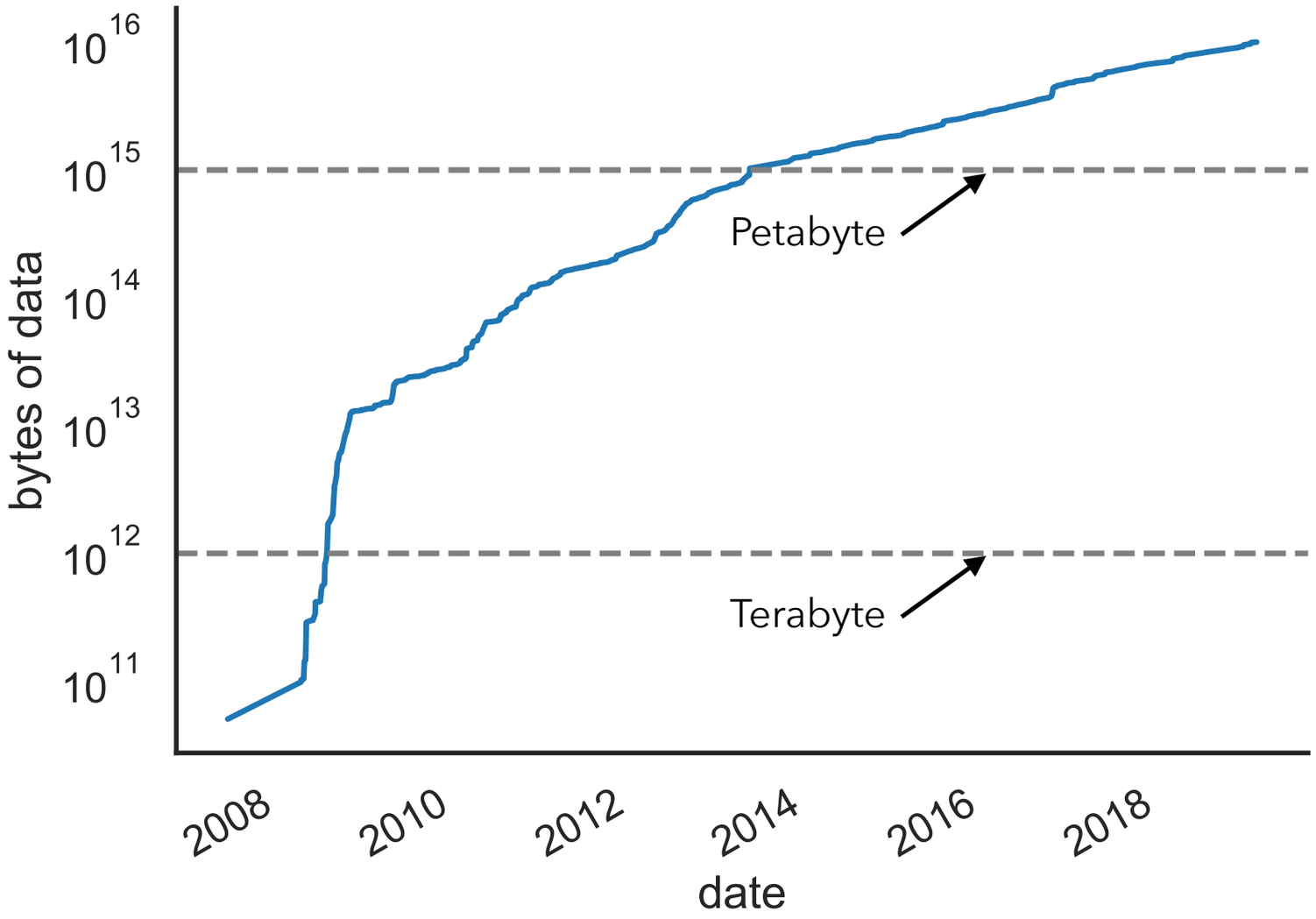
Hashing and AMQs for large-scale sequence search

UNIVERSITY OF MARYLAND

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**

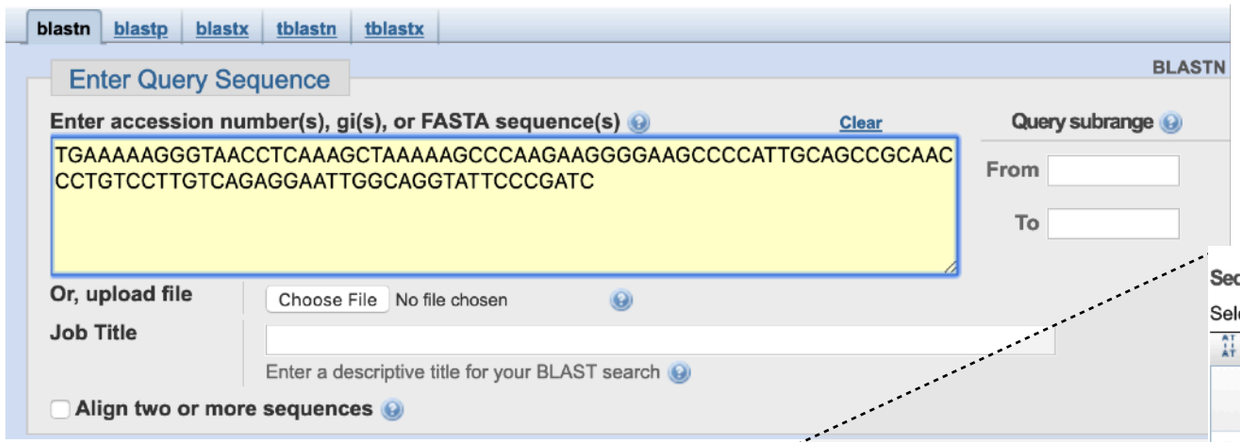
Q: What if I find e.g., a new disease-related gene, and want to see if it appeared in other experiments?

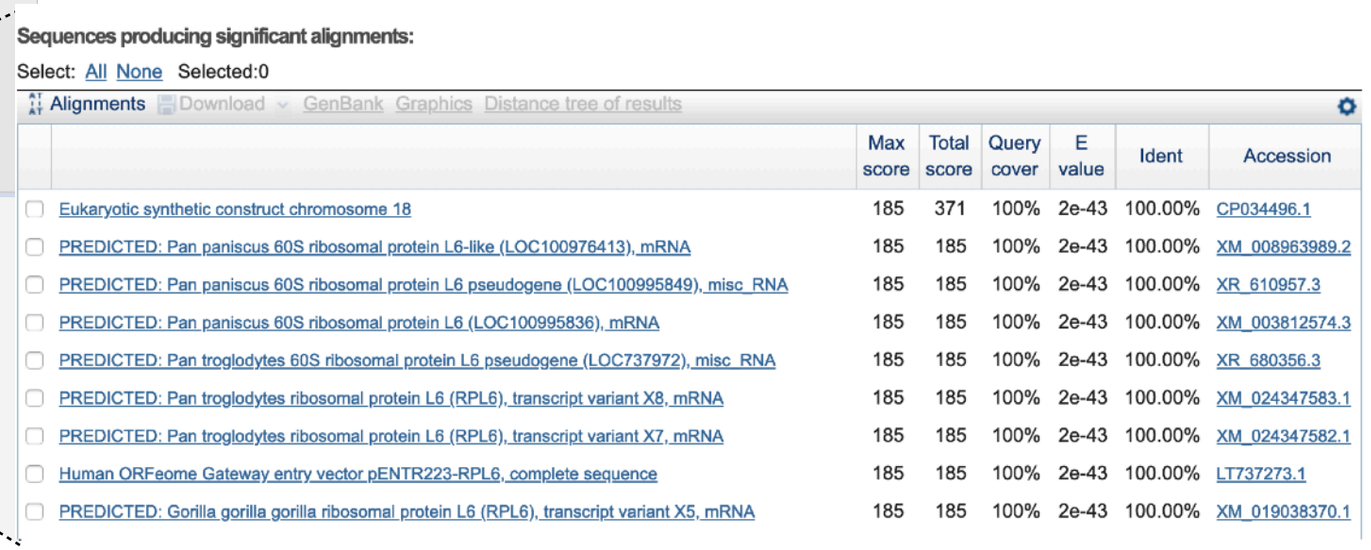
A: (basically) Too bad!

* 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.





	Max score	Total score	Query cover	E value	Ident	Accession
Eukaryotic synthetic construct chromosome 18	185	371	100%	2e-43	100.00%	CP034496.1
PREDICTED: Pan paniscus 60S ribosomal protein L6-like (LOC100976413), mRNA	185	185	100%	2e-43	100.00%	XM_008963989.2
PREDICTED: Pan paniscus 60S ribosomal protein L6 pseudogene (LOC100995849), misc_RNA	185	185	100%	2e-43	100.00%	XR_610957.3
PREDICTED: Pan paniscus 60S ribosomal protein L6 (LOC100995836), mRNA	185	185	100%	2e-43	100.00%	XM_003812574.3
PREDICTED: Pan troglodytes 60S ribosomal protein L6 pseudogene (LOC737972), misc_RNA	185	185	100%	2e-43	100.00%	XR_680356.3
PREDICTED: Pan troglodytes ribosomal protein L6 (RPL6), transcript variant X8, mRNA	185	185	100%	2e-43	100.00%	XM_024347583.1
PREDICTED: Pan troglodytes ribosomal protein L6 (RPL6), transcript variant X7, mRNA	185	185	100%	2e-43	100.00%	XM_024347582.1
Human ORFeome Gateway entry vector pENTR223-RPL6, complete sequence	185	185	100%	2e-43	100.00%	LT737273.1
PREDICTED: Gorilla gorilla gorilla ribosomal protein L6 (RPL6), transcript variant X5, mRNA	185	185	100%	2e-43	100.00%	XM_019038370.1

Essentially, the "Google of genomics":

Basic local alignment search tool

SF Altschul, W Gish, W Miller, EW Myers… - Journal of molecular …, 1990 - Elsevier Paperpile

A new approach to rapid sequence comparison, basic local alignment search tool (BLAST), directly approximates alignments that optimize a measure of local similarity, the maximal segment pair (MSP) score. Recent mathematical results on the stochastic properties of MSP …

☆ Cited by 76248 Related articles Web of Science: 52272 Import into BibTeX

However, even the scale of *reference* databases requires 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

Fast search of thousands of short-read sequencing experiments.

SBT introduced by **Solomon and Kingsford.**

Nature biotechnology. 2016 doi: 10.1038/nbt.3442

Problem:

The vast repository of publicly-available data (e.g., the SRA) is essentially unsearchable by sequence. Current solutions (BLAST, STAR) too slow. What if I find a novel txp and want to search the SRA for it?

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.

Recall the bloom filter

Bloom Filters

Originally designed to answer *probabilistic* membership queries:

Is element e in my set S?

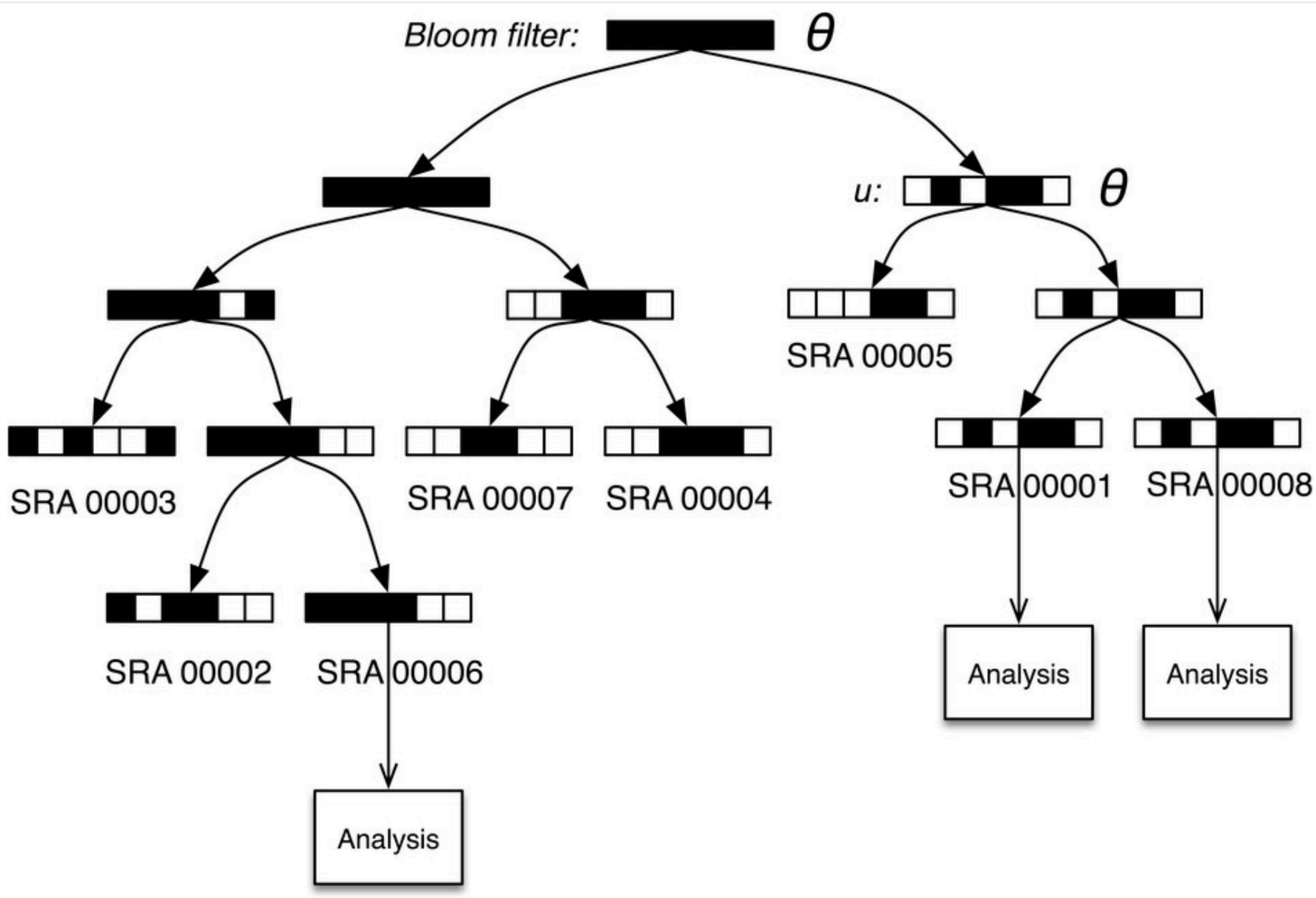
If yes, **always** say yes

If no, say no **with large probability**

False positives can happen; false negatives cannot.

SBT

An SBT is a binary tree of bloom filters, where leaves represent the k-mer set of a single *sample.*



Each node contains a bloom filter that holds the kmers present in the sequencing experiments under it. θ is the fraction of kmers required to be found at each node in order to continue to search its subtree. The SBT returns the experiments that likely contain the query sequence on which further analysis can be performed.

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

SBT Operations

Construction (repeatedly insert samples *s*):

Let b(*s*) be the bloom filter of sample *s*

Use b(*s*) to walk from the root of T to the leaves

For a node *u*:

If *u* has a single child, insert b(s) as the other child

If *u* has 2 children recurse into child with < hamming dist to b(s)

If *u* is a leaf (an experiment), create a parent with filter b(u) U b(s)

SBT Operations

Query (given collection of k-mers K_q, parameter θ):

For a node *u*:

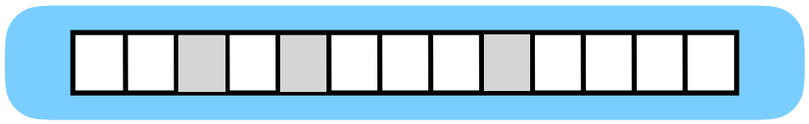
Hash elements of K_q and check if at least θ | K_q| k-mers exist

If not then this sub-tree cannot θ-match our query

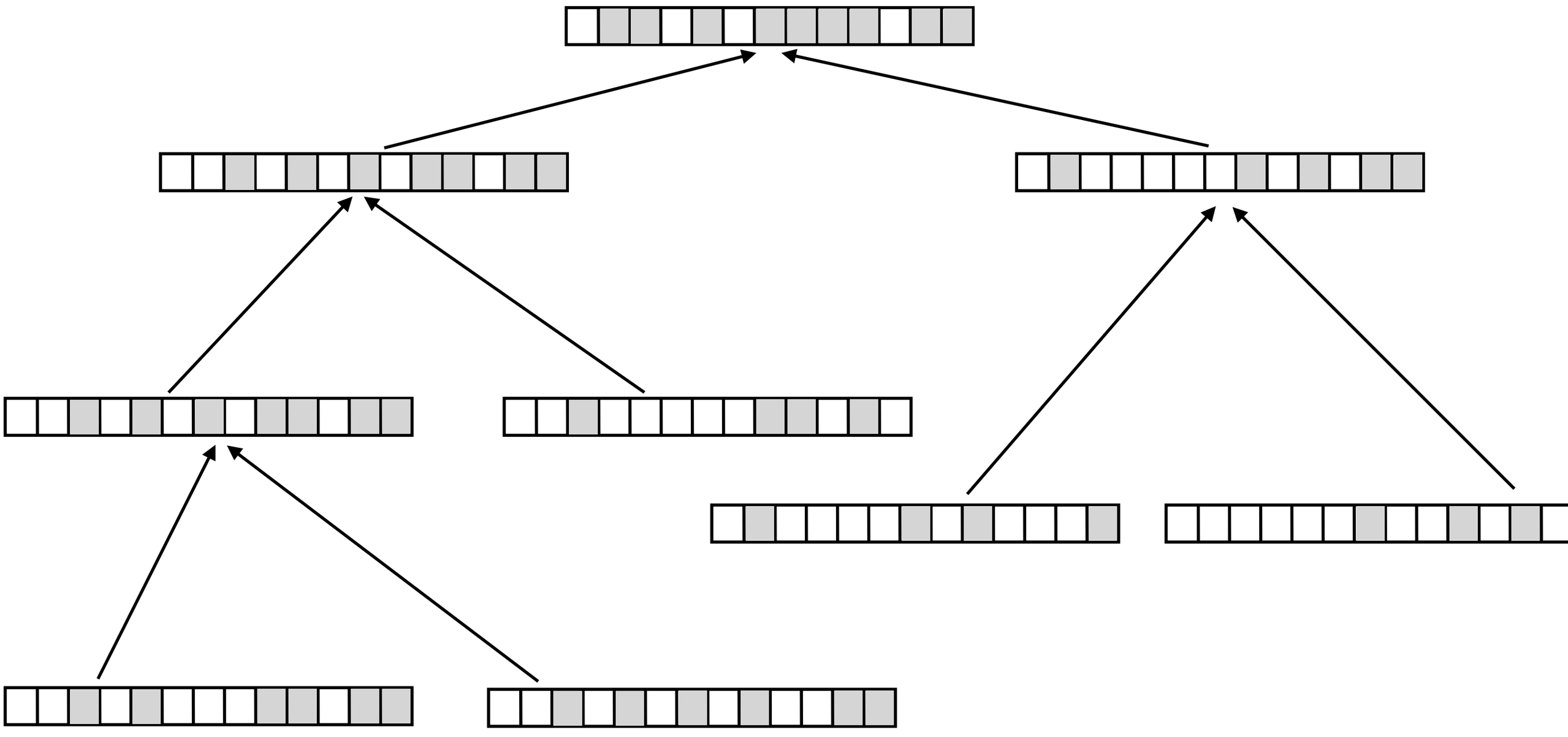
Else continue searching both children recursively

The implementation allows each query k-mer to be given a “weight” or importance.

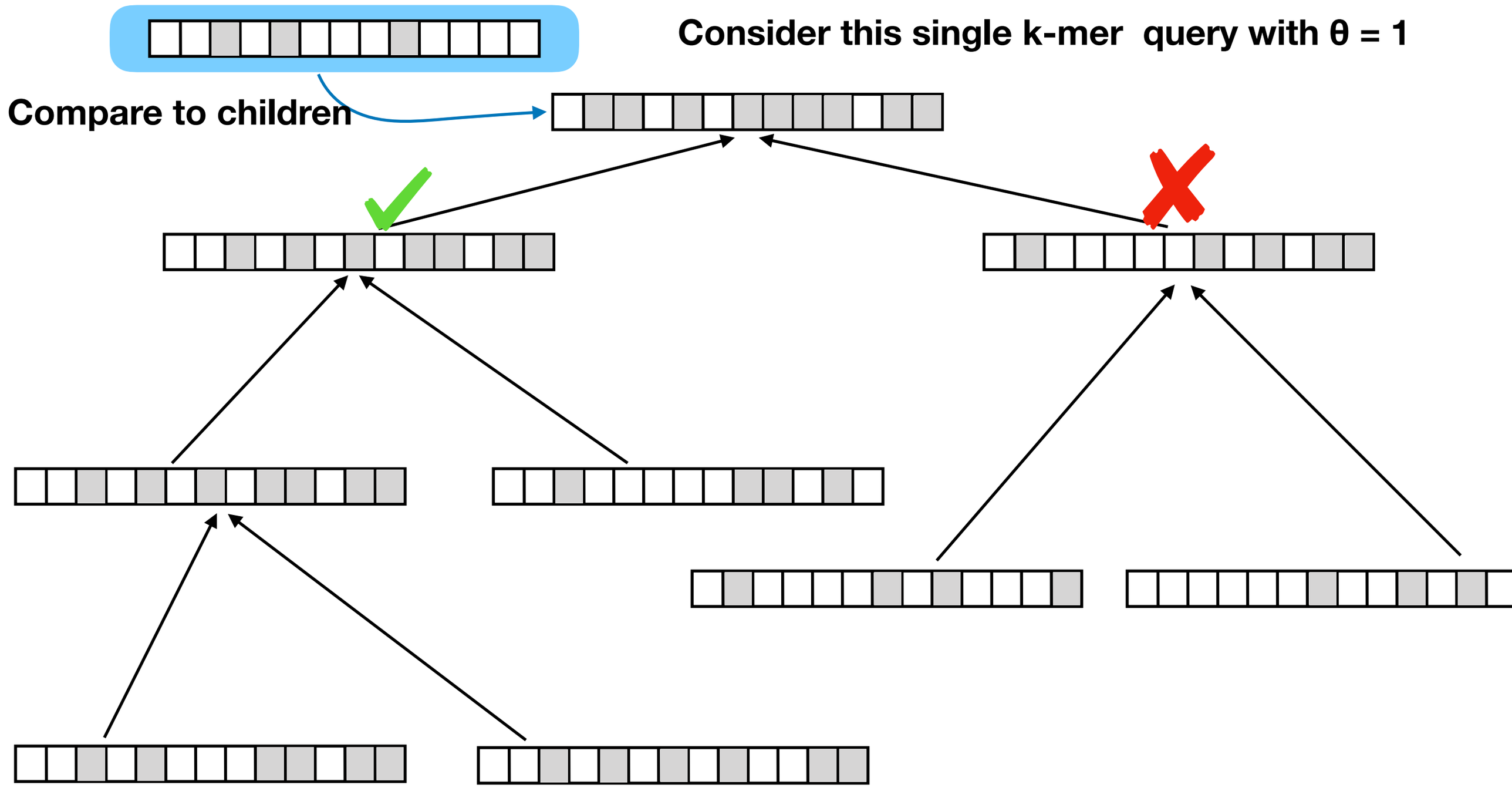
SBT Operations



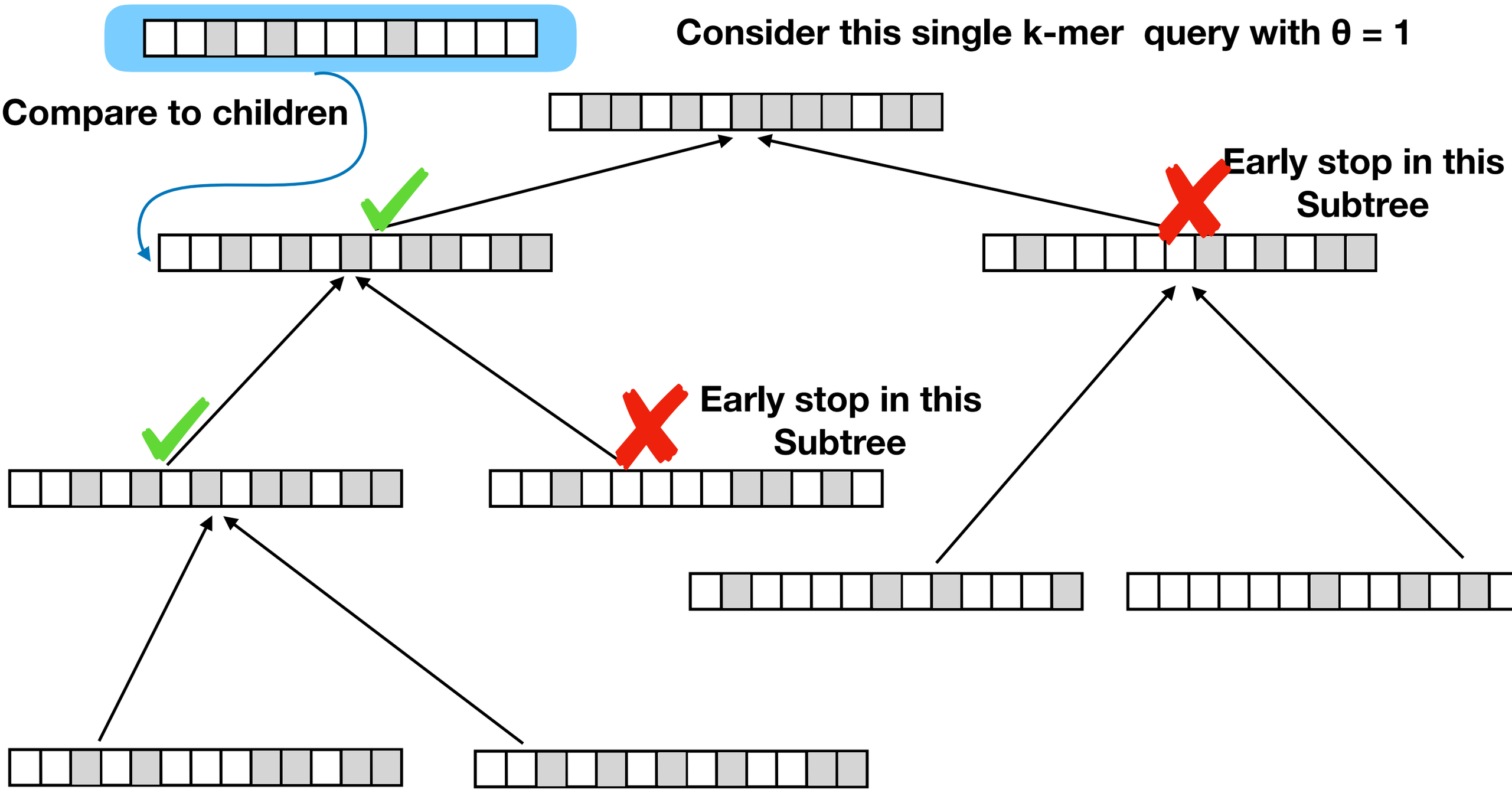
Consider this single k-mer query with θ = 1



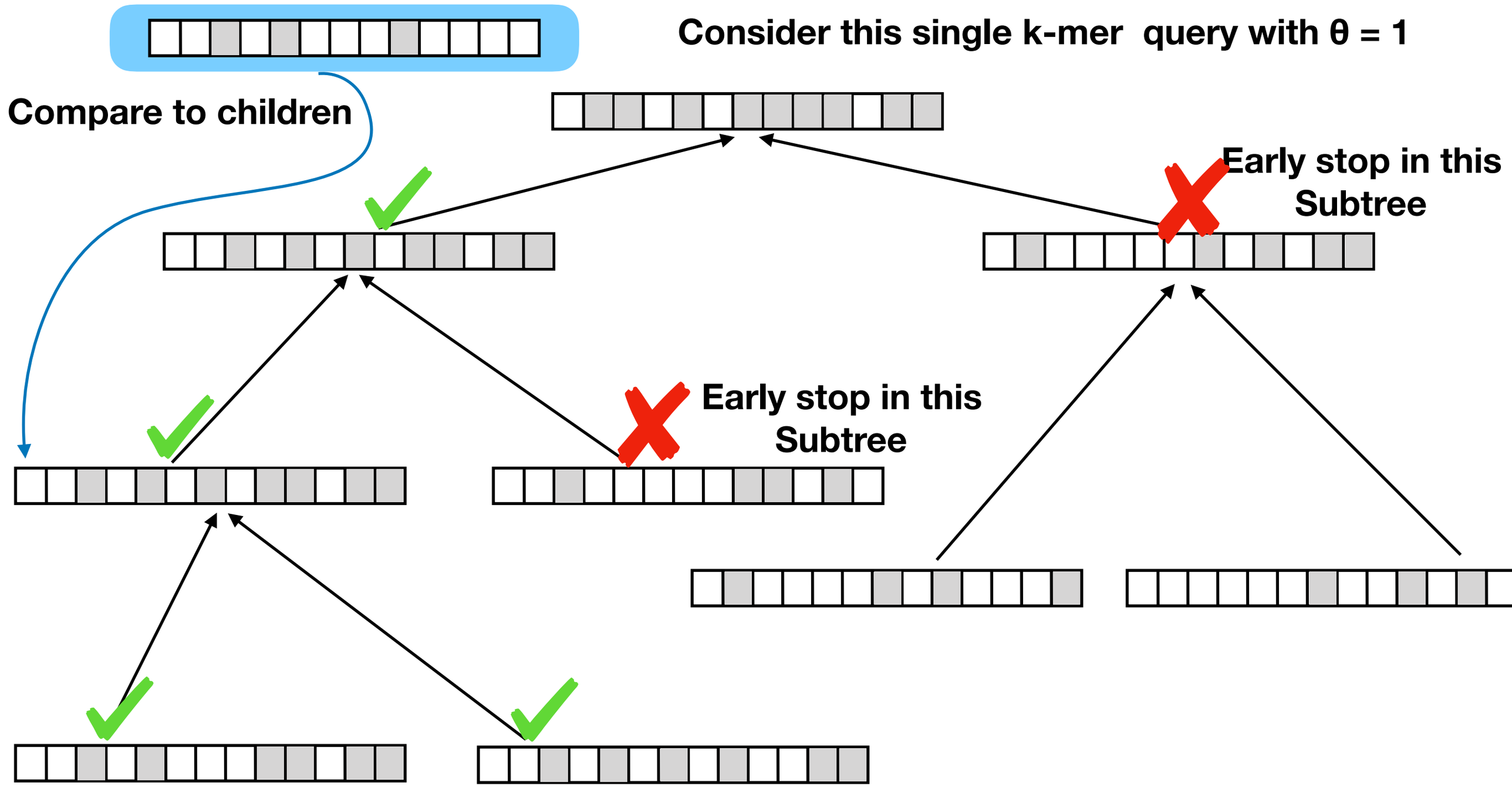
SBT Operations



SBT Operations



SBT Operations



SBT Operations

Query (given collection of k-mers K_q, parameter θ):

Thm 2.
Let q be a query string containing ℓ distinct k-mers. If we treat the k-mers of q as being independent, the probability that > $\lfloor \theta \ell \rfloor$ false-positive k-mers appear in a filter U with FPR ξ is

$$1 - \sum_{i=0}^{\lfloor \theta \ell \rfloor} \binom{\ell}{i} \xi^i (1-\xi)^{\ell - i}$$

Prob of > $\lfloor \theta \ell \rfloor$ false positives is simply the difference from 1

Prob of ≤ $\lfloor \theta \ell \rfloor$ false positives.

Because of assumed indolence of q, these are independent Binomial trials with "success" rate ξ

SBT Tricks

High false positive rate lets filters be small (& use only a single hash)

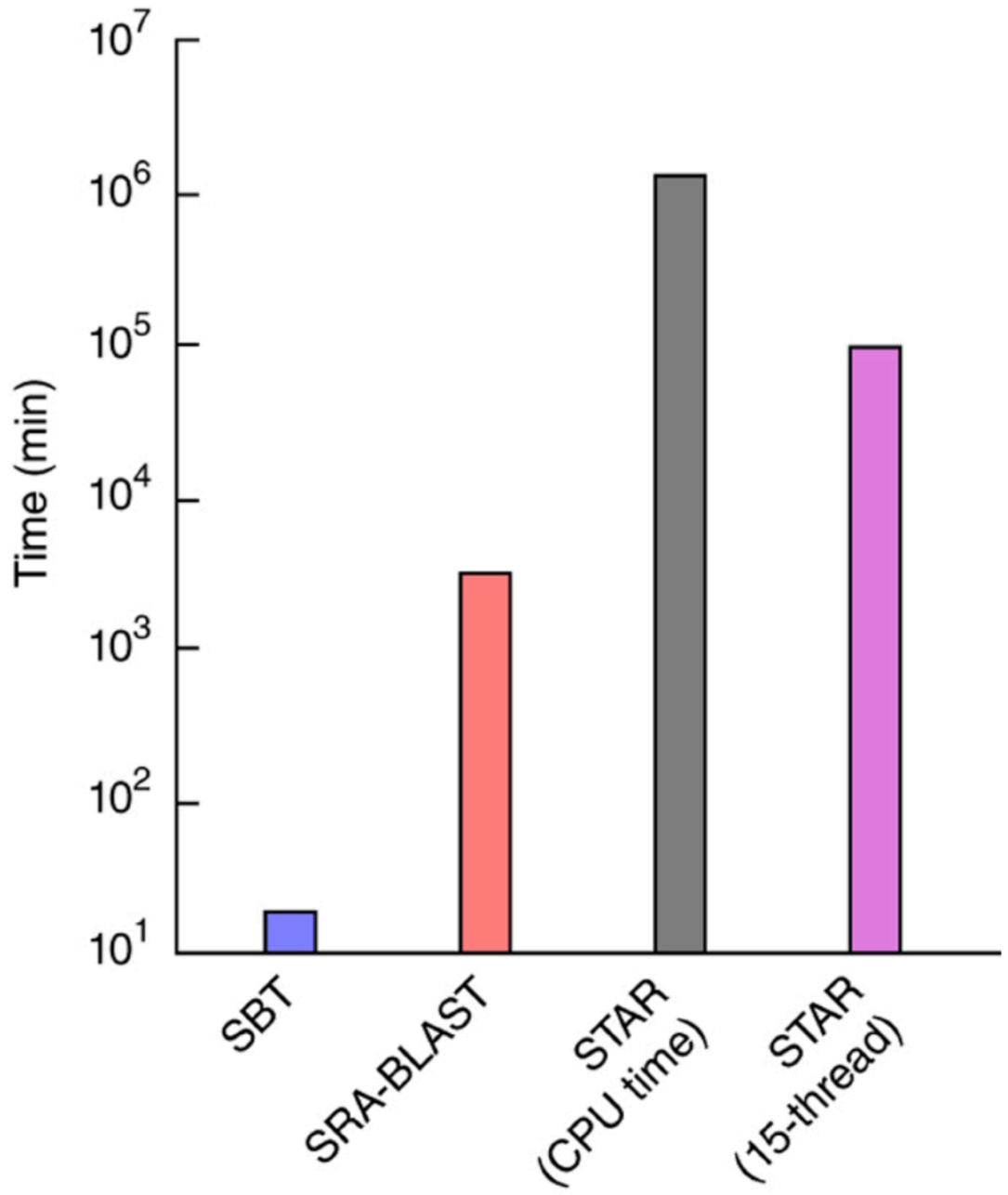
Insert in leaves only k-mers occurring > c times, set

> as follows: $count(s_i) = 1$ if s_i is 300 MB or less, $count(s_i) = 3$ for files of size 300–500 MB, $count(s_i) = 10$ for files of size 500 MB–1 GB, $count(s_i) = 20$ for files between 1 GB and 3 GB, and $count(s_i) = 50$ for files > 3 GB or larger FASTA files.

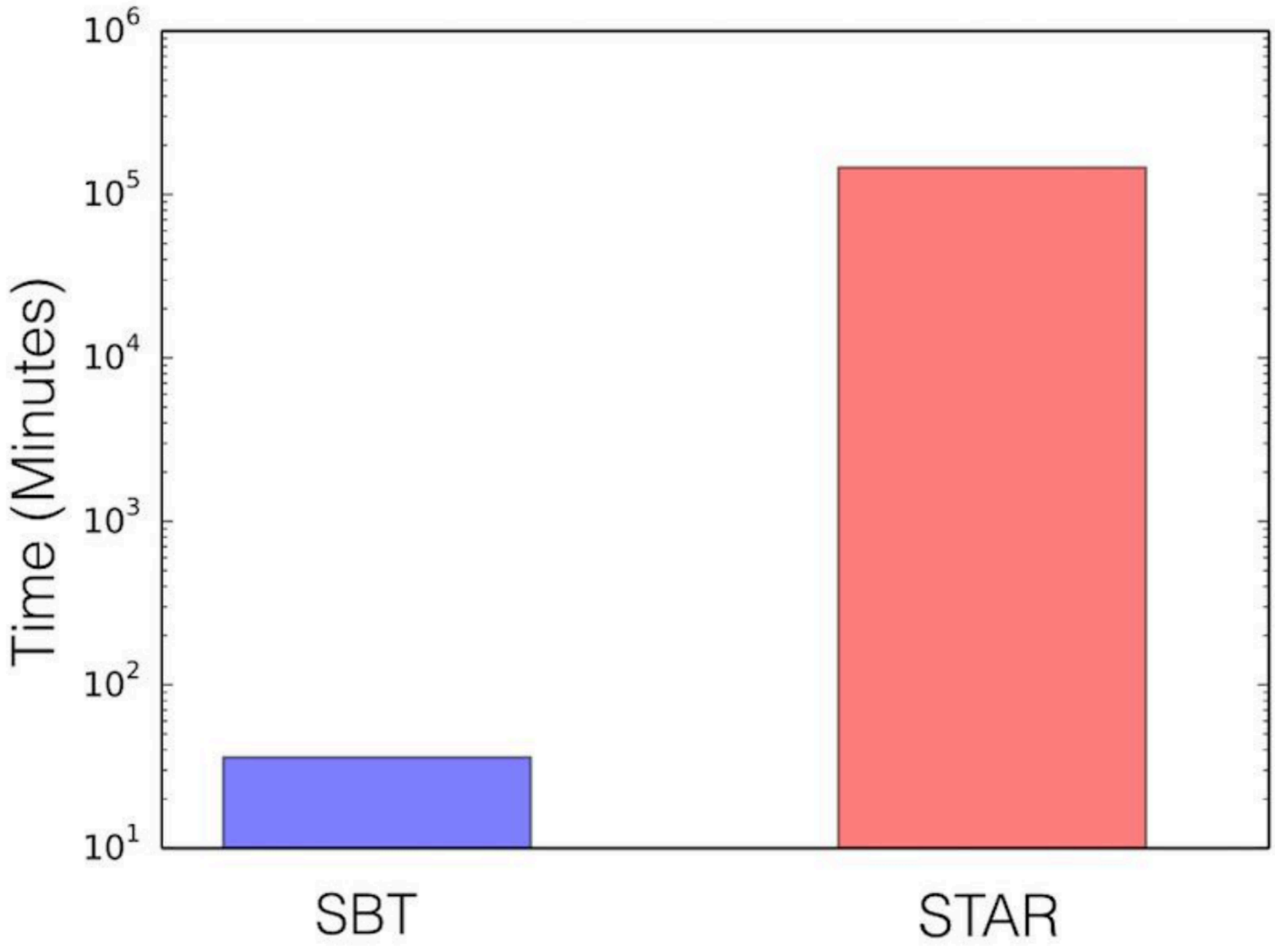
Store Bloom filter as RRR-compressed bit vectors. Greatly reduces storage space. Individual bits can be accessed *without* decompression in O(log m) time.

SBT Speed

Average search time for a single transcript over 2,652 RNA-seq experiments in the SRA for human blood, breast and brain tissues



SBT Speed

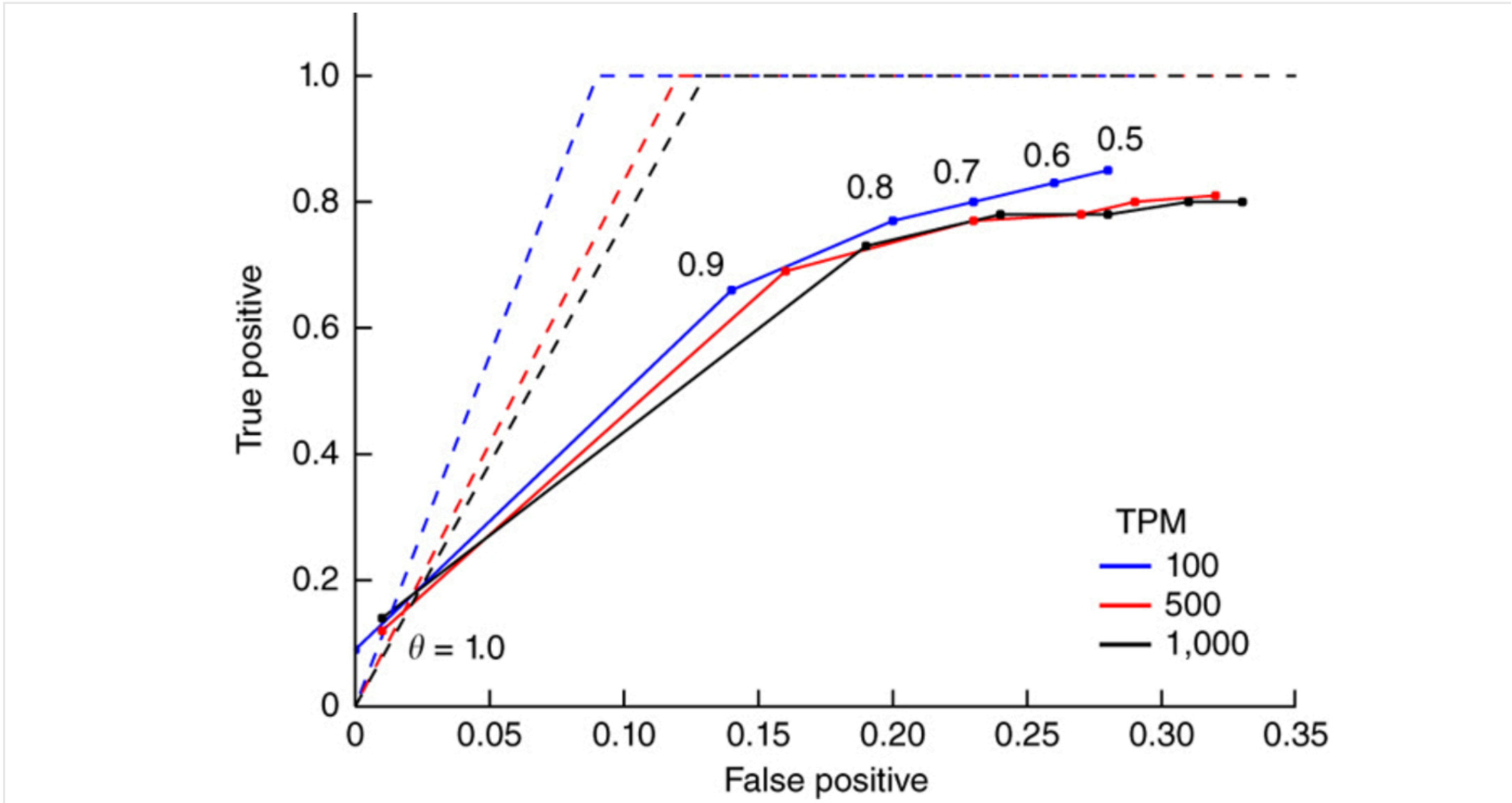


Supplementary Figure 2

Comparison with STAR on batched queries.

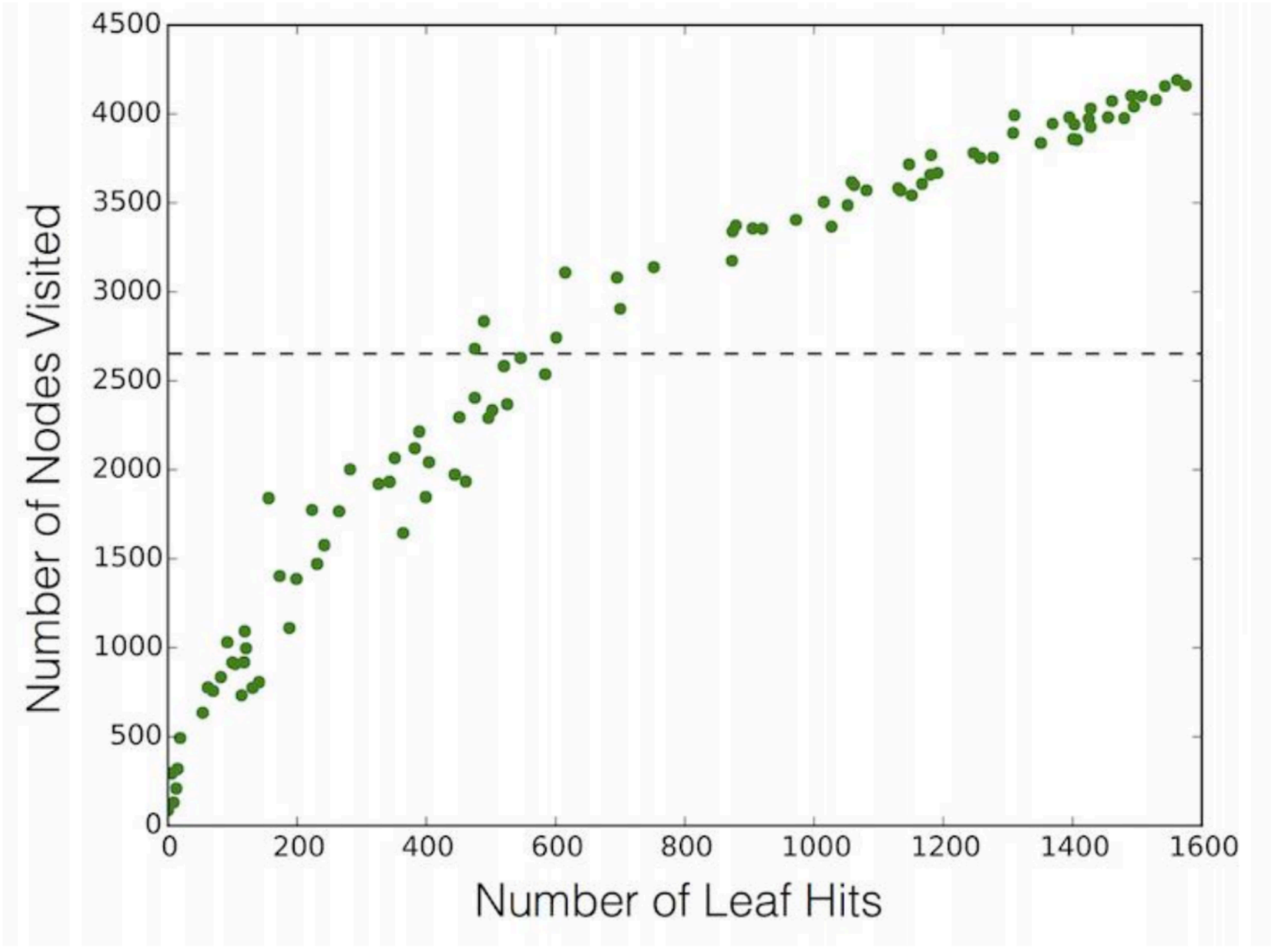
STAR was run using an index built from 100 batch-queries and a size 11 pre-index string. Both SBT and STAR were run using one thread and SBT was limited to a single filter in RAM. SBT is an estimated 4056 times faster than STAR under these conditions. STAR times are estimated from extrapolating from querying 100 random SRR files.

SBT Accuracy



Solid lines represent mean true-positive and false-positive rates, dashed lines represent the median rates on the same experiments. Relaxing θ leads to a higher sensitivity at the cost of specificity. In more than half of all queries, 100% of true-positive hits can be found with θ as high as 0.9.

SBT Efficiency

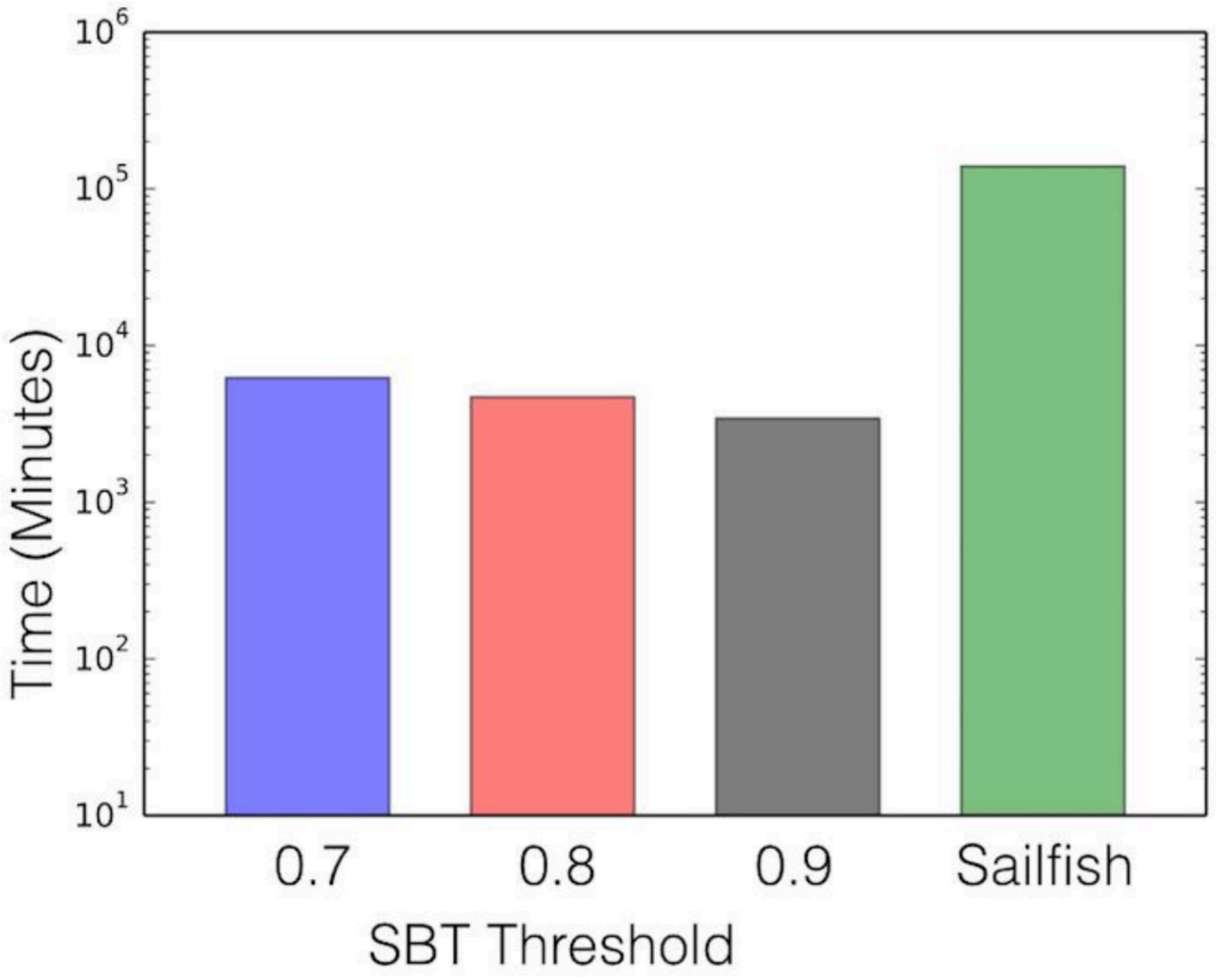


Supplementary Figure 5

Total number of Sequence Bloom Tree nodes visited as a function of the number of leaf hits when querying 100 random human transcripts in the Low query set.

Number of nodes includes both internal and leaf nodes of the SBT. Each point represents a single query. When a query is found in many of the leaves, the query must also visit a nearly equal number of internal tree nodes, and so the tree structure would not provide any benefit over merely searching all the leaf filters directly. On the other hand, when the query is found in only a few leaves, the total number of nodes visited can be significantly smaller than the number of leaves. For the SBT built here, we find that for queries that are found in 600 or fewer leaves, the tree structure and internal nodes result in an improvement of overall efficiency by visiting fewer than 2652 nodes. A naive approach that did not use the tree would require querying 2652 leaf filters for all queries (denoted by dashed line). Approximately half of the randomly selected queries known to be expressed in the included experiments fall below this threshold.

SBT Efficiency



Supplementary Figure 8

Time for querying all known human transcripts.

Total times (single-threaded) for querying all 214,293 human transcripts (in batch mode) against all publicly available blood, breast, and brain RNA-seq experiments in the SRA for θ = 0.7, 0.8, 0.9 as well as the extrapolated time to run Sailfish on the full dataset. Sailfish is significantly faster than nearly all other algorithms for RNA-seq quantification.

Two improved SBT-related papers (RECOMB 2017)

Improved Search of Large Transcriptomic Sequencing Databases Using Split Sequence Bloom Trees

Brad Solomon[1] and Carl Kingsford*[1]

AllSome Sequence Bloom Trees

Chen Sun*[1], Robert S. Harris*[2] Rayan Chikhi[3], and Paul Medvedev†[1,4,5]

Both share a core idea

All Some SBT:

$$B_{all}(u) = B_{\cap}(u) \setminus B_{\cap}(parent(u))$$
$$B_{some}(u) = B_{\cup}(u) \setminus B_{\cap}(u)$$

present in all leaves below u, but not in u's parent

present in some (but not) all leaves below u

Split sequence bloom tree (SSBT):

Store **2** filters at each node, r_{sim} and r_{rem}

$r_{sim} = \bigcap_{i=0}^{n} b_i$ present in all leaves below r

$r_{rem} = \bigcup_{i=0}^{n} (b_i - r_{sim})$ present in some (but not) all leaves below r

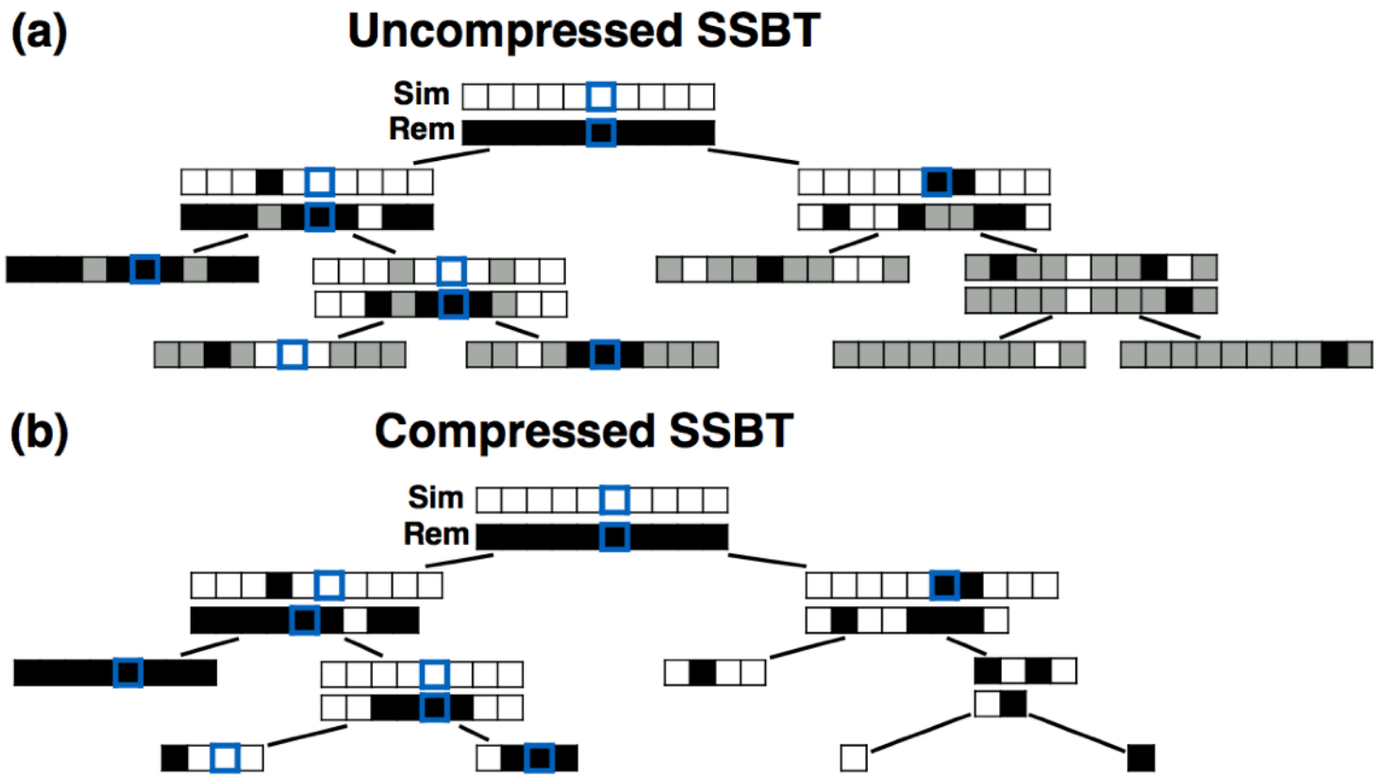
Both share a core idea

This allows an immediate optimization

if r_{sim} or B_{all} match the query, we can add all leaves below r/u without explicitly continuing the search

The details differ

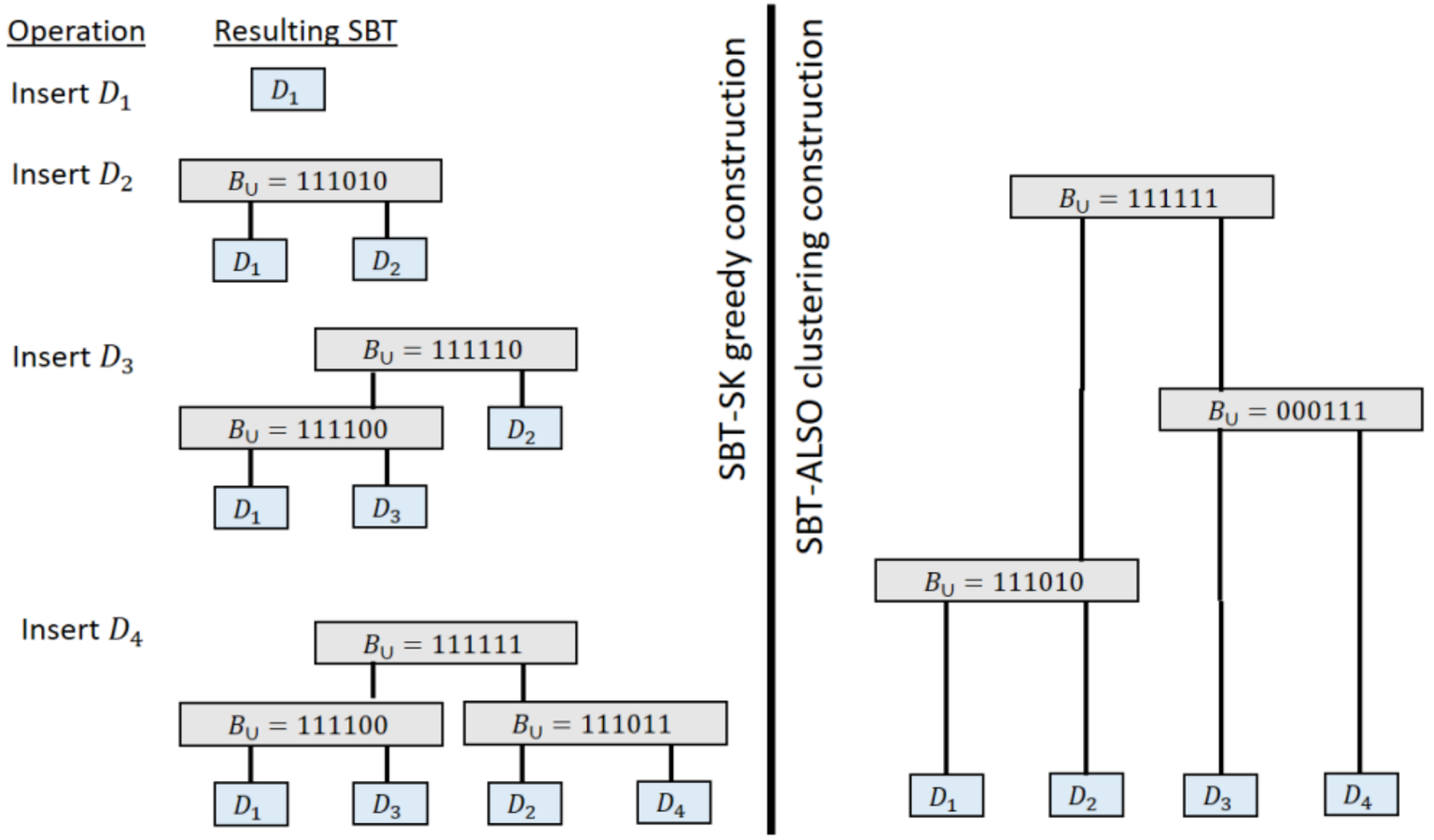
SSBT *explicitly* removed redundant elements



Both share a core idea

The details differ

SBT-AllSome don’t explicitly remove these bits, but they optimize tree construction to put similar filters together (agglomerative clustering)



SSBTs take longer to build than the original but require considerably less memory to store.

Data Index	BFT	SBT	SSBT
Build Time (Min)	195	6	19
Compression Time (Min)	-	6.5	17
Total Time (Min)	**195**	**12.5**	**36**

Table 4: Build and compression times for SBT, SSBT, and BFT constructed from a 50 experiment set. As SBT and SSBT were designed to be queried from a compressed state, we compare the time to build and compress against BFT's time to build.

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.

Data Index	BFT	SBT	SSBT
Build Peak RAM (GB)	23	21.5	15.6
Compress Peak RAM (GB)	-	24.2	16.2
Uncompressed Size (GB)	**9.2**	24	35
Compressed Size (GB)	-	**3.9**	**0.94**

Table 3: Build and compression peak RAM loads and on-disk storage costs for SBT, SSBT, and BFT constructed from a 50 experiment set. BFT does not have a built-in compression tool and cannot be queried when compressed. For these reasons, the uncompressed BFT is compared against the compressed SBT/SSBT.

SSBTs are also faster to query than SBTs

Index	**TPM ≥100**	**TPM ≥500**	**TPM ≥1000**
BFT	75 Sec (11.8 GB)	75 Sec (11.8 GB)	75 Sec (11.8 GB)
SBT	19 Sec (2.9 GB)	21 Sec (3.1 GB)	22 Sec (3.2 GB)
SSBT	5.8 Sec (0.64 GB)	6.2 Sec (0.65 GB)	6.3 Sec (0.66 GB)

Table 5: Comparison in query timing (and average peak memory) between SBT, SSBT, and BFT indices for 50 experiments.

Index	**TPM ≥100**	**TPM ≥500**	**TPM ≥1000**
SBT	19.7 Min	20.7 Min	20 Min
SSBT	3.7 Min	3.8 Min	3.6 Min

Table 6: Comparison in query timing between SBT and SSBT for 2652 experiments.

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.

AllSome SBTs are faster to construct than SBT (called SBT-SK here), but not much smaller.

	SBT-SK	SBT-ALSO
construction of tree topology (i.e. clustering)	N/A	27m
construction of internal nodes	56h 54m	26h 3m
temporary disk space	1,235 GB	2,469 GB
final disk space	200 GB	177 GB

Table 1. Construction time and space. Times shown are wall-clock times. A single thread was used. Note the SBT-SK tree that was constructed for the purposes of this Table differs from the tree used in [36] and in our other experiments because the insertion order during construction was not the same as in [36] (because it was not described there).

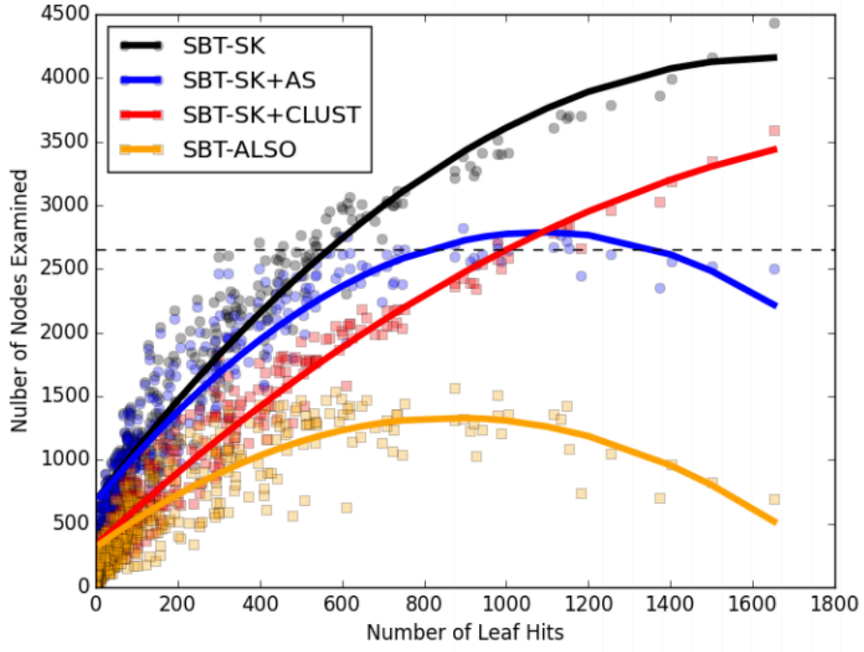


Fig. 3. Number of nodes examined per query for SBT-SK, SBT-ALSO, as well two intermediate SBTs. A set of 1,000 transcripts were chosen at random from Gencode set, and each one queried against the four different trees. A dot represents a query and shows the number of matches in the database (x-axis) compared to the number of nodes that had to be loaded from disk and examined during the search (y-axis). For each tree (color), we interpolated a curve to show the pattern. The dashed horizontal line represents the hypothetical algorithm of simply checking if the query θ-matches against each of the database entries, one-by-one. For θ, we used the default value in the SBT software ($\theta = 0.9$).

They examine fewer nodes than the original SBT too

Which makes them faster to query than the original SBT as well.

	SBT-SK	SBT-SK+CLUST	SBT-ALSO
1 query	1m 11s / 301 MB	56s / 299 MB	34s / 301 MB
10 queries	4m 4s / 305 MB	3m 17s / 304 MB	2m 4s / 313 MB
100 queries	7m 44s / 315 MB	6m 31s / 317 MB	4m 44s / 353 MB
1,000 queries	25m 31s / 420 MB	17m 22s / 418 MB	8m 23s / 639 MB
198,074 queries	3081m 42s / 22 GB	-	462m 39s / 63 GB

Table 2. Query wall-clock run times and maximum memory usage, for batches of different sizes. For the batch of 1,000 queries, we used the same 1,000 queries as in Figure 3. For the batch of 100 queries, we generated three replicate sets, where each set contains 100 randomly sampled transcripts without replacement from the 1,000 queries set. For the batch of 10 queries, we generated 10 replicate sets by partitioning one of the 100 query sets into 10 sets of 10 queries. For the batch of 1 query, we generated 50 replicate sets by sampling 50 random queries from Gencode set. The shown running times are the averages of these replicates. A dash indicates we did not run the experiment. For θ, we used the default value in the SBT software ($\theta = 0.9$).

	SBT-SK	SBT-ALSO		
	regular alg	regular alg	large exact alg	large heuristic alg
query time	1397m 18s	195m 33s	10m 35s	8m 32s
query memory	2.3 GB	4.7 GB	1.3 GB	1.2 GB

Table 3. Performance of different trees and query algorithms on a large query. We show the performance of SBT-SK and three query algorithms using SBT-ALSO compressed with ROAR: the regular algorithm, the large exact algorithm, and the large heuristic algorithm. We show the wall-clock run time and maximum RAM usage. We used $\theta = 0.8$ for this experiment. The ROAR compressed tree was 190 GB (7.3% larger than the RRR tree).

BIGSI

Article | Published: 04 February 2019

Ultrafast search of all deposited bacterial and viral genomic data

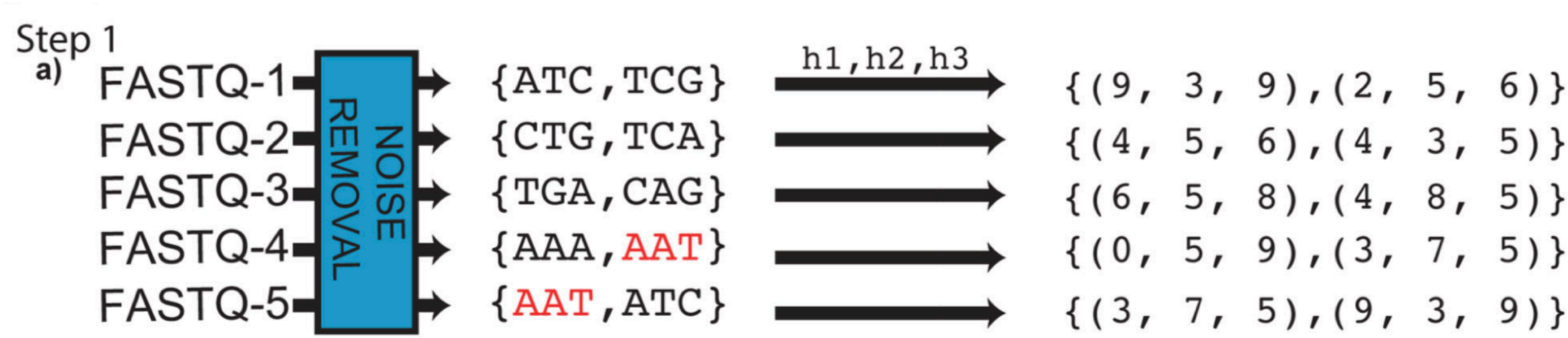
Phelim Bradley, Henk C. den Bakker, Eduardo P. C. Rocha, Gil McVean & Zamin Iqbal

Nature Biotechnology **37**, 152–159 (2019) | Download Citation

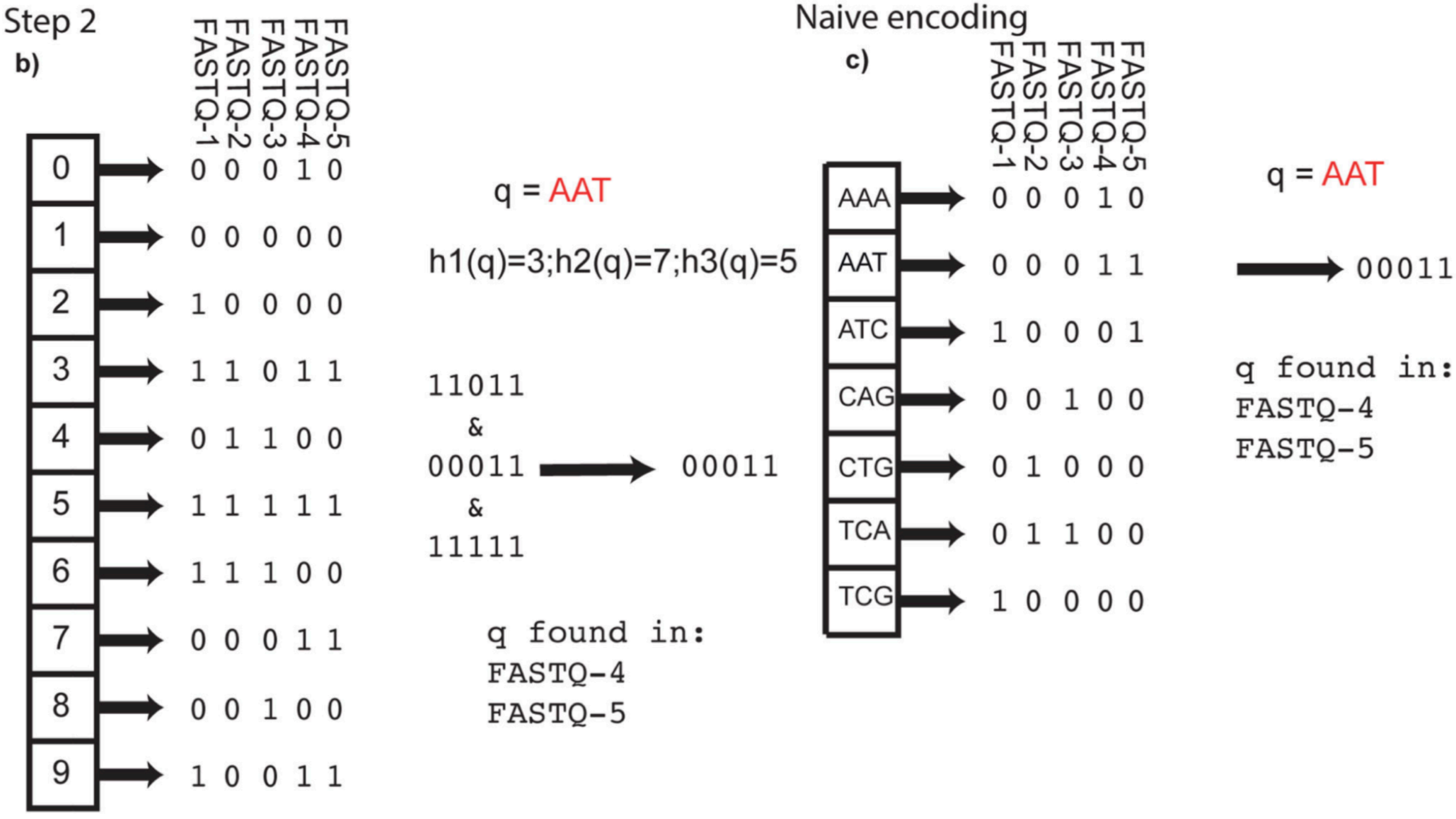
BIGSI

Bitsliced Genomic Signature Index (BIGSI)

Conceptually, a *matrix* of Bloom filters where each column is a Bloom filter.

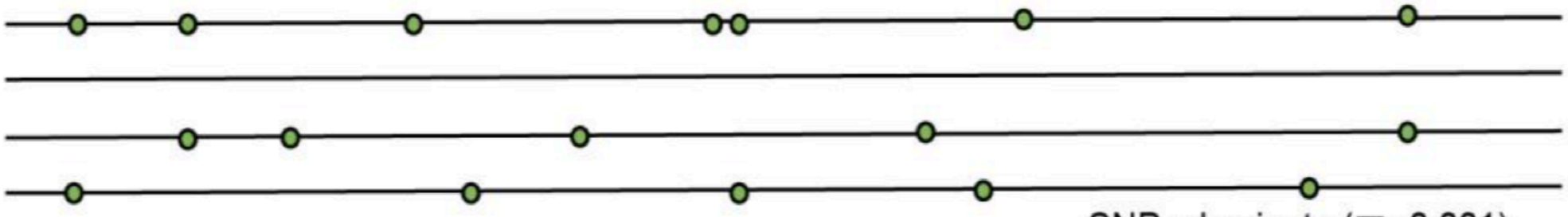


BIGSI


Step 2
b)
FASTQ-1
FASTQ-2
FASTQ-3
FASTQ-4
FASTQ-5
0 0 0 1 0
0 0 0 0 0
1 0 0 0 0
1 1 0 1 1
0 1 1 0 0
1 1 1 1 1
1 1 1 0 0
0 0 0 1 1
0 0 1 0 0
1 0 0 1 1
q = AAT
h1(q)=3;h2(q)=7;h3(q)=5
11011
&
00011
&
11111
00011
q found in:
FASTQ-4
FASTQ-5
Naive encoding
c)
AAA
AAT
ATC
CAG
CTG
TCA
TCG
0 0 0 1 0
0 0 0 1 1
1 0 0 0 1
0 0 1 0 0
0 1 0 0 0
0 1 1 0 0
1 0 0 0 0
q = AAT
00011
q found in:
FASTQ-4
FASTQ-5

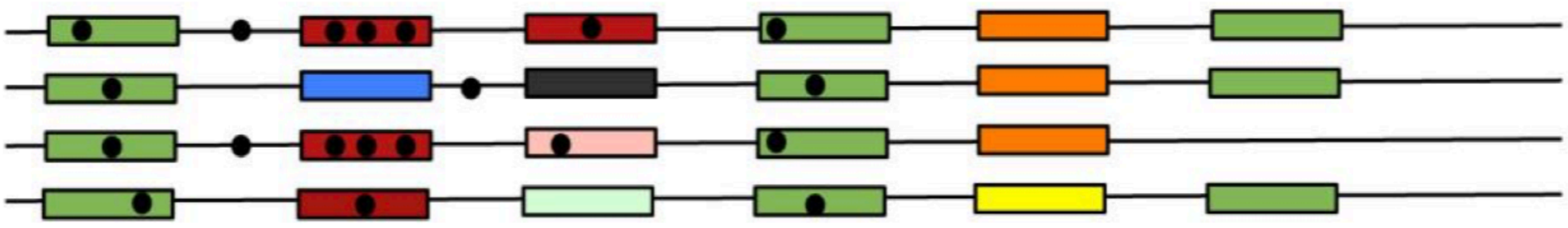
Why a different design?

4 human genomes



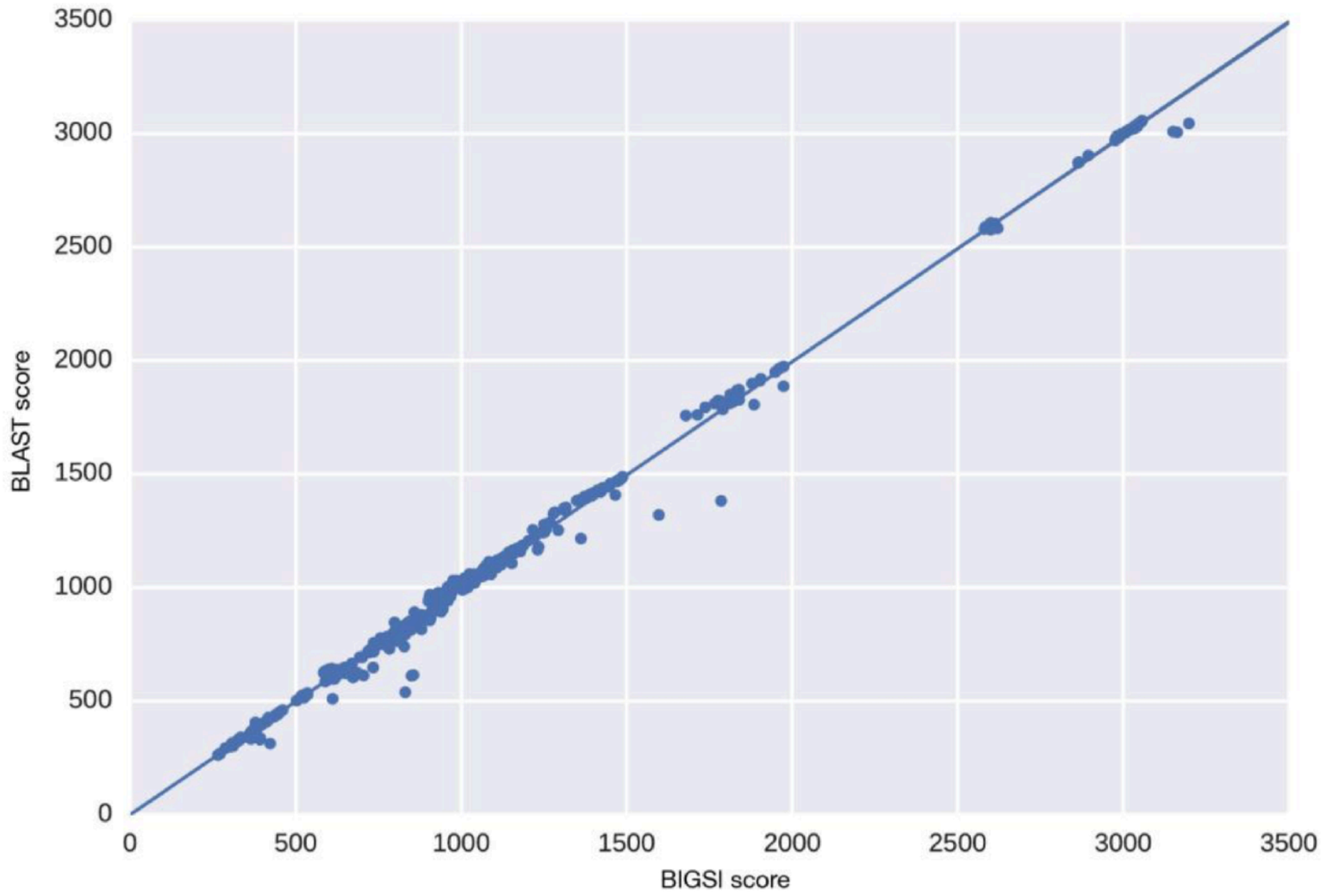
SNPs dominate (π=0.001)
0.7% of a typical genome in SV

4 *E. coli* genomes



Two genomes share 60% of genes (π=0.02)

For long queries, BIGSI lets you find the genomes of interest

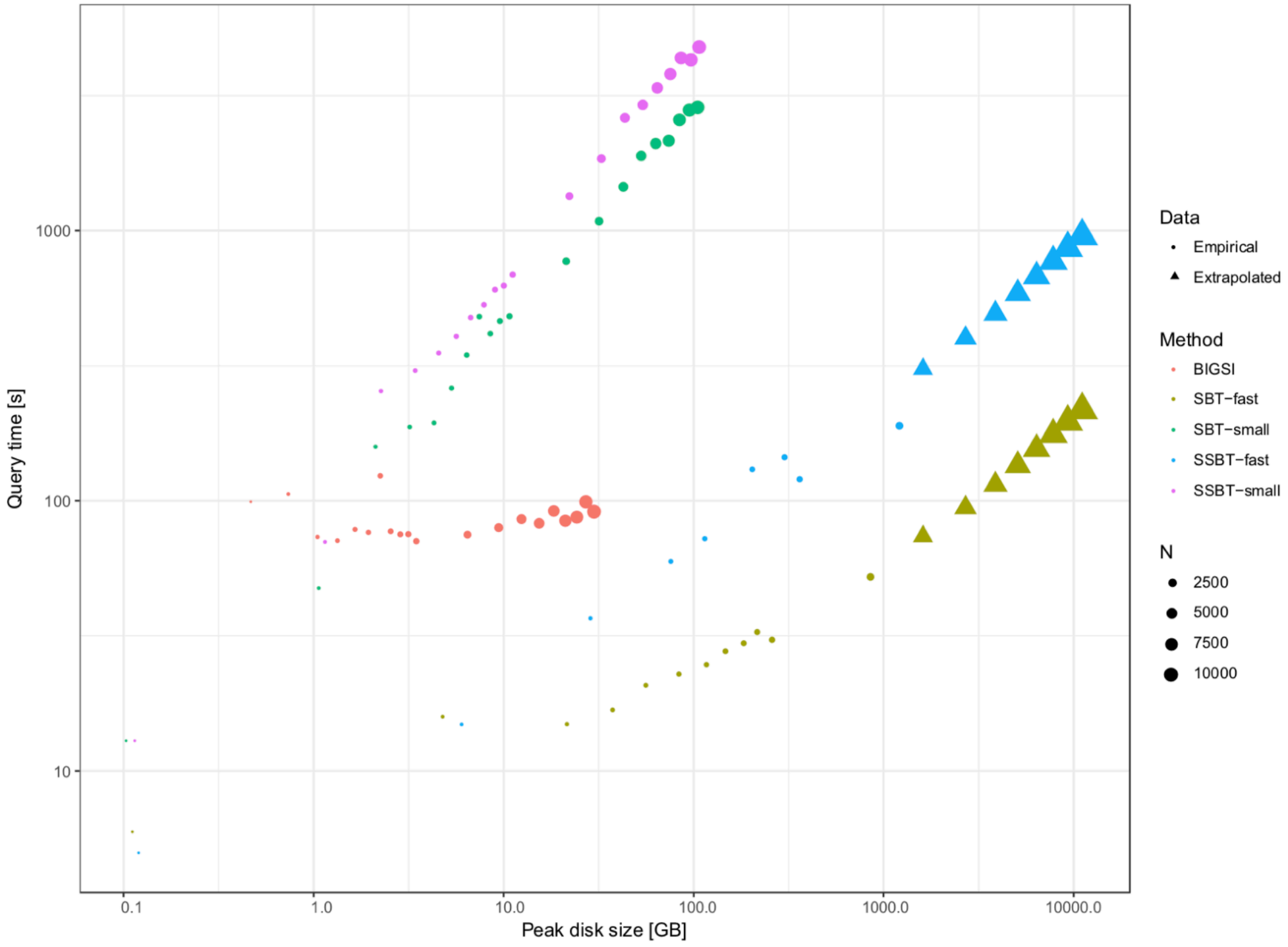


Supplementary Figure 3

BIGSI scores vs. megaBLAST scores.

megaBLAST scores for a search of 100 antimicrobial resistance genes in a BLAST database of RefSeq-81 vs. the equivalent BIGSI scores in a search of a BIGSI of RefSeq-81. Pearson correlation of the scores was r = 0.998.

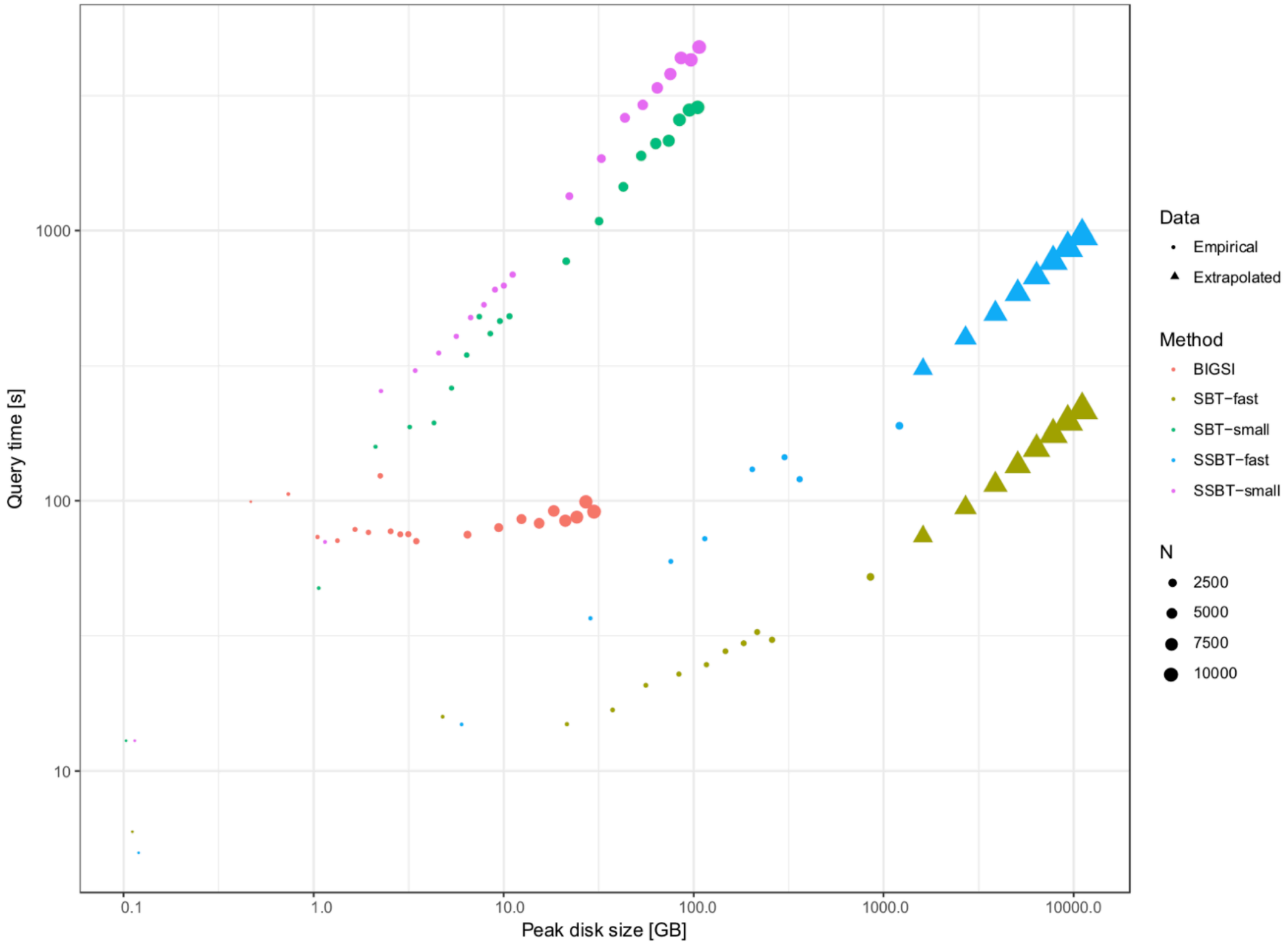
Size of different indices



Supplementary Figure 4

Speed–space tradeoffs for exact-match queries.
We show query time for 2,157 antimicrobial resistance genes with T = 100% vs. peak disk size when searching databases of sizes from 10–10,000 microbial datasets. BIGSI's query time does not increase significantly with N, as in this regime the query time is dominated by the constant time row lookups, rather than the bit-wise AND calculations.

Size of different indices



Supplementary Figure 4

Speed–space tradeoffs for exact-match queries.
We show query time for 2,157 antimicrobial resistance genes with T = 100% vs. peak disk size when searching databases of sizes from 10–10,000 microbial datasets. BIGSI's query time does not increase significantly with N, as in this regime the query time is dominated by the constant time row lookups, rather than the bit-wise AND calculations.

COBS

COBS: a Compact Bit-Sliced Signature Index

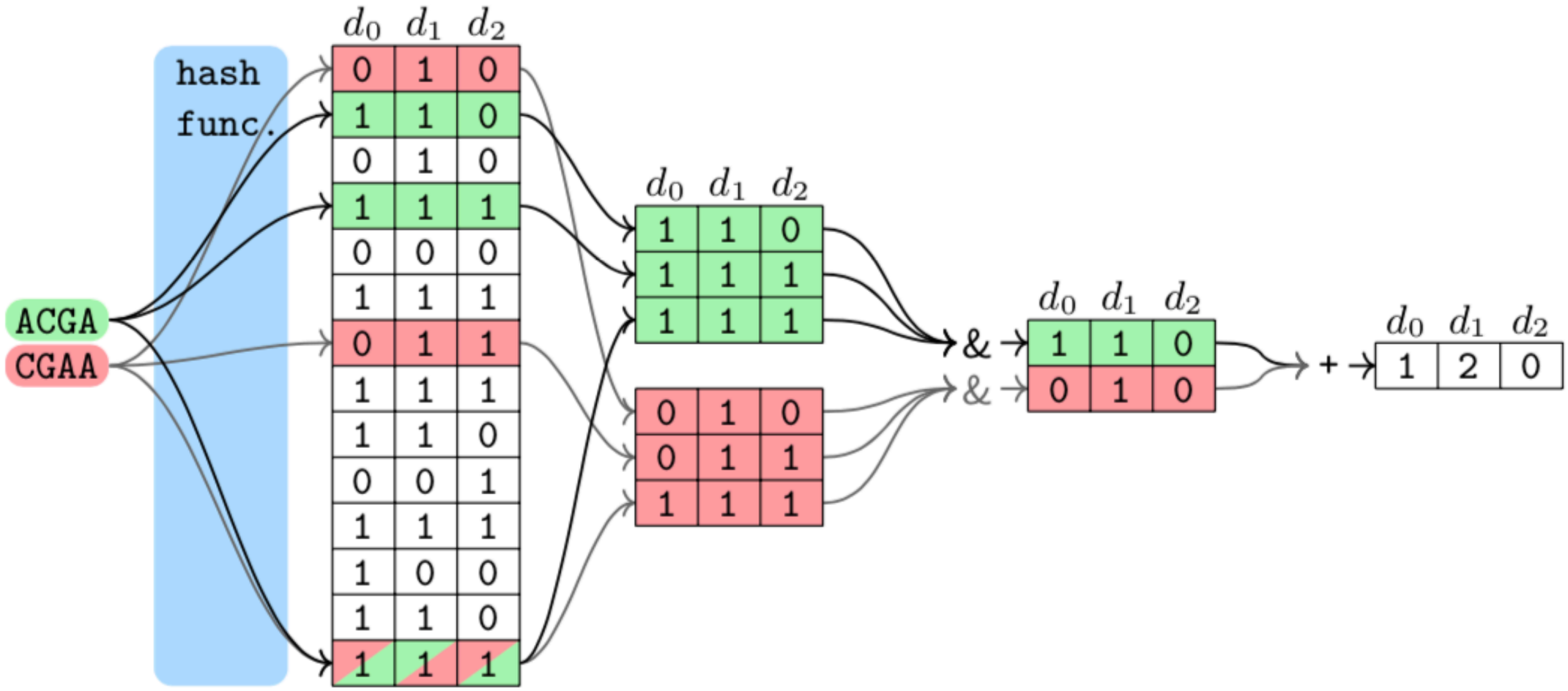
Timo Bingmann[1], Phelim Bradley[2], Florian Gauger[1], and Zamin Iqbal[2]

[1] Institute of Theoretical Informatics,
Karlsruhe Institute of Technology, Germany
[2] European Molecular Biology Laboratory,
European Bioinformatics Institute,
Cambridge, United Kingdom

Like BIGSI (same basic structure), but use different sized Bloom filters for “documents” of different size.

COBS

Conceptual query scheme for BIGSI & COBS



COBS

Idea: Some samples are small, some samples are medium, and some are big

Q : Why use a Bloom filter of the same size for all of them?

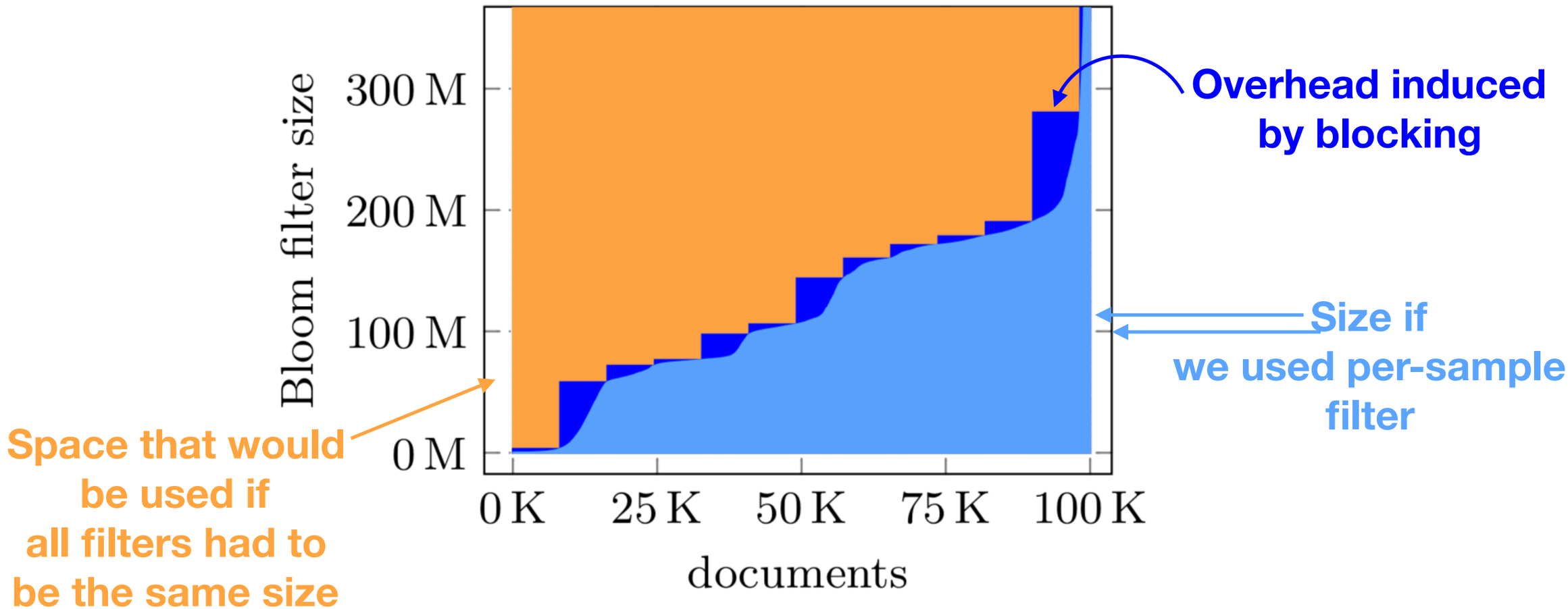
A : Don’t!

Q : So, use a different size for each experiment?

A : No, we want to store filters in chunks, and per-sample sizes make this hard.

Solution: Put samples into “blocks” based on the filter size we use.

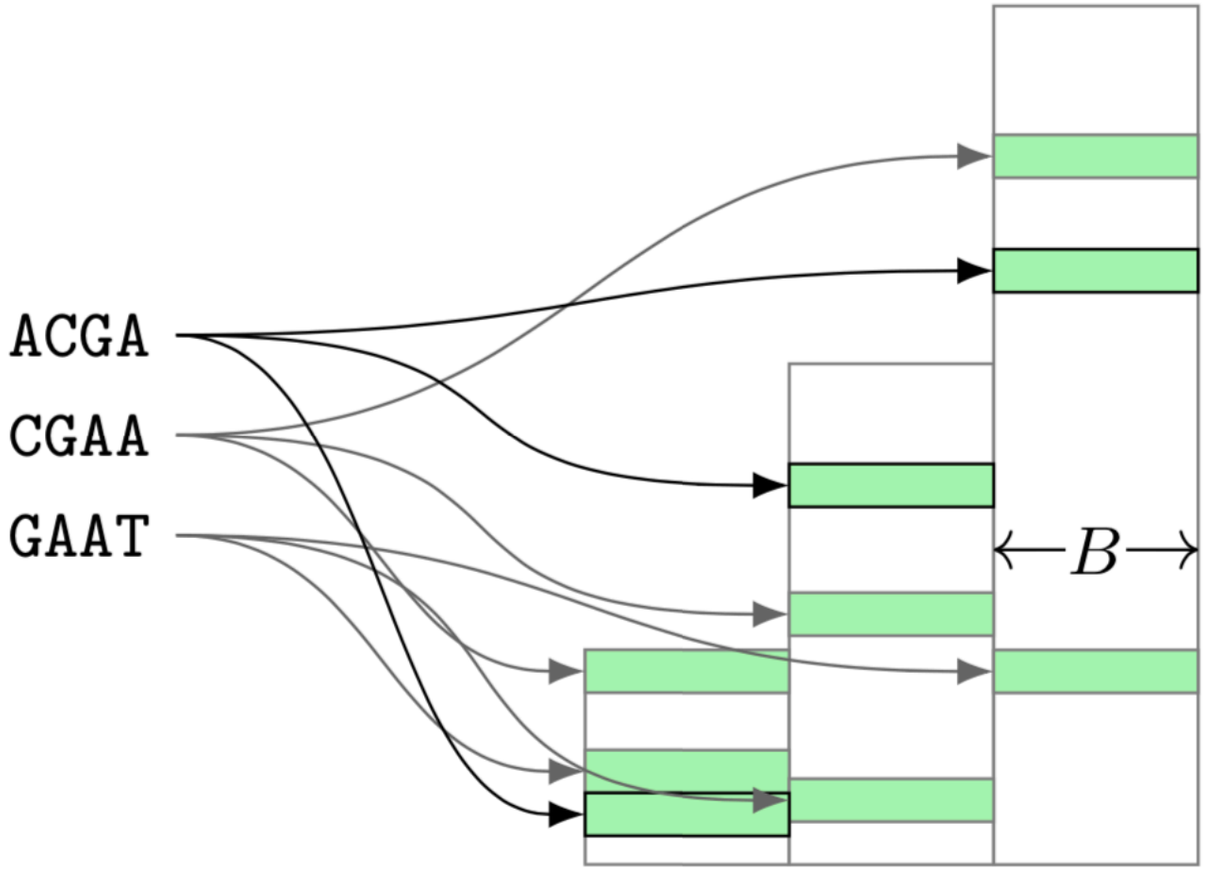
In practice, use a batch size of e.g. 8192 experiments



COBS

To search, we must look up results in filter blocks of all sizes. To avoid having to hash the keys multiple times, COBS uses the following idea:

> Instead of calculating a new hash function for each filter, we propose to use only one function with a larger output range and then use a modulo operation to map it down to each individual filter's size



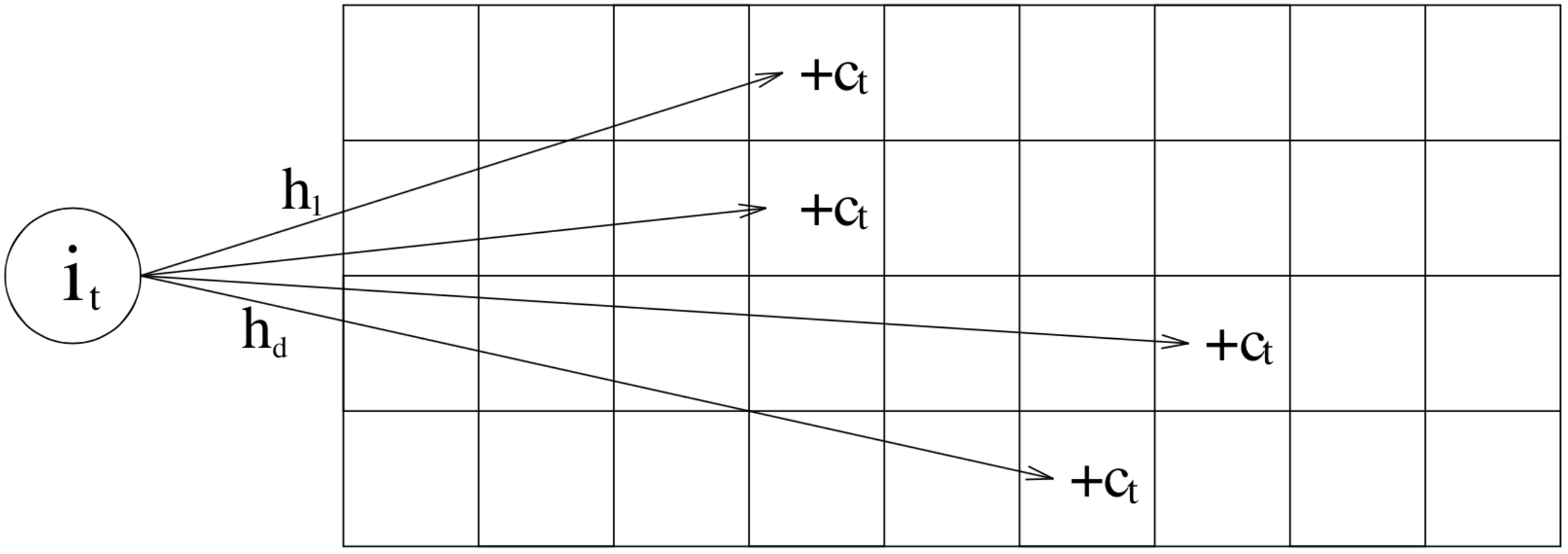
COBS

phase	SBT	SSBT	AllSome-SBT	HowDe-SBT	Seq-Othello	Mantis	BIGSI	ClaBS	COBS
ℓ	Query Wall-Clock Time in Seconds								
31 bp r0	31	80	20	34	62	12	281	10	8
31 bp r2	26	76	19	33	62	13	289	9	8
100 bp r0	663	3 183	100	600	73	22	783	14	9
100 bp r2	649	3 153	95	588	73	23	455	14	9
1000 bp r0	794	3 466	112	670	63	21	660	15	10
1000 bp r2	781	3 435	108	659	64	27	310	13	10
10000 bp r0	802	3 273	112	622	62	23	699	16	11
10000 bp r2	790	3 243	111	613	62	22	316	15	11
total r0–r2	6 775	29 833	1 007	5 710	783	252	5 177	154	114
	Document False Positive Rate for 31 bp Queries								
rate	0.004	0.004	0.004	0.004	0.001	0.000	0.027	0.024	0.227

	Index Size in MiB								
size	19 844	3 254	21 335	1 911	4 410	16 486	27 794	16 236	3 022

Intermezzo : The Count Min Sketch

A probabilistic data structure for counting :
Instead of a an array of m-bits, store a 2D, array, CM, of size d x w — d is called the depth of the array, and there are d *independent* hash functions, w is called with width of the array. This is an O(wd) data structure.



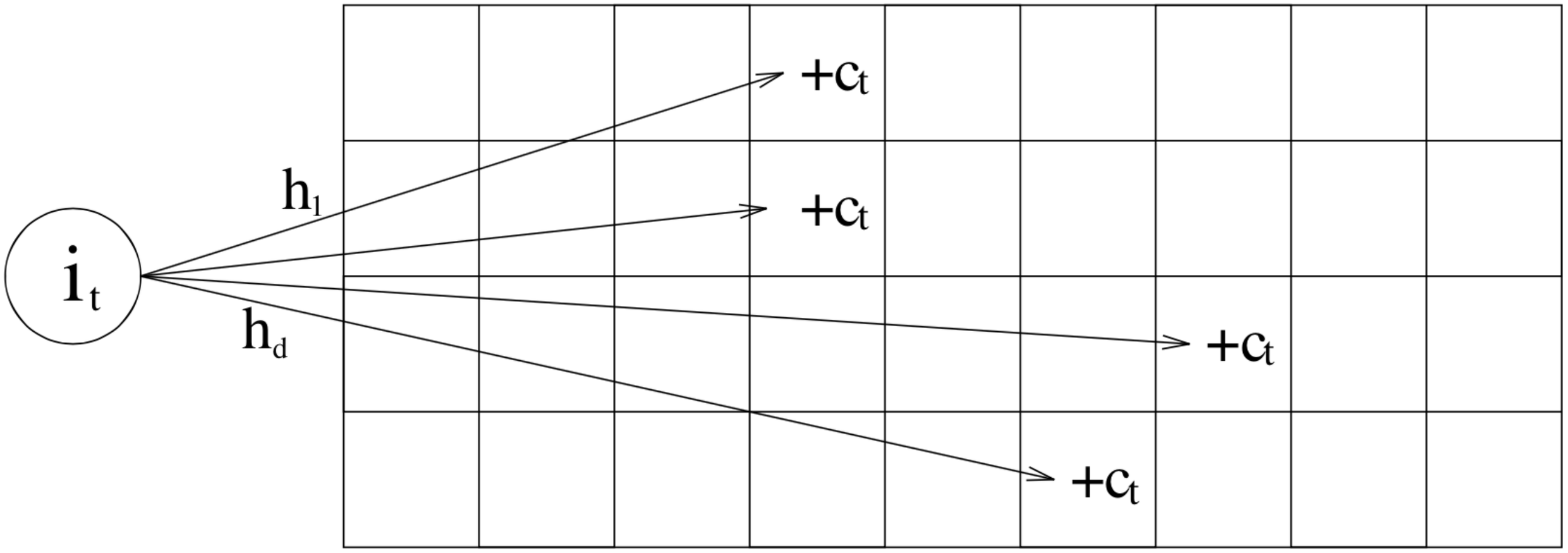
Intermezzo : The Count Min Sketch

Like Bloom filters, 2 mains operations:

Update (k, v) — for each entry CM[i, $h_i(k)$], where $0 < i < d$, increment the value by v.

Query (k) — compute $v = \min_{0 < i < d} CM[i, h_i(k)]$

Both are O(d) operations



Intermezzo : The Count Min Sketch

Similar error analysis to Bloom filters (won't prove bounds)

Let $\hat{a}_i$ be the result returned by Query(i). We have that:

$$a_i \leq \hat{a}_i \qquad \text{(always)}$$

$$\hat{a}_i \leq a_i + \epsilon \left|\left|\mathbf{a}\right|\right|_1 \qquad \text{(with probability at least } \frac{1}{\delta}\text{)}$$

where,

$$w = \left\lceil \frac{e}{\epsilon} \right\rceil, d = \left\lceil \ln(\frac{1}{\delta}) \right\rceil, \text{ and } \left|\left|\mathbf{a}\right|\right|_1 = \sum_{i=1}^{n} |a_i|$$

Using the CMS for approximate set membership

Sub-linear Sequence Search via a Repeated And Merged Bloom Filter (RAMBO)

Minghao Yan
Department of Computer Science
Rice University
Houston, U.S.
minghao.yan@rice.edu

Gaurav Gupta
Department of Electrical and Computer Engineering
Rice University
Houston, U.S.
gaurav.gupta@rice.edu

Benjamin Coleman
Department of Electrical and Computer Engineering
Rice University
Houston, U.S.
ben.coleman@rice.edu

Todd Treangen
Department of Computer Science
Rice University
Houston, U.S.
treangen@rice.edu

Anshumali Shrivastava
Department of Computer Science
Rice University
Houston, U.S.
anshumali@rice.edu

arXiv.org > q-bio > arXiv:1910.04358

(Submitted on 10 Oct 2019 (v1), last revised 11 Oct 2019 (this version, v2))

Quantitative Biology > Genomics

Using the CMS for approximate set membership

Main idea : Create a CMS of **Bloom Filters** rather than a CMS of counters. To test membership of a set of k-mers, test them in each of the Bloom filters and look at the datasets in the intersection of results.

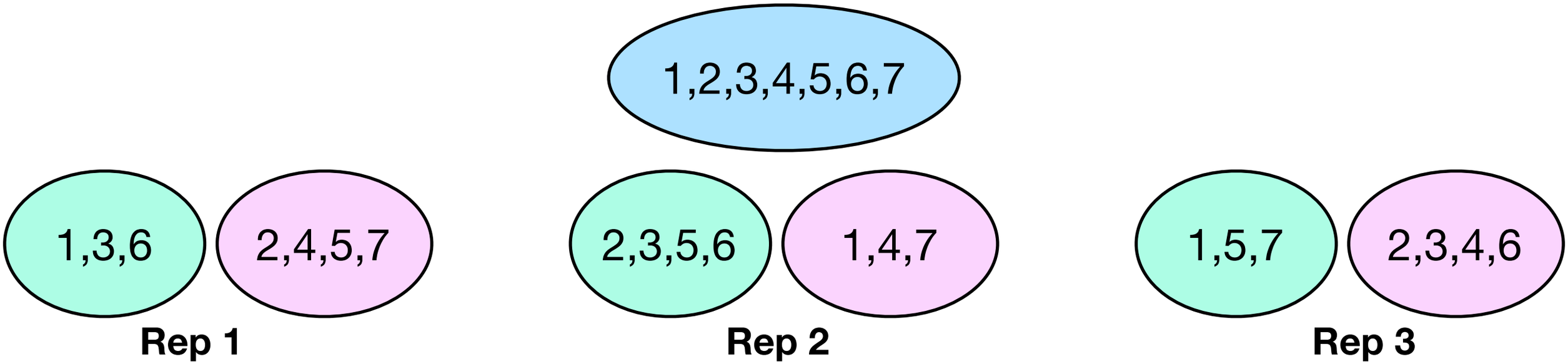
CMS counters → CMS Bloom Filters

Insertion : + → Insert in Bloom filters (set bits for each k-mer in each filter to 1)

Query : min(.) → Take intersection of all datasets (within each partition & repetition) that answer yes for the hash positions corresponding to this k-mer.

Using the CMS for approximate set membership

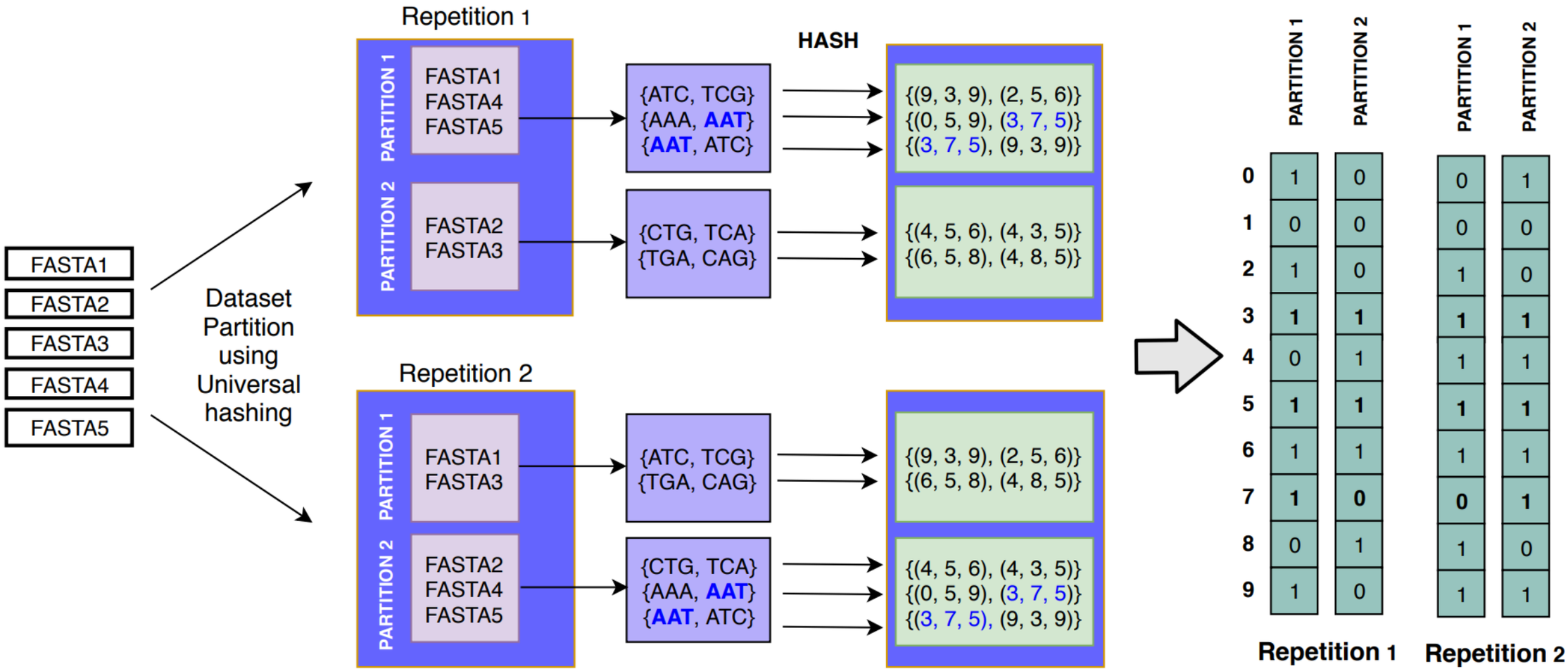
Key : Will partition the universe of datasets into W disjoint sets, and will repeat this process D times. When we issue a query, we will look at the datasets that, for each repetition, have the same "partition signature" as the query. Over many random trials (repetitions), the probability that a dataset appears in each partition shared with the query *by chance* falls off geometrically.



What is the probability that a "random" number has partition sig 1,1,2 ?

Additionally, the Bloom filter that falls in each part. In each rep. will have to say "Yes" to the query

Using the CMS for approximate set membership



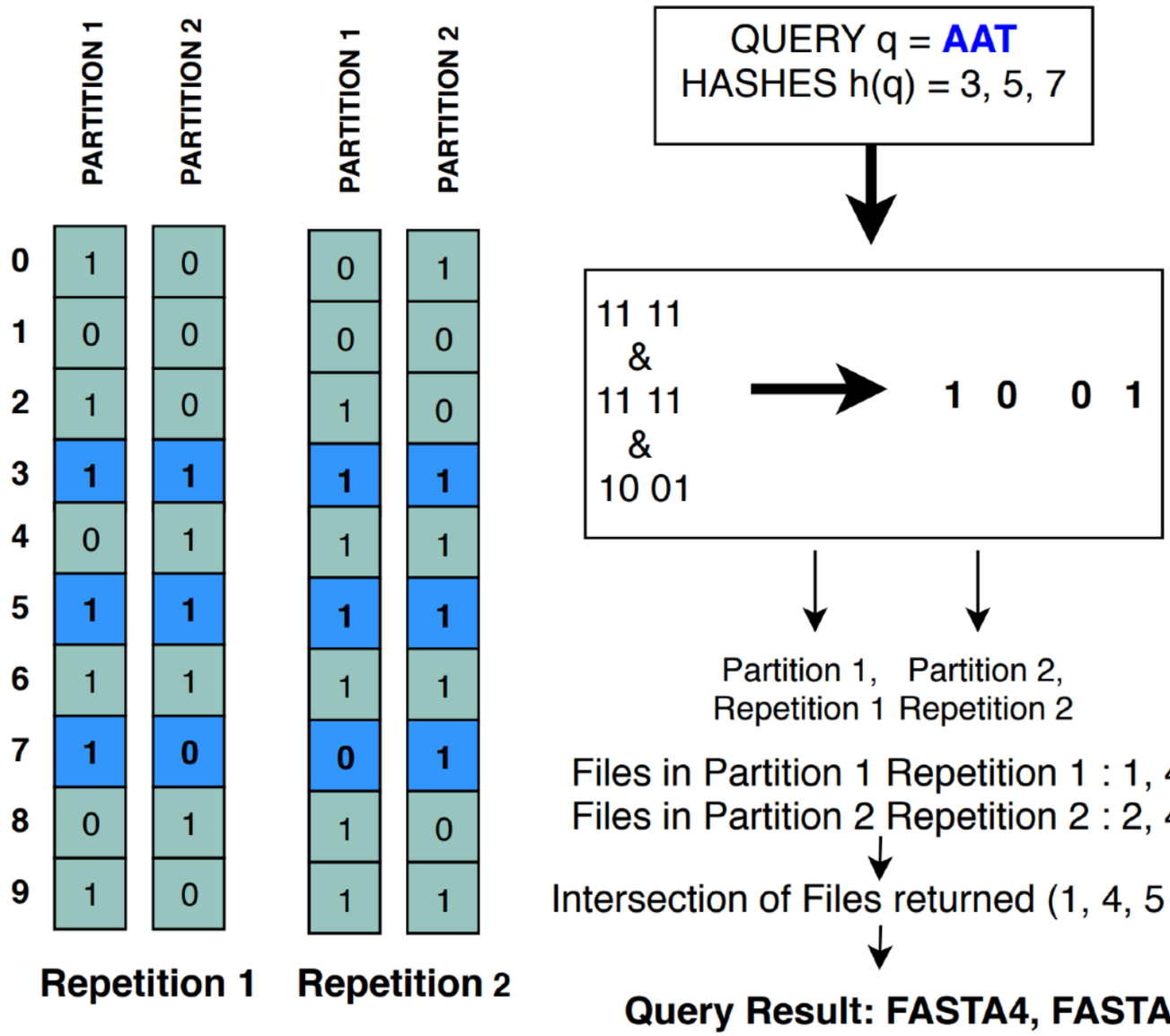
Note: We only need to hash k-mers once

Algorithm 1 Algorithm for insertion in RAMBO architecture

Input: Set $\mathcal{D}$ of N databases of k-mers
Given: Parameters $W \times D$ and false positive rate p
Result: RAMBO (size: $W \times D$)
Generate D partition hash functions $ph_1(\cdot), ...ph_D(\cdot)$
RAMBO $\leftarrow W \times D$ array of Bloom filters
while Input FASTA-i **do**
  for k-mer $x \in$ FASTA$_i$ **do**
    for $d = 1, ...D$ **do**
      insertBloomFilter(x, RAMBO[$ph_d(x), d$])
    end for
  end for
end while

Partition datasets into W (here = 2) disjoint partitions, and repeat the insertion process D (here = 2) times. Each partition within each repetition corresponds to a Bloom filter (column).

Using the CMS for approximate set membership



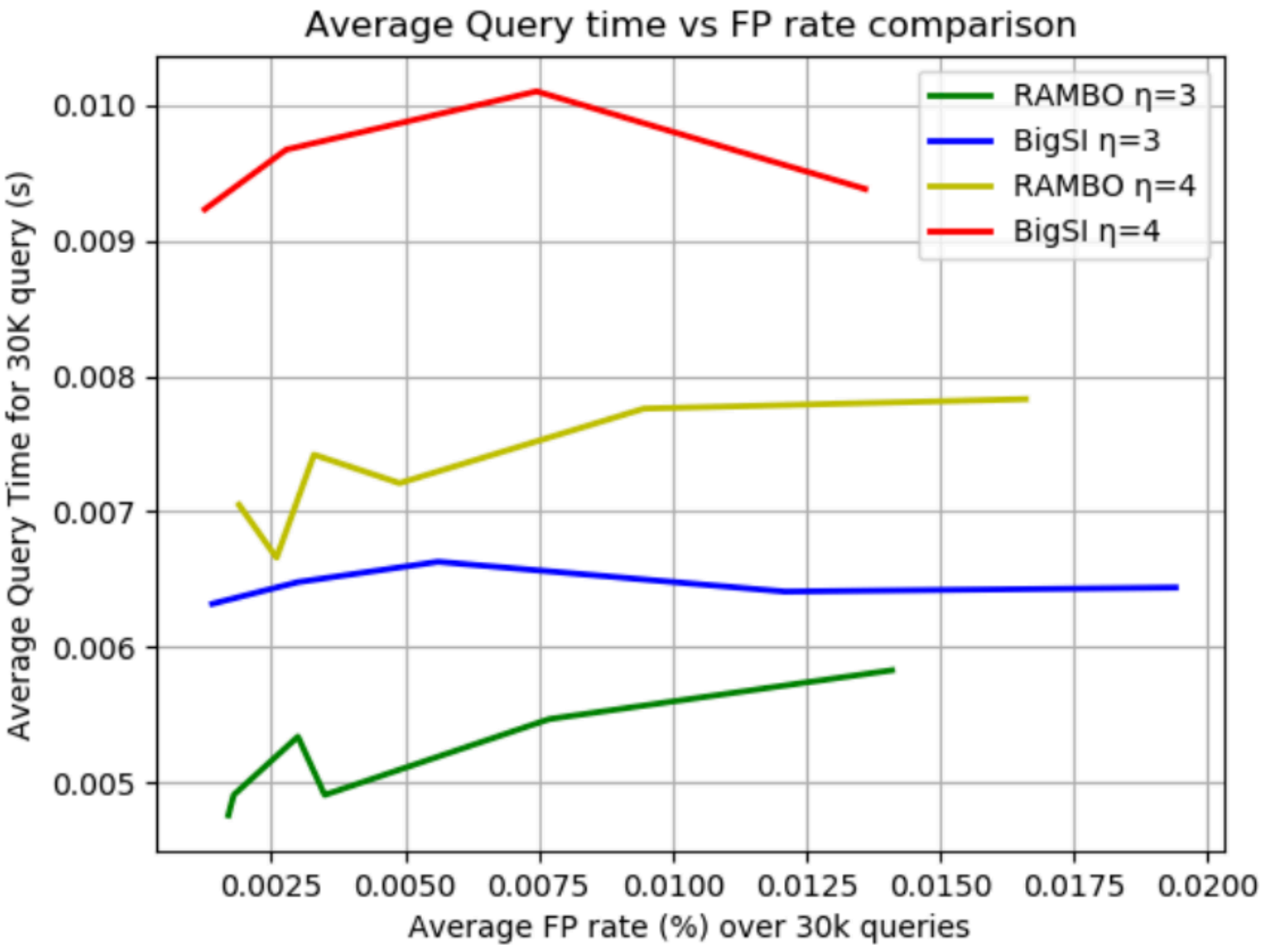
Algorithm 2 Algorithm for query using RAMBO architecture

Input: gene sequence q
Given: RAMBO bit matrix M (size: $W \times D$)
Result: Set of datasets, each of which contains q.
UNION = 0
for $d = 1 : D$ **do**
  ID = $h_d(q)$
  $bitslice_d$ = get row number ID from M
  UNION = $bitslice_d$ (Bitwise OR) UNION
end for
INTERSECTION = Everything
for i such that UNION[i] = 1 **do**
  INTERSECTION = INTERSECTION ∩ Partition[i]
end for
return INTERSECTION

Using the CMS for approximate set membership

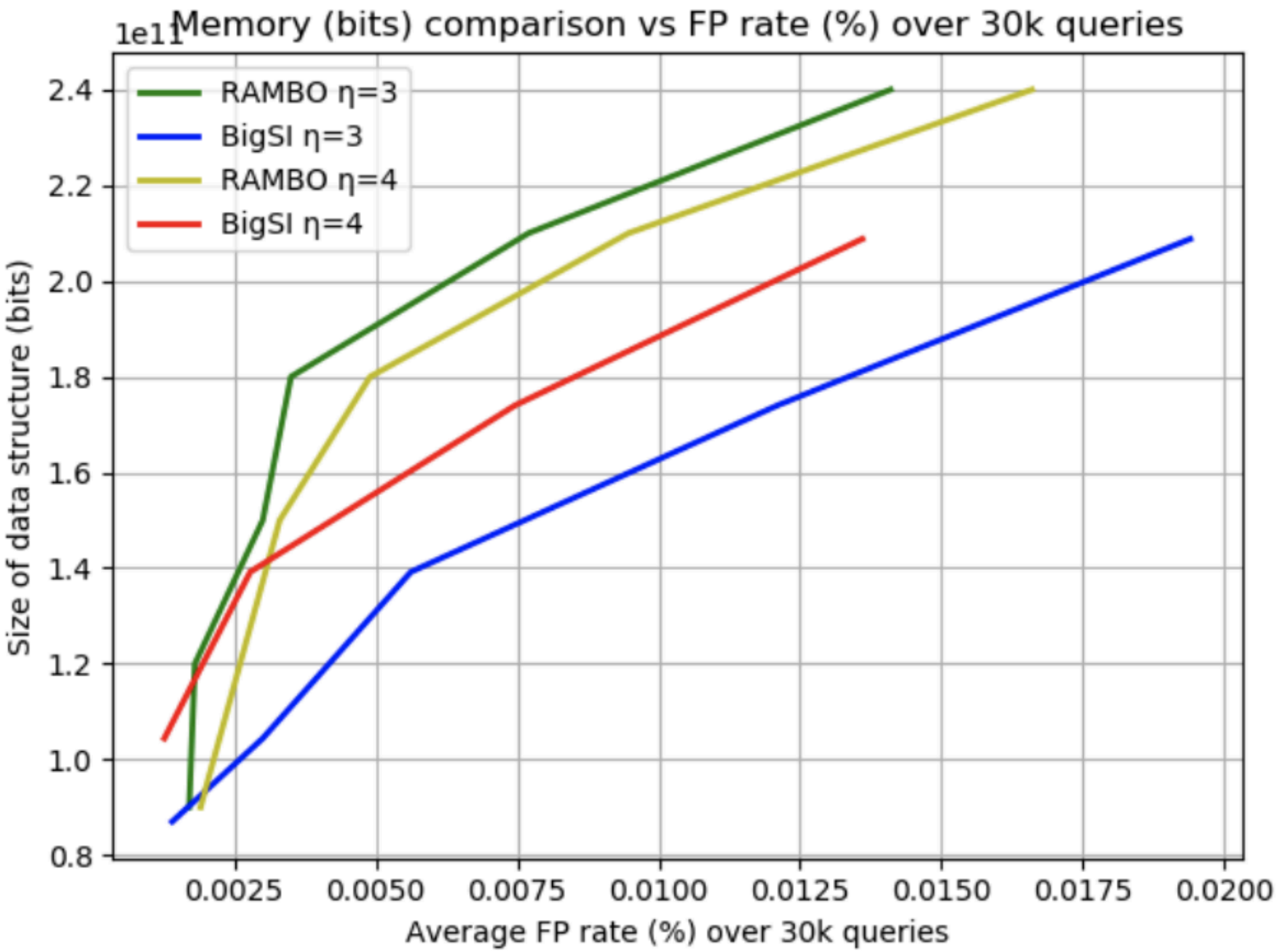
Tested index & query using random subsets of size 3,480 and 2,500 from 136,602 “unique” assemblies from NCBI RefSeq.
Note: The index here is built over assembled references, not raw queries. These data have *massively* different properties, so don’t compare this 3,480 / 2,500 to the 2,586 experiments from the SBT / SSBT / All Some paper.

Using the CMS for approximate set membership



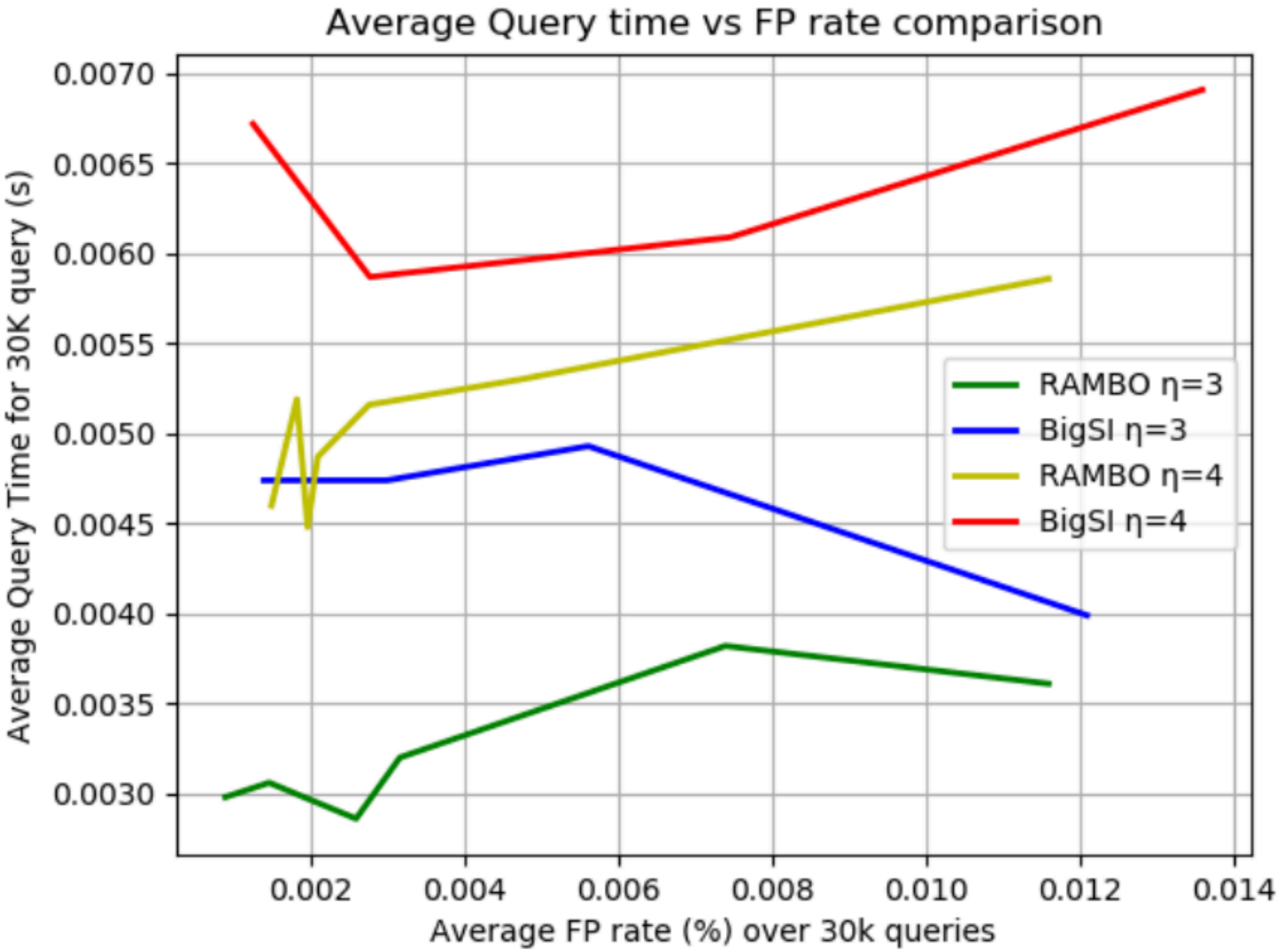
3,480 file index

Using the CMS for approximate set membership



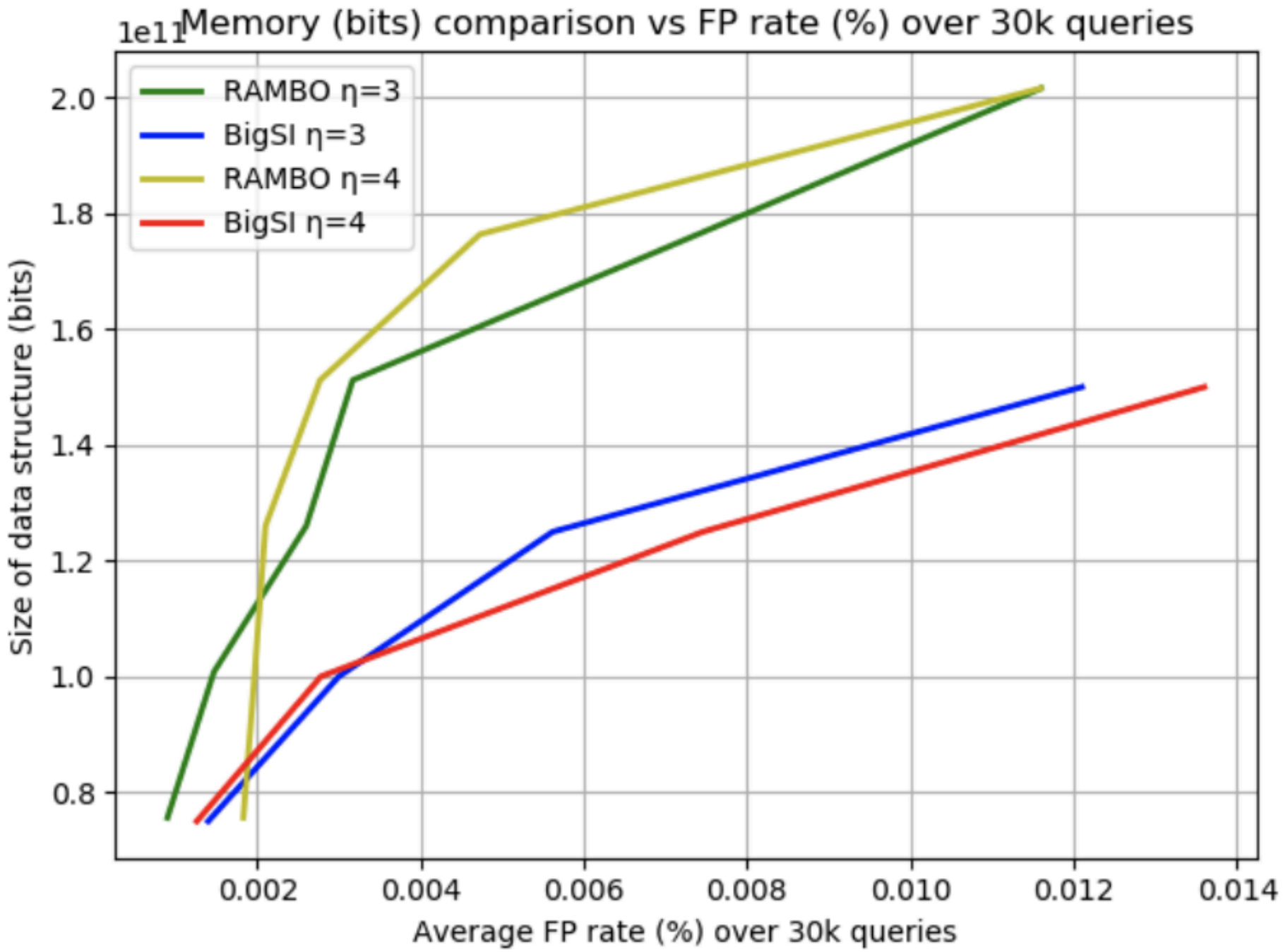
3,480 file index

Using the CMS for approximate set membership



2,500 file index

Using the CMS for approximate set membership



2,500 file index

Using the CMS for approximate set membership

Some major benefits of RAMBO:

Query scales sub-linearly in # of datasets

Adding a new datasets doesn’t require adding a new column to the index (as in BIGSI) — though it does increase the False Positive Rate.

Take-home message for LLSC part 1

Large-scale sequence search is a new but rapidly-growing field.

There is no “clear” best solution yet.

It’s an exciting place to try out new things.

Unfortunately, the cost of entry is high, since even getting the data to perform experiments has a large associated cost.