Spectrum Preserving Tilings Enable Sparse and Modular Reference Indexing

Giulio Ermanno Pibiri
Ca' Foscari University of Venice

@giulio_pibiri
@jermp

Join work with **Jason Fan**, **Jamshed Khan**, and **Rob Patro**
University of Maryland (USA)

Data Structures in Bioinformatics (DSB)
Delft, Netherlands, 21 March 2023

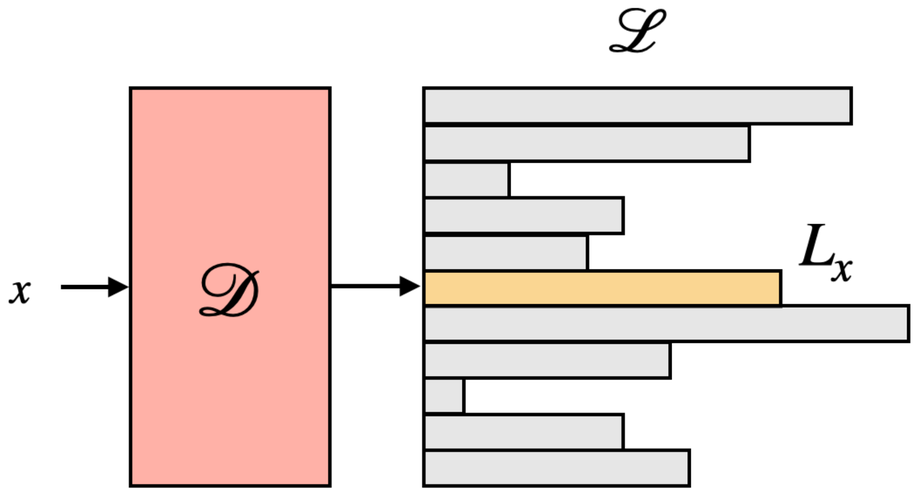
The reference indexing problem

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

Index overview



- All the distinct k-mers in $\mathscr{R} = \{R_1, \ldots, R_m\}$ are stored in a **dictionary** $\mathscr{D}$.
- What we want for a k-mer x is the map:

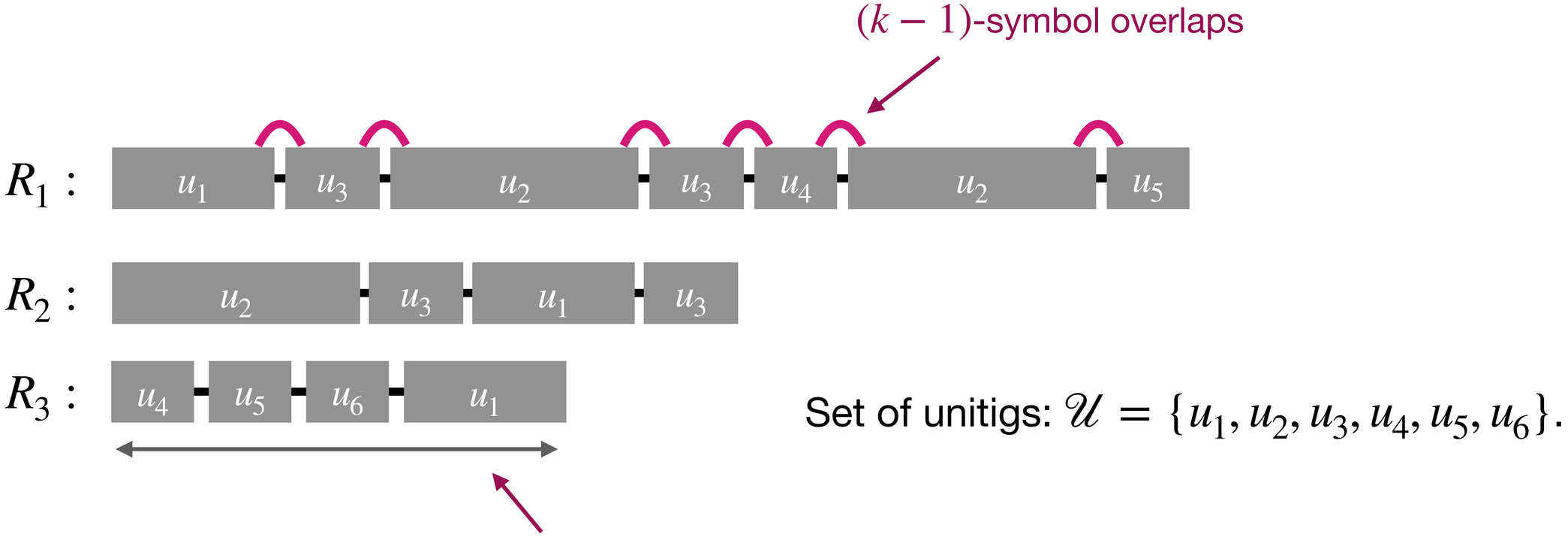
$$x \to L_x = \{(i, \{p_{ij}\}) \,|\, x \in R_i\},$$

where $p_{ij} :=$ position in R_i of the j-th k-mer of R_i. 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 $\{(i, \{p_{ij}\}) \,|\, x \in R_i\}$.

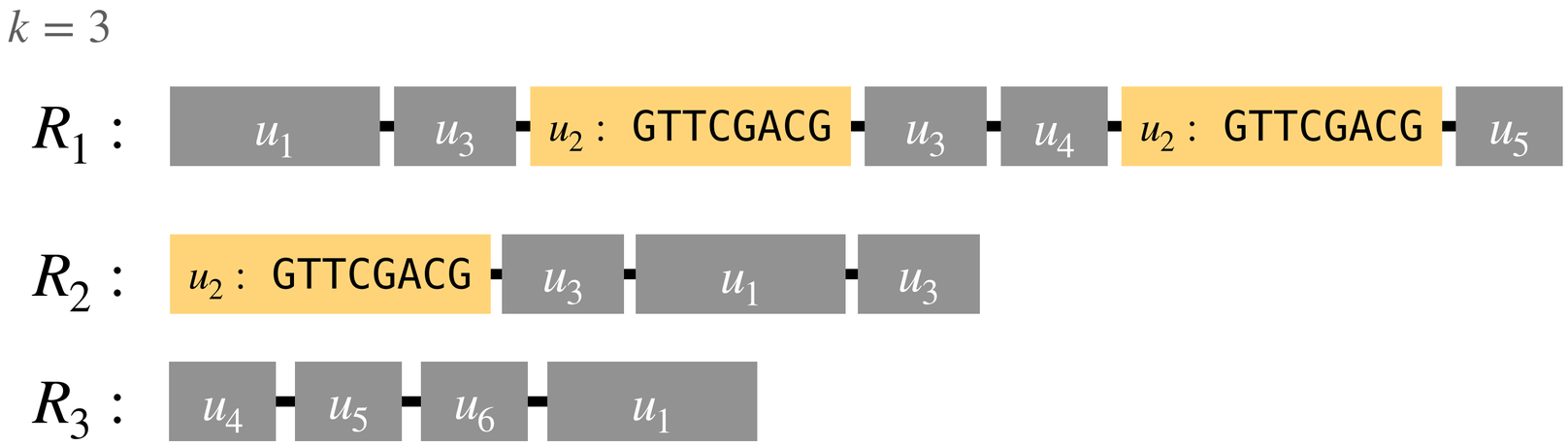
Spectrum preserving tilings (SPTs)

- From the references $\mathscr{R} = \{R_1, \ldots, R_m\}$ we build a ***reference*** **dBG.**
- Result: each reference is spelled by a **tiling of the unitigs** in the graph. We call this representation of $\mathscr{R}$ a “spectrum preserving tiling” (or SPT).
- (Each k-mer appears **once**, in a certain unitig.)



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

The inverted index



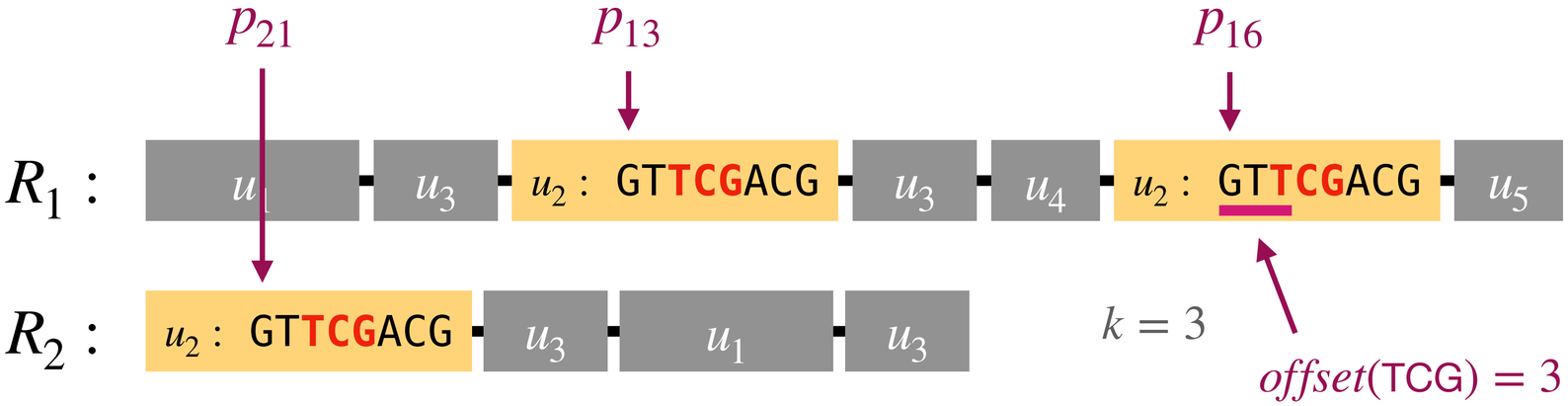
- **Q.** How are the inverted lists of the k-mers in the same unitig, say, u_2?
- **Property.** By construction of **SPTs**, the inverted lists of the k-mers in the same unitig are **identical**.
- So instead of keeping a separate inverted list for each k-mer in $\mathscr{D}$, we store inverted lists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

The interplay between $\mathscr{D}$ and $\mathscr{L}$

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

The interplay between $\mathscr{D}$ and $\mathscr{L}$

- Our map now is $unitig(x) \rightarrow L_{unitig(x)} = \{(i, \{p_{ij}\}) \,|\, unitig(x) \in R_i\}$.
- But we need the **position of the k-mer** in the reference (not that of the unitig)!
- **Solution. Store the unitigs in the dictionary $\mathscr{D}$ with SSHash** [P., 2022] to compute the relative position of the k-mer x in $unitig(x)$ on-the-fly. Let's call it $offset(x)$.
- The positions of x in $\mathscr{R}$ are computed as: $\{(i, \{offset(x) + p_{ij} - 1\}) \,|\, unitig(x) \in R_i\}$.



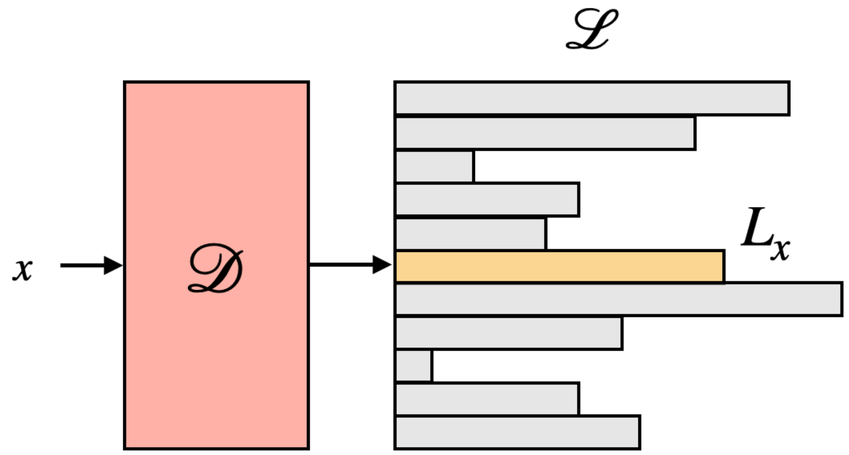
- $x =$ TCG
- The result is:
  $\{(1,\{3 + p_{13} - 1,\ 3 + p_{16} - 1\}),\ (2,\{3 + p_{21} - 1\})\}$

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.

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

The dictionary data structure

- From k-mer to unitig: $x \xrightarrow{\mathscr{D}} (unitig(x), offset(x))$

- For $\mathscr{D}$ we use **SSHash** — Sparse and Skew Hashing of k-mers [P., 2022]

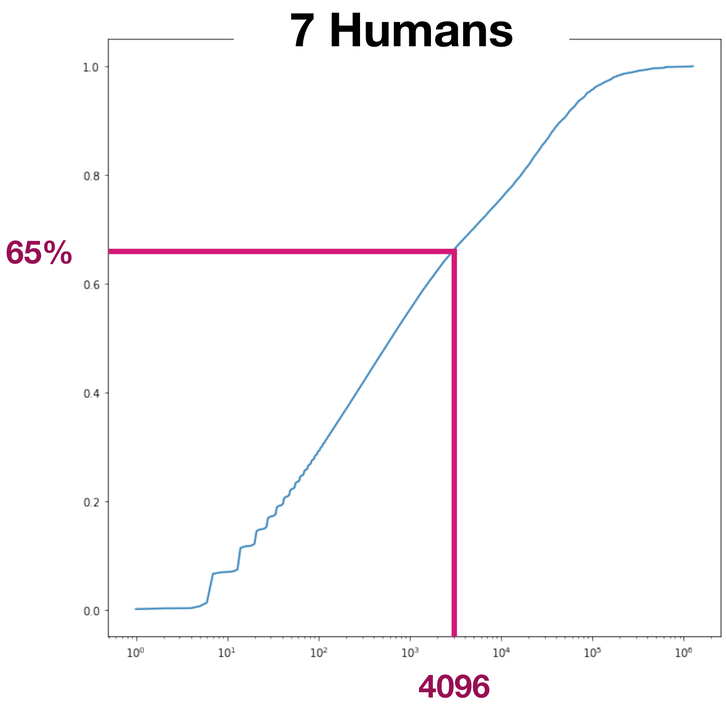
  - **Order-preserving**: *consecutive k-mers in the unitigs get consecutive hash codes* (that's how we implement the mapping)

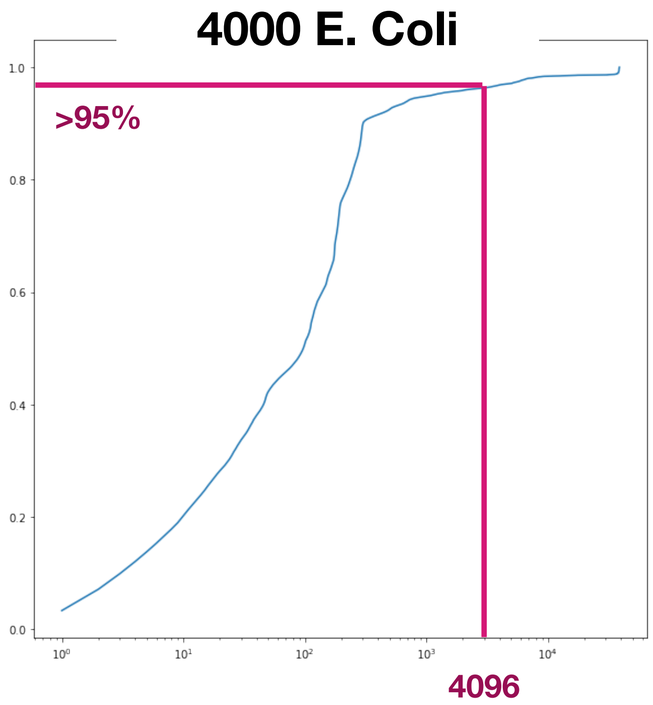
  - Fast and compact (builds on minimal perfect hashing and minimizers)
  - Exact, associative, weighted
  - Support for point/streaming/navigational queries
  - Scale to large datasets using external memory

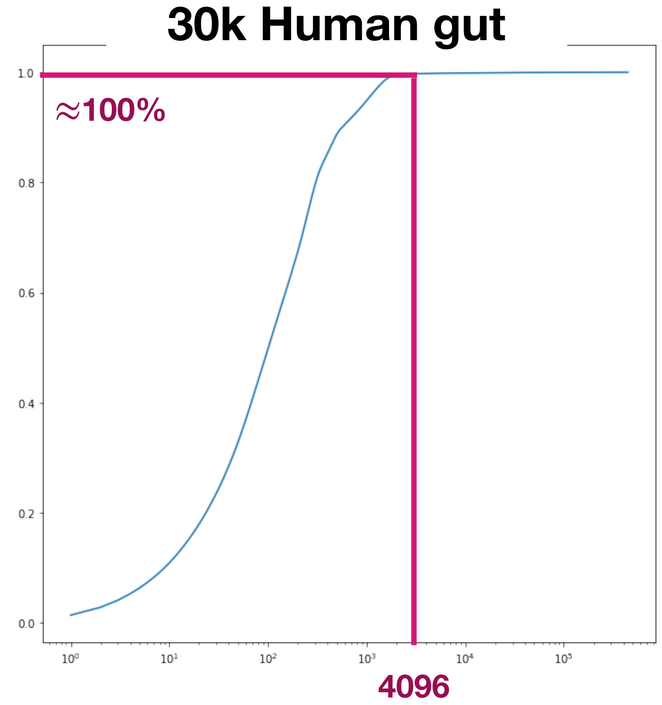
The inverted index data structure

- 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 to compress sorted integer sequences with different space/time trade-offs (see, e.g., [P. and Venturini, 2021]).
- But…on genomic collections, these lists are **short**. E.g., 90% or more are shorter than 1000 :(

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 (natural language) we retain 93%, 94%, 98% of the occurrences if we throw away lists shorter than 4096.

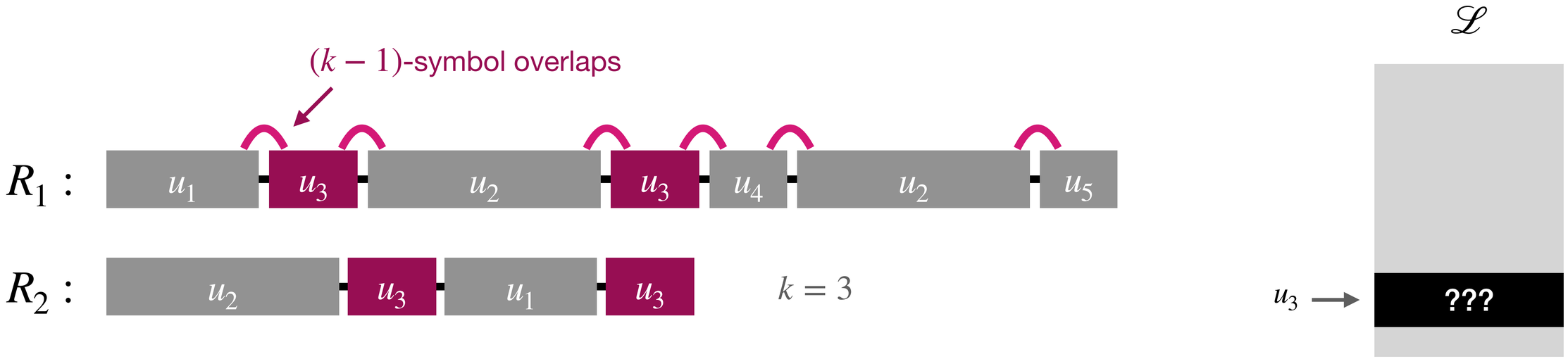
Sampling vs. Compression

- In [Fan, Khan, P., Patro 2023] 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 2023] 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**.

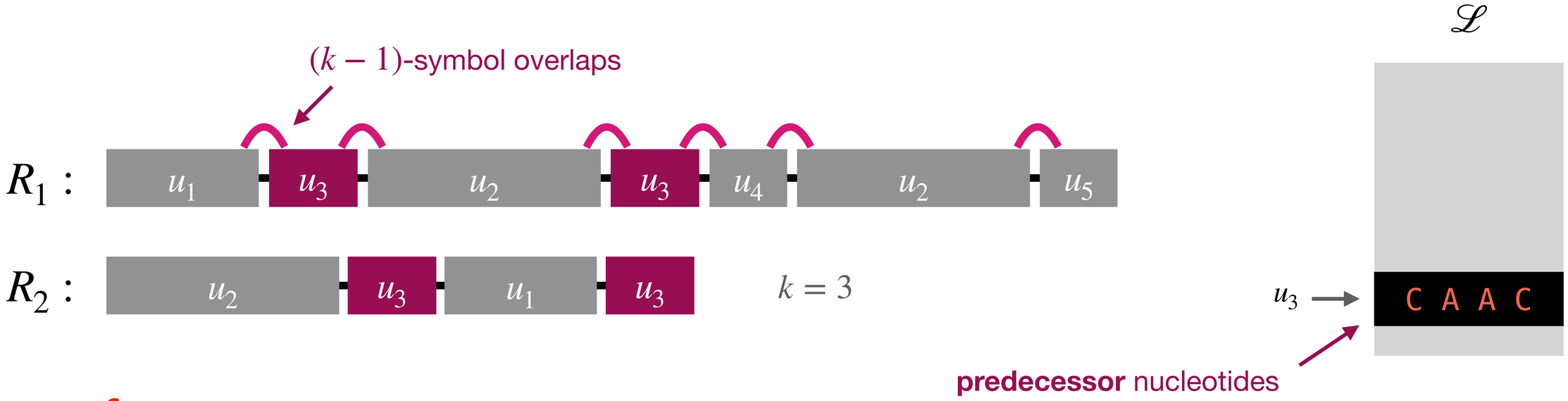
Suppose u_3 = **CGGT** is **not** sampled, but u_1 and u_2 are sampled.



Sampling vs. Compression

- In [Fan, Khan, P., Patro 2023] 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.



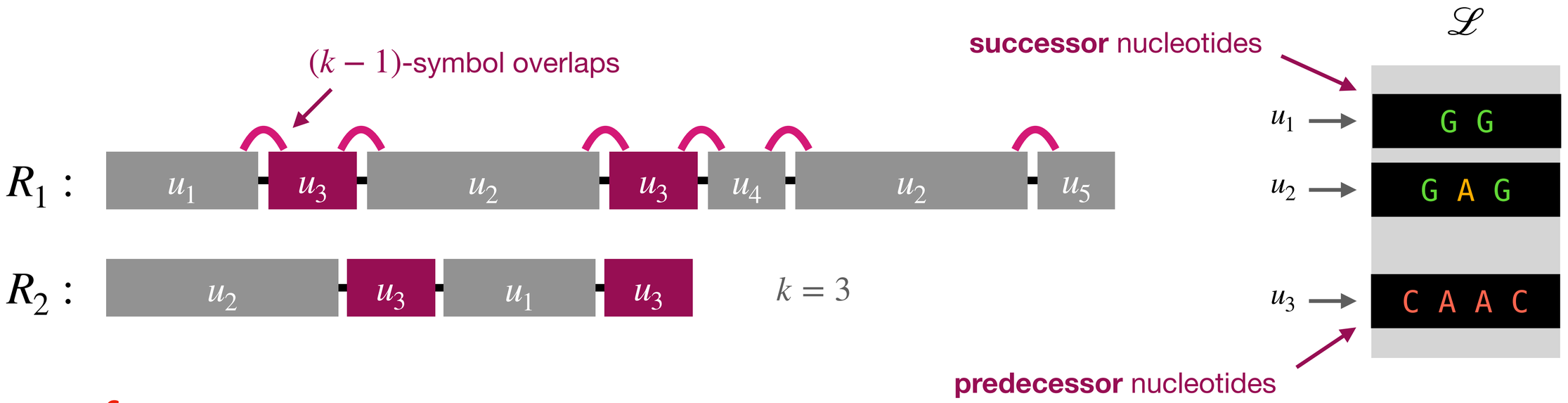
C
A **CGGT** = u_3 → query the **dictionary** for CCG and ACG:

$$unitig(\text{CCG}) = u_1 \text{ and } unitig(\text{ACG}) = u_2.$$

Sampling vs. Compression

- In [Fan, Khan, P., Patro 2023] 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:

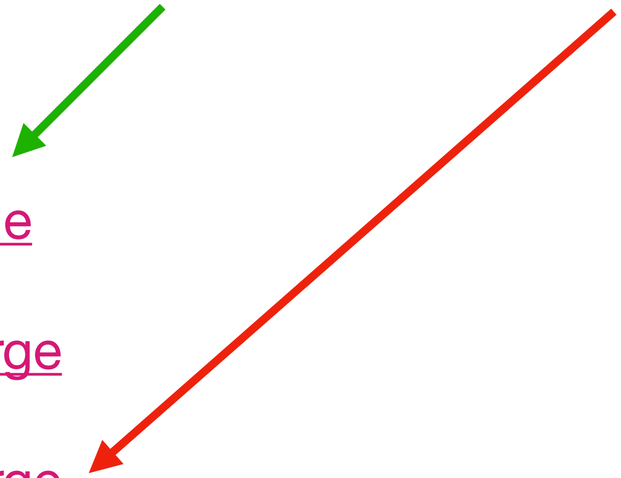
$$unitig(\text{CCG}) = u_1 \text{ and } unitig(\text{ACG}) = u_2.$$

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



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

Comparison against MONI

- **MONI** [Rossi et al., 2022] is based on the r-index [Gagie et al., 2018].
- Tested on 4,000 bacterial collection.
- Space: 51GB vs. 11GB (4.6X less space).
- 10M random (positive) k-mers: 1500 sec vs. 420 sec (3.6X faster).
- But MONI can find patterns of arbitrary length (not only k) and MEMs, which Pufferfish2 cannot do.

Conclusions

- **Spectrum preserving tilings (SPTs)** enable a modular and sparse solutions to the reference indexing problem based on 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 construction of **reference tilings:**
  - we can **reduce the number of lists and positions** stored in $\mathscr{L}$;
  - we can **sample** tiles' occurrences.

  This leads to a much better space usage compared to other state-of-the-art solutions.

- Depending on how $\mathscr{L}$ is represented (e.g., lossless/lossy, sampled/compressed, etc.): a whole *class* of related reference indexing data structures can be obtained.

- Try **Pufferfish2**: https://github.com/COMBINE-lab/pufferfish2.

Thank you for the attention!

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

“*Spectrum preserving tilings enable sparse and modular reference indexing*”
Jason Fan, Jamshed Khan, Giulio Ermanno Pibiri, and Rob Patro. 2023. RECOMB. To appear.