Big Data Processing, 2014/15

Lecture 3: Data streaming

Claudia Hauff (Web Information Systems)
ti2736b-ewi@tudelft.nl

Warm-up: a small quiz again

- Amount of data produced by security cameras in the UK per week
  - Between 4-6 million CCTV surveillance cameras
  - Storing all surveillance videos for 1 week (14GB/camera) = 66 Petabyte
  - Storing 640x480 videos for 1 year: 3.4 Exabytes (18 zeros!)
- Facebook: amount of data entering their databases every day
- Amount of data available about the average person in clinical databases

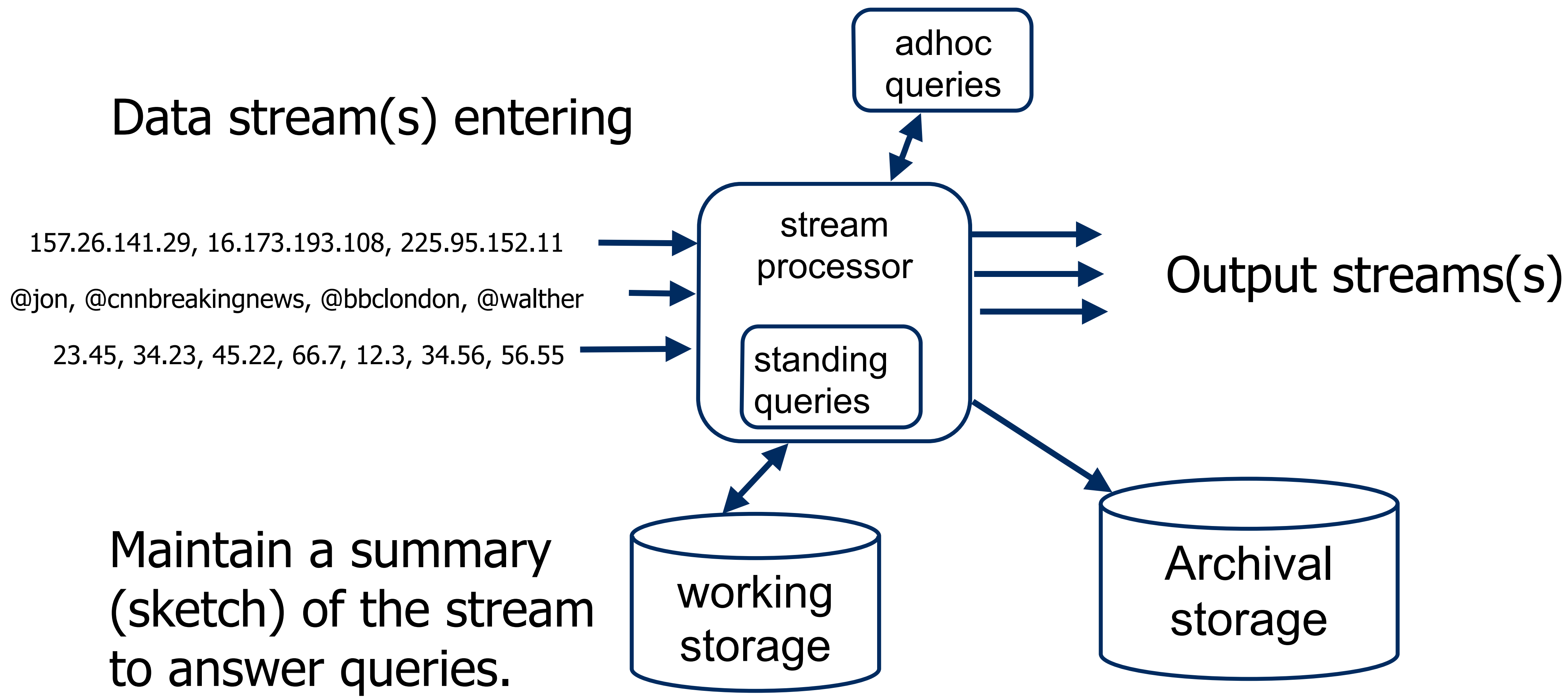
Course content

- Introduction
- **Data streams** 1 & **2**
- The MapReduce paradigm
- Looking behind the scenes of MapReduce: HDFS & Scheduling
- Algorithm design for MapReduce
- A high-level language for MapReduce: Pig 1 & 2
- MapReduce is not a database, but HBase nearly is
- Lets iterate a bit: Graph algorithms & Giraph
- How does all of this work together? ZooKeeper/Yarn

Learning objectives

- **Identify** suitable sampling strategies for a given query
- **Implement** an estimate for the number of distinct elements in a stream (FM-sketch)
- **Implement** an estimate for the order of moments (AMS)
- **Implement** an estimate of counts within a stream window (DGIM)

A reminder: the streaming architecture



A reminder: data streaming characteristics

- **Continuous** and rapid input of data
- **Limited memory** to store the data (less than linear in the input size)
- **Limited time** to process each element
- **Sequential** access
- Algorithms have **one** or very **few passes** over the data

Sampling: think about the query

A reminder: sampling

- Sampling: selection of a subset of items from a large data set
- **Goal**: sample retains the **properties** of the whole data set
- Important for drawing the right conclusions from the data

The setting

- A search engine's query log stream - tuples of `(user, query, time)`
- Working storage can hold **a tenth of the stream**

> **Query**: What fraction of a typical user's queries were repeated over the past month?

- Question: *How should we sample from this stream?*
- One idea: sample tuples **independently**
  - generate a random number [0-9] per tuple and keep the tuples tagged with a 0

The setting

- A search engine's query log stream - tuples of `(user,`
- Working
- **Query**: W… repeated…
- Question…

1337	kentucky fried chicken	2006-04-25 16:07:14
1337	dam kentucky fried chicken menu	2006-04-25 16:12:54
1337	ford	2006-04-26 09:22:34
1410	www.steparoundstep.com	2006-03-01 21:57:38
1410	target bowling green state	2006-04-13 17:41:26
1410	google	2006-05-01 21:40:54
1410	www.toledo11.com	2006-05-14 19:55:41
2005	wnmu homepage	2006-03-07 23:36:11
2005	wnmu webct	2006-03-07 23:47:49
2005	myspace.ocm	2006-03-09 23:12:40
2005	glitter graphics.com	2006-03-10 01:00:41
2005	wnmu.edu	2006-03-23 19:02:42
2005	google	2006-03-24 21:25:10
2005	ww.vibe.com	2006-03-26 21:21:51

Example from the (now) infamous AOL query log
(in a stream those tuples arrive sorted by time)

Sampling tuples independently

Query: What fraction of a typical user's queries were repeated over the past month?

$$a\ user = \begin{cases} s\ queries\ submitted\ once \\ d\ queries\ submitted\ twice \\ 0\ queries\ more\ than\ twice \end{cases} \qquad fr_{repeated} = \frac{d}{d+s}$$

wanted

Apply basic probabilities to compute the 3 choices:

- #queries sampled that were issued once
- #queries sampled twice that were issued twice
- #queries sampled once that were issued twice

Sampling tuples independently

#queries sampled that were issued once: $\frac{s}{10}$

we assume independence

#queries sampled twice that were issued twice: $\frac{d}{10\times 10} = \frac{d}{100}$

#queries sampled once that were issued twice:

$$d\left(\frac{1}{10}\times\frac{9}{10}+\frac{1}{10}\times\frac{9}{10}\right)=\frac{18d}{100}$$

the 1. time the query appears it is sampled

the 2. time the query appears it is sampled

Lets compute the fraction of repeated queries:

#queries sampled twice

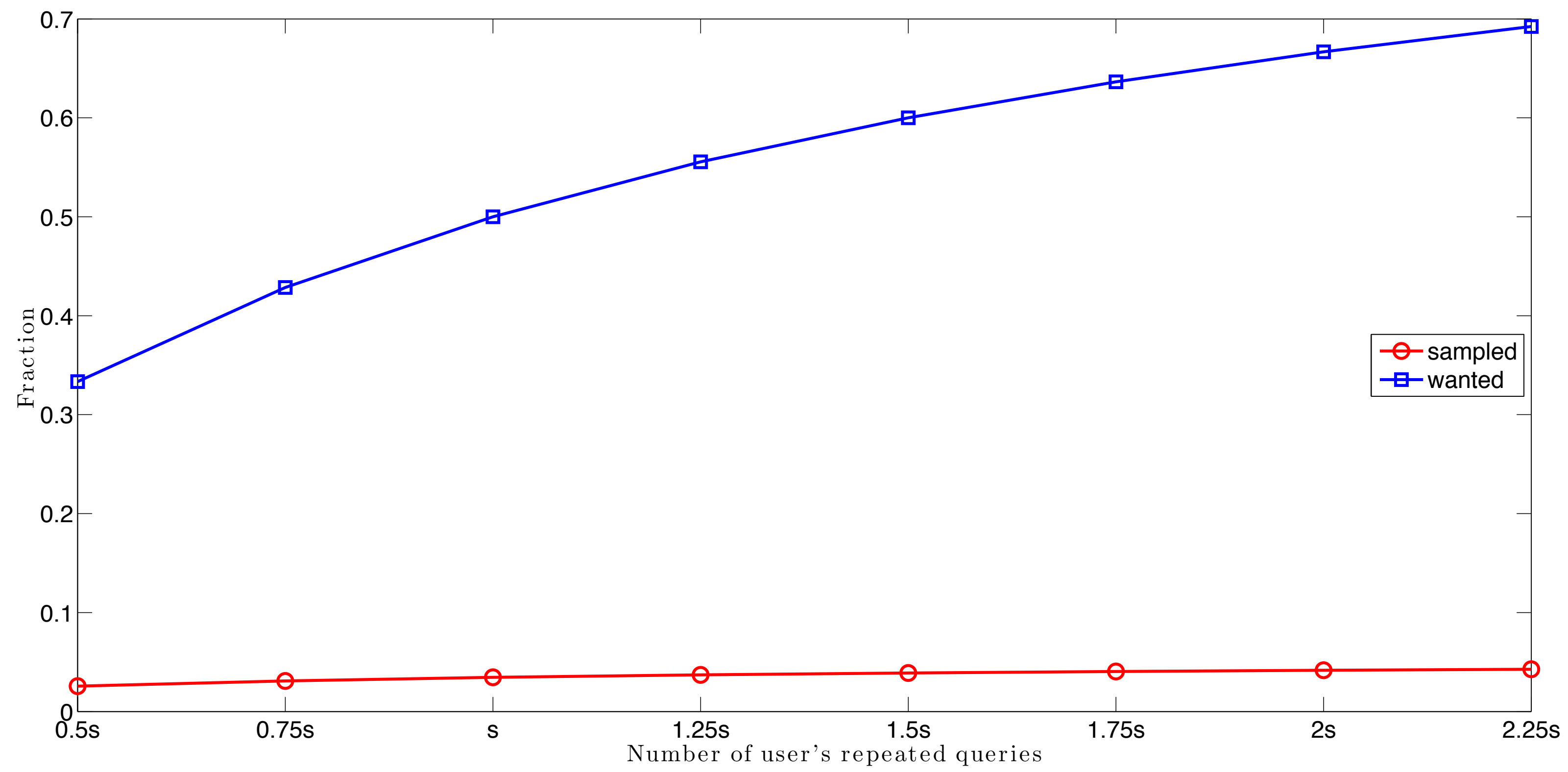
all queries sampled

$$\frac{\frac{d}{100}}{\frac{d}{100}+\frac{s}{10}+\frac{18d}{100}} = \frac{d}{10s+19d}$$

Sampling tuples independently

$$\frac{\frac{d}{100}}{\frac{d}{100}+\frac{s}{10}+\frac{18d}{100}} = \frac{d}{10s+19d} \quad vs. \quad fr_{repeated} = \frac{d}{d+s}$$

sampled **wanted**



How to do it correctly?

Query: What fraction of a typical user's queries were repeated over the past month?

Correct sampling strategy**:**
Sample users, and include all queries issued by a single user in the sample.

Lesson:
even in simple settings, convincing counter-examples exist. Always think about the query!

A little exercise …

Assume we have a data stream of university grades across the Netherlands. The stream elements have the form: *(university, courseID, studentID, grade)*.

Universities are unique, but a courseID is unique only within a university (i.e., different universities may have different courses with the same ID, e.g., "CS101") and likewise, studentID's are unique only within a university (different universities may assign the same ID to different students).

Suppose we want to answer certain queries approximately from a 1/10th sample of the data. For each of the following queries, indicate how you would construct the sample to end up with a good estimate:

(1) Estimate the average number of students in a course.

(university, courseID)

(2) Estimate the fraction of students who have a grade average of 8 or more.

(university, studentID)

(3) Estimate the fraction of universities teaching at least 20 courses.

(university)

Distinct element estimates

Why do we count distinct elements?

- Number of distinct queries issued
- Unique IP addresses passing packages through a router
- Number of unique users accessing a website per month
- Number of different people passing through a traffic hub (airport, etc.)
- ….

Question: how would you do it?

FM-sketch (Flajolet-Martin)

Task: given a data stream, estimate the number of distinct elements occurring in it.

- **Approach**: hash data stream elements uniformly to N bit values, i.e.: $h : a_i \rightarrow \{0,1\}^N$
- **Assumption**: the larger the number of distinct elements in the stream, the more distinct the occurring hash values, and the more likely one with an “unusual” property appears

FM-sketch (Flajolet-Martin)

- One possibility of interpreting "unusual" is the **hash tail**: the number of 0's a binary hash value ends in

  100110101110   100110101100   100110000000

- **Algorithm:**

$$\textbf{for all } a_i \in S,$$
$$h(a_i) \rightarrow \{0,1\}^N$$

Maximum hash tail seen so far

$$K = \max_{a_i \in S} h(a_i)$$
$$return\ |\tilde{S}| = 2^K$$

Important:
N must be long enough: there must be more possible results of the hash function than elements in the universal set.

FM-sketch (Flajolet-Martin)

- Intuitive justification:

$$P(h(a)\ \textit{has tail length of at least } r) = \frac{1}{2\times 2\ldots\times 2} = \frac{1}{2^r}$$

r 0’s occur

- When there are m distinct elements in the stream

$$P(\textit{none has tail length} \geq r) = \left(1 - \frac{1}{2^r}\right)^m$$

$$\textit{if } m \gg 2^r : \textit{the prob. of finding a tail } \geq r \textit{ reaches 1}$$

$$\textit{if } m \ll 2^r : \textit{the prob. of finding a tail } \geq r \textit{ reaches 0}$$

Thus: the proposed estimate is neither too low nor too high.

FM-sketch (Flajolet-Martin)

- **Practical setup**
  - axb independent sketches
  - a groups of b sketches each
  - *Median of means* for a stable result

Estimating moments

Moments vs. distinct elements

- Moments are a generalisation of the task just discussed

Let m_i be the frequency of the i^{th} element for any i.
The k^{th}-order moment of the stream is:

$F_k = \sum_i (m_i)^k$

- 0th moment: sum of 1 for each $m_i > 0$ (i.e. the number of distinct elements)
- 1st moment: sum of all m_i (i.e. the stream length)
- 2nd moment: sum of all m_i^2 (the “surprise” number)

Moments vs. distinct elements

- Moments are a generalisation of the task just discussed

Let m_i be the frequency of the i^{th} element for any i.
The k^{th}-order moment of the stream is:

$$F_k = \sum_i (m_i)^k$$

- 0th mom
  number
- 1st mom
- 2nd mor)

A stream with 100 elements, 11 distinct ones:
(S1) 10 elements occur 9 times, 1 element 10 times
(S2) 10 elements occur 1 time, 1 element 90 times

$$10^2 + 10 \times 9^2 = 910$$
$$90^2 + 10 \times 1^2 = 8110$$

AMS algorithm (Alon-Matias-Szegedy) for 2nd order moments

- Lets define:

$$
\begin{aligned}
&X = (element, value)\\
&X.element\text{: element of the universal set}\\
&X.value \quad \text{: counter of } X.element \text{ in the stream}\\
&\qquad\qquad\quad\ \text{starting at a randomly chosen position}
\end{aligned}
$$

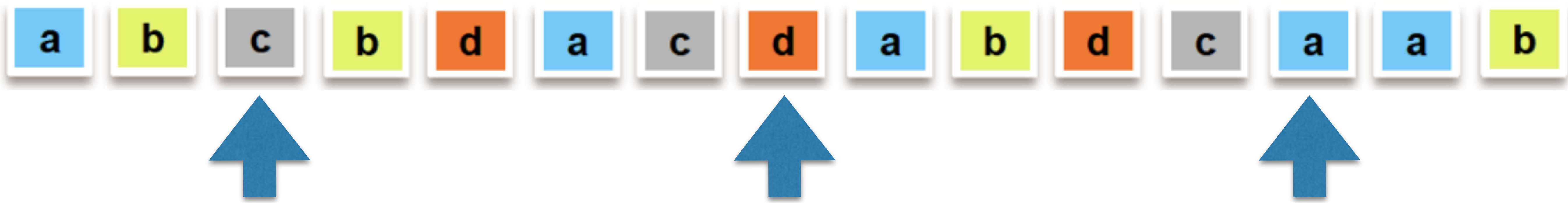
- Example:

a b c b d a c d a b d c a a b

$$
\begin{aligned}
&n = 15\\
&m_a^2 + m_b^2 + m_c^2 + m_d^2 = 5^2 + 4^2 + 3^2 + 3^2 = 59
\end{aligned}
$$

correct solution

AMS algorithm (Alon-Matias-Szegedy) for 2nd order moments



(1)Randomly pick 3 positions (3 variables to compute the 2nd order) from stream with **known length**

$X_1 = (c, 1)$

$X_1 = (c, 2)$

$X_2 = (d, 1)$

$X_2 = (d, 2)$

$X_1 = (c, 3)$

$X_3 = (a, 1)$

$X_3 = (a, 2)$

(2)Process the stream, one element at a time

AMS algorithm (Alon-Matias-Szegedy) for 2nd order moments

- **Estimate of the 2nd order moment** from any $X = (element, value)$:

argument will follow

$$n \times (2 \times X.value - 1)$$

- Applied to our example:

$$\begin{aligned}
&\text{estimate from } X_1\text{: } 15 \times (2 \times 3 - 1) = 75 \\
&\text{estimate from } X_2\text{: } 15 \times (2 \times 2 - 1) = 45 \\
&\text{estimate from } X_3\text{: } 15 \times (2 \times 2 - 1) = 45
\end{aligned}$$

$$AVG(X_1, X_2, X_3) = 55$$

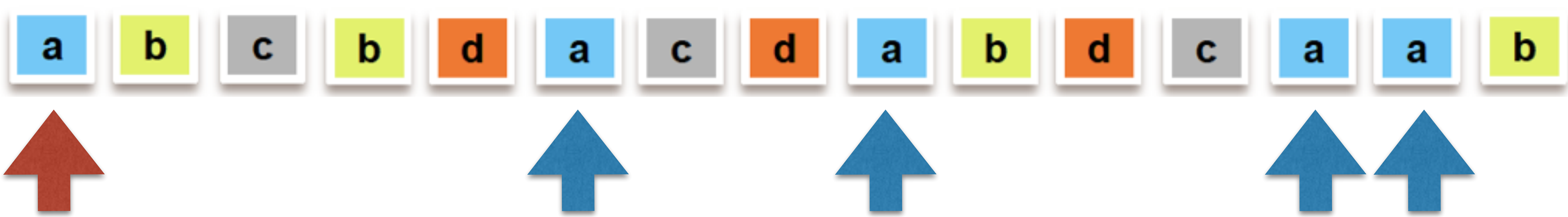
approximate solution

AMS algorithm (Alon-Matias-Szegedy) for 2nd order moments

- **To show**: expected value of any X constructed this way is the second moment of the stream

- **Notation:**

$e(i)$: stream element at position i in the stream
$c(i)$: number of times $e(i)$ appears starting at position i



ignored

$e(6) = a$
$c(6) = 4$

AMS algorithm (Alon-Matias-Szegedy) for 2nd order moments

Expected value:

$$E(X.value) = \frac{1}{n}\sum_{i=1}^{n} \overbrace{n \times (2 \times c(i) - 1)}^{n \times (2 \times X.value - 1)}$$

$$= \sum_{i=1}^{n} (2 \times c(i) - 1)$$

$$E(X_a.value) = \sum_{a} 1 + 3 + 5 ... + (2m_a - 1)$$ reordering, starting with the last position of element a;

$$= (m_a)^2$$

$e(i)$: stream element at position i in the stream

$c(i)$: number of times $e(i)$ appears starting at position i

AMS algorithm (Alon-Matias-Szegedy) for 2nd order moments

Lets look at our **example** again …

$$E(X_a.value) = \sum_a 1+3+5...+(2m_a-1)$$
$$=(m_a)^2$$

$c(a)=5$ $c(a)=4$ $c(a)=3$ $c(a)=2$ $c(a)=1$

a b c b d a c d a b d c a a b

$$(2\times5-1)+(2\times4-1)+(2\times3-1)+(2\times2-1)+(2\times1-1)$$
$$=9+7+5+3+1$$
$$=25$$
$$=5\times5$$

$e(i)$: stream element at position i in the stream

$c(i)$: number of times $e(i)$ appears starting at position i

Higher-order moments

- More generally, to estimate the kth order moment for any X:

$$X = (element, value) \text{ where } x = X.value \text{ and } k \geq 2$$

$$n \times \left(v^k - (v-1)^k\right)$$

Higher-order moments

- **So far**: we assumed to know the length of the stream
  - But: streams grow with time
  - Problematic when computing $n \times (2 \times X.value - 1)$
- What about our **selection procedure** for the positions of X?
  - We need a *uniform random sample*
  - Choosing early positions: with increasing length bias is introduced; overestimate of moment
  - Choosing recent positions: few encounters, unreliable estimates
- **Solution**: maintain as many variables as storage allows, replace them as the stream grows (reservoir sampling)

Stream windows

Stream windows: counting

- **So far**:
  - **Entire** stream is interesting
  - Frequent item estimates (MAJORITY, FREQUENT, etc.)

- **In practice** many estimates are based on the most recent stream elements
  - Trending new articles on Twitter
  - Trending sales on Amazon
  - …

Counting 1's in a binary stream

Task: given a stream of binary numbers and a window length N, how many 1's appear **in the last k bits** with $k \leq N$. We **cannot** store the full window.

$w = 10101$
$x = 01101$
$\Rightarrow k = 4$

- **Spatial complexity** of exact count
  - representation <N bits for N-bit window
  - 2^N distinct bit sequences of length N

2 different bit strings w, x have the same representation

  - w and x by definition differ in at least 1 bit (at pos. k)
  - representation is whatever bits both represent w and x
  - Query: how many 1's are in the last k bit?

DGIM (Datar-Gionis-Indyk-Motwani): Estimating counts of 1's

- **Preliminaries**
  - Each bit has a timestamp: stream position mod. N $\log_2 N \ \ bits$
  - Window is divided into buckets represented by
    - Timestamp of its right end $\log_2 N \ \ bits$
    - "Size": number of 1's in a bucket (pow. of 2) $\log_2 \log_2 N \ \ bits$

    Space complexity of a bucket: $O(\log N)$
- **DGIM rules:** $\text{e.g. } \ 2^j, \text{ encode } j \text{ and } j \leq \log_2 N$
  1. Right end of a bucket is always a 1 bit
  2. No position is in more than one bucket
  3. There are 1-2 buckets of any given size
  4. All bucket sizes are a power of 2
  5. Buckets cannot decrease in size as we move back in time

DGIM: Estimating counts of 1's

- Lets look at an **example**:

0's can be outside of buckets

. . 1 0 1 | 1 0 1 1 0 0 0 1 | 0 | 1 1 1 0 1 | 1 0 0 1 | 0 | 1 | 1 | 0

one bucket of size 8

two buckets of size 4

one bucket of size 2

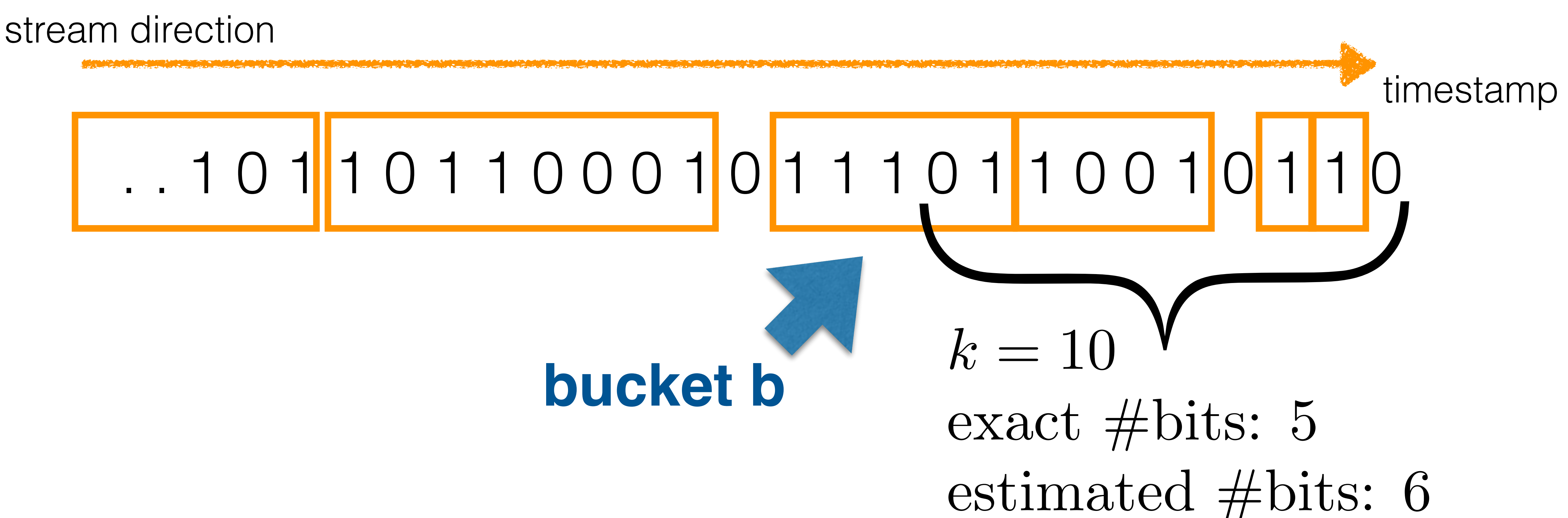
two buckets of size 1

- **Space complexity**:
  - Each bucket in: $O(\log N)$
  - Maximum number of buckets: $O(\log N)$

Overall: $O(\log^2 N)$

DGIM: Estimating counts of 1's

- **Estimation of 1's within the last k bits**
  1. Determine bucket **b** with the earliest timestamp that includes at least some of the k most recent bits
  2. Sum the sizes (#1's) of all buckets to the right of **b**
  3. Final estimate: add size(b)/2 to the sum



DGIM: Estimating counts of 1's

- What is the maximum error?

- **Case 1:** estimate is less than correct answer c
  - Worst case: all 1's of b are in range of the query but only half are included (by def. of the estimate)
  - Thus, estimate is at least 50% of c

- **Case 2**: estimate is greater than c
  - Worst case: only the rightmost bit of b is in range and only one bucket of each size smaller than b exists

$$c = 1 + 2^{j-1} + 2^{j-2} + ... + 1 = 2^j$$

$$\text{estimate} = 2^{j-1} + 2^{j-1} + 2^{j-2} + .. + 1 = 2^j + 2^{j-1} - 1$$

  - Thus, estimate is no more than 50% greater than c

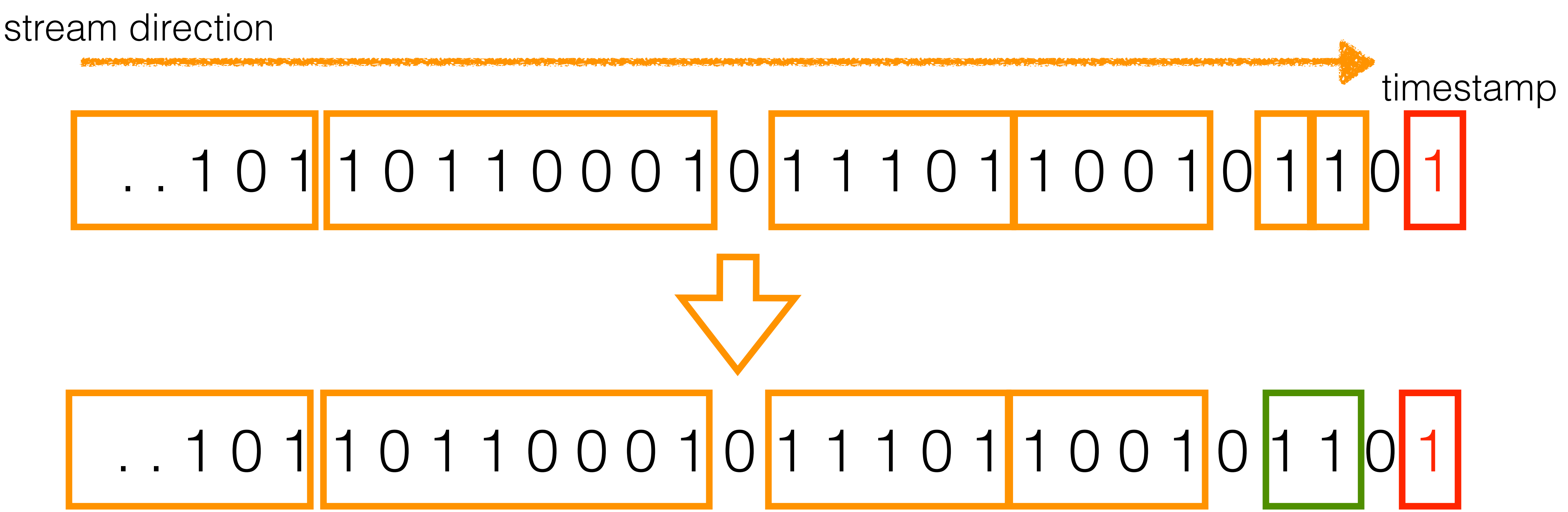
DGIM: Estimating counts of 1's

- Updating the buckets with increasing stream length
- **Starting condition**: window of length N with correct bucketing
- **Update**: new bit *nb* 'enters' the window
  - **Check timestamp** of leftmost bucket, drop it, if it completely falls outside the new window
  - If *nb* is set to 1:
    - **Create a new bucket** with current timestamp and size 1
      - If 3 buckets of size 1 exist, merge the two older ones, create a bucket of size 2
        - If 3 buckets of size 2 exist ….

Update time complexity: $O(\log N)$

DGIM: Estimating counts of 1's

- Example: one bit enters the window



What happens if now a 0 enters the stream?

What happens if now a 1 enters the stream?

Summary

- Sampling
- FM-sketch
- kth-order moments
- DGIM

THE END