Universität Bielefeld

Computational Methods for the Analysis of the Diversity and Dynamics of Genomes

Efficient querying of colored de Bruijn graphs

Tizian Schulz

(joint work with Guillaume Holley and Jens Stoye)

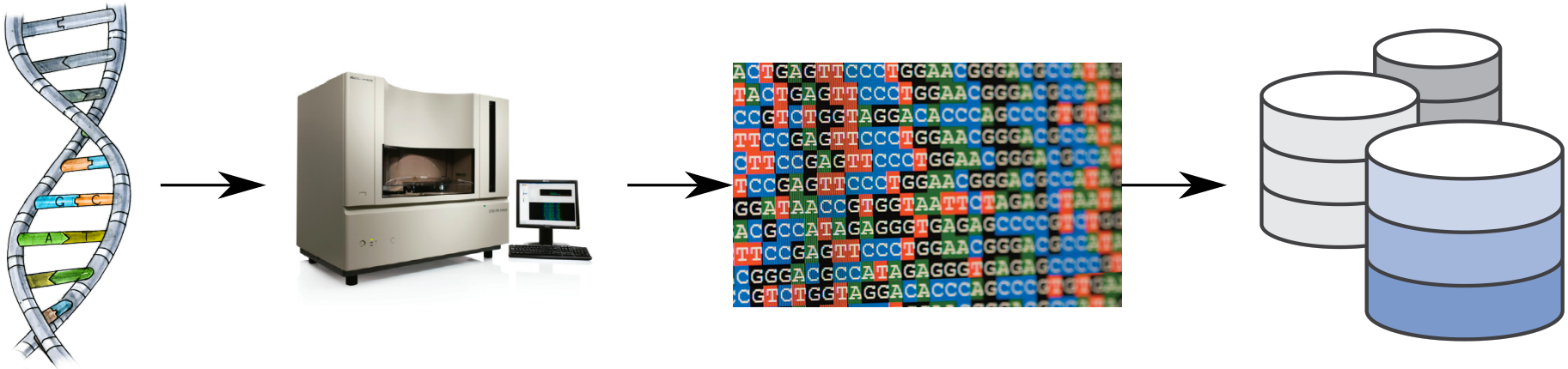
Faculty of Technology, Bielefeld University

February 21, 2018

Contents

Motivation

DNA sequencing in former times:



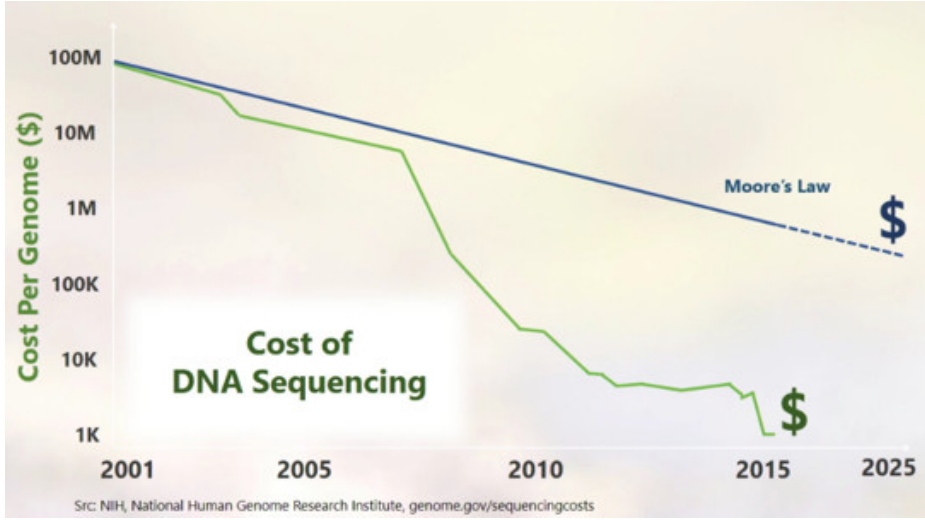
Especially, short DNA sequences were sequenced

- Single genes
- Viral genomes
- Small bacterial genomes

Motivation

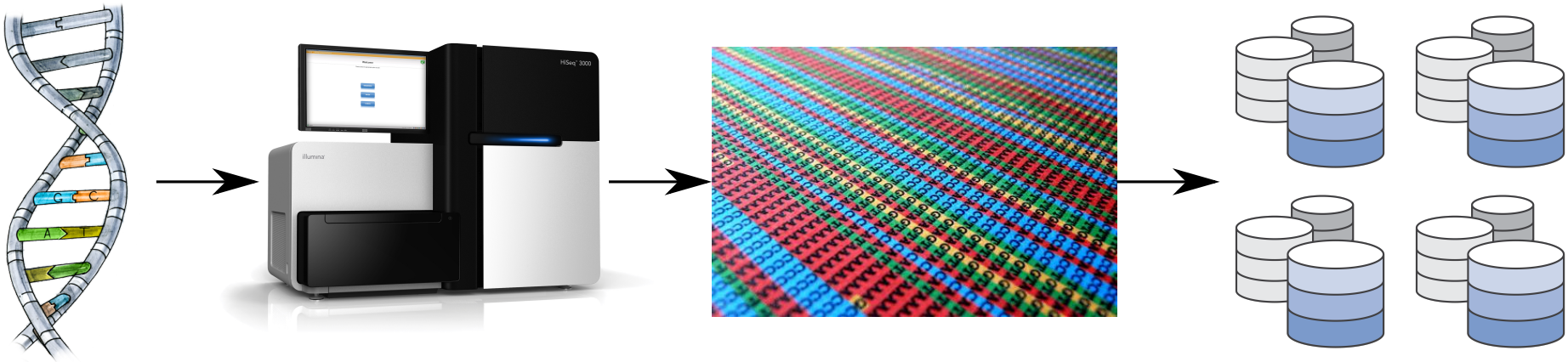
Current development:

- Sequencing technologies improve rapidly
- Sequencing becomes cheaper
- More and more sequencing projects are started



Motivation

DNA sequencing at the moment:

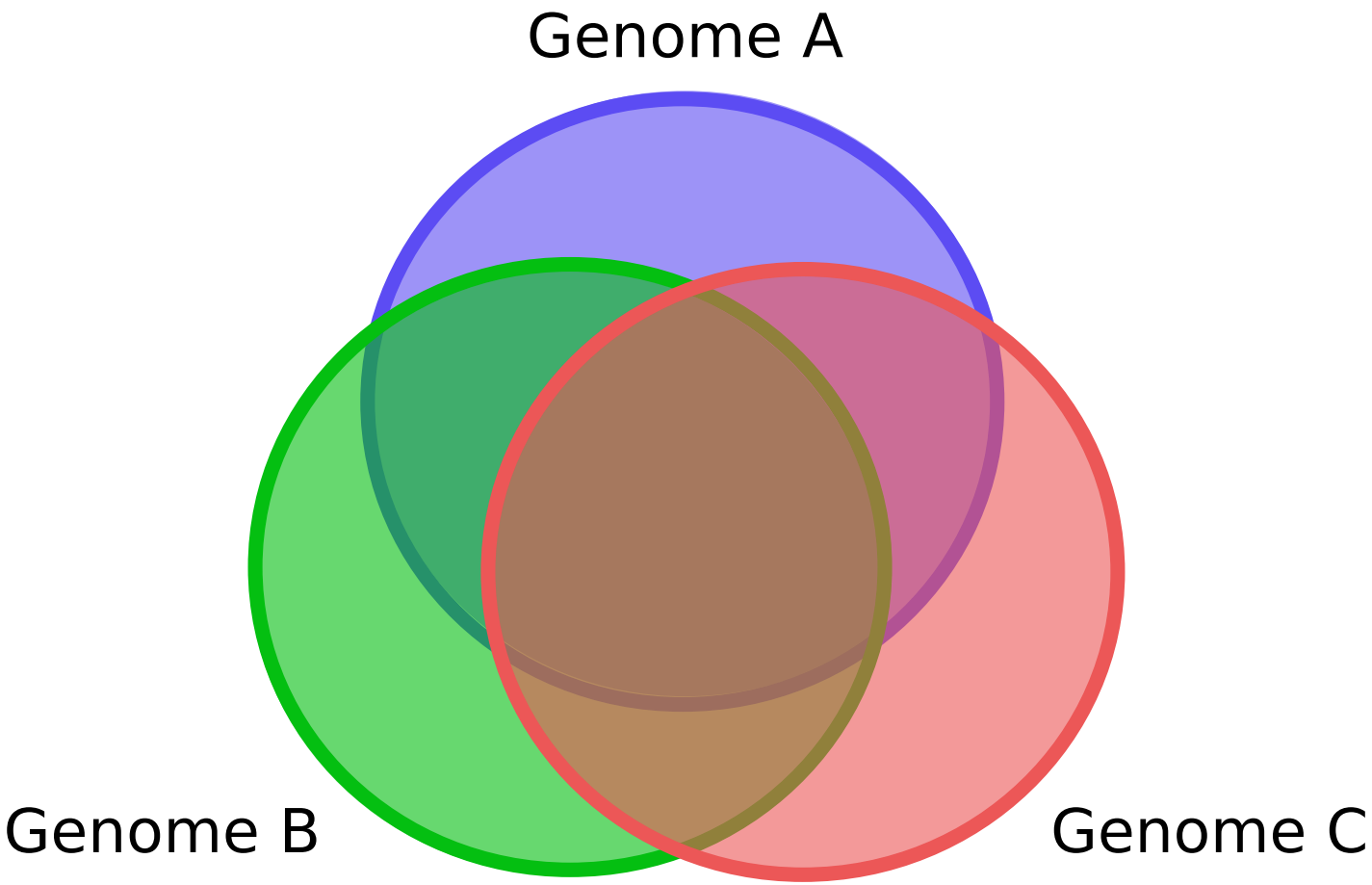


Long DNA sequences are sequenced:

- Whole genomes of complex plants and animals
- Many genomes per species

Motivation

How to store many highly similar DNA sequences, e. g. pan-genomes?



Colored de Bruijn graphs

Colored de Bruijn graphs

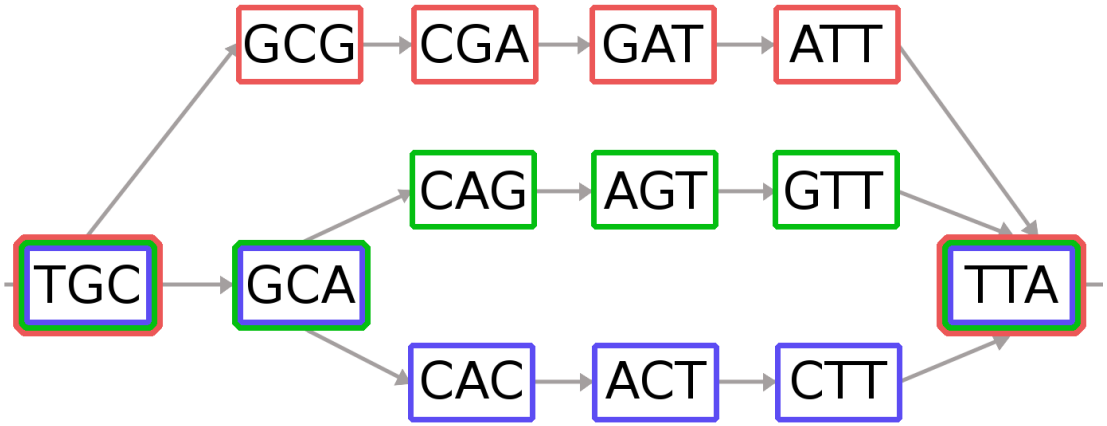
Colored de Bruijn graph (C-DBG)

$k = 3$

S_1 : TGCGATTA

S_2 : TGCAGTTA

S_3 : TGCACTTA



- Vertices represent substrings of length k (k-mers)
- Associated color represents sequence of origin
- Edges between vertices that share a $k-1$ overlap
- No explicit representation of edges necessary

Colored de Bruijn graphs

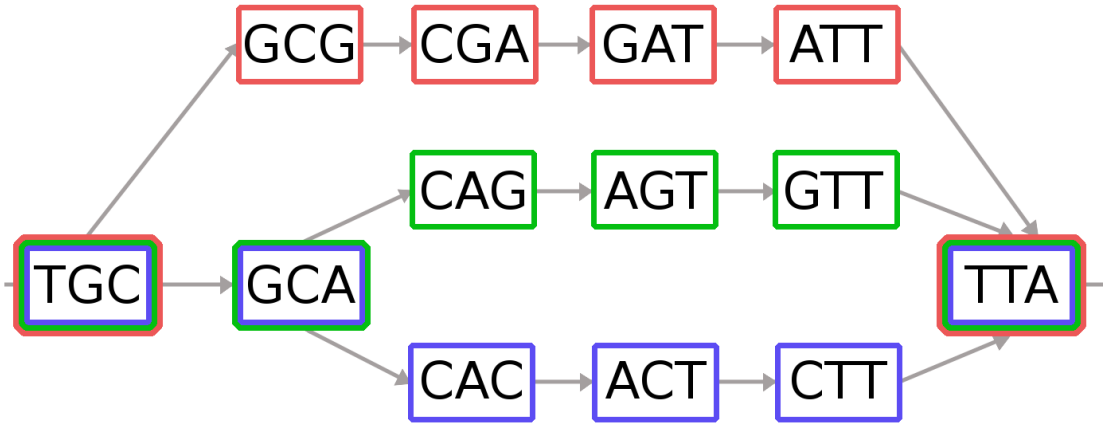
Colored de Bruijn graph (C-DBG)

$k = 3$

S_1 : TGCGATTA

S_2 : TGCAGTTA

S_3 : TGCACTTA



Graph allows:

1. Efficient storage of shared sequence parts
2. Reconstruction of sequences by graph traversal

Colored de Bruijn graphs

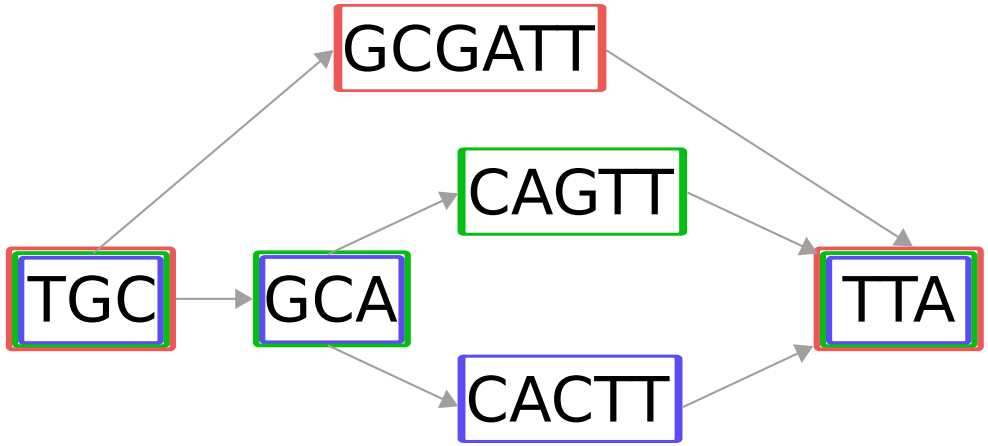
Compacted colored de Bruijn graphs

$k = 3$

S_1 : TGCGATTA

S_2 : TGCAGTTA

S_3 : TGCACTTA



- Vertices of unique paths are replaced by single vertex (unitig)
- Label of unitig is sequence spelled by the path

Colored de Bruijn graphs

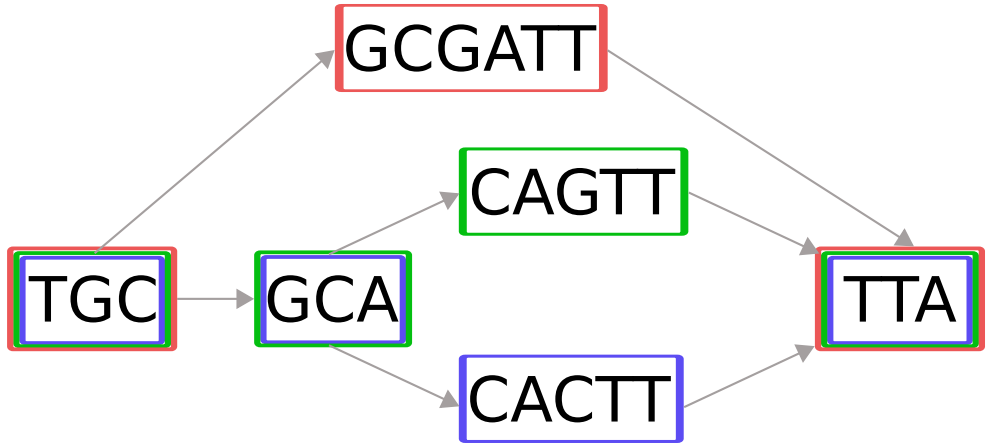
Compacted colored de Bruijn graphs

$k = 3$

S_1 : TGCGATTA

S_2 : TGCAGTTA

S_3 : TGCACTTA



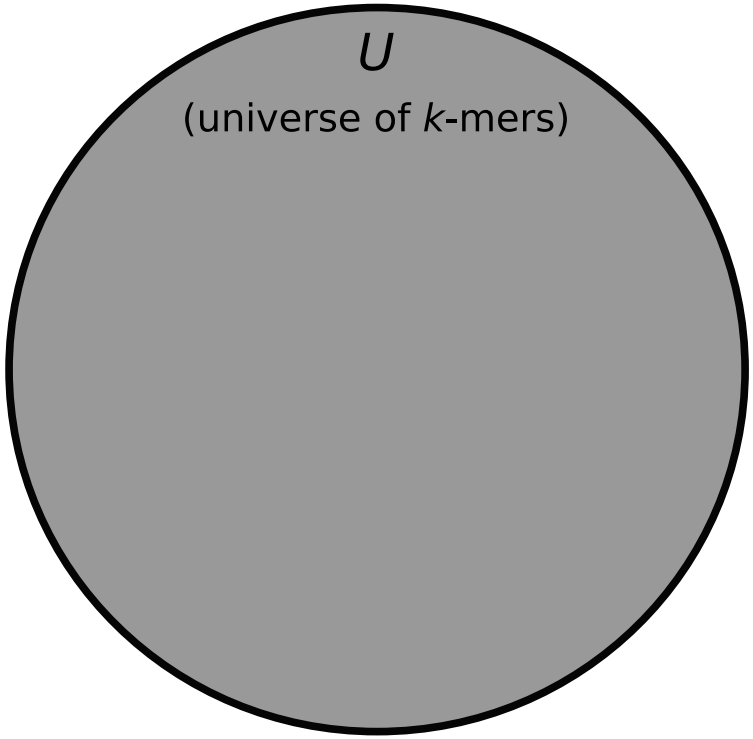
Extension allows:

1. Even more efficient storage
2. Faster sequence querying

Bloom Filter

Bloom Filter

Introduction Bloom filter

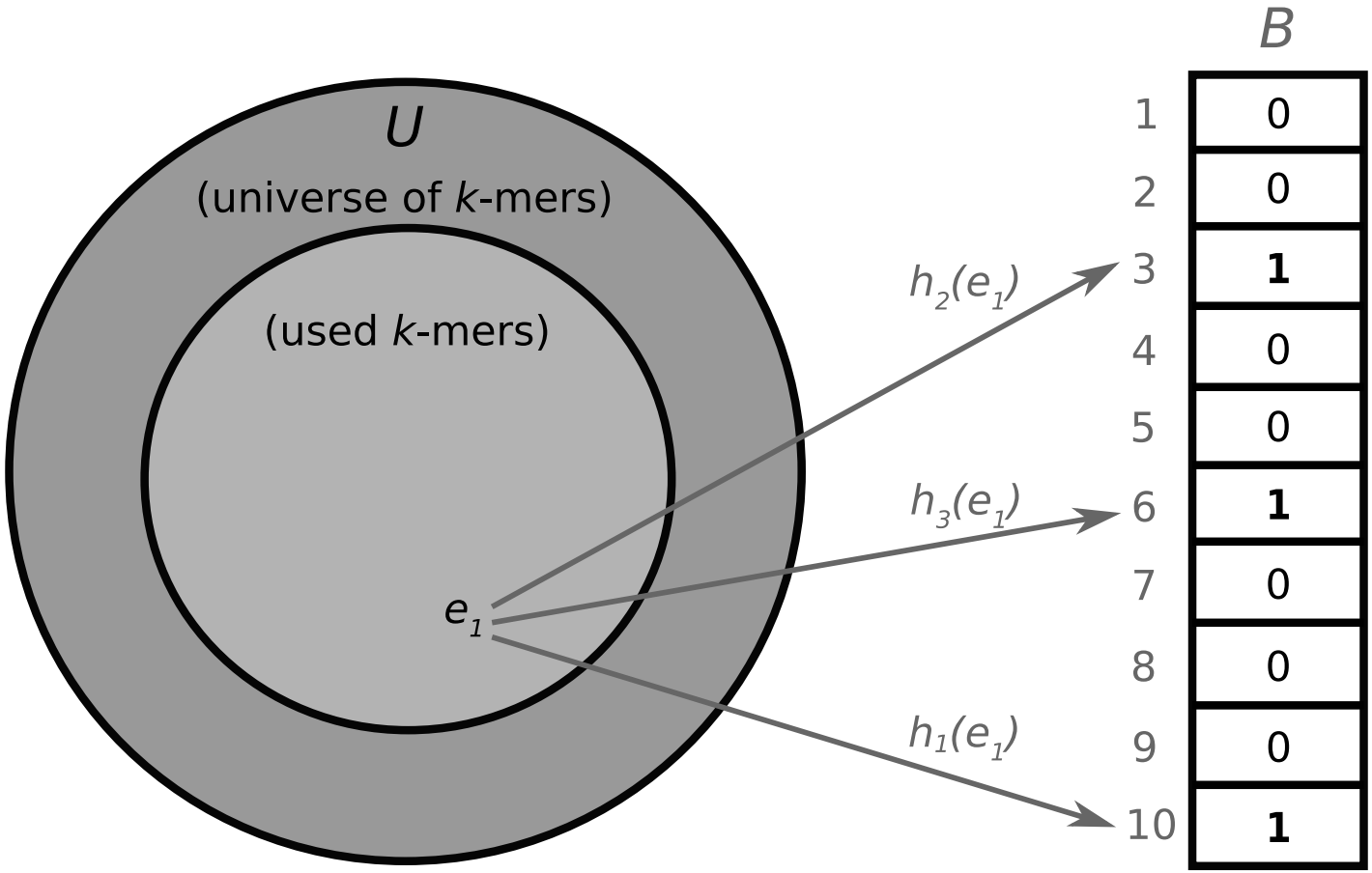


$h_1()$
$h_2()$
$h_3()$

	B
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	0

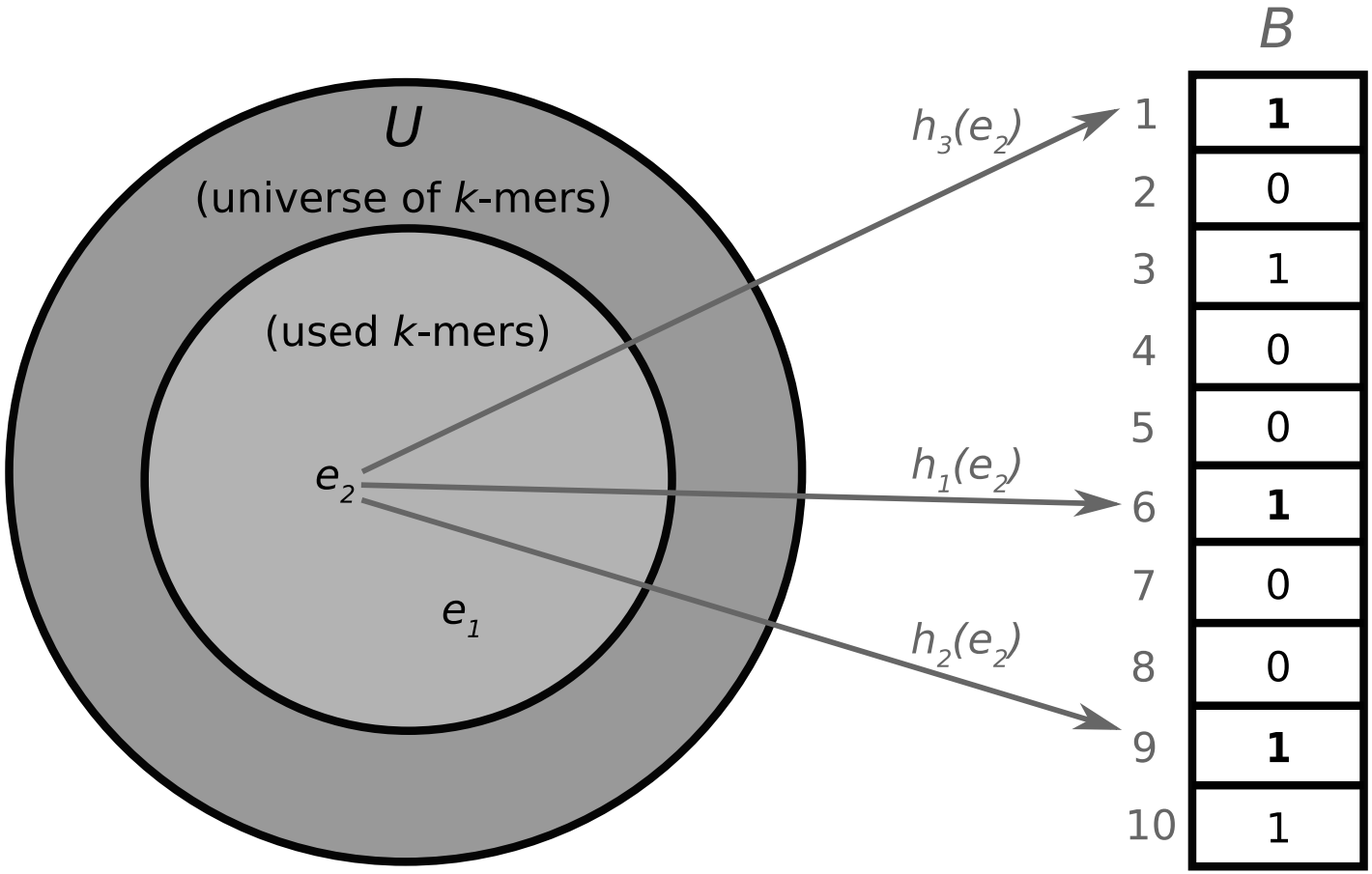
Bloom Filter

Introduction Bloom filter



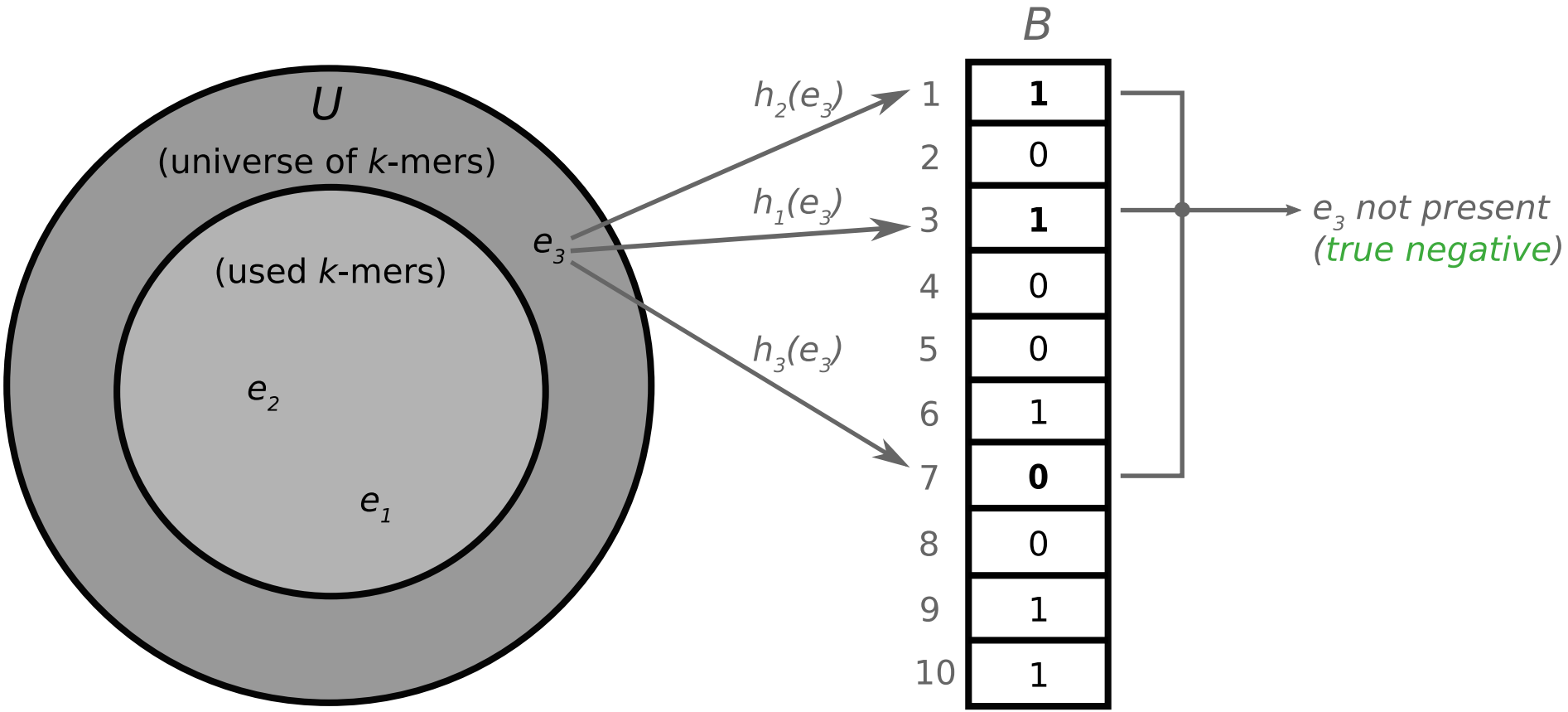
Bloom Filter

Introduction Bloom filter



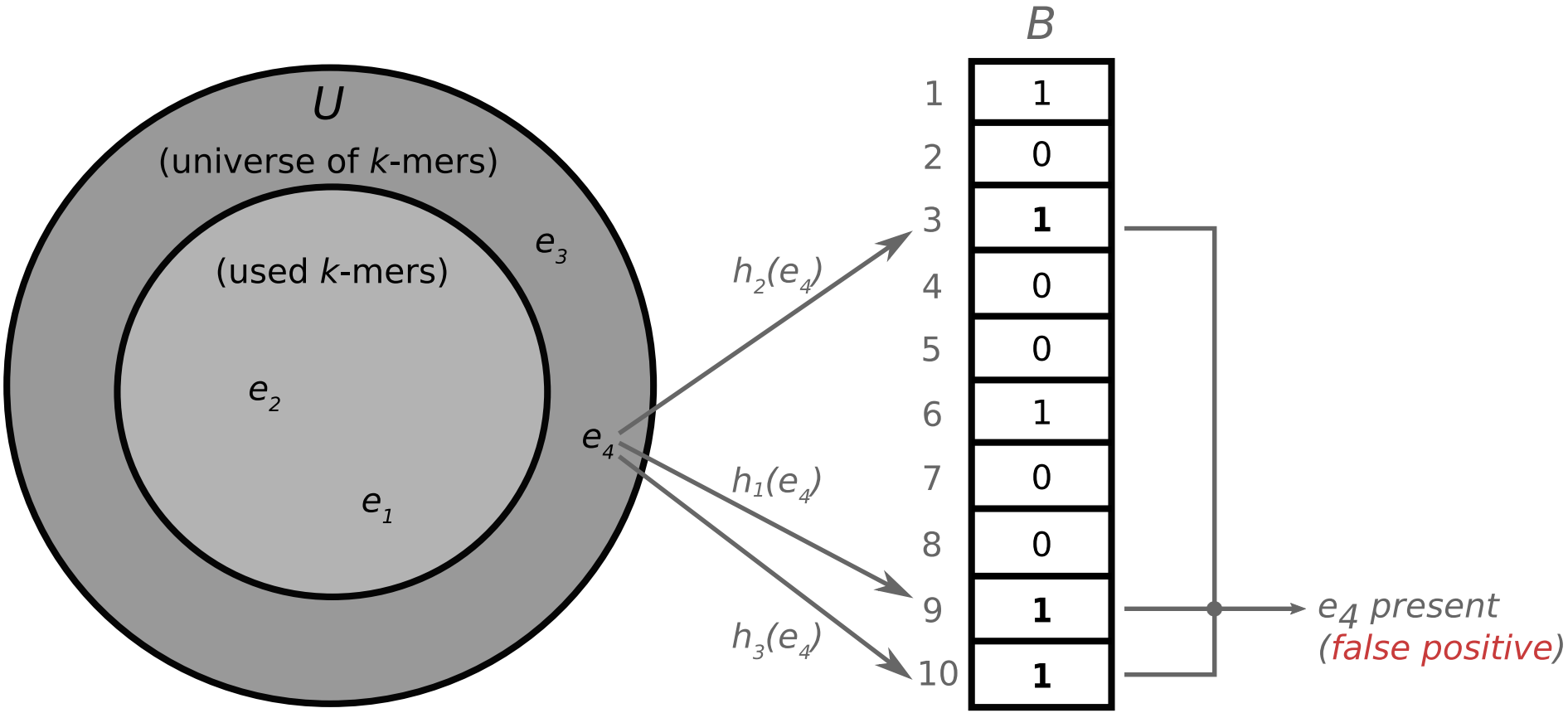
Bloom Filter

Introduction Bloom filter



Bloom Filter

Introduction Bloom filter



Note: Not possible to store color sets

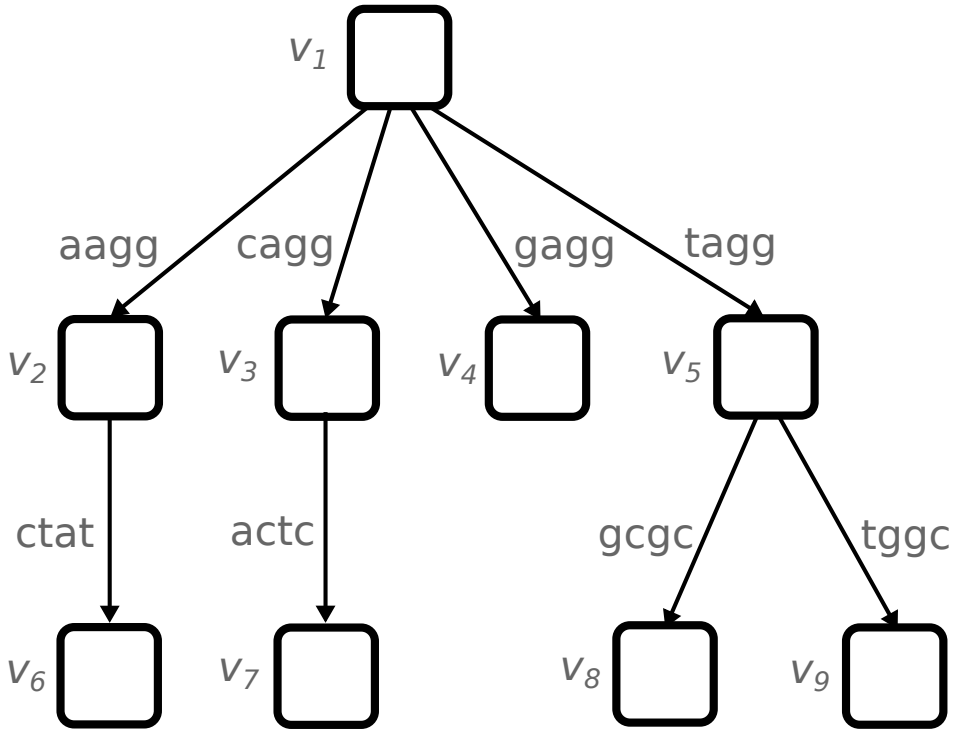
Bloom Filter Trie

Bloom Filter Trie

The Bloom Filter Trie – BFT (Holley *et al.*, 2016)

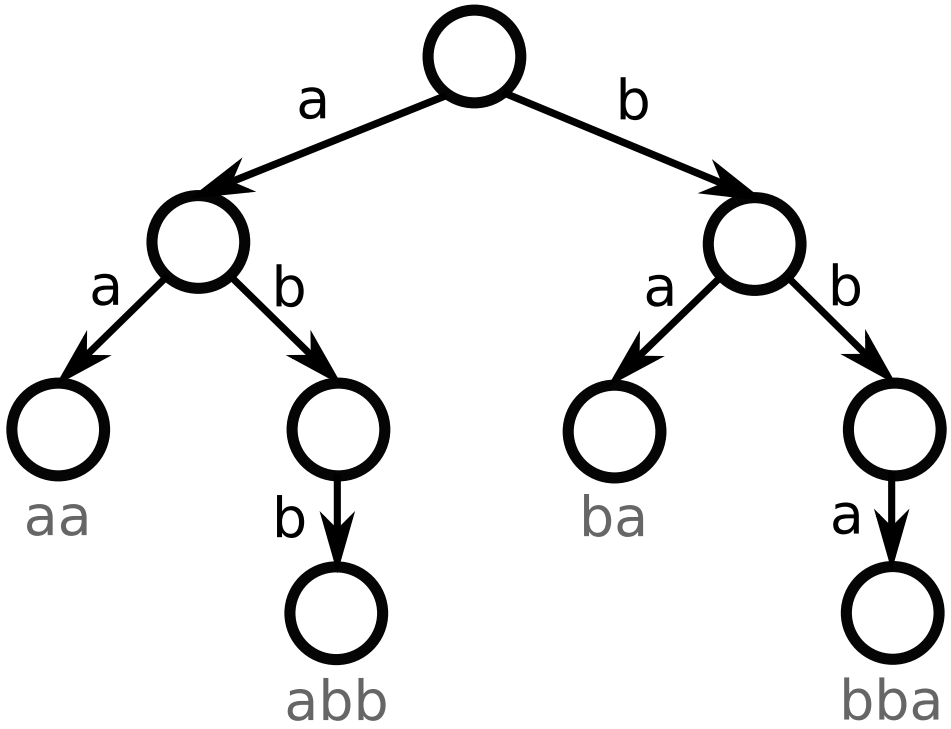
Implementation of a C-DBG

Only k-mers associated with colors are stored in a burst trie



Bloom Filter Trie

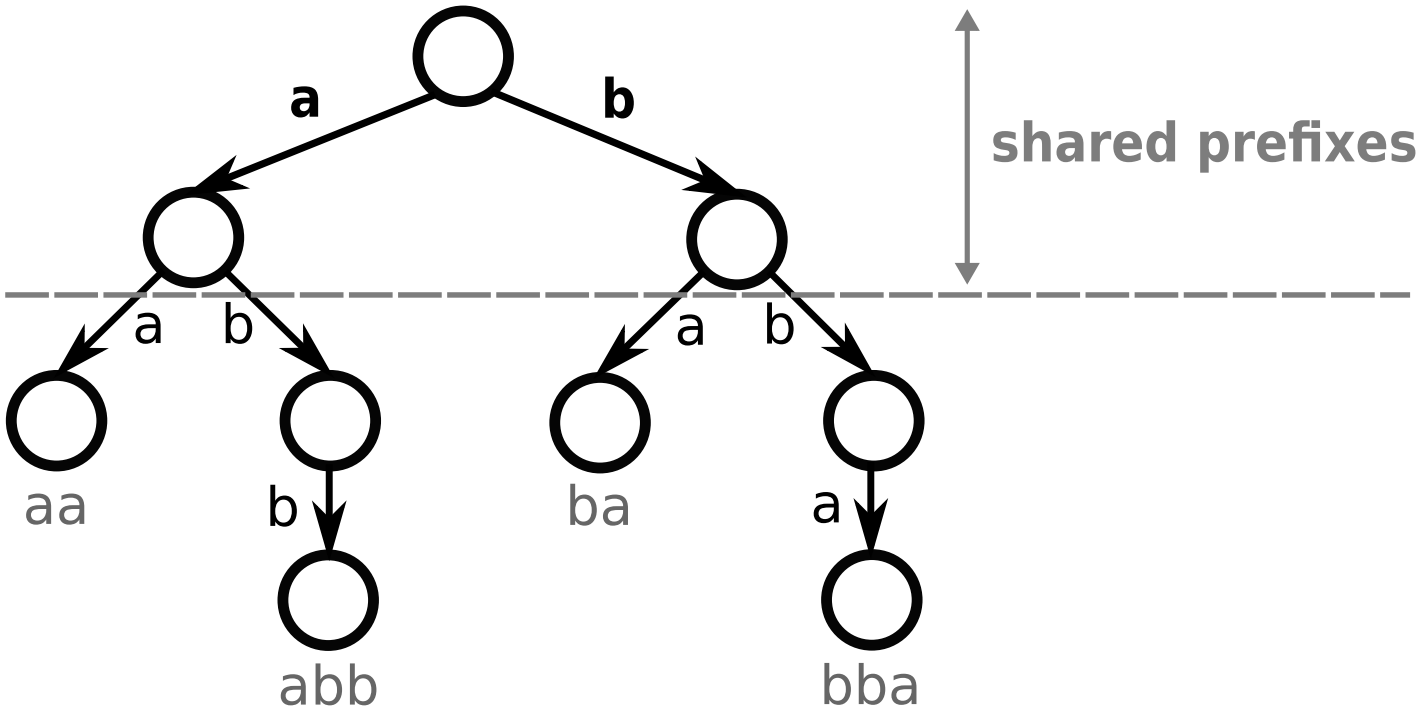
Introduction Trie



Trie of string set $\{aa, abb, ba, bba\}$

Bloom Filter Trie

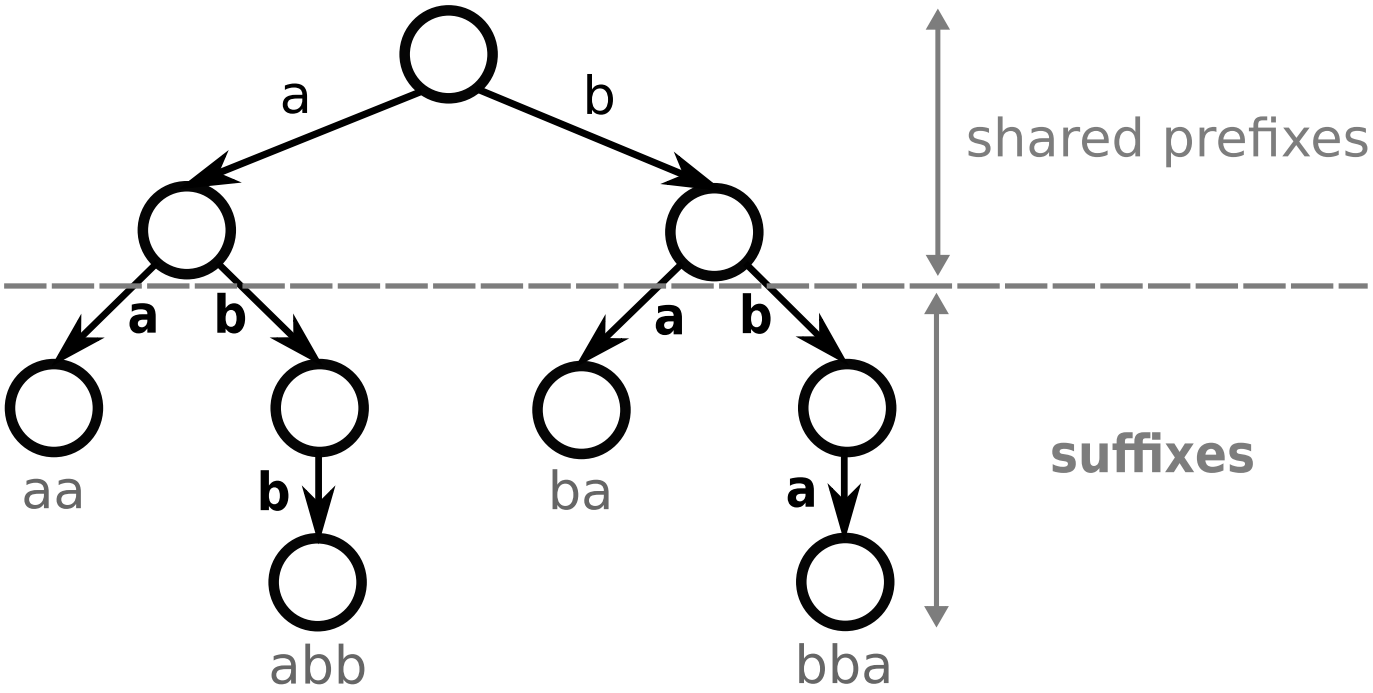
Introduction Trie



Trie of string set $\{\mathbf{a}\text{a}, \mathbf{a}\text{bb}, \mathbf{b}\text{a}, \mathbf{b}\text{ba}\}$

Bloom Filter Trie

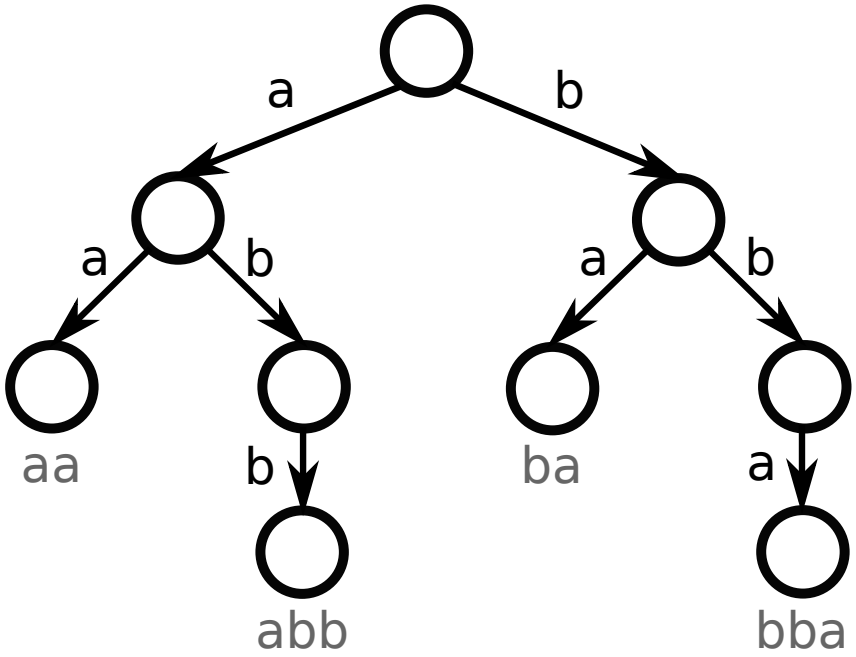
Introduction Trie



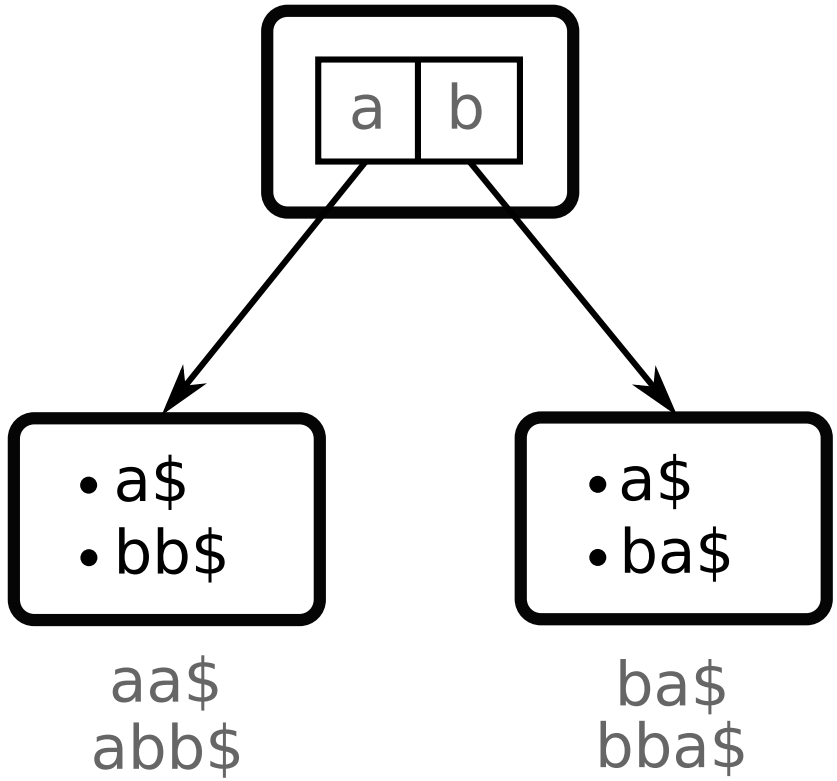
Trie of string set $\{a\mathbf{a}, a\mathbf{bb}, b\mathbf{a}, b\mathbf{ba}\}$

Bloom Filter Trie

Introduction Trie



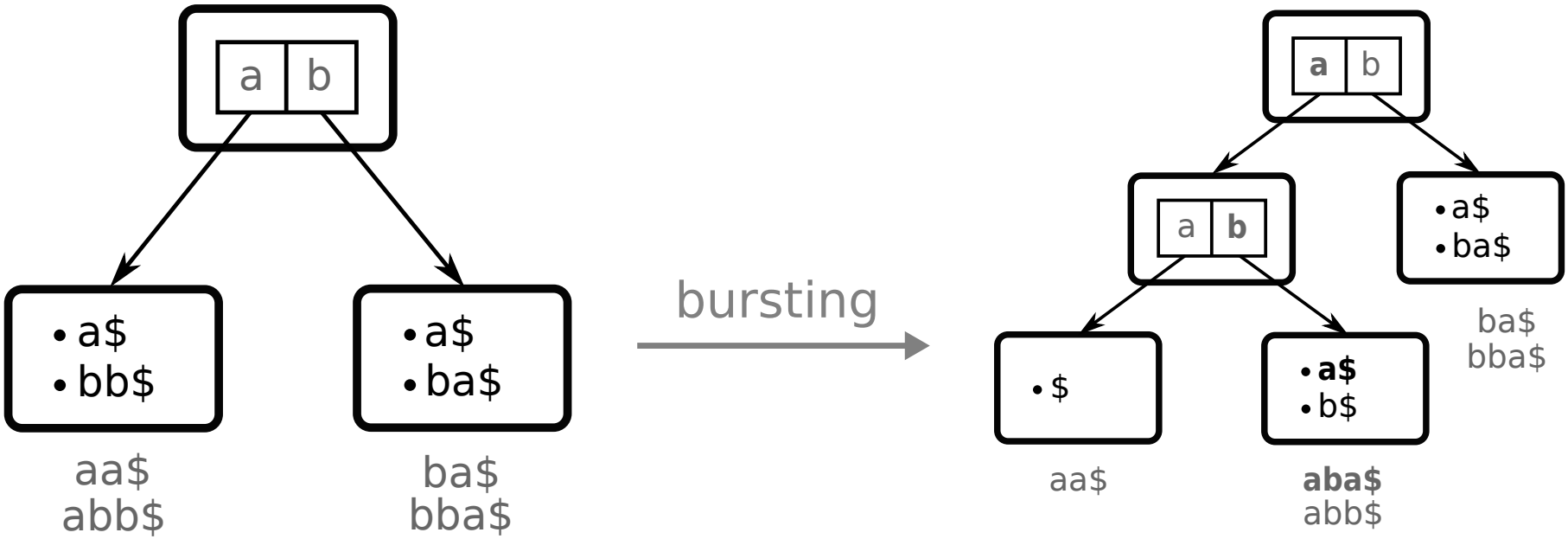
Trie



Burst-trie

Bloom Filter Trie

Introduction Trie

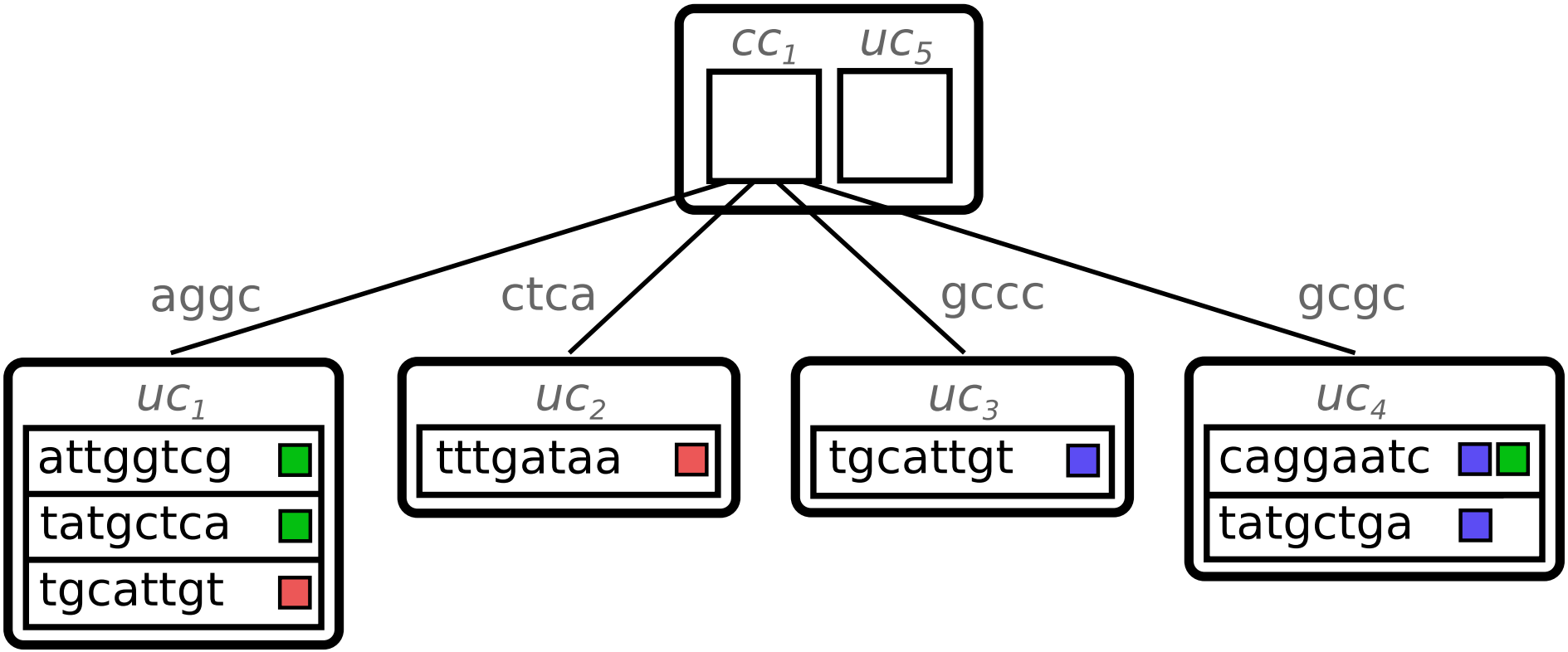


{aa, abb, ba, bba}

{aa, **aba**, abb, ba, bba}

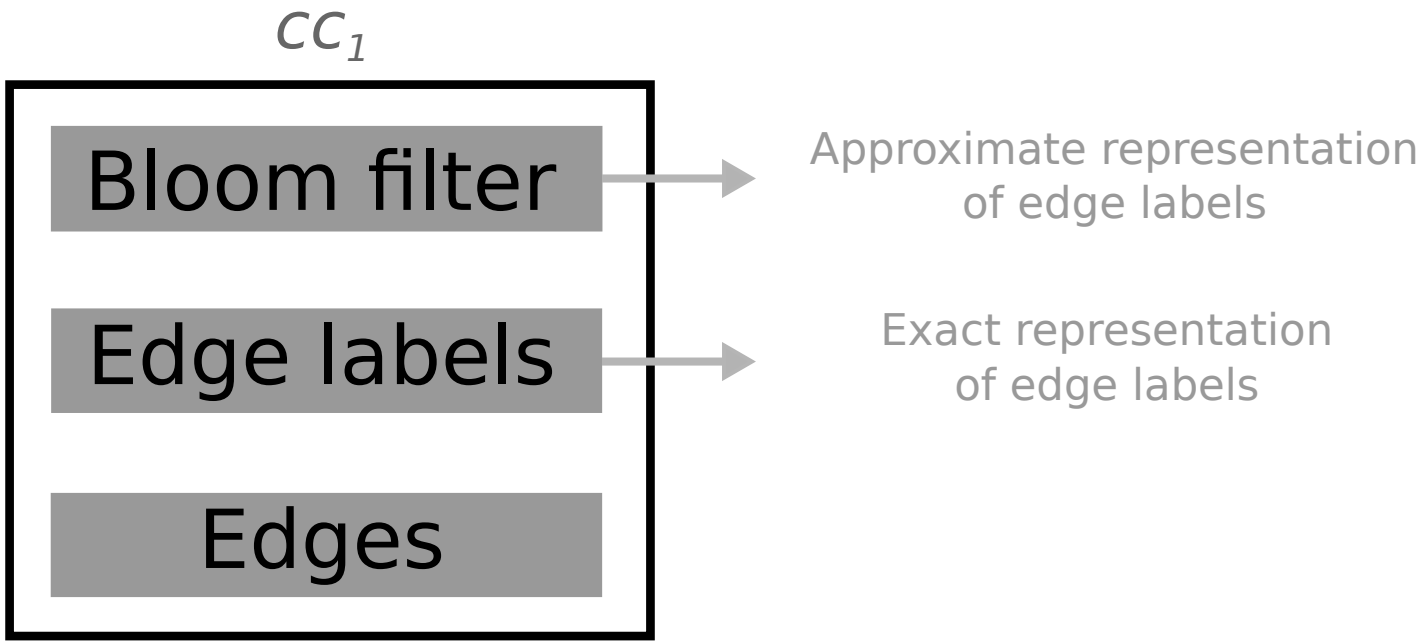
Bloom Filter Trie

Vertices of the Bloom Filter Trie



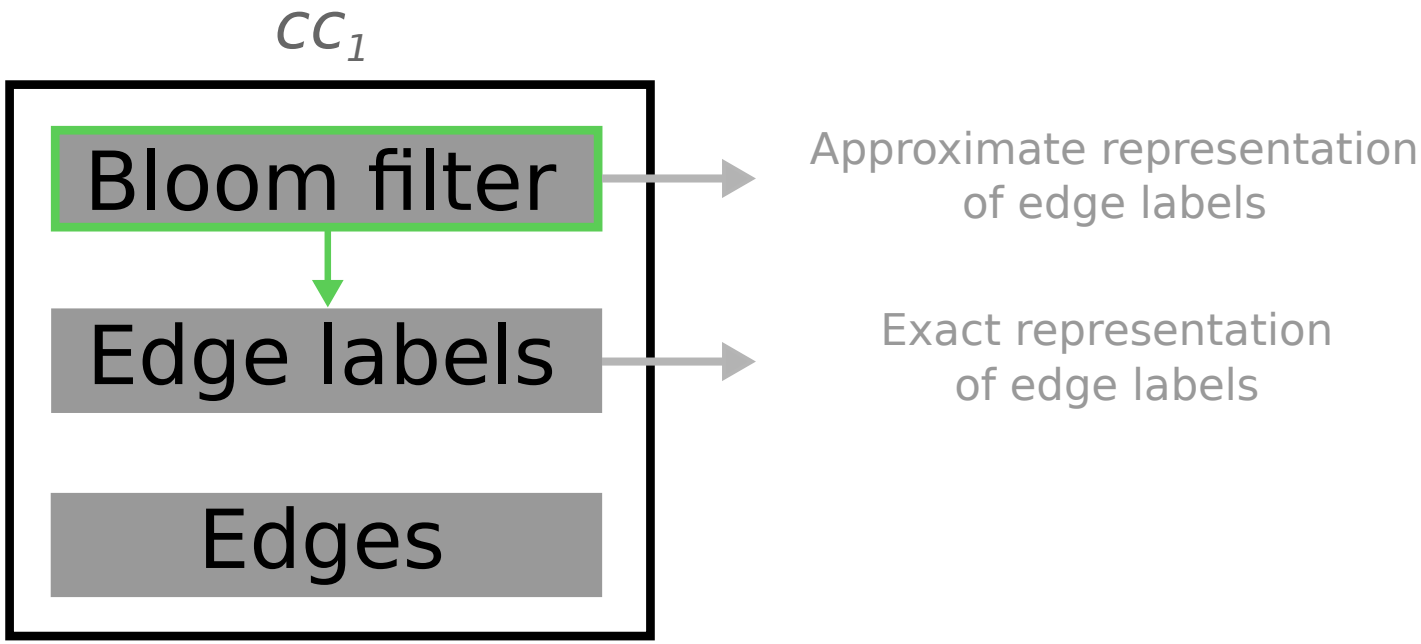
Bloom Filter Trie

Vertices of the Bloom Filter Trie



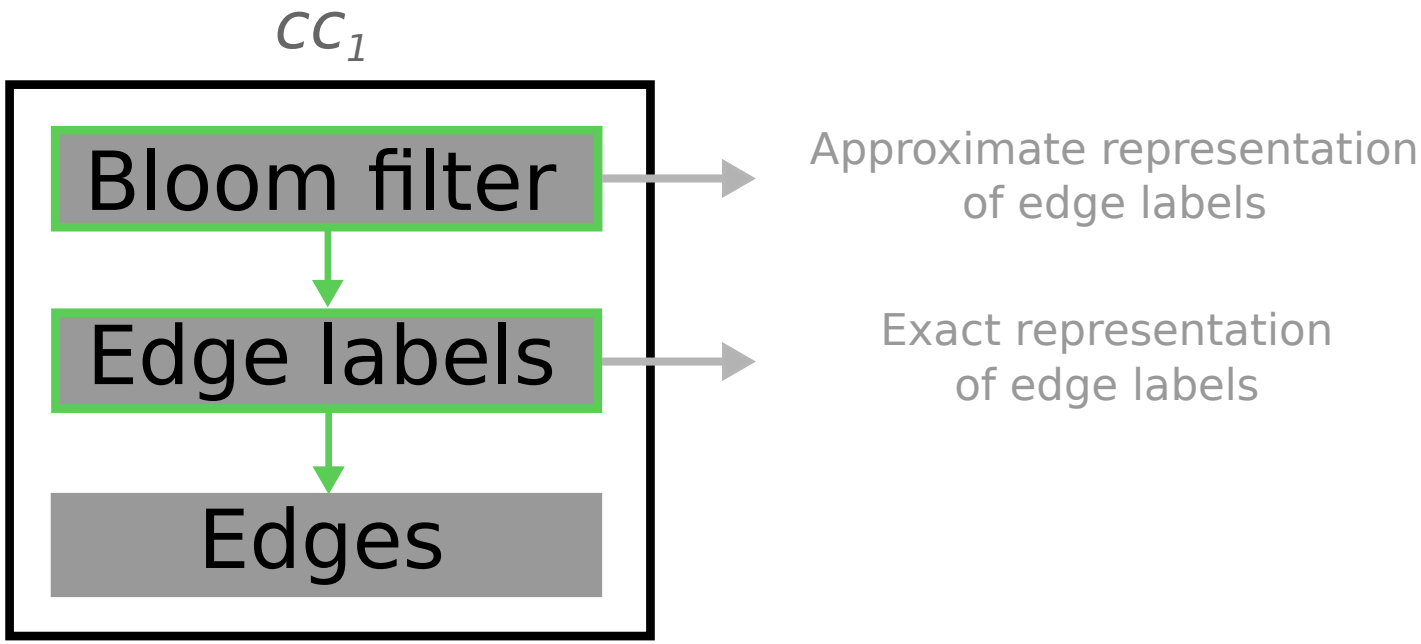
Bloom Filter Trie

Vertices of the Bloom Filter Trie



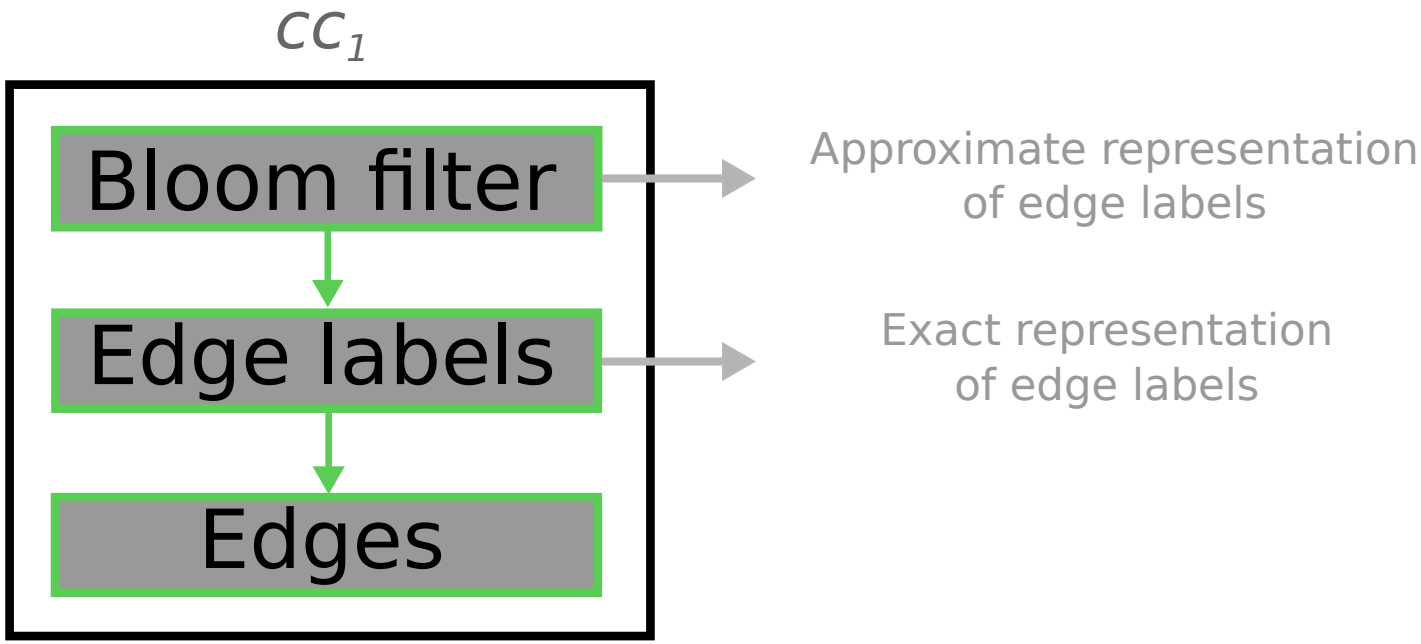
Bloom Filter Trie

Vertices of the Bloom Filter Trie



Bloom Filter Trie

Vertices of the Bloom Filter Trie



Bifrost

Bifrost

Bifrost (`https://github.com/pmelsted/bfgraph`)

Implementation of a compacted C-DBG

Currently under development

Unitigs can be accessed via minimizers they contain

Bifrost

Bifrost (`https://github.com/pmelsted/bfgraph`)

Implementation of a compacted C-DBG

Currently under development

Unitigs can be accessed via minimizers they contain

Minimizer: lexicographically smallest ℓ-mer of a sequence and its reverse complement

$s = \text{TGACTTCCATGT}$
$\ell = 4$

ℓ-mer		rev. comp.	
TGAC	TCCA	GTCA	TGGA
GACT	CCAT	AGTC	ATGG
ACTT	CATG	AAGT	CATG
CTTC	ATGT	GAAG	ACAT
TTCC		GGAA	

Bifrost

Bifrost (`https://github.com/pmelsted/bfgraph`)

Implementation of a compacted C-DBG

Currently under development

Unitigs can be accessed via minimizers they contain

Minimizer: lexicographically smallest ℓ-mer of a sequence and its reverse complement

$s = \text{TG}\mathbf{ACTT}\text{CCATGT}$
$\ell = 4$

ℓ-mer		rev. comp.	
TGAC	TCCA	GTCA	TGGA
GACT	CCAT	AGTC	ATGG
ACTT	CATG	**AAGT**	CATG
CTTC	ATGT	GAAG	ACAT
TTCC		GGAA	

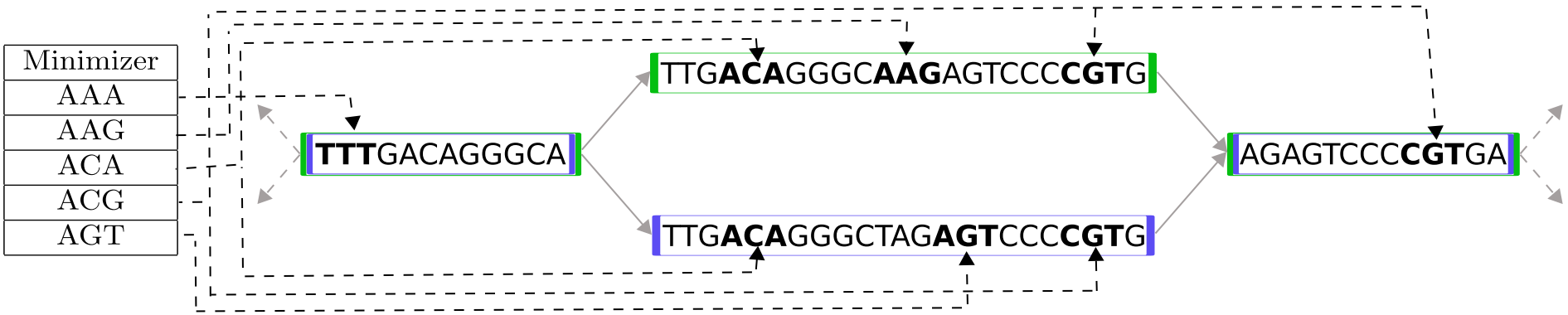
Bifrost

Bifrost (`https://github.com/pmelsted/bfgraph`)

Implementation of a compacted C-DBG

Currently under development

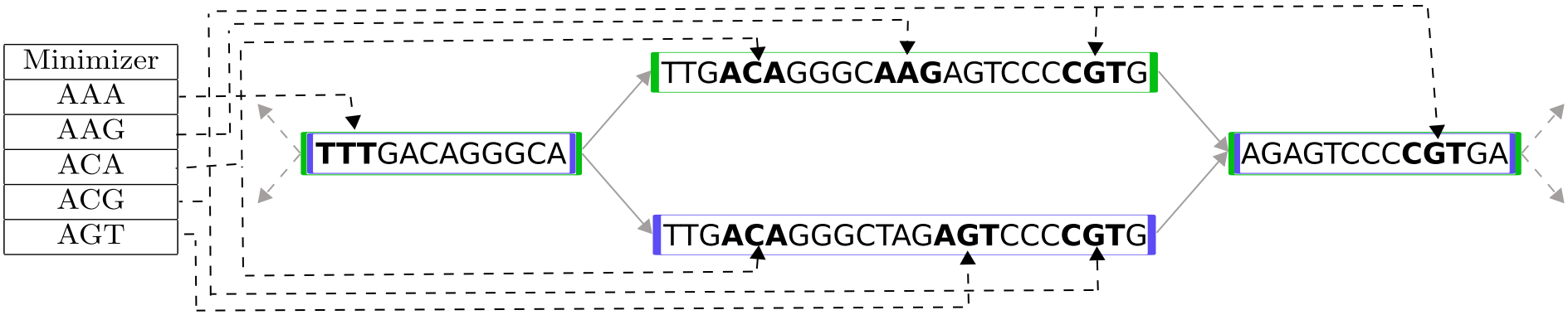
Unitigs can be accessed via minimizers they contain



Bifrost

Why does it make sense to use minimizers?

- Many consecutive k-mers have the same minizer
- Minimizer index much smaller than k-mer index



Querying a sequence DB - BLAST

Querying a sequence DB - BLAST

BLAST – Basic Local Alignment Search Tool

Developed by Stephen Altschul *et al.* in 1990 (second version in 1997)
Is used to query databases of DNA and protein sequences

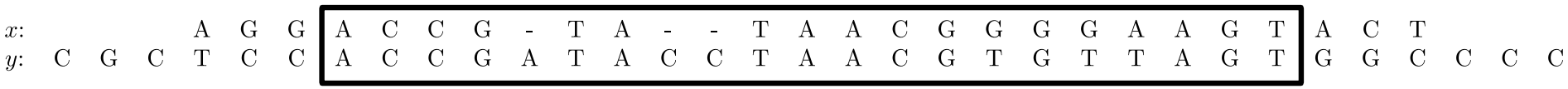
Different flavors are available:

- DNA→DNA (blastn)
- protein→protein (blastp)
- translated DNA→protein (blastx)
- protein→translated DNA (tblastn)
- ...

Querying a sequence DB - BLAST

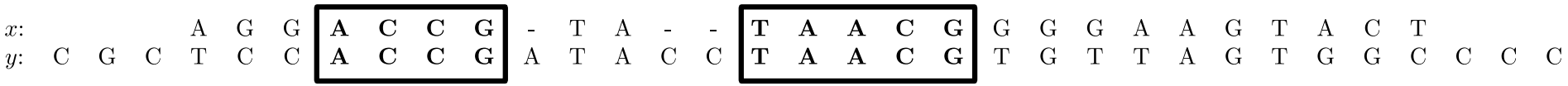
Problem statement:

Given a query sequence $x \in \Sigma^*$ and a sequence database Y, find highest scoring local alignments between x and $y \in Y$ above a certain significance score and return them along with score.



Querying a sequence DB - BLAST

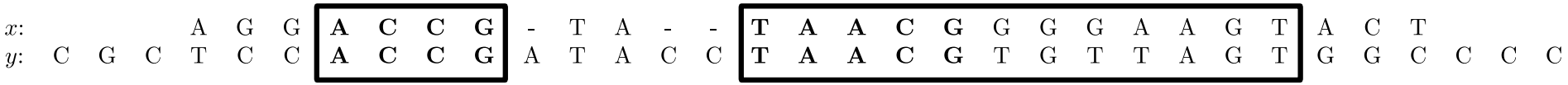
Algorithm procedure (blastn with default parameters):
1. Finding BLAST hits:
Find exact matches of minimal length w between x and all $y \in Y$.



$w = 4$

Querying a sequence DB - BLAST

2. Extension:
Extend hits to left and right using the *X-drop algorithm*.



Querying a sequence DB - BLAST

X-drop algorithm (extension to the right):
INPUT: query x, database sequence y, hit $h = (i, j)$, score s, drop-off parameter X
OUTPUT: extended hit h, increased score s

1: $s' \leftarrow 0$
2: $j' \leftarrow j + 1$
3: **while** end of x and y with reached **do**
4: **if** $s' += Score($next character in x and $y) > 0$ **then**
5: $s \leftarrow s'$
6: $j \leftarrow j'$
7: $s' \leftarrow 0$
8: **else if** $s' < X$ **then return** s and h
9: **end if**
10: Continue with next positions in x and y
11: $j' \leftarrow j' + 1$
12: **end while**

Querying a sequence DB - BLAST

X-drop algorithm (extension to the right):

INPUT: query x, database sequence y, hit $h = (i, j)$, score s, drop-off parameter X

OUTPUT: extended hit h, increased score s

1: $s' \leftarrow 0$
2: $j' \leftarrow j + 1$
3: **while** end of x and y not reached **do**
4: **if** $s' += Score(\text{next character in } x \text{ and } y) > 0$ **then**
5: $s \leftarrow s'$
6: $j \leftarrow j'$
7: $s' \leftarrow 0$
8: **else if** $s' < X$ **then return** s and h
9: **end if**
10: Continue with next positions in x and y
11: $j' \leftarrow j' + 1$
12: **end while**

Querying a sequence DB - BLAST

X-drop algorithm (extension to the right):
INPUT: query x, database sequence y, hit $h = (i, j)$, score s, drop-off parameter X
OUTPUT: extended hit h, increased score s

1: $s' \leftarrow 0$
2: $j' \leftarrow j + 1$
3: **while** end of x and y not reached **do**
4: **if** $s' += Score(\text{next character in } x \text{ and } y) > 0$ **then**
5: $s \leftarrow s'$
6: $j \leftarrow j'$
7: $s' \leftarrow 0$
8: **else if** $s' < X$ **then return** s and h
9: **end if**
10: Continue with next positions in x and y
11: $j' \leftarrow j' + 1$
12: **end while**

Querying a sequence DB - BLAST

X-drop algorithm (extension to the right):
INPUT: query x, database sequence y, hit $h = (i, j)$, score s, drop-off parameter X
OUTPUT: extended hit h, increased score s

1: $s' \leftarrow 0$
2: $j' \leftarrow j + 1$
3: **while** end of x and y not reached **do**
4: **if** $s' += Score($next character in x and $y) > 0$ **then**
5: $s \leftarrow s'$
6: $j \leftarrow j'$
7: $s' \leftarrow 0$
8: **else if** $s' < X$ **then return** s and h
9: **end if**
10: Continue with next positions in x and y
11: $j' \leftarrow j' + 1$
12: **end while**

Querying a sequence DB - BLAST

X-drop algorithm (extension to the right):
INPUT: query x, database sequence y, hit $h = (i, j)$, score s, drop-off parameter X
OUTPUT: extended hit h, increased score s

1: $s' \leftarrow 0$
2: $j' \leftarrow j + 1$
3: **while** end of x and y not reached **do**
4: **if** $s' += Score(\text{next character in } x \text{ and } y) > 0$ **then**
5: $s \leftarrow s'$
6: $j \leftarrow j'$
7: $s' \leftarrow 0$
8: **else if** $s' < X$ **then return** s and h
9: **end if**
10: Continue with next positions in x and y
11: $j' \leftarrow j' + 1$
12: **end while**

Querying a sequence DB - BLAST

X-drop algorithm (extension to the right):
INPUT: query x, database sequence y, hit $h = (i, j)$, score s, drop-off parameter X
OUTPUT: extended hit h, increased score s

1: $s' \leftarrow 0$
2: $j' \leftarrow j + 1$
3: **while** end of x and y not reached **do**
4: **if** $s' += Score(\text{next character in } x \text{ and } y) > 0$ **then**
5: $s \leftarrow s'$
6: $j \leftarrow j'$
7: $s' \leftarrow 0$
8: **else if** $s' < X$ **then return** s and h
9: **end if**
10: Continue with next positions in x and y
11: $j' \leftarrow j' + 1$
12: **end while**

Querying a sequence DB - BLAST

X-drop algorithm (extension to the right):
INPUT: query x, database sequence y, hit $h = (i, j)$, score s, drop-off parameter X
OUTPUT: extended hit h, increased score s

1: $s' \leftarrow 0$
2: $j' \leftarrow j + 1$
3: **while** end of x and y not reached **do**
4: **if** $s' += Score(\text{next character in } x \text{ and } y) > 0$ **then**
5: $s \leftarrow s'$
6: $j \leftarrow j'$
7: $s' \leftarrow 0$
8: **else if** $s' < X$ **then return** s and h
9: **end if**
10: Continue with next positions in x and y
11: $j' \leftarrow j' + 1$
12: **end while**

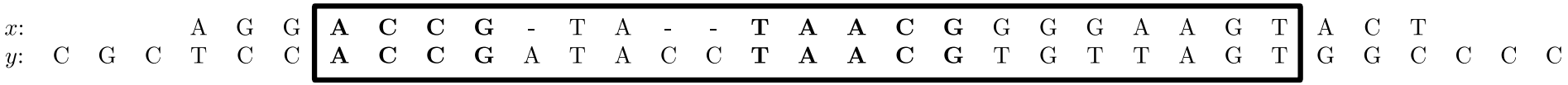
Querying a sequence DB - BLAST

X-drop algorithm (extension to the right):
INPUT: query x, database sequence y, hit $h = (i, j)$, score s, drop-off parameter X
OUTPUT: extended hit h, increased score s

1: $s' \leftarrow 0$
2: $j' \leftarrow j + 1$
3: **while** end of x and y not reached **do**
4: **if** $s' += Score(\text{next character in } x \text{ and } y) > 0$ **then**
5: $s \leftarrow s'$
6: $j \leftarrow j'$
7: $s' \leftarrow 0$
8: **else if** $s' < X$ **then return** s and h
9: **end if**
10: Continue with next positions in x and y
11: $j' \leftarrow j' + 1$
12: **end while**

Querying a sequence DB - BLAST

3. Combination of results:
Closely located, extended hits within the same sequence are combined.



Querying a colored de Bruijn graph - PLAST

Querying a colored de Bruijn graph - PLAST

Efficient methods to query a C-DBG are yet unknown

Idea: Develop a method that queries a C-DBG in a BLAST-like manner to find local alignments between a query sequence and the sequences stored in the C-DBG.

$\rightarrow$ Pan-genome Local Alignment Search Tool – PLAST

Querying a colored de Bruijn graph - PLAST

Problem statement: Given a set of sequences Y represented as a C-DBG G and a query sequence x, find all $s \in S$ and their local alignments with x that are scored highest and above a certain threshold value.

Procedure of PLAST:

1. Seed detection
2. Seed extension
3. Combination of results

Querying a colored de Bruijn graph - PLAST

1. Seed detection:

x: AGG**ACCG**TA**TAACG**GGGAAGTACT

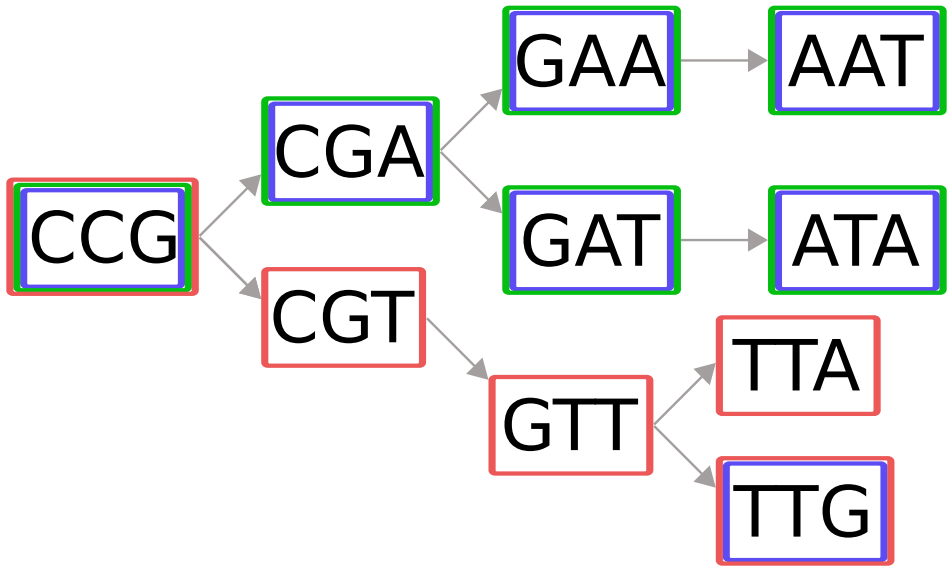
exact match length $w = 4$

Problems:

- Not efficiently possible with the BFT if $k > w$ (expected case)
- Minimizer size can be chosen arbitrarily with Bifrost, but is fixed after C-DBG creation and affects the computation time

Querying a colored de Bruijn graph - PLAST

2. Seed extension:

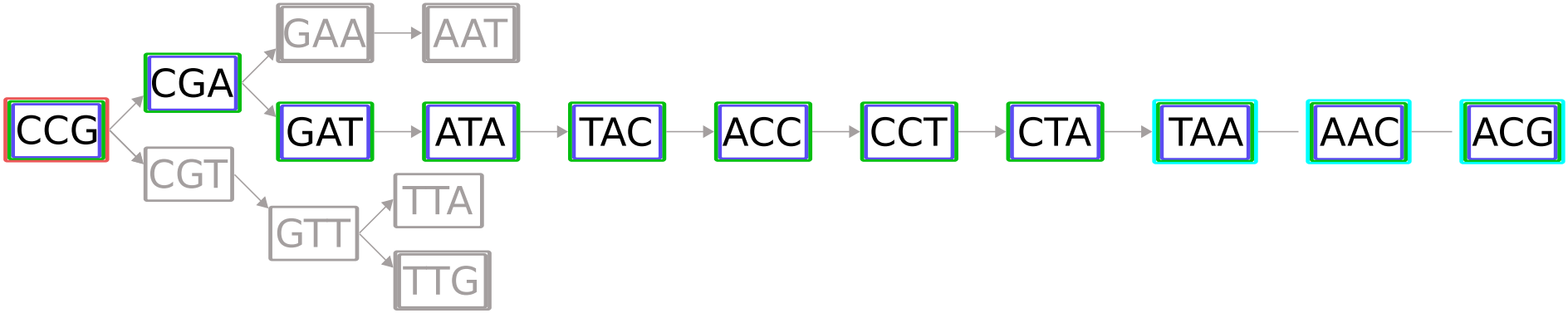


x: AGG**ACCG**TA**TAACG**GGGAAGTACT

Challenge: Smart graph traversal is crucial (especially if using the BFT)

Querying a colored de Bruijn graph - PLAST

3. Combination of results:



Same as extension step, but with gaps

Conclusion

- Current improvements in sequencing technologies necessitate the development of memory efficient ways to store sequencing data
- C-DBGs seem to be a suitable data structure to efficiently store highly similar sequences
- Fast and sensitive methods to query C-DBGs are required to allow their usage
- The development of PLAST is challenging but looks promising

End

Thank you for your attention!