Big Data Processing, 2014/15

Lecture 2: Data streaming

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

Warm-up: a small quiz

- How many emails are send per day across the world?
- What percentage of emails sent is spam?
- (Bonus: guess the most popular spam topic)

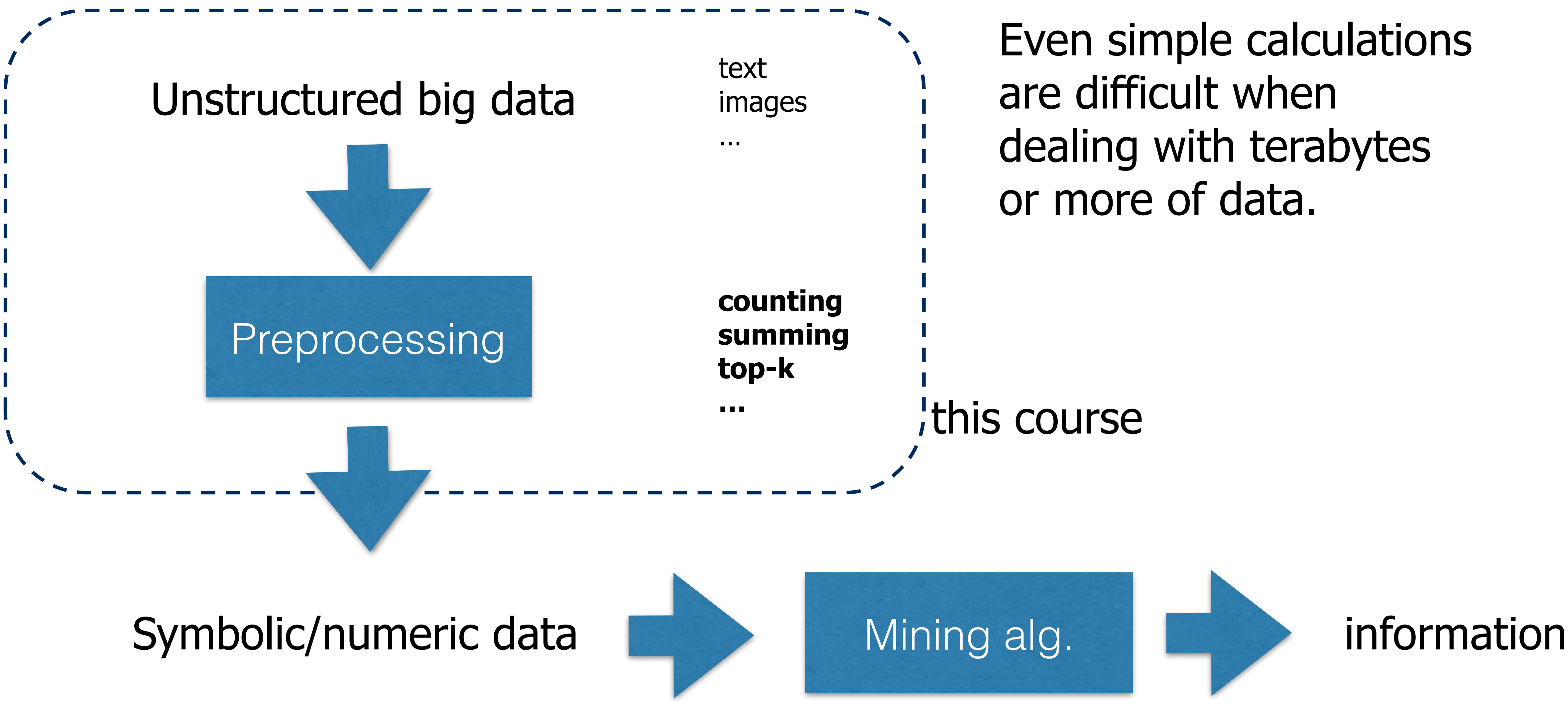
Course content

- 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

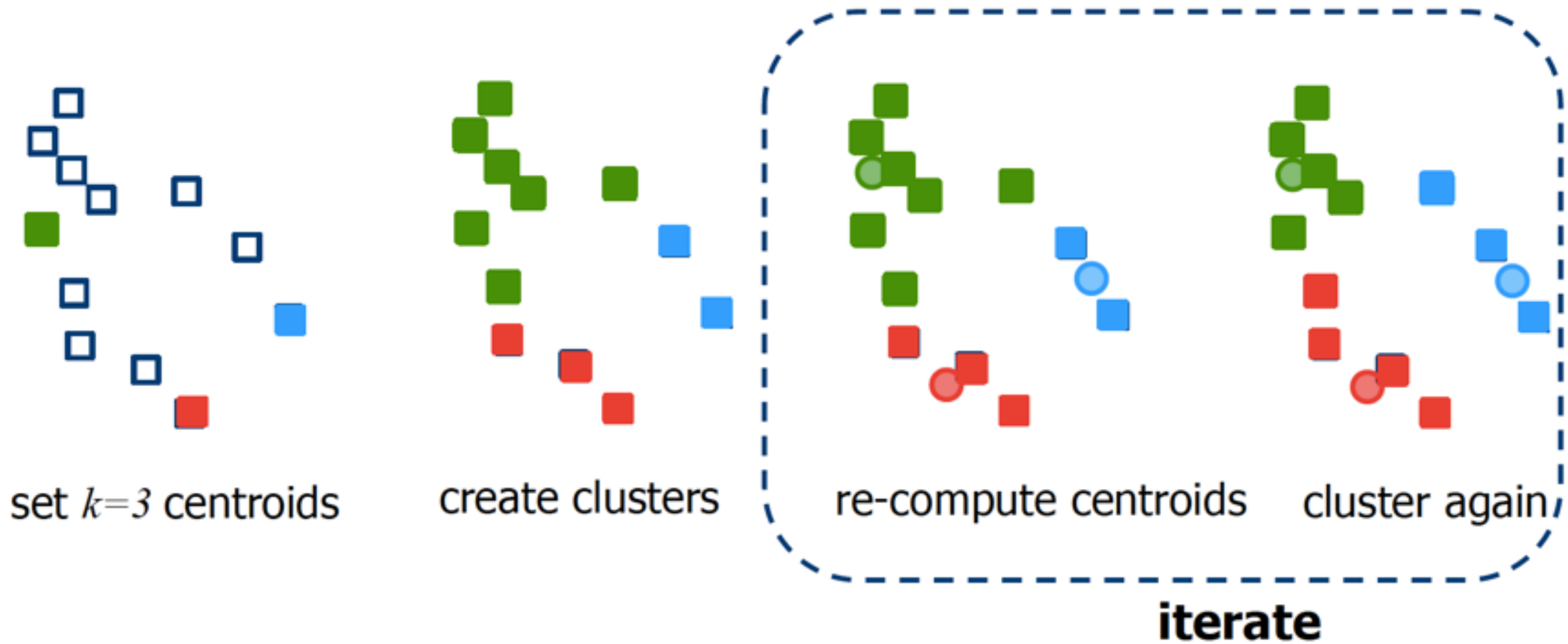
- **Explain** the limiting factors of data streaming & describe the different data stream models
- **Implement** sampling approaches for data streams
    - RESERVOIR sampling
    - MIN-WISE sampling
- **Implement** counter-based frequent item estimation approaches
    - MAJORITY
    - FREQUENT
- **Implement** BLOOM filters

BDP vs. Data Mining, Pattern Recognition, AI, …



Even simple calculations are difficult when dealing with terabytes or more of data.

K-means clustering

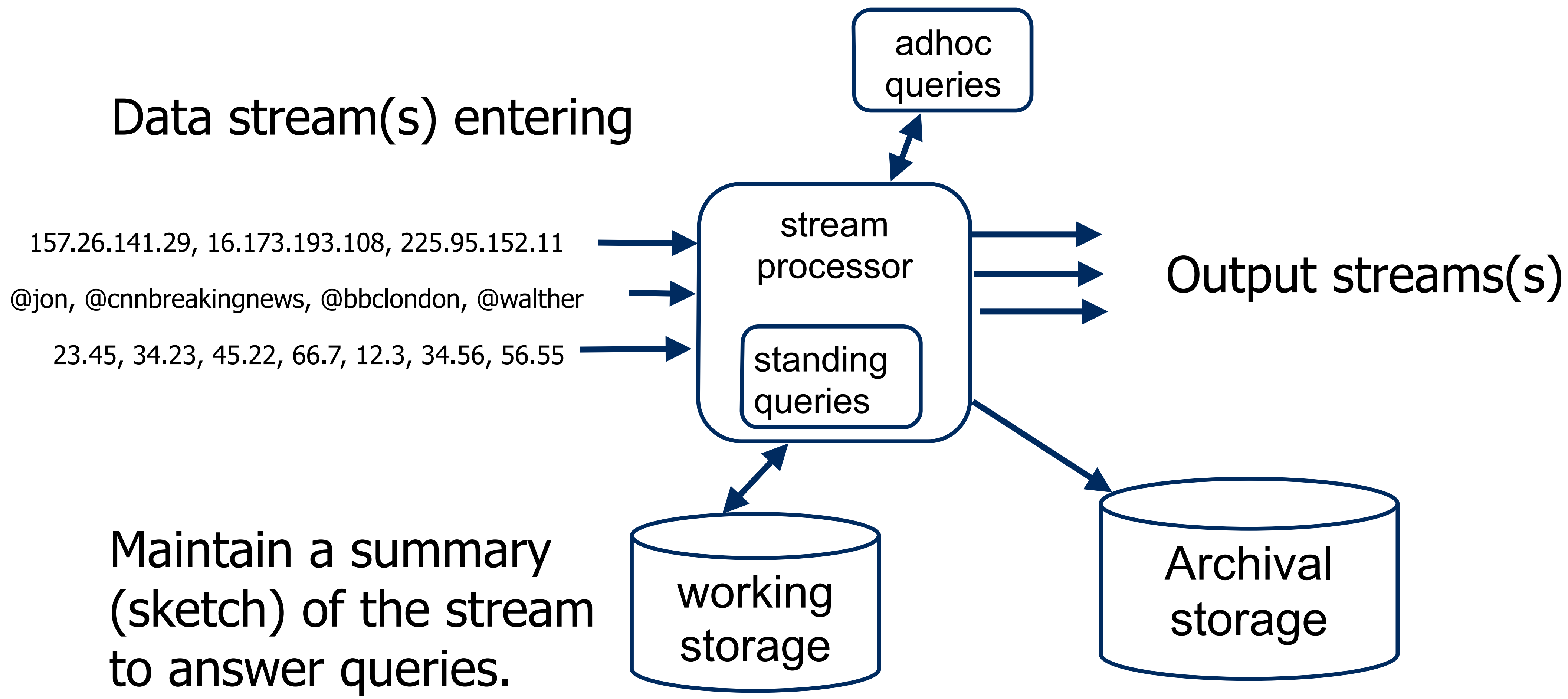


Goal: partition the N elements into k disjoint sets S_j with minimized sum of squares:

$$\sum_{j=1}^{k} \sum_{n \in S_j} \left| x_n - \mu_j \right|^2$$

Data streaming

Streaming architecture



Data streaming scenario

- **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

Data streaming scenario

- Typically: **simple functions** of the stream are computed and used as input to other algorithms
    - Number of distinct items
    - Heavy hitters
    - Longest increasing subsequence
    - ….

- Closed form solutions are rare - **approximation** and **randomisation** are the norm

Data stream models

- Massively long input stream

- Basic "vanilla" model:

not a restriction: requires a single preprocessing step to convert symbols to integers

$$\sigma =< a_1, a_2, a_3, .., a_m >$$

$$with\ elements\ drawn\ from\ [n] := 1, 2, ..., n$$

universe size

- Ultimate space complexity goal: to use s bits of random-access memory needed to store a constant number of counters and tokens where:

$$s = O(\log m + \log n) \qquad s = poly \log(min(m, n))$$

reality

Data stream models

- Massively long input stream

- Basic "vanilla" model:

$$\sigma =< a_1, a_2, a_3, .., a_m >$$

$$with\ elements\ drawn\ from\ [n] := 1, 2, ..., n$$

not a restriction: requires a single preprocessing step to convert symbols to integers

universe size

"For instance, estimating cardinalities [**number of distinct elements**] … of **a hundred million different records** can be achieved with m=2048 memory units of 5 bits each, which corresponds to **1.28 kilobytes of auxiliary storage** in total, the **error** observed being typically **less than 2.5%**."

use s bits of
store a
kens where:

$\log(min(m, n))$

reality

Data stream models

- Frequency vectors: computing some statistical property from the multi-set of items in the input stream

$$\mathbf{f} = (f_1, f_2, ..., f_n) \ \ where \ f_j = |i : a_i = j|$$

$$with \ \mathbf{f} \ starting \ at \ 0$$

- Turnstile model: elements can "arrive" and "depart" from the multi-set by variable amounts

$$upon \ receiving \ a_i = (j, c), update \ f_j \leftarrow f_j + c$$

- Cash register model: only positive updates are allowed

Data stream models

- Frequency vectors: computing some statistical property from the multi-set of items in the input stream

$$\mathbf{f} = (f_1, f_2, \ldots, f_n) \ \textit{where} \ f_j = |i : a_i = j|$$

$$\textit{with } \mathbf{f} \textit{ starting at } 0$$

- Turnstile model: elements can "arrive" and "depart" from the multi-set by variable amounts

-

A data streaming algorithm A takes the stream as input and computes a function $\phi(\sigma)$

Sampling

Sampling

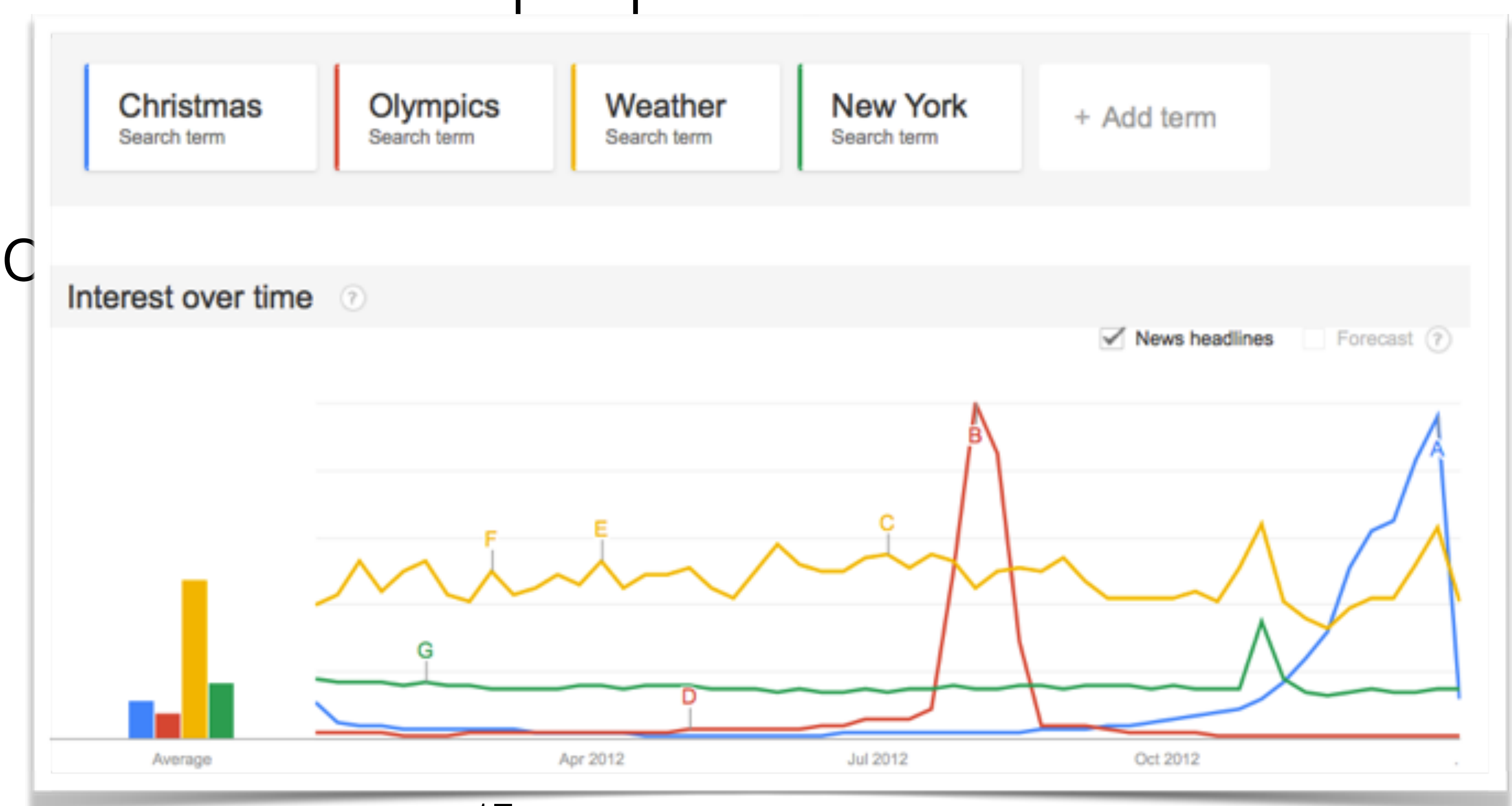
- Sampling: selection of a subset of items from a large data set
- **Goal**: sample retains the properties of the whole data set
- Important for drawing the right conclusions from the data

Sampling

- Sampling: selection of a subset of items from a large data set

- **Goal**: sample retains the properties of the whole data set

- Important fo
  the data



Sampling framework

- Algorithm A chooses every incoming element with a certain probability
- If the element is “sampled”, A puts it into memory, otherwise the element is discarded
- Depending on different situations, algorithm A may discard some items from memory after having added them
- For every query of the data set, algorithm A computes some function $\phi(\sigma)$ only based on the in-memory sample

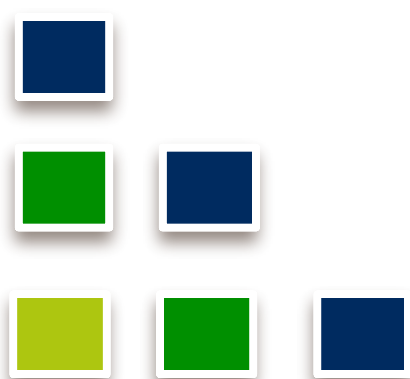
Reservoir sampling

Task: Given a data stream of unknown length, randomly pick k elements from the stream so that each element has the same probability of being chosen.

m=1

m=2

m=3



keep it

replace $\blacksquare$ with probability 1/2

replace $\blacksquare$ / $\blacksquare$ with probability 1/3

keep $\blacksquare$ / $\blacksquare$ with probability 2/3

$$P(\blacksquare) = 1 \times \frac{1}{2} \times \frac{2}{3} = \frac{1}{3}$$

$$P(\blacksquare) = \frac{1}{2} \times \frac{2}{3} = \frac{1}{3}$$

$$P(\blacksquare) = \frac{1}{3}$$

Reservoir sampling

1. Sample the first k elements from the stream
2. Sample the ith element (i>k) with probability k/i (if sampled, randomly replace a previously sampled item)

- **Limitations**:
  - Wanted sample fits into main memory
  - Distributed sampling is not possible (all elements need to be processed sequentially)

- Time/space complexity?

Reservoir sampling example

- Stream of numbers with a normal distribution *N(0,1)*

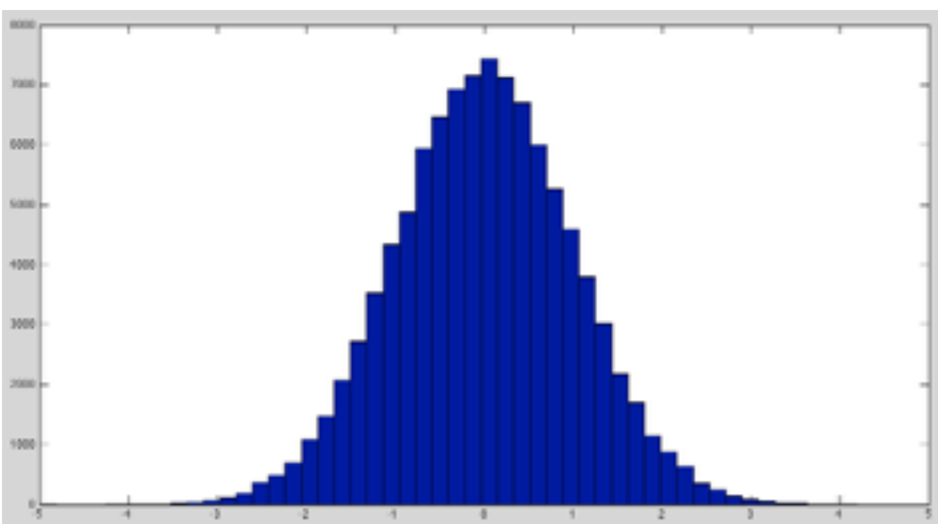
$$|S| = 100000$$

$$k = \{100, 500, 1000, 10000\}$$

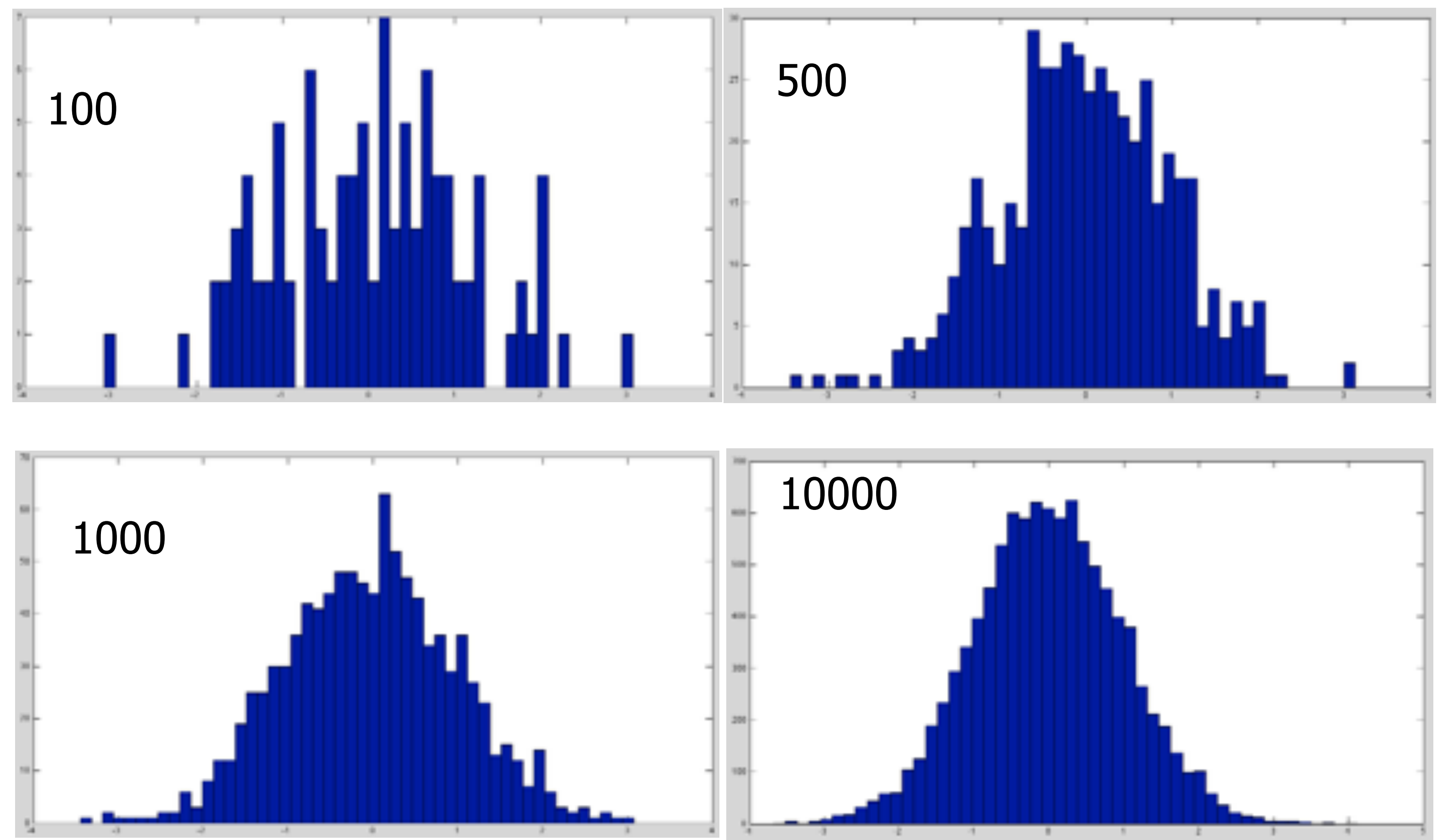
- Samples are plotted in histogram form

- Expectation: with larger k, the histograms become more similar to the full stream histogram

Reservoir sampling example



Histogram of entire stream



Min-wise sampling

Task: Given a data stream of unknown length, randomly pick *k* elements from the stream so that each element has the same probability of being chosen.

1. For each element in the stream, tag it with a random number in the interval [0,1]
2. Keep the k elements with the smallest random tags

Min-wise sampling

Task: Given a data stream of unknown length, randomly pick k elements from the stream so that each element has the same probability of being chosen.

- Can be run in a **distributed** fashion with a merging stage (every subset has the same chance of having the smallest tags)
- Disadvantage: **more memory/CPU intensive** than reservoir sampling

Sampling: summary

- **Advantages**:
  - Low cost
  - Efficient data storage
  - Classic algorithms can be run on it (all samples should fit into main memory)
- Not always that simple to retrieve a uniform sample
  - **Time-sensitive window**: only the last x items of the stream are of interest (e.g. in anomaly detection)
  - **Sampling from databases** through their indices (non-cooperative provider)
    - How many car repairs does Google Places index?
    - How many documents does Google index?

Frequency counter algorithms

MAJORITY algorithm

Task: Given a list of numbers [representing votes]; is there an absolute majority (an element occurring $> \frac{m}{2}$ times?

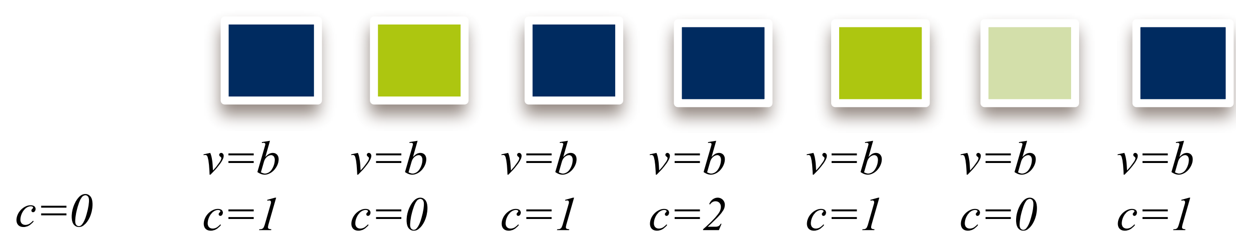
no absolute majority

blue wins

$c \leftarrow 0$; v unassigned;
for each i :
  if $c = 0$:
    $v \leftarrow i$;
    $c \leftarrow 1$;
  else if $v = i$:
    $c \leftarrow c+1$;
  else:
    $c \leftarrow c-1$;

MAJORITY algorithm

Task: Given a list of numbers [representing votes]; is there an absolute majority (an element occurring $> \frac{m}{2}$ times?

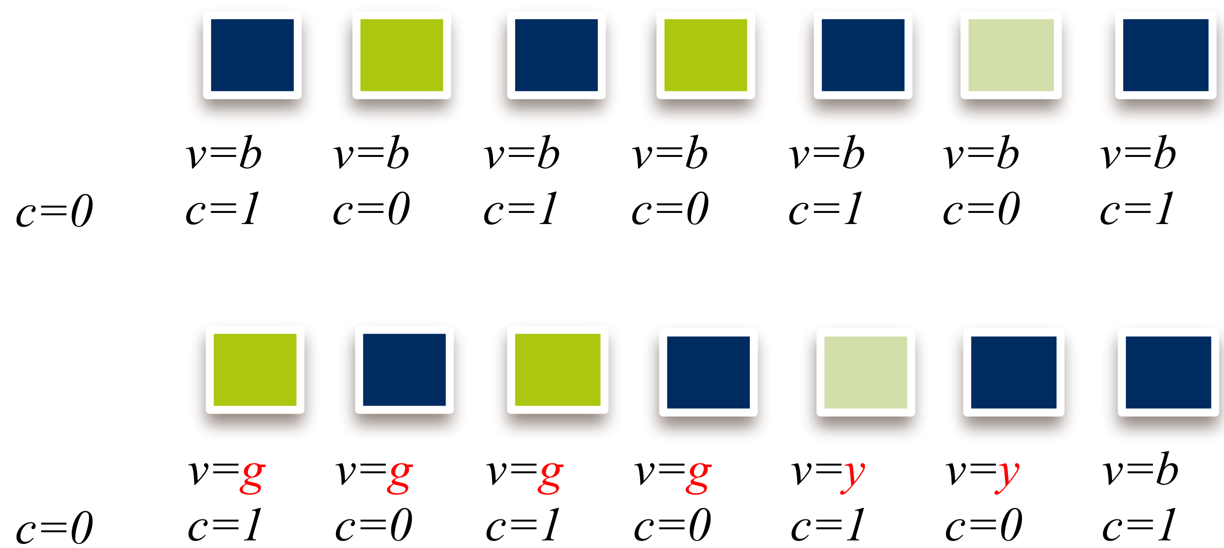


In this stream, the last item is kept.

A **second pass** is needed to verify if the stored item is indeed the absolute majority item (simply count every occurrence of b)

MAJORITY algorithm

Task: Given a list of numbers [representing votes]; is there an absolute majority (an element occurring $> \frac{m}{2}$ times?



Correctness based on pairing argument:

- Every non-majority element can be paired with a majority element
- After the pairing, there will still be majority elements left

FREQUENT algorithm (Misra-Gries)

- Generalization of MAJORITY: find all elements in a sequence whose frequency exceeds 1/k fraction of the total count (i.e. frequency >m/k)

- Output is an **estimate of the frequency** of the most often occurring elements
- Single pass algorithm

- Wanted: no false negatives, i.e. all elements with >m/k frequency should be reported

$c[1,..(k-1)] = 0; T \leftarrow \varnothing;$

for each i :

    if $i \in T$:

        $c_i \leftarrow c_i + 1;$

    else if $|T| < k-1$:

        $T \leftarrow T \cup \{i\};$

        $c_i \leftarrow 1;$

    else for all $j \in T$:

        $c_j \leftarrow c_j - 1;$

        if $c_j = 0$:

            $T \leftarrow T \setminus \{j\};$

FREQUENT algorithm (Misra-Gries)

Blue and green have been estimated to each occur 3 times.

k=3
c=0

v1=g	*v1=g*	*v1=g*	*v1=g*	*v1=g*	*v1=g*	*v1=g*	*v1=g*	*v1=g*	*v1=g*	*v1=g*	*v1=g*
c1=1	*c1=2*	*c1=2*	*c1=3*	*c1=3*	*c1=2*	*c1=1*	*c1=1*	*c1=2*	*c1=2*	*c1=3*	*c1=3*
v2 ---	*v2 ---*	*v2=b*	*v2=b*	*v2=b*	*v2=b*	~~*v2=b*~~	*v2=b*	*v2=b*	*v2=b*	*v2=b*	*v2=b*
c2=0	*c2=0*	*c2=1*	*c2=1*	*c2=2*	*c2=1*	*c2=0*	*c2=1*	*c2=1*	*c2=2*	*c2=2*	*c2=3*

Stream with m=12 elements, all elements with more than m/k (i.e. 12/3=4) occurrences should be reported.
The elements are reported correctly, the estimates are off by 2.

FREQUENT algorithm (Misra-Gries)

Green is estimated to have occurred once. No other element more than m/k occurrences.

$k=3$
$c=0$

$v1=g$	$v1=g$	$v1=g$	$v1=g$	$v1=g$	$v1=g$	$v1=g$
$c1=1$	$c1=2$	$c1=2$	$c1=3$	$c1=3$	$c1=2$	$c1=1$
$v2$ ---	$v2$ ---	$v2=b$	$v2=b$	$v2=b$	$v2=b$	$\cancel{v2=b}$
$c2=0$	$c2=0$	$c2=1$	$c2=1$	$c2=2$	$c2=1$	$c2=0$

Stream with m=7 elements, all elements with more than m/k (i.e. 7/3=2.333) occurrences should be reported. The element [Green] is reported correctly, the estimate is off by 2.

FREQUENT algorithm (Misra-Gries)

Blue is estimated to have occurred once, green 3 times.

$k=3$
$c=0$

$v1=g$	$v1=g$	$v1=g$	$v1=g$
$c1=1$	$c1=2$	$c1=2$	$c1=3$
$v2$ ---	$v2$ ---	$v2=b$	$v2=b$
$c2=0$	$c2=0$	$c2=1$	$c2=1$

Stream with m=4 elements, all elements with more than m/k (i.e. 4/3=1.333) occurrences should be reported.
The element [Green] is reported correctly.
[Blue] is not an element appearing more than 1/k times (a false positive), however remember that these algorithms provide an estimate only.

This is an issue on toy examples or very skewed datasets, not usually in practical streaming examples with millions/billions of elements.

FREQUENT algorithm (Misra-Gries)

- **Space complexity**
  - Implementation: associative array using a balanced binary search tree
  - Each key has a max. value of n, each counter has a max. value of m
  - At most (k-1) key/counter pairs in memory at any time

$$s = O(k(\log m + \log n))$$

FREQUENT algorithm (Misra-Gries)

- **Answer quality** of the frequency estimates (counters):
  - Counter cj is incremented only when j occurs, thus
    $$\tilde{f}_j \leq f_j$$
  - When cj is decremented, (k-1) counters are decremented overall (all distinct tokens); for a stream of size m, there can be at most m/k decrements, thus
    $$f_j - \frac{m}{k} \leq \tilde{f}_j \leq f_j$$

Filtering

Summarizing vs. filtering

- **So far:** all data is useful, summarise it due to the lack of space/ time
- **Now**: not all data is useful, some is harmful
- Classic example: spam filtering
  - Mail servers can analyse the textual content
  - Mail servers have blacklists
  - Mail servers have whitelists (very effective!)
  - Incoming mails form a stream; quick decisions needed (delete or forward)
- Applications in Web caching, packet routing, resource location, etc.

Problem statement

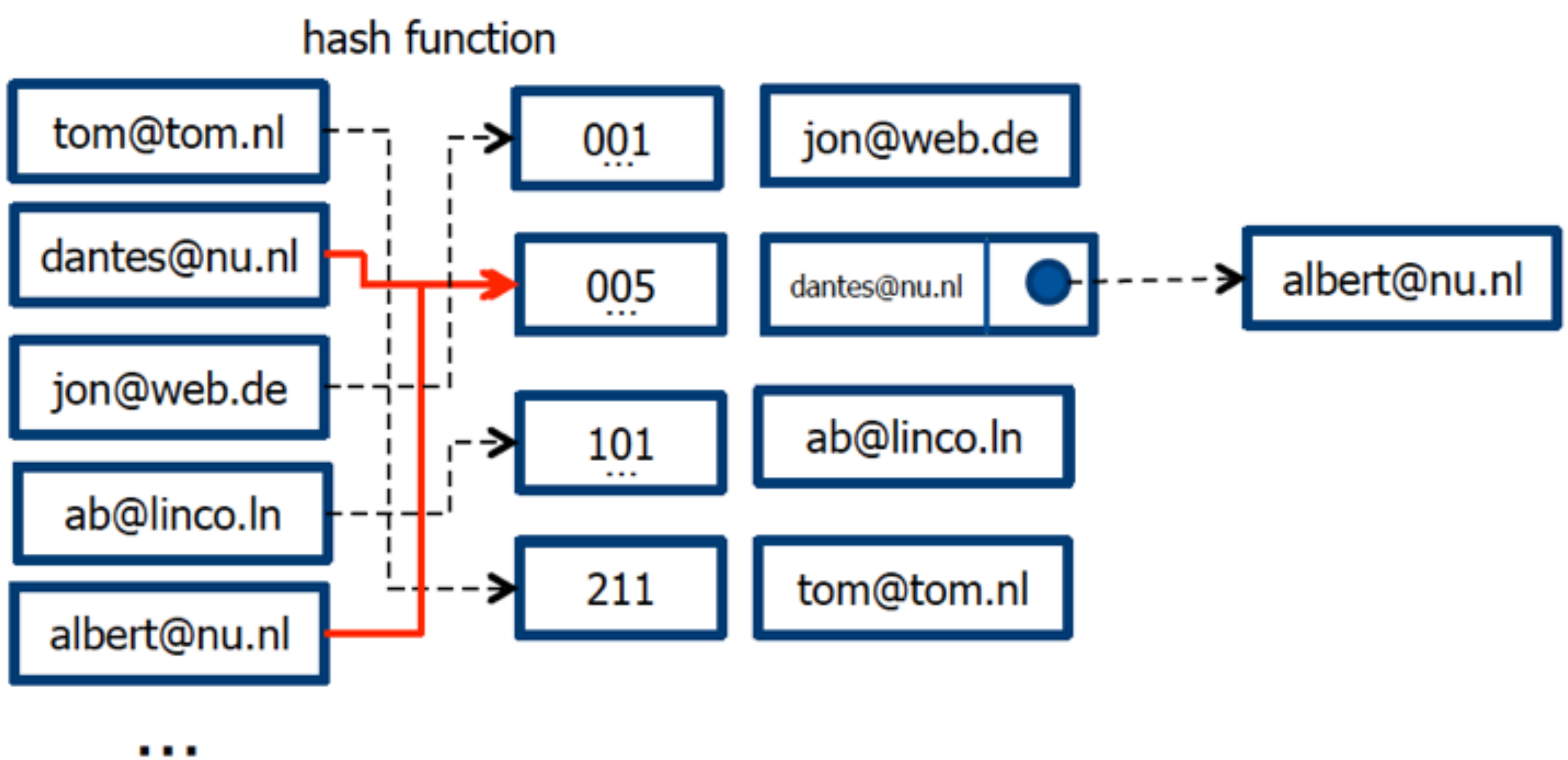
- A set W containing m values (e.g. IP addresses, email addresses, etc.)

- **Working memory of size n bit**

- **Goal**: data structure that allows efficient checking whether the next element in the stream is in W
  - return `TRUE` **with probability 1** if the element is indeed in W
  - return `FALSE` **with high probability** if the element is nto in W

A reminder: hash functions

- Each element is hashed into an integer (avoid hash collisions if possible)



Bloom filter

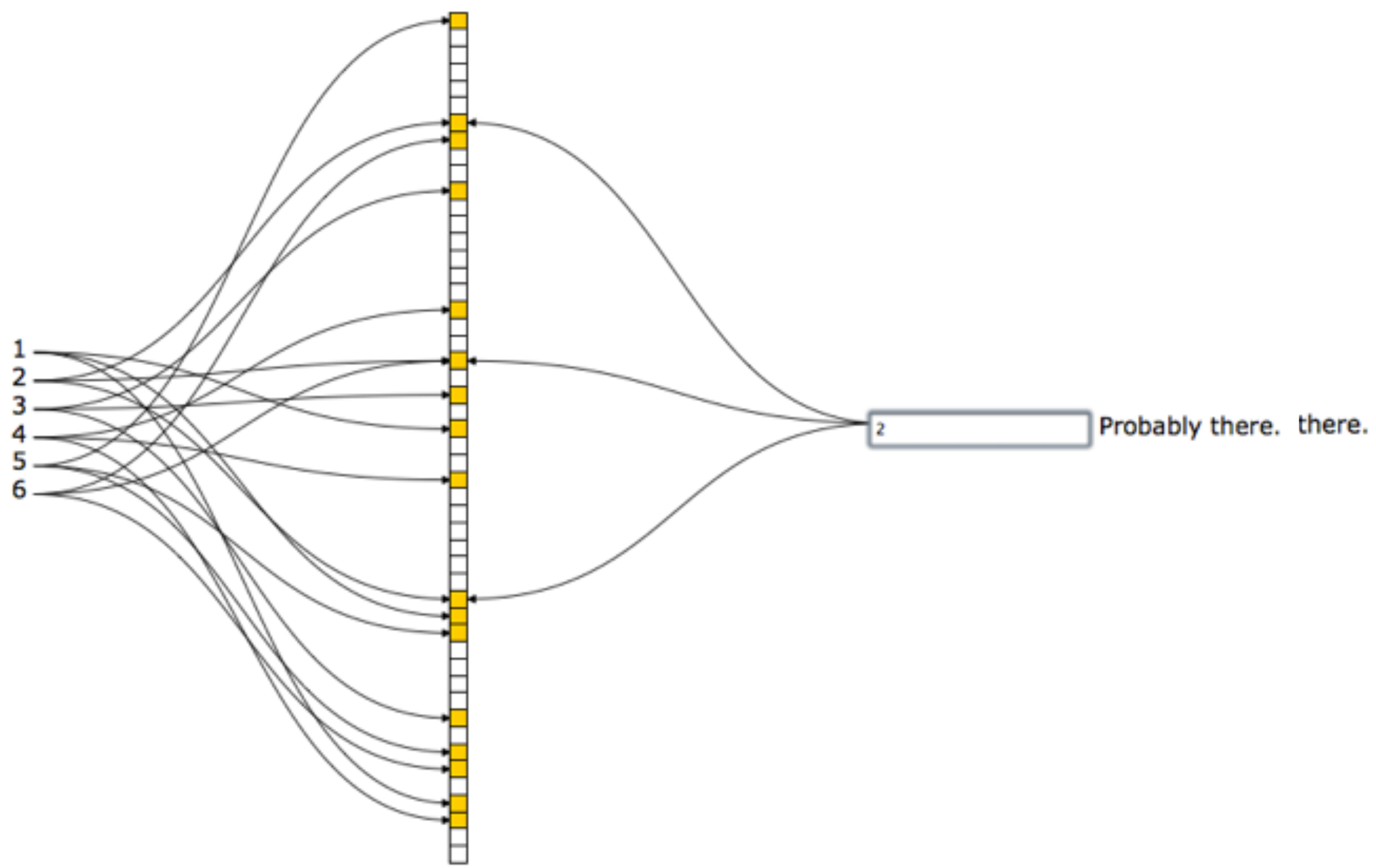
- **Given**
  - A set of hash functions $\{h_1, h_2, \ldots, h_k\}, h_i : W \rightarrow [1,n]$
  - A bit vector of size n (initialized to **0**)
- To **add** an element to W:
  - Compute $h_1(e), h_2(e), \ldots, h_k(e)$
  - Set the corresponding bits in the bit vector to 1

usually done once in bulk with few updates

- To **test** whether an element is in W:
  - Compute $h_1(e), h_2(e), \ldots, h_k(e)$
  - Sum up the returned bits
  - Return TRUE if sum=k, FALSE otherwise

operation on the data stream

Bloom filter: an online demo



http://www.jasondavies.com/bloomfilter/

Bloom filter: element testing

- **Case 1**: the element is in W
  - $h_1(e), h_2(e), \ldots, h_k(e)$ are all set to 1
  - `TRUE` is returned with probability 1

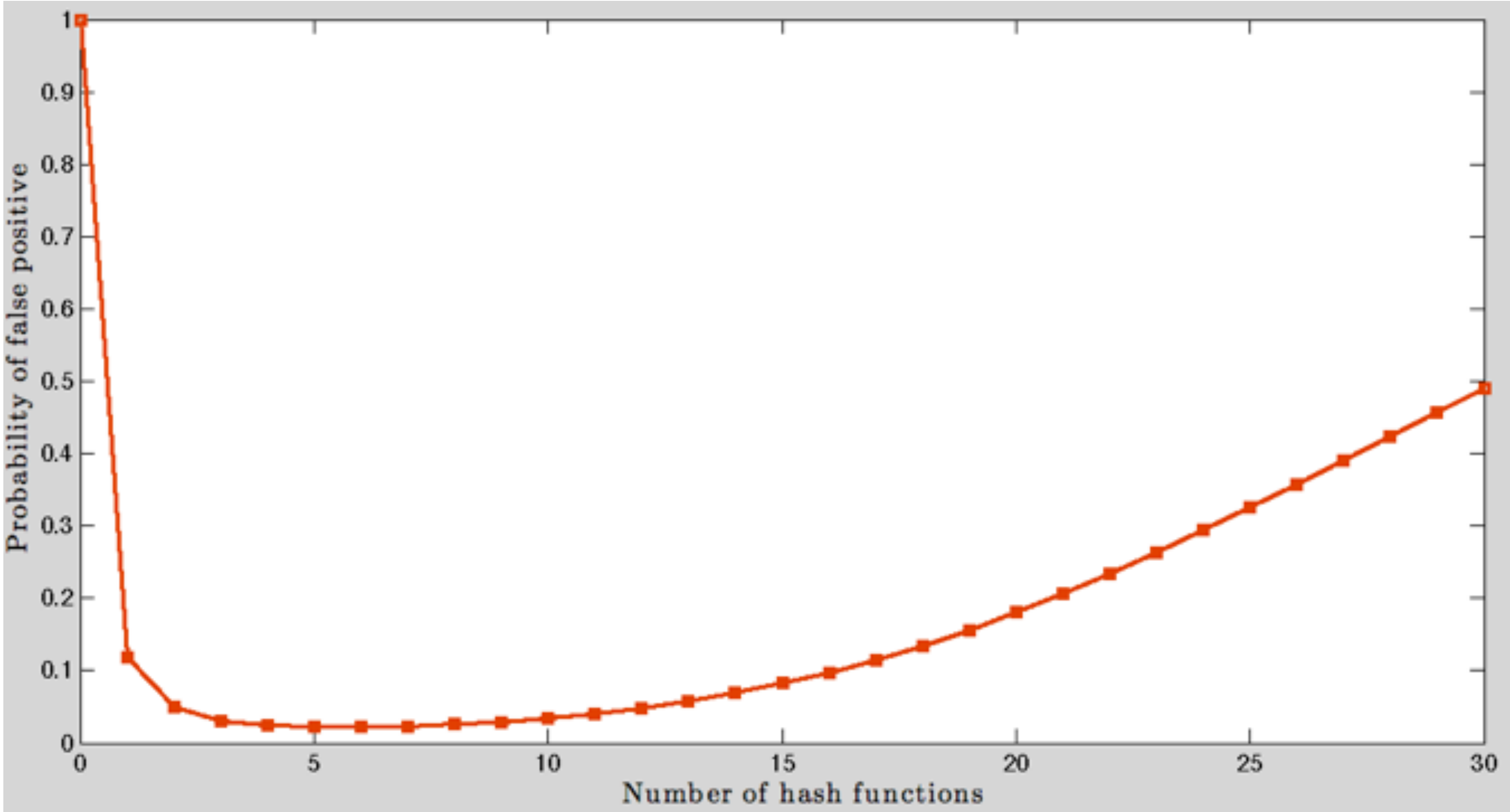
- **Case 2**: the element is not in W
  - `TRUE` is returned if due to some other element all hash values are set

$$
\begin{aligned}
P(BV_j \text{ set after } m \text{ inserts}) &= 1 - P(BV_j \text{ not set after } m \text{ inserts}) \\
&= 1 - P\left(BV_j \text{ not set after } k \times m \text{ hashes}\right) \\
&= 1 - \left(1 - \frac{1}{n}\right)^{k \times m}
\end{aligned}
$$

$$
P(\mathit{false\ positive}) = \left(1 - \left(1 - \frac{1}{n}\right)^{km}\right)^{k}
$$

Bloom filter: how many hash functions are useful?

- Example: m = 10^9 whitelisted IP addresses and n=8x10^9 bits in memory



Bloom filter tricks

- Union of two Bloom filters of the same type in terms of hash functions and bits
  - OR the two bit vectors

- To half the size of a Bloom filter with a filter size the power of 2
  - OR first and second half together
  - When hashing the higher order bit can be masked

- Bloom filter deletions?
  - Is it possible to take out a value?
  - (No, not in the standard setup); solution: counting bloom filters
  - Instead of bits, use counters that increment/decrement

Summary

- Data streaming
- Sampling approaches
- Frequency counters
- Bloom filters

THE END