Advanced topics on Algorithms

Luciano Gualà
www.mat.uniroma2.it/~guala/

Advanced Data Structure
Episode I

Data Structures for Big Data

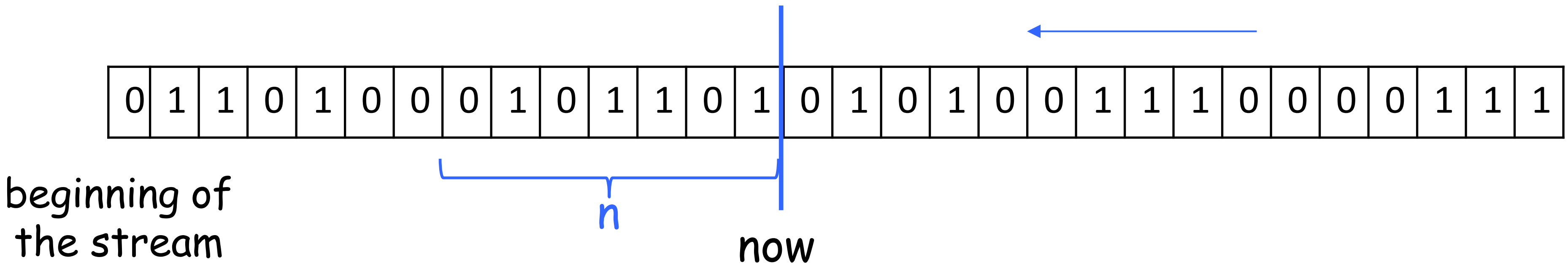
Counting 1s in a window

Datar-Gionis-Indyk-Motwani's (DGIM) algorithm

reference:

Algorithms for Massive Data (Lecture Notes)
Nicola Prezza
https://arxiv.org/abs/2301.00754

The problem



goal: process a stream of bits in order to answer queries of the type:

- how many 1s in the last n bits?

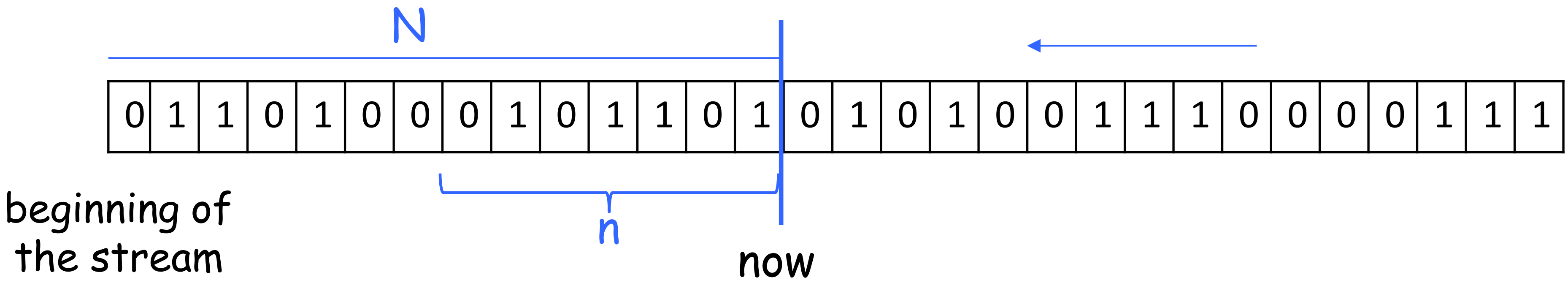
motivation: (approximately) count the events that meet a certain criterion.

Example:
Bank transactions are marked with a flag=1 when exceed a given threshold. Queries can be used to detect if the credit card's owner has changed behavior (hence detect potential frauds)

Example:
Posts/tweets are marked with a flag=1 when they are about a given topic. Queries can be used to detect if the interest on the topic changes.

main challenge: the stream is too large to be entirely stored.

The problem



goal: design a data structure maintaining a sequence of N bits subject to:

- query(n): return the number of 1s in the last n bits;
- update(b): add the next bit $b \in \{0,1\}$ to the sequence

notice: if you want exact answers you need $\Omega(N)$ bits.

DGIM data structure:

- quality: $1+\varepsilon$ approximated answers (for any $\varepsilon>0$)
- size: $O(\varepsilon^{-1} \log^2 N)$ bits
- update time: $O(\log N)$
- query time: $O(\varepsilon^{-1} \log n)$

DGIM data structure:

Let $B = \lceil 1/\varepsilon \rceil$.

Group the bits of the sequence in groups $G_1, \ldots, G_t$ satisfying:

1. each G_i begins and ends with a 1-bit;
2. between adjacent groups G_i G_{i+1} there are only 0-bits;
3. each G_i contains 2^k 1-bits, for some $k \geq 0$;
4. for any $1 \leq i < t$, if G_i contains 2^k 1-bits, then G_{i+1} contains either 2^k or 2^{k-1} 1-bits;
5. for each k except the largest one, the number Z_k of groups containing 2^k 1-bits satisfies $B \leq Z_k \leq B+1$. For the largest k, we only require $Z_k \leq B+1$.

G_1 (8) G_2 (4) G_3 (4) G_4 (2) G_5 (2) G_6 (1)

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

$B=1$

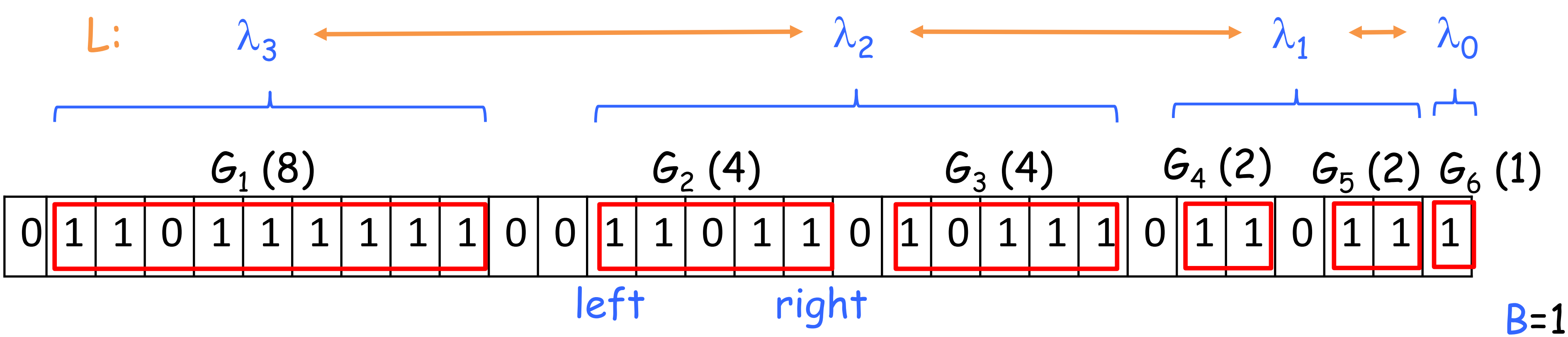
DGIM data structure:

group G_j is a pair of integers (left,right)

all adjacent groups having 2^i 1-bits are maintained by a doubly-linked list λ_i

λ_i stores: head, tail, and size

L: a global doubly-linked list storing all lists λ_i



- storing G_j requires $O(\log N)$ bits
- $|L| = O(\log N)$
- $|\lambda_i| \leq B+1 = O(\varepsilon^{-1})$

overall size of the DS: $O(\varepsilon^{-1} \log^2 N)$ bits

update operation

update(b): add the next bit $b \in \{0,1\}$ to the sequence

If b=0 do nothing. Otherwise (b=1):

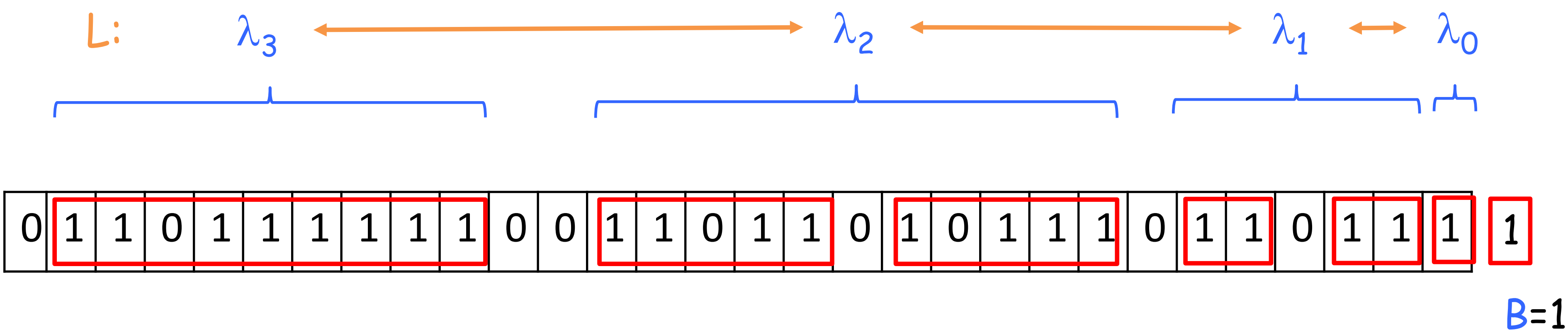
1. Create a new group with the new 1-bit and add it to λ_0;
2. if λ_0 contains B+2 groups, merge the two leftmost groups thus forming a new group of 2 1-bits and add it to λ_1 as a new rightmost group (notice that λ_0 now has B groups);
3. repeat step 2 for λ_i , i=1,2,...;

update operation

update(b): add the next bit $b \in \{0,1\}$ to the sequence

If b=0 do nothing. Otherwise (b=1):

1. Create a new group with the new 1-bit and add it to λ_0;
2. if λ_0 contains B+2 groups, merge the two leftmost groups thus forming a new group of 2 1-bits and add it to λ_1 as a new rightmost group (notice that λ_0 now has B groups);
3. repeat step 2 for λ_i , i=1,2,...;

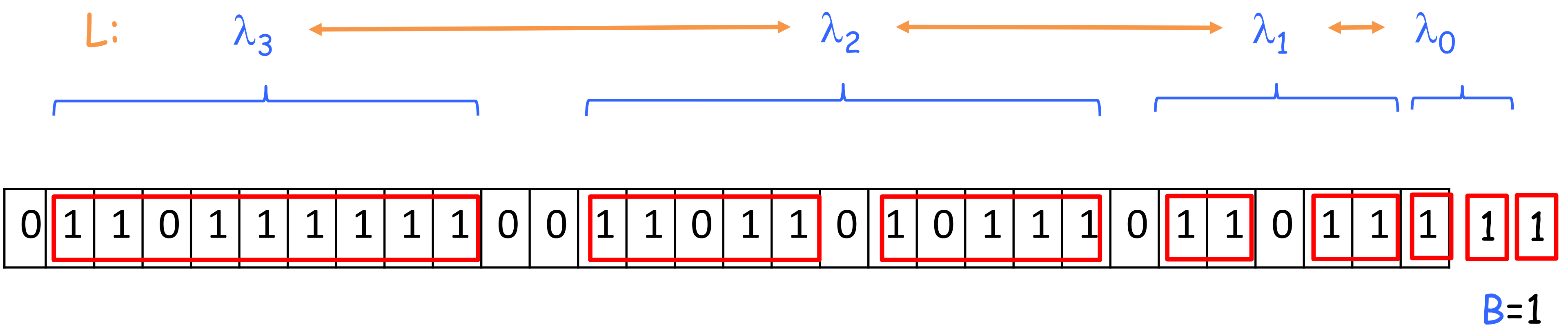


update operation

update(b): add the next bit $b \in \{0,1\}$ to the sequence

If b=0 do nothing. Otherwise (b=1):

1. Create a new group with the new 1-bit and add it to λ_0;
2. if λ_0 contains B+2 groups, merge the two leftmost groups thus forming a new group of 2 1-bits and add it to λ_1 as a new rightmost group (notice that λ_0 now has B groups);
3. repeat step 2 for λ_i , i=1,2,...;

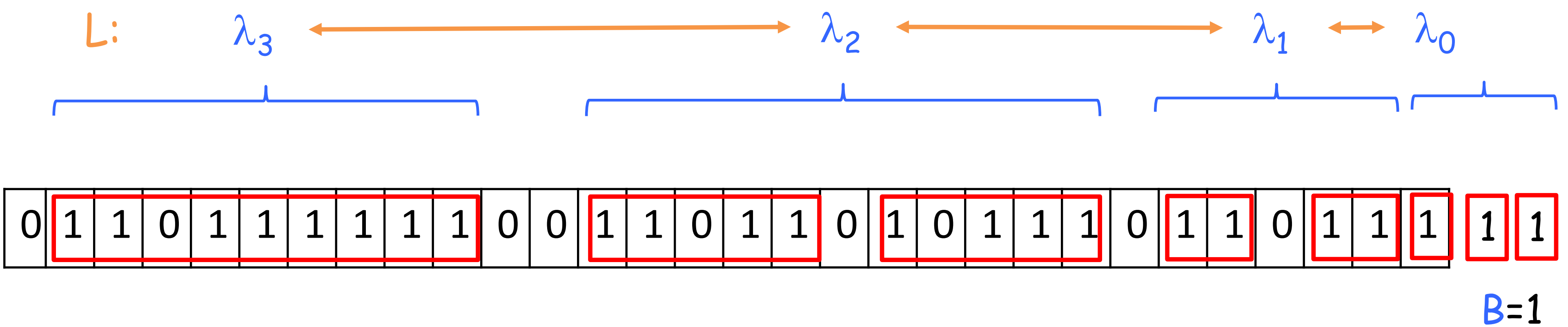


update operation

update(b): add the next bit $b \in \{0,1\}$ to the sequence

If b=0 do nothing. Otherwise (b=1):

1. Create a new group with the new 1-bit and add it to λ_0;
2. if λ_0 contains B+2 groups, merge the two leftmost groups thus forming a new group of 2 1-bits and add it to λ_1 as a new rightmost group (notice that λ_0 now has B groups);
3. repeat step 2 for λ_i , i=1,2,...;

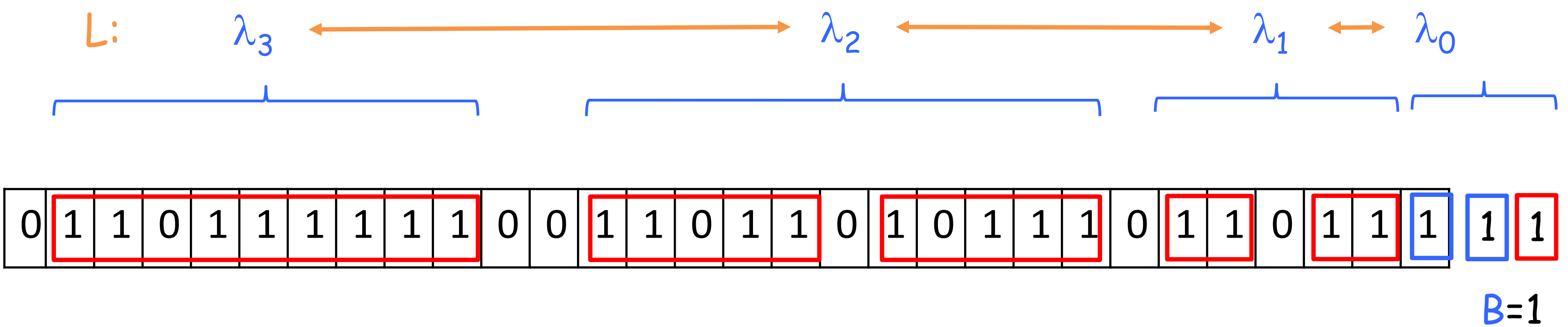


update operation

update(b): add the next bit $b \in \{0,1\}$ to the sequence

If b=0 do nothing. Otherwise (b=1):

1. Create a new group with the new 1-bit and add it to λ_0;
2. if λ_0 contains B+2 groups, merge the two leftmost groups thus forming a new group of 2 1-bits and add it to λ_1 as a new rightmost group (notice that λ_0 now has B groups);
3. repeat step 2 for λ_i , i=1,2,...;

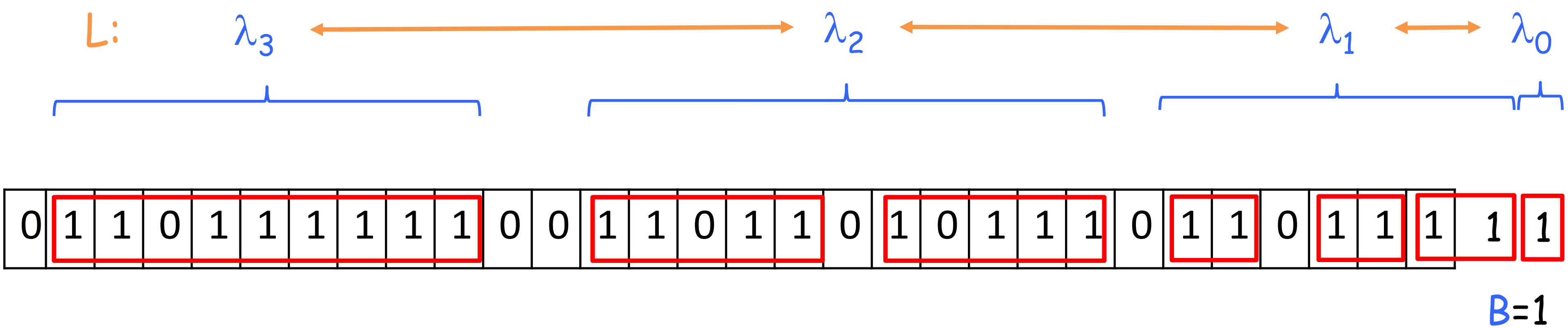


update operation

update(b): add the next bit $b \in \{0,1\}$ to the sequence

If b=0 do nothing. Otherwise (b=1):

1. Create a new group with the new 1-bit and add it to λ_0;
2. if λ_0 contains B+2 groups, merge the two leftmost groups thus forming a new group of 2 1-bits and add it to λ_1 as a new rightmost group (notice that λ_0 now has B groups);
3. repeat step 2 for λ_i , i=1,2,...;

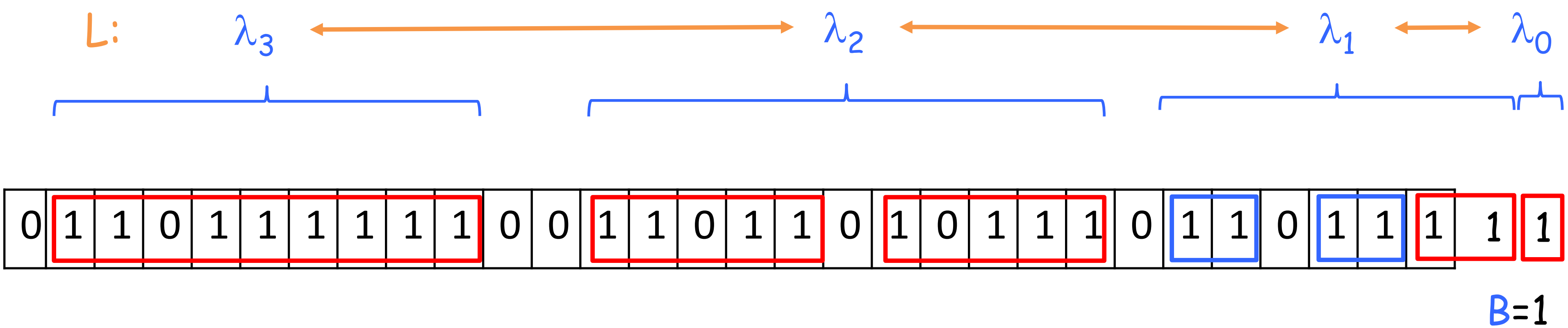


update operation

update(b): add the next bit $b \in \{0,1\}$ to the sequence

If b=0 do nothing. Otherwise (b=1):

1. Create a new group with the new 1-bit and add it to λ_0;
2. if λ_0 contains B+2 groups, merge the two leftmost groups thus forming a new group of 2 1-bits and add it to λ_1 as a new rightmost group (notice that λ_0 now has B groups);
3. repeat step 2 for λ_i , i=1,2,...;

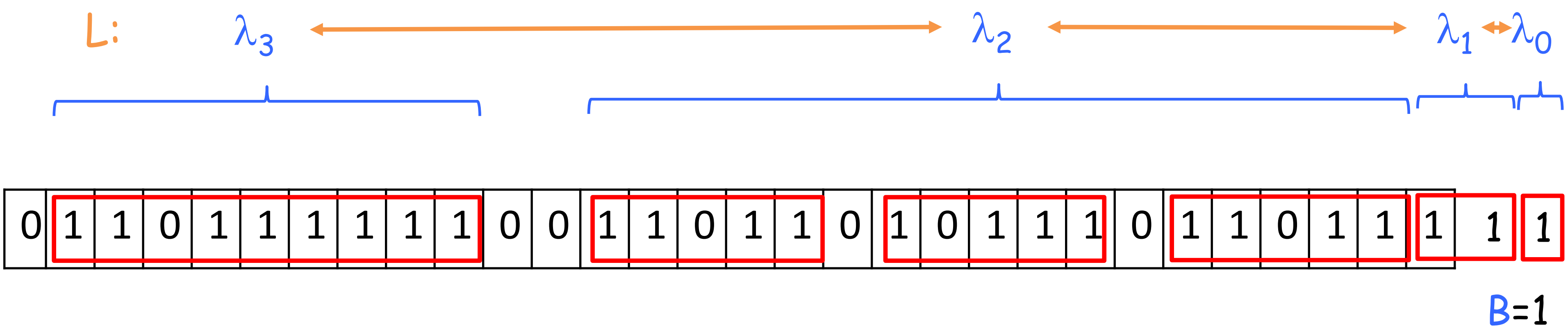


update operation

update(b): add the next bit $b \in \{0,1\}$ to the sequence

If b=0 do nothing. Otherwise (b=1):

1. Create a new group with the new 1-bit and add it to λ_0;
2. if λ_0 contains B+2 groups, merge the two leftmost groups thus forming a new group of 2 1-bits and add it to λ_1 as a new rightmost group (notice that λ_0 now has B groups);
3. repeat step 2 for λ_i , i=1,2,...;

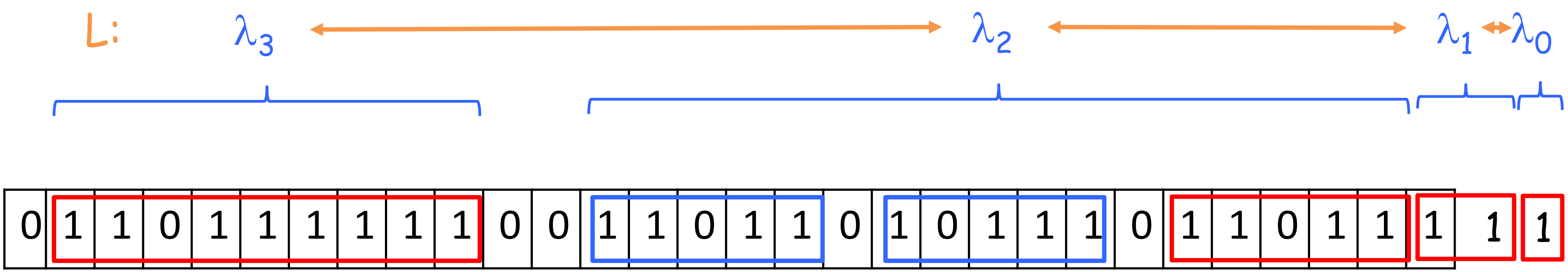


update operation

update(b): add the next bit $b \in \{0,1\}$ to the sequence

If $b=0$ do nothing. Otherwise ($b=1$):

1. Create a new group with the new 1-bit and add it to λ_0;
2. if λ_0 contains B+2 groups, merge the two leftmost groups thus forming a new group of 2 1-bits and add it to λ_1 as a new rightmost group (notice that λ_0 now has B groups);
3. repeat step 2 for λ_i , i=1,2,...;



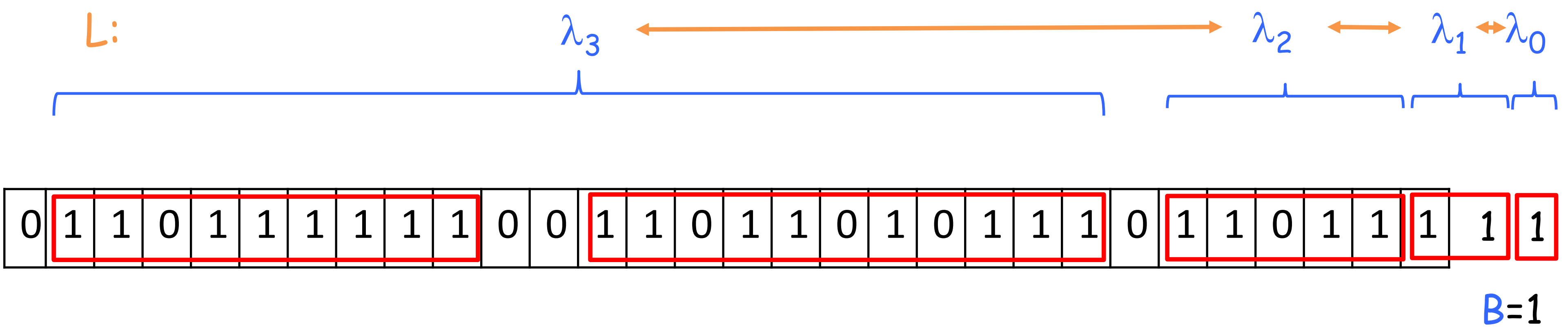
B=1

update operation

update(b): add the next bit $b \in \{0,1\}$ to the sequence

If b=0 do nothing. Otherwise (b=1):

1. Create a new group with the new 1-bit and add it to λ_0;
2. if λ_0 contains B+2 groups, merge the two leftmost groups thus forming a new group of 2 1-bits and add it to λ_1 as a new rightmost group (notice that λ_0 now has B groups);
3. repeat step 2 for λ_i , i=1,2,...;



update operation

update(b): add the next bit $b \in \{0,1\}$ to the sequence

If $b=0$ do nothing. Otherwise ($b=1$):

1. Create a new group with the new 1-bit and add it to λ_0;
2. if λ_0 contains B+2 groups, merge the two leftmost groups thus forming a new group of 2 1-bits and add it to λ_1 as a new rightmost group (notice that λ_0 now has B groups);
3. repeat step 2 for λ_i , i=1,2,...;

update time:

- creating/merging/moving a group takes O(1) time
- number of iterations O(|L|)

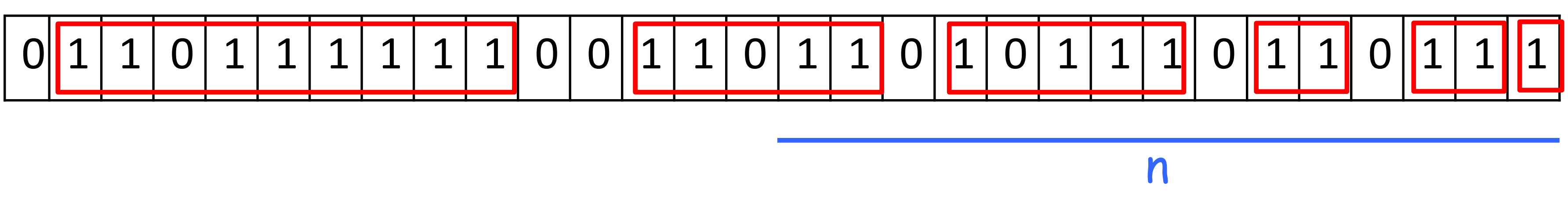
overall update time: O(log N)

query operation

query(n): return the number of 1s in the last n bits

- find all groups intersecting the last n bits
- return the number of 1-bits they contain

query time:
- navigating all groups from the streaming's head
- $O(\varepsilon^{-1} \log n)$ time



query operation: approximation

Let k be the integer s.t. the leftmost intersecting group has 2^k 1-bits

Y: right answer
X: returned answer

notice: if $k=0$ then $X=Y$ (so assume $k>0$)

$$X \le Y + 2^k - 1$$

$$Y \ge B\,2^{k-1} + B\,2^{k-2} + \ldots + B\,2^1 + B\,2^0 = B(2^k - 1)$$

$$\Rightarrow$$

$$X/Y \le (Y + 2^k - 1)/Y = 1 + (2^k - 1)/Y$$

$$\le 1 + 1/B \le 1 + \varepsilon$$

2^k 1-bits

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

n

Finding frequent items in a stream

An application of Sampling

references:

- G.S. Manku, R. Motwani:
Approximate Frequency Counts over Data Streams. VLDB (2002)
https://www.vldb.org/conf/2002/S10P03.pdf
- C. Demestrescu, I. Finocchi
Algorithms for Data Streams
http://www.dei.unipd.it/~geppo/PrAvAlg/DOCS/DFchapter08.pdf

The problem:

given a stream of elements, find the elements whose frequency is above a given threshold

Application domains:

- Data Base world
- Data Mining
- Network Monitoring
- ...

Data Base: iceberg queries

Choosing a good city for a trip...

SELECT City; COUNT(*)

FROM Irish_Pubs

GROUP BY City   HAVING COUNT(*) ≥ T

Data Mining: discovering association rules

I: set of items (products)

D: set of transactions

- a transaction: $T \subseteq I$

an association rule is an implication of the form

$X \Rightarrow Y$ (if you buy X then you also buy Y)

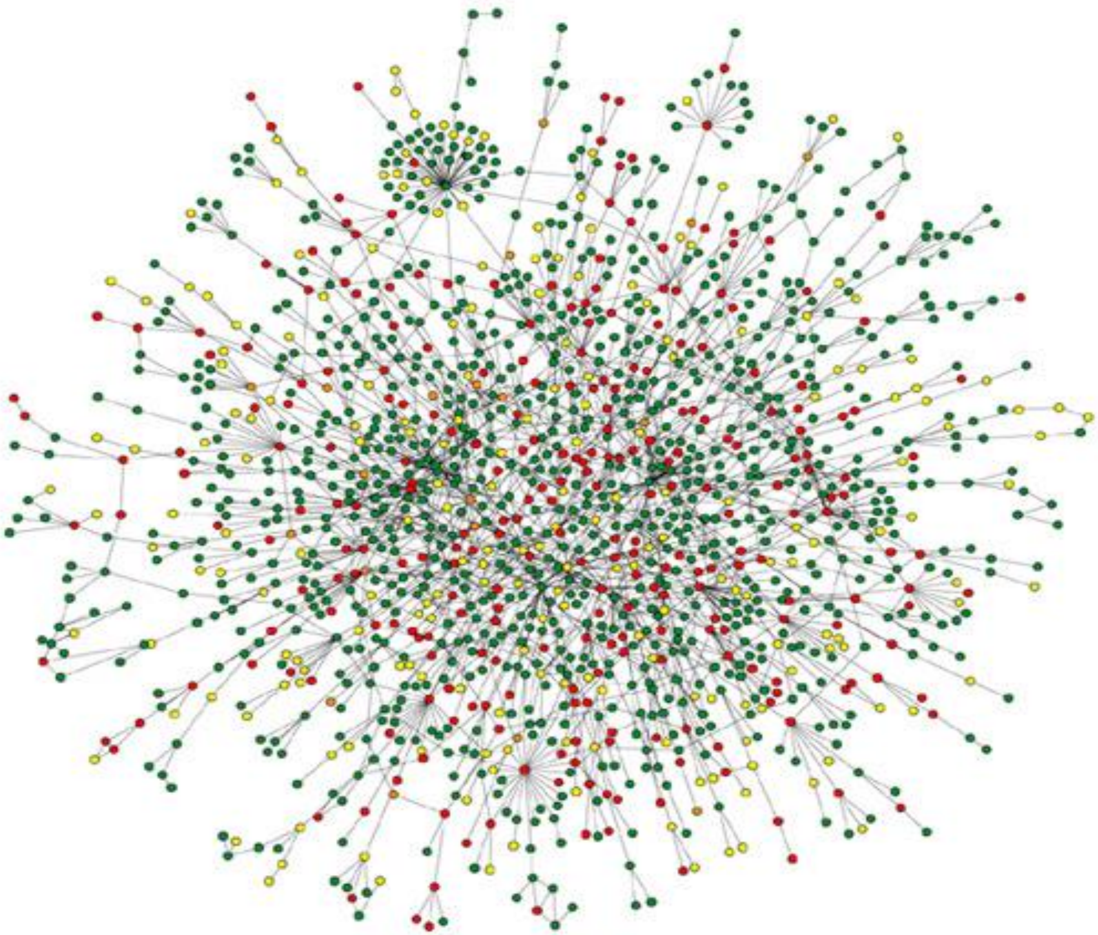
$X \Rightarrow Y$ holds in D with confidence c if c% of the transactions of D that contain X also contain Y

$X \Rightarrow Y$ has support s in D if s% of the transactions of D contain $X \cup Y$

goal: find association rules for D whose confidence and support are above certain thresholds

it reduces to the problem of finding frequent itemsets

Network Monitoring: Measurement and monitoring of network traffic



a flow: sequence of packets with the same source+destination addresses

goal: identifying large flows, i.e. flows sending more than a given threshold (> s% of the link capacity)

The problem

Given two parameters $0<\varepsilon<\varphi<1$, and a stream of n elements $x_1, x_2,\ldots,x_n$, find:

- all items whose frequency is at least φn (no false negative).
- no item with frequency smaller than $(\varphi-\varepsilon)n$.

Sticky sampling algorithm (Manku & Motwani, 2002)

- randomized
- meet the two goals with probability $1-\delta$
- maintain a sample of expected size of $2\varepsilon^{-1} \log (\varphi^{-1} \delta^{-1})$

\# of elements in sample

$0<\delta<1$: user-defined error parameter

notice: space is independent of the stream length n

The algorithm

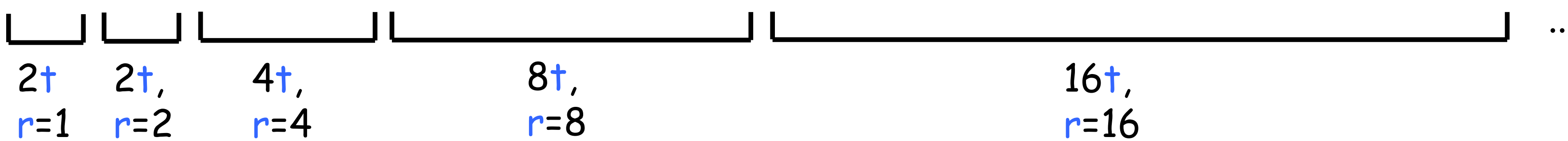
$t = \varepsilon^{-1} \log (\varphi^{-1} \delta^{-1})$

maintain a sample S : set of pairs $(x, f_e(x))$

- $f_e(x)$: estimation of the real frequency $f(x)$ of the element x

to handle potentially unbounded stream, computation proceeds in windows:

- each window has a size and a sampling rate r;
- next window is double the size and the rate of the previous one;
- at the beginning S is empty and r=1.



in a give window, when the next stream element x comes:

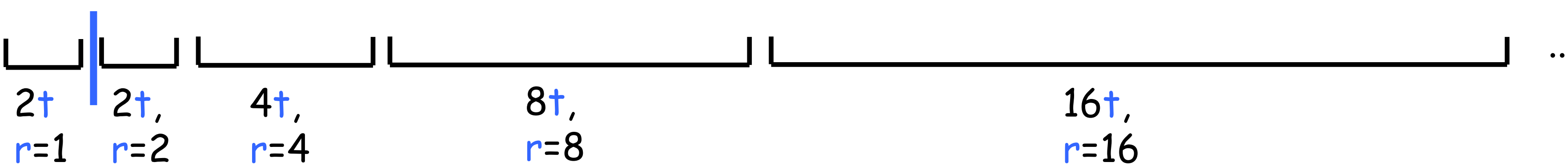
- if $x \in S$ then increase $f_e(x)$ by 1;
- otherwise, insert $(x,1)$ in S with probability $1/r$.

The algorithm

$t = \varepsilon^{-1} \log(\varphi^{-1} \delta^{-1})$

whenever the sampling rate changes from r to $2r$ an adjusting step is performed:

goal: transform the state of S to the one it would have been in if the new rate $2r$ had been used from the beginning

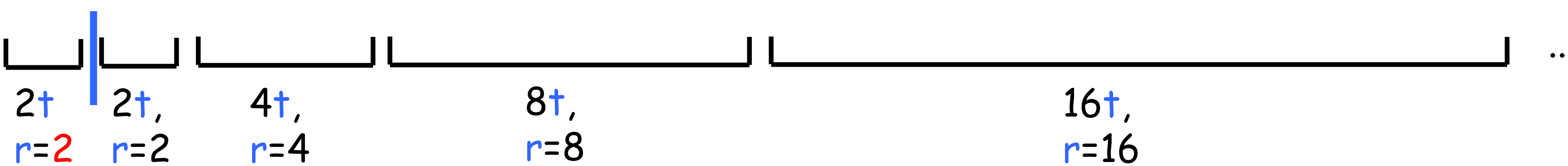


The algorithm

$t = \varepsilon^{-1} \log (\varphi^{-1} \delta^{-1})$

whenever the sampling rate changes from r to $2r$ an adjusting step is performed:

goal: transform the state of S to the one it would have been in if the new rate $2r$ had been used from the beginning

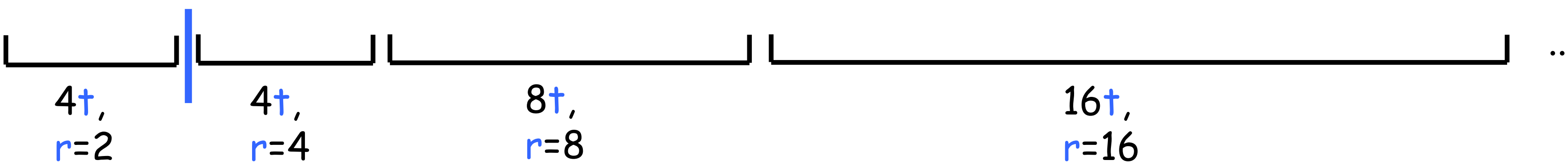


The algorithm

$t = \varepsilon^{-1} \log(\varphi^{-1} \delta^{-1})$

whenever the sampling rate changes from r to $2r$ an adjusting step is performed:

goal: transform the state of S to the one it would have been in if the new rate $2r$ had been used from the beginning

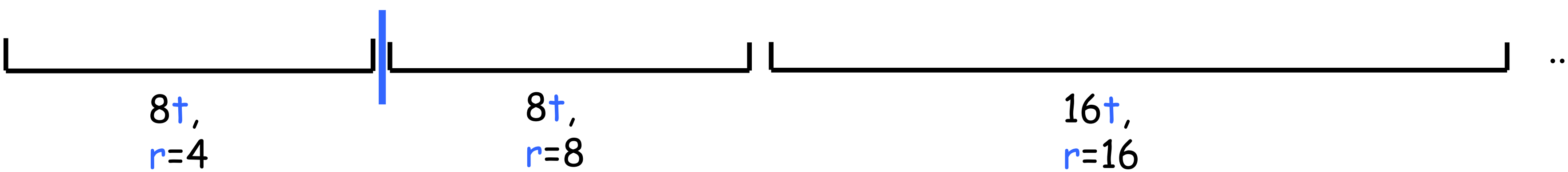


The algorithm

$t = \varepsilon^{-1} \log (\varphi^{-1} \delta^{-1})$

whenever the sampling rate changes from r to $2r$ an adjusting step is performed:

goal: transform the state of S to the one it would have been in if the new rate $2r$ had been used from the beginning

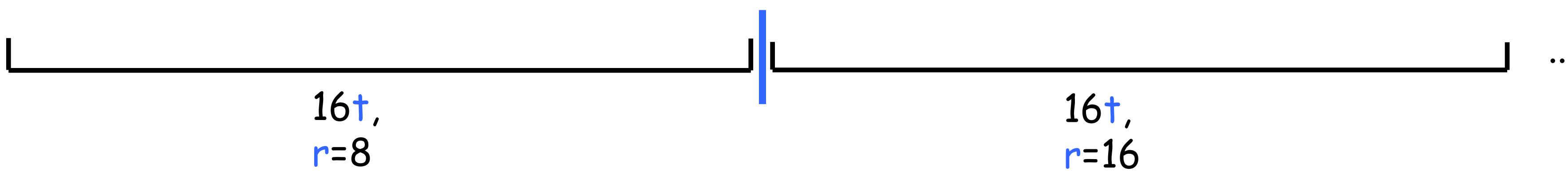


The algorithm

$t = \varepsilon^{-1} \log (\varphi^{-1} \delta^{-1})$

whenever the sampling rate changes from r to $2r$ an adjusting step is performed:

goal: transform the state of S to the one it would have been in if the new rate $2r$ had been used from the beginning

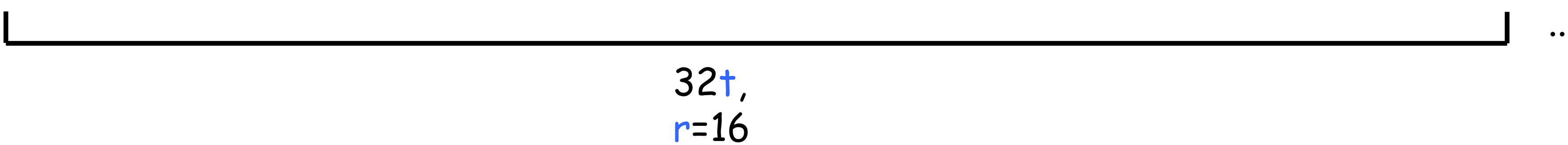


The algorithm

$t = \varepsilon^{-1} \log(\varphi^{-1}\delta^{-1})$

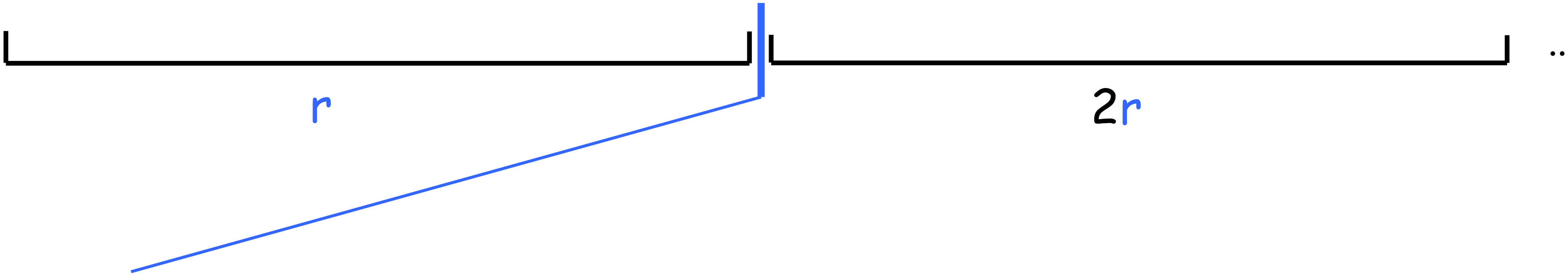
whenever the sampling rate changes from r to $2r$ an adjusting step is performed:

goal: transform the state of S to the one it would have been in if the new rate $2r$ had been used from the beginning



query(n): return all items in S with estimated frequency at least $(\varphi-\varepsilon)n$.

The algorithm: the adjusting step



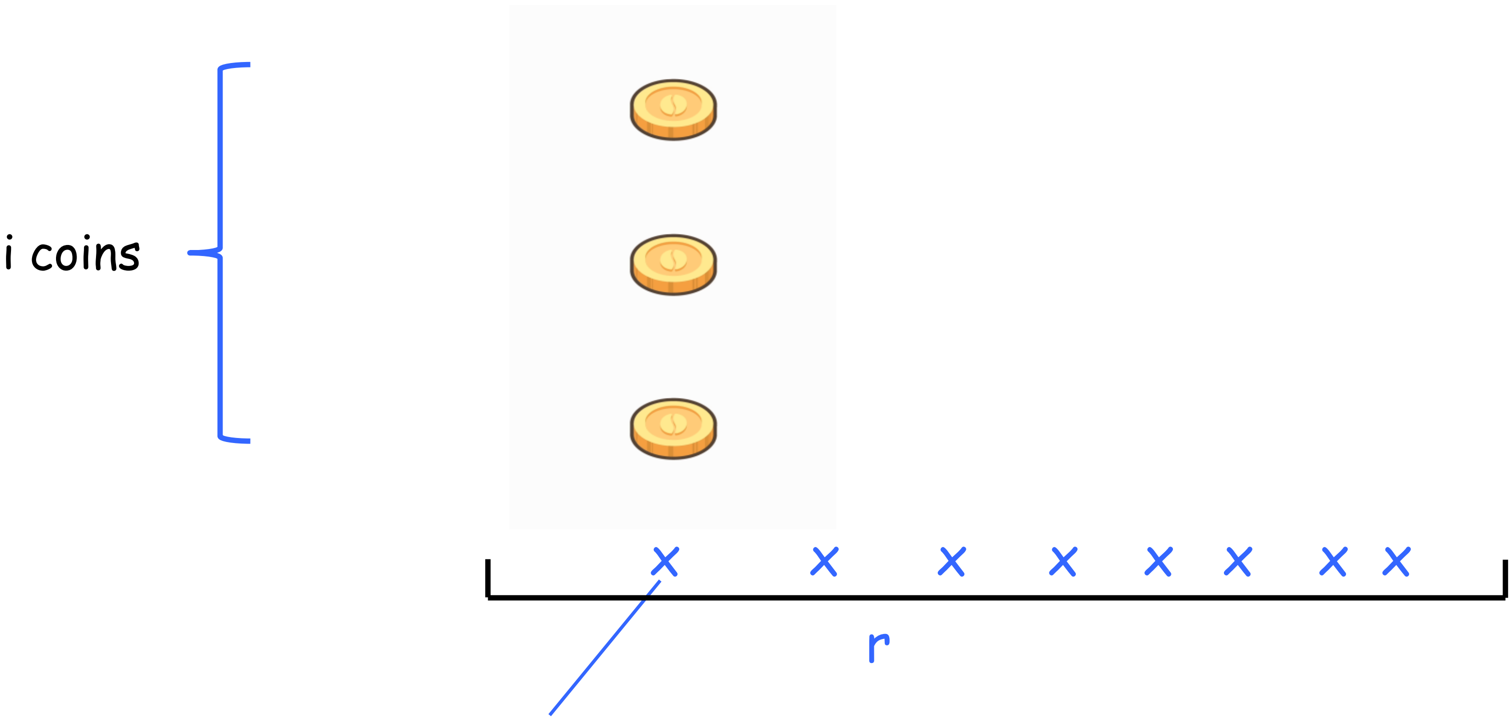
for each element $x \in S$:

- flip a fair coin
- if (Tail) then
  - repeatedly flip a coin with success probability of $1/(2r)$ until you get a success;
  - let k be the number of coin flips performed;
  - decrease $f_e(x)$ by k
  - if $f_e(x) \leq 0$ then remove x from S.

The adjusting step: the analysis

obs: r is always a power of 2 ⟹ assume $r=2^i$

focus on a given element x and consider all its occurrences in the window

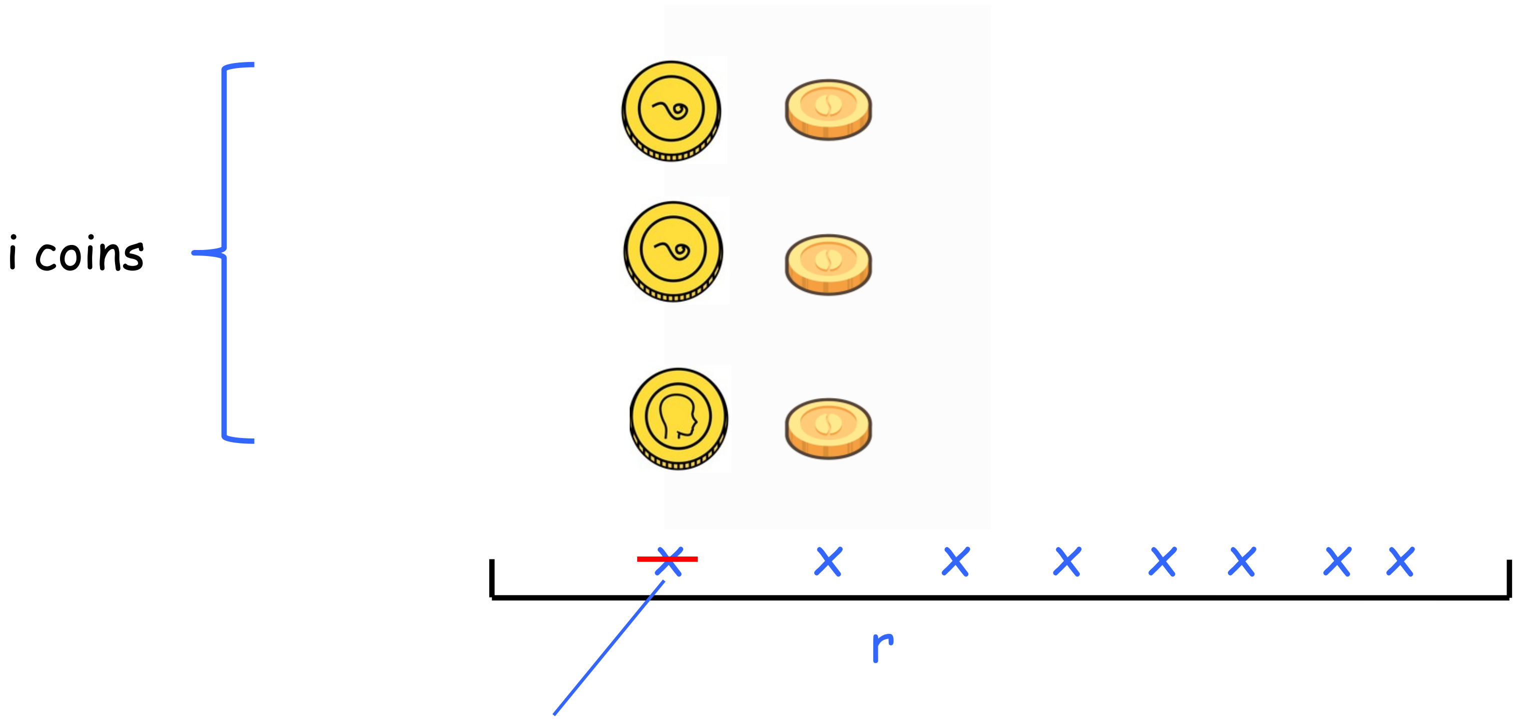


- first occurrence
- put in S with probability $1/r$

The adjusting step: the analysis

obs: r is always a power of 2 ⟹ assume $r=2^i$

focus on a given element x and consider all its occurrences in the window



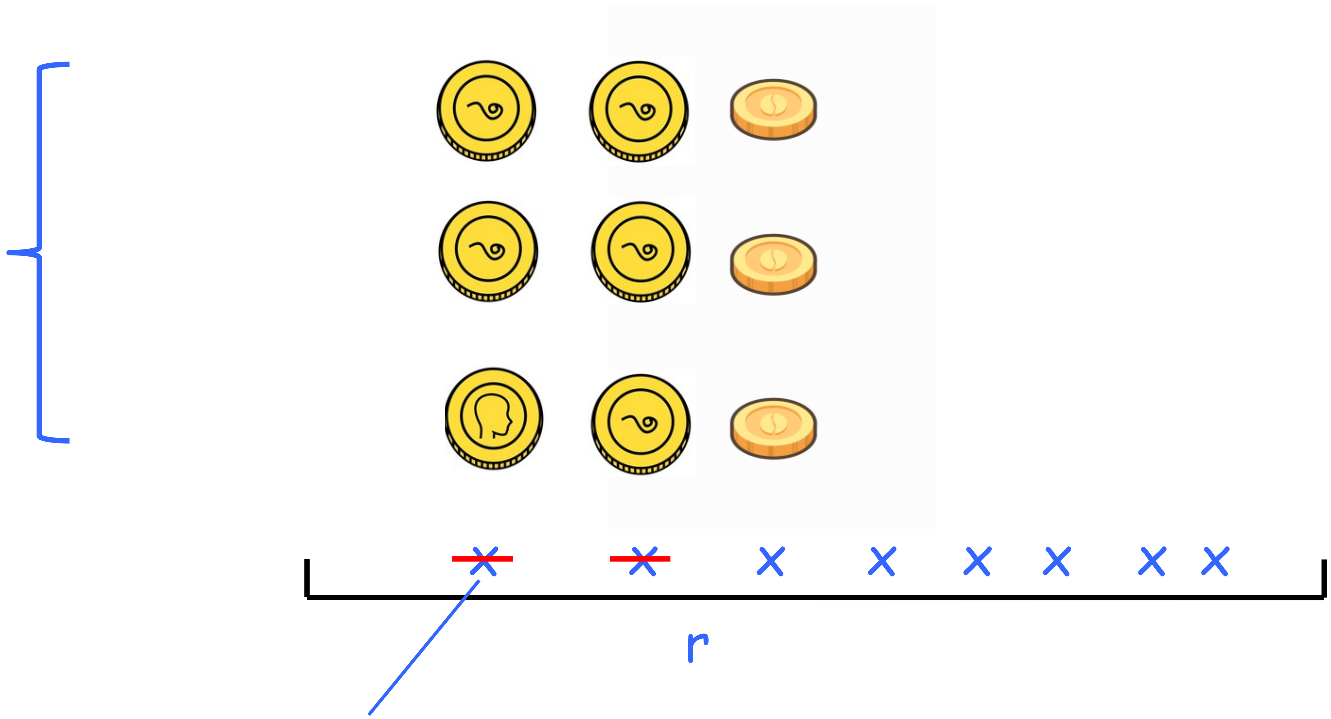
- first occurrence
- put in S with probability $1/r$

The adjusting step: the analysis

obs: r is always a power of 2 → assume $r=2^i$

focus on a given element x and consider all its occurrences in the window

i coins

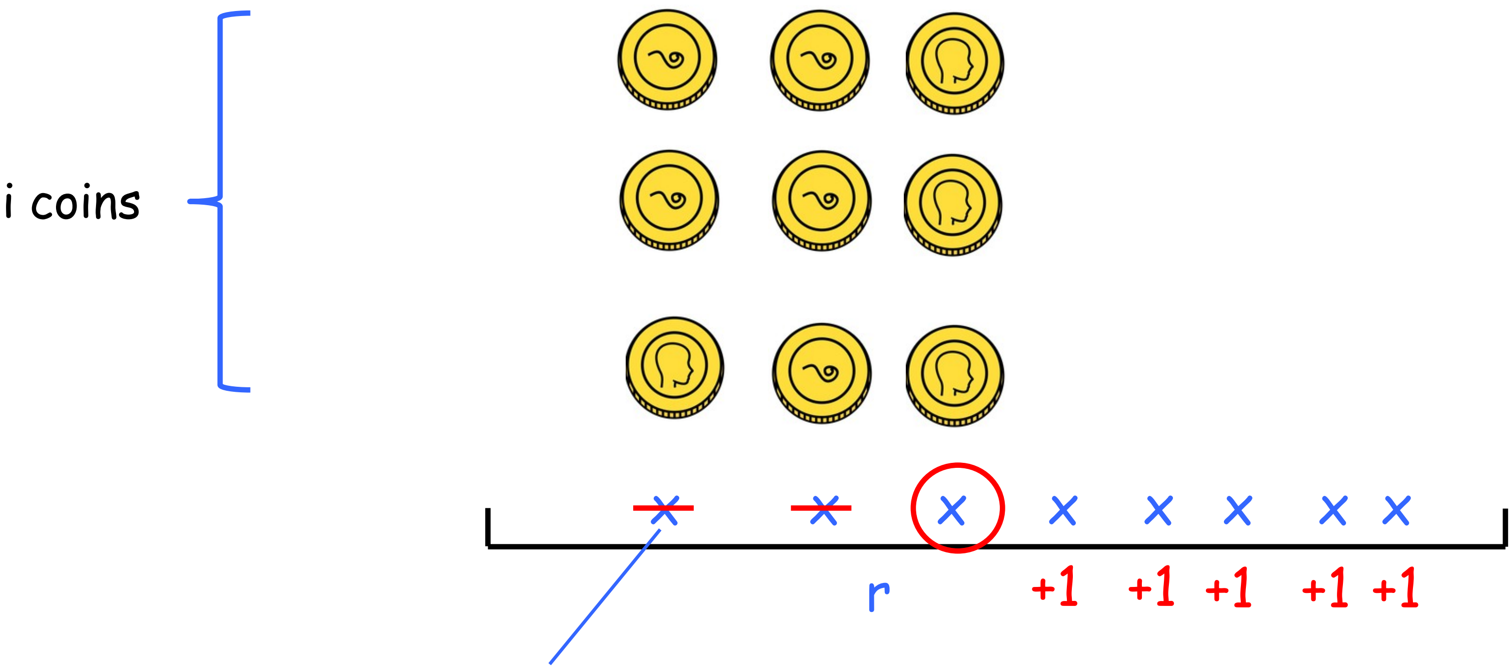


- first occurrence
- put in S with probability $1/r$

The adjusting step: the analysis

obs: r is always a power of 2 ⇒ assume $r=2^i$

focus on a given element x and consider all its occurrences in the window



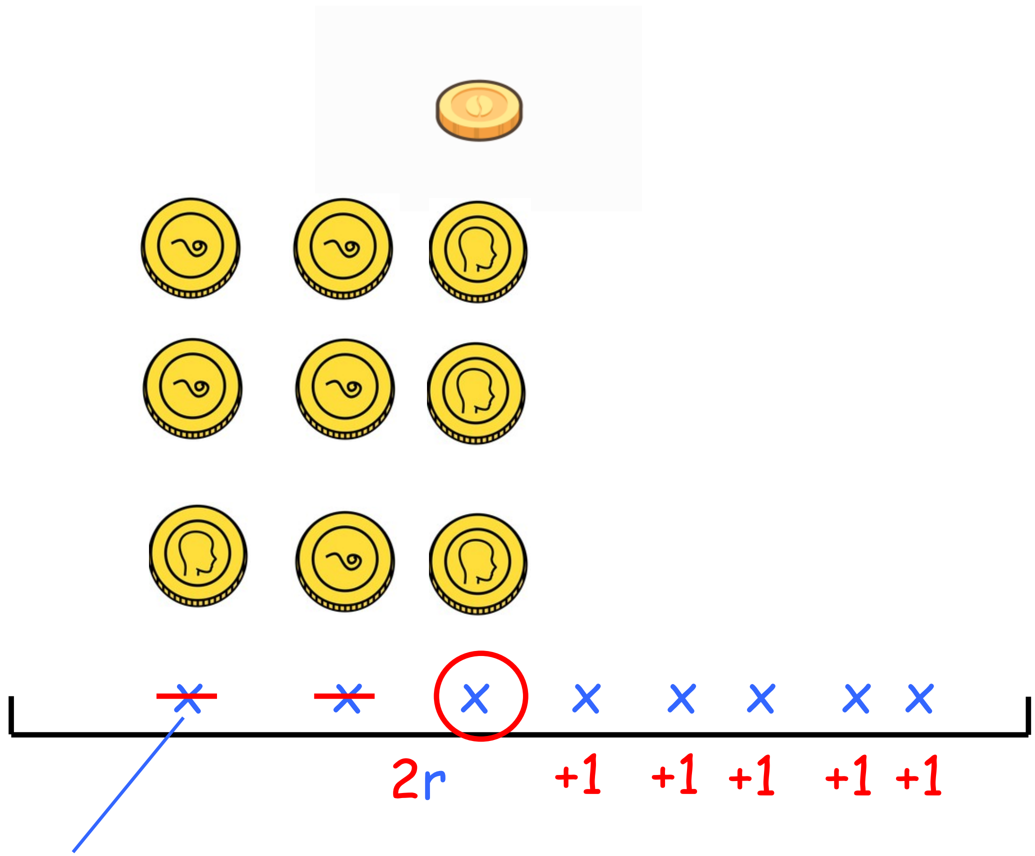
- first occurrence
- put in S with probability $1/r$

The adjusting step: the analysis

obs: r is always a power of 2 ⟹ assume $r=2^i$

focus on a given element x and consider all its occurrences in the window

$i+1$ coins

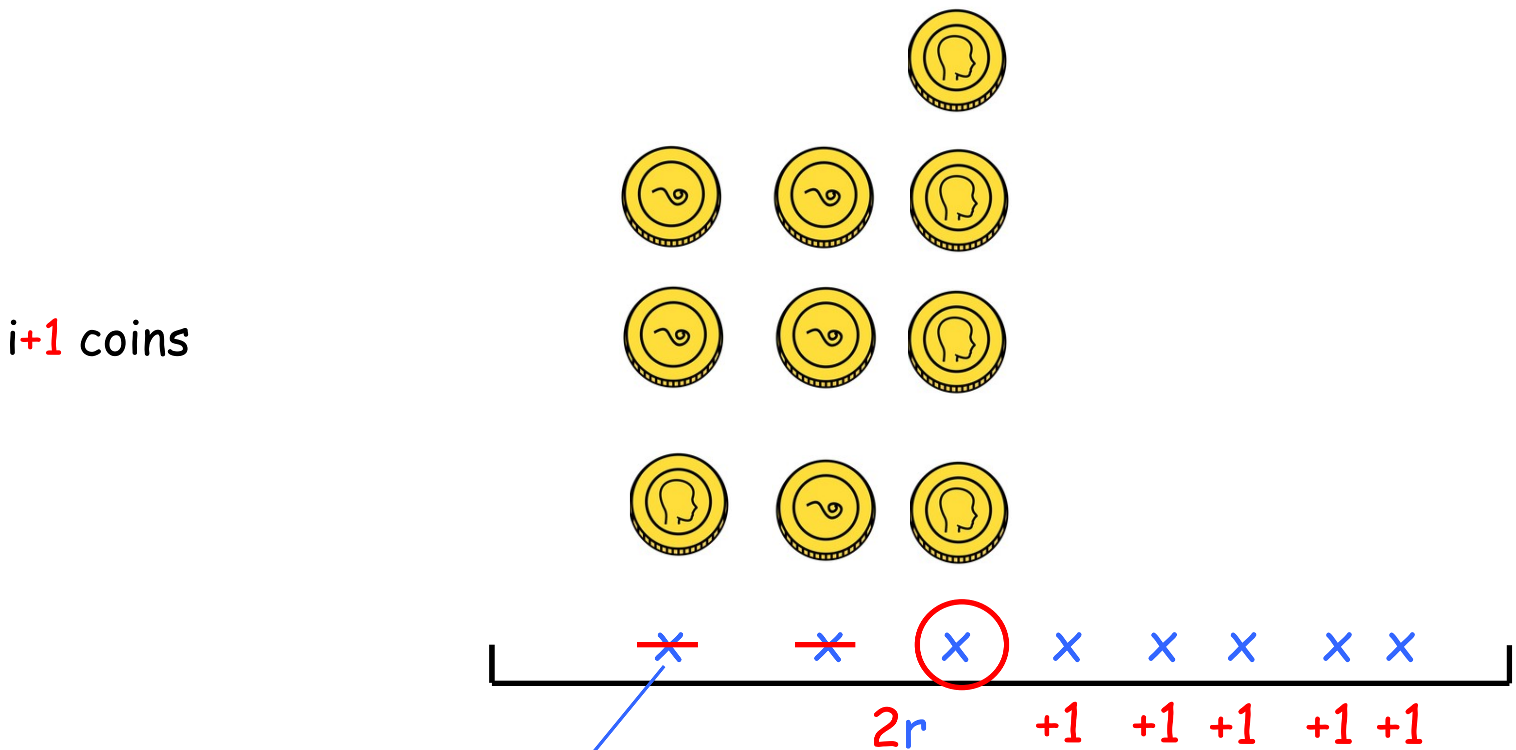


- first occurrence
- put in S with probability $1/r$

The adjusting step: the analysis

obs: r is always a power of 2 ⟹ assume $r=2^i$

focus on a given element x and consider all its occurrences in the window



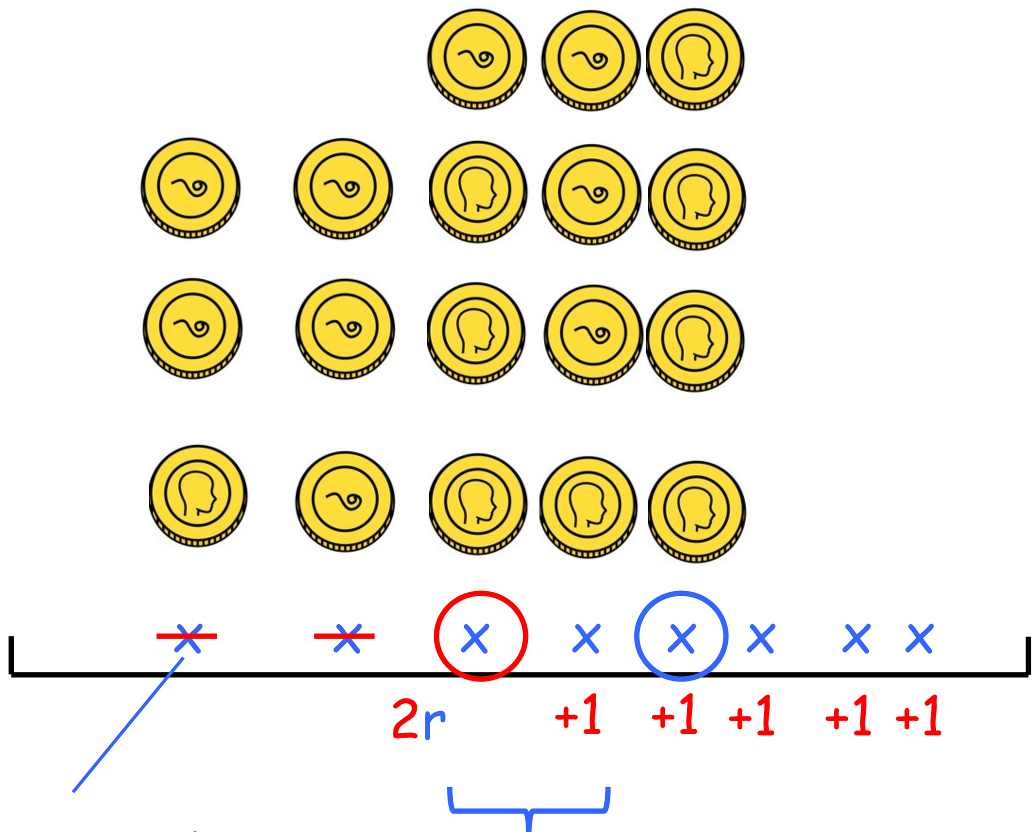
- first occurrence
- put in S with probability $1/r$

The adjusting step: the analysis

obs: r is always a power of 2 ⇒ assume $r=2^i$

focus on a given element x and consider all its occurrences in the window

$i+1$ coins



- first occurrence
- put in S with probability $1/r$

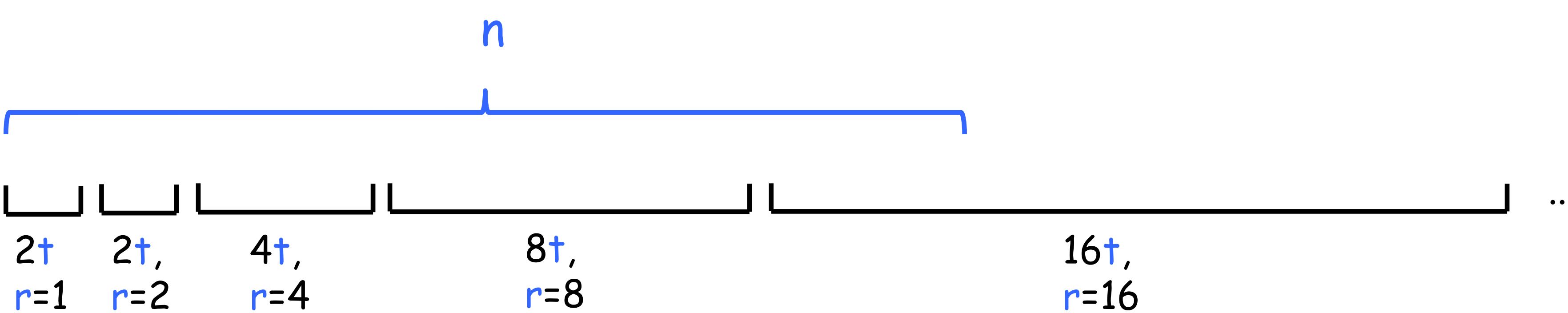
k (correction)

Lemma

Let n be the number of stream elements seen so far and assume that the current sample rate is r. Then $1/r \geq t/n$.

proof

$n \geq t\, r$



Theorem

For any ε, φ, $\delta \in (0,1)$, with $\varepsilon < \varphi$, sticky sampling solves the frequent items problem with probability at least $1 - \delta$ using a sample of expected size $2\varepsilon^{-1} \log (\varphi^{-1} \delta^{-1})$.

notice: $f_e(x) \le f(x)$ $\Rightarrow$ algorithm never return an item with $f(x) < (\varphi-\varepsilon)n$.

let $y_1,\ldots,y_k$, be the elements whose frequency is at least $\varphi\, n$.

Clearly: $k \le 1/\varphi$.

from previous Lemma

$$\Pr[f_e(y_i) < (\varphi-\varepsilon)n] \le (1-1/r)^{\varepsilon n} \le (1- t/n)^{\varepsilon n} \le e^{-t\varepsilon}$$

$$\Pr[\exists \text{ false negative}] \le \sum_{i=1}^{k} \Pr[y_i \text{ is not returned}] \le \sum_{i=1}^{k} \Pr[f_e(y_i) < (\varphi-\varepsilon)n]$$

union bound

$t = \varepsilon^{-1} \log (\varphi^{-1} \delta^{-1})$

$$\le k\, e^{-t\varepsilon} \le \frac{e^{-t\varepsilon}}{\varphi} = \delta$$

what about the size of S?

worst case: all stream elements are distinct

$n \leq 2\,t\,r$

2t   r=1

2t,   r=2

4t,   r=4

8t,   r=8

16t,   r=16

$$X_i \text{ r. v.} = \begin{cases} 1 & \text{if element } i \text{ is inserted in } S \\ 0 & \text{otherwise} \end{cases}$$

$$|S| = X = \sum_i X_i$$

$$E[X] = E\left(\sum_i X_i \right) = \sum_i \underbrace{E[X_i]}_{1/r} = n\,1/r \leq 2\,t$$

Another result for the problem

In the paper

G.S. Manku, R. Motwani:
Approximate Frequency Counts over Data Streams. VLDB (2002)

Sticky sampling algorithm

- randomized
- meet the two goals with probability $1-\delta$
- maintain a sample of expected size of $2\varepsilon^{-1} \log (\varphi^{-1} \delta^{-1})$

$0<\delta<1$: user-defined error parameter

Lossy counting algorithm

- deterministic (meet the two goals with probability 1)
- maintain a sample of size of $O(\varepsilon^{-1} \log (\varepsilon n))$