Victor Levallois - 21/11/2023

Pierre Peterlongo (Inria, Rennes)

Yoann Dufresne (Institut Pasteur, Paris)

SeqBIM

The Backpack Quotient Filter: A Space-Efficient Approach to Counting Quotient Filter

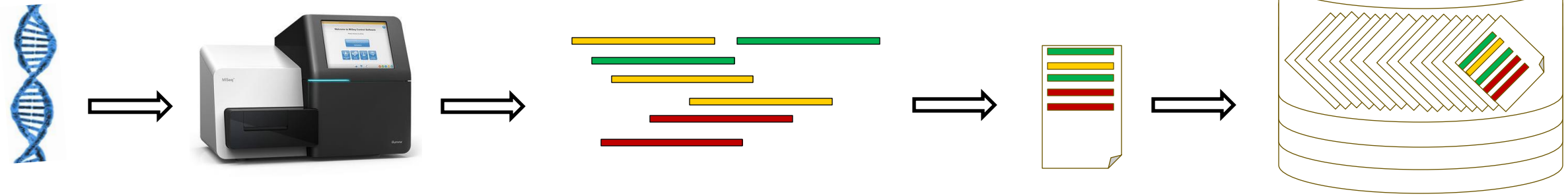
Université de Rennes

Inria

INSTITUT PASTEUR

GENSCALE

Motivations



DNA/RNA

Sequencing

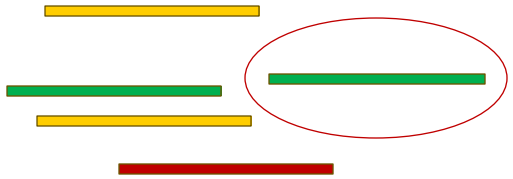
Reads

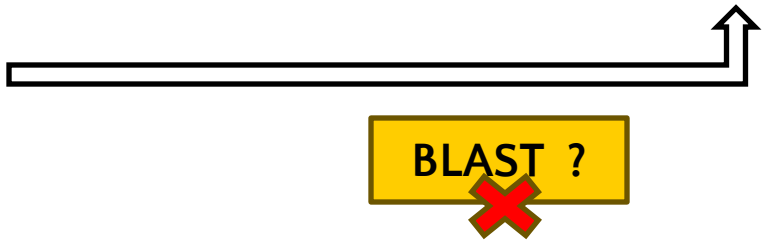
Fastq, fasta

ENA
48 PB

Goal :

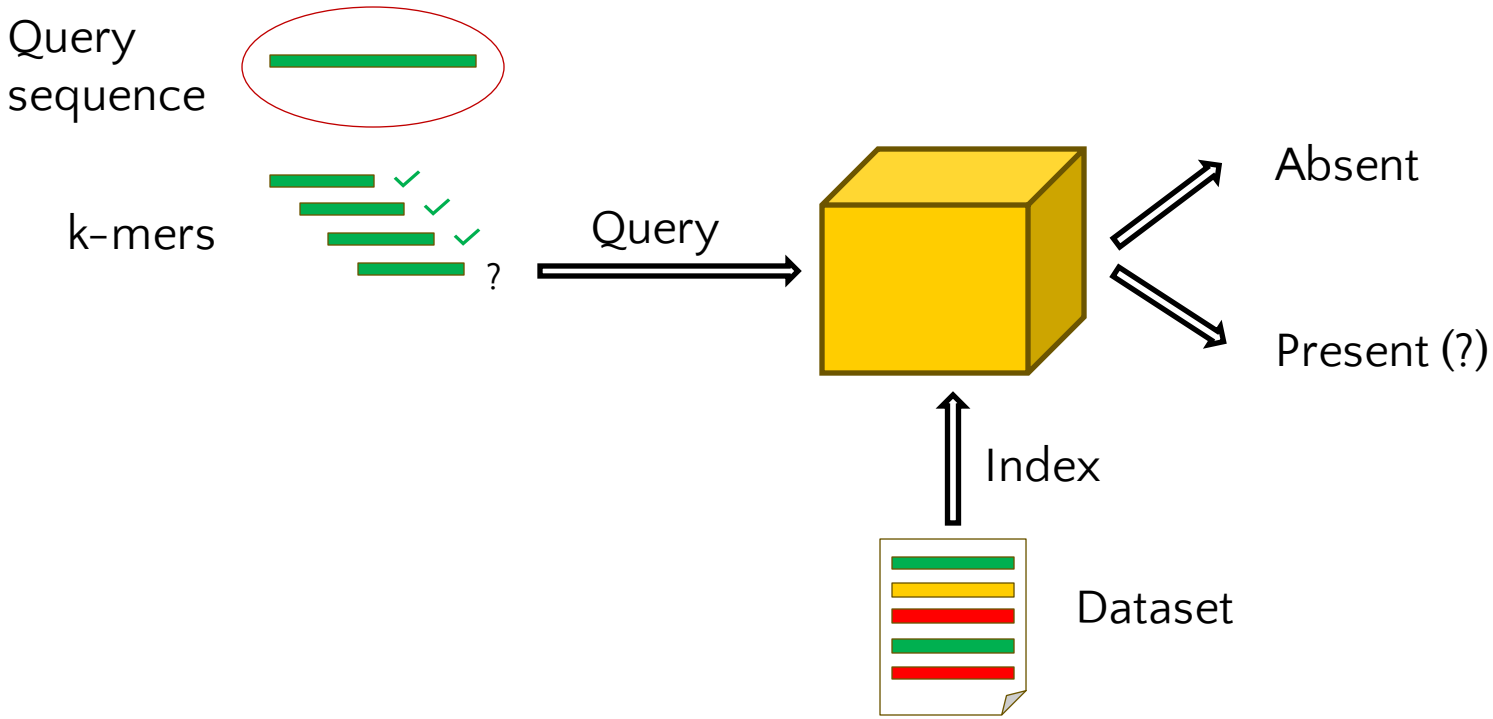
Query
sequence





Motivations

Main operation :



[1] (Fan & al., 2000)
[2] (Putze & al., 2010)
[3] (Solomon & Kingsford, 2016)
[4] (Breslow & al., 2018)
[5] (Dillinger & Walzer, 2011)
[6] (Graf & Lemire, 2022)
[7] (Pandey & al., 2017)

Motivations

AMQs : The Approximate Membership Queries

Bloom Filter

Counting Bloom filter [1]
Blocked Bloom filter [2]
Sequence Bloom Trie [3]

Cukoo Filter

Morton filter [4]

XOR Filter

Ribbon filter [5]
Binary fuse filter [6]

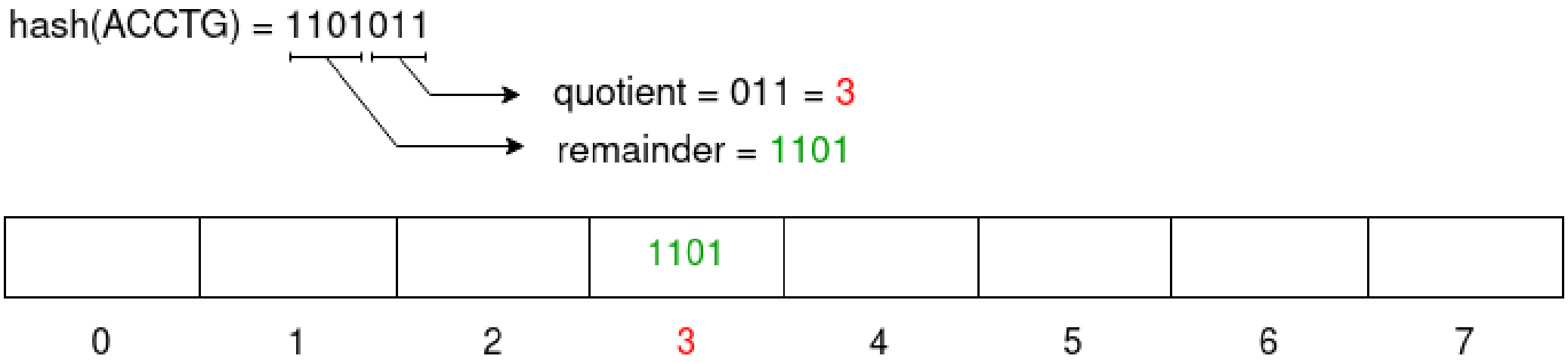
Quotient Filter

Counting Quotient Filter [7]
Backpack Quotient Filter

Context : Quotient Filter

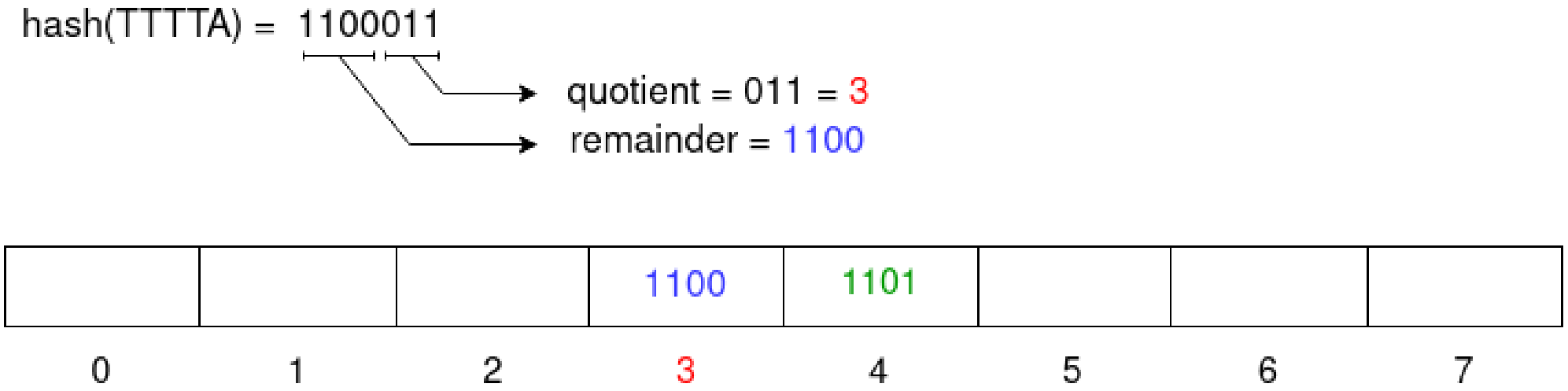
Table containing 2^q **slots** of size r

q = |quotient|
r = |remainder|



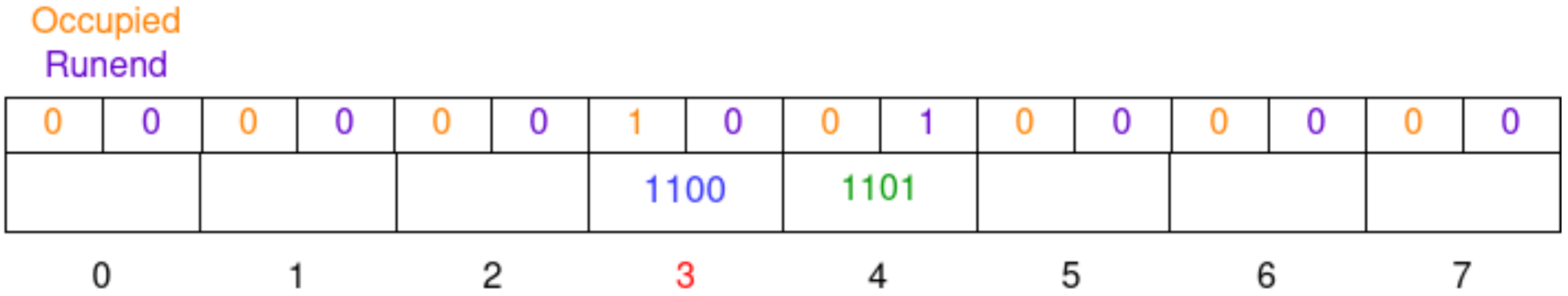
Quotient Filter

(soft) **Collisions** case : remainder shifting
(hard collision : false positive)



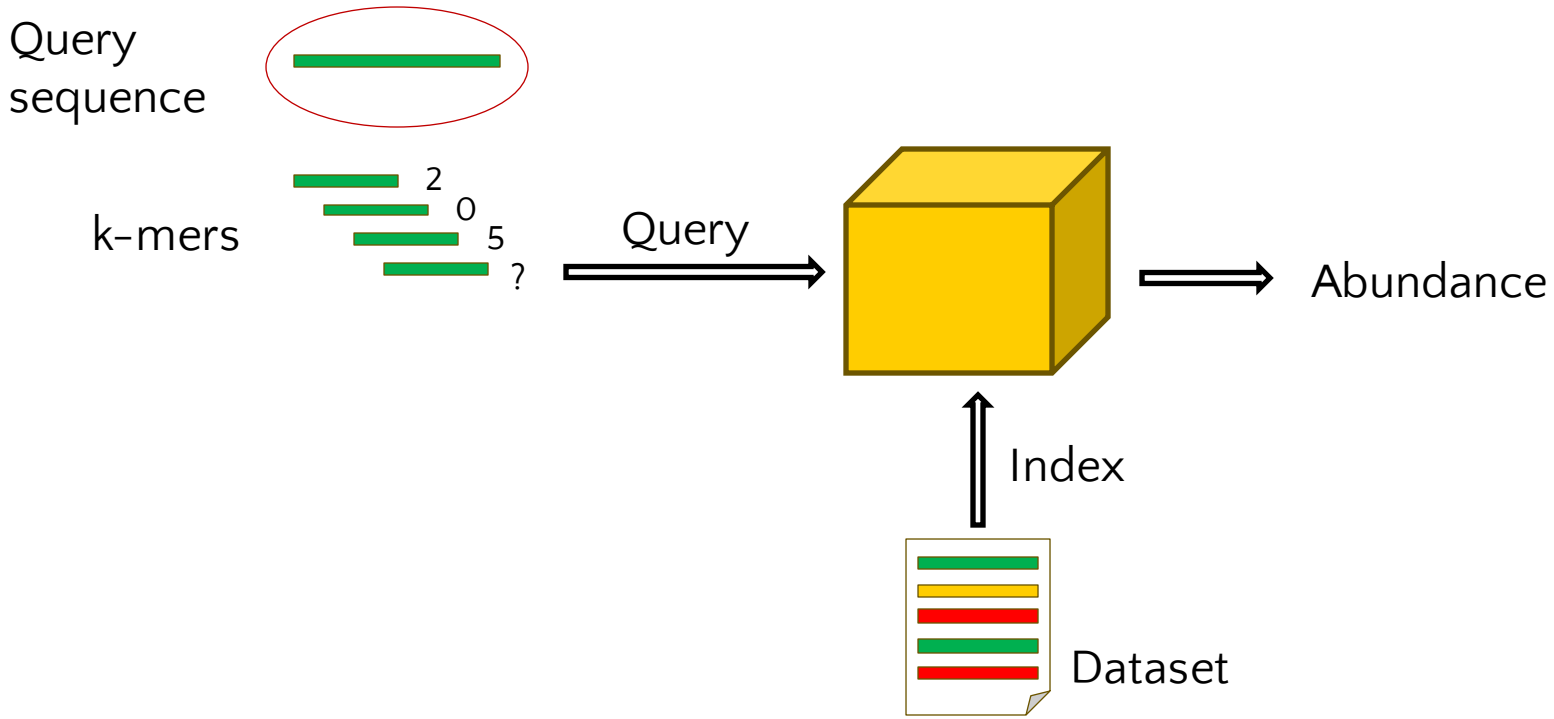
Quotient Filter

(soft) **Collisions** case : remainder shifting
(hard collision : false positive)



-> sub-optimal for multiple insertions of the same k-mer

Abundance Count

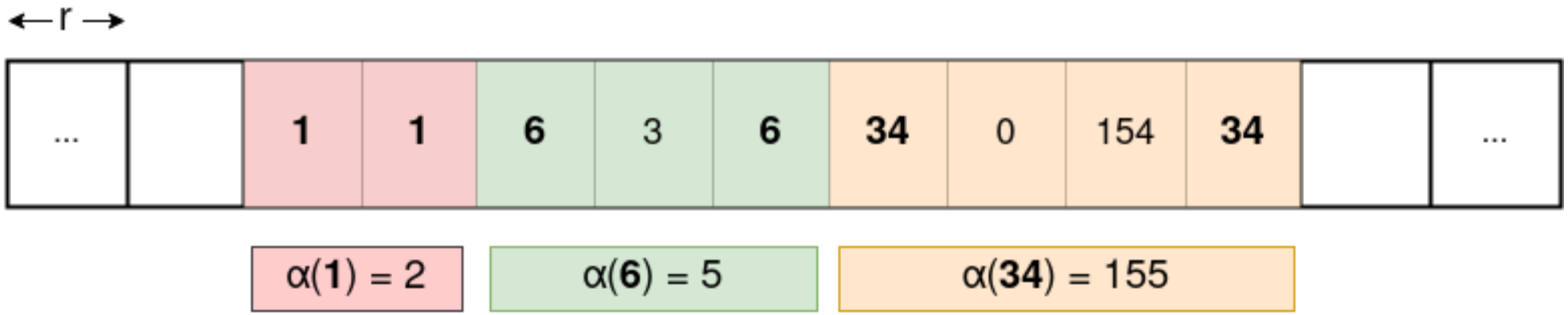


[7] A General-Purpose Counting Filter: Making Every Bit Count (Pandey et al., 2017)

CQF abundances

CQF [7] - Counting Quotient Filter

-> 1 slot = 1 remainder OR 1 count

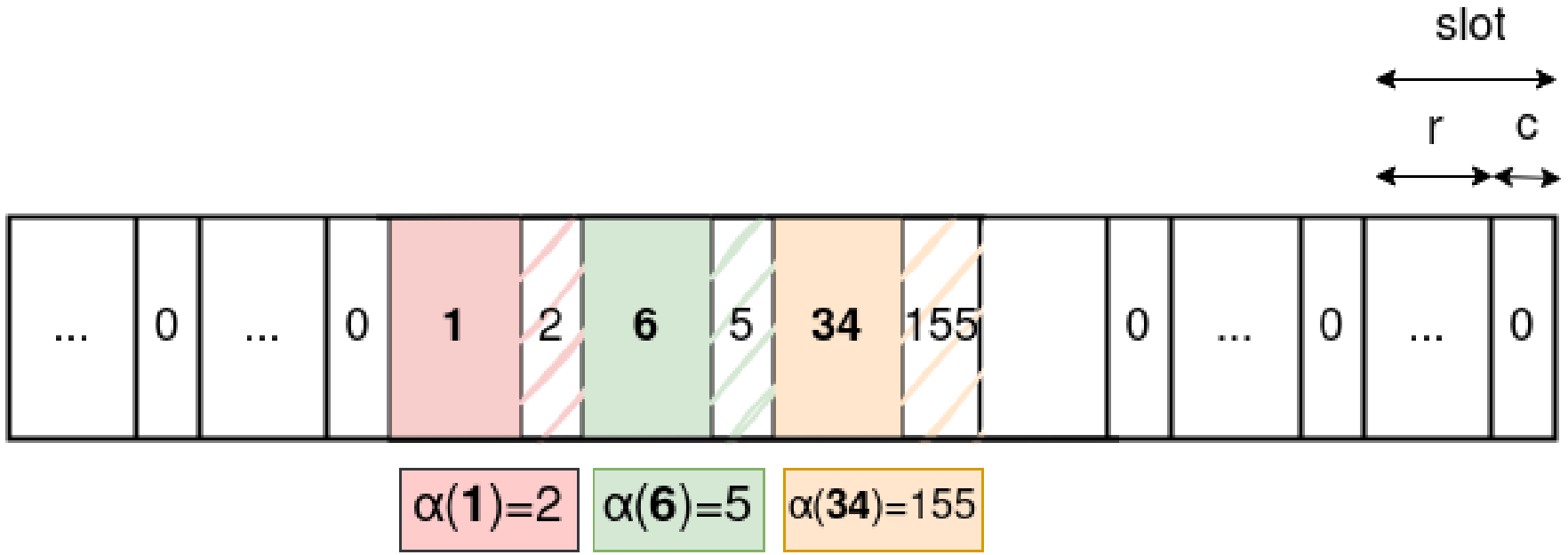


α(**x**) = kmer K abundance for which remainder(K) = **x**

BQF abundances

BQF – Backpack Quotient Filter, our contribution

–> 1 slot = 1 remainder AND 1 count



α(**x**) = kmer K abundance for which remainder(K) = **x**

[8] fimpera: drastic improvement of Approximate Membership Query data-structures with counts (Robidou et Peterlongo, 2022)

BQF abundances

BQF – Backpack Quotient Filter

uses **Fimpera** [8]

z = k-s = 2

/!\ construction false positive /!\

-> space gain (2*z bits / slot)

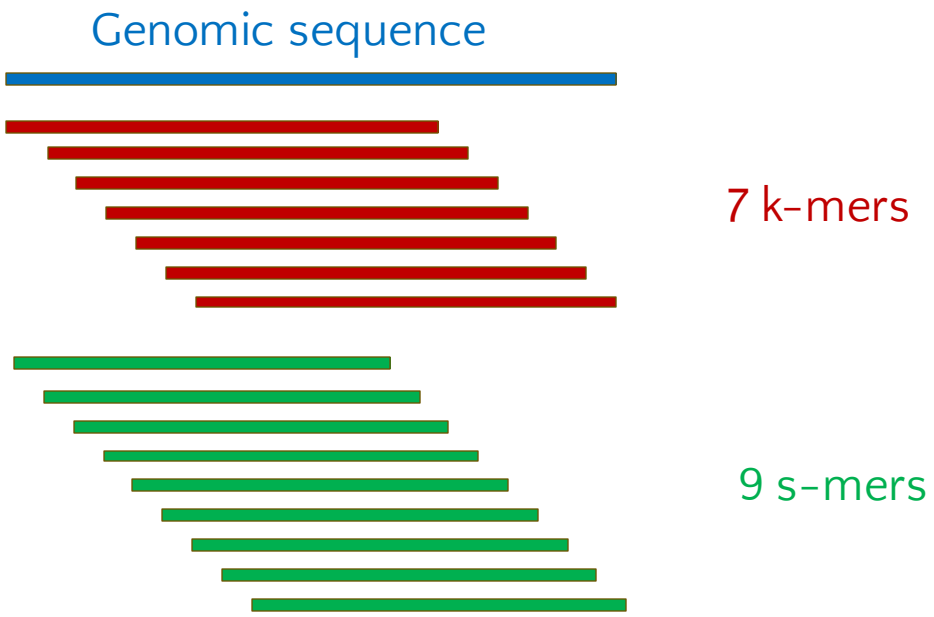
[8] fimpera: drastic improvement of Approximate Membership Query data-structures with counts (Robidou et Peterlongo, 2022)

BQF abundances

BQF – Backpack Quotient Filter

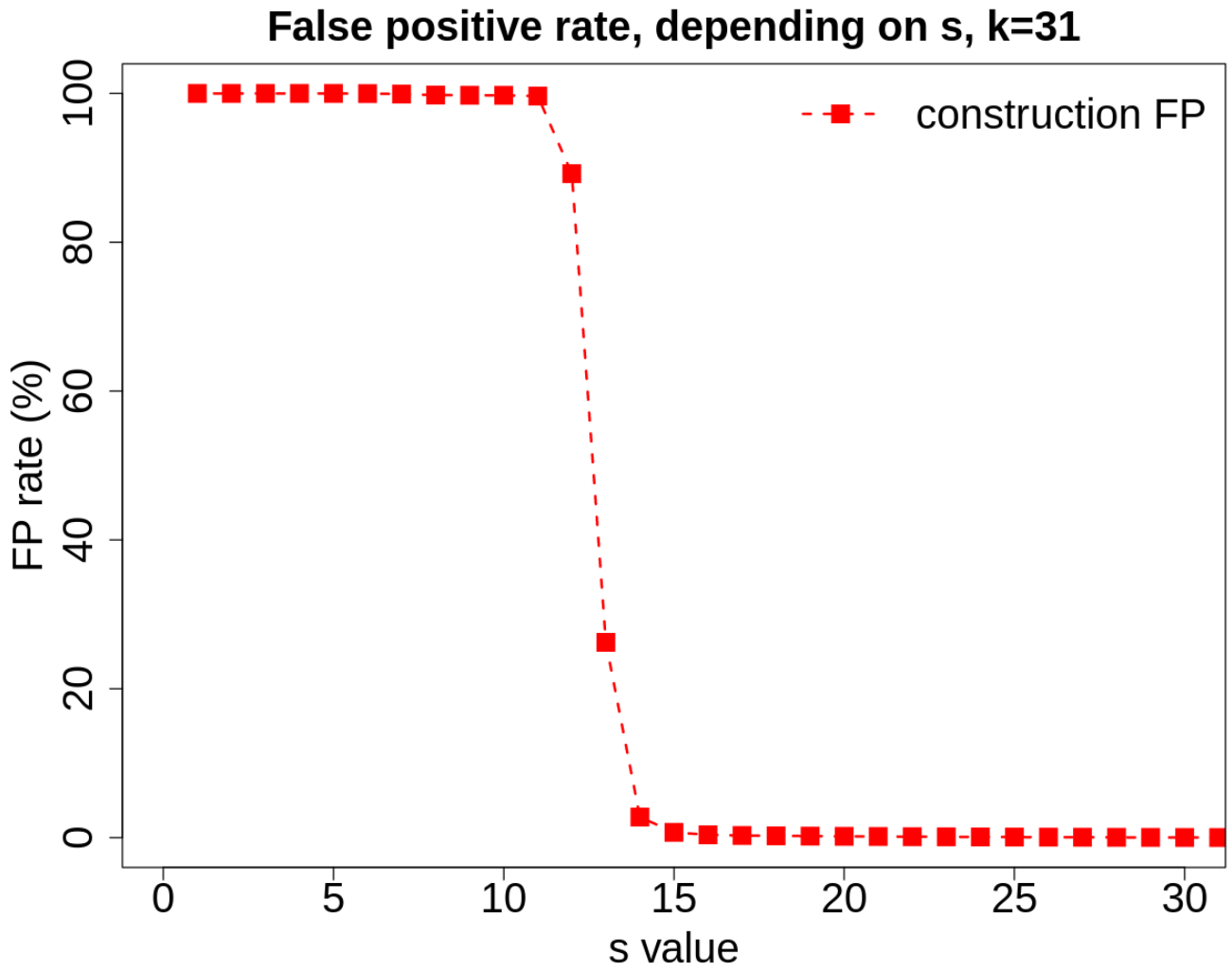
uses **Fimpera** [8]

Speed impact ?



BQF fimpera

-> Linear gain as **s** decreases, but threshold reached when requests become too sensitive



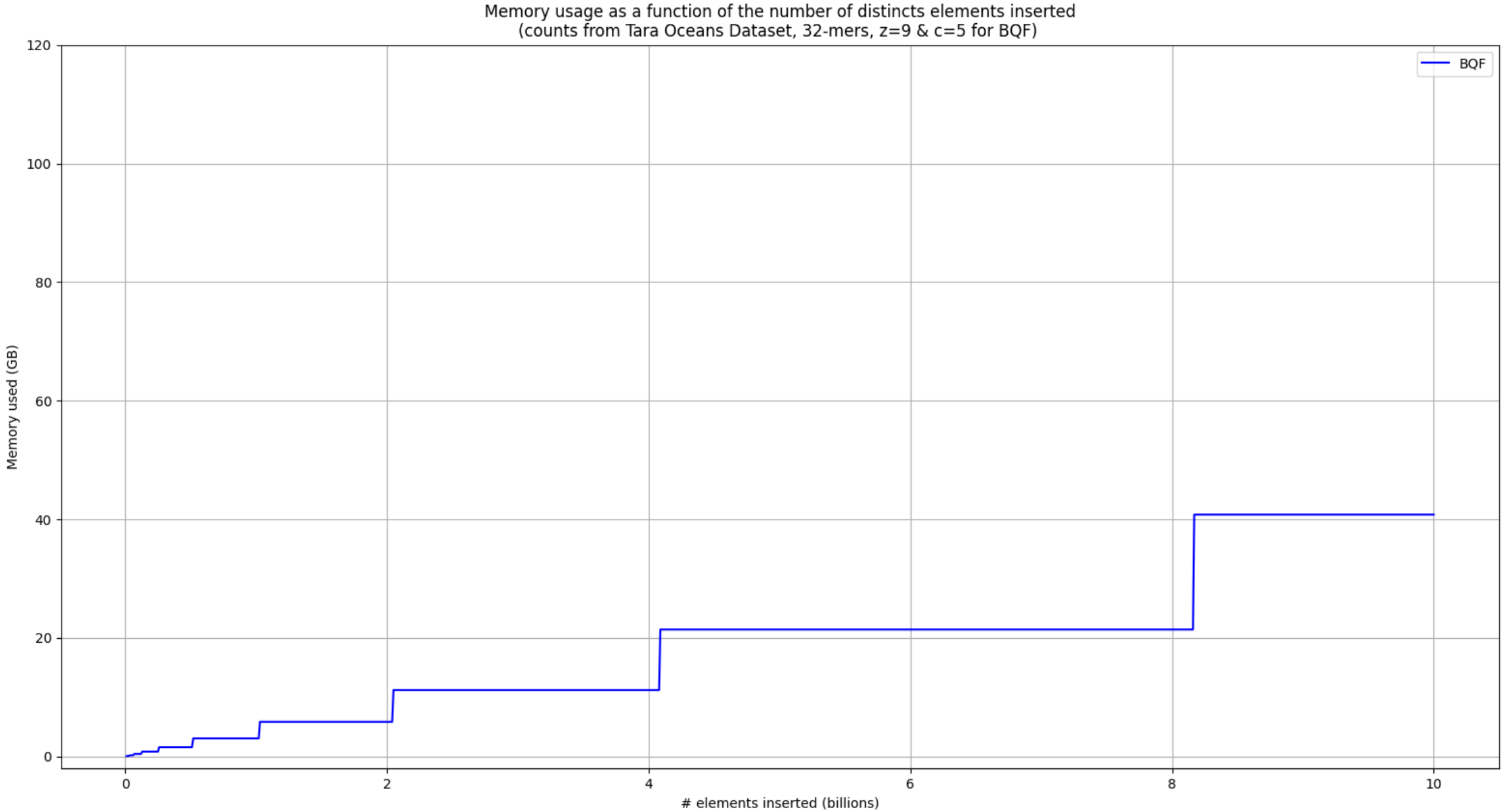
Results

Results from an experiment on metagenomic data (**Tara Oceans Project**)
-> https://www.ebi.ac.uk/ena/browser/view/ERS488262
-> **1.583 billions** of unique **32-mers** to index

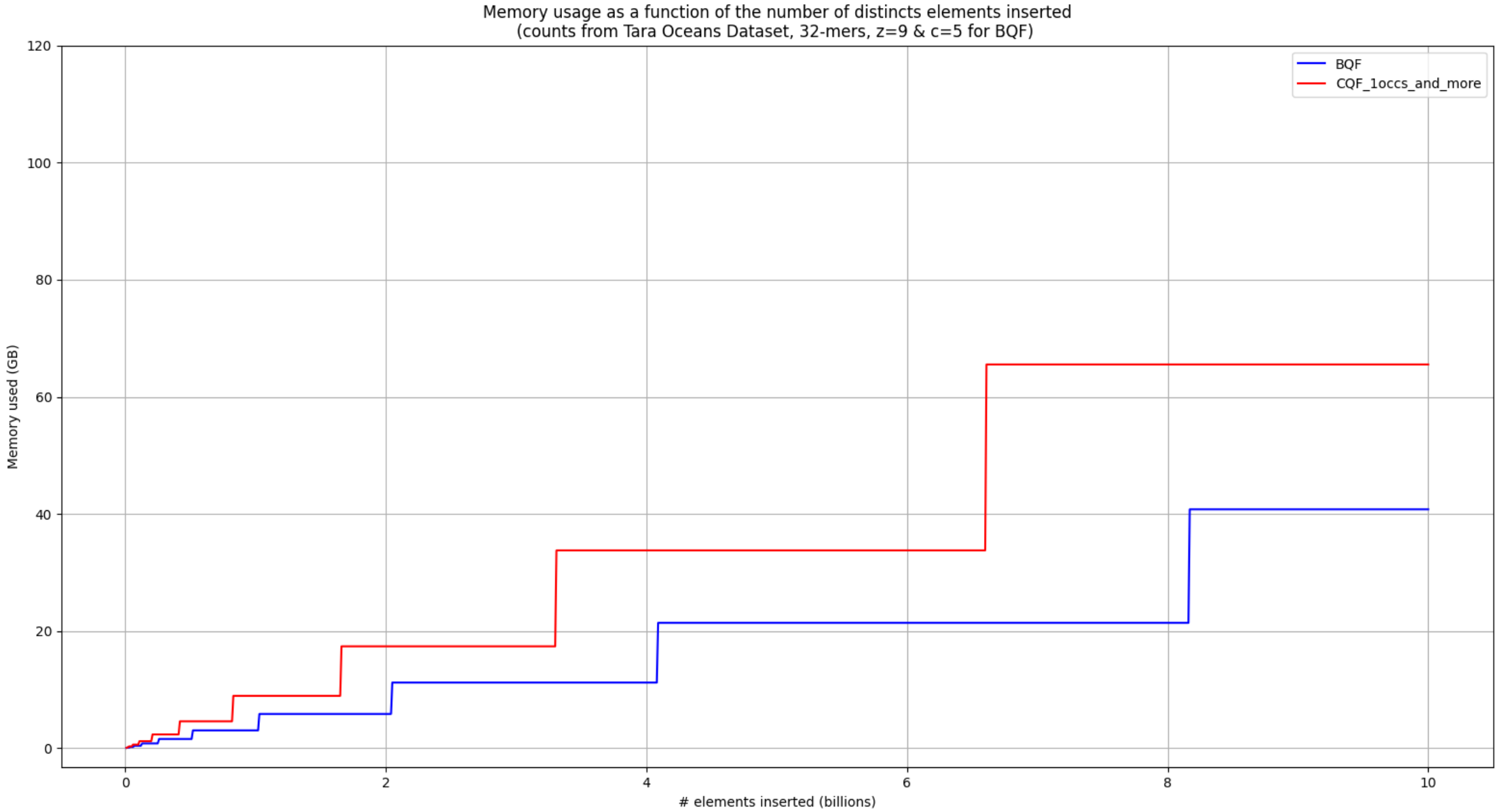
	CQF (q=31)	BQF (q=31, z=9, c=5)
Data Structure Size	**9,43 GB**	**6,17 GB**
Load factor	**90,9 %**	**74,8 %**
False-positive rate	**0**	**$2*10^{-11}$**
Building (insertions)	20 min	23 min
Positive query speed	2.5M kmer/s	4M kmer/s
Negative query speed	3M kmer/s	5.1M kmer/s

Performances measured while querying ~100bp long sequences

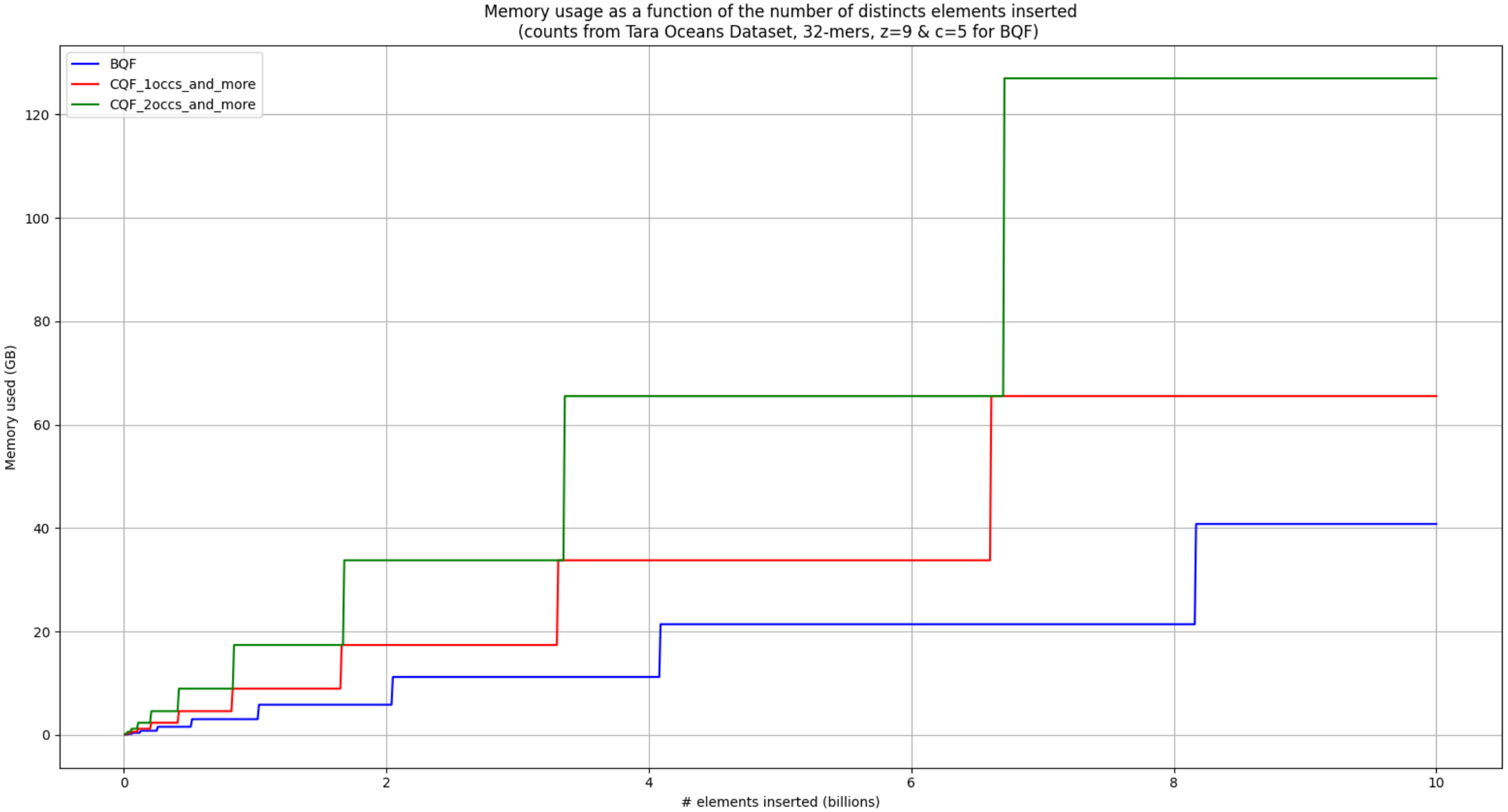
Results


Memory usage as a function of the number of distincts elements inserted
(counts from Tara Oceans Dataset, 32-mers, z=9 & c=5 for BQF)
BQF
Memory used (GB)
elements inserted (billions)
0
20
40
60
80
100
120
0
2
4
6
8
10

Results


Memory usage as a function of the number of distincts elements inserted
(counts from Tara Oceans Dataset, 32-mers, z=9 & c=5 for BQF)
BQF
CQF_1occs_and_more
Memory used (GB)
elements inserted (billions)
0
20
40
60
80
100
120
0
2
4
6
8
10

Results


Memory usage as a function of the number of distincts elements inserted
(counts from Tara Oceans Dataset, 32-mers, z=9 & c=5 for BQF)
BQF
CQF_1occs_and_more
CQF_2occs_and_more
Memory used (GB)
0
20
40
60
80
100
120
0
2
4
6
8
10
elements inserted (billions)

Conclusion

- CQF : High value abundances storage
- BQF :
  - Built-in counters -> less used slots
  - Fimpera -> space gain / slot
  - (+) Overall space gain
  - (-) Construction false positive
  - Project available
    - https://github.com/vicLeva/bqf
    - Usable tool
    - Detailed experiments

Perspectives

1. **BQF** publication
2. Fondation tara océan explorer et partager — Scaling up ?
3. Benefits from using locality preserving hash function ?
4. Indexing proteic datas ?

~60TB

Thank you

Questions ***time***

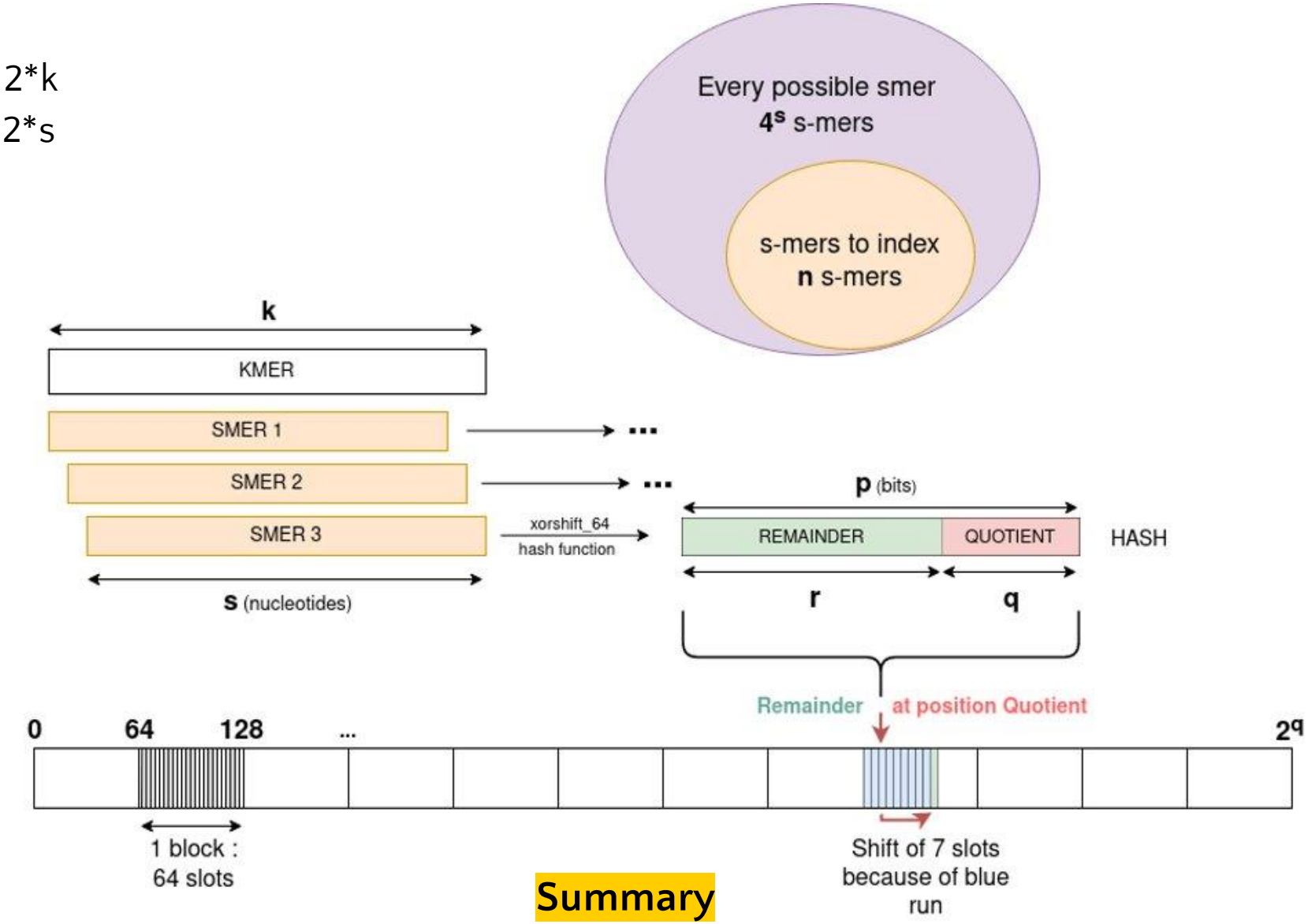
- https://github.com/vicLeva/bqf

  (forked from https://github.com/frankandreace/cqf_implementation, thanks to **Francesco Andreace**)

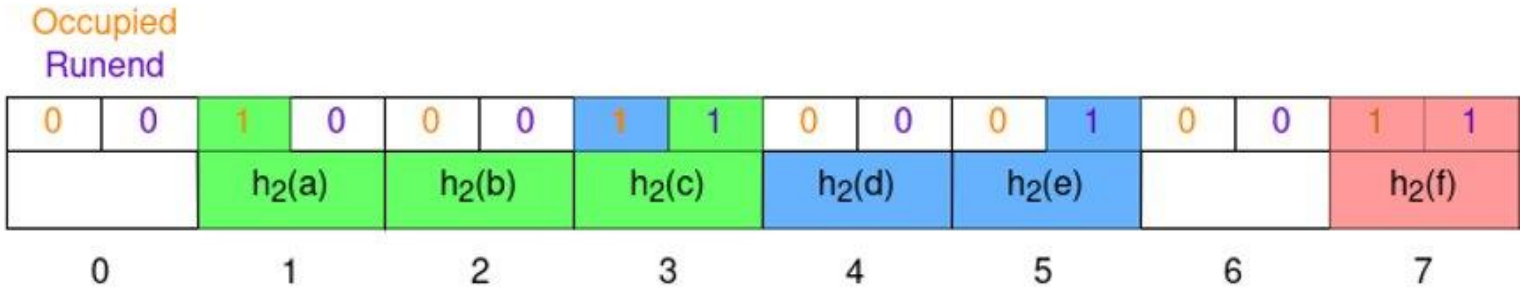
Additional resources

4^k k-mers -> |hash| = 2*k

4^s s-mers -> |hash| = 2*s



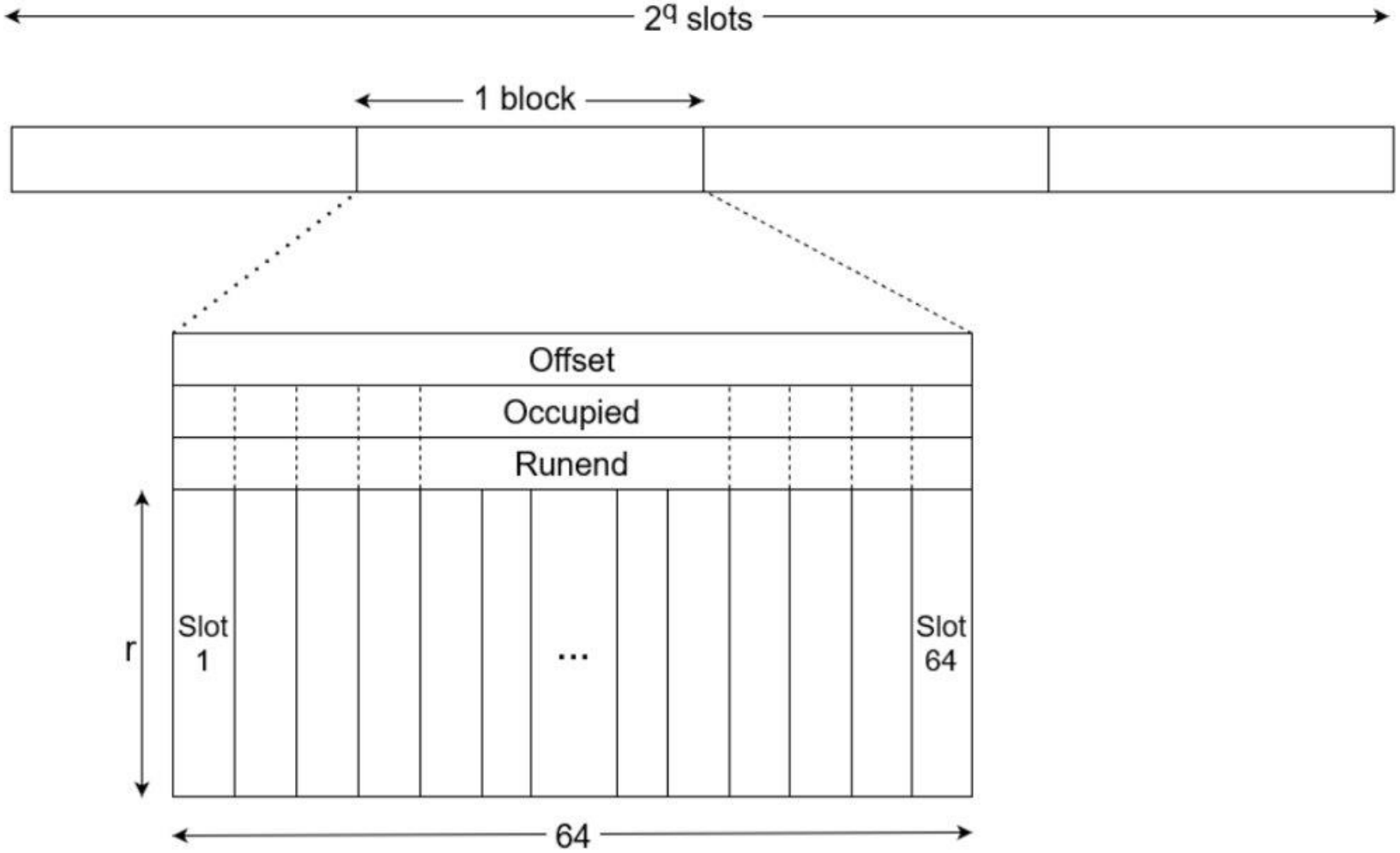
Summary



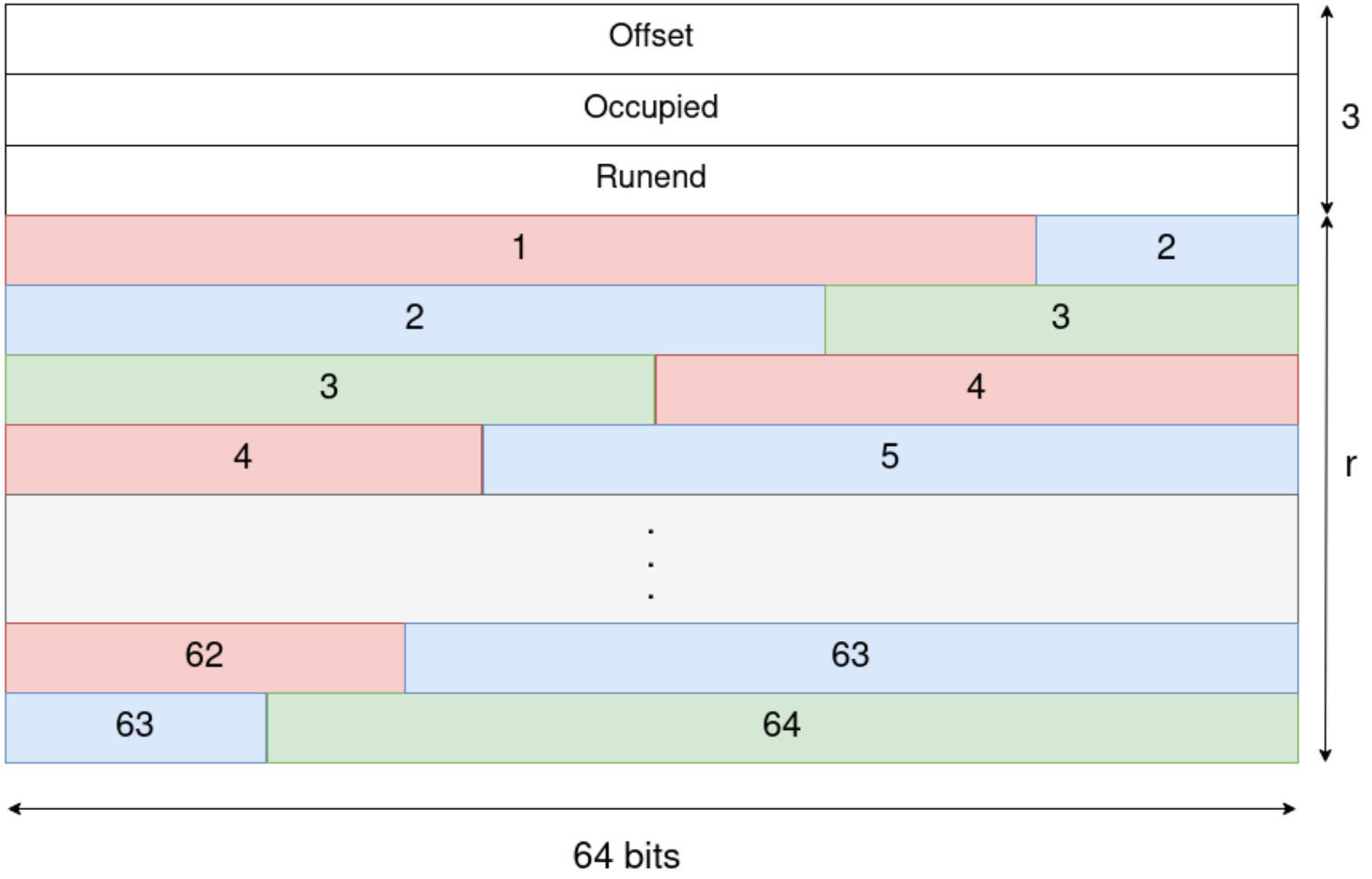
Shifted runs

Paper counting		Backpack counting		
Original	**Metadata**	**Bits addition**	**Bits sacrifice**	**Nucl sacrifice**
Counter size (remainder + count) ----> 1: r , 2:2r , 3+:3r (+r) , 2^r:4r (+r)	1: r+1 , 2+: 2r+2	1+ : r + c	1+ : r	1+ : r
Impact on filter space ----> each time an element has a count > 1: takes other elements' places	adds 2^q bits of metadata each time an element has a count > 1: takes other element's place	adds c x 2^q bits of data	None	None
Dynamicity ----> Yes	Yes	Exact count : yes Otherwise : no	No	~Yes (exact count) (rehash into cut kmer)
New False positive ----> None	None	None	FPrate : 0 -> $(2^s -1) / 2^r$	FPrate : 0 -> $(4^s -1) / 4^k$
Speed perfs ----> Needs special encoding / decoding for each counter	1 extra metadata lookup	Good	Good	Good
Use case ----> Original implementation Good for having exact and important counts dynamic **Better than the other variant if lots of 1&2 occurence(s)**	Good for having exact and important counts dynamic **Better with lots of 3+ occurences**	Flexible, can exact count with few bits or order of magnitudes if necessary Insert everything at init	Ultra space efficient at FP cost Insert everything at init	Ultra space efficient at FP cost dynamicity at even more FP cost

Theory



Overview



In memory