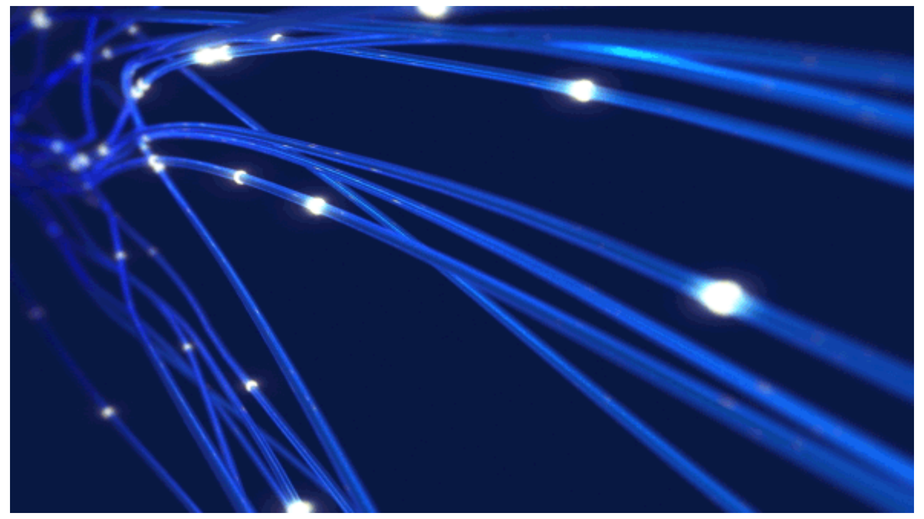
Data Streams

What is data stream?

- Large Data Set:
  - Continuous
  - Massive
  - Unbounded
  - Possibly infinite



- Fast changing and requires fast, real-time response
- Example: Traffic to popular website (Facebook, Google, Amazon)

Data stream management

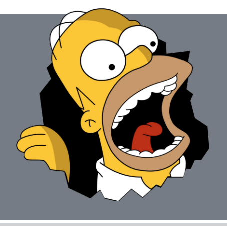
- Essential for many applications such as:
  - Financial applications
  - Network monitoring

Problems?

- Stream is hard to:
  - Store - Data is *continuously growing* faster than our ability to store or index it
  - Process
  - Transfer

Difficult to Process

Unbounded number of elements

					
Count	4	5	3	4	3

…but we want finite space, depending on the problem

Difficult to Process

- Short processing time for each element in the stream
- Only one single pass at the data

Why stream processing is important?

Answering queries about this sort of data requires clever observation techniques and data compressing methods

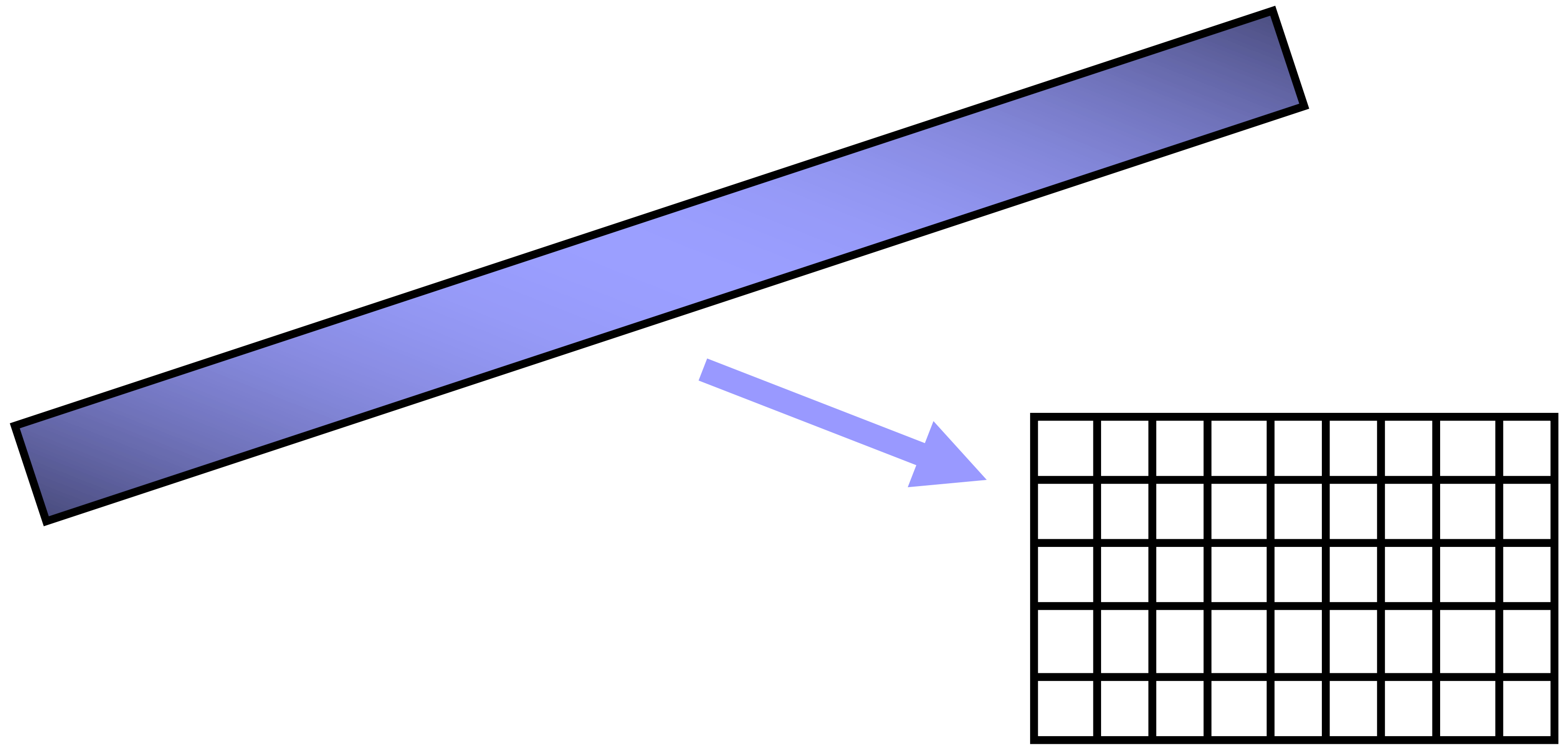
Generic solution

Stream processing algorithms often build compact approximate sketches of the input stream

Sketch

- Try to build a small data-structure to represent the data you want to obtain from the stream

- The smaller the data structure, the less accurate the results



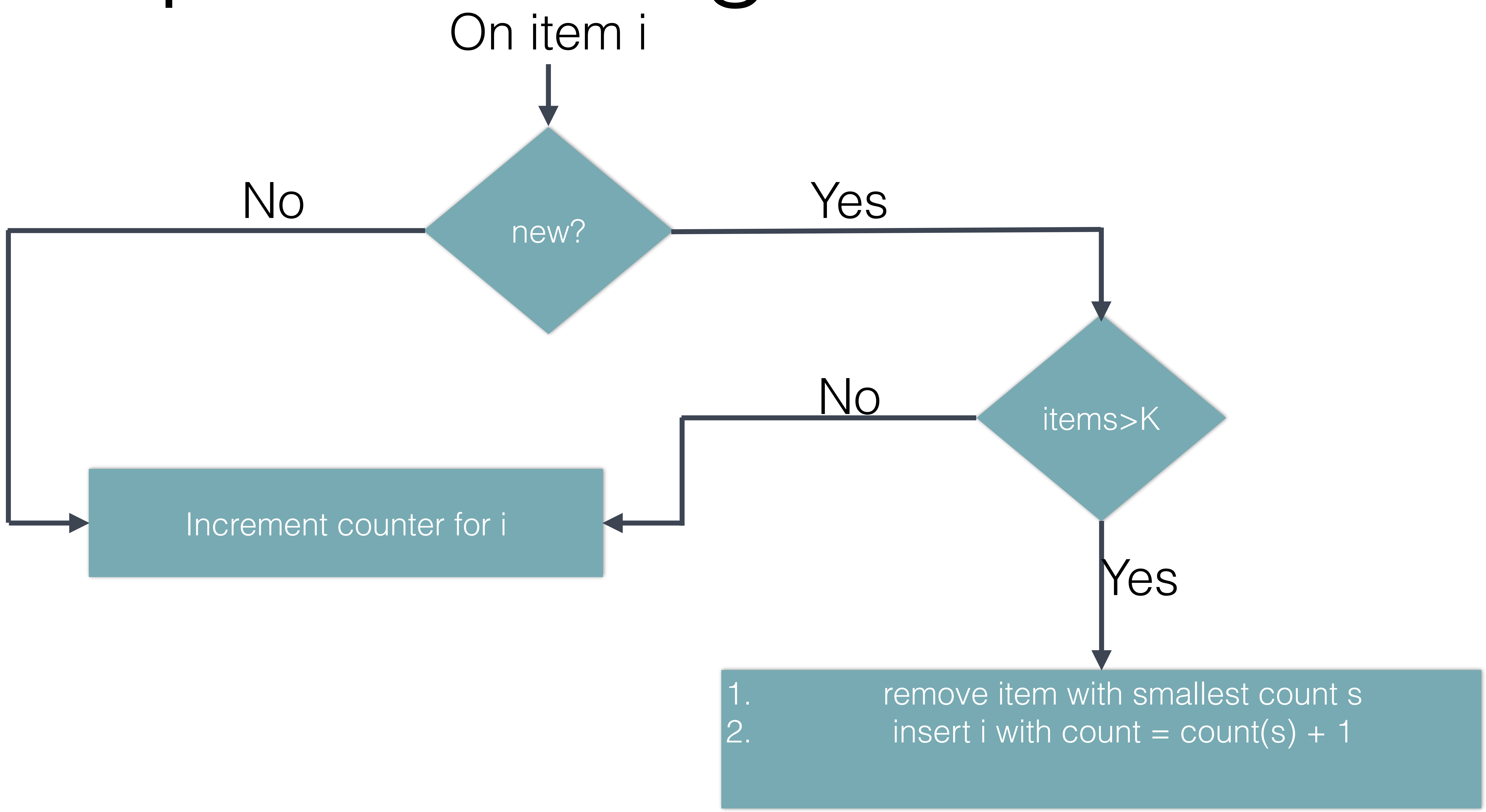
Generic solution

Almost all algorithms are approximate, answer with error and guarantee a bound on the error

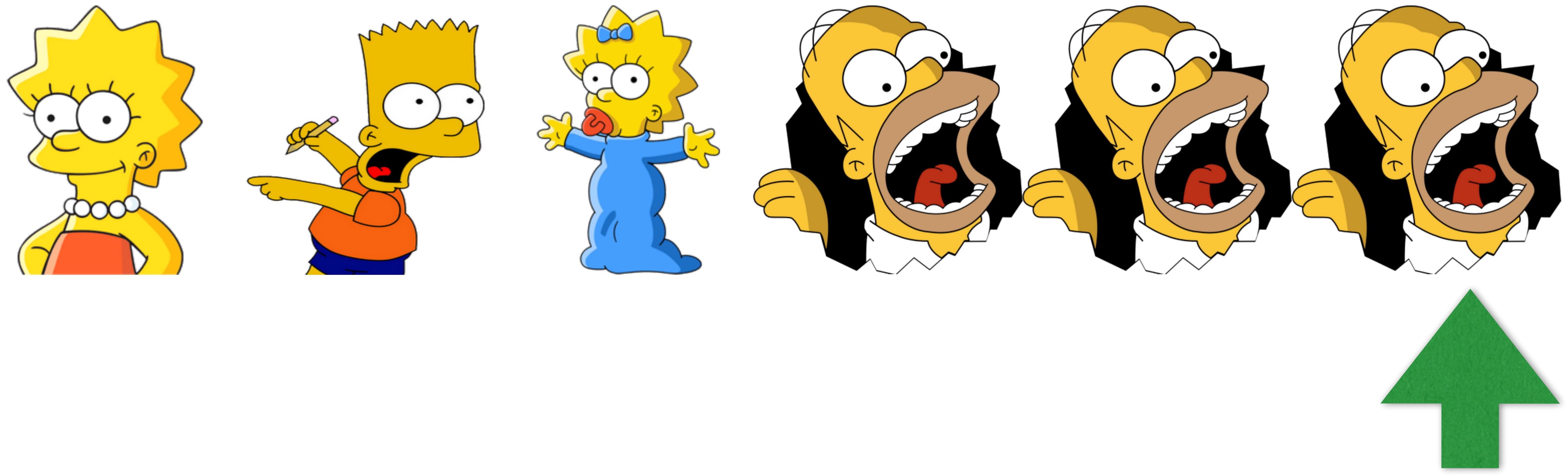
Space saving algorithm

- Keep `k` items and counts initially zero
- Count first `k` distinct item exactly

Space saving model



Space saving algorithm



K=3

	3
	2
	1

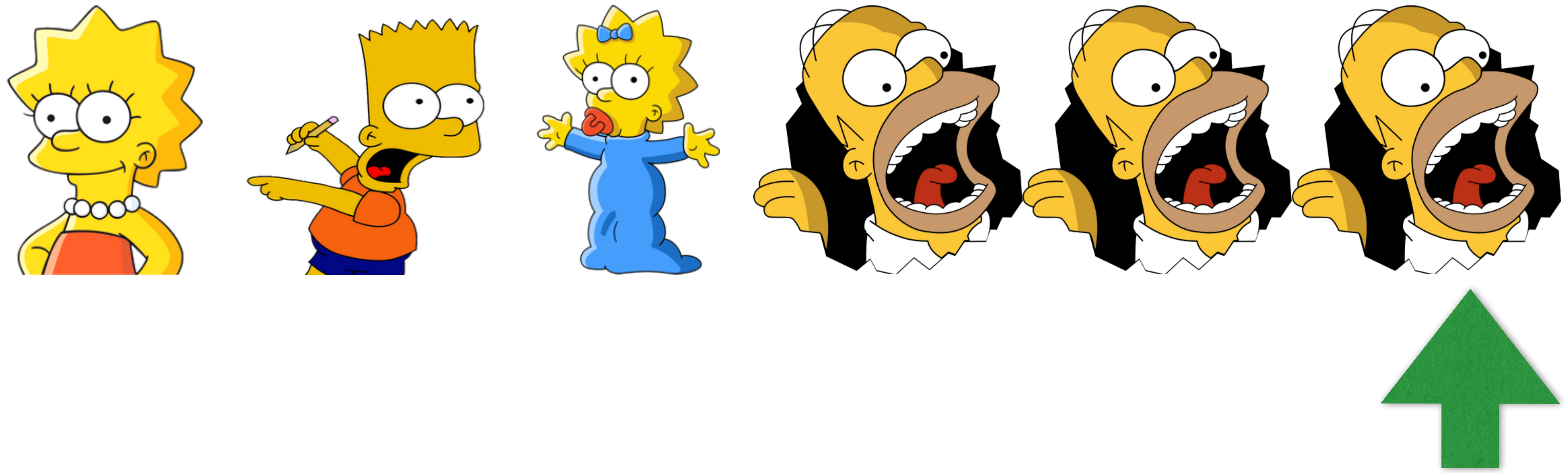
Space saving Guarantees

- When
  $k = 1/\varepsilon$
- We denote the overall number of insertion by Z
- The minimal counter is at most Z
- The estimation error is Z

ε

ε

Space saving algorithm

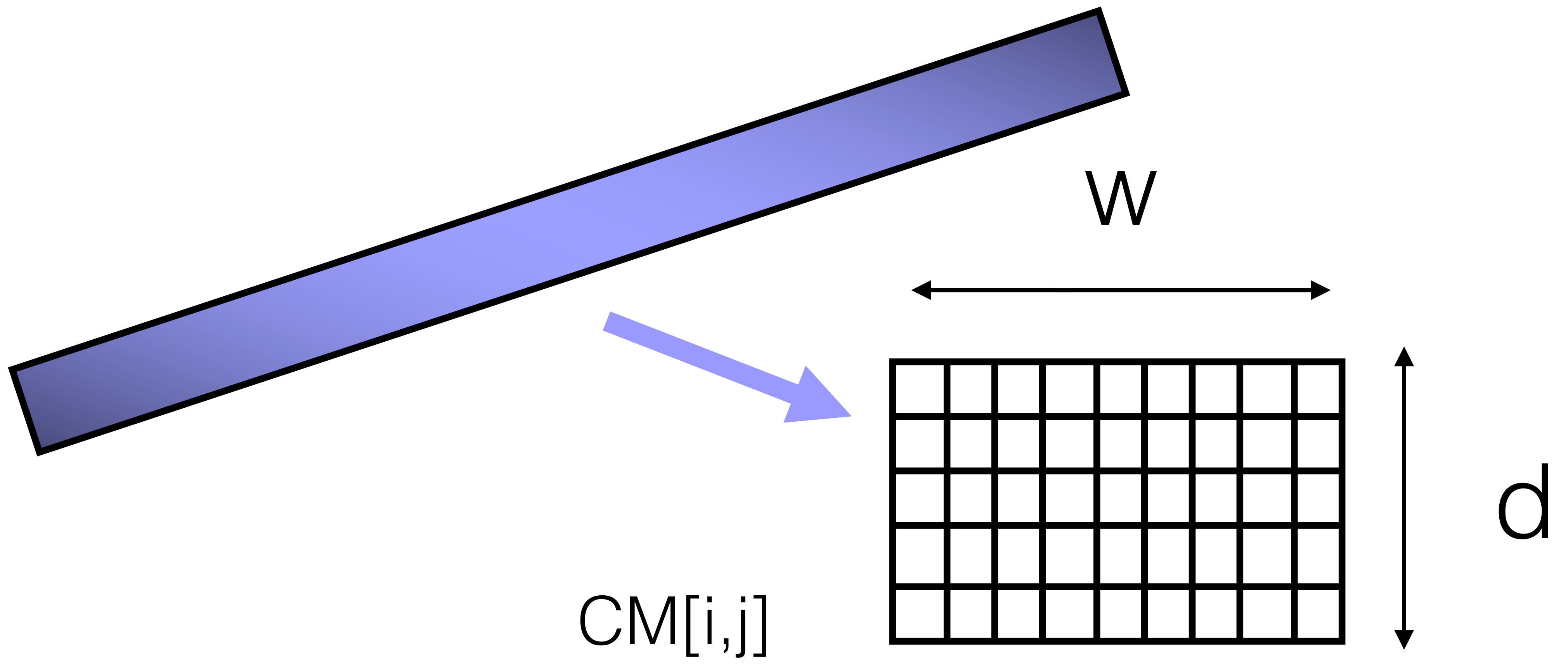


K=3

	3
	2
	1

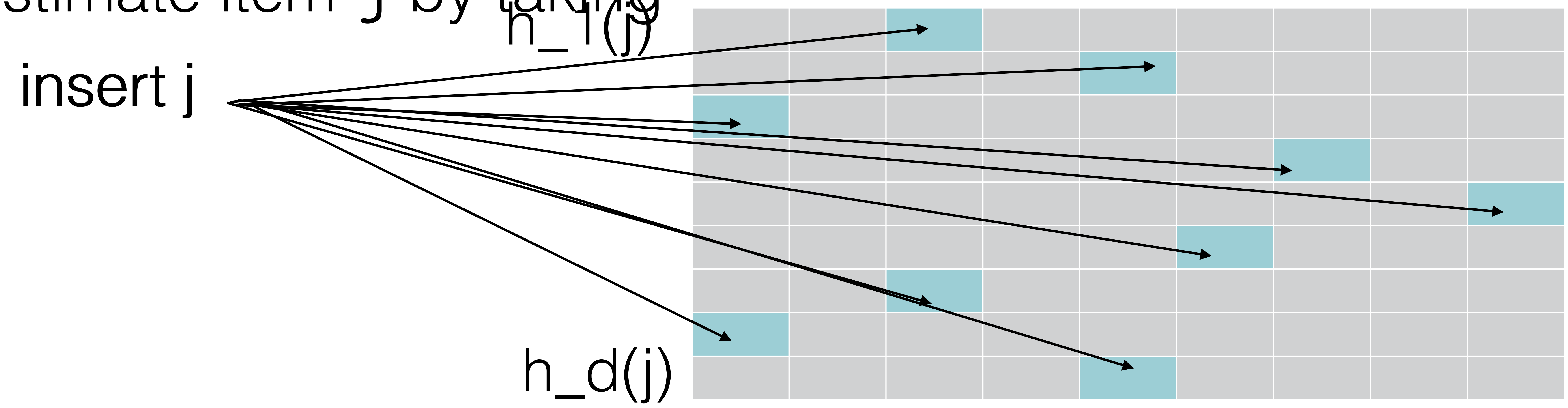
Count min sketch

- Sketch that estimate item's frequency over **`a`**
- Creates a small summary as an array of **`w`*****`d`**
- Use **`d`** hash functions to map to [**`1…w`**]



Count min sketch

- Estimate item `j` by taking



$$\min_k \{CM[k, h_k(j)]\}$$

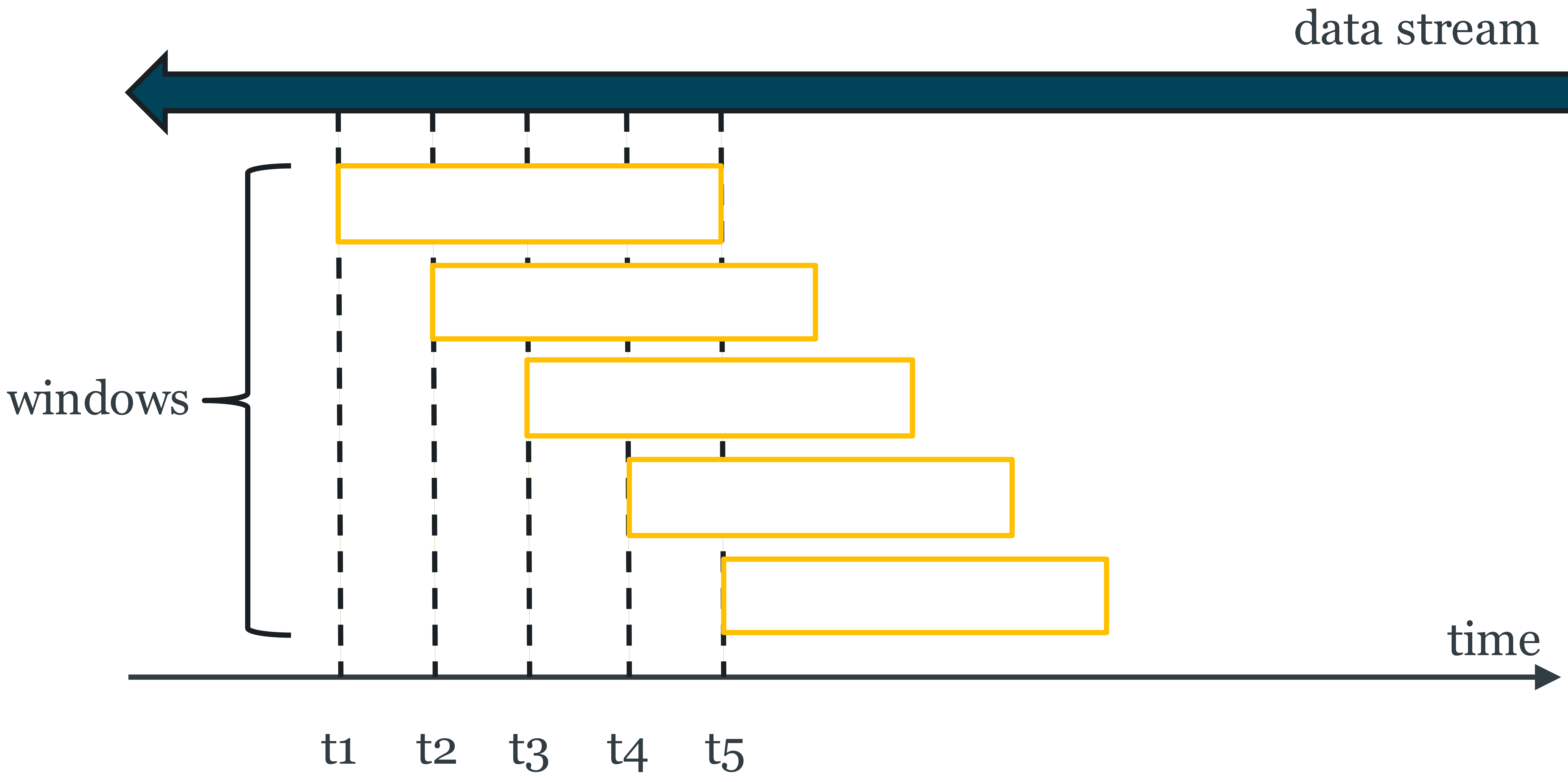
Count min sketch Guarantees

- CM sketch guarantees approximation error on point queries less than $\varepsilon||A||$(input stream as a vector A) in space $O(1/\varepsilon \log 1/\delta)$
- Probability of error is less than $1-\delta$

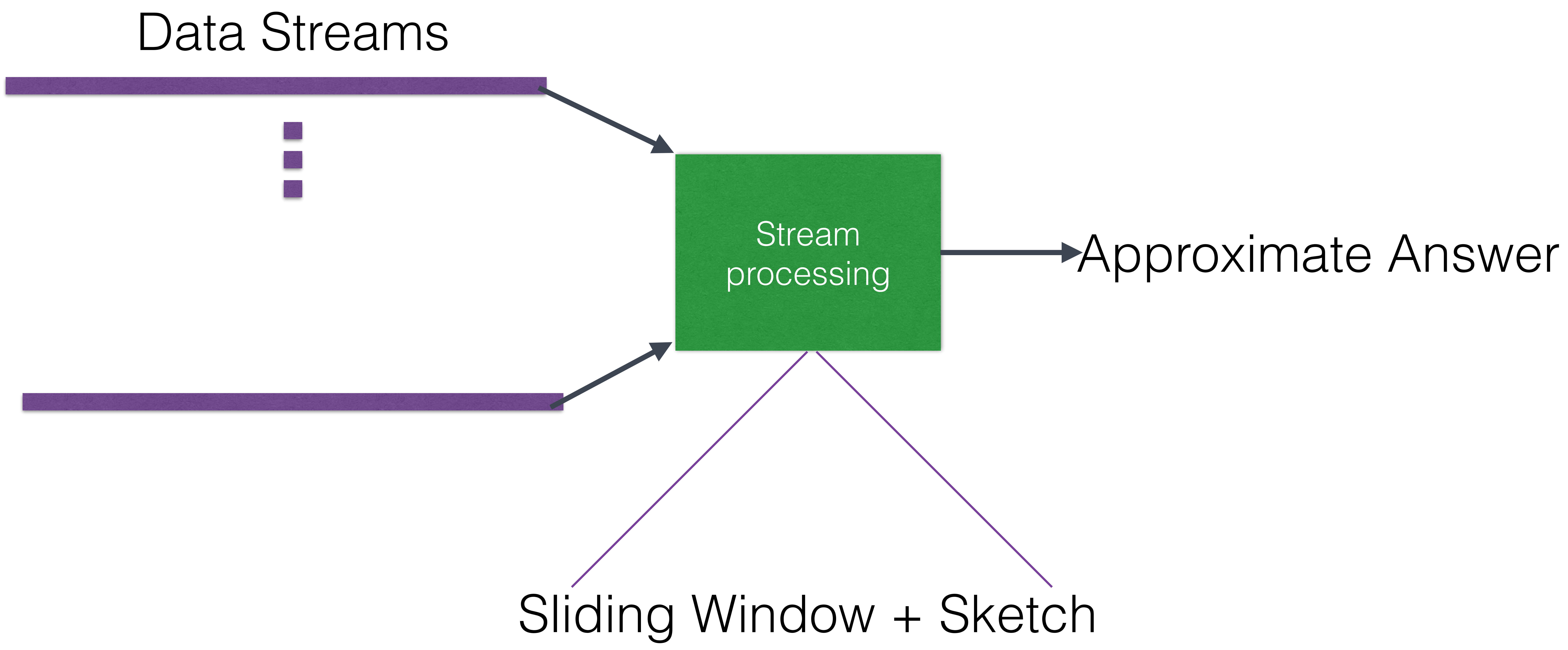
Sliding windows

- For most applications, OLD data is considered less relevant
- Apply aging mechanism for the sketches
- Sliding Window Model:
  - Only last “`W`” elements are considered

Sliding windows



Computation Model



The Problem with existing sliding window solutions

The window of interest may not be known a priori

OR

may be multiple interesting windows

Contribution

We study a model that allows the user to specify an interval of interest that is contained within the last `W` items at query time

We improve space and operation performance of the existing work

Existing Works - ECM

- Introduce sketching technique, called ECM - Exponential Count Min
- ECM combines count-min sketch with Exponential histograms
- Exponential histograms is a sliding window counter that can guarantee a bounded relative error

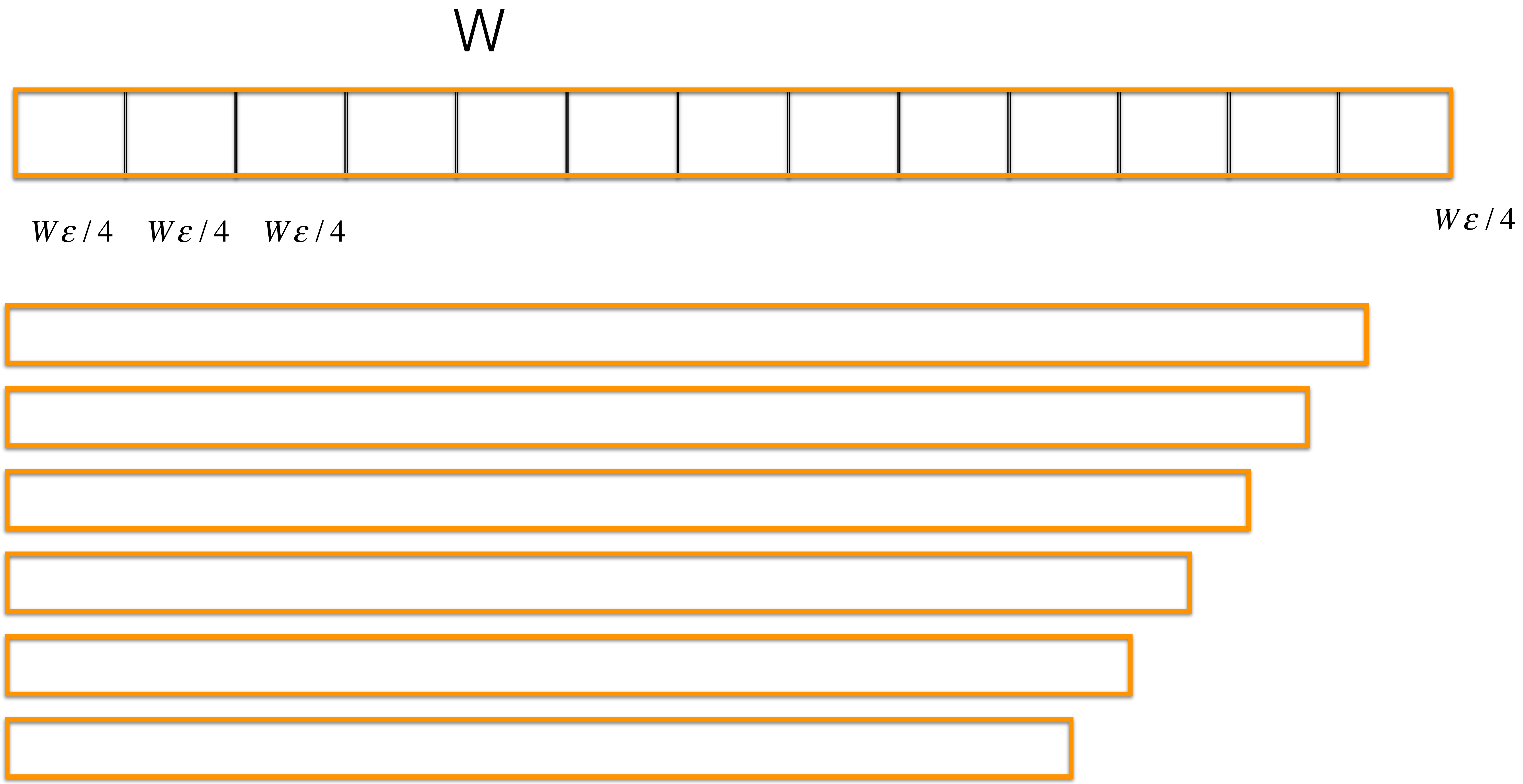
Existing Works - ECM

- ECM sketch replace each Count-Min counter with Exponential histogram

- ECM has an error probability

Naive solution: Raw

- Uses several instances of a black box algorithms that solves frequency estimated over fixed sized window
- Add(**x**): Add item **x** to all instances
- Interval Query:
  1. Query the relevant instances (closest to interval range)
  2. Subtract the result


W
$W\varepsilon/4$
$W\varepsilon/4$
$W\varepsilon/4$
$W\varepsilon/4$

RAW vs. ECM

- RAW achieves constant query time while ECM answers queries in $O(\varepsilon^{-1} \log W \log \delta^{-1})$
- Both consumes same amount of memory
- RAW is deterministic while ECM has an error probability

Problem Definition

- Add(`x`): Given an element `x`, append it to stream
- IntervalFrequency(`x,i,j`): Return an estimation of `x` frequency between the `i` and `j` most recent elements of `S`, denoted by $\hat{f}_x^{\,i,j}$

$$(W,\varepsilon)-IntervalFrequency:$$

$$f_x^{i,j} \le \hat{f}_x^{i,j} \le f_x^{i,j} + W\varepsilon$$

n-interval problem

- Arriving elements are inserted to blocks

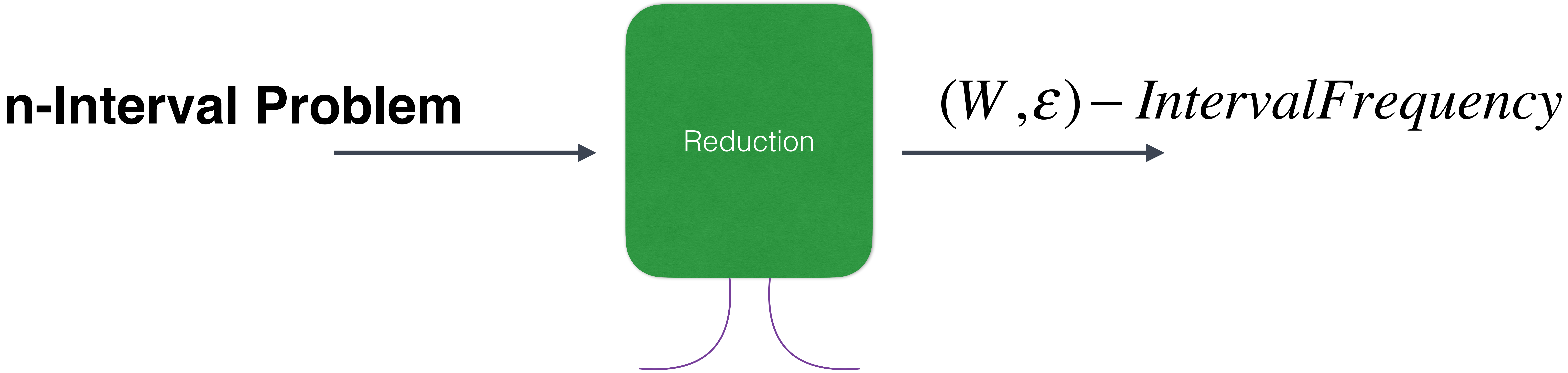
x	d	x			y	b,d,y		x,d	x	x		d		b		x	c

- Compute exact block interval frequencies within the blocks

N=18

N-Interval problem Definition

- n-Interval Problem: Block Interval Frequency
- Add(`x`): Given an element `x`, append it to stream
- EndBlock(): New block inserted, old block leaves
- IntervalQuery(`x,i,j`): Return the number (without error) of blocks `x` appears between blocks `i,j`



Decide when to add elements in the blocks

Reduction

1. Break the stream into `W` sized frames

2. Divide each frame into n-equal-sized blocks, each of size $W\varepsilon$

3. Employ Space Saving to track element frequency within each frame

    1. Whenever a counter reaches an integer multiple of the block size, associated it to most recent block

    2. When the frame ends, flush Space Saving instance

Implementing ADD(x)

x

Space Saving

offset:0

x

If result mod block size = 0

if offset mod block size =0
EndBlock

Implementing IntervalFrequency(`x,i,j`)

1. Compute relevant blocks number
2. Call Query of n-interval problem
3. Return block size * result

n-Interval Problem → Reduction → $(W,\varepsilon) - IntervalFrequency$

Advanced algorithms

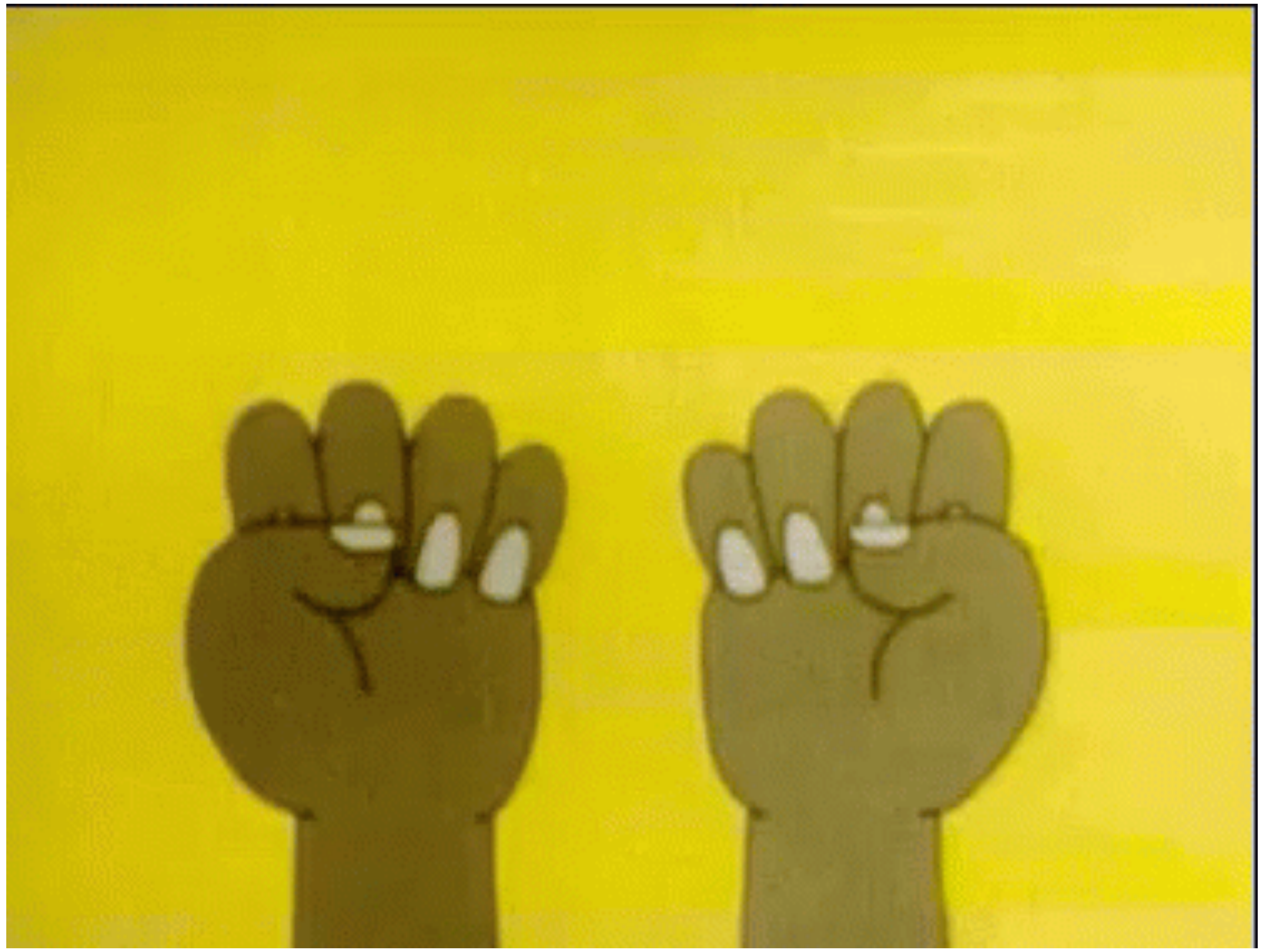
1. ACC_K Algorithms

2. HIT Algorithm

Solve n-Interval problem

Acc Algorithm

Approximate Cumulative Count



Acc algorithm

- Family of algorithms that solves n-interval problem
- ACC_k solves the problem using at most **`k`** hash tables for update and **`2k+1`** for queries
- The larger **`k`** is, The algorithm takes less space but is also slower
- Break the stream into frames of size n (maximal window size)
  - Any n sized window intersects with at most two frames

ACC_1 algorithm

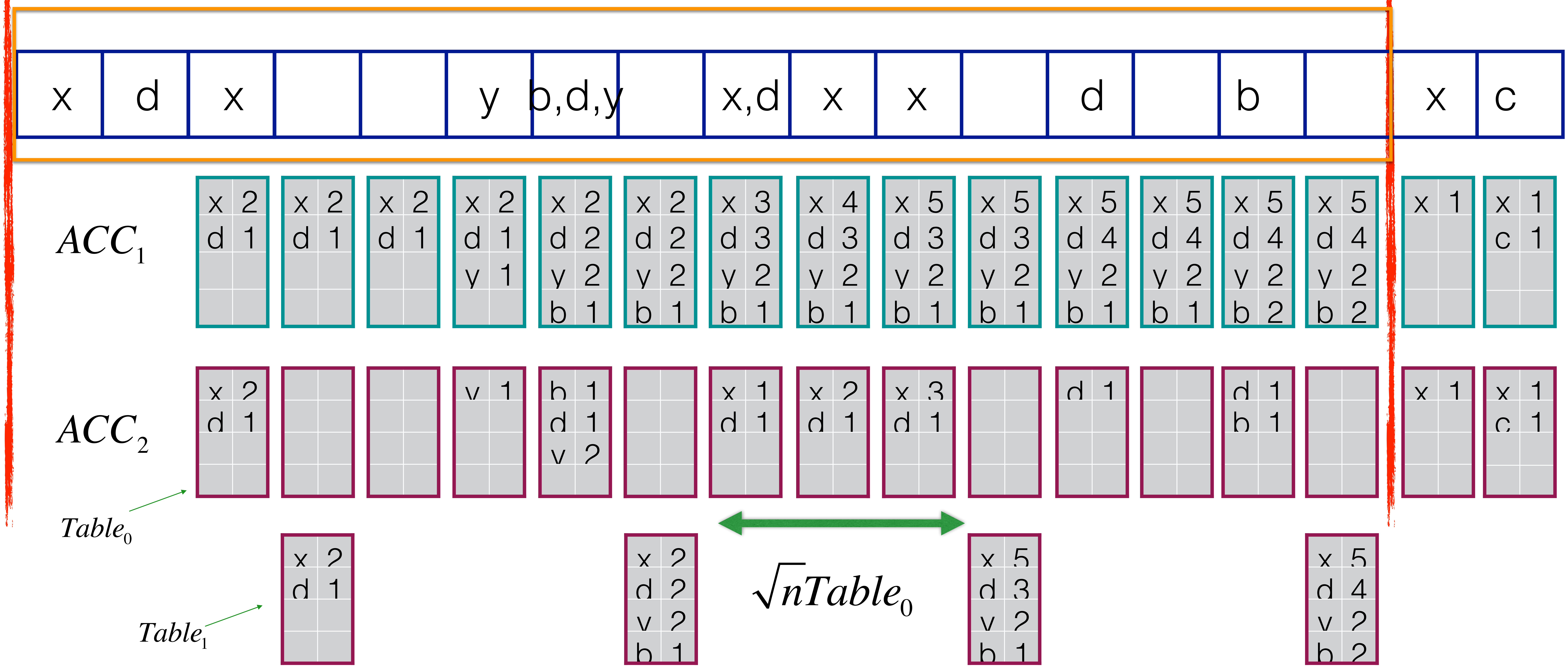
- Each block has a table that tracks how many times each item has arrived from the beginning of the frame
- Query at most 3 tables:
  - Within the frame - compute interval by subtracting 2 tables
  - If it crosses two frames, one additional query

wasteful?!

ACC_2 algorithm

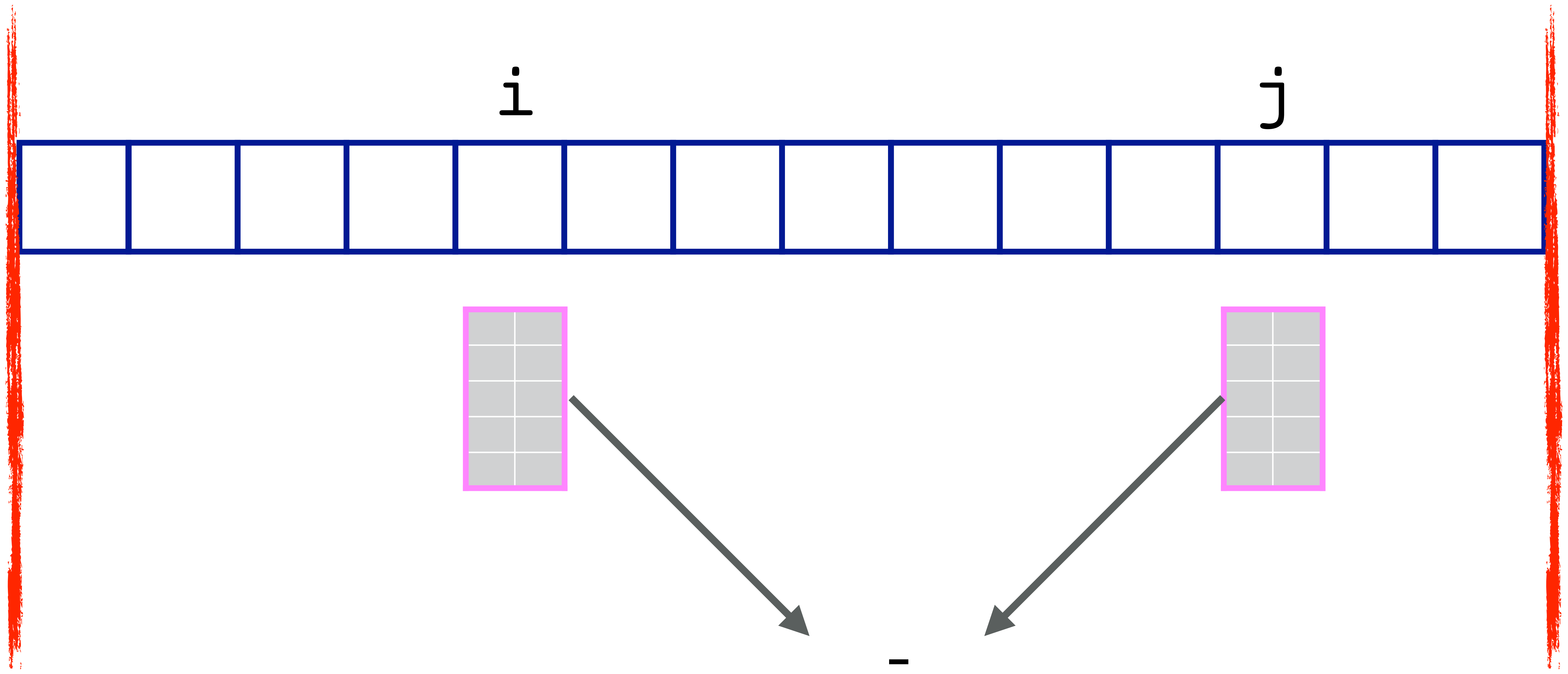
- Saves space at expense of additional table access
- Breaks each frame to $\sqrt{n}$ sized segments
- At end of each segments, we keep level-1 table that counts item frequencies from the beginning of the frame
- level-0 tables computes frequency within a segment for each block

ACC Algorithm



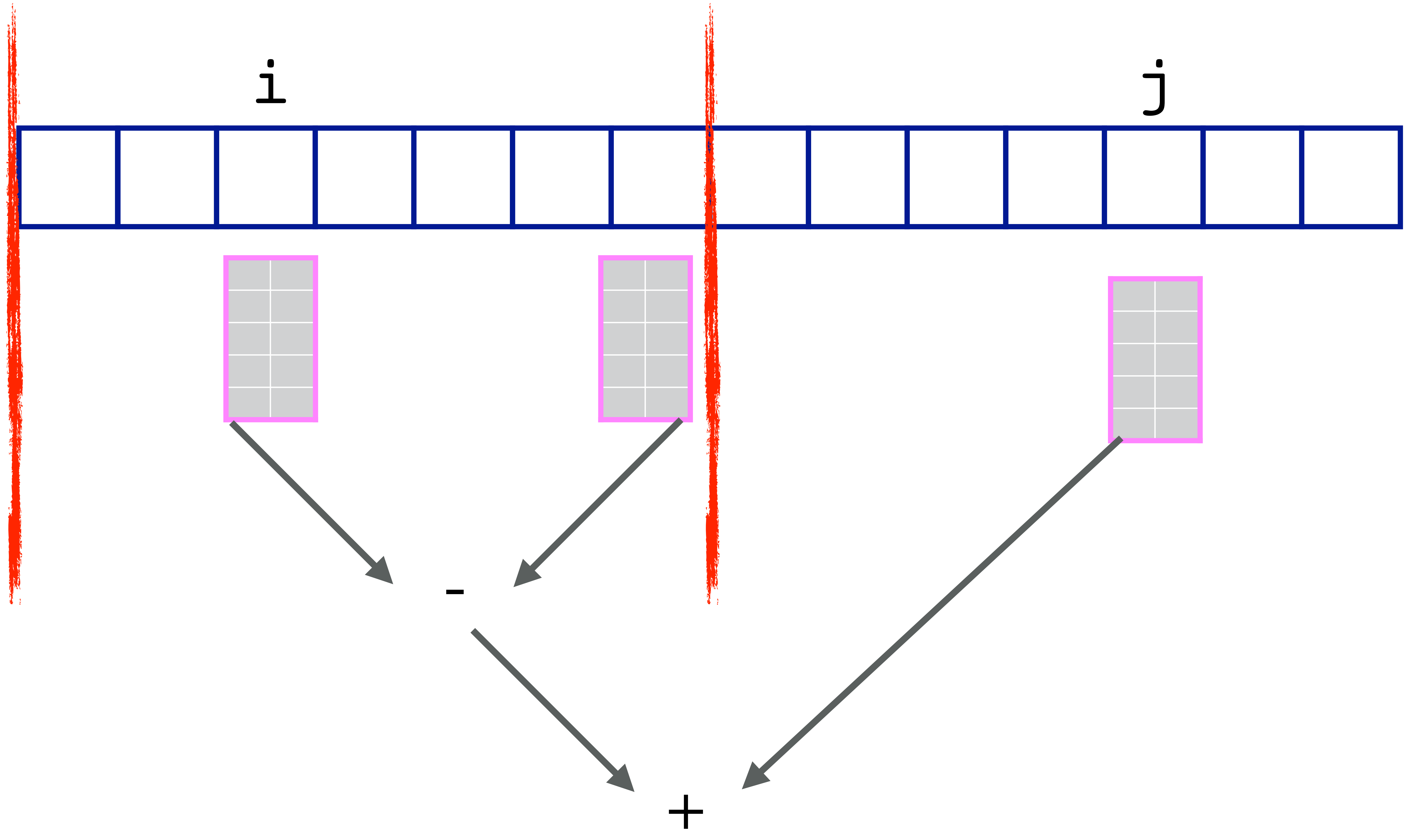
Answering interval Frequency query

- For [`i,j`], let block_`i`, block_`j` be the relevant blocks
    - If block_`i` and block_`j` are in the same frame:



Answering interval Frequency query

- If block_`i` and block_`j` are NOT in the same frame:



Answering interval Frequency query

- Corner case: table may includes blocks that already left the table $level_l$

- Solution: Maintain ghostTables for leaving segments

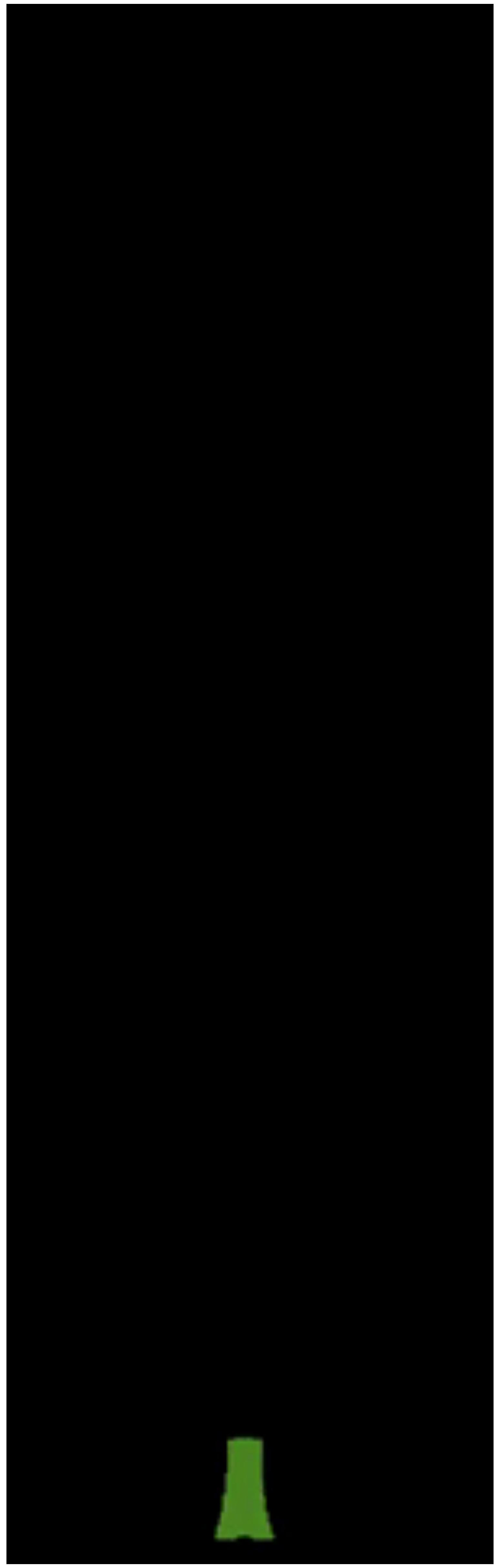
  - - contains table if last leaving block

- Subtract the corresponding ghostTable as well

$ghostTable[l]$ $level_l$

Hit Algorithm

Hierarchical Interval Tree



HIT Algorithm

- Uses hierarchical tree structure
- Nodes stores partial frequency of its sub-tree
- tracks how many times each item arrived within block
- $level_0$ of tracks how many items arrived between

$level_l$ $block_i$

$[block_{i-2^l+1}, block_i], 0 < l \leq trailing\_zeros(i)$

HIT Algorithm

- Each level contains tables for half the blocks of previous level
- Higher levels of the tree allow efficient time computation

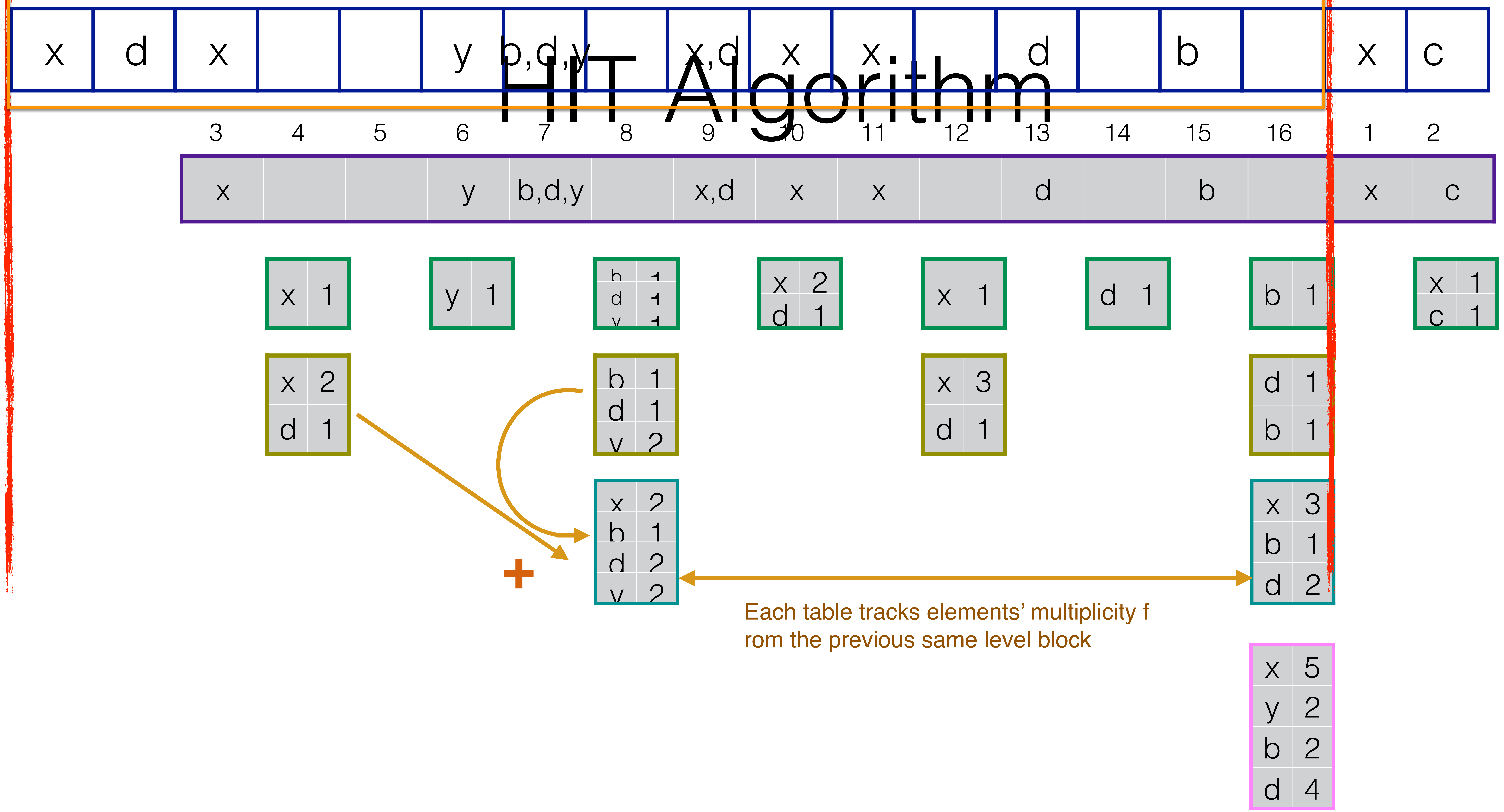
HIT Algorithm

Answering interval Frequency query

- For [`i,j`], let block_`i`, block_`j` be the relevant blocks
  - Scan backward from block_`j to` block_`i,` Greedily using the highest possible level at each point.
  - If block_`j`> block_`i` all tables are valid
  - Otherwise, use level_0 between block_0 to block_j and compute block_`i` to block_`n` as before

-

HIT Algorithm



Each table tracks elements' multiplicity f
rom the previous same level block

Answering interval Frequency query

- Query computation takes at most $2 \log n$ steps
- Corner case: Content of a table may refers to a departing block
- Solution: Choose always the highest valid level of valid tables

Evaluations

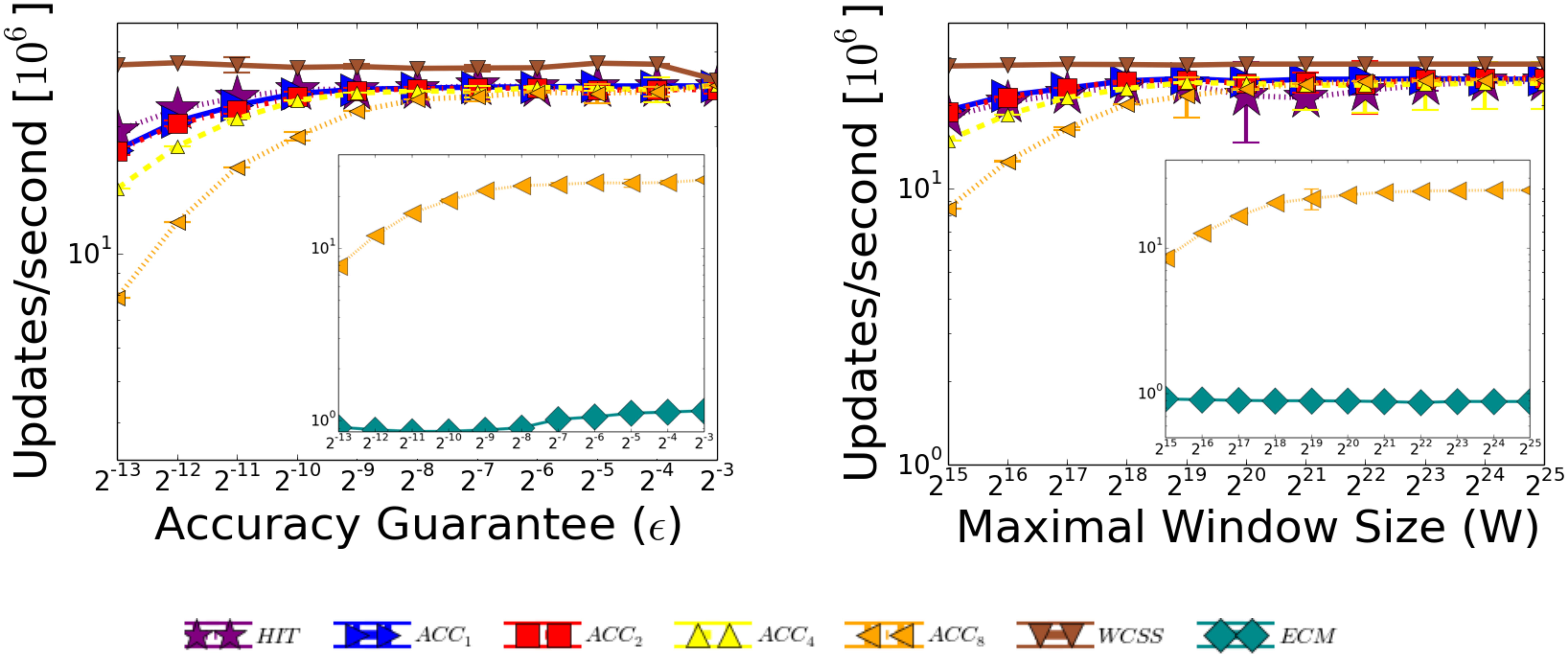
Setup

- C++ implementation
- ECM is configured with
- Backbone dataset

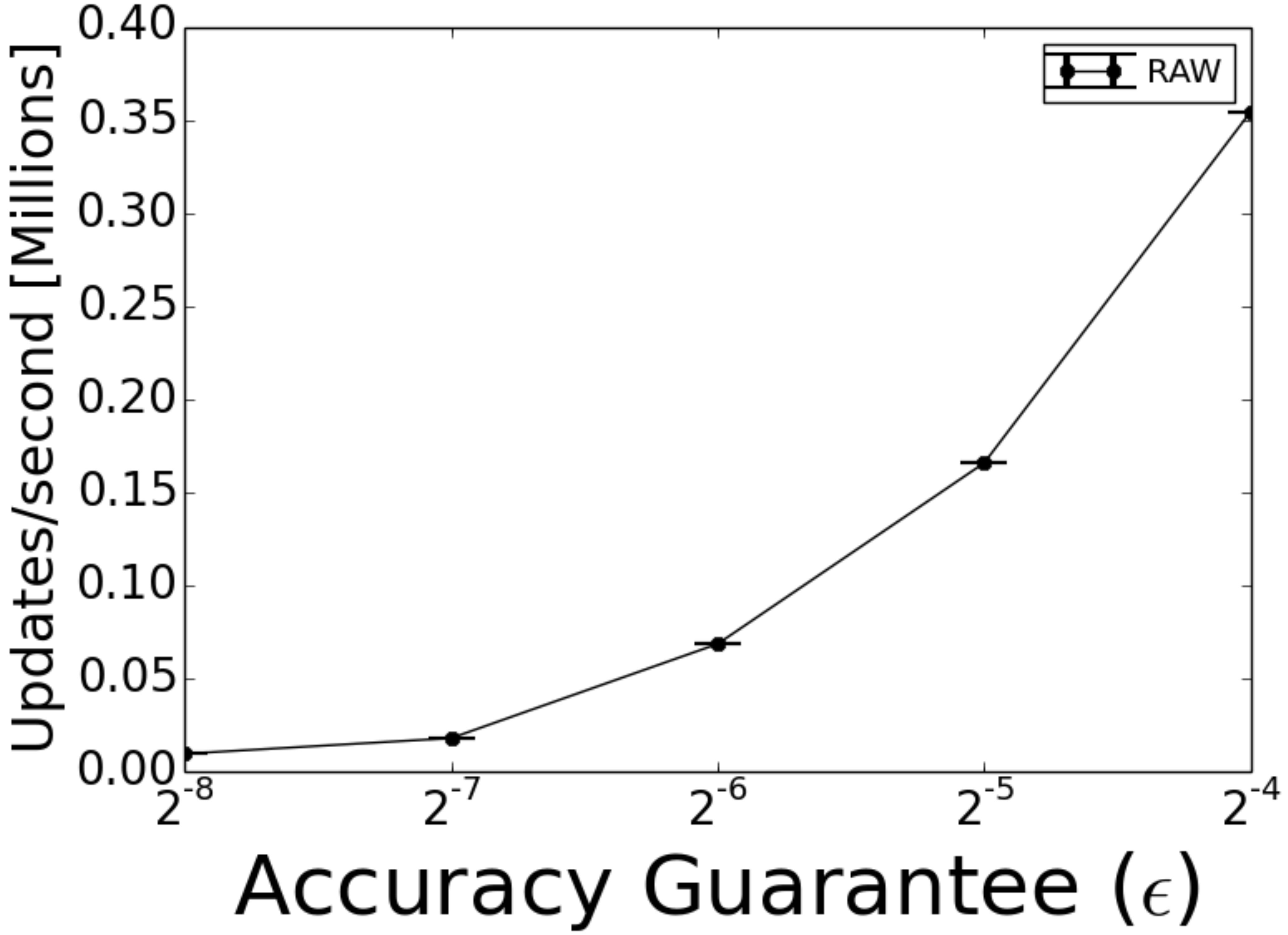
$$\delta = 0.01\%$$

```
W= 2^20, epsilon = 2^-8, interval size = 1%*Window
```

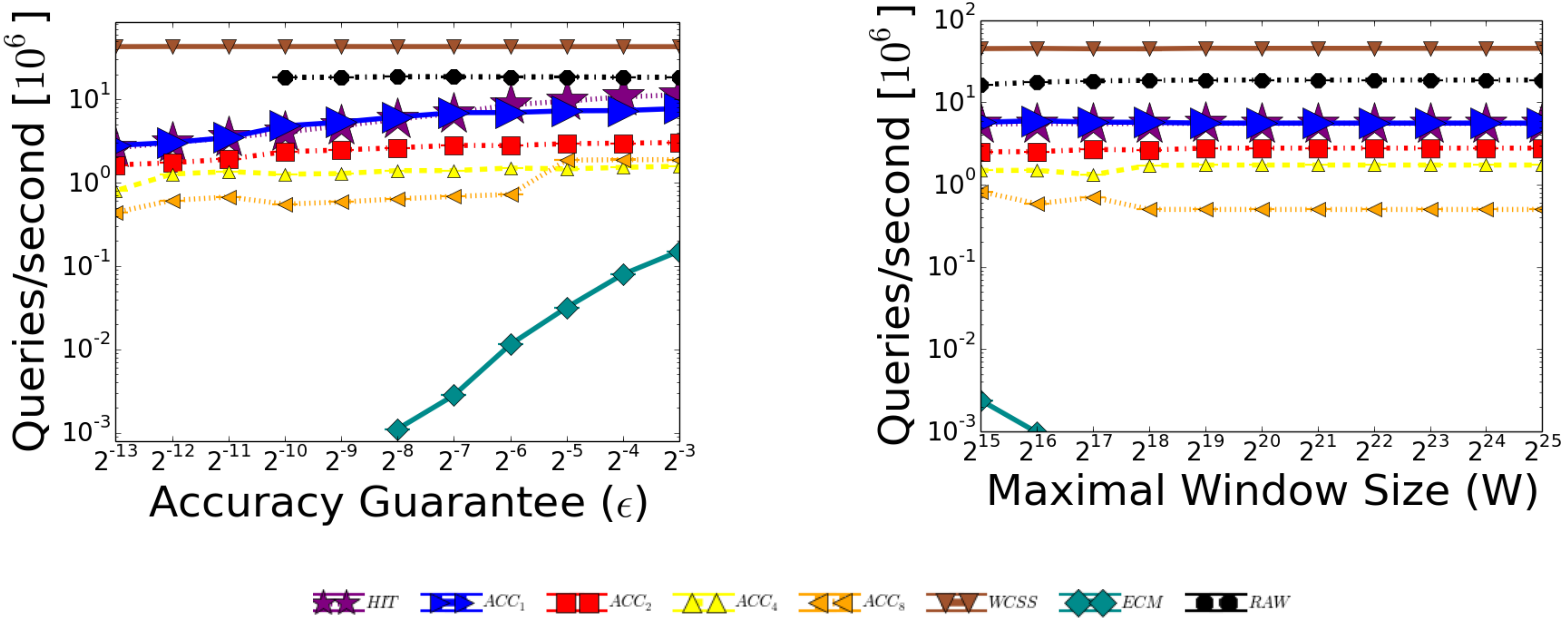
Update Speed comparison



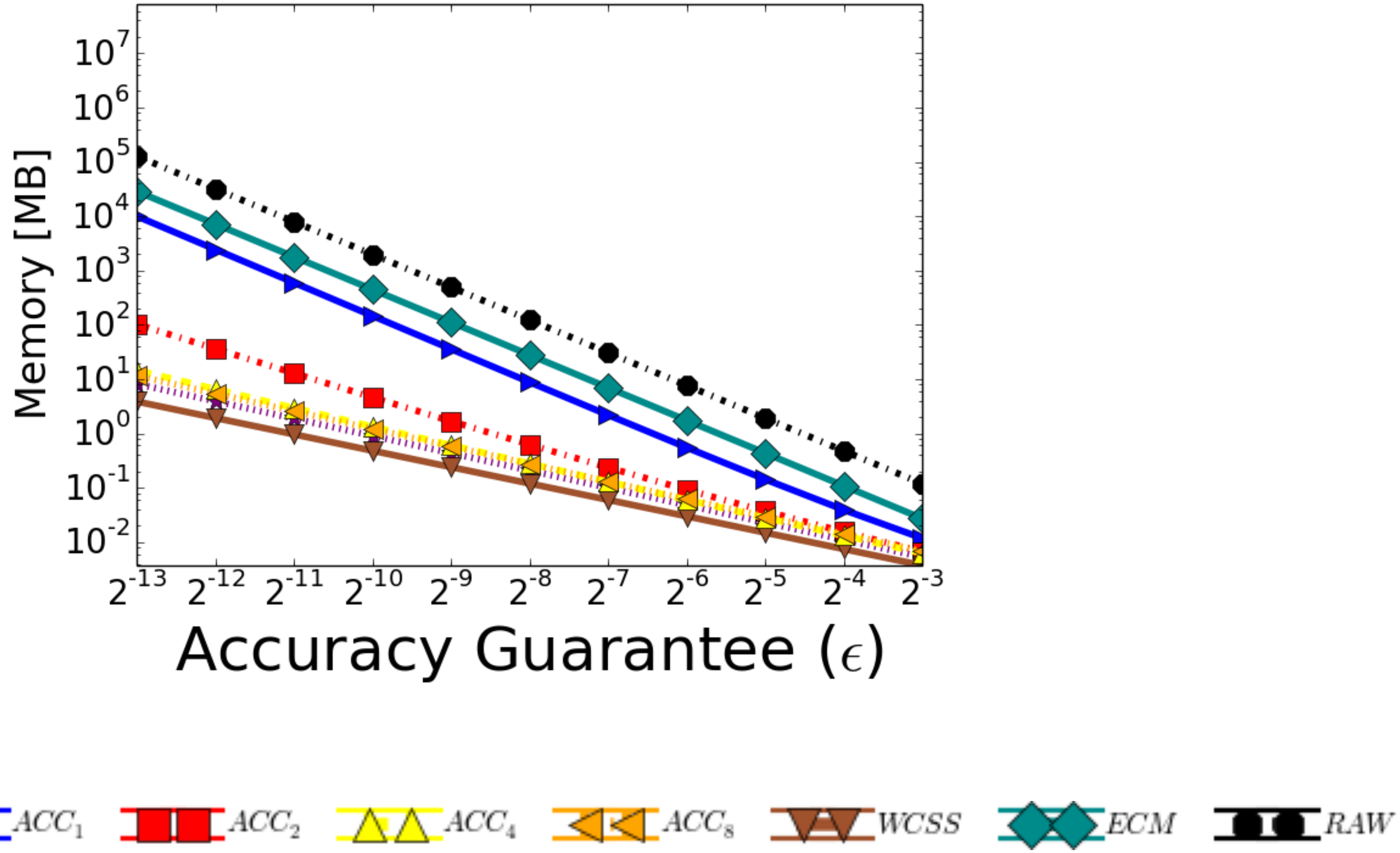
Update Speed comparison



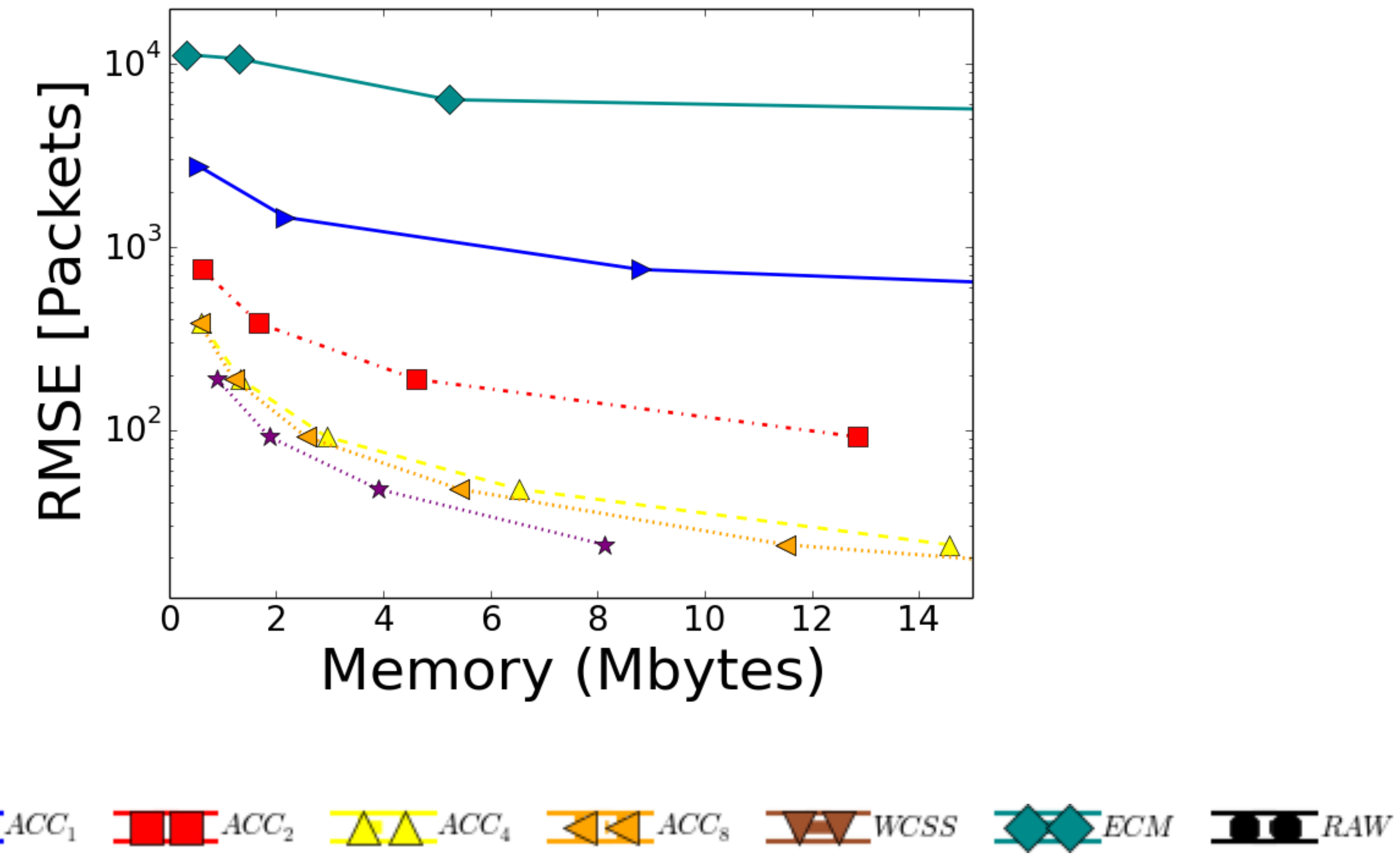
Query Speed comparison



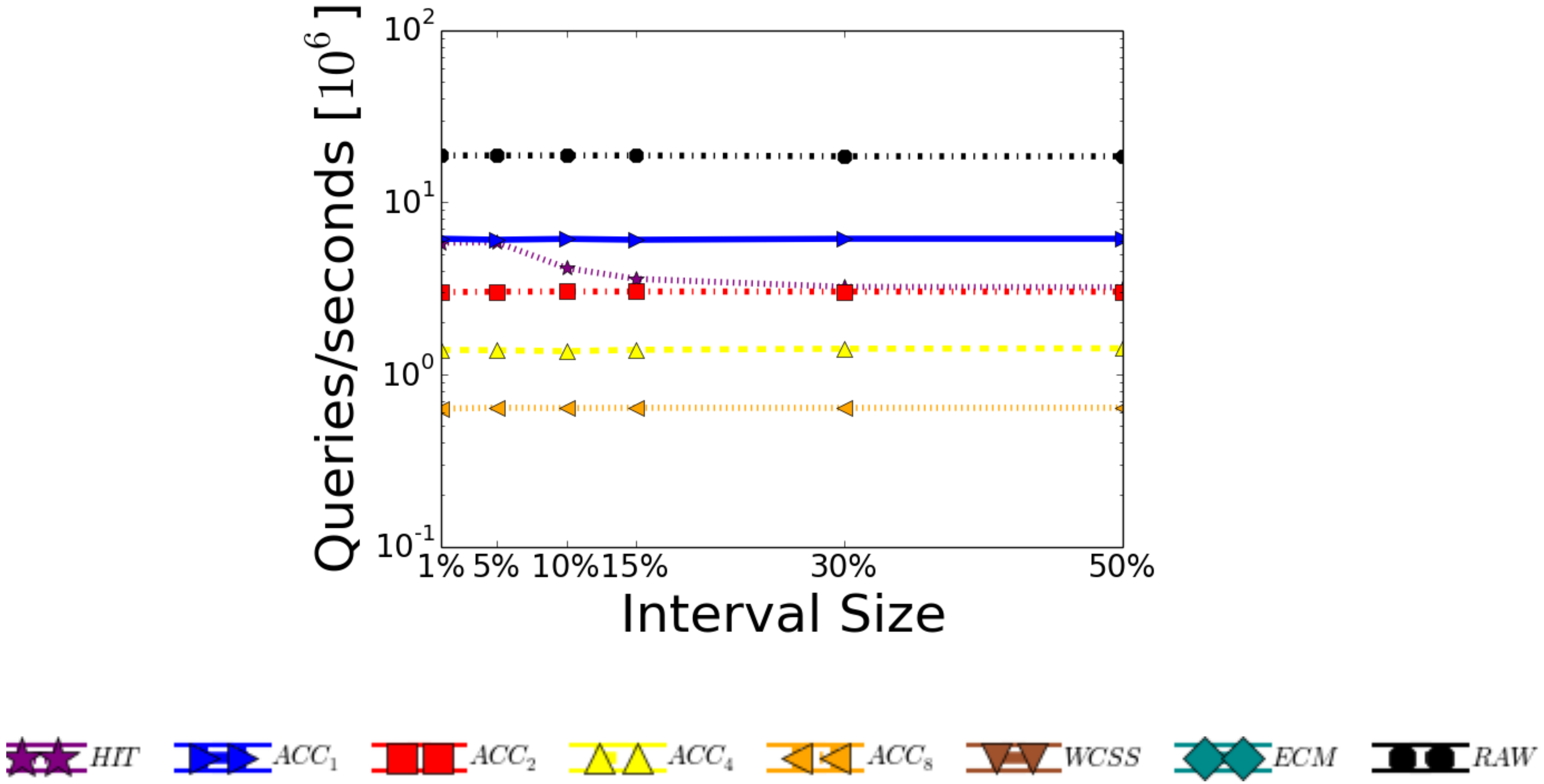
Memory Consumption



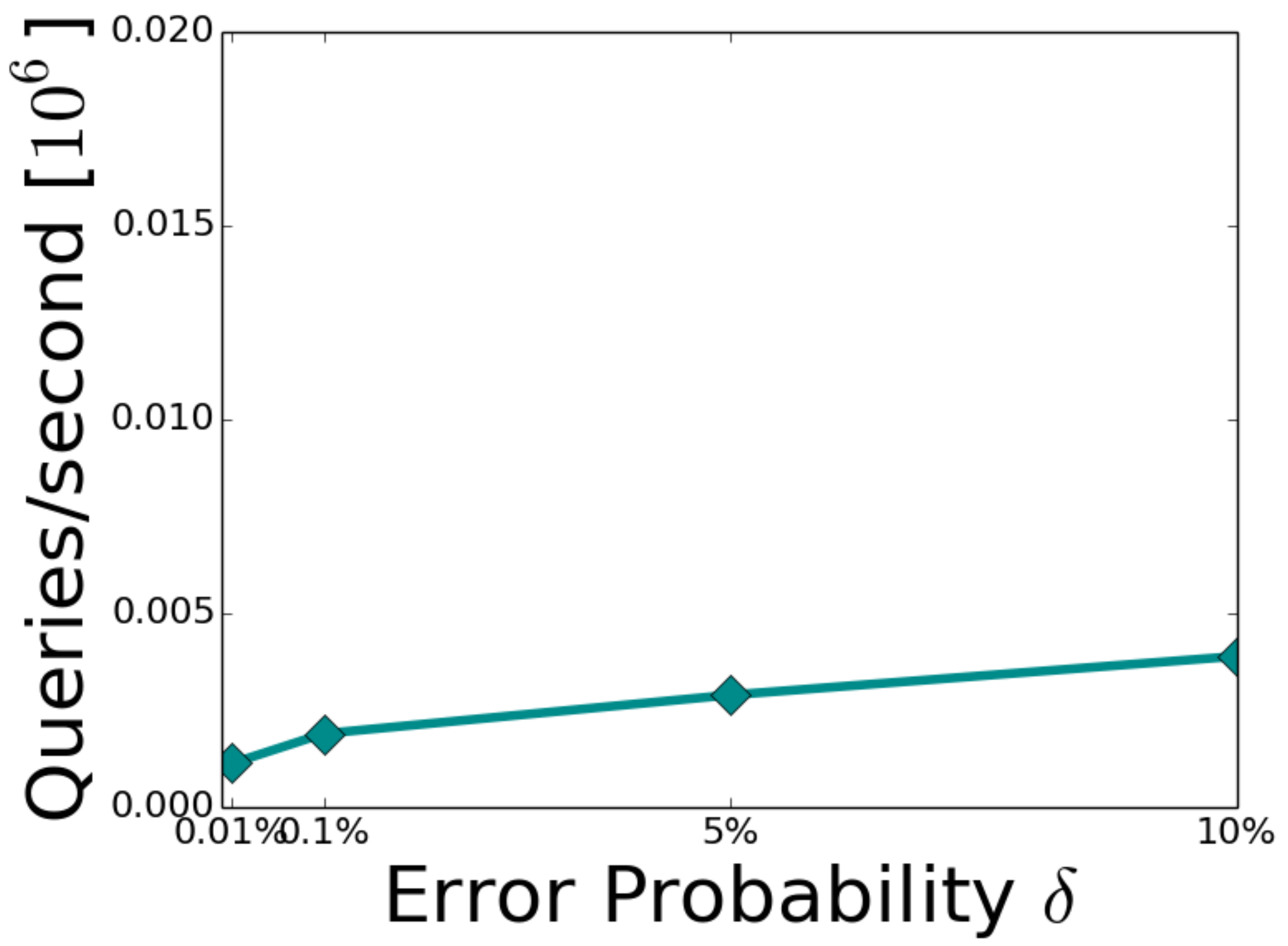
Observed error

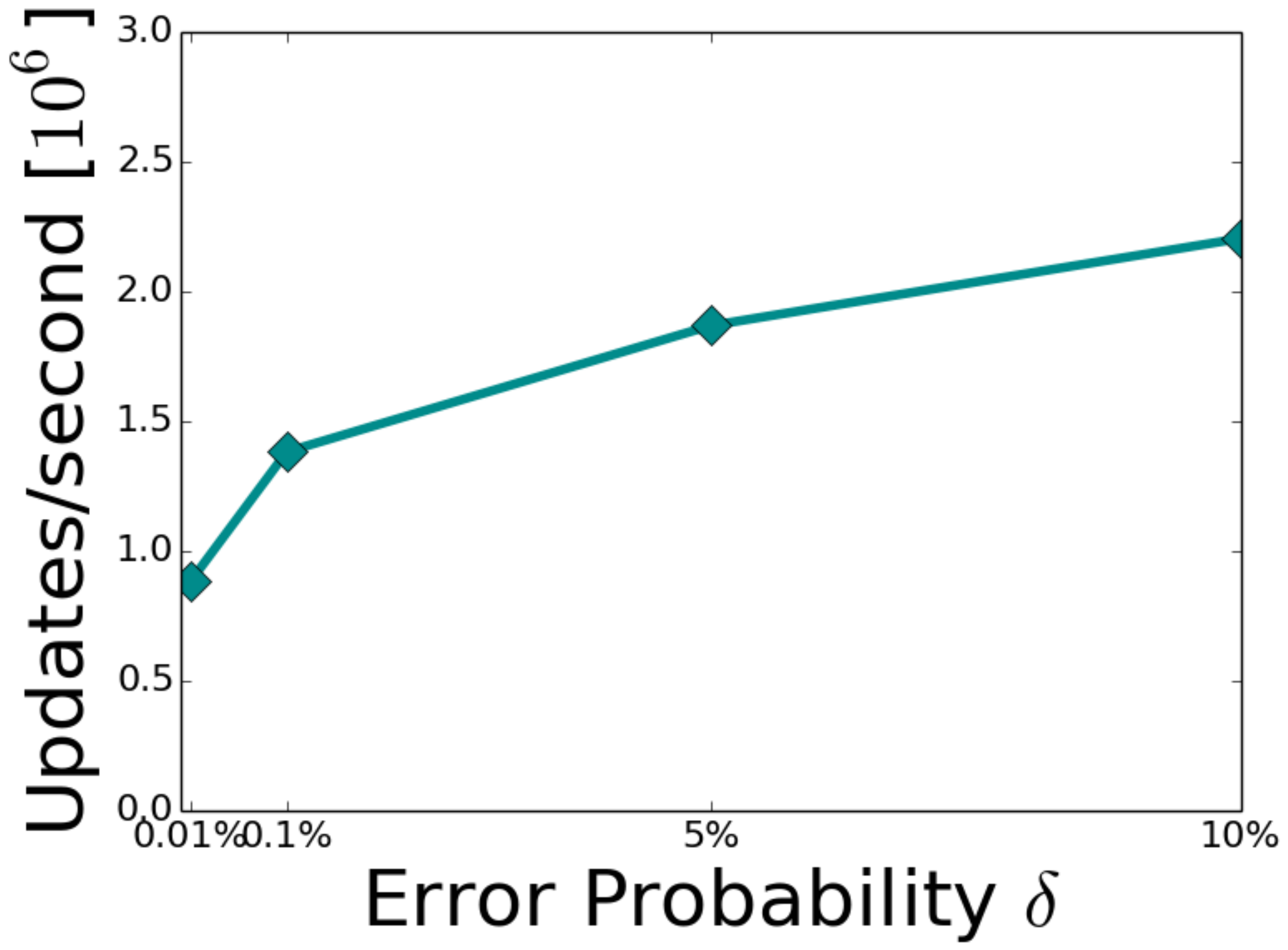


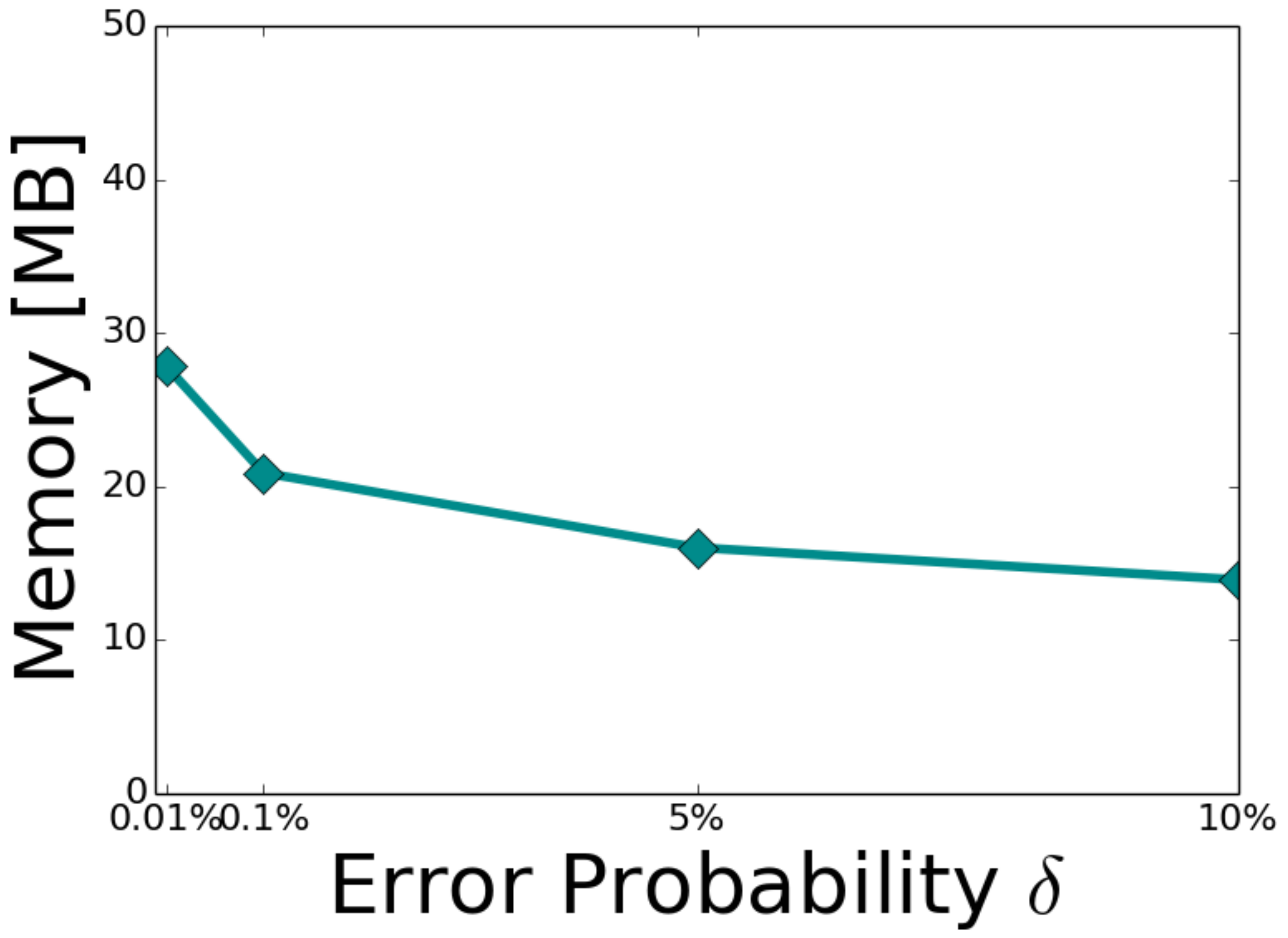
vart interval sizes



ECM space and performance comparison







Thank YOU!