Strata
DATA CONFERENCE

# Approximation Data Structures for Streaming Data Applications

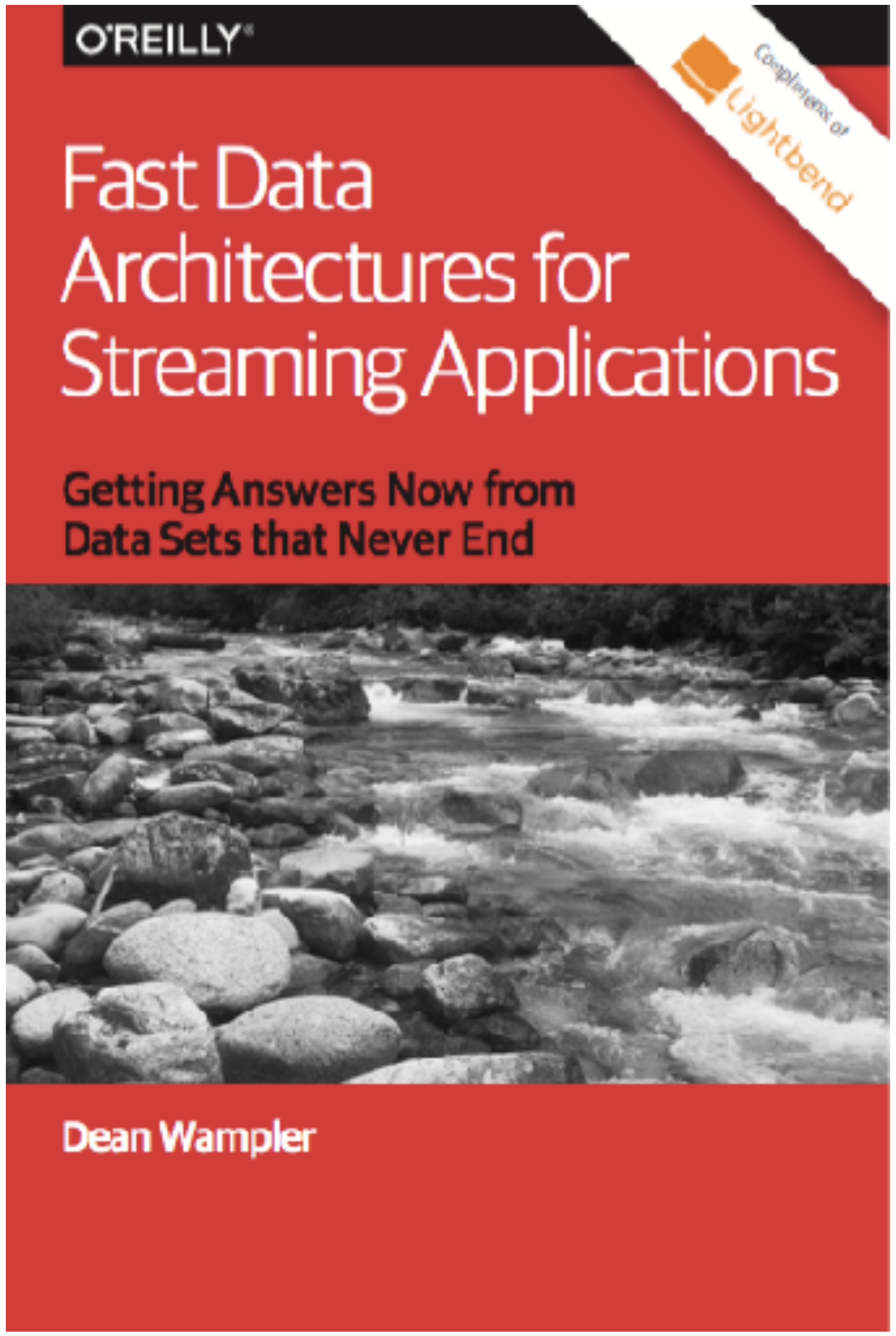
**Debasish Ghosh**
**(@debasishg)**

Lightbend

Semi-Structured
Unstructured data
Petabytes
analysis
content
BIG
DATA
Volume
Useful
Metadata
AMP
V
People driven
Decision making
Data
Unstructured
structured
Zettabyte
Framework
Smart content
database
VAST
Concept extraction
Text analytics
Semantic Metadata
Structured

## Big Data => Fast Data

- Volume
- Variety
- Velocity


O'REILLY

Compliments of Lightbend

Fast Data Architectures for Streaming Applications

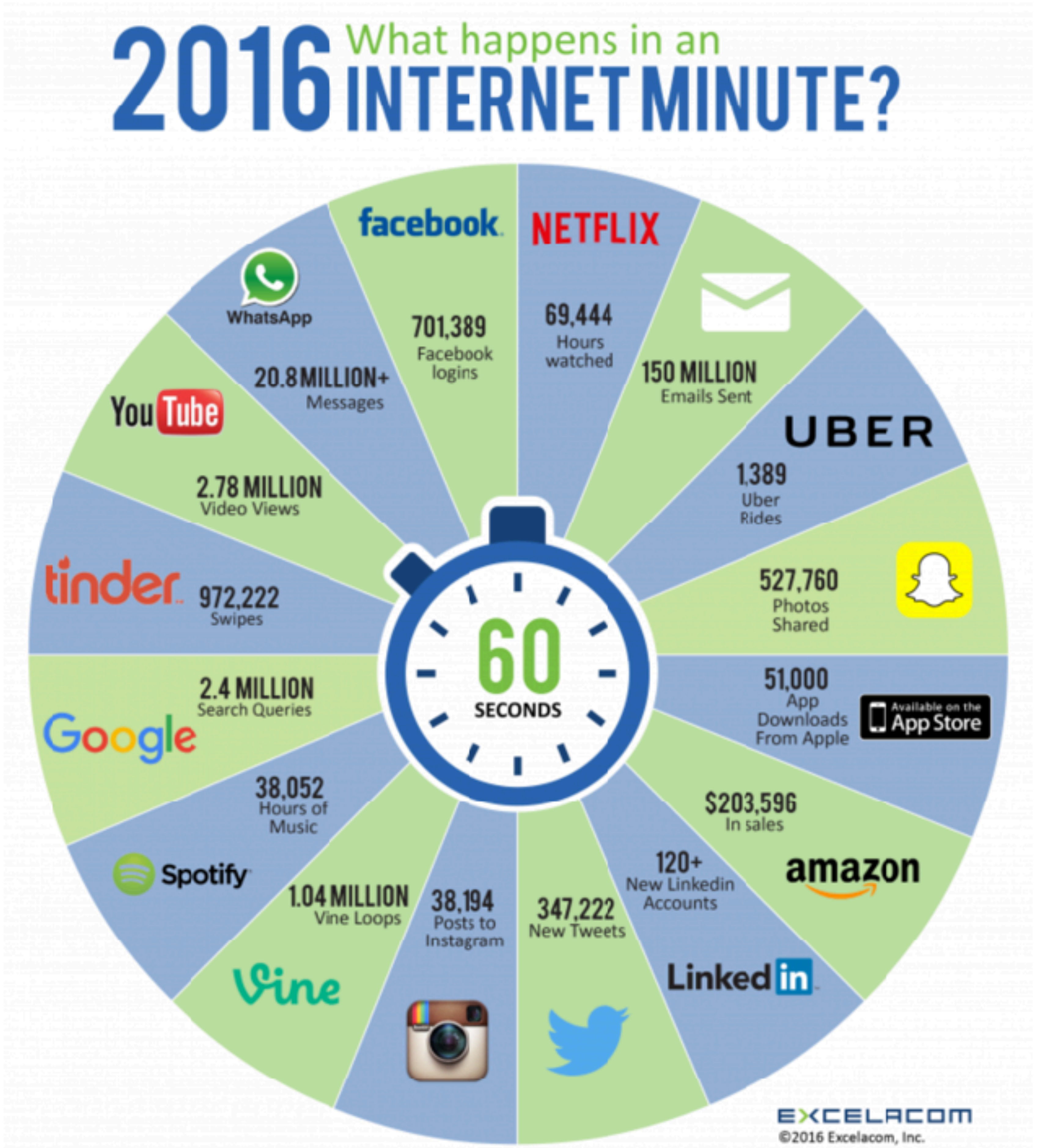
Getting Answers Now from Data Sets that Never End

Dean Wampler

https://whatsthebigdata.com/2016/04/22/what-happens-on-the-internet-in-60-seconds/

**What Happens on the Internet in 60 Seconds**

Posted on April 22, 2016



Lightbend

## A fundamental change in the shape of data that we need to process

Credit: http://www.doc.govt.nz/nature/habitats/freshwater/

## Data Stream Model

## Data Stream Model

- So big that it doesn't fit in a single computer (unbounded)

# Data Stream Model

- So big that it doesn’t fit in a single computer (unbounded)
- So big that a polynomial running time isn’t good enough

## Data Stream Model

- So big that it doesn’t fit in a single computer (unbounded)
- So big that a polynomial running time isn’t good enough
- An algorithm processing such data can only access data in a single pass

Lightbend

## Data Stream Model

- So big that it doesn’t fit in a single computer (unbounded)
- So big that a polynomial running time isn’t good enough
- An algorithm processing such data can only access data in a single pass
- And yet data needs to be processed with a low latency feedback loop with the consumers

Lightbend

# Motivating Use Cases

- Monitor events when a user visits a web site. Event streams drive analytics and generate various metrics on user behaviors
- Traffic monitoring in network routers based on IP addresses - explore heavy hitters (top traffic intensive IP addresses)
- Processing financial data streams (stock quotes & orders) to facilitate real time decision making
- Online clustering algorithms - similarity detection in real time
- Real time anomaly detection on data streams

Lightbend

## Algorithm Ideas

- Continuous processing of unbounded streams of data
- Single pass over the data
- Memory and time bounded - sublinear space
- Queries may not have to be served with hard accuracy - some affordance of errors allowed

## Can we have a deterministic and/or exact algorithm that meets all of these requirements ?

Lightbend

# Distinct Elements Problem

- **Input:** Stream of integers $i_1, \ldots, i_m \in [n]$
- **Where:** [n] denotes the Set { 1, 2, .. , n }
- **Output:** The number of distinct elements seen in the stream
- **Goal:** Minimize space consumption

# Distinct Elements Problem

- **Solution 1:** Keep a bit array of length n, initialized to all zeroes. When you see i in the stream, set the ith bit to 1.

  - **Space required:** $n$ bits of memory

# Distinct Elements Problem

- **Solution 1:** Keep a bit array of length n, initialized to all zeroes. When you see i in the stream, set the ith bit to 1.
  - **Space required:** $n$ bits of memory

- **Solution 2:** Store the whole stream in memory explicitly
  - **Space required:** $\lceil m \log_2 n \rceil$ bits of memory

Can we have a deterministic and/or exact algorithm that beats this space bound of $min\{n, \lceil mlog_2 n \rceil\}$ ?

Lightbend

# Sublinear with Deterministic & Exact - Possible ?

- Each element of the stream can be represented by $n$ bits. The entire stream can then be mapped to $\{0, 1\}^n$
- Suppose a deterministic & exact algorithm exists that uses $s$ bits of space where $s < n$
- Then there must exist some mapping from $n$-bit strings to $s$-bit strings i.e. $\{0,1\}^n$ to $\{0,1\}^s$
- And this mapping has to be injective (no 2 elements of the domain can map to the same element in co-domain)
- It can be proved that such a mapping does not exist (there cannot be an injective mapping from a larger set to a smaller set)

There exists NO deterministic and/or exact algorithm that implements *Distinct Elements* problem in sublinear space

## Randomized & Approximate

# Randomized & Approximate

- **Estimators** - the algorithm returns an estimator in response to a query

## Randomized & Approximate

- **Estimators** - the algorithm returns an estimator in response to a query

unbiased ?

variance ?

# Randomized & Approximate

- **Estimators** - the algorithm returns an estimator in response to a query

- **Error bound** - f(x) is accurate up to a certain bound ( $\epsilon$ bound )

# Randomized & Approximate

- **Estimators** - the algorithm returns an estimator in response to a query
- **Error bound** - f(x) is accurate up to a certain bound ( $\epsilon$ bound )
- **Confidence of accuracy** - probability that the estimator will be within the above bound ( $1-\delta$ )

Lightbend

## $\epsilon - \delta$ *Approximation*

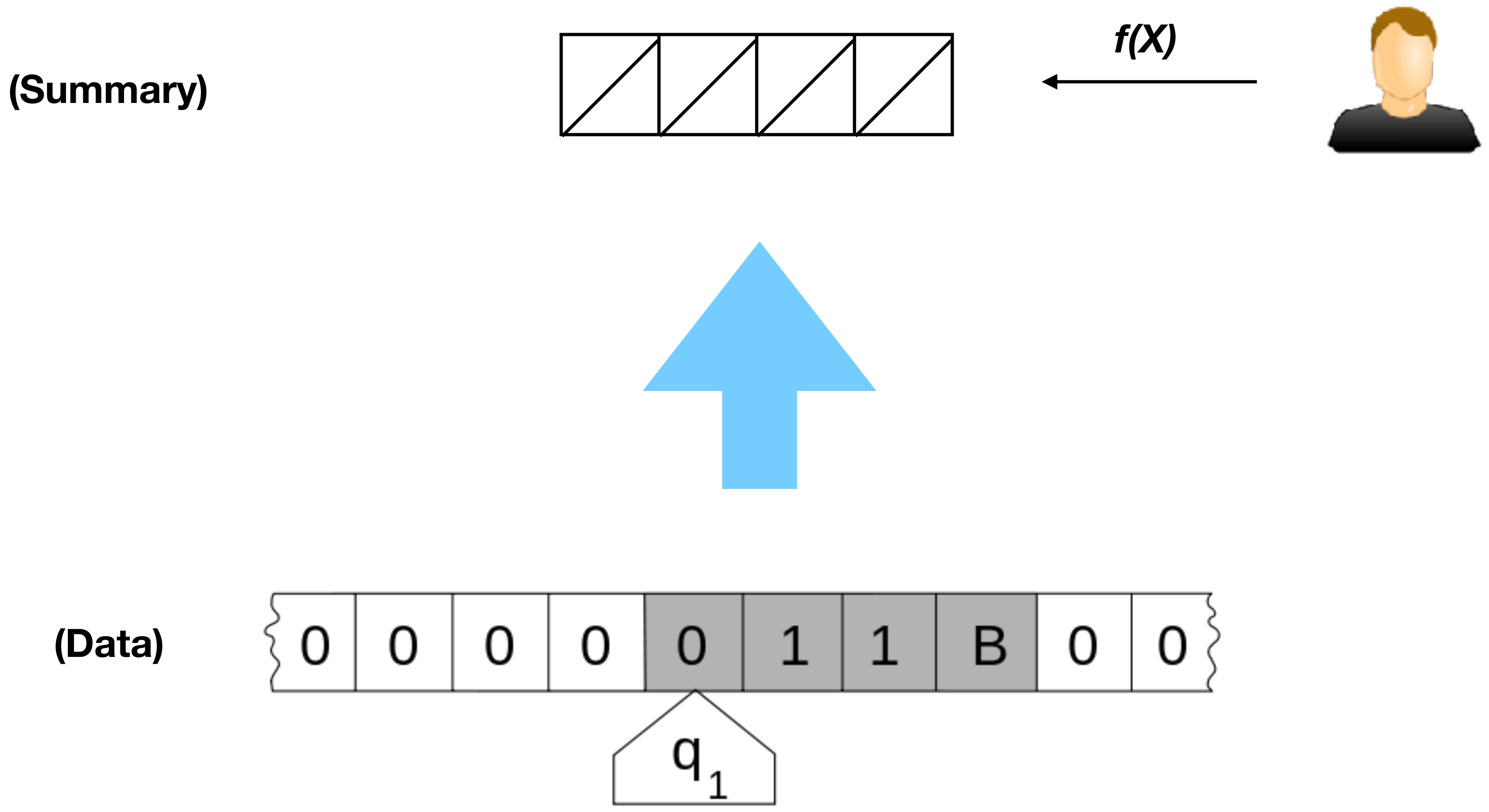
## $\epsilon - \delta\ Approximation$

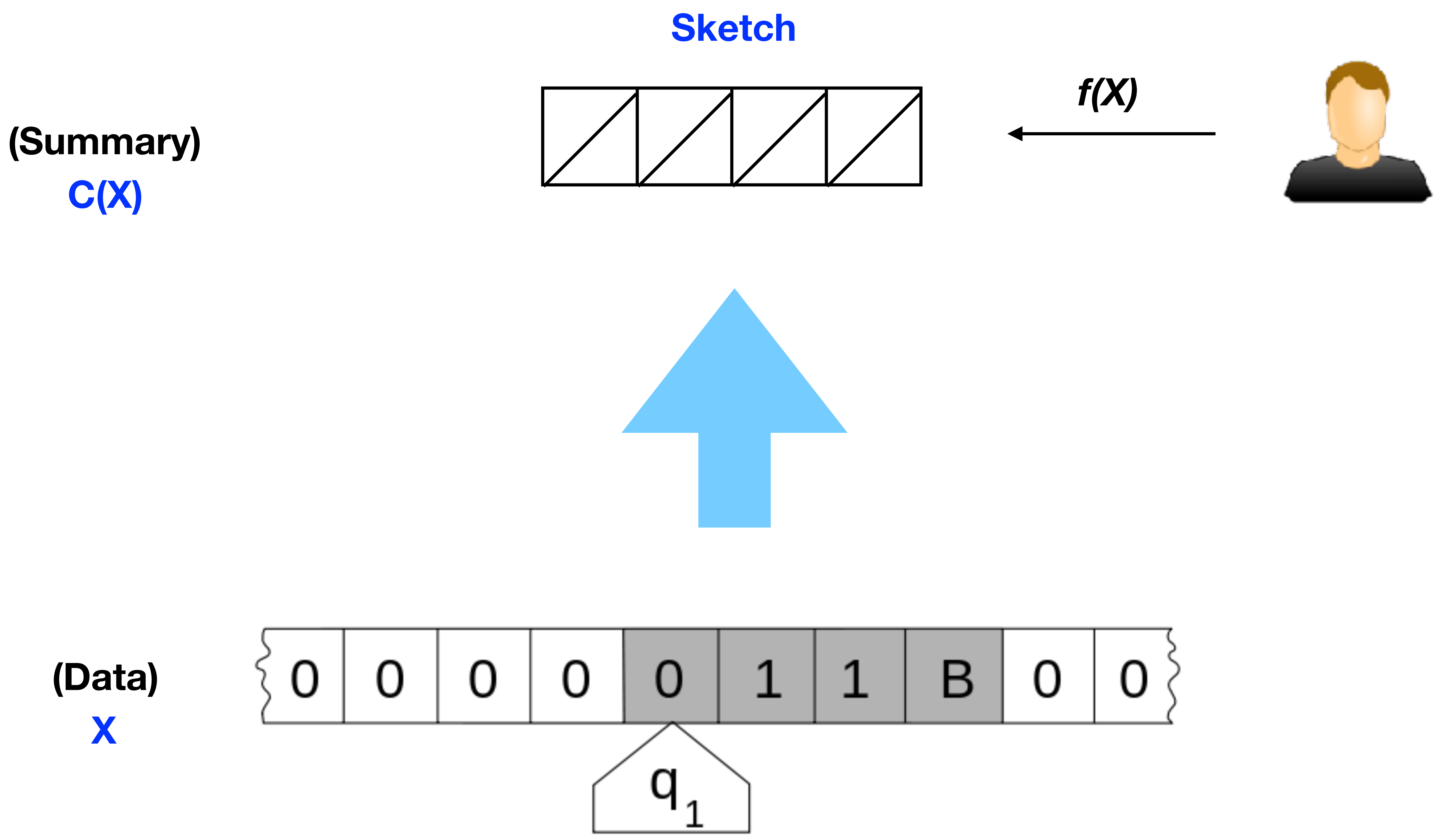
**Accuracy within** $\pm\epsilon$ **bounds with a failure probability of** $\delta$

## $\epsilon - \delta$ *Approximation*

**Accuracy within $\pm\epsilon$ bounds with a failure probability of $\delta$**

$$\mathbb{P}(\,|\,\tilde{n} - n\,| > \epsilon n) < \delta$$


(Summary)
f(X)
(Data)
0 0 0 0 0 1 1 B 0 0
q1


Sketch
(Summary)
C(X)
f(X)
(Data)
X
0 0 0 0 0 1 1 B 0 0
q1

- A **Sketch** C(X) of some data set X with respect to some function f is a **compression** of X that allows us to compute, or approximately compute f(X), **given access only to C(X)**

Lightbend

$$f(X, Y) = \sum_{z \in X \cup Y} z$$

Alice

Data set X, which is
a list of Integers

Bob

Data set Y, which is
a list of Integers

Lightbend

$$f(X, Y) = \sum_{z \in X \cup Y} z$$

Alice

Data set X, which is a list of Integers

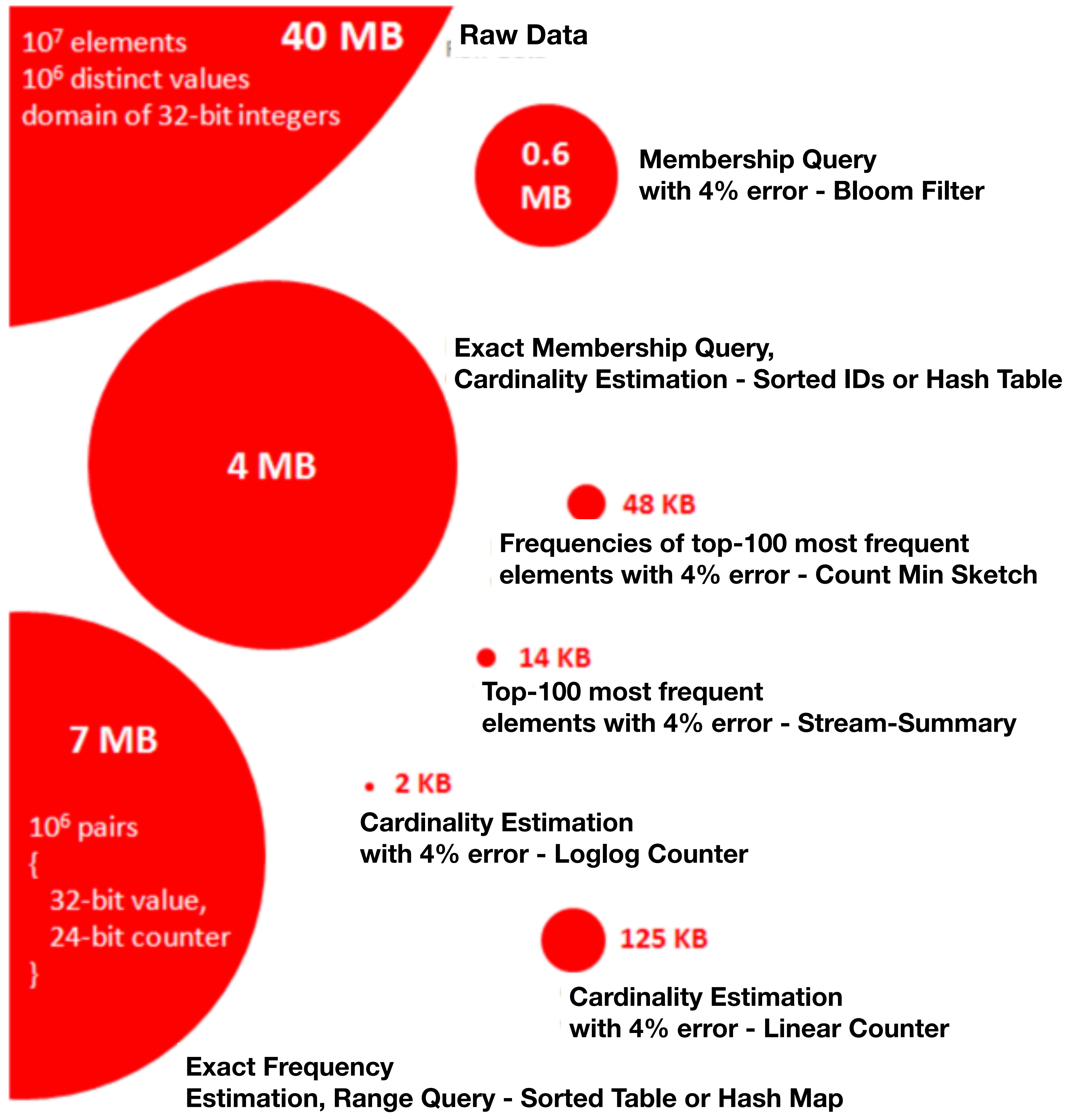
Maintain Sketch of X as the running sum of the integers

Bob

Data set Y, which is a list of Integers

Maintain Sketch of Y as the running sum of the integers

## Show me some data!



Source: https://highlyscalable.wordpress.com/2012/05/01/probabilistic-structures-web-analytics-data-mining/

## A Simple Counter

- Use Case - Monitor a stream of events
- At any point in time output (an estimate of) the number of events seen so far.You may have to report from multiple counters aggregated by event types
- Idea is to beat $O(\log_2 n)$ space. Any trivial algorithm can implement this using $\log_2 n$ bits

## $\epsilon - \delta$ *Approximation*

- Using a suitable ***sketch***, there exists an algorithm that returns an estimator of the counter within a bound of $k(1 \pm \epsilon)$
- and a small probability of failure $\delta$

# Approximate Counting (Morris ’78)

1. *Initialize* $X \longleftarrow 0$.
2. *For each update, increment* $X$ *with probability* $1/2^X$.
3. *For a query, output* $\tilde{n} = 2^X - 1$.

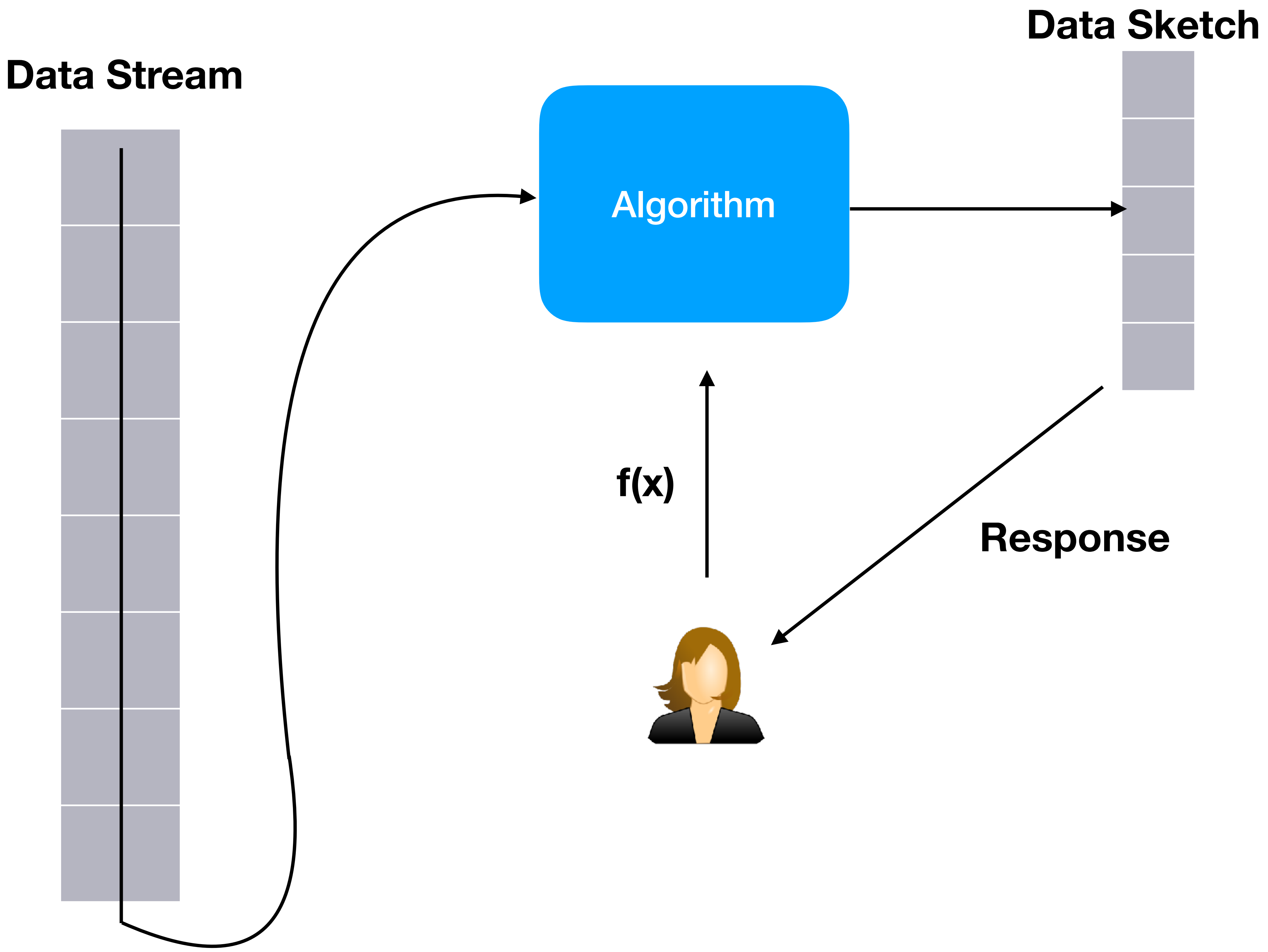
$$\mathbb{P}(\,|\,\tilde{n} - n\,| > \epsilon n) < \delta$$

The steps to analyze this algorithm generalize beautifully to all approximation data structures used to handle streaming data

# Generalization steps ..

- Compute the expected value of the estimator. In [Morris '78] we have $\mathbb{E}[2^X - 1] = n$
- Compute the variance of the estimator. In [Morris '78] we have $var[2^X - 1] = O(n^2)$
- Using median trick, establish $\epsilon - \delta\ Approximation$

# Sketch based Query Model

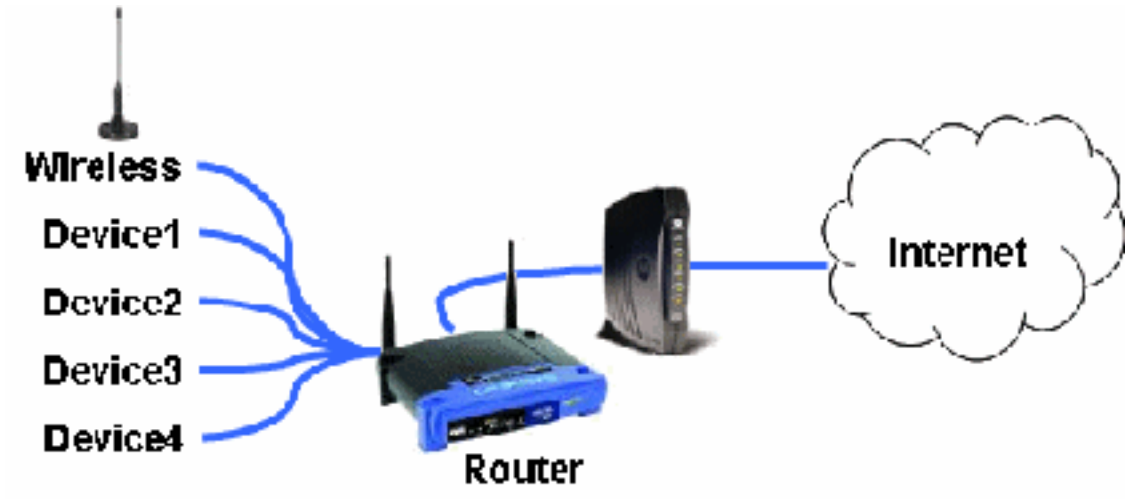


# Use Case

- Continuous stream of IP addresses hitting a router

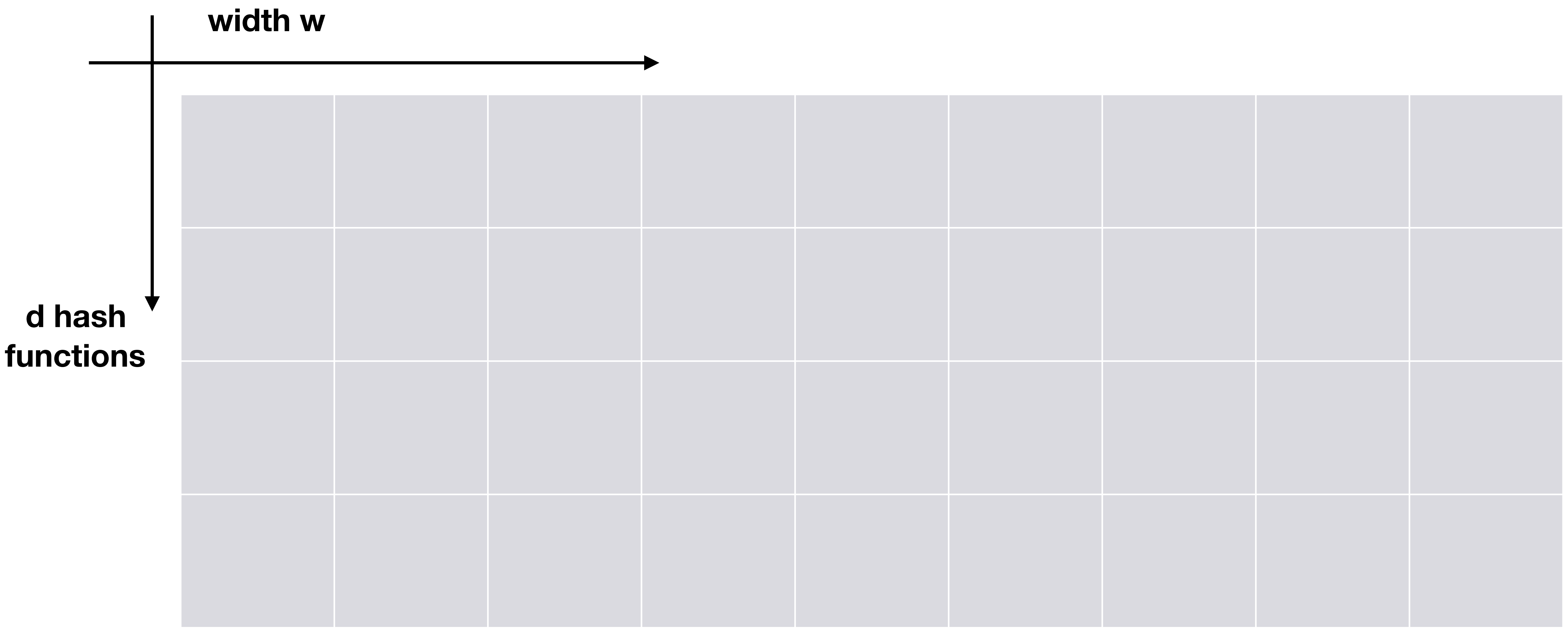
- Updates of the form (i, $\Delta$), which means the count of IP address i has to increase by by $\Delta$

- Want an estimate of how many times IP address i has hit the router at any point in time (Frequency Estimation)



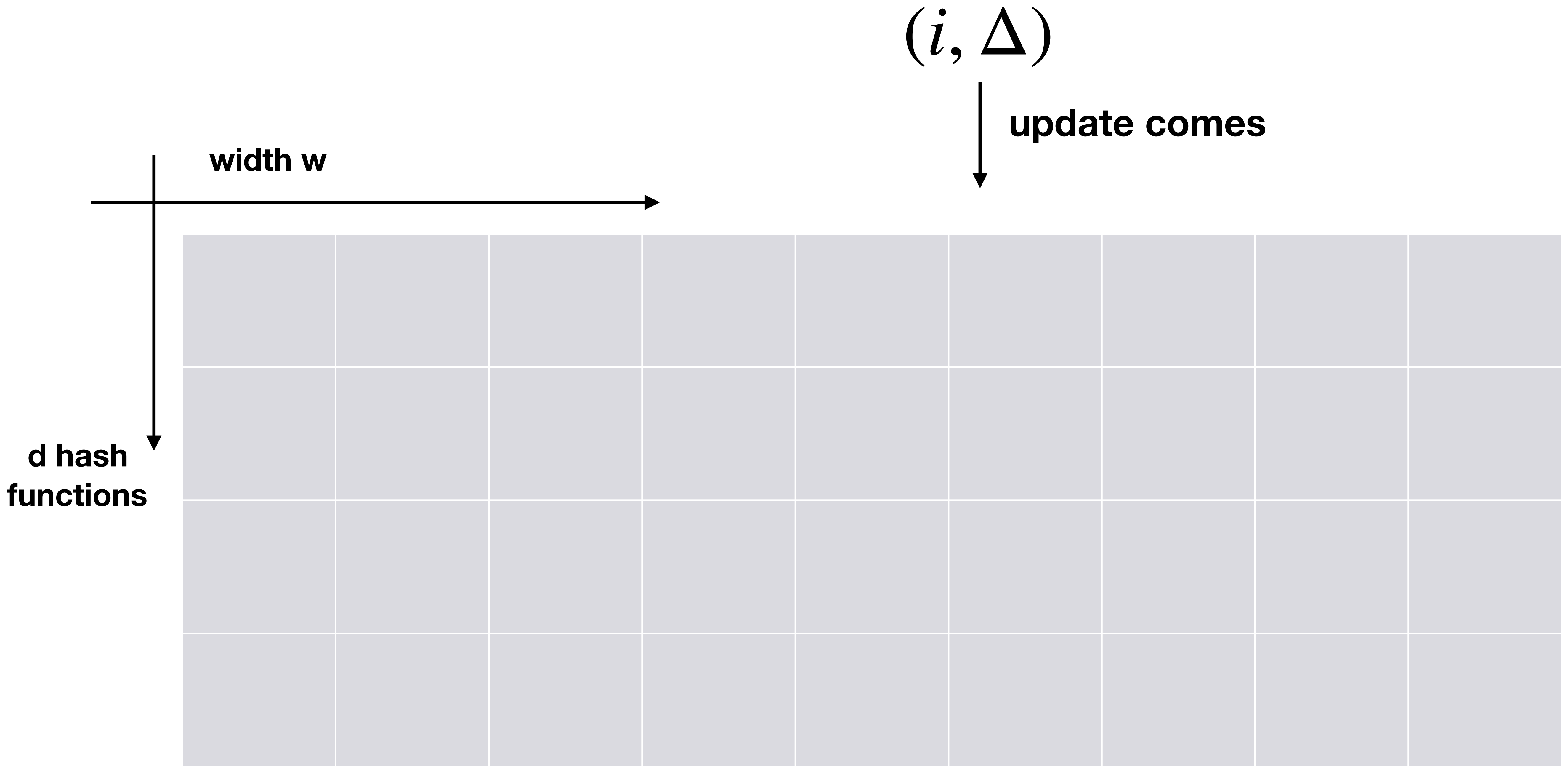
Credit: http://voipstuff.net.au/routers/

# Count Min Sketch

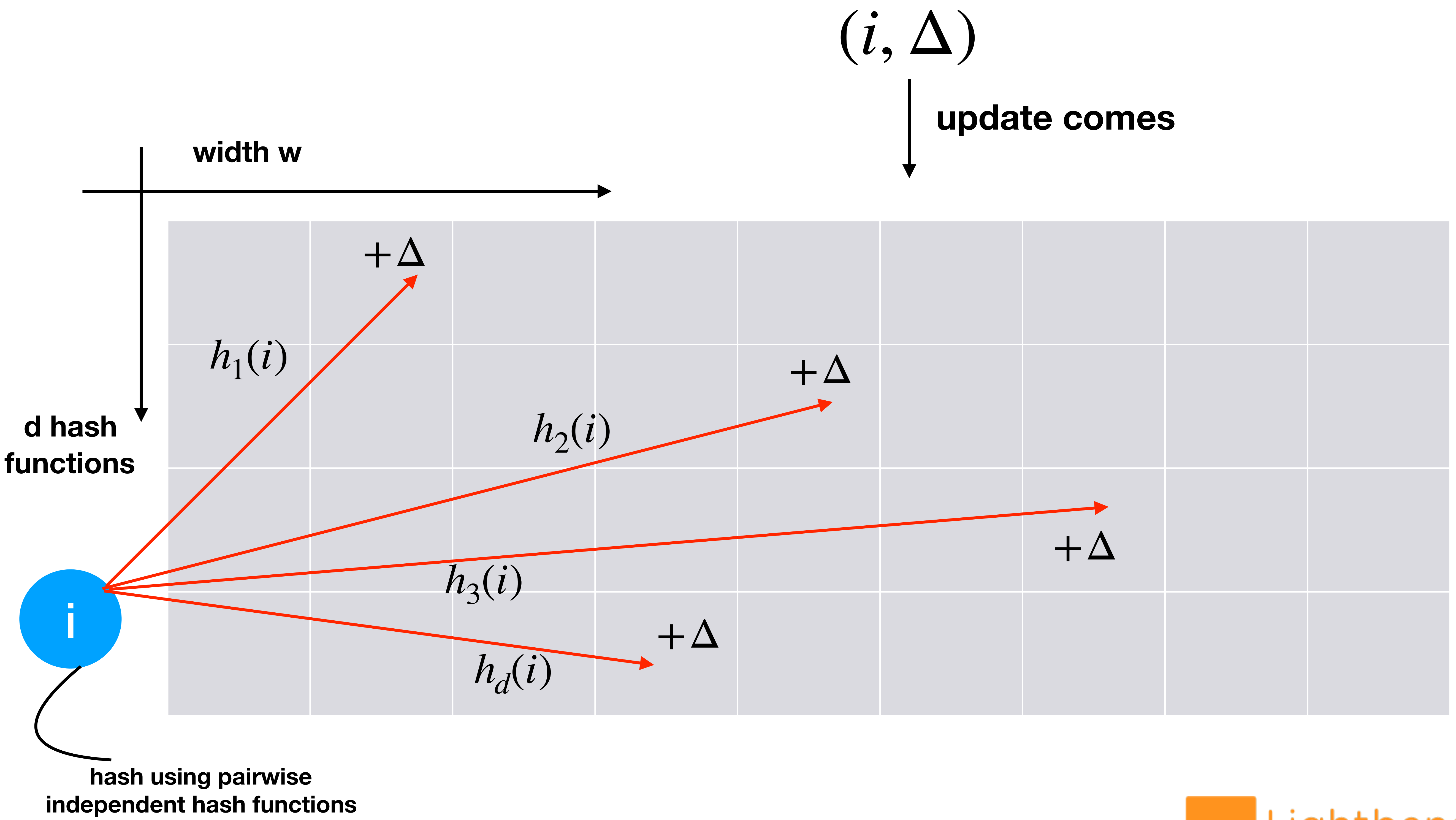


***An Improved Data Stream Summary: The Count-Min Sketch and its Applications***
**- Graham Cormode and S. Muthukrishnan (http://dimacs.rutgers.edu/~graham/pubs/papers/cm-full.pdf)**

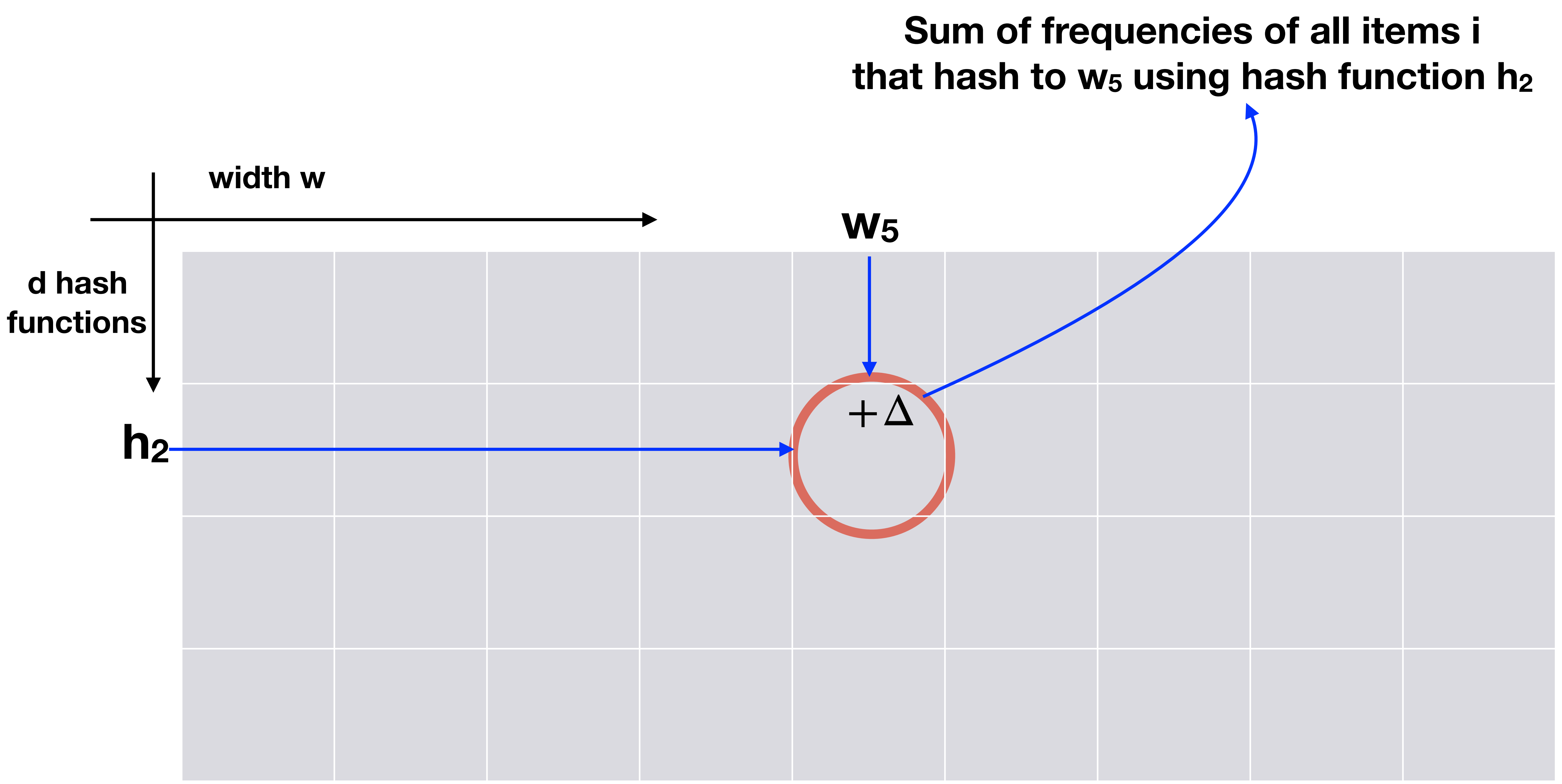
# Count Min Sketch



# Count Min Sketch

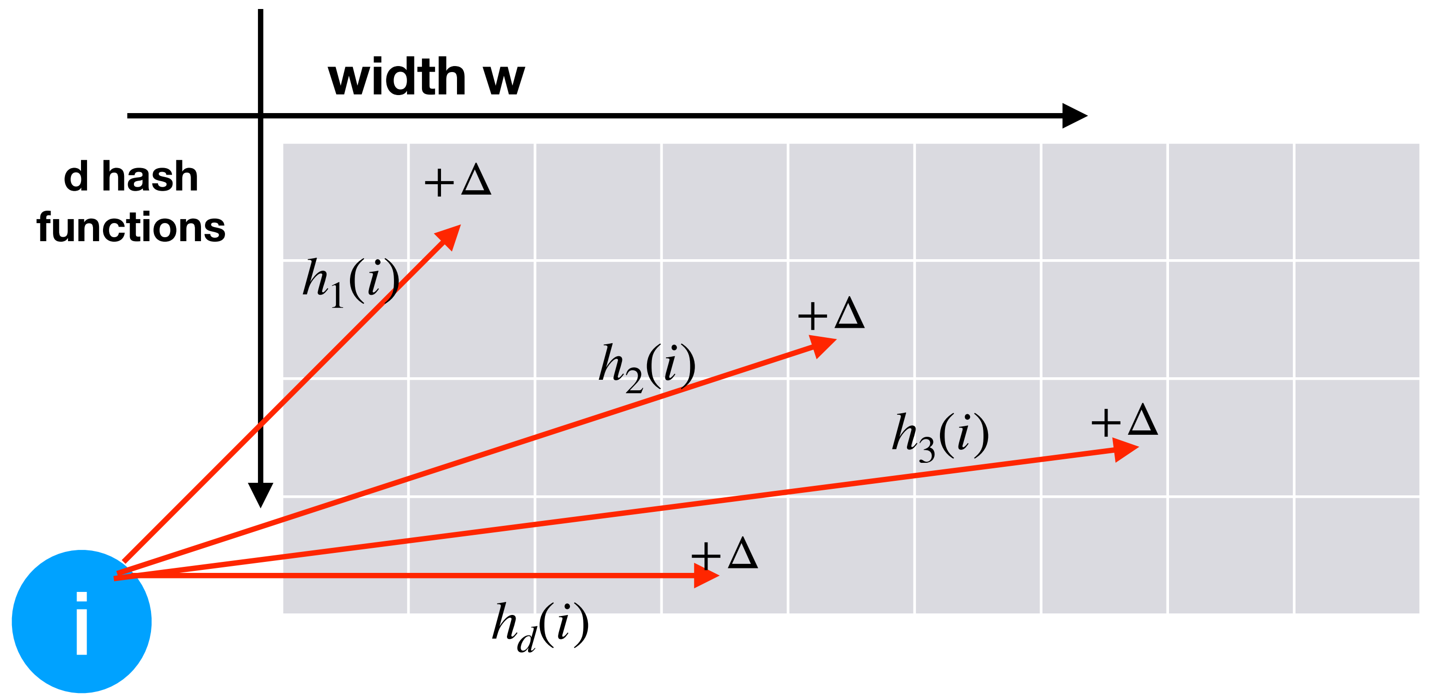


# Count Min Sketch



Sum of frequencies of all items i that hash to $w_5$ using hash function $h_2$
width w
d hash functions
$w_5$
$h_2$
$+\Delta$

# query(i)



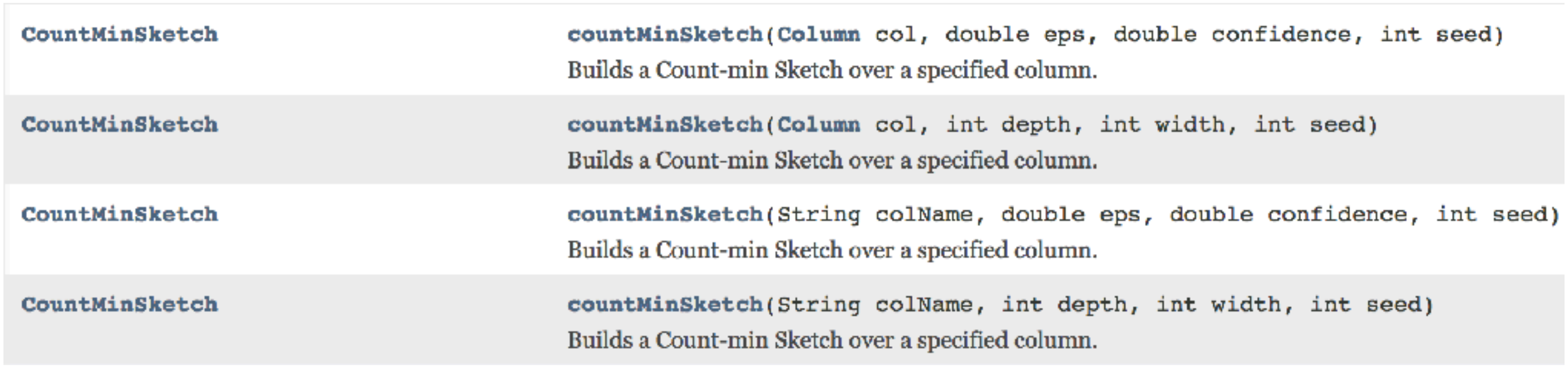
- Hash i using all d hash functions
- The results point to d cells in the table, each containing some frequency value
- Return the minimum of the d values as an estimate of query(i)

# Count Min Sketch

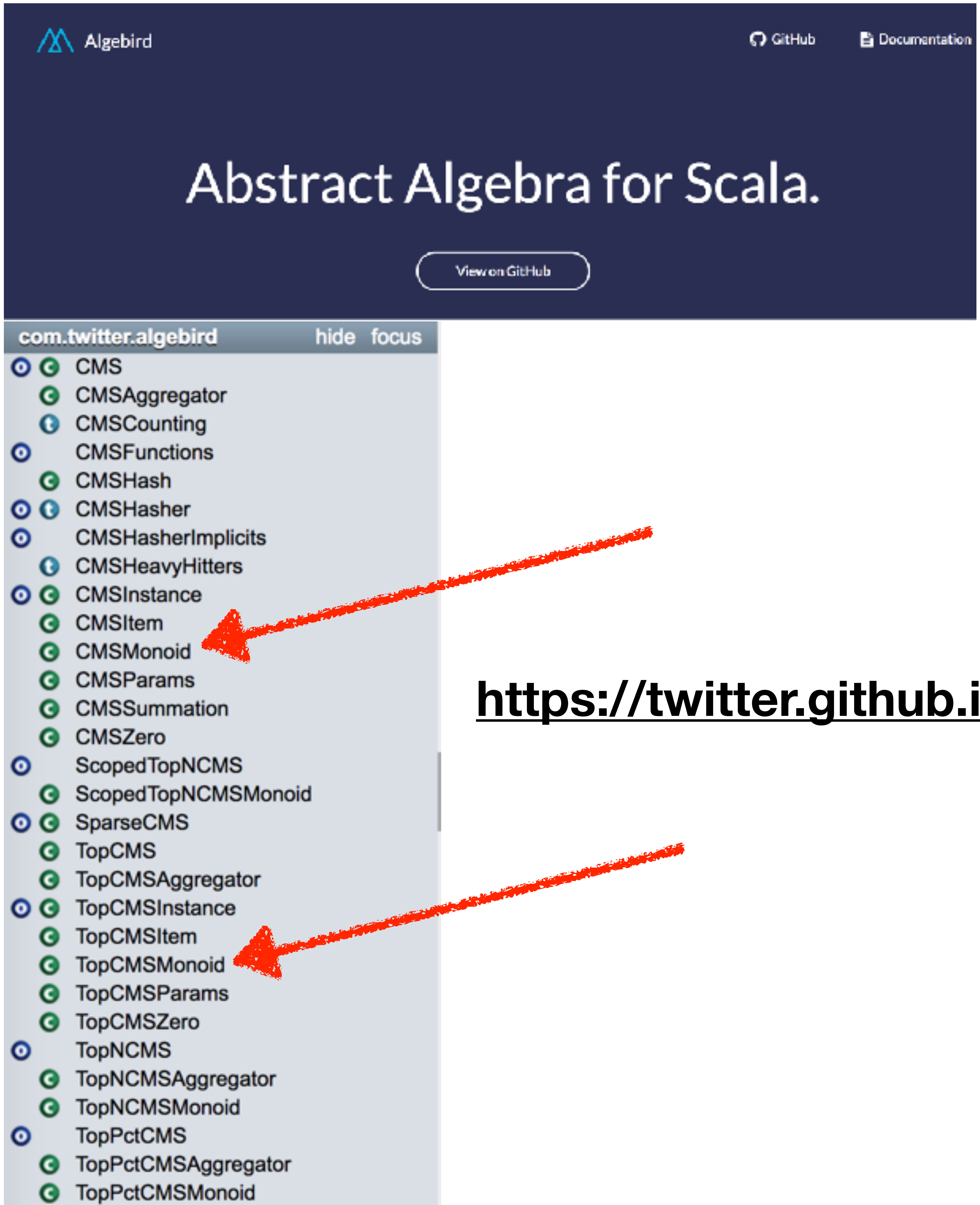
## Claim

1. $For\ \epsilon - point\ query\ with\ failure\ probability\ \delta\,.$
2. $query(i) = x_i \pm \epsilon \parallel x \parallel_1\ with\ prob \geq 1 - \delta\,.$
3. $Set\ \ w = \lceil 2/\epsilon \rceil\ \ and\ d = \lceil log_2(1/\delta) \rceil\,.$
4. $Space\ \ required\ \ is\ \ O(\epsilon^{-1} log_2(1/\delta)\,.$

# Count Min Sketch in Spark



| | |
|---|---|
| `CountMinSketch` | `countMinSketch(Column col, double eps, double confidence, int seed)`<br>Builds a Count-min Sketch over a specified column. |
| `CountMinSketch` | `countMinSketch(Column col, int depth, int width, int seed)`<br>Builds a Count-min Sketch over a specified column. |
| `CountMinSketch` | `countMinSketch(String colName, double eps, double confidence, int seed)`<br>Builds a Count-min Sketch over a specified column. |
| `CountMinSketch` | `countMinSketch(String colName, int depth, int width, int seed)`<br>Builds a Count-min Sketch over a specified column. |



CMS
CMSAggregator
CMSCounting
CMSFunctions
CMSHash
CMSHasher
CMSHasherImplicits
CMSHeavyHitters
CMSInstance
CMSItem
CMSMonoid
CMSParams
CMSSummation
CMSZero
ScopedTopNCMS
ScopedTopNCMSMonoid
SparseCMS
TopCMS
TopCMSAggregator
TopCMSInstance
TopCMSItem
TopCMSMonoid
TopCMSParams
TopCMSZero
TopNCMS
TopNCMSAggregator
TopNCMSMonoid
TopPctCMS
TopPctCMSAggregator
TopPctCMSMonoid

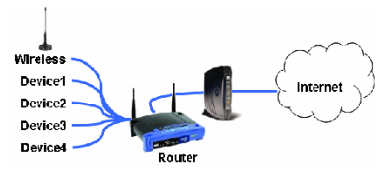
https://twitter.github.io/algebird/

# Algebra of a Monoid

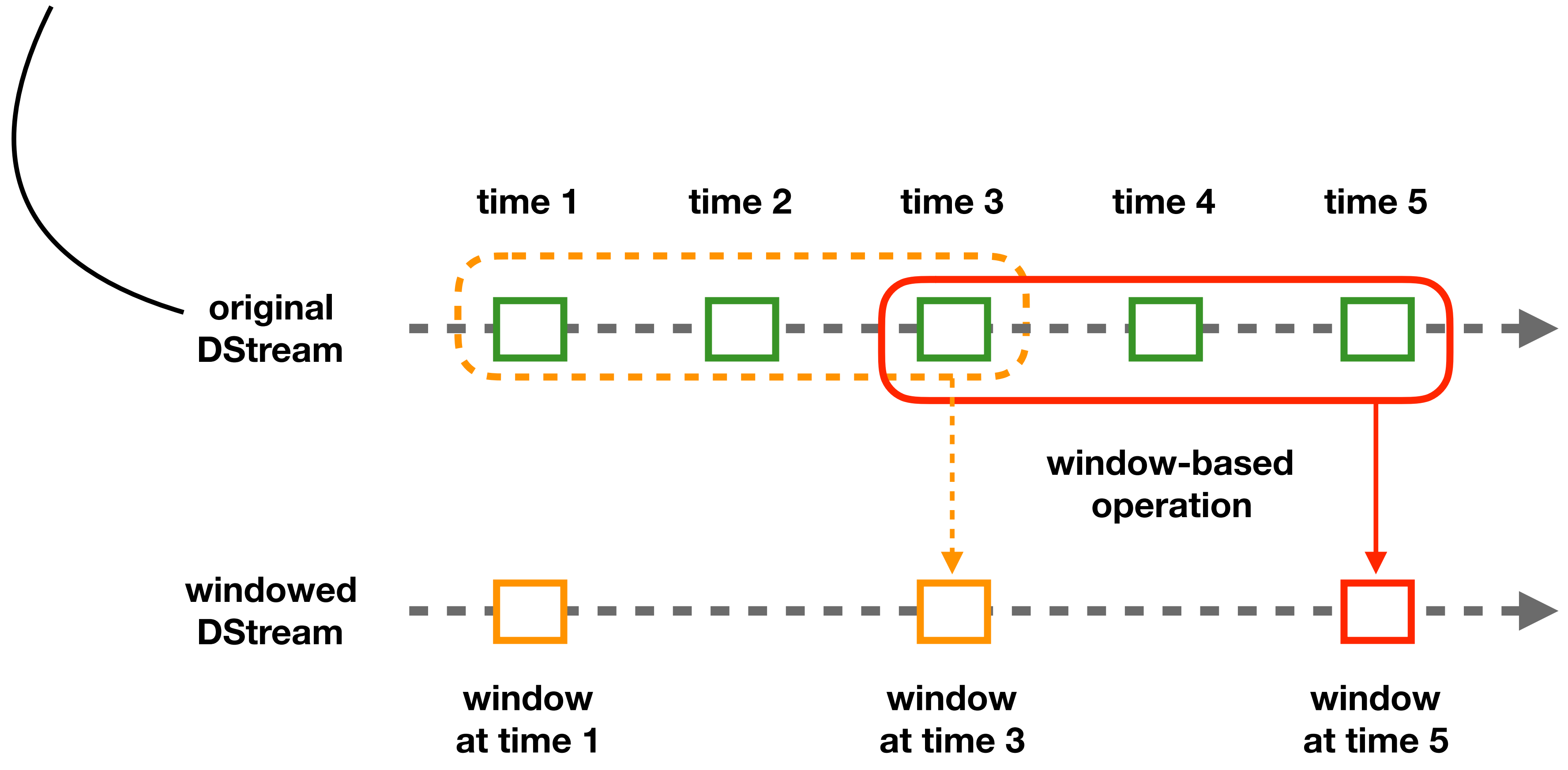
| | |
|---|---|
| given | $Set\,A$ |
| a binary operation | $\phi : A \times A \rightarrow A$ |
| associative | $for\ (a, b, c) \in A$ <br> $(a\ \phi\ b)\ \phi\ c\ = a\ \phi\ (b\ \phi\ c)$ |
| identity | $for\ (a, I) \in A$ <br> $a\ \phi\ I\ = I\ \phi\ a\ = a$ |

Lightbend

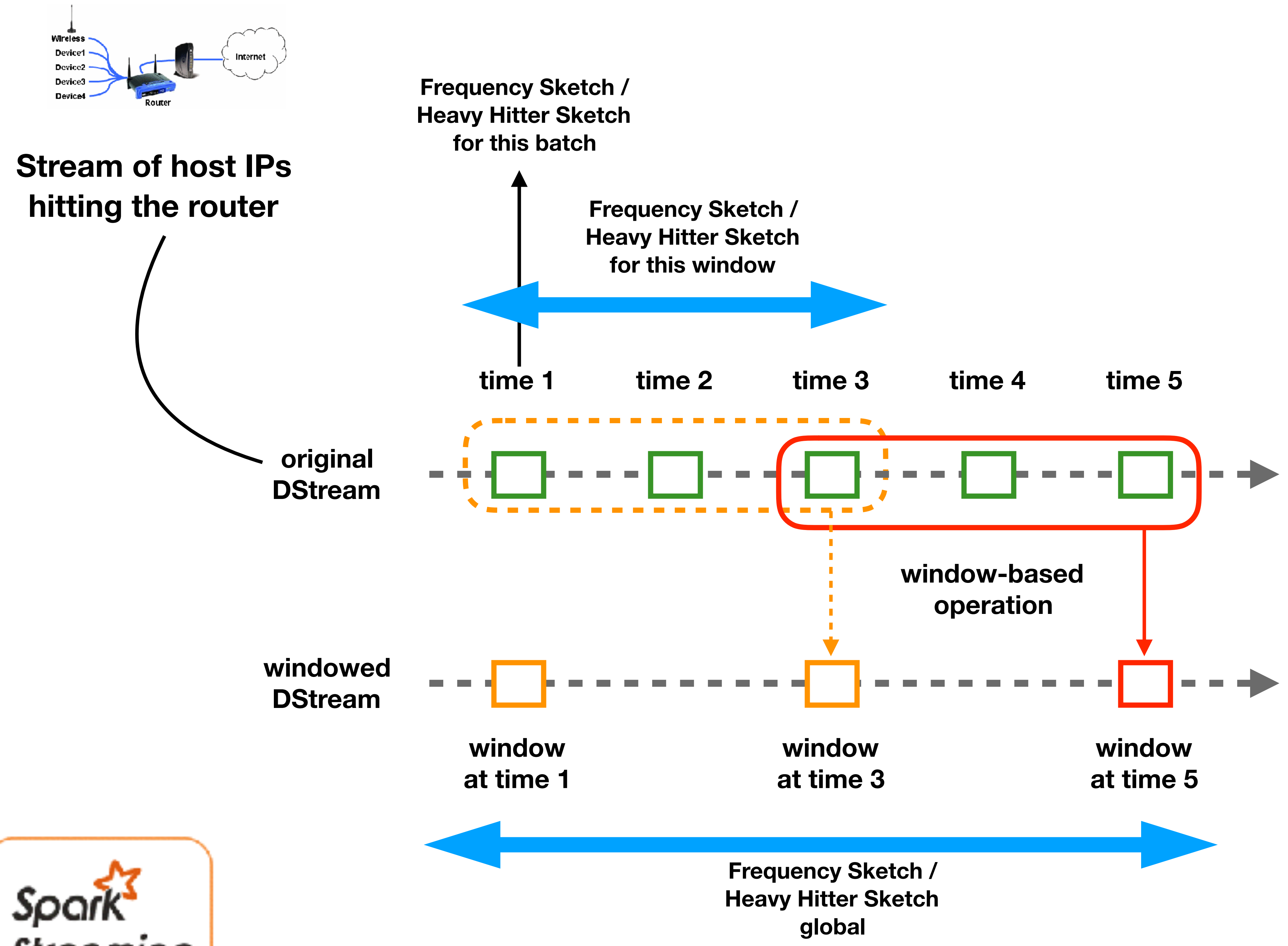
# CMS in the wild



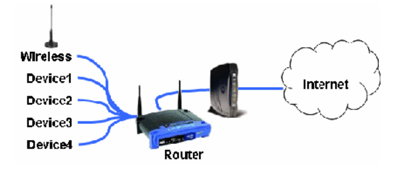
Stream of host IPs hitting the router

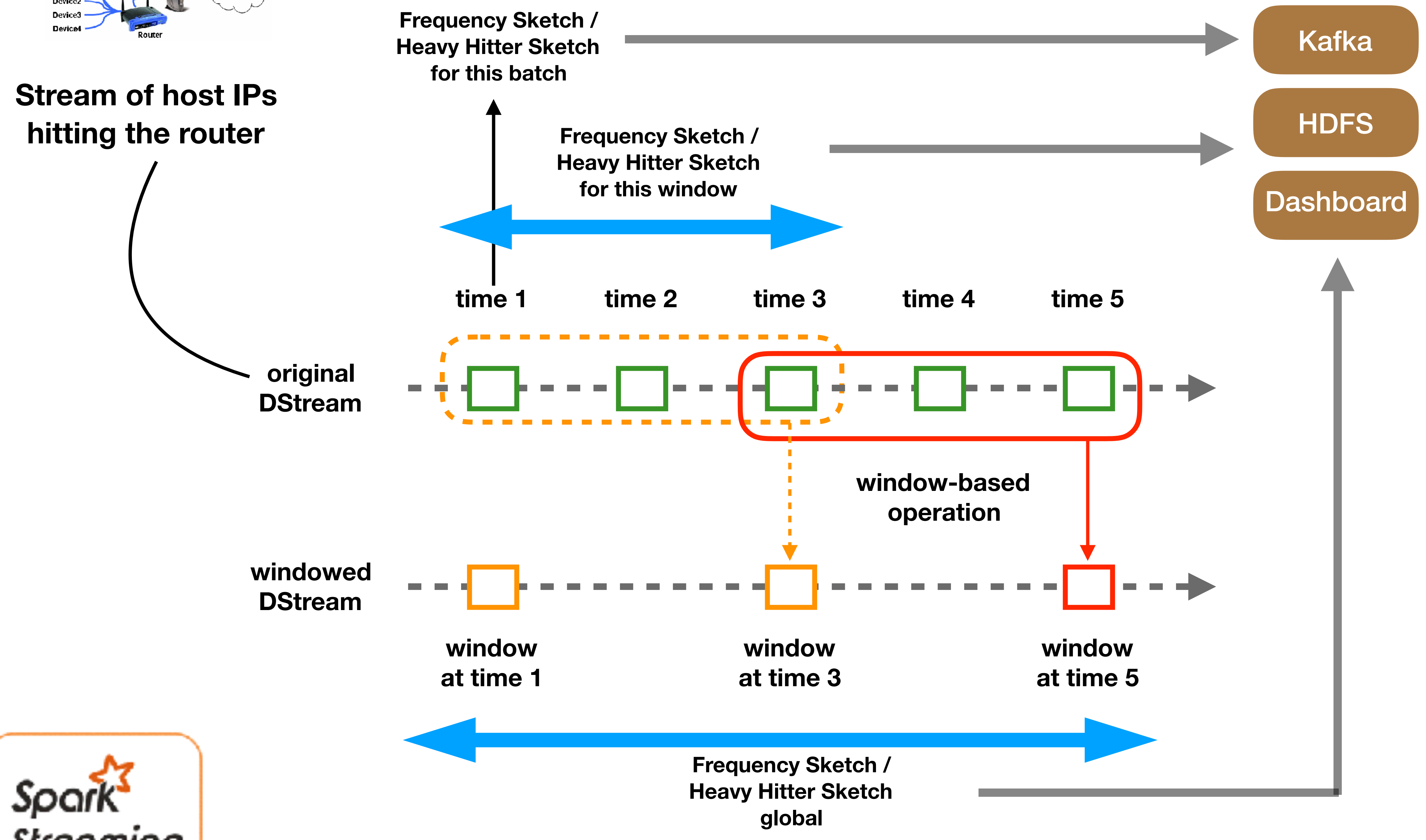


# CMS in the wild


Wireless
Device1
Device2
Device3
Device4
Router
Internet
Stream of host IPs hitting the router
Frequency Sketch / Heavy Hitter Sketch for this batch
Frequency Sketch / Heavy Hitter Sketch for this window
time 1
time 2
time 3
time 4
time 5
original DStream
window-based operation
windowed DStream
window at time 1
window at time 3
window at time 5
Frequency Sketch / Heavy Hitter Sketch global

# CMS in the wild




Stream of host IPs hitting the router
Frequency Sketch / Heavy Hitter Sketch for this batch
Frequency Sketch / Heavy Hitter Sketch for this window
Kafka
HDFS
Dashboard
time 1
time 2
time 3
time 4
time 5
original DStream
window-based operation
windowed DStream
window at time 1
window at time 3
window at time 5
Frequency Sketch / Heavy Hitter Sketch global

Spark Streaming

Lightbend

# Streaming CMS

```
// CMS parameters
val DELTA = 1E-3
val EPS = 0.01
val SEED = 1

// create CMS
val cmsMonoid = CMS.monoid[String](DELTA, EPS, SEED)
var globalCMS = cmsMonoid.zero

// Generate data stream
val hosts: DStream[String] = lines.flatMap(r ⇒
  LogParseUtil.parseHost(r.value).toOption)

// load data into CMS
val approxHosts: DStream[CMS[String]] = hosts.mapPartitions(ids ⇒ {
  val cms = CMS.monoid[String](DELTA, EPS, SEED)
  ids.map(cms.create)
}).reduce(_ ++ _)
```

# Streaming CMS

```
approxHosts.foreachRDD(rdd ⇒ {

  if (rdd.count() ≠ 0) {

    val cmsThisBatch: CMS[String] = rdd.first
    globalCMS ++= cmsThisBatch

    val f1ThisBatch = cmsThisBatch.f1
    val freqThisBatch = cmsThisBatch.frequency("world.std.com")

    val f1Overall = globalCMS.f1
    val freqOverall = globalCMS.frequency("world.std.com")

    // ..
  }
})
```

## Motivation of Streaming CMS

- Prepare the sketch online on streaming data
- Store it offline for future analytics
- It's a small structure - hence ideal for serialization & storage
- It's a commutative monoid and hence you can distribute many of them across multiple machines, do parallel computations and again aggregate the results

# Count Min Sketch - Applications

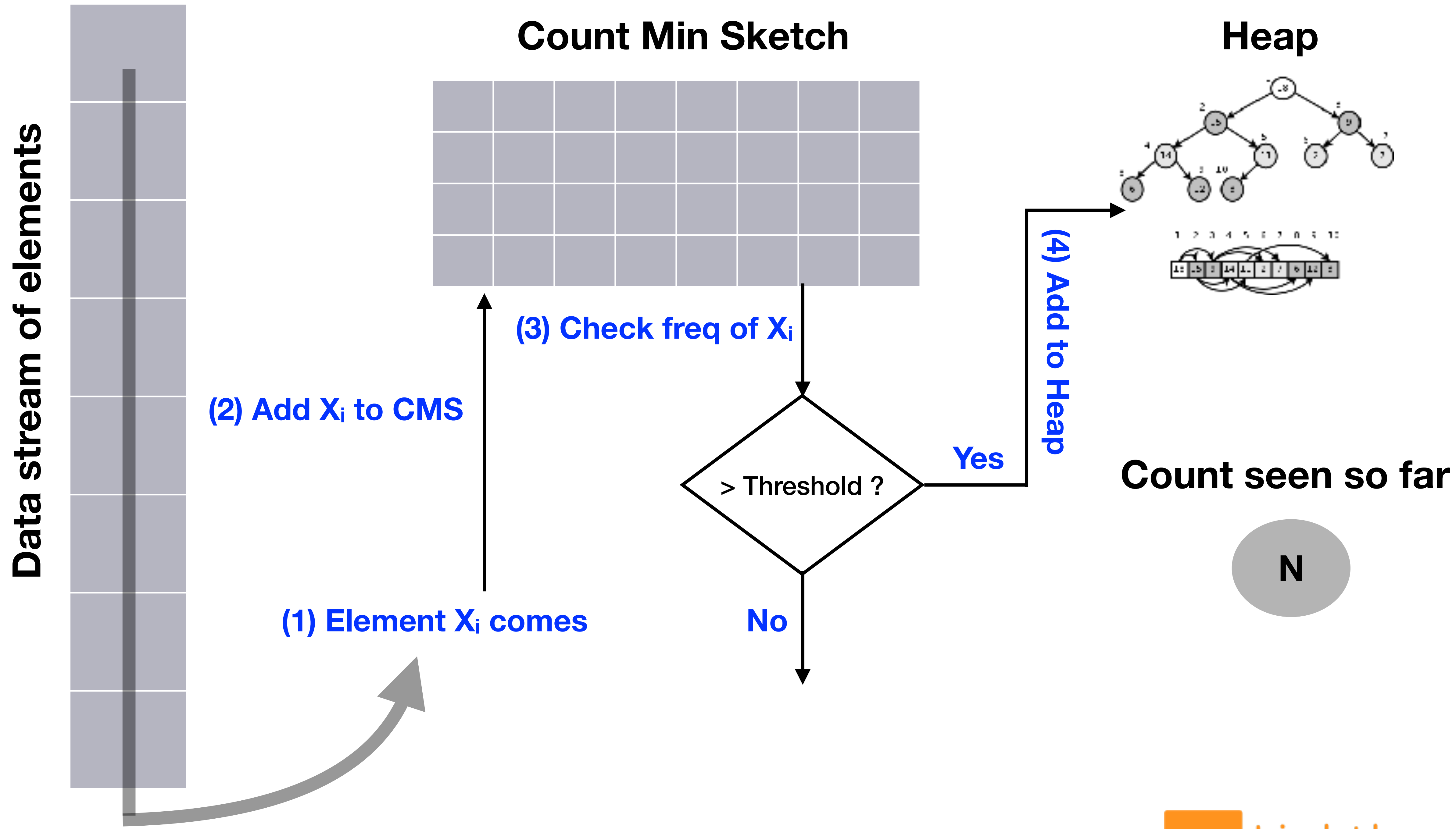
- AT&T has used it in network switches to perform network analyses on streaming network traffic with limited memory [1].
- Streaming log analysis
- Join size estimation for database query planners
- Heavy hitters -
  - Top-k active users on Twitter
  - Popular products - most viewed products page
  - Compute frequent search queries
  - Identify heavy TCP flow
  - Identify volatile stocks

[1] G. Cormode, T. Johnson, F. Korn, S. Muthukrishnan, O. Spatscheck, and D. Srivastava. Holistic UDAFs at streaming speeds. In Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data, pages 35–46, 2004.

# Heavy Hitters Problem

- Using a single pass over a data stream, find all elements with frequencies greater than k percent of the total number of elements seen so far.
  - unbounded data stream
  - will have to use sublinear space
- **Fact:** There is no deterministic algorithm that solves the Heavy Hitters problems in 1 pass while using sublinear space
- Hence $\epsilon - approximate\ Heavy\ Hitters\ Problem$

# Approximate Heavy Hitters using Count Min Sketch



Lightbend

# Streaming Approximate Heavy Hitters

```
// create heavy hitter CMS
val approxHH: DStream[TopCMS[String]] = hosts.mapPartitions(ids ⇒ {

  val cms = TopPctCMS.monoid[String](DELTA, EPS, SEED, 0.15)
  ids.map(cms.create(_))

}).reduce(_ ++ _)


// analyze in microbatch
approxHH.foreachRDD(rdd ⇒ {

  if (rdd.count() ≠ 0) {
    val hhThisBatch: TopCMS[String] = rdd.first
    hhThisBatch.heavyHitters.foreach(println)
  }

})
```

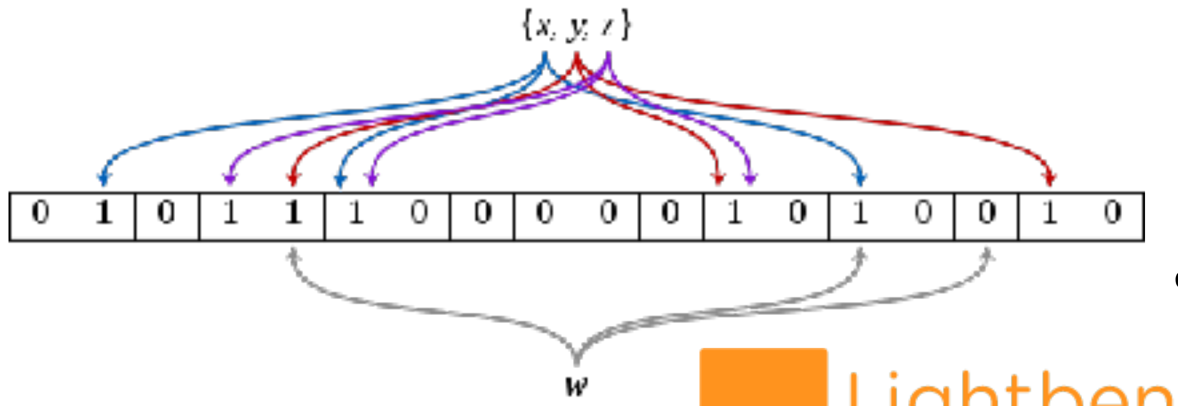
Lightbend

# Bloom Filter

- Another sketching data structure (based on hashing)
- Solves the same problem as Hash Map but with much less space
- Great tool to have if you want approximate membership query with sublinear storage
- Can give false positives

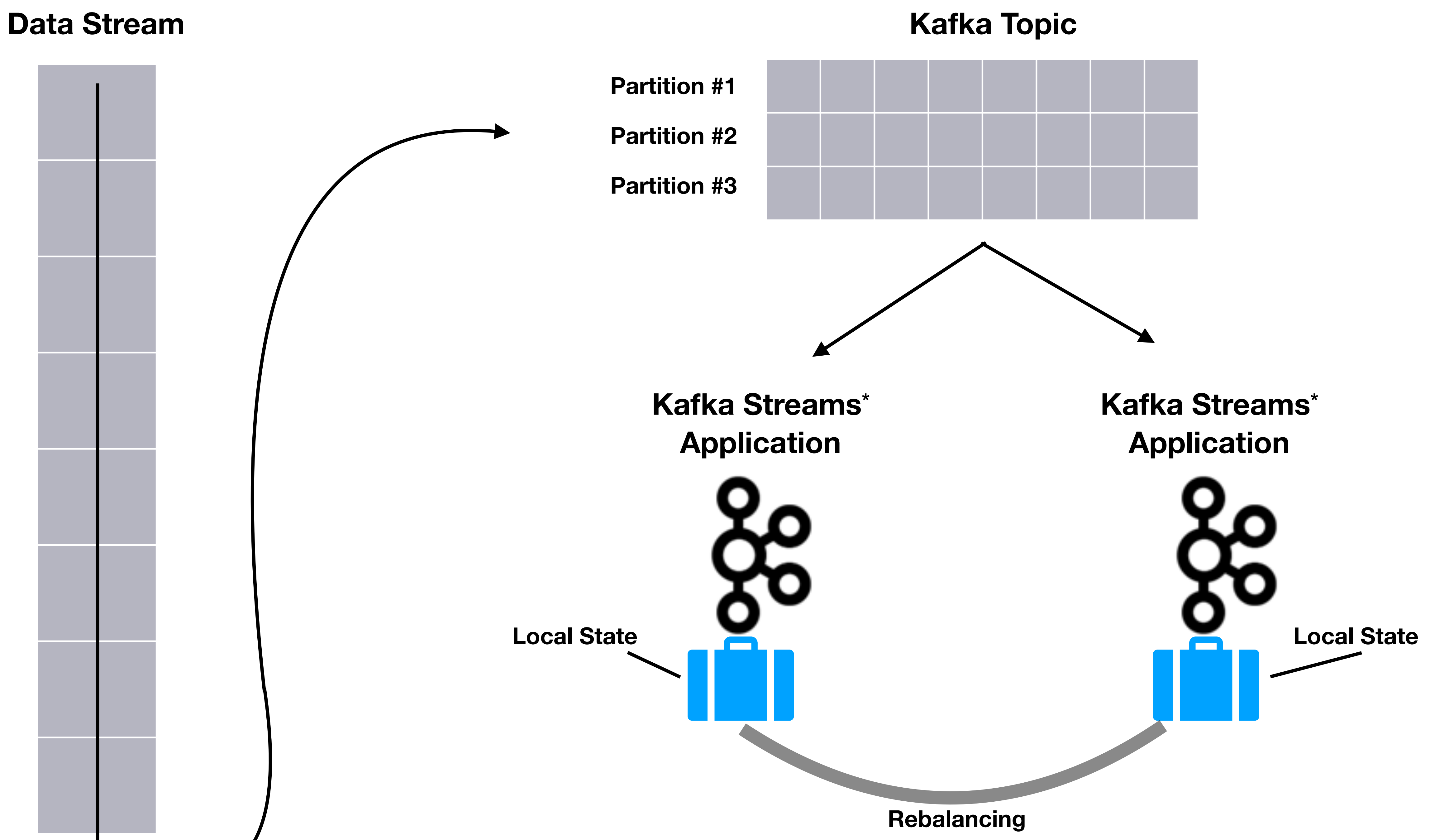
# Bloom Filter - Under the Hood

- **Ingredients**
  - Array `A` of `n` bits. If we store a dataset `S`, then number of bits used per object `= n/|S|`
  - `k` hash functions $(h_1, h_2, .., h_k)$ (usually `k` is small)
- **Insert(x)**
  - For $i=1,2,..,k$ set $A[h_i(x)]=1$ irrespective of what the previous values of those bits were
- **Query(x)**
  - if for every $i=1,2,..,k$ $A[h_i(x)]=1$ return `true`
  - No false negatives
  - Can have false positives



*Space/time trade-offs in hash coding with allowable errors* - B. H. Bloom. Communications of the ACM 13(7): 422-426. 1970.

# Bloom Filter as Application State



* 2 instances of the same application

# Bloom Filter State Store

```scala
// Bloom Filter as a StateStore. The only query it supports is membership.
class BFStore[T: Hash128](

  override val name: String,
  val loggingEnabled: Boolean = true,
  val numHashes: Int = 6,
  val width: Int = 32,
  val seed: Int = 1) extends WriteableBFStore[T] with StateStore {

  // monoid!
  private val bfMonoid =
    new BloomFilterMonoid[T](numHashes, width)

  // initialize
  private[processor] var bf: BF[T] = bfMonoid.zero

  //..
}
```

Lightbend

# Bloom Filter State Store

```scala
// Bloom Filter as a StateStore. The only query it supports is membership.
class BFStore[T: Hash128](

  override val name: String,
  val loggingEnabled: Boolean = true,
  val numHashes: Int = 6,
  val width: Int = 32,
  val seed: Int = 1) extends WriteableBFStore[T] with StateStore {

  // ..

  def +(item: T): Unit = bf = bf + item

  def contains(item: T): Boolean = {
    val v = bf.contains(item)
    v.isTrue && v.withProb > ACCEPTABLE_PROBABILITY
  }

  def maybeContains(item: T): Boolean = bf.maybeContains(item)

  def size: Approximate[Long] = bf.size
}
```

Lightbend

# BF Store with Kafka Streams Processor

```scala
// the Kafka Streams processor that will be part of the topology
class WeblogProcessor extends AbstractProcessor[String, String]

  // the store instance
  private var bfStore: BFStore[String] = _

  override def init(context: ProcessorContext): Unit = {
    super.init(context)

    // ..

    bfStore = this.context.getStateStore(
      WeblogDriver.LOG_COUNT_STATE_STORE).asInstanceOf[BFStore[String]]
  }
  override def process(dummy: String, record: String): Unit =
    LogParseUtil.parseLine(record) match {
      case Success(r) ⇒ {
        bfStore + r.host
        bfStore.changeLogger.logChange(bfStore.changelogKey, bfStore.bf)
      }
      case Failure(ex) ⇒ // ..
    }
  // ..
}
```

Lightbend

## Lightbend Fast Data Platform

The easy on-ramp to successful streaming Fast Data applications.

https://www.lightbend.com/products/fast-data-platform

Lightbend

# Questions?

Lightbend