findere:
fast and precise approximate membership query

Lucas Robidou, Pierre Peterlongo

Monday 4th October, 2021

Inria Rennes

Introduction

Introduction

findere

Summary

Introduction - Bloom filters

A Bloom filter is a data structure used to test wether an element X is in a set S.

- If $X \in S$, then the filter answers 'True'
- If $X \notin S$, the filter might still answers 'True' with a probability ϵ (collisions)

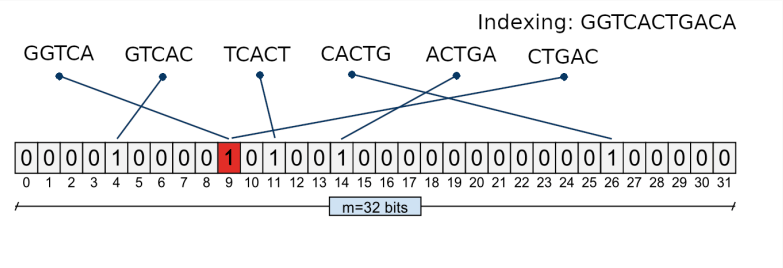
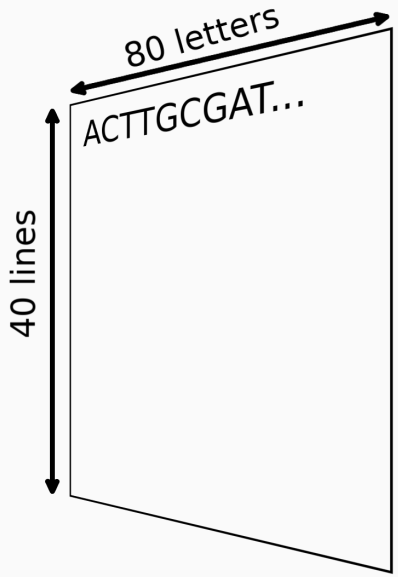


Figure 1: A Bloom Filter

Some context

Main goal:

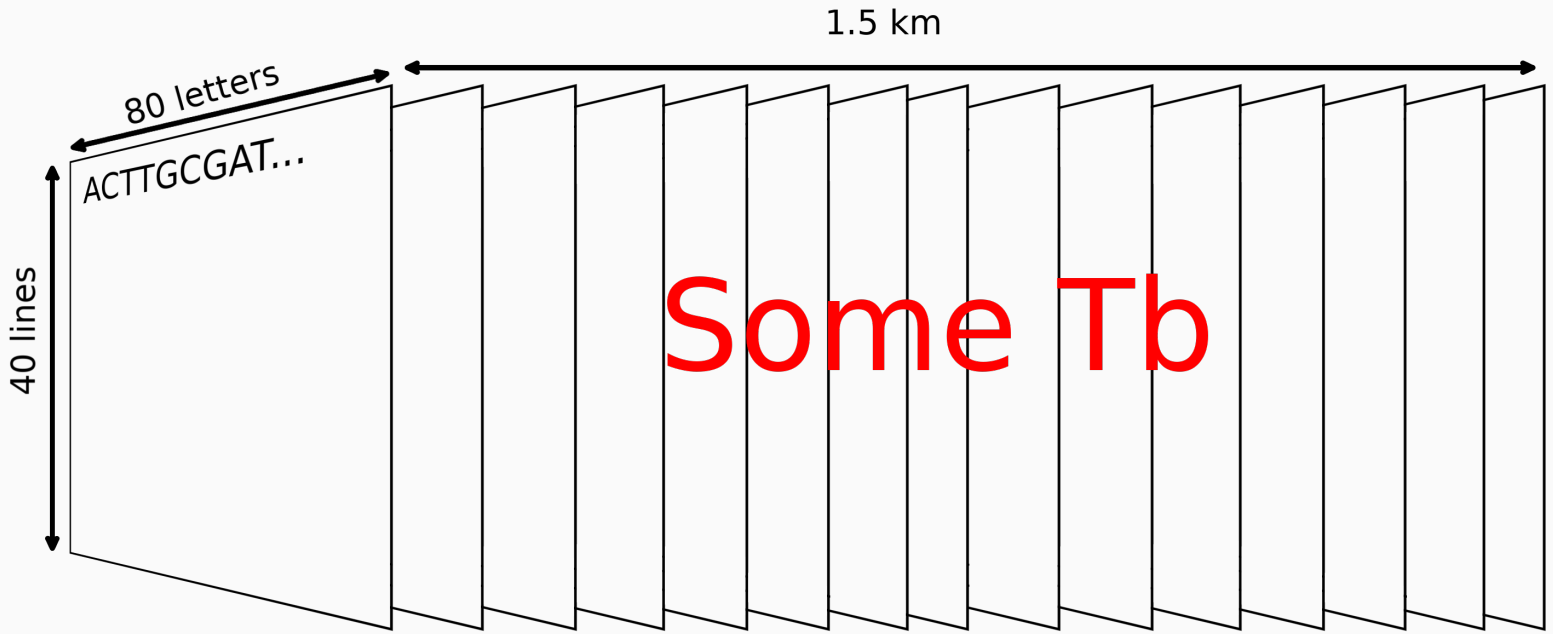
- to read and index (large) genomic datasets
- to query those indexed datasets



Some context

Main goal:

- to read and index (large) genomic datasets
- to query those indexed datasets



Some context

Challenges:

- indexation time
- abundance storage
- **index size**
- **query time**
- **false positive rate**

Some context

How do I index?

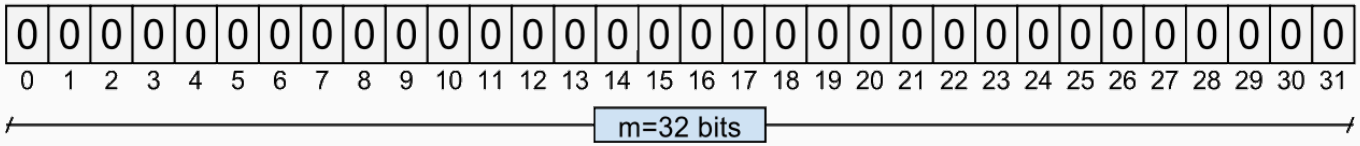
- extract every subsequence of size K (K-mers), index them
- query every K-mer from your queried sequence
- compute similarity using the proportion of shared K-mers

Classic indexation example

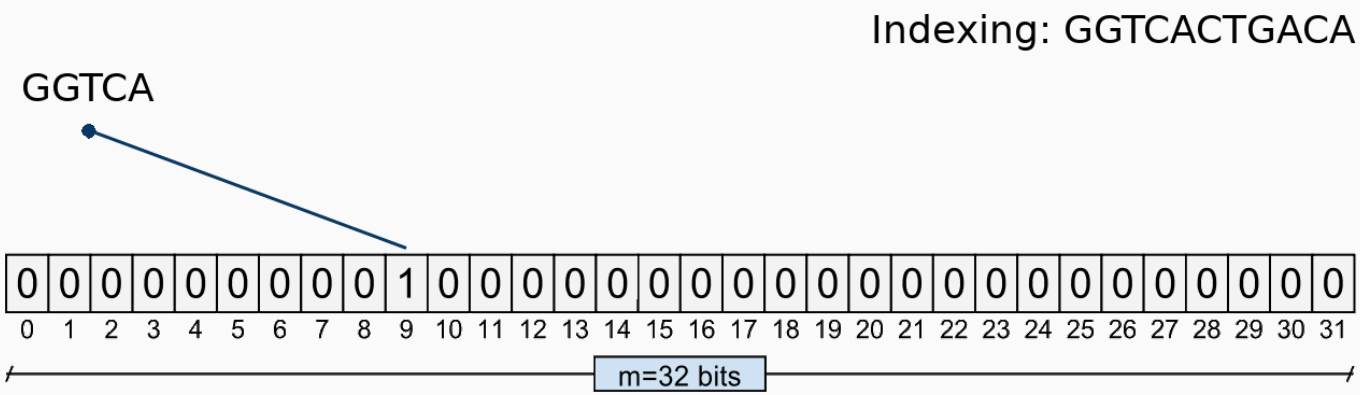
Indexing: GGTCACTGACA

Classic indexation example

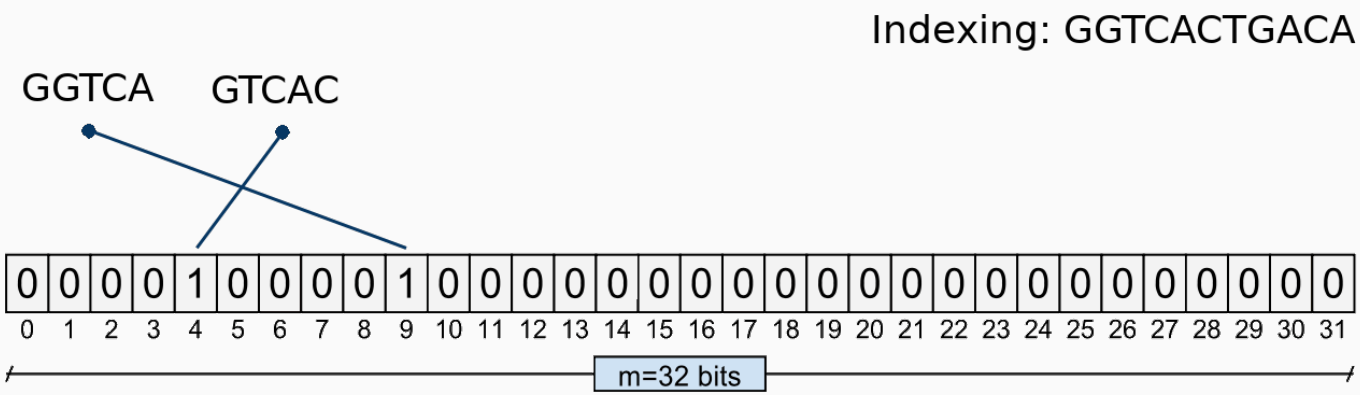
Indexing: GGTCACTGACA



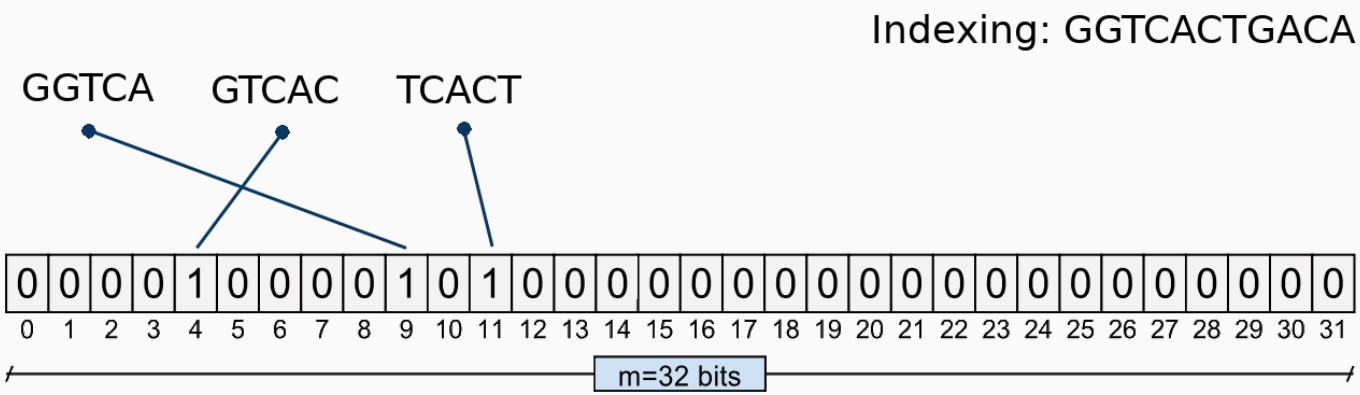
Classic indexation example



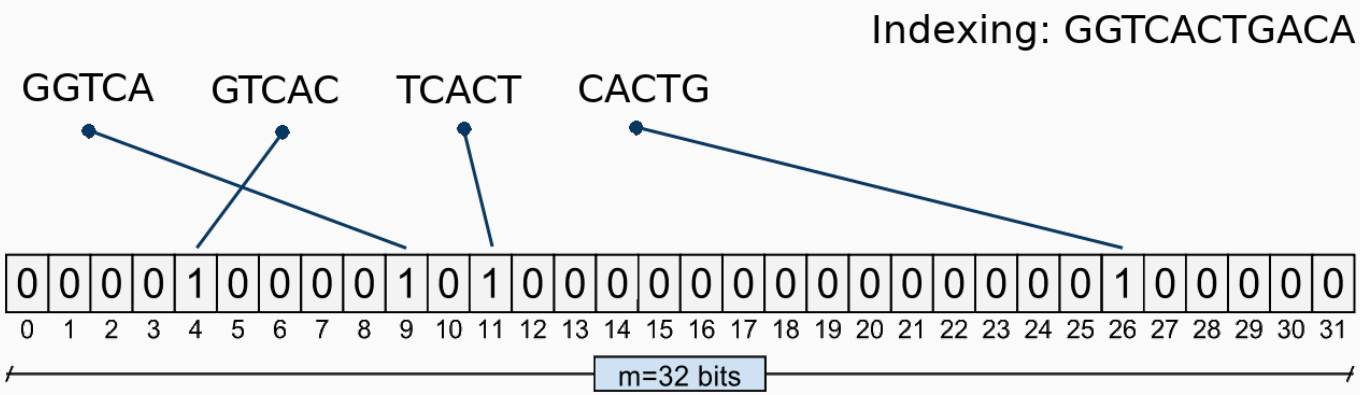
Classic indexation example



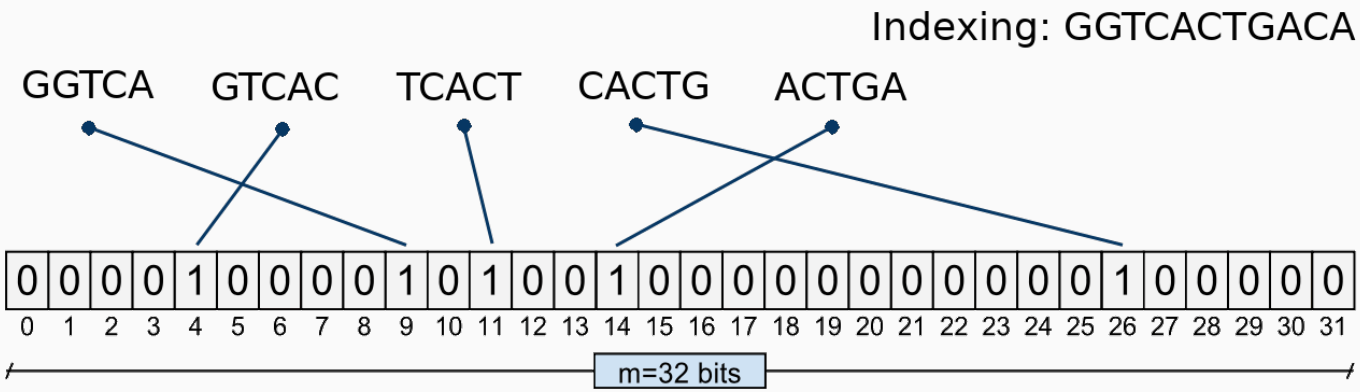
Classic indexation example



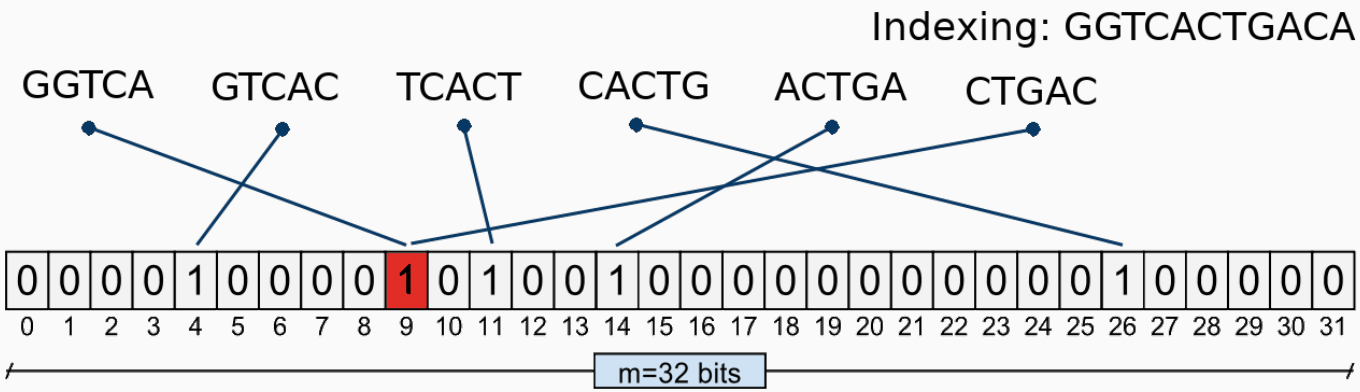
Classic indexation example



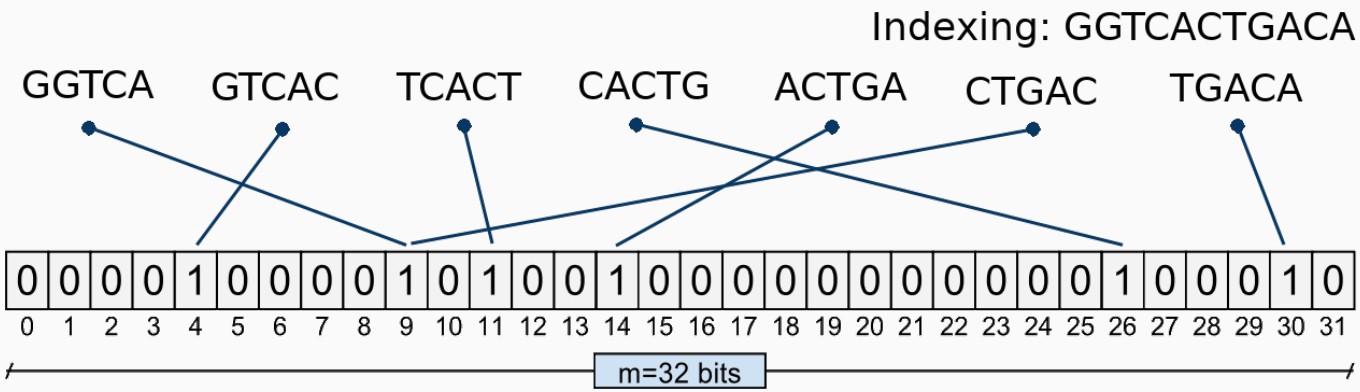
Classic indexation example



Classic indexation example



Classic indexation example



Summary

Main idea of findere

Let's consider the 7-mer 'biology'. Its 5-mers are:

- 'biolo'
- 'iolog'
- 'ology'

One of them not found $\implies$ 'biology' not found

New method: findere

Rather than indexing K-mers, **let's index k-mers**, $k < K$.
Let's introduce $z = K - k$, so that a K-mer is made of $z + 1$ smaller k-mers.

A K-mer is said 'found' iif the $z + 1$ k-mers composing it are found in the filter.

New method: findere

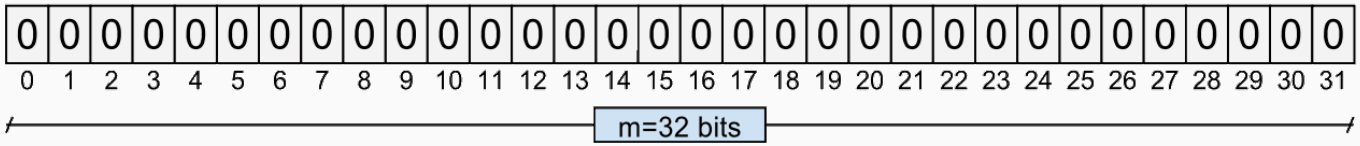
Let us change the indexation part a bit: we now need to index k-mers.

Findere indexation example

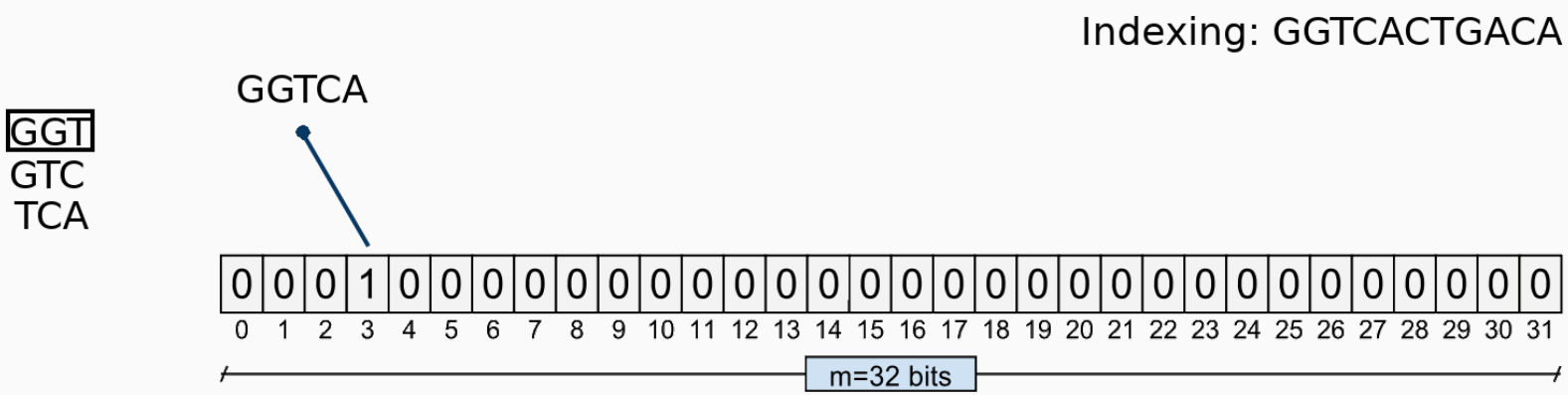
Indexing: GGTCACTGACA

Findere indexation example

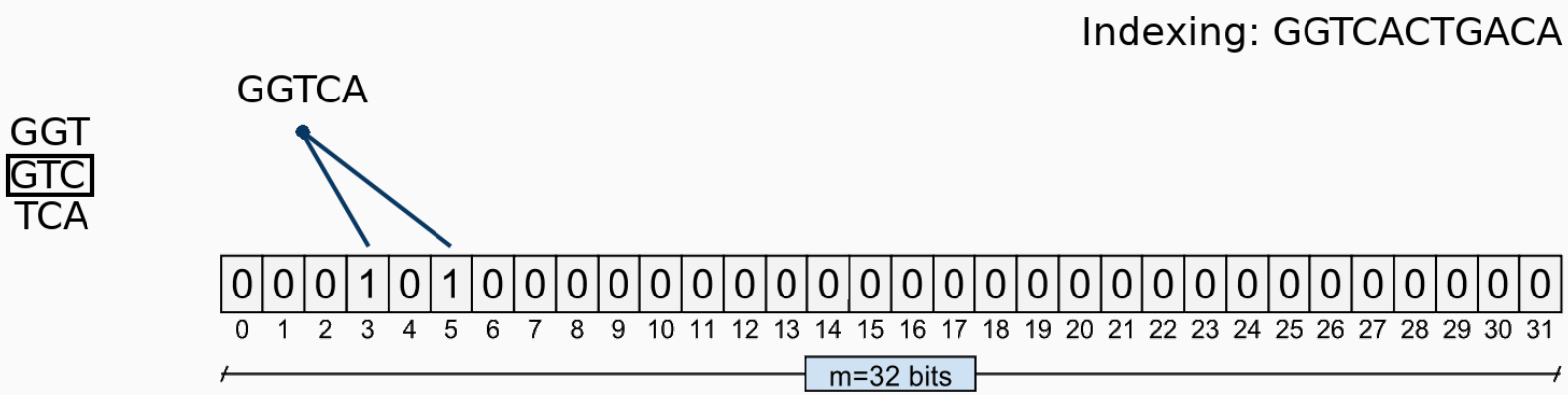
Indexing: GGTCACTGACA



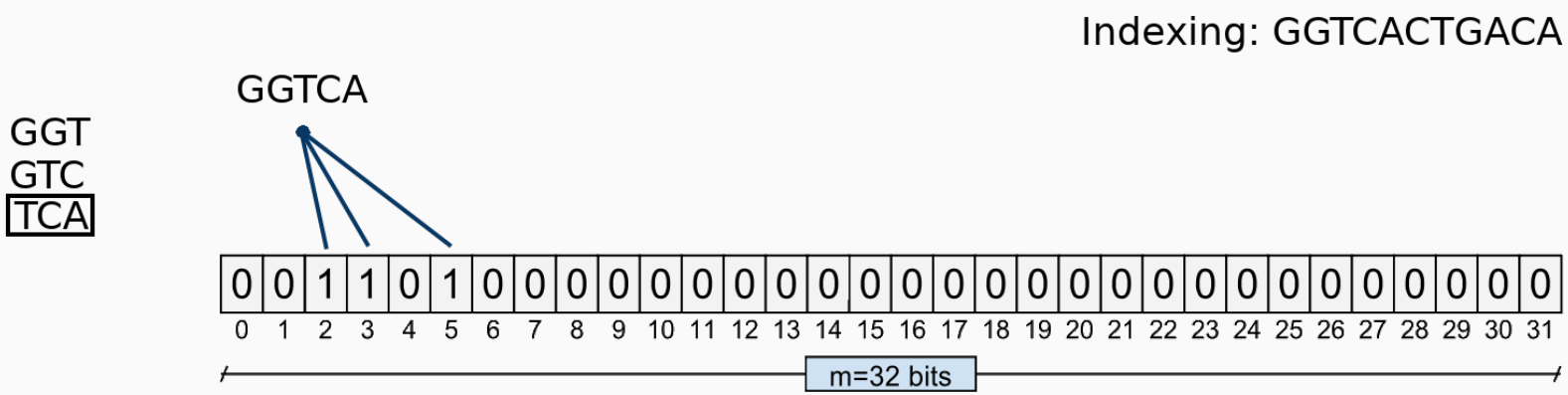
Findere indexation example



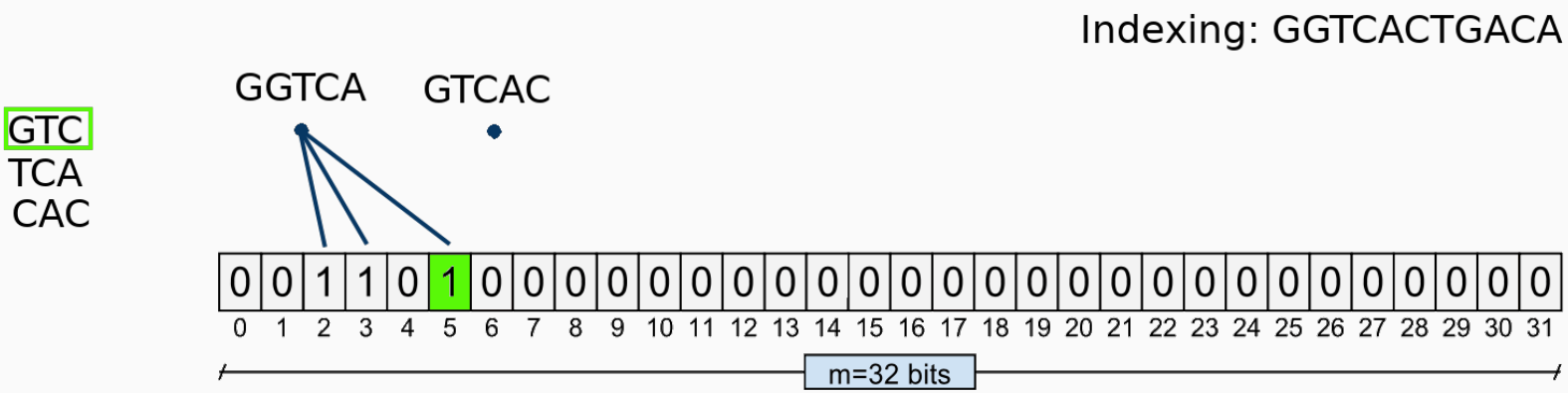
Findere indexation example



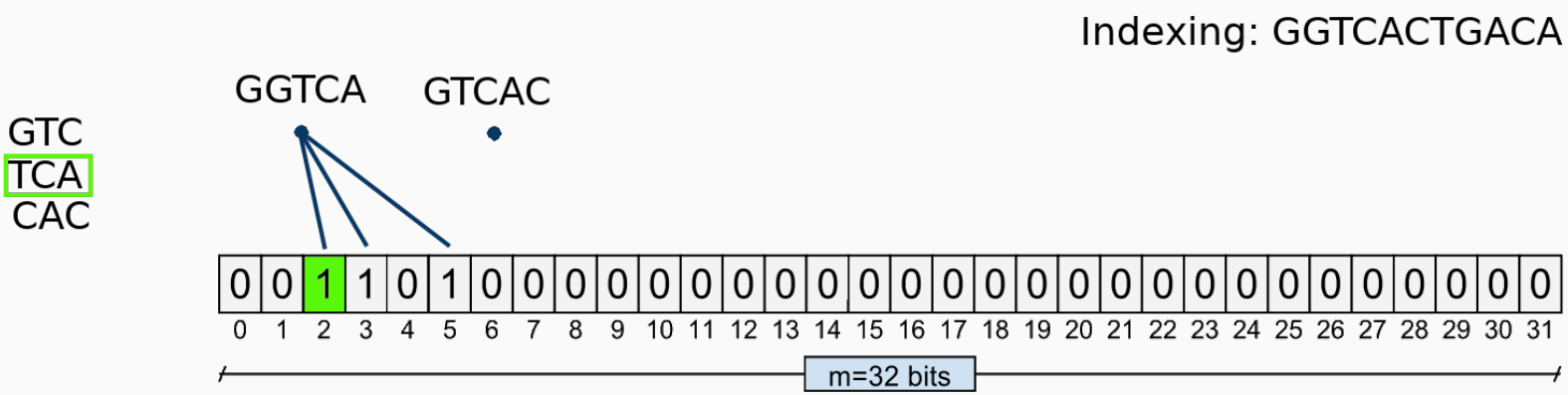
Findere indexation example



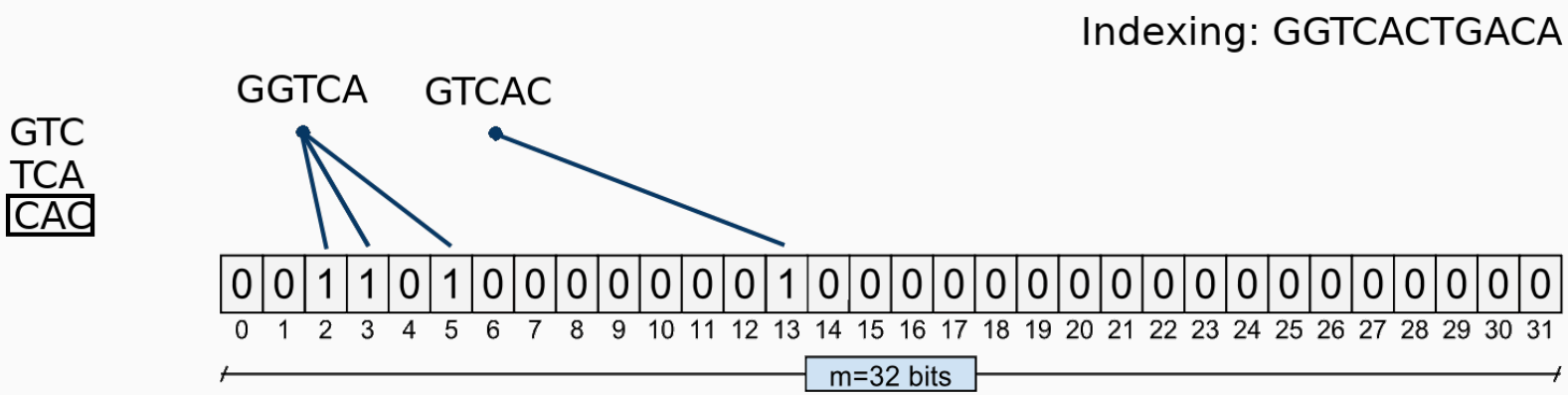
Findere indexation example



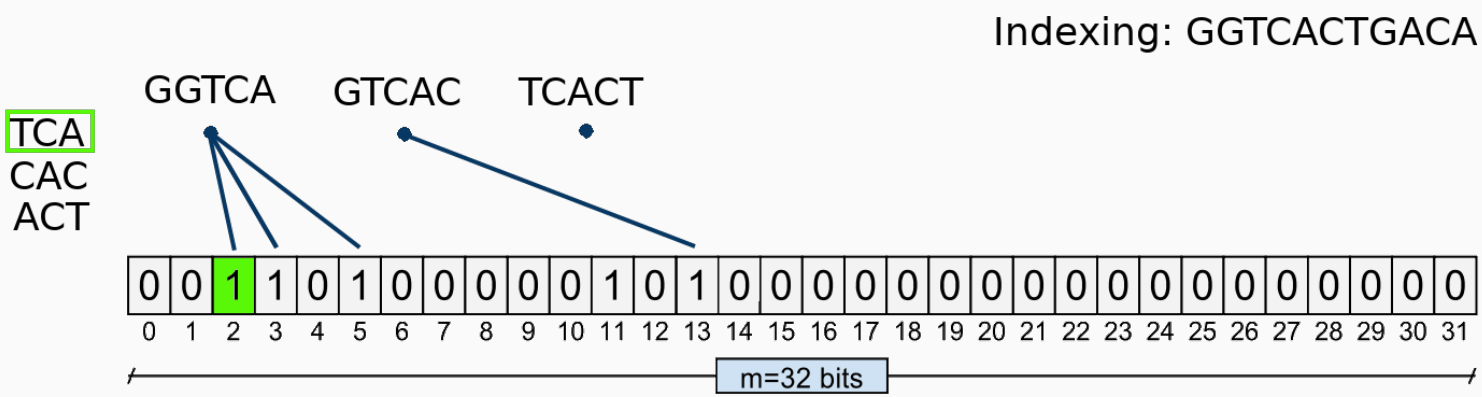
Findere indexation example



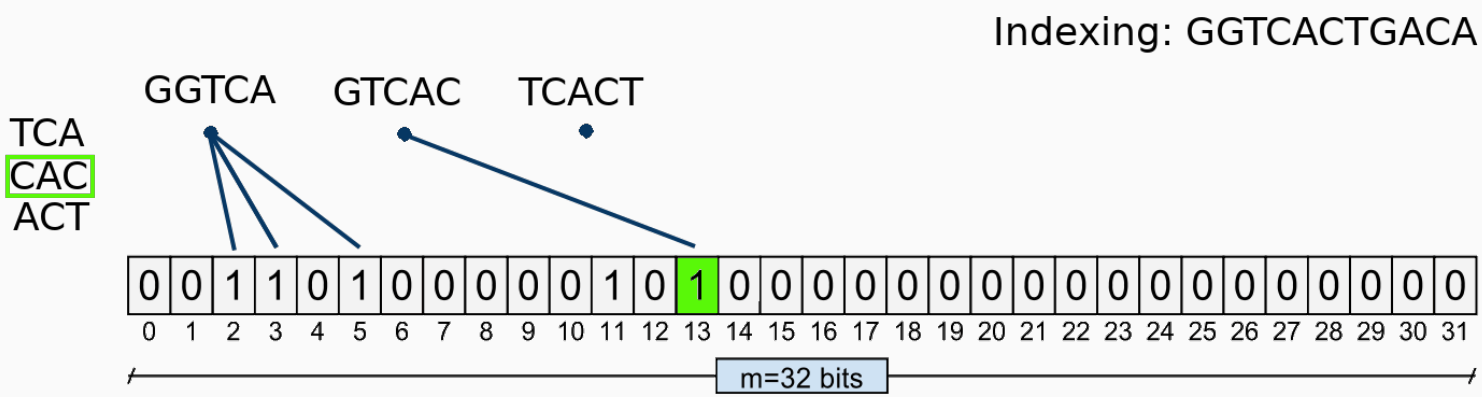
Findere indexation example



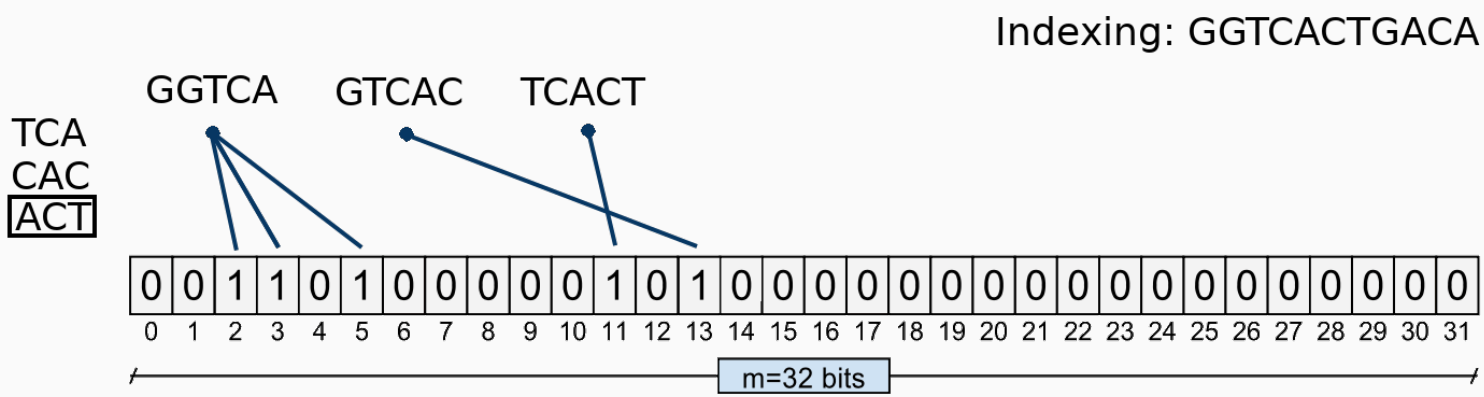
Findere indexation example



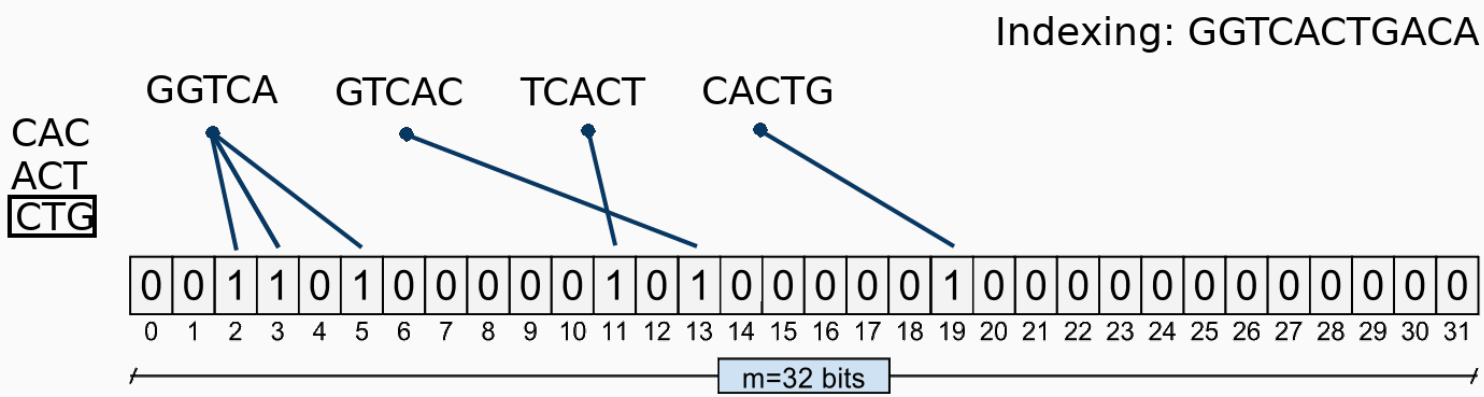
Findere indexation example



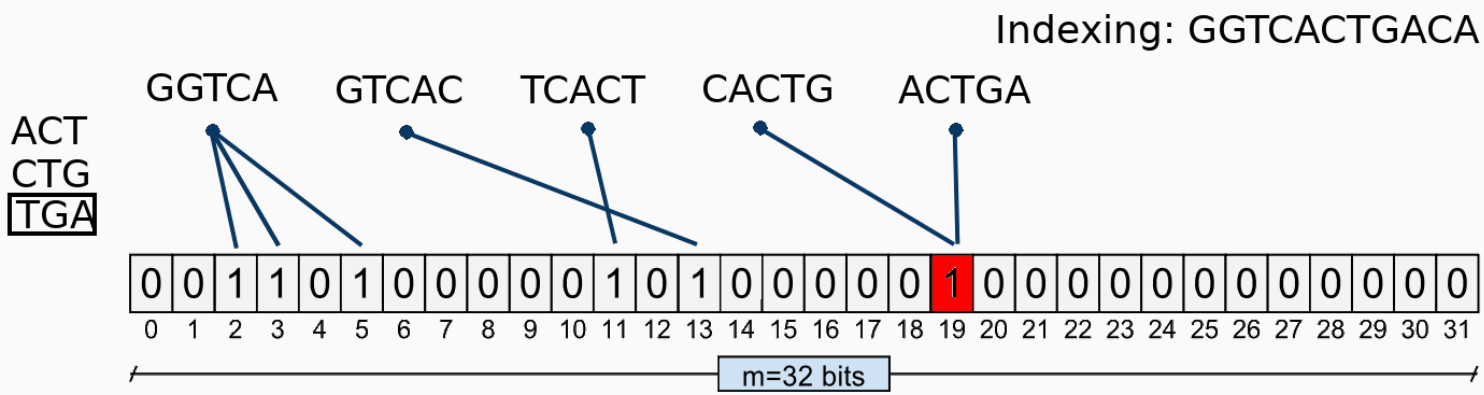
Findere indexation example



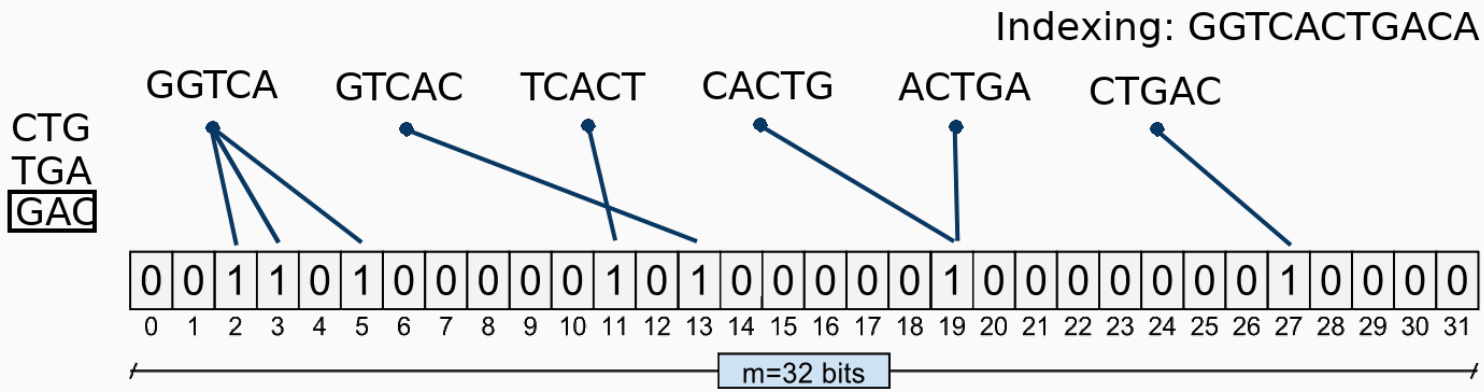
Findere indexation example



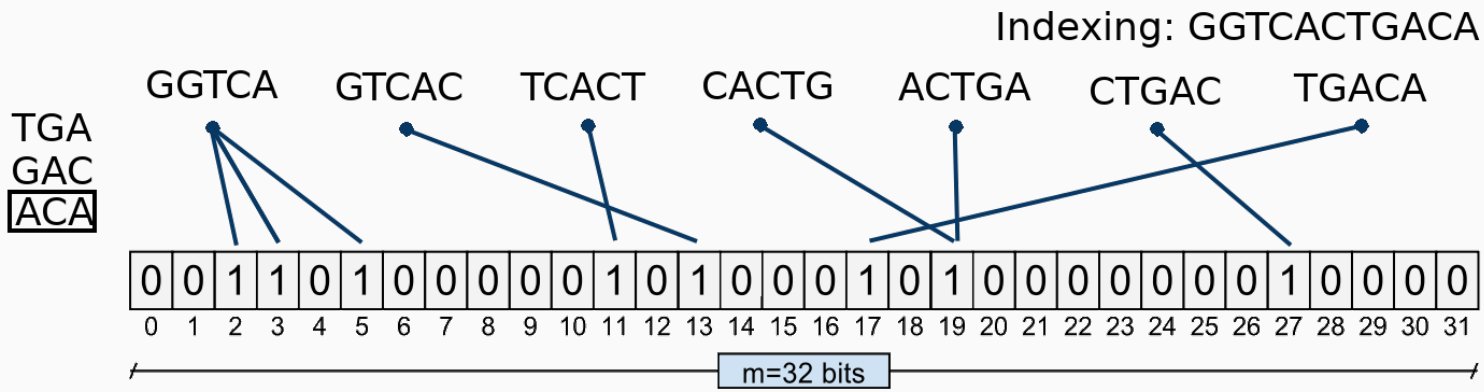
Findere indexation example



Findere indexation example



Findere indexation example



New method: findere

Let us change the query part now: we now need to query k-mers.
z k-mers are shared among two K-mers: no need to query them again

Findere query example

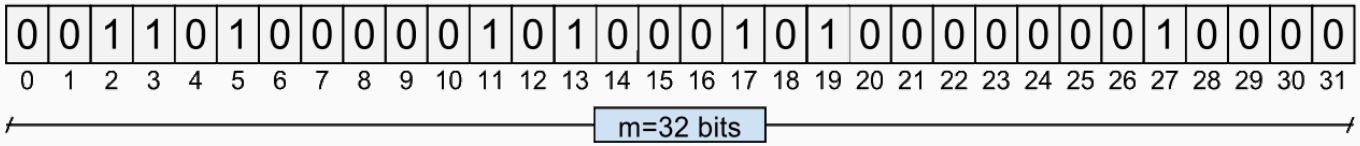
Indexed: GGTCACTGACA

Querying: GGTCACTGACA

Findere query example

Indexed: GGTCACTGACA

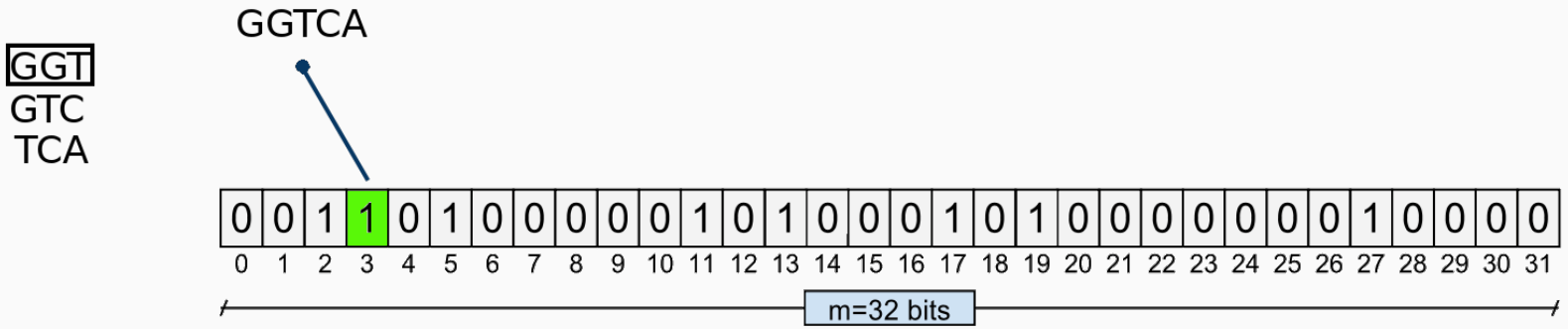
Querying: GGTCACTGACA



Findere query example

Indexed: GGTCACTGACA

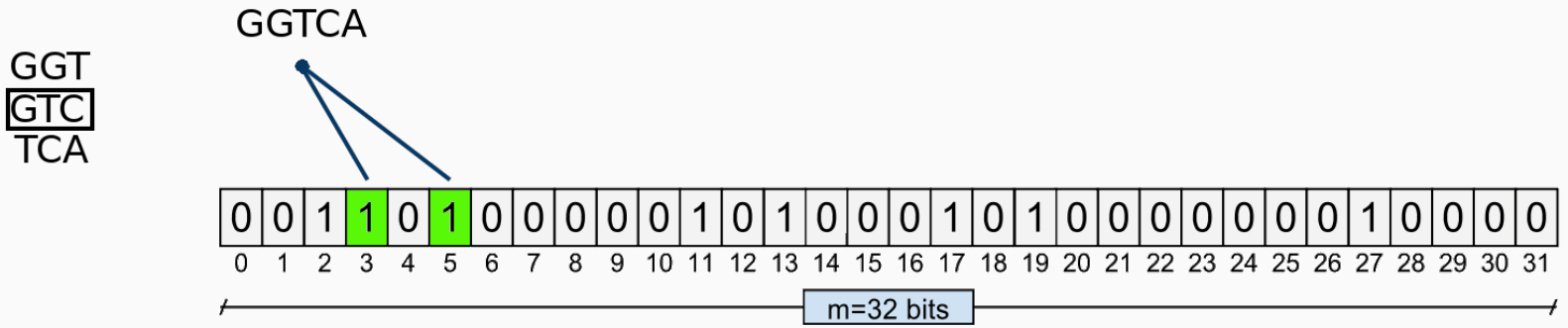
Querying: GGTCACTGACA



Findere query example

Indexed: GGTCACTGACA

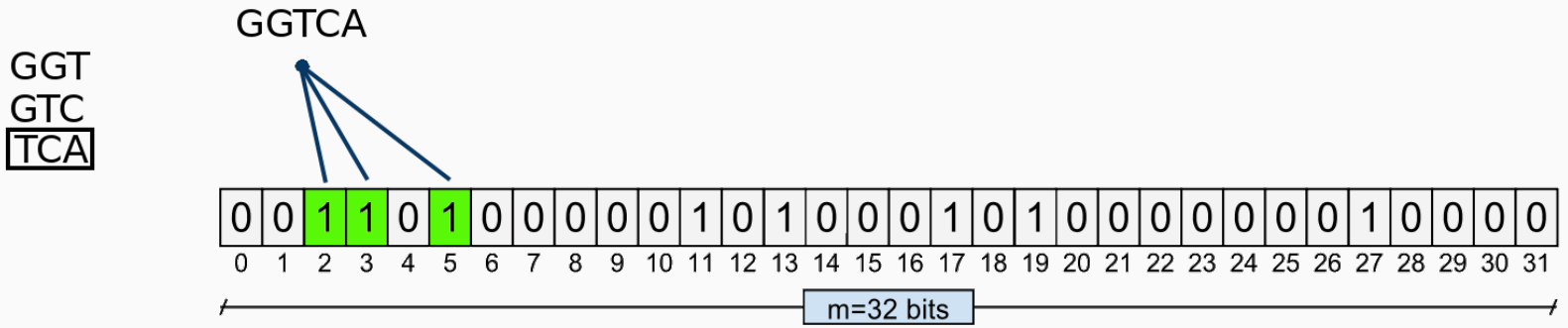
Querying: GGTCACTGACA



Findere query example

Indexed: GGTCACTGACA

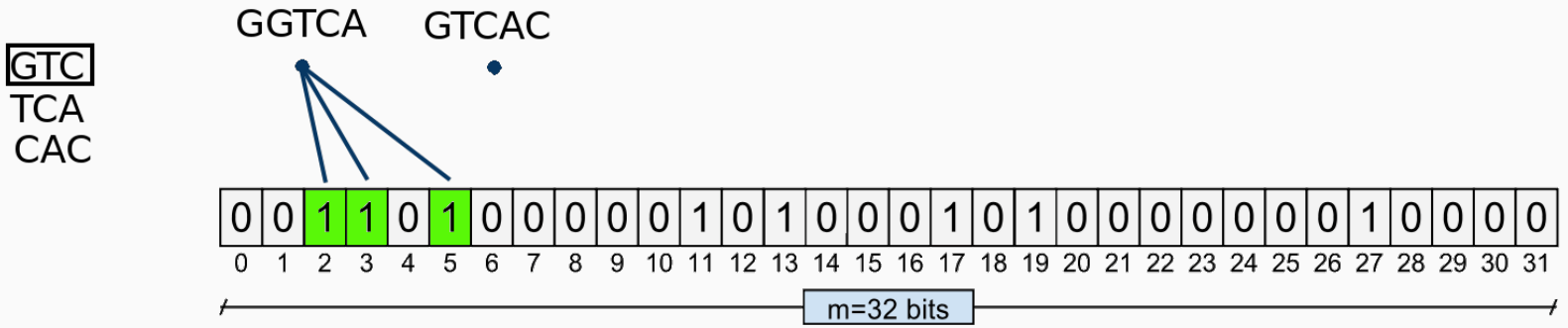
Querying: GGTCACTGACA



Findere query example

Indexed: GGTCACTGACA

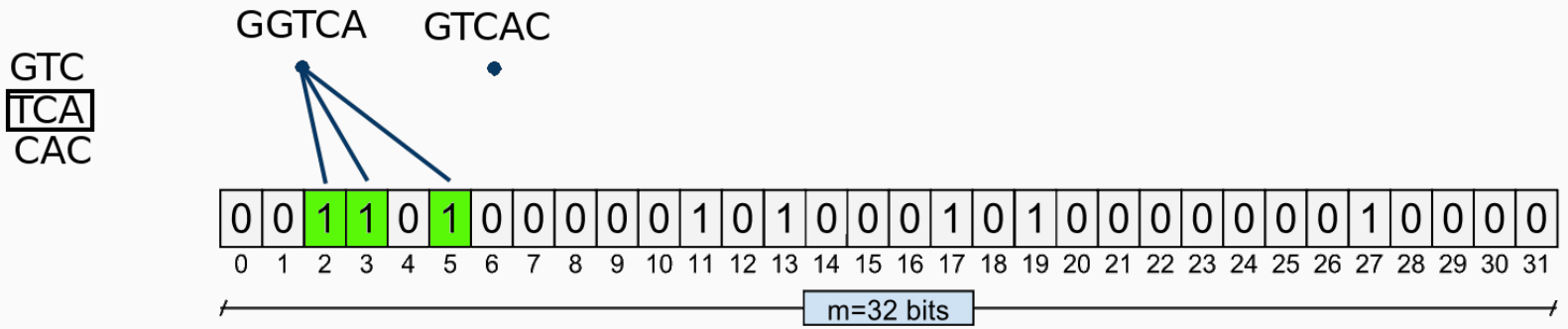
Querying: GGTCACTGACA



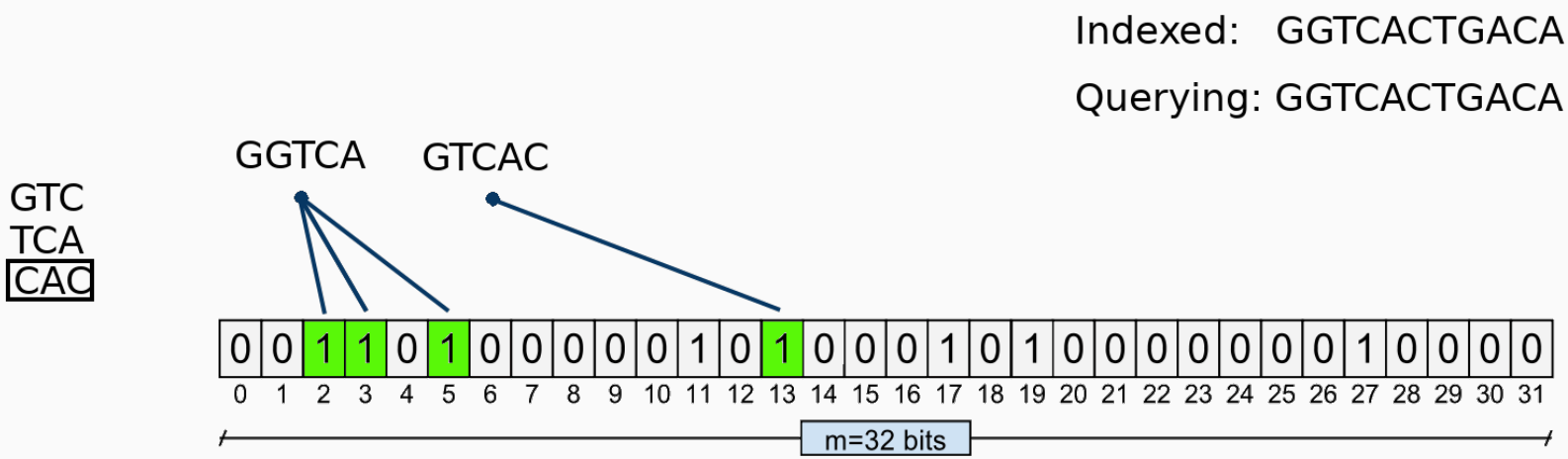
Findere query example

Indexed: GGTCACTGACA

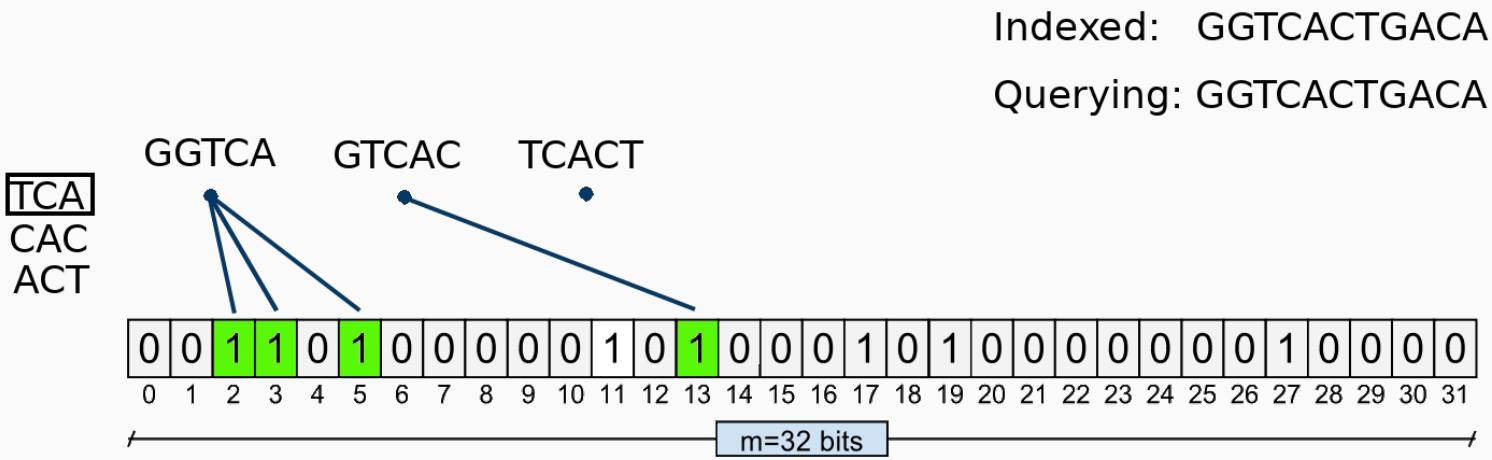
Querying: GGTCACTGACA



Findere query example



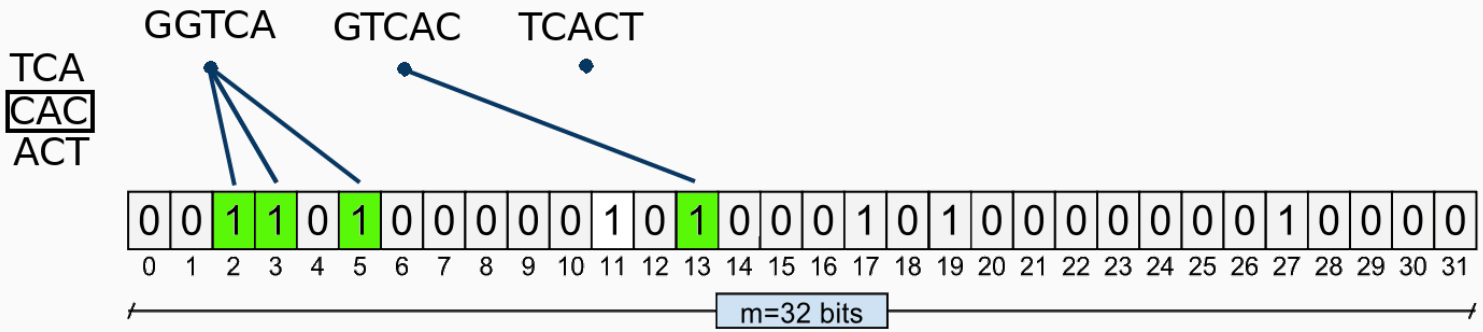
Findere query example



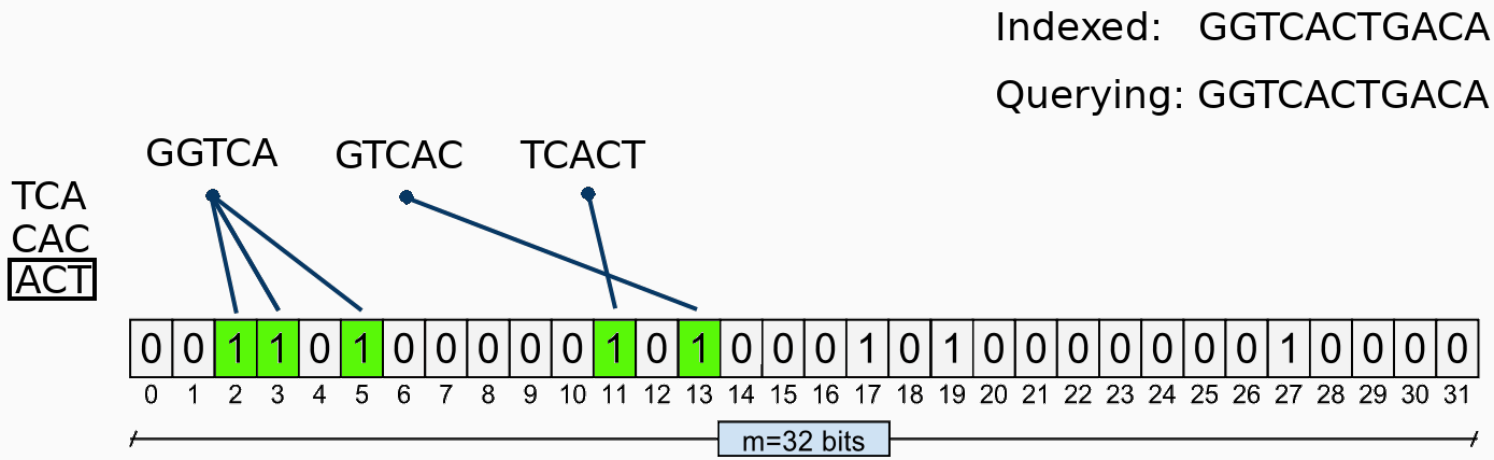
Findere query example

Indexed: GGTCACTGACA

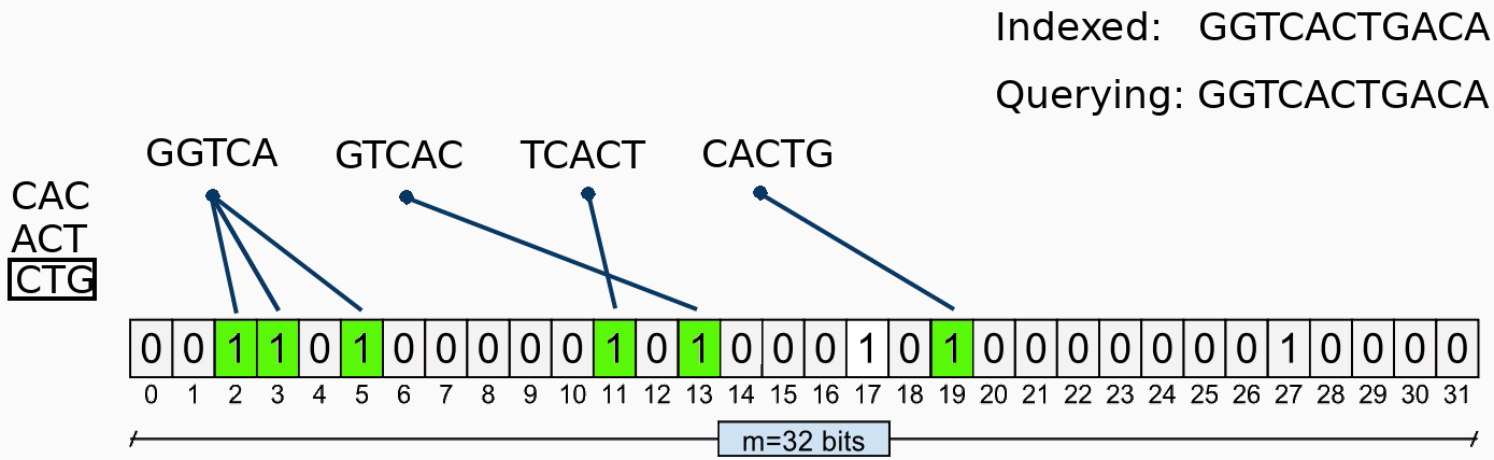
Querying: GGTCACTGACA



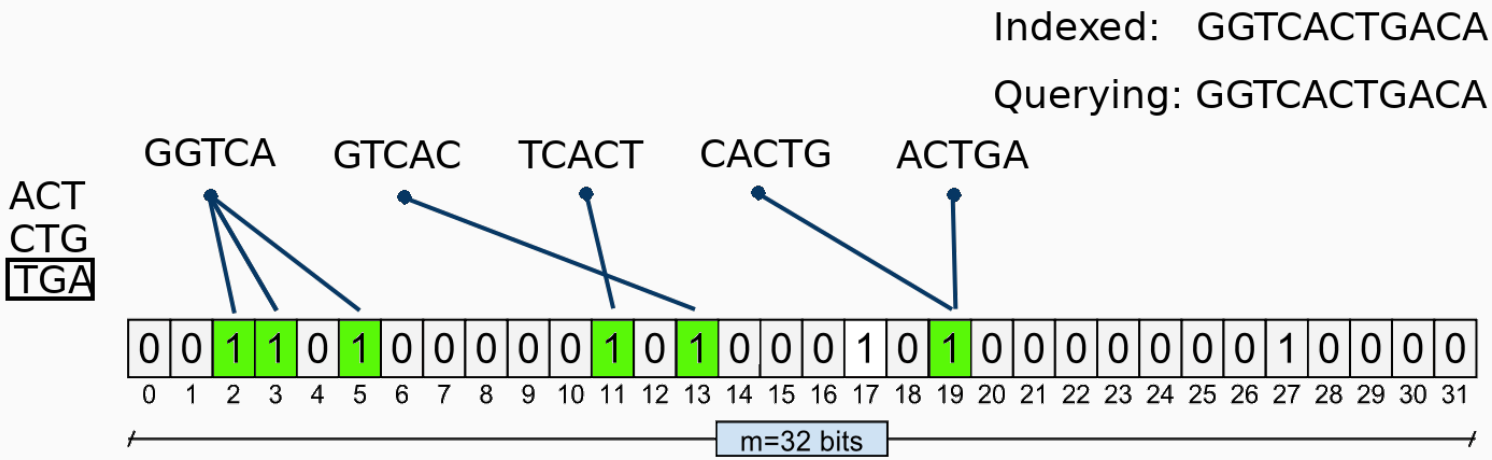
Findere query example



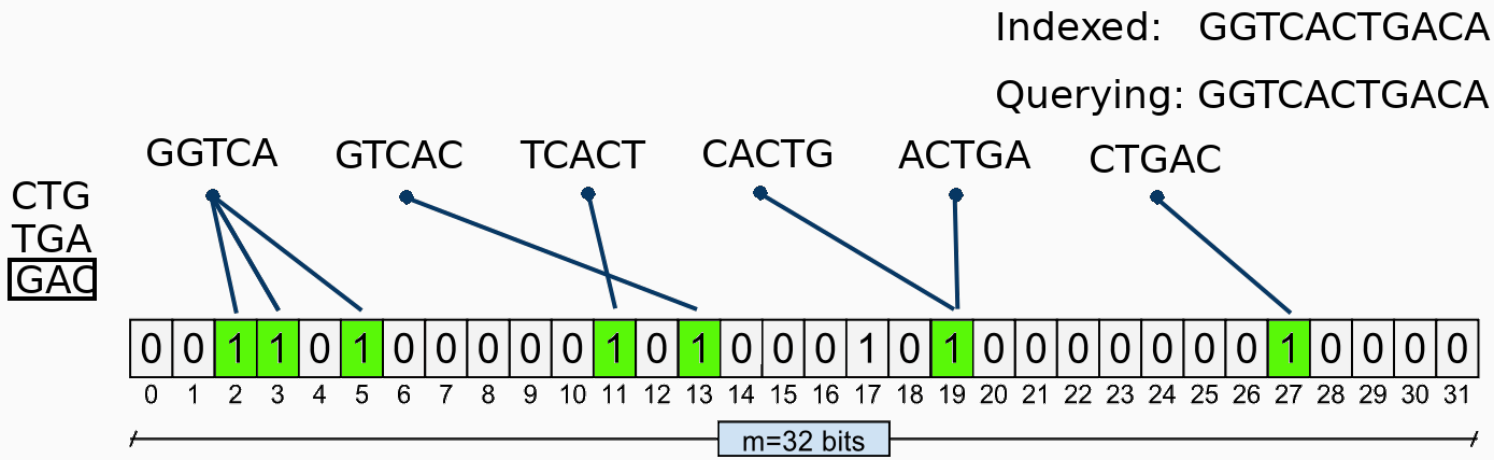
Findere query example



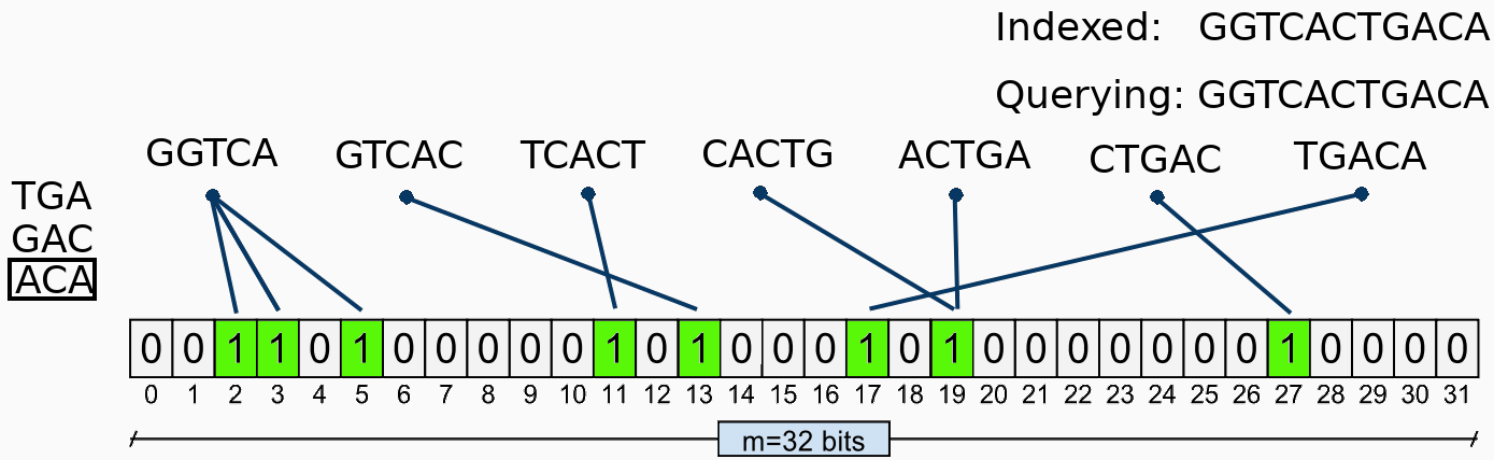
Findere query example



Findere query example



Findere query example



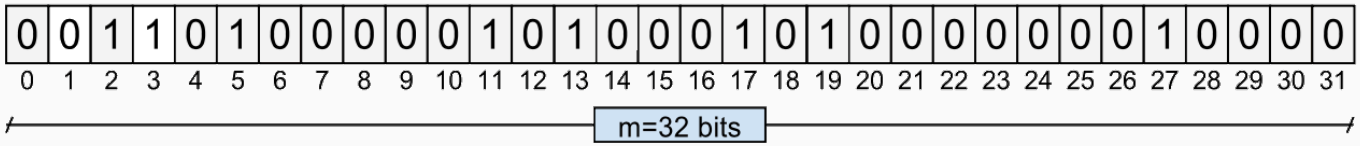
Findere (negative) query example

Indexed: GGTCACTGACA

Findere (negative) query example

Indexed: GGTCACTGACA

Querying: CGTCATTGGCA

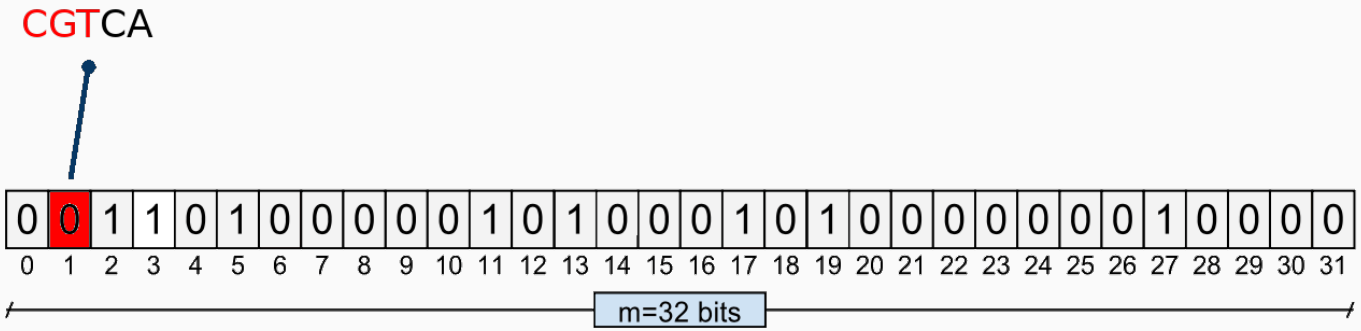


Findere (negative) query example

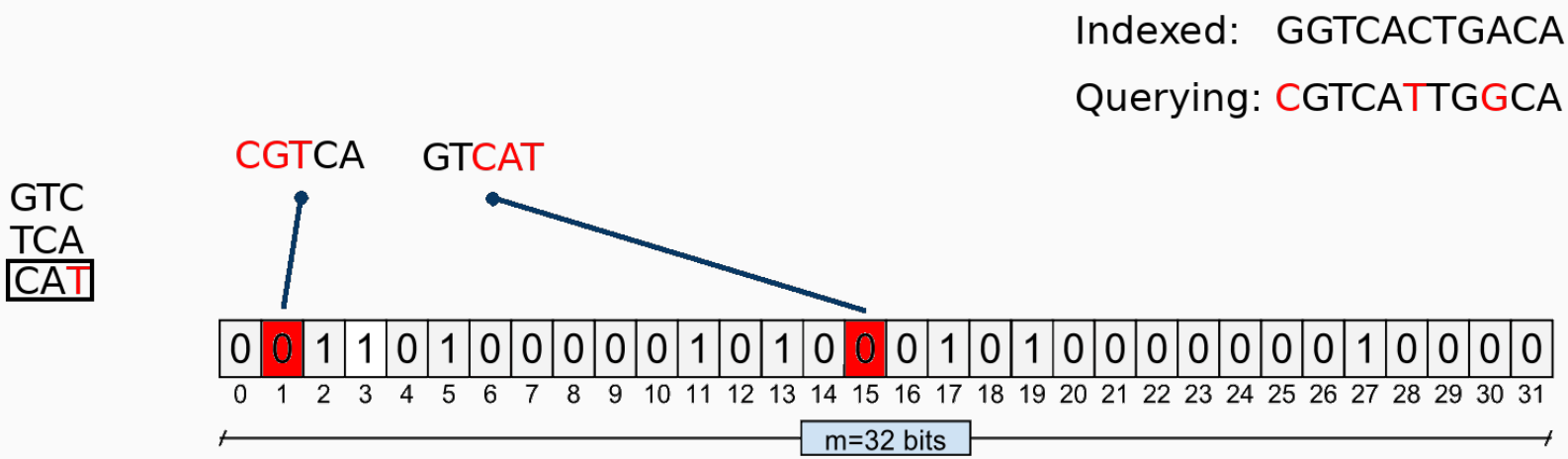
Indexed: GGTCACTGACA

Querying: CGTCATTGGCA

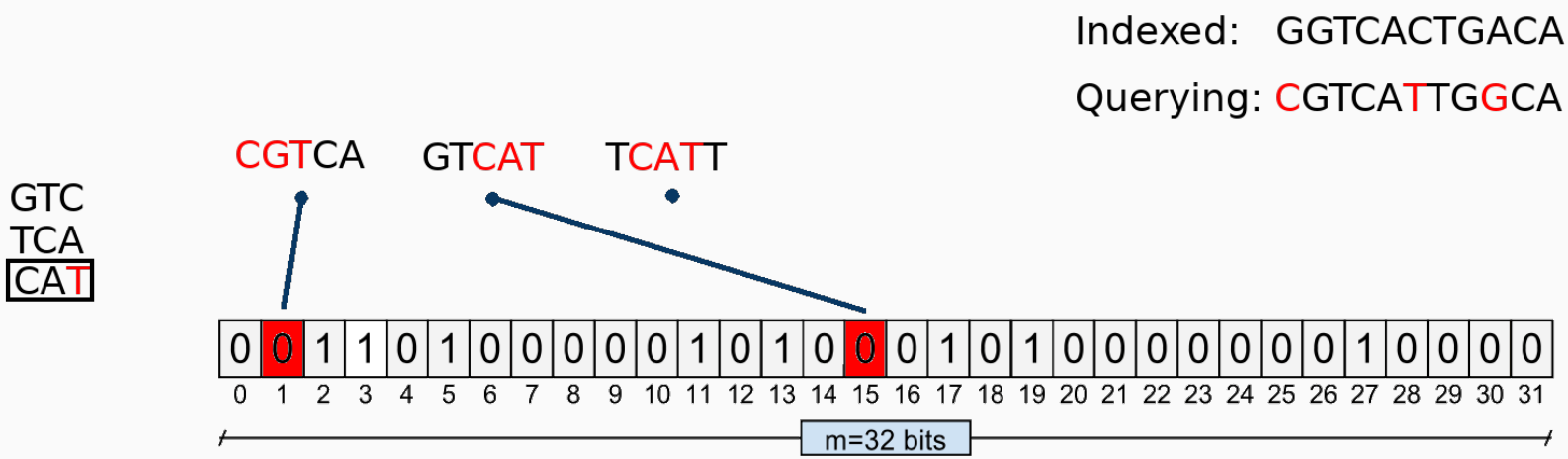
CGT
GTC
TCA



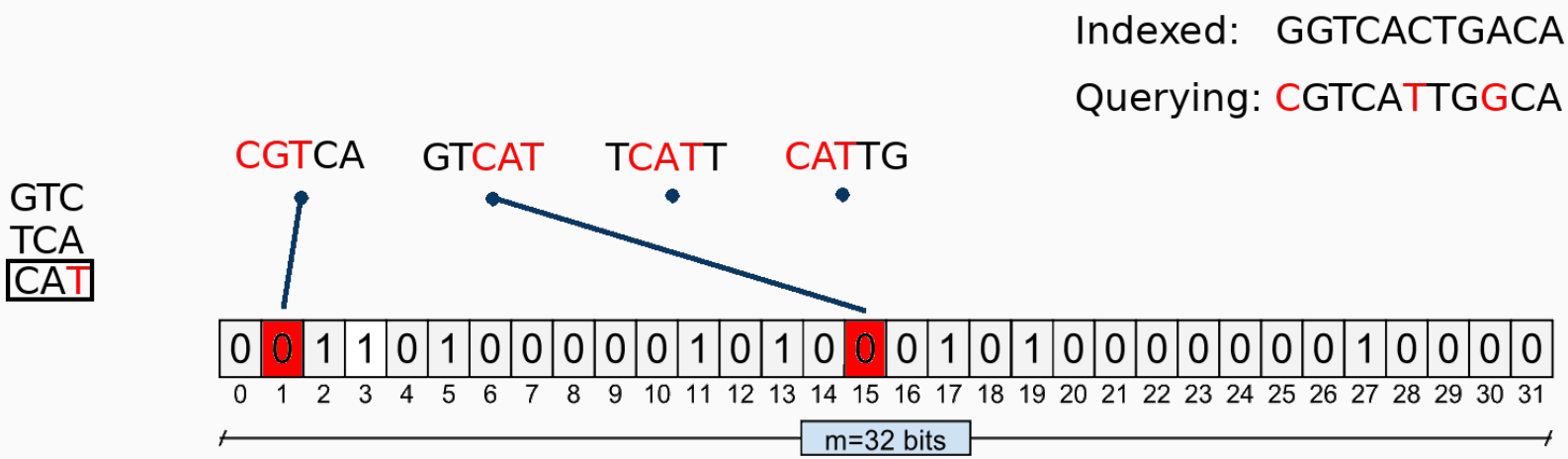
Findere (negative) query example



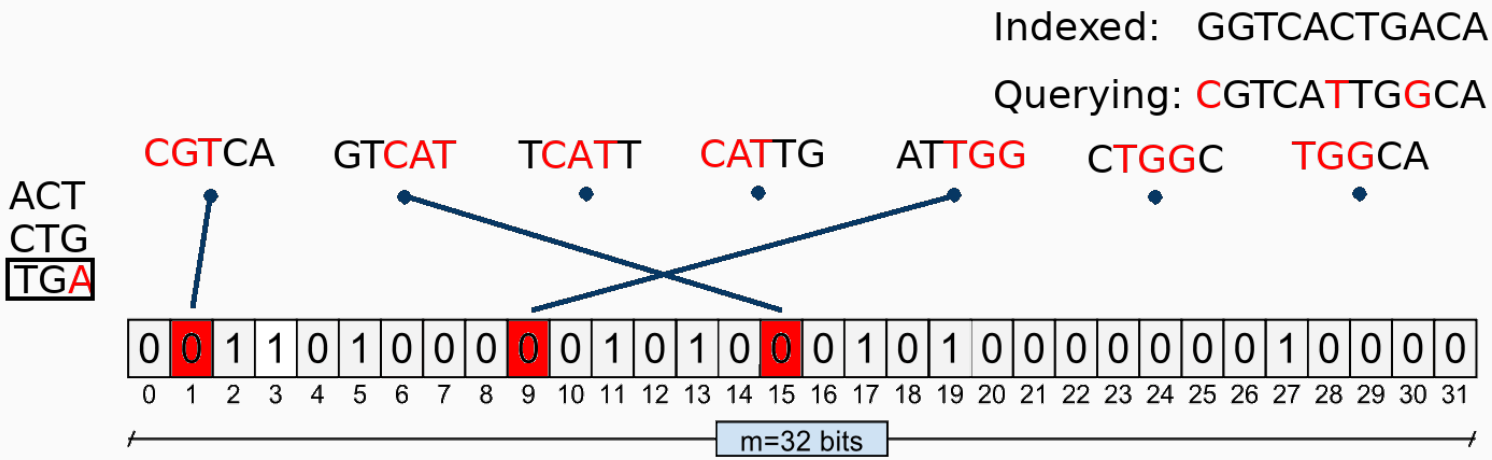
Findere (negative) query example



Findere (negative) query example



Findere (negative) query example



findere: 'skip' optimisation

If two negatives k-mers are z positions away, there is at most $z - 1$ positive k-mers in between

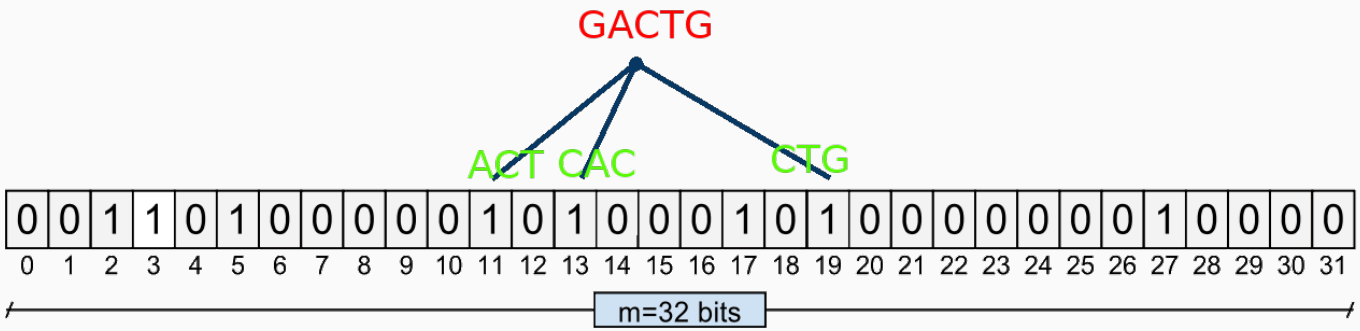
- Higher $z \implies$ less query in a negative stretch

Choosing z

For a chosen K, if z is too close to K, then findere will index and query very small k-mers. In such case, the probability of having indexed all those k-mers is *high*.

Findere query example

Indexed: GGTCACTGACA



Summary

Some results

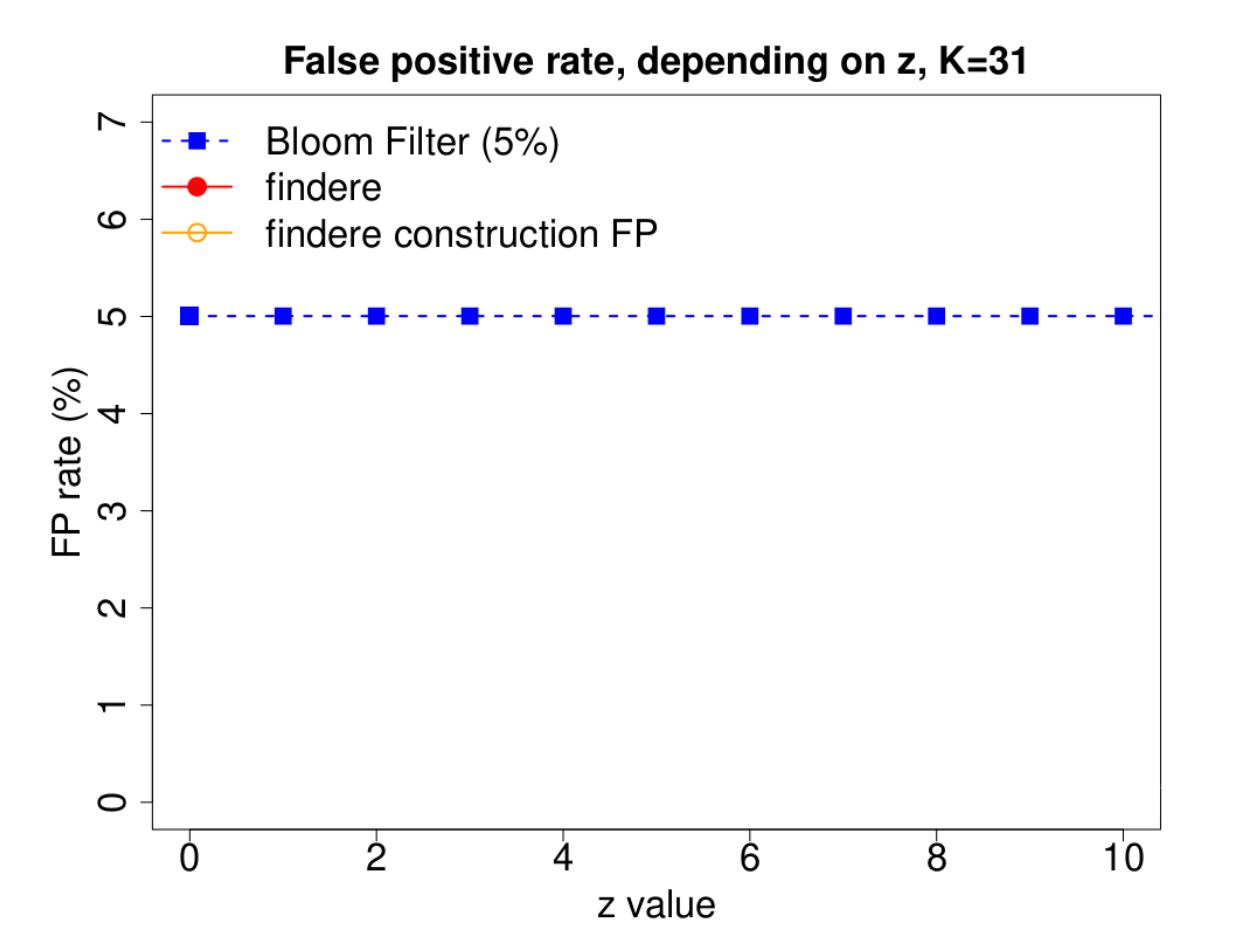


Figure 2: False positive rate for a classic query on a Bloom filter vs using findere. HMP sample SRS014107 queried against sample SRS016349.

Some results

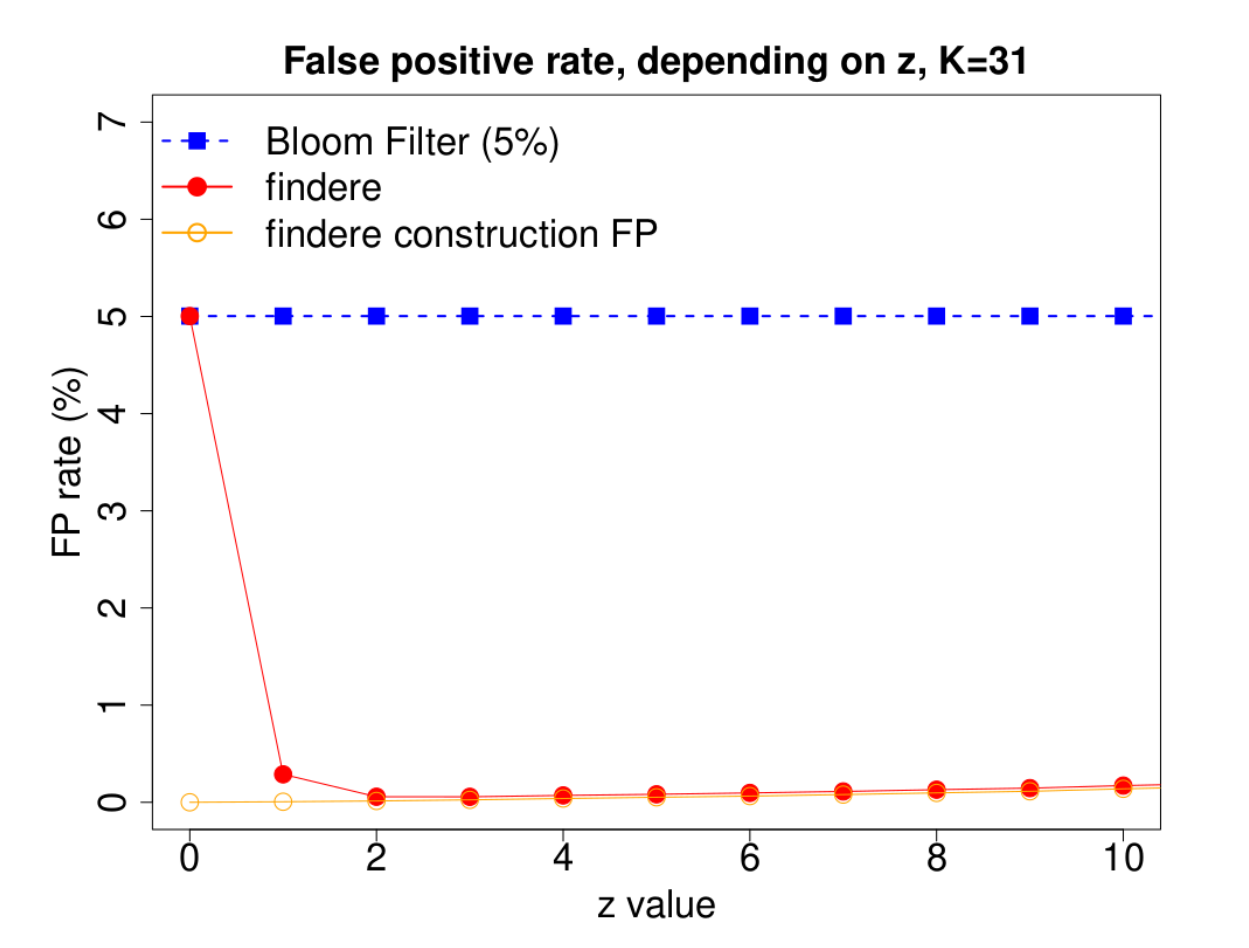


Figure 3: False positive rate for a classic query on a Bloom filter vs using findere. HMP sample SRS014107 queried against sample SRS016349.

Some results

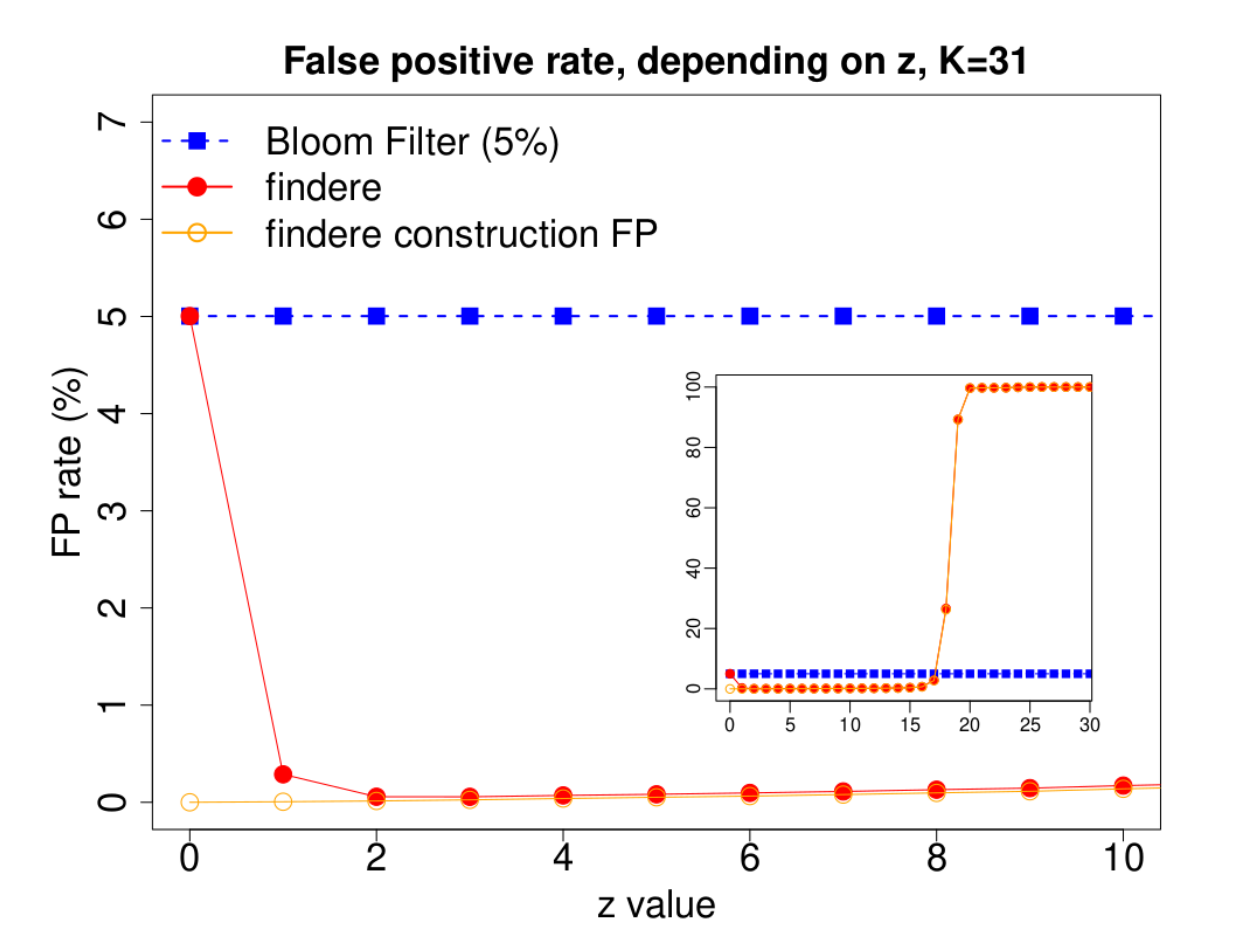


Figure 4: False positive rate for a classic query on a Bloom filter vs using findere. HMP sample SRS014107 queried against sample SRS016349.

Some results

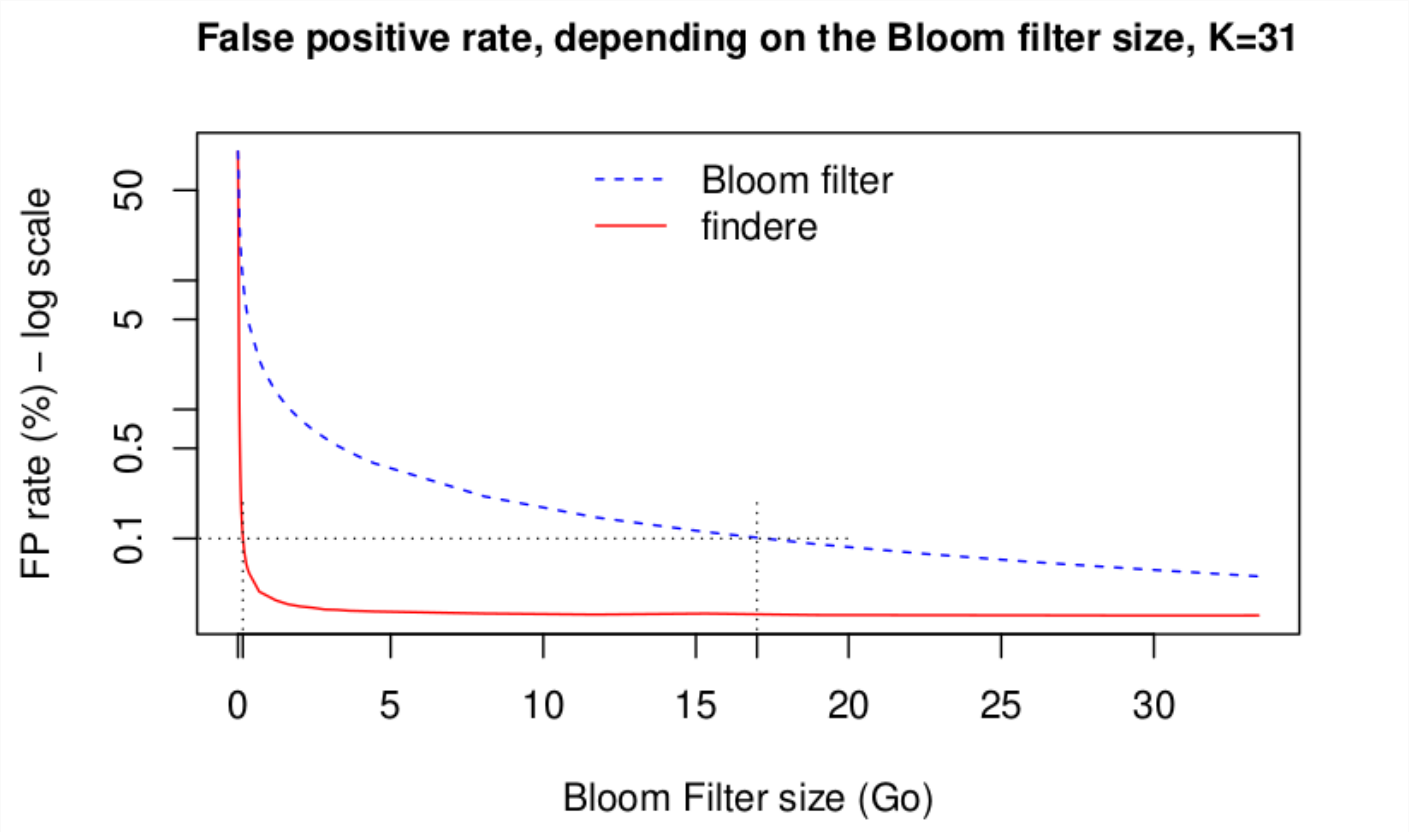


Figure 5: False positive rate for a classic query on a Bloom filter vs using findere wrt the size ($K31$, $z = 3$). HMP sample SRS014107 queried against sample SRS016349.

Some results

z	0	1	2	3	4	5	10
BF	42.4						
findere	42.9	43.7	24.3	17.5	14.1	12.0	8.6

Table 1: Query time (second) with and without findere, using a Bloom filter wrt z.

Summary

Work in progress

How to predict the false positive rate when using findere ?
Working on it with

- Mahendra Mariadassou
- Sophie Schbath
- Julie Aubert
- Stephane Robin

For now: able to have a rought estimation

Summary

Take home message

Using findere, we we are able to

- decrease the size required for a Bloom filter by a factor 100
- (or alternatively, decrease the false positive rate)
- while querying it three times faster