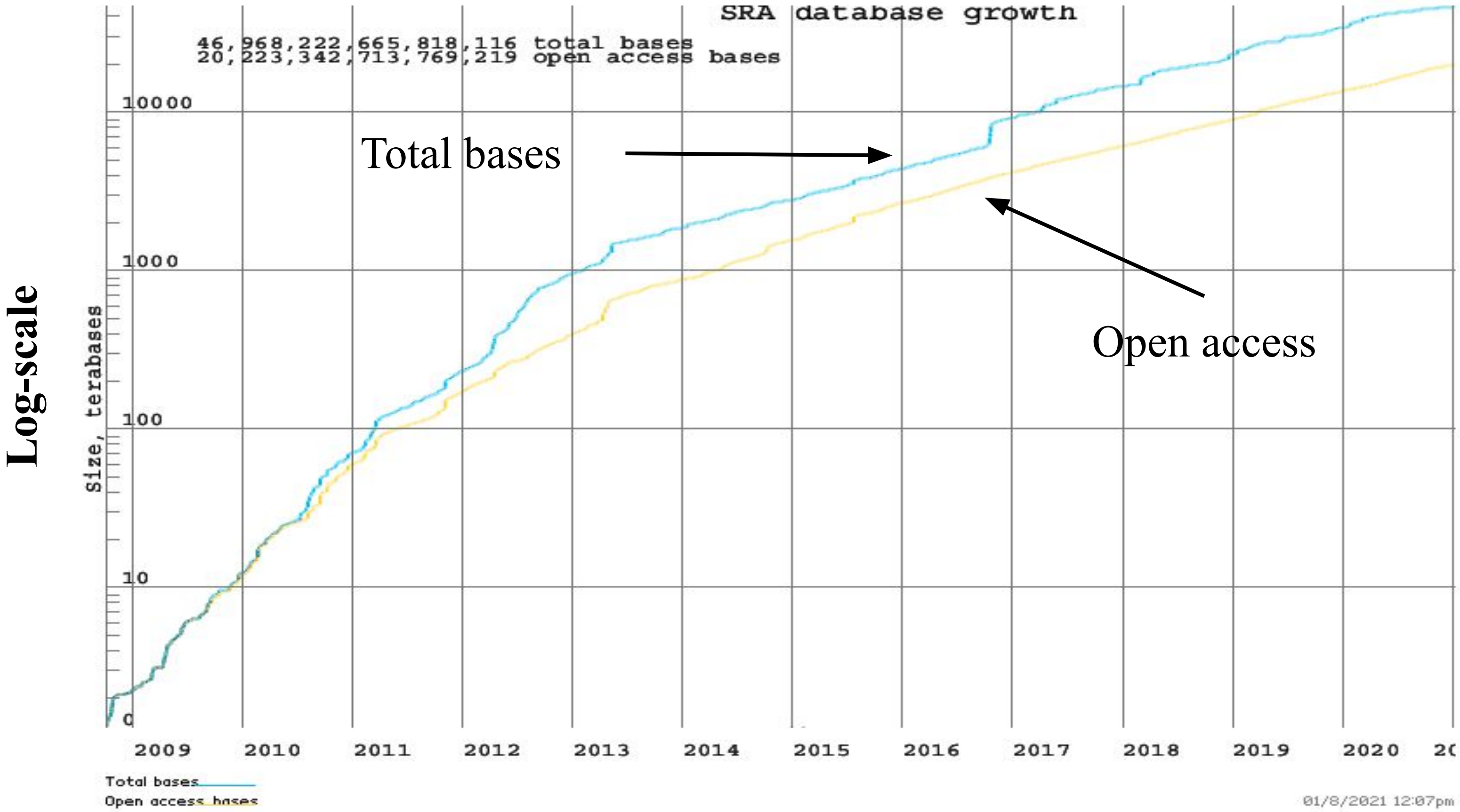
Data Science at Scale:

Scaling Up by Scaling Down and Out (to Disk)

Prashant Pandey
ppandey@berkeley.edu
Berkeley Lab/UC Berkeley

Sequence Read Archive (SRA) database growth



Log-scale

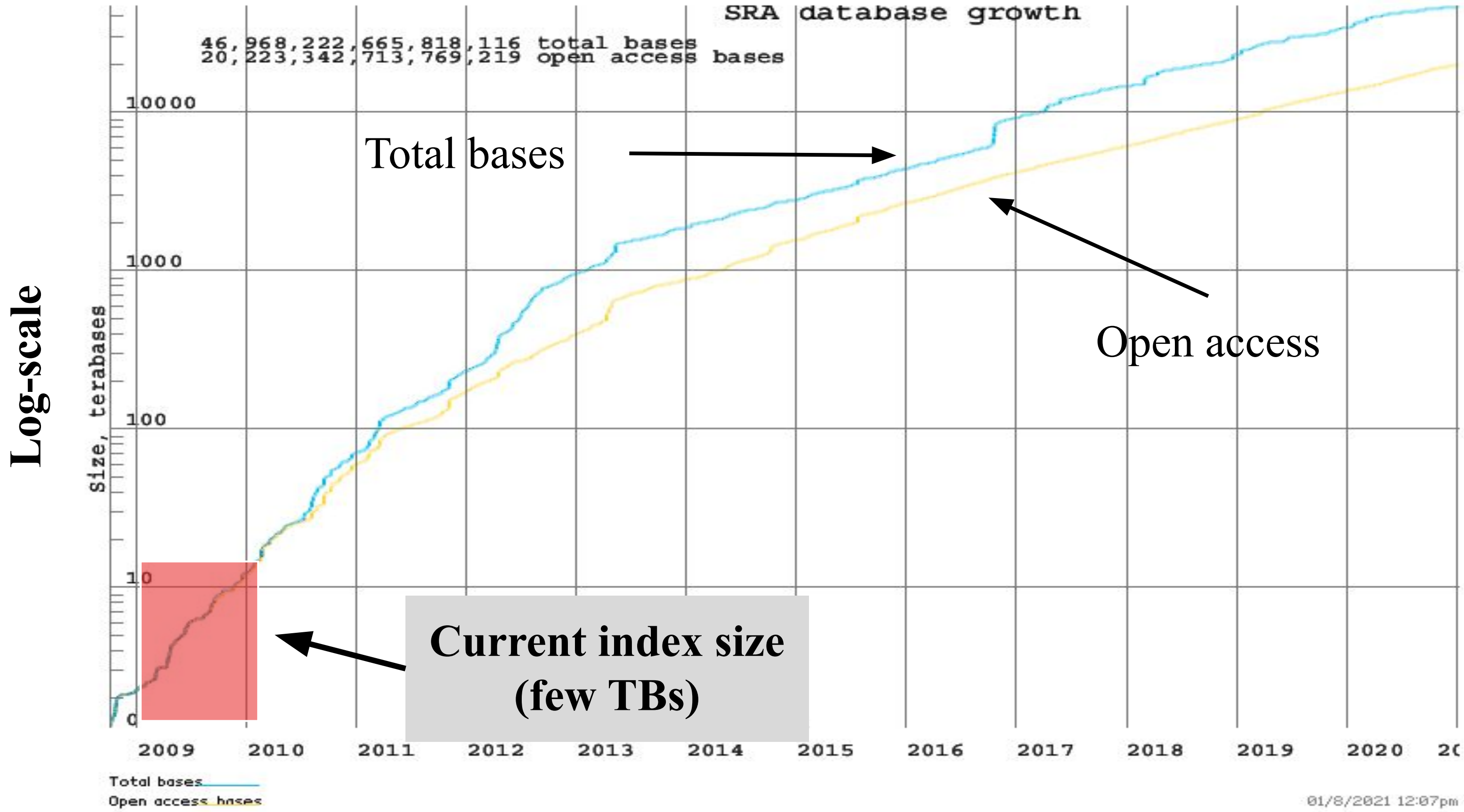
Total bases

Open access

SRA contains a lot of ***diversity information***

Goal: perform ***sequence searches*** on the database

Scalability is the bottleneck for data science



Log-scale

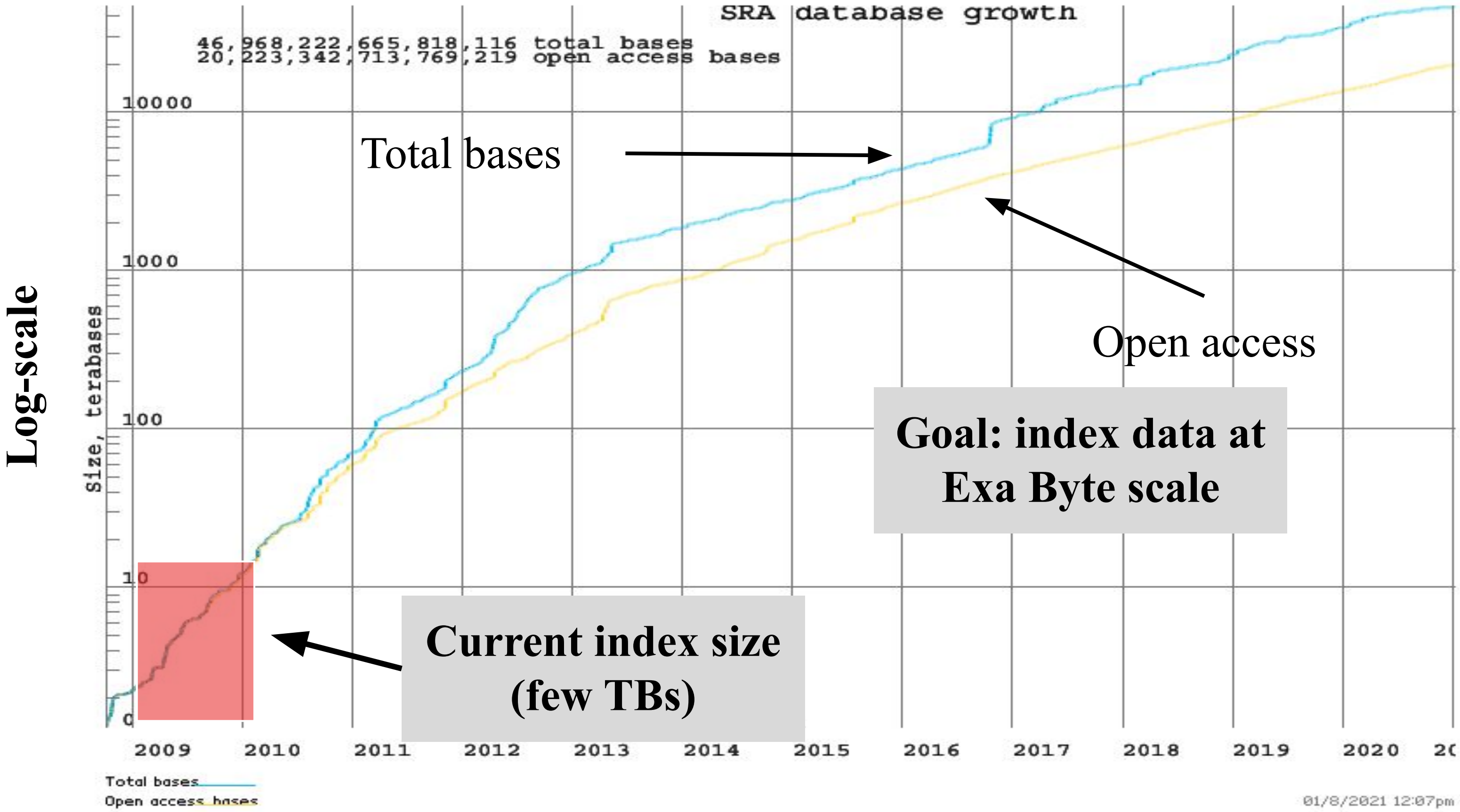
Total bases

Open access

Current index size (few TBs)

Data science applications only looking at a ***small portion*** of data

Scalable data systems → Scalable data science



Log-scale

Total bases

Open access

Goal: index data at Exa Byte scale

Current index size (few TBs)

My goal as a researcher is to build ***scalable data systems*** to ***accelerate*** and ***scale data science*** applications

Three approaches to handle massive data

Three approaches to handle massive data

Shrink it

Goal: make data smaller to fit in RAM

Techniques:

- Compact & succinct data structures
- Filters, e.g., Bloom, quotient, etc.

Three approaches to handle massive data

Shrink it

Goal: make data smaller to fit in RAM

Techniques:

- Compact & succinct data structures
- Filters, e.g., Bloom, quotient, etc.

Organize it

Goal: organize data in a disk-friendly way

Techniques:

- B-tree
- B^{ε}-tree
- LSM-tree

Three approaches to handle massive data

Shrink it

Goal: make data smaller to fit in RAM

Techniques:

- Compact & succinct data structures
- Filters, e.g., Bloom, quotient, etc.

Organize it

Goal: organize data in a disk-friendly way

Techniques:

- B-tree
- B^{ε}-tree
- LSM-tree

Distribute it

Goal: partition and distribute data on multiple nodes

Techniques:

- Distributed hash table
- Distributed key-value store

Research output

Data structures & Algorithms

(Counting) Quotient Filter
SIGMOD ‘17, arXiv ‘17

Buffered Count-Min Sketch
ESA ‘18

Order Min Hash
ISMB ‘19

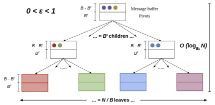
Research output

Data structures & Algorithms

(Counting) Quotient Filter
SIGMOD ‘17,
arXiv ‘17

Buffered Count-Min Sketch
ESA ‘18

Order Min Hash
ISMB ‘19

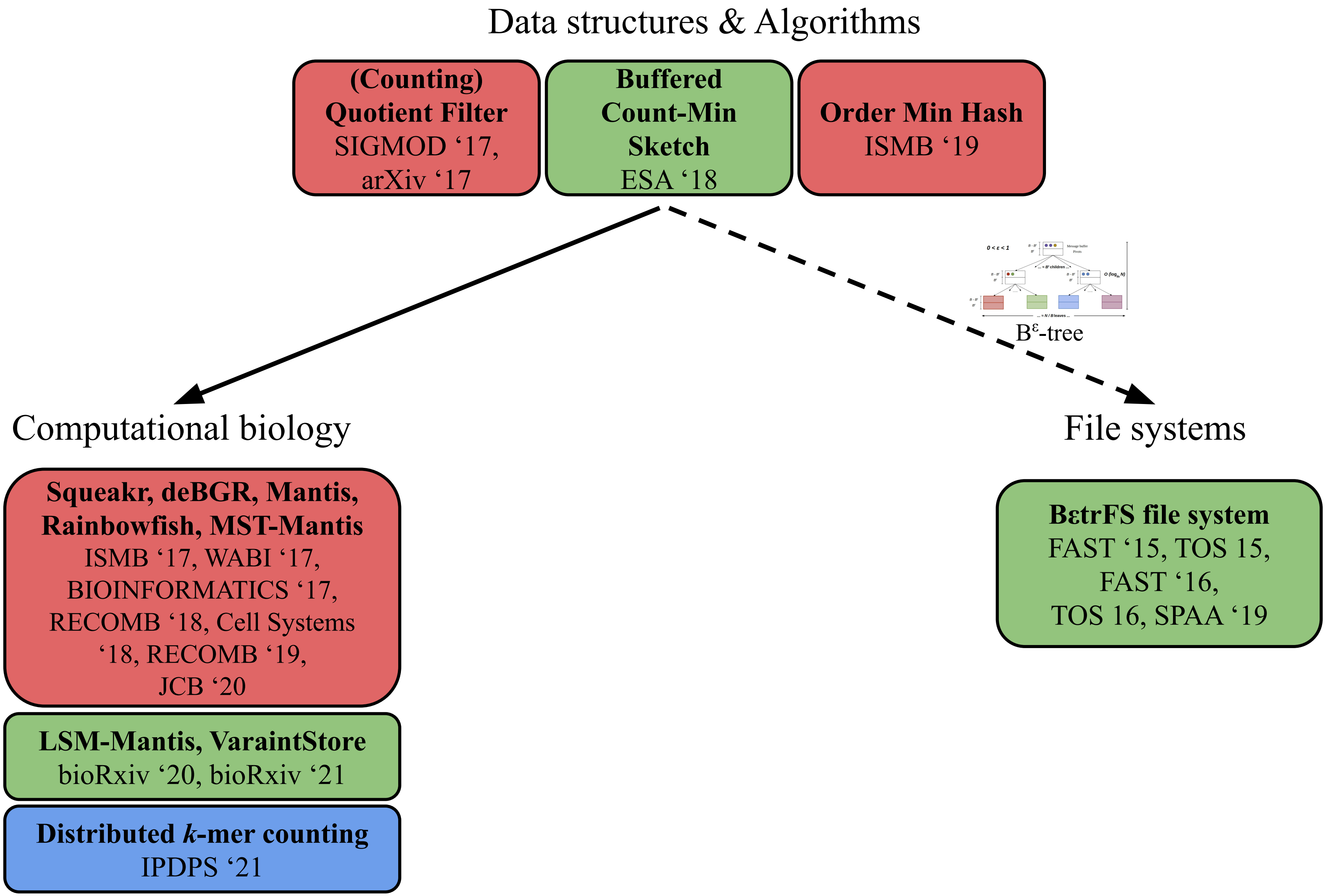


B^{ε}-tree

File systems

BεtrFS file system
FAST ‘15, TOS 15,
FAST ‘16,
TOS 16, SPAA ‘19

Research output



Data structures & Algorithms

(Counting) Quotient Filter
SIGMOD ‘17, arXiv ‘17

Buffered Count-Min Sketch
ESA ‘18

Order Min Hash
ISMB ‘19

B^ε-tree

Computational biology

Squeakr, deBGR, Mantis, Rainbowfish, MST-Mantis
ISMB ‘17, WABI ‘17, BIOINFORMATICS ‘17, RECOMB ‘18, Cell Systems ‘18, RECOMB ‘19, JCB ‘20

LSM-Mantis, VaraintStore
bioRxiv ‘20, bioRxiv ‘21

Distributed *k*-mer counting
IPDPS ‘21

File systems

BεtrFS file system
FAST ‘15, TOS 15, FAST ‘16, TOS 16, SPAA ‘19

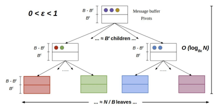
Research output

Data structures & Algorithms

(Counting) Quotient Filter
SIGMOD ‘17, arXiv ‘17

Buffered Count-Min Sketch
ESA ‘18

Order Min Hash
ISMB ‘19



B$^{\varepsilon}$-tree

Computational biology

Squeakr, deBGR, Mantis, Rainbowfish, MST-Mantis
ISMB ‘17, WABI ‘17, BIOINFORMATICS ‘17, RECOMB ‘18, Cell Systems ‘18, RECOMB ‘19, JCB ‘20

LSM-Mantis, VaraintStore
bioRxiv ‘20, bioRxiv ‘21

Distributed *k*-mer counting
IPDPS ‘21

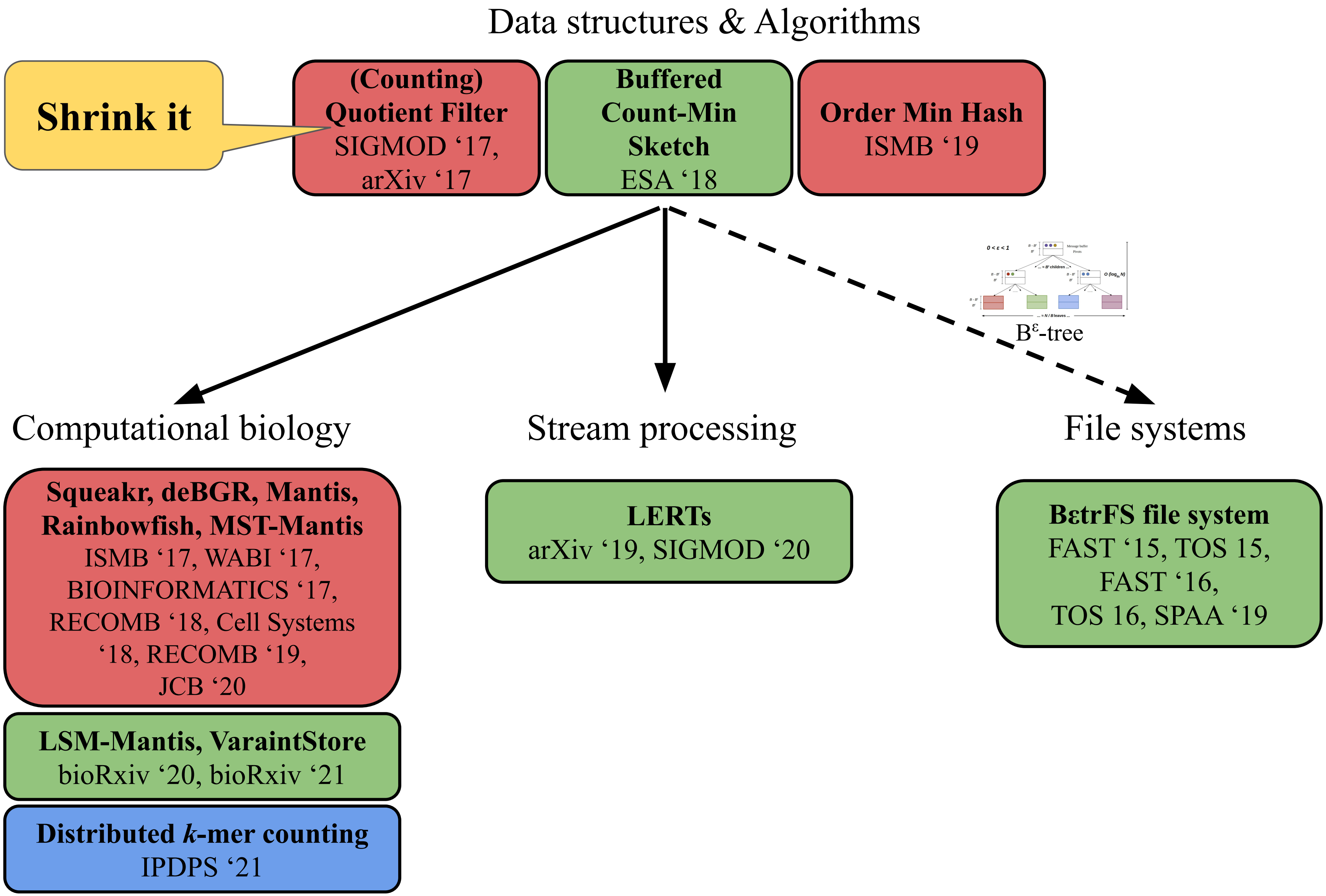
Stream processing

LERTs
arXiv ‘19, SIGMOD ‘20

File systems

BεtrFS file system
FAST ‘15, TOS 15, FAST ‘16, TOS 16, SPAA ‘19

In this talk



Data structures & Algorithms

Shrink it

(Counting) Quotient Filter
SIGMOD ‘17, arXiv ‘17

Buffered Count-Min Sketch
ESA ‘18

Order Min Hash
ISMB ‘19

B$^{\varepsilon}$-tree

Computational biology

Squeakr, deBGR, Mantis, Rainbowfish, MST-Mantis
ISMB ‘17, WABI ‘17, BIOINFORMATICS ‘17, RECOMB ‘18, Cell Systems ‘18, RECOMB ‘19, JCB ‘20

LSM-Mantis, VaraintStore
bioRxiv ‘20, bioRxiv ‘21

Distributed *k*-mer counting
IPDPS ‘21

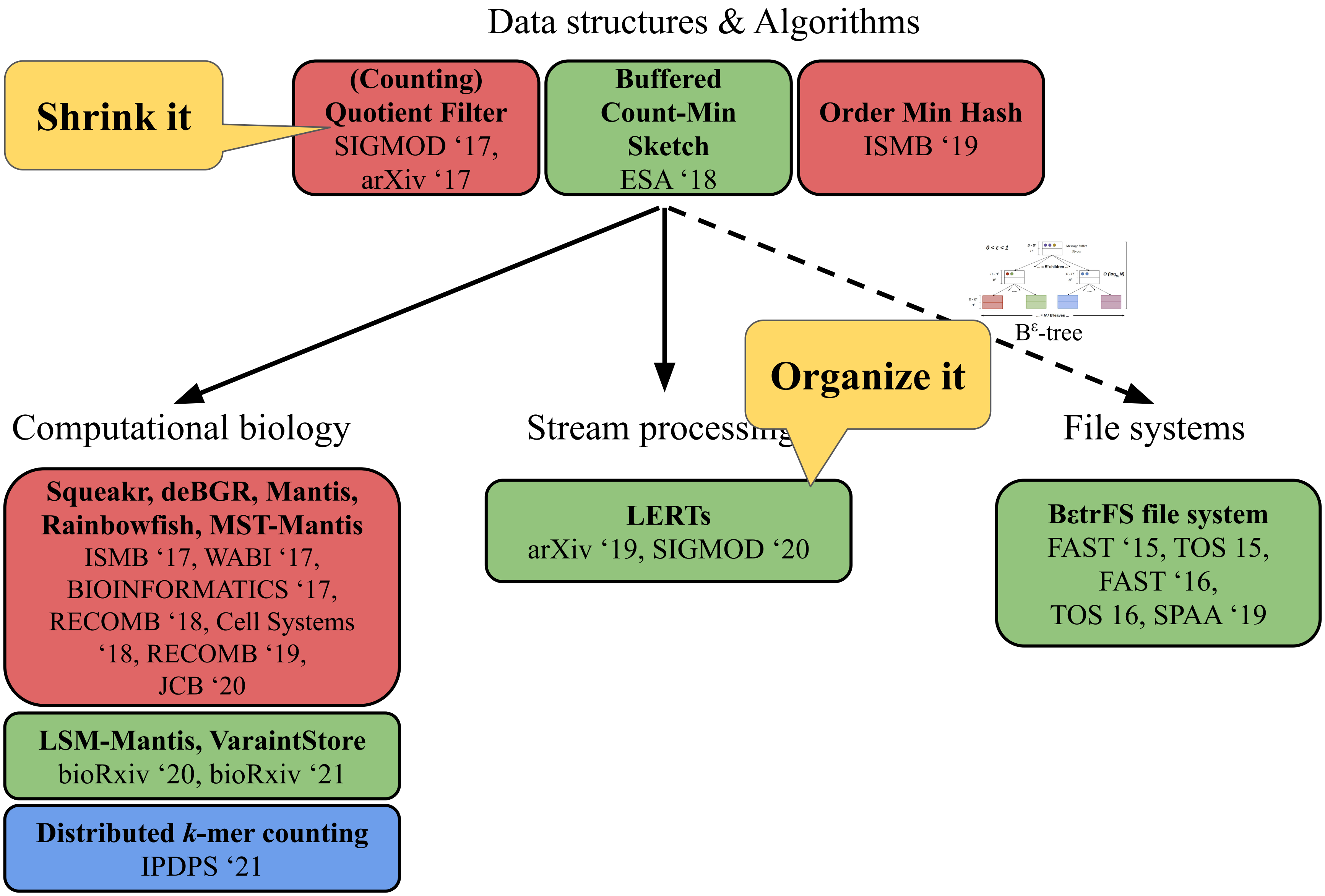
Stream processing

LERTs
arXiv ‘19, SIGMOD ‘20

File systems

BεtrFS file system
FAST ‘15, TOS 15, FAST ‘16, TOS 16, SPAA ‘19

In this talk



Data structures & Algorithms

Shrink it

(Counting) Quotient Filter
SIGMOD ‘17, arXiv ‘17

Buffered Count-Min Sketch
ESA ‘18

Order Min Hash
ISMB ‘19

B^{ε}-tree

Organize it

Computational biology

Squeakr, deBGR, Mantis, Rainbowfish, MST-Mantis
ISMB ‘17, WABI ‘17, BIOINFORMATICS ‘17, RECOMB ‘18, Cell Systems ‘18, RECOMB ‘19, JCB ‘20

LSM-Mantis, VaraintStore
bioRxiv ‘20, bioRxiv ‘21

Distributed *k*-mer counting
IPDPS ‘21

Stream processing

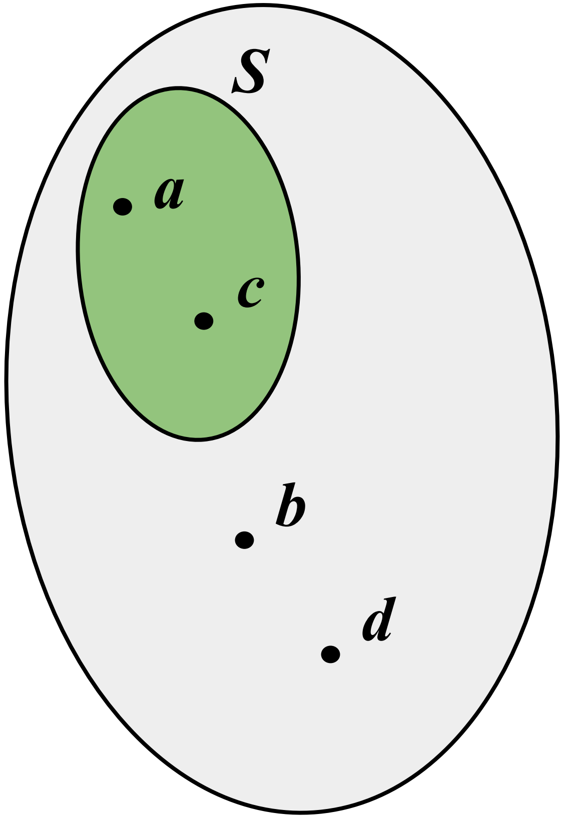
LERTs
arXiv ‘19, SIGMOD ‘20

File systems

BεtrFS file system
FAST ‘15, TOS 15, FAST ‘16, TOS 16, SPAA ‘19

Dictionary data structure

A dictionary maintains a set S from universe U.



membership(a): ✓

membership(b): ✗

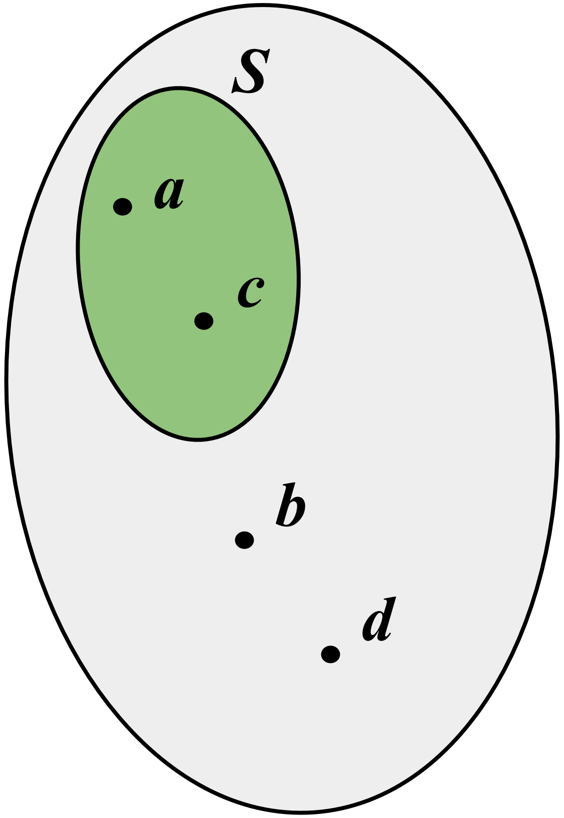
membership(c): ✓

membership(d): ✗

A dictionary supports membership queries on S.

Filter data structure

A filter is an ***approximate*** dictionary.



membership(a): ✓

membership(b): ✗

membership(c): ✓

membership(d): ✓ 👎 **false positive**

A filter supports ***approximate*** membership queries on S.

A filter guarantees a false-positive rate ε

if $q \in$ S, return ✓ with probability 1 **true positive**

if $q \notin$ S, return {

✗ with probability □1 - ε **true negative**

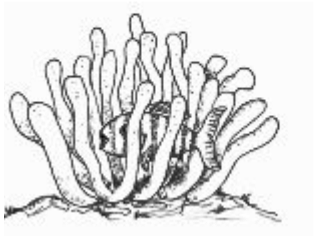
✓ with probability $\leq \varepsilon$ **false positive**

one-sided errors

False-positive rate enables filters to be compact

$$\text{space} \geq n \log(1/\epsilon)$$

$$\text{space} = \Omega(n \log |U|)$$



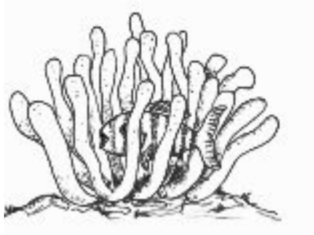
Filter

Dictionary

False-positive rate enables filters to be compact

$$\text{space} \geq \underbrace{n \log(1/\epsilon)}_{\text{Small}}$$

$$\text{space} = \Omega(n \log \underbrace{|U|}_{\text{Large}})$$



Filter

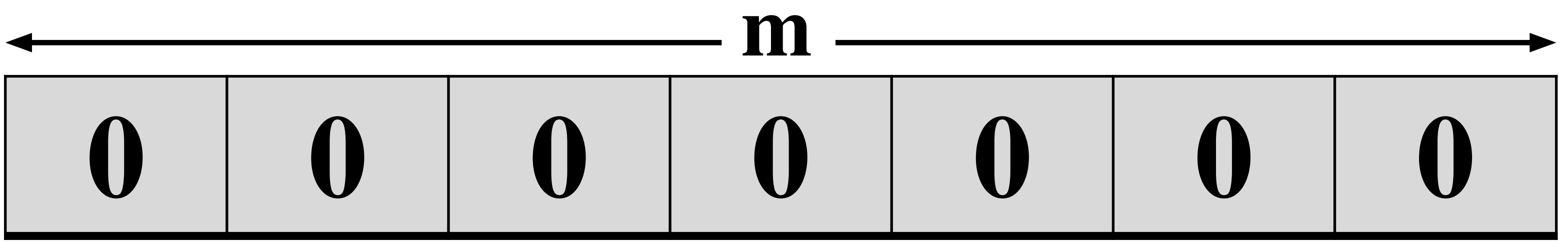
Dictionary

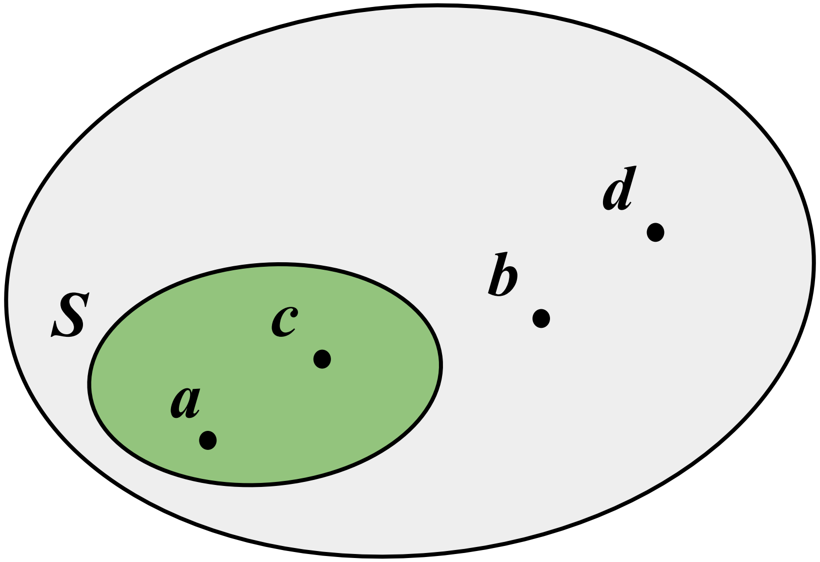
For most practical purposes:

ε = 2%, Bloom filter requires ≈ 8 bits/item

Classic filter: The Bloom filter [Bloom '70]

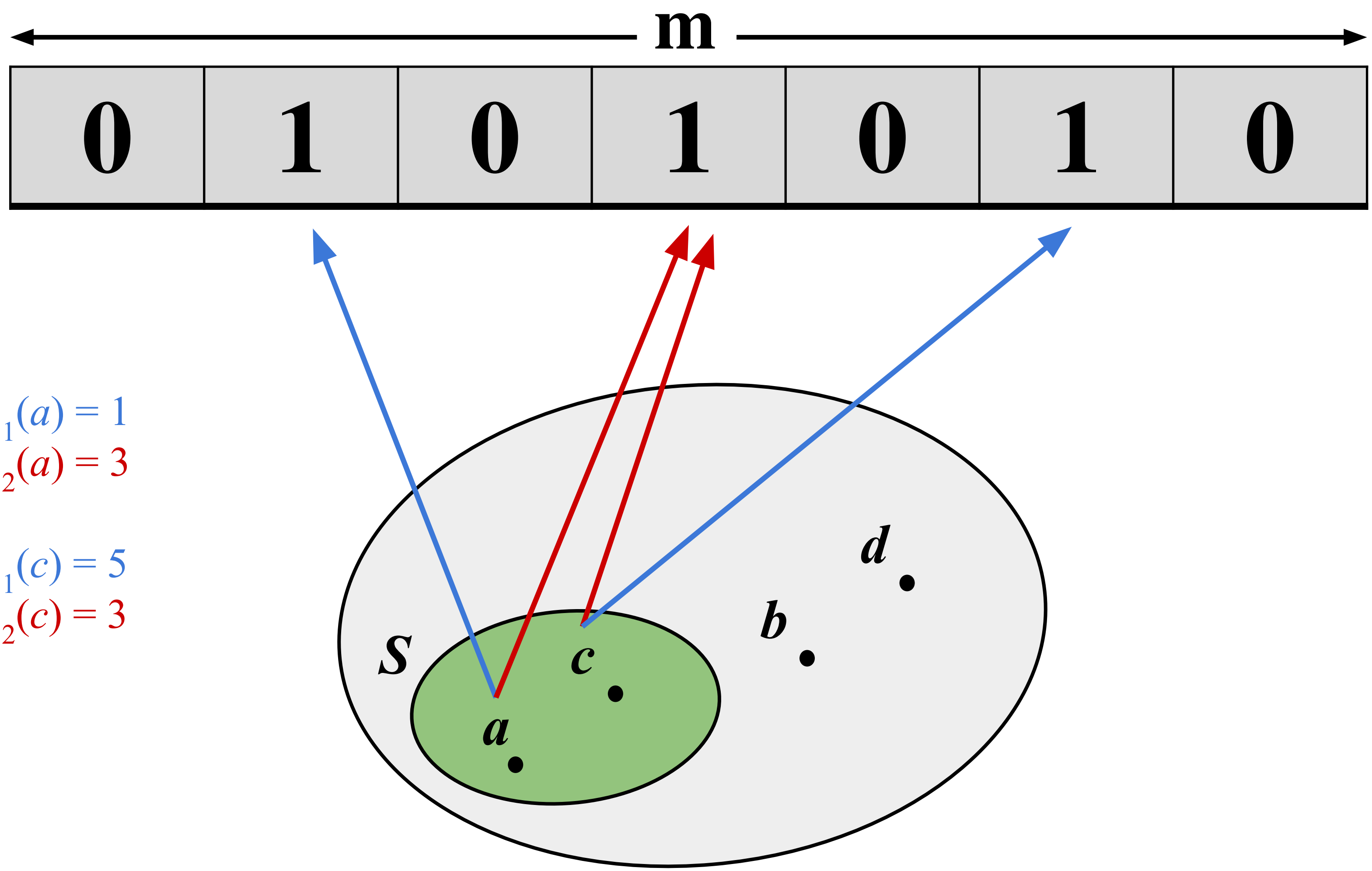
Bloom filter: a bit array + *k* hash functions





Classic filter: The Bloom filter [Bloom ‘70]

Bloom filter: a bit array + k hash functions (here $k = 2$)

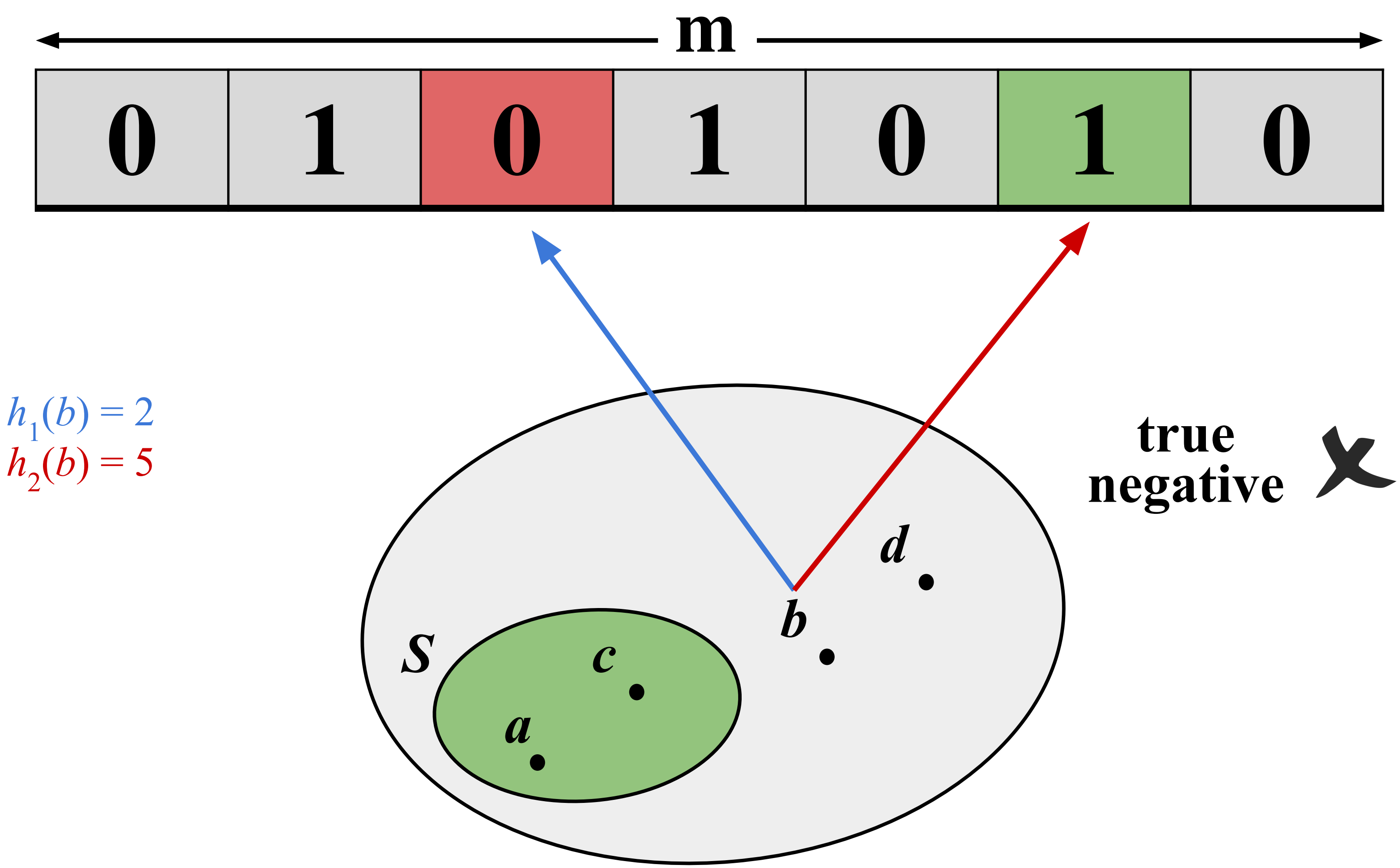


$h_1(a) = 1$
$h_2(a) = 3$

$h_1(c) = 5$
$h_2(c) = 3$

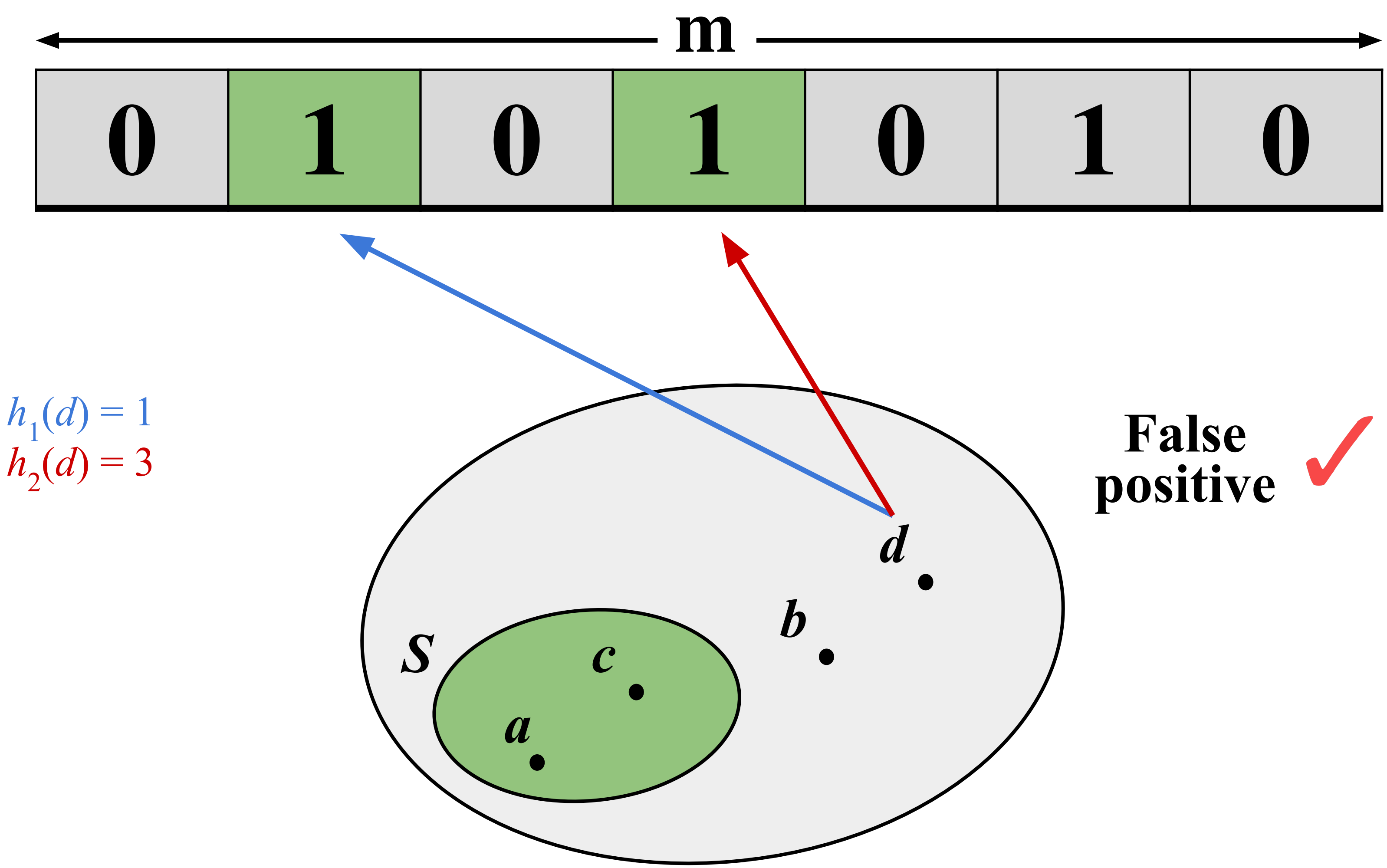
Classic filter: The Bloom filter [Bloom '70]

Bloom filter: a bit array + k hash functions (here k=2)



Classic filter: The Bloom filter [Bloom ‘70]

Bloom filter: a bit array + k hash functions (here k=2)



0	1	0	1	0	1	0

$h_1(d) = 1$
$h_2(d) = 3$

False positive ✓

Bloom filter are ubiquitous (> 4300 citations)

Streaming applications

Networking

Databases

Computational biology

Storage systems

Bloom filter have suboptimal asymptotics

	Bloom filter	Optimal
Space	$\approx 1.44\, n \log(1/\epsilon)$	$\approx n \ \log(1/\epsilon) + \Omega(n)$
CPU cost	$\Omega(1/\epsilon)$	$O(1)$
Data locality	$\Omega(1/\epsilon)$ probes	$O(1)$ probes

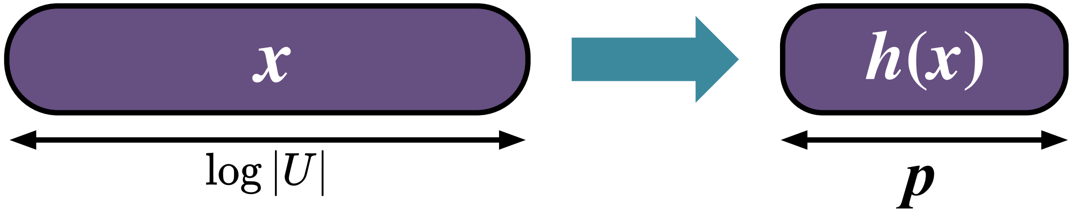
Application often work around Bloom filter limitations

Limitations	Workarounds
No deletes	Rebuild
No resizes	Guess *N*, and rebuild if wrong
No filter merging or enumeration	???
No values associated with keys	Combine with another data structure

Bloom filter limitations increase system complexity, waste space, and slow down application performance

Quotienting is an alternative to Bloom filters
[Knuth. Searching and Sorting Vol. 3, '97]

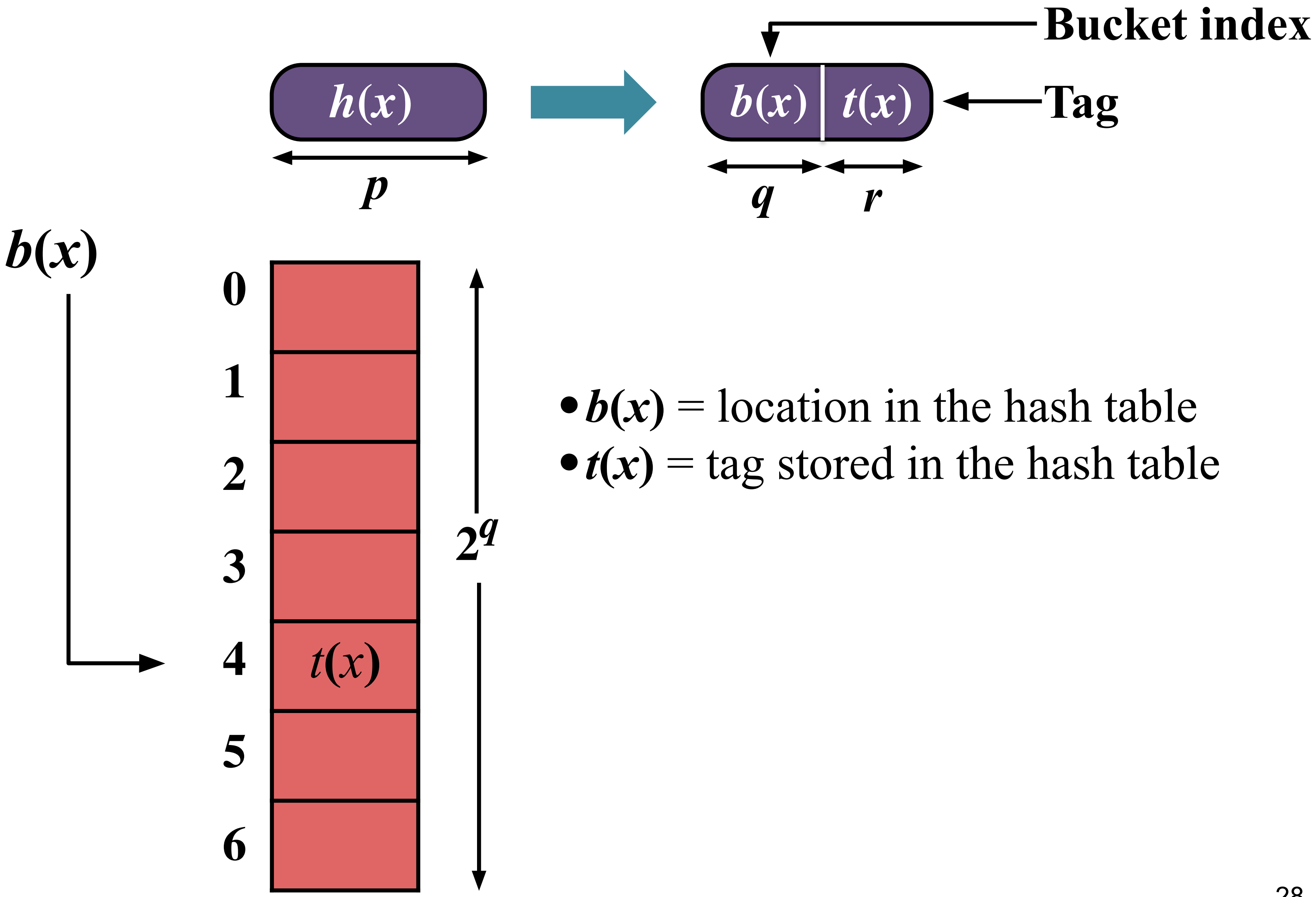
- **Store fingerprints compactly in a hash table.**
  - Take a fingerprint $h(x)$ for each element x.



- **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 positives

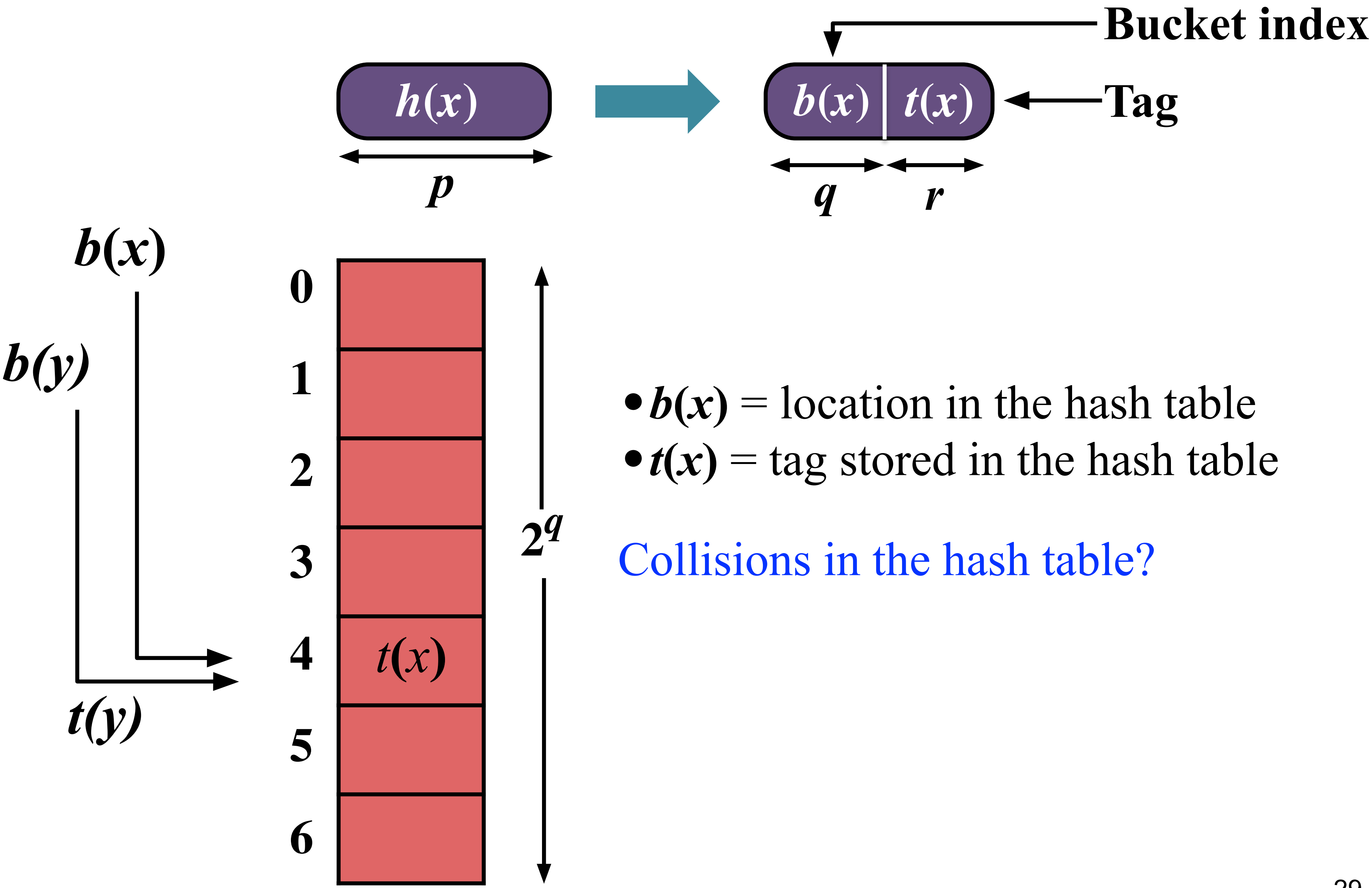
$$\Pr[\, x \text{ and } y \text{ collide}] = \frac{1}{2^p}$$

Storing fingerprints compactly



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

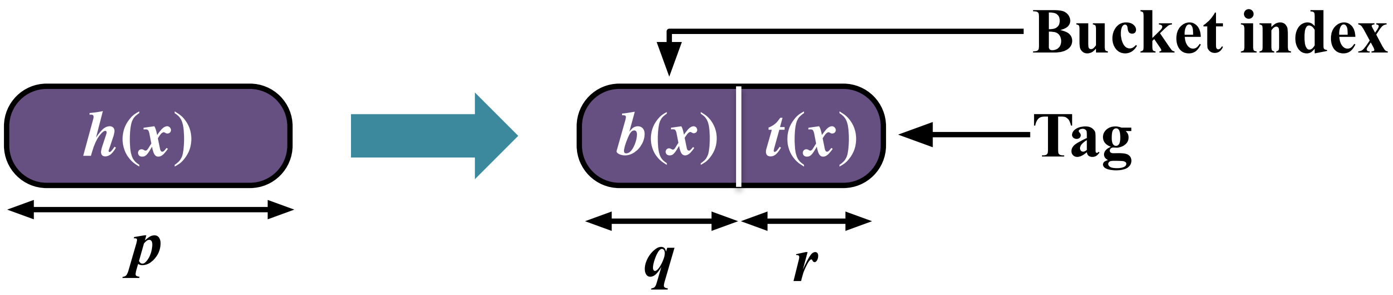
Storing fingerprints compactly

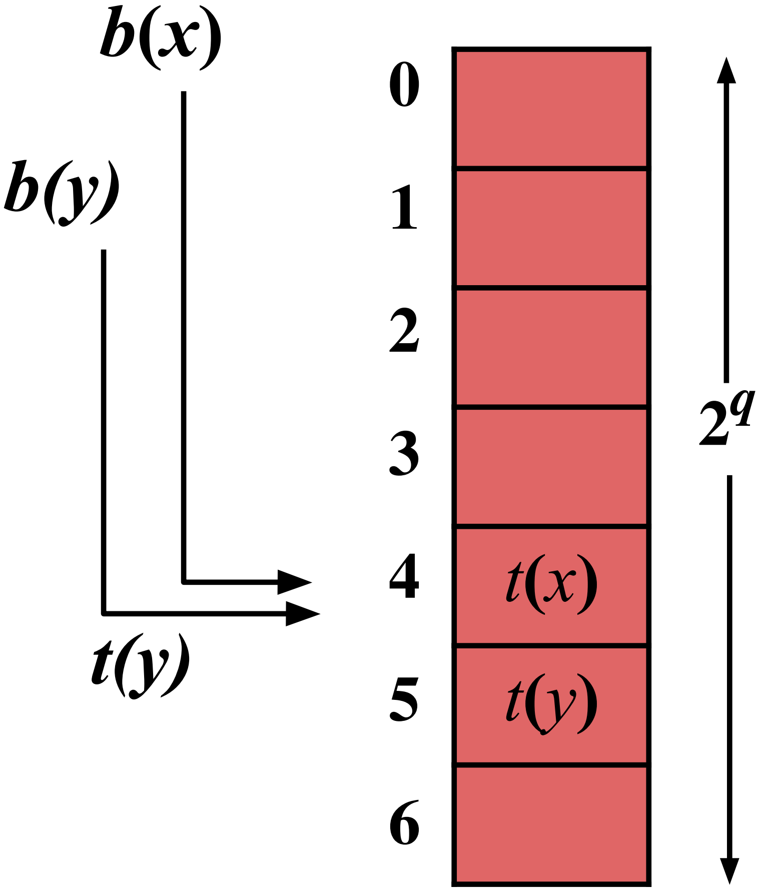


- $b(x)$ = location in the hash table
- $t(x)$ = tag stored in the hash table

Collisions in the hash table?

Storing fingerprints compactly



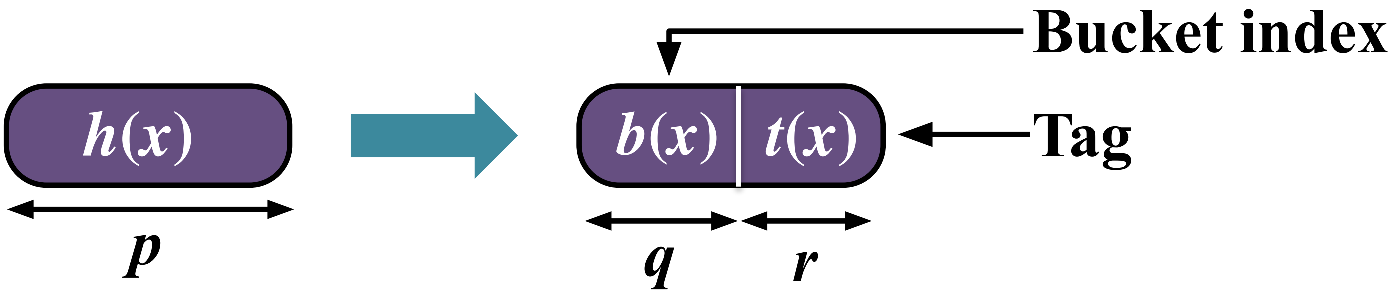


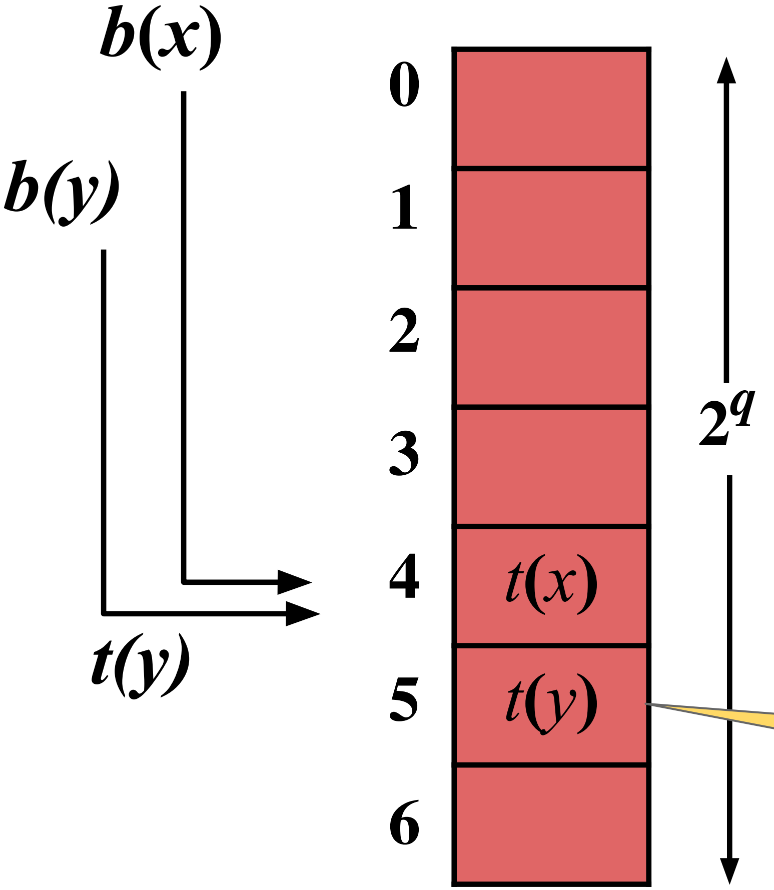
- $b(x)$ = location in the hash table
- $t(x)$ = tag stored in the hash table

Collisions in the hash table?

- Linear probing
- Robin Hood hashing

Storing fingerprints compactly





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

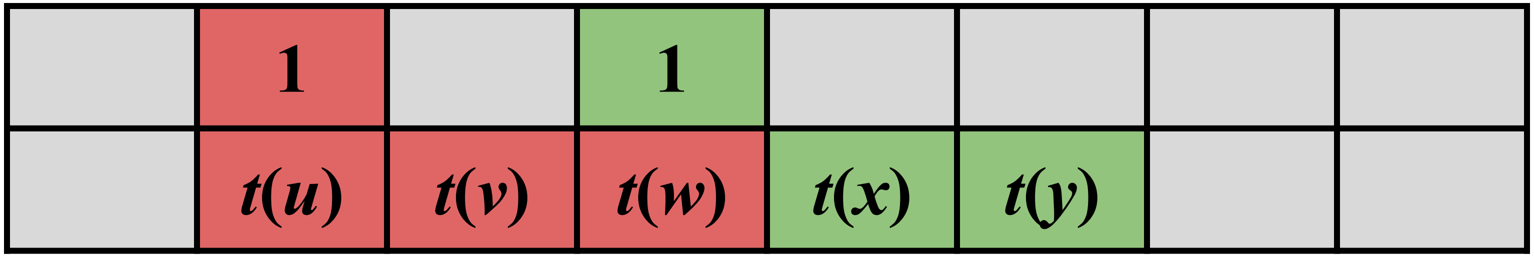
Collisions in the hash table?

- Linear probing
- Robin Hood hashing

$t(y)$ belongs to slots 4 or 5?

Resolving collisions in the QF [Bender ‘12, Pandey ‘17]

- QF uses two metadata bits to resolve collisions and identify home bucket

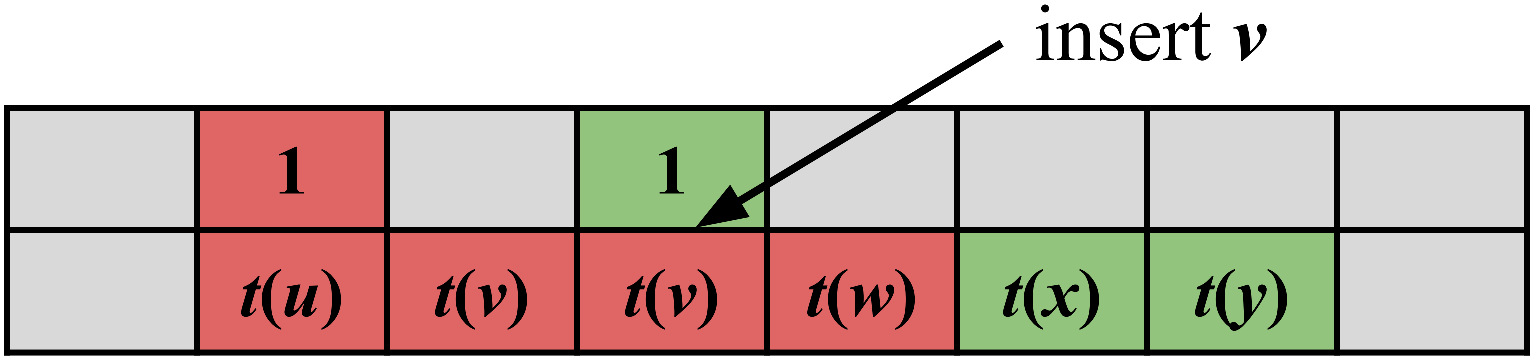


	1		1				
	t(*u*)	*t*(*v*)	*t*(*w*)	*t*(*x*)	*t*(*y*)		

- The metadata bits group tags by their home bucket

Resolving collisions in the QF [Bender '12, Pandey '17]

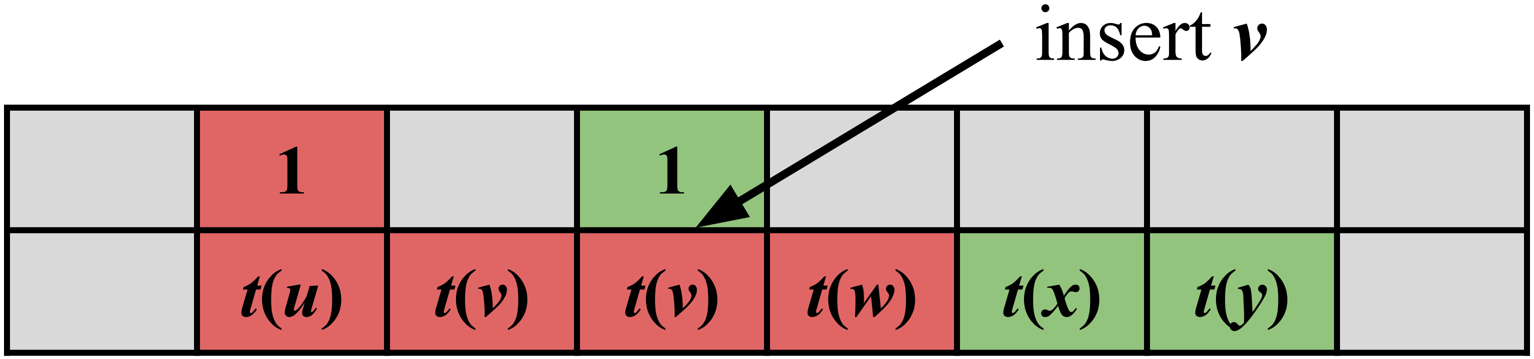
- QF uses two metadata bits to resolve collisions and identify home bucket



- The metadata bits group tags by their home bucket

Resolving collisions in the QF [Bender '12, Pandey '17]

- QF uses two metadata bits to resolve collisions and identify home bucket



insert ***v***

	1		**1**				
	***t*(*u*)**	***t*(*v*)**	***t*(*v*)**	***t*(*w*)**	***t*(*x*)**	***t*(*y*)**	

- The metadata bits group tags by their home bucket

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

More

Quotienting enables many features in the QF

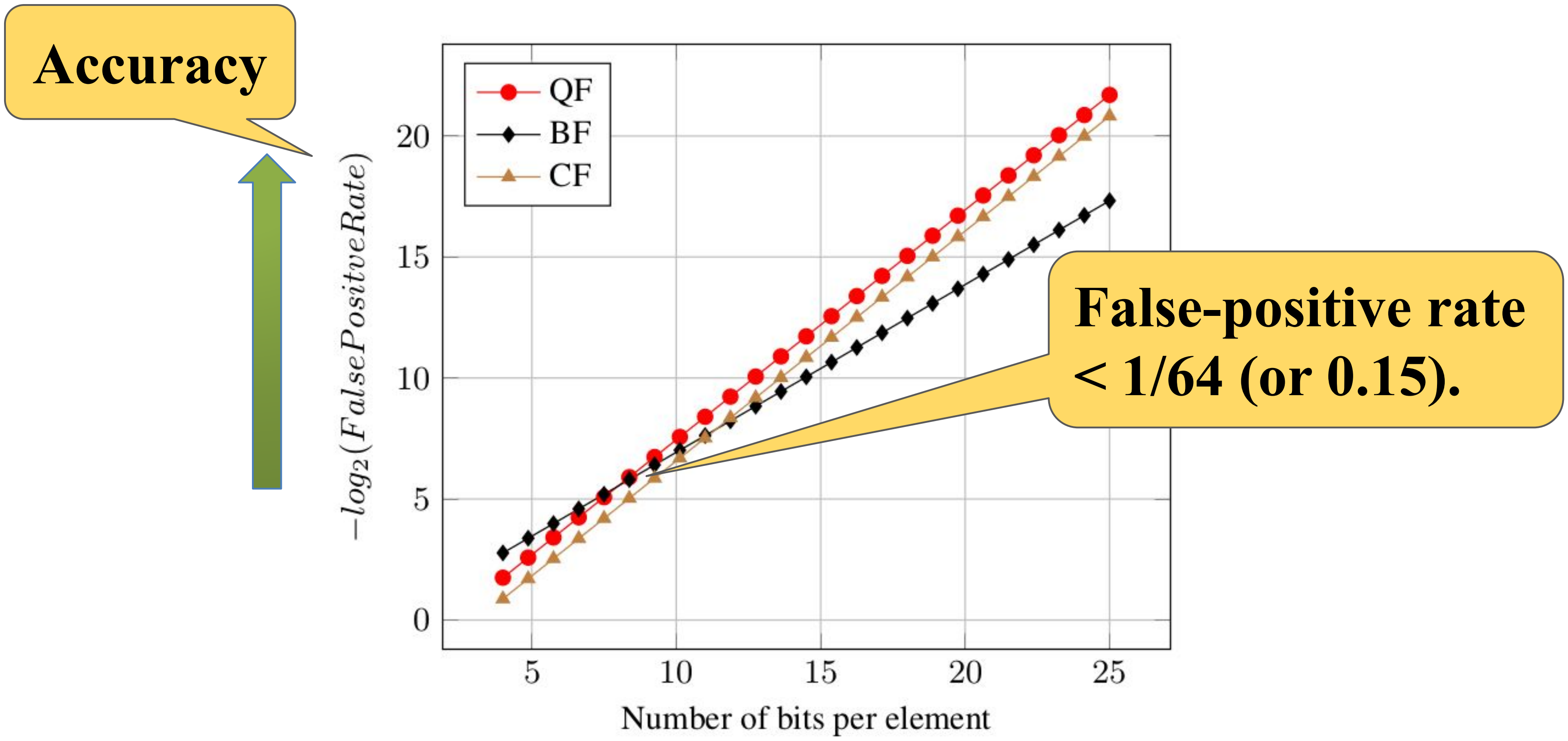
- Good cache locality
- Efficient scaling out-of-RAM
- Deletions
- Enumerability/Mergeability
- Resizing
- Maintains count estimates
- Uses variable-sized encoding for counts **[Counting quotient filter]**
    - **Asymptotically optimal space: $O(\sum |C(x)|)$**

Quotient filters use less space than Bloom filters for all practical configurations

	Quotient filter	Bloom filter	Optimal
Space	$\approx n \log(1/\epsilon) + 2.125n$	$\approx 1.44\, n \log(1/\epsilon)$	$\approx n \log(1/\epsilon) + \Omega(n)$
CPU cost	$O(1)$ expected	$\Omega(1/\epsilon)$	$O(1)$
Data locality	1 probe + scan	$\Omega(1/\epsilon)$ probes	$O(1)$ probes

The quotient filter has theoretical advantages over the Bloom filter

Quotient filters use less space than Bloom filters for all practical configurations

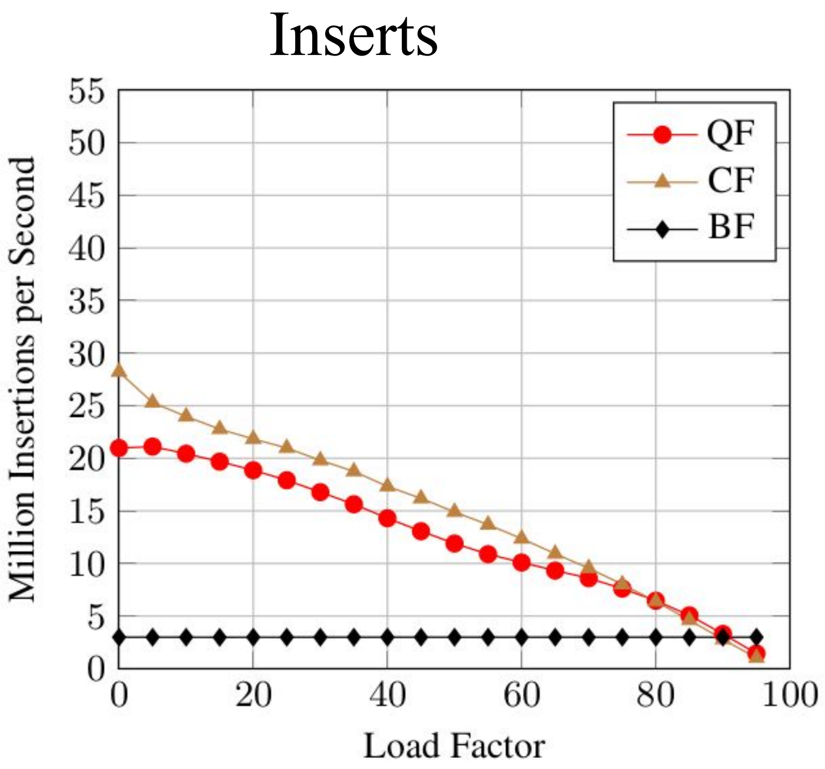


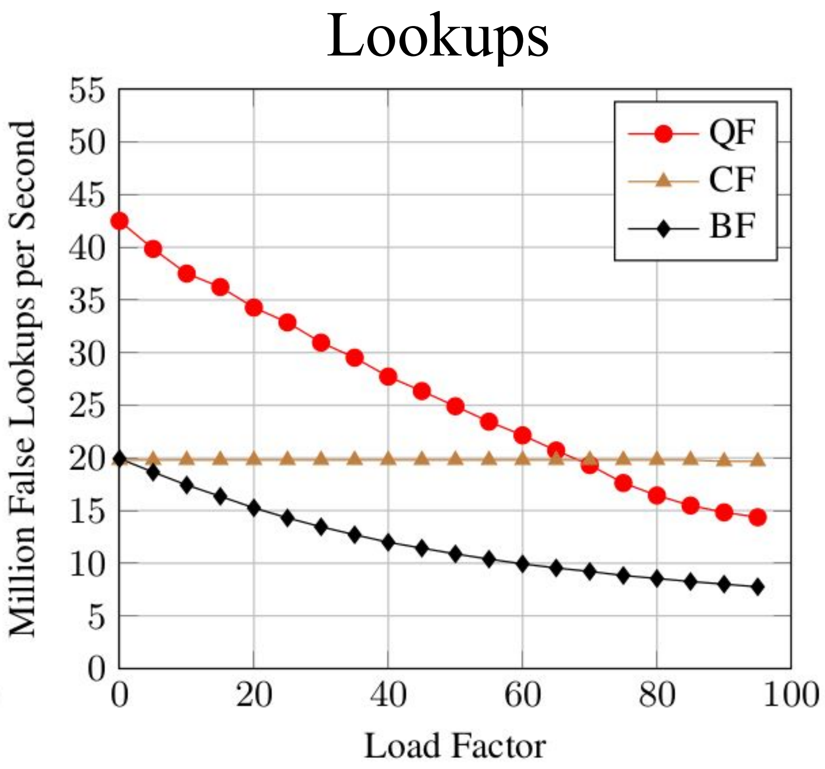
False-positive rate < 1/64 (or 0.15).

Bloom filter: ~*1.44 log*(*1/ε*) bits/element.

Quotient filter: ~*2.125* + *log*(*1/ε*) bits/element.

Quotient filters perform better (or similar) to other non-counting filters


Inserts


Lookups

- Insert performance is similar to the state-of-the-art non-counting filters
- Query performance is significantly fast at low load-factors and slightly slower at higher load-factors

Quotient filter's impact in computer science

Computational biology

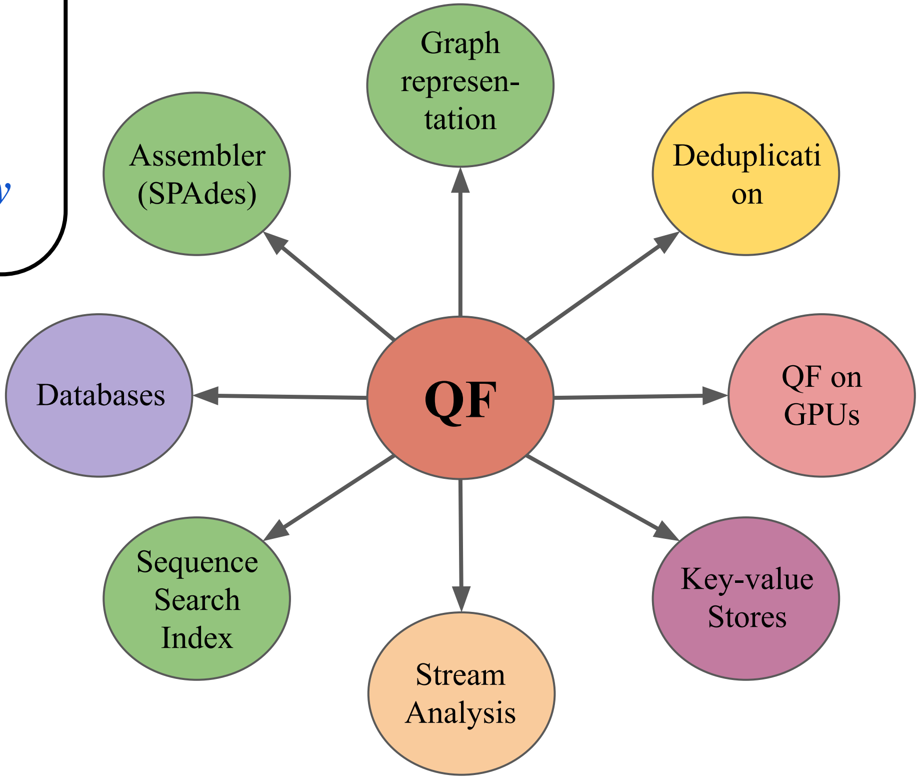
1. Squeakr
2. deBGR
3. Mantis
4. SPAdes assembler
5. Khmer software
6. MQF
7. VariantStore

Industry

1. VMware
2. Nutanix
3. Apocrypha
4. Hyrise
5. ***A data security startup***

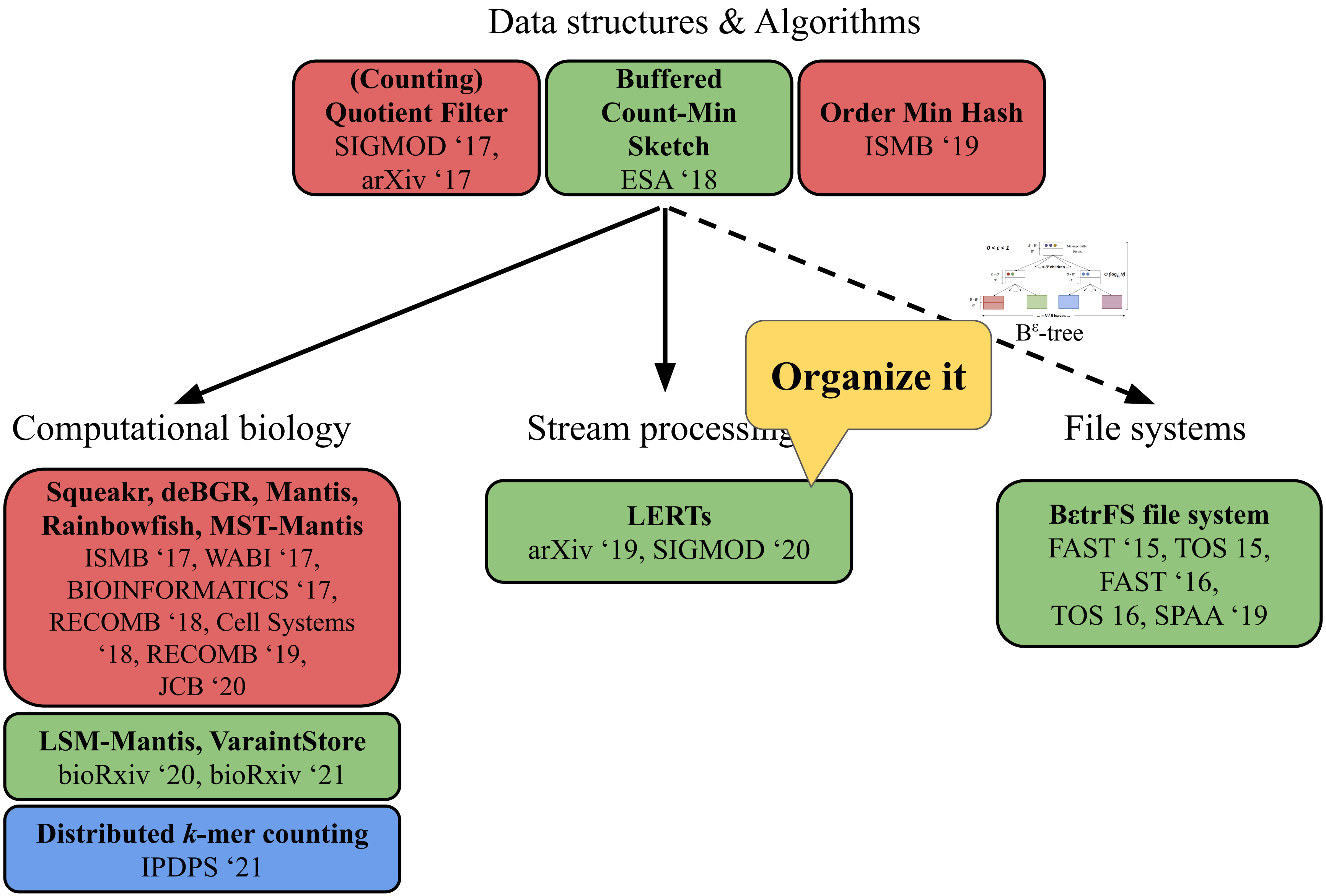
Databases/Systems

1. Counting on GPUs
2. Concurrent filters
3. Anomaly detection
4. BetrFS file system



Theoretically well-founded data structures can have a ***big impact*** on multiple subfields across ***academia and industry***

Learned “Shrink it”. Now “Organize it”



Data structures & Algorithms

(Counting) Quotient Filter
SIGMOD ‘17, arXiv ‘17

Buffered Count-Min Sketch
ESA ‘18

Order Min Hash
ISMB ‘19

B$^{\varepsilon}$-tree

Organize it

Computational biology

Squeakr, deBGR, Mantis, Rainbowfish, MST-Mantis
ISMB ‘17, WABI ‘17, BIOINFORMATICS ‘17, RECOMB ‘18, Cell Systems ‘18, RECOMB ‘19, JCB ‘20

LSM-Mantis, VaraintStore
bioRxiv ‘20, bioRxiv ‘21

Distributed *k*-mer counting
IPDPS ‘21

Stream processing

LERTs
arXiv ‘19, SIGMOD ‘20

File systems

BεtrFS file system
FAST ‘15, TOS 15, FAST ‘16, TOS 16, SPAA ‘19

Open problem in stream processing

- A **high-speed stream** of key-value pairs arriving over time
- **Goal:** report every key **as soon as** it appears ***T* times** without missing any
- Firehose benchmark (Sandia National Lab) simulates the stream https://firehose.sandia.gov/

Why should we care about this problem

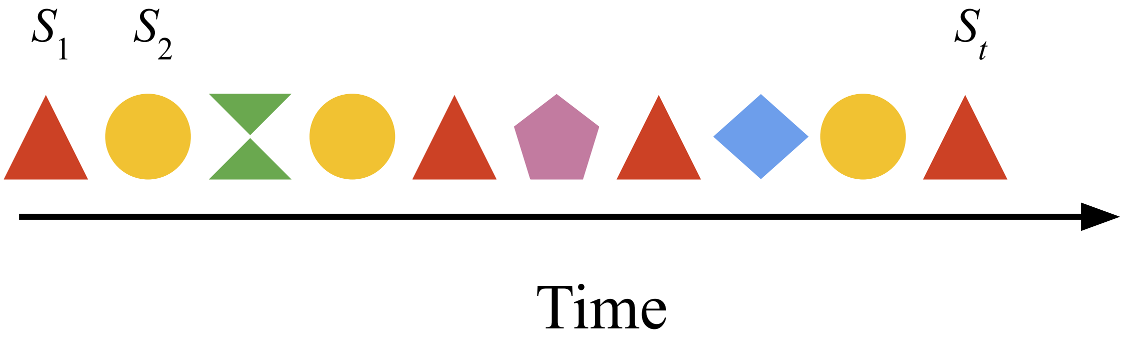
Defense systems for cyber security monitor high-speed streams for malicious traffic ➡ **Catch all malicious events**

Malicious traffic forms a small portion of the stream ➡ **Small reporting threshold**

Automated systems take defensive actions for every reported event ➡ **Minimize false positives**

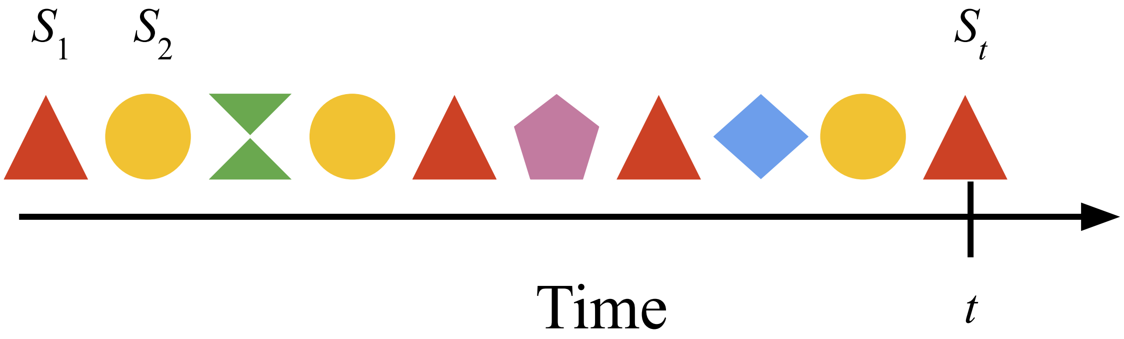
Timely event detection problem

- Stream of elements arrive over time



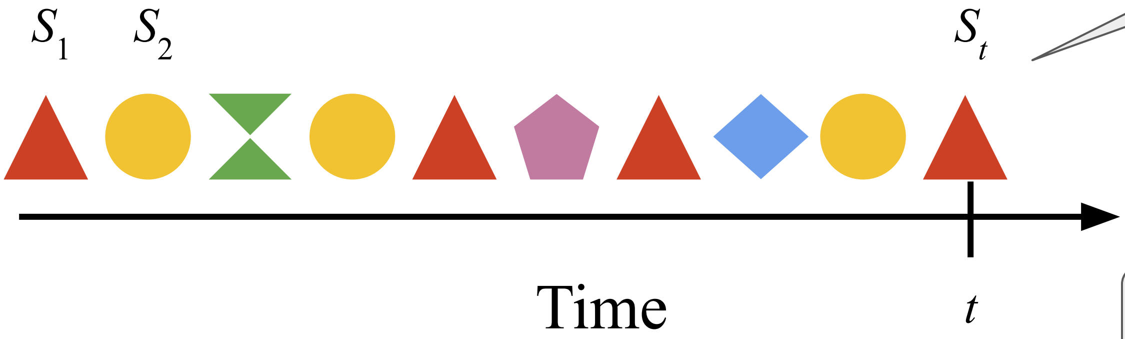
Timely event detection problem

- Stream of elements arrive over time
- An **event** occurs at time t if S_t occurs exactly T times in $(s_1, s_2 \ldots s_t)$



Timely event detection problem

- Stream of elements arrive over time
- An **event** occurs at time *t* if S_t occurs exactly *T* times in $(s_1, s_2 \ldots . s_t)$

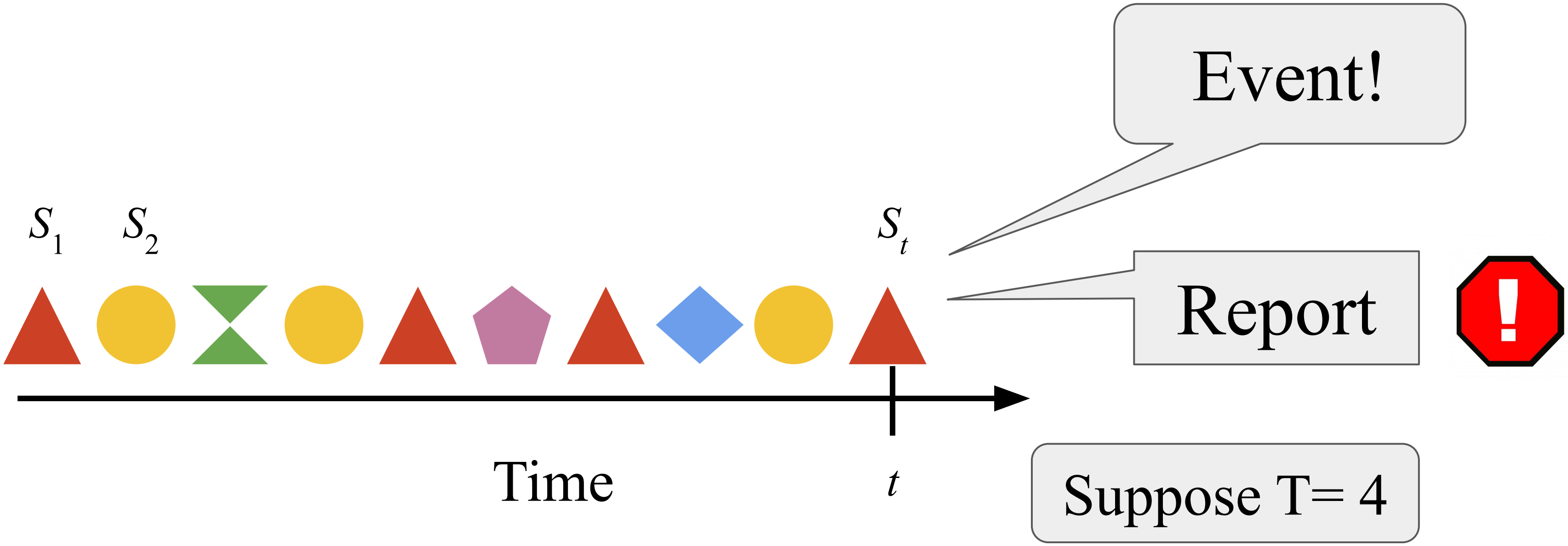


Event!

Suppose T= 4

Timely event detection problem

- Stream of elements arrive over time
- An **event** occurs at time *t* if S_t occurs exactly *T* times in $(s_1, s_2 \ldots . s_t)$
- In **timely event-detection problem (TED)**, we want to report all events shortly after they occur.



Suppose T= 4

Features we need in the solution

- Stream is large (e.g., terabytes) and high-speed (millions/sec)

High throughput ingestion

Features we need in the solution

- Stream is large (e.g., terabytes) and high-speed (millions/sec)

High throughput ingestion

- Events are high-consequence real-life events

No false-negatives; few false-positives

Timely reporting (real-time)

Features we need in the solution

- Stream is large (e.g., terabytes) and high-speed (millions/sec)

High throughput ingestion

- Events are high-consequence real-life events

No false-negatives; few false-positives

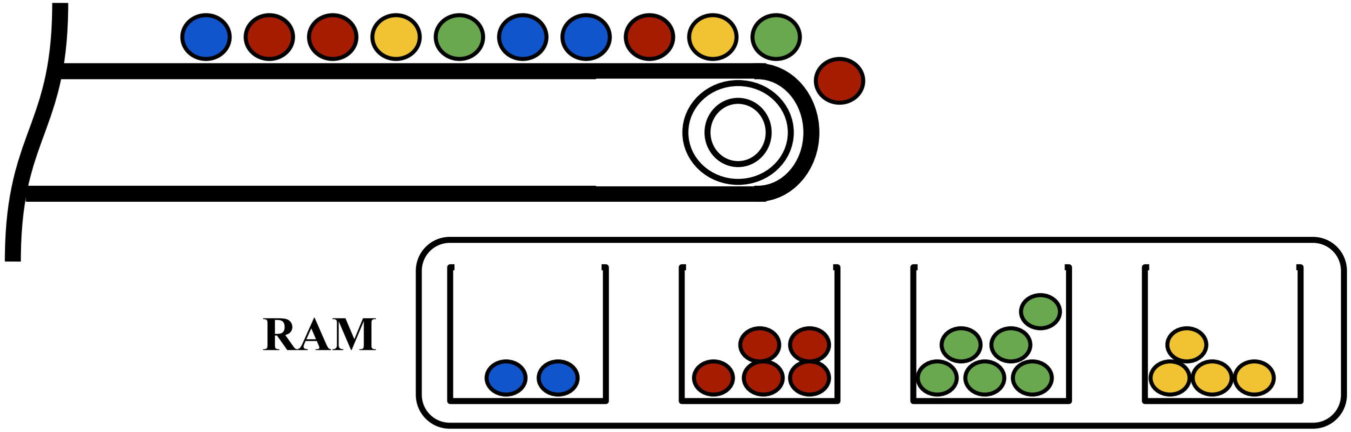
Timely reporting (real-time)

- Malicious traffic forms a small portion of the stream

Very small reporting thresholds

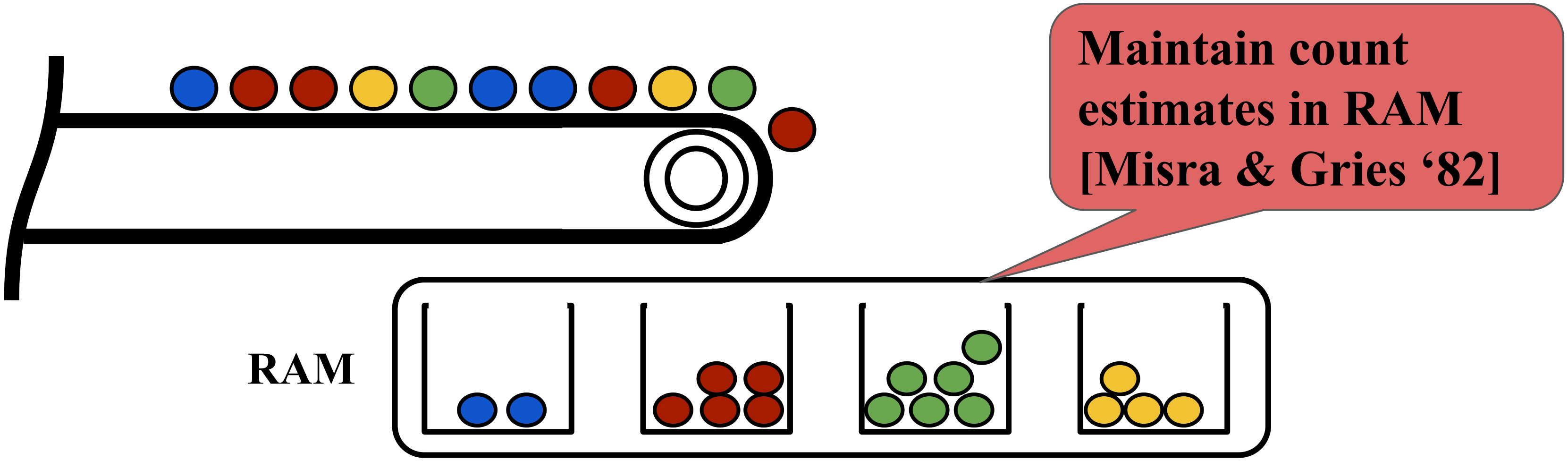
One-pass streaming has errors

- **Heavy hitter problem:** report items whose frequency ≥ φN
- Exact one-pass solution solution requires $\Omega(N)$ space



One-pass streaming has errors

- **Approximate solution**: report all items with count $\geq \varphi N$, none with $< (\varphi-\varepsilon)N$ [Alon et al. 96, Berinde et al. 10, Bhattacharyya et al. 16, Bose et al. 03, Braverman et al. 16, Charikar et al. 02, Cormode et al. 05, Demaine et al. 02, Dimitropoulos et al. 08, Larsen et al. 16, Manku et al. 02.]
- Approximate solutions requires: $\Omega(1/\varepsilon)$



Maintain count estimates in RAM [Misra & Gries '82]

Real time with false-positives!

One-pass streaming has errors

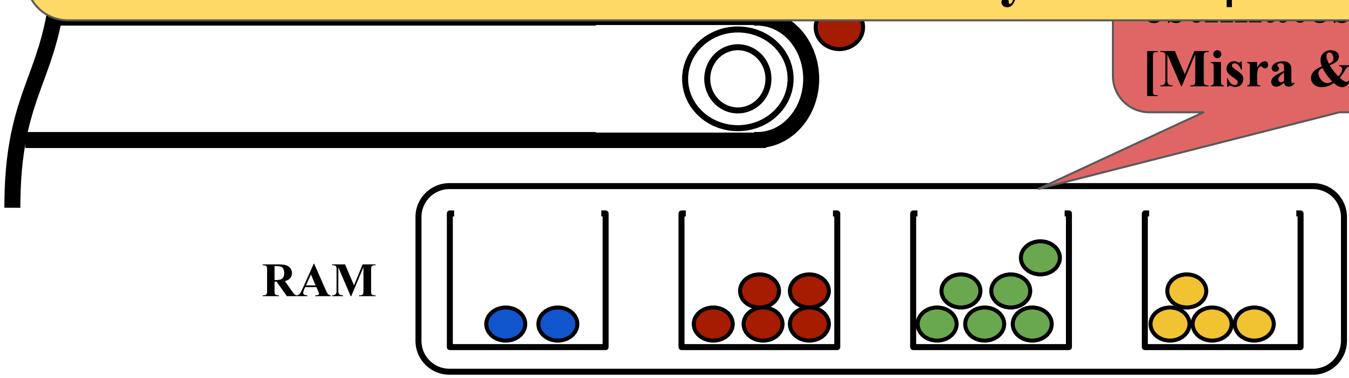
- **Approximate solution**: report all items with count ≥ $\boldsymbol{\varphi N}$**,** none with $< (\varphi-\varepsilon)N$ [Alon et al. 96, Berinde et al. 10, Bhattacharyya et al. 16, Bose et al. 03, Braverman et al. 16, Charikar et al. 02, Cormode et al. 05, Demaine et al. 02, Dimitropoulos et al. 08, Larsen et al. 16, Manku et al. 02.]
- Approximate solutions requires: $\boldsymbol{\Omega(1/\varepsilon)}$

For Sandia, φN is a small constant (e.g., 24),
So $\Omega(1/\boldsymbol{\varepsilon})$ is very very large!!
Can’t solve in RAM for very small φ

t

AM

[Misra & Gries ‘82]



Real time with false-positives!

One-pass solution has:

- Stream is large (e.g., terabytes) and high-speed (millions/sec)

High throughput ingestion ✓

- Events are high-consequence real-life events

No false-negatives; few false-positives ✓

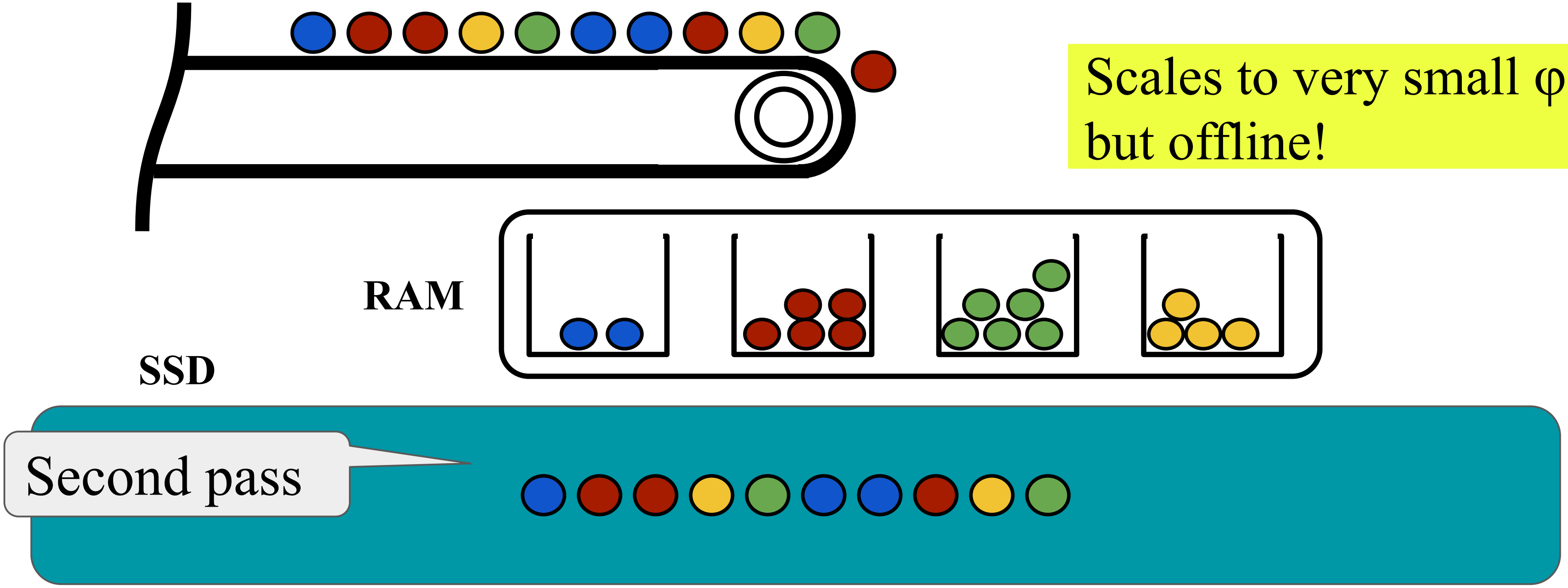
Timely reporting (real-time) ✓

- Malicious traffic forms a small portion of the stream

Very small reporting thresholds ✗

Two-pass streaming isn't real-time

- A second pass over the stream can get rid of errors
- Store the stream on SSD and access it later



Scales to very small φ but offline!

Two-pass solution has:

- Stream is large (e.g., terabytes) and high-speed (millions/sec)

High throughput ingestion ?

- Events are high-consequence real-life events

No false-negatives; few false-positives ✓

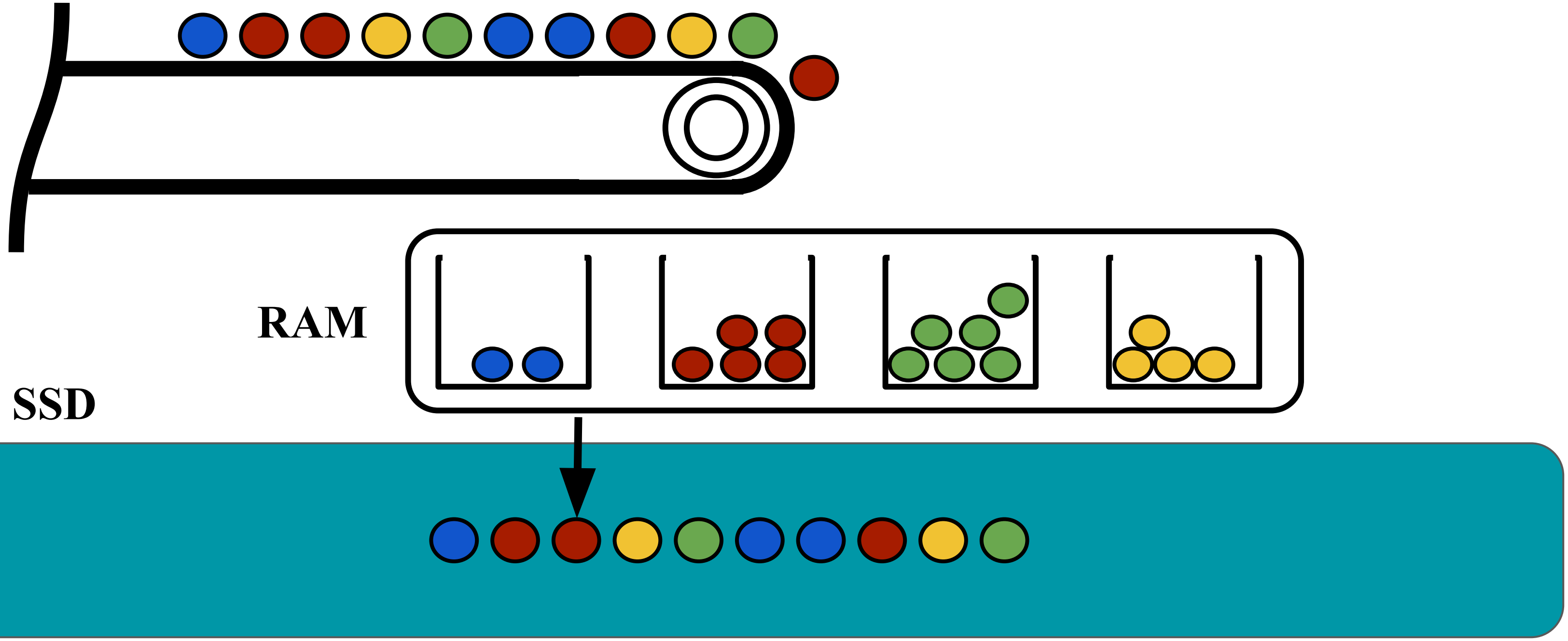
Timely reporting (real-time) ✗

- Malicious traffic forms a small portion of the stream

Very small reporting thresholds ✓

If data is stored: why not access it?

Why wait for second pass?



Idea: combine Streaming and EM

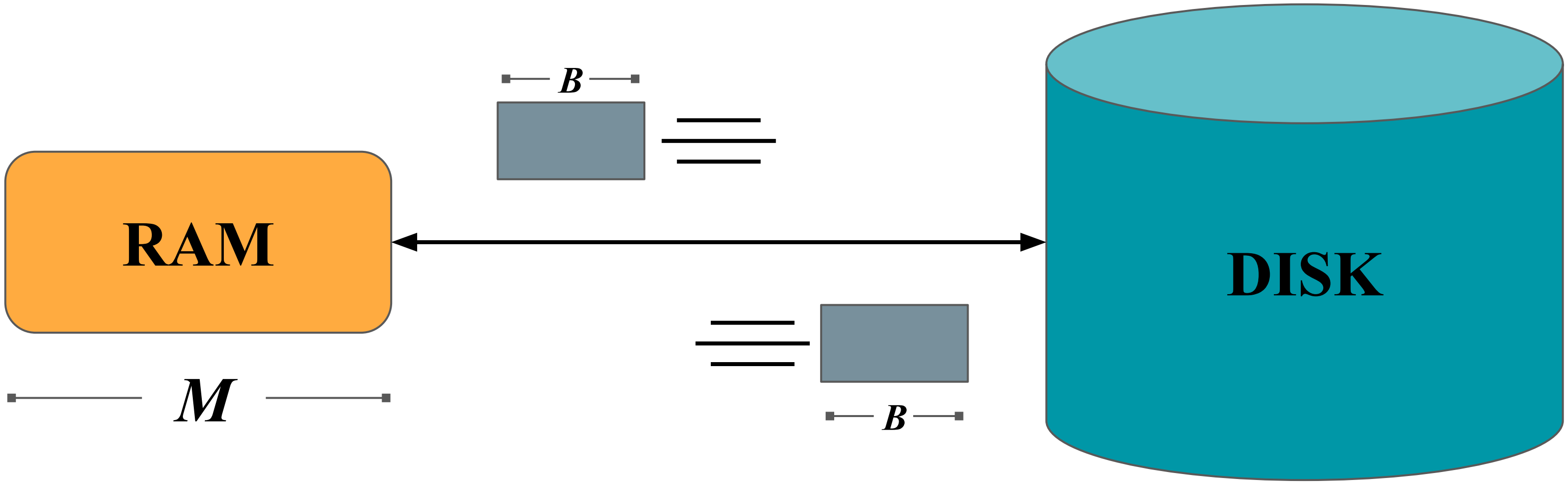
Use an efficient external-memory counting data structure to scale Misra-Gries algorithm to SSDs

Streaming model

External memory algorithms

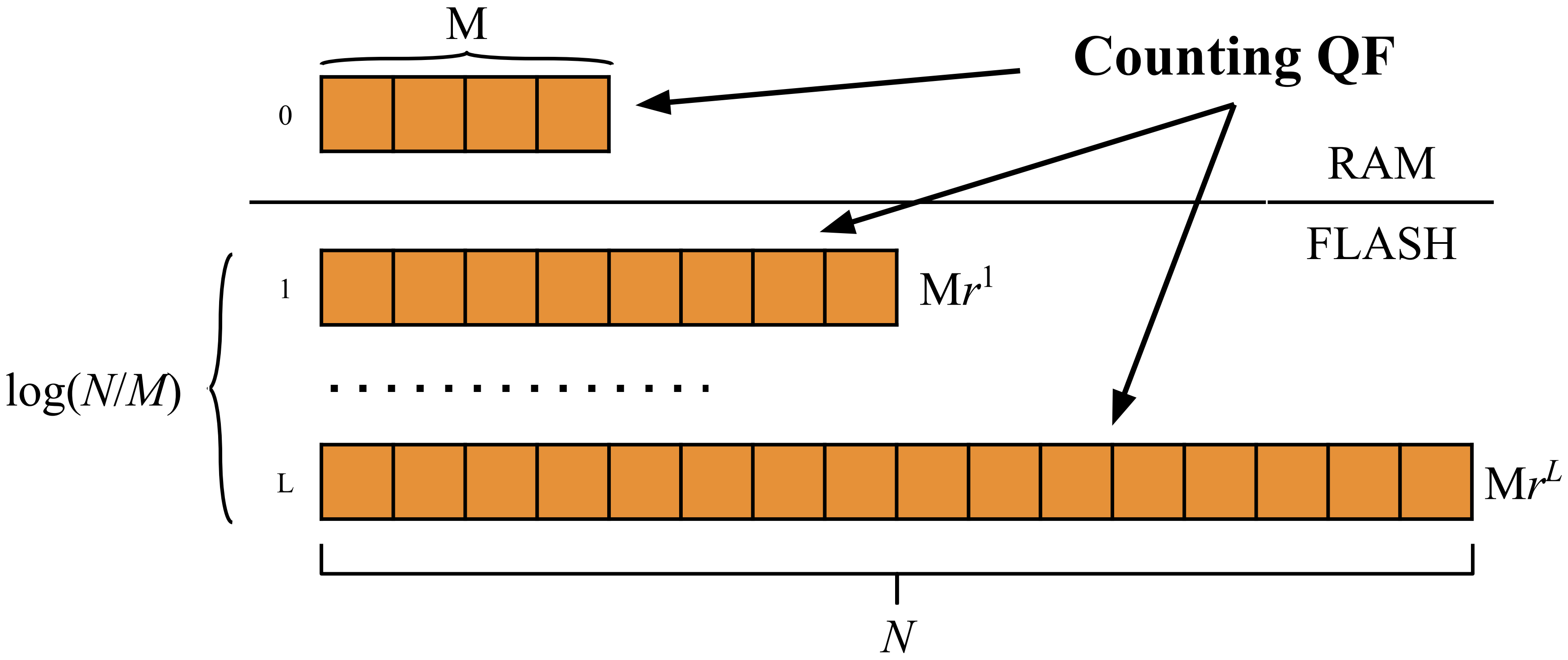
External memory model [Aggarwal+Vitter '08]

- **How computations work:**
  - Data is transferred in blocks between RAM and disk.
  - The number of block transfers dominate the running time.
- **Goal: Minimize number of block transfers**
  - Performance bounds are parameterized by block size B, memory size M, data size N.



Cascade filter: write-optimized quotient filter

[Bender et al. ‘12, Pandey et al. ‘17]



- The Cascade filter efficiently scales out-of-RAM
- It accelerates insertions at some cost to queries

More

Cascade filter operations

Insert	Query
$O\left(\frac{1}{B}\log\frac{N}{M}\right)$	$O\left(\log\frac{N}{M}\right)$

Cascade filter operations

Insert	Query
$O\left(\frac{1}{B}\log\frac{N}{M}\right)$	$O\left(\log\frac{N}{M}\right)$

< 1 I/O per observation

Cascade filter operations

Insert	Query
$O\left(\frac{1}{B}\log\frac{N}{M}\right)$	$O\left(\log\frac{N}{M}\right)$

< 1 I/O per observation

> 1 I/O per observation

Cascade filter doesn’t have real-time reporting

But every insert is also a query in real-time reporting!

Insert	Query
$O\left(\frac{1}{B}\log\frac{N}{M}\right)$	$O\left(\log\frac{N}{M}\right)$

$<$ 1 I/O per observation

$>$ 1 I/O per observation

Cascade filter doesn’t have real-time reporting

But every insert is also a query in real-time reporting!

Insert	Query
$O\left(\frac{1}{B}\log\frac{N}{M}\right)$	$O\left(\log\frac{N}{M}\right)$

Traditional cascade filter doesn’t solve the problem!

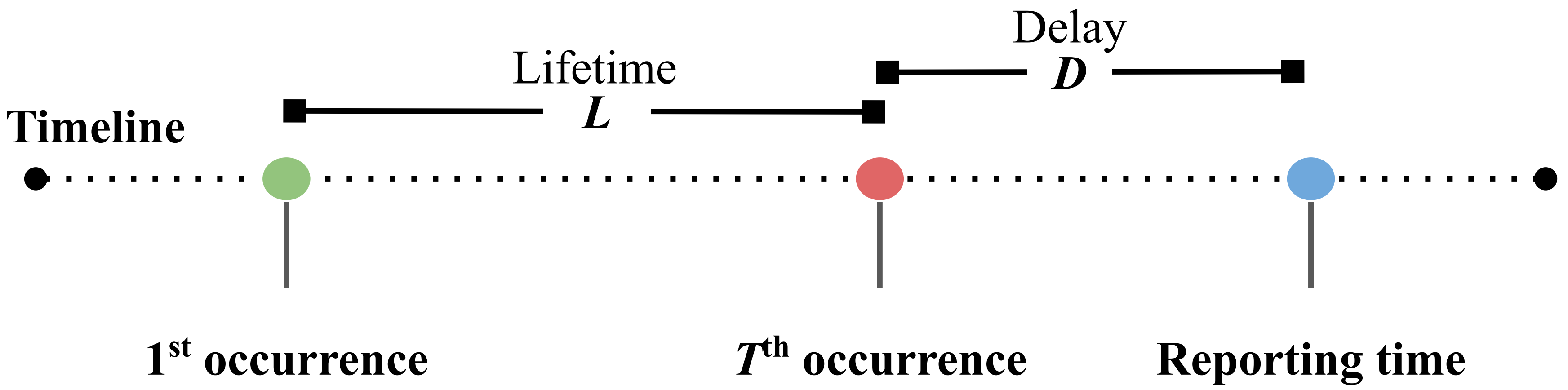
observation

observation

Idea: reporting with bounded delay

We define the time stretch of a report to be

$$\text{Time stretch} = 1 + \alpha = 1 + \frac{\text{Delay}}{\text{Lifetime}}$$



More

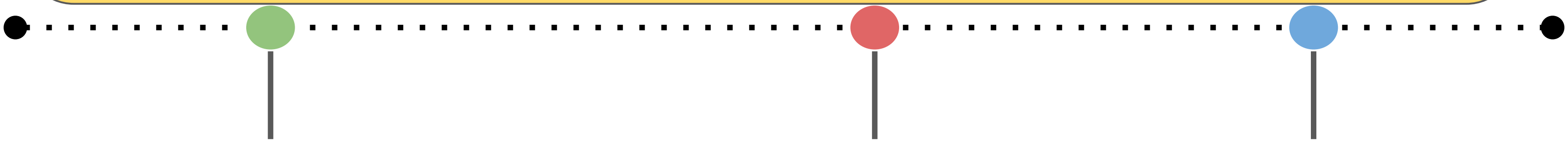
Idea: reporting with bounded delay

We define the time stretch of a report to be

$$\text{Time stretch} = 1 + \alpha = 1 + \frac{\text{Delay}}{}$$

Main idea: the longer the lifetime of an item, the more leeway we have in reporting it

Ti



1st occurrence

***T*th occurrence**

Reporting time

More

Leveled External-Memory Reporting Table (LERT) [Pandey '20]

- Given a stream of size N and $\varphi N > \Omega(N/M)$ the amortized cost of solving real-time event detection is

$$O\left(\left(\frac{1}{B}+\frac{1}{(\phi-1/M)N}\right)\log\frac{N}{M}\right)$$

- For a **constant** $\boldsymbol{\alpha}$, can support arbitrarily small thresholds φ with amortized cost

$$O\left(\frac{1}{B}\log\frac{N}{M}\right)$$

Takeaway: Online reporting comes at the cost of throughput but almost online reporting is essentially free!

Leveled External-Memory Reporting Table (LERT) [Pandey '20]

- Given a stream of size N and $\varphi N > \boldsymbol{\Omega(N/M)}$ the amortized cost of solving real-time event detection is

Can achieve timely reporting at effectively the optimal insert cost; no query cost

with amortized cost

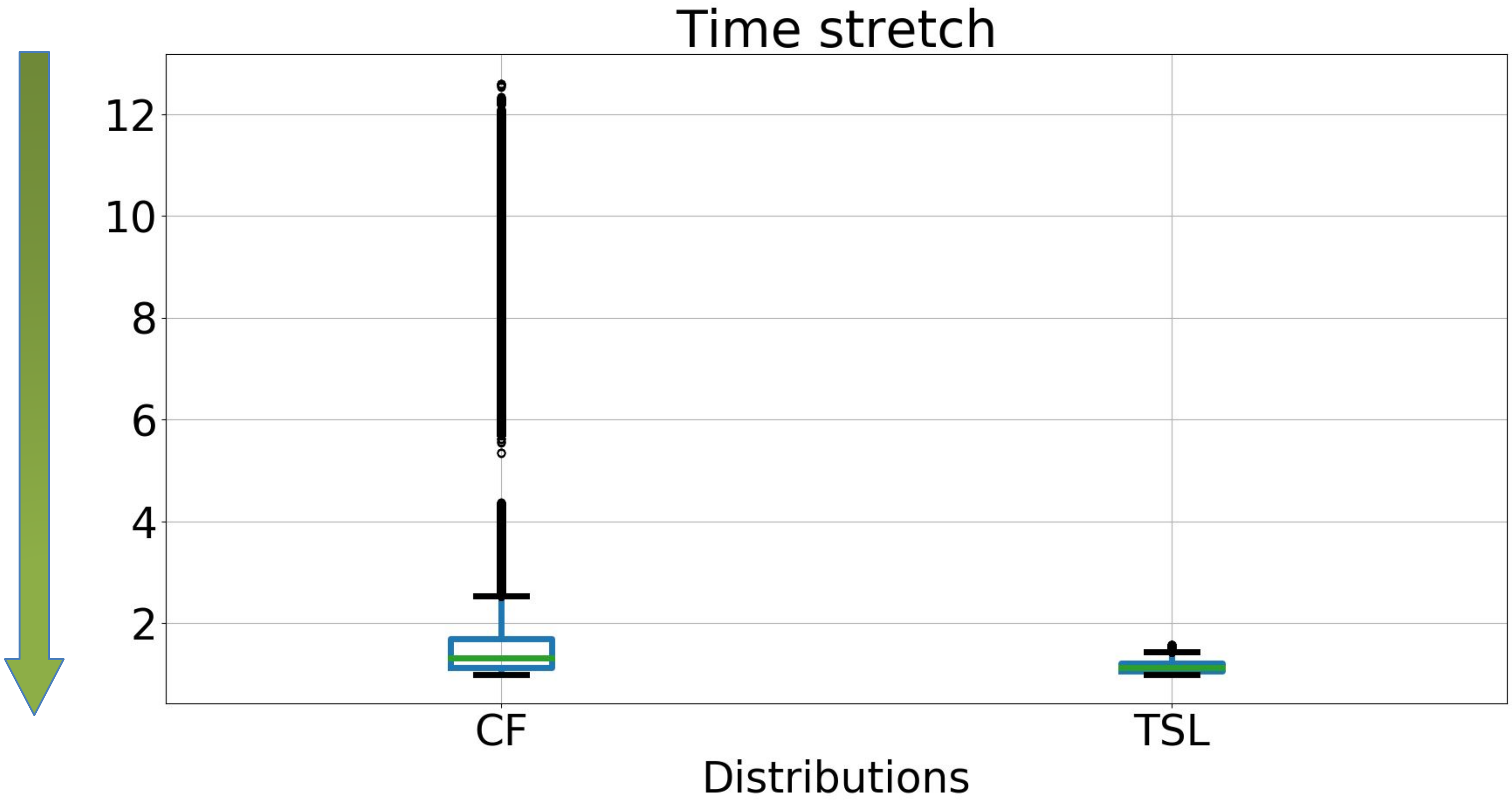
$$O\left(\frac{1}{B}\log\frac{N}{M}\right)$$

Takeaway: Online reporting comes at the cost of throughput but almost online reporting is essentially free!

Evaluation

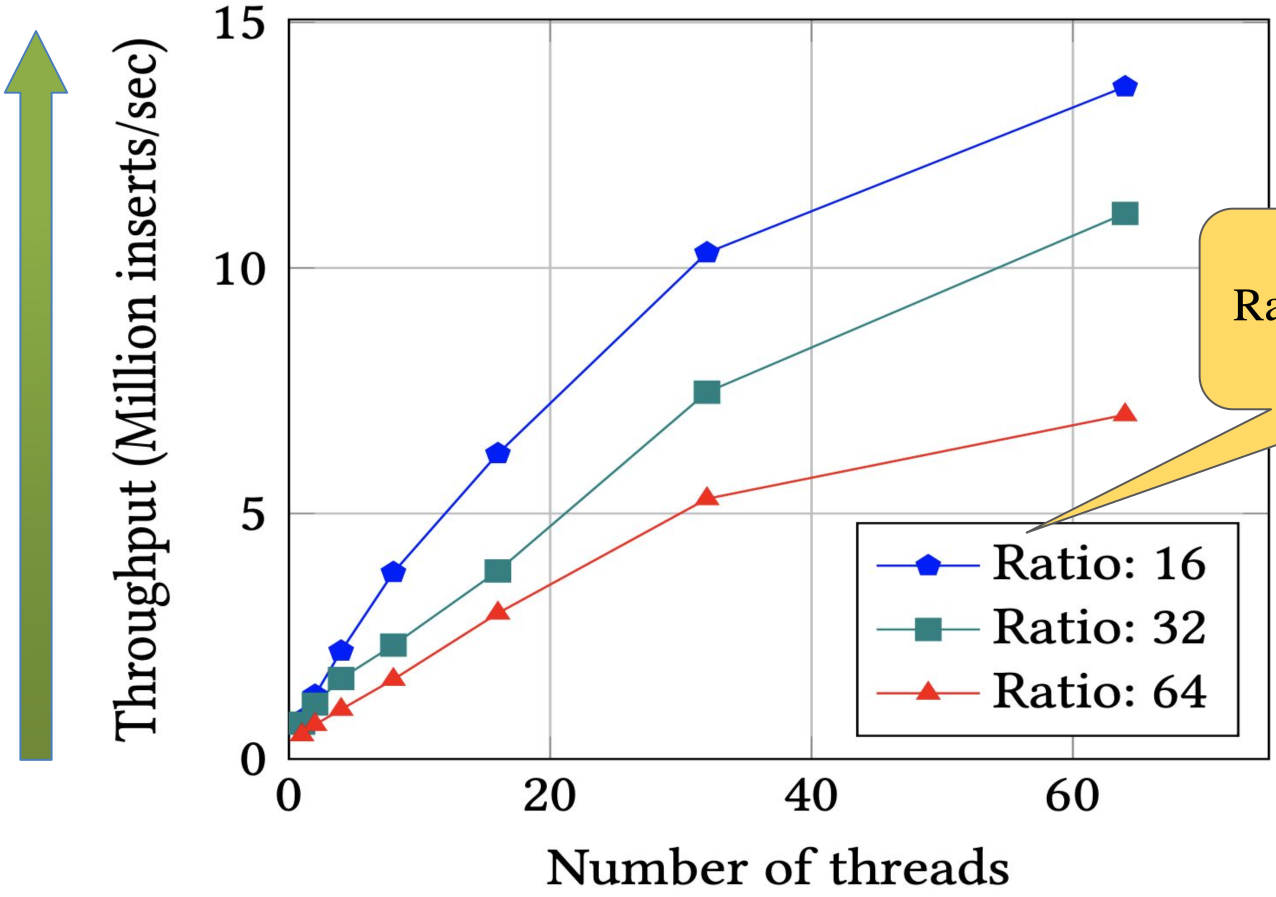
- Empirical timeliness
- High-throughput ingestion

Evaluation: empirical time stretch



Average time stretch is 43% smaller than theoretical upper bound.

Evaluation: scalability



$$\text{Ratio} = \frac{\text{Data Size}}{\text{RAM}}$$

The insertion throughput increases as we add more threads.

We can achieve > 13M insertions/sec.

LERT: supports scalable and real-time reporting

- Stream is large (e.g., terabytes) and high-speed (millions/sec)

High throughput ingestion ✓

- Events are high-consequence real-life events

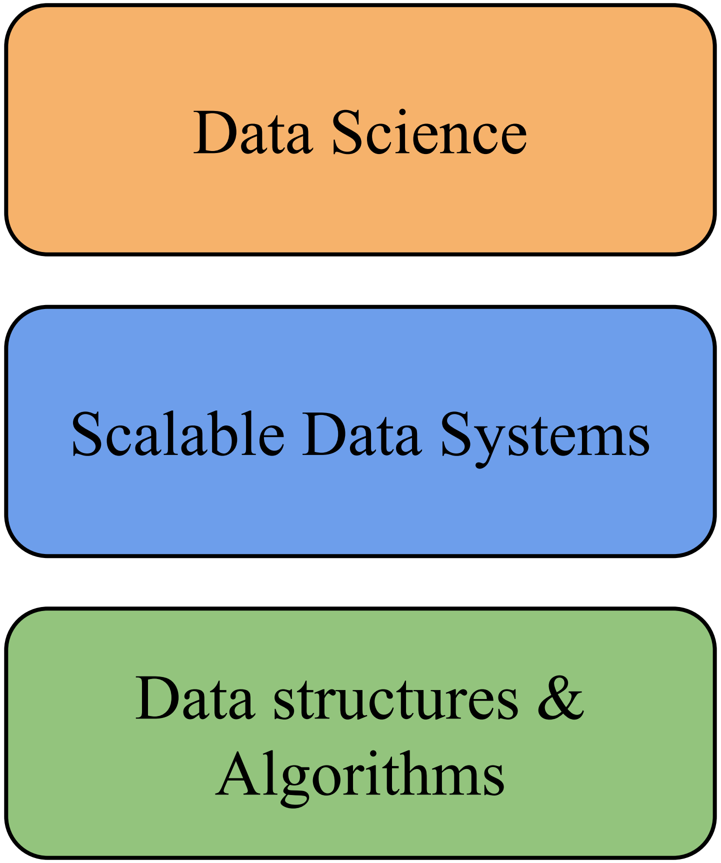
No false-negatives; few false-positives ✓

Timely reporting (real-time) ✓

- Malicious traffic forms a small portion of the stream

Very small reporting thresholds ✓

Future work overview



Data Science

Scalable Data Systems

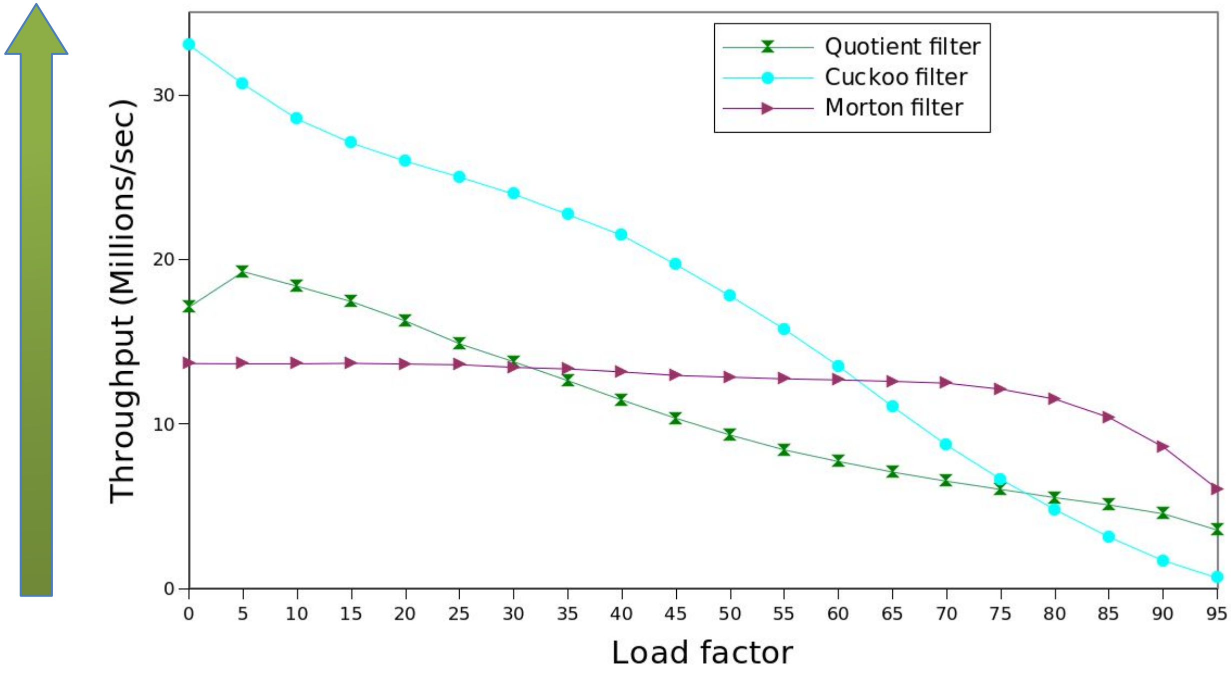
Data structures & Algorithms

Future work: Data Structures & Algorithms

Goal: Overcome ***decades-old*** data structure ***trade-offs*** using modern hardware and new algorithmic paradigms

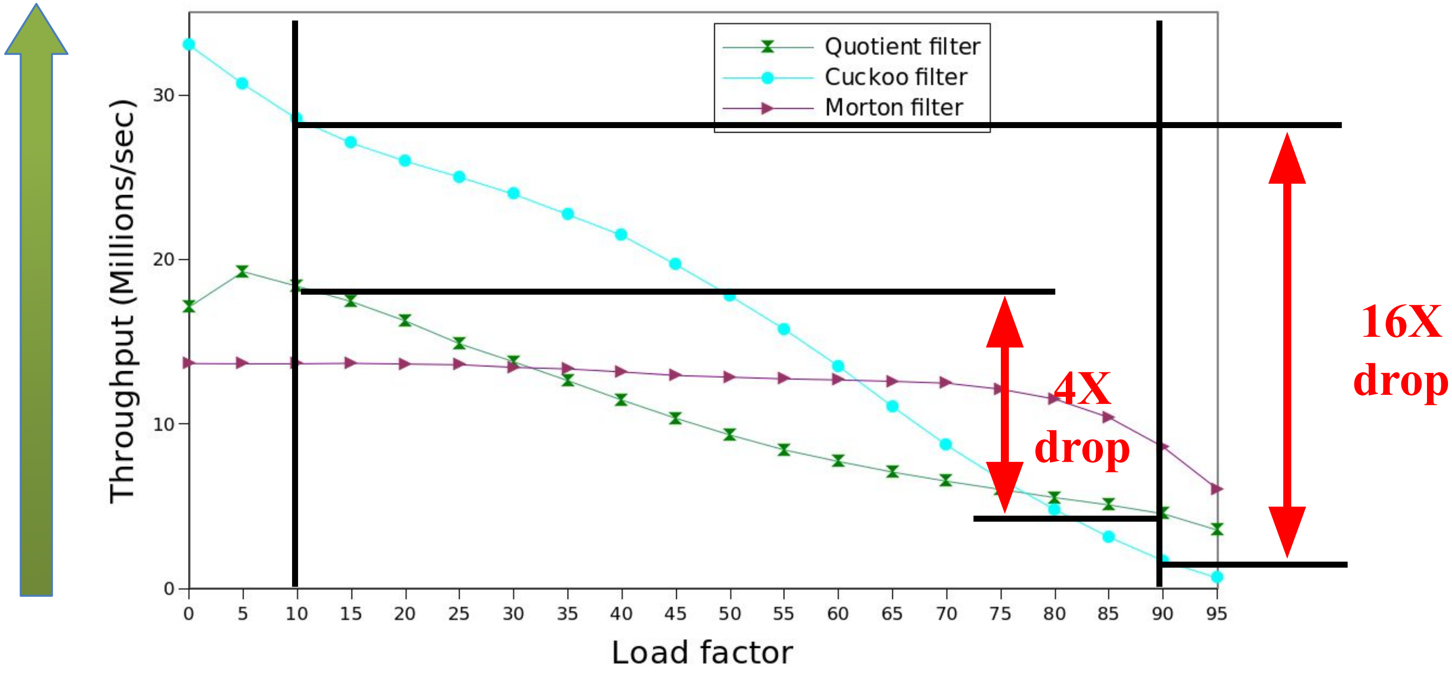
Trade-off 1: Insertion throughput degrades with load factor

Insertion throughput vs load factor of state-of-the-art filters



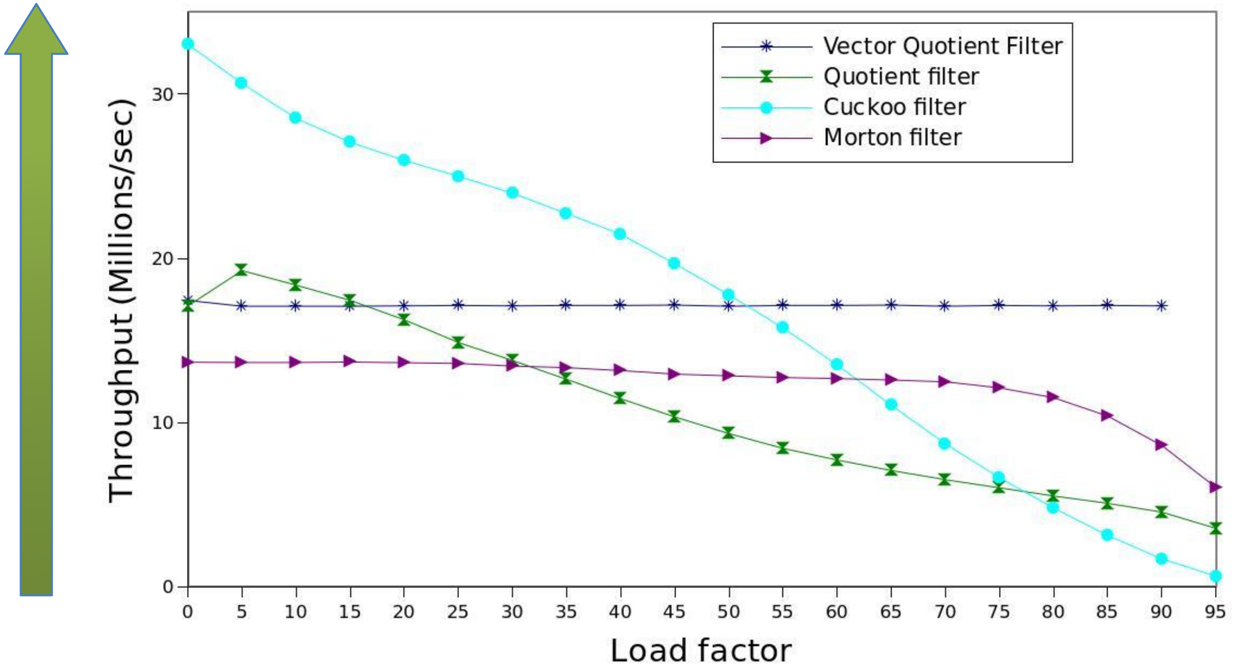
Many update-intensive applications (e.g., network caches, data analytics, etc.) maintain filters at high load factors

Trade-off 1: Insertion throughput degrades with load factor



Performance suffers due to high-overhead of ***collision resolution***

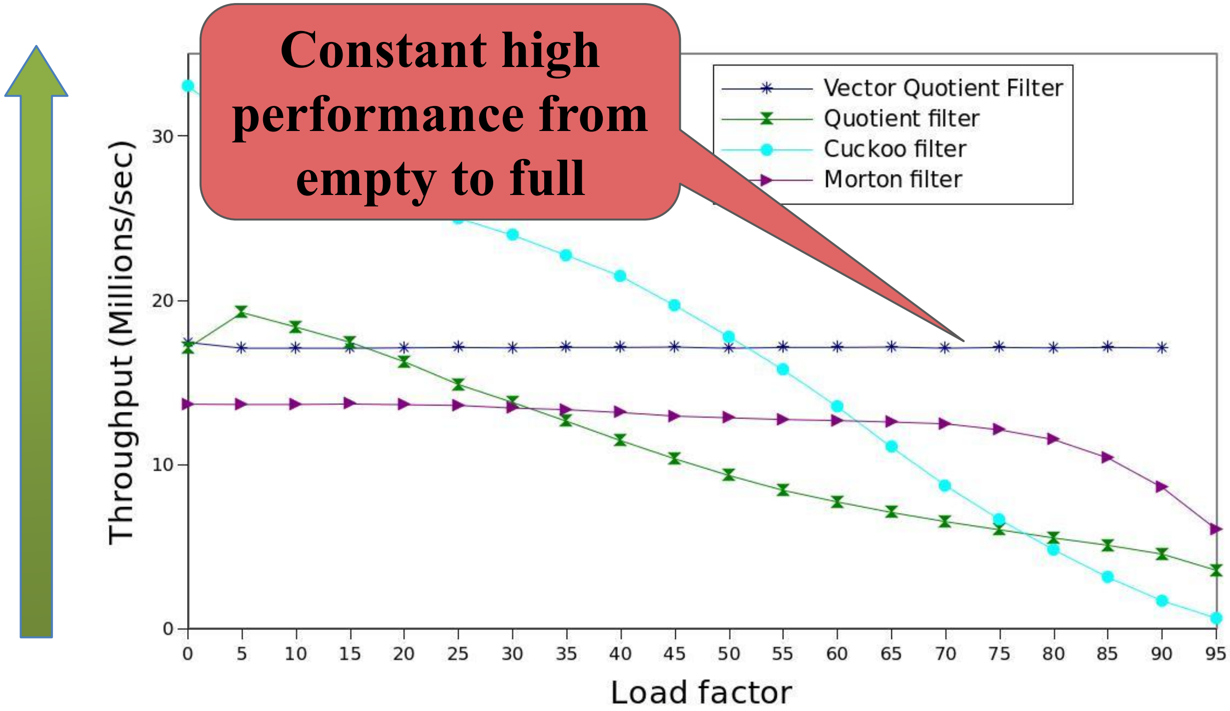
Combining techniques + new hardware



Combining hashing techniques (**Robin Hood + 2-choice hashing**)

Using ultra-wide vector operations (**AVX512-BW**)

Combining techniques + new hardware



Combining hashing techniques (**Robin Hood + 2-choice hashing**)

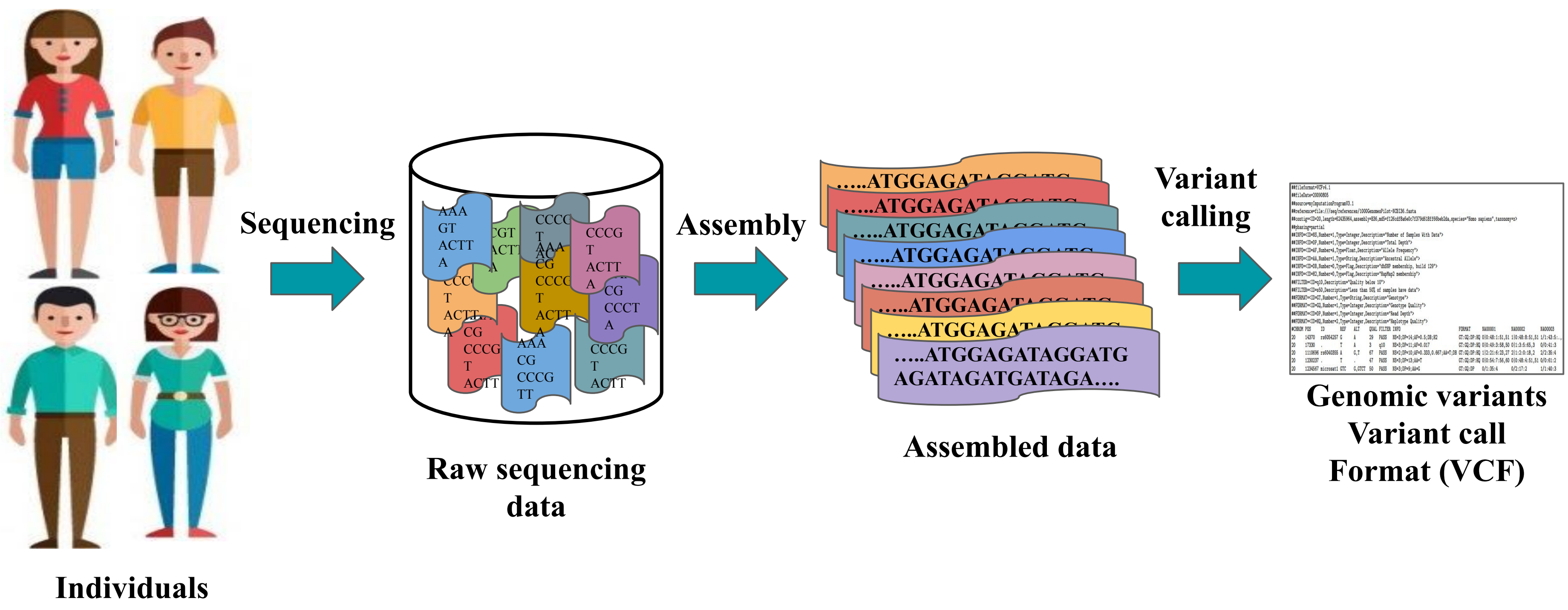
Using ultra-wide vector operations (**AVX512-BW**)

More

Future work: Data Systems

Goal: Build a ***population-scale*** index on variation data to enable downstream apps gain ***quick insights into variants***

Country-scale sequencing efforts produce huge amounts of sequencing data



Individuals

Sequencing

Raw sequencing data

Assembly

Assembled data

Variant calling

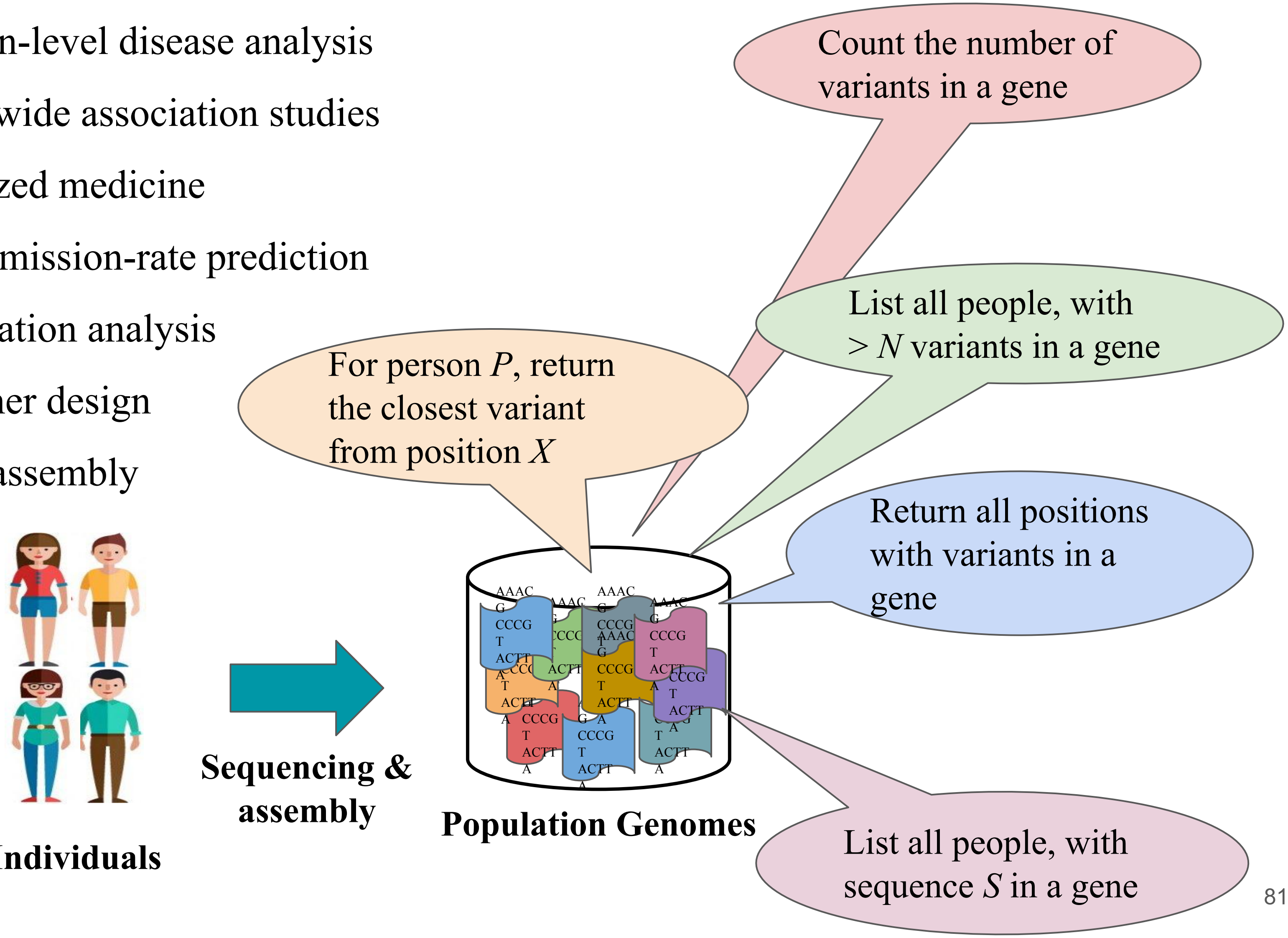
Genomic variants Variant call Format (VCF)

```
##fileformat=VCFv4.1
##fileDate=20090805
##source=myImputationProgramV3.1
##reference=file:///seq/references/1000GenomesPilot-NCBI36.fasta
##contig=<ID=20,length=62435964,assembly=B36,md5=f126cdf8a6e0c7f379d618ff66beb2da,species="Homo sapiens",taxonomy=x>
##phasing=partial
##INFO=<ID=NS,Number=1,Type=Integer,Description="Number of Samples With Data">
##INFO=<ID=DP,Number=1,Type=Integer,Description="Total Depth">
##INFO=<ID=AF,Number=A,Type=Float,Description="Allele Frequency">
##INFO=<ID=AA,Number=1,Type=String,Description="Ancestral Allele">
##INFO=<ID=DB,Number=0,Type=Flag,Description="dbSNP membership, build 129">
##INFO=<ID=H2,Number=0,Type=Flag,Description="HapMap2 membership">
##FILTER=<ID=q10,Description="Quality below 10">
##FILTER=<ID=s50,Description="Less than 50% of samples have data">
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
##FORMAT=<ID=GQ,Number=1,Type=Integer,Description="Genotype Quality">
##FORMAT=<ID=DP,Number=1,Type=Integer,Description="Read Depth">
##FORMAT=<ID=HQ,Number=2,Type=Integer,Description="Haplotype Quality">
#CHROM POS     ID        REF  ALT     QUAL FILTER INFO                              FORMAT      NA00001        NA00002        NA00003
20     14370   rs6054257 G    A       29   PASS   NS=3;DP=14;AF=0.5;DB;H2           GT:GQ:DP:HQ 0|0:48:1:51,51 1|0:48:8:51,51 1/1:43:5:.,
20     17330   .         T    A       3    q10    NS=3;DP=11;AF=0.017               GT:GQ:DP:HQ 0|0:49:3:58,50 0|1:3:5:65,3   0/0:41:3
20     1110696 rs6040355 A    G,T     67   PASS   NS=2;DP=10;AF=0.333,0.667;AA=T;DB GT:GQ:DP:HQ 1|2:21:6:23,27 2|1:2:0:18,2   2/2:35:4
20     1230237 .         T    .       47   PASS   NS=3;DP=13;AA=T                   GT:GQ:DP:HQ 0|0:54:7:56,60 0|0:48:4:51,51 0/0:61:2
20     1234567 microsat1 GTC  G,GTCT  50   PASS   NS=3;DP=9;AA=G                    GT:GQ:DP    0/1:35:4       0/2:17:2       1/1:40:3
```

- 1000 Genomes project [https://www.internationalgenome.org/]
- The Cancer Genome Atlas (TCGA) [https://portal.gdc.cancer.gov/]
- Genotype-Tissue Expression (GTEx) [https://gtexportal.org/home/]

Variation data analysis can improve downstream applications

- Population-level disease analysis
- Genome-wide association studies
- Personalized medicine
- Cancer remission-rate prediction
- Colocalization analysis
- PCR primer design
- Genome assembly



Count the number of variants in a gene

List all people, with > N variants in a gene

For person P, return the closest variant from position X

Return all positions with variants in a gene

List all people, with sequence S in a gene

Indexing in multiple coordinates is challenging

Reference-only indexes map positions only in the reference coordinate system

$$f(p_i, p_j) \rightarrow (v_i \ldots v_n),\ \text{where}\, p_i \leq p_j$$

Pan-genome analysis involves queries based on sample coordinate systems

$$\begin{array}{c}\text{Num}\\\text{Samples}\end{array}\left\{\begin{array}{l} f_1(p_i, p_j) \rightarrow (v_i \ldots v_n),\ \text{where}\, p_i \leq p_j \\ \vdots \\ f_s(p_i, p_j) \rightarrow (v_i \ldots v_n),\ \text{where}\, p_i \leq p_j \end{array}\right.$$

Maintaining thousands of mappings ***increases*** computational ***complexity*** and ***memory footprint***

Limits scalability to population-scale data

Indexing in multiple coordinates is challenging

Reference-only indexes map positions only in the reference coordinate system

$$f(p_i, p_j) \rightarrow (v_i \ldots v_n),\ \text{where}\ p_i \leq p_j$$

Pan-genome analysis involves queries based on sample coordinate systems

Existing systems don't support multiple coordinate systems. The ones that do, don't *scale* beyond a few thousand samples.

Nu
Sam

$$f_s(p_i, p_j) \rightarrow (v_i \ldots v_n),\ \text{where}\ p_i \leq p_j$$

Maintaining thousands of mappings ***increases*** computational ***complexity*** and ***memory footprint***

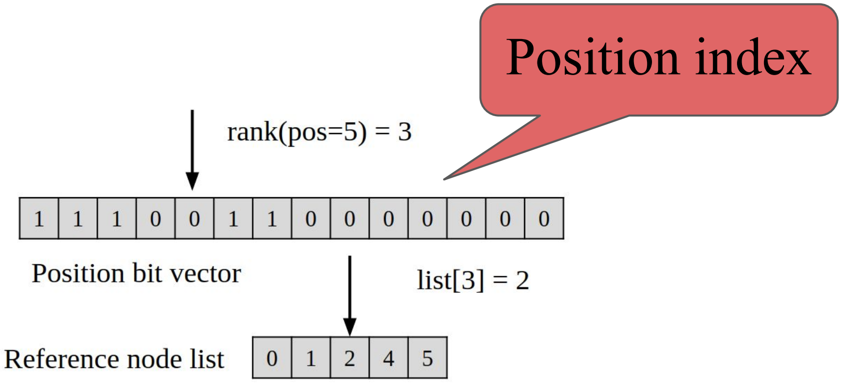
Limits scalability to population-scale data

An inverted index on the pan-genome graph

- Partition the variation graph based on coordinate ranges
- Store partitions on disk

} Queries often require loading 1-2 partitions

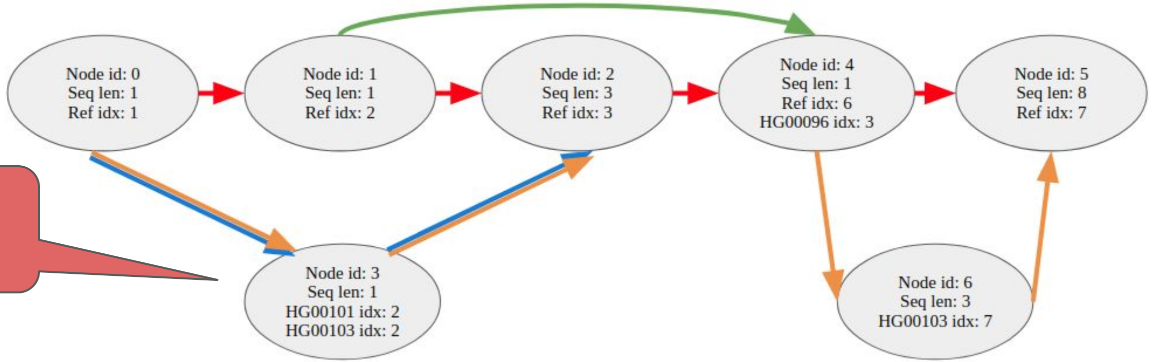
- Succinct index for reference coordinate system
- Local-graph exploration to map position from reference to sample coordinate



Position index

1	1	1	0	0	1	1	0	0	0	0	0	0	0

0	1	2	4	5

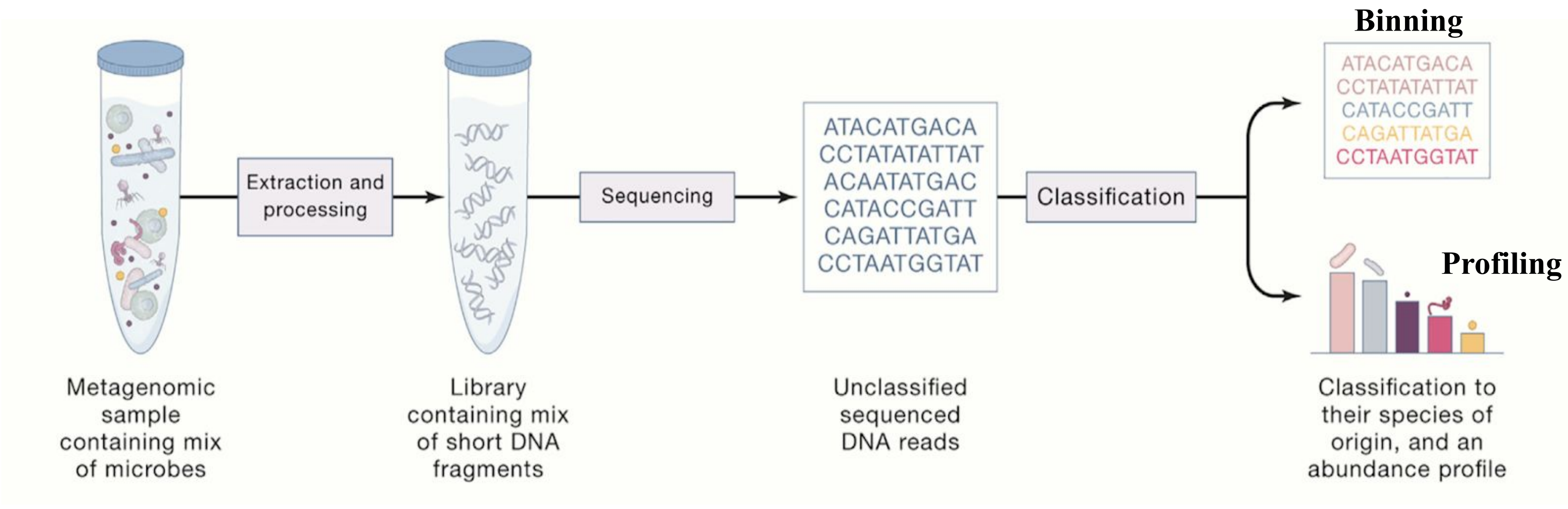


Variation graph

Future work: Data Science for genomics

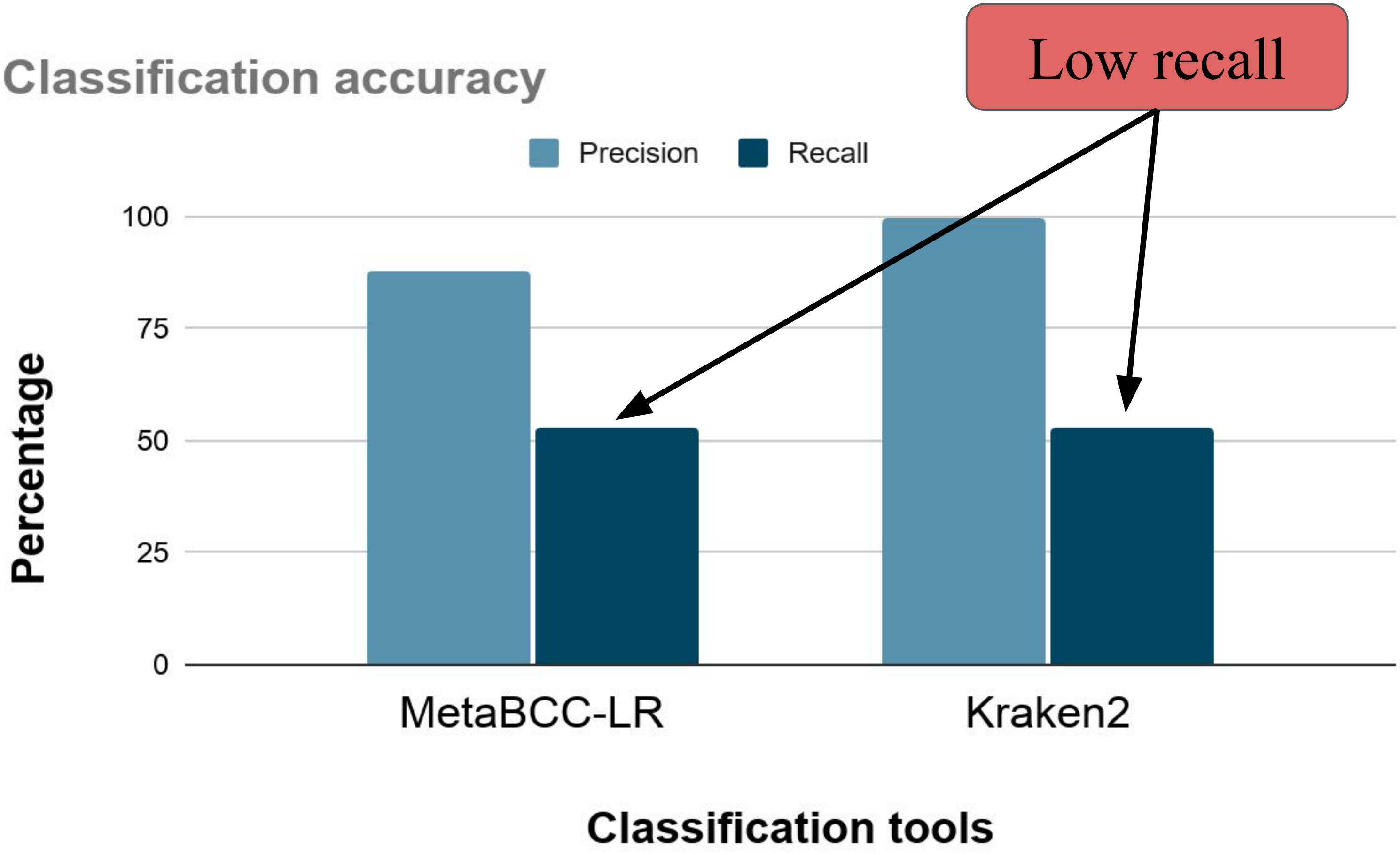
Goal: Classification of metagenomic reads and ***identification*** of ***novel species*** using ***graph neural networks*** (GNN)

Metagenomic classification pipeline


Binning
ATACATGACA
CCTATATATTAT
CATACCGATT
CAGATTATGA
CCTAATGGTAT
Profiling
Extraction and processing
Sequencing
ATACATGACA
CCTATATATTAT
ACAATATGAC
CATACCGATT
CAGATTATGA
CCTAATGGTAT
Classification
Metagenomic sample containing mix of microbes
Library containing mix of short DNA fragments
Unclassified sequenced DNA reads
Classification to their species of origin, and an abundance profile

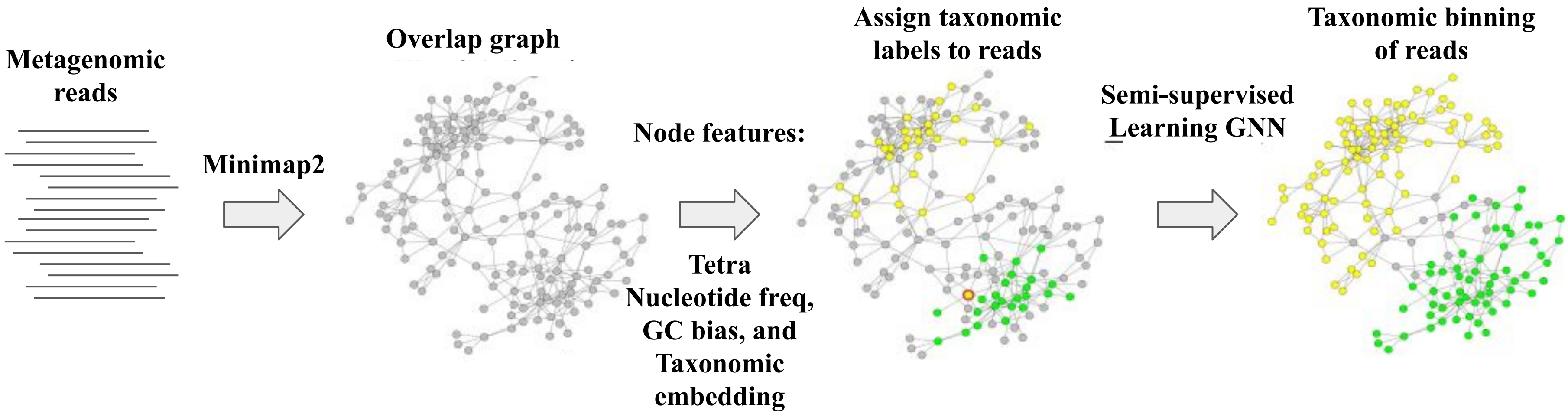
[Ye et al. 2019]

Existing techniques offer low recall



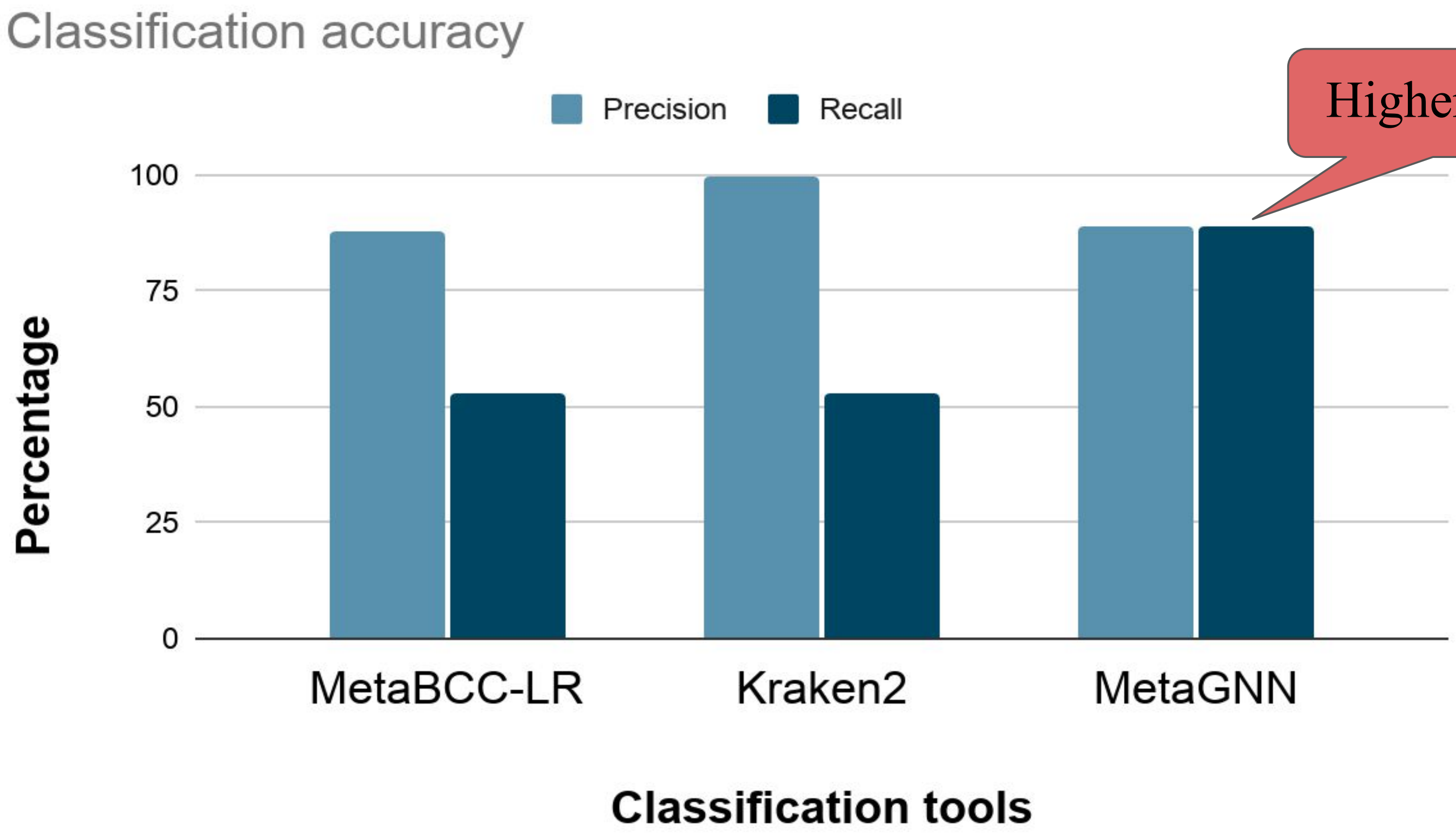
Classification is done based ***only on the read contents***

Use overlap relationship between reads


Metagenomic reads
Minimap2
Overlap graph
Node features:
Tetra Nucleotide freq, GC bias, and Taxonomic embedding
Assign taxonomic labels to reads
Semi-supervised Learning GNN
Taxonomic binning of reads

- Generate overlap graph: reads→nodes & overlap →edges
- Node features →Tetra nucleotide freq of reads
- Reference-based mapping as ground truth labels

Overlap graph + graph neural network (GNN)

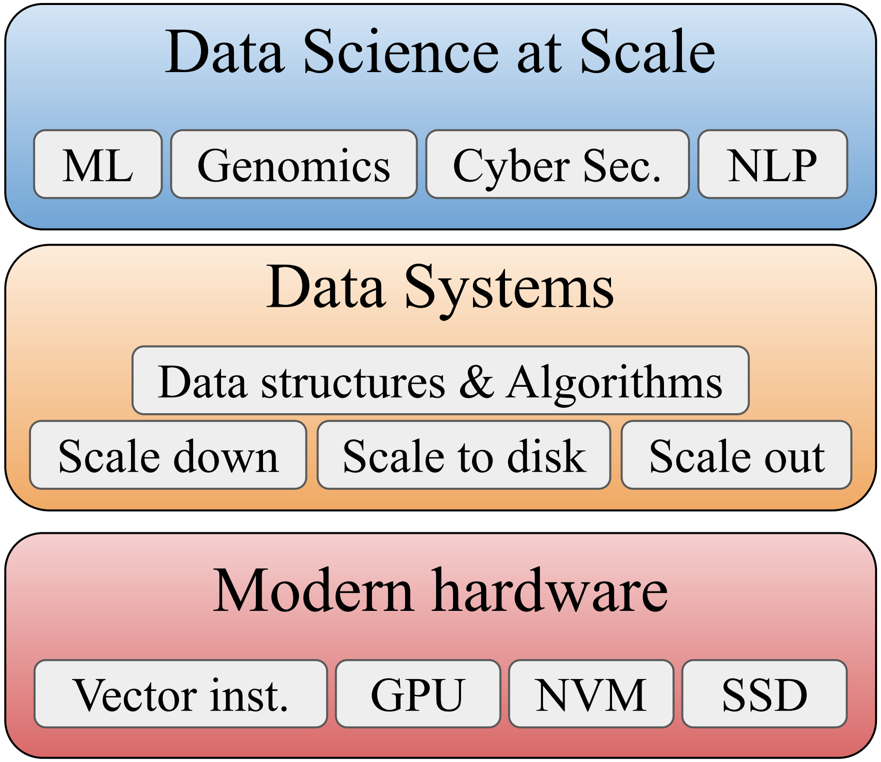


Higher recall

Can achieve high recall using graph learning

Conclusion

- Scalability of data management systems will be the biggest challenge in future
- Changing hardware give rise to new algorithmic paradigms



Data Science at Scale

ML | Genomics | Cyber Sec. | NLP

Data Systems

Data structures & Algorithms

Scale down | Scale to disk | Scale out

Modern hardware

Vector inst. | GPU | NVM | SSD

We need to ***redesign*** existing data structures to take full advantage of modern hardware and ***rebuild*** data systems to efficiently support ***future*** data science.

https://prashantpandey.github.io

Backup slides

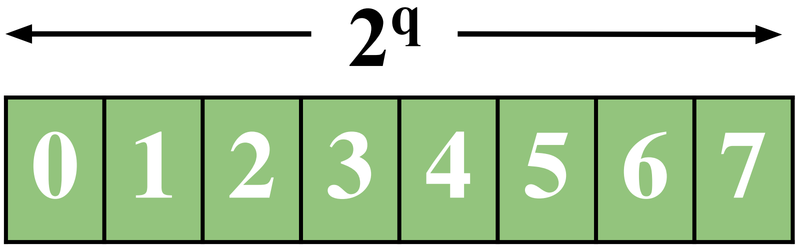
Quotient filter design

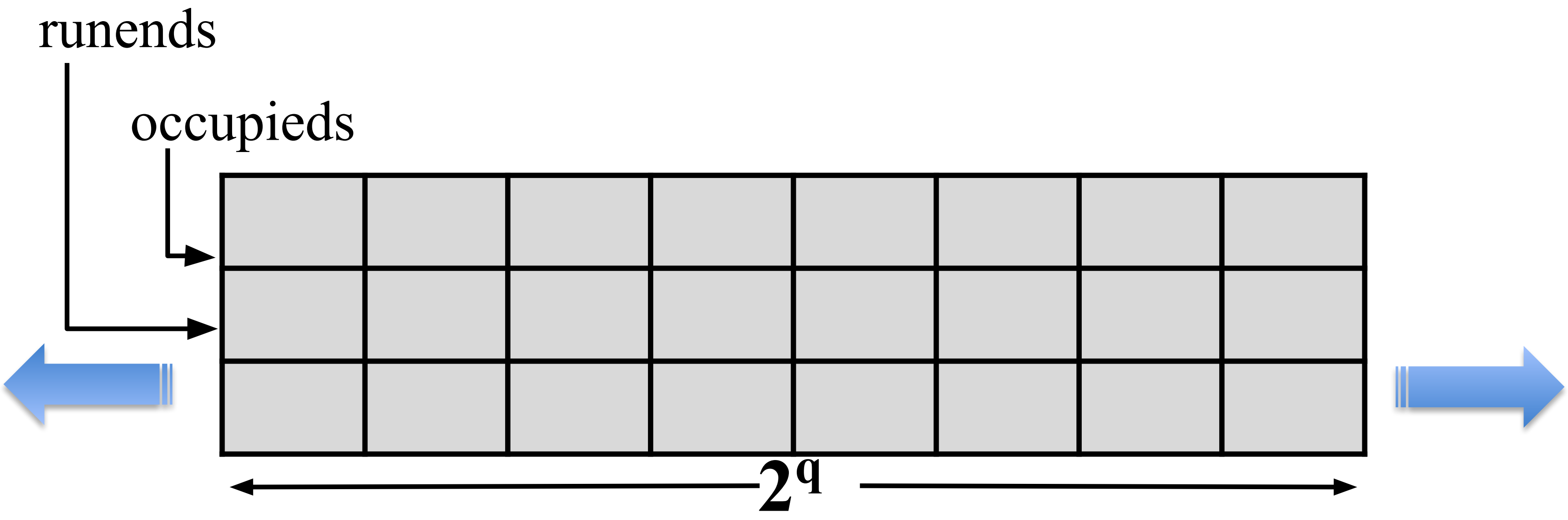
Implementation:

2 Meta-bits per slot.

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

Abstract Representation





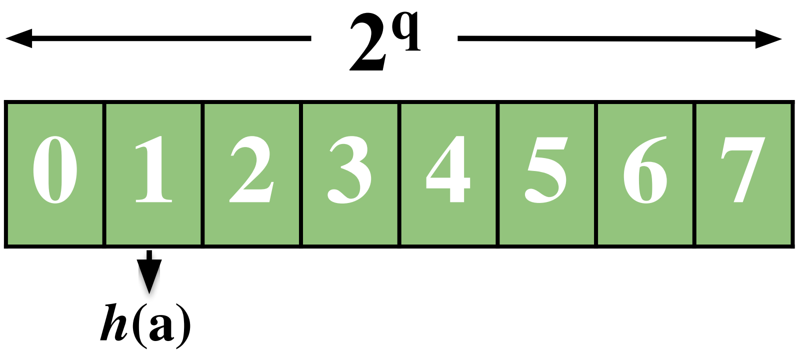
Quotient filter design

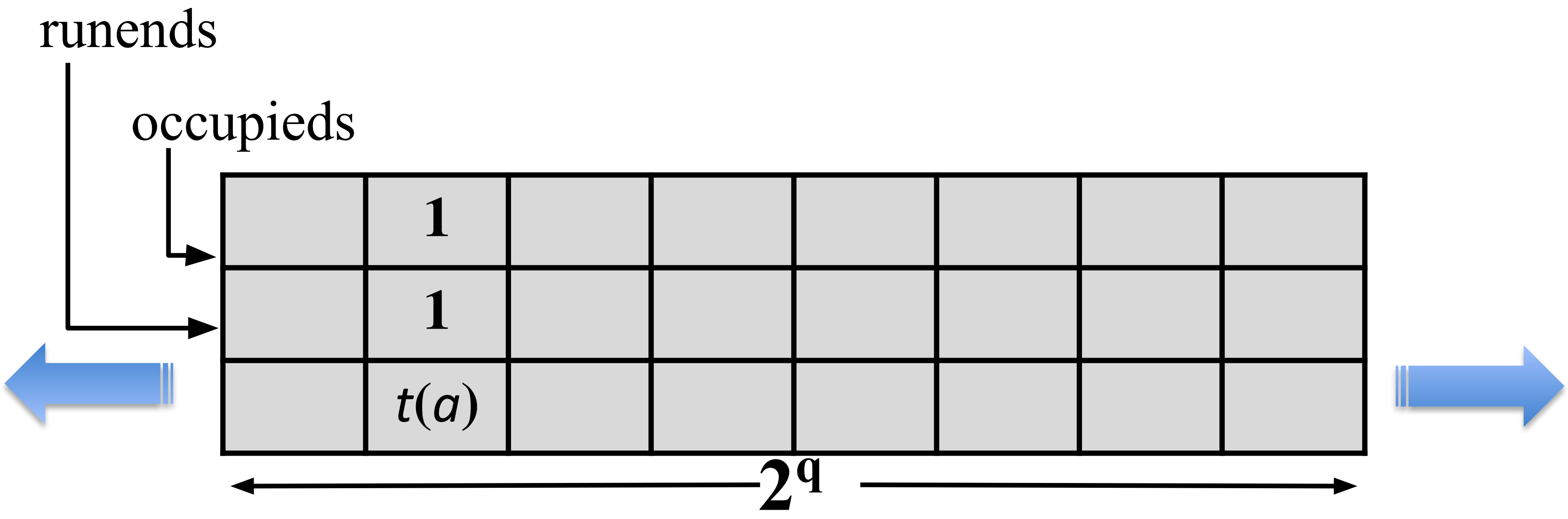
Implementation:

2 Meta-bits per slot.

$h(x) \dashrightarrow h_0(x) \,||\, h_1(x)$

Abstract Representation





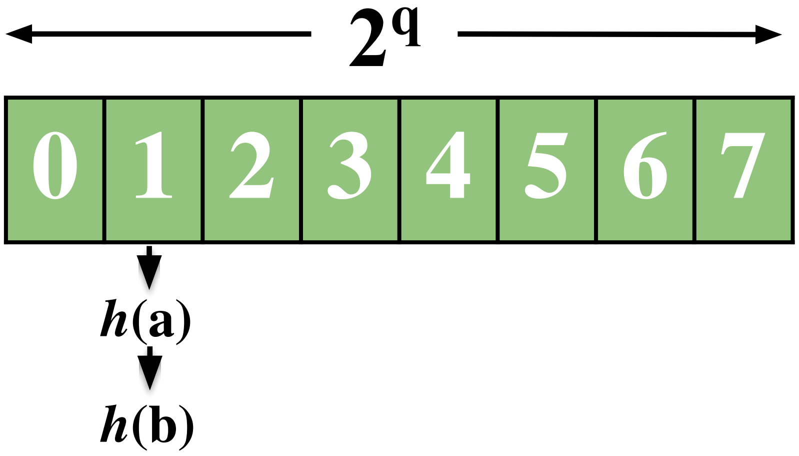
Quotient filter design

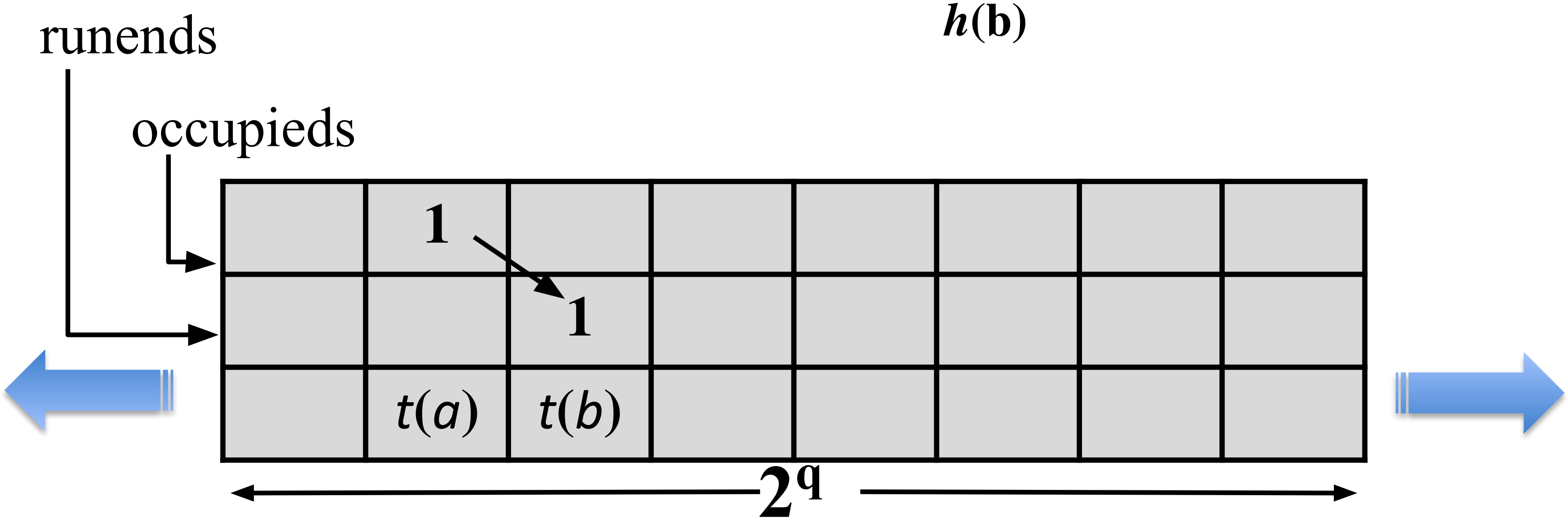
Implementation:

2 Meta-bits per slot.

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

Abstract Representation





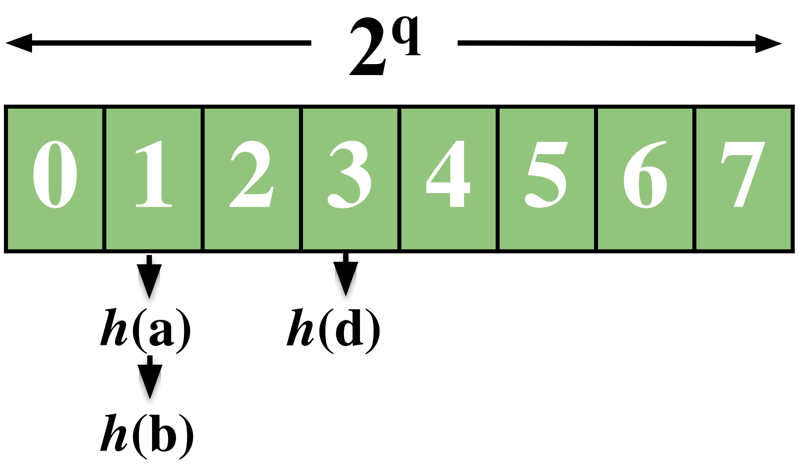
Quotient filter design

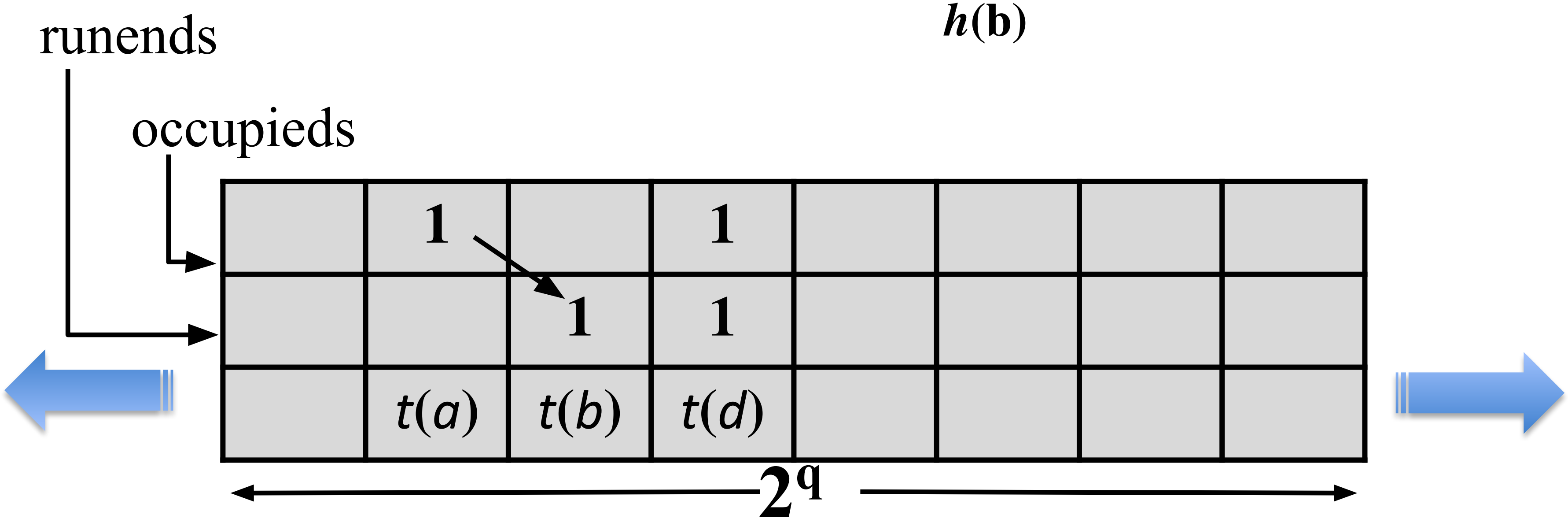
Implementation:

2 Meta-bits per slot.

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

Abstract Representation





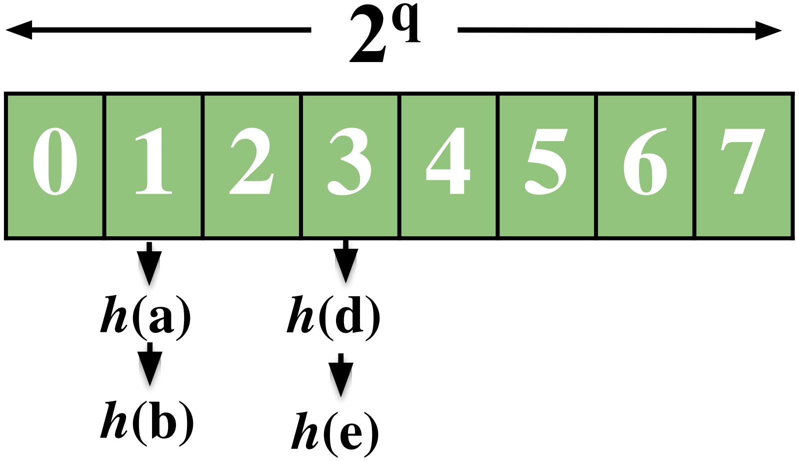
Quotient filter design

Implementation:

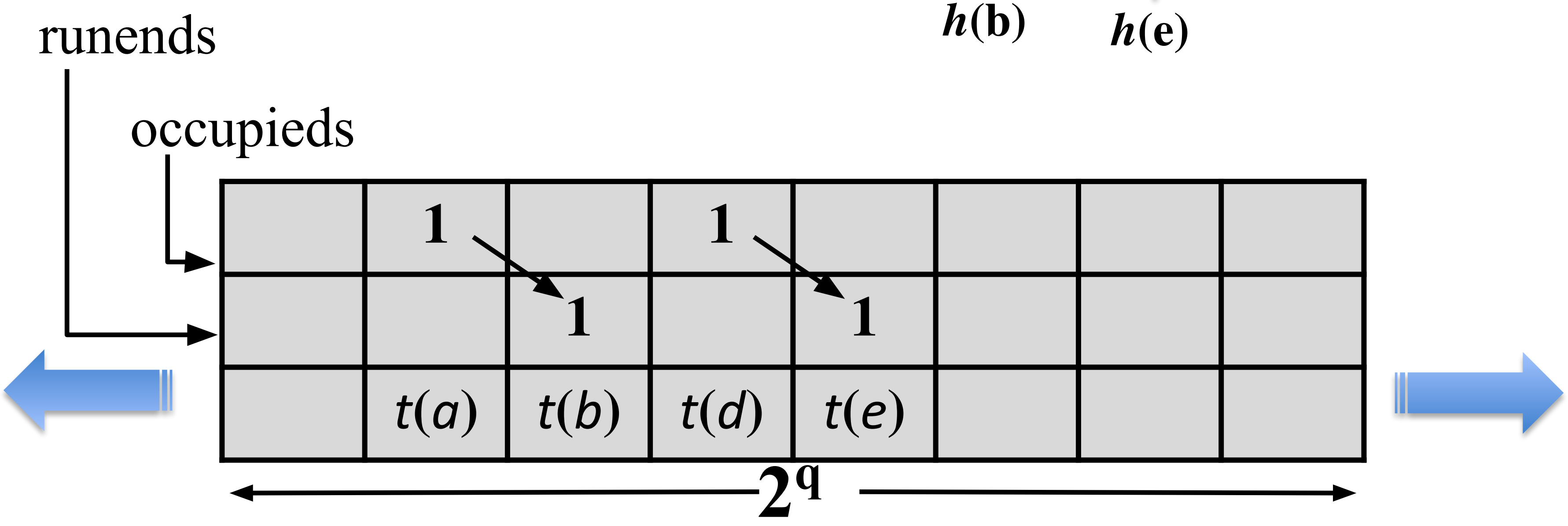
2 Meta-bits per slot.

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

Abstract Representation



0	1	2	3	4	5	6	7



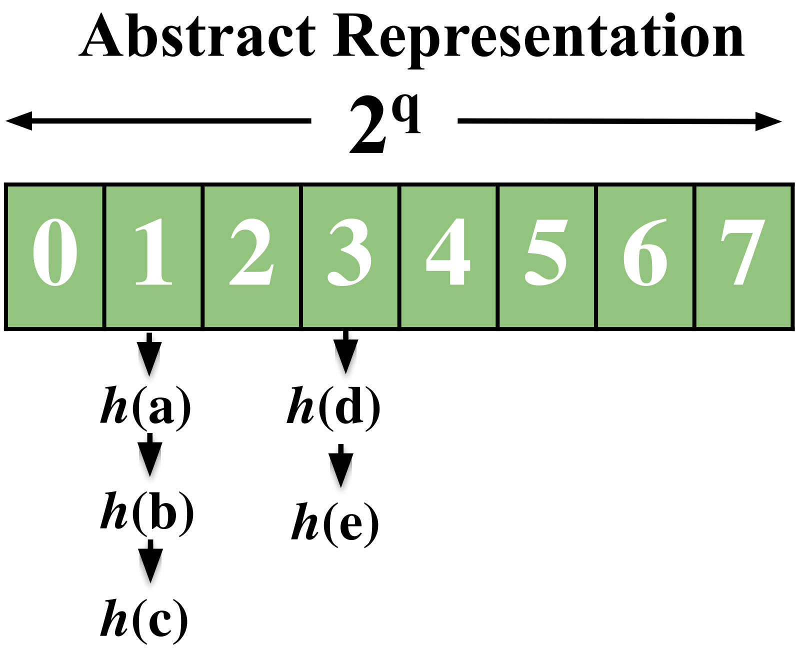
occupieds		1		1				
runends			1		1			
		t(a)	t(b)	t(d)	t(e)			

Quotient filter design

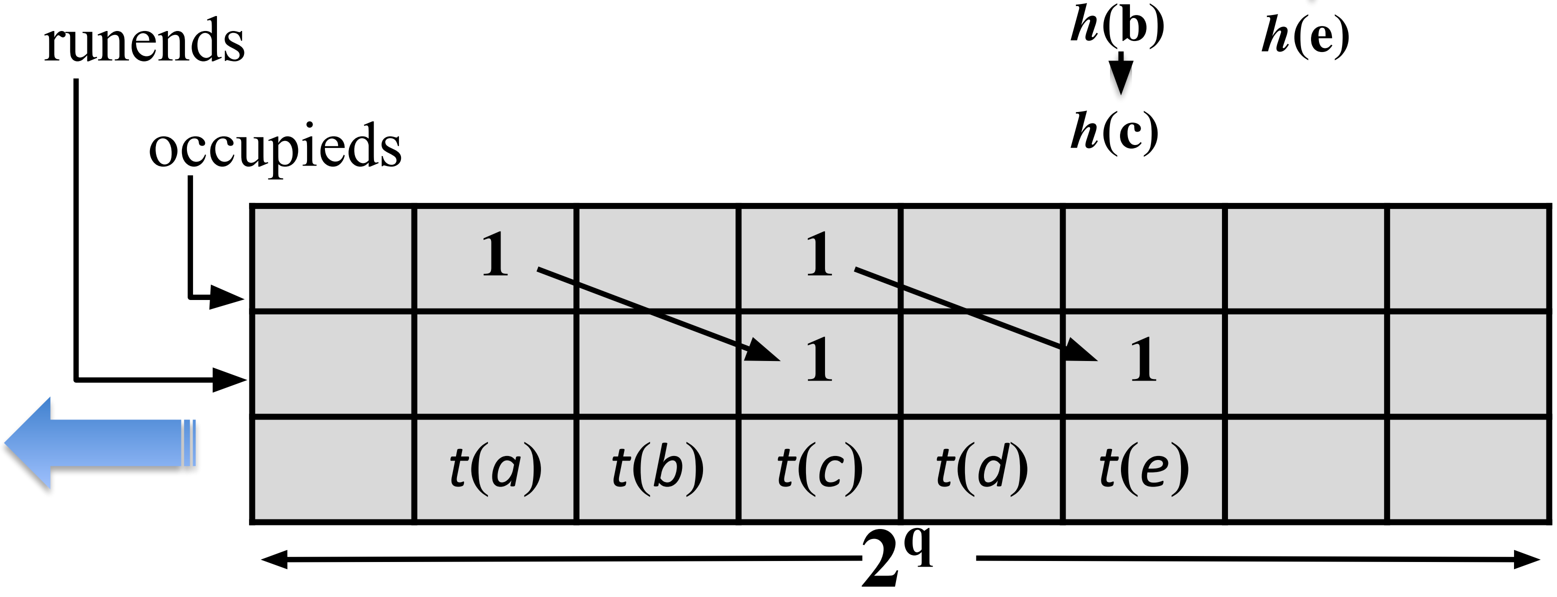
Implementation:

2 Meta-bits per slot.

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



Abstract Representation

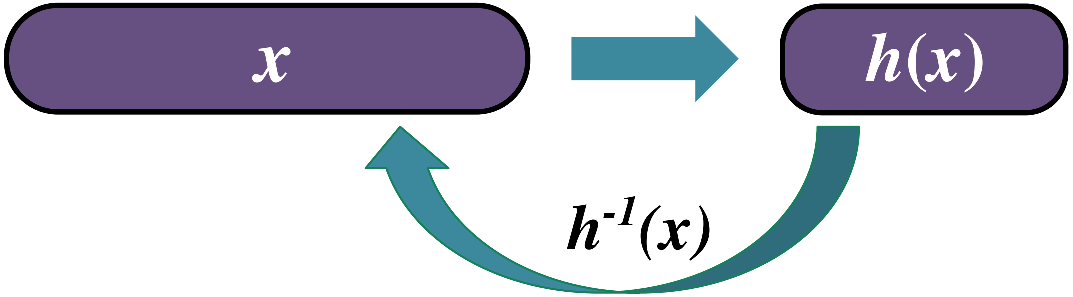


	1		1				
			1		1		
	t(a)	t(b)	t(c)	t(d)	t(e)		

Back

Quotient filters can also be exact

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

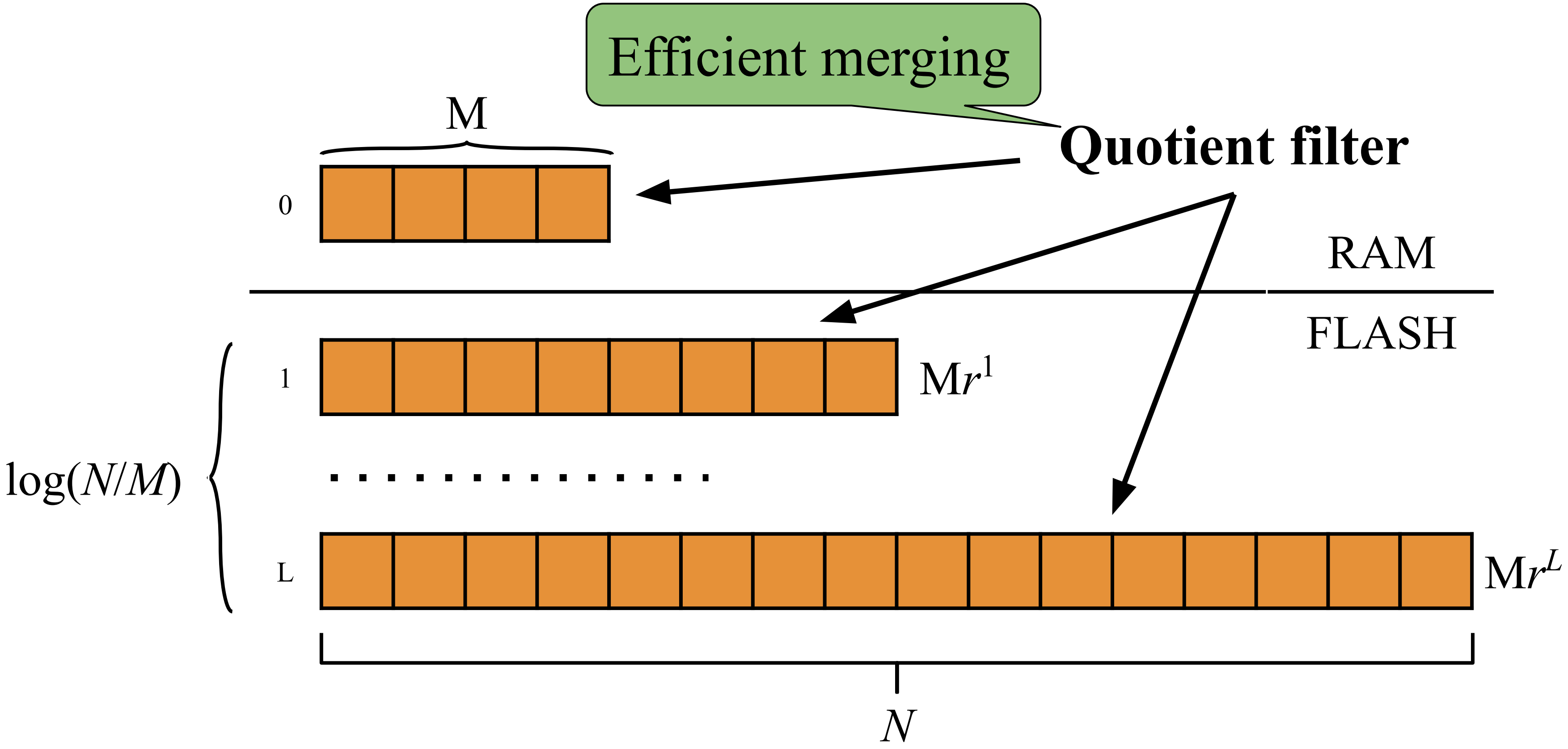


- **For *n* elements and *p*-bit hash function:**

Space usage: $\sim p - \log_2 n$ bits/element

Cascade filter: write-optimized quotient filter

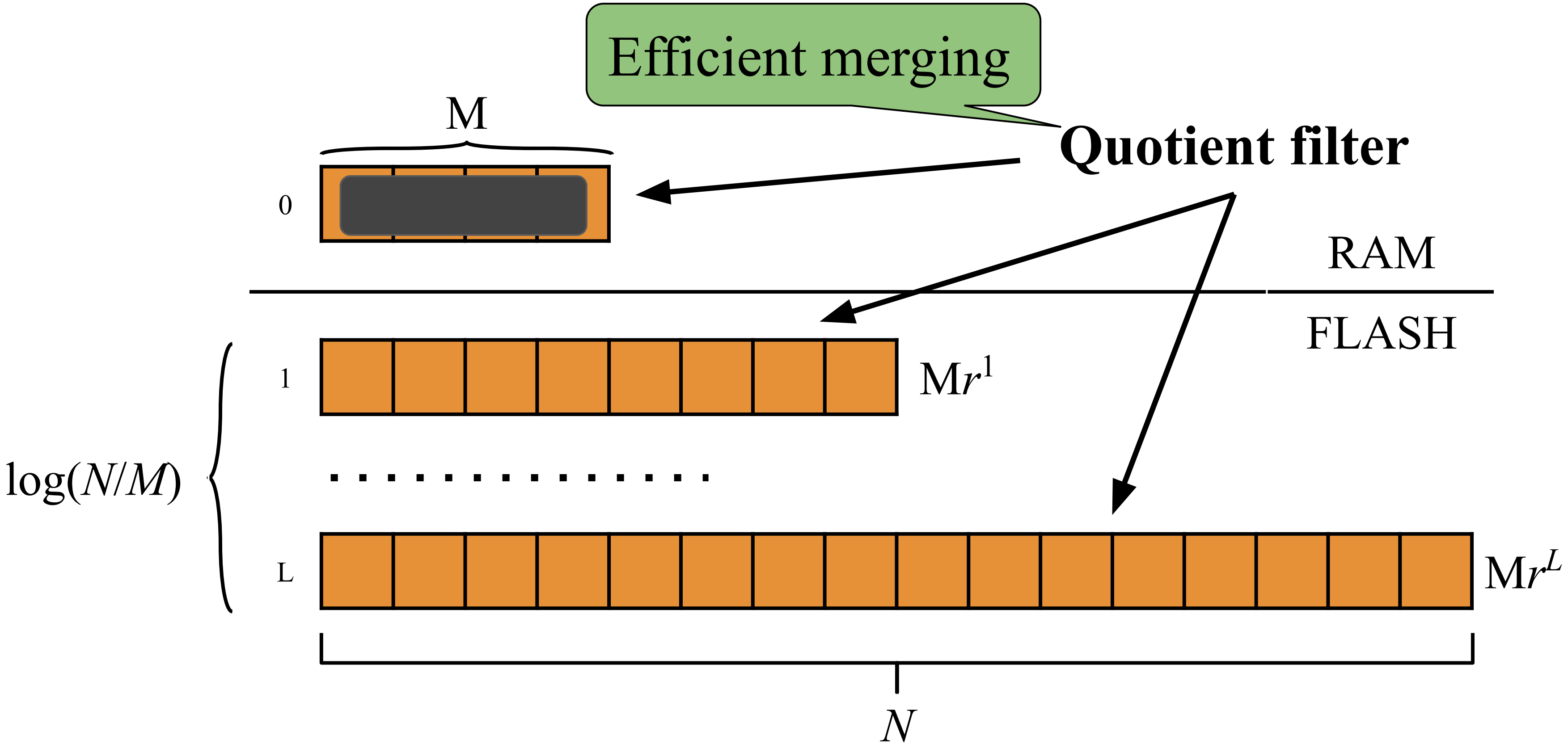
[Bender et al. ‘12, Pandey et al. ‘17]



- The Cascade filter efficiently scales out-of-RAM
- It accelerates insertions at some cost to queries

Cascade filter: flushing

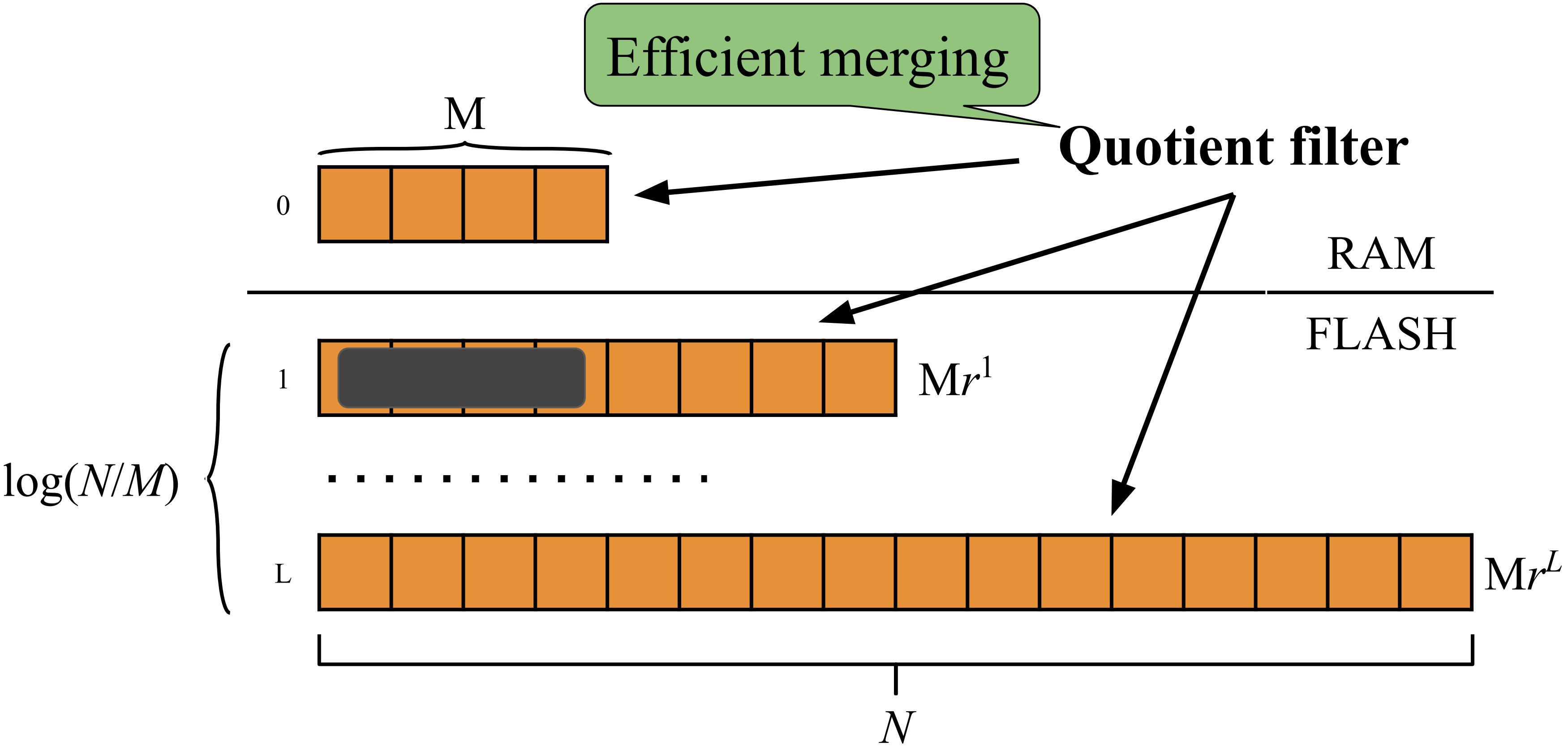
[Bender et al. ‘12, Pandey et al. ‘17]



Items are initially inserted in the RAM level

Cascade filter: flushing

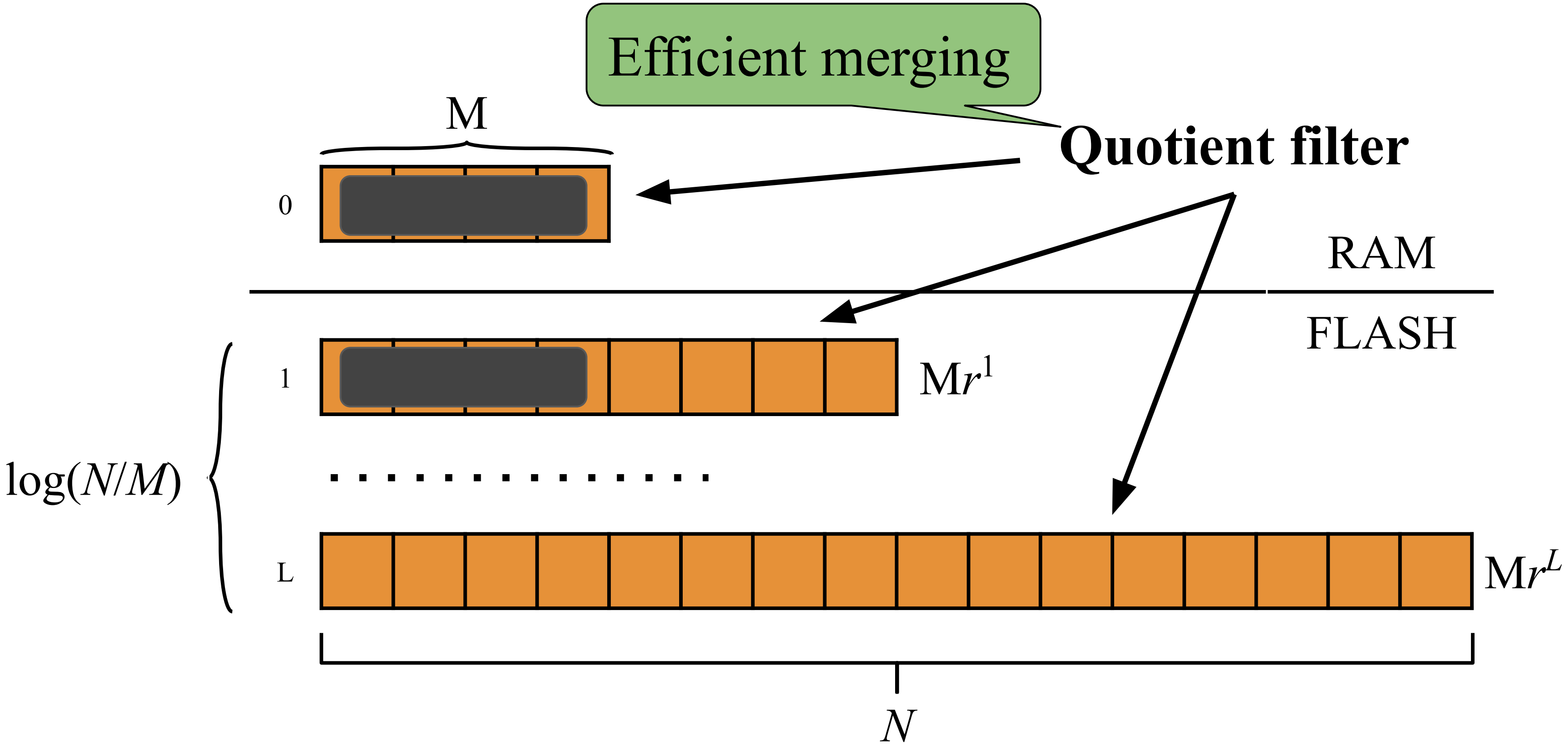
[Bender et al. ‘12, Pandey et al. ‘17]



When RAM is full, items are flushed to the smallest level on disk ***i*** with space to insert items in level **0** to ***i*-1**

Cascade filter: flushing

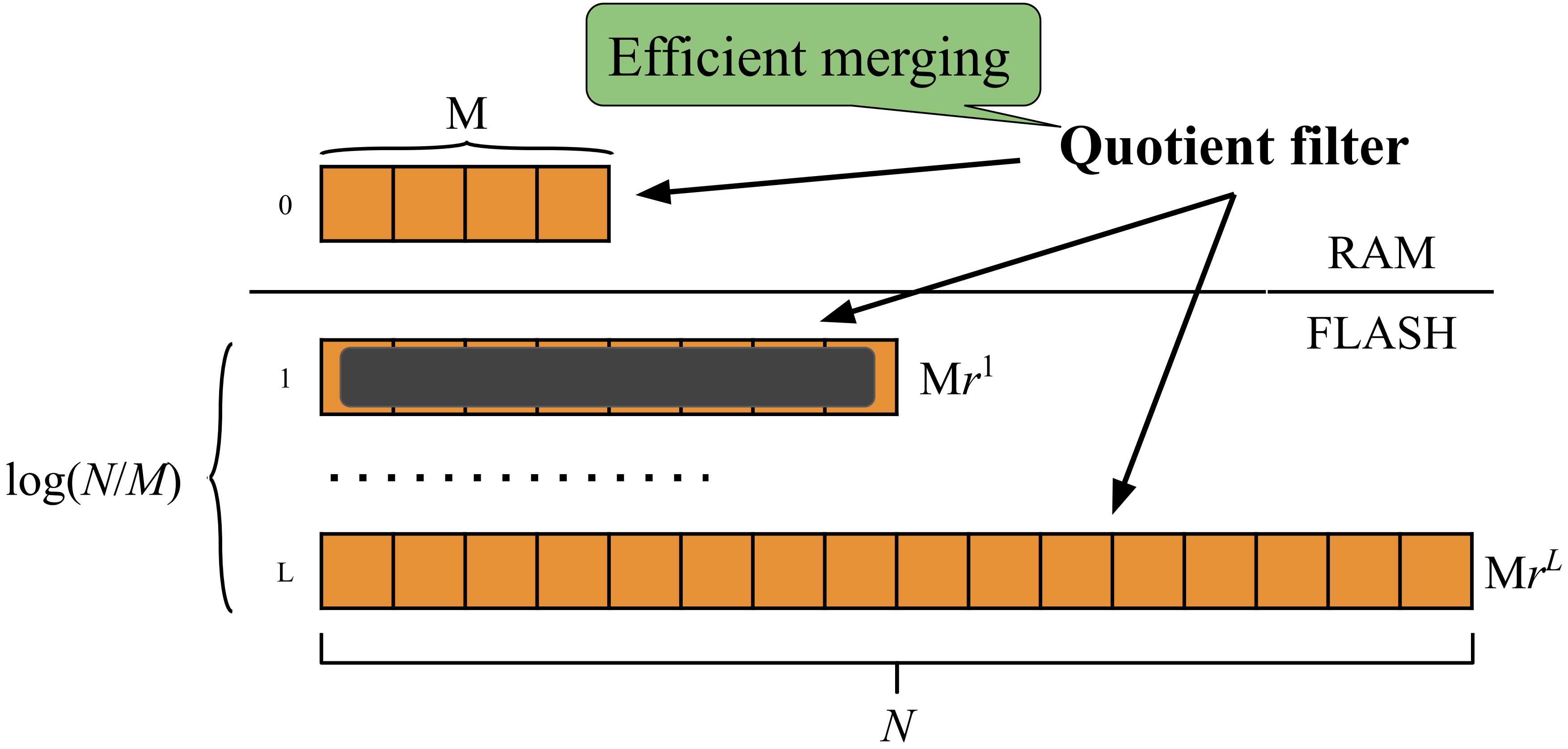
[Bender et al. '12, Pandey et al. '17]



When RAM is full, items are flushed to the smallest level on disk ***i*** with space to insert items in level **0** to ***i*-1**

Cascade filter: flushing

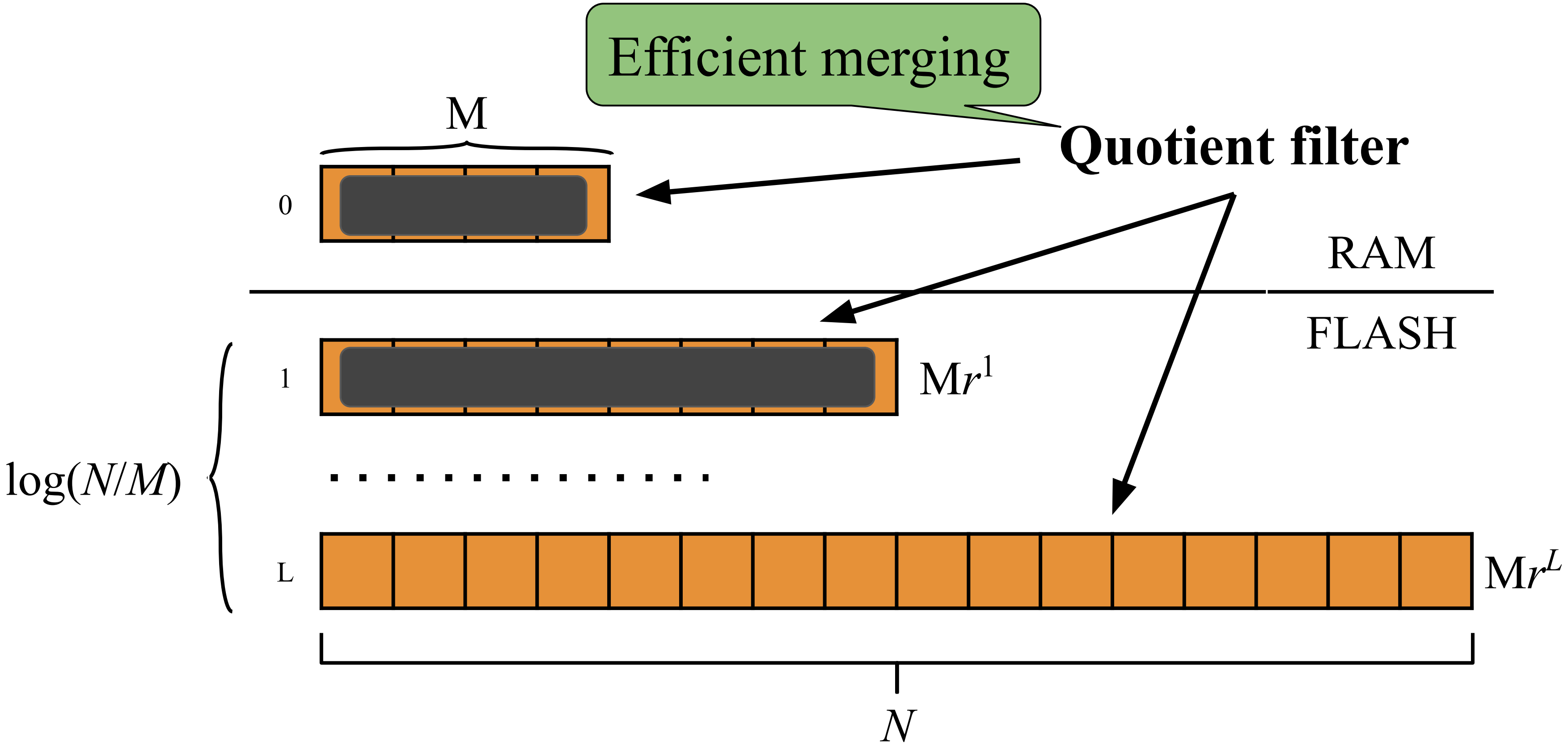
[Bender et al. ‘12, Pandey et al. ‘17]



When RAM is full, items are flushed to the smallest level on disk ***i*** with space to insert items in level **0** to ***i*-1**

Cascade filter: flushing

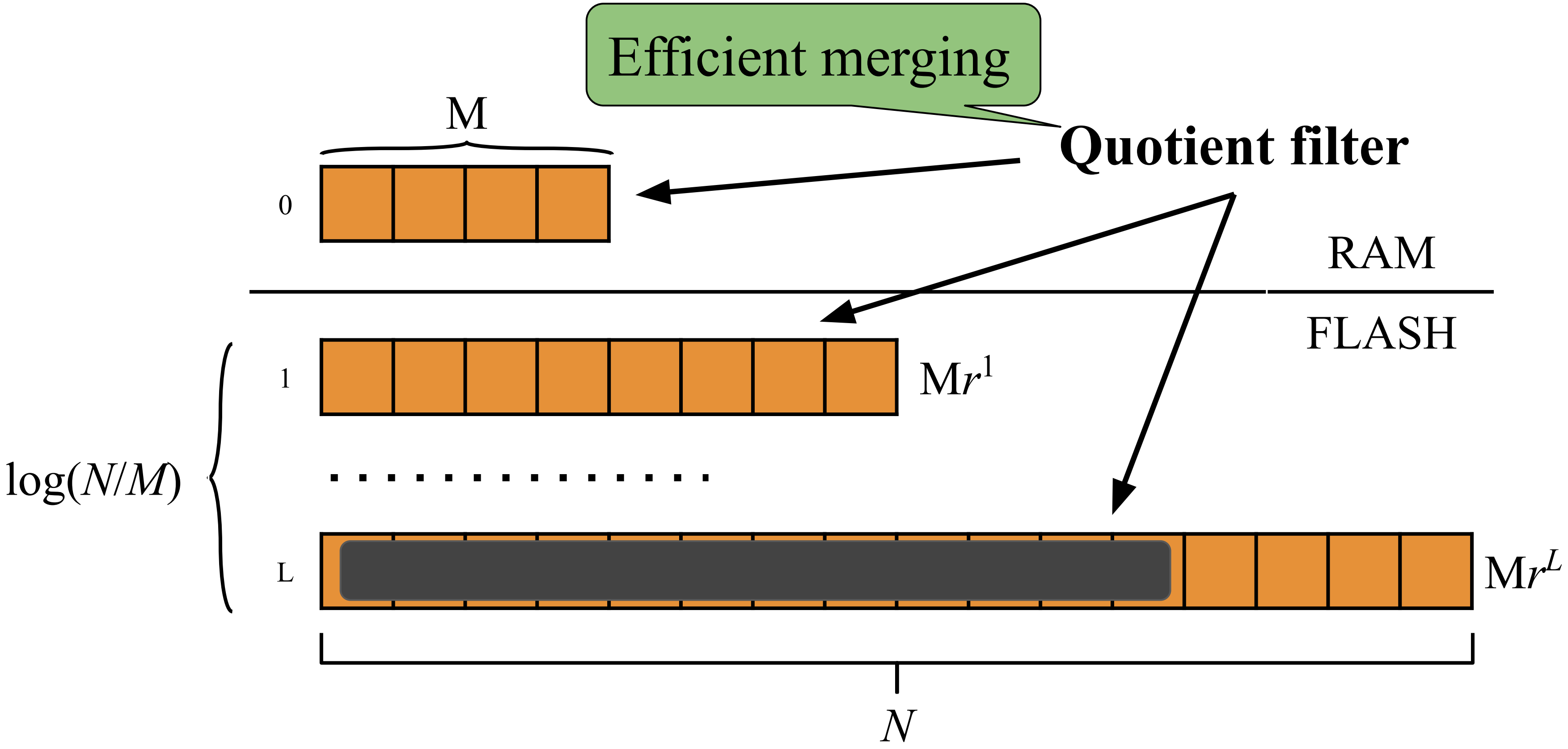
[Bender et al. ‘12, Pandey et al. ‘17]



When RAM is full, items are flushed to the smallest level on disk ***i*** with space to insert items in level **0** to ***i*-1**

Cascade filter: flushing

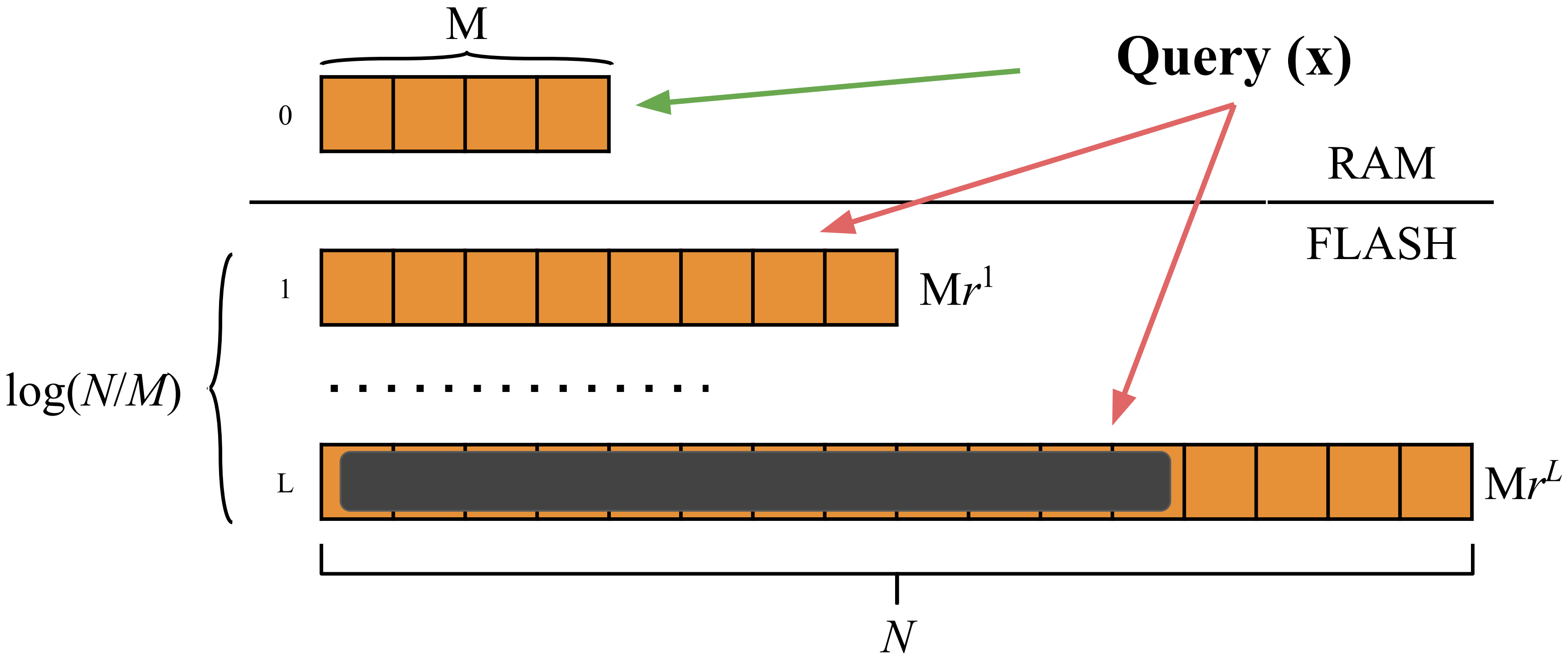
[Bender et al. ‘12, Pandey et al. ‘17]



When RAM is full, items are flushed to the smallest level on disk ***i*** with space to insert items in level **0** to ***i*-1**

Cascade filter: query

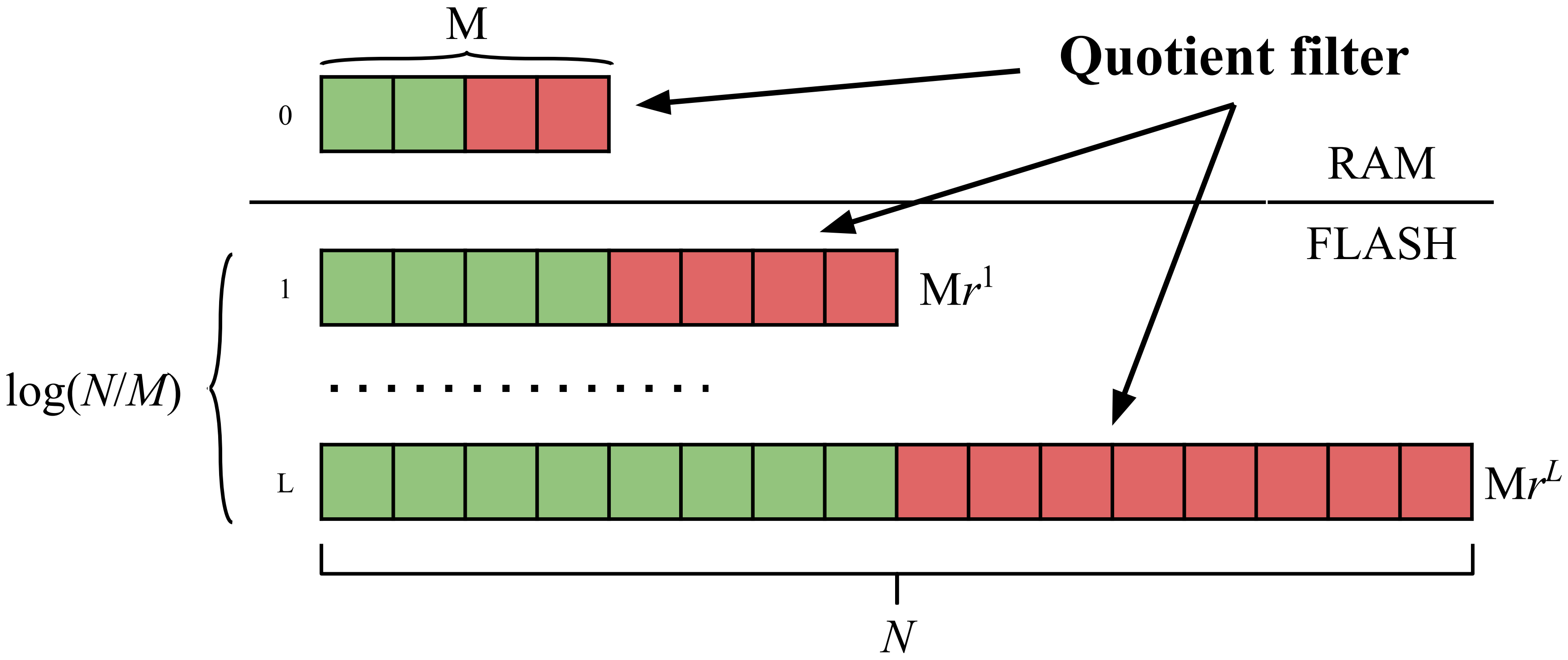
[Bender et al. ‘12, Pandey et al. ‘17]



A query operation requires a lookup in each non-empty level

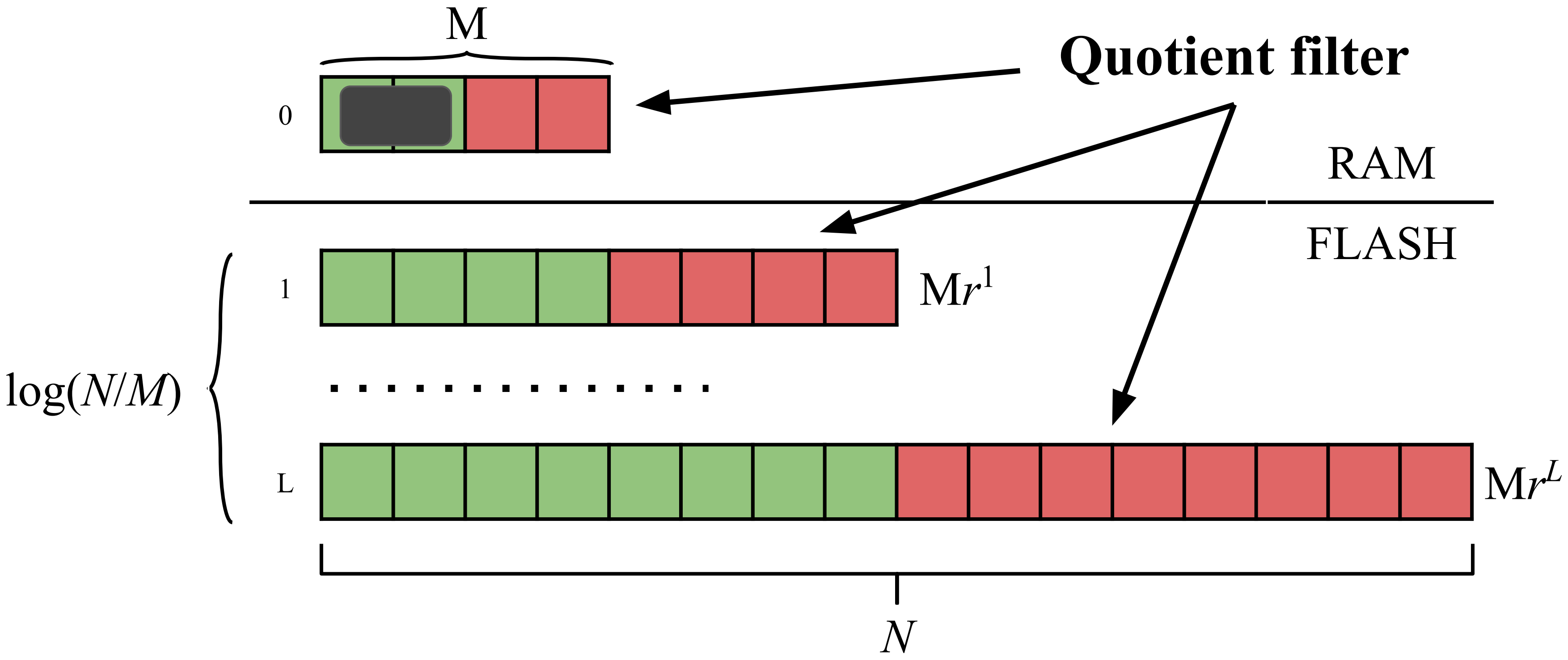
Back

Time-stretch LERT



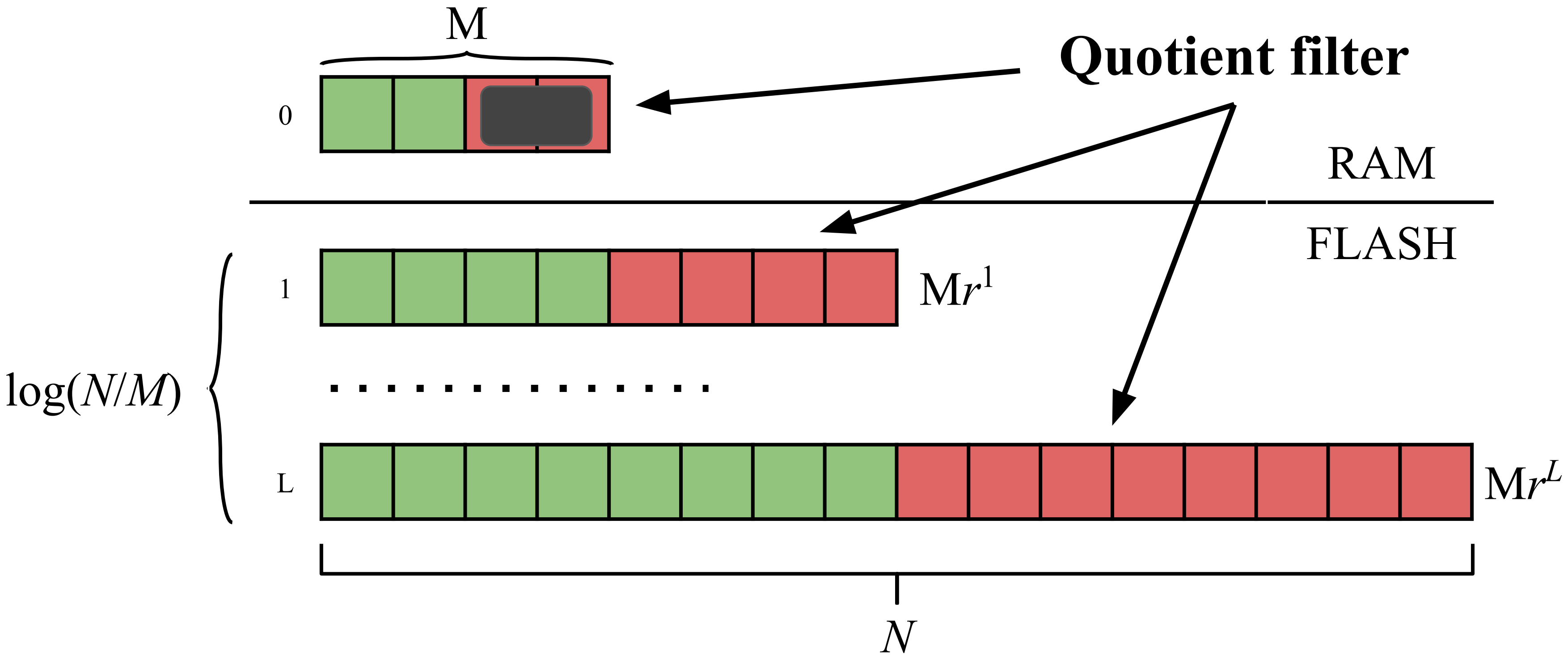
Divide each level into $1 + 1/\alpha$, equal-sized bins.

Time-stretch LERT



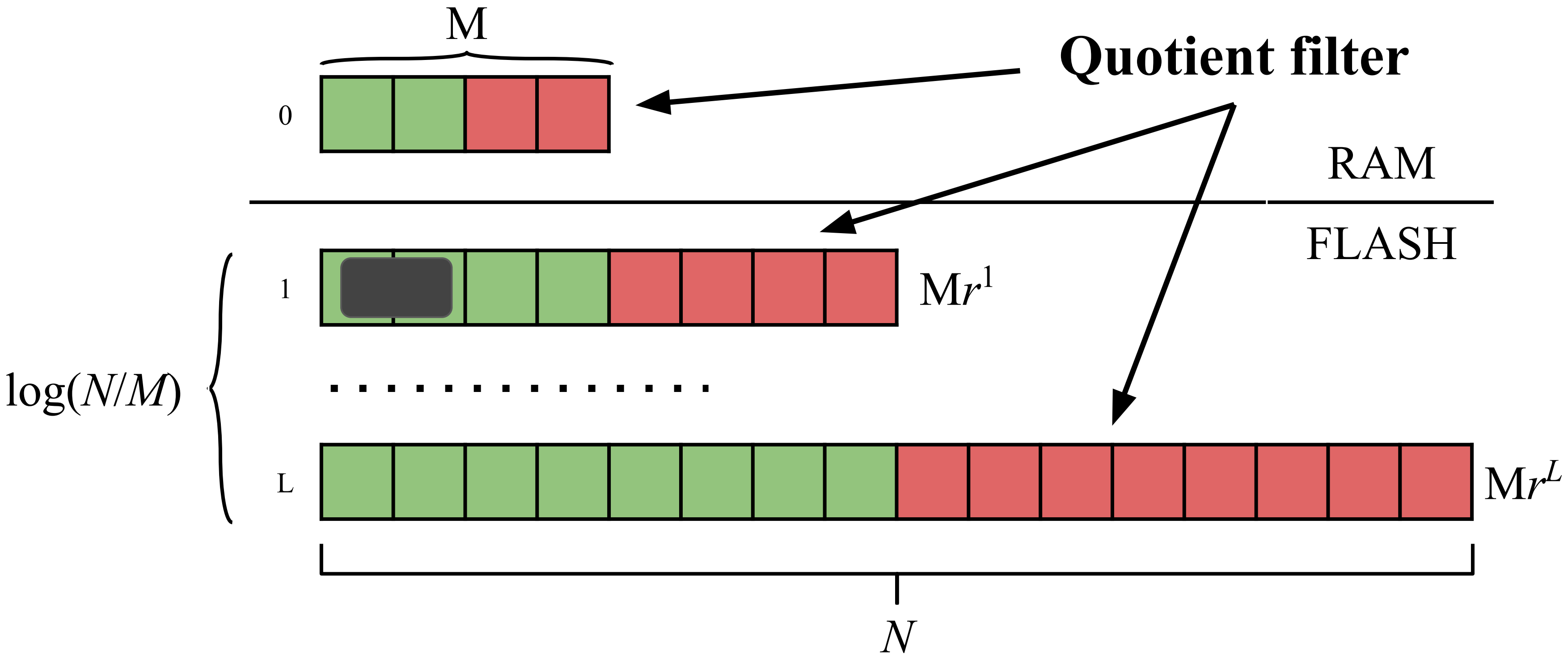
When a bin is full, items move to the adjacent bin

Time-stretch LERT



When a bin is full, items move to the adjacent bin

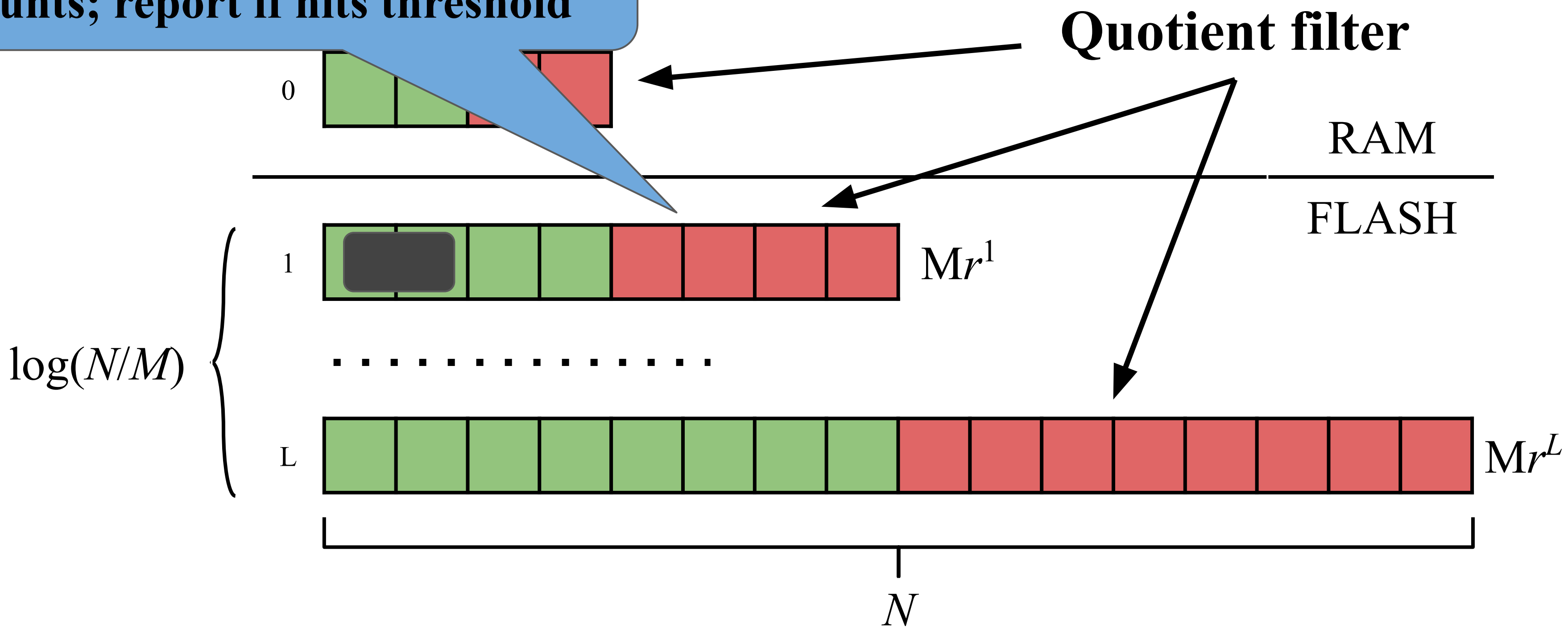
Time-stretch LERT



Last bin **flushed** to first bin of the next level

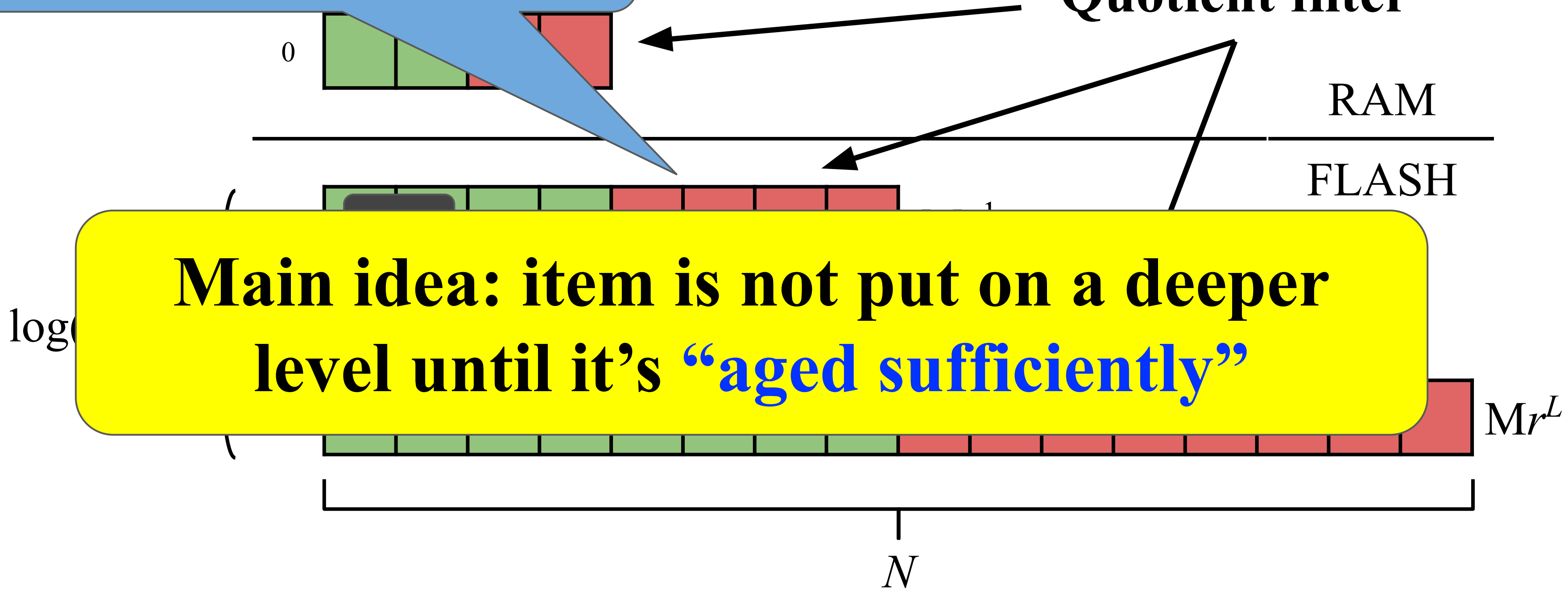
Time-stretch LERT

While flushing consolidate counts; report if hits threshold



Last bin **flushed** to first bin of the next level

Time-stretch LERT



While flushing consolidate counts; report if hits threshold

Quotient filter

Main idea: item is not put on a deeper level until it's "aged sufficiently"

Last bin **flushed** to first bin of the next level

Time-stretch LERT I/O complexity

$$O\left(\left(\frac{\alpha+1}{\alpha}\right)\frac{1}{B}\log\frac{N}{M}\right)$$

Optimal insert cost for Write-optimized data structure

Time-stretch LERT I/O complexity

$$O\left(\left(\frac{\alpha+1}{\alpha}\right)\frac{1}{B}\log\frac{N}{M}\right)$$

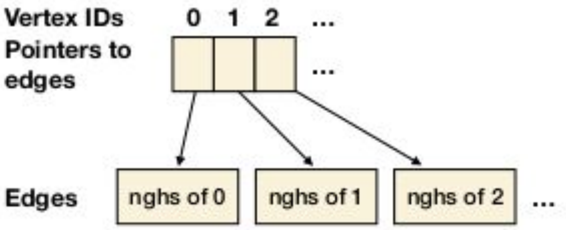
Extra cost because we only move one bin during a flush. Constant loss for constant α

Optimal insert cost for Write-optimized data structure

Back

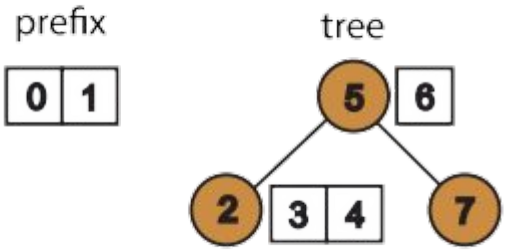
Trade-off 2: "One-size-fits-all" approach leaves performance on table

Static



LIGRA [Shun & Blelloch '13]

Dynamic



ASPEN [Dhulipala et al. '19]

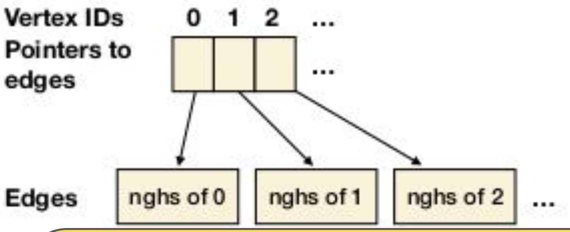
	LIGRA	ASPEN
add_edge	$O((\lvert E\rvert + \lvert V\rvert))/B)$	$O(\log \lvert V\rvert + c^2 \log(deg(u))/B)$
get_neighbors	$O(deg(u)/B)$	$O(\log \lvert V\rvert + deg(u)/B + deg(u)/c)$

Neighbor access requires at least ***two cache misses***

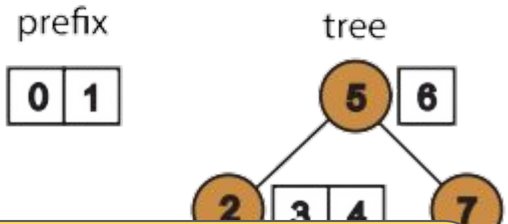
For dynamic, all operations have a ***log factor***

Trade-off 2: “One-size-fits-all” approach leaves performance on table

Static



Dynamic



Static → Fast computations; no updates

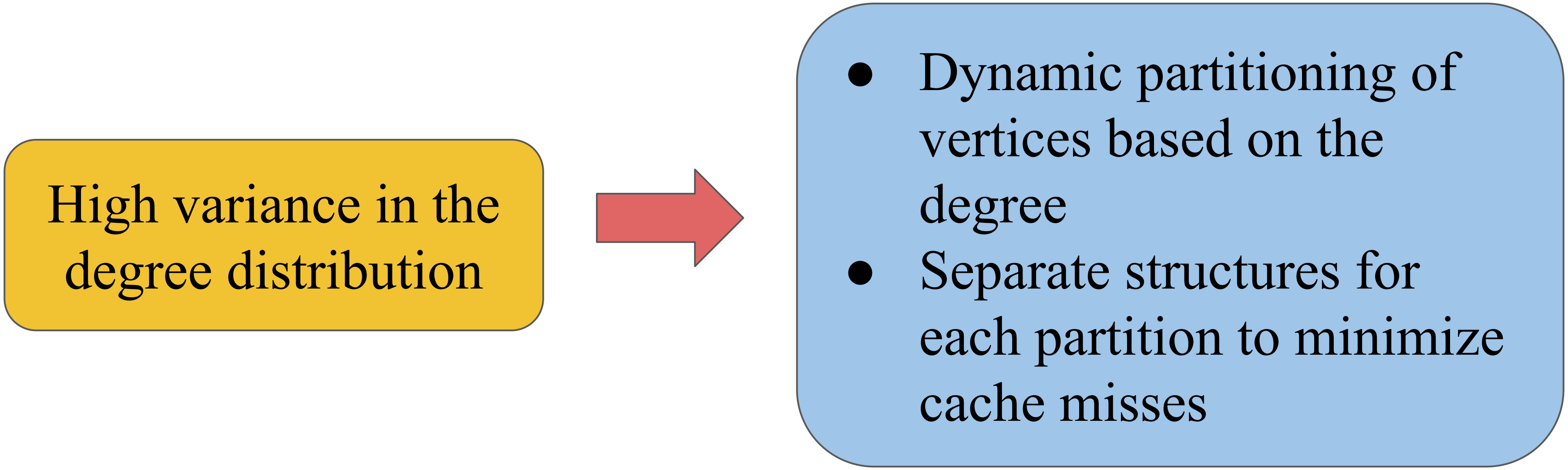
Dynamic → Slower computations; updates

	LIGRA	ASPEN
add_edge	$O((\|E\| + \|V\|))/B)$	$O(\log \|V\| + c^2 \log(deg(u))/B)$
get_neighbors	$O(deg(u)/B)$	$O(\log \|V\| + deg(u)/B + deg(u)/c)$

Neighbor access requires at least ***two cache misses***

For dynamic, all operations have a ***log factor***

Real world graphs are often skewed



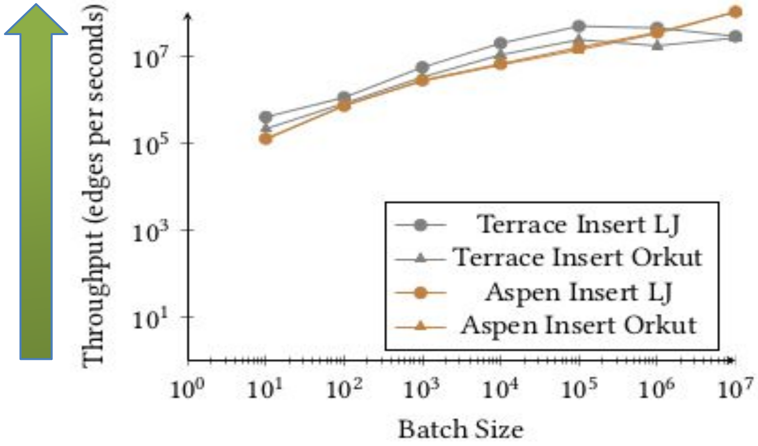
High variance in the degree distribution

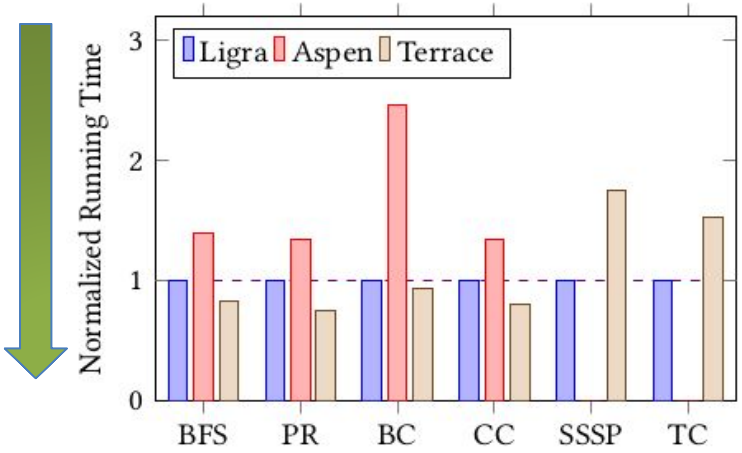
- Dynamic partitioning of vertices based on the degree
- Separate structures for each partition to minimize cache misses

Dynamic partitioning + hierarchical structure

High variance in the degree distribution

- Dynamic partitioning of vertices based on the degree
- Separate structures for each partition to minimize cache misses



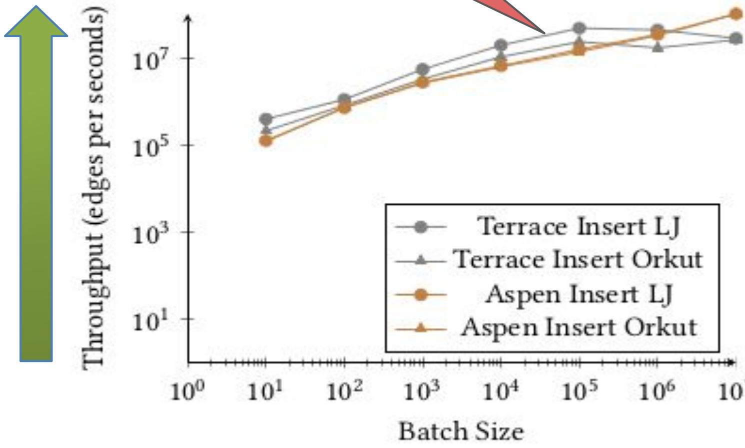


Dynamic partitioning + hierarchical structure

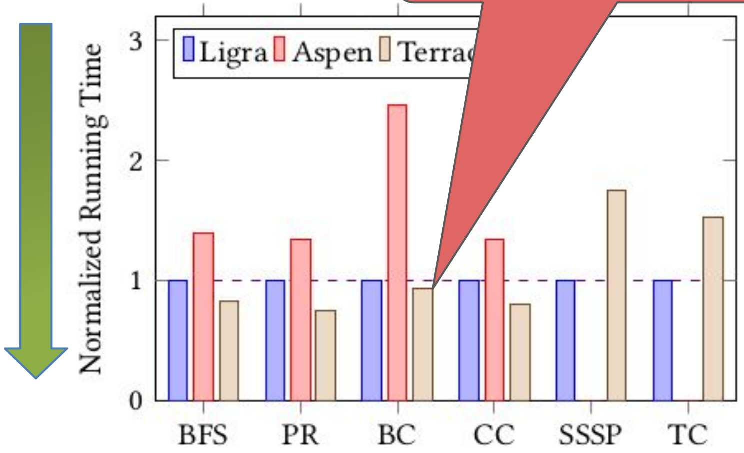
High variance in the degree distribution

- Dynamic partitioning of vertices based on the degree
- Separate structures for each partition to minimize cache misses

Terrace: Fast updates

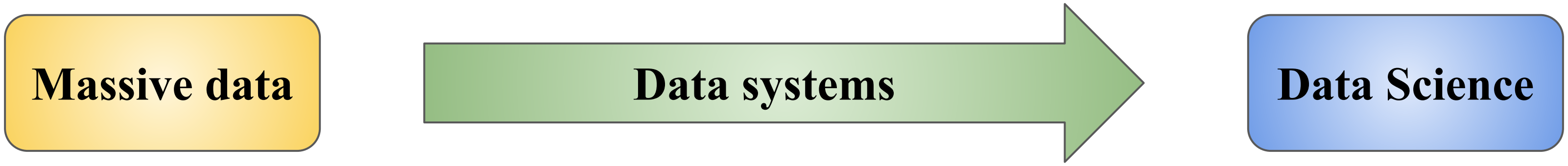


Terrace: Faster computations



Back

Scalable data systems → Scalable data science



Massive data

Data systems

Data Science

Biology
AstroPhysics
Chemistry
Cyber monitoring
Internet of Things
Environmental science
⋮

Machine Learning
Raw sequence search
Graph analytics
Cyber monitoring
Weather predictions
Personalized medicine
⋮

My goal as a researcher is to build ***scalable data systems*** to ***accelerate*** and ***scale data science*** applications

Our contribution

Streaming model

External memory algorithms

Combine streaming and EM algorithms to solve real-time event detection problem