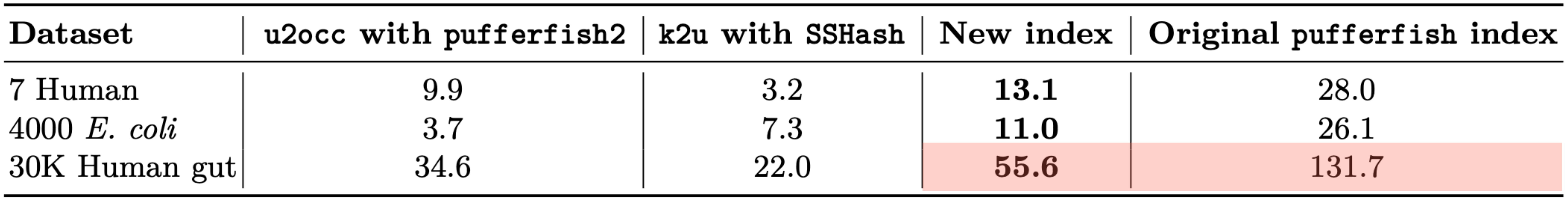
$unitig$(CCG) = u_1 and $unitig$(ACG) = u_2.

Sampling the inverted index



Dataset	Sampling strategy	Index size (GB)	100K reads (secs)
7 Humans	None	16.8	139.4
	Random ($s = 3, t = .05$)	7.8 (2.15×)	8092.8 (58.04×)
	Random ($s = 3, t = .25$)	9.9 (1.70×)	1466.2 (10.52×)
4000 *E. coli*	None	7.7	12.6
	Random ($s = 3, t = .05$)	3.7 (2.08×)	15.6 (1.24×)
	Random ($s = 3, t = .25$)	4.7 (1.63×)	15.5 (1.23×)
30K Human gut	None	86.3	178.7
	Random ($s = 3, t = .05$)	45.6 (1.90×)	570.2 (3.19×)
	Random ($s = 3, t = .25$)	54.4 (1.59×)	576.9 (3.23×)
	Random ($s = 6, t = .05$)	34.6 (2.50×)	644.8 (3.61×)
	Random ($s = 6, t = .25$)	45.6 (1.90×)	646.1 (3.56×)

Sampling the inverted index



Dataset	**u2occ with pufferfish2**	**k2u with SSHash**	**New index**	**Original pufferfish index**
7 Human	9.9	3.2	**13.1**	28.0
4000 *E. coli*	3.7	7.3	**11.0**	26.1
30K Human gut	34.6	22.0	**55.6**	131.7

Sampling the inverted index

Dataset	u2occ with pufferfish2	k2u with SSHash	New index	Original pufferfish index
7 Human	9.9	3.2	**13.1**	28.0
4000 *E. coli*	3.7	7.3	**11.0**	26.1
30K Human gut	34.6	22.0	**55.6**	131.7

AWS EC2 instances pricing:

- https://instances.vantage.sh/aws/ec2/x2gd.xlarge
  64 GiB of RAM — **243 USD** per month
- https://instances.vantage.sh/aws/ec2/x2gd.2xlarge
  128 GiB of RAM — **478 USD** per month
- https://instances.vantage.sh/aws/ec2/x2gd.4xlarge
  256 GiB of RAM — **975 USD** per month

Conclusions

- The *reference indexing problem* admits a modular solutions made up of two distinct abstract data types: a **dictionary** $\mathscr{D}$ and an **inverted index** $\mathscr{L}$.
- While substantial work has been done for $\mathscr{D}$, little work has been done for $\mathscr{L}$ (for DNA references).
- We have shown that, by exploiting k-mer *overlaps*, we can reduce the number of lists stored in $\mathscr{L}$.
- Reasoning in terms of **reference tilings** opens the possibility for **sampling** tiles' occurrences.
- Depending on how $\mathscr{L}$ is represented (e.g., lossless/lossy, sampled/compressed, …):
  a whole *class* of related reference indexing data structures can be obtained.

Challenges — Part 1

1. Improve the space/time/scalability trade-off of the dictionary.

2. How the compression techniques developed for IR (e.g., Interpolative Coding, Elias-Fano, DINT, PFor, etc.) fare in this case?

3. *Hybrid* strategies combining sampling with compression?

4. Investigate different sampling schemes. For example, instead of either sampling or not a unitig entirely (all its occurrences), we could sample the inverted lists directly, or the reference tilings.

5. What is the performance of more complex queries (for example, with query operators AND/OR, etc.) rather than just enumeration? This would improve the *expressiveness* of the index.

   For example: x AND y, where x, y are k-mers. Or *weighted-and*: return all documents where x AND y occur but *at least for t times*, for some user-defined $t > 0$.

Challenges — Part 2

6. What is the "best" set of tiles for the references $\mathscr{R} = \{R_1, \ldots, R_m\}$?

   We have used the unitigs, but it is also possible to use other "tigs" with some extra bookkeeping. However, here it is not clear if a smaller SPSS (longer "tigs") also implies a smaller reference tiling.

7. **What happens on pan-genomes?** If $\mathscr{R}$ is a pan-genome, we expect:
   - The dictionary not to grow that much by adding related genomes to $\mathscr{R}$;
   - The reference tilings to be *very similar*. This will also reflect on the inverted lists.

8. In this talk we have focused on *exact* (i.e., *lossless*) approaches: exact k-mer membership and exact colors/positions.

   What happens if we allow an *approximation* or *false-positives*?

   - Approximation: do not index all tiles.
   - False-positives: result of a query contains some "wrong" answers.

Thank you for the attention!

A special thank to
Jason Fan, **Jamshed Khan**, and **Rob Patro**
University of Maryland (USA)