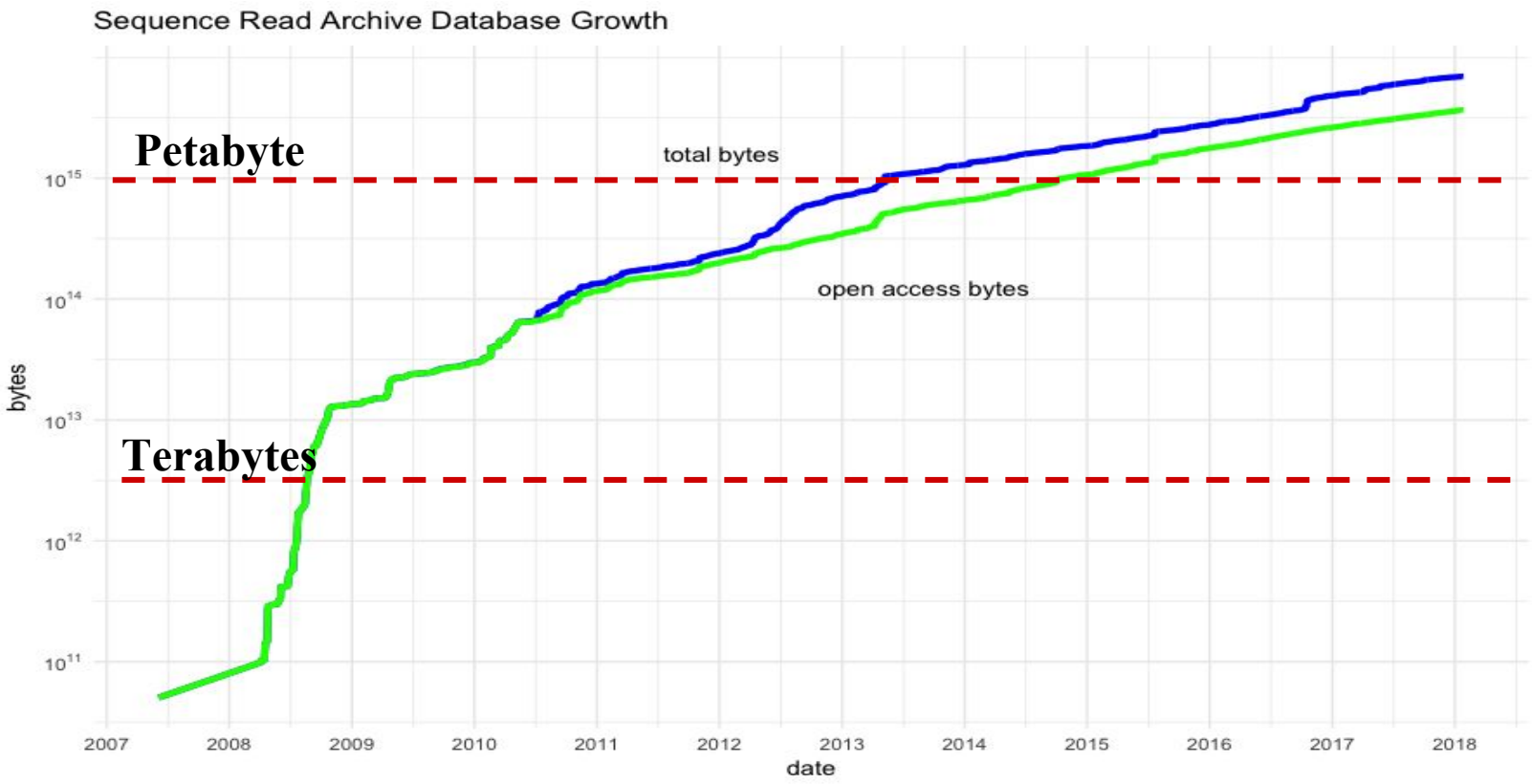
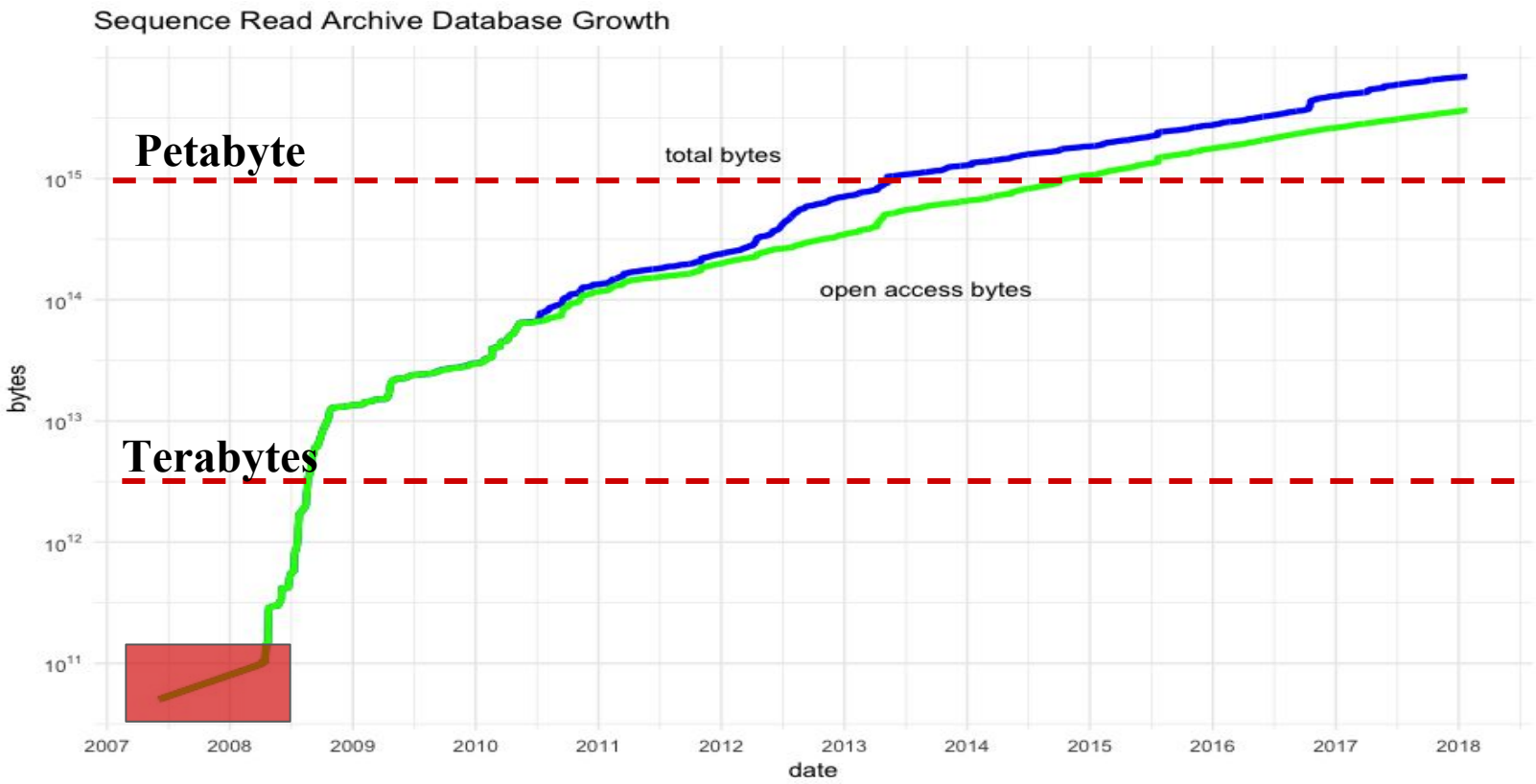
Fast and Space-Efficient Maps: Shrinking Big Data Down to Size

Prashant Pandey
Stony Brook University, NY

Sequence Read Archive (SRA) database growth



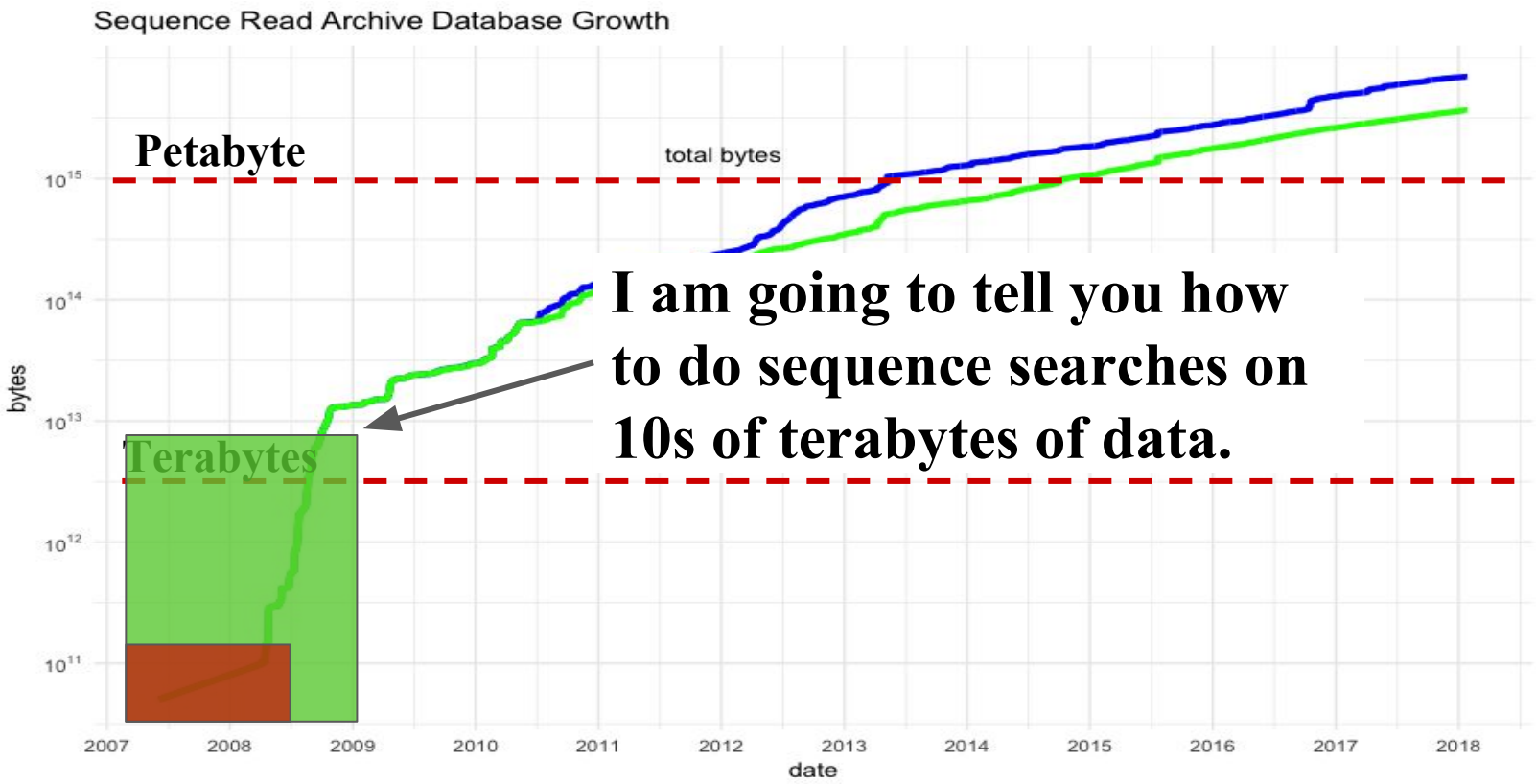
Sequence Read Archive (SRA) database growth



Currently, sequence searches are restricted to a limited number of experiments. [1]

[1] Solomon and Kingsford ‘2016

In this talk



How to build a space-efficient and performant map and use it to build an index on terabytes of raw sequencing data.

How to attack big data problems

How to attack big data problems

Shrink it

Computational biology

Streaming applications

Shrink data to achieve space-efficiency and speed.

How to attack big data problems

Shrink it

Organize it

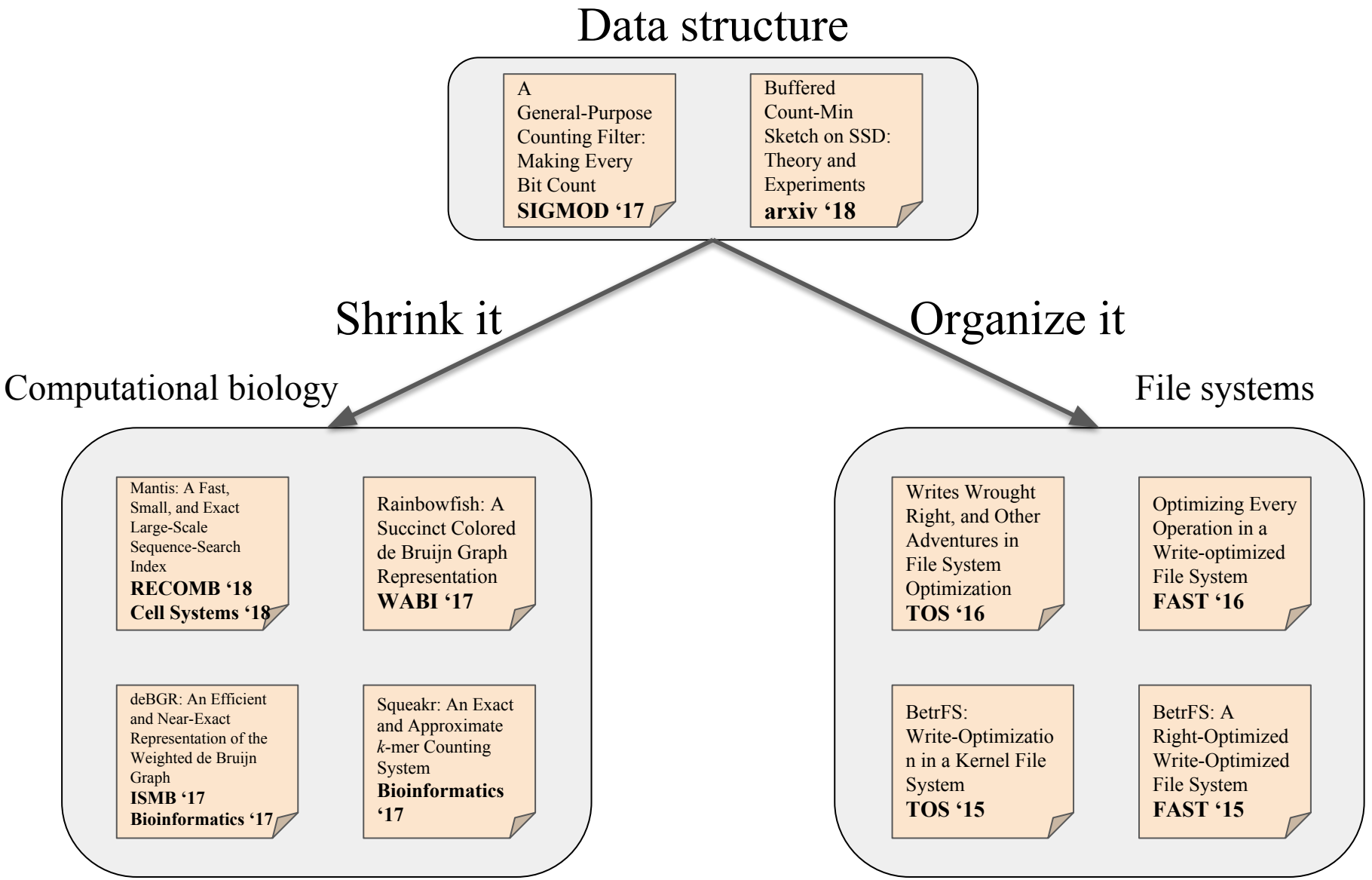
Computational biology

Streaming applications

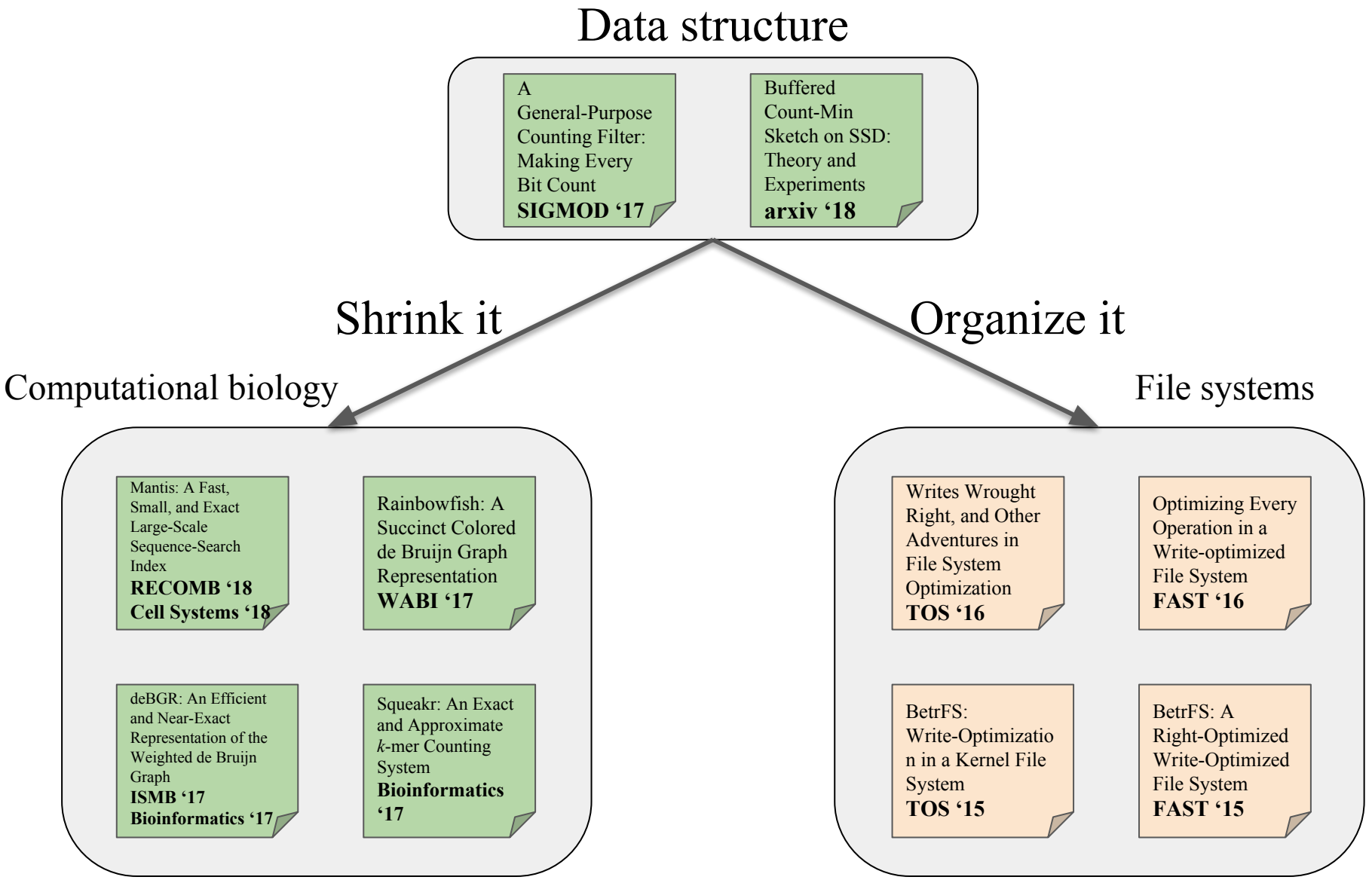
File systems

Organize data to achieve efficient disk accesses and better performance.

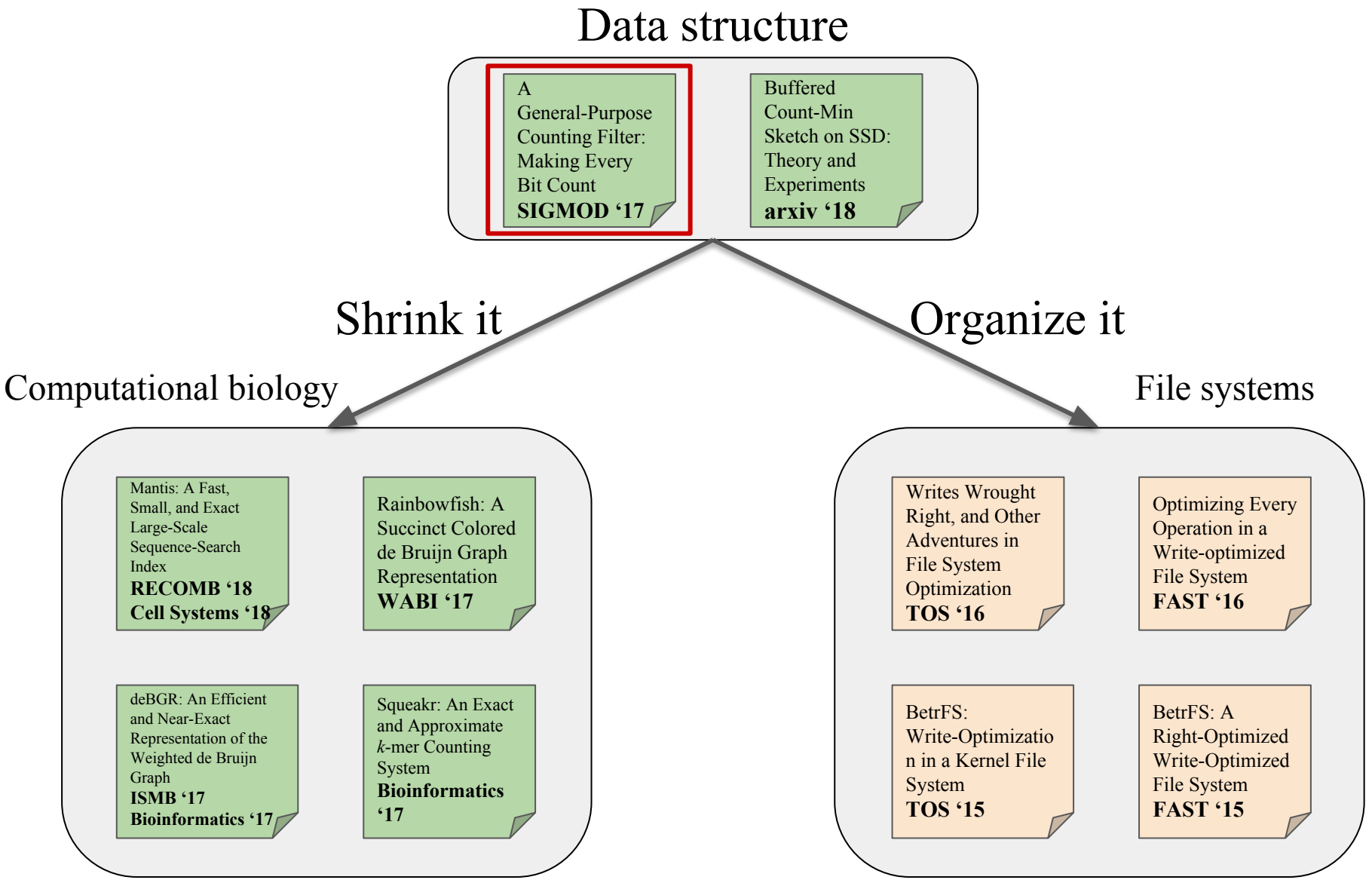
Research output



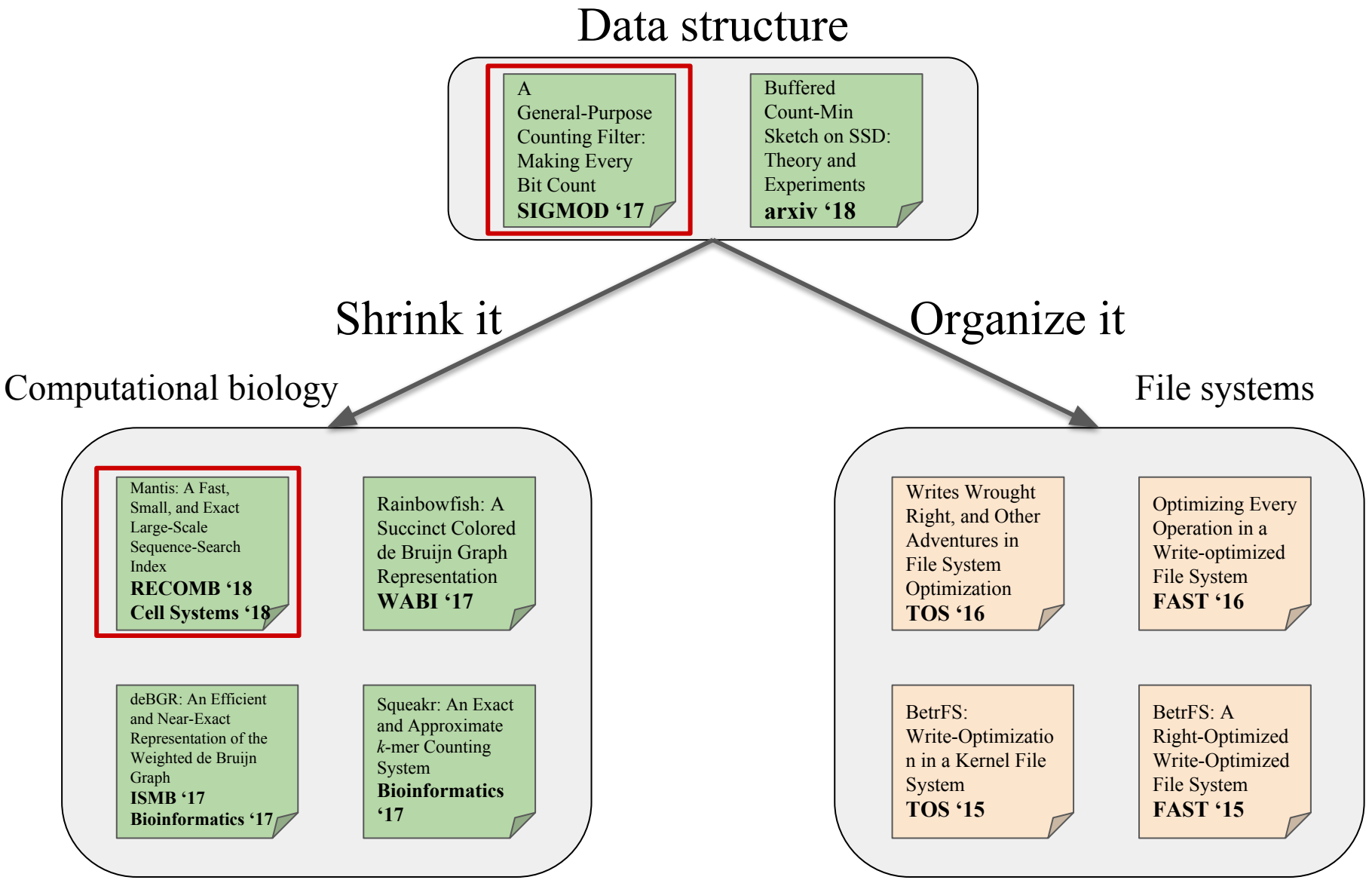
Research output



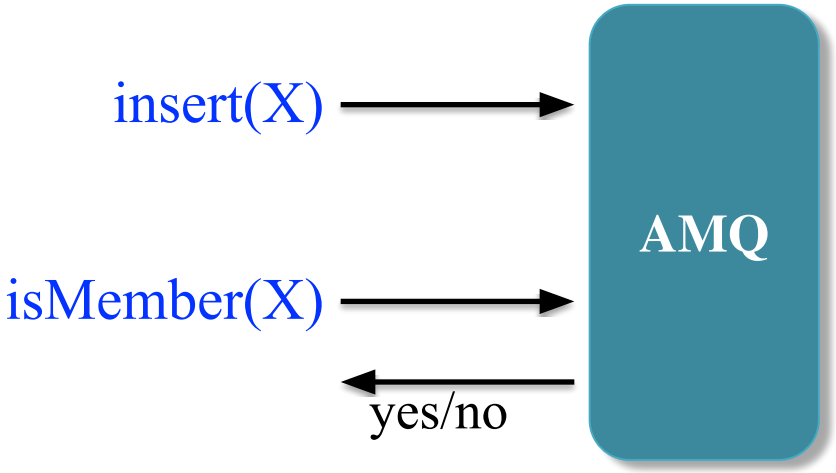
Research output



Research output

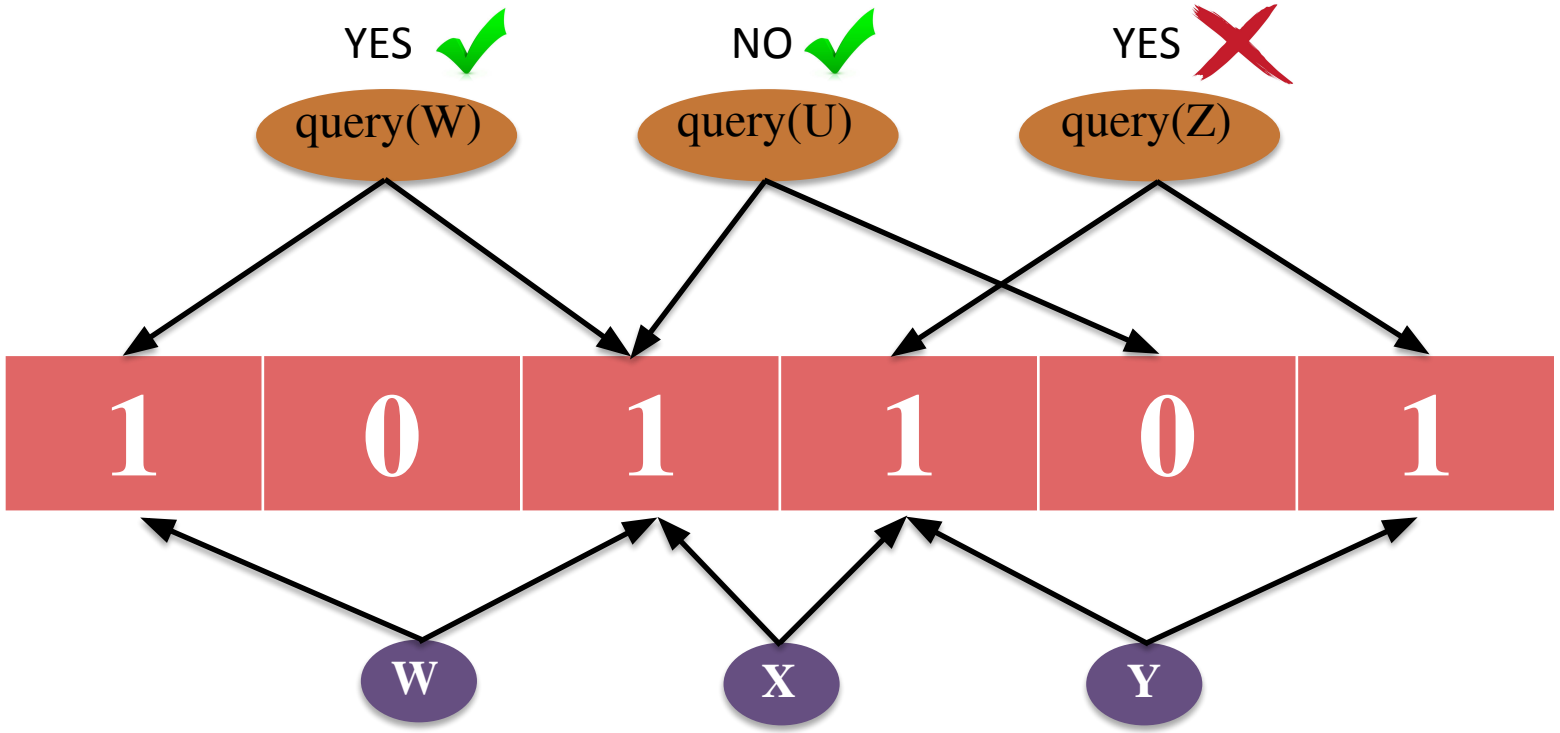


Approximate Membership Query (AMQ)



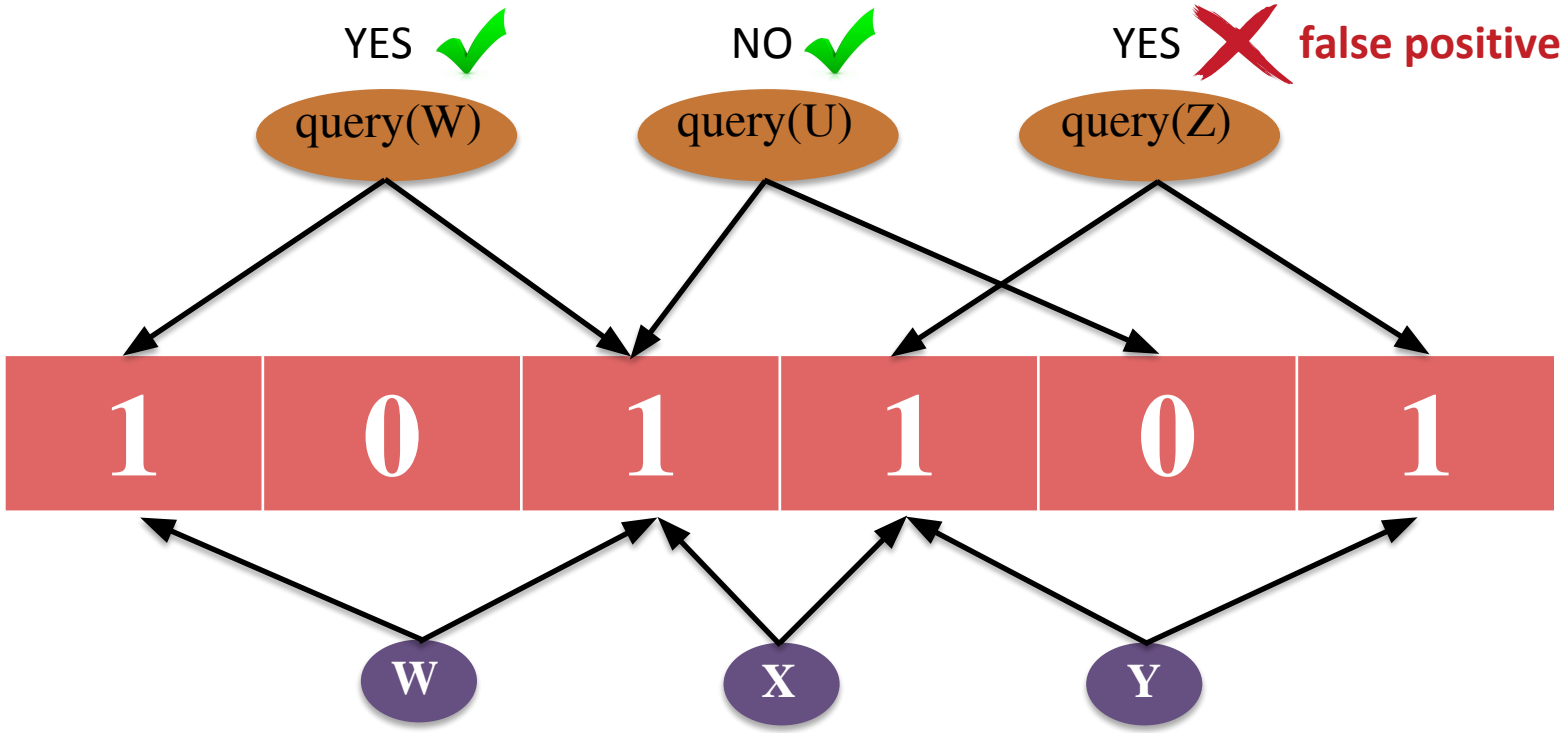
- **An AMQ is a lossy representation of a set.**
- **Operations: inserts and membership queries.**
- **Compact space:**
  - Often taking < 1 byte per item.
  - Comes at the cost of occasional false positives.

Bloom filters



The Bloom filter is a bit array + k hash functions.

Bloom filters



The Bloom filter has a bounded false-positive rate.

Bloom filters are ubiquitous

Streaming applications

Networking

Databases

Computational biology

Storage systems

Application often must work around Bloom filter limitations.

Limitations	Workarounds
No deletions	Rebuild
No resizes	Guess, rebuild if wrong
No enumeration	???
No values	Combine with another data structure

Application often must work around Bloom filter limitations.

Limitations	Workarounds
No deletions	Rebuild
No resizes	Guess, rebuild if wrong
No enumeration	???
No values	Combine with another data structure

Bloom filter limitations increase complexity, waste space, and hurt application performance.

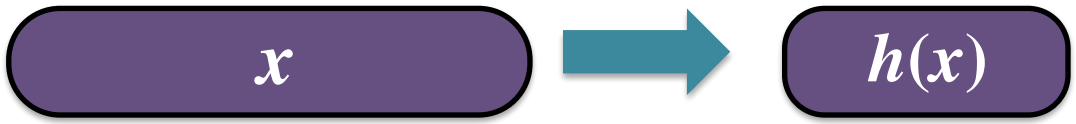
The Quotient filter (QF)

- A replacement for the (counting) Bloom filter.
- Space and computationally efficient.
- Can be used as a map for small key-value pairs.
- Uses variable-sized counts/values.

$$\text{QF space} \leq \text{BF space} + O(\Sigma\, |v(x)|)$$

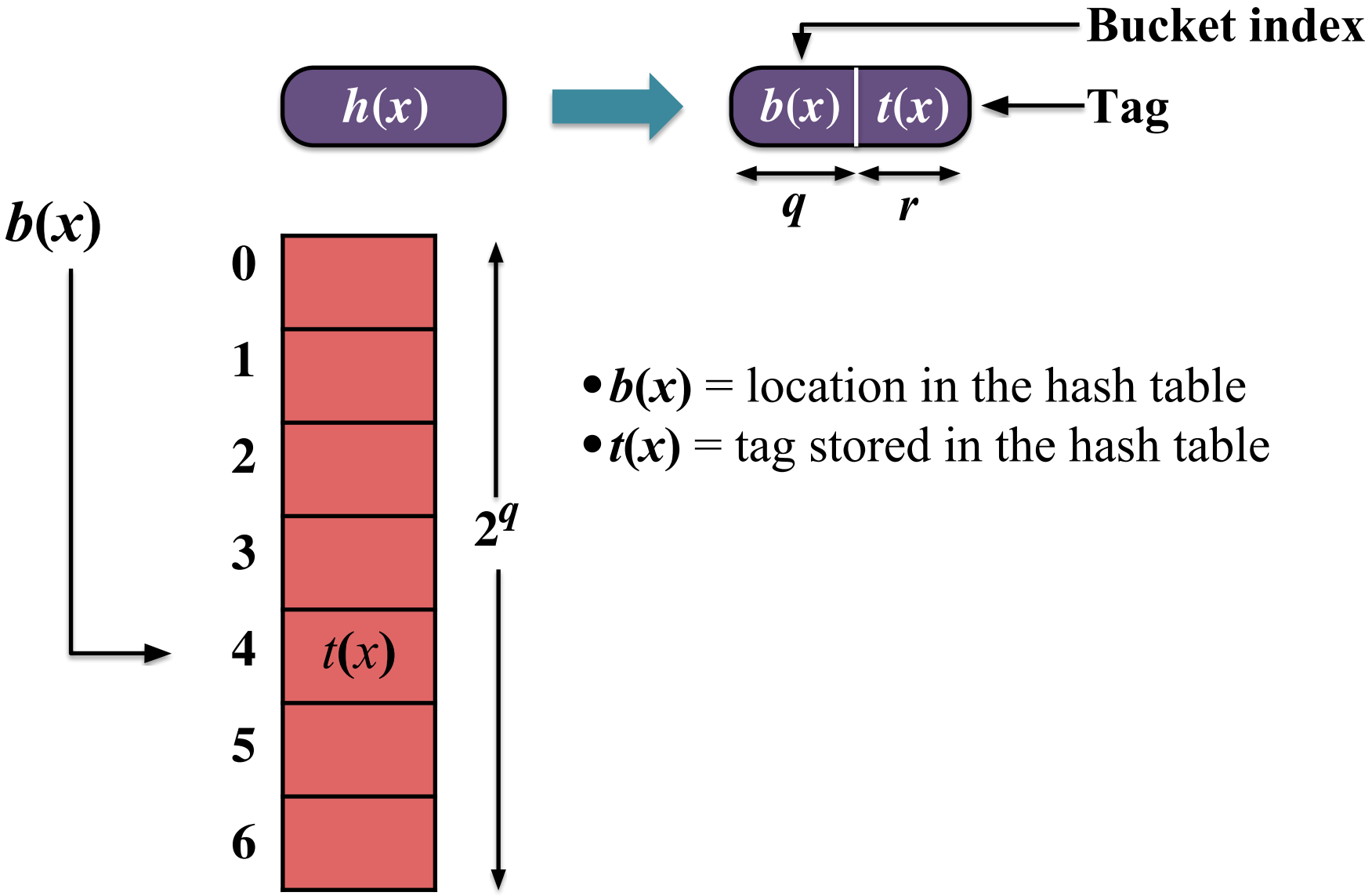
Quotienting: an alternative to Bloom filters

- **Store fingerprint compactly in a hash table.**
  - Take a fingerprint ***h(x)*** for each element ***x***.



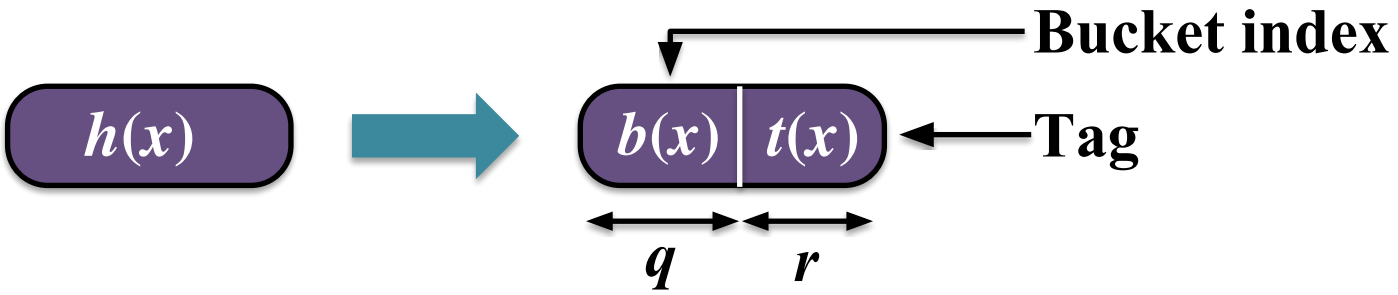
- **Only source of false positives:**
  - Two distinct elements ***x*** and ***y***, where ***h(x)*** = ***h(y)***.
  - If ***x*** is stored and ***y*** isn't, ***query(y)*** gives a false positive.

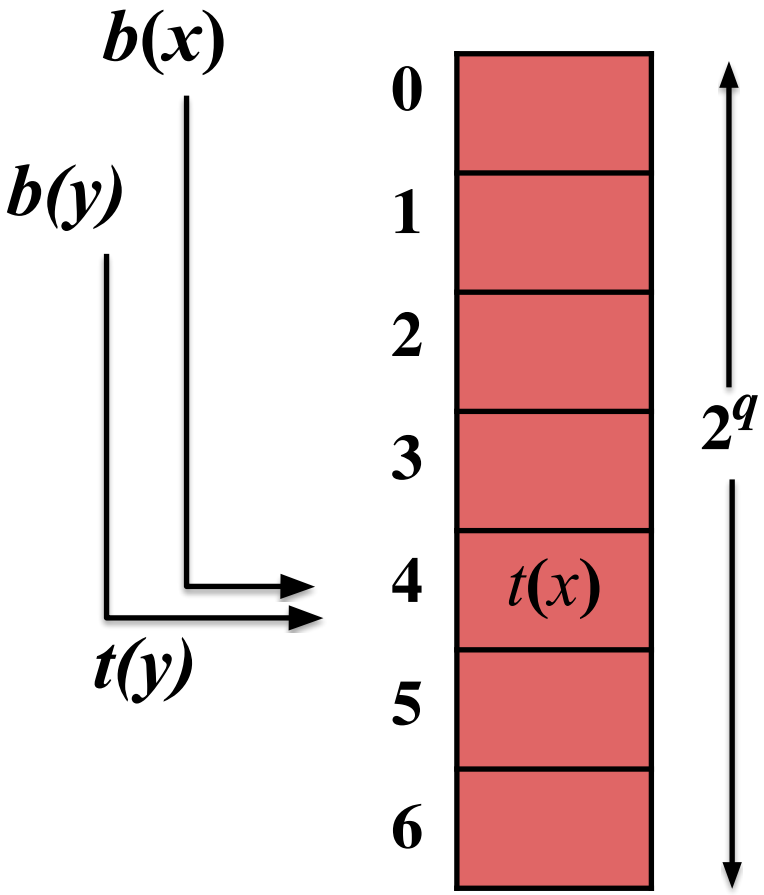
Storing compact fingerprints



- $b(x)$ = location in the hash table
- $t(x)$ = tag stored in the hash table

Storing compact fingerprints

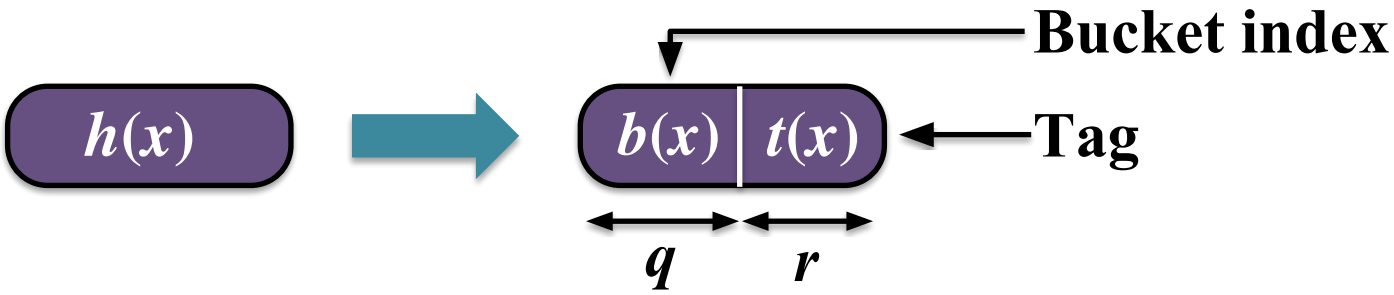


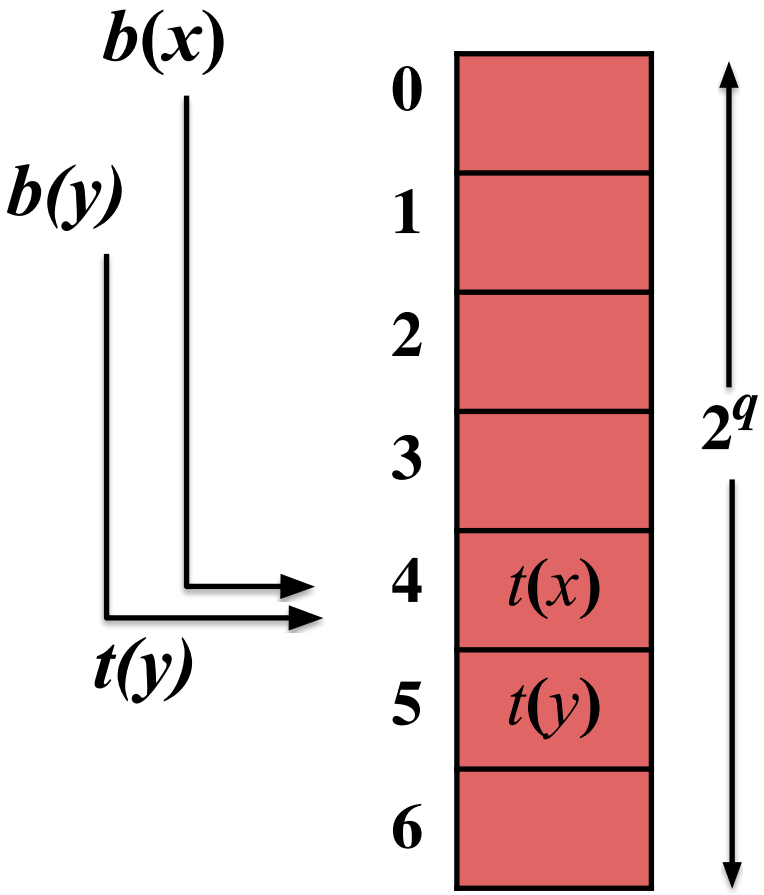


- $b(x)$ = location in the hash table
- $t(x)$ = tag stored in the hash table

Collisions in the hash table?

Storing compact fingerprints



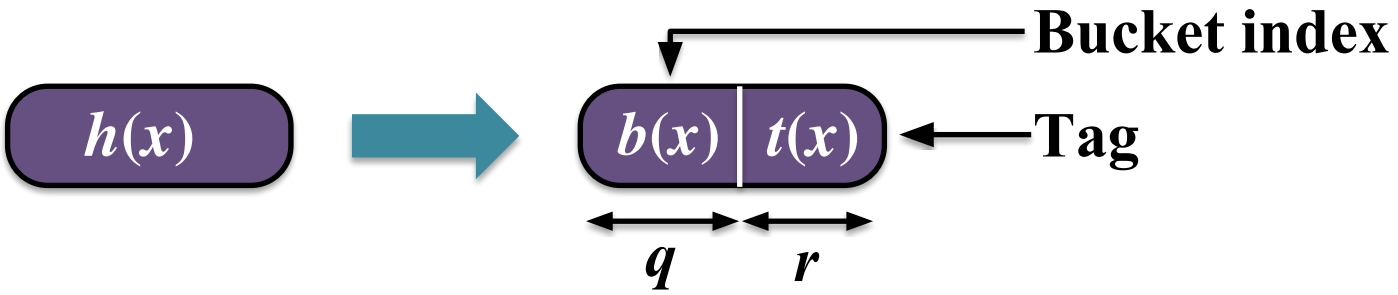


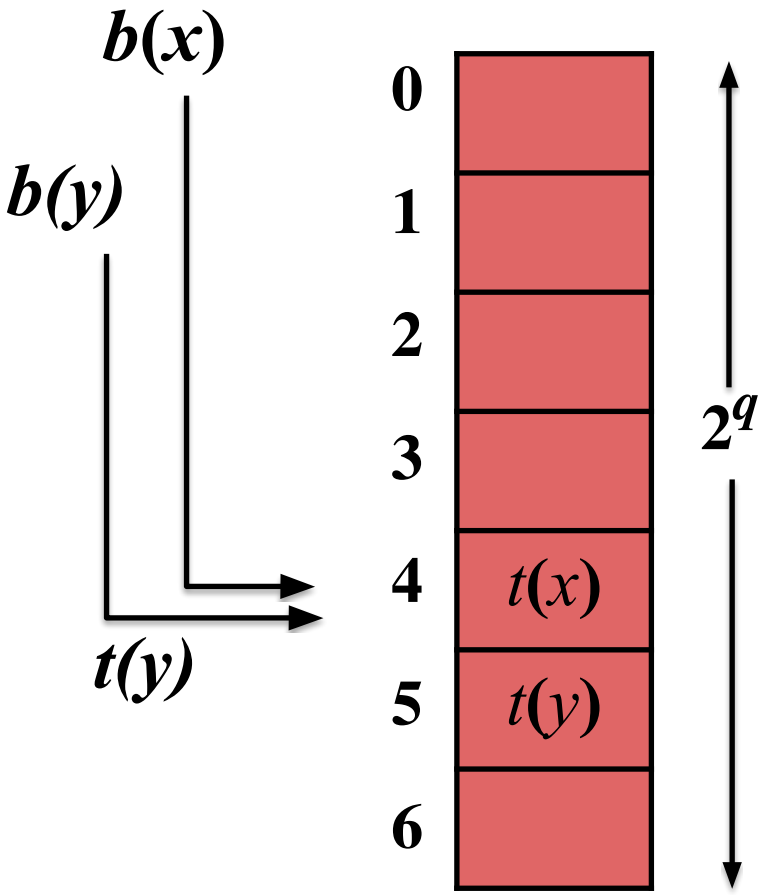
- $b(x)$ = location in the hash table
- $t(x)$ = tag stored in the hash table

Collisions in the hash table?

- Linear probing.

Storing compact fingerprints





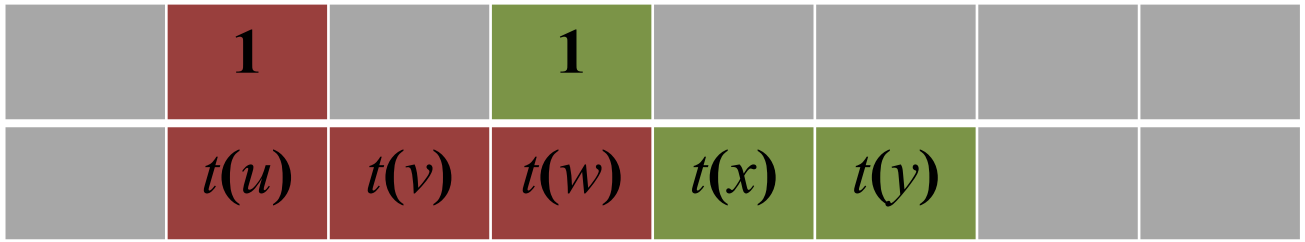
- $b(x)$ = location in the hash table
- $t(x)$ = tag stored in the hash table

Collisions in the hash table?

- Linear probing.
- $O(1)$ metadata bits per slot.

Resolving collisions in the QF

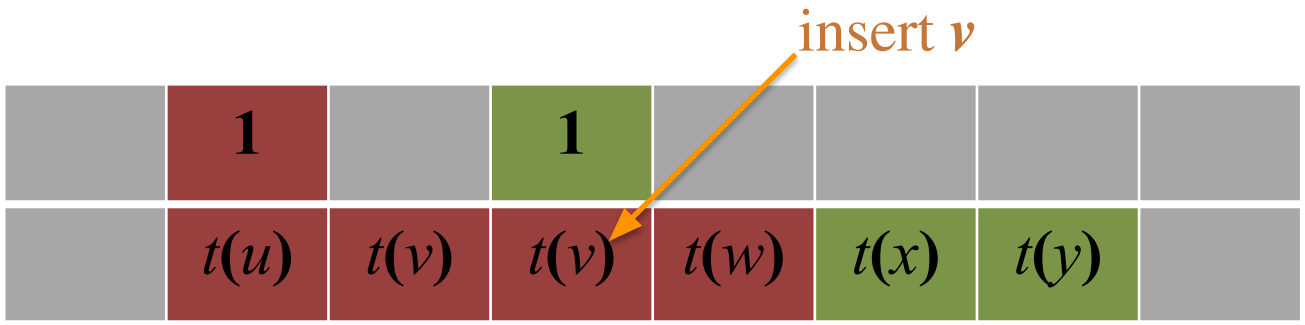
- CQF uses two metadata bits to resolve collisions and identify the home bucket.



- The metadata bits group tags by their home bucket.

Resolving collisions in the QF

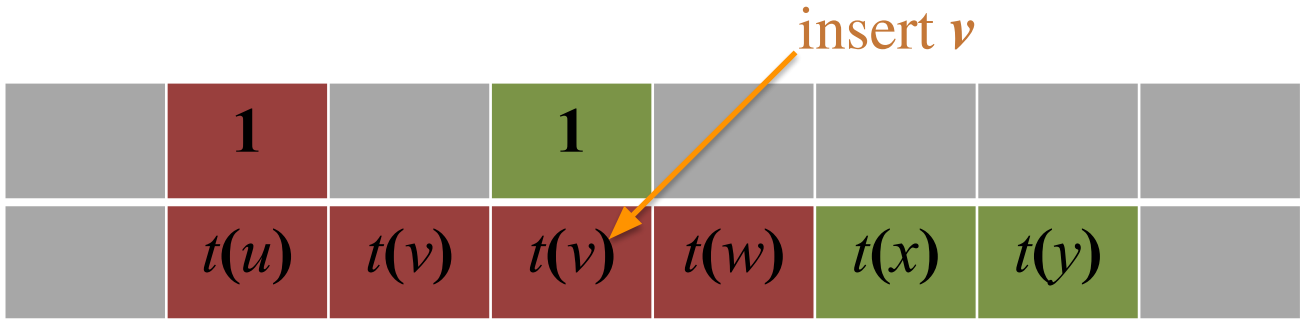
- CQF uses two metadata bits to resolve collisions and identify the home bucket.



- The metadata bits group tags by their home bucket.

Resolving collisions in the QF

- CQF uses two metadata bits to resolve collisions and identify the home bucket.



- The metadata bits group tags by their home bucket.

The metadata bits enable us to identify the slots holding the contents of each bucket.

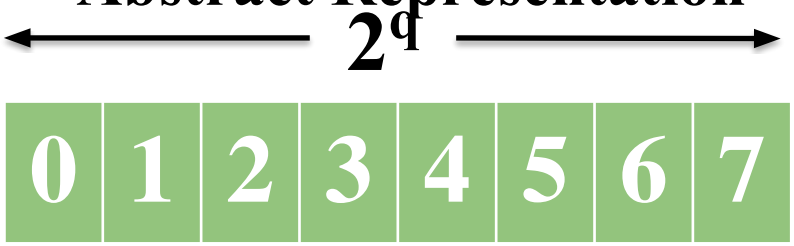
Quotient filter (QF)

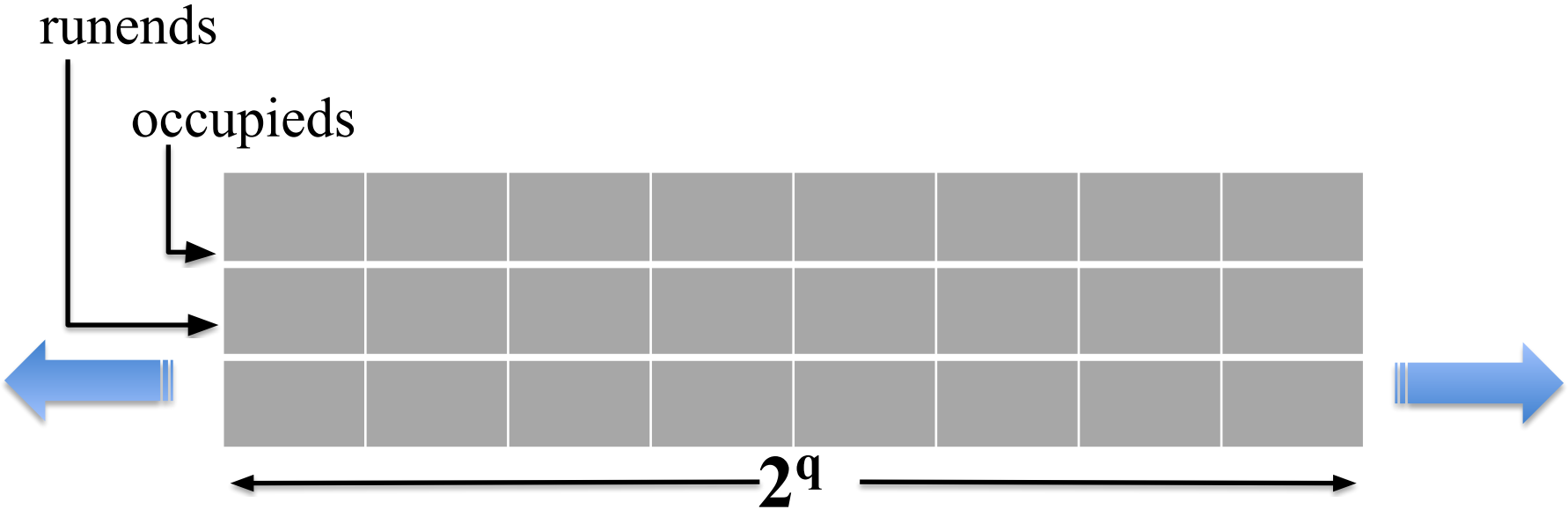
Implementation:

2 Meta-bits per slot.

h(x) --> h_0(x) || h_1(x)

Abstract Representation



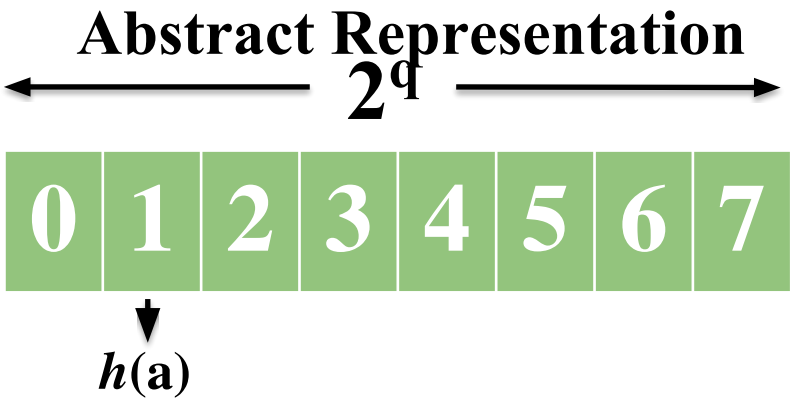


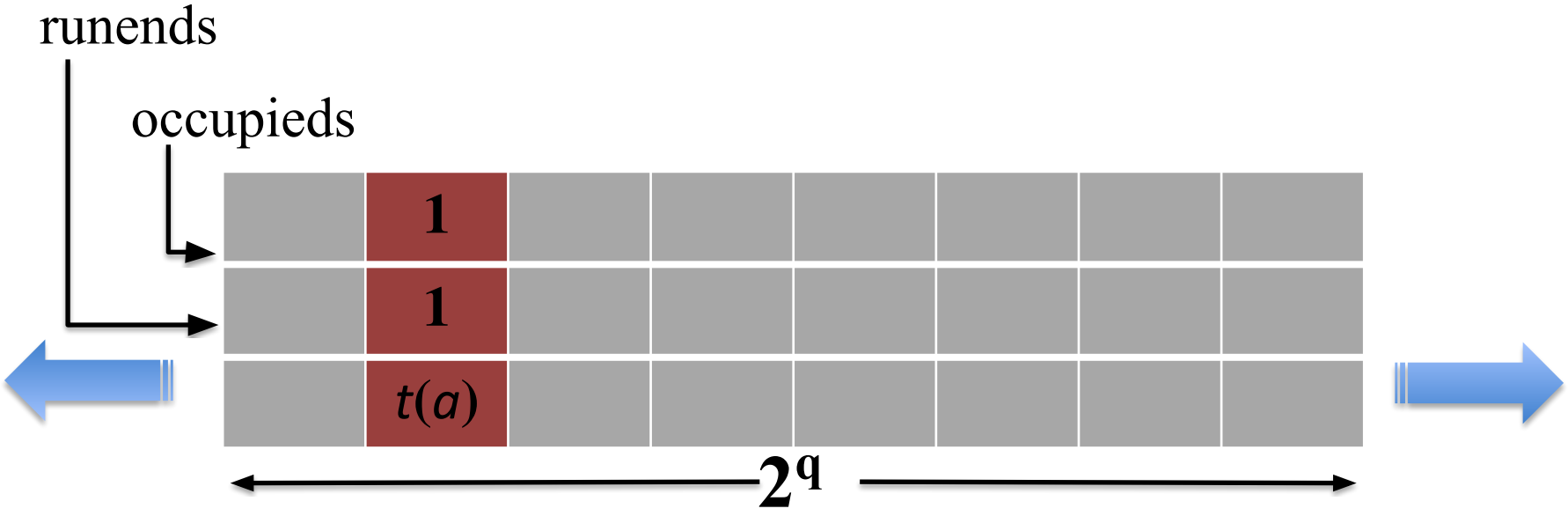
Quotient filter (QF)

Implementation:

2 Meta-bits per slot.

h(x) --> h_0(x) || h_1(x)



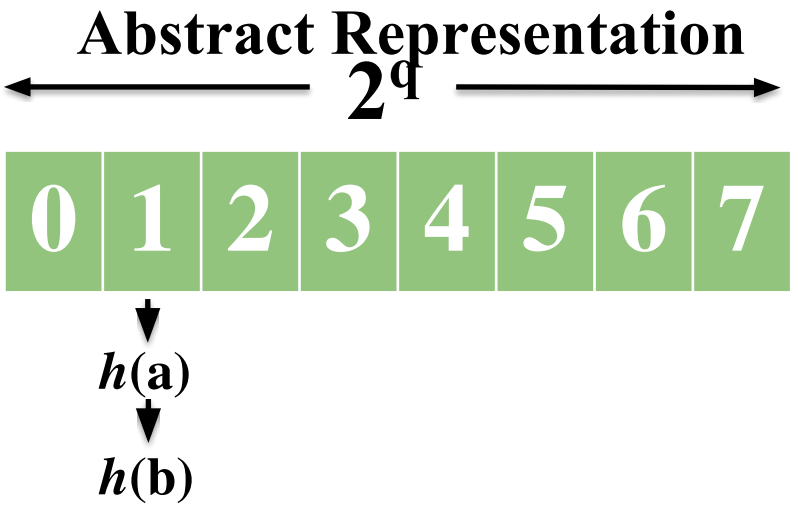


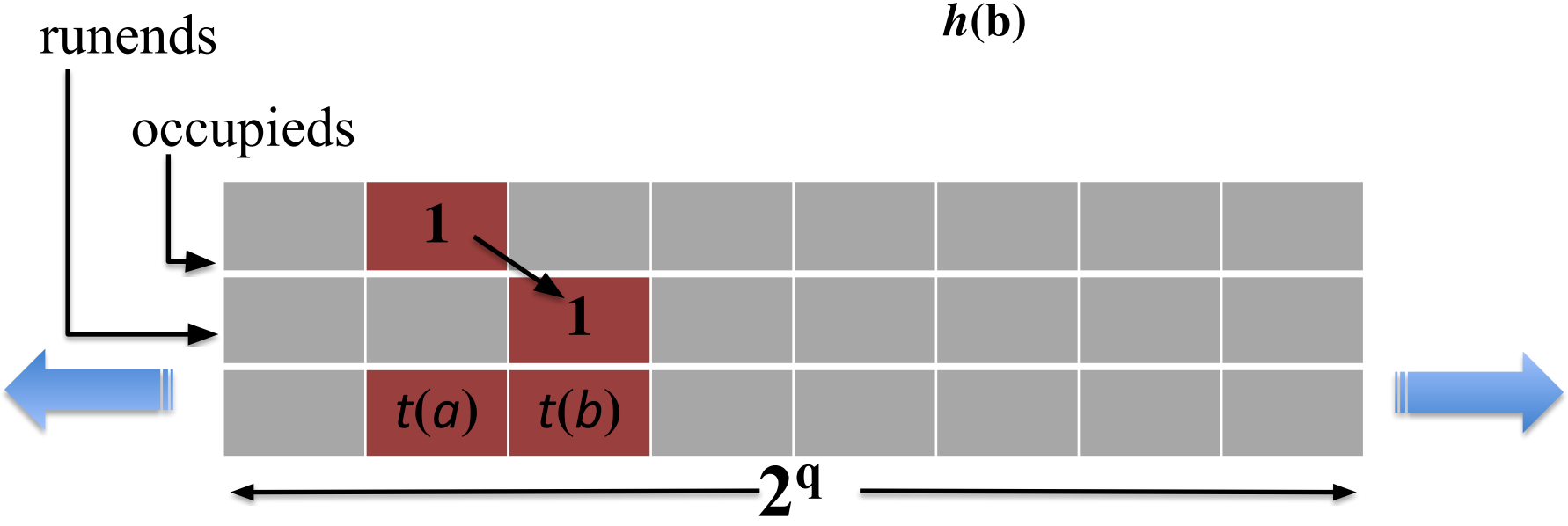
Quotient filter (QF)

Implementation:

2 Meta-bits per slot.

h(x) --> h_0(x) || h_1(x)



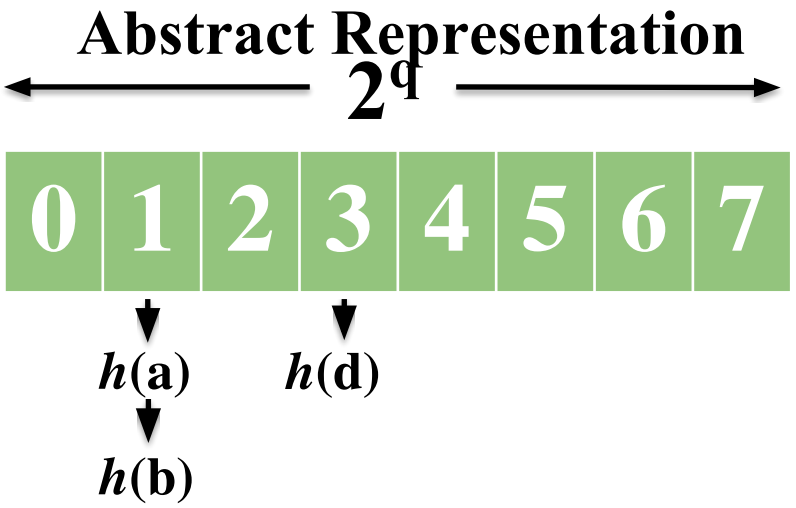


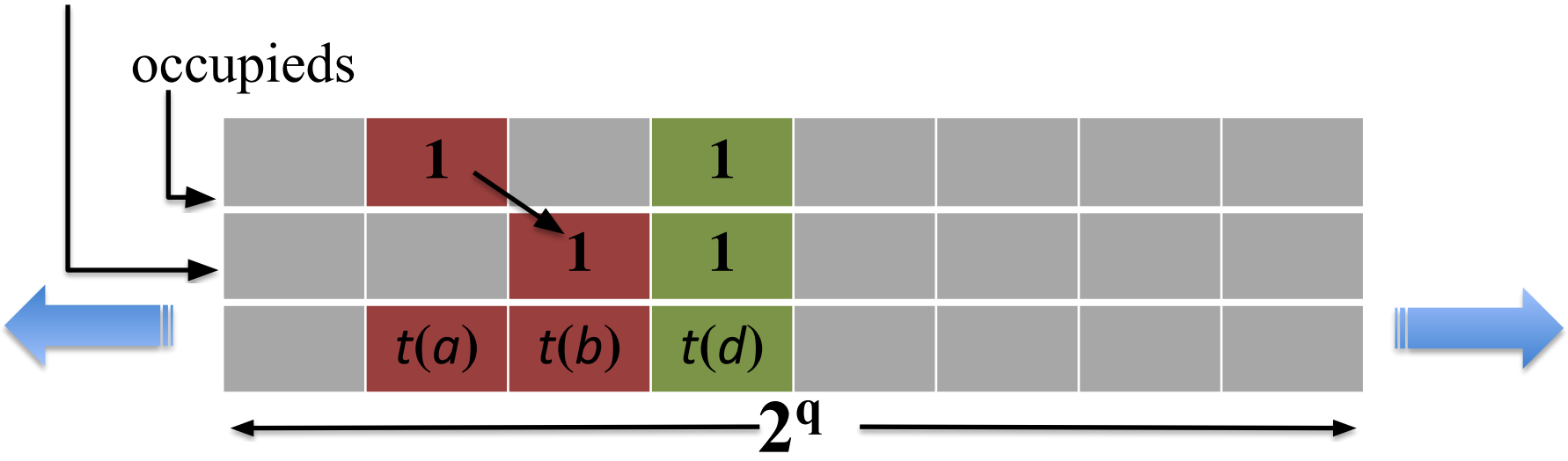
Quotient filter (QF)

Implementation:

2 Meta-bits per slot.

h(x) --> h_0(x) || h_1(x)



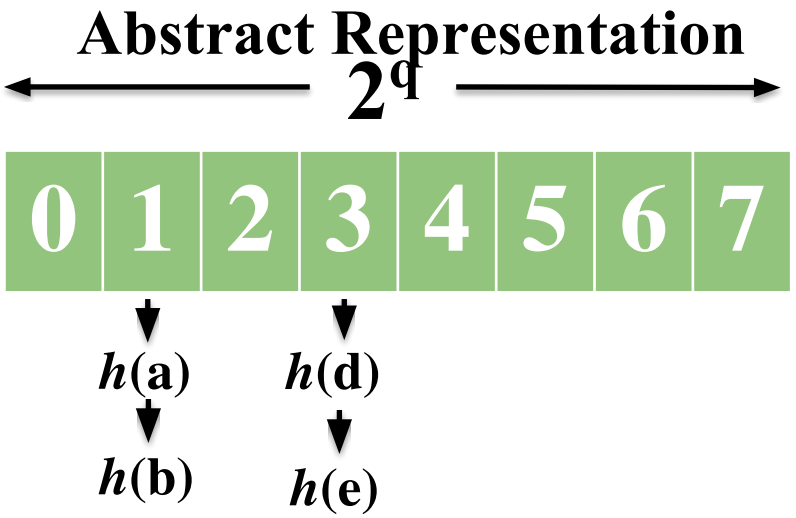


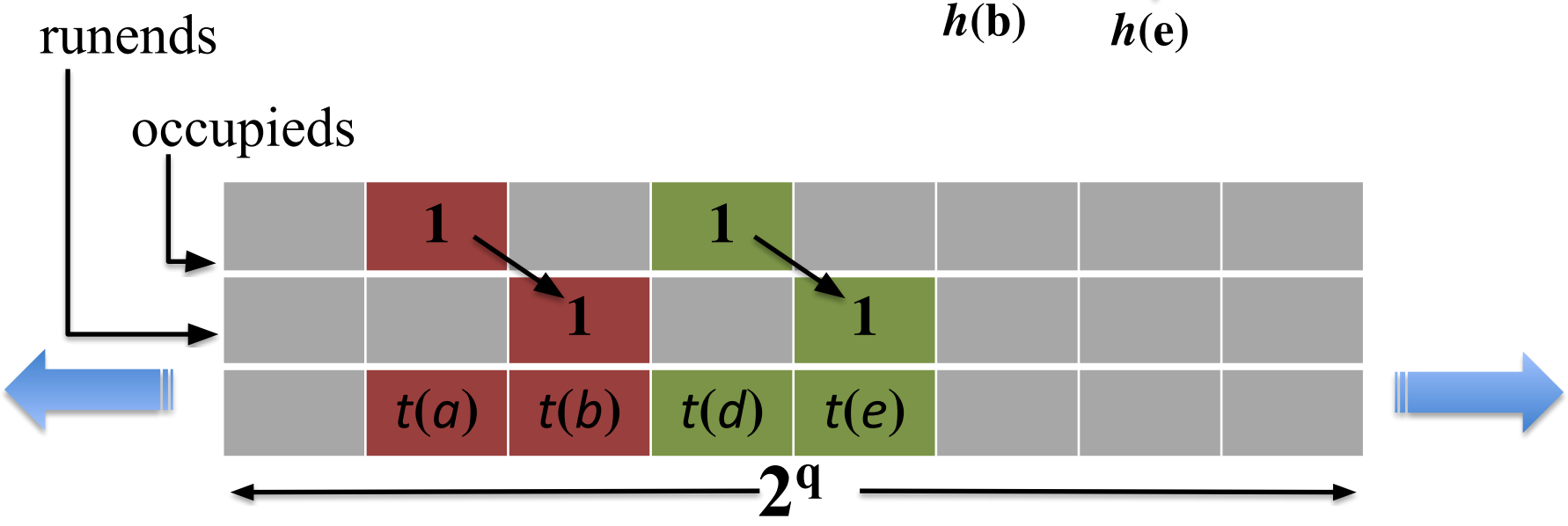
Quotient filter (QF)

Implementation:

2 Meta-bits per slot.

h(x) --> h_0(x) || h_1(x)



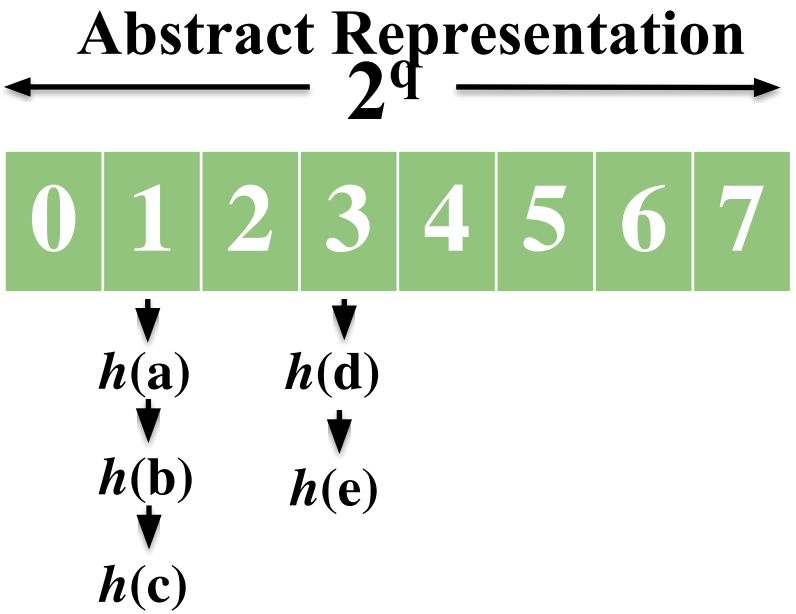


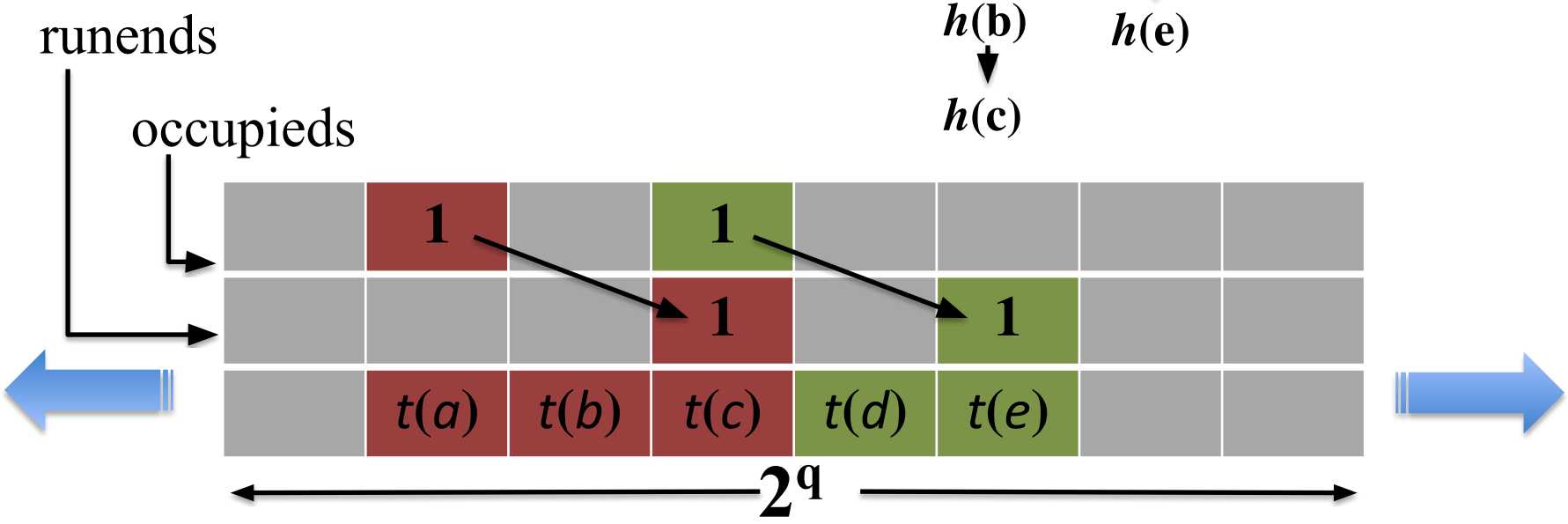
Quotient filter (QF)

Implementation:

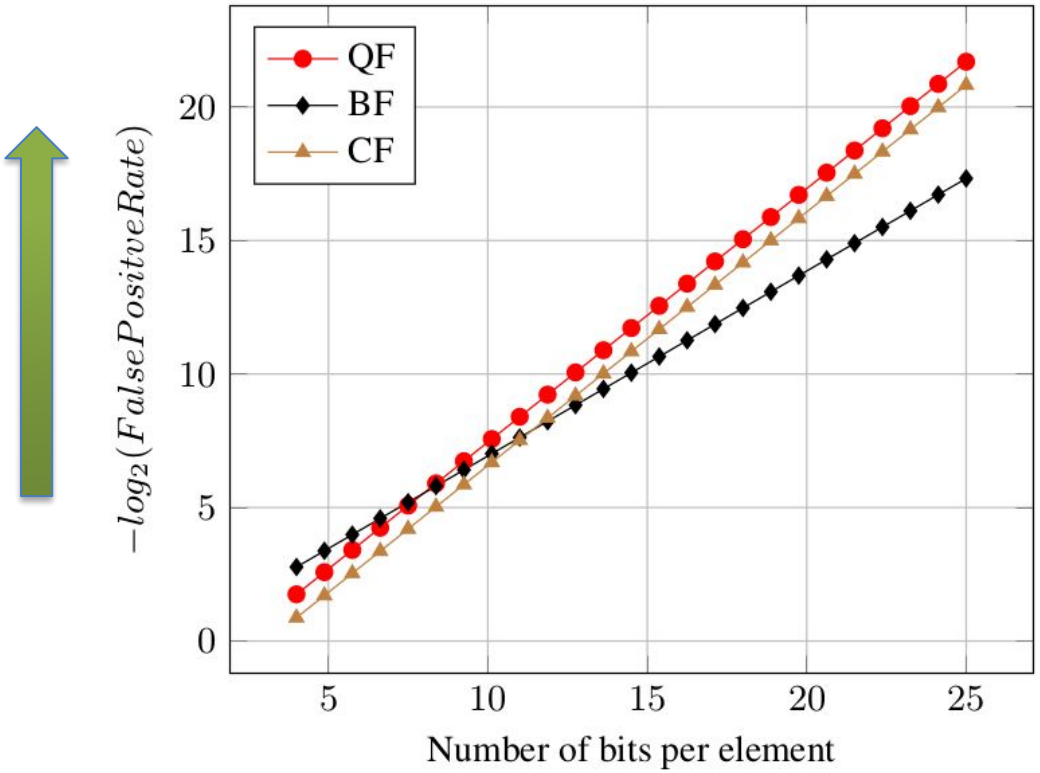
2 Meta-bits per slot.

h(x) --> h_0(x) || h_1(x)



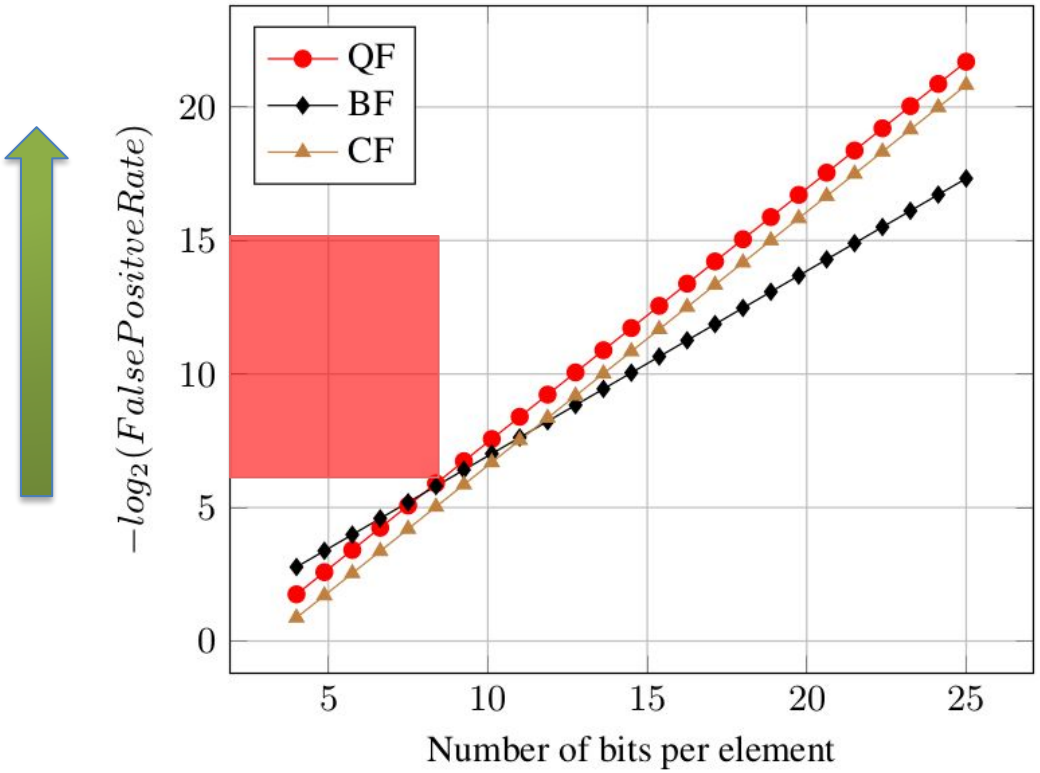


Quotient filters (QF) use comparable space to Bloom filters (BF)



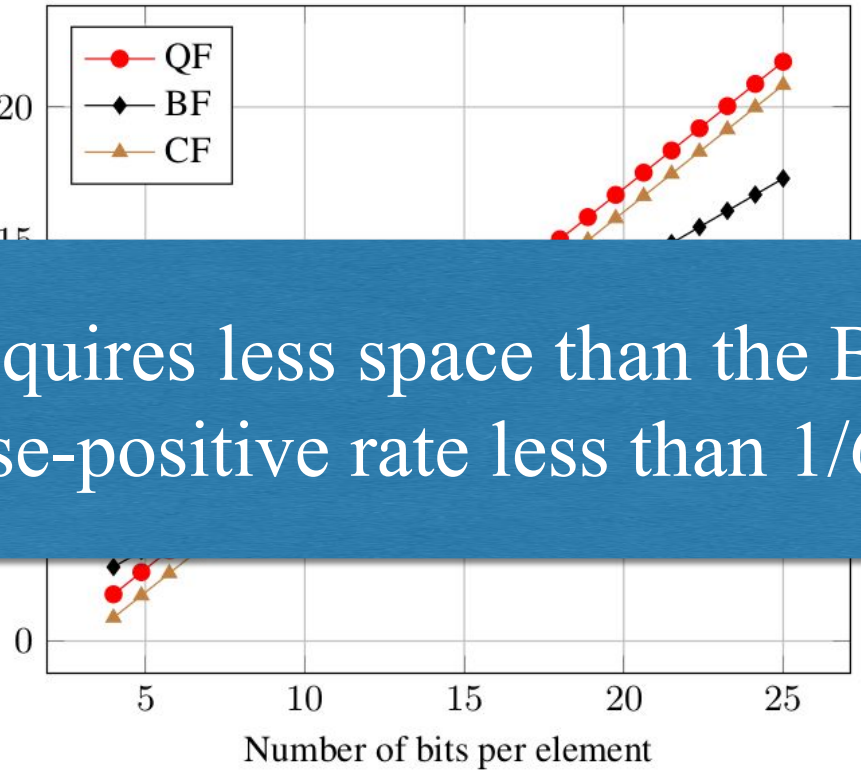
Bloom filters: ~$1.44log_2(1/\varepsilon)$ bits/element.
Quotient filters: ~$2.125 + log_2(1/\varepsilon)$ bits/element.

Quotient filters (QF) use comparable space to Bloom filters (BF)



Bloom filters: ~$1.44log_2(1/\varepsilon)$ bits/element.
Quotient filters: ~$2.125 + log_2(1/\varepsilon)$ bits/element.

Quotient filters (QF) use comparable space to Bloom filters (BF)

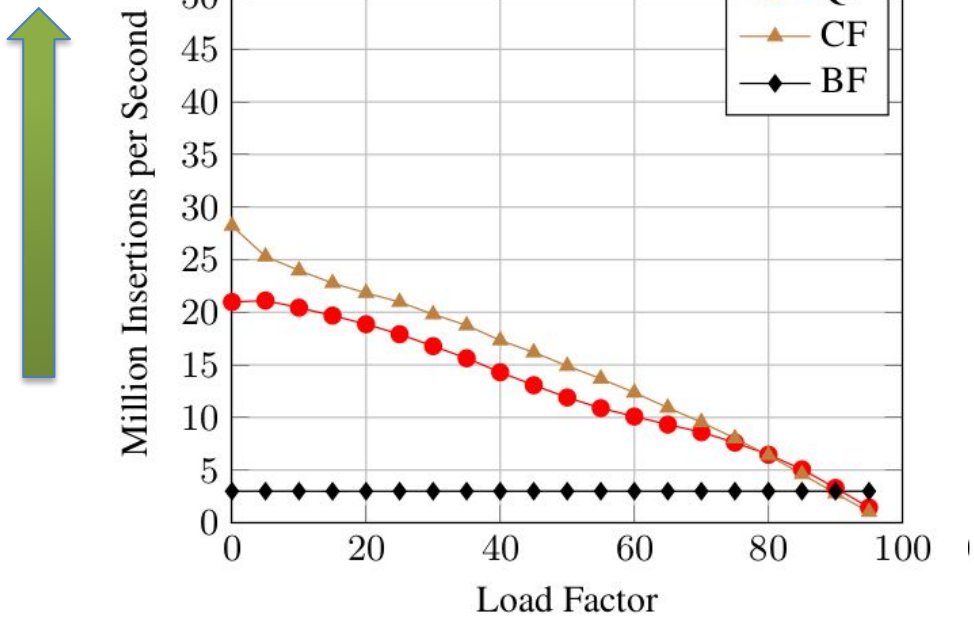


The QF requires less space than the BF for any false-positive rate less than 1/64

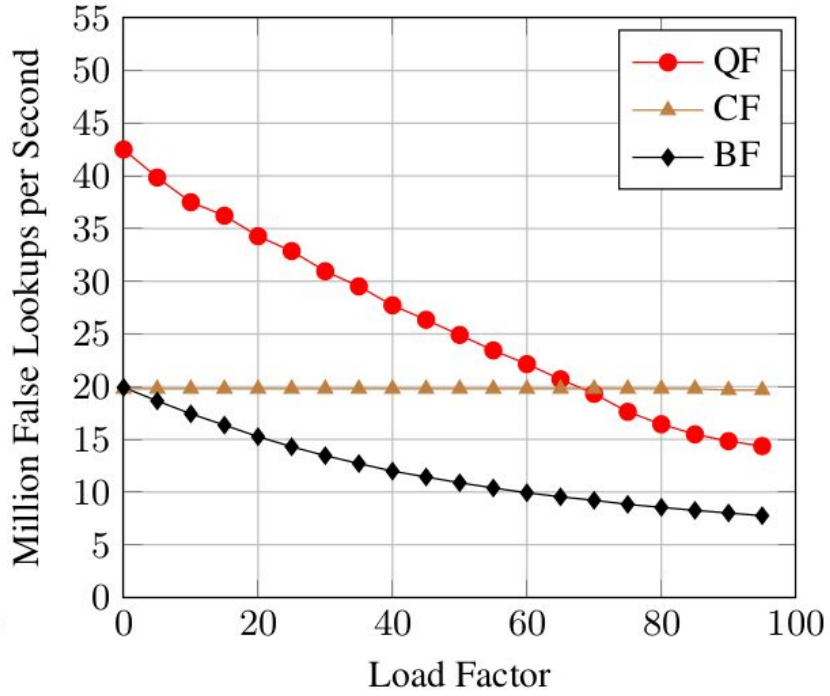
Bloom filters: $\sim 1.44 \log_2(1/\varepsilon)$ bits/element.
Quotient filters: $\sim 2.125 + \log_2(1/\varepsilon)$ bits/element.

Performance: In memory

Inserts



Lookups

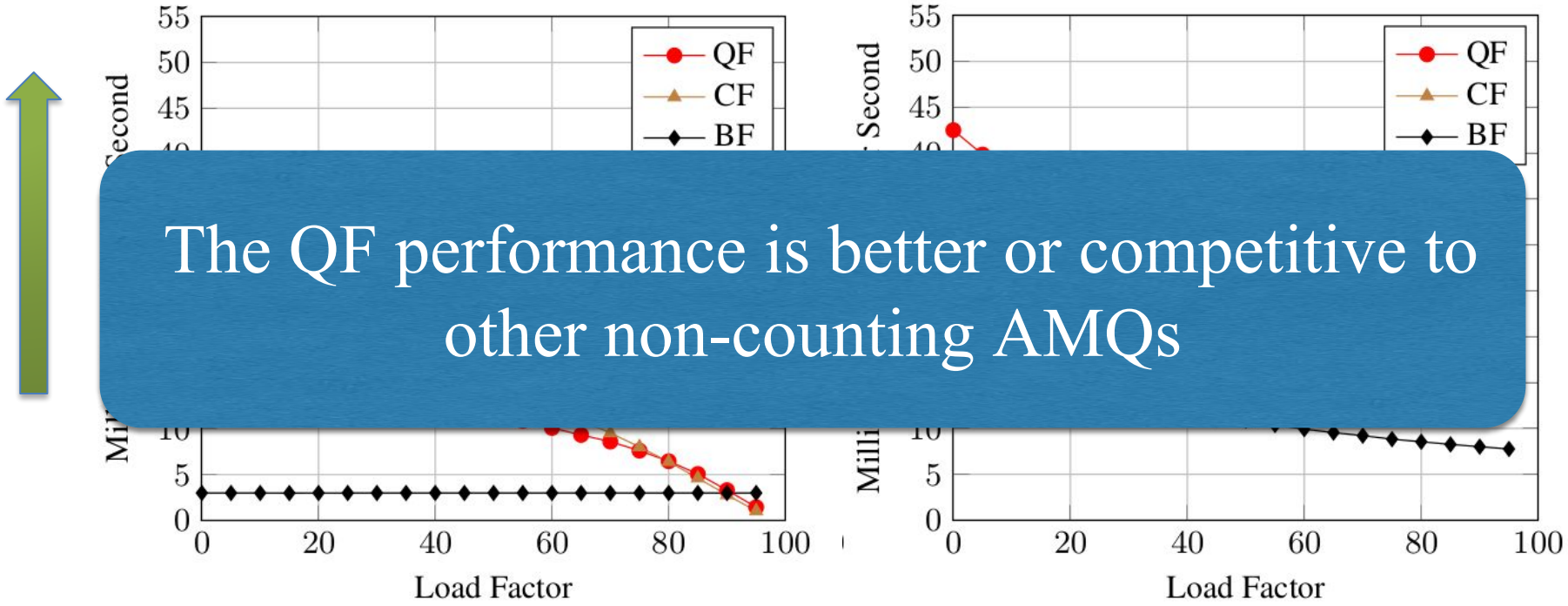


- The QF insert performance in RAM is similar to that of the state-of-the-art non-counting AMQ.
- The QF query performance is significantly fast at low load-factors and slightly slower at higher load-factors.

Performance: In memory

Inserts

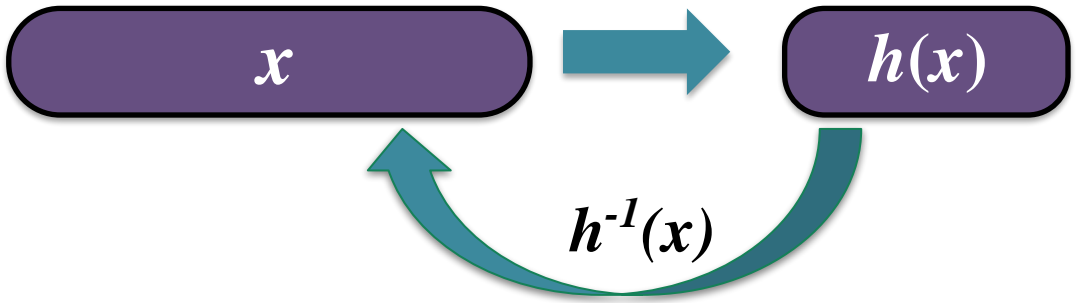
Lookups



- The QF insert performance in RAM is similar to that of the state-of-the-art non-counting AMQ.
- The QF query performance is significantly fast at low load-factors and slightly slower at higher load-factors.

Quotient filters can also be exact

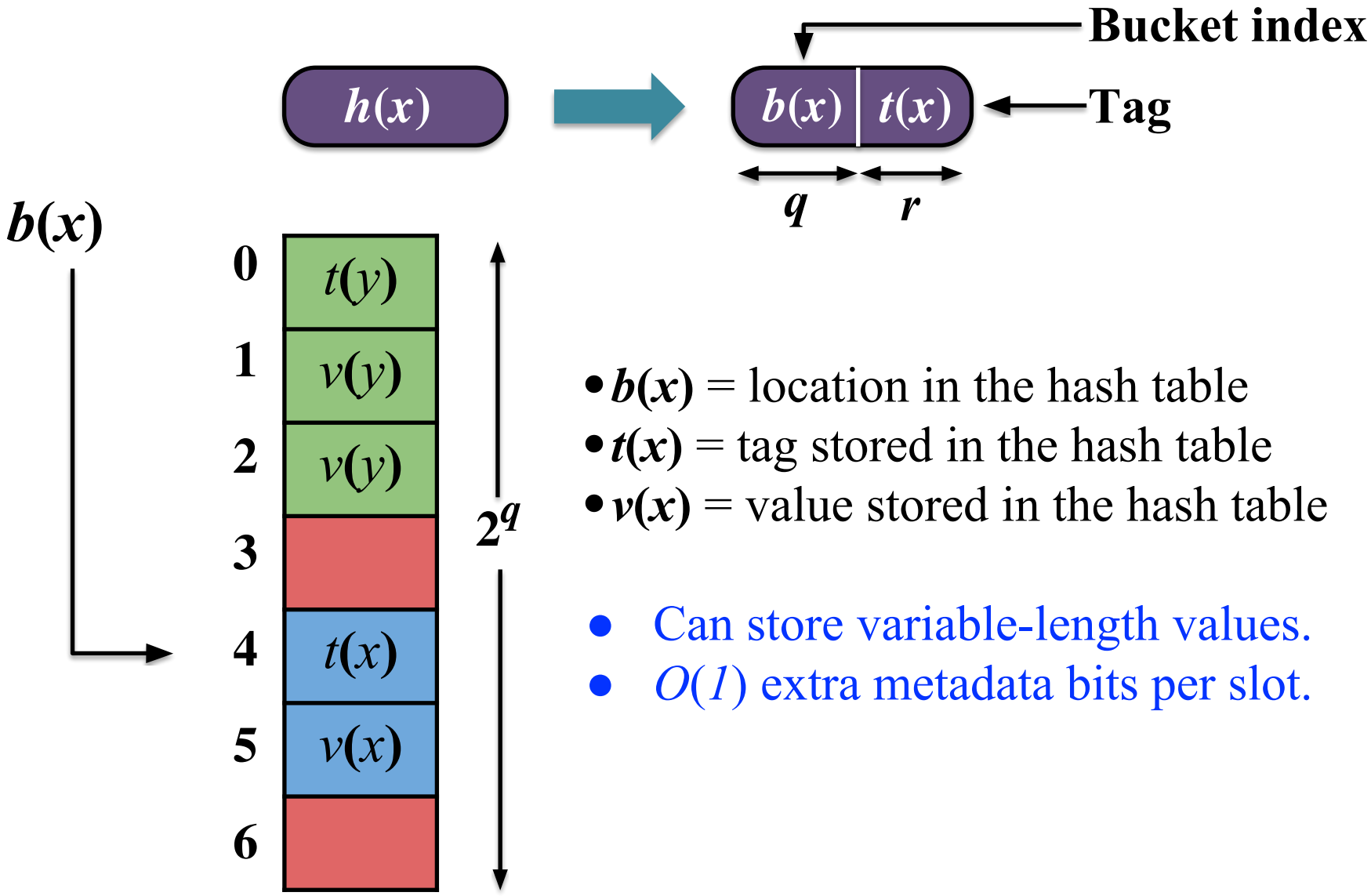
- **Quotient filters store *h(x)* exactly.**
- **To store *x* exactly, use an invertible hash function.**



- **For *n* elements and *p*-bit hash function:**

  Space usage: ~*p*-$log_2 n$ bits/element.

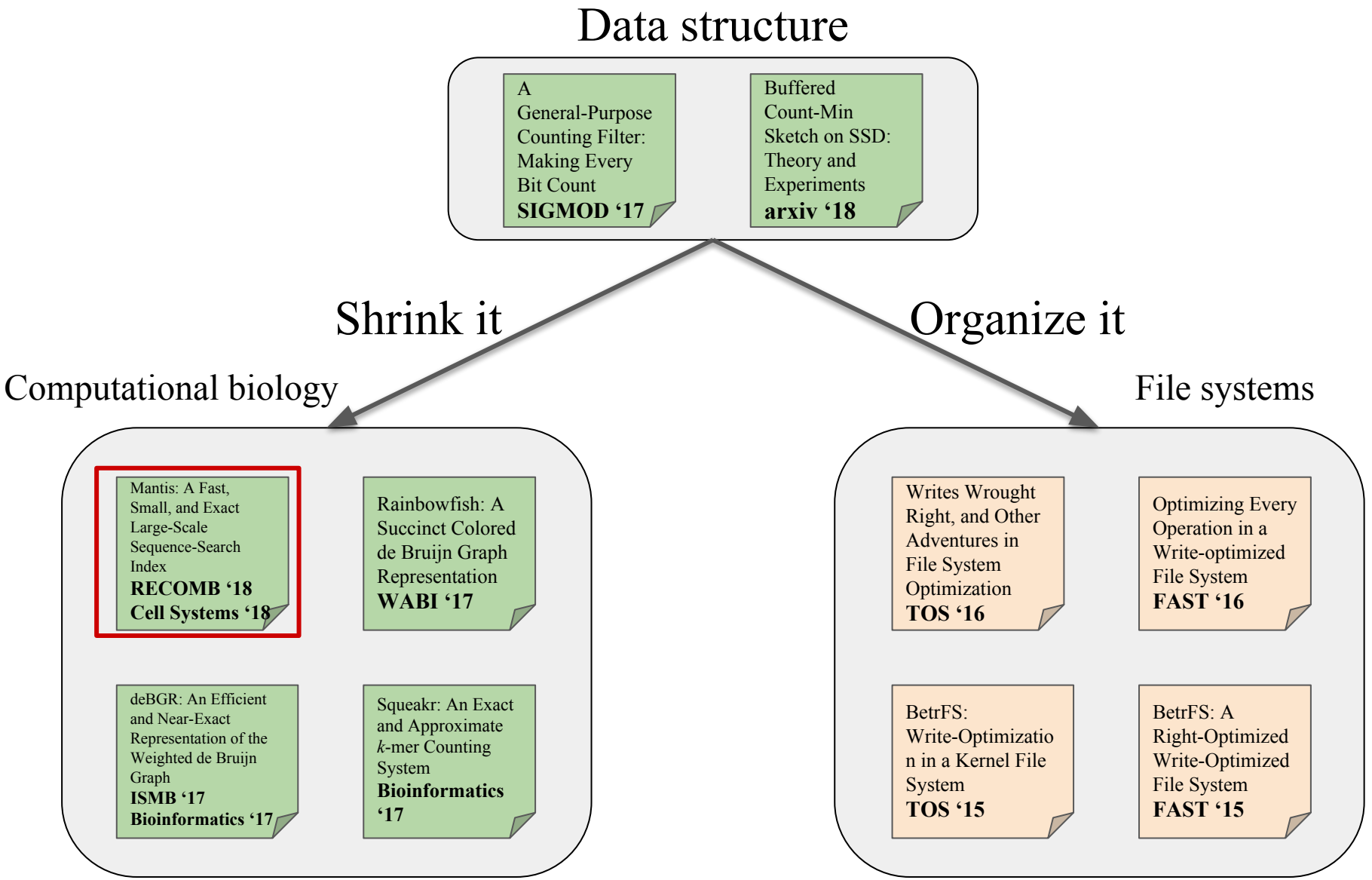
Storing key-value pairs



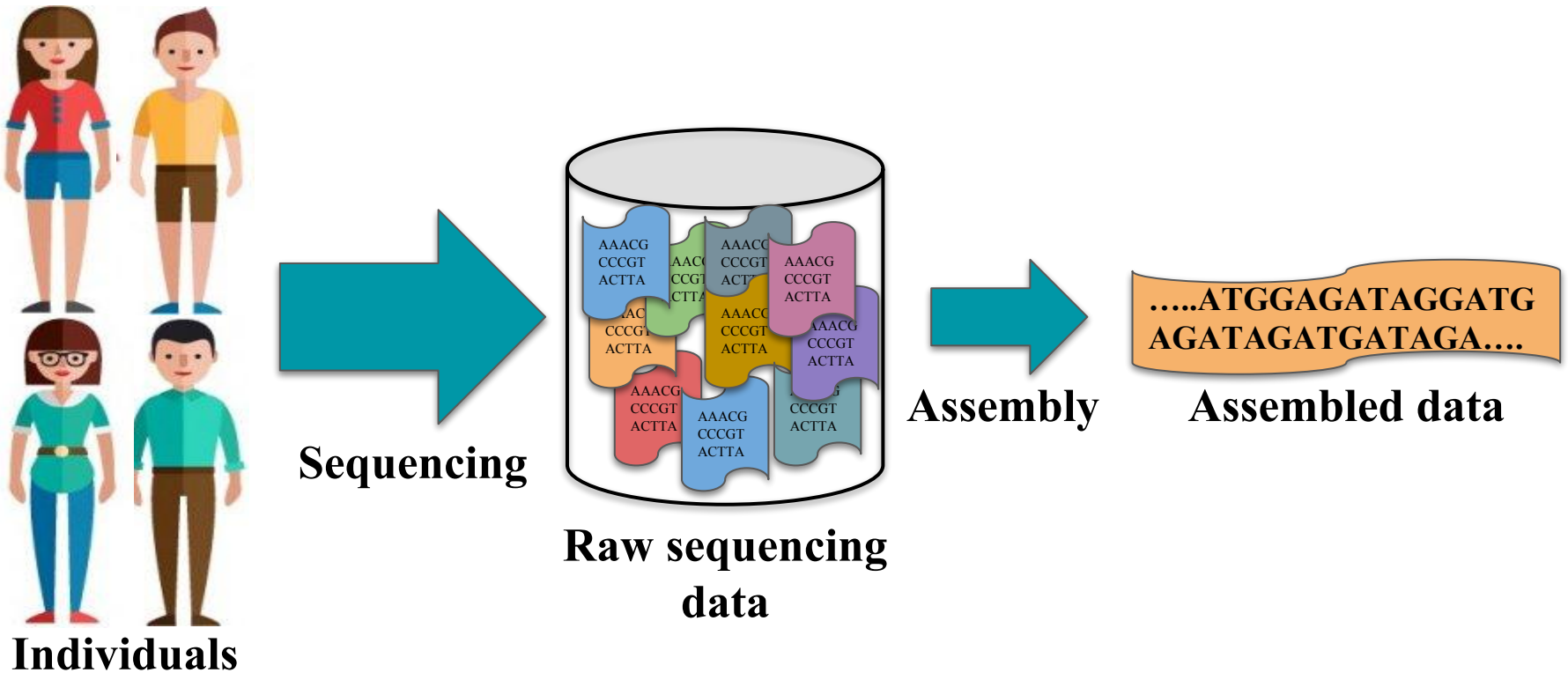
- $b(x)$ = location in the hash table
- $t(x)$ = tag stored in the hash table
- $v(x)$ = value stored in the hash table

- Can store variable-length values.
- $O(1)$ extra metadata bits per slot.

Research output



A huge amount of information is available only in raw sequencing data

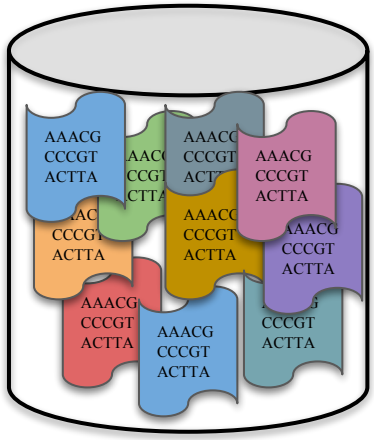


- Assembled data is hugely lossy**.** A lot of **variability information is lost during assembly**.
- And a lot of raw sequencing data never gets assembled at all.

Current tools (i.e., BLAST) can't answer diversity questions easily

Individuals

Sequencing

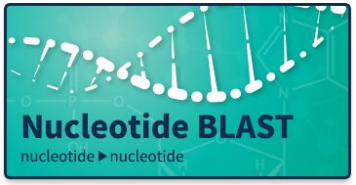


Raw sequencing data

Assembly

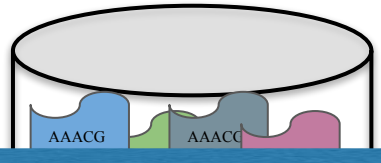
…..ATGGAGATAGGATG
AGATAGATGATAGA….

Assembled data



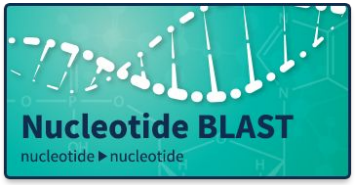
Current tools (i.e., BLAST) can't answer diversity questions easily

Individuals



Raw sequencing data

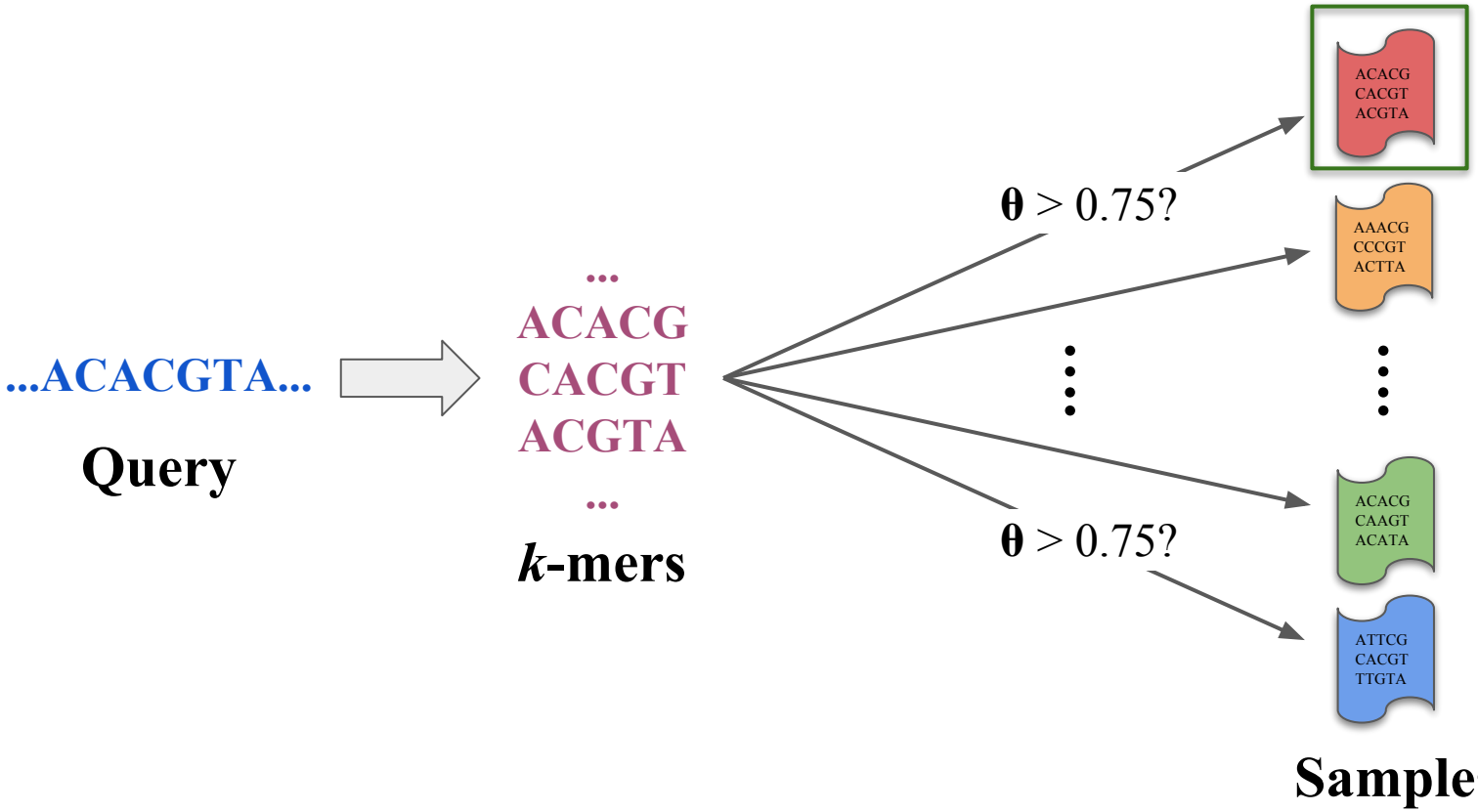
This renders what is otherwise an immensely valuable public resource **largely inert**.



Our Answer: Mantis

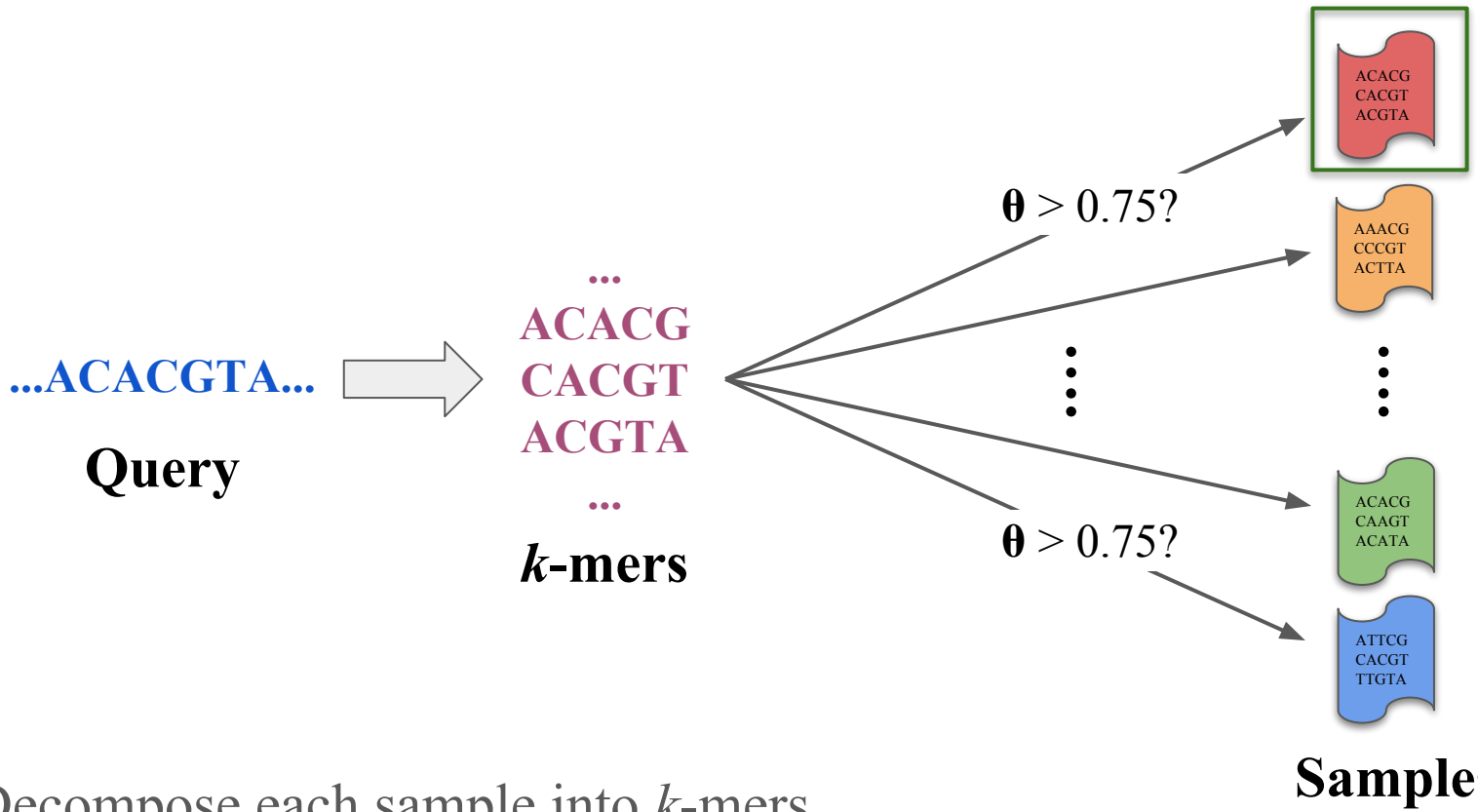
- A system to **index and search through large collections of raw sequencing samples**.
- Mantis uses **new data structures** to enable:
  - **6X faster index construction** than the state-of-the-art.
  - **6X-100X faster searches** than the state-of-the-art.
  - **20% smaller** index size.
  - **Exact results**, i.e., no false-positives or -negatives.
- Mantis is also a **colored de Bruijn graph:**
  - **Fast graph traversal**
  - **Topological analyses**

The sample discovery problem [Solomon and Kingsford]



Return all samples that contain at least some user-defined fraction θ of the k-mers present in the query string.

The sample discovery problem: solution



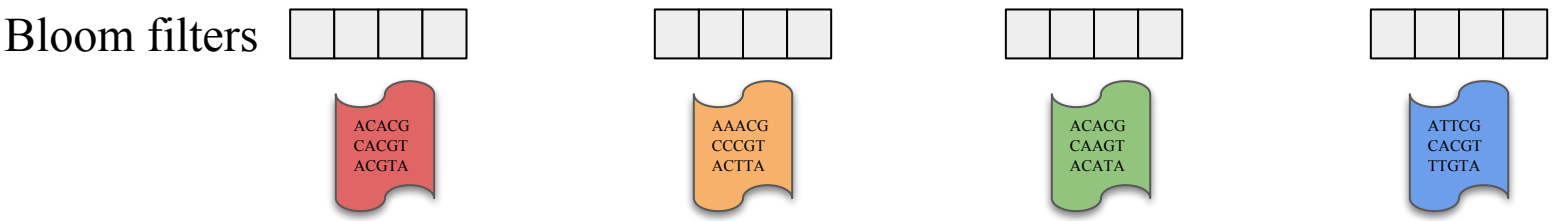
- Decompose each sample into k-mers.
- If more than θ-fraction k-mers from a query appear in an sample then there is a high chance that query appears in that sample.

Existing tools for sample discovery problem

- **SBT:** Solomon and Kingsford 2016
- **SSBT:** Solomon and Kingsford 2017
- **AllSome SBT:** Sun et al. 2017

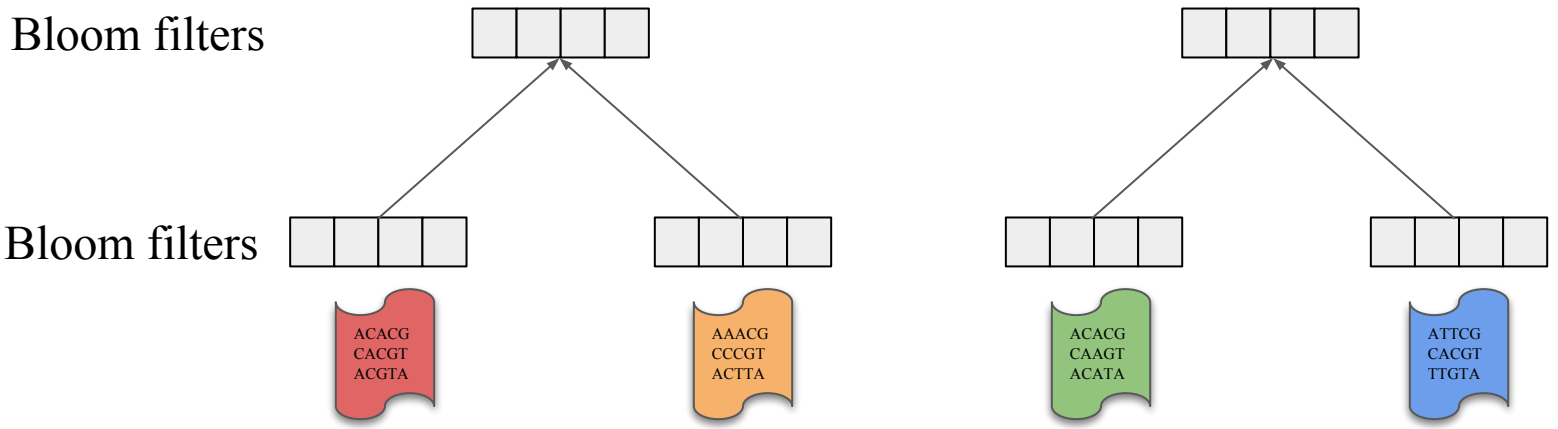
- All these tools use Bloom filters to represent *k*-mer content of samples.
- Using Bloom filter saves a lot of space but results contain false-positives.
- Also, all these tools have to work around the limitations of Bloom filters.

Sequence Bloom Tree (SBT): Construction



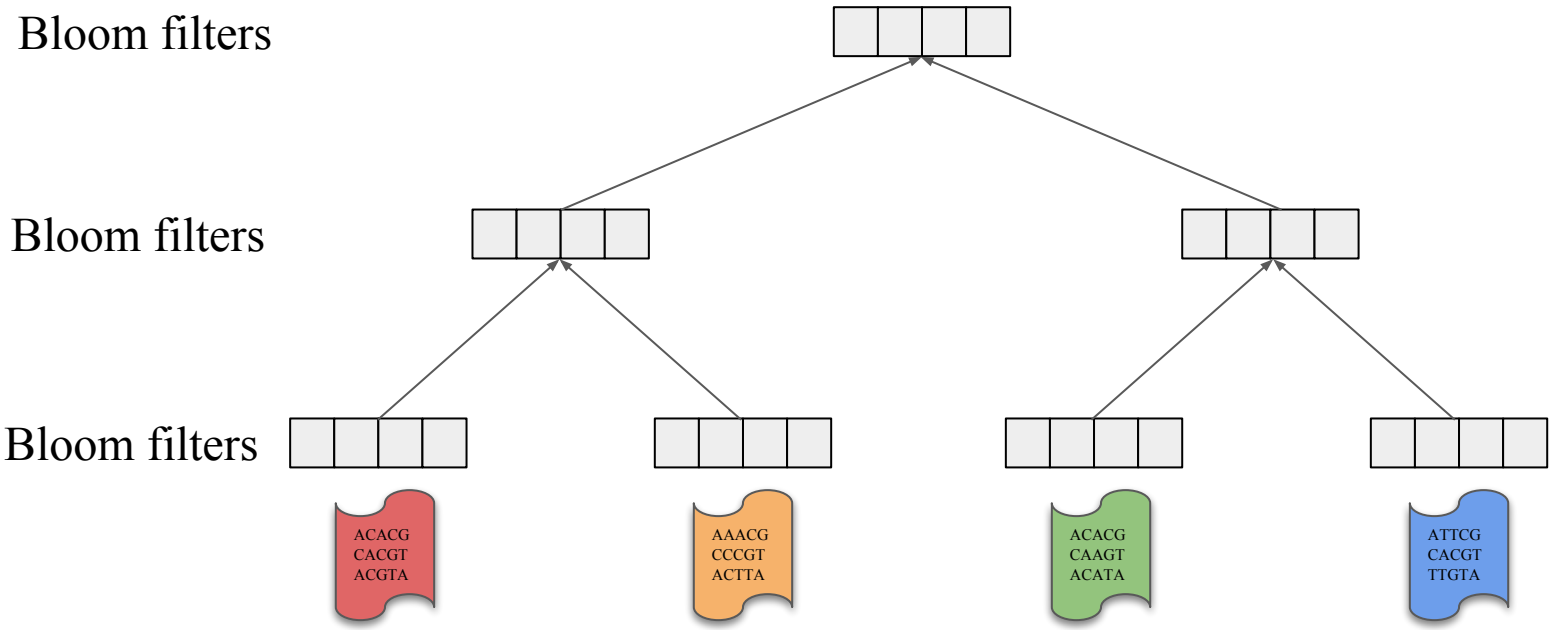
The SBT is a binary tree of Bloom filter where each leaf node corresponds to a sample.

Sequence Bloom Tree (SBT): Construction



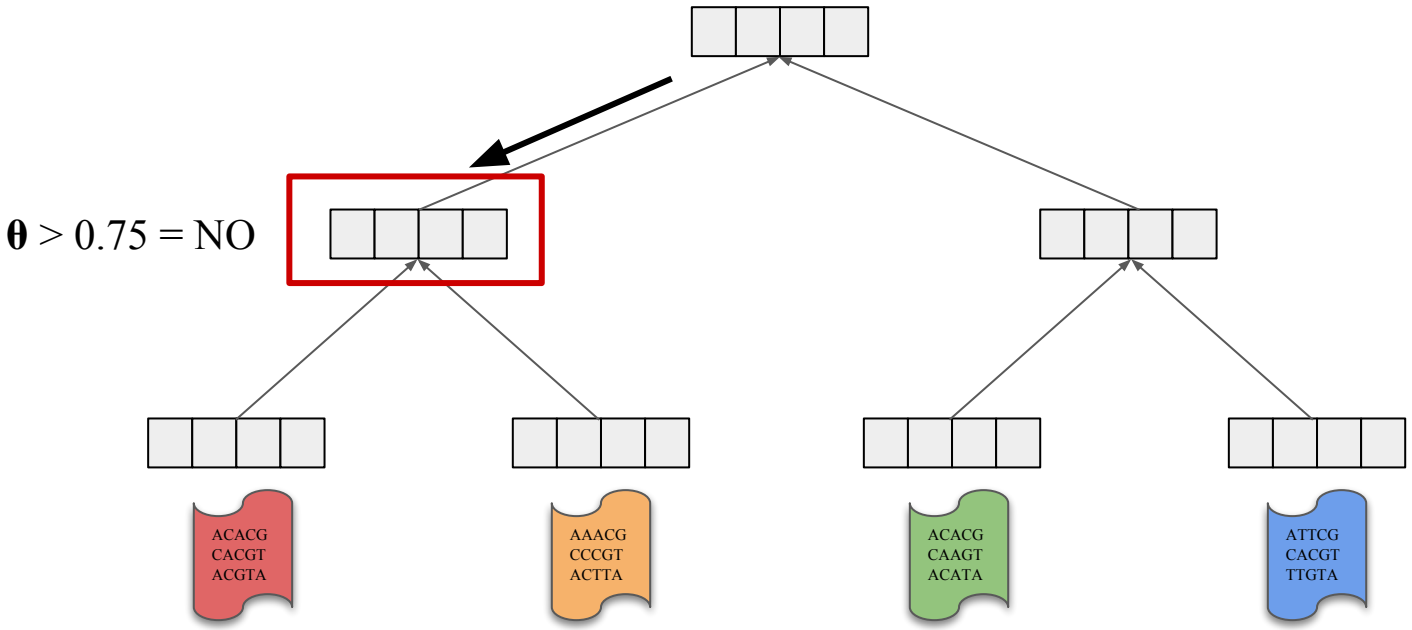
The SBT is a binary tree of Bloom filter where each leaf node corresponds to a sample.

Sequence Bloom Tree (SBT): Construction



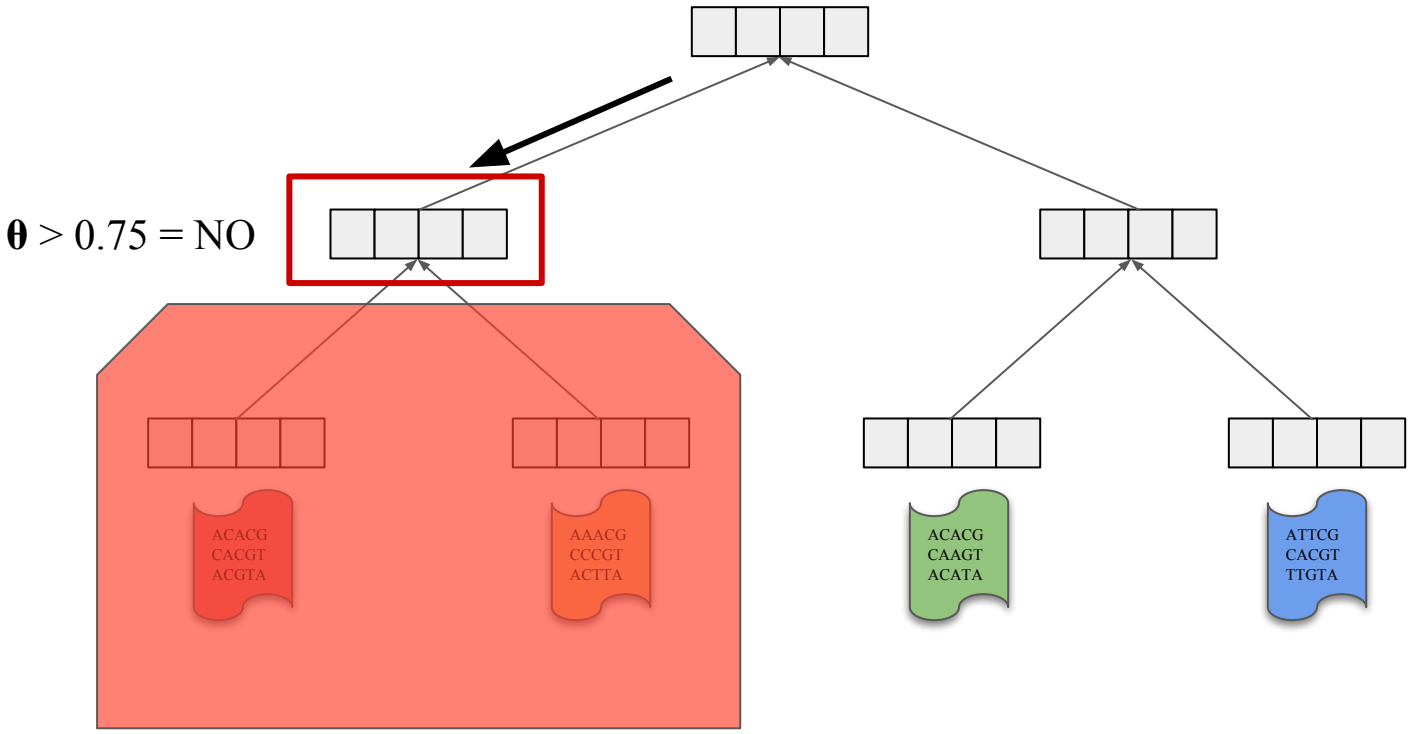
The SBT is a binary tree of Bloom filter where each leaf node corresponds to a sample.

Sequence Bloom Tree (SBT): Query



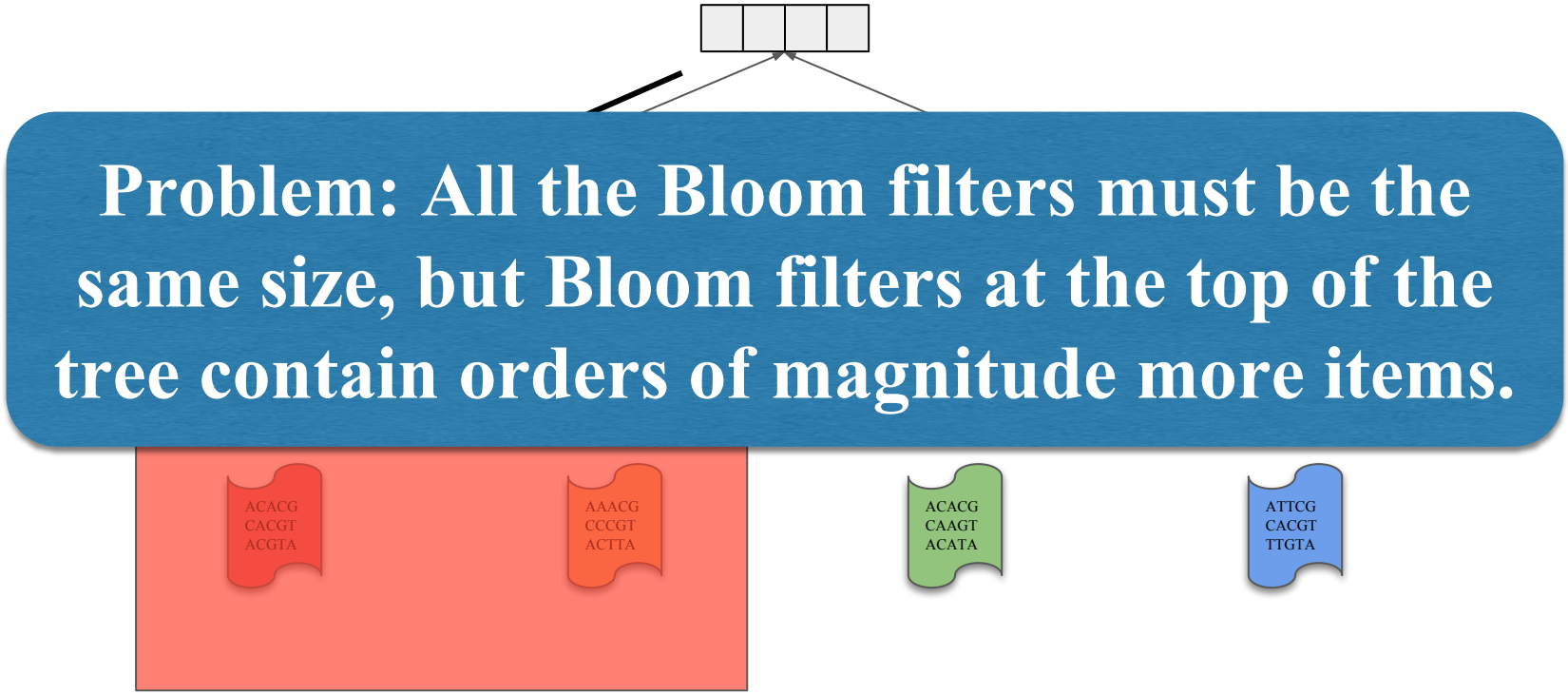
Traverse the subtree if greater than a ***θ***-fraction *k*-mers are present in the node.

Sequence Bloom Tree (SBT): Query



Traverse the subtree if greater than a ***θ***-fraction *k*-mers are present in the node.

Sequence Bloom Tree (SBT): Query



Traverse the subtree if greater than a $\boldsymbol{\theta}$-fraction k-mers are present in the node.

Mantis: A fundamentally different technique

Input Samples

S1	S2	S3	S4
	ACTG	ACTG	
ACTT			
		CTTG	CTTG
	TTTC	TTTC	
	GCGT	GCGT	GCGT
	AGCC	AGCC	

Map: *k*-mers to Samples

k-mer	Samples
ACTG	S2, S3
ACTT	S1
CTTG	S3, S4
TTTC	S2, S3
GCGT	S2, S3, S4
AGCC	S2, S3

- We want to map *k*-mers to the samples in which they appear.

Mantis: A fundamentally different technique

Input Samples

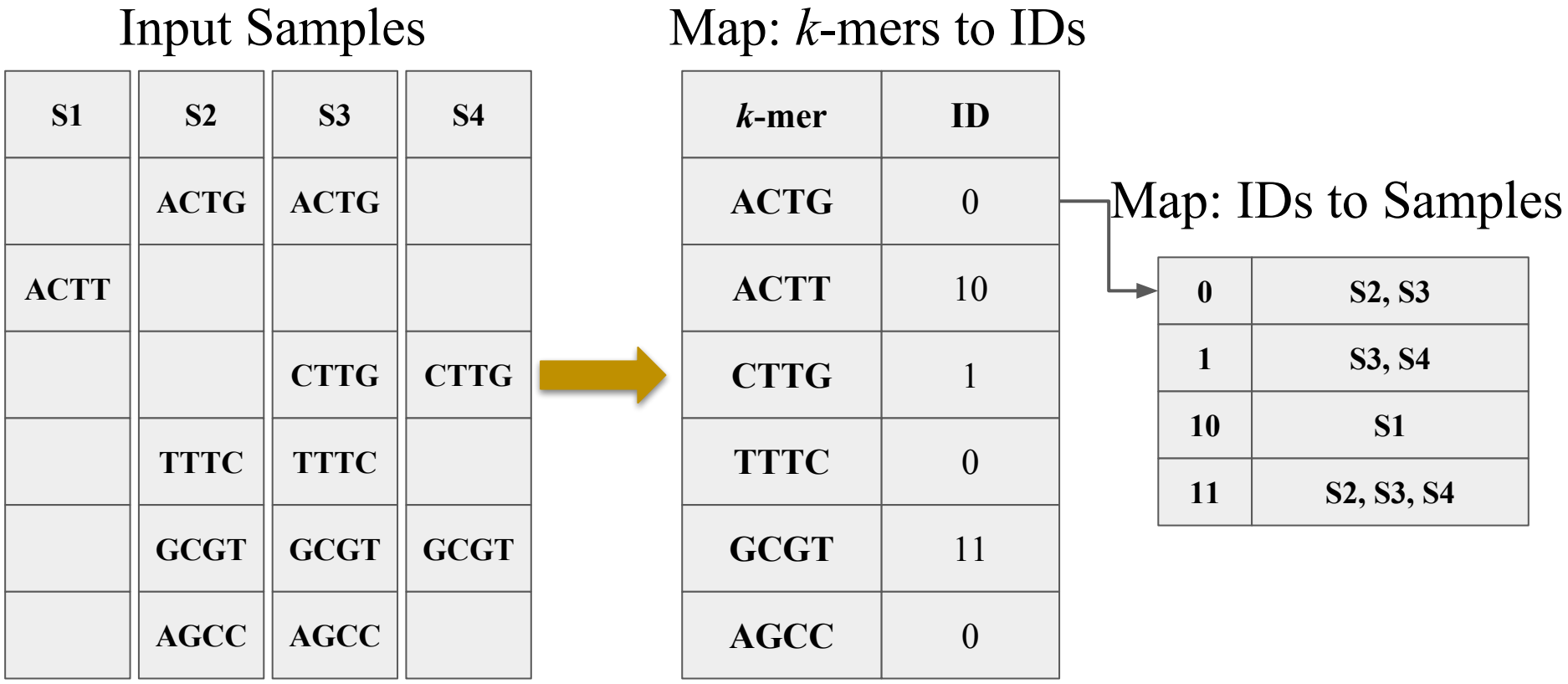
S1	S2	S3	S4
	ACTG	ACTG	
ACTT			
		CTTG	CTTG
	TTTC	TTTC	
	GCGT	GCGT	GCGT
	AGCC	AGCC	

Map: *k*-mers to Samples

k-mer	Samples
ACTG	S2, S3
ACTT	S1
CTTG	S3, S4
TTTC	S2, S3
GCGT	S2, S3, S4
AGCC	S2, S3

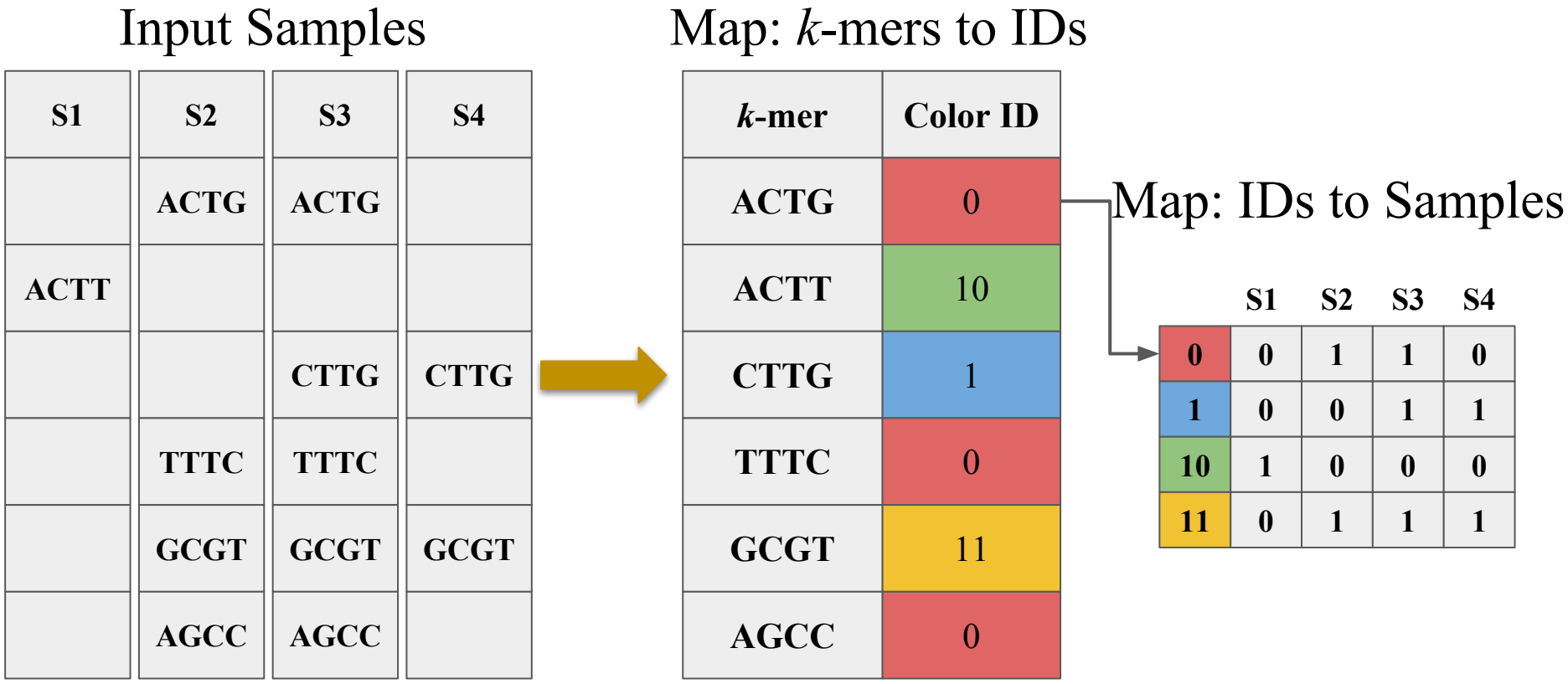
- There is an inherent redundancy in this this design.

Mantis: A fundamentally different technique



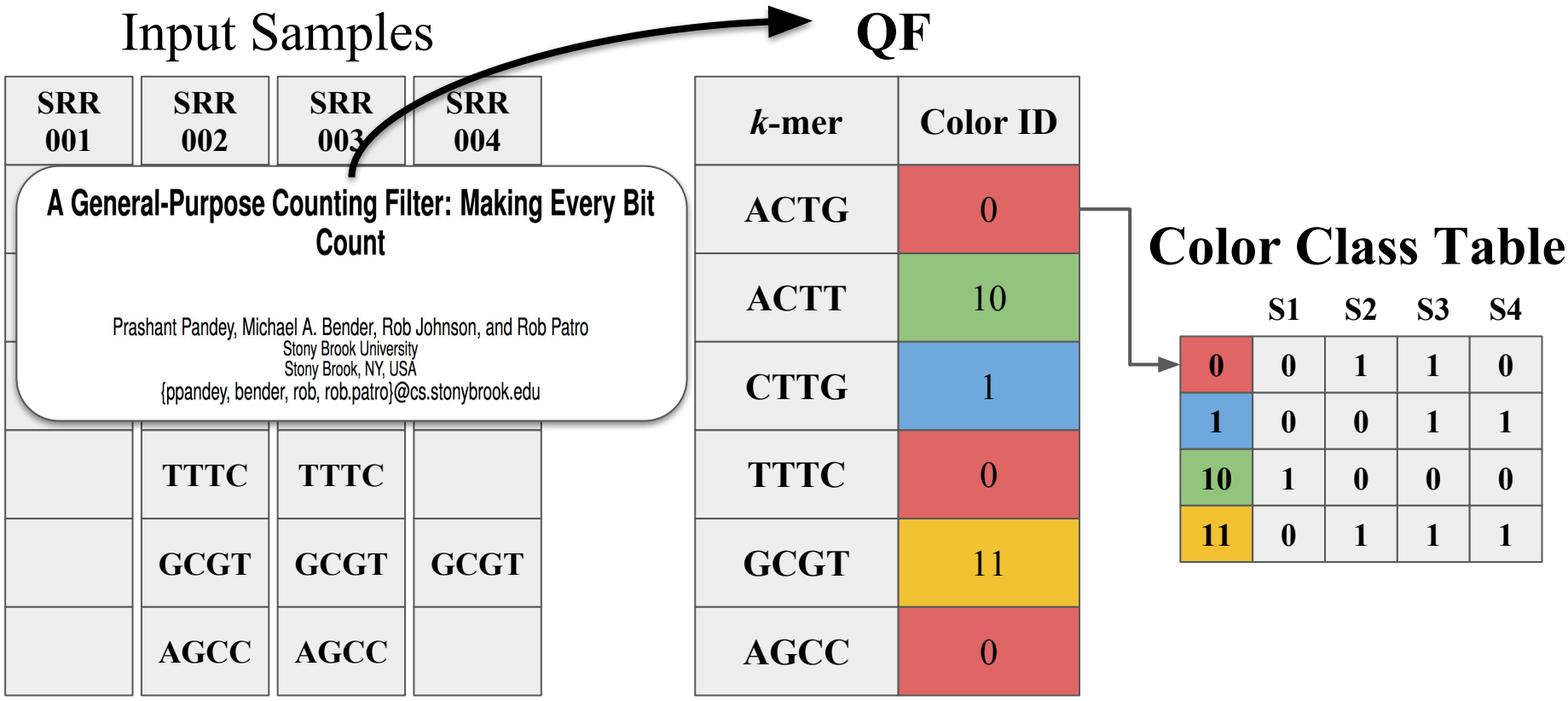
- We add another layer of indirection from IDs to sets of samples.

Mantis: A fundamentally different technique



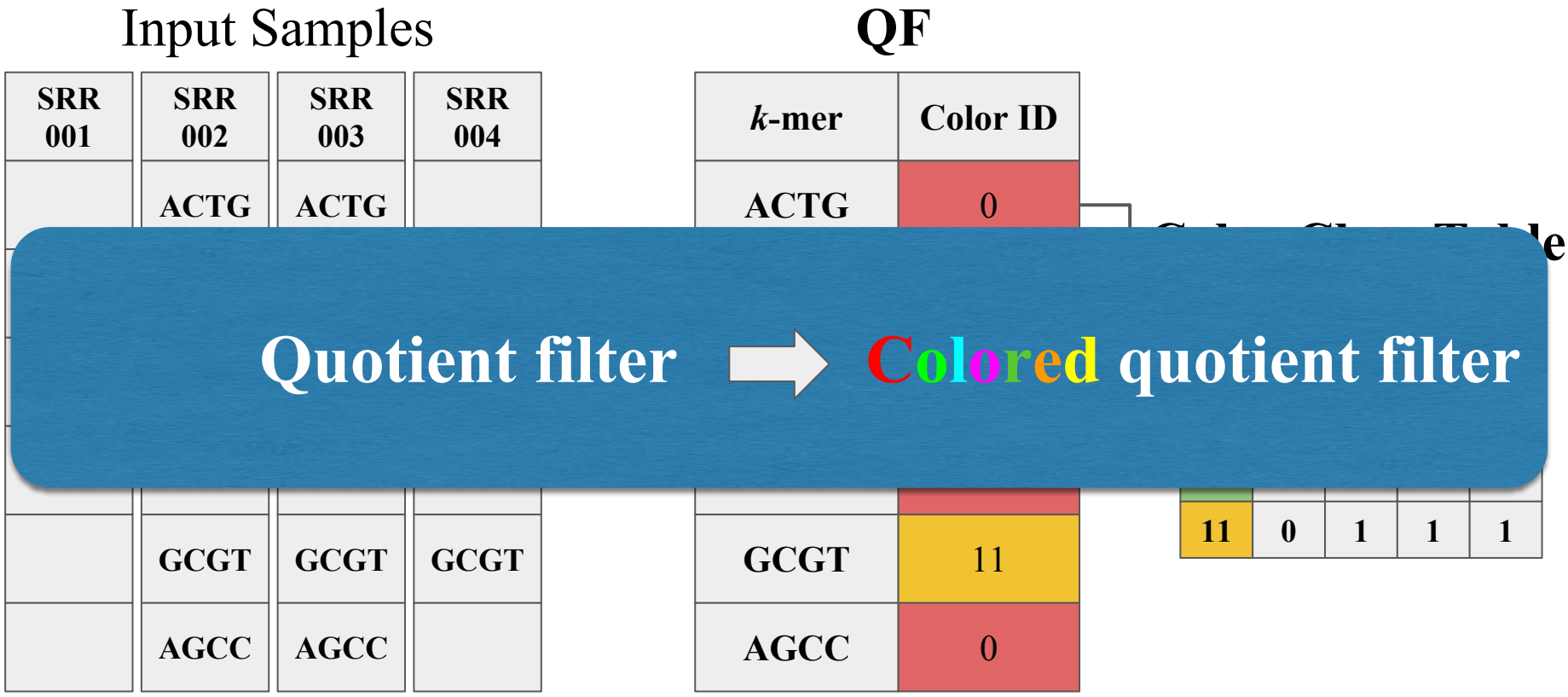
In Mantis, we build a map from *k*-mer to the set of samples in which they appear.

Mantis: A fundamentally different technique



In Mantis, we use the exact version of the QF to map *k*-mers to variable-sized color class IDs.

Mantis: A fundamentally different technique

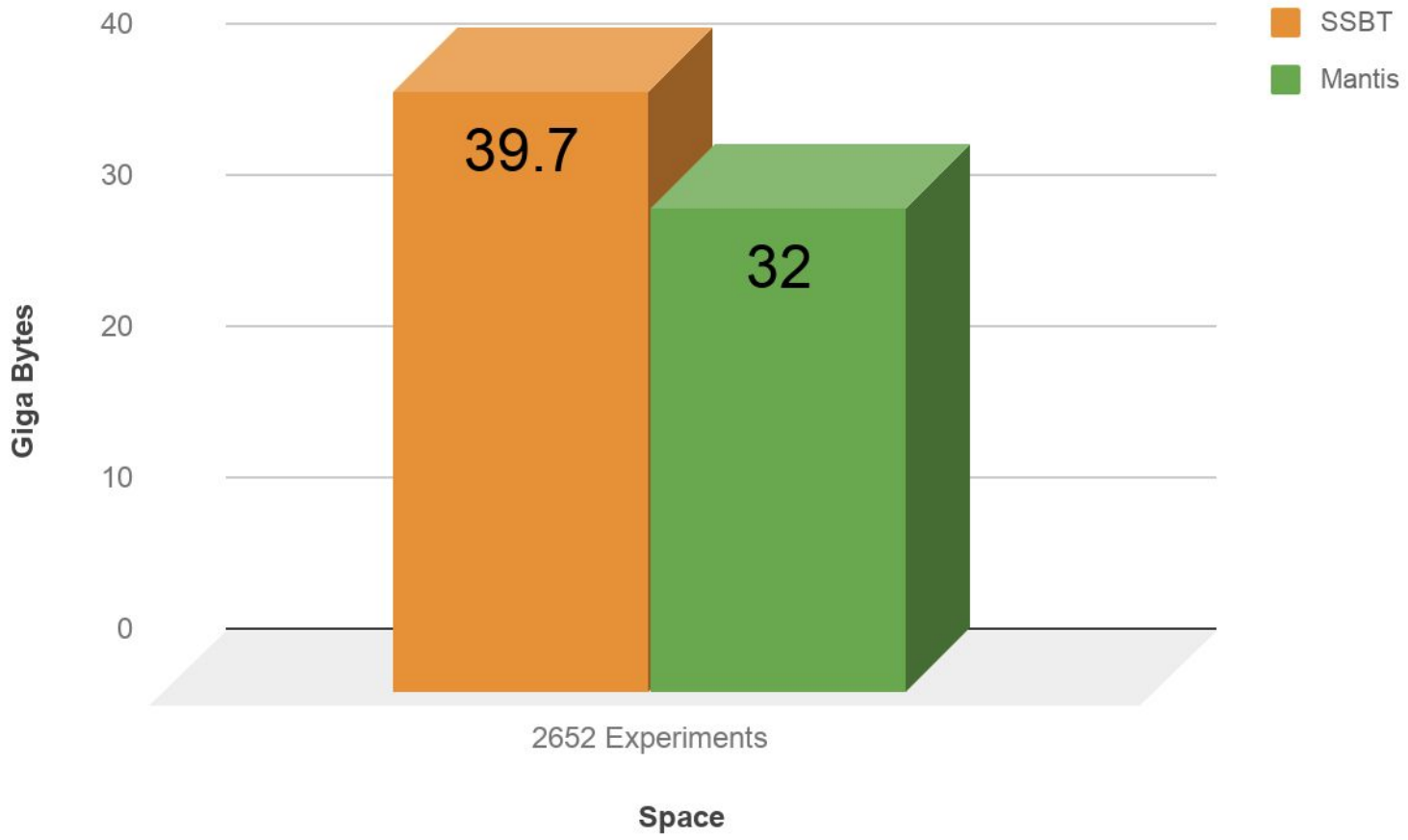


In Mantis, we use the exact version of the QF to map *k*-mers to variable-sized color class IDs.

Experimental setup

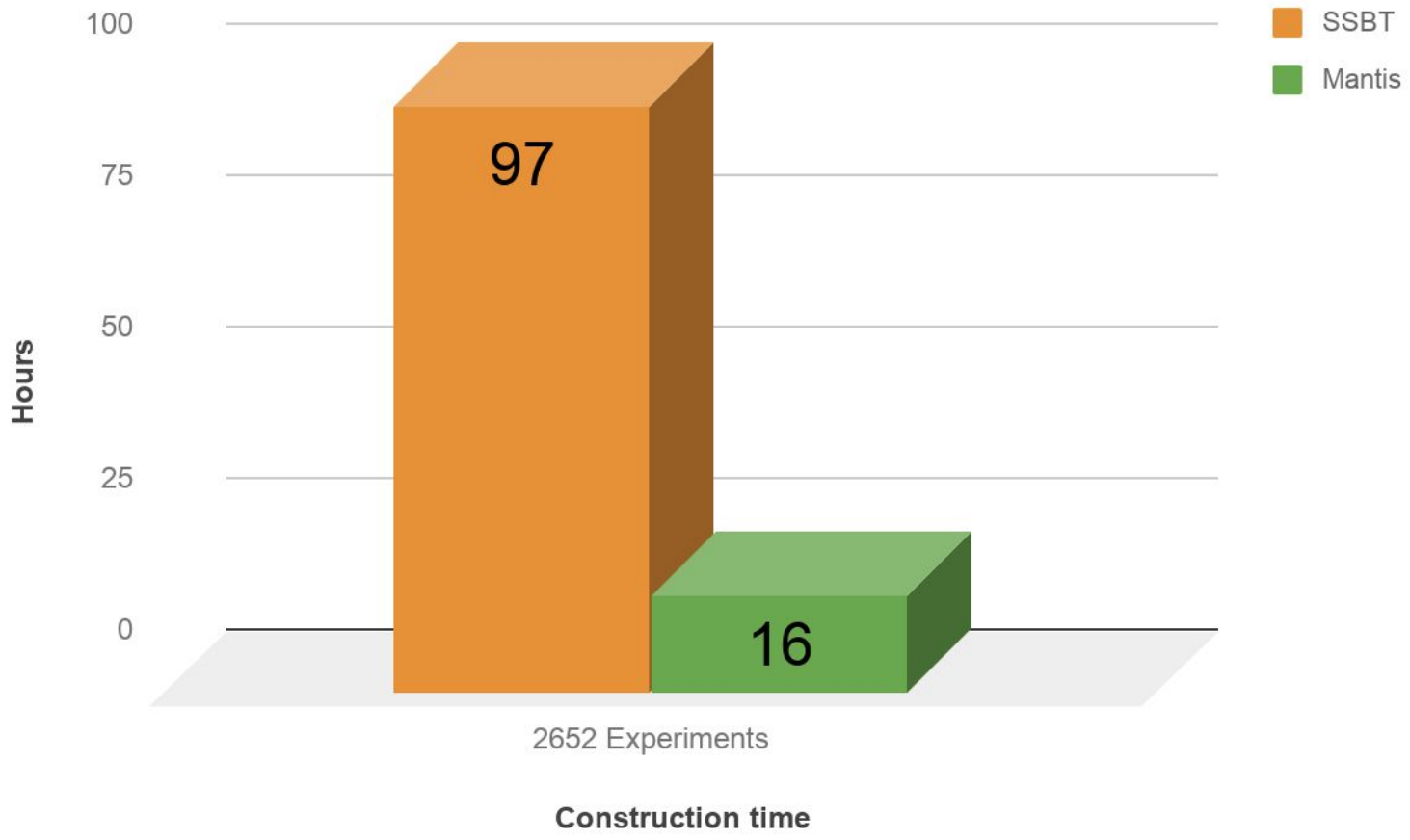
- Build index for 2652 samples of RNA-seq short-read sequencing runs of human blood, brain, and breast tissue.
- Total size of all the samples: ≅7 Terabytes
- Compared with SSBT.
- Evaluation metrics:
  - Index size
  - Construction time
  - Query performance
  - Quality of results

Index size



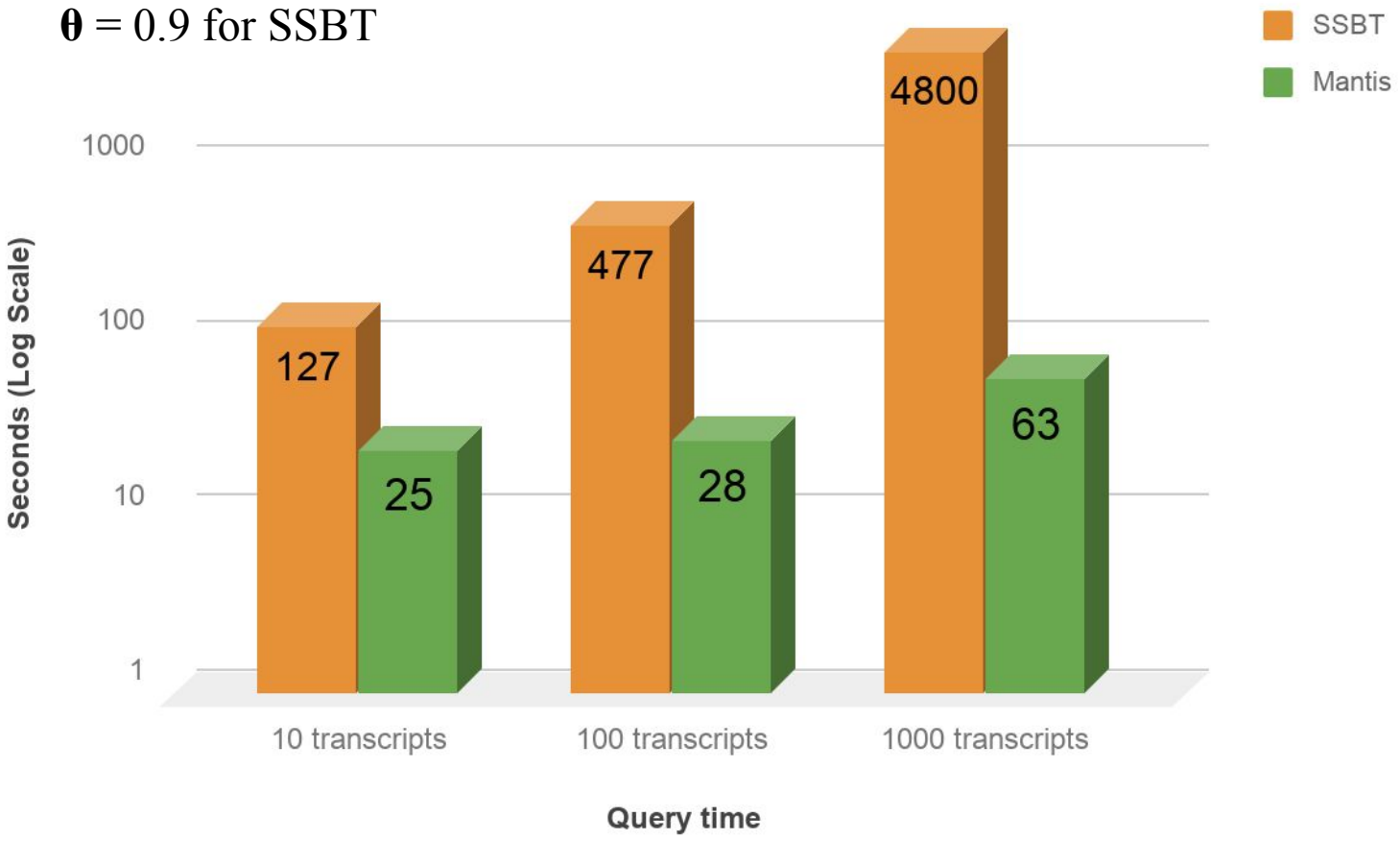
Mantis is 20% smaller than the SSBT.

Construction time



Mantis is 6X faster than the SSBT for construction.

Query performance



Mantis is 6X-100X faster than the SSBT for queries.

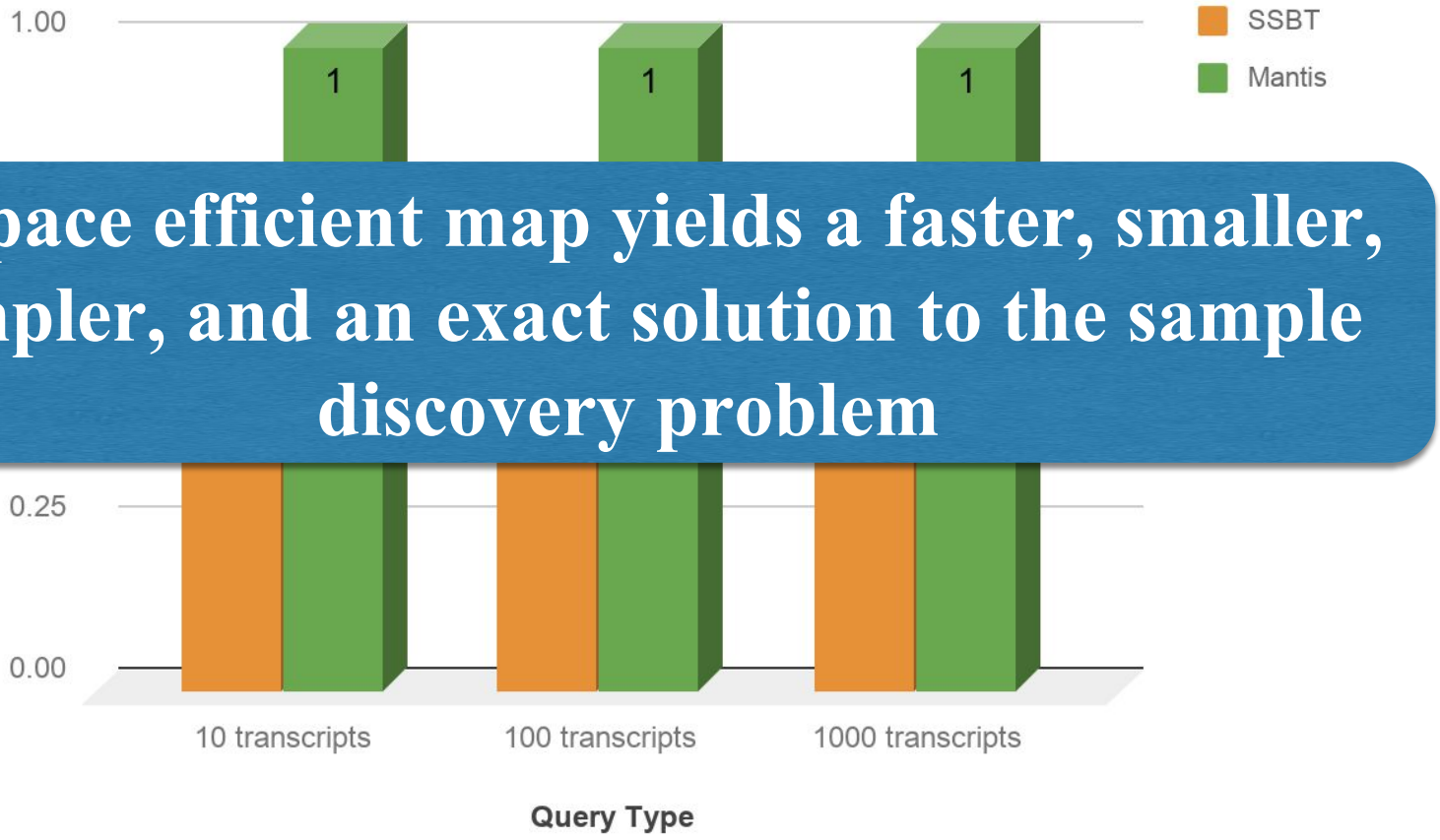
Quality of results

θ = 0.9 for SSBT

Mantis is exact.

Quality of results

θ = 0.9 for SSBT

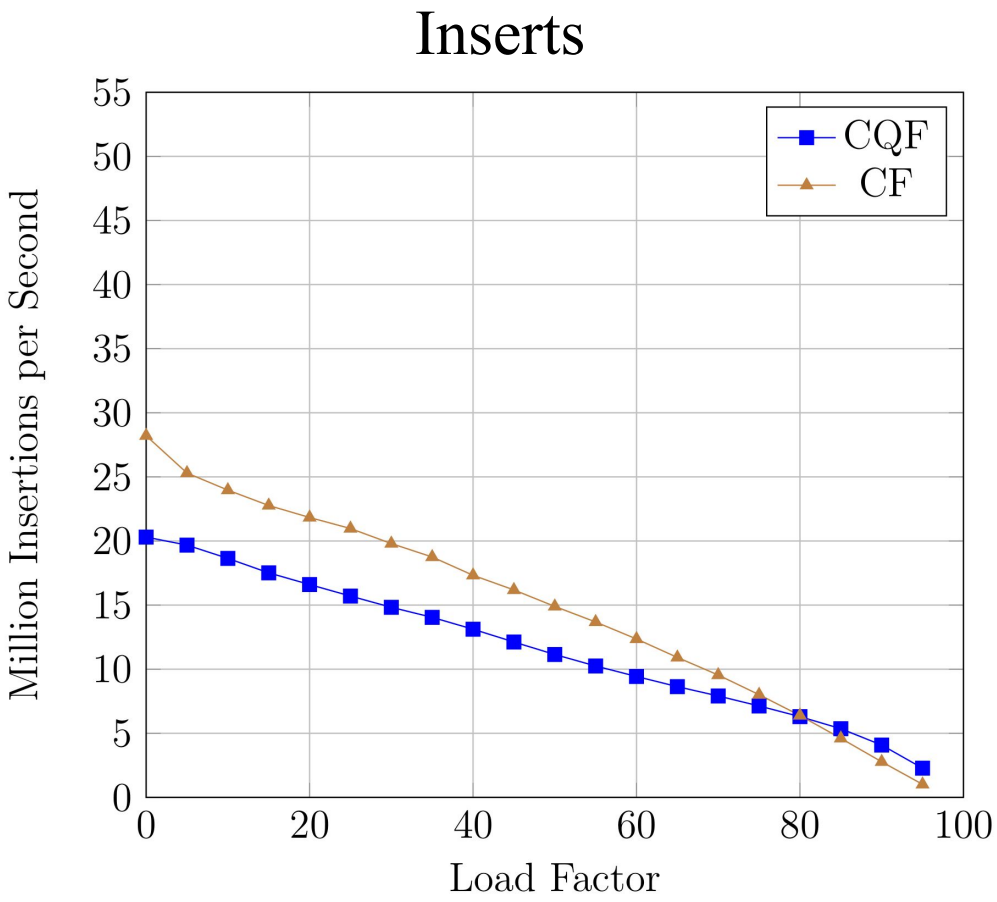


A space efficient map yields a faster, smaller, simpler, and an exact solution to the sample discovery problem

Mantis is exact.

Proposed work: Improving the QF insertion throughput at high load factors

Insertion throughput at high load factors

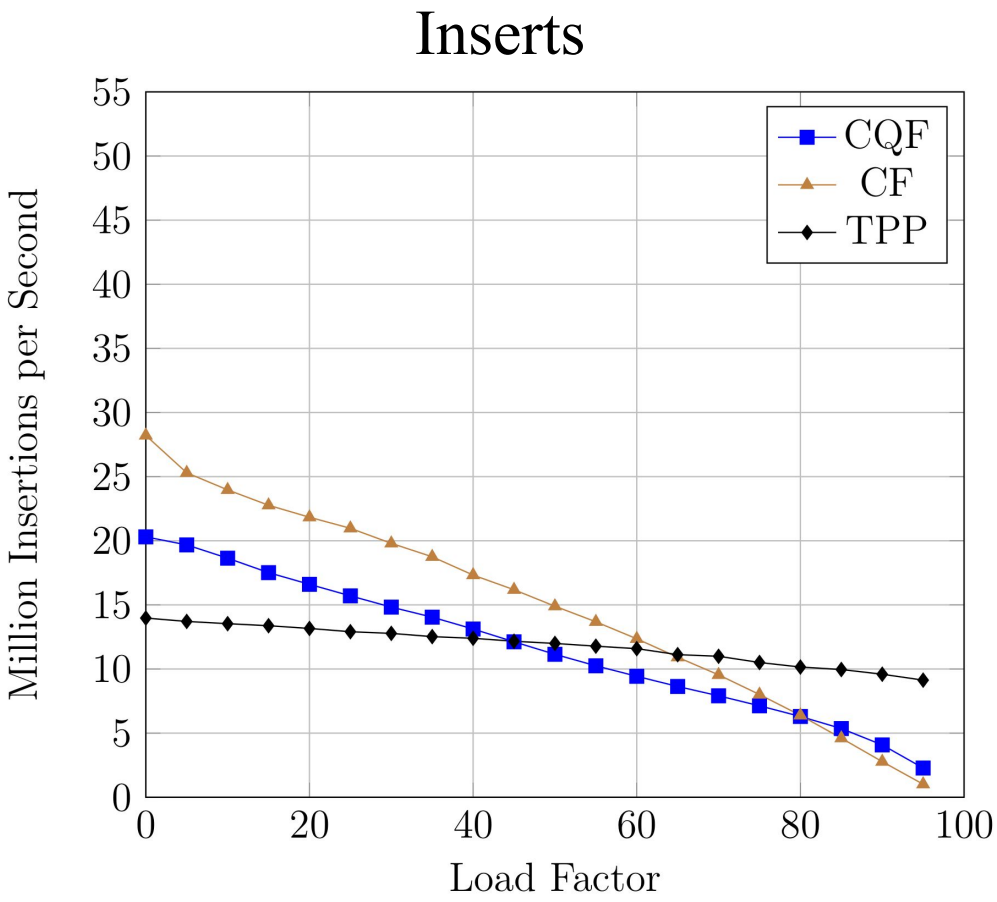


Insertion throughput at 90% load factor is 5×–10× slower than at 10% load factor.

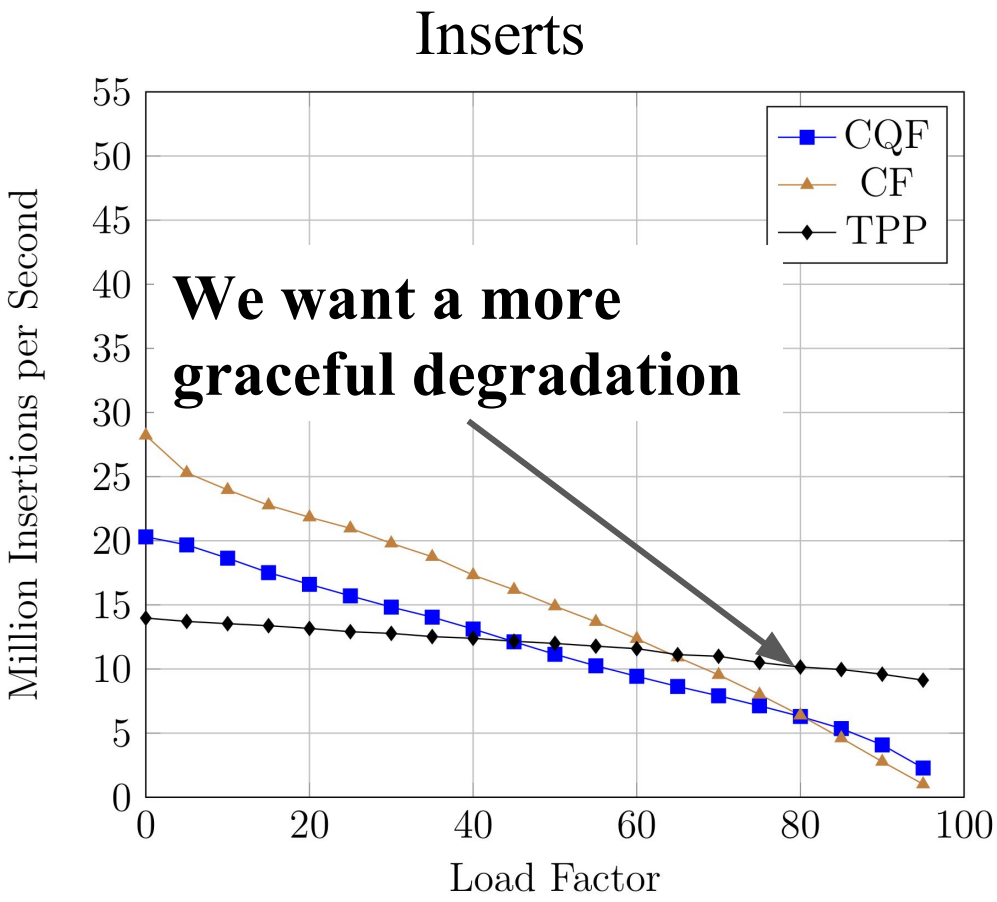
Improving insertion throughput at high load factors

- Use a small stash to keep a check on cluster lengths.
  - Use a stash of size $\cong(n/\log n)$ slots.
  - If the distance between the home slot and the next empty slot is greater than 64 (or 128), use the stash.
- Cache align qf-blocks.
- Give up counting to reduce shifting of tags.
  - No need to keep tags in orders inside runs.

Target performance profile (TPP)

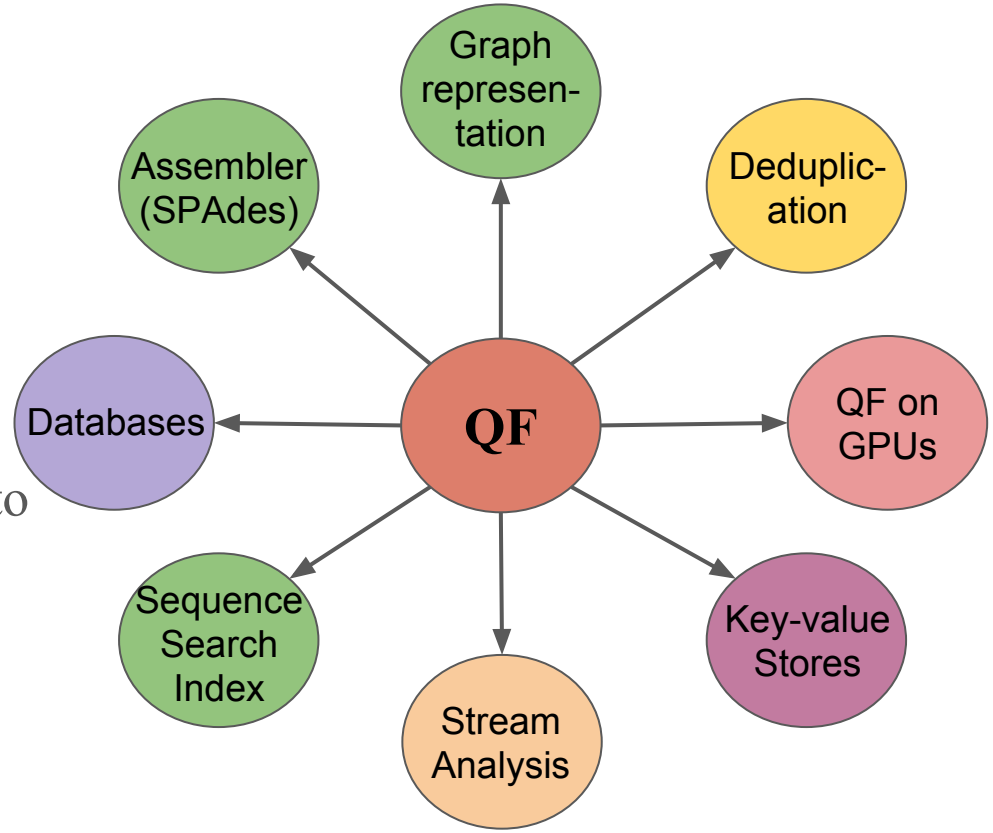


Target performance profile (TPP)



Conclusion

- Space-efficient and fast compact data structures can help solve big data problems across subfields.
- We need to redesign applications to get complete benefits of today's feature-rich AMQs.



Source code: https://github.com/splatlab/