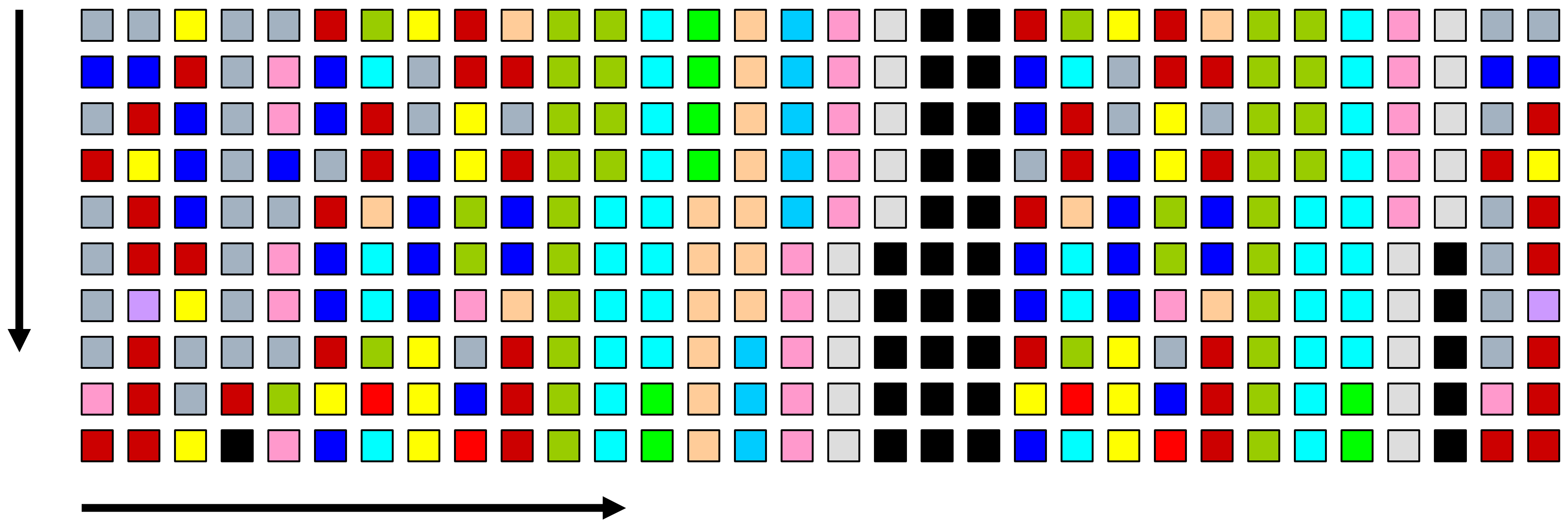
Frequency Counts over Data Streams

Gurmeet Singh Manku

Stanford University, USA

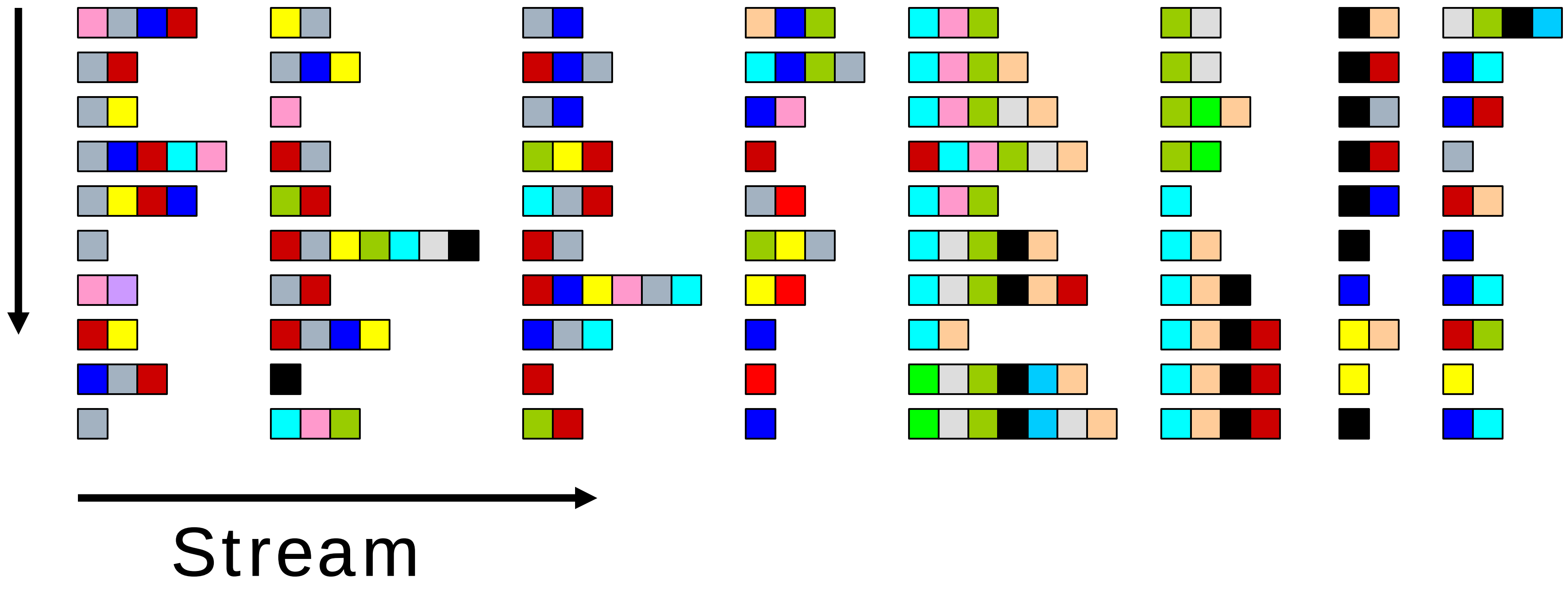
The Problem ...



Stream

- Identify all elements whose current frequency exceeds support threshold s = 0.1%.

A Related Problem ...



- ➢ Identify all subsets of items whose current frequency exceeds s = 0.1%.

Frequent Itemsets / Association Rules

Applications

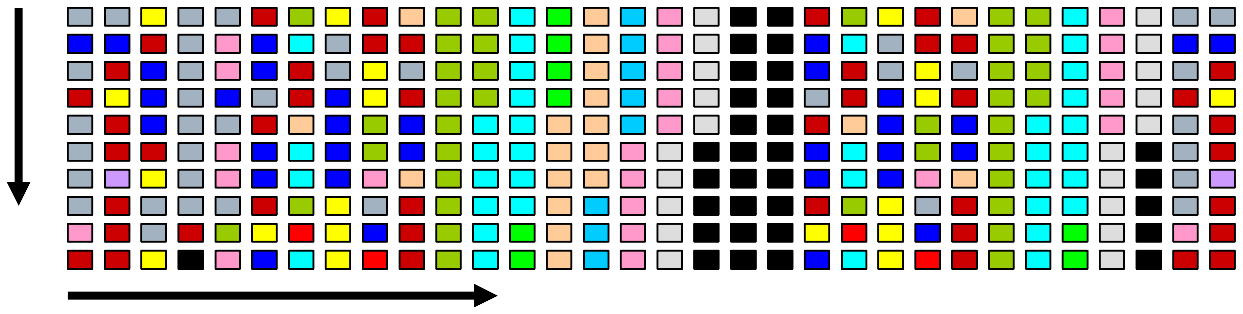
Flow Identification at IP Router [EV01]

Iceberg Queries [FSGM+98]

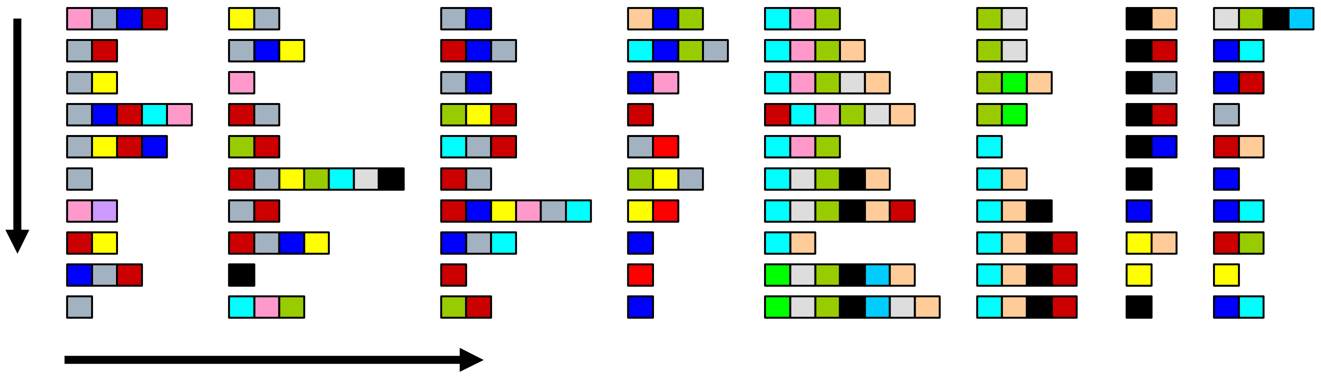
Iceberg Datacubes [BR99 HPDW01]

Association Rules & Frequent Itemsets
[AS94 SON95 Toi96
Hid99 HPY00 ...]

Presentation Outline ...



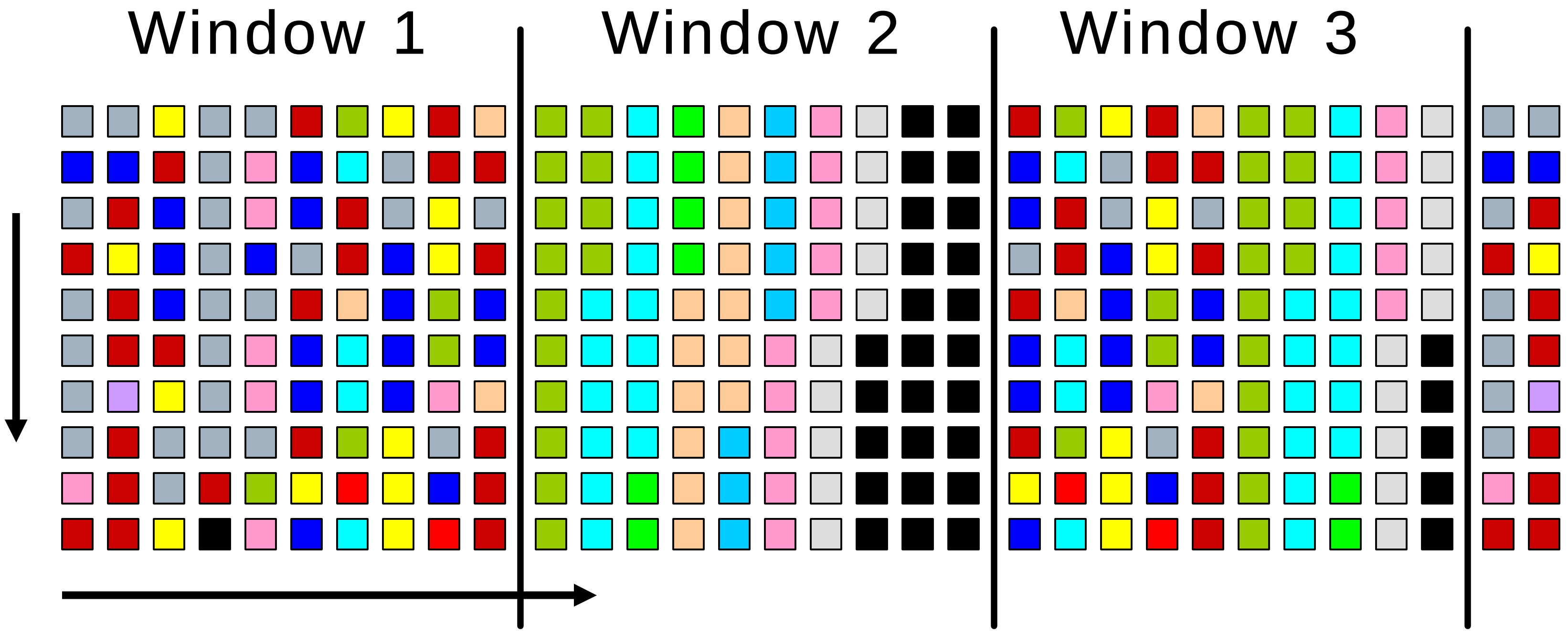
1. Lossy Counting
2. Sticky Sampling



3. Algorithm for Frequent Itemsets

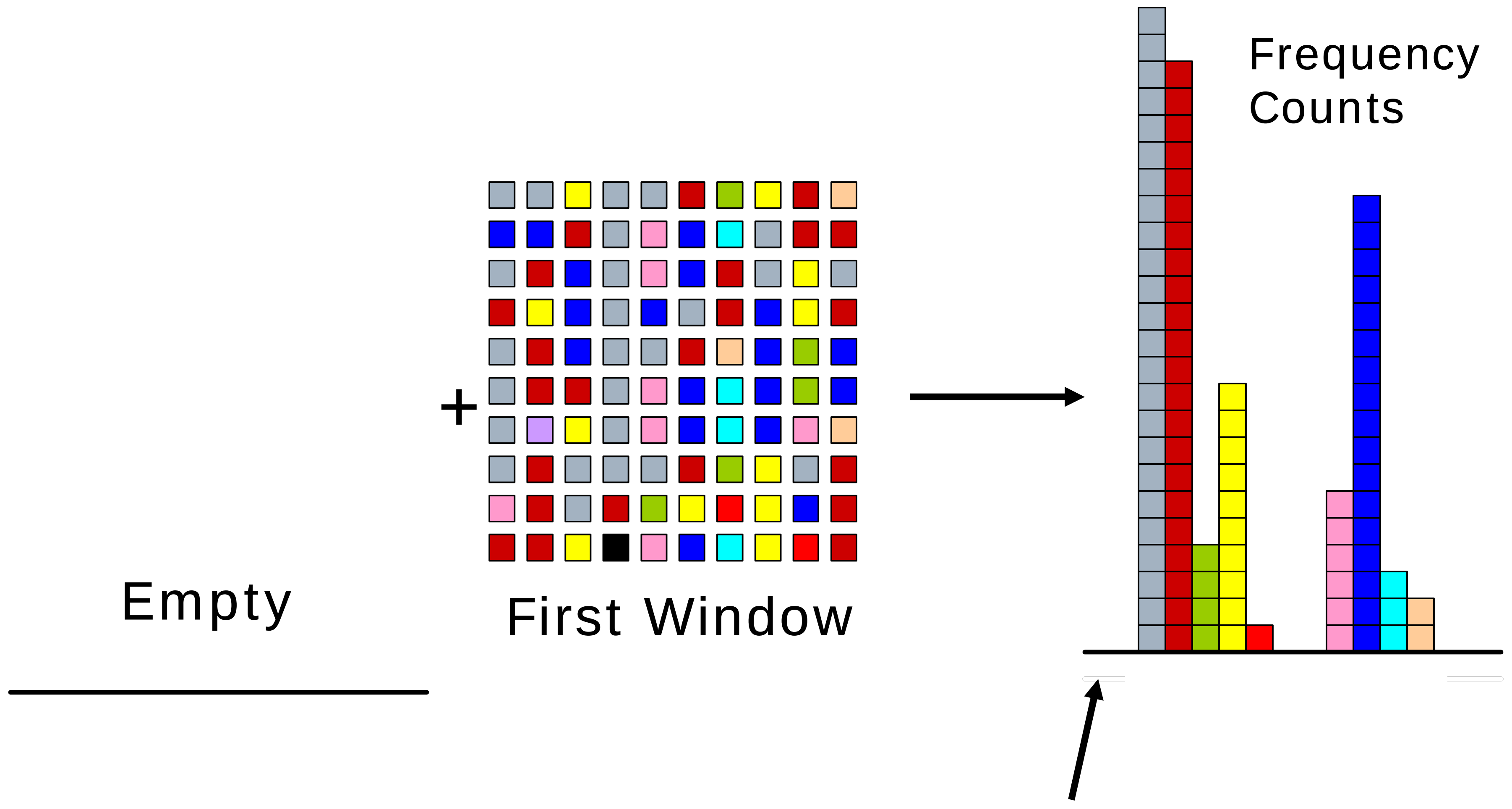
Algorithm 1: Lossy Counting

Step 1: Divide the stream into 'windows'



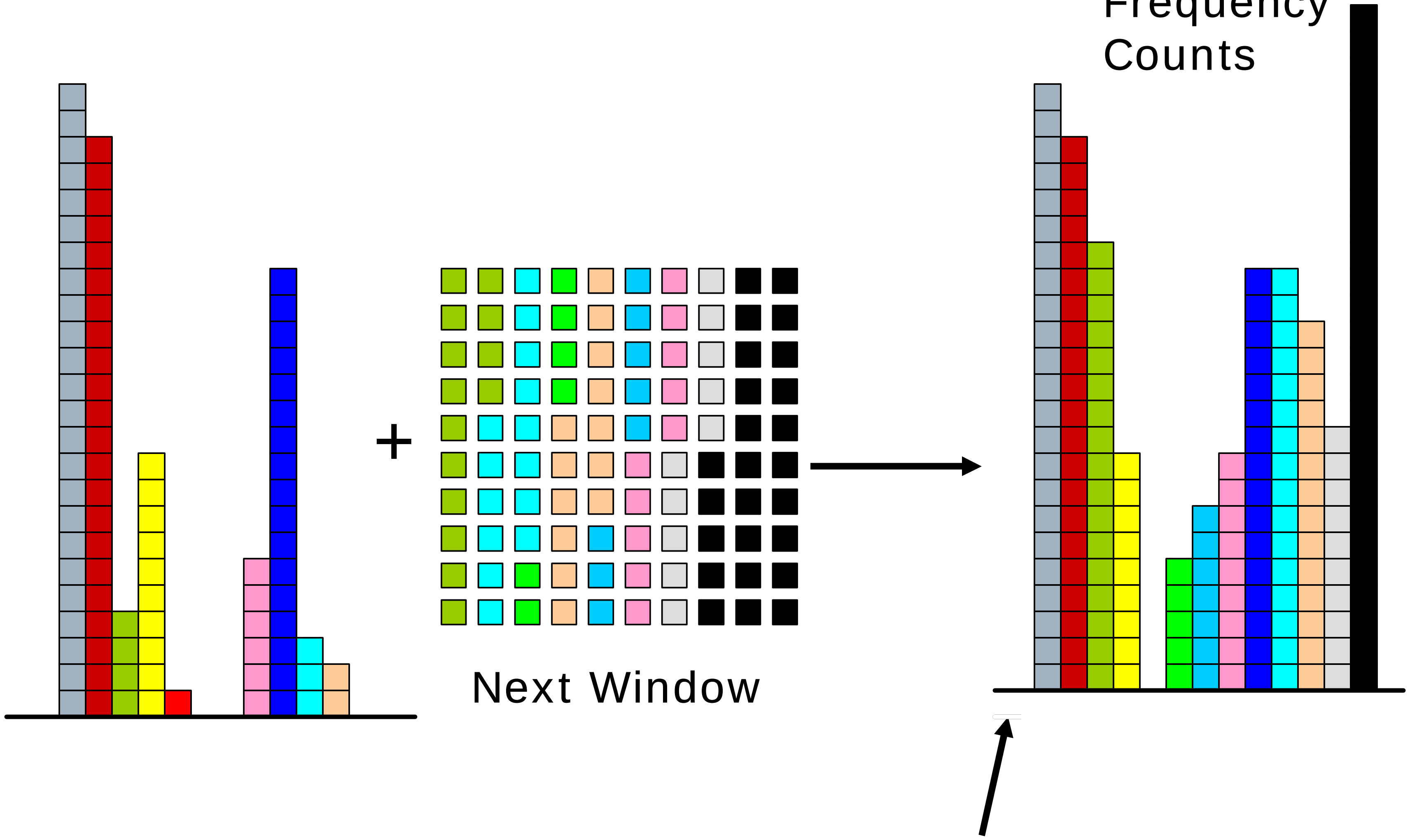
Is window size a function of support s? Will fix later…

Lossy Counting in Action ...



At window boundary, decrement all counters by 1

Lossy Counting continued ...



At window boundary, decrement all counters by 1

Error Analysis

How much do we undercount?

If current size of stream = N

and window-size = 1/e

then frequency error £ #windows = eN

Rule of thumb:

Set e = 10% of support s

Example:

Given support frequency s = 1%,

set error frequency e = 0.1%

Output:

Elements with counter values exceeding sN – eN

Approximation guarantees

- Frequencies underestimated by at most eN
- No false negatives
- False positives have true frequency at least sN – eN

How many counters do we need?

Worst case: 1/e log (e N) counters [See paper for proof]

Enhancements ...

Frequency Errors

For counter (X, c), true frequency in [c, c+eN]

Trick: Remember window-id's

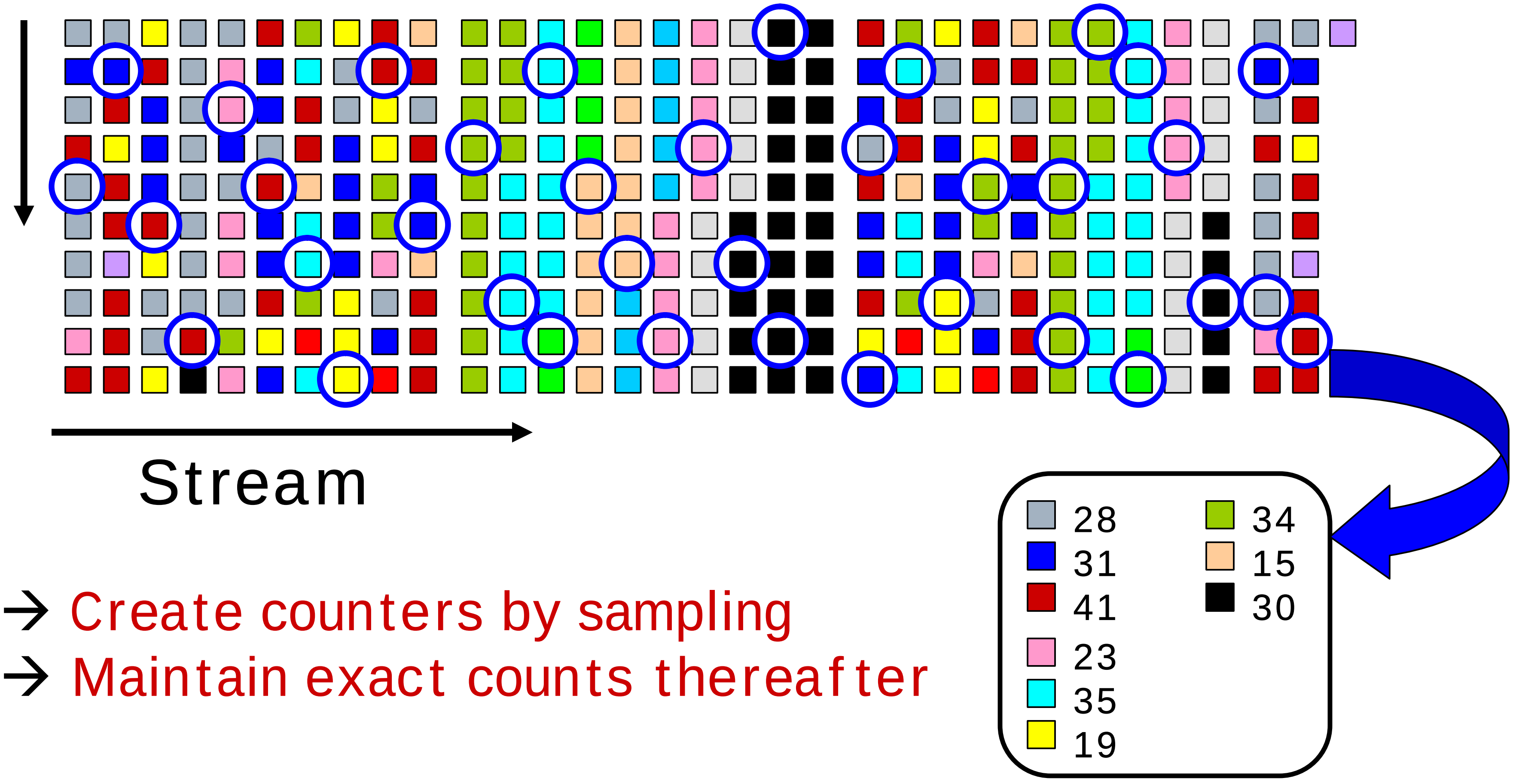
For counter (X, c, w), true frequency in [c, c+w-1]

If (w = 1), no error!

Batch Processing

Decrements after k windows

Algorithm 2: Sticky Sampling



→ Create counters by sampling
→ Maintain exact counts thereafter

What rate should we sample?

Sticky Sampling contd...

For finite stream of length N

Sampling rate = 2/Ne log 1/(sδ)

δ = probability of failure

Output:

Elements with counter values exceeding sN – eN

Approximation guarantees (probabilistic)

- Frequencies underestimated by at most eN
- No false negatives
- False positives have true frequency at least sN – eN

Same error guarantees as Lossy Counting but <u>probabilistic</u>

Same Rule of thumb:

Set e = 10% of support s

Example:

Given support threshold s = 1%,
set error threshold e = 0.1%
set failure probability δ = 0.01%

Sampling rate?

Finite stream of length N

Sampling rate: $2/Ne \ \log 1/(s\delta)$

Infinite stream with unknown N

Gradually adjust sampling rate (see paper for details)

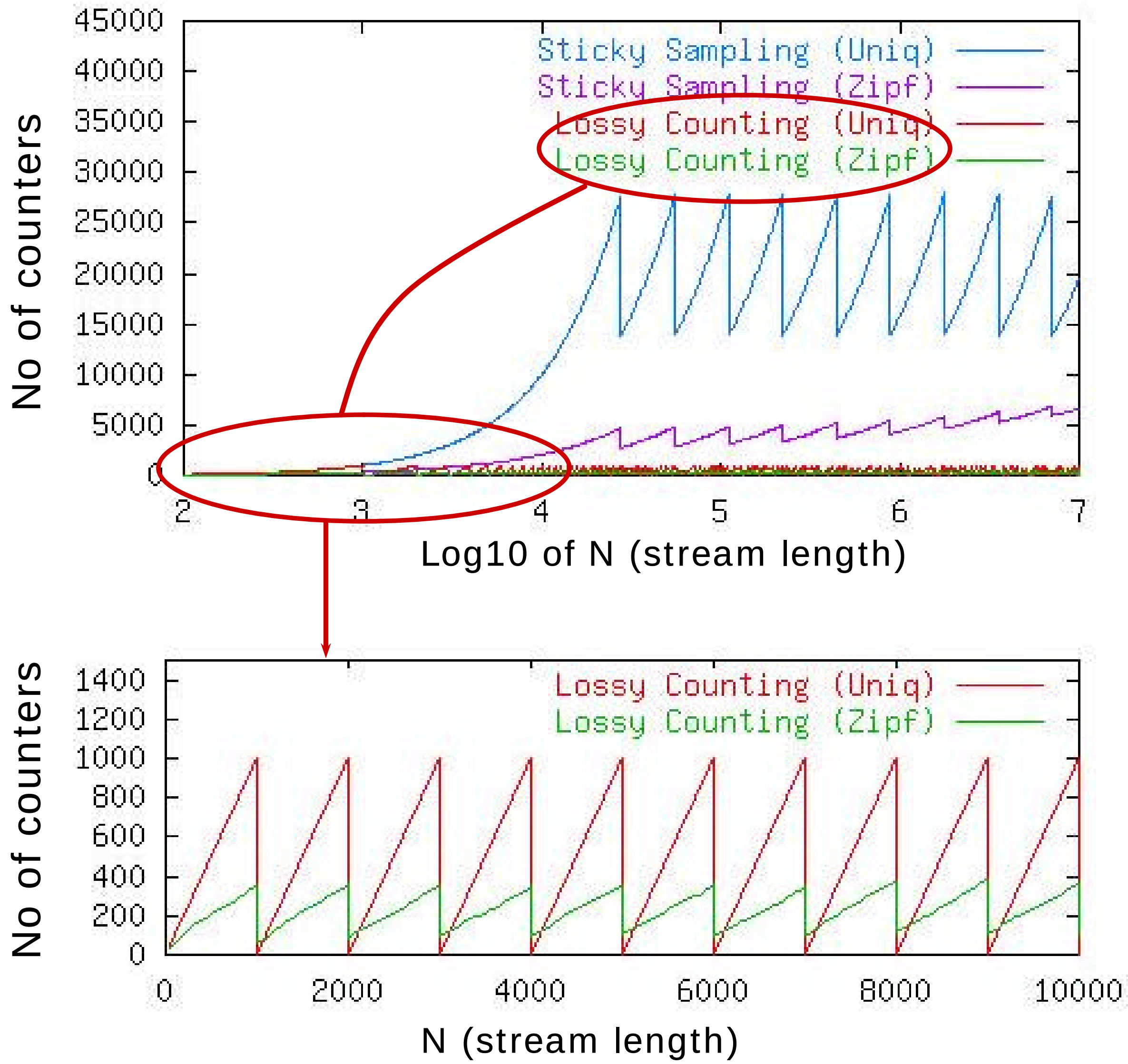
In either case,

Expected number of counters = $\boxed{2/\varepsilon \log 1/s\delta}$

Independent of N!

Sticky Sampling Expected: $2/\varepsilon \log 1/s\delta$
Lossy Counting Worst Case: $1/\varepsilon \log \varepsilon N$

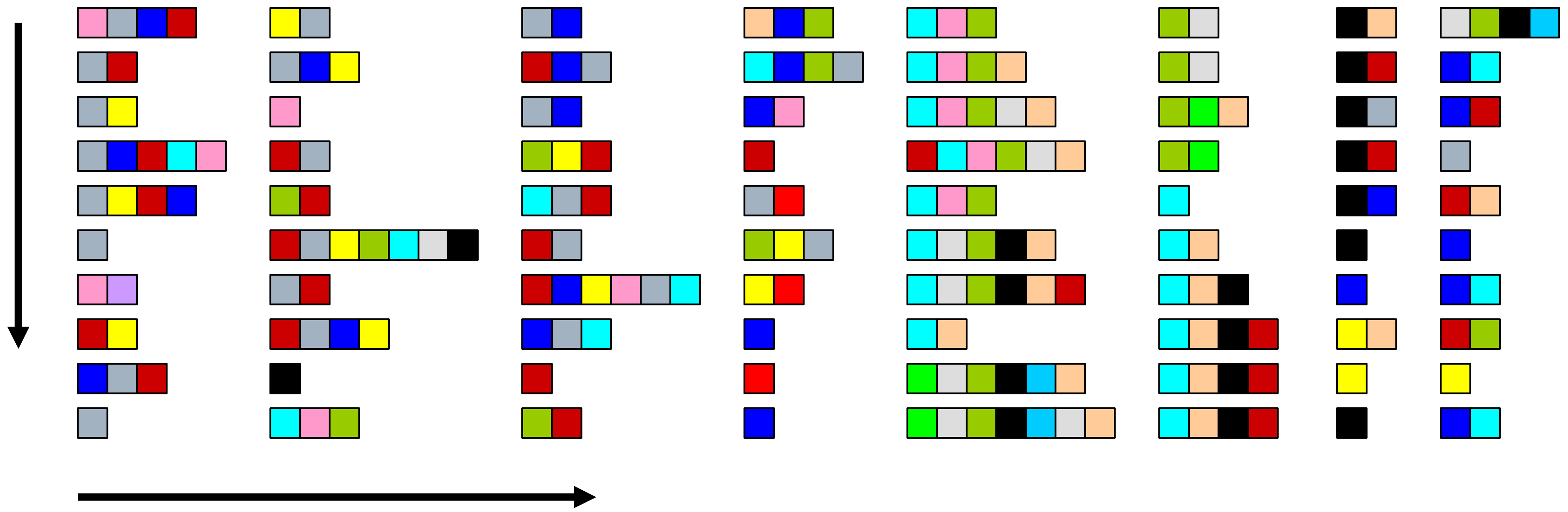
Support $s = 1\%$
Error $e = 0.1\%$



From elements

to *sets* of elements ...

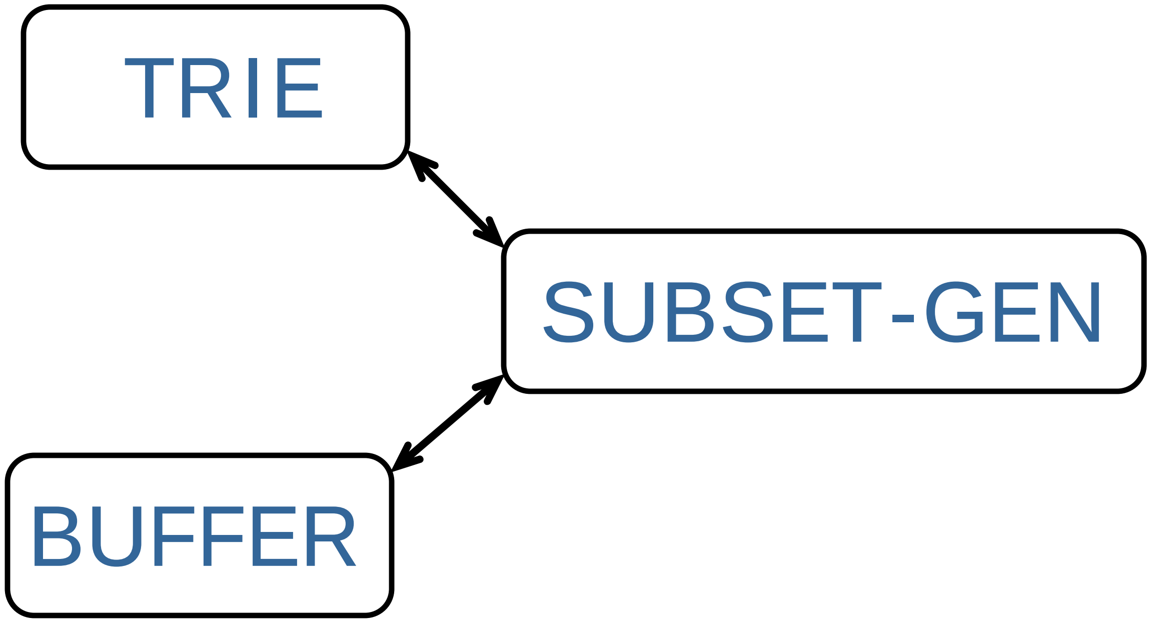
Frequent Itemsets Problem ...



- Identify all subsets of items whose current frequency exceeds s = 0.1%.

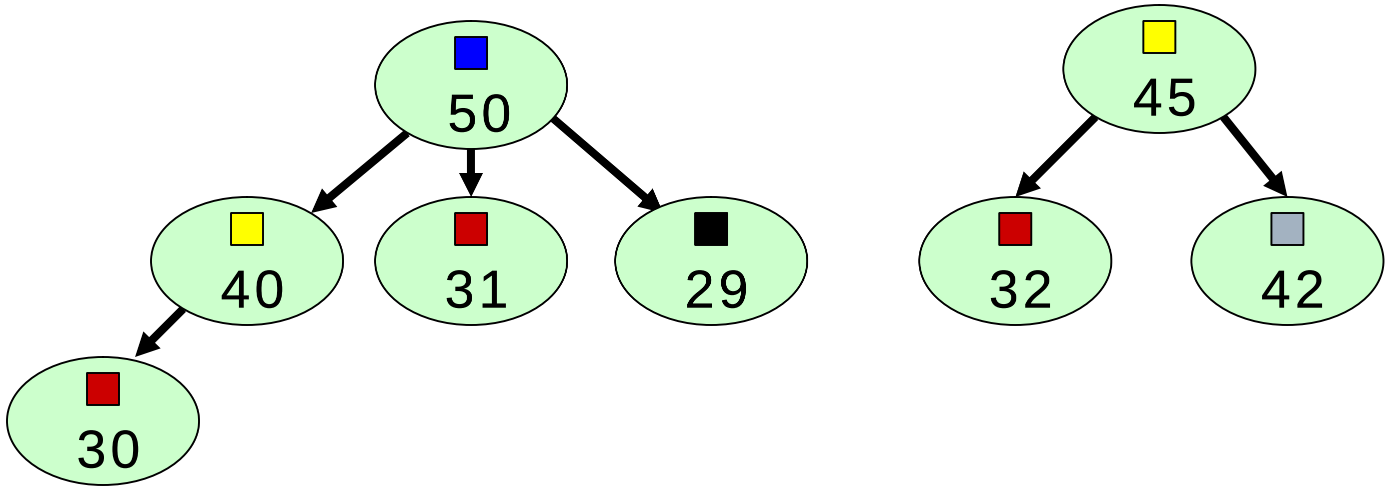
Frequent Itemsets => Association Rules

Three Modules



Module 1: TRIE

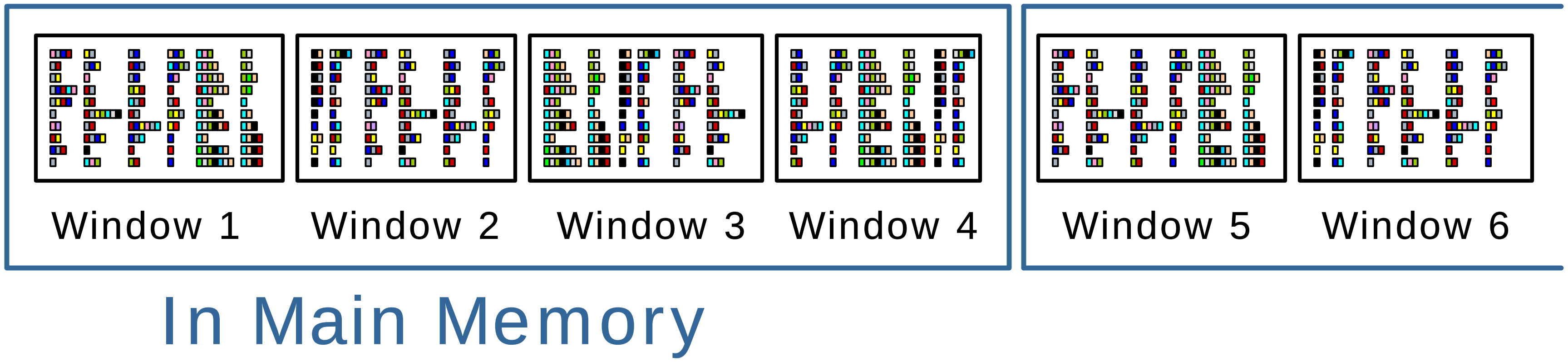
Compact representation of frequent itemsets in lexicographic order.



🟦 50	🟦🟨 40	🟦🟨🟥 30	🟦🟥 31	🟦⬛ 29
🟨 45	🟨🟥 32	🟨⬜ 42		

Sets with frequency counts

Module 2: BUFFER

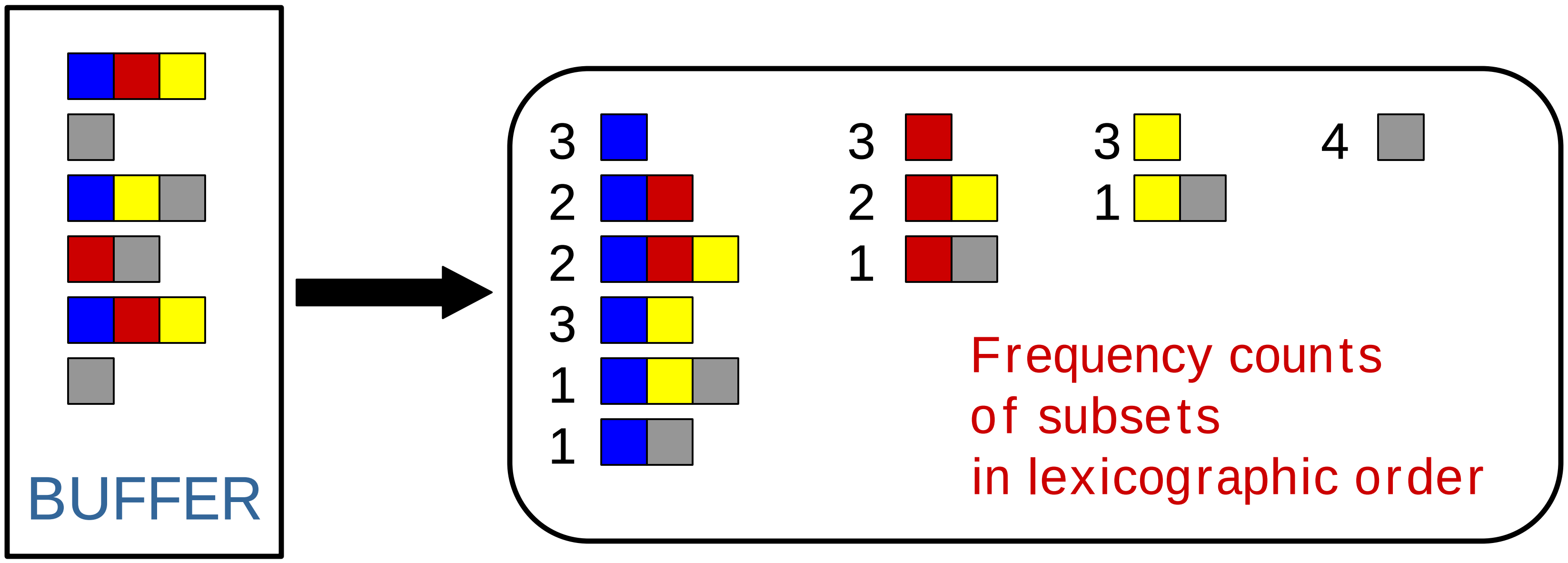


Compact representation as sequence of ints

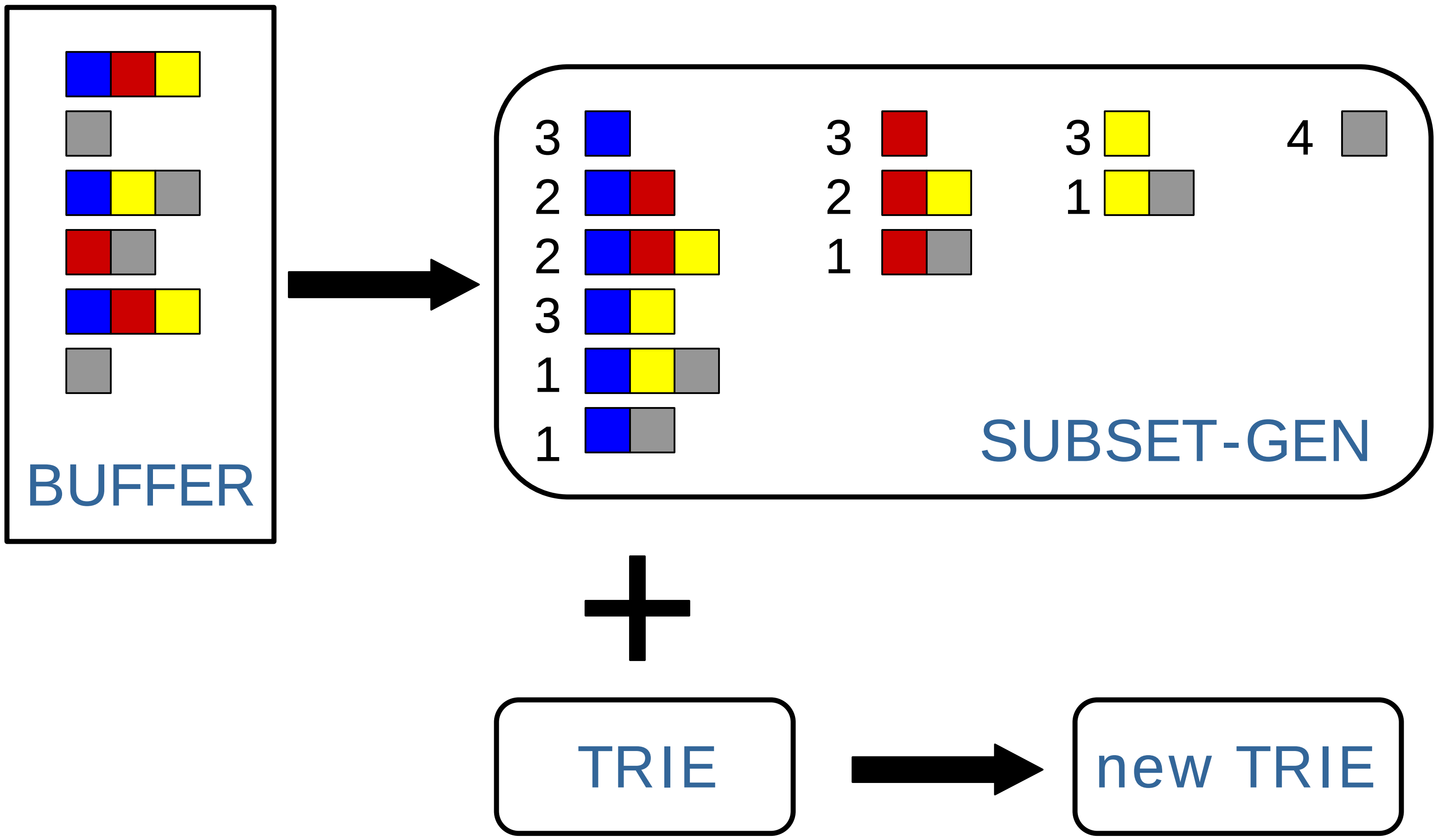
Transactions sorted by item-id

Bitmap for transaction boundaries

Module 3: SUBSET-GEN



Overall Algorithm ...



Problem: Number of subsets is exponential!

SUBSET-GEN Pruning Rules

A-priori Pruning Rule

If set S is infrequent, every superset of S is infrequent.

Lossy Counting Pruning Rule

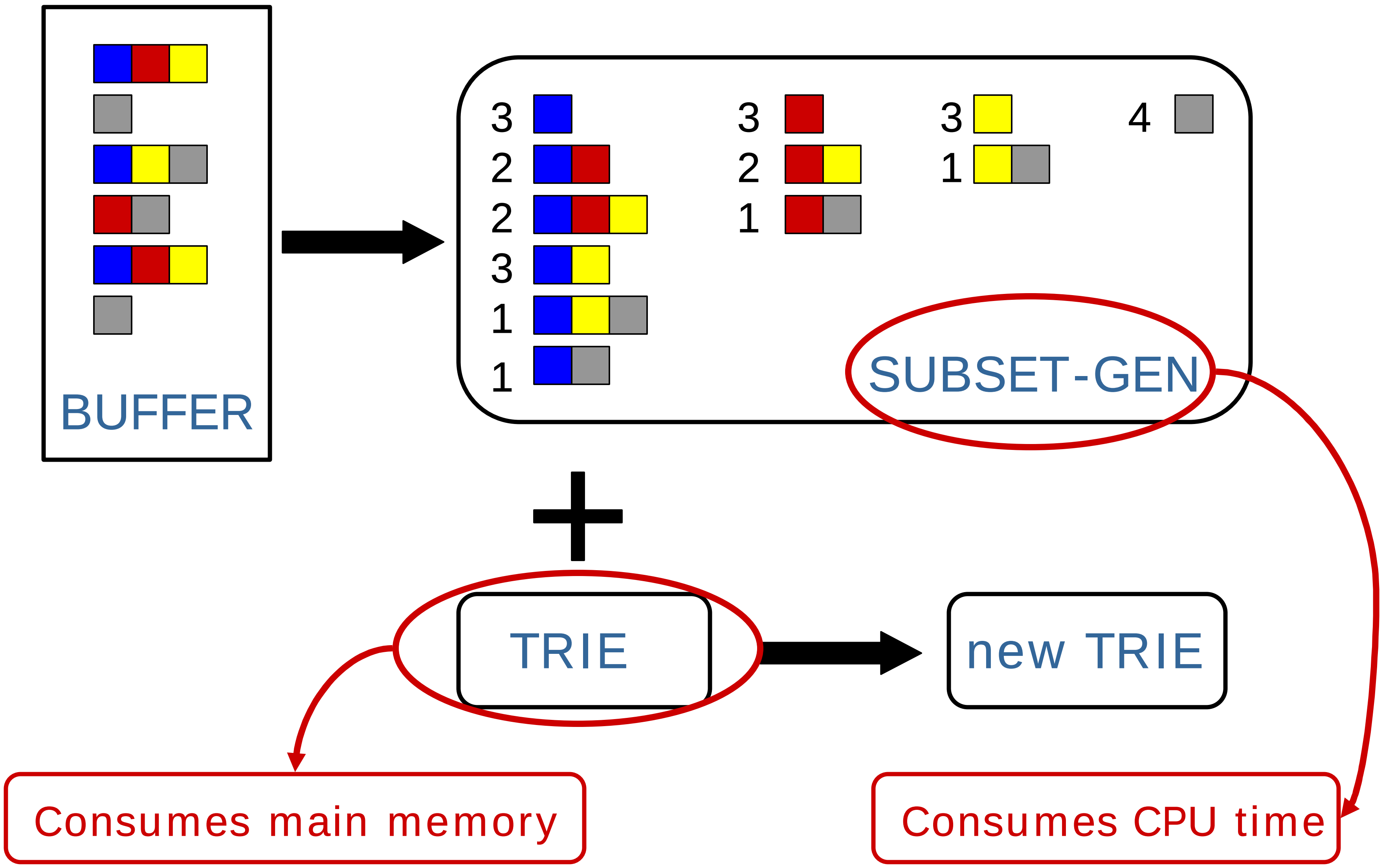
At each ‘window boundary’ decrement TRIE counters by 1.

Actually, ‘Batch Deletion’:

At each ‘main memory buffer’ boundary,
decrement all TRIE counters by b.

See paper for details ...

Bottlenecks …



Design Decisions for Performance

TRIE Main memory bottleneck

Compact linear array

→ (element, counter, level) in preorder traversal

→ No pointers!

Tries are on disk

→ All of main memory devoted to BUFFER

Pair of tries

→ old and new (in chunks)

mmap() and madvise()

SUBSET-GEN CPU bottleneck

Very fast implementation

→ See paper for details

Experiments …

IBM synthetic dataset T10.I4.1000K

N = 1Million   Avg Tran Size = 10   Input Size = 49MB

IBM synthetic dataset T15.I6.1000K

N = 1Million   Avg Tran Size = 15   Input Size = 69MB

Frequent word pairs in 100K web documents

N = 100K   Avg Tran Size = 134   Input Size = 54MB

Frequent word pairs in 806K Reuters newsreports

N = 806K   Avg Tran Size = 61   Input Size = 210MB

What do we study?

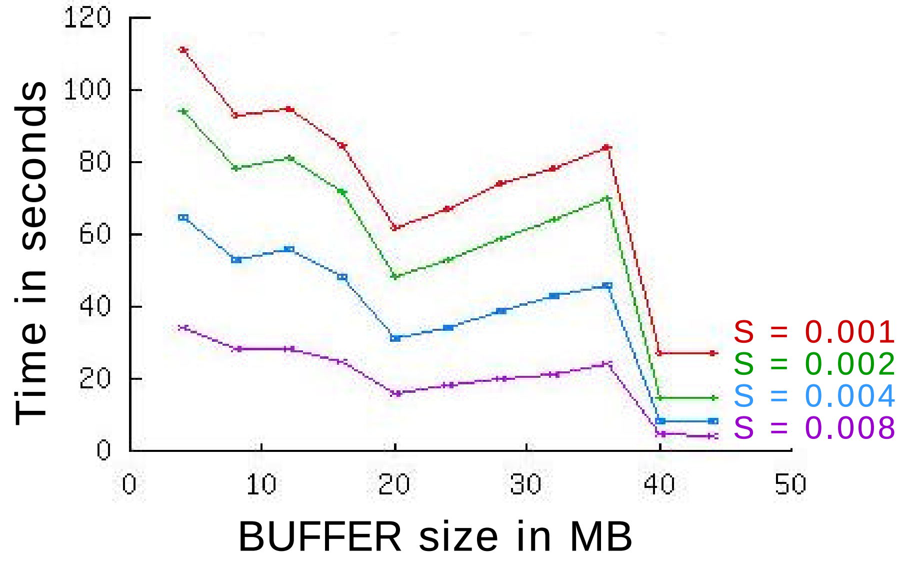
For each dataset

$$\left.\begin{array}{ll}\text{Support threshold} & s \\ \text{Length of stream} & N \\ \text{BUFFER size} & B\end{array}\right\} \begin{array}{l}\text{Three independent variables}\\ \text{Fix one and vary two}\end{array}$$

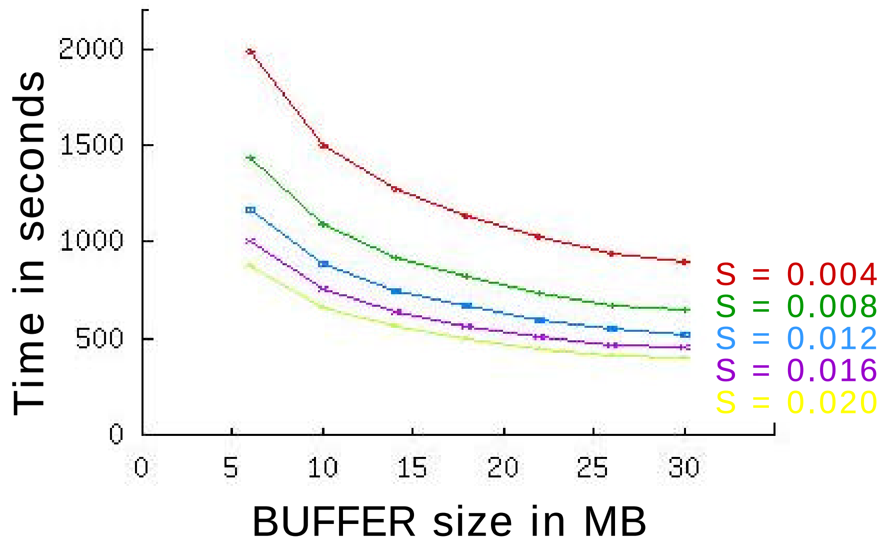
$$\left.\begin{array}{ll}\text{Time taken} & t\end{array}\right\} \text{Measure time taken}$$

$$\boxed{\text{Set } e = 10\% \text{ of support } s}$$

Varying support s and BUFFER B



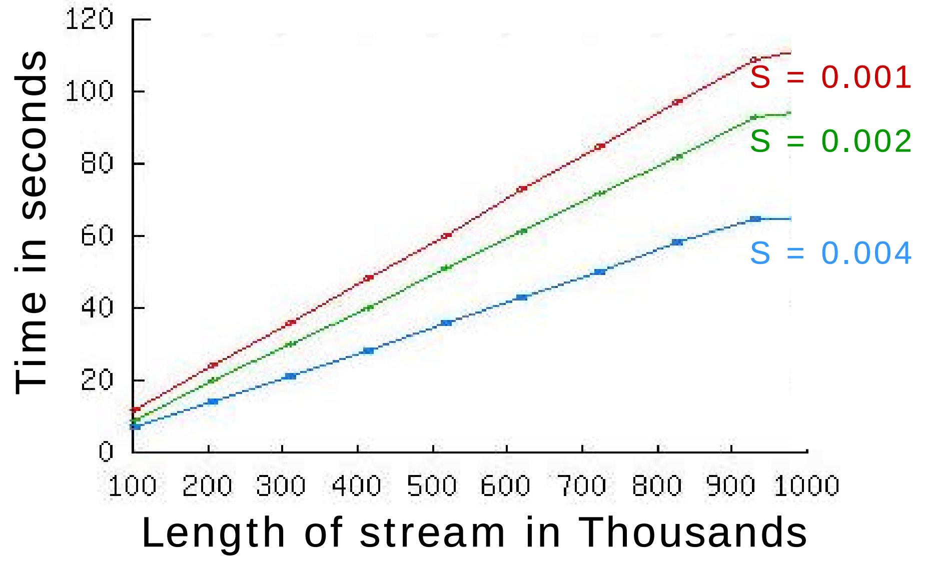
IBM 1M transactions



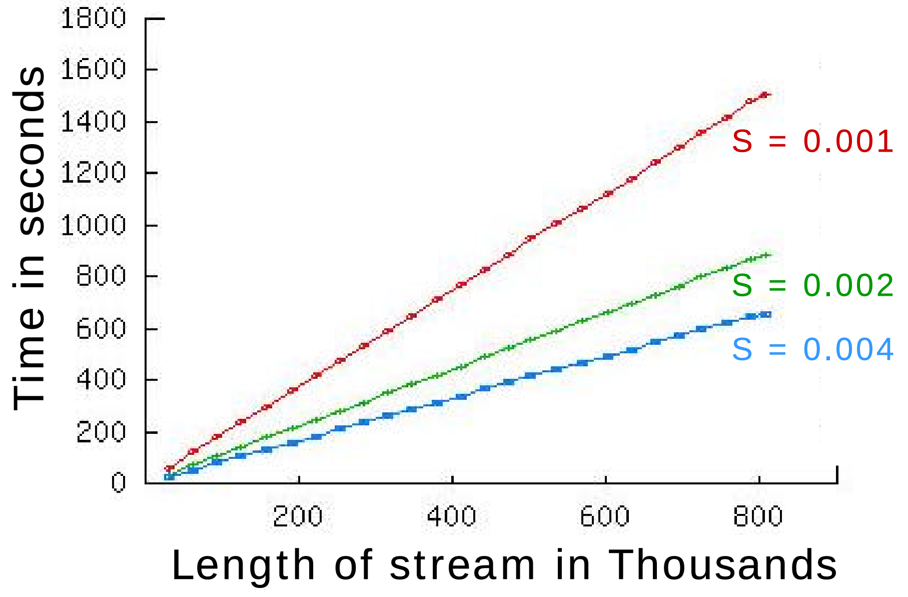
Reuters 806K docs

Fixed:	Stream length	N
Varying:	BUFFER size	B
	Support threshold	s

Varying length N and support s



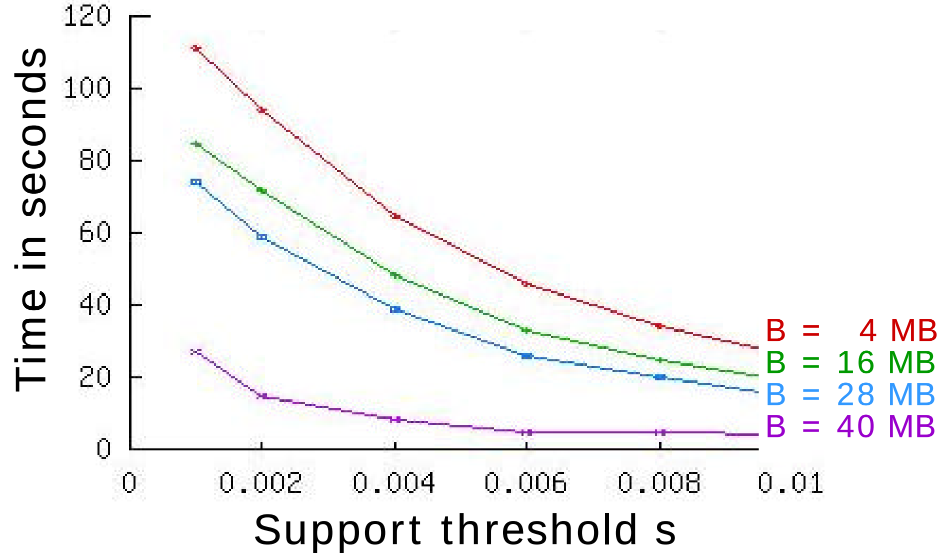
IBM 1M transactions

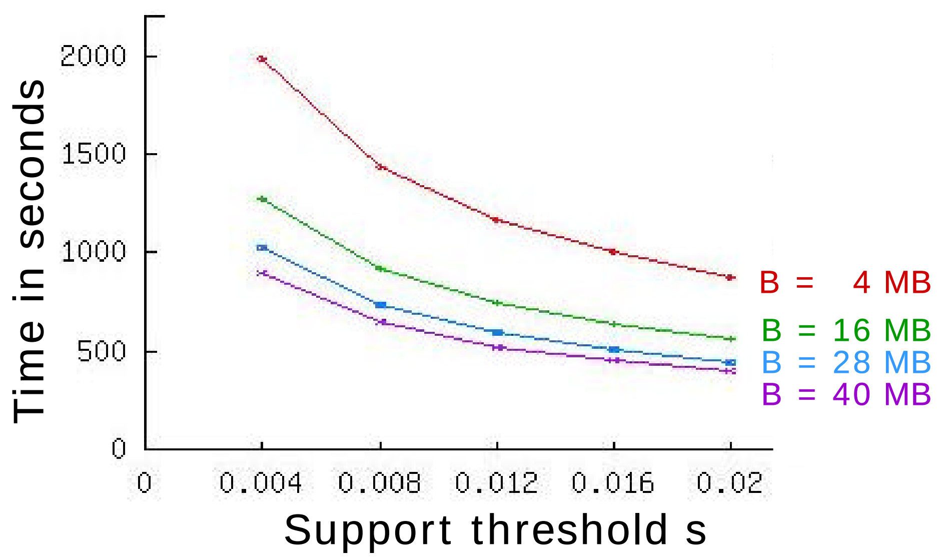


Reuters 806K docs

Fixed:	BUFFER size	B
Varying:	Stream length	N
	Support threshold	s

Varying BUFFER B and support s





IBM 1M transactions

Reuters 806K docs

Fixed:	Stream length	N
Varying:	BUFFER size	B
	Support threshold	s

Comparison with fast A-priori

	APriori		Our Algorithm with 4MB Buffer		Our Algorithm with 44MB Buffer	
Support	Time	Memory	Time	Memory	Time	Memory
0.001	99 s	82 MB	111 s	12 MB	27 s	45 MB
0.002	25 s	53 MB	94 s	10 MB	15 s	45 MB
0.004	14 s	48 MB	65 s	7MB	8 s	45 MB
0.006	13 s	48 MB	46 s	6 MB	6 s	45 MB
0.008	13 s	48 MB	34 s	5 MB	4 s	45 MB
0.010	14 s	48 MB	26 s	5 MB	4 s	45 MB

Dataset: IBM T10.I4.1000K with 1M transactions, average size 10.

A-priori by Christian Borgelt http://fuzzy.cs.uni-magdeburg.de/~borgelt/software.html

Comparison with Iceberg Queries

Query: Identify all word pairs in 100K web documents
which co-occur in at least 0.5% of the documents.

[FSGM+98] multiple pass algorithm:
7000 seconds with 30 MB memory

Our single-pass algorithm:
4500 seconds with 26 MB memory

Our algorithm would be much faster if allowed multiple passes!

Lessons Learnt …

Optimizing for #passes is wrong!

Small support s Þ Too many frequent itemsets!

Time to redefine the problem itself?

Interesting combination of Theory and Systems.

Work in Progress …

Frequency Counts over Sliding Windows

Multiple pass Algorithm for Frequent Itemsets

Iceberg Datacubes

Summary

Lossy Counting: A Practical algorithm for online frequency counting.

First ever single pass algorithm for Association Rules with user specified error guarantees.

Basic algorithm applicable to several problems.