# Repetition-aware compression and query of colored de Bruijn graphs

**Giulio Ermanno Pibiri**

Ca’ Foscari University of Venice

@giulio_pibiri

@jermp

**RIKEN-AIP Workshop: Compressed Data Structures for Advanced Data Analysis**
Tokyo, Japan, 4 July 2025

# The colored k-mer indexing problem

- We are given a collection $\mathscr{R} = \{R_1, \ldots, R_N\}$ of reference sequences.
  Each $R_c$ is a (long) sequence over the DNA alphabet {A,C,G,T}.

- **Problem.** We want to build an *index* for $\mathscr{R}$ so that we can retrieve the set
  ColorSet$(x) = \{c \,|\, x \in R_c\}$ efficiently for any k-mer $x$. Note that ColorSet$(x) = \emptyset$ if $x \notin \mathscr{R}$.

# The colored k-mer indexing problem

- We are given a collection $\mathscr{R} = \{R_1, \ldots, R_N\}$ of reference sequences.
  Each $R_c$ is a (long) sequence over the DNA alphabet {A,C,G,T}.

- **Problem.** We want to build an *index* for $\mathscr{R}$ so that we can retrieve the set $\text{ColorSet}(x) = \{c \,|\, x \in R_c\}$ efficiently for any k-mer $x$. Note that $\text{ColorSet}(x) = \emptyset$ if $x \notin \mathscr{R}$.

- A lot of hype in the indexing community for the case where $\mathscr{R}$ is a **pangenome**, i.e., a collection of related genomes.

- **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.

# Modular indexing layout

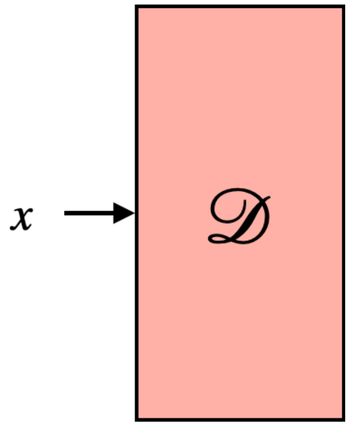
- **Goal.** What we want is the **map** $x \to \text{ColorSet}(x) = \{c \,|\, x \in R_c\}$.
- **Two** data structures:

# Modular indexing layout

- **Goal.** What we want is the **map** $x \to \text{ColorSet}(x) = \{c \,|\, x \in R_c\}$.
- **Two** data structures:

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

   $\mathscr{D}$ stores $n$ distinct k-mers and supports a Lookup($x$) operation which, given a k-mer $x$, returns a unique integer $1 \leq h \leq n$ if $x \in \mathscr{D}$, and $\perp$ otherwise.



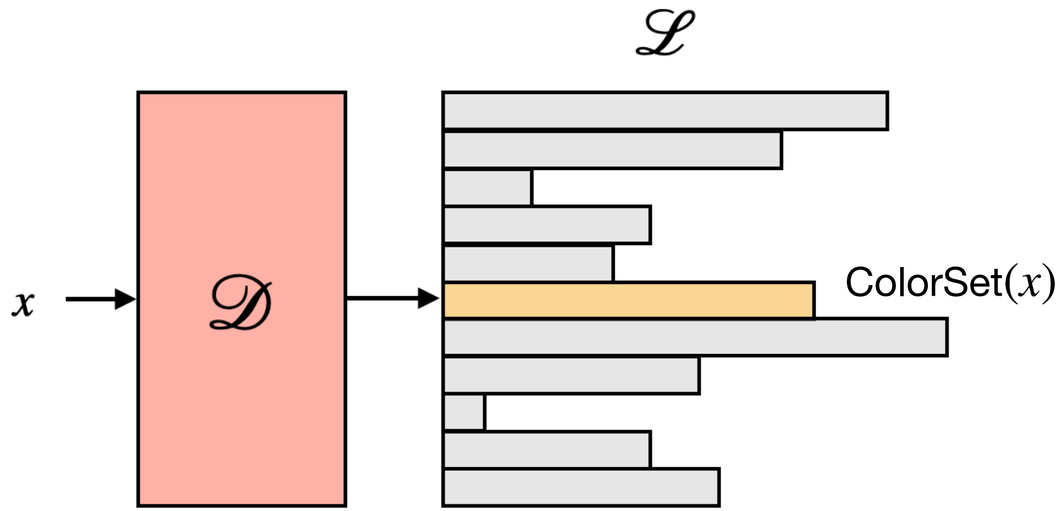
# Modular indexing layout

- **Goal.** What we want is the **map** $x \to \text{ColorSet}(x) = \{c \,|\, x \in R_c\}$.
- **Two** data structures:

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

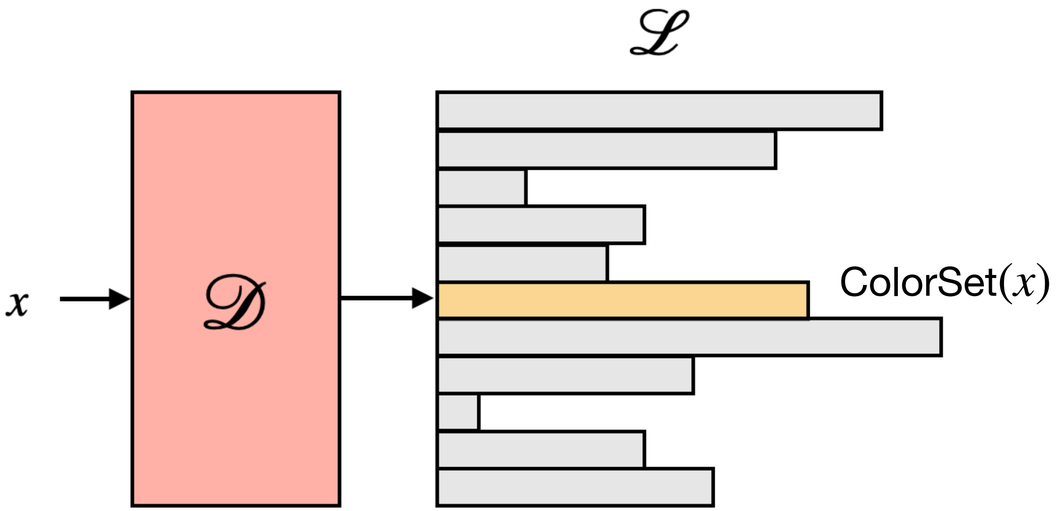
   $\mathscr{D}$ stores $n$ distinct k-mers and supports a Lookup$(x)$ operation which, given a k-mer $x$, returns a unique integer $1 \leq h \leq n$ if $x \in \mathscr{D}$, and $\perp$ otherwise.

2. The sets $\{\text{ColorSet}(x)\}_x$ are stored in order of Lookup$(x)$ in the **inverted index** $\mathscr{L}$.



# Modular indexing layout

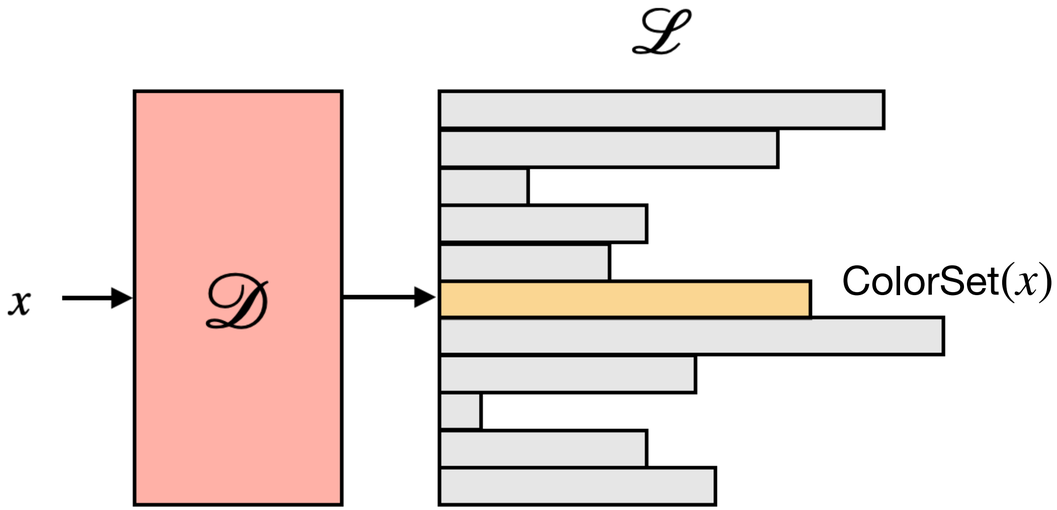
- Many k-mer based indexes (all of them?) are incarnations/adaptations of this **modular indexing layout**, $\mathscr{D} + \mathscr{L}$:




- deBGA [Liu et al. 2016]
- Kallisto [Bray et al. 2016]
- BIGSI [Bradley et al. 2017]
- Rainbowfish [Almodaresi et al. 2017]
- Mantis [Pandey et al. 2018]
- Pufferfish [Almodaresi et al. 2018]
- SeqOthello [Yu 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]
- Themisto [Alanko et. al 2023]
- **Fulgor** [Fan et. al 2023, 2024; P. et al. 2024; Campanelli et al., 2024, 2025]

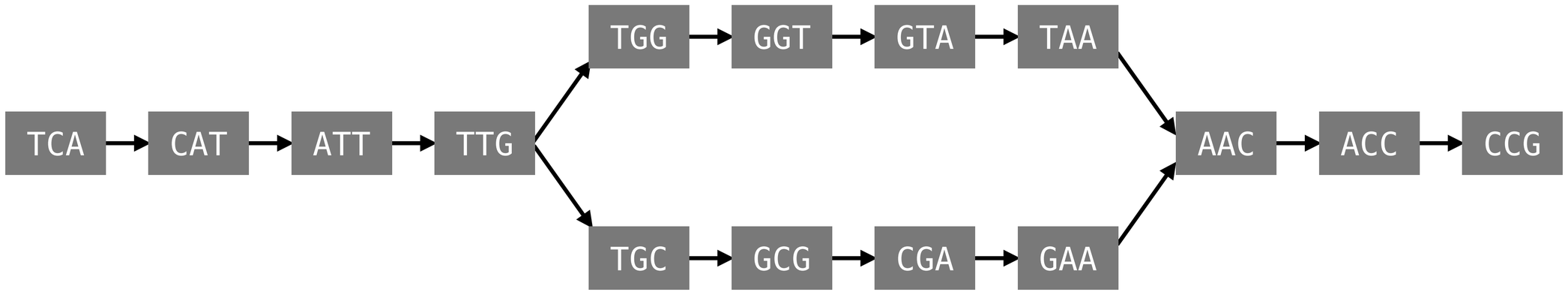
# Modular indexing layout

- Our problem reduces to that of **representing two data structures**, $\mathscr{D}$ and $\mathscr{L}$.
- To do so at best, we **must understand/exploit** the **properties** of our problem.
- **Q.** What are these properties?



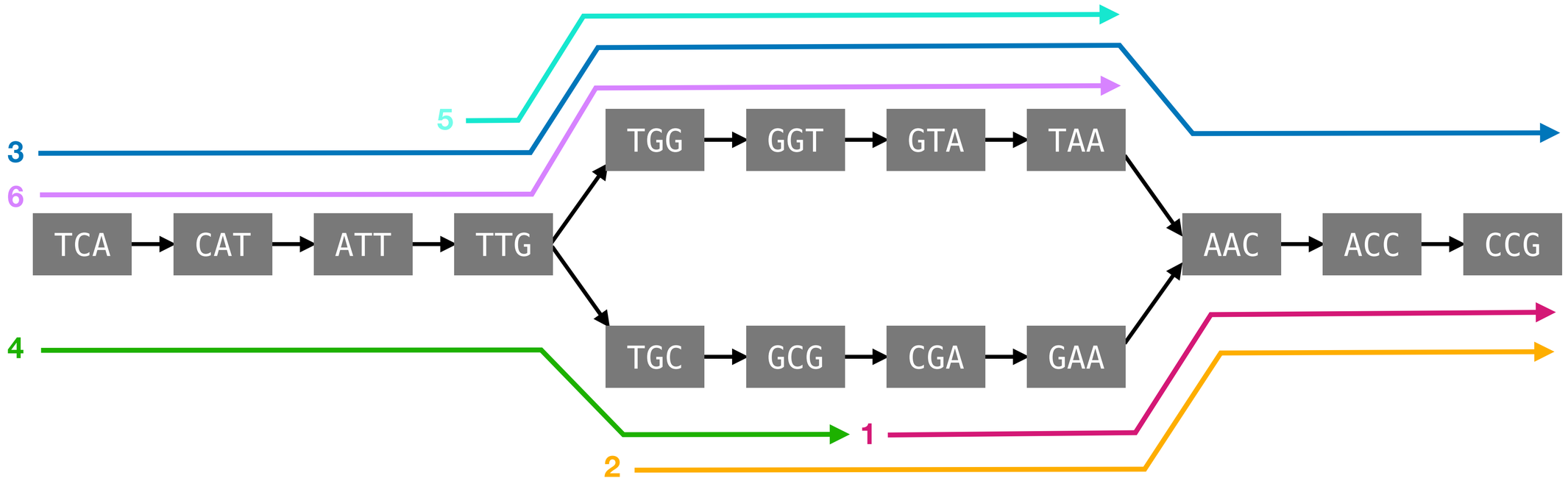
# de Bruijn graphs

- The dictionary $\mathscr{D}$ is a set of k-mers with (k-1)-symbol overlaps.
- One-to-one correspondence between $\mathscr{D}$ and a dBG.
- Example for **k = 3**.



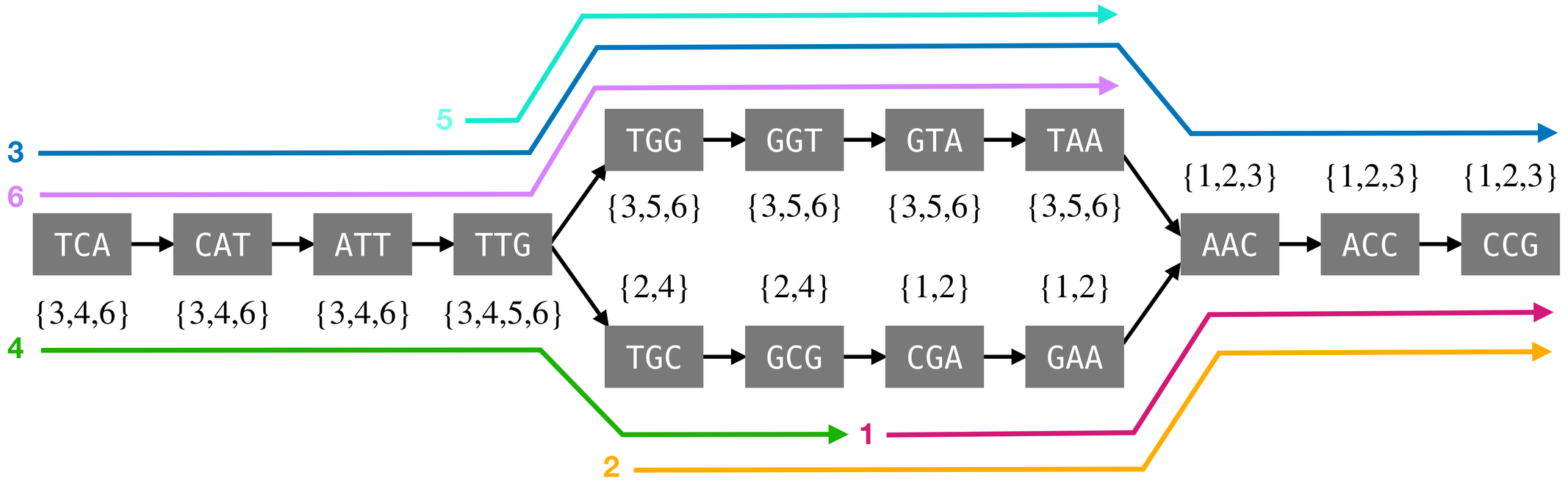
# Colored de Bruijn graphs

- Example for k = 3 and **N = 6 references.** References in $\mathscr{R}$ are spelled by **paths** in the graph.



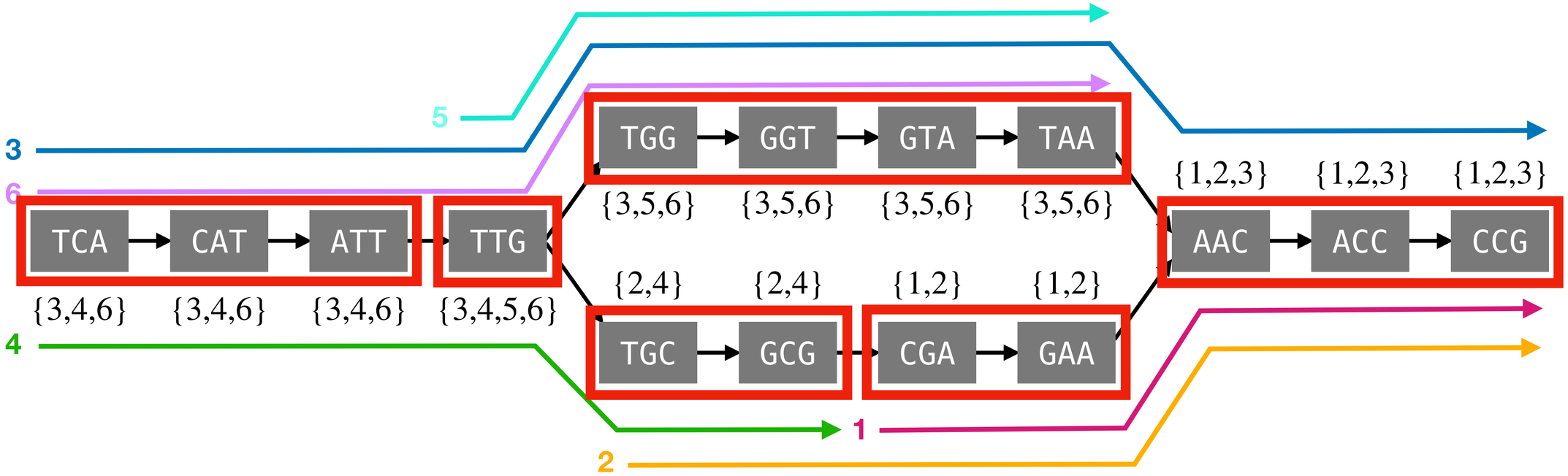
# Colored de Bruijn graphs

- Example for k = 3 and **N = 6 references.** References in $\mathscr{R}$ are spelled by **paths** in the graph.



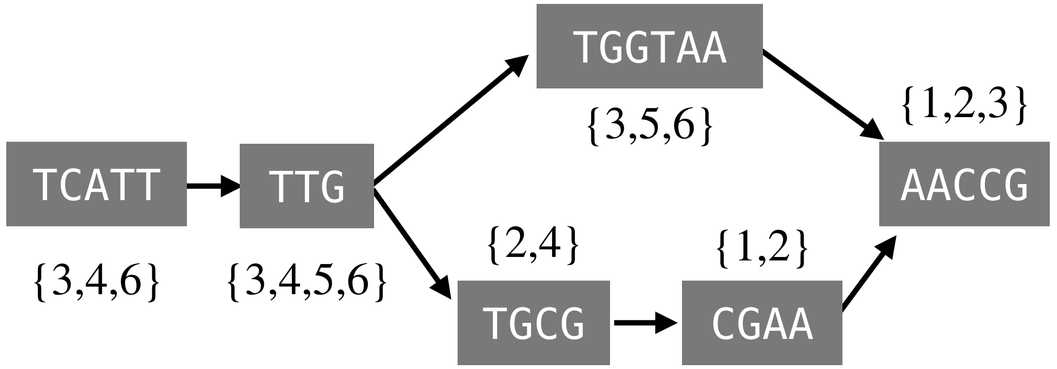
# Colored de Bruijn graphs

- Example for k = 3 and **N = 6 references.** References in $\mathscr{R}$ are spelled by **paths** in the graph.



# Colored compacted de Bruijn graphs

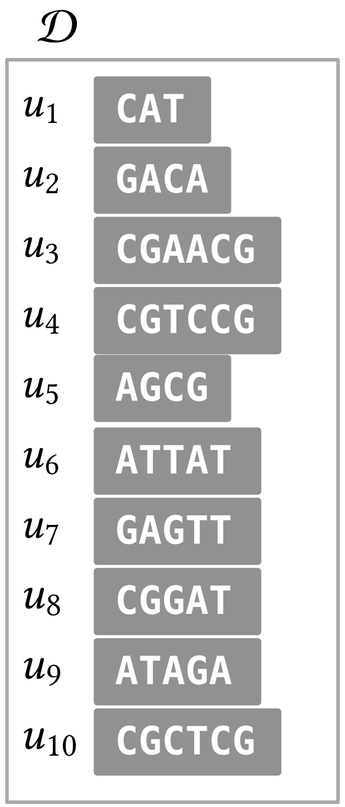
- Example for k = 3 and N = 6 references.
- Nodes having the **same color set along non-branching paths** are collapsed into **monochromatic unitigs**.




- Let’s now index this object!

# Properties of colored compacted dBGs

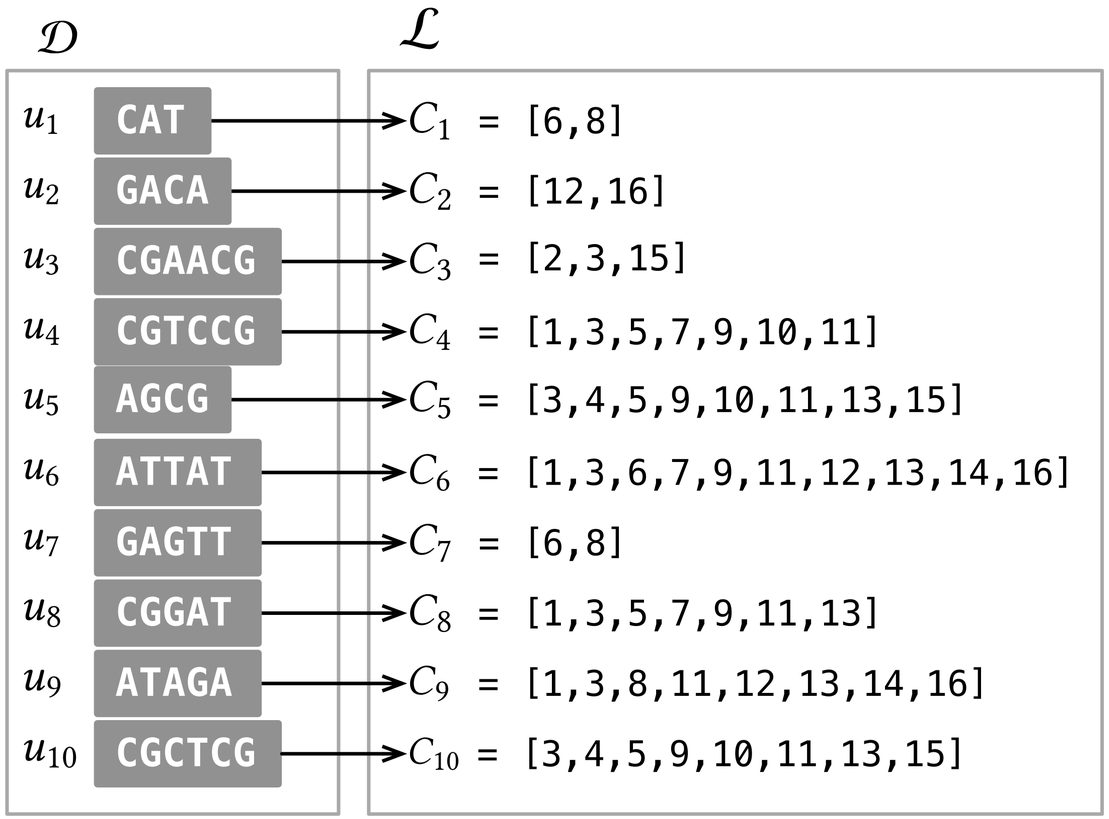
1. **Unitigs spell references in $\mathscr{R}$.**
   → We can **represent the set of unitigs** instead of the set of k-mers: represent $\mathscr{D}$ with **SSHash**.
   Better space effectiveness and cache locality (seen on Wed 2 July).



# Properties of colored compacted dBGs

2. **Unitigs are monochromatic.**
   → We store the color set of each unitig, rather than for each k-mer because ColorSet($x$) = ColorSet($y$) if k-mers $x$ and $y$ are part of the same unitig.
   Thus, we need an efficient **map from k-mers to unitigs**: $x \rightarrow$ Unitig($x$).

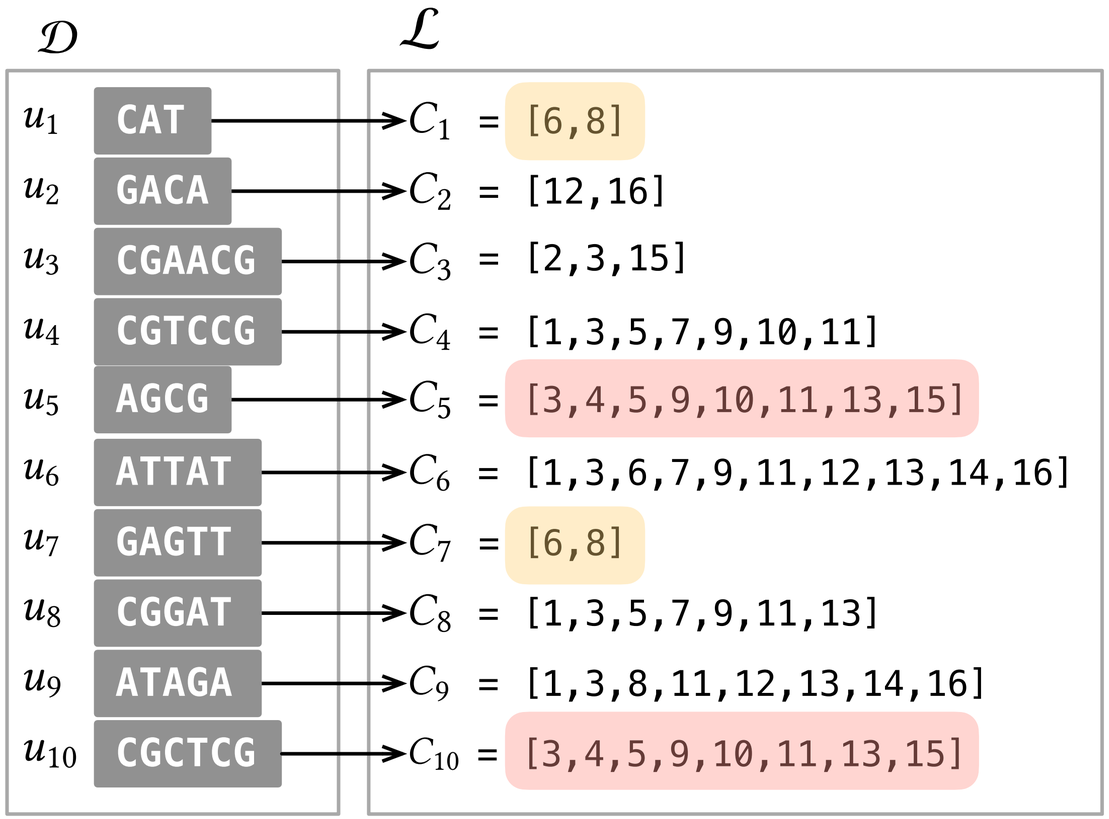
- For SSHash, it is easy to compute the **unitig identifier** Unitig($x$) given the k-mer $x$.
- Now $\mathscr{L}$ stores ColorSet($x$) for each unitig in the order given by Unitig($x$).



# Properties of colored compacted dBGs

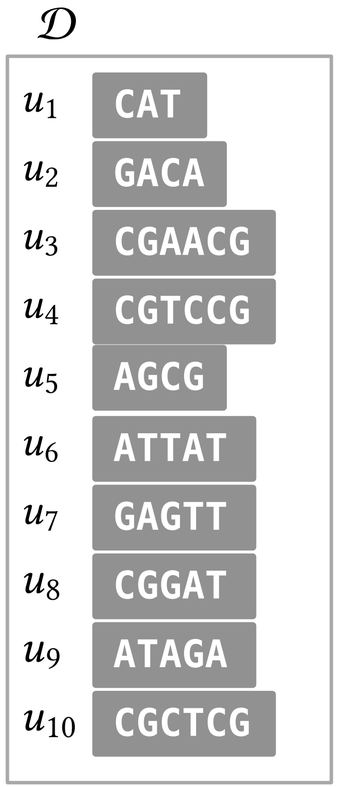
2. **Unitigs are monochromatic.**
   → We store the color set of each unitig, rather than for each k-mer because ColorSet($x$) = ColorSet($y$) if k-mers $x$ and $y$ are part of the same unitig.
   Thus, we need an efficient **map from k-mers to unitigs**: $x \rightarrow$ Unitig($x$).

- For SSHash, it is easy to compute the **unitig identifier** Unitig($x$) given the k-mer $x$.
- Now $\mathscr{L}$ stores ColorSet($x$) for each unitig in the order given by Unitig($x$).



# Properties of colored compacted dBGs

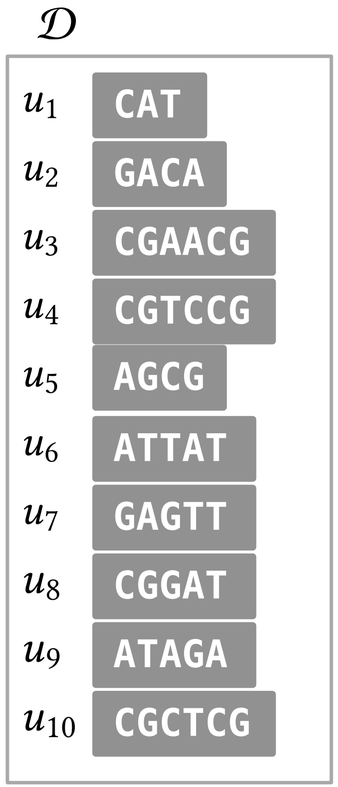
3. **Unitigs co-occur.** → Distinct unitigs often have the same color se, i.e., they co-occur in the same subset of references. We have **way less distinct color sets than unitigs**.
   We need an efficient **map from unitigs to distinct color sets:** Unitig($x$) → ColorSet($x$).



# Properties of colored compacted dBGs

3. **Unitigs co-occur.** → Distinct unitigs often have the same color se, i.e., they co-occur in the same subset of references. We have **way less distinct color sets than unitigs**.
   We need an efficient **map from unitigs to distinct color sets:** Unitig($x$) → ColorSet($x$).

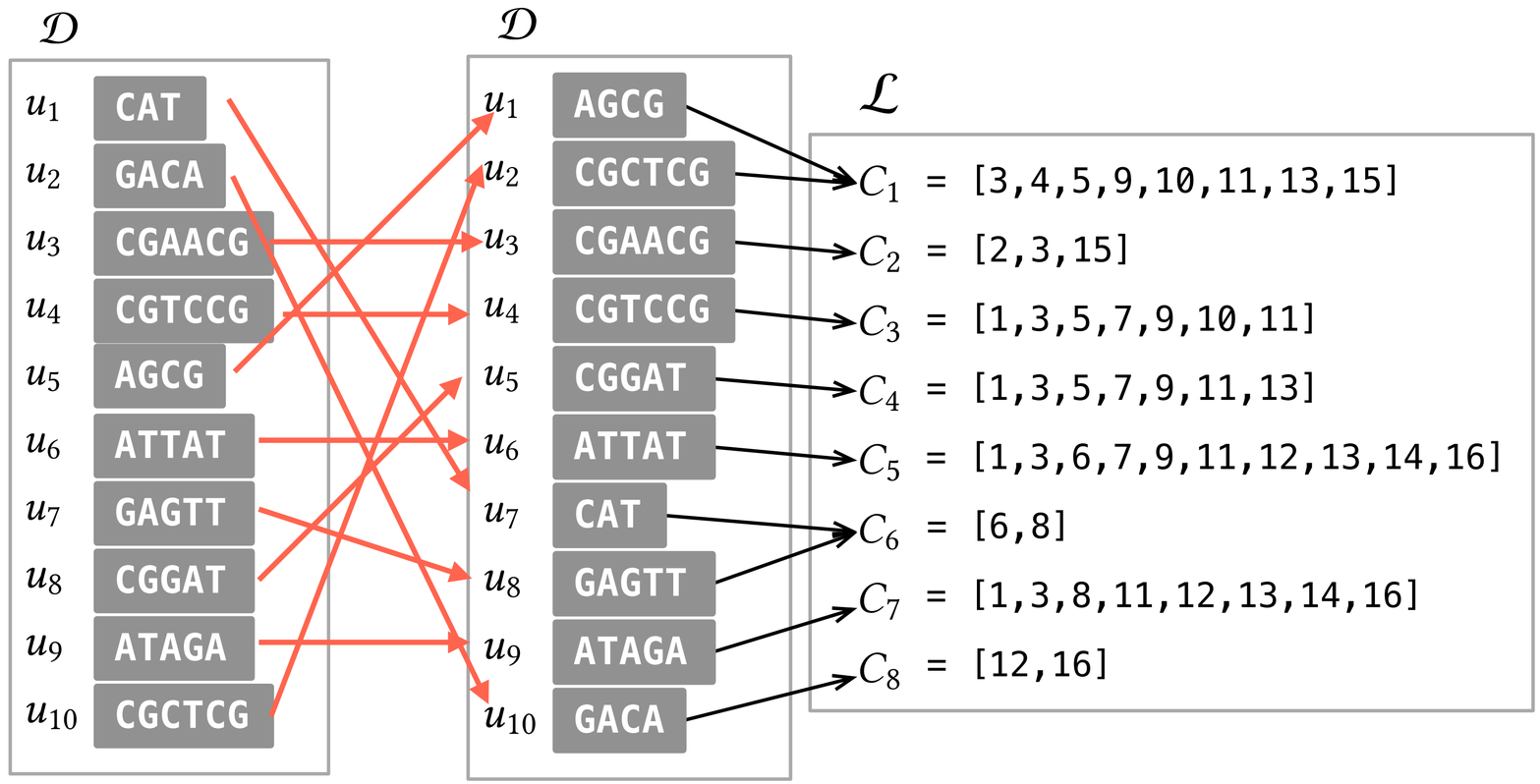
- Remember from Wed 2 July: SSHash stores a set of unitigs **in any wanted order** (*order-preserving*).
- We can thus **permute** the unitigs in $\mathscr{D}$ so that **consecutive unitigs have the same color set**.
- Then, mapping a unitig to its color is as simple as a **rank query** over a bitmap (see next).



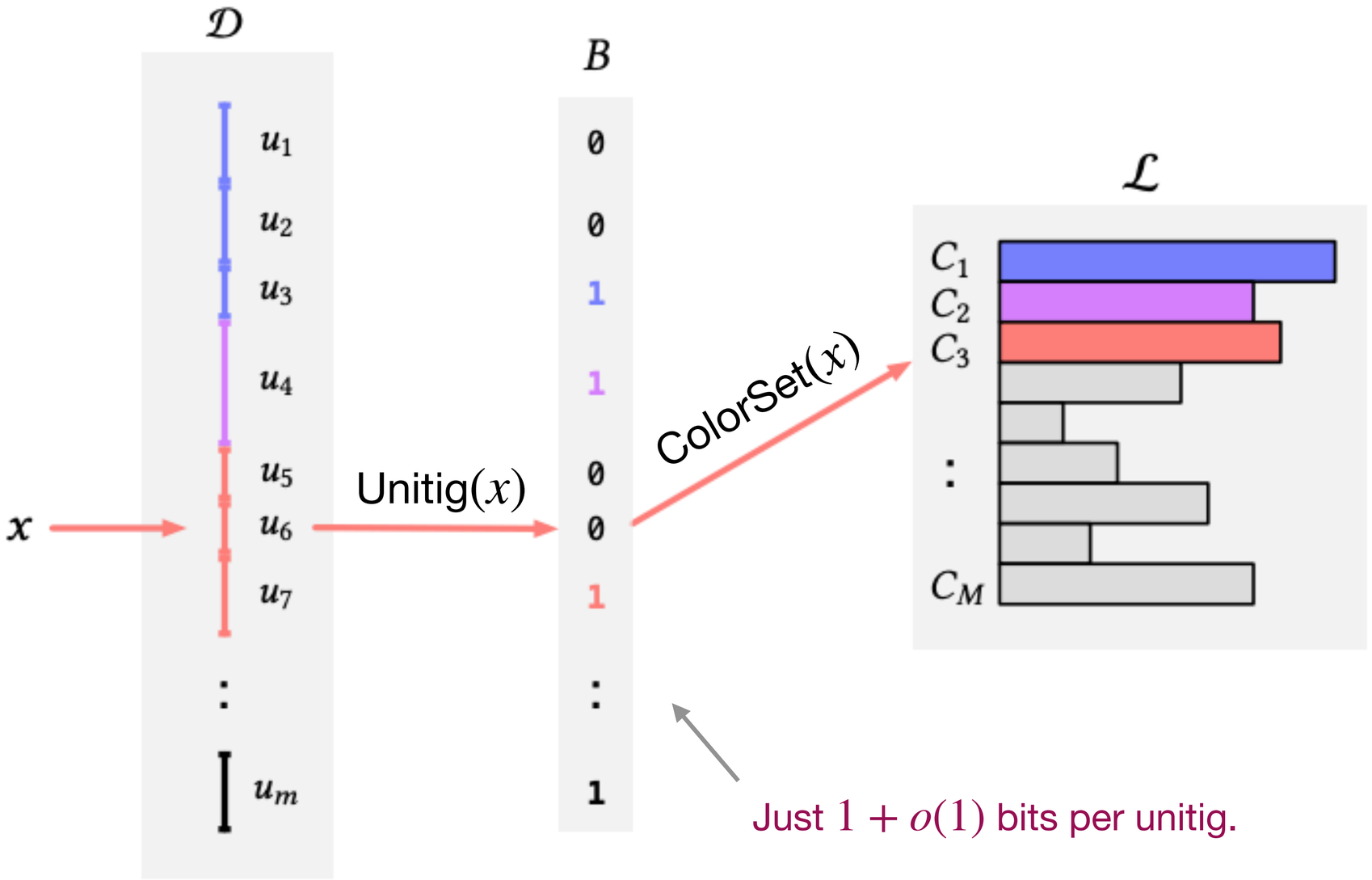
# Properties of colored compacted dBGs

3. **Unitigs co-occur.** → Distinct unitigs often have the same color se, i.e., they co-occur in the same subset of references. We have **way less distinct color sets than unitigs**.
   We need an efficient **map from unitigs to distinct color sets:** Unitig($x$) → ColorSet($x$).

- Remember from Wed 2 July: SSHash stores a set of unitigs **in any wanted order** (*order-preserving*).
- We can thus **permute** the unitigs in $\mathscr{D}$ so that **consecutive unitigs have the same color set**.
- Then, mapping a unitig to its color is as simple as a **rank query** over a bitmap (see next).



# Mapping unitigs to color sets in succinct space




Just $1 + o(1)$ bits per unitig.

# Compression of color sets

- **Any** method to compress integer sequences would work.

  Techniques for inverted index compression
  P. and Venturini, 2020, ACM Computing Surveys

- Here, we used three simple encodings based on the density of a set (ratio $|C|/N$):
  - If $|C|/N < 1/4$, the set is **sparse** and we compress the gaps with Elias' $\delta$.
  - If $|C|/N \geq 3/4$, the set is **very dense** and we compress the gaps of $\overline{C}$ (complementary set) with Elias' $\delta$.
  - Otherwise, the set is **dense** and we code it with a bitmap of $N$ bits.
  - (The thresholds 1/4 and 3/4 are optimal.)

# Results

## Fulgor (v1) https://github.com/jermp/fulgor/releases/tag/v1.0.0 (2023, 2024)

| | Genomes | Rate | Fulgor | | Themisto | | MetaG.-B | | MetaG.-NB | | COBS | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | mm:ss | GB | h:mm:ss | GB | mm:ss | GB | h:mm:ss | GB | h:mm:ss | GB |
| EC | 3,682 | 98.99 | 2:10 | 1.68 | 0:03:40 | 2.46 | 22:00 | 30.44 | 1:05:41 | 0.40 | 0:45:11 | 34.93 |
| SE | 5,000 | 89.49 | 1:16 | 0.82 | 0:03:50 | 1.82 | 14:14 | 36.54 | 0:20:32 | 0.33 | 0:38:34 | 41.93 |
| | 10,000 | 89.71 | 2:26 | 2.11 | 0:07:35 | 4.16 | 28:15 | 92.18 | 0:43:40 | 0.61 | 1:01:14 | 84.20 |
| | 50,000 | 91.25 | 19:15 | 18.53 | 0:42:02 | 33.14 | — | — | 4:30:03 | 2.72 | 3:54:18 | 408.82 |
| | 100,000 | 91.41 | 27:30 | 42.78 | 1:22:00 | 75.93 | — | — | 9:40:06 | 4.82 | 8:07:29 | 522.56 |
| | 150,000 | 91.52 | 42:30 | 70.55 | 2:00:13 | 124.27 | — | — | — | — | 7:47:14 | 522.63 |
| GB | 30,691 | 92.91 | 01:10 | 30.02 | 0:01:20 | 48.47 | 28:55 | 15.86 | 0:22:05 | 9.91 | 0:34:45 | 225.57 |

Full-intersection query mode; 16 processing threads; Max RSS in GB; output of `/usr/bin/time`.

# Yet another property!

1. **Unitigs spell references in $\mathscr{R}$.**
2. **Unitigs are monochromatic**.
3. **Unitigs co-occur.**

4. **Color sets are similar when indexing pangenomes.** → Opportunity to achieve **much better compression** if color sets are not compressed individually (each set independently of the others) but **common patterns are factored out and compressed once**.

# Color sets are similar when indexing pangenomes

$C_1$ = [3,4,5,9,10,11,13,15]

$C_2$ = [2,3,15]

$C_3$ = [1,3,5,7,9,10,11]

$C_4$ = [1,3,5,7,9,11,13]

$C_5$ = [1,3,6,7,9,11,12,13,14,16]

$C_6$ = [6,8]

$C_7$ = [1,3,8,11,12,13,14,16]

$C_8$ = [12,16]

- The pattern $\{3,5,9,11\}$ is currently represented **three times**.
- The pattern $\{1,11,12,13,14,16\}$ is represented **twice**.

# Color sets are similar when indexing pangenomes

$C_1$ = [3,4,5,9,10,11,13,15]

$C_2$ = [2,3,15]

$C_3$ = [1,3,5,7,9,10,11]

$C_4$ = [1,3,5,7,9,11,13]

$C_5$ = [1,3,6,7,9,11,12,13,14,16]

$C_6$ = [6,8]

$C_7$ = [1,3,8,11,12,13,14,16]

$C_8$ = [12,16]

- The pattern $\{3,5,9,11\}$ is currently represented **three times**.
- The pattern $\{1,11,12,13,14,16\}$ is represented **twice**.
- $C_3$ and $C_4$ are very similar.

# Color sets are similar when indexing pangenomes

$C_1$ = [3,4,5,9,10,11,13,15]

$C_2$ = [2,3,15]

$C_3$ = [1,3,5,7,9,10,11]

$C_4$ = [1,3,5,7,9,11,13]

$C_5$ = [1,3,6,7,9,11,12,13,14,16]

$C_6$ = [6,8]

$C_7$ = [1,3,8,11,12,13,14,16]

$C_8$ = [12,16]

- The pattern $\{3,5,9,11\}$ is currently represented **three times**.
- The pattern $\{1,11,12,13,14,16\}$ is represented **twice**.
- $C_3$ and $C_4$ are very similar.
- **Q.** How to **factor out** this redundancy?

# Two ways of partitioning the color sets

$C_1$ = [3,4,5,9,10,11,13,15]
$C_2$ = [2,3,15]
$C_3$ = [1,3,5,7,9,10,11]
$C_4$ = [1,3,5,7,9,11,13]
$C_5$ = [1,3,6,7,9,11,12,13,14,16]
$C_6$ = [6,8]
$C_7$ = [1,3,8,11,12,13,14,16]
$C_8$ = [12,16]

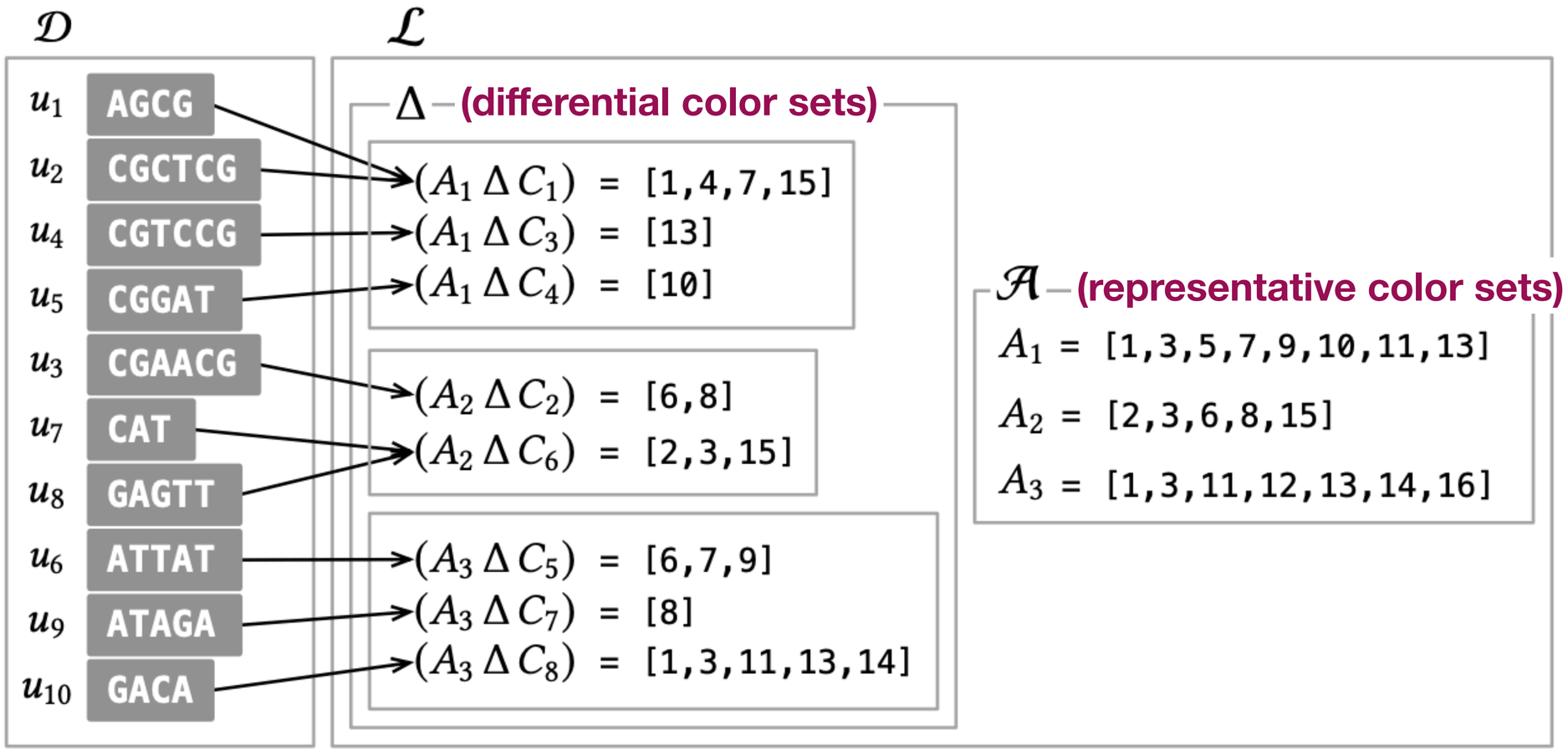
(a) Three horizontal partitions

$C_1$ = [3,4,5,9,10,11,13,15]
$C_2$ = [2,3,15]
$C_3$ = [1,3,5,7,9,10,11]
$C_4$ = [1,3,5,7,9,11,13]
$C_5$ = [1,3,6,7,9,11,12,13,14,16]
$C_6$ = [6,8]
$C_7$ = [1,3,8,11,12,13,14,16]
$C_8$ = [12,16]

(b) Four vertical partitions

# Horizontal partitioning: diff. and repr. color sets

- Example for **N = 16** references and **4** partitions.



# Vertical partitioning: meta and partial color sets

- Example for **N = 16** references and **4** partitions.

$$\begin{array}{rl} & \{\ 1\ 12\ 13\ 14\ 16\ \}\{\ 3\ 5\ 9\ \}\{\ 7\ 11\ \}\{\ 2\ \ 4\ \ 6\ \ 8\ \ 10\ 15\ \} \\ new\ identifiers \rightarrow & 1\ \ 2\ \ 3\ \ 4\ \ 5\ \ \ \ 6\ 7\ 8\ \ \ \ 9\ 10\ \ \ \ 11\ 12\ 13\ 14\ 15\ 16 \end{array}$$

this defines a permutation $\pi$

$\mathcal{L}$

$C_1$ = [3,4,5,9,10,11,13,15]

$C_2$ = [2,3,15]

$C_3$ = [1,3,5,7,9,10,11]

$C_4$ = [1,3,5,7,9,11,13]

$C_5$ = [1,3,6,7,9,11,12,13,14,16]

$C_6$ = [6,8]

$C_7$ = [1,3,8,11,12,13,14,16]

$C_8$ = [12,16]

$\pi$

$C_1 = [3|6,7,8|10|12,15,16]$

$C_2 = [6|11,16]$

$C_3 = [1|6,7,8|9,10|15]$

$C_4 = [1,3|6,7,8|9,10]$

$C_5 = [1,2,3,4,5|6,8|9,10|13]$

$C_6 = [13,14]$

$C_7 = [1,2,3,4,5|6|10|14]$

$C_8 = [2,5]$

# Vertical partitioning: meta and partial color sets

- Example for **N = 16** references and **4** partitions.

$$\begin{array}{rl} & \{\ 1\ 12\ 13\ 14\ 16\ \}\{\ 3\ 5\ 9\ \}\{\ 7\ 11\ \}\{\ 2\ \ 4\ \ 6\ \ 8\ \ 10\ 15\ \} \\ new\ identifiers \rightarrow & 1\ \ 2\ \ 3\ \ 4\ \ 5\ \ \boxed{6\ 7\ 8}\ \ 9\ 10\ \ 11\ 12\ 13\ 14\ 15\ 16 \end{array}$$

partition 2

this defines a permutation $\pi$

$\mathcal{L}$

$C_1$ = [3,4,5,9,10,11,13,15]

$C_2$ = [2,3,15]

$C_3$ = [1,3,5,7,9,10,11]

$C_4$ = [1,3,5,7,9,11,13]

$C_5$ = [1,3,6,7,9,11,12,13,14,16]

$C_6$ = [6,8]

$C_7$ = [1,3,8,11,12,13,14,16]

$C_8$ = [12,16]

$\pi$

$C_1 = [3|6,7,8|10|12,15,16]$

$C_2 = [6|11,16]$

$C_3 = [1|6,7,8|9,10|15]$

$C_4 = [1,3|6,7,8|9,10]$

$C_5 = [1,2,3,4,5|6,8|9,10|13]$

$C_6 = [13,14]$

$C_7 = [1,2,3,4,5|6|10|14]$

$C_8 = [2,5]$

# Vertical partitioning: meta and partial color sets

- Example for **N = 16** references and **4** partitions.

$$\begin{array}{rl} & \{\ 1\ 12\ 13\ 14\ 16\ \}\{\ 3\ 5\ 9\ \}\{\ 7\ 11\ \}\{\ 2\ \ 4\ \ 6\ \ 8\ \ 10\ 15\ \} \\ new\ identifiers \rightarrow & 1\ \ 2\ \ 3\ \ 4\ \ 5\ \ \boxed{6\ 7\ 8}\ \ 9\ 10\ \ 11\ 12\ 13\ 14\ 15\ 16 \end{array}$$

partition 2

this defines a permutation $\pi$

$\mathcal{L}$

$C_1$ = [3,4,5,9,10,11,13,15]
$C_2$ = [2,3,15]
$C_3$ = [1,3,5,7,9,10,11]
$C_4$ = [1,3,5,7,9,11,13]
$C_5$ = [1,3,6,7,9,11,12,13,14,16]
$C_6$ = [6,8]
$C_7$ = [1,3,8,11,12,13,14,16]
$C_8$ = [12,16]

$\pi$

$C_1 = [3|6,7,8|10|12,15,16]$
$C_2 = [6|11,16]$
$C_3 = [1|6,7,8|9,10|15]$
$C_4 = [1,3|6,7,8|9,10]$
$C_5 = [1,2,3,4,5|6,8|9,10|13]$
$C_6 = [13,14]$
$C_7 = [1,2,3,4,5|6|10|14]$
$C_8 = [2,5]$

# Vertical partitioning: meta and partial color sets

- Example for **N = 16** references and **4** partitions.

$$\begin{array}{rl} & \{\ 1\ 12\ 13\ 14\ 16\ \}\{\ 3\ 5\ 9\ \}\{\ 7\ 11\ \}\{\ 2\ \ 4\ \ 6\ \ 8\ \ 10\ 15\ \} \\ new\ identifiers \rightarrow & 1\ \ 2\ \ 3\ \ 4\ \ 5\ \ \boxed{6\ 7\ 8}\ \ 9\ 10\ \ 11\ 12\ 13\ 14\ 15\ 16 \end{array}$$

partition 2

this defines a permutation $\pi$

$\mathcal{L}$

$C_1$ = [3,4,5,9,10,11,13,15]
$C_2$ = [2,3,15]
$C_3$ = [1,3,5,7,9,10,11]
$C_4$ = [1,3,5,7,9,11,13]
$C_5$ = [1,3,6,7,9,11,12,13,14,16]
$C_6$ = [6,8]
$C_7$ = [1,3,8,11,12,13,14,16]
$C_8$ = [12,16]

$\pi$

$$\begin{aligned} C_1 &= [3|6,7,8|10|12,15,16] \\ C_2 &= [6|11,16] \\ C_3 &= [1|6,7,8|9,10|15] \\ C_4 &= [1,3|6,7,8|9,10] \\ C_5 &= [1,2,3,4,5|6,8|9,10|13] \\ C_6 &= [13,14] \\ C_7 &= [1,2,3,4,5|6|10|14] \\ C_8 &= [2,5] \end{aligned}$$

[6,7,8]
[6]
[6,8]

# Vertical partitioning: meta and partial color sets

- Example for **N = 16** references and **4** partitions.

$$\{\ 1\ 12\ 13\ 14\ 16\ \}\{\ 3\ 5\ 9\ \}\{\ 7\ 11\ \}\{\ 2\ \ 4\ \ 6\ \ 8\ \ 10\ 15\ \}$$
$$new\ identifiers \rightarrow 1\ \ 2\ \ 3\ \ 4\ \ 5\ \ \boxed{6\ 7\ 8}\ \ 9\ 10\ \ 11\ 12\ 13\ 14\ 15\ 16$$

partition 2

this defines a permutation $\pi$

$\mathcal{L}$

$C_1$ = [3,4,5,9,10,11,13,15]
$C_2$ = [2,3,15]
$C_3$ = [1,3,5,7,9,10,11]
$C_4$ = [1,3,5,7,9,11,13]
$C_5$ = [1,3,6,7,9,11,12,13,14,16]
$C_6$ = [6,8]
$C_7$ = [1,3,8,11,12,13,14,16]
$C_8$ = [12,16]

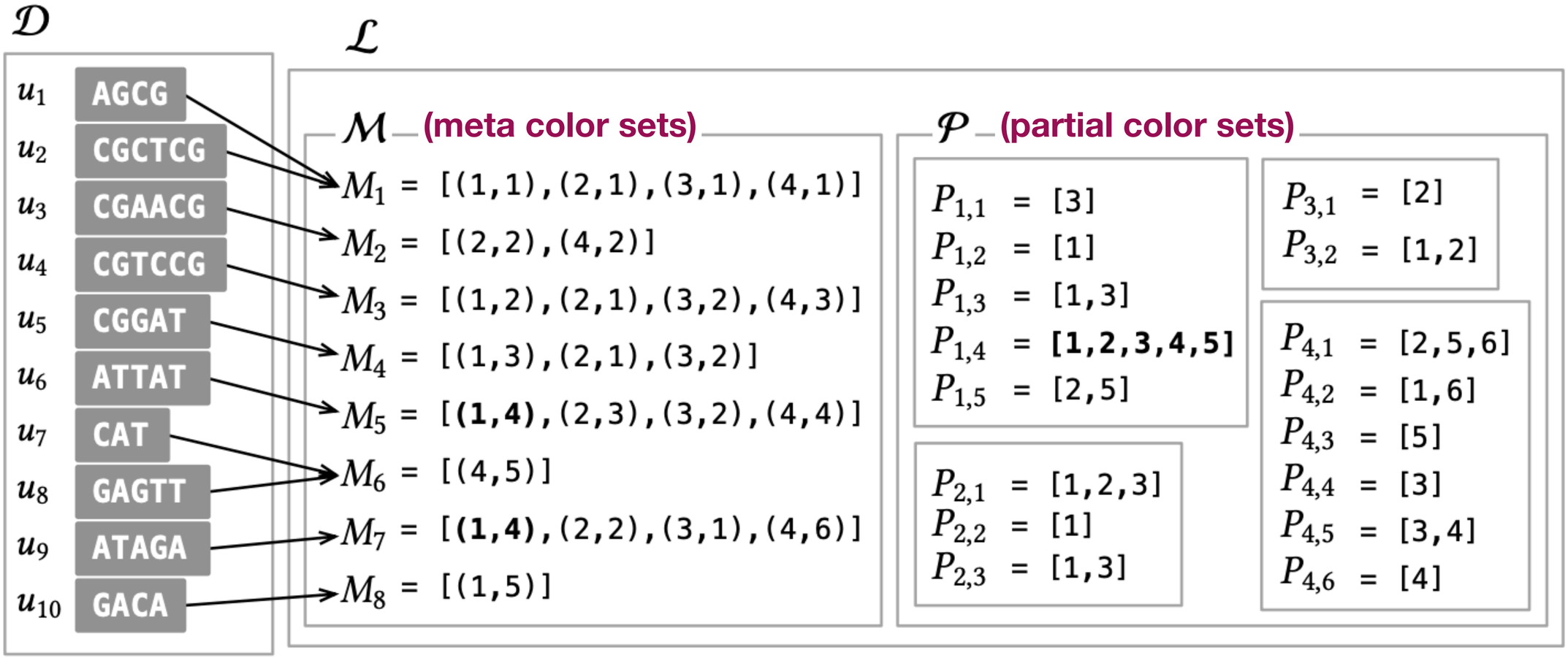
$\pi$

$C_1 = [3|6,7,8|10|12,15,16]$
$C_2 = [6|11,16]$
$C_3 = [1|6,7,8|9,10|15]$
$C_4 = [1,3|6,7,8|9,10]$
$C_5 = [1,2,3,4,5|6,8|9,10|13]$
$C_6 = [13,14]$
$C_7 = [1,2,3,4,5|6|10|14]$
$C_8 = [2,5]$

[6,7,8]
[6]
[6,8]
-5
[1,2,3]
[1]
[1,3]

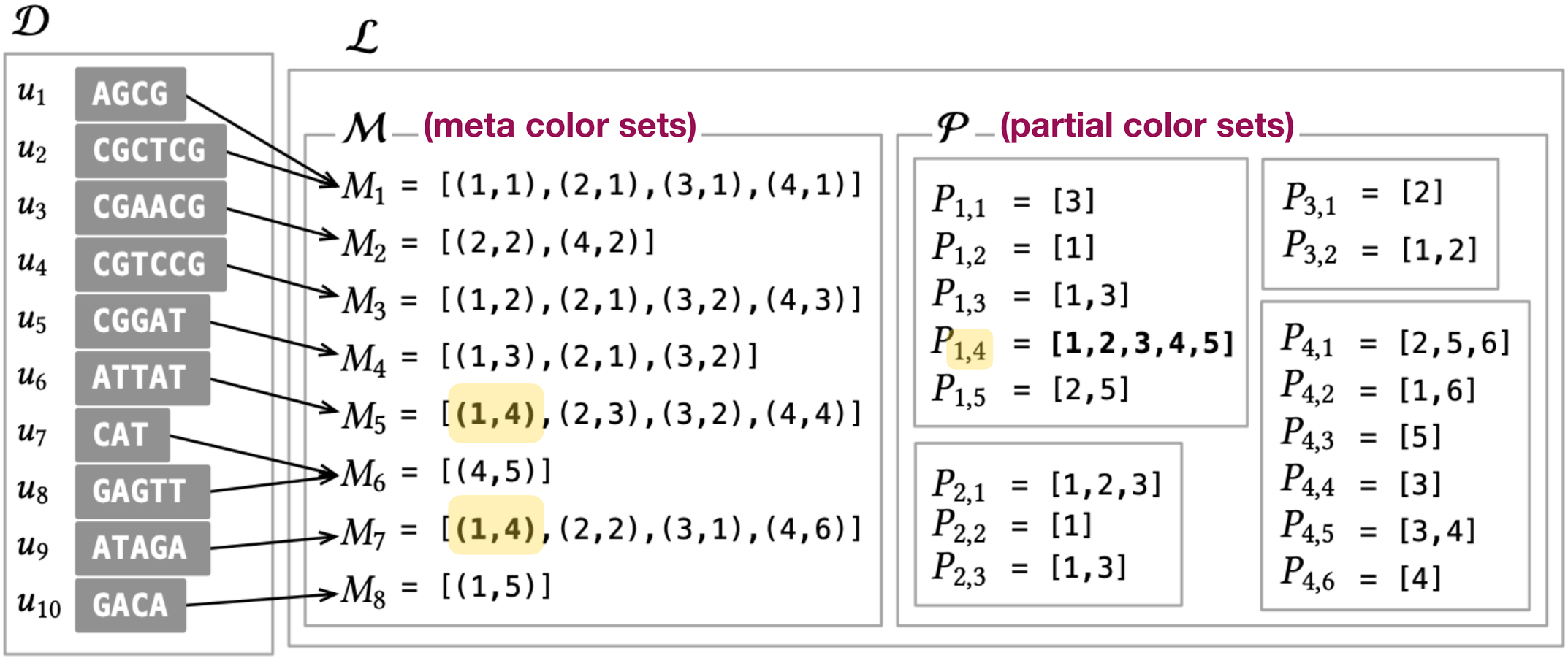
# Vertical partitioning: meta and partial color sets

- Example for **N = 16** references and **4** partitions.



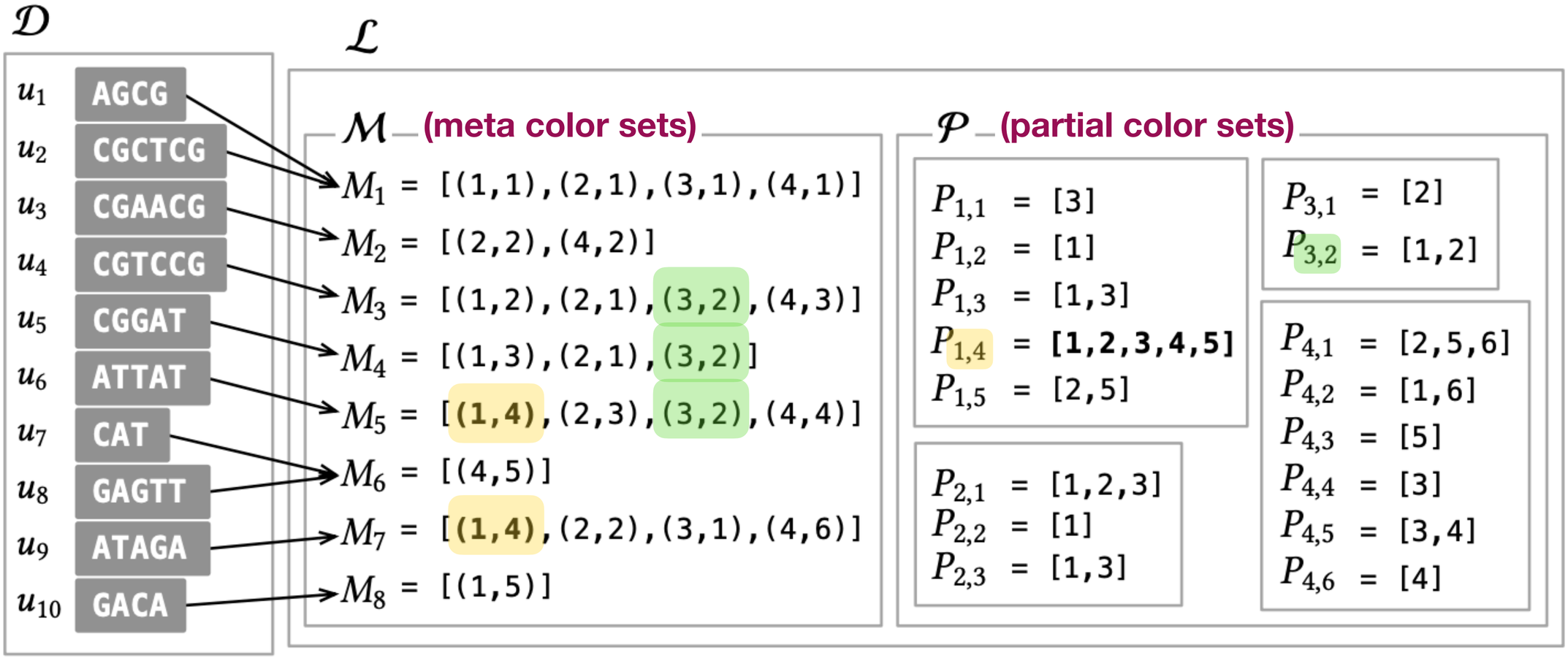
# Vertical partitioning: meta and partial color sets

- Example for **N = 16** references and **4** partitions.



# Vertical partitioning: meta and partial color sets

- Example for **N = 16** references and **4** partitions.



# Results — Index size

**Fulgor** (v3) https://github.com/jermp/fulgor/releases/tag/v3.0.0 (2024)

| Dataset | Fulgor | | | d-Fulgor | | m-Fulgor | | md-Fulgor | |
|---|---|---|---|---|---|---|---|---|---|
| | dBG | Color sets | Total | Color sets | Total | Color sets | Total | Color sets | Total |
| EC | 0.29 | 1.36 (83%) | 1.65 | 0.45 (61%) | 0.74 | 0.40 (58%) | 0.69 | 0.24 (45%) | 0.52 |
| SE-5K | 0.16 | 0.59 (79%) | 0.75 | 0.20 (56%) | 0.36 | 0.16 (50%) | 0.32 | 0.11 (40%) | 0.27 |
| SE-10K | 0.35 | 1.66 (83%) | 2.01 | 0.48 (58%) | 0.83 | 0.34 (49%) | 0.70 | 0.22 (39%) | 0.57 |
| SE-50K | 1.25 | 17.03 (93%) | 18.29 | 4.31 (77%) | 5.57 | 2.08 (62%) | 3.34 | 1.38 (52%) | 2.64 |
| SE-100K | 1.71 | 40.71 (96%) | 42.43 | 9.37 (84%) | 11.10 | 3.75 (68%) | 5.47 | 2.26 (57%) | 3.98 |
| SE-150K | 2.02 | 68.61 (97%) | 70.65 | 15.73 (89%) | 17.77 | 5.27 (72%) | 7.31 | 3.22 (61%) | 5.26 |
| GB | 21.29 | 15.54 (42%) | 36.83 | 7.51 (26%) | 28.81 | 9.16 (30%) | 30.46 | 6.19 (23%) | 27.48 |

# Save money!

**Amazon** EC2 instances pricing:

- https://instances.vantage.sh/aws/ec2/x2gd.medium
  16 GiB of RAM — **73 $** per month
- https://instances.vantage.sh/aws/ec2/x2gd.xlarge
  64 GiB of RAM — **292 $** per month
- https://instances.vantage.sh/aws/ec2/x2gd.2xlarge
  128 GiB of RAM — **584 $** per month
- https://instances.vantage.sh/aws/ec2/x2gd.4xlarge
  256 GiB of RAM — **1168 $** per month

Numbers taken on 25/06/2025.

md-Fulgor: **5.3** GB

m-Fulgor: **7.3** GB

d-Fulgor: **18** GB

Fulgor: **71** GB

Original data: **225** GB

(150,000 S. Enterica genomes, compressed with `gzip`)

# Results — Query time

## Fulgor (v3) https://github.com/jermp/fulgor/releases/tag/v3.0.0 (2024)

| Dataset | Mapping rate | h-Fulgor | | d-Fulgor | | m-Fulgor | | md-Fulgor | |
|---|---|---|---|---|---|---|---|---|---|
| | | | GB | | GB | | GB | | GB |
| EC | 98.99 | 2:12 | 1.67 | 4:52 | 0.78 | 3:08 | 0.73 | 6:07 | 0.57 |
| SE-5K | 89.49 | 1:14 | 0.80 | 1:54 | 0.41 | 1:25 | 0.37 | 2:10 | 0.32 |
| SE-10K | 89.71 | 2:29 | 2.06 | 4:14 | 0.90 | 2:56 | 0.77 | 4:55 | 0.65 |
| SE-50K | 91.25 | 14:05 | 18.24 | 27:25 | 5.82 | 17:00 | 3.64 | 33:25 | 2.95 |
| SE-100K | 91.41 | 29:00 | 42.40 | 58:10 | 11.58 | 34:40 | 6.08 | 1:09:00 | 4.63 |
| SE-150K | 91.52 | 44:30 | 70.55 | 1:31:00 | 18.55 | 53:00 | 8.29 | 1:50:00 | 6.29 |
| GB | 92.91 | 1:10 | 36.01 | 1:00 | 28.25 | 1:09 | 29.79 | 1:03 | 26.88 |

Full-intersection query mode; 16 processing threads; Max RSS in GB; output of `/usr/bin/time`.

# Faster query times!

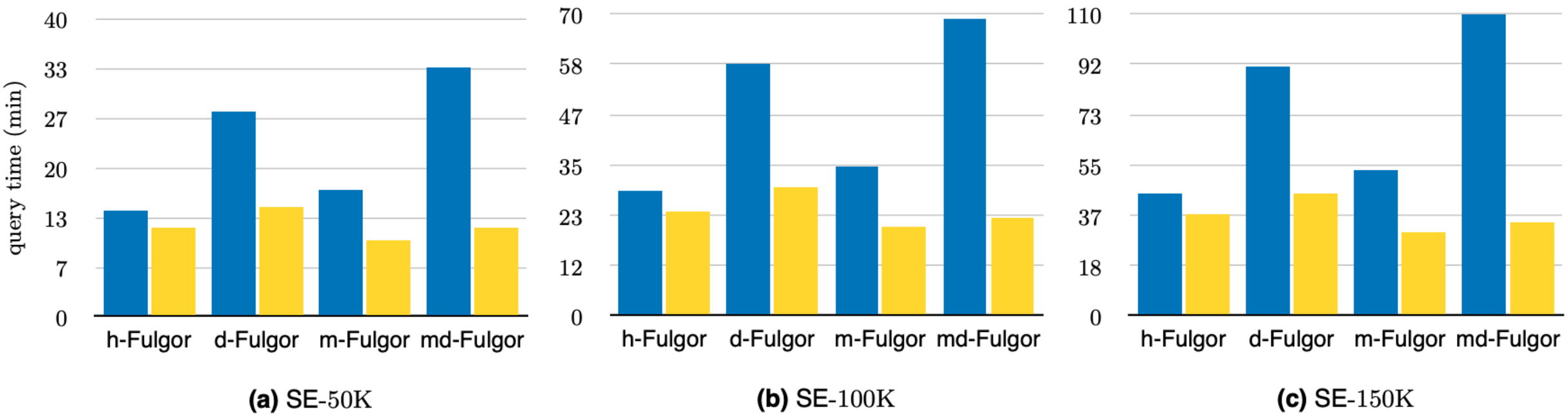
## Fulgor (v4) https://github.com/jermp/fulgor/releases/tag/v4.0.0 (2025)

| Dataset | Mapping rate | h-Fulgor | | | d-Fulgor | | | m-Fulgor | | | md-Fulgor | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | before | after | GB | before | after | GB | before | after | GB | before | after | GB |
| EC | 98.99 | 2:12 | **2:10** | 1.67 | 4:52 | **2:29** | 0.78 | 3:08 | **1:32** | 0.73 | 6:07 | **1:41** | 0.57 |
| SE-5K | 89.49 | 1:14 | **1:10** | 0.80 | 1:54 | **1:44** | 0.41 | 1:25 | **1:09** | 0.37 | 2:10 | **1:21** | 0.32 |
| SE-10K | 89.71 | 2:29 | **2:20** | 2.06 | 4:14 | **2:54** | 0.90 | 2:56 | **2:07** | 0.77 | 4:55 | **2:30** | 0.65 |
| SE-50K | 91.25 | 14:05 | **12:00** | 18.24 | 27:25 | **14:50** | 5.82 | 17:00 | **10:10** | 3.64 | 33:25 | **11:50** | 2.95 |
| SE-100K | 91.41 | 29:00 | **24:00** | 42.40 | 58:10 | **29:50** | 11.58 | 34:40 | **20:30** | 6.08 | 1:09:00 | **22:50** | 4.63 |
| SE-150K | 91.52 | 44:30 | **37:00** | 70.55 | 1:31:00 | **44:20** | 18.55 | 53:00 | **30:20** | 8.29 | 1:50:00 | **33:50** | 6.29 |
| GB | 92.91 | **1:10** | **1:10** | 36.01 | **1:00** | 1:26 | 28.25 | 1:09 | **1:02** | 29.79 | 1:03 | **1:02** | 26.88 |

Full-intersection query mode; 16 processing threads; Max RSS in GB; output of `/usr/bin/time`.

# Faster query times!

## Fulgor (v4) https://github.com/jermp/fulgor/releases/tag/v4.0.0 (2025)



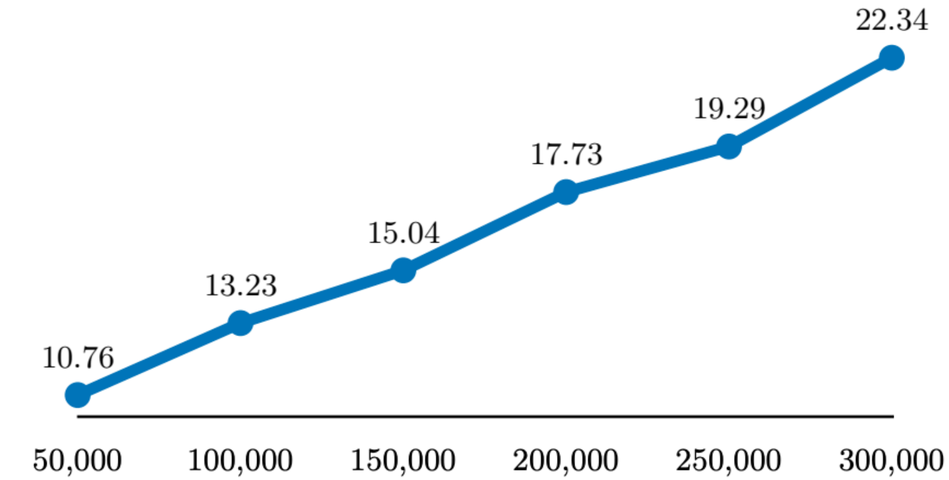

Full-intersection query mode; 16 processing threads; Max RSS in GB; output of `/usr/bin/time`.

# Faster query times — Intuition (for intersection)

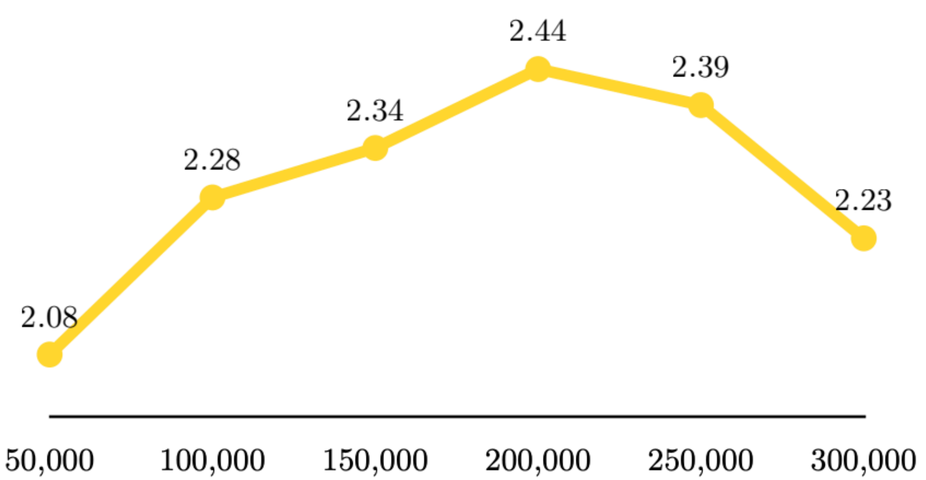
- If most color sets being intersected are **super-dense**: compute the union of complement sets, say $U$, and iterate over the complement of $U$.
- If color sets are relative to the **same representative color set**, their intersection can be computed **only using their symmetric set differences** (that are much shorter).
- Meta and partial color sets yield a **two-level** index organisation:
  we can first intersect meta color sets and only compute intersections of partial color sets for the common partitions only.
- Other tricks for (threshold-based) union of color sets apply.
- Check our new paper https://jermp.github.io/assets/pdf/papers/WABI2025.pdf out!

# Repetition-aware improvement factors

- Results obtained by indexing Salmonella Enterica genomes from the “AllTheBacteria” collection [Hunt et al., 2024], https://allthebacteria.org




**(a)** Space improvement




**(b)** Query time improvement

Full-intersection query mode; 16 processing threads

# Conclusions

- SSHash to obtain an **efficient map from k-mers to unitigs** (2022).
- Permute unitigs in color set order to enable a **space-efficient mapping from unitigs to color sets** (2023).
- Color sets can be factorized into “patterns” that **capture their repetitiveness** (2024): two paradigms — **horizontal and vertical partitioning** — that can be combined.
- These patterns can also be exploited to achieve faster query times (2025).

# Conclusions

- SSHash to obtain an **efficient map from k-mers to unitigs** (2022).
- Permute unitigs in color set order to enable a **space-efficient mapping from unitigs to color sets** (2023).
- Color sets can be factorized into “patterns” that **capture their repetitiveness** (2024): two paradigms — **horizontal and vertical partitioning** — that can be combined.
- These patterns can also be exploited to achieve faster query times (2025).

## A special thank to my co-authors!

**Rob Patro**

**Alessio Campanelli**

# Next goals

- Index more and more and make the indexes available to the community, .e.g.
- the entire “AllTheBacteria” https://allthebacteria.org
- Logan, https://github.com/IndexThePlanet/Logan
- All NCBI viruses https://www.ncbi.nlm.nih.gov/labs/virus/vssi
- etc.
- **Sharding to external memory** seems inevitable in the future.

Taken from https://www.biorxiv.org/content/10.1101/2024.03.08.584059v3.full