Indexing the (compacted) colored de Bruijn graph

UNIVERSITY OF MARYLAND

Scaling up fast reference-based indices

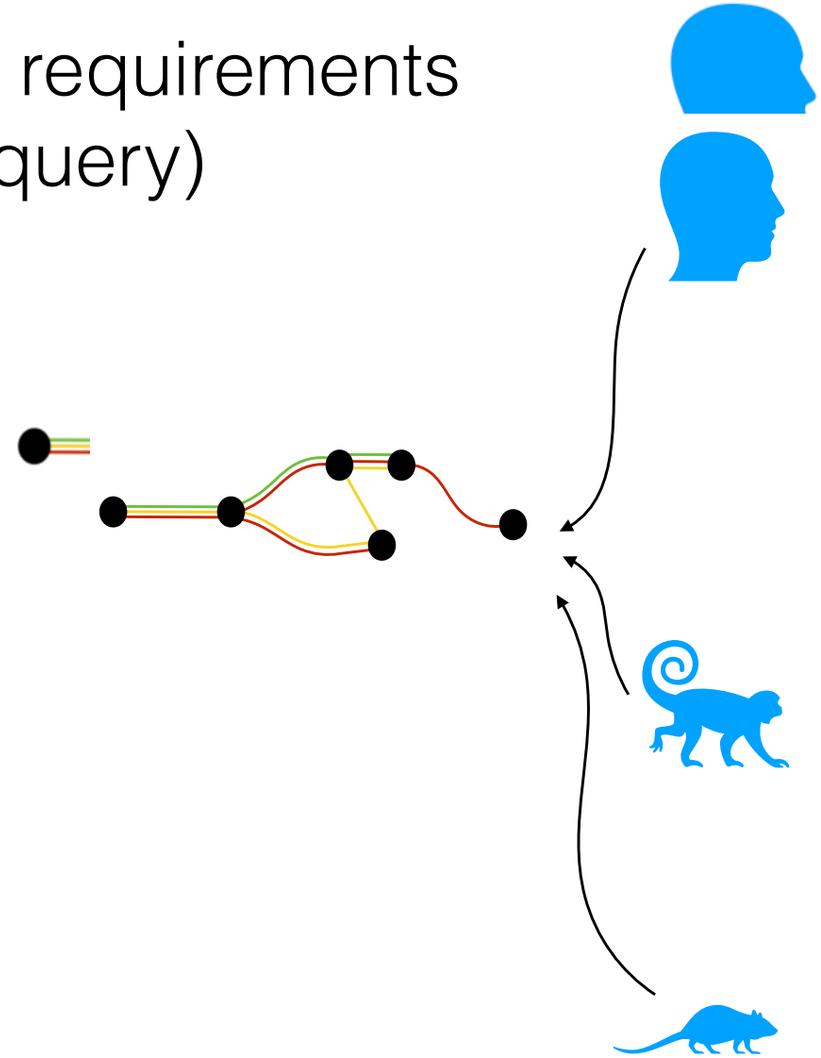
Motivation: Indices used in “ultra-fast” mapping approaches are typically very memory hungry. This is **OK** for transcriptome mapping, but **not scalable** to genomic, metagenomic, pangenomic or population mapping.

Goal: Develop an index with practical memory requirements that maintains the desirable performance (i.e. query) characteristics of the “ultra-fast” indices.

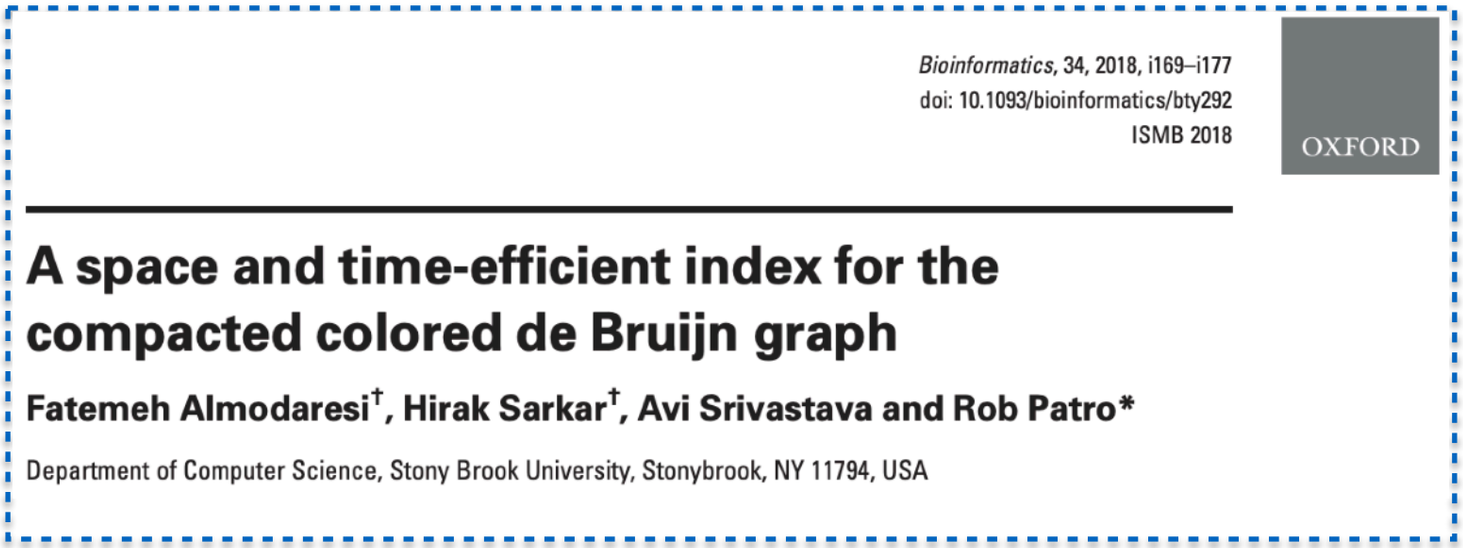
Compacted colored de Bruijn graph (ccdBG)

Built over 1 or more genomes / sequence collections

Index makes use of minimum perfect hashing succinct bit vector representations and (optionally) a new sampling scheme



Pufferfish: An efficient index for the ccdBG



Bioinformatics, 34, 2018, i169–i177
doi: 10.1093/bioinformatics/bty292
ISMB 2018

OXFORD

A space and time-efficient index for the compacted colored de Bruijn graph

Fatemeh Almodaresi†, Hirak Sarkar†, Avi Srivastava and Rob Patro*

Department of Computer Science, Stony Brook University, Stonybrook, NY 11794, USA

Appeared at **ISMB 2018**

- The past decade has largely been dominated by SA/BWT/FM-index-based approaches to reference sequence indexing (e.g. Bowtie, BWA, BWA-MEM, Bowtie2, STAR, etc.)
- There has been a renaissance of sorts for hash-based indexing (deBGA, Brownie, kallisto, mashmap, minimap & minimap2, etc.)
- Pufferfish goes the hashing-based route; *with a twist*.
- Not considering generalized path indices on general seq (e.g. GCSA2 (VG), HISAT2). Interesting, but a different problem.

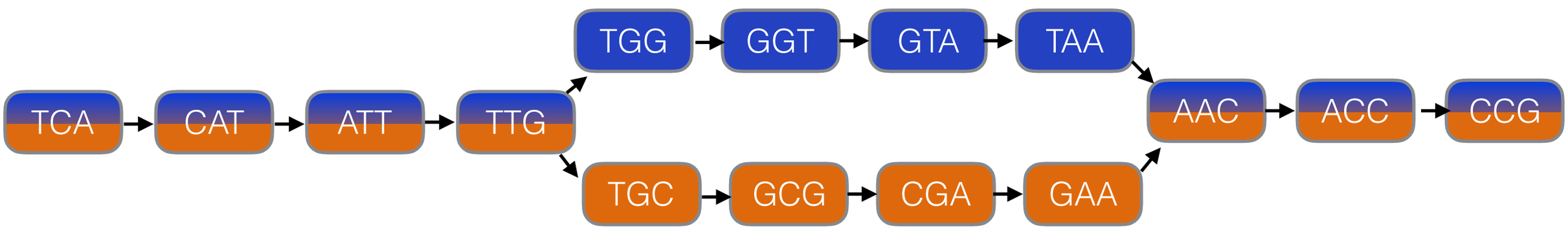
https://github.com/COMBINE-lab/pufferfish

Recall the “colored” de Bruijn Graph

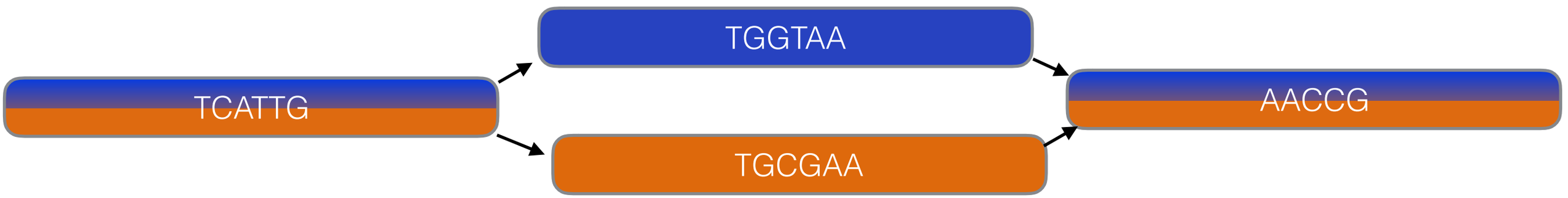
Nodes are k-mers (here k=3)

Edges exist between nodes that overlap by k-1 (in the input)*

Colors encode “origin” of k-mers (e.g., references where they exist)



compacted colored de Bruijn graph



Example from : https://algolab.files.wordpress.com/2016/10/chikhi-milan-18nov.pdf

There are multiple related (but distinct) definitions of the dBG in practice. We adopt the **edge-explicit** version.

The compacted colored dBG as a sequence index

- **Key idea**: represent a collection of sequences using the colored de Bruijn graph (dBG) (Iqbal ’12).

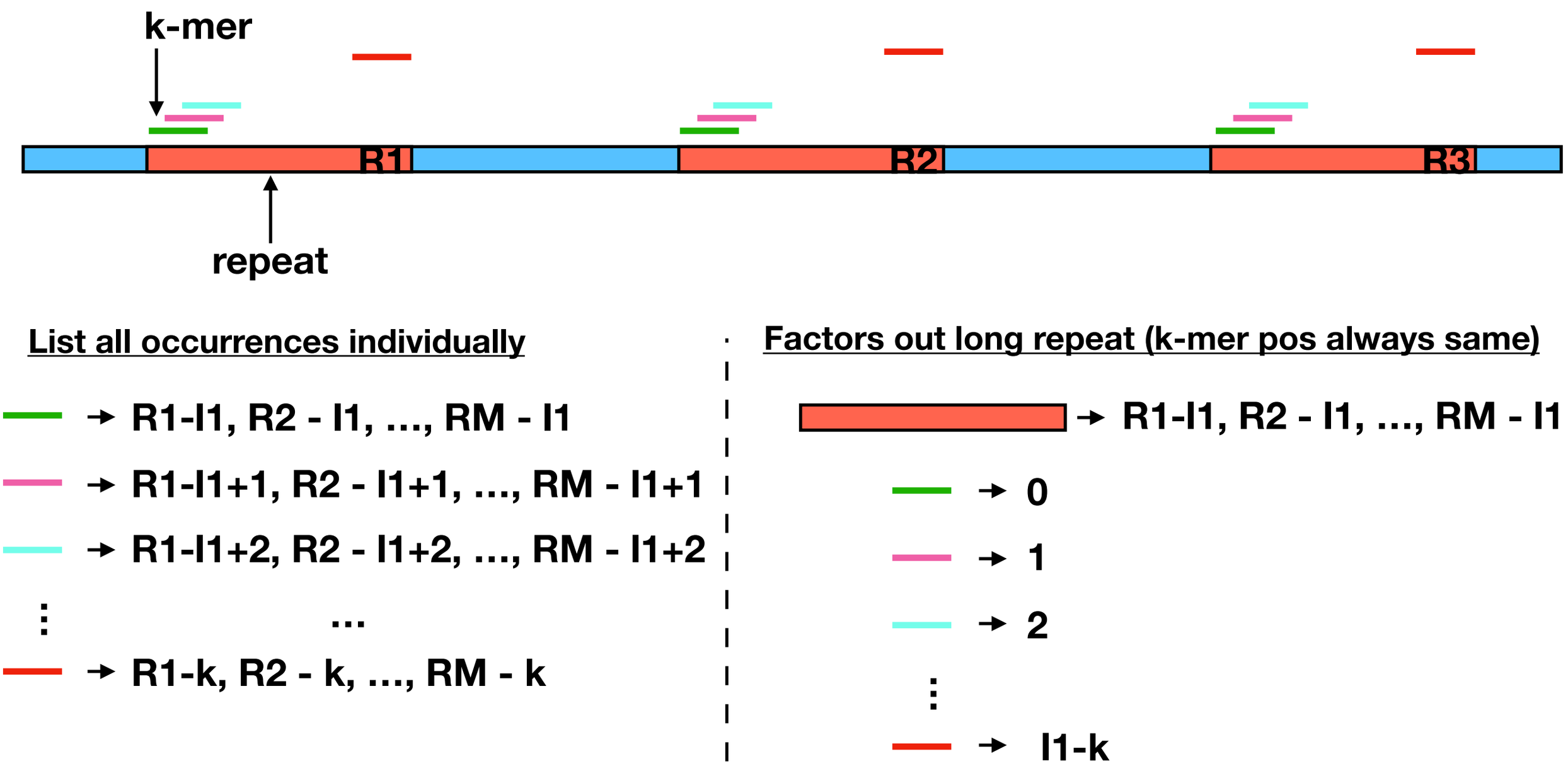
- Each color is an input reference (e.g. genome or transcript).

- Use the compacted colored dBG as an index for reference-based sequence search.

- Redundant sequences (repeats) are implicitly collapsed. **Why is this potentially *much* better than a naive hash?**

The compacted colored dBG as a sequence index

- Redundant sequences (repeats) are implicitly collapsed. **Why is this potentially *much* better than a naive hash?**



The cdBG removes redundancy by providing an extra level of indirection

The compacted colored dBG as a sequence index

- Redundant sequences (repeats) are implicitly collapsed. **Why is this potentially *much* better than a naive hash?**

Still, the biggest problem for these schemes, in practice, is *memory usage*

The main culprit is the hash table itself

The cdBG removes redundancy by providing an extra level of indirection

Recall: Minimum Perfect Hashing

Minimum Perfect Hash Function (MPHF)

$$\mathcal{K} \subseteq \mathcal{U}, \quad f : \mathcal{K} \rightarrow \mathbb{N}^{+}$$

if $x \in \mathcal{K}$ **then** $f(x) \in [1, \left|\mathcal{K}\right|]$

if $x \in \mathcal{U} \backslash \mathcal{K}$ **then** $f(x) \in [1, \left|\mathcal{U}\right|]$ (Like “false positives”)

f is a complete, injective function from $\mathcal{K} \rightarrow [1, \left|\mathcal{K}\right|]$

Best methods achieve ~2.1 bits/key ***regardless of key size***

Use BBHash :)

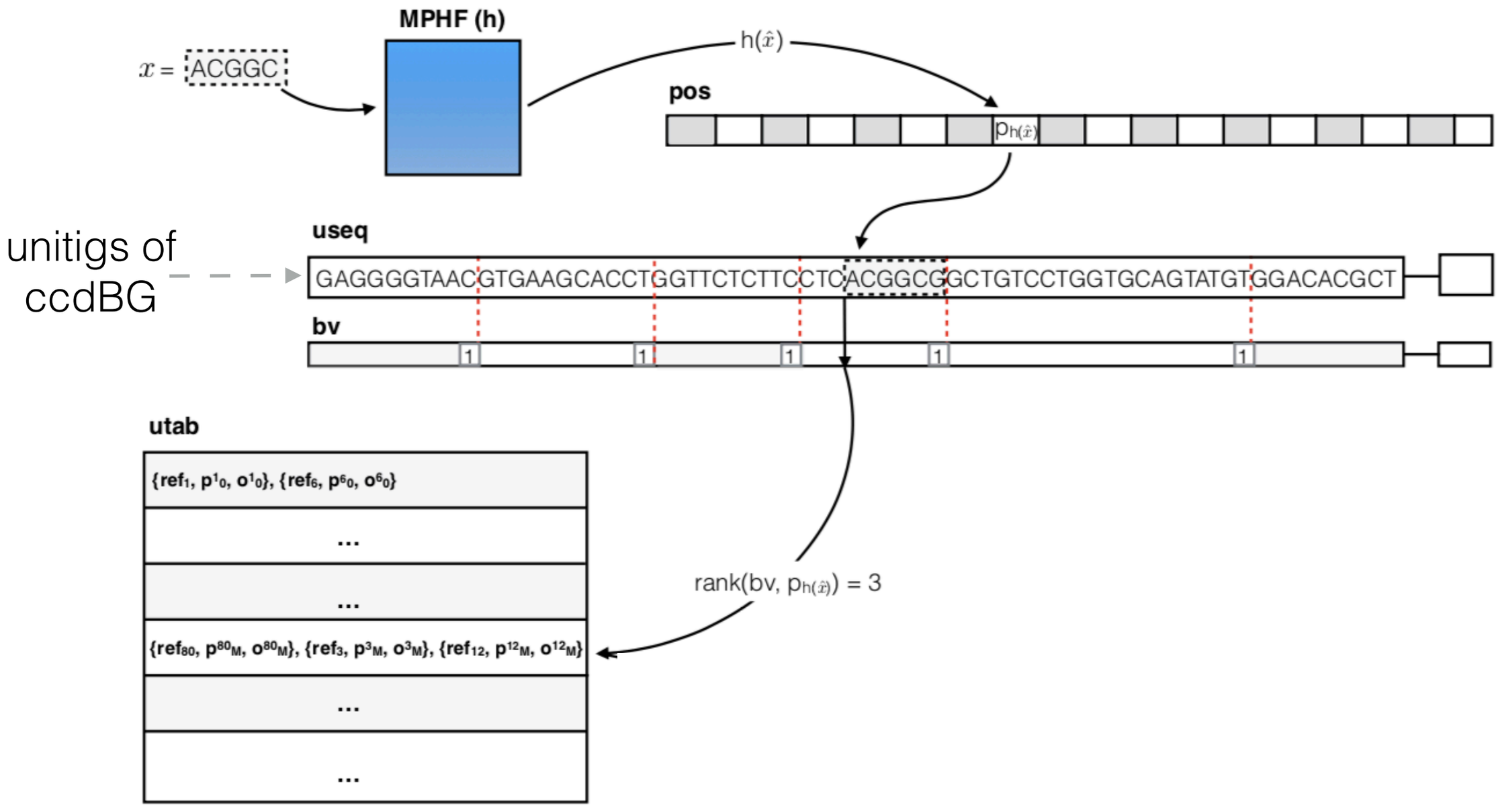
Fast and scalable minimal perfect hashing for massive key sets

Antoine Limasset[1], Guillaume Rizk[1], Rayan Chikhi[2], and Pierre Peterlongo[1]

1 IRISA Inria Rennes Bretagne Atlantique, GenScale team, Campus de Beaulieu 35042 Rennes, France
2 CNRS, CRIStAL, Université de Lille, Inria Lille - Nord Europe, France

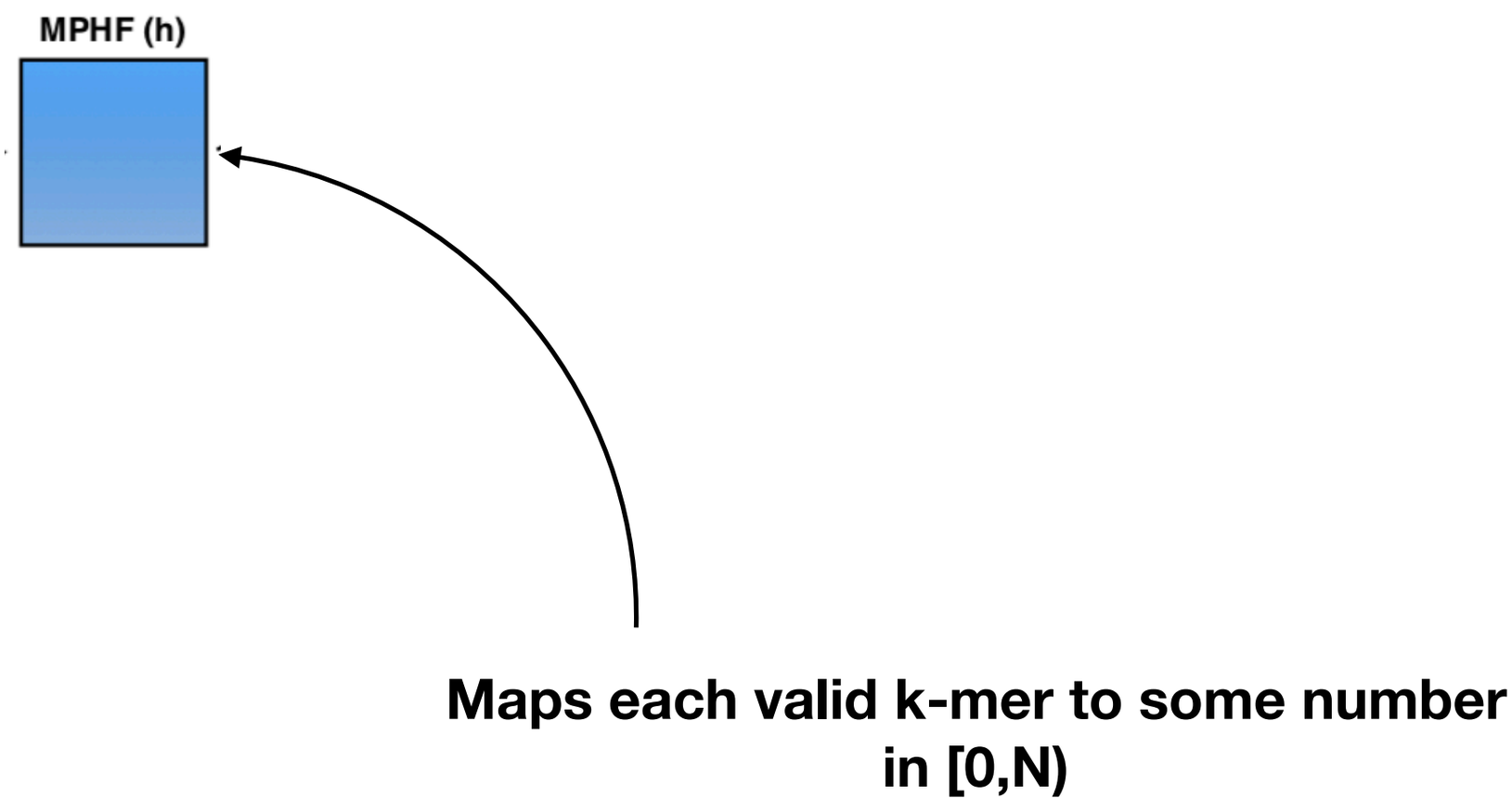
https://github.com/rizkg/BBHash

The **dense** Pufferfish index



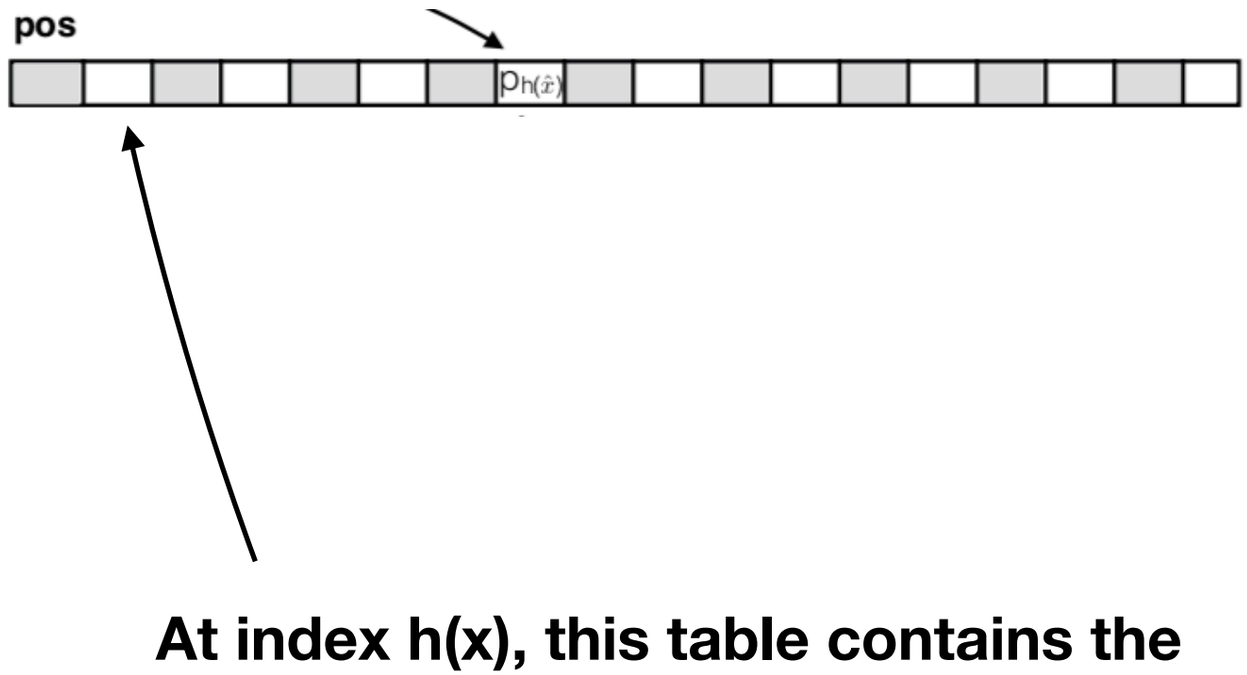
Optionally: explicit edge table, equivalence class table

The **dense** Pufferfish index



Optionally: explicit edge table, equivalence class table

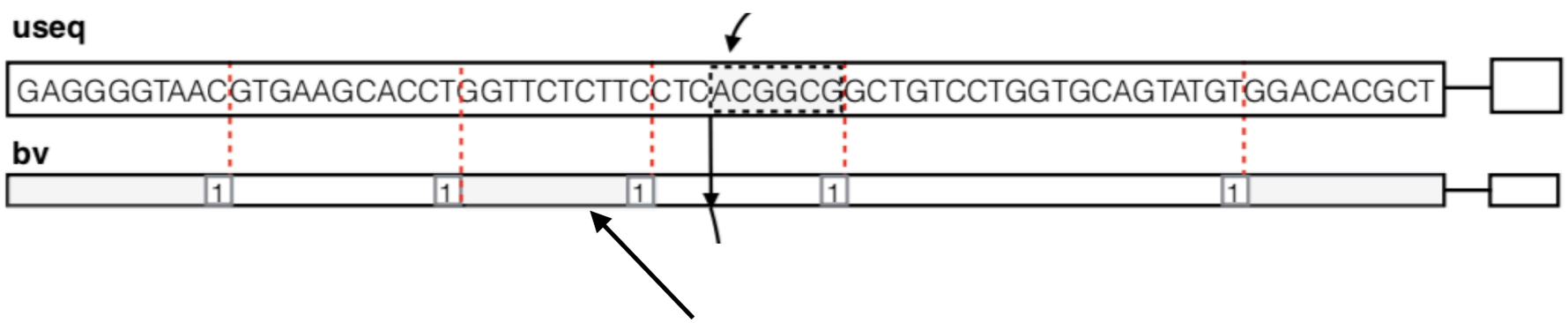
The **dense** Pufferfish index



At index h(x), this table contains the position, in the list of unitigs, of this k-mer

Optionally: explicit edge table, equivalence class table

The **dense** Pufferfish index

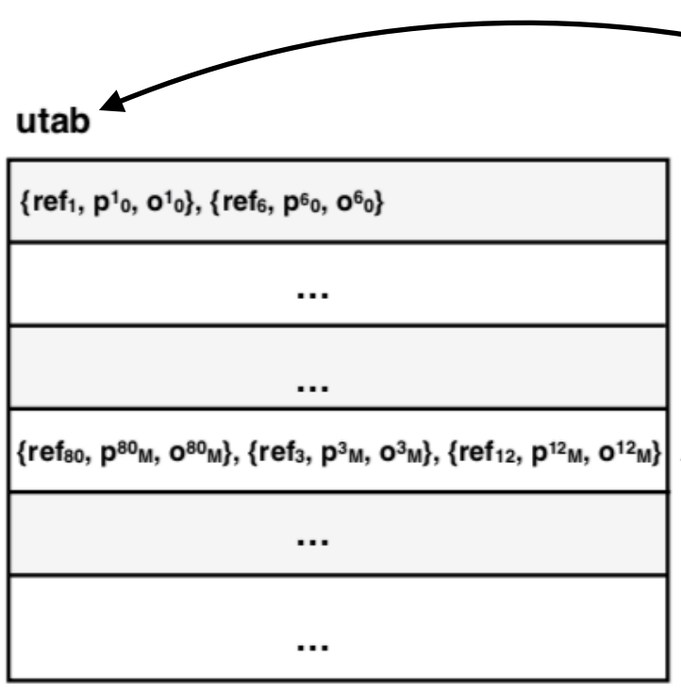


- **useq contains the uniting sequences concatenated together**

- **bv is a boundary vector that records a 1 at the end of each uniting, and a 0 elsewhere**

Optionally: explicit edge table, equivalence class table

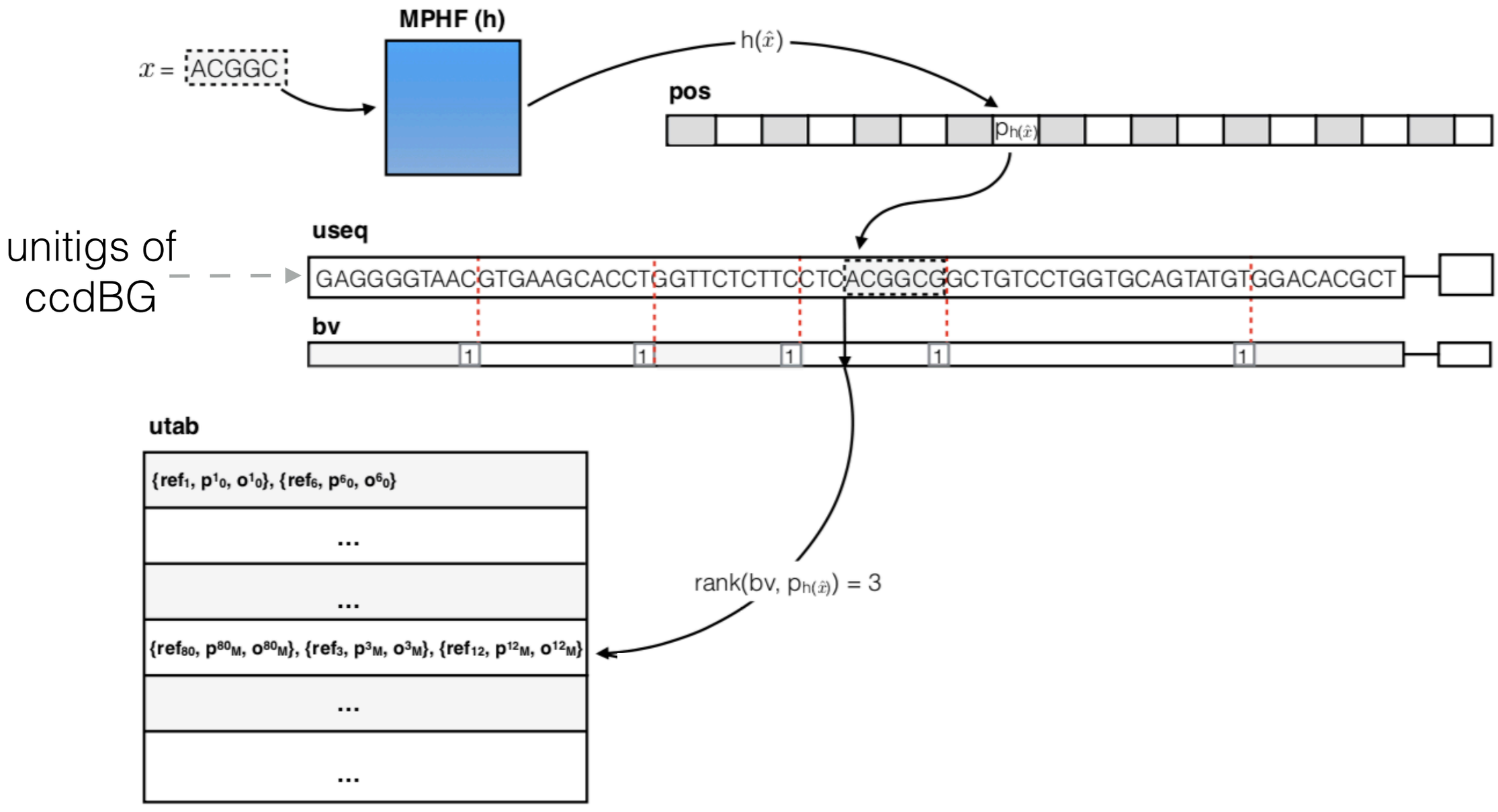
The **dense** Pufferfish index



Records, for each uniting, the list of references, positions and orientations in which it occurs

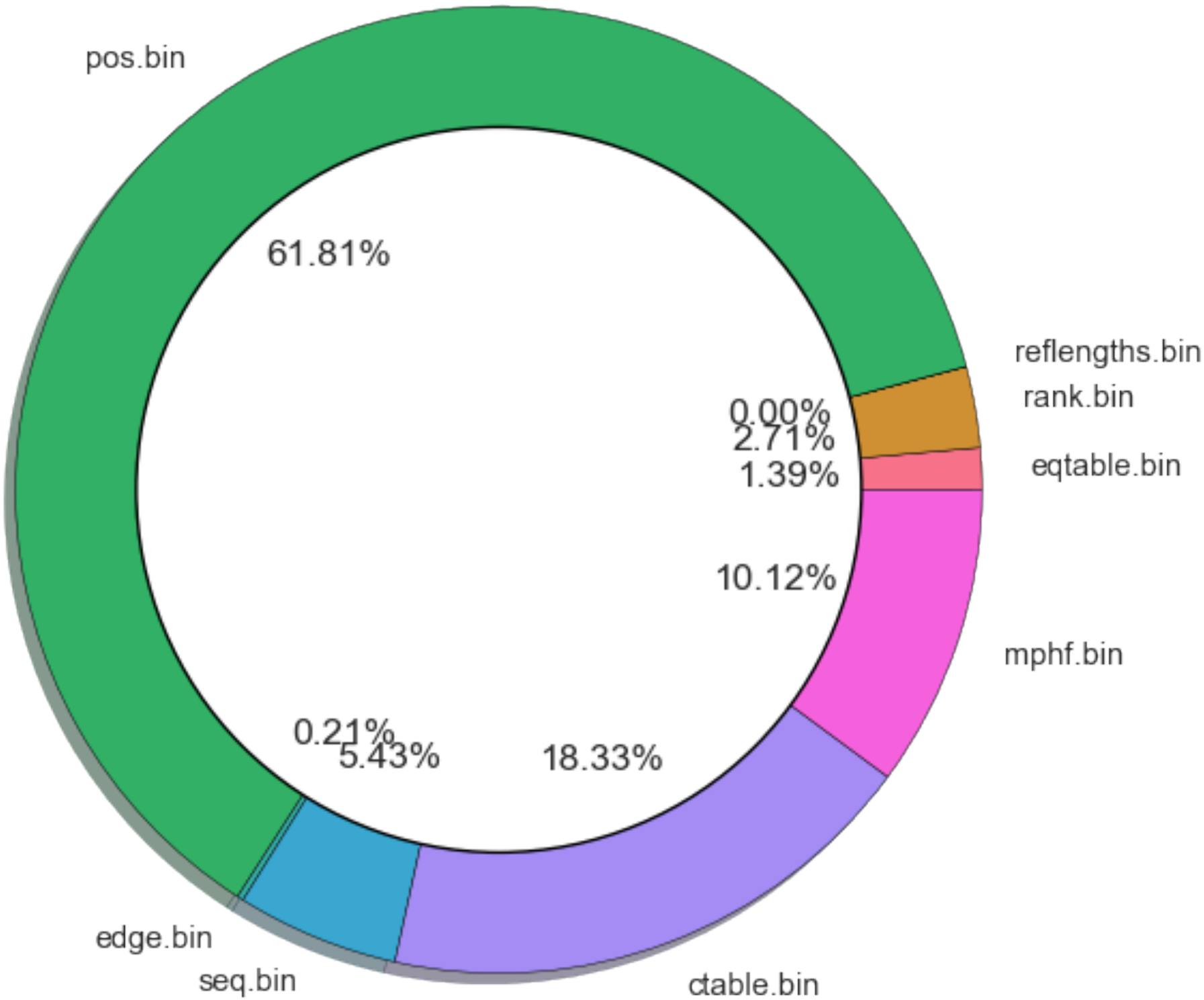
Optionally: explicit edge table, equivalence class table

The **dense** Pufferfish index

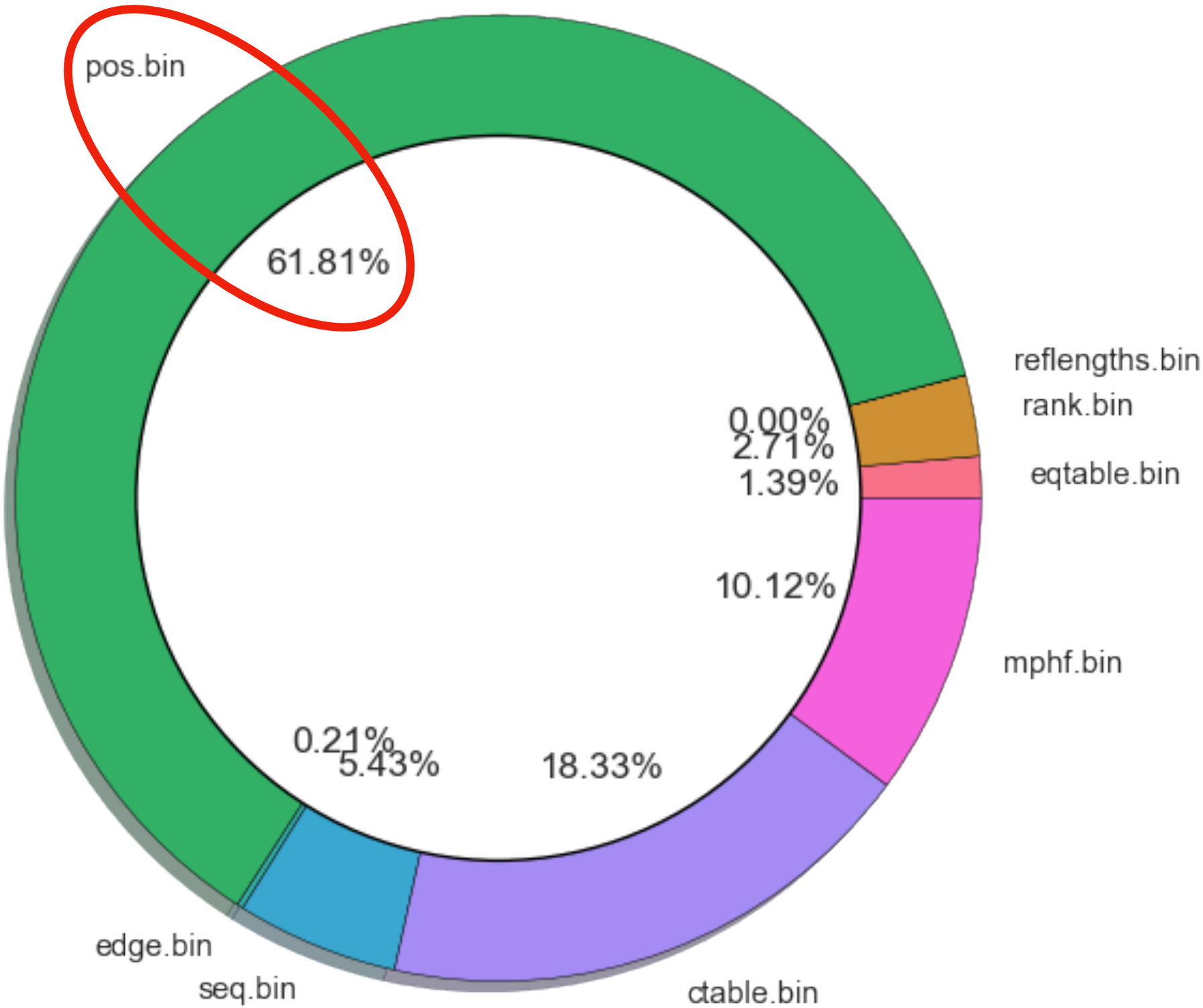


Optionally: explicit edge table, equivalence class table

Who’s the culprit?

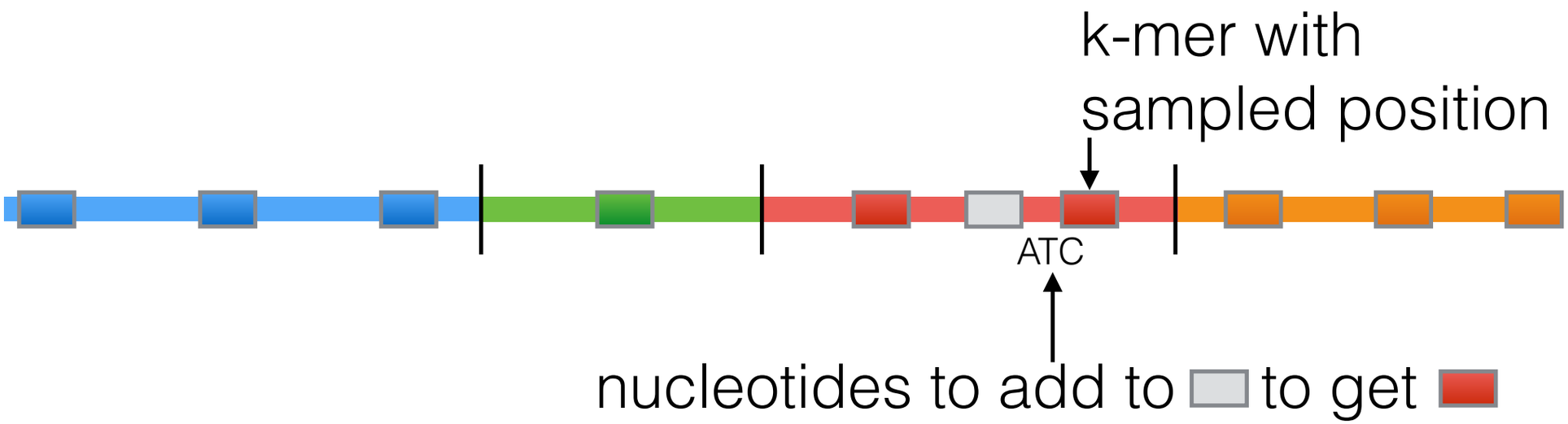


Who’s the culprit?



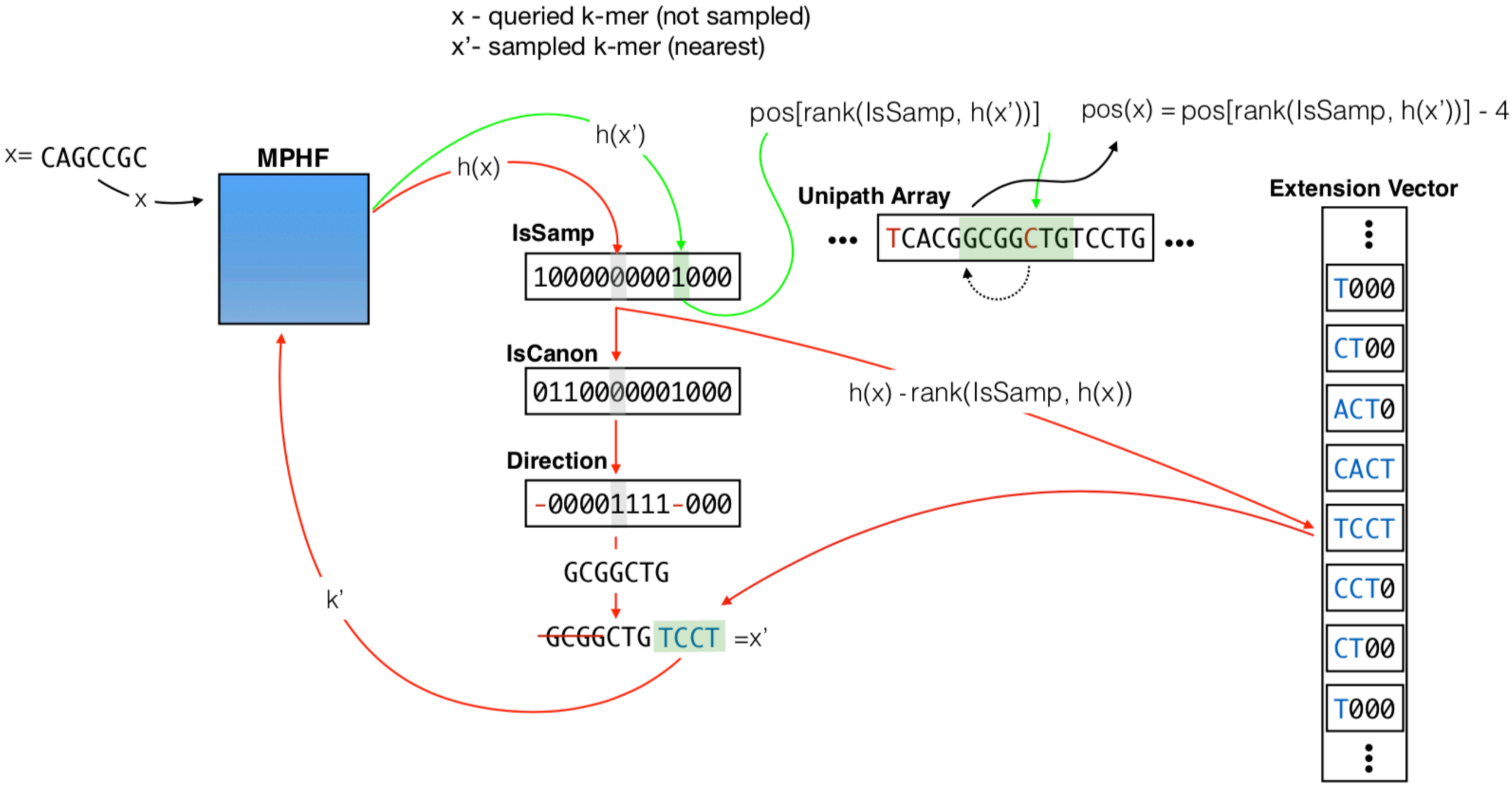
The **sparse** Pufferfish index

In large indices, the position table *dominates* index size



Intuition: Successors and predecessors in unipaths are *globally unique*, instead of storing position information for all k-mers, store positions only at sampled “landmarks” and say how to navigate to these landmarks (similar to bi-directional sampling in the FM-index).

The **sparse** Pufferfish index (in detail)

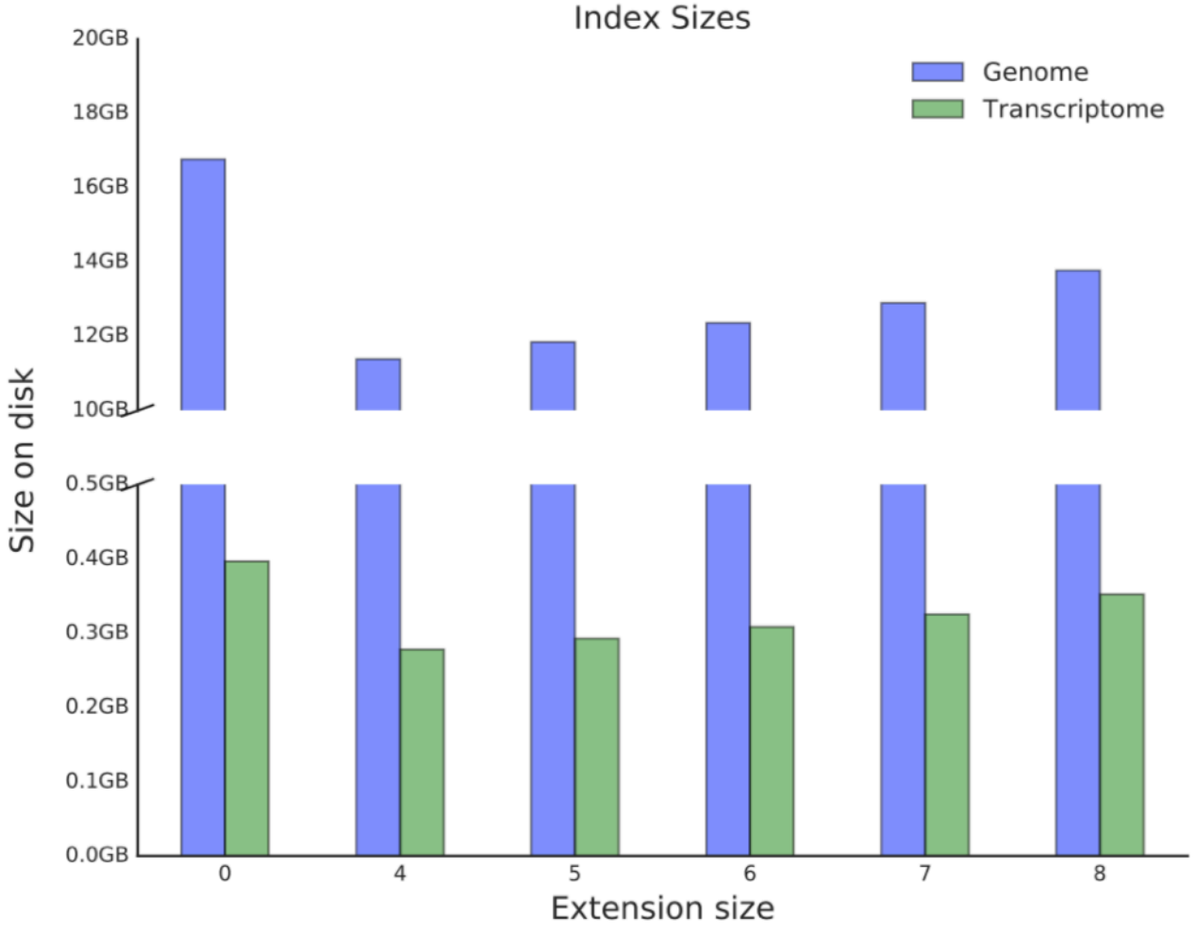


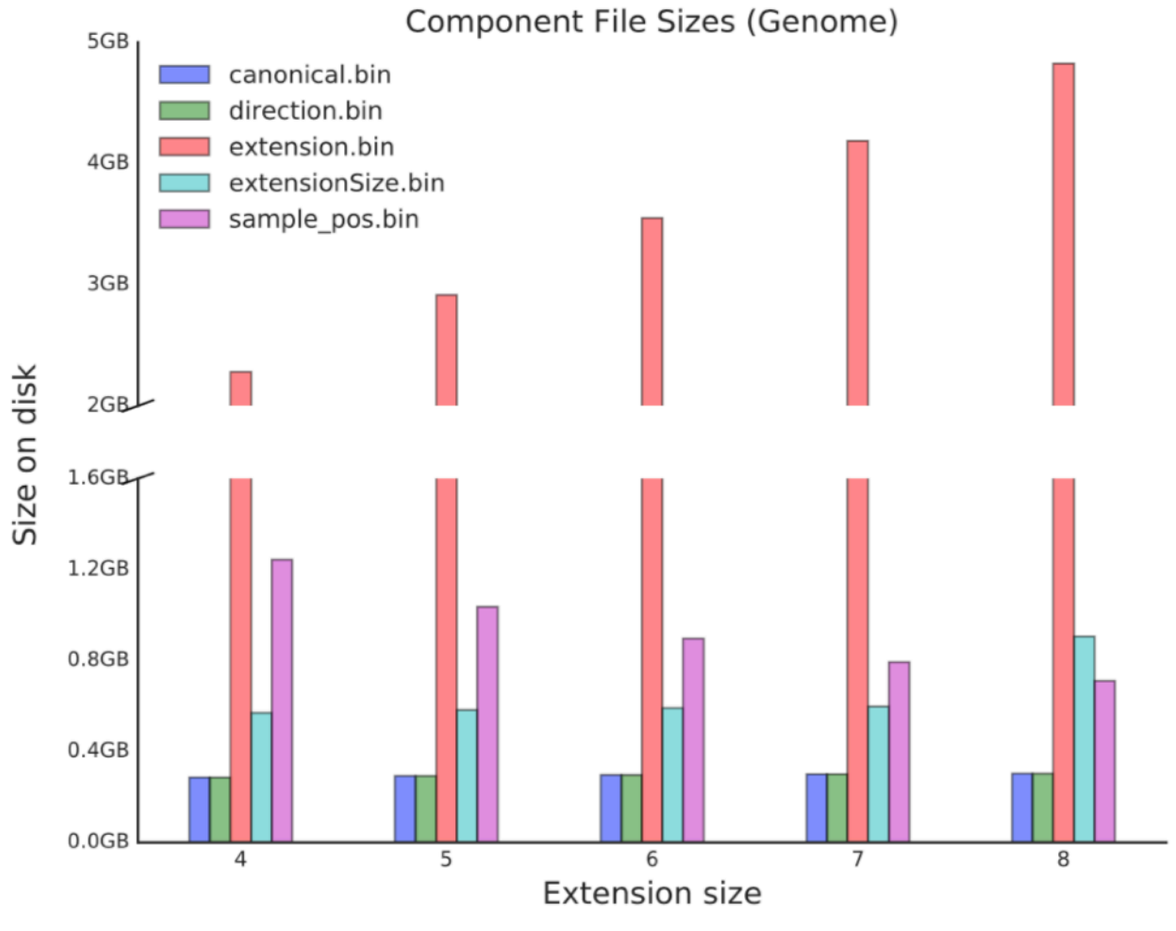
What sampling factor is right?

Tradeoff : Sparser sampling → less space but slower lookup

Fastest : Sampling factor s > 2·e+1 (Still a range of sizes)

Smallest : Extension size = 1, sampling = s





Index space & K-mer query time

Space of index + query in RAM

Tool	Memory (MB)		
	Human Transcriptome	Human Genome	Bacterial Genome
BWA	308	4,439	27,535
kallisto	3,336	110,464	232,353
pufferfish dense	454	17,684	41,532
pufferfish sparse	341	12,533	30,565

#Li, H. (2013). Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. arXiv Preprint arXiv:1303.3997.

^Bray, N. L., Pimentel, H., Melsted, P., and Pachter, L. (2016). Near-optimal probabilistic RNA-seq quantification. Nature Biotechnology, 34(5), 525–527.

Index space & K-mer query time

Time to look up all fixed-length substrings in an experiment

Tool	Time (h:m:s)		
	Human Transcriptome	Human Genome	Bacterial Genome
BWA	0:17:35	0:50:31	0:14:05
kallisto	0:02:01	0:19:11	0:22:25
pufferfish dense	0:02:46	0:10:37	0:06:03
pufferfish sparse	0:08:34	0:22:11	0:08:26
# queries:	747,842,900	7,508,576,020	509,143,360

Pufferfish summary (part 1)

- To keep memory usage reasonable, we have to be quite careful about our hashing-based schemes.

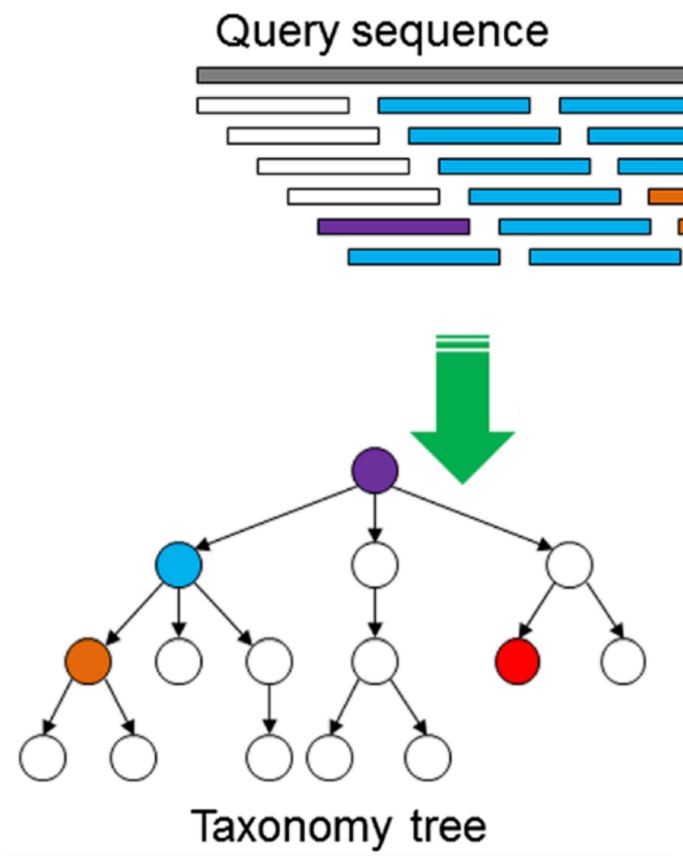
- The dense pufferfish index strikes a good balance between index space and raw query speed.

- At a constant factor (though not asymptotic) cost, index size is tunable with our sampling scheme.

- At least for fixed-length patterns, a good hashing approach can be *much faster* than (still asymptotically-optimal) full-text indexes.

An example application of Pufferfish

- Taxonomic read classification — for each read, assign it to the taxon (strain, species, genus) from which we think it derived. Related to, ***but distinct from***, taxonomic abundance estimation.



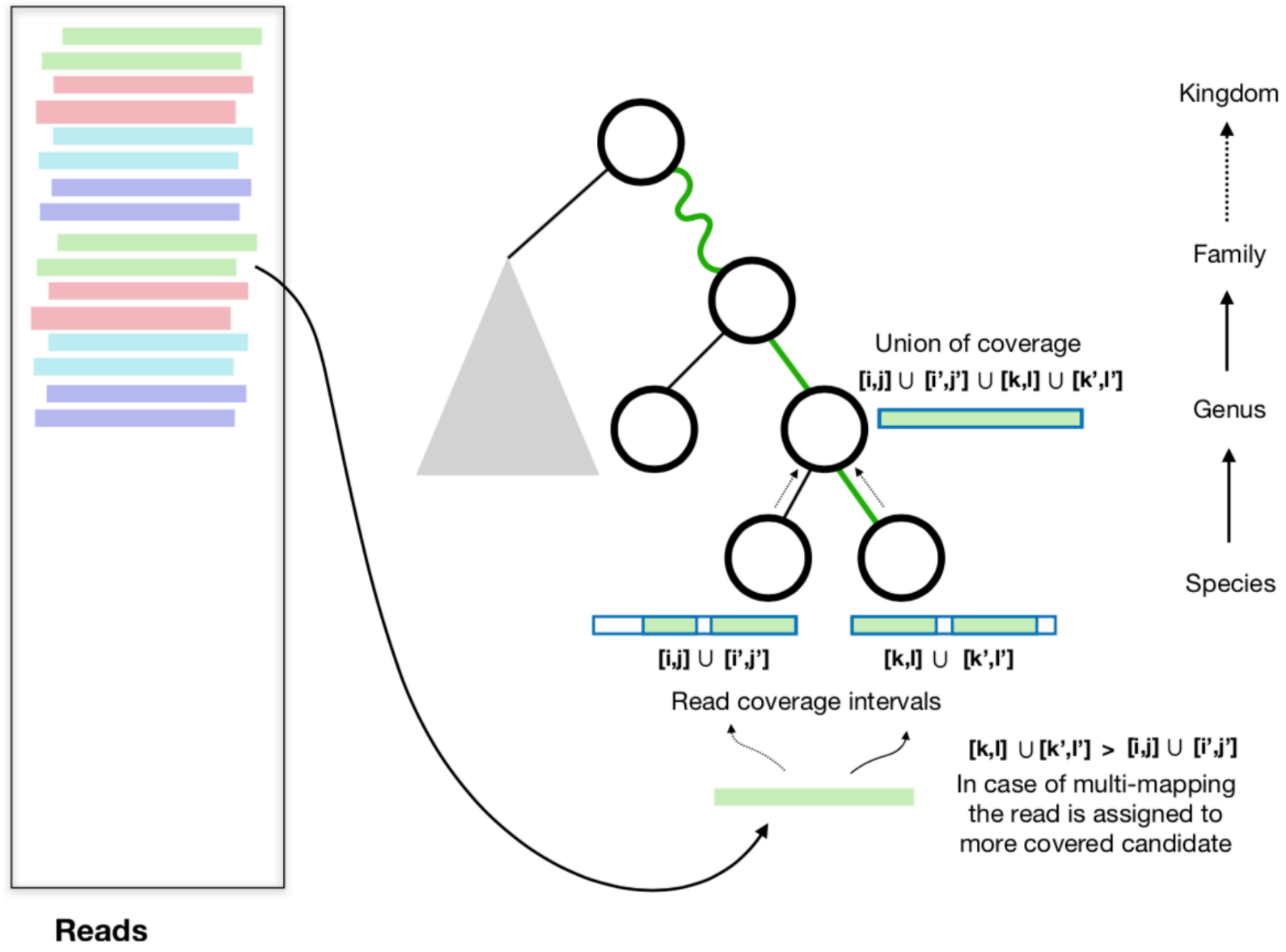
Figures adapted from: Wood, D.E. and Salzberg, S.L., 2014. Kraken: ultrafast metagenomic sequence classification using exact alignments. *Genome biology*, *15*(3), p.R46.

Pufferfish taxonomic assignment

We adopt what is essentially the algorithm of *Kraken**, but replace k-mer counting with lightweight mapping.

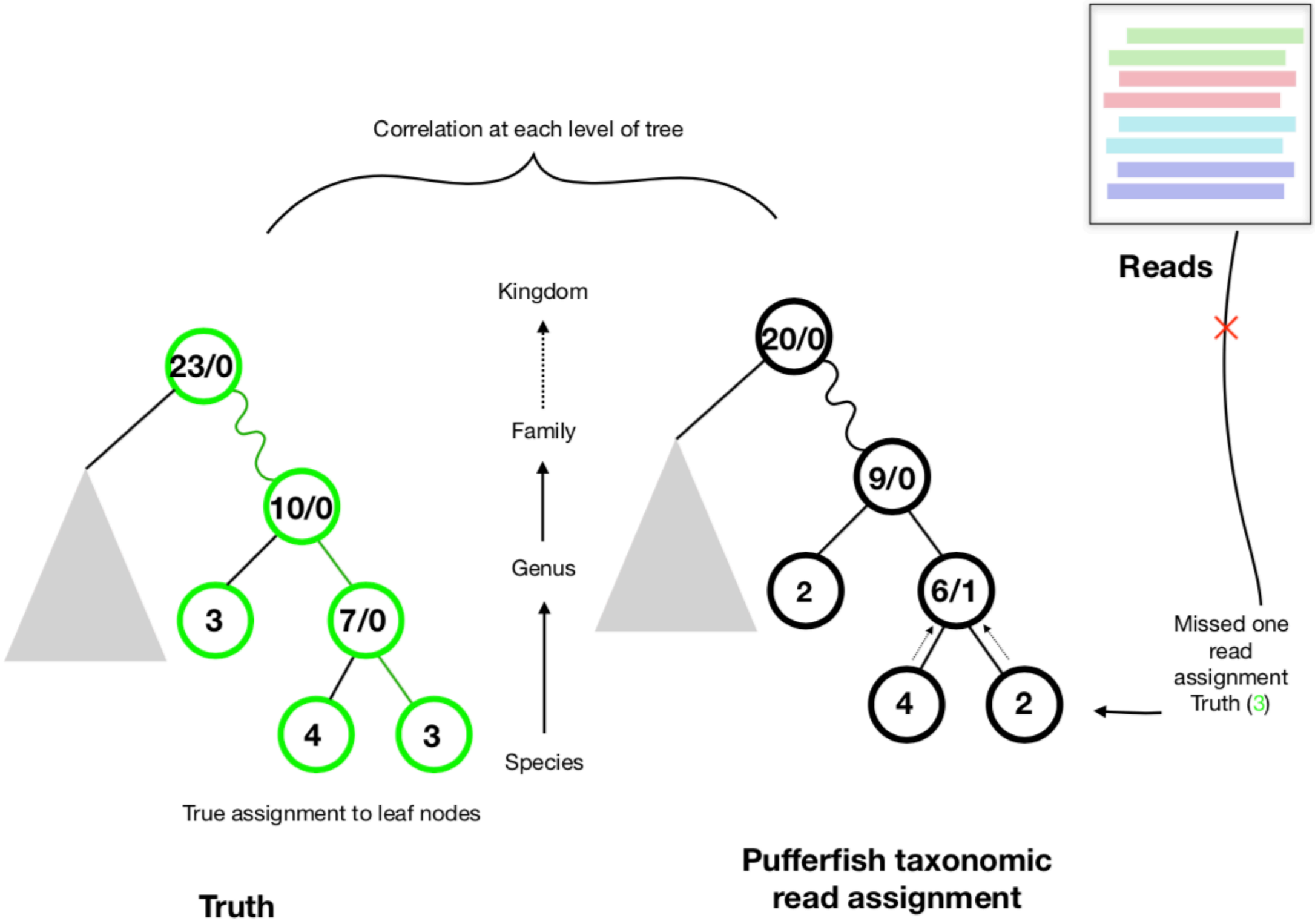
This *enforces positional & orientation consistency* of matches

- Score all root-to-leaf (RTL) paths
- Assign read to leaf of highest-scoring path
- In case of tie, assign read to LCA of all highest-scoring paths.

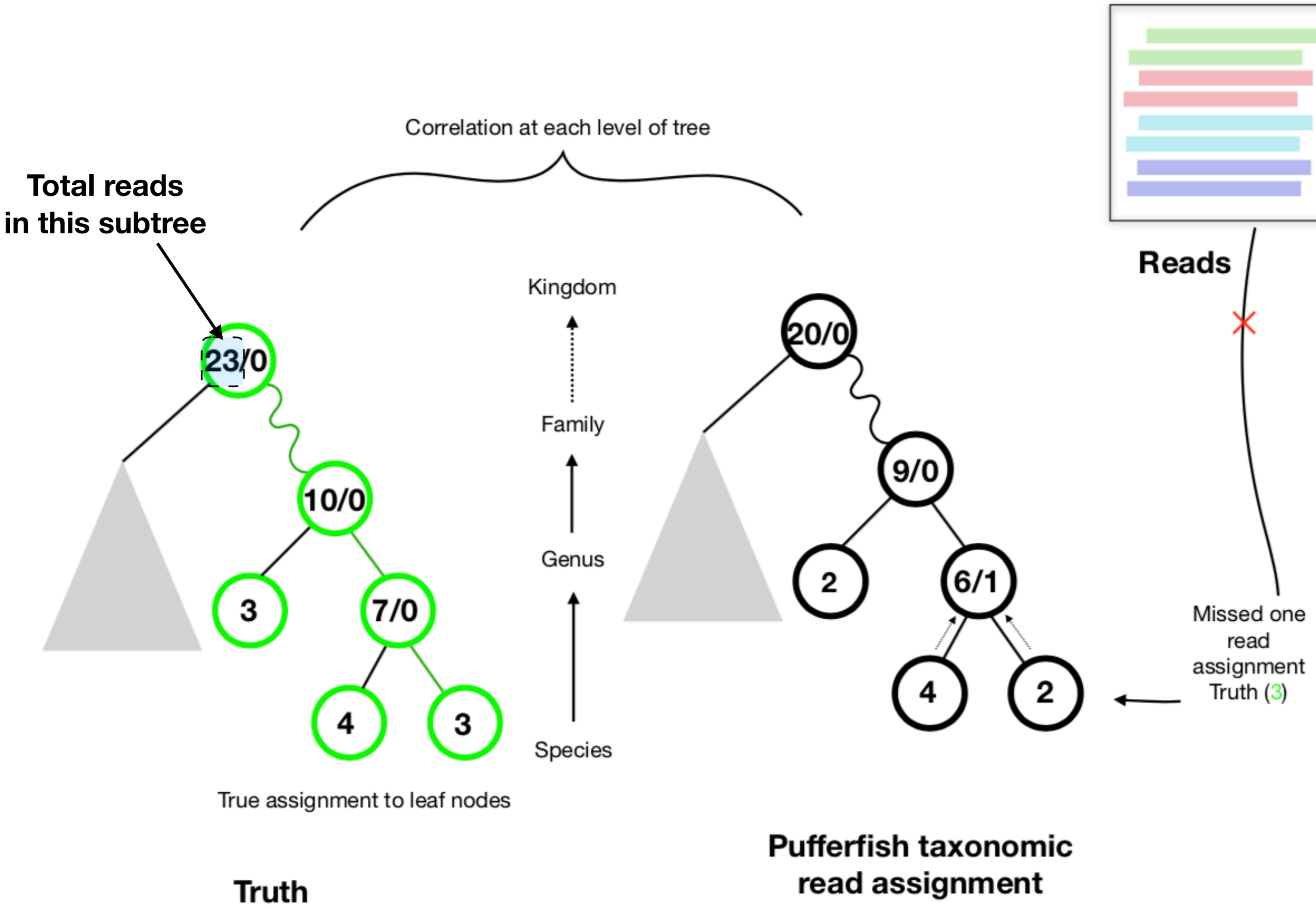


*Wood, D.E. and Salzberg, S.L., 2014. Kraken: ultrafast metagenomic sequence classification using exact alignments. *Genome biology*, *15*(3), p.R46.

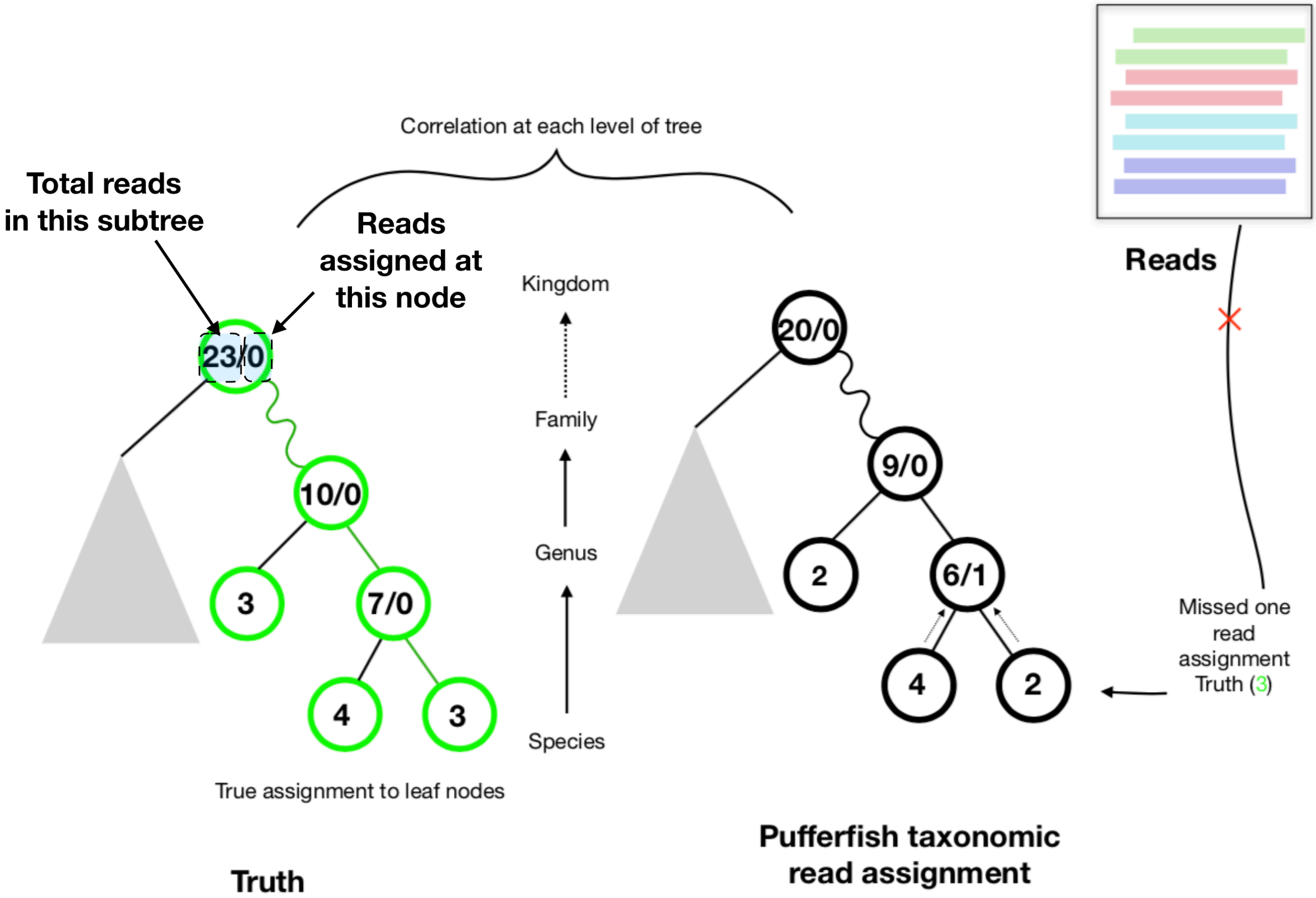
“Whole taxonomy” accuracy assessment



“Whole taxonomy” accuracy assessment

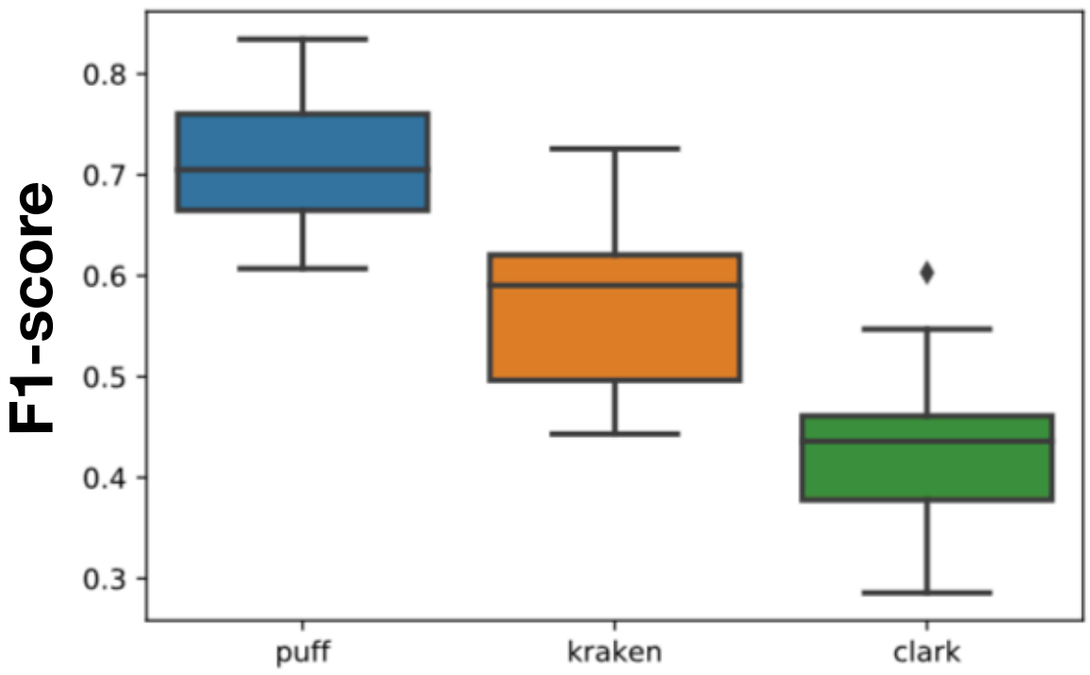


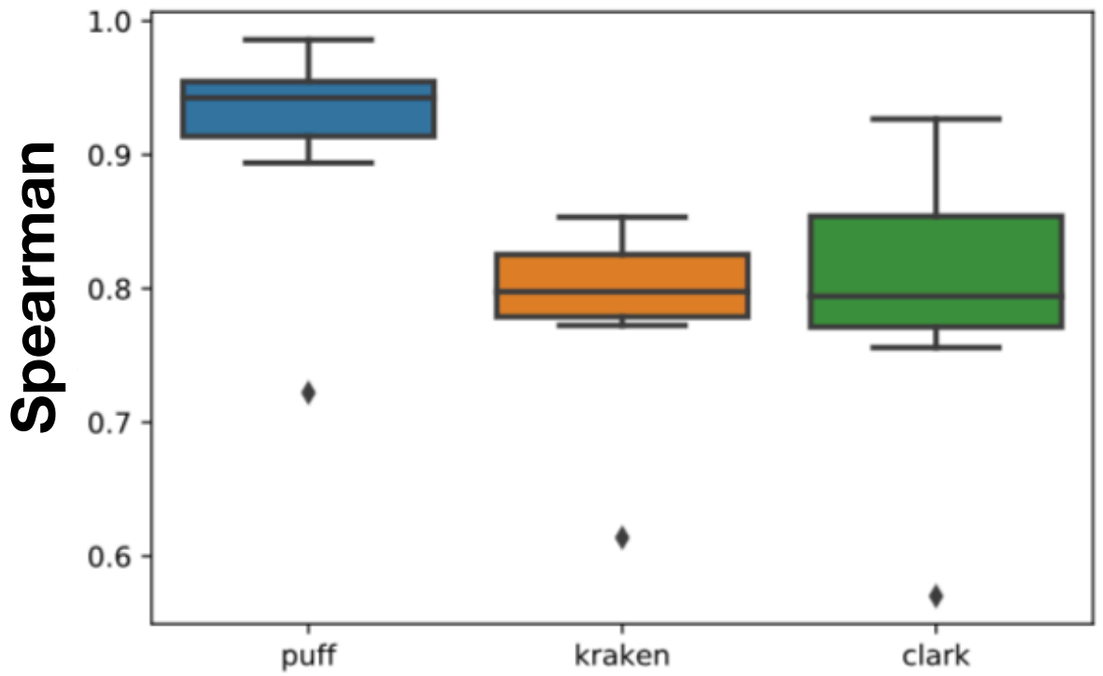
“Whole taxonomy” accuracy assessment



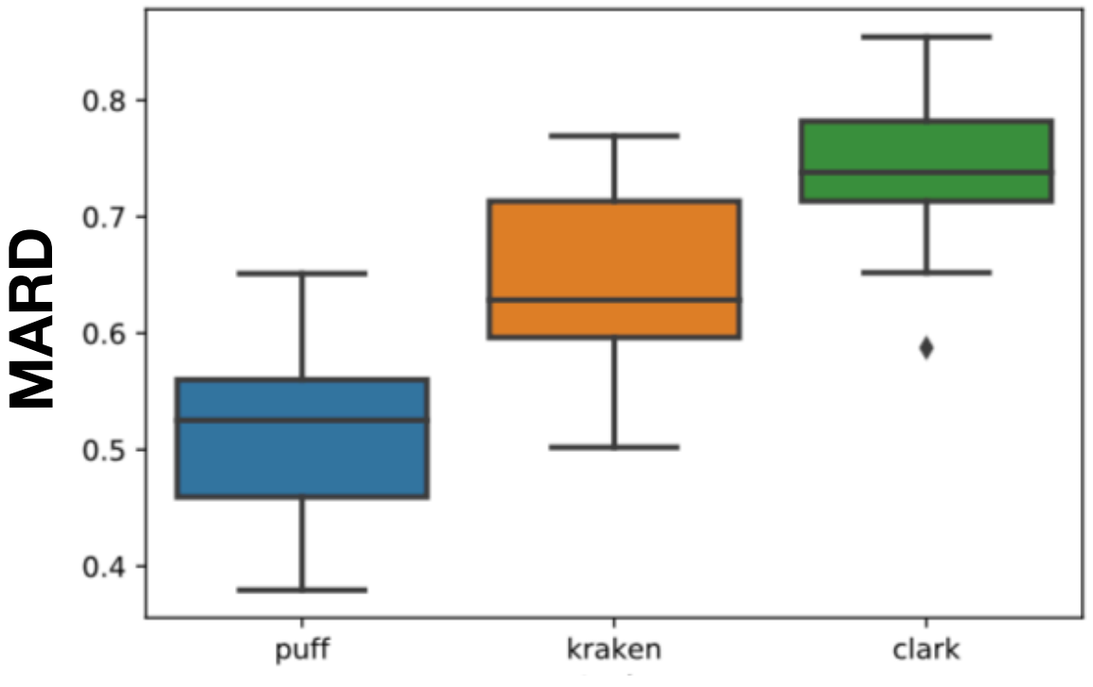
Pufferfish taxonomic assignment

Higher is better





Lower is better (distance)



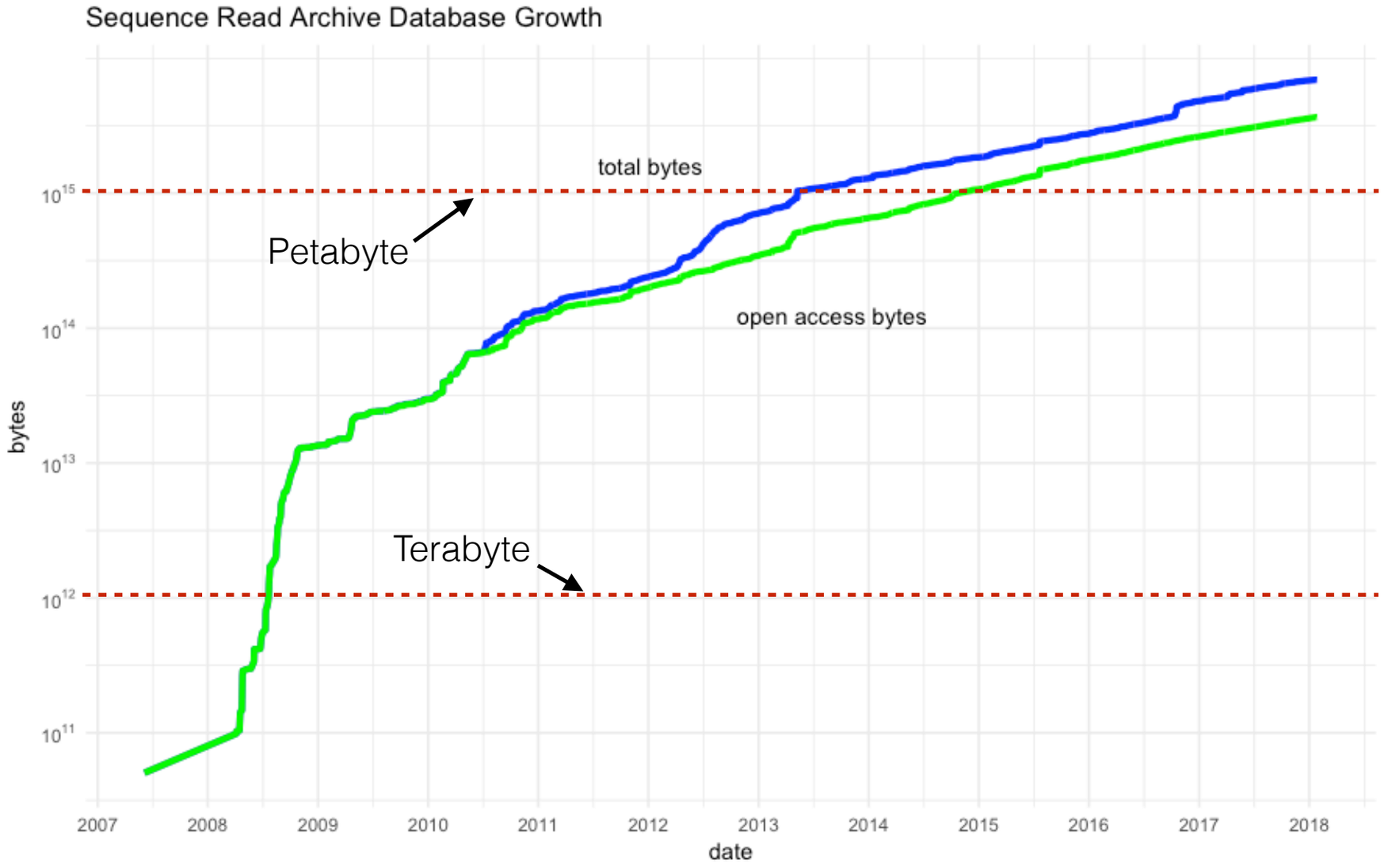
Simulated data from : McIntyre, et al. (2017).
Comprehensive benchmarking and ensemble approaches for metagenomic classifiers. Genome Biology, 18(1).

Simulations:
(LC1-8, HC1, HC2)

The colored de Bruijn Graph as an index for large-scale sequence search

Facing a New Challenge

The Sequence Read Archive (SRA) …



is not searchable by sequence* ! (Yes, I know!)

This renders what is otherwise an immensely valuable public resource **largely inert**

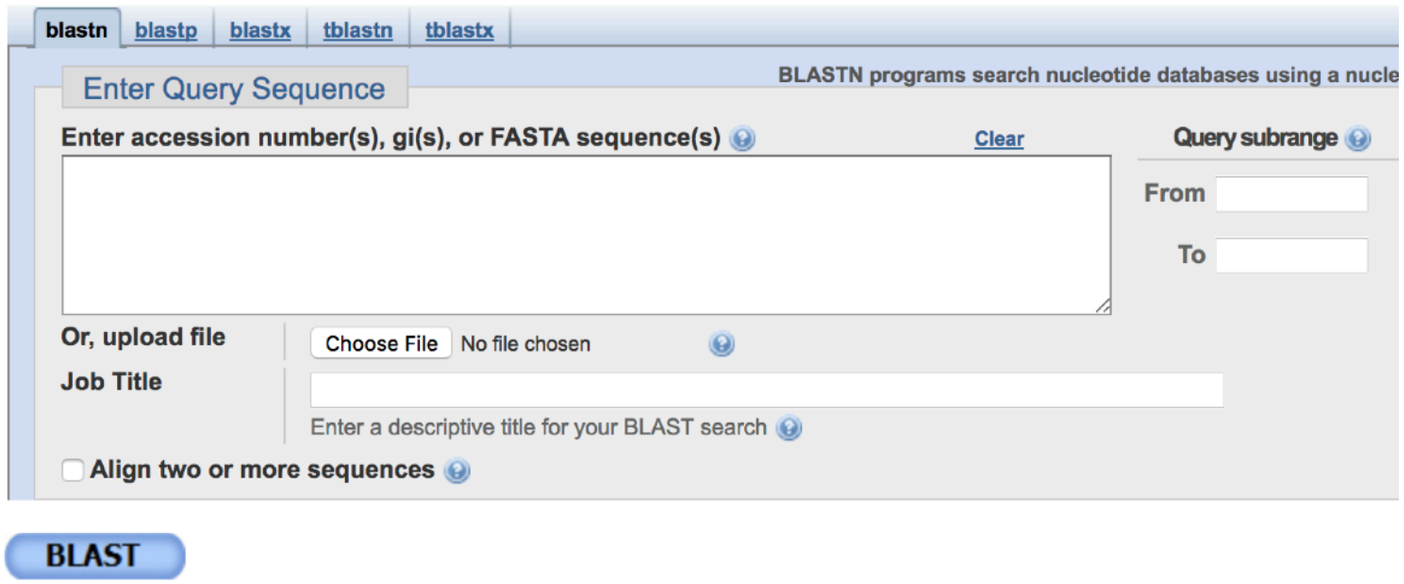
Q: What if I find e.g., a new disease-related gene, and want to see if it appeared in other experiments?

A: (basically) Too bad.

* there is an SRA BLAST, but functionality is limited

Facing a New Challenge

Contrast this situation with the task of searching *assembled*, *curated* genomes, For which we have an *excellent* tool; BLAST*.



Essentially, the “Google of genomics”

However, even the scale of *reference* databases requires *fundamental* algorithmic innovations

_computational BIOLOGY

COMMENTARY

Compressive genomics

Po-Ru Loh, Michael Baym & Bonnie Berger

Algorithms that compute directly on compressed genomic data allow analyses to keep pace with data generation.

Entropy-Scaling Search of Massive Biological Data

Y. William Yu,[1,2,3] Noah M. Daniels,[1,2,3] David Christian Danko,[2] and Bonnie Berger[1,2,*]
[1]Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA 02139, USA
[2]Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139, USA
[3]Co-first author
*Correspondence: bab@mit.edu
http://dx.doi.org/10.1016/j.cels.2015.08.004

*Altschul, S. F., Gish, W., Miller, W., Myers, E. W., & Lipman, D. J. (1990). Basic local alignment search tool. *Journal of molecular biology*, *215*(3), 403-410.

The Computational Problem

So why can't we just use BLAST for searching "raw" data?

- Patterns of interest might be spread across many reads (no contiguous substring)
- The pattern we are looking for may not be present in an assembled genome (we have genomes for only a small fraction of the ~8.7 Million* species on the planet — most of which can't be cultivated in labs)
- There is *so* much more raw data; there is redundancy in raw data, but also diversity. A reference genome reduces entire populations (e.g. humans) to a single string — hugely lossy
- BLAST-like algorithms & data structures just don't seem to scale!

*Mora, Camilo, et al. "How many species are there on Earth and in the ocean?." PLoS biology 9.8 (2011): e1001127.

A New Approach

> Solomon & Kingsford reframe this problem slightly, and suggested a direction toward a potential solution …

Solution:

A hierarchical index of k-mer content represented approximately via Bloom filters.

Returns “yes/no” results for individual experiments → “yes” results can be searched using more traditional methods

Split Sequence Bloom Trees

Split Sequence Bloom Trees : Solomon & Kingsford (RECOMB 2017)

Happy to discuss the algorithmic improvements over SBT offline

Data Index	SBT	Split SBT
Build Time	18 Hr	78 Hr
Compression Time	17 Hr	19 Hr
Uncompressed Size	1295 GB	1853 GB
Compressed Size	**200 GB**	**39.7 GB**

Table 2: Build statistics for SBT and SSBT constructed from a 2652 experiment set. The sizes are the total disk space required to store a bloom tree before or after compression. In SSBT's case, this compression includes the removal of non-informative bits.

Query Time:	θ**=0.7**	θ**=0.8**	θ**=0.9**
SBT	20 Min	19 Min	17 Min
SSBT	3.7 Min	3.5 Min	3.2 Min
RAM SSBT	31 Sec	29 Sec	26 Sec

Table 7: Comparison of query times using different thresholds θ for SBT and SSBT using the set of data at TPM 100.

Solomon, Brad, and Carl Kingsford. "Fast search of thousands of short-read sequencing experiments." *Nature biotechnology* 34.3 (2016): 300-302.

Solomon, B. and Kingsford, C., Improved search of large transcriptomic sequencing databases using split sequence bloom trees. In *International Conference on Research in Computational Molecular Biology* (pp. 257-271). Springer, Cham.

A fundamentally different approach

Our initial idea — the Bloom Filter is limiting.
What can we get by replacing it with a *better* AMQ

A General-Purpose Counting Filter: Making Every Bit Count

Prashant Pandey, Michael A. Bender, Rob Johnson, and Rob Patro
Stony Brook University
Stony Brook, NY, USA
{ppandey, bender, rob, rob.patro}@cs.stonybrook.edu

SIGMOD 2017

Interesting observation
about patterns of k-mer occurrence

Rainbowfish: A Succinct Colored de Bruijn Graph Representation*

Fatemeh Almodaresi[1], Prashant Pandey[2], and Rob Patro[3]

1 Stony Brook University, Stony Brook, NY, USA
falmodaresit@cs.stonybrook.edu
2 Stony Brook University, Stony Brook, NY, USA
ppandey@cs.stonybrook.edu
3 Stony Brook University, Stony Brook, NY, USA
rob.patro@cs.stonybrook.edu

WABI 2017

"I bet we can exploit
that for large-scale search"

Mantis: A Fast, Small, and Exact Large-Scale Sequence-Search Index

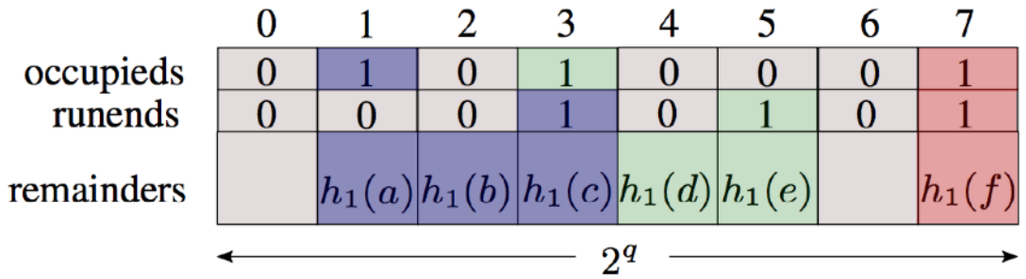
Prashant Pandey[1], Fatemeh Almodaresi[1], Michael A. Bender[1], Michael Ferdman[1], Rob Johnson[2,1], and Rob Patro[1]

[1] Computer Science Dept., Stony Brook University
{ppandey,falmodaresit,bender,mferdman,rob.patro}@cs.stonybrook.edu
[2] VMware Research
robj@vmware.com

RECOMB 2018 & Cell Systems (https://doi.org/10.1016/j.cels.2018.05.021)

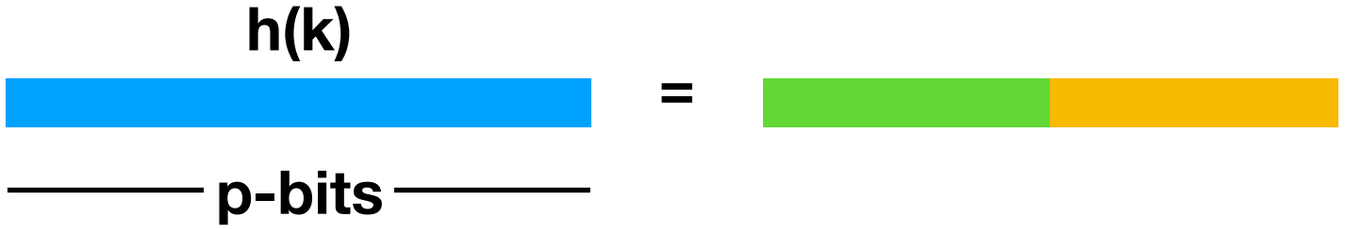
The CQF

Approximate *Multiset* Representation



Works based on quotienting* & fingerprinting keys

Let k be a key and h(k) a p-bit hash value



Clever encoding allows low-overhead storage of element counts
(use *key* slots to store *values* in base 2^r-1; smaller values ⇒ fewer bits)

Careful engineering & use of efficient rank & select to resolve collisions leads to a fast, cache-friendly data structure

* Idea goes back at least to Knuth (TACOP vol 3)

The CQF

Approximate *Multiset* Representation

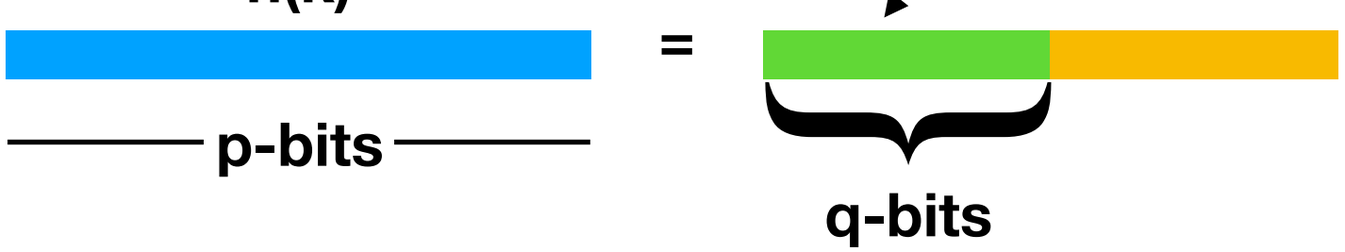
	0	1	2	3	4	5	6	7
occupieds	0	1	0	1	0	0	0	1
runends	0	0	0	1	0	1	0	1
remainders		$h_1(a)$	$h_1(b)$	$h_1(c)$	$h_1(d)$	$h_1(e)$		$h_1(f)$

2^q

Works based on quotienting* & fingerprinting keys

Let k be a key and h(k) a p-bit hash value

Determines position in array of size 2^q r-bit slots



Clever encoding allows low-overhead storage of element counts
(use *key* slots to store *values* in base 2^r-1; smaller values ⇒ fewer bits)

Careful engineering & use of efficient rank & select to resolve collisions leads to a fast, cache-friendly data structure

* Idea goes back at least to Knuth (TACOP vol 3)

The CQF

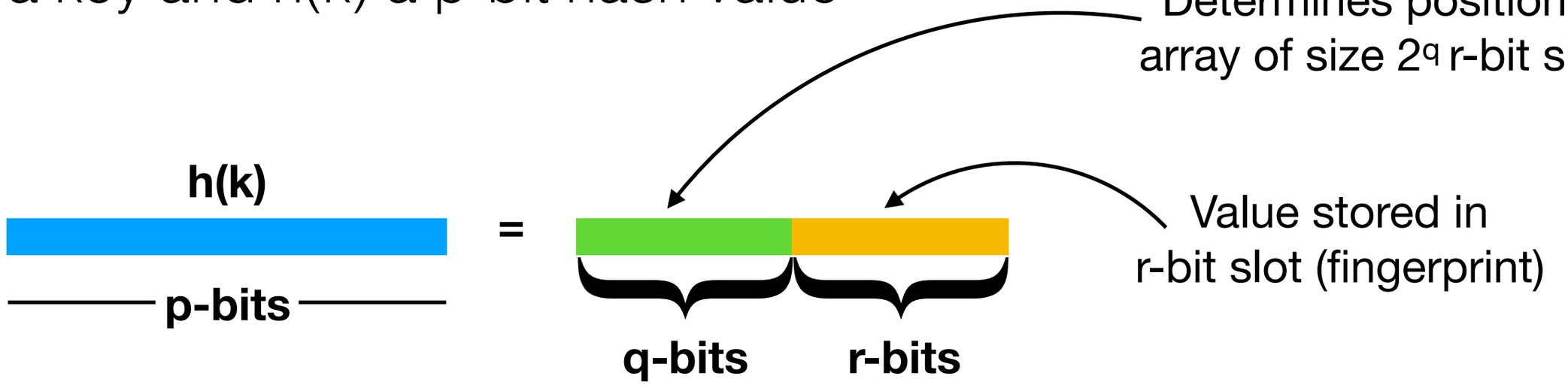
Approximate *Multiset* Representation

	0	1	2	3	4	5	6	7
occupieds	0	1	0	1	0	0	0	1
runends	0	0	0	1	0	1	0	1
remainders		$h_1(a)$	$h_1(b)$	$h_1(c)$	$h_1(d)$	$h_1(e)$		$h_1(f)$

2^q

Works based on quotienting* & fingerprinting keys

Let k be a key and h(k) a p-bit hash value



Clever encoding allows low-overhead storage of element counts
(use *key* slots to store *values* in base 2^r-1; smaller values ⇒ fewer bits)

Careful engineering & use of efficient rank & select to resolve collisions leads to a fast, cache-friendly data structure

* Idea goes back at least to Knuth (TACOP vol 3)

Mantis

Observation 1 : If I want to index N k-mers over E experiments, there are $\leq \min\left(N, 2^{|E|}\right)$ possible distinct “patterns of occurrence” of the k-mers, there are usually *many* fewer.

Observation 2 : These patterns of occurrence are *far* from uniform. Specifically, k-mers don't occur independently, occurrences are *highly correlated.*

Why?

https://github.com/splatlab/mantis

Mantis

Observation 1 : If I want to index N k-mers over E experiments, there are $\leq \min\left(N, 2^{|E|}\right)$ possible distinct "patterns of occurrence" of the k-mers, there are usually *many* fewer.

Observation 2 : These patterns of occurrence are *far* from uniform. Specifically, k-mers don't occur independently, occurrences are *highly correlated.*

Why? Consider e.g. a gene G (~1000 k-mers). If it is present in an experiment at moderate to high abundance, we will likely observe *all of it's k-mers*.

https://github.com/splatlab/mantis

Mantis

Observation 1 : If I want to index N k-mers over E experiments, there are $\leq \min\left(N, 2^{|E|}\right)$ possible distinct "patterns of occurrence" of the k-mers, there are usually *many* fewer.

Observation 2 : These patterns of occurrence are *far* from uniform. Specifically, k-mers don't occur independently, occurrences are *highly correlated.*

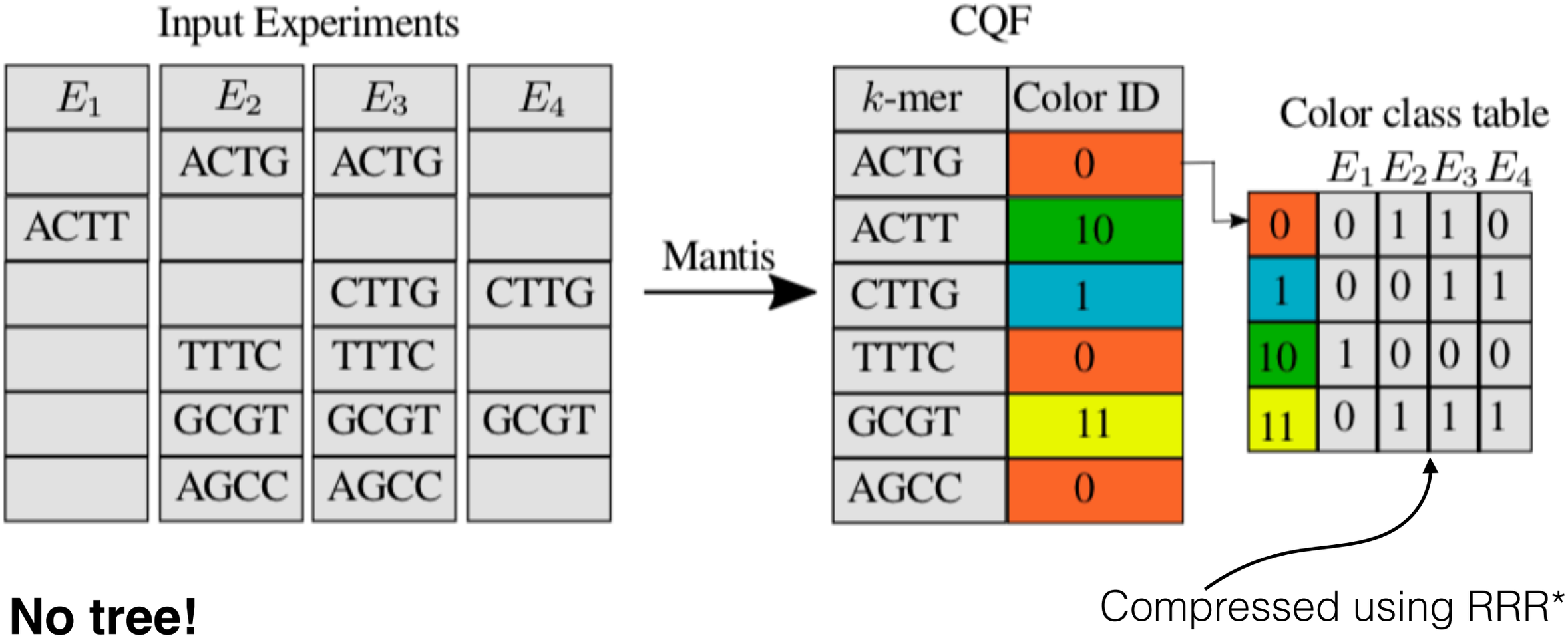
Why? Consider e.g. a gene G (~1000 k-mers). If it is present in an experiment at moderate to high abundance, we will likely observe *all of it's k-mers*.

What if we add a layer of indirection: Store each distinct pattern (color class) only once. *label* each pattern with with an index, s.t. frequent patterns get small numbers (think Huffman encoding)

David Wheeler approves … we think.

https://github.com/splatlab/mantis

The Mantis Index: Core Idea



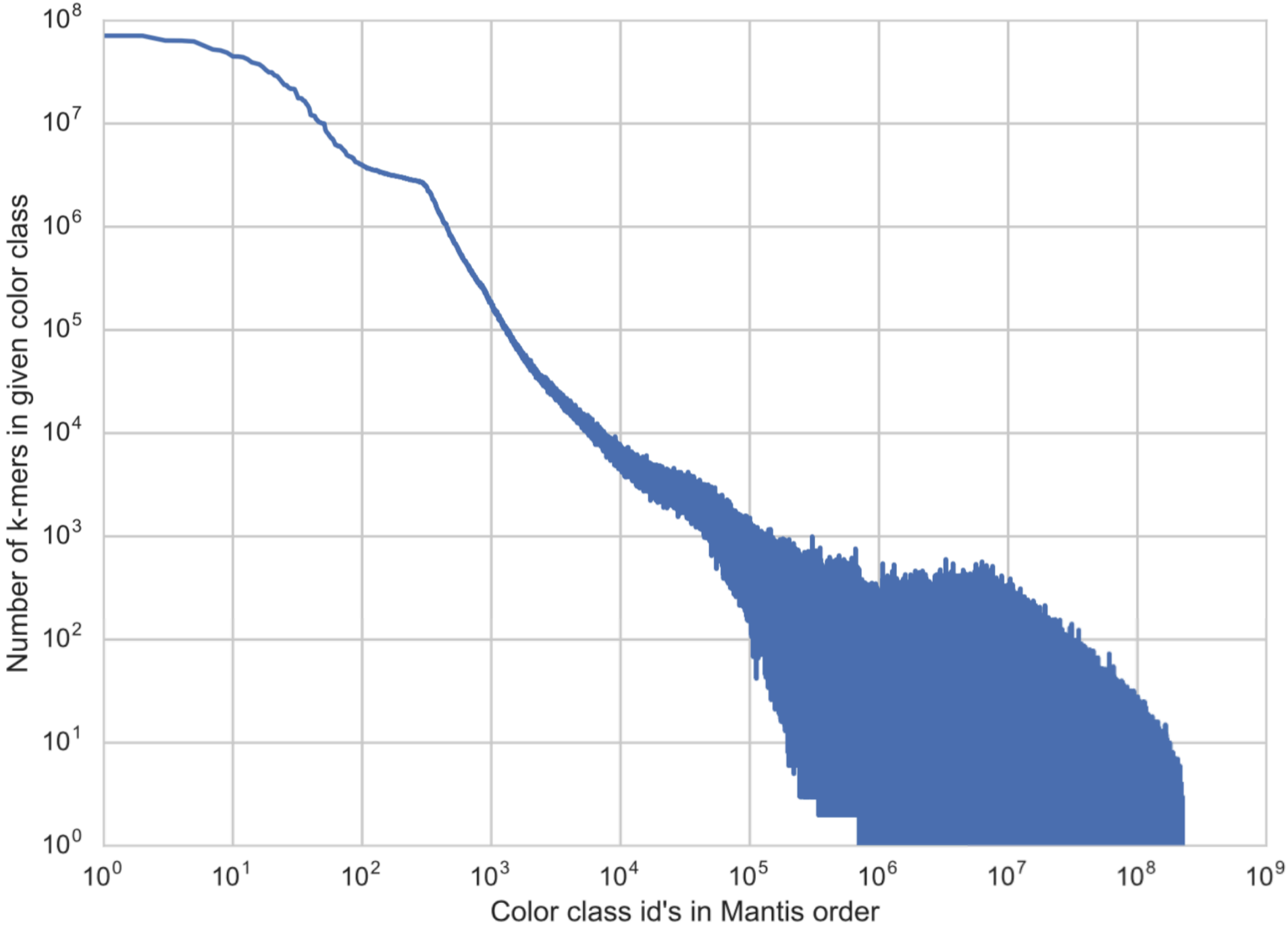
No tree!

- Build a CQF for each input experiment (can be different sizes, since CQFs of different sizes are mergeable)
- Combine them via multi-way merge
- CQF : key = k-mer, value = color class ID
- *Estimate* a good ordering of color class IDs from first few million k-mers

*Raman, et al. (2002). Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. In Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms, pages 233–242.

Why does this work?

The distribution of k-mers / color class is *highly skewed*



~3.7 Billion k-mers from ~2,600 distinct sequencing experiments

Mantis : Comparing to SSBT

Construction Time — How long does it take to build the index?

Index Size — How large is the index, in terms of storage space?

Query Performance — How long does it take to execute queries?

Result Accuracy — How many FP positives are included in query results?

Bonus: If the remainder + quotient bits = original key size & we use an invertible hash, the CQF is *exact*.

Mantis is compact enough that we can *exactly* rather than *approximately* index the k-mers in our experiment set.

This lets us ask useful questions about how other approaches perform.

Mantis : Construction Time & Index Size

Indexed 2,652 human RNA-seq (gene expression) experiments
~4.5TB (GZip compressed) of data

Table 1. Time and Space Measurement for Mantis and SSBT

	Mantis	SSBT
Build time	16 hr 35 min	97 hr
Representation size	32 GB	39.7 GB

- Mantis can be constructed ~6x faster than a comparable SSBT
- The final Mantis representation is ~20% smaller than the comparable SSBT representation.

Note: both results assume you already have per-experiment AMQs (either Bloom Filters or CQFs)

Mantis : Query Speed

Querying for the presence of randomly selected genes across all 2,652 experiments.

θ threshold for SSBT query

	Mantis	SSBT (0.7)	SSBT (0.8)	SSBT (0.9)
10 Transcripts	25 s	3 min 8 s	2 min 25 s	2 min 7 s
100 Transcripts	28 s	14 min 55 s	10 min 56 s	7 min 57 s
1000 Transcripts	1 min 3 s	2 hr 22 min	1 hr 54 min	1 hr 20 min

- Mantis is ~6 — 109x faster than (in memory) SSBT

Note: Mantis doesn't require a θ threshold for queries, though one can be applied *post hoc*.

A Mantis query returns, for each experiment containing at least one query k-mer, the *fraction* (true θ) of query k-mers contained in the experiment.

Mantis : Query Quality

Querying for the presence of randomly selected genes across all 2,652 experiments. SSBT θ = 0.8

	Both	Only Mantis	Only SSBT	Precision
10 Transcripts	2,018	19	1,476	0.577
100 Transcripts	22,466	146	10,588	0.679
1000 Transcripts	160,188	1,409	95,606	0.626

"Both" means the number of those experiments that are reported by both Mantis and SSBT. "Only Mantis" and "Only SSBT" mean the number of experiments reported by only Mantis and only SSBT. All three query benchmarks are taken from Table 2 for $\theta = 0.8$.

- Recall : Mantis is exact! Returns *only* experiments having ≥ θ fraction of the query k-mers.

Mantis : Query Quality

Querying for the presence of randomly selected genes across all 2,652 experiments. SSBT θ = 0.8

	Both	Only Mantis	Only SSBT	Precision
10 Transcripts	2,018	19	1,476	0.577
100 Transcripts	22,466	146	10,588	0.679
1000 Transcripts	160,188	1,409	95,606	0.626

- Recall : Mantis is exact! Returns *only* experiments having ≥ θ fraction of the query k-mers.

Due to a small number of corrupted SSBT filters — able to discover this b/c of Mantis' exact nature.

Some Remaining Challenges

- It improves greatly upon existing solutions; takes a different approach
- We demonstrate indexing on the order of 10^3 experiments, we really want to index on the order of 10^5 - 10^6
- Can be made approximate while providing strong bounds :

Theorem 1. *A query for q k-mers with threshold θ returns only experiments containing at least $\theta q - O(\delta q + \log n)$ queried k-mers w.h.p.*

but maybe not enough

Key Observation:

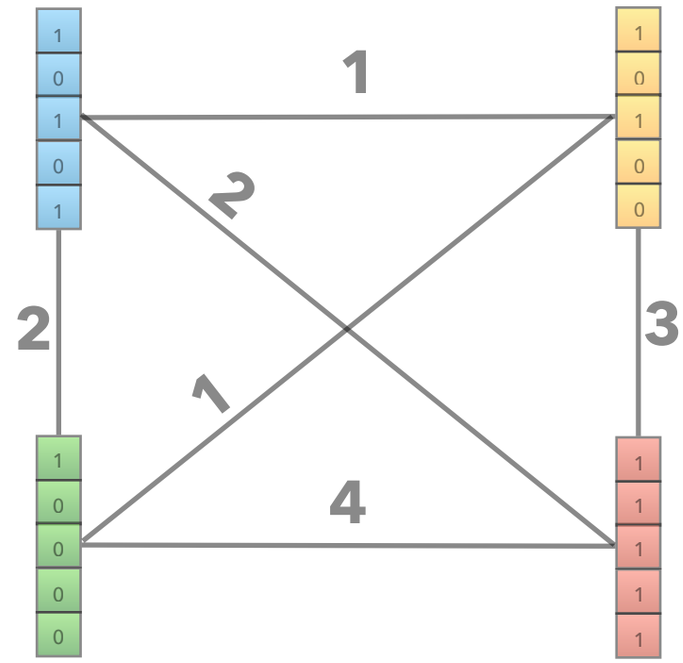
- K-mers grow at worst linearly
- Color classes increase super-linearly

Need a **fundamentally better** color class encoding; exploit *coherence* between rows of the color class matrix

Consider the following color class graph

Each color class is a vertex

Every pair of color classes is connected by an edge whose weight is the **hamming distance** between the color class vectors

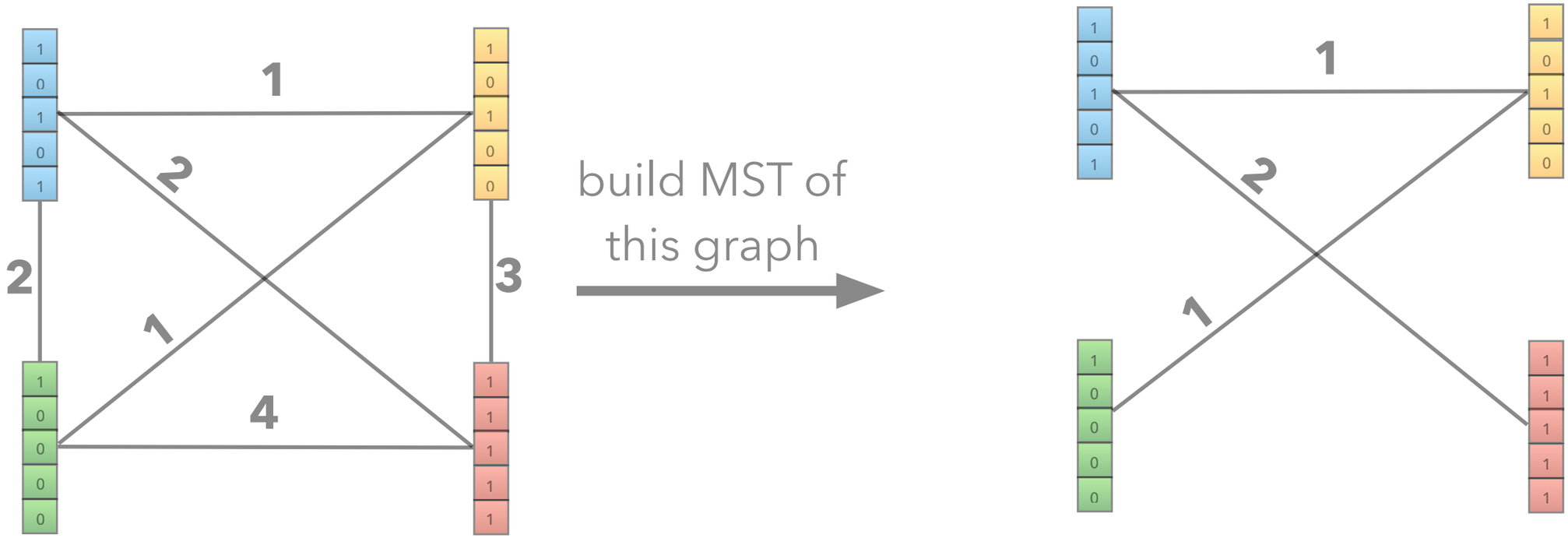


Bookstein, Abraham, and Shmuel T. Klein. "Compression of correlated bit-vectors." *Inf. Syst.* 16.4 (1991): 387-400.

Consider the following color class graph

Each color class is a vertex

Every pair of color classes is connected by an edge whose weight is the **hamming distance** between the color class vectors

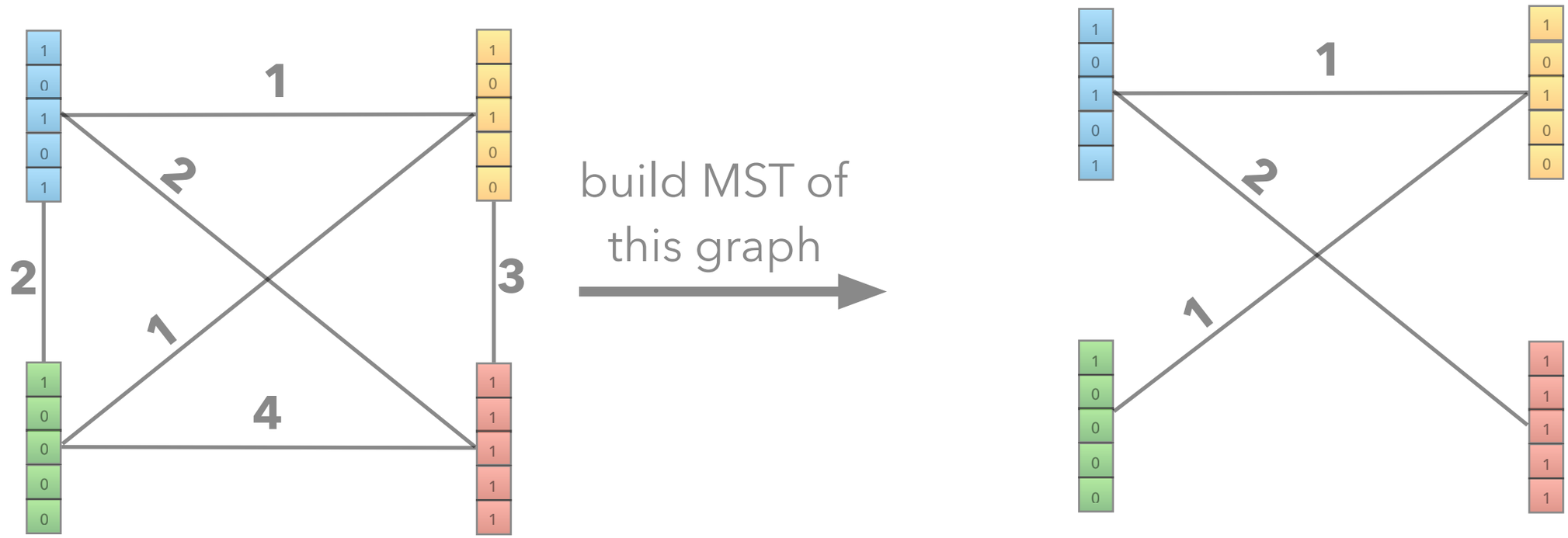


Bookstein, Abraham, and Shmuel T. Klein. "Compression of correlated bit-vectors." *Inf. Syst.* 16.4 (1991): 387-400.

Consider the following color class graph

Each color class is a vertex

Every pair of color classes is connected by an edge whose weight is the **hamming distance** between the color class vectors



Unfortunately:

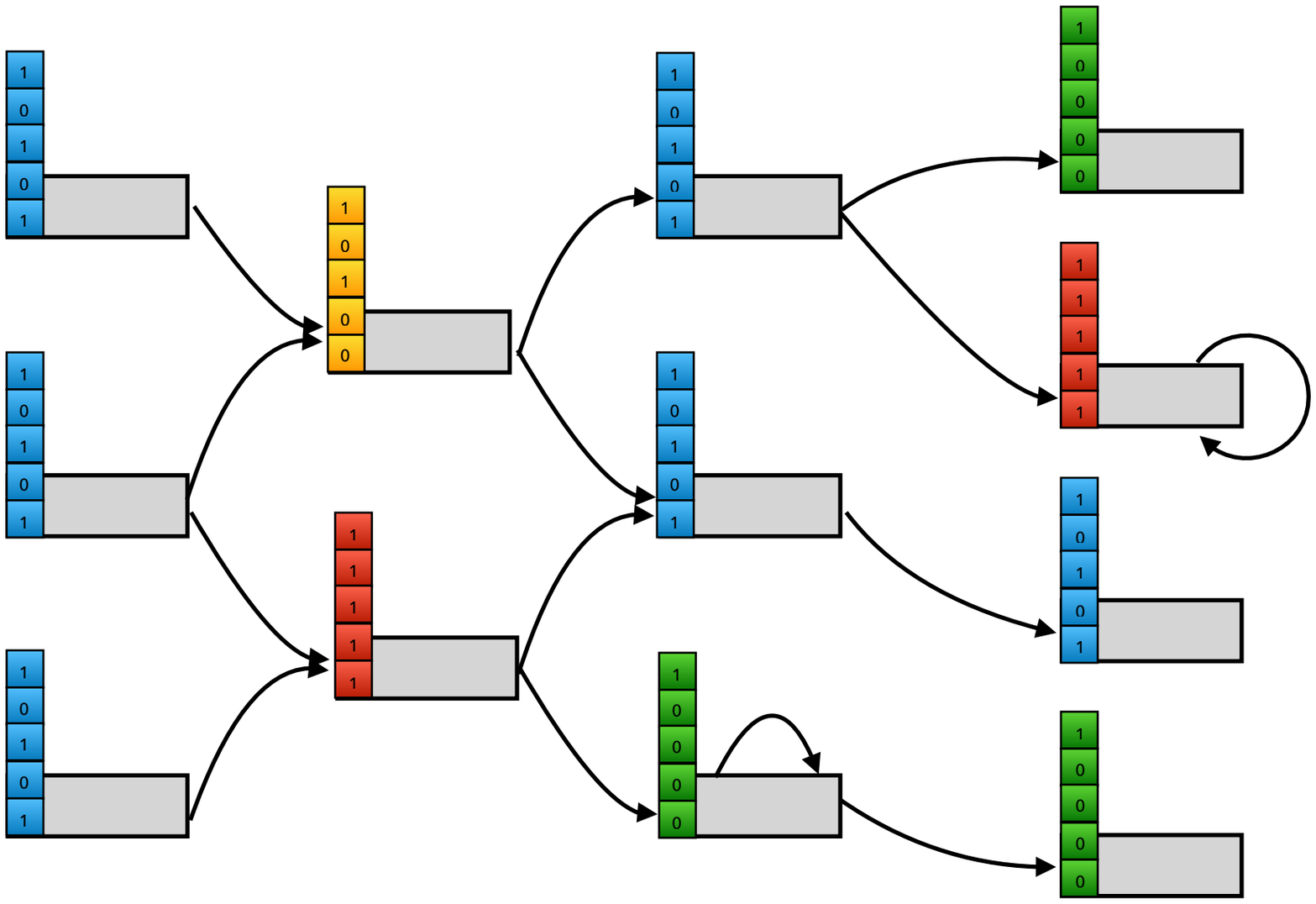
1) There are *many* color classes (full graph too big)
2) They are high-dimensional (# of experiments), neighbor search is very hard (LSH scheme seem to work poorly)

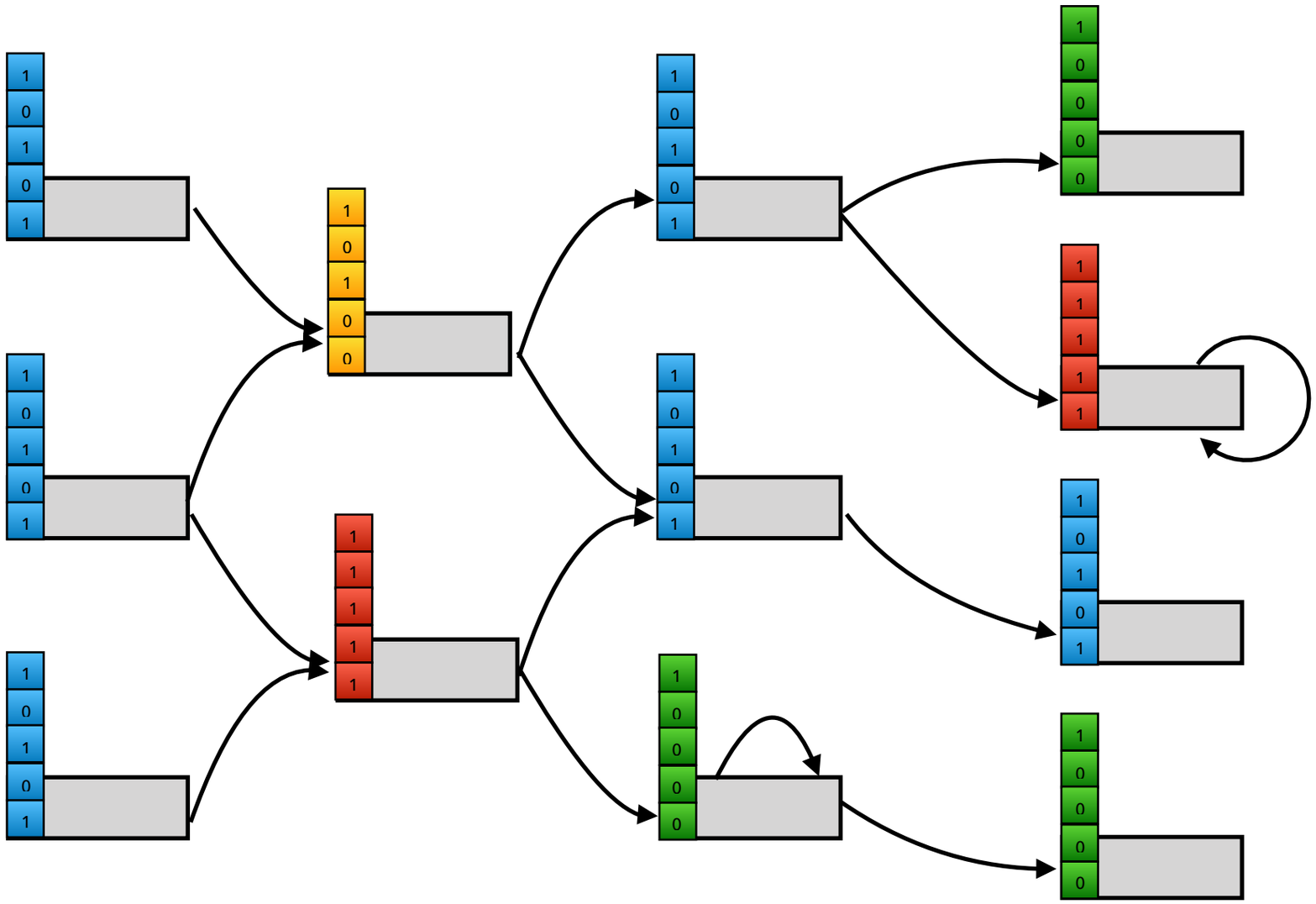
Bookstein, Abraham, and Shmuel T. Klein. "Compression of correlated bit-vectors." *Inf. Syst.* 16.4 (1991): 387-400.

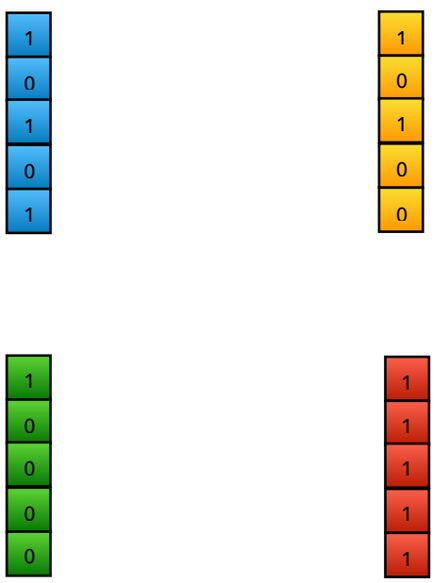
Mantis implicitly represents a colored dBG

Each CQF key represents a kmer → can explicitly query neighbors

Each k-mer associated with color class id → vector of occurrences

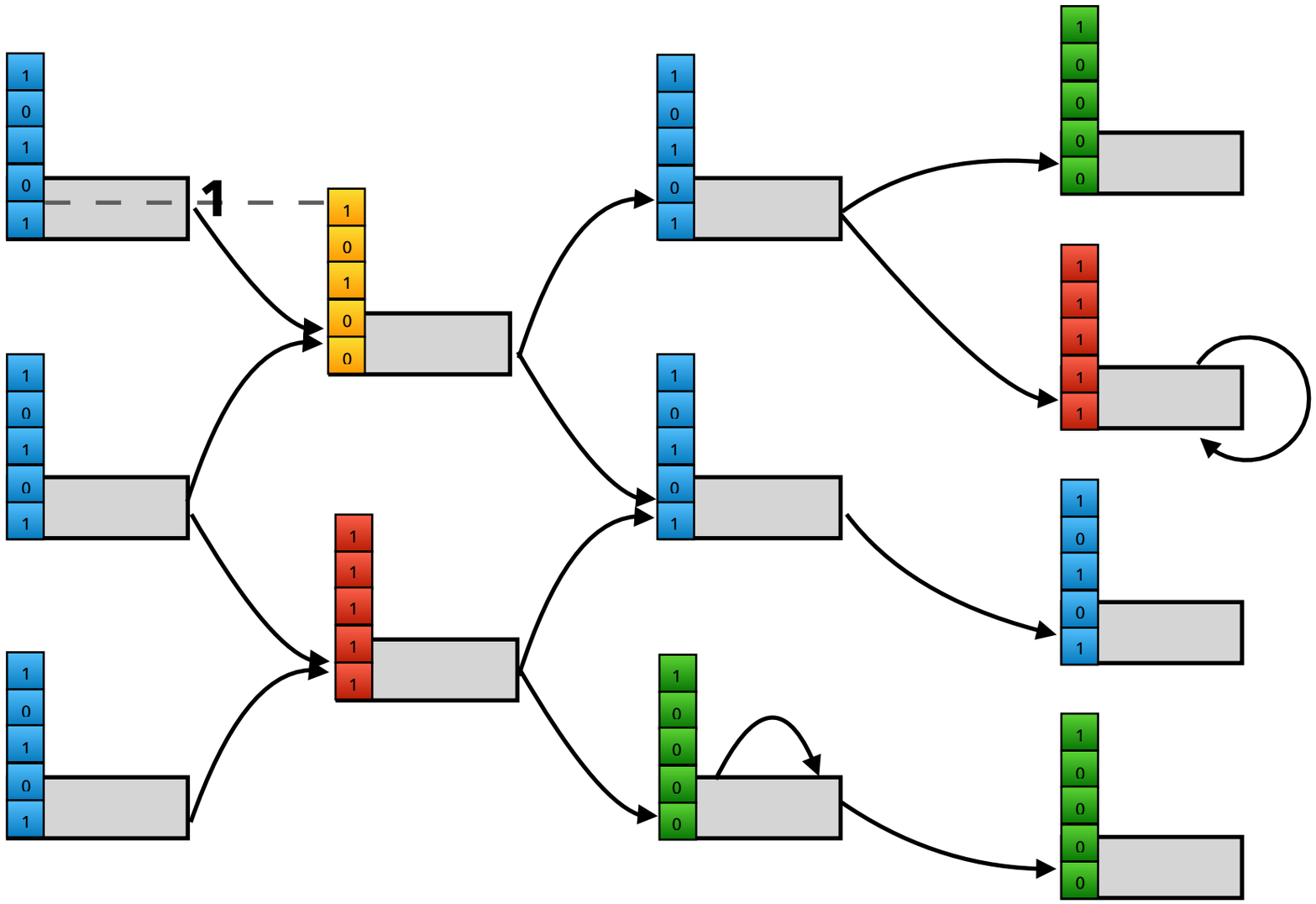


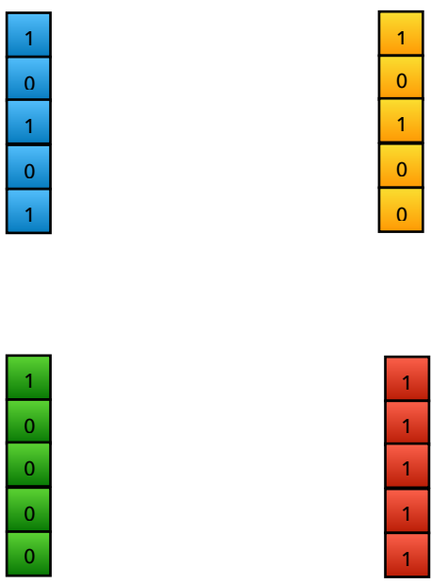




Use the **de Bruin graph** (dBG) as an efficient guide for near-neighbor search in the space of color classes!

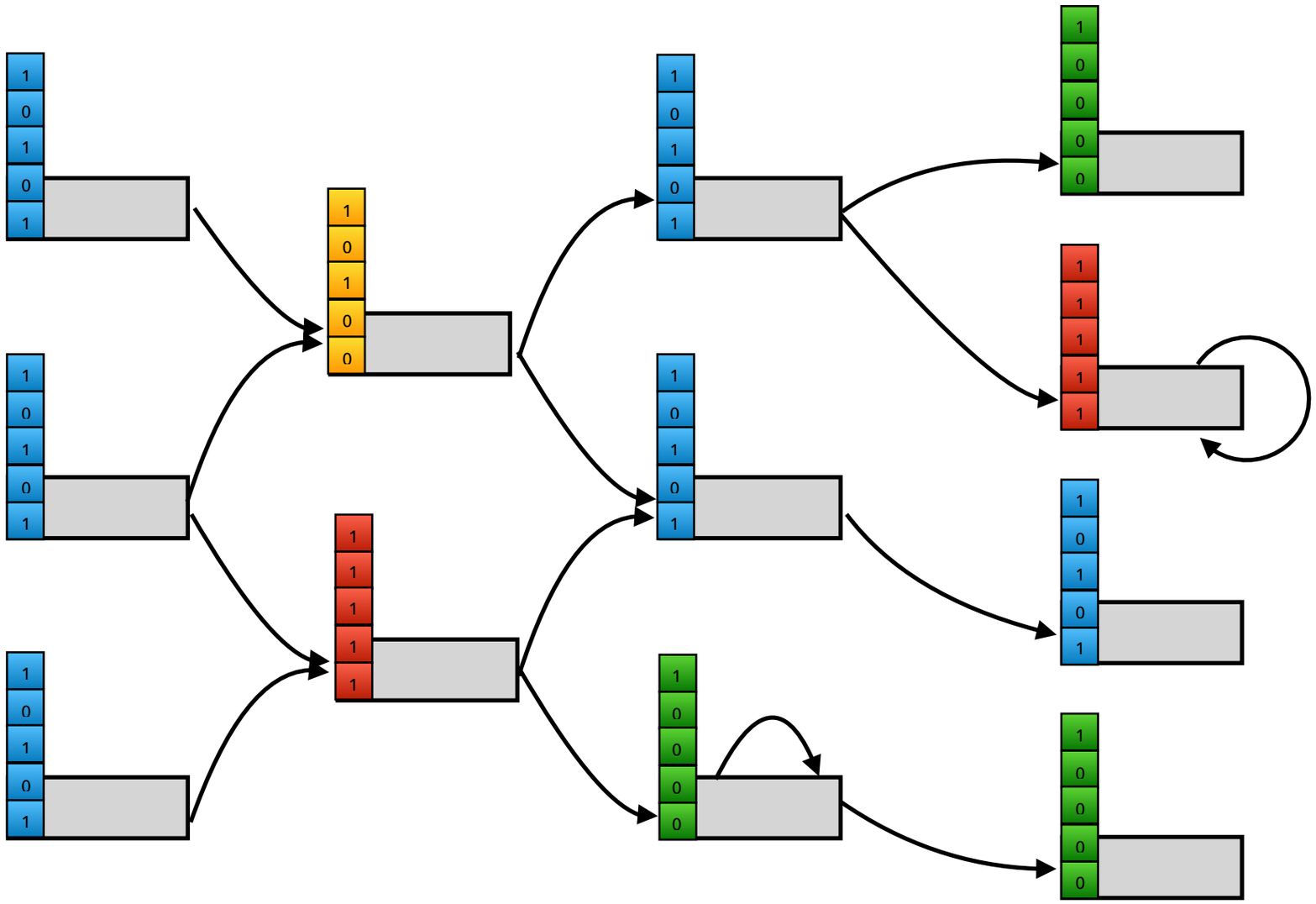
dBG common in genomics. Nodes u,v are k-mers & are *adjacent* if k-1 suffix of u is the same as k-1 prefix of v

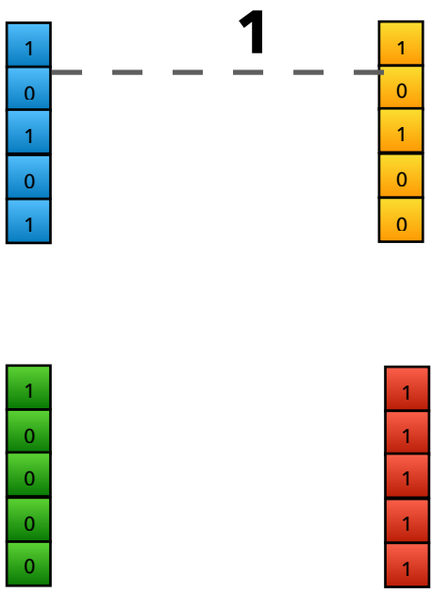




Use the **de Bruin graph** (dBG) as an efficient guide for near-neighbor search in the space of color classes!

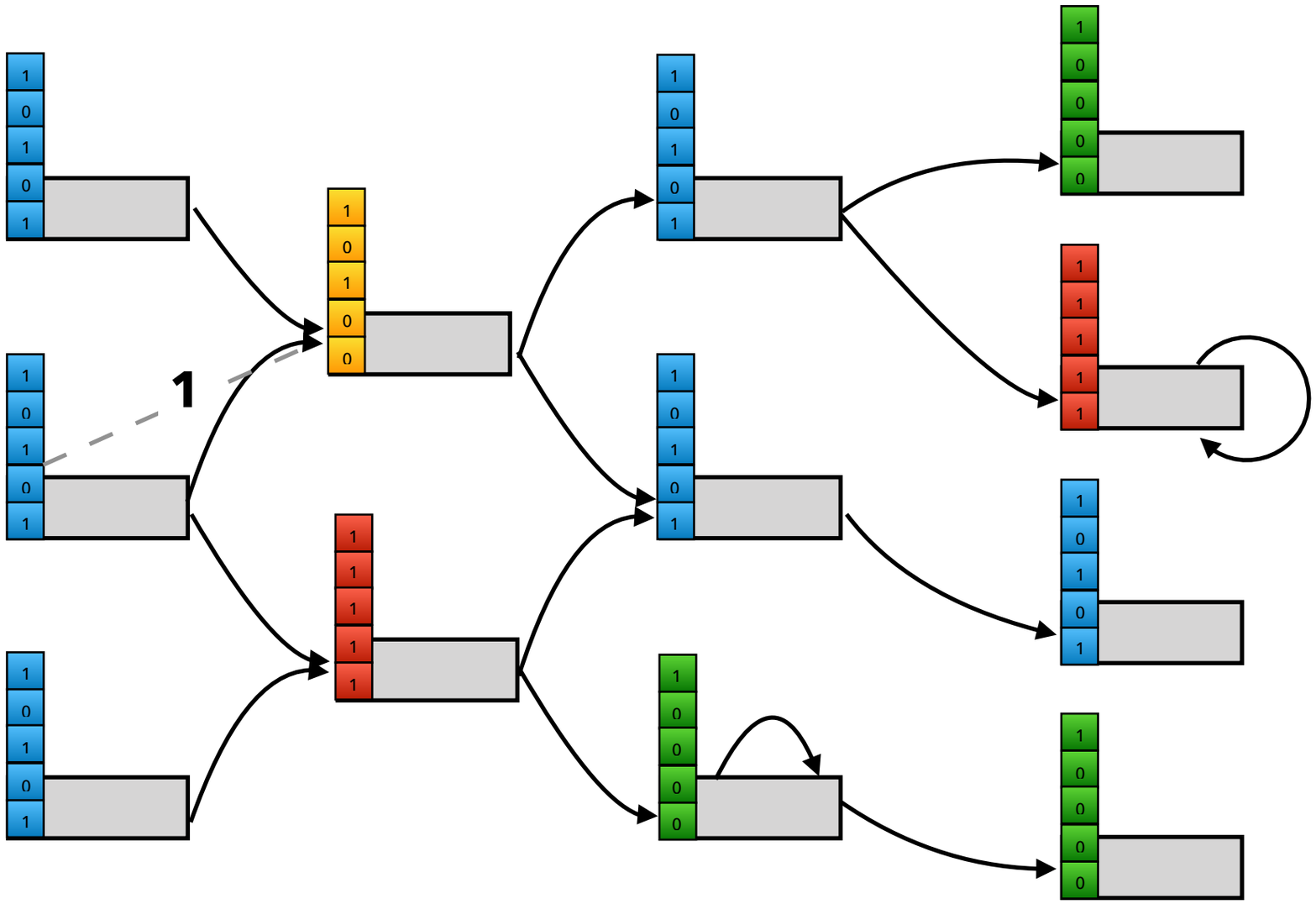
dBG common in genomics. Nodes u,v are k-mers & are *adjacent* if k-1 suffix of u is the same as k-1 prefix of v

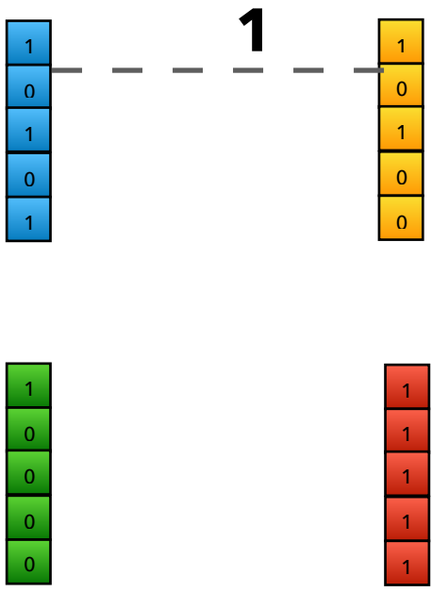




Use the **de Bruin graph** (dBG) as an efficient guide for near-neighbor search in the space of color classes!

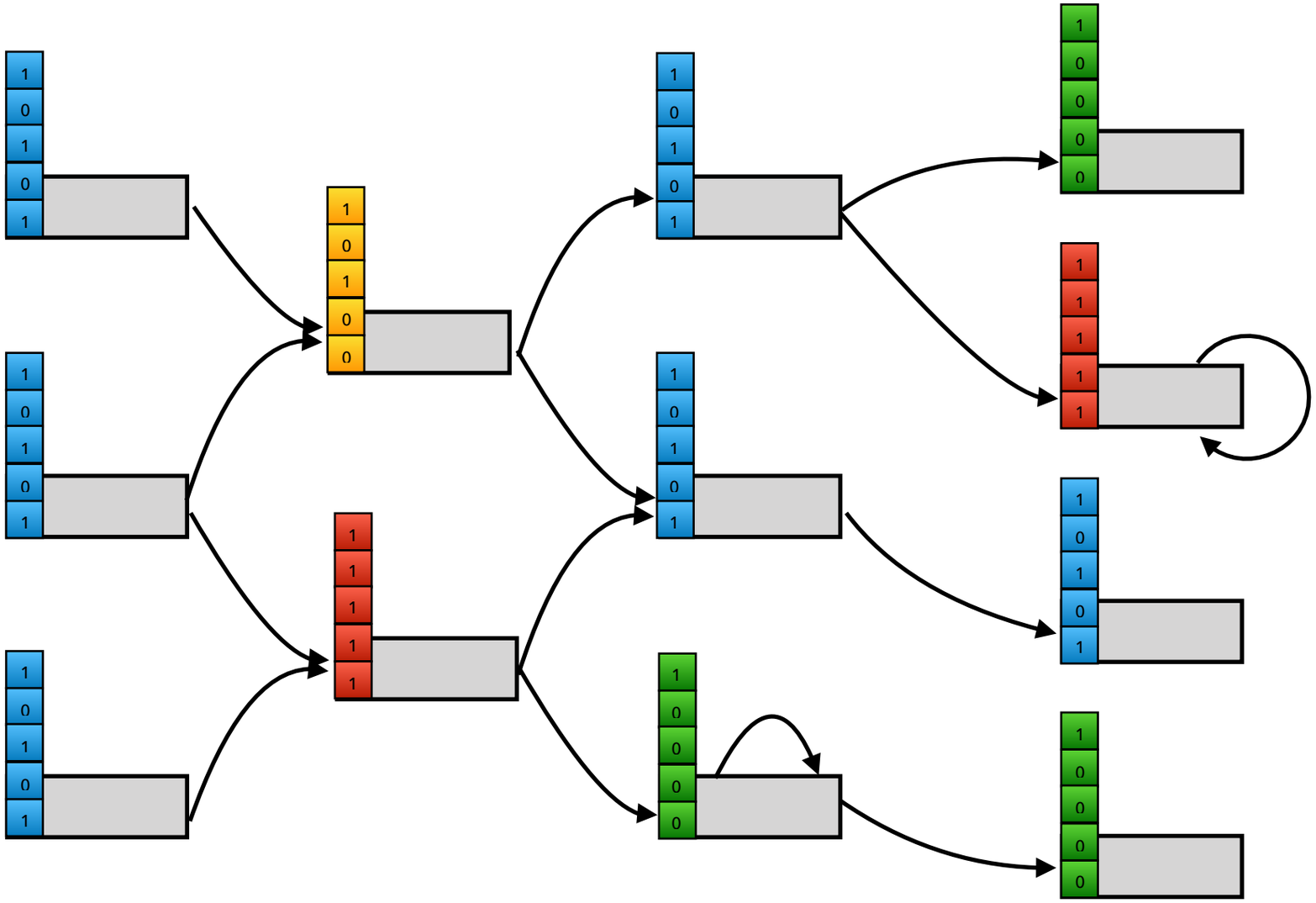
dBG common in genomics. Nodes u,v are k-mers & are *adjacent* if k-1 suffix of u is the same as k-1 prefix of v

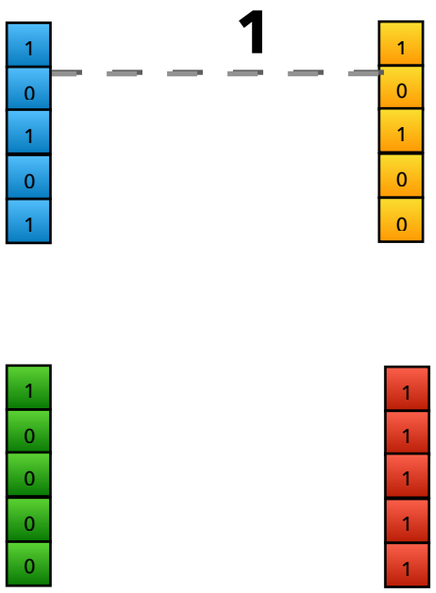




Use the **de Bruin graph** (dBG) as an efficient guide for near-neighbor search in the space of color classes!

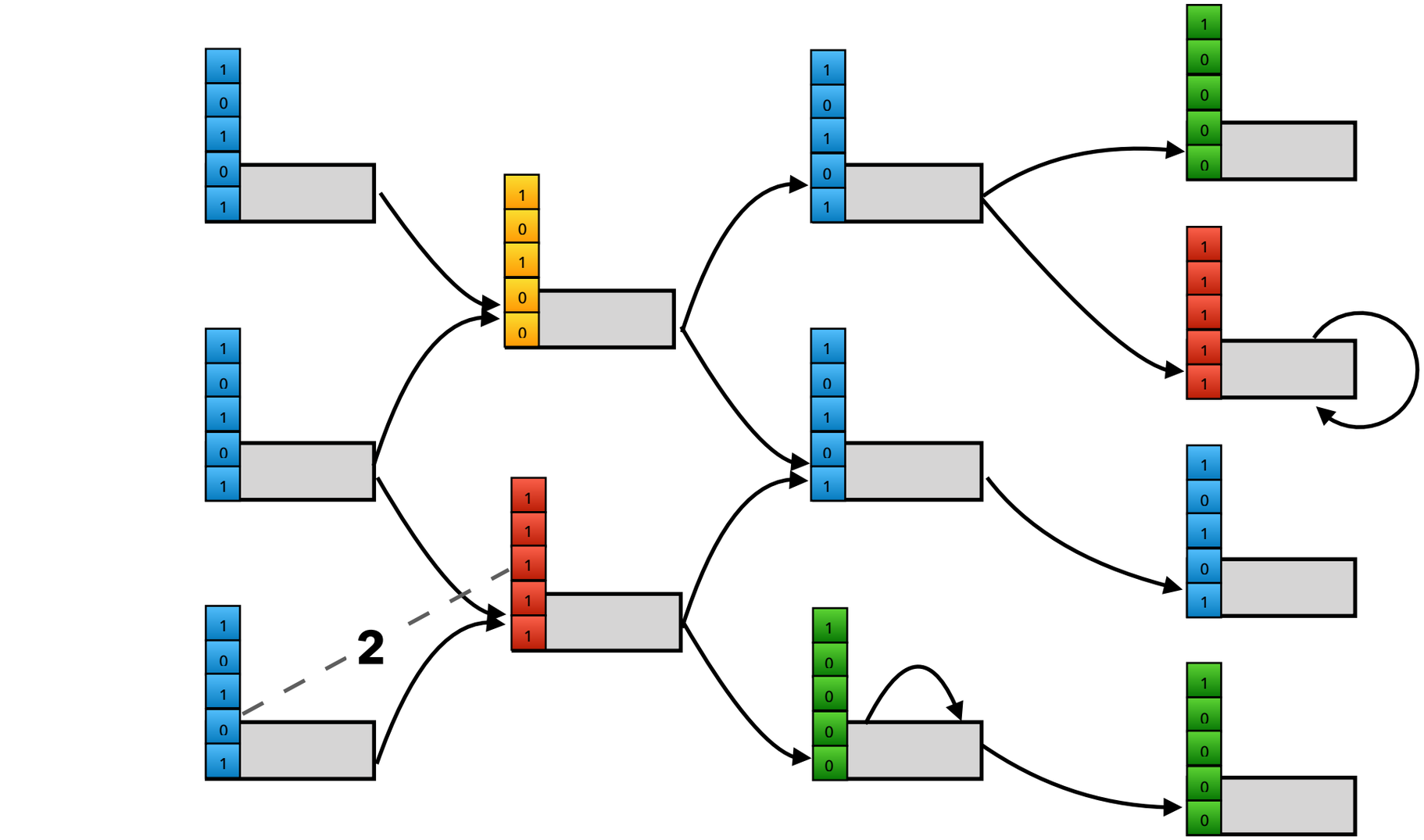
dBG common in genomics. Nodes u,v are k-mers & are *adjacent* if k-1 suffix of u is the same as k-1 prefix of v

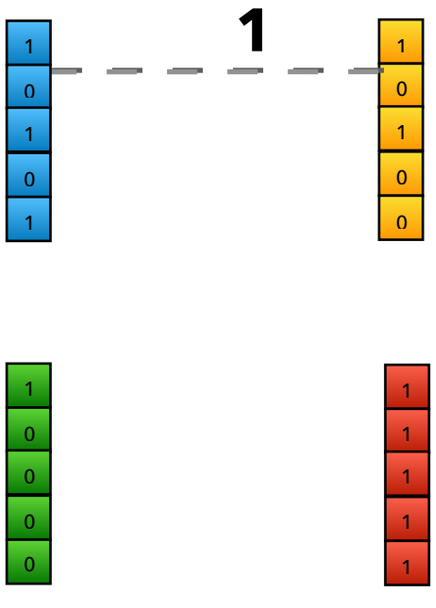




Use the **de Bruin graph** (dBG) as an efficient guide for near-neighbor search in the space of color classes!

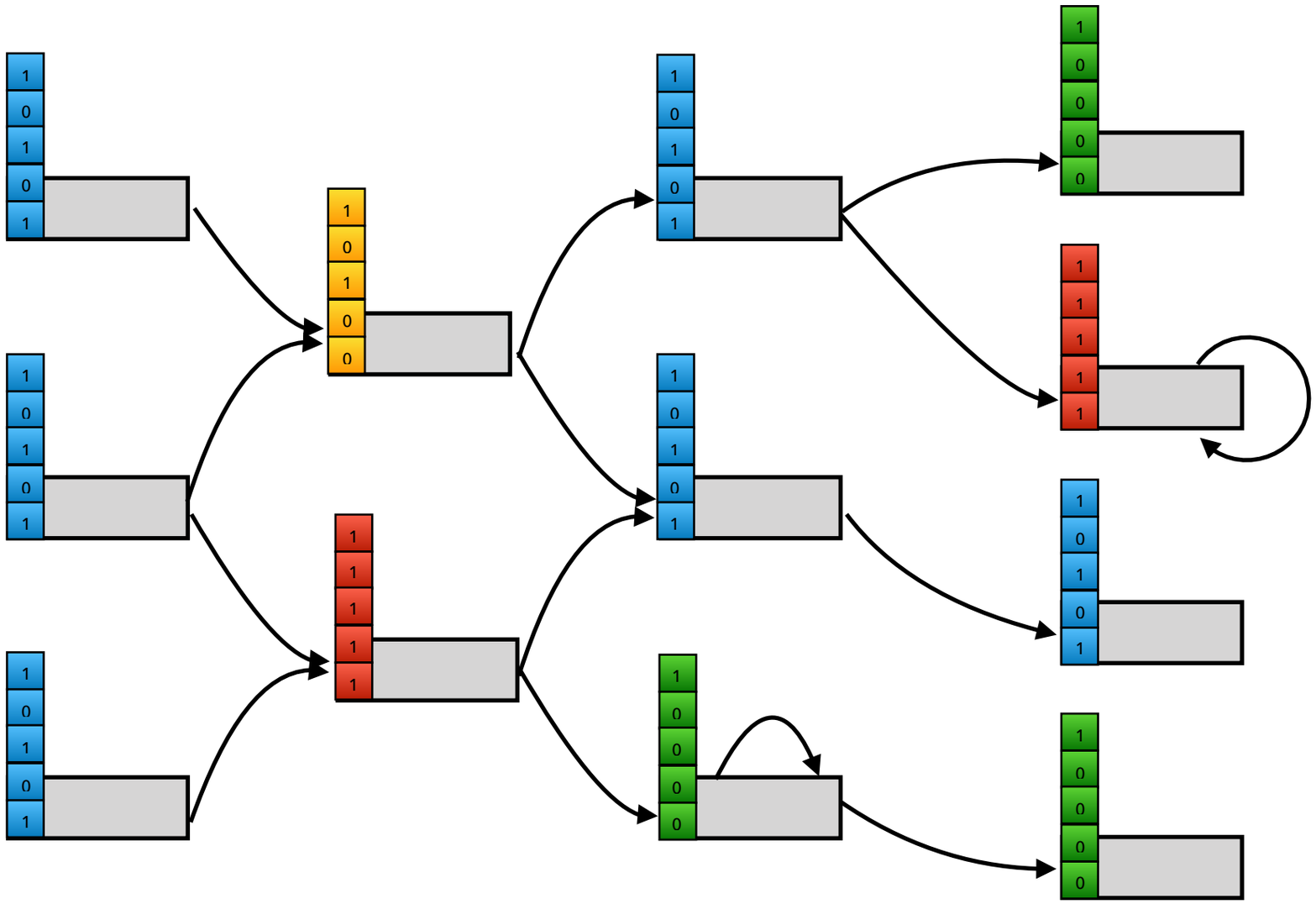
dBG common in genomics. Nodes u,v are k-mers & are *adjacent* if k-1 suffix of u is the same as k-1 prefix of v

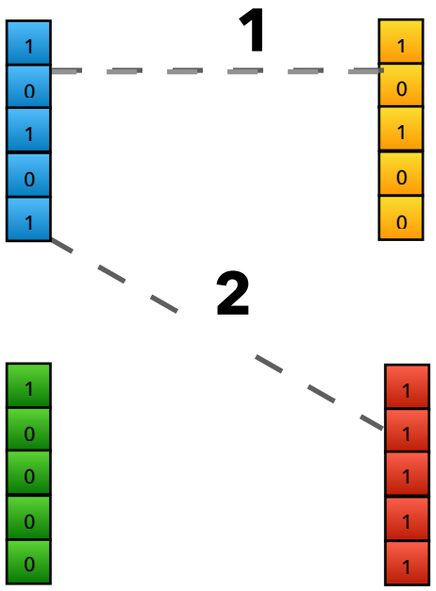




Use the **de Bruin graph** (dBG) as an efficient guide for near-neighbor search in the space of color classes!

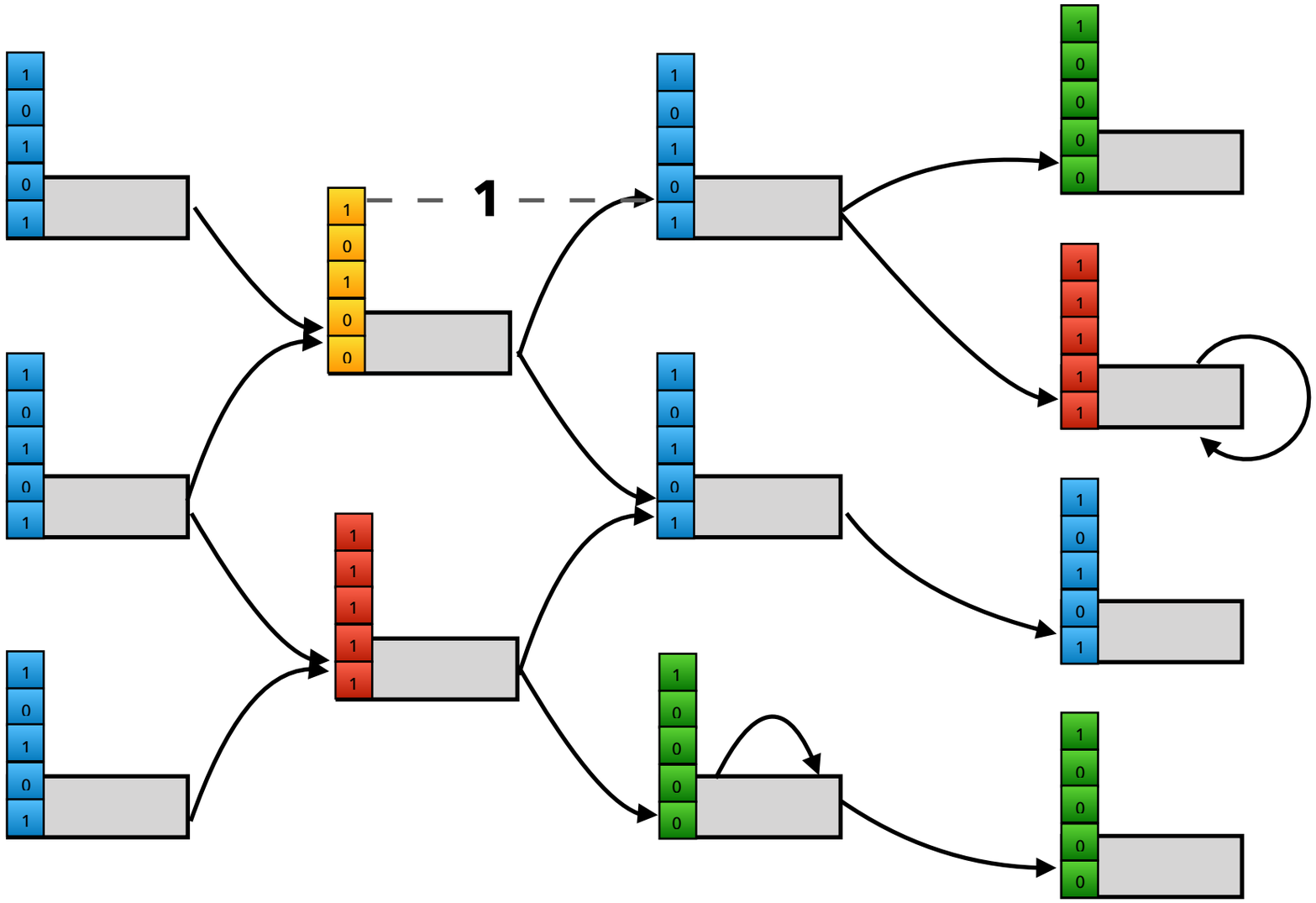
dBG common in genomics. Nodes u,v are k-mers & are *adjacent* if k-1 suffix of u is the same as k-1 prefix of v

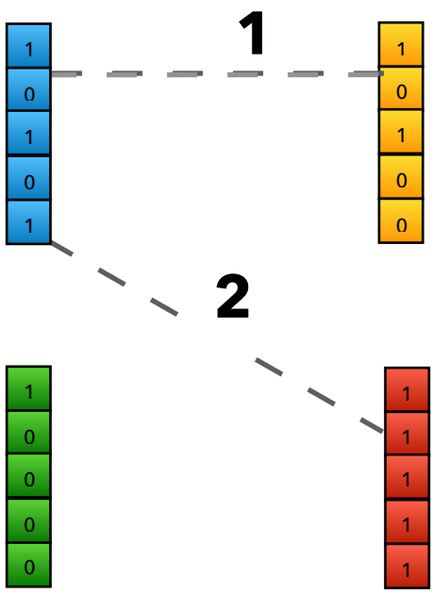




Use the **de Bruin graph** (dBG) as an efficient guide for near-neighbor search in the space of color classes!

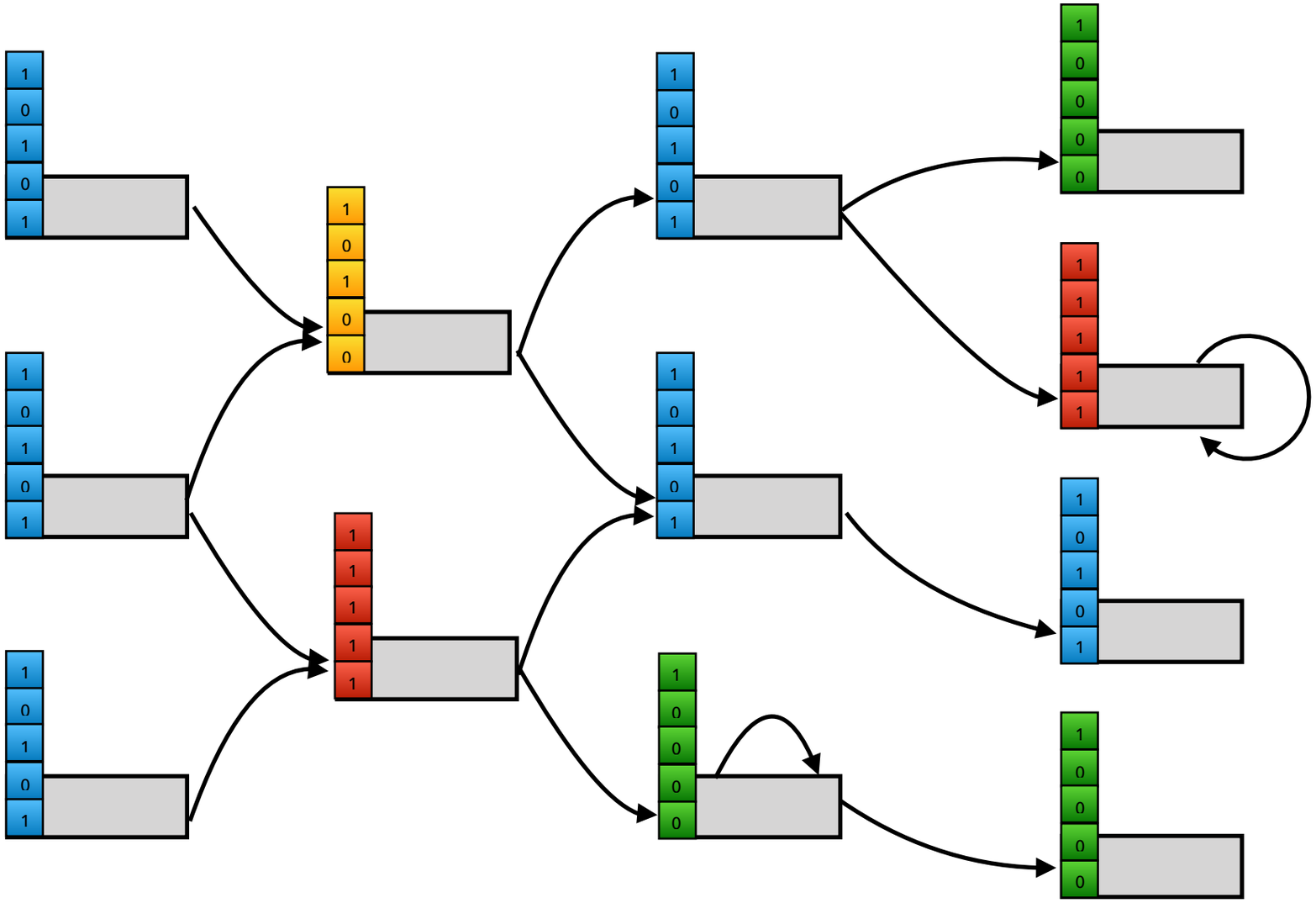
dBG common in genomics. Nodes u,v are k-mers & are *adjacent* if k-1 suffix of u is the same as k-1 prefix of v

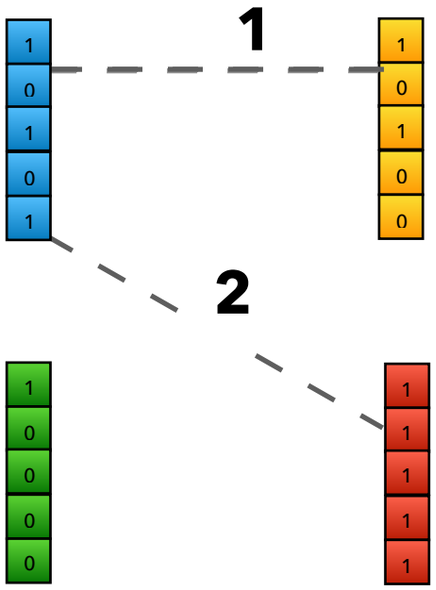




Use the **de Bruin graph** (dBG) as an efficient guide for near-neighbor search in the space of color classes!

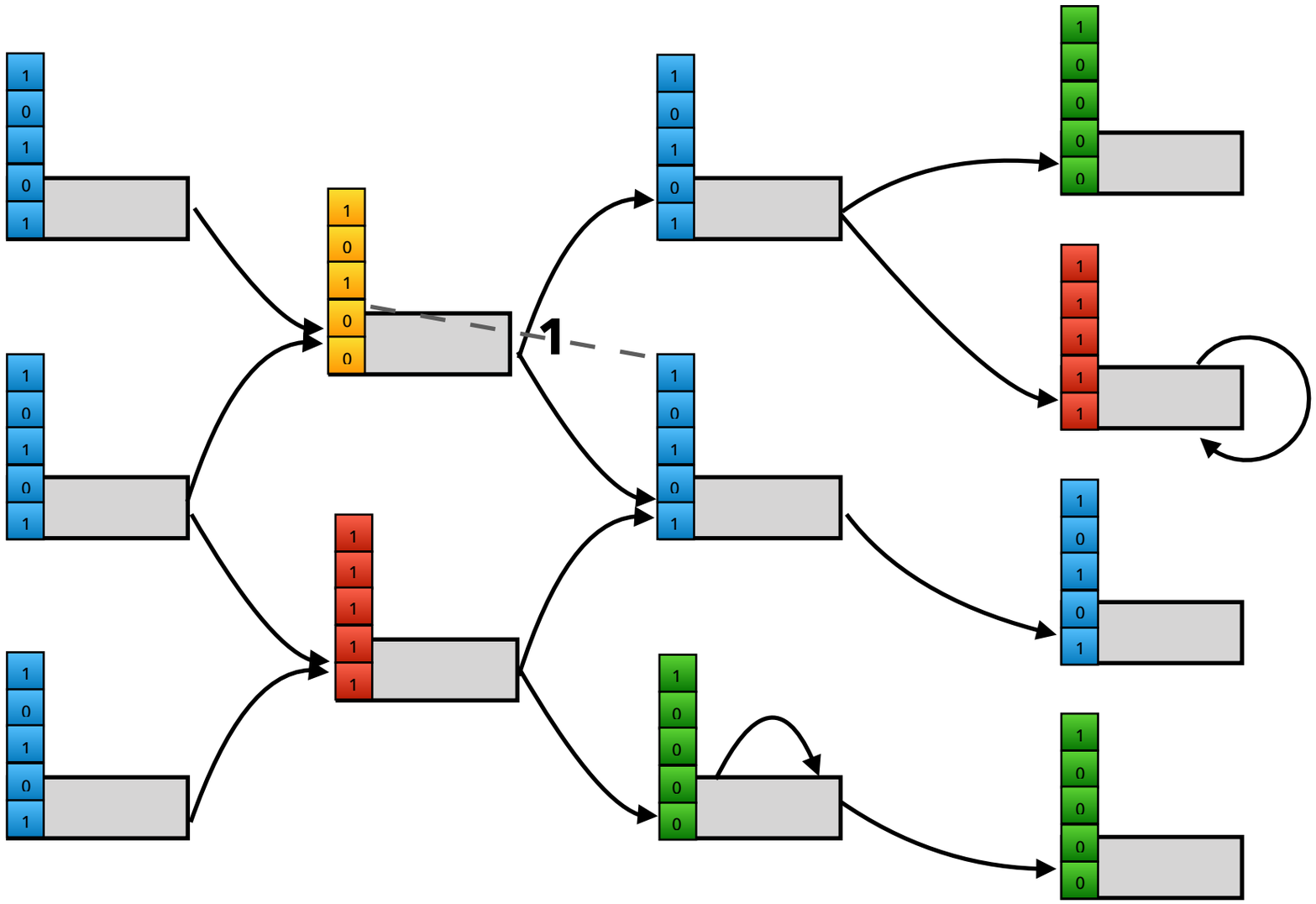
dBG common in genomics. Nodes u,v are k-mers & are *adjacent* if k-1 suffix of u is the same as k-1 prefix of v

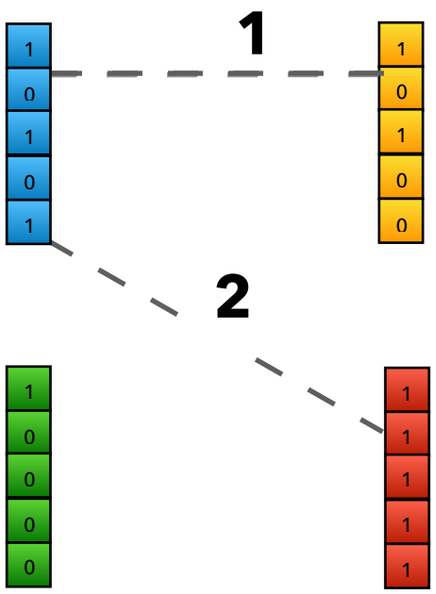




Use the **de Bruin graph** (dBG) as an efficient guide for near-neighbor search in the space of color classes!

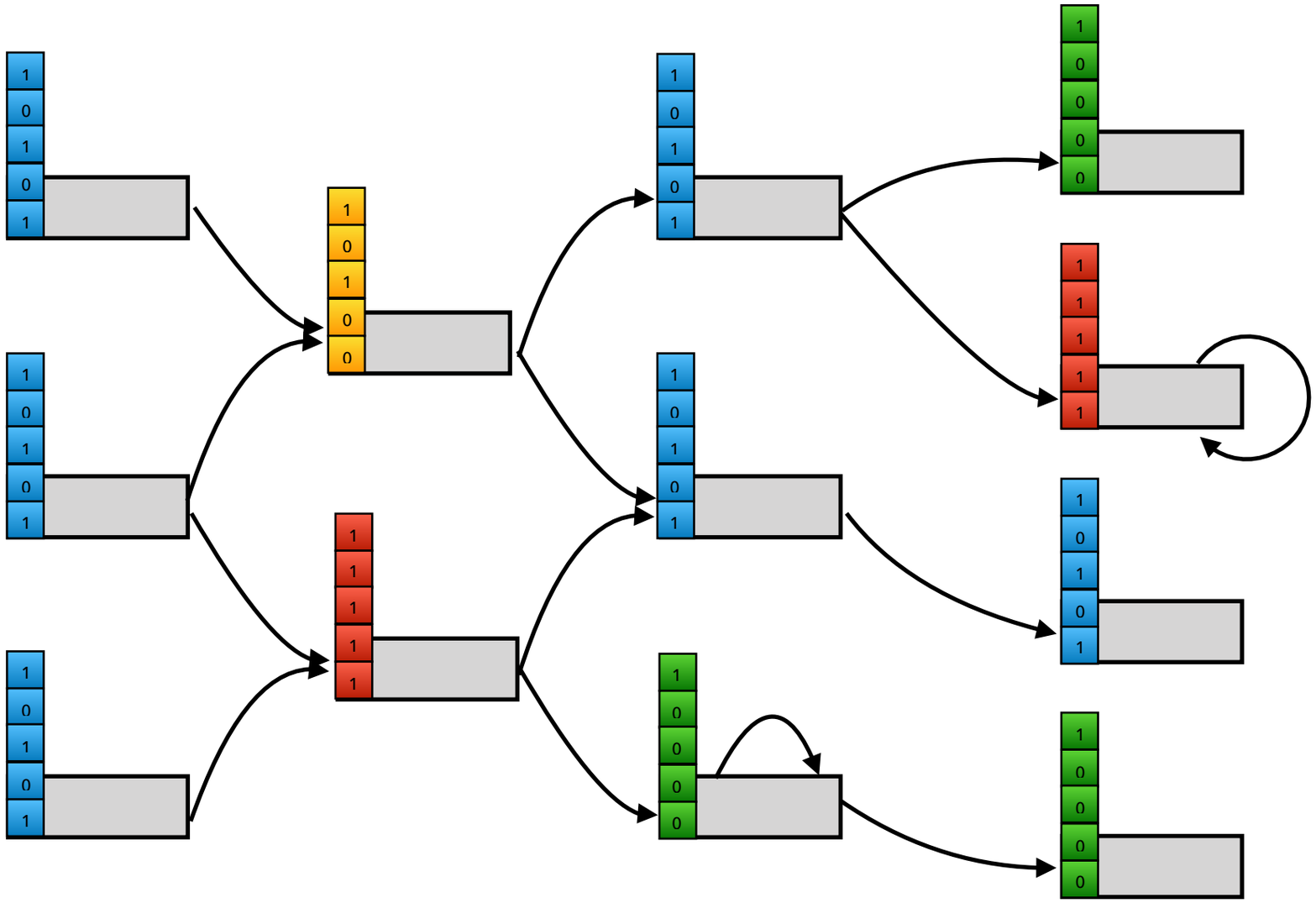
dBG common in genomics. Nodes u,v are k-mers & are *adjacent* if k-1 suffix of u is the same as k-1 prefix of v

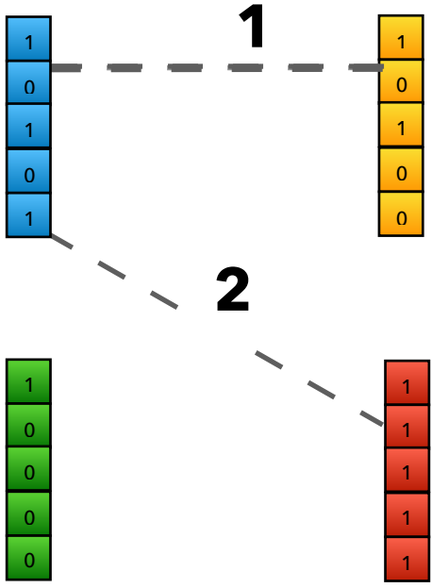




Use the **de Bruin graph** (dBG) as an efficient guide for near-neighbor search in the space of color classes!

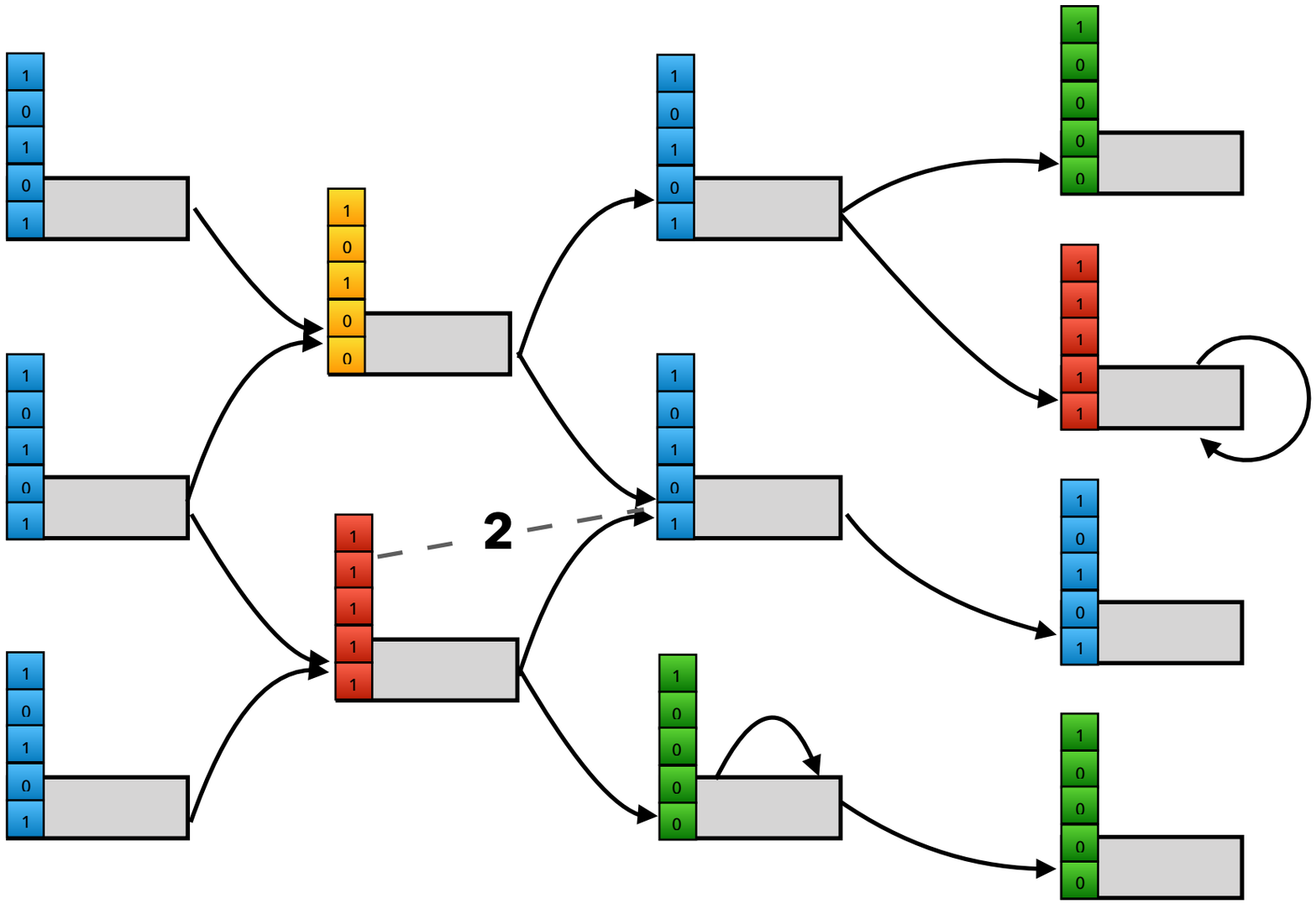
dBG common in genomics. Nodes u,v are k-mers & are *adjacent* if k-1 suffix of u is the same as k-1 prefix of v

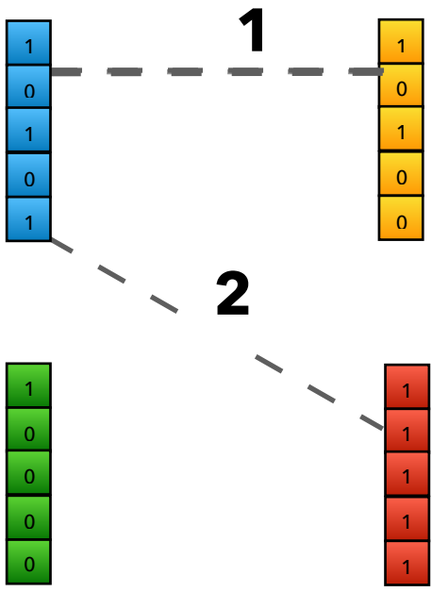




Use the **de Bruin graph** (dBG) as an efficient guide for near-neighbor search in the space of color classes!

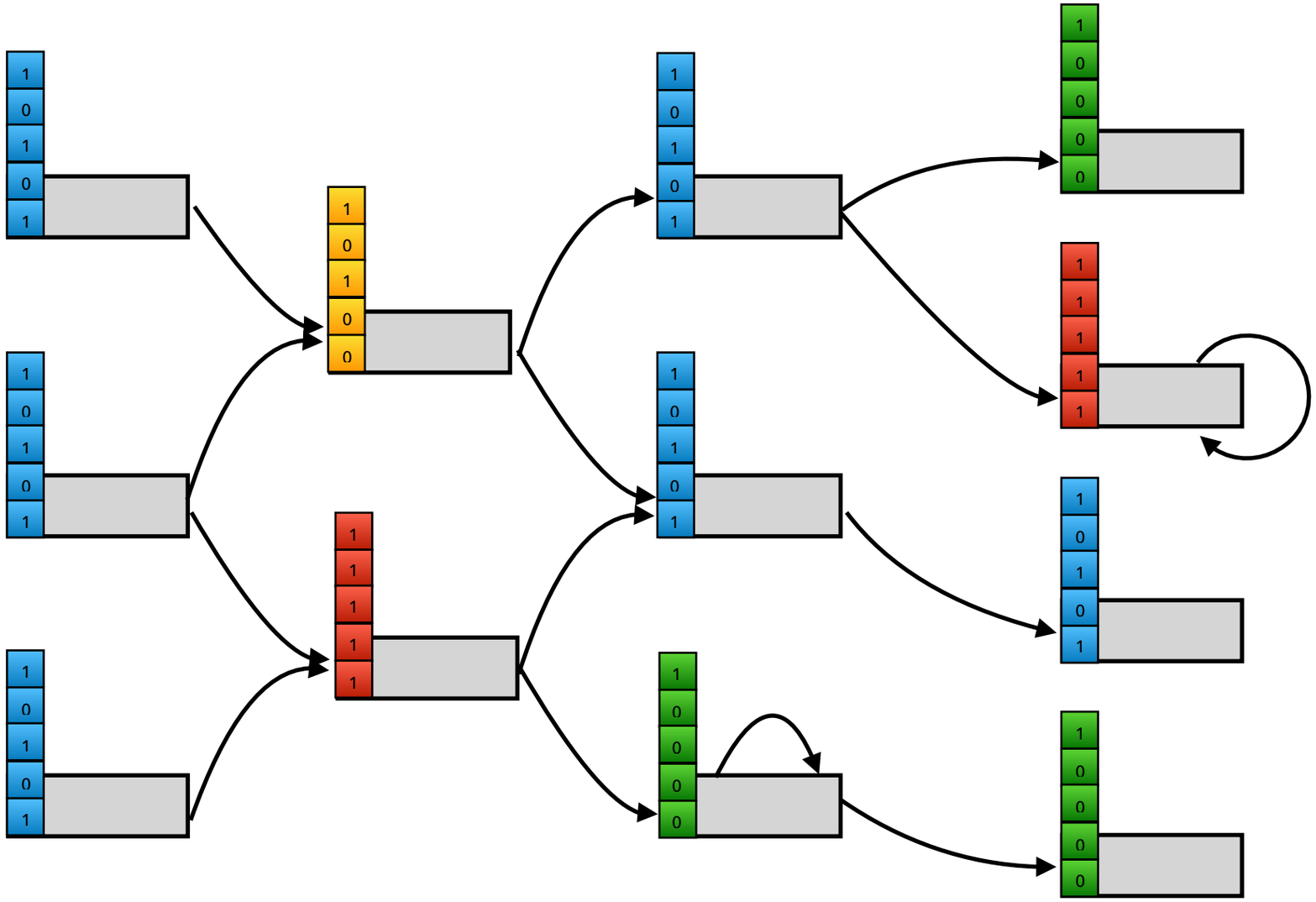
dBG common in genomics. Nodes u,v are k-mers & are *adjacent* if k-1 suffix of u is the same as k-1 prefix of v

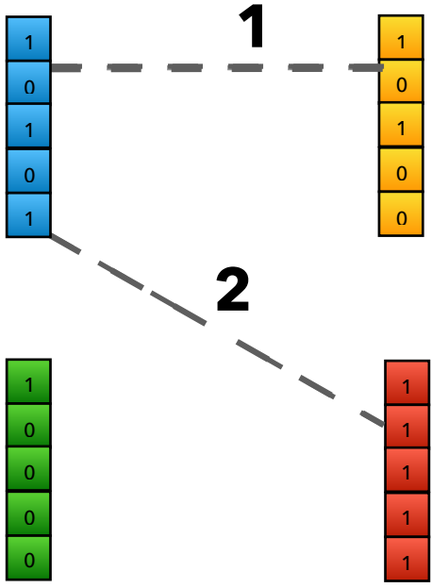




Use the **de Bruin graph** (dBG) as an efficient guide for near-neighbor search in the space of color classes!

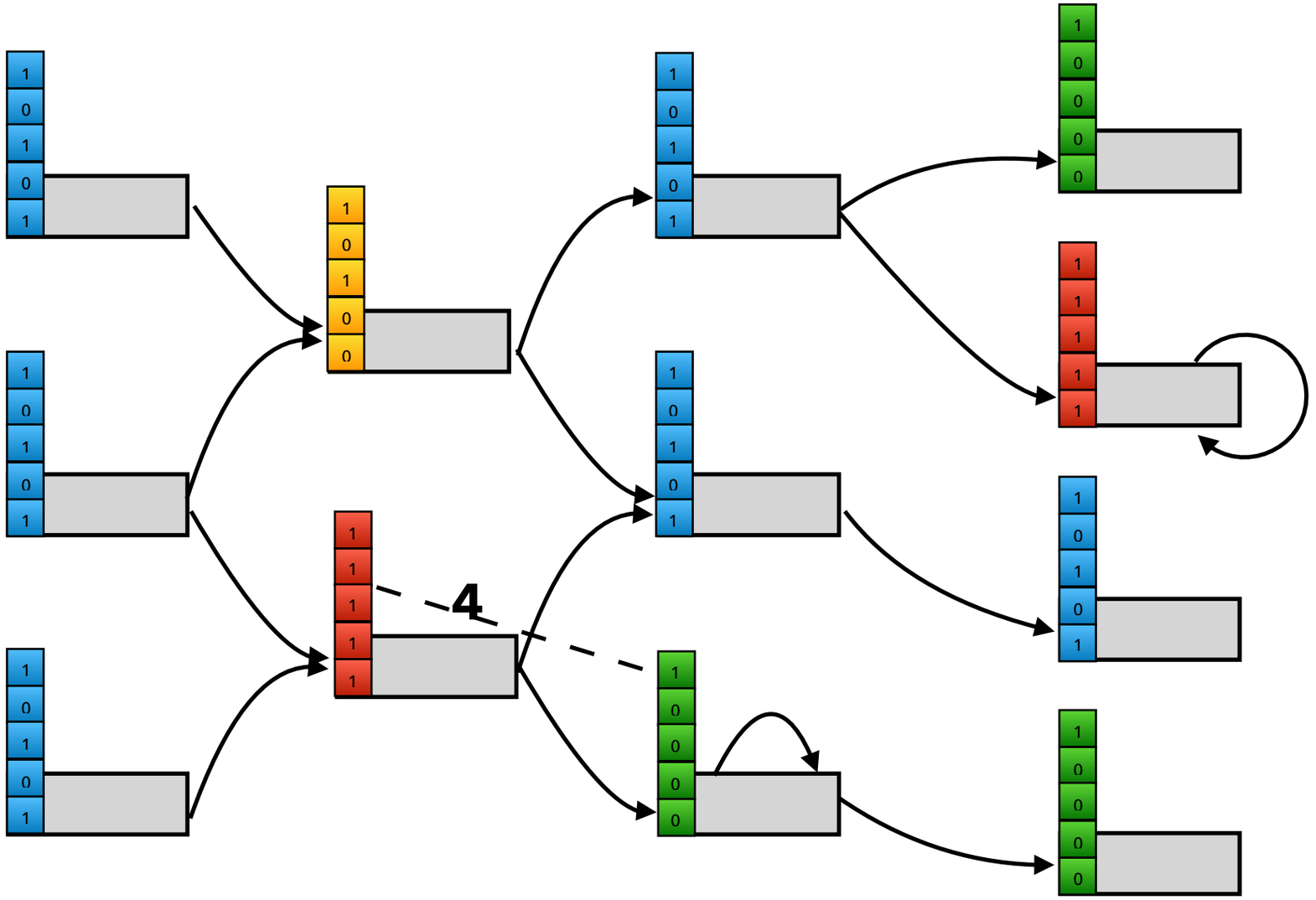
dBG common in genomics. Nodes u,v are k-mers & are *adjacent* if k-1 suffix of u is the same as k-1 prefix of v

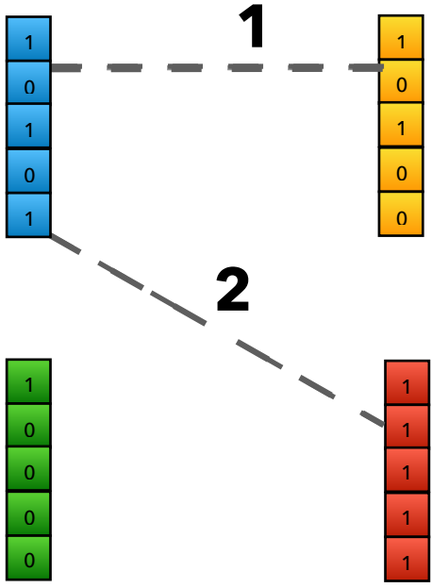




Use the **de Bruin graph** (dBG) as an efficient guide for near-neighbor search in the space of color classes!

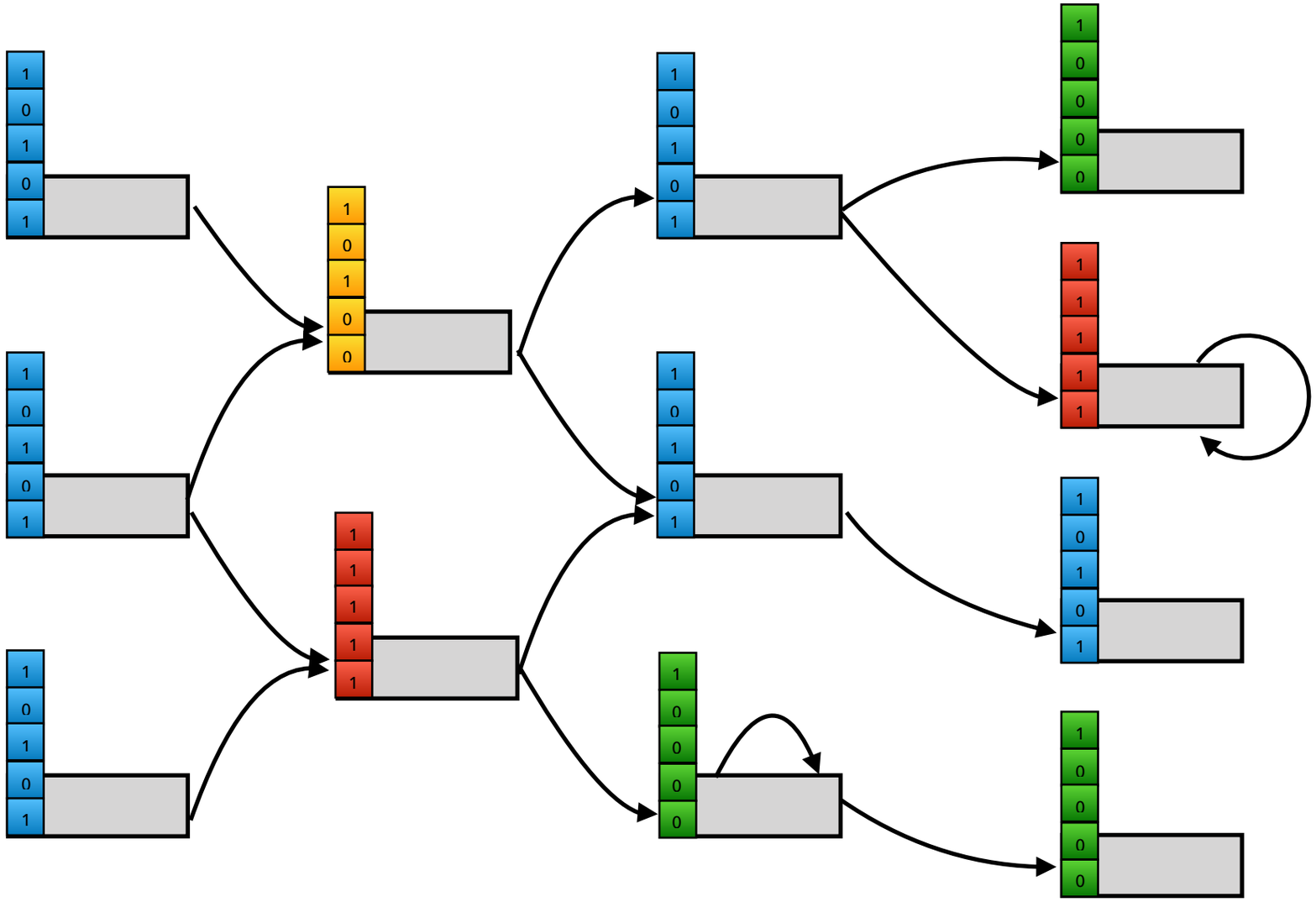
dBG common in genomics. Nodes u,v are k-mers & are *adjacent* if k-1 suffix of u is the same as k-1 prefix of v

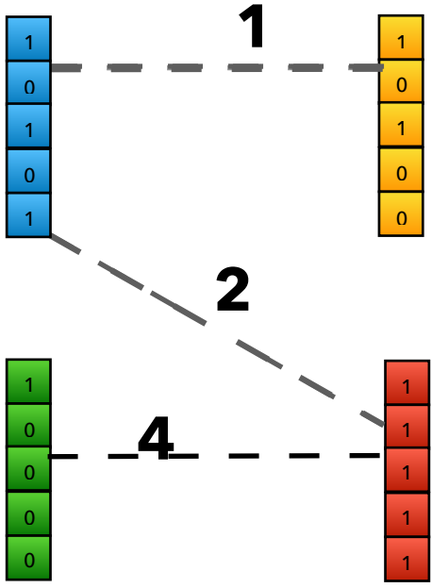




Use the **de Bruin graph** (dBG) as an efficient guide for near-neighbor search in the space of color classes!

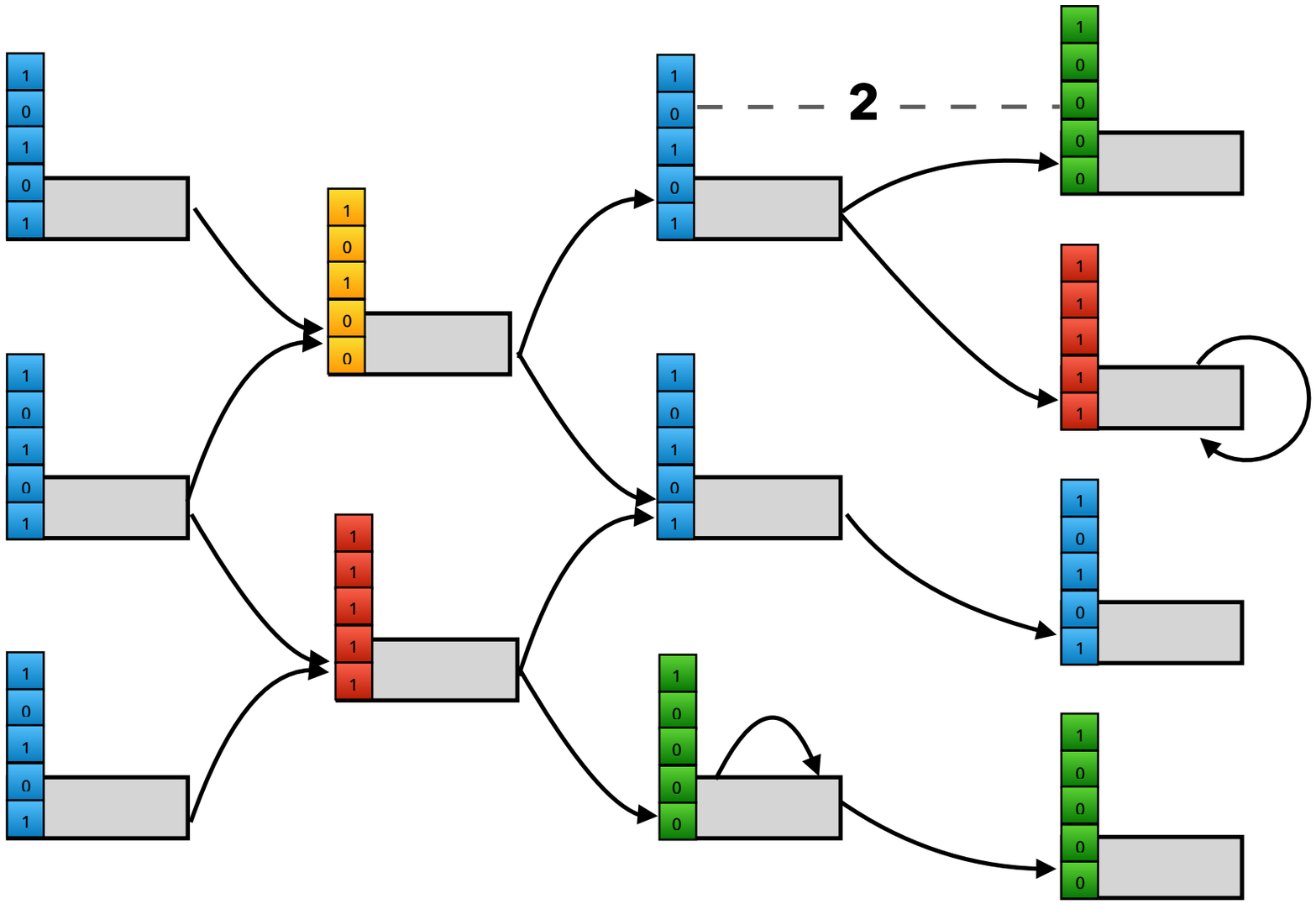
dBG common in genomics. Nodes u,v are k-mers & are *adjacent* if k-1 suffix of u is the same as k-1 prefix of v

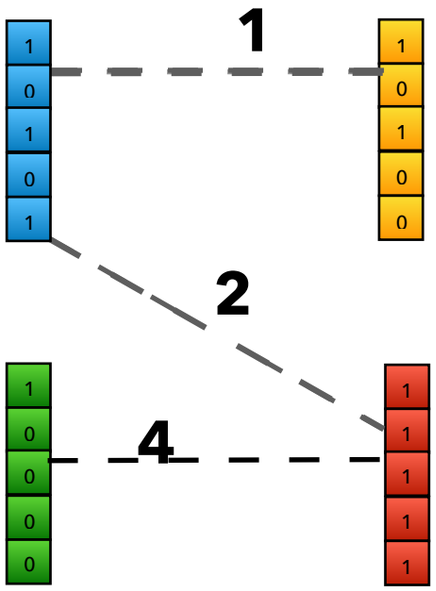




Use the **de Bruin graph** (dBG) as an efficient guide for near-neighbor search in the space of color classes!

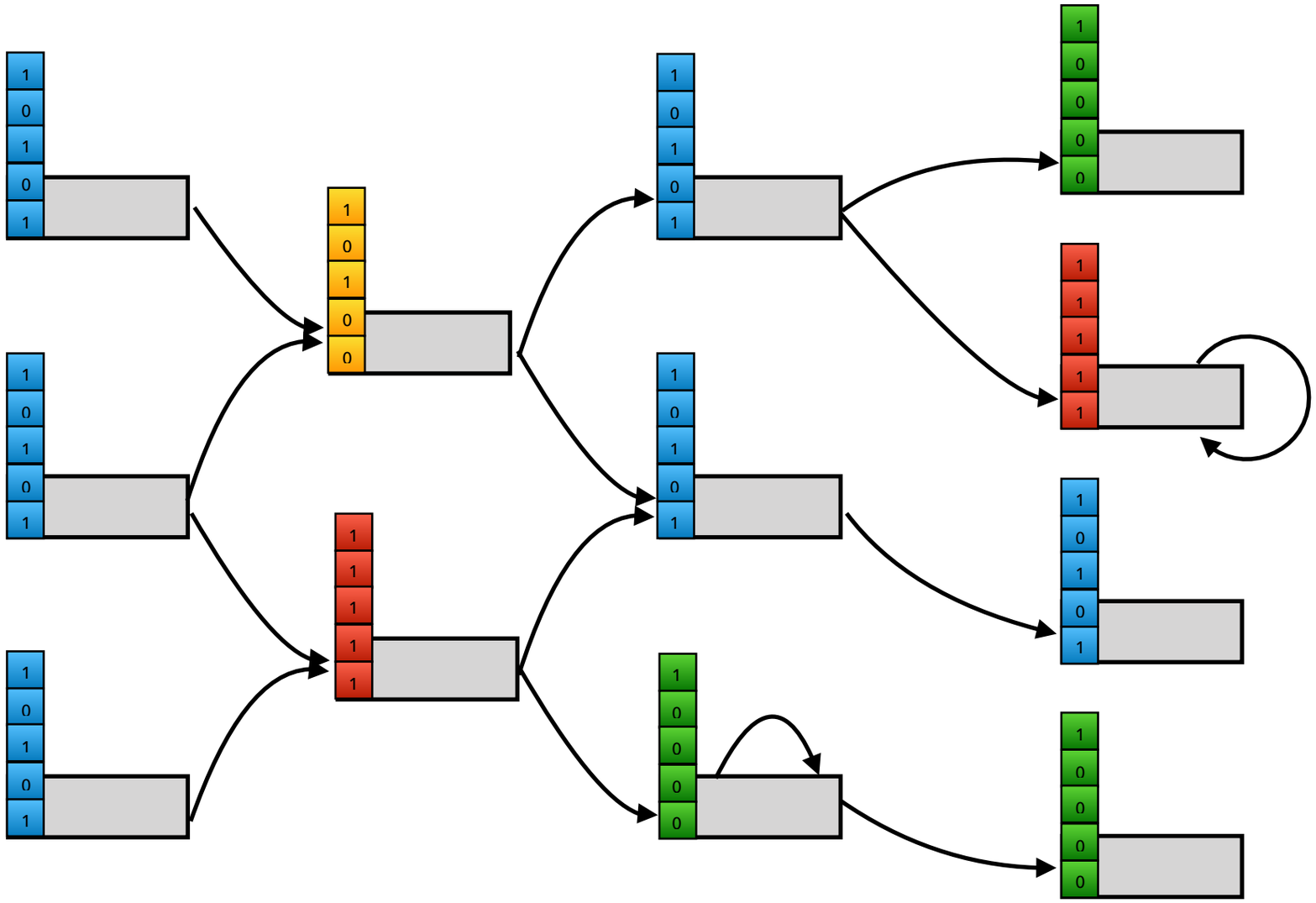
dBG common in genomics. Nodes u,v are k-mers & are *adjacent* if k-1 suffix of u is the same as k-1 prefix of v

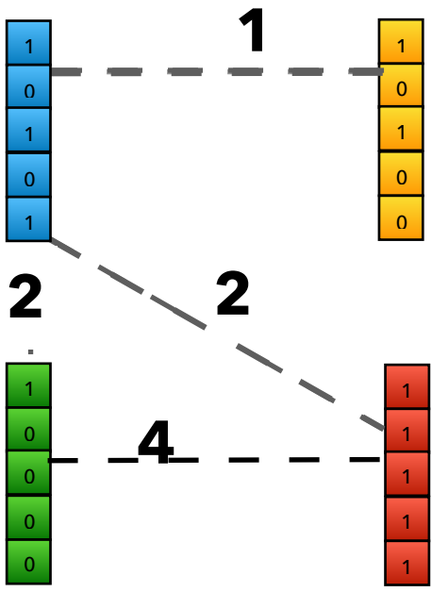




Use the **de Bruin graph** (dBG) as an efficient guide for near-neighbor search in the space of color classes!

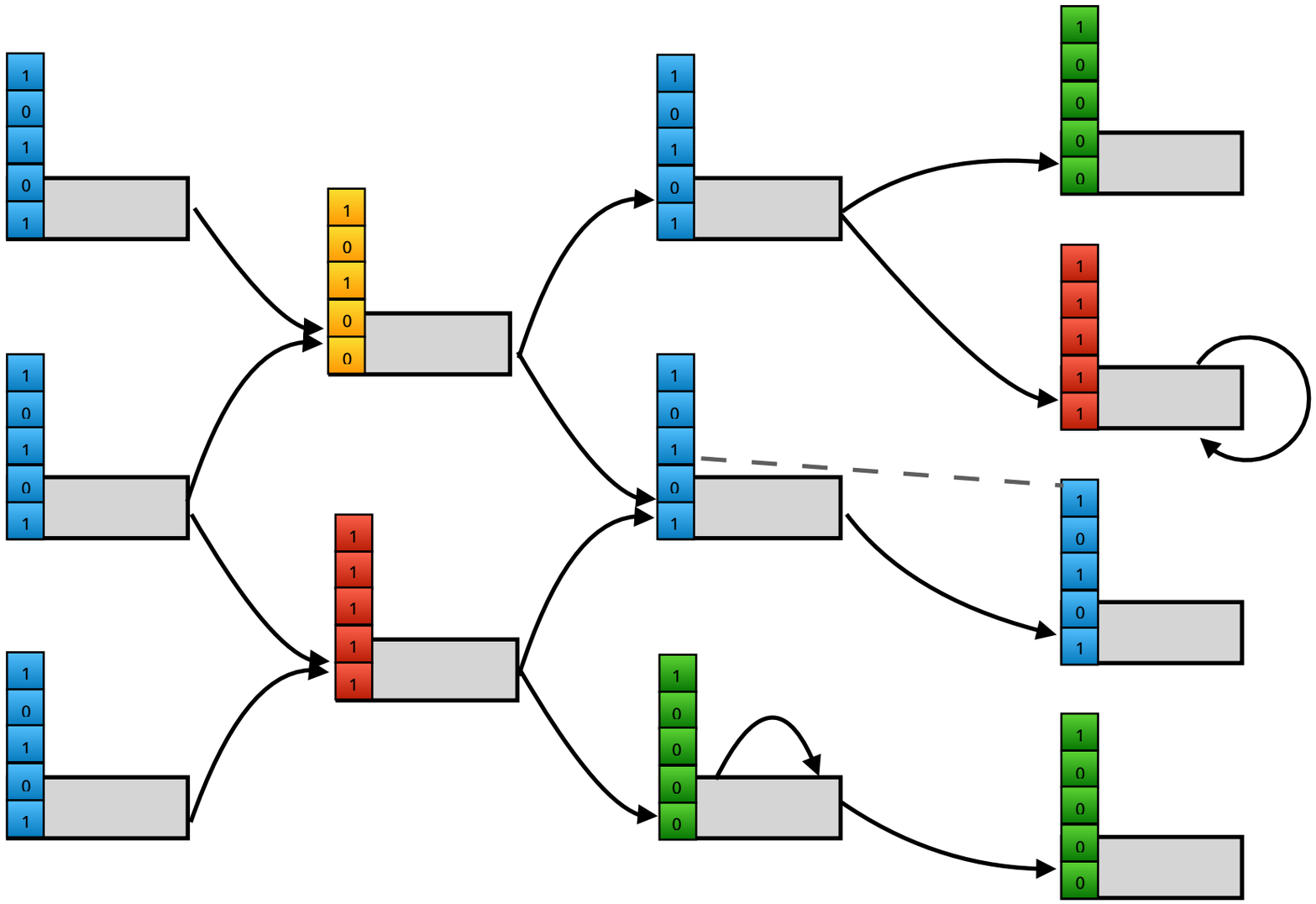
dBG common in genomics. Nodes u,v are k-mers & are *adjacent* if k-1 suffix of u is the same as k-1 prefix of v

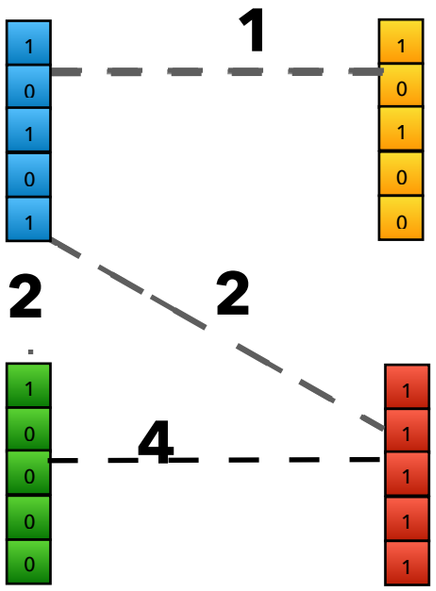




Use the **de Bruin graph** (dBG) as an efficient guide for near-neighbor search in the space of color classes!

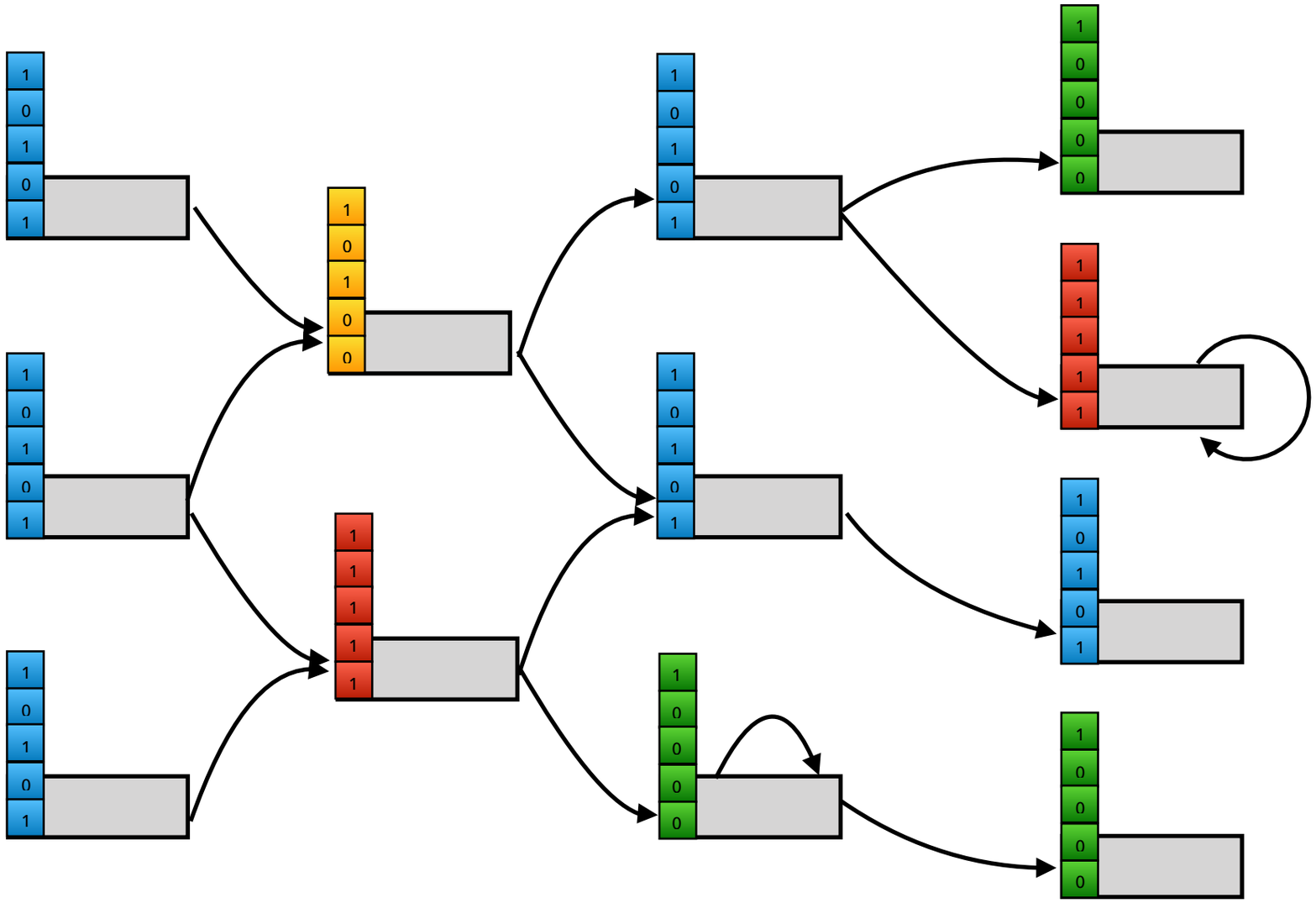
dBG common in genomics. Nodes u,v are k-mers & are *adjacent* if k-1 suffix of u is the same as k-1 prefix of v

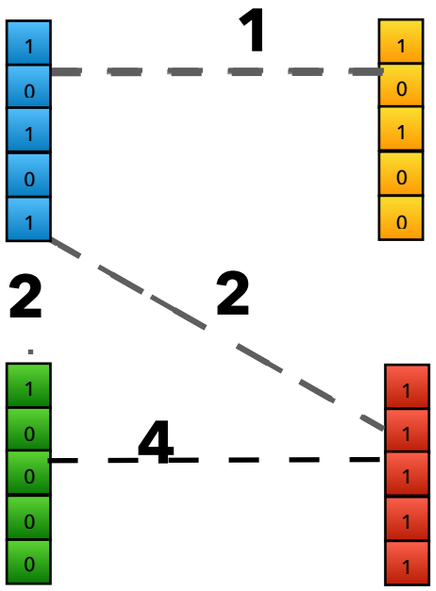




Use the **de Bruin graph** (dBG) as an efficient guide for near-neighbor search in the space of color classes!

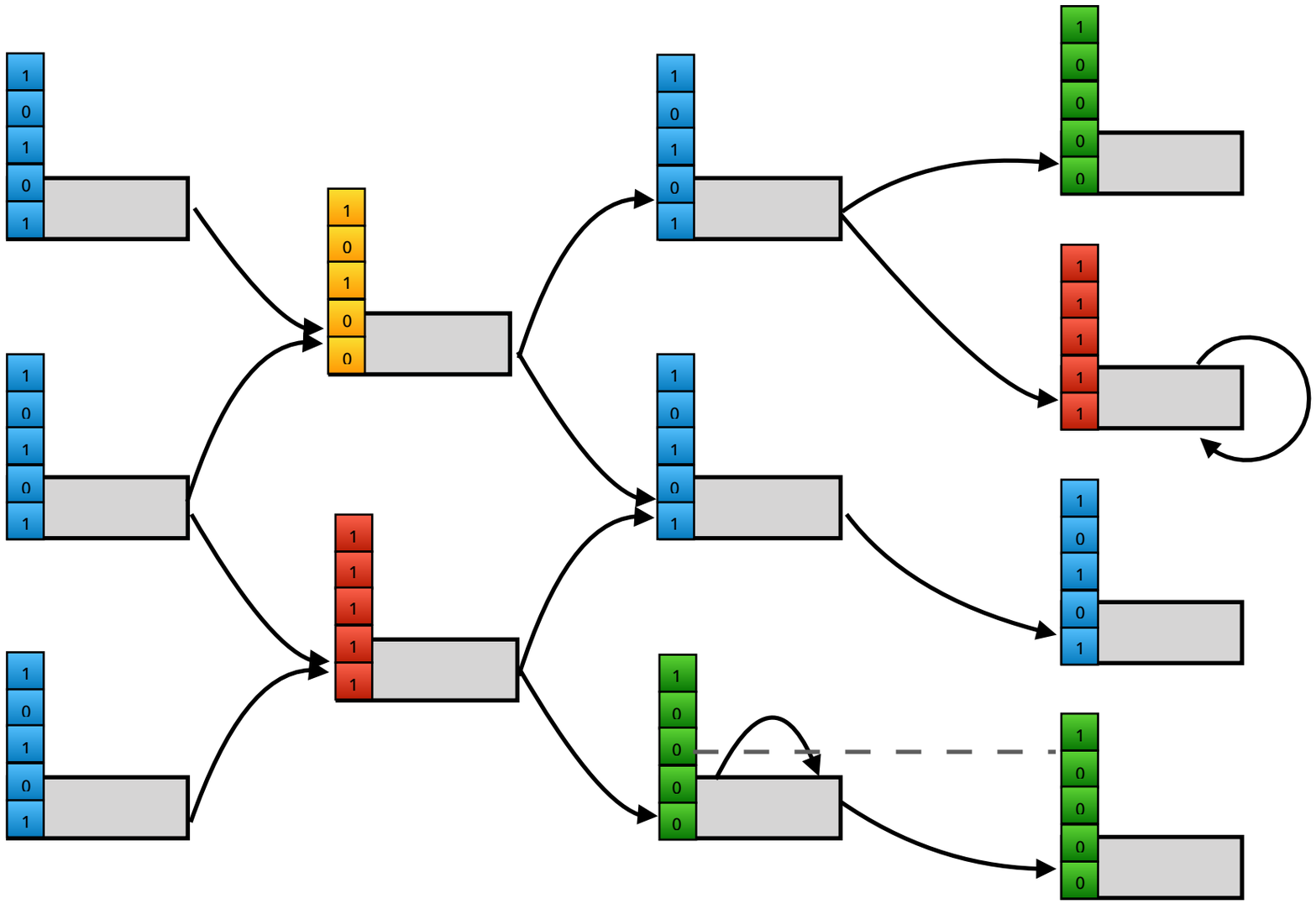
dBG common in genomics. Nodes u,v are k-mers & are *adjacent* if k-1 suffix of u is the same as k-1 prefix of v

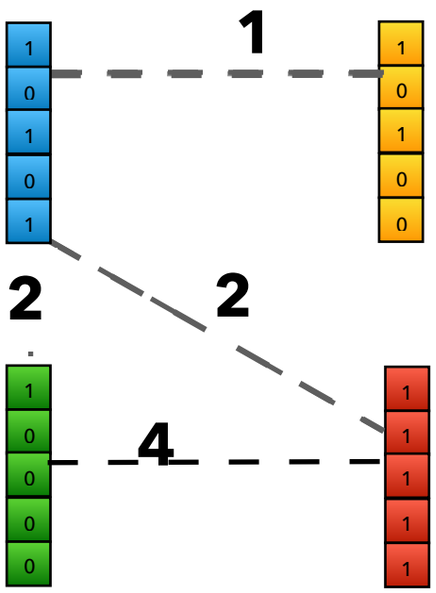




Use the **de Bruin graph** (dBG) as an efficient guide for near-neighbor search in the space of color classes!

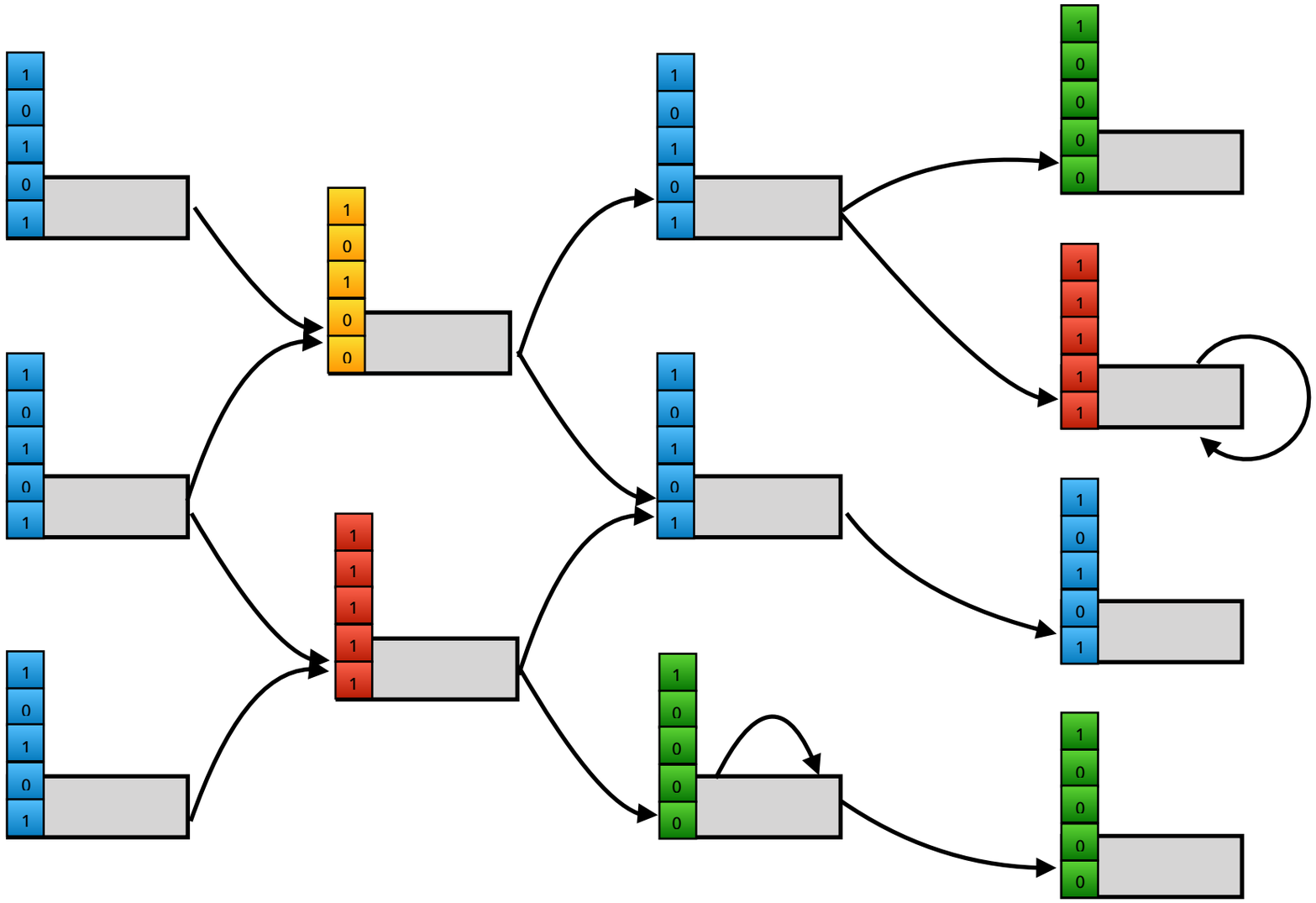
dBG common in genomics. Nodes u,v are k-mers & are *adjacent* if k-1 suffix of u is the same as k-1 prefix of v

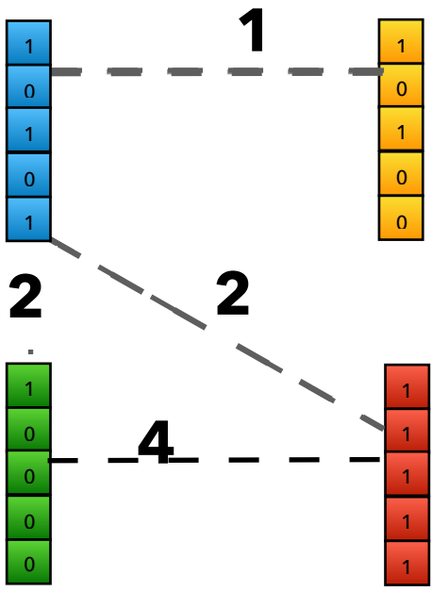




Use the **de Bruin graph** (dBG) as an efficient guide for near-neighbor search in the space of color classes!

dBG common in genomics. Nodes u,v are k-mers & are *adjacent* if k-1 suffix of u is the same as k-1 prefix of v

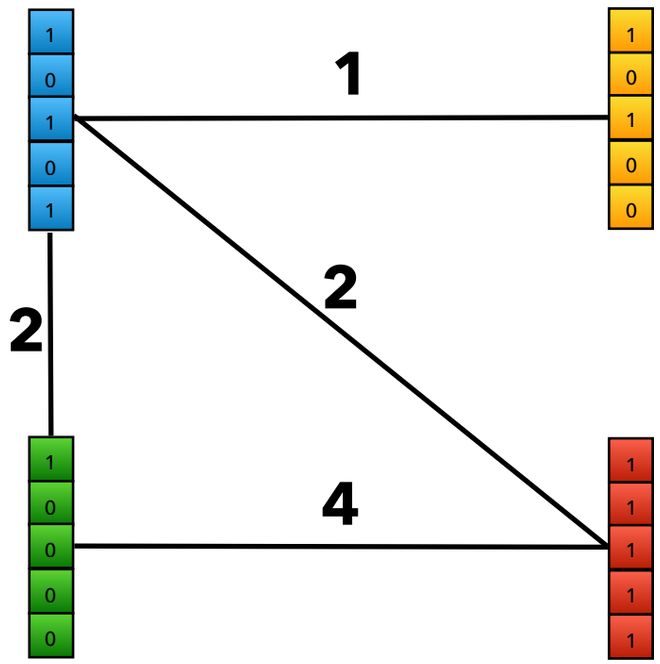




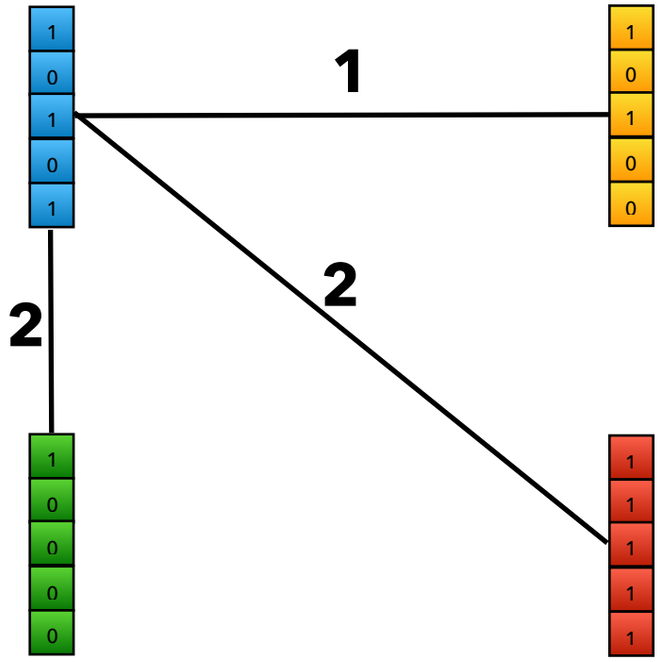
Use the **de Bruin graph** (dBG) as an efficient guide for near-neighbor search in the space of color classes!

dBG common in genomics. Nodes u,v are k-mers & are *adjacent* if k-1 suffix of u is the same as k-1 prefix of v

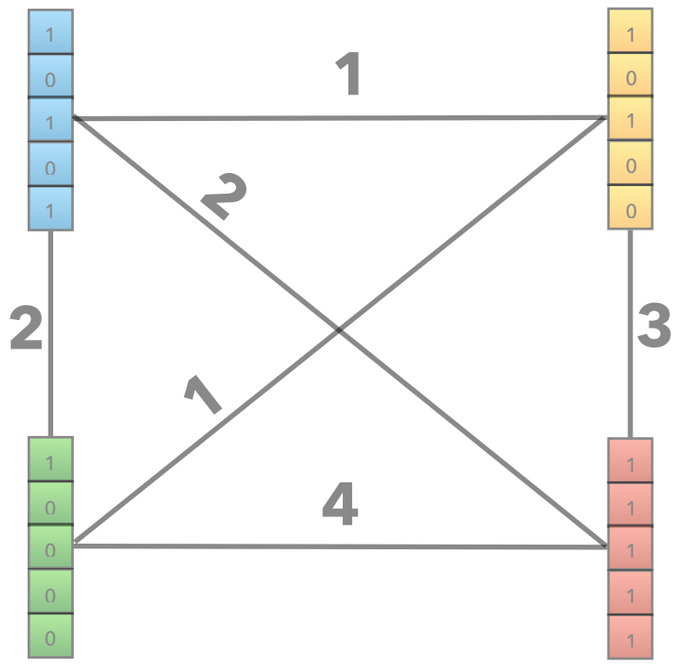
CCG derived from dbG



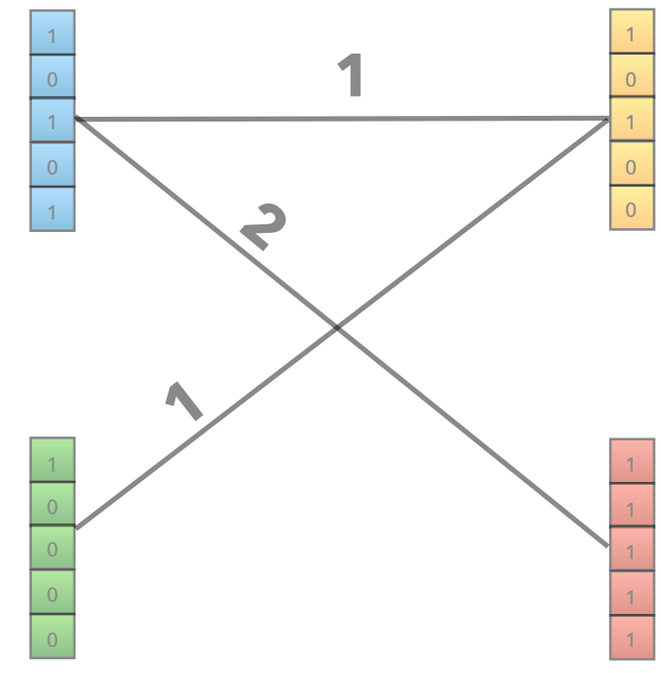
MST on our Graph



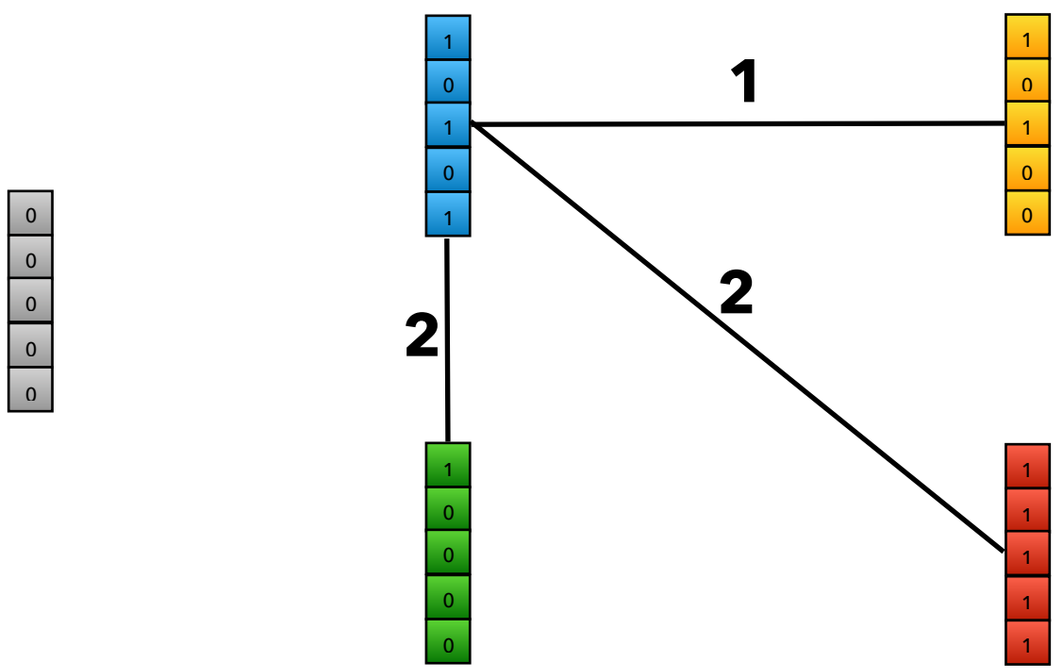
Complete CCG



Optimal MST

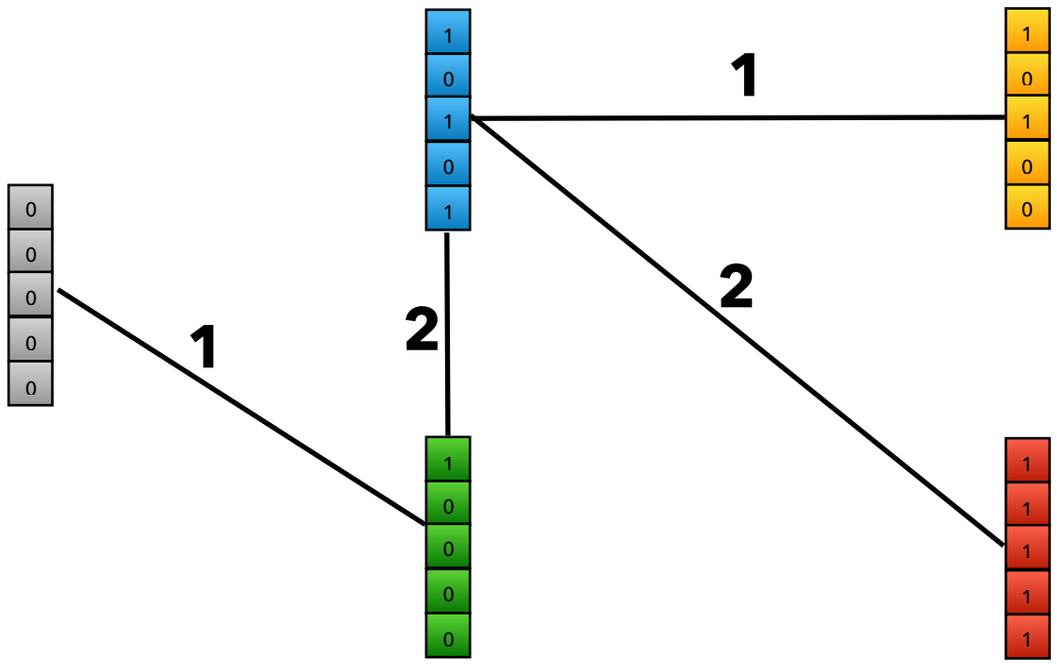


The MST efficiently encodes related color classes



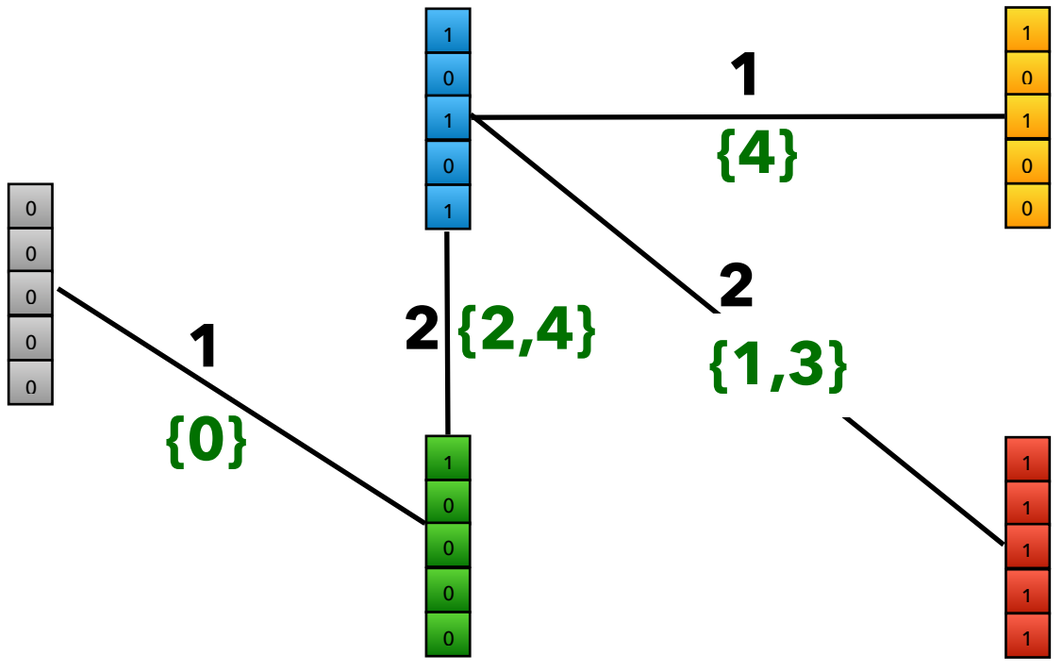
The MST efficiently encodes related color classes

Augment with all 0 color class to guarantee one, connected MST



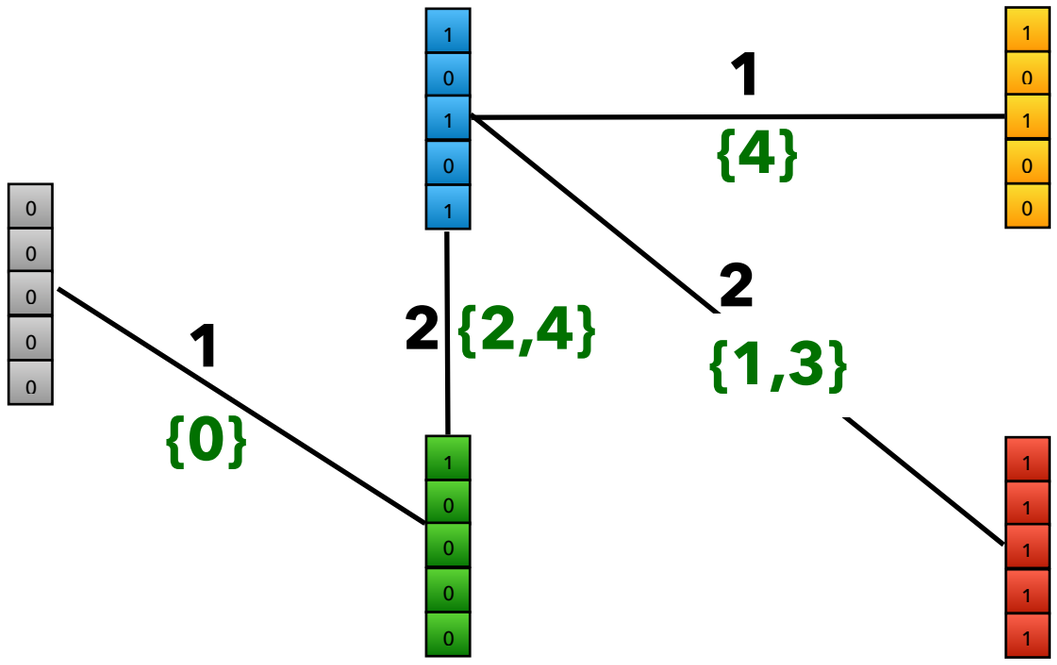
The MST efficiently encodes related color classes

Augment with all 0 color class to guarantee one, connected MST



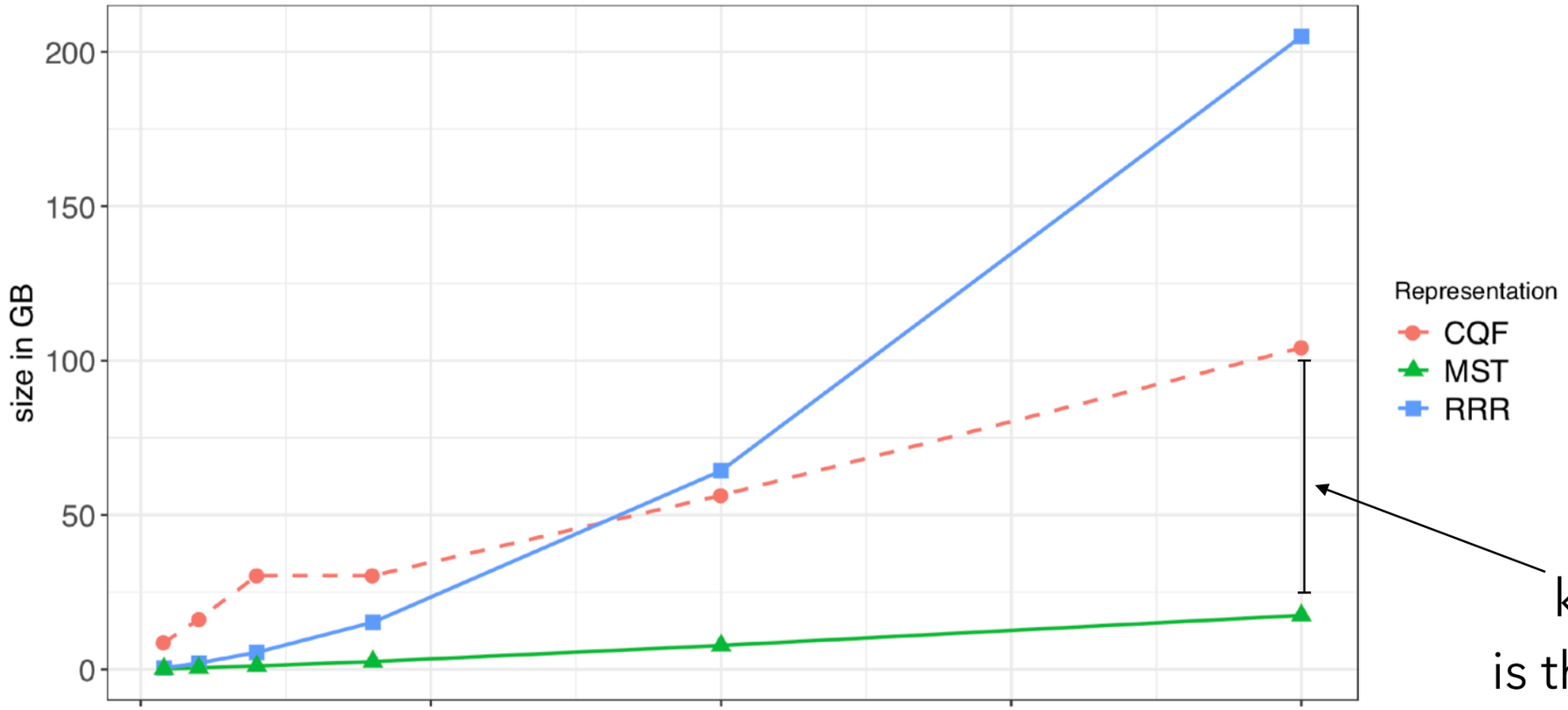
The MST efficiently encodes related color classes

Augment with all 0 color class to guarantee one, connected MST



To reconstruct a vector, walk from your node to the root, flipping the parity of the positions you encounter on each edge.

The MST approach scales very well



now the
k-mer table
is the bottleneck

Dataset	# samples	RRR matrix	MST Total space	MST Parent vector	MST Delta vector	MST Boundary bit-vector	$\frac{size(MST)}{size(RRR)}$
H. sapiens RNA-seq samples	200	0.42	0.15	0.08	0.06	0.01	0.37
	500	1.89	0.46	0.2	0.24	0.03	0.24
	1,000	5.14	1.03	0.37	0.6	0.06	0.2
	2,000	14.2	2.35	0.71	1.5	0.14	0.17
	5,000	59.89	7.21	1.72	5.1	0.39	0.12
	10,000	190.89	16.28	3.37	12.06	0.86	0.085
Blood, Brain, Breast (BBB)	2586	15.8	2.66	0.63	1.88	0.16	0.17

Improvement
over RRR improves
with # of samples

dataset from SBT / SSBT / Mantis paper

How does MST approach affect query time?

One concern is that replacing O(1) lookup with
MST-based decoding will make lookup slow; does it?

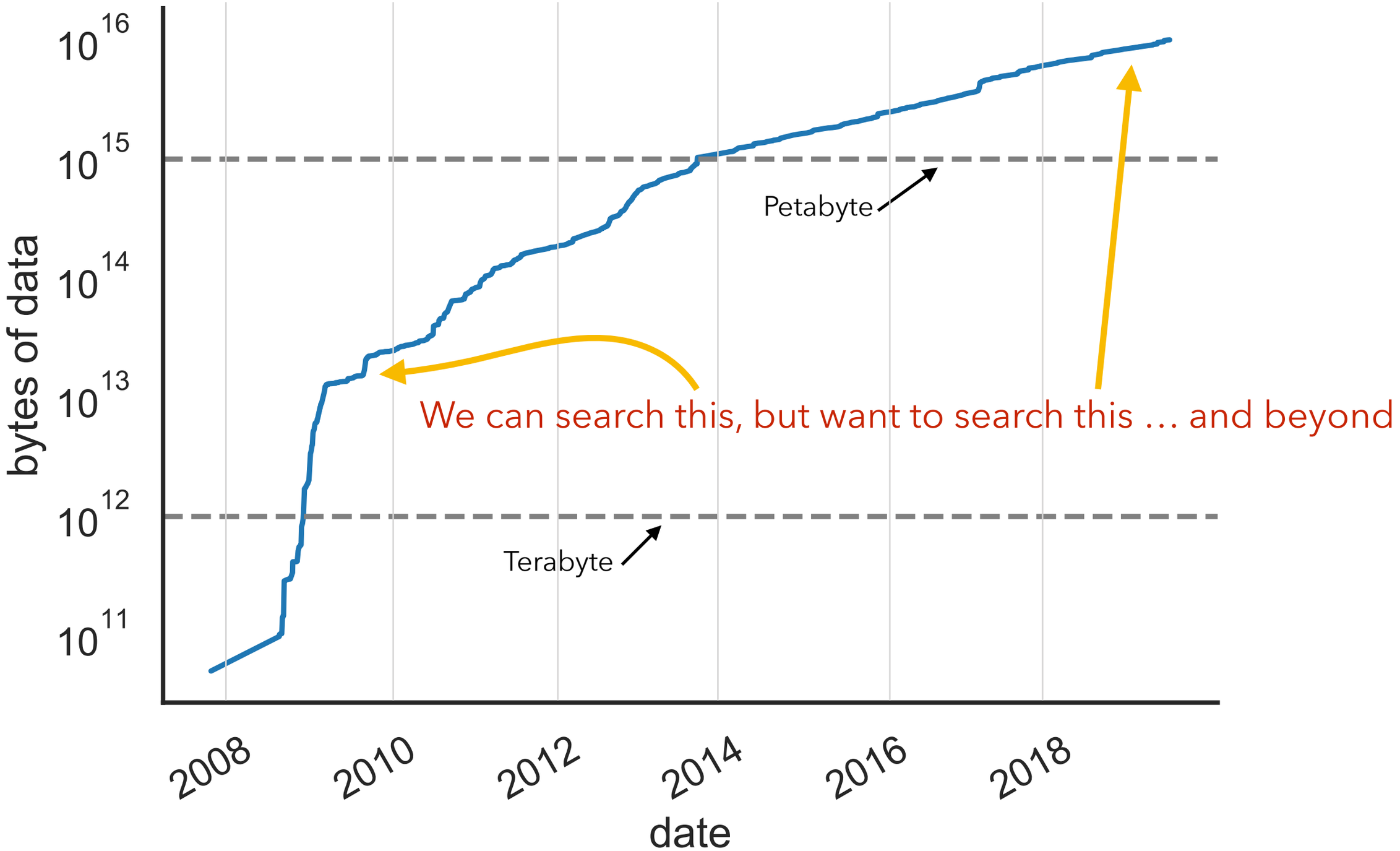
How does MST approach affect query time?

One concern is that replacing O(1) lookup with
MST-based decoding will make lookup slow; does it?

Turns out a caching strategy (an LRU over popular internal nodes)
keeps it just as fast as lookup in the RRR matrix

	Mantis with MST			Mantis		
	index load + query	query	space	index load + query	query	space
10 Transcripts	1 min 10 sec	0.3 sec	118GB	32 min 59 sec	0.5 sec	290GB
100 Transcripts	1 min 17 sec	8 sec	119GB	34 min 33 sec	11 sec	290GB
1000 Transcripts	2 min 29 sec	79 sec	120GB	46 min 4 sec	80 sec	290GB

A Call To Arms



> "It seems that some essentially new … ideas are here needed"
> — Paul Adrien Maurice Dirac*

*Principles of Quantum Mechanics 2nd edition, Chapter XIII, Section 81 (p. 297)

Data from: https://www.ncbi.nlm.nih.gov/Traces/sra/sra_stat.cgi