Analysis of Massive Data Sets

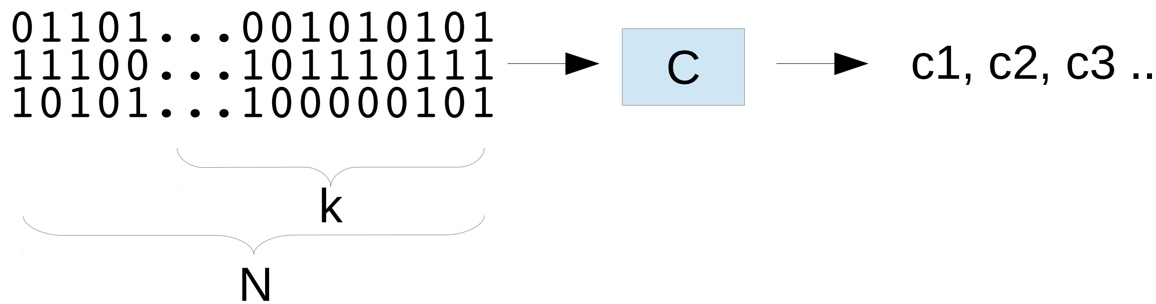
Stream Data Model and Processing (II)

Klemo Vladimir

Faculty of Electrical Engineering and Computing
Consumer Computing Laboratory

Counting 1's

- Problem
  - Given a stream of zeroes (0's) and ones (1's),
  - Count a number of ones in the last *k* bits
    - k <= N



- Obvious solution
  - Store all N bits
    - When new bit arrives, remove oldest bit
  - Performance
    - Query takes O(k) time
    - Space-inefficient since N can be large (or many streams)

Counting 1's

- Challenge
  - Cannot afford to store all of the (N) bits
- Problem
  - Exact count is not possible without all of the bits
- Proof
  - Representation uses fewer than N bits
  - There must be two different bit strings w and x that have the same representation

Counting 1's

- Proof
  - Since w is different from x:
    - They must differ in at least one bit
  - Let the last k-1 bits of w and x agree
    - Example

1	1	0	1
0	1	0	1

Real values

0	0	1
0	0	1

Representation

  - Query: how many 1's in the last k bits?
    - Answer: same for both w and x
    - Algorithm can se only representations
  - Thus, must use at least N bits

Counting 1's

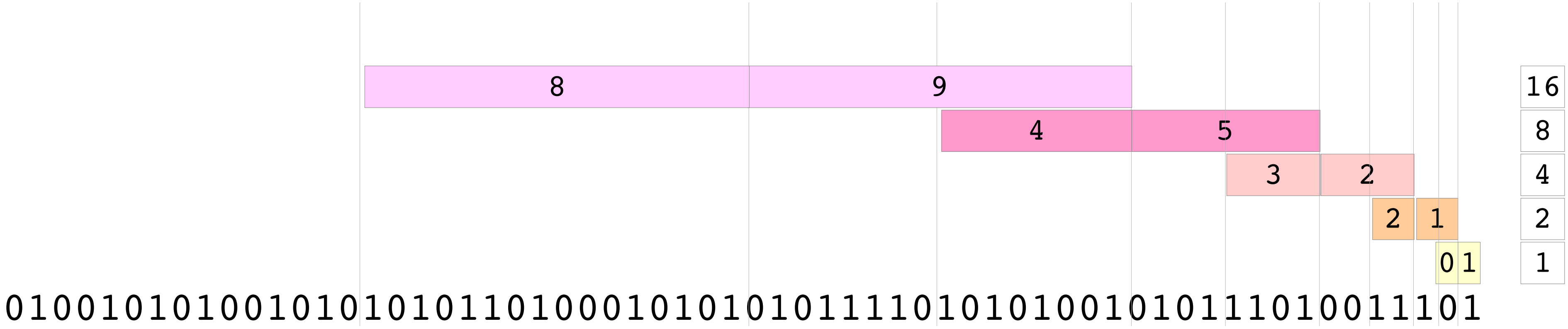
- Challenge
  - Count number of 1's with much less space/time requirements
  - Cannot afford to store all of the bits
- Exact solution
  - Exact count is not possible without all of the bits
- Approximation method
  - 1. Exponentially increasing blocks
  - 2. Datar-Gionis-Indyk-Motwani (DGIM) algorithm

Counting 1's

- Exponentially increasing blocks
  - Summarize exponentially increasing blocks of the stream, looking backward
    - 1, 2, 4, 8, 16, …
  - Summary is the number of ones in the block
  - Keep never more than two blocks of any size

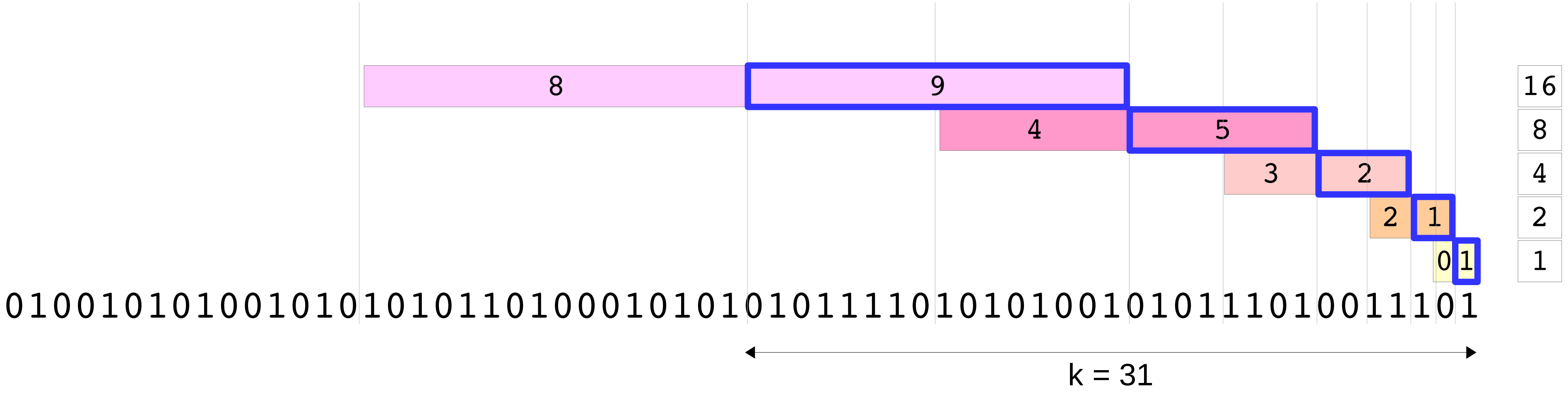
Counting 1's

- Exponentially increasing blocks
  - two blocks of any size



Counting 1's

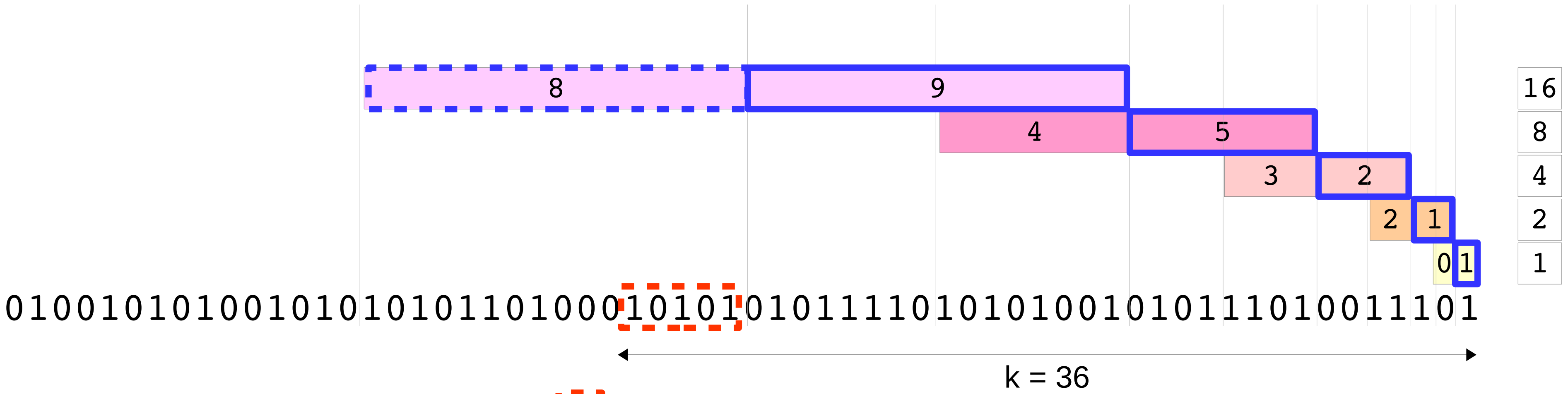
- Exponentially increasing blocks
  - Query: count 1's in the last **k=31** bits



Answer: 1 + 1 + 2 + 5 + 9 = 18

Counting 1's

- Exponentially increasing blocks
  - Query: count 1's in the last **k=36** bits



Answer: 1 + 1 + 2 + 5 + 9 + 3 = 21

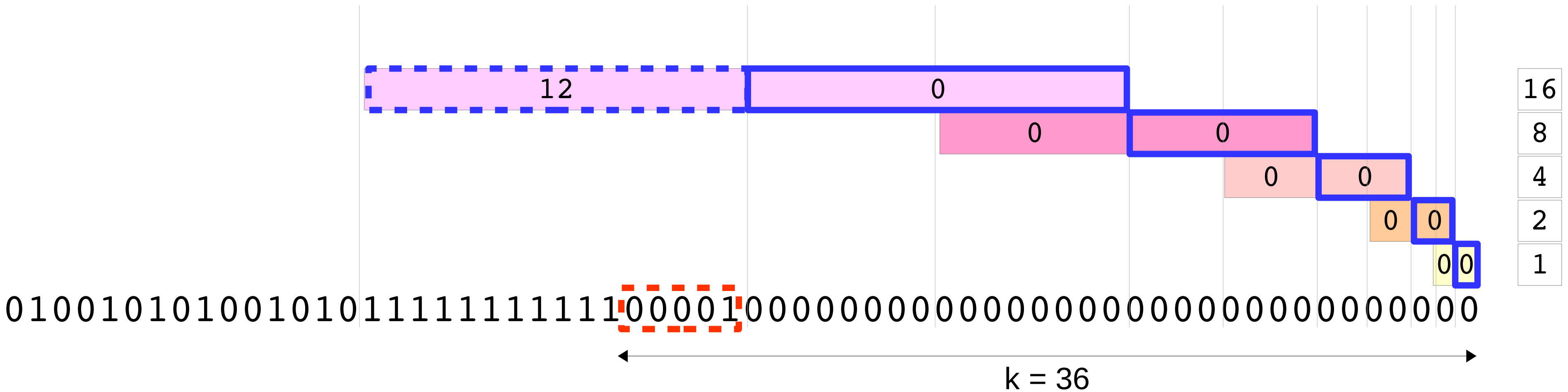
Estimate: 1 + 1 + 2 + 5 + 9 + x = ? x = 8/2=4 (half estimate), or
x = 8/16*5=2.5 (prop. guess)

Counting 1's

- Exponentially increasing blocks
  - Store $2 * \log_2 N * \log_2 N$ bits = $O(\log^2 N)$
    - First $\log_2 N$ for blocks (2 of each size)
    - Second $\log_2 N$ for the counter
  - Error
    - No greater than the number of zeroes in the last bits that are not covered by complete block
    - Depends on the distribution of 1's

Counting 1's

- Exponentially increasing blocks
  - Error
    - All the 1's are in the last bits that are not covered by complete block



Estimate: 0 + x = ? x = 12/2 = 6 (Right answer: **1** ones) Error: **600%**

Counting 1's

- Datar-Gionis-Indyk-Motwani (DGIM) algorithm
  - Similar to the previous algorithm
  - Avoids the problem of uneven distribution of 1's
  - Instead of fixed-length blocks, keep blocks with specific number of 1's
    - Exponential block sizes (size = num of 1's)
  - Stores $O(\log^2 N)$ bits per stream
  - Approximate answer
    - Error max 50% true count

Counting 1's

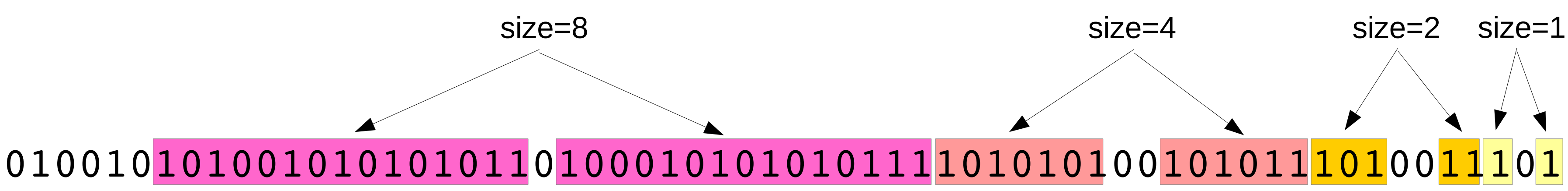
- DGIM algorithm
  - Timestamps
    - Each bit has timestamp (position): 1, 2, 3, ...
    - Timestamp is modulo N (window size)
      - need **$\log_2 N$ bits**
  - Buckes
    - Segment of the window defined by
      - Timestamp of it's end ($\log_2 N$ bits)
      - (Power of 2) number of 1's in the bucket
        - X = 1, 2, 4, 8, … → $2^0, 2^1, 2^2, 2^3, \ldots \rightarrow 2^y$
        - $\max(y) = \log_2 N$
        - Bucket memory representation → **$\log_2 \log_2 N$ bits**
      - Total: **O(logN) bits** for the bucket

Counting 1's

- DGIM algorithm
  - Total storage requirements
    - Window length → N
    - Largest bucket size → 2^y
      - $y < \log_2 N$
    - Bucket sizes: 1 … $\log_2 N$
  - Number of bits needed
    - O(logN) buckets * O(logN) bits per bucket
      - **$O(\log^2 N)$**

Counting 1's

- DGIM algorithm
  - Basic idea
    - Size of the bucket is power of 2 number of 1's in the bucket



Counting 1's

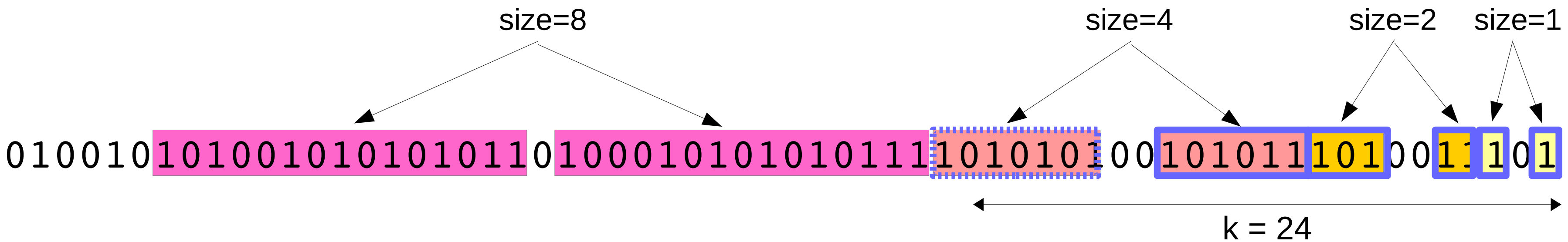
- DGIM algorithm
  - Rules
    - Right end of a bucket is always a position with 1
    - Every position with a 1 is in some bucket (no more than one)
    - One or two buckets with the same size
    - Sizes are power of 2
    - Buckets are sorted by size and do not overlap

Counting 1's

- DGIM algorithm
  - Query answering
    - How many 1's there are in the last k bits ($k \leq N$)
  - Procedure (O(logN))
    - Sum sizes of all buckets but the last
      - Last bucket → bucket with the earliest timestamp that includes at least some of the k most recent bits
    - Add half the size of the last bucket
  - Optimization (O(1))
    - Two counters:
      - TOTAL → sum of all buckets
      - LAST → size of the last bucket
    - Estimate → TOTAL + LAST/2

Counting 1's

- DGIM algorithm
  - Query answering
    - k=24



Answer: 1 + 1 + 2 + 2 + 4 + 4/2 = 12 (True count: 14)

Counting 1's

- DGIM algorithm
  - Maintaining buckets
    - When a new bit comes in
      - delete the oldest bucket if its end-time is prior to N time units before the current time (update LAST/TOTAL)
      - If the new bit is 0 → no other changes
      - If the new is is 1
        - Create new bucket with size 1 (for the new bit) and current timestamp (TOTAL++)
        - Count number of buckets with size 1
          - If there are 3 buckets of size 1 → merge oldest two into single bucket of size 2
          - If there are 3 buckets of size 2 → merge oldest two into single bucket of size 4
          - … (update LAST)

Counting 1's

- DGIM algorithm
  - Maintaining buckets

010010 101001010101011 0 100010101010111 1010101 00 101011 101 00 11 1 0 1

New bit: **1**

10010 101001010101011 0 100010101010111 1010101 00 101011 101 00 11 1 0 1 1

10010 101001010101011 0 100010101010111 1010101 00 101011 101 00 11 1011 1

10010 101001010101011 0 100010101010111 1010101 00 101011 1010011 1011 1

10010 101001010101011 0 100010101010111 101010100101011 1010011 1011 1

10010 1010010101010110100010101010111 101010100101011 1010011 1011 1

Counting 1's

- DGIM algorithm
  - Maintaining buckets

New bit: 1

0010 1010010101010110100010101010111 101010100101011 1010011 101 1 1

New bit: 0

010 1010010101010110100010101010111 101010100101011 1010011 101 1 1 0

New bit: 1

10 1010010101010110100010101010111 101010100101011 1010011 101 1 1 0 1

10 1010010101010110100010101010111 101010100101011 1010011 101 11 0 1

Counting 1's

- DGIM algorithm - Error analysis
  - Last bucket (2^y) approximation is $x/2 \rightarrow 2^{y-1}$
  - Existing bucket sizes → $1 \ldots 2^{y-1}$
    - True count → c
  - a) **Estimate is greater than c**
    - $\min(c) = 1 + 2 + 4 + \ldots + 2^{y-1} = 2^y - 1$
    - Add 1 from the single 1 in the last bucket
    - $\min(c) = 2^y - 1 + 1 = 2^y$
    - Estimate is at least **50%** of c

...0100101010010101010110010011111010100

min(c)

  - b) **Estimate is less than c**
    - Largest bucket size → $x = 2^y$
    - All of the 1's are in the range
    - Estimate misses 2^{y-1} bits
    - $\min(c) = 2^y - 1$
    - Estimate is no more than **50%** greater than c

1...100101010010101010110010011111010100

min(c)

Counting 1's

- Generalization
  - Allow **more than two** buckets of any size
  - More buckets of smaller sizes → stronger bound on the error
  - k = ceil(1/ε), k/2 is an integer
  - Memory requirements: $O(1/\varepsilon * \log^2 N)$
  - Algorithm update:
    - Merge if there are k/2 + 2 buckets of the same size
  - Error
    - Estimate is within factor 1 + ε
      - Simplest case: k=2 → error 50%

Counting 1's

- Literature
  - J. Leskovec, A. Rajaraman, and J. D. Ullman, "**Mining of Massive Datasets**“, 2014, **Chapter 4. Mining Data Streams**
  - Datar, Mayur, et al. "**Maintaining stream statistics over sliding windows.**" SIAM Journal on Computing 31.6 (2002): 1794-1813.